UNIVERSITY OF SOUTHAMPTON

ELEC6007 Final Report: Morphogenesis and Skin Pattern Generation for 3D Models

by

Robert Mills

David Newman

Simon Smith

Qing Yan Zhang

Faculty of Engineering, Science and Mathematics

School of Electronics and Computer Science

January 28, 2005

ABSTRACT

This report firstly examines how Morphogenesis can be interpreted through the use of computer modelling, concentrating on reaction-diffusion equations and how they can be used to simulate real-world animal skin patterns. The report then focuses on the work done by Greg Turk in his 1991 paper "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion" [19]. The work accomplished in Turk's paper is discussed; implementations based on his work in one, two and three dimensions are constructed and analysed for this report. This analysis compares the implementations' results to both those achieved by Turk and real-world skin patterns. The analysis also attempts to determine how reaction-diffusion models proposed by Turk and others function, in particular, how changing parameters affects the patterns produced. The report concludes with a discussion of whether reaction-diffusion is an appropriate model for the natural phenomenon of animal skin pattern formation. Finally, further work is proposed to extend the current implementations, as these provide a solid basis to construct a generator for more complex reaction-diffusion models.

Contents

Chapter 1

Introduction

1.1 Morphogenesis

The fourth edition of American Heritage of the English Language dictionary describes Morphogenesis as the "Formation of the structure of an organism or part; differentiation and growth of tissues and organs during development." [6]

Alan Turing was one of the first to consider the mathematical theory behind Morphogenesis. Turing produced a paper in 1952 [18] and several papers that remained unpublished until 1992 on the topic of Morphogenesis. These papers concentrated on how mathematics could be applied to naturally occurring phenomena such as 'Fibonacci Phyllotaxis' which concerns itself with explaining patterns such as the formation of the seed head of sunflowers or the spiral patterns that occur round the stems of Euphorbia. [17] This phenomenon is described as 'Fibonacci Phyllotaxis' because the Fibonacci number series defines these patterns. is' because the Fibonacci number series defines these patterns.

The main phenomenon that Turing concerned himself was that of 'Reaction-Diffusion', (R-D), which is a mathematical attempt to explain things such as animal skin patterns. An analogy that Turing used to describe this mathematical theory has become known as the 'missionaries and cannibals' model. This analogy defined the two basic components, inhibitor as missionaries and activator as cannibals. This model allowed only the cannibals to reproduce but two missionaries could convert a cannibal, (to a missionary). If this is considered over a 2D map of an island, this would generate pockets of cannibals surrounded by missionaries, which could be compared to the spotty pattern observed on a cheetah. Turing proposed that modifications to the rules of the model would produce other patterns such as stripes.

1.2 Purpose of Report

The course is designed to give a deep level of understanding in one area of Biologically-inspired Computing whilst gaining exposure to several subjects. The project aims to research an area, find a key paper, and replicate the work done in this paper. By in depth implementation and investigation, we achieve a good understanding of the topic. Short presentations inform other course-members of our research, and vice versa, building a good exposure to the world of biologically inspired computing.

This report details the work done investigating morphogenesis, which is realised through literary review, and the emulation of the work described in a key paper, selected from the reviewed literature. The key paper in this instance is work done by Greg Turk, and is described in Section 2.3. The project implementation also extends past Turk's progress, which is explained in chapter 4.

Chapter 2

Literature Review

2.1 Turing's Initial Paper

As already stated the dictionary definition of Morphogenesis is "The formation of the structure of an organism or part". [6] In detail, human beings begin as a single cell and through many biological processes develop into fully grown adults consisting of around 10^{15} cells. These cells of many different types are arranged into extremely complicated structures to perform a wide range of vital functions in the body.

However, Alan Turing, one of the most creative thinkers of the 20th century, used this term mostly in the sense of pattern formation rather than this more general meaning. In 1952, he published a paper that explained the chemical basis of morphogenesis in order to discuss a possible mechanism by which the genes of a zygote may determine the anatomical structure of the resulting organism [18]. In this paper, he proposed a Reaction-Diffusion system that explains the pattern formation using morphogenesis. He asked a fundamental question; how do patterns appear in a region which has nothing to serve as a template? [18] It is suggested that a system of chemical substances, called morphogens, reacting together and diffusing through a tissue, is adequate to account for the main phenomena of morphogenesis [18].

Alan Turing's paper has been widely cited in the mathematical theory of biological pattern formation [16], and his theory is extended and applied to many other pattern generation techniques.

2.2 Current Status

2.2.1 Morphogenesis Research Overview

Morphogenesis is one of the major outstanding problems in the biological sciences. It concerns a fundamental question of how biological form and structure are generated from a single cell. Morphogenesis covers a wide range of biological processes. It concerns adult as well as embryonic tissues, and includes an understanding of the maintenance, degeneration, and regeneration of tissues and organs as well as their formation [13]. Morphogenesis also addresses the problem of biological form at many levels, from the structure of individual cells, through the formation of multi-cellular arrays and tissues, to the higher order assembly of tissues into organs and whole organisms [13].

2.2.2 Morphogenesis and Skin Pattern Generation after Turing

One of the central issues in morphogenesis concerns the formation of skin pattern. In 1952, Alan Turing proposed a theory for pattern formation in which he showed that a system of chemicals could evolve spontaneously into a spatial pattern. He then hypothesized that this served as a pre-pattern for subsequent cell differentiation [18]. In 1974, Catmull introduced texture mapping, as a method of adding visual richness to a computer generated image without changing the geometry [5]. In 1977, Jonathan Bard firstly hypothesized a mechanism for the production of zebra stripes in the three species of extant zebra [3]; and later in 1981, Murray showed that the chevrons at the base of the zebra's limbs is the shape expected by the overlapping of two Turing-type reaction-diffusion systems [11]. In 1983, Oster, Murray and Harris proposed an alternative theory for morphogenesis, in which mechanical effects induced cell motion and cells formed clumps which then differentiated. The resulting system of equations described the evolution of cell density, extracellular matrix (ECM) displacement and density is a highly complex system of coupled non-linear partial differential equations of different types [1].

In 1991, two pieces of texture synthesis works using R-D were proposed. Witkin and Kass demonstrated that a wide variety of patterns can be created using variations of one basic R-D equation [20]. In detail, they showed the importance of anisotropy by introducing a rich set of new patterns that are generated by anisotropic reaction-diffusion [21]. Moreover, they also demonstrated how reaction-diffusion systems can be simulated rapidly using fast approximations to Gaussian convolution [19]. In the same year, Turk showed the technique consisting of R-D and mesh direct mapping methods to generate texture on arbitrary surfaces.

2.2.3 Morphogenesis used in Reaction-Diffusion

How do the cells of an embryo arrange themselves into particular patterns? Turing proposed that this could be controlled by a set of simple partial differential equations that describe the concentration of some chemical substances, morphogens as he called them, that can diffuse amongst the cells and also be produced within the cells. One morphogen, the Inhibitor, suppresses the production the other, the Activator. The levels of each chemical varies over time due to the reaction (production of morphogens) and diffusion, hence the name. With the correct parameters a steady-state can occur and hence a pattern is formed. Such chemical systems are known as Reaction-Diffusion systems.

2.3 Key Paper

"Generating Textures on Arbitrary Surface Using Reaction-Diffusion" written by Greg Turk was chosen as the key paper. This paper emphasizes on two methods that can be applied at different stages of pattern generation. The first one is a procedural method which is used for texture synthesis, whereas the second one focuses on how to fit a texture to a random surface. Although these two methods can be used separately, there are advantages to in using both together in terms of generating natural textures on complex models [19].

In detail, the first method described in this paper, known as Reaction-Diffusion, is a chemical mechanism for pattern formation; it introduces how two or more chemicals that diffuse across a surface and react with one another can be form stable patterns [19]. This paper firstly shows R-D implementations in 1D and 2D; by varying the parameters,forms different patterns. The paper then extends the method by cascading multiple R-D systems, to generate more complex patterns such as the ones seen on giraffes.

Having formed the patterns, they then need to be mapped to the surface of an object. In order to avoid distortion of the textures especially on 3D models, the paper introduces the second method of generating a mesh over the surface of a polyhedral model that can be used for texture synthesis [19]. R-D can then be processed directly on the mesh.

Finally, this paper suggests some possible on cascading techniques and other pattern creation methods.

2.4 Texture Mapping Methods

There are three main methods that can be applied for texture synthesis - texture placement, procedural texture synthesis and texture synthesis from samples.

The first method takes an existing texture from a rectangular pixel array and wraps it onto a surface [20]. This method aims to minimize noticeable seams between patches and stretching and distortion of the pattern. The lapped texture, proposed by Praun et al., is an extension of texture placement; it takes one or more irregularly shaped texture patches and place many copies of the patches in an overlapping fashion over a surface [12]. This technique produces excellent results since the nature of the overlapping patches is often unnoticeable (See Figure 2.1) [20].

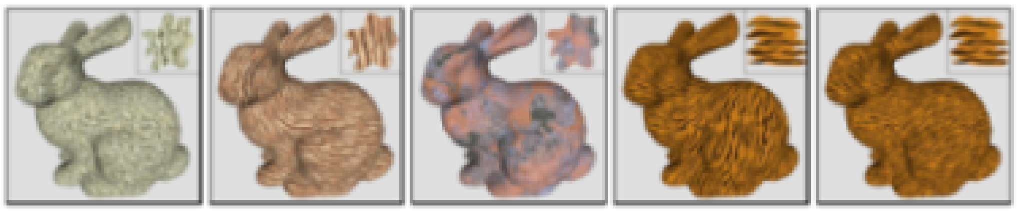
FIGURE 2.1: Four different textures pasted on the bunny model. The last picture illustrates changing local orientation and scale on the body.

The second technique is procedural texture synthesis; a great strength of it is that each new method can be used in conjunction with already existing functions. Reaction-Diffusion is one of the most representative methods here. In addition, R-D is used in different biological mechanisms proposed by different people to achieve different goals. For example, as mentioned before, Turk demonstrated the R-D proceeded directly on arrays of cells that have been placed over a polygonal surface in order to avoid noticeable seam and distortion, whereas Fleischer et all. showed how interacting texture elements on a surface to create texture geometry such as scales and thorns [7].

Since one pitfall of procedural texture synthesis is that it requires programmer keep writing and testing code until the output has the right 'look' [20], the third method, texture synthesis from samples, was developed. It allows the user to supply a small patch of the desired texture and to create more texture that looks similar to this sample [20]. A hierarchical mesh would be created using this approach.

By comparing with these three techniques, Reaction-Diffusion, one of the closest analogies to real-world biology, was finally chosen since this project is after biologically motivated solutions rather than graphical approaches. More specifically, Turk's paper is selected as the key literature since it provides not only the R-D method but also a direct mapping mechanism, which avoids distortion and noticeable seams.

Chapter 3

Goals

The first goal of the project was to perform a investigation into how reaction-diffusion works in one and two dimensions. This required a study of how reaction-diffusion systems evolve across one dimension, the number of iterations required to produce stability and the comparison between the levels of activator and inhibitor after stabilisation.

The main goal of this project was to replicate the work of the key paper, Greg Turk's "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion." [19] This goal has several sub-goals. These sub-goals include firstly replicating reaction-diffusion in two dimensions to produce patterns that both resemble real-world skin patterns such as zebra stripes and cheetah spots and the patterns produced by Turk.

The second sub-goal in emulating Greg Turk's work was to use procedural texture synthesis to directly map reaction-diffusion patterns onto three-dimensional models. To achieve this goal several appropriate models needed to be found. Methods and parameters for procedural textural synthesis were required, in order to emulate both the work done by Turk and resemble real-world examples. It was decided to concentrate on emulating two skin patterns, zebra stripes and cheetah spots and this left the emulation of cascading patterns, such as cheetah spots, as a time-permitting goal.

Chapter 4

Design and Implementation

4.1 Introduction to Reaction-Diffusion

Reaction-Diffusion (R-D) is the foundation upon which Turk's 1991 paper is built. R-D is where two or more chemicals diffuse over a surface, and react with each other. Diffusion is usually at unequal rates and results in stable patterns such as spots or stripes being produced. Turk's paper considers a number of different systems, as does this report. However, to understand the system it is easiest to inspect a two-chemical system first; the more complex systems can then be introduced as required.

The general equations for two chemical R-D are given in eqn 4.1.

$$\frac{\partial I}{\partial t} = F\left(I, A\right) + D_I \nabla^2 I \qquad \frac{\partial A}{\partial t} = G\left(I, A\right) + D_A \nabla^2 A \tag{4.1}$$

Where the two chemicals involved are the activator (A), and the inhibitor (I). These two equations define the change in chemicals, both have a similar structure. The first term is the reaction term, which is a function dependent on both the activator and the inhibitor. The second is the diffusion term which only depends on the 2nd order differential of the relevant chemical and its diffusion coefficient. Turing [18] proposed a specific implementation of this system, which is shown in equations 4.2 and 4.3.

$$\nabla I_j = s\left(16 - I_j A_j\right) + D_I\left(I_{j+1} - 2I_j\right) \tag{4.2}$$

$$\nabla A_j = s\left(I_j A_j - A_j - \beta_j\right) + D_A\left(A_{j+1} + A_{j-1} - 2A_j\right) \tag{4.3}$$

Turing worked with very early digital computers (Manchester Mark 5) and realised that he could represent a line of biological cells with a discrete data structure. Thus, his proposed equations considered a digital approximation to a second order differential, known as the

Laplacian. It can be derived from the difference between two first order differences:

$$f''(x) \cong f'(x) - f'(x+1) \qquad f''(x) \cong -f(x) + 2f(x+1) - f(x+2) \tag{4.4}$$

Which can be represented as a convolution template in Figure 4.1 It should be noted that

-1	2	-1

Figure 4.1: Laplacian for 1D Reaction-Diffusion

the operators coefficients sum to zero; in a region of continuity it would be expected that no diffusion would occur, so this operator gives a correct response. Equations 4.2 and 4.3 are used to build a system to experiment with 1D systems, as described in section 4.2.

4.2 Initial Matlab Implementation

4.2.1 Matlab

Matlab was thought to be a suitable language to implement investigative work for a number of reasons. These included the good environment for numerical computation, a large number of built in utilities and the ease of displaying functions. One further aspect is that Matlab lends itself to multi-dimensional processing. Thus, Matlab was chosen for the initial exploration, as described in the following sub-sections.

4.2.2 Progress

Source code was written to implement for the two-chemical R-D system described by equations 4.2 and 4.3, to model a line of cells. The implementation use a substrate, β, is initialised with some random variation in order to generate patterns which are not too uniform. The code was written so that the parameters for cell count, initial values for activator and inhibitor, and the reaction speed coefficient could be set without direct code editing.

The investigation used parameter values similar to those suggested in Turks paper [19]. A progression through 2800 iterations is shown in Figure 4.3; (only the activator levels are plotted here). As expected, the system took approximately 1200 iterations to stabilise and displays clear peaks and troughs, representing stripes along the line of cells. Figure 4.2 shows the values for both the inhibitor and the activator along the line of cells at 2800 iterations. It is clearly visible that the two chemicals have complementary values; where the inhibitor has a high concentration, the activator cannot exist, and in regions of high activator concentration, no inhibitor is present. If the activator represents a black pigment and the system is constructed upon a white substrate, we would see a line with black stripes centred at cells 26, 51, and 78, with white stripes in between.

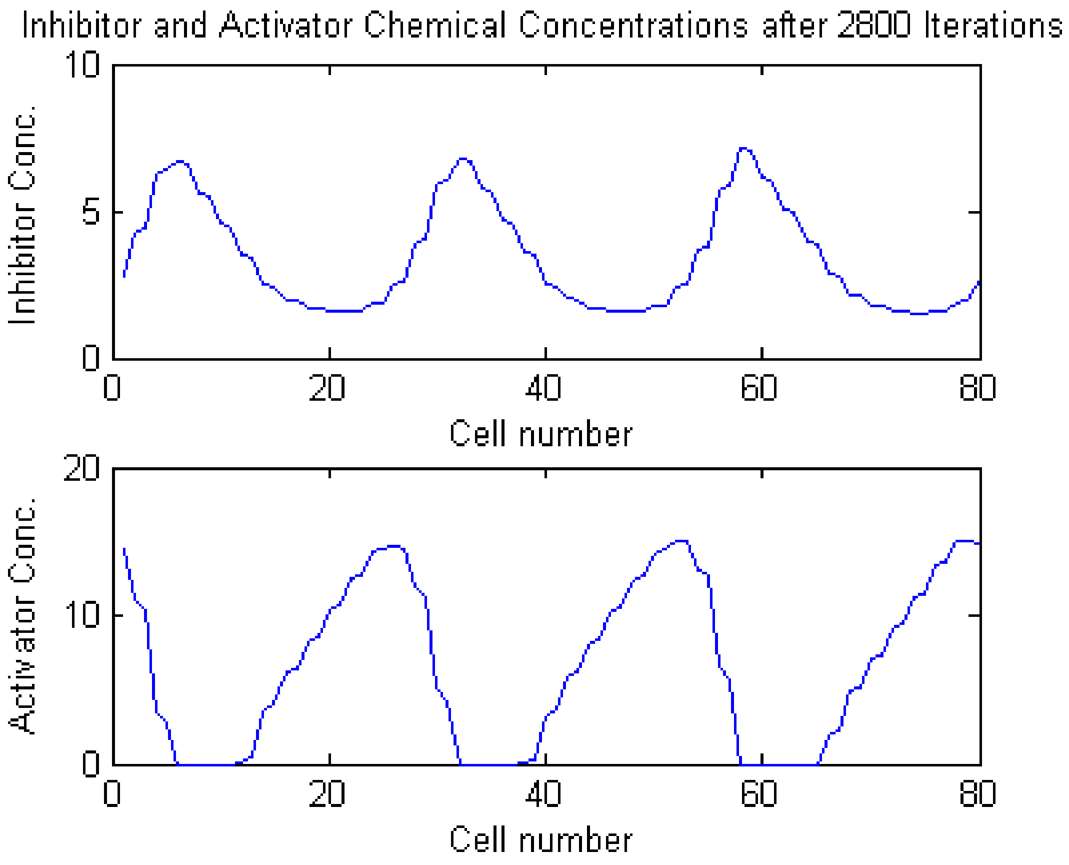

Figure 4.2: 1D Final Concentrations of Inhibitor and Activator

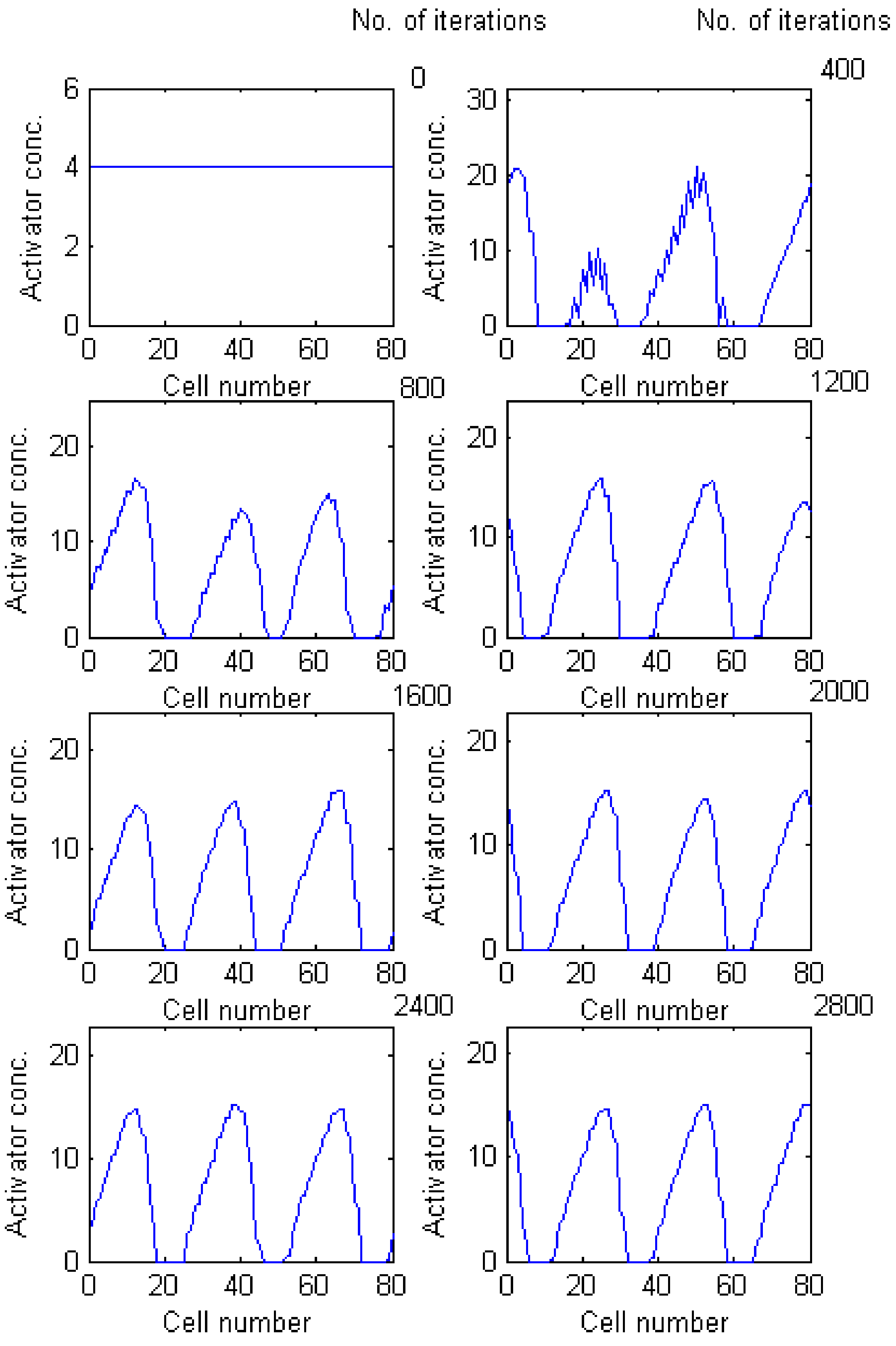

FIGURE 4.3: 1D Reaction-Diffusion of Activator Over Time

4.2.3 Extension to 2D

A 2D implementation in Matlab was started but the execution time was very slow. Since Matlab is interpreted rather than compiled, it is quick for prototyping but slow upon execution. This was prohibitive to development, so the group looked for a replacement language, which is discussed in 4.3.2.

4.3 2D Investigation

4.3.1 Mathematical Theory

As described in Section 4.2 a simple Laplacian is used to perform reaction-diffusion, Figure 4.1, this simple Laplacian can be expanded to consider two dimensions. The one-dimensional example considers that each cell has only two neighbours[1], when this is scaled up to two-dimensions each cell then has four neighbours. This is achieved by combining one horizontal with one vertical 1D Laplacian (see Figure 4.4). This produces diffusion coefficients in both the horizontal and vertical directions but the actual reaction terms remain unchanged. Like the 1D Laplacian the sum of all the values totals zero to maintain stability in the system.

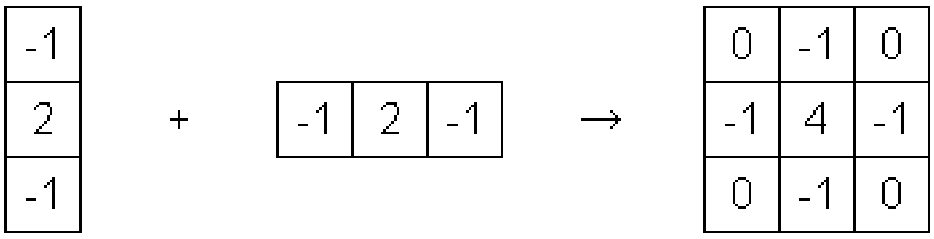

Figure 4.4: Laplacian for 2D Reaction-Diffusion

4.3.2 Pros of OpenGL

C with OpenGL was eventually chosen over Matlab to implement reaction-diffusion in 2D. Although the Matlab 1D R-D code was perfectly acceptable, initial trials with 2D R-D in Matlab executed very slowly; One 2D texture generation would take at least 20 minutes. Due to such huge increases in execution time, it was envisaged that the graphical processing required for the 3D R-D method, would lead to unacceptably high execution times. New tools were therefore needed to take the place of Matlab.

OpenGL, the premier environment for developing portable, interactive 2D and 3D graphics applications, was selected since it is compiled not interpreted resulting much faster processing. Many 2D R-D examples were found that had been implemented in C, suggesting that

[1]A neighbour is considered to be another cell that share an edge with the original cell.

C is a more suitable development environment for this kind of task. The 2D examples found could also be used as a basis for a 3D implementation. C has large amounts of freely available source code and therefore it was more likely (than with Matlab) to find pre-written applications to perform task such as 3D model readers.

4.3.3 Discussion of 2D Implementations Found

The world-wide web was scoured to find 2D implementations in OpenGL, which could be either have their source code or their parameters for reaction-diffusion modified to generate patterns that emulate Greg Turk's work and real-world patterns. Three significant implementations were found at [15].

The first implementation found was an attempt to emulate some of the work Greg Turk did. This implementation was capable of producing anything from zebra stripes to cheetah spots and various patterns in between. Figure A.2 shows some of the patterns generated. In this particular implementation five parameters determine reaction-diffusion,j these parameters can be found in appendix B.

The second implementation found was similar to the first but it allowed the seeding of cell as either inhibitor or activator through mouse clicking. In theory this should have created different pattern depending on which cells were seeded, unfortunately this did not seem to happen, this may have been because the diffusion speed was too quick and because the cells had to be seeded whilst reaction-diffusion was occurring. The technique used by this implementation may however prove useful when it comes to 3D direct-mapping when trying to affect the patterns generated on certain areas of a three-dimensional model such as the head. For this type of seeding to work effectively the seeds would probably have to be selected before reaction-diffusion begins.

The final implementation found used a cascading technique. This is where an initial reaction-diffusion the same as in implementation one takes place. After a certain number of reaction-diffusion cycles, (initially 20,000), all the inhibitor cells are frozen, all the activator cells are reset to neutral and a second reaction-diffusion occurs within the areas of activator cells. Reaction-diffusion using cascading can produce more than two colours as can be seen in Figure 4.5, where yellow is the primary inhibitor colour, black the primary activator and secondary inhibitor and a grey/yellow the secondary inhibitor, this produces a pattern similar to leopard spots. The techniques found in these three implementations can potentially all be adapted in the implementation of skin pattern mapping on three-dimensional models, to produce results that meet the goals of emulating Turk's work and resembling real-world animal skin patterns.

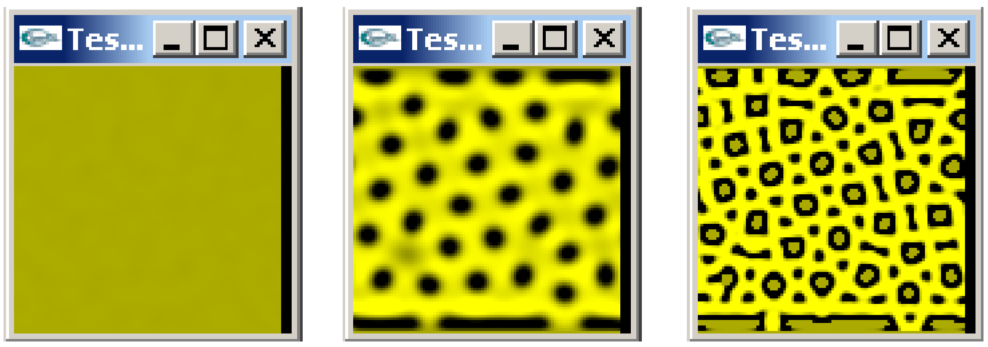

FIGURE 4.5: Reaction-diffusion using cascading

4.4 3D Mathematical Theory

The main goal of the project was to implement some kind of R-D system upon 3D objects, which requires further modification to the basic R-D equations introduced in section 4.1. Talking about three dimensions is perhaps slightly misleading since the reaction-diffusion system only acts on the surface of an object, and the surface itself is only two dimensional. However, this notation is used to distinguish the work focused on surfaces of 3D models from the simpler 2D work.

The surface is made up of triangular polygons described by the 3D model (see section 4.5). Each of these is used to represent a cell, similar to the pixels in the 2D grids. However, since the cells are of variable size, there is no formal grid on which the vertices of each polygon lie. The cells have three neighbouring cells which share an edge, rather than the four in the Cartesian grid structure. These two aspects require a method for approximating the diffusion differently to the previous approach. The lengths of the edge were considered as a measure of the level of diffusion between two cells, which is illustrated in Figure 4.6. When considering cell C_{ij}, the neighbouring cell N_3 has a shorter edge shared with C_{ij} than N_1. This causes its diffusion weighting to be lower than that of neighbour N_1.

4.4.1 Five-Chemical Reaction-Diffusion System

Turk also used a reaction-diffusion system proposed initially by Meinhardt [10] that involved the interaction of five chemicals across a substrate. This system uses lateral activation and local exclusivity to create patterns of stripes. If random perturbations are introduced, random stripe patterns are formed, but if stripe initiator cells are used regular stripe patterns can be generated.

Meinhardt's five-chemical system uses mutual exclusivity of the chemicals g_1 and g_2 brought about by a common repressor r, in order to ensure the presence of one or the other stripe colour but never both. The lateral activation which ensures stripe formation is provided by the diffusible substances s_1 and s_2; these two chemicals provide long-range activation from one feedback system to the other. This system is governed by equations 4.5 to 4.9.

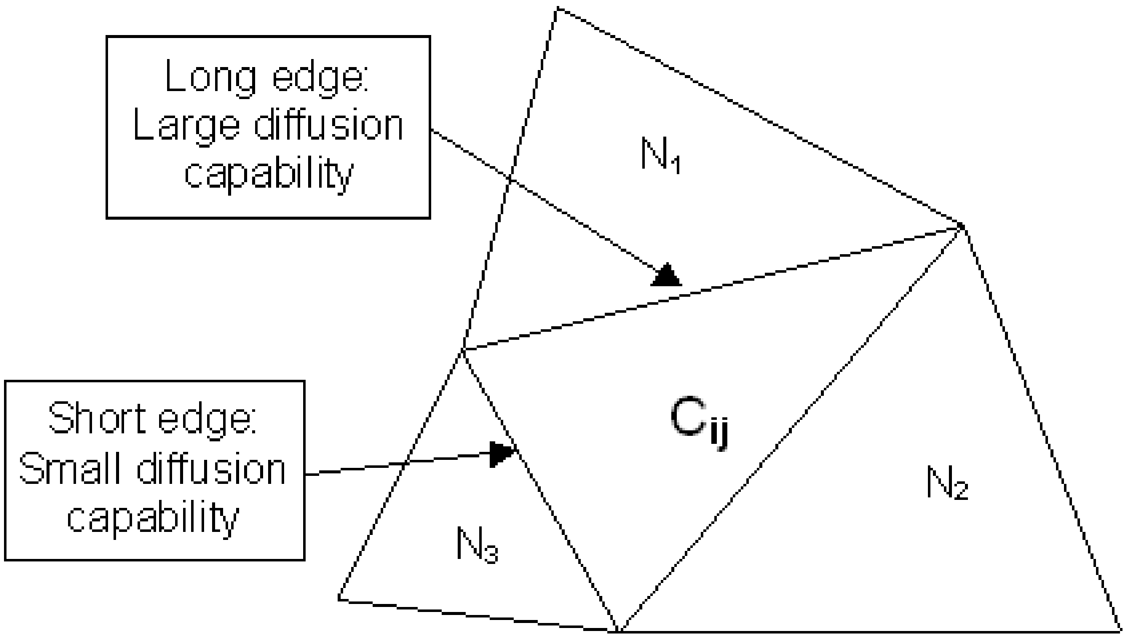
Long edge: Large diffusion capability
Short edge: Small diffusion capability
N1
Cij
N2
N3

FIGURE 4.6: Reaction-Diffusion in 3 Dimensions

$$\frac{\partial g_1}{\partial t} = \frac{c s_2 {g_1}^2}{r} - \alpha g_1 + D_g \frac{\partial^2 g_1}{\partial x^2} + \rho_0 \tag{4.5}$$

$$\frac{\partial g_2}{\partial t} = \frac{c s_2 {g_2}^2}{r} - \alpha g_2 + D_g \frac{\partial^2 g_2}{\partial x^2} + \rho_0 \tag{4.6}$$

$$\frac{\partial r}{\partial t} = c s_2 {g_1}^2 + c s_1 {g_2}^2 - \beta r \tag{4.7}$$

$$\frac{\partial s_1}{\partial t} = \gamma(g_1 - s_1) + D_s \frac{\partial^2 s_1}{\partial x^2} + \rho_1 \tag{4.8}$$

$$\frac{\partial s_2}{\partial t} = \gamma(g_2 - s_2) + D_s \frac{\partial^2 s_2}{\partial x^2} + \rho_1 \tag{4.9}$$

The relationship between the two g and s chemicals, can be thought of as analogous to the relationship between humans and plants. In this analogy, consider humans to be the chemical g_1, and plants to be the chemical g_2, then consider carbon dioxide is s_1, and oxygen is s_2. Humans require oxygen to respire and produce carbon dioxide as a waste product, which is poisonous to them. Plants need carbon dioxide for photosynthesis and produce oxygen as a waste product. Oxygen is not actually harmful to plants but for this analogy to work we assume it is. Therefore neither organism can exist without the other to produce its 'activator' chemical and break down its 'inhibitor'. The difference between this analogy and Meinhardt's system is the mutual exclusion brought about by the chemical r, in order to prevent the two organisms from existing in the same place.

4.5 3D Models

4.5.1 PLY

This is a simple object description file format for research work with polygon models. The files can exist in binary or ASCII formats - the ASCII format makes it easy to understand how the model is built up, whereas the binary format is more compact. It is made up of a header and a body, the former describing the information structure and content of the latter. Two attributes are standard in each PLY file: the vertices and faces. In addition, the format allows custom attributes to be defined by the user; the header must be modified to include details on the structure for these new attributes. Any attributes which are not expected by a later tool are ignored. [14]

4.5.2 3DS

This 3D object format is a proprietary format of AutoDesk who produce 3-D CAD software tools. [9] It is a binary format, made up of data blocks called chunks. Each chunk starts with an ID and a data length. The ID identifies the type of data in the current chunk, this is followed by the data. The format allows subordinate chunks to be included in this data, forming hierarchical structures. These are present if the data length is greater than that required for the particular data type. The structure requires all block types to be understood by the reader, and additional data requires new block types to be defined.

4.5.3 Model Choice

The 3DS format is widespread; many models are available both freely and commercially. The PLY format is less used but simpler to understand, process, and extend. The PLY format and supporting software[2], is freely available, due to the educational for its development. Thus, the PLY format was chosen for this project.

A shareware 3D tool was used to convert models into the PLY format in order to be compatible with 3D surface mapping. [8]

4.6 3D OpenGL Implementation

4.6.1 OpenGL 3D Rendering

The way in which objects and surfaces are rendered using the OpenGL GLUT toolkit is appropriate for the slightly simplified model of cell structure that is utilised by this project.

[2]Such as source code to read and write PLY data into C programs.

In our key paper Turk defines a method for placing cells so they are randomly and evenly distributed across an arbitrary surface. The technique he described is not possible with the GLUT toolkit, however a suitable simplified model can be used. In this simplified model, each cell is represented by a single polygon, (see Section 4.4), thus allowing GLUT to render each cell directly. Although on models with low polygon counts, this simplification introduces inaccuracies, the good results are obtained on large models ($> 50,000$ cells).

The GLUT function calls required to render a polygon are shown in code extract in listing 4.1. This method for rendering each polygon separately allows a different colour to be assigned to each cell, thus, the colour can be used to represent the activator concentration within each cell. It is therefore possible to loop through each cell in an array containing the vertex (point) coordinates and concentration levels, making the rendering of the model very efficient.

```
glColor3fv(colour);
glBegin(GL_POLYGON);
        glVertex3fv(vertex1_xyz);
        glVertex3fv(vertex2_xyz);
        glVertex3fv(vertex3_xyz);
glEnd();
```

Listing 4.1: GLUT code to render a polygon

4.6.2 Basic Cube Implementation

Using the rendering method described in the previous section, the implementation of a simple cube in OpenGL was constructed. A 2D array was used to store the x, y and z coordinates of each vertex, and another 2D array stored vertex indices for each polygon. The vertex coordinates and indices were hard-coded into the program, this allowed rapid prototyping of the program without the additional complication of external model files.

It was decided to attempt to use this simple unit cube for prototyping the 3D Reaction-Diffusion implementation. This decision identified a problem that we had failed to recognise previously. The R-D equations proposed by Turing and used by Turk are based on a model of a cell structure in which no two cells share a neighbour cell. The difference between this model and the rendering of the basic cube using four-sided polygons is shown in Figure 4.7.

To overcome this problem each face (polygon) of the cube was divided into two triangles, thus removing the possibility of shared neighbours. This was a more sensible work-around for the problem than attempting to rewrite the R-D equations because the 3D models that had been collected all contained triangular polygons. This required the R-D equations to be adapted as explained in Section 4.4 to account for having only three neighbouring cells. However, another problem was introduced; each cell no longer had two neighbours in each the x- and y- directions. This is addressed in Section 4.6.5.

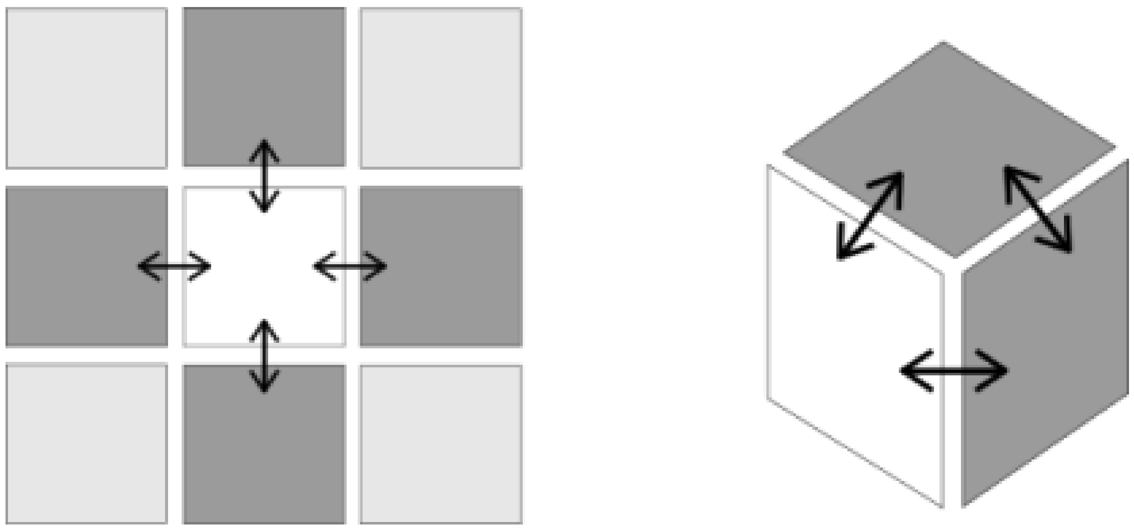

Figure 4.7: Left image shows 2D cells. Right shows cube with shared neighbours

To perform R-D simulation on the cube model, each cell's neighbours had to be known. Initially this was calculated by hand and hard-coded into the program for rapid prototyping. The altered R-D equations were implemented and tested on the cube (see Figure 4.8), which displayed variation of the colour of the faces, indicating that some process was occurring. However the simplicity of the model limited the extent of testing. Thus, work began on models defined in the PLY file format discussed in Section 4.5.

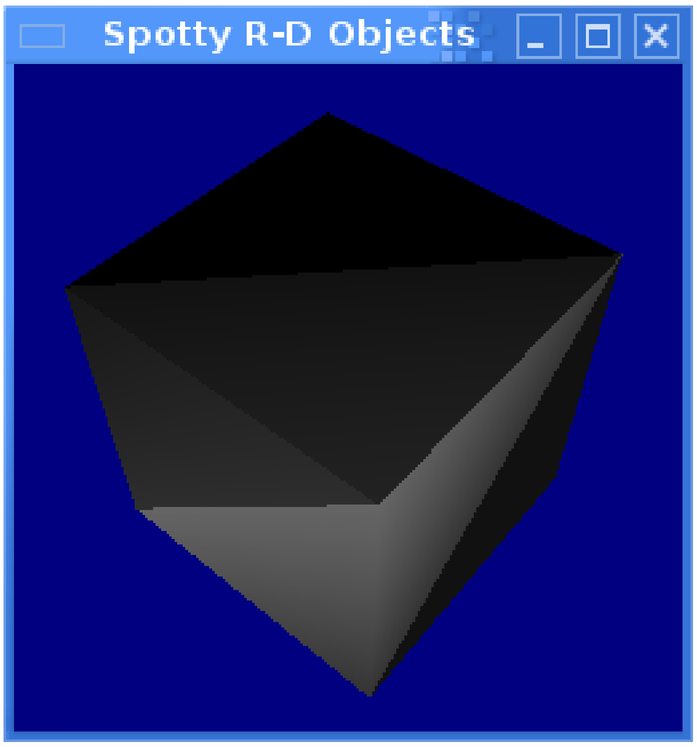

Figure 4.8: Reaction-Diffusion Cube Implementation

4.6.3 Reading in PLY Files

The PLY format was chosen as the input for this application for the reasons explained in earlier. There is a free, open-source C library for the reading and writing of PLY files, called RPly [2]. It is not the only library available but it is the most suitable for this project, since it can efficiently read and write both ASCII and binary encoded PLY files, as well as utilising callback functions for model data input.

RPly uses callbacks for retrieving data from the PLY files, which makes it very easy to accommodate custom extensions to the PLY format. This and the ability to write out PLY files are important for reasons explained in Section 4.6.4.

Reading the vertex coordinates and vertex indices for the polygons is very simple using RPly. The values are read one by one and the appropriate callback function is called, which stores each value into an array and increments an index pointer. Initially fixed-sized arrays were used for simplicity but subsequently dynamic memory allocation was used in order to make the process more general and able to cope with arbitrary sized models.

4.6.4 Additional Model Data

In order to perform R-D simulation on the imported models, each cell's neighbours must be known. For small models such as the basic cube, the values can be calculated on the fly but as the number of polygons increases the computational complexity of the neighbour search becomes prohibitively expensive.

It was decided that before R-D simulation should occur, the neighbours should be calculated and stored in a 2D array for use as a lookup table in the R-D function. The neighbour search algorithm loops through all three edges (vertex pairs) of each polygon, and then loops through every other polygon in the model to see if they share vertex pairs. Pseudo-code for this algorithm is shown in Algorithm 1. It may be possible to make the neighbour search algorithm more efficient by searching cells with similar indices first, and only if this fails then search the whole model. However because we are able to store the neighbour data in the PLY file, this search only needs to be performed once for each model and therefore the current algorithm was considered acceptable.

Algorithm 1 Pseudo-code for neighbour search algorithm

```
1: for all polygons in array do
2:     v1, v2, v3 = polygon vertices
3:     for all other polygons in array do
4:         if v1, v2, v3 match two of other polygon's vertices then
5:             mark this as a neighbour
6:         end if
7:     end for
8: end for
```

The neighbour search algorithm was later extended to store a normalised value for the relative length of the edges between adjacent cells, in order to implement the diffusion-weighting system Turk proposes. This system defines that more diffusion occurs across longer edges than across shorter edges over an equal amount of time, as discussed in Section 4.4. In terms of real-world biology, this system makes more sense than diffusion coefficients for x and y directions, since a larger surface area between cells would allow more chemical permeation through the cell membranes.

In order to use lighting techniques in OpenGL to give the model a more 3D look, the vertex normals must be known. This is required so that the renderer can calculate the incident light upon certain sections. Again this data is not normally provided in a PLY file (although some models did contain normal values) and therefore an algorithm was required to generate the normals from the PLY data.

The normal at each vertex is the average of the normals of each face around that point, therefore a method for calculating the normals of a surface given its three vertices was required. The normal of a plane can be calculated by the cross product of the difference between the vertex vectors:

$$[v1 - v2] \times [v2 - v3] \tag{4.10}$$

A function was written to calculate this, and this function was called from the loop that traverses the face and vertex arrays to generate the x, y and z normal vectors. The normals are averaged for the faces around each vertex, followed by normalisation to give a unit normal vector.

Generating all this extra data takes a considerable amount of time for large models ($\approx$ 20 minutes for bunny.ply with $\approx 70,000$ faces). Therefore it was sensible to store this additional information in a custom extension to the PLY file format. The PLY format was designed to allow this and the RPly library made writing the extra data into a correctly formatted PLY file trivial.

4.6.5 Stripe Generation in 3D

Considering all the principles explained previously in Section 4.6, a fully functioning R-D simulator was created that could synthesise random spots directly on the surface of any given 3D object described by a PLY file. However at this stage it was not possible to generate stripe patterns in this way due to the fundamental change in the model that occurred when transitioning to 3D.

In 2D there are diffusion coefficients defined for the x and y directions, this allowed stripes to be generated simply by making one of the coefficients greater than the other. Unfortunately in 3D using complex models with triangular faces, the x and y directions are not explicitly defined. The alignment of the neighbour cells is arbitrary and therefore cannot be defined to be in either the x or y direction.

One possible solution to this problem is to use the normal of the shared edge to define the direction of that neighbour. This would then allow diffusion to that neighbour to be proportional to the size of the x or y component of the normal vector. In this way it would be possible to create greater diffusion in one direction, however it would also introduce inaccuracies. There is also the question of how you classify the z-axis, would the z-component of a vector be considered to be the equivalent of x or y? With the standard use of the x, y and z axes, intuition tells us that it should be considered equivalent to the x-component. However the PLY format defines no standard orientation and therefore the models are often found not to be correctly aligned for this assumption to work.

Turk applied Meinhardt's five-chemical R-D system (see Section 4.4.1) to produce stripes in 3D, and so it was decided to follow this path. The implementation of this system in 3D required only a trivial extension to the simulator framework that had been developed, but adapting and implementing the new set of equations in 3D was harder. The initial implementation generated random stripe patterns by introducing perturbations to the substrate.

Further work was required to generate regular stripe patterns, this required the use of stripe initiator cells, which are cells that have a greater concentration of one chemical and are frozen in that state. This provides an input to destabilise the system which causes stripes to form radially around each initiator cell. For initial testing, stripe initiator cells were chosen randomly and to show that the technique worked.

A complicated mouse-picking function was required to allow the user to select cells with the mouse pointer that would act as stripe initiators. The implementation of this was tricky and far from perfect, cells are not always picked correctly when the model is zoomed out, but it is adequate for this application. With this system and the user interface described in Section 4.7.2, patterns such as the zebra in Figure 7.1 can be produced.

4.6.6 Cascading

In order to generate more complex patterns such as the leopard spots in Figure 7.2, Turk used a cascade process originally suggested by Bard [4] as the process responsible for complex mammalian coat patterns.

The cascade process involves running standard R-D on a random substrate, in order to generate a spot pattern, this pattern is then used as the substrate for a subsequent R-D simulation. Different patterns can be created by changing parameters between the two R-D runs, and by setting cells with certain concentration levels as 'frozen' for the second iteration, (see Section 4.3.3).

In Turk's implementation, the cascade occurs after a hard-coded amount of time has elapsed; presumably he selected this amount of time through trial error. This makes the process less effective on some larger models or when different parameters are used therefore a more universal general approach was desirable. It was hypothesised that once a stable pattern had been achieved the cumulative amount of movement creation of the chemicals would tend towards zero. So the ability to monitor the amount of change in the chemical levels was added to the simulator program, which would hopefully allow for the cascading to be triggered automatically upon convergence.

Testing with this monitoring capability showed that the assumption was incorrect; the amount of change of chemical concentrations remained at a high level even when the pattern stabilised. This is presumably because the chemicals still diffuse amongst the close neighbours, so the net effect does not alter the pattern.

Therefore it was necessary to perform profiling of the cumulative chemical change, in order to identify a characteristic point at which a pattern can be assumed stable. No graph-able data was produced from this, but Figure 4.9 shows the approximate shape of the graph of cumulative change. The point at which cascading occurred was chosen to be the bottom of the dip immediately after the main positive slope. This point can be detected automatically, by which time the pattern has stabilised on most of the models tested.

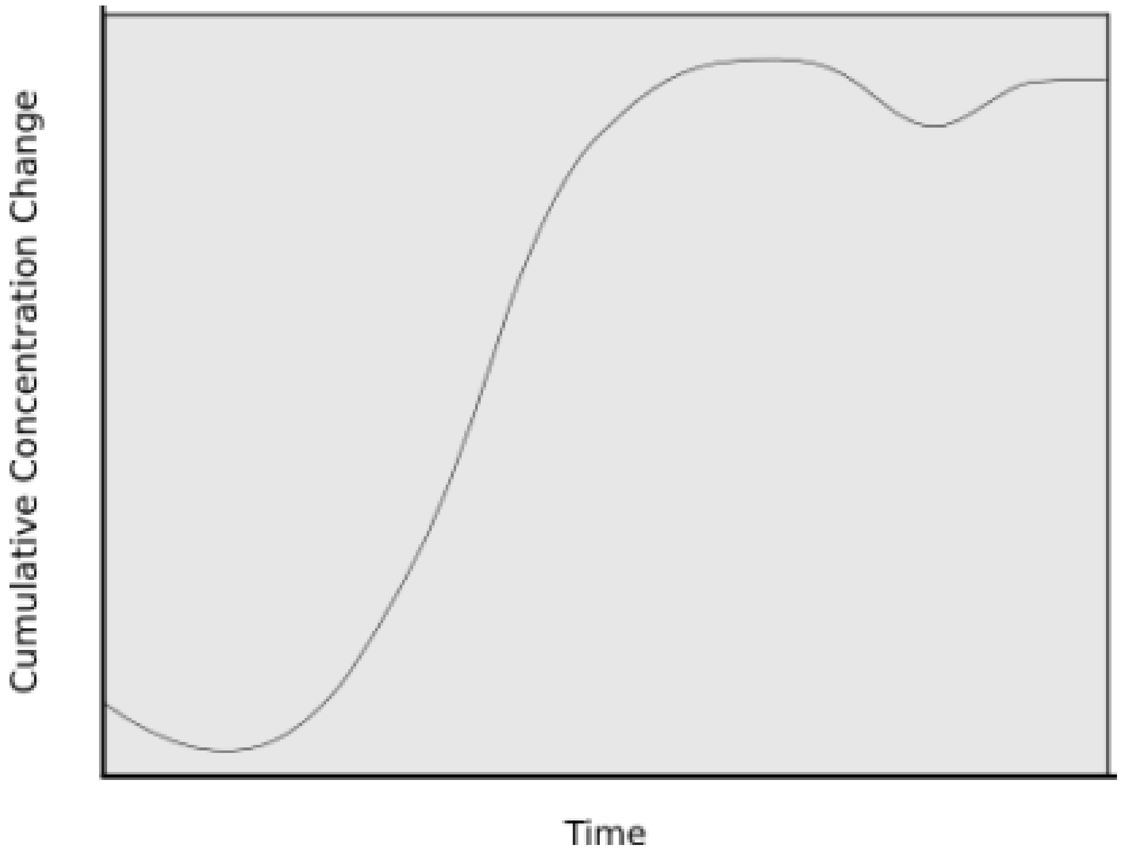

Figure 4.9: Graph of Cumulative Chemical Change

The leopard spot pattern seen in Figure 7.2 was created by first running R-D on the horse model with parameters set to generate large spots ($s = 0.003$). When convergence is detected, cells with activator concentrations between 0 and 2.5 were fixed at chemical levels so that neither is dominant ($cA = cI = 4$). Then R-D is run on this substrate with a large value for s (0.02), which ordinarily generates smaller spots. In this case because the frozen cells contain equal levels of activator and inhibitor, the spots tend to form in areas adjacent to the large spots, which results in the leopard style coat pattern.

4.7 User Interface

4.7.1 Motivation

There are several reasons why thought needs to be given to the design of the user interface for this system. Initially there needs to be a simple interface at the command line to set-up the type of reaction-diffusion to take place and the 3D PLY model to use. After this, when the OpenGL window is loaded there needs to be a user interface to control reaction-diffusion and the view of the 3D model.

When a 3D model is loaded from a file, the central point of the model is not necessary at the origin of the 3D space displayed in the OpenGL window. As the camera for the OpenGL window is always pointed at the origin, to be able to view the whole of the model, the camera viewpoint needs to be adjustable by users. Furthermore when a model is loaded, the initial orientation of the model from the perspective of the OpenGL camera is not necessarily the most suitable orientation for viewing reaction-diffusion, (see Figure A.1). Therefore the user needs to be able to rotate the model on all three axes.

The maximum and minimum x and y coordinates are recorded to ensure that the distance of the camera from the model is sufficiently far away that all the model can fit into the OpenGL window, (although this may require manual adjustment of the camera viewpoint by the user). However because the model may have to be rotated to produce a more aesthetic orientation, this may cause the model to no longer fit inside the window, a user controlled zooming function[3] is therefore necessary.

The ability to adjust the camera viewpoint, its distance from the model and the orientation of the model is useful for several other reasons. Firstly, for the five chemical (zebra stripe) reaction-diffusion implementation can require the seeding of cells, with the facilities previously described, the model can be adjusted to display the cell that needs to be seeded. Seeding also requires the user interface to provide a method for particular cells to be selected. The adjustment facilities are also useful to allow the user to focus on how reaction-diffusion evolves on different parts of the model or at different distances from the model.

[3]In real terms this is moving the camera so that it is either nearer or further away from the model.

As described in Section 4.6, reaction-diffusion directly mapped to models can be exported so they can be displayed in a viewer where no further reaction-diffusion takes place. Therefore a user interface is also required to facilitate this exporting.

In Section 4.6.6, convergence is detected automatically to allow cascading to begin, however it would be useful to start cascading manually if convergence is not detected or the user wishes to start cascading earlier.

4.7.2 Implementation

Before a user interface for the OpenGL window can be designed a small amount of set up is required at the command line. There are four different executables that can be run, three of them generate reaction-diffusion patterns on 3D models; the last is a viewer to load a model with a pre-generated reaction-diffusion pattern. The executable files with the parameters that need to be passed to them are listed below.

- $rdspots < PLYfile > [-p < Parametersfile >]$
- $rdstripes < PLYfile > [-p < Parametersfile >]$
- $rdcascade < PLYfile > [-p < Parametersfile >]$
- $rdviewer < Patternfile >$

Once the command is executed, the system generates neighbouring and normals data where necessary. When the rdstripes executable is loaded the command line asks how many cells the user wants to seed. Once this is determined the OpenGL model is loaded and the user must select the the cells to seed using the mouse as discussed in Section 4.6. Once all the cells have been seeded, reaction-diffusion can begin.

The rest of the OpenGL window user interface has been implemented with keyboard controls. To begin reaction-diffusion the user must press space, the user can also use space to pause reaction-diffusion. The arrow keys were determined to be the most logical choice for adjustment of the camera's viewpoint. Below is summary of the function each key performs:

- Right arrow: Moves camera viewpoint to the left.
- Left arrow: Moves camera viewpoint to the right.
- Up arrow: Moves camera viewpoint down.
- Down arrow: Moves camera viewpoint up.

Initially these controls may seem a little illogical but in real terms they mean that the model itself is moved in the direction of the pressed arrow key, which was found to be more intuitive for the user. It was also considered that the most logical keys for zooming in and out were the '+' and '-' keys.

The model in Figure 4.10 is a summary of how rotation about all three axes has been implemented. The model in Figure 4.10 allows rotation in both the clockwise and anti-

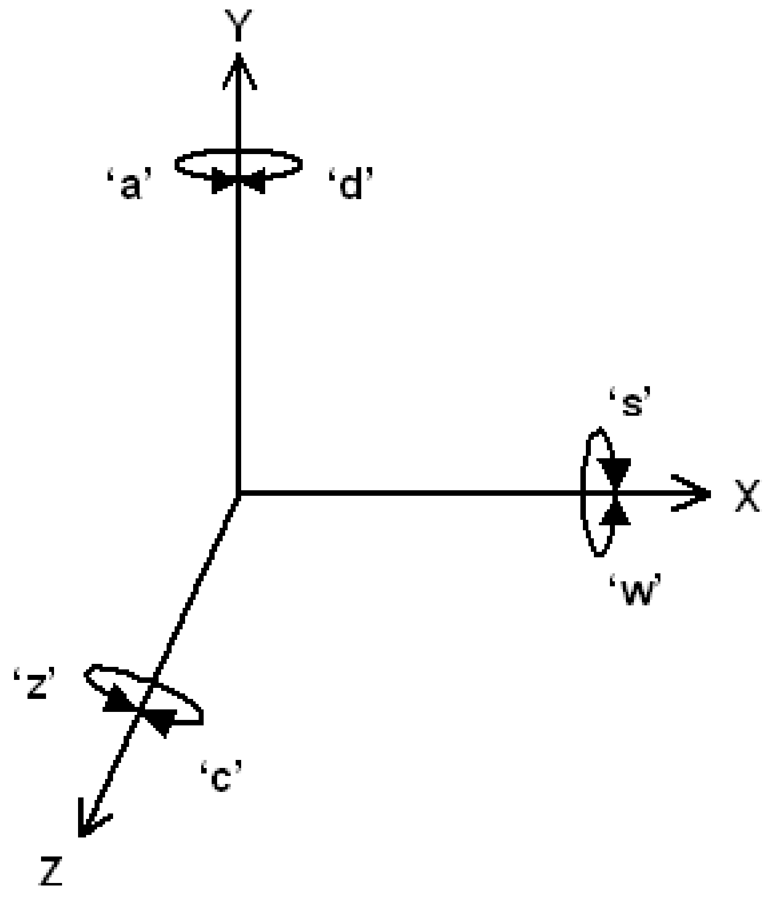

FIGURE 4.10: Representation of how to control model rotation

clockwise direction, for example, to rotate on the y-axis the user must press 'a' to rotate clockwise and 'd' to rotate anti-clockwise. The choice of keys was determined because of their proximity to each other; attempts were made to ensure the key choice was logical as possible but it was inevitably going to be difficult to convert three-dimensional rotation onto a two-dimensional keyboard.

After adjustments in viewpoint, model rotation and zooming it is possible for the user to reset to the initial values by pressing 'r', as it may be quicker to find the positioning required. As discussed in Section 4.7.1 the user must be able to specify when he or she wishes to export a skin pattern, thus, by pressing 'e' the user can export this 3D mapped model as a pattern file, which can be loaded in a separated viewer[4] along with PLY model file, where no further reaction-diffusion takes place. For the cascading implementation, despite convergence detection being implemented to start cascading, it was decided that space bar should be used to control the start of manual cascading, where the user desired it.

By clicking on the cross of the OpenGL window, the user can quit the system but ‘q’ has been implemented as a keyboard control for doing this.

[4]The controls for this viewer are the same as the main system, except for those which handle reaction-diffusion.

Chapter 5

Project Organisation and Milestones

5.1 Work Division

The initial research at the beginning of this project required to determine the key paper and obtain a background for the project was carried out by all four members of the group. Once the project was determined Robert Mills implemented a 1D reaction-diffusion system in Matlab. The 2D implementations were found by Qing Yan Zhang. These implementations were then modified by David Newman to produce source code that could be compiled on both Linux and Windows, as several of the implementations were not suited to Windows compilations. All the group members then set about modifying the parameter values in these 2D implementations to generate suitable patterns. Qing Yan Zhang then investigated different methods of texture synthesis to see whether procedural texture synthesis of reaction-diffusion is the most appropriate method to map skin patterns to 3D models.

Simon Smith was the first member to concentrate on 3D implementations, he firstly found source code to import PLY files and then adapted one of the 2D implementations to produce mapping of spot patterns to 3D models. He also developed a method to optimise reaction-diffusion texture mapping by determining neighbouring polygons and their normal vectors; this information is stored as part of the PLY file. After this David Newman formalised the user interface for the 3D implementation which enabled uncomplicated control of the 3D model and reaction-diffusion. At the same time Robert Mills attempted to find appropriate PLY models to perform mapping to, this often involved converting data from other formats to PLY. Finding models was quite difficult as very few appropriate models exist and those that do were often found to not be sufficiently detailed.

Once one 3D implementation was complete, the group researched how other types of reaction-diffusion could be mapped. Simon Smith took these ideas to produce further implementations by adapting the first. These provide reaction-diffusion stripes and cascading patterns on 3D models.

For both presentations and the final report all group members contributed an equal amount of slides/write-up. David Newman was responsible for integrating the each group member's write-up into one LaTeX document.

5.2 Key Milestones

This section describes the principal achievements of the team, providing an indicator of project flow in chronological order.

- Key paper determined
- Project goals determined
- 1D Reaction-Diffusion example implemented in Matlab to aid understanding of mathematics and the effect of varying parameters
- Initial progress presentation
- Matlab dropped in favour of OpenGL for 2D implementations
- 2D R-D examples found, and parameters varied to generate a range of patterns
- Basic 3D implementation in OpenGL, starting with a 12-cell cube
- PLY format selected for 3D models
- 3D implementation maps skin patterns onto animal models of arbitrary complexity
- Advanced 3D features added, including rotation and lighting
- User interface developed to control rotation etc. during execution
- 5-chemical R-D (Meinhardt) implemented to generate stripes
- 2nd presentation
- Cascading systems implemented

Chapter 6

Results

6.1 Spots Reaction-Diffusion System

Figure 6.1 is the output by using the default parameters[1]. From Figure 6.2, it can be seen that reaction speed S is the most significant factor that affects the R-D output. The first two pictures in Figure 6.2 show the outputs of R-D by increasing the S at different levels based on the default value of S, while the third image presents the output of R-D when the S is 5 times lower than the default value. From these three screen shots, it can be seen that the higher reaction speed results smaller spots. Through the testing process, it was also noticed that the higher reaction speed results shorter processing time. Figure 6.3 shows the

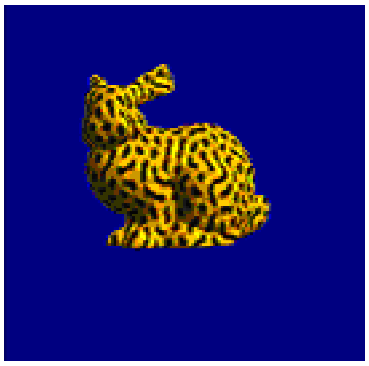

FIGURE 6.1: Default pattern, (DA = 0.0399, DI = 0.229, S = 0.005).

outputs of R-D with low diffusion rates of activator (DA) and inhibitor (DI) respectively. As shown in first image, by reducing the DA, the black spots become thinner. However, reduction of the DI causes bigger spots. While processing the testing, it was noticed that lower DA resulted shorter process time, but lower DI caused longer processing time. Furthermore, as shown in the third image, when the diffusion DI decreases to a certain level, the R-D process would not be able to be continued any more. Raising of diffusion

[1]A listing of all the parameter sets shown in the spotted bunny screen shots can be found in Section B.3.

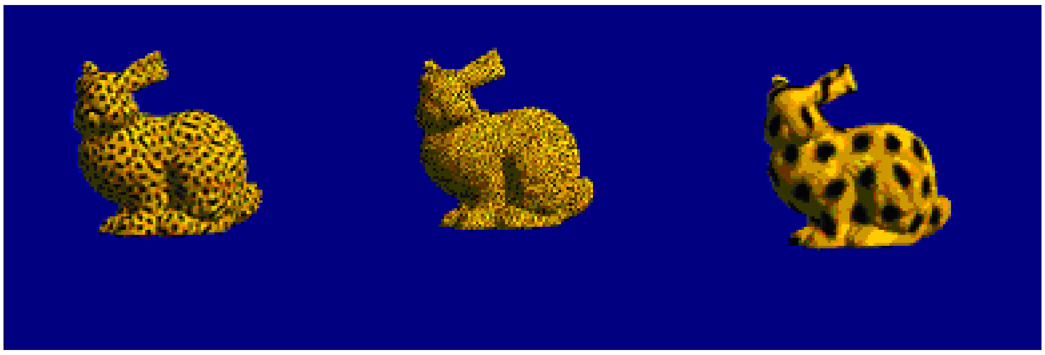

FIGURE 6.2: (a) reaction speed S doubled, (b) reaction speed S multiplied by 20, (c) reaction speed S divided by 5

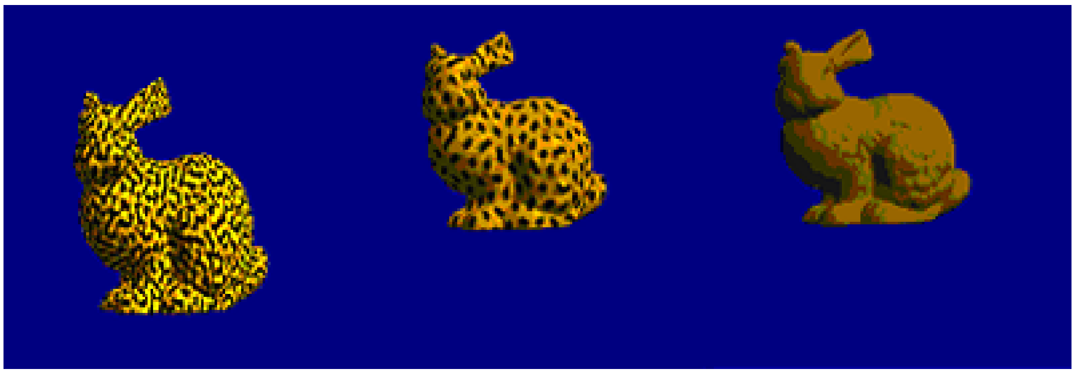

FIGURE 6.3: (a) reduced the activator's diffusion rate (DA), (b) reduced the inhibitor's diffusion rate ($DI = 0.2$), (c) further reduced the inhibitor's diffusion rate ($DI = 0.15$), R-D cannot continue.

rate obviously speeds up the R-D process. However, when the rate increases to a certain level, the output becomes completely saturated with inhibitor. Although the manner in which this occurs depending on whether the value for DI or DA is raised the final outcome is the same.

6.2 Stripes Reaction-Diffusion System

Meinhardt originally proposed the five-chemical reaction diffusion with the following parameters[2]. [10]
$p1 : 0.04$
$p2 : 0.06$
$p3 : 0.04$
$Dg : 0.05$
$Ds : 0.2$
Through using these parameters[3] produced (a) in Figure 6.4. The parameters previously

[2]A listing of all the parameter sets shown in the zebra screen shots can be found in Section B.4.
[3]Combined with seeding inhibitor cells on each hoof and the middle of the face and activator cells on the nose and both of the ears.

defined can affect the skin patterns produced in different ways. Dg and Ds affect the rate at which the two g and two s chemicals to diffuse therefore by increasing the value of Dg the stripes can be made wider as can be seen in (b) of Figure 6.4. If Dg is increased too much so that it is almost the same as Ds the pattern is overwhelmed with activator, if the Dg and Ds values are swapped over very little activator is produced and the reaction-diffusion ceases after a short time. When Meinhardt discussed his five-chemical system he

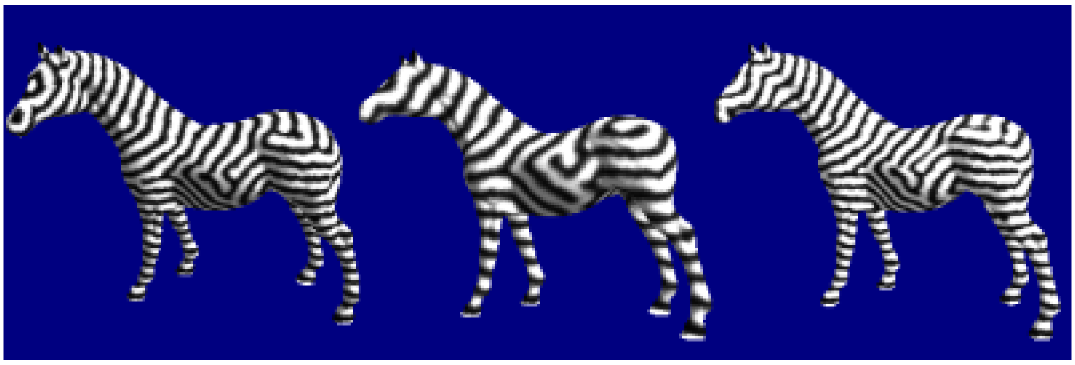

FIGURE 6.4: (a)Meinhardt's original parameters, (b)increased value for Dg, (c)increased value of $p2$.

did not explain directly what $p1$, $p2$ and $p3$ represented, except to say that $p1$ affects the g chemicals, $p2$ affects the r chemical and $p3$ the s chemicals. Therefore testing was required to find out how they affected reaction-diffusion. By increasing the value of $p1$ or $p3$ or by decreasing the value of $p2$ caused the pattern to be overwhelmed with activator. However, if the value of $p2$ is increased relative to the Meinhardt's original parameters the black stripes generated are thinner and the white stripes slightly thicker as can be seen in the (c) of Figure 6.4.

Due to the amount of time required to generate patterns, the amount of time required to generate definitive explanations of how each parameter effects the pattern would take many hours of investigation. Therefore the conclusions drawn in this section are based only on five to ten parameter sets.

6.3 Cascading Reaction-Diffusion System

The results of the cascading system are affected by many different variables, such as the point at which it cascades, as well as the standard R-D parameters. Only the standard parameters are readily available for the user to change without altering the source code, the results below were created by only changing those standard parameters.

The two leopard-print results seen in Figure 6.5 were created by generating large spots and then cascading with a smaller spot forming system. The large spots are set to have activator and inhibitor concentrations of 4 and frozen in this state. When this pattern is used as the substrate for the next pattern forming system, the new spots tend to form around the edges of the original spots.

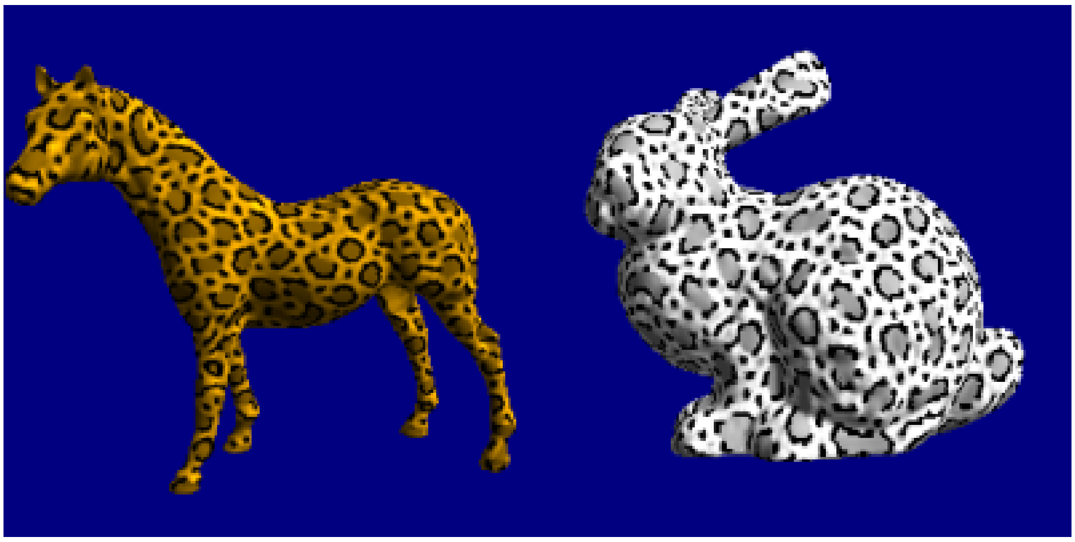

FIGURE 6.5: (a)coloured leopard-skin horse, (b)greyscale leopard-skin bunny.

The bunny was generated with the default parameter options, however the to generate the leopard pattern on the horse required modification of these parameters: DA was decreased to 0.0399, DI was left unchanged as 0.229, for the first iteration the speed of reaction S was set to 0.002, for the second run ($S2$) it was changed to 0.01.

There are many other complex patterns that can be created using the cascading technique as shown by Turk [19], however due to the length of time required to generate each pattern (double that of regular spot formation) experimentation in this area has been restricted.

6.4 Accompanying CD

This report is accompanied by a CD with the 3D Implementations that have been described in Section 4.6, along with PLY models, parameter and pattern files. Appendix C contains a map of this CD, which is broken down into four main directoires; two for Windows users and two for Linux users. One of the Windows directories contains pre-compiled executables of the 3D Implementations. These executables have been compiled on WindowsXP and should be executable by Windows users who have OpenGL with the GLUT toolkit installed. The second Windows directory contains the source code for the 3D implementations with a batch file to build the executables. The two Linux directories are very similar to the Windows directories; one contains pre-compiled executables and the second has the source code and a makefile to build the executables.

Chapter 7

Conclusions and Further Work

7.1 Conclusions

Many methods are available to generate skin patterns using reaction-diffusion systems; when combined with 3D model mapping the possibilities are extensive. This report describes the implementation and investigation of several systems, namely the 2-chemical arrangement upon which the field is founded on, a 5-chemical variant and cascading where one stabilised reaction-diffusion forms the substrate for a second. Seeding was also used in conjunction with the 5-chemical variant to improve resemblance of patterns to real-world animals.

The main achievement of the project is the bespoke software implementation of three fascinating R-D systems, which are capable of easily synthesising patterns on any arbitrary model supplied in the versatile PLY format, exporting and viewing patterns without re-generation, and allowing the user to define R-D system parameters without the need to re-compile the software. In addition, the code has been developed and tested on both Linux and Windows based operating systems, in the aim of platform independence for future investigation or development.

The main goal of the project was to replicate work from the key paper, which included generating patterns akin to a zebra's stripes and a leopard's spots. The systems aimed to generate patterns which were comparable to both those generated in the aforementioned key paper, as well as real-world animals. Figures 7.1 to 7.2 show patterns (a) created with software from this project, (b) from Turk's work in 1991, and (c), a real world example of a corresponding skin pattern. Modern lighting techniques make the patterns in (a) more impressive visually than in (b), but for both Figure 7.1 and Figure 7.2, the images are very comparable (remembering that each execution of the software has random permutations to begin with). The comparison to (c) in these cases shows similar traits, but more complexity in the case of the real animal; for instance the zebra has a completely white belly. If it was desired to emulate the skin of a single animal in great detail, more prior information would have to be built into the model. This could include specifications, which would allow different R-D systems to be applied to different regions, such as not allowing pigment variation on the belly. However the goal of this project was elsewhere.

In the case of Figure 7.3, Turk did not generate a 3D model with spots on, so the bunny was arbitrarily chosen as a test model (poor lab rabbits!). The pattern shows nice variation with a mixture of spots and wavy stripes, which are also exhibited to a certain extent in the rabbit shown in Figure 7.3 (b). In summary, the patterns generated show excellent

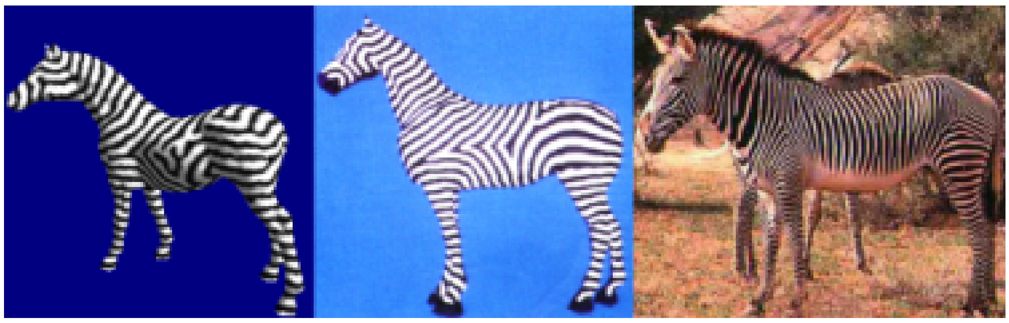

FIGURE 7.1: (a) this project's zebra, (b) Turk's zebra, (c) real-world zebra.

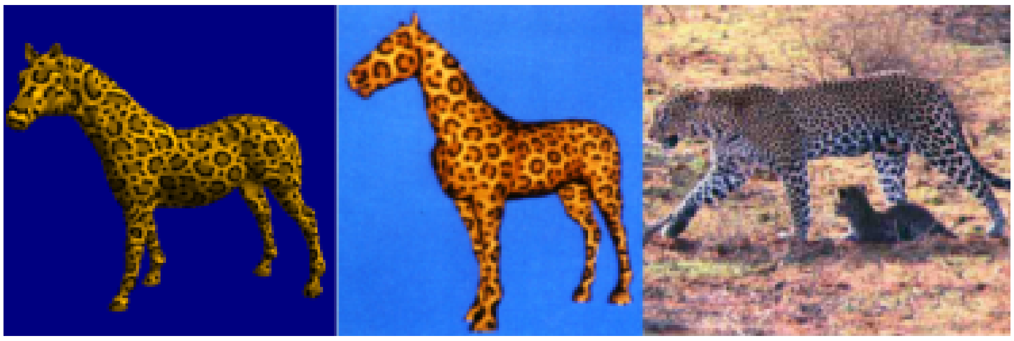

FIGURE 7.2: (a) this project's leopard pattern, (b) Turk's leopard pattern, (c) real-world leopard.

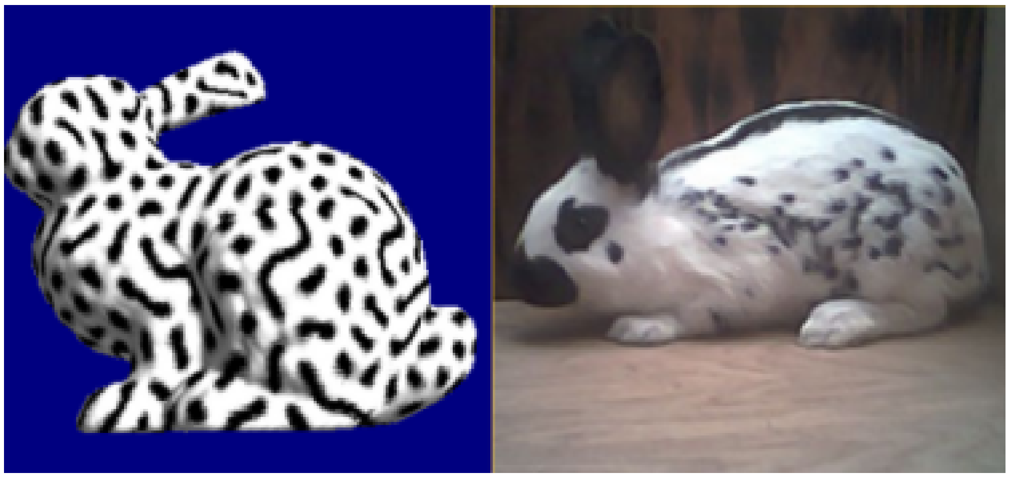

FIGURE 7.3: (a) this project's spotted bunny, (b) real-world bunny.

resemblance to the models seen in the key paper and exhibit characteristics seen in real-world animals, although the pattern complexity in genuine animals is greater than in the computer-generated cases. This provides an indication that the reaction diffusion systems implemented do indeed represent one mechanism employed by real-world biology in the generation of the wide variety of skin patterns seen across the animal kingdom.

7.2 Further Work

Though this project has made significant achievements against its goals, the area is still wide open for further research. To improve on the work described previously, the following extensions are proposed:

1. Optimisation of algorithms used in 3D implementation

Although C with OpenGL was used to improve execution speed over the initial attempts, overall operation is still quite slow, especially with large 3D meshes. Functionality was the primary objective, over execution times. A quicker execution would enable experimental investigation to be more widespread in parameter selection.

2. Practical comparison of texture mapping methods

Three categories of texture mapping methods are described in Section 2.4. Procedural texture synthesis was employed; this was deemed to be the best for the application theoretically. However, it would be interesting to generate patterns with methods from each category, in order to confirm or disprove the theoretical beliefs.

3. Implement other R-D methods

Many reaction-diffusion systems exist; three interesting examples are realised in this work. Using the framework for generating and displaying patterns on 3D models, extensions to implement further R-D systems should be relatively straightforward.

4. Improve Convergence Detection

The cascading implementation requires convergence to be detected. Although this is implemented and works well for the large 3D models, its performance could be improved for smaller models.

Appendix A

Screen Shots

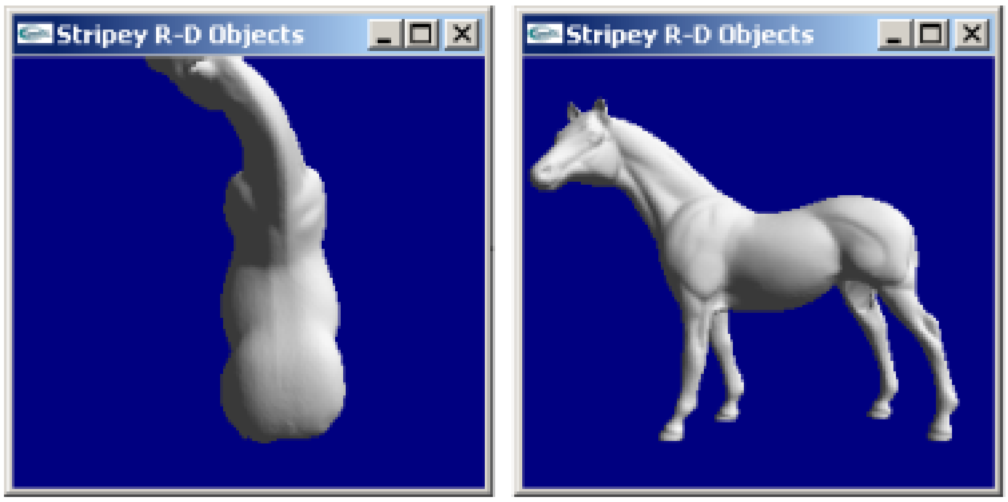

Figure A.1: Initial and Aesthetic Orientations for Horse Model

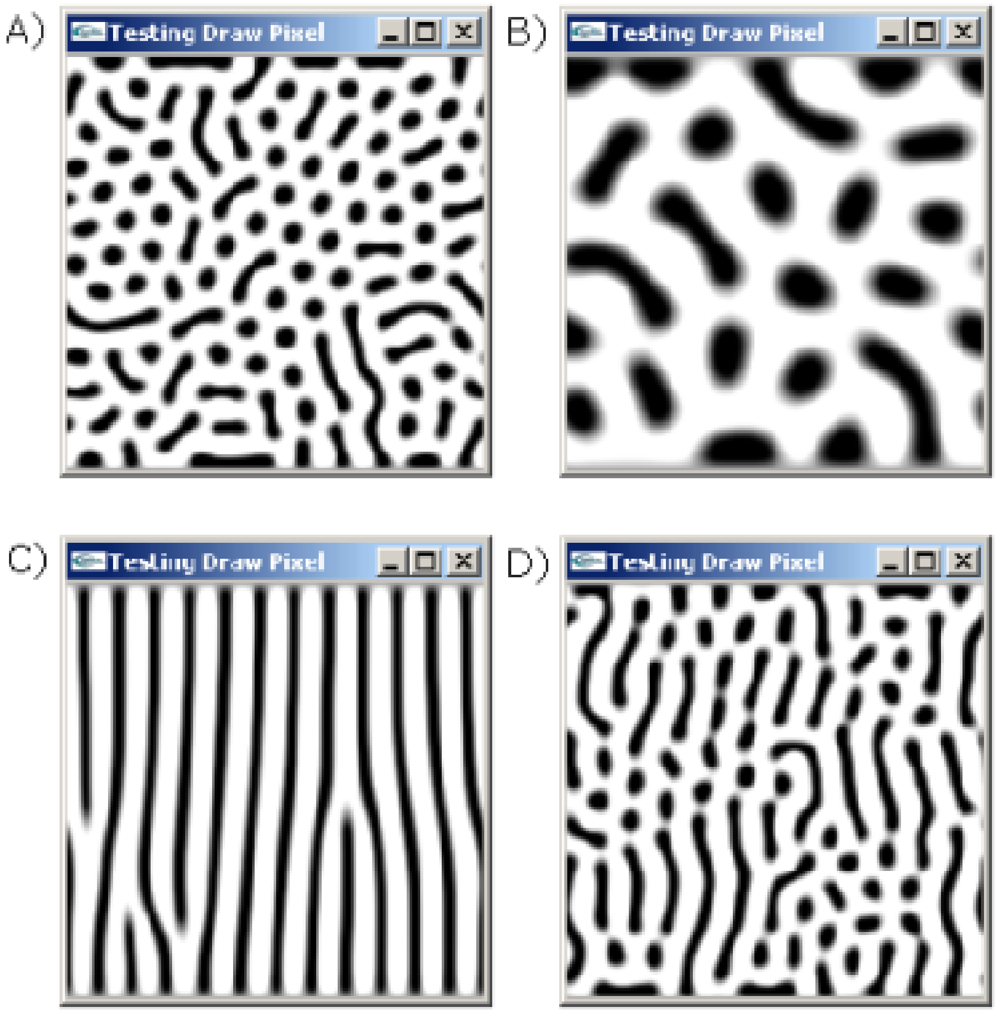

FIGURE A.2: A) Cheetah spots B) Cheetah spots with greater speed of diffusion C) Zebra Stripes D) Same as A but with a slightly lower horizontal inhibitor diffusion co-efficient

Appendix B

Parameter Files

B.1 Parameters for 1st 2D Implementation

DAX - Diffusion coefficient of Activator in the horizontal direction.
DAY - Diffusion coefficient of Activator in the vertical direction.
DIX - Diffusion coefficient of Inhibitor in the horizontal direction.
DIY - Diffusion coefficient of Inhibitor in the vertical direction.
S - Speed of diffusion.

A) Cheetah spots,
DAX 0.03199,
DAY 0.03199,
DIX 0.229,
DIY 0.229,
S 0.005.

B) Cheetah spots with greater speed of diffusion,
DAX 0.03199,
DAY 0.03199,
DIX 0.229,
DIY 0.229,
S 0.001.

C) Zebra stripes,
DAX 0.09597,
DAY 0.03199,
DIX 0.229,
DIY 0.229,
S 0.005.

D) Same as A but with a slightly lower horizontal inhibitor diffusion co-efficient,
DAX 0.03199,
DAY 0.03199,
DIX 0.209,
DIY 0.229,
S 0.005.

B.2 Parameters for 3rd (cascading) 2D Implementation

DA - Diffusion coefficient of Activator in the horizontal and vertical directions.
DI - Diffusion coefficient of Inhibitor in the horizontal and vertical directions.
S - Speed of diffusion.

DA 0.03125,
DI 0.125,
S 0.005.

B.3 Parameters for Spotted Bunny 3D Implementation

DA - Diffusion coefficient of Activator.
DI - Diffusion coefficient of Inhibitor.
S - Speed of diffusion.

1) Original Parameters in first screen shot,
DA 0.0399,
DI 0.229,
S 0.005.

2) Altered Reaction Speed, (S).

A)DA 0.0399,
DI 0.229,
S 0.01.

B)DA 0.0399,
DI 0.229,
S 0.1.

C)DA 0.0399,
DI 0.229,
S 0.001.

3) Altered Diffusion coefficients, (DA and DI).

A)DA 0.01,
DI 0.229,
S 0.005.

B)DA 0.0399,
DI 0.2,
S 0.005.

C)DA 0.0399,
DI 0.15,
S 0.005.

B.4 Parameters for Zebra 3D Implementation

p1 - Affects the diffusion of g chemicals.
p2 - Affects the diffusion of r chemical.
p3 - Affects the diffusion of s chemicals.
Dg - The diffusion rate of g chemicals.
Ds - The diffusion rate of s chemicals.

A)p1 0.04,
p2 0.06,
p3 0.04,
Dg 0.05,
Ds 0.2.

B)p1 0.04,
p2 0.06,
p3 0.04,
Dg 0.1,
Ds 0.2.

C)p1 0.04,
p2 0.08,
p3 0.04,
Dg 0.05,
Ds 0.2.

Appendix C

CD Map

- Linux
  - Executables
    - 5chemparams.txt - Parameters file for stripy reaction-diffusions.
    - bestzebra.pat - Pattern file with reference horse_nbs.ply and stripy skin pattern. pattern data to map to it.
    - bunny_nbs.ply - PLY file of a bunny with neighbouring and normals data.
    - dolphin_nbs.ply - PLY file of a dolphin with neighbouring and normals data.
    - horse_nbs.ply - PLY file of a horse with neighbouring and normals data. data mapped to it.
    - leopardbunny.pat - Pattern file with reference bunny_nbs.ply and cascading skin.
    - leopardhorse.pat - Pattern file with reference horse_nbs.ply and cascading skin.
    - params.txt - Parameters file for spots and cascading reaction-diffusions.
    - rdcascade - Executable for mapping cascading reaction-diffusion on 3D models.
    - rdspots - Executable for mapping reaction-diffusion spots on 3D models.
    - rdstripes - Executable for mapping five chemical reaction-diffusion stripes on 3D models.
    - rdviewer - Executable for viewing mapped reaction-diffusion patterns on 3D models.
  - Source
    - 5chemparams.txt - Parameters file for stripy reaction-diffusions.
    - bestzebra.pat - Pattern file with reference horse_nbs.ply and stripy skin pattern. pattern data to map to it.
    - bunny_nbs.ply - PLY file of a bunny with neighbouring and normals data.

    * dolphin_nbs.ply - PLY file of a dolphin with neighbouring and normals data.
    * horse_nbs.ply - PLY file of a horse with neighbouring and normals data. data mapped to it.
    * leopardbunny.pat - Pattern file with reference bunny_nbs.ply and cascading skin.
    * leopardhorse.pat - Pattern file with reference horse_nbs.ply and cascading skin.
    * Makefile - Builds four executable (rdcascade, rdspots, rdstripes and rdviewer).
    * params.txt - Parameters file for spots and cascading reaction-diffusions.
    * rdcascade.c - C file for mapping cascading reaction-diffusion on 3D models.
    * rdspots.c - C file for mapping reaction-diffusion spots on 3D models.
    * rdstripes.c - C file for mapping five chemical reaction-diffusion stripes on 3D models.
    * rdviewer.c - C file for viewing mapped reaction-diffusion patterns on 3D models.
    * rply.c - C file for read in of PLY file data.
    * rply.h - C header file for read in of PLY file data.

- Windows
  - Executables
    * 5chemparams.txt - Parameters file for stripy reaction-diffusions.
    * bestzebra.pat - Pattern file with reference horse_nbs.ply and stripy skin pattern. pattern data to map to it.
    * bunny_nbs.ply - PLY file of a bunny with neighbouring and normals data.
    * dolphin_nbs.ply - PLY file of a dolphin with neighbouring and normals data.
    * horse_nbs.ply - PLY file of a horse with neighbouring and normals data. data mapped to it.
    * leopardbunny.pat - Pattern file with reference bunny_nbs.ply and cascading skin.
    * leopardhorse.pat - Pattern file with reference horse_nbs.ply and cascading skin.
    * params.txt - Parameters file for spots and cascading reaction-diffusions.
    * rdcascade.exe - Executable for mapping cascading reaction-diffusion on 3D models.
    * rdspots.exe - Executable for mapping reaction-diffusion spots on 3D models.
    * rdstripes.exe - Executable for mapping five chemical reaction-diffusion stripes on 3D models.

    * rdviewer.exe - Executable for viewing mapped reaction-diffusion patterns on 3D models.

- Source
    * 5chemparams.txt - Parameters file for stripy reaction-diffusions.
    * bestzebra.pat - Pattern file with reference horse_nbs.ply and stripy skin pattern. pattern data to map to it.
    * bunny_nbs.ply - PLY file of a bunny with neighbouring and normals data.
    * dolphin_nbs.ply - PLY file of a dolphin with neighbouring and normals data.
    * horse_nbs.ply - PLY file of a horse with neighbouring and normals data. data mapped to it.
    * leopardbunny.pat - Pattern file with reference bunny_nbs.ply and cascading skin.
    * leopardhorse.pat - Pattern file with reference horse_nbs.ply and cascading skin.
    * run.bat - Builds four executable (rdcascade.exe, rdspots.exe, rdstripes.exe and rdviewer.exe).
    * params.txt - Parameters file for spots and cascading reaction-diffusions.
    * rdcascade.c - C file for mapping cascading reaction-diffusion on 3D models.
    * rdspots.c - C file for mapping reaction-diffusion spots on 3D models.
    * rdstripes.c - C file for mapping five chemical reaction-diffusion stripes on 3D models.
    * rdviewer.c - C file for viewing mapped reaction-diffusion patterns on 3D models.
    * rply.c - C file for read in of PLY file data.
    * rply.h - C header file for read in of PLY file data.

Bibliography

[1] The mechanochemical theory of morphogenesis. http://www.maths.ox.ac.uk/ maini/ public/gallery/mctom.htm, 2004. Last checked: 24/01/2005.

[2] Rply: Ansi c library for ply file format input and output. http://www.cs.princeton.edu/ diego/professional/rply/, July 2004. Last checked 26/01/2005.

[3] J.B.L. Bard. A unity underlying the different zebra striping patterns. *Zoological (London)*, 183:527 – 539, 1977.

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

[5] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, Department of Computer Science, University of Utah, December 1974.

[6] Dictionary.com. Morphogenesis. http://dictionary.reference.com/search?q= morphogenesis, January 2005.

[7] K. Fleischer, D. Laidlaw, B. Currin, and A. Barr. Cellular texture generation. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 95)*, pages 239–248, August 1995.

[8] Z. Krpti. 3d object converter. http://web.axelero.hu/karpo/, November 2004. Last checked: 22/01/2005.

[9] J. Lewis. The unofficial 3dstudio 3ds file format. http://www.the-labs.com/Blender/3DS-details.html, April 1998. Last checked: 22/01/2005.

[10] H. Meinhardt. *Models of Biological Pattern Formation.* Academic Press, London, 1982.

[11] J.D. Murray. A pre-pattern formation mechanism for animal coat markings. *Theoretical Biology*, 88:161–199, 1981.

[12] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2000)*, pages 465–470, July 2000.

[13] Rector and Visitors of the University of Virginia. Morphogenesis & regenerative medicine. http://www.morphogenesis.virginia.edu/whatis.htm, 2003. Last checked: 24/01/2005.

[14] H. Si. .ply - plyhedral file format. http://tetgen.berlios.de/fformats.ply.html, January 2005. Last checked: 22/01/2005.

[15] V. Swarup. Source code for programs written as a part of research and development. http://www.csse.monash.edu.au/ vipuls/source/, June 2004. Last checked: 25/01/2005.

[16] J. Swinton. Turings last, lost work. http://www.swintons.net/deodands/archives/000087.html, August 2003. Last checked: 27/01/2005.

[17] J. Swinton. Alan turing and morphogenesis. http://www.swintons.net/ jonathan/turing.htm, June 2004. Last checked: 23/01/2005.

[18] A.M. Turing. The chemical basis of morphogenesis. *Philosophical Transactons of the Royal Society B (London)*, 237(641):37–72, August 1952.

[19] G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics (SIGGRAPH 91)*, 25(4), July 1991.

[20] G. Turk. Texture synthesis on surfaces. *ACM SIGGRAPH 2001 Conference Proceedings*, pages 347–354, 2001.

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