

The Object Instancing Paradigm for Linear Fractal Modeling

John C. Hart

Electronic Visualization Laboratory
University of Illinois at Chicago

and

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

Abstract

The recurrent iterated function system and the L-system are two powerful linear fractal models. The main drawback of recurrent iterated function systems is a difficulty in modeling whereas the main drawback of L-systems is inefficient geometry specification. Iterative and recursive structures extend the object instancing paradigm, allowing it to model linear fractals. Instancing models render faster and are more intuitive to the computer graphics community.

A preliminary section briefly introduces the object instancing paradigm and illustrates its ability to model linear fractals. Two main sections summarize recurrent iterated function systems and L-systems, and provide methods with examples for converting such models to the object instancing paradigm. Finally, a short epilogue describes a particular use of color in the instancing paradigm and the conclusion outlines directions for further research.

Keywords: Constructive Solid Geometry, L-system, Linear Fractal, Object Instancing, Recurrent Iterated Function System.

1 Introduction

Linear fractals are sets possessing some sort of self-affinity, with detail at all levels of magnification. Many of the examples used to demonstrate the basic concepts of fractal geometry are linear fractals, such as those decorating the first part of [Mandelbrot, 1982]. Other accounts have referred to linear fractals as “self-similar/self-affine sets” [Mandelbrot, 1982], “recurrent sets” [Dekking, 1982] and “graftals” [Smith, 1984].

The two most common linear fractal models are the recurrent iterated function system (RIFS) and the Lindenmeyer parallel graph grammar (L-system).

The iterated function system (IFS) model was used to investigate the use of linear fractals in image synthesis in [Demko *et al.*, 1985; Barnsley *et al.*, 1988]. Their rendering method used, often quite sophisticated, point clouds. An extension to the IFS model has been introduced

and examined in various forms, under the adjectives “recurrent” [Dekking, 1982; Barnsley *et al.*, 1989; Cabrelli *et al.*, 1991], “Markov” [Womack, 1989], “controlled” [Prusinkiewicz & Lindenmayer, 1990], “language restricted” [Prusinkiewicz & Hammel, 1991] and “hierarchical” [Peitgen *et al.*, 1991]. Subtle differences in nomenclature and form have caused this heterogeneous development. Moreover, modeling objects using the RIFS model is difficult, and each of the above variations on the RIFS theme have distinct advantages for this task.

L-systems are parallel-graph grammars. Many have explored their applications in natural image synthesis, e.g. [Smith, 1984; Prusinkiewicz *et al.*, 1988; Prusinkiewicz & Lindenmayer, 1990], usually via the turtle graphics paradigm [Abelson & diSessa, 1982]. The L-system model provides a straightforward and powerful technique for modeling natural objects, though the geometry it produces using the turtle paradigm is not organized efficiently for rendering.

Both models, in many cases, can be simulated using the object instancing paradigm, which makes them more efficient for rendering. We begin with a review of the object instancing paradigm. Descriptions of the recurrent iterated function system and the L-system models follow, and each is completed with instructions for the translation of the model to the object instancing paradigm.

2 The Object Instancing Paradigm

Object instancing is a modeling technique that permits efficient internal representations of redundant objects. Often, complicated objects consist of many identical components. Object instancing capitalizes on the redundancy of these identical components. A good example of an ideal situation is seen in the animation “Megacycles” [Amanatides & Mitchell, 1989].

The object instancing paradigm consists of two basic kinds of objects: primitives and instances.

A *primitive* is an indivisible rendering atom. Boundary-representation systems use polygons or spline surfaces as primitives whereas solid modeling systems use natural quadrics, planes, tori and superquadrics as primitives.

An instance consists of a pointer to another object (a primitive or another instance, called the “master” object) and an affine transformation. The instance reproduces the referenced object, deformed by its affine transformation. [Sutherland, 1963].

An object may also be a CSG set-theoretic operation such as a union, intersection or difference. Their operands are pointers to other operations, and so, they too instance other objects. (For the purpose of this discussion, we will focus on the union operation.)

The object instancing paradigm contains another distinction: the global coordinate system where the final scene is assembled versus the local modeling coordinate systems where the canonical scene components are constructed. A canonical object is a shape in its default size, proportions and location. Canonical objects do not appear in the rendered scene. A final instancing operation is needed to transform the canonical objects from their local modeling-space coordinate systems into the global rendering-space coordinate system [Roth, 1982].

Primitives are canonical objects. The canonical sphere is the unit sphere centered at the origin; the canonical plane is the $y = 0$ plane intersecting the origin; the canonical cylinder is the cylinder of unit radius and height centered and oriented along the interval $[0, 1]$ of the y -axis; the canonical

cone is likewise defined, pointing in the positive y direction. One can render a canonical object in its default state by instantiating it from canonical space into rendering space using the identity transformation matrix. When modeling, instancing takes canonical objects to canonical objects. Upon completion of the modeling process, an instance operation converts the final canonical object into a rendered object for image synthesis.

2.1 Examples

Figure 1 (left) shows an approximation to Sierpinski's gasket by nine triangles. Figure 1 (right) shows a possible CSG instancing model with a tree topology. The circles denote CSG union operations of three instances.

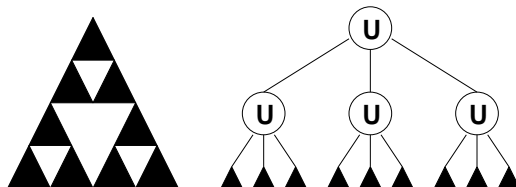


Figure 1: An approximation of Sierpinski's gasket (left) modeled with a tree topology (right).

In the absolute specification paradigm, each triangle is individually specified, commonly by its three endpoints. Further development of this tree-structured model of Sierpinski's gasket requires the specification of 3^{n-1} individual triangles, where n is the height of the tree. Hence, the size of an absolute specification model of a linear fractal grows exponentially as the detail increases.

The subtrees of Figure 1 (right) are redundant. Using the instancing paradigm, one creates the same gasket approximation, Figure 2 (left), using a graph topology, Figure 2 (right), with fewer nodes. Here, the small circles denote instancing operations.

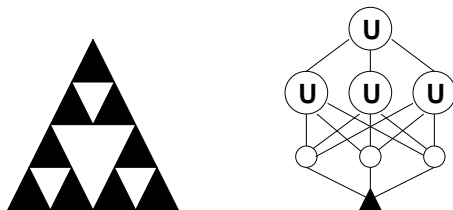


Figure 2: An approximation of Sierpinski's gasket (left) modeled with a graph topology (right).

The textual specification for such an instanced model might look like the following.

```
canonical object level_1_up
  triangle
  scale: 0.5, 0.5
  translate: 0.0, 0.5
```

```

}

canonical object level_1_left {
    triangle
    scale: 0.5, 0.5
    translate: -0.433, -0.25
}

canonical object level_1_right {
    triangle
    scale: 0.5, 0.5
    translate: 0.433, -0.25
}

canonical object level_2_up {
    union: level_1_up, level_1_left, level_1_right
    scale: 0.5, 0.5
    translate: 0.0, 0.5
}

canonical object level_2_left {
    union: level_1_up, level_1_left, level_1_right
    scale: 0.5, 0.5
    translate: -0.433, -0.25
}

canonical object level_2_right {
    union: level_1_up, level_1_left, level_1_right
    scale: 0.5, 0.5
    translate: 0.433, -0.25
}

object level_top {
    union: level_2_up, level_2_left, level_2_right
}

```

In the previous object instancing example there is no difference between a level two object such as `level_2_up` and a level one object such as `level_1_up` other than their name and the objects they instance. This is called iterative instancing and the size of its model grows linearly as the level of detail increases.

Instancing graphs are usually acyclic. A directed cycle in an instancing graph means an object has somehow instanced itself, and if rendered, may appear infinitely many times in the scene. This is called recursive instancing.

As shown in [Hart & DeFanti, 1991; Hart, 1991a], one can extend the object instancing paradigm to handle recursive instancing by culling the instancing process when a predefined global bounding volume become sufficiently small. This requires any instancing loop to be contractive, otherwise the instances would not converge.

For example, consider the limit of the shape described in Figures 1 and 2. Allowing recursive instancing, the hierarchy becomes cyclic and describes a linear fractal, specifically Sierpinski's gasket, in Figure 3.

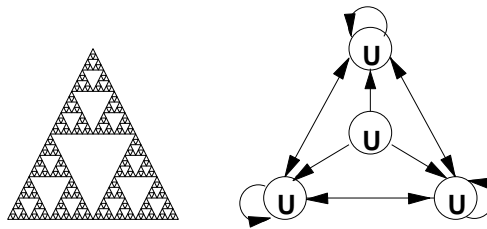


Figure 3: Sierpinski's gasket (left) modeled with a cyclic graph topology (right).

This linear fractal is specified, more tersely than before, with the following commands.

```
canonical object up {
    union: up, left, right
    scale: 0.5, 0.5
    translate: 0.0, 0.5
}

canonical object left {
    union: up, left, right
    scale: 0.5, 0.5
    translate: -0.433, -0.25
}

canonical object right {
    union: up, left, right
    scale: 0.5, 0.5
    translate: 0.433, -0.25
}

object top {
    union: up, left, right
}
```

A cyclic instancing graph represents infinitely high levels of detail. Hence, the size of a recursive instancing model remains constant as the level of detail increases.

3 The RIFS Model

A (hyperbolic) recurrent iterated function system $(\{w_i\}_{i=1}^N, G)$ consists of a finite set of N affine contractions w_i , and an N -vertex weakly-connected digraph G . Each vertex of this digraph corresponds to a contraction and each edge, an allowable contraction composition. Here, a digraph G is denoted by an ordered pair (G_v, G_e) . The vertex set G_v is a set integers $\{1 \dots N\}$. The edge set G_e is a set of ordered pairs $(i, j), i, j \in G_v$. such that the ordered pair $(i, j) \in G_e$ represents a directed edge from vertex i to vertex j .

3.1 The RIFS Attractor

Of fundamental importance to the study of iterated function systems as well as recurrent iterated function systems is the existence of corresponding unique compact non-empty invariant limit sets called attractors. One may recall the definition of the attractor of an IFS $\{w_i\}$ is the unique solution to the recurrence equation

$$A = \bigcup_{i=1 \dots N} w_i(A), \quad (1)$$

as originally shown in [Hutchinson, 1981]. The attractor of an RIFS $(\{w_i\}, G)$ is the union of the solution sets A_i to the recurrence equation

$$A_j = \bigcup_{(i,j) \in G_e} w_j(A_i) \quad (2)$$

as shown in [Barnsley *et al.*, 1989].

The attractor of an RIFS $(\{w_i\}_{i=1}^N, G)$ is a subset of the attractor of the IFS $\{w_i\}_{i=1}^N$. Thus, an RIFS is a restriction of an IFS, which results in the elimination of certain sections of attractor of the IFS. One example of this is the following set of four contractions

$$w_i(x, y) = \left(\frac{x \pm 1}{2}, \frac{y \pm 1}{2} \right) \quad (3)$$

taken over all combinations of signs. If G is a complete digraph of four vertices then the attractor of the RIFS $(\{w_i\}, G)$ is the attractor of the IFS $\{w_i\}$, namely the square $[-1, 1] \times [-1, 1]$. However, if G contains all edges except those of the type (i, i) , then the same map may not be applied twice in a row. The attractor of this RIFS, from [Cabrelli *et al.*, 1991], is the fractal pound sign shown in Figure 4.

3.2 Modeling

One models an object as a linear fractal by creating a RIFS whose attractor approximates the object within some degree of accuracy. Equation (1) suggests that approximating an object with an IFS is as simple as finding a set of contractions that take the entire object to each of its components --- the so-called “collage theorem” philosophy of modeling [Barnsley *et al.*, 1986].

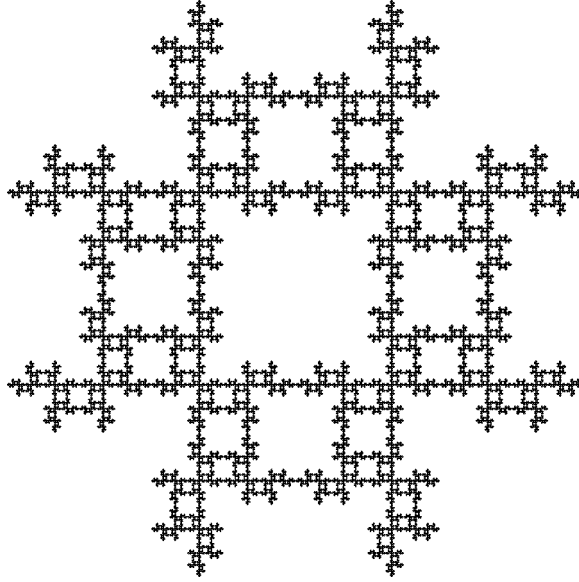


Figure 4: The fractal pound sign.

Equation (2) suggests a similar property, though for an RIFS model, the object is simulated by finding contractions that take parts of the object to smaller parts. In the previous example, the fractal pound sign consists of four images of three-fourths of the original pound sign. One can see the self-similarity of the fractal pound sign by comparing its first quadrant with the rest of the set.

Ordinarily, this partial self-similarity is quite difficult to see in objects. The collage theorem philosophy of modeling requires a self-tiling, which can be easily visualized. Its recurrent form, from [Barnsley *et al.*, 1989], requires a partial self-tiling which is significantly more difficult and, at best, non-intuitive. Hence, the recurrent collage theorem is a somewhat ineffective modeling tool for linear fractals, for which there are, unfortunately, few alternatives.

3.3 Specification via Instancing

The recursive object instancing paradigm is one alternative modeling method to the collage theorem. The drawback is one must still partially self-tiling an object to create a linear fractal model of it. The benefit is the paradigm incorporates tools familiar to the computer graphics community.

Converting an RIFS to an instancing structure requires three simple steps. First N canonical instancing objects are constructed. Each affine contraction of the RIFS becomes the affine transformation matrix of its corresponding instancing object. The control digraph of the RIFS dictates the masters of each instance. Finally, a rendered object instances the canonical instances along with an initial bounding volume into rendering space.

For example, the 3-D pound sign can be specified by the following commands. The canonical object's names are abbreviated versions of the instance's position, from lower-left-rear to upper-right-front.

```

canonical object llr {
  union: llf, lrr, lrf, ulr, ulf, urr, urf
  scale: 0.5, 0.5, 0.5
  translate: -0.5, -0.5, -0.5
}

canonical object llf {
  union: llr, lrr, lrf, ulr, ulf, urr, urf
  scale: 0.5, 0.5, 0.5
  translate: -0.5, -0.5, 0.5
}

.
.
.
canonical object urf {
  union: llr, llf, lrr, lrf, ulr, ulf, urr
  scale: 0.5, 0.5, 0.5
  translate: 0.5, 0.5, 0.5
}

object pound_sign {
  union: llr, llf, lrr, lrf, ulr, ulf, urr, urf
}

```

The object this instancing text models is shown in Figure 5.

4 The L-System Model

Whereas the RIFS resembles the finite state automaton, an L-system is a parallel graph grammar similar in design to a context-free grammar (CFG). It consists of an alphabet, a set of symbols, an initial axiom and a set of productions. CFG productions are applied one at a time and one per step, usually to the leftmost symbol whereas L-system productions are applied in parallel --- all symbols are replaced simultaneously at each step.

4.1 Turtle Geometry

The words of an L-system are commonly interpreted, using turtle geometry [Abelson & diSessa, 1982], as a graphical object. The alphabet for turtle control contains such symbols as F which draws a straight line, and $+$ and $-$ which turn the turtle left and right by some fixed angle. For example, if this fixed angle is 90° , then the word

$$F + F - F - F + F - F - F + F - F - F + F - F \quad (4)$$

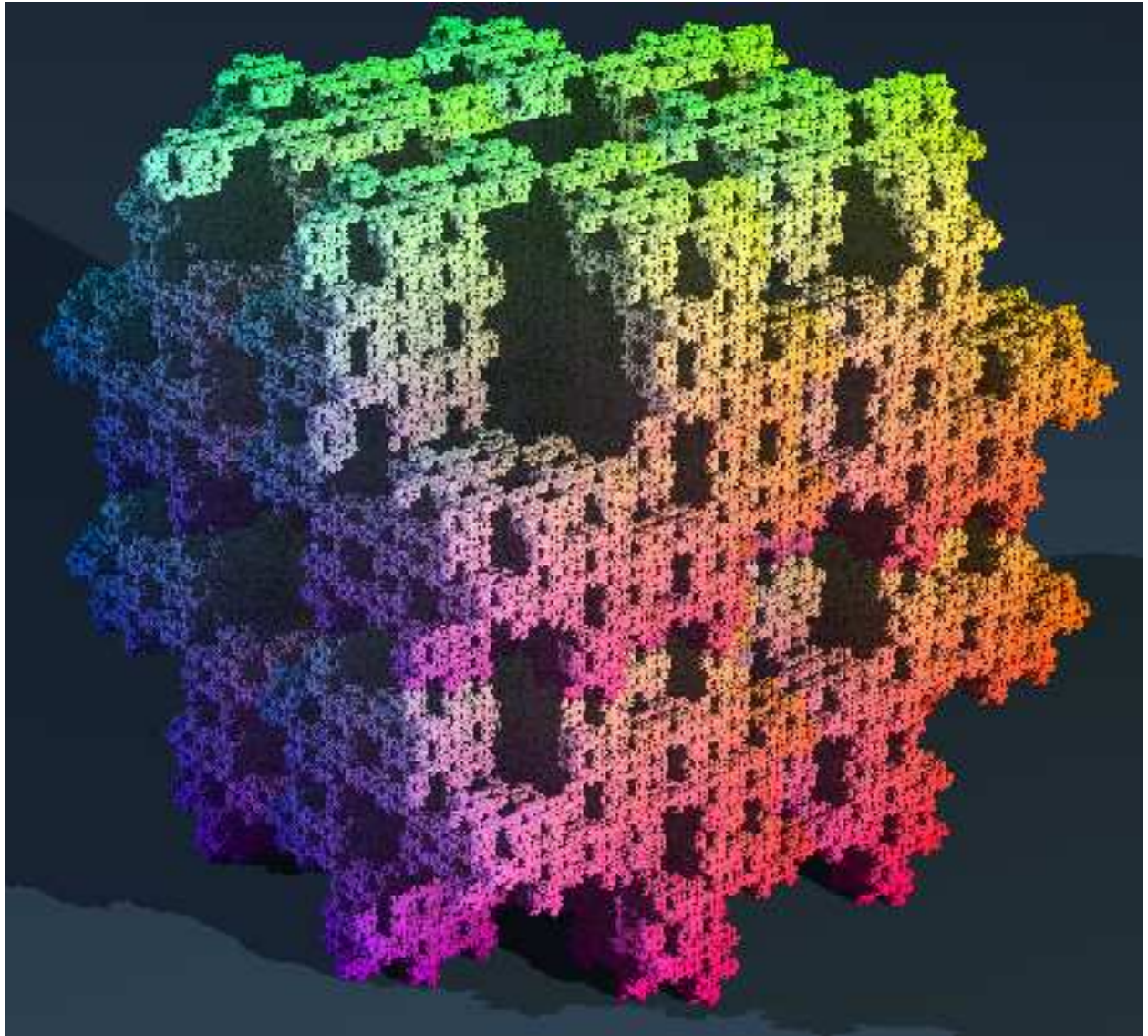


Figure 5: The 3-D fractal pound sign.

will draw the outline of a plus sign. In [Prusinkiewicz & Lindenmayer, 1990], a complete parametric language is used to develop a sophisticated 3-D turtle graphics paradigm.

Of particular utility are the symbols [and]. The left bracket pushes the current state of the turtle on a stack whereas the right bracket pops this stack, restoring the turtle to its previous state. This simulates branching structures, allowing the turtle to concentrate on an intricate branch without having to retrace its steps to get back to the base of the branch to draw the rest of the object.

For example, the following parametric L-system generates a ternary-branching tree, from Figure 2.8(a) in [Prusinkiewicz & Hammel, 1991],

$$\begin{aligned} A \rightarrow & \text{!}(\frac{\sqrt{3}}{3}) F(50) [\&(18.95) F(50) A] \\ & / (94.74) [\&(18.95) F(50) A] \\ & / (132.63) [\&(18.95) F(50) A] \end{aligned} \quad (5)$$

$$F(l) \rightarrow F(1.109l) \quad (6)$$

$$\text{!}(w) \rightarrow \text{!}(\sqrt{3}w) \quad (7)$$

where $\text{!}(\cdot)$ alters the width of lines (radius of cylinders) by the given factor and $F(\cdot)$ draws a line of the given length. The rotation symbol $\&(\cdot)$ alters the pitch of the turtle; the symbol $/(\cdot)$ alters the roll. This L-system begins with the word $\text{!}(1) F(50) A$.

In the previous example, the shortest length of any branch is 50. We can convert this L-system from “enlongating nodes” to “decreasing apices,” which makes the longest length of any branch 50. This produces the following L-system, a recurrent string, which is equivalent to the previous one except for scale,

$$\begin{aligned} A(w, l) \rightarrow & \text{!}(w) F(l) [\&(18.95) F(l) A(\frac{w}{\sqrt{3}}, \frac{l}{1.109})] \\ & / (94.74) [\&(18.95) F(l) A(\frac{w}{\sqrt{3}}, \frac{l}{1.109})] \\ & / (132.63) [\&(18.95) F(l) A(\frac{w}{\sqrt{3}}, \frac{l}{1.109})]. \end{aligned} \quad (8)$$

This L-system begins with the initial word $\text{!}(1) F(50) A(0.577, 45.1)$.

4.2 Specification via Instanting

Conversion from an L-system to an object instancing structure is more difficult than converting to an RIFS. In [Prusinkiewicz & Lindenmayer, 1990], steps were described for converting an L-system model to an RIFS model. Hence, using this result with the last section, we could convert an L-system indirectly to an instancing structure. Instead, we will outline techniques for directly converting L-system models to object instancing models.

In general, converting L-system productions to instances requires thorough knowledge of the current state of the turtle at each step in the production. With the exception of the left bracket, each of the symbols on the right-hand side of the productions affect the state of the turtle. Furthermore,

the line-drawing symbols cause a geometric addition to the scene as well as a change to the state of the turtle.

Each line-drawing symbol, e.g. F , can be interpreted as an instance of some canonical 3-D line, such as a cylinder. The cumulative effect of the symbols preceding the line-drawing symbol determine that line's size, position and orientation. Any bracket delimited symbols can be ignored. The affine transformation matrix of the line-instance is the product of the transformation matrices associated with the symbols preceding the current line-drawing symbol. For example, a $+$ corresponds to a rotation and an F (preceding the current symbols) corresponds to a translation.

Symbols denoting other productions are particularly difficult. These symbols can be replaced by an equivalent sequence of symbols denoting the production's cumulative affect on the turtle's state. Assessment of this cumulative affect can be quite cumbersome, particularly when the production is recursive.

Fortunately, in most botanical models, recursion is used to simulate branching patterns. Rather than having the turtle retracing its steps back to the branch root after drawing the branch, the branch production is bracketed. Hence, if a bracketed branch production symbol precedes the current line-drawing symbol, it can be ignored and its cumulative affect on the turtle's state need not be computed.

The following instancing structure approximates the L-system (8).

```
canonical object trunk {
  cylinder
}

canonical object A {
  union: trunk, branch_1, branch_2, branch_3
  scale: 0.902, 0.902, 0.902
  translate: 0, 1, 0
}

canonical object branch_1 {
  union: trunk, A
  rotate: x, 18.95
  translate: 0, 1, 0
}

canonical object branch_2 {
  union: trunk, A
  rotate: x, 18.95
  rotate: y, 94.74
  translate: 0, 1, 0
}
```

```
canonical object branch_3 {
  union: trunk, A
  rotate: x, 18.95
  rotate: y, 94.74
  rotate: y, 132.63
  translate: 0, 1, 0
}
```

```
object tree {
  union: trunk, A
}
```

This approximation of the L-system suffers from two major shortcomings.

First, the branch width changing factor is not represented in this model; the scaling transformation in the object named *A* is uniform. One might be tempted to use the scaling command: “scale: 0.577, 0.902, 0.577,” which would produce a properly proportioned branch segment, but the rest of its sub-branches would be artificially skewed along the original branch’s major axis.

Second, this L-system will be evaluated to the pixel level, producing very fine branches but no leaves. A more realistic model would terminate well before the limit structure, allowing the addition of leaves to the ends of the branches.

Both of these problems are solved by duplicating the above instancing text several times, into levels. This is equivalent to the use of conditions in a parametric L-system. The resulting instancing structure might look like the following, in a parametric instancing scheme.

```
canonical object A(0) {
  union: leaf
  translate: 0, 1, 0
}
```

```
canonical object A(n) {
  union: trunk(11-n), branch_1(n),
        branch_2(n), branch_3(n)
  scale: 0.902, 0.902, 0.902
  translate: 0, 1, 0
}
```

```
canonical object branch_1(n) {
  union: trunk(11-n), A(n-1)
  rotate: x, 18.95
  translate: 0, 1, 0
}
```

```

canonical object branch_2(n) {
  union: trunk(11-n), A(n-1)
  rotate: x, 18.95
  rotate: y, 94.74
  translate: 0, 1, 0
}

canonical object branch_3(n) {
  union: trunk(11-n), A(n-1)
  rotate: x, 18.95
  rotate: y, 94.74
  rotate: y, 132.63
  translate: 0, 1, 0
}

canonical object trunk(0) {
  instance: cylinder
}

canonical object trunk(n) {
  instance: trunk(n-1)
  scale: 0.64, 1, 0.64
}

object tree {
  union: trunk(0), A(10)
}

```

In this case, the factor 0.64 unscales the radius of each branch by the value 0.902, then rescales it by the proper value 0.577 without affecting the geometry of its sub-branches.

The above parametric instancing specification can be implemented for standard instancing interpreters by enumerating each object for all possible values of n , incorporating the value n into the name of the object. For example, instead of `trunk(8)` instancing `trunk(7)`, the object `trunk_level_8` would instance `trunk_level_7`. The latter is significantly more verbose, but is still much less than absolute specification of every branch. In fact, parametric instancing specifies in constant steps what standard instancing specifies in logarithmic steps --- what absolute specification specifies in linear steps.

A similar L-system was used to model the tree in Figure 6 (compare the trees in [Kay & Kajiya, 1986]). The branches are iteratively instanced cylinders, but the leaves are modeled by a recursive instancing structure derived from the IFS leaf model shown in [Demko *et al.*, 1985]. The grass is modeled as iteratively instanced as in [Snyder & Barr, 1987], though the grass blades here are cones.



Figure 6: Autumn --- an L-system tree with IFS leaves on instanced grass.

5 Incorporating Color

Color is incorporated into the linear fractal instancing model by associating a color and a weight with each instance. The color is specified by an RGB triple, the weight, by an alpha value. A current color and alpha value are maintained for the object during CSG hierarchy traversal. During rendering, when an operand is instanced, the instance tints the color of its parent in the hierarchy by its own individual color.

Formally, this tinting is based on the transfer functions from image compositing [Porter & Duff, 1984],

$$\mathbf{c} = \mathbf{c}_o + \alpha_i(1 - \alpha_o)\mathbf{c}_i, \quad (9)$$

$$\alpha = \alpha_o + \alpha_i(1 - \alpha_o), \quad (10)$$

where \mathbf{c} , α are the resulting color and alpha values, \mathbf{c}_p , α_p are those of the parent, and \mathbf{c}_i , α_i are the tinting values of the instance.

The eight instances used to generate Figure 5 had the eight associated colors: black, red, green, yellow, blue, magenta, cyan and white, each with an alpha value of one-half. This coloring scheme was also used in [Hart & Das, 1991; Hart, 1991b] to better clarify otherwise imperceptible features of highly intricate linear fractal surfaces.

The coloring of the elm trees in the Fractal Forest [Hart & DeFanti, 1991; Hart, 1991a] was carefully adjusted so that the trunk and most of the visible branches were brown but the leaves (actually tiny branches) were green. Assigning the trunk a brown tint with a high alpha value and the branches a green color with a low alpha value achieved this goal. Thus, once a trunk instance was used, all of its decedents would be mostly brown whereas green points in the set arose from almost exclusive application of branch instances.

6 Conclusion

With a few enhancements, the standard instancing paradigm can be adapted to model linear fractals. This paper discussed methods for converting two popular linear fractal models to the instancing paradigm, for efficient implementation in current computer graphics systems. Several examples were given, with modeling code and resulting images. Additionally, the use of color in linear fractal models was documented.

6.1 Further Research

As they become better understood, linear fractals are becoming more popular for their applications in natural modeling and image synthesis. A few problems still remain regarding their rendering, such as a rigorous treatment of the shading properties of a fractal surface [Hart, 1991a].

Currently, the most popular open problem regarding the modeling of linear fractals is the algorithmic solution to the inverse problem: given a shape, automatically determine the parameters of a linear fractal model that simulates the shape within a prescribed accuracy. One solution uses a

gradient search in the RIFS parameter space [Vrscay & Roehrig, 1989], but this method is prone to local minima traps. Further enhancement using genetic programming techniques appear promising [Vrscay, 1991], but much work remains along this specific course of attack.

One of the immediate applications of an automated solution to the inverse problem is image compression. Toward this end, block coding methods have been developed to solve this particular application of the inverse problem [Jacquin, 1991]. Nonetheless, image compression is only one tiny application of what could be a very powerful result. The ability to automatically model intricate 3-D structures with linear fractals would be a major result in computer graphics --- its research should not end with one simple solution geared toward an image compression application.

Finally, the classic object instancing paradigm is not fully suited to modeling many of the subtle enhancements to the turtle-geometric L-system model, such as tropism [Prusinkiewicz & Lindenmayer, 1990] or stochastic variations [Kajiya, 1983; Bouville, 1985]. More research on extensions to the object instancing paradigm would allow more efficient modeling and rendering of more realistic botanical structures.

6.2 Acknowledgements

Tom DeFanti and Larry Smarr arranged the funding of this research, through a postdoctoral grant from the National Science Foundation.

The author performed this research on an AT&T Pixel Machine 964dX and a 940dX and is grateful to AT&T for their equipment donations.

Finally, the author would like to thank John Amanatides, Don Mitchell, Daryl Hepting, Arnaud Jacquin, Przemyslaw Prusinkiewicz and Ed Vrscay for their communication.

References

- Abelson, H. and diSessa, A. A. *Turtle Geometry*. MIT Press, 1982.
- Amanatides, J. and Mitchell, D. P. Megacycles. *SIGGRAPH Video Review*, 51, 1989. (Animation).
- Barnsley, M. F., Ervin, V., Hardin, D., and Lancaster, J. Solution of an inverse problem for fractals and other sets. *Proceedings of the National Academy of Science*, 83:1975--1977, April 1986.
- Barnsley, M. F., Jacquin, A., Mallassenet, F., Rueter, L., and Sloan, A. D. Harnessing chaos for image synthesis. *Computer Graphics*, 22(4):131--140, 1988.
- Barnsley, M. F., Elton, J. H., and Hardin, D. P. Recurrent iterated function systems. *Constructive Approximation*, 5:3--31, 1989.
- Bouville, C. Bounding ellipsoids for ray-fractal intersection. *Computer Graphics*, 19(3):45--51, 1985.

- Cabrelli, C., Molter, U., and Vrscaj, E. R. Recurrent iterated function systems: Invariant measures, a collage theorem and moment relations. In *Proceedings of the First IFIP Conference on Fractals*. Elsevier, 1991.
- Dekking, F. M. Recurrent sets. *Advances in Mathematics*, 44:78--104, 1982.
- Demko, S., Hodges, L., and Naylor, B. Construction of fractal objects with iterated function systems. *Computer Graphics*, 19(3):271--278, 1985.
- Hart, J. C. and Das, S. Sierpinski blows his gasket. *SIGGRAPH Video Review*, 61, 1991. (Animation).
- Hart, J. C. and DeFanti, T. A. Efficient antialiased rendering of 3-D linear fractals. *Computer Graphics*, 25(3), 1991.
- Hart, J. C. *Computer Display of Linear Fractal Surfaces*. PhD thesis, EECS Dept., University of Illinois at Chicago, Sept. 1991.
- Hart, J. C. unNatural Phenomena. *SIGGRAPH Video Review*, 71, 1991. (Animation).
- Hutchinson, J. Fractals and self-similarity. *Indiana University Mathematics Journal*, 30(5):713--747, 1981.
- Jacquin, A. E. Image coding based on a fractal theory of iterated contractive image transformations. In Hart, J. C. and Musgrave, F. K., editors, *Fractal Models in 3-D Computer Graphics and Imaging*, pages 245--270. ACM SIGGRAPH '91 (Course #14 Notes), 1991.
- Kajiya, J. T. New techniques for ray tracing procedurally defined objects. *ACM Transactions on Graphics*, 2(3):161--181, 1983. Also appeared in *Computer Graphics* 17, 3 (1983), 91--102.
- Kay, T. L. and Kajiya, J. T. Ray tracing complex scenes. *Computer Graphics*, 20(4):269--278, 1986.
- Mandelbrot, B. B. *The Fractal Geometry of Nature*. W.H. Freeman, San Francisco, 2nd edition, 1982.
- Peitgen, H.-O., Jurgens, H., and Saupe, D. *Fractals for the Classroom*. Springer-Verlag, New York, 1991.
- Porter, T. and Duff, T. Compositing digital images. *Computer Graphics*, 18(3):253--259, 1984.
- Prusinkiewicz, P. and Hammel, M. Automata, languages and iterated function systems. In Hart, J. C. and Musgrave, F. K., editors, *Fractal Models in 3-D Computer Graphics and Imaging*, pages 115--143. ACM SIGGRAPH '91 (Course #14 Notes), 1991.
- Prusinkiewicz, P. and Lindenmayer, A. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.

- Prusinkiewicz, P., Lindenmayer, A., and Hanan, J. Developmental models of herbaceous plants for computer imagery purposes. *Computer Graphics*, 22(4):141--150, August 1988.
- Roth, S. D. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18(2):109--144, February 1982.
- Smith, A. R. Plants, fractals, and formal languages. *Computer Graphics*, 18(3):1--10, July 1984.
- Snyder, J. M. and Barr, A. H. Ray tracing complex models containing surface tessellations. *Computer Graphics*, 21(4):119--128, 1987.
- Sutherland, I. E. Sketchpad: A man-machine graphical communication system. *Proceedings of the Spring Joint Computer Conference*, 1963.
- Vrscay, E. R. and Roehrig, C. J. Iterated function systems and the inverse problem of fractal construction using moments. In Kaltofen, E. and Watt, S. M., editors, *Computers and Mathematics*, pages 250--259, New York, 1989. Springer-Verlag.
- Vrscay, E. R. Iterated function systems: Theory, applications and the inverse problem. In *Lectures of the NATO Advanced Study Institute on Fractal Geometry and Analysis*, Montreal, 1991. Kluwer.
- Womack, T. E. Linear and markov iterated function systems in fractal geometry. Master's thesis, Virginia Polytechnic Institute, May 1989.