First publ. in: SIGGRAPH '98 conference proceeding, Orlando, FL, USA, July 1998, S. 275-286

Realistic modeling and rendering of plant ecosystems

Oliver Deussen[1] Pat Hanrahan[2] Bernd Lintermann[3] Radomír Měch[4]
Matt Pharr[2] Przemyslaw Prusinkiewicz[4]

[1] Otto-von-Guericke University of Magdeburg
[2] Stanford University
[3] The ZKM Center for Art and Media Karlsruhe
[4] The University of Calgary*

Abstract

Modeling and rendering of natural scenes with thousands of plants poses a number of problems. The terrain must be modeled and plants must be distributed throughout it in a realistic manner, reflecting the interactions of plants with each other and with their environment. Geometric models of individual plants, consistent with their positions within the ecosystem, must be synthesized to populate the scene. The scene, which may consist of billions of primitives, must be rendered efficiently while incorporating the subtleties of lighting in a natural environment.

We have developed a system built around a pipeline of tools that address these tasks. The terrain is designed using an interactive graphical editor. Plant distribution is determined by hand (as one would do when designing a garden), by ecosystem simulation, or by a combination of both techniques. Given parametrized procedural models of individual plants, the geometric complexity of the scene is reduced by *approximate instancing*, in which similar plants, groups of plants, or plant organs are replaced by instances of representative objects before the scene is rendered. The paper includes examples of visually rich scenes synthesized using the system.

CR categories: I.3.7 **[Computer Graphics]**: Three-Dimensional Graphics and Realism, I.6.3 **[Simulation and Modeling]**: Applications, J.3 **[Life and Medical Sciences]**: Biology.

Keywords: realistic image synthesis, modeling of natural phenomena, ecosystem simulation, self-thinning, plant model, vector quantization, approximate instancing.

1 INTRODUCTION

Synthesis of realistic images of terrains covered with vegetation is a challenging and important problem in computer graphics. The challenge stems from the visual complexity and diversity of the modeled scenes. They include natural ecosystems such as forests or grasslands, human-made environments, for instance parks and gardens, and intermediate environments, such as lands recolonized by vegetation after forest fires or logging. Models of these ecosystems have a wide range of existing and potential applications, including computer-assisted landscape and garden design, prediction and visualization of the effects of logging on the landscape, visualization of models of ecosystems for research and educational purposes, and synthesis of scenes for computer animations, drive and flight simulators, games, and computer art.

Beautiful images of forests and meadows were created as early as 1985 by Reeves and Blau [50] and featured in the computer animation *The Adventures of André and Wally B.* [34]. Reeves and Blau organized scene modeling as a sequence of steps: specification of a terrain map that provides the elevation of points in the scene, interactive or procedural placement of vegetation in this terrain, modeling of individual plants (grass and trees), and rendering of the models. This general scheme was recently followed by Chiba *et al.* [8] in their work on forest rendering, and provides the basis for commercial programs devoted to the synthesis of landscapes [2, 49].

The complexity of nature makes it necessary to carefully allot computing resources — CPU time, memory, and disk space — when recreating natural scenes with computer graphics. The size of the database representing a scene during the rendering is a particularly critical item, since the amount of geometric data needed to represent a detailed outdoor scene is more than can be represented on modern computers. Consequently, a survey of previous approaches to the synthesis of natural scenes reflects the quest for a good tradeoff between the realism of the images and the amount of resources needed to generate them.

The scenes synthesized by Reeves and Blau were obtained using (structured) particle systems, with the order of one million particles per tree [50]. To handle large numbers of primitive elements contributing to the scene, the particle models of individual trees were generated procedurally and rendered sequentially, each model discarded as soon as a tree has been rendered. Consequently, the size of memory needed to generate the scene was proportional to the number of particles in a single tree, rather than the total number of particles in the scene. This approach required approximate shading calculations, since the detailed information about the neighborhood trees was not available. Approximate shading also reduced the time needed to render the scenes.

Another approach to controlling the size of scene representation is the reduction of visually unimportant detail. General methods for achieving this reduction have been the subject of intense research (for a recent example and further references see [24]), but the results do not easily apply to highly branching plant structures. Consequently, Weber and Penn [63] introduced a heuristic multiresolution representation specific to trees, which allows for reducing

* Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4 (*pwp@cpsc.ucalgary.ca*)

the number of geometric primitives in the models that occupy only a small portion on the screen. A multiresolution representation of botanical scenes was also explored by Marshall *et al.* [35], who integrated polygonal representations of larger objects with tetrahedral approximations of the less relevant parts of a scene.

A different strategy for creating visually complex natural scenes was proposed by Gardner [17]. In this case, the terrain and the trees were modeled using a relatively small number of geometric primitives (quadric surfaces). Their perceived complexity resulted from procedural textures controlling the color and the transparency of tree crowns. In a related approach, trees and other plants were represented as texture-mapped flat polygons (for example, see [49]). This approach produced visible artifacts when the position of the viewpoint was changed. A more accurate image-based representation was introduced by Max [37], who developed an algorithm for interpolating between precomputed views of trees. A multiresolution extension of this method, taking advantage of the hierarchical structure of the modeled trees, was presented in [36]. Shade *et al.* described a hybrid system for walkthroughs that uses a combination of geometry and textured polygons [53].

Kajiya and Kay [26] introduced volumetric textures as an alternative paradigm for overcoming the limitations of texture-mapped polygons. A method for generating terrains with volumetric textures representing grass and trees was proposed by Neyret [40, 41]. Chiba *et al.* [8] removed the deformations of plants caused by curvatures of the underlying terrain by allowing texels to intersect.

The use of volumetric textures limits the memory or disk space needed to represent a scene, because the same texel can be re-applied to multiple areas. The same idea underlies the oldest approach to harnessing visually complex scenes, object instancing [59]. According to the paradigm of instancing, an object that appears several times in a scene (possibly resized or deformed by an affine transformation) is defined only once, and its different occurrences (instances) are specified by affine transformations of the prototype. Since the space needed to represent the transformations is small, the space needed to represent an entire scene depends primarily on the number and complexity of *different* objects, rather than the number of their instances. Plants are particularly appealing objects of instancing, because repetitive occurrences can be found not only at the level of plant species, but also at the level of plant organs and branching structures. This leads to compact hierarchical data structures conducive to fast ray tracing, as discussed by Kay and Kajiya [27], and Snyder and Barr [56]. Hart and DeFanti [20, 21] further extended the paradigm of instancing from hierarchical to recursive (self-similar) structures. All the above papers contain examples of botanical scenes generated using instancing.

The complexity of natural scenes makes them not only difficult to render, but also to specify. Interactive modeling techniques, available in commercial packages such as Alias/Wavefront Studio 8 [1], focus on the direct manipulation of a relatively small number of surfaces. In contrast, a landscape with plants may include many millions of individual surfaces — representing stems, leaves, flowers, and fruits — arranged into complex branching structures, and further organized in an ecosystem. In order to model and render such scenes, we employ the techniques summarized below.

Multilevel modeling and rendering pipeline. Following the approach initiated by Reeves and Blau [50], we decompose the process of image synthesis into stages: modeling of the terrain, specification of plant distribution, modeling of the individual plants, and rendering of the entire scene. Each of these stages operates at a different level of abstraction, and provides a relatively high-level input for the next stage. Thus, the modeler is not concerned with plant distribution when specifying the terrain, and plant distribution is determined (interactively or algorithmically) without considering details of the individual plants. This is reminiscent of the simulations of flocks of birds [51], models of flower heads with phyllotactic patterns [16], and models of organic structures based on morphogenetic processes [14], where simulations were performed using geometrically simpler objects than those used for visualization. Blumberg and Galyean extended this paradigm to *multi-level* direction of autonomous animated agents [5]. In an analogous way, we apply it to multi-level modeling.

Open system architecture. By clearly specifying the formats of inputs and outputs for each stage of the pipeline, we provide a framework for incorporating independently developed modules into our system. This open architecture makes it possible to augment the complexity of the modeled scenes by increasing the range of the available modules, and facilitates experimentation with various approaches and algorithms.

Procedural models. As observed by Smith [55], procedural models are often characterized by a significant *data-base amplification*, which means that they can generate complex geometric structures from small input data sets. We benefit from this phenomenon by employing procedural models in all stages of the modeling pipeline.

Approximate instancing. We use object instancing as the primary paradigm for reducing the size of the geometric representation of the rendered scenes. To increase the degree of instancing, we cluster scene components (plants and their parts) in their parameter spaces, and approximate all objects within a given cluster with instances of a single representative object. This idea was initially investigated by Brownbill [7]; we extend it further by applying vector quantization (*c.f.* [18]) to find the representative objects in multidimensional parameter spaces.

Efficient rendering. We use memory- and time-efficient rendering techniques: decomposition of the scenes into subscenes that are later composited [12], ray tracing with instancing and a support for rendering many polygons [56], and memory-coherent ray tracing [43] with instancing.

By employing these techniques, we have generated scenes with up to 100,000 detailed plant models. This number could be increased even further, since none of the scenes required more than 150MB to store. However, with 100,000 plants, each plant is visible on average only in 10 pixels of a 1K $\times$ 1K image. Consequently, we seem to have reached the limits of useful scene complexity, because the level of visible detail is curbed by the size and resolution of the output device.

2 SYSTEM ARCHITECTURE

The considerations presented in the previous section led us to the modular design of our modeling and rendering system `EcoSys`, shown schematically in Figure 1.

The modeling process begins with the specification of a terrain. For this purpose, we developed an interactive editor `TerEdit`, which integrates a number of terrain modeling techniques (Section 3). Its output, a *terrain data* file, includes the altitudes of a grid of points superimposed on the terrain, normal vectors indicating the local slope of the terrain, and optional information describing environmental conditions, such as soil humidity.

The next task is to determine plant distribution on a given terrain. We developed two techniques for this purpose: visual editing of plant densities and simulation of plant interactions within an ecosystem

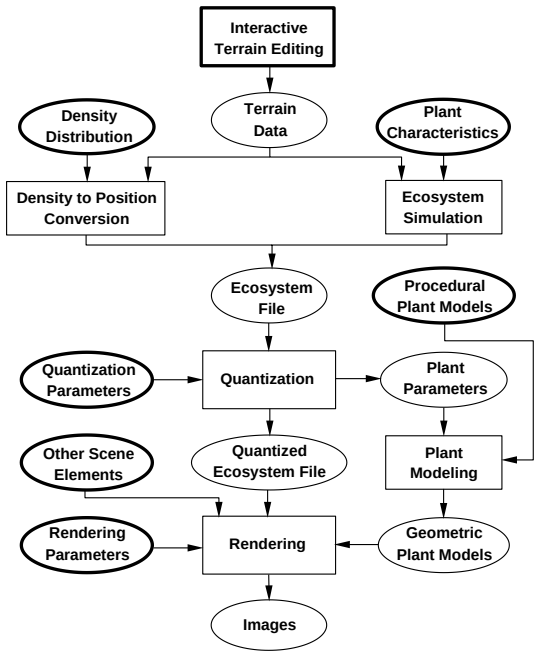

Figure 1: Architecture of the scene synthesis system. Bold frames indicate interactive programs and input files specified by the user.

(Section 4). The editing approach is particularly well suited to model environments designed by people, for example orchards, gardens, or parks. The user specifies the distribution of plant densities using a paint program. To convert this information into positions of individual plants, we developed the program `densedis` based on a half-toning algorithm: each dot becomes a plant. We can also specify positions of individual plants explicitly; this is particularly important in the synthesis of scenes that include detailed views of individual plants in the foreground.

To define plant distribution in natural environments, such as forests or meadows, we apply an ecosystem simulation model. Its input consists of terrain data, ecological characteristics of plant species (for example, annual or perennial growth and reproduction cycle, preference for wet or dry soil, and shade tolerance) and, optionally, the initial distribution of plants. The growth of plants is simulated accounting for competition for space, sunlight, resources in the soil, aging and death, seed distribution patterns, *etc*. We perform these simulations using the L-system-based plant modeling program `cpfg` [47], extended with capabilities for simulating interactions between plants and their environments [39]. To allow for simulations involving thousands of plants, we use their simplified geometrical representations, which are subsequently replaced by detailed plant models for visualization purposes.

Specification of a plant distribution may involve a combination of interactive and simulation techniques. For example, a model of an orchard may consist of trees with explicitly specified positions and weeds with positions determined by a simulation. Conversely, the designer of a scene may wish to change its aspects after an ecological simulation for aesthetic reasons. To allow for these operations, both `densedis` and `cpfg` can take a given plant distribution as input for further processing.

Plant distribution, whether determined interactively or by ecosystem simulation, is represented in an *ecosystem* file. It contains the information about the type, position and orientation of each plant (which is needed to assemble the final scene), and parameters of individual plants (which are needed to synthesize their geometric models).

Since we wish to render scenes that may include thousands of plants, each possibly with many thousands of polygons, the creation and storage of a separate geometric plant model for each plant listed in the ecosystem file is not practical. Consequently, we developed a program `quantv` that clusters plants in their parameter space and determines a representative plant for each cluster (Section 6). The algorithm performs quantization adaptively, thus the result depends on the characteristics of plants in the ecosystem. The quantization process produces two outputs: a *plant parameter* file, needed to create geometric models of the representative plants, and a *quantized ecosystem* file, which specifies positions and orientations of the instances of representative plants throughout the scene.

We employ two modeling programs to create the representative plants: the interactive plant modeler `xfrog` [10, 32, 33] and the L-system-based simulator `cpfg` [39, 47]. These programs input parametrized *procedural plant models* and generate specific *geometric plant models* according to the values in the plant parameter file (Section 5). To reduce the amount of geometric data, we extended the concept of instancing and quantization to components of plants. Thus, if a particular plant or group of plants has several parts (such as branches, leaves, or flowers) that are similar in their respective parameter spaces, we replace all occurrences of these parts with instances of a representative part.

Finally, the ecosystem is rendered. The input for rendering consists of the quantized ecosystem file and the representative plant models. Additional information may include geometry of the terrain and human-made objects, such as houses or fences. In spite of the quantization and instancing, the resulting scene descriptions may still be large. We experimented with three renderers to handle this complexity (Section 7). One renderer, called `fshade`, decomposes the scene into sub-scenes that are rendered individually and composited to form final images. Unfortunately, separating the scene into sub-scenes makes it impossible to properly capture global illumination effects. To alleviate this problem, we use the ray-tracer `rayshade` [29], which offers support for instancing and time-efficient rendering of scenes with many polygons, as long as the scene description fits in memory. When the scene description exceeds the available memory, we employ the memory-efficient ray-tracer `toro` [43], extended with a support for instancing.

In the following sections we describe the components of the `EcoSys` modeling pipeline in more detail. In Section 8, we present examples that illustrate the operation of the system as a whole.

3 TERRAIN SPECIFICATION

We begin the modeling of a scene with the specification of a terrain. The goal of this step is to determine elevation data, local orientations of the terrain, and additional characteristics, such as the water content in the soil, which may affect the type and vigor of plants at different locations.

Terrain data may have several sources. Representations of real terrains are available, for example, from the U.S. Geological Survey [30]. Several techniques have also been developed for creating synthetic terrains. They include: hand-painted height maps [65], methods for generating fractal terrains (reviewed in [38]), and models based on the simulation of soil erosion [28, 38].

In order to provide detailed control over the modeled terrain while taking advantage of the data amplification of fractal methods [55],

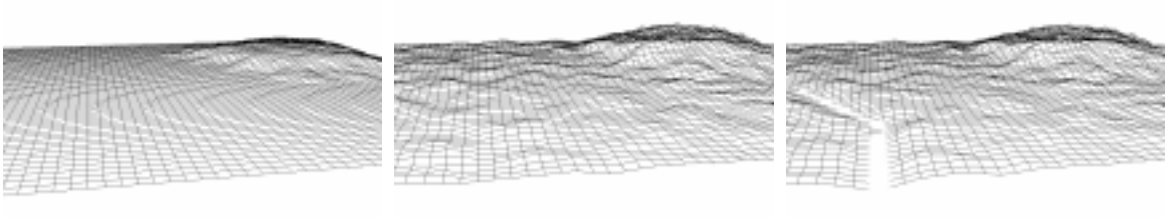

Figure 2: Three stages in creating a terrain: after loading a height map painted by hand (left), with hills added using noise synthesis (middle), and with a stream cut using the stream mask (right).

we designed and implemented an interactive terrain editing system `TerEdit`, which combines various techniques in a procedural manner. Terrain editing consists of *operations*, which modify the terrain geometry and have the spatial scope limited by *masks*. A similar paradigm is used in Adobe Photoshop [9], where *selections* can be used to choose an arbitrary subset of an image to edit.

We assume that masks have values between zero and one, allowing for smooth blending of the effects of operations. Both masks and operations can depend on the horizontal coordinates and the altitude of the points computed so far. Thus, it is possible to have masks that select terrain above some altitude or operations that are functions of the current altitude. The user's editing actions create a pipeline of operations with associated masks; to compute the terrain altitude at a point, the stages of this pipeline are evaluated in order. Undo and redo operations are easily supported by removing and re-adding operations from the pipeline and re-evaluating the terrain.

Examples of editing operations include translation, scaling, nonlinear scaling, and algorithmic synthesis of the terrain. The synthesis algorithm is based on noise synthesis [38], which generates realistic terrains by adding multiple scales of Perlin's noise function [42]. The user can adjust a small number of parameters that control the overall roughness of the terrain, the rate of change in roughness across the surface of the terrain, and the frequency of the noise functions used. Noise synthesis allows terrain to be easily evaluated at a single point, without considering the neighboring points; this makes it possible to have operations that act locally. Another advantage of noise synthesis is efficiency of evaluation; updating the wireframe terrain view (based on 256×256 samples of the region of interest) after applying an operation typically takes under a second. On a multiprocessor machine, where terrain evaluation is multi-threaded, the update time is not noticeable.

The editor provides a variety of masks, including ones that select rectangular regions of terrain from a top view, masks that select regions based on their altitude, and masks defined by image files. One of the most useful masks is designed for cutting streams through terrain. The user draws a set of connected line segments over the terrain, and the influence of the mask is based on the minimum distance from a sample point to any of these segments. A spline is applied to smoothly increase the influence of the mask close to the segments. When used with a scaling operation, the terrain inside and near the stream is scaled towards the water level, and nearby terrain is ramped down, while the rest of the terrain is unchanged.

The specification of a terrain using `TerEdit` is illustrated in Figure 2. In the first snapshot, the hill in the far corner was defined by loading in a height map that had been painted by hand. Next, small hills were added to the entire terrain using noise synthesis. The last image shows the final terrain, after the stream mask was used to cut the path of a stream. A total of five operators were applied to make this terrain, and the total time to model it was approximately fifteen minutes.

Once the elevations have been created, additional parameters of the terrain can be computed as input for ecosystem simulations or a direct source of parameters for plant models. Although the user can interactively paint parameters on the terrain, simulation provides a more sound basis for the modeling of natural ecosystems. Consequently, `TerEdit` incorporates a simulator of rain water flow and distribution in the soil, related to both the erosion algorithm by Musgrave *et al.* [38] and the particle system simulation of water on building facades by Dorsey *et al.* [11]. Water is dropped onto the terrain from above; some is absorbed immediately while the rest flows downhill and is absorbed by the soil that it passes through. A sample terrain with the water distribution generated using this approach is shown in Figure 3.

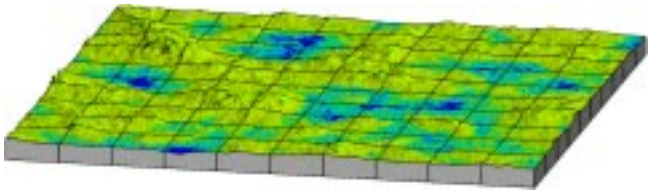

Figure 3: A sample terrain with the water concentration ranging from high (blue) to low (yellow)

4 SPECIFICATION OF PLANT POPULATIONS

The task of populating a terrain with plants can be addressed using methods that offer different tradeoffs between the degree of user control, time needed to specify plant distribution, and biological validity of the results. The underlying techniques can be divided into *space-occupancy* or *individual-based* techniques. This classification is related to paradigms of spatially-explicit modeling in ecology [3, 19], and parallels the distinction between space-based and structure-based models of morphogenesis [44].

The space-occupancy techniques describe the distribution of the *densities* of given plant species over a terrain. In the image synthesis context, this distribution can be be obtained using two approaches:

Explicit specification. The distribution of plant densities is measured in the field (by counting plants that occupy sample plots) or created interactively, for example using a paint program.

Procedural generation. The distributions of plant densities is obtained by simulating interactions between plant populations using an ecological model. The models described in the literature are commonly expressed in terms of *cellular automata* [19] or *reaction-diffusion* processes [23].

The individual-based techniques provide the location and attributes of *individual plants*. Again, we distinguish two approaches:

Explicit specification. Plant positions and attributes represent field data, for example obtained by surveying a real forest [25], or specified interactively by the user.

Procedural generation. Plant positions and attributes are obtained using a *point pattern generation model*, which creates a distribution of points with desired statistical properties [66], or using an *individual-based population model* [13, 58], which is applied to simulate interactions between plants within an ecosystem.

Below we describe two methods for specifying plant distribution that we have developed and implemented as components of `EcoSys`. The first method combines interactive editing of plant densities with a point pattern generation of the distribution of individual plants. The second method employs individual-based ecological simulations.

4.1 Interactive specification of plant populations

To specify a plant population in a terrain, the user creates a set of gray-level images with a standard paint program. These images define the spatial distributions of plant densities and of plant characteristics such as the age and vigor.

Given an image that specifies the distribution of densities of a plant species, positions of individual plants are generated using a halftoning algorithm. We have used the Floyd-Steinberg algorithm [15] for this purpose. Each black pixel describes the position of a plant in the raster representing the terrain. We also have implemented a relaxation method that iteratively moves each plant position towards the center of mass of its Voronoi polygon [6]. This reduces the variance of distances between neighboring plants, which sometimes produces visually pleasing effects.

Once the position of a plant has been determined, its parameters are obtained by referring to the values of appropriate raster images at the same point. These values may control the plant model directly or provide arguments to user-specified mathematical expressions, which in turn control the models. This provides the user with an additional means for flexibly manipulating the plant population.

Operations on raster images make it possible to capture some interactions between plants. For example, if the radius of a tree crown is known, the image representing the projection of the crown on the ground may be combined with user-specified raster images to decrease the density or vigor of plants growing underneath.

4.2 Simulation of ecosystems

Individual-based models of plant ecosystems operate at various levels of abstraction, depending on the accuracy of the representation of individual plants [58]. Since our goal is to simulate complex scenes with thousands of plants, we follow the approach of Firbank and Watkinson [13], and represent plants coarsely as circles positioned in a continuous landscape. Each circle defines the *ecological neighborhood* of the plant in its center, that is the area within which the plant interacts with it neighbors. Biologically motivated rules govern the outcomes of interactions between the intersecting circles. Global behavior of the ecosystem model is an emergent property of a system of many circles.

We implemented the individual-based ecosystem models using the framework of *open L-systems* [39]. Since L-systems operate on branching structures, we represent each plant as a circle located at the end of an invisible positioning line. All lines are connected into a branching structure that spreads over the terrain.

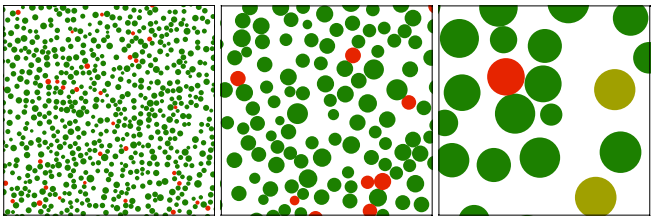

Figure 4: Steps 99, 134, and 164 of a sample simulation of the self-thinning process. Colors represents states of the plants: not dominated (green), dominated (red), and old (yellow). The simulation began with 62,500 plants, placed at random in a square field. Due to the large number of plants, only a part of the field is shown.

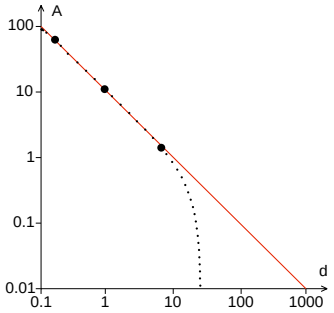

Figure 5: The average area of plants as a function of their density. Small dots represent the results of every tenth simulation step. Large dots correspond to the states of simulation shown in Figure 4.

For example, let us consider a model of plant distribution due to a fundamental phenomenon in plant population dynamics, *self-thinning*. This phenomenon is described by Ricklefs as follows [52, page 339]: "If one plots the logarithm of average plant weight as a function of the logarithm of density, data points fall on a line with a slope of approximately $-\frac{3}{2}$ [called the self-thinning curve]. [...] When seeds are planted at a moderate density, so that the beginning combination of density and average dry weight lies below the self-thinning curve, plants grow without appreciable mortality until the population reaches its self-thinning curve. After that point, the intense crowding associated with further increase in average plant size causes the death of smaller individuals."

Our model of self-thinning is a simplified version of that by Firbank and Watkinson [13]. The simulation starts with an initial set of circles, distributed at random in a square field, and assigned random initial radii from a given interval. If the circles representing two plants intersect, the smaller plant dies and its corresponding circle is removed from the scene. Plants that have reached a limit size are considered old and die as well.

Figure 4 shows three snapshots of the simulation. The corresponding plot shows the average area of the circles as a function of their density (Figure 5). The slope of the self-thinning curve is equal to -1; assuming that mass is proportional to volume, which in turn is proportional to area raised to the power of $-\frac{3}{2}$, the self-thinning curve in the density-mass coordinates would have the slope of $-\frac{3}{2}$. Thus, in spite of its simplicity, our model captures the essential characteristic of plant distribution before and after it has reached the self-thinning curve.

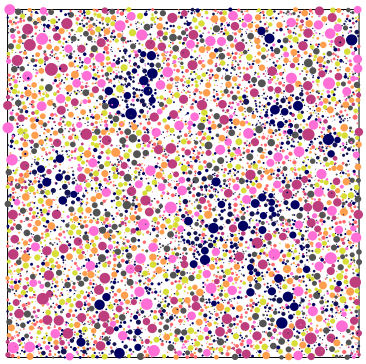

Figure 6: Simulated distribution of eight plant species in a terrain from Figure 3. Colors indicate plant species. Plants with a preference for wet areas are shown in blue.

A slightly more complicated model describes plant distribution in a population of different plant species. Each species is defined by a set of values that determine: (i) the number of new plants added to the field per simulation step, (ii) the maximum size of the plants, (iii) their average growth rate, (iv) the probability of surviving the domination by a plant with a competitive advantage, and (v) a preference for wet or dry areas. An individual plant is characterized by: (i) the species to which it belongs, (ii) its size, and (iii) its vigor. The vigor is a number in the range from 0 to 1, assigned to each plant as a randomized function of water concentration at the plant's location and the plant's preference for wet or dry areas. The competitive ability of a plant is determined as a product of its vigor and its relative size (the ratio between the actual and maximum size). When the circles representing two plants intersect, their competitive abilities are compared. The plant with a smaller competitive ability is dominated by the other plant and may die with the defined probability.

Figure 6 presents the result of a simulation involving a mix of eight plant species growing in a terrain shown in Figure 3. Plants with a preference for wet areas are represented by blue circles. Plants with a preference for dry areas have been assigned other colors. Through the competition between the species, a segregation of plants between the wet and dry areas has emerged.

Similar models can be developed to capture other phenomena that govern the development of plant populations.

5 MODELING OF PLANTS

Interactive editing of plant populations and the simulation of ecosystems determine positions and high-level characteristics of all plants in the modeled scene. On this basis, geometric models of individual plants must now be found.

Recent advances in plant measuring techniques have made it possible to construct a geometric model of a specific plant according to detailed measurements of its structure [54]. Nevertheless, for the purpose of visualizing plants differing by age, vigor, and possibly other parameters, it is preferable to treat geometric models as a product of the underlying procedural models. Construction of such models for computer graphics and biological purposes has been a field of active study, recently reviewed in [45]. Consequently, below we discuss only the issue of model parametrization, that is the incorporation of high-level parameters returned by the population model into the plant models. We consider two different approaches, which reflect different predictive values of *mechanistic* and *descriptive* models [60].

Mechanistic models operate by simulating the processes that control the development of plants over time. They inherently capture how the resulting structure changes with age [46, 47]. If a mechanistic model incorporates environmental inputs, the dependence of the resulting structure on the corresponding environmental parameters is an emergent feature of the model [39]. The model predicts the effect of various combinations of environmental parameters on the structure, and no explicit parametrization is needed. L-systems [47] and their extensions [39] provide a convenient tool for expressing mechanistic models. Within `EcoSys`, mechanistic models have been generated using `cpfg`.

Descriptive models capture plant architecture without simulating the underlying developmental processes. Consequently, they do not have an inherent predictive value. Nevertheless, if a family of geometric models is constructed to capture the key "postures" of a plant at different ages and with different high-level characteristics, we can obtain the in-between geometries by interpolation. This is equivalent to fitting functions that map the set of high-level parameters to the set of lower-level variables present in the model, and can be accomplished by regression [57] (see [48] for an application example). In the special case of plant postures characterized by a single parameter, the interpolation between key postures is analogous to key-framing [62], and can be accomplished using similar techniques. We applied interpolation to parametrize models created using both `xfrog` and `cpfg`.

6 APPROXIMATE INSTANCING

Geometric plant models are often large. A detailed polygonal representation of a herbaceous plant may require over 10MB to store; a scene with one thousand plants (a relatively small number in ecosystem simulations) would require 10GB. One approach for reducing such space requirements is to simplify geometric representations of objects that have a small size on the screen. We incorporated a version of this technique into our system by parameterizing the procedural plant models so that they can produce geometries with different polygonizations of surfaces. However, this technique alone was not sufficient to reduce the amount of data to manageable levels.

Instancing was used successfully in the past for compactly representing complex outdoor scenes (*e.g.* [56]). According to the paradigm of instancing [59], geometric objects that are identical up to affine transformations become instances of one object. To achieve a further reduction in the size of geometric descriptions, we extended the paradigm of instancing to objects that resemble each other, but are not exactly the same. Thus, sets of similar plants are represented by instances of a single representative plant. Furthermore, the hierarchical structure of plant scenes, which may be decomposed into groups of plants, individual plants, branches of different order, plant organs such as leaves and flowers, *etc.*, lends itself to instancing at different levels of the hierarchy. We create hierarchies of instances by quantizing model components in their respective parameter spaces, and reusing them.

Automatic generation of instance hierarchies for plant models expressed using a limited class of L-systems was considered by Hart [20, 21]. His approach dealt only with exact instancing. Brownbill [7] considered special cases of approximate instancing of plants, and analyzed tradeoffs between the size of the geometric models and their perceived distortion (departure from the original

geometry caused by the reduction of diversity between the components). He achieved reductions of the database size ranging from 5:1 to 50:1 with a negligible visual impact on the generated images (a tree and a field of grass). This result is reminiscent of the observation by Smith [55] that the set of random numbers used in stochastic algorithms for generating fractal mountains and particle-system trees can be reduced to a few representative values without significantly affecting the perceived complexity of the resulting images.

We generalize Brownbill's approach by relating it to clustering. Assuming that the characteristics of each plant are described by a vector of real numbers, we apply a clustering algorithm to the set of these vectors in order to find representative vectors. Thus, we reduce the problem of finding representative plants and instancing them to the problem of finding a set of representative points in the parameter space and mapping each point to its representative. We assume that plants with a similar appearance are described by close points in their parameter space; if this is not the case (for example, because the size of a plant is a nonlinear function of its age), we transform the parameter space to become perceptually more linear. We cluster and map plant parts in the same manner as the entire plants.

This clustering and remapping can be stated also in terms of vector quantization [18]: we store a code book of plants and plant parts and, for each input plant, we store a mapping to an object in the code book rather than the plant geometry itself. In computer graphics, vector quantization has been widely used for color image quantization [22]; more recent applications include reduction of memory needs for texture mapping [4] and representing light fields [31].

We use a multi-dimensional clustering algorithm developed by Wan *et al*. [61], which subdivides the hyperbox containing data points by choosing splitting planes to minimize the variance of the resulting clusters of data. We extended this algorithm to include an "importance weight" with each input vector. The weights make it possible to further optimize the plant quantization process, for example by allocating more representative vectors to the plants that occupy a large area of the screen.

7 RENDERING

Rendering natural scenes raises two important questions: (i) dealing with scene complexity, and (ii) simulating illumination, materials and atmospheric effects. Within `EcoSys`, we addressed these questions using two different strategies.

The first strategy is to to split the scene into sub-scenes of manageable complexity, render each of them independently using ray-casting, and composite the resulting RGBαZ images into the final image [12]. The separation of the scene into sub-scenes is a byproduct of the modeling process: both `densedis` and `cpfg` can output the distribution of a single plant species to form a sub-scene. The ray-casting algorithm is implemented in `fshade`, which creates the scene geometry procedurally by invoking the `xfrog` plant modeler at run time. This reduces file I/O and saves disk space compared to storing all of the geometric information for the scene on disk and reading it in while rendering. For example, the poplar tree shown in Figure 16 is 16 KB as a procedural model (plant template), but 6.7 MB in a standard text geometry format.

A number of operations can be applied to the RGBαZ images before they are combined. Image processing operations, such as saturation and brightness adjustments, are often useful. Atmospheric effects can be introduced in a post process, by modifying colors according to the pixel depth. Shadows are computed using shadow maps [64].

The scene separation makes it possible to render the scene quickly and re-render its individual sub-scenes as needed to improve the image. However, complex lighting effects cannot be easily included, since the renderer doesn't have access to the entire scene description at once.

The second rendering strategy is ray tracing. It lacks the capability to easily re-render parts of scenes that have been changed, but makes it possible to include more complex illumination effects. In both ray-tracers that we have used, `rayshade` [29] and `toro` [43], procedural geometry descriptions are expanded into triangle meshes, complemented with a hierarchy of grids and bounding boxes needed to speed up rendering [56]. `Rayshade` requires the entire scene description (object prototypes with their bounding boxes and a hierarchy of instancing transformations) to be kept in memory, otherwise page swapping significantly decreases the efficiency of rendering. In the case of `toro`, meshes are stored on disk; these are read in parts to a memory cache as needed for rendering computations and removed from memory when a prescribed limit amount of memory has been used. Consequently, the decrease in performance when the memory size has been reached is much slower [43]. We have made the straightforward extension of memory-coherent ray-tracing algorithms to manage instanced geometry: along with non-instanced geometry, the instances in the scene are also managed by the geometry cache.

Because rays can be traced that access the entire scene, more complex lighting effects can be included. For example, we have found that attenuating shadow rays as they pass through translucent leaves of tree crowns greatly improves their realism and visual richness.

8 EXAMPLES

We evaluated our system by applying it to create a number of scenes. In the examples presented below, we used two combinations of the modules: (i) ecosystem simulation and plant modeling using `cpfg` followed by rendering using `rayshade` or `toro`, and (ii) interactive specification of plant distribution using `densedis` in conjunction with plant generation using `xfrog` and rendering using `fshade`.

Figure 7 presents visualizations of two stages of the self-thinning process, based on distributions shown in Figure 4. The plants represent hypothetical varieties of *Lychnis coronaria* [47] with red, blue, and white flowers. Plant size values returned by the ecosystem simulation were quantized to seven representative values for each plant variety. The quantized values were mapped to the age of the modeled plants. The scene obtained after 99 simulation steps had 16,354 plants. The `rayshade` file representing this scene without instancing would be 3.5 GB (estimated); with instancing it was 6.7 MB, resulting in the compression ratio of approximately 500:1. For the scene after 164 steps, the corresponding values were: 441 plants, 125 MB, 5.8 MB, compression 21:1.

The mountain meadow (Figure 8 top) was generated by simulating an ecosystem of eight species of herbaceous plants, as discussed in Section 5. The distribution of plants is qualitatively similar to that shown schematically in Figure 6, but it includes a larger number of smaller plants. The individual plants were modeled with a high level of detail, which made it possible to zoom in on this scene and view individual plants. The complete scene has approximately 102,522 plants, comprising approximately $2 \cdot 10^9$ primitives (polygons and cylinders). The `rayshade` file representing this scene without instancing would be 200 GB (estimated), with the instancing it was 151 MB, resulting in a compression ratio of approximately 1,300:1.

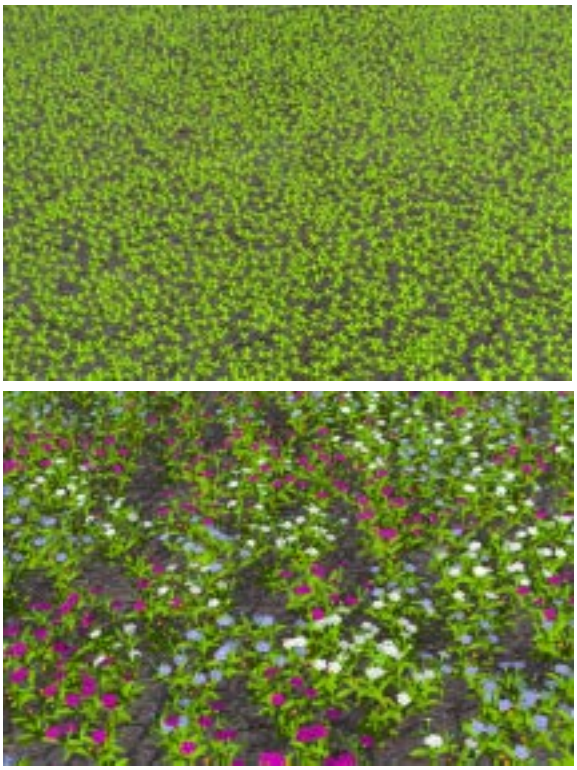

Figure 7: A *Lychnis coronaria* field after 99 and 164 simulation steps

The time needed to model this scene on a 150 MHz R5000 Silicon Graphics Indy with 96 MB RAM was divided as follows: simulation of the ecosystem (25 steps): 35 min, quantization (two-dimensional parameter space, each of the 8 plant species quantized to 7 levels): 5 min, generation of the 56 representative plants using `cpfg`: 9 min. The rendering time using `rayshade` on a 180 MHz R10000 Silicon Graphics Origin 200 with 256 MB RAM (1024×756 pixels, 4 samples per pixel) was approximately 8 hours. (It was longer using `toro`, but in that case the rendering time did not depend critically on the amount of RAM.)

In the next example, the paradigm of parameterizing, quantizing, and instancing was applied to entire groups of plants: tufts of grass with daisies. The number of daisies was controlled by a parameter (Figure 9). The resulting lawn is shown in Figure 10. For this image, ten different sets of grass tufts were generated, each instanced twenty times on average. The total reduction in geometry due to quantization and instancing (including instancing of grass blades and daisies within the tufts) was by a factor of 130:1. In Figure 11, a model parameter was used to control the size of the heaps of leaves. The heap around the stick and the stick itself were modeled manually.

Interactive creation of complex scenes requires the proper use of techniques to achieve an aesthetically pleasing result. To illustrate the process that we followed, we retrace the steps that resulted in the stream scene shown in Figure 15.

We began with the definition of a hilly terrain crossed by a little stream (Figure 2). To cover it with plants, we first created procedural models of plant species fitting this environment (Figure 12). Next, we extracted images representing the terrain altitudes and the stream position (Figures 13a and 13b) from the original terrain data. This

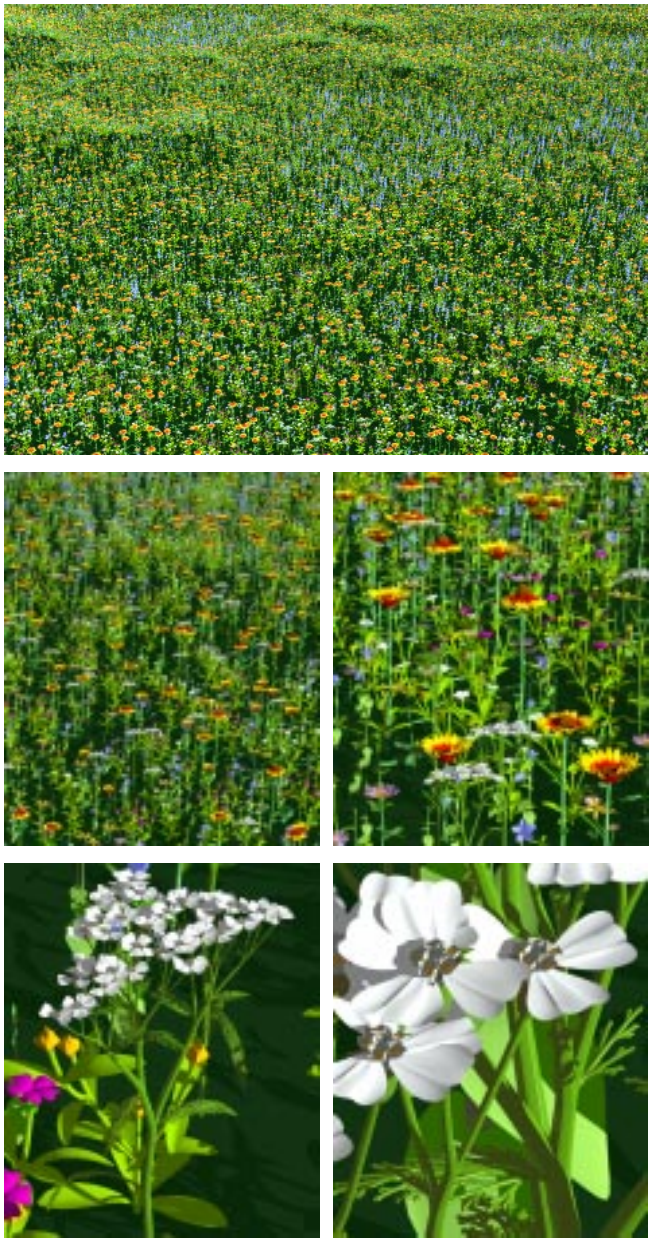

Figure 8: Zooming in on a mountain meadow

provided visual cues needed while painting plant distributions, for example, to prevent plants from being placed in the stream.

After that, we interactively chose a viewpoint, approximately at human height. With the resulting perspective view of the terrain as a reference, we painted a gray scale image for each plant species to define its distribution. We placed vegetation only in the areas visible from the selected viewpoint to speed up the rendering later on. For example, Figure 13c shows the image that defines the density distribution of stinging nettles. Since the stinging nettles grow on wet ground, we specified high density values along the stream. The density image served as input to `densedis`, which determined positions of individual plants. The dot diagram produced by `densedis` (Figure 13d) provided visual feedback that was used to refine the density image step by step until the desired distribution was found.

Figure 9: Grass tufts with varying daisy concentrations

Figure 10: Lawn with daisies

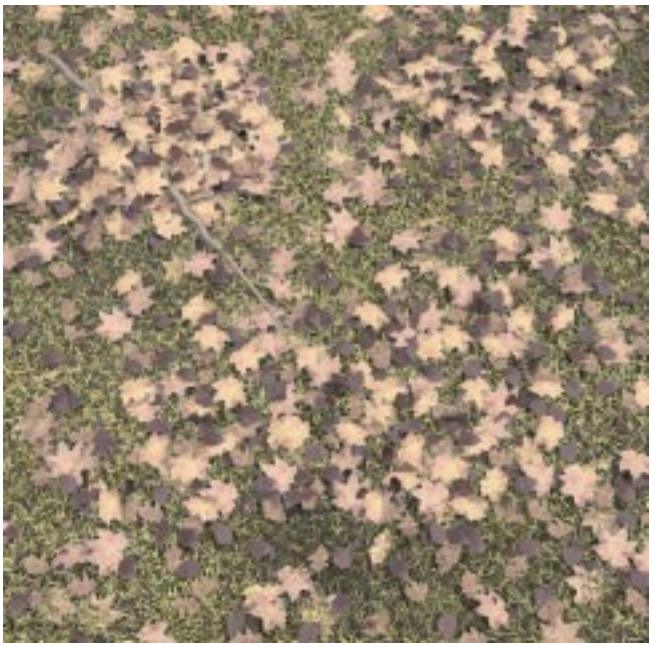

Figure 11: Leaves on grass

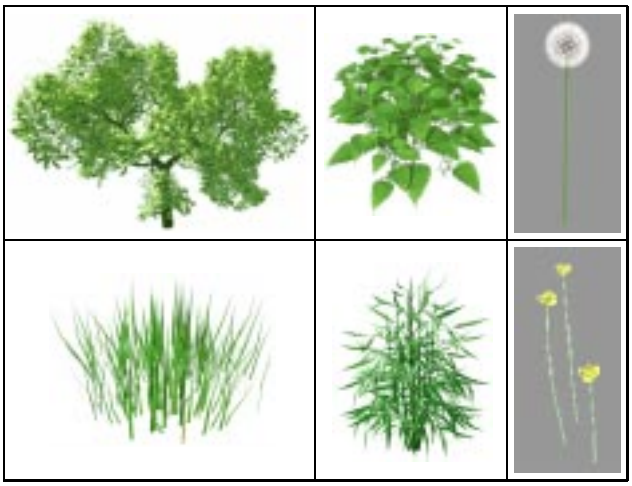

Figure 12: Sample plant models used in the stream scene. Top row: apple, stinging nettle, dandelion; bottom row: grass tuft, reed, yellow flower.

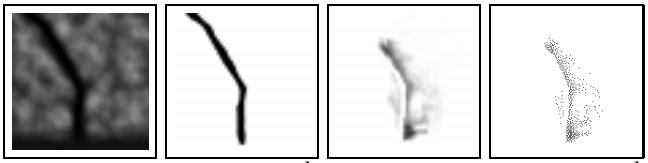

Figure 13: Creating distribution of stinging nettle: the heightmap of the covered area (a), the river image (b), the plant density distribution painted by the user (c), and the resulting plant positions (d).

Once the position of plants was established, we employed additional parameters to control the appearance of the plants. The vigor of stinging nettle plants, which affects the length of their stems and the number of leaves, was controlled using the density image for the nettles. To control the vigor of grass we used the height map: as a result, grass tufts have a slightly less saturated color on top of the hill than in the lower areas. Each tuft was oriented along a weighted sum of the terrain's surface normal vector and the up vector.

At this point, the scene was previewed and further changes in the density image were made until a satisfying result was obtained.

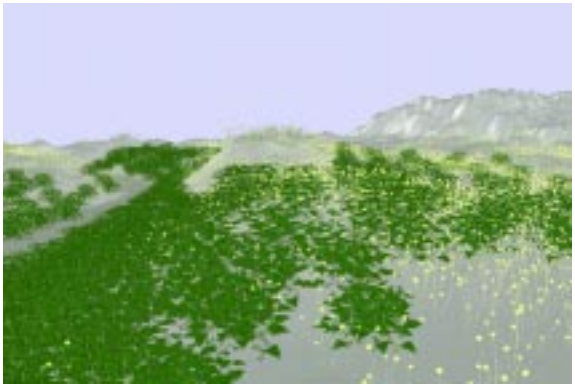

Figure 14: OpenGL preview of the stream scene including stinging nettle and yellow flowers

Figure 15: Stream scene

Figure 14 shows the preview of the distribution of stinging nettles and yellow flowers. To prevent intersections between these plants, the painted density image for the yellow flowers was multiplied by the inverted density image for the nettles.

The apple trees were placed by painting black dots on a white image. The final scene (Figure 15) was rendered using `fshade`. Images representing each species were rendered separately, and the resulting sub-scenes were composited as described in Section 7. The clouds were then added using a real photograph as a texture map. To increase the impression of depth in the scene, color saturation and contrast were decreased with increasing depth in a postprocessing step, and colors were shifted towards blue. Table 1 provides statistics about the instancing and geometric compression for this scene. The creation of this image took two days plus one day for defining the plant models. The actual compute time needed to synthesize this scene on a 195 MHz R10000 8-processor Silicon Graphics Onyx with 768MB RAM (1024 $\times$ 756 pixels, 9 samples per pixel) was 75 min.

Figures 16 and 17 present further examples of scenes with interactively created plant distributions. To simulate the effect of shadowing on the distribution of the yellow flowers in Figure 16, we rendered a top view of the spheres that approximate the shape of the apple trees, and multiplied the resulting image (further modified interactively) with the initial density image for the yellow flowers. We followed a similar strategy in creating Figure 17: the most important trees were positioned first, then rendered from above to provide visual cues for the further placements. Table 2 contains statistics about the geometry quantization in Figure 17.

plant	obj.	inst.	plant	obj.	inst.
apple	1	4	grass tuft	15	2577
reed	140	140	stinging nettle	10	430
dandelion	10	55	yellow flower	10	2751

Table 1: Number of prototype objects and their instances in the stream scene (Figure 15). Number of polygons without instancing: 16,547,728, with instancing: 992,216. Compression rate: 16.7:1.

plant	obj.	inst.	plant	obj.	inst.
weeping willow	16	16	reed	15	35
birch	43	43	poppy	20	128
distant tree	20	119	cornflower	72	20
St. John's wort	20	226	dandelion	20	75
grass tuft	15	824			

Table 2: Number of prototype objects and their instances in the Dutch scene (Figure 17). Number of polygons without instancing: 40,553,029, with instancing: 6,737,036. Compression rate: 6.0:1

Figure 16: Forest scene

Figure 17: Dutch landscape

9 CONCLUSIONS

We presented the design and experimental implementation of a system for modeling and rendering scenes with many plants. The central issue of managing the complexity of these scenes was addressed with a combination of techniques: the use of different levels of abstraction at different stages of the modeling and rendering pipeline, procedural modeling, approximate instancing, and the employment of space- and time-efficient rendering methods. We tested our system by generating a number of visually complex scenes. Consequently, we are confident that the presented approach is operational and can be found useful in many practical applications.

Our work is but an early step in the development of techniques for creating and visualizing complex scenes with plants, and the presented concepts require further research. A fundamental problem is the evaluation of the impact of quantization and approximate instancing on the generated scenes. The difficulty in studying this problem stems from: (i) the difficulty in generating non-instanced reference images for visual comparison purposes (the scenes are too large), (ii) the lack of a formally defined error metric needed to evaluate the artifacts of approximate instancing in an objective manner, and (iii) the difficulty in generalizing results that were obtained by the analysis of specific scenes. A (partial) solution to this problem would set the stage for the design and analysis of methods that may be more suitable for quantizing plants than the general-purpose variance-based algorithm used in our implementation.

Other research problems exposed by our experience with EcoSys include: (i) improvement of the terrain model through its coupling with the plant population model (in nature vegetation affects terrain, for example by preventing erosion); (ii) design of algorithms for converting plant densities to positions, taking into account statistical properties of plant distributions found in natural ecosystems [66]); (iii) incorporation of morphogenetic plasticity (dependence of the plant shape on its neighbors [58]) into the multi-level modeling framework; this requires transfer of information about plant shapes between the population model and the procedural plant models; (iv) extension of the modeling method presented in this paper to animated scenes (with growing plants and plants moving in the wind); (v) design of methods for conveniently previewing scenes with billions of geometric primitives (for example, to select close views of details); and (vi) application of more faithful local and global illumination models to the rendering of plant scenes (in particular, consideration of the distribution of diffuse light in the canopy).

Acknowledgements

We would like to acknowledge Craig Kolb for his implementation of the variance-based quantization algorithm, which we adapted to the needs of our system, and Christain Jacob for his experimental implementations and discussions pertinent to the individual-based ecosystem modeling. We also thank: Stefania Bertazzon, Jim Hanan, Richard Levy, and Peter Room for discussions and pointers to the relevant literature, the referees for helpful comments on the manuscript, Chris Prusinkiewicz for editorial help, and Darcy Grant for system support in Calgary. This research was sponsored in part by the National Science Foundation grant CCR-9508579-001 to Pat Hanrahan, and by the Natural Sciences and Engineering Research Council of Canada grant OGP0130084 to Przemyslaw Prusinkiewicz.

REFERENCES

[1] Alias/Wavefront; a division of Silicon Graphics Ltd. Studio V8. SGI program, 1996.

[2] AnimaTek, Inc. AnimatTek's World Builder. PC program, 1996.

[3] R. A. Armstrong. A comparison of index-based and pixel-based neighborhood simulations of forest growth. *Ecology*, 74(6):1707–1712, 1993.

[4] A. C. Beers, M. Agrawala, and N. Chaddha. Rendering from compressed textures. In *SIGGRAPH 96 Conference Proceedings*, pages 373–378, August 1996.

[5] B. M. Blumberg and T. A. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In *SIGGRAPH 95 Conference Proceedings*, pages 47–54, August 1995.

[6] B.N. Boots. *Spatial tesselations: concepts and applications of Voronoi diagrams*. John Wiley, 1992.

[7] A. Brownbill. Reducing the storage required to render L-system based models. Master's thesis, University of Calgary, October 1996.

[8] N. Chiba, K. Muraoka, A. Doi, and J. Hosokawa. Rendering of forest scenery using 3D textures. *The Journal of Visualization and Computer Animation*, 8:191–199, 1997.

[9] Adobe Corporation. Adobe Photoshop.

[10] O. Deussen and B. Lintermann. A modelling method and user interface for creating plants. In *Proceedings of Graphics Interface 97*, pages 189–197, May 1997.

[11] J. Dorsey, H. Køhling Pedersen, and P. Hanrahan. Flow and changes in appearance. In *SIGGRAPH 96 Conference Proceedings*, pages 411–420, August 1996.

[12] T. Duff. Compositing 3-D rendered images. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 19(3):41–44, 1985.

[13] F. G. Firbank and A. R. Watkinson. A model of interference within plant monocultures. *Journal of Theoretical Biology*, 116:291–311, 1985.

[14] K. W. Fleischer, D. H. Laidlaw, B. L. Currin, and A. H. Barr. Cellular texture generation. In *SIGGRAPH 95 Conference Proceedings*, pages 239–248, August 1995.

[15] R. W. Floyd and L. Steinberg. An adaptive algorithm for spatial gray scale. In *SID 75, Int. Symp. Dig. Tech. Papers*, pages 36–37, 1975.

[16] D. R. Fowler, P. Prusinkiewicz, and J. Battjes. A collision-based model of spiral phyllotaxis. *Computer Graphics (SIGGRAPH 92 Proceedings)*, 26(2):361–368, 1992.

[17] G. Y. Gardner. Simulation of natural scenes using textured quadric surfaces. *Computer Graphics (SIGGRAPH 84 Proceedings)*, 18(3):11–20, 1984.

[18] A. Gersho and R. M. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, 1991.

[19] D. G. Green. Modelling plants in landscapes. In M. T. Michalewicz, editor, *Plants to ecosystems. Advances in computational life sciences I*, pages 85–96. CSIRO Publishing, Melbourne, 1997.

[20] J. C. Hart and T. A. DeFanti. Efficient anti-aliased rendering of 3D linear fractals. *Computer Graphics (SIGGRAPH 91 Proceedings)*, 25:91–100, 1991.

[21] J.C. Hart. The object instancing paradigm for linear fractal modeling. In *Proceedings of Graphics Interface 92*, pages 224–231, 1992.

[22] P. Heckbert. Color image quantization for frame buffer display. *Computer Graphics (SIGGRAPH 82 Proceedings)*, 16:297–307, 1982.

[23] S. I. Higgins and D. M. Richardson. A review of models of alien plant spread. *Ecological Modelling*, 87:249–265, 1996.

[24] H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Conference Proceedings*, pages 189–198, August 1997.

[25] D. H. House, G. S. Schmidt, S. A. Arvin, and M. Kitagawa-DeLeon. Visualizing a real forest. *IEEE Computer Graphics and Applications*, 18(1):12–15, 1998.

[26] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. *Computer Graphics (SIGGRAPH 89 Proceedings)*, 23(3):271–289, 1989.

[27] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. *Computer Graphics (SIGGRAPH 86 Proceedings)*, 20(4):269–278, 1986.

[28] A. D. Kelley, M. C. Malin, and G. M. Nielson. Terrain simulation using a model of stream erosion. *Computer Graphics (SIGGRAPH 88 Proceedings)*, 22(4):263–268, 1988.

[29] C. Kolb. Rayshade. http://graphics.stanford.edu/∼cek/rayshade.

[30] M. P. Kumler. An intensive comparison of triangulated irregular networks (TINs) and digital elevation models (DEMs). *Cartographica*, 31(2), 1994.

[31] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 96 Conference Proceedings*, pages 31–42, August 1996.

[32] B. Lintermann and O. Deussen. Interactive structural and geometrical modeling of plants. To appear in the *IEEE Computer Graphics and Applications*.

[33] B. Lintermann and O. Deussen. Interactive modelling and animation of natural branching structures. In R. Boulic and G. Hégron, editors, *Computer Animation and Simulation 96*. Springer, 1996.

[34] Lucasfilm Ltd. The Adventures of André and Wally B. Film, 1984.

[35] D. Marshall, D. S. Fussel, and A. T. Campbell. Multiresolution rendering of complex botanical scenes. In *Proceedings of Graphics Interface 97*, pages 97–104, May 1997.

[36] N. Max. Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In X. Pueyo and P. Schröder, editors, *Rendering Techniques 96*, pages 165–174 and 288. Springer Wien, 1996.

[37] N. Max and K. Ohsaki. Rendering trees from precomputed Z-buffer views. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques 95*, pages 74–81 and 359–360. Springer Wien, 1995.

[38] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *Computer Graphics (SIGGRAPH 89 Proceedings)*, 23(3):41–50, 1989.

[39] R. Měch and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *SIGGRAPH 96 Conference Proceedings*, pages 397–410, August 1996.

[40] F. Neyret. A general and multiscale model for volumetric textures. In *Proceedings of Graphics Interface 95*, pages 83–91, 1995.

[41] F. Neyret. Synthesizing verdant landscapes using volumetric textures. In X. Pueyo and P. Schroöder, editors, *Rendering Techniques 96*, pages 215–224 and 291, Wien, 1996. Springer-Verlag.

[42] K. Perlin. An image synthesizer. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 19(3):287–296, 1985.

[43] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In *SIGGRAPH 97 Conference Proceedings*, pages 101–108, August 1997.

[44] P. Prusinkiewicz. Visual models of morphogenesis. *Artificial Life*, 1(1/2):61–74, 1994.

[45] P. Prusinkiewicz. Modeling spatial structure and development of plants: a review. *Scientia Horticulturae*, 74(1/2), 1998.

[46] P. Prusinkiewicz, M. Hammel, and E. Mjolsness. Animation of plant development. In *SIGGRAPH 93 Conference Proceedings*, pages 351–360, August 1993.

[47] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990. With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[48] P. Prusinkiewicz, W. Remphrey, C. Davidson, and M. Hammel. Modeling the architecture of expanding *Fraxinus pennsylvanica* shoots using L-systems. *Canadian Journal of Botany*, 72:701–714, 1994.

[49] Questar Productions, LLC. World Construction Set Version 2. PC program, 1997.

[50] W. T. Reeves and R. Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. *Computer Graphics (SIGGRAPH 85 Proceedings)*, 19(3):313–322, 1985.

[51] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics (SIGGRAPH 87 Proceedings)*, 21(4):25–34, 1987.

[52] R. E. Ricklefs. *Ecology. Third Edition.* W. H. Freeman, New York, 1990.

[53] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings*, pages 75–82, August 1996.

[54] H. Sinoquet and R. Rivet. Measurement and visualization of the architecture of an adult tree based on a three-dimensional digitising device. *Trees*, 11:265–270, 1997.

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

[56] J. M. Snyder and A. H. Barr. Ray tracing complex models containing surface tessellations. *Computer Graphics (SIGGRAPH 87 Proceedings)*, 21(4):119–128, 1987.

[57] R. R. Sokal and F. J. Rohlf. *Biometry. Third Edition.* W. H. Freeman, New York, 1995.

[58] K. A. Sorrensen-Cothern, E. D. Ford, and D. G. Sprugel. A model of competition incorporating plasticity through modular foliage and crown development. *Ecological Monographs*, 63(3):277–304, 1993.

[59] I. E. Sutherland. Sketchpad: A man-machine graphical communication system. Proceedings of the Spring Joint Computer Conference, 1963.

[60] J. H. M. Thornley and I. R. Johnson. *Plant and crop modeling: A mathematical approach to plant and crop physiology*. Oxford University Press, New York, 1990.

[61] S. J. Wan, S. K. M. Wong, and P. Prusinkiewicz. An algorithm for multidimensional data clustering. *ACM Trans. Math. Software*, 14(2):135–162, 1988.

[62] A. Watt and M. Watt. *Advanced animation and rendering techniques: Theory and practice*. Addison-Wesley, Reading, 1992.

[63] J. Weber and J. Penn. Creation and rendering of realistic trees. In *SIGGRAPH 95 Conference Proceedings*, pages 119–128, August 1995.

[64] L. Williams. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH 78 Proceedings)*, 12(3):270–274, 1978.

[65] L. Williams. Shading in two dimensions. In *Proceedings of Graphics Interface 91*, pages 143–151, June 1991.

[66] H. Wu, K. W. Malafant, L. K. Pendridge, P. J. Sharpe, and J. Walker. Simulation of two-dimensional point patterns: application of a latice framework approach. *Ecological Modelling*, 38:299–308, 1997.