231063

UCRL-JC-123366
PREPRINT

Hierarchial Rendering of Trees from Precomputed Multi-Layer Z-Buffers

N. Max

This paper was prepared for submittal to the
Special Interest Group on Computer Graphics '96
New Orleans, LA
August 4-9, 1996

February 1996

Lawrence Livermore National Laboratory

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Hierarchical Rendering of Trees from Precomputed Multi-layer Z-Buffers

Nelson Max

University of California, Davis

Abstract

Chen and Williams [2] show how precomputed z-buffer images from different fixed viewing positions can be reprojected to produce an image for a new viewpoint. Here images are precomputed for twigs and branches at various levels in the hierarchical structure of a tree, and adaptively combined, depending on the position of the new viewpoint. The precomputed images contain multiple z levels to avoid missing pixels in the reconstruction, subpixel masks for anti-aliasing, and colors and normals for shading after reprojection.

Introduction

The goal of this paper is to efficiently create images of botanical trees, which will look realistic from a distant viewpoint as well as one inside the tree. The standard rendering method for complex scenes involves a graphics pipeline which processes shaded polygons, perhaps with texture mapping. For a complex object like a tree, hundreds of thousands, or even millions, of polygons may be required. Weber and Penn [11] achieve detail degradation with viewing distance by gradually reducing the number of leaves and twigs rendered. Maciel and Shirley [6] suggest suddenly switching to a texture mapped "imposter" polygon for very distant views. In this paper I use a hierarchy of precomputed parallel projection z-buffer images of twigs and branches, corresponding to the hierarchical structure of the tree. The smaller subbranches and twigs are represented at higher absolute object space resolution. This allows the appropriate precomputed images to be selected for reprojection based on the level of detail required at the current viewpoint.

There has been recent interest in generating images from an arbitrary viewpoint by reprojecting images precomputed for, or photographed from, a small collection of specific viewpoints. If depth as well as color information is available at each input pixel, one may invert the viewing projection to find the corresponding 3D surface point, and then reproject it onto the image plane for another viewpoint. Chen and Williams [2] have done this for computer rendered images, where the necessary depth information is available in a z-buffer. They use several nearby precomputed input images to reconstruct the desired output image, because in the presence of occlusion, no single precomputed view can be expected to contain all the surface points which should be visible.

McMillan and Bishop [7] used cylindrical images, resampled from multiple video frames, in order to achieve the same effect. Their paper concentrates on fitting the parameters necessary to produce a cylindrical image from a video pan, and to specify the correspondence and disparity between two such images. Currently, they process a single color/disparity input image in near real time, by scanning the input image in a pattern which guarantees a back to front painting of the output pixels, so that a z-buffer is not needed. However, they are not yet able to use other images to fill in the data that may be occluded in the single input image. Further references on reprojection appear in [2], and [7], and also in [5], which is a report on an earlier version of this work, without the hierarchies, and without antialiasing.

Algorithm Summary

My approach differs from the work of [2] and [7] in several ways:

1) I transform each pixel independently, instead of using a region-based warping method. For an object like a tree with high visible depth complexity, adjacent input pixels often come from disjoint surfaces, so that region-based warping is inappropriate.

2) I use subpixel coverage masks to achieve antialiasing.

3) I use a hierarchical model for the tree, and precompute views for each level in the hierarchy, with the lower levels having higher resolution. During rendering from a particular viewpoint, I traverse the hierarchy, and decide the appropriate level at which to reproject each node, visiting child nodes if the parent's reprojection would be too coarse.

4) I use parallel rather than perspective views for the precomputed images. This slightly simplifies the resampling computation, and allows the precomputed images to be sampled on the 2-parameter surface of the unit sphere of viewing directions, rather than in the 3-parameter volume of possible viewpoints.

5) I use multiple z layers in each precomputed view, and also in the reprojected image. The extra data in the precomputed views helps fill in surface points which might not be visible as the front-most surface in any of the input views. The layers in the reprojected image are combined, using their respective subpixel masks, to provide a supersampled result for anti-aliasing the final image, as in the A-buffer method of Carpenter [1].

6) I store a surface normal at each precomputed pixel, and do shading after reprojection. This permits the illumination to be changed, and allows the subbranches of the tree to be repeated within the hierarchy at different positions and orientations. It also permits post-process shadow computations.

Basic Data Structure

An image is formed at multiple z layers. For each layer at a pixel, I store a surface color, a normal, a depth, a subpixel mask, a hit count, and a leaf bit. A leaf surface with the leaf bit set is given translucent shading if it is back lit, while other surfaces are considered opaque. During ren-

dering or reprojection, the color and normal components are accumulated as additional points are mapped to the same layer at a pixel, and the hit count is incremented. At completion, the color components are divided by the hit count, and the normals are normalized. For brevity below, I will refer to either the memory or disc representation as an M-buffer.

In memory, I keep the color and normal components as floating point values. In the disc files, the color components are stored in one byte each, and the normal is encoded as 7 bits latitude, and 8 bits longitude, leaving space for the one leaf bit. The depth is kept in floating point, and the hit count is no longer necessary. On disc, the depths at all non-empty layers are stored consecutively in pixel order in one array, with the empty background depths compressed away. The color and normal arrays are similarly compressed. A full sized array of one byte per pixel counts the number of occupied layers at the pixel, and is used to reestablish the pixel positions when the data is read back. Thus the empty sky in figures 3 to 7 occupies only one byte per pixel, so the compressed files can be read and processed quickly.

Basic Reprojection Algorithm

For an orthogonal z-buffer input view, the z value comes directly from the stored depth value, and the x and y values are determined from the pixel indices by an affine viewport to window transformation [8]. These 3D coordinates are transformed to the desired view using a standard 4 by 4 affine transformation matrix [8]. They are then reprojected into the new z-buffer, using either orthogonal or perspective projection. (The normal vectors are multiplied by only the 3 by 3 rotation part of the 4 by 4 transformation matrix.)

I use a 16-bit 4 by 4 subpixel coverage mask, which represents a considerable savings over brute force supersampling. First of all, I assume that the input and output z values for the resampling transformation are constant, so that subpixel displacements in the output mask involve mul-

tiplication by only a 2 by 2 matrix. In addition, only a single normal is transformed, and only a single color is transferred. The 16 subpixels of an input point map within a 3 by 3 square of pixel positions in the output raster, and are accumulated first into 9 temporary pixels. Then the non-empty temporary masks (at most 5) are combined into the output layers, so that fewer depth comparisons are required.

The depth comparisons use a depth discrimination tolerance ε, which may vary with viewing conditions and within the object hierarchy, but is constant for each reprojection pass. The n depth layers are stored in front-to-back order. If a new depth $znew$ is within ε of one of the existing layers, its mask is *or*ed with the mask already present, and the colors, normals, and depth are averaged (actually accumulated, as described above). If the $znew$ is within ε of two adjacent layers, the closest is chosen. If no sufficiently close layer is found, and $znew$ is closer than the nth layer (or if there are less than n layers present), a new layer is created by shifting the farther layers back to make space, perhaps discarding the farthest one.

The Hierarchy

Hierarchical models are particularly efficient in data specification, since repeated instances of a subobject can be repositioned and reoriented with respect to a parent node using a 4 by 4 affine transformation matrix. This is appropriate for trees, which can repeat basic leaf and subbranch structures. The hierarchy is an abstract tree, in direct correspondence with the botanical tree it represents. The root node is the whole scene description, which may be a single tree, or a scene with multiple trees. Leaf nodes represent the botanical leaves, formed from color/transparency textured polygons obtained by scanning actual leaves. Internal nodes may point to child nodes related by an affine transformation, and may also contain branches and twigs, formed as Bezier tubes, whose center paths and radii are defined by Bezier curves.

The object description of a node is given in an ascii file, which is parsed by the simplified psuedocode given in figure 1. Here `T` stands for the current transformation, initially the identity, and `*` is matrix multiplication. (Row vectors are multiplied by matrices on their right.) The file `nextfile` contains a child node object description, and `nextfile.latitude.longitude` is one of its M-buffer disc files, imaged from the viewing direction (latitude, longitude). Note that `Parse` is called recursively if the viewpoint is too close to a child node object for reprojection to be accurate, in particular, if subpixels in the output image mask might be missed between the reprojected input supbixels. Since reprojection will be adequate for some nodes, and others outside the view volume may be culled, only an initial subtree of the hierarchy will be traversed for any particular viewpoint.

Results

The videotape shows two sequences, at 360 by 270 resolution, in which a tree is approached along a spiral path, demonstrating both translation and rotation effects. The first sequence is of a maple tree in summer, and the second sequence shows two maples with fall colors. [Note to reviewers: the shading glitch on the bark and trunk in the first sequence was corrected in the second sequence.] These were constructed by resampling a 6 level hierarchy, as shown in figures 2 through 7. Figure 2 shows the scanned leaf, figure 3 has a twig with four leaves, and the succeeding figures show successively larger structures formed from the proceeding ones, culminating in the complete tree in figure 8. The tree has over 1.3 million polygons, mostly on the Bezier tubes forming the branches, and took 30 minutes on an SGI Indigo to render without reprojection, using the software version of the subpixel mask A-buffer algorithm which is part of my system. With reprojection, it took only five minutes on the same computer. Each of the intermediate structures was precomputed for 22 parallel projection directions on the unit sphere: one at each of the poles, 8 along the equator, and 6 along the 45 degree north and south latitude circles. Figures 3 through

```
Parse (file infile, M_buffer B)
   while( (command = read_string(infile)) != EOF)
      if (command == "transform")
         P = read_matrix(infile)
         T = P * T
      if (command == "{") push T onto matrix stack
      if (command == "}") pop matrix stack onto T
      if (command == "polygon")
         D = read_polygon_data (infile)
         scan convert D onto B using T
      if (command == "limb")
         E = read_Bezier_tube_data (infile)
         polygonalize E and scan convert onto B using T
      if (command == "use")
         nextfile = read_string (infile)
         G = read_bounding_box(nextfile.box)
         transform G to bounding box H using T
         if ( H intersects view volume)
            if (projected size of closest pixel in H < threshold)
               for each nearby (latitude, longitude) precomputed view
                  C = read_M_buffer (nextfile.latitude.longitude)
                  Q = inverse_viewing_matrix (latitude, longitude)
                  R = Q * T
                  reproject C onto B using R
            else Parse (nextfile, B)
```

Figure 1. Psuedocode for traversing the hierarchy.

7 show representative precomputed images at the same scale of 90 pixels per inch. For a new viewing direction near the equator, the four nearest (latitude, longitude) views were reprojected. Near the poles, only three were used, since the poles have no longitude.

I determined by timing tests that it actually slowed things down to reproject the twig in figure 3, because it contains so few polygons. Resampling is intrinsically slower per pixel than scan conversion, because it cannot take advantage of area coherence or use incremental calculations. However it is suitable for hardware implementation, because the independent samples can be processed in parallel. The 22 versions of each of the structures in figures 4 through 7 required 66 minutes for an SGI Indigo 2 to precompute, and took 232.7 megabytes of disc storage. I used 4 depth layers in the M buffers.

Figures 9 and 10 show additional frames from the autumn color animation sequence. The last 240 frames of this animation took a total of 23.4 hours to compute on an SGI Indigo 2. Figure 8 took the longest, 10 minutes, because it contained many structures that needed to be reprojected at intermediate resolution. Figure 9 took only 2.5 minutes, since many off-screen nodes were culled, and figure 7 also took 2.5 minutes because only the highest objects in the hierarchy were resampled. (The Indigo 2 is twice as fast as the Indigo.) Figures 11 and 12 show the same tree with green summer colors, and figure 13 shows two maple trees in fall. Figures 8 through 13 are all frames from the video.

Discussion and future work

Because I am doing post-process shading, it is straightforward to include a shadow buffer algorithm (Williams [12]), which I have done in the earlier version [5] of this work. With my current A-buffer scheme, I could use subpixel masks and multiple z layers in the shadow buffer to create higher effective resolution, without storing a floating point depth at each supersampled

pixel. However, in the current application, a single shadow buffer will not be of adequate resolution for a viewpoint very close to a leaf, and obvious shadow-edge aliasing would result. The "percentage closer filtering" of Reeves *et al.* [10] can turn large jaggies into large blurs, but these will still be inappropriate if another nearby leaf is casting the shadow. Some form of adaptive multi-layer, multi-resolution, shadow buffer algorithm may be able to give realistic shadows in this situation, perhaps organized or assisted by the resampling hierarchy described above. Since the surface normal is stored, the depth comparison bias used to keep surfaces from shadowing themselves could vary with the surface normal, as suggested by Grant [3].

The transitions between levels of detail may be visible in the videotape. I plan to apply a "screen door" dissolve on the subpixel masks to achieve a smoother transition, using frames where both the reprojection of a node and its explicit traversal are represented. The dissolve will be done per node, so different nodes can be at different stages in their transitions in the same frame.

When the depth discrimination threshold ε is small, and there are many subpixel objects, like the leaves in figure 7, the n depth layers are often used up before the union of their subpixel masks covers up the pixel, so the sky color leaks through inappropriately. If I had organized a linked-list structure for the layers, I would not need a uniform limit n, but execution would be a little slower, and more complicated for hardware implementation. Instead, I set ε to a larger value for distant views with high depth complexity. This sometimes caused the back and front surfaces of the branches to add together, giving an incorrect average normal [which caused the shading glitch in the first video sequence]. I solved this by not adding in reprojections whose transformed normals are back facing, except on the leaf surfaces, distinguished by the leaf bit, where both sides should be visible. Further work is needed to optimize the choice of ε, depending on the model dimensions and complexity, and the viewing parameters.

The interior of a tree appears dark because there is less sky illumination as well as less direct sunlight. Patmore [9] and Greene and Kass[4] take sky illumination into account, and it should be possible also to do this using my reprojection algorithm, either by directly accessing the precomputed images from all upward directions during the shading step, or by precomputing the sky illumination within the tree volume.

I have allowed for texture coordinates in the Bezier tubes representing the tree limbs, but have not yet added bark texture. Since I reproject both normal and depth values, bump mapped and/or displacement-mapped bark should also be possible.

This algorithm should be applicable to other cases, specifically to architectural models using repeated substructures like columns, windows, and whole houses in a subdivision. Architectural renderings could then include both buildings and landscape trees.

Acknowledgments

This work was performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under contract W-7405-ENG-48, with the preliminary work in [5] funded by the Hokkaido Overseas Guest Researchers Invitation Program. I wish to thank Przemyslaw Prusinkiewicz, Roger Crawfis, Eric Chen, Rebecca Springmeyer, Keiichi Ohsaki, and Gary Bishop for useful conversations, Cliff Stein for debugging help, and Jan Nunes for scanning the leaves and Eugene Cronshagen, Ross Gaunt, and Jan Nunes for recording the videotape.

References

[1] Carpenter, Loren, "The A-buffer, an Antialiased Hidden Surface Method, "Computer Graphics Volume 18, Number 3, July 1984 (Siggraph '84 Proceedings) pp. 103 - 108.

[2] Chen, Shenchang Eric, and Lance Williams, "View Interpolation for Image Synthesis," Computer Graphics Proceedings, Annual Conference Series, 1995, pp. 279 - 288.

[3] Grant, Charles, "Visibility Algorithms in Image Synthesis," Ph. D. dissertation, University of California, Davis (September 1992).

[4] Greene Ned, and Michael Kass, "Approximating Visibility with Environment Maps," Apple Computer Technical report #41 (November 1994).

[5] Max, Nelson and Keiichi Ohsaki, "Rendering Trees from Precomputed Z-buffer Views," in Rendering Techniques '95, (Hanrahan and Purgathofer, eds.) Springer, Vienna (1995) pp. 7 - 81 and p. 359.

[6] Maciel, Paulo, and Peter Shirley, "Visual Navigation of Large Environments Using Textured Clusters", Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics (April 1995), pp. 95 - 102.

[7] McMillan, Leonard and Gary Bishop, "Plenoptic Modeling: An Image-Based Rendering System," Computer Graphics Proceedings, Annual Conference Series, 1995, pp. 39 - 46.

[8] Newmann, William, and Robert Sproull, "Principles of Interactive Computer Graphics, second edition," McGraw Hill, New York (1979).

[9] Patmore, Chris, "Illumination in Dense Foliage Models," Proceedings of the Fourth Eurographics Workshop on Rendering, 1993, pp. 63 - 71.

[10] Reeves, William, David Salesin, and Robert Cook, "Rendering Antialiased Shadows with Depth Maps," Computer Graphics Vol. 21, No. 4 (July 1987) pp. 283 - 291.

[11] Weber, Jason and Joseph Penn, "Creation and Rendering of Realistic Trees," Computer Graphics Proceedings, Annual Conference Series, 1995, pp. 119 - 128.

[12] Williams, Lance, "Casting Curved Shadows on Curved Surfaces," SIGGRAPH '78 Conference Proceedings, pp. 270 - 274 (1978).

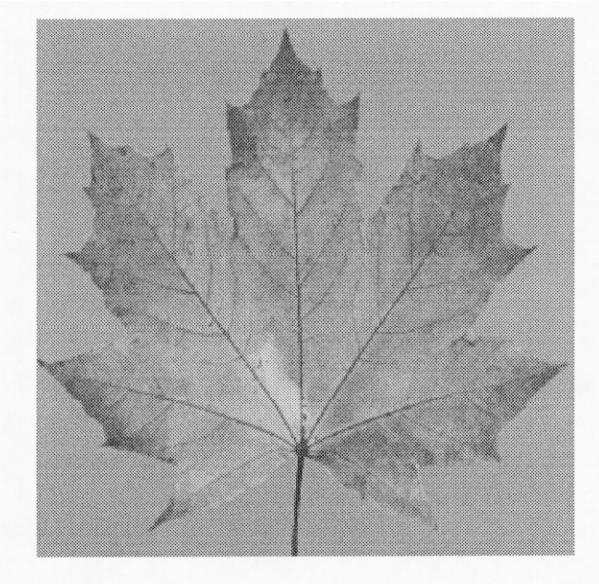

Figure 2. Scanned leaf.

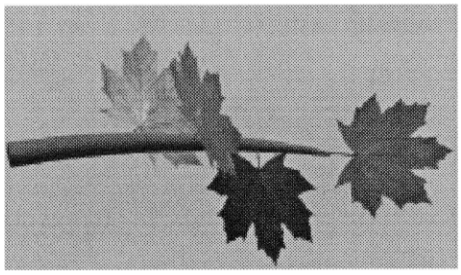

Figure 3. Twig.

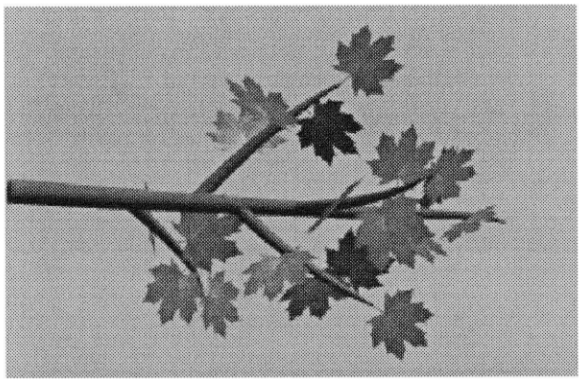

Figure 4. Bigtwig.

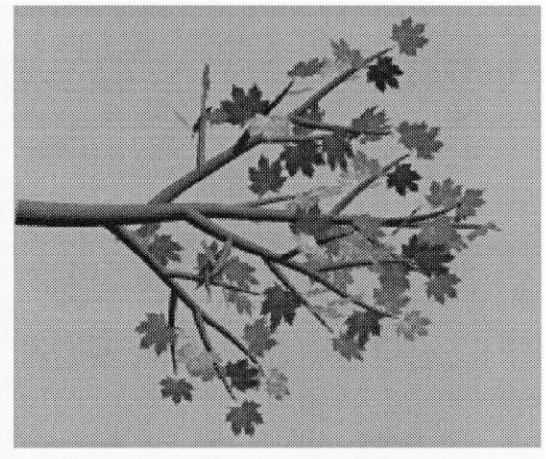

Figure 5. Subbranch.

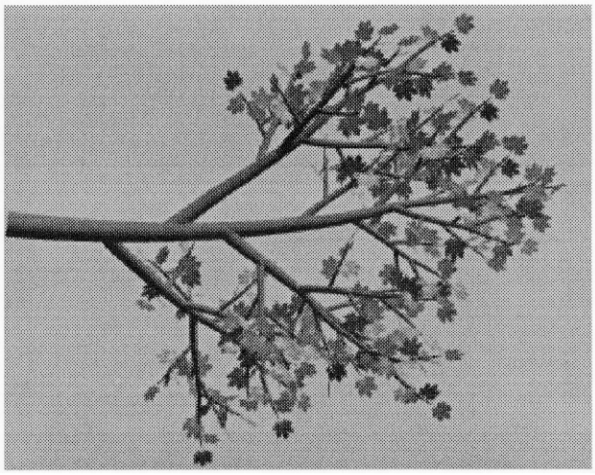

Figure 6. Branch.

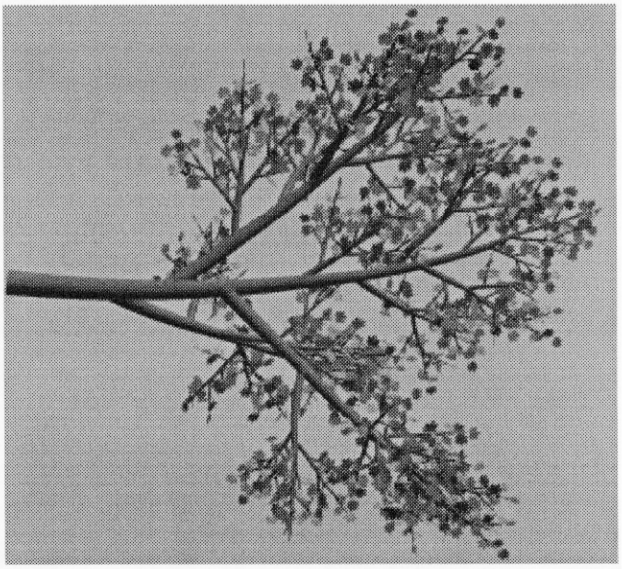

Figure 7. Bigbranch.

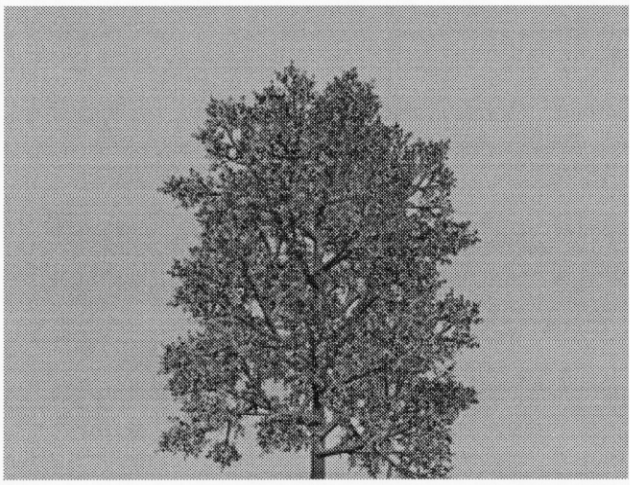

Figure 8. Maple tree in fall.

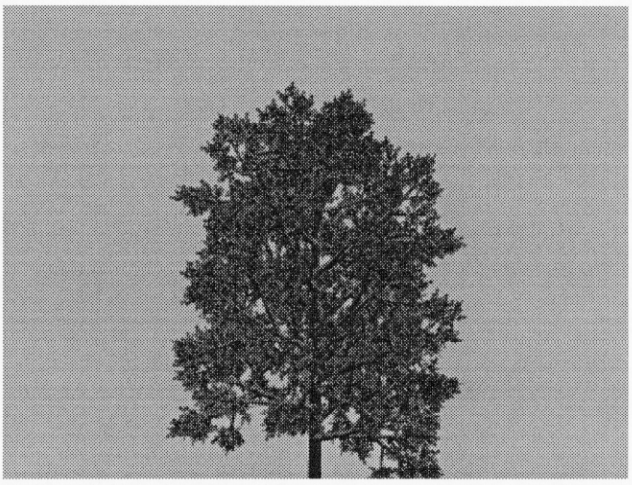

Figure 11. Maple tree in summer.

Figure 9. Medium view of maple tree.

Figure 12. Medium close up of maple.

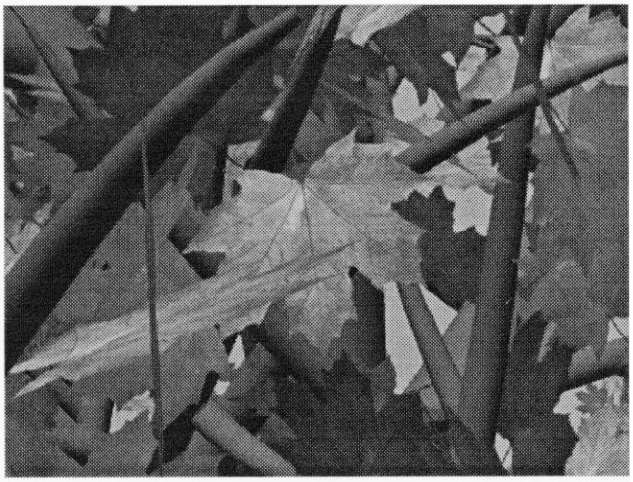

Figure 10. Close up of maple tree.

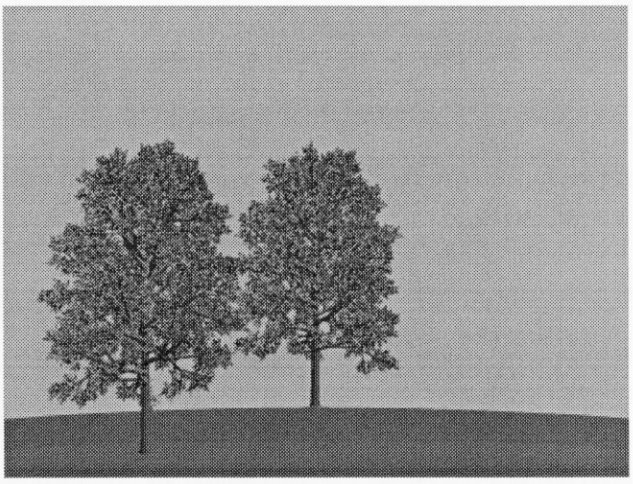

Figure 13. Two maple trees.