CHAPTER

Special Models for Animation 12

Often, how an object is defined is intimately woven with how it is animated. We have already seen examples of this, such as hierarchical linkages, special models for fluids, and so forth. This chapter contains a collection of such special models that often have a role to play in animation. These models are implicit surfaces, L-systems, and subdivision surfaces. Some of these modeling techniques have already been mentioned in previous chapters (e.g., implicit surfaces in Chapter 8 and subdivision surfaces in the Chapter 10). This chapter is intended to give the interested reader some additional information on these modeling techniques.

12.1 Implicit surfaces

Implicitly defined surfaces are surfaces defined by all those points that satisfy an equation of the form, $f(P) = 0$; $f(P)$ is called the *implicit function*. A common approach to using implicit surfaces to define objects useful for animation is to construct a compound implicit function as a summation of implicitly defined primitive functions. Interesting animations can be produced by animating the relative position and orientation of the primitive functions or by animating parameters used to define the functions themselves. Implicit surfaces lend themselves to shapes that are smooth and organic looking. Animated implicit surfaces are useful for shapes that change their topology over time. They are often used for modeling liquids, clouds, and animals.

An extensive presentation of implicit surface formulations is not appropriate material for this book, but it can be found in several sources, in particular in Bloomenthal's edited volume [3]. A brief overview of commonly used implicit surfaces and their use in animation is presented here.

12.1.1 Basic implicit surface formulation

Typically, an *implicit surface* is defined by the collection of points that are the zero points of some implicit function, $f(P) = 0$. The surface is referred to as implicit because it is only implicitly defined, not explicitly represented. As a result, when an explicit definition of the surface is needed, as in a graphics display procedure, the surface has to be searched for by inspecting points in space in some organized way.

Implicit surfaces can be directly ray traced. Rays are constructed according to the camera parameters and display resolution, as is typical ray tracers. Points along the ray are then sampled to locate a surface point within some error tolerance.

An explicit polygonal representation of an implicitly defined surface can be constructed by sampling the implicit function at the vertices of a three-dimensional mesh that is constructed so that its

extent contains the non-zero extent of the implicit function. The implicit function values at the mesh vertices are then interpolated along mesh edges to estimate the location of points that lie on the implicit surface. Polygonal fragments are then constructed in each cell of the mesh by using any surface points located on the edges of that mesh cell [10].

The best known implicit primitive is often referred to as the *metaball* and is defined by a central point (C), a radius of influence (R), a density function (f), and a threshold value (T). All points in space that are within a distance R from C are said to have a density of $f(distance(P,C)/R)$ with respect to the metaball (where $distance(P,C)$ computes the distance between P and C and where $distance(P,C)/R$ is the *normalized distance*). The set of points for which $f(distance(P,C)/R) - T = 0$ (implicitly) defines the surface, S.

In Figure 12.1, r is the distance at which the surface is defined for the isolated metaball shown because that is the distance at which the function, f, evaluates to the threshold value. Desirable attributes of the function, f, are $f(0.0) = 1.0$, $f(0.5) = 0.5$, $f(1.0) = 0.0$, and $f'(0.0) = f'(1.0) = 0.0$. A common definition for f, as proposed by Wyvill [20], is Equation 12.1.

$$f(s) = 1 - \left(\frac{4}{9}\right)s^6 - \frac{17}{9}s^4 - \frac{22}{9}s^2 \tag{12.1}$$

Two generalizations of this formulation are useful. The first uses the weighted sum of a number of implicit surface primitives so that the surface-defining implicit function becomes a smooth blend of the individual surfaces (Eq. 12.2).

$$F(P) = \Sigma w_i f_i(P) - T \tag{12.2}$$

The weights are arbitrarily specified by the user to construct some desired surface. If all of the weights, w_i, are one, then the implicit primitives are summed. Because the standard density function has zero slope at one, the primitives will blend together smoothly. Using negative weights will create smooth concavities. Several concavities can be created by making the weight more negative. Integer weights are usually sufficient for most applications, but noninteger weights can also be used (see Figure 12.2).

The second generalization provides more primitives with which the user can build interesting objects. Most primitives are distance based, and most of those are offset surfaces. Typical primitives use the same formulation as the metaball but allow a wider range of central elements. Besides a single

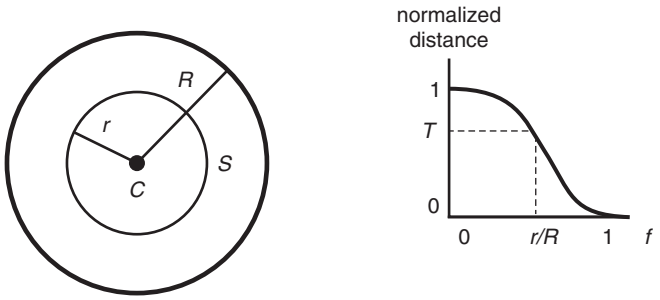

FIGURE 12.1

The metaball and a sample density function.

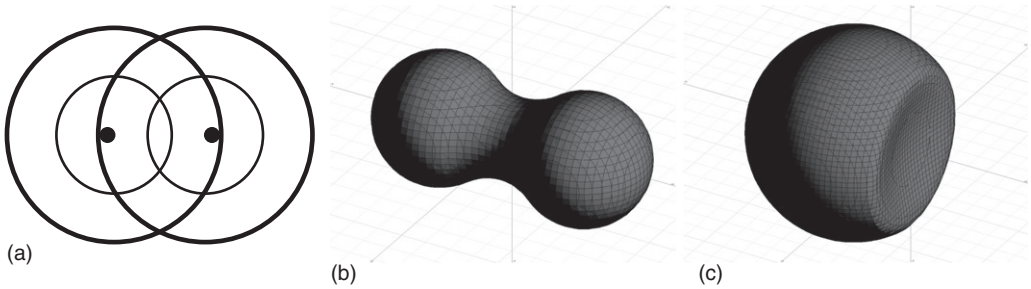

FIGURE 12.2

Overlapping metaball density functions. (a) Schematic showing overlapping density functions. (b) Surface constructed when positive weight are associated with density functions. (c) Surface constructed when one positive weight and one negative weight are associated with density functions.

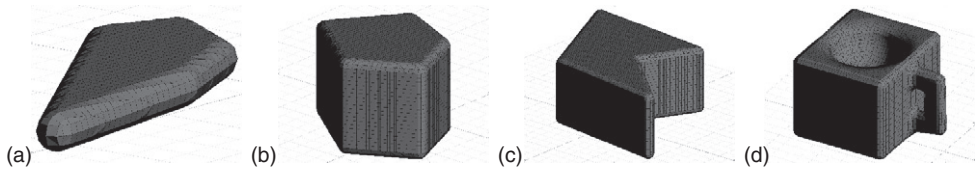

FIGURE 12.3

Various surfaces extracted from implicit distance-based functions. (a) Distance-based implicit primitive based on a single polygon. (b) Distance-based implicit primitive based on a single convex polyhedron. (c) Distance-based implicit primitive based on a single concave polyhedron. (d) Compound implicitly defined object.

point, a primitive can be defined that uses a line segment, a triangle, a convex polyhedron, or even a concave polyhedron. Any central element for which there is an associated well-defined distance function can be used. The drawback to using more complex central elements, such as a concave polyhedron, is that the distance function is more computationally expensive. Other primitives, which are not strictly offset surfaces but which are still distance based, are the cone-sphere and the ellipse. See Figure 12.3a–c for examples of distance-based primitives. As with metaballs, these implicit primitives can be combined to create interesting compound density functions whose isosurfaces can be extracted and displayed (Figure 12.3d).

12.1.2 Animation using implicitly defined objects

In Bloomenthal's book [3], Wyvill discussed several animation effects that can be produced by modifying the shape of implicit surfaces. The most obvious way to achieve these modifications is to control the movement of the underlying central elements. Topological changes are handled nicely by the implicit surface formulation. The points that define a collection of metaballs can be controlled by a simple particle system, and the resulting implicit surface can be used to model water, taffy, or clouds, depending on the display attributes and the number of particles. Central elements consisting of lines can be articulated as a jointed hierarchy.

Motion can also be generated by modifying the other parameters of the implicit surface. The weights of the different implicit primitives can be manipulated to effect bulging and otherwise control the size of an implicit object. A simple blend between objects can be performed by decreasing the weight of one implicit object from 1 to 0 while increasing another implicit object's weight from 0 to 1. The density function of the implicit objects can also be controlled to produce similar shape deformation effects. Modifying the central element of the implicit is also useful. The elongation of the defining axes of the ellipsoidal implicit can produce squashing and stretching effects. The orientation and length of the axes can be tied to the velocity and acceleration attributes of the implicit object.

12.1.3 Collision detection

Implicitly defined objects, because they are implemented using an implicit function, lend themselves to collision detection. Sample points on the surface of one object can be tested for penetration with an implicit object by merely evaluating the implicit function at those points in space (Figure 12.4). Of course, the effectiveness of this method is dependent on the density and distribution of the sample points on the first object. If a polyhedral object can be satisfactorily approximated by one or more implicit surface primitives, then these can be used to detect collisions of other polyhedral objects. Bounding spheres are a simple example of this, but more complex implicits can be used for more precise fitting.

Because implicit functions can be used effectively in testing for collisions, they can be used to test for collisions between polyhedral objects if they can fit reasonably well on the surface of the polyhedra (Figure 12.5). Of course, the effectiveness of the technique is dependent on the accuracy with which the implicit surfaces approximate the polyhedral surface.

12.1.4 Deforming the implicit surface as a result of collision

Cani has developed a technique to compute the deformation of colliding implicit surfaces [3] [6] [7] (published under the name of Gascuel). This technique first detects the collision of two implicit surfaces by testing sample points on the surface of one object against the implicit function of the other, as previously described. The overlap of the areas of influence of the two implicit objects is called the

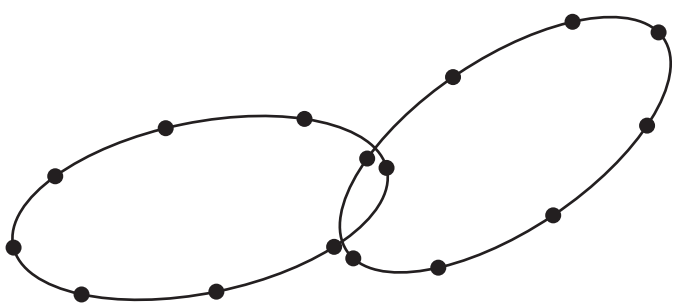

FIGURE 12.4

Point samples used to test for collisions.

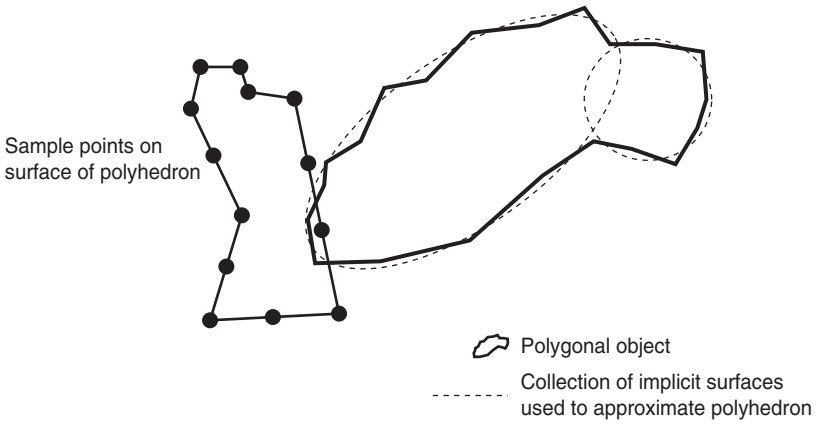

FIGURE 12.5

Using implicit surfaces for detecting collisions between polyhedral objects.

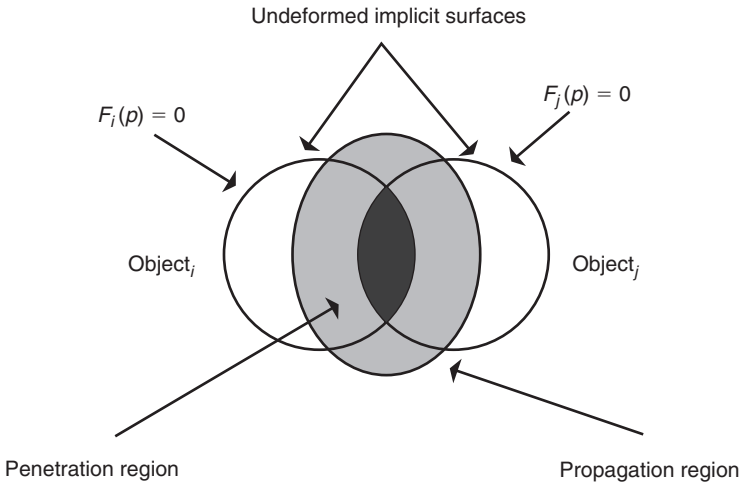

FIGURE 12.6

Penetrating implicit surfaces.

penetration region. An additional region just outside the penetration region is called the *propagation region* (Figure 12.6).

The density function of each object is modified by the overlapping density function of the other object to deform the implicitly defined surface of both objects so that they coincide in the region of overlap, thus creating a contact surface (Figure 12.7). As shown in the example in Figure 12.7, a deformation term is added to F_i as a function of Object$_j$'s overlap with Object$_i$, G_{ij}, to form the contact surface. Similarly, a deformation term is added to F_j as a function of Object$_i$'s overlap with Object$_j$, G_{ji}. The deformation functions are defined so that the isosurface of the modified density functions,

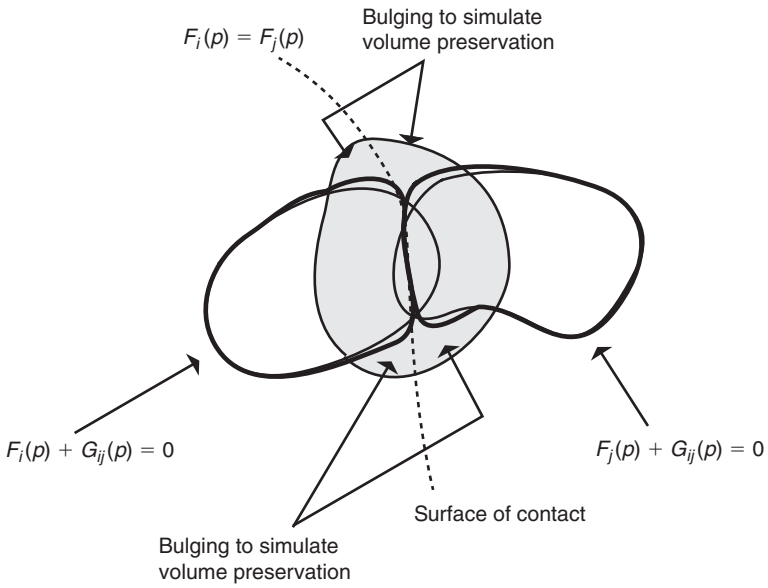

FIGURE 12.7

Implicit surfaces after deformation due to collision.

$F_i(p) + G_{ij}(p) = 0$ and $F_j(p) + G_{ji}(p) = 0$, coincide with the surface defined by $F_i(p) = F_j(p)$. Thus, the implicit functions, after they have been modified for deformation and evaluated for points p on the contact surface, are merely $F_i(p) - F_j(p) = 0$ and $F_j(p) - F_i(p) = 0$, respectively. Thus, G_{ij} evaluates to F_j at the surface of contact.

In the penetration region, the Gs are negative in order to compress the respective implicit surfaces as a result of the collision. They are defined so that their effect smoothly fades away for points at the boundary of the penetration region. Consequently, the Gs are merely the negative of the Fs for points in the penetration region (Eq. 12.3).

$$\begin{aligned} G_{ij}(p) &= -F_j(p) \\ G_{ji}(p) &= -F_i(p) \end{aligned} \tag{12.3}$$

To simulate volume preservation, an additional term is added to the Gs so that they evaluate to a positive value in the propagation region immediately adjacent to, but just outside, the penetration region. The effect of the positive evaluation will be to bulge out the implicit surface around the newly formed surface of contact (Figure 12.7).

In the propagation region, $G(p)$ is defined as a function, h, of the distance to the border of the interpenetration region. To ensure C^1 continuity between the interpenetration region and the propagation region, $h'(0)$ must be equal to the directional derivative of G along the gradient at point p (k in Figure 12.8). See Gascuel [6] and Gascuel and Gascuel [7] for more details.

Restoring forces, which arise as a result of the deformed surfaces, are computed and added to any other external forces acting on the two objects. The magnitude of the force is simply the deformation term, G; it is in the direction of the normal to the deformed surface at point p.

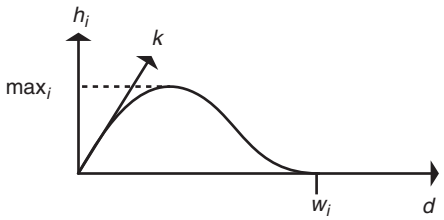

FIGURE 12.8

Deformation function in the propagation region.

12.1.5 Level set methods

"Level set methods add dynamics to implicit surfaces" [13]. These methods operate on signed distance functions using a grid representation for an object. Each grid cell contains the value of the signed distance function at that location in space. An isosurface, φ, is computed on the grid as the interface between positive and negative values. The isosurface is updated according to a velocity field defined over the interface. For example, in two-dimensional space, $d\varphi/dt=V(x,y,t)$. The function V can be given as an externally generated velocity field. The isosurface advects (i.e., moves) in the direction of the velocity function. A commonly used function uses the direction of the (positive or negative) gradient of φ and a constant magnitude. Thus, the isosurface advects in the direction of its normal at a constant speed. Alternatively, the speed may be based on the magnitude of the curvature, $d^2\varphi/dt^2$. A Euler update of $\varphi(t + Dt) = \varphi(t) + dt*V$ can be used to actually update the grid values. This is called *solving the level set equations*.

Every step taken in time corrupts the signed distance function of the grid as values are modified. Thus, the grid values need to be updated, called *renormalized*, in order to construct a signed distance field again.

The level set equations for advection of a surface by a vector field are represented by Equations 12.4 and 12.5.

$$H = V \times \nabla\phi \tag{12.4}$$

$$\nabla\phi = \left(\frac{d\phi}{dx}, \frac{d\phi}{dy}\right) \tag{12.5}$$

where V is a vector field.

Solving the level set equations moves the interface in the direction of the vector field. For each coordinate, calculate $V \bullet \nabla\varphi$ and see where it takes the interface. $\nabla\varphi$ is calculated from the grid representation.

To approximate $\nabla\varphi$, use the upwind scheme such that, in the horizontal case,

$$\text{if}, v_x < 0, \frac{d\phi}{dx} = \phi(x+1, y) - \phi(x, y)$$

$$\text{if}, v_x > 0, \frac{d\phi}{dx} = \phi(x, y) - \phi(x-1, y)$$

To solve the level set equations advecting in the normal direction, at a given point, get the point's normal and scale it to form vector field. Common functions used to control the speed are curvature at the point and a constant function. The interface can be updated by taking a Euler step in the direction of its velocity. Once updated, the distance function needs to be updated as well. For purposes of efficiency, the entire grid is not usually processed. Instead, just a band of grid values on either side of the interface is updated [13].

12.1.6 Summary

Although their display may be problematic for some graphic systems, implicit surfaces provide unique and interesting opportunities for modeling and animating unusual shapes. They produce very organic-looking shapes and, because of their indifference to changes in genus of the implicitly defined surface, lend themselves to the modeling and animating of fluids and elastic material.

12.2 Plants

The modeling and animation of plants represent an interesting and challenging area for computer animation. Plants seem to exhibit arbitrary complexity while possessing a constrained branching structure. They grow from a single source point, developing a branching structure over time while the individual structural elements elongate. Plants have been modeled using particle systems, fractals, and L-systems. There has been much work on modeling the static representations of various plants (e.g., [1] [2] [8] [12] [15] [17] [18]). The intent here is not to delve into the botanically correct modeling of particular plants but rather to explain those aspects of modeling plants that make the animation of the growth process challenging. The representational issues of synthetic plants are discussed in just enough detail to uncover these aspects. Prusinkiewicz and Lindenmayer [14] [16] provide more information on all aspects of modeling and animating plants.

The topology[1] of a plant is characterized by a recursive branching structure. To this extent, plants share with fractals the characteristics of self-similarity under scale. The two-dimensional branching structures typically of interest are shown in Figure 12.9. The three-dimensional branching structures are analogous.

An encoding of the branching structure of a given plant is one of the objectives of plant modeling. Plants are immensely varied, yet most share many common characteristics. These shared characteristics allow efficient representations to be formed by abstracting out the features that are common to plants of interest. But the representation of the static structure of a mature plant is only part of the story. Because a plant is a living thing, it is subject to changes due to growth. The modeling and animation of the growth process is the subject of this section.

12.2.1 A little bit of botany

Botany is, of course, useful when trying to model and animate realistic-looking plants. For computer graphics and animation, it is only useful to the extent that it addresses the visual characteristics of the plant. Thus, the structural components and surface elements of plants are briefly reviewed here.

[1]The term *topology*, as applied to describing the form of plants, refers to the number and arrangement of convex regions of the plant delineated by concave areas of attachment to other convex regions.

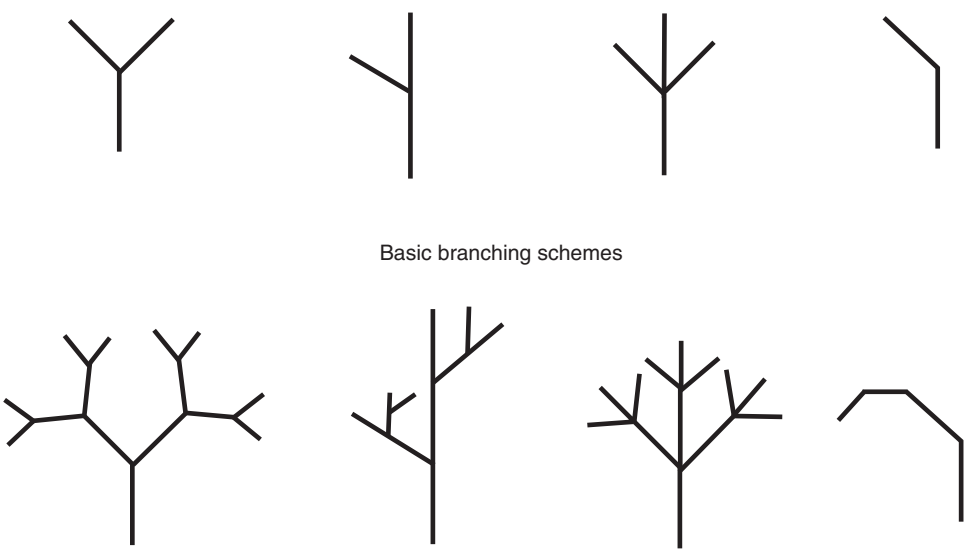

FIGURE 12.9

Branching structures of interest in two dimensions.

Simplifications are made to highlight the information most relevant for computer graphics modeling and animation.

The structural components of plants are *stems*, *roots*, *buds*, *leaves*, and *flowers*. Roots are typically not of interest when modeling the visual aspects of plants and have not been incorporated into these plant models. Most plants of interest in visualization have a definite branching structure. Such plants are either *herbaceous* or *woody*. The latter are larger plants whose branches are heavier and more structurally independent. The branches of woody plants tend to interfere and compete with one another. They are also more subject to the effects of wind, gravity, and sunlight. Herbaceous plants are smaller, lighter plants, such as ferns and mosses, whose branching patterns tend to be more regular and less subject to environmental effects.

Stems are usually above ground, grow upward, and bear leaves. The leaves are attached in a regular pattern at nodes along the stem. The portions of the stem between the nodes are called *internodes*. *Branching* is the production of subordinate stems from a main stem (the *axis*). Branches can be formed by the main stem bifurcating into two equally growing stems (*dichotomous*), or they can be formed when a stem grows laterally from the main axis while the main axis continues to grow in its original direction (*monopodial*).

Buds are the embryonic state of stems, leaves, and flowers; they are classified as either *vegetative* or *flower buds*. Flower buds develop into flowers, whereas vegetative buds develop into stems or leaves. A bud at the end of a stem is called a *terminal* bud; a bud that grows along a stem is called a *lateral* bud. Not all buds develop; nondeveloping buds are referred to as *dormant*. Sometimes in woody plants, dormant buds will suddenly become active, producing young growth in an old-growth area of a tree.

Leaves grow from buds. They are arranged on a stem in one of three ways: *alternate*, *opposite*, or *whorled*. Alternate means that the leaves shoot off one side of a stem and then off the other side in an alternating pattern. Opposite means that a pair of leaves shoot off the stem at the same point, but on opposite sides, of the stem. Whorled means that three or more leaves radiate from a node.

Growth of a cell in a plant has four main influences: *lineage*, *cellular descent*, *tropisms*, and *obstacles*. Lineage refers to growth controlled by the age of the cell. *Cellular descent* refers to the passing of nutrients and hormones from adjacent cells. Growth controlled exclusively by lineage would result in older cells always being larger than younger cells. Cellular descent can be bidirectional, depending on the growth processes and reactions to environmental influences occurring in the plant. Thus, cellular descent is responsible for the ends of some plants growing more than the interior sections. Plant hormones are specialized chemicals produced by plants. These are the main internal factors that control the plant's growth and development. Hormones are produced in specific parts of the plants and are transported to others. The same hormone may either promote or inhibit growth, depending on the cells it interacts with.

Tropisms are an important class of responses to external influences that change the direction of a plant's growth. They include *phototropism*, the bending of a stem toward light, and *geotropism*, the response of a stem or root to gravity. Physical obstacles also affect the shape and growth of plants. Collision detection and response can be calculated for temporary changes in the plant's shape. Permanent changes in the growth patterns can occur when such forces are present for extended periods.

12.2.2 L-systems

DOL-systems

L-systems are parallel rewriting systems. They were conceived as mathematical models of plant development by the biologist Aristid Lindenmayer (the "L" in L-systems). The simplest class of L-system is deterministic and context free (as in Figure 12.10); it is called a D0L-system.[2] A D0L-system is a set of production rules of the form $\alpha_i \rightarrow \beta_i$, in which α_i, the *predecessor*, is a single symbol and β_i, the *successor*, is a sequence of symbols. In deterministic L-systems, α_i occurs only once on the left-hand side of a production rule. A sequence of one or more symbols is given as the initial string, or *axiom*. A production rule can be applied to the string if its left-hand side occurs in the string. The effect of applying a production rule to a string means that the occurrence of α_i in the string is rewritten as β_i. Production rules are applied in parallel to the initial string. This replacement happens in parallel for all

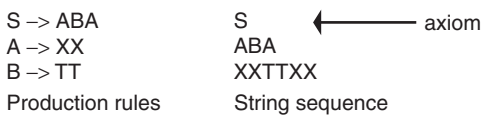

FIGURE 12.10

Simple D0L-system and the sequence of strings it generates.

[2]The D in D0L clearly stands for *deterministic;* the 0 indicates, as is more fully explained later, that the productions are context free.

occurrences of any left-hand side of a production in the string. Symbols of the string that are not on the left-hand side of any production rule are assumed to be operated on by the identity production rule, $\alpha_i \rightarrow \beta_i$. The parallel application of the production rules produces a new string. The production rules are then applied again to the new string. This happens iteratively until no occurrences of a left-hand side of a production rule occur in the string. Sample production rules and the string they generate are shown in Figure 12.10.

Geometric interpretation of L-systems

The strings produced by L-systems are just that—strings. To produce images from such strings, one must interpret them geometrically. Two common ways of doing this are *geometric replacement* and *turtle graphics*. In geometric replacement, each symbol of a string is replaced by a geometric element. For example, the string XXTTXX can be interpreted by replacing each occurrence of X with a straight line segment and each occurrence of T with a V shape so that the top of the V aligns with the endpoints of the geometric elements on either side of it (see Figure 12.11).

In turtle graphics, a geometry is produced from the string by interpreting the symbols of the string as drawing commands given to a simple cursor called a turtle. The basic idea of turtle graphics interpretation, taken from Prusinkiewicz and Lindenmayer [16], uses the symbols from Table 12.1 to control

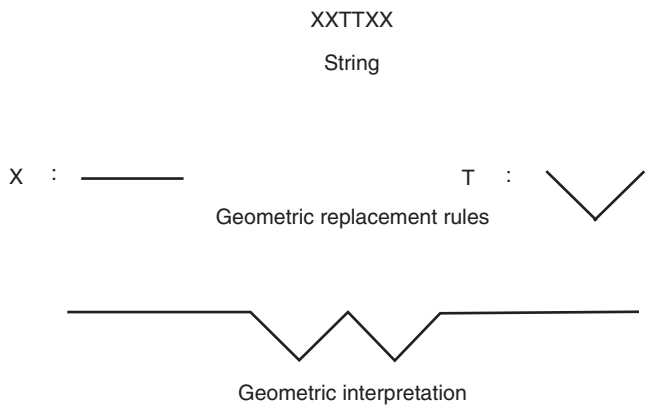

FIGURE 12.11

Geometric interpretation of a simple string.

Table 12.1 Turtle Graphics Commands

Symbol	Turtle Graphic Interpretation
F	Move forward a distance d while drawing a line. Its state will change from (x, y, α) to $(x + d \cos \alpha, y + d \sin \alpha, \alpha)$.
f	Move forward a distance d without drawing a line. Its state will change as above.
+	Turn left by an angle δ. Its state will change from (x, y, α) to $(x, y, \alpha+\delta)$.
−	Turn right by an angle δ. Its state will change from (x, y, α) to (x, y, $\alpha-\delta$).

the turtle. The state of the turtle at any given time is expressed as a triple (x, y, α), where x and y give its coordinates in a two-dimensional space and α gives the direction it is pointing relative to some given reference direction (here, a positive angle is measured counterclockwise from the reference direction). The values d and δ are user-specified system parameters and are the linear and rotational step sizes, respectively.

Given the reference direction, the initial state of the turtle (x_0, y_0, α_0), and the parameters d and δ, the user can generate the turtle interpretation of a string containing the symbols of Table 12.1 (Figure 12.12).

Bracketed L-systems

The D0L-systems described previously are inherently linear, and the graphical interpretations reflect this. To represent the branching structure of plants, one introduces a mechanism to represent multiple segments attached at the end of a single segment. In *bracketed L-systems*, brackets are used to mark the beginning and the end of additional offshoots from the main lineage. The turtle graphics interpretation of the brackets is given in Table 12.2. A stack of turtle graphic states is used, and the brackets push and pop states onto and off this stack. The state is defined by the current position and orientation of the turtle. This allows branching from a stem to be represented. Further, because a stack is used, it allows an arbitrarily deep branching structure.

Figure 12.13 shows some production rules. The production rules are context free and *nondeterministic*. They are context free because the left-hand side of the production rule does not contain any

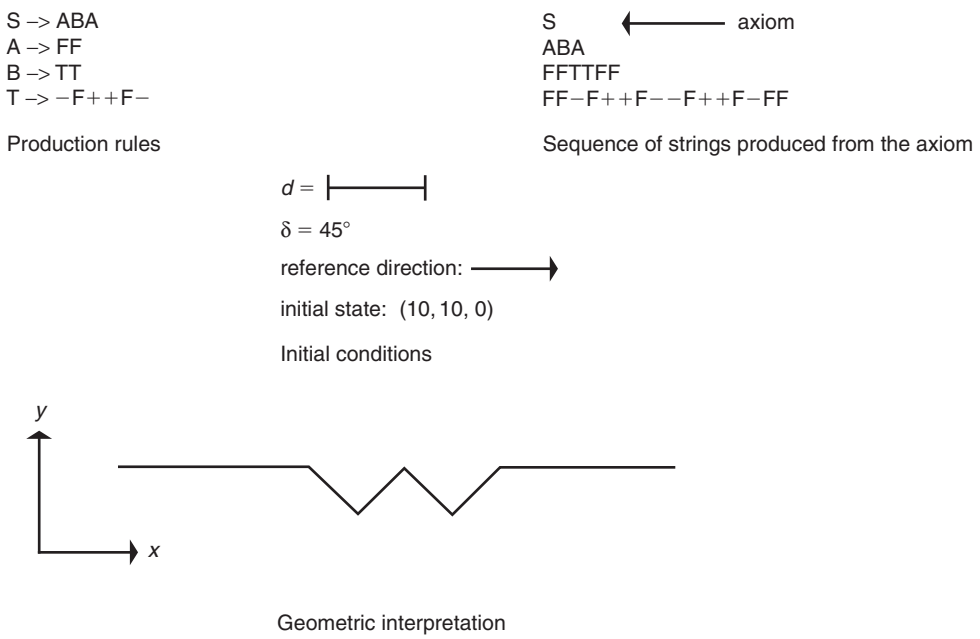

FIGURE 12.12

Turtle graphic interpretation of a string generated by an L-system.

Table 12.2 Turtle Graphic Interpretation of Brackets

Symbol	Turtle Graphic Interpretation
[	Push the current state of the turtle onto the stack
]	Pop the top of the state stack and make it the current state

$$S \Rightarrow FAF$$
$$A \Rightarrow [+FBF]$$
$$A \Rightarrow F$$
$$B \Rightarrow [-FBF]$$
$$B \Rightarrow F$$

FIGURE 12.13

Nondeterministic, context-free production rules.

context for the predecessor symbol. They are nondeterministic because there are multiple rules with identical left-hand sides; one is chosen at random from the set whenever the left-hand side is to be replaced. In this set of rules, *S* is the start symbol, and *A* and *B* represent locations of possible branching; *A* branches to the left and *B* to the right. The production stops when all symbols have changed into ones that have a turtle graphic interpretation.

Figure 12.14 shows some possible terminal strings and the corresponding graphics produced by the turtle interpretation. An added feature of this turtle interpretation of the bracketed L-system is the reduction of the size of the drawing step by one-half for each branching level, where *branching level* is defined by the current number of states on the stack. L-systems can be expanded to include attribute symbols that explicitly control line length, line width, color, and so on [16] and that are considered part of the state.

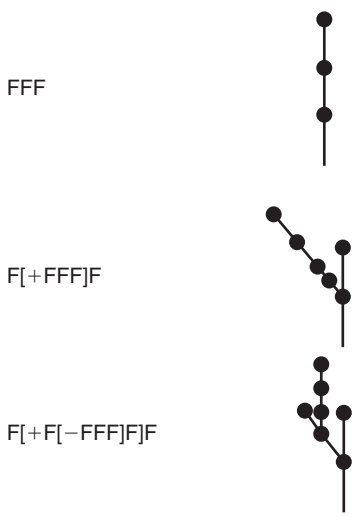

FIGURE 12.14

Some possible terminal strings.

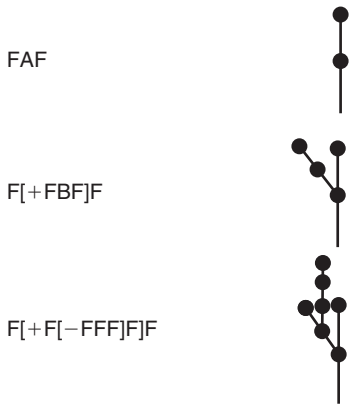

FIGURE 12.15

Sequence of strings produced by bracketed L-system.

Not only does this representation admit *database amplification* [19], but the expansion of the start symbol into a terminal string parallels the topological growth process of the plants. In some sense, the sequence of strings that progress to the final string of all turtle graphic symbols represents the growth of the plant at discrete events in its evolution (see Figure 12.15). This addresses one of the animation issues with respect to plants—that of animating the development of the branching structure. However, the gradual appearance and subsequent elongation of elements must also be addressed if a growing structure is to be animated in a reasonable manner.

Stochastic L-systems

The previous section introduced nondeterminism into the concept of L-systems, but the method used to select the possible applicable productions for a given symbol was not addressed. *Stochastic L-systems* assign a user-specified probability to each production so that the probabilities assigned to productions with the same left-hand side sum to one. These probabilities indicate how likely it is that the production will be applied to the symbol on a symbol-by-symbol basis.

Consider the productions of Figure 12.13 being assigned the probabilities shown in Figure 12.16. These probabilities will control how likely a production will be to form a branch at each possible branching point. In this example, left branches are very likely to form, while right branches are somewhat unlikely. However, any arbitrarily complex branching structure has a non-zero probability of occurring. Using such stochastic (nondeterministic) L-systems, one can set up an L-system that produces a wide variety of branching structures that still exhibit some family-like similarity [16].

$$
\begin{aligned}
S_{1.0} &\Rightarrow FAF \\
A_{0.8} &\Rightarrow [+FBF] \\
A_{0.2} &\Rightarrow F \\
B_{0.4} &\Rightarrow [-FBF] \\
B_{0.6} &\Rightarrow F
\end{aligned}
$$

FIGURE 12.16

Stochastic L-system.

Production rules	String sequence
$S \Rightarrow FAT$	S
$A > T \Rightarrow [+FBF]$	FAT
$A > F \Rightarrow F$	F[+FBF]F
$B \Rightarrow [-FAF]$	F[+F[−FAF]F]F
$T \Rightarrow F$	

FIGURE 12.17

Context-sensitive L-system production rules.

Context free versus context sensitive

So far, only context-free L-systems have been presented. *Context-sensitive L-systems* add the ability to specify a context, in which the left-hand side (the predecessor symbol) must appear in order for the production rule to be applicable. For example, in the deterministic productions of Figure 12.17,[3] the symbol A has different productions depending on the context in which it appears. The context-sensitive productions shown in the figure have a single right-side context symbol in two of the productions. This concept can be extended to n left-side context symbols and m right-side context symbols in the productions, called (n, m) *L-systems*, and, of course, they are compatible with nondeterministic L-systems. In (n, m) L-systems, productions with fewer than n context symbols on the left and m on the right are allowable. Productions with longer contexts are usually given precedence over productions with shorter contexts when they are both applicable to the same symbol. If the context is one-sided, then L-systems are referred to as nL-systems, where n is the number of context symbols and the side of the context is specified independently.

12.2.3 Animating plant growth

There are three types of animation in plants. One type is the flexible movement of an otherwise static structure, for example, a plant being subjected to a high wind. Such motion is an example of a flexible body reacting to external forces and is not dealt with in this section. The other types of animation are particular to plants and involve the modeling of the growth process.

The two aspects of the growth process are (1) changes in topology that occur during growth and (2) the elongation of existing structures. The topological changes are captured by the L-systems already described. They occur as discrete events in time and are modeled by the application of a production that encapsulates a branching structure, as in $A \Rightarrow F[+F]B$.

Elongation can be modeled by productions of the form $F \Rightarrow$ FF. The problem with this approach is that growth is chunked into units equal to the length of the drawing primitive represented by F. If F represents the smallest unit of growth, then an internode segment can be made to grow arbitrarily long. But the production rule $F \Rightarrow$ FF lacks termination criteria for the growth process. Additional drawing symbols can be introduced to represent successive steps in the elongation process, resulting in a series of productions $F_0 \Rightarrow F_1$, $F_1 \Rightarrow F_2$, $F_2 \Rightarrow F_3$, $F_3 \Rightarrow F_4$, and so on. Each symbol would represent a drawing operation of a different length. However, if the elongation process is to be modeled in,

[3] In the notation used here, as it is in the book by Prusinkiewicz and Lindenmayer [16], the predecessor is the symbol on the "greater than" side of the inequality symbols. This is used so that the antecedent can be visually identified when using either left or right contexts.

say, one hundred time steps, then approximately one hundred symbols and productions are required. To avoid this proliferation of symbols and productions, the user can represent the length of the drawing operation parametrically with the drawing symbol in *parametric L-systems*.

Parametric L-systems

In parametric L-systems, symbols can have one or more parameters associated with them. These parameters can be set and modified by productions of the L-system. In addition, optional conditional terms can be associated with productions. The conditional expressions are in terms of parametric values. The production is applicable only if its associated condition is met. In the simple example of Figure 12.18, the symbol A has a parameter t associated with it. The productions create the symbol A with a parameter value of 0.0 and then increase the parametric value in increments of 0.01 until it reaches 1.0. At this point the symbol turns into an F.

Context-sensitive productions can be combined with parametric systems to model the passing of information along a system of symbols. Consider the production of Figure 12.19. In this production, there is a single context symbol on both sides of the left-hand symbol that is to be changed. These productions allow for the relatively easy representation of such processes as passing nutrients along the stem of a plant.

Timed L-systems

Timed L-systems add two more concepts to L-systems: a *global time variable*, which is accessible to all productions and which helps control the evolution of the string; and a *local age value*, τ_i, associated with each letter μ_i. Timed L-system productions are of the form shown in Equation 12.6. By this production, the letter μ_0 has a *terminal age* of β_0 assigned to it. The terminal age must be uniquely assigned to a symbol. Also by this production, each symbol μ_i has an *initial age* of α_i assigned to it. The terminal age assigned to a symbol μ_i must be larger than its initial age so that its *lifetime*, $\beta_i - \alpha_i$, is positive.

$$(\mu_0, \beta_0) \Rightarrow ((\mu_1, \alpha_1), (\mu_2, \alpha_2), \ldots, (\mu_n, \alpha_n)) \tag{12.6}$$

A timed production can be applied to a matching symbol when that symbol's terminal age is reached. When a new symbol is generated by a production, it is commonly initialized with an age of zero. As global time progresses from that point, the local age of the variable increases until its terminal age is reached, at which point a production is applied to it and it is replaced by new symbols.

A string can be derived from an axiom by jumping from terminal age to terminal age. At any point in time, the production to be applied first is the one whose predecessor symbol has the smallest

```
S              => A(0)
A(t)           => A(t + 0.01)
A(t) : t>=1.0  => F
```

FIGURE 12.18

Simple parametric L-system.

```
A(t0)<A(t1)>A(t2): t2>t1 & t1>t0 => A(t1+0.01)
```

FIGURE 12.19

Parametric, context-sensitive L-system production.

axiom: (*A*,0)

(*A*,3) => (*S*,0) [+ (*B*,0)] (*S*,0)
(*B*,2) => (*S*,0)

FIGURE 12.20

Simple timed L-system.

difference between its terminal age and its local age. Each symbol appearing in a string has a local age less than its terminal age. The geometric interpretation of each symbol is potentially based on the local age of that symbol. Thus, the appearance of buds and stems can be modeled according to their local age.

In the simple example of Figure 12.20, the symbol *A* can be thought of as a plant seed; *S* can be thought of as an internode stem segment; and *B* can be thought of as a bud that turns into the stem of a branch. After three units of time the seed becomes a stem segment, a lateral bud, and another stem segment. After two more time units, the bud develops into a branching stem segment.

Interacting with the environment

The environment can influence plant growth in many ways. There are local influences such as physical obstacles, including other plants and parts of the plant itself. There are global influences such as amount of sunlight, length of day, gravity, and wind. But even these global influences are felt locally. The wind can be blocked from part of the plant, as can the sun. And even gravity can have more of an effect on an unsupported limb than on a supported part of a vine. The nutrients and moisture in the soil affect growth. These are transported throughout the plant, more or less effectively depending on local damage to the plant structure.

Mech and Prusinkiewicz [11] describe a framework for the modeling and animation of plants that bidirectionally interacts with the environment. They describe *open L-systems*, in which communication terms of the form ? $E(x_1, x_2, \ldots, x_m)$ are used to transmit information as well as request information from the environment. In turtle graphic interpretation of an L-system string, the string is scanned left to right. As communication terms are encountered, information is transmitted between the environment and the plant model. The exact form of the communication is defined in an auxiliary specification file so that only relevant information is transmitted. Information about the environment relevant to the plant model includes distribution of nutrients, direction of sunlight, and length of day. The state of the plant model can be influenced as a result of this information and can be used to change the rate of elongation as well as to control the creation of new offshoots. Information from the plant useful to the environmental model includes use of nutrients and shade formation, which, in turn, can influence other plant models in the environment.

12.2.4 Summary

L-systems, in all the variations, are an interesting and powerful modeling tool. Originally intended only as a static modeling tool, L-systems can be used to model the time-varying behavior of plantlike growth and motion. See Figure 12.21 (Color Plate 13) for an example from a video that used L-systems to animate plantlike creatures. Because of the iterative nature of string development, the topological

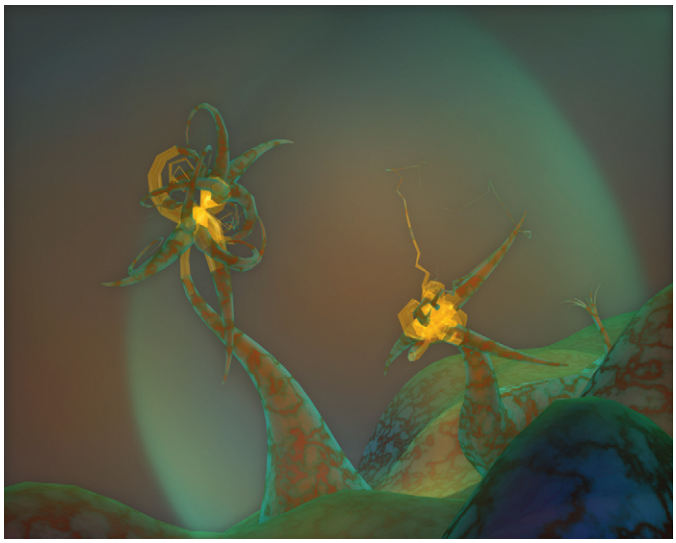

FIGURE 12.21

Example from video using L-systems to animate plantlike figures.

(Image courtesy of Vita Berezine-Blackburn, ACCAD.)

changes of plant growth can be successfully captured by L-systems. By adding parameters, time variables, and communication modules, one can model other aspects of plant growth. Most recently, Deussen et al. [5] have used open L-systems to model plant ecosystems.

12.3 Subdivision surfaces

Subdivision surfaces are useful in animation for designing objects in a top-down fashion [4]. Starting from a coarse polyhedron, the geometry is refined a step at a time. Each step introduces more complexity to the object, usually by rounding corners and edges. There are various methods proposed in the literature for conducting the subdivision and much research has been performed in determining what the limit surfaces look like and what their mathematical properties are.

As a simple subdivision example, consider the sequence in Figure 12.22 in which each vertex is replaced by a face made from vertices one-third along the way down each edge emanating from the vertex (Figure 12.23) and each face is redefined to include the new vertices on the original edges (Figure 12.24).

One of the more popular subdivision schemes is due to Charles Loop [9]. For a closed, two-dimensional, triangulated manifold, the Loop subdivision method creates a new vertex at the midpoint of each edge and is repositioned. In addition, each original vertex is repositioned, and each triangle is divided into four new triangles (Figure 12.25). The new edge midpoints are positioned according to $V_M = (3\,V_1 + 3\,V_2 + V_A + V_B)/8$ where V_1 and V_2 are the end vertices of the original edge and

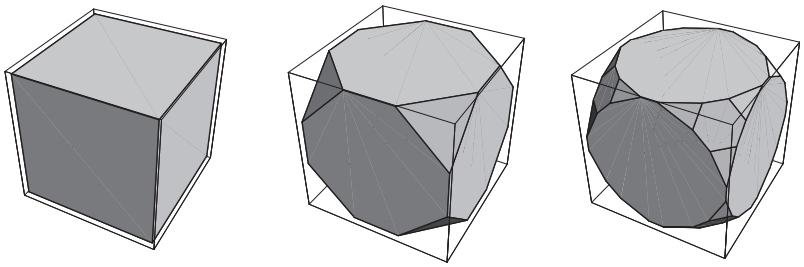

FIGURE 12.22

Sequence of subdivision steps.

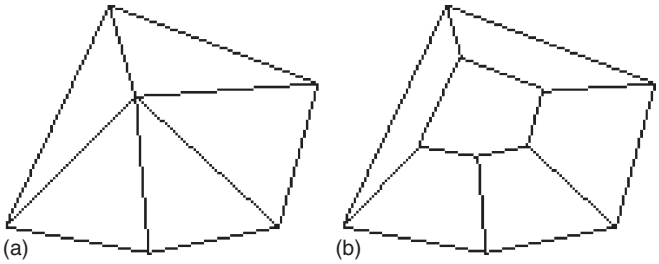

FIGURE 12.23

During subdivision, each vertex is replaced by a face. (a) Original vertex of object to be subdivided. (b) A face replaces the vertex by using new vertices defined on connecting edges.

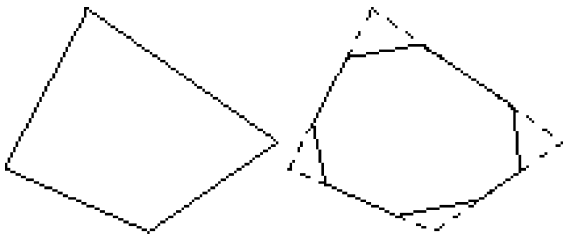

FIGURE 12.24

During subdivision, each face is replaced by a face using the newly defined vertices on its edges. Original face of object to be subdivided.

V_A and V_B are the other vertices of the faces that share the edge. Each original vertex is repositioned by a weighted average of its original position and those of all the vertices connected to it by a single edge. There have been many schemes proposed for repositioning the original vertices. A particularly simple one is $V' = (1 - s)V + s\sum_{i=1}^{n} W_i$ where V is the original vertex position and the n W_{is} are the connected vertices, $s = \frac{3}{4}n$.

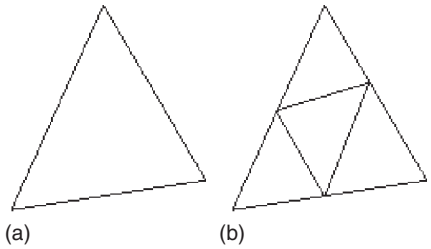

FIGURE 12.25

Loop subdivision. (a) Original triangle. (b) After subdivision.

12.4 Chapter summary

Implicit surfaces are useful in a variety of situations, most notably when trying to capture the qualities of organic shapes. Plants exhibit enormous complexity, a well-defined structure, and almost endless variety. Subdivision surfaces have proven to be a powerful design tool in animation applications. These qualities make capturing their essence both intriguing and fascinating.

References

[1] Aono M, Kunii TL. Botanical Tree Image Generation. IEEE Comput Graph Appl May 1984;4(5):10–34.
[2] Bloomenthal J. Modeling the Mighty Maple. In: Barsky BA, editor. Computer Graphics. Proceedings of SIGGRAPH 85, vol. 19(3). San Francisco, Calif.; August 1985. p. 305–11.
[3] Bloomenthal J, Bajaj C, Blinn J, Cani-Gascuel M-P, Rockwood A, Wyvill B, et al. Introduction to Implicit Surfaces. San Francisco: Morgan Kaufmann; 1997.
[4] DeRose T, Kass M, Truong T. Subdivision Surfaces for Character Animation. In: Cohen M, editor. Computer Graphics. Proceedings of SIGGRAPH 98, Annual Conference Series. Orlando, Fla.: Addison-Wesley; July 1998. p. 85–94. ISBN 0-89791-999-8.
[5] Deussen O, Hanrahan P, Lintermann B, Mech R, Pharr M, Prusinkiewicz P. Realistic Modeling and Rendering of Plant Ecosystems. In: Cohen M, editor. Computer Graphics. Proceedings of SIGGRAPH 98, Annual Conference Series. Orlando, Fla: Addison-Wesley; July 1998. p. 275–86. ISBN 0-89791-999-8.
[6] Gascuel M-P. An Implicit Formulation for Precise Contact Modeling between Flexible Solids. In: Kajiya JT, editor. Computer Graphics. Proceedings of SIGGRAPH 93, Annual Conference Series. Anaheim, Calif.; August 1993. p. 313–20. ISBN 0-201-58889-7.
[7] Gascuel J-D, Gascuel M-P. Displacement Constraints for Interactive Modeling and Animation of Articulated Structures. Visual Computer March 1994;10(4):191–204 ISSN 0-178-2789.
[8] Greene N. Voxel Space Automata: Modeling with Stochastic Growth Processes in Voxel Space. In: Lane J, editor. Computer Graphics. Proceedings of SIGGRAPH 89, vol. 23(3). Boston, Mass.; July 1989. p. 175–84.
[9] Loop C. Smooth Subdivision Surfaces Based on Triangles. M.S. Thesis. University of Utah; 1987.
[10] Lorensen W, Cline H. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In: Proceedings of SIGGRAPH 87, vol. 21(4). p. 163–9.
[11] Mech R, Prusinkiewicz P. Visual Models of Plants Interacting with Their Environment. In: Rushmeier H, editor. Computer Graphics. Proceedings of SIGGRAPH 96, Annual Conference Series. New Orleans, La.: Addison-Wesley; August 1996. p. 397–410. ISBN 0-201-94800-1.

[12] Oppenheimer P. Real-Time Design and Animation of Fractal Plants and Trees. In: Evans DC, Athay RJ, editors. Computer Graphics. Proceedings of SIGGRAPH 86, vol. 20(4). Dallas, Tex.; August 1986. p. 55–64.
[13] Osher S, Fedkiw R. Level Set Methods and Dynamic Implicit Surfaces. New York: Springer-Verlag; 2003.
[14] Prusinkiewicz P, Hammel M, Mjolsness E. Animation of Plant Development. In: Kajiya JT, editor. Computer Graphics. Proceedings of SIGGRAPH 93, Annual Conference Series. Anaheim, Calif.; August 1993. p. 351–60. ISBN 0-201-58889-7.
[15] Prusinkiewicz P, James M, Mech MR. Synthetic Topiary, In: Glassner A, editor. Computer Graphics. Proceedings of SIGGRAPH 94, Annual Conference Series. Orlando, Fla.: ACM Press; July 1994. p. 351–8. ISBN 0-89791-667-0.
[16] Prusinkiewicz P, Lindenmayer A. The Algorithmic Beauty of Plants. New York: Springer-Verlag; 1990.
[17] Prusinkiewicz P, Lindenmayer A, Hanan J. Developmental Models of Herbaceous Plants for Computer Imagery Purposes. In: Dill J, editor. Computer Graphics. Proceedings of SIGGRAPH 88, vol. 22(4). Atlanta, Ga.; August 1988. p. 141–50.
[18] Reffye P, Edelin C, Francon J, Jaeger M, Puech C. Plant Models Faithful to Botanical Structure and Development. In: Dill J, editor. Computer Graphics. Proceedings of SIGGRAPH 88, vol. 22(4). Atlanta, Ga.; August 1988. p. 151–8.
[19] Smith AR. Plants, Fractals, and Formal Languages, In: Computer Graphics. Proceedings of SIGGRAPH 84, vol. 18(3). Minneapolis, Minn.; July 1984. p. 1–10.
[20] Wyvill B, McPheeters C, Wyvill G. Data Structure for Soft Objects. Visual Computer 1986;2(4):227–34.