Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion

Greg Turk
University of North Carolina at Chapel Hill

Abstract

This paper describes a biologically motivated method of texture synthesis called *reaction-diffusion* and demonstrates how these textures can be generated in a manner that directly matches the geometry of a given surface. Reaction-diffusion is a process in which two or more chemicals diffuse at unequal rates over a surface and react with one another to form stable patterns such as spots and stripes. Biologists and mathematicians have explored the patterns made by several reaction-diffusion systems. We extend the range of textures that have previously been generated by using a cascade of multiple reaction-diffusion systems in which one system lays down an initial pattern and then one or more later systems refine the pattern. Examples of patterns generated by such a cascade process include the clusters of spots on leopards known as rosettes and the web-like patterns found on giraffes. In addition, this paper introduces a method by which reaction-diffusion textures are created to match the geometry of an arbitrary polyhedral surface. This is accomplished by creating a mesh over a given surface and then simulating the reaction-diffusion process directly on this mesh. This avoids the often difficult task of assigning texture coordinates to a complex surface. A mesh is generated by evenly distributing points over the model using relaxation and then determining which points are adjacent by constructing their Voronoi regions. Textures are rendered directly from the mesh by using a weighted sum of mesh values to compute surface color at a given position. Such textures can also be used as bump maps.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.5 [Computer Graphics]: Three-Dimensional Graphics and Realism - Color, shading, shadowing and texture; J.3 [Life and Medical Sciences]: Biology.

Additional Keywords and Phrases: Reaction-diffusion, biological models, texture mapping.

Introduction

Texture mapping was introduced in [Catmull 74] as a method of adding to the visual richness of a computer generated image without adding geometry. There are three fundamental issues that must be addressed to render textures. First, a texture must be acquired. Possibilities include creating a texture procedurally, painting a texture, or digitally scanning a texture from a photograph. Next, we need to define a mapping from the texture space to the space of the model to be textured. Defining this mapping should not require a great deal of a user's time. This mapping should not noticeably distort the texture. Finally, we require a method of sampling the texture during rendering so that the final image contains no artifacts due to aliasing or resulting from the underlying texture representation [Heckbert 89]. These three issues are often interrelated, and this is true of the techniques in this paper.

This paper explores a procedural method for texture synthesis and also introduces a new method for fitting a texture to a surface. Either of these techniques can be used separately, but the examples given here shows the strength of using them together to produce natural textures on complex models. After a discussion of previous texturing methods, the majority of the paper is divided into two parts, one for each of these topics.

The first part of this paper describes a chemical mechanism for pattern formation know as *reaction-diffusion*. This mechanism, first described in [Turing 52], shows how two or more chemicals that diffuse across a surface and react with one another can form stable patterns. A number of researchers have shown how simple patterns of spots and stripes can be created by reaction-diffusion systems [Bard 81; Murray 81; Meinhardt 82]. We begin by introducing the basics of how a reaction-diffusion system can form simple patterns. We then introduce new results that show how more complex patterns can be generated by having an initial pattern set down by one chemical system and further refined by later chemical systems. This widens the range of patterns that can be generated by reaction-diffusion to include such patterns as the rosettes found on leopards and the multiple-width stripes found on some fish and snakes. These patterns could be generated on a square grid and then mapped onto an object's surface using traditional techniques, but there are advantages to synthesizing the pattern directly on the surface to be textured in a manner that will be described next.

The second part of this paper presents a method of generating a mesh over the surface of a polyhedral model that can be used for texture synthesis. The approach uses relaxation to evenly distribute points across the model's surface and then divides the surface into cells centered at these points. We can simulate reaction-diffusion systems directly on this mesh to create textures. Because there is no mapping from texture space to the object, there is no need to assign texture

 ACM-0-89791-436-8/91/007/0289 $00.75

coordinates to polygons and there is no distortion of the textures. At no time is the texture stored in some regular $m \times n$ grid, as are most textures. It is likely that other texture generation methods in addition to reaction-diffusion could also make use of such a mesh. Images of surfaces that have been textured using a mesh do not show aliasing artifacts or visual indication of the underlying mesh structure. These textures can also be used for bump mapping, a technique introduced in [Blinn 78] to give the appearance of a rough or wrinkled surface. The three steps involved in texturing a model as in Figures 4, 5 and 6 are: (1) generate a mesh that fits the polyhedral model, (2) simulate a reaction-diffusion system on the mesh (solve a partial differential equation) and (3) use the final values from the simulation to specify surface color while rendering the polygons of the model.

Artificial Texture Synthesis

A great strength of procedurally generating textures is that each new method can be used in conjunction with already existing functions. Several methods have been demonstrated that use composition of various functions to generate textures. Gardner introduced the idea of summing a small number of sine waves of different periods, phases and amplitudes to create a texture [Gardner 85]. Pure sine waves generate fairly bland textures, so Gardner uses the values of the low period waves to alter the shape of the higher period waves. This method gives textures that are evocative of patterns found in nature such as those of clouds and trees. Perlin uses band-limited noise as the basic function from which to construct textures [Perlin 85]. He has shown that a wide variety of textures (stucco, wrinkles, marble, fire) can be created by manipulating such a noise function in various ways. [Lewis 89] describes several methods for generating isotropic noise functions to be used for texture synthesis. A stunning example of using physical simulation for texture creation is the dynamic cloud patterns of Jupiter in the movie 2010 [Yaeger and Upson 86].

Recent work on texture synthesis using reaction-diffusion is described in [Witkin and Kass 91]. They show the importance of anisotropy by introducing a rich set of new patterns that are generated by anisotropic reaction-diffusion. In addition, they demonstrate how reaction-diffusion systems can be simulated rapidly using fast approximations to Gaussian convolution.

A texture can be created by painting an image, and the kinds of textures that may be created this way are limitless. An unusual variant of this is to paint an "image" in the frequency domain and then take the inverse transform to create the final texture [Lewis 84]. Lewis demonstrates how textures such as canvas and wood grain can be created by this method. An extension to digital painting, described in [Hanrahan and Haeberli 90], shows how use of a hardware z-buffer can allow a user to paint interactively onto the image of a three-dimensional surface.

Mapping Textures onto Surfaces

Once a texture has been created, a method is needed to map it onto the surface to be textured. This is commonly cast into the problem of assigning texture coordinates (u,v) from a rectangle to the vertices of the polygons in a model. Mapping texture coordinates onto a complex surface is not easy, and several methods have been proposed to accomplish this. A common approach is to define a mapping from the rectangle to the natural coordinate system of the target object's surface. For example, latitude and longitude can be used to define a mapping onto a sphere, and parametric coordinates can be used when mapping a texture onto a cubic patch [Catmull 74]. In some cases an object might be covered by multiple patches, and in these instances care must be taken to make the edges of the patches match. A successful example of this is found in the bark texture for a model of a maple tree in [Bloomenthal 85].

Another approach to texture mapping is to project the texture onto the surface of the object. One example of this is to orient the texture square in $\mathbf{R}^3$ (Euclidean three-space) and perform a projection from this square onto the surface [Peachey 85]. Related to this is a two-step texture mapping method given by [Bier and Sloan 86]. The first step maps the texture onto a simple intermediate surface in $\mathbf{R}^3$, such as a box or cylinder. The second step projects the texture from this surface onto the target object. A different method of texture mapping is to make use of the polygonal nature of many graphical models. In this approach, taken by [Samek 86], the surface of a polyhedral object is unfolded onto the plane one or more times and the average of the unfolded positions of each vertex is used to determine texture placement. A user can adjust the mapping by specifying where to begin the unfolding of the polyhedral object.

Each of the above methods has been used with success for some models and textures. There are pitfalls to these methods, however. Each of the methods can cause a texture to be distorted because there is often no natural map from the texture space to the surface of the object. This is a fundamental problem that comes from defining the texture pattern over a geometry that is different than that of the object to be textured. Some of these techniques also require a good deal of user intervention. One solution to these problems for some images is the use of solid textures. A *solid texture* is a color function defined over a portion of $\mathbf{R}^3$, and such a texture is easily mapped onto the surfaces of objects [Peachey 85; Perlin 85]. A point (x,y,z) on the surface of an object is colored by the value of the solid texture function at this point in space. This method is well suited for simulating objects that are formed from a solid piece of material such as a block of wood or a slab of marble. Solid texturing is a successful technique because the texture function matches the geometry of the material being simulated, namely the geometry of $\mathbf{R}^3$. No assignment of texture coordinates is necessary.

Quite a different approach to matching texture to surface geometry is given in [Ma and Gagalowicz 85]. They describe several methods for creating a local coordinate system at each point on the surface of a given model. Statistical properties of a texture are then used to synthesize texture on the surface so that it is oriented to the local coordinate system.

Part One: Reaction-Diffusion

This section describes a class of patterns that are formed by reaction-diffusion systems. These patterns are an addition to the texture synthesist's toolbox, a collection of tools that include such procedural methods as Perlin's noise function and Gardner's sum-of-sine waves. The reaction-diffusion patterns can either be used alone or they can be used as an initial pattern that can be built on using other procedures. This section begins by discussing reaction-diffusion as it relates to developmental biology and then gives specific examples of patterns that can be generated using reaction-diffusion.

A central issue in developmental biology is how the cells of an embryo arrange themselves into particular patterns. For example, how is it that the cells in the embryo of a worm become organized into segments? Undoubtedly there are many organizing mechanisms working together throughout the development of an animal. One possible mechanism, first described by Turing, is that two or more chemicals can diffuse through an embryo and react with each other until a stable pattern of chemical concentrations is reached [Turing 52]. These chemical pre-patterns can then act as a trigger for cells of different types to develop in different positions in the embryo. Such chemical systems are known as *reaction-diffusion* systems, and the hypothetical chemical agents are called *morphogens*. Since the introduction of these ideas, several mathematical models of such systems have been studied to see what patterns can be formed and to see how these matched actual animal patterns such as coat spotting

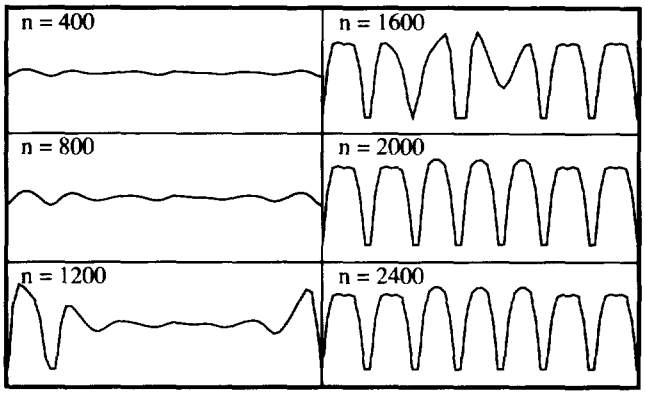

Figure 1: One-dimensional example of reaction-diffusion. Chemical concentration is shown in intervals of 400 time steps.

and stripes on mammals [Bard 81; Murray 81]. Only recently has an actual reaction-diffusion system been observed in the laboratory [Lengyel and Epstein 91]. So far no direct evidence has been found to show that reaction-diffusion is the operating mechanism in the development of any particular embryo pattern. This should not be taken as a refutation of the model, however, because the field of developmental biology is still young and very few mechanisms have been verified to be the agents of pattern formation in embryo development.

The basic form of a simple reaction-diffusion system is to have two chemicals (call them a and b) that diffuse through the embryo at different rates and that react with each other to either build up or break down a and b. These systems can be explored in any dimension. For example, we might use a one-dimensional system to look at segment formation in worms, or we could look at reaction-diffusion on a surface for spot-pattern formation. Here are the equations showing the general form of a two-chemical reaction-diffusion system:

$$\frac{\partial a}{\partial t} = F(a,b) + D_a \nabla^2 a$$

$$\frac{\partial b}{\partial t} = G(a,b) + D_b \nabla^2 b$$

The first equation says that the change of the concentration of a at a given time depends on the sum of a function F of the local concentrations of a and b and the diffusion of a from places nearby. The constant D_a says how fast a is diffusing, and the Laplacian $\nabla^2 a$ is a measure of how high the concentration of a is at one location with respect to the concentration of a nearby. If nearby places have a higher concentration of a, then $\nabla^2 a$ will be positive and a diffuses toward this position. If nearby places have lower concentrations, then $\nabla^2 a$ will be negative and a will diffuse away from this position.

The key to pattern formation based on reaction-diffusion is that an initial small amount of variation in the chemical concentrations can cause the system to be unstable initially and to be driven to a stable state in which the concentrations of a and b vary across the surface. A set of equations that Turing proposed for generating patterns in one dimension provides a specific example of reaction-diffusion:

$$\Delta a_i = s\,(16 - a_i\, b_i) + D_a\,(a_{i+1} + a_{i-1} - 2a_i)$$
$$\Delta b_i = s\,(a_i\, b_i - b_i - \beta_i) + D_b\,(b_{i+1} + b_{i-1} - 2b_i)$$

These equations are given for a discrete model, where each a_i is one "cell" in a line of cells and where the neighbors of this cell are a_{i-1} and a_{i+1}. The values for β_i are the sources of slight irregularities in chemical concentration across the line of cells. Figure 1 illustrates the progress of the chemical concentration of b across a line of 60 cells as its concentration varies over time. Initially the values of a_i and b_i were set to 4 for all cells along the line. The values of β_i were clustered around 12, with the values varying randomly by ± 0.05. The diffusion constants were set to $D_a = .25$ and $D_b = .0625$, which means that a diffuses more rapidly than b, and $s = 0.03125$. Notice how after about 2000 iterations the concentration of b has settled down into a pattern of peaks and valleys. The simulation results look different in detail to this when a different random seed is used for β_i, but such simulations have the same characteristic peaks and valleys with roughly the same scale to these features.

Reaction-Diffusion on a Grid

The reaction-diffusion system given above can also be simulated on a two-dimensional field of cells. The most common form for such a simulation is to have each cell be a square in a regular grid, and have a cell diffuse to each of its four neighbors on the grid. The discrete form of the equations is:

$$\Delta a_{i,j} = s\,(16 - a_{i,j}\, b_{i,j}) + D_a\,(a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1} - 4a_{i,j})$$
$$\Delta b_{i,j} = s\,(a_{i,j}\, b_{i,j} - b_{i,j} - \beta_{i,j}) + D_b\,(b_{i+1,j} + b_{i-1,j} + b_{i,j+1} + b_{i,j-1} - 4b_{i,j})$$

In this form, the value of $\nabla^2 a$ at a cell is found by summing each of the four neighboring values of a and subtracting four times the value of a at the cell. Each of the neighboring values for a are given the same weight in this computation because the length of the shared edge between any two cells is always the same on a square grid. This will not be the case when we perform a similar computation on a less regular grid, where different neighbors will be weighted differently when calculating $\nabla^2 a$.

Figure 2 (upper left) shows the result of a simulation of these equations on a 64×64 grid of cells. Notice that the valleys of concentration in b take the form of spots in two dimensions. It is the nature of this system to have high concentrations for a in these spot regions where b is low. Sometimes chemical a is called an *inhibitor* because high values for a in a spot region prevent other spots from forming nearby. In two-chemical reaction-diffusion systems the inhibitor is always the chemical that diffuses more rapidly.

We can create spots of different sizes by changing the value of the constant s for this system. Small values for s ($s = 0.05$ in Figure 2, upper left) cause the reaction part of the system to proceed more slowly relative to the diffusion and this creates larger spots. Larger values for s produce smaller spots ($s = 0.2$ in Figure 2, upper right). The spot patterns at the top of Figure 2 were generated with $\beta_{i,j} = 12 \pm 0.1$. If the random variation of $\beta_{i,j}$ is increased to 12 ± 3, the spots become more irregular in shape (Figure 3, upper left). The patterns that can be generated by this reaction-diffusion system were extensively studied in [Bard and Lauder 74] and [Bard 81].

Reaction-diffusion need not be restricted to two-chemical systems. For the generation of striped patterns, Meinhardt has proposed a system involving five chemicals that react with one another [Meinhardt 82]. See the appendix of this paper for details of Meinhardts's system. The result of simulating such a system on a two-dimensional grid can be seen in Figure 3 (lower left). Notice that the system generates random stripes that tend to fork and sometimes form islands of one color or the other. This pattern is like random stripes found on some tropical fish and is also similar to the pattern of right eye and left eye regions of the ocular dominance columns found in mammals [Hubel and Wiesel 79].

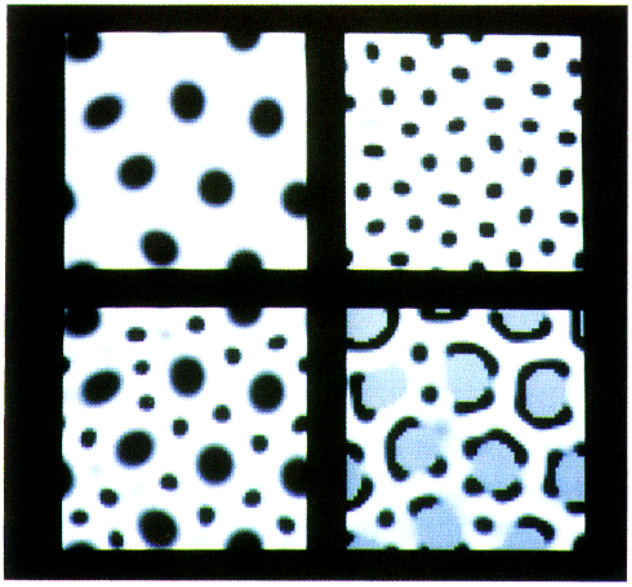

Figure 2: Reaction-diffusion on a square grid. Large spots, small spots, cheetah and leopard patterns.

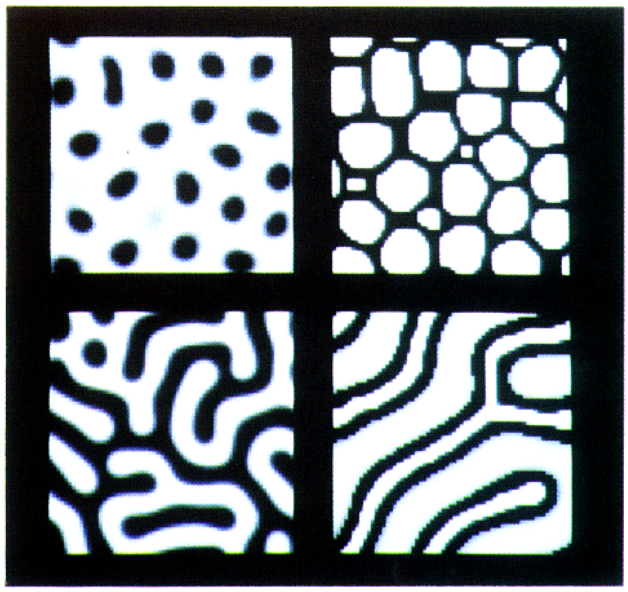

Figure 3: Irregular spots, reticulation, random stripes and mixed large-and-small stripes.

Complex Patterns

This section shows how we can generate more complex patterns using reaction-diffusion by allowing one chemical system to set down an initial pattern and then having this pattern refined by simulating a second system. One model of embryogenesis of the fruit fly shows how several reaction-diffusion systems might lay down increasingly refined stripes to give a final pattern that matches the segmentation pattern in the embryo [Hunding 90]. Bard has suggested that such a *cascade process* might be responsible for some of the less regular coat patterns of some mammals [Bard 81], but he gives no details about how two chemical systems might interact. The patterns shown in this section are new results that are inspired by Bard's idea of cascade processes.

The upper portion of Figure 2 shows how we can change the spot size of a pattern by changing the size parameter s of Turing's reaction-diffusion system from 0.05 to 0.2. The lower left portion of Figure 2 demonstrates that these two systems can be combined to create the large-and-small spot pattern found on cheetahs. We can make this pattern by running the large spot simulation, "freezing" part of this pattern, and then running the small spot simulation in the unfrozen portion of the computation mesh. Specifically, once the large spots are made (using $s = 0.05$) we set a boolean flag *frozen* to TRUE for each cell that has a concentration for chemical b between 0 and 4. These marked cells are precisely those that form the dark spots in the upper left of Figure 2. Then we run the spot forming mechanism again using $s = 0.2$ to form the smaller spots. During this second phase all of the cells marked as frozen retain their old values for the concentrations of a and b. These marked cells must still participate in the calculation of the values of the Laplacian for a and b for neighboring cells. This allows the inhibitory nature of chemical a to prevent the smaller spots from forming too near the larger spots. This final image is more natural than the image we would get if we simply superimposed the top two images of Figure 2.

We can create the leopard spot pattern of Figure 2 (lower right) in much the same way as we created the cheetah spots. We lay down the overall plan for this pattern by creating the large spots as in the upper left of Figure 2 ($s = 0.05$). Now, in addition to marking as frozen those cells that form the large spots, we also change the values of chemicals a and b to be 4 at these marked cells. When we run the second system to form smaller spots ($s = 0.2$) the small spots tend to form in the areas adjacent to the large spots. The smaller spots can form near the large spots because the inhibitor a is not high at the marked cells. This texture can also be seen on the horse model in Figure 4.

In a manner analogous to the large-and-small spots of Figure 2 (lower left) we can create a pattern with small stripes running between larger stripes. The stripe pattern of Figure 3 (lower right) is such a pattern and is modelled after the stripes found on fish such as the lionfish. We can make the large stripes that set the overall structure of the pattern by running Meinhardt's stripe-formation system with diffusion rates of $D_g = 0.1$ and $D_s = 0.06$ (see Appendix). Then we mark those cells in the white stripe regions as frozen and run a second stripe-forming system with $D_g = 0.008$ and $D_s = 0.06$. The slower diffusion of chemicals g_1 and g_2 (a smaller value for D_g) causes thinner stripes to form between the larger stripes.

We can use both the spot and stripe formation systems together to form the web-like pattern called *reticulation* that is found on giraffes. Figure 3 (upper right) shows the result of first creating slightly irregular spots as in Figure 3 (upper left) and then using the stripe-formation system to make stripes between the spots. Once again we mark as frozen those cells that comprise the spots. We also set the concentrations of the five chemicals at the frozen cells to the values found in the white regions of the patterns made by the stripe-formation system. This causes black stripes to form between the marked cells when the stripe-formation system is run as the second step in the cascade process. Figure 5 is an example of how such a texture looks on a polyhedral model.

Regular Stripe Patterns

The chemical system that produces random stripes like those in Figure 3 (lower left) can also be used to produce more regular stripe patterns. The random stripes are a result of the slight random perturbations in the "substrate" for the chemical system. If these random perturbations are removed so the system starts with a completely homogeneous substrate, then no stripes will form anywhere. Regular stripes will form on a mesh that is homogeneous everywhere except at a few "stripe initiator" cells, and the stripes will

radiate from these special cells. One way to create an initiator cell is to slightly raise or lower the substrate value at that cell. Another way is to mark the cell as frozen and set the value of one of the chemicals to be higher or lower than at other cells. The pseudo-zebra in Figure 6 was created in this manner. Its stripes were initiated by choosing several cells on the head and one cell on each of the hooves, marking these cells as frozen and setting the initial value of chemical g_i at these cells to be slightly higher than at other cells.

Varying Parameters Across a Surface

On many animals the size of the spots or stripes varies across the coat. For example, the stripes on a zebra are more broad on the hind quarters than the stripes on the neck and legs. Bard has suggested that, after the striped pattern is set, the rate of tissue growth may vary at different locations on the embryo [Bard 77]. This effect can be approximated by varying the diffusion rates of the chemicals across the computation mesh. The pseudo-zebra of Figure 6 has wider stripes near the hind quarters than elsewhere on the model. This was accomplished by allowing the chemicals to diffuse more rapidly at the places where wider stripes were desired.

Part Two: Mesh Generation and Rendering

This section describes how to generate a mesh over a polyhedral model that can be used for texture synthesis and that will lend itself to high-quality image generation. The strength of this technique is that no explicit mapping from texture space to an object's surface is required. There is no texture distortion. There is no need for a user to manually assign texture coordinates to the vertices of polygons. Portions of this section will describe how such a mesh can be used to simulate a reaction-diffusion system for an arbitrary polyhedral model. This mesh will serve as a replacement to the regular square grids used to generate Figures 2 and 3. We will create textures by simulating a reaction-diffusion system directly on the mesh. It is likely that these meshes can also be used for other forms of texture synthesis. Such a mesh can be used for texture generation wherever a texture creation method only requires the passing of information between neighboring texture elements (mesh cells).

There are a wide variety of sources for polyhedral models in computer graphics. Models generated by special-effects houses are often digitized by hand from a scale model. Models taken from CAD might be created by conversion from constructive solid geometry to a polygonal boundary representation. Some models are generated procedurally, such as fractals used to create mountain ranges and trees. Often these methods will give us few guarantees about the shapes of the polygons, the density of vertices across the surface or the range of sizes of the polygons. Sometimes such models will contain very skinny polygons or vertices where dozens of polygons meet. For these reasons it is unwise to use the original polygons as the mesh to be used for creating textures. Instead, a new mesh needs to be generated that closely matches the original model but that has properties that make it suitable for texture synthesis. This mesh-generation method must be robust in order to handle the wide variety of polyhedral models used in computer graphics.

Mesh generation is a common problem in finite-element analysis, and a wide variety of methods have been proposed to create meshes [Ho-Le 88]. Automatic mesh generation is a difficult problem in general, but the requirements of texture synthesis will serve to simplify the problem. We only require that the model be divided up into relatively evenly-spaced regions. The mesh-generation technique described below divides a polyhedral surface into cells that abut one another and fully tile the polyhedral model. The actual description of a cell consists of a position in $\mathbf{R}^3$, a list of adjacent cells and a list of scalar values that tell how much diffusion occurs between this cell and each of its neighbors. No explicit geometric

Figure 4: Leopard-Horse

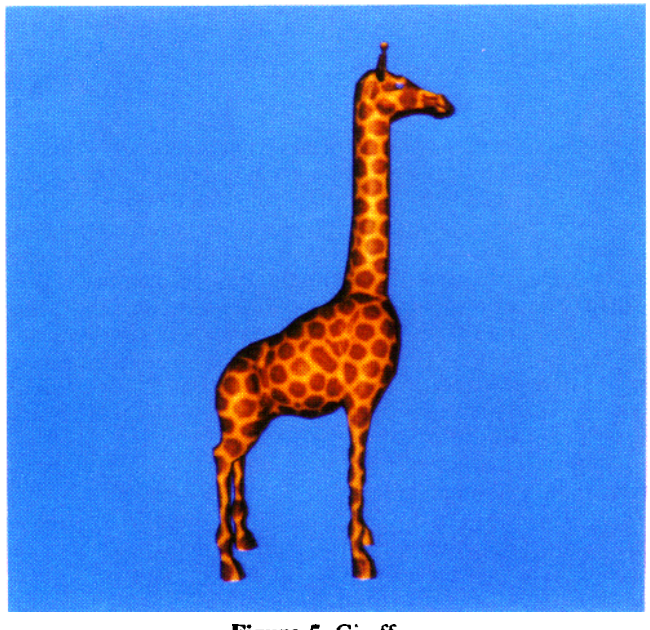

Figure 5: Giraffe

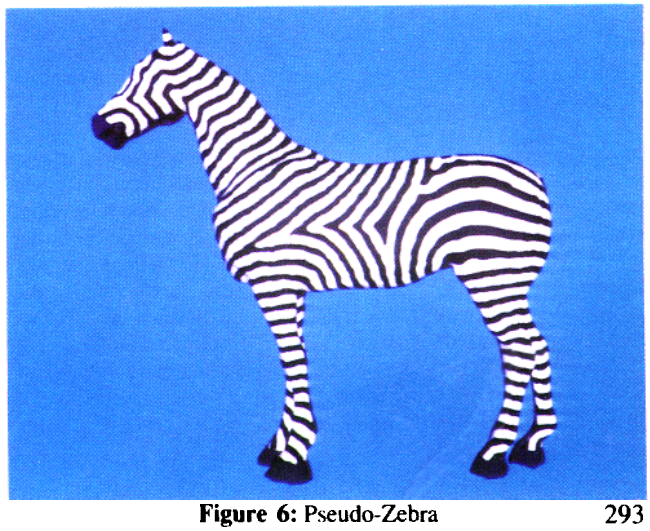

Figure 6: Pseudo-Zebra

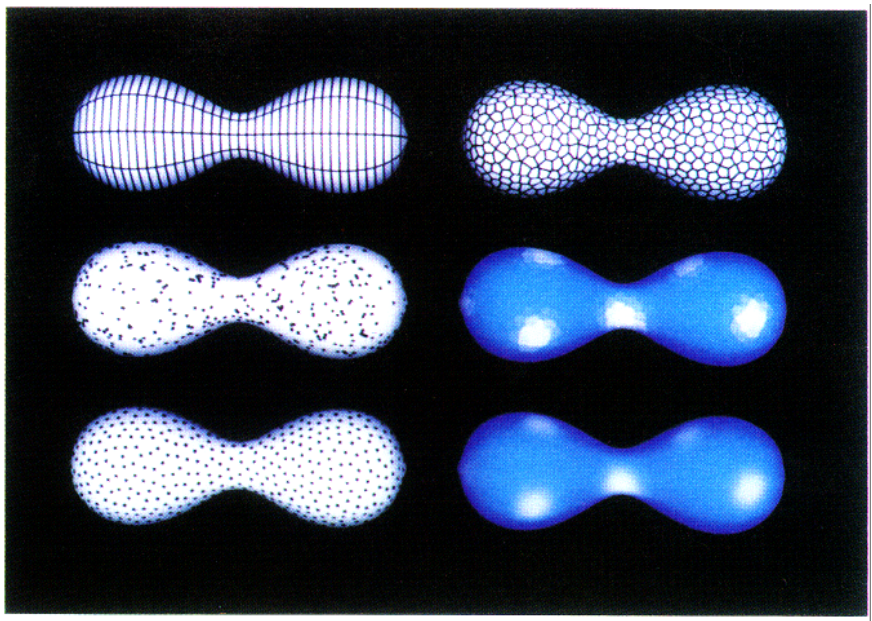

Figure 7: Mesh construction.

representation of the boundaries of these cells is necessary. Given a value for cell density, the mesh-generation method first randomly distributes the appropriate number of points on the surface of the polyhedral model. A relaxation procedure then moves these points across the surface until they are fairly evenly spaced from one another. At this stage, each point gives an (x,y,z) position that is a part of the description of a single cell. The region surrounding each point is examined to determine which pairs of cells will diffuse to one another, and the result of this step gives the adjacency information and the diffusion coefficients that complete the cell's description. The only user-supplied parameter for this mesh-generation method is the desired density of cells in the mesh.

Relaxation of Random Points

The first step in mesh generation is to distribute n points randomly on the model's surface. In order to distribute points randomly over a polyhedral model, care must be taken so that the probability of having a point deposited at any one location is the same everywhere on the surface. To place a random point on the model we need to make an area-weighted choice of the polygon on which a point should be placed. This can be accomplished using a binary search through a list of partial sums of the polygon areas in the model. Now a random point on this polygon can be chosen [Turk 90].

Once the proper number of points has been randomly placed across the surface, we need to move the points around until they are somewhat regularly spaced. This is accomplished using relaxation. Intuitively, the method has each point push around other points on the surface by repelling neighboring points. The method requires choosing a repulsive force and a repulsive radius for the points. We use a repulsive force that falls off linearly with distance. Two points that are greater than the repulsive radius r from one another do not affect each other. The relaxation method also requires a method for moving a point that is being pushed across the surface, especially if the point is pushed off its original polygon. Here is an outline of the relaxation process:

loop k times
  for each point P on surface
    determine nearby points to P
    map these nearby points onto the plane
      containing the polygon of P
    compute and store the repulsive forces that the
      mapped points exert on P
  for each point P on surface
    compute the new position of P based on the
      repulsive forces

Each iteration moves the points into a more even distribution across the polyhedron. Figure 7b shows an initially random distribution of 1000 points over a polyhedral model, and Figure 7c gives the positions of the same points with $k = 40$ iterations of the relaxation procedure. The underlying polygons of the model are outlined in Figure 7a.

The repulsive radius of the points should be chosen based on the average density of the points across the whole surface. The meshes used in this paper were created using a radius of repulsion given by:

$$r = 2\sqrt{a / n}$$

a = area of surface
n = number of points on surface

The above value for r gives a fixed average number of neighboring points to any point, independent of the number of points on the surface and independent of surface geometry. This is important because uniform spatial subdivision can then be used to find neighboring points quickly.

To compute how nearby points repel a given point P on polygon A, these other points are mapped onto the plane containing polygon A. Points that already lie on polygon A remain where they are. Points

on polygons that share an edge with A are rotated about the common edge until they lie within the given plane. Points on more remote polygons are first rotated about the nearest edge of A and then projected onto the plane. We use this method for mapping nearby points onto the plane because of its simplicity. A different method, at the cost of execution speed and algorithm complexity, would be to search for a geodesic path between P and a given nearby point and then to unfold along this path.

Making the points repel one another becomes straightforward once we can map nearby points onto a given point's plane. For each point P on the surface we need to determine a vector S that is the sum of all repelling forces from nearby points. The new position for the point P on polygon A will be $P' = P + kS$, where k is some small scaling value. In many cases the new point P' will lie on A. If P' is not on A, it will often not even lie on the surface of the polyhedron. In this case, we determine which edge of A that P' was "pushed" across and also find which polygon (call it B) that shares this edge with A. The point P' can be rotated about the common edge between A and B so that it lies in the plane of B. This new point may not lie on the polygon B, but we can repeat the procedure to move the point onto the plane of a polygon adjacent to B. Each step of this process brings the point nearer to lying on a polygon and eventually this process will terminate. Most polygons of a model should have another polygon sharing each edge, but some polygons may have no neighbor across one or more edges. If a point is "pushed" across such an edge, the point should be moved back onto the nearest position still on the polygon.

Mesh Cells from Voronoi Regions

The positions of these points become the locations of the mesh cells once relaxation has evened out the distribution of points on the surface. Now regions need to be formed around each point to determine adjacency of cells and to give the diffusion coefficients between adjacent cells. In keeping with many finite-element mesh-generation techniques, we choose to use the Voronoi regions of the points to form the regions surrounding the points. A description of Voronoi regions can be found in a book on computational geometry, e.g., [Melhorn 84]. Given a set of points S in a plane, the Voronoi region of a particular point P is that region on the plane where P is the closest point of all points in S. For points on a plane, the Voronoi regions will always be bounded by line segments positioned halfway between pairs of points. When we simulate a diffusing system on such a set of cells, we will use the lengths of the edges separating pairs of cells to determine how much of a given chemical can move between the two cells. Figure 7d shows the Voronoi regions for the set of points shown in Figure 7c.

Finding the exact Voronoi regions of the points on a polyhedral surface is not simple since one of these regions might be parts of several different polygons. Instead of solving this exactly, a planar variation of the exact Voronoi region for a point is used to determine the lengths of edges between cells. Using the same procedure as before, all points near a given point P are mapped onto the plane of the polygon A containing P. Then the planar Voronoi region of P is constructed and the lengths of the line segments that form the region are calculated. It is the lengths of these segments that are used as the diffusion coefficients between pairs of cells. In general, computing the Voronoi regions for n points in a plane has a computational complexity of $O(n \log n)$ [Melhorn 84]. However, the relaxation process distributes points evenly over the surface of the object so that all points that contribute to the Voronoi region of a point can be found by looking only at those points within a small fixed distance from that point. In practice we have found that we need only consider those points within $2r$ of a given point to construct a Voronoi region, where r is the radius of repulsion used in the relaxation process. Because uniform spatial subdivision can be used to find these points in a constant amount of time, constructing the Voronoi regions is of $O(n)$ complexity in this case.

The above construction of the Voronoi regions assumes that the diffusion over a surface is isotropic (has no preferred direction). The striking textures in [Witkin and Kass 91] show that simulation of anisotropy can add to the richness of patterns generated with reaction-diffusion. Given a vector field over a polyhedral surface, we can simulate anisotropic diffusion on the surface if we take into account the anisotropy during the construction of the Voronoi regions. This is done by contracting the positions of nearby points in the direction of anisotropy after projecting neighboring points onto a given point's plane. Then the Voronoi region around the point is constructed based on these new positions of nearby points. The contraction affects the lengths of the line segments separating the cells, and thus affects the diffusion coefficients between cells. This contraction will also affect which cells are neighbors. Figure 8 shows that anisotropic diffusion creates spots that are stretched when Turing's system is simulated on the surface of a model.

Reaction-Diffusion on a Mesh

We can create any of the reaction-diffusion patterns described earlier on the surface of any polyhedral model by simulating the appropriate chemical system directly on a mesh for the model. The square cells of a regular grid are now replaced by the Voronoi regions that comprise the cells of the mesh. Simulation proceeds exactly as before except that calculation of the Laplacian terms now takes into account that the segments that form the boundaries of the cells are not all the same length. These boundary lengths are the diffusion coefficients, and the collection of coefficients at each cell should be normalized so they sum to one. $\nabla^2 a$ is computed at a particular cell by multiplying each diffusion coefficient of the cell by the value of a at the corresponding neighboring cell, summing these values for all neighboring cells, and subtracting the value of a at the given cell. This value should then be multiplied by four to match the feature sizes generated on the regular square grid. When the simulation is complete, we have a concentration for each participating chemical at each cell in the mesh. The next section tells how these concentrations are rendered as textures.

Rendering

Once the simulation on a mesh is finished, we require a method for displaying the resulting chemical concentrations as a texture. First, we need a smooth way of interpolating the chemical concentrations across the surface. The chemical value can then be used as input to a function that maps chemical concentration to color. We have chosen to let the chemical concentration at a location be a weighted sum of the concentrations at mesh points that fall within a given radius of the location. If the chemical concentration at a nearby mesh cell Q is $v(Q)$, the value $v'(P)$ of an arbitrary point P on the surface is:

$$v'(P) = \frac{\sum_{Q \text{ near } P} v(Q)\, w(|P - Q| / s)}{\sum_{Q \text{ near } P} w(|P - Q| / s)}$$

The weighting function w can be any function that monotonically decreases in the range zero to one. The function used for the images in this paper is:

$$w(d) = 2d^3 - 3d^2 + 1 \qquad \text{if } 0 \le d \le 1$$
$$w(d) = 0 \qquad \text{if } d > 1$$

This function falls smoothly from the value 1 down to 0 in the range [0,1], and its first derivative is zero at 0 and at 1 [Perlin and Hoffert 89]. Any similar function that falls off smoothly could be used for

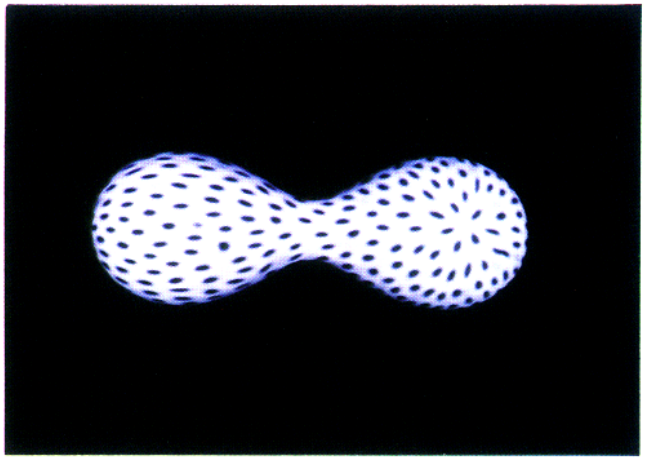

Figure 8: Anisotropic diffusion

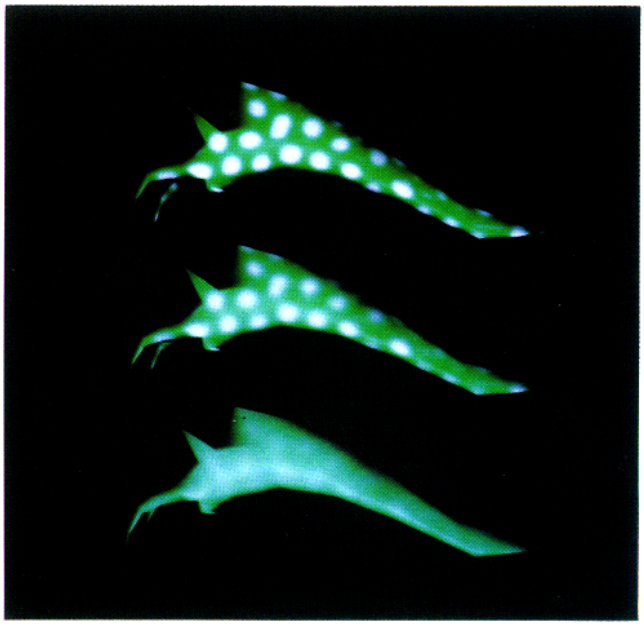

Figure 9: Blur levels for anti-aliasing

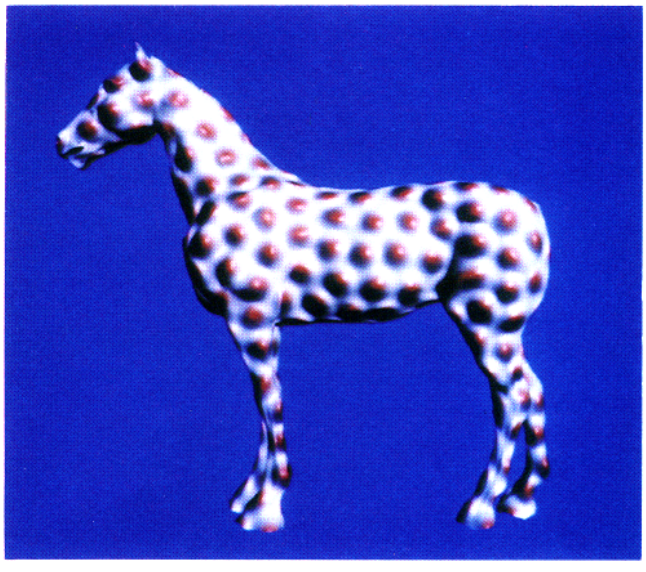

Figure 10: Bump mapping

the weighting function. The images in this paper have been made using a value of $s = 2r$, where r is the repulsive radius from the relaxation method. Much smaller values for s make the individual mesh points noticeable, and values much larger than this will blur together the values of more nearby mesh points. Uniform spatial subdivision makes finding nearby mesh points a fast operation because only those mesh points within a distance s contribute to a point's value. Figure 7e shows the individual cell values from a pattern generated by reaction-diffusion, where the chemical concentrations have been mapped to a color gradation from blue to white. Figure 7f shows the surface colors given by the weighted average for $v'(P)$ described above.

The method described above gives smooth changes of chemical concentration across the surface of the model, and rendered images do not show evidence of the underlying mesh used to create the texture. Aliasing of the texture can occur, however, when a textured object is scaled down small enough that the texture features, say stripes, are about the same width as the pixels of the image. Super-sampling of the texture is one possible way to lessen this problem, but computing the texture value many times to find the color at one pixel is costly. A better approach is to extend the notion of levels of increasingly blurred textures [Williams 83] to those textures defined by the simulation mesh.

The blurred versions of the texture are generated using the simulation mesh, and each mesh point has associated with it an RGB (red, green, blue) color triple for each blur level. Level 0 is an unblurred version of the texture and is created by evaluating the concentration-to-color function at the mesh points for each concentration value and storing the color as an RGB triple at each mesh point. Blur levels 1 and higher are created by allowing these initial color values to diffuse across the surface. When the values of a two-dimensional gray-scale image are allowed to diffuse across the image, the result is the same as convolving the original image with a Gaussian filter. Larger amounts of blurring (wider Gaussian filters) are obtained by diffusing for longer periods of time. Similarly, allowing the RGB values at the mesh points to diffuse across the mesh results in increasingly blurred versions of the texture given on the mesh. The relationship between diffusion for a time t and convolution with a Gaussian kernel of standard deviation σ is $t = \sigma^2 / 2$ [Koenderink 84]. The blur levels of Figure 9 were generated so that each level's Gaussian kernel has a standard deviation twice that of the preceding blur level.

The texture color at a point is given by a weighted average between the colors from two blur levels. The choice of blur levels and the weighting between the levels at a pixel is derived from an approximation to the amount of textured surface that is covered by the pixel. This estimate of surface area can be computed from the distance to the surface and the angle the surface normal makes with the direction to the eye. The natural unit of length for this area is r, the repulsion radius for mesh building. The proper blur level at a pixel is the base two logarithm of the square root of a, where a is the area of the surface covered by the pixel in square units of r. We have produced short animated sequences using this anti-aliasing technique and they show no aliasing of the textures.

Bump mapping is a technique used to make a surface appear rough or wrinkled without explicitly altering the surface geometry [Blinn 78]. The rough appearance is created from a gray-scale texture by adding a perturbation vector to the surface normal and then evaluating the lighting model based on this new surface normal. Perlin showed that the gradient of a scalar-valued function in $\mathbf{R}^3$ can be used as a perturbation vector to produce convincing surface roughness [Perlin 85]. We can use the gradient of the values $v'(P)$ of a reaction-diffusion simulation to give a perturbation vector at a given point P:

$d = r / 100$
$gx = (v'(P) - v'(P + [d,0,0])) / d$
$gy = (v'(P) - v'(P + [0,d,0])) / d$
$gz = (v'(P) - v'(P + [0,0,d])) / d$
perturbation vector = $[k * gx, k * gy, k * gz]$

The above method for computing the gradient of v' evaluates the function at P and at three nearby points in each of the x, y and z directions. The value d is taken to be a small fraction of the repulsive radius r to make sure we stay close enough to P that we get an accurate estimate for the gradient. The gradient can also be computed directly from the definition of v' by calculating exactly the partial derivatives in x, y and z. The scalar parameter k is used to scale the bump features, and changing k's sign causes bumps to become indentations and vice versa. Figure 10 shows bumps created in this manner based the results of a reaction-diffusion system.

Implementation and Performance

Creating a texture using reaction-diffusion for a given model can be a CPU-intensive task. Each of the textures in Figures 4, 5 and 6 took several hours to generate on a DEC 3100 workstation. These meshes contained 64,000 points. Perhaps there is some consolation in the thought that nature requires the same order of magnitude in time to lay down such a pattern in an embryo. Such texture synthesis times would seem to prohibit much experimenting with reaction-diffusion textures. It is fortunate that a given reaction-diffusion system with a particular set of parameters produces the same texture features on small square grids as the features from a simulation on much larger meshes. The patterns in this paper were first simulated on a 64×64 grid of cells where the simulations required less than a minute to finish. These simulations were run on a Maspar MP-I, which is a SIMD computer with 4096 processing elements connected in a two-dimensional grid. A workstation such as a DEC 3100 can perform similar simulations on a 32×32 grid in about two minutes, which is fast enough to explore new textures. Once a texture is generated by reaction-diffusion, the time to render the model with a texture is reasonably fast. The image in Figure 4 required 70 seconds to render at 512×512 resolution without anti-aliasing on a DEC 3100. The same horse without texture takes 16 seconds to render.

Future Work

The cascade processes that formed the textures in this paper are just a few of the patterns that can be generated by reaction-diffusion. More exploration should be made on how one chemical system can leave a pattern for later systems. For example, one chemical system could affect the random substrate of a second system. What patterns can be formed if one system causes different rates of diffusion in different locations in a second system?

Other methods of pattern creation could be performed on the meshes used for texture synthesis. Examples that might be adapted from cellular automata [Toffoli and Margolus 87] include two-dimensional annealing, diffusion-limited aggregation and the Belousov-Zhabotinskii reaction.

Acknowledgments

I would like to thank those people who have offered ideas and encouragement for this work. These people include David Banks, Henry Fuchs, Albert Harris, Steve Molnar, Brice Tebbs, and Turner Whitted. Thanks also for the suggestions provided by the anonymous reviewers. Linda Houseman helped clean up my writing and David Ellsworth provided valuable photographic assistance. Thanks to Rhythm & Hues for the horse, Steve Speer for the giraffe and Apple Computer's Vivarium Program for the sea-slug.

This work was supported by a Graduate Fellowship from IBM and by the Pixel-Planes project. Pixel-Planes is supported by the National Science Foundation (MIP-9000894) and the Defense Advanced Research Projects Agency, Information Science and Technology Office (Order No. 7510).

Appendix: Meinhardt's Stripe-Formation System

The stripes of Figure 3 (lower images) and Figure 6 were created with a five-chemical reaction-diffusion system given in [Meinhardt 82]. The equations of Meinhardt's system are as follows:

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

$$\frac{\partial g_2}{\partial t} = \frac{c s_1 g_2^2}{r} - \alpha g_2 + D_g \frac{\partial^2 g_2}{\partial x^2} + \rho_0$$

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

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

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

In this system, the chemicals g_1 and g_2 indicate the presence of one or the other stripe color (white or black, for instance). The concentration of r is used to make sure that only one of g_1 and g_2 are present at any one location. Chemicals s_1 and s_2 assure that the regions of g_1 and g_2 are limited in width. A program written in FORTRAN to simulate this system can be found in [Meinhardt 82].

References

[Bard 77] Bard, Jonathan, "A Unity Underlying the Different Zebra Striping Patterns," *Journal of Zoology*, Vol. 183, No. 4, pp. 527–539 (December 1977).

[Bard 81] Bard, Jonathan B. L., "A Model for Generating Aspects of Zebra and Other Mammalian Coat Patterns," *Journal of Theoretical Biology*, Vol. 93, No. 2, pp. 363–385 (November 1981).

[Bard and Lauder 74] Bard, Jonathan and Ian Lauder, "How Well Does Turing's Theory of Morphogenesis Work?," *Journal of Theoretical Biology*, Vol. 45, No. 2, pp. 501–531 (June 1974).

[Bier and Sloan 86] Bier, Eric A. and Kenneth R. Sloan, Jr., "Two-Part Texture Mapping," *IEEE Computer Graphics and Applications*, Vol. 6, No. 9, pp. 40–53 (September 1986).

[Blinn 78] Blinn, James F., "Simulation of Wrinkled Surfaces," *Computer Graphics*, Vol. 12, No. 3 (SIGGRAPH '78), pp. 286–292 (August 1978).

[Bloomenthal 85] Bloomenthal, Jules, "Modeling the Mighty Maple," *Computer Graphics*, Vol. 19, No. 3 (SIGGRAPH '85), pp. 305–311 (July 1985).

[Catmull 74] Catmull, Edwin E., "A Subdivision Algorithm for Computer Display of Curved Surfaces," Ph.D. Thesis, Department of Computer Science, University of Utah (December 1974).

[Gardner 85] Gardner, Geoffrey Y., "Visual Simulation of Clouds," *Computer Graphics*, Vol. 19, No. 3 (SIGGRAPH '85), pp. 297–303 (July 1985).

[Hanrahan and Haeberli 90] Hanrahan, Pat and Paul Haeberli, "Direct WYSIWYG Painting and Texturing on 3D Shapes," *Computer Graphics*, Vol. 24, No. 4 (SIGGRAPH '90), pp. 215–223 (August 1990).

[Heckbert 89] Heckbert, Paul S., "Fundamentals of Texture Mapping and Image Warping," M.S. Thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley (June 1989).

[Ho-Le 88] Ho-Le, K., "Finite Element Mesh Generation Methods: A Review and Classification," *Computer Aided Design*, Vol. 20, No. 1, pp. 27–38 (January/February 1988).

[Hubel and Wiesel 79] Hubel, David H. and Torsten N. Wiesel, "Brain Mechanisms of Vision," *Scientific American*, Vol. 241, No. 3, pp. 150–162 (September 1979).

[Hunding 90] Hunding, Axel, Stuart A. Kauffman, and Brian C. Goodwin, "*Drosophila* Segmentation: Supercomputer Simulation of Prepattern Hierarchy," *Journal of Theoretical Biology*, Vol. 145, pp. 369–384 (1990).

[Koenderink 84] Koenderink, Jan J., "The Structure of Images," *Biological Cybernetics*, Vol. 50, No. 5, pp. 363–370 (August 1984).

[Lengyel and Epstein 91] Lengyel, István and Irving R. Epstein, "Modeling of Turing Structures in the Chlorite–Iodide–Malonic Acid–Starch Reaction System," *Science*, Vol. 251, No. 4994, pp. 650–652 (February 8, 1991).

[Lewis 84] Lewis, John-Peter, "Texture Synthesis for Digital Painting," *Computer Graphics*, Vol. 18, No. 3 (SIGGRAPH '84), pp. 245–252 (July 1984).

[Lewis 89] Lewis, J. P., "Algorithms for Solid Noise Synthesis," *Computer Graphics*, Vol. 23, No. 3 (SIGGRAPH '89), pp. 263–270 (July 1989).

[Ma and Gagalowicz 85] Ma, Song De and Andre Gagalowicz, "Determination of Local Coordinate Systems for Texture Synthesis on 3-D Surfaces," *Eurographics '85*, edited by C. E. Vandoni.

[Meinhardt 82] Meinhardt, Hans, *Models of Biological Pattern Formation*, Academic Press, London, 1982.

[Melhorn 84] Melhorn, Kurt, *Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, 1984.

[Murray 81] Murray, J. D., "On Pattern Formation Mechanisms for Lepidopteran Wing Patterns and Mammalian Coat Markings," *Philosophical Transactions of the Royal Society B*, Vol. 295, pp. 473–496.

[Peachey 85] Peachey, Darwyn R., "Solid Texturing of Complex Surfaces," *Computer Graphics*, Vol. 19, No. 3 (SIGGRAPH '85), pp. 279–286 (July 1985).

[Perlin 85] Perlin, Ken, "An Image Synthesizer," *Computer Graphics*, Vol. 19, No. 3 (SIGGRAPH '85), pp. 287–296 (July 1985).

[Perlin and Hoffert 89] Perlin, Ken and Eric M. Hoffert, "Hypertexture," *Computer Graphics*, Vol. 23, No. 3 (SIGGRAPH '89), pp. 253–262 (July 1989).

[Samek 86] Samek, Marcel, Cheryl Slean and Hank Weghorst, "Texture Mapping and Distortion in Digital Graphics," *The Visual Computer*, Vol. 2, No. 5, pp. 313–320 (September 1986).

[Toffoli and Margolus 87] Toffoli, Tommaso and Norman Margolus, *Cellular Automata Machines*, MIT Press, 1987.

[Turing 52] Turing, Alan, "The Chemical Basis of Morphogenesis," *Philosophical Transactions of the Royal Society B*, Vol. 237, pp. 37–72 (August 14, 1952).

[Turk 90] Turk, Greg, "Generating Random Points in Triangles," in *Graphics Gems*, edited by Andrew Glassner, Academic Press, 1990.

[Williams 83] Williams, Lance, "Pyramidal Parametrics," *Computer Graphics*, Vol. 17, No. 3 (SIGGRAPH '83), pp. 1–11 (July 1983).

[Witkin and Kass 91] Witkin, Andrew and Michael Kass, "Reaction-Diffusion Textures," *Computer Graphics*, Vol. 25 (SIGGRAPH '91).

[Yeager and Upson] Yeager, Larry and Craig Upson, "Combining Physical and Visual Simulation — Creation of the Planet Jupiter for the Film 2010," *Computer Graphics*, Vol. 20, No. 4 (SIGGRAPH '86), pp. 85–93 (August 1986).