TREE-MAKER: A User Tool

William Kenneth Neighbors III
Department of Aeronautics and Astronautics
Stanford University
Stanford, California 94305

Larry F. Hodges
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

TREE-MAKER was designed as a tool for generating and rendering realistic images of trees. Each tree is created from a *tree template* that specifies simple branching rules and leaf definitions. Many different types of trees may be created by varying the parameters of the template. The flexibility provided by this approach makes it suitable for users who are not familiar with programming techniques for modeling natural objects by still allowing them creative control of the image. Users are free, then, to concentrate on the design of trees, not the programming of special techniques.

Key Words: *Particle System, Fractal, Self-Similarity, Tree Template.*

I. Introduction

Graphic models of natural phenomena such as clouds, plants, coastlines, ocean waves, or mountains must reproduce an overwhelming amount of variety and detail. Methods used to model natural objects usually rely on some type of recursive self-similarity to construct complex objects from a relatively small amount of information. These methods can be based upon fractals, grammars, iterated function systems, or particle systems [4,10].

In this paper we describe a program for generating and rendering images of trees. Several different techniques have been described in the computer graphics literature during the past five years to model trees. These techniques can be adapted from mathematical theory, or can be based on a model of the biological or geometric characteristics of trees. For example, Oppenheimer has created trees with spiraling branches, a characteristic of many fractal plants [6]. Barnsley, et. al. encodes and models shapes and textures of two-dimensional images of trees and leaves using iterated function systems[2]. Another adapted mathematical theory used to model plants is L-systems, such as in Prusinkiewicz, Lindenmayer, and Hanan's attempt to capture the developmental process as a key to realism [7].

These mathematical methods, as well as the biological models, usually use some type of branching rule to specify the angles of the branches for a particular species of tree. Aono and Kunii have used a set of botanical rules in a grammatical form to generate natural looking branches that model real trees [1]. The branching rules are based on the true biological characteristics of the tree. This approach has been expanded by de Reffye, et. al. to model a richer variety of plants and trees [5]. These models also include the effects of wind and gravity.

Regardless of the method of modeling, a realistic image must be created from the tree model. With this task in mind, it is advantageous to base the model on what a tree looks like. Bloomenthal uses this approach in modeling the maple tree [3]. The limb surface, leaves, and shadows are calculated exactly, including a bump map from real bark, creating a very realistic looking maple tree. However, when close up views are unimportant and a large number of trees must be generated, as in a forest, it is necessary to use stochastic modeling and more approximate rendering. This method reduces the work of the user describing the tree and of the computer rendering the image. Bill Reeves' forests are a splendid example of this method using structured particle systems [8]. The shading and visible surface elimination are approximations, but this has no noticeable effect because of the richness in detail.

Because of its efficiency and richness in detail we chose to follow a particle system approach with probabilistic branching while developing TREE-MAKER. Our goal was to create a tool which could be used by anyone with a basic knowledge of computers (i.e., how to use the editor and execute a program on the host system). This approach allows the user to concentrate on the design of trees, not the programming of special techniques. This freedom is accomplished through the use of a *tree template* to specify simple branching rules and leaf definitions. The information in the template describes the lighting model, rules for growth, and leaves (or flowers) of a particular tree. Many different types of trees may be created by varying the parameters of the template.

II. Growth of the Tree

The growth of the tree is much like a particle system described by Reeves [9]. A system of particles is repeatedly updated with growth vectors, and new particles are added using the branching rules from the template. At each iteration, the current position of all particles is stored, and the paths of the particles define the limbs.

The branching rule defines the range for the number of branches off a limb and specifies an acceptable range for the direction and angle of the branch. These two angles are shown in figure 1. All ranges given in the tree template are uniformly distributed random variables. Another parameter in the rule specifies how much to reduce the growth rate of the current branch compared to that of the parent branch. A set of these three parameters, augmented with a probability of occurrence for each triplet, constitutes a branching rule in the tree template. The format of a branching rule is shown in figure 6. This time-invariant stochastic branching rule suffices for those trees with random branching. For trees with more systematic

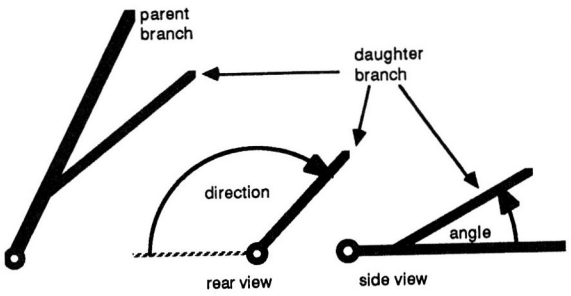


Figure 1. Branching Angles

branching, such as alternating or spiral patterns, a time varying set of probabilities would have to be implemented, where the probabilities are varied according to previous branching selections.

To build a tree, the program starts with a single particle on the ground with a growth vector pointing up, as specified in the tree template. The current position of this particle is stored, and then the next position is calculated by adding the growth vector to the current position. The program shortens the growth vector to make subsequent branches closer together, using an aging factor from the tree template. The direction of the vector is also changed slightly toward the ground by subtracting a gravity acceleration from the z-component (up), and re-normalizing to keep the magnitude the same. This gravity acceleration is calculated from a gravity parameter in the tree template and the cosine of the the angle between the growth vector and the horizontal.

The program then checks the branching rules to decide how many branches will occur and in which direction they will grow. A particle is added to the system at the current position for every new branch, with its own growth vector determined from the branching rule and the parent branch's growth vector. A width is stored for each of the particle's past positions, and these widths are incremented by an amount specified in the tree template. This procedure is repeated until all limbs have been generated as shown in figure 2.

After the tree has grown through the number of iterations specified in the tree template, the program adds a leaf to the end of every branch. The user defines these leaves in the template as a set of triangles and lines. Flowers and fruits can also be defined in the same way, and the user states the probability of each different type of leaf in the template. The program transforms this set of points to place the leaf on the end of each branch in a random orientation limited by the constraints in the template.

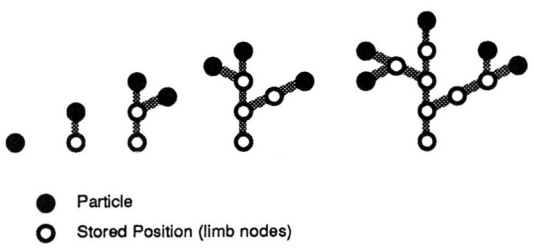


Figure 2. Growth of the Tree

III. Shading and Rendering

Shading of each element of the tree depends on its position in the tree as well as its orientation with respect to the sun. Five parameters specified in the template determine the relative weight to give to each of five shading criteria. Ambient light and Lambert's cosine law, as well as height, radius, and angle factors affect the amount of shade to give an element. For example, leaves that are closer to the ground, closer to the trunk, or on the backside of the tree relative to the sun are shaded darker to simulate the shadowing of leaves upon themselves. Figure 3 illustrates these shading techniques.

When calculating a normal, N, for leaf line elements and limbs, the ambiguity in direction is resolved by calculating the normal that lies in the plane with the light source. The radius, r_0, and height, h_0, of the tree is stored as the tree is growing, where the radius is the largest horizontal distance of any particle from the first particle's position. The height factor, h/h_0, is a ratio of the height of the element over the height of the tree, and the radius factor, r/r_0, is a ratio of the element's distance from a vertical trunk over the radius of the tree. The angle, q, is measured relative to the trunk and the light source.

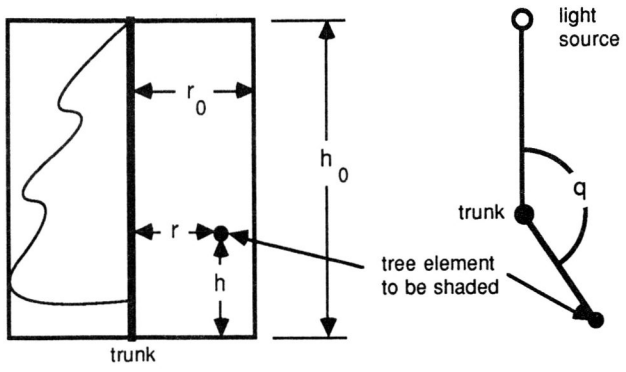
To add more realism to the image, the program generates a shadow of the tree by copying the final tree representation completely, and then casting all the points onto the ground, using a vector from the sun. The colors of this shadow tree are then changed to the shadow color of the ground.

After shading, the program renders the tree by performing the viewing transformation and then adding the elements to a z-buffer. Three graphics primitives are used: lines and triangles for the leaves, and 4-sided polygons for the limbs. To reduce computation, each primitive is added to the z-buffer as a whole, instead of calculating the z-position of each pixel in the primitive. This approximation is not very noticeable since the elements are many and their sizes are small.

IV. Data Structures

The tree is represented in memory as a multiple linked list. Figure 5 shows the major nodes and links in this representation. The *branching rules* are linked together in order of their levels. Level 1 branches occur off the trunk; level 2 branches occur off these; etc. It is not necessary to have a branching rule for each level because the program will use the nearest level less than the current level. Each branching rule consists of three randomized branching parameters that determine the characteristics of new particles added to the system: angle and direction of branch and growth reduction factor.

A *leaf definition* consists of the leaf elements (triangles and lines), coordinate system (leaf or limb), and allowable rotation and deviation in direction. The limb coordinate system is used for trees when the direction of the major axis of the leaf is determined from the way the branch is pointing. The leaf coordinate system is used when the major axis of the leaf depends on the horizontal, such as when a heavy leaf is hanging toward the ground. These coordinate systems are shown in figure 4.



N = normal to tree element
L = vector to light source from element

shade = MAX_INTENSITY * (ambient_parameter +
lambert_parameter * (N•L) +
height_parameter * h/h_0 +
radius_parameter * r/r_0 +
angle_parameter * sin $\frac{q}{2}$)

Figure 3. Shading Techniques

The actual structure of the tree is made up of linked lists of branches, limbs, and particles. The *branch node* has pointers to the appropriate branching rule for its level, to the first limb node, and to the particle at the end of the branch. It also has a level number and base color for the limbs. The *limb nodes* consist of its position, width, lateral growth rate and aging factor, color, and pointer to the next limb node. The *particle node* is at the end of the limb list. This node consists of its position, longitudinal growth vector and aging factor, as well as pointers to the last limb in the list for adding new limbs, and to the first leaf element when the leaves are added. This structure is also shown in figure 5.

V. A Tree Template

The tree template contains the characteristics to direct the program in creating a tree. At least one branching rule, one particle, and a leaf definition are absolutely necessary. The general format of the template is shown in figure 6. As a particular example we present a template for generating a dogwood tree.

```
iterations { i= 13 }

gravity   { g= 0.08 }

light     { ambient=0.1 lambert=0.4 height=0.4 radius=0.3
            angle=0.2 }

color     { green= 0,1,0 }
color     { brown= 1,0.9,0.2 }
color     { white= 1,1,1 }

branch    { level= 1  number= 2.0 to 3.0
            (p=1  angle= 40 to 80  dir= 0 to 360  factor= 0.5 to
            0.7)
            }

branch    { level= 2  number= 0.4 to 1.6
            (p=0.5  angle= 40 to 60  dir= 0 to 20  factor= 0.8 to
            1.0)
            (p=0.5  angle= 40 to 60  dir= 160 to 180  factor= 0.8
            to 1.0)
            }
branch    { level= 3  number = 0.4 to 1.6
            (p=0.45  angle= 40 to 60  dir= 0 to 20  factor= 0.8 to
            1.0)
            (p=0.45  angle= 40 to 60  dir= 160 to 180  factor=
            0.8 to 1.0)
            (p=0.10  angle= 40 to 80  dir= 0 to 180  factor= 0.1
            to 0.2)
            }
branch    { level= 7  number= 0.4 to 1.5
            (p=0.5  angle= 30 to 90  dir= 0 to 50  factor= 0.6 to
            1.0)
            (p=0.5  angle= 30 to 90  dir= 130 to 180  factor= 0.6
            to 1.0)
            }
branch    { level= 9  number= 0.4 to 1.5
            (p=1  angle= 50 to 110  dir= 0 to 180  factor= 1.0 to
            1.0)
            }
particle  {
            pos=0,0,0  mag= 15 to 25  dir= 0,0,1:10  acc= -0.8
            growth= 0.4  age= 0  color= brown  brightness= 0.2
            to 0.5
            }
leaf      { p= 0.6
            coord= leaf  out= 30  up= 10
            color= green  brightness= 0.4 to 0.8
            tri= 0,0,0 0,-9,3 -3,-3,4  tri= 0,0,0 0,-9,3 3,-3,4
            }
```

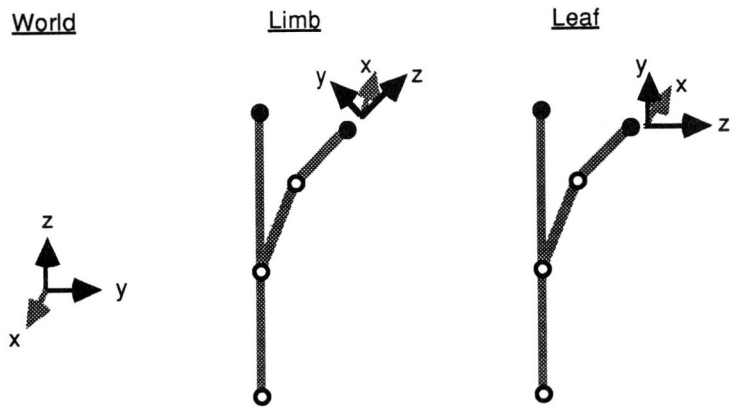


Figure 4. Coordinate Systems

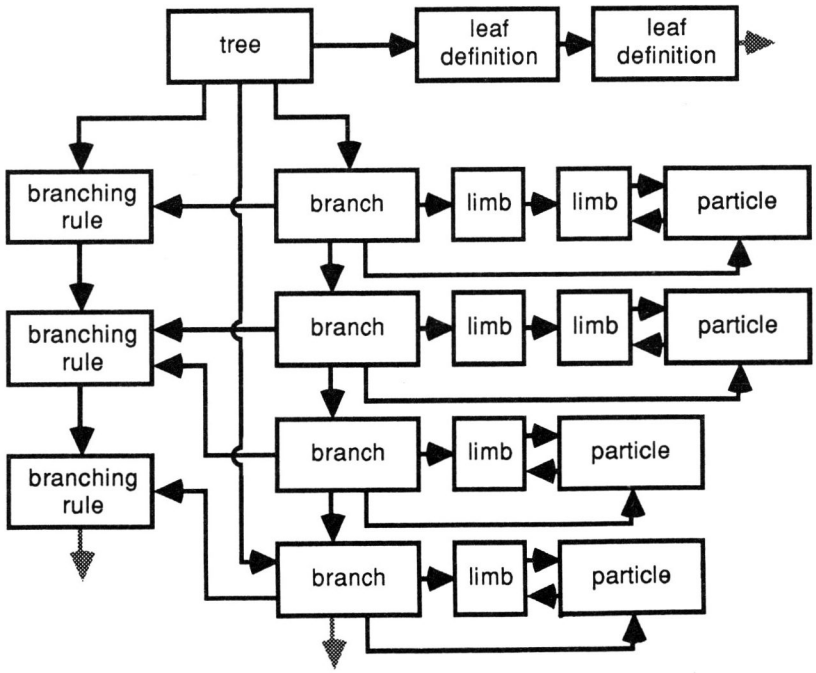


Figure 5. Data Structures

```
leaf    { p= 0.4
          coord= leaf  out= 40  up= 40
          color= white  brightness= 0.9 to 1.0
          tri= 0,0,0 3,1,-1 3,1,1  tri= 0,0,0 -3,1,-1 -3,1,1
          tri= 0,0,0 1,1,3 -1,1,3  tri= 0,0,0 1,1,-3 -1,1,-3
          color= green  brightness= 0.5 to 0.8
          tri= -1,1,-1 1,1,-1 0,1,2
          }
```

Explanations of each section are as follows:

iterations { i= 13 }
Specifies the number of iterations to grow the tree. There will be 13 segments in the trunk.

gravity { g= 0.08 }
Simulates the effect that gravity has on the limbs. This factor is multiplied by the cosine of the angle between the growth vector and the horizontal, then this value is subtracted from the z-component of the growth vector.

light { ambient=0.1 lambert=0.4 height=0.4 radius=0.3 angle=0.2 }
These parameters are the five shading factors used when calculating the color of the tree elements. In this tree, Lambert's law and the relative height and radius of the element in the tree are most influential.

color { green= 0,1,0 }
color { brown= 1,0.9,0.2 }
color { white= 1,1,1 }
These values define the colors used in the tree. Each triplet of numbers represents the weighting to give to red, green, and blue. Only the ratios are important here; magnitudes have no affect, since the shade of the color is determined using the allowable range given in the particle and leaf definitions.

```
branch  { level= 1  number= 2.0 to 3.0
          (p=1  angle= 40 to 80  dir= 0 to 360  factor= 0.5 to
           0.7)
          }
```

This is the branching rule for particles that are spawned off the trunk, the level 1 branches. At each segment of the trunk, there will be 2 or 3 branches. The angle from the trunk will be between 40 and 80 degrees, and these branches will grow from any direction around the tree (0 to 360 degrees). The growth reduction factor for these branches will be between 0.5 and 0.7, which will reduce the width and length of these limbs as compared to the segments in the trunk.

```
branch  { level= 2  number= 0.4 to 1.6
          (p=0.5  angle= 40 to 60  dir= 0 to 20  factor= 0.8 to
           1.0)
          (p=0.5  angle= 40 to 60  dir= 160 to 180  factor=
           0.8 to 1.0)
          }
```

This rule governs the growth of branches off the side of those off the trunk. There will be 0 to 1 branches growing off the parent at each segment, since the random number generated between 0.4 and 1.6 is truncated. This specification is different from "0 to 1" because of the uniformly distributed random variable. The mean of "0 to 1" is 0.5, while the mean of "0.4 to 1.6" is 1, which will produce more branches. There are two possible rules for these branches, each with equal probability of occurrence. The only difference in these rules is the direction angles. The branch can either grow 0 to 20 degrees above the horizontal on one side, or 0 to 20 degrees above the horizontal on the other side (specified as 160 to 180). It will be at an angle between 40 and 60 degrees from the parent limb, and its growth will be a factor of 0.8 to 1.0 less than its parent's.

```
branch  { level= 3  number= 0.4 to 1.6
          (p=0.45  angle= 40 to 60  dir= 0 to 20  factor= 0.8
           to 1.0)
          (p=0.45  angle= 40 to 60  dir= 160 to 180  factor=
           0.8 to 1.0)
          (p=0.10  angle= 40 to 80  dir= 0 to 180  factor= 0.1
           to 0.2)
          }
```

This branching rule will apply for all branches from level 3 to level 6. The difference between this rule and the previous is the addition of a new parameter triplet. This last triplet has a small probability of occurrence, but its purpose is to spawn limbs that will grow in any upward direction. These limbs will be small and short, since their growth is reduced by a large factor. These limbs are useful for increasing the number of leaves on the tree, since a leaf will be added at the end of every branch.

```
branch  { level= 7  number= 0.4 to 1.5
          (p=0.5  angle= 30 to 90  dir= 0 to 50  factor= 0.6 to
           1.0)
          (p=0.5  angle= 30 to 90  dir= 130 to 180  factor=
           0.6 to 1.0)
          }
branch  { level= 9  number= 0.4 to 1.5
          (p=1  angle= 50 to 110  dir= 0 to 180  factor= 1.0
           to 1.0)
          }
```

These branching rules are similar to the previous ones, but with more relaxed constraints. The branches of level 9 through 13 are allowed to grow in any upward direction, without any further reduction in their growth as compared to the parent. Of course, the aging factor is in effect to continually reduce the growth vector, as specified by the parameter, "acc", in the particle definition.

```
particle  {
          pos= 0,0,0  mag= 15 to 25  dir= 0,0,1:10  acc=
           -0.8
          growth= 0.4  age= 0  color= brown  brightness=
           0.2 to 0.5
          }
```

There is only one particle to start this dogwood tree, which is located at the base of the trunk. At each iteration, the growth vector, which has a magnitude varying between 15 and 25, and a direction up varying by 10 degrees, will be added to this position. The acceleration is also added to the magnitude, which will reduce each limit of the range by 0.8 at each iteration. The width of these limbs will be incremented by 0.4 each time, and the aging factor, 0, is added to the growth value. This age parameter can be used to slow the lateral growth of the limb as it gets older. The limbs will be colored brown, with a brightness ranging between 0.2 and 0.5 of the maximum.

```
leaf    { p= 0.6
          coord= leaf  out= 30  up= 10
          color= green  brightness= 0.4 to 0.8
          tri= 0,0,0  0,-9,3  -3,-3,4  tri= 0,0,0  0,-9,3  3,-3,4
          }
```

This leaf definition is for the green leaves of the dogwood tree, which are composed of two triangles. Since the probability is 0.6, there will be approximately 60% leaves and 40% flowers on the tree. The leaf coordinate system is used to define the triangles, since dogwood leaves usually hang down, no matter which way the branch is pointing. The out vector for this leaf, which is the z-axis in this coordinate system, can vary by 30 degrees. The up vector, y-axis, for this leaf can vary by 10 degrees. Of course, the leaves of a flowering dogwood are usually green, here with a brightness ranging from 0.4 to 0.8 of maximum.

```
iterations   { i= number_of_iterations }
gravity      { g= gravity_factor }

color        { color_name = red, green, blue }
light        { ambient=ambient_parameter   lambert=lambert_parameter
               height=height_parameter   radius=radius_parameter
               angle=angle_parameter }

branch       { level = level_number   number= low to high
               (p=probability   angle=low to high
                dir= low to high   factor= low to high )
               }
branch       { level = level_number   number= low to high
               (p=probability   angle= low to high
                dir= low to high   factor= low to high )
               (p=probability   angle= low to high
                dir= low to high   factor= low to high )
               }
particle     {
               pos=x,y,z   mag= low to high   dir= vx,vy,vz:maxdev_angle
               acc=acceleration
               growth= growth_reduction_factor   age= age_factor
               color= color_name   brightness= low to high
               }
leaf         { p= probability
               coord= limb or leaf   out= out_angle   up= up_angle
               color= color_name   brightness= low to high
               ln= x1,y1,z1  x2,y2,z2
               color= color_name   brightness= low to high
               tri= x1,y1,z1  x2,y2,z2  x3,y3,z3
               }
```

Notes:

1. Words in bold are section tags or keywords that must be typed as is.
2. Words and numbers in italics are the parameters of arguments.
3. Any kind of spacing is allowed, as long as the curly braces, parentheses, and equal signs are correct.
4. The branching rules **must** occur before the particle(s).
5. The probabilities in the branching rules and leaf specifications **must** add up to one.
6. The **to** keyword signifies a range from minimum to maximum allowed values.

Figure 6. General Format of the Tree Template

```
leaf         { p = 0.6
               coord= leaf  out= 40  up= 40
               color= white  brightness= 0.9 to 1.0
               tri= 0,0,0  3,1,-1  3,1,1  tri= 0,0,0  -3,1,-1  -3,1,1
               tri= 0,0,0  1,1,3  -1,1,3  tri= 0,0,0  1,1,-3  -1,1,-3
               color= green  brightness= 0.5 to 0.8
               tri= -1,1,-1  1,1,-1  0,1,2
               }
```

This leaf definition specifies the flowers, which are made from 4 white triangles and 1 green triangle in the center. The z-axis can vary by 40 degrees and the y-axis can vary by 40 degrees, which gives the flowers more freedom of orientation than the leaves.

VI. Summary and Future Directions

The flexibility provided by the tree template makes this program suitable for users who are not familiar with programming techniques for modeling natural objects by still allowing them creative control of the image. We feel that TREE-MAKER is suitable for applications such as landscaping, art, and adding surface features to digital terrain data. The simple shading and shadowing methods allow the program to execute in less time than more exact methods, so the user can make changes in position and study the effect of other parameters. Future plans for TREE-MAKER are to include it in a package of tools for modeling natural objects for use by students and researchers at Georgia Tech.

VII. Acknowledgment

The authors wish to thank Mike Williams, a graduate student at Georgia Tech, for porting TREE-MAKER to run on both Sun and Silicon Graphics workstations.

VIII. References

1. Aono, M. and T.L. Kunii, "Botanical Tree Image Generation" in IEEE Computer Graphics and Applications 4, 5 (May 1984) 10-33.

2. Barnsley, M.F., et. al., "Harnessing Chaos for Image Synthesis," Proc. of SIGGRAPH '88 (Atlanta, August 1-5, 1988) in Computer Graphics 22, 3 (August 1988) 131-140.

3. Bloomenthal, J., "Modeling the Mighty Maple," Proc. of SIGGRAPH 85 (San Francisco, July 22-26, 1985) in Computer Graphics 19, 3 (July 1985) 305-311.

4. Demko, S., L. Hodges, B. Naylor, "Construction of Fractal Objects with Iterated Function Systems,," Proc. of SIGGRAPH '85 (San Francisco, July 22-26) in Computer Graphics 19, 3 (July 1985) 271-278.

5. de Reffye, P., et. al., "Plant Models Faithful to Botanical Structure and Development," Proc. of SIGGRAPH '88 (Atlanta, August 1-5, 1988) in Computer Graphics 22, 3 (August 1988) 151-158.

6. Oppenheimer, P., "Real Time Design and Animation of Fractal Plants and Trees," Proc. SIGGRAPH 86 (Dallas, August 18-22, 1985), in Computer Graphics 20, 4 (August 1986) 55-64.

7. Prusinkiewicz, P., et. al., "Developmental Models of Herbaceous Plants for Computer Imagery Purposes," Proc. of SIGGRAPH '88 (Atlanta, August 1-5, 1988) in Computer Graphics 22, 3 (August 1988) 141-148.

8. Reeves, W.T. and R. Blau, "Approximate and Probabilistic Algorithms For Shading and Rendering Structured Particle Systems," Proc. of SIGGRAPH '85 (San Francisco, July 22-26, 1985) in Computer Graphics 19, 3 (July 1985) 313-322.

9. Reeves, B., "Particle Systems — A Technique for Modeling a Class of Fuzzy Objects" in ACM Transactions on Graphics 2 (April 1983) 91-108.

10. Smith, A.R., "Plants, Fractals, and Formal Languages," Proc. of SIGGRAPH '84 (Minneapolis, July 23-27, 1984) in Computer Graphics 18, 3 (July 1984) 1-10.