

Reconstruction of 3D Tree Models from Instrumented Photographs

by

Ilya Shlyakhter

and

Max Rozenoer

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degrees of
Master of Engineering in Computer Science and Engineering

at the

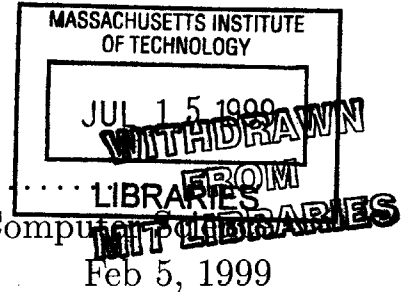
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Feb 1999

June 1999

© Massachusetts Institute of Technology 1999. All rights reserved.

ENG



Author

Department of Electrical Engineering and Computer Science

Certified by

Julie Dorsey
Associate Professor
Thesis Supervisor

Certified by

Seth Teller
Associate Professor
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Students

Reconstruction of 3D Tree Models from Instrumented Photographs

by

Ilya Shlyakhter

and

Max Rozenoer

Submitted to the Department of Electrical Engineering and Computer Science
on Feb 5, 1999, in partial fulfillment of the
requirements for the degrees of
Master of Engineering in Computer Science and Engineering

Abstract

To landscape creators, the ability to model trees is of utmost importance since the realism of the resulting architectural model is greatly enhanced by their presence. Although currently many techniques exist that can produce a convincing tree of a particular type, virtually no work has been done on reconstructing an already existing tree, which represents a canonical inverse problem.

We propose a hybrid approach to solving this inverse problem. It involves first constructing a *plausible* skeleton of the tree (trunk and major branches), and growing the rest of the tree with an L-system, a powerful procedural model. This approach is also interesting from the scientific point of view, since very little work has been done on using a procedural model to solve an inverse problem.

We conclude that although procedural models are indeed very hard to control, it is not impossible and well worth the effort to harness their power.

Thesis Supervisor: Julie Dorsey
Title: Associate Professor

Thesis Supervisor: Seth Teller
Title: Associate Professor

Acknowledgments

We would like to thank Dr. Przemyslaw Prusinkiewicz for his generous L-systems advice and providing us with the versions of his groups' CPGF (Plant and Fractal Generator with Continuous Parameters, an excellent L-systems software package). We are very grateful to Ned Greene, who promptly replied to our request for one of his excellent papers. We want to shake the hand of Neel Master, who helped us out immensely with determining camera pose. But most of all, we want to express all our gratitude to professors Julie Dorsey and Seth Teller, who had put up with us over the scope of several years, offering their advice freely, but never restraining us from probing on our own.

Chapter 1

Introduction.

*No town can fail of beauty, though its walks were gutters and its houses hovels,
if venerable trees make magnificent colonnades along its streets.*

Henry Ward Beecher

1.1 Motivation.

Trees form an integral part of architectural landscapes to such an extent that a treeless street will almost inevitably look bare, deserted and unnatural. Thus, to landscape creators, the ability to model trees is of utmost importance since the realism of the architectural model is greatly enhanced by their presence. Many techniques (e.g., [REEV83], [SMIT84], [AONO84], [BLOO85], [OPPE86], [DERE88], [ARVO88], [VIEN89], [GREE89], [WEBE95], [MECH96]) and modeling packages (e.g., [TREE98], [MAXT98]) have been developed toward this end, allowing for construction of a tree of a particular type. However, variations between two trees of a single type can be very significant, depending heavily on factors ranging from growth conditions (such as amount and direction of sunlight reaching the tree) to human intervention (such as cutting off branches). Hence, the abovementioned techniques and packages are at best unwieldy when the goal is to reconstruct a **particular** tree, or to create a model which bears strong resemblance in appearance to the existing original.

Automated reconstruction of an existing tree could clearly be a useful tool to an architectural modeler, and no such tool exists. The goal of our project is to propose a solution for the inverse problem of reconstructing a 3D model of a foliated tree from a set of instrumented photographs.

1.2 Scientific Interest.

Procedural models have now been used for several decades (reaction-diffusion textures ([WITK91], [TURK91]), fractal terrains ([MUSG89], [EBER94]), and others). Their power lies in having a high *database amplification factor* (a term coined by Smith in [SMIT84]), or their ability to generate a wide range of complex structures from a small set of specified parameters. However, precisely because of the nature of their power, the procedural models are hard to control as a small perturbation in the parameter space results in a significant change in the resulting structure. Perhaps this is the reason that virtually no work has been done on inverse procedural models, namely, forcing a procedural model to create an entity which already exists. Because of the above property of procedural models, an approach to construct an inverse procedural model by a parameter space search is intractable, and other techniques need to be developed.

In his sketch of the only attempt at tree model reconstruction in the literature ([SAKA98]), Sakaguchi recognizes the power of the procedural growth models, but points out that the impossibility of accurately inferring parameters of such a model has driven him to develop another, unrelated approach. While we agree that an inverse procedural model is hard to construct, we do not believe that attempts to employ procedural models to solve an inverse problem should be abandoned. Once the power of a procedural model has been harnessed, the amount of payback is tremendous, allowing for a range of variations in the appearance of the reconstructed model.

In this paper, we propose a hybrid approach to solving the inverse problem of reconstructing a foliated tree. Our method involves first constructing the trunk and the major branches of the tree, and then applying an L-system. L-systems are one of the better known procedural models in the graphics community, especially after their popularization by Lin-

denmayer and Prusinkiewicz [PRUS90]. The following section provides some background on L-systems.

1.3 Background.

In 1968, Lindenmayer proposed a model of development based on string rewriting rules or productions [LIN68]. This model, known as L-systems, originally provided a formal description of the development of simple multicellular organisms. Since growth in the plant kingdom is characterized by self-similarity and, as a result, a high *database amplification factor* inherent to L-systems, the model was later extended to higher plants ([PRUS88], [PRUS90], [FOWL92], [PRUS94], [PRUS95]). As plant appearance is also influenced to a large extent by environmental effects, the L-systems model evolved into open L-systems in [PRUS94] and [MECH96], where the developing plant model is allowed to interact with a model of its environment (also see [MECH96] for a more detailed overview of the L-systems evolution). The following short description of L-systems is reproduced, in a reduced form, from [PRUS94].

An L-system is a parallel string rewriting system, in which simulation begins with an initial string called the *axiom*, which consists of *modules*, or symbols with associated numerical parameters. In each step of the simulation, *rewriting rules* or *productions* replace all modules in the predecessor string by successor modules. The resulting string can be visualized, for example, by interpreting the modules as commands to a LOGO-style turtle.

Even very simple L-systems are able to produce plant-like structures. An example of an interesting L-system with only two productions can be found below in Figure 1-1.

We proceed to outline our approach to the problem of reconstructing a 3D model of a foliated tree from a set of instrumented photographs.

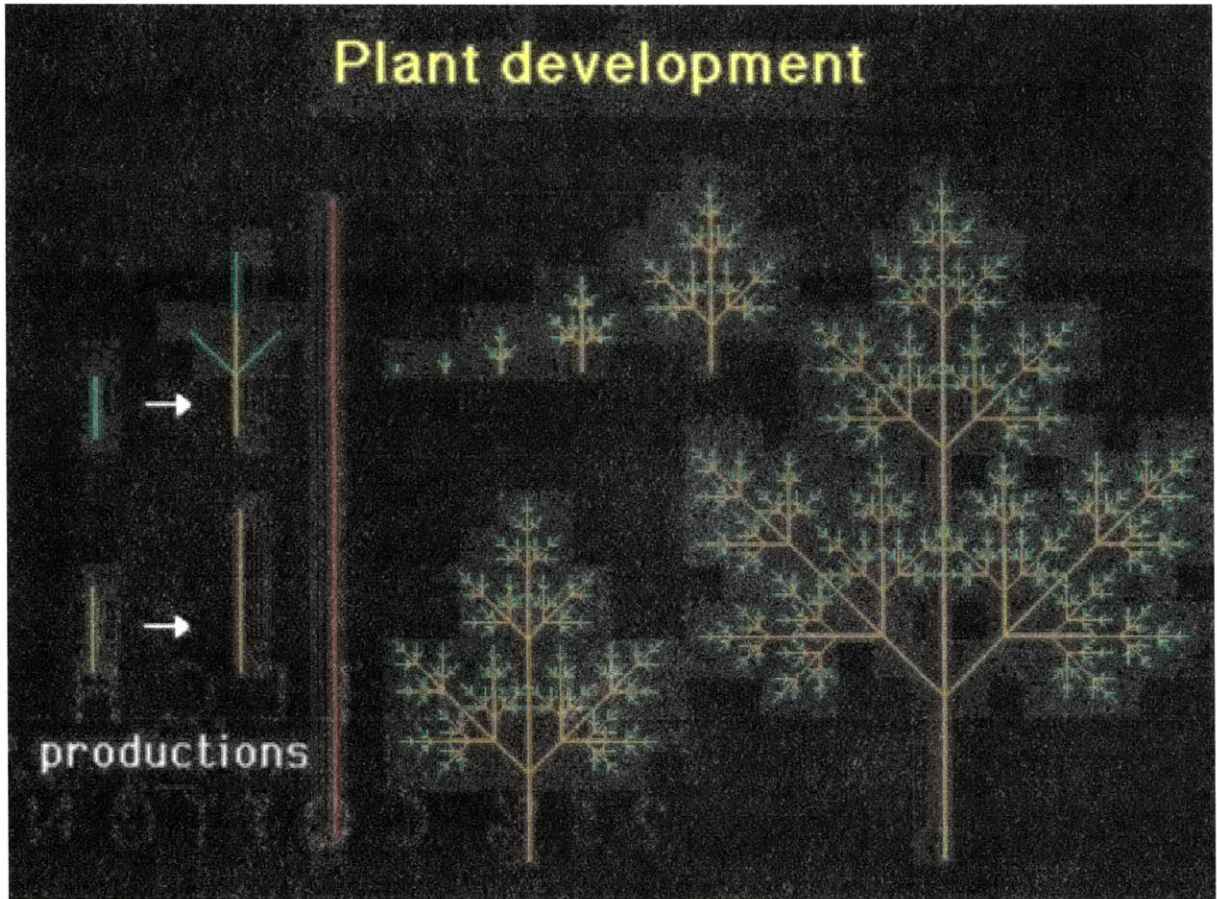


Figure 1-1: A Simple L-system(reproduced from [VMM]). The first production specifies that the apex, or terminal segment, shown in green, creates a branching substructure with three apices and an internode. The second production increases the internode length. The developmental process starts with an apex and yields a compound leaf structure, resembling those found in some fern fronds.

Chapter 2

Approach.

*I like trees because they seem more resigned to the way
they have to live than other things do.*

Willa Cather

Structurally, trees are immensely complex. Bearing that in mind, we can only hope for a reconstruction scheme that produces an similar tree which, although significantly differing from the original in structure, nevertheless retains its overall impression. This is the reason we have restricted the scope of this project to foliated trees: it is hard to reproduce the impression of an unfoliated (winter) tree without achieving a high-fidelity structural reconstruction of its branching system. With foliated trees, however, a *plausible* branching structure is sufficient to preserve the overall impression of the original, since we can not see the original branches in any case. Observing a foliated tree, we infer an intuitive branching structure from the shape of the foliated tree and one's knowledge of trees in general. This process serves as the motivation for our approach.

Despite the excellent results on synthetic L-systems topiary in [PRUS94], in which bushes are grown and pruned so as to fill predefined shapes (most memorably the unforgettable topiary dinosaur), bushes are only remotely related to trees and the impressions produced by a similarly shaped tree and a bush are quite different. On the other hand, our attempts to infer the L-systems parameters to grow a tree inside a predefined shape have been unsuccessful for the reason stated in the introduction: large database amplification factor leads to parameter

space search intractability. The resulting tree's branches often did not reach several large portions of the shape, particularly having trouble navigating the narrow crevices that often arise in the tree shape since they correspond to the major side branches of the original tree.

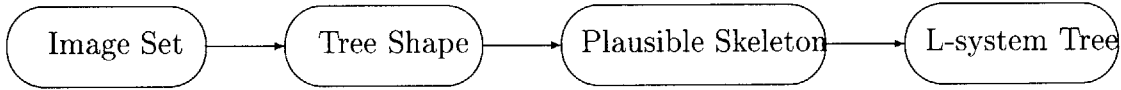


Figure 2-1: Our Approach.

For these reasons, our approach is to infer the *axiom* of the L-system rather than its parameters, applying the L-system only after a *tree skeleton* has been obtained. We construct the skeleton (the trunk and the major branches) of the tree by finding an approximation to the medial axis of the tree shape. This skeleton is not the same as the skeleton of the actual tree, but it is a plausible skeleton for a tree of this shape.

Once we have obtained a plausible skeleton, it is used as an axiom to an L-system for the given tree type (which is entered by the user). The L-system is then applied to grow the low order branches and leaves until the tree shape is sufficiently filled with foliage.

The next chapter includes an overview of our system and the implementation details.

Chapter 3

System Details.

Beware of the man who won't be bothered with details.

William Feather

The three dots ‘...’ here suppress a lot of detail – maybe I should have used four dots.

Donald Knuth

The reconstruction process consists of four stages, (see Figure 3-1: System Diagram). The input to the system is a set of images of a tree, usually numbering between 4 and 15 and uniformly covering at least 135 degrees around the tree. The camera “pose” (relative position and orientation) data is assumed recorded.

3.1 Stage 1: Image Segmentation.

Before anything else may proceed, the input images need to be segmented into the tree and the background.

We have developed a preliminary version of a tree-specific segmentation algorithm, but at this point it is not sufficiently robust to be used for actual tree reconstruction. Consequently, images were segmented manually, by outlining the foliated tree region in a graphics editor. Information on the tree-specific segmentation algorithm we have developed appears in Section 7.1.

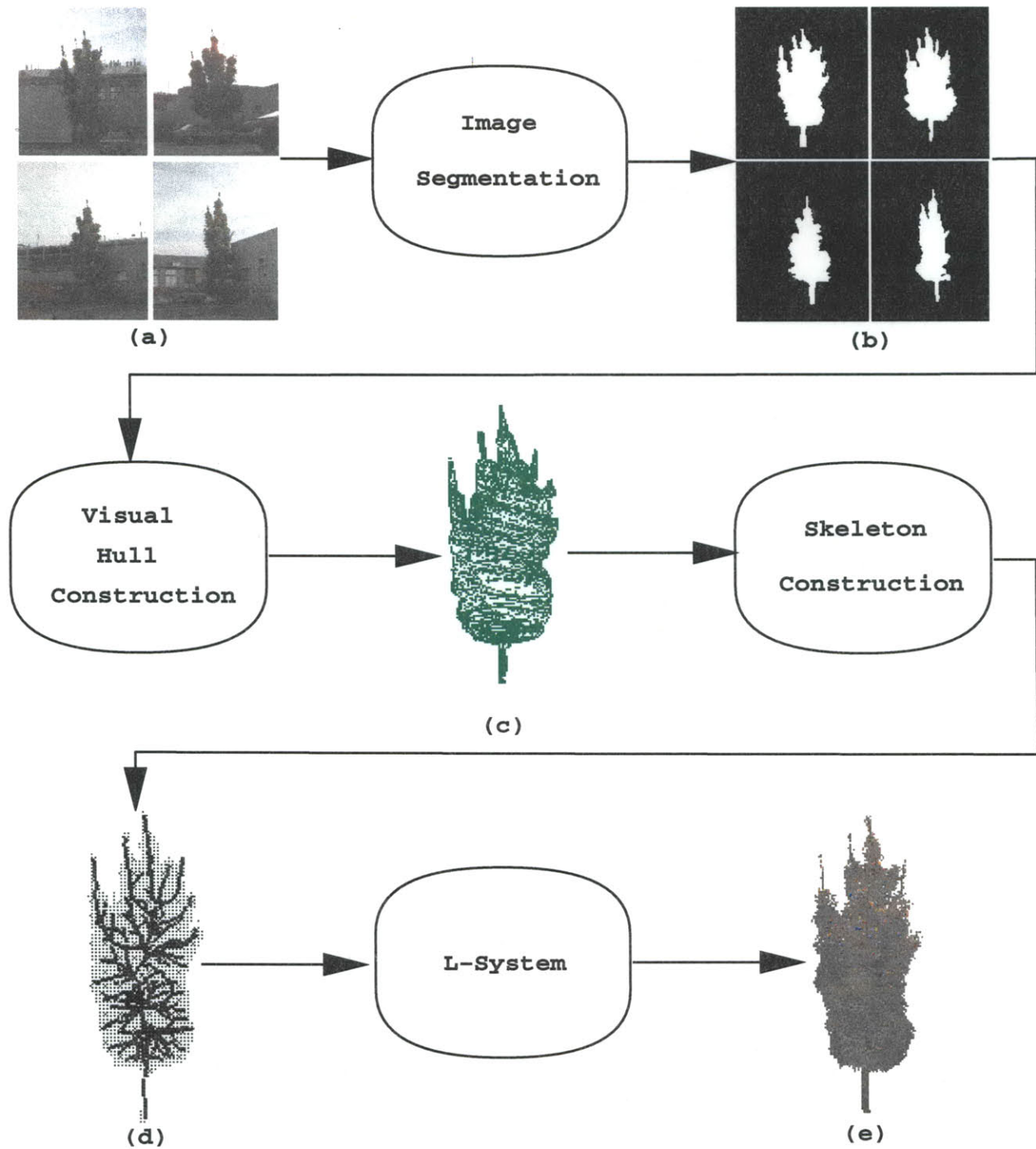


Figure 3-1: System Diagram. Each of the input images (a) is segmented into the tree and the background (b), and the silhouettes are used to construct the visual hull (c). A tree skeleton (d) is constructed as an approximation to the medial axis of the visual hull, and an L-system is applied to grow the small branches and the foliage (e).

3.2 Stage 2: Visual Hull Construction.

From Stage 1, for each photo we have a “silhouette”, or outline, of the tree in that photo. We now use these outlines, together with pose information for each image, to compute a mesh approximation to the 3D tree shape. This mesh will be used to infer the “skeleton” (major branching structure) of the tree in Stage 3 of the algorithm.

We approximate the tree shape by approximating the “visual hull” [LAUR94] of the tree. A visual hull of an object is a structure containing the object, such that no matter from where you look, the silhouettes of the visual hull and the object coincide. Note that the visual hull cannot capture some of the details of the tree shape. For instance, small indentations in a purely spherical shape cannot be seen, since silhouettes in all photographs are still circular. Nevertheless, the visual hull captures most of the appearance-defining elements of the tree shape. Other approaches, such as shape from shading [HORN86], might allow more accurate reconstruction, but for our purposes, the simpler silhouette-based approach seems sufficient.

The general technique used for reconstructing the visual hull is volumetric intersection [LAUR94]. The tree silhouette in each image is approximated by a simple polygon. Back-projecting the edges of this polygon yields a cone – a 3D shape containing all world rays that fall inside the polygonized tree silhouette. Each cone restricts the tree to lie inside the cone, so by computing the 3D intersection of all cones we get an approximation to the 3D tree shape. This process is depicted in Figure 3-2.

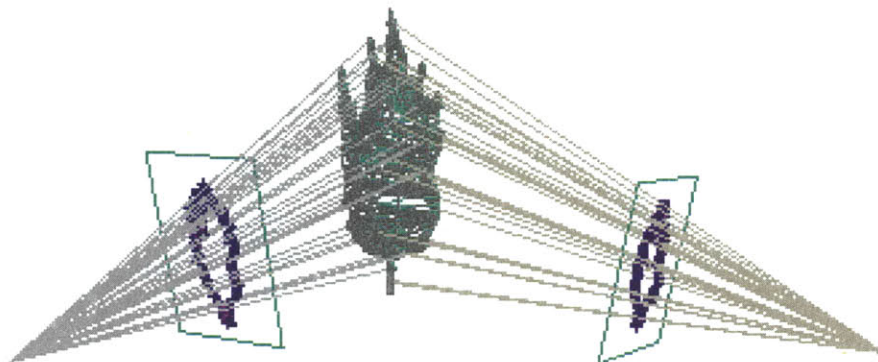


Figure 3-2: Reconstruction of tree shape by silhouette backprojection.

In its simplest form, the visual hull reconstruction algorithm runs as follows:

```
PolygonizeSilhouettesInAllPhotos();  
ShapeSoFar = BackprojectSilhouette(FirstSilhouette);  
for Silhouette in RemainingSilhouettes  
    ShapeSoFar = Intersect(ShapeSoFar,  
                           BackprojectSilhouette(Silhouette));  
ApproxVisualHull = ShapeSoFar;
```

The algorithm's running time is dominated by the Intersect step. Intersecting general 3D polyhedra is a time-consuming operation. However, a backprojected silhouette (a cone) is not a general 3D polyhedron: being a direct extrusion of a simple polygon, it carries only 2D information. This observation lets us perform the intersection in 2D rather than 3D, with significant performance gains. The initial ShapeSoFar is still constructed by backprojecting the silhouette from the first photograph. Subsequent silhouettes are intersected as follows: instead of backprojecting the silhouette and intersecting it in 3D, ShapeSoFar is projected into the silhouette's plane, intersected with the silhouette, and the intersection unprojected back into 3D to create the next ShapeSoFar.

ShapeSoFar is represented as a collection of simple polygons (not necessarily convex) together with adjacency information. At each step, we intersect ShapeSoFar with the next cone – in other words, determine the portion of ShapeSoFar which lies inside the next cone. In terms of the boundary representation that we use, this amounts to deciding, for each of ShapeSoFar polygons, whether it is completely inside the next cone, completely outside, or partially inside and partially outside; in the latter case, we need to determine the portion that's inside. The specific procedure is as follows:

For each Poly in ShapeSoFar polygons Project Poly into the plane of the Silhouette
If the projected Poly's boundary intersects the Silhouette's boundary: Intersect, in 2D, the projected Poly with the Silhouette
Backproject the pieces of Poly falling inside the Silhouette onto the original Poly to find parts of Poly falling inside the corresponding cone

This process is illustrated in Figures 3-3 and 3-4.

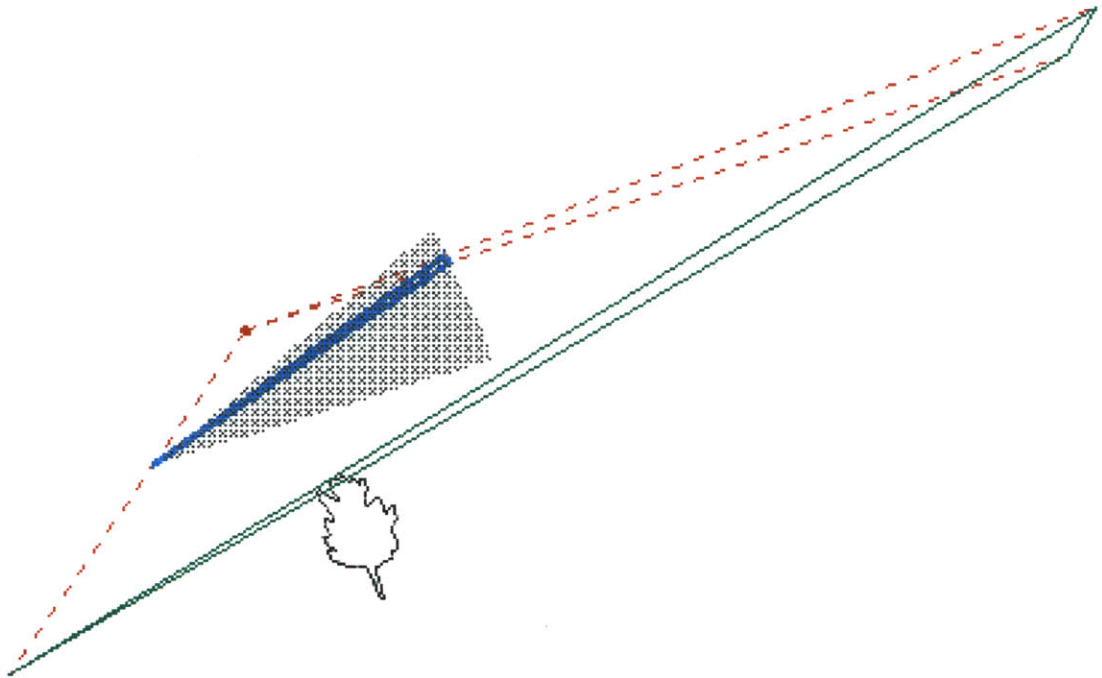


Figure 3-3: A ShapeSoFar polygon is projected into the plane of the new silhouette.

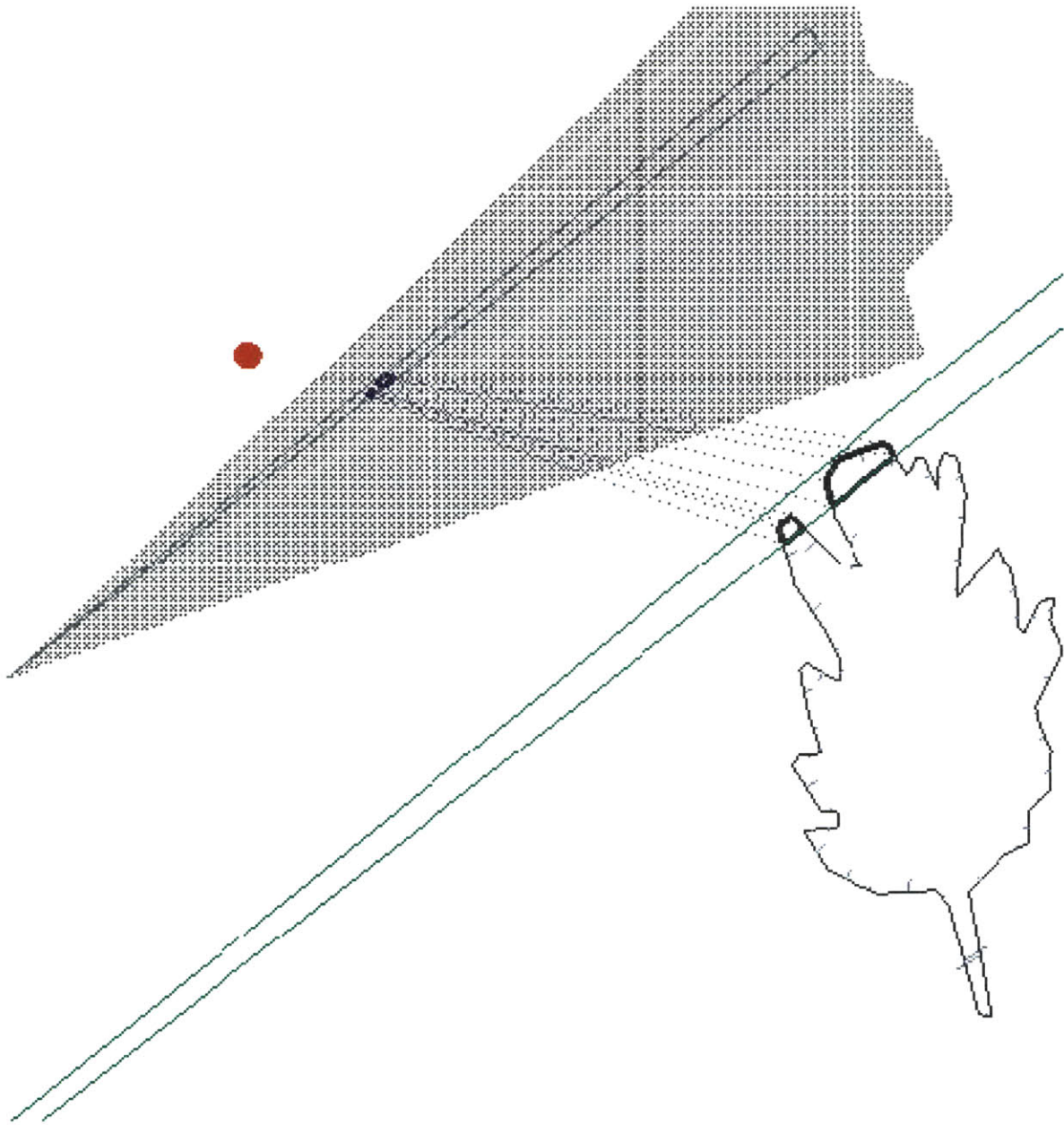


Figure 3-4: 2D intersection of the projected ShapeSoFar polygon with the new silhouette is computed. The result is backprojected onto the original plane of the ShapeSoFar polygon, to obtain the portion of the original polygon that is inside the new cone.

Each of the remaining polygons of ShapeSoFar is either entirely inside or entirely outside the cone. We utilize adjacency information to find polygons that are inside. During the cutting of projected ShapeSoFar polygons in the preceding step, we encounter edges of the original ShapeSoFar falling entirely inside the cone. Starting at each such edge, we perform a “flood-fill” among uncut polygons to reach all polygons falling completely inside the cone. This procedure is taken from [ELBE97].

At this point, we have determined all polygons that will make up the next ShapeSoFar, except for polygons cut from the sides of the cone by ShapeSoFar. (Each side of the cone corresponds to one backprojected segment of the silhouette.) Since we have already determined all edges of the new ShapeSoFar, we can now use the adjacency information to trace out polygons required to close the model.

3.3 Stage 3: Construction of a Plausible Tree Skeleton.

A tree skeleton, as we define it, is the trunk of the tree and the first two levels of branching. We construct a plausible tree skeleton via finding an approximation to the medial axis of the visual hull of the tree.

The medial axis (see [SHER95], [OGNI95]) of a 3D object A is defined as a set of points inside of A , with the following property: for each point in the medial axis, the largest sphere contained in A and centered at that point is properly contained in **no** other sphere contained in A . It is hard to develop an intuition for medial axes from the definition alone; we need examples. In Figures 3-5 and 3-6, we show examples of the 2D analog of the medial axis, where the spheres in the definition become circles.

As its name suggests, the medial axis in some sense always appears in the middle of the object. This is the reason we believe that a medial axis of the tree shape makes a plausible skeleton for that tree. Also, it is likely to be somewhat similar to the original skeleton because of the following considerations.

Trees sustain themselves by converting the sunlight that reaches their leaves into photosynthates, which then propagate from lower-order to higher-order branches, eventually reaching the trunk ([CHIBA94]). Since each branch costs the tree energy to maintain, the

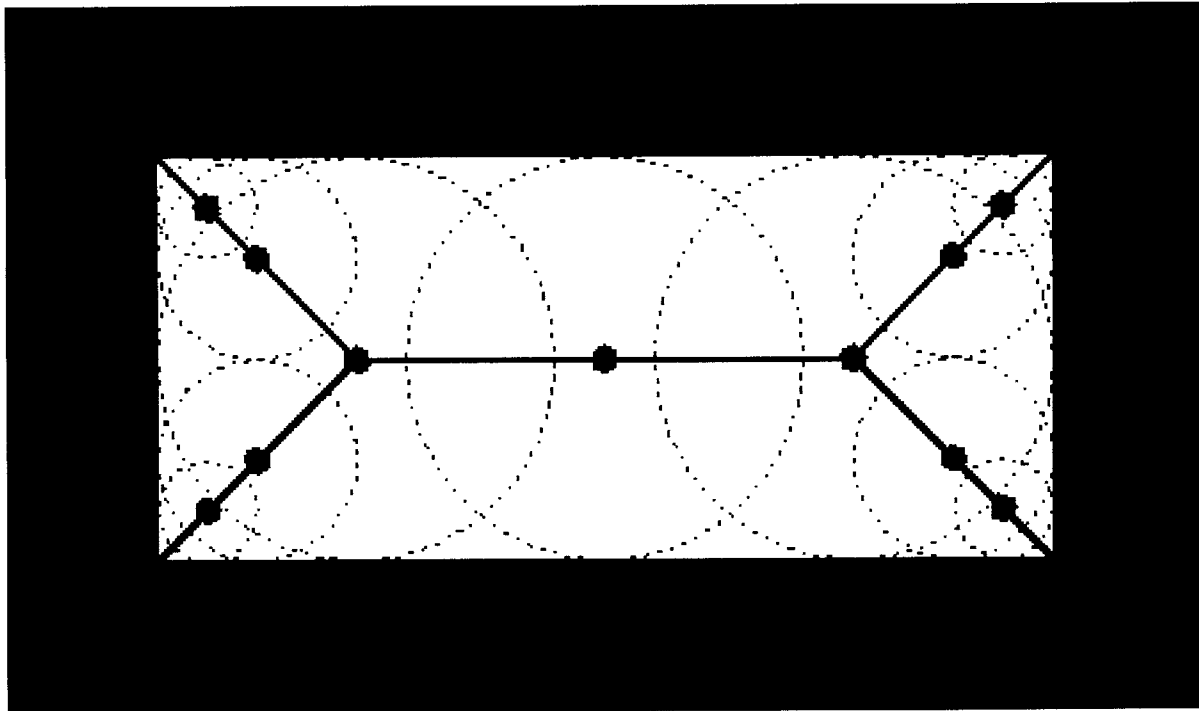


Figure 3-5: The simplest example of a medial axis is that of a rectangle. Notice how the “maximal” circles centered at the medial axis points touch at least two points on the rectangle, so that no other circle inside the rectangle contains them. (The image is reproduced from <http://image.korea.ac.kr/HIPR/HTML/skeleton.htm>).

sane trees’ (and most trees are sane) branching structures are such that their branches reach the most possible space in the most economical way. These observations suggest that the structure of the tree skeleton is similar to the medial axis of the shape of the tree, which is, in some economical sense, the optimal skeleton.

To construct an approximation to the medial axis, we use the algorithm from [TEIC98]. The algorithm was originally designed for constructing the skeletons of characters for animation. It takes a closed polygonal mesh as input and computes its Voronoi diagram, retaining only the Voronoi nodes which fall inside the mesh. The user then manually fixes a number of Voronoi nodes, indicating the points that he wants to be present in the resulting medial axis approximation. The algorithm enters a loop, on every pass through which it removes one Voronoi node from every biconnected component¹ of the modified Voronoi diagram; the

¹A biconnected component of a graph has the property that if any one of its edges is removed, the

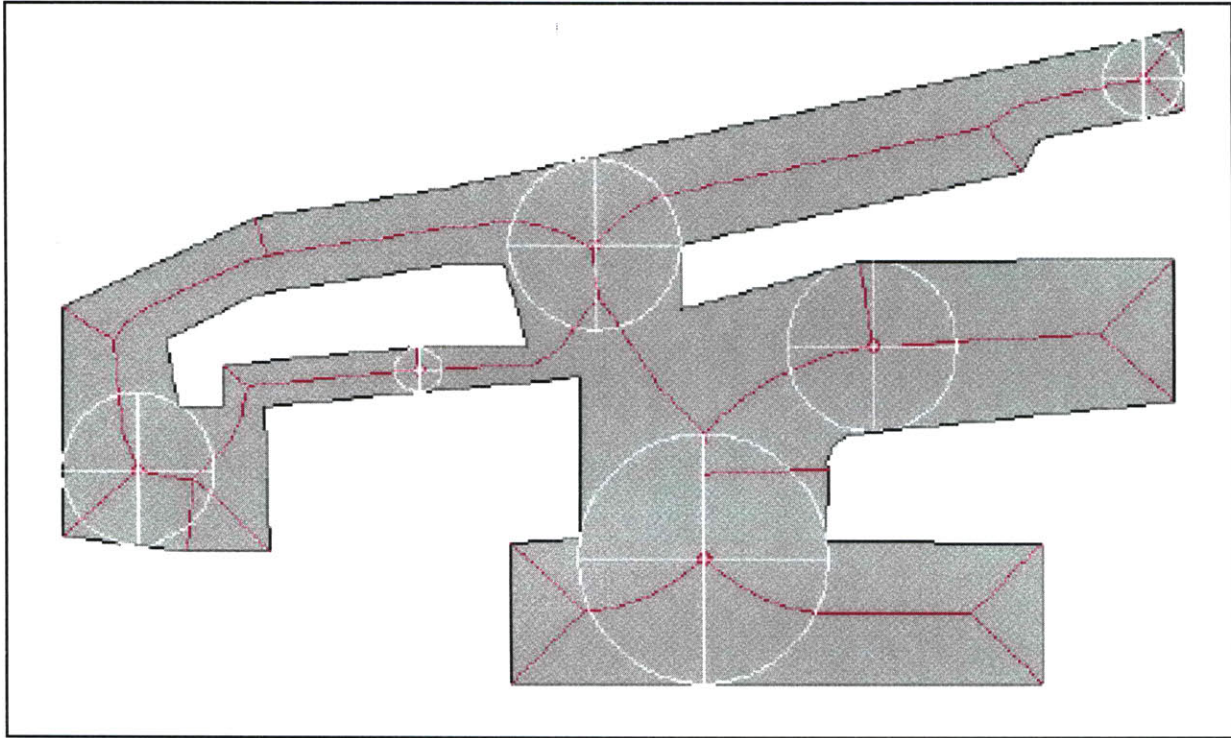


Figure 3-6: This example is a bit more involved, perhaps giving an idea of why the medial axis is called *medial*. (The image is reproduced from <http://www.fegs.co.uk/motech.html>).

nodes initially marked are not removed. This continues until no biconnected components are remaining.

This is what the algorithm really does. Imagine that the polygonal model is filled with a large number of balls². The balls at the tips of the model, the places you definitely want the skeleton to reach, are marked with a pencil. The algorithm runs around the model, grabs the balls it can reach (those closer to the surface of the model) and throws them into a trash bin. It cannot throw out the marked balls, however; moreover, each of the remaining balls should be touching at least one other ball. Subject to these constraints, the algorithm stops throwing out balls when he can not throw out any more.

Our extension to the algorithm automatically fixes a set of Voronoi nodes, which intu-

component remains connected.

²Really, the balls correspond to the circumspheres of the tetrahedrons resulting from the Delaunay triangulation of the mesh vertices.

itively should correspond to major branch tips, as well as the bottom of the trunk. These are obtained by first finding “interesting” vertices (those corresponding to major branch tips) in the 2D images, then backprojecting them and finding the intersections of backprojected rays to determine interesting points on the 3D visual hull.

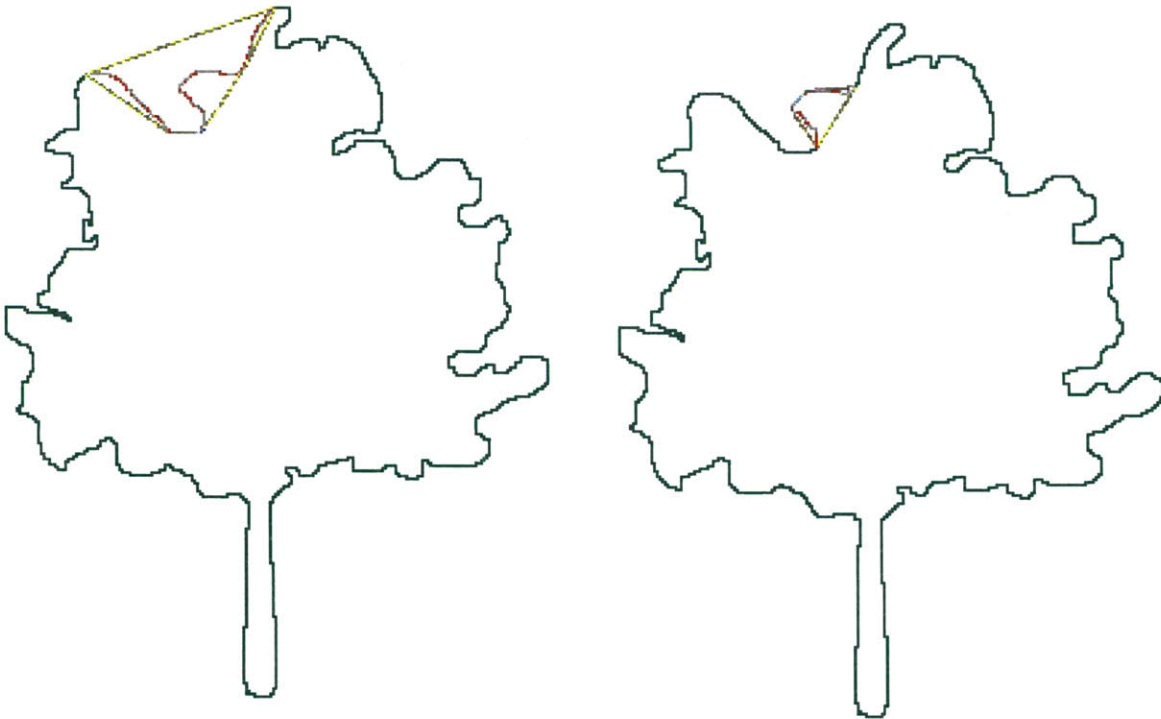


Figure 3-7: Examples of 1st and 2nd order convex hulls (shown with the tree contour, on the upper side). No vertices are ever selected from the 1st order convex hull. One vertex will be selected from the 2nd order convex hull shown, and it indeed corresponds to a branch tip.

In 2D, *interesting* vertices are selected from the convex hull of the tree contour and the even-order convex hulls (or even-level nodes of the convex difference tree ([RAPP92])). An n -th order convex hull is obtained by computing the convex hull of a set of contour vertices that lie between any two successive vertices of the $n - 1$ -st order convex hull (see Figure 3-7 for an example). The original convex hull of the whole contour is considered to be 0-th order.

Of the vertices thus selected, *interesting* vertices can be either ones at which a convex hull makes a sharp turn, or ones on which long hull edges are incident. These heuristics

ensure that there cannot be many *interesting* points on the 2D tree contour, and hence on the 3D visual hull.

A sample output of the selection algorithm can be found below in Figure 3-8.

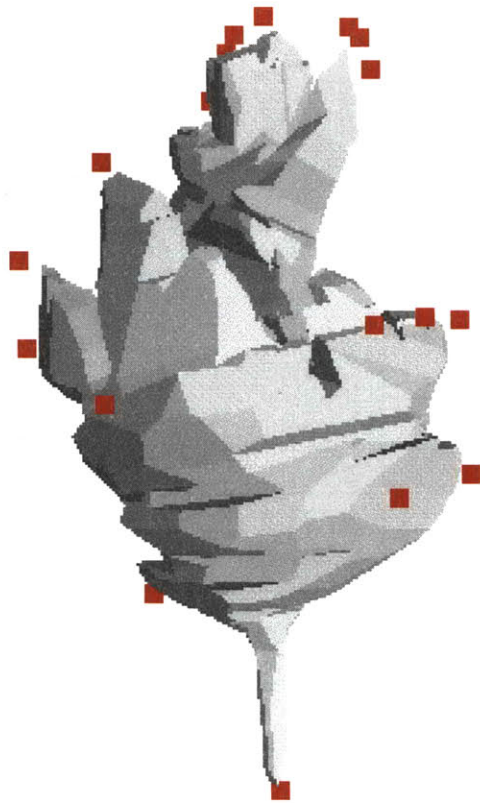


Figure 3-8: A sample output of the automatic selection algorithm: Voronoi nodes near the red points in the figure will be marked for the medial axis approximation algorithm.

Once these interesting 3D point are found, for each such point, the Voronoi vertex closest to it is found and marked to be fixed for the medial axis approximation algorithm. The algorithm is then run and an initial skeleton is obtained. However, this initial skeleton does not cover the space inside the visual hull sufficiently for L-systems to be successfully applied. The initial skeleton is then extended by a second pass of the algorithm from [TEIC98], in which all the Voronoi nodes belonging to the initial skeleton are fixed, as well as a number of other Voronoi nodes, uniformly covering the space inside the tree shape. This produces the effect of adding another level of branching to the initial skeleton.

At this point, we have an tree skeleton consisting of the trunk and the first few levels of

branches, reaching into all areas of the tree. The stage is now set for the L-systems to be applied.

3.4 Stage 4: L-systems.

As a twig is bent the tree inclines.

Virgil

The tree skeleton from the preceding step is written out as an axiom to an L-system, in such a way that the last two levels of branches are endowed with buds, allowing for further levels of growth. We now apply an open L-system for tree growth corresponding to the type of the input tree, which communicates with two environmental processes. The first process ensures that the resulting tree's shape is close to that of the input tree; the second increases botanical fidelity of the branching pattern and the leaf distribution.

The first environmental process is responsible for pruning the branches that in the course of their growth reach outside of the tree shape. The last branch segment is deleted and a new bud appears at the end of the freshly pruned branch. The new bud's parameters are selected so that the branch that will grow out of it on the next iteration of the L-system will be shorter than the currently pruned branch by a constant factor defined in the L-system.

The second environmental process computes the amount of sunlight reaching each leaf in the tree. The process augments the mechanism for modeling competition for sunlight and the flow of photosynthates inside the tree ([CHIBA94]) that is implemented in the L-system. The amount of sunlight reaching each leaf is converted into a quantity of photosynthates, which then make their way up from the thinner branches to the thicker and, eventually, the trunk. On the way, each branch collects a toll, corresponding to the amount of energy that it needs for maintenance. If a branch does not get enough energy, it is deleted (provided that it is small enough, we don't want to destroy a significant portion of the tree).

An L-system *iteration* corresponds to a single year of tree growth. In the course of an iteration, buds grow into branches with new buds on them, branches outside of the tree shape are pruned, flow of photosynthates is computed and small branches that don't get enough

energy wither and die and, finally, branch thickness is recomputed to take into account the new branches.

The L-system is run for several iterations, so as to sufficiently fill the space inside the visual hull with foliage. This is not fully automated in our system, but in our experience it takes three or four L-System iterations to grow the tree to the desired thickness of foliage. It is safer to overgrow the tree than to undergrow it, since after the L-System is done leaves can be removed via the procedure outlined below.

Once the L-system simulation is finished, colors are mapped back to the leaves from the images. If a mapped back color from any of the images clearly can not belong to the tree, the leaf is deleted. This usually corresponds to an opening in the thin foliage where the sky or a background wall shows through. Foliage density is one of the key factors controlling the appearance, and this procedure partially solves the foliage thickness problem, which will be further discussed in Section 6.1.

As should be expected, the L-system model contains a number of parameters that are tree-type specific, such as, for example, leaf tropism factor, branching angles, and leaf shape. Being able to modify the appearance of the final result by varying model parameters is an important feature of our approach as compared to [SAKA98].

In the next chapter, we present some of the results obtained using our system.

Chapter 4

Results.

Results! Why, man, I have gotten a lot of results. I know several thousand things that won't work.

Thomas A. Edison

4.1 Visual Hull Construction.

A novel robust visual hull construction algorithm has been designed and implemented. It is data-driven in that it works with silhouettes represented as concave polygons, and thus any level of detail up to image resolution can be chosen. The algorithm works with exact arithmetic (using LEDA), and so is robust. Since all the intersections are performed in 2D, and the adjacency information is heavily relied upon to construct the new polygons, the algorithm is relatively fast (under 20 minutes on an SGI Reality Engine to construct a visual hull from 10 100-edge silhouettes¹

It is also public (<http://graphics.lcs.mit.edu/treerec/>).

4.2 Tree Skeleton Reconstruction.

Plate 1 contains three examples of tree skeletons constructed by our method.

¹Using more silhouettes doesn't really change the model after that point, if the initial ones are relatively well-placed.

4.3 Tree Reconstruction: Results.

Plates 2 and 3 contain snapshots of three reconstructed trees, with the originals for comparison.

Chapter 5

Conclusion.

In this paper, we proposed a solution to the inverse problem of reconstructing an existing foliated tree. In particular, we employed L-systems, a powerful procedural model, to solve the inverse problem. We conclude that although procedural models are indeed very hard to control, it is not impossible and well worth the effort to harness their power.

In the domain of tree generation, all existing models fall into one of two broad classes: biology-based and geometry-based. Biology-based models incorporate laws biological constraints on growth, producing botanically correct trees; but the user has relatively little control over the final outcome, in particular over the shape. On the other hand, geometry-based models can be used to achieve greater control over the final geometry, but botanical fidelity can become problematic. Our hybrid approach combines the advantages of these two model types: we can dictate the overall tree geometry, yet let the model grow most of the tree without direct control.

While at this point the method is not fully automatic, we believe that most steps are amenable to complete or nearly complete automation. Our method provides a useful framework, making it possible to experiment with varying solutions to the individual steps. Eventually, the reconstruction algorithms described in this work could be used to support applications such as architectural planning and virtual reality simulation.

Chapter 6

Future Work.

The true meaning of life is to plant trees, under whose shade you do not expect to sit.

Nelson Henderson

So why don't you make like a tree... and get outta here.

Biff, "Back to the Future"

The problem of reconstructing a 3D tree model from a real image set is a recent one and a hard one. It should then come as no surprise to the reader that the Future Work Chapter of this paper is quite sizeable. The directions of future work are conveniently separated into the sections below.

6.1 Tree-Specific Segmentation.

Automatically segmenting the image into the tree and the background is a separate and well-defined vision problem, albeit one that seems very hard to solve completely because of a number of reasons such as the great variety of tree shapes and background trees in the image. Our approach to tree-specific segmentation is very ad hoc (see Section 7.1) and we are sure that an easier and more robust approach can be developed based on texture segmentation methods (see [REED93] for a survey of such methods). Also, the algorithm should in some form employ the knowledge of what trees look like in general.

The thickness of tree foliage plays an important role in the overall impression produced by a tree. To solve the foliage thickness problem we ideally would like to be able to tell **exactly** which pixels are occupied by the tree in each image. Where the background is a bright sky, this is easy; however, in other places (for example, in the lower half of the tree), more sophisticated texture analysis methods are needed to be applied. This extension to the segmentation algorithm is another possible direction of future work.

Another completely separate vision problem, preceding tree-specific segmentation in the pipeline, is that of determining whether the image contains a tree and, if so, locating it in the image.

6.2 Tree Skeleton Construction.

In [SAKA98], Sakaguchi states that “Special [branching] rules are not needed for each kind of tree because the form differences of various species can almost be represented [inferred] from the volume data.” We think that this is indeed true, but only to an extent. The shape of the tree does impose strong restrictions on its branching structure, but the difference between branching angles and types of branching (monopodial, dipodial) of trees of different types results in a noticeable effect when the tree is viewed without its foliage.

Using tree type-dependent L-systems solves this problem only partially, since the tree skeleton is constructed relying solely on the shape information. Designing a skeleton construction algorithm to accurately model branching angles based on tree type information is then another direction of future work.

6.3 L-systems.

In our experience, L-system design is not an easy task. The L-systems used in our examples are almost the simplest possible systems for achieving the given purpose. Also, although different L-systems were used for trees of different types, the changes made were **very** slight; for example, even the branching angles and types used were all the same. These models can be made arbitrarily more elaborate, considerably improving the resulting tree models.

Generally, a library of tree types needs to be developed, with an L-system for each tree type. Since collections of different type parametric 3D trees already exist ([TREE98], [MAXT98]), creating such a tree library is a question of time.

One possible improvement to the L-system model would be to utilize the reconstructed visual hull more extensively. Having the hull allows the L-system, for example, to determine how far it can grow in a particular direction before hitting the tree boundary. This information can be used to select an appropriate initial branch length, to ensure that as the branch grows, it does not overshoot the tree boundary too noticeably. Right now, all branches start out with fixed length segments, irrespective of whether they're growing in a wide or a narrow region of the tree.

Automatically stopping the growth of the L-system is another possible improvement to our system. A possible way to accomplish that would be to project the tree to all the images after each L-system iteration, and compute a measure of how well the tree fills the filtered images; this is also closely related to the work on foliage thickness. Based on that measure, a decision can then be made whether the L-system should run for another iteration.

6.4 Miscellaneous.

As the poet said, 'Only God can make a tree' – probably because it's so hard to figure out how to get the bark on.

Woody Allen

We have done no work on making the tree model more visually appealing than a collection of cylinders for branches and colored polygons for leaves. Although not apparent when viewed from a distance, the shortcomings of this simple model take their toll when the viewer is close to the model. A needed extension to the system is the application of the methods enhancing the visual appearance of the tree. These include, to name a few, the work of Bloomenthal on the appearance of a maple [BLOO85], and the truly excellent work of Greene, who figured out how to get the bark onto the tree and more with a voxel space automaton [GREE91]. Application of these and other models would further enhance the

photorealism of the reconstructed tree.

The reconstruction algorithm uses the tree type information, and so far we have assumed that this information will be entered by the user. Another direction of future work is to automatically identify the tree type by performing feature analysis on the tree image texture as well as the 3D shape of the tree. This, together with a robust automatic filter, would make the whole system fully automatic.

Lastly, no work has been done on reconstructing the winter, non-foliaged trees. In the case of winter trees, one can actually *see* the branching structure, and the problem is to infer it from the images. In this case, we possess much more information than in the foliated case, where we only had the tree shape; however, the expectations rise accordingly and a *plausible* branching structure is no longer enough. A completely different approach is needed, based on finding the branches in the image, backprojecting them and determining possible 3D structures that fit the images. The problem of reconstructing non-foliaged trees is only remotely related to the corresponding foliated trees problem.

Chapter 7

Appendix.

7.1 A: A Tree-Specific Segmentation Algorithm.

We use a straightforward approach to automatically filter a foliated tree in an image. The first stage of the multistage algorithm involves high-pass filtering the image using a Gaussian filter. The motivation for high-pass filtering the image is that the foliage in the image is an area of a two-dimensional high frequency signal, and most of the points left in the (thresholded) image after filtering should belong to a tree. In the second stage, more points are thrown out on the basis of color analysis: the algorithm finds trees which are roughly green. The remaining points are joined into structures by first allocating a structure for every point and then successively joining nearby structures. The largest such structure is retained and floodfilled. The tree in the resulting segmentation corresponds to the structure, and everything else is considered to be background.

Two tree-specific segmentations obtained with our filter can be found in Plates 4 and 5.

Since the tree is often dark in its lower half, a gap sometimes results in that area. We make an attempt to close off this gap by not allowing any “dents” in the structure from below. This explains the straight lines in the segmentation result in Plate 5. It is better for us for the segmented tree to be too large rather than too small, because of the nature of volumetric intersection which follows the segmentation: an “extra” area in the segmentation will most likely be cut off by another view, but if an piece of the tree is missing, nothing will

revive it.

The algorithm works well in a number of cases, but is far from robust and its list of deficiencies includes that it only works on near-green trees, and that it does not find the trunk of the tree. We discuss construction of a better filter in Section 6.1.

7.2 B: Determining Pose.

We mention in Section 3 that we assume the camera “pose” (relative position and orientation) to be known. We, however, did not possess this information and needed to solve the problem of motion recovery from point correspondences. That is, given a set of images and a set of pairs of corresponding points on them, we need to compute relative position and orientation of the cameras. Although problem is extremely well studied in the literature (see [FAUG93]), we were not able to find a comprehensive public domain software package that would solve the problem robustly.

Our approach to solving the problem is as follows. First, we manually select pairs of corresponding points in the images. Then, we apply the public software by Zhengyou Zhang, implementing the algorithm from [ZHAN96]. In our experience, the algorithm is not very reliable (decreases in accuracy as the magnitude of rotations between the cameras increases); moreover, using the algorithm we are able to obtain only the normalized translations (directions) between the cameras. However, since the actual distances between the cameras were roughly the same, we are able to use the results of the Zhang algorithm to obtain initial estimates for use with an iterative solver.

The iterative solver has been developed as part of the City Scanning Project at MIT. The key point of the algorithm it implements is that it involves alternatively improving the estimates for the structure (the positions of 3D points which project to the 2D points making up the correspondence pairs) and the motion (positions and orientations of the cameras). We have found that the algorithm works excellently, allowing for accurate tree shape construction.

7.3 C: Source Code.

The source code for this project will appear on <http://graphics.lcs.mit.edu/treerec/>.

Chapter 8

Bibliography.

- [AONO84] M. Aono, T. L. Kunii, Botanical Tree Image Generation, IEEE Computer Graphics & Applications 4,5 (1984), 10-34.
- [ARVO88] J. Arvo, D. Kirk. Modeling Plants with Environment-Sensitive Automata, Proceeding of Ausgraph '88, 27-33.
- [BLOO85] J. Bloomenthal. Modeling the mighty maple. Proceedings of SIGGRAPH'85, San Francisco July 22-26, 19(3), pp. 305-311.
- [CHIBA94] N. Chiba, S. Ohkawa, K. Muraoka, and M. Miura, Visual simulation of botanical trees based on virtual heliotropism and dormancy break. The Journal of Visualization and Computer Animation 5 (1994), 3-15.
- [COOR97] S. Coorg, S. Teller. Matching and Pose Refinement with Camera Pose Estimates. Proceedings of the 1997 Image Understanding Workshop, New Orleans.
- [DERE88] P. de Reffye, C. Edelin, J. Franson, M. Jaeger, C. Puech. Plant models faithful to botanical structure and development. Computer Graphics, 22(4), August 1988, pp. 151-158.
- [EBER94] D. S. Ebert, Editor, F. K. Musgrave, D. Peachey, K. Perlin, S. Worley. Texturing and Modeling: A Procedural Approach. Academic Press, Inc., 1994.
- [ELBE97] G. Elbe. Private Communication.
- [FAUG93] O. Faugeras. Three-Dimensional Computer Vision: A Geometric Viewpoint. The MIT Press, 1993.

- [FOWL92] D. Fowler, P. Prusinkiewicz and J. Battjes. A collision-based model of spiral phyllotaxis. *Computer Graphics*, 26(2), July 1992, pp. 361-368.
- [GARD84] G. Gardner. Simulation of natural scenes using textured quadric surfaces. *Computer Graphics*, 18(3), July 1984, pp. 11-20.
- [GREE89] N. Greene. Voxel space automata: modeling with stochastic growth processes in voxel space. *Computer Graphics*, 23(3), July 1989, pp. 175-184.
- [GREE91] N. Greene. Detailing tree skeletons with voxel automata. SIGGRAPH'91 course notes on photorealistic volume modeling and rendering techniques (1991), 7:1-7:15.
- [HORN86] B. K. P. Horn. *Robot Vision*. MIT Electrical Engineering and Computer Science Series, MIT Press, 1986.
- [LAUR94] A. Laurentini. The Visual Hull Concept for Silhouette-Based Image Understanding. *PAMI*(16), No. 2, February 1994, pp. 150-162.
- [LIN68] A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology*, 18:280-315, 1968.
- [MAXT98] MaxTrees, collection of parametric 3D trees for 3D Studio Max, Copyright Nsight Studios.
- [MECH96] R. Mech and P. Prusinkiewicz. Visual models of plants interacting with their environment. SIGGRAPH'96 Proceedings.
- [MUSG89] F. K. Musgrave, C. E. Kolb, R. S. Mace. The synthesis and rendering of eroded fractal terrains. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, pp 41-50, July 1989.
- [OGNI95] R. L. Ogniewicz, O. Kübler. Hierarchic Voronoi Skeletons. *Pattern Recognition*, 28(3):343-359, 1995.
- [OPPE86] P. Oppenheimer. Real time design and animation of fractal plants and trees.
- [PRUS88] P. Prusinkiewicz, A. Lindenmayer and J. Hanan. Developmental models of herbaceous plants for computer imagery purposes. *Computer Graphics*, 22(4), Aug 1988.
- [PRUS90] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J.S. Hanan, F.D. Frachhia, D.R. Fowler, M.J.M. de Boer, and L. Mercer.
- [PRUS94] P. Prusinkiewicz, M. James and R. Mech. Synthetic Topiary. *Computer graphics*

proceedings, Annual Conference Series, 1994.

[PRUS95] P. Prusinkiewicz. Visual models of morphogenesis. In *Artificial Life. An Overview*. Christopher G. Langton, 1995.

[RAPP92] A. Rappoport. An efficient adaptive algorithm for constructing the convex differences tree of a simple polygon. *Computer Graphics Forum*, 11(4), pp. 235-240, October 1992.

[REED93] T. Reed, J. M. Hans du Buf. A review of recent texture segmentation and feature extraction techniques. *CVGIP: Image Understanding*, 57(3), May, pp. 359-372, 1993.

[REEV83] W. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *Computer Graphics*, 17(3), July 1983, pp. 359-376.

[SAKA98] T. Sakaguchi, Botanical Tree Structure Modeling Based on Real Image Set, SIGGRAPH'98 Technical Sketch (SIGGRAPH'98 Proceedings, p. 272).

[SHER95] E. C. Sherbrooke, N. M. Patrikalakis, E. Brisson. Computation of the Medial Axis Transform of 3-D Polyhedra. In *Proc. ACM SIGGRAPH Symposium on Solid Modeling*, pp 187-199, 1995.

[SMIT84] A. Smith. Plants, fractals and formal languages. *Computer Graphics*, 18(3), July 1984.

[TEIC98] M. Teichmann, S. Teller. Assisted Articulation of Closed Polygonal Models. SIGGRAPH'98 Technical Sketch.

[TREE98] Tree Professional Software. Copyright 1992-1998 Onyx Computing.

[TURK91] G. Turk. Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion. *Computer Graphics*, 25(4), July 1991.

[VIEN89] X. Viennot, G. Eyrolles, N. Janey and D. Arques. Combinatorial analysis of ramified patterns and computer imagery of trees. *Computer Graphics*, 23(3), July 1989.

[VMM] Visual Models of Morphogenesis: A Guided Tour. P. Prusinkiewicz, M. Hammel, R. Mech. <http://www.cpsc.ucalgary.ca/Redirect/bmv/vmm-deluxe/>.

[WEBE95] J. Weber and J. Penn. Creation and rendering of realistic trees. *Computer Graphics Proceedings, Annual Conference Series*, 1995, pp. 119-127.

[WITK91] A. Witkin and M. Kass. Reaction-Diffusion Textures. *Computer Graphics*,

25(4), July 1991.

[ZHAN96] Z. Zhang. A new multistage approach to motion and structure estimation: From essential parameters to euclidian motion via fundamental matrix. Research Report 2910, INRIA Sophia-Antipolis, France.

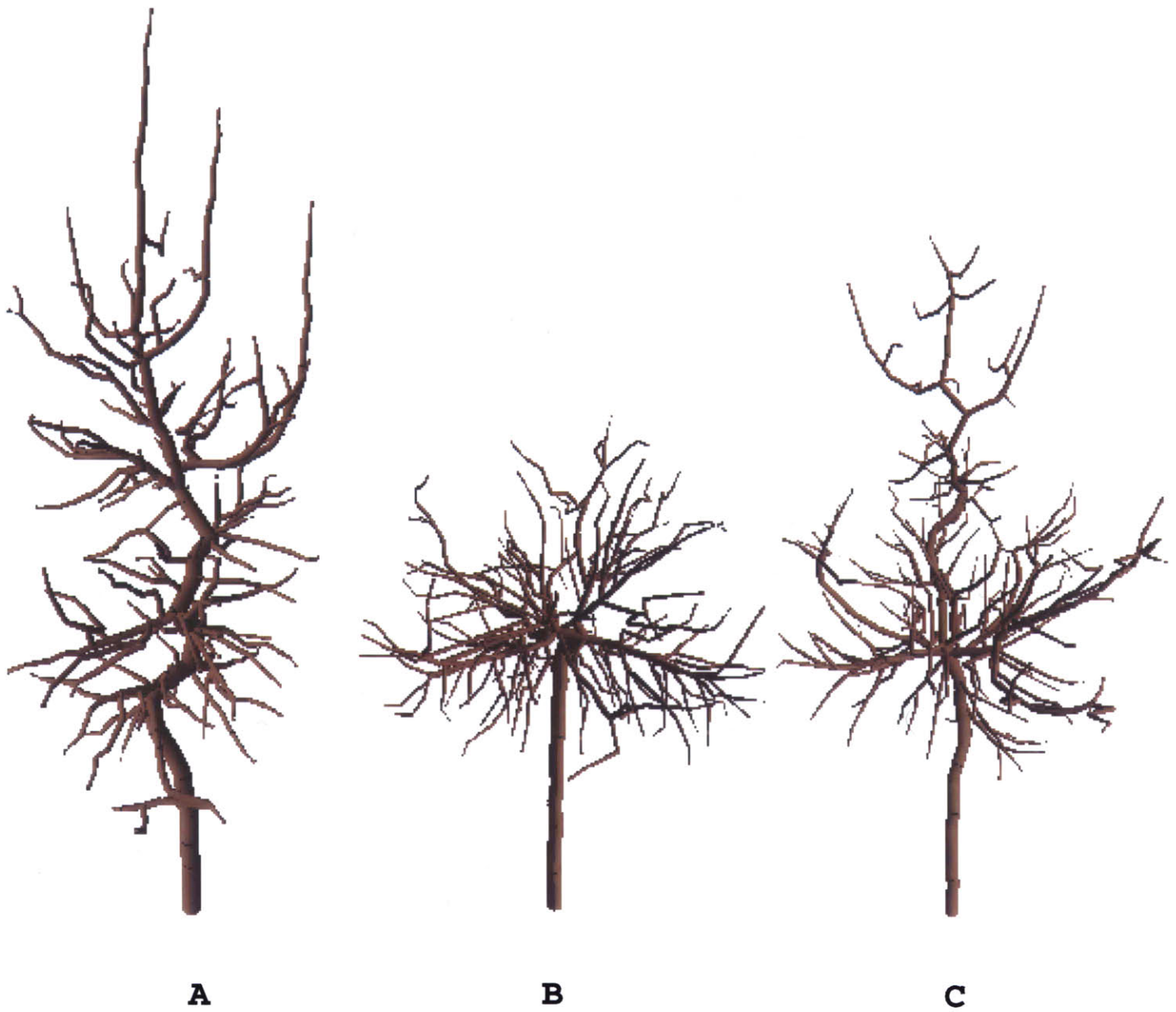


Plate 1: *Three examples of tree skeletons constructed by finding an approximation to the medial axis of the tree shape. The branch thickness is computed in the L-system.*



Plate 2: Comparison of reconstructed 3D model **A** (*right*) to the original tree (*left*). The viewpoints are the same.



Plate 3: Comparison of reconstructed 3D model **B** (*above, right*) to the original tree (*above, left*), and reconstructed 3D model **C** (*below, right*) to its original (*below, left*). The viewpoints for each pair of views are the same.



Plate 4. The tree-specific segmentation algorithm.
Top: Original photograph containing a tree
Bottom: Pixels classified as belonging to the tree



Plate 5. Another example of tree-specific segmentation.
Top: Original photograph containing a tree
Bottom: Pixels classified as belonging to the tree