GPU-generated “Parameterized” Trees

Amit Shesh[*]
Illinois State University

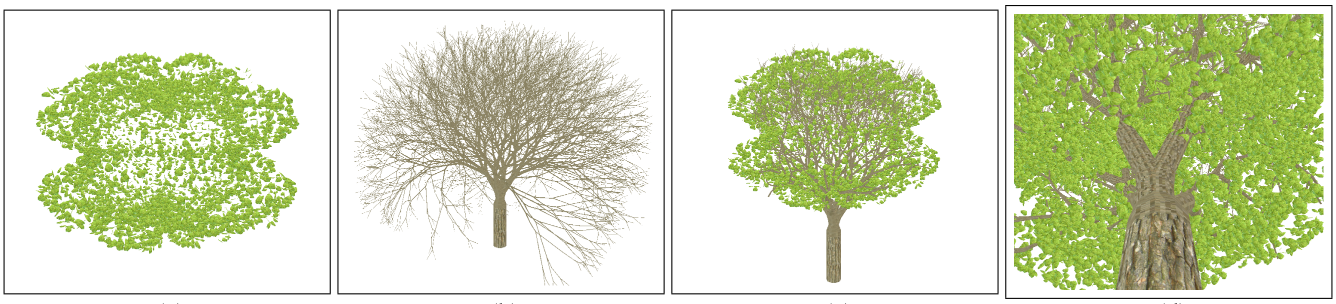

(a) (b) (c) (d)

Figure 1: **Trees as parameterized functions.** We create trees by independently creating the canopy shape using several 2D functions and then combining it with a pre-selected and pre-computed branch structure to create the tree model. (a) The canopy shape generated from 2D functions. Each leaf is drawn as an oriented alpha-textured quad. (b) An example branch structure out of a pre-generated library of branch structures. This particular example was constructed using an L-system. (c) The canopy and the branch structure are fitted together by clipping the branches to form a complete tree. (d) Unsupported leaf locations which are a possibility in our technique are not apparent even up close, making our technique suitable for many 3D applications. In this paper we concentrate on creating different tree canopies instead of branching structures as canopies define the shape of large, mature trees.

Abstract

Incorporating different-looking trees in a single graphics application involves either loading numerous polygonal/point models, or generating them algorithmically. In any case, this increases the demand of rendering resources both in terms of computing power and memory to hold all models simultaneously. This paper presents a novel method that produces different-looking trees from a common domain starting from the same polygonal model. Since the shape of the canopy decides the appearance of large trees, we focus on generating canopies automatically from the same polygonal model and some parameters. Thus the generated canopies can be thought of as functions of a parameterized domain. A branch structure created separately then completes the tree model. Using this method, a program can maintain a single copy of the polygonal model, and create different tree models from it by merely changing these parameters. Our method can be efficiently implemented on a GPU, thereby allowing us to store only a few models directly in GPU memory and creating different-looking tree models from them at runtime.

CR Categories: I.3.7 [Computing Methodologies]: Computer Graphics—Three-Dimensional Graphics and Realism;

Keywords: Trees, canopy design, trees on GPU

1 Introduction

Being able to render trees and vegetation has been a sought-after goal in computer graphics for decades. Since trees are part of most outdoor environments, they add realism to any computer-graphics-generated outdoor scene. Technically they pose several challenges. The external structure of vegetation, especially large trees, is fairly diverse because of a large number of species. It is also affected by a wide variety of natural phenomena such as its surroundings, wind, weather, etc. Although the geometry may not be topologically complex, tree models are usually large in size due to a large number of leaves and branch structures needed to make them look realistic. This places a great strain on available computing resources when rendering many different-looking trees in a single graphics application that demands real-time interaction, like games and simulations. We envision that such an application would be able to simply “drop in” tree models of various kinds interactively in a 3D scene without affecting the speed of interaction adversely. Accordingly, this paper proposes a technique that generates models of trees using a few parameters and a few pre-cached geometric models with little to no user interaction and is particularly suitable for a GPU implementation. This makes our technique applicable to real-time applications that strive for a balance of realism and speed.

A remarkable observation about trees and plants is that their seemingly complex geometric structure can be approximately using a set of relatively simple recursive rules and algorithms. This has made algorithmic generation of tree models a very popular and successful technique in computer graphics. Grammar-based techniques like L-systems [Lindenmayer 1968] have been able to produce stunningly realistic models of many plants and small trees. From the point of view of interactive generation, algorithmic techniques suffer from a conceptual redundancy. Most of these techniques model the growth process to create the final plant/tree model instead of directly modeling the final appearance. Modeling this growth process requires computing power that makes it difficult to perform in real-time. Moreover tweaking the growth process requires skilled user interaction. Thus most tree models are generated algorithmically off-line, and are loaded into an application as static mesh, point, volume or hybrid models. Such models generated algorithmically or by other means may be quite large in size, due to which it is difficult to fit several of them in video memory simultaneously. Thus loading and rendering models of several trees of different appear-

[*]e-mail: ashesh@ilstu.edu

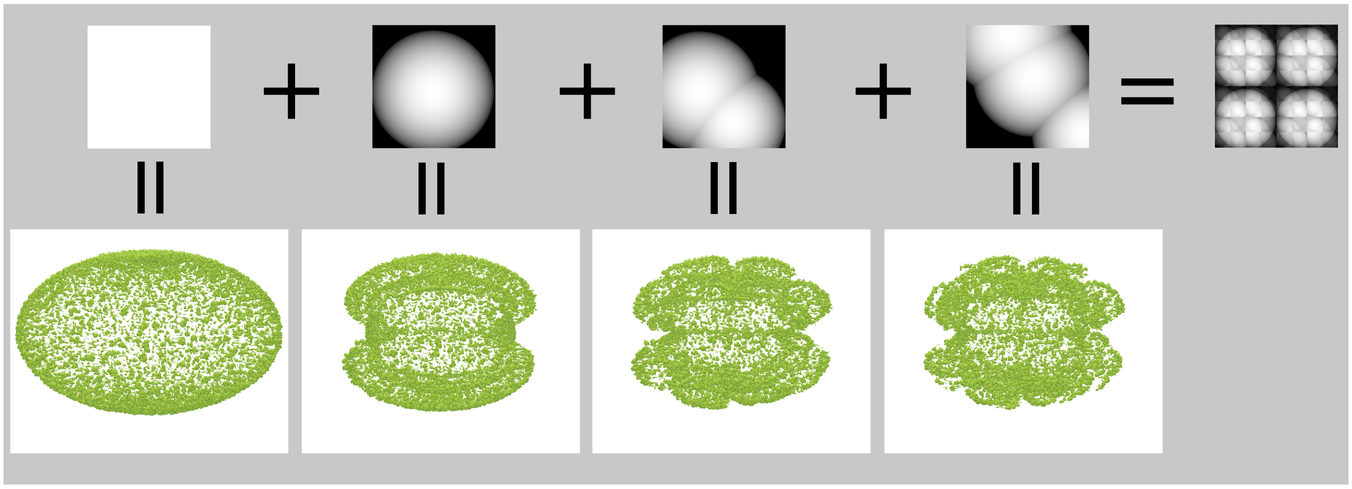


Figure 2: **Summary of our canopy generation technique.** A canopy is generated by combining several 2D functions which are visualized as images in the top row. Each 2D function modulates the radius of an ellipsoidal domain. The bottom row shows the resulting canopy. As each layer is added, the shape of the canopy changes from a perfect ellipsoid to a composite "blob" structure that resembles the canopy of a tree. The 2D functions are sampled at different frequencies and combined using Equation 1 (a persistence value of 0.7 has been used for all examples in this paper). The result of this equation can be visualized by the rightmost image in the top row.

ance at interactive rates is challenging. We take a hybrid approach to alleviate this problem. We start from a few "template" polygonal models and convert them into different tree models based on a few parameters. This conversion process is in the form of geometric transformations. Thus we are able to cache a base polygonal model and then use it to create tree models at run-time based on parameters. A different set of parameters yields a tree of a different appearance starting from the same cached model. As we do not model growth or data acquired directly from trees, our trees are not guaranteed to support all leaves through a path to the roots. However the effects of this deficiency are not visible in 3D navigation except from very few and unique camera positions (e.g. zooming in on a set of leaves from "inside" the tree, etc.). We rely on the empirical observation that such views are typically not encountered in 3D navigation. This preference of visual realism over structural soundness allows us to efficiently create and render different, visually believable tree models suitable for many applications.

Most tree-generation techniques like L-systems create a branch structure and then attach leaves, flowers and fruit models to them (the growth model of L-systems grows branch structures). However the overall shape of a matured tree is defined more by the shape of the canopy than by the branching structure. Accordingly, our approach creates a tree model by modeling the shape of the canopy, and then appending the branching structure. This allows us to directly influence the shape of the canopy and thereby, the overall appearance of the tree. Our technique is based on a hierarchical combination of functions. Since these functions are two-dimensional, they can be visualized and manipulated as images which makes tree design simple. Lastly our tree generation algorithm is amenable to a GPU implementation using a single base polygonal model kept in video memory, making it fast.

2 Related Work

Trees are an important part of any outdoor scenery, and thus they play an important role in adding realism to a computer-generated 3D scene. Therefore it is no surprise that creating and rendering trees has been a sought-after research problem in computer graphics. The problem of tree modeling has been approached from three primary directions. One of the most popular way of creating tree models is by using algorithmic models. The L-system by Lindenmeyer [Lindenmayer 1968] is based on creating trees by specifying rules of biological development. The L-system is essentially a growth model, i.e. it creates the model of a plant by simulating its growth. L-systems have been extensively used to create very realistic models of many plant species [Prusinkiewicz and Lindenmayer 1996], their biomechanics [Jirasek et al. 2000] and their interactions with the environment [Měch and Prusinkiewicz 1996]. Given the correct grammar L-systems are capable of working automatically. However in general they tend to be less user-friendly, as a skilled user with intimate understanding of the underlying grammar is required to "steer" the L-system towards an intended result. Although L-systems can even be tuned to replicate existing plants and trees [Prusinkiewicz et al. 2001] they also require significant user skill.

Images of trees can be used to render them in a 3D scene. The simplest methods are billboard-based, which show one of several perspectives of a tree depending on the view point [Ervin and Hasbrouck 2001]. Multiple photographs of a tree can be used to approximate a "visual hull" of its canopy. This approach is taken by Shlyakhtar *et al.* [2001] to derive tree models by completing the visual hull with an L-system-based branch structure. Our approach is similar to this, with two exceptions: we do not rely on existing photographs of trees and we do not create unique branching structures for each tree model, gaining some efficiency in the process. 3D scanning devices have been used to scan existing trees and creating 3D models from the resulting point clouds [Xu et al. 2007]. These methods work well to replicate existing trees and are capable of creating detailed models of trees. We contend that there are a lot of applications where such accuracy or replication are not needed. In such situations, some accuracy may be traded for benefits of efficiency in rendering and simplicity in user interaction, which are the goals of this work.

Sketch-based tree creation has gained popularity in the last few years. Chen *et al.* [2008] expand user-drawn sketches of preliminary branching structures into 3D models by comparing the projections of a database of tree models with the drawn sketch. Okabe *et al.* [2005] use rough sketches of branching structures to derive 3D models of small plants and trees. Ijiri *et al.* [2005] use hand-drawn sketches to create models of leaves and flowers in plant models. Wither *et al.* [2009] use sketches to derive models of mature trees. They address the problem by designing the tree canopy first and then deriving a branching structure to complete the tree model. This leads credence to our assumption that the appearance of trees is determined by the shape of its canopy rather than its branching structure. Instead of sketches, we rely on a more algorithmic approach to creating the tree canopy, in the form of 2D functions that can be designed off-line. Our method is conceptually different from theirs because while they target interactive design of trees, we target efficient rendering of many different-looking trees in an application with a secondary benefit of interactive design.

3 Tree Creation and Rendering

3.1 Parameterized domain

We begin by regarding every tree as a union of two geometric entities: canopy and branch structure. In contrast with growth-based systems, we treat the two as independent from each other. This gives us greater flexibility in designing each one individually. Thus, a tree can be represented mathematically as $T(\mathscr{C},\mathscr{B})$ where $\mathscr{C}$ is a function that represents the 3D geometry of the canopy, and B represents that of the branch structure. Section 3.2 discusses the design of $\mathscr{C}$ and Section 3.4 discusses the design of $\mathscr{B}$ and how the two functions are combined into a single tree model.

3.2 Creation of Tree Canopy

The canopy is fully described by the 3D positions of all its constituent leaves. The "overall" shape of a tree canopy is often easier to express. For example, it is ellipsoidal for many deciduous trees, conical for pine trees, etc. Benes *et al.* [2009] follow a similar approach by defining the overall shape of a canopy using surfaces of revolution. We mathematically express the "surface" shape of the canopy using one of two conventional domains: ellipsoidal or conical. Specifically for an ellipsoidal canopy, we express the canopy function $\mathscr{C}$ motivated above as $\mathscr{C}(r_x, r_y, r_z, \phi, \theta)$ where (r_x, r_y, r_z) are the radii and (ϕ, θ) are the polar angles. Further, we restrict the relative values of radii as $[r_x r_y r_z]^T = [k_x k_y k_z]^T f(\phi, \theta)$, where $k = \{k_x, k_y, k_z\}$ are constants and $f(\phi, \theta)$ is a 1-D scalar function for each value of (ϕ, θ). This formulation allows us to express different canopy shapes. For example, keeping $k = \{1, 1, 1\}$ creates a canopy of spherical shape, keeping $k_x, k_z < k_y$ creates an oblong canopy, and so on.

The scalar function $f(\phi, \theta)$ determines the variations of the canopy shape. Our formulation of this function is based on visual observations of many tree canopies. Tree canopies are made of "blobs", i.e. locally spherical or ellipsoidal characteristics. Trees are often sketched using blobbed outlines and such a description of their shape has been used for their modeling [Wither et al. 2009]. The surface of a tree canopy has self-similarity properties. There is an overall shape of the canopy that is modulated with large blobs that are related to the main branches directly connected to the trunk. These blobs are in turn modulated by smaller blobs, giving the canopy its overall appearance. It may be noted that this hierarchical blob structure of the canopy is intimately related to the hierarchical self-similar nature of its branching structure that is the basis of grammar-based tree modeling. Instead of expressing the branch structure hierarchically we offer the same treatment to the surface shape of the canopy by designing $f(\phi, \theta)$ suitably.

We model $f(\phi, \theta)$ as a combination of several 2D functions similar to generation of 2D Perlin noise:

$$f(\phi, \theta) = \sum_{i=0}^{n} p^i g_i(2^i \phi, 2^i \theta) \tag{1}$$

Thus we start from several 2D functions $(g_i(.))$ whose domain is governed by (ϕ, θ) as the parameters for the canopy generation process, and then combine them to create the function that modulates the radii of the canopy in the ellipsoidal domain. Each 2D function is sampled at a certain frequency: specifically $g_i(.)$ is sampled at twice the frequency as $g_{i-1}(.)$. The term p is a scalar and is referred to as persistence in fractal literature. It controls the contributions of the various frequencies towards f. A value of p between 0 and 1 decreases the contributions of higher frequencies. The function $g_0(.)$ is the "base" layer and is a constant function. The remaining functions $\{g_i(.), i > 0\}$ are in general noise functions. In the case of trees, they represents blobs of differing extents (see Figure 2). As we choose a persistence value in the range $(0 \leq p \leq 1)$, $g_0(.)$ defines the overall shape of the canopy in the chosen canopy domain, while the remaining functions $\{g_i(.), i > 0\}$ define modulations of decreasing amplitudes over the basic shape. Visually the process may be imagined as starting from a perfectly ellipsoidal canopy and then modulating the surface bumpiness progressively by providing additional 2D functions (see Figure 2). In practice we also perturb the values of ϕ and θ by a small amount using a white noise function before sampling the functions to increase randomness.

Finally, we attach one of several leaf structures to each leaf position generated by $\mathscr{C}$ using the above technique. Every leaf is modeled as a simple quad with an alpha-texture created from photographs of actual leaves. The orientation of the quad is computed randomly. Alternatively one can specify the position of the sun to modulate this random assignment so that all leaves face the approximate direction of the sun.

3.3 GPU Implementation of Tree Canopy Generation

The above formulation leads to an efficient GPU implementation. The functions $g_i(.)$ can be modeled using 2D images parameterized by the angles θ and ϕ that are designed off-line (they are illustrated in the top row of Figure 2). Since they represent a 1D scalar function at every pixel, four functions can be packed in a single texture. Moreover if Equation 1 is pre-computed for a given set of functions, the canopy can be generated using only one texture (see rightmost figure in the top row of Figure 2). Every leaf of the tree canopy starts as a canonical quad of unit size centered at the origin. Each vertex of this quad gets two sets of texture coordinates. The first texture coordinate is the corresponding value of (ϕ, θ) for the leaf, and is common to all vertices of the quad. The second texture coordinate is used to paste the leaf texture on the quad. In a vertex shader, the 2D textures representing $g_i(.)$ are used to calculate the function $f(\phi, \theta)$ using Equation 1 to determine the 3D position of the sample (i.e. leaf location). This position and the normal direction are used to transform vertices of the quad and finally it is texture-mapped using the leaf texture. Thus the "canonical" tree canopy (consisting of numerous leaf quads centered at the origin with randomized normals) can be pre-generated and cached using a display list to avoid being sent to GPU memory for every tree.

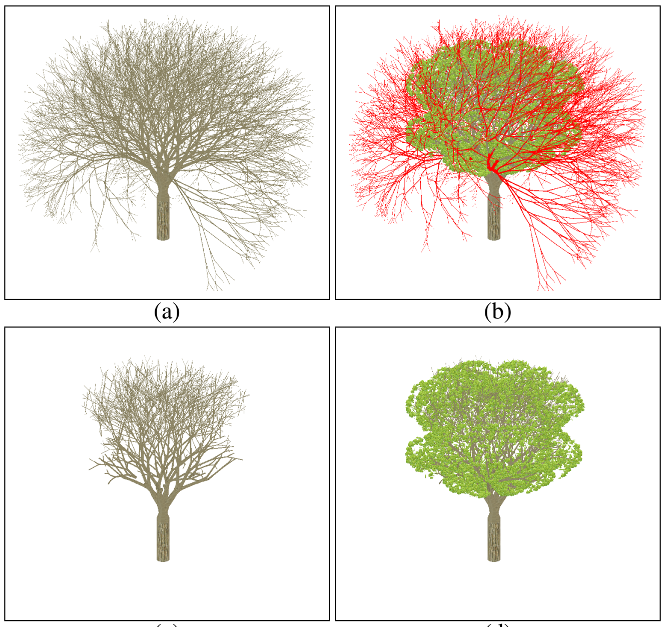


Figure 3: **Creation of tree branches.** (a) the original branch structure. This example was constructed using an L-system. Such canonical branch structures are clipped to fit a generated canopy. (b) illustration of the clipping process. The branches shown in red are clipped. This process is done at runtime in GPU shaders. (c) the clipped branches from the model in (a) for a particular canopy. (d) the final tree model with the canopy and the clipped branches.

3.3.1 Designing 2D functions

Images for the 2D functions $g_i(.)$ can be designed using any image program. Since $g_i(.)$ modulate the radius of an ellipsoidal domain, ellipsoidal "blobs" can be produced by creating constant-color rectangles in the corresponding images (a rectangle in g_i increases the radii of points in a particular range of ϕ and θ by a constant amount). If constant-color ellipses are drawn in the images, they produce *involute* blobs on the tree canopy (a circle would produce a 3D Archimedes' spiral, illustrated in the second column of Figure 2) which we have found by visual observation to produce better-looking tree canopies. Functions sampled at higher frequencies can also be white noise textures, adding greater randomness to the canopy. Instead of constant-color shapes, smoothly varying colors can also be used to get greater and gradual variation in the contours of the tree canopy (the examples in Figure 2 were generated by centering gaussians at random positions in the image). In general gradual variations can be produced by ensuring that there are no sharp edges in the images.

3.4 Creation of Tree Branches

We now complete the tree model by appending the branch structure to the canopy. In actual trees branches serve as pathways for moisture and nutrients from the roots of the tree to every leaf, flower and fruit. Thus every leaf is connected via a single path to the root. Furthermore there are allometric constraints on the branching angles, branch thickness, etc. In order for an artificially generated tree model to be perfectly realistic (structurally), it must obey all these conditions. However if trees are viewed from a distance as they often are, all these structural properties are not readily visible. In fact the thickness of the canopy hides many of the tree's structural aspects, especially those in its crown. Similarly, if a tree model is not viewed at very specific, close angles in a graphics application, the burden of achieving and maintaining a structurally accurate model is lessened. In other words, a branch model that looks realistic enough from reasonable camera distances and angles suffices for many applications.

This assumption allows us to treat branch generation as independent from canopy generation. This separation, although theoretically flawed and even unnecessary, provides distinct implementation benefits. First, we can leverage the vast amount of prior work and literature devoted to generating realistic-looking trees using a variety of methods. As a proof of concept, we use a simple L-system to create branch structures. This method itself is capable of generating very diverse branch structures using grammar and can be constrained with conditions based in allometry. All the examples of branches in this paper have been created using a simple L-system that obeys Murray's rule [Murray 1927]. Secondly, instead of storing many branching structures and selecting the appropriate one for a given canopy, we store only a few "template" branching structures. Then we fit one of them to a given canopy during run-time to create a complete tree model. Thus different tree canopies that are generated using different input images can be used to render trees in the same scene using one or a few branching structures. Figure 3 illustrates the process. We select a pre-generated branching structure that "outgrows" a generated canopy. Then, we clip it against the surface of the canopy to create a branching structure that fits within it. The clipped branching structure is then rendered with the canopy to create the final tree model. In order to complete this process in real-time, we propose a fast but approximate clipping procedure on the GPU. Although this clipping process creates a model that represents a "pruned" branching structure instead of a naturally shaped one, it satisfies our goal of creating an overall visually realistic tree model. If more structural realism is desired, a separate algorithm may be employed to evolve a custom branching structure for each generated canopy similar to Shlyakhtar *et al.* [2001] or Benes *et al.* [2009].

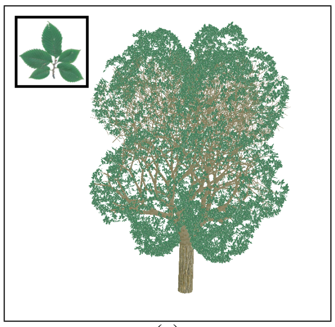
3.5 GPU Implementation of Tree Branches

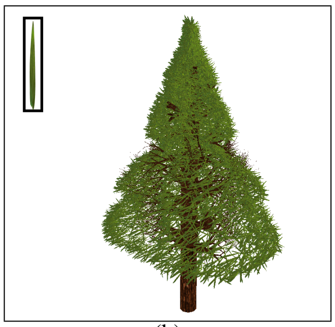
The template branch structures are modeled using a series of points and lines and therefore easily fit in memory. At run-time cylinders are rendered for every branch. The length and orientation of every branch is associated with every vertex of its corresponding cylinder. In a vertex shader, each cylinder is transformed to align with the corresponding branch. The shader computes the polar angles ϕ and θ for each end point of the branch. Then it re-computes the leaf location for these angles in the canopy. If the cylinder extends beyond the leaf location, it is made fully transparent. It may be noted that this is an approximation of true surface-based clipping. Ideally one must determine where the branch would intersect the canopy surface to clip it. However our approximation produces visually correct results in most cases. A threshold for clipping can be changed to manually remove protruding branches. This simple technique results in a visual clipping of protruding branches. A corresponding pixel shader then applies a bark texture to the branch to give it a realistic appearance.

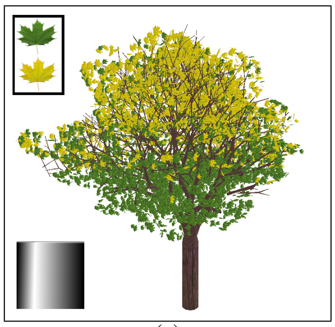
Thus, the overall technique can be summarized as follows:

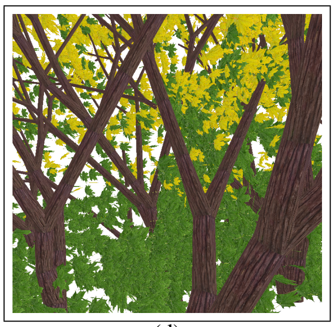
1. Design overall shape of the tree canopy by creating image for g_0 and choosing a domain.
2. Design images for other g_i.
3. During rendering:

[1]http://en.wikipedia.org/wiki/File:Maple_leaf_Fcb981.JPG


(a)


(b)


(c)


(d)

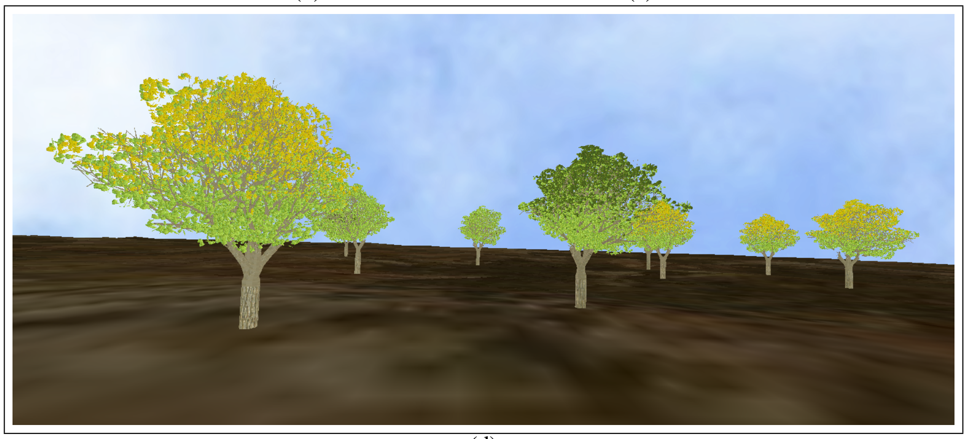

(d)

Figure 4: **More results.** Insets in (a)-(c) show the leaf texture used for each image. (a) A tree with small leaves (leaf texture taken from Xu *et al.*[2007]). This tree uses a transformed version of the same branching structure as that in Figure 1(c). (b) A conical pine tree. This tree was generated by parameterizing the tree in the conical domain. A different branch structure was created by the same L-system and used for this example. (c) A maple tree (leaf texture taken from WikiMedia Commons[1] and the color was altered). Instead of using a constant base layer function (as shown in Figure 2, top left) the function illustrated in the lower inset was used. This layer biases the overall canopy on one side. Maple leaves of two colors were used to illustrate the effect of fall colors. (d) A zoomed view of the same maple tree. (e) A scene with multiple trees and a terrain generated using a noise texture (terrain texture taken from http://www.spiralgraphics.biz). This scene contains 16 trees and is rendered at 20 fps.

(a) Determine position of every leaf using the g_i functions and the appropriate ellipsoidal/conical domain.

(b) For every branch position, determine whether it protrudes the surface of the canopy. If so, clip it by making it transparent.

(c) Apply textures to leaf and branch quads to render final tree model.

4 Results

All results shown in this paper have been created on a Dell XPS M1530 computer equipped with an NVidia GeForce 8600M GT graphics card. The application is written in C++ and OpenGL and uses GLSL for its GPU shaders. Each tree shown in this paper has 10000 leaves and a total of 146500 quads.

Figures 1 and 4 show some results as a proof of concept. Figure 4(a) was rendered using a branch of 5 leaves at every leaf location in the canopy (see accompanying inset). Figure 4(b) shows how our technique can be used to create conical tree models as well. This result was generated by simply parameterizing the canopy in the conical domain instead of an ellipsoidal domain, and using a different branching structure. Figure 4(c) shows a maple tree. This result illustrates the effect of a customized base layer function (shown in the lower inset). The vertical white line in the function corresponds a longitudinal slice in the ellipsoidal domain. Thus the function has the effect of increasing the canopy girth at a particular longitude and smoothly spreading it over the entire ellipsoidal domain, visually making the canopy “lopsided” and denser on one side. Two different leaf textures were used to simulate the effect of fall. The fragment shader chooses the appropriate texture for a particular leaf location based on its coordinate (ϕ, θ) in the ellipsoidal domain and a random noise function. Figure 4(d) shows a zoomed view of this maple tree. Although our method can potentially create unsupported leaves, they are not apparent even when zoomed into the tree. Thus our method works well even for many view points inside trees, thereby making its compromised structure less visually relevant (Figure 1(d) provides another example). Figure 4(e) shows a scene with 16 tree models and a textured terrain created by displacement mapping. Preliminary performance results are tabulated in Table 1. These performance results represent an “unoptimized” implementation of our algorithm. Specifically, Equation 1 is implemented in a vertex shader instead of using a single image as shown in Figure 2 and inbuilt trigonometric functions in GLSL are used instead of relying on faster lookup tables.

# quads/tree	# leaves/tree	# trees rendered	Frame rate (fps)
146500	10000	5	55
		10	28
142900	6400	5	55
		10	30
139000	2500	5	60
		10	34

Table 1: **Performance**. Performance tests were run on GLSL shaders. The function $f(\phi, \theta)$ was calculated using Equation 1 in the vertex shader.

5 Future Work

The idea of using functions as parameters to draw trees can be extended in several ways. More intuitive interaction techniques that are more directly related to tree generation can be devised to design the various 2D functions. Sketch-based design is a good alternative that we wish to try in the future, where a 2D sketch would modulate one or more of these 2D functions to create a tree model that obeys the sketched hints.

Secondly since our technique is independent of how the branch structure is created, branches structures obtained from other sources like 3D scans can be incorporated into our framework. In addition, it is possible to use the self-similarity of the generated canopy to create its corresponding branch structure. A challenge in this process is the mismatch between the number of functions that create the canopy and the number of branching levels: typically only 3-4 functions are required to create a canopy, while a branching structure has many more levels. Lastly since the branches are generated independently, they may be modified to match the structure and shape of the canopy more closely by taking a growth-based approach, similar to Streit *et al.* [2005].

References

BENES, B., ANDRYSCO, N., AND STAVA, O. 2009. Interactive modeling of virtual ecosystems. In *Proc. Eurographics Workshop on Natural Phenomena*, 9–16.

CHEN, X., NEUBERT, B., XU, Y.-Q., DEUSSEN, O., AND KANG, S. B. 2008. Sketch-based tree modeling using markov random field. *ACM Trans. Graph. 27*, 5, 1–9.

ERVIN, S., AND HASBROUCK, H. 2001. *Landscape Modeling: Digital Techniques for Landscape Visualization*. McGraw-Hill Professional Publishing.

IJIRI, T., OWADA, S., OKABE, M., AND IGARASHI, T. 2005. Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. In *Proc. SIGGRAPH*, 720–726.

JIRASEK, C., PRUSINKIEWICZ, P., AND MOULIA, B. 2000. Integrating biomechanics into developmental plant models expressed using l-systems. In *Proc. Plant Biomechanics*, 615–624.

LINDENMAYER, A. 1968. Mathematical models for cellular interaction in development: Parts i and ii. *Journal of Theoretical Biology 18*, 280–315.

MURRAY, C. 1927. A relationship between circumference and weight in trees and its bearing on branching angles. *J. Gene. Physiology 10*, 725–729.

MĚCH, R., AND PRUSINKIEWICZ, P. 1996. Visual models of plants interacting with their environment. In *Proc. SIGGRAPH*, 397–410.

OKABE, M., OWADA, S., AND IGARASHI, T. 2005. Interactive design of botanical trees using freehand sketches and example-based editing. In *Proc. Eurographics*, 487–496.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1996. *The algorithmic beauty of plants*.

PRUSINKIEWICZ, P., MÜNDERMANN, L., KARWOWSKI, R., AND LANE, B. 2001. The use of positional information in the modeling of plants. In *Proc. SIGGRAPH*, 289–300.

SHLYAKHTER, I., ROZENOER, M., DORSEY, J., AND TELLER, S. 2001. Reconstructing 3d tree models from instrumented photographs. *IEEE Comput. Graph. Appl. 21*, 3, 53–61.

STREIT, L., FEDERL, P., SOUSA, M. C., STREIT, L., FEDERL, P., AND SOUSA, M. C. 2005. Modelling plant variation through growth. In *Proc. Eurographics*, 497–506.

WITHER, J., BOUDON, F., CANI, M.-P., AND GODIN, C. 2009. Structure from silhouettes: a new paradigm for fast sketch-based design of trees. In *Proc. Eurographics*, 541–550.

XU, H., GOSSETT, N., AND CHEN, B. 2007. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Trans. Graph. 26*, 4, 19.