Pattern Formation And Mapping For Animal Skin Synthesis

Teo How Jiann

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Master of Engineering

2006

Acknowledgements

First of all, I would like to thank my project supervisor, Dr. Wong Kok Cheong, for giving me a chance to start this research project. This is an interesting project, which is challenging and is having many rooms for advancement. With his guidance, I can clearly see the research direction of this project. His high expectation on research quality gives me much pressure. However, this is necessary in order to complete the project within the three years tight schedule.

Next, I would like to thanks Prof. Marcelo Walter, from Universidade do Vale do Rio dos Sinos, Brazil. He is the original creator of the renowned *Clonal Mosaic* model. His kindness in sharing his source code of the *Clonal Mosaic* model is highly appreciated. This gives me a very good starting point in understanding the detail of his model.

I would like to thanks the research staffs in our center as well. They are Low Boon Hean, Lim Swee Kim and Golam Ashraf. They give their hands by solving some of the technical problems that I had met, while using the Alias Maya software as a development tools. Besides, Goh Chun Keong, Goh Chun Hoong and Lai Swee Kim, who are the computer graphics artists, gave me various feedbacks related to the practical problems of the developed system.

Contents

Acknowledgement I

Contents III

List of Figures V

List of Tables VI

Abstract VII

1 Introduction 1
- 1.1 Motivation and Objectives . . . 1
- 1.2 Major Contribution of the Thesis . . . 2
- 1.3 Organization of the Thesis . . . 4

2 Literature Review 6
- 2.1 Pattern Formation in Biology . . . 6
- 2.2 Texture Creation and Pattern Formation in Computer Graphics . . . 10
  - 2.2.1 Image-based Texturing . . . 10
  - 2.2.2 Procedural Texturing . . . 12
- 2.3 Summary of Literature Review . . . 14

3 Clonal Mosaic Model 15
- 3.1 Basic Concepts . . . 15
- 3.2 Details of Clonal Mosaic Simulation Process . . . 16
  - 3.2.1 Representation . . . 17
  - 3.2.2 Initialization . . . 18
  - 3.2.3 Simulation . . . 19
  - 3.2.4 Voronoi Diagram Computation . . . 24
- 3.3 Discussion and Summary . . . 25

4 Canonical Reference Plane Structure 28
- 4.1 Neighbor Mapping Function in Three-Dimension . . . 29
- 4.2 Creation of Canonical Reference Plane Structure . . . 32
- 4.3 Neighbor Mapping Function on Reference Plane . . . 34
- 4.4 Discussion . . . 36

5 Enhancement on Clonal Mosaic Simulation 37
5.1 Pattern Simulation . . . 38
5.1.1 Cell Division and Relaxation . . . 38
5.1.2 Global Relaxation . . . 39
5.1.3 First Approach of Local Relaxation . . . 40
5.1.4 Second Approach of Local Relaxation . . . 42
5.2 Cell-in-Triangle Test . . . 44
5.3 Discussion . . . 47

6 Rendering Texture Pattern 49
6.1 Render Clonal Mosaic Pattern as Voronoi Diagram . . . 50
6.2 Implementation of Closest-Cell-Searching . . . 52
6.3 Discussion . . . 54

7 Experimental Results 57
7.1 Comparison of Classical and Optimized Clonal Mosaic Models . . . 58
7.1.1 Performance with Different Number of Cells . . . 59
7.1.2 Performance with Different Number of Triangles . . . 61
7.1.3 Performance with Different Repulsive Weights . . . 64
7.1.4 Performance with Different Number of Simulation Steps . . . 66
7.2 Performance of Texture Rendering . . . 68
7.2.1 Performance of Rendering with Different Resolutions . . . 68
7.2.2 Performance of Rendering with Different Number of Cells . . . 70
7.3 Comparison with Real Animal Skin Patterns . . . 71
7.4 Rendering on Arbitrary Polygonal Objects . . . 75
7.5 Summary of Experimental Results . . . 82

8 Conclusion 84
8.1 Contributions . . . 85
8.2 Recommendation for Further Research . . . 86
8.2.1 Synthesis of Highly Complex Skin Patterns . . . 86
8.2.2 Parameter Estimation for Clonal Mosaic Model . . . 88

Bibliography i

Appendix - Publications iv

List of Figures

2.1 Sea shell pigmentation generated by using Activator-Inhibitor model [11] . 7
2.2 Animal coat pattern simulated with Reaction-Diffusion model [6] 7
2.3 Fish skin pattern generated with Reaction-Diffusion system [14] 8
2.4 Pattern formation on geometry with different shapes [10] 8
2.5 Effects of pattern formation on body with different sizes 9
2.6 Various texture synthesis approaches 11
2.7 Simulate Reaction-Diffusion texture on arbitrary surfaces [8] 13
2.8 Texture simulation from Clonal Mosaic model [37] 13

3.1 Overall workflow of a Clonal Mosaic model 17
3.2 Initialization of Clonal Mosaic simulation 19
3.3 Calculation of repulsive forces among cells 21
3.4 Mapping cell from one triangle to another triangle 22
3.5 Neighbor Mapping of triangles 23
3.6 Voronoi diagram on polyhedron 24
3.7 Workflow of the system with the proposed solution methods 26

4.1 Rotate a triangle about a sharing edge 29
4.2 Result of mapping neighbors by rotation about intersection line 30
4.3 Cell-floating problem 32
4.4 Obtaining a Reference Triangle 33
4.5 Canonical Reference Plane Structure 33
4.6 Neighbor Mapping for Reference Triangles 34
4.7 Transformation between master and neighboring triangle 34

5.1 Pseudo codes for pattern simulation 38
5.2 An event queue 39
5.3 Pseudo codes for the Global Relaxation process 40
5.4 Pseudo codes for the first approach of the Local Relaxation process 41
5.5 Computation of cell repulsion for Local Relaxation 42
5.6 Pseudo codes for the second approach of the Local Relaxation process . . . 43
5.7 Overlapped neighborhoods in the Local Relaxation process 44
5.8 Various point-in-polygon tests 45
5.9 Pseudo codes for Cell-in-Triangle test 46
5.10 Line cross polygon test 47
5.11 Line cross polygon test for neighbor triangle 47

6.1 Rendering of Clonal Mosaic texture pattern 51
6.2 Pseudo codes for closest-cell-searching 53
6.3 Closest-cell-searching from neighboring triangles 54
6.4 Voronoi diagram showing cells' regions 55
6.5 Voronoi error introduced from distance approximation 55

7.1 Performance of Clonal Mosaic simulation obtained using different simulation models with different number of cells 59
7.2 Texture patterns generated using different simulation models with different number of cells 60
7.3 Performance of Clonal Mosaic simulation obtained using different simulation models with different number of triangles 61
7.4 Results of the Neighbor Mapping with different number of triangles 62
7.5 Texture patterns generated using different simulation models with different number of triangles 63
7.6 Performance of Clonal Mosaic simulation obtained using different simulation models with different repulsive weights 64
7.7 Texture patterns generated using different simulation models with different repulsive weights 65
7.8 Performance of Clonal Mosaic simulation obtained using different simulation models with different simulation steps 66
7.9 Texture patterns generated using different simulation models with different simulation steps 67
7.10 Rendering result obtained using various texturing methods 68
7.11 Rendering performance of various texturing methods obtained with different render resolutions 69
7.12 Rendering performance with different number of cells 70
7.13 Cheetah 73
7.14 King Cheetah 73
7.15 Leopard 73
7.16 Jaguar 74
7.17 Giraffe 1 74
7.18 Giraffe 2 74
7.19 Stanford Bunny model with 3560 triangles and simulated with 15000 cells . 76
7.20 Cow model with 5795 triangles and simulated with 20000 cells 77
7.21 Dophin model with 4198 triangles and simulated with 20000 cells 78
7.22 Horse model with 9924 triangles and simulated with 30000 cells 79
7.23 Triceratops model with 5660 triangles and simulated with 20000 cells . . . 80
7.24 Triceratops with 11328 triangles and simulated with 30000 cells 81

8.1 Highly complex skin patterns 87

List of Tables

3.1 Attributes of a cell 19

4.1 Operation counts for transforming single and multiple points [44] 31

7.1 Key parameter settings for the experiment with different number of cells . . 59
7.2 Clonal Mosaic simulation time(s) obtained using different simulation models with different number of cells 59
7.3 Key parameter settings for the experiment with different number of triangles 61
7.4 Clonal Mosaic simulation time(s) obtained using different simulation models with different number of triangles 61
7.5 Key parameter settings for the experiment with different repulsive weights . 64
7.6 Clonal Mosaic simulation time(s) obtained using different simulation models with different repulsive weights 64
7.7 Key parameter settings for the experiment with different simulation steps . 66
7.8 Clonal Mosaic simulation time(s) obtained using different simulation models with different simulation steps 66
7.9 Rendering time(s) of various texturing methods with different render resolutions 69
7.10 Rendering time(s) of Clonal Mosaic model with different number of cells . 70
7.11 Description of key parameters 71

Abstract

In this project, we aim to develop an effective system for synthesizing skin patterns of animals on three-dimensional surfaces.

To accomplish the task of pattern synthesis, we have proposed and developed a system [1], which is inspired by *Clonal Mosaic* model [2]. A *Clonal Mosaic* model simulates cell-to-cell repulsion on arbitrary surface. By assigning different repulsiveness values on these cells, a regulated spatial cell arrangement is likely to be formed. This arrangement of cells shows appealing result, which is comparable with those natural patterns found on animal skin. However, a typical *Clonal Mosaic* process requires a complex computation, where distance of cells across the surface has to be measured and movement of cells has to be constrainted on the surface. As a result, a time consuming process is envisaged for generating patterns on an arbitrary surface.

In order to resolve these major problems and issues encountered in the standard *Clonal Mosaic* model, the following solution methods have been devised and implemented:

- A *Canonical Reference Plane Structure* is proposed and implemented to reduce the computational cost incurred in the evaluation of the distance between cells. In this structure, an efficient neighboring triangle mapping and triangle transformation functions are derived.
- The concept of *Local Relaxation* is introduced to avoid redundant computations for cells repulsion. The spatial arrangement of cells of the result obtained using the proposed *Local Relaxation* method is identical to those obtained in the standard *Clonal Mosaic* model. However, our method leads to a faster simulation time.
- A rendering process, which works directly on the simulation result, is proposed. A search function is used to locate a relevant cell with which a pixel to be rendered is associated. The color of the pixel is then determined by the property of the corresponding cell.

Extensive experimental results have been presented to verify the quality of the skin patterns and computational performance of the proposed techniques.

Chapter 1

Introduction

This chapter gives an overview of our research works. Firstly, the motivation and objectives of the proposed project are described. Next, the major contributions of our research works are summarized. Lastly, the organization of the thesis is presented.

1.1 Motivation and Objectives

Animals acquire their skin pattern for various reasons. In a tropical forest, a spotted leopard will integrate better with the surrounding, giving it a higher chance to sneak up on its prey. On the other hand, a stripped zebra hidden in vertical striations of long bushes is hardly visible to their predators. Other than keeping itself from a predator and making itself closer to a prey, patterns do help animals to recognize other members of their species.

The aim of this research project is to synthesize the fascinating color patterns of animals in the application domain of computer graphics. Several biological models for the pattern formation on animal skin were studied [3][4][5][6][7]. It is found that an empirical biological model for the animal skin pattern formation is still under active research. The existing models have successfully predicted the pigmentation processes of the seashells and the color pattern formation on several species of fishes.

Several computational approaches have also been proposed for the texture synthesis [2] [8]. The works on simulation of simple patterns (e.g. spots of cheetah, stripes of zebra and patches of giraffe) on mammalian coats achieved certain success. Despite this achievement, it is very difficult to obtain acceptable computational performance and/or realistic results by using the existing methods for synthesizing animal skin with complex and intricate patterns.

Representative complex patterns are various type of rosette patterns studied from jaguar, ocelot and clouded leopard.

The high demands in realistic and appealing appearance of computer generated imagery (CGI) are the current trend in the computer animation industry. However, in order to produce a highly complex image, it often requires an extensive simulation and rendering process. As for the texture pattern synthesis, the current approaches are still considerably far from the high expectations from the entertainment industry. Practically, this is generally true in high quality production as the models are relatively complex (e.g. large number of polygons) and their rendered images are expected to be intricately complex and subtle (i.e. huge numbers of texture pattern elements in great variation)

The problem of pattern formation for animal skin synthesis is carefully studied. A system, which can efficiently synthesize a desired skin pattern, is developed. In order to realize the synthesis of realistic complex patterns of animal skins, we will have to accomplish the following tasks:

1. To devise an effective theoretical and computational model for synthesizing relatively complex skin pattern using concepts gleaned from biological pattern formation mechanisms.

2. To develop a technique for applying the synthesis model on arbitrary three-dimensional surfaces.

1.2 Major Contribution of the Thesis

A procedural texturing method named as *Clonal Mosaic* model, which is proposed by Walter [9], is taken as a basis for this project. His model works with a large number of points (cells), which are distributed over a piece of arbitrary geometry. The movement and division of cells are controlled by some parameters, so that by grouping cells with similar properties and keeping away cells with different properties, a regulated spatial arrangement of cells can be formed. The resulted spatial arrangement of cells, once associates with a Voronoi diagram, represents a tessellation of cell region, which are visually identical with color patches on animal skin. A *Clonal Mosaic* model is able to create spots on cheetah, patches of giraffe and rosettes on leopard. It has the potential to create more fascinating results of complicated rosette patterns on some big cats, by giving more controls on differentiating

cells belong to individual pattern elements. However, a typical *Clonal Mosaic* simulation process requires a manipulation of large number of parameters. Besides this drawback, a long simulation time will also introduce practical difficulty in the exploration of generating more complicated and highly varied pattern formations.

The *Clonal Mosaic* model is rigorously studied and the major limitations and drawbacks of this model are presented. To remedy some of these problems, the following techniques are proposed and developed:

- A *Canonical Reference Plane Structure* is proposed. It converts a given piece of three-dimensional surface into a two-dimensional representation. This approach effectively reduces the computational cost incurred in the point's transformation and evaluation of the distance between points.
- The concept of *Local Relaxation* is introduced to avoid redundant computations for cell repulsion. In the worst case scenario, the computational performance of our method is still compatible to the classical model.
- A rendering process, which works directly on the simulation result, is proposed. A searching function is used to locate a relevant cell with which a pixel to be rendered is associated. The color of the pixel is then determined by the property of the corresponding cell. It is worth to mention that the generation of Voronoi diagram is not required for our proposed model.

The contributions mentioned above dominate the major modules in the pattern formation system of this project. In our framework, the numbers of input parameters and the quantity of input data to be processed are highly optimized. The computational costs in the simulation and rendering process are also efficiently reduced. Hence, the proposed system is able to handle complicated geometry and synthesize high quality texture patterns.

1.3 Organization of the Thesis

The organization and structure of the remaining chapters of the thesis are summarized as follows:

- **Literature Review** (Chapter 2):
  This chapter provides a literature review on pattern formation in biological development and texturing problems in the domain of computer graphics.

- **Clonal Mosaic Model** (Chapter 3):
  The concepts and implementation of *Clonal Mosaic* model are described in detail. Problems and issues arisen from the *Clonal Mosaic* model are identified and discussed.

- **Canonical Reference Plane Structure** (Chapter 4):
  The integration of preprocessing of an arbitrary surface into a *Canonical Reference Plane Structure* is presented . The computation of surface distances and transformation of faces are simplified with this approach.

- **Enhancement on Clonal Mosaic Simulation** (Chapter 5):
  A *Local Relaxation* process, which greatly reduces the computation cost for a *Clonal Mosaic* simulation is proposed. With this enhancement, a considerable amount of the computation processing of cells repulsion involved in the standard *Clonal Mosaic* model can be avoided. Beside this merit, a further improved cell-in-triangle test is also devised for easing the manipulation of cells on surface constraint.

- **Rendering Texture Pattern** (Chapter 6):
  In this chapter, we propose a rendering algorithm that directly works with the simulation result, which are a set of cells with spatial coordinate. A closest-cell-searching method is performed, in order to find the closest cell from a render point. This strategy avoids the complex computation of a *Polyhedral Voronoi Diagram*, while still gives a correct tessellation of Voronoi polygon at the end of rendering.

- **Experimental Results** (Chapter 7):
  This chapter provides a detailed analysis of experimental results obtained from the proposed techniques. The comparison of the computational performance of the simulation based on the standard and our modified model is presented. Next, the speed

of our rendering algorithm is studied. We also present several patterns that are achievable from our system and compare them with real world animal skin pattern. Lastly, the patterns are synthesized directly on several three-dimensional models.

- **Conclusion** (Chapter 8):

  A conclusion on the research works reported in this thesis is drawn. Our major contributions and merits in pattern texture synthesis are highlighted. Lastly, the limitations of our current approaches are addressed. To alleviate some of these problems, potential further research works are proposed and discussed.

Chapter 2

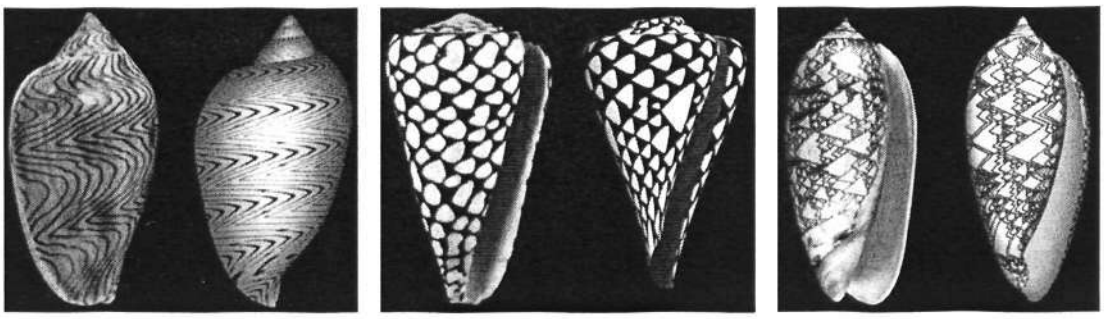
Literature Review

In this chapter, the pattern formation models in the biological field are first reviewed. Next, representative existing methods for synthesizing real world patterns in the domain of computer graphics are discussed.

2.1 Pattern Formation in Biology

Ball [10] conducted a comprehensive survey on animal skin pattern formation to date. One of the earliest discussions on pattern formation in biological discipline was initialized by Alan Turing in 1952 [3]. He proposed a *Reaction-Diffusion* system, which explains how the anatomical structure of an organism is developed from its zygote. The remarkable element about Turing's idea was that the diffusion of chemical compounds through a medium, which they were dispersed, could be the driving force for symmetry breaking. This goes against intuition, as normal diffusion is seemed as a mechanism to produce uniformity, which smoothes out inhomogeneities in a system.

In Turing's chemical system, diffusion is acting in competition with another process, namely an autocatalytic chemical reaction. A chemical compound A undergoes an autocatalytic reaction to produce more of itself. The production rate of A depends on the amount of A already present. The existence of another compound B acts as an inhibitor, which inhibits the production of more A. The key element for obtaining a spatial pattern is that, A and B diffuse through the reaction medium at different rates. This ensures that the effective ranges of their respective influences are different. It also means that A and B can dominate in distinct regions.

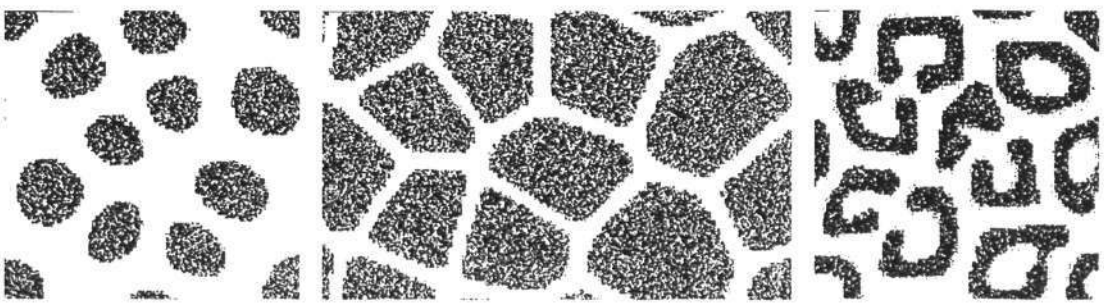


(a) Amoria undulata (b) Oliva porphyria (c) Conus marmoreus

Figure 2.1: Sea shell pigmentation generated by using Activator-Inhibitor model [11]

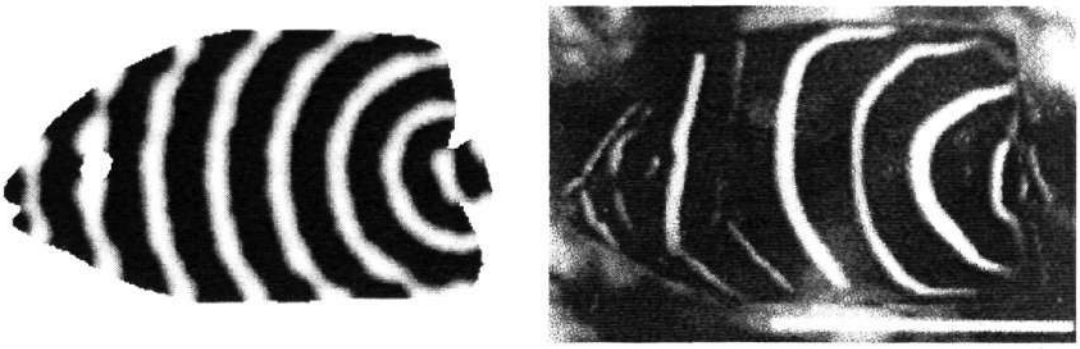
It is not until 1972, Gierer and Meinhardt [4] described an *Activator-Inhibitor* and *Activator-Substrate* model, which are derived from the *Reaction-Diffusion* system. In an *Activator-Inhibitor* model, a simple molecular interaction with pattern-forming capability would consist of an activator, whose autocatalysis is slowed down by a long ranging inhibitor. On the other hand, an *Activator-Substrate* model describes a possible interaction between an activator and a substrate that is consumed during the autocatalysis. These methods are proven to be true in generating remarkable seashell pigmentation [5] [11] (See Figure 2.1). Shell grows bigger by continual accretion of calcified material onto the outer edge. As a result, the pattern observed from the surface of a shell is a trace of the pigment distribution along a one-dimensional line at the shell's edge. The *Activator-Inhibitor* and *Activator-Substrate* models have correctly predicted the pigment concentration during the growth of various species of mollusc shells [12].

Despite the seashell pigmentation, some efforts in describing mammalian coat pattern with the *Reaction-Diffusion* model also persist. Kochy and Meinhardt [6] used the previously described *Activator-Inhibitor* and *Activator-Substrate* models to simulate several types of mammalian coat pattern like cheetah, giraffe and leopard (See Figure 2.2). However, the simulated patterns did not give compatible results as compared with the real world



(a) Cheetah (b) Giraffe (c) Leopard

Figure 2.2: Animal coat pattern simulated with Reaction-Diffusion model [6]



(a) Stripe pattern simulated with Reaction-Diffusion system

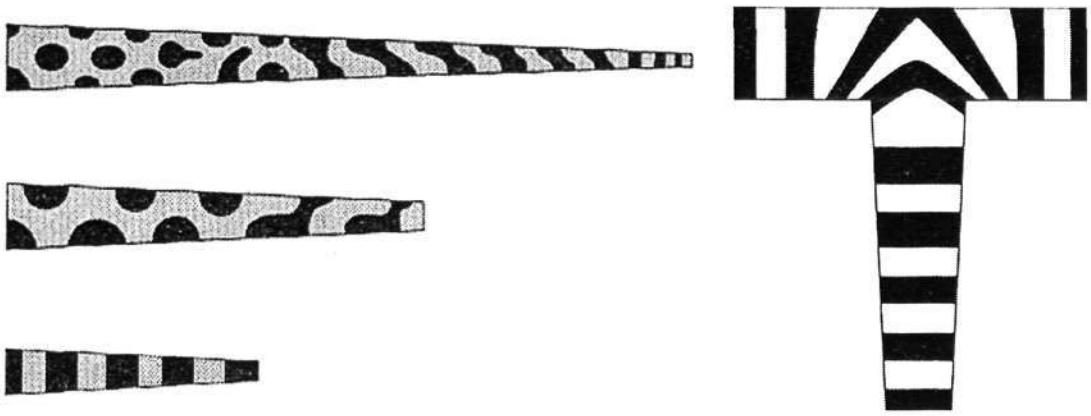
(b) Real fish

Figure 2.3: Fish skin pattern generated with Reaction-Diffusion system [14]

patterns.

Various attempts in generating skin pattern of fishes with *Reaction-Diffusion* model are presented by Painter [13] [14] and Shoji [7]. Painter simulated a *Reaction-Diffusion* system on an irregular domain determined by tracing the boundary of a real fish (See Figure 2.3). Shoji suggested that the pattern selection for spots and stripes is determined by an equilibrium level of the activator. Besides, he also suggested that an anisotropic diffusion is responsible for the direction of stripes on a body of the fishes.

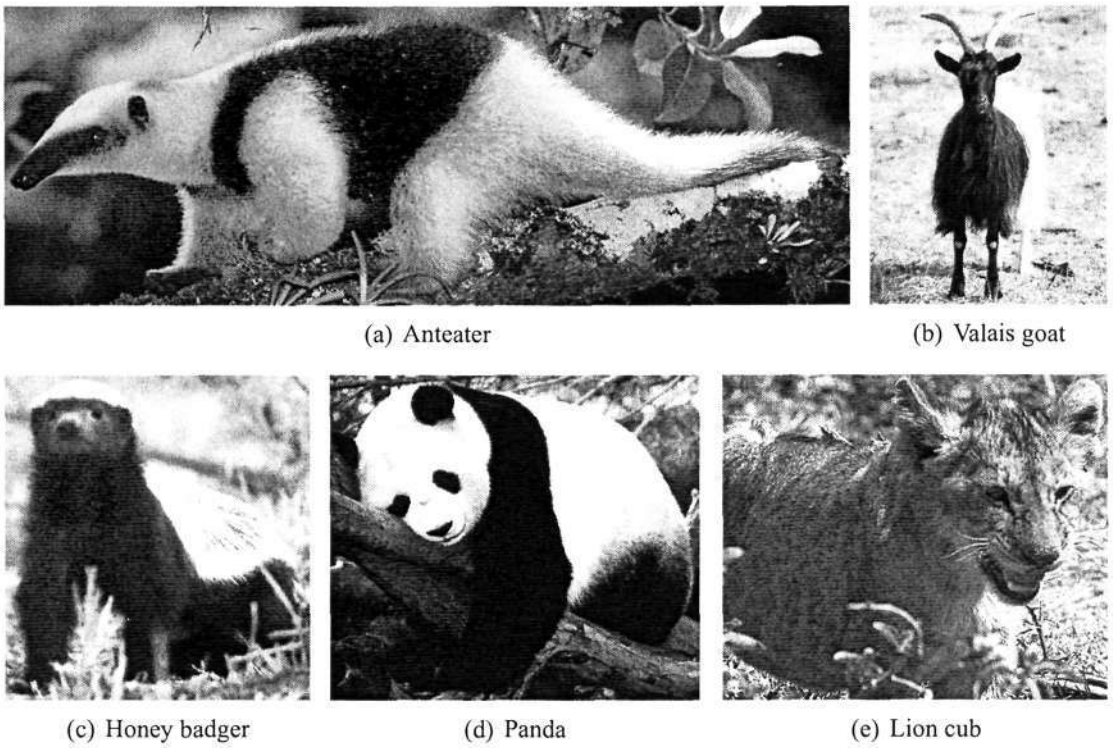
Murray [15] [16] pointed out some important features of pattern formation on mammalian coat. He suggested that the size and shape of the region, where a chemical process is occurring, also influences the pattern formation process. A *Reaction-Diffusion* system is generated on tapered cylinders with different sizes (See Figure 2.4(a)). If the model tail is small, only bands are formed. If the tail is larger, spot patterns can be formed. For a more slowly tapering tail, the transition from bands to spots is obvious. Besides, the stripes



(a) Pattern formation on tail

(b) Chevron pattern of zebra

Figure 2.4: Pattern formation on geometry with different shapes [10]



(a) Anteater (b) Valais goat

(c) Honey badger (d) Panda (e) Lion cub

Figure 2.5: Effects of pattern formation on body with different sizes

pattern of zebra, which forms a chevron pattern at the junction of the leg and body, is also correctly predicted in the geometry shown in Figure 2.4(b)

Murray also suggested that the formation of pattern on mammalian coat starts in the embryonic stage. A genet's embryo has thin tail, which is uniform in diameter, so bands are formed. Whereas an embryonic leopard tail is short and sharply tapered, so allows spots. Another observation from different species of zebras shows that stripe pattern with different widths is possible, if the timings to acquire the pattern are different.

Another important issue raised by Murray is that the scale of body size has dramatic effect in pattern formation. Small animals with short gestation periods have less complex skin patterns than larger animals, as their smaller embryos support fewer modes. Some good examples are shown in Figures 2.5(a) to 2.5(d), where an abrupt division of body into white and black half is observed.

On the other hand, it was found that the apparence complexity of a pattern diminishes when the size of an animal becomes large. This is because as more and more modes become possible on the patterned embryo, the features start to merge as the dividing lines between them become squeezed out. This is obvious in the pattern of giraffe. Besides, spot patterns occur on cubs of lions (See Figure 2.5(e)) disappear when they grow up to adults. It is

believed that these patterns merge together and become invisible as the size of a body becomes larger.

2.2 Texture Creation and Pattern Formation in Computer Graphics

In computer graphic, textures are added onto a piece of geometry in order to increase its realistic appearance. Quote a word from Turk [8]:

Texture *is any change in surface appearance across an object that can be effectively separated from the gross geometry of that object.*

Texture synthesis methods are proposed to increase the texture appearance and provide texture variation on a surface. Two major methods are widely applied for texturing a piece of geometry in computer graphics, namely image-based texturing and procedural texturing. In an image-based texturing, two-dimensional images are needed as an input. A texture mapping process is required to create correspondences between two-dimensional texture coordinates and three-dimensional surface coordinates. For a procedural texturing, mathematical models are used to generate texture directly on a surface. In the subsequent sections, some of the existing recent works on solving some of the problems encountered from these two approaches are discussed.

2.2.1 Image-based Texturing

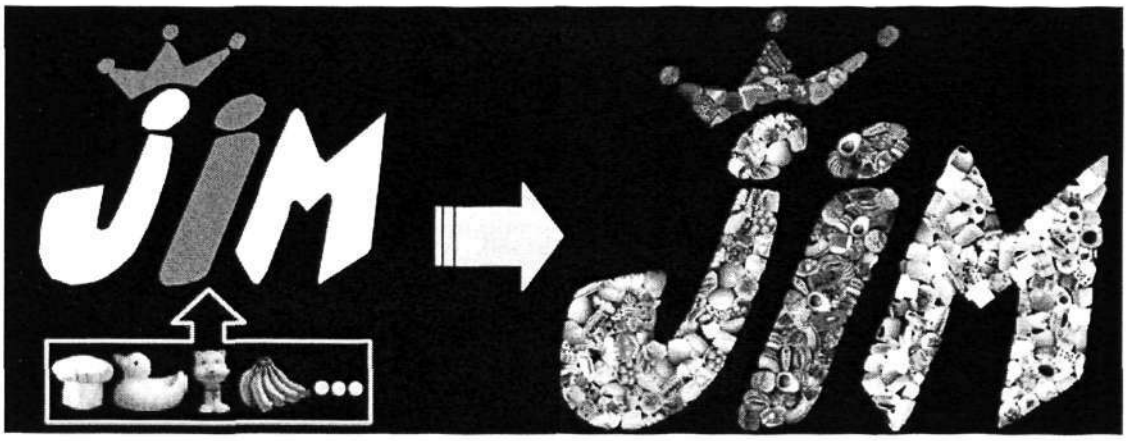
An image-based texturing technique makes use of one or multiple pieces of two-dimensional images. These images can be painted by artist, captured with image scanner or digital camera. A texture mapping process is needed to create correspondences between surface points and texture image pixels. There are two potential problems generally arisen from image-based texturing approach. First, a given texture might not be large enough to cover the entire object surface. Methods have been proposed to alleviate this problem by repeatedly tiling the texture image. However, it is very difficult to blend the patterns at the sides of the texture images. This is also commonly referred as a texture seam problem. Furthermore, even if the tiling is continuous, the repetitive appearance of the wrapped object may be looked unnatural.

A texture synthesis method tries to solve the texture seam problem. Using this method, one or more smaller texture images are used as an input to the system for generating a larger and richer image [17] [18]. This resultant image is then used to wrap onto the entire object surface.

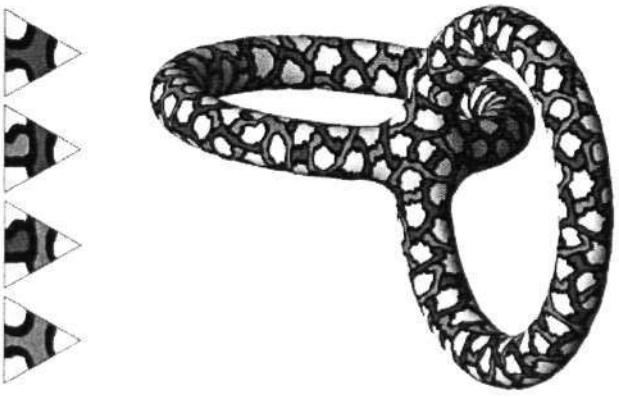
An interesting work by Kim [19] shows arbitrary-shaped image tiles are filled onto an arbitrary-shaped container image. The container is filled as compact as possible and the color of tiles is changed to match the color of a container (See Figure 2.6(a)).

In a method proposed by Neyret [20], several triangular texture samples, which follow specific boundary conditions, are chosen and mapped in a non-periodic fashion onto a surface (See Figure 2.6(b)). Efros [21] uses an image-quilting technique, which small patches of an image are stitched together to create a larger image with similar appearance.

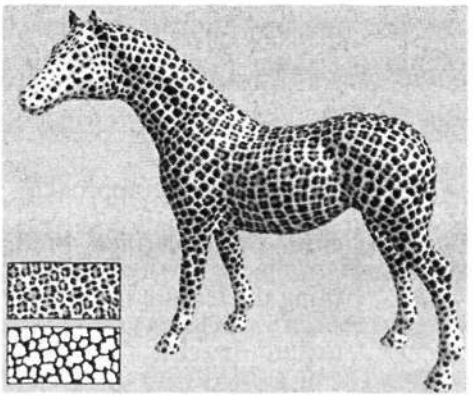
Another problem is that a two-dimensional image does not always fit well to an arbitrary surface manifold. Distortion and texture discontinuity are the common problems by using this kind of technique. Various methods have been proposed to overcome this problem.



(a) Jigsaw image mosaic [19]



(b) Surface mapped with triangular texture samples [20]



(c) Progressively variant texture [22]

Figure 2.6: Various texture synthesis approaches

Xin [23] describes a bi-directional texture function, which creates seamless texture from an input texture. Praun [24] gives an approach to cover an arbitrary surface with texture patches from an input image. This results in a surface, which is covered with overlapping texture patches. A method of rendering the texture in real time is also proposed by the composition of overlapping surface patches.

Soler [25] proposed a hierarchical pattern mapping technique, which iteratively filled a geometry with small patches of an input image in different sizes and shapes. The output of the model is the texture coordinates of the original image for each triangle of the mesh, hence minimizes the memory cost and preserves the rendering speed.

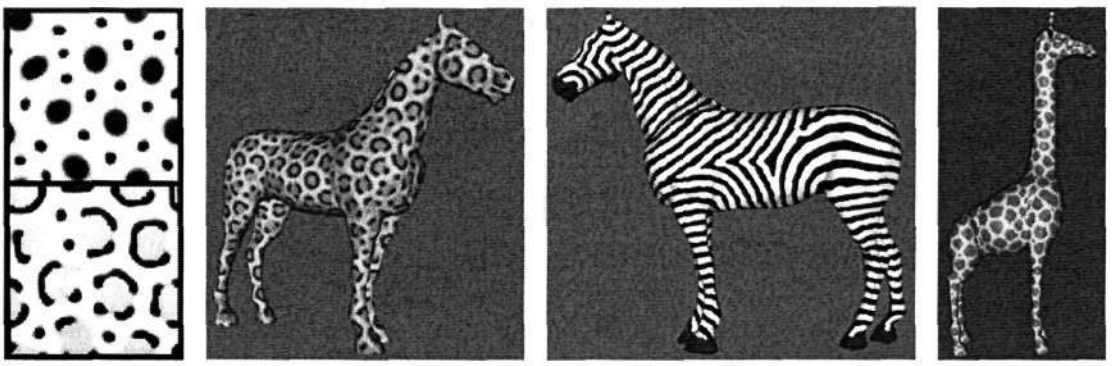
Jingdan [22] shows a progressively variant synthesis on arbitrary surface (See Figure 2.6(c)). A featured-based warping allows the user to control variation of texture elements and a featured-based blending creates smooth transition between any two homogeneous textures. A texton mask is also used to mark prominent texture element in the texture sample. This is to maintain integrity of the synthesized texture elements on the target surface.

Kun Zhou [26] suggested a *TextureMontage* technique, which creates texture atlas for a geometry from multiple input images. Correspondences are specified between a geometry partition and an image. Texture inpainting technique is used to fill in those places where no correspondence is specified.

2.2.2 Procedural Texturing

The most widely used procedural texture is a *Perlin Noise* [27]. It can be used to generate variety of two-dimensional and three-dimensional textures, including marble, water, fire and many other kinds of natural phenomena [28] [29] [30]. It consists of a noise function, which generates random numbers and an interpolation function, which smooth out the return values from noise function.

Turk [31] [8] used a biological pattern formation technique, *Reaction-Diffusion* model, as a tool to generate mammalian coat pattern on arbitrary surfaces. The system simulates the interaction of chemical pigments as they diffuse over the surface. Turk used a suggestion from Bard [32], where a complex pattern can be generated by a cascade process. The process combines the results of several *Reaction-Diffusion* systems by freezing one of the



(a) Cascade Reaction-Diffusion process (b) Leopard horse (c) Zebra (d) Giraffe

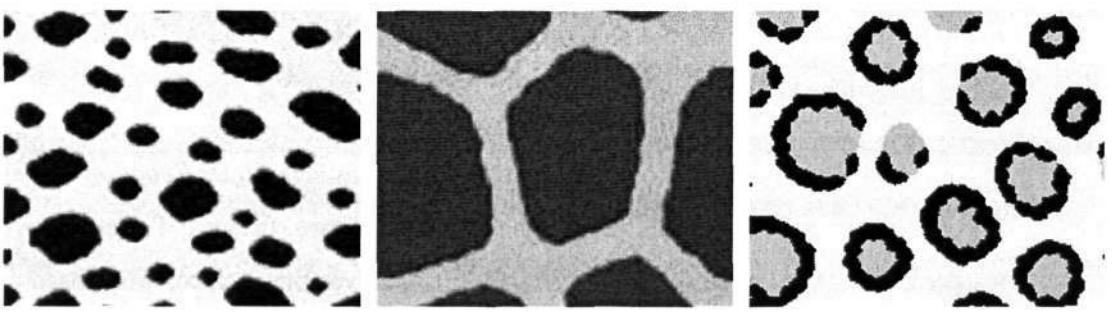
Figure 2.7: Simulate Reaction-Diffusion texture on arbitrary surfaces [8]

processes while running the other process. This cascading process successfully simulated the typical large and small spots on cheetah and rosette patterns on leopard (See Figure 2.7(a)). The results are natural looking animal spots and strips (See Figure 2.7(b) - 2.7(d)). Additional efforts are also given in controlling local variation of patterns, which depends on the curvature of the geometry.

Witkin and Kass [33] introduced the anisotropy concept into the *Reaction-Diffusion* system. The concept allows different diffusion rates for horizontal and vertical directions. The anisotropy information is given by a diffusion-map, which must be defined by the user.

Fleischer [34] [35] [36] proposed a *Cellular Texture Generation* method to simulate surface details such as scales, feathers and thorns. The approach used a biologically motivated cellular development simulation to generate patterns of geometric elements.

Walter [9] [37] [2] proposed a *Clonal Mosaic* model, where attractive patterns are formed by computing repulsion among cells (See Figure 2.8). This model considers cell



(a) Cheetah (b) Giraffe (c) Leopard

Figure 2.8: Texture simulation from Clonal Mosaic model [37]

division, cell mutation and cell repulsion as the key elements to form a pattern. The detailed description of this model will be given in Chapter 3.

2.3 Summary of Literature Review

The finding of exact mechanism for pattern formation in biological development is still an on-going process. Several biological models were proposed, which describe the formation of various color patterns found on animal skins. However, these models only achieve correct approximation on several types of real world patterns, for example, color pigmentation of seashell and color pattern on some fishes. The estimation of mammalian coat pattern is still a problem, where the quality of some of the simulation results is still considerably different from the real world patterns.

In the computer graphics context, there are two types of techniques being used to generate texture on arbitrary surfaces, namely image-based texturing and procedural texturing. In image-based texturing, one or more input images are always needed. Various advancements in this technique give rise to a synthesized texture, which are seamless and non-distorted. The variation of texture elements and transition between any two images are also achievable in an image-based texturing. However, the result from this method is confined by the quality of the input images. On the other hand, some of the complex variations of texture elements, like spots-to-stripes and chevron patterns, are extremely difficult to be obtained by tiling and blending the images.

A procedural texturing approach generates texture patterns directly on object surfaces in an algorithmetic way with a set of controlled parameters. A texture generated by this technique will be seamless and non-distorted. In terms of simulating animal skin patterns, there are two major approaches. The *Reaction-Diffusion* system simulates the diffusion of color pigments on the surface and results in appealing color patterns. The *Clonal Mosaic* model, which simulates cell interaction on surface, gives comparable results.

Chapter 3

Clonal Mosaic Model

Clonal Mosaic model is a method, which aims at simulating mammalian coat patterns on arbitrary polygonal surfaces. The model proposes that the typical spot, rosette and strip patterns, which occur in several species of mammals, reflect a spatial arrangement of epithelial cells. The model also suggests that these epithelial cells are derived from a single progenitor. The color of furs and hairs are then affected by these underlying cells. This highly regulated spatial arrangement of cells can be achieved by controlling three types of cell action, which are the division of cells, mutation of cells and repulsion among cells.

In this chapter, the original concept of the *Clonal Mosaic* model is introduced. The approaches to bring the theoretical model into computer graphics application are discussed in the subsequent sections. At the end of this chapter, the problems and limitations of the existing models are discussed.

3.1 Basic Concepts

Clonal Mosaic model suggests that groups of contiguous cells in an organ are clones, that is, descendants of a common ancestor. For the case of fur formation, cells in different color regions are generated from different progenitors. For example, two types of cell, yellow and black, form the spot pattern are found on cheetah. As mentioned before, there are three types of cell action affecting the pattern formation, which are the division of cell, mutation of cell and cell repulsion.

A cell divides itself into two, in order to produce more progenies of its type. The rate of cell production plays an important role in the pattern formation. Different types of cells

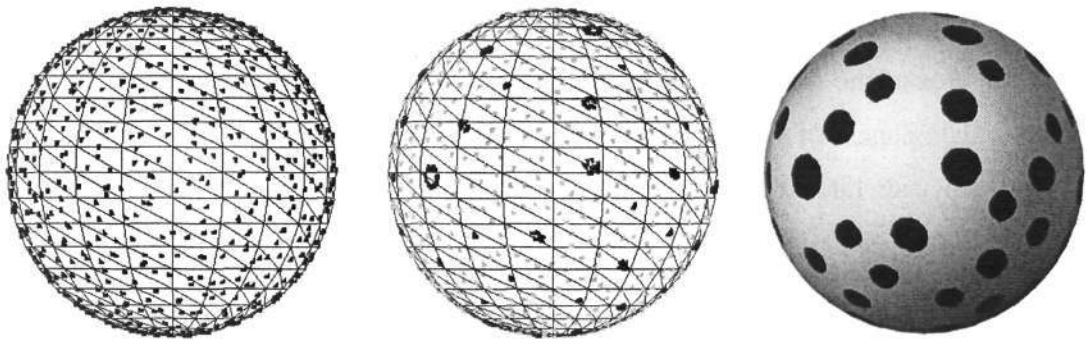
has different division rates. Cells with higher division rate are likely to dominate a certain area of the surface after several generation of evolution. These cells form the significant component of a pattern and, hence are called as foreground cells. Those cells with lower division rate are called as background cells.

If a cell has the ability to mutate itself into another type, a change of pattern is possible. Rosette pattern found on leopard, jaguar and ocelot skin are obtainable by mutating the black cells in a spot pattern. This produces progenies of dark buff color. As the dark buff cells continue to produce more progenies, the rosette pattern is formed.

Another important attribute, which affects the spatial arrangement of cells, is the repulsiveness among cells. In addition to produce more progenies, it is also important that cells of same type will group themselves together. This is the basis in formulating individual elements of a pattern. To achieve this, a mechanism to attract the same type of cells and push away different types of cells is needed. This phenomenal is realizable through a relaxation process, where cells are repulsed according to a repulsiveness value, which is defined between any two types of cell. In order to form a spot pattern, black cells have lower repulsiveness among themselves and higher repulsiveness with yellow cells. This configuration attracts neighboring black cells together and pushes neighboring yellow cells away. Once the process is stabilized, the black cells are likely to be bounded by yellow cells and the pattern is formed.

3.2 Details of Clonal Mosaic Simulation Process

The theoretical *Clonal Mosaic* model describes the properties and behaviors of epithelial cell in the real world. To develop an effective synthesis technique based on the theoretical *Clonal Mosaic* model, both simplification and generalization will have to be considered. Figure 3.1 shows the overall workflow of a *Clonal Mosaic* model. Figure 3.1(a) shows that cells are randomly distributed on a surface. Figure 3.1(b) shows that the cells go through a *Clonal Mosaic* simulation and a regulated pattern is formed. 3.1(c) shows the rendering result of the *Clonal Mosaic* pattern. The details of the process for simulating the biological *Clonal Mosaic* model will be presented in the subsequent sub-sections.



(a) Distribution of cells (b) Clonal Mosaic simulation (c) Rendering of Clonal Mosaic pattern

Figure 3.1: Overall workflow of a Clonal Mosaic model

3.2.1 Representation

It is very tedious to exhaustively represent all the properties of the biological cells and their interactions for synthesizing the formation of animal skin patterns. As a result, a simplified representation scheme is defined, where a system cell represents a group of biological cells. This representation is a feasible option for two valid reasons. First, if the production of biological cells is context-insensitive, the ratio of system cells to biological cells remains the same after several cell divisions. Secondly, if individual cells are subjected to the same force, they can then be replaced by a single cell, which subjects to a resultant force. Using this representation, there may be cases whereby the desired size of patterns is smaller than the one generated by the simulation. Under this situation, the required patterns cannot be synthesized using this representation. A realistic result can be obtained by representing individual biological cell as a single system cell in our simulation. However, the computational performance will be highly affected by this representation as millions of system cells will have to be involved in the simulation. In order to attain reasonable system performance for the simulation, a system cell is employed for abstracting a group of biological cells, but at the expense of slightly lower accuracy. The term *cell* will be used to refer to a system cell in the rest of the thesis.

In computer graphics, an animal skin can be represented in several forms, mainly meshes and NURBS[1]. A cell can be represented on a NURBS surface by specifying its two-dimensional UV coordinate. However, it is difficult to evenly distribute cells on a freeform NURBS surface. Besides, distances between cells, which are important in calculating repulsive force, are also difficult to be approximated on a NURBS surface.

[1]Non-uniform rational B-spline

In the current trend of animation productions, mesh geometry (i.e. polygons) is preferred for modeling organic objects like human and animal, due to its flexibility. It is relatively easy to compute the area of an individual triangle. Hence, even distribution of cell is easily achievable. Besides, distance between cells on the mesh surface can be computed by finding the shortest path between two cells. Although finding the shortest path is a non-trivial task, various approximation methods can still be applied to simplify the computation. The spatial position of the cell on the mesh can be represented in world coordinate. Alternatively, a cell residing in a polygon can also be measured in barycentric coordinate.

In this work, mesh surface (i.e. polygon surface) is chosen to describe animal skins as it is commonly used for organic modeling in animation production. This is particularly true in the recent years due to its high flexibility and manageable level in facilitating the design and implementation of algorithms. The system requires all mesh model to be triangulated. A triangle is the simplest form of a surface. It is straightforward to compute the area, normal and barycentric coordinate [38] from a triangle. Besides, triangle guaranteed a planar surface, which is an important attribute for distance approximation.

3.2.2 Initialization

Random points are distributed over the polygonal surface to identify the position of cells. The algorithm presented by Turk [39] is used for this purpose, as it can ensure a randomly uniform distribution of points over a given polygonal surface. Turk provides a method, which first picks a random triangle and then subsequently picks a random point within the triangle. The random number follows a uniform distribution function [40]. Each cell is positioned by their barycentric coordinate in the triangle, in which it resides.

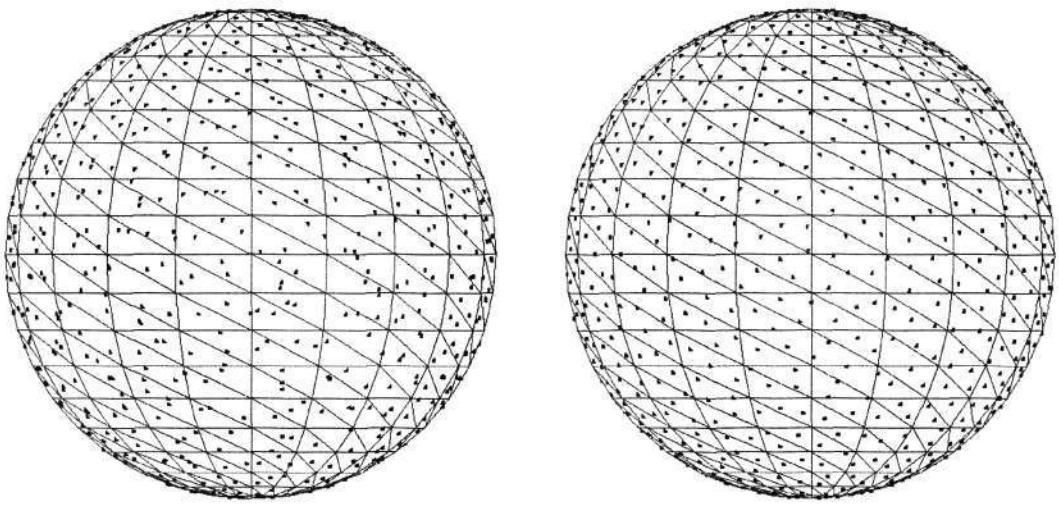
All cells on the surface will be assigned to a type. The properties of a cell type are cell color, division rate, mutation probability and repulsiveness value. Each cell will be assigned with a probability of being classified to a particular type. The mutation probability and repulsiveness values are defined for each pair of types. Table 3.1 shows the attributes.

Cells are randomly distributed over the mesh surface according to the ratio of the area of a triangle over the total area of surface. The cells are evenly distributed over the surface but not evenly spread on the surface. As mentioned before, a relaxation process distributes cells according to the repulsiveness among cells. If all cells are relaxed with the same

Attribute	Meaning	Type
Color	Red, green and blue color	float [0.0, 1.0]
Division rate	Mean time between division of cell	float
Initial probability	Probability of a cell to be assigned to a type	float [0.0, 1.0]
Mutation probability	Probability of a cell to mutate into another type	float [0.0, 1.0]
Repulsiveness	Value to control the repulsiveness between two cells	float [0.0, 1.0]

Table 3.1: Attributes of a cell

repulsiveness value, an evenly spaced cell collection will be obtained. Figure 3.2(a) shows the distribution of cells with their positions computed using a uniformly random generator. Figure 3.2(b) shows the evenly spaced cells after the relaxation.



(a) Random distribution of cells (b) Evenly spaced cells after relaxation

Figure 3.2: Initialization of Clonal Mosaic simulation

3.2.3 Simulation

As mentioned before, the formation of a *Clonal Mosaic* pattern relies on three types of cell action, namely, cell division, cell mutation and cell repulsion. These events can be modeled through an event priority queue implemented as a heap [41]. A heap is a specialized tree-based data structure, which helps in organizing the prioritized cell division and mutation events. Since a mutation of cell always happens at the time of cell division, mutation and division can then be treated as one single event. Besides, the movement of individual cells due to repulsion can be described by an overall relaxation process. In a relaxation event, the cells repulse themselves according to the repulsiveness value defined between two different types of cell. As a result, only the division and relaxation event are needed to be modeled.

Division Event

Typically, the queue will have many regular relaxation events and some irregular division events. For a division event, a *parent cell* divides itself and produces a *child cell*. Depending on the mutation probability, it is possible to have the type of a child cell to be different from its parent. The position of the child cell is randomly chosen within a repulsive circle. The exact time of a division event for a cell is determined by a Poisson distribution with an average equal to the cell's division rate. It models the small variations on the timing for division, so that all the cells will not be divided at the same time.

Relaxation Event

The rate of relaxation is controllable by the users. A larger number of relaxation events, as compared with the number of division events, are needed, in order to allow time for relaxation forces to balance over the domain. As the pattern will evolve at different simulation times, the synthesis should be terminated once the desired pattern has been attained. Hence, the users will have to visually monitor the process for attaining desired result.

The relaxation is an expensive task in the simulation process. Cell's movement on the surface, due to the repulsion from their neighboring cells, is computed in this process. It leads to a cost of $O(n^2)$ if the neighbors of each cell are to be determined among all the cells. The problem can be simplified on a polyhedron, as the neighboring cells are likely to be resided on neighboring triangles of the master triangle, which a cell is located. Hence, the problem is reduced to a searching of neighboring cells from the nearby triangles. There is still a problem if the size of the triangles is smaller than the ideal size of a cell, which means a large number of neighboring triangles have to be traversed. The ideal case will be a search, which is limited to primary and secondary neighbors[2] of a triangle.

The relaxation process consists of several important steps to ensure small approximation error and error free cells' movement.

Cell Repulsion Cells repulse each other according to a preset repulsiveness value, in order to form a regulated pattern. A cell receives repulsive forces from neighboring cells,

[2]Primary neighbor refers to a triangle, which is connected through an edge. Secondary neighbor refers to a triangle, which is connected through a vertex. For those neighbors, which are not directly connected, are referred as the tertiary neighbors

which reside within a repulsive circle. The repulsive circle has a repulsive radius, which is proportional to the average "ideal" area of each cell. For a given total surface area A and m number of cells, the repulsive radius is given as $r = w_r\sqrt{A/m}$, where w_r is a scaling value. The resultant force from neighboring cells is computed and the movement of a cell is applied according to the direction and strength of the force.

Generally, a cell receives larger force from closer cells than further ones. Hence, we formulate the repulsion function by using the distance vector between any two cells. Figure 3.3 shows a cell P with neighboring cells of P_1, P_2 and P_3. f_1, f_2 and f_3 are the displacement vectors applying on P due to the repulsive force from their respective neighboring cells. Meanwhile, each of the neighboring cells is also displaced by opposite displacement vectors f_1', f_2' and f_3'. A composite displacement $f = f_1+f_2+f_3$ is computed. P is then displaced by f and moves to P_{new}.

The distance between cells is used to judge the strength of repulsion. The distance vector d_i between a cell P and its neighboring cell P_i is first calculated.

$$d_i = P - P_i$$

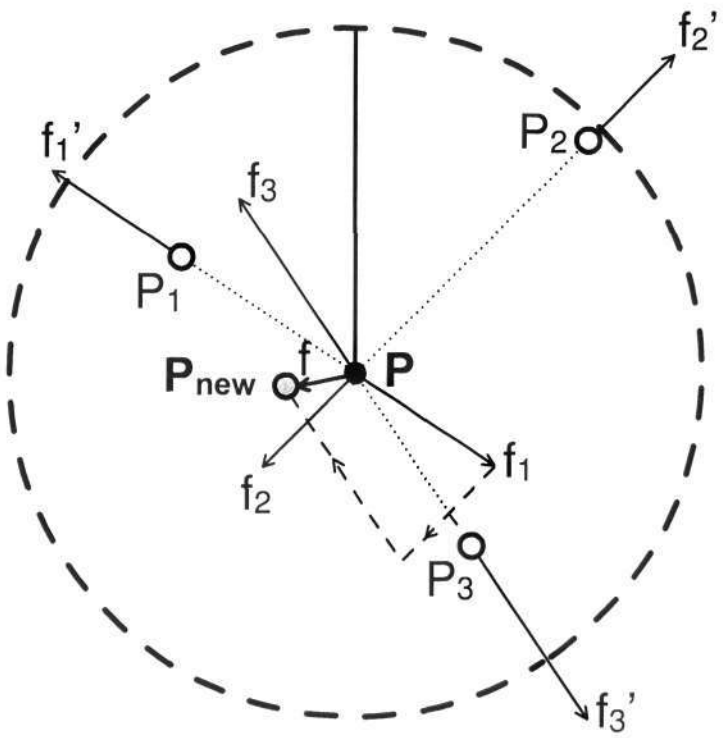


Figure 3.3: Calculation of repulsive forces among cells

A scalar a_i for controlling the strength of repulsion is given as follows:

$$a_i = 1.0 - \frac{|d_i|}{r}$$

Where r indicates the repulsive radius. The displacement is then given by following equation:

$$f_i = w_d \frac{d_i}{|d_i|} a_i \alpha_{p_i c} r$$

Where $\alpha_{p_i c}$ is the repulsiveness value defined between any two types of cell. Repulsive weight w_d is a weighting factor defined by the users. Finally, the total displacement[3] P_{new} is given by summing up the displacements resulted by n number of neighboring cells:

$$P_{new} = P + \sum_{i=1}^{n} f_i \tag{3.1}$$

Cell-in-Triangle Test A given displacement vector may move a cell out from its original triangle. Since an arbitrary mesh surface is generally non-planar, this movement will cause the cell floating in the air. The mechanism to detect if a cell crosses the boundary of a triangle is called as a cell-in-triangle Test. Taylor [42] gives a summary and comparison on various test methods.

The edge, which the cell crosses, will be identified. The cell is then mapped onto the triangle, which is on the other side of the edge. The test continues until the cell is rotated onto and is completely residing within a triangle (See Figure 3.4).

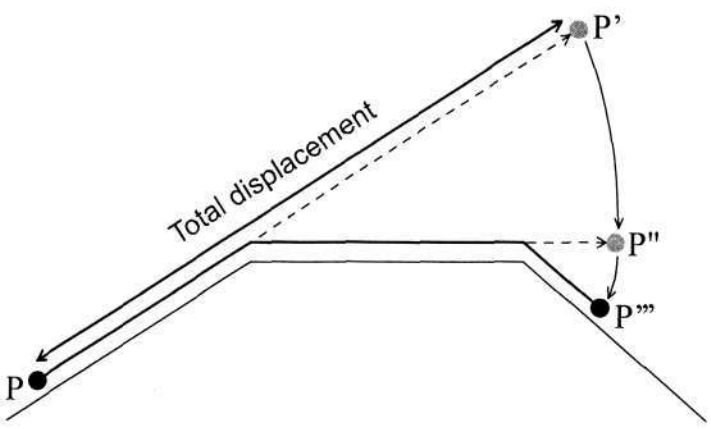


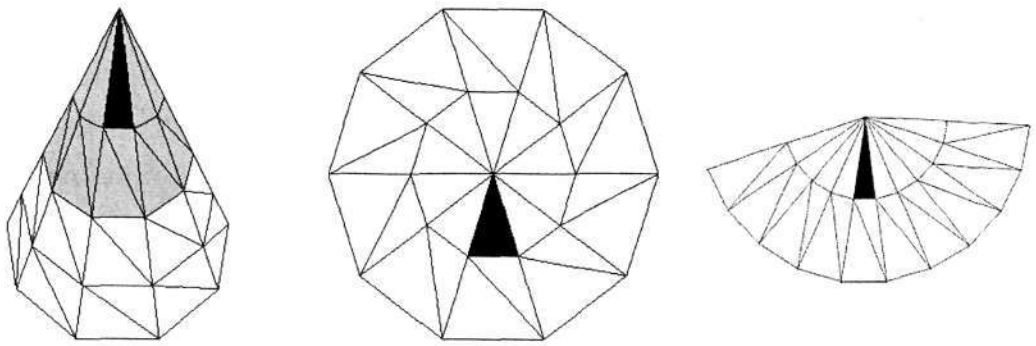
Figure 3.4: Mapping cell from one triangle to another triangle

[3]The classical Clonal Mosaic model [37] considers an anisotropy factor, which displaces a cell in a certain direction, in order to form strip patterns. It is omitted here for simplicity

Triangle and Cell Mapping Distance vector between two cells is obtained during the repulsion computation. Euclidean distance can be used if both cells reside in the same triangle. If the triangles, which the cells reside, are not coplanar, the Euclidean distance will be erroneous.

The triangle, where a cell in question resides, is named as master triangle. The neighboring triangles can be mapped onto a common plane with the master triangle and the approximated distance can be measured. The process to map the neighboring triangles and cells onto a common plane is called as a *Neighbor Mapping* process. *Neighbor Mapping* can be done in various ways, mainly projection and rotation. A projection preserves the topology of the original triangles. However, the distance will only be properly or appropriately approximated if this neighborhood is nearly flattened. Figure 3.5(a) masks the neighborhood of a master triangle (in black) and its neighboring triangles (in grey) on a cone. Figure 3.5(b) shows that this set of triangles are projected onto a common plane. The topology of the neighborhood is maintained, but the size and shape of the triangles are distorted.

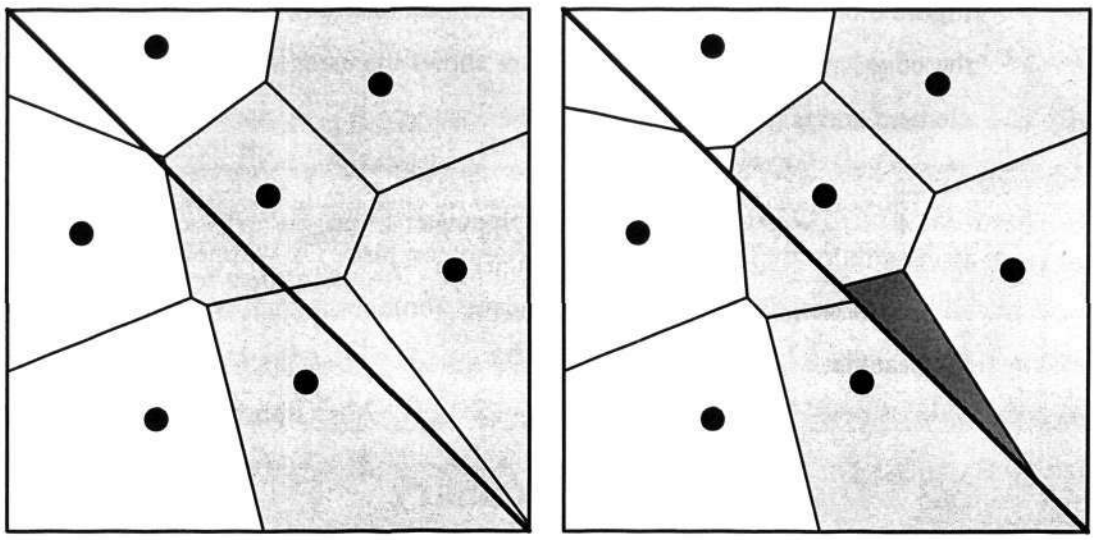
On the other hand, the mapping can be done by rotating a neighboring triangle. Primary neighbors are rotated about the sharing edge by the amount equal to its angular difference with the master triangle. A series of rotation is applied for secondary and tertiary triangles. This method preserves the size and shape of a triangle and hence preserves the distance of cells on any two connected triangles. However, the overall topology of the neighborhood might be distorted. Figure 3.5(c) shows the result of rotating the neighboring triangles of a master triangle.



(a) A neighborhood of triangles (b) Neighbor Mapping by projection (c) Neighbor Mapping by rotation

Figure 3.5: Neighbor Mapping of triangles

3.2.4 Voronoi Diagram Computation



(a) Classical Voronoi diagram (b) Approximated Voronoi diagram

Figure 3.6: Voronoi diagram on polyhedron

Computing Voronoi diagram on a polygonal surface is a complicated task. Okabe [43] defines a *Polyhedral Voronoi Diagram* as a Voronoi diagram where the sites are defined on the surface of a polyhedron and the distances are measured on this surface. For a polyhedron surface, there may be multiple paths for a site. It is also difficult and computationally expensive to determine the shortest path. As a result, most of the solutions to this problem use an approximation to a real Voronoi diagram.

For the case of the classical *Clonal Mosaic* model, the Voronoi diagram on each triangle is computed separately. The final Voronoi diagram is then the combination of the Voronoi diagram on individual triangles. This approach is faster in computation as compared with a classical Voronoi diagram evaluation. The improvement in computational speed is resulted by the avoidance in finding the shortest path between any two points on a polygonal surface. However, the Voronoi polygons, which are supposed to be continuous across triangles, will be distorted along the triangle edges. Nevertheless, the approximation is still valid for two reasons. First, the pattern is defined through a large number of cells per triangle. This approach only affects those cells, which are nearest to the edge of a triangle. Second, the pattern is defined by the overall combination of many cells, which possibly spread over many triangles. This error is non-visible if it occurs within a given pattern element.

Figure 3.6 shows a difference between an actual Voronoi diagram and an approximated Voronoi diagram on a polyhedron. There are two types of cells shown in the figures, white

and gray. A diagonal black line indicates an edge, which is shared by two triangles. Figure 3.6(a) shows an classical Voronoi diagram, which is continuous between the triangles. Figure 3.6(b) shows an approximated Voronoi diagram, which has its sides clipped against the edge. A darken area in this figure shows the error section, which resides in a pattern element and is non-visible.

The computation of Voronoi diagram for each triangle is straightforward. A triangle and its neighbors are first mapped onto a common plane. A Voronoi diagram is then computed for all the neighboring cells. Finally, the Voronoi diagram is clipped against the edges of the triangle.

3.3 Discussion and Summary

Clonal Mosaic model provides a reliably robust and simple mechanism to synthesize animal coat patterns on arbitrary surfaces. It avoids tedious texture mapping process, where additional attention is needed for texture stretching and discontinuity. The model grows the pattern directly on a piece of geometry, as a result, no discontinuity of patterns would be found under most circumstances. However, there are several limitations for this model.

First, the relaxation process is a complicated and CPU intensive task. The most expensive computation cost is incurred during the repulsion computation, where cells are mapped onto a common plane for distance measurement. Another time consuming process is the checking of cell displacement, which ensures a cell is constrained in a neighboring triangle after the displacement. Both of the tasks involve the computation of three-dimensional rotation matrix between any two triangles. It is necessary to devise a more effective method for simplifying the triangle and cell mapping function. In this regard, we propose a *Canonical Reference Plane Structure* [1] onto which an object is pre-processed by mapping all its triangles. Once done, the subsequent computation will only involve a simple two-dimensional transformation and distance calculation. This will further facilitate the necessity in achieving surface planarity, *Neighbor Mapping* and constraining cells on a plane. The details of this proposed method are presented in Chapter 4.

A relaxation process aims at balancing the forces among cells, especially after a cell division. Given a region where no new cell is created, it is a waste of computational power to relax this region, if a stable spatial arrangement has already been reached. Computation

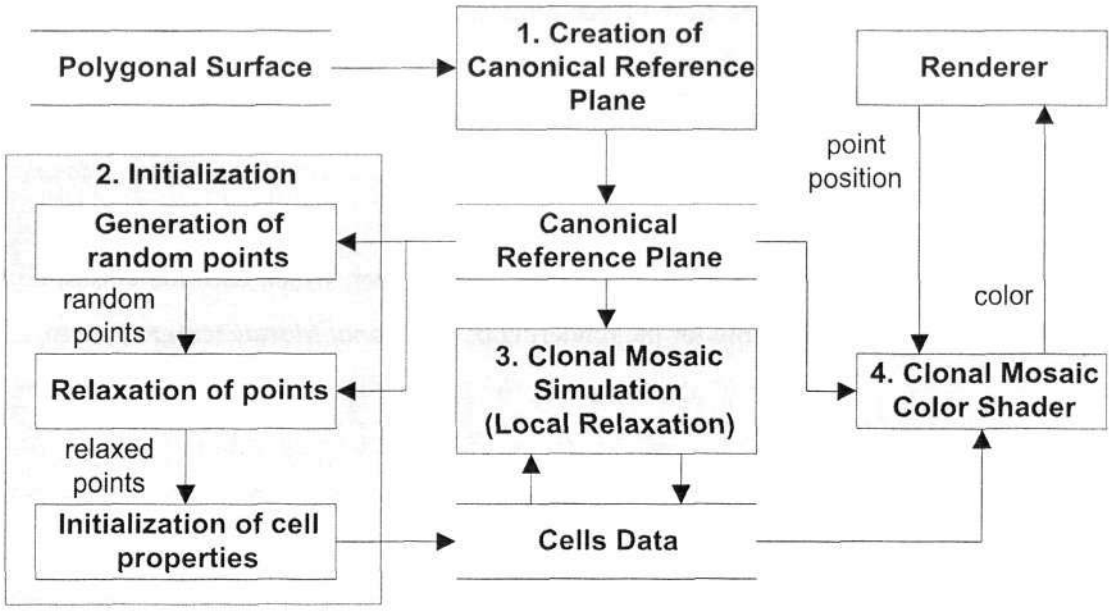


Figure 3.7: Workflow of the system with the proposed solution methods

cost can be saved, if the relaxation process is only applied on the region, where a cell is newly created. The classical relaxation process is hence referred as a *Global Relaxation* process, which relaxes all cells on a surface in each relaxation event. In this project, a *Local Relaxation* process is proposed [1] for relaxing only cells in a small region, whenever a new cell is produced.

The Voronoi diagram of cells is approximated by computing the diagram for individual triangles. The result is acceptable as most of the approximation errors are non-visible in the final pattern. However, it is expensive if this structure is to be directly used for rendering. Whenever a point on the surface is to be rendered, the system has to determine which Voronoi polygon does this point reside, so that a correct color can be assigned. A point-in-polygon test is then executed until a Voronoi Polygon is found. Another option is to preprocess the data and create a two-dimensional image for texture mapping purposes.

This returns the problem to the need of creating a good UV map for the polygonal model. Even worst, the problem of stretching and distortion cannot be controlled from the model. We proposed a method, which searches the structure for a closest cell to a point in question. A point will then be given the color of the closest cell, which implies a Voronoi diagram in the result of rendering. Further discussion will be given in Chapter 6.

To summarize, the *Clonal Mosaic* model imitates biological cell actions, which produce realistic animal skin patterns. However, it suffers from heavy simulation and rendering process. Several methods are proposed in order to improve the overall system performance.

Figure 3.7 shows the flow of our proposed system. A polygonal surface is taken as the system input. First, the surface is converted into our proposed *Canonical Reference Plane Structure*. Second, an initialization process takes places. Random points are generated, relaxed and initialized with proper cell properties. Next, the *Clonal Mosaic* simulation process, which uses the *Local Relaxation* approach, produces cells in a regulated spatial arrangement. Lastly, a *Clonal Mosaic* color shader, which uses the closest-cell-searching method, is responsible for the rendering of the *Clonal Mosaic* texture pattern.

Chapter 4

Canonical Reference Plane Structure

The *Clonal Mosaic* model relies heavily on the calculation of distance vector between cells on an arbitrary surface. The length of a distance vector decides the strength of repulsion and its direction decides the movement of a cell. As mentioned before, the exact distance between any two cells on a polygonal surface is difficult to be computed. A good approximation of the distance is to map the triangles, which the cells in question reside, onto a common plane. The distance between the cells on the common plane is then taken as the approximated distance. This operation is computationally expensive, as three-dimensional rotation matrix has to be computed for each mapping of triangles. Besides, whenever a cell is pushed out from a triangle, a series of rotation is again needed to map the cell onto a neighboring triangle. The previously proposed mapping method requires one rotation for each primary neighbor and several rotations for secondary and tertiary neighbors, in order to form a flatten neighborhood.

In this chapter, we first study the original *Neighbor Mapping* function, which is done by applying a series of rotation on neighboring triangles. Next, we propose a *Canonical Reference Plane Structure* [1], which converts all the triangles and cells' coordinates into two-dimensional format. This structure helps simplifying the three-dimensional transformation into two-dimensional transformation. Lastly, the algorithm for computing the *Neighbor Mapping* with this data structure is presented.

4.1 Neighbor Mapping Function in Three-Dimension

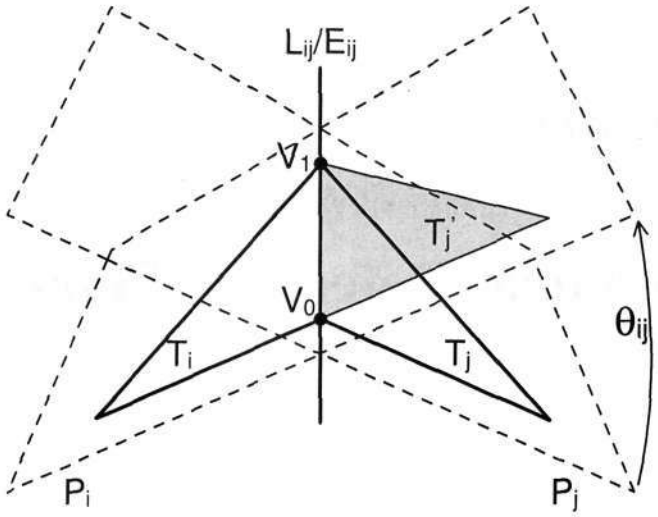
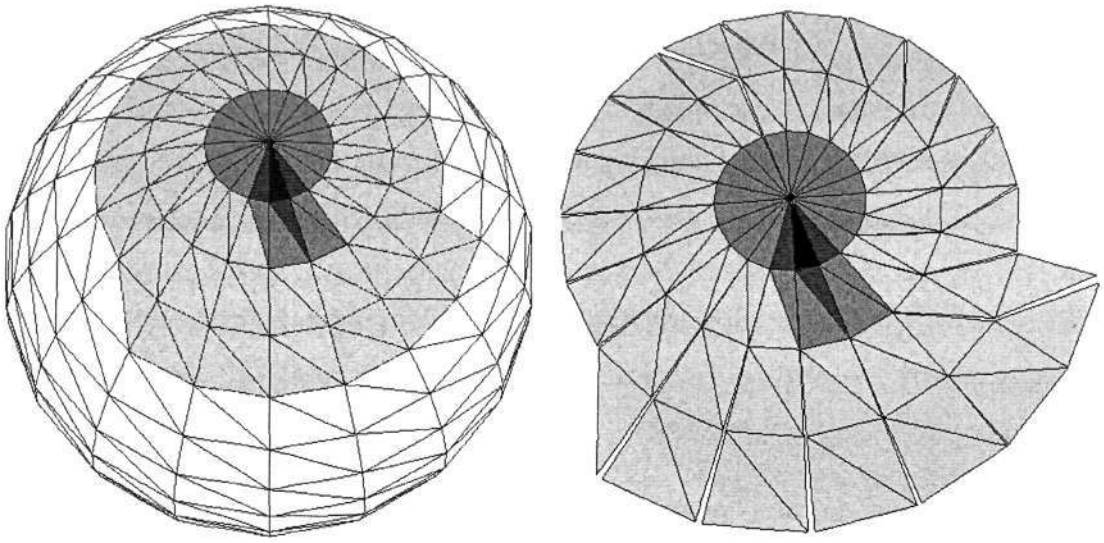


Figure 4.1: Rotate a triangle about a sharing edge

As mentioned in Section 3.2.3, a neighboring triangle is rotated about the sharing edge, in order to map it onto the common plane where a master triangle resides. All of the primary neighbors of a triangle are rotated about their shared edges. Meanwhile, a sequence of edges is obtained as a path to rotate any secondary and tertiary triangles. The solution becomes expensive, as the total rotation is given by a concatenation of all rotations along the path.

For any two non-parallel planes P_i and P_j, they intersect at a line L_{ij}. The intersection line L_{ij} for these planes (where two neighboring triangles reside) can then be taken as a rotation axis for mapping. The intersection line of a master triangle T_i with a primary neighbor T_j is also their sharing edge E_{ij} (See Figure 4.1). Given a triangle T_i and a neighboring triangle T_j, the rotation axis A_{ij} is represented in a vector form and it is computed by subtracting position vectors V_0 and V_1 of the end vertices of the sharing edge E_{ij}. The angle θ_{ij} to rotate T_j to T_i is then given by the angle difference between the normals N_i and N_j of the triangles.

$$\begin{aligned} A_{ij} &= V_0 - V_1 \\ \theta_{ij} &= cos^{-1} \frac{N_i \cdot N_j}{|N_i||N_j|} \end{aligned}$$



(a) Original surface of a sphere (b) A mapped neighborhood

Figure 4.2: Result of mapping neighbors by rotation about intersection line

The secondary and tertiary neighbors are mapped in a breath-first searching approach. Those triangles, which are already mapped to the neighborhood, are named as parent triangles. All primary neighbors of each of the parent triangles are obtained and added into the neighborhood if it is not yet mapped. The transformation of a secondary neighbor is given by the multiplication of its transformation matrix to a primary neighbor and the transformation matrix of the primary neighbor to the master triangle. Subsequently, the tertiary neighbors are also mapped in the same way. This exploration keeps running till a given repulsive radius constraint is reached. These triangles, which satisfied the repulsive radius constraint, is named as a *neighborhood* in subsequent discussion.

A repulsive radius is specified for an example shown in Figure 4.2, all neighbors within this radius have to be mapped. Master triangle is marked in black, primary neighbors in dark gray, secondary neighbors in medium gray and tertiary neighbors in light gray. Figure 4.2(a) shows the original neighborhood before mapping. Figure 4.2(b) shows the flatten neighborhood after applying the *Neighbor Mapping*.

There are several ways to compute a three-dimensional rotation, for example, rotation matrix, axis-angle and quaternion. A detailed comparison of these methods is conducted by Eberly [44] (See Table 4.1). The 'Memory' column lists the numbers of float data needed for each of the methods. The 'Single' and 'Multiple' columns list the numbers of operations, which are needed for transforming a single point and multiple points, respectively. The 'A' and 'M' columns denote the numbers of additions and multiplications, respectively.

Representation	Memory (float)	Single		Multiple		Notes
		A	M	A	M	
Matrix	9	6	9	6n	9n	
Axis-angle	4	12	18	12n	18n	
Quaternion	4	24	32	24n	32n	Using generic quaternion multiplies
Quaternion	4	18	21	12+6n	12+9n	Convert to matrix, then multiply

Table 4.1: Operation counts for transforming single and multiple points [44]

The 'n' under the 'Multiple' column refers to the number of points being rotated in a single transformation operation. A matrix rotation operation is fast to compute but it requires more memory. A quaternion requires less memory but suffers from slower computations.

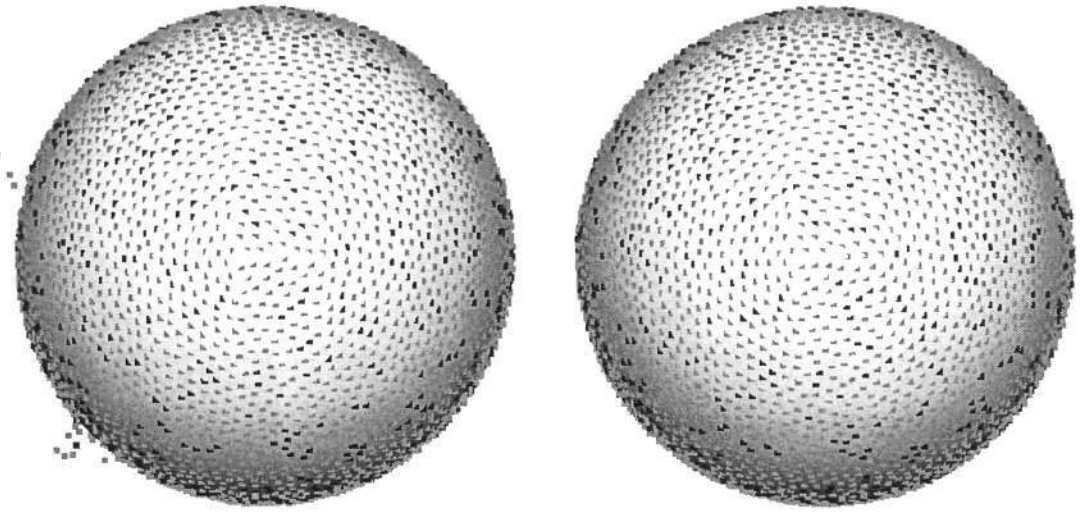
An optimum choice for less memory and faster computation time is to store the rotation information in quaternion and convert it into matrix whenever a rotation is performed. This method is useful, if a number of points are to be rotated together. It requires storage of four floats as quaternion, twelve additions and twelve multiplications for matrix conversion, six additions and nine multiplications to rotate a point.

Neighbor Mapping is not only needed to map a cell onto a common plane for repulsion computation, but is also needed when a cell is being pushed out from a triangle. It ensures that the cell will reside back onto a neighboring triangle after the repulsion. However, during the simulation process, a cell is generally being repulsed for hundred times or even more. After a series of *Neighbor Mapping* process, a cell may not be coplanar with a triangle in which it is supposedly resided. This causes the cell leaving the surface. The major reasons of this phenomenon are listed below:

- Numerical errors introduced in each computation are accumulated.
- The cell, which is not coplanar with the triangle, will repulse other cells situated on the plane with the forces in parallel to direction of the cell leaving the plane.

We shall call this phenomenon as cell-floating problem in the subsequent discussion. Figure 4.3(a) shows an example of the occurrence of the cell-floating problem. In this example, two instances of such problem are observed. Each of these floating cell clusters is likely generated by the repercussion of the effect of a floating cell.

It is certainly not appropriate to use these floating cells for pattern generation during the simulation. Besides, it is also impossible to compute a Voronoi diagram based on this



(a) A worst case where cells are floating outside of the surface (b) An ideal case where cells are lying on the surface

Figure 4.3: Cell-floating problem

set of cells. To resolve the floating-cell problem, a mechanism will have to be devised to check and re-assign a floating cell to an appropriate position on a patch surface (See Figure 4.3(b)). A closest point to plane calculation is used in the classical *Clonal Mosaic* model. For a floating cell to be reverted back to its original position, the calculation of the closest distance of the cell from the plane is required in the classical *Clonal Mosaic* model. This evaluation will have to be conducted for each of the floating cells in order to restore them back onto the surface. The overall computation incurred in this operation can be considerably significant as the number of floating cells is generally large, occupying about 15% of the total number of cells at the end of the simulation.

4.2 Creation of Canonical Reference Plane Structure

In this section, a given geometry is transformed into another structure, which reduces the computational complexity of the classical *Clonal Mosaic* model. This simplification increases the flexibility and reduces the computational cost of the system. By introducing this method, the cell-floating problem will also be resolved.

We first rotate all triangles of a surface in such a way that their normals are aligned with Y-axis. In Figure 4.4, a rotation θ, which rotates normal N_j to $-N_y$ (points to Y-direction), is applied on a triangle. By ignoring the Y-coordinate, all of the triangles are mapped onto a

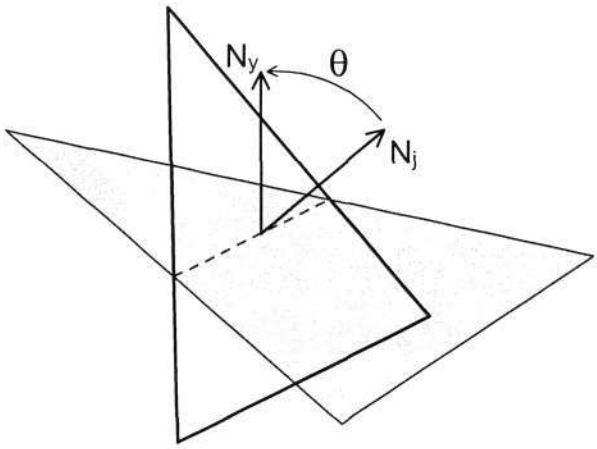
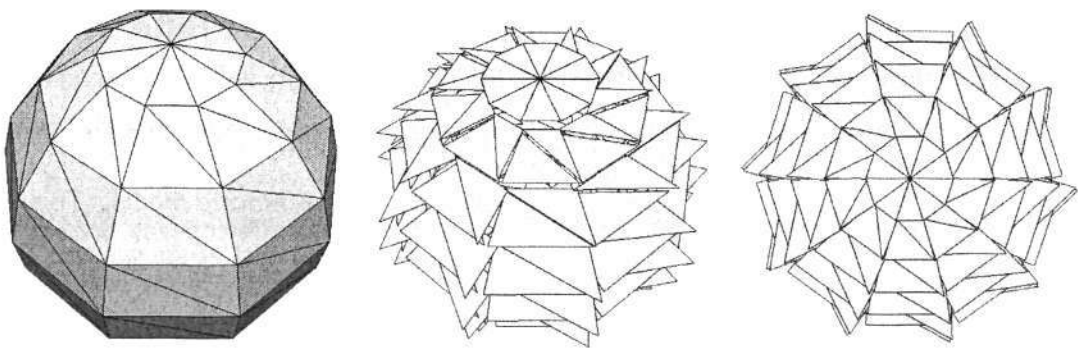


Figure 4.4: Obtaining a Reference Triangle

common plane, which is the X-Z plane. We call the X-Z plane as the *Reference Plane*, and the rotated triangles as the *Reference Triangles*. This mapping keeps the size and shape of the original triangles, which are important in the distance computation. This method also reduces the computation of a distance vector to a two-dimensional problem.

Figure 4.5(a) shows a polygonal geometry. Figure 4.5(b) shows thee *Reference Triangles* after the triangles of the geometry is rotated. Figure 4.5(c) shows the result, which their Y coordinates are ignored. All the triangles are now residing on a two-dimensional domain. As one can observe, triangles under this configuration are disconnected from and overlapping with each other. Nevertheless, the *Neighbor Mapping* function mentioned in Section 4.1 has to be modified. In order to form a neighborhood, a neighbor T_j of a master triangle T_i is first rotated about Y-axis. This will turn it to a proper orientation. Next, it is translated on the *Reference Plane*, which connects its edge with T_i. (See Figure 4.6) The detail computation of a *Neighbor Mapping* function is given in Section 4.3.



(a) Original surface (b) Reference Triangles (c) Reference Triangles in two-dimensional domain

Figure 4.5: Canonical Reference Plane Structure

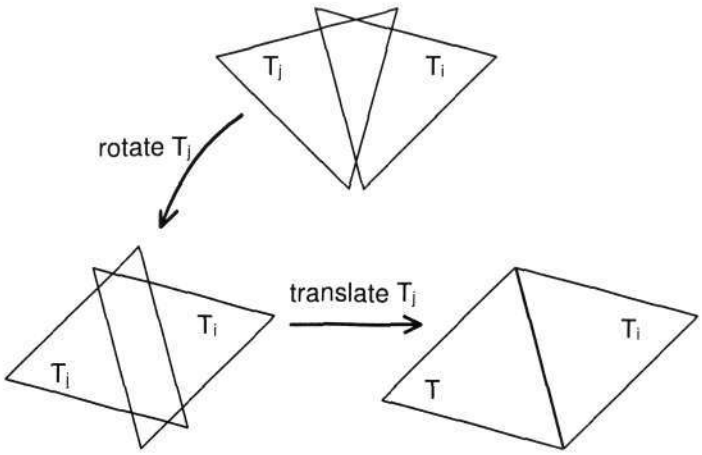
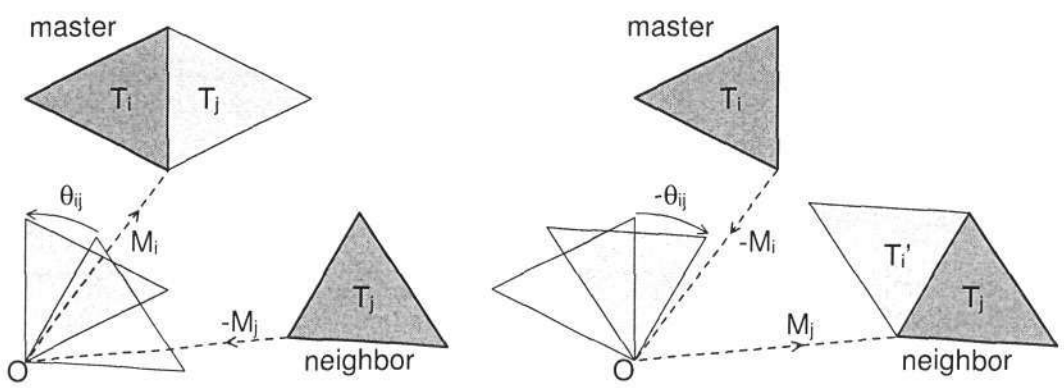


Figure 4.6: Neighbor Mapping for Reference Triangles

The *Canonical Reference Plane Structure* is created before commencing the *Clonal Mosaic* simulation. Initial position of the cells on the Reference Plane is computed through their barycentric coordinates. Once the *Clonal Mosaic* simulation is done, the triangles and cells will undergo a reverse process, in order to restore their actual three-dimensional positions.

4.3 Neighbor Mapping Function on Reference Plane



(a) Transform neighbor to master frame (b) Transform master triangle to a neighbor frame

Figure 4.7: Transformation between master and neighboring triangle

First, the frame, which all neighbors of a master triangle are transformed to, is named as a master frame. The frame, which a neighbor originally resides, is named as a neighbor frame. Figure 4.7(a) shows how a neighbor is transformed to a master frame from a neighbor frame. T_i and T_j are the master and neighbor triangles, respectively. The triangles are connected with a vertex, which has a position of M_i in master frame and M_j in neighbor frame. As

mentioned in the previous section, a rotation and a translation are needed to transform a neighbor to a master frame. In order to bring the neighbor T_j to the master frame, we first translate it to the origin O with $-M_j$, rotate it about the origin O with θ_{ij} and then finally translate it to the master frame with M_i. The translation is decomposed down into two translations, $-M_j$ and M_i. The concatenated transformation function is derived as follows:

$$F_{ji} = T(M_i) \cdot R(\theta_{ij}) \cdot T(-M_j) = \begin{bmatrix} C_{ij} & -S_{ij} & A_x \\ S_{ij} & C_{ij} & A_y \\ 0 & 0 & 1 \end{bmatrix}$$

where

$$\begin{aligned} C_{ij} &= \cos\theta_{ij} \\ S_{ij} &= \sin\theta_{ij} \\ A_x &= -M_{jx}C_{ij} + M_{jy}S_{ij} + M_{ix} \\ A_y &= -M_{jx}S_{ij} - M_{jy}C_{ij} + M_{iy} \end{aligned}$$

Figure 4.7(b) shows the transformation of a master triangle T_i to the neighbor frame in a reverse order. The transformation function is given as follows:

$$F_{ij} = T(M_j) \cdot R(-\theta_{ij}) \cdot T(-M_i) = \begin{bmatrix} C_{ij} & S_{ij} & B_x \\ -S_{ij} & C_{ij} & B_y \\ 0 & 0 & 1 \end{bmatrix}$$

where

$$\begin{aligned} C_{ij} &= \cos\theta_{ij} \\ S_{ij} &= \sin\theta_{ij} \\ B_x &= -M_{ix}C_{ij} - M_{iy}S_{ij} + M_{jx} \\ B_y &= M_{ix}S_{ij} - M_{iy}C_{ij} + M_{jy} \end{aligned}$$

For a tertiary triangle T_n, more than one transformation is needed. A path, which consists of a list of connected triangles from the tertiary triangle to the master triangle, is obtained. The transformation of the triangle is then given by a combined transformation F_T of the intermediate triangles.

$$F_T = F_{01} \cdot F_{12} \ldots F_{(n-1)n}$$

Instead of storing the 3×3 matrix in the memory, we only store the value of C_{ij}, S_{ij}, A_x and A_y for each pair of primary and secondary neighbor. A cell can be transformed to a frame at any time by getting this transformation information for that pair of neighbors. The

Neighbor Mapping function for a cell is given as follows:

$$\begin{aligned} P'_x &= P_x C_{ij} - P_y S_{ij} + A_x \\ P'_y &= P_x S_{ij} + P_y C_{ij} + A_y \end{aligned}$$

As we can see, the operations of transforming a point are reduced to four additions and four multiplications.

4.4 Discussion

The classical algorithm requires storage of four floats, in order to store the transformation information for a pair of neighboring triangles as a quaternion. Nevertheless, twelve additions and twelve multiplications are required for matrix conversion, six additions and nine multiplications are required to rotate a point. This can be an expensive computational process as thousands of cells will have to be properly transformed for every iteration of the simulation. Besides, the cell-floating problem may occur, as numerical error will be accumulated due to the rapid transformation of a cell.

The *Neighbor Mapping* function used in the classical *Clonal Mosaic* transforms cells to a common plane for repulsion computation. We propose the *Canonical Reference Plane Structure*, which transforms all triangles onto a common plane. Cells' positions in this two-dimensional domain is retrieved through their barycentric coordinate. Since all cells are residing on a common plane, the three dimensional operation, which is used in classical *Clonal Mosaic* model, can then be reduced to a two dimensional problem. The *Neighbor Mapping* function is modified to include a two-dimensional rotation and translation of triangles and cells. The storage of four floats is still required for each pair of neighboring triangles. However, the computational cost of transforming a cell is reduced to four additions and four multiplications. Under this configuration, the cell floating problem is implicitly solved, as cells will always be coplanar with the triangles in a two-dimensional domain.

The *Neighbor Mapping* function provides a fast computation for irregular meshes, which also gives acceptable result in the distance approximation. However, for a complex model, it might be more efficient to simplify the model, as speed can be increased by the reduction of polygon count. The drawback is that there might be a lost of information between the simplified and original complex model.

Chapter 5

Enhancement on Clonal Mosaic Simulation

The generation of texture pattern, which bases on a *Clonal Mosaic* model, involves two phases, namely initialization phase and pattern simulation phase. During the initialization phase, random points representing the cells' location are generated and distributed on a surface. These cells are subjected to a relaxation process, which evenly spaces out the cells. The attributes of each cell are then initialized with appropriate cell type.

In the actual pattern simulation, a series of events is queued up in a heap structure. The relaxation and division events are processed as scheduled. In a division event, a cell divides itself and creates a child cell. A mutation of cell might happen during a division event, depending on its mutation rate. During the convergence of the relaxation process, cells re-align themselves until all the forces exerted on the new cells are neutrally canceled out.

In Chapter 3, the classical *Clonal Mosaic* model is presented in detail. Its potential problems and issues are discussed. In Chapter 4, a *Canonical Reference Plane Structure* and a *Neighbor Mapping* function are proposed for the existing model. This proposed modification gives considerable improvements on the computation of the distance between cells across an arbitrary surface and also the transformation of cells on the surface. In this chapter, we propose effective techniques for further improving the computational efficiency of the pattern formulation process.

In the next section, the simulation process of the classical *Clonal Mosaic* model will be first elaborated. Having explained the simulation process, the enhancement of the relaxation process of the model is proposed and implemented. Next, a point-in-polygon strategy is accustomed to trace the cell movement on the surfaces, in order to correctly update the

association of the cell to a triangle.

5.1 Pattern Simulation

In the pattern simulation process, cells on the surface are repulsed and finally converged to a stable spatial arrangement. In this section, the operations of cell division and relaxation events of the simulation process are first described. We will then examine the detailed procedure of the relaxation process applying to cells on a surface. Having realized the drawbacks of the classical *Global Relaxation* process, two approaches for relaxing the cells locally are proposed and developed for alleviating these problems.

5.1.1 Cell Division and Relaxation

1. Obtain division event D_i from event queue
2. If division time T_{D_i} < T_{cur}
   - Perform cell division D_i and return to step 1
3. Else
   - Perform relaxation
   - T_{cur} += ΔT
4. Return to step 1 if T_{cur} < T_{end}

Figure 5.1: Pseudo codes for pattern simulation

A simulation holds a schedule, which contains a series of division event D_i. Figure 5.1 shows the pseudo codes for simulation process. Each of the division events has a time value T_{D_i}, which is the time when a cell should divide itself. T_{end} designates the ending time for a simulation and T_{cur} records the current time. T_{cur} is incremented by ΔT after each iteration of simulation. An element is popped from the event queue upon the start of the simulation. The division is executed if the time value T_{D_i} for this event is smaller than the current time. Next division event will be popped after the previous division event has been executed and the time is again compared. If there is no division to be processed, a relaxation will be called and T_{cur} is incremented. The simulation ends when T_{cur} reaches a predefined ending time T_{end}.

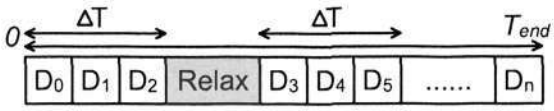


Figure 5.2: An event queue

Figure 5.2 shows a typical event queue. Three divisions occur within the first ΔT. A relaxation is performed after the division events. Several other division events occur in the next ΔT. The simulation continues until T_{end} is reached.

As mentioned before, relaxation is an expensive but important process, which decides the final formation of a texture pattern. The classical relaxation process relaxes all the cells on a surface whenever a relaxation event is launched. Hence, it is called as a *Global Relaxation* process. There exists redundant computation for those neighborhood, where a stable spatial arrangement is already reached. A stable spatial arrangement of a neighborhood is interrupted only when a new cell is added into the neighborhood. As a result, relaxing only cells on these interrupted neighborhoods will efficiently reduce the computational load. As the relaxation operation is only applied to cells within a local neighborhood, it is called as *Local Relaxation* process.

In the next section, the procedure of the *Global Relaxation* process is detailed and its major drawbacks are discussed. Two approaches for alleviating these drawbacks are presented. These methods are mainly applied to cells within a local neighborhood. The first approach is applied onto the interrupted neighborhood, whenever a cell division occurs. The second approach is to maintain a list of interrupted neighborhoods and processes them all together in an event queue.

5.1.2 Global Relaxation

Relaxation events are scheduled in a fixed time interval. In each relaxation event, all cells on the surface are relaxed by calculating a resultant force from the neighboring cells (See Equation [3.1] on page 22). A neighboring cell resides within a repulsive circle with a given repulsive radius r.

Figure 5.3 shows the pseudo codes of a *Global Relaxation* process. The repulsive forces among cells on a triangle T_i are first computed. Neighboring triangles of T_i, which are within the repulsive circle is then obtained. The triangles and their cells are mapped onto T_i with the *Neighbor Mapping* function, which is mentioned in Section 4.3. For those cells,

```
For each iteration of relaxation, perform the
following operations:

 1. For each triangle T_i on the surface

      • Compute repulsive forces among cells on T_i
      • Obtain neighbors of T_i within a radius r
      • Perform Neighbor Mapping to map the neighbors
        and cells onto T_i
      • For each neighbor N_j of T_i
          - Compute repulsive forces between cells on T_i
            and cells on N_j

 2. For each triangle T_i on the surface

      • For each cell C_k on T_i
          - Displace C_k with the resultant force
          - Check if C_k moves out from T_i
```

Figure 5.3: Pseudo codes for the Global Relaxation process

which are within the repulsive circle, their repulsive forces with cells on T_i are computed. Once all of the triangles are processed, all cells are displaced with the resultant forces. To update the association of a cell to a triangle, a cell-in-triangle test will have be performed. Details of the test will be discussed later in Section 5.2.

The advantage of a *Global Relaxation* is that all the cells are always under a relaxed condition. This ensures a solid pattern formation at the end of the simulation. However, the major drawback of this approach is the redundant computations incurred in the simulation. To alleviate this problem, a new approach avoiding these redundant computations is proposed and presented in the following sections.

5.1.3 First Approach of Local Relaxation

Before a simulation begins, several iterations of *Global Relaxation* process are executed, in order to give stable cells arrangement. Subsequent relaxations are performed only when an occurrence of a cell division is detected. Figure 5.4 shows the pseudo codes of this *Local Relaxation* process. The symbol N denotes the number of relaxation to be processed in each cell division event. In our implementation, this number will have to be specified by

For each division event for a cell C_{new}, perform the following operations for N iterations:

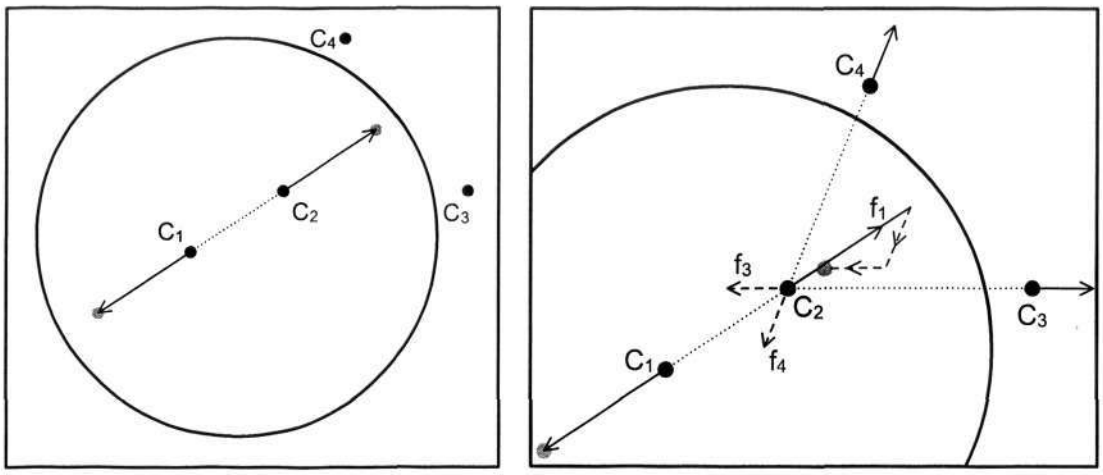
1. Obtain neighbors of T_{new}, where C_{new} resides
2. Add T_{new} and its neighbors into a relax list L_{relax}.
3. For each triangle T_i in L_{relax}
   - Compute repulsive forces among cells on T_i
   - Obtain neighbors of T_i within a radius r
   - Perform Neighbor Mapping to map the neighbors and cells onto T_i
   - For each neighbor N_j of T_i
     - Compute repulsive forces between cells on T_i and cells on N_j
4. For each triangle T_i in L_{relax}
   - For each cell C_k on T_i
     - Displace C_k with the resultant force
     - Check if C_k moves out from T_i

Figure 5.4: Pseudo codes for the first approach of the Local Relaxation process

the users. In order to relax a surface region interrupted by the newly divided cell C_{new}, all neighboring triangles within the repulsive circle are obtained and added to a relax list L_{relax}. The subsequent operations are similar to the *Global Relaxation*.

Repulsions of cell on all these triangles are computed, so that cells can be rearranged. However, relaxing all the cells within this neighborhood does not guarantee a proper arrangement of cells. In Figure 5.5(a), the region bounded by a circle shows the interrupted neighborhood within a repulsive circle. Two internal cells C_1 and C_2 within this region repulse with each other and move towards the boundary of this neighborhood. There are two external cell C_3 and C_4 residing outside but near to this region. It is observed that, the internal cells are too close to these external cells after the displacement. This is undesirable, as a stable arrangement is not met after the relaxation.

To overcome this problem, the repulsion among internal cells and external cells are also computed. This gives the result as shown in Figure 5.5(b). f_1 is a force from internal cell C_1. C_3 and C_4 give the forces f_3 and f_4, which push C_2 backward. A resultant force from f_1, f_3 and f_4 will then give a smaller displacement of C_2, which keeps it away from



(a) Internal cells are repulsed towards boundary (b) Internal cells are depressed from the boundary

Figure 5.5: Computation of cell repulsion for Local Relaxation

the boundary. C_3 and C_4 will not be displaced with the computed repulsive forces as this displacement may affect the arrangement of other neighboring cells.

This approach gives a reasonably fast processing time and acceptable results. However, several division events may occur on a neighborhood in a near interval. It will be more efficient to conduct division events and compute the resultant repulsive forces on all those cells being divided and assigned to the pending list for relaxation. The next approach advances the *Local Relaxation* process by maintaining a list of triangles. The cells on each triangle in the list will be processed and re-distributed at a fixed time interval.

5.1.4 Second Approach of Local Relaxation

Figure 5.6 shows the pseudo codes for the second approach of the *Local Relaxation* process. This method consists of two phases. A cell division event will be performed in the first phase. Once done, a process which is similar to the *Global Relaxation* are scheduled in a fixed time interval ΔT. Whenever a cell division occurs, the neighbors of the triangles T_{new}, where the new cell C_{new} resides, are obtained and added to a list $L_{neighbor}$. Each of the triangles is attributed with a relaxation step field, which defines the iteration number of a relaxation event. All the triangles in $L_{neighbor}$ will have their relaxation step set to a user defined value S. Next, all the triangles are appended to a list of ralaxation L_{relax}, if they have not yet been placed in the list.

In each division event for a cell C_{new}, perform the following operations:

1. Obtain neighbors of T_{new}, where C_{new} resides
2. Add T_{new} and its neighbors into $L_{neighbor}$
3. Set relaxation steps for all triangles in $L_{neighbor}$ to S
4. Add all triangles in $L_{neighbor}$ into a relax list L_{relax}.
5. Triangles are sorted and duplications are removed.

In each iteration of relaxation event, perform the following operations:

1. For each triangle T_i in L_{relax}
   - Compute repulsive forces among cells on T_i
   - Obtain neighbors of T_i within a radius r
   - Perform Neighbor Mapping to map the neighbors and cells onto T_i
   - For each neighbor N_j of T_i
     - Compute repulsive forces between cells on T_i and cells on N_j
2. For each triangle T_i in L_{relax}
   - For each cell C_k on T_i
     - Displace C_k with the resultant force
     - Check if C_k moves out from T_i
   - Decrease the relaxation step of T_i
   - If relaxation step equals to 0, remove T_i from L_{relax}

Figure 5.6: Pseudo codes for the second approach of the Local Relaxation process

In the first approach, relaxation process will have to be iteratively applied to the same interrupted neighbor for every cell division event. This iteration event will have to be performed on each of the interrupted neighbors individually. In Figure 5.7, there are four divisions occurring within a time interval ΔT. Their interrupted neighborhoods are overlapped. If the first *Local Relaxation* approach is used, those cells located in overlapped neighborhoods will be displaced in each division event. Considering the overlapped region between R_1 and R_2, the cell in this region is receiving forces from both cells in R_1 and R_2. Once the total resultant force is evaluated, the cell will be displaced only once. Comparing with the previous approach, some of the external cells are now located inside other neighborhoods. As a result, they will not be stopped from repulsing each others. This makes the approach similar with a *Global Relaxation* process as more cells are being relaxed in each relaxation event.

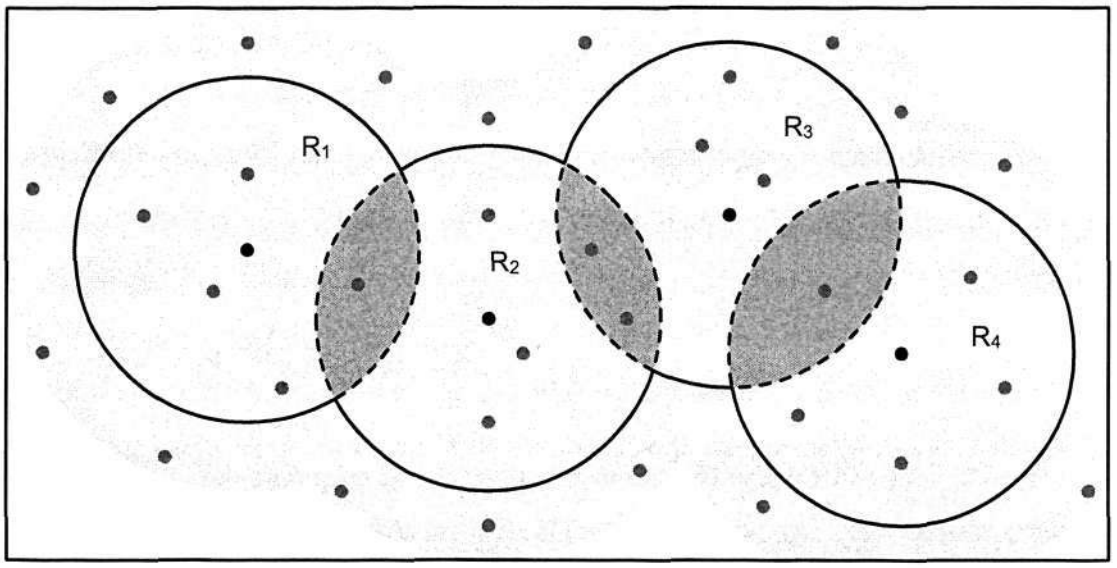
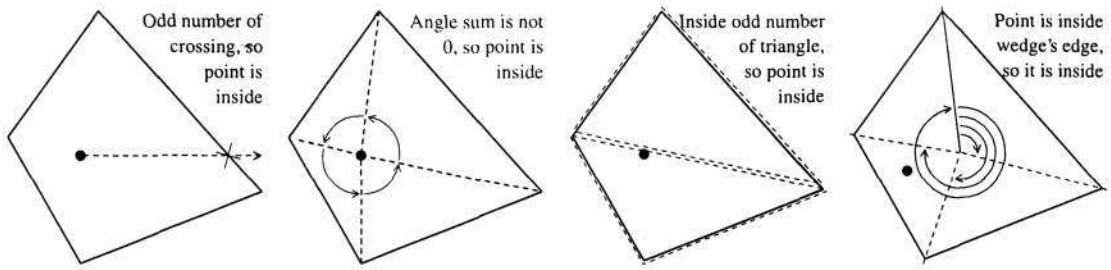


Figure 5.7: Overlapped neighborhoods in the Local Relaxation process

5.2 Cell-in-Triangle Test

Whenever a cell is displaced, it might either stay in the original triangle or move to a neighboring triangle. This problem can be divided into two steps. First, a cell is checked to see if it is still staying inside the original triangle. If it is not, a search is then performed to see which direct neighbor does this cell move to. It is possible that a cell may move to a tertiary neighbor. Hence, several cell-in-triangle tests and neighbor searching are needed until a cell is found resting within a triangle.



(a) Crossing Test (b) Angle Sum Test (c) Triangle Test (d) Inclusion Test

Figure 5.8: Various point-in-polygon tests

We adopt the idea of a commonly available point-in-polygon test. Haines [45] introduces a variety of point-in-polygon testing methods, for example, crossing test, angle summation test, triangle test and inclusion test (See Figure 5.8). Despite having an efficient point-in-polygon strategy, it is preferred that this strategy can also determine the neighbor at which a "out" cell is situated. In this aspect, Taylor [42] describes a fast and efficient method for conducting a crossing test by using a point vector representation of a straight line. The intersection of line segments is computed in order to determine crossing. For a line segment L defined by two end points $P_0(x_0, y_0)$ and $P_1(x_1, y_1)$, it is represented as:

$$L = \{P_0 + r(P_1 - P_0) | 0 < r < 1\}$$

For two line segments L and L', an intersection occurs when

$$P_0 + r(P_1 - P_0) = P_0' + r'(P_1' - P_0')$$

There exist an intersection if both r and r' lie within $0, 1$. r and r' can be obtained as follows:

$$\begin{aligned} D &= (y_1 - y_0)(x_1' - x_0') - (x_1 - x_0)(y_1' - y_0') \\ D_1 &= (y_1' - y_0')(x_0' - x_0) - (x_1' - x_0')(y_0' - y_0) \\ D_2 &= (y_0' - y_0)(x_1 - x_0) - (x_0' - x_0)(y_1 - y_0) \end{aligned}$$

Then

$$r = \frac{D_1}{D}, \qquad r' = \frac{D_2}{D}$$

If L and L' are parallel, D will be zero. If L and L' are collinear, then D, D_1 and D_2 will all be zero. A real number r lies between $0, 1$ if and only if the product $r(1-r)$ is non-negative. If both r and r' satisfy this test, a crossing is obtained.

This strategy uses a test ray emitted from a point and calculates its intersection with the edges of a polygon. Here, the modification on Taylor's algorithm, which solves both the

For each movement of a cell P_0, set T_c as the current triangle, which P_0 resides. Set L_p as the motion path of P_0. Following operations are performed:

1. If L_p is inside T_c, return.
2. If L_p crosses a boundary edge, restore original position of P_0 and return.
3. Set the triangle, which shares the edge that L_p is crossing, as T_c.
4. Map P_0 and L_p onto T_c.
5. If L_p is not crossing T_c, return.
6. Else go back to Step 2.

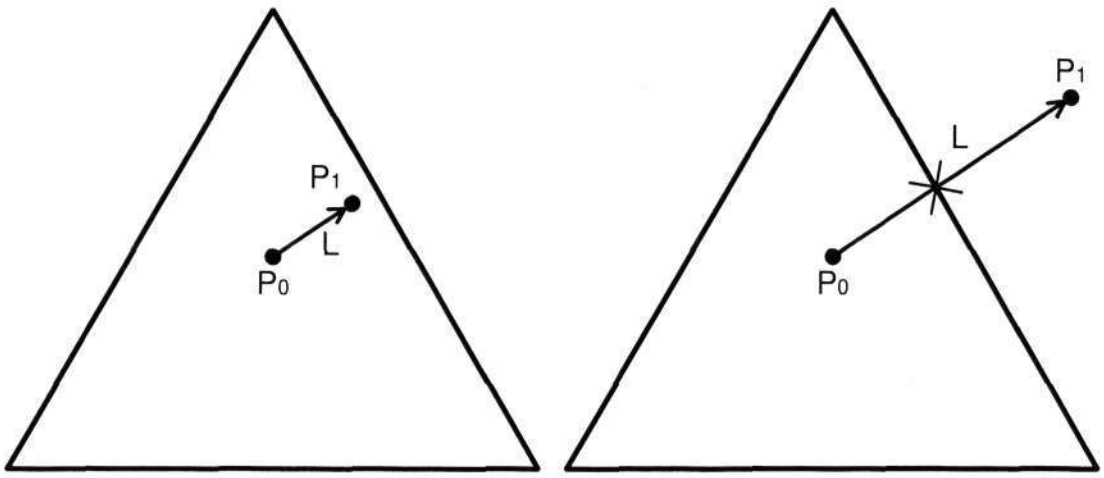
Figure 5.9: Pseudo codes for Cell-in-Triangle test

cell-in-triangle testing and neighbor searching, is introduced. Instead of issuing a test ray for examining each cell, we check the intersection of an edge with the motion path of a cell, which can also be represented as a line segment. The initial position of any cell is always situated inside a triangle. As a result, a cell is found to be moving out from a triangle, whenever there is an intersection between its motion path and an edge. It is also obvious that the neighbor, which shares the intersection edge, is where the cell is moving towards.

Figure 5.9 shows the pseudo codes for the cell-in-triangle test. A motion vector L_p, which displaces a cell P_0 to a new position P_1, is computed by using the resultant repulsive force. The current triangle, which P_0 resides, is called as T_c. L_p is first tested to see if it resides inside T_c (See Figure 5.10(a)). Figure 5.10(b) shows the case where L_p crosses an edge. In this case, further test is needed to check which neighboring triangles the cell is moving to. If the edge, which the cell crosses, is the boundary edge of an open surface, the cell will be stopped from moving and its original position is restored.

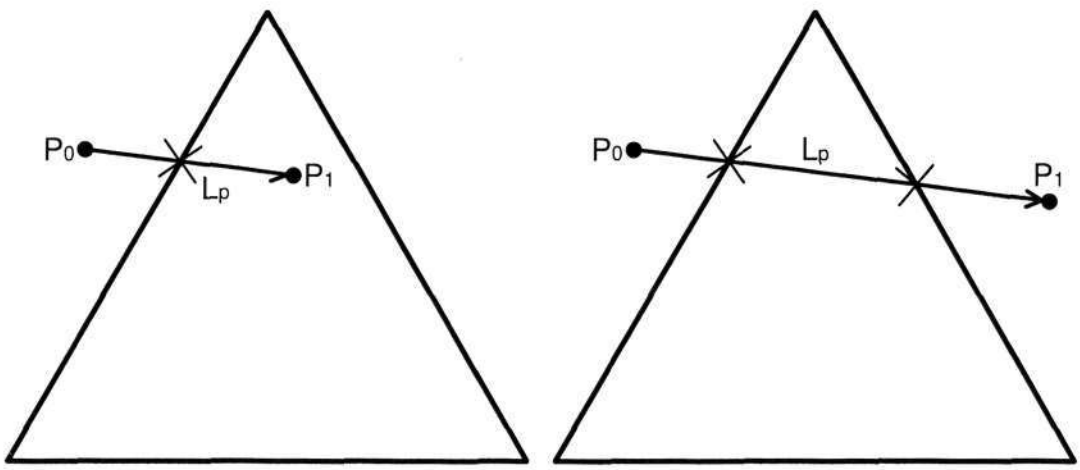
Since we are still working under the *Reference Plane* domain, triangles are disconnected and overlapped with each other. *Neighbor Mapping* is needed, in order to map a cell onto a neighboring triangle. P_0 and L_p are mapped onto the triangle, which shares the crossing edge. This triangle is marked as T_c now. Another test is performed, in order to check if L_p crosses T_c. Figure 5.11(a) shows the case where L_p crosses only one edge and P_0 is moved to P_1, which is inside T_c. Figure 5.11(b) shows another case, where L_p crosses two edges

of T_c. In this case, it means P_0 is moving into another neighboring triangle of T_c. Step 2 (Shown in Figure 5.9) is repeated, until the condition in 5.11(b) is satisfied.



(a) Motion path within triangle (b) Motion path intersects with edge

Figure 5.10: Line cross polygon test



(a) Cell moves into the neighbor triangle (b) Cell moves across the neighbor triangle

Figure 5.11: Line cross polygon test for neighbor triangle

5.3 Discussion

The *Clonal Mosaic* simulation process in our system is discussed in this chapter. Two major modules are presented, the relaxation process and the cell-in-triangle test.

The computationally expensive classical *Global Relaxation*, which relaxes all cells in each relaxation calculation, aims at uniformly arranging the cells in a fixed interval. To improve the computation performance of the process, we proposed two approaches of the *Local Relaxation* for distributing only those interrupted neighborhood within certain

specified spatial conditions during cell division events. The first approach relaxes (i.e. re-aligns) cells within a neighborhood, whenever a cell division event happens. However, this will lead to a severe problem in the situation where rapid cell divisions are performed in the same neighborhood. This proposed approach is pragmatically feasible with relatively low cell division rate.

The second approach solves this problem by grouping all neighborhoods, which are affected by division events occuring in a given time interval. *Local Relaxation* is performed on these neighborhoods in a fixed time interval. This approach highly improves the system performance, as fewer cells are being relaxed in each relaxation event. In the worst-case scenario where cell division occurs in all triangles of the surface, the performance of this approach will still be at least compatible to the performance of a *Global Relaxation* process. This is because the relaxation computation will be applied on all the cells, which is what a typical *Global Relaxation* process does.

The second approach of the *Local Relaxation* is chosen as the replacement of the classical *Global Relaxation* approach. The concept and development of the first proposed approach is aimed at establishing the foundation for the development of the second approach for the *Local Relaxation* of cells.

The cell-in-triangle test introduces a simple and fast method to check the location of a cell after it is displaced by the resultant repulsive force. Through a series of line intersection tests, it identifies the triangle in which a cell resides.

Both the *Local Relaxation* and cell-in-triangle test exploit the merit of the *Canonical Reference Plane Structure*. Whenever computation of cell distances or cell movement in a neighborhood is required, cells are mapped to a local frame by using our *Neighbor Mapping* function. This result in a fast and efficient simulation process, as only two-dimensional information and computation are needed to be considered.

Chapter 6

Rendering Texture Pattern

Texture pattern of *Clonal Mosaic* model is achieved through a regulated spatial arrangement of cells and their color property on a surface. Each of the cells is a progeny of certain type and all cells from same progenitor inherit the same set of color property. This color property is visible on the surface, as spatial area occupied by the cells eventually reveals their cell colors. The set of cells, which is obtained from the *Clonal Mosaic* simulation, can be considered as a discretized version of the original surface. As a result, the problem of rendering the *Clonal Mosaic* pattern is a problem to recover the color information on the original surface, according to this discrete set of cells.

Turk [8] discussed similar problem in his *Reaction-Diffusion* model, where the cells distributed on a surface defining the chemical concentration of color pigments. In his model, the original surface is replaced by a mesh, which takes the distributed cells as its vertices. For rendering of low-resolution texture, the color of a polygon is linearly interpolated among its vertices. For high-resolution texture, the color value of any point on the surface is obtained by using a weighted sum of chemical concentration from nearby vertices (cells).

On the other hand, the problem can also be solved by using a Voronoi diagram. Voronoi diagram has long been used to represent the tessellation of cells in biological research. This idea is also used in Walter's *Clonal Mosaic* model [37]. As mentioned before, it is difficult to compute an actual Voronoi diagram on meshes. Hence, the Voronoi diagram on individual triangles is computed and clipped against triangles' edges. The final result is then approximated by a combination of Voronoi diagrams on individual triangles. Under most of the cases, the errors are invisible (See Section 3.2.4).

In a typical rendering process, the color for each pixels of a two-dimensional image is computed. A pixel is the projection of one or multiple points from the objects in a three-dimensional space. The coordinate of a point on an object is identified through ray-tracing or ray-casting method. The color of this point is computed from the lighting, shading and texture information of the object. For the case of *Clonal Mosaic* pattern, the diffuse color for a surface point is computed. This color value is decided by the cell region, where the point is positioned.

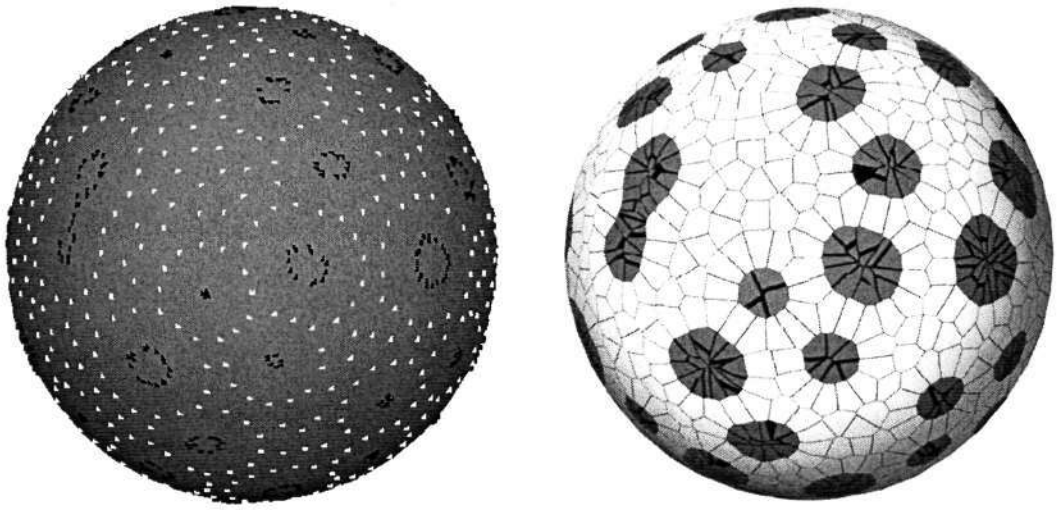
In this chapter, we describe the method to visualize the *Clonal Mosaic* cells as a Voronoi diagram. A closest-cell-searching method, which helps in simplifying the visualization of a Voronoi diagram, is presented.

6.1 Render Clonal Mosaic Pattern as Voronoi Diagram

Given a set of points P in a space, a tessellation of the space can be formed by associating every locations of the space, to a closest member p_i of the point set P. This produces regions that are associated with members of the point set P, which is called Voronoi diagram. Each of the regions is then named as a Voronoi polygon [43]. This general definition for Voronoi diagram applies to our case, where the tessellation of cell region on a mesh surface has to be obtained.

As mentioned in Section 3.2.4, the classical *Clonal Mosaic* model generates Voronoi diagram on each of the triangles. The result is then the combination of Voronoi diagrams from all the triangles. Each *Clonal Mosaic* cell is associated with each of the computed Voronoi polygon. The computation of a Voronoi diagram is a complicated process. The boundaries and vertices of each Voronoi polygon on the given surface is computed and stored. In a rendering process, the color for a render point r on the surface is requested. A heavy point-in-polygon test is needed, in order to decide which Voronoi polygon does r belongs to. Once the association of Voronoi polygon is found, the color of respective *Clonal Mosaic* cell is then returned as the color for r.

We propose an efficient searching method, which identifies the association of a render point r with a *Clonal Mosaic* cell during a rendering process. Computation for the boundaries and vertices for each Voronoi polygons are not needed. The final result produced by the method implicitly returns the Voronoi diagram.



(a) Cell positions after Clonal Mosaic Simulation (b) Voronoi Diagram created by using closest-cell-searching

Figure 6.1: Rendering of Clonal Mosaic texture pattern

In a rendering process, the coordinate of a render point r is computed through ray-casting or ray-tracing. A render point r is first transformed onto the *Reference Plane*. The basic concept of the proposed method is to map all neighboring cells of r onto a neighborhood through the *Neighbor Mapping* function. The distances from r to each of the neighboring cells are compared. A closest cell is selected and r is given the color of the closest cell. This is called as a closest-cell-searching rendering method.

Figure 6.1 shows the result, where the closest-cell-searching method is used to render the *Clonal Mosaic* pattern. Figure 6.1(a) shows the position of the *Clonal Mosaic* cells after the simulation. Figure 6.1(b) shows the render result. In this example, we render those points, which have equal distance to more than one cell, as black. These black points form a set of black lines. These lines clearly identify the boundaries of the Voronoi Polygons, which is associated to each of the *Clonal Mosaic* cells. From the figures, they clearly illustrate that our method successfully presents a valid Voronoi diagram on a polygonal object without incurring heavy computation.

However, the performance of this method can be greatly degraded, if there is a large number of neighboring triangles and cells. In here, the algorithm of the basic concept is presented and optimized for attaining fast computation time.

6.2 Implementation of Closest-Cell-Searching

As mentioned previously, a neighborhood of cells is mapped in order to determine the closest cell to a render point r. A closest neighboring cell C_{min} can be either residing on the triangle T_c, which r is residing, or on any other neighboring triangles of T_c.

There are two major issues, which affect the performance of the proposed approach. First, triangles and cells are mapped onto a neighborhood through the *Neighbor Mapping* function. Multiple mappings are required for a triangle consisting of more than one cell. Second, the number of distance computations and comparisons will affect the rendering time.

In order to solve the first problem, we use a reverse *Neighbor Mapping* technique. Instead of transforming all cells on a triangle onto the neighborhood, we map r onto the neighboring triangle. Hence, only one mapping is needed for each triangle as compared to the original multiple mappings.

For the case of repulsion computation mentioned in Section 3.2.3, repulsive radius is used to define a neighborhood. For the case of rendering, we used an adaptive radius, which reduces the size of neighborhood whenever a closer cell is obtained. This helps in reducing the number of distance computations .

Figure 6.2 presents an efficient algorithm, where the closest cell is obtained with optimal computational cost. First, the render point r is transformed onto the *Reference Plane*. We maintain a list of triangles T, in which a closest cell might be residing. The first triangle being introduced into the list is the triangle T_c, where point r is residing. The secondary and tertiary neighbors of T_c will be added into the list subsequently. The adaptive radius D_{min} is used to define the size of the neighborhood. It is set to be infinite at the beginning.

For each triangles T_i in T, several operations are performed. First, r is mapped onto T_i with a reverse *Neighbor Mapping* process. Next, the distance of r to the vertices and edges of T_i is measured. T_i will be skipped for subsequent operations, if it falls outside the coverage of D_{min}. A closest cell is selected among all the cells residing in T_i by comparing their distances to r. The closest distance, which is smaller than r, is taken as the new adaptive radius D_{min}. This reduces the neighborhood size. Next, all the neighboring triangles of T_i are added into list T. The operation continues until the last triangle in T is processed.

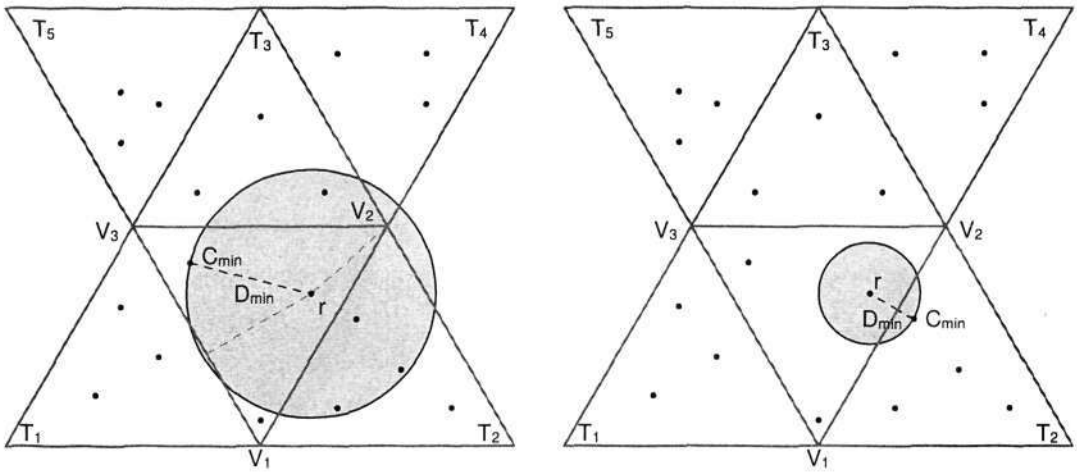
For rendering of a surface point r, the following operations are performed

1. Transform point r onto the *Reference Plane*
2. Add the current triangle T_c, which r resides, into the triangle list T
3. For each triangles T_i in T
   - Perform reverse Neighbor Mapping to transform r onto T_i
   - If the distance from r to any vertices of T_i is larger than D_{min}, continue
   - Else if the distance from r to any edges of T_i is larger than D_{min}, continue
   - Obtain the closest cell C_{min} in T_i, which has a distance value D smaller than D_{min}. D_{min} is updated as D.
   - Add neighboring triangles of T_i into T

Figure 6.2: Pseudo codes for closest-cell-searching

Figure 6.3 gives a clear illustration on the details of the algorithm processes. In Figure 6.3(a), C_{min} is the closest cell obtained from T_c where D_{min} is the current closest distance. The shaded circle defines the region covered by the current adaptive radius. The neighboring triangles, T_1, T_2 and T_3 are added into T. T_1 passes the distance test, as its edge hits the shaded region. However, none of its cells are closer to r as compared with C_{min}. Hence, the adaptive radius is unchanged. T_2 passes the distance test, as its vertex falls inside the shaded region. One of its cells is selected as a new C_{min} and D_{min} is updated. Figure 6.3(b) shows the cell with a smaller adaptive radius after the update. Since T_3 is no longer hitting the shaded region, it is ignored for the subsequent operations.

All the distance comparisons use the squared distance between two points. Calculation of a square root in a typical distance function is avoided and hence reduces the overall computational cost.



(a) Obtain closest cell from current triangle (b) Obtain closest cell from neighboring triangles

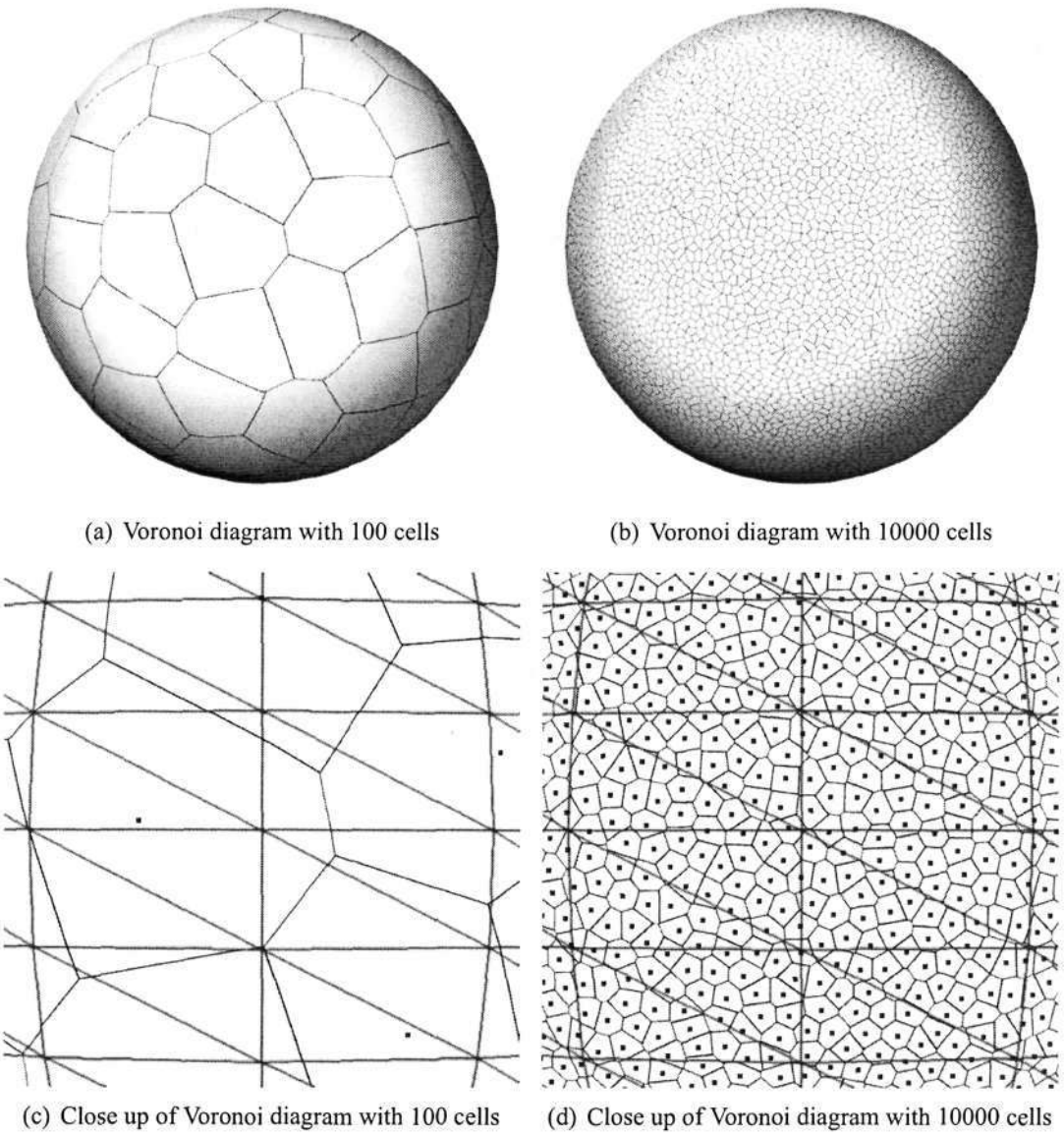
Figure 6.3: Closest-cell-searching from neighboring triangles

6.3 Discussion

In this chapter, we presented an efficient method to render the *Clonal Mosaic* pattern on a given surface. The method avoids the complicated computation of a Voronoi diagram, while giving acceptable result. The proposed method determines the closest cell to a render point by comparing the distances of the point to all cells on a given neighborhood. Since the distances are approximated, errors may be introduced in the solution.

Figure 6.4(a) shows the result of a Voronoi diagram computed from 100 cells, which are populated on a polygonal object comprising 700 triangles. By contrast, the result evaluated from the 10000 cells distributed on the same object is shown in Figure 6.4(b). The close up of their results are given in Figure 6.4(c) and 6.4(d) respectively. The triangles and cell positions are also depicted. For the case of 100 cells, minor discontinuity is observable at the edges of the triangles.

Figure 6.5 explains how the error is introduced into the solution. We examine a render point r located on an edge sharing between triangle T_1 and T_2. Figure 6.5(a) shows the neighborhood of triangles and cells of T_1. Since the original surface is non-planar, cracks occur in the flatten neighborhood. Cell C_{min} is selected as the closest cell to r. D_{min} is the distance between r and C_{min}. D_{min} does not cross over any crack (i.e. non-planarity area) in this neighborhood. In Figure 6.5(b), the neighborhood of T_2 is shown. D_{min} crosses a small crack in this neighborhood. Hence, the value of D_{min} in this neighborhood is slightly



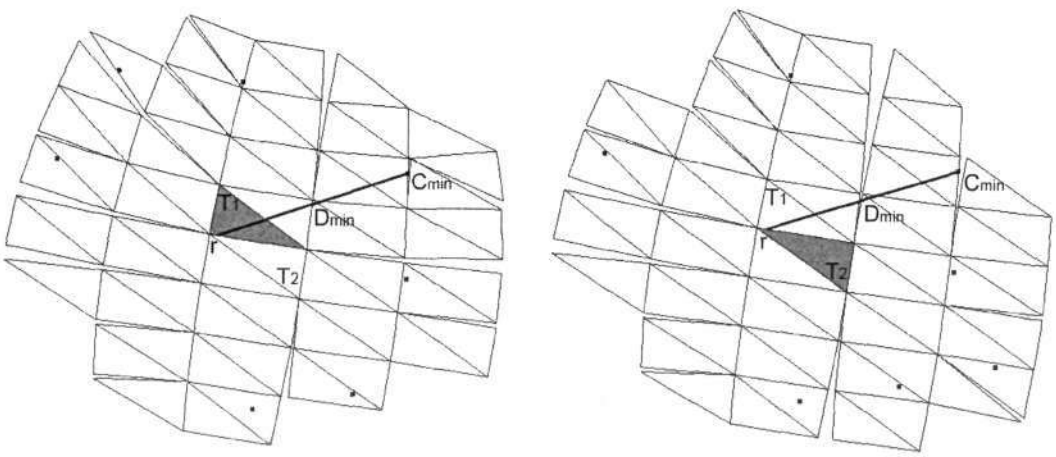
(a) Voronoi diagram with 100 cells

(b) Voronoi diagram with 10000 cells

(c) Close up of Voronoi diagram with 100 cells

(d) Close up of Voronoi diagram with 10000 cells

Figure 6.4: Voronoi diagram showing cells' regions



(a) Distance measured from T_1

(b) Distance measured from T_2

Figure 6.5: Voronoi error introduced from distance approximation

larger than the one measured in the neighborhood of T_2. This inconsistent measurement of distance is responsible for the Voronoi error shown in Figure 6.4.

It is noted that the error will be introduced whenever the distance crosses a crack in a neighborhood. A crack normally occurs at the edge of a secondary or tertiary neighbor. Hence, the error can be avoided, if the closest cell is detected from the master triangle or primary neighbors. The object, which has a large number of triangles and is populated with a small number of cells, is likely to suffer from the Voronoi error. This is because a closest cell can only be determined from a secondary or tertiary neighbor cell. In most of the cases, a large number of cells are populated on a surface. Hence, a Voronoi error has a small chance to occur. Besides, a *Clonal Mosaic* pattern element (e.g. a black spot) normally comprises of dozens of cells. Hence, the occurrences and effects of the Voronoi error will be kept to the minimum. Even if a Voronoi error does occur inside pattern elements, it will not be noticeable as these pattern elements of a *Clonal Mosaic* model generally comprise of dozens of cells. The only situation where the error is visible is when the Voronoi error occurs at the outer cell of a pattern element, as discontinuity appears on the boundary of the pattern element will be noticeable.

In conclusion, an efficient method is introduced to render the *Clonal Mosaic* pattern. It capitalizes the merit of the *Canonical Reference Plan Structure* presented in Chapter 4. Cells are rendered as Voronoi polygons without undergoing the heavy computations involved in the classical process of generating a Voronoi diagram.

Chapter 7

Experimental Results

Several experiments were conducted for examining the predicted performance of the proposed methods presented in the preceding chapters. The experimental results are presented and analyzed in moderate detail.

First, the performance of the previously mentioned classical *Clonal Mosaic* m odel and our proposed optimized models are compared. The proposed models include the two-dimensional *Canonical Reference Plane Structure* for optimizing the transformation operations and the *Local Relaxation* technique for simulation optimization. The algorithms are tested with different configurations and controlled parameters. The simulation time and synthesized texture patterns obtained from our models are presented and compared with the existing ones.

Next, the performance evaluation of our proposed texture rendering module is conducted with different parameter settings. The results are compared with some of the existing texturing methods.

Lastly, several types of the real world animal skin patterns are shown. Our optimized *Clonal Mosaic* model is used to generate comparable animal skin texture patterns. Having demonstrated the synthesized patterns on a sphere geometry, the practicality and generality of the proposed models are then verified with polygonal objects in different shapes.

7.1 Comparison of Classical and Optimized Clonal Mosaic Models

As mentioned in previous chapters, the classical *Clonal Mosaic* model uses the three-dimensional transformation technique to re-arrange cells onto a common plane. The repulsion process is then performed using the information of the distance between any two cells. We propose the *Canonical Reference Plane* method for transforming all cells onto a common reference plane at the pre-processing stage. A simple operation is then performed to measure the two-dimensional Euclidean distance between any two cells during repulsion computation. Besides, the *Global Relaxation* technique of the classical model involves redundant computation. Hence, the *Local Relaxation* method is proposed to minimize the computational load.

In this section, experiments are designated for evaluating the performance of the four different simulation models, namely, the classical *Clonal Mosaic Model* (ORG), the model optimized with *Canonical Reference Plane Structure* (CRPS), the model optimized with *Local Relaxation* (LR) and lastly the model optimized with both *Canonical Reference Plane Structure* and *Local Relaxation* (CRPS-LR). The abbreviations in the brackets are used for indicating different simulation models in the subsequent sections.

The key parameters significantly impacting the simulation speed are the number of cells, number of triangles, repulsive weight and number of simulation steps. Experiments were conducted on the simulation models by appropriately varying the range of these parameters. The results obtained are analyzed and discussed.

The initial setup for the experiments contained two types of cells, 10 percent of black cells and 90 percent of white cells. The black cells have a division rate of 2.0, while white cells are designated not to be divided. Values of 0.1, 0.5 and 1.0 were chosen for the repulsiveness among the black cells, among the white cells and between the white and black cells, respectively. Having designated a relatively low repulsive value among the black cells, larger black patches would probably be formed by clustering of these cells. On the other hand, smaller boundary regions would likely be generated by the white cells with relatively higher repulsiveness. A highest repulsive value between the white and black cells is aimed at generating the distinctive texture pattern on the skin. In this settings and configuration, a cheetah skin pattern is expectedly formulated and synthesized.

7.1.1 Performance with Different Number of Cells

No. of Triangles	Repulsive Weight	Simulation Step
760	2.0	50

Table 7.1: Key parameter settings for the experiment with different number of cells

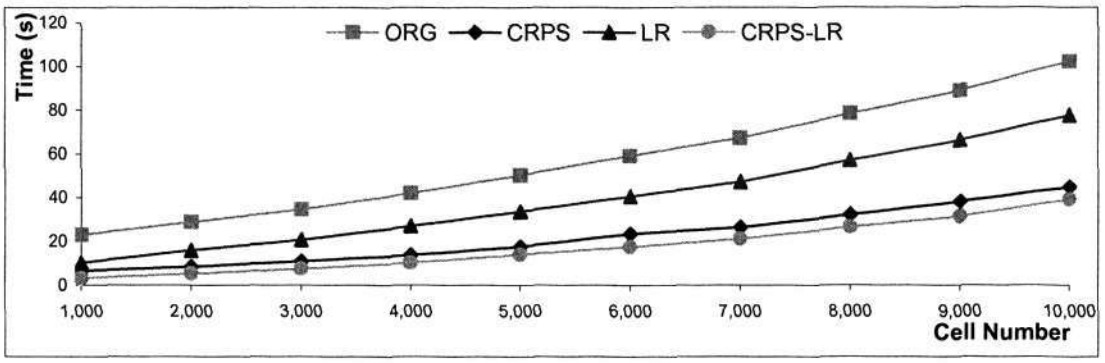


Figure 7.1: Performance of Clonal Mosaic simulation obtained using different simulation models with different number of cells

Cell No	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
ORG	22.88	28.80	34.44	41.99	50.13	59.27	67.30	78.39	88.85	102.18
CRPS	6.56	8.35	10.94	13.80	17.61	23.46	26.45	32.22	38.17	44.81
LR	10.15	15.88	20.66	27.03	33.69	40.65	47.35	57.05	66.31	77.66
CRPS-LR	3.17	5.23	7.36	10.19	13.95	17.50	21.14	26.55	31.37	39.18

Table 7.2: Clonal Mosaic simulation time(s) obtained using different simulation models with different number of cells

In this experiment, the computational performance of the simulation models will be studied and compared with the increments of the total number of cells. The key parameter settings are listed in Table 7.1.

Figure 7.1 shows the performance graph of the test models and Table 7.2 shows the corresponding data values. LR successfully reduced the simulation time by half, with relatively small number of cells. However, as more cells were introduced into the object model, extensive computation was required to be devoted in cell transformation.In this case, the improvement of the LR method was not as significant as before. As, the massive transformation computation plays the key role in the overall system performance. On the other hand, the simulation time of CRPS method was almost constantly improved by half even with the large increments of cells. This is because that the CRPS reduces the complexity of transformation computation.

Figure 7.2 shows the cell formulation under different simulation models with different number of cells. The percentage of black cells is fixed for all of the test models. When

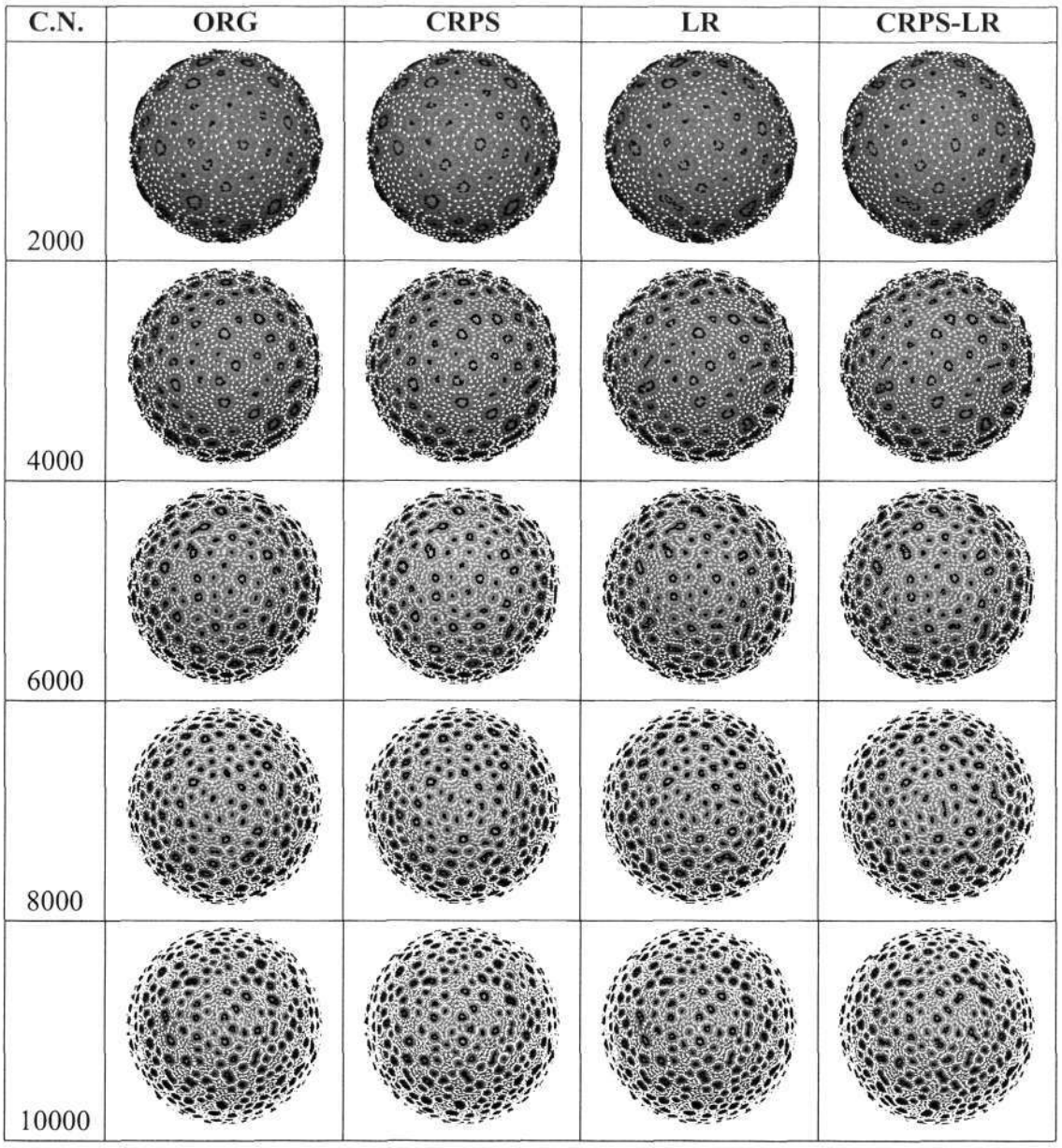


Figure 7.2: Texture patterns generated using different simulation models with different number of cells

the total number of cells increased, it is obvious that the number of black cells (with fix percentage) is increased. The black spots, which are formulated by the black cells, are hence increased in their quantity. Since the surface area is also fixed, when the number of cells is increased, the unit space occupied by each of the cell will be reduced. This results in forming smaller spots. From this observation, we arrive at the conclusion that the overall system performance is not materially affected by this phenomenon. Similarly, the same phenomenon was also observed in simulation process of the other three smulation models. The positions of the cells were almost identical for all the test cases using these different simulation models. Interestingly, there were a few artifacts (i.e. non-rounded shapes) introduced into the synthesized results obtained from processing 8000 cells with LR

simulation method. The replacement of a few rounded shape cells with V-shape ones was due to the pre-defined condition imposing that only neighboring cells which were situated within the repulsive circle would be considered for relaxation.

7.1.2 Performance with Different Number of Triangles

No. of Cells	Repulsive Weight	Simulation Step
2000	2.0	50

Table 7.3: Key parameter settings for the experiment with different number of triangles

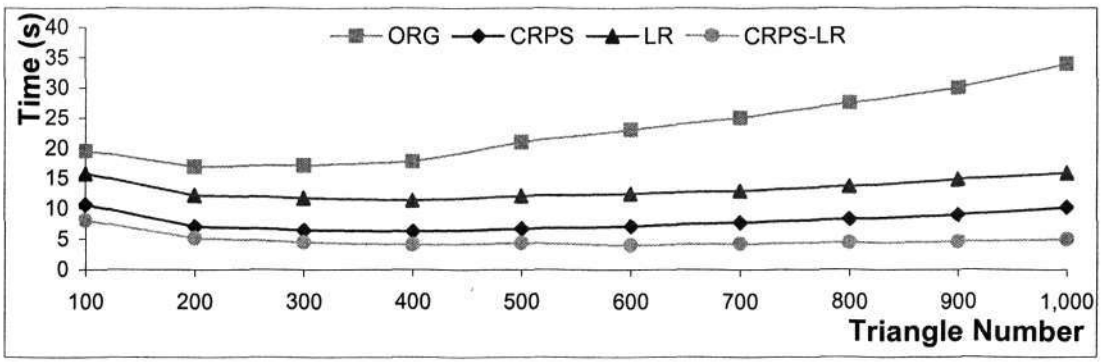


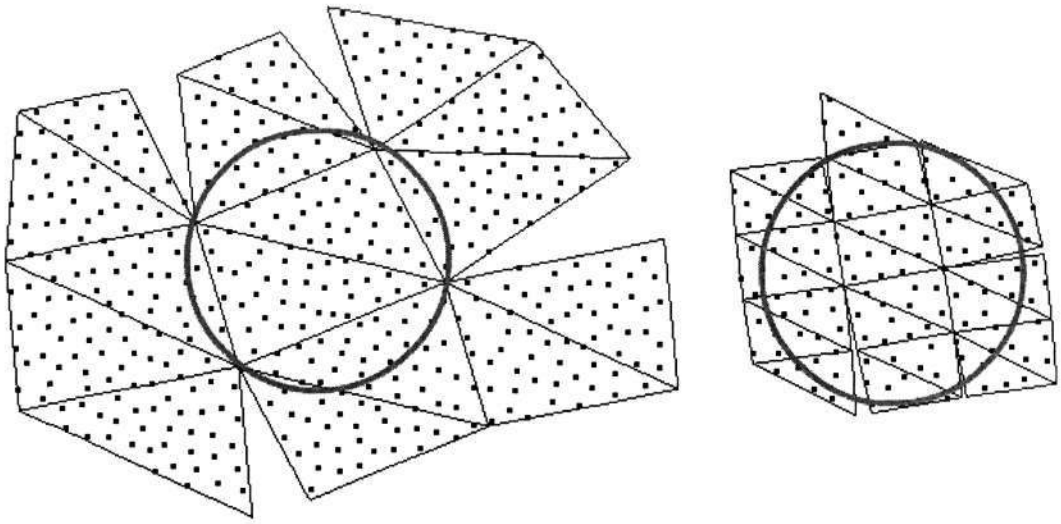
Figure 7.3: Performance of Clonal Mosaic simulation obtained using different simulation models with different number of triangles

Tri. No	100	200	300	400	500	600	700	800	900	1000
ORG	19.50	16.93	17.09	17.70	20.94	22.99	24.96	27.44	29.94	33.8961
CRPS	10.60	7.05	6.41	6.21	6.67	7.07	7.69	8.28	8.95	10.21
LR	15.72	12.22	11.73	11.32	12.12	12.44	12.90	13.68	14.82	15.89
CRPS-LR	8.02	5.09	4.45	4.02	4.32	4.00	4.23	4.38	4.54	4.9509

Table 7.4: Clonal Mosaic simulation time(s) obtained using different simulation models with different number of triangles

In this experiment, the computational performances of the simulation models are compared with the increments of the total number of triangles. The experiment settings are listed in Table 7.3.

Figure 7.3 and Table 7.4 show the performance graph and its detailed numerical values of the experiment, respectively. It shows that the simulation time of the classical *Clonal Mosaic* (i.e. ORG) increases with larger number of triangles in most cases. In the case of our proposed models, the simulation times are faster than ORG and they remain almost constant even with increasing number of triangles. Given a specified object with a fixed number of cells, the repulsive radius is a constant value. Hence, there will be more triangles intersecting the repulsive circle for an object with larger tessellation. Figure 7.4 shows



(a) Number of triangles = 100 (b) Number of triangles = 500

Figure 7.4: Results of the Neighbor Mapping with different number of triangles

samples of neighboring triangles from different tessellations intersecting a repulsive circle with a constant radius. In the case of less tessellated object (e.g. 100 triangles), although a small number of triangles were selected as neighboring triangles, a relatively large number of cells to be later transformed were covered by these triangles (See Figure 7.4(a)). On the other hand, a large number of triangles were determined as neighboring triangles in the case of high tessellated object (e.g. 500 triangles), a smaller number of cells in fact were covered by the triangles (See Figure 7.4(b)). This explains why slower simulation times were observed in applying all the simulation models to test objects with small number of triangles.

As compared to ORG model, all the three proposed optimized models have significant improvement in simulation time. LR improved the system performance by dealing with fewer cells. Since the mapping process incurred a significant portion of the overall computational efforts, CRPS still performed better than LR as its mapping process, which makes use of the *Canonical Reference Plane Structure*, was optimally effective. By applying CRPS-LR, the system performance was further reduced. The simulation time was nearly constant as observed from the graph.

Figure 7.5 shows the cell formulation generated using the simulation models with different number of triangles. The synthesized patterns has no major distinctive differences. Similar to the experiment presented in Section 7.1.1, there were some minor cell artifacts (i.e. non-rounded shapes) generated with the LR approach.

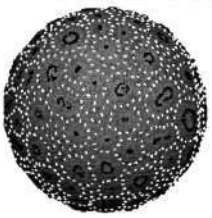
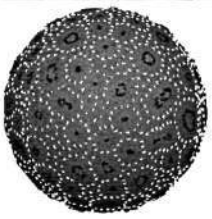
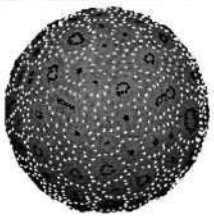
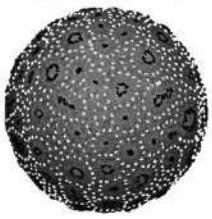
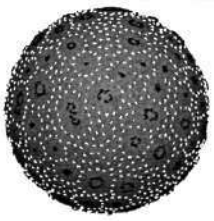
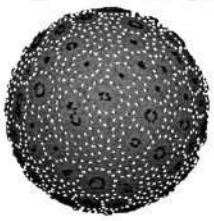
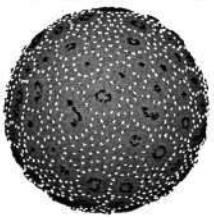
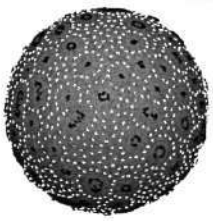
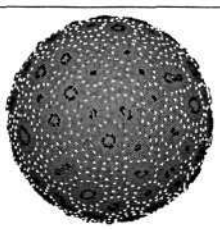
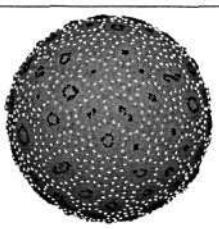
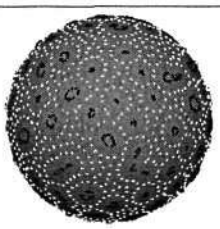
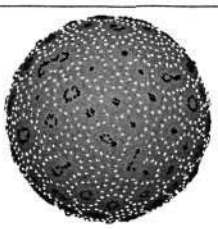
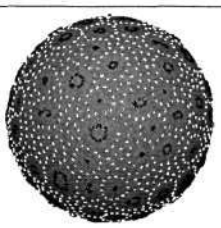
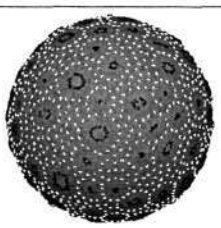
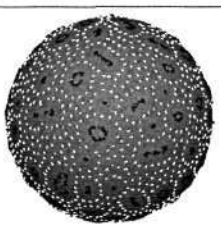
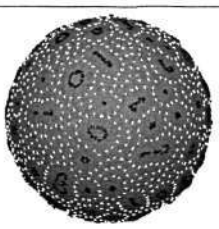
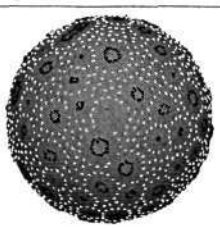
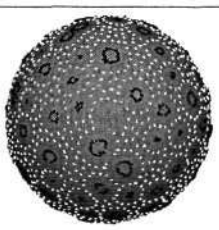
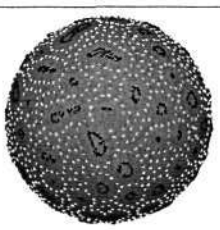
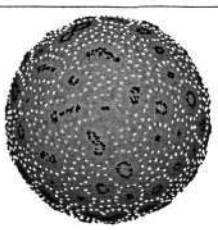
T.N.	ORG	CRPS	LR	CRPS-LR
200				
400				
600				
800				
1000				

Figure 7.5: Texture patterns generated using different simulation models with different number of triangles

7.1.3 Performance with Different Repulsive Weights

No. of Cells	No. of Triangles	Simulation Step
2000	760	50

Table 7.5: Key parameter settings for the experiment with different repulsive weights

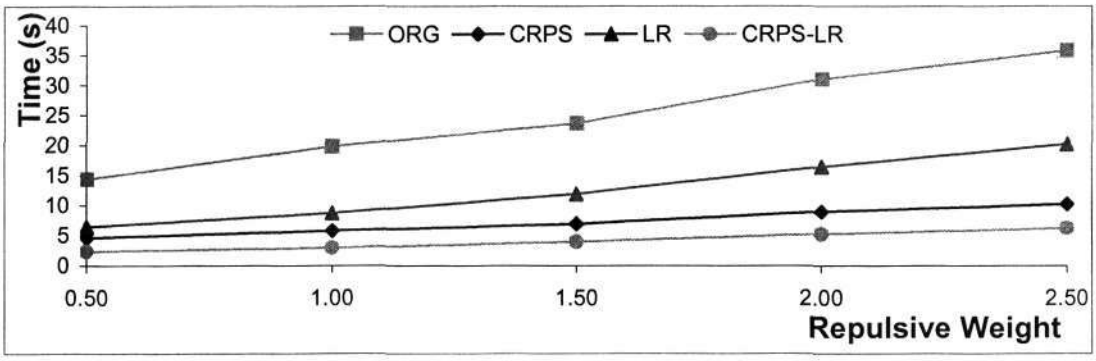


Figure 7.6: Performance of Clonal Mosaic simulation obtained using different simulation models with different repulsive weights

R.W.	0.5	1.0	1.5	2.0	2.5
ORG	14.36	19.79	23.64	31.00	35.83
CRPS	4.64	5.81	6.92	8.93	10.20
LR	6.42	8.70	11.92	16.41	20.17
CRPS-LR	2.34	2.96	3.94	5.24	6.20

Table 7.6: Clonal Mosaic simulation time(s) obtained using different simulation models with different repulsive weights

In this experiment, the computational performances of the simulation models are compared with the increments of the repulsive weight. The experiment settings are listed in Table 7.5.

Figure 7.6 shows the performance graph of the simulation models and Table 7.6 shows the corresponding data values. The repulsive weight affects the size of the repulsive radius. As the repulsive radius increased, more neighboring triangles and cells were selected for repulsion process. As compared to ORG model, a much shorter simulation time of LR model was achieved by ignoring those triangles which did not require cell relaxation. CRPS completed the synthesis process in less than half of the simulation time required for ORG model. By applying the CRPS-LR model, we could synthesize the target pattern in less than one quarter of the simulation time incurred in the ORG model.

The cell formulations obtained using the various simulation models with different repulsive weights are shown in Figure 7.7. When the repulsive weight was small, cell repulsion

was only performed on a very small number of cells. Hence, it was both difficult to form the black cell clusters and also to keep the black and white cells being separated. On the other hand, more noticeable pattern clusters were formed with larger repulsive weights. For example, larger spot patterns were clustered with repulsive weight value greater than 1.5.

Similar results were obtained with different repulsive weight values. Similarly, a few subtle different patterns (i.e. non-circular shapes) were noticeably evolved.

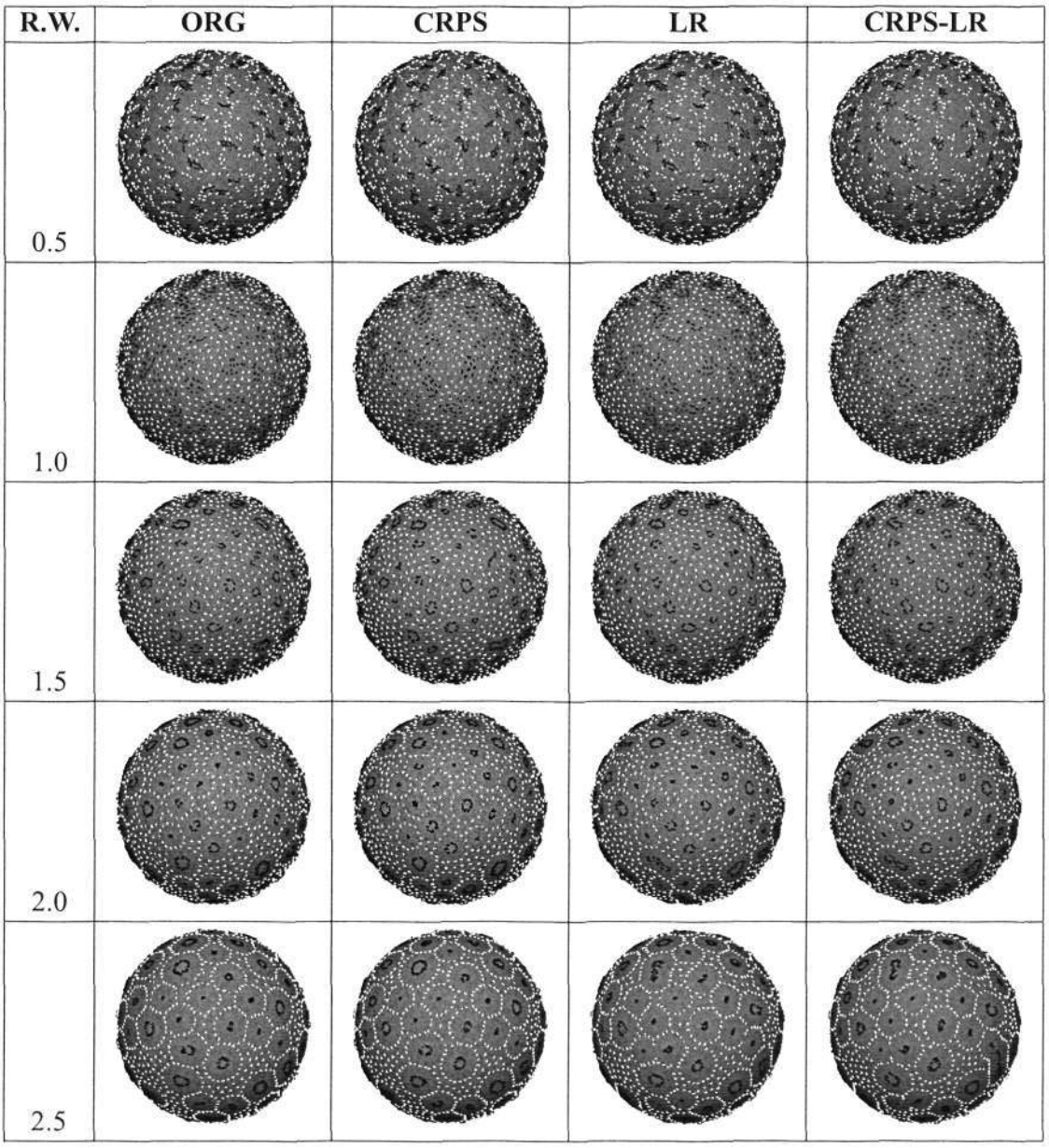


Figure 7.7: Texture patterns generated using different simulation models with different repulsive weights

7.1.4 Performance with Different Number of Simulation Steps

No. of Cells	No. of Triangles	Repulsive Weight
2000	760	2.0

Table 7.7: Key parameter settings for the experiment with different simulation steps

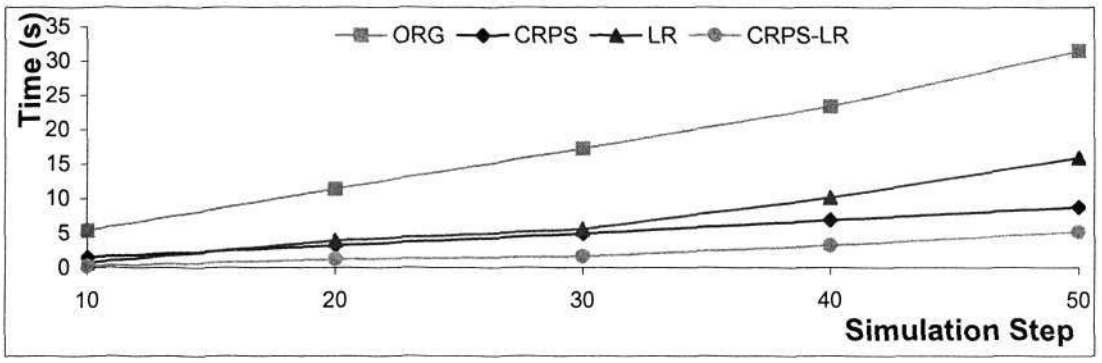


Figure 7.8: Performance of Clonal Mosaic simulation obtained using different simulation models with different simulation steps

Step	10	20	30	40	50
ORG	5.32	11.39	17.25	23.36	31.48
CRPS	1.49	3.19	4.91	6.81	8.69
LR	0.66	3.92	5.52	10.10	15.88
CRPS-LR	0.20	1.22	1.59	3.17	5.17

Table 7.8: Clonal Mosaic simulation time(s) obtained using different simulation models with different simulation steps

By increasing the number of simulation step in this experiment, the statistical computational results of the various simulation models is reported. The settings of the experiment are tabulated in Table 7.7.

Figure 7.8 shows the performance graph of the test models and Table 7.8 shows the corresponding data values. With less number of iterations, all the simulation models finished their pattern synthesis at a very fast pace as only a small number of cells were being divided. Conversely, longer simulation time was expected with larger number of iterations as a greater number of cells were being divided. By applying CRPS-LR model, the pattern synthesis was realized at a relatively short simulation time. It was generally kept within one fifth of the simulation time incurred in the ORG model.

As LR model generally considers only a smaller mapping region, it will require longer iterations before generating any noticeable pattern clusters. This phenomenon was convincingly realized in applying the LR and CRPS-LR simulation models with low iterations (i.e.

10 iterations) shown in Figure 7.9. As the iteration number increased, all the simulation models started to form distinctive clusters.

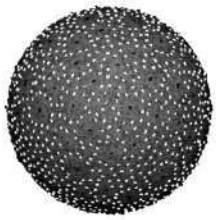
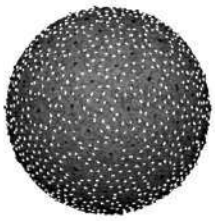
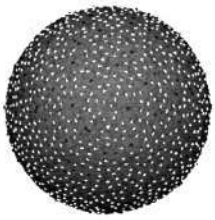
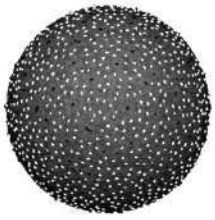
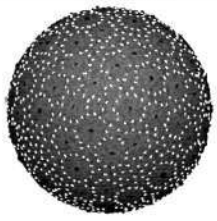
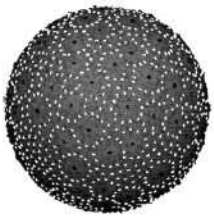
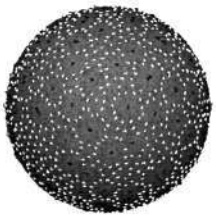
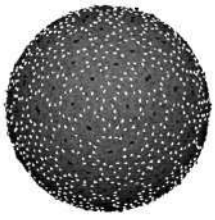
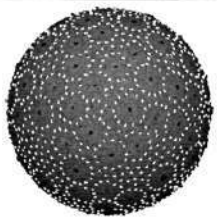
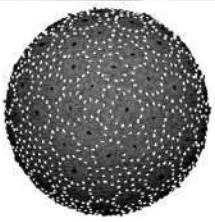
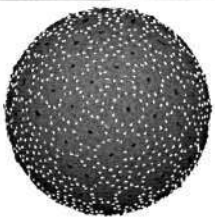
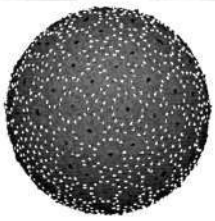
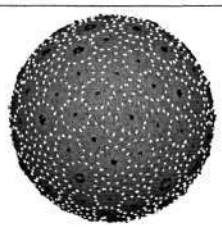
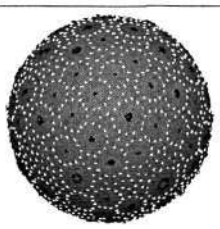
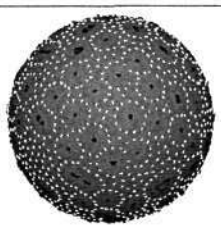
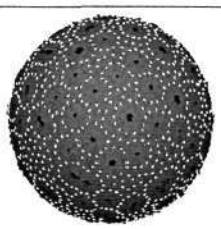
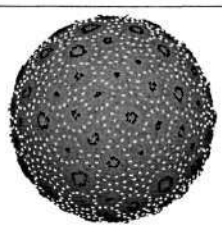
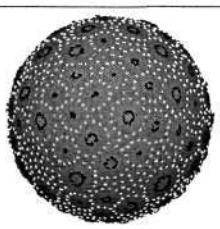
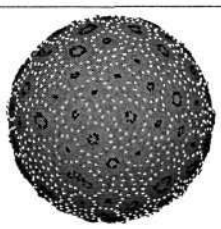
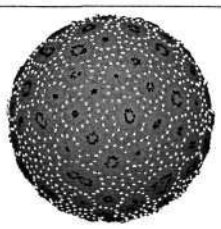
Step	ORG	CRPS	LR	CRPS-LR
10				
20				
30				
40				
50				

Figure 7.9: Texture patterns generated using different simulation models with different simulation steps

7.2 Performance of Texture Rendering

First, the rendering speed of our modified *Clonal Mosaic* model is compared with some representative texturing methods on different image resolutions. Besides the comparison, we tested our proposed rendering model on a spherical object with different number of cells.

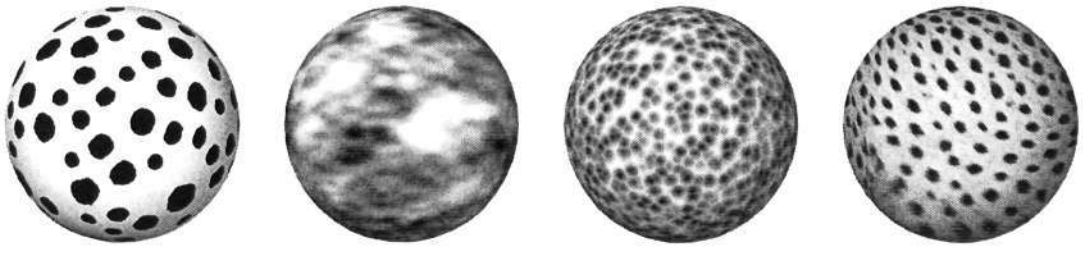
7.2.1 Performance of Rendering with Different Resolutions

A *Clonal Mosaic* model is classified as a procedural texturing model for generating animal skin texture. In this section, the rendering speed of our modified *Clonal Mosaic* model is compared with a standard texturing mapping method and two notorious procedural techniques based on *Perlin Noise* [30] and *Worley Noise* [46].

A texture mapping method uses a two-dimensional image. To determine the color of the surface point, the point will have to be first transformed into a point in the UV coordinate system. The value of the texture map associated to this UV coordinate can be appropriately processed and used as the color of the surface point.

A *Perlin Noise* computes the color of an object point during a rendering by using mathematical functions. Similar to a classical texture mapping technique, in Worley's approach, a UV point is first determined by transforming a surface point. Once done, the nearest cells from the projected UV point are searched from its eight neighboring grids.

Figure 7.10 shows the rendering results of the above mentioned methods. Figure 7.11 shows the performance graph of each rendering method and Table 7.9 shows the corresponding numerical result.



(a) Clonal Mosaic texture (b) Perlin Noise (c) Worley Noise (d) Texture mapping

Figure 7.10: Rendering result obtained using various texturing methods

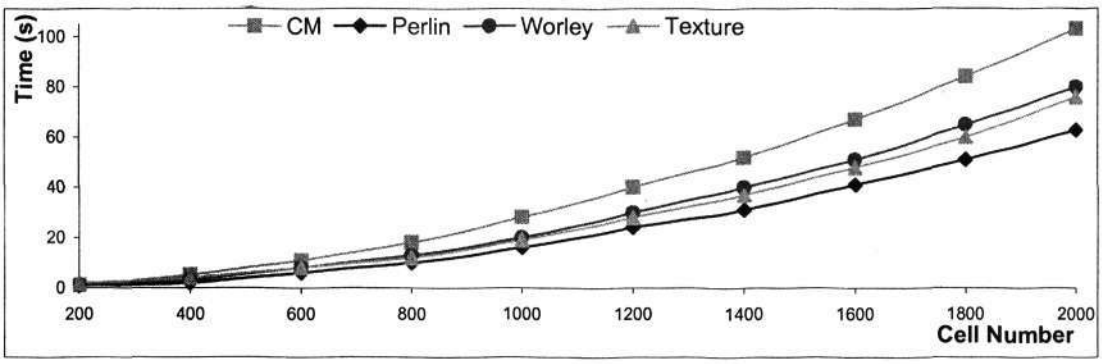


Figure 7.11: Rendering performance of various texturing methods obtained with different render resolutions

Resolution	Clonal Mosaic	Perlin Noise	Worley Noise	Texture Mapping
200 × 200	1	1	1	2
400 × 400	5	2	3	4
600 × 600	11	6	8	8
800 × 800	18	10	13	12
1000 × 1000	28	16	20	19
1200 × 1200	40	24	30	28
1400 × 1400	52	31	40	37
1600 × 1600	67	41	51	48
1800 × 1800	84	51	65	60
2000 × 2000	103	63	80	76

Table 7.9: Rendering time(s) of various texturing methods with different render resolutions

As one can observe from the graph, *Perlin Noise* achieved the fastest rendering time, as the mathematical functions are simple and efficient. Although, the concept of the texture mapping method is straightforward, the computation is still slower than the *Perlin Noise*. Both the *Clonal Mosaic* model and *Worley Noise* rely on the closest-cell-searching to decide the color value. The rendering method based on *Worley Noise* was still relatively faster. This is because the number of neighbors required to be searched for the closest cells in our modified *Clonal Mosaic* was generally greater than the Worley's case (i.e. fixed eight neighboring grids). Furthermore, a *Neighbor Mapping* process was required in our case before the searching of the closest cells could be conducted. This certainly incurred extra computational effort.

One of the possible improvements is to skip the *Neighbor Mapping* process completely by conducting the search of the closest cell in a three-dimensional Euclidean space. However, the result might prone to erroneous as the three-dimensional Euclidean distance between two cells may be much shorter than their distance measured by traversing on the surfaces. Another plausible solution is to generate a texture map using our modified *Clonal*

Mosaic model. Once done, the color of each surface point can then be easily retrieved from its corresponding UV coordinate.

7.2.2 Performance of Rendering with Different Number of Cells

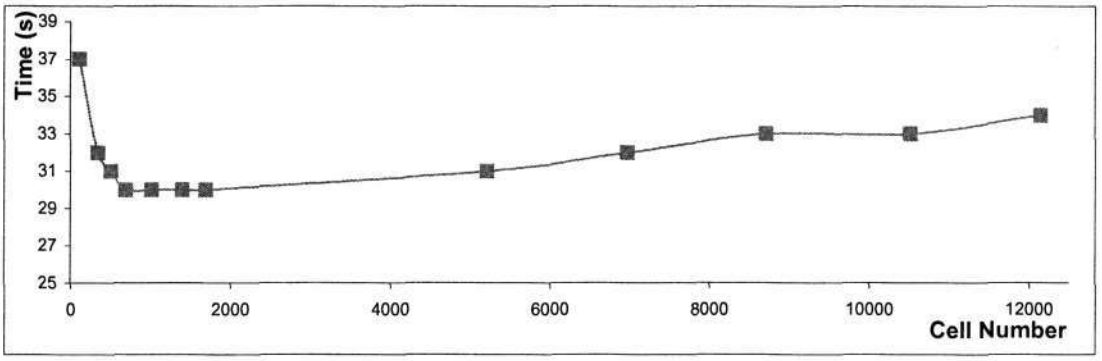


Figure 7.12: Rendering performance with different number of cells

Cell No.	111	341	503	693	1013	1399	1692	5210	6971	8712	10515	12148
Time(s)	37	32	31	30	30	30	30	31	32	33	33	34

Table 7.10: Rendering time(s) of Clonal Mosaic model with different number of cells

In this experiment, we shall examine the rendering time of our model by increasing the number of cells. These cells were generated on the tessellated surfaces of the object. In this test case, there are 760 tessellated triangles.

During the rendering process, the color of each rendering point was obtained from its closest cell. In fact, the overall rendering speed was greatly influenced by the computation time incurred in all the operations leading to finding of the closest cells. These included the *Neighbor Mapping* process and the measurement of the distance.

As one can observe from the experimental results shown in Figure 7.12 and Table 7.10, the object was rendered at a longer time with less cell numbers (e.g. 37 seconds for 111 cells). By given a fixed surface area, cells are sparsely distributed when there are a small number of cells. The closest cell from the rendering point in consideration took relatively longer time to be detected. This is because a series of computationally expensive *Neighbor Mapping* operations are involved in the search of the closest cell. As the number of cell started increasing, the rendering time was gradually reduced as the closest cell of a rendering point could probably be determined within the triangle resulting in the avoidance of *Neighbor Mapping* process.

7.3 Comparison with Real Animal Skin Patterns

Variables	Description
I	Number of simulation Iteration
W_r	Repulsive Weight
P_F	Probability of foreground cells
P_B	Probability of background cells
P_I	Probability of intermediate cells
D_F	Division rate of foreground cells
D_B	Division rate of background cells
D_I	Division rate of intermediate cells
R_{FF}	Repulsiveness between foreground cells
R_{BB}	Repulsiveness between background cells
R_{II}	Repulsiveness between intermediate cells
R_{FB}	Repulsiveness between foreground cells and background cells
R_{FI}	Repulsiveness between foreground cells and intermediate cells
R_{BI}	Repulsiveness between background cells and intermediate cells
C_F	Number of foreground cells after simulation
C_B	Number of background cells after simulation
C_I	Number of intermediate cells after simulation
T	Total time spent for simulation

Table 7.11: Description of key parameters

In this section, the texture patterns of several animal skins are synthesized using our modified *Clonal Mosaic* model. These synthesized results are then compared with the image of the texture patterns gleaned from the real animal skins. To generate the list of target skins realistically, the lists of the values of controlled parameters are required to be determined and applied. Representative controlled parameters are listed in Table 7.11.

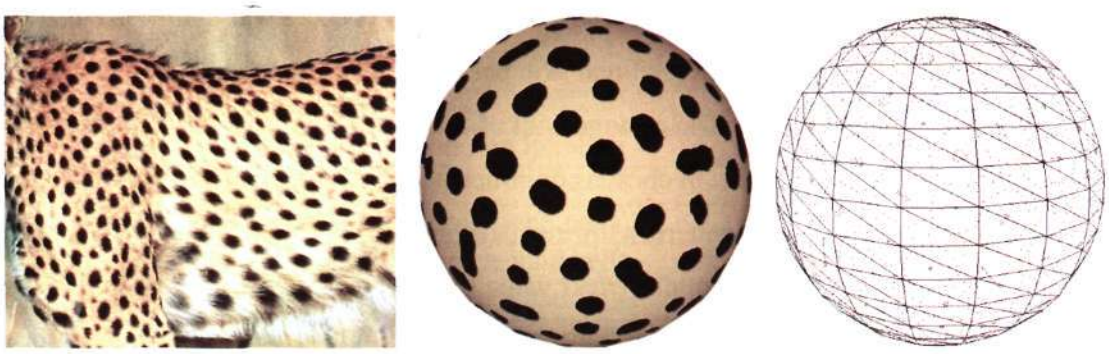
There were either two or three types of cells being used for the synthesis of skin patterns. These included foreground, background and intermediate cell types. During the initial phase, these cells were randomly distributed over a surface according to their preset percentages of the populations to be generated. Once done, division rates for controlling the frequency of splitting cells were defined for different cell types. In the synthesis test, the division rate for the background cell was always set to zero for providing a suitable formation of the boundaries of the target pattern features. An appropriate repulsiveness value was chosen for controlling the stickiness between two types of cell. The initial number of cells for each test was set to 5000. For each of the test cases, we would include the illustration of a real skin pattern, the synthesized skin pattern and its corresponding arrangement of cells. Next, we shall report our first synthesized result, a skin pattern of cheetah.

Cheetah In this experiment, the skin pattern of cheetah was synthesized. Figure 7.13(a) and 7.13(b) show a sample of real skin pattern and synthesized version of a cheetah texture pattern, respectively. The arrangement of the cells for the synthesized pattern is shown in Figure 7.13(c). The list of chosen controlled parameters is shown in Figure 7.13(d). In order to realize the desired pattern, a low repulsiveness value (i.e. R_{FF}) between foreground cells, and a relatively high repulsiveness value (i.e. R_{FB}) between foreground and background cells were employed. The repulsive weight (i.e. W_r) was optimally chosen for controlling the size of the spot features.

King Cheetah The real sample, the synthesized results and list of selected controlled parameter values of the skin patern of king cheetah are shown in Figure 7.14. In order to obtain this targeted pattern, both the repulsiveness between foreground cells and repulsiveness between background cells were set to the same value (i.e. 0.5). This was to avoid the higher repulsion forces from the background cells to push the foreground cells together and forming spot patterns. It is also important to set the repulsiveness between background and foreground cells to zero for avoiding the mixing of these two types of cells.

Leopard Figure 7.15 shows the relevant illustrations of the synthesized results of the skin pattern of a leopard. Compared with the previous two synthesis tests, three cell types instead of two were used for generating the leopard skin pattern. A mutation rate of 0.5 was chosen for the foreground cells. Hence, these cells would have some likelihood to evolve themselves to intermediate cell type. This created the possibility of generating a color feature within the black spot.

Jaguar Figure 7.16 shows a pattern of jaguar. The approach of deriving the controlled parameters for synthesizing the jaguar skin pattern is similar to the case used in leopard skin pattern synthesis described in the previous test. However, the intermediate cell was given a mutation rate of 0.1 to increase its chances for evolving itself to a foreground cell. This arrangement allowed the black spot to be created inside the ring feature.



(a) Real (b) Simulation (c) Cells arrangement

I	W_r	P_F	P_B	D_F	R_{FF}	R_{BB}	R_{FB}	C_F	C_B	T
20	2.5	0.2	0.8	0.25	0.1	0.5	1.0	2605	4000	7.656

(d) List of appropriate controlled parameters

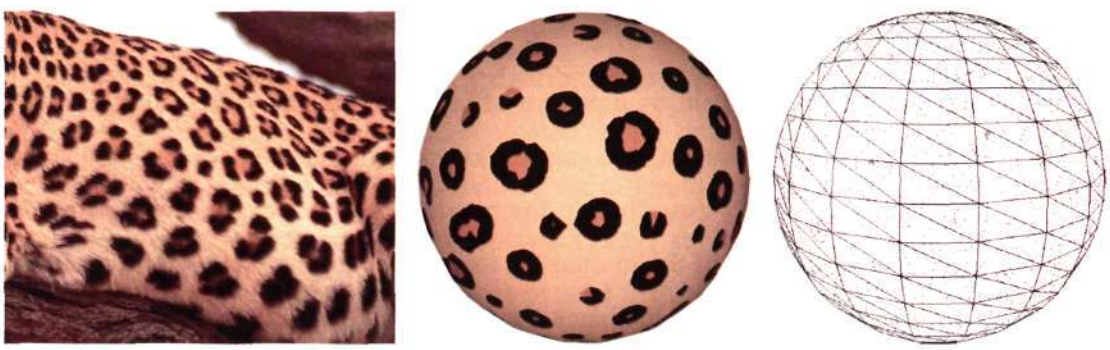
Figure 7.13: Cheetah

(a) Real (b) Simulation (c) Cells arrangement

I	W_r	P_F	P_B	D_F	R_{FF}	R_{BB}	R_{FB}	C_F	C_B	T
20	2.0	0.3	0.7	0.25	0.5	0.5	1.0	3932	3600	6.828

(d) List of appropriate controlled parameters

Figure 7.14: King Cheetah

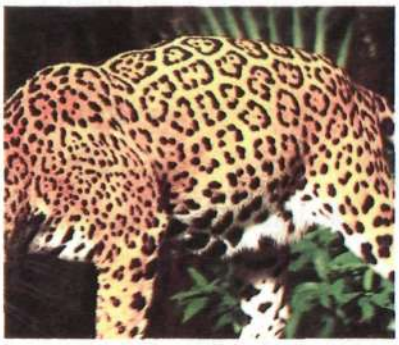


(a) Real (b) Simulation (c) Cells arrangement

I	W_r	P_F	P_B	P_I	D_F	D_I	R_{FF}	R_{BB}	R_{II}	R_{FB}	R_{FI}	R_{BI}	C_F	C_I	C_B	T
30	3.3	0.15	0.85	0	0.15	0.3	0.1	0.5	0.1	0.7	0.1	1.0	1959	4250	2265	19

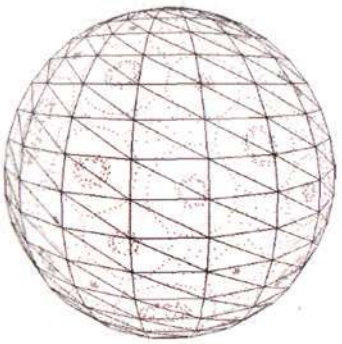
(d) List of appropriate controlled parameters

Figure 7.15: Leopard



(a) Real

(b) Simulation



(c) Cells arrangement

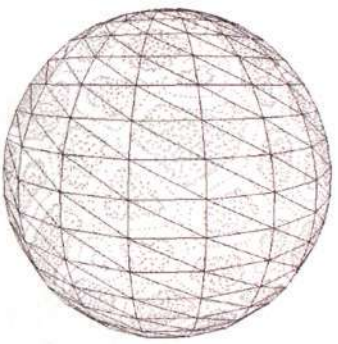
I	W_r	P_F	P_B	P_I	D_F	D_I	R_{FF}	R_{BB}	R_{II}	R_{FB}	R_{FI}	R_{BI}	C_F	C_I	C_B	T
35	4.1	0.1	0.9	0	0.1	0.5	0.1	0.5	0.1	0.8	0.1	1.0	1784	4473	3851	19.19

(d) List of appropriate controlled parameters

Figure 7.16: Jaguar

(a) Real

(b) Simulation

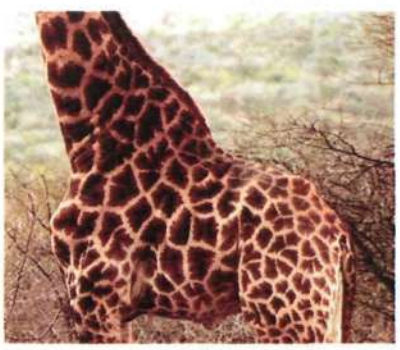


(c) Cells arrangement

I	W_r	P_F	P_B	D_F	R_{FF}	R_{BB}	R_{FB}	C_F	C_B	T
45	3.2	0.2	0.8	0.2	0.2	0.5	1.0	21261	4000	51.953

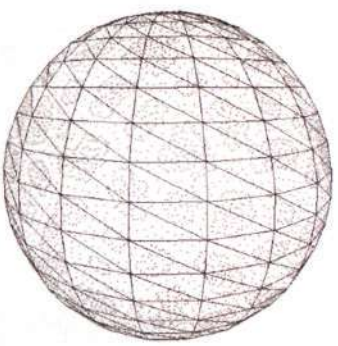
(d) List of appropriate controlled parameters

Figure 7.17: Giraffe 1



(a) Real

(b) Simulation



(c) Cells arrangement

I	W_r	P_F	P_B	D_F	R_{FF}	R_{BB}	R_{FB}	C_F	C_B	T
45	2.3	0.2	0.8	0.2	0.15	0.5	1.0	21261	4000	39.281

(d) List of appropriate controlled parameters

Figure 7.18: Giraffe 2

Giraffe 1 Figure 7.17 shows the synthesized giraffe pattern. The pattern was generated by using a large repulsive weight and longer simulation step. The assigned repulsive weight first brought a large number of cells together. The divisions of cells soon expanded this cell group to cover a region larger than the repulsive radius. Cells would not be grouped in this case. Instead, they were relaxed along the boundary and flat boundaries were formed.

Giraffe 2 Figure 7.18 shows another type of synthesized giraffe pattern. As one can see from the real skin pattern, some of the pattern features were slightly merged together. This situation was simulated by giving a smaller repulsive weight. In this case, the repulsive influence lost the capability for cell grouping much earlier than the previous example. As a result, the cells on different regions would start to group together, if they were too close to each other.

7.4 Rendering on Arbitrary Polygonal Objects

In previous section, several types of animal skin patterns were synthesized on a sphere. In this section, we applied these patterns on arbitrary polygonal objects. Six models were tested, the Stanford bunny model, cow, dophin, horse, and two versions of a triceratops, a low and a high resolution. The patterns of cheetah, king cheetah, leopard, jaguar, giraffe1 and giraffe2 obtained using our model in Section 7.3 were applied on these object models with identical simulation settings. Figure 7.19 to 7.24 show the results of the synthesis of skin patterns. The original polygonal surface is shown on the top row of each figure. The synthesized texture patterns on the surface are shown in the subsequence.

Figure 7.19: Stanford Bunny model with 3560 triangles and simulated with 15000 cells

Figure 7.20: Cow model with 5795 triangles and simulated with 20000 cells

Figure 7.21: Dophin model with 4198 triangles and simulated with 20000 cells

Figure 7.22: Horse model with 9924 triangles and simulated with 30000 cells

Figure 7.23: Triceratops model with 5660 triangles and simulated with 20000 cells

Figure 7.24: Triceratops with 11328 triangles and simulated with 30000 cells

7.5 Summary of Experimental Results

Four sets of experiment were performed. The first experiment was designed to examine the performance of our modified *Clonal Mosaic* models. The efficiency of our rendering method was demonstrated with the second experiment. In the third experiment, the plausibility of our techniques was evaluated with the generation of real animal skin pattern. The last experiment was dedicated to verify the practicality and generality of our proposed synthesis methods.

In the first experiment, the effectiveness of the proposed *Canonical Reference Plane Structure* (CRPS) and *Local Relaxation* (LR) methods was examined. CRPS attained success in all the test cases by showing fast simulation speed and compatible quality with those produced by the classical *Clonal Mosaic* model. This was mainly credited to the proposed canonical structure representation resulting in casting the three-dimensional transformation problem involved in the classical *Clonal Mosaic* model to much simple two-dimensional transformation problem. This certainly improves the overall computational performance of the simulation.

As compared to ORG model, a much shorter simulation time of LR model was achieved by ignoring those triangles which did not require cell relaxation. As a result, only a small number of triangles will had to be considered for the repulsion process. This led to noticeable improvement in overall performance. In some cases of the LR simulations, there were a few artifacts (i.e. non-rounded shapes) introduced into the synthesized results from processing relatively larger number of cells. The generation of these non-rounded patterns was largely due to the condition imposing that only neighboring cells situated within the repulsive circle will be relaxed. Hence, non-rounded shapes will be gradually formed, if a pattern covers a bounded circle with a radius larger than the repulsive radius. In general, there are about 2% of non-rounded shapes appear on a simulated animal skin. Hence, the quantity of such artifacts is hardly noticeable in the synthesized patterns and certainly acceptable in animation production.

The experiment in testing the rendering method shows that a texture pattern generated using our modified *Clonal Mosaic* model requires a slightly longer rendering time than several other promising texturing methods. It should be noted that these existing methods are not feasible for controlling and generating highly intricate animal skin patterns. Although a longer rendering time is generally expected from our method, it is still practically feasible

for pre-rendered animation production. To improve the rendering speed of the proposed model, it is envisaged that a more efficient closest-cell-searching methods will have to be developed.

Our proposed simulation method has shown successful results in synthesizing real animal skin patterns such as spot patterns on cheetah skin and patch patterns on giraffe skin. To date, our results of synthesizing the rosette skin patterns of both the leopard and jaguar could only be achieved at a moderate level of realism. This is mainly a limitation of the *Clonal Mosaic* model in the degree of controlling the interactions between pattern elements for generating complex patterns. Although attempts had been made in generating a more complex patterns, such as skin patterns on clouded leopard and ocelot, it was noted that the identification of appropriate sets of controlled parameters and the determination of their values are an extremely difficult and tedious task. In this regard, we will have to devise effective methods to at least facilitate, if not fully automate, the workflow of searching relevant parameter sets and their values for synthesizing highly complex animal skin patterns.

In the last experiment, the practicality and generality of the proposed models had been demonstrated by applying them to synthesize appealing patterns on several polygonal objects. In the synthesis process, the size, shape and complexity of these objects were varied. In all cases, appealing patterns were successfully synthesized on the target objects.

Chapter 8

Conclusion

The aim of the project is to develop a novel and effective theoretical and computational model for synthesizing the patterns of animal skins. The performance of the proposed system has been verified with extensive experimental results. The practicality and generality of the proposed simulation methods have been demonstrated with complex object models.

In this thesis, the problems and issues pertaining to pattern synthesis on three-dimensional arbitrary surface have been identified and discussed. A survey of the representative existing works has been conducted. Due to the generality and computational effectiveness of *Clonal Mosaic* model, it has been studied and described in detail. Having studied the model, the major drawbacks of the *Clonal Mosaic* model have been identified and carefully addressed. Relevant enhancements have been proposed and implemented to resolve some of these drawbacks. These include introducing the *Canonical Reference Plane Structure* for the distance computation and cells' transformation, optimally selecting the relevant neighborhoods for repulsion computation of cells till their forces are neutralized, and rendering of texture pattern in a highly effective way. The performance gain of these proposed techniques has been successfully verified with relevant test examples.

In this chapter, our contributions in synthesizing animal skin patterns are first presented. Next, the potential issues and problems of the proposed models are addressed. Finally, the recommendation of future works for alleviating some of these problems is discussed.

8.1 Contributions

To date, we have successfully developed a system inspired by *Clonal Mosaic* model for synthesizing skin patterns efficiently with moderate complexity. The key contributions of our proposed system are summarized as follows.

Canonical Reference Plane Structure

A *Canonical Reference Plane Structure* is introduced to compute the relevant information for facilitating the cell repulsive process. A three-dimensional polygonal surface is converted into this structure, whereby all the triangles are transformed onto a two-dimensional *Reference Plane*. An efficient *Neighbor Mapping* function is also derived to properly map triangles and cells onto a neighborhood. This neighborhood simplifies and facilitates the tasks of measuring of the distance between cells and movement of cells, as only two-dimensional computation is involved. The experiments clearly show that, the simplification from a three-dimensional to a two-dimensional problem has effectively reduced the computational complexity and significantly improved the overall system performance.

Enhancement on Clonal Mosaic Simulation

The process of *Clonal Mosaic* simulation, consists of cell division and relaxation process, is computationally expensive. The key modules of the simulation are repulsion of cells, *Neighbor Mapping* and movement of cells. These modules will have to be applied to each cell in every iteration of the process. To alleviate this problem, an effective scheme has been developed to optimally process cells which are affected during the simulation cycle. The *Local Relaxation* process updates a list of interrupted triangles in each time interval. The cells on these interrupted triangles will be undergone a relaxation process. This can avoid the processing of those surface regions, which are not interrupted by cell division. The efficiency of the computational performance of the proposed *Local Relaxation* method is experimentally proven to be higher than the classical approach.

During the relaxation process, it is generally necessary to associate a moving cell to the identity of a triangle on which the cell is situated at stable condition. To establish this correspondence, a cell-in-triangle strategy has been devised to efficiently locate the relevant triangle for a given stable cell.

Rendering Texture Pattern

To determine the color value of a screen pixel during rendering, a relevant corresponding spatial point of the pixel will have to be first identified. The color of the *Clonal Mosaic* pattern is realized through a Voronoi diagram, whereby a spatial point will be associated to the closest *Clonal Mosaic* on the surface. The color of the cell will then be assigned to the point. In our case, a mechanism is proposed for locating the corresponding target cell without the need of *Polyhedral Voronoi Diagram* being used in the classic *Clonal Mosaic* model. We have developed a *Canonical Reference Plane Structure* representation to facilitate a process of determining a closest cell from a given rendered point. This improves the overall efficiency of the rendering process without any degradation of the quality of the rendered result.

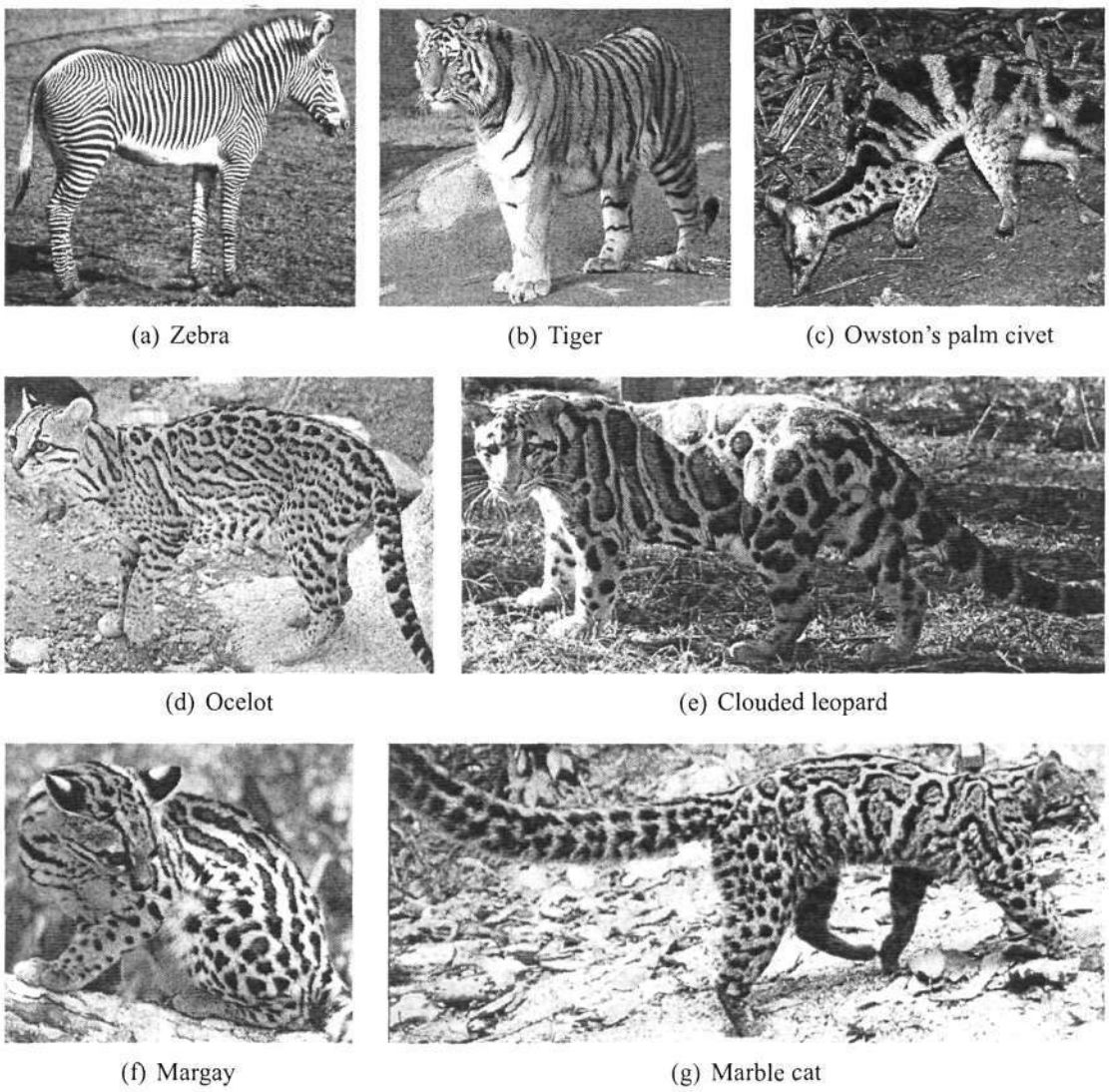
8.2 Recommendation for Further Research

In this section, we will discuss two major problems of the existing synthesis techniques. Firstly, it is difficult to achieve impressive synthesized results for highly complex animal skin patterns using the existing texture synthesis systems. The second problem is that the existing methods require considerable efforts in identifying and establishing extensive lists of parameters for controlling the simulation processes and visual intricacies of the synthesized patterns. We shall discuss these two problems in greater detail in the subsequent sections.

8.2.1 Synthesis of Highly Complex Skin Patterns

Despite the success of synthesizing some of the animal skin patterns using *Clonal Mosaic* model, there are still not promising methods for realistically synthesizing animal skins with highly complex patterns such as skin stripe patterns of tiger, zebra and rosette patterns of various cats as shown in Figure 8.1.

The stripe pattern is hard to control, as the direction of the stripes varies on different parts of the body, especially on the head of zebra and tiger. For a palm civet, spots and stripes are coexist on some of the body parts. It is obvious to find some linkage between the rosette patterns on various cats, where a dark ring structure is circling a darker patch



(a) Zebra (b) Tiger (c) Owston's palm civet

(d) Ocelot (e) Clouded leopard

(f) Margay (g) Marble cat

Figure 8.1: Highly complex skin patterns

of color region. Pattern on leopard has the most fundamental structure of a rosette, where several black spots form a ring. Jaguar has the similar formation, but the ring of spots is mostly connected into line and black spots occur within the ring. A marble cat has similar ring pattern as jaguar, but its shape is more irregular. For ocelot and margay, the shape of rings is irregular, but follows a direction that gives an impression of stripes. A clouded leopard gives a much more strange formation of pattern, where a ring is narrower towards the head and thicker towards the other side. Besides, the size of rosette patterns is much bigger than those on other cats. Although the patterns on the body are quite different, there exist similar spots on their limbs, tail and head.

In order to generate highly complex skin patterns and their variations, it is important that the proposed model can be easily incorporated with the relevant knowledge for controlling the process of pattern formation on an arbitrary three-dimensional surface. The existing

Reaction-Diffusion model and *Clonal-Mosaic* model both rely on the behavior of individual cell in pattern formation. Although the concept is reasonably valid from biological view, this type of mechanism is difficult to be controlled in computer graphics, as the modelling of animal surfaces and the texturing process do not necessary take into the consideration of the biological development.

As an efficient computational tool, a planning process, which allows local control of pattern variations should be provided. It may be possible to realize this local control by developing a subdivision *Clonal Mosaic* model on cells at different stages.

8.2.2 Parameter Estimation for Clonal Mosaic Model

As mentioned before, a pattern generation from *Clonal Mosaic* model requires the manipulation of a large number of parameters. It is difficult to obtain a desired pattern without knowing the influence of each parameter. It is generally a labor intensive and tedious task to identify and determine lists of controlled parameters for generating desired complex texture patterns.

A similar problem is studied by Xuejie [47]. He proposed a genetic-based multi-resolution approach to estimate parameter values for a given procedural texture. The key idea is to use an efficient genetic-based search algorithm to find appropriate values of the parameters. During the evolving process, texture patterns are generated for lists of parameter values. These patterns are then automatically compared with the target texture pattern. The search will be terminated once the preset matching requirements are met.

Ruiz-Del-Solar, Parada and Koppen [48] [49] developed a texture retrieval system, named TEXRET-System. A database is created to keep all the available texture patterns. Any requested texture which is missing from the database will be synthesized with *Causal Autoregressive Model* and *Interactive Genetic Algorithm*. This system allows a continuous growing of the database.

A database can be constructed to store the parameter sets and their corresponding texture patterns. Now, a *Clonal Mosaic* will synthesize a texture pattern if a sample pattern is same as the desired pattern of a given input image being found in the database. If not, the parameter estimation is required to be performed for generating a pattern which is compatible to the desired one. To produce an appropriate pattern, a genetic-based approach

will have to be developed and integrated into the *Clonal Mosaic* model for determining an relevant parameter set.

Bibliography

[1] How Jiann Teo and Kok Cheong Wong. Texture pattern generation using clonal mosaic. *Journal of Computer Science and Technology*, 21(2):173–180, March 2006.

[2] Marcelo Walter, Alain Fournier, and Mark Reimers. Clonal mosaic model for the synthesis of mammalian coat patterns. In *Graphics Interface*, pages 82–91, 1998.

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

[4] A.Gierer and H.Meinhardt. A theory of biological pattern formation. *Kybernetik*, 12(1):30, 1972.

[5] H.Meinhardt and M.Klinger. A model for pattern formation on shells of molluscs. *Journal of Theoretical Biology*, 126:63–69, 1987.

[6] A.J.Kochy and H.Meinhardt. Biological pattern formation: from basic mechanisms to complex structures. *Reviews of Modern Physics*, 66(4):1481–1507, October 1994.

[7] Hiroto Shoji and Yoh Iwasa. Pattern selection and the direction of stripes in two-dimensional turing systems for skin pattern formation of fishes. *Forma*, 18(1):3–18, 2003.

[8] Greg Turk. *Texturing Surfaces Using Reaction-Diffusion*. PhD thesis, University of North Carolina, 1992.

[9] Marcelo Walter, Alain Fournier, and Daniel Menevaux. Integrating shape and pattern in mammalian models. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 317–326, New York, NY, USA, 2001. ACM Press.

[10] Philip Ball. *The Self-Made Tapestry: Pattern Formation in Nature*, pages 7–99. Oxford University Press, December 1998.

[11] Deborah R. Fowler, Hans Meinhardt, and Przemyslaw Prusinkiewicz. Modeling seashells. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 379–387, New York, NY, USA, 1992. ACM Press.

[12] Hans Meinhardt. *The Algorithmic Beauty of Sea Shells*. Springer Verlag, December 1995.

[13] K. J. Painter, P. K. Maini, and H. G. Othmer. Stripe formation in juvenile pomacanthus explained by a generalized turing mechanism with chemotaxis. In *Proceedings of The National Academy of Sciences of the United States of America*, pages 5549–5554. National Academy of Sciences, May 1999.

[14] K.J. Painter. Models for pigment pattern formation in the skin of fishes. In P.K. Maini and H.G. Othmer, editors, *Mathematical Models for Biological Pattern Formation*, volume 121 of *The IMA Volumes in Mathematics and its Applications*, pages 59–82. Springer, 2000.

[15] J. D. Murray. On pattern formation mechanisms for lepidopteran wing patterns and mammalian coat markings. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 295(1078):473–496, October 1981.

[16] J.D. Murray. *Mathematical Biology II: Spatial Models and Biomedical Applications*, pages 141–191. Springer, 3 edition, January 2003.

[17] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000.

[18] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (TOG)*, 20(3):127–150, 2001.

[19] Junhwan Kim and Fabio Pellacini. Jigsaw image mosaics. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 657–664. ACM Press, 2002.

[20] Fabrice Neyret and Marie-Paule Cani. Pattern-based texturing revisited. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 235–242, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[21] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, New York, NY, USA, 2001. ACM Press.

[22] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics (TOG)*, 22(3):295–302, 2003.

[23] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 665–672, New York, NY, USA, 2002. ACM Press.

[24] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 465–470, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[25] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 673–680, New York, NY, USA, 2002. ACM Press.

[26] Kun Zhou, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. Texturemontage. *ACM Transactions on Graphics (TOG)*, 24(3):1148–1155, 2005.

[27] Ken Perlin. An image synthesizer. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296. ACM Press, 1985.

[28] Matt Zucker. The perlin noise math faq. http://www.robo-murito.net/code/perlin-noise-math-faq.html, February 2001.

[29] Paul Bourke. Perlin noise and turbulence. http://astronomy.swin.edu.au/ pbourke/texture/perlin/, January 2000.

[30] Hugo Elias. Perlin noise. http://freespace.virgin.net/hugo.elias/models/m_perlin.htm.

[31] Greg Turk. Generating textures on arbitrary surfaces using reaction -diffusion. In *Computer Graphics (SIGGRAPH'91) Proceedings*, volume 25, pages 289–298, July 1991.

[32] Jonathan B. L. Bard. A model for generating aspects of zebra and other mammalian coat patterns. *Journal of Theoretical Biology*, 93(2):363–385, November 1981.

[33] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 299–308, New York, NY, USA, 1991. ACM Press.

[34] Kurt W. Fleischer, David H. Laidlaw, Bena L. Currin, and Alan H. Barr. Cellular texture generation. *Computer Graphics*, 29(Annual Conference Series):239–248, 1995.

[35] Kurt Fleischer and Alan H. Barr. A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In *Artificial Life III*, pages 389–416. Addison-Wesley, 1993.

[36] Kurt Fleischer. Investigations with a multicellular developmental model. In *Artificial Life V*, pages 229–236, 1996.

[37] Marcelo Walter. *Integration of Complex Shapes and Natural Patterns*. PhD thesis, Federal University of Rio Grande do Sul - Brazil, December 1998.

[38] Mark Meyer, Haeyoung Lee, Alan Barr, and Mathieu Desbrun. Generalized barycentric coordinates on irregular polygons. *j-J-GRAPHICS-TOOLS*, 7(1):13–22, 2002.

[39] Greg Turk. Generating random points in triangles. In Andrew S Glassner, editor, *Graphics Gems*, pages 24–28. Boston : Academic Press, 1990.

[40] William H.Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition edition, February 2002.

[41] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition edition, September 2001.

[42] George Taylor. Point in polygon test. *Survey Review*, 32(254):479–484, 1994.

[43] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley and Sons, 2 edition, 1999.

[44] David Eberly. Rotation representations and performance issues. http://www.geometrictools.com/Documentation/RotationIssues.pdf, January 2002.

[45] Eric Haines. Point in polygon strategies. In Paul Heckbert, editor, *Graphics Gems IV*, pages 24–46. Academic Press, 1994.

[46] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294. ACM Press, 1996.

[47] Xuejie Qin and Yee-Hong Yang. Estimating parameters for procedural texturing by genetic algorithms. *Graphical Models*, 64(1):19–39, 2002.

[48] Javier Ruiz-Del-Solar, Patricio Parada, and Mario Koppen. Texture synthesis using soft-computing. In *Proc. of the Fourth Asian Fuzzy Systems Symposium - AFSS 2000*, pages 605–610, 2000.

[49] Ruiz-del-Solar J. Koppen M. Parada, P. and W. Plagges. Interactive texture synthesis. In *Proc. of the 11th Int. Conf. on Image Analysis and Processing - ICIAP 2001*, pages 434–439, 2001.

Appendix - Publications

There are two publications from the author of this thesis.

- Texture Pattern Generation Using Clonal Mosaic in *Journal of Computer Science and Technology, Volumn 12, Number 2.*
- Optimization of Clonal Mosaic Model with Canonical Reference Plane and Local Relaxation in *13th Pacific Conference on Computer Graphics and Applications.*

The photocopy of the publications are attached together with this thesis.

Mar. 2006, Vol.21, No.2, pp.173–180 J. Comput. Sci. & Technol.

Texture Pattern Generation Using Clonal Mosaic*

How Jiann Teo (张厚健) and Kok Cheong Wong (黄国长)

Center for Graphics and Imaging Technologies, School of Computer Engineering
Nanyang Technological University, Singapore

E-mail: pa12432790@ntu.edu.sg; askcwong@ntu.edu.sg

Revised September 12, 2005.

Abstract In this paper, an effective system for synthesizing animal skin patterns on arbitrary polygonal surfaces is developed. To accomplish the task, a system inspired by the Clonal Mosaic (CM) model is proposed. The CM model simulates cells' reactions on arbitrary surface. By controlling the division, mutation and repulsion of cells, a regulated spatial arrangement of cells is formed. This arrangement of cells shows appealing result, which is comparable with those natural patterns observed from animal skin. However, a typical CM simulation process incurs high computational cost, where the distances among cells across a polygonal surface are measured and the movements of cells are constrained on the surface. In this framework, an approach is proposed to transform each of the original 3D geometrical planes of the surface into its Canonical Reference Plane Structure. This structure helps to simplify a 3D computational problem into a more manageable 2D problem. Furthermore, the concept of Local Relaxation is developed to optimally enhance the relaxation process for a typical CM simulation. The performances of the proposed solution methods have been verified with extensive experimental results.

Keywords procedural texture, clonal mosaic, pattern synthesis, geometry processing

1 Introduction

The aim of this project is to synthesize the fascinating color patterns of animals in the application domain of computer graphics. Several biological models for the pattern formation on animal skin are studied. Computational approaches for the synthesis task have also been examined. The existing models have successfully predicted the pigmentation processes of the seashells and the color pattern formation on several species of fishes. Besides, the previous work on simulating simple patterns on mammalian coats also achieved certain success. Despite this achievement, it is difficult to obtain acceptable computational performance and/or realistic results by using existing methods for synthesizing animal skin with complex patterns.

Our work is inspired by a procedural texture method, namely, *Clonal Mosaic* (CM) model, proposed by Walter[1]. This model works with a large number of points (cells), which are distributed over a piece of geometry. The movement and division of cells are controlled by some parameters, so that by grouping cells with similar properties and keeping away cells with different properties, a regulated spatial arrangement of cells can be formed. The resulting arrangement of cells, once associated with a *Voronoi Diagram*, represents a tessellation of cell regions, which are visually identical with color patches on animal skin. A CM model is able to create various animal skin patterns like spots on cheetah, patches on giraffe and rosettes on leopard. However, a typical CM texture requires an extensive simulation process.

Our goal is to improve the computational performance of the CM simulation on arbitrary polygonal surfaces, while maintaining the quality of a simulated texture pattern. Two techniques are successfully developed, in order to overcome the drawbacks and limitations of the existing CM model. First, an intermediate spatial configuration of polygonal object, which is called as the *Canonical Reference Plane Structure* (CRPS), is proposed. This geometrical structure aims at simplifying the computational cost incurred during a simulation process. Second, a *Local Relaxation* is introduced, so that the simulation process can be optimized by avoiding redundant computation of cell repulsions.

2 Related Work

Ball[2] conducted a comprehensive survey on biological animal skin pattern formation. One of the earliest discussions on pattern formation in biological discipline was initialized by Alan Turing[3]. He proposed a *Reaction-Diffusion* (RD) system, which explains how the anatomical structure of an organism is developed from its zygote. In Turing's chemical system, diffusion is acting in competition with an autocatalytic chemical reaction. A chemical compound A undergoes an autocatalytic reaction to produce more of itself. While another compound B acts as an inhibitor, which inhibits the production of more A. The key element for obtaining a spatial pattern is to diffuse A and B through the reaction medium at different rates.

Gierer and Meinhardt[4] described an *Activator-Inhibitor* and *Activator-Substrate* model, which are types of RD systems. These methods are proven true in generating remarkable seashell pigmentation[5,6]. The concepts were also used by Koch and Meinhardt[7] to simulate several types of mammalian coat pattern such

*A preliminary version of this paper appeared in Proc. Pacific Graphics 2005, Macau.

as the cheetah, giraffe and leopard. However, the model did not give comparable realistic result with the real world patterns. Painter[8] simulated an RD system on an irregular domain determined by tracing the boundary of a real fish.

Murray[9] suggested that the formation of pattern on mammalian coat starts in the embryonic stage. Besides, the scale of body size has dramatic effect on pattern formation. Small animals with short gestation periods have less complex skin patterns than larger animals, as their smaller embryos support fewer modes. The size and shape of the region, where a chemical process occurs, also influence the pattern formation process. For example, the formation of either spots or stripes pattern on a cat's tail is decided by the length and shape of the tail.

In computer graphics, textures are attached onto a piece of geometry in order to increase its realistic appearance. There exist two major ways to texture a piece of geometry, namely image-based texturing and procedural texturing.

An image-based texturing technique employs a piece of 2D image, which is either painted by an artist, captured with image scanner or digital camera. A texture mapping process is needed to create a correspondence between 3D surface point and a pixel on the image. There are several problems for this approach. First, a given image might not be large enough to cover the entire surface. Methods have been proposed to solve this problem by repeatedly tiling the texture image[10,11]. However, it is difficult to blend the patterns at the sides of the texture images (texture seam problem). Secondly, a 2D image does not always fit well to an arbitrary surface manifold. Hence, stretching and distortion are common. Next, it is normally difficult to create variation and transition of texture elements in this approach.

A texture synthesis method attempts to solve some of these potential problems. One or more smaller texture images are used as input to the system for generating a larger and richer image. Praun[12] gives an approach to cover an arbitrary surface with overlapped texture patches from an input image. Tong[13] describes a bi-directional texture function, which creates seamless texture from an input image. Soler[14] proposed a hierarchical pattern mapping technique, which iteratively fills geometry with small patches of an input image in different sizes and shapes. Zhang[15] shows a progressively variant synthesis on arbitrary surface.

For procedural texture, mathematical models are used to generate texture directly on a surface. The most widely used procedural texture is *Perlin Noise*, which can be used to generate a variety of 2D and 3D textures (including marble, water, fire and other kinds of natural phenomena).

Turk[16] used the biological pattern formation technique, RD model, to generate mammalian coat pattern on arbitrary surfaces. The system simulates the interaction of chemical pigments as they diffuse over the surface. Turk adopted a suggestion from Bard[17], where a complex pattern can be generated by a cascade process. The process combines the results of several RD systems by freezing one of the processes while running the other process. This cascading process successfully simulated the typical large and small spots on cheetah and rosette patterns on leopard.

Witkin and Kass[18] introduced the anisotropy concept into the RD system. The concept allows different diffusion rates for horizontal and vertical directions. The anisotropy information is given by a diffusion-map, which is defined by the user.

Walter[1] proposed the CM model, where attractive patterns are formed by simulating cell reactions. Cell division, mutation and repulsion are the key elements to obtain these patterns.

3 Overview

CM model is a method, which simulates mammalian coat patterns on arbitrary 3D surfaces. The model proposes that the typical spot, rosette and strip patterns, which occur in several species of mammals, reflect a spatial arrangement of epithelial cells. These cells are derived from a single progenitor. The color of furs and hairs are hence affected by these underlying cells. The generation of CM patterns includes the simulation of three cell actions, which are the cell division, cell mutation and cell repulsion.

A CM cell is represented as a point in a given domain. Several types of cells, where each type is responsible for the synthesis of a single color appearance of the entire texture pattern, are introduced into the system. Typical properties of a given type of cell are cell color, division rate, mutation rate and repulsive value. Division rate controls the frequency of a cell to divide itself. Mutation rate controls the probability of a cell to mutate itself into another type of cell. Repulsive value defines repulsiveness between any two types of cells.

All the cells in the system divide themselves into two according to their division rate. The progenies inherit all attributes of their parent if no cell mutation happens. Whenever a division event happens, a newly produced cell repulses its neighboring cells. Repulsion computation is needed to displace the cells accordingly.

The process to compute repulsion among cells and displace the cells according to the repulsive forces is named as a relaxation process. The total repulsive force received by a single cell $\boldsymbol{P}_c$ is computed by summing all repulsive forces from its neighboring cells $\boldsymbol{P}_i$. For a total area A and m number of cells, the neighbor cells are those who reside within a given repulsive radius r from $\boldsymbol{P}_c$, where $r = w_r\sqrt{A/m}$. The repulsive weight w_r acts as the scaling factor of the repulsive radius. Fig.1 shows a cell $\boldsymbol{P}_c$ with two neighbor cells $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, which are located within the repulsive radius r. $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ each apply an acting force $\boldsymbol{f}_{1c}$ and $\boldsymbol{f}_{2c}$ onto $\boldsymbol{P}_c$. $\boldsymbol{P}_c$ is hence receiving a total acting force of $\boldsymbol{F} = \boldsymbol{f}_{1c} + \boldsymbol{f}_{2c}$, which

then displaces $\boldsymbol{P}_c$ to a new position $\boldsymbol{P}'_c$. The distance vector $\boldsymbol{D}_{ic}$ is given by

$$\boldsymbol{D}_{ic} = \boldsymbol{P}_c - \boldsymbol{P}_i$$

The repulsive force $\boldsymbol{f}_{ic}$ from $\boldsymbol{P}_i$ to $\boldsymbol{P}_c$ is given by

$$\boldsymbol{f}_{ic} = \alpha_{ab}(r - |\boldsymbol{D}_{ic}|)\frac{\boldsymbol{D}_{ic}}{|\boldsymbol{D}_{ic}|}$$

where α_{ab} is the repulsion value defined between cells of type a and type b. The term $\boldsymbol{D}_{ic}/|\boldsymbol{D}_{ic}|$ gives the direction of the repulsive force. The term $(r - |\boldsymbol{D}_{ic}|)$ is used to scale the strength of the repulsive force according to the cell distance. The new position of $\boldsymbol{P}_c$ is then determined by:

$$\boldsymbol{P}'_c = \boldsymbol{P}_c + w_d \sum_{i=1}^{n} \boldsymbol{f}_{ic}$$

where w_d is a weighting factor and n is the total number of neighboring cells.

The simulation process formulates cells in a regulated spatial arrangement on the given geometry. The geometry can then be shaded according to the color information of the cells, in order to produce realistic skin pattern. The classic CM model used an approximation of the *Polyhedral Voronoi Diagram* to represent tessellation of cells on the surface. Shading of the surface is done by filling each of the *Voronoi Polygons* with the cell color.

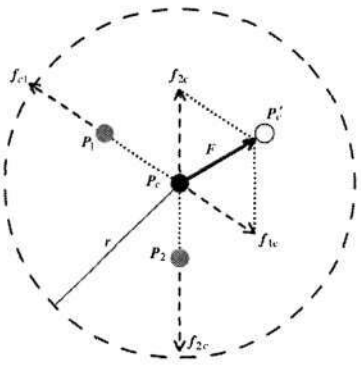


Fig.1. Calculation of repulsive forces among cells.

4 Proposed Techniques

A system is carefully designed, so that a CM pattern is efficiently generated on arbitrary polygonal surfaces. The workflow of the proposed system is shown in Fig.2. A polygonal surface is transformed into the proposed CRPS. This structure keeps all polygons and cells on a common plane, so that subsequent computations for cells' operations are highly simplified. A series of initialization processes take place in Step 2. Points are randomly distributed over the geometry. These points undergo a relaxation process to form a regularly spaced point set. Lastly, each of the points is associated with a cell type. Cell type defines the color properties, repulsiveness, division rate and mutation rate of cells. The number of cell types involves in the simulation affects the diversity of a CM pattern.

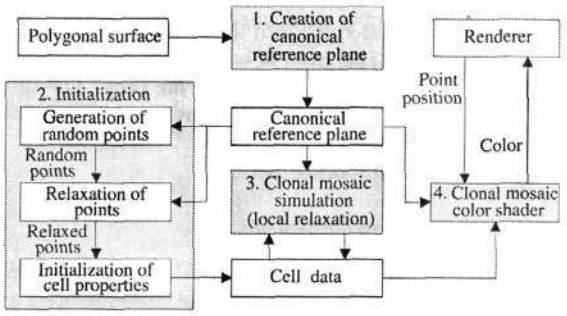


Fig.2. Workflow of the system.

The actual simulation of CM pattern is conducted in Step 3. *Local Relaxation* is proposed to substitute the classical relaxation process, which is performed globally. The *Local Relaxation* process avoids redundant computation of cell repulsions and hence reduces the simulation time. Lastly, a *CM Pattern Shader*, which determines surface color during rendering, is introduced. A *Closest Cell Searching* is performed for each of the surface point. A surface point will be shaded with the color of the cell, which is closest to it. This module eliminates the complex computation of *Polyhedral Voronoi Diagram*. Nevertheless, the resulting texture pattern gives identical result as those generated by computing a *Voronoi Diagram*.

4.1 Canonical Reference Planes Structure (CRPS)

A CM simulation relies heavily on repulsion computation, where repulsive forces among cells are obtained. The problem becomes complicated as the shortest path and distance among cells are computed on an arbitrary polygonal surface. Techniques were proposed in order to approximate the distance by finding the shortest path across the surface[19,20]. These techniques are accurate, but they are complicated and require heavy computation.

Other techniques[1,16] provide a mapping function, which map cells on neighboring polygons of a polygon onto the common plane in which the polygon resides. The distances among cells are then approximated by measuring their Euclidean distances on the plane. The computation of this method is rather straightforward and the approximation error is acceptable.

However, this method involves massive computation of rotations and cell mappings. During repulsion computation for a target cell, cells residing on neighboring polygons are transformed onto the common plane

in which the target cell resides. Transformation matrixes are pre-computed for each pair of neighboring polygons, which share either an edge (primary neighbor) or a vertex (secondary neighbor). For neighboring polygons, which are connected through a primary or secondary neighbor, a series of transformation is concatenated. Distances among neighboring cells are measured as their Euclidean distances on the common plane. The resultant force is computed by summing the repulsive forces from the neighboring cells. The target cell is displaced with the resultant force. The system will verify whether the displacement causes the cell to move onto a neighboring polygon. A series of transformation is again needed in order to properly rotate the cell onto the neighboring polygon.

Eberly[21] made an analysis on the performance issues of various rotation techniques. He suggested that an optimum choice for less memory and faster computation be to store the rotation information in quaternion and convert it into matrix whenever a rotation is performed. It requires storage of four floats as quaternion, twelve additions and twelve multiplications for matrix conversion, six additions and nine multiplications to rotate each point.

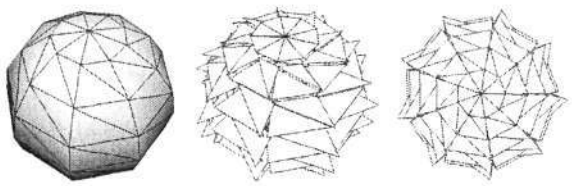


Fig.3. Canonical Reference Plane Structure. (a) Original Geometry. (b) All polygons are aligned with XZ plane. (c) Overlapped polygons in a Canonical Reference Plane Structure on a 2D domain.

A data structure, which transforms all polygons of an object onto a common plane, is proposed here. Transformation and distance measurement of cells are then performed on this common plane. Since all polygons are on a common plane, the problem is simplified from a 3D computational problem into a more manageable 2D problem. Fig.3(a) shows a polygonal sphere. Fig.3(b) shows that all polygons of the sphere are transformed in such a way that they are aligned with the XZ plane. By ignoring the Y coordinate, all polygons are now residing on a 2D domain (see Fig.3(c)). Polygons in this 2D domain are overlapped with each other and have their normals pointing towards Y-axis. Cells are then populated on the polygons in the 2D domain. Neighbor mapping function is derived for the CRPS, so that neighbors of a polygon are correctly mapped during a repulsion computation. The distances among cells are then measured as their 2D distances on the neighborhood after the neighbor mapping.

4.1.1 Creation of CRPS

The creation of CRPS for a polygonal object is straightforward. All polygons of the polygonal object are rotated in such a way that their normals are aligned with Y-axis. By ignoring the Y-coordinate, the polygon is mapped onto the XZ plane. The XZ plane is called as a *Reference Plane* and the rotated polygon is called as a *Reference Polygon*.

CRPS consists of two sets of data. The first one stores a list of *Reference Polygons*. Each of the *Reference Polygons* is assigned with attributes as shown in Table 1. The Normal, Yoffset and Center of a polygon are stored for transforming the *Reference Polygons* and their cells back to the 3D domain. Note that the position of polygon center and vertices are stored as 2D coordinate. Cells are randomly distributed over the *Reference Polygons* according to the algorithm suggested by Turk[22]. The cells' positions are also stored as their 2D coordinate on the *Reference Plane*.

The second set of data contains neighbor mapping information of *Reference Planes*. It will be further discussed in the next subsection.

Table 1. Attributes of a Reference Polygon

Attribute	Meaning	Type
Normal	Normal vector of original polygon	Float3
Yoffset	Vertical distance of the original polygon to XZ plane	Float
Center	2D position of polygon center	Float2
Vertices	2D positions of polygon vertices	Float2

4.1.2 Neighbor Mapping with Reference Plane

Reference Polygons in CRPS are disconnected from and overlapped with one another. In a cell repulsion computation, neighbors of a polygon in which a target cell is residing, are properly transformed to form a neighborhood. Euclidian distances among cells on this neighborhood are taken as the approximated distances among the cells. Fig.4(a) shows several *Reference Polygons* and Fig.4(b) shows a neighborhood, which is formed after the neighbor mapping. The shaded polygon identifies the polygon in which a target cell resides.

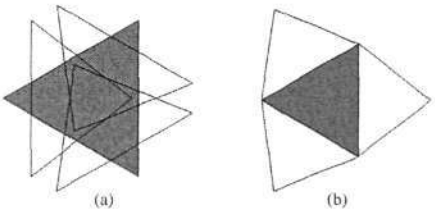


Fig.4. Reference plane structure. (a) Overlapped reference polygons. (b) Mapped neighborhood.

For any two non-parallel planes P_i and P_j (where two neighboring polygons reside), they intersect at a line L_{ij}. L_{ij} is taken as the rotation axis for mapping. For primary neighbors, the intersection line of a master

polygon R_i with a neighbor R_j is also their sharing edge E_{ij} (see Fig.5(a)). For a secondary neighbor, it is the line passing through a vertex $\boldsymbol{V}_{ij}$, which connects both polygons (see Fig.5(b)).

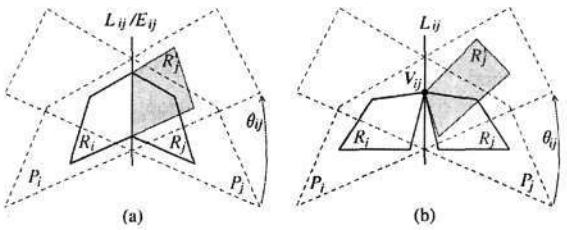


Fig.5. Intersection line between two connected neighbors. (a) Intersection line as a sharing edge. (b) Intersection line passes through a vertex.

Given a polygon R_i and a neighboring polygon R_j, the intersection line L_{ij} can be computed by getting a cross product C_{ij} of the polygons' normal $\boldsymbol{N}_i$ and $\boldsymbol{N}_j$. The position of a vertex $\boldsymbol{V}_{ij}$, which connects both polygons, is taken as origin of the line. The angle θ_{ij} to rotate R_j to R'_j is then given by the angle difference between $\boldsymbol{N}_i$ and $\boldsymbol{N}_j$.

Fig.6(a) shows how the 2D transformation information is derived. Given a polygon P_i and its neighbor P_j, rotation $\boldsymbol{R}(\theta_i)$ and $\boldsymbol{R}(\theta_j)$ are found to give *Reference Polygons* R_i and R_j. Their sharing edge (i.e., the intersection line) E_{ij} is then rotated to E'_i and E'_j respectively by the rotation $\boldsymbol{R}(\theta_i)$ and $\boldsymbol{R}(\theta_j)$. An angle θ_{ij} between E'_i and E'_j can then be used to rotate R_j to an orientation, which aligns with R_i. Next, a translation is needed to connect R_i and R_j together (see Fig.6(b)).

The rotation and translation required to map a *Reference Polygon* onto a neighborhood can be represented by a 2D transformation matrix. First, the frame, where all neighbors of a polygon are transformed to, is named as a master frame. The frame, where a neighbor is originally resided, is named as a neighbor frame. In Fig.7, R_i is a master polygon and R_j is a neighbor. They are connected with a vertex, which has a position of $\boldsymbol{M}_i$ in the master frame and $\boldsymbol{M}_j$ in neighbor frame. We first translate R_j into the origin O with $\boldsymbol{T}(\boldsymbol{M}_i)$. Next, R_j is rotated about the origin O with $\boldsymbol{R}(\theta_{ij})$. Lastly, R_j is translated to the master frame with $\boldsymbol{T}(-\boldsymbol{M}_j)$. The concatenated transformation function $\boldsymbol{F}_{ji}$ is derived as follows:

$$\boldsymbol{F}_{ji} = \boldsymbol{T}(\boldsymbol{M}_i) \cdot \boldsymbol{R}(\theta_{ij}) \cdot \boldsymbol{T}(-\boldsymbol{M}_j) = \begin{bmatrix} C_{ij} & -S_{ij} & A_x \\ S_{ij} & C_{ij} & A_y \\ 0 & 0 & 1 \end{bmatrix}$$

where $C_{ij} = \cos(\theta_{ij})$, $S_{ij} = \sin(\theta_{ij})$, $A_x = -M_{jx}C_{ij} + M_{jy}S_{ij} + M_{ix}$, and $A_y = -M_{jx}S_{ij} - M_{jy}C_{ij} + M_{iy}$.

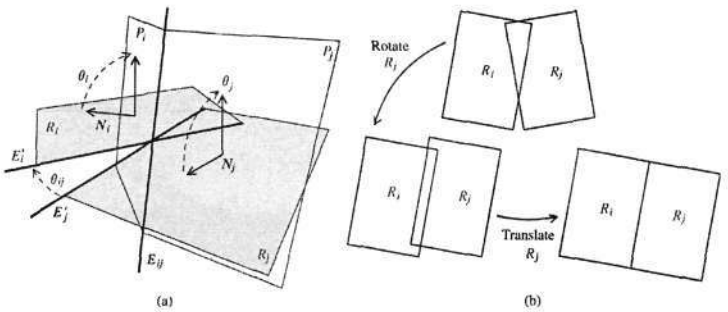


Fig.6. Reference polygon and neighbor transformation. (a) Creating reference polygon. (b) Transforming reference polygons.

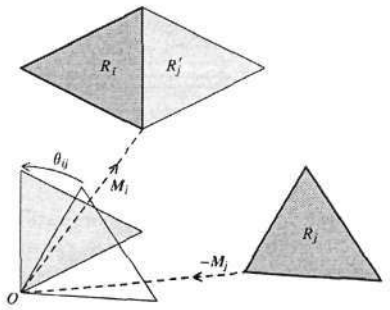


Fig.7. Transformation between master and neighbor polygons.

Instead of storing the 3×3 matrix in the memory, we only store the values of C_a, S_a, A_x and A_y for each pair of neighboring polygons. A cell can be transformed to a frame at any time by using this transformation information between that pair of neighbors. The transformation equation for a cell is given as follows:

$$\boldsymbol{P}'_x = \boldsymbol{P}_x C_{ij} - \boldsymbol{P}_y S_{ij} + A_x$$
$$\boldsymbol{P}'_y = \boldsymbol{P}_x S_{ij} + \boldsymbol{P}_y C_{ij} + A_y.$$

As we can see, memory storage of four floats is needed and the cost for transforming a cell is reduced to four additions and four multiplications as compared to previous six additions and nine multiplications. Fig.8 shows the performance comparison. "3D" identifies the data from the original model, where polygons and cells are rotated with quaternion and matrices. "CRPS" app-

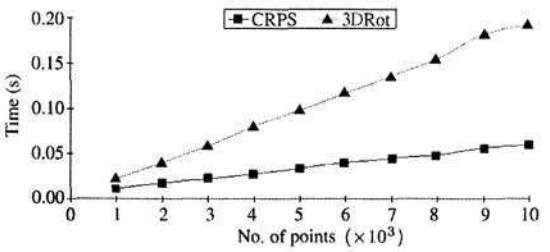


Fig.8. Performances comparison of canonical reference plane structure with original 3D rotation computation.

roach identifies the data from our system, where polygons and cells are transformed in the space of the *Reference Plane* with 2D transformation matrices. This test is conducted on an object with 200 polygons. In this test, five iterations of cell relaxation are performed. When the number of cells increases, "CRPS" has introduced a relatively smaller increment in its computation time than the "3D" approach, as the overhead spent in transforming a cell is relatively cheaper than the original model.

4.2 Clonal Mosaic Simulation

The CM Simulation process performs the simulation of cells' reactions on a given surface. These reactions are cell division, cell mutation and cell repulsion. The division and mutation actions are realized through an event queue. Each of the cells is scheduled for division according to its division rate. The cell mutation happens when a new cell is created and it is controlled by its mutation rate.

A series of division events D_i are queued in a heap structure (see Fig.9). Each of the division events has a time value T_{Di}, which is the time for a cell to divide itself. All division events within a given time frame ΔT are processed before a relaxation process is performed. Each newly divided cell will be introduced back to the queue for future division. The simulation keeps running until the ending time T_{end} is reached.

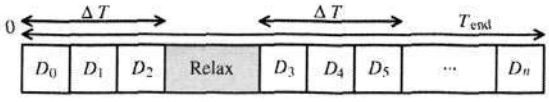


Fig.9. Event queue for CM simulation.

The major computational load incurred in a CM simulation is the computation for cell repulsion. The classical relaxation method relaxes all the cells on a surface together in each interval (called as *Global Relaxation*). Hence, there exists redundant computation for those regions, where relaxed spatial arrangement of cells is already obtained. In general, a stable spatial cells arrangement in a region is only interrupted when a new cell is introduced into the region. As a result, by relaxing cells only on these local region will effectively reduce the computational load. This proposed method is named as *Local Relaxation*.

4.2.1 *Local Relaxation*

The idea of *Local Relaxation* is to relax only the small region, which is interrupted by a cell division event. The simplest idea to achieve this is to relax the neighboring cells, which are within a given repulsive radius of a newly divided cell. However, the drawback of this approach is that an additional overhead will be introduced into the system for performing the relaxation process on the cells and yet without any significant improvement in synthesized results. This is true if the division of a cell occurs in the region so rapidly within a given time frame of ΔT. One more step to optimize this process is to keep a record of all the interrupted regions within a given time frame ΔT and then relax them simultaneously. In terms of implementation, a region mentioned above means the polygons, which are located within the repulsive radius of a given cell.

Whenever a cell division happens, the neighbors of the polygon, where the new cell resides, are obtained. These polygons are added onto a *Relaxation List*, if they are not yet placed in the list. Those polygons, which are relaxed and already formed a stable spatial cell arrangement, will be removed from the *Relaxation List*.

Fig.10(a) shows the performance comparison of the *Global Relaxation* process with the proposed *Local Relaxation* process. *Global* identifies the global relaxation process. *Local_1* identifies the *Local Relaxation*, which is performed after each division event. *Local_2* identifies the *Local Relaxation*, which is optimized with the integration of a *Relaxation List*.

Local_1 shows an overhead at simulation step 38. This is because the increment of the total number of cells results in an increment of cell division events. In Fig.10(b), simulation times are compared for objects with different number of polygons. Both *Local_1* and *Local_2* have faster simulation time, while the simulation time of *Global* becomes longer when the number of polygons increased.

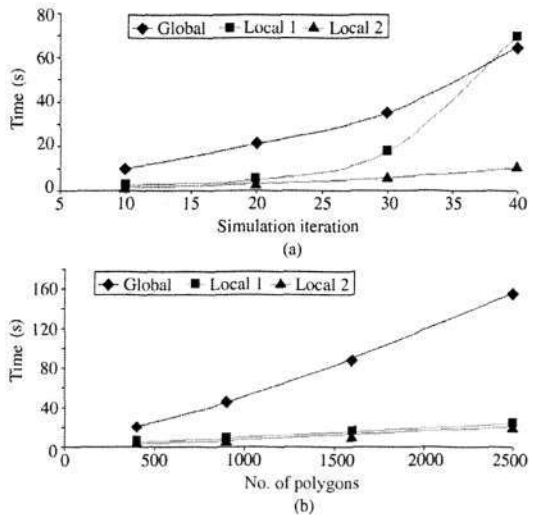


Fig.10. Performances of relaxation processes. (a) Comparison with different number of steps. (b) Comparison with different number of polygons.

Fig.11 shows the data flow diagram of a CM simulation with *Local Relaxation* implemented.

Fig.12 shows the simulation results of the relaxation processes. As one can observe, the results are almost

identical with each other. However, the simulation time with a Local Relaxation process is much faster.

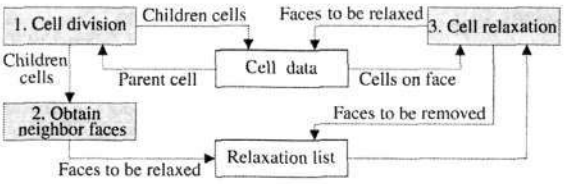


Fig.11. Data flow of simulation.

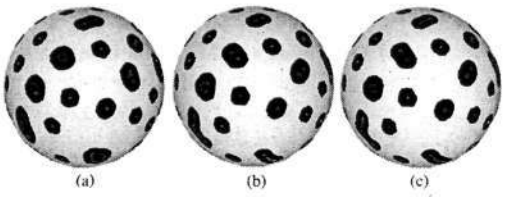


Fig.12. Simulation results of various relaxation. (a) Global relaxation. (b) Local relaxation after cell division. (c) Local relaxation optimized with relaxation list.

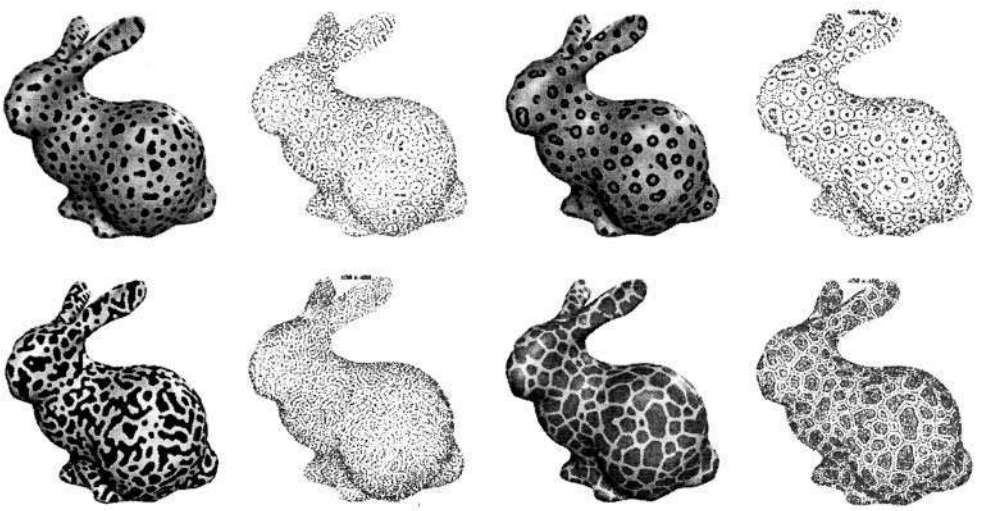


Fig.13. Results of various CM patterns and their cell arrangements on a Stanford Bunny.

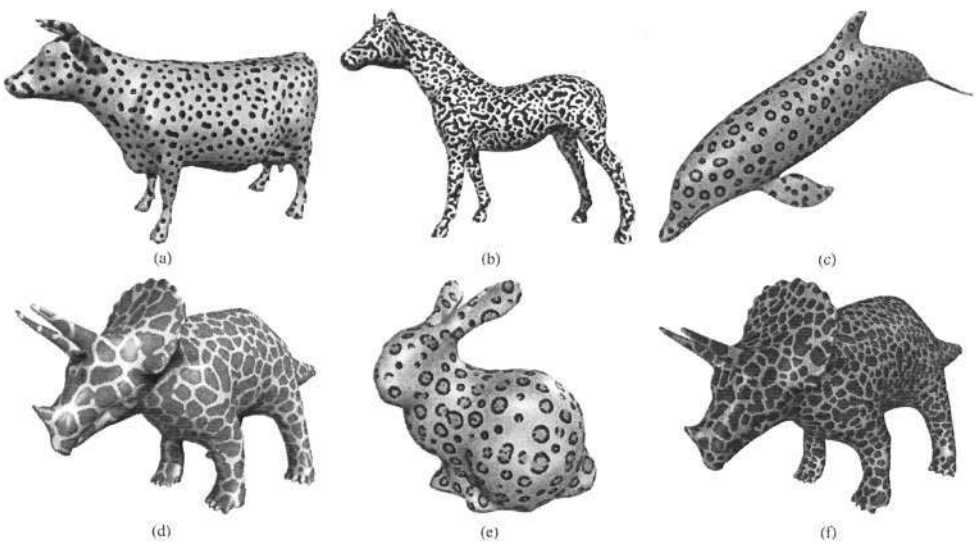


Fig.14. Results of various CM patterns. (a) Cow with cheetah spots (Polygon: 5,795, Cell: 26,500, T: 205.9s). (b) Horse with king cheetah markings (Polygon: 9,924, Cell: 44,628, T: 186.2s). (c) Dolphin with leopard rosettes (Polygon: 4,198, Cell: 32,321, T: 228.4s). (d) Bunny with jaguar rosettes (Polygon: 3,560, Cell: 33,245, T: 160.4s). (e) Triceratops with giraffe patches (Polygon: 5,660, Cell: 100,733, T: 473.7s). (f) Triceratops with giraffe patches (Polygon: 11,328, Cell: 151,538, T: 340.5s).

5 Results

The classical CM model computes the *Voronoi Diagram*, in order to decide the color of surface points according to which *Voronoi Polygon* the points belong to. A *CM Pattern Shader* is developed, which makes use of the CRPS. A search for the closest cell to a surface point is performed during rendering with our neighbor mapping function and distance approximation. This method is straightforward and easy to implement by using the previously derived techniques as discussed in Subsection 4.1.2. Fig.13 shows some of the CM patterns, which are generated with our system. The cell arrangements of each pattern are also shown. Fig.14 shows more examples where models with different complexity are used. Statistics for each of the examples are also given.

6 Conclusions

In this paper, the problems and issues pertaining to pattern synthesis on arbitrary polygonal surface have been identified and discussed. Relevant enhancements have been proposed and implemented to resolve some of these drawbacks. These include introducing the CRPS for computation effectiveness and optimally relaxing a local region for establishing a stable spatial arrangement of cells. The performance gain of these proposed techniques has been successfully verified with relevant test examples.

References

[1] Marcelo Walter, Alain Fournier, Daniel Menevaux. Integrating shape and pattern in mammalian models. In *SIGGRAPH'01: Proc. the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, USA, 2001, pp.317–326.

[2] Philip Ball. The Self-Made Tapestry: Pattern Formation in Nature. Oxford University Press, December 1998, pp.7–99.

[3] Turing A M. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc.* (B), 1952, 237: 37–72.

[4] Gierer A, Meinhardt H. A theory of biological pattern formation. *Kybernetik*, 1972, 12(1): 30.

[5] Meinhardt H, Klinger M. A model for pattern formation on shells of molluscs. *J. Theoretical Biology*, 1987, 126: 63–69.

[6] Deborah R Fowler, Hans Meinhardt, Przemyslaw Prusinkiewicz. Modeling seashells. In *SIGGRAPH'92: Proc. the 19th Annual Conf. Computer Graphics and Interactive Techniques*, ACM Press, New York, USA, 1992, pp.379–387.

[7] Koch A J, Meinhardt H. Biological pattern formation: From basic mechanisms to complex structures. *Reviews of Modern Physics*, October 1994, 66(4): 1481–1507.

[8] Painter K J. Models for pigment pattern formation in the skin of fishes. In Mathematical Models for Biological Pattern Formation, Maini P K, Othmer H G (eds.), Vol.121 of the IMA Volumes in Mathematics and Its Applications, Springer-Verlag, Berlin/Heidelberg, 2000, pp.59–82.

[9] Murray J D. Mathematical Biology II: Spatial Models and Biomedical Applications. 3rd Edition, Springer Verlag, January 2003, pp.141–191.

[10] Fabrice Neyret, Marie-Paule Cani. Pattern-based texturing revisited. In *SIGGRAPH'99: Proc. the 26th Annual Conf. Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, USA, 1999, pp.235–242.

[11] Alexei A Efros, William T Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH'01: Proc. the 28th Annual Conf. Computer Graphics and Interactive Techniques*, ACM Press, New York, USA, 2001, pp.341–346.

[12] Emil Praun, Adam Finkelstein, Hugues Hoppe. Lapped textures. In *SIGGRAPH'00: Proc. the 27th Annual Conf. Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, USA, 2000, pp.465–470.

[13] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH'02: Proc. the 29th Annual Conf. Computer Graphics and Interactive Techniques*, ACM Press, New York, USA, 2002, pp.665–672.

[14] Cyril Soler, Marie-Paule Cani, Alexis Angelidis. Hierarchical pattern mapping. In *SIGGRAPH'02: Proc. the 29th Annual Conf. Computer Graphics and Interactive Techniques*, ACM Press, New York, USA, 2002, pp.673–680.

[15] Jingdan Zhang, Kun Zhou, Luiz Velho *et al.* Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graphics (TOG)*, 2003, 22(3): 295–302.

[16] Greg Turk. Texturing Surfaces Using Reaction-Diffusion [Dissertation]. University of North Carolina, 1992.

[17] Bard J B L. A model for generating aspects of zebra and other mammalian coat patterns. *Journal of Theoretical Biology*, November 1981, 93(2): 363–385.

[18] Andrew Witkin, Michael Kass. Reaction-diffusion textures. In *SIGGRAPH'91: Proc. the 18th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, New York, USA, 1991, pp.299–308.

[19] Pankaj K Agarwal, Sariel Har-Peled, Meetesh Karia. Computing approximate shortest paths on convex polytopes. In *SCG'00: Proc. the 16th Annual Symp. Computational Geometry*, ACM Press, New York, USA, 2000, pp.270–279.

[20] Sariel Har-Peled. Approximate shortest paths and geodesic diameters on convex polytopes in three dimensions. In *SCG'97: Proc. the 13th Annual Symp. Computational Geometry*, ACM Press, New York, USA, 1997, pp.359–365.

[21] David Eberly. Rotation representations and performance issues. http://www.geometrictools.com/Documentation/RotationIssues.pdf, January 2002.

[22] Greg Turk. Generating random points in triangles. In *Graphics Gems*, Andrew S Glassner (ed.), Boston: Academic Press, 1990, pp.24–28.

How Jiann Teo received the B.Eng. degree in applied science (computer engineering) from Nanyang Technological University, Singapore. He is now doing his M.Eng in Center for Graphics and Imaging Technology, Nanyang Technological University, Singapore. His research interests include computer graphics and computer animation.

Kok Cheong Wong is the Director of Sparky Animation Pte Ltd and an associate professor in Nanyang Technological University, Singapore. Kok Cheong received his B.Eng. in electronics and electrical engineering and his Ph.D. in computer science from University of Surrey, UK in 1989 and 1993, respectively. He won local and overseas awards in both technical journal publications and directing short computer animated films. His research interests include computer animation and game development.

Optimization of Clonal Mosaic Model with Canonical Reference Plane and Local Relaxation

How Jiann Teo, Kok Cheong Wong
School of Computer Engineering
Nanyang Technological University, Singapore
E-mail: hjteo@ntu.edu.sg, askcwong@ntu.edu.sg

Abstract

In this paper, an effective system for synthesizing animal skin patterns on arbitrary polygonal surfaces is introduced. The system uses the Clonal Mosaic (CM) *model [3], which simulates cells' reactions on arbitrary surface. A* CM *simulation forms regulated spatial arrangement of cells, which is comparable with natural patterns observed from animal skin. A typical* CM *simulation process incurs high computational cost, where the distances among cells across a polygonal surface are measured and the movements of cells are constrained on the surface. An approach to transform each of the original 3D geometrical planes of the surface into its* Canonical Reference Plane Structure (CRPS) *is proposed in this paper. This structure helps to simplify a 3D computational problem into a more manageable 2D problem. Furthermore, the concept of* Local Relaxation (LRX) *is developed to enhance the relaxation process for a typical* CM *simulation.*

Keywords: Clonal Mosaic, Procedural Texture

1. Introduction

Our work is inspired by a procedural texture method, namely, *Clonal Mosaic (CM)* model [3]. This model works with a large number of points (cells), which are distributed over a piece of geometry. Cell's movement and division are carefully controlled, in order to form a regulated spatial arrangement of cells. The resulting arrangement of cells, once associates with a Voronoi diagram, shows a tessellation of cell regions, which are visually identical with color patches on animal skin.

Generation of a *CM* patterns includes the simulation of cell division, cell mutation and cell repulsion. A *CM* cell is represented as a point in a given domain. Several types of cell, where each of the types is responsible for the synthesis of a single color appearance of the entire texture pattern, are introduced into the system.

All the cells in the system divide themselves into two according to their division rate. The progenies inherit all attributes of their parent if no cell mutation happens. Whenever a division event happens, a newly produced cell repulses its neighboring cells. Repulsion computation is needed to displace the cells accordingly. The process to compute repulsion among cells and displace those cells according to their repulsive forces is named as a relaxation process.

Two techniques are successfully developed in order to overcome the drawbacks and limitations of the existing *CM* model. First, an intermediate spatial configuration of polygonal object, named *Canonical Reference Plane Structure (CRPS),* is proposed. This geometrical structure reduces the computational cost incurred during a simulation process. Next, the *Local Relaxation (LRX)* optimizes the simulation process by avoiding redundant computation of cell repulsions.

2. Proposed framework and techniques

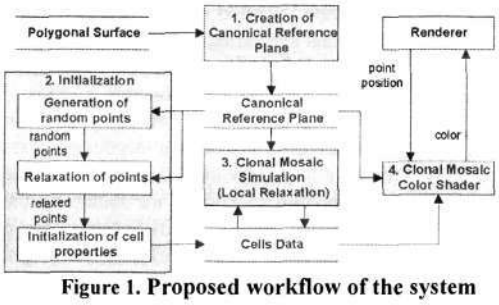


Figure 1. Proposed workflow of the system

Figure 1 shows the workflow of the proposed system. A polygonal surface is transformed into *CRPS*. This structure keeps all polygons and cells on a common plane, so that the subsequent computations for cells' reactions are highly simplified. A series of initialization processes take place in step 2. Points are randomly distributed over the geometry [2]. These points undergo a relaxation to form a regularly spaced points set. Lastly, each of the points is associated with a cell type, which defines color properties, repulsiveness, division rate and mutation rate of cells. In step 3, the *LRX* substitutes the classical relaxation process, which is performed globally. *LRX* avoids redundant computation of cell repulsions, hence reduces the simulation time.

2.1 Canonical Reference Plane Structure

CM simulation relies heavily on repulsion computation, where repulsive forces among cells are obtained. The problem is complicated as the shortest path and distance among cells are computed on an arbitrary polygonal surface. Accurate techniques, which compute shortest path across the surface, require extensive computation. Other technique [3] provides a mapping function, which maps cells on neighboring polygons onto the common plane in which a master polygon resides. The distances among cells are then approximated by measuring their Euclidean distances on the plane. This method involves massive computation of rotations and cell mappings. For an optimum choice for less memory and faster computation [1], the rotation information is stored in quaternion and is converted into matrix whenever a rotation is performed. It requires storage of four floats as quaternion, twelve additions and twelve multiplications for matrix conversion, six additions and nine multiplications to rotate each point.

A data structure, which transforms all polygons onto a common plane, is proposed. As all polygons are on a common plane, the original problem is simplified from a 3D computational problem into a more manageable 2D problem. Figure 2(a) shows a polygonal sphere. Figure 2(b) shows that all polygons of the sphere are transformed in such a way that they are aligned with the XZ plane. By ignoring the Y coordinate, all polygons are now residing on a 2D domain (See Figure 2(c)). The XZ plane is called as a *Reference Plane* and the rotated polygons are called as *Reference Polygons*. Polygons in this 2D domain are overlapped with each other and have their normals pointing towards Y-Axis.

A neighborhood is formed by transforming neighbors of a polygon, in which a target cell is residing, during a repulsion computation. Euclidian distances among cells on this neighborhood are taken as the approximated distances. Figure 2(d) shows several *Reference Polygons* and Figure 2(e) shows a neighborhood, which is formed after neighbor mapping. The shaded polygon identifies the polygon where a target cell resides.

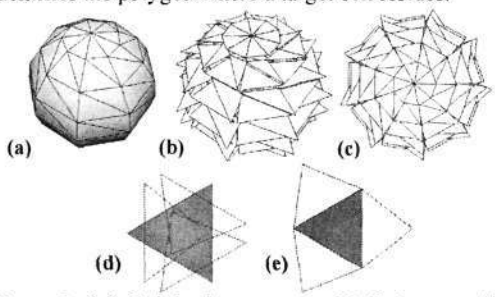


Figure 2. (a) Original geometry (b) Polygons align with XZ plane (c) Polygons in CRPS on a 2D domain (d) Overlapped polygons (e) A neighborhood

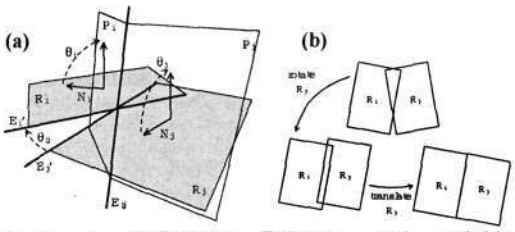


Figure 3. Reference Polygon and neighbor transformation (a) Creating Reference Polygon (b) Transforming Reference Polygons

Figure 3(a) shows how the 2D transformation information is derived. Given a polygon P_i and its neighbor P_j, rotation $R(\theta_i)$ and $R(\theta_j)$ are found to give *Reference Polygons* R_i and R_j. Their sharing edge E_{ij} is then rotated to E'_i and E'_j respectively by rotation $R(\theta_i)$ and $R(\theta_j)$. An angle θ_{ij} between E'_i and E'_j is used to rotate R_j to an orientation, which aligns with R_i. Next, a translation A is needed to connect R_i and R_j together (See Figure 3(b)). The rotation and translation required to map a *Reference Polygon* onto a neighborhood can be represented by a 2D transformation matrix F_{ji}:

$$F_{ji} = \begin{bmatrix} \cos\theta_{ij} & -\sin\theta_{ij} & A_x \\ \sin\theta_{ij} & \cos\theta_{ij} & A_y \\ 0 & 0 & 1 \end{bmatrix}$$

where

$$A_x = -M_{jx}\cos\theta_{ij} + M_{jy}\sin\theta_{ij} + M_{ix}$$
$$A_y = -M_{jx}\sin\theta_{ij} - M_{jy}\cos\theta_{ij} + M_{iy}$$

M_i and M_j denote the 2D position of a vertex, which connects both R_i and R_j in the original 3D domain. Instead of storing the 3×3 matrix in the memory, only the value of $cos\theta_{ij}$, $sin\theta_{ij}$, A_x and A_y are stored for each pair of neighboring polygons. A cell can be transformed to a frame at any time by using this transformation information between that pair of neighbors. The transformation equation for a cell is given as follows:

$$P'_x = P_x\cos\theta_{ij} - P_y\sin\theta ij + A_x$$
$$P'_y = P_x\sin\theta_{ij} + P_y\cos\theta ij + A_y$$

Memory storage of four floats is needed and the cost for transforming a cell is reduced to four additions and four multiplications. Figure 4 shows the performance comparison. "3D" identifies the data from the original model, where polygons and cells are rotated with quaternion and matrices. "*CRPS*" identifies the data from our system, where polygons and cells are transformed in the domain of the *Reference Plane* with 2D transformation matrices. When the number of cells increases, "*CRPS*" has introduced a relatively small increment in its computation time than the "3D" approach, as the overhead spent in transforming a cell is lighter than the original model.

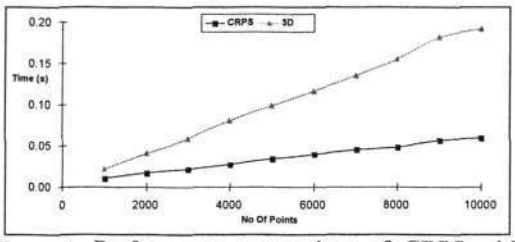


Figure 4. Performance comparison of CRPS with original 3D rotation computation

2.2 Clonal Mosaic Simulation

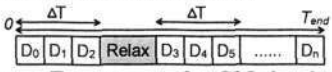


Figure 5. Event queue for CM simulation

The *CM* simulation process simulates cell division, cell mutation and cell repulsion on a surface. Division and mutation reactions are realized through an event queue. Cells are scheduled for division according to their division rate. Cell mutation happens when a new cell is created and is controlled by its mutation rate.

A series of division events D_i are queued in a heap structure (See Figure 5). Each of the division events has a time value T_{Di} , which is the time for a cell to divide itself. All division events within a given time frame ΔT are processed before a relaxation process is performed. Each newly divided cell will be introduced back to the queue for future division. The simulation keeps running until the ending time T_{end} is reached.

The major computational load incurred in a *CM* simulation is the computation for cell repulsion. The classical relaxation method relaxes all cells on a surface together in each interval (called as *Global Relaxation (GRX)*). Hence, there exists redundant computation for those regions, where relaxed spatial arrangement of cells is already obtained. In general, a stable spatial cells arrangement in a region is only interrupted when a new cell is introduced into the region. As a result, by relaxing cells only on these local region will effectively reduce the computational load. This proposed method is named as a *Local Relaxation (LRX)*.

LRX relaxes a small region, which is interrupted by a cell division event. The approach relaxes neighboring cells, which are within a given repulsive radius of a newly divided cell. However, the drawback of this approach is that an additional overhead will be introduced. This is true if the division of a cell occurs in the region so rapidly within a given time frame of ΔT. We further optimize this process by keeping a record of all interrupted regions within a given time frame ΔT and then relax them together. In terms of implementation, a region mentioned above means the polygons, which are located within the repulsive radius of a given cell.

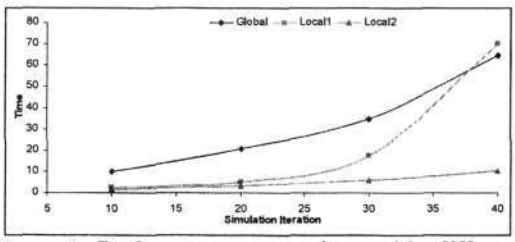


Figure 6. Performance comparison with different number of simulation step

Figure 6 shows the performance comparison of the original *GRX* process with the proposed *LRX*. *Global* identifies the *GLX*. $Local_1$ identifies the *LRX*, which is performed after each division event. $Local_2$ identifies the *LRX*, which performs a collective relaxation within ΔT. $Local_1$ shows an overhead at simulation step 38, as increment of total number of cells results in an increment of cell division events.

Figure 7 shows some of the CM patterns, which are generated with our system. The cells arrangements of each pattern are also shown.

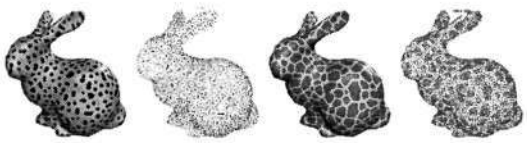


Figure 7. Result of various CM patterns and their cells arrangements on a Stanford Bunny

3. Conclusions

In this paper, the problems and issues pertaining to pattern synthesis on arbitrary polygonal surface have been identified and discussed. Relevant enhancements have been proposed and implemented to resolve some of these drawbacks. These include introducing the *CRPS* for computation effectiveness and optimally relaxing a local region for establishing a stable spatial arrangement of cells. The performance gain of these proposed techniques have been successfully verified with relevant test examples.

4. References

[1] D.Eberly. Rotation representations & performance issues. http://www.magic-software.com/Documentation.html, January 2002. Magic Software, Inc.

[2] G.Turk. Generating random points in triangles. In A.S.Glassner, editor, Graphics Gems, pages 24–28. Boston: Academic Press, 1990.

[3] M.Walter, A.Fournier, & D.Menevaux. Integrating shape & pattern in mammalian models. Proceedings of the 28th Annual Conference on Computer Graphics & Interactive Techniques, pages 317–326. ACM Press, 2001.