Computer-Generated Pen-and-Ink Illustration of Trees

Oliver Deussen[*] Thomas Strothotte
Faculty of Computer Science, University of Magdeburg, Germany

Abstract

We present a method for automatically rendering pen-and-ink illustrations of trees. A given 3-d tree model is illustrated by the tree skeleton and a visual representation of the foliage using abstract drawing primitives. Depth discontinuities are used to determine what parts of the primitives are to be drawn; a hybrid pixel-based and analytical algorithm allows us to deal efficiently with the complex geometric data. Using the proposed method we are able to generate illustrations with different drawing styles and levels of abstraction. The illustrations generated are spatial coherent, enabling us to create animations of sketched environments. Applications of our results are found in architecture, animation and landscaping.

CR Categories: I.3.3 [Picture/Image Generation]: Display algorithms— [I.3.7]: Three-Dimensional Graphics and Realism—Animation

Keywords: Biological Systems, Frame Buffer Tricks, Non-Realistic Rendering

1 Introduction

During the last years, a variety of techniques have been proposed to sketch and non-photorealistically render objects. Research in this area was driven by the realization that drawings are able to convey visual information in a different way than photorealistic images do [21]. This is one of the reasons why a large percentage of images in many books are drawings (cf. [22]).

While the proposed methods allow creating line drawings of many objects and in many different styles, the illustration of plants has so far been neglected. This is surprising because drawings of these objects are needed in areas like architecture and landscaping. In both cases early designs are preferentially visualized as abstract line drawings that often include many trees [18].

In this paper we propose a method for automatic pen-and-ink illustration of trees. The approach allows us to create a variety of illustration styles. The underlying models are realistic 3-d plant geometries generated with the xfrog modeling system proposed by Lintermann and Deussen [8], but any other surface-oriented plant model can also be used.

[*]Universitätsplatz 2, D-39106 Magdeburg, Germany, odeussen@acm.org http://isgwww.cs.uni-magdeburg.de/˜deussen

In comparison to the art-based illustration styles for trees invented by Kowalski et al. [7], we are more interested in visually representing specific plants than to create generic representations. Our aim is to provide the user with a transition from a tree illustration with a realistic plant-specific look to an abstract representation consisting of only a few strokes. This enables the user to select a global degree of abstraction while at the same time enabling the system to draw plants in the background with a higher abstraction level. In combination with different drawing styles, this helps to adapt the visual appearance of the plants to other objects and also, for instance, allows the user to focus the viewer's attention on a certain part of the scene.

Among the various plant types and their combinations, we focus on complex trees and bushes. Collections of these objects are most interesting in architecture and landscaping. Also both categories require abstract visual representations as it is impossible to draw all the geometry in detail.

1.1 Related Work

Related work in illustrating trees was done in the field of non-photorealistic rendering and also in botanical plant generation.

Probably the first article with illustrated plants was presented by Yessios [25]. In an architectural framework he used abstract plant symbols and combined them with stones and ground materials.

Alvy Ray Smith, one of the early authors dealing with fractals and formal plant descriptions created a “cartoon tree” with small disks representing bunches of leaves [19]. A similar representation with smaller disks was used by Reeves and Blau [14] to constitute their structured particle systems for rendering realistic trees. The idea of representing numerous botanical leaves by an abstract geometric primitive inspired us (like Kowalski et al. [7]) to work on pen-and-ink illustrations of trees.

A line drawing is usually created by combining a number of brush or pencil strokes. Researchers in non-photorealistic rendering resemble that process by using virtual brushes. Strassmann [20] proposed the “path-and-stroke” metaphor: a path is defined and a physically simulated brush is used to generate the stroke. Hsu et al. [6] extended the metaphor by using general objects like textures, images and recursively defined fractals that are drawn along a given path.

Salisbury et al. [16] described a method for directing the strokes in line drawings on the basis of vector fields. In their paper they also showed an interactively generated tree image. Winkenbach and Salesin [23, 24] presented a variety of methods for the automatic generation of pen-and-ink illustrations. In contrast to Strassmann and Hsu et al. they do not work with individual strokes but with artistically elaborate stroke textures.

Sasada [17] presented some tree sketches in an architectural environment. He used images of synthetic tree skeletons that were mapped onto view-facing polygons. The method of Aono and Kunii [1] was used to create the skeletons, the foliage was not visualized in their computer-generated trees.

Kowalski et al. [7] generated abstract sketches of trees by using geometric primitives like spheres for defining rough approximations of a tree's foliage. These primitives were rendered conventionally to achieve gray-scale images. In a second step the images were used to place graftals – small objects representing leaves or hair – on the surfaces by applying the "difference image algorithm" proposed earlier by Salisbury et at. [16]. Doing so it is possible to create sketched images of generic trees, bushes, and grass.

In our work we start from a different point. Our models are detailed tree models consisting of a tree skeleton and leaves. Our line drawings are the result of visually combining many drawing primitives instead of placing graftal objects on some large geometries. A drawback of our approach is that we potentially have to deal with more input data. The solution to this problem is to represent a tree at several levels of detail. This makes it possible to adapt the geometric representation to what should be presented on the screen: If a more detailed drawing is to be created, a more detailed geometric tree description is used.

The use of realistic tree models thus offers some major advantages: We can make use of existing tree libraries, our tree illustrations can be abstract but we are also able to draw a specific plant. If the scene is to be rendered photorealistically later, the visual representation does not differ much from its illustrated counterpart. Having access to the detailed 3-d data enables us also to animate the complex line drawings with sufficient spatial and temporal coherency. Another advantage is the correct, tree-like shadow generation of our models.

The main contribution of our work is an efficient way of generating the illustration of realistic plant models using abstract drawing primitives; furthermore, we present a "depth difference algorithm" to determine important silhouette lines, which allows us to generate different levels of visual abstraction.

The remainder of this paper is organized as follows: Section 2 reviews the artistic work on illustrating trees, in Section 3 our synthetic illustration algorithm is given. Section 4 shows results, and in Section 5 we give some conclusions.

2 Traditional Illustration of Trees

Among the various styles used by artists to render trees (for a large set of examples see [3]) one can distinguish between flat styles that solely represent the shape of a tree and others that also approximate natural light interaction (cf. [9]).

The tree skeleton is usually drawn up to the second branching level, primarily by silhouette lines and crosshatching on the stem surface. The shape of the foliage is either represented by an abstract outline or by a collection of many small objects which do not necessarily resemble natural leaves but instead represent the characteristics of the foliage. In addition, the outline is sometimes drawn by many small line segments or just a few strokes.

The visual appearance of the foliage can be divided into three areas. The top of the tree is usually in the direct light and is therefore visualized by only some details and its outline. In the half shadow, more details are drawn to achieve an appropriate gray level. In this area the outline of the leaves is often drawn in detail. The third area is the shaded part. The three areas are generally not found in a single illustration, often only the half shadow and the full shadow region is drawn. Sometimes the complete foliage is represented uniformly.

Artists use different methods to generate shadows on the foliage: in many styles more details are drawn and thick lines are used, sometimes with whole areas being drawn in black. Other styles add crosshatching to the foliage.

A method for the synthetic illustration of trees must propose solutions to several subproblems: First, the stem skeleton must be represented properly by silhouette lines and crosshatching. Second, an abstract leaf representation must be found that enables the user to represent different types of leaves as well as different illustration styles. Third, drawing the leaves must be modulated by the three areas: the leaves in the light must be represented solely by the outline of the foliage, leaves in the half shadow should be drawn with detailed outline or additional crosshatching, and regions of deep shadow are to be added appropriately.

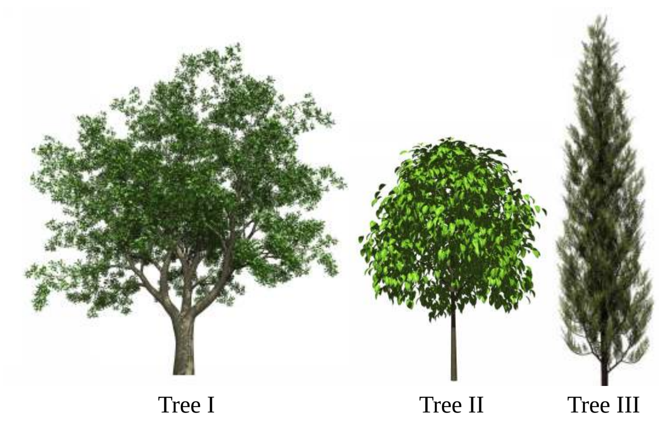

Figure 1: Photorealistically rendered images of the synthetic sample trees: Tree I: complex tree; Tree II: young lime tree; Tree III: conifer.

3 Automated Illustration of Trees

The first step to create a tree illustration is to create a tree with a conventional tree modeling program. As mentioned above, we use the xfrog modeling system [5, 8] for that purpose. The final model – some of them are shown in Figure 1 – is preprocessed and two files are created.

In the first file, the geometry of the tree skeleton is stored. Like artists we only draw the trunk and branches up to the second order in most of our illustrations with higher order branches being removed.

The second file stores the leaves as particles each with a position and a normal vector. The normal vectors are obtained by using the normal vector of the original leaves. If too much data is generated for all the leaves – Tree I in Figure 1 has about 183,000 leaves – we reduce them in the modeling system by reducing the number of leaves at each twig. If this is still too much we position the particles at the branching positions of the highest-order twigs. In the case of Tree I we end up with 8,800 particles.

The illustrations are generated as follows: The trunk and branches are drawn by applying techniques known from non-photorealistic rendering. The foliage is rendered by representing each leaf by a drawing primitive – a disk or arbitrary polygon facing the viewer – and by applying the depth difference algorithm to determine which part of the primitive outlines are to be drawn. Shadows can be applied at this stage, vegetation on the ground can also be added and is processed the same way. The resulting drawings are then composed to constitute the final image.

3.1 Drawing the tree skeleton

The tree skeleton is an assembly of generalized cylinders each representing a branch. The surface is more or less smooth, which allows us to apply analytical silhouette algorithms such as the one proposed by Markosian et al. [10] or the hybrid solution of Rakar and Cohen [13] to generate the outline. The depth difference algorithm proposed below can also be applied (see Figure 2).

In addition, the skeleton is shaded conventionally to find dark regions. These regions are then crosshatched in the drawing. The "Difference Image Algorithm" [16] that places individual strokes according to the local gray tone of an image is one solution to this problem. For our purpose a simpler method is sufficient that works with a variant of the Floyd Steinberg method [4].

The algorithm places short strokes instead of pixels if the cumulated gray scale is above a given threshold. The area of the stroke is determined and the corresponding error value is subtracted from the neighboring pixel values. The direction of the strokes is either at random or affected by the normal vector of the stem geometry. A similar technique for directing strokes was already used in [10].

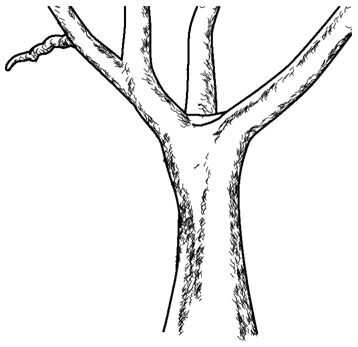

Figure 2: The trunk and main branches of Tree I are extracted and rendered by silhouette lines and cross hatching.

3.2 Drawing the foliage

The foliage of a tree differs by its very nature from all smooth surfaces and therefore must be handled separately. Several thousand individual surfaces must be combined visually into a shape or a set of strokes. In our first experiments, we placed special textures on the leaves of our realistic tree models that looked like strokes. This is a fast and simple method, but the generated images never appeared like drawings.

The observation that artists do not draw leaves correctly but try to represent their visual appearance led us to use abstract drawing primitives. Each leaf is represented by the outline of such a primitive, whereas its position is determined by the 3-d leaf position and the size is controlled by the user. A very simple drawing primitive is a view-facing disk. While other abstract drawing primitives are given below, we first describe the second ingredient of our approach, the depth difference algorithm, by using this primitive.

Depth differences

Depth differences are used to determine what part of each drawing primitive is to be drawn to constitute the foliage. Saito and Takahashi [15], two of the early authors in non-photorealistic rendering, used the depth-buffer to determine the outline of objects which were used to enhance photorealistic images. First and second order derivatives in the depth-buffer were additionally computed to find important lines on the surface of the objects.

While first and second order depth derivatives are helpful to find important lines on smooth surfaces, zero order derivatives are helpful for determing important lines in collections of isolated surfaces like assemblies of drawing primitives: The outline of a primitive is drawn if the maximal depth difference of the surface to the neighboring surfaces is above a given threshold.

Instead of computing the differences analytically - which in the case of complex tree models is computationally expensive - we use the depth buffer for this purpose. The primitives are drawn as solids, the depth buffer is obtained, and for each pixel the depth difference is computed by comparing its depth value with all neighbor values. The maximal positive difference for each pixel is taken. This value indicates how far the pixel is in front of its neighboring pixels. It is stored in a separate buffer.

For interactive applications those pixels with a depth difference exeeding a given depth difference threshold are directly used to create a bitmap of the outlines. For printing purposes a vectorization is performed to obtain stroke pathes (see Section 3.4).

It is well known that the values in the depth buffer have a non-linear characteristic. The depth z in the camera coordinate system or eye coordinates rsp. is determined from a depth value d ($d \in [0..1]$) by

$$z = \frac{\frac{z_1 z_0 (d_1 - d_0)}{z_1 - z_0}}{d - \frac{(z_1 + z_0)(d_1 - d_0)}{2(z_1 - z_0)} - \frac{(d_1 + d_0)}{2}} \qquad (1)$$

where d_0 and d_1 are minimal and maximal values represented in the depth buffer, and z_0 and z_1 the corresponding depth values of the near and far clipping plane in the camera projection (cf. [11]).

The depth differences can be computed for the depth values in eye coordinates to achieve linear differences or directly for the depth buffer values. In the second case depth differences for remote objects correspond to much larger differences in eye coordinates. In consequence the objects are represented by fewer lines.

To determine a depth difference threshold sufficient for the eye coordinates we compute the depth range of the tree and choose a percentage of this range, for example 10 percent. Analogously this is done with depth buffer values. The examples in this paper were rendered using depth buffer values directly by setting $d_0 = 0, d_1 = 65535, z_0 = 1$, and $z_1 = 11$. The depth difference in eye coordinates ($z_1 - z_0$) is approximately the one of real trees.

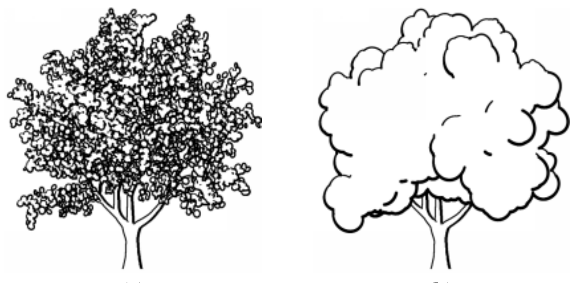

Figure 3: Tree I rendered with varying disk size and depth difference threshold: a) size=0.15, threshold=1000; b) size=0.7, threshold=2000.

Figure 3 shows two sketches of Tree I. In Figure 3(a) small disks are used and the threshold is low. This results in high detail and a good approximation of the real model. A more abstract rendering is achieved if disk size and threshold are enlarged (Figure 3(b)).

The threshold can be constant over the whole image or can be modulated by other buffers. In Figure 4(c) a shadow buffer was used to reduce the threshold in the shadow. The resulting image shows more detail in this area.

Abstract drawing primitives

Apart from disks, a number of drawing primitives can be used to represent the leaves. In Figure 4(a) a set of nine polygons was generated to represent leaves from different views. The normals of the given particles were used to interpolate the individual shapes of the leaves from the polygons. Using this interpolation scheme, a 3-d shape can be denoted without strictly adhering to perspective transformations.

If appropriate polygons are used, a representation similar to the graftals in [7] can be generated, but our interpolation method offers more freedom, allowing nearly all forms of leaves to be used.

The user is also able to decide to what extent the 3-d impression is generated: the leaves in Figure 4(b) are not drawn from the full range of views, instead a subset is used to generate a style between uniform shapes and the full 3-d impression. In Figure 4(c) the shape of the leaves is drawn only in the shadow region, additionally the linewidth is increased.

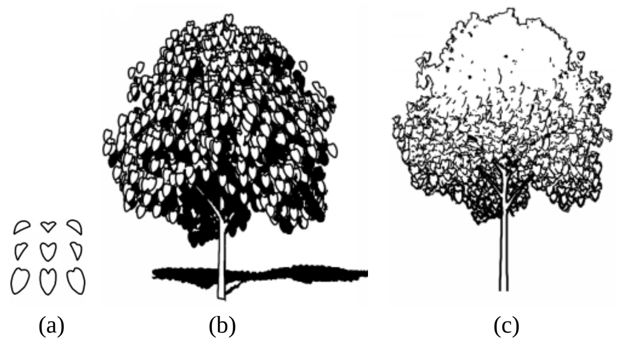

Figure 4: Two sketches of Tree II. a) The leaves are rendered using interpolated polygons from the nine given samples; b) Shadow is drawn in black, threshold=100. c) Threshold is set to 6,000, shadow is represented by detail.

3.3 Level-of-Abstraction

As mentioned above, the differences in the depth buffer have a non-linear characteristic. If they are used directly instead of reprojecting them into eye coordinates, the same tree that is drawn in the front with high detail will be sketched automatically by a few strokes if it is at the back.

The effect can be modulated by changing the z_1 to z_0 ratio of the perspective projection which is the basis for Equation (1). A small ratio causes a small non-linearity, a large ratio above 100:1 results in less depth resolution in the background and therefore in a small number of strokes.

This visual level-of-abstraction can be supported by scaling the primitive size for trees in the background. In [7] a formula for a scale factor r for the object size of graftals is suggested which uses a weighted average between a scaling d/s (d desired screen space, s current screen space of the object) that generates primitives of visual constant size and primitives that have a constant object size

$$r = w(d/s) + (1 - w) \qquad w \in [0..1].$$

In our case, we additionally allow w to be above one and in this case omit the second term. Now, the abstract drawing primitives appear larger in the background, which helps to achieve clear object shapes here.

In Figure 5 the process is shown. In the tree sequence of Figure 5(a) level-of-abstraction was done on the basis of depth differences only, in Figure 5(b) the size of the drawing primitives is doubled for the tree at the back.

3.4 Software Framework

The proposed method was designed to work in two environments. First, a fast method for interactive systems was needed. Second, high quality images should be produced for printouts, animations and architectural sketches. As a consequence the software works in stages that are partly omitted for the interactive process.

In the first step, depth differences have to be determined. In the interactive environment stem and foliage are rendered together, the depth buffer is obtained and all pixels above the given depth difference threshold are drawn in black. The resulting bitmap is directly used and blended with other geometries of the scene to constitute the final image.

For drawing purposes - and also for animations with high temporal coherency - the stem and the foliage are rendered separately, the images are combined by their depth buffer values to handle occlusion. For each image a separate depth difference threshold is used later.

For many styles shadows have to be introduced. We have to use a software implementation of shadows because volume shadows based on stencil buffering (cf. [11]) do not work for the huge number of isolated surfaces in the foliage. The result is stored in a separate shadow buffer. In the interactive case, shadows are omitted.

Now the threshold is applied and the pixels above the threshold are marked. As mentioned above, the threshold can be modulated by a shadow buffer, other G-buffers (cf. [15]) or by an arbitrary spatial function.

For generating high quality images, the bitmaps of the stem and the foliage are vectorized. We implemented two methods: The first algorithm determines vectors globally for the bitmaps by applying least square fitting [12]. The second algorithm adds an index buffer, a bitmap that stores at each pixel position the primitive identification as a color value.

For each depth value above the threshold, it is now possible to obtain the primitive number, therefore vectorization can be performed for each primitive separately. This results in a higher image quality, for instance closed primitive outlines can now easily be determined and represented by closed polygons. As a drawback, the method which is slow already needs even more time since the index buffer has to be rendered and processed additionally.

In both cases the polygons are drawn by spline interpolation, and line styles may be applied. As an example, line styles are responsible for the shading effect on the tree in Figure 3(b). Among varying the line width, which was done here, the styles may also affect the direction of the line or alter the endpoints.

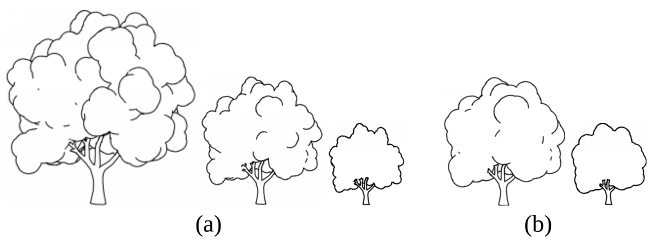

Figure 5: Tree I rendered for three different distances. a) Primitive sizes and threshold are constant for all distances. Visual abstraction is achieved automatically. b) Primitive sizes are enlarged up to the factor of two for the tree in the back.

4 Results

In Figure 6(a) and (b), Tree III is drawn using view-facing elliptic primitives of random orientation. After determining which part of each primitve has to be drawn, a small deformation was applied to each outline. This helps to achieve a more sketched drawing style.

In Figure 6(b) all visible outlines are drawn, and a threshold of 400 is used. The drawing of Figure 6(a) was created using a slight modification of the algorithm: Only the lower part of each ellipse is drawn when visible, the threshold having a value of 100. Rendering is performed in 10 seconds on our SGI Octane (Maximum Impact), the conifer consists of 13,200 particles.

The maple tree of Figure 6(c) consists of 16,200 particles which is far below the original model with 200,000 leaves. The parametrization of Figure 6(a) was used, threshold was set to 1,000.

Figure 6(d) was created similar to Figure 6(a). Only 2,300 particles are used, this causes nearly each ellipse to be visible, as a result a lot of semicircles appear. Figure 6(e) used drawing primitives in the form of real leaves, a very small threshold of 10 causes all visible outlines to be drawn.

The tree in Figure 6(f) consists of 90,000 particles, very small ellipses were used, shadow is added as black regions. The ground is represented by 23,000 elliptic primitives of larger size. Only the shadow is drawn, no primitive outlines are used. In this case rendering is performed in about one minute.

In the interactive version of the proposed algorithm it is possible to render three trees consisting of 20,000 primitives each and 25,000 ground particles with three frames per second on our SGI Onyx 2 at lower image quality. We hope to improve this in the future.

5 Conclusion and Future Work

We have presented a framework for rendering trees in pen-and-ink. The tree skeleton and the foliage are processed separately. The trunk and branches are represented by silhouette lines augmented by crosshatching in dark areas. The foliage is drawn by using abstract drawing primitives that represent leaves. Such primitives can be circles, ellipses or other polygons. An interpolation scheme allows us to adapt the form of the primitives to the normal vector of the particles that are used as input. Depth differences are used to determine what part of the primitives are drawn.

Our experiments reveal that it is possible to create various illustration styles with our approach, and they have opened several areas of future research:

- So far, shadows were introduced into the images by shadow buffers or by raising detail in shadow regions. As mentioned in Section 2 artists sometimes use crosshatching on the leaves to represent shadow. The hatching lines in this case must interact with the leaves. An intersection method as proposed in [2] can be applied here.

- To reduce the amount of geometric data, level-of-detail has to be applied to the tree models. Currently we work with some discrete representations that sometimes cause visual artifacts if representations are changed. A continuous level-of-detail algorithm for trees will improve performance while maintaining the visual quality.

- The primary goal of our paper was to provide pen-and-ink illustrations for architecture and landscaping. Another important application are cartoons. New styles and colored versions of our images need to be developed for that purpose.

References

[1] M. Aono and T. L. Kunii. Botanical tree image generation. *IEEE Computer Graphics and Applications*, 4(5):10–34, May 1984.

[2] O. Deussen, J. Hamel, A. Raab, S. Schlechtweg, and T. Strothotte. An illustration technique using hardware-based intersections and skeletons. In *Proceedings of Graphics Interface 99*, pages 175–182. Canadian Human-Computer Communications Society, 1999.

[3] L. Evans. *The New Complete Illustration Guide: The Ultimate Trace File for Architects, Designers, Artists, and Students*. Van Nostrand Reinhold Company, 1996.

[4] R. W. Floyd and L. Steinberg. An adaptive algorithm for spatial grey scale. *Proc. Soc. Inf. Display*, 17:75–77, 1976.

[5] Greenworks GbR. Home page of the xfrog modelling software. http://www.greenworks.de.

[6] S. Hsu and I. Lee. Drawing and animation using skeletal strokes. In *SIGGRAPH '94 Conference Proceedings*, pages 109–118. ACM SIGGRAPH, July 1994.

[7] M. Kowalski, L. Markosian, J. D. Northrup, L. Burdev, R. Barzel, L. Holden, and J. F. Hughes. Art-based rendering of fur, grass, and trees. In *SIGGRAPH '99 Conference Proceedings*. ACM SIGGRAPH, August 1999.

[8] B. Lintermann and O. Deussen. Interactive modeling of plants. *IEEE Computer Graphics and Applications*, 19(1):56–65, January/February 1999.

[9] F. Lohan. *The drawing handbook*. Contemporary Books, Chicago, 1993.

[10] L. Markosian, M. A. Kowalski, S. J. Trychin, L. D. Bourdev, D. Goldstein, and J. F. Hughes. Real-time nonphotorealistic rendering. In T. Whitted, editor, *SIGGRAPH '97 Conference Proceedings*, pages 415–420. ACM SIGGRAPH, 1997.

[11] T. McReynolds and D. Blyth. Advanced graphics programming techniques using OpenGL. SIGGRAPH '98 Course Notes, ACM SIGGRAPH, 1998.

[12] J. R. Parker. Extracting vectors from raster images. *Computers & Graphics*, 12(1):75–79, 1988.

[13] R. Raskar and M. Cohen. Image precision silhouette edges. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 135–140. ACM SIGGRAPH, April 1999.

[14] W. T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 313–322, July 1985.

[15] T. Saito and T. Takahashi. Comprehensive rendering of 3-d shapes. In *Computer Graphics (Proc. SIGGRAPH 90)*, volume 24(4), pages 197–206. ACM SIGGRAPH, 1990.

[16] M. Salisbury, M. Wong, J. F. Hughes, and D. Salesin. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH '97 Conference Proceedings*. ACM SIGGRAPH, 1997.

[17] T. T. Sasada. Drawing natural scenery by computer graphics. *Computer Aided Design*, 19(4):212–218, 1987.

[18] J. Schumann, T. Strothotte, A. Raab, and S. Laser. Assessing the effect of non-photorealistic images in computer-aided design. In *ACM Human Factors in Computing Systems, SIGCHI '96*, pages 35–41, April 13-15 1996.

[19] A. R. Smith. Plants, fractals and formal languages. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):1–10, July 1984.

[20] S. Strassmann. Hairy brushes. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(3):225–232, 1986.

[21] C. Strothotte and T. Strothotte. *Seeing Between the Pixels: Pictures in Interactive Systems*. Springer-Verlag, Berlin-Heidelberg-New York, 1997.

[22] T. Strothotte, B. Preim, A. Raab, J. Schumann, and D. R. Forsey. How to render frames and influence people. *Computer Graphics Forum*, 13(3):455–466, 1994.

[23] G. Winkenbach and D. Salesin. Computer-generated pen-and-ink illustration. In *SIGGRAPH '94 Conference Proceedings*, pages 91–100. ACM SIGGRAPH, 1994.

[24] G. Winkenbach and D. Salesin. Rendering parametric surfaces in pen and ink. In *SIGGRAPH '96 Conference Proceedings*, pages 469–476. ACM SIGGRAPH, 1996.

[25] C. I. Yessios. Computer drafting of stones, wood, plant and ground materials. *Computer Graphics (Proceedings of SIGGRAPH 79)*, 13(3):190–198, 1979.

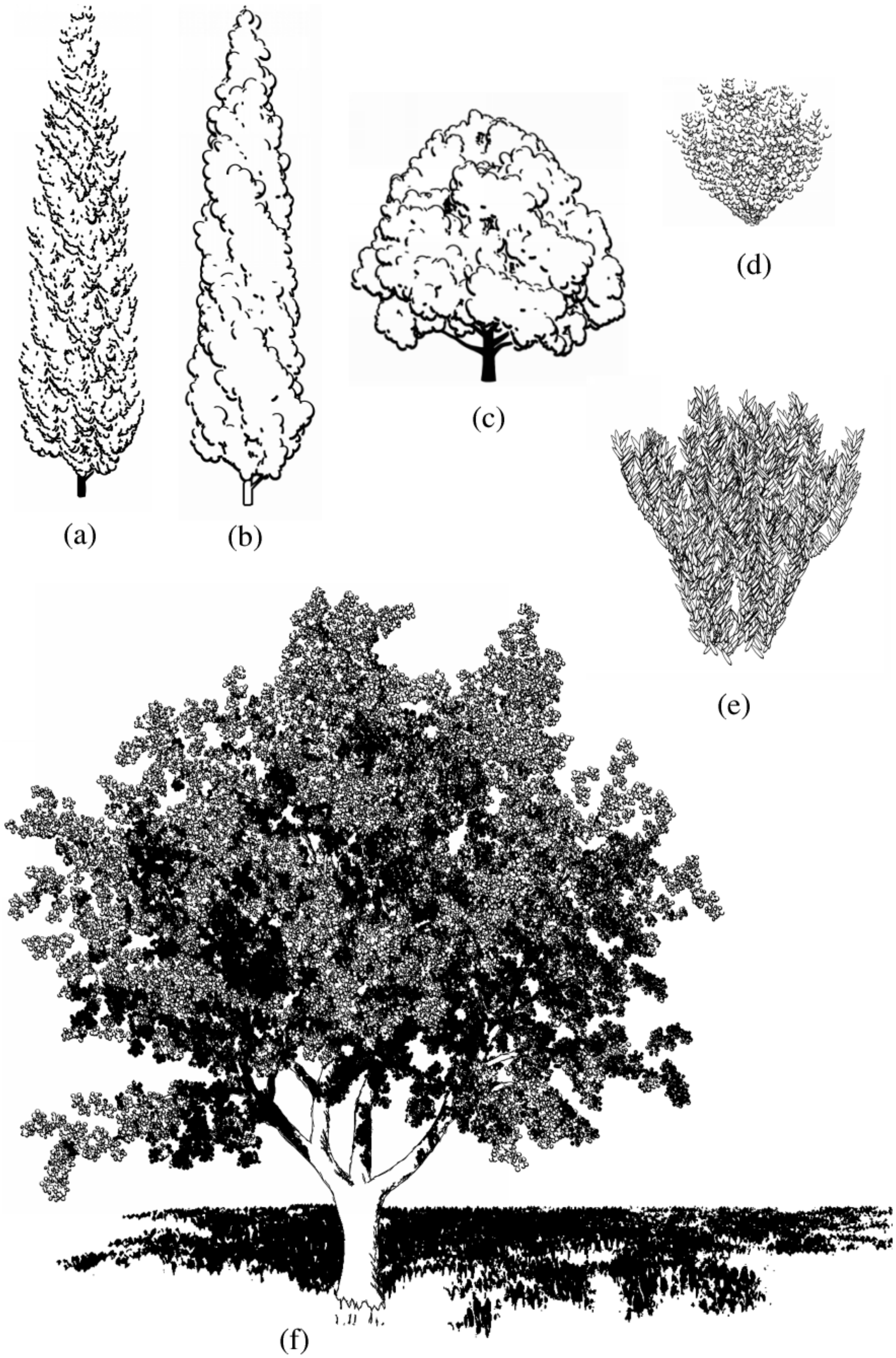

Figure 6: Several trees shaded with different styles. See Section 4 for details.