Cellular Texture Generation

Kurt W. Fleischer
David H. Laidlaw
Bena L. Currin
Alan H. Barr

California Institute of Technology
Pasadena, CA 91125

email: {kurt, dhl, bena, barr}@druggist.gg.caltech.edu

Abstract

We propose an approach for modeling surface details such as scales, feathers, or thorns. These types of *cellular textures* require a representation with more detail than texture-mapping but are inconvenient to model with hand-crafted geometry.

We generate patterns of geometric elements using a biologically-motivated cellular development simulation together with a constraint to keep the cells on a surface. The surface may be defined by an implicit function, a volume dataset, or a polygonal mesh. Our simulation combines and extends previous work in developmental models and constrained particle systems.

Key Words: particle systems, developmental models, data amplification, constraints, texture mapping, bump mapping, displacement mapping

1 Introduction

For several years computer graphics researchers and practitioners have been grappling with the problem of creating and displaying surfaces having an organic appearance. Texture maps, bump maps, and related methods often attain the *appearance* of detailed geometry without actually creating it. These techniques do not suffice, however, when the viewpoint is close enough that the three-dimensional (3-D) geometric structure of a surface texture is apparent.

We are interested in making images of surfaces covered with interacting geometric elements, such as scales, feathers, thorns, and fur. We model these elements as small 3-D cells constrained to lie on a surface. The cells interact to form *cellular textures*: surface textures with 3-D geometry, orientation, and color. Our approach combines properties of particle systems, developmental models, and reaction-diffusion methods into one system. Figure 7 shows an example combining all of these approaches.

There are a few challenges in making images of these types of materials:

- The geometry is often too pronounced for using texture- or bump-maps.
- It is often difficult to map appropriate texture coordinates onto the global geometry and topology.
- The placement, orientation, coloration, and shape of the individual elements may depend on:
  - neighboring elements,
  - surface characteristics such as local curvature, or
  - global phenomena such as sunlight.

Figure 1: Thorny Head: Both flat and thorn-shaped cells are constrained to lie on a surface defined by a polygonal dataset of a human head. Flat cells are used in the neck and chest regions, while thorn-shaped cells are used on the head. The orientation of each thorn approaches that of its neighbors, leading to a continuous field of thorns that sweeps across the head. The size of the thorns is related to the level of detail of the model; smaller thorns are placed on smaller features.

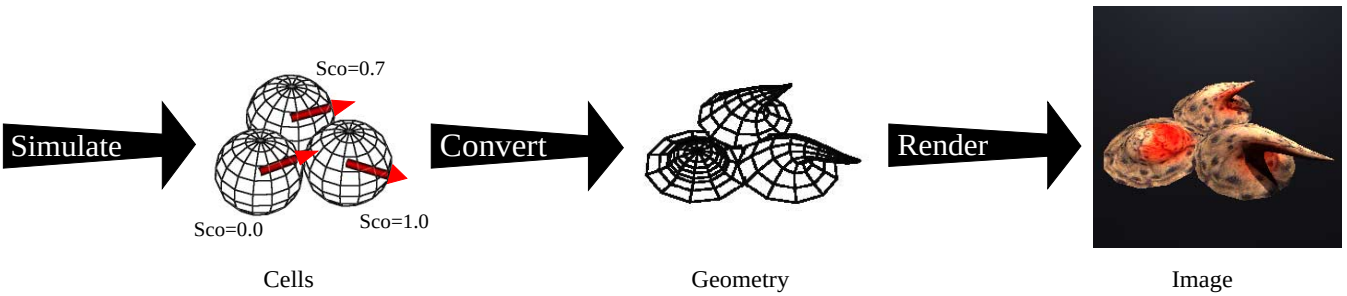

Figure 2: The cellular particle simulator computes the locations, orientations, and other values associated with the cells. This information is converted to geometry and appearance parameters, which is then passed to a renderer to create the image. Note that the cell orientations (red arrows) become the orientations of the thorns. Using a geometric modeler, we created a geometric object that changes shape from a bump to a thorn based on a single parameter [31]. We use the cell state variable S_{c0} to control this parameter (Section 4).

Because of the potentially complicated interdependencies of the elements, it is difficult to create either geometric or textural models of such objects by hand. So we turn to automatic data-amplification techniques, which are similar to the structured particle systems used to generate models of plants [26, 30].

Developmental Approach For generating organic patterns, it is natural to consider a biologically-based simulation. In previous work [8], we developed a biological developmental model to simulate and study patterns generated by the motions and interactions of discrete cells (Figure 3). These artificial cells move about, grow, and divide in a simulated petri dish, the *extracellular environment.* The extracellular environment can contain physical barriers, diffusing chemicals, gravity, etc. The ability to form a variety of interesting patterns with the system has prompted us to explore its application to geometric texture generation (Figure 1).

The textures we model are formed from many interacting geometric elements. Actual fur, scales, and thorns may be formed from single cells or multiple cells [19]. In either case, we assume that the texture patterns arise from the interactions of discrete elements capable of movement and orientation change, and model each of these elements as one cell. The patterns are formed as the cells experience physical processes of collision, adhesion, and other local interactions.

Software Structure The approach advocated by this paper is to automatically grow cellular textures by simulating discrete cells on surfaces. We then convert the resulting cellular information into model geometry and coloration, which is rendered. The images of this paper were generated using oriented, spherical cells, which are converted into thorns, scales, and other shapes for rendering (Figure 2).

Overview The remainder of this paper is structured as follows. Section 2 describes related work. It is followed by an overview of the system architecture in Section 3. Section 4 describes the cellular particle system, and includes examples of cell programs that implement various behaviors. In Section 5 we discuss the particle converter, which produces geometry from the cell positions, orientations and other parameters.

Results are presented in Section 6, which describes the examples shown in the figures. The final section presents a discussion of the approach and some directions for future work.

2 Related Work

This approach is a synthesis and extension of work ranging from morphological models to general texture mapping. In this section, we discuss our approach in the context of four related areas:

- Levels of Detail
- Biologically Motivated Morphogenesis
- Reaction-Diffusion Methods
- Particle Systems

Levels of Detail Choosing the appropriate level of detail for image synthesis at a given viewing distance has long been recognized as an important topic in computer graphics [4, 14, 15]. At large scales, geometric models are necessary; intermediate scales, texture mapping and similar techniques may be sufficient; at the smallest scales, illumination models suffice to describe the microgeometry of the object [38].

The level of detail of the models addressed in this paper falls somewhere between the use of hand-crafted geometric models and bump- or texture-mapping. A range of geometric levels is available to us because of the modular nature of our technique.

Complex, oriented textures have been created and rendered in many ways, notably with texels [15]. The texel approach is intermediate between geometry and mapping techniques, but leaves open the question of how to arrange the texel elements appropriately. Our approach addresses this problem, and can produce models to be rendered using texels.

Displacement mapping is another technique for adding geometric detail to surfaces [3]. As with texels, the displacement mapping technique does not address the problem of determining which displacements are necessary to create a specific effect, such as a field of similarly oriented thorns. A possible application of our technique

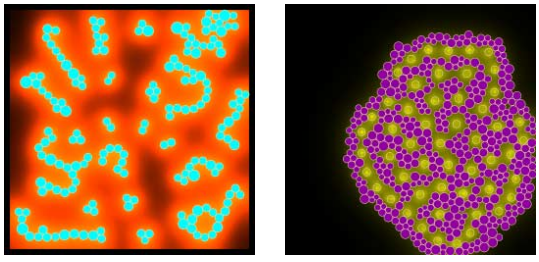

Figure 3: These images demonstrate the pattern formation capabilities of our 2-D cell simulator [8].

is to create such displacement maps, for example by creating flow fields [21].

Biologically Motivated Morphogenesis The cellular development system which forms the basis of this work [6, 7, 8] incorporates elements of several established biological models of morphogenesis: Turing's morphogens [35], Odell's mechanical models [20], and Lindenmayer-system cell lineage determinants [24], as well as our own model of cell contact and adhesion.

Much well-known computer graphics work is biologically based. The combination of developmental models with geometric constraints enables the creation of many organic patterns. It has been explored in work on plant growth [13, 23], plant organ placement [10], and seashell patterning [9].

Interacting geometric elements were used by [10] to model the placement of plant organs. Our cells are a generalization of these elements, with many additional capabilities, including independent movement, adhesion, and changes in size and orientation due to cell-cell interaction.

In [9], pigmentation patterns on seashells are modeled using reaction-diffusion equations on surfaces defined by sweeping a generating curve along a logarithmic spiral. This shares with our work the concept of applying pattern formation models on 3-D surfaces. Their use of continuous reaction-diffusion equations to generate the patterns differs from our use of discrete cells. For the types of cellular texture we are investigating, the choice of a discrete model seems appropriate.

Spatially-oriented models of plant growth are capable of generating attractive plant images [1, 13]. The placement of geometric objects in the environment of plants affects their growth. The importance of combining environmental and endogenous mechanisms in forming organic shapes in computer graphics has also been demonstrated using environmentally-sensitive L-systems [23], which allow interaction between the environment and the development of a structure defined by an L-system. In an application to synthetic topiary, elements sense their global position and orientation, and are pruned according to a bounding surface. Our work also combines geometric environmental factors with an endogenous developmental model to describe cell behavior. We differ from these plant models by the use of discrete motile cells that are able to move and rotate independently.

Reaction-Diffusion Methods Reaction-diffusion equations were first proposed as a model for morphogenesis by Turing [35]. They are a continuous approximation to a sheet of many discrete cells interacting over time. Our system models discrete cells explicitly, and can generate patterns similar to continuous reaction-diffusion equations since it is actually a more detailed model of the same biological system.

Reaction-diffusion equations have been successfully applied to the generation of texture maps [36, 40]. Because they are based on natural phenomena, they have an appealing organic quality. In addition, they avoid problems of parameterization and topology by creating the pattern directly on a surface. Our approach shares both of these benefits.

In our 2-D implementation (Figure 3), we include both discrete cells and a continuous reaction-diffusion computation. The two models are also able to interact, since the discrete cells can sense and emit the continuously diffusing chemicals. The 3-D implementation does not yet support continuously diffusing and reacting chemicals. However, we are able to reproduce some forms of reaction-diffusion behavior using cell-cell interaction in the discrete model.

Particle Systems Early particle systems [25, 28] had little or no interparticle interaction, unlike later work based on molecular models and other criteria [17, 34]. Our work includes elements of Witkin and Heckbert's surface-constrained particles [41], and the orientation constraints of Szeliski and Tonneson [33]. Reynold's "boids" [27] introduced somewhat more sophisticated interacting particles with programmable behaviors. In addition, his boids, like our interacting cells, can sense and react to each other and to their environment.

3 Software Architecture

We arrange the process into three software modules (Figure 2).

cellular particle simulator with surface constraints: computes locations, orientations, sizes and other parameters of cells based on a behavioral specification. Allows particles to be constrained to a surface (implicit, polygon mesh, or volume dataset isosurface).

parameterized particle-to-geometry converter: converts cell positions, orientations and other parameters into shape and appearance parameters.

renderer: takes shape and appearance parameters plus a scene description and renders the scene. The images in this paper were generated with John Snyder's ray tracer [32], which was chosen primarily for its speed on large datasets.

The implementation of our framework involves the addition of cell-cell interactions, orientation constraints and surface constraints to a more traditional particle simulator. A simple version of a particle converter can be implemented using a geometric modeler to place a geometric object at each particle's location with the appropriate size and orientation. The cellular particle simulator and particle converter are described in further detail in the Sections 4 and 5.

4 Cellular Particle Simulator

The cellular particle system combines cell-cell interactions, cell-cell adhesion, oriented particles, and surface constraints into one unified framework. Additional discussion of the cell simulator and its implementation can be found in [7, 8].

Our system allows the user to specify cell behaviors such as 'go to a surface' and 'align with neighbors' by combining modular *cell programs*. Cell programs are first-order differential equation terms that modify the cell's state (see Table 1).

In Section 4.1 we discuss how to use the simulator, and then delve into a more detailed mathematical description of the cell programs in Section 4.2. Section 4.3 describes the cell programs used to create simulations like those shown in the figures. Section 4.4 presents methods for incorporating various types of surfaces into our surface constraint method.

4.1 Using the Simulator

For a particular simulation run, the user defines

- the cell state variables,
- the extracellular environment, and
- the cell programs.

The user also specifies initial placements and other initial conditions for the cells.

Users can control the simulation by writing cell programs to describe the behaviors of the cells, and by putting surfaces into the environment. More direct interaction is also possible during the simulation process. The user can halt the simulation, change the cell programs, and choose individual cells or groups of cells to modify or remove. The simulation is then restarted with the modifications. It is sometimes convenient to freeze certain cell values when they

have reached a desirable state. Frozen values remain fixed while others continue to vary when the simulation is restarted. Particular seed cells can also be placed and frozen in situations where the user wishes to achieve a certain effect. For instance, frozen cells with a particular orientation could encourage fur to run in a particular direction on a surface.

4.2 Definitions

A *cell* is an entity that has position, orientation, shape, and an arbitrary length state vector for parameters such as chemical concentrations in a reaction-diffusion simulation. It is a generalization of a particle in a particle system.

Cell State Variables The state of a cell,

$$\begin{aligned}\mathbf{S} \;=\; (\;& \mathbf{p}, \mathbf{q}, r, S_{die}, S_{split} \\ & S_{c0}, S_{c1}, S_{c2}, \cdots, S_{a0}, S_{a1}, S_{a2}, \cdots)\end{aligned}$$

is a vector containing values representing position ($\mathbf{p}$), orientation ($\mathbf{q}$), size (r), and concentrations of chemicals within the cell (S_{ci}) or in the cell membrane (S_{ai}). The variable S_{split} is used to trigger an event, the cell splitting. This occurs when the value of this state variable exceeds a threshold, θ_{split}. S_{die} is defined similarly, with an associated threshold, θ_{die}.

In real cells, chemicals in the cell membranes of adjacent cells can bind together and enable cells to sense that they are in contact. The chemicals can also be adhesive. The binding of membrane chemicals is specific; some chemicals bind in complementary pairs, and others bind to themselves. Our model allows the user to specify the adhesive properties of the membrane chemicals, and provides the amount of each bound membrane chemical as an environmental parameter (described below).

To define a cell's motion, we specify cell programs that contribute to $\mathbf{p}'$, the viscous force on the cell. This is the *attempted* motion of the cell, which is further modified by the influence of collisions, adhesion, and viscous drag. We do not currently compute inertial dynamics, but instead use viscous dynamics ($F = mv$), which makes the cells easier to control and predict. Collision forces are computed using a polynomial penalty function (kx_o^n where x_o is the overlap between two cells).

We represent cell orientation in 3-D using a quaternion. In the exposition that follows, we sometimes refer to the cell's local coordinate frame using the three basis vectors: $\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z$. This is the coordinate frame obtained by rotating from the lab frame using quaternion $\mathbf{q}$.

Extracellular Environment The cell's external environment is a vector of parameters that are provided as an input to the cell programs. These parameters describe everything the cell can sense from its current position:

$$\begin{aligned}\mathbf{\Lambda} \;=\; (\;& \Lambda_{a0}, \Lambda_{a1}, \Lambda_{a2}, \cdots, \Lambda_{p0}, \nabla\Lambda_{p0}, \Lambda_{p1}, \nabla\Lambda_{p1}, \cdots, \\ & \mathbf{\Lambda}_{v0}, \mathbf{\Lambda}_{v1}, \cdots, \\ & \mathbf{\Lambda}_{\omega x}, \mathbf{\Lambda}_{\omega y}, \mathbf{\Lambda}_{\omega z}, \Lambda_{u0}, \Lambda_{u1}, \cdots)\end{aligned}$$

The Λ_{ai} values represent the amounts of membrane chemicals that are bound to membrane chemicals on neighboring cells. The value and gradient of a potential field, such as an implicit function or the concentration of a diffusing chemical, are provided in Λ_{pi} and $\nabla\Lambda_{pi}$. These fields are evaluated at the current location of the cell, and will generally have different values at different locations. Other scalar and vector fields can be provided in Λ_{ui} and $\mathbf{\Lambda}_{vi}$, which can also be functions of position.

The orientation of a cell relative to its neighbors is made available to the cell programs in the vectors $\mathbf{\Lambda}_{\omega i}$. This vector describes the rotation that would align this cell's $\mathbf{e}_i$ axis with the average orientation of the adjacent cells. This parameter is used to align the orientations of cells, as shown in Figure 8(a) and others. The direction $\mathbf{\Lambda}_{\omega i}$ specifies the axis of rotation, and the magnitude specifies the rotation angle (similar to angular velocity). As an example, consider the computation of the average relative *x*-axis for a cell *b*, computed as a sum over neighboring cells *c*:

$$\mathbf{\Lambda}_{\omega x} = \frac{1}{n} \sum_{c \in \text{ neighbors}} \frac{\mathbf{e}_x^b \times \mathbf{e}_x^c}{\|\mathbf{e}_x^b \times \mathbf{e}_x^c\|} \cos^{-1}(\mathbf{e}_x^b \cdot \mathbf{e}_x^c)$$

where $\mathbf{e}_x^c$ is the *x*-axis of the cell *c*, and cell *b* has *n* neighbors.

Cell Programs Each cell has several cell programs, which are first order differential equations describing how its state changes over time. Examples are given in Table 1 and Section 4.3. A cell program is a function of the cell's current state $\mathbf{S}$ and its environment as expressed by $\mathbf{\Lambda}$. Different types of cells use different cell programs or different combinations of the same cell programs to define their behaviors. Even if two cells share the same set of cell programs, they will generally behave differently because they experience different local conditions depending on their position.

The entire system of differential equations to be solved is obtained by superposing ordinary differential equations from the cell programs for every cell. Additional equations arise from computation in the environment (e.g., diffusion of chemicals, although this is not in the current 3-D implementation). In order to handle discontinuous changes, such as when cells are created or die, we use a piecewise ordinary differential equation solver [2, App. C].

Mathematical Basis for Cell Programs Differential equations are a general tool for creating dynamic behavior. In our cell programs, we employ equations arising from physical models, as well as those arising from constraint solution techniques.

Higher order linear differential equations, such as those for mechanical or chemical systems, can be rewritten as multiple first order differential equations (i.e., cell programs) with the addition of state variables. In this case, the simulation dynamics reflect the dynamics of the equations.

In order to write constraints as cell programs, we formulate them as energy functions[1] to be minimized [39]. Each constraint is expressed as an energy function $E_i(\mathbf{S}, \mathbf{\Lambda})$ of the state of the system, $\mathbf{S}$, and the parameters describing the environment, $\mathbf{\Lambda}$. Hard constraints could also fit into this framework using Lagrange multipliers [22].

Using relative constants k_i to weight the soft constraints, we express the overall energy to minimize as:

$$\hat{E}(\mathbf{S}, \mathbf{\Lambda}) = k_1 E_1(\mathbf{S}, \mathbf{\Lambda}) + \cdots + k_n E_N(\mathbf{S}, \mathbf{\Lambda})$$

We can minimize this energy according to gradient descent by modifying the state according to

$$\frac{dS_\ell}{dt} = -\sum_{i=1}^{N} k_i \frac{\partial E_i}{\partial S_\ell}$$

Superposition of Cell Programs Because the overall energy is expressed as a sum, the cell programs dS_ℓ/dt are also sums of terms, one for each constraint. We find it convenient to write cell programs as incremental collections of constraints. We write this as $dS_\ell/dt \mathrel{+}= \{\text{constraint term}\}$ in our example programs in Table 1. Multiple cell programs can thus be added together conveniently.

[1] Note that when we use constraint-based cell programs, the dynamics of our simulation depends upon the gradient descent algorithm, and is not necessarily physically meaningful.

Behavior	Environment Requirements	Cell Program
Go to a surface.	An implicit surface $f(\mathbf{x}) = 0$.	$\mathbf{p}' \mathrel{+}= -k\, f(\mathbf{p}) \nabla f(\mathbf{p})$
Die if too far from surface.	An implicit surface $f(\mathbf{x}) = 0$.	$S'_{die} \mathrel{+}= \frac{1}{d}\, f(\mathbf{p})\, \theta_{die} - S_{die}$
Align an axis with a vector field.	A vector field, $\mathbf{v}(\mathbf{x})$. $\mathbf{e}_y$ is the cell's *y*-axis.	$\omega_v \equiv k \frac{\mathbf{e}_y \times \mathbf{v}(\mathbf{p})}{\|\mathbf{e}_y \times \mathbf{v}(\mathbf{p})\|} \cos^{-1}(\mathbf{e}_y \cdot \frac{\mathbf{v}(\mathbf{p})}{\|\mathbf{v}(\mathbf{p})\|})$ $\mathbf{q}' \mathrel{+}= (1/2)\, \omega_v\, \mathbf{q}$
Align *x*-axis with neighbors.	$\mathbf{\Lambda}_{\omega x}$, *x*-axis orientation relative to neighbors.	$\mathbf{q}' \mathrel{+}= (1/2)\, \mathbf{\Lambda}_{\omega x}\, \mathbf{q}$
Align *z*-axis with neighbors.	$\mathbf{\Lambda}_{\omega z}$, *z*-axis orientation relative to neighbors.	$\mathbf{q}' \mathrel{+}= (1/2)\, \mathbf{\Lambda}_{\omega z}\, \mathbf{q}$
Maintain unit quaternion.		$\mathbf{q}' \mathrel{+}= 4k(1 - \mathbf{q} \cdot \mathbf{q})\mathbf{q}$
Adhere to other cells	Membrane chemical *a2* which binds to itself.	$S'_{a2} \mathrel{+}= 1.0 - S_{a2}$
Divide until surface is covered.	Λ_{a2}, amount of *a2* which is bound.	$S'_{split} \mathrel{+}= \phi(\gamma, \Lambda_{a2})\, \theta_{split} - S_{split}$
Set size relative to surface feature size	Λ_{u0}, a value which reflects the size of the nearest feature on the surface.	$r' \mathrel{+}= \Lambda_{u0} - r$
Example of reaction-diffusion in discrete cells.	Λ_{a0}, Λ_{a2} amounts of bound membrane chemicals. The user has specified that the membrane chemical *a0* binds to *a1*, and that *a2* binds to itself.	$S'_{c0} \mathrel{+}= -20\, \Lambda_{a0} / \Lambda_{a2} +$ $10\, S_{c0}^2 / (1 + S_{c0}^2) - 0.5\, S_{c0} + 13$ $S'_{a1} \mathrel{+}= 0.95\, \Lambda_{a0} / \Lambda_{a2}$ $S'_{a1} \mathrel{+}= \phi(3, S_{c0})\, S_{c0}$ $S'_{a1} \mathrel{+}= -S_{a1}$ $S'_{a0} \mathrel{+}= 5 - S_{a0}$ $S'_{a2} \mathrel{+}= 1 - S_{a2}$

Table 1: Example Cell Programs. Scalar or vector fields are given as a function of spatial location, $\mathbf{x}$, and are evaluated at the current location of the cell, $\mathbf{p}$. See cell program descriptions in Section 4.3.

4.3 Example Cell Programs

The cell programs listed in Table 1 exemplify the types of cell programs used to make the figures in this paper. We describe each briefly below.

Remember that the contributions from multiple terms are added together to make a single differential equation for each state variable (using the += notation). Many of the cell programs shown here are of the form $S'_i(t) = dS_i/dt = \beta - S_i(t)$ for some constant β, which causes S_i to quickly approach the value β.

Go to a surface. This cell program implements a constraint to keep a cell on the implicit surface $f(\mathbf{x}) = 0$. An approximation to the gradient, $\nabla f(\mathbf{x})$, is also available in the environment. As the simulation runs, a cell with this program will descend the gradient and come to rest on the surface. The parameter k determines the speed with which the particle approaches the surface.

Die if too far from surface. Recall that when the variable S_{die} crosses the threshold θ_{die}, it triggers cell death (Section 4.2). In this cell program, we cause S_{die} to rise towards the threshold quickly if the cell is greater than a certain distance from the surface. Computing this requires a measure of the distance from the surface. For the implicit surfaces used in the figures, $f(\mathbf{x})$ is an approximation to the distance from the surface.

The equation in Table 1 causes cells at a distance greater than d to die. Similar cell programs can cause a cell to die if it becomes too large, its orientation strays too far from neighboring cells, or due to any other condition that is a function of the cell's environment and internal state.

Align with a vector field. In this example, we align the cell's *y*-axis, $\mathbf{e}_y$ with a given vector field $\mathbf{v}(\mathbf{x})$. The vector field is evaluated at the cell's current location, $\mathbf{p}$.

We first compute the vector ω_v, which represents the transformation required to rotate the $\mathbf{e}_y$ into $\mathbf{v}$. ω_v is the axis of rotation, and the length of ω_v specifies the angle through which to rotate. The formulation works with any of the cell's axes.

The form of the cell program comes from the equation

$$\mathbf{q}' = (1/2)\, \omega\, \mathbf{q},$$

which defines the rate of change of a quaternion, $\mathbf{q}$, for angular velocity ω [12].

If we have multiple cell programs of this form, the ω_i terms add, which allows us to constrain one, two, or all three axes of the cells. If the orientation constraints conflict, the cell's orientation will approach the average orientation. If they don't conflict, all constraints will become satisfied.

Align with neighbors. The three orientation constraints on the cell's *x*-, *y*- and *z*-axes fully constrain its orientation. Constraining two axes would be sufficient in most cases, except where the vector field $\mathbf{v}(\mathbf{x})$ happened to be collinear with the *x*- or *z*-axis. Having the extra constraint keeps us from running into problems in that case, and also aids the convergence of the cell alignment process.

Maintain unit quaternion. It is wise to add a constraint to ensure that the quaternion does not stray too far from a unit quaternion during the integration of the differential equations. We can do this with another simple constraint. Table 1 shows a term for this constraint that comes from minimizing the energy expression $E = (1 - \mathbf{q} \cdot \mathbf{q})^2$, which describes the deviation of $\mathbf{q}$ from a unit quaternion.

Adhere to other cells. This equation causes the variable S_{a2} (representing the surface chemical *a2*) to approach and stay at the value 1.0. A pair of cells expressing *a2* will stick together once they come in contact, and a force is required to pull them apart. The environment vector for each cell will report the amount of chemical bound on each cell, Λ_{a2}, which may be used in other cell programs to determine if the cell has contacted another cell. The amount bound is computed from the contact area between the two cells, and the concentrations on each cell.

Divide until surface is covered. Divide until the amount of bound surface chemical *a2* reaches the level γ. Λ_{a2} reports the total amount bound from all cells that are in contact, which gives the cell a means of determining how many neighbors it has. Note that the mechanism has more general utility than just counting neighbors. For instance, a cell with twice the concentration of *a0* will contribute more to Λ_{a2}.

The auxiliary function $\phi(\gamma, \Lambda_{a0})$ is used in this cell program to compute a continuous version of the condition $(\Lambda_{a0} > \gamma)$. The function $\phi(a, b)$ computes a continuous version of the boolean condition $(b > a)$:

$$\phi(a, b) \equiv (tanh((b - a)) + 1)/2.$$

The value of this function will be near one for $(b >> a)$ and near zero for $(a << b)$.

Set size relative to surface feature size. In Figure 1, the cell sizes are related to the sizes of features in the polygonal database. This is achieved by providing the cells with a value Λ_{u0} that represents the area of the nearest triangle. The value Λ_{u0} could be used to pass information about local curvature or any other parameter that we wish to use to change the cell behavior.

Example of reaction-diffusion in discrete cells. The full derivation [7] of this set of equations is beyond the scope of this paper, however we will describe the equations briefly. The first equation in this set defines a genetic switch [18] that tends to drive S_{c0} towards one of two values, depending on the influence of the term $\Lambda_{a0}/\Lambda_{a2}$. In terms of Meinhardt's activator-inhibitor models [16], S_{c0} is the activator and S_{a1} is the inhibitor, which is propagated by the activity of membrane chemicals. The other equations determine interactions of membrane chemicals that lead to an effective diffusion of the value of S_{a1} among the cells. The value of S_{c0} can then be used to determine the final rendered shape of the cell, as illustrated in to Figures 2 and 7.

4.4 Surface Constraints

We have applied surface constraints to a variety of surface classes:

- polygonal mesh,
- implicit function, and
- isosurfaces of volume data.

The surface constraint cell program evaluates an implicit function to enable the cell to find and stay on the surface (Table 1).

Several of the surfaces used to create the figures are defined by triangular meshes. We create a rough approximation to an implicit function for these meshes. Any implicitization method will work, and in fact it doesn't have to be very exact. We implement this function by constructing an approximate *kd*-tree for the triangular mesh. In constructing a true *kd*-tree, each additional vertex may add several new partitions. Our approximation adds only the one necessary partition for each added vertex. This makes the tree smaller and faster to precompute, but no longer actually gives the closest point. To evaluate the function, we look up the triangle center in the *kd*-tree supposedly nearest to a given point. We then check adjacent triangles to see if they are closer, to improve the characteristics of the approximation. We then compute the direction and distance to that triangle, and use it as we would the gradient of an implicit function. We find the approximation, in conjunction with the local search, to be satisfactory for this application.

5 Particle Converter

The particle converter converts information about the particles and their environment into geometry and appearance parameters for rendering. It receives all of the results of the simulation, including the position, orientation and size of each cell, concentration of reaction-diffusion chemicals, and other arbitrary user-defined parameters, such as type or color. It also may have access to information about how far each cell is from the surface and properties of the surface near the cell (e.g, curvature) The converter also knows which cells contact each other.

The particle converter concept has proven to be extremely convenient. It enables us to do a variety of useful operations, including:

- choosing an appropriate representation for each cell based on its screen size (Figure 4);
- smoothly changing the appearance of a cell based on a continuously varying parameter (Figure 7);
- using the cell positions to generate a spatial subdivision (similar to [33, 37, 41]);
- using the cell orientations to compute a flow field on the surface (useful for displacement maps [21]); and
- experimenting with various colorations and geometries using the same simulation dataset.

The output of the particle converter is a collection of geometry and appearance information suitable for a particular renderer. This collection will generally include one or more geometric primitives for each cell, and the local texture, transparency, or bump information. The geometry can be simple, as in Figure 5, where each cell is rendered as a few polygons with a mottled-green texture map. Or it can be more complicated, as in the parameterized 3D thorn shape that curls based on cell state information (Figure 7 (c)).

The particle converter can also use contact information to calculate the size or shape of geometric primitives based on neighboring cell proximity, or to interpolate parameters such as orientation between cell centers.

We have implemented two particle converters. One provides options for choosing a particular geometry and texture for each cell. In addition, it considers information associated with the underlying polygons, such as which body part it represents in an anatomical model (lips, eyes, etc.) This can be used to change the rendering of cells in certain areas, as can be seen on the lips of the man in Figure 1.

Our second particle converter was implemented using a general purpose modeler [31]. Taking advantage of the flexibility of this modeler, we can create parameterized objects such as the bump-to-thorn shape shown in Figure 2. The modeler is used to create the

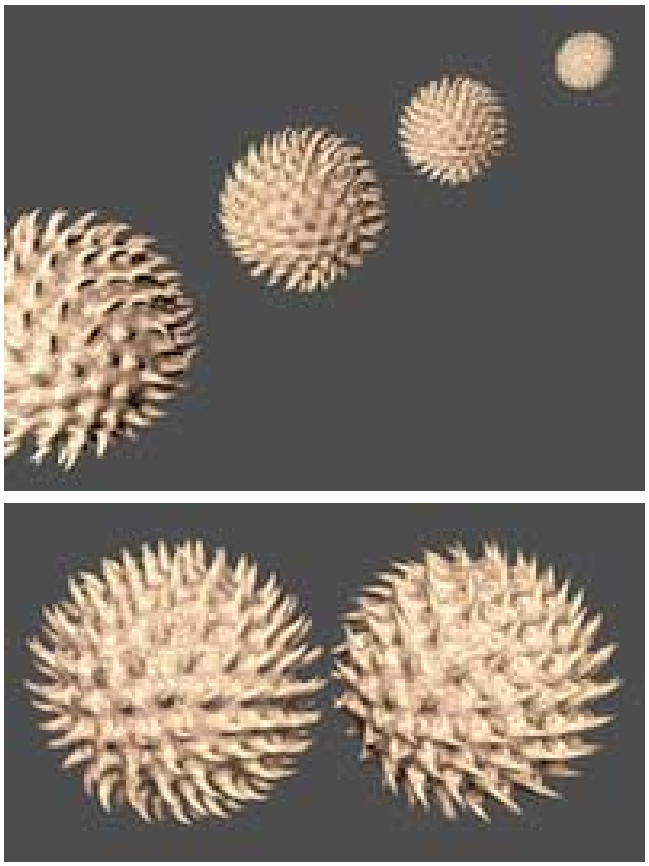

Figure 4: In the top image, the thorny spheres at further distances are rendered with fewer polygons. The bottom image shows a closeup of the nearest and furthest objects, so we can see the reduced number of polygons.

thorny spheres in Figure 7.

Rendering an appropriately scaled representation One of the drawbacks of data amplification techniques [30, 26] such as ours is their ability to generate a ridiculous amount of geometry to render. To ameliorate this, we use the particle converter to choose geometric primitives appropriate to the size of the object in the final image (Figure 4). This approach could be carried even further, for instance, by creating texture maps based on the cell positions.

6 Results

In this section we list a series of examples that highlight features of our system.

Scales Figure 5 shows four views of a spherical object with a uniform covering of similarly-oriented cells. The cell programs used here incorporate terms to divide until the surface is covered, to stay on the surface, and to die if pushed too far off of the surface. Initially, several cells were placed near the surface, and allowed to divide and wander. The cells were also given a soft constraint to align their y-axes with the gradient of the surface implicit function, and to align their x- and z-axes with their neighbors.

Note that two singularities in the orientations of cells arise naturally on the sphere, due to its topology. One is visible on the near

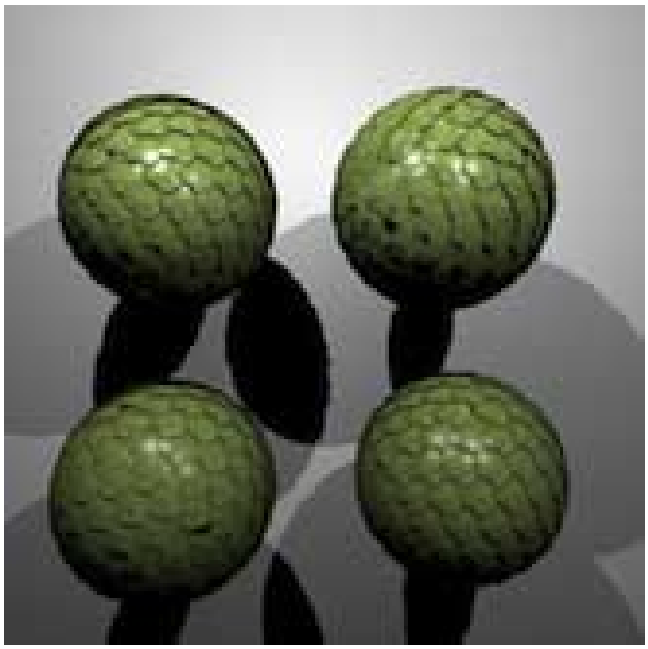

Figure 5: Scales: Four views of a spherical object uniformly covered with similarly oriented cells. Each cell is rendered as a group of four polygons with a texture and transparency map. The polygons are tilted slightly to give a layered appearance.

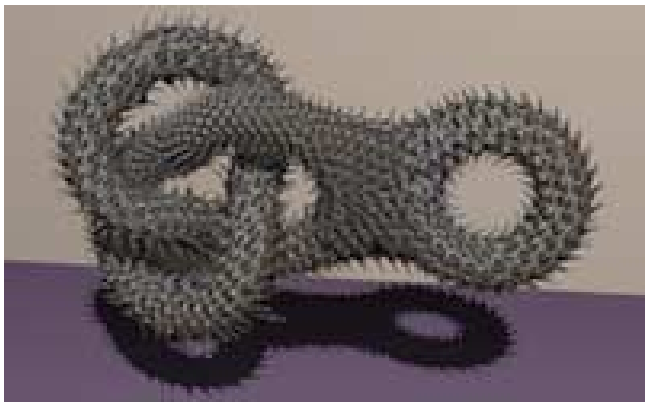

Figure 6: Cellular textures can handle unusual topologies.

side of the upper right sphere. Unlike standard texture mapping, this method introduces no particular parameterization problems, such as stretching or shrinking of the texture.

A Knotty Problem Figure 6 shows that the cellular texture approach is capable of creating textures for surfaces with unusual topologies. It is not necessary to have a parameterization for the surface. This surface was designed by John Hughes and John Snyder.

Thorny Head: Changing cell size to match surface features The examples described so far have used cells that are relatively

uniform in size. Figure 1 shows an example where the cell size is related to the detail level in the underlying polygonal model. We achieve this by providing the cells with another environmental variable: the area of the nearest triangle in the underlying polygon mesh. Note the finer texture and geometry around the eyes and mouth.

Different rendering parameters were chosen according to properties of the underlying polygonal model. Each polygon in the underlying database is associated with a region of the body. The particle converter assigned different shading properties to the cells in the head and neck regions. At the eyes, the underlying polygonal representation shows through the cell texture.

Thorny Spheres: Differentiated cellular textures This example shows several important capabilities of the system. It shows

- the creation of simple reaction-diffusion patterns on a surface,
- the use of the concentrations of cell chemicals to change parameters of the rendered geometry, and
- the ability to restart simulations from an previous state with new cell programs, causing new behaviors to occur.

These cells are using reaction-diffusion equations similar to those in Table 1 to create patterns of chemicals in the cells. The diffusion of chemicals occurs by contact between cell membranes, thus it can only occur between adjacent cells.

Using a geometric modeler, we created a parameterized geometric object that changes from a bump to a thorn based on a single parameter [31]. The particle converter sets this parameter to the value of a state variable representing the concentration of one of the reaction-diffusion chemicals.

We can see that there is a patch of cells on the front of the sphere with very little of the chemical (rendered as bumps), and a larger patch on the back with more of the chemical (rendered as thorns). In addition to the sharp boundary between the patches, note that the height of the thorns on the back patch varies continuously as they sweep around the sphere.

These simulations began where the earlier sphere simulation of Figure 5 left off, with new rules to cause the cells to differentiate. This is a common motif of user interaction with the system: halt a simulation, modify cell programs and parameters, and then continue simulating.

A Bear of a Surface In Figure 8, we show a fur-covered model of a bear defined as an isosurface of sampled volume data. We would like for the bear's fur to have a natural-looking orientation [15]. The bear on the left, with the fully combed fur, started from a single cell and used a set of rules similar to those used for Figure 5 to distribute and orient the cells. Each cell on the bear is rendered with a group of geometric objects meant to roughly represent a hunk of thick hair.

The bear on the right, with the patchy fur, was the result of a serendipitous combination of unintentional cell programs. Rather than having each cell align with all of its neighbors, each cell chooses one neighbor to align with. Also, cells do not attempt to align with neighbors that are oriented in the directly opposing direction. This bear started from about 2000 arbitrarily chosen cells on the surface.

Additional, more specific, orientation constraints could cause the fur to run more naturally down the limbs. Other cell programs could be added to cause the fur to be shorter in the region of the face and longer on the haunches, or to change the coloration based on the orientation or curvature of the bear's features.

7 Discussion

The combination of particle constraint techniques with developmental models enables the generation of a variety of cellular textures, as shown in the figures. We have found the approach to be a powerful method of creating attractive computer graphics models of organic objects. In our experience with making cellular textures, we encountered some difficulties, which we describe below. Some of these limitations are associated with our current implementation, and can be remedied without changing the basic framework. The problems with simulation speed and data explosion are less easily finessed, and will require further research to address fully.

Some commercially produced computer graphics films and videos contain models that have textures that appear similar to ours. The techniques used to generate them are generally proprietary and unpublished, hence we cannot definitively compare them with our work. Software for orienting fur on a CG character has been developed at Industrial Light & Magic [5]. It is interesting to note that their discussion of the difficulties encountered closely parallels our own experience.

Shapes The spherical shapes of cells in a simulation generally are not the shapes we want to render, and so the particle converter might make objects with undesirable intersections. This can be minimized by a careful choice of cell geometry, but a more robust solution is to use the desired geometric shape directly in the cellular particle simulation. This would allow cell programs to calculate collisions based on more accurate geometry.

Experience with Writing Cell Programs Writing cell programs can be difficult. Programming independently moving cells by specifying differential equations has many desirable properties, but requires a different intuition than other types of programming, and often takes a while to get right. As with many tasks, it gets easier with practice. Here are some suggestions for using this programming paradigm:

- Copy and combine known cell programs from other researchers, such as surface or orientation constraints [41, 33].
- Think about the constraints in the energy formulation (Section 4, and [39]).
- Satisfy one constraint at a time; e.g., first get cell positions right, then modify other attributes.
- Force certain problem cells to be a certain way (through direct interaction, Section 4.1).
- Kill problem cells and regrow (Section 4.1).
- Apply artificial evolution [29], and be patient.

Simulation Speed Simulations can be slow for some kinds of cell programs. We have some that run in seconds, and others, like the large datasets, that take many hours (e.g., the bear in Figure 8, and the head in Figure 1). Generally, performance degrades as the differential equations get stiff [11]. For some behaviors, clever cell programs like those described in [41] avoid creating stiff differential equations, and so run faster.

Data Explosion The data produced both by the simulation and by the particle converter can get very large. We have partially addressed this by parameterizing the particle converter output by viewing distance (Figure 4). However, the simulation still has to compute enough cells to cover the surfaces, independent of viewing distance.

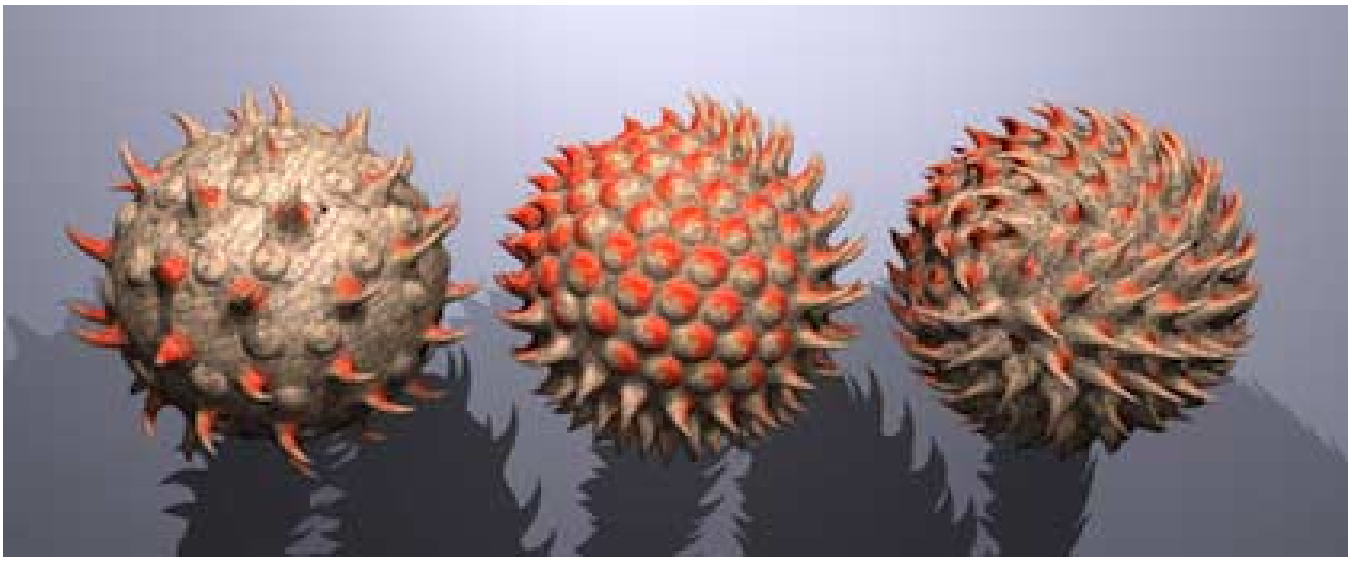

Figure 7: Varying Thorns. Reaction-diffusion-like equations determine the pattern of bumps and thorns on these spheres. Note the continuously varying thorn height and thorn curvature on the center and rightmost spheres.

Figure 8: The bear on the left is fully combed, with all cells oriented like their neighbors. The bear on the right has patches of similarly-oriented cells.

Future Work

There are several directions in which we would like to extend this work. First, we plan to continue extending and refining the cell programs to generate more complex cellular textures. We also are interested in running simulations on objects as they move and change shape. Modeling the motion of feathers on the wings of a flying bird, or hair on a running animal would be exciting. Initial experiments (not discussed in this paper) indicate that this will be feasible.

Implementing more sophisticated cell geometries in the particle simulator will give us more realistic placement of detail, and avoid self-intersections in the rendering. Finally, we would like to explore the possibilities of creating shapes directly from the fundamental interactions of the cells, without the surface constraint.

Acknowledgments

Many thanks to Erik Winfree for designing and implementing the *kd*-tree approximation, as well as for providing many helpful suggestions. We are grateful to Allen Corcorran, Matt Avalos, Cindy Ball, Dan Fain, Louise Foucher, Marcel Gavriliu, Barbara Meier, Mark Montague, Alf Mikula, Preston Pfarner, Ravi Ramamoorthi, Dian De Sha, and Denis Zorin for valuable discussions, support, code, and proofreading. MRI data was taken at the Huntington MRI center in Pasadena, CA.

This work was supported in part by grants from Apple, DEC,

Hewlett Packard, and IBM. Additional support was provided by NSF (ASC-89-20219) as part of the NSF/ARPA STC for Computer Graphics and Scientific Visualization, by the DOE (DE-FG03-92ER25134) as part of the Center for Research in Computational Biology, the Beckman Foundation, and by the National Institute on Drug Abuse and the National Institute of Mental Health as part of the Human Brain Project. All opinions, findings, conclusions, or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

REFERENCES

[1] James Arvo and David Kirk. Modeling plants with environment-sensitive automata. In *Proceedings of Ausgraph '88*, pages 27–33, 1988.

[2] Ronen Barzel. *Physically-Based Modeling: A Structured Approach*. Academic Press, Cambridge, MA, 1992.

[3] Robert L. Cook. Shade trees. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 223–231, July 1984.

[4] F. C. Crow. A more flexible image generation environment. In *Computer Graphics (SIGGRAPH '82 Proceedings)*, volume 16, pages 9–18, July 1982.

[5] Jody Duncan. The making of a rockbuster. *Cinefex: the Journal of Cinematic Illusions*, 58:34–65, June 1994.

[6] Kurt Fleischer. Cells: Simulations of multicellular development. In *Siggraph Video Review*, 1994. *A video presented at Siggraph 94*.

[7] Kurt W. Fleischer. *A Multiple-Mechanism Developmental Model for Defining Self-Organizing Structures*. PhD dissertation, Caltech, Department of Computation and Neural Systems, June 1995.

[8] Kurt W. Fleischer and Alan H. Barr. A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In *Artificial Life III*. Addison-Wesley, 1994.

[9] Deborah R. Fowler, Hans Meinhardt, and Przemyslaw Prusinkiewicz. Modeling seashells. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 379–388, July 1992.

[10] Deborah R. Fowler, Przemyslaw Prusinkiewicz, and Johannes Battjes. A collision-based model of spiral phyllotaxis. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 361–368, July 1992.

[11] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

[12] Goldstein. *Classical Mechanics*. Addison-Wesley, 1980.

[13] Ned Greene. Voxel space automata: Modeling with stochastic growth processes in voxel space. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 175–184, July 1989.

[14] James T. Kajiya. Anisotropic reflection models. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 15–21, July 1985.

[15] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 271–280, July 1989.

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

[17] Gavin Miller and Andrew Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309, 1989.

[18] J. D. Murray. *Mathematical Biology*. Springer-Verlag, New York, 2nd edition, 1993.

[19] B. N. Nagorcka, V. S. Manoranjan, and J. D. Murray. Complex spatial patterns from tissue interactions – an illustrative model. *Journal of Theoretical Biology*, 128:359–374, 1987.

[20] Garrett M. Odell, George Oster, P. Alberch, and B. Burnside. The mechanical basis of morphogenesis. *Developmental Biology*, 85, 1981.

[21] Hans Køhling Pedersen. Displacement mapping using flow fields. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 279–286. ACM Press, July 1994.

[22] John Platt. *Constraint Methods for Neural Networks and Computer Graphics*. PhD dissertation, Caltech, Department of Computer Science, Pasadena, CA, 91125, 1989.

[23] Przemyslaw Prusinkiewicz, Mark James, and Radomiŕ Měch. Synthetic topiary. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 351–358. ACM Press, July 1994.

[24] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.

[25] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[26] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 313–322, July 1985.

[27] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 25–34, July 1987.

[28] Karl Sims. Particle animation and rendering using data parallel computation. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 405–413, August 1990.

[29] Karl Sims. Artificial evolution for computer graphics. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 319–328, July 1991.

[30] Alvy Ray Smith. Plants, fractals and formal languages. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 1–10, July 1984.

[31] John Snyder. *Generative Modeling for Computer Graphics and CAD: Symbolic Shape Design using Interval Analysis*. Academic Press, 1992.

[32] John M. Snyder and Alan H. Barr. Ray tracing complex models containing surface tessellations. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 119–128, July 1987.

[33] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 185–194, July 1992.

[34] Demetri Terzopoulos, John Platt, and Kurt Fleischer. From goop to glop: Heating and melting deformable models. In *Graphics Interface 89*, 1989.

[35] Alan Turing. The chemical basis of morphogenesis. *Phil. Trans. B.*, 237, 1952.

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

[37] Greg Turk. Re-tiling polygonal surfaces. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[38] Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. Predicting reflectance functions from complex surfaces. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 255–264, July 1992.

[39] Andrew Witkin, Kurt Fleischer, and Alan Barr. Energy constraints on parameterized models. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 225–232, July 1987.

[40] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 299–308, July 1991.

[41] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 269–278. ACM Press, July 1994.