TILED TEXTURE SYNTHESIS

A Thesis

by

LORI ANNE GREEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2003

Major Subject: Visualization Sciences

TILED TEXTURE SYNTHESIS

A Thesis

by

LORI ANNE GREEN

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Ergun Akleman (Chair of Committee)	John Keyser (Member)
Mary Saslow (Member)	Phillip Tabb (Head of Department)

December 2003

Major Subject: Visualization Sciences

ABSTRACT

Tiled Texture Synthesis. (December 2003)

Lori Anne Green, B.A., DePauw University

Chair of Advisory Committee: Dr. Ergun Akleman

In this thesis a new image-based texturing method has been developed. This new method allows users to synthesize tiled textures that can be mapped to any quadrilateral mesh without discontinuity or singularity. An interface has been developed that allows user control over output textures. Three methods have been included in the interface to create aperiodic looking textures for 3D models and two methods have been developed to create wallpaper images (repeating textures on a 2D surface). Using these texturing methods, texturing problems are simplified, and more time can be spent solving artistic problems.

To my family

ACKNOWLEDGMENTS

I would like to thank Ergun Akleman, who pushed me to do my best and taught me to accept criticism. I would like to thank Mary Saslow, who pushed me artistically and creatively and encouraged me to aim higher. I would also like to thank John Keyser, who was an excellent committee member who was always prompt and helpful when I asked for his aid.

I would like to thank my family for being supportive of my goals and dreams. My parents have been supportive in many ways that can not be counted, and my sister and brother-in-law have provided much friendship and love during my time at Texas A&M. Also I would like to thank my aunts and cousins who have all been supportive and also believe in my potential.

Lastly, thanks goes to my friends, the staff, and the faculty at the vizlab who made the vizlab environment and my experience here such a good one. In particular, I would like to thank Jeff Smith for his consistent and daily support that helped me to achieve more than I first thought possible. I also would like to thank the many good friends I have made here whose encouragement has helped me and whose work has inspired and influenced me. They are Cindy Hong, Felice House, Pradeep Mistry, Hyemee Choi, Charu Sharma, and Avneet Kaur to name just a few.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION AND MOTIVATION

I.1. Introduction

In this thesis a new texturing method has been developed. This new method allows users to synthesize tiled textures that can be mapped to any quadrilateral mesh without discontinuity or singularity. Using this texturing method texturing problems are simplified, and more time can be spent solving artistic problems.

I.2. Texturing

In the early development of computer graphics, one problem was that the 3D objects had unrealistic looking surfaces. Texture mapping, introduced by Catmull [4], is a popular solution for this problem in computer graphics applications. It allows the creation of complicated looking images without increasing the complexity of the surface geometry. Catmull's approach is to create grid-like coordinates in the *u* and *v* directions and to wrap the image around the surface based on these *u* and *v* coordinates (see Figure 1). This approach is widely used in current computer graphics applications especially to texture toroidal, planar, spherical and cylindrical surfaces. However, even for these surfaces there exist some problems such as distortion and discontinuities.

Even the simplest type of *u*-*v* mapping, planar mapping, can suffer from distortions, which means the texture is stretched or squeezed. In these cases the texture does not look like the type of texture it is supposed to represent. To solve such distortions several methods have been developed [20, 3, 15].

The journal model is *IEEE Visualization.*

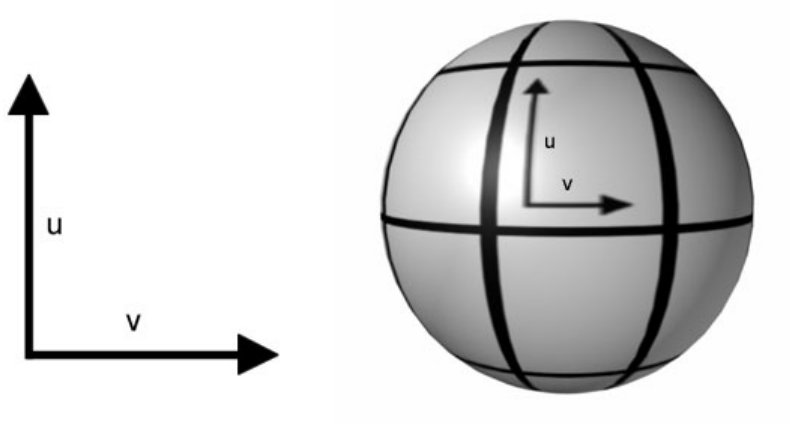


Fig. 1. UV mapping takes a grid and applies it to an object so that textures can be applied to the grid.

For toroidal and cylindrical surfaces, in addition to distortions, there can be discontinuities (seams) at the boundaries of the patches. This can be corrected using wallpaper images as textures as shown in Figure 2.

Mapping one patch to a sphere can cause an additional problem because of the singularities at the poles of the spheres. In such singular points, conflicting *u* and *v* coordinates can meet in one place. Figure 3 shows an example of such singularities that cause problems on spherical surfaces. Note that this problem cannot be corrected since the root of the problem is topological [8].

For other types (other than toroidal, cylindrical, and spherical) of surfaces, texture mapping faces a further mathematical problem that needs to be solved. Namely, for arbitrary surfaces, there is generally a need for more than one *u*-*v* texture patch to cover a whole surface. The requirement of more than one texture patch creates discontinuities at the patch boundaries. These discontinuities cannot be solved easily by using wallpaper images and they become even more visible in singularity points.

Another important problem in texturing is to maintain a balance between repetition

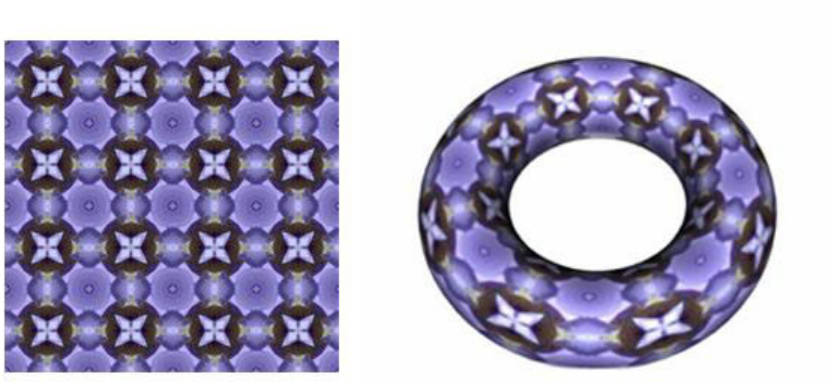


Fig. 2. A wallpaper image mapped to a torus seamlessly.

and randomness. Andrew Glassner showed that repetition and randomness lead to patterns that draw in viewers' attention and can keep their interest [10].

This randomness problem is related to aperiodic tiling problems in mathematics. If there's a set of one or more tiles that fit together to create exclusively nonperiodic tilings, that set of tiles and the resulting pattern is called aperiodic [10]. Amman and Robinson found a method to create such sets [10]. Stam used their ideas to create aperiodic textures on planar surfaces [23].

Textures can be created in various ways. Some textures can be made by hand using a digital painting program such as Photoshop. This method can be aided by creating textures using traditional painting methods or photography and then scanning those images to create digital textures. However, using this method makes it extremely hard to avoid singularity, seam, and distortion problems.

Another texture creation method is painting on a 3D surface directly. 3D paint programs allow users to paint directly on a 3D surface to create a 2D texture [13]. In this case, all the problems mentioned above can be avoided. However, this is an extremely

Fig. 3. Top: Example of an image with seams. Bottom: Example of an image with singularities.

time-consuming process that requires a large amount of skill and artistic experience.

Textures can also be created automatically using procedural or algorithmic methods. Automatic texture creation is called texture synthesis. For many patterns that can occur in nature, it is possible to develop textures procedurally. For these types of textures, patterns can be created based on science and math principles such as Turk's reaction-diffusion textures [25]. It is also possible to create new textures from a sample image. Many times the goal of the algorithm is to have the new texture appear to have the same properties [28].

I.3. Objectives

This research develops another image-based texture synthesis approach that is particularly connected to a texture mapping concept, namely tiled textures [2]. Using tiled textures, 3D models can be mapped in such a way that a set of tiles could easily be applied to the object without problems of seams or discontinuities. The main problem with tiled textures

is that the textures have to be created by hand, which can be a very complicated and time-consuming process. In this thesis, a concept is presented to develop a user interface for synthesizing tiled textures from sample images. Based on this concept, we have developed a tiled texture synthesis system and created a variety of tiled textures from images.

Figure 4 shows two tiled texture images created using the research presented in this thesis from photographed images of Miro and Kandinsky paintings that are mapped to a surface of a sphere.

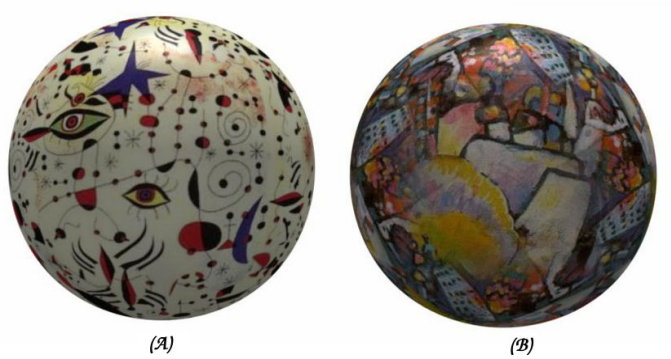


Fig. 4. Mapping two tiled texture images created from (A) Miro and (B) Kandinsky paintings to a spherical shaped mesh.

CHAPTER II

PREVIOUS WORK

The research in texturing can be classified in two categories: (1) Texture Synthesis (2) Texture Mapping. In this work, we have developed a user friendly texture synthesis approach for a particular texture mapping method, namely tiled textures.

II.1. Texture Synthesis

Texture Synthesis has been a widely researched area in computer graphics. Existing texture synthesis approaches can be classified into two categorizes:

- **Science based texture synthesis:** Textures are created by using scientific ideas, models, algorithms and procedures.
- **Image based texture synthesis:** Textures are created directly from sample images using digital signal processing approaches. In this case, no prior information about the nature of the textures in the images. The presented method is an image based texture synthesis.

II.1.1. Science Based Texture Synthesis

In science based texture synthesis, any good and powerful scientific model in biology, physics, chemistry or mathematics can be used to create textures.

A significant example in science based texture synthesis is reaction-diffusion. Turing introduced reaction-diffusion equations to represent processes in which two or more chemicals diffuse at unequal rates over a surface and react with one another to form stable patterns [24]. Turk, and Witkin and Kass [25, 29] applied Turing's reaction-diffusion

equations to computer graphics to create patterns such as spots and stripes on surfaces [24].

Another interesting example is genetic algorithms created in computer science to solve optimization problems using Darwinian concepts of evolution in biology. Genetic algorithms allow objects to evolve with reproduction, selection and fitness. Sims showed that genetic algorithms can be used to create textures [21].

Fractal geometry developed by Mandelbrot is also used for texture synthesis [16]. For instance, perlin noise is nothing but a fractal noise that is used to create a variety of realistic textures such as clouds, fire, water, stars, marble, wood, rock, soap films and crystal [19]. Van Wijk's spot noise can also be considered a form of fractal textures. Worley further develops this idea by using functions based on distributing points or bombing. Results of his work produced textures of organic crusty skin, crumpled paper, ice, rock, mountain ranges, and craters [31].

Similarly the cellular automata concept developed by Wolfram is later used for computer graphics [30]. Cellular automation models are systems that are composed of simple components limited to local interactions. Although cellular automation is base on simple interactions sophisticated behaviors can emerge. Gobron and Chiba created crack patterns on 3D surfaces using cellular automation [11]. By creating cells with independent behaviors various cracks in textures can be simulated.

II.1.2. Image Based Texture Synthesis

Textures can also be created from sample images using Digital Signal Processing algorithms. These algorithms take samples from real life and then create textures that have the same characteristics. These types of textures are based on research on human perception that concludes that two textures are difficult to discriminate when the orientation and distribution of certain characteristics are similar [14].

Heeger and Bergen's research on pyramid-based texture analysis and synthesis is a method that is motivated by this concept of human perception [14]. Using a sample image, noise is created that resembles the characteristics of the original image. This method is limited to homogenous textures, otherwise the texture appears blotchy and is unsuccessful.

Wei and Levoy recently developed a very powerful algorithm for image based texture synthesis [28]. Their algorithm is based on neighboring pixels. The goal of their algorithm is to base the pixel color of the generated output image on one with a similar neighborhood in the input image. To optimize this algorithm, a tree-structured data structure is used.

Ashikhmin further improves the Wei and Levoy algorithm and develops a faster and more powerful method to synthesize textures from a sample. [1]. He increases computational speed by limiting the search to already processed neighboring pixels. In particular, textures that occur in nature, such as field grass, tree branches and leaves, pebbles, blossoming flowers, bushes, and forest undergrowth, had very high quality results. He also allows user control in which users can input an image to help control the results of the output image.

Another significant algorithm in image based texturing is image quilting [7]. Image quilting is created by taking squares from an input texture and patching them together to create a new texture. To minimize edge seams, squares that have similar neighbors are chosen. To further minimize blockiness between the seams the overlapping regions are cut where a minimum color difference occurs. This work was also applied to 3D models by Soler, Cani and Angelidis [22].

II.2. Texture Mapping

An area of research related to texture synthesis is texture mapping. This work is concerned with how a 2D image is mapped onto a 3D model. Catmull's solution to this problem is to

use u-v coordinates to map textures, however, this approach alone does not solve problems of seams at the edge of the texture patches and distortion of the texture when it is mapped [4].

Neyret and Cani [18] developed a texture-mapping method that provides an effective solution to the problems of texture mapping. This work, which is based on triangular tiles with specific boundaries, avoids singularities regardless of the topology of the underlying triangular mesh. Their method also provides seamless and continuous boundaries across the surface. This method is similar in spirit to the method developed by Stam [23] for aperiodically texturing planes and toroids. However, the Neyret-Cani method is more general since it can be used to seamlessly texture any triangular mesh and avoids periodicity.

Their first step in their method is to create triangular tiles that obey specific boundary conditions. They showed that with a minimum number of tiles, rotation can provide all possible tile boundaries, and that it is possible to seamlessly cover any triangular mesh with these tiles. The simplicity and the power of the Neyret-Cani method makes it very attractive for texture mapping applications. From a practical point of view, there are two challenges with this method:

(1) The artist cannot automatically create tiles from existing images. To create intricate and homogeneous textures that look like those of natural objects, several different texture samples have to be hand-designed and boundary conditions must be applied, which is a difficult task.

(2) Using a set of triangular texture images is not compatible with current texture mapping methods and requires special programming. Most commercial renderers (like those in Maya, Softimage etc), allow only one single rectangular texture image to be read for each parameter of a shader.

Soler, Cani and Angelidis, who as mentioned previously developed a method based on

image quilting, introduced a powerful extension to the Neyret-Cani techniques [22]. Their algorithm creates a texture for a 3D model with an arbitrary surface with various sized patches that are created from an input image. Their method not only solves the challenges of texture mapping but also creates anisotropic texturing. Anisotropic textures, are textures with a direction, such as bark, wood, stripes on an animal, or text. However, their algorithm is complicated and can be quite hard to implement. Although their implementation is efficient, it still takes several minutes to compute a texture map. Also, if the texture image is changed there is a need to re-compute the texture mapping.

Recently a texturing method based on Wang Tiles was published [6, 12]. This work is based on quadrilateral tiles [6]. Wang Tiles, much like the Neyret and Cani method, use edge matching to create large aperiodic or non-repetitive textures [18]. In this method four tiles are used to create each square Wang tile. These tiles are blended together using the same method as Efros and Freemans's image quilting method [7]. The problem with this method is that it only works on models with grid-like structures. The algorithm fails on arbitrary surfaces as the north-south, east-west edge constraints are confused.

Akleman and Kaur have presented an alternative extension to the Neyret-Cani method [2], which provides an effective solution to the challenges mentioned above. Their system produces tiled texture image files, which consist of a set of square tiles. This algorithm assigns texture coordinates to each corner of the mesh which enables seamless and continuous boundaries across the surface. Once the texture coordinates are assigned, they can be used with tiled texture images in commercial 3D software packages.

II.3. Tiled Textures

There are three ways in which Akleman-Kaur simplify the Neyret-Cani method:

(1) using numbers to identify the boundary types instead of labels,

(2) using square tiles instead of triangular tiles, and

(3) using all possible tiles instead of only a minimal set [2].

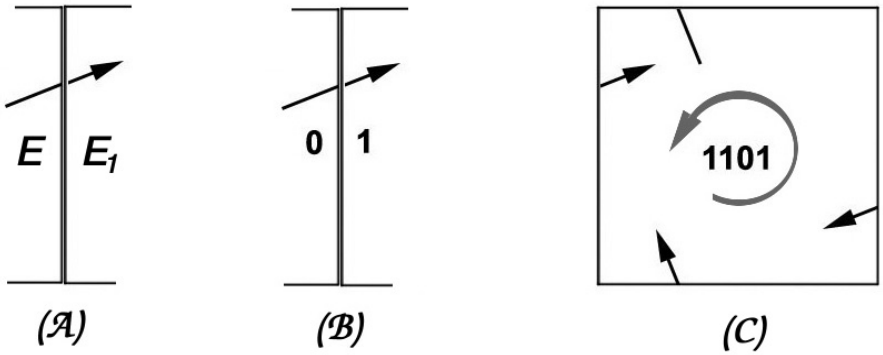


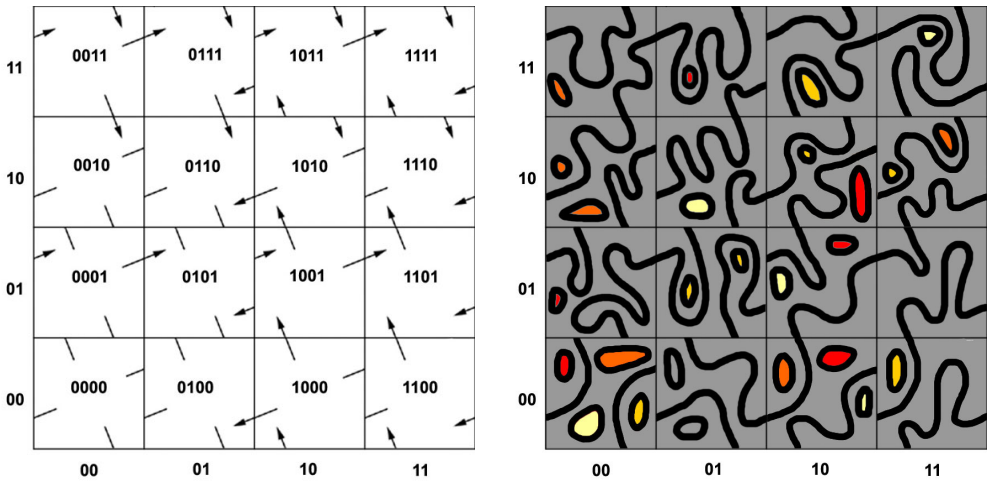
Fig. 5. Numbering the boundary types and naming the tiles with these number.

Neyret and Cani use labels as shown in Figure 5(A). In this image E and E_1 are used to denote boundary types and an arrow is used to denote the matching boundary edge. Akleman and Kaur used numbers instead of labels to denote boundary types as seen in Figure 5(B). In this image 0 is used instead of E and 1 instead of E_1. This new naming convention allows for a simple organization of the tiles into a texture image which aided them in developing efficient algorithms for texture mapping.

Beginning from the lower-left corner of a square and moving counter-clockwise, a unique name is generated for each tile, which is a four-digit binary number as shown in Figure 5(C). Each 90^0 rotation is represented by a cyclic shift of the name.

To seamlessly cover a quadrilateral mesh, there must be at least six tiles, such as $0000, 1000, 1100, 1110, 1010, 1111$ [12]. The problem with using such a minimal set is that there is not a simple way to organize the tiles into one texture file and the texture mapping algorithm becomes complicated. However, Kaur and Akleman use all possible tiles instead of a minimal set, and the problem is greatly simplified. As shown in Figure 6(A) they used

the boundary names as if they were positions in a matrix. Here, the first two digits of the name give the column number and the second two digits give the row number.



(A) (B)

Fig. 6. (A). An example of tiled texture image that shows how tiles are organized in a matrix. (B). A hand created tiled texture image.

A hand created example is shown in Figure 6(B). In this figure, the tile boundary edges are shown so that separate tiles are visible. One tile in the tiled texture image is assigned to each quadrilateral in the mesh while still maintaining boundary conditions. Figures 7 and 8 show two examples of the results mapped onto 3D objects.

The examples above have one distinct boundary condition, but this concept can be easily generalized to n distinct conditions by using four-digit $2n$-ary numbers. A boundary k is represented by two boundary types that are denoted by $2k$ and $2k+1$ where $k = 0, 1, \ldots, n-1$ and a tiled texture image consists of $(2n)^4$ tiles. The tile boundaries are used to find the name of the tiled texture that is to be placed in the quadrilateral. The current Akleman-Kaur algorithm is implemented for n distinct boundary conditions. This texture mapping algorithm was added to an existing modeling program called DLFL. DLFL

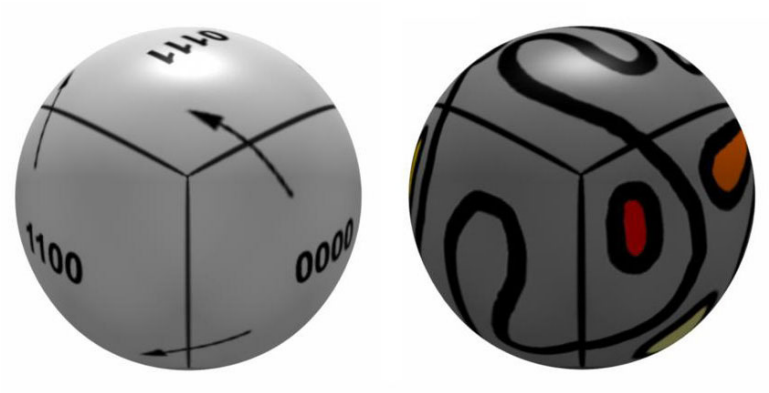


Fig. 7. A simple texture mapping example. A hand created texture has been applied to a cube. The cube is then smoothed using Catmull-Clark subdivision [5] in Maya.

Fig. 8. This example shows a genus-2 surface. The original shape is also smoothed by Catmull-Clark [5] after mapping the texture.

can output models with these reassigned coordinates. Then these models with reassigned coordinates can be taken into Maya or other 3D software programs.

However, the main problem with using Akleman-Kaur method is that to create a wide variety of tiled texture images by hand can be extremely hard. This thesis proposes to develop a simple approach to automatically create tiled textures from any given image. Using this method, users interactively synthesize texture image files from existing images with a simple interface.

CHAPTER III

METHODOLOGY

In this work, a method for automatically synthesizing tiled textures from sample images has been developed. Sets of tiles are created using the technique presented in this thesis work and can then be applied to objects that have been texture mapped using the Akleman-Kaur method [2].

The goal is to produce a set of tiles to create a texture that appears aperiodic. The tiles that are created can be thought of as having a grid or quilt structure. Each tile consists of two important regions: an interior region and an edge region. These tiles have to be blended to create a seamless and non-repeating pattern.

There are two important steps to create a tile. One is that the boundaries of the tile must have a way to match other tiles so that no seam is formed on the edge of the tiles when they are place on a 3D model. The second step is that a sample from the input image must be chosen fosecr the interior portion of the tiles and there must be a way to blend this sample for the interior of the image with the boundary areas of the tile.

Three different techniques for matching the edges of the tiles have been developed, each with increasing complexity. They are referred to as the $n = 0$, $n = 1$, and $n = 2$ case, where n represents the number of matching edges used in the algorithm. A matching edge can be defined as a place where two tile boundaries meet, and where a non-repeating pattern of one edge continues without seams or any visible discontinuity in the image. Also two different methods for creating wallpapers have been developed using the same methods developed for the tiled texture research. The interior blending uses the same blending method for all of cases presented in this paper. It is important to eliminate the seams and discontinuities on the interior of the image as well as between the boundaries of the tiles.

III.1. Algorithm for $n = 0$

The $n = 0$ method is one option for creating seamless textures. This option creates edges based on reflecting a triangular shaped portion of the original image 8 times. If all edges of the image are the same, then all of the edges of the tiled textures match, and there are no problems of seams in the texture. This concept was demonstrated in Figure 2. The $n = 0$ case is motivated by this idea.

Fig. 9. Methodology for n=0. A sample area is chosen from the input image.

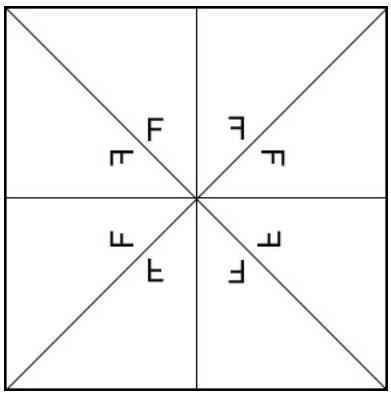


Fig. 10. Reflective algorithm. The triangular sample area is reflected to create the boundary image.

Fig. 11. Methodology for n=0. A sample from the input image is used to create the boundary image. The boundary image plus the background image are used to create one tile in the tiled texture.

Figure 9 shows an example of a data from a sample image that is used to create the boundary image. Figure 10 shows how the sample data is reflected. The triangle that has the F facing in the proper direction is the original triangle that is reflected and used as data for all of the other triangles in the square.

Figure 11 shows the basic concept of the $n = 0$ algorithm. A triangular sample of the image is chosen. This image is reflected to become the boundary image. A background image is chosen to fill the interior portion of the image. Note that for each tile a different background image is chosen from the input image. The background image becomes the center of the tiled texture, so that none of its boundaries will be used to form the final tiles.

After the boundary image and the background image have been selected, the boundary image is then blended from each side with the background image. Starting from the bottom edge of the background image and going counter-clockwise, the boundary image is blended with the background image. This order of blending is arbitrary, but does not impede the results.

III.2. Algorithm for $n = 1$

The $n = 1$ case has 4X4 individual tiles per tiled texture. These tiles are stored in a matrix format that can function with the texture mapping method. As show in Figure 12 two squares that are directly next to each other are sampled to create two boundary images. These square regions can be called the 0 and 1 boundary regions. The arrows that cut throught the boundaries show which edges are matching.

Fig. 12. Methodology for the n=1 case. The boundary images are sampled from the input image.

In order to understand how to use these boundary images, the name of the tile must be referenced. The boundaries for this case are created based on the name of the tile. The name of the tile is based on where in the final output texture the tile will be placed as shown in Figure 6[A]. There are 16 possible tiles in this case. In Figure 13 the example tile is named 0101. The boundary images are added to the background image. Starting at the bottom of the background image the 0 boundary is the first image to be blended to the background image.

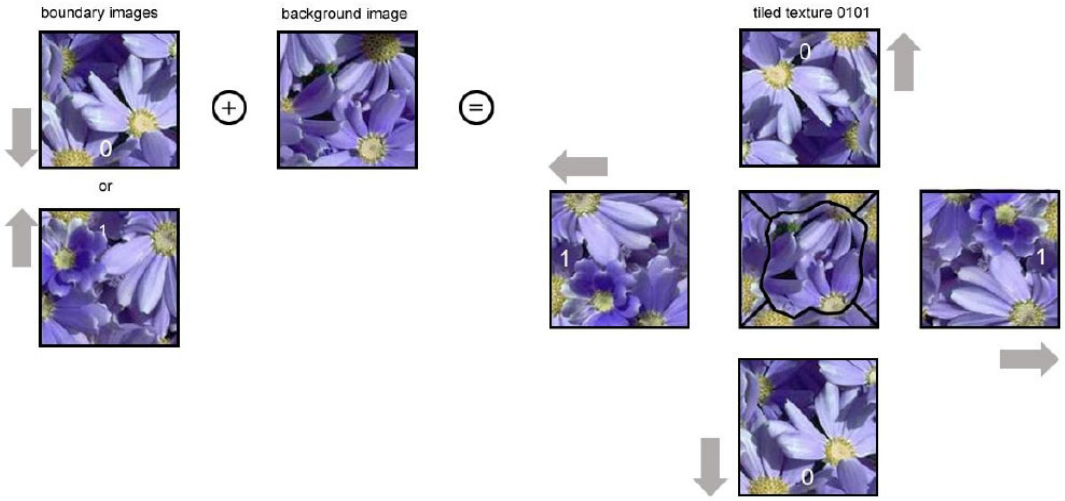


Fig. 13. Methodology for the n=1 case. The tiled texture is created by using both the boundary images plus the background image.

Notice that the arrows always point the matching edge to the outside of the image. The matching edge is determined by looking at the sampled squares. The sampled area is cut in half and the cut edge becomes the matching edge. The boundary image needs to always be rotated so that the matching edge is facing outwards. After the first 0 image is blended, the next step is to blend the edge that is 90 degrees counter-clockwise. The 1 edge according to the name of the tile is the next edge that is blended into the background image. After that the remaining edges must be rotated and blended according to which edge comes next in the tile name. In the end an image similar to the one in Figure 13 is the result. The center of the image represents the portion of the tiled texture that uses the color information from the background image and the edge regions represent the portions of the final image that use data from the boundary images. The final tiled texture is one of 16 that need to be created. Figure 6[A] shows where the tile 0101 would be placed in the final output image.

III.3. Algorithm for $n = 2$

	00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33
33	0033	0133	0233	0333	1033	1133	1233	1333	2033	2133	2233	2333	3033	3133	3233	3333
32	0032	0132	0232	0332	1032	1132	1232	1332	2032	2132	2232	2332	3032	3132	3232	3332
31	0031	0131	0231	0331	1031	1131	1231	1331	2031	2131	2231	2331	3031	3131	3231	3331
30	0030	0130	0230	0330	1030	1130	1230	1330	2030	2130	2230	2330	3030	3130	3231	3330
23	0023	0123	0223	0323	1023	1123	1223	1323	2023	2123	2223	2323	3023	3123	3223	3323
22	0022	0122	0222	0322	1022	1122	1222	1322	2022	2122	2222	2322	3022	3122	3222	3322
21	0021	0121	0221	0321	1021	1121	1221	1321	2021	2121	2221	2321	3021	3121	3221	3321
20	0020	0120	0220	0320	1020	1120	1220	1320	2020	2120	2220	2320	3020	3120	3220	3320
13	0013	0113	0213	0313	1013	1113	1213	1313	2013	2113	2213	2313	3013	3113	3213	3313
12	0012	0112	0212	0312	1012	1112	1212	1312	2012	2112	2212	2312	3012	3112	3212	3312
11	0011	0111	0211	0311	1011	1111	1211	1311	2011	2111	2211	2311	3011	3111	3211	3311
10	0010	0110	0210	0310	1010	1110	1210	1310	2010	2110	2210	2310	3010	3110	3210	3310
03	0003	0103	0203	0303	1003	1103	1203	1303	2003	2103	2203	2303	3003	3103	3203	3303
02	0002	0102	0202	0302	1002	1102	1202	1302	2002	2102	2202	2302	3002	3102	3202	3302
01	0001	0101	0201	0301	1001	1101	1201	1301	2001	2101	2201	2301	3001	3101	3201	3301
00	0000	0100	0200	0300	1000	1100	1200	1300	2000	2100	2200	2300	3000	3100	3200	3300

Fig. 14. Methodology for n=2. Names of all of the n=2 tiles.

The $n = 2$ case has 16X16 individual tiles per tiled texture. This data is also stored in a matrix format just like the $n = 1$ case; however, because the algorithm must use all possible sets, the tile is four times larger. Figure 14 shows all the names of the tiles that are created for the $n = 2$ case.

Fig. 15. Methodology for n=2. The boundary images are sampled from the input image.

The $n = 2$ boundary condition not only creates more complexity, but allows for anisotropic textures or textures with a direction such as bark or zebra stripes.

The main difference between the $n = 2$ case and the $n = 1$ case is that there are four boundary images to choose from instead of two. The boundary images are named 0,1,2 and 3. The boundary template in Figure 15 shows what part of the input image is used to create the boundary images. The 0 and 1 image used are the same as in n=1. The boundary image 2 is actually the same as image 0. The difference is that a different matching edge is used on that image. In this instance the side edge of that image is the image 2 border and the image to the right of that edge is the image 3 border. Once again the image is blended with the background image. The example tile created in Figure 16 is 0123. Starting at the bottom the correctly rotated image is blended with the background image, once again rotating to the counter-clockwise direction. Note if there is a corner with 0 and then 2 next to each other there is a more seamless looking corner based on the way that the data is selected (see Figure 16).

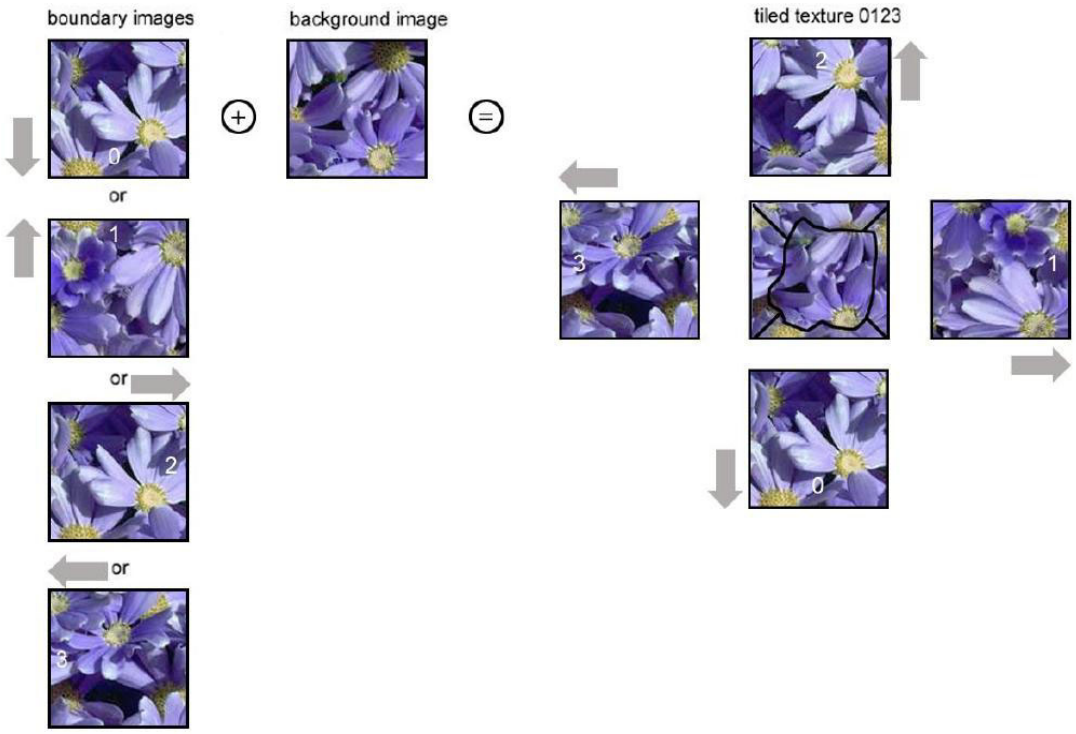


Fig. 16. Methodology for n=2. The boundary images plus the background image create the tile.

III.4. Wallpaper Algorithm

Although a plane can be covered using the n=0, n=1 and n=2 algorithms, other algorithms have been developed that repeat a single tile. These algorithms create repeating patterns that can be applied to a surface. The same principles of edge matching and blending apply when creating these wallpapers. The boundaries of the wallpaper images must appear seamless and the interior of the images must be seamless as well. Two type of wallpapers have been developed. One is called a tiled wallpaper and the other is called a reflective wallpaper.

The tiled wallpaper uses the same algorithm as the $n = 1$ case, however only one

square is made. Its name is 0011 so that a 0 edge will always be matched to a 1 edge and vice-versa.

The reflective wallpaper is made similarly to the $n = 0$ algorithm, except that no interior image is place in the tiles. It is only the reflective triangle image.

III.5. Interior Blending Algorithm

For all cases presented in this thesis, a blending algorithm for the interior portion of the tiles has been developed. There are two steps in this blending algorithm: a system for creating the interior portion for each tile and a system for blending this interior portion of the tile with the boundary image.

The interior portion of the tile is randomly selected from the sample image and becomes the background image for the tile. The size of this background image is the same size as the size of one tile. The edges of the background image are replaced with the boundary images in the final output version of the tile.

The interior portion of the image is filled with a background image. To create each tile of a tiled texture image, a background image is composited with four boundary images. This approach can be considered a variant of image quilting [7].

After the boundary image and the background image have been selected, the boundary image is then blended from each side with the background image. Starting from the bottom edge of the background image and going counter clockwise, the boundary image is blended with the background image.

The amount of blending of the boundary image with the interior image is determined by input values to functions. Every edge except for the edge that is being blended with the center image will be transparent. Figure 17 graphically shows the four functions that are used in the interior blending algorithm. Functions, f_1, f_2 and f_3, guarantee that boundaries

of the background image become transparent and the boundary images will be visible. Function f_1 determines how far back the boundary image blend into the background image along a straight line. The a and b lines show in Figure 17 are the areas in the image where the colors of the background image and the boundary image are mixed. The t variable, standing for threshold, is a value that can alter the outcome of the ratio. When the pixel is past line b, the ratio is equal to 0 and no part of the boundary image is included in the output image. Function f_2 determines how a value is mixed from the left angle. As shown in Figure 17 there is also a range between lines a and b, however, in this case the areas that are being mixed occur between two diagonal lines. Once again t can alter the ratios of mixing between lines a and b. Function f_3 is the same as function f_2 except that it controls the blending for the right angle of the tile. Function f_4 guarantees that for every column of the image, once color values of two image layers become similar, the boundary image becomes transparent. The boundary image is then composited onto the background image. The background image together with the new composited edge become the new background image. This process is repeated 4 times until there is a new edge for each side of the interior image and one tile of the tiled texture has been created. Figure 18 show how the four boundary images are composited over the background image. The white area represents the part of the boundary image that is used in the final tile of the tiled texture.

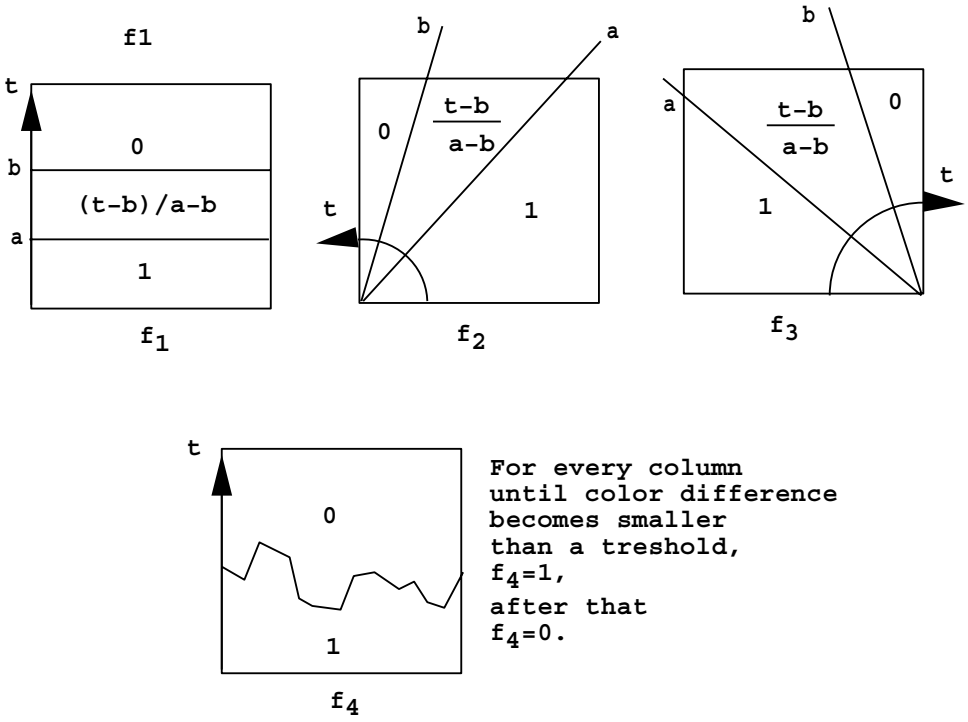


Fig. 17. The functions that are used to blend the boundary images with the background image.

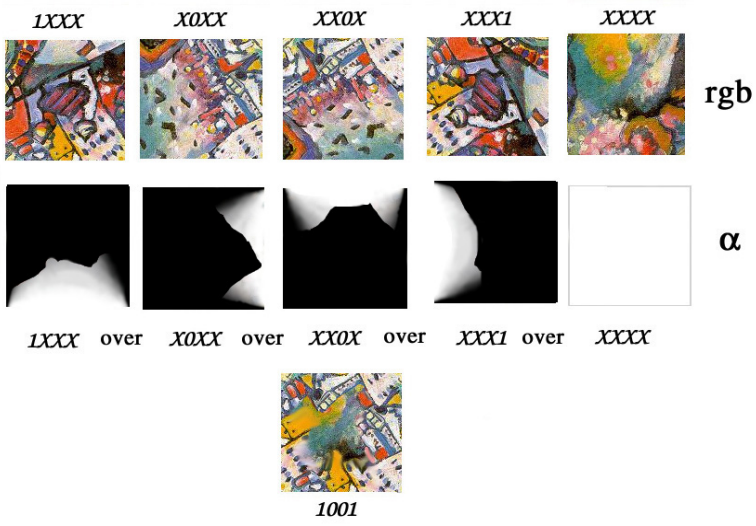


Fig. 18. Creation of the tile 1001 by compositing five images.

CHAPTER IV

IMPLEMENTATION

The current texture synthesis program, Aperiodic Demo, is implemented as a stand-alone system in C++ with FLTK, openGL and IFL libraries. The program currently runs on an SGI-O2.

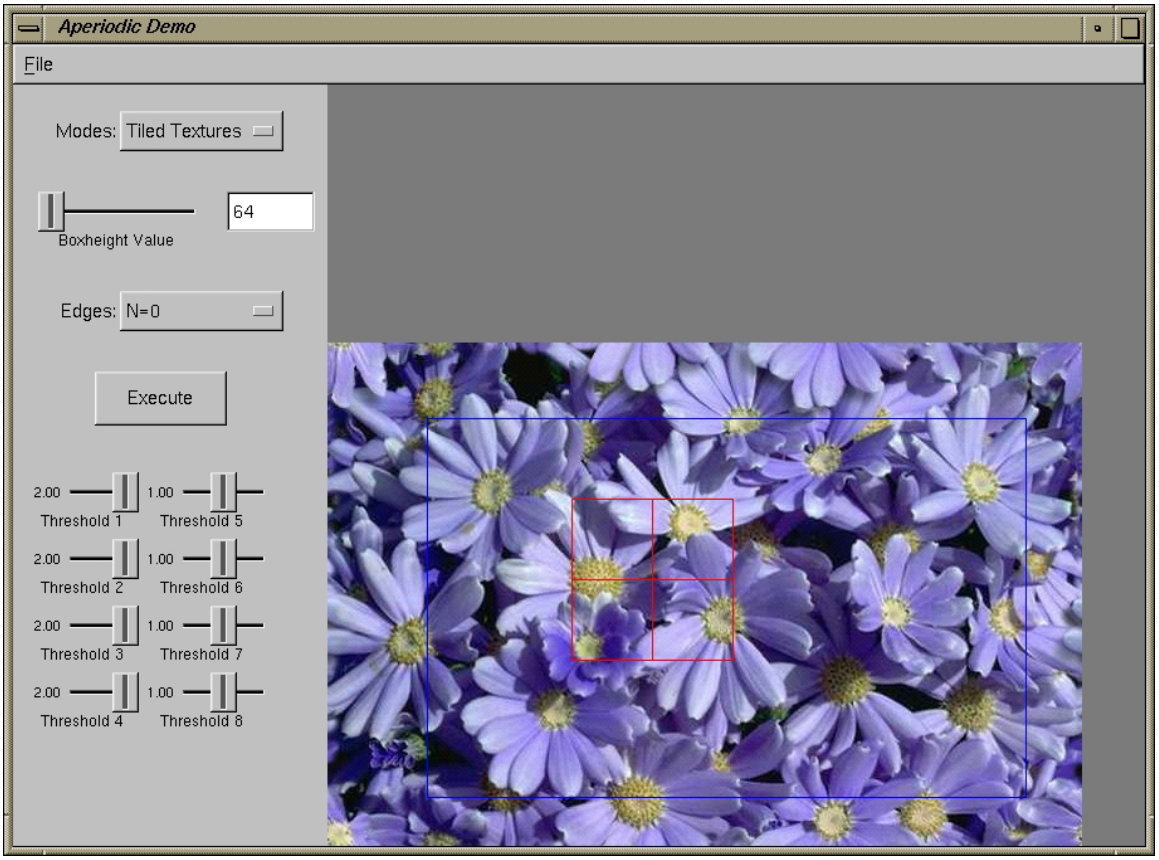


Fig. 19. The interface of the tiled texture image synthesis system. The red template represents the areas from which the boundary images are chosen and the blue areas represent the areas from which the background images are chosen.

The interface that has been developed is simple and intuitive (see Figure 19). The user

is given control over the final output image. To accomplish this, a blue overlay has been created to select the background images of each tile in the tiled texture and a red overlay has been created to select the boundary image. There is also a set of sliders to control the amount of blending that occurs between the background image and the boundary image.

The interface has a drop file menu at the top, and a side panel with many options and a large window where a texture is loaded. From this texture, a tiled texture or wallpaper can be created and output into a popup window.

In this procedure, the information from the input devices first needs to be collected. To get the x and y coordinates of the red and blue overlays, FLTK mouse events are handled. The red overlay is controlled by the left-mouse button and the blue overlay is controlled by the middle-mouse button.

When the left-mouse button is pushed down the red overlay can be dragged anywhere over the image until the mouse button is released. When the middle-mouse button is pressed, one corner of the blue rectangle is chosen. Then the width and height of the edges change as the mouse is dragged. When the mouse is released the opposite corner of the blue rectangle is chosen. The width and height of the edges change as the mouse is dragged.

These two mouse controlled overlays determine the output of the image. The red overlay selects boundary regions of the output image. The blue overlay selects the background region of the tiles. The default region selected by the blue overlay is the entire image.

The file menu at the top contains the options Open, Save and Quit. Open loads an image that will be used in creating the tiled texture. Compatible formats are .jpg, .sgi or .png. Save writes out the output image into a .jpg, .sgi or .png file. Quit ends the application.

The side panel contains the following options:

- *Boxheight* - Boxheight is a slider that controls the size of the output images. All of

the tiled textures are made up of many boxes. Boxheight controls the size of each tile in the tiled texture. The output images are made with varying numbers of tiles. For example an $n = 1$ image, which outputs a 4X4 box image, of boxheight 64 will result in an image size of 256X256. In order for the images to be compatible with the texture coordinates output from DLFL the boxheights need to be powers of 2. The boxheight sizes allowed in this application are 64, 128, 256 or 512.

- *Modes* - There are two modes: Tiled Textures and Wallpapers. There are three types of tiled textures: $n = 0$, $n = 1$, $n = 2$, where n represents the number of matching edges used to create the tiled texture.

- *Tiled Textures* - Tiled textures are the output images that work with tiled texturing in DLFL. Any quadrilateral model can be exported from DLFL with reassigned tiled texture coordinates and can then use the textures that are created with this program.

- *n=0 Mode* - The $n = 0$ mode uses a triangle which is 1/8th of a boxheight square and reflects the triangle eight times to create a reflective image. This image is blended with a background image, resulting in a seamless image. The output image contain 4X4 boxes. To create models that are compatible with this mode, models should be quadrilateral and the tiled texture setting in DLFL should be $N = 2$. (see section IV.1.1)

- *n=1 Mode* - The $n = 1$ mode uses the left top and bottom square of the red overlay to find the edges. This mode outputs 4X4 boxes, 16 in all and has an image size of Boxheight*4 X Boxheight*4 as the width and height. To create models that are compatible with this mode, models should be quadrilateral and the tiled texture setting in DLFL should be $N = 2$ (see section IV.1.2).

- *n=2 Mode* - The $n = 2$ mode provides even more complex results. Using more than

one boundary condition to create the textures provides an even more random pattern, which creates more convincing results. This mode outputs 16X16 boxes and has an image size of Boxheight*16 X Boxheight*16 as the width and height. To create models that are compatible with this mode, models should be quadrilateral and the tiled texture setting in DLFL should be $N = 4$ (see section IV.1.3).

Two wallpaper options are also available. One is called Tiled Wallpaper and the other is called Reflective Wallpaper.

- *Tiled Wallpaper* - The tiled wallpaper is one single box that has the pattern 0011. This setting outputs an image that has the dimension Boxheight X Boxheight and has 1X1 boxes (see section IV.2.1).

- *Reflective Wallpaper* - The reflective wallpaper uses a triangle which is 1/8th of a boxheight square and reflects it eight times to create a reflective image. It is similar to the $n = 0$ case, except that no background region is blended with the reflected image. It is simply the reflected image. This mode output 4X4 boxes, 16 in all and has an image size of Boxheight*4 X Boxheight*4 as the width and height (see section IV.2.2).

- *Execute* - This button executes the option that has been specified with the mode and edge or wallpaper options. All of the settings on the side panel are applied to the output image.

- *Thresholds* - The thresholds are set to values of 2 and 1 in the default settings. These values can be changed to obtain more desirable blending results. See subsection IV.3 and Figure 17 for more detailed explanation of how the functions are affected by the thresholds. The default threshold values were chosen because the results worked well in the testing.

- *Thresholds 1 and 5* - These sliders alter the color tolerance between the boundary regions and the background image. Larger thresholds will extend the tolerance of the color matching. Default value for threshold 1 is 2.0 and the default value for threshold 5 is 1.0.

- *Thresholds 2 and 6* - These sliders determine the distance that the opacity of the boundary region will blend with the background region. Default value for threshold 2 is 2.0 and the default value for threshold 6 is 1.0.

- *Thresholds 3 and 7* - These sliders determines the angle of the right diagonal threshold. Default value for threshold 3 is 2.0 and the default value for threshold 7 is 1.0.

- *Thresholds 4 and 8* - These sliders determines the angle of the left diagonal threshold. Default value for threshold 4 is 2.0 and the default value for threshold 8 is 1.0.

The user chooses the mode and the other options on the side bar. When the settings have been chosen the user then hits execute. A function is called based on the to the variable passed.

There are two modes that have been implemented: Tiled Textures and Wallpapers. Three options have been implemented under the tiled texture mode. They are the $n = 0$, $n = 1$ and $n = 2$ cases. Two options have been developed in the wallpaper mode. They are tiled wallpaper and reflective wallpaper.

IV.1. Tiled Texture Mode

IV.1.1. Implementation for n=0

The first step in creating a tiled texture for case $n = 0$ is selecting a triangular area is selected from the red square overlay. The triangular area is chosen from the bottom left side of the first square in the red overlay as shown in Figure 15.

This data is reflected 8 times and this data is stored in a new pixmap of size boxheight X boxheight (see Figure 10).

When the execute button is hit with the dropdown option $n = 0$ chosen, the function $execute0()$ is called. A method to create a new tile, $createNew()$, is called 16 times in $execute0()$ in order to create the 16 tiles needed to fill a 4X4 image. Counter values of width and height are passed to this function to determine where the new tile will be placed in the output image.

To create a tile, a square of size boxheight must be chosen from the input image. This tile is randomly chosen from the area within the blue overlay region that is selected by the user. A random X and Y value is chosen by the function $randomNum(char)$, using the $rand()$ function, from within the blue overlay region. These values become the coordinates of the bottom left corner of the square. The chosen square becomes the background image and is composited with the boundary region to create the tiled texture.

In the $n = 0$ case the boundary region is the reflected image. One tile is created each time $createNew()$ is called. The tile is created by using the reflected image as the boundary image and blending each edge of the boundary region with the background image. This process begins with the bottom edge of the tile and rotates counterclockwise 90 degrees to the next edge. In this case the reflective image is used as the boundary image for all four sides of the tiled texture.

After a tile is created it is placed in the output pixmap based on the width and height counter values that are passed into the function. Multiplying the boxheight by the counter values creates these values. Finally, after all the tiles have been created, the pixmap that is created is viewable in an output window that pops up into a new window. The image can be saved to a file if desired.

IV.1.2. Implementation for n=1

The implementation of $n = 1$ and $n = 0$ cases are very similar. When the $n = 1$ edge option is chosen, $execute1()$ is called. This function calls the function $createNew1(rowNum, colNum)$ based on the number of columns and rows that the $n = 1$ case needs. In this case, $n = 1$ needs to be a 4X4 matrix of tiled textures, so the number of rows is 4 and the number of columns is 4. Each time $createNew1()$ is called a new tile is created. The name of the tile is created based on the information passed to the function. This information is stored in the corner[] array. Each corner is given a value that represents the value of the edge.

The main difference with the n=0 case is that there are 2 boundary images instead of one. These two boundary images are chosen from the red overlay. They are stored in an array of pixmaps called squares[]. All matching edges are rotated so that they are stored facing the bottom edge of the pixmap. When they are called to be used the matching edge is rotated so that the edge that is matched is always be facing outwards. This can be done by rotating 90 degrees for each edge away from the bottom edge (see the arrows in Figure 13).

IV.1.3. Implementation for n=2

The $n = 2$ case is implemented in the same way as the $n = 1$ case except that there are four boundary images instead of one and instead of 16 possible tiles there are 64 possible tiles. The four boundary images are stored in an array of pixmaps. The procedure $execute2()$ is called when the execute button on the side bar if the choice is $n = 2$. The four boundary images have two matching edges. The boundary images are stored face down in the pixmap with the matching edge rotated so that is faces the bottom of the pixmap (see Figure 16).

IV.2. Wallpaper Mode

IV.2.1. Implementation for Tiled Wallpaper

A tiled wallpaper uses the same function to create the tiles as the n=1 function. In this function one tiled texture with the name 0011 is created. With this value of the tiled texture the edges will always match each other and the tile can be repeated with no worries of seams.

IV.2.2. Implementation for Reflective Wallpaper

The reflective wallpaper was discovered when creating the $n = 0$ textures. Because the patterns that are created by simply reflecting a triangle 8 times often create beautiful images, the option was added to the interface. The reflective wallpaper is created by simply storing the reflected image in a pixmap and repeating it 16 times in a 4X4 output image.

IV.3. Implementation of Interior Blending Algorithm

For all of the cases a method to blend the boundary images with the background images was developed. At each edge a blending algorithm is used to blend the boundary region with the background region. The background image is blended with the boundary image using functions that produce ratios. Threshold values are variable that alter these ratios. There are four different threshold ranges, determined from the slider values that are input from the interface. Ultimately a value from the variable $colormix$ determines how much of the boundary image is mixed with the background image.

As described in the III.5 in Figure 17 the functions f_1, f_2, and f_3 are used in blending the 2 diagonal edges and the straight edge of the boundary image into the background image. The f_1 function determines the distance that it will take to blend the boundary in

with the background image. In Figure 17 lines marked a and b show where this blending takes place. Similarly functions f_2 and f_3 have a range where a ratio is calculated to determine the amount of each image that will be blended together. Function f_4 is based not on the boundary, but on the distance of the colors of the background and the boundary image that is being compared. A function $distancecompute$ is used to calculate the color differences.

The $distancecompute$ function determines the red, green, and blue values for the final output value. In this function values from the background square and from the boundary square are compared. A ratio is calculated to determine the distance in color using the equation:

ratio = abs(r0-r1) + abs(g0-g1) +abs(b0-b1);

The ratio value for the distance is returned and used as one of four ratios to create a value $colormix$. The four ratios are determined using the functions values. The four functions are calculated for each pixel in the boundary image, using the threshold value as a variable that is input through the sliders (see Figure 17 for the function equations). The functions are multiplied together to create a $colormix$ value. If the $colormix$ variable is 1, then the boundary image is the pixel used for the output image. If the colormix value is 0, then the background image pixel is used. If there is a value in between 0 and 1, then the value is used as a percentage to compute the amount of background and boundary image color for the output pixel.

CHAPTER V

RESULTS

To obtain results, three steps must be taken. First a tiled texture is created. Next the three dimensional model is opened in DLFL and assigned tiled texture coordinates. Finally, the models with the reassigned texture coordinates are imported into Maya or other 3D software packages.

In this example three tiled textures have been created all of a burlwood texture. Tiled textures of case n=0, n=1, and n=2 have been created. Figure 20 shows the resulting textures.

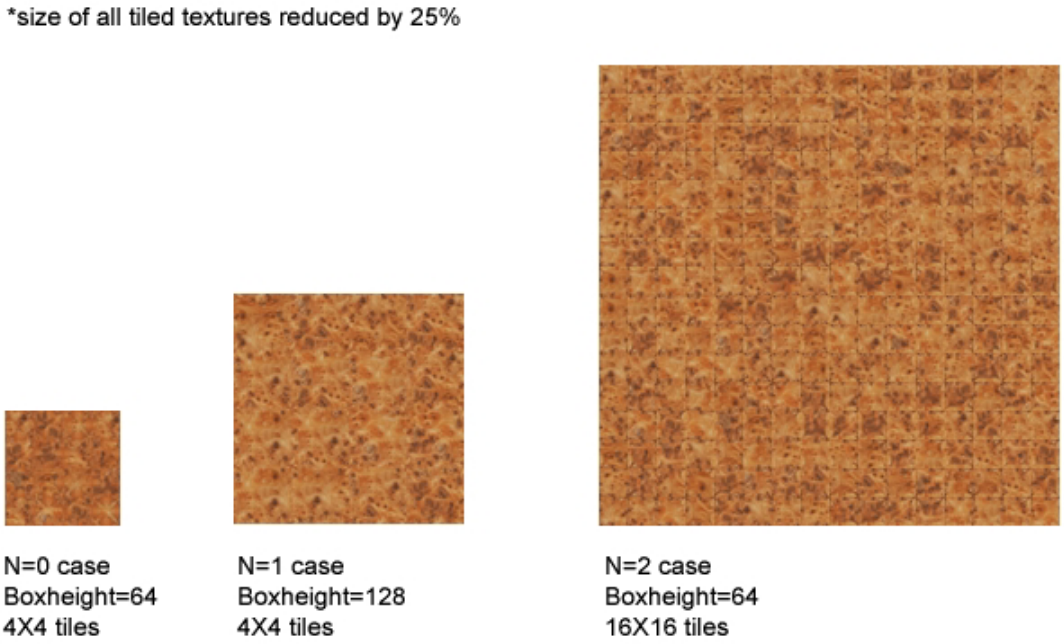


Fig. 20. Tiled textures of cases n=0, n=1, and n=2.

Next the models are taken into DLFL. Two different models are saved. One model is

saved with coordinates that are created for the $n = 1$ case, and one is saved with texture coordinates for the $n = 2$ case. The model that is made for the $n = 1$ case is used for texturing the $n = 0$ case, since the edges of tiles will match in any case.

Finally, the textures are assigned to the models and images are rendered. Figure 21 shows the examples of all three tiled texture cases mapped to a 3D object.

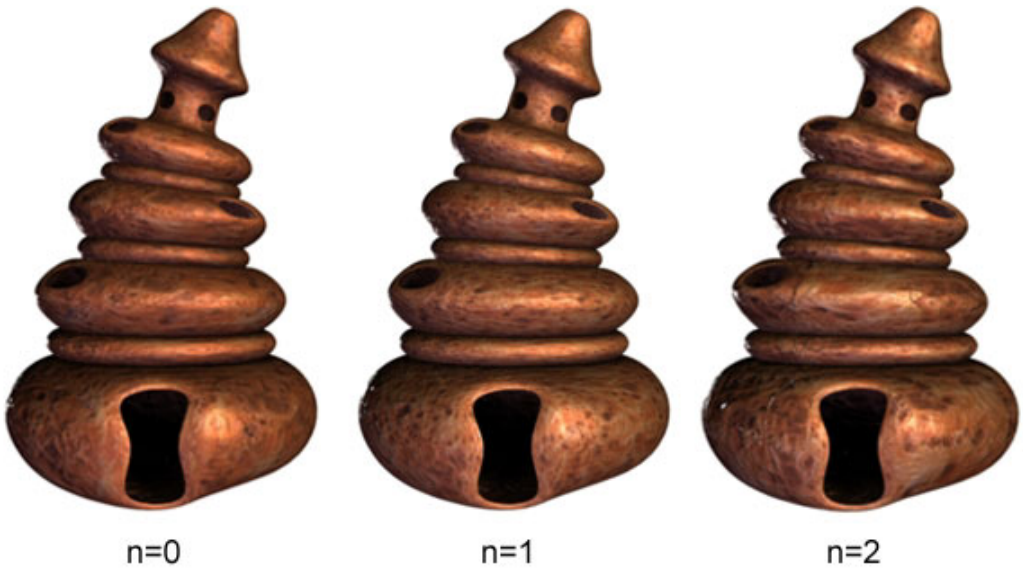


Fig. 21. Results of cases n=0, n=1, and n=2 mapped on a 3D object.

The usability and results of the program were tested in a graduate level computer graphics course. 25 students with diverse backgrounds including art, architecture and computer science took the course. All the students, regardless of their background, were able to successfully texture a wide variety of very high genus quadrilateral meshes with a wide variety of textures.

Some examples of student work are shown in Figures 22, 23, 24 and 25.

To further test and apply the work, an animation has been created. The animation idea was drawn from themes of surrealism. With a surrealistic and whimsical theme, even tex-

Fig. 22. Example 1. Work of Jeff Alcantara. Top: A tiled texture created from a photograph of rusted iron. Middle: An object textured by above tiled texture. Bottom: A detailed view that shows how the texture is mapped. Here the same texture is also used as a bump map.

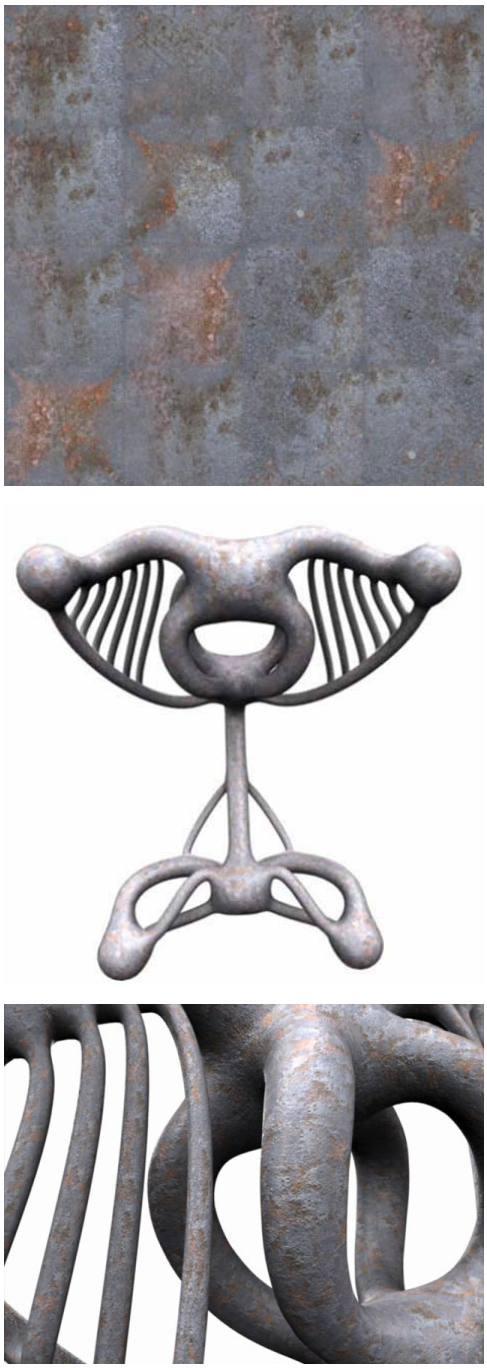


Fig. 23. Example 2. Work of Jeff Alcantara. Top: A tiled texture created from a photograph of rusted iron. Middle: An object textured by above tiled texture. Bottom: A detailed view that shows how the texture is mapped. Here the same texture is also used as a bump map.

Fig. 24. Example 3. Work of Charu Sharma. Top: A tiled texture created from a photograph of a rusted metal surface. Middle: An object textured by above tiled texture. Bottom: A detailed view that shows how the texture is mapped.

Fig. 25. Example 4. Work of John Truong. Top: A tiled texture created from a photograph of rusted iron. Middle: An object textured by above tiled texture. Bottom: A detailed view that shows how the texture is mapped.

tures that are not realistic are acceptable. Figure 26 and Figure 27 show concept drawings of the piece.

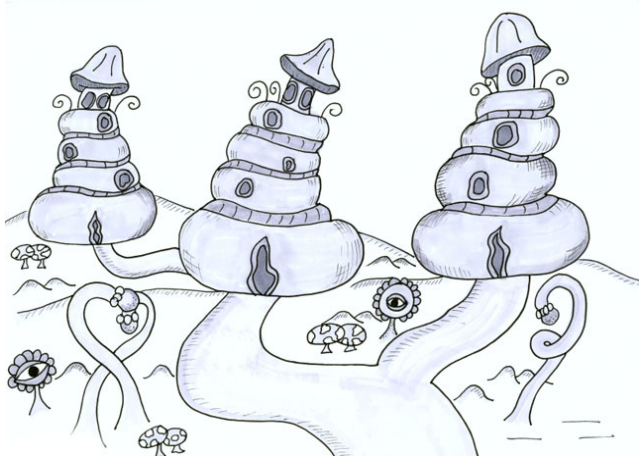


Fig. 26. Concept drawing 1.

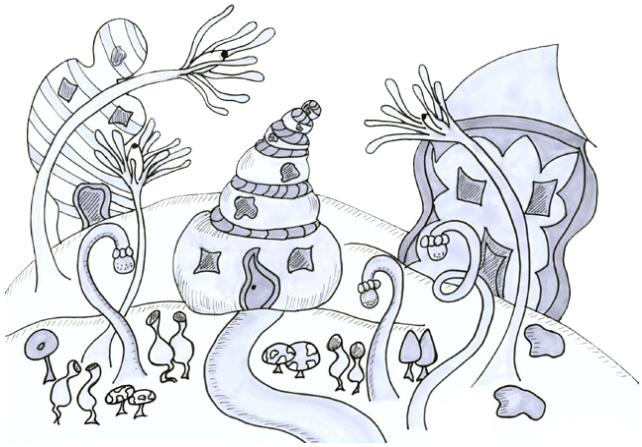


Fig. 27. Concept drawing 2.

The piece has simple geometric models, but relies on texturing and lighting to create complexity in the scene. Tiled texturing images were used in the scene along with other texturing methods that include Photoshop and Maya. Figures 28, 29, 30, 31 are stills from the animation.

Fig. 28. An image from shot 1 of the animation.

Fig. 29. An image from shot 2 of the animation.

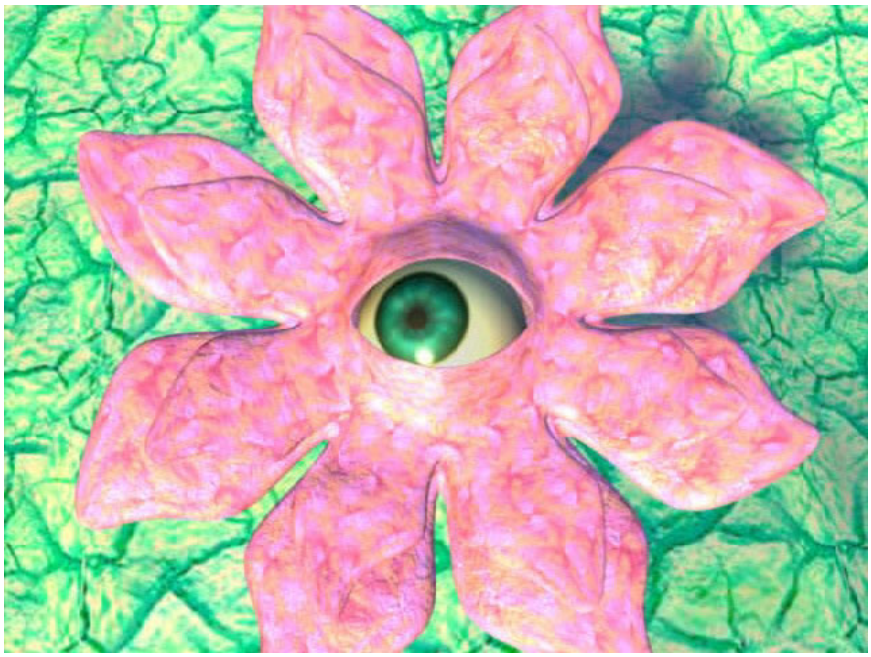


Fig. 30. An image from shot 4 of the animation.

Fig. 31. An image from shot 5 of the animation.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

In this research a simple tool to create a tiled texture from any image has been created. The program facilitates easy texture creation with $n = 0$, $n = 1$, and $n = 2$ boundary conditions. The program allows the user to choose the boundary regions, the background regions, and blending options of the tiled texture, allowing for some artistic choices. Textures can be used not only for color mapping of objects, but also for reflection, transparency, bump, displacement and many of types of mapping. The $n = 2$ case allows for anisotropic textures, although the texture mapping algorithm will have to be changed in order for that to work. The program also allows for creation of some simple wallpapers.

Because this tool creates seamless 2D textures it proves to be very useful for creating aperiodic textures. The textures the software creates have small file sizes making them easy to animate. Unlike 3d textures that often give similar results, there is no worry about swimming textures. The tool is easy to use and accomplishes the desired results, and is therefore a successful application.

Although there is some user control, the results are still somewhat random, and there is a need for more control over final results. Promising future extensions would be to develop a painting approach for tiled textures, and to provide more precise control over the background image for each tile. Perhaps this could be turned into a Photoshop plug-in that could take advantage of many of the functions of Photoshop while still maintaining the requirements of the tiled texture.

Another extension would be to create textures for the Neyret and Cani technique of mapping textures. This texture mapping algorithm uses triangles instead of squares to map the textures, so a new system of texture synthesis would have to be developed. This

idea could be extended further to include texturing for many different shapes including pentagons, hexagons, and octagons. Perhaps seamless textures could even be created for a surface with many different shapes.

Animated textures would be another interesting extension to this application. The program could be used to create textures that animate across the surface while still maintaining seamless textures. More wallpaper patterns could easily be added as well. A wallpaper with more boundary types and more background images would provide more variation and provide a more aperiodic looking image.

REFERENCES

[1] M. Ashikhmin. Synthesizing natural textures. In *Proceedings of 2001 ACM Symposium on Interactive 3D Graphics*, pages 217-226, March 2001.

[2] E. Akleman and A. Kaur. Tiled Textures. http://www-viz.tamu.edu/faculty/ergun/research/index.html.

[3] C. Bennis, J. Vezien, and G. Iglesias. Piecewise surface flattening for non-distorted texture mapping. In *Proceedings of ACM Siggraph 1991*, Computer Graphics Proceedings, Annual Conference Series, pages 237-246, July 1991.

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

[5] E. Catmull and J. Clark. Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes, *Computer Aided Design*, 10:350-355, September 1978.

[6] M. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. In *Proceedings of ACM Siggraph 2003*, Computer Graphics Proceedings, Annual Conference Series, pages 287-294, July 2003.

[7] A. Efros and W. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of ACM Siggraph 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 341-346, July 2001.

[8] P. Firby and C. Gardiner. *Surface Topology*, Ellis Horwood Limited, West Sussex, England, 1982.

[9] K. Fleisher, D. Laidlaw, B. Currin, and A.Barr. Cellular Texture Generation. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 239-248, September 1995.

[10] A. Glassner. Aperiodic tiling, part 1, *IEEE Computer Graphics and Applications*, 18(4):83-90, May/June 1998.

[11] S. Gobron. Cracked pattern based on 3D surface cellular automation. *Computer Graphics International*, pages 153-162, June 19-24, 2000.

[12] B. Grunbaum and G. Shephard. *Tilings and Patterns*. W.H. Freeman and Company, New York, 1987.

[13] P. Hanrahan and P. Haeberli. Painting and texturing of 3D shapes. In *Proceedings of ACM Siggraph 1990*, Computer Graphics Proceedings, Annual Conference Series, pages 215-223, July 1990.

[14] D. Heeger and J. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of ACM Siggraph 1995*, Computer Graphics Proceedings, Annual Conference Series, pages 229-238, August 1995.

[15] P. Litwinowicz and G. Miller. Efficient techniques for interactive texture placement. In *Proceedings of ACM Siggraph 1994*, Computer Graphics Proceedings, Annual Conference Series, pages 119-222, August 1994.

[16] B. Mandelbrot. *The Fractal Geometry of Nature.* W. H. Freeman and Co., New York, 1982.

[17] M. Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, Rockville, MA, 1988.

[18] F. Neyret and M. Cani. Pattern based texturing revisited. In *Proceedings of ACM Siggraph 1999*, Computer Graphics Proceedings, Annual Conference Series, pages 235-242, August 1999.

[19] K. Perlin. An image synthesizer. In *Proceedings of ACM Siggraph 1985*, Computer Graphics Proceedings, Annual Conference Series, pages 287-296, July 1985.

[20] D. Piponi and G. Borshukov. Seamless texture mapping of subdivison surfaces by model pelting and texture blending. In *Proceedings of ACM Siggraph 2000*, pages 471-488, July 2000.

[21] K. Sims. Artificial evolution for computer graphics. In *Proceedings of ACM Siggraph 1991*, Computer Graphics Proceedings, Annual Conference Series, pages 319-328, July 1991.

[22] C. Soler, M. Cani, and A. Angelidis. Hierarchical pattern mapping. *Proceedings of ACM Siggraph 2002*, Computer Graphics Proceedings, Annual Conference Series, pages 673-680, August 2002.

[23] J. Stam. Aperiodic texture mapping. Technical Report R046, European Research Consortium for Informatics and Mathematics (ERCIM). http://ercim.org/publications/technical_report/046-abstract.html, 1997.

[24] A. Turing. The chemical basis of morphogenesis, *Philosophical Transactions of the Royal Society(B)*, 237:37-72, 1952.

[25] G. Turk. Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion. In *Proceedings of ACM Siggraph 1991*, Computer Graphics Proceedings, Annual Conference Series, pages 289-298, July 1991.

[26] G. Turk. Texture synthesis on surfaces. In *Proceedings of ACM Siggraph 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 347-352, August 2001.

[27] J. van Wijk. Spot noise: texture synthesis for data visualization. In *Proceedings of ACM Siggraph 1991*, Computer Graphics Proceedings, Annual Conference Series, pages 309-318, July 1991.

[28] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of ACM Siggraph 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 479-488, July 2000.

[29] A. Witkin and M. Kass. Reaction-diffusion textures. In *Proceedings of ACM Siggraph 1991*, Computer Graphics Proceedings, Annual Conference Series, pages 299-308, July 1991.

[30] S. Wolfram. *Theory and Applications of Cellular Automata: Including Selected Papers 1983-1986.* World Scientific Publishing Co., Inc., River Edge, NJ, 1986.

[31] S. Worley. A cellular basis function. In *Proceedings of ACM Siggraph 1996*, Computer Graphics Proceedings, Annual Conference Series, pages 291-294, August 1996.

VITA

Lori Anne Green
7139 Waterman
St. Louis, MO 63130
lagreen1999@yahoo.com

Education
M.S. in visualization sciences, Texas A&M University, College Station, TX 12/03
B.A. in computer science, DePauw University, Greencastle, IN 5/97

Research Interests
Texture Synthesis
Lighting and Shading Techniques

Employment
Texas A&M University Department of Architecture, College Station, TX
Graduate Assistant, 1/02 - 5/02
Texas A&M University Department of Biology, College Station, TX
Graduate Assistant, 8/00 - 8/01
Cadre Computer Resources, Cincinnati, OH
Software Developer, 8/97 - 8/00