Modeling Plants with Environment-Sensitive Automata

James Arvo and David Kirk*, Apollo Computer, Inc.,
330 Billerica, Chelmsford, MA 01824, USA.

*current address: California Institute of Technology,
Computer Science 256-80, Pasadena, CA 91125, USA

Abstract

We present a new approach for modeling plants which allows for adaptation to geometrical and material properties of a simulated environment. The form of a plant is derived from the paths of one or more automata which begin as seeds and progressively explore the environment. The automata sense their surroundings by casting rays and using intersection information to influence subsequent patterns of growth. In this way the power of well-known ray tracing techniques is applied to a problem of geometric modeling. By seeking locations which satisfy various environmental criteria such as proximity to surfaces and availability of light, the automata extend the repertoire of simulated plant growth to include emulation of phenomena such as clinging vines and heliotropism. This approach to database amplification can be used in conjunction with previous methods such as particle systems and graftals.

Introduction

Many methods have been devised for modeling botanical forms for computer graphics. Lindenmayer [10] introduced rewriting grammars to model simple forms of biological growth, and later extended this to bracketed L-systems which could model branching trees. Aono and Kunii [1] improved upon the L-system models through additional parametric control of branching patterns and by taking factors such as wind, sunlight, and gravity into account.

Smith [15] introduced a deterministic grammar-based modeling approach called "graftals" which could model plants and trees. Graftals are fractal-like in their ability to create forms of great complexity from simple instructions, yet are more versatile than fractals which are strictly self-similar.

A very different approach, introduced by Reeves [11], was used to simulate fire, grass, and fireworks by rendering the trajectories of stochastically controlled particle systems. This was later extended to model and render trees [12]. In this approach large numbers of particles are given independent "lifetimes" in which to contribute to the environment.

Generation and Interpretation

Most botanical modeling approaches begin by generating a relatively simple scaffolding, or *skeleton*, on which to build the more detailed geometry of the final plant or tree. Adopting Smith's terminology [15], we refer to the process of constructing the skeleton as *generation* and the subsequent modeling as *interpretation*. In biological terms the skeleton and resulting plant are *genotype* and

[1]In *Proceedings of Ausgraph '88*, pages 27-33, Melbourne, Australia, July 1988. Recreated from original manuscript, January, 1996.

phenotype respectively.

While Smith focused almost exclusively on the grammar-based mechanism for generation, others have attacked specific problems of interpretation. For example, Bloomenthal [3] addressed the problem of modeling a particular type of maple tree. His main emphasis was on fine geometric features of the model, such as bark and leaves, as opposed to problems of generation such as branching patterns or other parameters of growth.

Environment-Sensitive Automata

The generation process which we present in this paper can be viewed in terms of particle systems since the skeleton is simply a record of particle trajectories. We shall refer to these particles, along with their governing rules, as *automata* in order to distinguish a broader range of capabilities. In particular, we shall focus on automata endowed with elaborate behaviors which are neither purely random, as in [11], nor completely deterministic, as in [15]. We accomplish this by allowing the automata to sense and adapt to their surroundings. This important ability defines a class which we call *environment-sensitive automata*, or *ESA*.

Most approaches which have included context-sensitivity have done so only with respect to previous "generations" of the model in isolation. Though Aono and Kunii [1] have included coarse environmental factors such as wind and sunlight, none have employed detailed awareness of the environment geometry during the growth process. ESA possess such an awareness, and consequently exhibit behaviors which were previously difficult or impossible to simulate.

Each automaton is an independent probabilistic "machine" which produces a simple or complex network of paths through the environment to serve as a skeleton for interpretation as a plant. Particle systems, graftals, and ESA are all similar in that their primary role is carried out during the generation phase, leaving open many possible interpretations.

The Connection with Ray Tracing

The central distinguishing feature of ESA is their ability to sense their surroundings. With the exception of resource functions, which we discuss later, all contact with the environment is through ray-environment intersection testing. In this context ray tracing provides the key to "global" geometric modeling. This is a parallel to its original role in global illumination [18].

Some geometric information cannot be obtained through ray tracing. This is similar to the problem of picture reconstruction from point samples. However, this mechanism is well suited to stochastic navigation of "particles" and allows the ESA to thrive in any environment which can be ray traced. It is this connection with ray tracing which extends the repertoire of ESA beyond previous techniques for modeling plants.

The ray-environment intersection testing is performed exclusively through a single abstraction which represents a capability present in almost any ray tracer. Though ray tracing has a reputation for slow execution, many techniques have been proposed in recent years for optimizing this particular operation. These include bounding volume hierarchies [14, 17, 8], space subdivision [5, 7], plane sets [9], and ray classification [2]. Virtually any such technique is applicable to ESA if the generation and interpretation phases are performed independently. As we shall see, the situation becomes more complex when these phases are interleaved.

Environmental Criteria

The list of environmental features which can be influenced by ESA is virtually limitless. Two of the most important geometric features are interference and proximity. An automaton at point **P** may choose not to move to a randomly chosen point **Q** is that point is not in **P**'s line of sight, or if it is too distant from or too close to other surfaces in the environment. The former criterion is simply a linear interference test while the latter is an example of a property which typically must be determined probabilistically. The proximity criterion is reminiscent of John Conway's game of "Life" [6], an example of cellular automata in which "organisms" die of isolation if there are too few neighbors, or of overcrowding if there are too many. In this case, however, "neighbors" are other organisms instead of static geometry.

By rejecting moves which would penetrate or stray too far from a surface, we can generate a skeleton which roughly conforms to arbitrary surface contours. This can be used to model clinging vines. Of course there are a number of other factors which must also be taken into account before we can convincingly simulate ivy-covered surfaces.

Additional features of the environment can be accessed through *resource functions* which are defined on surfaces or throughout space. These functions provide a convenient means of encoding the availability of natural or abstract resources which affect growth in some way. For instance, the ESA may use information obtained as part of the ray-environment intersection operation to determine availability of water or richness of soil at a point $\mathbf{Q}$ before moving there. This ensures that plants will not "take root" in inappropriate substrates such as metal or glass.

The meaning of the resource function need not correspond to anything occurring in nature, but may merely provide convenient control over the ESA. As an example, spatial resource functions may include attractors or inhibitors [1] which encourage or discourage growth in particular regions, regardless of other factors.

A very important naturally occurring resource is light. The ESA may wish to determine if a static light source is visible from a point $\mathbf{Q}$, or how often a moving light source, like the sun, is visible from that point. The former involves nothing more than the shadowing operation commonly used in ray tracing. Stochastic sampling techniques used for simulating motion blur [4] can be employed in the case of moving lights. By considering not only the overall quantity of light reaching the point $\mathbf{Q}$ but also its direction, the ESA ban bias their progress toward the light, thereby simulating heliotropism.

Internal State

An environment-sensitive automaton is a particle which navigates through a given environment in discrete "time" steps with the aid of environmental information collected and used according to some plant-specific strategy. As in particle systems or the game of Life, the strategy for behavior also controls "death" and "birth." At any generation an automaton may die or spawn one or more new automata which give rise to new branches or leaves of a tree or plant. The branching rate can be purely probabilistic, or a purely deterministic function of generation, or some combination. Except for the probabilistic case, the generation count affects the behavior of the automaton and consequently must be carried along as part of its internal state. Its current position is also part of this internal state.

There is no fixed amount of internal state associated with ESA since they cover an enormous range of possible algorithms for behavior. Thus far the authors have investigated ESA in which generation count is the only indication of past behavior. However, ESA which avoid self-crossing by "remembering" all previous positions are possible as well.

An alternative to internalizing the entire history of spatial positions for self-avoidance is to interleave the generation and interpretation phases. By incrementally updating the environment with the geometrical interpretation of the ESA during generation, subsequent growth will automatically navigate around previous paths. This requires a ray tracing data structure which admits efficient incremental update. Examples of such data structures are those described in [5], [7], and [8].

The state information which initiates the growth process of an automaton is called a *seed*. In the examples which we discuss below, a seed consists of a point on some surface along with the normal vector at that point. Seeds can be conveniently "planted" by casting rays into the environment and using intersection data to construct the required initial state.

Primitive Ray Tracing Functions

For greatest generality the ESA glean all geometrical information about their environment from ray-environment intersection tests, or ray tracing. This gives us tremendous freedom in choosing geometric primitives and operations. Environments may contain tesselations, quadric surfaces, CSG models, etc. The ESA are insulated from the implementation details of the geometry by the single function **Intersect** shown below.

$$\mathbf{Intersect}(\mathbf{R};\mathbf{E}) \rightarrow (d, \mathcal{M}, \mathbf{N})$$

Here **R** is a ray and **E** is the environment. Henceforth we shall suppress the explicit reference to **E**. This function returns the distance to the closest point of intersection (or "infinity" if no intersection exists), and the material descriptor and normal vector of the surface at that point. The exact encodings of these arguments are not important and will vary among ray tracers. All the techniques which we discuss can be based on this function. For convenience we also introduce another function, **Interfere**, shown below, which can be built using **Intersect**.

$$\mathbf{Interfere}(\mathbf{A},\mathbf{B}) \rightarrow \mathbf{True/False}$$

Here **A** and **B** are two 3D points. The function returns **True** if and only if the line segment between **A** and **B** intersects something in the environment.

Two Applications: Ivy and Grass

We demonstrate this approach by creating ivy which conforms to arbitrarily complex surfaces in the environment, and grass which thrives on patches of "ground" where conditions are determined to be favorable.

Figure 1 is a pseudo code outline of the ivy algorithm which incorporates pre-defined constants

Select a seed $(\mathbf{A}_0, \mathbf{N}_0)$
$\mathbf{P}_0 = \mathbf{A}_0 + c_1 * \mathbf{N}_0$
$k = 0$
[1] add $\mathbf{P}_0$ etc. to the skeleton
[2] **REPEAT**
    Pick random $\mathbf{T}$ orthogonal to $\mathbf{N}_k$
    $\mathbf{Q} = \mathbf{P}_k + c_2 * \mathbf{T}$
UNTIL NOT Interfere$(\mathbf{P}_k, \mathbf{Q})$
$d =$ infinity
[3] **FOR 1 TO** c_3 **DO BEGIN**
    Select random ray $\mathbf{R}$ at $\mathbf{Q}$
    $(d', \mathcal{M}, \mathbf{N}') = \mathbf{Intersect}(\mathbf{R})$
    IF $\mathcal{M}$ is an inappropriate
        material **THEN GO TO** [3]
    IF $d' < d$ **THEN BEGIN**
        $d = d'$
        $\mathbf{N}_{k+1} = \mathbf{N}'$
        $\mathbf{A}_{k+1} = \mathbf{Q} + d * \mathbf{R}$
        $\mathbf{P}_{k+1} = \mathbf{A}_{k+1} + c_1 * \mathbf{N}'$
        END IF
    END FOR
IF $d > c_4$ **THEN GOTO** [2]
IF Interfere$(\mathbf{P}_k, \mathbf{P}_{k+1})$ **GOTO** [3]
$k = k + 1$
quit with probability c_5
branch with probability c_6
GOTO [1]

Figure 1: *Outline of the algorithm for growing ivy.*

$c_1, \ldots, c_6$. We have omitted many of the worrisome details in the interest of clarity. For instance, care must be taken to prevent the automaton from getting stuck.

Common remedies include placing maximum iteration counts on loops and using various forms of randomization. The latter can be achieved by replacing the pre-defined constants with randomly sampled intervals. Such randomizations are also useful for adding variation to the model.

We start the algorithm by selecting a seed $(\mathbf{A}_0, \mathbf{N}_0)$, where $\mathbf{A}_0$ is a point on some surface and

$\mathbf{N}_0$ is the corresponding normal. We then compute $\mathbf{P}_0$, a point close to the surface $\mathbf{A}_0$. From $\mathbf{P}_0$ we randomly sample a number of directions in the tangent plane until we can step a given distance to a new point, $\mathbf{Q}$, without interference. From $\mathbf{Q}$ we randomly sample directions looking for the closest surface. If no sufficiently close surface is found, we try another $\mathbf{Q}$. Otherwise, we construct a new candidate point close to $\mathbf{Q}$'s nearest surface and verify that this new point is visible from $\mathbf{P}_0$. If not we seek a different surface near $\mathbf{Q}$. Once this has succeeded, the process begins again from the new point. See Figure 1.

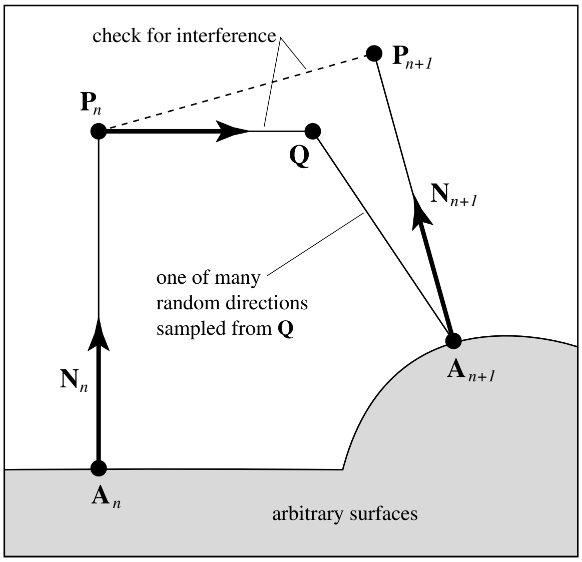

Figure 2: *ESA following a surface.*

The previous algorithm is only intended as a rough guide. Many variations are possible in the methods of selecting $\mathbf{Q}$ and the random rays $\mathbf{R}$, and the stopping and branching criteria.

After the skeleton has been generated by these constrained random walks, the interpretation phase must supply the actual geometry. Here we have chosen to model the branches of the plant with cylinders and the leaves with several triangles. The branch cylinders are constructed by transforming a canonical cylinder as in Figure 3. The transformation matrix is easily constructed from two successive points along a path and the desired branch radius. A more elaborate interpretation might include texture mapping, smooth transitions between branches, and detailed leaves, as described in [3].

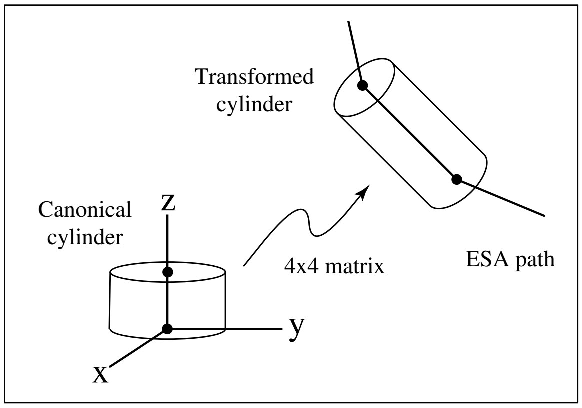

Figure 3: *Transformation of a cylinder.*

This algorithm is easily modified to simulate grass instead of ivy. We force the ESA to explore only surfaces designated as "soil" by the surface properties, generate "blades" which do not interfere with anything in the environment, and finally apply a new interpretation to the resulting skeleton. To make our example more interesting we have used proximity information to create taller grass near obstacles, even though this is more a property of lawn mowers than of grass.

A trivial interpretation is possible in the case of grass since individual blades can be modeled by single disconnected primitives. Observe, however, that information pertaining to blade height and orientation must be communicated via the skeleton because interference checking takes place in the generation phase. A more realistic interpretation might include accurately modeled grass blades, as done by Snyder and Barr [16].

Rendering

The ESA approach is virtually independent of the method of interpretation and subsequent rendering. Since the ESA require the existence of a ray-environment intersection routine, we may employ the same routine for the purpose of rendering via ray tracing. In fact, this is how the authors have created the images appearing in this paper.

It should be noted, however, that the use of ray tracing for rendering is potentially several orders of magnitude more costly than its use during the generation phase. This is due to the vast increase in the number of rays which are traced, and also to the potentially enormous number of primitive objects used in modeling the botanical forms. Thus it is even more critical that an efficient ray tracing algorithm be used for the rendering than for generation. The authors have found uniform subdivision [5] and ray classification [2] to be effective in both phases.

A very useful alternative rendering technique is depth buffering. Since the depth buffer algorithm permits incremental updates of geometry, we can initially render the environment without the interpretation of the skeleton, plant the seeds interactively, and watch the ESA progress. Current state-of-the-art workstation graphics hardware is capable of carrying out the incremental updates in real time. This opens up the possibility of interactively controlled resource functions which influence growth directions and rates.

Results

Figures 4 through 7 show results of the ivy algorithm. In figures 4 and 5 a seed was planted at the base of each vertical cylinder of the trellis. The pattern of growth differs in these two cases because a resource function was used in the second to encourage growth at higher altitudes, and a lower branching probability was used. Figure 6 shows a closeup view of the ivy, and Figure 7 is an example of a more complex environment. Note that the ivy has grown into the recessed areas formed by CSG subtract operations. Figure 8 is an example of grass growing among several obstacles. Six seeds were planted to generate this grass. Note that it is taller near the obstacles. Finally, Figure 9 shows the number of CPU seconds required by the generation phase and the number of primitive objects created by the interpretation phase. All benchmarks were performed on an Apollo DN590-T with 16 Mb of physical memory.

Future Directions

The power of ESA does not lie in a specific algorithm, but rather in the simplicity and generality of the underlying concept. The ivy and grass examples are two immediate applications of this concept, but there are clearly many others. The surface-following property used in the ivy example could be used to grow barnacles on a ship's hull, or coral on a rock, or even stalactites on the ceiling of a cave.

The simple and approximate collision detection of the ESA can also be used to animate "small" creatures in the environment. Examples include fish swimming among coral and insects crawling on surfaces. This suggests a possible connection with the behavioral models of flocks, schools, and herds introduced by Reynolds [13].

Finally, rain or snow can be modeled by ESA which begin in the sky and fall or flutter to earth, interacting with arbitrarily complex obstacles. Accumulation can be modeled by particles which continue to exist after impact and affect the rest positions of subsequent particles.

Summary

We have introduced the notion of environment-sensitive automata, or ESA, for modeling a variety of botanical forms. Though the ESA "reproduce" and "die" as in particle systems, they are unique in that these events, as well as growth directions, may be influenced by material and geometrical properties of their surroundings.

Favorable growth directions are selected stochastically based on environmental criteria including the approximate distance to the nearest surface or expected availability of light and other resources.

Organic-looking forms such as ivy and grass have been created by using the network of paths generated by the ESA as a modeling framework.

Though the generation of this framework, or skeleton, relies upon the ray-environment intersection capability of ray tracing, the resulting geometry may be rendered in other ways, including depth buffering.

It is clear that the botanical forms which we have investigated constitute a very small sampling of the phenomena which can be modeled using this technique.

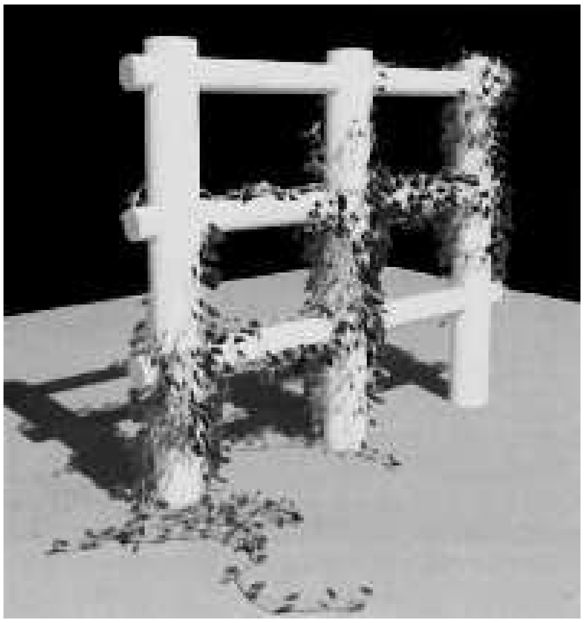

Figure 4: *Ivy and trellis 1.*

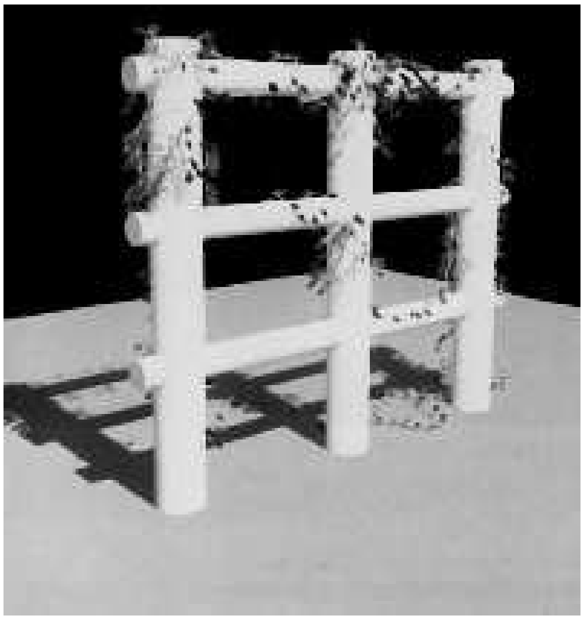

Figure 5: *Ivy and trellis 2.*

Figure 6: *Closeup of ivy and trellis 1.*

References

[1] M. Aono and T. L. Kunii. Botanical tree image generation. *IEEE Computer Graphics and Applications*, 4(5):55–64, 1982.

[2] James Arvo and David Kirk. Fast ray tracing by ray classification. *Computer Graphics*, 21(4):55–64, July 1987.

[3] Jules Bloomenthal. Modeling the mighty maple. *Computer Graphics*, 19(3):305–311, July 1985.

[4] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(3):137–145, July 1984.

[5] Akira Fujimoto and Takayuki Tanaka. ARTS: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, April 1986.

[6] Martin Gardener. Mathematical games. *Scientific American*, 223(4):120–123, October 1970.

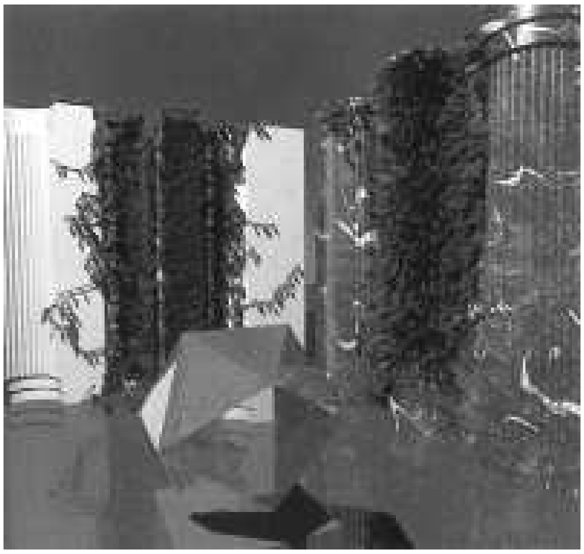

Figure 7: *Ivy and a complex CSG model.*

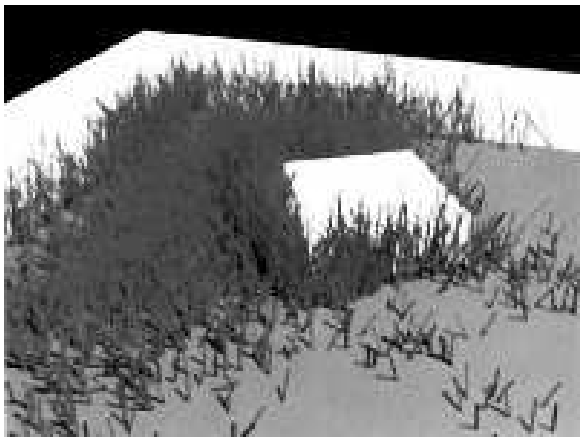

Figure 8: *Grass and obstacles.*

Scene	CPU seconds	number of primitives
Trellis 1	61	3804
Trellis 2	21	1266
Grass	36	3173

Figure 9: *Ivy and grass generation statistics.*

[7] Andrew Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[8] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[9] Timothy L. Kay and James Kajiya. Ray tracing complex scenes. *Computer Graphics*, 20(4):269–278, August 1986.

[10] Aristid Lindenmayer. Mathematical models for cellular interactions in development. *Journal of Theoretical Biology*, 18:280–315, 1968.

[11] William T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *Computer Graphics*, 17(3):359–376, July 1983.

[12] William T. Reeves. Approximate and probabilistic algorithms for shading and rendering structured particle systems. *Computer Graphics*, 19(3):313–322, July 1985.

[13] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, July 1987.

[14] Steve Rubin and Turner Whitted. A three-dimensional representation for fast rendering of complex scenes. *Computer Graphics*, 14(3):110–116, July 1980.

[15] Alvy Ray Smith. Plants fractals and formal languages. *Computer Graphics*, 18(3):1–10, July 1984.

[16] John M. Snyder and Alan H. Barr. Ray tracing complex models containing surface tessellations. *Computer Graphics*, 21(4):119–128, July 1987.

[17] Hank Weghorst, Gary Hooper, and Donald Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, January 1984.

[18] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 32(6):343–349, June 1980.