VOXEL SPACE AUTOMATA: MODELING WITH STOCHASTIC GROWTH PROCESSES IN VOXEL SPACE

Ned Greene

NYIT Computer Graphics Lab†
Old Westbury, New York

Abstract

A novel stochastic modeling technique is described which operates on a voxel data base in which objects are represented as collections of voxel records. Models are "grown" from predefined geometric elements according to rules based on simple relationships like intersection, proximity, and occlusion which can be evaluated more quickly and easily in voxel space than with analytic geometry. Growth is probabilistic: multiple trials are attempted in which an element's position and orientation are randomly perturbed, and the trial which best fits a set of rules is selected. The term *voxel space automata* is introduced to describe growth processes that sense and react to a voxel environment.

Applications include simulation of plant growth, for which voxel representation facilitates sensing the environment. Illumination can be efficiently estimated at each plant "node" at each growth iteration by casting rays into the voxel environment, allowing accurate simulation of reaction to light including heliotropism.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Curve, surface, solid and object representation. I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism. I.6 [Simulation and Modeling]: Applications.

Additional Keywords and Phrases: Voxel, automata, stochastic processes, illumination, heliotropism, radiosity

† Author's current address:
Apple Computer, 20525 Mariani Ave., Cupertino, CA 95014

1. Voxel Space

By a *voxel space* we mean a region of three dimensional space partitioned into identical cubes (volume elements or *voxels*), typically a region bounded by a rectangular solid so that it can be represented as an array or octree of voxel records. Modeling and rendering techniques which operate on a voxel space are the subject of increasingly active research. Various volume rendering techniques have been developed to visualize data produced by 3D medical imaging devices and computational simulation [5], [19], [21]. In the synthesis domain, voxel spaces have been employed to create surfaces defined by implicit functions [2], [13], [22].

Alternatively, arbitrary three dimensional shapes can be represented in voxel space by marking the voxels that they intersect as "occupied" [11]. This representation is necessarily approximate, since it only indicates which voxels are intersected by the object, not the object's actual surface. Multiple objects can be distinguished from each other by assigning each a unique voxel value, so any collection of non-intersecting objects can be represented. Although partitioning space into voxels makes geometric calculations only approximate and puts a lower limit on the size of objects that can be distinguished, for many applications these disadvantages are outweighted by convenience and speed.

Suppose we wish to initialize voxel space with an environment modeled as a collection of geometric primitives such as lines, polygons, and polyhedra. The process of identifying and labeling voxels that are intersected by a primitive object is referred to as *tiling*. Kaufman has outlined incremental techniques for tiling various primitives [11], although his criterion for tiling a voxel is somewhat different than the simple intersection criterion employed herein: a voxel is tiled if the cube representing its extent is intersected. Figure 1 shows voxel representations of a line and a polygon.

Voxel representation of an environment simplifies geometric operations such as intersection testing and measuring the relative proximity of objects. Whether an object intersects another object already represented in voxel space may be determined by testing each voxel that it intersects to see if it's already occupied. This method of sensing intersection is only approximate in the sense that two non-intersecting objects can intersect the same voxel, in which case intersection will be falsely detected. But it is faster and more convenient than the conventional method of intersecting one geometric element with all other elements in its vicinity using analytic geometry. The conventional approach can be difficult to implement, particularly if a model is constructed from different surface types (polygons, quadric surfaces, patches, etc.). With the voxel method, tiling is the only geometric operation which must be performed, and testing an element of one surface type against an element of another surface type only requires the ability to tile voxel space with each. In addition, performance is independent of scene complexity and does not depend on the number of nearby objects.

Voxel representation also simplifies determining proximity relationships. Given a point in space we may identify the nearest object by scanning voxels in the neighborhood and comparing distances to occupied voxels encountered. Alternatively, the process may be facilitated by adding "boundary layers" of voxels to objects in the environment. According to this scheme, voxels adjacent to voxels which are part of object N are marked as being in boundary layer 1 of object N, voxels one additional layer removed from object N are marked as being in boundary layer 2, and so forth up to some specified number of boundary layers. If L boundary layers have been added to all objects, for any point in voxel space we may immediately determine whether it lies within L voxels of an object, and if so, the identity of the nearest object.

As the discussion will show, voxel representation also facilitates ray casting and a variety of other geometric operations.

2. Growth Systems

The computer graphics literature includes a variety of approaches to simulating plant growth including particle systems [17],[20], graftals [20], and fractals [14]. Recently, Prusinkiewicz et al. and de Reffye et al. have approached the problem with empirically based models of plant development, producing relatively realistic models of actual plant species [16], [18].

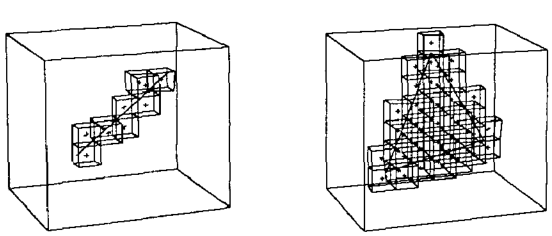


Figure 1.

An 8x8x8 voxel space with voxel representations of a polygon and a line

Less attention has been paid to the problem of simulating the effects of environmental factors on plant development, which are particularly important in complex environments where plants interact as they compete for space and light. To faithfully mimic a natural growth process which senses and reacts to the environment, a simulated growth process must sense and react to the environment. In crude terms, growth is affected by conditions within the local environment: obstacles should be avoided, proximity to objects or other organisms may inhibit or promote growth, and growth is modulated by available light. At a minimum, simulated growth processes should be aware of these conditions.

Arvo and Kirk have described growth processes capable of sensing the environment which they refer to as "environment-sensitive automata" [1]. Their method performs ray casting to test for intersection and proximity, and they have applied the technique to simulate clinging vines and patches of grass which avoid obstacles. They mention that the method could be extended to simulate heliotropism (sun seeking) and other effects. Their sole means of sensing the environment is ray-object intersection, which limits the type of information that can be obtained.

This article argues that voxel representation simplifies sensing of the environment by growth processes. In particular, it is easier to obtain geometric information by scanning or sampling a voxel environment than by ray casting a conventional model. From any point in voxel space the size, shape, and proximity of neighboring objects can be determined by inspecting the records of nearby voxels. Voxel records may include information about material properties, making it straightforward to confine growth to appropriate regions of the environment. A variety of statistics about the local environment such as "center of mass" and "density" are readily obtained.

Illumination, which depends on the global environment, can be estimated by sampling with ray casting. In this context, ray casting means tiling a ray in voxel space; a ray is occluded if it intersects an occupied voxel. While this means of estimating illumination isn't as accurate or general as ray tracing, it is sufficient to estimate exposure to sunlight and "skylight" in an outdoor scene, and it can be performed very efficiently since it does not require ray-object intersection or any substantial analytic geometry. Fujimoto et al. discuss incremental methods for tiling rays in the context of using uniform spatial subdivision (like a coarse voxel space) to enhance ray tracing performance [6] . The efficiency of ray casting in voxel space makes it feasible to build an illumination table at each active plant "node" at each iteration, allowing accurate simulation of reaction to light including heliotropism.

A paradigm for growth in voxel space may be outlined as follows. The initial state of voxel space is specified, either "empty" or tiled with a three dimensional model of an environment. Beginning at specified seed points, models are grown from predefined geometric elements, added one by one to the model subject to satisfying a set of rules. Typically, rules consist of geometric constraints based on simple relationships like intersection, proximity, and occlusion which are evaluated by sensing the voxel representation of objects. Growth is probabilistic: multiple trials are attempted in which an element's position and orientation are randomly perturbed, and the trial which best fits the rule set (if any trial does) is selected. Various methods of propagation can be employed for choosing possible sites for new growth - tree-structured "random walk," "diffusion," etc. Experience has shown that a few simple rules are sufficient to simulate some complex phenomena.

The term *voxel space automata* will be applied to growth processes that sense and react to a voxel environment. The term automata is used informally here, and the approach presented in this article is only loosely related to the formal mathematical realm of cellular automata [15]. While both approaches are rule-based and operate on a matrix of cells, contrary to the spirit of cellular automata the implementation presented here is driven by geometry, and voxel representation functions primarily to simplify geometric operations. Nevertheless, some of the methods described are formally developed in the cellular automata literature, for example rules based on inspection of neighborhoods.

With respect to plant simulation, this paper is primarily concerned with environmental factors - obstacle avoidance, reaction to light, etc. The developmental model employed, a form of random walk subject to constraints, is not adequate for realistic simulation of most plant species. In this sense, the examples cited are more a detailed illustration of how voxel space automata work than a serious attempt to simulate plant growth. The latter objective requires a sophisticated model of plant development in addition to the ability to sense the environment, which suggests the possibility of combining the two in a voxel space automaton.

3. Geometric Operations in Voxel Space

Constraints based on geometric rules such as intersection avoidance and proximity constraints provide a high level means of controlling growth. The skeleton of Figure 2 represents a tree-structured "random walk" through voxel space constrained by intersection avoidance. If growth in one randomly perturbed direction results in intersection with an occupied voxel, that trial is rejected and growth in another direction is attempted. Incidentally, the term "random walk" is used informally throughout this paper; segment directions are not chosen at random, but are generated by perturbing the direction of the last segment according to a parameterized distribution function.

Since we are generating and evaluating randomly perturbed trials, this approach to intersection avoidance may be considered a "Monte Carlo" method [10]. In a sense, the growing model feels its way through voxel space by sensing the voxel representation of objects.

Figure 3 illustrates how growth can be controlled with simple biases and constraints. Panel A shows the initial state of voxel space in cross-section: a cylindrical column is surrounded by four boundary layers (records for voxels in this region include information about proximity to the column). Panel B shows growth with a slight bias to grow upward, implemented by interpolating trial segment directions with a vertical vector. Panel C shows growth with an upward bias and subject to a proximity constraint (on average, voxels intersected by a segment must lie within two voxels of the column). Panel D shows growth with biases to grow upward and helically twist, subject to a proximity constraint.

Growth rules can be based on any relationship that can be evaluated by reading voxel records. For a particular application, "center of mass" within a certain region or the "density" of a certain object within a certain radius may be of interest. The helical bias apparent in Figure 3D was imparted by perturbing trial segment directions toward one side of a plane defined by the last segment and the local "center of mass."

If information about a large region of voxel space is desired, sampling is an alternative to examining every voxel within the region. In evaluating relationships like illumination that depend on the global environment, sampling methods may be the only feasible approach.

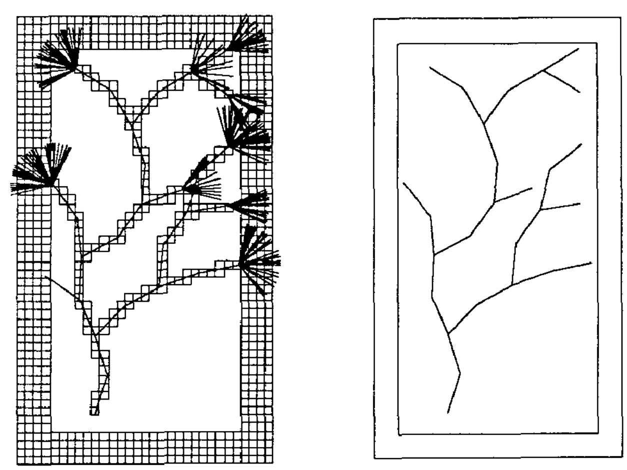


Figure 2.

Skeleton generated by tree-structured random walk through 2D voxel space with intersection avoidance. Frame at left shows all attempts to place segments and voxels intersected by successfully placed segments. Fan-shaped clusters represent unsuccessful trials. Skeleton generated is shown at right.

The vine model of Figure 5 illustrates how a complex model can be produced from a few simple rules. The vines were grown in 51 iterations in a 165x150x195 voxel space which was initialized by tiling with a polygonal model of the wall and ground plane and then adding four boundary layers. Figure 4 lists the growth rules which generated the vines. The proximity constraint which held vine growth close to the wall has already been discussed. Illumination rules which confined vine growth to regions of the wall with substantial exposure to light are discussed in the following section.

4. Determining Available Light for Plant Simulation

Light is one of the most important environmental factors to be considered in simulating growth of photosynthetic plants. For typical species, normal development requires illumination within a certain range, light intensity affects rate of growth, and shadowing within the local environment may affect direction of growth. Since different parts of an organism may react differently to light, and shadowing changes from iteration to iteration, accurate simulation of reaction to light requires finding illumination at numerous sites on an organism at each iteration. For example, development of an individual tree limb is affected by illumination within the local environment which changes over time as neighboring limbs develop. On a smaller scale, development of an individual leaf may depend on available illumination in its local environment. Ideally, we would like to be able to estimate illumination at each plant "node" at each growth iteration. The efficiency of sampling methods for estimating illumination in voxel space makes this feasible to do.

In an outdoor scene, "direct" illumination comes from the sky hemisphere. To estimate exposure to the sky of an arbitrary point in voxel space we cast rays from the point toward points on the sky hemisphere. In this context, "casting a ray" means tiling a ray in voxel space; it is occluded if it intersects an occupied voxel. If we cast 100 rays from a point toward the sky and 40 of them are occluded, the point's exposure to sky is 0.6. In growth rules, this quantity will be called *sky_exposure*, which ranges in value between 0 (complete occlusion) and 1 (complete exposure).

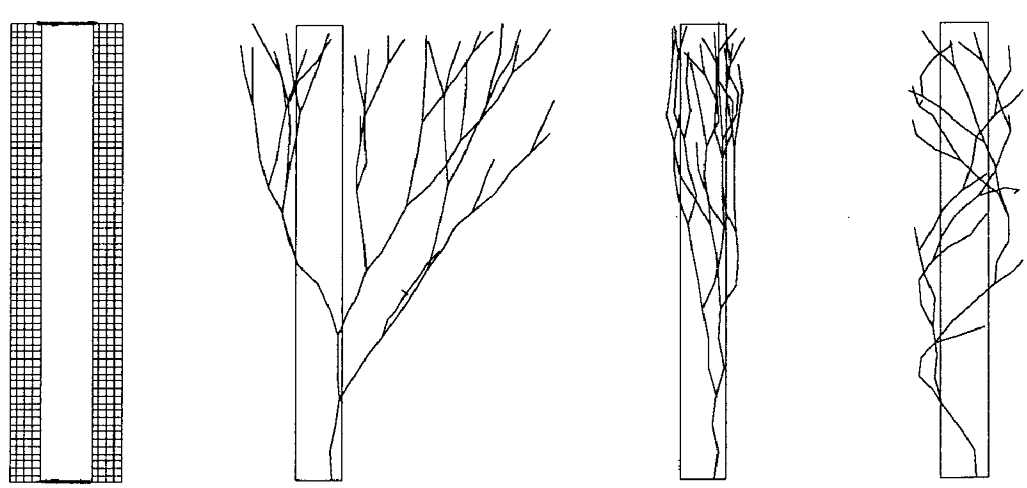
Similarly, to estimate exposure of an arbitrary point to direct sunlight, rays are cast towards points on a 180 degree arc representing the sun's trajectory and the fraction of occluded rays is determined. In growth rules, exposure to the sun's trajectory will be called *sun_exposure*, which ranges in value between 0 and 1.

To confine growth of a plant species to regions of the environment having the appropriate exposure to light we may specify minimum and maximum values for exposure expressed as a blend of *sun_exposure* and *sky_exposure*. For example, the illumination constraint from the growth rules for the vines of Figure 5

```
light {
  blend  sun=0.8  sky=0.2
  exposure  min=0.3  max=1.0
  boost 1.8
}
```

confines growth to regions of the environment with between

Figure 3.



A B C D

30 and 100 percent of full exposure, and exposure is measured as an 80%/20% weighted average of *sun_exposure* and *sky_exposure*. The "blend" ratio is a way of specifying the relative importance of *sun_exposure* and *sky_exposure* which depends on mean climatic conditions (e.g., degree of cloud cover) and the illumination requirements of a particular species. In the scene of Figure 5 the sun's trajectory is inclined at 20 degrees from vertical, "behind" the wall with a window, so vine growth is confined to the brighter regions of the wall as expected.

Figures 6A-6F were rendered from the voxel representation of the scene. Panels A and B, showing *sun_exposure* and *sky_exposure* respectively, were produced by estimating exposure at each occupied voxel by casting rays as previously described. Note that the image of *sun_exposure* is not a "shadow matte" corresponding to shadows cast by the sun in a fixed position, but rather a time exposure indicating average exposure to direct sunlight in the course of a day. Panel C shows the region of the environment above the illumination threshold for the vines, which corresponds nicely with the actual growth pattern.

Of course growth processes don't need to know about illumination in the whole environment; they determine illumination at specific sites in voxel space as needed to evaluate illumination constraints. For example, as the vines grew, *sun_exposure* and *sky_exposure* were determined at one location for each growing tendril at each iteration, and growth at a particular tendril stopped whenever illumination fell below the threshold.

The "boost" parameter in the constraint affects rate of growth, average number of leaves per segment, and leaf size. For example, a simple linear relationship between exposure and scale produced leaves having full exposure (1.0) that were 1.8 times the scale of leaves having the minimum exposure (0.3). Accordingly, leaves are larger and more numerous at the top of the wall where illumination is high than on the wall's vertical faces. Of course this crude intuitive model for modulating growth would benefit from empirical study.

Figures 6A and 6B were created to show that the method for estimating illumination is accurate enough to evaluate illumination constraints. They also suggest that estimating illumination in this manner may have application to surface shading. A detailed discussion of rendering issues is beyond the scope of this paper, but a few observations are in order.

5. Estimating Diffuse Reflection

Estimating illumination by ray casting on a voxel by voxel basis as previously described is essentially a radiosity approach which estimates diffuse reflection [7], and interreflection of light among objects in the environment can be simulated by making multiple passes through the voxel data. In producing Figures 6A and 6B a ray sample was black if the ray was occluded, otherwise white. On a second pass through the voxel data, occluded rays can be assigned the gray level of the intersected voxel (instead of black), improving accuracy, and subsequent passes further refine the image. The same strategy can be applied to color rendering if color information is stored at each voxel. A "first pass" color rendering of the scene (Panel F) has been simulated by blending a shadow matte (Panel D) with the image of *sky_exposure* (Panel B) to approximate overall illumination, and matting an image of surface color (Panel E) through this image.

Figures 6A-6F were produced with a conventional polygon renderer by drawing a cube for each occupied voxel in the scene. Alternatively, colors associated with occupied voxels can be applied to a polygonal model by dicing each polygon to the voxel grid and assigning each fragment the color of the corresponding voxel. Figures 5 and 9A were produced in this manner.

Figure 4: Growth rules for vines of Figure 5

```
/* 17 seed locations (at base of wall, inside and outside the courtyard) */
seed 0.50 -0.89 0.56
seed 0.35 -0.89 0.56
 ...
seed 0.56 -0.89 -0.60

/* skeleton parameters */
random_tree {
  length 2.4 /* limb length in "voxel widths" */
  branch_age_range 1 3 /* 1, 2, or 3 iterations before branching (picked at random) */
  branch_angle 60 /* (degrees) */
  vertical_bias .08 /* slight bias to grow upward */
  no_of_trials_max 300 /* try 300 randomly perturbed trials before giving up */
  no_of_trials_min 150 /* try 150 trials before picking best proximity fit */
  seekprox 1.5 /* pick trial with avg. proximity closest to 1.5 "voxel widths" */
  maxprox 3.0 /* reject trials with avg. proximity greater than 3.0 "voxel widths" */
}

/* illumination */
light {
  blend sun=0.8 sky=0.2 /* blend of sun_exposure and sky_exposure */
  exposure min=0.3 max=1.0 /* nodes with less than 30% expo. become inactive */
  boost 1.8 /* full exposure nodes grow 1.8 times faster than 30% exposure nodes */
}

/* leaf element */
element {
  number_of_trials 30 /* attempt placement 30 times before giving up */
  expected_frequency 1.5 /* try to place 1.5 leaves per branch segment */
  model { /* coordinates of 2 polygons in leaf model */
    polygon (0,.32,.04),(-.23,.13,-.01),(-.36,.44,-.11),(-.24,.76,-.11),(0,.99,-.06)
    polygon (0,.32,.04),(0,.99,-.06),(.28,.76,-.13),(.41,.45,-.13),(.25,.13,-.02)
  }
}
```

In making these images, voxel colors were obtained by multiplying surface color, shown in Panel E, by *sky_exposure*, like Panel B but estimated by casting 100 rays from each occupied voxel toward randomly selected points on the sky hemisphere. Using random ray directions overcomes quantization caused by shooting the same fixed pattern of rays at each voxel (apparent in Panels A and B) but introduces noise, most evident in Figure 5 in the salt and pepper texturing of the wall. Noise can be reduced by using Cook's stochastic sampling method of picking ray directions at random, but rejecting directions that are within some small angular displacement of a previously selected direction to avoid clustering of samples [4]. Of course shooting more rays also reduces noise.

If conventional volume rendering techniques are applied, jagged edges can be avoided by storing an alpha value at each voxel indicating the fraction of a voxel's volume that is occupied by intersecting objects. Then ray marching from the eyepoint through voxel space, accumulating opacity along each ray, would produce images free of aliasing, provided that the limit on spatial frequencies imposed by the sampling theorem is observed [5] (actually, this requirement is not met by the voxel environment of Figure 6).

As presently implemented, ray samples shot from a voxel are weighted equally without regard to direction, which fails to simulate Lambertian reflection as the radiosity model dictates [7]. Proper simulation of Lambertian reflection requires estimation of a "surface normal" at each voxel.

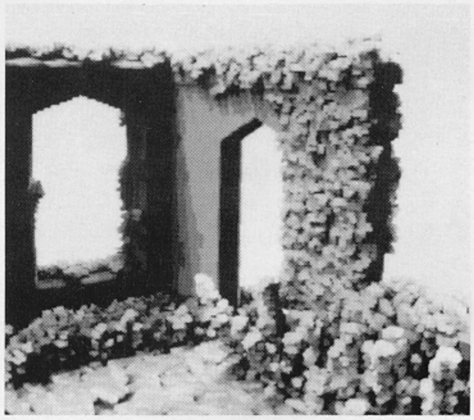
Estimating diffuse reflection by ray casting may prove to be more practical than a conventional radiosity approach for complex scenes. If a voxel environment is represented as a 3D array, the computational cost of ray casting at a voxel is proportional to the linear resolution of voxel space and otherwise independent of scene complexity. So, for example, doubling the x, y, and z resolution of voxel space, which allows representing an environment of eight times the complexity, would increase the cost per voxel of estimating diffuse reflection by a factor of two, and the total cost of estimating diffuse reflection for the scene by a factor of sixteen. In general, a K-fold increase in complexity increases running time by a factor of $K^{4/3}$. With octree representation of the environment, complexity characteristics for some environments are considerably better. However, this analysis assumes constant time for voxel access which becomes less feasible as storage requirements increase.

With a conventional radiosity approach, estimating illumination for N patches is an $O(N^2)$ computation [3], so a K-fold increase in scene complexity produces an $O(K^2)$ increase in running time.

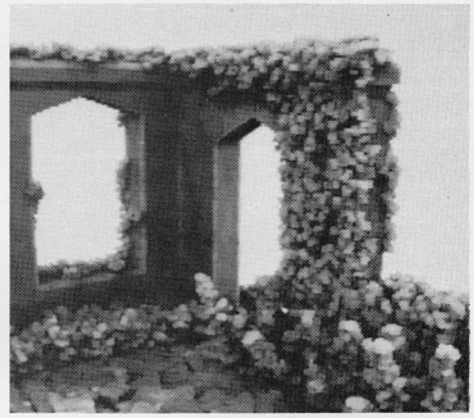
As is apparent from the preceding discussion, the author's experience with this approach to rendering voxel environments is very limited. Presently, the method should be considered a curiosity deserving of further study due to its favorable complexity characteristics.

Figure 5.

This image was created by estimating diffuse reflection at occupied voxels and then dicing the polygonal model of the scene to the voxel grid, assigning each fragment the color of the corresponding voxel. There are approximately 27,000 polygons in the scene and image generation took roughly 30 hours on a sun4. The color table is nonlinear, making dark areas appear brighter.



6A. *sun_exposure*

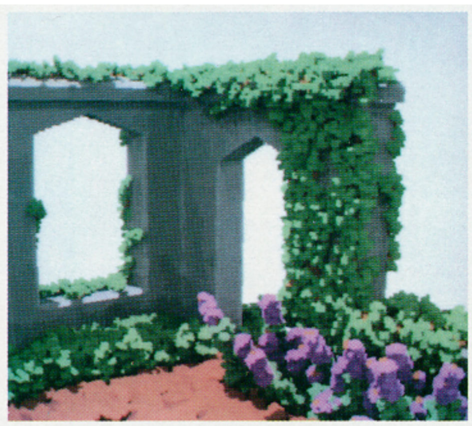


6B. *sky_exposure*

6C. Illumination threshold for vines

6D. Shadow matte

6E. Surface color



6F. Simulated "first pass" color rendering

Figure 6.

This environment was represented as a 165x150x195 voxel space, initialized by tiling with a polygonal model of the wall and groundplane. A growth program produced the configuration of paving tiles in addition to the vegetation. Images were created with a conventional polygon renderer by drawing a cube for each occupied voxel in the scene.

6. Simulating Heliotropism

Heliotropism (sun seeking) can be simulated by constructing a latitude-longitude illumination table at each node of a plant skeleton at each iteration and biasing growth in the direction of the "brightest spot" in the table. To build a table we cast rays originating from the node in the directions of the azimuths and elevations in the table. Obstructed rays are represented by black table entries, unobstructed rays with white.

Once a table has been constructed, the direction of the brightest spot can be estimated by low-pass filtering the table and looking for the greatest value. Applying a technique which has been used to make diffuse illumination tables for environment mapping, the table can be filtered by convolving with a Lambert's law cosine function, a kernel covering one hemisphere of the environment [12],[8].

The process is illustrated in the example of Figure 7 where a single tendril grows directly toward the brightest spot in the sky, beginning from a point sheltered from direct sunlight. At each of five iterations a latitude-longitude illumination table of the environment is constructed from the "viewpoint" of the tendril's tip (Column A). This table is matted with a table representing background illumination (mean brightness of the sky at latitude-longitude coordinates), shown at bottom left, to produce the tables of Column B. These tables are then low-pass filtered to estimate the brightest spot in the sky, marked with a cross.

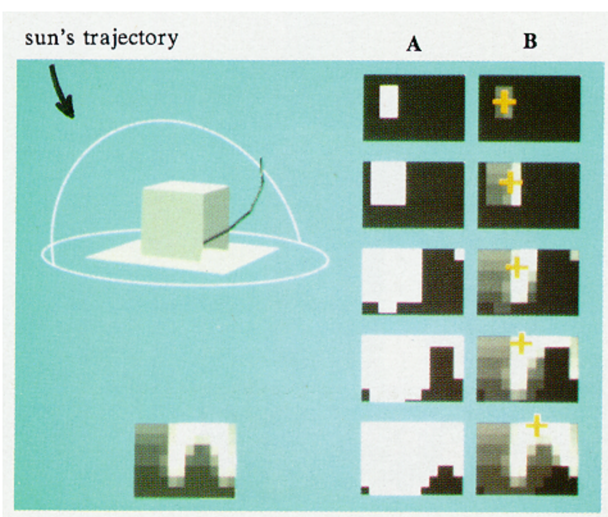


Figure 7.

Illustration of heliotropism: tendril sheltered by cube with one side open grows towards "brightest spot" in the sky at each of five iterations, growing out open side and upward toward apex of sun's trajectory.

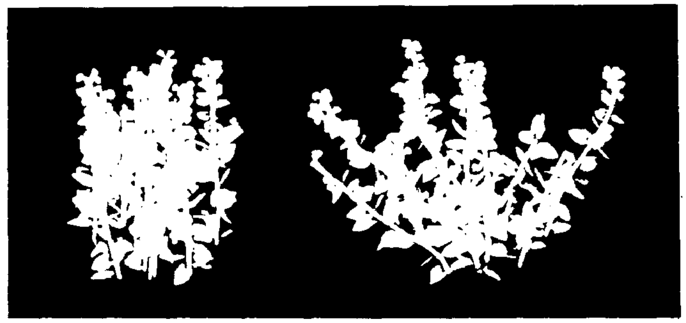


Figure 8.

The efficiency of ray casting in voxel space makes simulation of heliotropism practical in complex environments. In growth rules, heliotropism is expressed as a bias, ranging in value between 0 and 1, 0 meaning none and 1 meaning that growing limbs point directly toward the brightest spot in the sky as in the example of Figure 7 (the numerical value applies to simple linear interpolation of direction vectors).

The two clusters of plants shown in Figure 8 were grown from identical growth rules, except that the cluster on the right included a bias for heliotropism which was expressed as follows:

```
light {
  blend sun=0.0 sky=1.0
  exposure min=0.0 max=1.0
  seeksun_bias 0.15
}
```

While the plant on the right obviously looks more natural, no claim is made that heliotropism produces such pronounced effects in nature. But the example does suggest that this simple intuitive approach to simulating heliotropism can enhance realism. As a second example, the growth rules for the flowering plants of Figure 5 included a similar bias for heliotropism which accounts for their tendency to grow away from the wall and toward open space.

Summarizing the discussion of light, methods have been presented to confine development of plants to regions of the environment with suitable illumination, to modulate growth by available illumination, and to bias growth in the direction of brightest light. Of course reaction to light is a complex phenomenon that varies from species to species. Although rules based on intuitively obvious relationships such as the ones given in this paper may produce reasonable looking results, simulation that is true to nature requires rules based on empirical study. In any case, voxel representation provides a convenient and efficient way of sensing illumination in the environment.

7. Stochastic Detailing of Geometric Models

Thus far, the discussion has focused on simulation. A second application of voxel space automata, which may be described as "stochastic detailing," involves producing a detailed model from a 3D "rough sketch" of underlying geometry. Given a simple model which has been placed in voxel space by tiling, we may produce a detailed counterpart by tracking features of the model and adding predefined geometric elements to the environment according to rules based on geometric constraints or other conditions.

Various techniques enhance the method's versatility. Different growth rules may be applied to different regions of the model by partitioning the underlying model into discrete regions, associating a set of growth rules with each region, and then selecting among rule sets as growth proceeds depending on which region of the underlying model is in closest proximity.

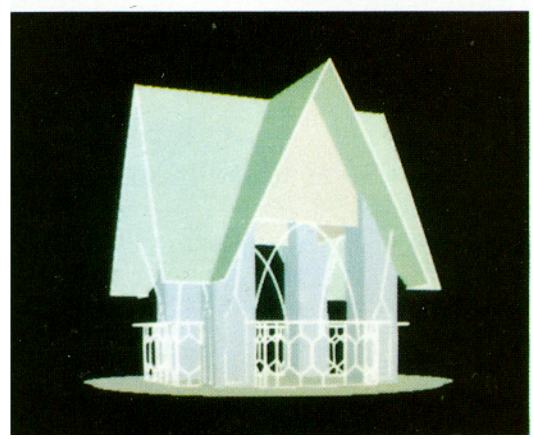
Regions of an underlying model may also be used to represent different material properties. In the scene of Figure 5, for example, paving tiles are one region and the associated rule set did not permit plant growth.

Another mechanism for switching among rule sets involves specifying multiple rule sets, and if a "primary" rule set is not satisfied in a given situation, switching to a "secondary" rule set, and so forth.

As an illustration of these techniques, the model of Figure 9C served as a crude template for the model of Figure 9A, shown without foliage in Figure 9B. A set of growth rules was associated with each of several regions of the underlying model, and in this way the character of different regions of the model was independently controlled: limbs near the gables were biased to grow away from the apices of the arches, limbs near the edge of the roof were biased to grow down a few iterations and then stop, and so forth.

9A

9C



9B

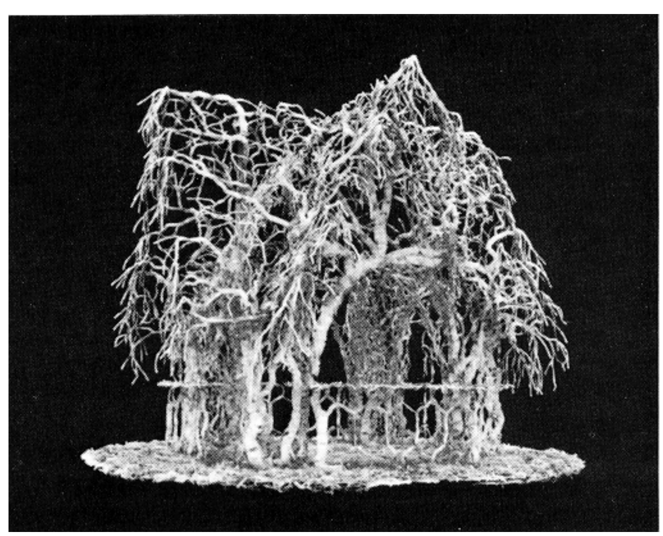


Figure 9.

Above: Model from "Organic Architecture" [9], animation of growth in a 300x300x300 voxel space from the Siggraph '88 Film Show. Panel C shows the crude model that the growth program tracked and Panel B shows the model without foliage.

Panel A was produced by applying voxel color assignments to the polygonal model, which consists of roughly 650,000 polygons. This "first pass" rendering took about 35 hours on a sun4 (voxel resolution: 180x150x180).

Primary and secondary rule sets were employed to track lines delineating the "railing" in the underlying model. To make the skeleton branch where lines in the underlying model forked and nowhere else, a primary rule set attempted to branch everywhere, subject to a proximity constraint, succeeding only in the neighborhood of a fork. When branching failed, a secondary rule set attempted to place a single limb segment subject to a proximity constraint, often succeeding, allowing growth to continue. This strategy produced full delineation of the railing tracery, as is apparent in Figure 9B.

The essential method of rule-based stochastic growth in voxel space is more general than the examples presented suggest. Mode of propagation need not be a random walk; alternatives include some form of "diffusion" - given successful placement of an element, new sites for potential growth in the neighborhood can be chosen according to a distribution function. The configuration of paving tiles in Figure 5 was produced from three primitive elements with this propagation mode, subject to constraints on proximity and intersection. This Monte Carlo approach to arranging primitive elements in close proximity while avoiding intersection may prove to have wide application in assembling complex models from randomly arranged elements.

8. Conclusion

At arbitrary locations in voxel space, the local environment can be easily and efficiently scanned and the global environment can be easily and efficiently sampled. These properties make voxel representation useful for growth processes that sense and react to an environment.

9. Acknowledgements

This paper is the core of a masters thesis submitted to the Computer Science Department of the Courant Institute of Mathematical Sciences, New York University. I would like to thank Prof. Ken Perlin for acting as my thesis advisor and most of all for encouraging me to enter the masters program at Courant. I would also like to thank Mike Gwilliam who very generously lent his time to help with image generation, producing Figures 5 and 9A. Paul Heckbert and Jules Bloomenthal reviewed the manuscript and offered many helpful suggestions, as did one of the anonymous reviewers. Photographic work by Ariel Shaw.

10. References

1. Arvo, James, David Kirk, Modeling Plants with Environment-Sensitive Automata, *Proceedings of Ausgraph '88*, 27-33.

2. Bloomenthal, Jules, Polygonization of Implicit Surfaces, *Computer Aided Geometric Design*, 5, 4 (Nov. 1988), 341-355.

3. Cohen, Michael F., Shenchang E. Chen, John A. Wallace, Donald Greenberg, A Progressive Refinement Approach to Fast Radiosity Image Generation, *Computer Graphics*, 22, 4 (Aug. 1988), 75-84.

4. Cook, Robert L., Stochastic Sampling in Computer Graphics, *ACM Transactions on Graphics*, 5, 1 (Jan. 1986), 51-72.

5. Drebin, Robert A., Loren Carpenter, Pat Hanrahan, Volume Rendering, *Computer Graphics*, 22, 4 (Aug. 1988), 65-74.

6. Fujimoto, Akira, Tanaka Takayuki, Kansei Iwata, ARTS: Accelerated Ray-Tracing System, *IEEE Computer Graphics and Applications*, 6, 4 (Apr. 1986), 16-26.

7. Goral, Cindy M., Kenneth E. Torrance, Donald P. Greenberg, Modeling the Interaction of Light Between Diffuse Surfaces, *Computer Graphics*, 18, 3 (July 1984), 119-128.

8. Greene, Ned, Environment Mapping and Other Applications of World Projections, *IEEE Computer Graphics and Applications*, 6, 11 (Nov. 1986), 21-29.

9. Greene, Ned, Organic Architecture [videotape], Siggraph Video Review 38, (Aug. 1988), ACM Siggraph, New York, segment 16.

10. Halton, J. H., A Retrospective and Prospective Survey of the Monte Carlo Method, *SIAM Rev.*, 12, 1 (Jan. 1970), 1-63.

11. Kaufman, Arie, 3D Scan Conversion Algorithms for Voxel-Based Graphics, *Proceedings of 1986 Workshop on Interactive 3D Graphics*, (Oct. 1986), 45-75.

12. Miller, Gene S., and C. Robert Hoffman, Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments, *SIGGRAPH 84: Advanced Computer Animation Seminar Notes*, (July 1984).

13. Norton, Alan, Generation and Display of Geometric Fractals in 3D, *Computer Graphics*, 16, 3 (July 1982), 61-67

14. Oppenheimer, Peter, Real Time Design and Animation of Fractal Plants and Trees, *Computer Graphics*, 20, 4 (Aug. 1986), 56-64.

15. Preston, Kendall, and J. B. Duff, Modern cellular automata: theory and applications, Plenum, New York, 1984.

16. Prusinkiewicz, Przemyslaw, Aristid Lindenmayer, James Hanan, Developmental Models of Herbaceous Plants for Computer Imagery Purposes, *Computer Graphics*, 22, 4 (Aug. 1988), 141-150.

17. Reeves, William T., Particle Systems - A Technique for Modeling a Class of Fuzzy Objects, *Computer Graphics*, 17, 3 (July 1983), 359-376.

18. de Reffye, Philippe, Claude Edelin, Jean Francon, Marc Jaeger, Claude Puech, Plant Models Faithful to Botanical Structure and Development, *Computer Graphics*, 22, 4 (Aug. 1988), 151-158.

19. Sabella, Paola, A Rendering Algorithm for Visualizing 3D Scalar Fields, *Computer Graphics*, 22, 4 (Aug. 1988), 51-58.

20. Smith, Alvy Ray, Plants, Fractals, and Formal Languages, *Computer Graphics*, 18, 3 (July 1984), 1-10.

21. Upson, Craig, Michael Keeler, VBUFFER: Visible Volume Rendering, *Computer Graphics*, 22, 4 (Aug. 1988), 59-64.

22. Wyvill, Brian, Craig McPheeters, Geoff Wyvill, Data Structure for Soft Objects, *The Visual Computer*, 2, 4 (1986), 227-234.