Multiresolution Rendering of Complex Botanical Scenes

Dana Marshall
Donald S. Fussell
A.T. Campbell, III

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
Applied Research Laboratories
The University of Texas at Austin
1000 Burnet Road
Austin, TX 78758
Phone: +1-512-835-3743 Fax: +1-512-490-4220
email: dane@cs.utexas.edu

Abstract

This paper presents a system for rendering very large collections of randomly parameterized plants while generating manageable scene geometries for rendering. A given botanical description of a plant is compiled into a hierarchical volume approximation. This is then integrated into a multiresolution rendering system that uses adaptive volume refinement. For objects that are close to the viewer, explicit polygons are generated, while objects hidden or further away are rendered as groups of microsurfaces. This system can be extended to any polygon intensive rendering.

1 Introduction

In this paper, we present a method that efficiently models and renders scenes containing a large number of plants, each of which can differ macroscopically in form as well as in detail from all the others. Our target is scene models containing on the order of 100 million primitives if modeled using polygons or particles, and we prevent the actual synthesis and manipulation of such large databases. This method is intended for use in speeding the rendering times of high resolution imagery as well as generating simple models for low resolution applications such as motion previews.

Note that in this paper trees, nodes and children refer to data structures while plants, branches and leaves refer to biological structures. We first summarize previous work and give an overview of this work. We describe the data structure of hierarchical tetrahedra, discuss the representation we use for plant life and then describe how both are used in the modeling and rendering algorithm.

2 Previous Work

Previous work on generating and rendering plants includes fractals and L-systems [14], animating plant development [11],[12], topological details [3], and generating biological plants true to nature.[10],[9] In [4], plants were categorized according to the different structures including order, ramification, and phyllotaxy. These systems are good at generating accurate plant models and are not concerned with the problem of rendering the complex scenes that can be generated.

Systems have been implemented that render large collection of microsurfaces using voxels [5], texels [6] , or particles [13], but they don't address the transition from polygon to texel or polygon to particle. Other systems have managed to render scenes with a large amount of detail but only by repeatedly duplicating the representation of a small set of objects [13, 16]. One method has introduced multiresolution texels using octrees [7, 8]. One method [17] improved on this problem by using less complex copies for plants farther from the viewer.

3 Overview

The rendering process is one of adaptive refinement. The space within a scene is continually subdivided until image quality no longer improves with further subdivision. This occurs when a subvolume occu-

pies some user defined minimum area in the final image (say, one pixel), when the subdivision is empty, when the subdivision is largely obscured by other partitions, or when the subdivision is highly visible and contains so few primitives that furthur subdivision could not improve rendering speed. Subdivisions that are small or largely hidden are rendered as an approximation of the surfaces that they contain, while the contents of visible volumes are explicitly modeled and rendered.

Space is partitioned using hierarchical irregular tetrahedra. This partitioning creates a binary tree than can be traversed in a manner similar to BSP trees. A tetrahedron does not represent a partitioning of a space by the polygons of a particular scene, but may contain any number of polygons. The contents can then be rendered directly, or the tetrahedron itself can be rendered as an microfacet-based approximation of the contents. Tetrahedra can be subdivided in many ways, thus allowing the scene to be subdivided according to the complexity the viewer can see, and/or the complexity of the scene itself.

We also take advantage of a procedural, volumetric parameterized plant model that allows us to use an approximation of a plant's or sub-plant's estimated shading attributes, volume and surface area to render a plant or part of a plant as a collection of microsurfaces without calculating and sampling the explicit polygons that would be generated.

This method allows us to use the space traversal methods of BSP trees without generating the enormous BSP tree that a complex scene would spawn. It enables us to take advantage of a hierarchical bounding volume structure without overlapping volumes, and allows us to use a more adaptive subdivision method than is possible with octrees.

4 The Tetrahedral Model

Our basic scheme begins with a tetrahedral volume as a first approximation to an object, which is then recursively refined as needed. The tetrahedron has the advantages that it can be adaptively subdivided according to the complexity of the scene, and the resulting partitioning readily translates to Gouraud shaded triangles.

The refinement process subdivides tetrahedra that constitute a scene. A tetrahedron divided by a plane can result in 2,3,4, or 6 smaller tetrahedra depending upon whether the plane intersects one, two, three, or four edges, respectively. (See Fig. 1) There is some freedom in partitioning the resulting halves if they are not tetrahedra themselves. One possibility would be to sample the scene and subdivide according to model complexity. Our renderer, however, simply chooses a subdivision depending on the aspect ratio of the resulting tetrahedra in order to avoid sliver subvolumes.

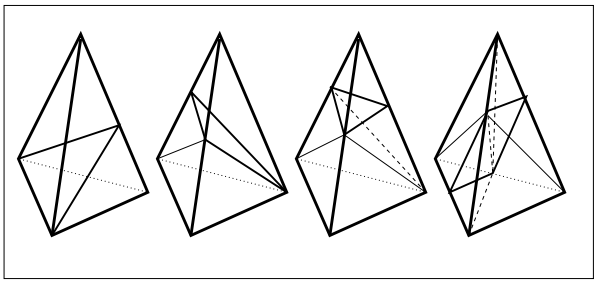

Figure 1: Cases of tetrahedron partitioning

If a plane intersects four edges of a tetrahedron, it is possible, by halving the two unintersected edges, to subdivide a tetrahedron into eight children. This generates a balanced tree and further avoids sliver tetrahedra. (See Fig. 2)

The volume of each tetrahedron can be calculated easily from Herron's formula [1] and the equations of the tetrahedron's bounding planes. The resulting data structure is a binary tree with tetrahedra at some of the interior nodes and all of the terminal nodes. Interior nodes contain a plane which divides the space of the child nodes. This allows a front to back or back to front traversal such as can be achieved using BSP trees.

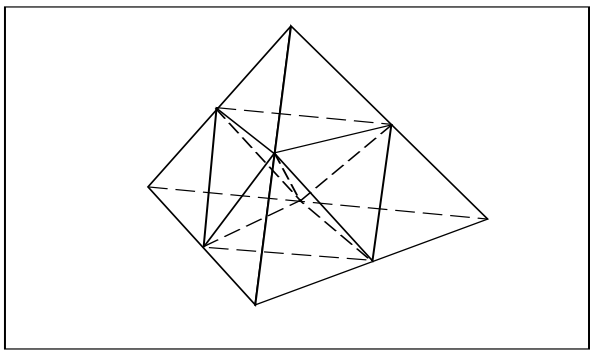

Figure 2: Canonical tetrahedron partitioning

When rendering the model, either the primitives contained within a tetrahedron, or the tetrahedron itself, may be drawn. To render a tetrahedron as a translucent volume approximating a collection of

microsurfaces, note that any tetrahedron projects to at most 4 triangles on the viewing plane. (See Fig. 3) Illumination can be sampled at all vertices and given a shading function, density information, and tetrahedron thickness, the resulting triangles can be Gouraud shaded.

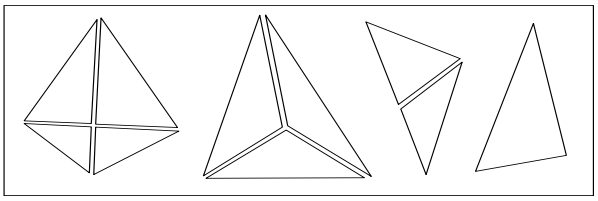

Figure 3: Tetrahedral projections

5 Plants

The method described in this paper can be used to speed the rendering of any large number of polygons. It also allows us to avoid the creation of many of those polygons to begin with. Although our method of plant generation is not the subject of this paper, we were able to take advantage of the algorithmic nature of plant structures in order to defer as much polygon generation as possible.

We describe individual plants by specifying the characteristics of each order branch. An order 0 branch corresponds to the trunk, order 1 to the next largest branches, etc. Attributes for each order include length, thickness, orientation, leaf color, branch color, number of leaves and branches. Additional information include leaf orientation heuristics and a pointer to a polygon list modeling a sample leaf/leaves.

Bounding volumes are calculated for the highest order branches by comparing the locations obtained by rotating the longest possible branch by the largest angle. This bounding volume information is used to calculate the extremes of the next lower order branches. Bounds are generated for all species of plants and sub-plants before any modeling or rendering is done.

Average surface area and color/reflectivity information for each order is collected as well. The surface area of the sample leaf/leaves is measured and an estimated total surface area and sample orientation is generated for each order branch. This allows any branch and its child branches to be treated as an individual plant during visibility and rendering estimation.

Actual instances of plants are unique collections of pseudo random choices of the branch attributes. The minimal information needed to add a plant to the scene is the type and location of the plant and a random number seed. The values of the lowest order branches are generated in set sequence so that the same plant is created every time. This ensures that the same scene will be modeled every time, down to the last leaf. Parts of a scene that may travel out of view may be discarded without concern because they can be re-generated if needed.

If polygons are needed, we generate cylinders for the branches and replicate variations of a leaf modelled by the user, positioning according to length, thickness, orientation and color ranges given in the plant's description. The transition from this coarse modeling to a closer detail concerning smooth bifurcations, texture and bump mapping is given in [3]. This gives the user several scales to model in. Explicit polygons can be included in the scene and they are either drawn or their attributes are added to a subvolume's summary. Plants can be described exactly, or more general descriptions can be used that allow the algorithm to generate random versions with varying colors. Lower orders of plants can be modelled to give a general shape to a plant and the computer can fill in the rest of the details. Locations of plants themselves can be allocated automatically using terrain following algorithms or can be placed explicitly by the user.

6 Scene Subdivision

6.1 Preprocessing

Initially the algorithm inputs the list of plants and polygons that describe the scene and the viewing pyramid is divided into two tetrahedra. Space outside the pyramid is also represented so that plants which fall outside of the range of view have a home somewhere in the data structure. If any light sources fall outside of the viewing pyramid, the space between them also needs to be refined in order to calculate accurate shadows. Explicit (previously modeled) polygons are then divided among the original tetrahedra. As the algorithm progresses and these tetrahedra are subdivided, the polygons are divided among the child nodes and clipped if necesary, eventually being drawn if their eventual partition is rendered explicitly, otherwise their shading attributes are added to the tetrahedron's approximation.

6.2 Main Loop

The main body of the routine is a three step loop. The first is the visibility step, in which the contribution of each tetrahedron to the final image is approximated. The second step is the refinement step, in which tetrahedra are subdivided if needed or possibly flagged for explicit polygon generation. The third is the update step, in which the contents of the subdivided tetrahedra are spread among its new children, the children's rendering attributes are updated accordingly, and the contents of tetrahedra which are so flagged may be modelled explicitly into polygons.

6.3 Visibility

A low resolution image is rendered of tetrahedra only - all polygons are rendered as their microsurface approximation for speed. The binary tree is traversed and the scene is rendered starting from front to back so that transparency values can be accumulated in an alpha buffer along the way. In this way we get rough values for 1) the total number of pixels that each tetrahedron affects in the final image, and 2) the opaqueness of the scene in front of each tetrahedron, thus given us the total visibility of each tetrahedron.

6.4 Refinement

We traverse unmodeled terminal tetrahedra and given their contents and visibility decide if they need subdivision, or flagging for future modelling.

Tetrahedra that will not benefit from furthur subdivision are flagged for polygons. This includes small tetrahedra close to the viewer and tetrahedra that contain few polygons.

Otherwise, if the overall contribution of a tetrahedron to the final image is large then it is subdivided. This would include medium sized tetrahedra that are very visible and tetrahedra that are larger but are partially hidden by other parts of the scene. Any extant polygons in these tetrahedra are divided among the children. The contribution size cutoff is set by the user. For high resolution images the ideal size would be one pixel or less. We found in practice higher sizes still produced good images without any subdivision artifacts showing.

This tetrahedra that remain are small in relation to the final image or are tetrahedra that are hidden and both will be rendered as approximations.

6.5 Update

Contents of newly subdivided tetrahedra are divided among the new children and shading characteristics are updated. Polygons are clipped against the sub-tree's clipping planes and then their fragments are sorted among the children.

A plant may be stored in a tetrahedron in three ways of ascending detail and memory usage. When a plant or sub-plant's bounding volume falls within a tetrahedron it will be added to that tetrahedron's plant list. Individual branches may be stored by saving their endpoints and enough information to generate the branch and its leaves. And plants can be modeled into explicit polygons and stored in the tetrahedron's polygon list.

Plants are sorted according to each individual plant's characteristics. A plant's bounding volume is compared against the top level partitioning plane. If it is entirely on one side or the other then the routine recurs on the lower level tetrahedron. If the bounding volume is divided then the algorithm calculates the position of the high order branch(es) and their corresponding subplants. The subplant's bounding volumes are compared against the partitioning plane as if they were an individual plant and the algorithm recurs.

The newly calculated branch positions are also compared against the partitioning plane. If the endpoints are on either side then the branch is clipped by the partitioning plane and the two half branches recur on lower level tetrahedra.

Once a terminal tetrahedron is reached then plants are added to that tetrahedron's plant list, branches are added to that tetrahedron's branch list, polygons are added to that tetrahedron's polygon list and the shading information of that tetrahedron is updated accordingly. Total surface area, color and sample orientation are all updated with the arrival of each new plant, branch, or polygon, weighted according to the contributing amount of surface area.

When the location of a plant ends up in a subdivision that has been flagged for explicit polygon generation, all of it's offspring branches and leaves are modelled and stored. In order to minimize memory requirements, this modelling can be defered until final image rendering.

7 Rendering

After the final update step, there is a shadow pass in which the scene is rendered from the viewpoint of

the individual light sources and illumination samples are taken at the set of key points for each tetrahedron or polygon. The rendering is done front to back with volume shadow information being collected in a gray scale image buffer and polygon shadows being collected in a z-buffer of proportional size to the image resolution. Once light values have been collected for all tetrahedra and polygons, the final image is rendered back to front.

If tetrahedra are rendered as an approximation of their contents the equation used is one to simulate the rendering of a collection of microsurfaces comprising a low albedo volume [6, 2].

If $\rho(s)$ is the density at a given position in space $x(s), y(s), z(s)$ and r is a conversion factor from density to light attenuation, the total transparency of a ray from $tnear$ to t is given by:

$$T = e^{-r \int_{tnear}^{t} \rho(s) ds}$$

And the brightness of a ray is:

$$B = \int_{tnear}^{tfar} T \sum_{i} I_i(t) p \cos(\phi) \rho(t) dt$$

Where T is the transparency and $I_i(t)$ is the ith light source arriving at position t along the ray. The light is multiplied by the phase factor $p \cos(\phi)$ which approximates the light reflected when striking a spherical particle. (The factor p an adjustable shading parameter) In our system we assumed the density ρ constant and the illumination $I(t)$ linear throughout the tetrahedra to allow the integral to be evaluated as a summation.

In [6], the phase factor $p \cos(\phi)$ was replaced by a reflection function derived from a sample plane orientation to approximate the contents of a texel:

$$p \cos(\alpha) + s \cos(\beta)^{spexp}$$

where α is the angle between the sample normal and the light source, β is the angle between the view vector and the reflection of the sample normal and p, s, and $spexp$ are user specified shading parameters.

Using the original particle model equation was not really satisfactory due to a lack of plane dependent specular highlights. The plane equation, however, is primarily useful if all elements inside a subdivision are of like direction, so a weighted average of the two was used, the weight determined by the resulting length of the average normal. The average plane orientation was determined by the variation of plane orientations within a tetrahedron. An entirely random collection of plane orientations would be approximated by the particle model equation and an entirely uniform collection would be approximated by the sample plane rendering equation.

8 Results

8.1 Test Scenes

Figure 4 shows the catalog of flora used for this project. Figure 6 is a test image. It took 21 minutes to model and refine and 6 minutes to render in software on a SPARC Center 2000E. Shadows were not mapped to polygon surfaces but illumination was sampled only at polygon vertices. Thus the shape of the shadows on the ground plane reflects the illumination of the vertices of the resulting polygons' subdivision. The model is a row of eighty trees which if expanded would contain 3.79 million polygons. The image contains 784049 polygons and 2053 (out of 15857 generated) tetrahedra. Figure 5 shows a top view of the model generated for this scene. The viewer is positioned at the left where one can see parts of the model clipped off due to the hither plane and viewing pyramid. As one looks to the right, one can see fewer polygons generated and coarser volume subdivision. Figure 8 shows the refine sequence. We generated an image for each of the latter steps in the refine procedure.

Figure 9 shows a larger model. Haze was added to the view. It took 68 minutes to model and refine and each image took 28 minutes to render. The image has 9.4 million polygons and 18512 (out of 108217 generated) tetrahedra. The model is generated from 8636 plants which, if fully expanded, would make 78.2 million polygons.

8.2 Animation

Animation results are promising. For any individual frame the number of subvolumes that need to be divided is small. We created an animation by moving the viewer location in the test scene. While the initial frame of the test scene takes 21 minutes of modeling and 6 minutes of rendering time, subsequet frames took on average 6.64 minutes apiece, with only 40-70 additional volumes subdivided each frame. In addition, the subdivision criteria can be set to generate coarser images for doing test scenes. In these scenes the transition between tetrahedron and polygons can be seen, but the frame rate increses proportionally.

8.3 Memory

The biggest demand for memory is the number of generated polygons. Given the large polygon per pixel ratio for images such as these even with 100% accurate hidden polygon elimination there is on order of 4 million polygons for a modest image (say 512×512 with 4 samples per pixel).

This can be avoided in still images by saving any explicit polygon generation until the rendering step. Polygons can be generated as each tetrahedron is drawn and then discarded. This shifts the greatest memory demand to the storage of tetrahedra, plants, and branches - an order of magnitude less.

Shadow information must be calculated from tetrahedra instead of polygons, but increasing the density of spatial subdivision gives satisfactory results. Figures 6 and 7 were generated using both methods. In Figure 6, polygons were calculated and stored whenever a tetrahedron was flagged for polygon generation. In Figure 7, all polygon modeling was deferred until the final rendering pass. This method includes all of the modeling in the final rendering step which increases the time counted for the final rendering step but the overall calculation time is the same.

9 Conclusion

Large collections of flora contitute one of the most complex and time consuming collections of models to render. We have developed a procedural volumetric plant model which is designed to allow very large collections of plants to be rendered using multiresolution space partitioning techniques to avoid dealing with prohibitively large databases. Our modeling technique combines desirable properties of bounding volumes and the subspace orderings provided by BSP trees as part of the model itself. Future work includes developing heuristics for scene dependent splitting plane selection and use of hierarchical volume subdivision for calculation of global illumination.

10 Acknowledgements

I would like to thank David Brant and ARL for use of its computing facilities. Thanks also go to GradyAlan Inc., Dan Watkins, Denice Goldshmidt, Arthur Marx, Mrs. Miller, Mr. Bryant, and, of course, Don Fussell.

References

[1] Beyer, William H., Standard Mathematical Tables, 24th Ed. 1976, CRC Press, P. 7.

[2] Blinn, J.F., "Light Reflection Functions for Simulation of clouds and Dusty Surfaces", Computer Graphics, Vol. 16, No. 3, 1982, pp. 21-29.

[3] Bloomenthal, J., "Modeling the Mighty Maple", Computer Graphics, Vol. 19, No. 3, 1985, pp. 305-311.

[4] de Reffye, Philippe, Edelin, Claude, Francon, Jean, Jaeger, Marc, and Puech, Claude, "Plant Models Faithful to Botanical Structure and Development", Computer Graphics, Vol. 22, No. 4, 1988, pp. 151-158.

[5] Greene, Ned, "Voxel Space Automata: Modelling with Stochatic Growth Processes in Voxel Space", Computer Graphics, Vol. 23, No.4, 1989, pp. 175-184.

[6] Kajiya, James T., "Rendering Fur with Three Dimensional Textures", Computer Graphics, Vol. 23, No. 3, 1989.

[7] Neyret, Fabrice, "A General and Multiscale model for Volumetric Textures", Graphics Interface '95.

[8] Neyret, Fabrice, "Synthesizing Verdant Landscapes using Volumetric Textures", Institut National de Recherche en Informatique et en Automatique, RR-2846, March 1996

[9] Oppenheimer, Peter E., "Real Time Design and Animation of Fractal Plants and Trees", Computer Graphics, Vol. 20, No. 4, 1986, pp. 55-64.

[10] Prusinkiewicz, Przemyslaw, Lindenmayer, Aristid, and Hanan, James, "Developmental Models of Herbaceous Plants for Computer Imagery Purposes", Computer Graphics, Vol. 22, No. 4, 1988, pp. 141-150.

[11] Prusinkiewicz, Przemyslaw, Hammel, Mark S., and Mjolsness, Eric, "Animation of Plant Development", Computer Graphics, Vol. 27, No. 3, 1993, pp. 351-360.

[12] Prusinkiewicz, Przemyslaw, James, Mark, and Mech, Radomir, "Synthetic Topiary", Computer Graphics, Vol. 28, No. 3, 1994, pp. 351-358.

[13] Reeves, William T., "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems", Computer Graphics, Vol. 19, No. 3, 1985, pp. 313-322.

[14] Smith, Alvy Ray, "Plants, Fractals, and Formal Languages", Computer Graphics, Vol. 18, No. 3, 1984, pp. 1-10.

[15] Thompson, Karl K., "Ray Tracing with Amalgams", Dissertation for The University of Texas at Austin, 1991.

[16] Ward, Gregory J., "The RADIANCE Lighting Simulation and Rendering System", Computer Graphics, Vol. 28, No. 3, 1994, pp. 459-472.

[17] Weber, Jason and Penn, Joseph, "Creation and Rendering of Realistic Trees", Computer Graphics, Vol. 29, No. 3, 1995, pp. 119-128.

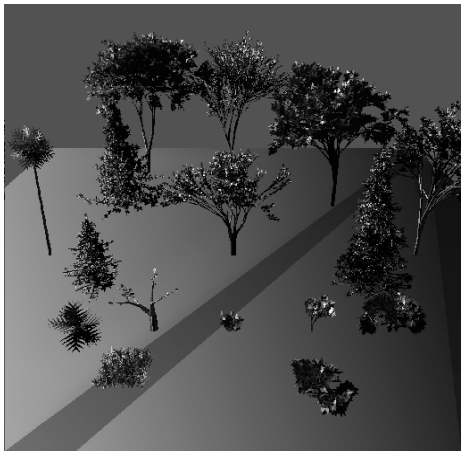

Figure 4: Plant Calalog

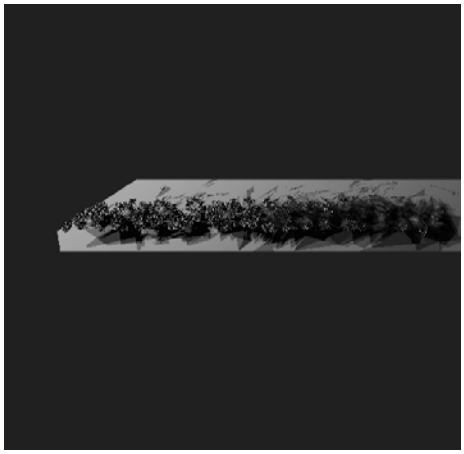

Figure 5: Test Scene - Top View

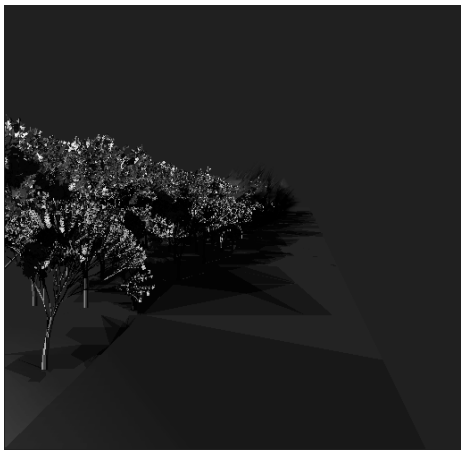

Figure 6: Test Scene - High Memory

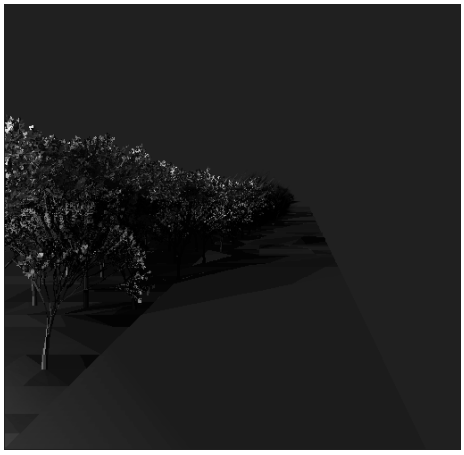

Figure 7: Test Scene - Low Memory

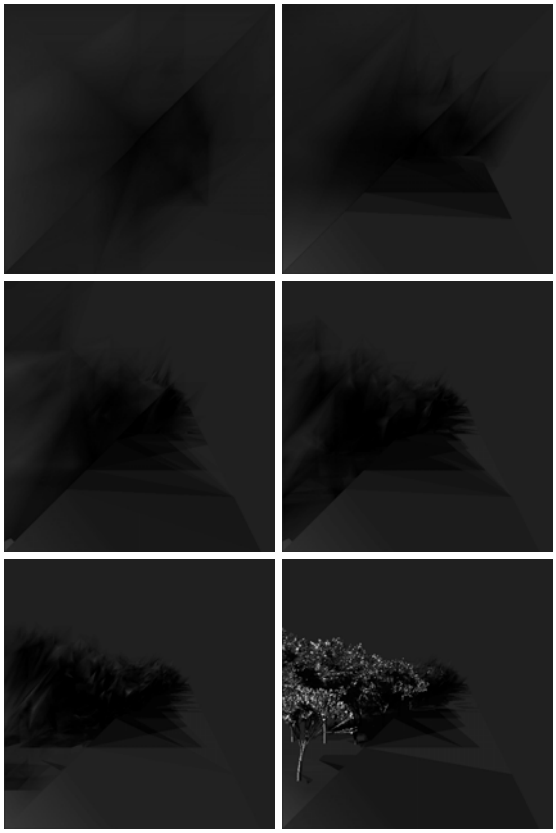

Figure 8: Refine Sequence

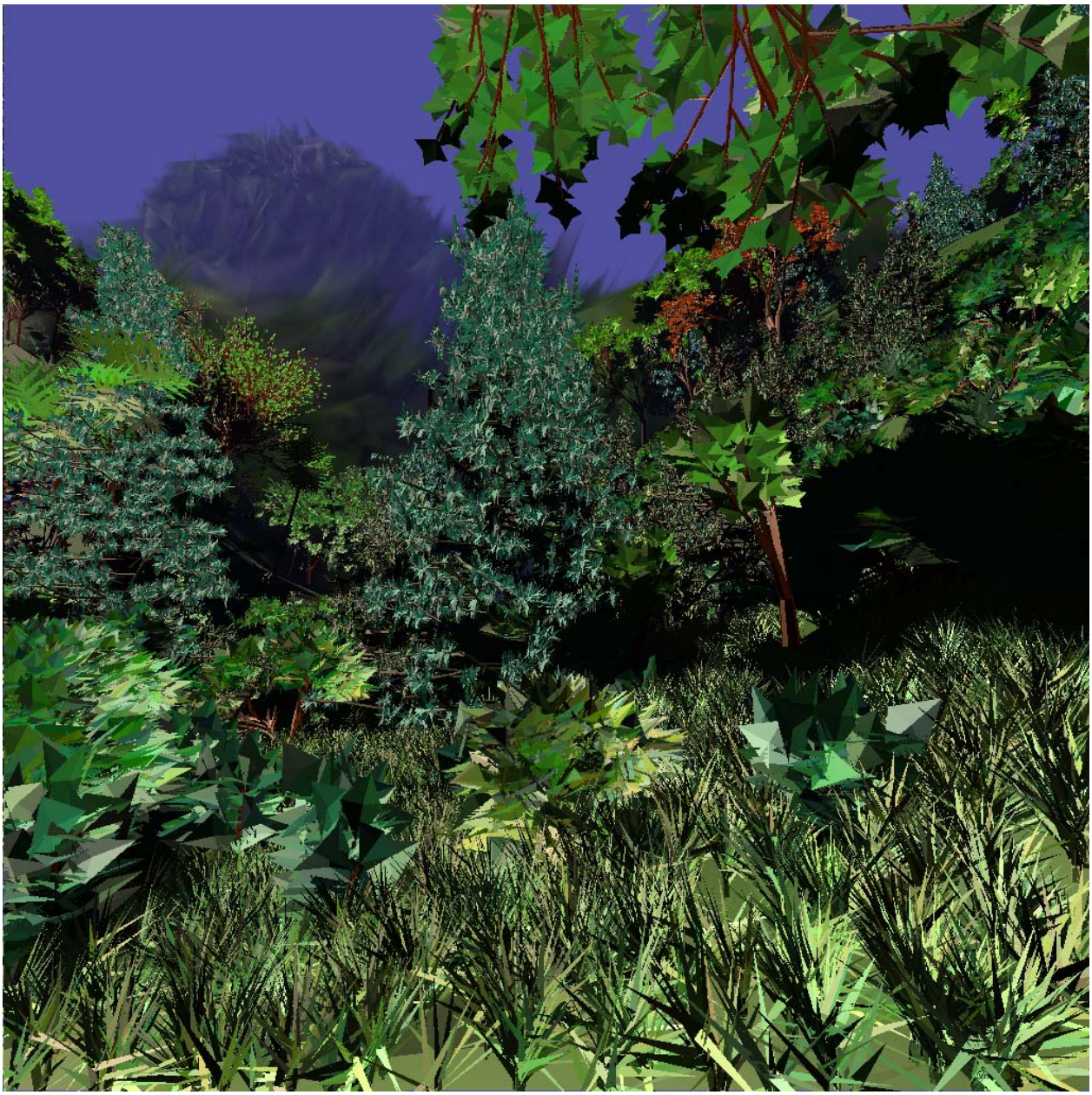

Figure 9: Complex Scene