Shape Modeling of Plant Leaves with Unstructured Meshes

by

Sung Min Hong

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2005

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The plant leaf is one of the most challenging natural objects to be realistically depicted by computer graphics due to its complex morphological and optical characteristics. Although many studies have been done on plant modeling, previous research on leaf modeling required for close-up realistic plant images is very rare. In this thesis, a novel method for modeling of the leaf shape based on the leaf venation is presented. As the first step of the method, the leaf domain is defined by the enclosure of the leaf boundary. Second, the leaf venation is interactively modeled as a hierarchical skeleton based on the actual leaf image. Third, the leaf domain is triangulated with the skeleton as constraints. The skeleton is articulated with nodes on the skeleton. Fourth, the skeleton is interactively transformed to a specific shape. A user can manipulate the skeleton using two methods which are complementary to each other: one controls individual joints on the skeleton while the other controls the skeleton through an intermediate spline curve. Finally, the leaf blade shape is deformed to conform to the skeleton by interpolation. An interactive modeler was developed to help a user to model a leaf shape interactively and several leaves were modeled by the interactive modeler. The ray-traced rendering images demonstrate that the proposed method is effective in the leaf shape modeling.

Acknowledgments

First of all, I would like to thank my supervisor, Prof. Bruce Simpson for helping me to start my master study and guiding me to complete my master thesis. I also would like to express special thanks to my co-supervisor, Prof. Gladimir Baranoski for encouraging me whenever I felt frustrated and providing me with invaluable suggestions. I also would like to express my gratitude to Prof. Peter Forsyth and Prof. Justin Wan for kindly accepting to be part of my thesis committee and for their encouraging comments on my thesis. I also would like to express my gratitude to Prof. Jeff Orchard for giving me many pieces of valuable advice on my work. I would like to thank all the members of the Scientific Computing Laboratory in the School of Computer Science for providing an enjoyable atmosphere to work in.

I cannot thank enough my family for having supported me financially and mentally with endless patience.

Thanks to all of you for reading this. I hope you will read on ...

Sung Min Hong,
Waterloo - April 2005.

Dedication

To my parents and my wife

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Much of the research involved in computer graphics has focused on creating realistic images that mimic the world we see around us. Especially, techniques for modeling and rendering plants have been extensively explored. However, most of plant models based on L-system[1] [27] are focused on overall appearances instead of detail modeling. It is partly because computing hardware has not been adequate to provide realistic close-up images of plant organs like foliage and also partly because the biological processes affecting leaf shape and growth are poorly understood. It is only recently that powerful computer hardware has become available to achieve realistic foliage modeling, which has been relatively undeveloped [21].

In this thesis, a new shape[2] modeling method for plant leaves, which is based on the control of leaf venation, is proposed. For many dicot[3] leaves, leaf venation architecture is closely related to leaf structural stability [28]. That is, the deformation of leaf venation is believed to cause deformation of leaf blade[4]. On the other hand, the leaf blade deformation may be responsible for the deformation of the leaf venation. In either case, since the leaf shape conforms to the leaf venation, if the leaf venation is used as a skeleton for shaping the leaf, the deformed leaf shape can be more biologically faithful. This is the basic idea of the venation based leaf shape modeling.

Leaf shape modeling starts from extracting the leaf outline from the leaf image,

[1] An L-system or Lindenmayer system is a formal grammar (a set of rules and symbols) most famously used to model the growth processes of plant development, though able to model the morphology of a variety of organisms.

[2] In this context, 'shape' means 3D surface.

[3] See Appendix A.

[4] See Figure 3.1.

which is obtained in terms of a closed continuous line. Then, a leaf venation is modeled interactively because any automatic extraction method of the leaf venation from the leaf image is not known up to now to author's knowledge. The veins are approximated with piecewise continuous lines and the leaf venation model becomes a hierarchical tree structure of which each node corresponds to a vein. This venation model is basically a geometric skeleton [6], so an articulated skeleton is constructed from the geometric skeleton for manipulating the leaf shape. After the leaf venation is constructed, the leaf domain is triangulated using the venation as constraints for the triangulation. As the meshes used for triangulating the leaf domain become finer, the number of joints in the skeleton increases. Too many joints in a vein make the interactive shape modeling of the vein more complicates. A new method of deforming the articulated skeleton, which uses only small number of joints, is presented in this thesis.

The leaf venation is deformed interactively by way of the articulated skeleton, and then the leaf blade deformation is calculated using a finite volume method (FVM) [33], where the deformed leaf venation and the leaf boundary are used as Dirichlet[5] and Neumann[6] boundary conditions, respectively. As the venation shape varies, the resulting leaf shape also varies conforming to the venation. Consequently, the deformation of the leaf shape is faithful to the biological characteristics and rendering the leaf with the deformed mesh model can give a very realistic image. The leaf blade, which is modeled as unstructured meshes, is rendered triangle by triangle and the leaf venation, which is modeled as a skeleton, is rendered using generalized cylinders. Since this leaf model starts from a real leaf image, the image can be used as a texture image for the leaf blade, which makes the rendered image even more realistic.

An interactive modeler was developed to facilitate the leaf shape modeling, which carries out the tasks of extracting the leaf boundary, modeling the venation, triangulating the leaf domain, deforming the venation and deforming the leaf blade. Among the modeling processes, modeling and deforming the venation are time consuming processes and can be done only interactively. The modeler provides three orthogonal viewing windows and one perspective viewing window, which is a popular layout many 3D modelers adopt. With real-time OpenGL[35] rendering, the modeler can show the user how the change looks whenever the leaf model is changed, so the user can respond instantly. Figure 1.1 shows a close-up image of three leaves, which are modeled using the interactive modeler and rendered by

[5]A Dirichlet boundary condition specifies the values a solution is to take on the boundary of the domain [26].

[6]A Neumann boundary condition specifies the values the derivative of a solution is to take on the boundary of the domain [26].

an off-line ray tracer. The realistic looking details in the image demonstrates the capabilities of the interactive modeler and the effectiveness of the proposed method.

In Chapter 2, previous work related to the leaf modeling are briefly introduced. In Chapter 3, some basic venation characteristics of plant leaves are given because that knowledge is required to modeling the leaf venation, some botanical terminologies are briefly explained. Chapter 4 provides a summary overview of modeling processes briefly. Chapter 5, 6 and 7 explain the modeling processes in detail. In Chapter 5, the scanning image and extracting boundary processes are explained in detail. In Chapter 6, how the leaf venation is modeled and then how the leaf domain is triangulated, are described. In Chapter 7, two methods for deforming the leaf venation are given and then how to deform the leaf shape is described. Chapter 8 gives a brief description of the interactive modeler and then several renderings of plant leaves obtained using the method proposed in this thesis and the interactive modeler are shown. Finally, Chapter 9 gives brief concluding remarks and makes comments on possible future works.

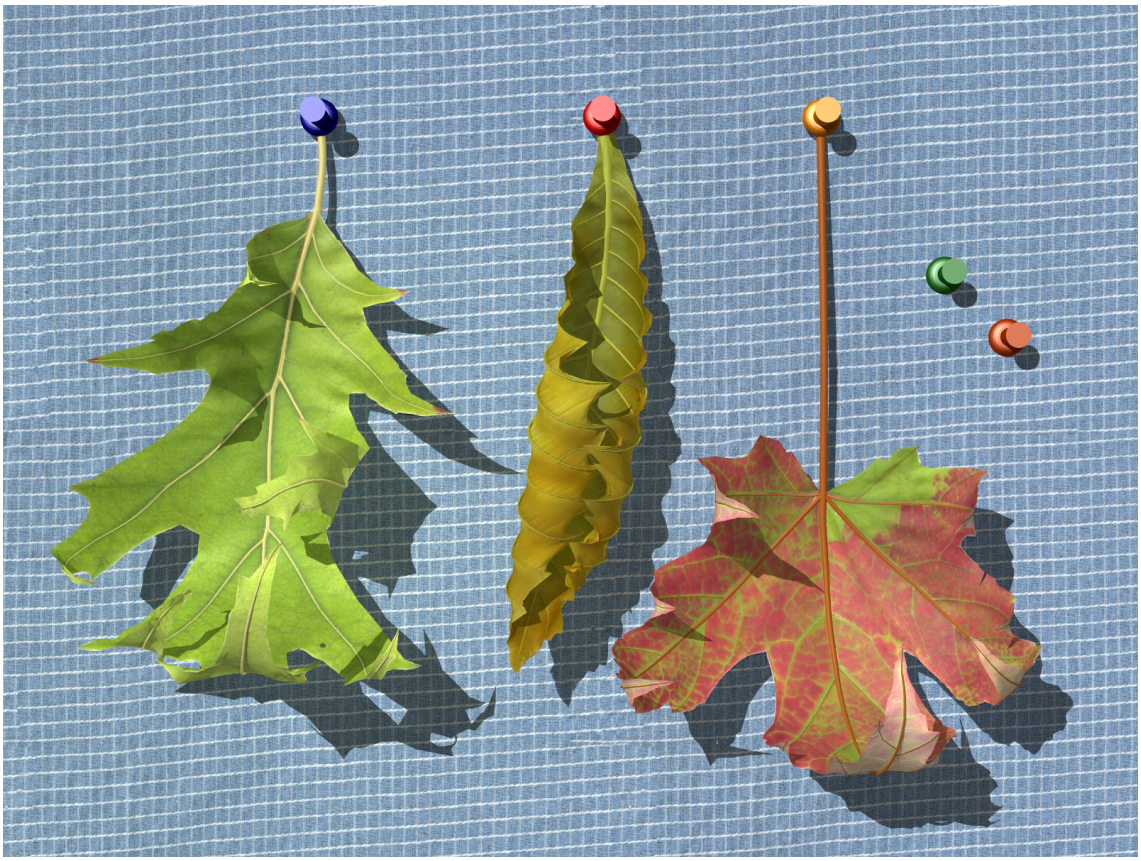


Figure 1.1: Three dried leaves, which are an oak, a chestnut and a maple leaves from the left, hanging on the board

Chapter 2

Related Work

The realistic modeling of vegetation is an important area in computer graphics. In particular, tree modeling and rendering have been an active part of computer graphics research for last two decades because trees add a significant power of realism to a scene. Most difficulties related to tree modeling originate from biological diversity and complexity. For example, there exist many branching patterns [27] in trees. Leaf shapes are most diverse, so especially 2D leaf profiles are used to classify trees. Moreover, even in a tree, the leaves may have different shapes depending on their ages and functions. Consequently, leaf shape modeling faithful to its biological characteristics has been difficult and is still a challenging research area.

In 1985, Bloomenthal [5] made a complete computer model for a mature maple tree, where a leaf texture map is obtained by scanning a leaf photograph. The leaf veins are retouched by a painting program to give a dramatic effect. The resultant color texture is mapped onto a three-polygon structure, where polygons are neighboring along two folding lines in the leaf. The folding degree depends on the strength of a hypothetical wind as well as the extent to which the leaf faces the wind. The crease at hinge is not visible if the structure is Phong shaded[1]. The shadow side of the leaf is shaded with light colors to simulate the lighter underside of the maple leaf. Although the overall appearance of the maple tree looks good, that is, similar to a real maple tree, a close-up image of the leaves is far from being realistic.

Hammel et al. [14] used a branching skeleton, which is generated by an L-system [27], for modeling lobes in compound leaves. Then, the leaf margin is constructed using an implicit function along the skeleton. The leaf object composed

[1]A method of shading surfaces in which the incident light color and intensity are calculated for a series of points along each edge of a polygon and then interpolated across the entire polygon [11].

of the skeleton and the surface can be deformed to give a realistic leaf appearance. The procedure for creating compound leaves can be fully automated, however, the quality of the creation mechanism depends on how faithfully a branching skeleton and a leaf margin[2] are generated. If the skeleton and the margin are purely geometric, then the resultant leaf shape cannot be realistic. Therefore the paper also mentions that among the physical characteristics of leaves for shape modeling, venation patterns and dentations need further consideration for realistic shape modeling.

Streit and Heidrich [31] parameterized the structure of individual feathers based on biological structure and substructures of actual feathers. With the parameterization, they generated a large variety of feathers at multiple levels of detail. Several similarities between a feather and a leaf exist, for example, a rachis[3] and barbs[4] in a feather resemble a midrib[5] and secondary veins in a leaf, respectively. However, feather barbs are more dense than secondary veins in leaves and vein shapes in plant leaves are generally much more complicate than feather barbs. Therefore, some key barbs can be modeled as a low order polynomial and other barbs are interpolated using the key barbs but such interpolation may not be applied to secondary veins because the vein shapes are a lot more independent unlike the feather barbs. Besides, feather barbs have no branches. However, secondary veins have child veins, so the interpolation scheme cannot be used for the leaf veins because the higher order veins cannot be interpolated with the proposed method.

Maierhofer and Tobler [19] developed an interactive plant modeling system. In the model, leaves are modeled from photographs instead of scanned images. It seems that they tried to avoid difficulties in scanning leaves because sometimes 2D scanning is not possible for plant leaves. However, it is almost certain that using only photographs taken with an oblique angle cannot give correct texture maps which are used for rendering. The parameters used in the model are a leaf outline, a main axis, axial and lateral cross section data, so these are not closely related with physical characteristics of plant leaves. Therefore, the output of the modeling scheme may not be faithful to real leaf shapes.

Bloomenthal and Lim [6] describes a general concept of manipulating a general shape using skeleton embedded in the shape. The scheme is composed of 3 steps. First, the geometrical skeleton is derived from a static object using Delau-

[2]See Figure 3.1.

[3]A main axis or shaft, such as the main stem of an inflorescence, the stalk of a pinnately compound leaf, or the spinal column [1].

[4]Parallel filaments projecting from the main shaft of a feather [1].

[5]See Figure 3.1.

nay triangulation[6] and so forth. Second, an articulated skeleton is derived from the geometric skeleton and used to manipulate the geometric skeleton because this geometric skeleton itself is not adjustable. After the articulated skeleton is modified, the geometric skeleton is accordingly modified, then the shape is transformed using appropriate shape deformation methods. In [6], they demonstrated how to deform a horn-like 3D shape using the above scheme.

Unlike the method used in [14], Mündermann et al. [21] proposed a leaf shape modeling method which starts from scanning a leaf. That is, in the previous paper [14], the leaf shape is modeled with a branching skeleton generated by an L-system, however, in [21], the branching skeleton is obtained as a medial axis[7] from the scanned leaf image. Instead of using implicit functions for a leaf margin, a contour extracted from the scanned image is used for the leaf margin. The space between the medial axes and the margin is filled with rail-like rectangular panels. The leaf shape can be deformed by modifying the branching skeleton and rotating the rail-like panels around the skeleton. Although this method can be almost automated because the leaf skeleton is mathematically constructed as medial axes of the leaf outline, the skeleton represented by medial axes does not seem to be able to replace the real venation. Because real leaves are not mathematically constructed, that is, real leaf blades are generally asymmetric about their venation, the venation patterns do not match such geometrically constructed skeletons, so the deformation with the skeleton deviates from the realistic deformation existing in natural leaves.

The method used in this thesis adopts the general idea[8] given in [6]. The geometrical skeleton corresponds to the venation of a leaf which is interactively modeled and the articulated skeleton can be the venation or the simplified venation with reduced number of nodes, which is explained in Chapter 6 and 7. After the venation is modified, the leaf shape is transformed using the FVM for conforming to the modified venation.

[6] See § 6.2.

[7] The medial axis of a 2D region is defined as the locus of the center of all the maximal inscribed circle of the object.

[8] Bloomenthal and Lim [6] roughly describe how skeletons are used for transforming 3D shapes , but there is no detail explanation about how to find the skeletons and how to transform the shapes using the skeletons.

Chapter 3

Leaf Shapes and Venation Patterns

3.1 Leaf Shapes

The enormous diversity of leaf shapes, sizes, forms, and arrangements is the result of plants adapting to conditions in a vast range of habitats. Leaf shape may not be consistent within a species: in some, it depends on a leaf's position on the stem; in others, on whether plants are juvenile or adult. Certain aquatic plants produce one type of leaf under water, another type above the surface [8].

The basic component of a leaf shown in Figure 3.1 is the blade, which is also called lamina. Simple leaves consist of one continuous blade, while compound leaves are divided into separate leaflets. Most leaves are attached to the stem by a slender stalk (petiole), but some, as in the case of many monocotyledons[1], are stalkless (sessile). Although some are divided into separate leaflets, the leaves of most dicotyledons and virtually all monocotyledons consist of a single, flat leaf blade [8].

Simple leaves consist of one blade with a continuous surface. The leaves are classified into shallowly lobed, pinnatifid, or palmately lobed leaves according to varying degrees of lobing [8]. Shallowly lobed and pinnatifid leaves in Figure 3.2 have lobes cut no deeper than halfway to the midrib. Palmately lobed leaves have deeper, more distinct lobes. Compound leaves, which are (d) and (e) in Figure 3.2, have blades that are fully divided into leaflets. In palmate leaves, leaflets arise

[1] See Appendix A.

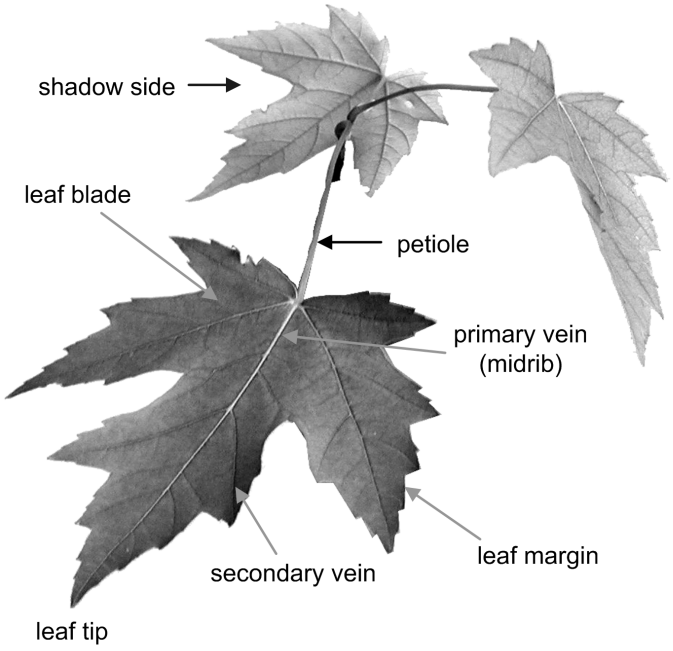


Figure 3.1: External structure of a dicotyledon leaf

from a single point at the top of the leaf stalk. In pinnate leaves, the leaflets arise on both sides of a main axis. The leaflets may be stalkless, and may themselves be subdivided. Two features can help to show that compound leaves are single entities, whatever their size or complexity: in many cases, they are shed as a single unit, and while buds form in the axil of a compound leaf, they do not occur in the axils of individual leaflets [8].

3.2 Leaf Venation Patterns

The classification of an angiospermic[2] venation pattern starts with the primary vein, or, if more than one primary vein is present, with all primary veins entering the leaf from the petiole and the secondary veins branching off the primary veins(s), which are shown in Figure 3.1. Primary and secondary veins are termed the major vein class and represent lower order veins [13]. The classification proceeds with

[2]See Appendix A.

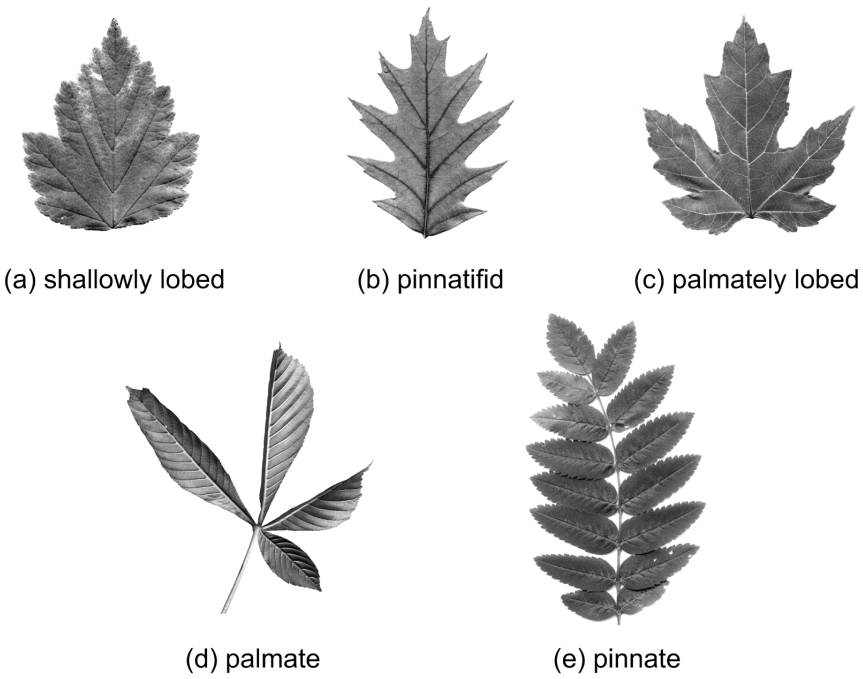


Figure 3.2: Lobing and division of leaves

progressively higher order veins until the areolation[3] which terminates the vein system.

Most angiosperm leaves have between four and seven orders of venation [13]. The primary vein or veins are somewhat analogous to the main trunk or trunks of a tree: they are the widest veins, they usually taper along their length, and they generally run from at, or near, the base of the leaf to the margin. Secondary veins are analogous to the major limbs of a tree. They are the next set in width after the primary(s), they also usually taper along their course, and they ordinarily run from either the base of the leaf or from a primary vein toward the margin [15]. For tertiary and higher order veins the analogy with the branching system of a tree breaks down. Tertiary veins are usually considerably narrower than the secondary set and have courses that connect primary and secondary veins to one another in a similar fashion throughout the leaf. This procedure implicitly utilizes a basic feature of angiospermic leaf venation patterns: the hierarchical organization of veins which is reflected by vein diameter [15].

[3]A small space or interstice in a tissue or part, such as the area bounded by small veins in a leaf [1].

The arrangement of veins of a certain order in relation to other architectural features of the leaf is also important in identifying a certain venation type [15]. The existence of cycles in the arrangement of lower order veins is an important criterion for identifying the leaf venation. That is, if there are cycles in the arrangement of the lower order veins, then it is classified into a closed venation, which is called 'Brochidodromous[4]', otherwise, it is an open venation. There are two kinds of arrangement of lower veins in the open type arrangements depending on whether the lower order veins (especially secondary veins) can reach the leaf margin or not, which are called 'Craspedodromous[5]' and 'Eucamptodromous[6]', respectively. One basic feature of angiospermic venation patterns is their hierarchical character regardless of being open or closed. Figure 3.3 shows the three different types of lower order vein arrangement in the case of a single primary vein.

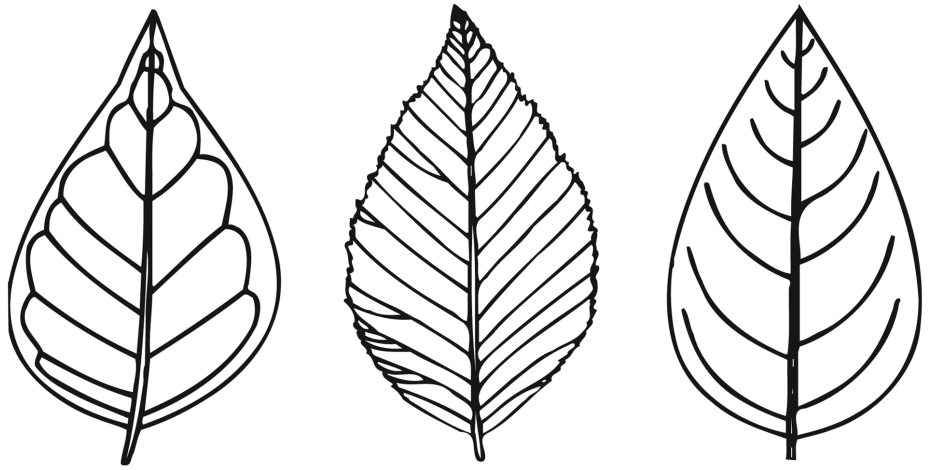


Figure 3.3: Leaf venation types (redrawn from [28])

[4]The secondary veins are joined together in a series of prominent arches [15].
[5]The secondary veins are terminating at the margin [15].
[6]The secondaries are upturned and gradually diminishing apically inside the margin [15].

Chapter 4

Overview of Leaf Shape Modeling

Figure 4.1: Image scanning and binarization

The shape modeling of plant leaves proposed in this thesis starts from a leaf image, which can be obtained by scanning a real leaf. If a real leaf cannot be properly scanned, for example, because it is severely folded, then drawing the leaf in 2D with similar colors may give a possible substitute. The scanned leaf image (Figure 4.1(a)) will be used as a texture map on to a mesh model of the leaf, which will be generated by using this leaf modeling process. This scanning procedure is discussed in § 5.1.1.

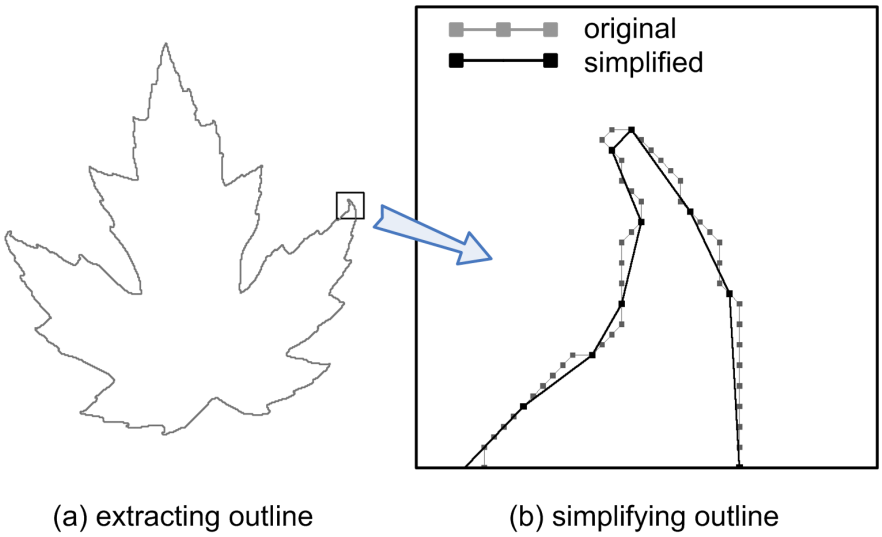
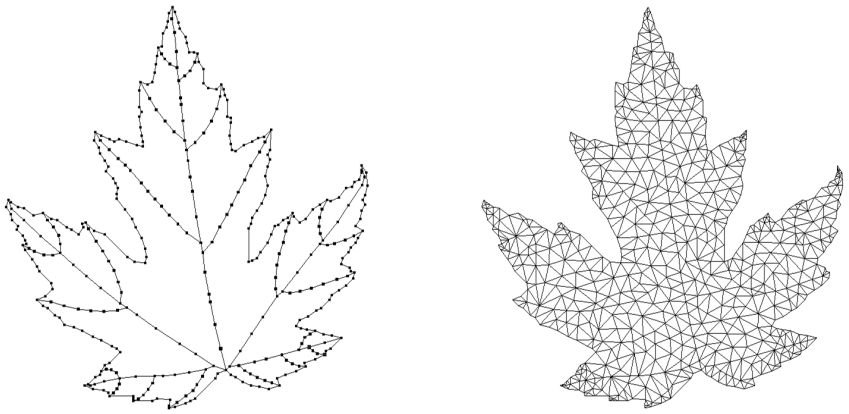


Figure 4.2: Extraction of outline and simplification

The leaf image is also used for extracting its boundary and venation pattern. The leaf boundary is extracted by using a marching squares algorithm, which is a 2D version of the marching cubes algorithm [16, 36]. Before applying the marching squares algorithm to the scanned image, which is scanned in color, is converted to a black and white image in order to obtain a clear boundary (Figure 4.1(b)). The binarization procedure is discussed in § 5.2.1. Once an image of a leaf is obtained, the marching squares algorithm gives a discretized boundary (Figure 4.2(a)). The discretization accuracy of the outline can be adjusted by changing the square size used in the marching squares algorithm. If a large square is used, a coarsely discretized outline is obtained. The discretized boundary from the marching squares has an inherent problem due to digitization, that is, there are some unnatural jagged portions in the outline. Besides that, there may be excessive nodal points in the outline, especially in case of using small squares. A simplification algorithm can be applied to the jagged and unnecessarily densely discretized outline while preserving characteristics of the outline (Figure 4.2(b)). This produces a polygonal curve that approximates the leaf profile. The details are discussed in § 5.2 and § 5.3.

Venation modeling is based on the scanned image of the leaf. A shadow side[1] image may be suitable for the vein creation because venation patterns are more clear in shadow side than sunny side. As described in Appendix A.3, veins are

[1]Although a botanically accurate terminology is 'abaxial surface' in this situation, instead 'shadow side' is used in this thesis. Similarly, 'sunny side' is used instead of 'adaxial surface'.



(a) generating veins (b) triangulating leaf

Figure 4.3: Vein generation and triangulation

worth to be modeled up to 3rd order (tertiary vein) because those kinds of veins are responsible for leaf structure stability[2]. It is based on the assumption that the leaf structural stability is mostly given by the leaf venation. However, if necessary, high order veins may be added to the vein modeling. Vein modeling process starts from generating primary veins, then proceeds to secondary veins, and so forth, in a hierarchical manner(Figure 4.3(a)). This is discussed in § 6.1.

It should be noticed that there are two functions of the leaf venation. The first function is the main idea of this thesis, that is, the venation system provides a skeleton for transforming the leaf blade. It means that the leaf veins are regarded as structural components to support the leaf shape. As the configuration of the leaf venation changes, the leaf blade wrapping the venation changes conforming to the venation. The second function is the appearance of the leaf venation itself. If a vein protrudes considerably from the leaf blade and is big in its size, the vein is modeled as a generalized cylinder with its center displaced from the leaf blade. The vein as a generalized cylinder off-centered from the leaf blade gives both shadowing and masking effects shown in a real leaf. Therefore this venation modeling plays a critical role in making the leaf rendering image look realistic. In this thesis, lower order veins are modeled as generalized cylinders using their radii and an offset value from the leaf blade.

[2]In this context, 'structural stability' means structural strength to maintain the leaf shape.

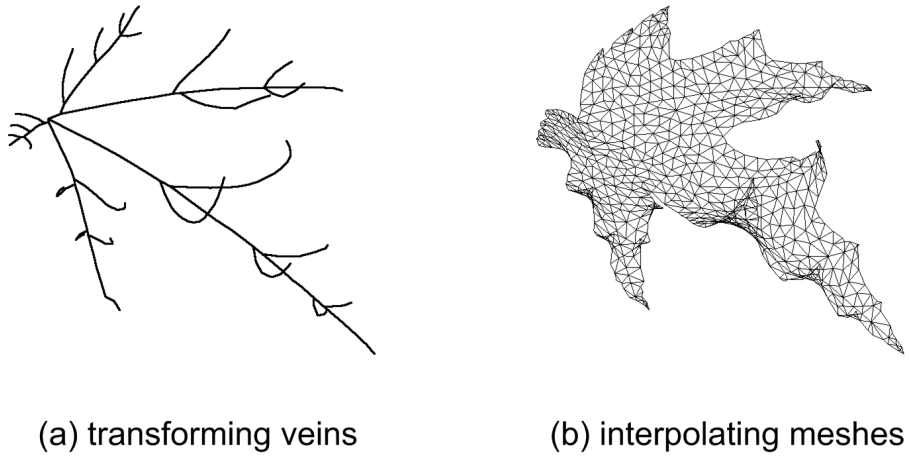
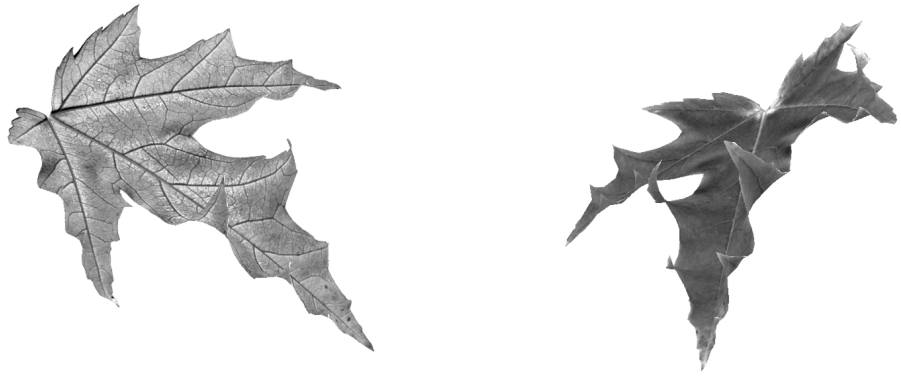


Figure 4.4: Vein transformation and interpolation

Now a leaf domain is defined with a outline and the domain has a set of veins, which are interactively generated. The leaf domain is triangulated (Figure 4.3(b)) with Triangle [29] which is a well known unstructured triangular mesh generation program and is very robust. This triangulation provides the leaf model, which had only a boundary and veins before, with a leaf blade tissue. The leaf outline becomes a domain boundary and the veins become constraints for the Conforming Delaunay Triangulation (CDT) [29]. There are two control parameters for the CDT, which are a minimum angle constraint and maximum area constraint, respectively. These two control parameters are used to generate quality meshes for better rendering because skinny triangles or comparatively big triangles are not desirable for rendering. The triangulation algorithm creates new nodes on both interior and constrained edges where the veins lie while triangulating the leaf domain. After this triangulation, the vein objects are updated by adding the newly generated nodes on the vein edges of the mesh. The detail is given in § 6.2.

The updated leaf venation model is used for deforming the leaf mesh in 3D. This 3D deformation of the leaf mesh can be done in two steps. The first step modifies the vein representation. The second step modifies the mesh that is the leaf blade tissue representation. The vein transformation, which is the first step, can be done by one of the following two ways. The one way is to control every vein segment independently, so numerous configurations are possible but manipulating every segment individually may be tedious because there are usually too many



(a) mapping shadow side texture (b) mapping sunny side texture

Figure 4.5: Rendering with texture mapping

segments in a vein. The other way is to select a small number of control nodes in the vein nodes to construct a spline curve and then use the curve to modify the vein. A modified shape of a leaf venation is shown in Figure 4.4(a). The details are discussed in § 7.1.

The coordinates of the modified veins are used for interpolating the other mesh nodes, which corresponds to the second step mentioned above. The interpolation procedure is based on the FVM [33, 32], which uses Voronoi diagrams obtained when triangulating the leaf domain. In this way, modifying veins and then interpolating the meshes result in a 3D leaf shape (Figure 4.4(b)). This is discussed in § 7.2.

Once a 3D leaf shape in the unstructured meshes is obtained, then the rendering of the leaf can be done instantly using texture mapping in OpenGL, which is shown in Figure 4.5(a) and (b). Since such a rendering by OpenGL is very fast, a user can use it as a preview and improve the leaf shape. If the leaf shape model is good enough to use, then the mesh data can be exported to POV-Ray [24], which is a public domain off-line ray tracer. In general, off-line renders like POV-Ray can provide various rendering effects like soft shadow, so excellent rendering images can be obtained with the mesh data. With the off-line renderer, the leaf venation can be rendered by a bunch of generalized cylinders. And the leaf blade can be rendered with multiple textures, for example, sunny side and shadow side textures.

Besides, if bump map[3] or displacement map[4] data is available, such a mapping can be done on the leaf model depending on the capabilities of the off-line renderer. The examples of the off-line rendering are shown in Chapter 8.

[3]Bump mapping relies on normal perturbations to create the appearance of 'bumps' on the surface of the object [2]; the surface of the object is not changed.

[4]Unlike bump mapping, where the normals are skewed to give an illusion of a bump, displacement map creates real bumps [2].

Chapter 5

Capturing the Leaf Profile

For modeling realistic leaves, it does not seem to be appropriate to generate leaf profiles and textures automatically or manually without referring to actual leaf images because some leaves like maple have very complicate profiles and textures. Therefore, in this thesis, real leaves are used for capturing leaf profiles and textures. A leaf profile is captured as a polygonalized curve in three steps. First, a leaf is properly scanned without any serious wrinkles. Second, the scanned image is binarized to give an accurate boundary profile of the leaf. Finally, the marching squares algorithm is applied to the binarized image to capture the profile.

5.1 Image Scanning

In general, sunny sides[1] of plant leaves are much different from their shadow sides, so both sides of the leaves are required for texture mapping when rendering the leaves. Since the both side images should be aligned, it may be required that the leaf images are rotated and scaled using digital image processing tools. To express details of the leaf images, the scanning should be done in 24 bit RGB[2] color which is corresponding to compound color of 8 bit red, 8 bit green and 8 bit blue. The 24 bit RGB color scheme can have $2^{24} \approx 16$ millon different colors. The image resolution can be determined depending on where the leaf model is used. If a very close-up rendering image is required, then the texture image also should be made in high resolution. In this thesis, the scanning of real leaves is done in 300 DPI[3]

[1] As mentioned in Chapter 4, the sunny side means the adaxial surface of the leaf throughout this thesis. Similarly, the shadow side does the abaxial surface.

[2] 'RGB' is an acronym of 'Red, Green and Blue'.

[3] 'DPI' is an acronym of 'Dots Per Inch'.

(a) sunny side (b) shadow side

Figure 5.1: Sunny and shadow sides of English oak leaf

and 24 bit RGB color scheme. Both images of a silver maple leaf is already shown in the previous chapters. Figure 5.1 shows the sunny and shadow side images of an oak leaf.

5.1.1 Difficulties in Image Scanning

In fact scanning a leaf with a flat bed scanner is not easy because real leaves are not perfect flat. For example, some leaves are considerably curved along their veins and margins of some leaves are severely undulated. Therefore, scanning such a leaf while pressing it with a scanner cover to reduce the leakage of the scanning light may cause the leaf to be wrinkled and result in a stained image with wrinkles. A leaf image with small wrinkles may be acceptable, but if the wrinkles are obvious, then the image cannot be used as a texture image for off-line rendering.

However, even in scanning a relatively flat leaf like a silver maple leaf, there may be another problem to cause the scanned image to be inappropriate for a texture image. If the veins of the leaf seriously extrude from its blade, then some unwanted shadows by the veins appear. These vein shadows are usually not matched with a light direction used in rendering. Such self shadowing from scanning should be avoided but it is often inevitable. Therefore, for some leaves with high blade curvatures, serious marginal undulation or extremely extruded veins, 2D image

scanning may not be possible, which gives an image used for extracting the leaf outline and a texture for rendering. Scanning a leaf should be carefully done to avoid image artifacts mentioned above. Sometimes color corrections of scanned leaf images are required because scanning a real leaf does not give exact same colors like those in a real leaf. The color difference between the scanned image and the real leaf should be minimized in order to make the leaf rendering, when the image is used as a texture, look realistic.

5.2 Extraction of Leaf Profile

The marching squares algorithm used in this thesis is a modified 2D version of the well known "Marching Cubes" algorithm [16, 36]. Here is an overview of the marching squares algorithm applied to an image of a leaf in order to extract its outline. The basic assumption of the application is that a leaf profile can be approximated by a polygon, that is, a closed curve. Any holes in a leaf blade are not considered in extracting the outline because such holes can be treated by using an appropriate texture map. Initially, a square scans the leaf image from the left-top corner to the right bottom corner until at least one of its corners runs into the leaf image domain. When the square crosses the leaf boundary, then the square starts to walk along the boundary of the leaf domain counter-clockwise instead of the parallel line scanning. Since the leaf domain is assumed a solid continuous domain, if the size of the square is appropriate, this marching squares algorithm gives a profile close to the actual leaf image. The outline of the leaf image depends on the square size used in the algorithm because the polygon edge can be one of an edge or a half diagonal of the square. The detailed description will be given in § 5.2.2.

However, the scanned leaf image in 24 bit RGB color scheme may not be appropriate for extracting its boundary because the marching square algorithm determines whether a corner the square is in the leaf image or not according to the value of the image pixel lying on the corner. Therefore, the color image needs to be converted into a black and white image where an image pixel can have only two values, that is, 0 or 1. Before the marching square method is applied to the scanned leaf image, the color image is binarized to make the leaf image domain clear from the background.

5.2.1 Image Binarization

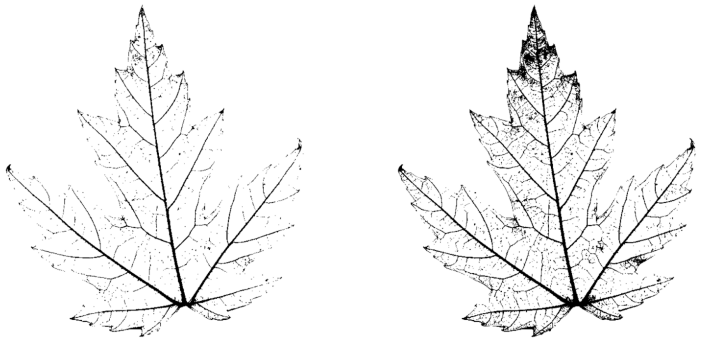
The purpose of the binarization is to make it easy and clear to decide which pixel in a leaf image belongs to a leaf or not. In image binarization, any pixel in a leaf image of which value satisfying a criterion is converted to a black pixel and anything else to a white pixel, so after binarization process the leaf image becomes a black and white image. If an image pixel has only two values instead of $2^{24} \approx 16$ million values, then such decision will be simple and clear, which is a general benefit of image binarization.

Such conversion depends on a threshold value, which is used a criterion for the conversion. A criterion used for the image binarization is called a binary threshold. If the threshold value set for the image binarization is greater than an average color value of a pixel, then the pixel is converted to a black pixel, otherwise the pixel becomes a white pixel. Figure 5.2 show the effect of the different binary threshold values on the image binarization. Although a high threshold value is desirable for the maple leaf shown in Figure 5.2, such a high threshold value may show some background noise in other cases, so a proper value should be used.

5.2.2 Marching Squares Algorithm

The marching squares algorithm used in this thesis is a specialized algorithm for extracting an outline from a binarized 2D leaf image. The basic notion is that a square is defined by the pixel values at its four corners. If one, two or three corner pixels are black, the square must cross the boundary of the binarized leaf image. If no corners are black, then the square must lie in the exterior of the leaf image. In the meantime, if all corners are black, then the entire square must lie on the leaf image. By determining which edges of the square are intersected by the boundary, a segment of the leaf outline is created. By connecting these segments obtained while the square tracing along the leaf image boundary, an outline is extracted.

In the marching squares algorithm, the possible number of outline segment configurations is $2^4 (= 16)$ because one corner can have only one of two values, that is, a black or a white pixel. If all corners of a square are totally white or totally black, there is no boundary segment crossing the edges of the square. For the remaining 14 cases, there exists one or two boundary segments crossing the square edges, which are shown in Figure 5.3. Among them, the two crossing patterns, (7) and (10) in Figure 5.3 are ambiguous cases. Although the pattern (7) shows two separate boundary segments connecting edges 1-2 and 3-4, there may be another



(a) threshold value = 100

(b) threshold value = 110

(c) threshold value = 120

(d) threshold value = 130

(e) threshold value = 140

(f) threshold value = 150

Figure 5.2: Effect of threshold value on image binarization

crossing pattern with two boundary segments connecting edges 1-3 and edges 2-4, which is shown in Figure 5.4. In this thesis, only solid continuous images are

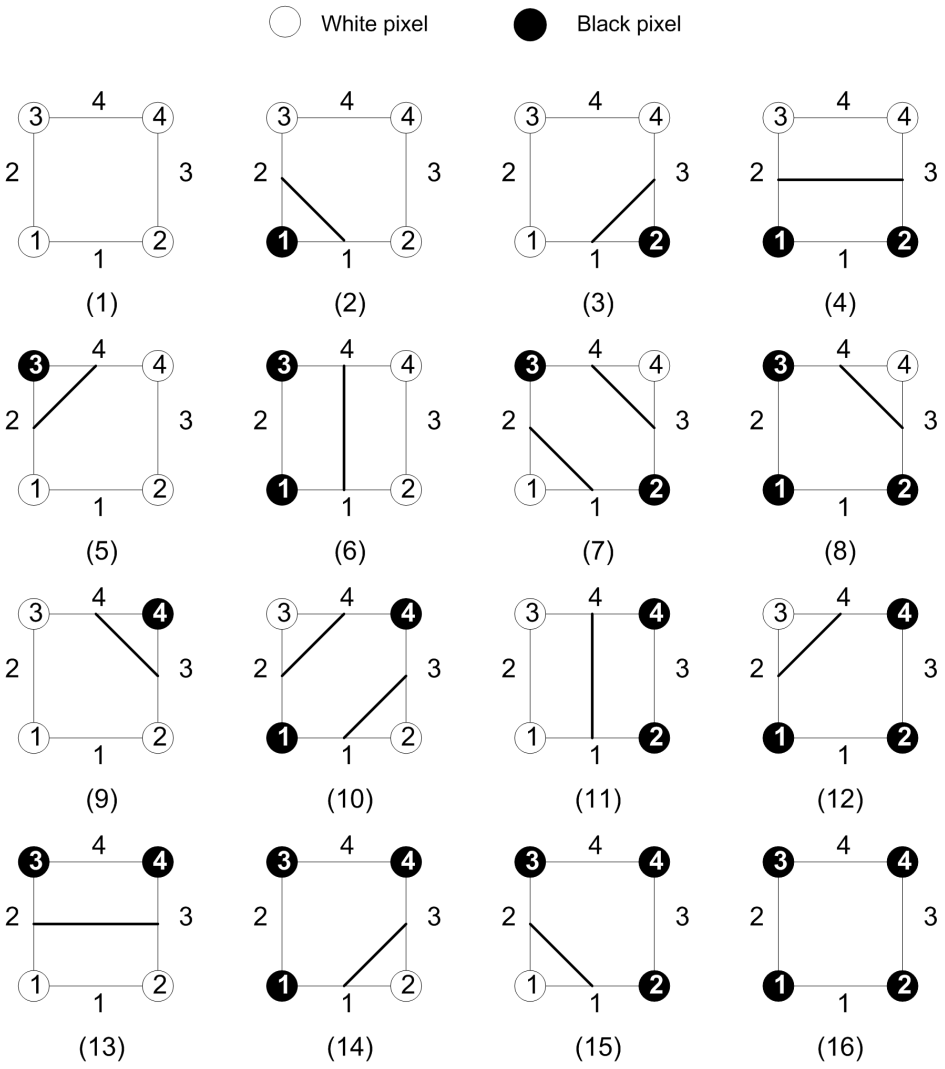


Figure 5.3: Crossing patterns of marching squares algorithm

considered, so such ambiguity can be avoided by not allowing the crossing patterns like (7) and (10) when querying a crossing pattern. Although it is possible to have (7) or (10) pattern for a narrow image region, then reducing the size of the marching square can eliminate such ambiguous patterns because the crossing pattern with the small size square may change .

As mentioned earlier, a square does not march the whole image area but traces a boundary of a leaf, which is shown in Figure 5.5. At the instant the square crosses

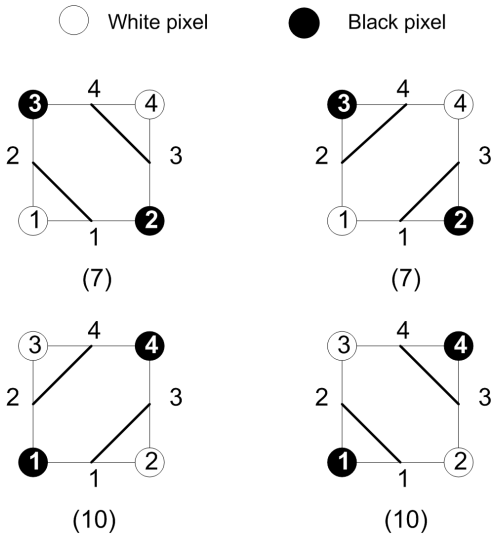


Figure 5.4: Ambiguous patterns

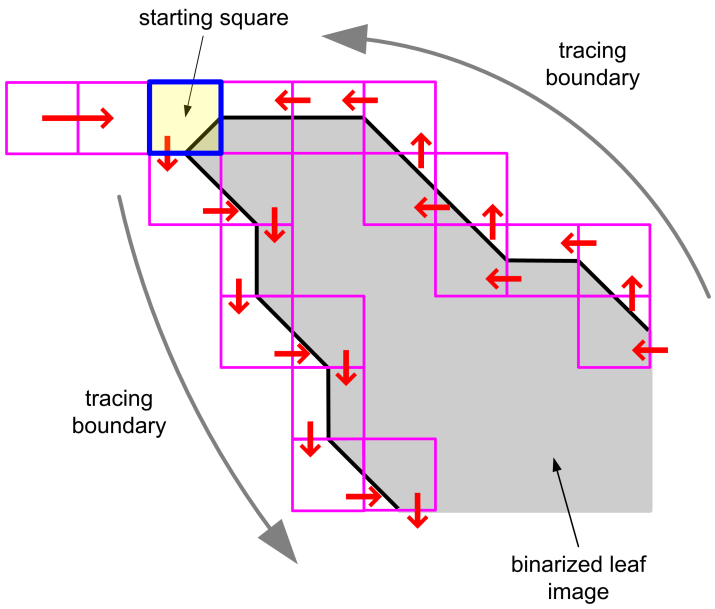


Figure 5.5: Tracing boundary with marching squares

the leaf image boundary during running from the top-left corner to the bottom-right in line by line scanning, the square starts to walk along the boundary of the image

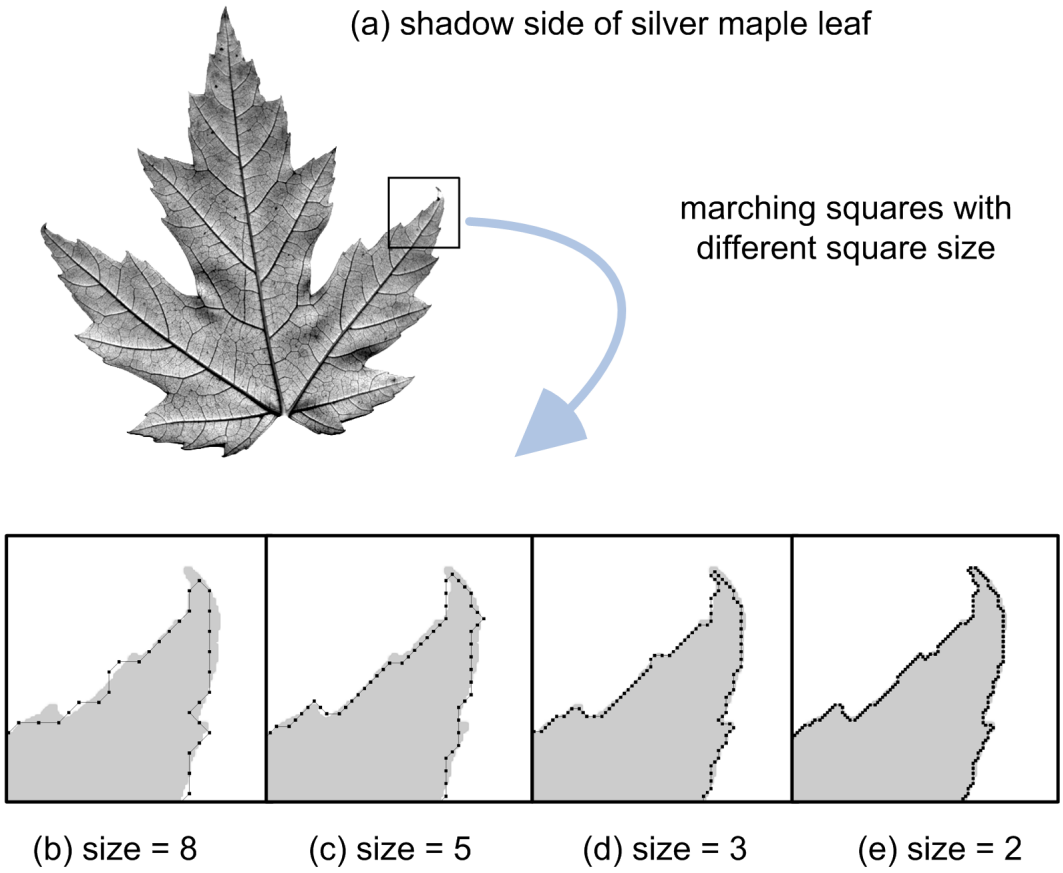


Figure 5.6: Application of marching squares algorithm to silver maple leaf

globally counter-clockwise. For example, if the crossing pattern of the square is (3) in Figure 5.3 when the square run into the leaf image, a following square should be neighboring at the bottom edge of the current square and the crossing pattern of the following square should be one of (5)-(10) because the following square must have a crossing at its top edge. The tailored marching squares algorithm is summarized as follows.

Step 1. Select the size of the square which is used for tracing the boundary. If the size is too big, then it cannot trace the boundary curve with high curvatures, so the details of the boundary will be lost.

Step 2. Start scanning the binary image with the square from the top-left corner

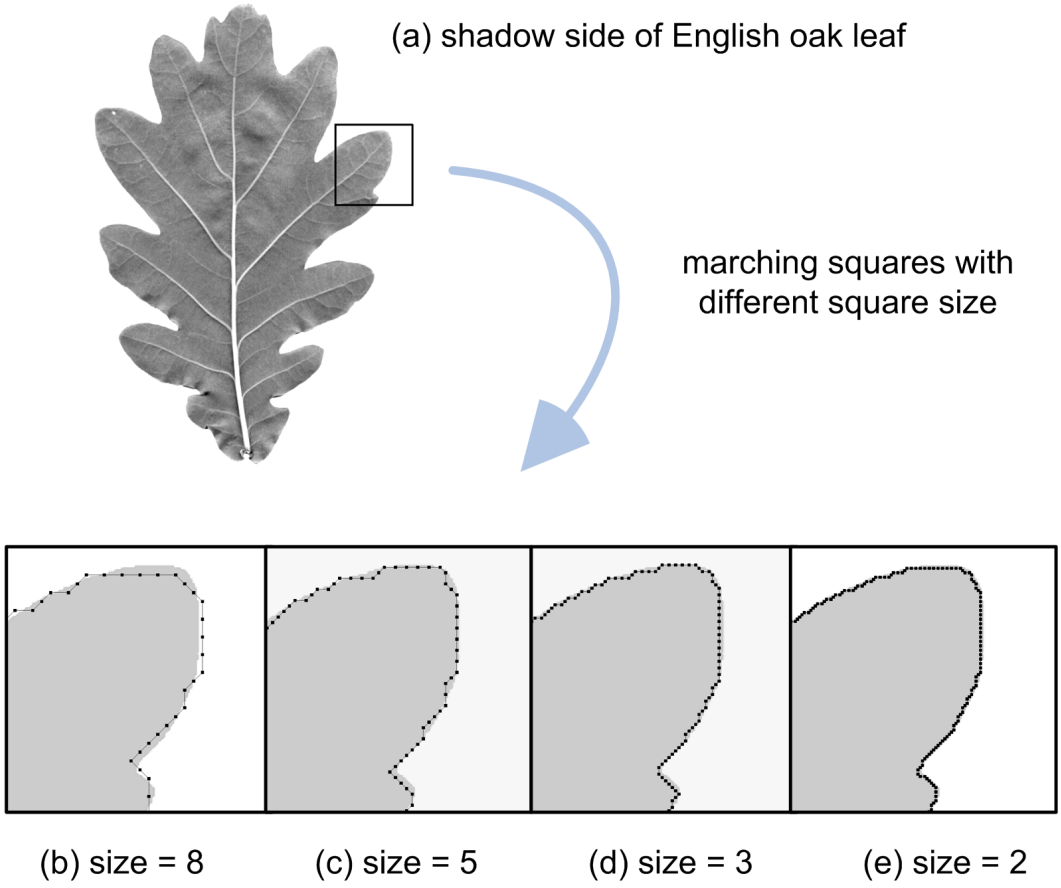


Figure 5.7: Application of marching squares algorithm to English oak leaf

until it runs into the boundary, which is shown in Figure 5.5.

Step 3. Instead of moving the square from the top-left to bottom-right, move the square along the boundary using the crossing pattern shown in Figure 5.3.

Step 4. If the boundary segments make a closed loop, stop tracing.

The above algorithm is applied to a silver maple leaf and a English oak leaf to extract their outlines with different square sizes. Both leaves have multiple lobes, but a silver maple leaf has acutely angled lobes while an English oak leaf has round lobes. The resultant outlines of the silver maple leaf and the English oak leaf are shown in Figure 5.6 and Figure 5.7, respectively. As previously mentioned,

Table 5.1: Effects of the square size on the number of nodes.

Square size (pixel unit)	Number of nodes	
	Maple leaf	Oak leaf
8	512	449
5	849	735
3	1,433	1,229
2	2,155	1,845

when small squares are used, detailed outlines are obtained for both leaves no matter how complicated those are, but small squares increase the number of nodes in the profile and the complexity of the profile. These adverse effects of using small squares are clearly shown in Figure 5.6 and Figure 5.7. Table 5.1 shows how many boundary nodes are generated as the square size decreases in a quantitative manner. As expected, as the square size decreases, the number of nodes increases proportionally. That is, if the square size is reduced in half, the number of nodes is almost doubled. How to reduce the complexity while preserving prominent outline characteristics will be addressed in the following section.

5.2.3 Algorithm Details

The marching squares algorithm uses a special look-up table for querying crossing edge patterns, which is called "`lineTable`". The look-up table maps the vertices of the marching square to the intersecting edges shown in Figure 5.3. The table uses a 4 bit index in binary called "`squareIndex`". Each bit in the "`squareIndex`" corresponds to a vertex. For example, if the index in binary is "0010", then it means that vertex 2 is in the leaf domain. In other words, a pixel corresponding to vertex 2 is black and others are white, so edges connecting to vertex 2 cross the leaf boundary. The crossing pattern is equivalent to (3) in Figure 5.3. Now let `imagePixel[i][j]` be the brightness of the *i-th* row and *j-th* column pixel. A white pixel has a brightness value of 255 and a black pixel does a value of 0. Then, `squareIndex` of the square is obtained as follows.

```
squareIndex = 0;
if(pixels[i][j] < 128) squareIndex += 1;
if(pixels[i + squareSize][j] < 128) squareIndex += 2;
```

```
if(pixels[i][j + squareSize] < 128) squareIndex += 4;
if(pixels[i + squareSize][j + squareSize] < 128) squareIndex += 8;
```

With the above routine and Figure 5.3, the edge crossing pattern look-up table is obtained as follows. The table is a collection of the crossing patterns shown in Figure 5.3, which can be queried with `squareIndex`.

```
int lineTable[16][4] =
    {{0, 0, 0, 0}, {1, 2, 0, 0}, {1, 3, 0, 0}, {2, 3, 0, 0},
    {2, 4, 0, 0}, {1, 4, 0, 0}, {1, 2, 3, 4}, {3, 4, 0, 0},
    {3, 4, 0, 0}, {1, 3, 2, 4}, {1, 4, 0, 0}, {2, 4, 0, 0},
    {2, 3, 0, 0}, {1, 3, 0, 0}, {1, 2, 0, 0}, {0, 0, 0, 0}};
```

Looking up the table with the `squareIndex` returns a set of 4 numbers standing for edge numbers. If only a pixel value of vertex 2 is black, the `squareIndex` becomes 2 and querying the crossing pattern with `lineTable[2][]` gives {1, 3, 0, 0}. This indicates that there is a boundary segment crossing edges 1 and 3. If the `squareIndex` is 7, then querying the look-up table gives {1, 2, 3, 4}. It means there are two edges connecting 1-2 and 3-4, respectively. Since only two kinds of pixels, white and black, are considered, the crossing point should be the midpoint of the crossing edge. Now the procedure of extracting boundary segments for a square can be extended to squares along the leaf image boundary.

5.3 Simplification of Leaf Outlines

If the size of the marching square is small, the obtained boundary curve can be close to the original boundary, but there may be too many points to be used for triangulating the maple leaf. If there are unnecessarily many nodes in the boundary curve, then the excessive nodes may cause extremely fine meshes near the boundary curve, which is not desirable in terms of memory usage and speed. The concept of the simplification algorithm is shown in Figure 5.8. As shown in Figure 5.8, the basic idea of simplification is eliminating intermediate nodes in a straight boundary segment except two end nodes. Based on this idea, the algorithm inherently includes a routine for simplifying a jagged boundary made of many small segments. This is an artifact caused by the marching squares algorithm.

The boundary simplification algorithm proposed in this thesis is based on the concept that newly generated boundary segments are within a preset tolerance from

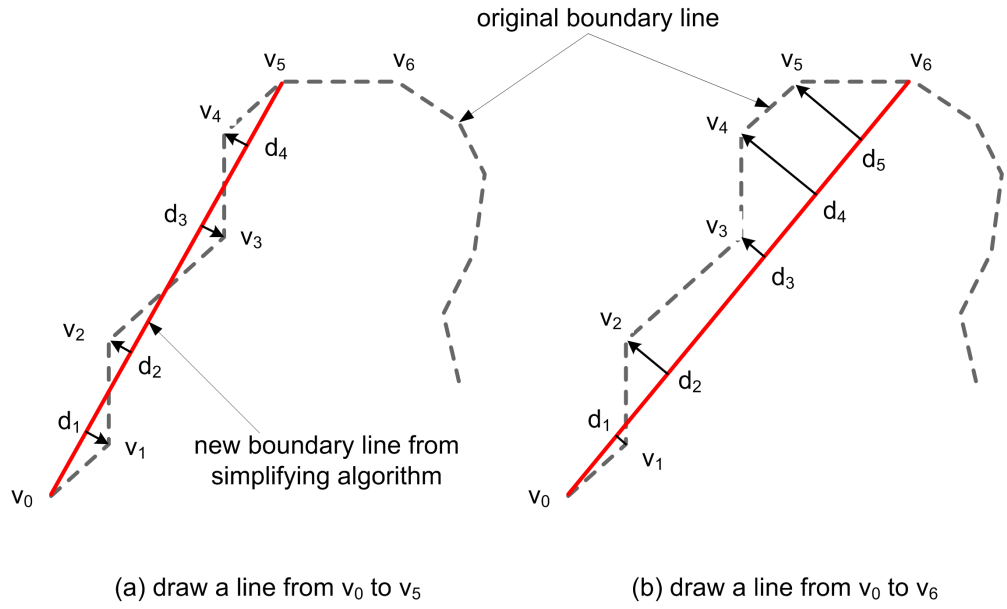


Figure 5.8: Basic concept of boundary simplification

original boundary segments from the marching squares algorithm. At first, two deviation tolerance schemes were examined. The one is limiting a maximum error between newly generated simplifying boundary segments and original boundary segments. The other is limiting a sum of errors between two different boundary segments. The latter scheme gave locally big deviations between the two boundaries although it limited a total accumulation of errors. So the former scheme was chosen for simplifying the boundary segments from the marching squares. The following steps describe the simplification algorithm limiting the maximum error of newly generated boundary segments.

Step 0. Assume that there are n nodes, $v_0, v_1, v_2, \ldots, v_{n-1}$ in the leaf outline obtained by the marching squares algorithm. Let τ be a preset tolerance to limit the difference between the original boundary and the simplified boundary. Let B^s be a container storing the nodes of the simplified boundary.

Step 1. Let i and j be the nodal indices of a boundary segment. Initialize $i \leftarrow 0$ and $j \leftarrow 2$. Put v_i into B^s.

Step 2. If $j = n - 1$, stop the simplification routine. Otherwise calculate a line, ℓ_{ij}, connecting v_i and v_j.

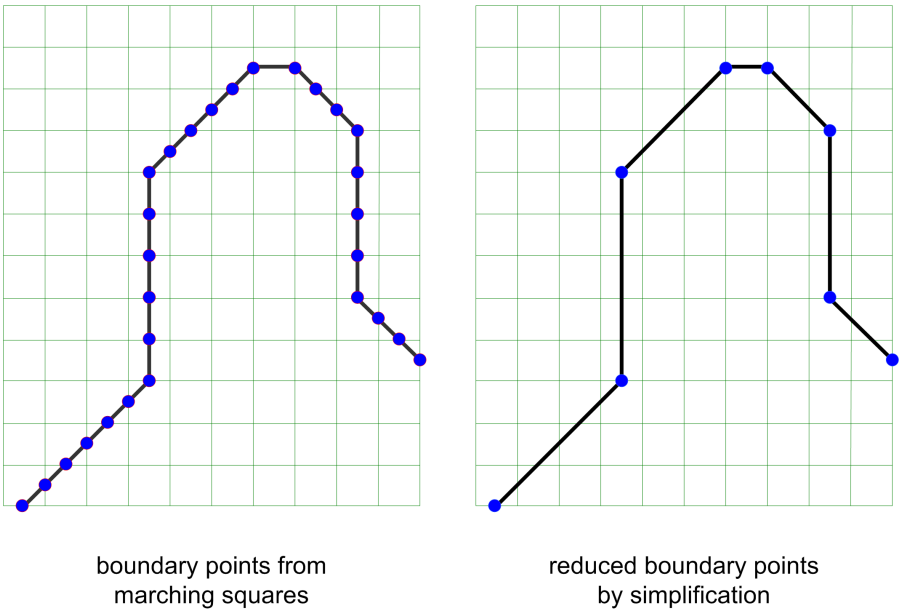


Figure 5.9: Simplification algorithm of boundary segments

Step 3. Calculate the distances from the intermediate nodes, v_k, where $i < k < j$, to ℓ_{ij}. Let the distances be d_k and the maximum among d_k be d_{max}.

Step 4. If $d_{max} < \tau$, then $j \leftarrow j + 1$ and go to step 2. Otherwise, that is, if $d_{max} \geq \tau$, put v_{j-1} into B^s, $i \leftarrow j - 1$, and $j \leftarrow j + 1$. Then, go to step 2.

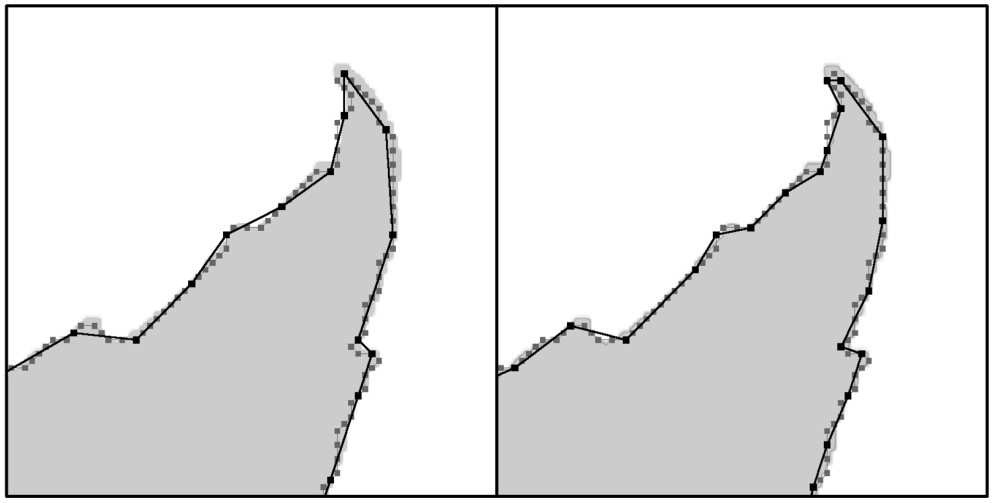
After the simplification algorithm is applied to the original leaf boundary, B^s has the nodes of the simplified boundary. Figure 5.9 explains the above algorithm graphically. The example shown in Figure 5.9 can have a simplified boundary segment from v_0 to v_5 because the difference between the intermediate nodes and the new line segment is limited to the preset tolerance. The next new line segment from v_0 to v_6, on the other hand, does not satisfy this requirement because the difference between the new segment and the original exceeds the tolerance. Now $\overline{v_0 v_5}$ becomes a new simplified boundary segment and the algorithm seeks a next simplified segment starting from v_5 as explained in the above.

This simplification routine can smooth a jagged boundary as well as reduce the number of boundary points obtained from the marching squares. The simplification algorithm is applied to both the outlines of the silver maple leaf and the English oak leaf which are obtained by applying the marching squares algorithm. Table 5.2

Table 5.2: Effects of the deviation tolerance on the number of nodes.

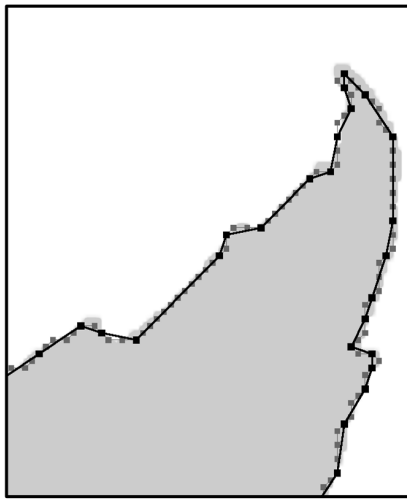
Deviation tolerance (pixel unit)	Number of nodes	
	Maple leaf	Oak leaf
8	45	47
5	72	58
3	125	88
2	195	122
1	493	198

shows how the algorithm simplifies the boundaries quantitatively. According to the table, $80 \sim 90\%$ of the original nodes are reduced in the example cases which use the square size of 3 in pixel unit for the marching squares algorithm and the deviation tolerances of $1 \sim 8$ pixels. An interesting observation is that the reduction ratio of the oak leaf is greater than that of the maple leaf because the outline of the oak leaf is smoother. The simplified boundary segments are shown in Figure 5.10 and Figure 5.11, where the square size is 3 pixels. The Figures show that the simplification algorithm greatly reduces the complexity of the original outlines and smoothes the jagged part. It is noticeable that as the deviation tolerance decrease, the newly generated boundary segments come close to the original boundary segments. Even in the case that the deviation tolerance is 1 pixel, the complexity of the boundary is considerably reduced because intermediate nodes in straight lines are eliminated.

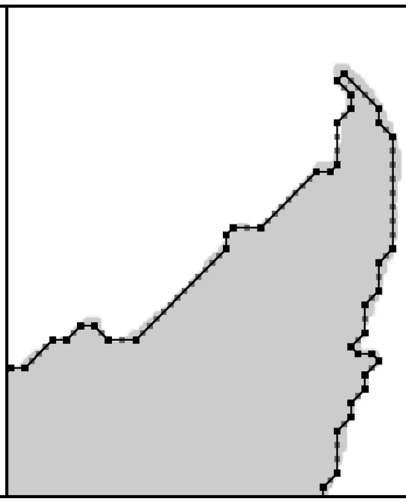


(a) deviation tolerance = 8

(b) deviation tolerance = 5

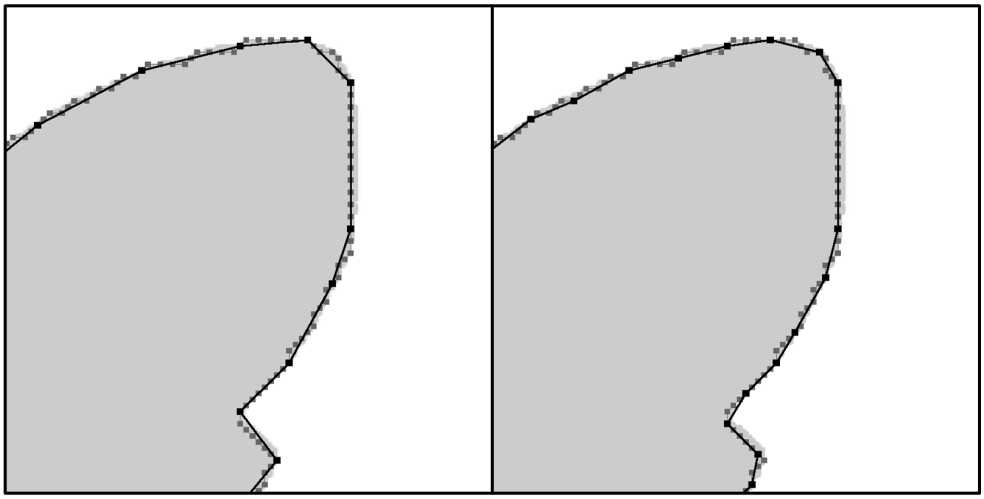


(c) deviation tolerance = 3

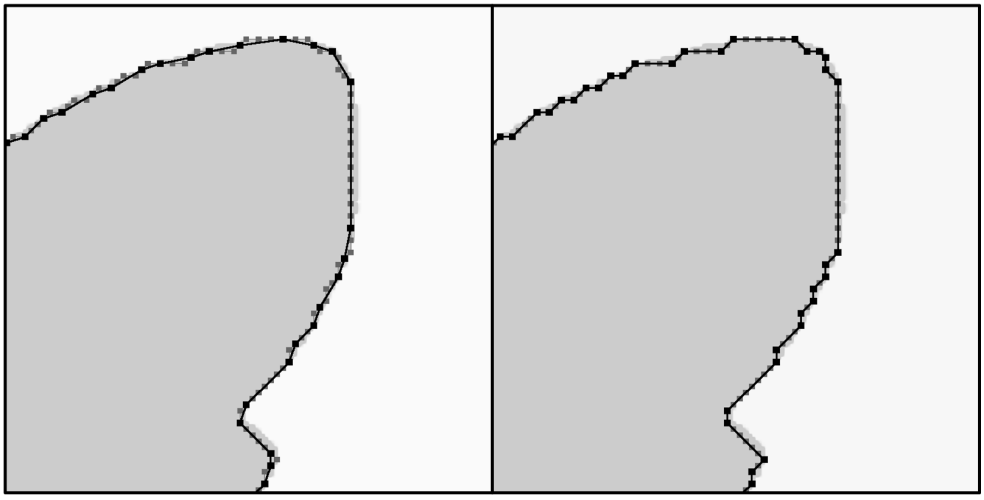


(d) deviation tolerance = 1

Figure 5.10: Effects of deviation tolerance on simplifying silver maple leaf



(a) deviation tolerance = 8 (b) deviation tolerance = 5



(c) deviation tolerance = 3 (d) deviation tolerance = 1

Figure 5.11: Effects of deviation tolerance on simplifying English oak leaf

Chapter 6

Modeling Leaf Venation and Triangulation

A leaf domain is defined with a polygonalized boundary, which is described in the previous chapter. On the leaf domain, a leaf venation is interactively modeled by a user. A vein is modeled as a piecewise linear curve of which nodes are created by a user. Then, the leaf domain is triangulated using the venation as constraints. The triangulated leaf domain and the constrained venation are two important data structures for the leaf shape modeling. The detail processes of modeling venation and triangulation are discussed in this chapter. A comprehensive review of plant venation modeling is beyond the scope of this thesis. The interested readers can refer to a state of art review by Taylor-Hell, et al. [34]

6.1 Modeling Leaf Venation

Modeling of a leaf venation is based on the characteristics of the leaf venation discussed in Chapter 3. First, for dicot leaves, the structural stability generally depends on veins up to third order, that is, primary, secondary, and tertiary veins play dominant roles in the leaf structure because the lower order veins are much stronger than higher order veins. Second, there is a network-like or a tree-like low order venation in a dicot leaf while higher order veins usually make networks. Third, a low order vein structure existing in a dicot leaf is generally a hierarchical structure, so it is natural to model the structure as a hierarchical system. That is, primary veins in a leaf starts from a petiole and secondary veins branches out from the primary veins, and so on (see Figure 3.1).

6.1.1 Characteristics of a Dicot Leaf Venation

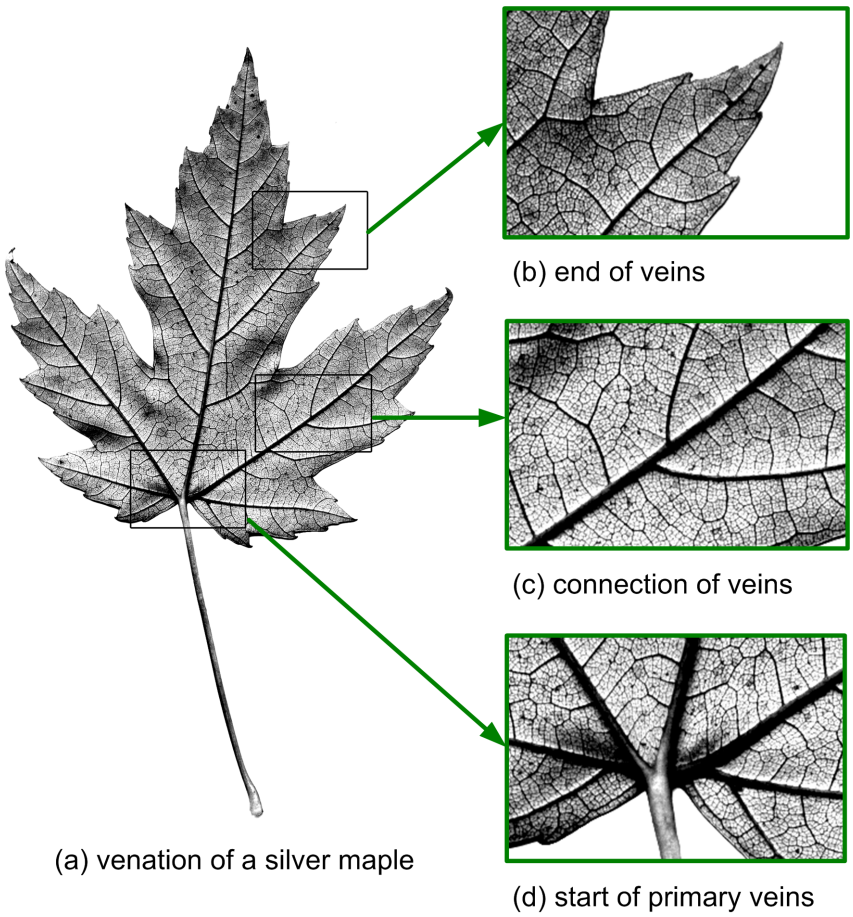


Figure 6.1: Venation of a silver maple leaf (shadow side)

Figure 6.1 shows close-up images of the familiar silver maple leaf, which show all the venation characteristics mentioned above. A silver maple leaf is a palmately lobed leaf, so the mid-veins running through the middle regions of the lobes are primary veins according to the vein classification rule [15]. These veins are bigger than any other veins in the leaf and start from the leaf base, which connects to the petiole, which is shown in Figure 6.1 (a) and (d). Low order veins, especially primary veins, have a lot of fibers below their vascular tissue, so it is worth to notice that the veins bulge toward the shadow side of the leaf and they do not project toward the sunny side of the leaf [20], which is shown in Figure 6.1 (d).

In modeling a leaf, low order veins in the leaf are modeled as generalized cylinders with a radius-varying circular cross section. Low order veins have their biggest radii at where their branches start and smallest radii at the end of their branches when they are modeled as circular generalized cylinders, which is shown in Figure 6.1 (b), (c) and (d). The protrusion of the leaf venation toward the shadow side is modeled by assigning an offset value to each generalized cylinder, so the centerline of every generalized cylinder is off-centered by the given offset value from the leaf blade toward the shadow side.

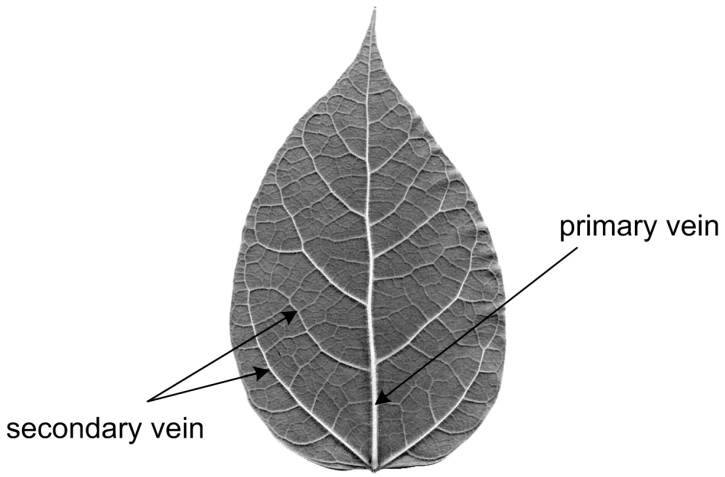


Figure 6.2: Venation of a catalpa leaf

For the silver maple leaf, the venation system can be modeled by a tree-like structure because its low order veins do not normally make any cycles while higher order veins make numerous nets. A hierarchical venation structure is observed in Figure 6.1. Figure 6.1 (c) shows that two secondary veins branch out from the primary vein of the right big side lobe to the small sub-side lobes. Vein orders are usually determined based on such hierarchies, that is, primary veins start from the petiole connection at the leaf blade boundary, secondary veins from the primary veins, and tertiary veins from the secondary veins in the venation modeling proposed in this thesis. It is worth to notice that this kind of classification is different from the one actually used in the botany [15], where the size of the vein is an important factor in vein ordering. In Figure 6.1 (c), the secondary veins have significantly small diameters compared to the diameter of the primary vein they start from. In most dicot leaves, low order veins up to third order almost uniformly spread over

the leaf blade. It means the transformation of those vein curves can affect all the leaf domain. In other words, various leaf shapes can be made by a venation model considering only low order veins. The details are discussed in § 7.1.

A catalpa leaf, however, have a network-like venation because secondary veins are joining instead of reaching to the leaf boundary or just ending, which is shown in Figure 6.2. Such a tree-like or a network-like venation pattern may be modeled as a graph [18]. The cycle in the graph model makes it difficult to regard the vein in the cycle as a secondary vein because the vein has no direction that is a characteristic of the tree-like venation. If the vein in the cycle is to be divided into two veins, it is difficult to distinguish one vein from another vein in the cycle. In addition to such ambiguity, using two veins in a cycle as constraints for triangulation gives a redundant constraint which makes the vein modeling and the triangulation complicated. Therefore, the modeling method proposed in this thesis considers only tree-like venation, that is, no cycle in the leaf venation. Network-like venation existing in a catalpa leaf can be approximated by tree-like graphs to avoid the difficulties mentioned above.

6.1.2 Data Structure of Venation System

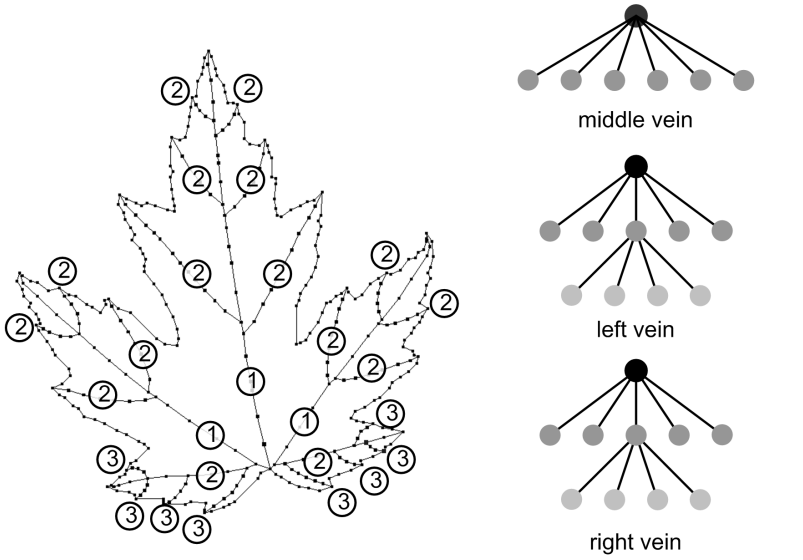


Figure 6.3: Venation modeling of a maple leaf

The overall data structure of the venation system for a dicot leaf used in this thesis is a venation tree. It will be referred to as the venation model in § 6.2.2. The term 'node' is used with two different meanings in this description. A vein is a sequence of position **nodes** which are point objects in 3D. On the other hand, the venation tree is a general tree structure whose **nodes** are vein objects. In the general tree, a parent node can have more than two child nodes unlike a binary tree where a parent node can have only two child nodes. Here, a node in the tree structure corresponds to a vein and the root node of the tree stands for a primary vein of a leaf if there is one primary vein while there can be several primary veins. The vein class has a set of position nodes, a set of edges, and a set of child veins which has the information of the hierarchy existing in the venation system. The data structures used in this thesis are shown as follows.

```
/* The class header files are written in C++,
   where STL (C++ Standard Library) is used. */

class Node {
    float x, y, z; //position data of a node
};

class Edge {
    Node *vPtr[2]; //connectivity information of two nodes
};

class Vein {
    vector<Node *> nodeVector; //a container of nodes
    vector<Edge *> edgeVector; //a container of edges
    vector<Vein *> veinVector; //a container of child veins
};
```

For the maple leaf, high order veins over third order have almost same diameter as the thickness of the leaf blade, so those veins do not contribute to the structural stability more than the leaf blade tissue part. Therefore, it is appropriate to consider veins usually up to third order when modeling the leaf venation as a hierarchical tree system. For the silver maple leaf shown in Figure 6.1, there are three primary veins. A root node corresponds to a primary vein and child nodes of the root become secondary veins starting from the primary vein, and so forth. The venation model corresponding to the maple leaf shown in Figure 6.1 (a) is shown in Figure 6.3, where the hierarchical structure of the venation is also given.

6.2 Triangulation of a Leaf with Vein Constraints

A leaf domain is defined as a region enclosed by a piecewise linear curve which is extracted from the leaf image by the marching squares method. Since some dicot leaves have very complicated shapes, the boundaries are topologically the same but their boundaries are very complex. Therefore, simple geometrical shapes like ellipse or rhombus cannot represent the complex leaf domains, so triangulation is needed to represent such complicated domains with a set of triangles. It is because the triangle is the simplest 2D geometry and the most basic shape used in many applications like finite element methods (FEM) and computer graphics.

6.2.1 Triangulating the Leaf Domain

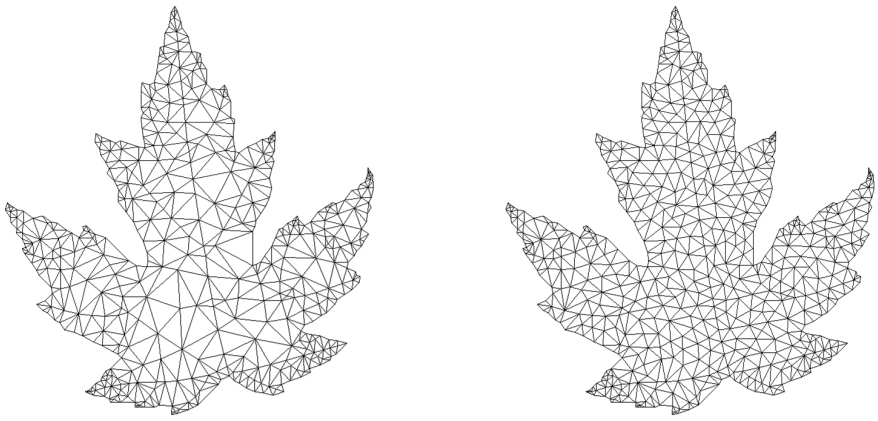
There are two kinds of triangulation. The one kind of triangulation gives structured meshes[1] and the other kind does unstructured meshes[2]. Structured meshes are most commonly used in computational fluid dynamics (CFD) because the target objects of most applications in the CFD field have simple shapes and the structured meshes are suitable to such simple shapes. On the other hand, unstructured meshes demand on storage space and spend more processing time compared to the structured meshes, but the unstructured meshes are very flexible, so can be applied to very complicated domains like dicot leaves.

Unstructured mesh generation is generally based on the Delaunay triangulation [30], where a triangles is created by joining nearest neighboring nodes as its edges. The Delaunay triangulation of a vertex set always maximizes the minimum angle among all possible triangulations of that vertex set. A mesh with large angles or small angles makes a very thin triangle which results in bad shaping of the leaf mesh and poor rendering quality, so should be avoided.

The input for mesh generation is a Planar Straight Line Graph (PSLG). The PSLG includes both the polygonalized boundary and the veins made of piecewise linear lines. A PSLG is a set of vertices and segments that satisfies two constraints [30]. The first one is that each segment in PSLG has always the two end points and the second is that the segments in PSLG connect only at their end points. Conforming constrained Delaunay triangulation (CCDT) of a PSLG is similar to the Delaunay triangulation, but every input segment appears as an edge of the

[1] In structured meshes, all internal nodes have equal number of adjacent elements.

[2] Unlike structured meshes, every internal node does not require equal number of adjacent nodes.



(a) max area = 1000 square pixels (b) max area = 400 square pixels

Figure 6.4: Triangulation of a maple leaf with different size constraints

triangulation. Some of the edges of the CCDT are constrained Delaunay. CCDT's are not necessarily Delaunay triangulations [30]. In order to make CCDT become Delaunay triangulation, additional vertices should be inserted into the mesh until all constraints on minimum angle and maximum area are met. This triangulation is called a conforming Delaunay triangulation.

The triangulation tool used in this thesis is Triangle [29], which is a well known unstructured triangular mesh generation program and has many control parameters. Although there are many options in the independently operating Triangle, the embedded form of Triangle can have only two control parameters because the GUI of the interactive modeler provides those two input boxes. The two control parameters are minimum angle and maximum area of triangles. The details of the GUI is given in Appendix C. Triangle also gives the Voronoi diagram data for the meshes, which is used for interpolating the leaf mesh nodes. The detail of the interpolation is discussed in § 7.2.

Figure 6.4 shows two results obtained from applying the CDT to the maple leaf domain with varying size constraints. The leaf image is given in a square area of 800×800 p^2 where 'p' stands for pixel. The two size constraints limiting the maximum size of the triangles are 1,000 p^2 and 400 p^2, respectively. The minimum angle constraint is set as 20°. Figure 6.4 shows that the constraint of 1,000 p^2 causes some size irregularities in the triangulation, but the triangulation with 400 p^2 gives uniform and small triangles. The uniform and small triangles

(a) max area = 1000 square pixels (b) max area = 400 square pixels

Figure 6.5: Voronoi diagrams of a maple leaf with different size constraints

are desirable because rendering of the mesh shows smooth surface and minimize shadow artifacts. However, as the maximum area constraint becomes smaller, the calculation time and the number of triangles increase.

Table 6.1 shows this trend in a quantitative way. Since the leaf domain is so complicate, there is no simple relationship between the triangle numbers and the constraint size. For example, from Table 6.1, when the area constraint changes in

Table 6.1: Effects of the area constraint on the triangulation.

Maximum size (square-pixel)	Maple leaf (minimum angle = 20°)		
	nodes	edges	triangles
300	726	1,920	1,195
400	626	1,621	996
500	575	1,469	895
600	549	1,391	843
700	528	1,329	802
800	520	1,306	787
900	511	1,279	769
1000	508	1,270	763

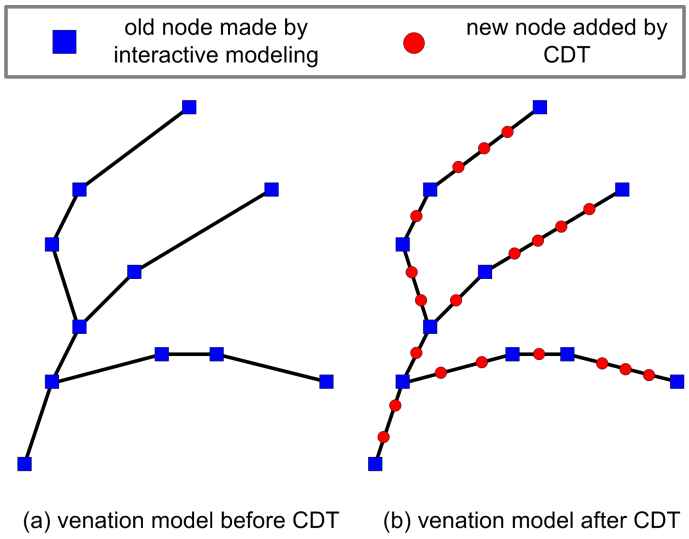


Figure 6.6: Change in the venation model by conforming DT

$600 \sim 1000$ p^2, there is no significant changes in the triangle numbers because the minimum angle constraint is more important in the actual triangle size. While triangulating the leaf, the Voronoi diagram is obtained, which is used in transforming the shape. The Voronoi diagrams are shown in Figure 6.5.

6.2.2 Updating the Venation Model

The triangulation process (CDT) of the leaf domain, which is defined by the boundary and constrained by the venation, adds new mesh nodes to the constrained edges that form the boundary and the veins. These new nodes must also be added to the venation model. Updating the venation model after the triangulation is based on the property that the new nodes for the venation model are always created on the original straight lines of the vein, which is shown in Figure 6.6.

The update scans the list of nodes in the original venation model. For a node, v in the original venation, let S be a segment of the venation starting at v. The sequence of mesh nodes on S can be identified iteratively by finding the mesh neighboring node of v in the direction of S. For this, it is necessary to be able to access the neighboring nodes of each mesh node. The update of the venation model can be done by the following algorithm that automatically updates the venation model after the triangulating the leaf domain.

Preprocess for updating

Make a neighboring node pointer array for every node in the leaf mesh. The node pointer arrays are stored in a container which is used for querying about neighboring nodes of a node. Let V_i^o, where $i = 0, \ldots, n_V - 1$, be vein i in the original venation model and v_{ij}, where $j = 0, \ldots, n_{V_i} - 1$, be node j in V_i^o. Let V_i^u stand for the updated $i - th$ vein of the venation model.

Main routine for updating

For $i = 0, \ldots, n_V - 1$, do

For $j = 0, \ldots, n_{V_i^o} - 1$, do

$v \leftarrow v_{ij}$.
put v_{ij} into V_i^u.
While ($j \neq n_{V_i^o} - 1$ && $v \neq v_{i(j+1)}$), do
find a neighboring node v^n of v such that $\overline{vv^n} \parallel \overline{vv_{i(j+1)}}$.
put v^n into V_i^u.
$v \leftarrow v^n$.
endWhile

endFor

endFor

Chapter 7

Generation of a 3D Leaf Shape

A 3D leaf shape is generated in the form of a 3D parametric surface, which is obtained by transforming the 2D leaf veins into 3D veins and interpolating the nodal points of the 2D leaf mesh into 3D, using the 3D veins. This idea is to use the leaf venation as an embedded skeleton in the leaf and the leaf blade tissue as a 'skin' spanned by the skeleton. Since the skin is assumed very flexible and the skeleton is assumed very stiff, once the skeleton has a specific shape, the skin is stretched or shrunk to conform the skeleton shape. As mentioned earlier, for a general dicot leaf, its low order veins are responsible for the mechanical stability of the leaf, so such shape modeling technique can give a botanically plausible 3D shape of the leaf.

The transformation of 2D leaf veins into 3D veins is done interactively by using the modeling tool developed in this thesis (see Appendix C). Then, the leaf blade mesh nodes are transformed to conform to the transformed vein curves, which is automatically done by interpolation using the FVM, which will be discussed in Appendix B. The processes of transforming veins and interpolating leaf blade nodes are discussed in the following two sections.

7.1 Transformation of Leaf Venation

The basic assumptions for transforming 2D veins into 3D veins used in this thesis are two; the first one is that a vein is not stretchable, that is, the vein length is fixed and the second is that the vein cannot be twisted. The first assumption is made not to change the overall size of the leaf while transforming its veins. Although this assumption limits interesting applications in the shape modeling like the growth of

the leaf or the morphing of the leaf, such an assumption is helpful in preventing a user's modeling mistake. Of course, this assumption can be eliminate later by allowing for a user to select whether the vein length is stretchable or not.

The second assumption is introduced to make the vein transformation algorithm simple without any significant loss of accuracy. Based on this assumption, an edge in a leaf vein cannot spin about its axis. The basic user interactive operation for the leaf shape modeling is the rotation of a vein edge about its node closest to the vein's root node. This node is referred to as the hinge node of the edge. Since the leaf venation is a hierarchical skeleton system, the sub-trees appending this edge rotate with it as if they are a rigid body, which is shown in Figure 7.1.

The interactive modeler uses three windows that provide 3 orthogonal views of the veins, a front view (X-Y), a side view (Y-Z) and a bottom view (Z-X). In the shape modeling, the subtree rotations are accomplished by 2D rotations in these windows. In fact, since a vein does not elongate or shorten, a user's translational drag of the target node[1] by using the mouse is expressed by a rotational move of the edge (see Figure 7.1).

7.1.1 The Method of Controlling Individual Nodes

The leaf shape modeling scheme developed in this thesis uses two different ways for transforming the 2D leaf veins into the 3D veins, which are complement to each other. The first one is to control every node except a root node in a vein, on the other hand, the second one is to use a temporary curve for transforming veins.

Here, a root node of a primary vein is the node connecting to a petiole and a root node of a higher order vein is the node connecting to the parent vein (see § 6.1.2). If a node in a vein is selected to move, then the node can be rotated about its hinge node close to the root node of the vein, which is shown in Figure 7.1. In the first method, when a node in a vein is rotated about its hinge node, all descendant veins branching out from the vein after the hinge node are rotated together. This motion can be easily obtained because the leaf venation model is hierarchical. The following pseudo recursive routine shows how such hierarchical motions are made.

[1]This node is one of two nodes in the controllable edge, which is a distant node from the root node of the corresponding vein. The node close to the root node is a hinge node. When a user clicks a node to move, the node becomes the target node automatically: the corresponding hinge node is also automatically determined according to the above rule. See Figure 7.1.

```
/* Vein and Node class header files are shown
   in the previous chapter. */
rotateEdge (float ang, Vein V, Node hingeNode, Node targetNode)
{
    for a node v in V starting from targetNode, do
    {
        rotate v around hingeNode by the angle ang;

        for a child vein VC of V branching out from v, do
            call rotateEdge (ang, VC, hingeNode, v);
    }
}
```

Since only rotational motions of the node about its hinge node are allowed, the segment lengths of the edges do not change. Therefore, the overall size characteristics of the leaf are preserved under such transformation. This scheme is analogous to moving a bone in a human skeleton, that is, rotating a upper arm causes to

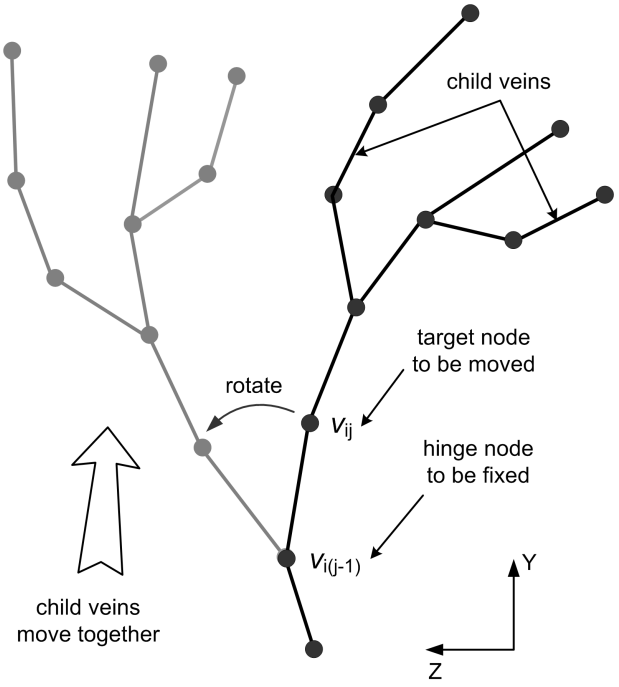


Figure 7.1: Basic concept of moving a node

rotate its appendages including its lower arm, a hand connecting to the lower arm, and so forth.

The individual control of many nodes makes the detail local shaping of the veins possible and this scheme is simple to implement. However, there may be too many nodes in a vein to control individually, so it may be tedious to rotate every nodes in order to make a specific leaf shape. In order to alleviate such difficulty, the second method of transforming vein curves is proposed in the following subsection.

7.1.2 The Method of Using Spline Curves for Veins

The main difficulty of interactively modeling the 3D position of every node in veins is the fact that there are too many nodes to be controlled. Therefore, if a small number of nodes are selected and the vein curve can be changed close to a spline curve specified by a user using only those nodes, then transforming the veins becomes much easier than the previous method of controlling individual nodes. The main idea of this method of using spline curves is basically to reduce the degree of freedom of each veins by replacing it with a spline curve, which is constructed by a small number of nodes in the vein. The basic idea of this method is illustrated in Figure 7.2.

The method using spline curves starts from calculating the length of the each line segment of the veins because the length is used as a parameter for calculating new nodal coordinates. The vein segment lengths are calculated and stored just after updating the venation model. Therefore this procedure can be regarded as a preprocess. First, the control nodes for one vein are selected interactively (Figure 7.2 (a)). The control nodes should be chosen based on which part of the vein is transformed. The root and the tip nodes of the vein must be selected. Second, construct a temporary vein with only those nodes (Figure 7.2 (b)). Third, modify the temporary vein as if the new vein represents the original vein (Figure 7.2 (c)). Unlike the previous method, any descendant veins do not move at this time. Fourth, now a spline curve which interpolates the nodes of the temporary vein is constructed (Figure 7.2 (d)). Fifth, the original nodes are mapped onto the spline curve by using the parameters calculated at the preprocessing stage (Figure 7.2 (e)). Sixth, recover the original vein by automating the method of controlling individual nodes, which is described in the previous subsection (Figure 7.2 (f)). At this stage, all nodes except the root node in the vein are rotated so that the nodes on the transformed curve may be close to the nodes mapped on the spline. At this time, all descendant veins are moved by the recursive routine mentioned.

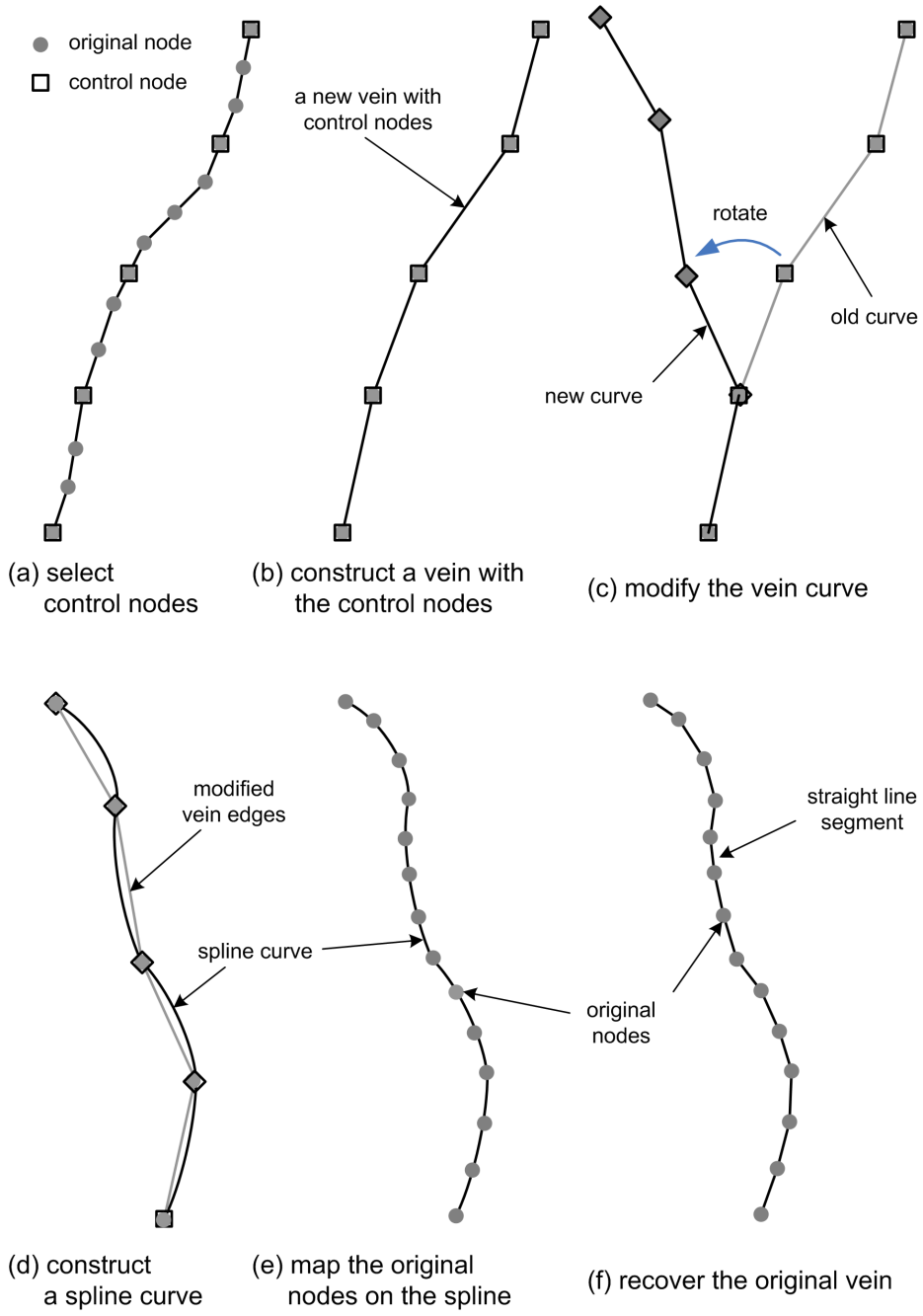


Figure 7.2: Schematics of using splines for transforming veins

The two methods for transforming vein curves are not entirely different methods. In fact, the second method of using a temporary spline curve uses the first method of controlling individual nodes as a subroutine at its final stage of the vein modification. The second method has an easy interface because there can be small number of nodes and can make a smooth vein curve more easily than the first method because the second method uses spline curves as an intermediate step. These two methods can be used together in a mixed mode, that is, with the second method, the overall vein curves can be easily modified and with the first method, the local details of the vein curves can be changed. Therefore, the two methods complement each other.

7.2 Interpolation of a Leaf Mesh

The nodes in the leaf blade meshes which are obtained by triangulating the leaf domain with the vein constraints in 2D, are interpolated to give a 3D shape by using the transformed veins in 3D. Basically, the idea of the interpolation is based on the assumption that the veins of the leaf are uniformly spread over the leaf blade, so any node of the mesh is well surrounded by the constraining veins of which shapes are already known and fixed. As mentioned earlier, for the leaves where low order veins play a great role in providing the mechanical stability, the low order veins generally cover the whole leaf domain. For such leaves, the interpolation of the nodes using the vein constraints can give photo-realistic results.

The leaf blade mesh nodes are interpolated to conform to the veins transformed in 3D using Laplace interpolation [32]. Let $\mathbf{f}(r,s) = (x(r,s), y(r,s), z(r,s))$ be the function representing the 3D leaf mesh shape, where r and s are parameters for a position in the 2D leaf domain Ω. Let $\partial\Omega$ stand for the outline of the leaf of Chapter 5 and let Γ be the updated 2D leaf venation which is interactively modeled and automatically updated during the triangulation as discussed in Chapter 6. There are two kinds of boundary conditions applied to the domain Ω, the one is a Neumann boundary condition defined at $\partial\Omega$ because $\partial\Omega$ is considered as free and the other is a Dirichlet boundary condition[2] defined at Γ, which are illustrated in Figure 7.3. Let $\mathbf{g}(r,s)$ be the forced boundary condition at Γ which is given by the 3D vein curves modeled interactively as discussed in § 7.1. The interpolation problem is mathematically defined as follows.

[2]Although the leaf venation is not the boundary of the domain, the position coordinates of the leaf venation are treated as if these are Dirichlet boundary condition.

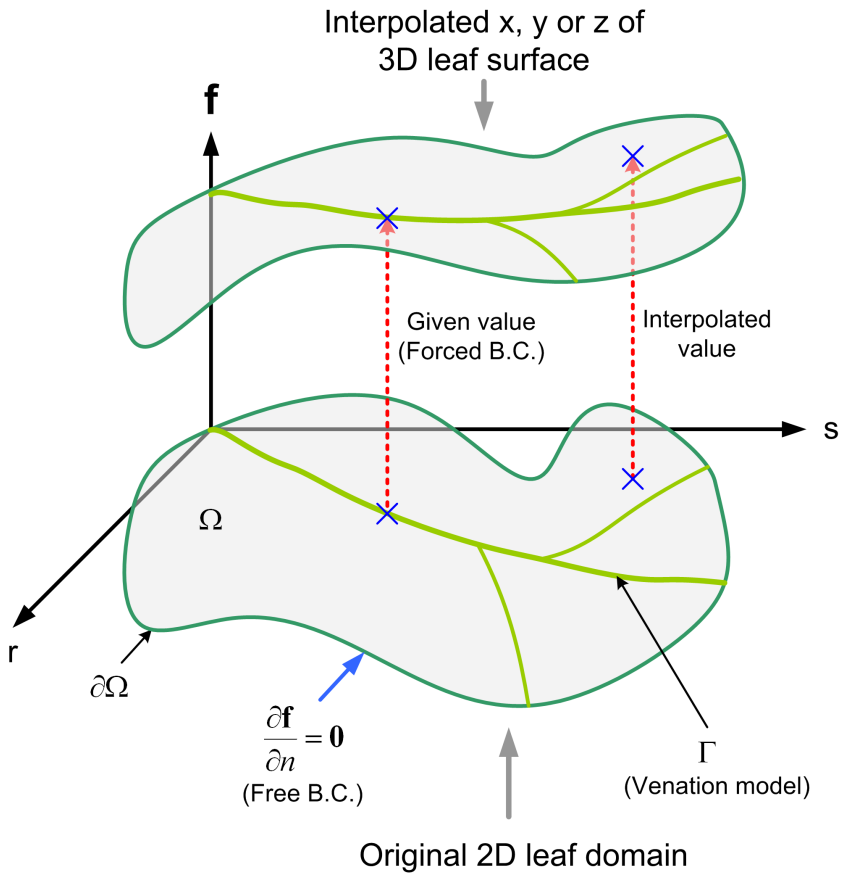


Figure 7.3: Interpolation of 3D leaf surface coordinates

$$\begin{aligned} \nabla^2\mathbf{f} = \frac{\partial^2\mathbf{f}}{\partial r^2} + \frac{\partial^2\mathbf{f}}{\partial s^2} &= 0 \text{ for } (r,s) \in \Omega \\ \mathbf{f}(r,s) &= \mathbf{g}(r,s) \text{ for } (r,s) \in \Gamma \\ \frac{\partial\mathbf{f}}{\partial n} &= 0 \text{ for } (r,s) \in \partial\Omega \end{aligned} \tag{7.1}$$

Functions that satisfy Laplace's equation, $\nabla^2 F = 0$, are called harmonic functions. The Laplace equation is interpreted as a partial differential equation governing an elastic membrane problem [25]. The function $\mathbf{f}$ specified by Equation 7.1 can be viewed as 3 harmonic functions defined in Ω that interpolates $\mathbf{g}(r,s)$ on Γ. The application of this mathematical concept to the leaves with complicated boundaries

starts from discretizing the domain Ω into simple geometrical shapes like triangles. The parameters r and s represents the coordinates of the nodes in the discretized 2D domain Ω. The coordinates of the transformed veins in 3D space are treated as the discretized values of $\mathbf{g}$ at Γ. Now the Laplace interpolation problem of the leaf mesh is solved by a finite volume method (FVM) defined for the discretized domain. The details of FVM in this application is described in Appendix B.

The procedure developed in this chapter is summarized as follows. First, the veins of the venation model are transformed using the methods developed in Section 7.1. Then, the leaf mesh nodes are interpolated conforming the transformed veins. Finally, texture mapping gives a realistic image to the interpolated mesh model. The rendered images shown in Figure 7.4 are obtained using OpenGL [35]. OpenGL rendering can be done in real time, so this quick feedback helps a user to model a leaf interactively. If the user is satisfied with the the resultant leaf shape, then the mesh model can be exported to a off-line renderer like POV-Ray [24]. Otherwise, the user can repeat the above procedure. The application examples for a maple leaf and a chestnut leaf are given in Figure 7.4 and 7.5. Figure 7.4 (a) shows that small sub-lobes beside the three main lobes are bent a little upward. On the other hand, Figure 7.4 (b) shows the same maple leaf but with slightly bent tips. Figure 7.5 shows two chestnut leaves with a same venation model. In Figure 7.5 (a), the chestnut leaf blade is bent about its midrib slightly, but in Figure 7.5 (b), the blade is bent more significantly.

In reality, however, there are some leaves where strong leaf blades are responsible for their structural stability. These leaves have vague venation patterns or some regions of the leaf blade are not properly covered by the veins. For these leaves, the interpolation using the venation model based on the apparent veins may give poor results. For example, there may be unnatural wrinkles in the resultant 3D leaf shape. In order to prevent such undesirable interpolation, several methods can be considered. A more accurate approach is to include the stiffness of the leaf blade by using the finite element method (FEM). Although this method may solve such interpolation problem, it creates many other difficult problems. One of those is to measure the material properties of the leaf blade, for example, Young's modulus E[3]. Therefore, in this thesis, artificial veins which have no physical dimension in their thickness, may be introduced to improve the interpolation quality in order to avoid such complicate procedure and those fictitious veins do not appear in their rendering.

[3]The ratio of the stress to the strain in a material [9].

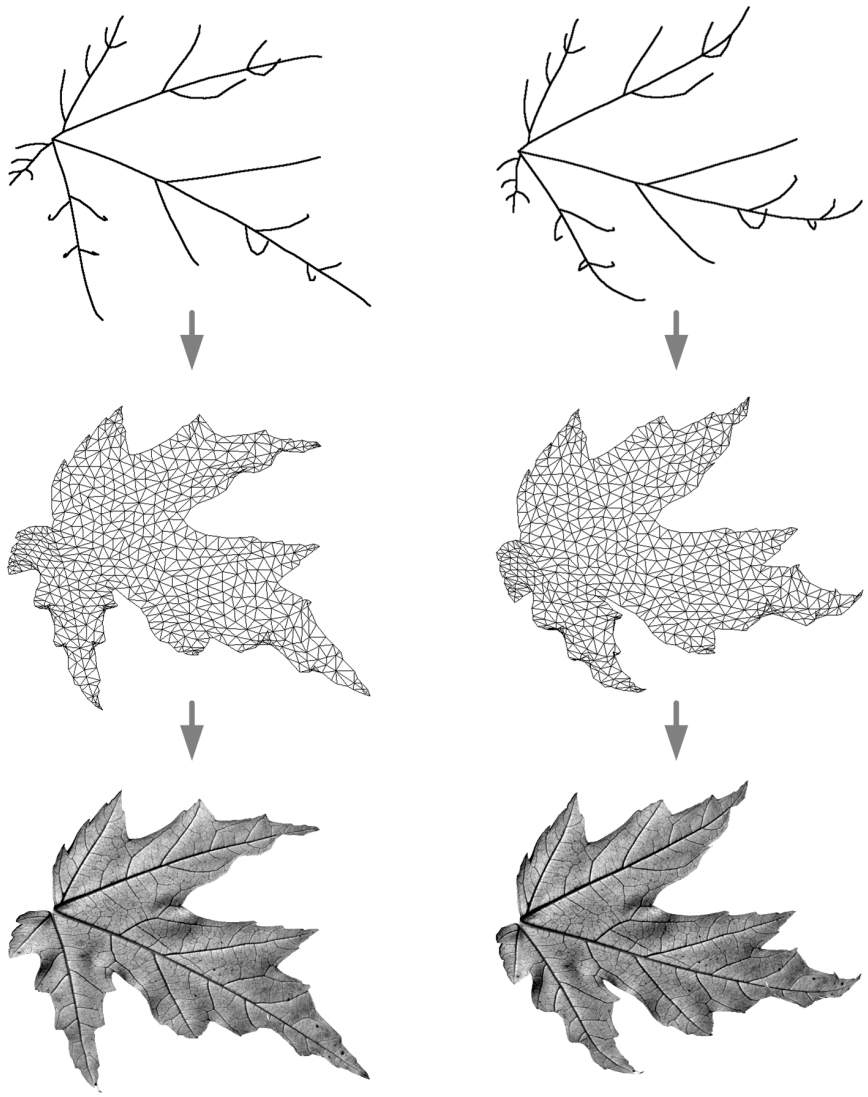


Figure 7.4: Transforming venation and interpolating mesh of a maple leaf

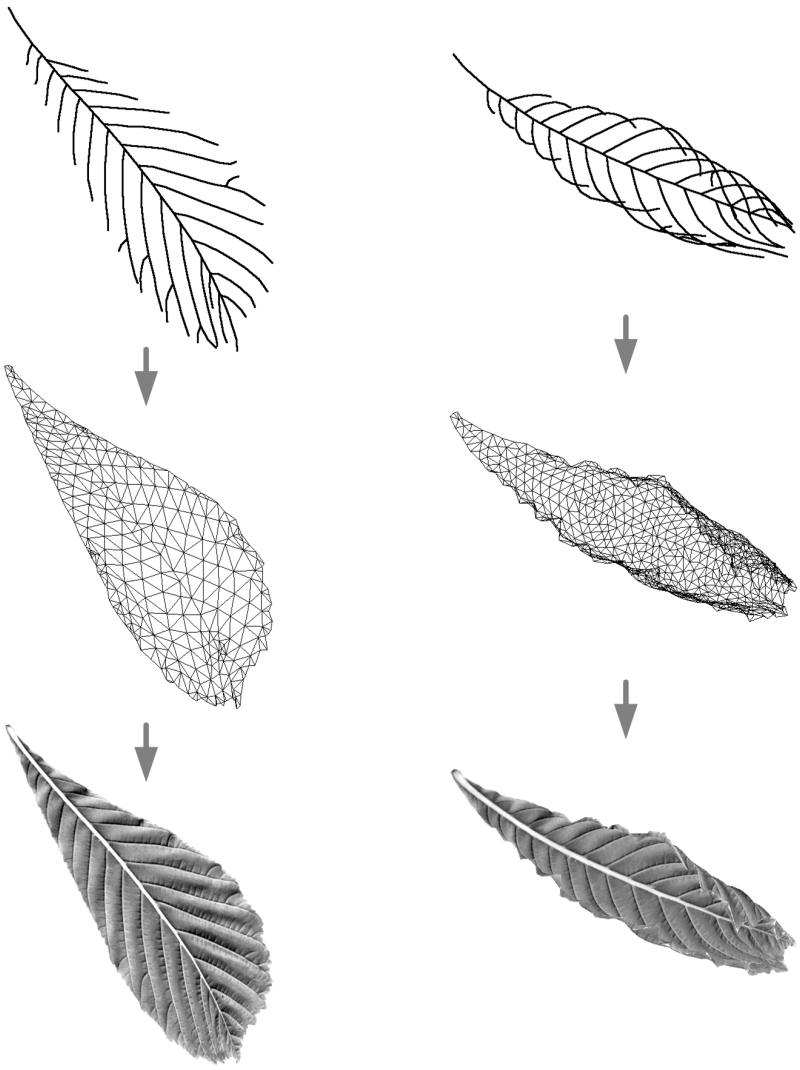


Figure 7.5: Transforming venation and interpolating mesh of a chestnut leaf

Chapter 8

Interactive Modeling of a 3D Leaf Shape

8.1 Interactive Modeler of a Leaf

An interactive modeler has been developed to help a user to model plant leaf shapes as realistically as possible. The operating system where the modeler is developed is Microsoft Windows XP and the language used for the modeler is Microsoft Visual C++. Qt[1] [4] library for Visual C++ is used for constructing the GUI and the OpenGL [35] API is used for rendering the leaf mesh model in real time. As mentioned in § 6.2, the mesh generating routine uses Triangle [29]. According to the guideline enclosed in Triangle distribution package for Unix and Linux operating systems, the source code is slightly modified so that the mesh generation tool is embedded in the interactive modeler. The modeler has the following capabilities and most of them are explained in the previous chapters.

- Capturing a leaf profile and simplifying the profile (Chapter 5).
- Creating a venation model and generating a mesh model of the leaf (Chapter 6).
- Transforming the venation model and interpolating the mesh (Chapter 7)
- Rendering the mesh model with the leaf image as an instant feedback, where the rendering is done through OpenGL.

[1] An open source graphics library for the Microsoft Windows and the X Window System on GNU/Linux, which is used by programmers to create buttons, menus, and other graphical objects.

- Save and load the leaf venation and mesh model for later modification.
- Export the mesh model to an off-line renderer.

Like most organic materials, plant leaves, especially dicot leaves, are generally very complicate in their shapes. Manual generation of such complex models for the leaves is not impossible but very tedious, so a modeling tool to facilitate the modeling processes described in the previous chapters is indispensable. The modeler makes it easy to generate many realistic variants from a leaf shape model by transforming the venation model differently, which is the greatest benefit in using such a modeler.

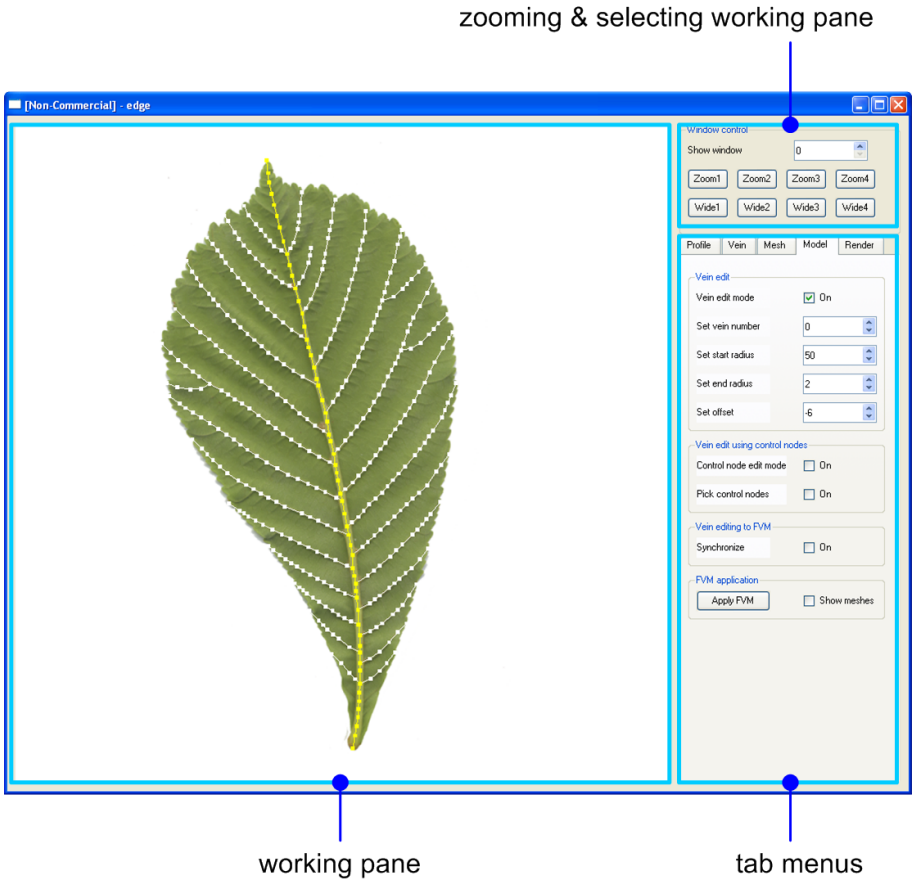


Figure 8.1: Structure of the interactive modeler

The graphic user interface (GUI) of the modeler is shown in Figure 8.1. The GUI is partitioned into 3 parts to use the limited space efficiently. The first part is

a square window shown in the left of the GUI, which contains 4 different working panes. The window can show one working pane at a time or four panes by splitting the window into four quadrants, which is shown in Figure 8.2. The second part is located in the top-right of the GUI. This part has a spin box for selecting which working pane to use and buttons for zooming a specific region in the working pane, where the region is specified by the user. The bottom-right part has five tab menus, which are providing the functions mentioned above. The details of the tab menus are described in Appendix C.

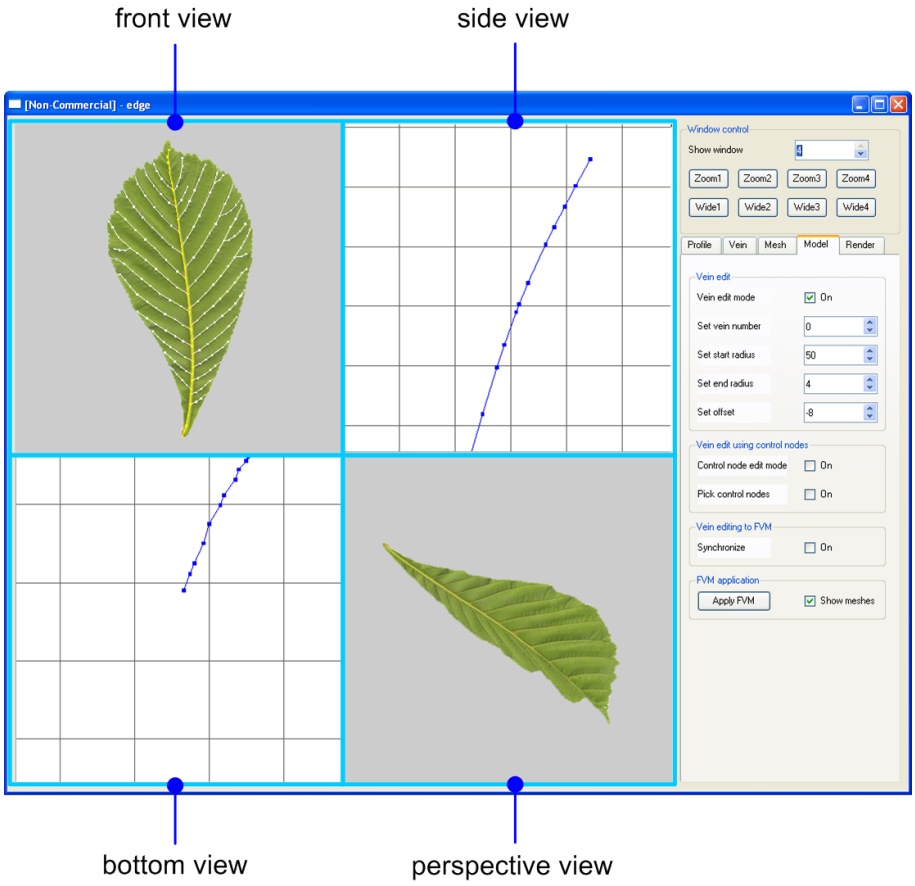


Figure 8.2: Working panes in the interactive modeler

The working pane shown in Figure 8.1 has a fixed size of 800×800 p^2, where 'p' stands for pixel. The square size for the marching squares and the tolerance for the simplification of the profile are given in p while the maximum size criteria for meshing is given in p^2. The purpose of using such pixel unit is to normalize

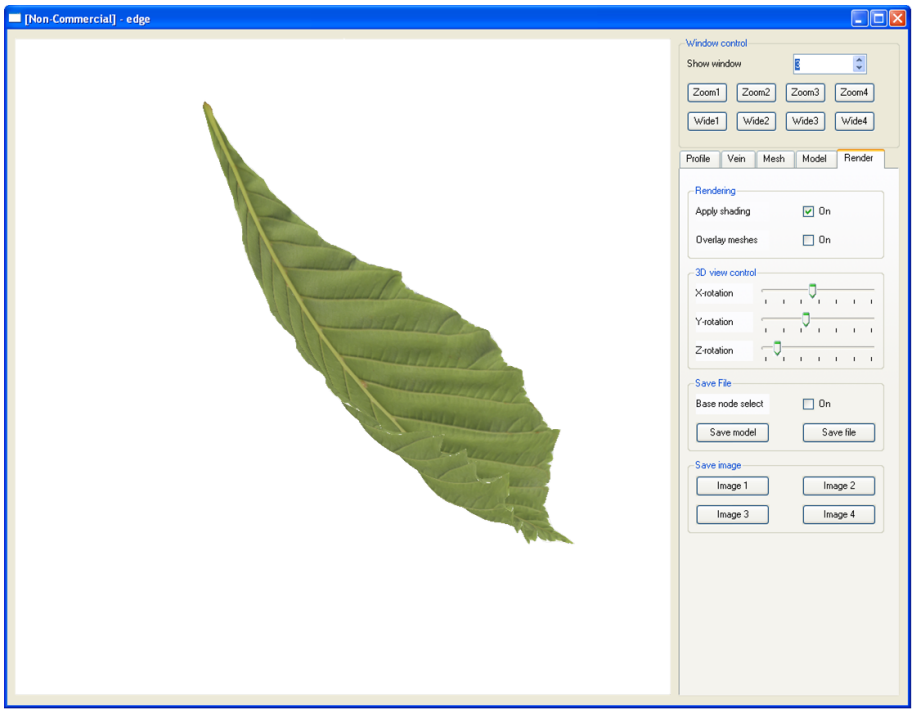


Figure 8.3: Perspective view in the interactive modeler

the physical characteristics of the leaf model because the leaf image size is fixed to 800×800. The venation model for the chestnut leaf shown in Figure 8.1 is generated in a hierarchical manner. First, a primary vein is created by clicking the mouse at positions where the nodes of the primary veins are located. Primary veins have their root nodes on the leaf boundary where the petiole is connected. Then, child veins branching out from their parent veins are created. Therefore, there should be a node where a parent vein and a child vein are supposed to connect. Using the leaf image as a background layer image, a user can make a venation model for the leaf with ease.

As mentioned in Chapter 2, Hammel et al. [14] proposed the automatic generation of leaf venation by using L-system and leaf boundary by using implicit functions along the venation. Their method can be fully automatic in generating the leaf venation and margin, but the resulting venation and margin are not satisfactory because the appearance of the leaf is far from being realistic. Mündermann et al. [21] generated the leaf venation mathematically by extracting the medial axis from the scanned leaf image. However, the leaf venation by their method is not

acceptable because the medial axis is much different from the real venation of the leaf. Although their methods can be improved by adding probabilistic components, at present the interactive modeling of the leaf venation used in this thesis seems to give most realistic venation.

The working pane shown in Figure 8.1 can be split into four sub-panes of which size is 400×400. These four panes shown in Figure 8.2 are used specially for transforming veins. This kind of GUI is a popular form of GUI found in many 3D modelers like Maya[2]. The top-left pane is the default pane which appears at starting the modeler and shows a front view (X-Y). This pane is the main working place for extracting a leaf profile, creating a leaf venation model, etc. The top-right (a side view, Y-Z) and the bottom-left (a bottom view, Z-X) panes are auxiliary panes for transforming veins where only one vein is shown in order to avoid any confusion with other veins.

Unlike the above mentioned three working panes, the last pane, which is the bottom-right pane in the working window, is not working place. That is, a user cannot modify anything about the leaf model in this pane. It just shows a perspective view of the venation model, the mesh model or the rendering of the mesh model through OpenGL. The purpose of this pane is to give the user an instant feedback of the model the user is currently making. The perspective view can be given with a various viewing angle because the model can be rotate around x, y or z axis of the model by using the slide bars in the last tab menu, which is shown in Figure 8.3. If the user is satisfied with the model he has made, the venation model or the mesh model can be saved for later uses and the mesh model of the leaf shape can be exported to an off-line renderer.

8.2 Rendering Leaves

The interactive modeler described in the previous section provides a real-time rendering for the leaf which is based on OpenGL functions like texture mapping and transformation. The rendering through OpenGL is very fast, so such rendering is helpful for a user to get a instant feedback to his/her shape modeling. However, the rendering based on OpenGL is not very satisfactory in its quality because it is not easy to provide proper light conditions, material properties and so forth. To circumvent this difficulty, the modeler has a capability to export the leaf shape model to other off-line renderers like POV-Ray [24]. Such off-line renderers in general use ray

[2]Maya from Alias is an integrated graphics software providing 3D modeling, animation, effects, and rendering solution.

tracing techniques to give high quality rendering, that is, many sophisticated effects like anti-aliasing and soft shadow are available without complicated programming. Although most ray tracers are commercial, POV-Ray is a copyrighted freeware ray tracer to provide a lot of excellent rendering functions. Therefore, the modeler has a function to export the mesh model to POV-Ray for high quality rendering while it has a simple rendering function based on OpenGL to give an instant feedback to a user. In this section, some ray traced rendering examples are given to demonstrate the validity of the proposed method.

8.2.1 Common Horse Chestnut Leaves

Chestnut leaves are compound leaves which have blades that are fully divided into leaflets. In chestnut leaves, leaflets arise from a single point at the top of the leaf stalk and the leaflets are almost stalkless. It is interesting to know that compound leaves are usually shed as a single unit. According to [8], the botanical name of the common horse chestnut is '*Aesculus hippocastanum*' and the common horse chestnut tree is vigorous, spreading, rounded trees with 5 to 7 palmate, mid-green leaves consisting of obovate leaflets, 30 cm or long. Some of leaves turn deep yellow or red in autumn.

A chestnut leaflet, which is shown in Figure 8.4, is modeled into several shapes simulating changes which are usually observed in common chestnut trees at the change of seasons. The corresponding ray-traced images are shown in Figure 8.5. In case of the chestnut leaflet, there is a primary vein, a lot of secondary veins and several tertiary veins. The primary vein is very thick and strong, so the curvature of the vein is normally small, however, the secondary veins may have high curvatures because of their low stiffness. If the leaflet is extremely dried, then the primary vein can have a rather high curvature and the secondary veins are severely deformed, so almost wrapped like a roll cake. The compound leaves consisting of the various shaped leaflets are shown in Figure 8.6. The rendering images look realistic, and demonstrate that the proposed method and the interactive modeler are suitable for modeling shapes of plant leaves in a botanically faithful manner.

The veins existing in the chestnut leaflet are modeled and rendered using generalized cylinders. Although the cross-section of the generalized cylinder is circular whereas the cross-sections of the actual veins are not so regularly shaped, the introduction of the generalized cylinder for the leaf vein dramatically improves the realistic appearance of the rendering images. Figure 8.7 shows how realistic the rendering images of the chestnut leaflet looks when the shadow lines appear because of the veins. If the cross-section of the generalized cylinder has more natural

shapes, then the rendering image can be enhanced and more realistic.

8.2.2 Maple Leaves

There are many kinds of maple trees. Among them, silver and sugar maple trees are very common in Ontario, Canada. According to [8], the botanical name of the silver maple is '*Acer saccharinum*' and the silver maple is spreading, fast-growing, deciduous tree, often with pendent branches. The silver maple leaves are sharply-toothed, shallowly to deeply 5 lobed leaves, 10 to 20 cm long. The leaves are lightly green above, silvery white beneath, and turn yellow to orange or red in autumn.

Several maple leaves are modeled starting from a silver maple, which is shown in Figure 8.8. Those leaf models are simulating the aging phenomena of the maple leaves. When the leaf is young, the shape is almost flat, but as the season turns from spring to autumn, the the leaf blade becomes dried and distorted while the leaf color is turning orange or red. The shape deformation according to the season changes is not deterministic but tends to follow the deformation of the venation as mentioned in Chapter 3. The rendering images of several different shapes of the silver maple leaf are shown in Figure 8.9.

The maple leaves shown in Figure 8.10 make a realistic scene against a background of light gray and yellowish wooden fence. In total, 6 maple leaves are hanging on a twig of a silver maple tree. The sun light casts the interesting shadow of the leaves in the bottom left of the scene. The petioles and veins are modeled with generalized cylinders. Although the scene itself is not entirely biologically predictable since there are both greenish and reddish leaves in a twig, it has a overall natural appearance. This scene shows that the proposed method is suitable for modeling realistic plant leaves.

Figure 8.4: Sunny and shadow sides of a horse chestnut leaf

Figure 8.5: Chestnut leaves simulating aging phenomena (getting older in clockwise)

Figure 8.6: Compound chestnut leaves consisting of different aged leaflets

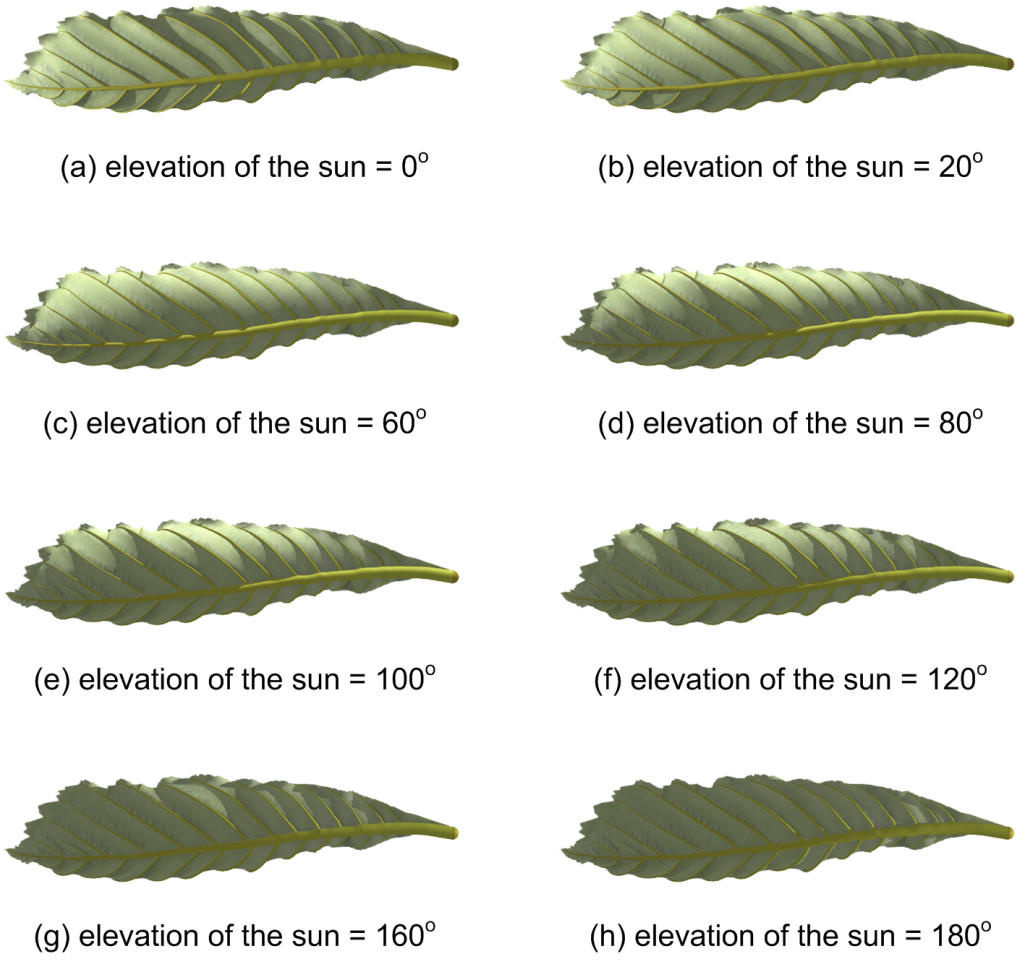


Figure 8.7: Moving shadow lines while the sun is moving

Figure 8.8: Sunny and shadow sides of a silver maple leaf

Figure 8.9: Maple leaves simulating aging phenomena (getting older in clockwise)

Figure 8.10: A twig with different aged silver maple leaves

Chapter 9

Conclusions and Future Work

9.1 Conclusions

In this thesis, a novel shape modeling method of plant leaves has been presented. The main idea of the method is based on the premise that the leaf venation is responsible for the leaf structural stability. In this method, the leaf venation is modeled as a hierarchical skeleton. Two methods for deforming the skeleton, which complement each other, have been developed: one is to control individual joints of the skeleton, on the other hand, the other is to control the skeleton through an intermediate spline approximation. Then the leaf blade, which forms the realistic leaf shape, deforms conforming to the transformed skeleton.

An interactive modeler has been developed to help a user to model and deform the leaf venation quickly as well as easily. Interactive manipulation of the skeleton model and real-time rendering with OpenGL API are two important characteristics of the modeler. With the interactive modeler, several leaf clusters, which are maple and chestnut leaves, were modeled and rendered using POV-Ray. The ray traced images look very realistic, so the rendered images prove that the proposed method is very promising in shape modeling of plant leaves.

9.2 Future Work

Multi-resolution Leaf Shape Modeling

Although the number of nodes in a leaf mesh model used in this thesis is at most two thousands, in general there are several hundred of thousands such leaves in a mature tree. Therefore, it is almost impossible to use the leaf mesh model with several thousand nodes in the level of a full grown tree. Multi-resolution mesh models, where the mesh complexity varies depending on the level of detail, are commonly used to alleviate this problem [17]. If the mesh model proposed in this thesis is capable of changing its complexity according to the required level of detail, its applicability will be greatly increased.

Keyframe Animation Using the Leaf Shape Model

Even though the number of nodes in a practical leaf mesh model is at least a thousand, the number of joints in the skeleton can be reduced to a manageable level, that is, less than a twentieth of the total nodes. Keyframes can made with this simplified skeleton embedded in the mesh model and then the remaining frames will be interpolated using the simplified skeletons in the keyframes. Animation of realistic looking leaves will dramatically improve the reality of outdoor scenes with the leaves. Further research is necessary to determine how to make such keyframes and how to interpolate intermediate scenes.

Appendix A

Some Background of Botany

A.1 Plant Classification (excerpted from [8])

The plant kingdom, Plantae, is divided into progressively smaller groups according to shared botanical characteristics, usually represented as a family tree. The most basic division is between vascular and non-vascular plants [8]. Primitive, non-vascular plants, such as liverworts[1] and mosses[2], lack conductive tissue for the circulation of water and nutrients, and are thus confined to a moist environment. Vascular plants, on the other hand, which include both flowering and non-flowering plants, are very diverse and the adaptability of their root and shoot[3] systems have enabled them to thrive in many habitats.

Vascular plants that bear seed are divided into gymnosperms and angiosperms [8]. Gymnosperms produce seed that is only partly enclosed by tissues from the parent plant. Conifers normally bear seed on the scales of cones. Angiosperms, which are usually referred to as flowering plants, produce seed in an ovary - a protective chamber that forms part of the fruit when seed ripen and often aids in their dispersal. They are further defined as monocotyledons or dicotyledons, according to their seed leaves and other differences in their anatomy and growth patterns [8]. Monocots have a single seed leaf, leaves with veins that run parallel to their length, slender, non-woody stems (except in palms). Dicots have two seed leaves. Apart

[1] Any of numerous small green nonvascular plants of the class Hepaticopsida growing in wet places and resembling green seaweeds or leafy mosses.

[2] Any of various other unrelated plants having a similar appearance or manner of growth, such as the club moss, Irish moss, and Spanish moss.

[3] A bud, young leaf, or other new growth on a plant.

from two seed leaves, dicots have other distinctive anatomical features. Their adult leaves (those other than the seed leaves) have veins, usually arranged in a net-like pattern with a distinctive central vein or midrib, and the base of the leaf usually tapers to a point (see Figure 3.1).

A.2 Anatomy of a Dicot Leaf (excerpted from [10])

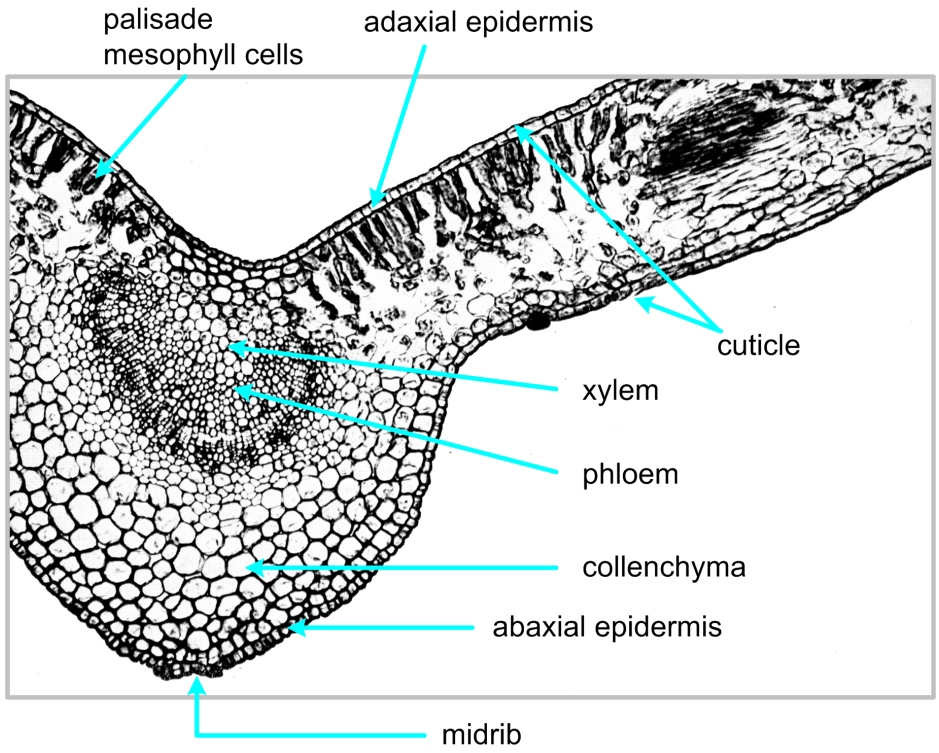


Figure A.1: Anatomy of a dicot leaf midvein (*Ligustrum*) (redrawn from [7])

An anatomy of a typical dicot leaf midrib[4] is illustrated in Figure A.1 [7]. The blade consists of an upper and lower epidermis and a spongy layer of tissue, called the mesophyll [10]. The epidermis is the leaf blade's skin. It is a thin, usually transparent, colorless layer of cells that covers both the upper and lower surfaces of the blade. The epidermis prevents the leaf from losing excessive amounts of water and protects it against injury. In most plants the epidermis is covered with

[4]The vein in the center of a leaf, the primary vein of the leaf.

cutin, a waxy substance secreted by the epidermal cells. The layer of cutin, called the cuticle, is responsible for the glossy appearance of some leaves. The midrib is enclosed with collenchyma cells, which have thick cellulose cell walls thickened at the corners and serve as supporting and strengthening the vein [10]. As shown in Figure A.1, the low order veins up the third order are thick and strengthened parts in the leaf blade, so they usually play an important role in the structural stability of the leaf. In general, veins are running through the middle of the mesophyll and branching out to all of its cells. The veins extend into the petiole and connect with other veins in the stem of the plant.

Dicot leaves usually present a differentiated mesophyll which is responsible for their bifacial appearance, that is, the abaxial surfaces look pale because they have higher reflectance. Monocots usually present an undifferentiate mesophyll and have a unifacial appearance [3].

A.3 Mechanical Aspects of Leaf Venation (excerpted from [28])

A major function of the veins is to help support the leaf blade [28]. Each type of plant has a characteristic pattern of veins forming lines and ridges in the blade. The veins of a leaf are made up of two specialized tissues, xylem and phloem, which are shown in Figure A.1. Xylem usually forms the upper half of the vein. It consists of tubular open-ended cells that are arranged end to end. The walls of the cells are thick and rigid. Xylem conducts water and dissolved minerals to the leaf blade from the rest of the plant. Phloem lies on the underside of the vein. It is made up of thin-walled tubular cells with tiny openings at their ends, somewhat like a sieve. These cells are also arranged end to end. The mechanical stabilization is based on the lignified xylem and sclerified elements which can be associated with the conducting bundle system of a leaf [22].

Most commonly, a leaf is a surface structure that maximizes the surface-to-volume ratio. It is then adequately described as a flat lightweight structure which is spread by mechanically stabilizing structures, usually the veins (as is the case in, for example, insect wings). The leaf lamina can thus be viewed as a stress-skin panel or a poly-laminated sandwich which is stiffened and stretched by interconnecting stringers represented by the leaf veins [23]. Inherent folding or curling can also contribute to the structural stiffness. Large and mechanically stiff midribs are desirable, because the greatest mechanical stress occurs along the longitudinal axis of a leaf. On the whole, the mechanical reinforcement of the leaf blade is primarily

due to lower order veins, that is, veins up to the third order are responsible for leaf structural stability.

Appendix B

Finite Volume Method

B.1 Voronoi Diagram (excerpted from [33])

Consider a set of distinct nodes $\mathbf{N} = \{n_0, n_1, \ldots, n_m\}$ in $\mathbb{R}^2$. The Voronoi diagram of the set $\mathbf{N}$ is a subdivision of the plane into regions V_i is associated with a node n_i, such that any point in V_i is closer to node n_i than to any other node $n_j \in \mathbf{N}$ $(j \neq i)$. The Voronoi diagram in essence partitions space into closest-point regions, which is shown in Figure B.1. The region V_i for a node n_i within a convex hull is a convex polygon in $\mathbb{R}^2$:

$$V_i = \{\mathbf{x} \in \mathbb{R}^2 : d(\mathbf{x}, \mathbf{x}_i) < d(\mathbf{x}, \mathbf{x}_j) \ \ \forall j \neq i\} \tag{B.1}$$

where $d(\mathbf{x}_i, \mathbf{x}_j)$ is the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$.

B.2 Laplace Interpolant (excerpted from [32])

The Voronoi cell, which is shown in Figure B.1, is adopted as the computation cell and an integral balance law is used to derive the finite difference for the diffusion operator. Consider the following 2D steady-state diffusion equation with Dirichlet boundary conditions:

$$\mathcal{L}f = \nabla^2 f(\mathbf{x}) = 0 \ \text{ for } \ \mathbf{x} \in \Omega \subset \mathbb{R}^2 \tag{B.2}$$

$$f(\mathbf{x}) = g(\mathbf{x}) \ \text{ for } \ \mathbf{x} \in \partial\Omega \tag{B.3}$$

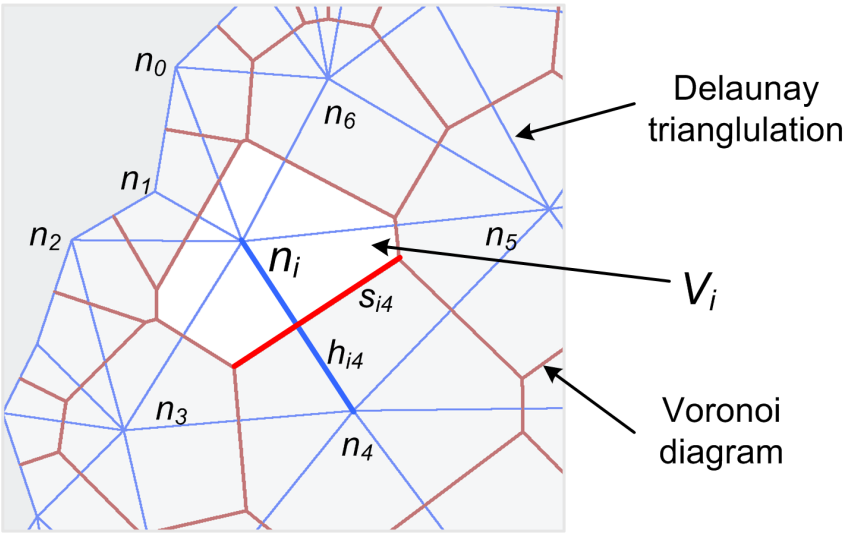


Figure B.1: Voronoi diagram and natural neighbors

where $\mathcal{L}$ is the Laplacian operator, ∇ is the gradient operator, Ω is an open set in 2D, and $\partial\Omega$ is the boundary of Ω. The above diffusion equation, which written in flux form, is used to describe diffusion processes such as heat or mass transfer, or the potential in electrostatics.

The model diffusion problem in Equation B.2 and B.3 is solved using a finite difference method. The discrete form for the model problem is written as

$$\mathcal{L}_h f = \nabla^2 f(\mathbf{x}_i) = 0, \quad i = 0, 1, \ldots, m-1 \text{ for } \mathbf{x}_i \in \Omega \subset \mathbb{R}^2 \tag{B.4}$$

$$f(\mathbf{x}_i) = g(\mathbf{x}_i) \text{ for } \mathbf{x}_i \in \partial\Omega \tag{B.5}$$

where m is the number of the nodes in the domain, $\mathcal{L}_h$ is the discrete diffusion operator, and h denotes a measure of the nodal spacing.

Now, let's find the discrete approximation for the diffusion operator at node i, that is, $\mathcal{L}_h f(\mathbf{x}_i)$ given in Equation B.4. The starting point is the balance (conservation) law for the divergence of the flux over the Voronoi cell V_i (see Figure B.1). The flux form can be written as

$$(\nabla \cdot \mathbf{q})_i = \lim_{A_i \to 0} \frac{\int_{A_i} \nabla \cdot \mathbf{q} d\Omega}{\int_{A_i} d\Omega} = \lim_{A_i \to 0} \frac{\int_{\partial A_i} \mathbf{q} \cdot \mathbf{n} d\Gamma}{A_i} \tag{B.6}$$

where Gauss' (divergence) theorem has been invoked, and A_i is the area of the Voronoi cell V_i. Since the flux can be given $\mathbf{q} = \nabla f$,

$$(\nabla^2 f)_i = \lim_{A_i \to 0} \frac{\int_{\partial A_i} \nabla f \cdot \mathbf{n} d\Gamma}{A_i} = \lim_{A_i \to 0} \frac{\int_{\partial A_i} \frac{\partial f}{\partial n} d\Gamma}{A_i} \tag{B.7}$$

To find the discrete form for the diffusion operator, the evaluation of Equation B.7 on the boundary of the Voronoi cell, V_i and a simple central-difference (cell-based) approximation for the derivative of f normal to the Voronoi edge (see Figure B.1) are used.

$$(\mathcal{L}_h f)_i = \frac{\int_{\partial A_i} \frac{\partial f}{\partial n} d\Gamma}{A_i} = \frac{1}{A_i} \sum_{j=0}^{n-1} \frac{f_j - f_i}{h_{ij}} s_{ij} = 0 \tag{B.8}$$

where n is the number of natural neighbors for node i, h_{ij} is the distance between nodes i and j, and s_{ij} is the length of the Voronoi edge associated with nodes i and j (see Figure B.1). Now Equation B.8 can give the following interpolation formulae for f_i with $f_j, \quad j = 0, 1, \ldots, n-1$, which are the values at natural neighbors for $\mathbf{x}_i$.

$$f_i = \frac{\sum_{j=0}^{n-1} \frac{s_{ij}}{h_{ij}} f_j}{\sum_{j=0}^{n-1} \frac{s_{ij}}{h_{ij}}} = \sum_{j=0}^{n-1} c_{ij} f_j \quad \text{where} \quad c_{ij} = \frac{\frac{s_{ij}}{h_{ij}}}{\sum_{k=0}^{n-1} \frac{s_{ik}}{h_{ik}}} \tag{B.9}$$

As shown in Figure B.1, the number of the neighboring nodes of node i is generally quite small compared to the total number of nodes existing in the leaf model. Therefore the system matrix of which the i-th rows is expressed by the coefficients in Equation B.9 becomes a sparse matrix. The system interpolation equations having such a sparse matrix can be efficiently solved by a sparse matrix solver like SparseIt++ [12].

Appendix C

Manual of the Interactive Modeler

C.1 Profile Tab Menu

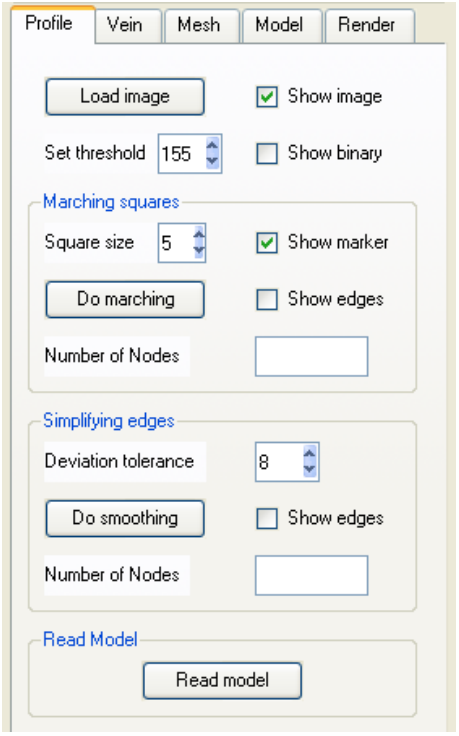


Figure C.1: Profile tab menu

1. To load an image, push [**Load image**] button. The image should be square and in png[1] format.

2. To show the loaded image, click the check box labeled '**Show image**'.

3. To binarize the loaded image, set a threshold value for the binarization, which should be in from 0 to 255, which corresponds to $2^8 - 1$. The spin box labeled '**Set threshold**' is used for setting for the threshold value.

4. To show the binarized image, click the check box labeled '**Show binary**'. At this time, the check box labeled '**Show image**' should be unchecked.

5. To extract outline of the leaf image, first set the size of the square used in the marching square using the spin box labeled '**Square size**', then push [**Do marching**] button. The text box labeled '**Number of Nodes**' shows how many nodes are generated while applying the marching square method to the leaf image.

6. To show the node markers, click the check box labeled '**Show marker**'.

7. To show the edge lines, click the check box labeled '**Show edges**'.

8. To decide how faithful to the original boundary the simplified boundary is, set the tolerance required for the simplifying method using the spin box labeled '**Deviation tolerance**'.

9. To simplify the original boundary with too many unnecessary nodes, push [**Do smoothing**] button.

10. To use the model made before, push [**Read model**] button. Then, a file open dialog box pops up. A user can select a model data file using the dialog box.

C.2 Vein Tab Menu

1. To edit the nodes which are generated by the marching square method and reduced by the simplifying method as if the nodes lie on a plastically deformable wire, click the check box labeled '**Node edit mode**'. If this box is checked, then a user freely move a node on the screen.

[1]This is an acronym for Portable Network Graphics and is an extensible file format for the lossless, portable, well-compressed storage of raster images.

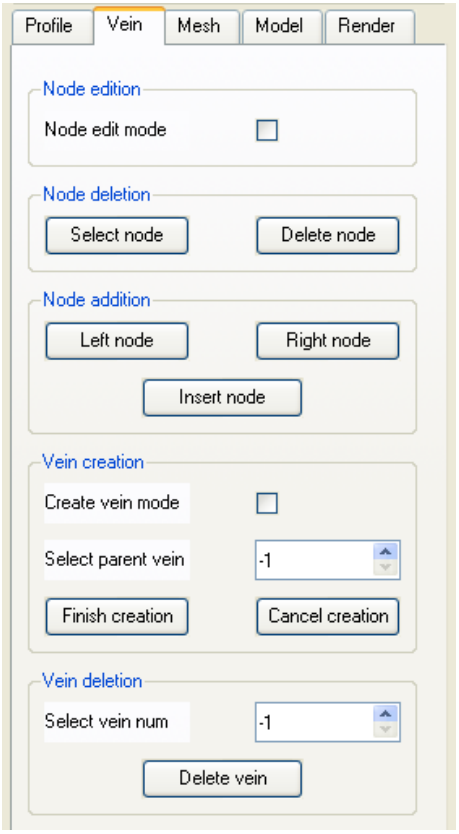


Figure C.2: Vein tab menu

2. To delete a node, push [**Select node**] button and if the color of the node is changed, then push [**Delete node**] button. If the node is deleted, then two neighboring nodes are connected.

3. To add a node between two nodes, first select the select the left node by pushing [**Left node**] button, then clicking the left. By the similar way, select the right node. After pushing [**Insert node**] button, then click an appropriate position on the screen where a new node will be located.

4. To create veins, click the check box labeled '**Create vein mode**'.

5. To specify a parent vein, set the parent vein using the spin box labeled '**Select parent vein**'. If the number is '-1', then it means the newly generated vein starts from the leaf boundary. The first created vein has '0' and then the vein

number is increased by 1. If a parent vein is chosen, the color of the vein is changed, so the user can easily recognize which vein is the parent vein.

6. If a parent vein is selected, then the newly created vein can only start from the parent vein, that is, the user should select a node in the parent vein as a starting node. If something wrong happens, then the user can cancels the vein creation by pushing [**Cancel creation**] button.

7. When a new node created on the new vein corresponds to a node on the boundary, the vein creation is automatically ended and the color of the newly created vein is changed. Then the user should push [**Finish creation**] button to complete the vein creation. If the user want to end the vein creation, the user can push [**Finish creation**] button even when the end node of the newly created vein does not reach the boundary.

8. To delete a vein and all its child veins, first select the vein number using the spin box labeled '**Select vein num**', then push [**Delete vein**] button.

C.3 Mesh Tab Menu

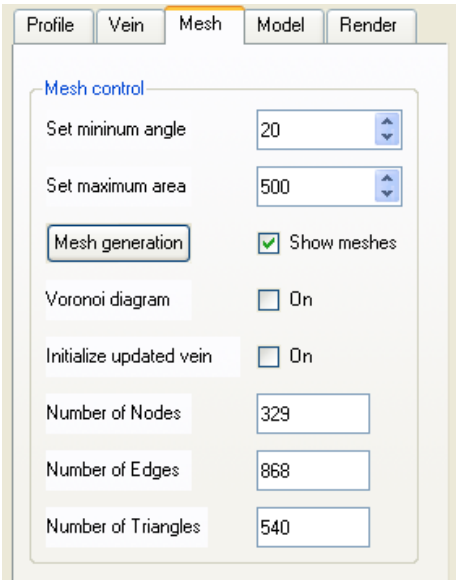


Figure C.3: Mesh tab menu

1. To triangulate the leaf domain using the conforming Delaunay triangulation, the user should set two parameters to limit the minimum angle and the maximum area in the triangles generated. Two spin boxes labeled '**Set minimum angle**' and '**Set maximum area**' can be used for setting the parameters. The default values for the two parameters are 20° and 500 p^2, respectively.

2. After setting the parameters for the conforming Delaunay triangulation, push [**Mesh generation**] button. Once the triangulation is completed, the three text boxes show how many nodes, edges and triangles are generated.

3. To show the meshes, click the check box labeled '**Show meshes**'.

4. To show the Voronoi cells, which are generated while triangulating the leaf and required for the Finite Volume Method, click the check box labeled '**Voronoi Diagram**'.

5. The check box labeled '**Initialized updated vein**' is used to decide whether the triangulation update the venation model or not. If a new leaf is modeled, then this check box is unchecked, which is a default state. If a previous model is loaded, then this check box is checked.

C.4 Model Tab Menu

1. To transform the venation model, click the check box labeled '**Vein edit mode**'. If this check box is checked, the split windows show up. However, if necessary, the user can switch the windows.

2. To select the vein to be transformed, set the vein number using the spin box labeled '**Set vein number**'. As the vein number changes, the color of the corresponding vein changes, so the user can easily know which vein is chosen. If a vein is selected, then the user can modify the vein shape in 3D.

3. Three spin boxes are used set the physical dimensions of the vein like starting radius, ending radius and offset from the leaf blade. A vein is modeled as a generalized cylinder in ray-traced rendering.

4. To change the default transformation mode with the reduced control nodes mode, click the check box labeled '**control node edit mode**'.

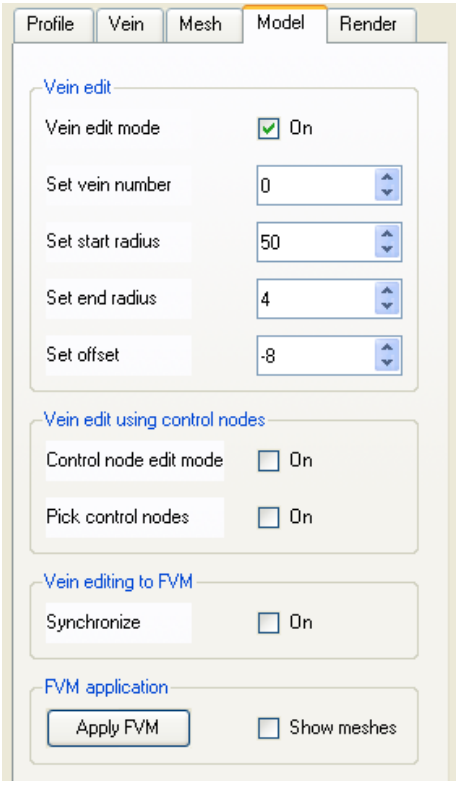


Figure C.4: Model tab menu

5. To select specific control nodes, click the check box labeled '**Pick control nodes**'. The '**control node edit mode**' check box should be checked for selecting control nodes. If user clicks a node on the selected vein, the node changes its color. The starting node and the ending node should be selected.

6. To transform the selected vein with the reduced control nodes, the check box labeled '**Control node edit mode**' should be checked and the check box labeled '**Pick control nodes**' should be unchecked. The user can deform the vein by clicking a control node on the vein and dragging the node while pressing.

7. To interpolate the leaf blade mesh to conform to the deformed venation model, push [**Apply FVM**] button.

8. To synchronize the deformation of the vein to the deformation of the blade, click the check box labeled '**Synchronize**'. This shows instantly the deformed

blade whenever the user transforms the vein.

C.5 Render Tab Menu

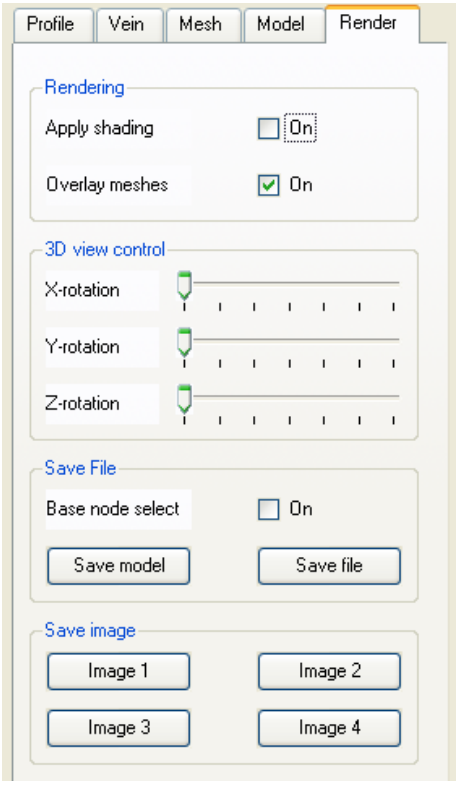


Figure C.5: Render tab menu

1. To show an OpenGL rendering image texture mapped with the loaded image, click the check box labeled '**Apply shading**'.

2. To remove the overlayed meshes, which is default, uncheck the '**Overlay meshes**' check box.

3. The 3 slide bars are used to rotate the leaf mesh model with respect to x, y and z axes of the leaf. Therefore, the user can control the perspective view of the OpenGL rendering image.

4. To set the coordinates of a specific node as $0, 0, 0$, click the check box labeled '**Base node select**', then click the node.

5. To save the mesh model for later modification, push [**Save model**] button. Then a file save dialog box shows up and the user can save the model.

6. To export the mesh model for an off-line ray tracer, which is POV-Ray [24], push [**Save file**] button. See Figure C.6 for the sample export file.

7. To save the current screen as an image file, push one of 4 buttons in the '**Save image**' box.

```
#declare vein0 = sphere_sweep {
        cubic_spline,
        n //number of nodes
        < x, y, z>, r // nodel position and radius
        ...
}

... // give data for other veins using the above format

#declare vein = union
{
        object {vein0}
        //give data for other veins
}

#declare leaf = mesh2 {

        vertex_vectors {
              n //number of vertices
              <x, y, z>, //nodal position
              ...
        }

        normal_vectors {
              n //number of normal vectors
              <x, y, z>, //normal vector component
              ...
        }

        uv_vectors {
              n //number of uv nodes
              <u, v>, //uv position
              ...
        }

        face_indices {
              n //number of faces (triangles)
              <n1, n2, n3>, //nodal number
              ...
        }
}
```

Figure C.6: A sample export file to POV-Ray [24]

Bibliography

[1] *The American Heritage Dictionary of the English Language*. Houghton Mifflin Company, 4th edition, 2000.

[2] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering*. A K Peters, Natick, Massachusetts, 2nd edition, 2002.

[3] G.V.G. Baranoski and J.G. Rokne. *Light Interaction with Plants: A Computer Graphics Perspective*. Horwood Publishing, Chichester, UK, 2004.

[4] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming With Qt 3*. Bruce Perens' Open Source Series. Prentice Hall, Upper Saddle River, New Jersey, 2004.

[5] Jules Bloomenthal. Modeling the mighty maple. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 305–311. ACM Press, 1985.

[6] Jules Bloomenthal and Chek Lim. Skeletal methods of shape manipulation. In *SMI '99: Proceedings of the International Conference on Shape Modeling and Applications*, page 44. IEEE Computer Society, 1999.

[7] Brian Bracegirdle and Patricia H. Miles. *An Atlas of Plant Structure*, volume 1. Heinemann Educational Books, London, 1971.

[8] Christopher Brickell, Trevor Cole, and Judith D. Zuk. *A-Z Encyclopedia of Garden Plants*. Dorling Kindersley Limited, London, England, 2003.

[9] S. H. Crandall, N. C. Dahl, and T. J. Lardner. *An Introduction to the Mechanics of Solids*. McGraw-Hill, 2nd edition, 1978.

[10] A. Fahn. *Plant Anatomy*. Pergamon Press, Oxford, England, 3rd edition, 1982.

[11] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics (2nd ed. in C): principles and practice*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[12] P. A. Forsyth. *SPARSEIT++ User Guide*. Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1, 2002.

[13] Leaf Architecture Working Group. *Manual of Leaf Architecture: morphological description and categorization of dicotyledonous and net-veined monocotyledonous angiosperms*. Smithsonian Institution, Washington, 1999. 65p.

[14] M. S. Hammel, P. Prusinkiewicz, and B. Wyvill. Modeling compound leaves using implicit contours. *Visual Computing: integrating computer graphics with computer vision*, pages 119–131, 1992. Proceedings of Computer Graphics International 92.

[15] Leo J. Hickey. *Anatomy of the dicotyledons*, volume 1, chapter 4. A revised classification of the architecture of dicotyledonous leaves, pages 25–39. Oxford: Clarendon Press, 2nd edition, 1979.

[16] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.

[17] David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, and Amitabh Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., 2002.

[18] N. MacDonald. *Trees and Networks in Biological Models*. John Wiley & Sons Ltd., Chichester, 1983. A Wiley-Interscience publication.

[19] Stefan Maierhofer and Robert F. Tobler. Creation of realistic plants using semi-automatic parameter extraction from photographs. Technical report, VRVis Research Center, Donau-City-Strasse 1, A-1220 Vienna, Austria, 2001.

[20] James D. Mauseth. *Plant Anatomy*. Series in the life sciences. The Benjamin/Cummings Publishing Company, Inc., Menro Park, California, 1988.

[21] L. Mündermann, P. MacMurchy, J. Pivovarov, and P. Prusinkiewicz. Modeling lobed leaves. *Proceedings of Computer Graphics International*, pages 60–65, 2003. CGI 2003.

[22] Karl J. Niklas. *Plant Biomechanics: An Engineering Approach to Plant Form and Function*. The University of Chicago Press, Chicago, 1992.

[23] Karl J. Niklas. A mechanical perspective on foilage leaf form and function. *New Phytologist*, 143:19–31, 1999.

[24] Persistence of Vision Pty. Ltd. Persistence of vision (tm) raytracer [computer software], 2004. http://www.povray.org.

[25] Carl E. Pearson. *Handbook of Applied Mathematics: Selected Results and Methods*. Van Nostrand Reinhold, 2nd edition, 1983.

[26] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[27] P. Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., 1990.

[28] Anita Roth-Nebelsick, Dieter Uhl, Volker Mosbrugger, and Hans Kerp. Evolution and function of leaf venatioin architecture: A Review. *Annals of Botany*, 87:553–566, 2001.

[29] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[30] Jonathan Richard Shewchuk. *Lecture Notes on Delaunay Mesh Generation*. University of California at Berkeley, Berkeley, CA, 1999.

[31] Lisa Streit and Wolfgang Heidrich. A biologically-parameterized feather model. *Comput. Graph. Forum*, 21(3), 2002.

[32] N. Sukumar. Voronoi cell finite difference method for the diffusion operator on arbitrary unstructured grids. *Intl. J. for Numerical Methods in Engineering*, 57:1–34, 2003.

[33] N. Sukumar, B. Moran, A. Semenov, and V. Belikov. Natural neighbor Galerkin methods. *Intl. J. for Numerical Methods in Engineering*, 50:1–27, 2001.

[34] Julia F. Taylor-Hell, Gladimir V. G. Baranoski, and Jon G. Rokne. State of the art in the realistic modeling of plant venation systems. *International Journal of Image and Graphics*, 5(3):1–16, 2005.

[35] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison Wesley, third edition, 1999.

[36] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data Structure for Soft Objects. *The Visual Computer*, 2(4):227–234, February 1986.

Index