1 Introduction

In 1952, the mathematician and pioneering computer scientist Alan Turing proposed a chemical mechanism by which patterns of tissue could be created in a developing embryo [Turing 52]. This mechanism, called *reaction-diffusion,* is still a contender among the theories of pattern formation in the embryonic field of developmental biology. As examples of reaction-diffusion, Turing showed how such a system could be used to create patterns like the arrangement of tentacles on a hydra and like the dappling patterns found on cows. His use of the Manchester Mark I computer for pattern generation and a cathode ray tube for displaying these patterns qualifies him as an early practitioner in the field of computer graphics [Hodges 83, p. 445].

This paper shows how Turing's reaction-diffusion mechanism can be used to generate textures for use in computer graphics. A number of researchers have explored the range of patterns that can be generated by reaction-diffusion, but the results of these simulations have not been in a form usable for high quality image generation. This paper is aimed at filling this gap. The paper begins by covering previous approaches to texture generation and texture mapping. Then the basic ideas behind reaction-diffusion are introduced, and examples are given of spot and striped patterns that can be created by reaction-diffusion models. A detailed description is given on how these patterns can be generated across the surface of an arbitrary polyhedral model. Then a method is presented for displaying these patterns without aliasing artifacts. The final section describes possible directions for research to widen the range of patterns that can be generated by reaction-diffusion.

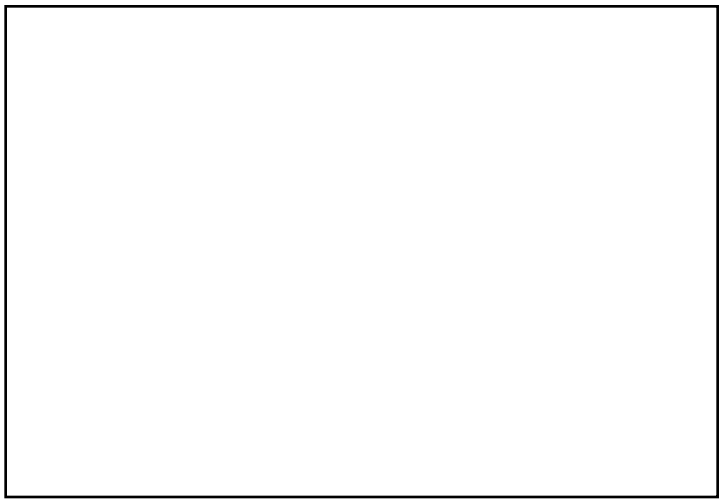

Figure 1: Spot pattern on a sea slug generated by reaction-diffusion.

2 Texture Mapping

Texture mapping was introduced by [Catmull 74] as a way to add visual detail to the surface of an object without explicitly modeling the fine geometry of the surface. A texture map is a function that associates a color to each point on an object's surface. Texture mapping can be extended to simulate reflection [Blinn and Newell 76] and rough or wrinkled surfaces [Blinn 78]. One important issue in texture mapping is how the texture is acquired. Often a texture is just a scanned image of a photograph. Other textures are generated algorithmically, and this has the advantage that one has some control over the features of the texture through the algorithm parameters.

2.1 Artificial Texture Synthesis

Several methods have been proposed 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 perturb 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 anisotropic noise functions to be used for texture synthesis.

Of course, 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]. That paper reports that textures such as canvas and wood grain can be created by this method.

2.2 Mapping Textures onto Surfaces

Once a texture has been created, a method is needed to map it onto the surface to be textured. Often a texture is represented as a two-dimensional array of color values, and one can think of such a texture as resting in the unit square $[0,1] \times [0,1]$. Mapping these values 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 unit square 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 may 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 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 3-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 3-space 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. This approach was taken by [Samek 86], where the surface of a polygonal 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 polygonal object.

Each of the above methods have 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 square to an object's surface. 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. One solution to this problem is the use of solid textures.

A *solid texture* is a color function defined over a portion of 3-space, 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 3-space.

2.3 Textures from Simulation

None of the methods for texture synthesis described above attempt to model the actual physical processes that produce patterns in the real world. We believe that generation of graphical models will come to rely more on physical simulation as the demand continues for greater realism in computer images. Some of the textures generated by function composition produce images that look quite real, but some physical phenomena are likely to prove too difficult to mimic without modeling the underlying processes that create the texture. One stunning example of using physical simulation for texture creation is the dynamic cloud patterns of Jupiter in the movie 2010 [Yaeger and Upson 86]. Another example of how physical simulation can be used to generate textures is the texture synthesis method using reaction-diffusion that is presented in this paper.

The texture generation method described in this paper creates a texture pattern by simulation on the

surface of the target object. Because texture generation is performed on the object's surface, there is no later step necessary to map from texture space to object space. This means there is no texture distortion. This should also result in texture features that arise as a result of local surface geometry. For example, simulation of some reaction-diffusion systems on a long cone show patterns of spots at the wide end of the cone but a change towards a pattern of stripes as the cone tapers [Murray 81]. This trend of spots-to-stripes is found on the tails of some mammals. Another example of how texture is influenced by geometry is the triangular pattern found where the stripes of a zebra merge at the joining of the legs with the body.

3 Reaction-Diffusion Systems

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. Turing gave the name *morphogens* to these hypothetical chemical agents of embryo development, and such chemical systems are known as *reaction-diffusion* systems. Since the introduction of these ideas, several 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 and stripes on mammals [Bard 81] [Murray 81]. 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.

3.1 A Simple System

The basic form of a simple reaction diffusion system is to have two chemicals, call them x and y, that diffuse through the embryo at different rates and that react with each other to either build up or break down x and y. 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:

$$\frac{\partial x}{\partial t} = F(x,y) + D_x \nabla^2 x$$

$$\frac{\partial y}{\partial t} = G(x,y) + D_y \nabla^2 y$$

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

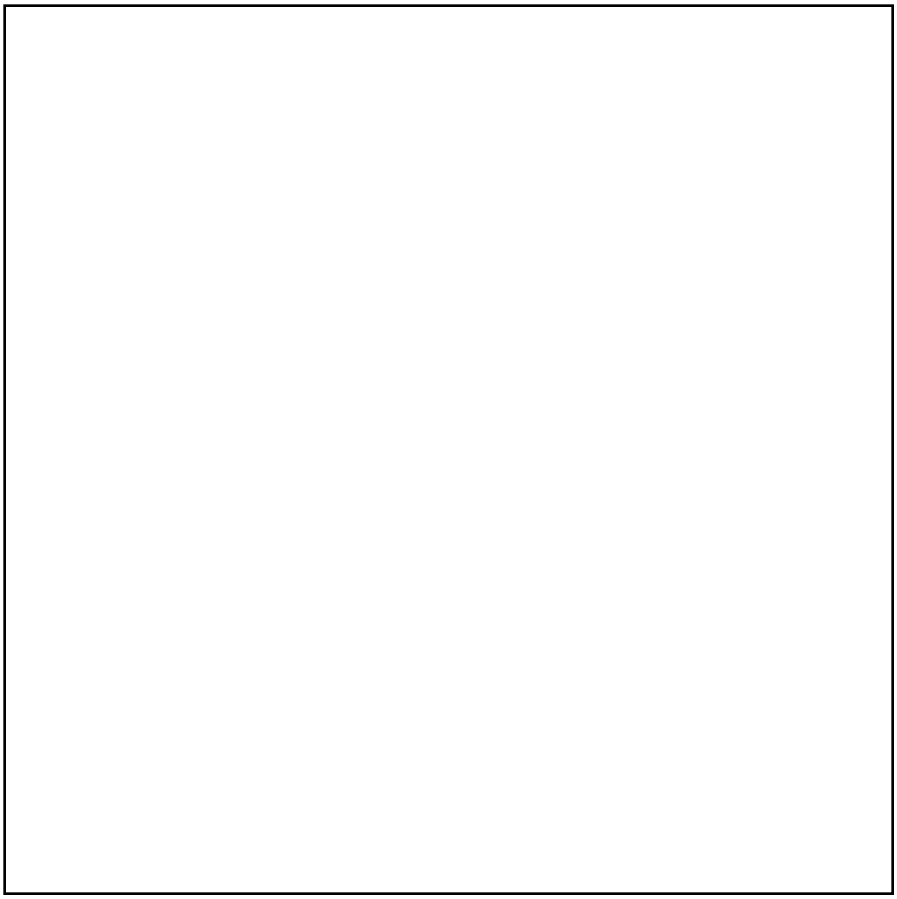

Figure 2: One-dimensional example of reaction-diffusion. Chemical concentration is shown in intervals of 220 time steps.

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 initially be unstable and be driven to a stable state in which the concentrations of x and y vary across the surface. To give a specific example of reaction-diffusion, here is a set of equations that Turing proposed for generating patterns in one dimension:

$$\Delta x_i = s\,(16 - x_i\,y_i) + D_x\,(x_{i+1} + x_{i-1} - 2x_i)$$
$$\Delta y_i = s\,(x_i\,y_i - y_i - \beta_i) + D_y\,(y_{i+1} + y_{i-1} - 2y_i)$$

These equations are given for a discrete model, where each x_i is one "cell" in a line of cells and where the neighbors of this cell are x_{i-1} and x_{i+1}. The values for β_i are the sources of slight irregularities in chemical concentration across the line of cells. Figure 2 shows the progress of the chemical concentration of y across a line of 60 cells as its concentration varies over time. Initially the value of

x_i and y_i were set to 4 for all cells along the line. The value of β_i were clustered around 12, with the values varying randomly by ± 0.05. The diffusion constants were set to D_x = .25 and D_y = .0625, which means that x diffuses more rapidly than y, and s = 0.03125. Notice how after about 2000 iterations the concentration of y 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.

3.2 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 are:

$$\Delta x_{i,j} = s\,(16 - x_{i,j}\,y_{i,j}) + D_x\,(x_{i+1,j} + x_{i-1,j} + x_{i,j+1} + x_{i,j-1} - 4x_{i,j})$$
$$\Delta y_{i,j} = s\,(x_{i,j}\,y_{i,j} - y_{i,j} - \beta_{i,j}) + D_y\,(y_{i+1,j} + y_{i-1,j} + y_{i,j+1} + y_{i,j-1} - 4y_{i,j})$$

Figure 3 shows the result of such a simulation on a 64 by 64 grid of cells. Notice that the valleys of concentration in y take the form of spots in two dimensions. Changing the value of the constant s results in spots of different sizes. Both spot patterns of Figure 3 were generated with $\beta_{i,j} = 12 \pm 0.1$. If the random variation of $\beta_{i,j}$ is increased to 12 ± 3 then the spots become more irregular in shape (Figure 4). The patterns that can be generated by this reaction-diffusion system were extensively studied in [Bard and Lauder 74] and [Bard 81].

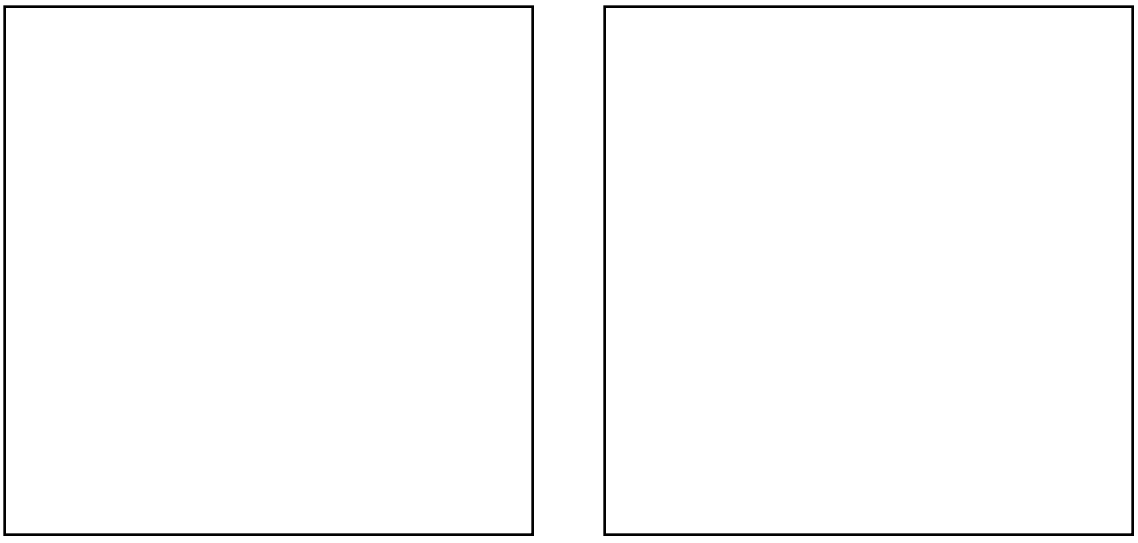

Figure 3: Large and small spots created with reaction-diffusion (s = 0.1 and s = 0.4)

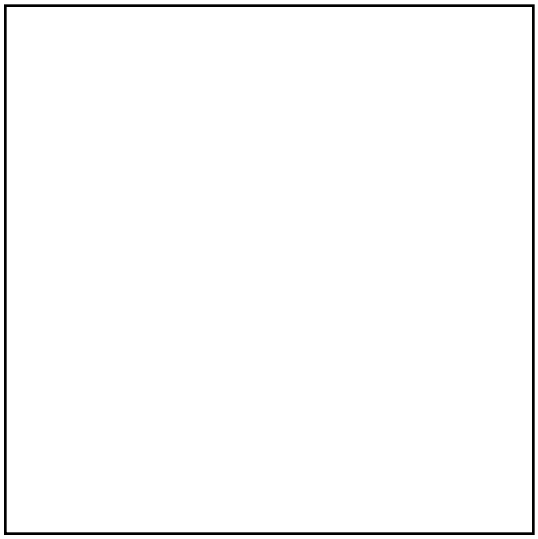

Figure 4: More irregular spots can be created by increasing the variation of $\beta_{i,j}$.

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, p.125]. Two of the chemicals are for indicating the presence of one or the other stripe color (white or black, for instance), another chemical makes sure only one or the other of the first two chemicals are present in any one location, and the last two chemicals determine the widths of the stripes. The details of the equations and a complete FORTRAN program to generate patterns using this system can be found in [Meinhardt 82]. The result of simulating such a system on a two-dimensional grid can be seen in Figure 5. Notice that the system generates random stripes that tend to fork and sometimes generate 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]. Generating stripes that are more regular, such as those found on zebras, has proved to be more difficult with a reaction-diffusion model, although there are a couple of methods that show promise [Bard 81].

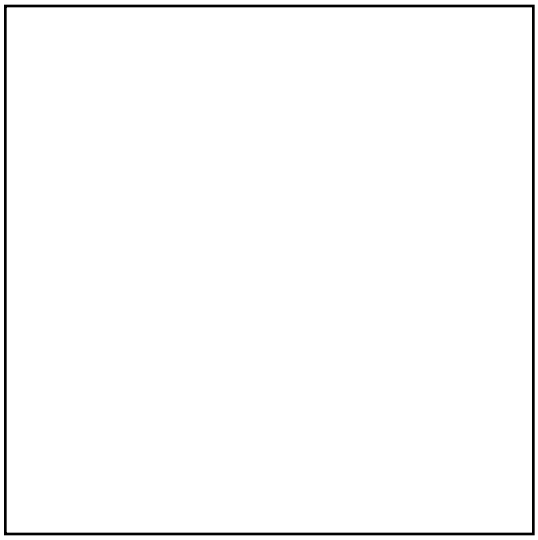

Figure 5: Random stripes created by a five chemical reaction-diffusion system.

4 Meshes on Arbitrary Polyhedral Surfaces

The goal of the work presented in this paper is to simulate reaction-diffusion systems on the surfaces of models to be used for high-quality image generation. For example, we may have a model of a cheetah that needs a spot pattern for its coat. How can the simulation technique that works on a square grid be extended to arbitrary surfaces? The surfaces used in this work have been restricted to the class of polyhedral models with faces that are convex polygons. Closed form solutions do not exist for reaction-diffusion equations on polyhedral surfaces. The alternative to this is to subdivide the surface and solve the equations on this discrete mesh. Thus we seek an automatic method for breaking up the surface of a polyhedral model into a mesh consisting of cells and interfaces between these cells. 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 reaction-diffusion systems will serve to simplify the problem.

4.1 Requirements for Mesh Generation

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. None of these methods give us any 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 have anomalies such as edges that belong to only one polygon 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 simulation. Instead, a new mesh needs to be generated that closely matches the original model but that has properties that make it suitable for simulation of chemical diffusion. This mesh generation method must be robust in order to handle the wide variety of polyhedral models used in computer graphics.

One requirement for a mesh to be used for reaction-diffusion is that the cells should all be roughly the same size. The patterns generated by reaction-diffusion systems have feature sizes that do not vary widely across the surface. For a particular set of parameters, the spots and stripes generated by the systems described in section 3 fall within a fairly narrow size range. If mesh cells are too large the fine features will be lost. Simulating on a mesh with cells much smaller than the feature size of the generated pattern is a waste of compute time. Uniform cell size, then, is our first requirement for mesh generation. Another requirement is for the shapes of the cells to be fairly regular so that the chemicals will diffuse isotropically across the surface. Ideally we would like a mesh to be formed by cells all of exactly the same shape, such as regular squares or hexagons. Unfortunately, this is not possible for arbitrary models.

With the requirements of roughly uniform cell size and shape in mind, this paper describes a fully automatic mesh generation procedure to be used for simulation of reaction-diffusion systems over polyhedral surfaces. The method first randomly distributes points on a model and then moves these points over the surface until they are evenly distributed. Then the region surrounding each point is examined to determine which pairs of cells will diffuse to one another, and the result of this step is the final mesh on which the simulation will take place. The only user-supplied parameter for this method is the desired density of cells across the mesh. Even this density parameter could be determined automatically from equation coefficients if the feature sizes resulting from a reaction-diffusion system is known, but this is not yet automated in the mesh generation system.

4.2 Random Points Across a Polyhedron

Distributing points randomly over a polyhedral model is non-trivial. Care must be taken so that the probability of having a point deposited at any one location is the same everywhere on the surface. The first step is to make an area-weighted choice of the polygon on which a point should be placed. Let each polygon P_i in the model have an area A_i, then let the values S_i be the sum of the areas A_i of all polygons up to and including P_i. To choose a random polygon, first pick a random value r in the range from zero to the total surface area of the model. Use binary search over the list of values S_i to determine which polygon the value r specifies. Now a random point on this polygon needs to be chosen.

In what follows we will assume that we are picking a random point on a triangle. Any polyhedral model may be triangulated to satisfy this requirement. Call the vertices of the triangle A, B and C. Let s and t be two random values chosen from the interval [0,1]. Then the code below picks a random point Q in the triangle:

if $s + t > 1$ **then**
    $s = 1 - s$
    $t = 1 - t$
endif

$a = 1 - s - t$
$b = s$
$c = t$

$Q = aA + bB + cC$

Without the “if” statement, the point Q will be a random point in the parallelogram with vertices A, B, C and $(B + C - A)$ (Figure 6). A point that lands in the triangle B, C, $(B + C - A)$ is moved into the triangle A, B, C by reflecting it about the center of the parallelogram.

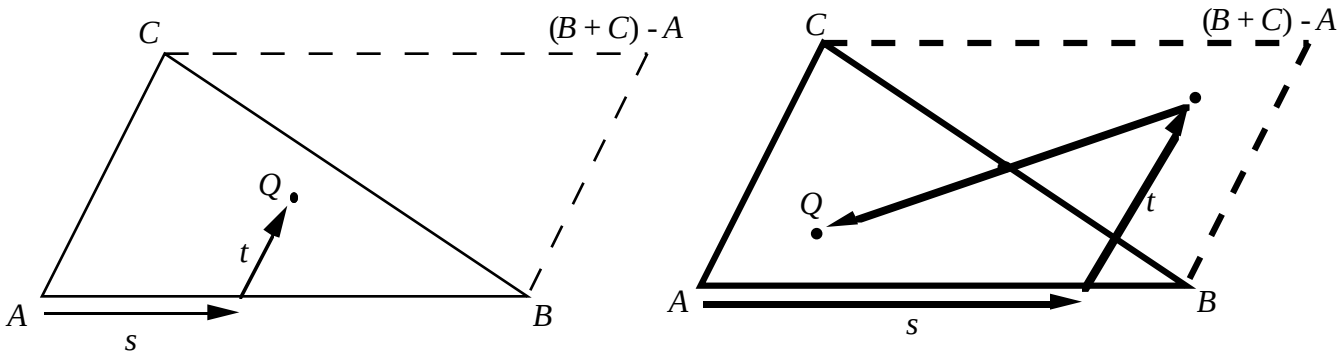

Figure 6: Choosing a random point Q in triangle ABC. The left diagram shows choice of Q when $s + t \leq 1$, and right diagram shows choice when $s + t > 1$.

4.3 Point Relaxation

Once the proper number of points have been randomly placed across the surface, we need to move the points around until they are somewhat regularly spaced. This is accomplished by 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. It 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 pseudo-code giving 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 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 new position of P based on repulsive forces
```

Each iteration moves the points into a more even distribution across the polyhedron. Figure 7 shows an initially random distribution of 200 points in a square and the positions of the same points with k = 50 iterations of the relaxation procedure.

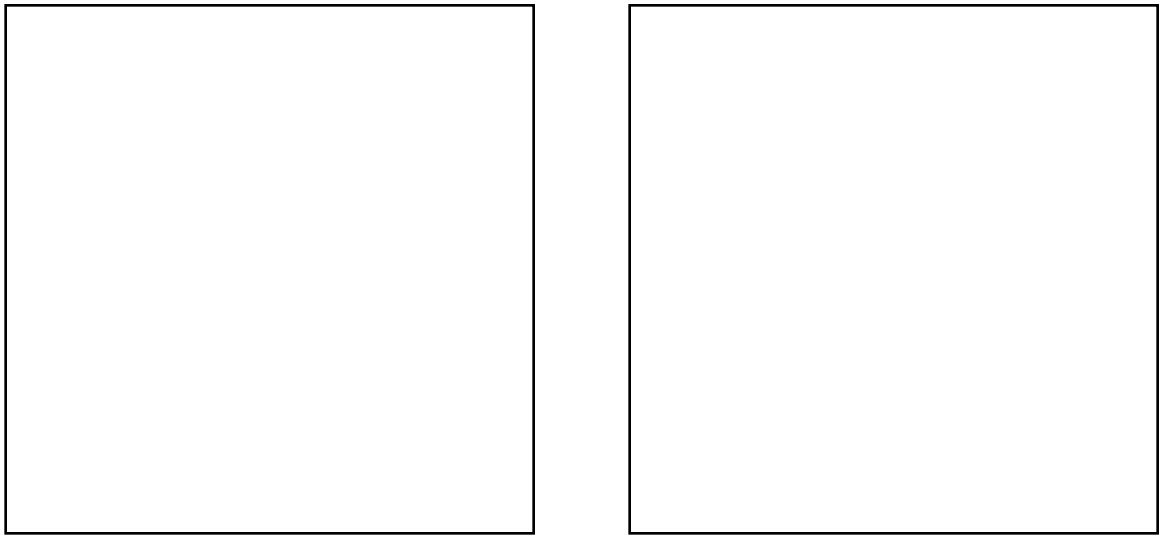

Figure 7: Random points in a square (left) and the points after 50 relaxation steps (right).

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:

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

n = number of points on surface
a = area of 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 quickly find neighboring points. See the appendix for details of this technique.

For repulsion by nearby points, we require a distance function across the surface. For points that lie on the same polygon the Euclidean distance function can be used. For points that lie on different polygons we need to do something reasonable. A logical choice is to pick the shortest distance between the two points over all possible versions of the polyhedral model where each model has been unfolded and flattened onto a plane. Unfortunately, determining the unfolding that minimizes this distance for any two points is not easy. As a compromise, we choose only to use this flattened distance when the two points are on adjacent polygons, and we will punt if the points are further removed than this. We can pre-compute and store a transformation matrix M for adjacent polygons A and B that specifies a rotation about the shared edge that will bring polygon B into the plane of

polygon *A*. With this transformation, the distance between point *P* on polygon *A* and point *Q* on polygon *B* is determined by applying *M* to *Q* and then finding the distance between this new point and *P*.

For points *P* and *Q* on widely separated polygons *A* and *B*, we first find in which direction *Q* lies with respect to the polygon *A*. Then the transformation matrix of the edge associated with this direction is applied to *Q*. Usually this will not bring *Q* into the plane of *A*, so this point is projected onto the plane of *A* and we use the distance between this new point and *P* as our final distance. This gives a distance function across the surface, and in addition it gives a method of making every point near a given point *P* seem as though it lies on the plane of the polygon of *P*. Let the procedure of moving a point *Q* onto polygon *A* be called MapToPlane(*Q*,*A*). Figure 8 give examples of MapToPlane().

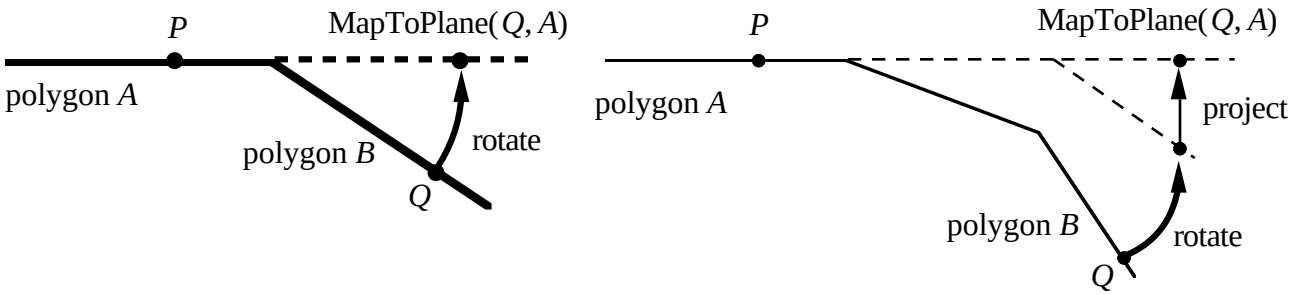

Figure 8: Moving a point *Q* onto the plane of polygon *A* when it is on an adjacent polygon (left) or when it is on a remote polygon (right).

With a distance function in hand, making the points repel each other becomes straightforward. 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. Here is the determination of *S* based on the repulsive radius *r*:

```
S = 0
for each point Q near point P
    map Q onto plane of P's polygon, call the new point R
    V = normalized vector from R to P
    d = distance from R to P
    if d < r then
        S = S + (r - d) V
```

Once this is done for each point on the surface, the points need to be moved to their new positions. The new position for the point *P* on polygon *A* will be $P' = P + kS$, where *k* is some small scaling value. The meshes of this paper were made with $k = 0.15$. In many cases the new point P' will lie on *A*. If P' is not on *A* then 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 and edge then the point should be moved back onto the nearest position still on the polygon.

4.4 Mesh Cells from Voronoi Regions

Once relaxation has evened out the distribution of points on the surface, regions need to be constructed to form cells surrounding each point. The purpose of defining these cells is to know how much of a chemical is to be passed between nearby points in the mesh during a simulation step. In keeping with many finite-element mesh generation techniques, we choose to use the Voronoi regions of the points to form the cells. A description of Voronoi regions can be found in a book on computational geometry such as [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 portions of lines 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 9 shows the Voronoi regions for the set of random points shown earlier.

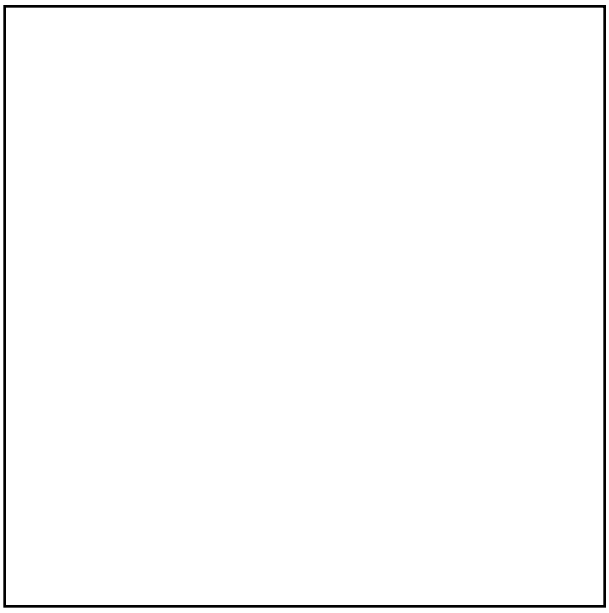

Figure 9: Voronoi regions for the points from Figure 7.

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 will be used to determine the lengths of edges between cells. When the distinction is important this other region will be called the *planar Voronoi region*, to distinguish it from the actual Voronoi region across the surface. Using MapToPlane(), all points near a given point *P* are mapped onto the plane of the polygon containing *P*. We first construct the planar Voronoi region of *P* in the plane of polygon containing *P*, compute the lengths of the edges bounding this region and then store the edge lengths along with a reference to which point shared each edge with the cell of *P*. For many points *P* and *Q* the length of their shared edge as computed from the planar Voronoi region of *P* will agree with the length given by the planar Voronoi region of *Q*. For some points this might not be the case, due to the nature of the unfolding and projection that is used to bring points onto the same plane. The average of the computed lengths is used when the shared edge lengths between a pair of points disagree.

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 at just those points within a small fixed distance from that point. In practice we have found that we need only consider those points within 2*r* of a given point to construct a Voronoi region, where *r* is the radius of repulsion used in the relaxation process. Uniform spatial subdivision can be used to find these points in a constant amount of time, so constructing the Voronoi regions is of *O(n)* complexity in this case.

This automatic mesh generation method has been successfully used to create meshes on a wide variety of polyhedral models. Such models include a sea slug and a horse, an icosahedron and a dodecahedron, a model of UNC's Old Well and a mathematical surface of minimum curvature. Figure 10 shows the Voronoi regions generated by the mesh generation technique for a polyhedral model of a horse.

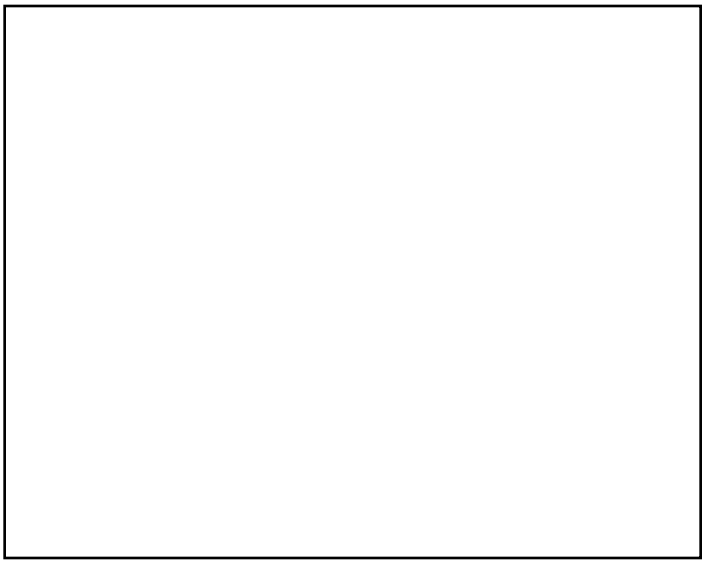

Figure 10: The Voronoi regions of 4000 points distributed over the model of a horse.

5 Simulation on Arbitrary Meshes

Any reaction-diffusion system can be simulated over a surface once a mesh has been generated for the model. The simulation can be performed by Euler integration as was done for square grids in section 3. The only difference between simulation on a square grid and on an arbitrary mesh is that the computation of the diffusion term must take into account the Voronoi region adjacency. On a square grid the diffusion term for a particular cell C is the sum of the concentrations of the four neighboring cells of C minus four times the concentration in C. On an arbitrary mesh, the diffusion term at a point P is the weighted sum of the concentrations of neighboring points of P minus the concentration at P. These weights should sum to one and the diffusion constant D_x for chemical x should be multiplied by four in order to match the feature size of the simulation on a square grid. The weights governing the amount of diffusion between neighboring mesh points should be proportional to the length of the common side between the Voronoi regions of the points. Chemicals do not directly diffuse between mesh points whose Voronoi regions do not share a common edge.

Figures 11 and 12 shows two examples of reaction-diffusion that were simulated on a mesh for a model of a horse. Figure 11 shows a pattern generated by Turing's spot generating equations, and Figure 12 shows the result of Meinhardt's random stripe generation model. Notice how the features of these textures follow the geometry of the surface.

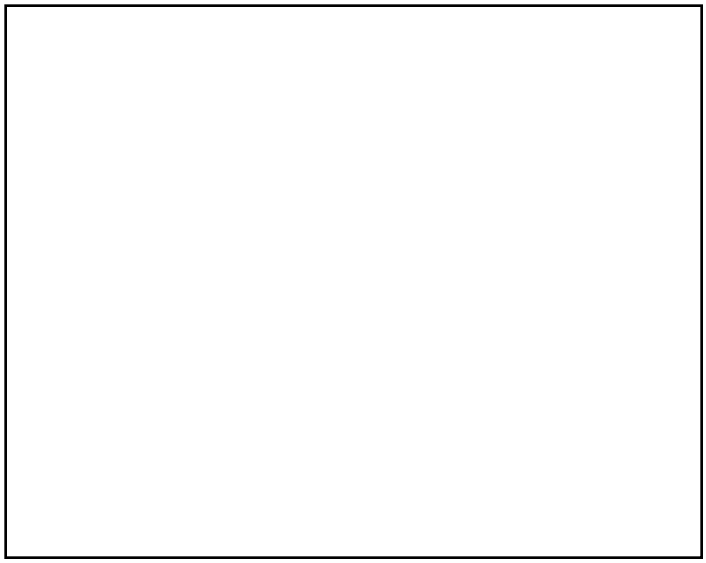

Figure 11: Horse model with spots. Simulation performed on a 16,000 point mesh.

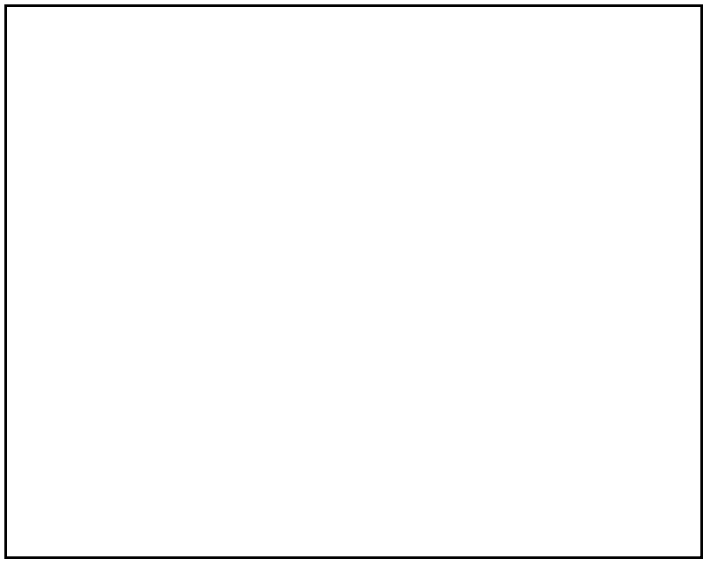

Figure 12: Horse model with random stripes. Simulation performed on a 16,000 point mesh.

6 Rendering of Textures from Meshes

Once the simulation of a reaction-diffusion system is finished, the resulting pattern given by chemical concentrations can be displayed. The chemical concentrations of the system are only specified at the mesh points, so some form of interpolation is needed to extend these values to all points on the surface. Once the chemical concentration of a point is known, this value can be used as an index into a color table. For the purpose of rendering, we have chosen let the chemical concentration at a location be a weighted sum of the mesh points within a given radius of the location. Once again, uniform spatial subdivision can be used to speed the search for nearby points. If the value at a nearby mesh point Q is $V(Q)$, then the value of an arbitrary point P on the surface is:

$$V(P) = \frac{\sum_{Q \text{ near } P} V(Q)\, W(s\,|P - Q|)}{\sum_{Q \text{ near } P} W(s\,|P - Q|)}$$

The weighting function W can be any function that monotonically decreases in the range zero up to a cutoff radius. The function we use was suggested by Brice Tebbs:

$W(d) = 2d^3 - 3d^2 + 1$ if $0 \le d \le 1$
$W(d) = 0$ if $d < 0$ or $d > 1$

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. Any similar function (like a Gaussian) could be used for the weighting function. For efficiency, a table of weighting values can be pre-computed so that the function evaluation can be a quick table lookup. The images in this paper have been made using a value of $s = 1 / 2r$, where r is the repulsive radius from the relaxation method above. This means only those mesh points within $2r$ of a given point will contribute to the value. Much larger values for s make the individual mesh points noticeable, and values much smaller than this tend to blur together the values of more nearby mesh points.

The interpolation method described above gives smooth variation of values across the surface so that any portion of the surface may be magnified without aliasing. If, however, the rendered image of the object is reduced greatly then point sampling of the texture is likely to produce aliasing. Super-sampling of the texture is one possible solution, although this can become costly and will be unsuitable for very small images of the surface. Another possibility, as yet unimplemented, makes use of the fact that diffusion (without reaction) of a set of values on a surface is essentially the same as convolving the values using a Gaussian filter. Thus the same diffusion mechanism that was used to create the pattern

can be used to help generate increasingly blurred versions of the texture to be rendered. When a surface is rendered, the final texture value can be a weighted average of two values taken from appropriately blurred versions of the texture, where the choice of blur levels and the weight between the levels is a function of how small the object appears in the image. Three sets of color values (red, green, blue) for each blurred level will be necessary if the color lookup table from concentration is not a linear ramp. The easiest way to store these blurred versions of the texture is to keep a texture value from each level at each point in the texture mesh. The notion of using increasingly blurred versions of a texture for anti-aliasing is now in common use, and the idea was introduced in [Williams 83].

7 Future Work

The variety of patterns that can be generated by reaction-diffusion needs to be expanded before this technique will become truly useful for computer graphics. So far only the most simple patterns, spots and random stripes, have been created on polyhedral surfaces. The theoretical biology literature describes reaction-diffusion systems that produce other patterns, most notably some ideas about how to create more regular striped patterns [Bard 81], and these methods should be tried on complex surfaces. There are many naturally occurring patterns on animals that might be explained by reaction-diffusion but where the patterns have not yet been matched by equations. Some promising candidates include the spot clusters found on cheetahs and jaguars know as rosettes, the alternating spot and stripe pattern of some squirrels, the large spots of giraffes and the variable-width stripes of the lionfish. Some of these more complex patterns might be explained by a cascade of reaction-diffusion systems. The idea of a cascade system is to have a coarse pattern set down by one system and then have the pattern further refined by a second process. This notion was suggested in [Bard 81], but we have not found any computer simulations of cascade systems in the literature.

Although this paper concentrates on patterns created from reaction-diffusion systems, the basic notion of carrying out simulation on a model's surface could be used for generating quite different patterns by simulating other physical systems. For example, the cracking pattern of bark over a tree's surface might be created by simulating the growth and cracking processes. Other examples of patterns in nature produced by physical processes include lichen growth on rocks, the branching tendrils carved into the ground by erosion and the colorful thin film swirls of oil on water. There are many patterns in nature that are just waiting to find their way into computer-generated images. Let's get simulating!

8 Acknowledgements

I would like to thank those people who have offered ideas and encouragement for this work. These people include David Banks, Henry Fuchs, Lee Nackman and Brice Tebbs. I would especially like to thank Al Harris for stimulating discussions and for many good pointers into the biology literature. Thanks to David Ellsworth for help with the photographs. I would also like to thank Rhythm & Hues for kindly allowing me to use their beautiful horse model, a model which the textures do not yet do justice.

This work was supported by a Graduate Fellowship from IBM and by the Pixel-Planes project. Pixel-Planes is funded by the National Science Foundation, Grant No. DCI-8601152, the Defense Advanced Research Projects Agency, DARPA ISTO Order No. 6090 and the Office of Naval Research, Contract No. N0014-86-K-0680.

Appendix: Hash-Based Uniform Spatial Subdivision

One problem that arose several times during implementation of this work was to find all points from a set that are within a given radius *r* of a position in space. This is needed to find the points to repel during the relaxation process and it is used again to find nearby mesh points to average during texture rendering. In addition, one common polygon format at UNC does not specify which vertices are shared between polygons, so a fast way to match nearly identical vertices is needed when an object description is read from a file. Uniform spatial subdivision with hashing is a simple answer to all of these problems [Bentley and Friedman 79]. When the density of the set of points is bounded this method determines nearby points to a query point in constant time.

The basic idea is to divide up space into cells that have side lengths equal to the radius *r* of search. One way to do this is to allocate a three-dimensional array of such cells that is known to contain all points in the set. An easier solution is to use a large linear array of cells and use a hash function to give a single index into the array. One simple hash function begins by computing the indices that would be used for a three dimensional array of cells, then multiplies two of these indices by primes and finally adds together all these values together. The pseudo-code below outlines the method of uniform spatial subdivision with hashing:

table_size = size of the hash table
I = small prime number (say 17)
J = larger prime (say 101)
s = 1 / *r*

hash function
hash (*x*,*y*,*z*) = (*I*⌊*sx*⌋ + *J*⌊*sy*⌋ + ⌊*sz*⌋) modulo *table_size*

table initialization
for each point *P* = (*x*,*y*,*z*) in the set
    insert *P* in linked list at position *hash* (*x*,*y*,*z*) in the table of cells

search (find points near *P* by looking in cell containing *P* and its eight neighboring cells)
for a = -1 to 1
    for b = -1 to 1
        for c = -1 to 1
            index = *hash* (*x*+*ra*,*y*+*rb*,*z*+*rc*)
            for each point *Q* in the list at *index* in the cell table
                if distance from *P* to *Q* is less than *r*, add *Q* to list of points near *P*

References

[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).

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

[Bentley and Friedman 79] Bentley, Jon Louis and Jerome H. Friedman, "Data Structures for Range Searching," *Computing Surveys*, Vol. 11, No. 4, pp. 397 - 409 (December 1979).

[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 and Newell 76] Blinn, James F. and Martin E. Newell, "Texture and Reflection in Computer Generated Images," *Communications of the ACM*, Vol. 19, No. 10, pp. 542 - 547 (October 1976).

[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).

[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).

[Hodges 83] Hodges, Andrew, *Alan Turing: The Enigma*, Simon & Schuster, 1983.

[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).

[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).

[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).

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

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

[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).