Technical Report

UCAM-CL-TR-584
ISSN 1476-2986

Number 584

UNIVERSITY OF
CAMBRIDGE

Computer Laboratory

Concise texture editing

Stephen Brooks

March 2004

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted October 2003 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Jesus College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

ABSTRACT

Many computer graphics applications remain in the domain of the specialist. They are typically characterized by complex user-directed tasks, 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 or object manipulation commands embedded in a variety of interface components. The complexity of these graphic editing tools often requires that the user possess a correspondingly high level of expertise.

Concise Texture Editing is aimed at addressing the over-complexity of modern graphics tools and is based on the intuitive notion that the human user is skilled at high level decision making while the computer is proficient at rapid computation. This thesis has focused on the development of interactive editing tools for 2D texture images and has led to the development of a novel texture manipulation system that allows:

- the concise painting of a texture;
- the concise cloning of textures;
- the concise alteration of texture element size.

The system allows complex operations to be performed on images with minimal user interaction. When applied to the domain of image editing, this implies that the user can instruct the system to perform complex changes to digital images without having to specify copious amounts of detail. In order to reduce the user's workload, the inherent self-similarity of textures is exploited to interactively replicate editing operations globally over an image. This unique image system thereby reduces the user's workload through semi-automation, resulting in an acutely concise user interface.

Table of Contents

CHAPTER 1 INTRODUCTION

CHAPTER 2 RELATED RESEARCH

CHAPTER 3 SELF-SIMILARITY BASED TEXTURE PAINTING

CHAPTER 4 SELF-SIMILARITY BASED TEXTURE CLONING

CHAPTER 5 SELF-SIMILARITY BASED TEXTURE WARPING

CHAPTER 6 CONCLUSIONS & FUTURE WORK

APPENDICES

LIST OF FIGURES

Caption **Page**

LIST OF TABLES

Caption **Page**

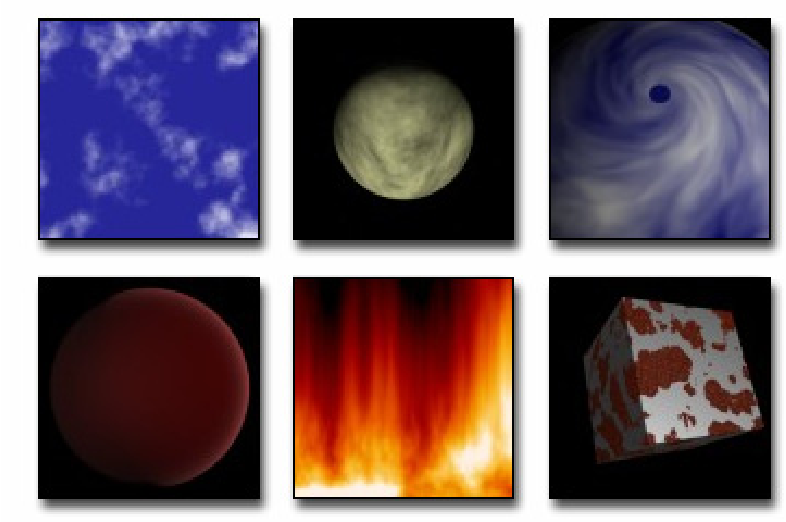
Acknowledgments

I would like to thank Neil Dodgson, my supervisor for help, advice and support throughout my PhD. I would also like to thank Marc Cardle, Tony Polichroniadis and all those in our lab who provided valuable feedback. This work was supported by the Cambridge Commonwealth Trust and an Overseas Research Scholarship. Additional conference travel support was provided by Jesus College, Data Connection, the Royal Academy of Engineering and the Cambridge Philosophical Society.

Finally I would like to thank Su, family, friends, felines and canines for all the support and encouragement they have given me over the last three years.

Chapter 1

INTRODUCTION



Chapter Structure

This introductory chapter surveys the industrial use of surface texturing, defines background concepts relevant to the problems that will be addressed in later chapters and discusses the current research aims.

We will begin in Section 1 with an examination of the role of textures in realistic image synthesis. In Section 2 we map out an overview of the extent to which textures are used in industry. We provide a synopsis of the important application areas of texturing such as product design, architecture and computer games. While examining industry usage, the motivation for the present work will emerge. Section 3 considers various methods of texture creation and editing that have been developed to address this industry need, including procedural synthesis and image editing suites. Section 4 then presents the current research aims and provides an outline of the overall dissertation structure.

1. Modelling the Complexity of the Real World

The natural world has an arbitrarily high level of visual complexity, dependent only on the level at which the observer chooses to examine it. This complexity can be found in terms of both geometry and texture (see Figure 1). The simulation of realism in computer imagery has been one of the main, if not the primary, research direction in computer graphics since the field originated, producing many significant advances in areas such as modelling and rendering. However, it is clear that a computer, being a finite machine, will only be able to at best approximate this level of complexity due to limited resources. From this emerges the interesting problem of maximizing the utility of the available resources for visual accuracy. But, it is not simply the case that all computer graphics issues can be addressed by increasing processor speeds and memory bandwidth. Without accurate data that adheres to the complex visuals of the natural world while at the same time meeting the demands of the scene designer, the additional power would merely be capable of rendering inaccuracies at higher resolutions. At present, the convincing production of realistic imagery is still as much an art as it is a science.

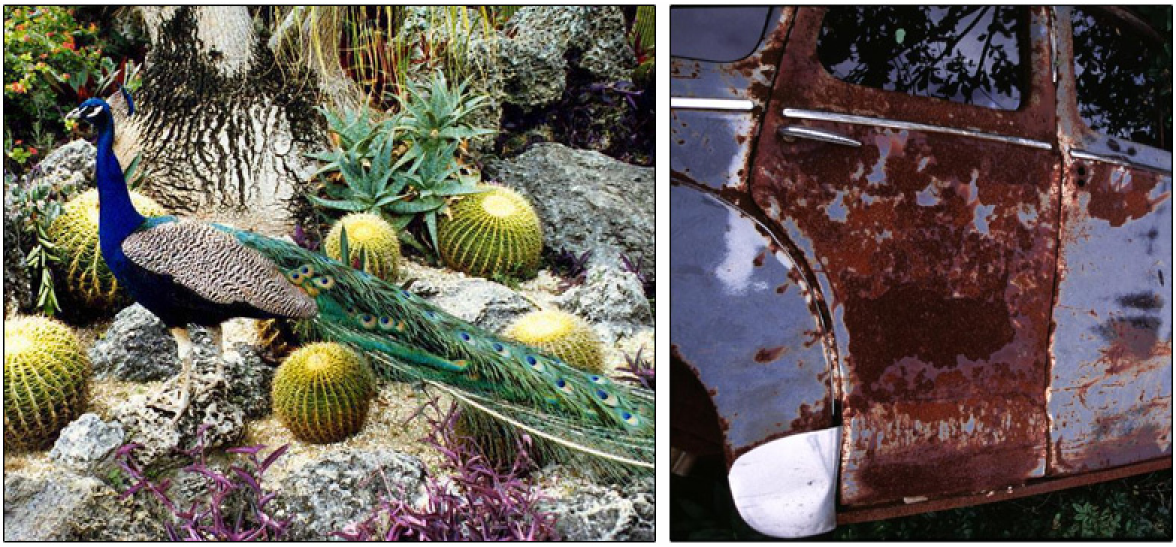


Figure 1: Scene complexity: Naturally occurring textures (left), and a natural texture that has asserted itself over a man-made object (right).

At this point, it is useful to define what is meant by realism and visual accuracy. For still images, the goal is to accurately reconstruct a scene at a particular point in time. In this case, realism refers to the accurate modelling of geometry and light reflection properties of surfaces: we wish to create an image that is indistinguishable from a photograph of a scene. However, we must also keep in mind that different circumstances require different levels of approximation to "realism". When the problem is extended in time one must also consider the quality of movement for character animation, cloth, fur, smoke, water and wind to name but a few examples. When the problem is subjected to the further demand of interactivity, such as in Virtual Reality (VR) environments, then real-time response to user direction also becomes critical. In computer graphics, the traditional conflict is, of course, cost versus accuracy.

The degree of realism sought can also depend on the particular application. For high-budget films, no expense is spared in the quest for realism. Conversely, CAD model renderings typically forgo detail for clarity, as excessive detail can interfere with the effective display of information (see Figure 2). Moreover, the target audience can be important as well. For example, cartoons for children do not attempt to model the natural world even to the extent that is now possible with current off-the-shelf graphics software.

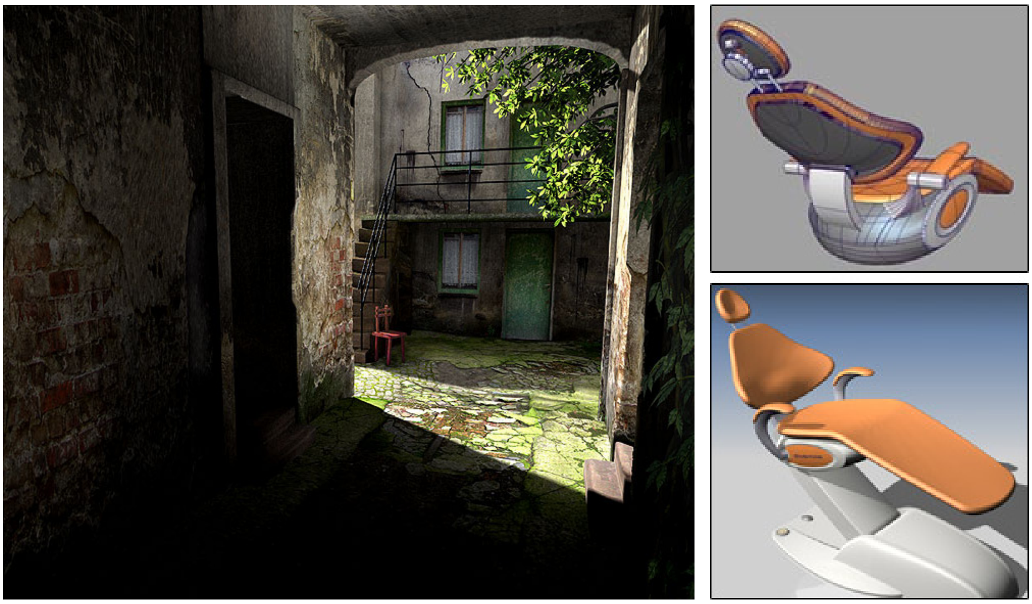


Figure 2: Rendering complexity: A photorealistic rendering [Benedet03] generated by an animation package (left) and low-detail CAD renderings [frogdesign03] of a chair design (right).

There are many vehicles for approximating the complexity of a rendered scene. For example, Level-Of-Detail (LOD) geometry can be employed so that as the object gets farther away from the viewer, the high resolution geometry is replaced with decimated versions of itself, or even with image billboards. With regard to motion, a further example is the pre-computation of special effects such as fire, in order to avoid the expense of calculating the trajectory of particles on demand. In this thesis we are concerned with that aspect of realism, surface reflectance properties, which can be well approximated with the application of surface texture maps.

In the left image of Figure 1 we have seen a collection of highly complex objects each with its own equally complex set of surface attributes. Generally speaking, these high frequency surface features could be sufficiently approximated using a great many colour polygons, but this would quickly overwhelm a graphics system both in terms of the memory and the processing requirements. In addition, there is also the problem of modelling or acquiring such geometric detail. Recognizing these limitations to the attainment of realistic simulation, the technique of texture mapping was introduced by [Catmull74] to provide the illusion of complexity at a reasonable cost. This simple idea of tiling a 2D image across a 3D surface spawned much research on how these textures are specified, edited, generated, mapped, and correctly rendered.

2. The Role of Textures in Industry

To someone unfamiliar with the history of Computer Generated Imagery (CGI), it may incorrectly appear that this form of visual representation had suddenly been invented within the last decade. In reality, the commercial computer graphics industry appeared in a nascent form in 1969 with the incorporation of the first commercial graphics company Evans & Sutherland which focused its efforts on simulation systems, particularly flight simulators. However, this misapprehension is understandable as it is only in the last fifteen years that computer graphics has become pervasive in the entertainment industry and in home computer systems.

Although there were a few earlier attempts at the integration of CGI in movies, such as *TRON* and the genesis effect in *Star Trek*: *The Wrath of Kahn*, it was not until the formation of the computer graphics company, Pixar, and the development of the Pixar's Renderman® standard that CGI entered into mainstream consciousness. The development of Pixar's Renderman® has had a profound impact on the use of texturing in CGI applications. This is partly due to the influence of the individuals involved but it is also a result of the flexibility and quality of its procedural Shader language [Pixar89] of which we will see more in Section 3.1. The use of CGI in Hollywood films has now reached such a fine art that it is common for the audience to be unaware of the subtle image processing and editing techniques used to create effects that are infeasible by more traditional methods.

But there are other, less conspicuous, yet equally important application areas of surface texturing that have developed in parallel with advances in CGI:

Product Design and Architecture: The computerization of design [Mitchell95] and architecture [Greenberg91] has freed humans from many of the tedious aspects of the creative process. The use of texturing has permitted the rapid generation of high quality perspective renderings of these designs (see Figure 3).

Image Analysis: Image analysis of textures has become a mainstay of computer vision research, even reaching into areas as unlikely as the food industry [Davies00] and ceramic tile production [Dodgson95]. Other applications are wide ranging, and include 3D object recognition, automated manufacturing and quality control.

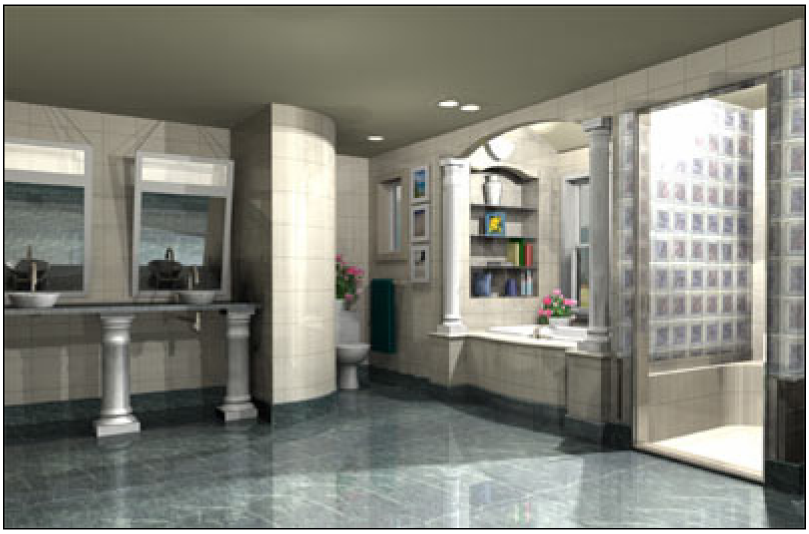


Figure 3: Photorealistic rendering of an architectural design using texture mapping [ART03].

Electronic Publishing: Textures are often used to enhance the visual impact of documents and websites. For example, a common practice is to use tilable texture backgrounds which simulate the visual appearance of a variety of paper grains. Other, more overt uses include the layering of textures to add visual complexity to the overall page design.

Digital Art: There are increasing numbers of artists, many with backgrounds in the more traditional mediums, who are turning to the computer as a creative tool [nydigitalsalon03]. Textures have played an important role in the work of many digital artists.

Visualization: In commercial as well as scientific research, vector fields are frequently needed to describe information. Textures are one method that allows the system to give an impression of the nature and properties of vector field data [vanWijk02]. These directional textures provide an intuitive way for representing vector fields at a high spatial resolution (see Figure 4). Another visualization application, medical imaging systems, is one of the fastest-growing areas of graphics technology. Here, colour and texture information are used to add valuable information to medical representations [Knapp97].

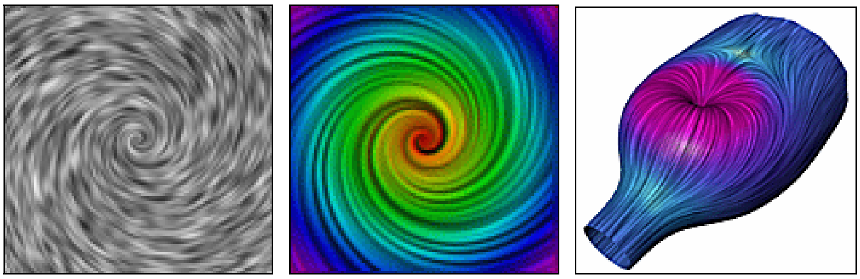


Figure 4: Using texture to convey vector flow [Battke97].

Computer Games: It is tempting to disregard the importance of the computer games industry as frivolous and so treat it as a secondary applications area. However, it is widely known that it is the video game industry which is the main driver of the on-going development of computer graphics hardware. Without computer games we would not have, so quickly, reached the point where we can use this advanced hardware for more serious applications.

The widespread use of computer games has produced some astonishing statistics in recent years [IDSA03]:

- 50% of all Americans over the age of 6 play computer games.
- The average age of computer games players is 29.
- 43% of game players are women.
- Sales of computer games reached $6.9 billion (USD) in 2002.
- The vast majority of people who play games do so with friends and family.

Computer games have thus moved from being the preserve of the teenage boy to being ubiquitous; some predict they will become as widespread as television is today. Moreover, it has also been argued that computer games include mentally stimulating features that can be used as a model of learning [Gee03]. The role of textures in this expansive industry has been fundamental due to the sharp constraints that interactivity places on levels of geometric detail (see Figure 5).

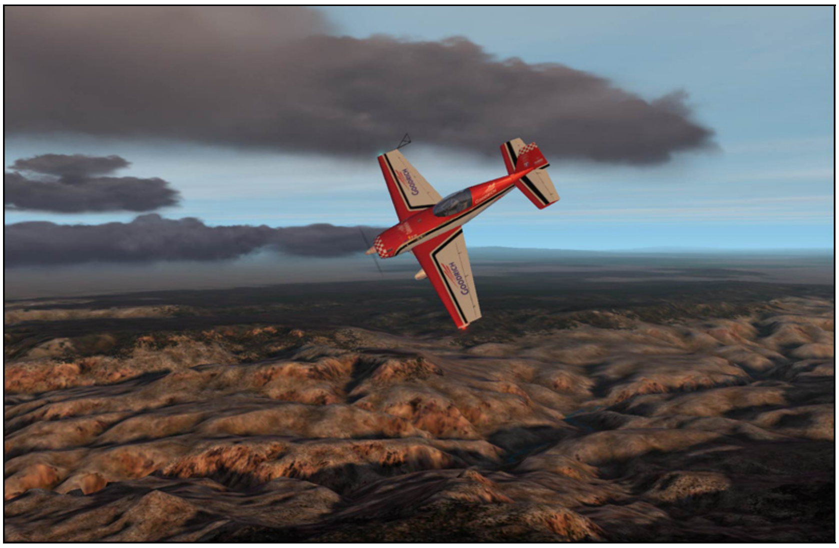


Figure 5: Extensive use of texturing in Microsoft® Flight Simulator 2002 [MICROSOFT02].

Clearly, the applicability of texturing methods is extensive and as we will see when discussing related research in Chapter 2, this utility continues to expand. In the next section we briefly consider the product of past research which has found its way into software packages and design methods now used in industry.

3. Standard Texturing Methods and Software

Texture mapping is one of the most effective techniques used in image synthesis, greatly enhancing the visual detail of computer generated images while demanding only a relatively small increase in computation. Texturing has moved from the obscurity and expense of the university/industrial lab to the ubiquity of the desktop and now, even the mobile phone. The early research on texturing has been adopted to such an extent that it is now more reasonable to consider these early approaches as being industry standards.

We should begin our discussion by defining a simple shading model for rendering a triangle face in a scene so that we can then enumerate the various types of map that are used to alter this equation in some fashion. A shading model takes as input the available information about sources of light in the environment, the position and orientation of the triangle and data about the roughness and colour of the face to approximate how light reflects from the surface to the virtual camera [Hill90].

An approximated shading model will typically use two types of light source (point lights and ambient) along with two types of reflection of that light (diffuse scattering and specular reflection). Diffuse light is that portion of incident light that slightly penetrates the surface and is subsequently reemitted uniformly in all directions. Specular reflection does not penetrate the surface but is instead directly reflected, producing sharp highlights of that same colour as the incident light. In the reduced case of achromatic lighting, the diffuse contribution to the amount of light reflected from a point on the surface *p* is calculated as:

$$I_d = I_s r_d (u_s \cdot u_n)$$

where:

n is the normal vector to the surface at *p*,

s is the vector from *p* to the point light source,

u_n is the normalized version of vector *n*,

u_s is the normalized version of vector *s*,

I_s is intensity of the point light source,

r_d is the diffuse reflection coefficient of the surface.

This is a physically correct representation of perfect Lambertian diffuse reflection.

The specular contribution to the amount of light reflected is:

$$I_{sp} = I_s r_s (u_r \cdot u_v)^f$$

where:

v is the vector from *p* to the viewer's eye,

u_v is the normalized version of vector *v*,

u_r is the normalized reflectance direction,

I_s is intensity of the point light source,

r_s is the specular reflection coefficient of the surface,

f is a reflection factor that concentrates the specular reflected light.

This model of specular reflection is due to Phong [Phong75]. It is only an approximation to true specular reflection, which is a far more complex physical phenomenon. Phong's approximation, nevertheless, is the standard model used in computer graphics.

Finally, the ambient contribution to the amount of light reflected is simply:

$$I_{am} = I_a r_a$$

where:

I_a is intensity of the ambient light,

r_a is the ambient reflection coefficient of the surface.

Ambient illumination is, again, an approximation. It represents the inter-surface inter-reflections in the scene by a single global variable, I_a. This is a gross approximation but is computationally much simpler than any other method which explicitly calculates these inter-reflections.

The summation of the diffuse, specular and ambient contributions to the amount of light reflected from a point on the surface *p* is therefore calculated as:

$$I = I_a r_a + I_s \left[r_d (u_s \cdot u_n) + r_s (u_r \cdot u_v)^f \right]$$

This function generalizes easily to a colour reflectance model with similar expressions for each of the red green and blue colour channels.

With our surface reflectance model in hand we can now consider the ways in which texture mapping can alter the calculation of this equation on a point-by-point basis [Haeberli93]. Here we enumerate the variations of the texture mapping theme beginning with the most common:

> *Texture mapping*: This method allows the point-wise control of the three colour components of the diffuse reflection coefficient r_d in order to enhance the surface definition [Blinn76]. This uses either full colour images or *RGB* values from a computed function which are applied to the model's surface to create the appearance of fine colour detail. An object's surface is typically parameterized with (*s*, *t*) coordinates that map each surface point, *p*, to a pixel in the texture image file that is then used for the diffuse colour of that point.
>
> *Diffusion mapping*: In practice, a high level of realism often requires many layers of textures. Diffusion mapping is a term used in industry for using a secondary texture map layer specifically to darken or lighten areas of the surface. Typical uses are to make certain portions of object appear wet or dirty, such as water on a sidewalk. The calculations involved are the same as for texture mapping and both layers are combined by multiplying the two together yielding a final pixel colour product at each point.
>
> *Specularity mapping*: This type of mapping is essentially the same as diffuse texture mapping, except that it is applied to the specular reflection coefficient r_s. Specularity mapping is often used for such special effects as soap bubbles, mother-of-pearl, oil slicks and the surface of CD-ROMs, as this type of surface exhibits coloured patterns in its specular reflections.
>
> *Bump mapping*: The concept of mapping an image over a geometric surface has been extended to permit the point-wise control of the surface normal, *n*. Bump mapping adds per-pixel surface relief shading, increasing the apparent complexity of the surface. Since the light scattered from a diffuse surface depends only on the dot product of surface normal, *n,* and the light source direction, *s,* by altering the normal in this way the surface is shaded as if it were a varying shape [Blinn78]. In this case,

the texture map is treated as a single-valued function which represents height. The gradient across the height field is calculated and this is used to perturb the surface normal.

Displacement mapping: This can be seen as a more accurate, yet more costly, version of bump mapping. But in this case, the texture map is used to actually move the surface point, *p*, with each pixel holding a displacement value. By deforming the actual geometry it avoids the visual anomalies that can form when using bump mapping. For example, a bump mapped surface will not cast shadows correctly, because the surface is actually flat.

Transparency mapping: This form of mapping is used to indicate transparent or semi-transparent areas on the surface [Gardner85]. As with bump mapping, the texture is treated as a single-valued function with the transparency of the object affected by the intensity of the pixel values. The surface is transparent when the texture image is white and opaque when it is black. The inclusion of non-refractive transparency requires us to augment our standard illumination model by interpolating the individual rendered pixels values from the transparent surface, S_1, and the surface that is directly behind it, S_2. This can be viewed as a form of compositing as we interpolate the light, I_1, reflected from a point p_1 on surface S_1 with the light, I_2, reflected from point p_2 on surface S_2. The final reflected light value is computed as:

$$I = (1 - T(u,v)) I_1 + T(u,v)\, I_2$$

where $T(u,v)$ is the transparency map value ranging from 0 to 1 at point p_1 on the surface S_1.

Light mapping: Light maps store pre-computed lighting information as texture maps on surfaces. This technique has been used extensively in computer games to reduce computational expense under static lighting conditions. Both texture maps and light maps are multiplied together at run-time and cached for efficiency.

Environment mapping: Environment mapping simulates an object reflecting its surroundings and in its simplest form it gives rendered objects a chrome-like appearance. Environment mapping assumes that an object's environment is infinitely distant from that object and, therefore, can be encoded in an omni-directional image. These image maps are a record of the global reflection and lighting on an object and are re-sampled during rendering to extract view specific information which is then applied as a texture [Greene86].

Mipmapping: Mipmapping is the process in which a set of filtered texture maps of decreasing resolution are generated from a single high resolution texture and used to improve efficiency and accuracy during texture mapping [Williams83]. It is a method of complexity reduction which pre-computes multiple versions of the texture image, each with decreasing levels of detail. Depending on the area that a textured triangle occupies on the screen, the graphics system chooses the appropriate level of the mipmap. In this way, objects which are far away can be textured with lower resolution images. This alleviates the flickering problem that can arise from excessively down-sampling a texture during rendering. Mipmapping can be used with any type of texture mapping.

Although these techniques are only approximations to the complexity of the real world, when combined these mapping methods can produce spectacular results as we have already seen in Figures 2, 3, and 4. Moreover, the recent advent of *pixel shaders* has magnified the power of these maps [nVidia02]. Pixel shaders are a facility that has only become available on the most recent graphics chips. The shaders are compiled programs which execute directly on the Graphics Processing Unit (GPU). Embedded into the texturing pipeline, these pixel shaders vastly extend the flexibility of texture mapping by performing complicated shading operations on each individually rendered pixel.

Now that we have seen how texture maps are used, we will next look at some of the most popular methods of creating them. These methods include procedural synthesis and the use of 2D and 3D painting packages for hand painting textures. We first look at the procedural synthesis of surface textures.

3.1 Procedural Synthesis

Procedural synthesis has enjoyed extensive use as a means of automatically generating 2D and 3D textures [Worley96, Wyvill87, Ebert94]. Indeed, the procedural shading language of Pixar's Renderman interface which is centred on the procedural model is often cited as its main strength. Procedural synthesis is computed from some function on a pixel by pixel basis. An advantage of this approach is that the function's domain can be either in R^2 or R^3, meaning that an object can be cut in half or 'carved into' and its surface appearance maintains an appearance of object consistency throughout [Peachey85].

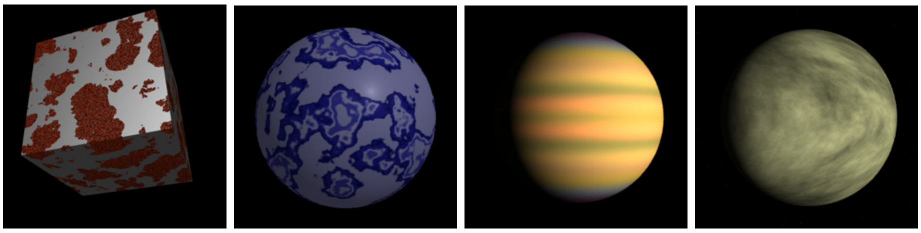


Figure 6: Examples of procedural textures.

In this model the user does have some control over synthesis parameters. But, these are indirect controls in the sense that they alter broad characteristics of textures constructed from interpolated arrays of pseudo-noise [Lewis89]. In practice this makes it almost impossible to allow the user to specify that a certain vertex's colour values should be (*r*, *g*, *b*). Moreover, not all types of texture can be synthesized with this method. Stochastic textures such as those shown in Figure 6 can be synthesized as can other types of vortices, clouds and even woods. However, highly structured textures can present difficulties. And even for those textures that are well suited to the method, the process of coding procedural textures is considered something of a black art. Another drawback is that these methods typically cannot operate on existing textures.

3.2 Image Editing Packages

Another common way to produce texture maps is to hand paint them directly on 3D surfaces [Williams90] or edit photographs in an image editing suite. There are a number of image editing packages available ranging from the highly capable Adobe™ Photoshop system to very limited editors such as Microsoft® Paint! which is bundled with the operating system. More advanced editors provide facilities such as natural painting and drawing, colour correction, photo enhancement, special effects, shadowing, colour space conversion, and adding text to graphics.

Of particular relevance to the present work are the patch tool, the cloning tool ("rubberstamp") and the healing brush of Adobe™ Photoshop 7.0 [ADOBE02]. These tools are used to manually copy details, repair damaged photos, remove blemishes, and correct other flaws. The function of the cloning tool is to copy the content from one part of an image to another. The user first defines the position of the clone source within the image and the details from the source position are then copied to the area being drawn over with the mouse. The healing brush works in a similar fashion to the cloning tool. The difference lies in the fact that the healing brush analyzes the target area underneath the mouse during cloning and preserves its original tonality and shading. The patch tool performs a similar operation, but on pre-selected areas. Taken together, these cloning tools are quite powerful in the hands of an expert user.

4. The Current Research Aims

The focus of the present work is to make concrete the idea that the user suggests and the software articulates. The domain we are concerned with is the alteration of texture images for computer graphics applications. Therefore, the following ideas primarily deal with the cooperative and iterative interaction between the user's guidance and the software's manipulation of texture images. As we see in Chapter 2, there have been attempts towards determining what the optimal division of labour should be between the user and the software in order to automate the process of constructing graphical objects of sufficient realism. However, the integration of automation with user preference remains an open problem in the context of texture editing.

4.1 Motivation

Image editing software is often characterized by a seemingly endless array of toolbars, filters, transformations and layers. Many of the software applications listed in this chapter require the skills of an artist as well as the ability to navigate complicated software packages. The time spent learning graphics software and the rarity of people capable in both skill sets has placed a greater emphasis on the cost of the designer's time which is both labour intensive and expensive. This becomes ever more significant relative to the progression of the low-cost trend in graphics hardware which has experienced dramatic reductions in the price of high quality graphics chips and memory. For example, one can now purchase a state-of-the-art consumer graphics board, such as nVidia's Geforce4™ or ATI's Radeon™ 9000, for a fraction of the cost of a dedicated graphics workstation. We might then speculate that the development of graphics software which further reduces the user's workload should be considered a critical aim in light of where the true cost of graphics applications now resides.

Software used for the design of graphical elements can provide three types of benefits to the user. Well-implemented software can assist in the instantiation of a previously established design, increase the efficiency of the design process and support the exploration design alternatives. While instantiating a design, the user already has a clearly defined outcome that must be produced and it is the role of the software to help crystallize the user's intentions. While exploring design alternatives, the user is as yet uncertain as to the final output and uses the software as a tool for rapid experimentation. In this exploratory case, a novice or even an expert user may simply want the software to "do what I intend" without demanding a full specification of all graphical aspects down to the pixel level. But, whether the task is instantiation or exploration, efficiency and flexibility is always a concern. Design software which is slow or difficult to use increases the viscosity of the interface, inhibiting the progress of the user through editing alternatives.

The present research seeks to streamline the process of texture editing while giving the computer graphics artist a high degree of control over the final appearance. Self-similarity based editing assists the user in performing complex editing tasks while instantiating an intended design and can also aid the user by providing facilities for rapid experimentation.

The principal contributions of this dissertation are novel texture editing techniques which allow the user to perform replicated texture editing operations with minimal input. The self-similarity based texture editing system can be seen as a new instance of the Computer-Human Collaborative style of interaction [Ryall97]. The user is able to minimally specify alterations to a digital image, whilst relying on the system to perform repetitive, time-consuming tasks. The combination of domain knowledge and user direction will together provide us with sufficient information to alter a given texture. Consequently, the style of interaction lies between automation and complete user manipulation.

4.2 The Texture Editing System

This thesis presents a method of interactive texture editing that utilizes self-similarity to replicate intended operations globally over an image. Inspired by the recent successes of hierarchical approaches to texture synthesis, self-similarity based texture editing also uses multi-scale neighbourhoods to assess the similarity of pixels within a texture. However, neighbourhood matching is not employed to generate new instances of a texture. We instead locate similar neighbourhoods for the purpose of replicating editing operations on the original texture itself. The result is not a new instance of a texture but is a fundamentally new texture.

To make an alteration to the texture, the user clicks on a pixel within the texture image. The local neighbourhood of the chosen pixel is then compared against that of every other pixel's neighbourhood in the image. Changes then affect not just the pixel selected but also a subset of all pixels in the image: those that have local neighbourhoods whose difference to the selected pixel are within a threshold. This allows the following concise texture editing operations which have been implemented:

1. Replicated Painting
2. Replicated Cloning
3. Replicated Warping

Replicated painting alters the colour or brightness of all similar pixels to the user selected pixel. Replicated cloning is a similar operation in that it also overlays colours onto the image being edited. The distinction lies in the fact that cloning does not paint with a solid (*r*, *g*, *b*) colour, it instead paints with colour values derived from a second image or texture. Replicated warping is distinct from the other tools in that it does not affect pixel colour; rather it modifies the shape of texture elements under the user's guidance. Those pixels whose local neighbourhoods are sufficiently similar to the user selected point are expanded locally. Since the overall area remains the same, some regions are compressed while others are expanded. These global operations are performed interactively, most often directed with just a single mouse movement.

4.3 Dissertation Structure

The remainder of this dissertation provides an account of related research and how the current work progresses the state-of-the-art in image editing. The problems being addressed are made explicit and the importance and utility of the solutions are argued. The work is presented in one background research chapter and three work chapters which provide a detailed account of the texture editing techniques. Each of the work chapters represents one of the applications of the self-similarity based editing concept. Chapter 3 introduces the first of the self-similarity based texture editing tools: replicated painting. Chapter 4 then broadens the concept of replicated editing to the controlled cloning of one image over another. In Chapter 5, self-similarity editing is then extended to the case of replicated texture warping. Finally, we conclude with Chapter 6 which discusses the limitations of the work and presents possible directions for future research.

Chapter Summary

In this chapter, we considered how textures can be used to visually approximate the geometric complexity of the real world. We then examined the widespread use of textures in industries ranging from product design to medical diagnosis and examined the various texturing methods that have enjoyed common usage in these application areas. This included a detailing of the various forms of 'mapping' along with the tools that have been developed and deployed to assist users in the creation and manipulation of these 'mappings'. We then offered the motivational argument for the current work as well as an overview of the texture editing system.

GLOSSARY

Bump Mapping – A shading technique using greyscale textures to simulate wrinkled or bumped surfaces. Bump mapping provides a surface with the appearance of surface detail, such as dimples on a orange, without a corresponding increase in the geometric complexity.

Mipmapping – A method of complexity reduction which pre-computes multiple versions of the texture image, each with decreasing levels of detail. Depending on the area that the texture object occupies on the screen, the graphics system chooses the appropriate level of the mipmap. In this way, objects which are far away are can be textured with lower resolution images.

Pixel Shaders – Programmable hardware functions that calculate surface effects on a per-pixel basis. This allows the scene designer to create effects beyond the triangle level by determining the lighting, shading, and colour of each individually rendered pixel.

Texture – Many objects in the real world exhibit changes in colour across their surface; this is commonly known as the object's texture. An example of a texture is the colour changes of bark over a tree's surface. Similarly, in computer graphics applications, 3D objects can be assigned a changing pattern of colour to give the appearance of surface complexity.

Vertex Shaders – Programmable hardware functions that calculate surface and geometric effects on a per-vertex basis. This allows the scene designer to perform complex transformations on an object's geometry such as real-time displacement mapping or triangle mesh morphing.

Chapter 2

RELATED RESEARCH

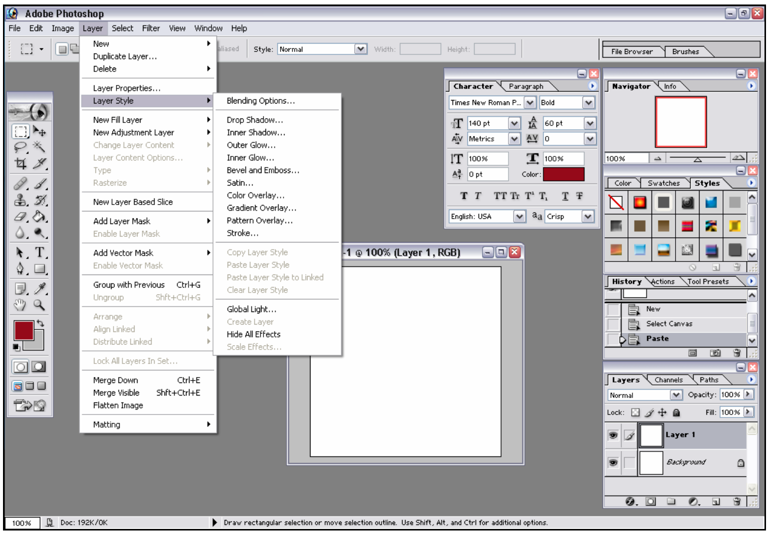
Chapter Structure

Graphics research inevitably moves beyond what is available as commercial product. Recent results relating to texture synthesis and semi-automatic methods of computer graphics are detailed in this chapter. It includes a discussion of the strengths and deficiencies of prior work, noting those areas which are open to improvement. By examining prior research, we will delineate the boundary between the past and present work.

Section 1 discusses how recent prototype systems have been developed which attempt to move away from the over complexity of modern image editing tools. Section 2 then goes on to describe in more detail a number of such systems including design galleries, constraint-based graphics and evolutionary art. This is followed in Section 3 by methods of texture synthesis, a subset of which have algorithmic similarities with the present work. The chapter concludes in Section 4 with an examination of previous texture and image editing systems.

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.



Draw rectangular selection or move selection outline. Use Shift, Alt, and Ctrl for additional options.

Figure 7: Adobe™ Photoshop 7.0 Interface [ADOBE02].

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 (see Figure 7), these semi-automated prototype systems offer interaction at a higher semantic level, consequently minimizing the amount of user interaction and the number of controllable variables.

In the following sections, an overview of computer graphics techniques which bear a direct relation to this thesis will be presented. This serves the dual purpose of presenting the reader with requisite concepts as well as delimiting the boundary between background work and the research of the present thesis. The amount of related published material is quite large, as the length of the reference section will attest. And so, it has been necessary to limit the discussion to only those papers that are fundamentally related work to the present work.

2. Computer-Human Collaboration in Graphics

The central thesis of Ryall [Ryall97] is an attempt to reformulate the human computer interface problem by delineating the roles of the user and the computer more equitably. A collaborative framework is established which seeks to strike a balance between the responsibilities of the human and the computer. In more traditional interface approaches, the interface is seen as an input language for the user, an output language for the machine, and a protocol for interaction. In this master-slave framework the two forms of interaction are manual (tool-based packages) and fully-automatic (oracle-based software). The focus of the computer-human collaboration framework and also of the present thesis is on the range of semi-automatic control that lies in between.

The strengths of the user lie in a broad range of experience and often intuition is the source of many problem solving skills. On the other hand, a computer is proficient at fast computation. The computer-human collaboration framework tries to establish a division of labour that optimally draws on the strength of both interacting agents. It does so by characterizing the interaction as an optimization problem wherein the user is responsible for the global aspects of the search and the computer is responsible for finding the local minima. It is also argued that the design process is inherently dynamic and based on an interactive refinement which can be decomposed into the two phases of conceptualization and articulation. In the conceptualization phase objects are selected and organized. In the articulation phase the precise location of objects are determined. It is the user who places the computer into alternate areas of the search space, and determines when an acceptable solution has been reached; the computer finds the local minimum, displays the results to the user and provides simple mechanisms to facilitate the interaction.

2.1 Design Galleries

An application of the computer-human framework is that of parameter specification for computer graphics algorithms. In this paradigm, the user makes aesthetic judgments over design alternatives that are pre-computed in a batch mode prior to interaction. This Design Gallery system presents the user with a broad selection of perceptually differing graphic images and animations of the same essential content [Ryall97, Marks97]. For rendering, this would involve producing a number of images of a scene under different lighting conditions. The user can then browse through the presented selection of outputs in real-time. The selected outputs are displayed as an arrangement of thumbnail images that may be clicked on to call up a full-size version. An example design gallery is shown in Figure 8. Here multiple renderings of an artificial scene are shown in three rows at the top of the figure. These 24 renderings of the same scene use lights of different type, position and direction. Since the selection of these lighting parameters is made to ensure a broad selection of rendering outputs, it is likely that the user will find a rendering that roughly suits his or her needs.

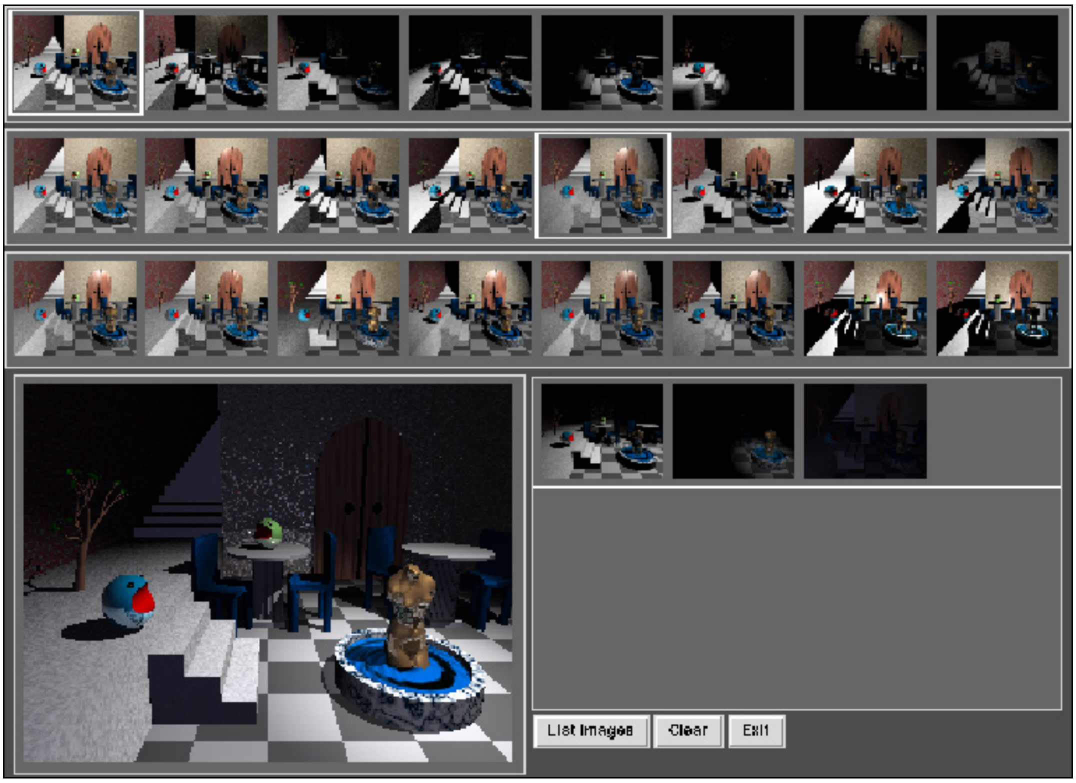


Figure 8: Design Gallery for different light placements [Ryall97].

Although the semi-automatic theme of Ryall's thesis parallels the current thesis, it is more focused on the computer-human collaborative framework than in finding solutions to specific computer graphics problems. It also addresses distinctly different applications and, as will be seen, uses different solution methods.

2.2 Constraint-based graphics software

An alternate form of this framework is constraint-based software, where the user places constraints on the output of a graphical system. The system of constraints that result is then solved, typically with an iterative descent method. This type of constrained editing has been in practice since Sutherland's Sketchpad which introduced interactive constraint-based graphics [Sutherland63]. Other early systems include ThingLab [Borning79], Juno [Nelson85] and Juno2 [Nelson94]. Another constraint-based system, called GLIDE, uses an intuitive local constraint-satisfaction scheme that behaves in a predictable manner from the point of view of the user [Ryall97]. Figure 9 shows an example of using the GLIDE system to transform an arbitrarily arranged graph (left) into a more easily readable form (right). The user has requested two horizontal symmetries by inserting the red and green lines into the graph. The user has also introduced a circular arrangement with the blue circle. Given these user preferences, the system then determines the exact placement of the graph nodes.

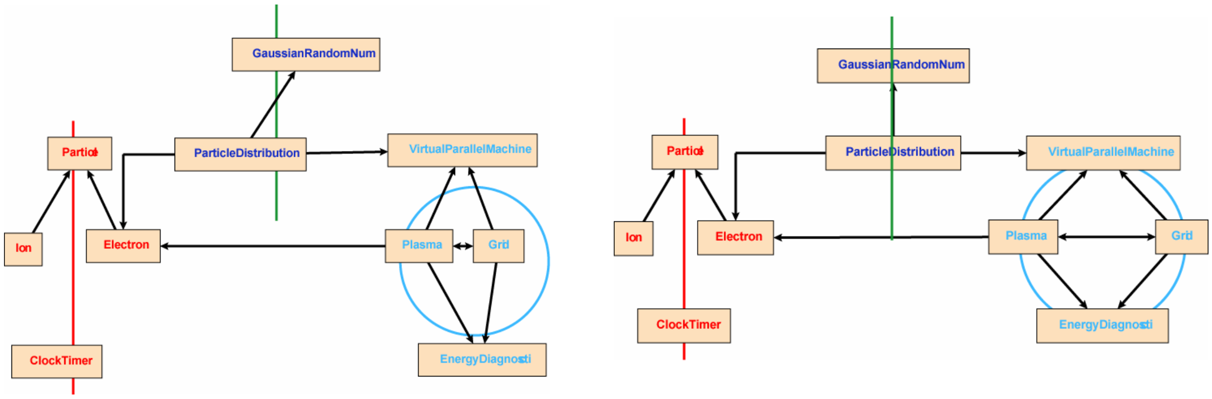


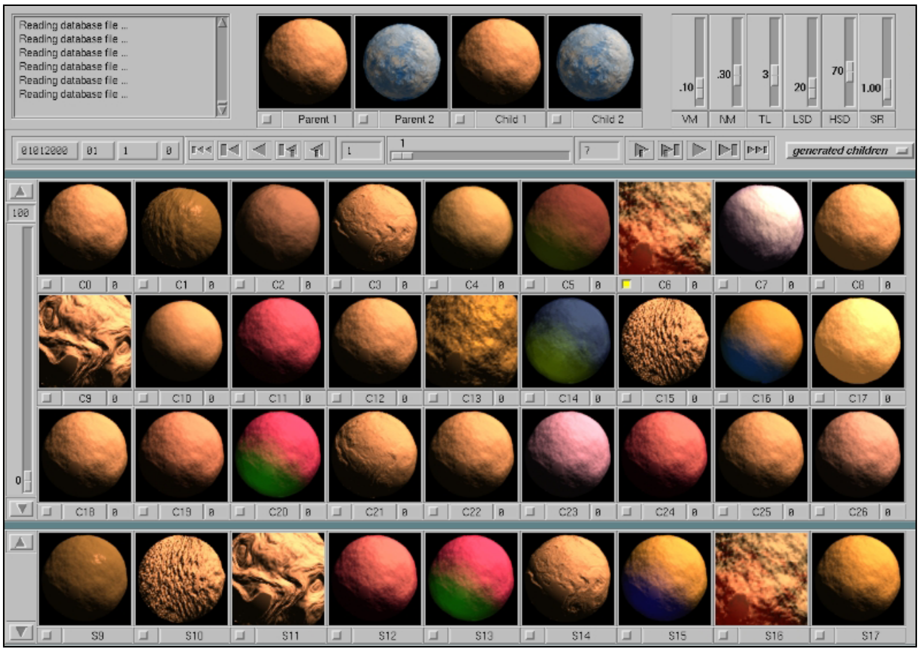
Figure 9: Spatial arrangement of a graph using GLIDE [Ryall97].

2.3 Evolutionary Art

The interactive evolution of textures using genetic algorithms also lies between manual and automatic design methodologies. There are six main concepts involved in evolutionary computation: genotype, phenotype, expression, selection, fitness and reproduction. The genetic pool is comprised of a set of individuals that are each fully determined by their associated genotype which is the genetic information that encodes an individual's characteristics. For textures, this usually takes the form of procedural formulas. The phenotype is then the texture output that results from the developmental rules (genotype). The process that creates the phenotype from the genotype is called the expression and is simply the calculation of texture values at each (x, y) coordinate.

Based on a Darwinian metaphor, the computer's primary role here is to present candidate graphics to the user from the design space. Selection is the process by which individuals are chosen for reproduction and is most often performed by the user according to an implied aesthetic fitness measure. The user picks the surviving graphic objects which are used as a basis from which the next generation of graphical outputs is produced. This can involve both mutation and sexual combination of texture formulas.

Example systems include Karl Sims' LISP-based system for evolving textures [Sims93], Dawkins' generation of recursive tree structured textures [Dawkins86, Dawkins89], Todd's system which is based on repeated atomic elements such as tori and spheres [Todd92] and Genshade which is now a commercial product that evolves high-level Renderman shaders (example results shown in Figure 10) [Ibrahim98]. Also, a few systems have recently emerged that do not rely on interactive user guidance. Rather, they engage in a non-interactive genetic search for target textures [Wiens00].



```
Reading database file ...
Reading database file ...
Reading database file ...
Reading database file ...
Reading database file ...
Reading database file ...
```

Figure 10: Renderman shaders evolved within the Genshade system [Ibrahim98].

However, like other forms of procedurally generated textures, a drawback that genetic based synthesis share is the lack of user override control. Genetic algorithms do not provide scalable control in that a user cannot specify that a particular vertex be a particular colour. If the user requires slight modification of a presented texture then s/he must hope that such a modification occurs in the next generation. Also, although the output is often good, it has the limitation that the generation of many real world structured textures can present difficulties because the genotype is essentially a procedural texture. On the positive side, the evolutionary approach does overcome the black-art specification issue associated with procedural texture synthesis.

2.4 Simulation of Natural Painting and Sketching

Non-Photorealistic Rendering (NPR) assists the user in constructing images that are meant to emulate hand-drawn or hand-painted styles, such as conte sketching, pen and ink illustration, or oil painting [Gooch01]. NPR applications are therefore another type of semi-automation and can often involve the extensive use of texturing. NPR systems can be roughly grouped by the three distinct types of input:

1. images for processing
2. 3D scenes for rendering
3. user direction

There are a number of image processing techniques which transform an image into a style that suggests a painting or other artistic styles [Hertzmann98]. The images can either be photographic or pre-rendered artificially. A subset of these also have the ability to process a progression of images, as from a video or film camera, and to produce a stylized image that remains stable from frame to frame, producing something akin to animated paintings [Hertzmann00].

In the second category, the NPR rendering of 3D scenes produces images of the simulated world in a style other than realism [Meier96]. Some of the more recent systems even generate real-time non-photorealistic renderings and illustrations usually with the aid of pre-computed NPR surface textures [Praun01, Salisbury97].

The third category derives input from the user. For example, the Natural-Media® tools of Corel™ Painter mimic the experience of real painting and drawing [Corel03]. However, user direction has also been integrated with the other two forms of input as well. Haeberli's system allows user control of the brush stroking over existing images [Haeberli90]. Ostromoukhov's digital face engraving system for 3D models relies on the manual specification of line curvature and segmentation [Ostromoukhov99]. The WYSIWYG NPR system developed by Kalnins *et al.* [Kalnins02] also follows the user's direction in that the designer can directly paint a 3D model with naturalistic strokes from one or more viewpoints. The system then adapts the style of the strokes to maintain the same hand-painted look from any chosen viewpoint. Figure 11 shows two alternate viewpoints of a NPR rendered cup and saucer scene. The user has hand-painted the sketched surface properties and the system subsequently adapts these sketch strokes to arbitrary viewpoints.

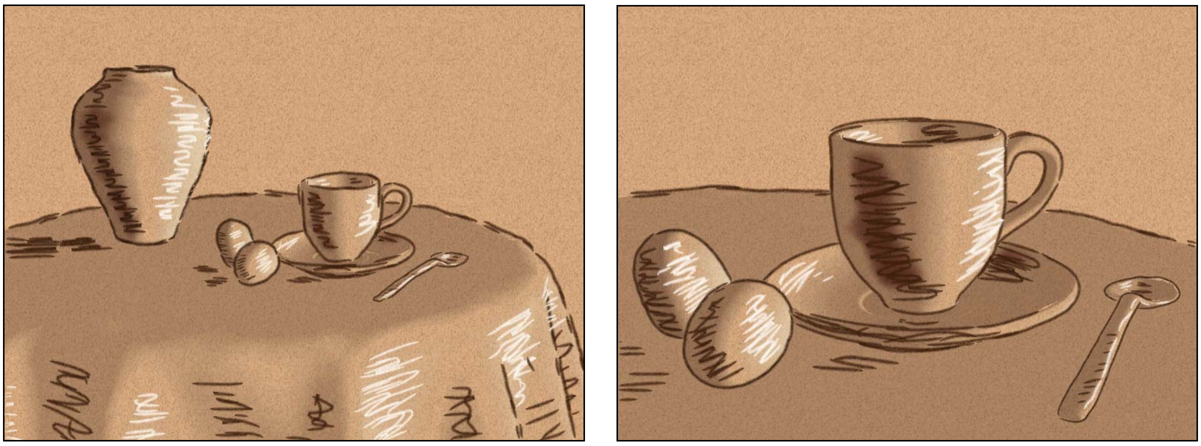


Figure 11: Adaptive non-photo-realistic rendering of an artificial scene [Kalnins02].

But, perhaps the most extreme form of automation in NPR (or otherwise) that still permits some degree of user input is the image stylization system of DeCarlo *et al.* Here the user's task is simply to view the image in a natural way [DeCarlo02]. The focus of the user's eyes is tracked during the viewing. The duration that the user lingers over each portion of the image is used to assign priority to details for a non-photorealistic rendering of the same image.

3. Texture Synthesis

The visual complexity of a scene is as much a function of texturing as it is of the geometry itself. Without realistic textures even a complex model will appear sterile and lack the convincing detail of colour variation and surface imperfection. Producing these realistic textures is by no means an easy task nor is the correct mapping of the textures to a surface [Heckbert89]. Along with correct filtering, this usually requires the skill of a trained artist. Recently, there have been a number of texture methods that have attempted to assist or fully automate this process with ever increasing degrees of success.

3.1 Synthesis by Physical Simulation

There exist a loosely grouped collection of texture synthesis methods that are in some way based on the notion of physical reactions across surfaces. The metaphor of physical simulation is made use of in two slightly different ways. The first is to model the effects of natural interactions of the surface with the environment, examples of which are the weathering, corroding and oxidation of stones and metals (see Figure 12) [Dorsey96, Dorsey99]. The second use of a chemical metaphor is more abstract and generates textures from simplified reactions of what might be chemicals across a surface. As these chemicals react they leave colour traces and it is these traces that comprise the final texture [Turk91, Witkin91, Walter98].

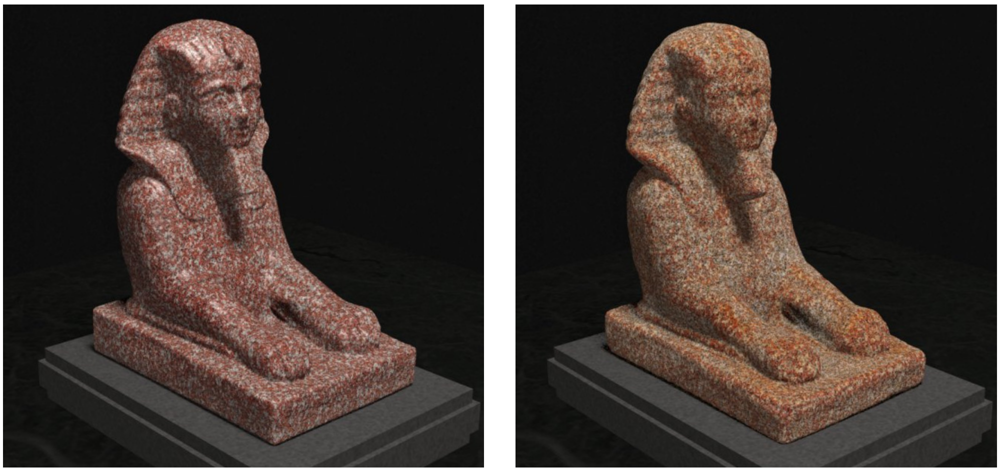


Figure 12: Fresh granite (left) that has been subjected to an erosion simulation by iron and salt (right) [Dorsey99].

Although physically simulated textures can be highly effective, they can be costly to construct and are specifically targeted to narrow classes of surface types. For example, reaction diffusion methods typically only handle animal stripes and spots well. In addition, they often do not allow for significant user control.

3.2 Synthesis by Example

Texture synthesis by example is algorithmically close to the present work so we will look at this branch of research in some detail. The goal of example-based texture synthesis is to generate a new instance of a texture image that appears to be from the same source as a given input texture. The advantage is that textures can be generated to an arbitrarily large size and be made seamlessly tilable at the edges. As there has been a flood of research on this topic in recent years, it is informative to impose a structure on the variety of algorithms that perform this task. We examine the methods in light of two polarities:

1. Single-level vs. multi-scaled
2. Patch based vs. pixel based

These two polarities are concerned with spatial frequency content and synthesis element size respectively. Single-level algorithms simply operate on the original texture itself, while multi-scaled methods decompose the input texture into multiple levels of detail for increased efficiency and for the additional structural information that it can provide. The difference between pixel and patch based synthesis concerns the size of the atomic building blocks. Pixel-based synthesis uses individual pixels to generate a new texture whereas patch-based methods construct the new texture using coherent pieces of the original texture.

For clarity, we will place each of the associated references in a table according to these categories:

	Single-Level	Multi-Scaled
Pixel-Based	[Ashikhmin01] [Efros99] [Harrison01] [Nealen03]	[Bar-Joseph01] [Heeger95] [Hertzmann01] [Wei00]
Patch-Based	[Ashikhmin01] [Cohen03] [Efros01] [Kwatra03] [Liang01] [Nealen03]	[DeBonet97] [Stahlhut03]

Table 1: Four categories of texture synthesis.

Before we examine each of these four categories, we note that Ashikhmin [Ashikhmin01] incorporates aspects of both patch and pixel based approaches as do Nealen and Alexa [Nealen03]. Moreover, the early work of Heeger and Bergen [Heeger95] does not fit neatly into this framework, because although it is multi-scaled, it is neither patch nor pixel based. However, we shall consider these special cases at relevant points within the framework.

3.2.1 Pixel-Based, Single-Level

The simplest method of texture synthesis is the non-parametric method of Efros and Leung which we refer to as EL [Efros99]. The system generates a new texture from an initial random seed on a pixel by pixel basis. Based on a Markov model of textures, the conditional distribution of a pixel given its local neighbourhood is implicitly estimated by sampling the pixels in the original texture which possess similar neighbourhoods. For example, in the right (zoomed) image of Figure 13 a new instance of a texture is being synthesized in scan line fashion, one pixel at a time. The next pixel colour (bordered with green) needs to be determined. This is done by probabilistically sampling those pixel colours from the original texture (shown to the left) that have similar causal neighbourhoods (shown as a blue grid). The amount of randomness is set by a user controlled parameter, without which the system would simply produced copies of the original texture. The results of this synthesis method are fairly good but can suffer from excessive blurring due to the effects that result from using the Level 2 norm as a similarity measure between neighbourhoods.

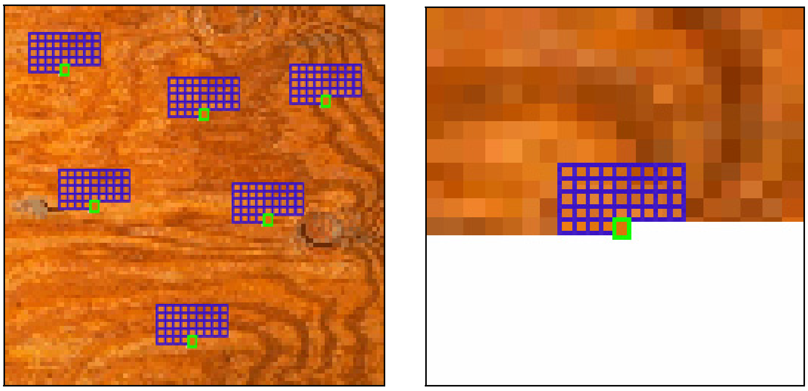


Figure 13: Non-parametric synthesis.

Ashikhmin modifies the EL algorithm by exploiting the correlation between neighbouring pixels to optimize the synthesis process and maintain coherence within the local area [Ashikhmin01]. By only considering the predicted neighbours from already synthesized pixels, this EL variant encourages the verbatim copying of pieces of the input sample. The algorithm effectively produces irregular vertical strips of coherent texture as can be seen in Figure 14. This algorithm is fast and works well for a certain class of texture which the author describes as those consisting of an arrangement of small objects. The method encounters difficulty when applied to more continuously varying textures. As noted earlier, the Ashikhmin algorithm blurs the lines between pixel and patch based synthesis. Although each pixel is synthesized separately, by constraining the choice of synthesized colour to be from predicted neighbours, coherent strips of texture are produced which are effectively patches.

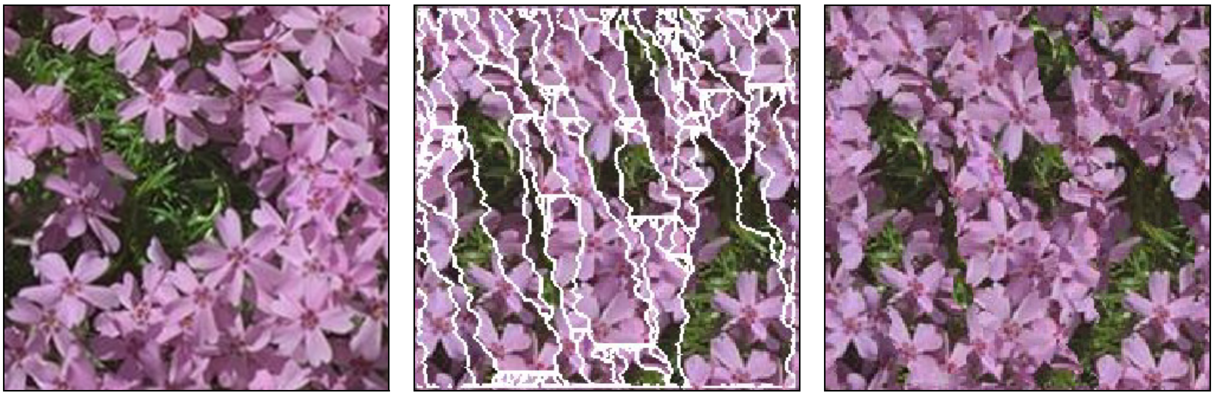


Figure 14: Synthesis of coherent texture strips with the original texture (left) the generated texture strips (centre) and the final result (right) [Ashikhmin01].

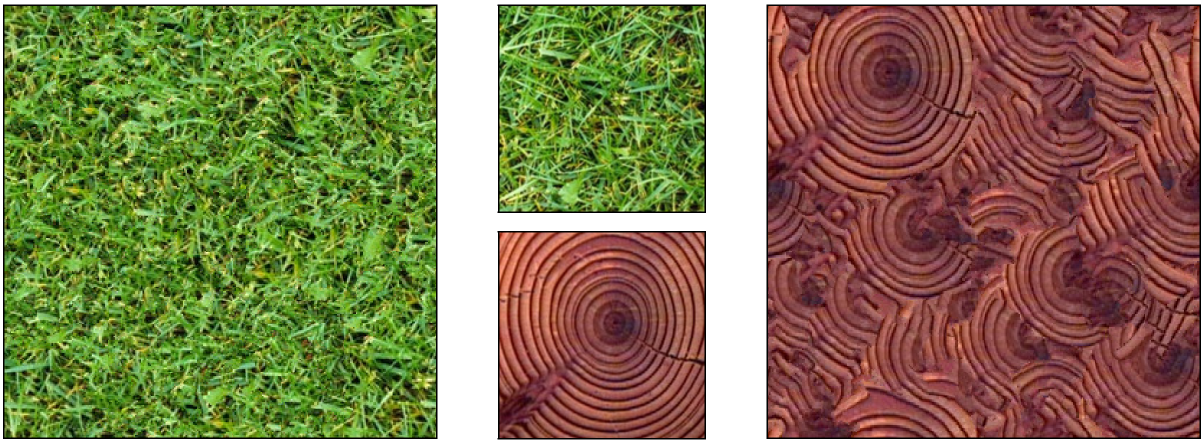


Figure 15: Examples non-hierarchical synthesis results of [Harrison01]. Centre column shows the original textures with re-synthesized texture to the right and left.

Harrison's entropy-based synthesis method is the final in this category [Harrison01]. Harrison proposes that the ordering of pixel synthesis is critical to the quality of the output and that a simple scan line synthesis is not optimal. The results of this work are generally very good as can be seen in Figure 15. However, the synthesis is also particularly slow, requiring many minutes to construct the new texture.

3.2.2 Pixel-Based, Multi-Scale

As multi-resolution approaches to texture synthesis provide the inspiration for the present work, these methods will now be examined in some detail. Multi-resolution methods are among the most recent and also among the most capable in terms of the scope of textures that they can construct. The output of these techniques are, again, synthesized textures of arbitrary size that appear to have been produced from the same underlying process as the finite input textures. In order to construct a new texture, a hierarchy is constructed from the input texture, comprised of multiple spatial frequency bands of the same texture (see Figure 16) [Burt83a].

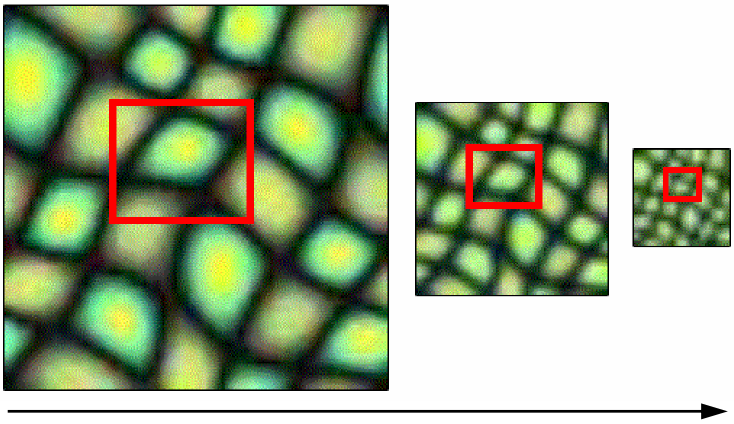


Figure 16: Moving up the spatial frequency hierarchy.

Textures are represented and generated in a hierarchical fashion. The goal is to produce a new texture hierarchy from the original which has structural changes at the macro level yet maintains the intermediate and micro level structures which characterize the input texture. This operates under the assumption that higher levels of the synthesized tree can be re-ordered without affecting the general appearance of the texture. The process of generating the new hierarchy is carried out from the (2 × 2) top level down to the ($n \times n$) texture at the original size of the input texture. There have been three variants of the multi-scale pixel-based approach. The three methods ordered chronologically and also by quality of results are given by Heeger and Bergen [Heeger95], Wei and Levoy [Wei00] and Bar-Joseph *et al.* [Bar-Joseph01].

It is appropriate to begin with the pioneering work of Heeger and Bergen [Heeger95], in which a new instance of a texture is created through hierarchical histogram matching. The method of Heeger *et al.* operates by first generating a new hierarchy of noise (level 0 shown in Figure 17, center). This hierarchy is then manipulated so that the histograms of each level of the new hierarchy match the corresponding histograms of the input hierarchy. They also apply the system to the mixing of two textures by using the colour palette from one texture and the pattern from a second. In Figure 17 we see a successful example of texture synthesis (right) from an input texture (left). An unsuccessful example is shown in Figure 18, indicating that the method tends to perform poorly when faced with more structure in the input texture. Although this work was a great advance over prior work, the results and the range of applicability have been surpassed by recent approaches.

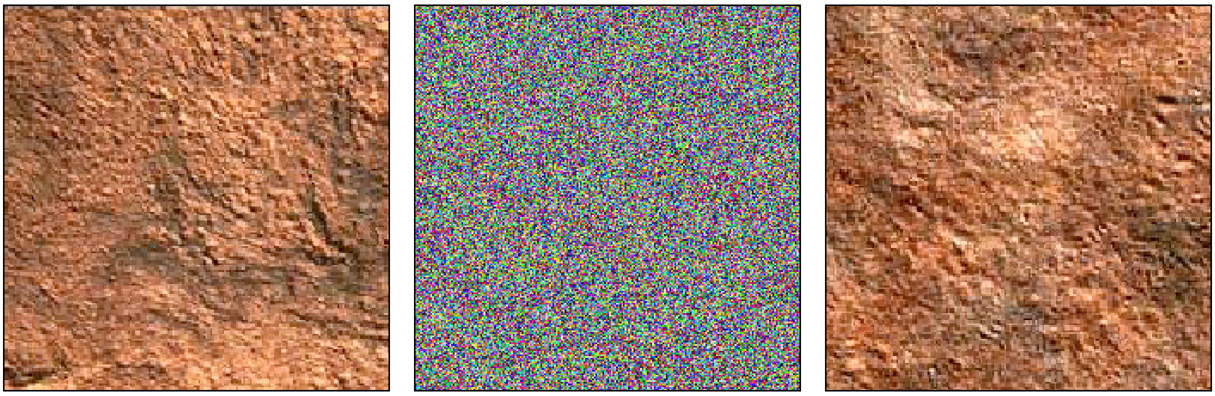


Figure 17: Re-synthesis of a texture [Heeger95]: original input texture (left), level 0 of generated noise pyramid (centre) and synthesized texture (right).

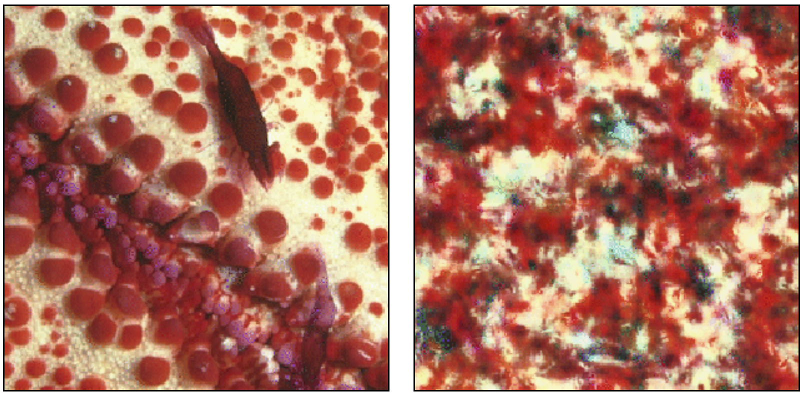


Figure 18: An unsuccessful example of texture synthesis (right) from a structured input texture (left) [Heeger95].

It is also worth describing the multi-scale algorithm of Wei and Levoy [Wei00] as the ideas contained in this thesis use similar algorithms, though for an entirely different purpose. This approach of Wei and Levoy can be viewed as a multi-scale version of the non-parametric synthesis developed by Efros and Leung [Efros99], only differing significantly in the required computation time. From the top-down, each pixel in each level of the new texture hierarchy is chosen from the input texture hierarchy at the same level. As with Efros and Leung, synthesis takes place in scan-line fashion and the local neighbourhoods are strictly above the current pixel in the area that has already been synthesized. However, the levels in the new hierarchy are not synthesized independently. The neighbourhood of the synthesized pixel is augmented to include the corresponding neighbourhoods of all ancestor pixels. The reason for doing so is that the size of the neighbourhood is critical in obtaining correct results, but a larger neighbourhood demands more search time. By including ancestor neighbourhoods the width of the total neighbourhood is implicitly increased. Another innovation lies in applying a tree-structured vector quantization (TSVQ) compression scheme for contracting the search space [Gersho92]. This compression leads to a speed increase of two orders of magnitude though often at the expense of quality. Moderately successful and unsuccessful texture synthesis results are shown in Figure 19 and Figure 20 respectively.

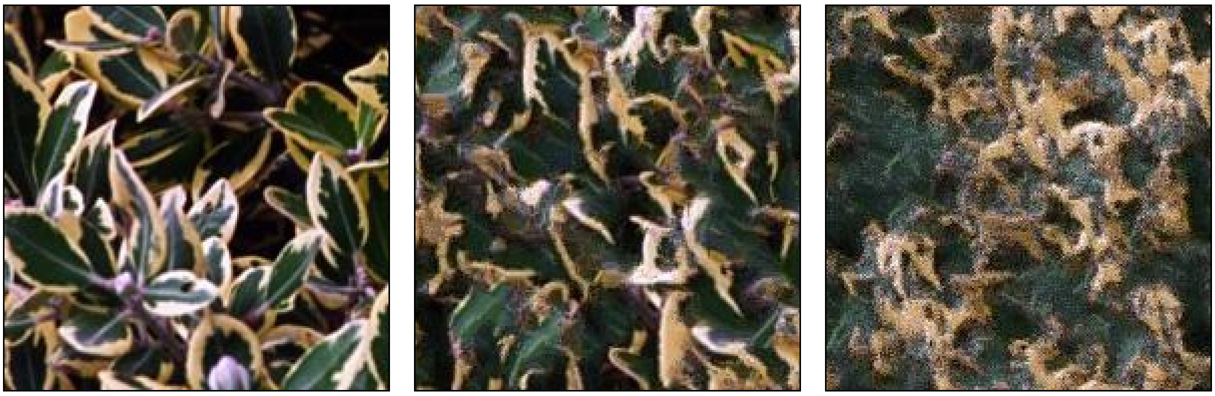


Figure 19: A moderately successful example of texture synthesis [Wei00]: original texture (left), multi-resolution synthesis (centre) and a degraded synthesis using TSVQ compression (right).

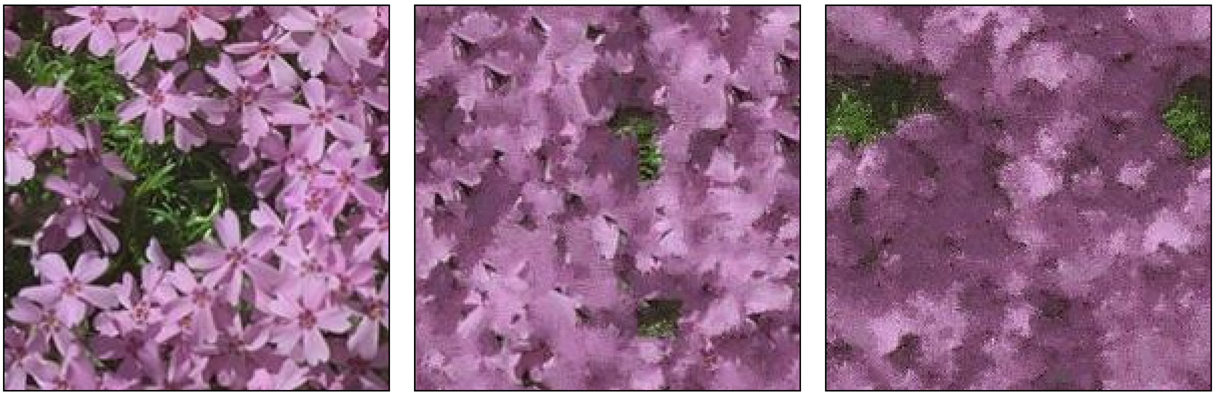


Figure 20: An unsuccessful example of texture synthesis [Wei00]: original texture (left), multi-resolution synthesis (centre) and a degraded synthesis using TSVQ compression (right).

The most recent work in this category is the wavelet based synthesis of Bar-Joseph *et al.* [Bar-Joseph01]. In their approach, the input textures are treated as 2D signals generated by a stochastic process. To capture the qualities of the signal, a wavelet tree transform is constructed from the texture. New trees are then generated from the source tree, sampling the conditional probabilities of the paths in the original tree using a longest matching suffix distance metric. The inversion of these random trees back into signals results in new random textures. Their system produces results that are generally of a good quality (see Figure 21). The same method is also used to generate 2D texture mixtures that simultaneously capture the appearance of a number of different input textures. This is accomplished by probabilistically sampling from both of the texture wavelet trees. Results of this texture mixing appear to be less successful, or at least less meaningful, as can be seen in Figure 22.

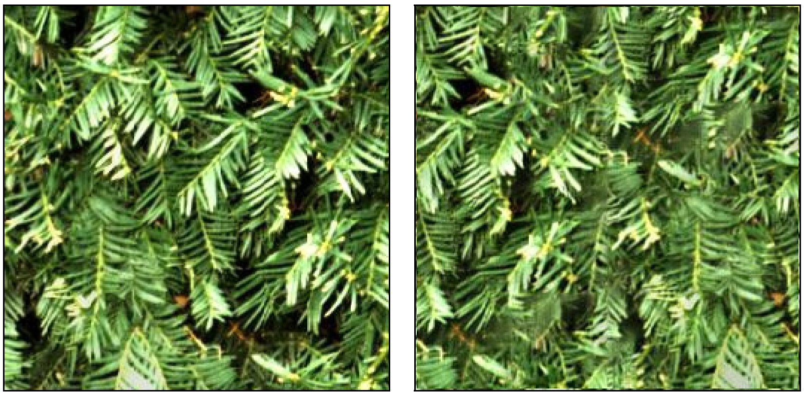


Figure 21: A new instance (right) of a source texture (left) is constructed using wavelet based texture synthesis [Bar-Joseph01].

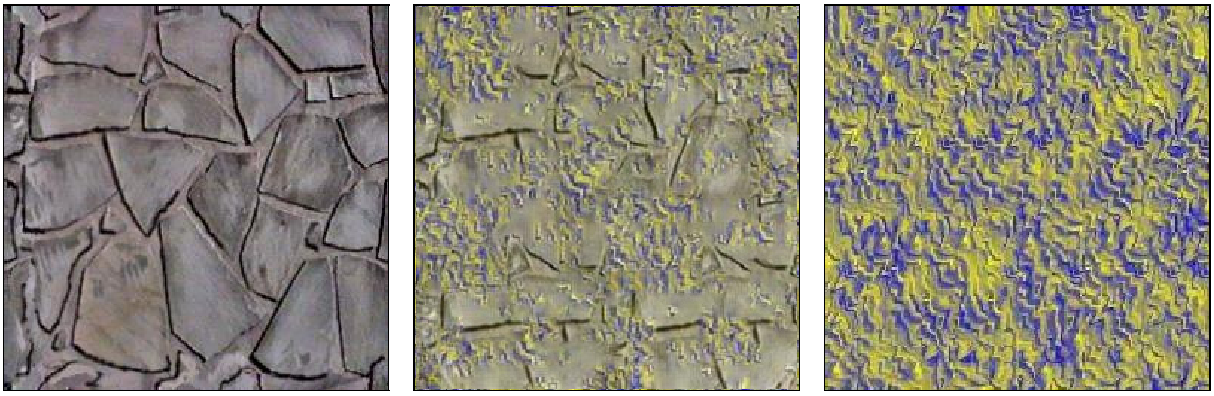


Figure 22: Two textures (right and left) are combined (centre) using wavelet based texture synthesis [Bar-Joseph01].

3.2.3 Patch-Based, Single-Level

Patch based approaches construct a new texture by repeatedly inserting patches of texture of a size greater than a single pixel. The Image Quilting approach of Efros and Freeman [Efros01] generates a grid of partially overlapping block patches (shown in Figure 23, centre) with each block patch taken from the original texture (Figure 23, left). The top left corner of the centre image of Figure 23 shows the mutual overlapping of two blocks B1 and B2. Once the patches are in place, the system executes a minimum-error-boundary-cut within the overlapping region thereby reducing discontinuity artefacts. The top left corner of the right image of Figure 23 shows the optimal cut between the two blocks B1 and B2.

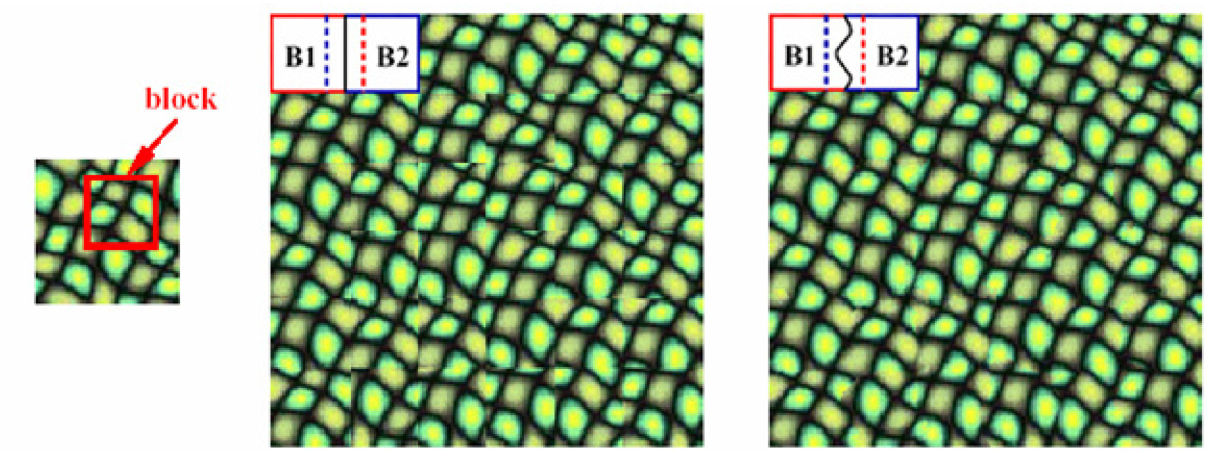


Figure 23: Image Quilting: block patch taken from the original texture (left), a grid of partially overlapping block patches (centre) and the final result of the minimum-error-boundary-cut within the overlapping block regions [Efros01].

Image Quilting generally produces high quality results. However, there remain some artefacts which are primarily due to the imposition of the grid patch structure. In Figure 24 we can see a fairly successful synthesis of an apple texture. On close inspection vertical discontinuities become evident in the bottom right quadrant of the synthesized image. Another problem with texture synthesis at this level of detail is highlighted by the fact that it is obvious that all fifteen of the apples which have their bases pointing toward the viewer are actually copies of the same source apple.

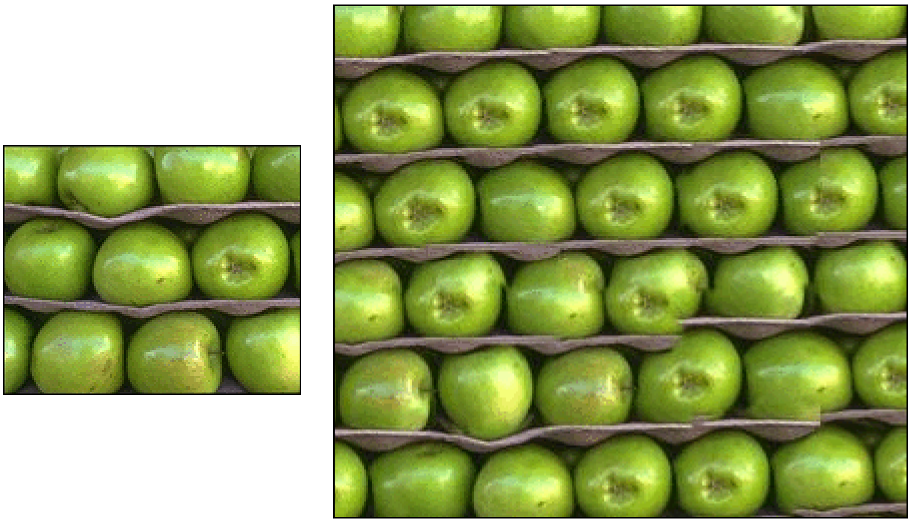


Figure 24: Partially successful synthesis using Image Quilting [Efros01]. Original texture is left, with synthesized output to the right.

A similar technique is used by Liang *et al.* [Liang01] to insert patches into the newly synthesized texture. However, instead of using a minimum-error-boundary-cut, they simply "feather" the overlapping regions with alpha blending. The results of Liang *et al.* tend to be less effective than the method of Efros *et al.*, however the simplicity of the approach does allow for real-time rates of synthesis.

The hybrid texture synthesis method of Nealen and Alexa merges aspect of both patch and pixel based synthesis [Nealen03]. They argue that patch-based synthesis is able to better preserve global structure but often introduces visual artefacts at patch boundaries. On the other hand, pixel-based synthesis tends to blur small objects. By combining approaches, they aim to overcome the deficiencies of both. The method adaptively splits patches so as to use patches as large as possible. In addition, remaining errors in the overlap regions are eliminated using pixel based re-synthesis. The results from hybrid texture synthesis tend to be of a fairly similar quality to those of Image Quilting, but it does appear to perform slightly better in certain cases, as can be see in Figure 25.

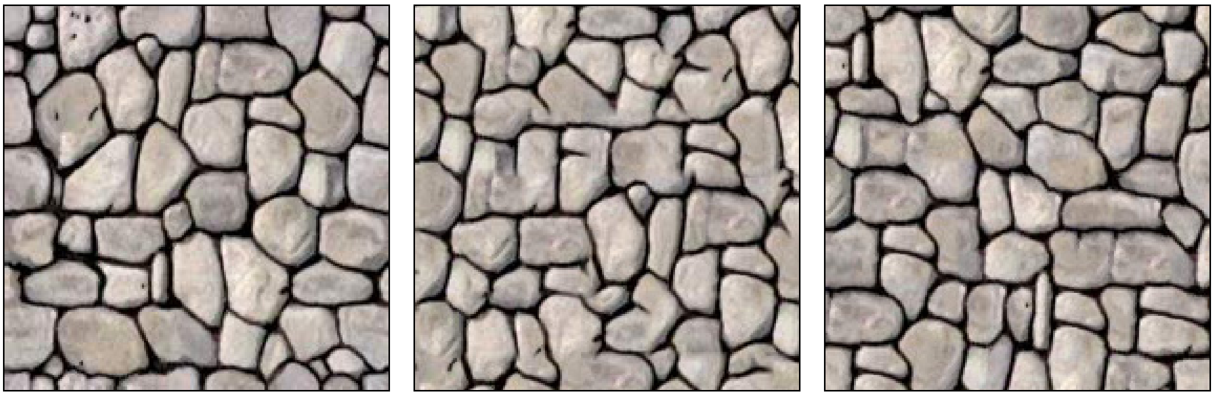


Figure 25: Original texture (left), Image Quilting synthesis (centre) and hybrid texture synthesis (right) [Nealen03].

We will also mention two further approaches to texture synthesis developed concurrently with this thesis: Wang Tiles [Cohen03] and Graph Cut Textures [Kwatra03]. Wang Tiles is a simple stochastic system for non-periodic tiling of texture patches which are pre-computed using an Image Quilting [Efros01] construction. The primary contribution of Wang Tiles is the speed gain that results from the simple snapping together of pre-computed tiles which have mutually compatible borders. Twelve Wang tiles marked *a* to *m* are shown in Figure 26 (left) along with the actual content of the twelve tiles in the centre. Tiles can be placed against other tiles which have the same colour label along a border. A final result of this process is shown in Figure 26 (right). As Image Quilting is used to construct the individual tiles, the quality of the results is roughly equivalent to that of Efros *et al.*

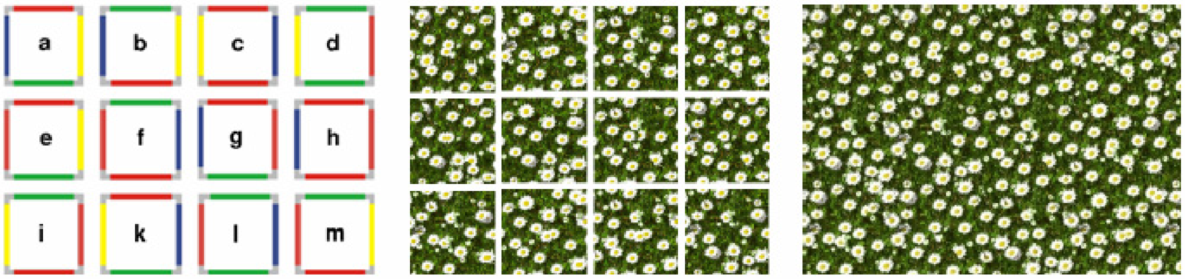


Figure 26: Pre-computed Wang tiles are shown figuratively (left) and as pre-rendered texture pieces (centre). The final result of snapping together the tiles is shown to the right [Cohen03].

Graph Cut Textures [Kwatra03] also improves upon Image Quilting by avoiding the artificial grid structure that Image Quilting imposes. The principal of cutting a path between texture patches is again used for Graph Cut Textures, but in this case the cuts can be of an arbitrary shape (see Figure 27). By loosening the grid constraint, the system is often able to find better paths. And so, unlike Wang tiles, the improvement here is of quality, not efficiency.

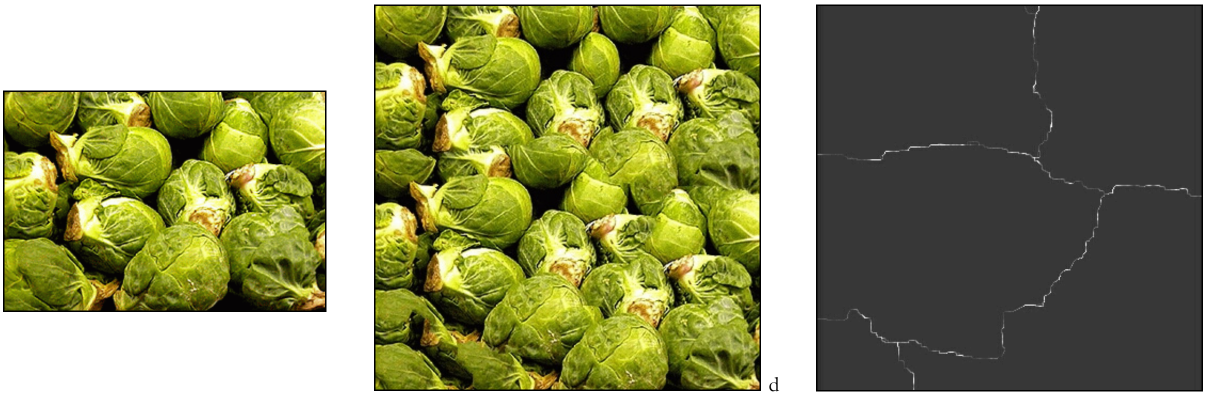


Figure 27: Graph Cut Synthesis: original texture (right), synthesized texture (middle) and cuts (rights).

3.2.4 Patch-Based, Multi-Scale

De Bonet [DeBonet97] introduced a higher quality variant of the multi-scale [Heeger95] approach. The basic idea that drives DeBonet's algorithm is to swap square regions of the original texture at multiple levels of resolution. The lower-down (higher resolution) in the hierarchy that the swapping occurs, the larger square area of pixels are swapped. But, the swapping does not occur randomly. Finer details are generated while being probabilistically constrained by what has already been produced at the same location at higher levels. That is, the lower levels are subject to conditional probabilities along paths of the tree. Swapping is only permitted when the differences between the swapped areas are below a certain threshold. Those differences are taken into account not just at the same level in the hierarchy but for all areas in the hierarchy that are ancestors of those swapped areas. Here differences are calculated in terms of local responses to oriented filter banks. These chosen filters are capable of discriminating variations across some patches that are perceived by humans to be different textures. As can be seen in Figure 28, the resulting synthesized textures are generally of a higher quality for structured textures than those of Heeger and Bergen [Heeger95], though not as convincing as some of the more recently published work. For example, the concurrent synthesis method of Stahlhut produces high quality results by using neighbourhood comparisons of small patches [Stahlhut03].

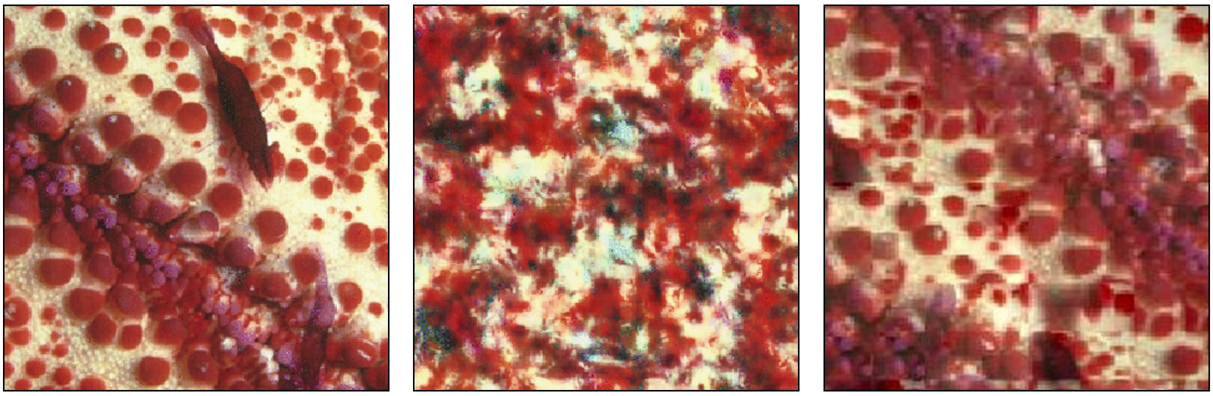


Figure 28: Comparison of synthesis results: original texture (left), synthesis by Heeger and Bergen's method (centre) [Heeger95] and by De Bonet's method (right) [DeBonet97].

3.3 Surface Texture Synthesis

Surface texture synthesis generates a new instance of a texture directly over a 3D geometry. The main benefit of this is that it does not require a complicated mapping to be constructed between a 2D image and a 3D surface for rendering. Many surface texture synthesis methods are adaptations of 2D methods. For example the work of Turk [Turk01] and Wei and Levoy [Wei01] extend the prior pixel-based multi-scale synthesis of Wei and Levoy [Wei00] to work directly on a densely tessellated input mesh.

Dischler *et al.* presents the Texture Particles (TPs) system which extends patch based synthesis techniques by using the distribution of texture elements [Dischler02]. The spatial arrangement of TPs are analyzed and the synthesis is performed by recomposing a similar texture directly on arbitrary surfaces according to particle co-occurrences. Other work by Soler *et al.* extends earlier patch based synthesis methods to 3D surfaces [Soler02] and the concurrent work of [Zhang03] synthesizes progressively variant textures on arbitrary surfaces (see Figure 29). The latter does so by first synthesizing reduced histogram versions of the two input textures which are then used to continuously transition between the two sources.

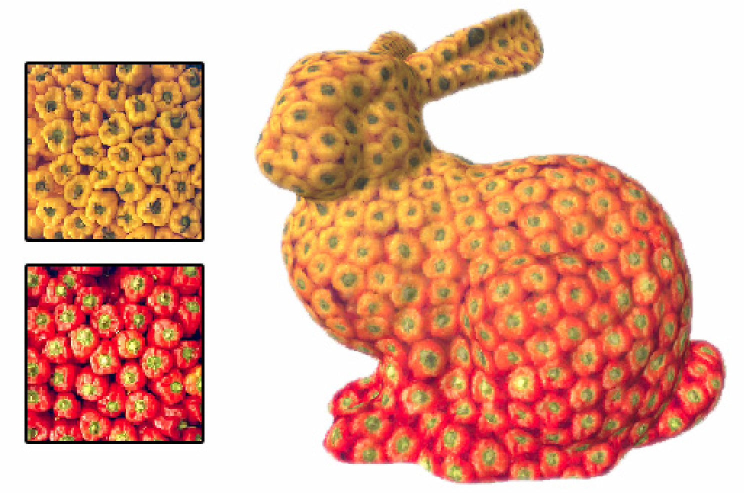


Figure 29: Continuously varying the texture over a surface (right) between two input sources (left) [Zhang03].

Finally, Tong *et al.* extends Ashikhimin's algorithm [Ashikhmin01] to synthesize bi-directional texture functions (BTFs) on arbitrary surfaces [Tong02]. When a surface has prominent macroscopic features, the appearance has a complex dependence on illumination and viewing direction. BTFs represent the appearance of texture as a function of viewing and illumination direction but at a very high memory cost. A result of this work can be seen in Figure 30.

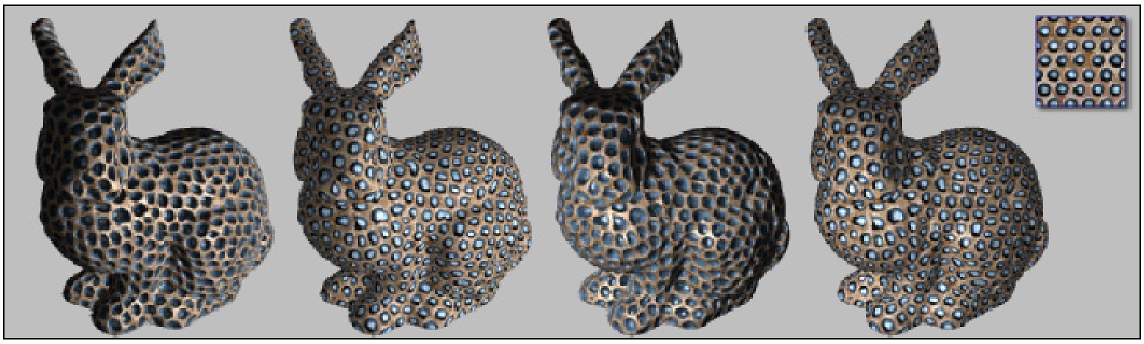


Figure 30: Synthesis of bi-directional texture functions directly on surfaces [Tong02]. The original texture is shown top right.

3.4 Temporal Texture Synthesis

Temporal texture synthesis extends the synthesis of texture in time. Several of the previously mentioned methods have been adapted to this end, such as procedural textures [Eber94]. In fact, there is little computational difference between solid procedural textures and two-dimensional textures that vary over time. Temporal procedural textures are adept at approximating swirling gases and similar phenomenon. Likewise the multi-resolution synthesis of methods of Wei and Levoy [Wei00] and Bar-Joseph [Bar-Joseph01] are extended in time by using a stack of video frames as input and comparing the local neighbourhoods as cubes which extend locally in both time and space. Wei and Levoy's method tends to suffer from a blurring tendency in time, as it does in space, due to its reliance on the L2 norm as a local similarity metric. Bar-Joseph's algorithm is more successful in time as it uses different similarity metrics for time versus space. The space metric is computed from the steerable wavelet, while the Gabor wavelet is used as the metric over time.

The only patch based synthesis method to be used temporally is the Graph Cut Textures algorithm as it is applicable in any dimension [Kwatra03]. It has been used to synthesize texture and video with excellent results. However, there are other approaches, such as Video Textures, which are entirely focused on altering the ordering of frames over time [Schödl00]. Here spatial synthesis does not occur. Instead the set of input frames are analyzed for similarities in order to identify common transition frames.

4. Texture and Image Editing

An entirely new class of image editing tool has emerged which employs advances in computer vision, image blending and texture synthesis to perform sophisticated image editing operations that require only a relatively small measure of user direction. These are related to the present work as they each attempt to assist the user in image editing tasks such as image re-arrangement or in the semi-automatic positioning of image editing tools.

4.1 Directed Texture Synthesis

Tools have been developed that use texture synthesis to remove entire objects from scenes [Igehy97, Drori03, Labrosse03]. Here the user selects a cropping area and a synthesis area. The cropped patch is then replaced with a synthesized texture derived from the synthesis area. After the operation is complete, there are no traces of the content that previously occupied the cropping area.

Similarly, other user-directed synthesis methods [Ashikhmin01, Efros01, Hertzmann01] permit the user to rearrange an entire image to conform to a paint-by-numbers sketch of the final image. Directed synthesis thereby becomes a rearrangement form of texture editing. In Figure 31, a cloud texture has been re-ordered into the shape of the Rainbow lab name. The top left image is the hand painted input map of the original texture which is shown to the top right. The bottom left is the output map which designates the spatial arrangement of the re-ordered texture shown to the bottom right. Together, the two user-provided input and output maps tell the system where the texture has come from and where the texture needs to be synthesized to. We will return to this topic in later chapters.

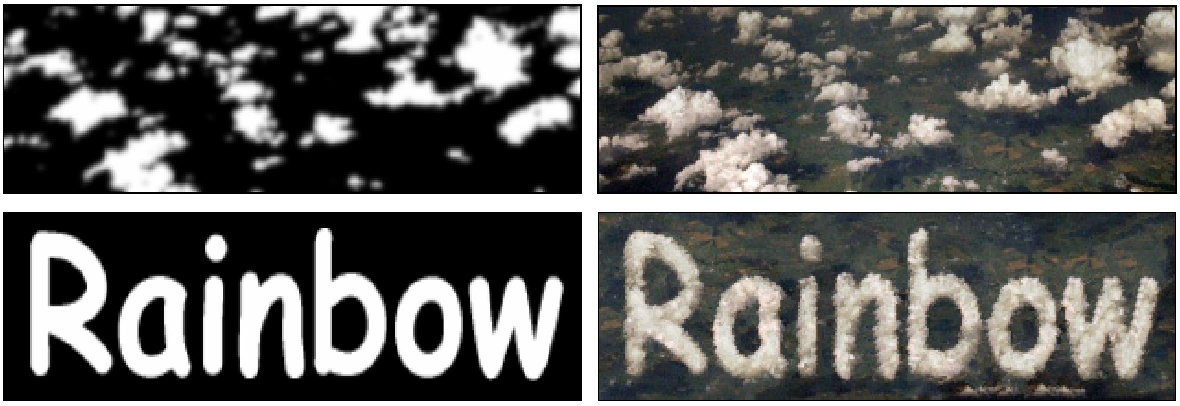


Figure 31: Texture-by-Numbers re-arrangement using Harrison's method. Top row: input map, original texture. Bottom row: output map, rearranged texture.

4.2 Interactive Texture Editing

Early work on direct WYSIWYG painting on the surfaces of 3D objects was published by Hanrahan and Haeberli which allowed the user to apply materials alterations to particular points on the surface [Hanrahan90]. These material attributes include diffuse, specular and surface roughness properties. This method is also direct in the sense that it offers no assistance to the user in the particular application of the material values.

Subsequent work on interactive texture editing sought to assist the user in the editing of textures. For example, Dischler *et al.* [Dischler99] describe a unique hybrid approach to macro-structured surface generation that combines texture analysis and geometric modelling. The complex textures that are produced through this process are visually quite interesting, but it might be claimed that they are also unusual and do not appear to be representative of a wide range of textures.

The Live Paint system of Perlin and Velho [Perlin95] can be seen as an attempt to provide some control over procedural image synthesis within a 2D painting program. Perlin and Velho use the concept of a multi-resolution painting system, first introduced by Berman *et al.* [Berman94] that allows the user to linearly combine procedural textures at multiple scales. But in practice the resulting texture tends to produce combinations of disjoint textures as can be seen from Figure 32. Lewis' [Lewis84] early paper presents another interactive procedure for generating textures in the frequency domain. Although the idea is novel and interesting, the resulting textures are limited.

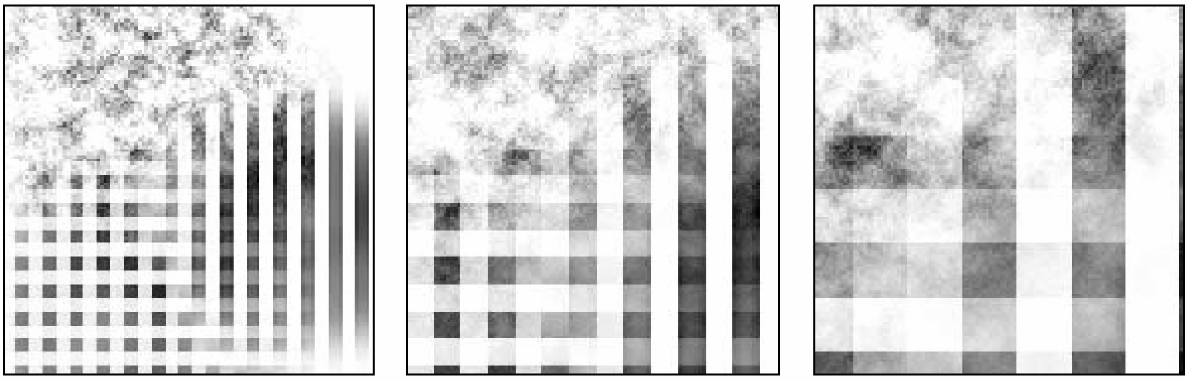


Figure 32: LivePaint Results [Perlin95].

A system which manipulates vector based 'textures' is the search and replace method of Kurlander and Bier [Kurlander88, Kurlander92]. Conceptually, this system is most similar to the work of this thesis. However, their system differs significantly both algorithmically and in terms of the user's work flow as it operates on vector images that are composed of distinct geometrically defined objects, unlike the self-similarity based raster image editing tools that will be presented in later chapters. Figure 33 shows an example of using the tool to replace all instances of a yellow traffic sign with an orange one, respecting the position and orientation of each instance.

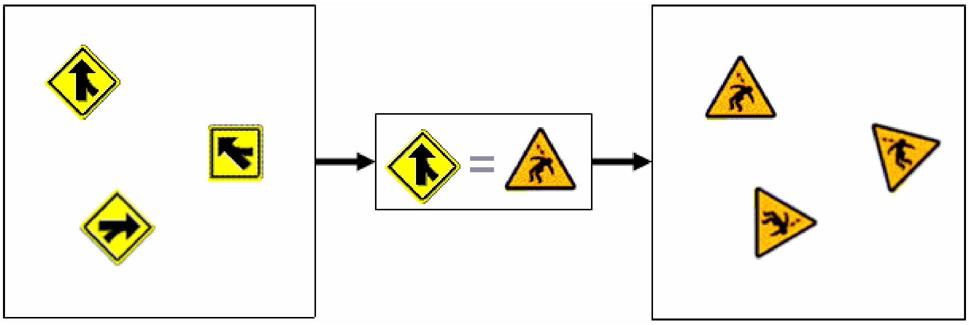


Figure 33: Vector-object search and replace.

4.3 Semi-Automatic Image Editing

Another fruitful source of user assistance in image editing has come from advances in the computer vision community [Cameron99] including intelligent image selection [Mortensen95] and snapping [Gleicher95] tools. An example of intelligent selection is shown in Figure 34. When using this tool, the user only needs to roughly select the outline of the object that s/he is interested in. The system then automatically adjusts the selection curve to better match object's outline.

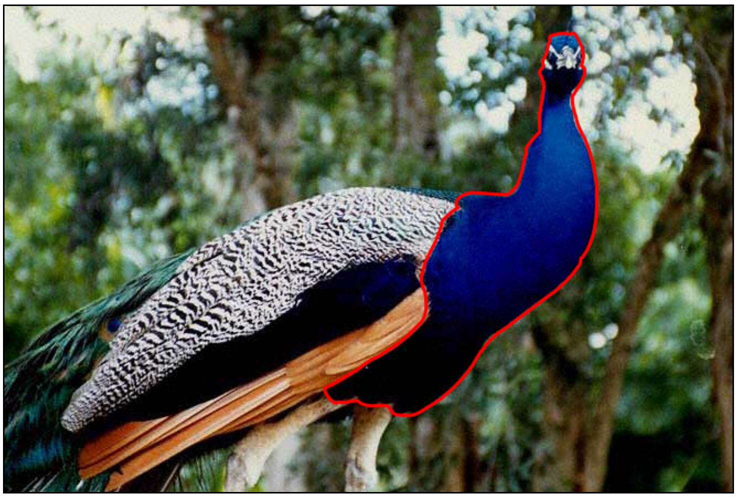


Figure 34: Active Contours: semi-automatic selection of object boundaries.

The concept of active contours (or snakes) has been pushed further in work of Labrosse *et al.* wherein active contours are used as a basis for constructing continuous representations of raster images [Froumentin00, Labrosse01]. Under modest user guidance, the image is segmented into structural regions which are converted into a vector form first using a method of relaxation labelling and then using active contours for further accuracy. Once in vector form the image can be more easily subjected to operations such as zooming and colourization as well as various spatial deformations.

Another characteristic example of an image editing system which employs both texture synthesis and simple computer vision techniques is Barrett and Cheney's object-based image editing system [Barrett02]. In their interactive system, image objects are user-selected by manually collecting sub-object regions detected by a watershed algorithm. Once selected, image objects can be scaled, stretched, bent, warped or even deleted with automatic hole filling. However, their system is not targeted for editing texture images and the manual sub-object collection phase can be cumbersome for the user.

Other related applications with an interactive component have emerged and deserve a brief note. The first uses a combination of low pass filtering and stochastic texture synthesis to remove noise from images [Hirani96]. Elder and Goldberg [Elder98] also offer a novel editing system that operates in an invertible contour domain.

And finally, since we will be introducing a modified texture cloning tool there are unique forms of image compositing [Porter84, Froumentin99] that deserve mention. Burt and Adelson introduced the concept of the multi-resolution image spline for blending between two images [Burt83b]. The primary insight in their paper is that there is no ideal width for a blending filter between two images. If the width is too great then the high-frequency details are lost; if the width is too narrow then the transition zone between the two images will be too narrow, appearing discontinuous. They address this problem by decomposing the image into multiple spatial frequency bands. They then use a blending filter width which is appropriate for each distinct band, giving lower frequency bands a wider filter than the higher frequency bands. An example of the image blending method is shown in Figure 35.

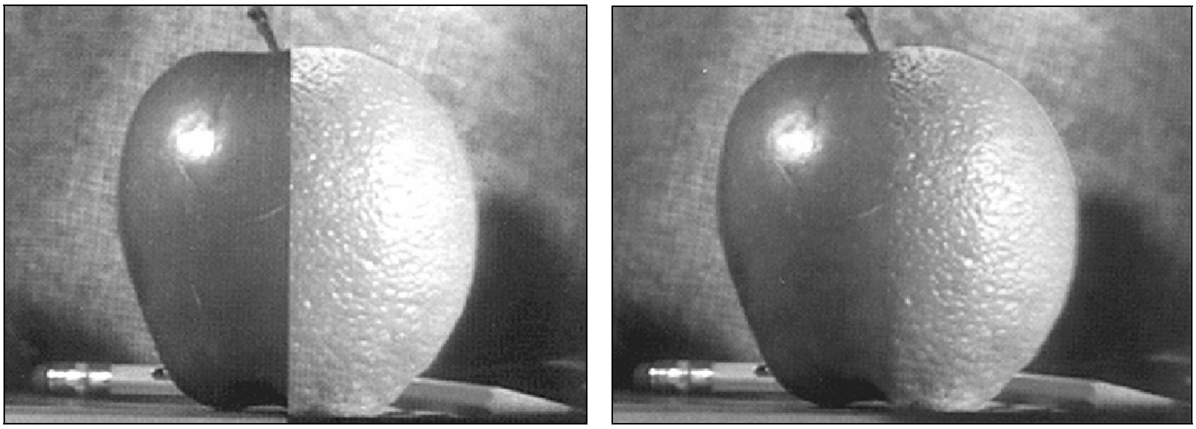


Figure 35: Blending between an apple and an orange. Final result of image spline blending shown to the right [Burt83b].

Pérez *et al.* improve upon the work of Burt and Adelson by formulating the blending problem as a Poisson equation and solving [Pérez03]. In doing so, the method avoids the long range mixing of pixel data in the lower frequencies that can introduce undesirable effects in dissimilar pairs of images. The results are generally excellent as can be seen in Figure 36. In this example multiple image regions including a bear and two children (top two images, left) are manually copied into a water image (bottom image, left). After the regions are copied (centre image), the regions are then blended seamlessly (right image).

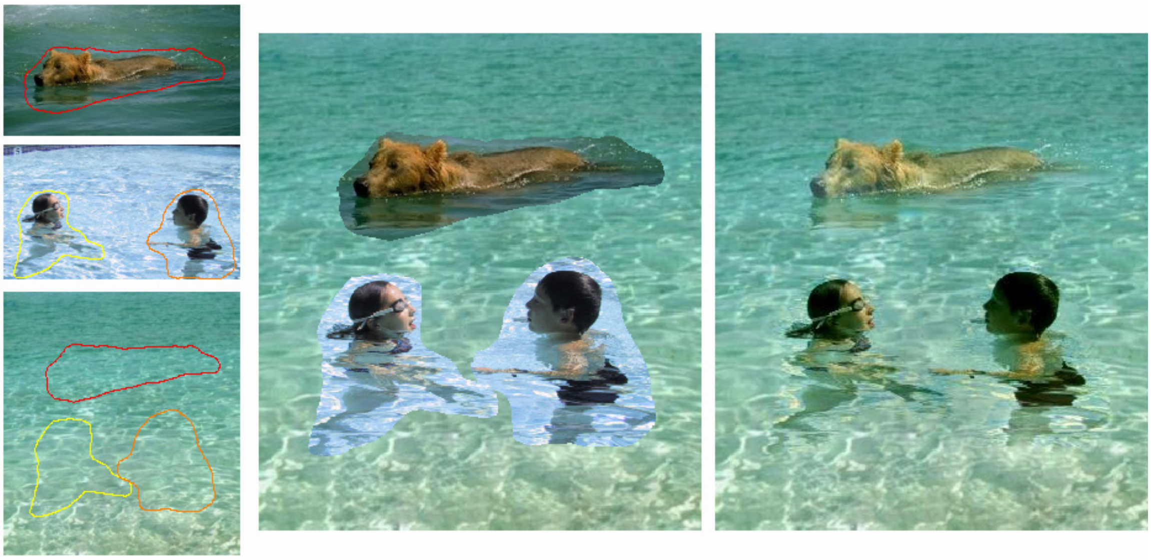


Figure 36: The seamless copying and subsequent seamless blending of image regions [Pérez03].

Chapter Summary

In this chapter, we investigated current state-of-the-art graphics research related to the work of this thesis. Like the present work, many of these systems, such as design galleries and evolutionary art, have explored the semi-automation of common computer graphics tasks. In the section that followed, the relative merits of various methods of texture synthesis were then compared. As we will see in later chapters, a subset of these methods possesses algorithmic similarities to the present work. Finally, we examined a number of texture and image editing systems that offer some assistance to the user when performing complex tasks.

GLOSSARY

Computer-human Collaboration – The computer-human collaboration framework seeks to strike a balance between the responsibilities of the human and the computer. It does so by characterizing the interaction as an optimization problem wherein the user is responsible for the global aspects of the search and the computer is responsible for finding the local minima.

Raster Images – Pixel based images formed as an array of colour dots. These images can have any appearance. Within raster images there are no separate objects and separating one image region from another can be a difficult task.

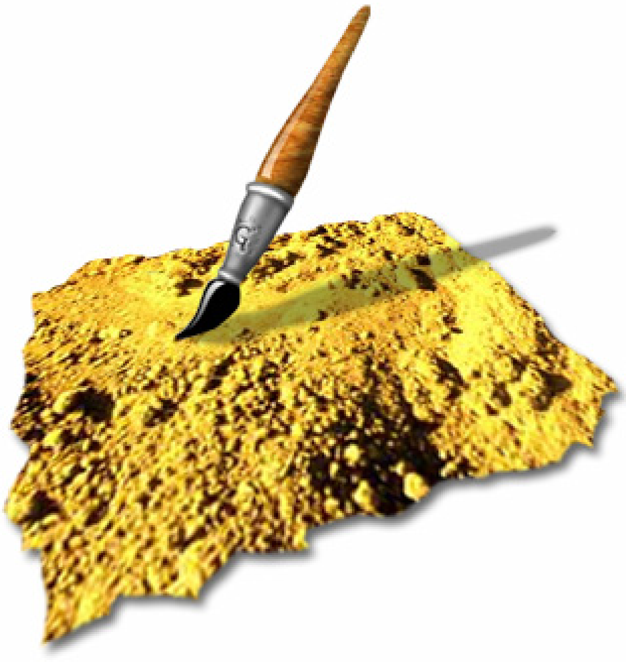
Texture-by-numbers – Also known as user-directed synthesis, these methods permit the user to rearrange an entire image to conform to a paint-by-numbers sketch of the final image. Directed synthesis can therefore be seen as a rearrangement form of texture editing.

Texture Synthesis – The goal of example-based texture synthesis is to generate a new instance of a texture image that appears to be from the same source as a given input texture. Textures can also be synthesised directly over an object surface thereby avoiding the difficulty of mapping a square image over an arbitrary geometry.

Vector Images – Geometrically defined images formed as a collection of lines and filled regions. These images may be characterized visually as containing large regions with the same solid colour. These images generally appear artificial, and it is a trivial task to identify and separate objects within these images.

Chapter 3

SELF-SIMILARITY BASED TEXTURE PAINTING



Chapter Structure

In this chapter, a system of interactive texture editing is presented that utilizes self-similarity to replicate painting operations globally over an image. Section 1 discusses the fundamentals of the self-similarity based painting system, where changes made to a particular pixel are made to affect all pixels that exhibit similar local neighbourhoods. An initial measure of similarity between pixels is presented and the flexibility of the system is improved with the introduction of multi-point Boolean similarity expressions.

In Section 2 the efficiency of the system is improved through the use of a compressed multi-scaled distance measure. This is followed in Section 3 with the improvement of the similarity measure with regards to its perceptual validity and sharpness of response. In the final section the system's interface components are contrasted with an industry standard image editing package.

1. Replicated Texture Painting

In the self-similarity based editing system, changes made to a particular pixel by the user are made to affect all pixels that exhibit similar local neighbourhoods. Inspired by the recent use of pixel-based approaches to texture synthesis, self-similarity based texture editing uses local neighbourhoods to assess the similarity of pixels within a texture. However, neighbourhood matching is not employed to generate new instances of a texture. We instead locate similar neighbourhoods for the purpose of replicating editing operations on the original texture itself. The result is not a new instance of a texture but is a fundamentally new texture. In its simplest form, the system replicates painting operations globally over an image. To illustrate, Figure 37 shows an example of the replicated painting of the colour purple on a texture composed of six balls, with the original texture shown to the left. The pixel under the selection point (shown in green) is painted purple, as are all pixels similar to the user-selected point.

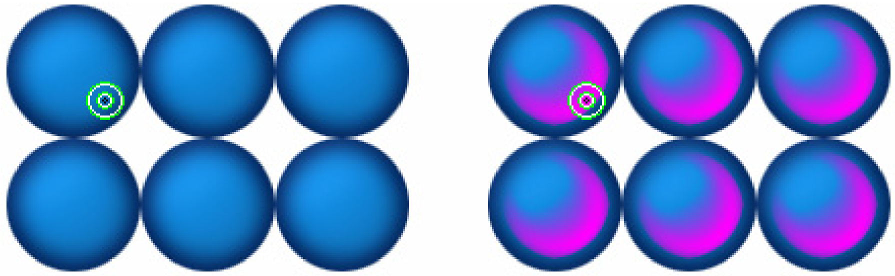


Figure 37: Replicated painting (right) of the colour purple on a ball texture (left).

1.1 An Initial Similarity Measure

Self-similarity based painting alters the colour or brightness of similar pixels to that which the user selects. We therefore need some kind of metric that tells us how similar pixel A is to pixel B. In order to define our initial choice of similarity metric it would be prudent to first define the subset of general images that are considered to belong to the class of texture. However, developing a precise definition of a texture has proved to be a difficult problem which is partly due to the fact that a texture is as much a semantic and perceptual concept as it is technical. Although there is as yet no conclusive definition of texture, there are some distinguishing features that all textures generally possess. The distinctive quality that separates textures from other images is that, at some scale, a given portion of the texture is similar to any other portion.

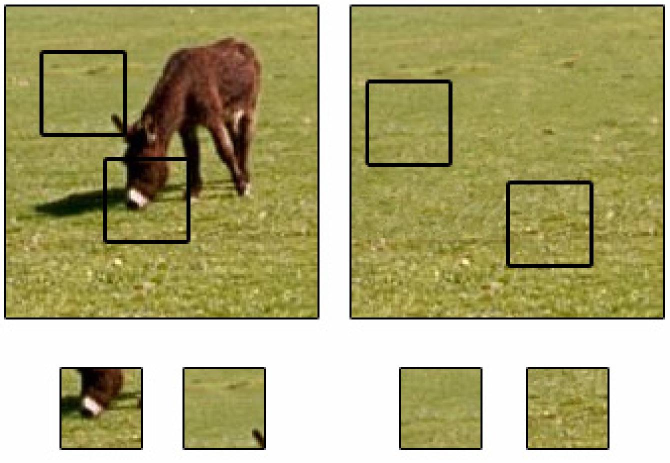


Figure 38: Example of how textures differ from general images. In the top left is shown a general image with two dissimilar windows of the same image beneath. In the top right is shown a texture image with two similar windows of the same image beneath.

1.1.1 Textures as Markov Random Fields

Models of texture have been put forth that aim to clarify this self-similarity criterion and perhaps the most significant formalization of texture images are Markov Random Fields (MRFs) [Zhang00]. MRFs model a texture under the assumption that the image is a local and stationary random process. The image is local if each pixel is predictable from a small set of neighbouring pixels and is independent of the rest of the image. Signals are considered stationary if the statistics that are present in a region are invariant to the region's location. This means that each pixel of a texture image is characterized by a small set of spatially neighbouring pixels, and this characterization is the same anywhere in the image. The intuition behind this model is demonstrated in Figure 38. The top left image shows an example of a general image containing a donkey on a grass field. But, once we remove the donkey (shown right) using Harrison's hole filling method [Harrison01], what remains is a fairly consistent texture image. We can clarify this further if we restrict our viewing of each of the two images by observing them through the small windows shown beneath each of the images. We can imagine that as the windows are moved the viewer observes different parts of the images. If the image is stationary the observable portion always appears similar [Heeger95].

This can be stated more formally. A random field x is a collection of random variables arranged on a lattice L:

$$x = \{x_{i,j} \mid (i,j) \in L\}$$

where the lattice is defined as a regular array of elements:

$$L = \{(i,j) \mid 1 \le i \le N, 1 \le j \le M\}$$

meaning that the random field is simply a set of random pixels and as such can be completely characterized by the field's probability measure $p(x)$. As the random variables $x_{i,j}$ are discrete in the case of digital images, $p(x)$ denotes the joint probability distribution. Although complete in its characterization of the random field, $p(x)$ is also computationally inefficient, because even for modestly sized images (i.e. with $N = M = 256$), $p(x)$ must explicitly characterize the joint statistics of a large number of elements (in the example, $N \times M = 65{,}536$ elements).

Fortunately, textures belong to the more tractable subset of general random fields: Markov Random Fields. Markovianity in a one-dimensional process, x_n, imposes the restriction that past (x_p) and future (x_f) events are decoupled:

$$p(x_f | x_n, x_p) = p(x_f | x_n) \quad \text{and} \quad p(x_p | x_n, x_f) = p(x_p | x_n).$$

In 2D, we refer to a radial boundary, where the elements beyond the boundary have no influence on $p(x)$. Each pixel therefore has a local neighbourhood, $N_{i,j}$ which probabilistically characterizes element $x_{i,j}$:

$$p(x_{i,j} | \{x_{k,l}, (k,l) \in L\}) = p(x_{i,j} | \{x_{k,l}, (k,l) \in N_{i,j}\}).$$

For our texture model, we use Markov Random Fields since they have been shown to cover the widest variety of useful texture types [Wei00]. The success of recent approaches to texture synthesis which consider textures as 2D MRF processes lend further credence to the validity of the model [Ashikhmin01, Harrison01, Nealen03, Turk01, Wei00, Wei01]. As the synthesized instances of texture appear to be from the same source, this implies that the MRF model is adequate for at least the class of textures which it can be used to successfully synthesize.

1.2 Assessing Similarity

As mentioned, to perform a replicated painting operation the user moves what we will refer to as the selection point onto a pixel within the original image. The local circular neighbourhood of the chosen selection point is then compared against that of every other pixel's neighbourhood in the image. 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.

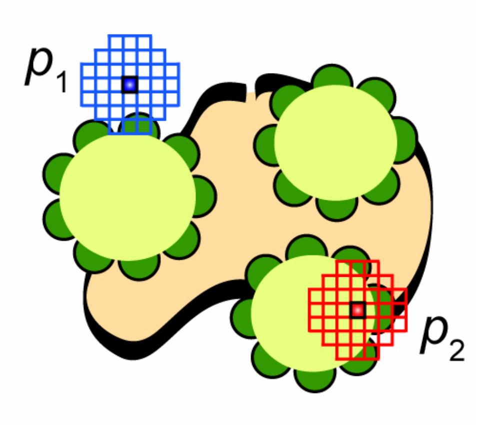


Figure 39: Local neighbourhoods surrounding pixels p_1 and p_2.

The definitions of neighbourhood, threshold and neighbourhood similarity require clarification. For small textures, where efficiency is not a serious constraint, the neighbourhood is simply defined as those pixels bounded by an immediate circle of pixel diameter *d*. Figure 39 shows two pixels p_1 and p_2 with their two corresponding local neighbourhoods shown in blue and red respectively.

The distance measure is the sum of square differences between each corresponding neighbourhood pixel. Specifically, the distance measure between two points, p_1 and p_2 is:

$$d_{SSD}(p_1, p_2) = \sum_{(i,j)\in S} \left(I(x_1+i, y_1+j) - I(x_2+i, y_2+j)\right)^2$$

where:

$$S = \left\{ (i,j) \mid 0 \le \sqrt{i^2+j^2} \le d/2 \right\}$$

I is the image being edited,

d is the neighbourhood diameter,

$p_1 = (x_1, y_1)$ 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.

For example, with a neighbourhood diameter of 9, this will define a circular neighbourhood of 49 pixels. And so, to compute the distance metric between the neighbourhoods requires summing 49 squared differences. We will also note that as we are operating on colour images (see Figure 40), we need to concatenate the neighbourhoods of surrounding the pixels in each of the R, G and B channels leading to the summing of $3 \times 49 = 147$ squared differences in this case.

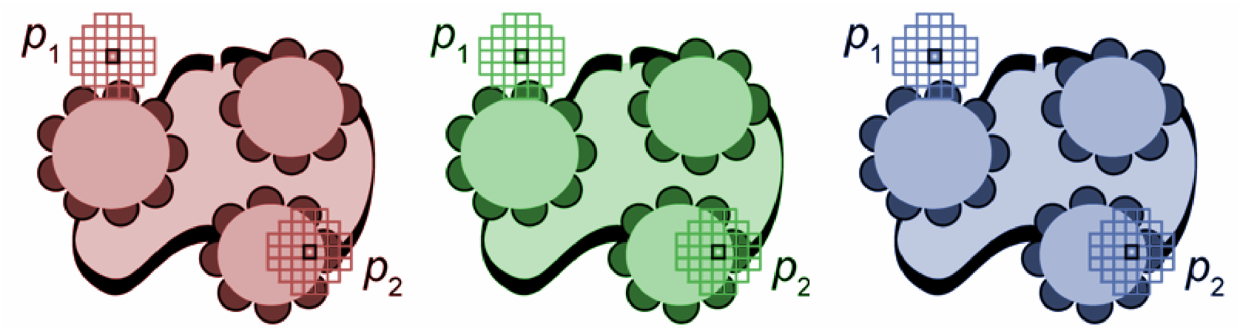


Figure 40: Concatenation of RGB neighbourhoods surrounding pixels A and B.

1.2.1 Weighting the Neighbourhood

As described, the neighbourhood metric assigns the same weight to any mismatched pixel, whether near the centre or at the edge of the neighbourhood. Since we would like to preserve the local structure of the texture as much as possible, pixels further from p_1 and p_2 should have a smaller relative weighting. To achieve this we modify d_{SSD} so that the differences between the neighbourhoods of p_1 and p_2 are appropriately weighted:

$$d_{SSD}(p_1, p_2) = \sum_{(i,j)\in S} M(i,j) \times \left(I(x_1+i, y_1+j) - I(x_2+i, y_2+j)\right)^2$$

where M is a 2D Gaussian kernel. We will return to the definition of Gaussian kernels in Section 4.

1.2.2 Computing the Opacity

In the self-similarity painting system, 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. The user is also provided with a global strength multiplier, which reduces or increases the opacity for all affected pixels. The formula for the opacity of any pixel, p_1, given the user-selected pixel, p_2, is therefore:

$$opacity(p_1, p_2) = \max\left(s \times \left(\frac{t - d_{SSD}(p_1, p_2)}{t} \right), 0 \right)$$

where:

t is the distance threshold set by the user's slider and

s is the global strength, ranging from 0.0 to 1.0.

1.3 Properties of the Painting System

The green texture in Figure 41 succinctly conveys the way in which self-similarity based editing operates from the user's perspective. With the original texture shown to the left, the four remaining images show four consecutive positions that the user has selected within the texture. Moving from the top-middle image clockwise to the bottom-middle image, we see the successive painting of the 'top', 'right', 'bottom' then 'left' sides of all the texture elements simultaneously.

The reader will note two significant visual properties. The first is the soft gradient of each painted region. The second is the directional control of the tool. By directional control we refer to the ability of the user to successively select the 'top', 'right', 'bottom' then 'left' areas of the texture elements. These aspects would not be present if only the pixels themselves were compared without the neighbourhood measure. This texture was chosen for the purpose of illustration because it strongly exhibits these behaviours, but as we will later see, both of these properties are exploitable to a greater or lesser extent in most textures.

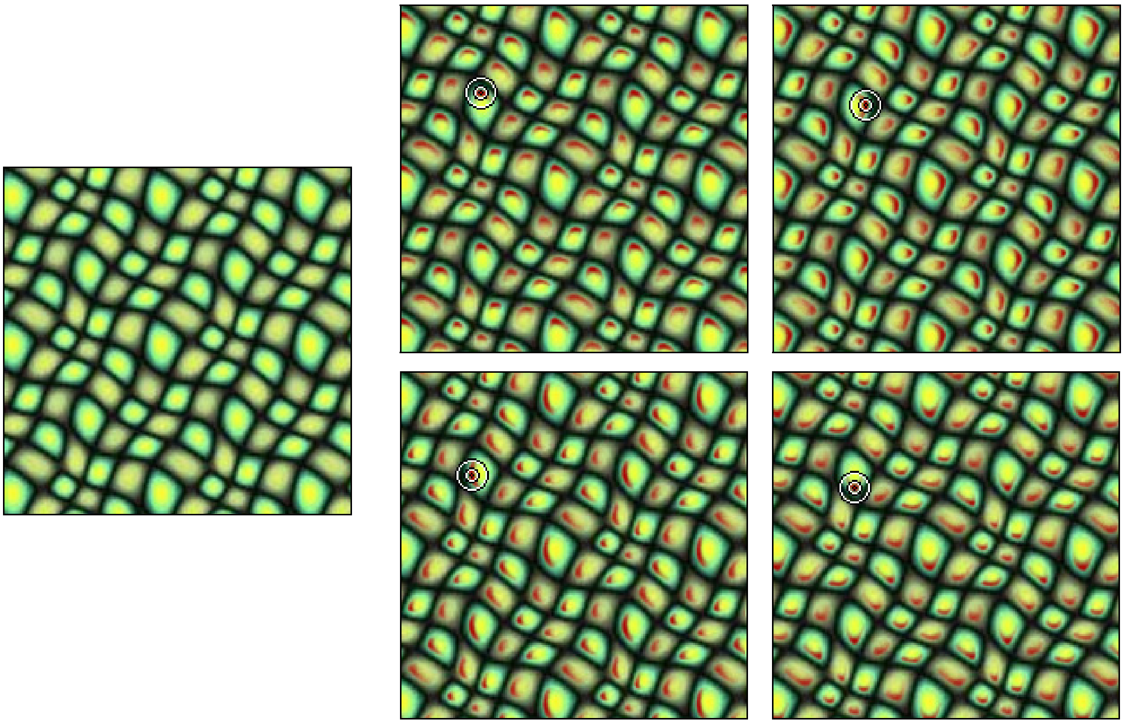


Figure 41: Similarity painting: application of replicated red paint to each of the four sides of the texture elements. Settings for the images: s = 1.0, t = 800K, d = 9.

1.3.1 Applying Multiple Operations

We will next show an example of applying multiple painting operations to the same texture. Figure 42 shows three stages of painting the now familiar green-blob texture. In the left image we have painted red along the bottom of all the texture elements. In the middle image we have applied dark green along the top. And, finally, we have applied a small white dot to the top of all of the texture elements by reducing the distance threshold to 500K from 800K.

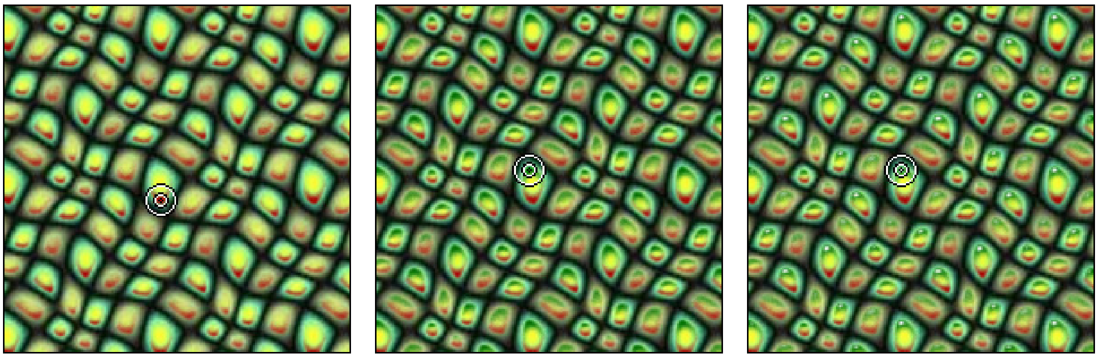


Figure 42: Similarity painting: multiple paintings applied. Settings for left and middle images: s = 1.0, t = 800K, d = 9. Settings for right image: s = 1.0, t = 500K, d = 9.

1.3.2 Altering the Threshold and Strength

As mentioned, the user has two controls which affect the amount of applied paint. The left graph in Figure 43 depicts the altering of the global opacity multiplier (Strength) while maintaining a constant threshold (Distance) of 1.5 million. Conversely, the right graph holds the Strength at 75% with the Distance ranging from 0.5 to 2.0 million. This is simply meant to indicate that the final opacity levels are a function of both the global opacity multiplier and the current threshold.

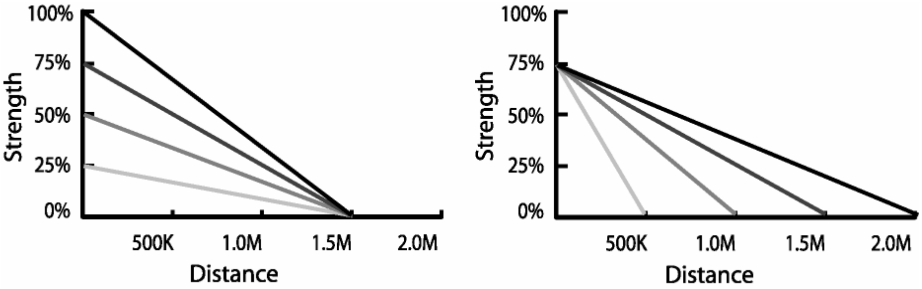


Figure 43: Altering global strength (left) or distance threshold (right).

An example of adjusting the distance threshold is shown in the top row of Figure 44. Note how as the threshold is increased, more and more pixels are affected. An example of holding the threshold constant while adjusting the distance global opacity multiplier is shown in the bottom row of Figure 44. Note how the same numbers of pixels are affected, yet they are painted over with increasingly opaque colour. Together the two controls can be used to control the number of pixels that are affected as well as applied opacity.

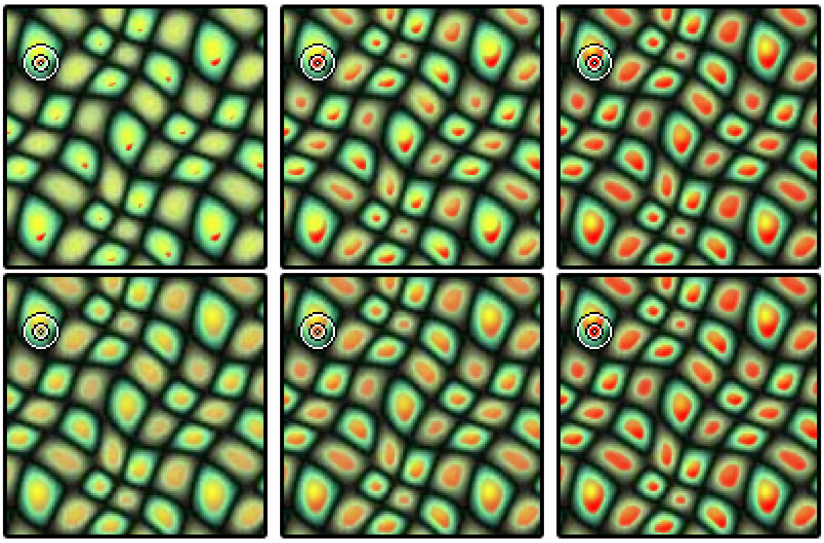


Figure 44: Top row: increasing the distance threshold from left to right. Bottom row: increasing the global strength from left to right. The right-hand images in both rows have the same settings.

1.4 Boolean Similarity Expressions

We now describe an extension to the system that allows the user to select multiple similarity points within the texture which together comprise a Boolean similarity expression. In Figure 45, we see an example of using multiple similarity points. On 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*.

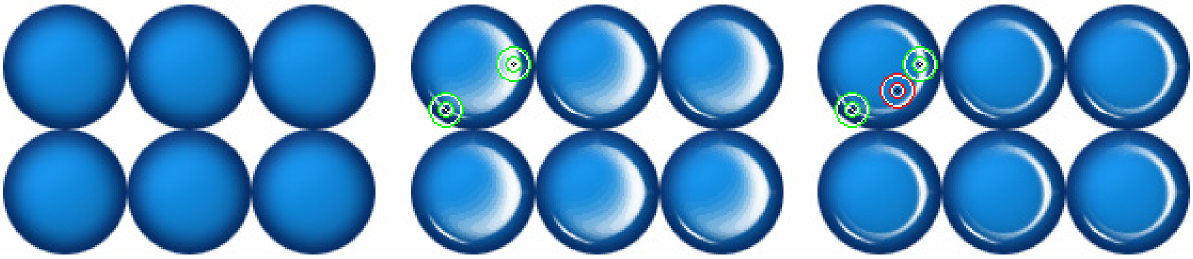


Figure 45: Example use of a multi-point Boolean similarity expression. The green rings denote positively weighted similarity points, and the red is negative. Left: original repeating blue-dot texture. Middle: two positively weighted similarity points used to paint white onto similar pixels. Right: The expression is asking the system to "paint white those pixels that are similar to the green points but dissimilar to the red".

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:

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

where:

t is the distance threshold set by the user's slider,
s is the global strength, ranging from 0.0 to 1.0,
n is the number of positive similarity points and
m is the number of negative similarity points.

1.4.1 Visualization of Boolean Similarity Expressions

If we consider the original case of using a single positive similarity point, then we can visualize the similarity point's neighbourhood as a vector, v, in a high dimensional space R^{147}. In reality, with a diameter of 9 this neighbourhood is a 147 dimensional vector (49 sets of RGB values), but for the purposes of visualization we will imagine that the neighbourhood vector of the selected point is a single set of RGB values in R^3 (see Figure 46, left). Surrounding the vector v is a density function representing its similarity to other neighbourhoods in the space. At v itself, this similarity value is 1.0 which tails off to 0.0 as the distance threshold is reached. Note that the density function is not depicted as spherical due to the relative weightings of the Gaussian, $M(i, j)$, which vary over each axis.

The introduction of additional positive and negative similarity points further refines the affected area of similar neighbourhoods (see Figure 46, right). But we should note that Boolean similarity expressions are not Boolean in the strict sense, as the positive and negative areas of influence are density clouds and do not delineate discrete boundaries.

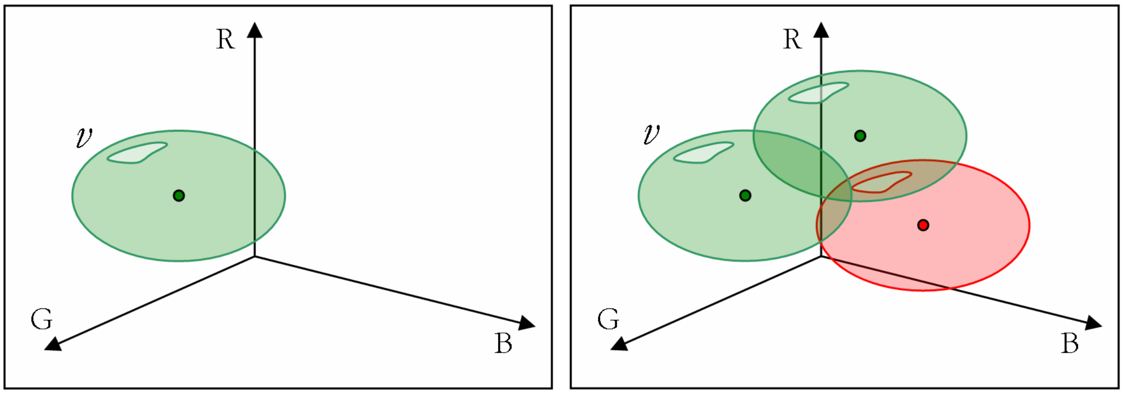


Figure 46: Illustration of similarity neighbourhoods in R^3. Left: single positive similarity point. Right: two positive similarity points and one negative.

1.4.2 Results

Figure 47 and Figure 48 show the use of multiple similarity point paining. Using both positive and negative similarity points allows the 'similar' regions to be carved out with greater accuracy. And as we will see in later chapters Boolean similarity expressions will provide greater control for all of the self-similarity operations including warping and cloning.

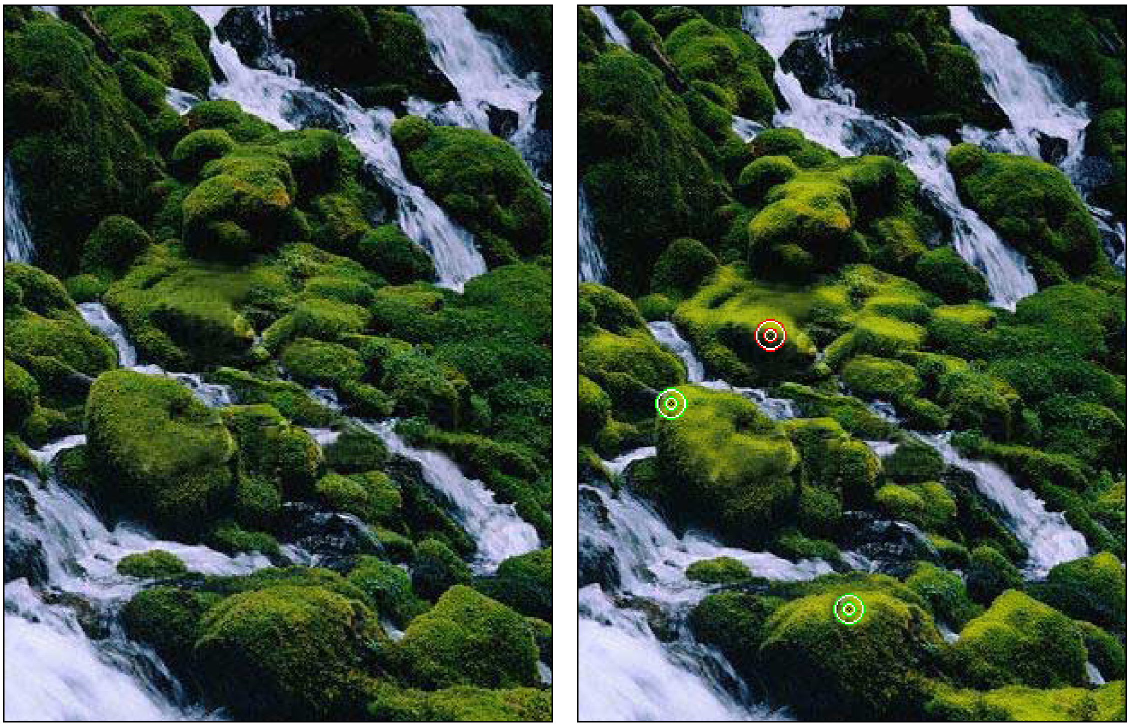


Figure 47: Painting of sunlight (yellow) on top side of mossy rocks. The negative (red) similarity point is positioned just under the edge of a mossy rock to further restrict the application of paint.

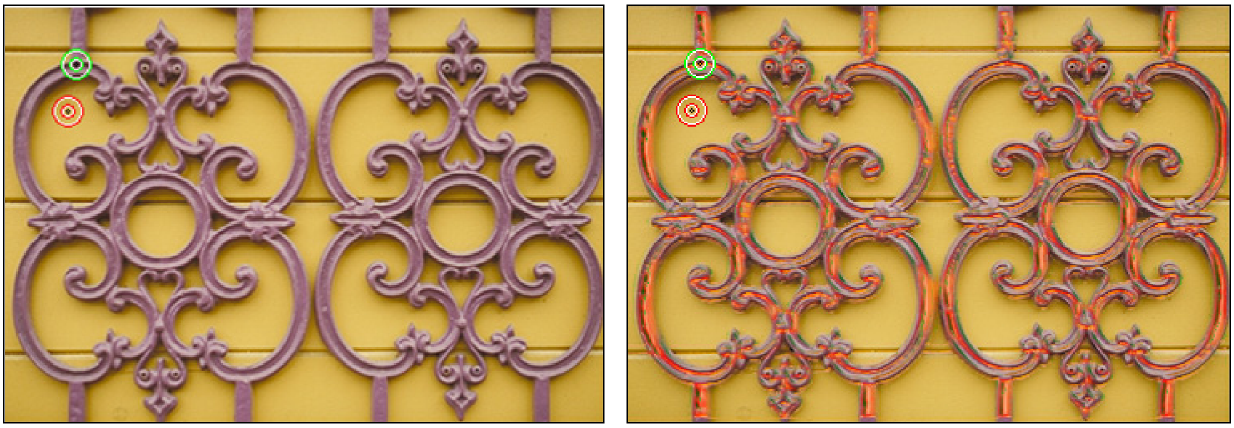


Figure 48: Painting of rust on metal grating using multiple similarity points. The original image is to the left. Multiple colours (red, orange, green) have been applied to the right-hand image.

1.4.3 Frame Rates

The number of frames per second as a function of image size and number of similarity points used is shown in Table 2. The frame rate was calculated by averaging the number of frames over 60 seconds of operation in each case. The program was executed on an off-the-shelf 2.66 GHz PC with an Nvidia GeForce4 MX 440 graphics card. Because the same number of RGB comparisons are required for each pixel regardless of image content, the timings are consistent for all images of a given size.

	Number of Similarity Points		
Image Size	1	2	3
128×128	26.30	19.36	15.52
256×256	6.80	5.32	4.14
512×512	1.63	1.27	1.06
1024×1024	0.46	0.34	0.25

Table 2: Frames per second as a function of image size and number of similarity points used (diameter of 9).

1.5 Limitations of the SSD Distance Measure

Although Markov Random Fields are a good, broadly applicable model of textures, the RGB sum of squared differences distance metric computed between local neighbourhoods does suffer from the following limitations:

1. As it is summing contributions from a neighbourhood of pixels, there is an inherent blurring tendency [Ashikhmin01] which is not always desirable.
2. The RGB colour space itself is perceptually non-uniform [Jackson94].
3. The speed of execution for large textures is poor, especially when using multiple similarity points (see Table 2).

We will address each of these problems in the sections that follow. To overcome the inherent blurring tendency we will provide the user with a means of adjusting the tool's sharpness by incorporating wavelet responses into our distance measure. The issue of perceptual non-uniformity in RGB colour space will be addressed by transforming the local neighbourhoods into a perceptually uniform colour space prior to the distance calculations being performed. Finally we will present two methods of improving the speed of execution for the replicated painting operations.

2. Improving Efficiency

Markov Random Fields have been shown to be a good model for a broad range of textures. A drawback of Markov Random Fields, though, is that they are computationally expensive. Multi-scale processing using image pyramids and the use of principal components analysis makes Random Field models of textures more tractable.

2.1 Multi-Scaled Distance Measure

To avoid sluggish response times with large textures, we will amend the neighbourhood to be multi-scaled. In order to do so we will need to construct multiple levels of the original texture by constructing a Gaussian image pyramid from the original texture [Ashikhmin01; Hertzmann01; Wei00]. An image pyramid is a multi-resolution representation of an image which stores the image responses from a bank of filters at different scales. There exist a variety of filters that can be used for constructing different types of pyramids. For the construction of the Gaussian pyramid, a low-pass Gaussian filter is used to decompose the image into a set of band-pass images. The Gaussian pyramid essentially produces a recursive reduction of image resolution with respect to the original image.

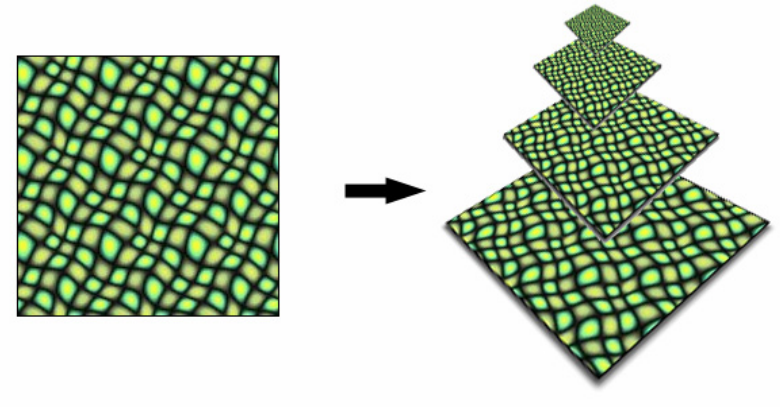


Figure 49: Multi-scale Gaussian pyramid.

In a Gaussian pyramid, each level is produced by smoothing the level below using a symmetric Gaussian kernel which reduces the bandwidth by one octave. Level 0 of the pyramid is the original texture itself and level *n*+1 is a low pass filtered version of level *n*. After this smoothing, the highest frequencies have been removed and the image contains redundant pixels. The filtered version of level *n* is then re-sampled to obtain level *n*+1 with half the width and half the height of level *n*. An example construction of a Gaussian pyramid is shown in Figure 49.

The following pseudo-code procedure accomplishes the construction of the Gaussian pyramid *G* from input image *I*(*x*,*y*):

```
Set level 0 of the Gaussian pyramid G to the original image I
for each level l from to 1 to max(levels)
    set imageTemp to the convolution of $G_{l-1}$ with low pass filter $w(m,n)$
    down-sample imageTemp by factor of 2 in each dimension
    set $G_l$ to imageTemp
end for
```

More formally, we compute each higher level G_{l+1} of from level G_l as:

$$G_{l+1}(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) \times G_l(2i+m, 2j+n)$$

where $w(m,n)$ is the separable Gaussian low-pass weighting kernel defined as:

$$w(m,n) = \hat{w}(m)\hat{w}(n)$$

and where $\hat{w}(m)$ is symmetric and defined as:

$$\hat{w}(m) = \left(\left(\frac{1}{4} - \frac{a}{2}\right), \frac{1}{4}, a, \frac{1}{4}, \left(\frac{1}{4} - \frac{a}{2}\right)\right)$$

This 5-tap filter mask is designed such that the centre pixel has more weight than its neighbours and the remaining terms are chosen so that their sum is equal to 1. This convolution weighting is symmetric and has equal contribution in that all nodes at a given level contribute the same total weight to the nodes at the next higher level. The value of a is chosen to be in the range of 0.3 to 0.6 and determines its standard deviation. A value of 0.3 is broader than Gaussian, a value of 0.5 is triangular, and a value of 0.6 is trimodal. The value of 0.4 is normally used to create a Gaussian-like weight function. The size of the weighting filter is not critical [Burt83]. We have selected the 5-tap pattern because it provides adequate filtering at low cost. Proper edge handling in the convolution operations is also important. Although there is no perfect solution to the problem, by reflecting at the border (see Figure 50) we usually avoid spurious responses.

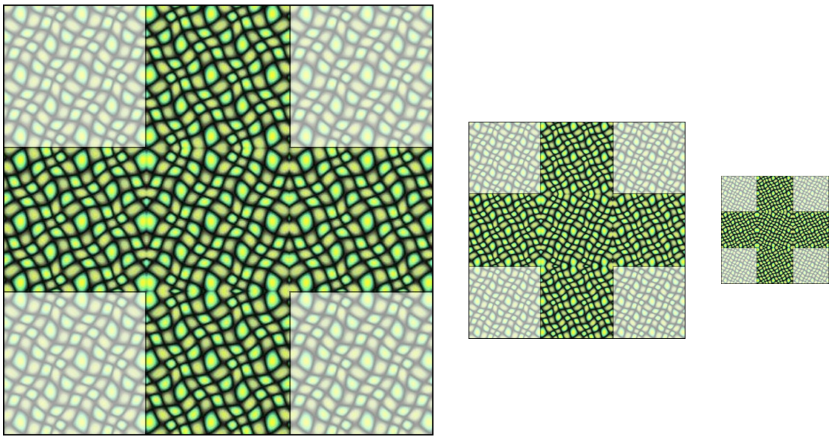


Figure 50: Reflecting at borders. A horizontal border requires a vertical reflection; a vertical border requires a horizontal reflection; and, a corner requires both a vertical and a horizontal reflection.

Once we have the Gaussian pyramid in hand, we can compute our distance metrics over multi-resolution neighbourhoods. We will achieve better interactivity under the assumption that distant pixels are less important to the similarity calculation and can therefore be approximated more and more coarsely. As is illustrated in Figure 51, the neighbourhood of the selection point is now a concatenation of multi-resolution neighbourhoods.

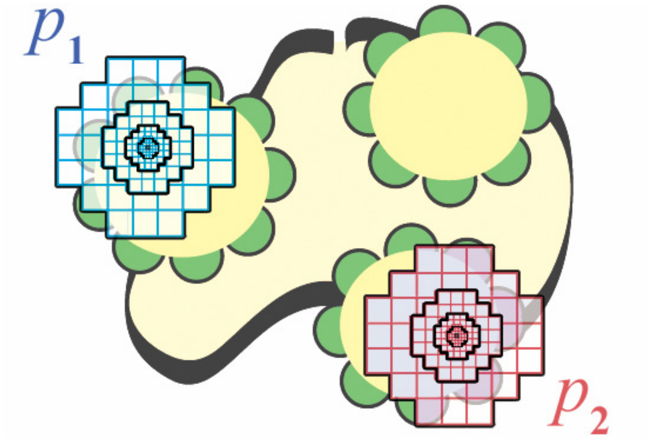


Figure 51: Multi-resolution neighbourhood comparisons.

Our distance measure is now computed as the sum of squared differences between each corresponding pixel in circular multi-resolution neighbourhoods surrounding p_1 and p_2. Using the Gaussian pyramid neighbourhoods, the formulation of the distance measure now becomes:

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

where:

$$S = \left\{ (i,j) \in \mathrm{N}^2 \quad \middle| \quad 0 \le \sqrt{i^2 + j^2} \le d/2 \right\},$$

G_l is level l of the Gaussian pyramid,

S is the circular neighbourhood,

L is the number of Gaussian levels used,
M is a 2D Gaussian weighting function and
d is the neighbourhood diameter.

Using two neighbourhood levels each with a diameter $d = 5$, yields two circular neighbourhoods of 13 pixels each. This requires the summing of 26×3=78 squared RGB differences, a number in the range of 0 to $256^2 \times 3 \times 23 \approx 4.5$M. These parameters allow the system to perform replicated painting operations at the frame rates shown in Table 3, running on an off-the-shelf 2.66 GHz PC with an Nvidia GeForce4 MX 440 graphics card. Furthermore, since the same number of RGB comparisons are required for each pixel regardless of image content, the timings are consistent for all images of a given size.

If we compare the timing results of Table 2 with those of Table 3 we see that by incorporating higher-level neighbourhoods into the similarity measure we take into account a wider area at a lower cost, while the lower level neighbourhoods retain priority for nearer pixels. The use of multi-resolution neighbourhoods almost doubles the rates of interaction; yet, at large image sizes the frame rate is still inadequate. This effect is exacerbated with the use of multi-point Boolean similarity expressions, requiring us to take additional measures to ensure interactivity.

	Number of Similarity Points		
Image Size	1	2	3
128×128	44.88	34.55	29.88
256×256	11.50	9.09	7.46
512×512	2.88	2.19	1.83
1024×1024	0.74	0.55	0.43

Table 3: Multi-resolution neighbourhoods: frames per second as a function of image size and number of similarity points used. The frame rate was calculated by averaging the frame times over 60 seconds of operation in each case.

2.2 PCA Compression of Neighbourhoods

When computing the Gaussian pyramid distance measure between two pixels, each pixel is characterized as a vector composed of a concatenation of all RGB values in each of the levels. When working with multiple similarity points, the system must compute multiple distance values for each pixel. Computationally, the vector of RGB values is augmented to include the weighted RGB neighbourhood values for all positive and negative selection points which put a strain on the response time of the system. To maintain interactivity, we employ principal components analysis (PCA) to reduce the dimensionality of the concatenated neighbourhood vectors [Jollife86].

Principal component analysis can be used to increase the efficiency of computing the distance metric without significant loss of fidelity. PCA is a mathematical method for determining the linear transformation of a sample of points in *N*-dimensional space so that the variance of the sample data is exhibited most clearly along the first few coordinate axes.

We begin with by arranging our vector population, so that each neighborhood is as follows :

$$N_i = (R_1, G_1, B_1, R_2, \ldots, G_n, B_n)^T \in R^{3n}$$

where n is the number of pre-weighted pixels in each neighbourhood. The mean $\overline{N}$ is then denoted as:

$$\overline{N} = \sum_{i=1}^{m} \frac{N_i}{m}$$

where m is the number of neighbourhoods in the image. With the average neighbourhood vector in hand we compute the covariance matrix C_N from the neighbourhood differences $\Delta N_i = N_i - \overline{N}$. To do this we define a $n \times m$ matrix P_N that concatenates all the ΔN_i; essentially, each of the vertical difference vectors for the m neighbourhoods are lined up as abutting columns:

$$P_N = \{\Delta N_1, \Delta N_2, \ldots, \Delta N_m\}$$

Now, to compute the covariance matrix C_N we simply multiply P_N by its own transpose:

$$C_N = P_N P_N^T$$

Finally, to obtain the orthogonal basis vectors for the neighbourhood data we compute the eigenvectors of C_N. Each eigenvector $i = 1\ldots(m-1)$ we will denote s_i. As it has been described, this is quite an expensive operation. In practice, an alternate method is used to derive only the $m-1$ most significant eigenvectors but for clarity of argument the above holds.

We can now represent any one of our original neighbourhoods, N_x, as points in the space defined by s_i. This means that any neighbourhood, N_x, can be represented as a linear combination of s_i:

$$N_x = \overline{N} + \sum_{i=1}^{m} \alpha_i s_i$$

A neighborhood, N_x, is transformed from R^{3n} to R^{m-1} by computing the eigenvector coefficients, α_i. To compute the α_x vector for a given neighborhood, N_x, we must perform a coordinate transformation, pre-multiplying the neighborhood difference matrix, ΔN_x, by the transpose of the matrix composed of columns of the neighborhood eigenvectors s_i:

$$\alpha = [s_1, s_2, \ldots s_{m-1}]^T \left(N_x - \overline{N}\right)$$

The fact that the neighbourhoods are now concentrated in a linear subspace has compressed the data without much loss of information. We can generally gain an order of magnitude reduction in the size of the vectors without an appreciable reduction of quality. By retaining the eigenvectors having the largest eigenvalues we are able to keep over 99% of the variance.

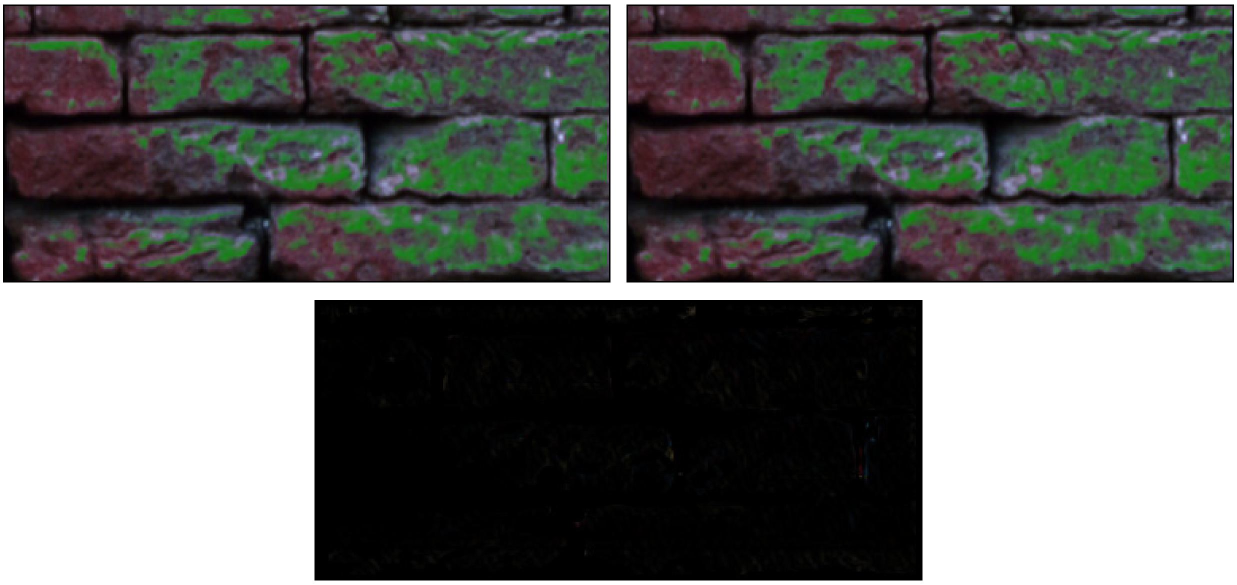


Figure 52: Residual difference (bottom) between replicated painting with (top-right) and without (top-left) the use of principal component analysis. The neighbourhood vector has been reduced by an order of magnitude.

For example, the top row of images in Figure 52 show the replicated painting of a solid green colour over areas of a brick texture. The left-hand image was created without the use of PCA, requiring the full 26×3=78 squared RGB comparisons per pixel. For the right-hand image only the first 7 principal components were used in the similarity computations. Since the results with and without the use of PCA are very similar, it is easier to visualize the marginal loss of fidelity if we look at the difference image between the two resulting images. As we can see in bottom image of Figure 52, the residual difference is nominal, appearing almost black.

This is confirmed by examining the rapidly diminishing variances of the first 10 eigenvectors, s_i, for 5 randomly chosen textures, I_x, each of size 256×256:

	Texture Images				
Eigenvectors	I_1	I_2	I_3	I_4	I_5
s_1	5.3926	5.0182	2.2607	3.6550	2.2326
s_2	0.5081	0.8852	0.0919	0.3388	0.1492
s_3	0.2575	0.2891	0.0743	0.3388	0.0475
s_4	0.1660	0.1542	0.0179	0.0415	0.0218
s_5	0.0682	0.1350	0.0110	0.0186	0.0170
s_6	0.0565	0.0471	0.0090	0.0161	0.0151
s_7	0.0456	0.0454	0.0075	0.0127	0.0142
s_8	0.0200	0.0316	0.0062	0.0087	0.0096
s_9	0.0163	0.0297	0.0040	0.0072	0.0081
s_{10}	0.0138	0.0202	0.0027	0.0049	0.0075

Table 4: The first ten eigenvectors, s_i, and the corresponding variances for 5 randomly chosen textures, I_x, each of size 256×256.

If we wish to guarantee quality by keeping at least 99% of the original variance, we need to vary the number of principal components used for each image. This is because the number of principal components needed in order to reach 99% of variance can vary from image to image. For the examples, $I_1 \ldots I_5$, of Table 4 the number of principal components needed in order to reach 99% of variance are 14, 20, 9, 10 and 22, respectively. The average number of principal components needed to reach 99% of variance, computed from 100 randomly chosen textures, are shown in Table 5 as a function of image size; also shown are the standard deviations, maximums and minimums of the required number of principal components as a function of image size.

	Required Number of Principal Components			
Image Size	Mean	Std Dev.	Min.	Max.
128×128	10.40	4.43	5	20
256×256	14.00	4.85	9	22
512×512	19.15	5.33	13	28
1024×1024	25.87	6.35	18	35

Table 5: The average number of principal components needed to reach 99% of variance. Also shown are the standard deviations, maximums and minimums of the required number of principal components as a function of image size.

The frame rates obtained using the average number of principal components required for each image size are shown in Table 6 (running on an off-the-shelf 2.66 GHz PC with an Nvidia

GeForce4 MX 440 graphics card). If we compare the timing results of Table 3 with those of Table 6 we see that, on average, we are able to more than double our efficiency even for 1024×1024 images which require the most principal components. For smaller images, which require fewer principal components, the efficiency gains are even greater.

	Number of Similarity Points		
Image Size	1	2	3
128×128	282.31	224.57	184.94
256×256	56.04	44.86	34.44
512×512	10.62	7.93	6.68
1024×1024	2.05	1.54	1.18

Table 6: PCA compressed multi-resolution neighbourhoods: average number of frames per second as a function of image size and number of similarity points used.

It is important to note that the variation in the number of frames per second is only between images and is not dependant on the particular positions of the similarity point within an image. This implies that the user will experience a consistent frame rate when continuously operating on a given image.

The pre-computation time needed to determine the eigenvectors of images as a function of image size is shown in Table 7. These PCA calculations only need to be performed once per image and do not appear to vary significantly between images of the same dimensions.

Image Size	**Time (s)**
128×128	2.79
256×256	7.25
512×512	25.88
1024×1024	63.14

Table 7: PCA computation times as a function of image size.

3. Improving the Similarity Measure

Having improved the efficiency of the distance calculations, there still remain two deficiencies of our distance metric under the Markov model assumption:

1. There is an inherent blurring tendency.
2. The RGB colour space itself is perceptually non-uniform.

We will now address the perceptual non-uniformity in RGB colour space by transforming the local neighbourhoods into the perceptually uniform CIE LAB colour space prior to the distance calculations being performed. To overcome the inherent blurring tendency we will provide the user with a means of adjusting the tool's sharpness by incorporating steerable wavelet responses into our distance measure.

3.1 Perceptually Uniform Colour Space

The tristimulus RGB colour space (see Figure 53) is the most commonly used in computer graphics, primarily because it is directly supported by most colour monitors. However, Euclidean distances in RGB space do not correspond to colour differences as perceived by human beings [Jackson94]. As we will now describe, RGB space is a subset of CIE XYZ space which itself is not perceptually uniform.

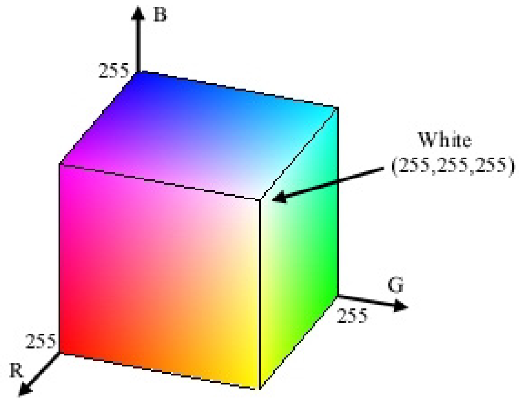


Figure 53: RGB colour cube with corners of black (hidden), the three primaries (red, green, and blue), the three secondaries (cyan, magenta, and yellow), and white.

3.1.1 CIE XYZ Colour Space

Although the human visual system has only three types of colour receptor, there does not exist a set of three primary colours which can be positively combined to produce all colours of the visible spectrum. To overcome this constraint, the International Commission on Illumination (CIE) in 1931 defined three super-saturated primaries (X, Y, and Z) such that all visible colours can be specified with positive combinations of these primaries. Like RGB space, the CIE XYZ space allows colours to be expressed as a mixture of the three tristimulus values X, Y, and Z, with the primary Y intentionally defined to correspond to the luminance quality of a colour as perceived by a human observer.

We can understand the non-uniform aspect of the XYZ color space when it is first converted into Yxy space which expresses the XYZ values in terms of x and y chromaticity coordinates. The following formulas are used to convert XYZ into Yxy:

$$Y = Y$$
$$x = X / (X + Y + Z)$$
$$y = Y / (X + Y + Z)$$

Note that the Z tristimulus value is incorporated into the new coordinates, and does not appear by itself. Since Y still correlates to the lightness of a colour, the other aspects of the color are found in the chromaticity coordinates x and y. This allows colour variation in Yxy space to be plotted on a two-dimensional diagram. Figure 54 (left) shows the layout of colours in the x and y plane of Yxy space.

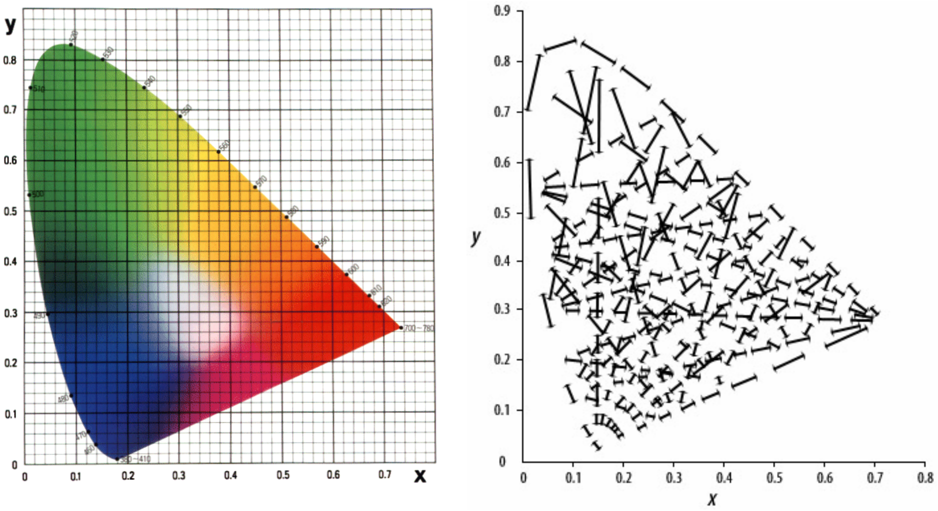


Figure 54: Chromaticity diagram (left) and perceptually equivalent distances shown as line segments (right).

We can see from the left-hand diagram that the amount of area covered by what we would loosely describe as green is substantially larger than that covered by purple, blue or any other colour. More precisely, each line in the right-hand diagram shown in Figure 54 represents a colour difference of equal proportion. The distance between the end points of each line segment are perceptually the same according to the 1931 CIE standard observer. As you can see, the lines vary in length, sometimes greatly, depending on what part of the diagram they are in. This disparity in line length indicates the amount of perceptual distortion between areas of the chromaticity diagram.

RGB space is a subset of XYZ subjected to the following affine transformation [Wyszecki82]:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 2.36461 & -0.89654 & -0.46807 \\ -0.51517 & 1.42641 & 0.08876 \\ 0.00520 & -0.01441 & 1.00920 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

And, inversely:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.49000 & 0.31000 & 0.20000 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00000 & 0.01000 & 0.99000 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

In Figure 55, RGB space is shown as a subset of XYZ space, contained within the black triangle. Since the RGB is simply an affine transformation of XYZ, RGB is therefore also perceptually non-uniform.

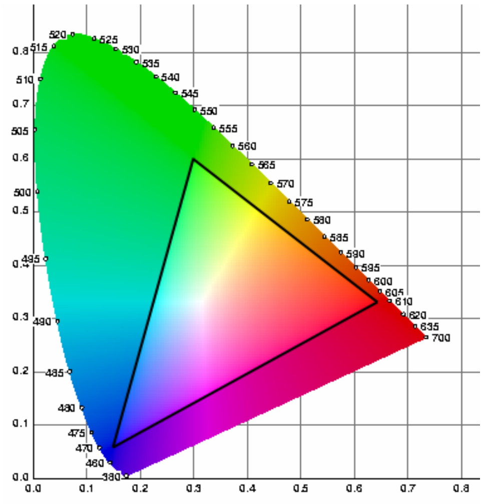


Figure 55: RGB space (in black triangle) shown as a subset of XYZ space. Note that colours outside of the RGB triangle cannot be truly represented in this figure as only colour within the triangle can be reproduced on a colour monitor.

3.1.2 Perceptually Correct Colour Distances

To compute perceptually correct colour distances we need to first transform the image from RGB space into a perceptually uniform colour space such as CIE LAB. CIE LAB colour space is a perceptually uniform derivation of the standard CIE XYZ space, meaning that colours that are equally distant in the colour space are equally distant perceptually.

CIE LAB is an opponent colour system adopted by International Commission on Illumination (CIE) in 1976, based on the earlier (1942) system of Richard Hunter called L, a, b. CIE LAB is a colour space in which colours are located within a three-dimensional rectangular coordinate system. CIE LAB space defines colours relative to a reference white point which is represented in terms of XYZ space, and is usually based on the whitest light that can be generated by a given device. The three channels are a lightness dimension (L), and two opponent colour dimensions redness/greenness (a) and yellowness/blueness (b).

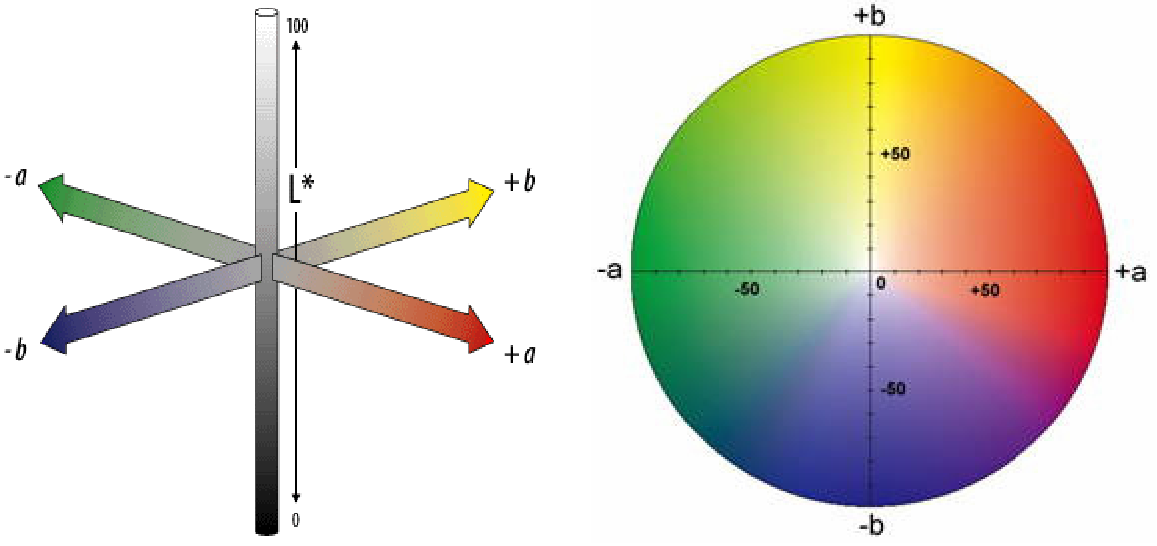


Figure 56: CIE LAB colour space axes (left) and the circular CIE LAB chromaticity diagram (right).

The left-hand diagram in Figure 55 shows a representation of LAB space with its chromaticity diagram shown to the right. The central vertical axis represents lightness (signified as L*) whose values run from 0 (black) to 100 (white). The colour axes are based on the fact that a colour cannot be both red and green, or both blue and yellow, as these colours oppose each other; on these axes the values run from positive to negative. For both axes, zero is neutral grey. This formulation of colour oppositions correlates with discoveries in the mid-1960s that somewhere between the optical nerve and the brain, retinal colour stimuli are translated into distinctions between light and dark, red and green, and blue and yellow.

Conversion from CIE XYZ to CIE LAB involves a nonlinear transformation performed as follows:

$$
\begin{aligned}
&\text{if } (Y / Y_n > 0.008856)\\
&\quad L = 116.0 \times 1/3 \times (Y / Y_n) - 16.0\\
&\text{else}\\
&\quad L = 903.3 \times (Y / Y_n)\\
&a = 500.0 \times 1/3 \times ((X / X_n) - (Y / Y_n))\\
&b = 500.0 \times 1/3 \times ((Y / Y_n) - (Z / Z_n))
\end{aligned}
$$

where X_n, Y_n and Z_n are the XYZ values for the reference white point.

Once we have converted the image into LAB space, we can compute perceptually valid colour distances for our neighbourhood metric. In addition, we can replicate lightening and darkening operations based on self-similarity. For those pixels whose neighbourhoods are sufficiently similar to the user-selected pixel, we can increase or decrease the luminance in isolation without affecting the colour. In Figure 57 we see an example where we have deepened a texture by selectively lightening and darkening areas to give the impression of specular highlights and shadow.

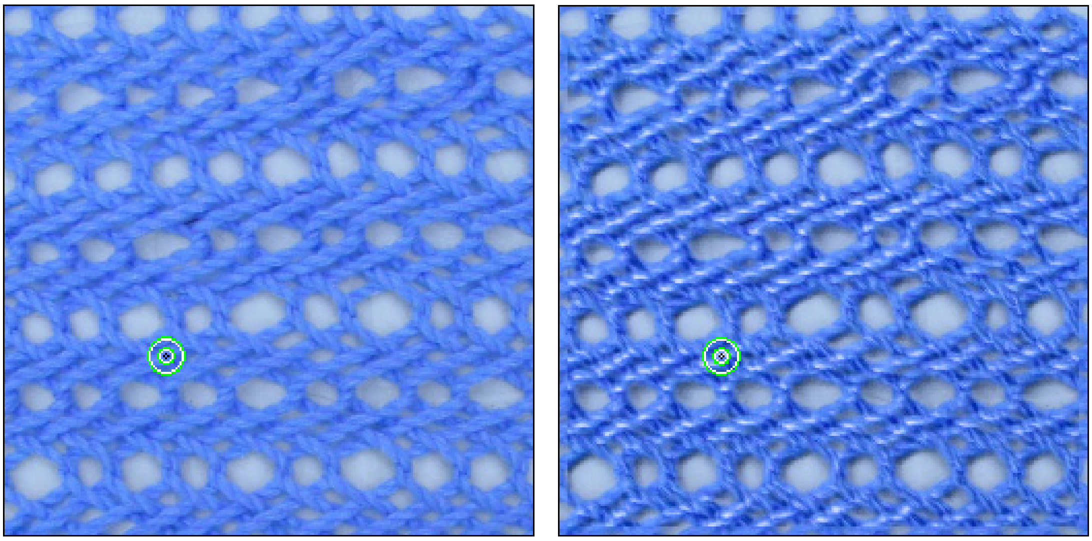


Figure 57: Deepening a texture with replicated highlights and shadows.

3.2 Wavelet Based Similarity

There remains the issue of the inherent smoothing tendency of Gaussian pyramid neighbourhood distance calculations. As described, the system does not perform as well for textures that contain sharp features. To address the limitations of the original similarity measure, we include multi-scale responses from a steerable wavelet pyramid constructed from the texture being edited. Moreover, we give the user finer control by providing a 'sharpness' slider that specifies what proportion of neighbourhood versus wavelet responses are to be used in the similarity calculation.

3.2.1 Wavelets

When analyzing a signal, such as a 2D image, a key issue is the choice of representation. One can operate directly on the original 2D array of discrete pixel values, or alternatively, the signal can be transformed into another domain such as the frequency domain via the Fourier transform. Such transformations represent a signal, $f(x)$, as a weighted summation of basis functions, $g_i(x)$:

$$f(x) = \sum_i y_i \, g_i(x)$$

where y_i are the basis coefficients. These coefficients are obtained by projecting the signal, $f(x)$, onto the set of projection functions, $h_i(x)$:

$$y_i = \int h_i(x) f(x) \, dx$$

The Fourier transform, having sine and cosine waves of increasing frequencies as its set of basis functions, has enjoyed widespread use as a tool of analysis. Yet for 2D texture signals a more appropriate representation has emerged that better characterizes the texture details at multiple scales within a texture image. This alternate representation is the wavelet transform, which has become a popular mathematical tool for analyzing *n*-dimensional objects at multiple scales [Stollnitz96].

The defining characteristic of a wavelet transform is that the basis functions are translations and dilations of a common filter kernel. When dealing with images, wavelets allow us to decompose the

2D signal into the overall pattern plus levels of finer detail. A wavelet tree is a multi-scale decomposition of an image where each level stores a projection of the image with the wavelet basis function of a certain resolution, at all translations of the basis functions. The basis set for wavelets is (usually) orthonormal, meaning that the basis functions are linearly independent and self-inverting. The term *linearly independent* refers to the fact that there is no linear combination of the basis functions that yields a zero vector. The property of *self-inversion* simply implies that $h_i(x) = g_i(x)$. A further attribute that most wavelets possess is critical sampling which implies that the number of wavelet coefficients is equal to the input signal's sample rate. By capitalizing on these properties, wavelets have been successfully used in applications such as image compression, texture synthesis and level-of-detail control to name a few.

3.2.2 Steerable Pyramids

Like the Gaussian pyramid, the steerable wavelet transform decomposes an image into several spatial frequency bands [Simoncelli95]. Furthermore, it divides each frequency band into a set of orientation bands which respond strongly to rotationally varying edges. The basis functions for steerable pyramids are directional derivative operators that come in different sizes and orientations. The pyramid is termed *steerable* because, by using only a small number of filters corresponding to a few directions, the output of the filter in any direction can easily be computed as a weighted sum of pre-calculated responses from the basis filters. The steerable pyramid thereby provides us with a robust and computationally efficient scheme to extract a set of orientation selective band pass responses of the input textures.

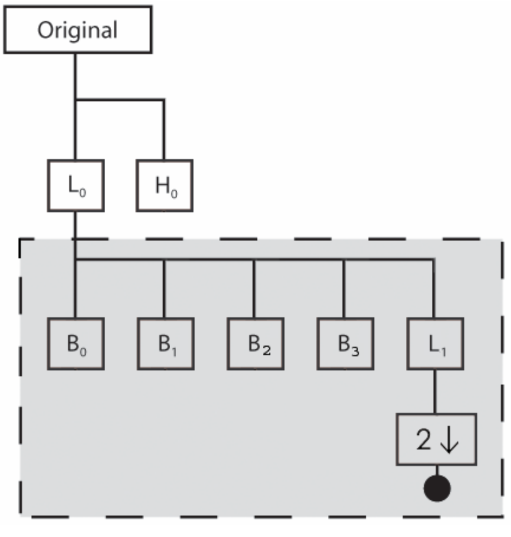


Figure 58: Steerable wavelet decomposition.

To decompose a texture image into a steerable pyramid, the image is subjected to a recursive hierarchy of filtering operations. A block diagram for the steerable pyramid decomposition is shown in Figure 58. In the first stage, the texture signal is split into low and high-pass subbands. This is achieved by convolving the image with a low-pass, L_0, and a high-pass, H_0, filter respectively. Both of these filters have radially symmetric frequency responses with the high-pass band corresponding to the four corners of the spatial frequency domain. Next, the low-pass subband is further decomposed into a set of four oriented subbands and a low(er)-pass subband.

The orientation subbands are computed using four orientation-selective filters, B_i, at orientations of 0, 45, 90, and 135 degrees. The low-pass subband is convolved with a second low pass filter, L_1, and is subsequently sub-sampled by a factor of two in both dimensions. The recursive diagram of the pyramid construction continues by inserting a copy of the shaded portion of the diagram at the location of the black circle (i.e. the low-pass branch). That is, each successive level of the pyramid is constructed from the previous level's low-pass band by convolving it with a bank of orientation filters and a low-pass filter.

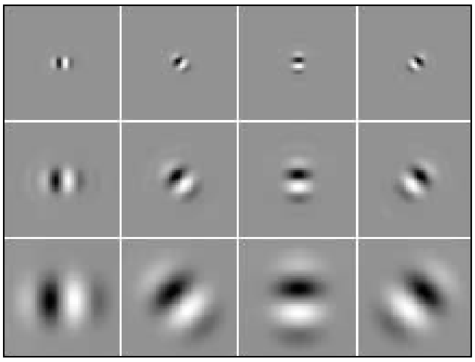


Figure 59: A set of steerable basis functions at three levels with four orientations per level [Heeger95]. Medium grey is equal to 0, black is negative and white is positive.

Figure 59 shows the set of basis functions at three levels of detail and four orientations. The manner in which the steerable wavelet filters are constructed is by no means trivial. The filters themselves are generated with a recursive design procedure wherein the system iteratively converges upon the steerable filters. This is achieved by minimizing a weighted sum of errors which are deviations from a number of design constraints. The design constraints include a unity system response amplitude, a recursion relationship over the low-pass branch of the decomposition and steerability constraints on the orientation-selective filters, B_i. Because of this, the filters do not lend themselves to a brief analytic specification. However, a well established implementation has been used which is available at ftp://ftp.cis.upenn.edu/pub/eero/steerpyr.tar.gz, and to ensure continued reproducibility, the exact filter values are included in Appendix A. For further details on the filter design procedure, the reader is directed to [Karasaridis96].

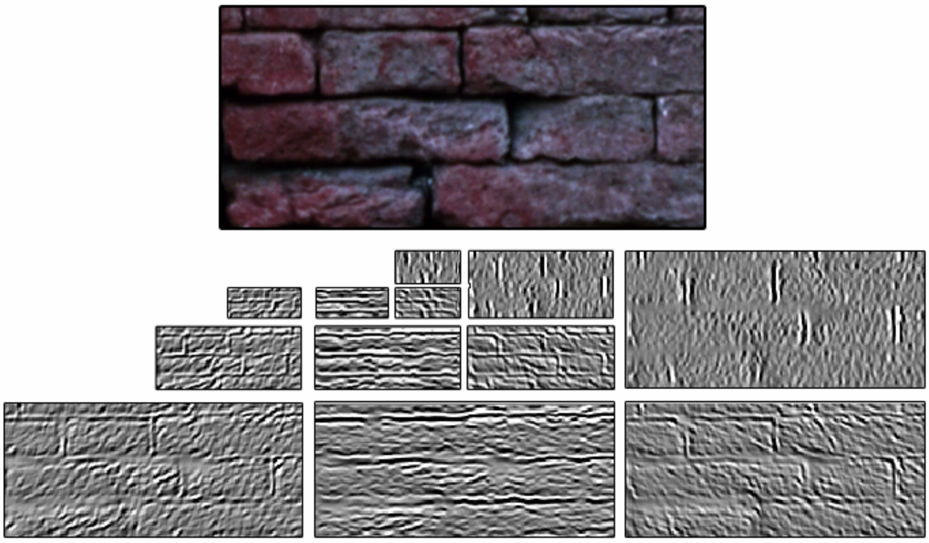


Figure 60: The steerable wavelet filter responses of a brick texture.

Figure 60 shows an example of the wavelet transform applied to a brick texture. At the top of the figure is the original image and along the bottom are the corresponding steerable pyramid subband images for the brick texture. Shown are three levels at four orientations per level. It is clear from the subband images that the steerable wavelet transform is responding to the edges in the image, particularly at the sharp discontinuities found between each brick.

3.2.3 Justification for the Inclusion of Steerable Wavelet Responses

A key reason for including steerable pyramid responses stems from current theories of human texture discrimination. These theories are founded on the fact that two textures are often difficult to discriminate when they produce a similar distribution of responses from a bank of orientation and spatial-frequency selective linear filters [Heeger95]. Psychophysical and physiological experiments suggest that image information is represented in the visual cortex by similar orientation and spatial-frequency selective filters. The steerable pyramid captures the oriented structure of images in a comparable fashion to the way in which this information is represented in the human visual system.

Moreover, the recent success of using wavelets as an intermediate representation for texture synthesis lends further credibility to their use as tools for texture analysis. As discussed in Section 3.2, the synthesis algorithm of Bar-Joseph *et al.* uses wavelet tree decompositions to extract the salient features of input textures. These features are subsequently used to construct new wavelet trees that, when inverted, appear to come from the same stochastic source [Bar-Joseph01].

But, beyond questions of human perception, this particular wavelet transform has a number of technical strengths with respect to aliasing, translational invariance and rotational invariance that set it apart from other types of wavelets [Simoncelli95].

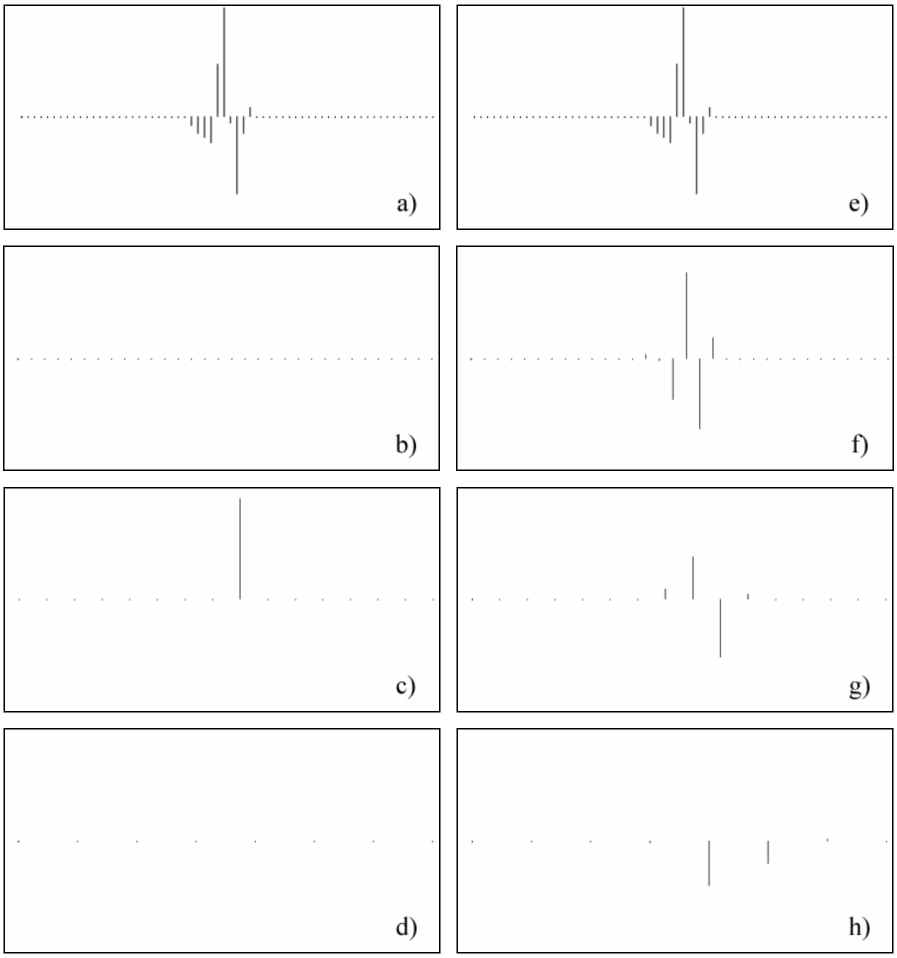


Figure 61: Effects of minor translation on Daubechies wavelet decomposition [Simoncelli92].

The translational invariance of steerable pyramids ensures that all information represented within a subband remains in that subband as the signal is translated. This is not generally a property of other popular wavelet transforms. We can generate a clear example of this by decomposing an input signal that is itself equal to a Daubechies wavelet basis function. Figure 61 shows two separate decompositions of 1D signals. The two input signals are shown in the top row. The first input signal (a) is equal to a Daubechies wavelet basis and the second input signal (e) is simply a slight translation of the first.

Figure 61(b-d) shows three levels of decomposition of the original input signal (a). Figure 61(f-h) shows three levels of decomposition of the translated input signal (e). Since the input signal (a) is equal to a particular translation of the wavelet basis at the second level of detail, all other responses in the wavelet decomposition are necessarily zero. This is in stark contrast to the second case (e) in which the signal has only been slightly translated to the right. As the signal no longer finds an exact positional match with the Daubechies wavelet, it causes a dramatic change in response both within the same level and across levels. This suggests that in the absence of translational invariance, our multi-scale similarity measure would exhibit different behaviour as responses migrate from level to level under spatial translation; but, by being translationally invariant, the steerable pyramid avoids this instability. Furthermore, the steerable pyramid is designed to be over-complete in order to avoid aliasing within each subband as well.

The steerability of the pyramid is another important feature of our chosen decomposition because it implies that even though we are only using a fixed number of oriented filters, all of the necessary edge information has been accounted for. Since the orientation decomposition at each level of the pyramid is steerable, the response of the filter tuned to any orientation can be obtained through a linear combination of the responses of the four basis filters computed at the same location. This implies that the pyramid representation is locally rotation-invariant [Simoncelli95].

3.2.4 Wavelet Enhanced Similarity Measure

We will now look at an augmentation to the original Gaussian pyramid measure. Complementing the Gaussian pyramid decomposition, the texture image is decomposed into a steerable pyramid which produces a multi-scale, multi-orientation linear signal decomposition of the original texture. The hierarchical neighbourhood distance measure is augmented with the wavelet responses in order to mitigate the inherent blurring tendency of the local Gaussian neighbourhood distance measure. Moreover, we give the user control over the weighting between Gaussian neighbourhood and steerable wavelet responses. In effect, this relative weighting acts as a 'sharpness' control for the replicated painting brush.

We recall that the original formulation of the distance measure between the two points, *p₁* and *p₂*, using Gaussian neighbourhoods is:

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

The wavelet enhanced measure takes into account 6 pyramids in total: three Gaussian pyramids (one per colour channel) and three steerable pyramids (one per colour channel). When our feature vector is augmented with the wavelet responses, the distance measure becomes:

$$d'_{SSD}(p_1, p_2) = \begin{array}{c} \beta\left(\sum_{l=0}^{L-1} \sum_{(i,j)\in S} 4^l \times M(2^l i, 2^l j) \times \left(G_l\left(\lfloor x_1/2^l \rfloor + i, \lfloor y_1/2^l \rfloor + j\right) - G_l\left(\lfloor x_2/2^l \rfloor + i, \lfloor y_2/2^l \rfloor + j\right)\right)\right) + \\ (1-\beta)\left(\sum_{l=0}^{L-1} \sum_{a=0}^{3} \left(W_{l,a}\left(\lfloor x_1/2^l \rfloor, \lfloor y_1/2^l \rfloor\right) - W_{l,a}\left(\lfloor x_2/2^l \rfloor, \lfloor y_2/2^l \rfloor\right)\right)^2\right) \end{array}$$

where

$\beta \in [0..1]$ is a weighting value controlled by the user's sharpness slider,

$W_{l,a}$ is orientation angle $\pi \times (a/4)$ of level *l* of the steerable pyramid, and

L is the number of wavelet and Gaussian pyramid levels used.

There are now two large terms representing the Gaussian neighbourhoods and the wavelet responses, respectively. The Gaussian term is as before. The wavelet term is computed as the sum of squared differences at multiple levels, l, and orientations, $\pi \times (a/4)$. Lastly, the weighting variable, β, determines the relative priority of the Gaussian and wavelet terms which is itself directly controlled by the user's sharpness slider.

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. This alleviates the problem of excessive smoothing that can result from relying solely upon Gaussian pyramid neighbourhoods.

We note that the remaining residual low pass band from the steerable pyramid is not utilized as it is reduced to such an extent that it contains virtually no information. The high pass residual is also not used as it is noisy and is not comprised of orientated responses. A further technical note is that because the steerable distances tend to be much smaller than the Gaussian neighbourhood distances, the overall contribution of the steerable versus neighbourhood distances are normalized before applying the relative weighting, β. This is ensures that the 'sharpness' slider behaves in a more balanced fashion from the user's perspective.

3.2.5 Comparison of Results

We will now illustrate the additional flexibility that the inclusion of wavelet responses allows. Figure 62 shows the painting of a solid green colour onto two separate textures. The original images are shown at the top. The centre images show results using only Gaussian pyramid neighbourhood responses and the images to the bottom show results using both neighbourhood and wavelet responses. With the inclusion of wavelet features, the overall response is similar but on close inspection there are significant differences in the amount of detail that the tool is sensitive to. As can be seen from the zoomed inset images, by incorporating the wavelet responses into the similarity distance measure, the self-similarity tool is able to respond to the finer edge details in the original images.

Figure 63 shows the effects of gradually altering the weighting variable, β. The left image shows the original wood shingle texture. The four remaining images were created with the weighting variable, β, set at values of ¾, ½, ¼ and 0 (from left to right). In this way, the user can dictate the extent to which these edge details influence the final outcome with the continuous adjustment of the sharpness slider. It is also important to note that using both neighbourhood and wavelet responses allows the sharpness to be controlled while still retaining the brush's directional control.

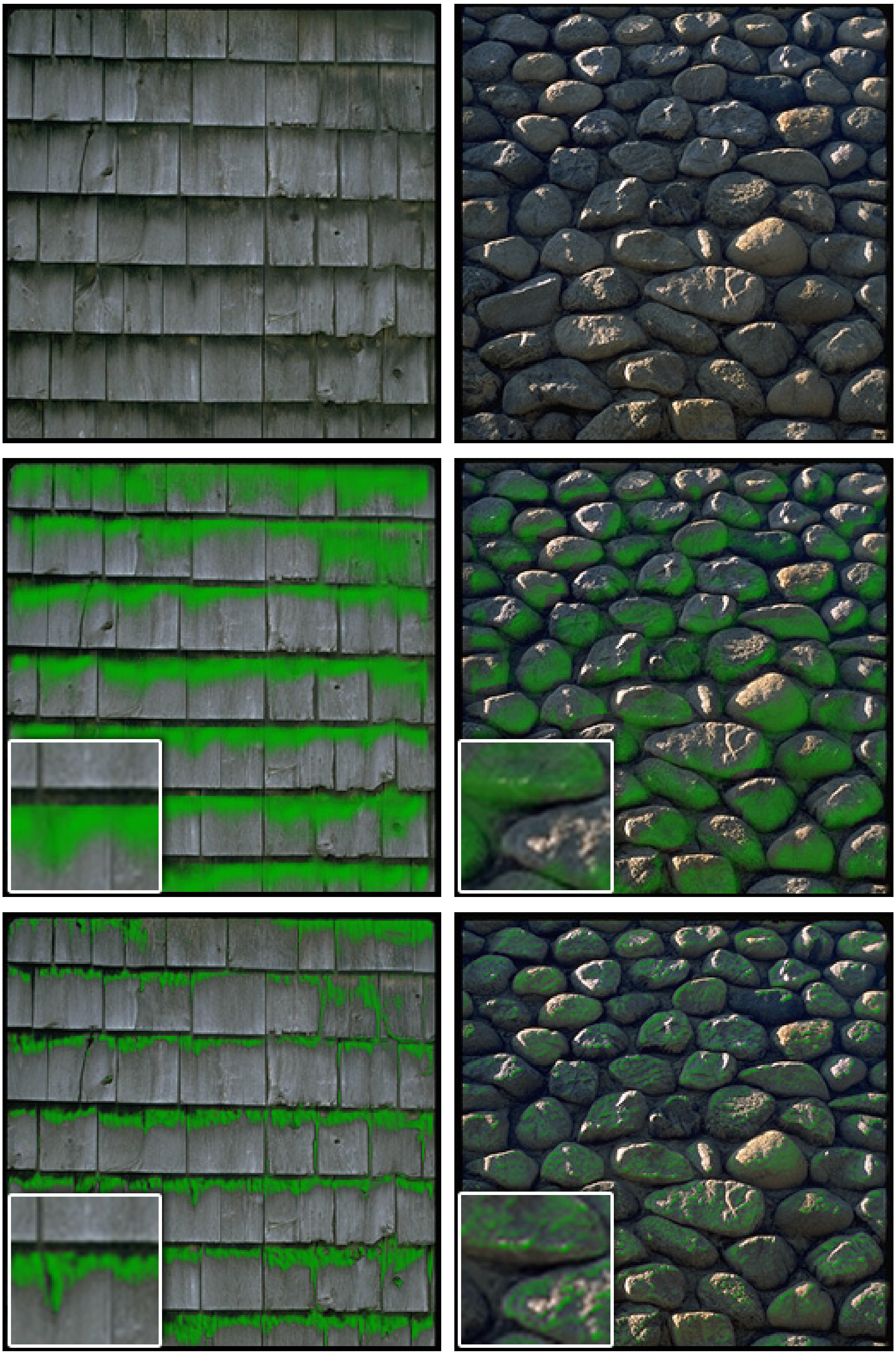


Figure 62: Painting of solid green colour onto textures. Top row: original. Middle: Painted using only Gaussian pyramid neighbourhood responses. Bottom: Both neighbourhood and wavelet responses used.

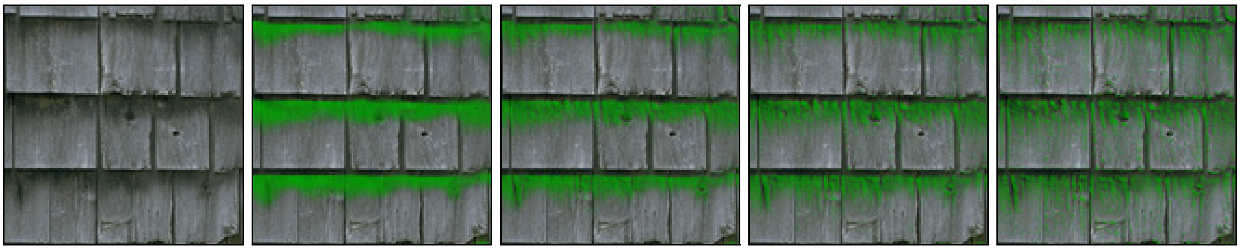


Figure 63: Gradually decreasing the weighting value, β. The left image shows the original wood shingle texture. The remaining images shows the effects of decreasing the weighting value, β, from left to right.

Further examples are shown in Figure 64 and Figure 65. Figure 64 shows the application of white paint over texture regions of a doorway to give the impression of snowfall. For this image, multiple painting operations were performed to cover the various texture regions. The top image shows the original scene. The bottom left shows the same image having been painted using only Gaussian pyramid neighbourhood responses. The image to the bottom right has been subjected to painting operations that have employed both neighbourhood and wavelet responses. In Figure 65 we revisit the example of painting sunlight (yellow paint) onto the top sides of mossy rocks. Once again, the inclusion of wavelet responses has allowed the system to respond to fine details in the moss texture.

We will see more results of the use of wavelet based similarity in the next chapter on self-similarity based texture cloning. But first, in the section that follows, we will examine the user interface to offer the reader a more concrete understanding of the system's usage.

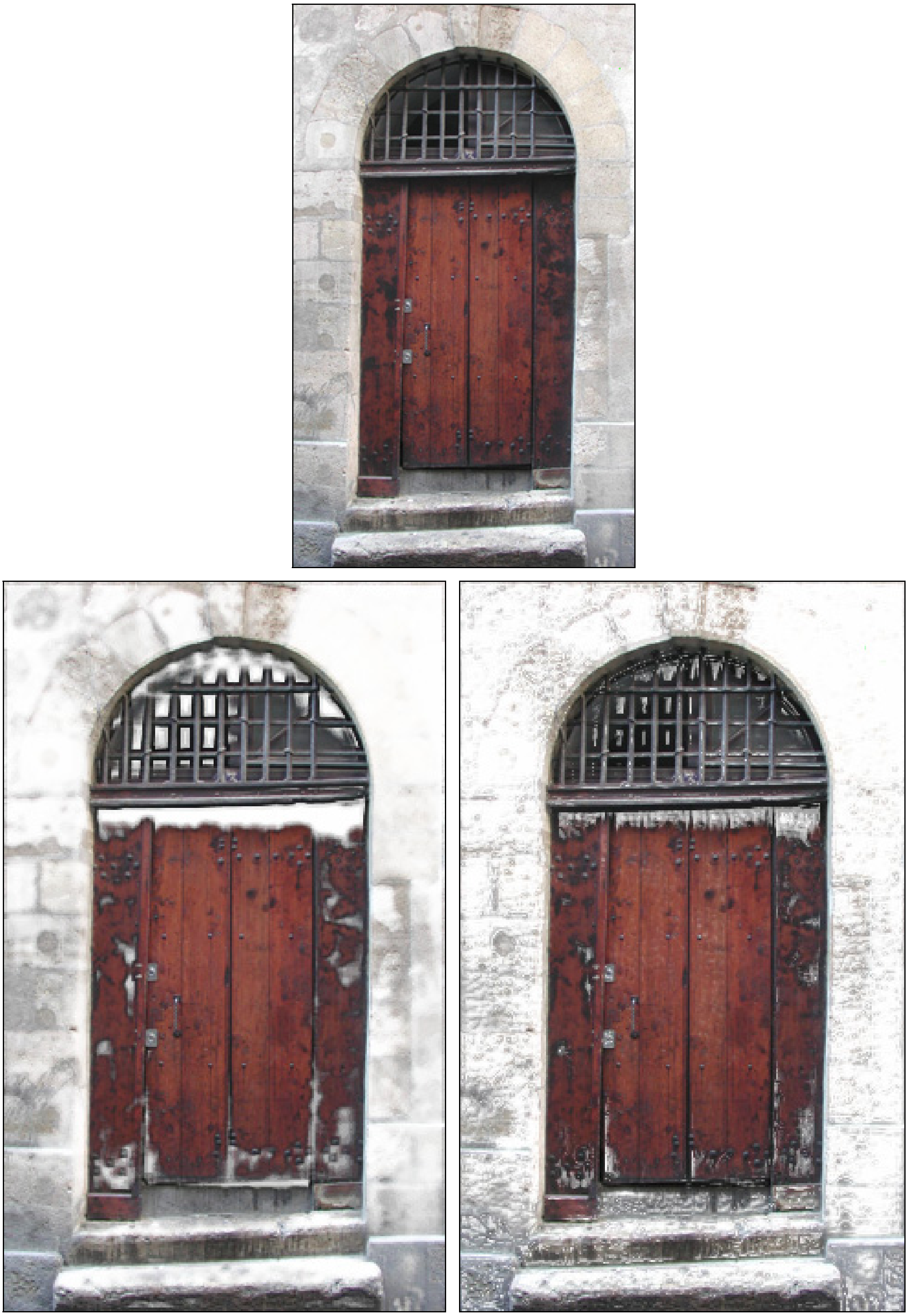


Figure 64: Multiple painting of snow (white paint) applied to a doorway. Top: Original. Left: Painted using only Gaussian pyramid neighbourhood responses. Right: Both neighbourhood and wavelet responses used.

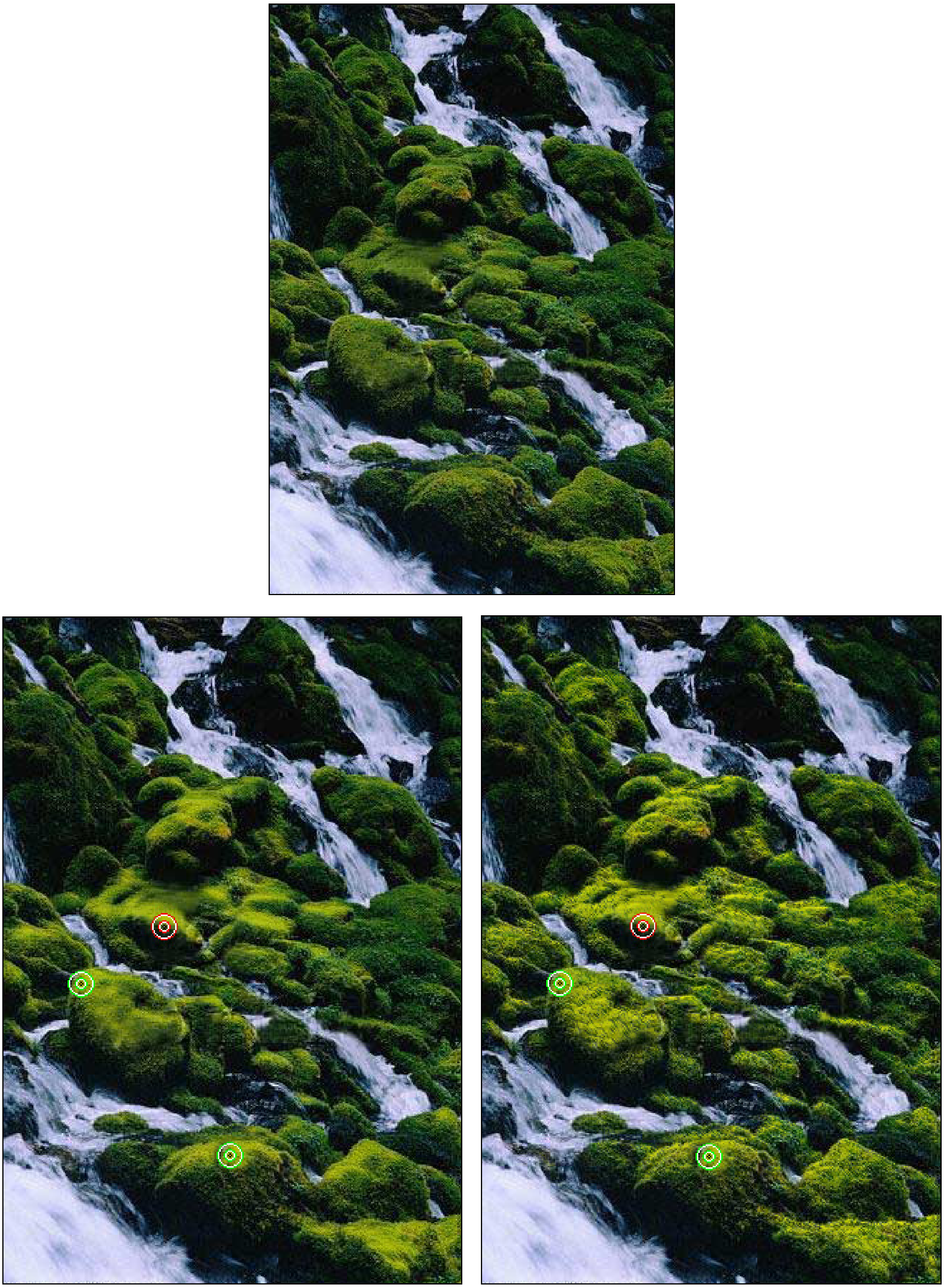
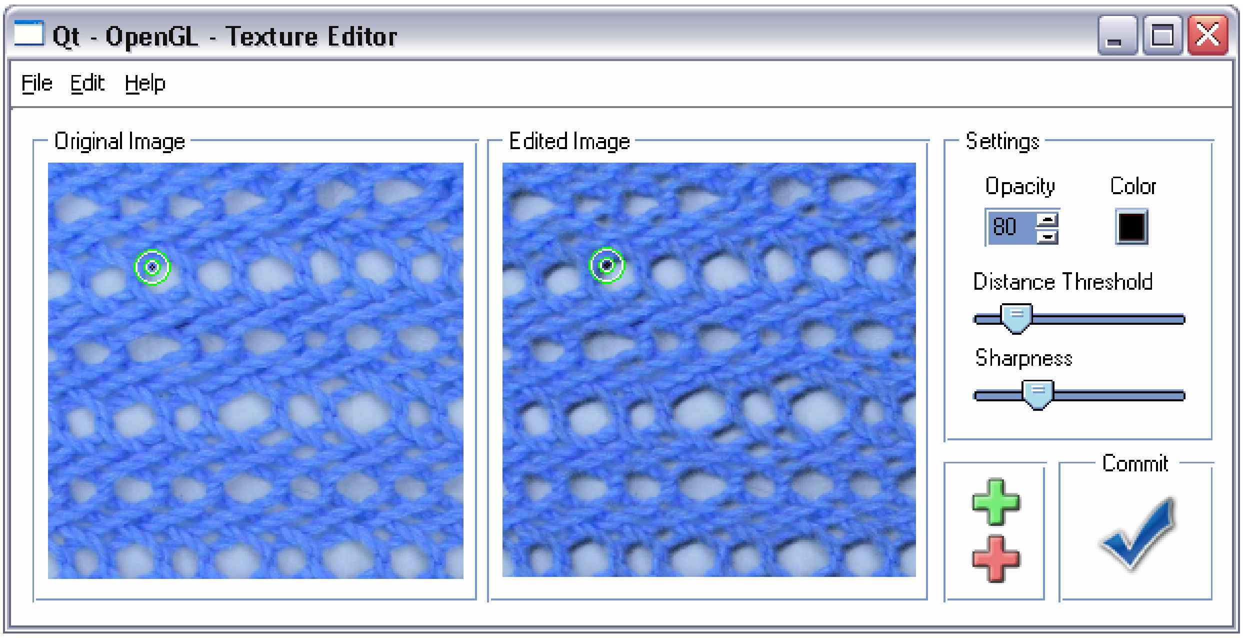


Figure 65: Painting sunlight on top side of mossy rocks. The original is shown at the top. The bottom left image uses only Gaussian neighbourhoods. The bottom right image includes the use of wavelet responses.

4. The Interface

The system's style of interaction is based on the idea that the user suggests and the software articulates. The user's decisions are replicated globally across the image, and in so doing, tedious editing operations are performed automatically. We now briefly describe the interface components that are required to make this possible.



Opacity 80

Figure 66: Replicated painting editor. The user paints at a single location in the image. The system replicates the painting operation to all similar areas within the image.

As can be seen in Figure 66, the interface is divided into five panels. The leftmost panel shows the original image which does not change unless a new image is opened. The next panel, in the middle, shows the image as it is undergoing editing operations. To the right are three additional panels. The topmost houses the various control settings that the user has to alter the behaviour of the painting tool. The left panel, directly underneath, contains two buttons that add positive (green) and negative (red) similarity points to the texture editing area. Lastly, the bottom right panel contains a commit-to-changes button.

4.1 User Workflow

Figure 67 shows the typical work flow for the system. At the start of the interaction process the image upon which the editing operations will be performed is opened. The user then enters a cycle of altering the painting settings and committing to an edit. Finally, when the user is satisfied with the results, the image is saved to an output file.

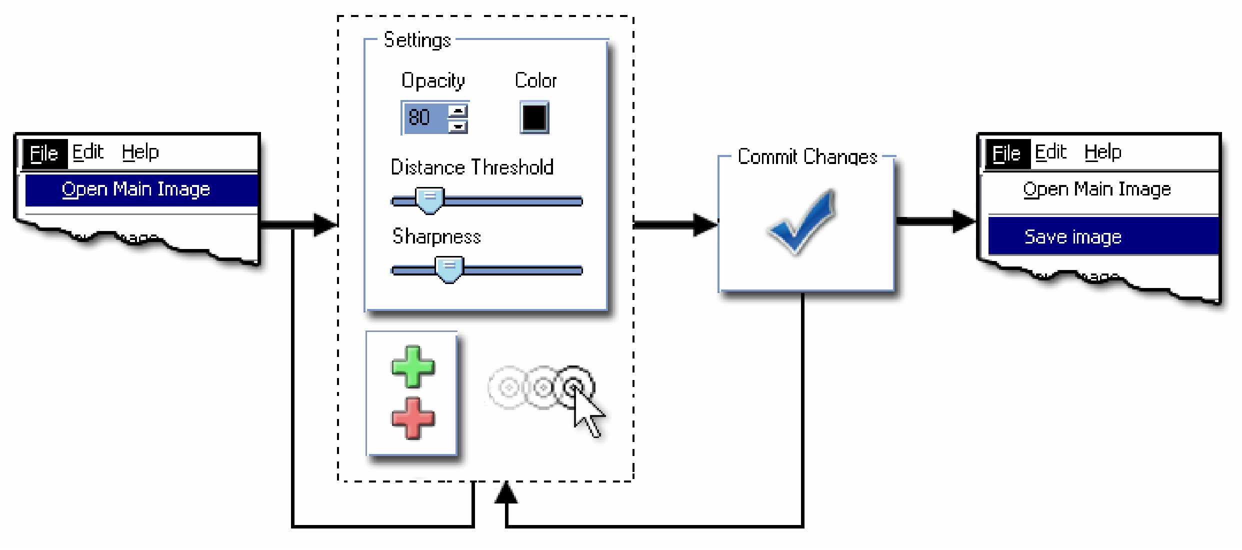


Figure 67: User workflow for self-similarity painting.

4.1.1 Replicated Painting Parameters

The aim of replicated painting is to make it easy for the user to perform global editing operations and this is reflected in the simplicity of the user interface. There are at most six parameters that the user can adjust to affect the replicated painting tool: the user can select the current painting colour, position the selection point (or points), add or remove additional selection points, adjust the global opacity level, adjust the sharpness and adjust the distance threshold.

To facilitate positioning of the selection points, each one has rings that are meant to convey the extent of the relevant local area that is used in determining similarity. The user grabs the ringed selection point with the mouse pointer and moves it to the desired position where it remains until actively moved again. The ringed selection point can be moved in either the original image view or the view of the image that is currently being edited. Either way, the two copies of the same selection point move in lock step. The local neighbourhood of the chosen pixel is then compared against that of every other pixel's neighbourhood in order to replicate the editing operation.

The distance threshold and opacity level controls are both monotonically increasing. By this we mean that moving the slider along one direction consistently increases the number of pixels that are affected (for distance threshold) and the overall level of opacity (for global opacity). In practice this means that there are no surprises at any point along the slider.

With regards to the sharpness slider, the intuitive description of 'sharpness' was preferred over a more technical labelling. The purpose of this is to insulate the user from the underlying technical meaning of the slider, which is to increase and decrease the wavelet contribution of the painting tool. Likewise, a conscious decision was made to hide the specific numeric distance threshold from the user by providing a simple unmarked slider. Having been given the simple instruction that slider the increases and decreases the number of pixels in the image that are affected, we believe that the user can obtain a direct understanding of the distance threshold's general function without having to comprehend the nature of the neighbourhood similarity distance measure. It must, however, be stressed that this is as yet a subjective claim.

When a positive or negative similarity point is added the new point is initially positioned at a random location in the image. Specific similarity points are removed by clicking on them with the right mouse button. All other interface operations are performed with the left mouse button. If two similarity points become coincident, the user is still able to move them independently. When the user attempts to move a coincident point, it is the point that had been moved most recently that is affected. This minor interface refinement is achieved by time stamping the most recent motion of each similarity point. In this way the similarity points can exhibit a temporary state of overlapping.

The reader may have noticed that the diameter of the neighbourhood is not included in this list. That is because in our subjective experience, the diameter of the multi-scale neighbourhood can be left at a value of 5 for good results (all examples in this thesis have consistently used this value). However, once again, this is as yet a subjective claim and a detailed user study would be required to confirm this. Such a study is beyond the scope of this dissertation.

4.1.2 Committing to an Operation

Once the selection point or points have been positioned and the parameters have been set, the user commits to the editing operation using the checkmark button seen in Figure 66. The commit button instructs the system to keep the current alteration of the original image. The user can thereby explore alternatives, commit to a change, and then attempt further edits before saving to file.

4.2 A Comparison with Adobe Photoshop®

There are several popular software packages for image editing, Adobe Photoshop® being one of the most prevalent and capable. We will now compare tasks performed with our editing system and with this commercially available editor.

If we consider the basic case, without the use of Boolean similarity expressions, the interaction sequence is simply an arbitrary permutation of the following concise operations:

1. Position the selection point.
2. Adjust the distance threshold.
3. Adjust the opacity.
4. Adjust the sharpness.

By contrast, in Photoshop® the user must manually paint each texture element separately. This can either be done with or without the use of layers. We first consider the case without the use of layering as many novice users do not make use of that advanced facility. This requires two embedded interaction loops:

1. For each texture element:
   Paint the individual element to a certain spatial extent and degree of opacity.
2. Check that the extent and opacity of the painting is globally consistent with the original intent and also consistent between the individual texture elements. If not, repeat from 1.

If the user is familiar with layering then limited global changes can be performed. In this case, the painting is carried out on a higher layer whose opacity can be globally adjusted at any time. The interaction loop is then:

1. For each texture element:
   Paint an individual element to a certain spatial extent.
2. Adjust the painting layer's overall opacity, if necessary.

3. Check that the extent of the painting is globally consistent with the original intent and also that the extent and opacity consistent between the individual texture elements. If not, repeat from 1.

Although with layering the global control of opacity is a benefit, this does not represent a fundamental improvement as the individual elements still need to be individually painted.

Given the amount of manual interaction required for these editing tasks, the user cannot easily make global painting adjustments to textures in Photoshop®. To perform textural changes in any standard image editor the user must manually manipulate the entire image. Not only is this time consuming but a non-artist would likely struggle to maintain consistency of results over all texture elements. Our texture painting system essentially performs global changes with a move of the mouse.

Chapter Summary

In this chapter, we introduced a system of interactive texture editing that utilizes self-similarity to replicate painting operations globally over an image. We began the discussion with a minimal self-similarity based painting system, in which changes made to single selection point were made to affect all pixels that exhibit similar local neighbourhoods in RGB space. The system was then improved with the introduction of multi-point Boolean similarity expressions. Although multi-point expression gave the system more flexibility, a number of limitations remained relating to the efficiency and quality of the neighbourhood similarity measure. Each of these issues was addressed in turn.

The efficiency of the distance calculations was improved through the use of compressed, multi-scaled neighbourhoods. This was followed by the enhancement of the similarity measure with regards to its perceptual validity and sharpness of response. In the final section, the system's interface components were compared against an industry standard image editing package.

GLOSSARY

CIE LAB – The colour space in which equal Euclidean distances in the space approximately represent equal perceptual colour difference.

CIE Primaries – The three standard primaries, defined by CIE in 1931 and called X, Y, Z, that can be used to match, with only positive weights, all visible colours.

Image Pyramid – Given an input image, an image pyramid can be generated where each higher level in the pyramid is a smaller version of the level below. Usually, each level occupies ¼ the area of the level beneath, with each side having ½ the length. Image pyramids allow the analysis of the image at multiple scales which often yields dramatically improved computational efficiency for certain applications.

Stationarity Criteria – The stationarity criteria is what differentiates textures from other images. Images that obey the stationarity criteria have the property that at some scale a given portion of the texture is statistically the same as any other portion.

Chapter 4

SELF-SIMILARITY BASED TEXTURE CLONING

Chapter Structure

In this chapter, we extend the self-similarity based editing system to replicate texture cloning operations globally over an image. Section 1 discusses how the cloning of one texture onto another can be performed by modifying the painting system to paint with texture rather than a solid RGB value. A number of example results are shown and the limitations of this initial approach to texture cloning are highlighted. In Section 2, the flexibility of the cloning system is improved by combining replicated cloning with the use of texture-by-numbers synthesis. This is followed by the introduction of a semi-automatic tool for the construction of texture-by-numbers masks. The discussion then turns to the maintaining of interactivity when re-arranging textures prior to replicated cloning. In the final section, the system's interface components specific to self-similarity based cloning are examined.

1. From Painting to Cloning

When constructing realistic renderings of a 3D scene, there often arises the need to layer multiple textures over a surface in order to further approximate the visual complexity of the natural world. A given surface will frequently host *secondary* textures which partially cover the *primary* texture. An example of this can be seen in Figure 68. In this instance we would define the *primary* texture as the bark which covers the entire surface of the tree trunk in the left-hand image. In the right-hand image there are two additional secondary textures which partially overlay the underlying bark. The first is a rich moss and the other is a grouping of pink flowers. One could even say in this case that the pink flowers are a *tertiary* texture that is partially covering the *secondary* moss texture.

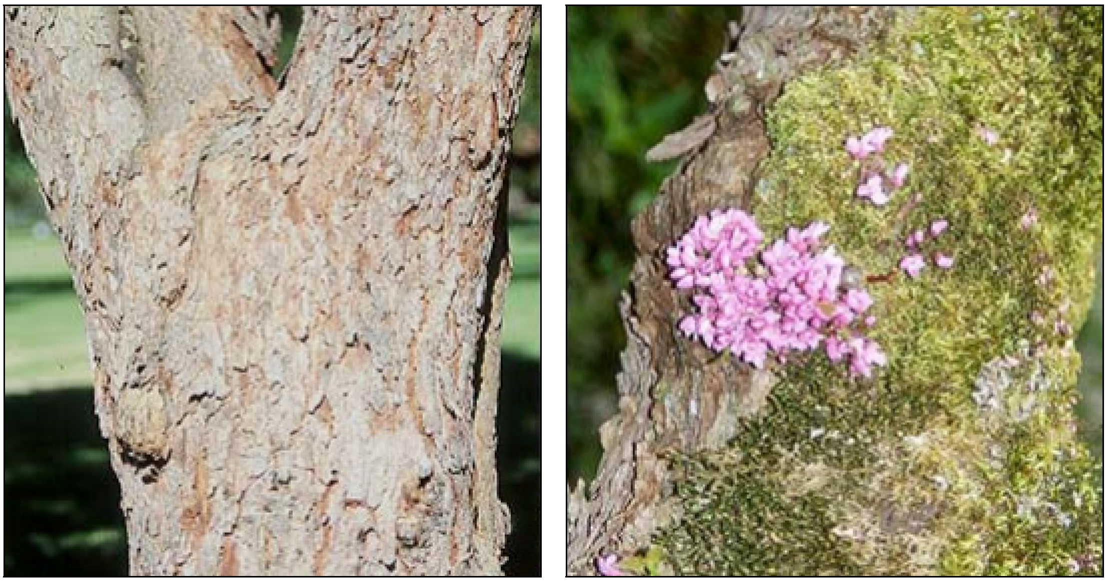


Figure 68: Primary (left) and secondary (right) textures in nature.

As discussed so far, the replicated painting system can only produce very coarse approximations to *secondary* textures such as moss. The top row of Figure 69 shows an example of the replicated painting of a solid green colour over specific areas of a bark texture. As the system is only capable of painting a solid RGB colour at varying opacities, the intricacies of the moss texture are lost. To improve the quality of results and the flexibility of the system, it would be advantageous to allow the user to paint with *texture* rather than just a solid colour. We will call this replicated painting of one texture over another *texture cloning*.

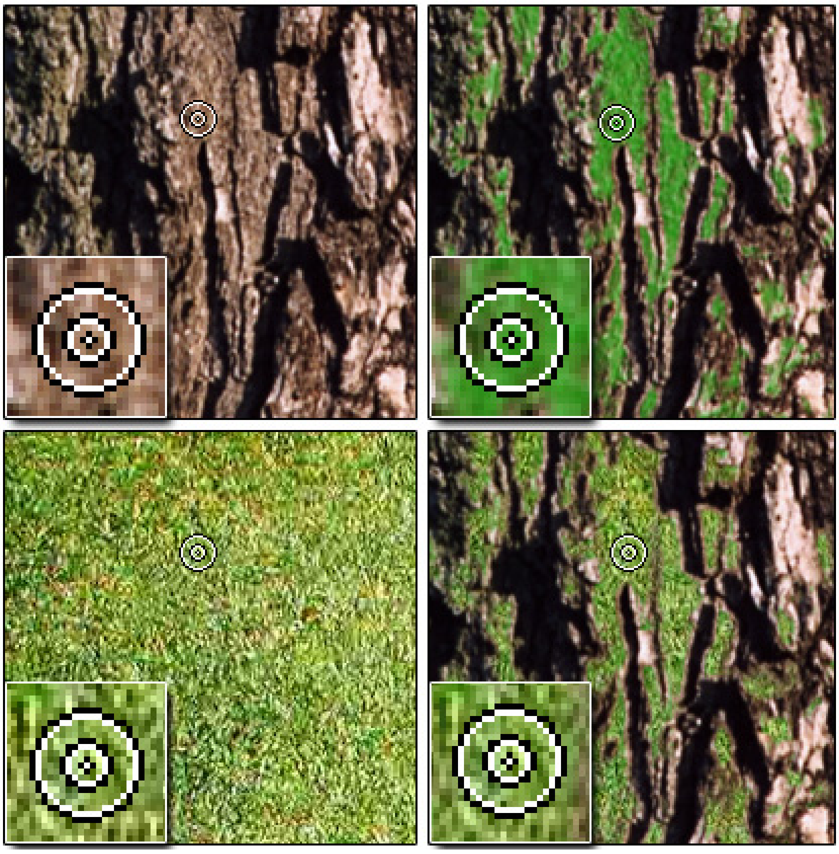


Figure 69: Clock-wise from top left: original texture, replicated painting, replicated cloning, cloning texture.

For a relatively unstructured texture such as moss, moving from replicated painting to replicated cloning is plainly a matter of positioning the cloning texture and using the corresponding colour values from the cloning texture instead of a solid RGB value. We can formally specify the computed colour at pixel, p as:

$$colour(p, A, B, I_T, I_C) = I_C(p + v_o) \times opacity(p, A, B) + I_T(p) \times (1 - opacity(p, A, B))$$

where:

I_T is the target image (bark) which is being cloned onto,

I_C is the cloning image (moss),

A is a user-selected set of positive similarity points,

B is a user-selected set of negative similarity points and

v_o is a constant offset vector which positions the cloning texture.

The bottom row of Figure 69, shows the replicated cloning of a moss texture over the same areas of the bark texture. In this case, the offset vector, v_o, is simply set to zero. We can see from the images in the right column that the ability to paint with texture can yield greater visual richness in the final image. Further examples of cloning textures can be seen in Figure 70 and Figure 71.

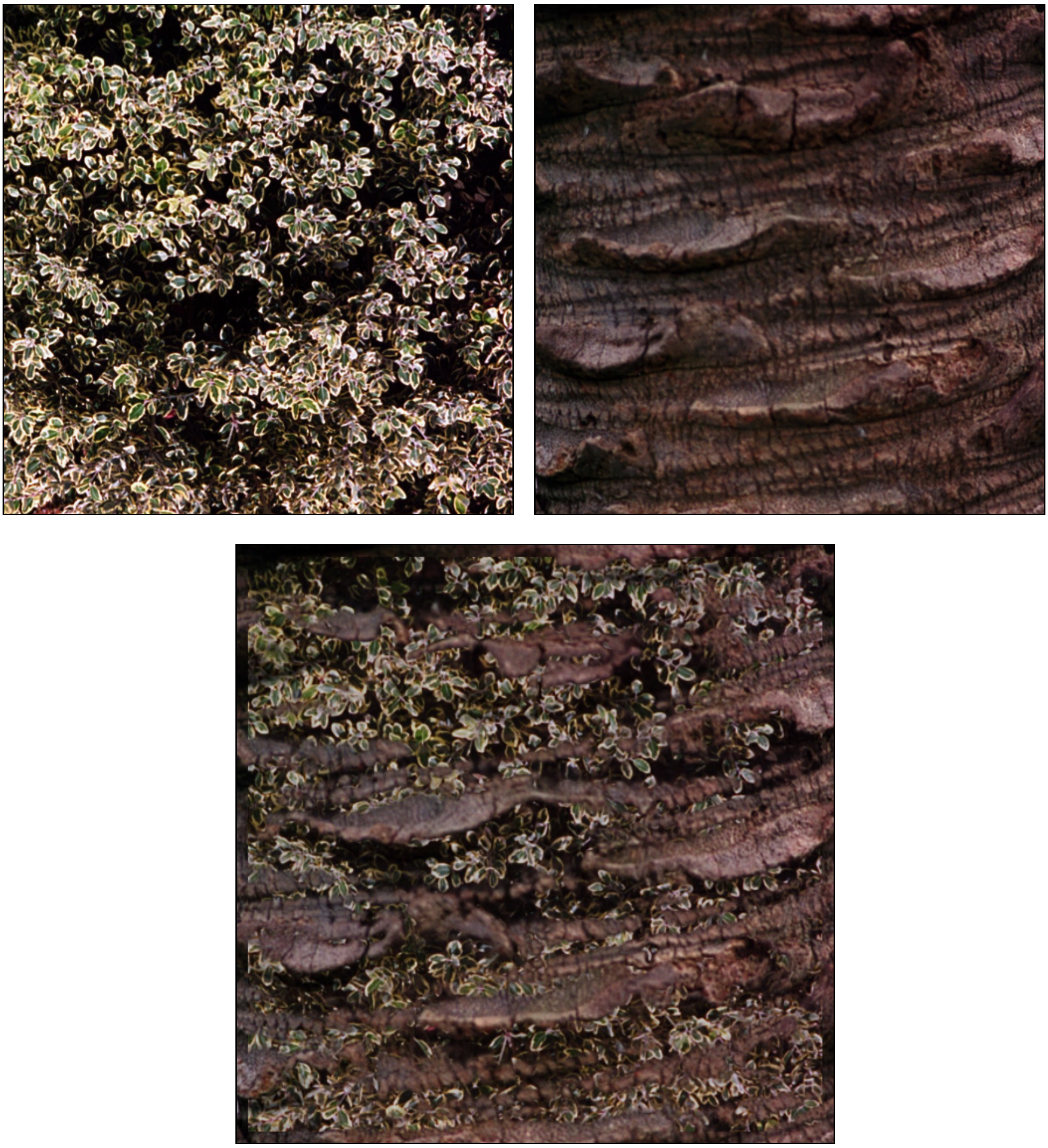


Figure 70: Leaves cloned onto a bark texture.

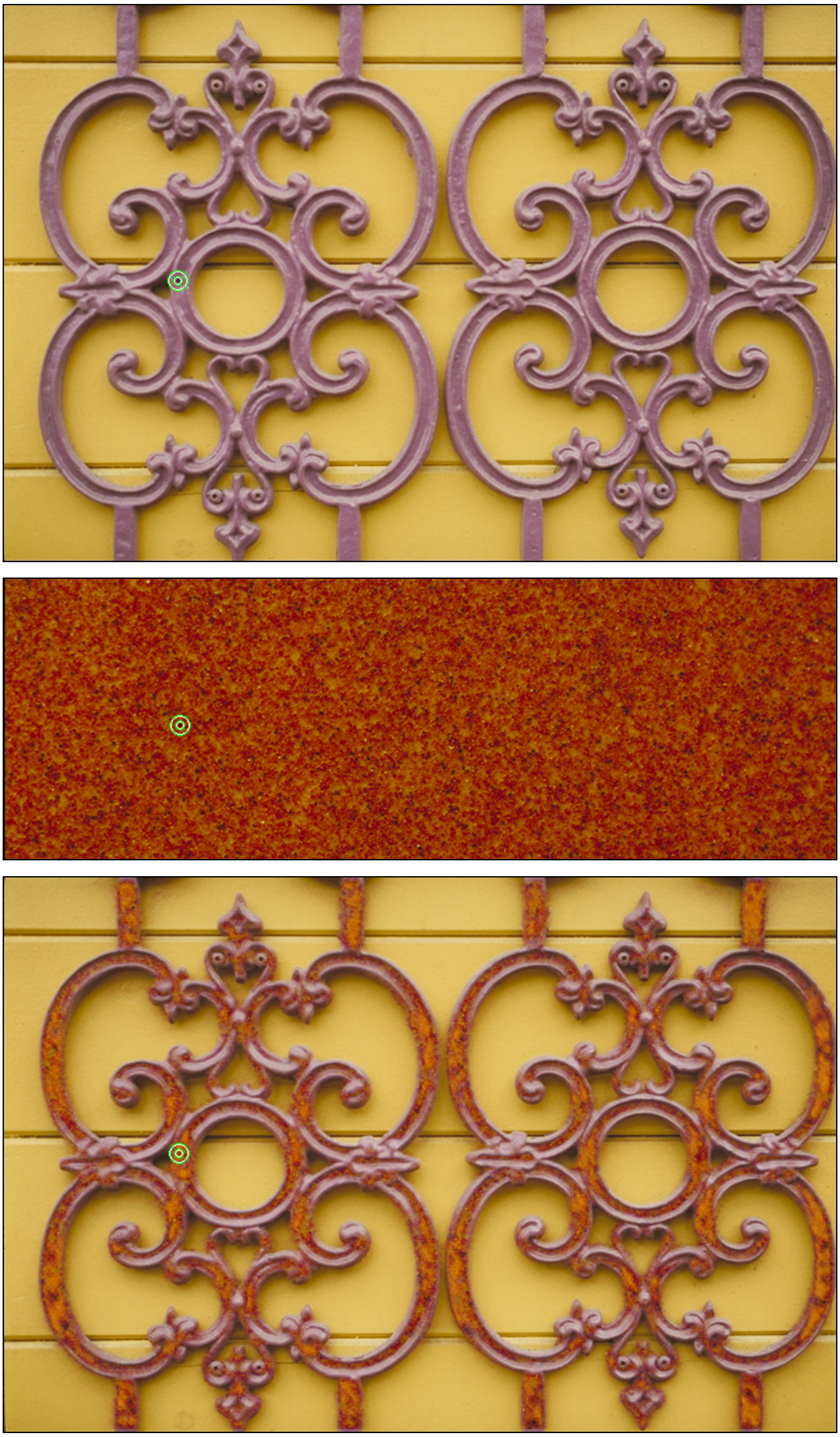


Figure 71: Simulating rust in two stages: first a red noise texture (middle image, cropped vertically) is cloned onto iron railings (top image), and then a light orange colour is painted over the cloning texture to add tone.

It is also possible to clone more structured images onto a texture to simulate the painting of a sign onto a surface. Figure 72 shows an example of cloning a SIGGRAPH 2002 logo onto a brick wall.

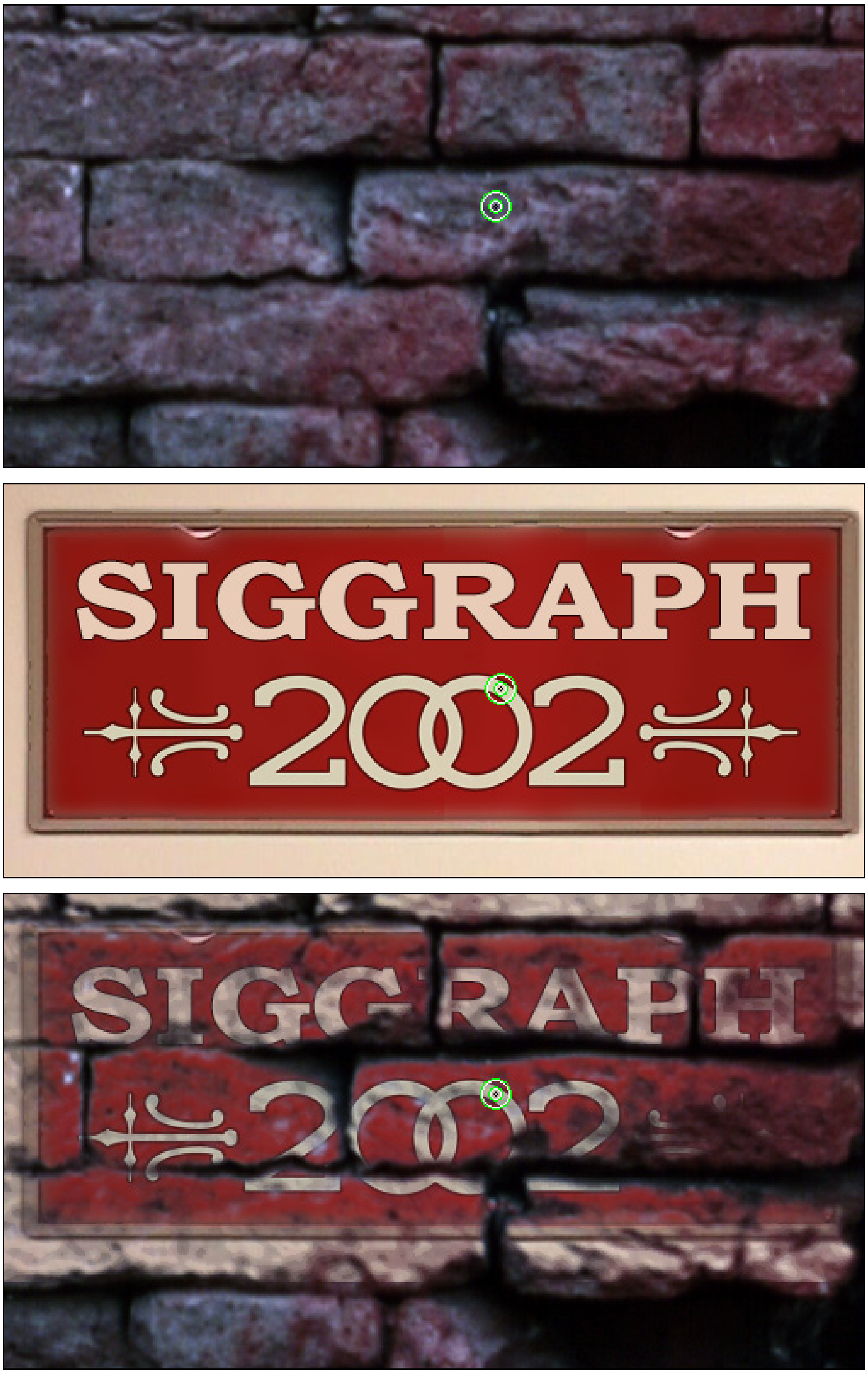


Figure 72: Similarity cloning: (top) original texture, (middle) cloning texture, (bottom) single point cloning.

Moreover, we can compare the use of wavelet responses in the cloning operation. Figure 73 shows the same cloning operation with the contribution of wavelet responses increasing from the top-most image to the bottom-most.

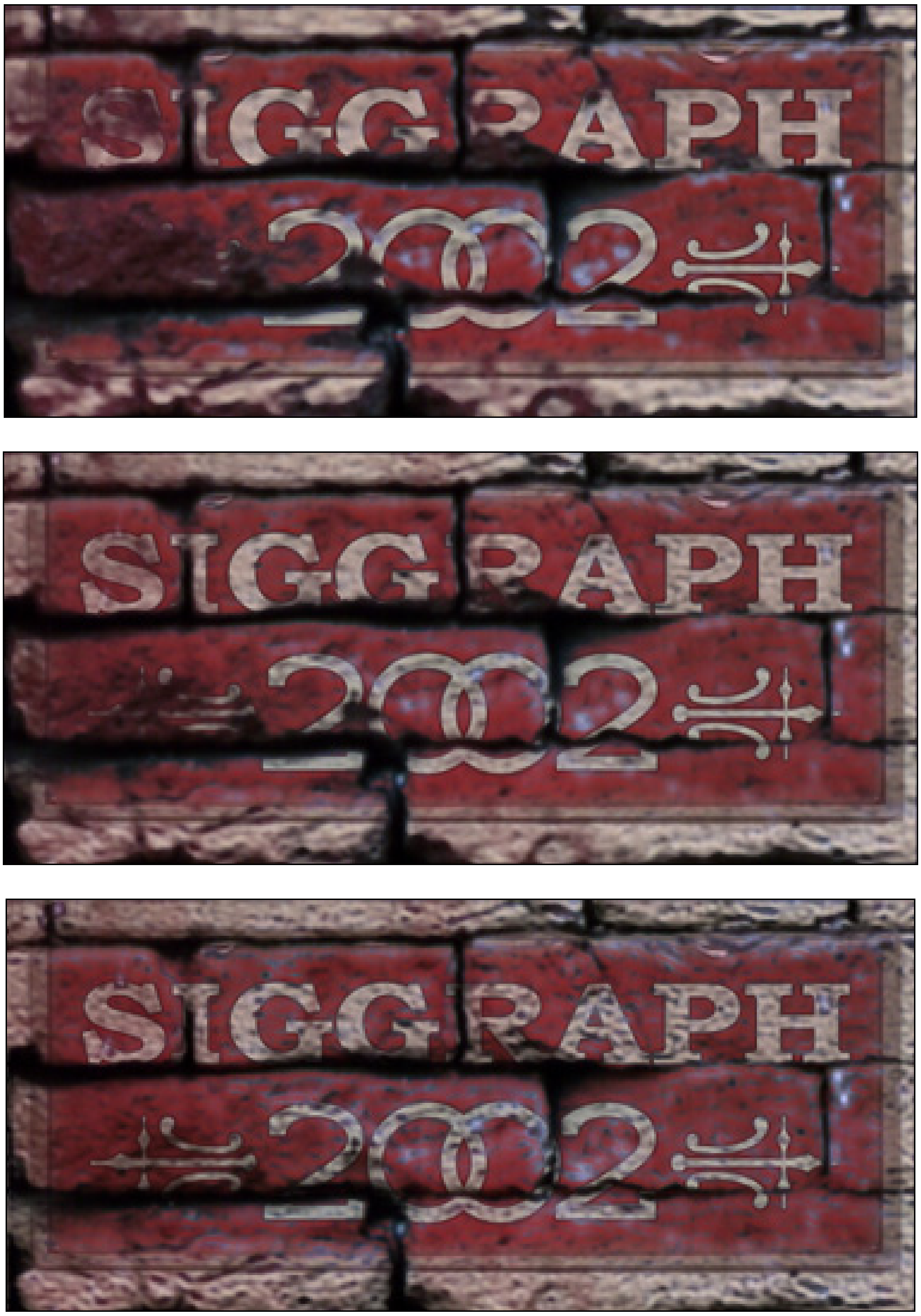


Figure 73: Increasing the wavelet contribution (from top to bottom).

Finally, in order to establish that the system is not relying solely on colour content, we see in Figure 74 an example of cloning one greyscale image onto another using only the luminance channel of the LAB colour space.

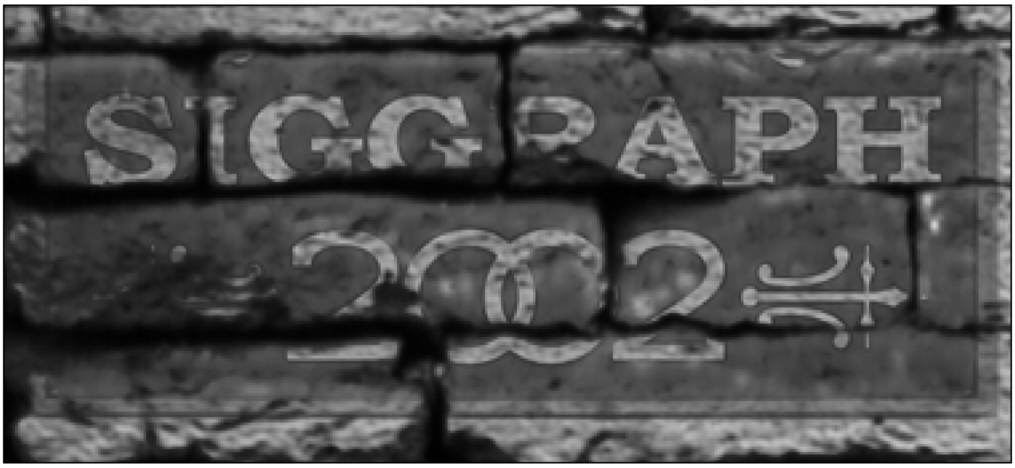


Figure 74: Greyscale computed with the luminance channel in the LAB colour space

Moving from replicated painting to replicated cloning has increased the flexibility of the system. However, apart from the painting of signs onto surfaces, there are only a limited number textures, such as moss and rust, that have a sufficiently limited structure to allow replicated cloning in this simple fashion. As we will see in the next section, for the cloning of more structured textures, further control is needed over the spatial arrangement of the cloning image with respect to the target image.

2. The Spatial Matching of Target and Source

There exist multi-layered textures in which the structure and position of the *secondary* texture(s) is dependant on the *primary* texture. However, as presented thus far, replicated cloning does not have the ability to adapt the cloning texture to spatially match the target texture; the cloning texture must match the image being cloned onto *a priori*.

We will now discuss an extension to self-similarity based texture cloning that does not simply clone the image as given, but instead re-arranges the cloning image to better match the target image. It would be a trivial extension of the system to allow the user to stretch the cloning texture under affine or perspective transformations to better match the perspective of the target texture. However, as we will see this would not fully solve the problem since the transformation required in most cases is arbitrary and non-linear.

2.1 Texture Re-arrangement

As mentioned in the related work section, a new method of controllable texture synthesis, called texture-by-numbers, has emerged which allows the user to re-arrange an image. In Figure 75, a terrain texture has been re-ordered into a new spatial arrangement. The top-left image is the hand-painted input mask of the original texture which is shown to the top-right. The bottom-left image is the output mask which designates the new spatial arrangement of the reordered texture shown to the bottom-right. The user-painted input and output masks indicate to the system where the texture is to be sourced from and where the texture is to be re-synthesized, respectively. In this example, the blue areas of the input and output masks indicate to the system where the river is in the original image and where it must be generated in the output image. However, we note that the choice of a blue label for the river is arbitrary; the labelling scheme need only be consistent. As is clear from this example, texture-by-numbers operates in a similar fashion to a child's paint-by-numbers kit. But instead of solid colours, texture is automatically synthesized into the output mapping.

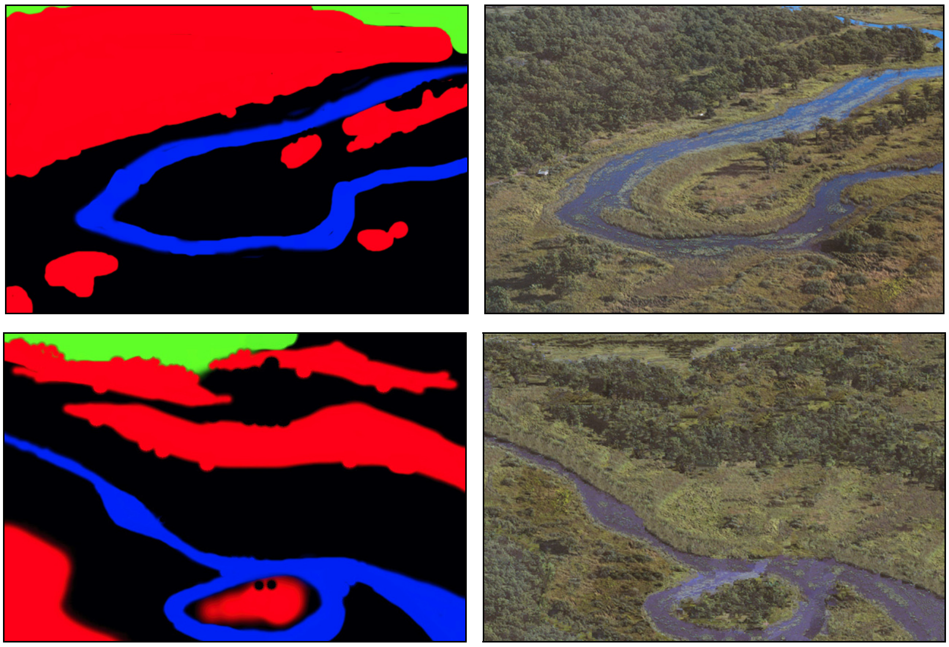


Figure 75: Texture-by-numbers re-arrangement using Hertzmann's method [Hertzmann01]. Top row: input mask, original texture. Bottom row: output mask, re-arranged texture.

As the essentials of texture-by-number synthesis can be explained fairly briefly, we will now outline the algorithm for clarity. There are three pixel-based methods of texture-by-numbers synthesis that have been developed in recent years [Ashikhmin01, Harrison01 and Hertzmann01]. In this discussion we will concentrate on the Image Analogies variant of texture-by-numbers published by Hertzmann *et al.* We choose Image Analogies for our system as, unlike Harrison's non-hierarchical approach, it offers the speed advantage of multi-resolution feedback. The approach of Ashikhmin can be considered a subset of Hertzmann's work.

The contribution of this work section is two-fold. Firstly, at the conceptual level, we adapt the texture-by-numbers synthesis framework of Hertzmann *et al.* to a novel task; texture-by-numbers is used as a means to improve replicated cloning. Secondly, we provide the user with an easy to use, semi-automatic tool for the construction of texture-by-numbers masks, rather than requiring the user to manually paint texture-by-numbers masks.

2.1.1 Image Analogies

In the notation of Image Analogies, the texture-by-numbers re-synthesis of an image is expressed as a complex 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:

$$A : A' :: B : B'$$

By this we mean that we wish to find the analogous image B' that relates to B as A' relates to A. In our terrain example of Figure 75, the left and right images in the top row are denoted as A and A' , while the left and right images in the bottom row are B and B' respectively. That is, A and B are the input and output masks whereas A' and B' are the input and output textures. The analogy therefore specifies that the input texture relates to the input mask as the output texture must relate to the provided output mask.

The synthesis procedure used to generate the output texture, B', extends the multi-level pixel-based synthesis methods that were discussed earlier in Chapter 2. As before, the Image Analogies algorithm proceeds by synthesizing each new pixel in the output image, B', in scan-line order by finding pixels with matching local neighbourhoods in the original texture. Where the Image Analogies algorithm differs from normal texture synthesis is with the treatment the analogous relation of $B : B'$ to $A : A'$. When selecting the next synthesized pixel, the local neighbourhood comparison uses a concatenation of the neighbourhood in the input texture, A', with the corresponding neighbourhood in the input mask, A. A depiction of the synthesis of a single level of the output image pyramid is shown in Figure 76.

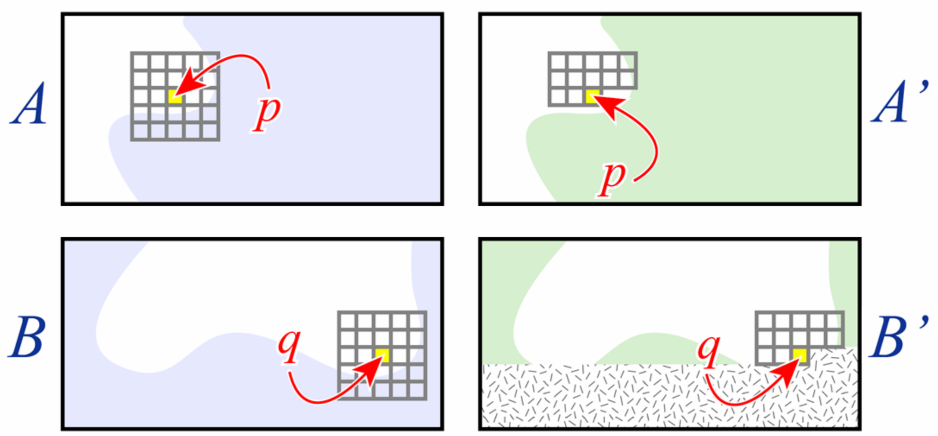


Figure 76: Single level of the texture-by-numbers synthesis procedure. To synthesize the new pixel value at *q* in the output image *B'* we consider the set of pixels in *B* and *B'* in the local neighbourhood around *q*. The system searches for the pixel *p* which has similar local neighbourhoods in both *A* and *A'*.

In order to synthesize the new pixel value at position *q* in the output image, B', we consider the two sets of pixels in the output mask, B, and the output texture, B', contained in the local neighbourhoods around *q*. We search for the pixel *p* which has similar local neighbourhoods in both the input mask, A, and the input texture, A'. As with regular texture synthesis, only half of the local neighbourhood around the synthesized pixel in the output texture, A', is known at any time. The remaining images, A, B and B' are fully known throughout the process. The neighbourhood vector is therefore the concatenation of the half-neighbourhood surrounding pixel *q* in the output texture, along with the full-neighbourhood surrounding pixel *q* in the output mask.

The distance calculation used to determine the colour value at *q* becomes the sum of square differences between each of the RGB values in the concatenated neighbourhoods of the input (texture)&(mask) and the output (texture)&(mask). The output colour value at *q* is set to the colour value of pixel *p* in input texture with the lowest (concatenated) neighbourhood distance from *q*. By considering the local neighbourhoods in the input and output textures, the properties on the input texture are maintained during synthesis; by simultaneously considering the local neighbourhoods in the masks we force the system to obey the desired spatial configuration of the output mask.

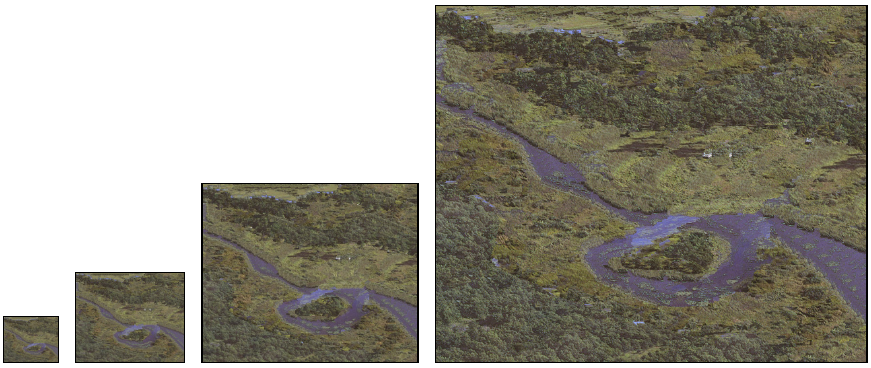


Figure 77: Texture-by-numbers synthesis with increasing resolutions (from left to right) of output.

We have reviewed the synthesis procedure of texture-by-numbers at a single level. However, as with standard texture synthesis, this texture-by-numbers algorithm is actually conducted over multiple levels of a Gaussian pyramid from the lowest to the highest resolution. Like standard texture synthesis, Image Analogies first constructs Gaussian pyramids of the input texture, A'. The system then also constructs Gaussian pyramids for both of the control masks, A and B. Next, each new pixel in the output image, B', is synthesized in scan-line order on each level of the pyramid. While synthesizing each level l, higher pyramid levels which have already been synthesised are taken into account. The local neighbourhoods (in both the texture and the mask) at the same position at higher levels of the pyramid are concatenated into the neighbourhood vector when computing the distance. To be clear, the neighbourhood vector is now the concatenation of RGB values surrounding the pixel in both the texture and its associated mask over multiple levels of the pyramid. An example of texture-by-numbers synthesis producing increasingly higher resolution outputs is shown in Figure 77.

We can write the complete Image Analogies algorithm more precisely in pseudo-code as follows:

```
function ImageAnalogies ($A$ , $A'$ , $B$ )
    compute Gaussian pyramids for $A$ , $A'$ and $B$
    for each level $l$, from coarsest to finest, do
        for each pixel $q \in B_l'$ , in scan-line order, do
            $p$ = BestMatch ($A$ , $A'$ , $B$ , $B'$ , $l$, $q$)
            $B_l'(q) = A_l'(p)$
    return $B_l'$
```

where:

```
function BestMatch ($A$ , $A'$ , $B$ , $B'$ , $l$, $q$)
    $d_{\min} = \infty$
    for each pixel $s \in A_l'$ , in scan-line order, do
            $d_s = (F(A,s) - F(B,q))^2 + (F(A',s) - F(B',q))^2$
            if ($d_s < d_{\min}$)
                $p = s$
                $d_{\min} = d_s$
    return $p$
```

and where $F(I, p)$ is the neighbourhood vector at point p in image I composed of RGB values. Further details concerning multi-resolution texture-by-numbers can be found in Hertzmann *et al.* [Hertzmann01].

2.1.2 Self-Similarity Based Texture-By-Numbers

If we use the self-similarity cloning tool in combination with texture-by-numbers, it introduces the potential for controlling how the cloning image is re-arranged prior to cloning. The user can tailor the cloning image to better match the target image spatially. An example of this can be seen in Figure 78. If Figure 78A is cloned into Figure 78B, the arrangement of the flowers would be arbitrary with respect to the rusted ring. The result of this cloning operation appears in Figure 78C which shows that the flowers have been cloned irrespective of the shape of the ring. Alternatively, when the flower texture (Figure 78A) is re-arranged (Figure 78D) to match the target ring (Figure 78B) prior to cloning, it results in a more appropriate cloning operation (Figure 78E) with the flower texture following the shape of the ring image.

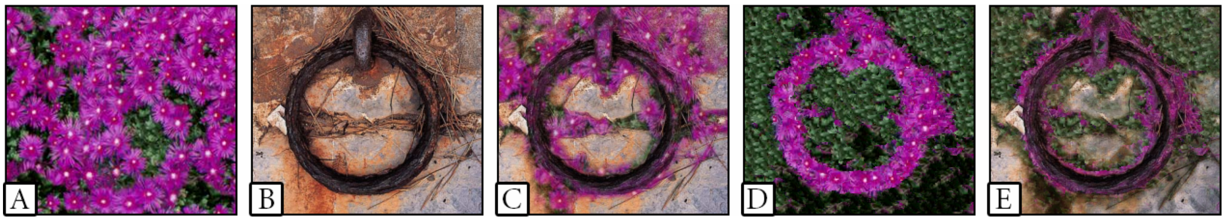


Figure 78: Semantically meaningful texture cloning: A) Texture source for cloning. B) Image target for cloning. C) Cloning with the original source texture. D) Re-arranged version of A using self-similarity masks. E) Final cloning.

To re-arrange a texture prior to cloning, the user must provide both an input mask and an output mask. The input mask segments the input texture into the distinct regions that we wish to spatially re-arrange. A hand-painted input mask of the flower texture is shown as the right image in Figure 79. This input mask segments the image into regions of purple flower versus green leaves. The output mask denotes the new spatial arrangement of the flowers which is the ring shape of Figure 78B. The output mask must therefore be made to match this ring shape. A hand-painted output mask which separates the ring from the rest of the image is shown as the right image in Figure 80.

Manually constructing the output masks for both the input texture and the output texture is a perfectly valid way to re-arrange the cloning texture. However, in the next section we will present a semi-automatic tool which provides the user with further assistance when constructing these masks.

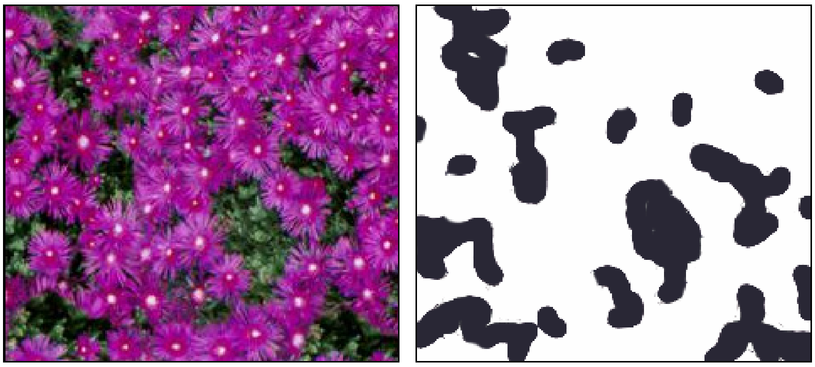


Figure 79: Input texture (left) and hand-painted input (right) mask.

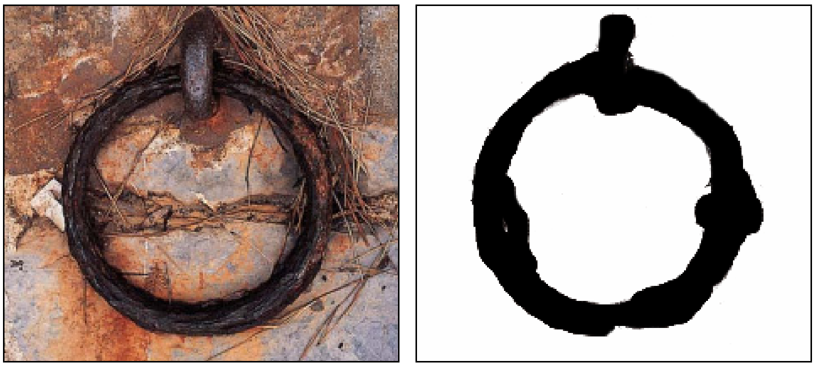


Figure 80: Target image (left) and hand-painted output (right) mask.

2.2 Semi-Automatic Construction of Masks

We could force the user to painstakingly construct the texture-by-numbers masks by hand but this would not be in keeping with the concise nature of self-similarity based editing. However, the automatic construction of image masks is equivalent to image segmentation which is a very difficult problem that we have not tried to solve. We have instead provided the user with intelligent tools to assist him or her with the construction of the necessary masks. To automate time-consuming mask constructions, we have developed an addition to the self-similarity editing system that separates an input texture into distinct regions under user guidance. For those familiar with the terminology of commercial image editing systems, this tool can be seen as sophisticated *magic wand.*

Like other self-similarity based editing tools, the masking *wand* compares the multi-scale neighbourhoods of all pixels to the neighbourhood of the similarity point(s) that the user has selected. But instead of a single distance threshold that controls opacity, in the case of the masking *wand* there are now multiple thresholds each controlled with separate points on the same slider. Points that have *lower* computed similarity to the similarity point(s) are assigned one colour and those with *higher* similarity are assigned another. Although we provide default values for what is considered *lower* versus *higher* similarity, it is the user who is able to alter these designations by moving positions on the multi-point slider.

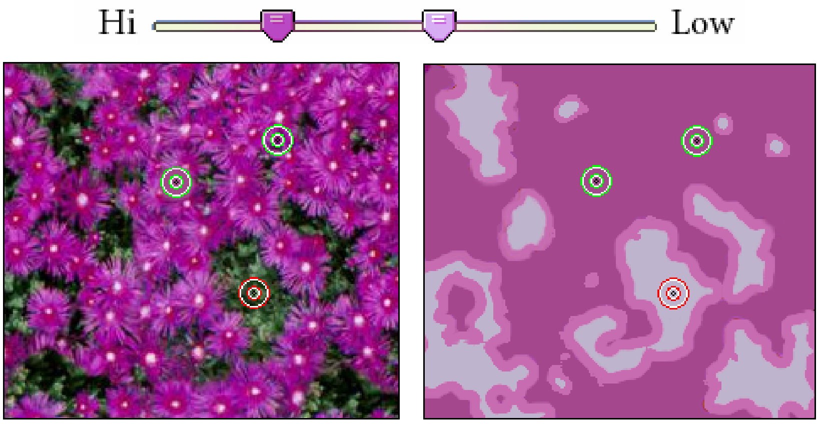


Figure 81: Semi-automatic masking of source texture.

In Figure 81 we show the construction of the input mask. The left hand image shows that two positive similarity points have been positioned over the flowers and one negative similarity point has been positioned over the leaves. According to our distance calculations this will assign pixels with 'flower-like' neighbourhoods a *high* degree of similarity and assign pixels with 'leaf-like' neighbourhoods a *low* degree of similarity. Those pixels with neighbourhoods along the borders between flower and leaf might be assigned *medium* degree of similarity. At the top of Figure 81 is a two-point slider that is used to separate the cloning image's pixels into these three classes (*high*, *medium* and *low*). Pixels with computed distance values that are within the *high* level of similarity are denoted with the darkest purple in the right-hand mask image. Pixels that are less similar than the *high* threshold yet are above the *medium* threshold of similarity are shown in medium purple. Pixels whose value lies beyond the *medium* level are deemed to have a *low* similarity and are given the default value of light purple. In fact, the tool could separate the image into an arbitrary number of colour sets with the addition of more similarity thresholds along the distance slider. We can formalize this as grouping of all the image pixels, I , into *n* disjoint sets:

$$I \Leftrightarrow I_1 \cup I_2 \cup .. \cup I_n$$

where:

$$p \in I_i \quad if \quad t_i \leq d_{SSD}(p) < t_{i+1}$$

and where $t_1 = 0$, $t_{n+1} = \infty$. The set of t_i, with $1 < i < n$, consists of positive, monotonically increasing values set by each user-positioned threshold on the multi-point threshold slider. $d_{SSD}(p)$ is the computed sum of squares distance measure for pixel *p*.

The semi-automatic masking procedure is applied to both the cloning and the target image to derive the input and output masks, respectively. The resulting input and output masks are shown as the right-hand images of Figure 81 and Figure 82. Once the similarity masks are constructed, they are then used for a texture-by-numbers guided re-synthesis prior to cloning (see Figure 83).

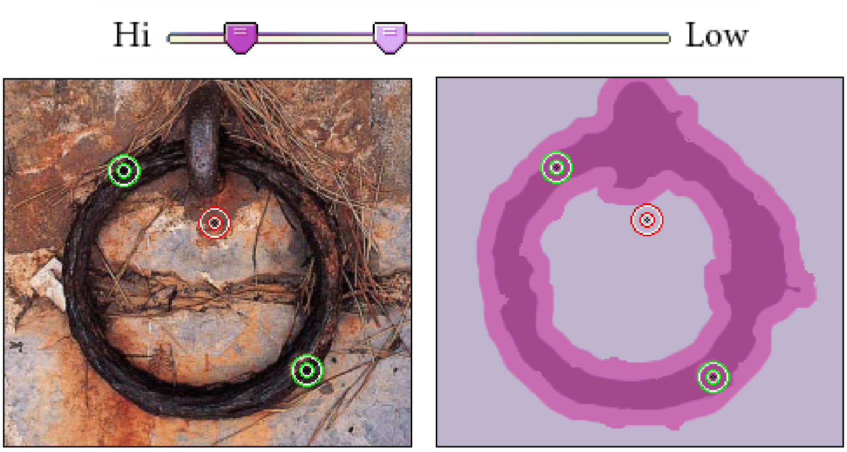


Figure 82: Semi-automatic masking of target texture.

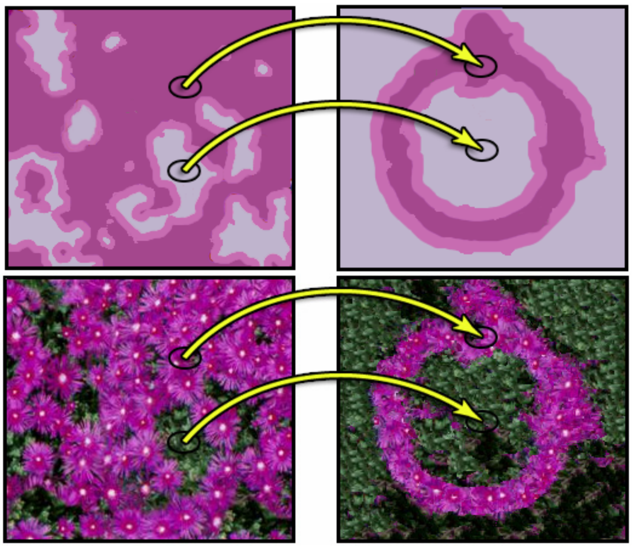


Figure 83: Texture re-arrangement prior to cloning.

The final result of the cloning operation with the newly re-arranged texture is shown in Figure 84. By re-arranging the cloning texture to match the target texture in this way, we greatly expand the range of combinations that can be explored allowing more creative freedom for the user.

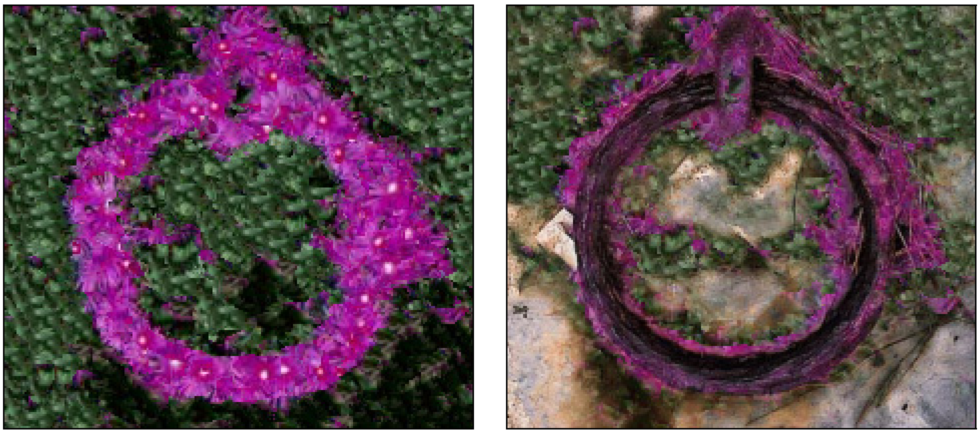


Figure 84: Final result of cloning flowers onto ring.

2.3 Maintaining Interactivity

An issue concerning texture-by-numbers which we have thus far ignored is interactivity. The consideration here is maintaining user feedback during cloning. Even with additional optimizations (which are detailed in [Hertzmann01]) the texture-by-numbers example shown in Figure 84 required 6 minutes and 7 seconds on a 2.66 GHz PC to re-order the 256×256 image prior to cloning. This cannot be considered interactive. We address this issue by providing the user with increasingly accurate approximations of the final outcome in multi-resolution passes.

Firstly, to obtain a rough estimate of the cloning operation, the user can simply perform replicated painting operations with colours representing the key areas of the cloning image. An example of this can be seen in the top 2 rows of Figure 85. In this example, the user is seeking to place snow on the rusty ring and leafy texture elsewhere in the same image by re-arranging and cloning the snow and leaf texture (Figure 85a) onto the ring image (Figure 85c). But, as given, the tree image does not correspond spatially with the ring image. So, the user begins by painting with solid colours that represent the snow and leaves. In Figure 85d the colour green has roughly been painted in area that the user intends to place the leaves. In the same image, the colour white has been painted over portions of the ring.

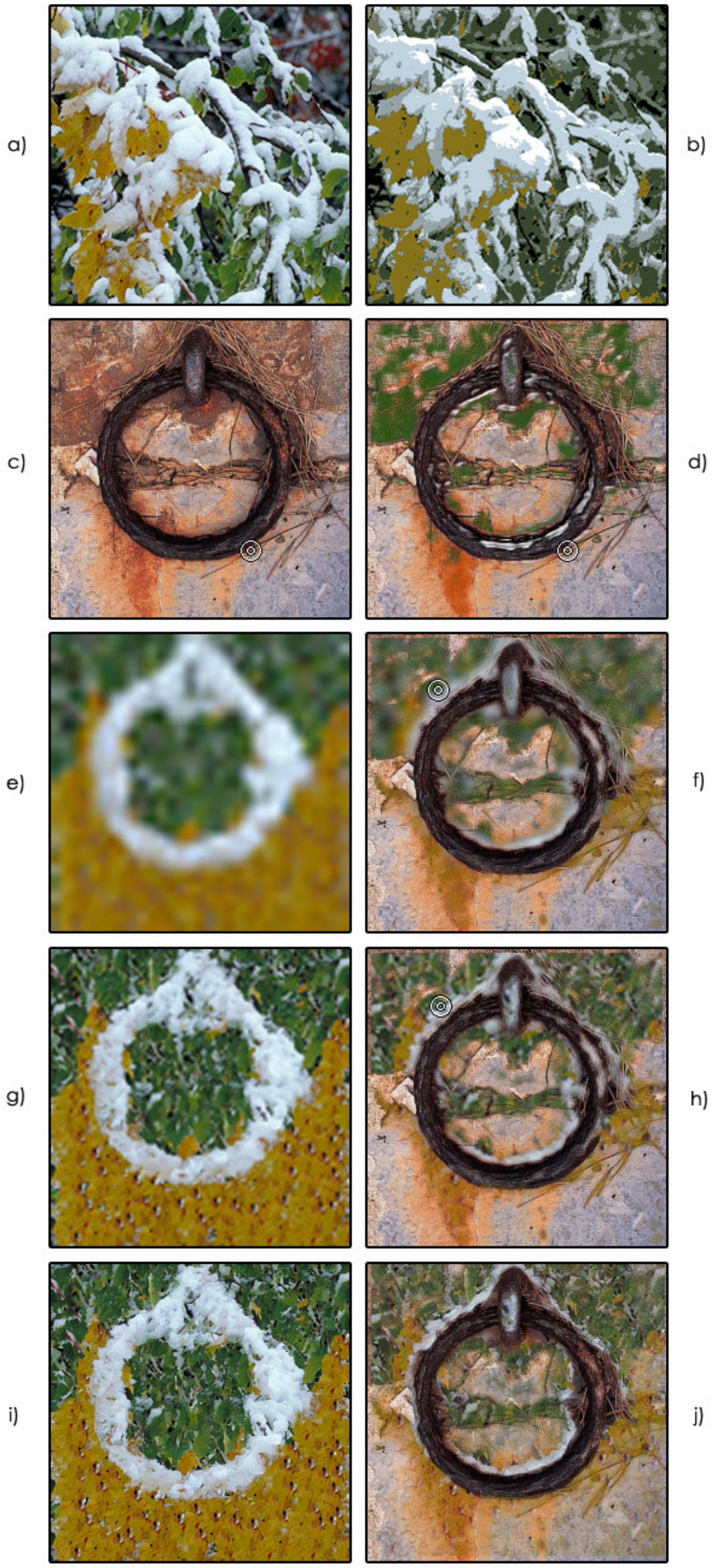


Figure 85: Texture-by-Numbers re-arrangement: a snowy leaf texture is re-ordered and cloned onto a rusting ring with increasing fidelity.

To assist in this approximation of the final outcome, our system presents the user with a reduced palette version (Figure 85b) of the cloning texture (Figure 85a) from which the user can select representative colours present in the cloning image. The user then engages in replicated painting as previously described. Using this fully interactive approximation method, the user is able to experiment freely without having to wait for the system to compute every detail.

Once the user is content with the replicated painting approximation, the user then creates the texture-by-numbers input and output masks with the masking tool, using the replicating painting approximation (Figure 85d) as a guide. With the texture-by-numbers masks in hand, the system proceeds to re-arrange the texture in increasing passes of accuracy. The passes are scaled to match the original image size and provide increasingly more accurate approximations of the intended cloning operation. We argue that these intermediary synthesis resolutions are adequate approximations to final synthesis. This is because the higher resolution levels of the synthesis tree use the lower detail levels as synthesis constraints. The overall shape and average colour of the separate texture regions are thereby maintained which helps to ensure that *what-you-see-is-*(*roughly*)-*what-you-get*.

With regards to the speed of synthesis, the first pass at a size of 32×32 (Figure 85e-f) took less than 1 second to compute; an intermediate pass at a size of 64×64 (Figure 85g-h) took 6.6 seconds to compute; and, the final pass of size 256×256 (Figure 85i-j) took 6 minutes and 7 seconds to compute. Average synthesis times for images up to 512×512 are presented in Table 8 based on the timings of 20 randomly chosen texture images. Also shown are the standard deviations, minimums and maximums for the same timings. These timings assume that the input texture is the same size as the output texture so that, for example, we are using a 512×512 input texture to synthesis a new 512×512 image.

	Synthesis Times			
Image Size	Mean	Std Dev.	Min.	Max.
64×64	0m 7.0s	0m 0.4s	0m 6.2s	0m 7.4s
128×128	1m 22.6s	0m 8.0s	1m 13.8s	1m 29.3s
256×256	8m 17.8s	2m 8.3s	5m 51.2s	9m 49.4s
512×512	126m 32.5s	43m 1.3s	77m 32.9s	158m 9.3s

Table 8: Texture-by-numbers synthesis times required to re-order textures of increasing size.

The reader may have noticed that timings for images of size 1024×1024 were not given in Table 8. This is because the lengthy duration of synthesis, which we estimate to be of the order of 3 days per texture, prohibits the gathering of such statistics. Even as a post process, a re-synthesis time of 3 days for large textures is unacceptable. Fortunately, as a benefit of using texture synthesis, we do not require that the input texture be the same size as the output texture. Texture synthesis can always produce more texture than it is given as input. Therefore, large re-synthesized textures can be generated at a much lower cost. Table 9 shows the required synthesis times for 512×512 and 1024×1024 textures using a 256×256 texture as input, based on the timings of 20 randomly chosen texture images. When we compare these timing results with those of Table 8, we see a dramatic improvement in efficiency for 512×512 textures.

	Synthesis Times			
Image Size	Mean	Std Dev.	Min.	Max.
512×512	14m 17.4s	2m 25.1s	11m 37.7s	16m 21.2s
1024×1024	152m 50.1s	29m 44.1s	122m 9.2s	181m 31.4s

Table 9: Synthesis times for large textures using a 256x256 input texture.

2.4 Results

An interesting application of texture-by-numbers cloning is fire over wood. Figure 86 through Figure 88 show examples of re-ordering fire texture so that it appears to flow around the contours of wood. To achieve this, masks of the fire textures were created by selecting a point within the fire and setting the masking thresholds appropriately. The output masks for the wood textures were then constructed by selecting the areas between wood tiles or logs, as the case may be. Looking closely at Figure 86 in particular, one can argue that the result obtained would be extremely difficult to achieve manually using an image editing package. It would certainly require the skills of an artist and even then would take considerable time to produce.

Further results using the rusted ring image are shown in Figure 89 and Figure 90. Finally, in Figure 91 we see a more creative use of the cloning tool wherein a texture composed of red and black berries are re-ordered into the shape of bark texture. When cloned, the bark texture appears to be hosting an unusual fungus.

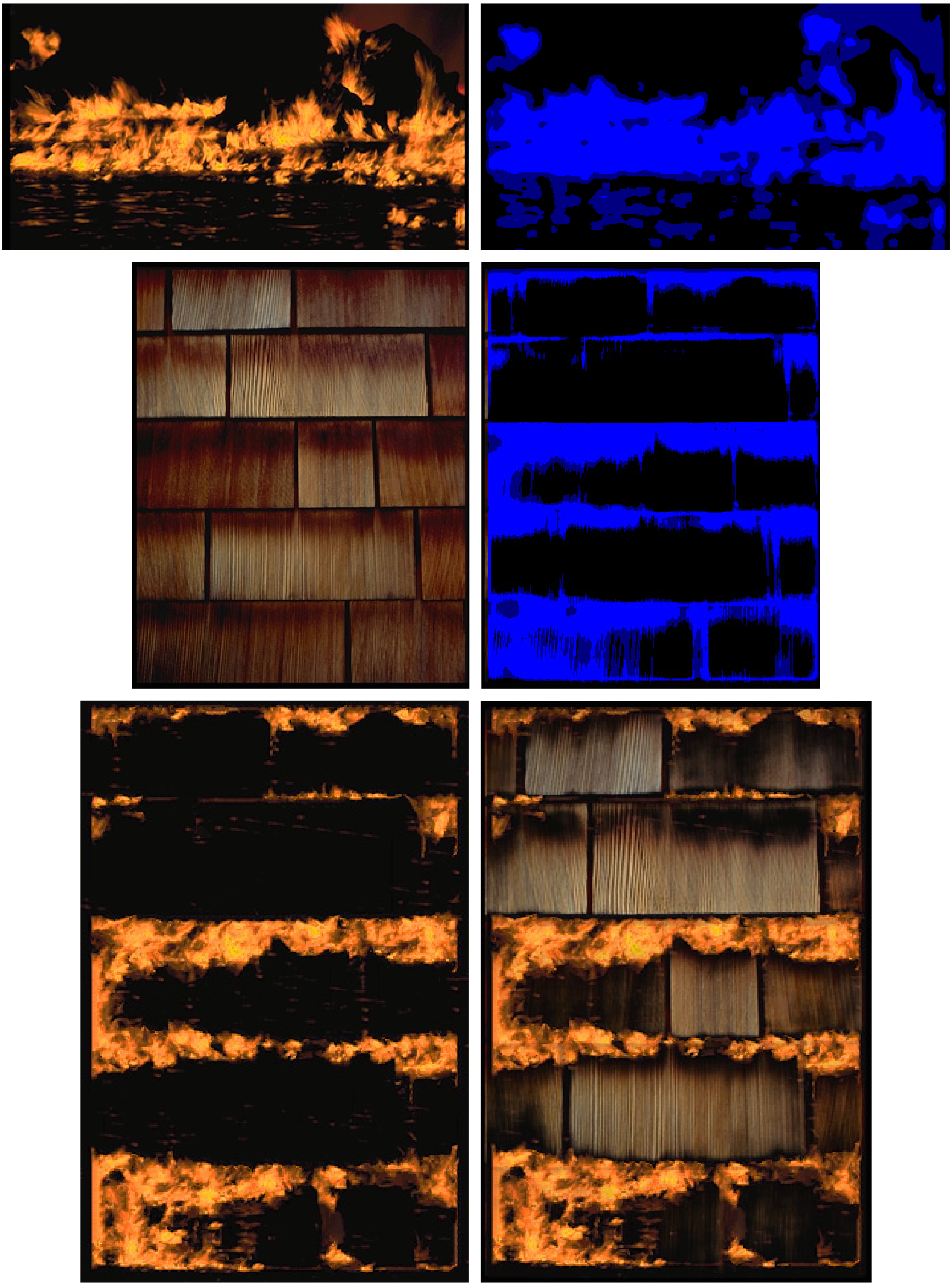


Figure 86: Fire texture is re-ordered and cloned onto wood shingles. Top row: cloning texture and mask. Middle row: target texture and mask. Bottom row: re-ordered fire texture and final cloning result.

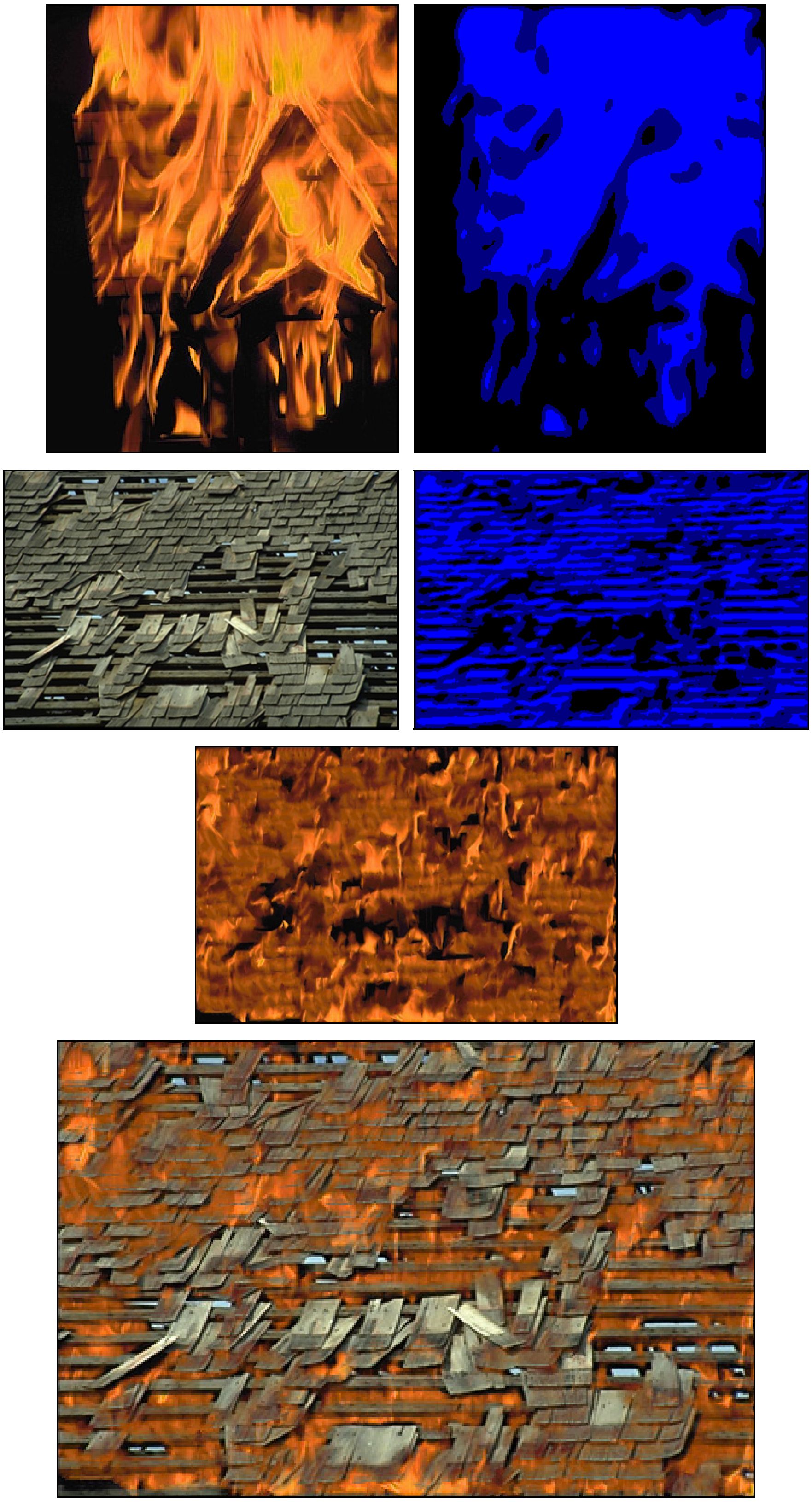


Figure 87: Fire is re-ordered and cloned onto a shingled roof. Top row: cloning texture and mask. Second row: target texture and mask. Third row: re-ordered fire texture. Bottom row: final cloning result.

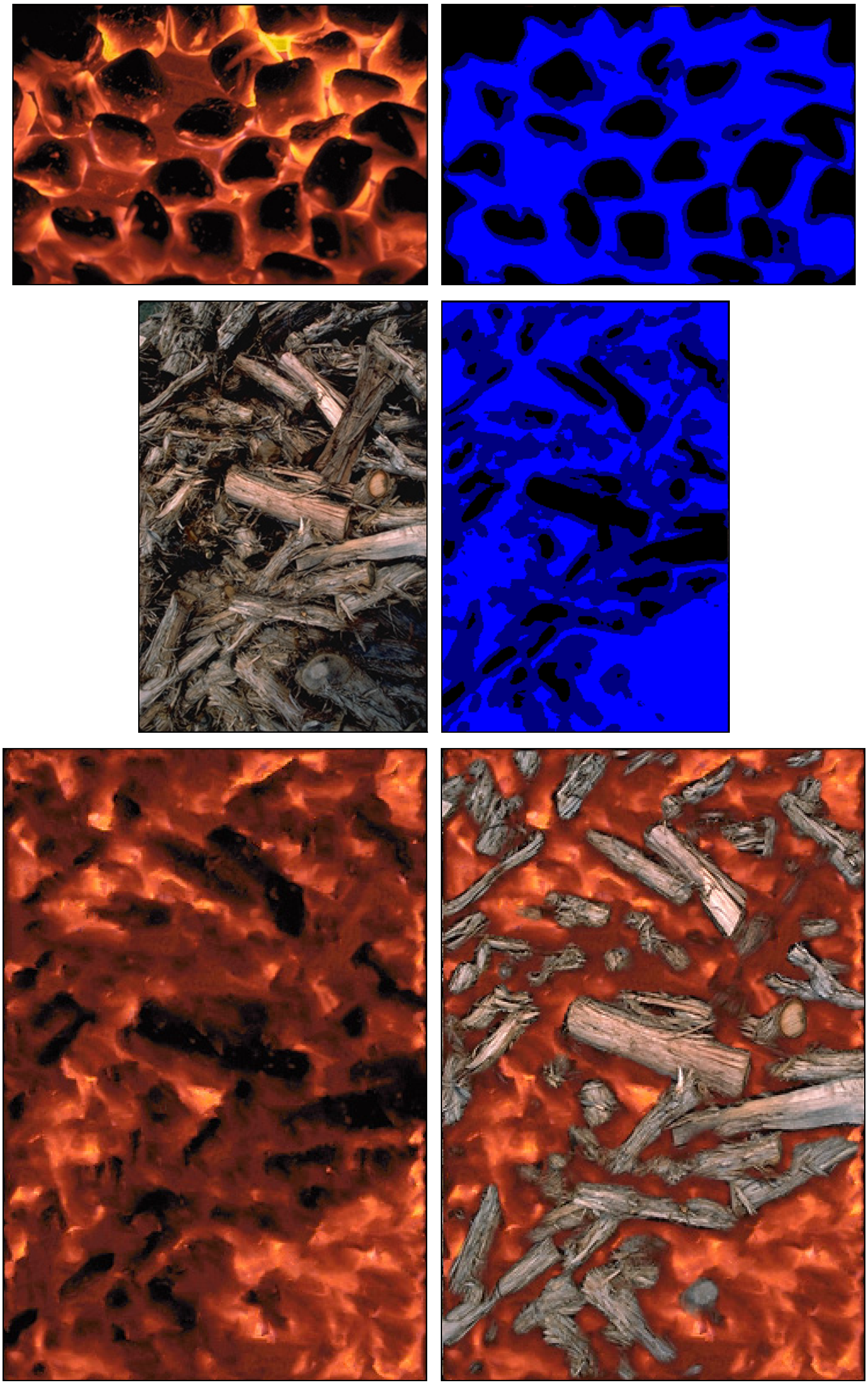


Figure 88: Fire is re-ordered and cloned onto a wood pile. Top row: cloning texture and mask. Middle row: target texture and mask. Bottom row: re-ordered fire texture and final cloning result.

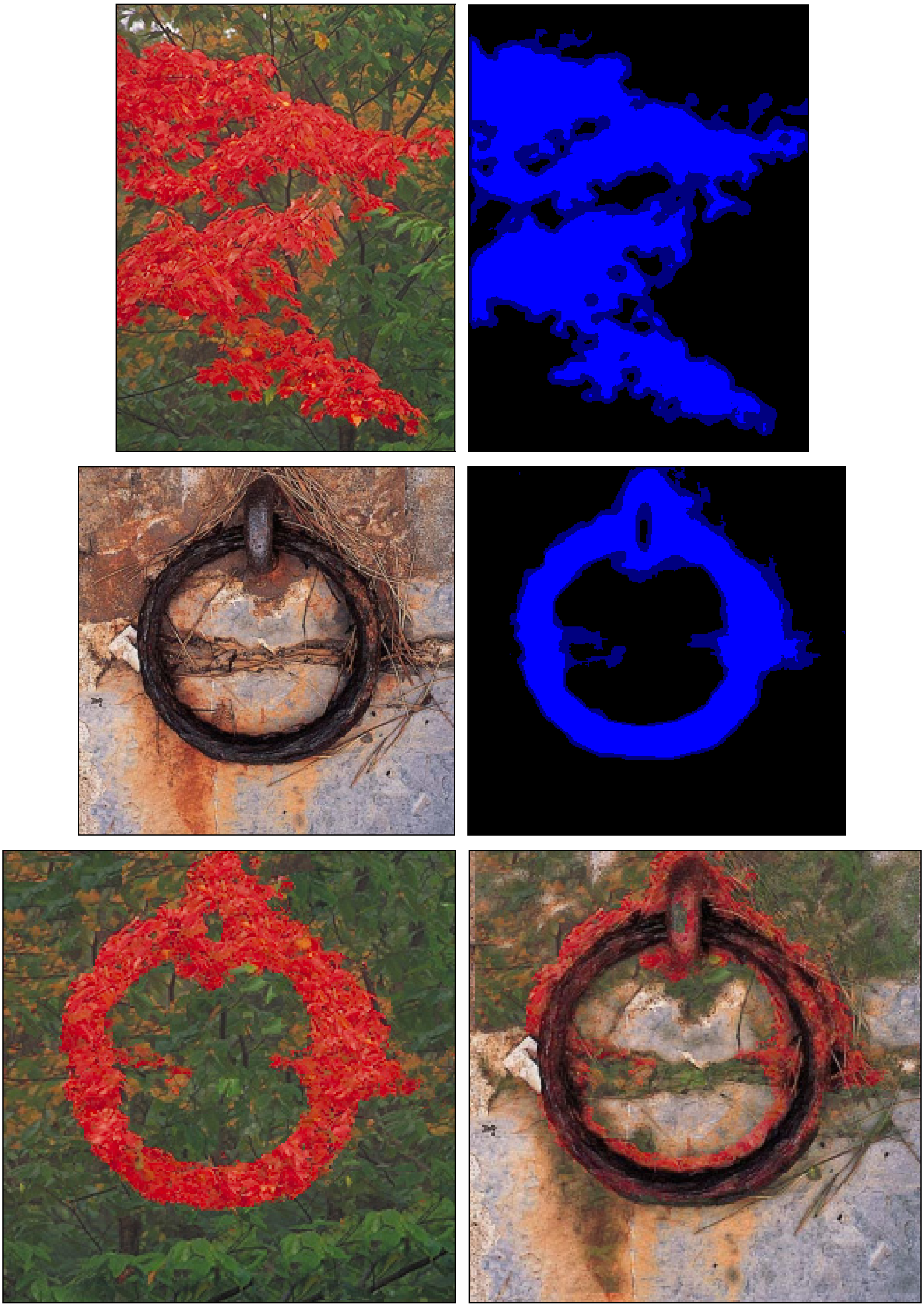


Figure 89: Red and green leaves are re-ordered and cloned onto a rusting ring. Top row: cloning texture and mask. Middle row: target texture and mask. Bottom row: re-ordered leaf texture and final cloning result.

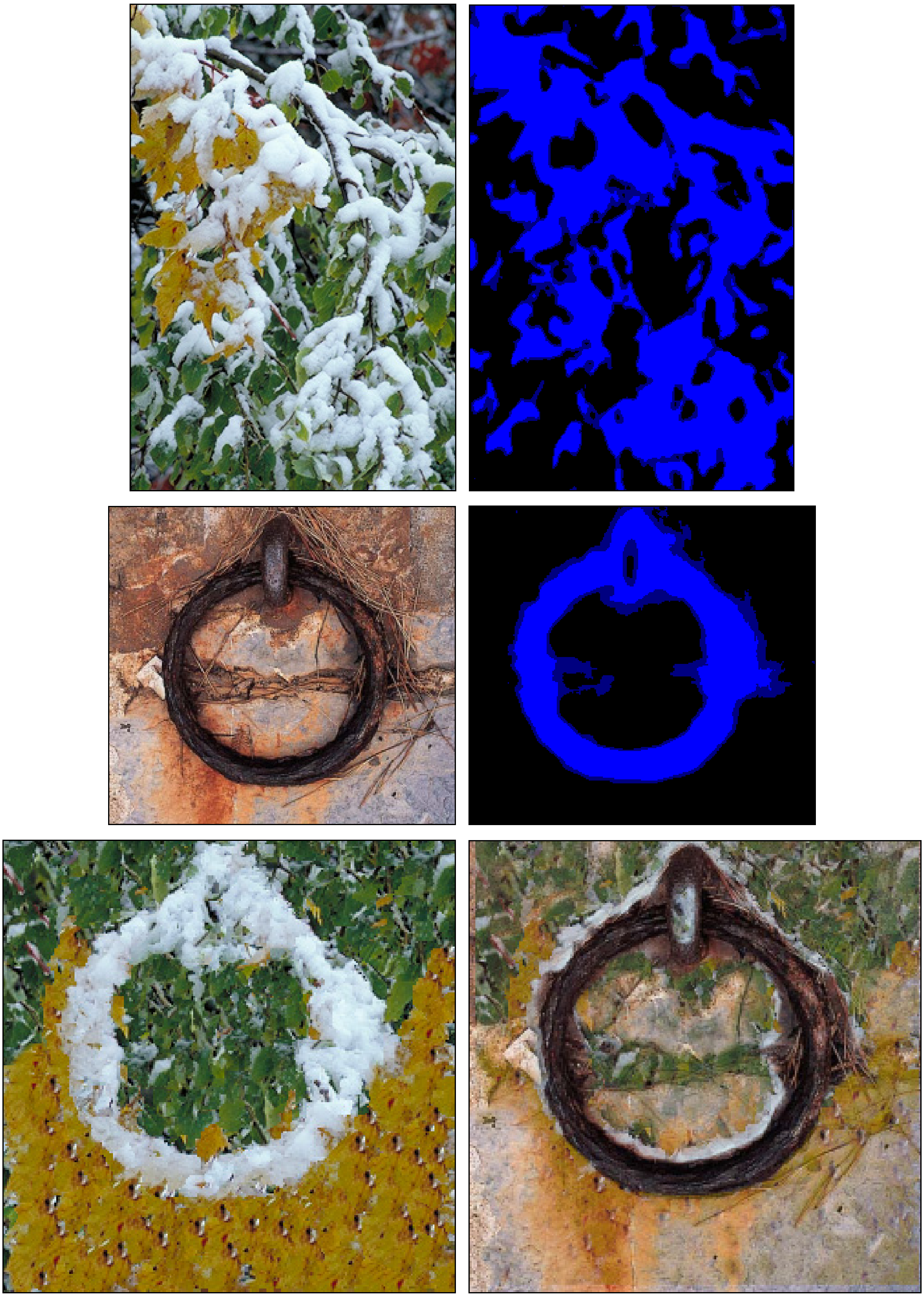


Figure 90: Snow and leaves are re-ordered and cloned onto a rusting ring. Top row: cloning texture and mask. Middle row: target texture and mask. Bottom row: re-ordered snowy leaf texture and final cloning result.

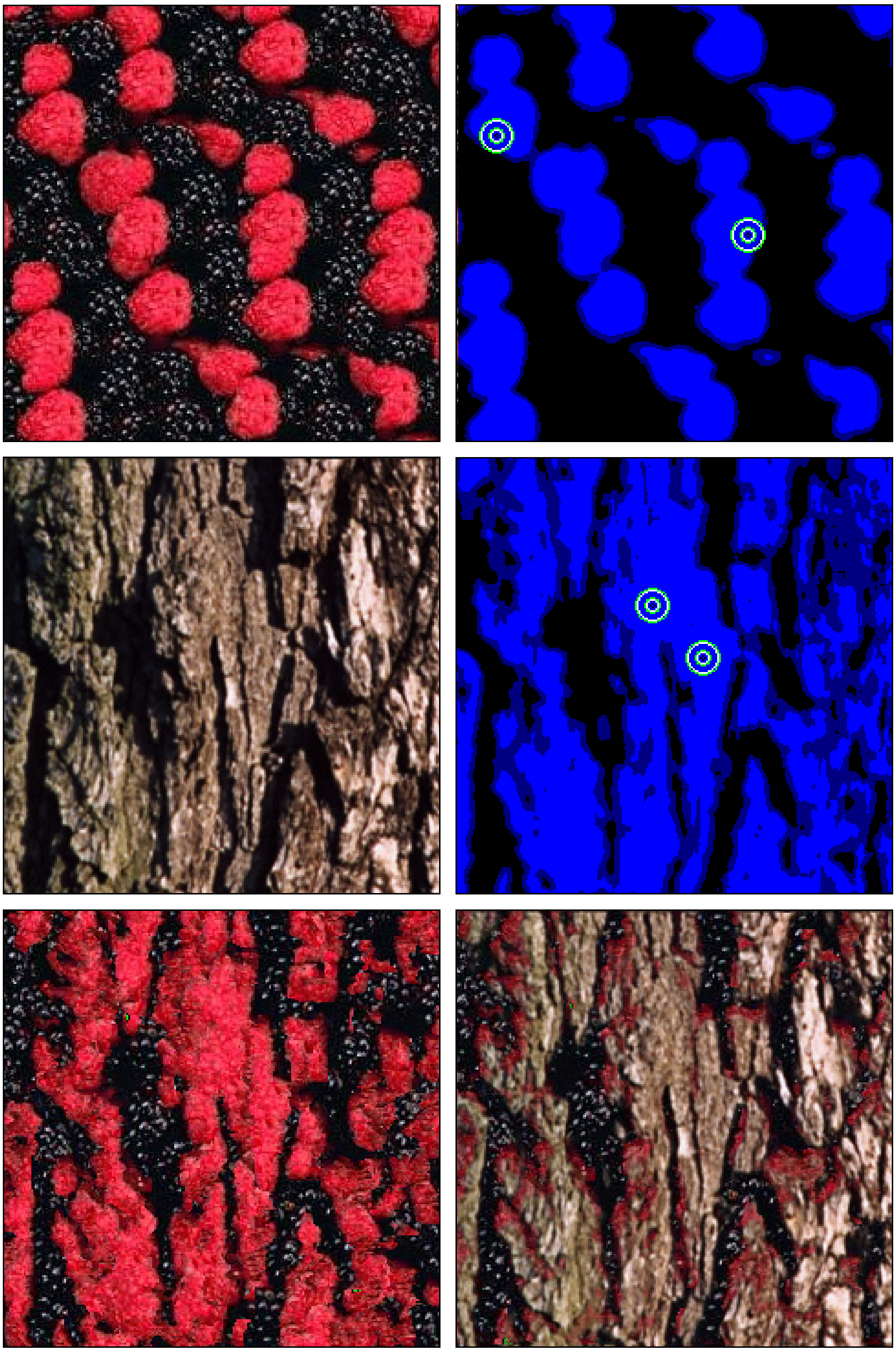
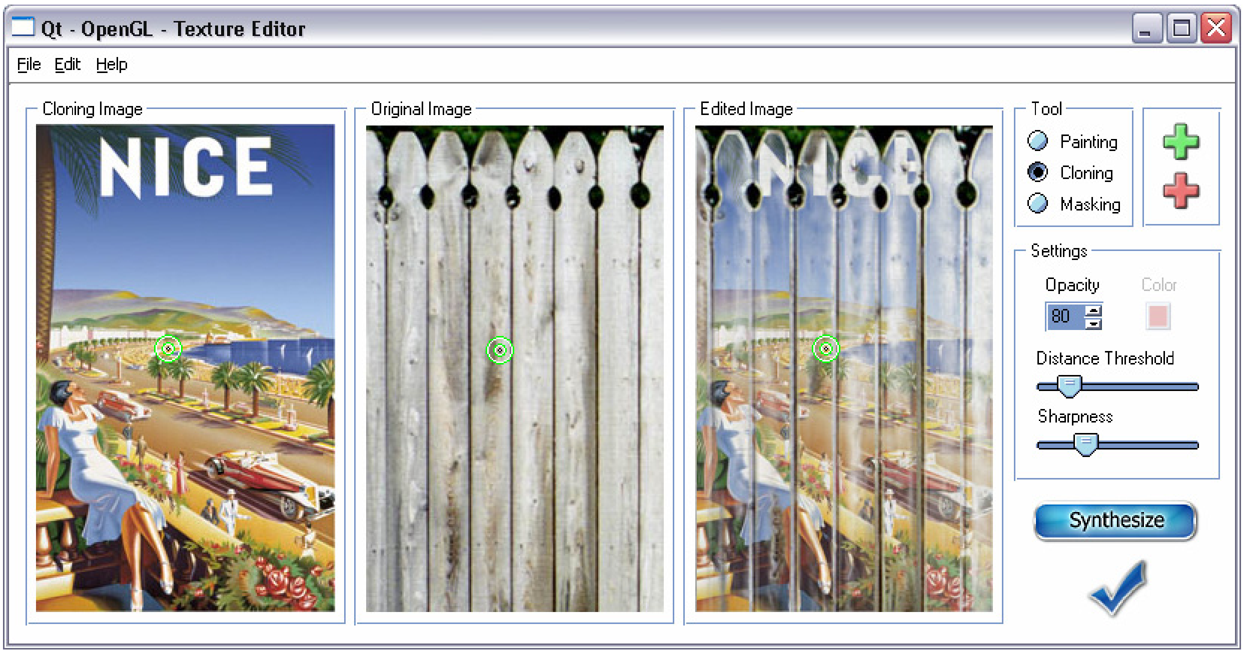


Figure 91: Red and black berries are re-ordered and cloned onto bark. Top row: cloning texture and mask. Middle row: target texture and mask. Bottom row: re-ordered berry texture and final cloning result.

3. The Texture Cloning Interface

In order to understand how replicated texture cloning is integrated with the rest of the system, we now briefly examine the additional interface components that are needed, and then follow this with a description of the user's workflow. A screen capture of the augmented interface is shown in Figure 92 which illustrates the state of the user interface after committing to a replicated cloning operation. In this instance, the cloning image (shown in the leftmost panel) has been cloned onto an old fence, giving the appearance of an aging sign. The addition of the cloning image viewing panel means that we now have three copies of the selection point(s) moving in an identical pattern as dictated by the user.



Tool

○ Painting

● Cloning

○ Masking

Opacity 80

Figure 92: Replicated cloning editor.

Like replicated painting, the cloning tool is subject to the modifier controls that are, again, contained in the panel that is labelled 'Settings'. The cloning tool's parameters are identical except that the current painting colour does not have a meaning, and so is seen greyed-out. We also note in Figure 92 that there is a new 'Tool' panel containing radio buttons offering the choice between painting with colour, cloning with texture or texture masking. A final addition to the interface is the 'Synthesize' button which is used to initiate a texture-by-numbers re-synthesis of the cloning image.

We will now briefly look at how the cloning operations are carried out in terms of the user's workflow. There are a number of additions to the user's workflow which can be seen in Figure 93. There is an added stage in the process wherein the user selects one of the three tools for editing the texture images: painting, cloning and masking. The user engages in arbitrary permutations of these operations.

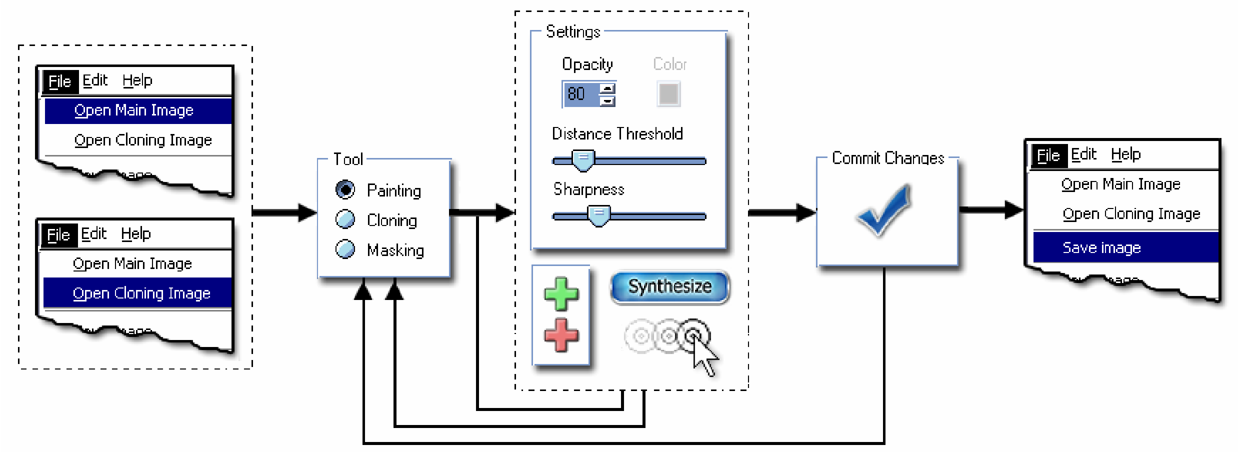


Figure 93: User workflow for self-similarity based cloning.

We will first consider the case of cloning without rearranging the cloning texture to match the target texture which does not require the use of replicated masking. Since a full comparison with Adobe Photoshop® for replicated cloning would be much the same as for painting, we will only note a few points of difference. At the start of the session, an extra cloning image must be opened in addition to the texture being edited. The main texture being edited (shown as the middle image in Figure 92) we will now call the target texture. The cloning operations themselves are performed in a similar manner to painting for our system: the user enters a cycle of positioning the similarity point(s), altering the tool settings and committing to an edit.

When performing this task in Photoshop®, the "Cloning Stamp Tool" is used in place of the paint brush. This tool allows the user to copy samples of an image and subsequently apply them over areas of another image or areas of the same image. With this tool, the user first picks a point in the cloning image and then manually paints with the corresponding pixel values from that source. Once the cloning position is selected in Photoshop®, the interaction loops are the same as described for the case of painting, requiring the same degree of manual interaction.

3.1 Cloning with Texture Re-Synthesis

To first approximate the cloning of a re-synthesized texture, the user is able to paint with representative colours sampled from the cloning texture. We facilitate this by automatically displaying a reduced histogram version of the cloning image in the left-hand image panel (see Figure 92) whenever a cloning image is open and the system is in painting mode. Clicking anywhere in the reduced histogram version of the cloning image chooses the colour at that location to paint with. This mode of approximation may well be desirable from the point of view of the artist who is seeking a rough idea of what the final result will look like, while not being forced to deal with too much visual information in the early stages.

Once a rough approximation to the cloning of a re-synthesized texture is made, the user must then construct the input and output masks prior to initiating the texture-by-numbers re-synthesis of the cloning texture. This multi-stage cloning workflow is shown in Figure 94. The user begins by opening the target and cloning images. Masks are then created for both the target and the cloning images using the self-similarity masking tool. Next, the user initiates the multi-resolution texture synthesis by pressing the 'Synthesize' button. The output of the synthesis operation is then used in the cloning operation as previously described.

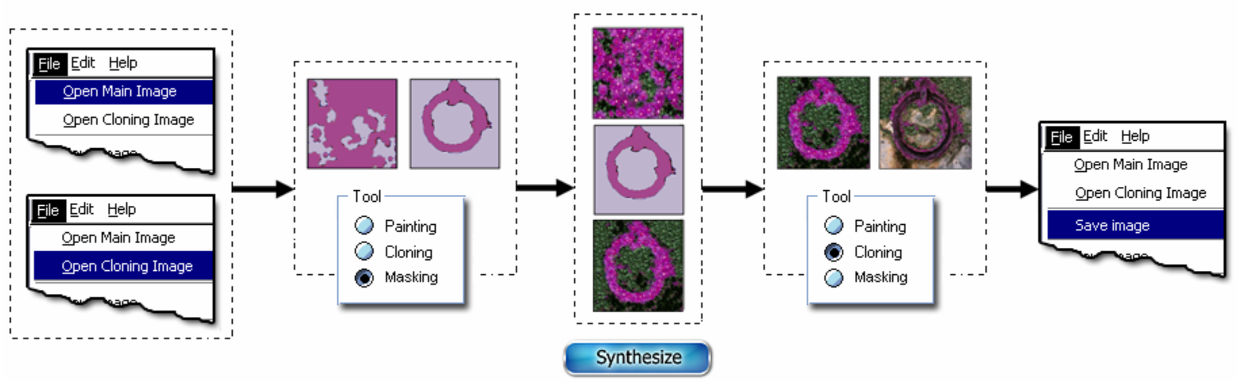
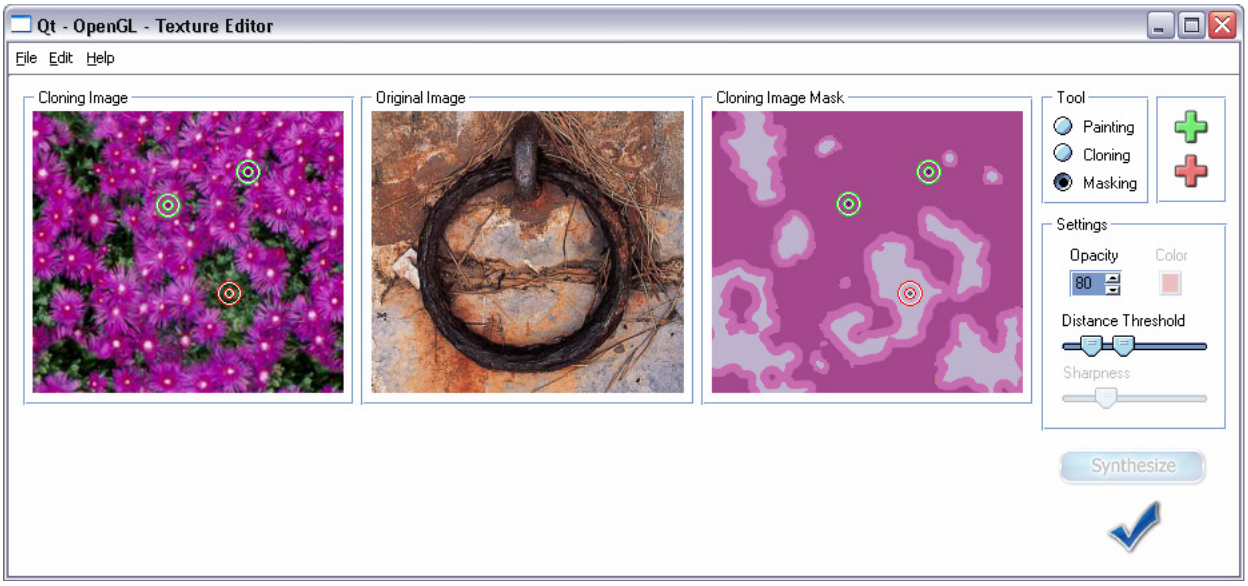


Figure 94: Texture-by-numbers workflow. 1) Open target image and cloning texture. 2) Perform self-similarity masking. 3) Re-arrange cloning texture using automatic Texture-by-Numbers synthesis. 4) Clone re-arranged texture onto target image. 5) Save final image.

3.1.1 Context Switching

In masking mode, the user is able to switch between working with the cloning texture or the target texture at any time. Clicking anywhere in either the cloning or the target texture switches the editing context between the two. A separate editing context is stored for both the cloning texture mask and the target texture mask. This is in addition to the original editing context needed for the painting/cloning operation. Each context separately stores the position of its similarity point(s) and editing parameter values. The masking result (or editing result in cloning mode), which is shown in the right-most image panel, changes to correspond with the active editing context. A screen capture of the interface during a masking operation will help to clarify these behaviours. In Figure 95, the system is in masking mode and the user is currently operating on the cloning texture. The resulting mask is shown in the rightmost image panel which has been appropriately re-labelled as 'Cloning Image Mask'. Likewise, if the user is operating on the target texture in masking mode, the image editing panel is re-labelled as 'Target Image Mask'. Another point of note is that the distance threshold slider changes from single point to multi-point in masking mode. This allows the user to specify multiple thresholds for the masking similarity levels in the generated mask. In this case there are three masking levels.



Tool
○ Painting
○ Cloning
● Masking

Settings
Opacity 80
Color
Distance Threshold
Sharpness

Figure 95: Replicated masking editor.

3.1.2 Final Notes on Masking

The cloning and target masks are used to re-arrange the cloning texture using automatic texture-by-numbers synthesis. However, the re-synthesize button is only available when masks have been constructed for both the source and the target images; otherwise, it remains greyed-out (see Figure 95). After re-synthesis has been initiated, cloning is then performed with the re-synthesized version of the cloning texture. The reader will recall that the re-synthesis procedure is conducted in multi-resolution passes. As each level of the synthesis tree is constructed for the cloning texture, higher-resolution updates are introduced with a linear transition over one second of time thereby avoiding the distraction of a popping effect.

A final note about replicated masking is that although self-similarity masking benefits from the use of Boolean similarity expressions (Figure 96), the use of steerable wavelets in the similarity calculations is not generally desirable. Since we are trying to separate the texture into broad areas of distinct texture, the use of wavelet responses is not appropriate as they respond sharply to individual features. And so, the sharpness slider is greyed-out when the system is in masking mode. Furthermore, for better masking results, small regions of less than 10 pixels are automatically discarded when texture-by-number synthesis is initiated.

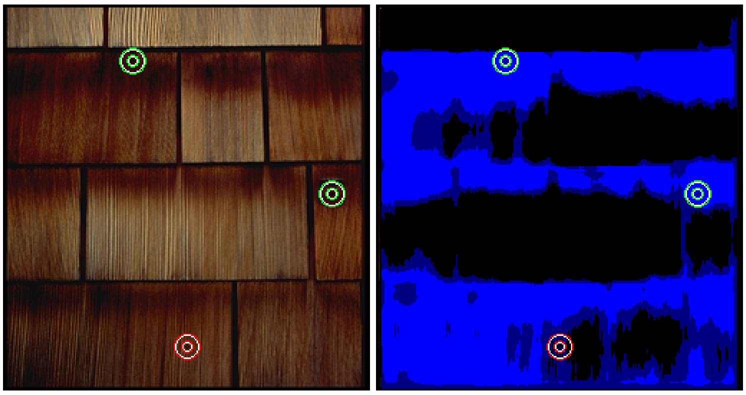


Figure 96: Using a three point Boolean similarity expression to construct a synthesis mask.

Chapter Summary

In this chapter, we introduced an addition to our system of interactive texture editing that utilizes self-similarity to replicate texture cloning operations globally over an image. The discussion began with an alteration of the self-similarity based painting system which allows the user to paint with texture rather than a solid RGB colour.

The flexibility of the system was then improved by combining replicated cloning with texture-by-numbers synthesis. With this use of texture reordering, it was shown how the user is able to tailor the spatial arrangement of the cloning image to better match the target texture. Furthermore, a semi-automated masking tool was presented which assists the user with the construction of the necessary texture-by-numbers masks. Our discussion then considered the means by which interactivity can be maintained when using texture-by-numbers in combination with replicated cloning; namely, through the use of incremental rendering towards to the final result.

In the final section the interface components which are specific to self-similarity based cloning were examined.

GLOSSARY

Image Analogies – Image Analogies is a framework for processing images by example [Hertzmann01]. Rather than designing individual filters, the system attempts to automatically learn filters from training data. Applications include super-resolution synthesis, texture transfer, artistic filters, texture-by-numbers and image colorization.

Cloning Stamp Tool – The cloning stamp tool, found in many image editing packages, allows the user to copy samples of an image and subsequently apply them over areas of another image or areas of the same image. Each stroke of the tool paints on more of the sample.

Image Masks – With image masks the user is able to isolate and protect areas of an image as colour changes, filters, or other effects are applied to the rest of the image. In addition, masking layers are frequently used to save and reuse time-consuming selections.

Secondary Texture – Surfaces are often textured with multiple layers in order to add visual complexity to a scene. A given surface will frequently host secondary textures which partially cover the primary texture. A typical example of this is the partial covering of moss over a bark texture.

Chapter 5

Self-Similarity Based Texture Warping

Chapter Structure

In this chapter we extend the self-similarity editing framework by allowing the user to alter the shape of texture elements rather than their colour values. The notion of replicated texture warping is developed wherein the degree of local neighbourhood similarity is used as a measure of local area expansion. All pixels with neighbourhoods similar to the user-selected pixel(s) are made to expand locally; those that are dissimilar are made to contract.

We begin the chapter by introducing the general issues involved and then proceed to a detailed discussion of how a field of scalar similarity values can be converted into a 2D warp over a mesh grid. A collection of results are presented within a discussion of the approach's strengths and weaknesses. We then improve these results by re-synthesizing the lost high frequency detail into expanded areas. Finally, we briefly examine necessary additions to the user interface.

1. Altering the Shape of Texture Elements

In previous chapters we exploited the inherent self-similarity of texture images to globally alter the colour of texture elements. In this chapter we will deviate from this by altering the shape of texture elements instead. The goal here is to allow the user to expand all of the texture elements at once. As before, the user positions similarity point(s) to guide the editing operation. Areas that are deemed similar to the user-selected (positive) similarity point(s) are expanded locally.

When the user positions the similarity point(s) the system generates a scalar field of similarity values. At each point in this scalar similarity field the calculated similarity value is stored as a value between 0.0 and 1.0. Until this stage we had referred to the scalar similarity level as the level of applied opacity. For example, in the left-hand image of Figure 97 the user has positioned a single positive similarity point over a texture of chrysanthemums. This generates a grid of scalar similarity values, shown in the centre image as a height field, with heights in the range of 0.0 to 1.0. These self-similarity values are now used to interactively drive area magnification. If in painting mode the pixel would have received 75% opacity, now the local area instead increases by 75%. To accomplish this we need a means of converting the grid of scalar similarity values (Figure 97, middle) into a 2D image warp (Figure 97, right).

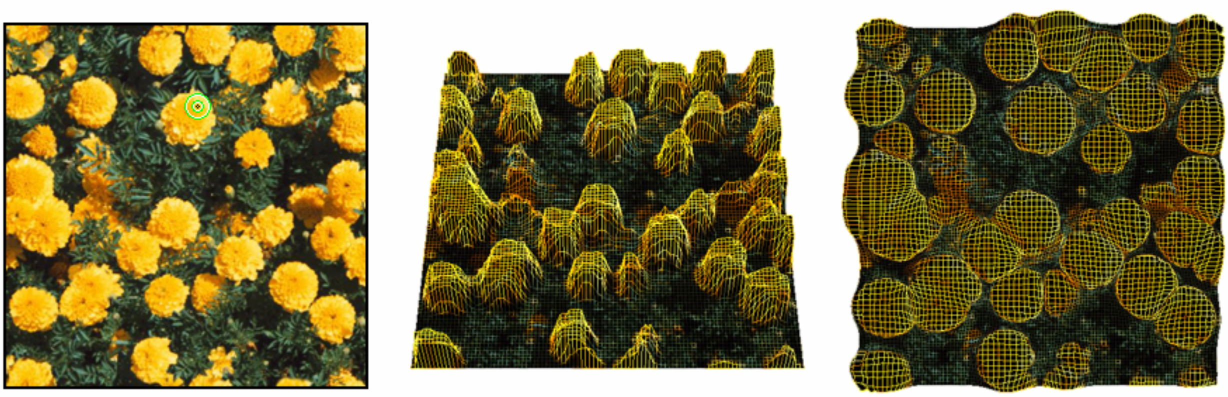


Figure 97: From scalar similarity values (middle) to 2D texture warp (right).

1.1 Integrating the Scalar Similarity Field

In addition to published research [Arad94, Lee95, Glasbey98, Froumentin00, Labrosse01, Milliron02] there are a number of commercially available image warping tools [Scansoft03]. Many of these image warping methods focus on morphing one image into another [Beier92, Wolberg98]. The common aspect of all of these methods is that they have a target to work towards, whether it is another image (see Figure 98) or a set of key points or curves (see Figure 99). In both cases, the system is provided with a set of hard constraints to which an image must be adapted.

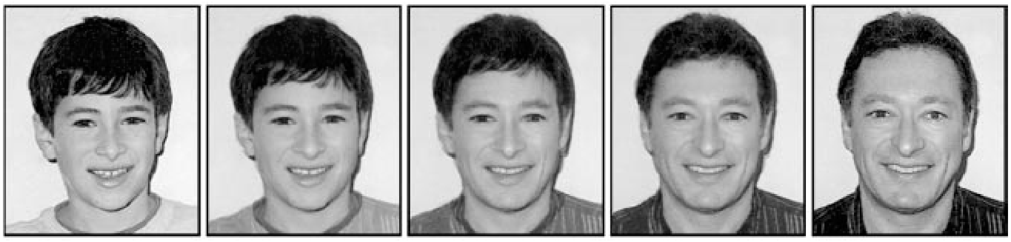


Figure 98: Morphing from one image to another [Wolberg98].

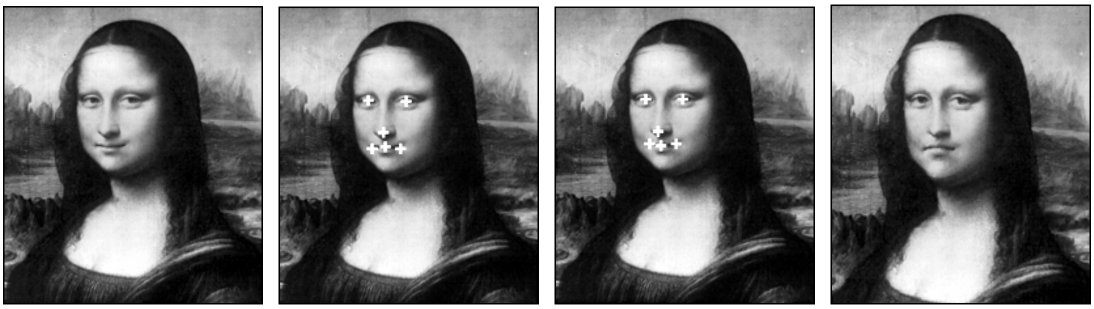


Figure 99: Image warping using six key points [Arad94].

Our problem is rather different; in the case of replicated warping the constraints are not as specific. Self-similarity based texture warping uses neighbourhood similarity as a measure of local area expansion and the question becomes how to convert scalar similarity values derived from neighbourhood distances into 2D area expansions. Here we are not given correspondences between the original image and the warped image, nor can we infer them using methods of computer vision. Our constraints are far vaguer in that they dictate that a set of points must be expanded locally. In two dimensions this is an under-constrained problem: there are an infinite number of ways that a grid of scalar values can be converted into 2D area expansions. We therefore require a method that performs this in a 'reasonable' fashion.

To accomplish this we borrow the interactive image-warping scheme of Keahey *et al.* entitled non-linear magnification fields [Keahey97]. Their work provides a means of converting between scalar magnification fields and 2D transformations (see Figure 100). The approach is a numerical and iterative process for converting scalar magnification fields into non-occluding transformations over 2D grids. In their work they allow the magnification field to be either user-driven (where the user manually magnifies areas with a magnification brush) or data-driven (as a pre-computed field of magnification values). Our case contains something of both as the user is providing guidance to the magnification, but the data contained in the image itself is used to amplify the user's actions globally.

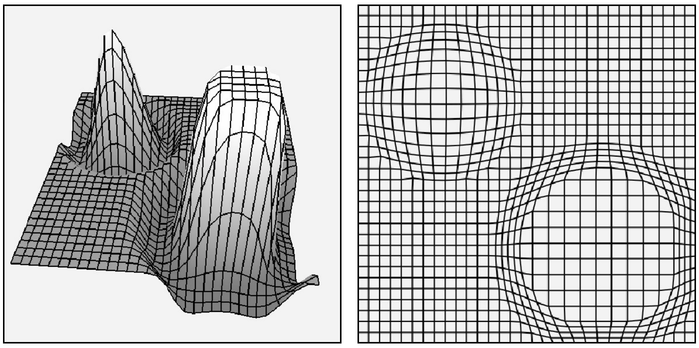


Figure 100: A scalar magnification field (left) and its corresponding 2D transformation warp (right).

We will begin with a number of clarifying definitions. Like the opacity levels described in the previous chapters, the magnification field, $M(x,y)$, is generated by the neighbourhood similarity computations:

$$M(x,y)=\min\left(\max\left(\sum_{i=1}^{n}\max\left(s\times\frac{\left(t-d_{SSD}\left(x,y,A_i\right)\right)}{t},0\right)-\sum_{j=1}^{m}\max\left(s\times\frac{\left(t-d_{SSD}\left(x,y,B_j\right)\right)}{t},0\right),\ 0\right),\ 1\right)$$

where:

t is the distance threshold set by the user's slider,
s is the global strength, ranging from 0.0 to 1.0,
n is the number of positive similarity points, A_i, and
m is the number of negative similarity points, B_j.

A maximum expansion value $M(x,y)=1.0$ will double the local area around (x,y). Although at its maximum the warp can only double the local area, this is not a restriction, as multiple cycles of committed warping operations can be performed if an arbitrarily high amount of expansion is required.

The magnification field, $M(x,y)$, represents the magnification values (Figure 100, left) which are implicit in the desired 2D grid transformation function, $T(x,y)$, shown in the right image of Figure 100. This transformation function $T(x,y)=(x',y')$ directly stretches and compresses the 2D space. The magnification function is essentially the derivative, $M(x,y)=(\partial T(x,y)/\partial x)\times(\partial T(x,y)/\partial y)$, of the desired 2D transformation function $T(x,y)$.

Although we wish to generate the transformation function, $T(x,y)$, from the given magnification field, $M(x,y)$, it is instructive to first consider this simpler case of deriving $M(x,y)$ from $T(x,y)$. This involves numerically computing an approximate derivative of T. For this we need an area function $a(x,y)$ computed for each node in T. This local area is approximated as the convex hull of the 4-connected neighbours $\{T(i+1,j),T(i-1,j),T(i,j+1),T(i,j-1)\}$. It is, of course, possible to improve the accuracy by finding each of the 4-connected neighbours' areas and summing, but that does not appear to give noticeably better results [Keahey97]. We then define C_a as the constant area associated with any $T(x,y)$ in the untransformed sampling grid. The magnification of a point $T(x,y)$ is then simply given by $M(x,y)=a(x,y)/C_a$. We will make use of this derivative approximation at a later stage as the ease of converting from the transformation function to the magnification field will facilitate the conversion in the opposite direction.

Deriving the magnification field from the transformation function is a straight forward computation. However, we need to compute the reverse; we need to derive the 2D transformation function from the magnification field. Since the $M(x, y)$ is essentially the derivative of $T(x, y)$, we therefore need to integrate the magnification grid values in order to construct the transformation grid. The fundamental problem is that there are unlimited possibilities: when expanding around a given point, we can always extend one direction at the expense of another. Since the problem is under-constrained, we need to impose criteria on the solution method, which are as follows:

1. Efficiency: $T(x, y)$ needs to be computed at faster than interactively rates.
2. No self-occlusion: $T(x, y)$ must be at least C^0 continuous and order preserving.
3. The transformation should be symmetric and centred around magnification maxima.
4. The total area before and after the warping operation should be equal.

To address these criteria a numerical algorithm is used to approximate the integration of M, yielding an estimate, T_c, of the transformation function, T, at each iteration. At iteration zero, T_0 is initialized to the identity transformation. At each subsequent stage, the corresponding approximate magnification function, M_c, is directly computed from T_c, allowing the resultant error, $M_e = M - M_c$, to be directly calculated. T_c is then further modified on a vertex-by-vertex basis. Effectively, the neighbouring vertices are moved outwards in T_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.

As so far described, Keahey's solution is as yet imprecise and a number of details need to be specified which we will now enumerate. Firstly, the error metric determines which neighbours of $T(x, y)$ are to be displaced at each iteration. As our error metric is 4-connected, we only displace those 4-connected neighbours. When altering the position of neighbouring points, an additional constraint is added to avoid self-collusion. The transformations are restricted at each iteration so that given $T(x_1, y_1) = (x_1', y_1')$ and $T(x_2, y_2) = (x_2', y_2')$ then $x_1 < x_2$ always implies that $x_1' < x_2'$.

The algorithm also benefits from various optimizations detailed by Keahey. The algorithm converges faster if we multiply the error $M_e(x, y)$ by the specified magnification $M(x, y)$ so that regions of higher magnification are more strongly weighted. This causes the neighbourhood displacements to be increased for that node at each iteration. Moreover, if we weight the displacement by the distance to its neighbours it further increases the rate of convergence. Displacements are weighted so that closer neighbours are pushed a greater distance than farther neighbours.

We, like Keahey, also use a refinement coefficient C_r which globally affects the integration step. C_r ranges between 0.0 and 1.0 where $C_r = 0.0$ means no displacement and $C_r = 1.0$ means full displacement (with enforced ordering). In all examples shown in this chapter we consistently use a coefficient of $C_r = 0.1$. With these enhancements we can, for example, compute the displacement amount that is applied to a node $T_C(x+1, y) = (x_1', y_1')$ which neighbours $T_C(x, y) = (x', y')$ as:

$$\frac{1}{4}M_e(x,y)\times M(x,y)\times C_r\times\frac{(y_2'-y')+(x'-x_3')+(y'-y_4')}{((x_1'-x')+(y_2'-y')+(x'-x_3')+(y'-y_4'))}$$

where $T_C(x,y+1)=(x_2',y_2')$, $T_C(x-1,y)=(x_3',y_3')$ and $T_C(x,y-1)=(x_4',y_4')$ are the 3 remaining 4-connected neighbours. We also require a measure of convergence in order to determine the number of iterations that are necessary. The measure used is the root mean squared error:

$$RMSE=\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}M_e(i,j)^2}$$

Table 10 shows the convergence performance of the algorithm for the flowers example shown in Figure 101 using the $RMSE$ measure. This typical example shows the system reaching convergence at about 160 iterations. The reason why the $RMSE$ never reaches 0 is that the system is forced to compress some areas in order to allow other areas to expand which contributes to the final error. The system reaches a compromise favouring the expansion of areas with higher magnification. The important thing to note is the convergence of the error and not its absolute value.

Iterations	*RMSE*
1	43.52715
20	37.96724
40	34.99347
60	32.87227
80	31.26793
100	30.02649
120	29.02798
140	28.21058
160	26.95499
180	26.63973
200	26.67038
220	26.60548

Table 10: Convergence of warping operation.

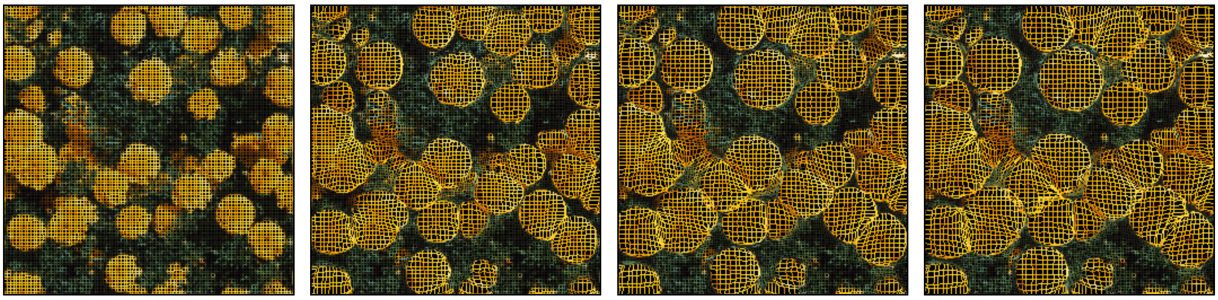


Figure 101: Iterations 0, 40, 80, 120 of the warping operation.

In practise we do not want the system to exhibit variable response rates within an image, so we fix the number of iterations at the moderately conservative level of 180. The value of 180 has been empirically determined from the selection of examples shown in the following results section.

Our image warping algorithm is implemented with hardware texture mapping by defining a correspondence between a uniform polygonal mesh (representing the original image) and a warped mesh (representing the warped image). This relegates the image interpolation operations to the graphics hardware thereby placing almost no burden on the CPU. For the warping examples in this chapter, 256×256 images are textured over a warped 2D mesh. We find that for an N×N image, the mesh itself need only be (N/2)×(N/2) for quality results. Table 11 shows the frame rates obtained for replicated warping as a function of image size and number of similarity points used (running on an off-the-shelf 2.66 GHz PC with an Nvidia GeForce4 MX 440 graphics card). The frame rate was calculated by averaging the number of frames over 60 seconds of operation for 20 randomly chosen images.

		Number of Similarity Points		
Image Size	**Mesh Size**	1	2	3
128×128	64×64	34.31	29.88	26.93
256×256	128×128	6.66	5.69	5.24
512×512	256×256	0.52	0.44	0.42

Table 11: Replicated warping with PCA compressed multi-resolution neighbourhoods: average number of frames per second as a function of image size and number of similarity points used.

2. Results

We will now examine a collection of replicated warping results. In all of the examples from Figure 102 to Figure 107 the middle row houses the original textures. The top and bottom rows contain images with opposing expanded and contracted areas. As most of the examples are self-explanatory, we will only focus on a few instructive cases.

In the first examples shown in Figure 102 we see the growing and shrinking of flowers. The chrysanthemums in the left-hand column appear to be shrinking as we move down the page. Daisy petals have been made to expand in the top-right image. In the bottom-right image the daisy centres have been expanded, deforming the overall shape to approach that of a sunflower. There are three points to be made about these particular examples.

The first is that in the top-left image, there is a loss of high frequency detail where the flower heads have been expanded. There is a similar loss of detail in the centres of the daisies in the bottom-right image. The original details have been made to occupy a wider area, causing blurring artefacts. We will address this important issue in the next section.

The second issue is that when touching areas exist in the original texture and these areas are subjected to contraction, an elongated threading can form between the contracted areas. An example of this can be seen in the bottom-left image of Figure 102. The reader will note the instances where two flower heads are touching in the original middle-left image. In the bottom-left image these abutting flowers are now connected with a thin threading of yellow. It is as if the flowers which were close in the original image are now exerting a gravitational pull upon each other. Although we do not have a solution to this problem, we would argue that this not a prominent artefact and it is only moderately noticeable in certain cases.

The third point is that these examples may give the impression that the warping operation produces only circular expansions. However, if we will look at the left-hand example of Figure 103, we see that this is not true in general. In this case, the giraffe's patches have roughly maintained their Voronoi-like shape in all stages of the warping operation. The shape of the deformation primarily depends on the shape of the original area being expanded.

Finally, for the sake of comedic value, Figure 107 shows screen captures of a human face being subjected to replicated warping. In the top image the user has moved the selection point over a portion of hair. All hair-like regions of the image have therefore been expanded simultaneously, producing an Elvis-like caricature. In the bottom image the skin has been expanded at the expense of the hair and eyes. Although our editing system is primarily targeted towards 2D textures that obey the stationarity criteria, as can be seen from this example, the system can be applied to general images in certain cases to achieve particular effects. We believe that the warping performs well in this instance due to the fact that the lighting conditions are fairly even and because faces are roughly symmetric.

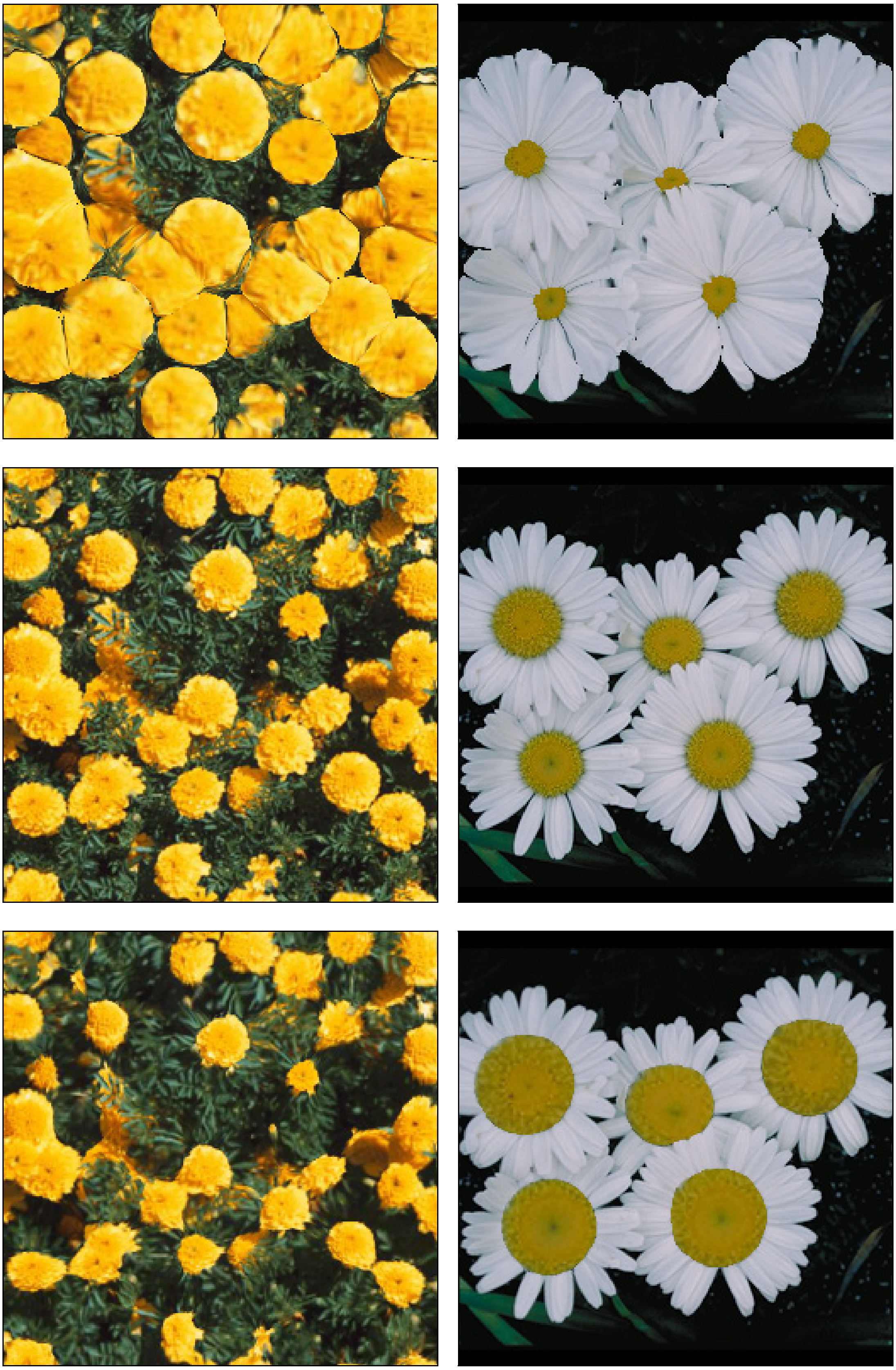


Figure 102: Replicated warping of flowers.

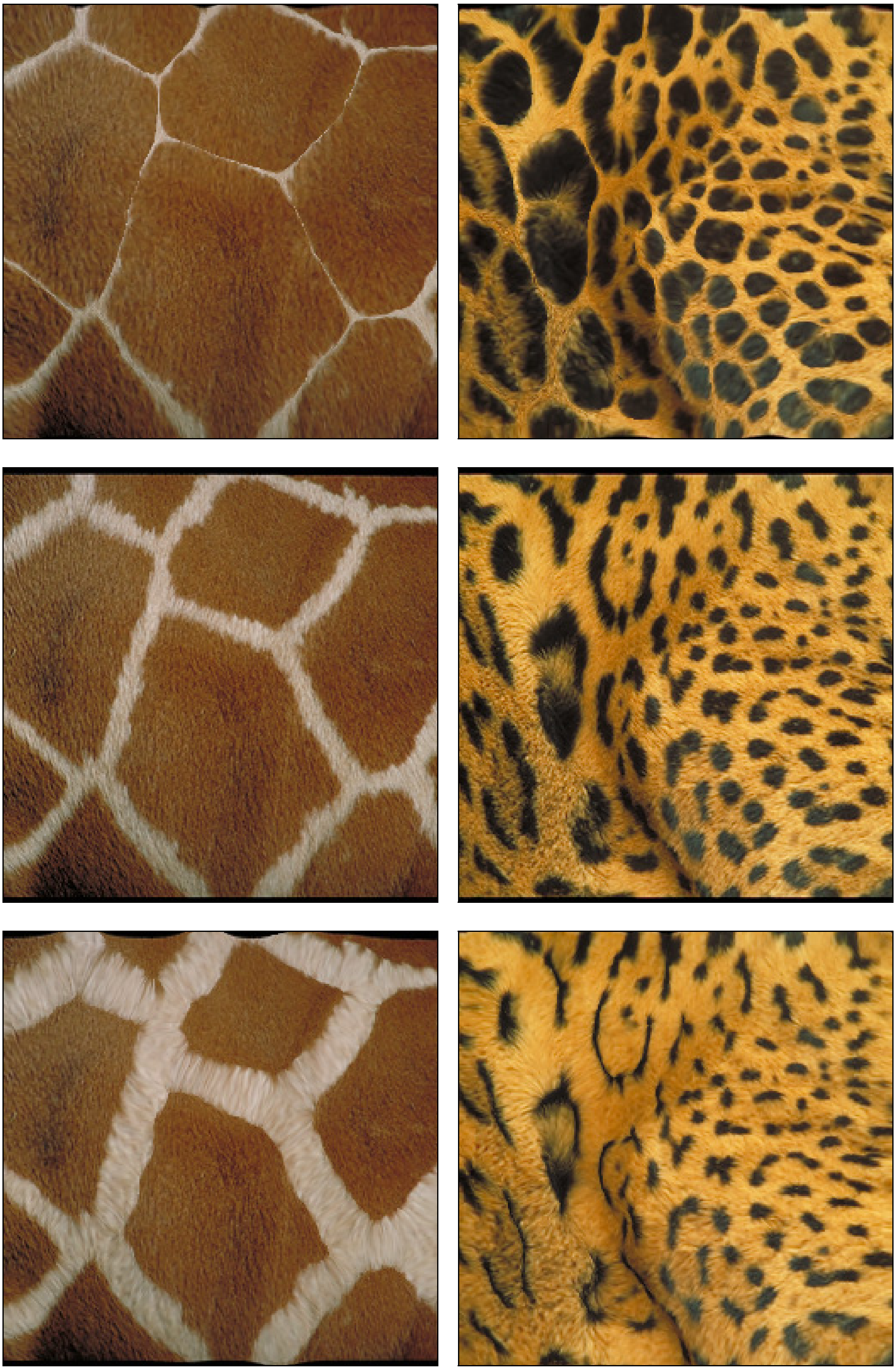


Figure 103: Replicated warping of animal furs.

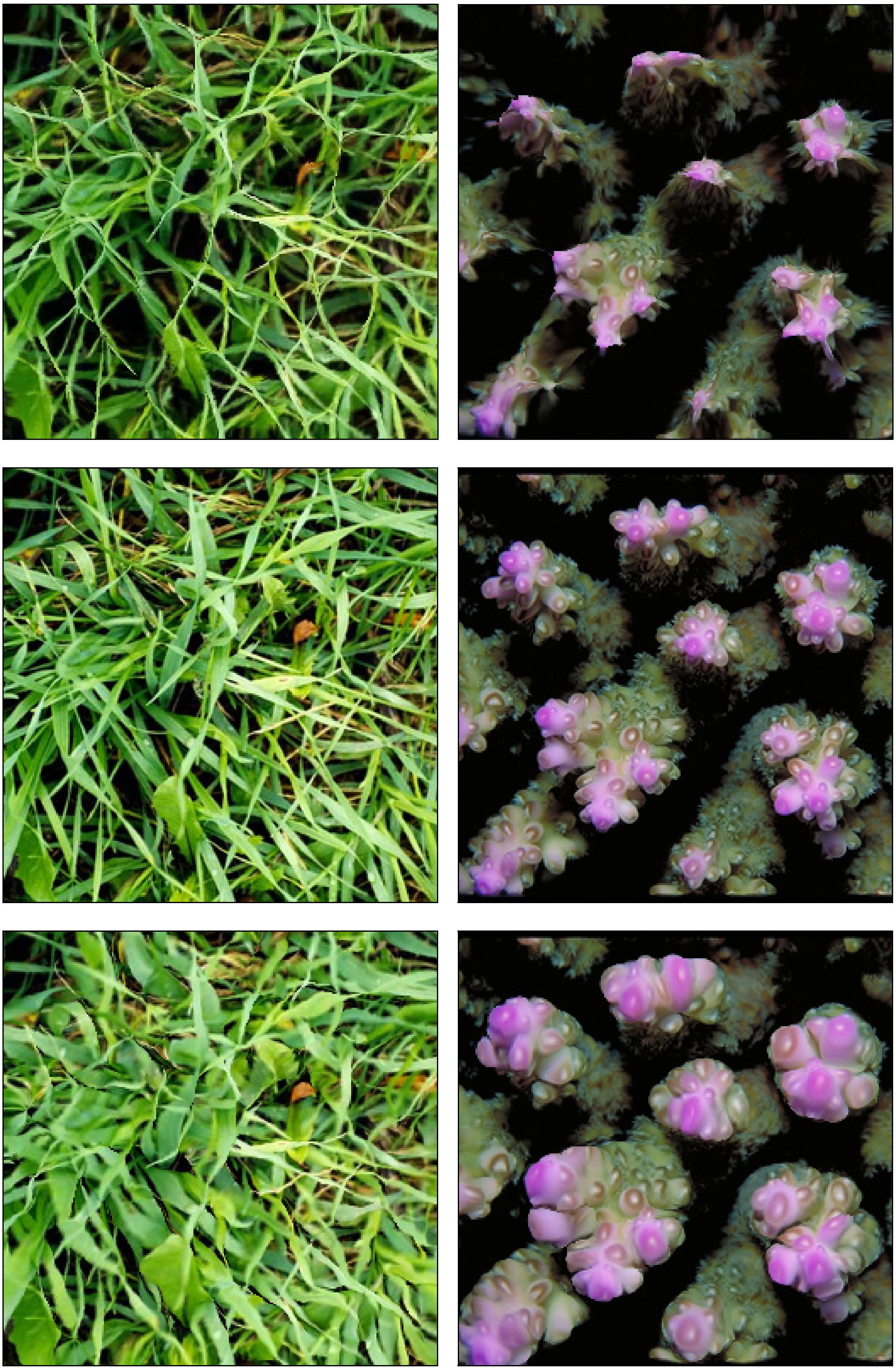


Figure 104: Replicated warping leaves (left) and underwater vegetation (right).

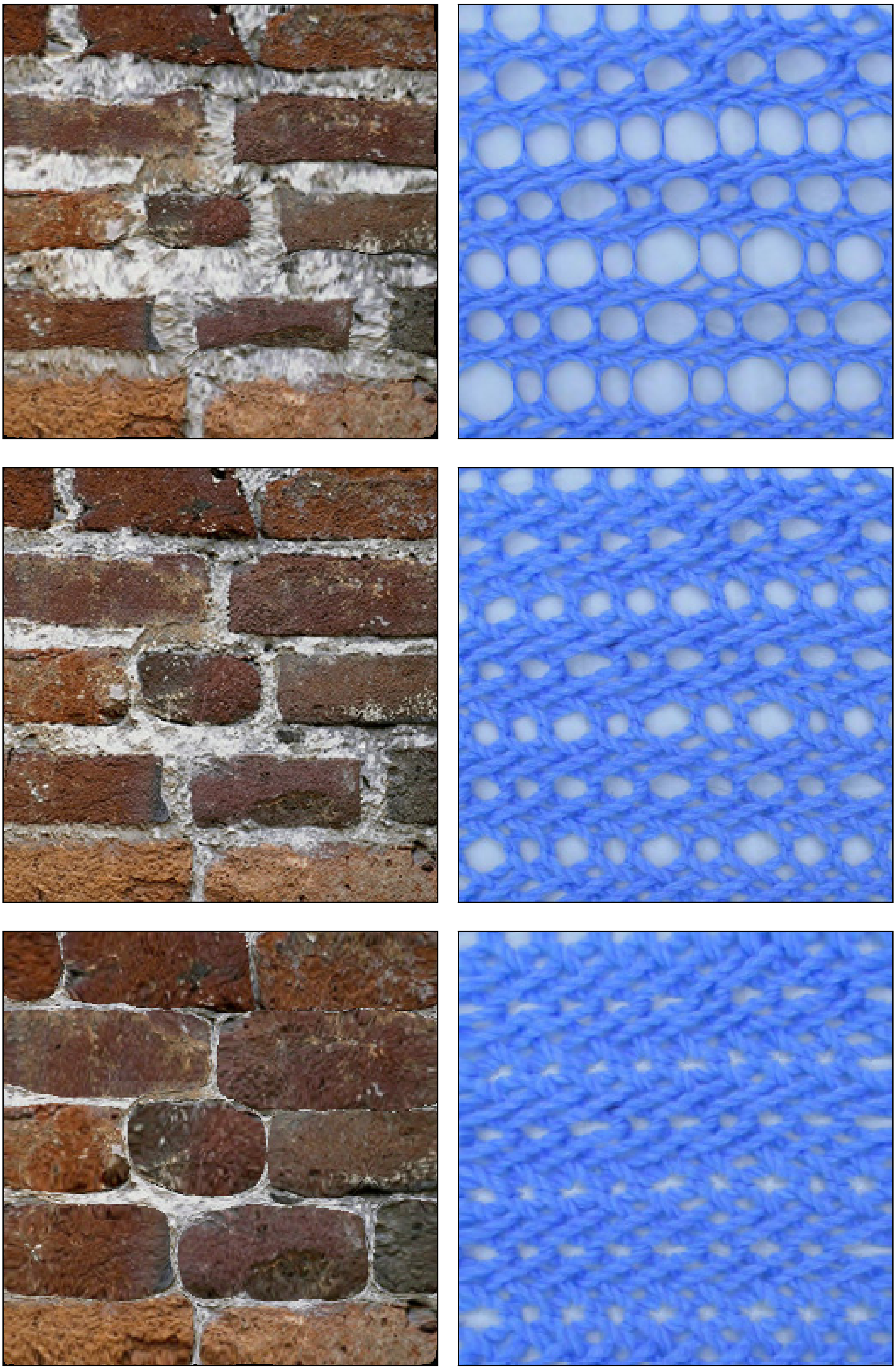


Figure 105: Replicated warping of bricks (left) and knit work (right).

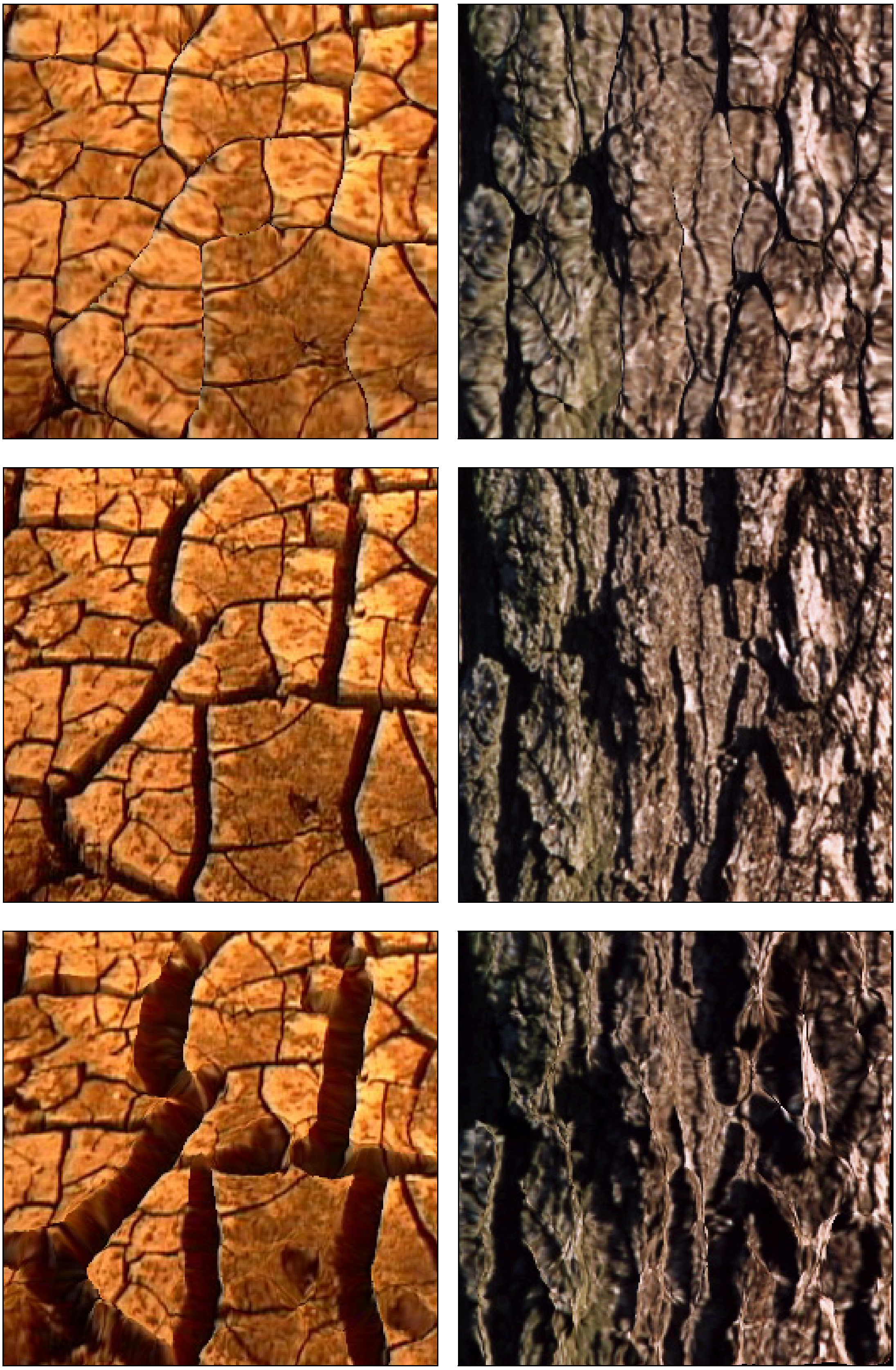


Figure 106: Replicated warping of cracks (left) and bark (right).

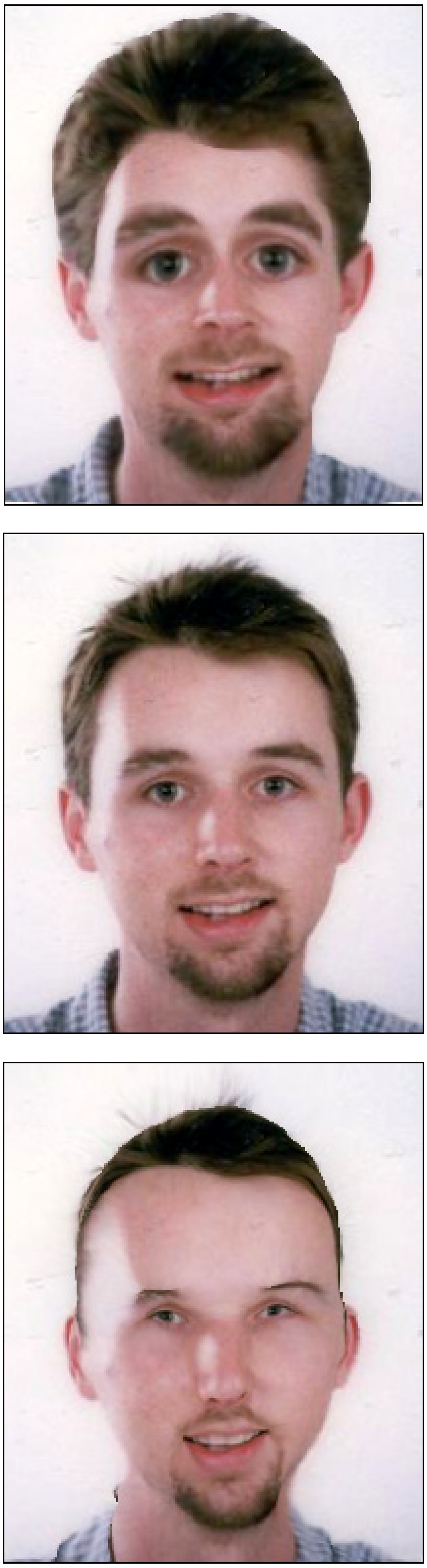


Figure 107: Replicated warping of a human face. We leave it as an exercise for the reader to guess which the original image is.

3. Re-synthesis of High Frequency Detail

Raster images lack resolution independence which means that pixel-based images cannot be enlarged significantly beyond the resolution they were originally sampled at without a loss of visual quality. The use of bilinear or bicubic interpolation [Keys81] methods can mitigate the aliasing of frequencies but this does not fundamentally address the loss of detail. Likewise, our warped textures can suffer a loss of high frequency detail in expanded areas. Figure 108 shows a number of examples where the loss of detail is evident.

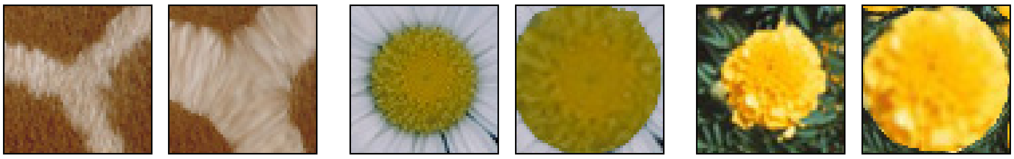


Figure 108: Loss of high frequency details. The left-hand image in each pair is taken from the original texture with the warped version to its right.

Recent methods have emerged which are able to introduce detail into low-resolution images. Called super-resolution synthesis, it is a form of data amplification that takes as input the low-resolution image and produces a detail-enhanced output. The details are typically sourced from a database of similar images [Freeman02] or from multiple low-resolution images taken from the same scene [Huang84]. In our case, we will use super-resolution synthesis in a novel way by sourcing the details from the original, undistorted image. With this novel use of super-resolution synthesis, we overcome the loss of high-frequency details by using the newly warped texture as a constraining image for super-resolution synthesis.

This returns us again to texture synthesis and, specifically, to the Image Analogies synthesis framework of Hertzmann *et al.* [Hertzmann01]. Recall that in the framework of Image Analogies, the texture-by-numbers re-synthesis of an image states that 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:

$$A : A' :: B : B'$$

As Hertzmann *et al.* describe, this framework can be adapted to perform super-resolution synthesis if we treat B as the unfiltered target image and set *both* A and A' to be the original, undistorted texture. Setting both A and A' to be the original texture may seem unintuitive until we again consider exactly how the synthesis algorithm operates.

As before, the Image Analogies algorithm proceeds by synthesizing each new pixel in the output image, B', in scan-line order by finding pixels with matching local neighbourhoods in the original texture, A'. When selecting the next synthesized pixel, the distance calculation used to determine the new colour value at a given pixel *q* becomes the sum of square differences between each of the RGB values in the concatenated neighbourhoods of the images (A & A') and the images (B & B'). The output colour value at *q* is set to the colour value of pixel *p* in input texture with the lowest (concatenated) neighbourhood distance from *q*.

By comparing the local neighbourhoods in the original undistorted texture, A', against those of the output texture, B' (which we are synthesizing), the properties on the input texture are maintained during synthesis; by simultaneously comparing the local neighbourhoods in the original undistorted texture, A, against those of the warped texture, B, we force the system to obey the desired spatial configuration of the newly warped texture. In effect, the original texture and the warped texture are used as the input and output masks respectively during synthesis.

Final results of applying super-resolution synthesis to warped textures are shown in Figure 109. In each row the image to the far left is the original texture and the middle image shows the warped texture. The result of re-introducing high frequency details into the warped images using super-resolution synthesis is shown to the right. In all cases of super-resolution synthesis we use the original N×N image to re-synthesize the newly warped N×N image. The computation timings for the super-resolution synthesis of warped textures are therefore the same as those detailed in Table 8.

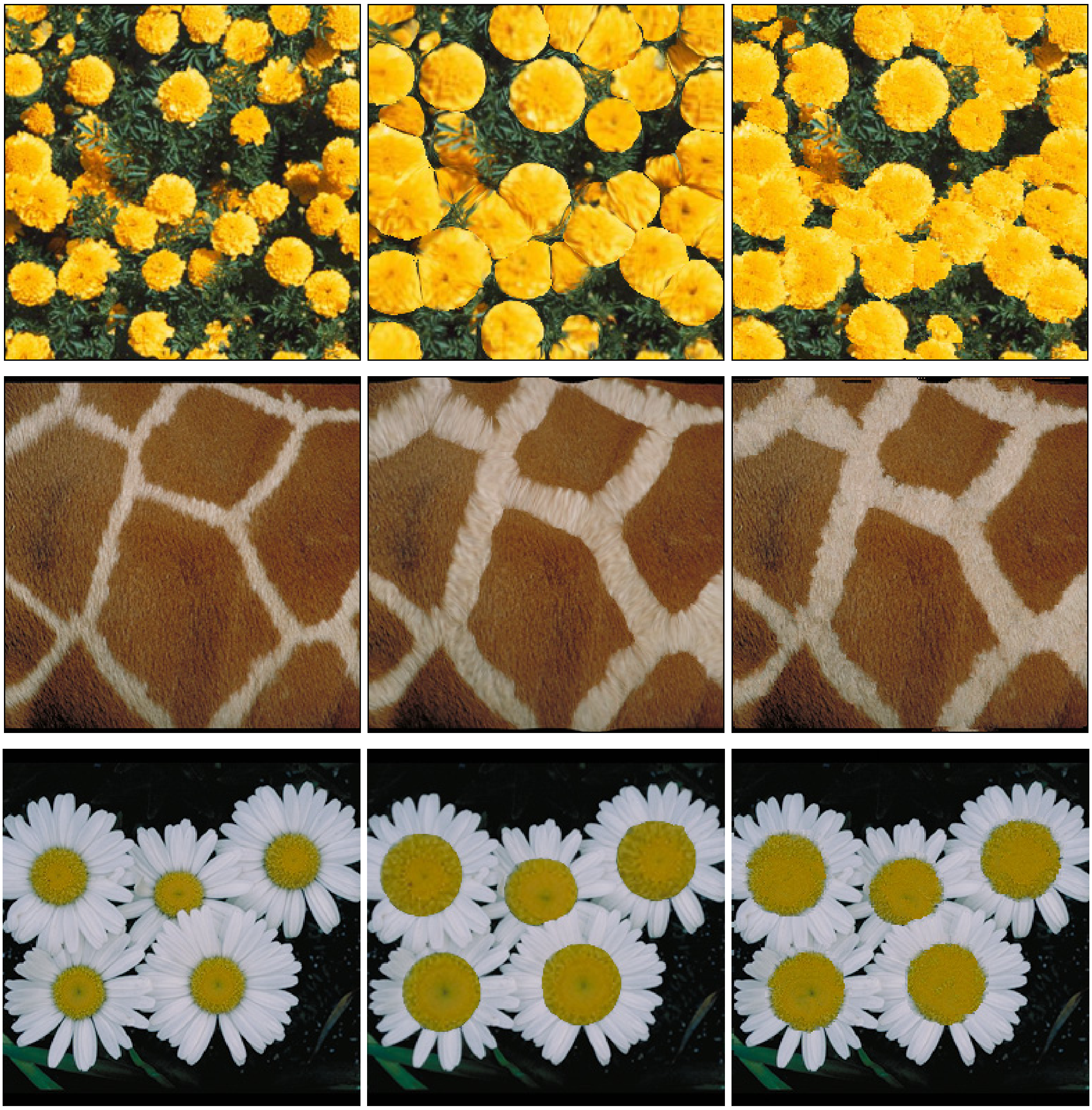


Figure 109: Re-introducing high frequency details into the warped images using super-resolution synthesis.

The loss of detail is, of course, not an issue for areas that have not been subjected to any change. Figure 110 shows a number of square areas taken from the same images sampled in Figure 108 but where the loss of detail is not as significant. The degree of high frequency detail that originally exists in the expanded area is also an important consideration as to whether there will be a smoothing problem in the expanded areas. The more the original area approaches a smooth and continuous transition, the less of a problem will result from the expansion.

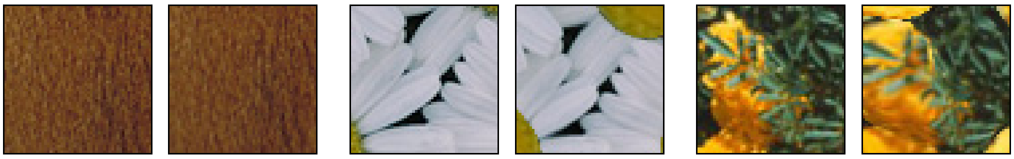


Figure 110: Minimal loss of high frequency details. The left-hand image in each pair is the original with the warped version to its right.

Ideally, we would like to re-introduce high frequency details only into those areas that have suffered blurring due to expansion as it is wise to leave unchanged those areas that have not been affected by warping. In practice, this is the default result of the super-resolution synthesis algorithm. When the synthesis algorithm encounters an unwarped area, it is able to find exact matches with the original image. The synthesis procedure replaces an unaffected pixel with a copy of itself and we therefore do not require any special processing to handle this case. As an avenue of future work, this might be exploited to improve the efficiency of re-synthesis by only re-synthesizing areas that have expanded significantly. Furthermore, since we know roughly where an expanded area had previously existed in the original texture, it might also be possible to restrict the Image Analogies algorithm to only search in that original area for re-synthesis details.

It is also worth noting that in the course of developing the self-similarity based editing system an alternate method of super-resolution synthesis was tested for warped textures. In addition to the Image Analogies method of Hertzmann *et al.*, we also conducted re-synthesis experiments using Harrison's entropy-based super-resolution synthesis [Harrison01]. A number of results using Harrison's method are shown in Figure 111. As these images clearly show, Harrison's method has difficulty maintaining coherence within texture regions. Another important consideration is that Harrison's method does not offer multi-resolution feedback and requires more time to compute the final result (an average of 16 minutes for a 256×256 texture). Although Harrison's method is an innovative approach which works well for other tasks, for the purpose of re-synthesizing warped textures, it does not perform as well or as quickly. For these reasons we will not discuss the details of Harrison's method in depth; we instead refer the reader to [Harrison01] for further information.

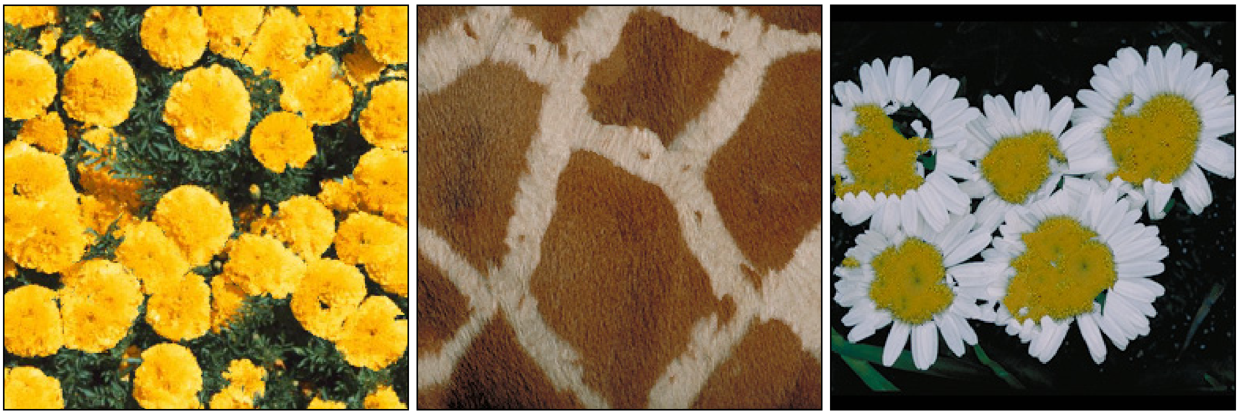
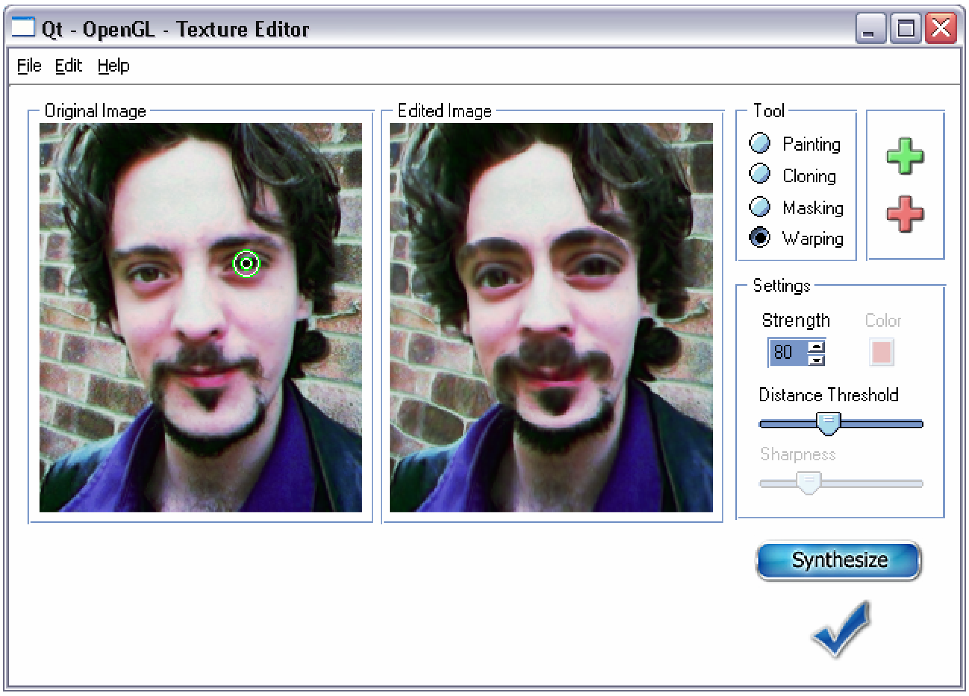


Figure 111: Attempts at super-resolution synthesis using Harrison's method [Harrison01].

4. The Warping Interface

When using our system to warp textures, the interaction sequence follows a similar pattern to that of replicated painting. However, the warping interface (shown in Figure 112) is slightly more constrained. The user can only control the position of the selection point from within the original image because the shape of the warped image is changing as the mouse moves. To allow the user to operate directly over the warping image would be disquieting for the user, as the image would be swimming beneath the mouse pointer.



○ Painting
○ Cloning
○ Masking
● Warping

Strength 80

Distance Threshold

Sharpness

Figure 112: Warping editor.

As with replicated cloning, the value of 'Colour' is undefined and 'Sharpness' is not relevant. The 'Synthesize' button is available to perform super-resolution synthesis on the warped image at any time. Other minor points of note are that there is an additional 'Warping' radio button and what was called 'Opacity' in the painting/cloning interface has now been relabelled 'Strength'. The warping specific parameters such as iteration step, C_r, and number of iterations are held constant over all examples, so there is no need to offer these as user controls.

With regards to comparing replicated warping with existing product interfaces, there are commercial tools which allow the user to locally warp an image. The most notable of such tools is Scansoft's SuperGoo® (formerly known as Kai's Power Goo™) whose interface is shown in Figure 113 [Scansoft03]. The metaphor for the SuperGoo® interface is that the image is suspended in a viscous liquid. When the user pushes and pulls the liquid, it distorts the image locally. There are a number of interactive variations on this theme which include the *smear*, *nudge* and *grow/shrink* brushes.

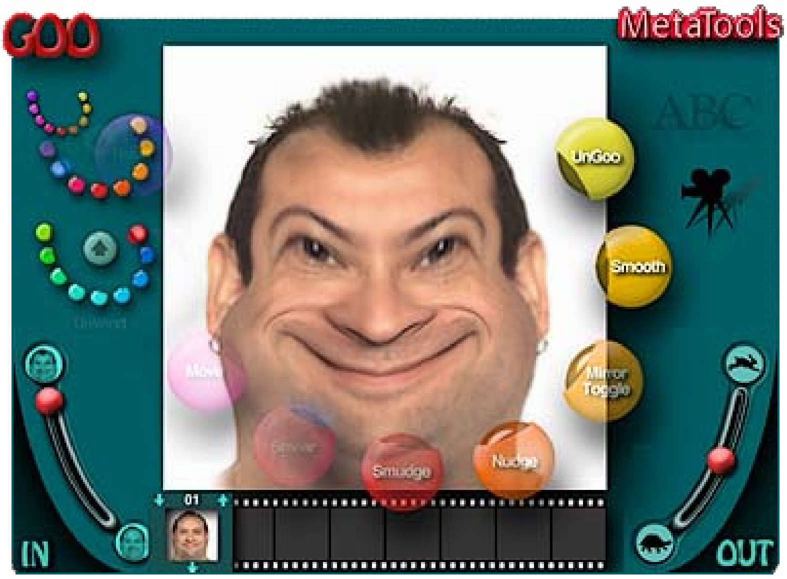


Figure 113: The SuperGoo® interface.

The most relevant of these tools is the *grow/shrink* brush which locally grows and shrinks the area under the mouse in a symmetric pattern. Analogous to our comparison with Photoshop's® painting tools, by using the *grow/shrink* brush in SuperGoo® the user could roughly mimic the results of our replicated warping tool. However, from an interaction point of view, it would again require the same degree of repeated manual intervention at every texture element. Furthermore, they do not offer a facility to re-synthesize detail into the expanded areas.

Chapter Summary

In this chapter, we extended the texture editing system to include spatial deformations of the texture elements. With the introduction of the replicated warping tool, the user is able to globally expand all texture elements simultaneously.

The chapter began with a brief discussion of general image warping methods followed by specific account of how a field of scalar similarity values can be converted into a 2D warp over a mesh grid. A number of results were then considered within an examination of the tool's properties. The discussion then turned to the improvement of the results by re-synthesizing high frequency details into expanded areas. In the final section, we reviewed the minor changes to the user interface that were required.

GLOSSARY

Image Morphing – Image metamorphosis, or morphing for short, is commonly referred to as the gradual transformation of one digital image to the other. The entire process consists of warping two images so that they have the same shape and cross dissolving the warped images. As the morphing proceeds, the first image is gradually distorted and is faded out, while the second image starts is gradually un-distorted and is faded in.

Super-resolution Synthesis – A method to generate higher resolution renderings of pixel-based images. Super-resolution synthesis artificially introduces higher frequency texture information into an image while respecting the existing spatial arrangement of the existing lower frequency content.

Transformation Function – A distorting function which directly stretches and compresses the underlying 2D space.

Magnification Field – The magnification function represents the magnification values which are implicit in the desired transformation function. The magnification function is essentially the derivative of the transformation function.

Chapter 6

Conclusions & Future Work

Chapter Structure

This final chapter provides a broad review of the contents of this thesis. In the first section, the novel contributions to research are outlined and the strengths of the self-similarity editing system are catalogued. This is followed by a discussion of the system's limitations with respect to special cases of texture and to general images. In the last section we sketch possible future research directions which include the editing of 3D geometry, video editing and alternate 2D image editing operations.

1. Summary of Achievements

Existing commercial editing systems typically force the user to consider images as an array of RGB pixels. But to the average user this is a foreign representation: an image's value is its content, not its storage format. With our system the user is able to manipulate texture images at a higher semantic level, allowing editing at a level much closer to how users perceive texture images. Without reference to pixels the user can, for example, simultaneously paint colour onto all the shingles or stones on a wall. This provides a better match between the user's innate understanding of textures as repetitive visual entities and the operation of the interface tools.

Our system amplifies the user's input by replicating painting, cloning and warping operations over a texture. We now review the novel contributions developed within each type of editing. Each of these contributions have already been published; the publications cover the basic editing system [Brooks02] as well as the more advanced material [Brooks03].

> **Painting** – Our system of interactive texture editing was first introduced in the context of replicated painting wherein an initial measure of similarity between pixels was presented. Building upon this concept, the flexibility and the efficiency of the system were both improved with the introduction of multi-point Boolean similarity expressions and the use of a compressed multi-scaled distance measure. The similarity measure was enhanced further with regards to its perceptual validity. Finally, we improved the system's sharpness of response by allowing the user to control the relative contribution of Gaussian neighbourhood versus steerable wavelet responses in the similarity computations.
>
> **Cloning** – Having established the system of replicated painting, the editing system was extended to replicate texture cloning operations globally over an image. We modified the framework of replicated editing to the task of cloning one texture onto another, thereby allowing the user to paint with texture rather than a solid RGB value. Next, the flexibility of this new cloning system was significantly improved by combining replicated cloning with the use of texture-by-numbers synthesis. To this end, we adapted the Image Analogies framework of Hertzmann *et al.* by introducing a semi-automatic tool for the construction of texture-by-numbers masks [Hertzmann01].
>
> **Warping** – With the introduction of this third incarnation of replicated texture editing, the self-similarity editing framework allows the user to alter the shape of texture elements as well as their colour values. The notion of replicated texture warping was developed in which the degree of local neighbourhood similarity is used as a measure of local area expansion. Borrowing the interactive warping scheme of Keahey, we showed how a field of scalar similarity values can be converted into a 2D warp over a mesh grid [Keahey97]. Our final contribution introduced the idea of using details found in the original image to improve the results of an image warping operation. The warping results were enhanced by re-synthesizing the lost high frequency detail into expanded areas.

It is important to note that some of the more complex operations that self-similarity editing can perform simply cannot be achieved using off-the-shelf software by anyone who is not a gifted digital artist. For example, we would argue that the replicated cloning result (which for convenience is re-produced in Figure 114) would be extremely difficult to create manually. Our editing system places expressive tools in the hands of novice users without loss of quality. Furthermore, since all changes occur simultaneously there is less need for a long recorded 'undo' history of changes. We also eliminate the need for complex layering procedures and in doing so the user does not need to keep track of the particular layer they are currently operating within.

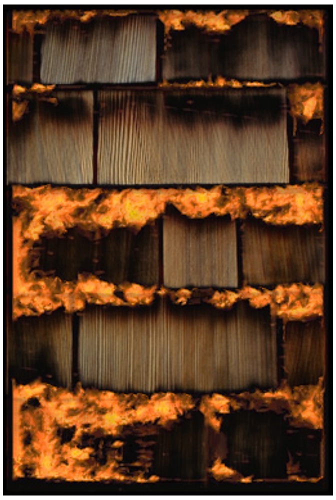


Figure 114: A replicated cloning result that would be difficult to paint manually.

The informal feedback that we have received from users has been very positive. Generally speaking, the response has included a measure of surprise that the complex changes can be performed with such sparse interaction. In Figure 115 we have included the results of using our system by a frequent Windows™ user, who infrequently uses image editors. The top row shows three targets which the user must match. The middle row shows the user's attempt at reproducing the targets using her favourite painting software. The bottom row shows the results of using our system, for which the user was given no instructions, except for pointing out the controls which could be adjusted. When using their favourite editor, the user noted having particular difficulty in maintaining consistency over all texture elements. We would like to stress, however, that the inclusion of this example is not meant to imply that a formal user study has been conducted; it is simply meant to show how difficult it can be for some users to manually paint with a digital image editor using a mouse.

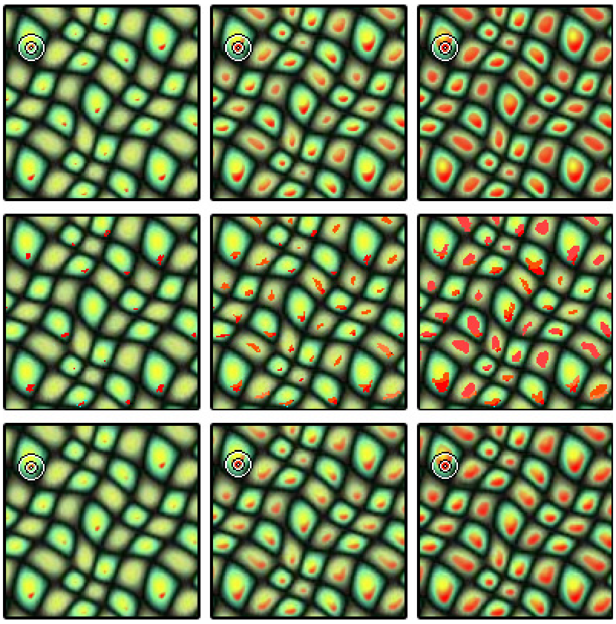


Figure 115: Top row: what the user must do. Middle row: a user's attempt using her favourite image editor. Bottom row: the same user's attempt using replicated painting without instruction.

2. Limitations

Our system of texture editing has advanced the state of the art of interactive image processing. There remain, however, a number of limitations to the system that we will now discuss.

2.1 Non-uniform Lighting

Currently our approach works best for textures which are uniformly lit. Non-uniform lighting leads to poorer results. Illumination correction is a difficult problem that we have not sought to solve. Nevertheless, we believe that this restriction might be addressed by integrating self-similarity based editing with an existing photo editing system such as that of Oh *et al.* which permits texture illumination correction [Oh01]. The illumination correction system would be executed as a pre-process to avoid any impact on interactivity.

2.2 General Images

As discussed in earlier chapters, our editing techniques are not generally suitable for non-texture images. However, we believe that this could be partly overcome by combining our system with a system of object segmentation such as that used in the aforementioned object-based image editing system of Barrett and Cheney [Barrett02]. Their system would be used in an initial stage to segment an image into separate areas of uniform texture. Once segmented, the effects of our replicated editing system could then be constrained to operate only within the current selected area. This would avoid the possibility of spurious neighbourhood matches between distinct areas textures within a general image.

2.3 Stochastic Textures

There also remains a further semantic limitation within the class of texture images. Texture images have often been subdivided into two categories: deterministic textures and stochastic textures [Heeger95, Efros99]. A fully deterministic texture is composed of an arrangement of identifiable primitives (e.g. a tile floor). On the other hand, a stochastic texture does not have easily identifiable primitives (e.g. granite, bark, sand). In practice, many real-world textures defy a clear classification and exhibit a mixture of these two characteristics (e.g. woven fabric, wood grain, ploughed fields). This grey area of textures we would call semi-deterministic textures.

Although Markov random field texture models can successfully synthesize stochastic as well as deterministic and semi-deterministic textures [Efros99], in our experience self-similarity based texture editing works best operating on deterministic and semi-deterministic textures. When using our system, the user is attempting to alter all texture elements at once; this only has meaning if there are identifiable texture elements (such as bricks) to begin with. The method is therefore not appropriate for textures that exhibit little or no structure. For example, in Figure 116 we see a highly unstructured texture which has been subjected to replicated painting with a green colour. Although our system does produce results in these cases, the operation does not have the same semantic significance.

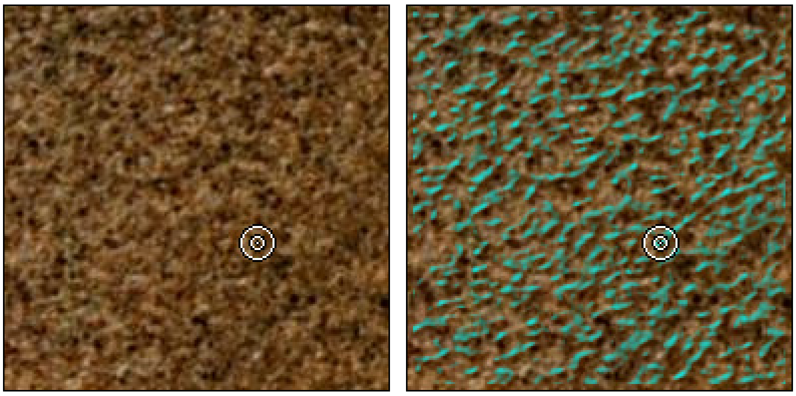


Figure 116: Lack of semantic meaning when replicated painting is applied to highly unstructured textures. The original texture is shown to the left. The right-hand image shows the same texture having been subjected to replicated painting with a green colour.

3. Future Directions of Research

We have presented a novel editing system that replicates painting, cloning and warping operations over a texture. The significance of this is that it allows the user to perform complex tasks in real-time with minimal effort. There remain however significant opportunities to extend the capabilities of our system:

Geometry Editing – The technique might 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. Other possibilities include the editing of bump maps, displacement maps or other material properties. It might also be possible to apply similar methods to the editing of 4 dimensional bi-directional texture functions (BDTFs). A key issue for BDTFs editing would be efficiently handling the larger amounts of data.

Alternate 2D Applications – Further replicated editing operations might be explored such as self-similarity based image filtering.

Self-Similarity Based Procedural Texturing – For this extension, the texture similarity levels (which for replicated painting were called opacity) would be used as an input parameter for the generation of a new procedural texture. This could be useful in two ways. Firstly, it would extend the variety of results that are possible with replicated cloning: once the procedural texture is generated to match certain areas of the target texture, it would then be cloned into the target texture. Secondly, it might be possible to generate a procedural texture that spatially matches a real texture for the purpose of replacing it. For example, we might wish to generate a procedural texture of marble that has similar spatial properties as a real marble texture. The procedural version of the original texture would have the advantage of resolution independence and would give further control over other attributes of the texture. This second use would be similar in purpose to the resolution independence work of Labrosse *et al.* [Froumentin00, Labrosse01].

Interface Improvements – The system would benefit from a detailed HCI study to better understand the strengths and weaknesses of the interface and to develop appropriate language that best represents the parameters. Such a study was beyond the scope of this dissertation and outside the expertise of the author.

Replicated Video Editing – Our editing techniques could also be applied to multi-frame video. We have, in fact, conducted preliminary work on this extension which we will now discuss.

The addition of video editing capabilities requires that a standard set of time position controls be added to the interface as can be seen in the bottom left panel in Figure 117. The set of buttons in the panel are the standard video controls (from left to right): move to beginning of sequence, step back one frame, playback animation sequence, step forward one frame, stop animation and move to end of sequence. A positional slider is also available beneath the control buttons for time positioning and frame position status.

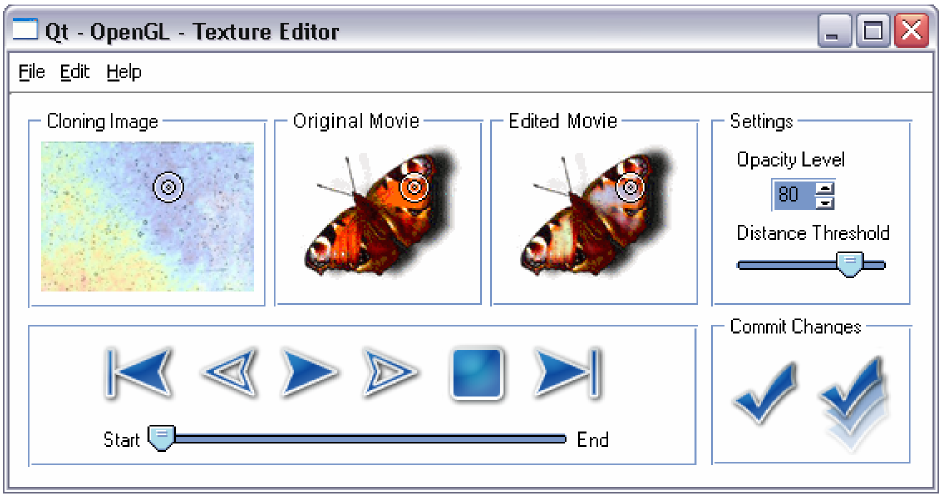


Figure 117: Video cloning editor.

In general, the editing of video sequences follows a similar user editing pattern to that of still images. The additional consideration lies in the handling of neighbourhood comparisons not just within the same image frame but also between multiple image frames.

For editing across video frames we can provide facilities that range from manual to fully automated interaction. Manual editing is the simplest solution to this issue and requires the user to set the position of the selection point in each frame and process each frame separately. To edit the video manually, the user moves sequentially through the sequence of images using the step forward button, setting the position of the similarity point and pressing the commit button at each frame. This in fact provides the most control over the final outcome. However, fully manual editing is not in keeping with the semi-automated style of interface that we have adopted.

At the other extreme, the system not only replicates the user's editing operation within the current frame but also cascades the operation across all time frames as well. To edit the entire sequence of images using this second approach, the user needs to position the selection point in one frame only. The user then presses the commit-all-frames button which is another addition to the image editing interface. The commit-all-frames button can be seen as a second large check mark in the lower right hand corner of Figure 117. It is differentiated from the single frame commit button (to its left) because in this case the check mark has been visually ghosted multiple times to suggest the duplication of its function across multiple frames in the animation.

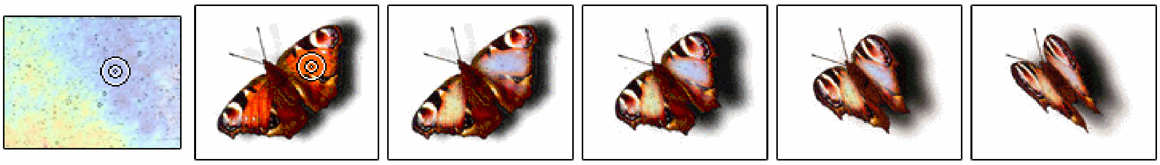


Figure 118: Cascaded video cloning. A colourful texture (leftmost) has been cloned onto multiple frames of an animation sequence. Note that the left wing has become yellow and the right wing, blue.

An example of multi-frame replicated cloning can be seen in Figure 118, where a colourful texture has been cloned onto multiple frames of an animation sequence. The user has selected a single point on a wing of a butterfly in a single frame of the animation sequence. The neighbourhood of this similarity point in the selected frame is used for similarity comparisons for all subsequent frames in the sequence. The left wing, which in terms of position corresponds to the yellow portion of the cloning image, has become cloned with the yellow portion of the texture, and the right wing, with the blue. In this case, the cloning has remained effective in all the frames even though the shape of the butterfly is changing. We would not, however, argue that this behaviour holds for all video sequences in general, a topic to which we will return later in the discussion.

In some cases, the user may wish to exercise varying degrees of control over the replication of editing operations across time. For this, we have developed neighbourhood key-framing which lies between manual and fully automatic modes of editing across time. The user can add more and more editing key-frames to the video by moving to a new time position and adjusting the selection point and/or the threshold and opacity levels. On each of the key-frames the neighbourhood of the selected point for that frame is used, as before. Between key-frames A and B with corresponding selection points SA and SB, the RGB values in the neighbourhoods around SA and SB are linearly interpolated, as are the threshold and opacity settings. With in-between frames controlled by nearby key-frames, it allows more control over the final outcome while not requiring the user to perform editing operating on every frame in the video.

As we have seen for still images, the user's input is amplified within the same image through replication. In the case of video we also obtain an amplification of user input across both time and space by cascading editing operations across a sequence of frames.

However, the reason why this work on video editing has been placed in the future work section is that the most significant challenges still need to be addressed. Since a typical frame of video is a general image and not a texture, the development of self-similarity video editing includes the challenge of extending the editing system to general images. This problem is compounded with the need to ensure temporal coherence which might necessitate object tracking. Both of these non-trivial issues remain open research problems in the context of self-similarity based video editing.

4. Conclusion

Semi-automatic image editing offers powerful new tools to both experienced graphic artists and novice users. With the introduction of self-similarity based texture editing we have made novel contributions to this growing paradigm of user-assisted image editing. Our system's style of interaction is based on the idea that the user suggests and the software articulates; the system replicates the user's decisions globally across the image. This amplifies the user's input and in so doing tedious editing operations are performed automatically. The implication of this is that it allows the user to perform difficult tasks in real-time with nominal effort.

BIBLIOGRAPHY

[ADOBE02] ADOBE. *Photoshop* c 7.0 *User Guide.* Adobe Systems Incorporated, 2002.

[Arad94] Arad, N., Dyn, N., Reisfeld, D., and Yeshurun, Y. Image Warping by Radial Basis Functions: Application to Facial Expressions. *Computer Vision, Graphics and Image Processing: Graphical Models and Image Processing*, 56, pp. 161-172, 1994.

[ART03] ART, Inc. *Chief Architect 9.0.* http://www.chiefarch.com/, 2003.

[Ashikhmin01] Ashikhmin, M. Synthesizing Natural Textures. *ACM Symposium on Interactive 3D Graphics*, pp. 217–226, 2001.

[Bar-Joseph01] Bar-Joseph, Z., El-Yaniv, R., Lischinski, D., and Werman, M. Texture Mixing and Texture Movie Synthesis Using Statistical Learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2), pp. 120-135, 2001.

[Benedet03] Raphael Benedet. 3D.ARTISTS. www.raph.com, 2003.

[Barrett02] Barrett, W. and Cheney, A. Object-Based Image Editing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), pp. 777-784, August 2002.

[Battke97] H. Battke, D. Stalling and H.-C. Hege. Fast Line Integral Convolution for Arbitrary Surfaces in 3D. *Visualization and Mathematics*, pp. 181-195, Springer, Heidelberg, 1997.

[BBC00] BBC News. Quake blows away design problems. http://www3.arct.cam.ac.uk/westC/cl/... BBCQuake.mht, October, 2000.

[Beier92] Beier, T., and Neely, S. Feature-based Image Metamorphosis. In *SIGGRAPH 92 Conference Proceedings*, pp. 35-42. ACM SIGGRAPH, July 1992.

[Berman94] Berman, D., Bartell, J., Salesin, D. Multiresolution Painting and Compositing. *Proceedings of ACM SIGGRAPH 94*, Annual Conference Series, pp. 85-90, July 1994.

[Blinn76] Blinn, J. F. and Newel, M. E. Texture and Reflection in Computer Generated Images. In *Communications of the ACM*, pp. 542-547, 1976.

[Blinn78] Blinn, J. F. Simulation of Wrinkled Surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, pages 286-292. ACM SIGGRAPH, Addison Wesley, August 1978.

[Borning79] Borning, Alan, Thinglab: A Constraint-Oriented Simulation Laboratory, *XEROX PARC report* SSL-79-3, July 1979.

[Brooks02] Brooks, S. and Dodgson, N.A. Self-Similarity Based Texture Editing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), pp. 653-656, July 2002.

[Brooks03] Brooks, S., Cardle, M. and Dodgson, N.A. Enhanced Texture Editing using Self Similarity. *Vision, Video, and Graphics 2003*, Bath, July 2003.

[Burt83a] Peter J. Burt and Edward H. Adelson. The Laplacian Pyramid as a Compact Image Code. In *IEEE Transactions on Communications*, volume 31(4), pp. 532-540. IEEE Computer Society, April 1983.

[Burt83b] Peter J. Burt and Edward H. Adelson. A Multiresolution Spline with Application to Image Mosaics. *ACM Transactions on Graphics*, 2(4), pp. 217-236, October 1983.

[Cameron99] Gordon Cameron, Ed. FOCUS: Applications of Computer Vision to Computer Graphics. *Computer Graphics Quarterly*, 33(4) ACM SIGGRAPH, November 1999.

[Catmull74] Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces,* PhD Thesis, University of Utah, 1974.

[Cohen03] Michael F. Cohen, Jonathan Shade, Stefan Hiller and Oliver Deussen. Wang Tiles for Image and Texture Generation. To appear in *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, July 2003.

[Corel03] Corel Corporation. *Corel Painter* 8. http://www.corel.com . 2003.

[Dawkins86] Dawkins, R. 1986. The Blind Watchmaker. *Longman Scientific & Technical* Pub. 14, 1986.

[Dawkins89] Dawkins, R. The Evolution of Evolvability. Artificial Life. The *Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, Vol. VI, Langton, C. G. (ed.), Los Alamos, New Mexico, Addison-Wesley Pub. Corp, pp.201-220, September, 1989.

[Davies00] E. R. Davies. *Image Processing For the Food Industry,* World Scientific, May 2000.

[DeBonet97] J. S. De Bonet. Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pp. 361-368. ACM SIGGRAPH, Addison Wesley, Aug 1997.

[DeCarlo02] Doug DeCarlo and Anthony Santella. Stylization and Abstraction of Photographs. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), pp. 769-776, August 2002.

[DeValois88] De Valois RL, De Valois KK. *Spatial Vision.* Oxford University Press: New York, 1988.

[Dischler99] Jean-Michel Dischler and Djanchid Ghazanfarpour. Interactive Image-based Modelling of Macrostructured Textures. In *IEEE Computer Graphics and Applications*, 19(1), pp. 66-74. IEEE Computer Society, January 1999.

[Dischler02] Jean-Michel Dischler, Karl Maritaud, Bruno Lévy and Djamchid Ghazanfarpour. Texture Particles. Computer Graphics Forum, 21(3), September 2002.

[Dodgson95] N. A. Dodgson and N. E. Wiseman, Graphic design for ceramic tiles. Eurographics UK 95, Loughborough University, pp.27–38, March 1995.

[Dorsey96] Julie Dorsey and Pat Hanrahan. Modelling and Rendering of Metallic Patinas. In *Computer Graphics (SIGGRAPH '96 Proceedings),* pp. 387-396. ACM SIGGRAPH, Addison Wesley, August 1996.

[Dorsey99] Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis, and Hans Kohling Pederson. Modelling and Rendering of Weathered Stone. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, pages 225-234. ACM SIGGRAPH, Addison Wesley, August 1999.

[Drori03] Iddo Drori, Daniel Cohen-Or and Hezy Yeshurun. Fragment-Based Image Completion. To appear in *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, July 2003.

[Ebert94] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin and Steven Worley. *Texturing and Modelling: A Procedural Approach.* AP Professional, Cambridge, MA, 1994.

[Efros99] Alexei A. Efros and Thomas K. Leung. Texture Synthesis by Non-parametric Sampling. *IEEE International Conference on Computer Vision (ICCV'99)*, Corfu, Greece, September 1999.

[Efros01] Efros, A., and Freeman, W. Image Quilting For Texture Synthesis and Transfer. *ACM SIGGRAPH 2001*, pp. 341–346, 2001.

[Elder98] Elder, J., And Goldberg, R. Image Editing In the Contour Domain. *IEEE Computer Vision and Pattern Recognition*, 374-381, 1998.

[Freeman92] William T. Freeman. *Steerable Filters and Local Analysis of Image Structure*. PhD thesis, MIT, Media Laboratory, Cambridge MA, 1992.

[Freeman02] William T. Freeman, Thouis R. Jones, and Egon C. Pasztor. Example-Based Super-Resolution. *Computer Graphics and Applications*, 22(2), pp. 56-65, March 2002.

[frogdesign03] frog design inc. http://www.frogdesign.com/, 2003.

[Froumentin99] Max Froumentin and Philip Willis. An Efficient 2.5D Rendering and Compositing System. *Computer Graphics Forum* 18(3) (Eurographics '99 Conference Issue), pp. C385-C394 and C428, 1999.

[Froumentin00] Max Froumentin, Frédéric Labrosse, and Philip Willis. A Vector-based representation for image warping. *Computer Graphics Forum*, 19(3), pp. C419-C425 and C543, 2000.

[Gardner85] Geoffrey Y. Gardner. Visual Simulation of Clouds. *Computer Graphics* (*ACM SIGGRAPH '85 Proceedings*), pp. 297-303, July, 1985.

[Gee03] James Paul Gee. *Power Up: What Video Games Have to Teach Us About Learning and Literacy,* Palgrave Macmillan, April 2003.

[Gersho92] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

[Glasbey98] Glasbey, C., and Mardia, K. A Review of Image-Warping Methods. *Journal of Applied Statistics*, 25(2), pp. 155-172, 1998.

[Glassner95] Andrew S. Glassner. *Principles of Digital Image Synthesis.* Morgan Kaufmann, San Francisco, 1995.

[Gleicher95] Gleicher, M. Image Snapping. *ACM SIGGRAPH 95*, pp. 183-190, 1995.

[Gooch01] Amy Gooch & Bruce Gooch. *Non-Photorealistic Rendering.* A.K. Peters Ltd, Publishers, 2001.

[Greenberg91] Greenberg, Donald P. Computers in Architecture. *Scientific American*, 264, 2, pp. 104-109, February 1991.

[Greene86] Ned Greene. Applications of World Projections. *Proceedings of Graphics Interface '86*, pp. 108-114, May, 1986.

[Haeberli93] Paul Haeberli and Mark Segal. *Texture Mapping as a Fundamental Drawing Primitive. GRAFICA Obscura.* http://www.sgi.com/grafica/, 1993.

[Hanrahan90] P. Hanrahan, P. Haeberli, Direct WYSIWYG Painting and Texturing on 3D Shapes, *ACM SIGGRAPH 90 Conference Proceedings*, pp. 215-224, 1990.

[Harrison01] P. Harrison. A Non-Hierarchical Procedure for Re-Synthesis of Complex Textures. *WSCG'2001*. February 2001.

[Heckbert89] Paul S. Heckbert. Fundamentals of Texture Mapping and Image Warping. *Technical Report No. UCB/CSD 89/516*, California Institute of Technology, 1989.

[Heeger95] D. J. Heeger and J. R. Bergen. Pyramid-Based Texture Analysis/Synthesis. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pp. 229-238. ACM SIGGRAPH, Addison Wesley, August 1995.

[Hertzmann98] Hertzmann, A. Painterly rendering with curved brush strokes of multiple sizes. *ACM SIGGRAPH 1998,* pp. 453-460, 1998.

[Hertzmann00] Hertzmann, A. and Perlin K. Painterly Rendering for Video and Interaction. *ACM SIGGRAPH 2000,* pp. 327-340, 2000.

[Hertzmann01] Hertzmann, A., Jacobs, C., Oliver, N., Curless, B., and Salesin, D. H. Image Analogies. *ACM SIGGRAPH 2001,* pp. 327-340, 2001.

[Hill90] Francis S. Hill, Jr., *Computer Graphics*, Macmillan Publishing Co., New York NY, 1990.

[Hirani96] A. N. Hirani and T. Totsuka. Combining Frequency and Spatial Domain Information for Fast Interactive Image Noise Removal. *Computer Graphics*, 30 (Annual Conference Series):269-276, 1996.

[Huang84] T.S. Huang and R. Tsai. Multiframe image restoration and registration. *Advances in Computer Vision and Image Processing*, 1, pp.317–339, 1984.

[Ibrahim98] A. Ibrahim. *GenShade: An Evolutionary Approach to Automatic and Interactive Procedural Texture Generation.* PhD thesis, Texas A&M University, May 1998.

[Igehy97] H. Igehy and L. Pereira. Image Replacement Through Texture Synthesis. In *International Conference on Image Processing*, volume 3, pp. 186189, Oct 1997.

[IDSA03] Interactive Digital Software Association. *Essential Facts about the Computer and Video Game Industry*, www.theesa.com/EF2003.pdf, 2003.

[Jackson94] Richard Jackson, Lindsay MacDonald and Ken Freeman. *Computer Generated Color: A Practical Guide to Presentation and Display*. John Wiley & Sons, 1994.

[Jollife86] I.T. Jollife. *Principal Component Analysis.* Springer-Verlag, New York, 1986.

[Kalnins02] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes and Adam Finkelstein. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), pp. 755-762, July 2002.

[Karasaridis96] Anestis Karasaridis and Eero Simoncelli. A filter design technique for steerable pyramid image transforms. In Proceedings of ICASSP-96, Atlanta, GA, May 1996.

[Keahey97] Keahey, A., and Robertson, E. Nonlinear Magnification Fields. *IEEE Symposium on Information Visualization*, pp. 51-58, 1997.

[Keys81] R. Keys. Bicubic Interpolation. *IEEE Transactions Acoustic Speech, Signal Processing*, 29, pp. 1153–1160, 1981.

[Knapp97] D. Knapp, J.P. Kerr, and M. Sellberg, Patient Specific Colour Texture Mapping of CT-based Anatomical Surface Models Utilizing Cryosection Data, *Medicine Meets Virtual Reality*, ISO Press, pp. 608-617, 1997.

[Kosak94] Corey Kosak, Joseph Marks, and Stuart Shieber. Automating the Layout of Network Diagrams with Specified Visual Organization. *Transactions on Systems, Man, and Cybernetics*, 24(3), pp. 440-454, 1994.

[Kurlander88] Kurlander, D., and Bier, E. Graphical Search and Replace. In *Computer Graphics (Proceedings of ACM SIGGRAPH 88)*, 22(4), ACM, pp. 113-120, 1988.

[Kurlander92] Kurlander, D. and Feiner, S. Interactive Constraint-Based Search and Replace. In *Proceedings of CHI '92*. pp. 609-618. May 1992.

[Kwatra03] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk and Aaron Bobick. Graphcut Textures: Image and Video Synthesis Using Graph Cuts. To appear in *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, July 2003.

[Labrosse01] Frédéric Labrosse and Philip Willis. Towards Continuous Image Representations**.** In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pp. 206-213, Plzen, Czech Republic, 2001.

[Labrosse03] Frédéric Labrosse. On the Editing of Images: Selecting Cutting and Filling-in. *Vision, Video, and Graphics 2003*, Bath, July 2003.

[Lee95] Seung-Yong Lee, Kyung-Yong Chwa, Sung Yong Shin, and George Wolberg. Image Metamorphosis Using Snakes and Free-Form Deformations. In *SIGGRAPH 95 Conference Proceedings*, pp. 439-448. ACM SIGGRAPH, August 1995.

[Lewis84] J. P. Lewis. Texture Synthesis for Digital Painting. *Computer Graphics*, 18(3), pp. 245-252, 1984.

[Lewis89] J. P. Lewis. Algorithms for solid noise synthesis. In *ACM SIGGRAPH '89 Conference Proceedings*, pp. 263-270. ACM, 1989.

[Liang01] Liang, L., Liu, C., Xu, Y., Guo, B., and Shum, H. Real-Time Texture Synthesis by Patch-Based Sampling. *ACM Transactions on Graphics*. 20(3), pp. 127–150, 2001.

[Marks97] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pster, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. In Turner Whitted, editor, *Proceedings of ACM SIGGRAPH*, pp. 389-400. Addison Wesley, 1997.

[Meier96] Barbara J. Meier. Painterly rendering for animation. In *ACM SIGGRAPH 96 Conference Proceedings*, pp. 477-484. August, 1996.

[MICROSOFT02] Microsoft ®. *Flight Simulator 2002*. http://www.microsoft.com/games/fs2002/, 2002.

[Milliron02] Tim Milliron, Robert J. Jensen, Ronen Barzel and Adam Finkelstein. A Framework for Geometric Warps and Deformations. ACM Transactions on Graphics. 21(1), pp. 20-51, January 2002.

[Mitchell95] Mitchell, William J. and Malcolm McCullough. *Digital Design Media*. Van Nostrand Reinhold, New York, NY, 1995.

[Mortensen95] Mortensen, E., and Barrett, W. Intelligent Scissors for Image Composition. *ACM SIGGRAPH 95,* pp. 191-198, 1995.

[Nealen03] Andrew Nealen and Marc Alexa. Hybrid Texture Synthesis. To appear in the *Proceedings of the Eurographics Symposium on Rendering,* 2003.

[Nelson85] Nelson, G. Juno, a constraint-based graphics system. In *Proceedings of SIGGRAPH 1985*.

[Nelson94] Nelson, G. and Heydon, A. The Juno-2 constraint-based drawing editor. *Technical Report 131a*, Digital Systems Research Centre, Palo Alto, California, 1994.

[nVidia02] NVIDIA 2002. *GeForce4 Ti Product Overview*. http://www.nvidia.com/... docs/lo/1467/SUPP/PO_GF4Ti_2.05.02.pdf, 2002.

[nydigitalsalon03] New York Digital Salon. http://www.nydigitalsalon.com/, June 2003.

[Oh01] Oh, B., Chen, M., Dorsey, J., and Durand, F. Image-Based Modelling and Photo Editing. *ACM SIGGRAPH 2001,* pp. 433-442, 2001.

[Oka87] Masaaki Oka and Kyoya Tsutsui and Akio Ohba and Yoshitaka. Real-Time Manipulation of Texture-Mapped Surfaces, *Computer Graphics* (SIGGRAPH '87 Proceedings), July, 1987.

[Ostromoukhov99] Victor Ostromoukhov. Digital Facial Engraving, In *Proceedings of SIGGRAPH '99*, Annual Conference Series, pp. 417-424, 1999.

[Peachey85] D. Peachey. Solid Texturing on Complex Surfaces. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3), pp. 279-286. ACM SIGGRAPH, Addison Wesley, July 1985.

[Pérez03] Patrick Pérez, Michel Gangnet and Andrew Blake. Poisson Image Editing. To appear in *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, July 2003.

[Perlin95] K. Perlin and L. Velho. Live Paint: Painting with Procedural Multiscale Textures. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, pp. 153-160, 1995.

[Phong75] Bui Tuong Phong. Illumination for computer-generated pictures. *Communications of the ACM*, 18(6), pp.311-317, June 1975.

[Pixar89] Pixar. *The RenderMan Interface, version 3.1 official specification.* Published by Pixar, 1989.

[Porter84] Porter, T., and Duff, T. Compositing Digital Images. *Computer Graphics*, 18, 3, 253-259, 1984.

[Praun01] E. Praun, H. Hoppe, M. Webb and A. Finkelstein. Real-Time Hatching. Proceedings of ACM SIGGRAPH 2001, pp. 579-584, 2001.

[Reinhard01] Erik Reinhard, Michael Ashikhmin, Bruce Gooch and Peter Shirley. Colour Transfer between Images. *IEEE Computer Graphics and Applications*, pp. 34 41, 2001.

[Ryall97] Kathleen Ryall. *Computer-Human Collaboration in the Design of Graphics.* PhD thesis, Harvard University, Cambridge Massachusettes, 1997.

[Salisbury97] Salisbury, Michael P., Michael T. Wong, John F. Hughes and David H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *ACM SIGGRAPH '97 Conference Proceedings*, pp. 401-406, August 1997.

[Scansoft03] Scansoft. *SuperGoo*®. http://www.scansoft.com/supergoo/. Scansoft Incorporated, 2003.

[Schödl00] Arno Schodl, Richard Szeliski, David H. Salesin and Irfan Essa. Video Textures. In *Computer Graphics (SIGGRAPH '00 Proceedings)*, pp. 489-498. ACM SIGGRAPH, Addison Wesley, August 2000.

[Shneiderman83] Ben Shneiderman. *Direct manipulation: A step beyond programming languages.* IEEE Computer, 16(8), pp. 57-69, August 1983.

[Simoncelli92] [Simoncelli92] E P Simoncelli, W T Freeman, E H Adelson and D J Heeger. Shiftable Multi-Scale Transforms [or, "What's Wrong with Orthonormal Wavelets"]. *IEEE Trans. Information Theory, Special Issue on Wavelets.* 38(2), pp. 587-607, March 1992.

[Simoncelli95] E P Simoncelli and W T Freeman. The Steerable Pyramid: A Flexible Architecture for Multi-Scale Derivative Computation. *Second International Conference on Image Processing*, pp. 444-447, 1995.

[Sims93] Karl Sims. Interactive evolution of equations for procedural models. The *Visual Computer*, volume 9, pp. 466--476, 1993.

[Soler02] Cyril Soler, Marie-Paule Cani and Alexis Angelidis. Hierarchical Pattern Mapping. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), pp. 673-680, July 2002.

[Stahlhut03] Oliver Stahlhut. Extending Natural Textures with Multi-Scale Synthesis. *Vision, Video, and Graphics 2003*, Bath, July 2003.

[Stollnitz96] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.

[Sutherland63] I. Sutherland. Sketchpad: A man-machine graphical communication system. *IFIPS Proceedings of the Spring Joint Computer Conference*, January 1963.

[Todd92] Todd, S. and Latham, W. *Evolutionary Art and Computers*. Academic Press, 1992.

[Tong02] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo and Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), pp. 665-672, July 2002.

[Turk91] G. Turk. Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion. *Computer Graphics*, 25(4), pp. 298-298, 1991.

[Turk01] Greg Turk. Texture Synthesis on Surfaces. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2001)*, pp. 347-354, July 2001.

[vanWijk91] J.J. van Wijk. Spot noise - Texture synthesis for data visualization. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pp. 263-272, July 1991.

[vanWijk02] Jarke J. van Wijk. Image Based Flow Visualization. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), pp. 745-754, July 2002.

[Walter98] Marcelo Walter, Alain Fournier, and Mark Reimers. Clonal Mosaic Model for the Synthesis of Mammalian Coat Patterns. In *Graphics Interface (Proc. Graphics Interface)*, pages 82-91. Canadian Human-Computer Communications Society, Oct 1998.

[Wei00] Li-Yi Wei and Marc Levoy. Fast Texture Synthesis using Tree-Structured Vector Quantization. In *Computer Graphics (SIGGRAPH '00 Proceedings)*, pp. 479-488. ACM SIGGRAPH, Addison Wesley, August 2000.

[Wei01] Li-Yi Wei and Marc Levoy. Texture Synthesis over Arbitrary Manifold Surfaces. In *Computer Graphics (SIGGRAPH '01 Proceedings)*, pp. 355–360. ACM SIGGRAPH, Addison Wesley, August 2001.

[Weins00] Andrea Wiens and Brian J Ross. Gentropy: Evolutionary 2D Texture Generation, *Proceedings GECCO 2000*, Late Breaking Papers, July, 2000.

[Welsh02] T. Welsh, M. Ashikhmin and K. Mueller. Transferring Colour to Greyscale Images. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3), pp. 277-280, August 2002.

[Williams83] Lance Williams. Pyramidal Parametrics. *Computer Graphics* (SIGGRAPH '83 Proceedings), Pages 1-11, July, 1983.

[Williams90] Lance Williams. 3D Paint. *Symposium on Interactive 3D Graphics*, pp. 225-233, 1990.

[Witkin91] A. Witkin and M. Kass. Reaction-diffusion textures. *Computer Graphics (SIGGRAPH '91 Proccedings)*, 25(4): pp. 299-308, July 1991.

[Wolberg98] G. Wolberg. Image Morphing: A Survey. *The Visual Computer*, 14(8/9) pp.360-372, 1998.

[Worley96] Steven P. Worley. A Cellular Texture Basis Function. *Computer Graphics (SIGGRAPH '96 Proceedings)*, pp. 291-294, Aug 1996.

[Wyszecki82] G Wyszecki and W S Stiles. *Color Science: Concepts and Meth*ods, Quantitative Data and Formulae. Wiley, New York, 1982.

[Wyvill87] Wyvill, G., Wyvill, B., McPheeters, C., Solid Texturing Of Soft Objects, *CGA*, 7(12), pp. 20-26, 1987.

[Zhang00] J. Zhang, P. Fieguth and D. Wang. *Handbook of Image & Video Processing,* A. Bovik, ed., Chapter 4.3, "Random field models", pp. 301-312, March 2000.

[Zhang03] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo and Heung-Yeung Shum. Synthesis of Progressively Variant Textures on Arbitrary Surfaces. To appear in *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, July 2003.

[Zhu98] S. C. Zhu, Y. Wu, and D. Mumford. Filters Random Fields and Maximum Entropy (FRAME): To A Unified Theory For Texture Modelling, *International Journal of Computer Vision*, 12(2), pp. 1-20, March/April 1998.

Appendix A: Steerable Pyramid Filters

$$B_0 = 0^o,\ B_1 = 45^o,\ B_2 = 90^o,\ B_3 = 135^o$$

$\mathrm{B}_0(9 \times 9) = \{$

{-8.113e-004, 3.910e-003, 1.346e-003, 7.470e-004, 0.000e+000, -7.470e-004, -1.346e-003, -3.910e-003, 8.113e-004},

{4.445e-003, 4.457e-003, -3.774e-003, -3.652e-004, 0.000e+000, 3.652e-004, 3.774e-003, -4.457e-003, -4.445e-003},

{1.232e-002, -5.872e-003, 8.258e-003, -2.252e-002, 0.000e+000, 2.252e-002, -8.258e-003, 5.872e-003, -1.232e-002},

{1.396e-002, -2.876e-003, 3.944e-002, -1.106e-001, 0.000e+000, 1.106e-001, -3.944e-002, 2.876e-003, -1.396e-002},

{1.418e-002, 8.527e-003, 5.361e-002, -1.768e-001, 0.000e+000, 1.768e-001, -5.361e-002, -8.527e-003, -1.418e-002},

{1.396e-002, -2.876e-003, 3.944e-002, -1.106e-001, 0.000e+000, 1.106e-001, -3.944e-002, 2.876e-003, -1.396e-002},

{1.232e-002, -5.872e-003, 8.258e-003, -2.252e-002, 0.000e+000, 2.252e-002, -8.258e-003, 5.872e-003, -1.232e-002},

{4.445e-003, 4.457e-003, -3.774e-003, -3.652e-004, 0.000e+000, 3.652e-004, 3.774e-003, -4.457e-003, -4.445e-003},

{-8.113e-004, 3.910e-003, 1.346e-003, 7.470e-004, 0.000e+000, -7.470e-004, -1.346e-003, -3.910e-003, 8.113e-004} };

$\mathrm{B}_1(9 \times 9) = \{$

{0.000e+000, 8.285e-004, 5.711e-005, -4.011e-005, -4.667e-003, -8.087e-003, -1.481e-002, -8.620e-003, 3.122e-003},

{-8.285e-004, 0.000e+000, 9.748e-004, 6.972e-003, 2.087e-003, -2.330e-003, 4.481e-003, -1.492e-002, -8.620e-003},

{-5.711e-005, -9.748e-004, 0.000e+000, 1.215e-002, 2.443e-002, -5.080e-002, -3.279e-002, 4.481e-003, -1.481e-002},

{4.011e-005, -6.972e-003, -1.215e-002, 0.000e+000, 1.511e-001, 8.250e-002, -5.080e-002, -2.330e-003, -8.087e-003},

{4.667e-003, -2.087e-003, -2.443e-002, -1.511e-001, 0.000e+000, 1.511e-001, 2.443e-002, 2.087e-003, -4.667e-003},

{8.087e-003, 2.330e-003, 5.080e-002, -8.250e-002, -1.511e-001, 0.000e+000, 1.215e-002, 6.972e-003, -4.011e-005},

{1.481e-002, -4.481e-003, 3.279e-002, 5.080e-002, -2.443e-002, -1.215e-002, 0.000e+000, 9.748e-004, 5.711e-005},

{8.620e-003, 1.492e-002, -4.481e-003, 2.330e-003, -2.087e-003, -6.972e-003, -9.748e-004, 0.000e+000, 8.285e-004},

{-3.122e-003, 8.620e-003, 1.481e-002, 8.087e-003, 4.667e-003, 4.011e-005, -5.711e-005, -8.285e-004, 0.000e+000} };

$\mathrm{B}_2(9 \times 9) = \{$

{8.113e-004, -4.445e-003, -1.232e-002, -1.396e-002, -1.418e-002, -1.396e-002, -1.232e-002, -4.445e-003, 8.113e-004},

{-3.910e-003, -4.457e-003, 5.872e-003, 2.876e-003, -8.527e-003, 2.876e-003, 5.872e-003, -4.457e-003, -3.910e-003},

{-1.346e-003, 3.774e-003, -8.258e-003, -3.944e-002, -5.361e-002, -3.944e-002, -8.258e-003, 3.774e-003, -1.346e-003},

{-7.470e-004, 3.652e-004, 2.252e-002, 1.106e-001, 1.768e-001, 1.106e-001, 2.252e-002, 3.652e-004, -7.470e-004},

{0.000e+000, 0.000e+000, 0.000e+000, 0.000e+000, 0.000e+000, 0.000e+000, 0.000e+000, 0.000e+000, 0.000e+000},

{7.470e-004, -3.652e-004, -2.252e-002, -1.106e-001, -1.768e-001, -1.106e-001, -2.252e-002, -3.652e-004, 7.470e-004},

{1.346e-003, -3.774e-003, 8.258e-003, 3.944e-002, 5.361e-002, 3.944e-002, 8.258e-003, -3.774e-003, 1.346e-003},

{3.910e-003, 4.457e-003, -5.872e-003, -2.876e-003, 8.527e-003, -2.876e-003, -5.872e-003, 4.457e-003, 3.910e-003},

{-8.113e-004, 4.445e-003, 1.232e-002, 1.396e-002, 1.418e-002, 1.396e-002, 1.232e-002, 4.445e-003, -8.113e-004} };

$B_3\left(9\times9\right)=\{$

{3.122e-003, -8.620e-003, -1.481e-002, -8.087e-003, -4.667e-003, -4.011e-005, 5.711e-005, 8.285e-004, 0.000e+000},

{-8.620e-003, -1.492e-002, 4.481e-003, -2.330e-003, 2.087e-003, 6.972e-003, 9.748e-004, 0.000e+000, -8.285e-004},

{-1.481e-002, 4.481e-003, -3.279e-002, -5.080e-002, 2.443e-002, 1.215e-002, 0.000e+000, -9.748e-004, -5.711e-005},

{-8.087e-003, -2.330e-003, -5.080e-002, 8.250e-002, 1.511e-001, 0.000e+000, -1.215e-002, -6.972e-003, 4.011e-005},

{-4.667e-003, 2.087e-003, 2.443e-002, 1.511e-001, 0.000e+000, -1.511e-001, -2.443e-002, -2.087e-003, 4.667e-003},

{-4.011e-005, 6.972e-003, 1.215e-002, 0.000e+000, -1.511e-001, -8.250e-002, 5.080e-002, 2.330e-003, 8.087e-003},

{5.711e-005, 9.748e-004, 0.000e+000, -1.215e-002, -2.443e-002, 5.080e-002, 3.279e-002, -4.481e-003, 1.481e-002},

{8.285e-004, 0.000e+000, -9.748e-004, -6.972e-003, -2.087e-003, 2.330e-003, -4.481e-003, 1.492e-002, 8.620e-003},

{0.000e+000, -8.285e-004, -5.711e-005, 4.011e-005, 4.667e-003, 8.087e-003, 1.481e-002, 8.620e-003, -3.122e-003} };

$L_0\left(9\times9\right)=\{$

{-8.701e-005, -1.354e-003, -1.601e-003, -5.034e-004, 2.524e-003, -5.034e-004, -1.601e-003, -1.354e-003, -8.701e-005},

{-1.354e-003, 2.922e-003, 7.523e-003, 8.224e-003, 1.108e-003, 8.224e-003, 7.523e-003, 2.922e-003, -1.354e-003},

{-1.601e-003, 7.523e-003, -7.061e-003, -3.769e-002, -3.297e-002, -3.769e-002, -7.061e-003, 7.523e-003, -1.601e-003},

{-5.034e-004, 8.224e-003, -3.769e-002, 4.381e-002, 1.812e-001, 4.381e-002, -3.769e-002, 8.224e-003, -5.034e-004},

{2.524e-003, 1.108e-003, -3.297e-002, 1.812e-001, 4.376e-001, 1.812e-001, -3.297e-002, 1.108e-003, 2.524e-003},

{-5.034e-004, 8.224e-003, -3.769e-002, 4.381e-002, 1.812e-001, 4.381e-002, -3.769e-002, 8.224e-003, -5.034e-004},

{-1.601e-003, 7.523e-003, -7.061e-003, -3.769e-002, -3.297e-002, -3.769e-002, -7.061e-003, 7.523e-003, -1.601e-003},

{-1.354e-003, 2.922e-003, 7.523e-003, 8.224e-003, 1.108e-003, 8.224e-003, 7.523e-003, 2.922e-003, -1.354e-003},

{-8.701e-005, -1.354e-003, -1.601e-003, -5.034e-004, 2.524e-003, -5.034e-004, -1.601e-003, -1.354e-003, -8.701e-005}, };

$L_1(17 \times 17) = \{$

{-4.350e-005, 1.208e-004, -6.771e-004, -1.243e-004, -8.006e-004, -1.597e-003, -2.517e-004, -4.202e-004, 1.262e-003, -4.202e-004, -2.517e-004, -1.597e-003, -8.006e-004, -1.243e-004, -6.771e-004, 1.208e-004, -4.350e-005},

{1.208e-004, 4.461e-004, -5.815e-004, 5.622e-004, -1.369e-004, 2.326e-003, 2.890e-003, 4.287e-003, 5.589e-003, 4.287e-003, 2.890e-003, 2.326e-003, -1.369e-004, 5.622e-004, -5.815e-004, 4.461e-004, 1.208e-004},

{-6.771e-004, -5.815e-004, 1.461e-003, 2.161e-003, 3.761e-003, 3.081e-003, 4.112e-003, 2.221e-003, 5.538e-004, 2.221e-003, 4.112e-003, 3.081e-003, 3.761e-003, 2.161e-003, 1.461e-003, -5.815e-004, -6.771e-004},

{-1.243e-004, 5.622e-004, 2.161e-003, 3.176e-003, 3.185e-003, -1.777e-003, -7.432e-003, -9.057e-003, -9.637e-003, -9.057e-003, -7.432e-003, -1.777e-003, 3.185e-003, 3.176e-003, 2.161e-003, 5.622e-004, -1.243e-004},

{-8.006e-004, -1.369e-004, 3.761e-003, 3.185e-003, -3.531e-003, -1.260e-002, -1.885e-002, -1.751e-002, -1.649e-002, -1.751e-002, -1.885e-002, -1.260e-002, -3.531e-003, 3.185e-003, 3.761e-003, -1.369e-004, -8.006e-004},

{-1.597e-003, 2.326e-003, 3.081e-003, -1.777e-003, -1.260e-002, -2.023e-002, -1.109e-002, 3.956e-003, 1.439e-002, 3.956e-003, -1.109e-002, -2.023e-002, -1.260e-002, -1.777e-003, 3.081e-003, 2.326e-003, -1.597e-003},

{-2.517e-004, 2.890e-003, 4.112e-003, -7.432e-003, -1.885e-002, -1.109e-002, 2.191e-002, 6.807e-002, 9.058e-002, 6.807e-002, 2.191e-002, -1.109e-002, -1.885e-002, -7.432e-003, 4.112e-003, 2.890e-003, -2.517e-004},

{-4.202e-004, 4.287e-003, 2.221e-003, -9.057e-003, -1.751e-002, 3.956e-003, 6.807e-002, 1.445e-001, 1.774e-001, 1.445e-001, 6.807e-002, 3.956e-003, -1.751e-002, -9.057e-003, 2.221e-003, 4.287e-003, -4.202e-004},

{1.262e-003, 5.589e-003, 5.538e-004, -9.637e-003, -1.649e-002, 1.439e-002, 9.058e-002, 1.774e-001, 2.120e-001, 1.774e-001, 9.058e-002, 1.439e-002, -1.649e-002, -9.637e-003, 5.538e-004, 5.589e-003, 1.262e-003},

{-4.202e-004, 4.287e-003, 2.221e-003, -9.057e-003, -1.751e-002, 3.956e-003, 6.807e-002, 1.445e-001, 1.774e-001, 1.445e-001, 6.807e-002, 3.956e-003, -1.751e-002, -9.057e-003, 2.221e-003, 4.287e-003, -4.202e-004},

{-2.517e-004, 2.890e-003, 4.112e-003, -7.432e-003, -1.885e-002, -1.109e-002, 2.191e-002, 6.807e-002, 9.058e-002, 6.807e-002, 2.191e-002, -1.109e-002, -1.885e-002, -7.432e-003, 4.112e-003, 2.890e-003, -2.517e-004},

{-1.597e-003, 2.326e-003, 3.081e-003, -1.777e-003, -1.260e-002, -2.023e-002, -1.109e-002, 3.956e-003, 1.439e-002, 3.956e-003, -1.109e-002, -2.023e-002, -1.260e-002, -1.777e-003, 3.081e-003, 2.326e-003, -1.597e-003},

{-8.006e-004, -1.369e-004, 3.761e-003, 3.185e-003, -3.531e-003, -1.260e-002, -1.885e-002, -1.751e-002, -1.649e-002, -1.751e-002, -1.885e-002, -1.260e-002, -3.531e-003, 3.185e-003, 3.761e-003, -1.369e-004, -8.006e-004},

{-1.243e-004, 5.622e-004, 2.161e-003, 3.176e-003, 3.185e-003, -1.777e-003, -7.432e-003, -9.057e-003, -9.637e-003, -9.057e-003, -7.432e-003, -1.777e-003, 3.185e-003, 3.176e-003, 2.161e-003, 5.622e-004, -1.243e-004},

{-6.771e-004, -5.815e-004, 1.461e-003, 2.161e-003, 3.761e-003, 3.081e-003, 4.112e-003, 2.221e-003, 5.538e-004, 2.221e-003, 4.112e-003, 3.081e-003, 3.761e-003, 2.161e-003, 1.461e-003, -5.815e-004, -6.771e-004},

{1.208e-004, 4.461e-004, -5.815e-004, 5.622e-004, -1.369e-004, 2.326e-003, 2.890e-003, 4.287e-003, 5.589e-003, 4.287e-003, 2.890e-003, 2.326e-003, -1.369e-004, 5.622e-004, -5.815e-004, 4.461e-004, 1.208e-004},

{-4.350e-005, 1.208e-004, -6.771e-004, -1.243e-004, -8.006e-004, -1.597e-003, -2.517e-004, -4.202e-004, 1.262e-003, -4.202e-004, -2.517e-004, -1.597e-003, -8.006e-004, -1.243e-004, -6.771e-004, 1.208e-004, -4.350e-005}};

$H_0(15\times 15)=\{$

{-4.048e-004, -6.260e-004, -3.783e-005, 8.839e-004, 1.545e-003, 1.924e-003, 2.069e-003, 2.090e-003, 2.069e-003, 1.924e-003, 1.545e-003, 8.839e-004, -3.783e-005, -6.260e-004, -4.048e-004},

{-6.260e-004, -3.273e-004, 7.744e-004, 1.587e-003, 2.175e-003, 2.563e-003, 2.289e-003, 1.976e-003, 2.289e-003, 2.563e-003, 2.175e-003, 1.587e-003, 7.744e-004, -3.273e-004, -6.260e-004},

{-3.783e-005, 7.744e-004, 1.179e-003, 1.405e-003, 2.225e-003, 2.115e-003, 3.358e-004, -8.337e-004, 3.358e-004, 2.115e-003, 2.225e-003, 1.405e-003, 1.179e-003, 7.744e-004, -3.783e-005},

{8.839e-004, 1.587e-003, 1.405e-003, 1.296e-003, -4.927e-004, -3.130e-003, -4.575e-003, -5.101e-003, -4.575e-003, -3.130e-003, -4.927e-004, 1.296e-003, 1.405e-003, 1.587e-003, 8.839e-004},

{1.545e-003, 2.175e-003, 2.225e-003, -4.927e-004, -6.322e-003, -2.756e-003, 5.363e-003, 7.303e-003, 5.363e-003, -2.756e-003, -6.322e-003, -4.927e-004, 2.225e-003, 2.175e-003, 1.545e-003},

{1.924e-003, 2.563e-003, 2.115e-003, -3.130e-003, -2.756e-003, 1.396e-002, 7.805e-003, -9.381e-003, 7.805e-003, 1.396e-002, -2.756e-003, -3.130e-003, 2.115e-003, 2.563e-003, 1.924e-003},

{2.069e-003, 2.289e-003, 3.358e-004, -4.575e-003, 5.363e-003, 7.805e-003, -7.950e-002, -1.554e-001, -7.950e-002, 7.805e-003, 5.363e-003, -4.575e-003, 3.358e-004, 2.289e-003, 2.069e-003},

{2.090e-003, 1.976e-003, -8.337e-004, -5.101e-003, 7.303e-003, -9.381e-003, -1.554e-001, 7.304e-001, -1.554e-001, -9.381e-003, 7.303e-003, -5.101e-003, -8.337e-004, 1.976e-003, 2.090e-003},

{2.069e-003, 2.289e-003, 3.358e-004, -4.575e-003, 5.363e-003, 7.805e-003, -7.950e-002, -1.554e-001, -7.950e-002, 7.805e-003, 5.363e-003, -4.575e-003, 3.358e-004, 2.289e-003, 2.069e-003},

{1.924e-003, 2.563e-003, 2.115e-003, -3.130e-003, -2.756e-003, 1.396e-002, 7.805e-003, -9.381e-003, 7.805e-003, 1.396e-002, -2.756e-003, -3.130e-003, 2.115e-003, 2.563e-003, 1.924e-003},

{1.545e-003, 2.175e-003, 2.225e-003, -4.927e-004, -6.322e-003, -2.756e-003, 5.363e-003, 7.303e-003, 5.363e-003, -2.756e-003, -6.322e-003, -4.927e-004, 2.225e-003, 2.175e-003, 1.545e-003},

{8.839e-004, 1.587e-003, 1.405e-003, 1.296e-003, -4.927e-004, -3.130e-003, -4.575e-003, -5.101e-003, -4.575e-003, -3.130e-003, -4.927e-004, 1.296e-003, 1.405e-003, 1.587e-003, 8.839e-004},

{-3.783e-005, 7.744e-004, 1.179e-003, 1.405e-003, 2.225e-003, 2.115e-003, 3.358e-004, -8.337e-004, 3.358e-004, 2.115e-003, 2.225e-003, 1.405e-003, 1.179e-003, 7.744e-004, -3.783e-005},

{-6.260e-004, -3.273e-004, 7.744e-004, 1.587e-003, 2.175e-003, 2.563e-003, 2.289e-003, 1.976e-003, 2.289e-003, 2.563e-003, 2.175e-003, 1.587e-003, 7.744e-004, -3.273e-004, -6.260e-004},

{-4.048e-004, -6.260e-004, -3.783e-005, 8.839e-004, 1.545e-003, 1.924e-003, 2.069e-003, 2.090e-003, 2.069e-003, 1.924e-003, 1.545e-003, 8.839e-004, -3.783e-005, -6.260e-004, -4.048e-004}};