TEXTURE MAPPING USING TILED TEXTURES

A Thesis

by

AVNEET KAUR

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

May 2003

Major Subject: Visualization Sciences

TEXTURE MAPPING USING TILED TEXTURES

A Thesis

by

AVNEET KAUR

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)	Donald House (Member)
Jianer Chen (Member)	Phillip Tabb (Head of Department)

May 2003

Major Subject: Visualization Sciences

ABSTRACT

Texture Mapping Using Tiled Textures. (May 2003)

Avneet Kaur, B.Arch., Birla Institute of Technology

Chair of Advisory Committee: Dr. Ergun Akleman

This thesis work presents a simple and practical technique for seamlessly texturing quadrilateral meshes. Using this technique, an isotropic homogeneous texture can be mapped to any quadrilateral mesh without any discontinuity or singularity in the resultant texturing. The method involves organizing a set of square *texture tiles* that satisfy specific boundary conditions into one texture image file which is called a *tiled texture*. Based on the tiled textures, we have developed an extremely simple texture mapping algorithm that randomly assigns one tile to every patch in any given quadrilateral mesh. The mapping technique developed yields singularity free textures, regardless of the singularities existing in the quadrilateral mesh, gives seamless and continuous boundaries across textures, and provides an aperiodic and interesting look to the entire textured surface.

To my parents

ACKNOWLEDGMENTS

I would like to take a moment to thank my committee chair, Dr. Ergun Akleman for his knowledge and guidance which helped me to accomplish this thesis and led me through the difficulties in this research. Special thanks to Dr. Donald House who not only served on my committee, but also helped me to grow and to learn in the lab. I would also like to thank another committee member, Dr. Jianer Chen, for his enthusiasm and encouragement. A very special thanks to Karen Hillier, who helped me grow and see artistically. My sincere gratitude to Dr. Frederick Parke, Carol La Fayette, and Mary Saslow for their advice and tremendous help throughout the past two years.

I would also like to thank Vinod Srinivasan for his help and advise in programming in my thesis work. Special thanks to Lori Green, Jinnah and Hunter for their friendship and advise at various stages of my thesis work. Thanks to Charu Sharma for allowing me to use her 3d model of a face. Thanks to staff members Glen Vigus and Margaret Lomas for always being there when I needed help.

There are some people who I truly want to thank for their invaluable friendship. Thanks Ruchi, Vinod, Preeti, Joe and Rohan, whose friendships helped me in every aspect of my stay in TAMU. Also, I would not be able to finish this thesis without a very special person, Esan, who gave me hope, support, strength, friendship and love.

Above all I would like to thank my parents and my brother for their encouragement, understanding and unconditional love which has helped me all through my life.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

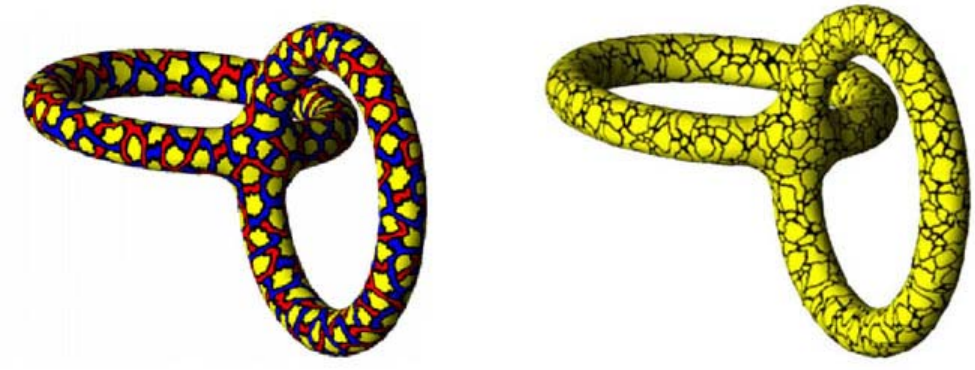
INTRODUCTION

I.1. Motivation

Shape and texture are two primary characteristics of the objects that inhabit our visual space. Since every detail of the real world cannot be modeled at the geometric level, Computer Graphics practitioners often use textures in order to give the fine detail of explicit modeling. Traditionally, texture functions have been defined on the two-dimensional surface coordinate systems of individual surface patches. A texture map can modify any of the surface characteristics of a Computer Generated object.

Most of the surface textures we need for texturing natural objects are homogeneous, i.e., their appearance is similar across the entire surface of the object. Homogenous textures can be classified into primarily two groups: Natural and man-made. Examples of natural homogenous textures are, patterns on the barks of trees, spots and stripes on animals, flowers, vegetation and rocks on distant landscapes. Examples of arbitrary meshes with aperiodica mapping using homogeneous textures can be seen in Figure 1 and Figure 2. Man-made homogenous textures may include fabric patterns and stone patterns on walls as seen in Figure 3.

The journal model is *IEEE Transactions on Visualization and Computer Graphics.*



(A) (B)

Fig. 1. Examples of aperiodic texture mapping using homogeneous textures.

(A) (B)

Fig. 2. Examples of aperiodic texture mapping using homogeneous textures.

Fig. 3. Examples of homogeneous textures patterns.

I.1.1. Texture Synthesis Techniques

The methods that are used for homogenous texture synthesis include the following

1. Procedural textures: These texture are created by using a set of algorithms. For example the “Lapped Textures” method used by [16].

2. Painted textures: These texture are painted by artists. For example by using a 2D-painting software. Most of the textures designs used by the method of texture mapping, presented in this work have been created likewise.

3. Photographed textures: These textures are obtained by photographing an existing texture and scanning the photograph. For example scanning a texture pattern of spots on a leopard, from a close up photograph of leopard skin.

I.1.2. Texture Mapping Techniques

Texture mapping techniques can be predominantly classified into three types:

1. 2D-Texture Mapping: The texture is defined on a two dimensional domain (typically a square) and mapped onto a surface. This texture is then applied on an object's surface. The image exists in a parametric coordinate space called the texture space. Texture mapping assigns a coordinate in the texture space to each vertex of the 3D object . We shall discuss this method a little more in the subsection below.

2. Solid Texture: In the second method the texture is a function called a solid texture defined for each point in space. Solid textures calculate their results based upon the shading point's position in 3-D space. Solid textures are able to generate arbitrarily complex and coherent patterns. Shaders that implement solid textures include the marble or wood color shaders. Solid textures create the appearance that an object has been carved out of a solid block of material. The visual appearance at any points on the object's exterior are determined by intersecting the object's surface with the solid material.

3. 3D Texture Synthesis: In the third group the texture is "grown" on the surface itself for example by using "Reaction-Diffusion Process". "Reaction-Diffusion" is a process in which two or more chemicals diffuse over a surface and react with one another to produce stable patterns. Reaction-diffusion can produce a variety of spot and stripe patterns, much like those found on many animals.

Among these the most popular method is 2D-texture mapping since it gives maximum freedom to the Computer Graphics artists. However, despite its power, mapping textures is still a challenge for most Computer Graphics artists. There are three main challenges the artists faces when mapping homogenous textures to arbitrary polyhedral meshes:

1. Avoiding noticeable seams between texture patches,
2. Avoiding singularities even when underlying mesh has singularities as seen in Figure 4, and
3. Minimizing the amount of stretching/distortion of the texture.

Fig. 4. Examples of textured meshes having distortions and singularities.

Neyret and Cani [11], developed a 2D homogeneous texture mapping method to provide an effective solution to first two problems. Their method is based on triangles. It avoids the singularities regardless of the topology of underlying triangular mesh. Their method also provides seamless and continuous boundaries across the surface.

In this work the goal is to develop a new 2D homogeneous texture mapping method that is inspired from the Neyret-Cani method. Unlike the Neyret-Cani method, the method presented in this thesis work is based on quadrilaterals. The method presented also avoids the singularities in the textured mesh regardless of the topology of underlying quadrilateral mesh. It also provides seamless and continuous boundaries across the surface. It also minimizes the distortions across the geometry, but that is not primarily due to the algorithm used in the method presented. Most importantly, unlike Neyret-Cani method, the method in this thesis work is compatible with existing commercial software (example Maya, Softimage etc). Also, instead of using several texture files as in their method we use only one texture file that consists of a grid made up of texture tiles. This makes the method simple and efficient to implement.The texture mapping algorithm is extremely fast. In addition, once texture coordinates are computed, these coordinates can be used for any tiled texture image, i.e., there is no need to recompute texture coordinates for different textures. The texture image files we create, consist of a set of square shaped tiles. Consistent organization of these tiles in the texture image yields an extremely simple texture mapping algorithm. The algorithm assigns texture coordinates to each corner of the mesh while providing seamless and continuous boundaries across the surface. Once the texture coordinates are assigned, the resulting textured mesh can be used in any commercial modeling or animation system such as Maya or Softimage. Figure 5 shows reproductions of Miro and Kandinsky paintings, used to create texture tiles. Figure 6 shows these texture tiles mapped to a surface of a sphere, using the method presented in this work.

Fig. 5. Reproductions of (left) Miro and (right) Kandinsky paintings that are used to create tiled textures in Figure 6.(These reproductions are from www.artsunframed.com).

Fig. 6. Mapping two tiled texture images created from Miro and Kandinsky paintings to a spherical shaped mesh.

CHAPTER II

BACKGROUND/PREVIOUS WORK

II.1. Texture Mapping

Texture mapping is a successful technique for high-quality image synthesis. It creates rich visual illusions inexpensively, and allows mapping a texture image onto a surface in a 3D scene.

II.1.1. Standard 2D-Texture Mapping

2D-Mapping involves the mapping of a single image of a desired texture onto a synthetic object. In order to carry out 2D Texture Mapping, a global parameterization of the object surface is required. As a consequence, there will necessarily be discontinuities of the texture somewhere on the surface if the object is closed or has a higher topological order. Moreover, the texture may be highly distorted if the object has an arbitrary geometry. Optimization techniques such as those in [8] can be used to reduce distortions, either locally, or by allowing the introduction of cracks', i.e., discontinuities. Entirely suppressing distortions by editing the mapping is impossible, except if the object's surface can be unfolded onto a plane (such as a cloth). This is not the case for natural shapes. A solution for the user to eliminate apparent texture distortions is to draw a pre-distorted texture that will compensate for the distortions due to the mapping. However, this requires high designer skills, and the work needs to be re-done from scratch for every new object. An alternative is to use pattern-based texture mapping, which consists of repetitively mapping a small rectangular texture patch representing a tile of the desired texture pattern onto the surface. The tile image has to obey specific boundary conditions in order to join correctly with it-

self. More precisely, it needs to have a toroidal topology: the texture on the left edge must fit the texture on the right, and respectively the top edge has to fit with the bottom. Such texture tiles can be created by editing pictures or drawings using interactive 2D painting systems. An advantage with respect to the previous approach is that, being small, the texture tile will be stored at a higher resolution, and will demand less redundant work by the artist. Moreover, it can be re-used for texturing other objects. Discontinuity and distortion problems, however, will be exactly the same as for a single texture map as long as a global parameterization is used to map the texture pattern.

II.1.2. Interactive Techniques of Texture Mapping

The problem of finding good local parameterizations for the surfaces is solved in patch-based interactive texturing systems by leaving the user to tile the surface [13]. In [13], the latter interactively subdivides an implicit surface into square patches. Surface geodesics are used for fitting the borders of these patches to the surface. Optimization is then used for deriving a minimally-distorted local parameterization inside each patch. This approach, which can be extended to parametric surfaces as well, can be combined with pattern-based texturing in order to cover an object with a given pattern. However, using a local instead of a global parameterization is not sufficient for avoiding texture discontinuities on closed surfaces (to be convinced, try to map a texture tile with a toroidal topology onto a cube): texture discontinuities will appear across some of the edges, since the neighboring borders of the tile image cannot be those expected everywhere. Entirely avoiding both distortions and discontinuities can be achieved by using interactive texture painting software. As in the first method, a single texture map corresponding to a global parameterization of the surface is used. However the texture content is directly designed on the object's surface before being stored as a map. The texture map may then appear distorted and discontinuous, but

it will be correct when it is rendered. Depending on the user's skills, a homogeneous non-periodic texture may be designed using this method. However, this technique yields a high memory cost (as in the first approach) and consumes lots of user's time since texture details must be drawn all over the surface. Moreover, the user work is almost never re-usable on another shape.

II.1.3. Texture Synthesis Techniques

An alternative to painting the texture onto the surface is to automatically generate it, which has the advantage of saving user's time by replacing the redundant design work by a high level control of the texture features. A wide range of parametric texture synthesis techniques that are convenient for generating natural textures have been proposed. One such method is solid texturing, which involves defining a 3D material field (e.g. marble, wood) which is intersected with the object's surface to create the texture [23] and [15]. No distortion nor discontinuity across the object's edges will be produced, since no surface mapping occurs. However the method is restricted to texture patterns that intrinsically come from a 3D phenomenon: it cannot capture surface properties such as the detailed appearance of the skin (e.g. regular scales). Another drawback is that synthesizing the texture during rendering will not allow real-time performance, since it is a per-pixel based computation. An alternative would be to store 3D texture tables at a high memory cost. Other procedural techniques such as reaction diffusion [22], can be used to generate a pattern-based texture directly on an object's surface. These methods are computationally costly or have a high memory cost, depending whether the texture is generated on the fly or precomputed and stored. We can note that Perlin's and Worley's techniques may also be used on surfaces (as opposed to solid material). However Perlin's noise requires a grid to be generated, so a global parameterization needs to be introduced. Lastly, all the listed procedural techniques

can easily be extended to the automatic generation of square 2D texture tiles that have a toroidal topology. The latter can then be used in pattern-based as birds, fishes or reptiles. Penrose studies aperiodic tilings of the plane, and shows that some specific sets of tiles always form non periodic patterns.

II.2. Texture Mapping Techniques Used in the Past

Texture mapping was first introduced by Catmull, as a method of adding visual richness to a computer generated scene without adding geometry [4]. He used an approach which defined mapping from a rectangle to the natural coordinate system of the target object's surface. He used latitudes and longitudes to define mapping onto a sphere, and used parametric coordinates when mapping a texture onto a cubic patch. In some cases the object might be covered by multiple patches, and in these instances care must be taken to make the edges of the patches match. A successful example of this can be found in the bark texture for the model of a maple tree [3].

Another approach to texture mapping was projecting the texture onto the surface of the object. One example of this is to orient the texture square in R3 (Euclidean three space), and perform a projection from this square on the surface [12].

Related to this is a two-step texture mapping method given by [2]. The first step maps the texture onto a simple intermediate surface in R3, such as a box or cylinder. The second step projects the texture from this surface onto the target object.

A different method of texture mapping is to make use of the polygonal nature of many graphical models. In this approach, taken by [17], the surface of a polyhedral object is unfolded onto the plane one or more times, and the average of the unfolded positions of each vertex is used to determine texture placement. A user can adjust the mapping by specifying where to begin the unfolding of the polyhedral object.

Witkin and Kass demonstrated that a wide variety of patterns can be created by using variations of one basic reaction-diffusion equation (Reaction-diffusion is a chemical process that builds up patterns of spots and stripes), and this process can be simulated to create textures [22]. Figure 7 shows examples of texture mapping by Witking and Kass.

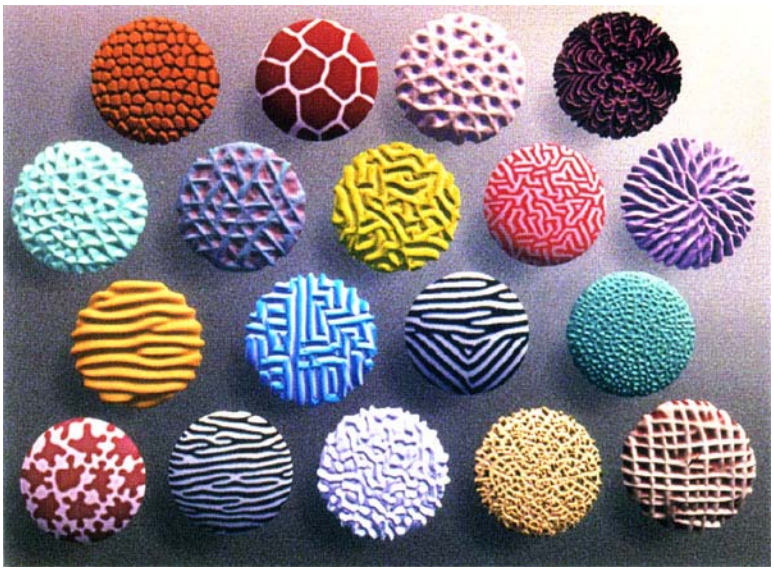


Fig. 7. Texture Mapping of 3d-models of buttons, by using the Reaction-Diffusion technique. Row 1: (a) reptile, (b) giraffe, (c) coral, (d) scalloped. Row 2: (a) spiral, (b) triweave, (c) twisty maze, (d) replication, (e) purple thing. Row 3: (a) sand, (b) maze, (c) zebra horse, (d) radial. Row 4: (a) space giraffe, (b) zebra, (c) stucco, (d) beats us, (e) weave.

In Stam's work on aperiodic textures [19], his aim is to render water surfaces and caustics. Stam tiles the plane with a standard grid of square patches, where he maps 16 different texture tiles of a homogeneous texture. To do so in a non-periodic fashion, he uses an algorithm for aperiodically tiling the plane with convex polygons of different colors (the colors of the tiles in the mathematical theory correspond to the boundaries of *tiled textures*. The boundary conditions between texture tiles are met by using the same input noise for generating the texture in the rectangular regions that surround a shared edge. This

method is restricted to applying textures onto a plane, otherwise the usual parameterization problems yielding distortions and discontinuities would appear. Moreover, the problem of synthesizing the texture tiles is only addressed for a specific texture, and the algorithm is not explicitly described. Figure 8 and Figure 9 show examples of Stam's work on aperiodic textures.

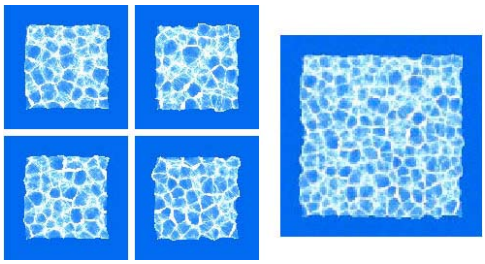


Fig. 8. For each of the 16 tiles used to create the water surface, the corresponding caustic texture was computed. The caustic map on the right is obtained by compositing the four maps on the left.

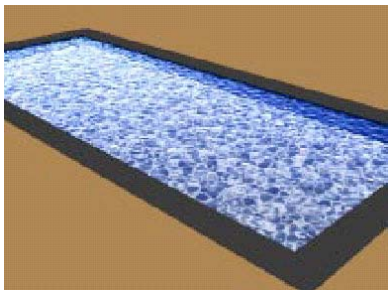


Fig. 9. Ray traced image of a pool, tiled aperiodically with the texture shown in Figure 8, on the bottom of the pool.

Turk demonstrated that a simulation of reaction-diffusion can be performed on an array of cells that have been placed over a polygonal surface. Because the simulation proceeds directly on the surface, the spot and stripe patterns of this method are undistorted and without seams [20]. Figure 10 shows an example of Turk's Reaction-Diffusion technique.

Malliot et al used a deformation tensor to describe an energy measure of distortion that they minimized over a surface made up of triangles [9]. Levy and Mallet also used a similar approach, which utilizes an energy term which minimizes distortions but also incorporates additional constraints such as a user defined curves and cuts in the surface [8]. Figure 11 shows an example of Levy's and Mallot's approach.

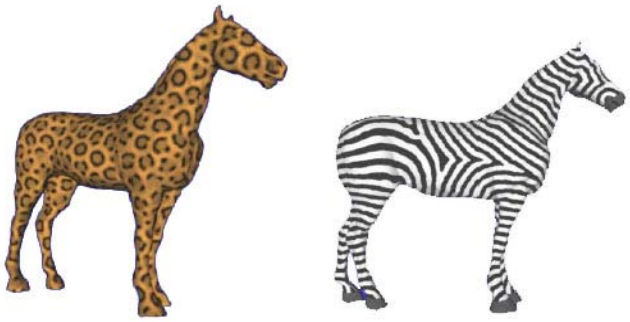


Fig. 10. Leopard horse and zebra texture patterns produced by Turk's "Reaction-Diffusion" technique.

Fig. 11. Texture mapping of a face using Levy and Malliot's "Non-Distorted Texture Mapping for Sheared Triangular Meshes".

Neyret and Cani use an algorithm that enables them to map sets of triangular textures (chosen from a desired pattern), onto a curved surface that may be closed, while meeting boundary conditions everywhere [11]. Their mapping algorithm enables them to freely tune the scale of the texture with respect to the object's geometry while minimizing distortions. Figure 12 and Figure 13 show examples of texture mapping by the Neyret-Cani method.

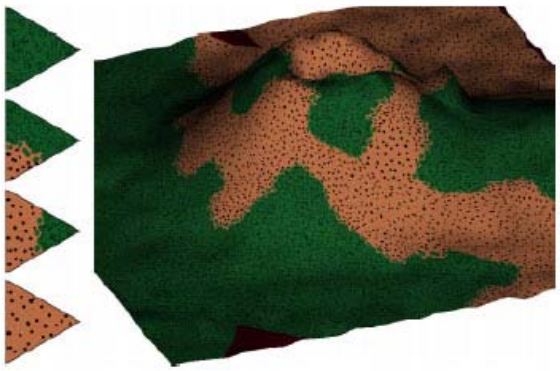


Fig. 12. Texture mapping of a mountain covered forest by using the four triangular texture tiles on the left, by the Neyret-Cani method.

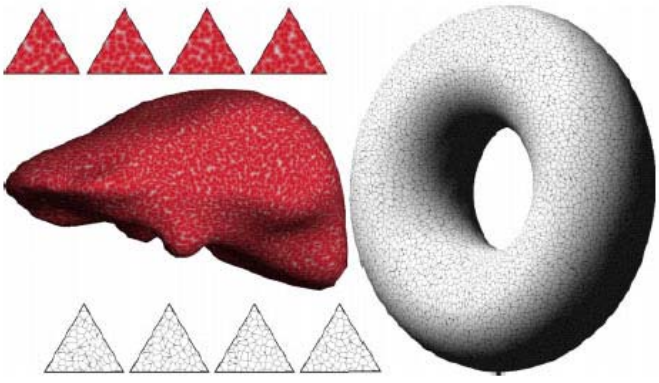


Fig. 13. Set of textures tiles created by Neyret-Cani's extension of Worley's synthesis technique and the images produced by non-periodically mapping them onto a human-liver(left) and a torus(right).

Turk's new method, [21] draws on the previous texture synthesis methods [20]. Given a texture tile in the form of an image, they create a similar texture over an irregular mesh hierarchy that has been placed on a given surface. Figure 14 shows examples of patterns generated on arbitrary meshes using Turk's texture synthesis.

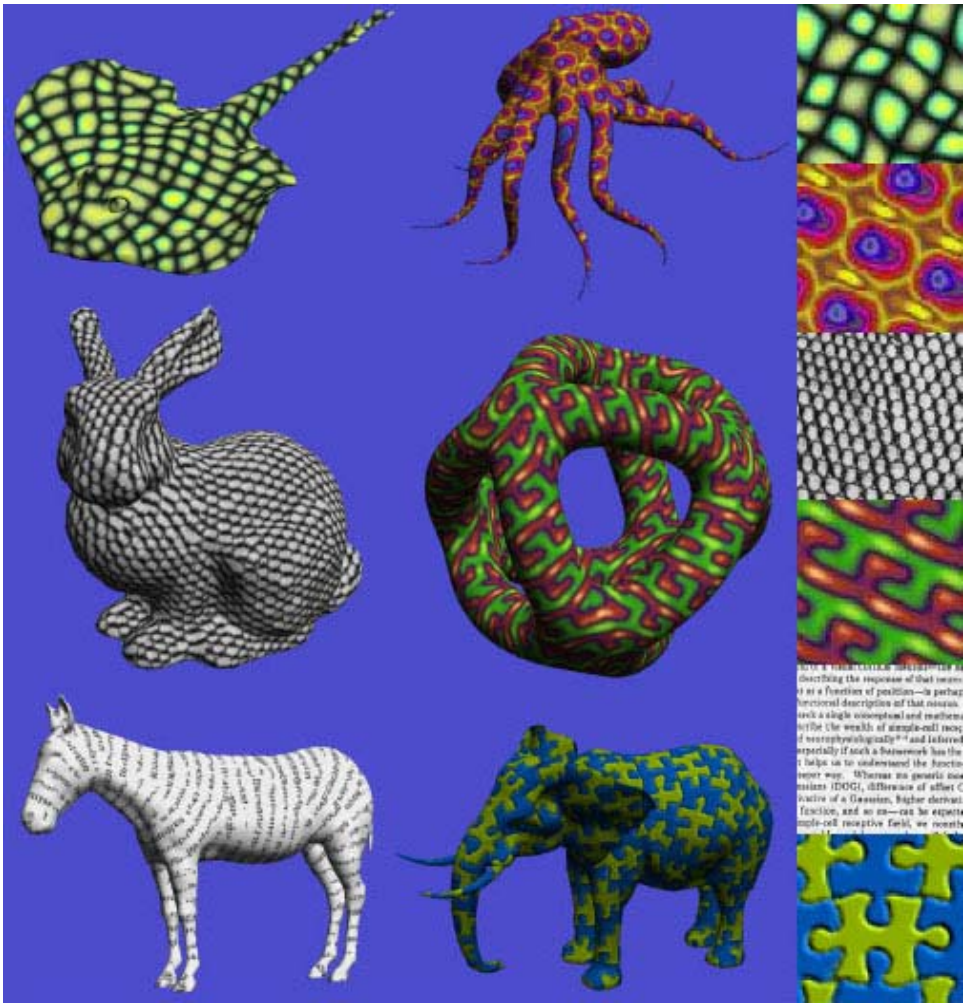


Fig. 14. Results of Turk's texture synthesis method applied onto six arbitrary meshes. Input textures shown on the right.

Praun, Finkelstein and Hoppe [16] lapped texture technique takes one or more irregularly shaped texture patches and places many copies of the patches in an overlapping fashion over a surface. The patches locally overlap and align their main features with a predefined vector field. The method generates very nice results, but is again only applicable to a specific class of textures, the latter should not be sensitive to discontinuities (in practice, the textures used have internal discontinuities), and should have no low frequency variations. As with the previous method, this one demands some specific user input (the user must define texture tiles that fit with the intrinsic texture discontinuities). Lastly, rendering requires either composition operations or the storage of a global texture map. Figure 15 shows examples of texture mapping done by using the “Lapped Textures” method.

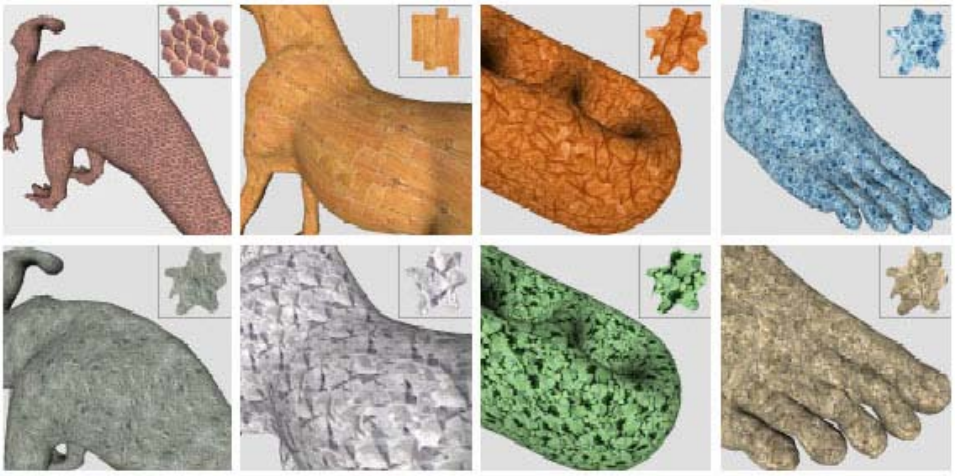


Fig. 15. Texture mapping examples by using “Lapped Textures” technique.

Soler, Cani and Angelidis have recently introduced a very powerful extension to the Neyret-Cani method [18]. They present a multi-scale algorithm for “Hierarchical mapping” of a texture defined by an input image onto an arbitrary surface. Their idea is to progressively cover the surface by texture patches of various sizes and shapes, selected from a single input image. The process starts with large patches. A mapping that minimizes the texture fitting error with already textured neighboring patches is selected. When this error is above a threshold, the patch is split into smaller ones, and the algorithm recursively looks for good fits at a smaller scale. The process ends when the surface is entirely covered. Their method correctly handles a wide set of texture patterns, both isotropic and anisotropic. Figure 16 shows examples of texture mapping using “Hierarchical Pattern Mapping” technique.

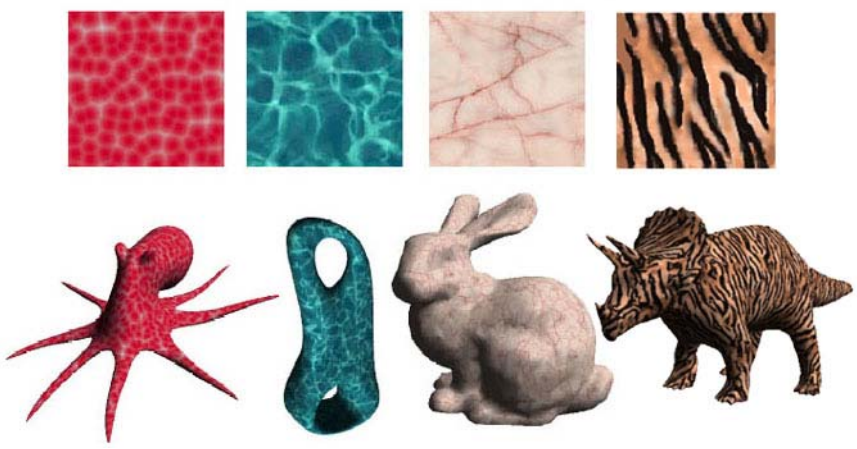


Fig. 16. Texture Mapping by [18] using Isotropic and Anisotropic patterns.

Since this thesis work is inspired by the Neyret and Cani’s work we shall describe their method in more detail.

II.3. Inspiration from the Neyret-Cani Method

Neyret and Cani's solution is inspired from Escher's work [6], since the surface was not to be tiled with square patches as usual, but rather with triangles, on which equilateral texture tiles were mapped.

Their method, 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 [19] 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.

To achieve this, their first step was to create triangular tiles that obey specific boundary conditions. They showed that if one can create a minimum number of tiles that with rotation can provide all possible tile boundaries, it is possible to seamlessly cover any triangular mesh with these tiles. The texture tiles they created, can be classified into two cases, on the basis of the specific boundary conditions, they obey. Texture tiles having similar boundary conditions on all three sides of the triangles and texture tiles having dissimilar boundary conditions on two or more sides.

Case 1: Texture tiles with similar boundaries.

In this case they created texture tiles that had similar boundary conditions on all the three side of each tile that they would use, such that the two patterns in the neighborhood of the border separating the two patches, fit perfectly with each other. Figure 17 shows an example of the triangular texture tiles they used, and how they were mapped onto a mesh made up of triangular patches, such that the boundaries look continuous and seamless across the entire surface of the polygonal mesh. Note however, that in order to use these texture tiles, the user has to read each texture tile separately.

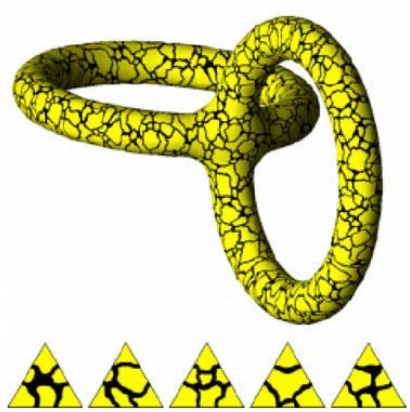


Fig. 17. *Neyret-Cani method - Case 1:* An example of texture tiles with same boundaries on all three sides, mapped onto the surface of an arbitrary mesh.

Case 2: Texture tiles with dissimilar boundaries.

In this case they used triangular texture tiles with dissimilar boundary conditions, or what they call “oriented edges”. They denoted the two different boundary conditions of the triangular patch with letters such as E and E1, such that that the texture tiles should fit on both sides of “e” as seen in Figure 18. Then, at least four texture triangles respectively obeying the boundary conditions (E; E; E), (E; E1; E), (E; E1; E1;), or (E1; E1; E1) must be provided. These four tiles form what they call the “minimal set”, as seen in Figure 19. The other possible values for the boundaries conditions (such as (E; E1 ;E) for instance) were met by a rotated instance of one of these triangles. In the general case of “n” different boundary conditions (i.e. two times the number of oriented edges), the number of texture triangles needed were

$$n + n(n-1) + n(n-1)(n-2)/3$$

The first term corresponds to the condition where the same edge was used three times, the second one to conditions with two different edge values, and the third one to solutions with three different edge values. For instance, twenty four texture triangles were needed if two oriented-edges are used instead of one (i.e. 4 boundary conditions instead of E and

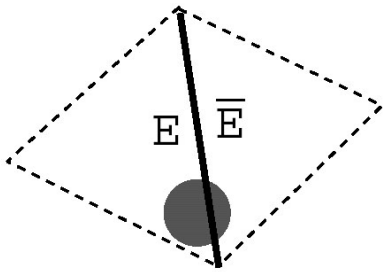


Fig. 18. Example of how an oriented edge matches in case of Neyret-Cani method.

E1). Since the number of triangles required increases as a power of 3, their algorithm is not convenient for a larger number of edges.

From a practical point of view, there are two challenges with this method for Computer Graphics applications:

1. The artist cannot automatically create tiles from existing images. In order to create homogeneous textures that look like those of natural objects, several different texture tiles have to be hand-designed, which can be extremely difficult.

2. Using a set of triangular texture images is not compatible with current texture mapping methods and requires special programming.

3. 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' method does solve the challenges mentioned above and also provides anisotropic texturing. However, their algorithm is complicated and can be extremely hard to implement for novice programmers. Although, their implementation is very efficient, it still takes several minutes to compute a texture mapping. Moreover, if the

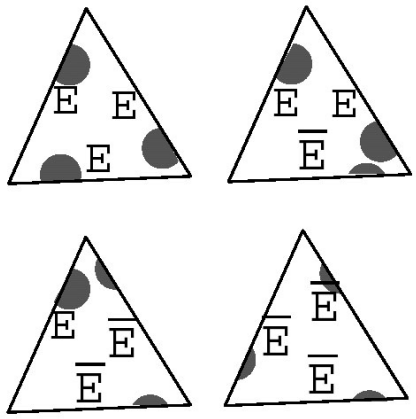


Fig. 19. *Neyret-Cani method - Case 2:* An example case of the "minimal set", ie. four triangular tiles, that are needed to be created, to fit the different boundary conditions produced in the case of oriented edges "E" and "E1" as shown.

texture image is changed there is a need to recompute the texture mapping.

CHAPTER III

METHODOLOGY

III.1. Outline of the Method

In order to create homogeneous textures that look like those of natural objects, several different texture tiles have to be designed and mapped in a non-periodic manner onto a surface. Existing commercial software for texturing (like Maya, Softimage etc), allow only a single rectangular texture map to be read and mapped onto a patch on a polyhedral mesh.

In this work, we present a texture mapping algorithm that reads all the different texture tiles, in the form of one single *tiled texture*. In order to do this, we arrange the different texture tiles, into one single *tiled texture* in a certain "matrix formation". This provides us with an easy logic in writing the algorithm, such that a *tiled texture* can be read by the texture mapping program. Texture coordinates are assigned to the patches in the arbitrary polygonal mesh, in a manner that a single texture tile, (of the many that make up the single *tiled texture*) may be then assigned to it. Assigning texture coordinates in this way, makes the method we present, compatible with existing commercial programs like Maya, Softimage etc.

The method presented in this thesis work produces, singularity free texturing, whatever be the underlying topology of the mesh below. It also yields seamless and continuous boundary conditions across the entire surface of the mesh, and most importantly an aperiodic look is achieved as in the case of homogeneous, isotropic textures. This method makes use of homogeneous texture patterns drawn on square tiles. These texture tiles can be created either procedurally or by using any 2D-painting program.

There are basically two different cases of boundary conditions that we deal with in this work. Case 1, describes tiles with similar boundary conditions and Case 2 which describes

tiles with dissimilar boundary conditions.

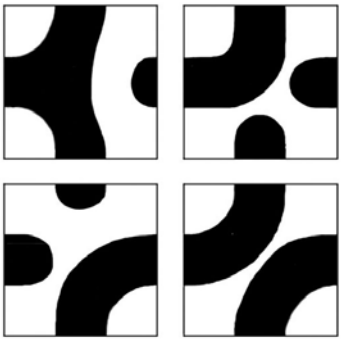


Fig. 20. *Case 1:* An example of the texture tiles we create with similar boundaries on all four sides.

III.1.1. Case I

In the first case we deal with texture tiles with similar boundary conditions on all four sides. We create homogeneous texture patterns on square tiles, either procedurally or by using a 2D-painting system. Each of this texture tile has a similar boundary condition, but a different pattern inside as seen in Figure 20. These tiles can be rotated, mirrored and when placed next to each other, form continuous and seamless boundaries across any surface mesh. We arrange the texture tiles in a certain “n x n” matrix formation. For example a “2 x 2” formation as seen in Figure 21, or “8 x 8” formation as seen in Figure 22. Here “n” stands for number of different texture tiles with similar boundary conditions(each tile may have a different pattern inside). Using this matrix formation enables us to arrange all the texture tiles in the form of one *tiled texture*, and makes it simple to write an algorithm. The algorithm we thus develop, is able to read the *tiled texture,* and each of the texture tiles is then mapped in a random manner, onto each patch on the surface of the polyhedral mesh, yielding an aperiodic looking surface which has seamless and continuous boundaries all

over. Using this algorithm we are able to generate more complex and interesting patterns, as we use many texture tiles in one single *tiled texture*. On comparing the two *tiled textures* seen in Figure 22, we observe that the two tiles have the same number and type of texture tiles, however the arrangement of the tiles is different in both, yielding different and interesting patterns.

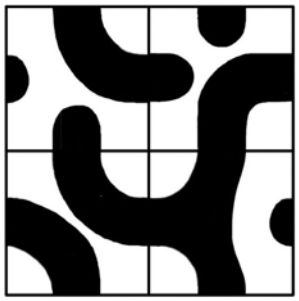


Fig. 21. *Case 1:* Example of a "2 x 2" texture tile created from the four smaller shown in Figure 20.

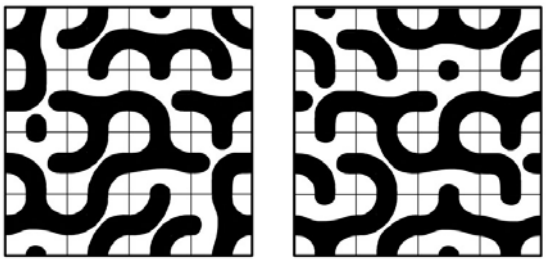


Fig. 22. *Case 1:* Examples of two "8 x 8" texture tiles, created with rotating/mirroring the tiles shown in Figure 20.

III.1.2. Case II

In the second case, the square tiles we use to create the *tiled texture*, have dissimilar boundary conditions, i.e., the boundaries are not the same on all four sides. We can also say that the tiles have "oriented edges". The problem here is to find out how many texture tiles are required in order to guarantee that the continuity constraints at boundaries are respected, and to allow sufficient variation across the texture. Most importantly, we need to arrange these texture tiles in a manner such that the program we develop is able to read them all as one single texture file.

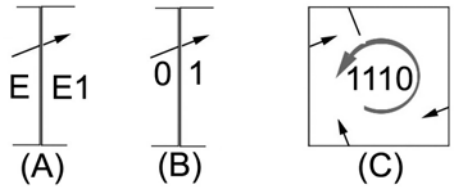


Fig. 23. Numbering the boundary types and naming the tiles with these numbers.

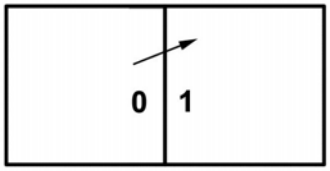


Fig. 24. *Case 2:* Example of two different boundary conditions matching in the method presented.

Neyret and Cani use labels such as "E" and "E_1" to denote boundary types. Figure 23(A) shows their labeling scheme, in which the boundary cuts through an arrow image. It turns out that if we use numbers instead of labels to denote boundary types we can organize the tiles into a texture image and develop simple and efficient algorithms for texture mapping. Figure 23(B) shows the numbering scheme we use in the method presented

in this work. As seen in this figure, we use 0 instead of "E" and "1" instead of "E_1". A counter-clockwise walk starting from the lower-left corner of a square tile gives each tile a unique name which is a four-digit binary number as shown in Figure 23(C). These names are also useful to relate the tiles to each other. A cyclic shift of the name corresponds to a 90^0 rotation of the boundaries.

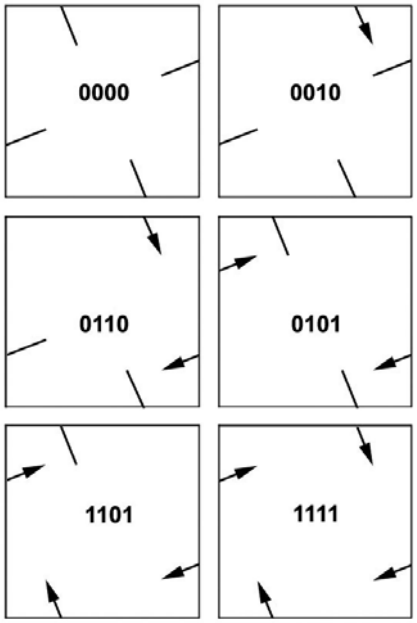


Fig. 25. *Case 2:* An example case of the six quadrilateral texture tiles, or what we call a "minimal set"(in case of oriented edges), required to seamlessly cover a quadrilateral mesh.

For the case of a quadrilateral tile with oriented edges we denote the edges as "0" and "1", such that the texture tiles should fit on both sides of an edge to form a continuous and seamless boundary as seen in Figure 24. Using these numbers we are able to assign a specific name for every texture tile we use, on the basis of the boundary conditions each one obeys. To seamlessly cover a quadrilateral mesh, we need at least six tiles, such as $0000, 1000, 1100, 1110, 1010, 1111$, as seen in Figure 25. The main problem with any such

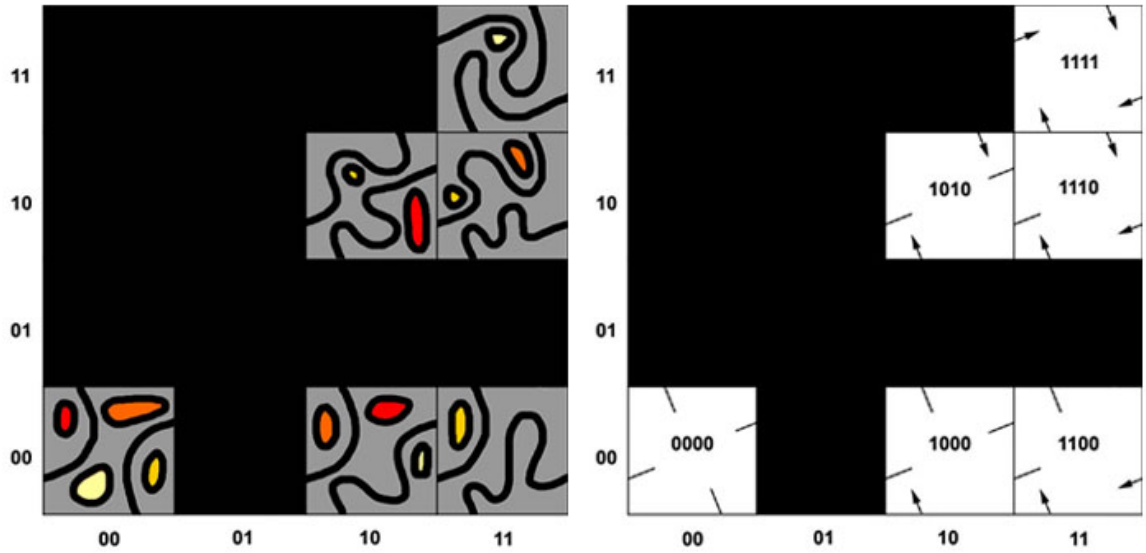


Fig. 26. *Case 2:* (A) and (B) show how the six basic texture tiles forming a "minimal set", are organized into one "4 x 4" *tiled texture* using a matrix formation. Tiles in (A) use the notion of basic arrows and numbers to denote boundary types. Note that tiles in (B) have patterns inside, but the corresponding tiles in (A) and (B) have exactly similar boundary conditions.

"minimal set" is that it is difficult to develop a simple scheme to meaningfully organize these tiles into one texture file. However, if we use all possible tiles instead of a "minimal set", the organization problem greatly simplifies and the numbering scheme we use, and allows us to organize these tiles in a texture map in a meaningful way. As shown in Figure 26 (A) and (B), we can use their 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.

The black patches in Figure 26 (A), (B), indicates that the tiles which are required to fill in that part of the *tiled texture*, can be created by merely rotating one of the existing six tiles forming the "minimal set". On comparing texture tiles in Figure 26(A) and Figure 27(A), and tiles in Figure 26(B) and Figure 27(B), we can see that the tiles with

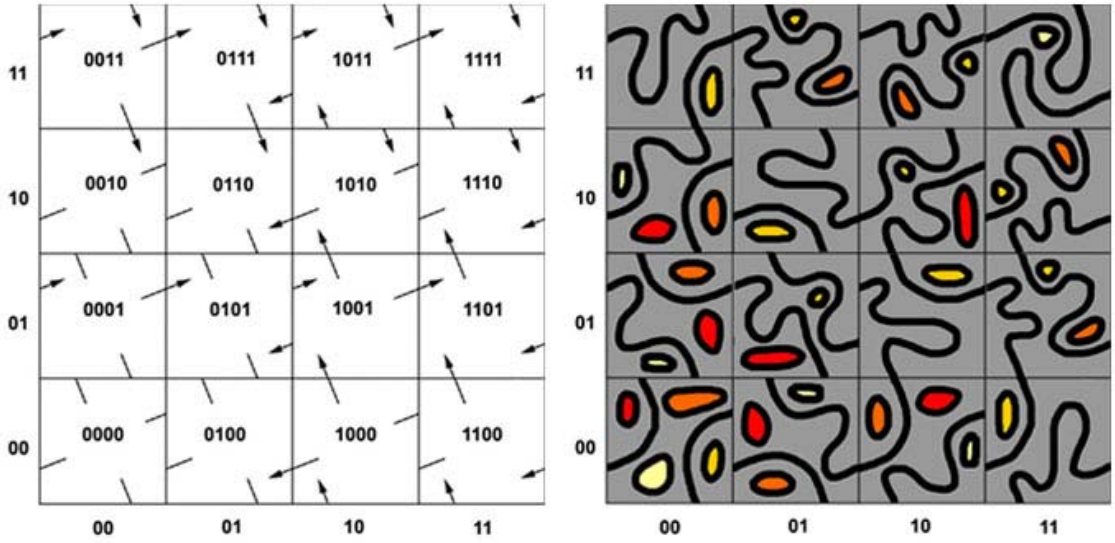


Fig. 27. *Case 2: Tiled texture* made up of all the possible tiles which have the boundary conditions “0” and “1” at the oriented edge. Tiles in (A) and (B) have boundary conditions similar to one of the six tiles in thr minimal set. Tiles in (A) and (B), sitting in the areas which fall under the black patches in Figure 26 (A) and (B), are rotated/ mirrored instances of the minimal set.

rotationally identical boundaries can be different. For instance in Figure 27(A) 1010 and 0101 do not have to be rotated versions of each other. We see that the texture tile (1; 0; 1; 0), is the same as (0; 1; 0; 1), except that the latter is a rotated instance of the former. Each of these individual texture tiles can be of a different pattern, as long as their boundary conditions remain the same. By maintaining the specific boundary conditions, (as determined by the locations of the tiles in the matrix) we can create innumerable design patterns within each specific tile of the texture tile. On comparing the corresponding tiles in Figure 27(B) and Figure 28, we see that the tiles have rotationally similar boundary conditions in both, but unlike the those in the former, the ones in the latter, have different patterns inside each one. In this manner we can use the 2D texture space more efficiently since having more variety of tiles helps to achieve an aperiodic look. Each matrix structure we form in Fig-

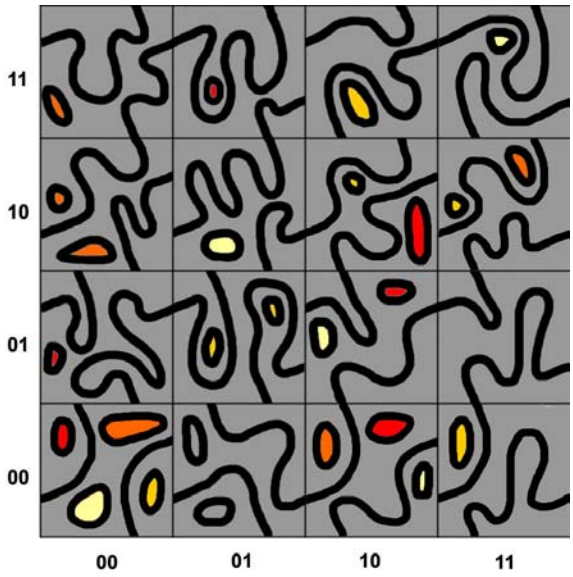


Fig. 28. *Case 2:* Example of a *tiled texture* which shows each of the texture tiles having a pattern which is different from its corresponding tiles in Figure 27 (B), but all the corresponding tiles in both the Figures have similar boundary conditions.

ure 27 (A), (B), and Figure 28, is an example of a texture tile which we use as an input texture map in the mapping algorithm presented in this work. This matrix provides us with an easy logic in writing the algorithm for texture mapping. The mathematical expression for denoting any such matrix that we use can be said to be “square of n x square of n”, where “n” is the number of different boundary conditions which a texture tile may have. Thus by using this matrix formation we are able to consistently arrange all the texture tiles in the form of one *tiled texture*.

III.1.3. Tiled Texture Image Synthesis

Tiled textures in Figure 27 (A), (B), and Figure 28 are all hand-drawn. These tiles were created in a 2D-painting and drawing software. However it is tedious and time-consuming to create a wide variety of tiled texture images by hand. Therefore a simple and fast method has been developed to automatically create the *tiled textures* from any given image. This method forms an integral part of Lori Green's Master of Science thesis work [7]. The approach used to create the *tiled textures*, can be considered a variant of image quilting. To read more about how the tiled textures can be created procedurally, please refer to [7].

III.2. Texture Mapping Algorithm

Based on the convention used by us to denote boundary conditions, it is extremely easy to come up with an algorithm for mapping the tiled texture images. The algorithm, we develop assigns one texture tile from a tiled texture image to each quadrilateral, in an arbitrary mesh in such a way that all boundary conditions are satisfied. The texture mapping algorithm consists of two procedures (see Figure 29). The first procedure **AssignBoundaries(**G, n**)** guarantees that boundary conditions are satisfied. This procedure randomly assigns "0" or "1" to each half-edge [10] of every edge of the mesh by using two random variables. We use the half-edge concept since it helps to explain the algorithm more concisely. However, implementation of the algorithm does not require a half-edge data structure. The quadrilateral shape in Figure 29 shows how these boundary types can be used to find four boundary types. The name of the texture tile assigned to each quadrilateral can be found. Based on the choice of the first half-edge we can choose different tiles. For instance, let us assume that "1101" is assigned to the half-edges of a quadrilateral. If we start the boundary walk from the first edge we choose the texture tile "1101", but if we start from the third edge, then we choose the tile 0111. This choice does not effect the continuity; it only gives further randomness.

Once **AssignBoundaries(**G, n**)** is completed, four boundary types e_0, e_1, e_2, e_3 are assigned to each quadrilateral of the mesh G. Note that the sequence $e_0e_1e_2e_3$ is actually the name of the texture tile to be assigned to each quadrilateral. Since the name of each tile also gives its position in the texture image file, it is straightforward to compute these texture coordinates, as it is given in the second procedure **AssignTextureCoordinates(**G, n**)** as seen in Figure 29. The equations to compute texture coordinates turn out to be quite simple since the square tiles are placed in logically consistent locations within the *tiled texture*. In these equations we assume that the texture image is stored with the origin at the bottom-left (i.e.

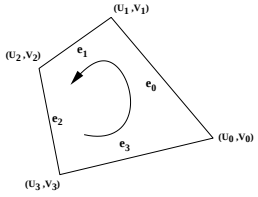
texture coordinate (0,0), width and height of texture image is one unit (i.e. the coordinate of upper-left is (1,1)), each quadrilateral is given in counter-clockwise order. Figures 30 and 31 show two examples of the results of the texture mapping algorithm presented in this work. The examples seen make use of hand drawn texture tiles created in Figure 27(A) and (B) respectively.

Subroutine AssignBoundaries(G, n**)**
for each edge e in G **do**
$i \leftarrow rand() \bmod 2$
$e_0 \leftarrow i$
$e_1 \leftarrow (i+1) \bmod 2$

Comment: e_0 *and* e_1 *are two integers representing boundary types of two half-edges of* e

Subroutine AssignTextureCoordinates(G, n**)**
for each quadrilateral f in G **do**
for $\forall i \in \{0,1,2,3\}$ **do**
$j \leftarrow \lfloor ((i+1) \bmod 4)/2 \rfloor$
$u_i \leftarrow (2e_0 n + e_1 + \lfloor j/2 \rfloor)\,/(4n^2)$
$v_i \leftarrow (2e_2 n + e_3 + \lfloor i/2 \rfloor)\,/(4n^2)$

Comment: u_i, v_i *are texture coordinates of each corner of* f *and* e_i *are boundary types assigned to* f *where* $i = 0, 1, 2, 3$.
Relative positions of the texture coordinates and boundary types of half-edges are shown in the example below.



Algorithm TiledTextureMapping(G, n**)**
AssignBoundaries(G, n)
AssignTextureCoordinates(G, n)

Comment: G *is a quadrilateral manifold mesh and* n *is the number of boundaries.*

Fig. 29. Texture mapping algorithm.

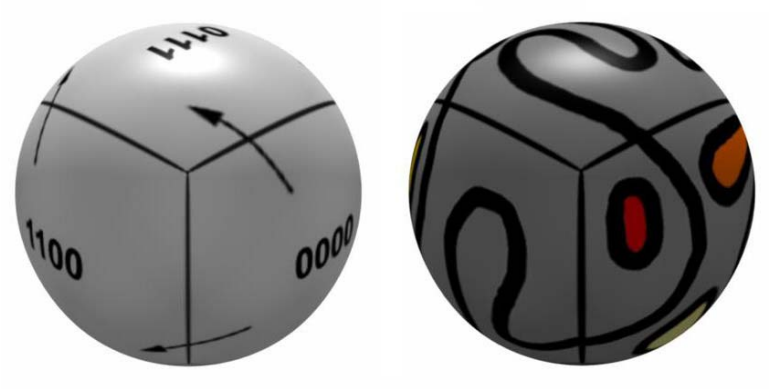


Fig. 30. A simple texture mapping example. Here all vertices are valence 3. We have applied the texture to a cube and then smoothed the cube using Catmull-Clark subdivision [5] in Maya.

Fig. 31. This example shows a genus-2 surface that includes valence 3, 4, 5 and 6 vertices. The original shape is also smoothed by Catmull-Clark subdivision [5] after mapping the texture.

CHAPTER IV

IMPLEMENTATION

IV.1. Implementing the Algorithm

The texture mapping algorithm is implemented in C++ and included in an existing 2-manifold mesh modeling system “DLFL mesh modeling system”(DLFL is a type of data structure used by this mesh modeling system and stands for stand for Doubly Linked Face List) as an option. We extend the capabilities of the DLFL modeling system [1], to incorporate the algorithm we have developed. Both systems currently run on SGI-Iris, Linux and Windows platforms. All of the interactive examples we have produced were run on an SGI-O2. All of the examples in this work were created using these two systems. The modeling system allows the creation of models with texture coordinates and the resulting mesh can be exported to any commercial software package using the classical “object” file format. The required texture coordinates are then assigned to the mesh, using the texture mapping algorithm that we have developed. A texture map is then assigned to the mesh, either in the DLFL program itself, or by exporting the mesh to any professional texturing and rendering software. The unique nature of the algorithm we present, allows us to read one single texture map(which is made of a specific pattern of texture tiles), in order to carry out the texture mapping. This is what makes the texture mapping method presented in this thesis work compatible with any existing commercial software, as all the commercial software are able to read only a single image at one time, into a shader parameter(e.g. texture mapping, bump mapping displacement mapping etc.) of any polygonal mesh. Each quadrilateral patch on the surface of the polyhedral mesh is assigned one of the many texture tiles from the single *tiled texture* that is read by the mapping program. Each texture tile in the *tiled texture*, has a specific name and rotation order. It is this name and order of each tile,

which is referenced by the texture mapping algorithm at the time of assigning texture co-ordinates. This ensures that when the texture coordinates are assigned to adjacent patches on the polyhedral mesh, they work such that the boundaries across all patches appear continuous and seamless. Since we also use a random function in the algorithm, in order to choose and assign these texture tiles to the respective patches in the mesh, this method also provides an aperiodic and interesting look to the entire surface after texturing.

IV.2. The User Interface

The User-Interface for the Texture mapping program was developed as an extension to the existing User-Interface of the DLFL Mesh Modeling system by [1]. The texture tiles may be created procedurally or can be hand-drawn. Texture tiles can be created procedurally using the Texture synthesis program developed by [7]. The User-Interface for the Texture Synthesis was developed by [7]. Following few pages have screen captures of the User Interface, demonstrating how texture mapping processes are carried out. Figures 32, Figures 33, Figures 34, Figures 35, Figures 36, Figures 37, and Figures 38 are all examples of the user interface created for the texture synthesis and texture mapping program.

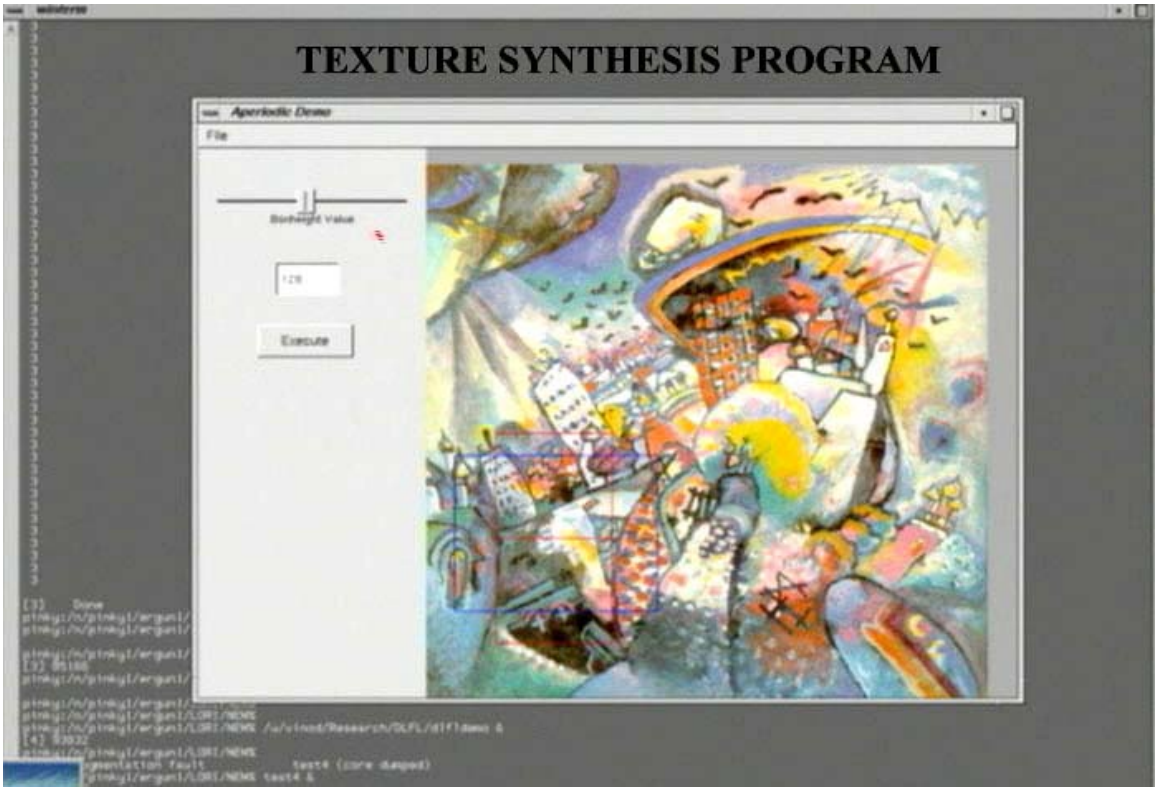


Fig. 32. The interface of the texture synthesis system. The image used for the procedural *tiled texture* creation is a reproduction of a Kandinsky painting. Interface created by [7].

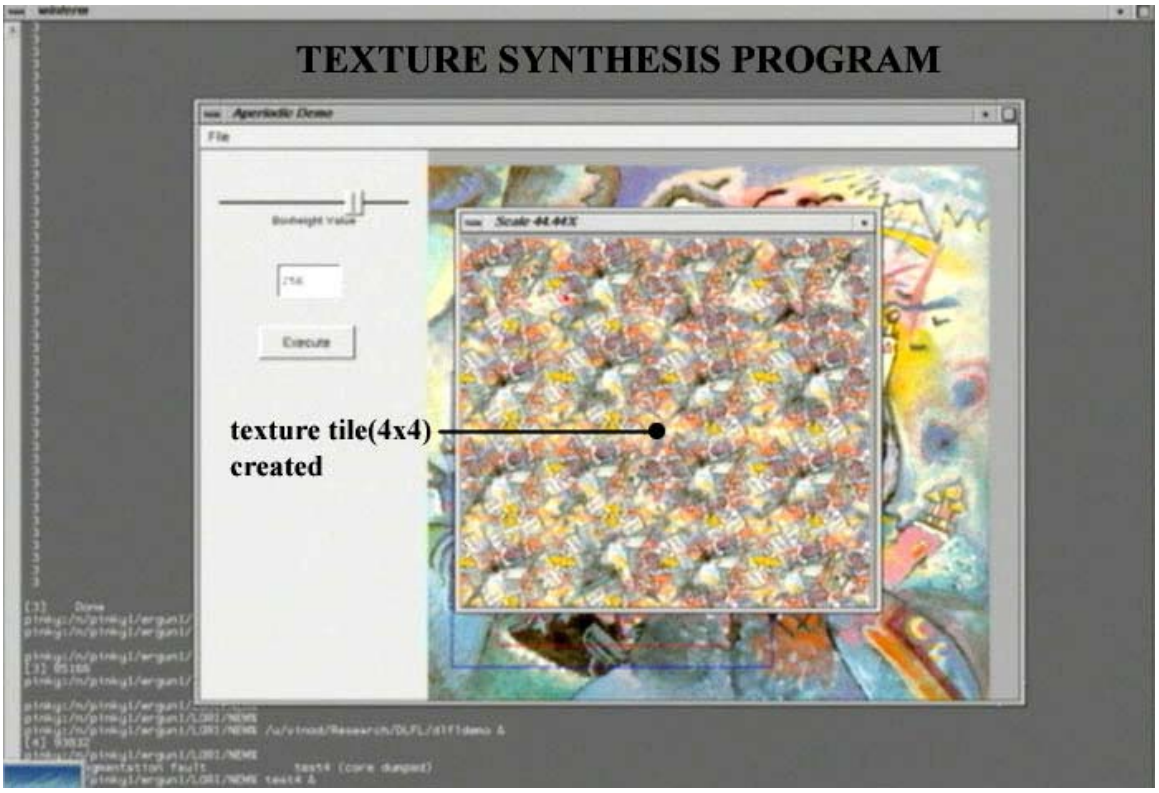


Fig. 33. A “4 x 4” *tiled texture* created from a Kandinsky painting using the interface of the tiled texture image synthesis system.

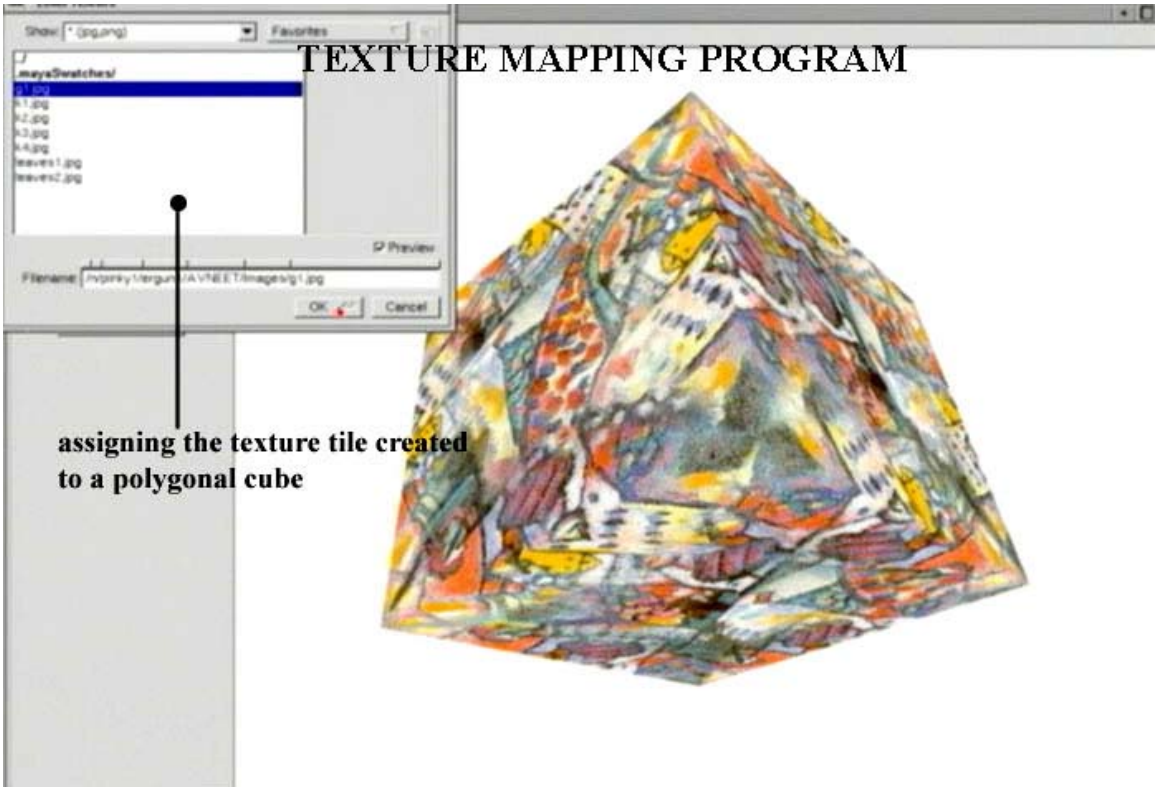


Fig. 34. The Interface of the Texture Mapping System shows how a *tiled texture* (saved as an image file) is accessed to be assigned to a polygonal object. In the above case the object is imported from Maya into the DLFL mesh modeling system.

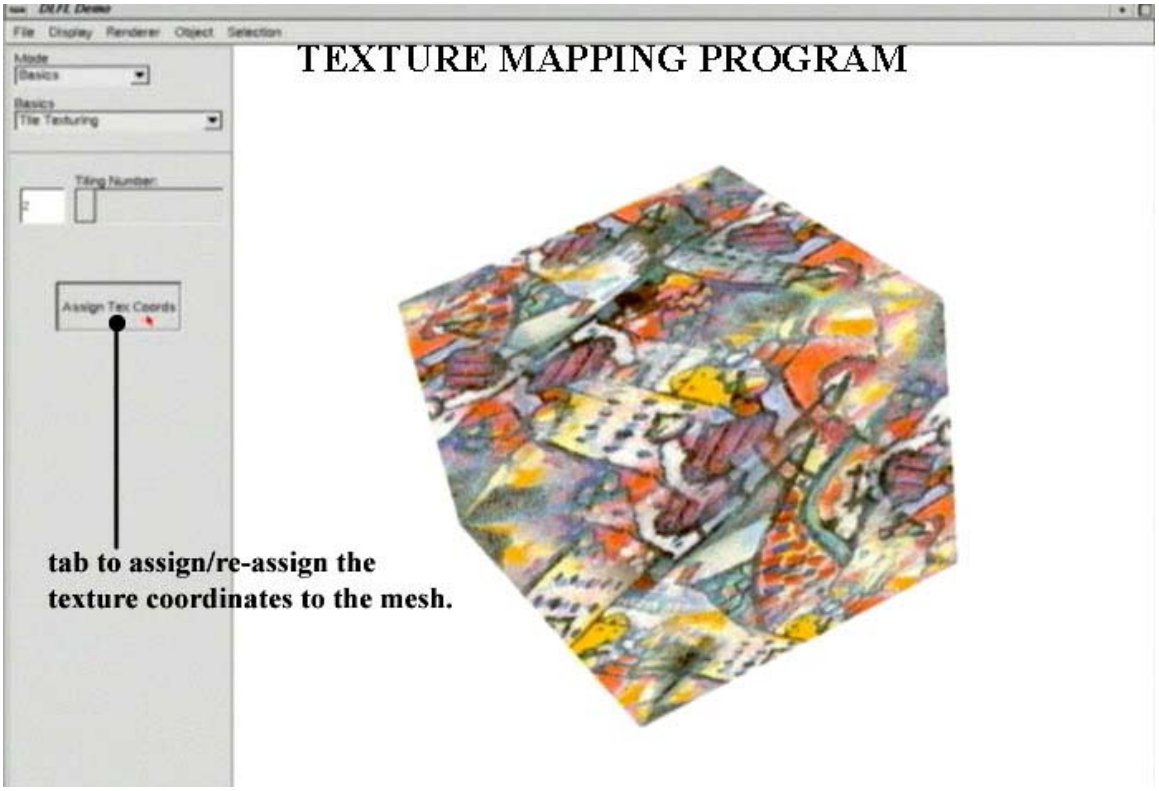


Fig. 35. On re-assigning the texture coordinates you see that the texture tile gets mapped in a different manner to the cube than that shown in Figures 34. This happens in a random manner every time we hit the “assign coordinates” tab.

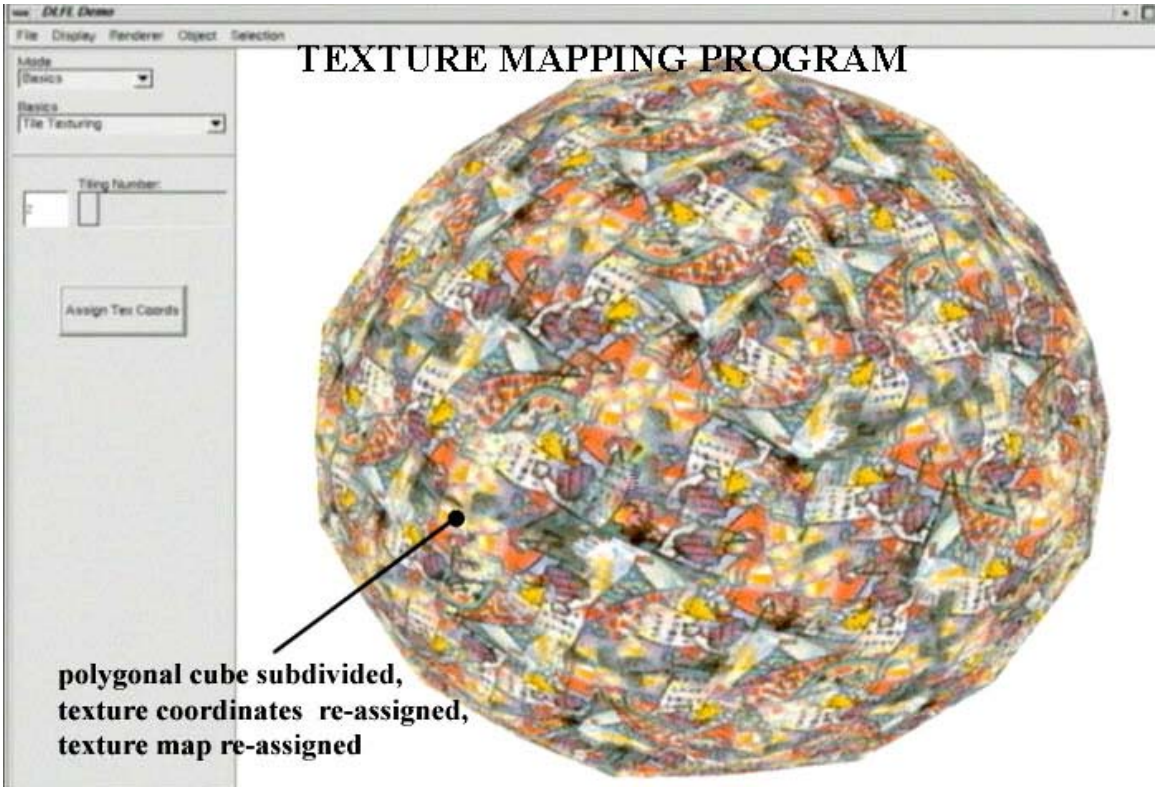


Fig. 36. The polygonal cube in Figure 35 is subdivided using “Catmull Clark Subdivision Scheme” and the texture coordinates re-assigned, thus the texture tiles gets mapped differently.

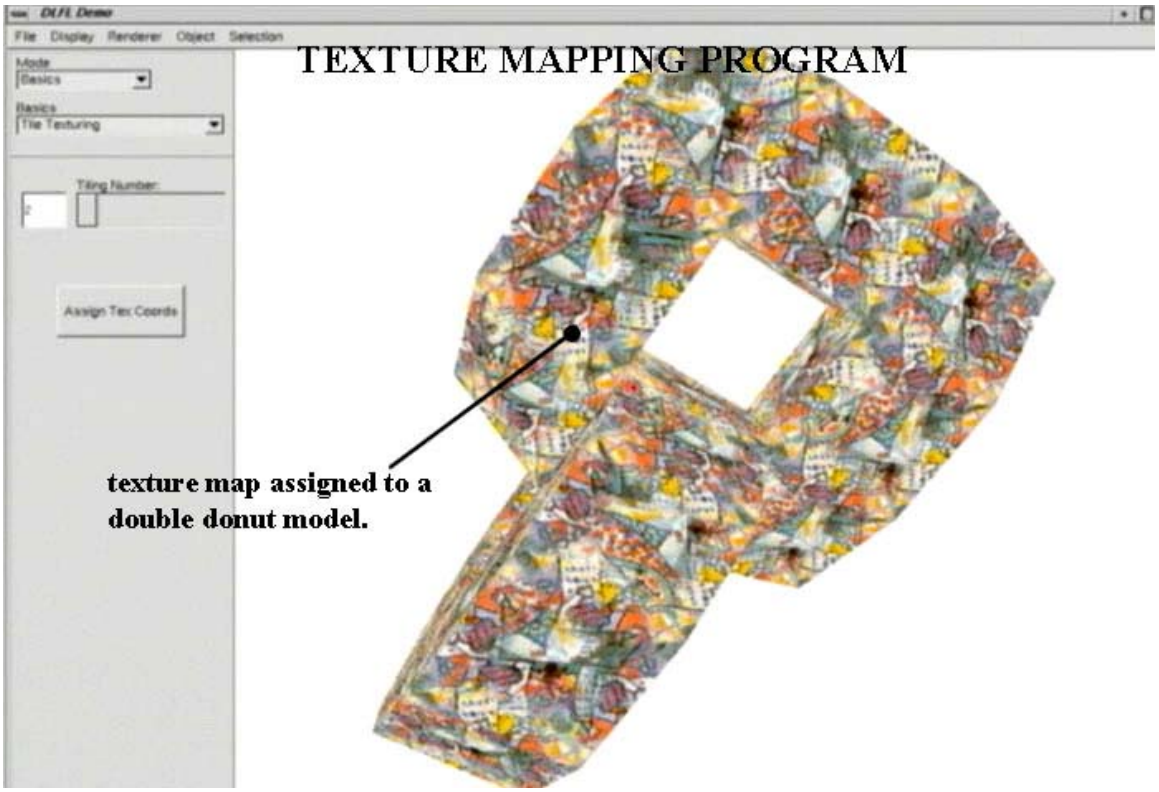


Fig. 37. Texture coordinates and texture map (created in Figure 33) assigned to a polygonal model of a double donut (genus-2 surface that includes valence 3, 4, 5 and 6 vertices).

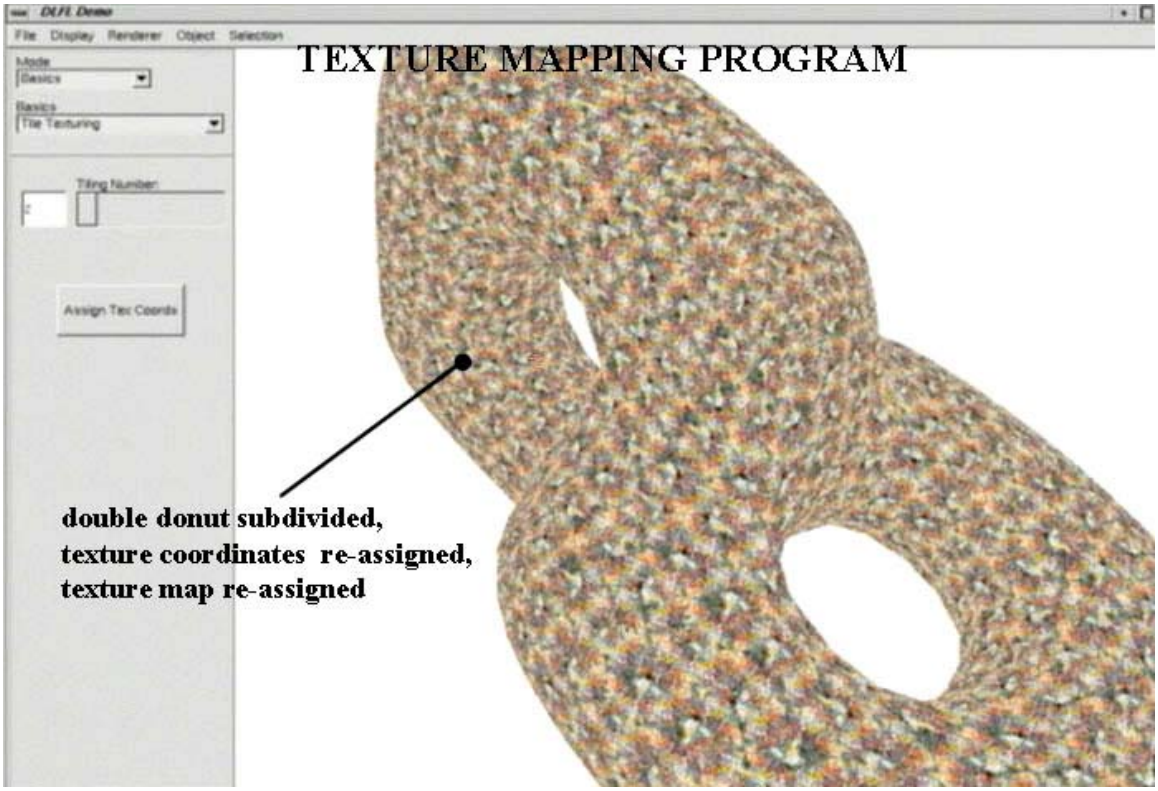


Fig. 38. The double donut in Figure 37 is subdivided using “Catmull Clark Subdivision Scheme” and the texture coordinates re-assigned, thus the texture tile gets mapped differently.

CHAPTER V

RESULTS

As a proof of concept, we have created all final images and animations in Maya. The usability of the system was tested in a graduate level computer graphics course. 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. Following are the most significant results that have been achieved, by using the texture mapping technique presented in this thesis work:

1. Seamless and Continuous boundaries across textured meshes.

2. Singularity free texturing.

3. Aperiodic looking texturing.

Tiled textures created by us, which were used as input texture maps for the mapping program presented in this work, can be seen in Figures 39 and Figures 40. *Tiled textures* as seen in Figures 39(A), (B) and Figures 40(G) are hand drawn, while *tiled textures* in Figures 39(C), (D) and Figures 40(E), (F), and (H) have been procedurally created using the texture synthesis technique by [7]. Rendered examples created by mapping these *tiled textures* and the texture mapping method presented in this work, can be seen in Figures 41, Figures 41, Figures 42, Figures 43, Figures 44, Figures 45, Figures 46, Figures 47, and Figures 48 respectively.

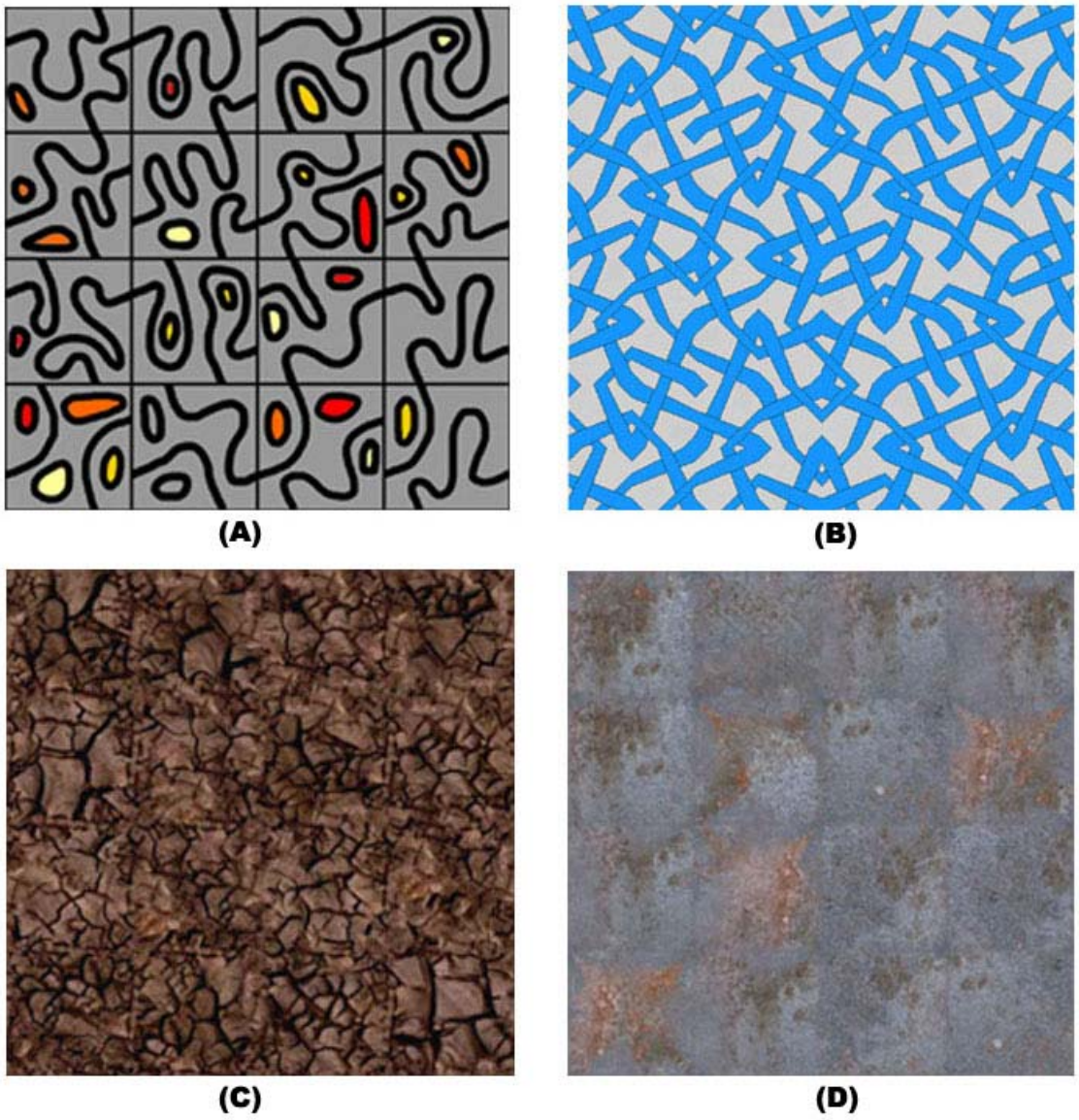


Fig. 39. Examples of *tiled textures* created for being used as input texture maps for the mapping program presented in this work. (A), (B), (C) and (D) are example of *tiled textures* used in Figures 41, Figures 42, Figures 43 and Figures 44 respectively.

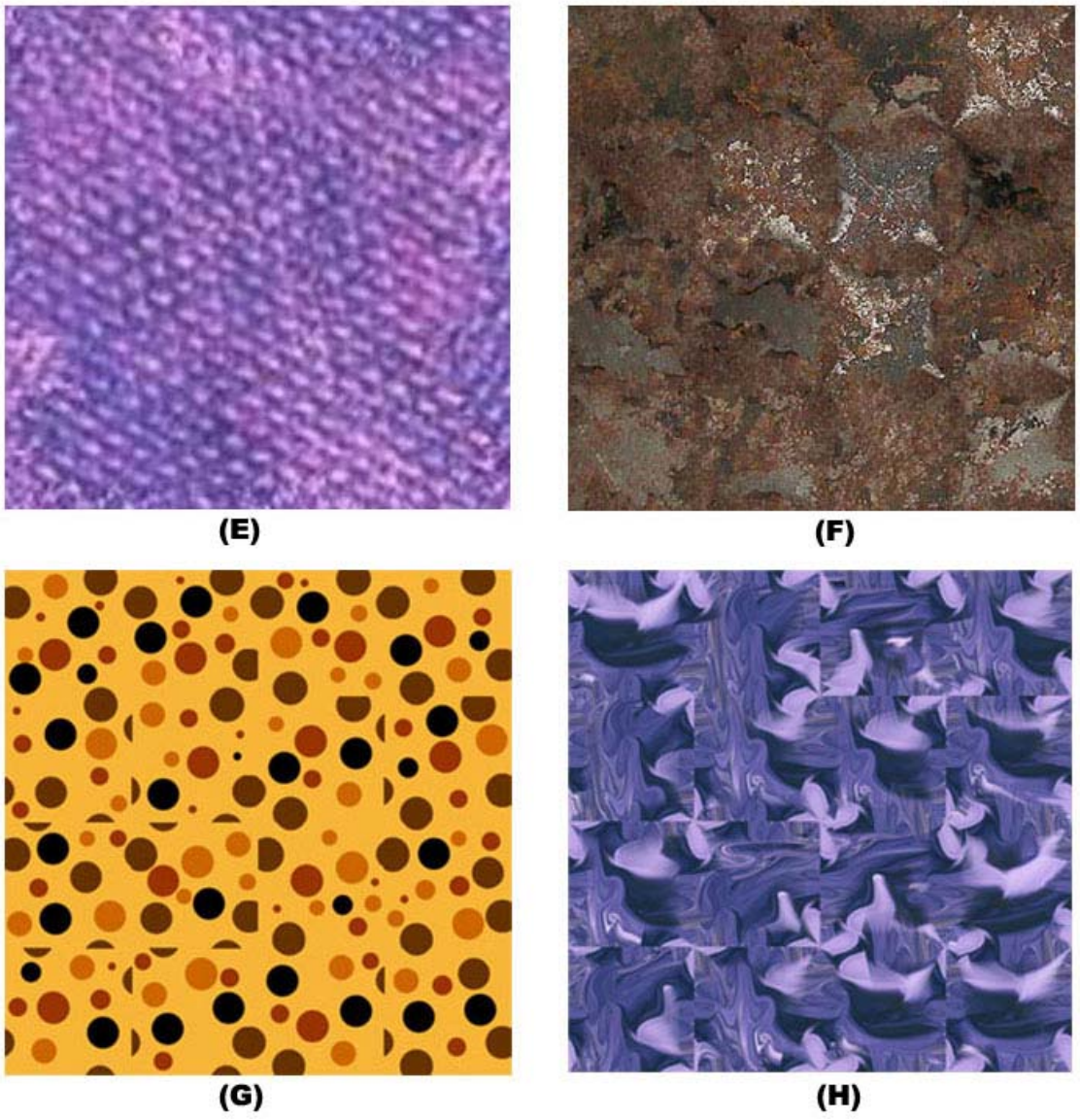


Fig. 40. More examples of tiled textures created for being used as input texture maps for the mapping program presented in this work. (E), (F), (G) and (H) are example of tiled textures used in Figures 45, Figures 46, Figures 47 and Figures 48 respectively.

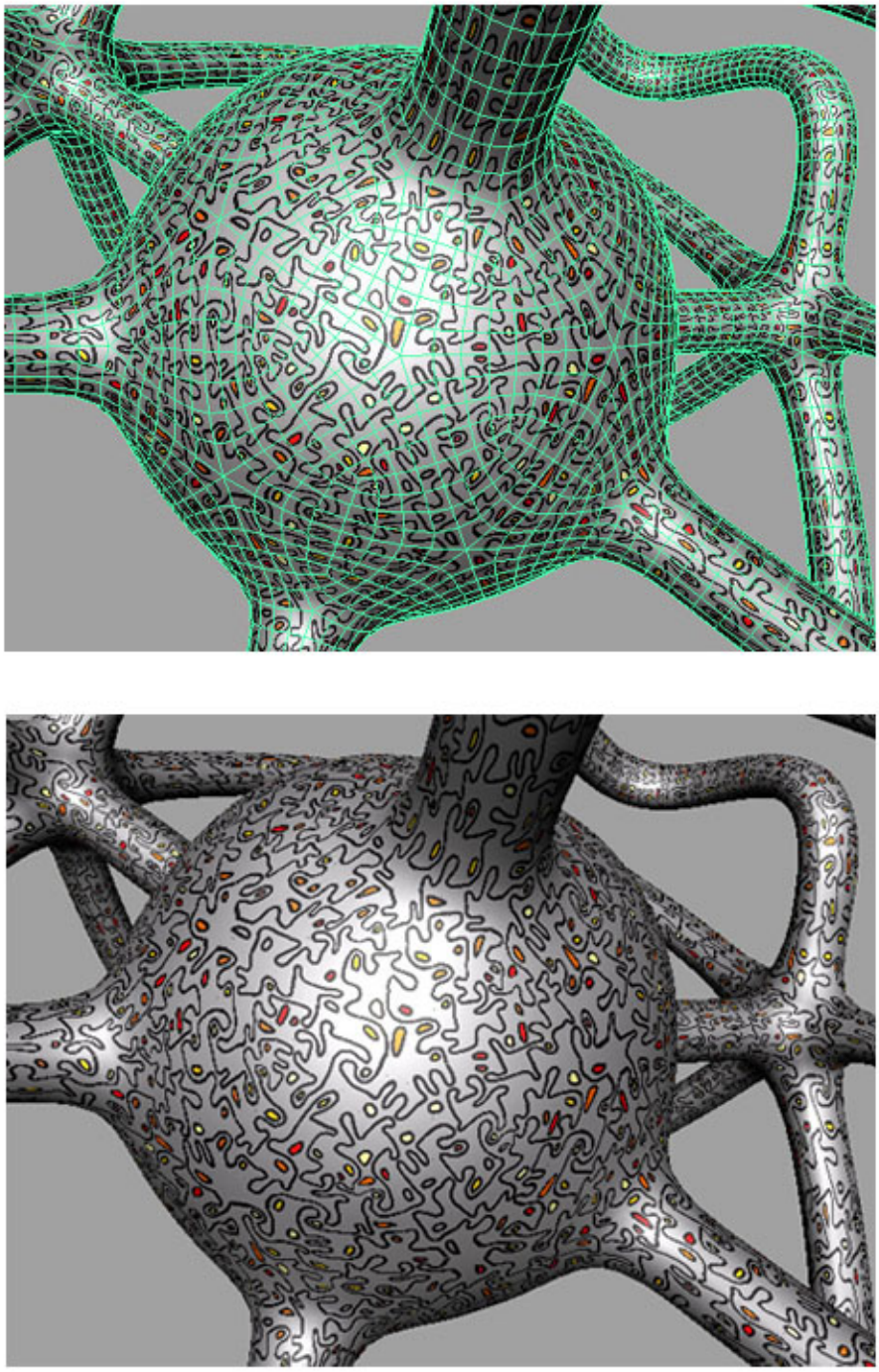


Fig. 41. An arbitrary polygonal mesh (having valence 3, 4, and 5 vertices) mapped with a hand drawn tiled texture seen in Figures 39(A).

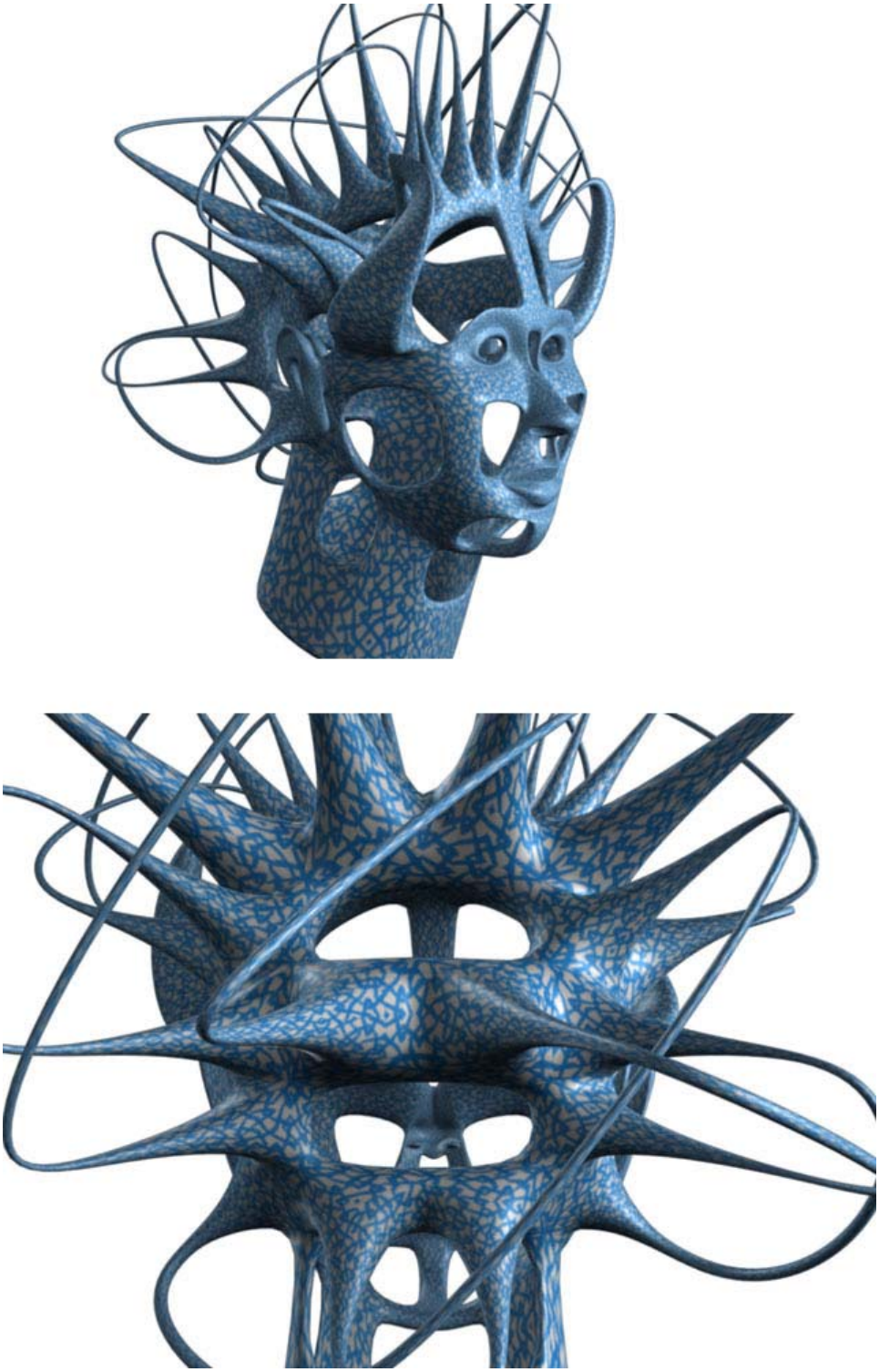


Fig. 42. Polygonal model of a face (having valence 3, 4, 5 and 6 vertices) mapped with a tiled texture inspired by “Escher’s Tile Patterns” as seen in Figures 39(B).

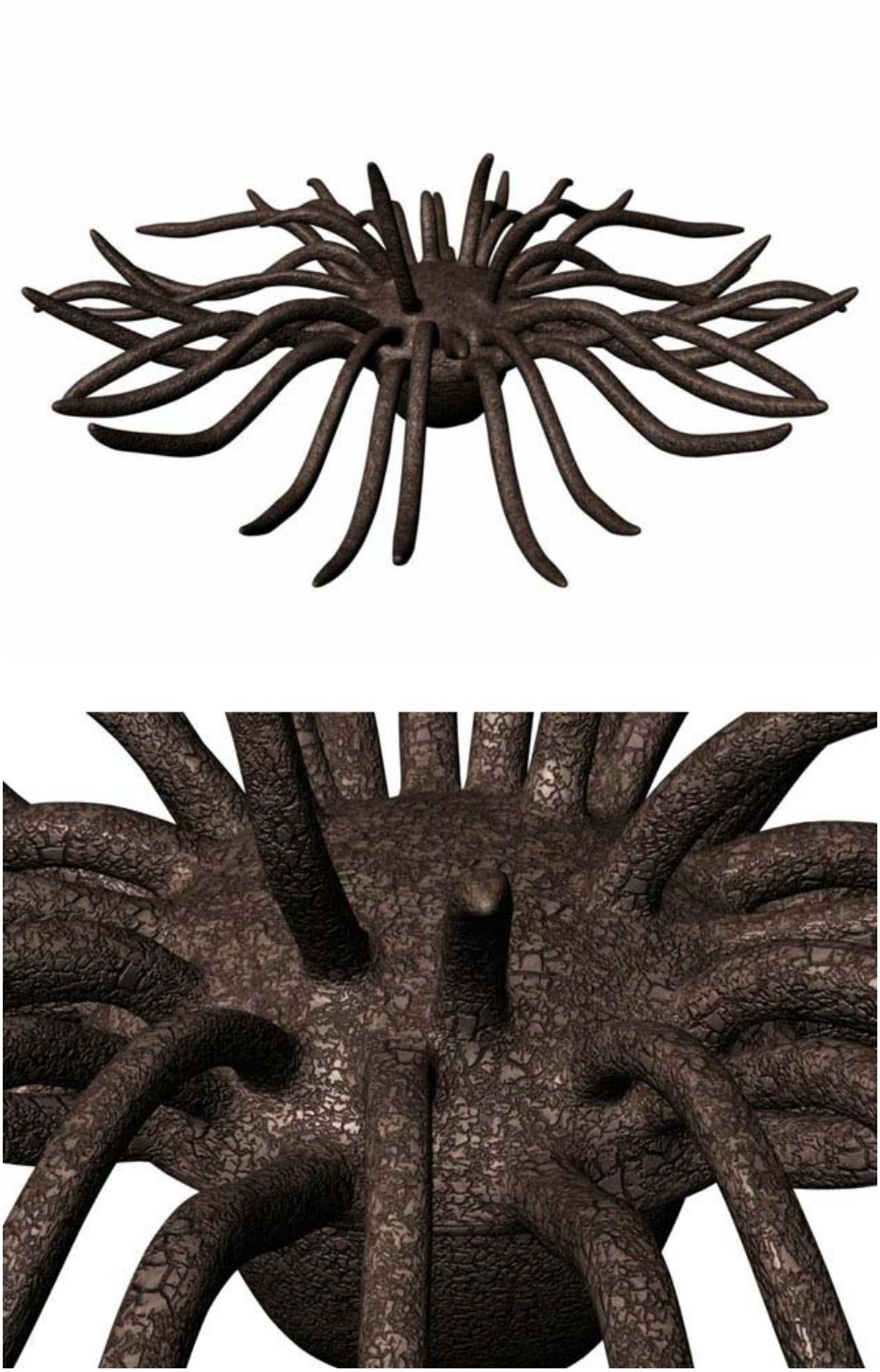


Fig. 43. An arbitrary polygonal mesh (genus-1 surface that includes valence 3, 4, 5 and 6 vertices) mapped with a tiled texture inspired by cracked mud pattern seen in Figures 39(C).

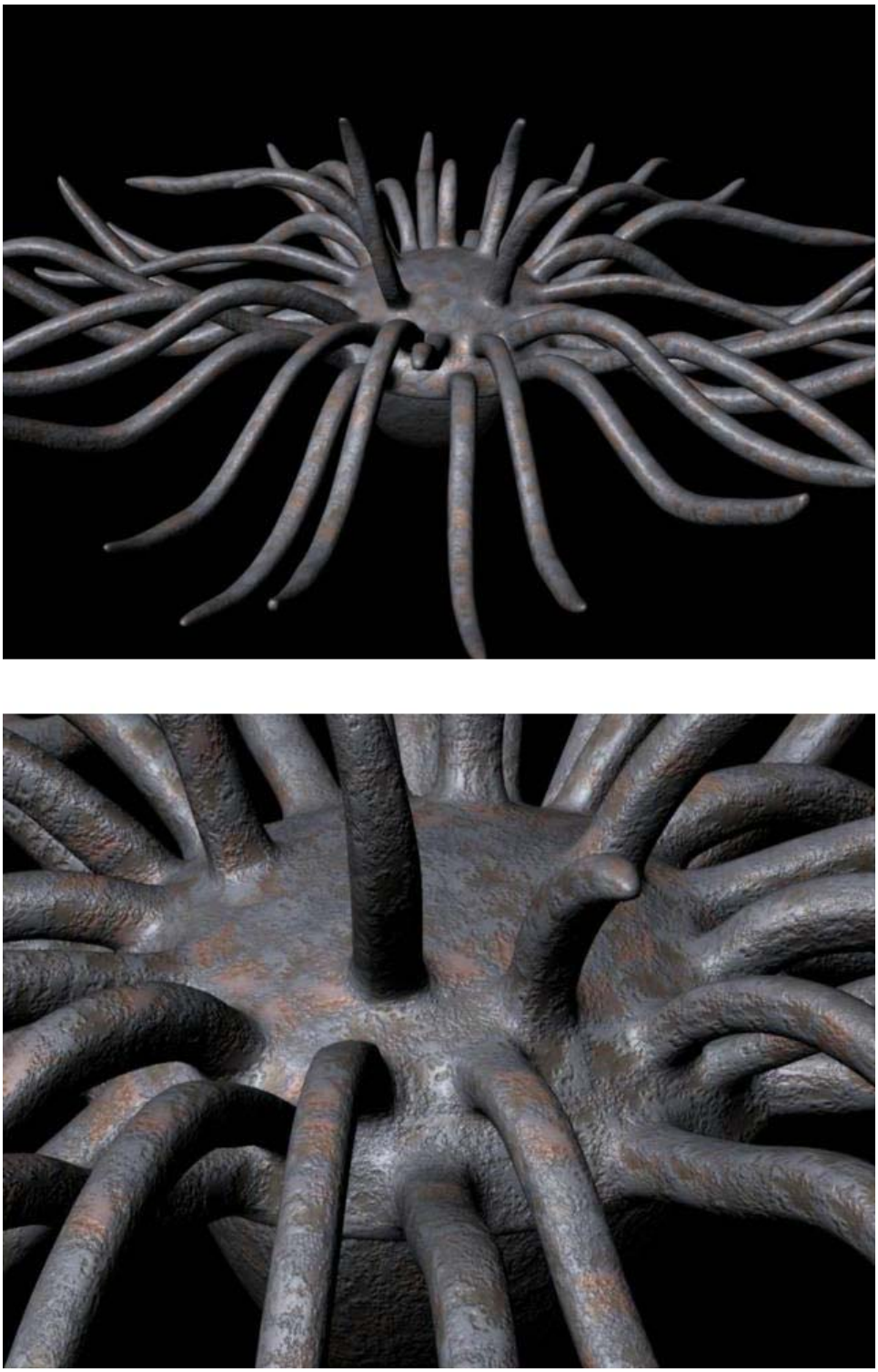


Fig. 44. Polygonal model of a mesh (genus-2 surface that includes valence 3, 4, 5 and 6 vertices) mapped with a tiled texture inspired by a rusted metal pattern as seen in Figures 39(D).

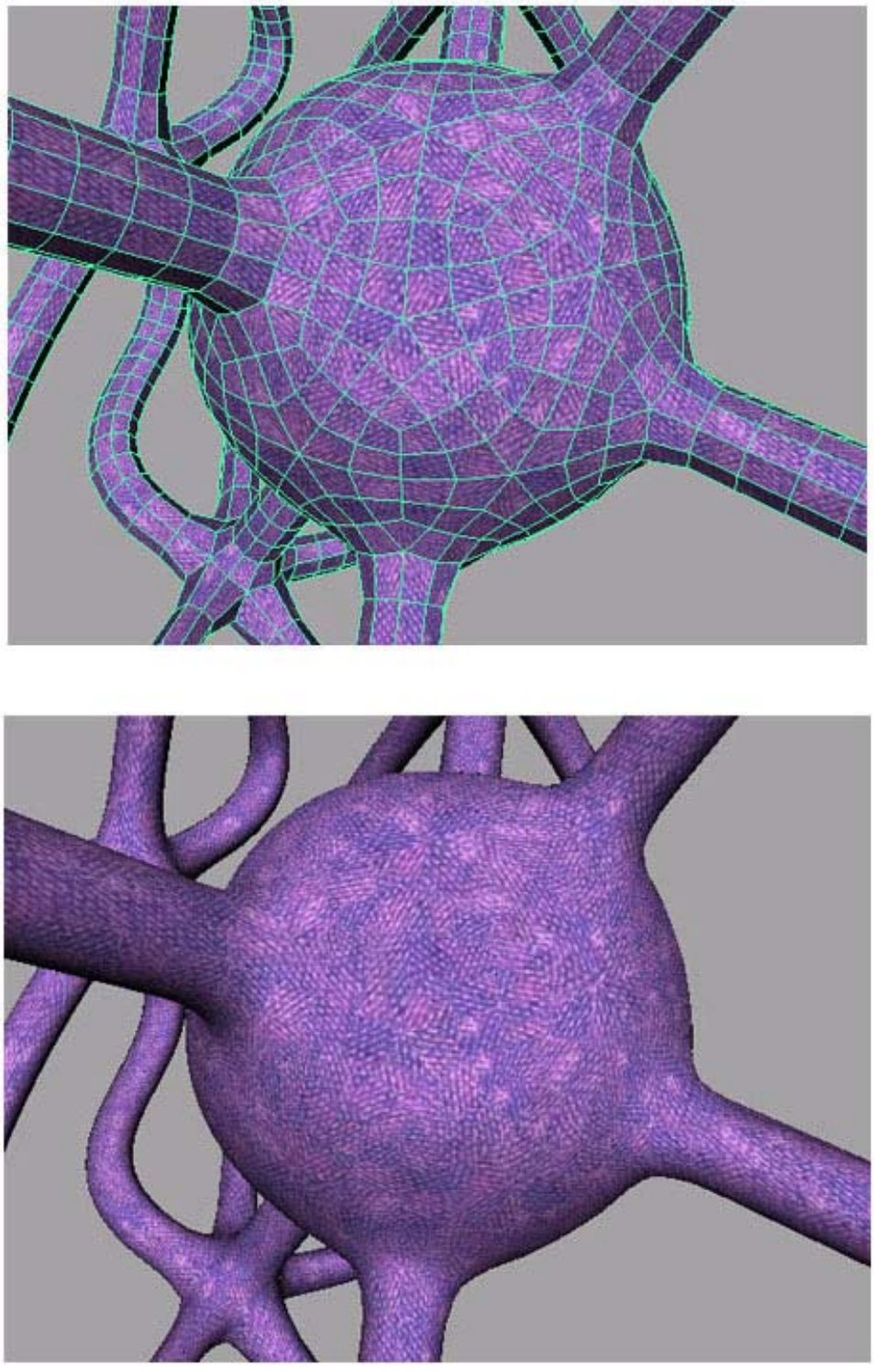


Fig. 45. An arbitrary polygonal mesh (genus-2 surface that includes valence 3, 4, and 5 vertices) mapped with a tiled texture seen in Figures 40(E).

Fig. 46. Polygonal model of a face (having valence 3, 4, 5 and 6 vertices) mapped with a tiled texture inspired by a rust pattern as seen in Figures 40(F).

Fig. 47. An arbitrary polygonal mesh (genus-1 surface that includes valence 3, 4, 5 and 6 vertices) mapped with a tiled texture inspired by spotted skin pattern used as a texture map and a bump map, as seen in Figures 40(G).

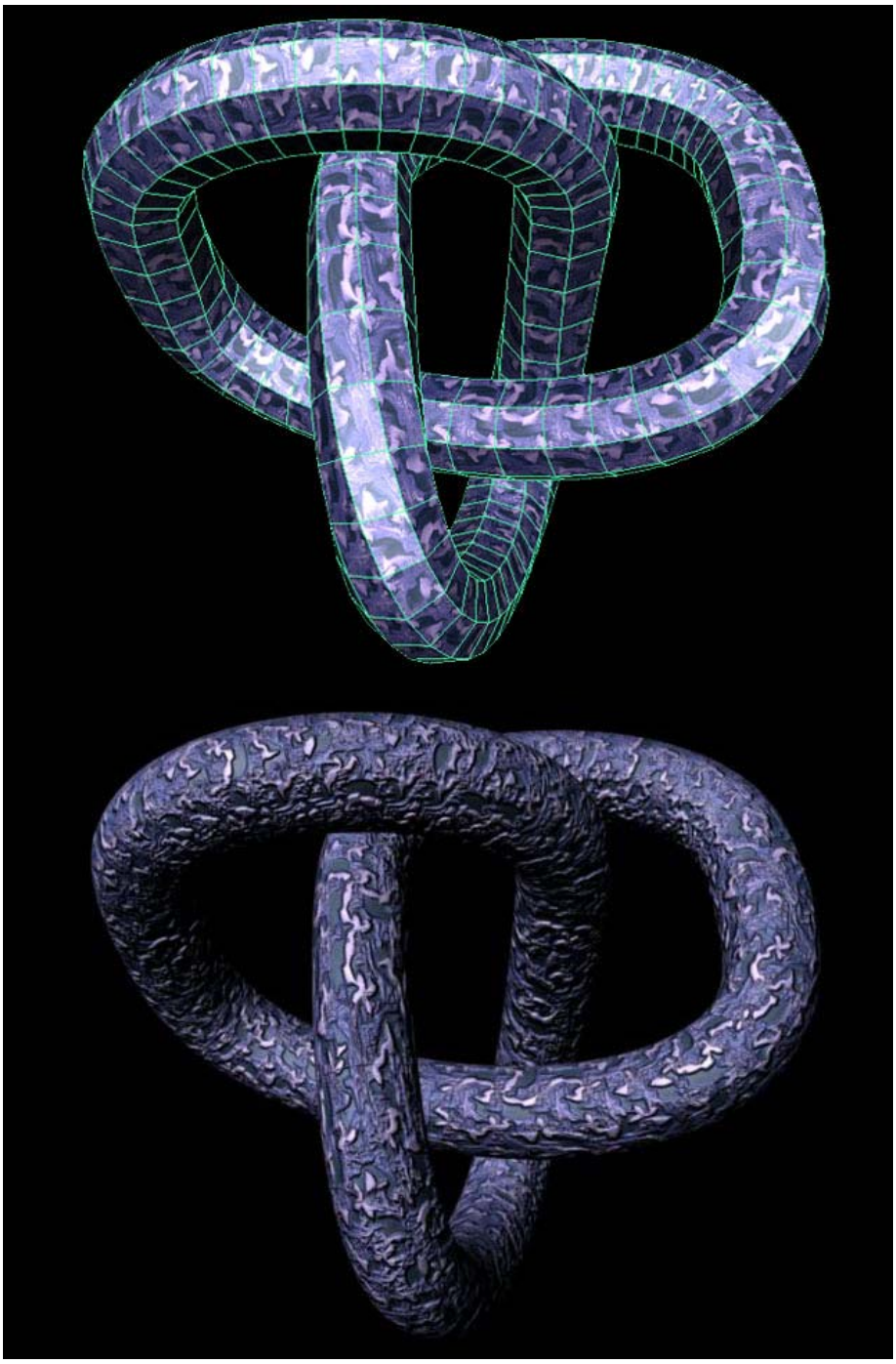


Fig. 48. An arbitrary polygonal mesh, mapped with a tiled texture, used as a texture map and a bump map, as seen in Figures 40(H).

CHAPTER VI

CONCLUSION AND FUTURE WORK

VI.1. Conclusion

In this thesis work we presented a simple and practical technique for seamlessly texturing quadrilateral meshes. Quadrilateral meshes are very common in computer graphics and any mesh can easily be converted to a quadrilateral mesh using a vertex insertion scheme such as Catmull-Clark subdivision [5]. Hence the texture mapping method we present can be used for mapping any arbitrary mesh.

The texture mapping method we present in this work, provides singularity free texturing for surfaces, whatever the topology of the underlying polyhedral mesh. It provides continuous and seamless boundaries all across the textured surface. It also yields an aperiodic look, such that the texture pattern appears more interesting and non-repeating all over the surface of the object. The user has more freedom in choosing what texture patterns he may want to use, without having to bother about the problems of singularity, discontinuity, seams across boundaries, non-interesting and repetitive looking patterns, over any topology and geometry whatsoever. This thesis work is unique in the fact that it requires the user to read in just one texture file. This is what makes it extremely convenient for the user to use the method presented in this work, unlike the Neyret-Cani method. This texture mapping method is hence compatible with currently existing commercial packages (like Maya, Softimage, etc), which allows texturing of polygonal meshes. The examples shown in this work have two distinct boundaries each, but this concept can easily be generalized to n distinct boundaries by using a four-digit $2n$-ary numbers. A boundary k is be represented by two boundary types that are be denoted by $2k$ and $2k+1$ where $k = 0, 1, \ldots, n-1$ and a tiled texture image consists of $(2n)^4$ tiles. The boundary types(e.g. “0” and “1”, in

case of n =2) assigned to the individual texture tiles are used as their names and respective positions in the matrix formation of the *tiled texture*. The first two digits give the column number and the second two digits give the row number. The texture mapping algorithm can be thus easily modified to support n distinct boundaries by adding only one additional line to **AssignBoundaries** procedure used in the algorithm that we have developed. Neyret and Cani work also mention self-continuous boundaries. The method presented by us currently supports them too, as we maintain C1 continuity across boundaries of texture tiles mapped onto a mesh.

VI.1.1. Limitations of the Texture Mapping Method

The texture mapping method, as presented in this thesis work, does not provide for anisotropic textures(textures which have distinct oriented patterns, e.g.: tiger or zebra stripes). However, by changing the texture mapping algorithm and using more than one boundaries, it is possible to allow for mapping anisotropic textures.

At the moment the mapping algorith we have presented , allows successful texture mapping with texture tiles having lower values of n, where n is the number of different boundary conditions, or oriented edges in texture tile. $2n^4$ gives the size of the texture tile we create.For example in case of a texture tile having 1 oriented edge, as in all the example cases presented in this work, the size of the texture tile becomes 2^4 = 16 . Since $2n^4$ quickly becomes a very large number, the size of the texture file can become a problem when n becomes large. For instance, current texture map files are generally not larger than 4 megapixels. If we have 8 boundaries with 16 boundary types, each tile cannot be larger than 8×8 pixels.

The mapping approach presented in this work does not guarantee to minimize the amount of stretching/distortion of the texture. However, application of the Catmull-Clark

scheme reduces distortion significantly. Moreover, an existing surface mesh can also be re-meshed to obtain better quality or smoother looking quadrilateral mesh.

At the moment this work and examples we have shown, supports only distinct boundaries conditions.

VI.2. Future Work

There are many directions in which possible future work may be carried on with respect to work presented in this thesis.

One possibility may be to include the introduction of user-controlled pattern anisotropy, which is an important feature of many natural textures. Using anisotropic patterns directly in the scheme we have presented would not be a good idea, as it does not give results which appear visually correct.

We are also currently working on the development of new organization schemes that can use the texture space more efficiently, in order to handle more than two different boundary conditions. This involves cases when ”n”(n is the number of different boundary conditions), becomes a larger number and hence the texture tile becomes very big to create and store. (Example in case of 3 different boundary conditions, the texture tile would become a 16 x 16 matrix).

Another possible extension of this work could be the development of “Multi resolution *tiled texture* images” such that we will be able to handle, cases in the mesh topology, where some quadrilateral faces in the mesh may be big, and some faces may be smaller. In other words the quadrilaterals patches in the arbitrary mesh may vary greatly in size from one another. We will then be able to choose the resolution of the texture tile(from a multiresoltion *tiled texture*) required to fit appropriately onto a specific size of a patch in arbitrary mesh. Doing so, will allow to achieve a uniform and non-distorted texture

mapping even on a meshes which have varying sizes of quadrilateral patches.

It may prove useful to give more control to the users, to create the *tiled textures*. The future extensions can be to develop a painting approach for creating the tiled textures.

Another extension of this work, could involve the texture mapping of non-quadrilateral meshes, or meshes which may be made up of triangles, hexagons etc. For this purpose there might be a need to create a new method for both designing the texture maps and creating a texture mapping method accordingly.

REFERENCES

[1] Ergun Akleman, Jianer Chen. Guaranteeing 2-manifold property for meshes by using doubly linked face list. *International Journal of Shape Modeling*, volume 5, no. 2, pages 149-177, August 2000.

[2] Eric A. Brier and Kenneth R. Sloan. Jr. Two-part texture mapping. *IEEE Computer Graphics and Applications*, volume 6, no. 9, pages 40-53, September 1986.

[3] Jules Bloomenthal. Modeling the mighty maple. In *Computer Graphics, volume 19, no. 3 (SIGGRAPH '85)*, pages 305-311, July 1985.

[4] Edwin E. Catmull. A subdivision algorithm for computer display of curved surfaces. Ph.D. Dissertation, Department of Computer Science, University of Utah, December 1974.

[5] Edwin E. Catmull and James Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, no. 10, pages 350-355, September 1978.

[6] H. S. M. Coxeter, M. Emmer, R. Penrose, and M. L. Teuber, editors. *M. C. Escher: Art and Science*. North Holland, Amsterdam, 1986.

[7] Lori Green. Tiled Texture Synthesis. unpublished manuscript, Dept. of Architecture, Texas A&M University, August 2003.

[8] Bruno Levy and Jean-Laurent Mallet. Non-distorted texture mapping for sheared triangulated meshes. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, pages 343-352. ACM SIGGRAPH, Addison Wesley, July 1998.

[9] Jerome Malliot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH 93 Proceedings)*, volume 27, pages 27-34, August 1993.

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

[11] Fabrice Neyret and Marie-Paule Ciani. Pattern based texturing revisited. In Robert Cook, editor, *SIGGRAPH 99 Conference Proceedings*, pages 235-242. ACM SIGGRAPH, Addison Wesley, August 1995.

[12] Darwyn R. Peachey. Solid Texturing of Complex Surfaces. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 279-286, July 1985.

[13] Hans Kohling Pedersen. Decorating implicit surfaces. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 291-300. ACM SIGGRAPH, Addison Wesley, August 1995.

[14] Hans Kohling Pedersen. A framework for interactive texturing operations on curved surfaces. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 295-302. ACM SIGGRAPH, Addison Wesley, August 1996.

[15] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH 85 Proceedings)*, volume 19, pages 287-296, July 1985.

[16] Emil Praun, Adam Finkelstein and Hugues Hoppe. Lapped textures. In Kurt Akeley, editor, *Computer Graphics (SIGGRAPH 00 Proceedings)*, pages 465-470, ACM SIGGRAPH, Addison Wesley, July 2000.

[17] Samek, Marcel, Cheryl Slean and Hank Weghorst. Texture mapping and distortion in digital graphics, *The Visual Computer, volume 2, no. 5*, pages 313-320, September 1986.

[18] Cyril Soler, Marie-Paule Cani and Alexis Angelidis. Hierarchical Pattern Mapping, In Robert Cook, editor, *Computer Graphics (SIGGRAPH 02 Proceedings)*, pages 673-680, ACM Press, New York, USA, July 2002.

[19] Jos Stam. Aperiodic texture mapping. Technical Report R046, European Research Consortium for Informatics and Mathematics (ERCIM), January 1997. http://www.ercim.org/publication/technical reports/046-abstract.html.

[20] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH 91 Proceedings)*, volume 25, pages 289-298, July 1991.

[21] Greg Turk. Texture synthesis on surfaces. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH 01 Proceedings)*, pages 347-354, August 2001

[22] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In Thomas W. Sederberg, editor, *Computer Graphics (Siggraph 91 Proceedings)*, volume 25, pages 299-308, July 91.

[23] Steven P. Worley. A cellular texturing basis function. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 291-294. ACM SIGGRAPH, Addison Wesley, August 1996.

VITA

Avneet Kaur
A-216, Visualization Laboratory
Langford Architecture Center, TAMU
College Station-77843
avneet@viz.tamu.edu

Education

M.S. in Visualization Sciences	Texas A&M University, 05/03
B.Architecture	Birla Institute of Technology, 5/99

Research Interests

Texture Mapping
Lighting and Shading Techniques
3D-Modeling

Employment

Texas A&M University, College Station, TX Graduate Assistant, 9/00 - 12/02

Honors

Scholarship from Texas A&M University, 2000-2001