Interactive Visualization of Complex Plant Ecosystems

Oliver Deussen[1] Carsten Colditz[1] Marc Stamminger[2,3] George Drettakis[2]

[1] Faculty of Computer Science, Dresden University of Technology, Germany
[2] REVES/INRIA, Sophia Antipolis, France, [3] now at Bauhaus-Universität Weimar, Germany

Abstract

We present a method for interactive rendering of large outdoor scenes. Complex polygonal plant models and whole plant populations are represented by relatively small sets of point and line primitives. This enables us to show landscapes faithfully using only a limited percentage of primitives. In addition, a hierarchical data structure allows us to smoothly reduce the geometrical representation to any desired number of primitives. The scene is hierarchically divided into local portions of geometry to achieve large reduction factors for distant regions. Additionally, the data reduction is adapted to the visual importance of geometric objects. This allows us to maintain the visual fidelity of the representation while reducing most of the geometry drastically. With our system, we are able to interactively render very complex landscapes with good visual quality.

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

Keywords: Synthetic Plants, Ecosystems, Point-based rendering, Level-of-detail Algorithms

1 INTRODUCTION

Interactive rendering of large outdoor scenes is an important task for applications such as landscaping, architecture or computer games as well as virtual reality and simulation. The display of rich natural vegetation enhances comprehension of spatial relations in a planned landscape, to judge the visual impact of new buildings or to imagine the shape of a landscape in the future.

In recent years, a number of plant generation systems have been developed that allow the user to generate the geometry of single plants with high quality. Various aspects in designing plant shapes, growth of plants and their interaction have been investigated by Prusinkiewicz et al. [15] based on L-Systems. Other approaches use parameterised algorithms [4] or a set of components like the xfrog system [10].

Based on these models, it is possible to generate large landscapes with many different plants. An open system architecture for such a system as well as algorithms for distributing and rendering complex scenes was presented by Deussen et al. [5], while commercial products like Bryce (http://www.metacreations.com), Animatek World Builder (http://www.digi-elements.com) use plants to populate whole planets. The limiting factor for interactive rendering of scenes is the huge amount of geometry that has to be processed for each frame.

Vegetation geometry differs from other objects in a significant manner. While man-made objects such as houses, streets or cars are usually represented by a limited set of relatively large connected surfaces, vegetation usually consists of many small and isolated surfaces. Thus, most traditional level-of-detail schemes cannot be applied effectively to such scenes.

On the other hand, in recent years new rendering techniques have been proposed that allow efficient rendering of complex objects using the most simple primitives one can imagine: points. These methods seem to be appropriate for plant scenes because collections of discrete points can convey the shape of distant plants naturally.

The importance of *interactive* yet realistic rendering of these very complex ecosystem models cannot be overstated. For landscape design, or any kind of architectural or public works assessment, interactive viewing opens enormous potential for interactive experimentation and trial-and-error evaluation, which has previously been very hard to achieve for models of this scale and complexity. Such an interactive renderer is an invaluable tool in all architecture/design applications, as well as for virtual reality or video games.

In this paper, we present a system for interactively rendering lage outdoor scenes. Our contributions are the introduction of an appropriate method for pre-processing the model data, an efficient rendering algorithm using a point- and line-based level-of-detail approach and optimization of the scene using hierarchies.

For pre-processing we introduce a combined polygon/point and line representation; in contrast to previous methods we use lines in addition to points. Long and thin parts of plants such as leaves or branches are represented by lines, while more compact objects are approximated by points. This enables us to better convey the shape for both object categories in distant views.

Using our representation, the number of displayed vertices per second can be minimized keeping image quality high. Additionally, a new method of importance reduction allows us to represent visual important parts of a scene in a higher qualilty than others. Doing so, we can drastically reduce unimportant parts of the scene while keeping the visual impression stable. Although we focus on representing plants, the method can be combined with other level-of-detail approaches for different kinds of geometry to form a full purpose system for interactive display of complex scenes.

For rendering, we introduce a novel approach for blending through representations, which is both efficient and results in high quality visual results. Blending is done in a way that more and more polygons of a plant are represented by points and lines. Level-of-detail is obtained by reducing the displayed number of points and lines in correspondence to the viewing distance. Both polygonal data,

points, as well as line approximations are stored in vertex arrays that allow us to efficiently display any desired portion of data.

We optimize the performance of our system using two types of hierarchies: Single plants such as trees that have high geometric complexity are spatially subdivided using an octree data structure. On the other hand, collections of many small plants are grouped locally and each group is entirely represented by a point and line representation. This enables us to display scenes of nearly arbitrary size efficiently.

After discussing related work we proceed by giving a system overview and describe how the data has to be pre-processed. The two forms of hierarchy are then introduced and rendering issues are presented. We conclude with results and some ideas of further work.

2 RELATED WORK

Previous approaches that are related to our work can be divided into two domains: level-of-detail representations and control for complex data and point-based rendering methods.

Level-of-detail (LOD) control: Among the many existing LOD methods one can distinguish between static methods that store an object by a small set of discrete representations that are blended into each other and dynamic methods that allow finer control of the geometry generated. In combination with outdoor scenes, the most simple way to form a discrete LOD is to use billboards that contain images of single plants[1] and replace the plant geometry for distant views. If the viewpoint changes, no parallax can be seen which makes the scenes flat and unrealistic. Due to its simplicity, this method is currently used in games and VR applications. Improvements are sets of impostors [19] and sprites with depth [20] that offer a limited amount of parallax.

Max et al. [12, 11] hierarchically replace parts of trees by precomputed views and is able to smoothly blend between these. Unfortunately, the proposed technique is too inefficient for interactive display. Perbert and Cani [13] explore an efficient rendering and animation method for meadows based on impostors.

Dynamic LOD representations for smooth objects were presented by Hoppe et al. [7]. Objects are represented by a base geometry and a set of vertex split operations that refine the triangle mesh. Methods applied to terrain rendering were presented by Duchaineau et al. [6] and by Hoppe [8]. Unfortunately, foliage cannot be processed this way. The discrete structure of the leaves allows edge collapses only to a very limited extend. Nevertheless, Rossignac and Borrel reduced geometry by collapsing nearby vertices regardless of the connection structure [17]. This can be applied to the foliage but changes the visual appearance drastically.

Point rendering: The first general idea of using points as display primitives was presented by Levoy and Whitted [9]. In the same year, Reeves and Blau [16] rendered complex trees using a set of small disks representing the foliage. Weber and Penn presented a level-of-detail representation of trees using sets of points for the leaves and lines for the tree skeleton [24]. As mentioned above, we extend this idea to model whole plant populations. Shade et al. [20] sampled geometric plant models using a modified ray tracing procedure. Based on the resulting point sets they were able to render images from various view points. This works well for distant views but results in holes and a grainy look for close-ups.

An LOD method based on points is presented by Rusinkiewicz and Levoy[18]. A hierarchy of point representations allows smooth reduction of the amount of geometry for smooth surfaces. Pfister et al. [14] present sophisticated rendering methods for point representations. Their method works in software and therefore is too slow for interactive rendering highly complex scenes.

Cohen et al. [3] combine a LOD representation for polygonal objects with point rendering using a multi-resolution graph which describes simplification operations that lead to a point representation. By displaying the prefix of this graph, geometry reduction is performed. In this manner, they are able to smoothly blend between geometry and point representations. A similar transition method is presented by Chen and Nguyen[2]. However, for our highly complex objects an explicit representation of refinement operations seems to be too inefficient. Given a single tree with half a million of leaves (see figure 7(b)) the resulting graph would need a large amount of memory. Instead, we use vertex arrays that store the geometry of plants in an appropriate manner that allows us to reduce geometry by showing only the first part of such an array.

Stamminger and Drettakis [21] present a point based method for plants using random sample positions on the surface. Each plant is described by a set of points obtained from these positions. A similar approach is described in [23], where a plant is rendered by a random set of points, which is just dense enough not to exhibit holes. As the authors determine the point positions individually for each frame, aliasing is introduced.

3 SYSTEM OVERVIEW

Plants for our system are generated from surface oriented plant descriptions produced by the plant modellers described previously. In a pre-processing step these models are converted into point and line sets. Both the polygonal description and its point or line approximation are stored in a data file for each plant.

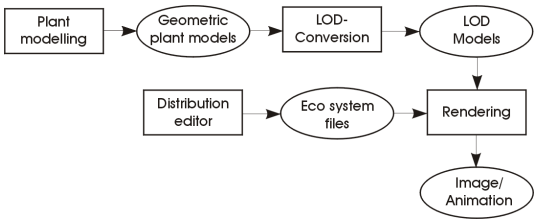


Figure 1: System structure, following the outline given in [5]

A separate editor, similar to the one described in [5], is used to combine the plants to generate complex scenes. This involves model quantization to reduce the number of different plant geometries. For a single plant population in most cases a set of five to ten different plant models is sufficient, the whole landscape may require several dozens.

Our scenes are described by a set of plant models and so called *eco files* that store positions of plants and instancing information. Instances of the plant models are spread over the terrain to form the plant population. Eco files can be used hierarchically, i.e., one eco file may by a combination of reference to other eco files.

The rendering system reads in eco files and plant geometries. For the entire content of each eco file a point approximation is determined. In distant views only this very sparse point representation is used. When zooming in, all the plants of the eco file are displayed. Visibility culling on the basis of bounding boxes is used to determine the visible plants. The system decides whether a polygonal representation will be used or its approximation as a function of the projected size of a plant. Figure 1 displays the system outline.

In the following, level-of-detail control is done with respect to the vertex count. This count describes the number of vertices that max-

imally can be processed by the graphics processor per second. In most cases, our scenes consist of many small triangles. The pixel fill rate of the graphics processor does not play a significant role in this case and the performance is thus fully determined by the vertex count.

4 PRE-PROCESSING OF MODELS

During modelling of a plant, each plant is divided into elements that are approximated by points and others that are approximated by lines in distant views. The user decides which parts of a plant model are represented by which primitive. This is only done during modelling the plants in an early stage. If several instances of a plant type with slightly varying geometry are used in a scene, modelling, with its corresponding effort, is only performed once and therefore is negligible. Another attribute that has to be set by the user is the visual importance of the plant parts that is later used for showing visually important plant parts with higher precision than others.

4.1 Point Representation

Point representations are typically used for leaves of a tree, petals etc. By default, for an object of n triangles, $2n$ points are generated, reflecting the fact that a separate triangle requires about as much time for rendering as three points and therefore an approximation by two points needs less rendering time. The $2n$ points are distributed randomly over the object, in no particular order, but in a way that equal surfaces receive equal numbers of points. This means that small triangles might not be attributed any points, whereas big triangles can be represented by many. All the random points are stored in a single list. During rendering, only a prefix of this list with length depending on the projected size of the plant is drawn [21] (see Fig. 2).

Figure 2: Point representation of a pine. Polygonal model and representation by 13,000, 6,500, 3,250, and 1,625 points. Low sampling density has been selected to visualize the point models.

A different sampling technique is applied if all triangles of the object have about the same area. In this case, we generate two sample points on each of the triangles. This corresponds to stratified sampling and leads to more even sampling. Furthermore, the triangles and points are randomly reordered in the same order, so we still know the point-triangle correspondence. This allow us to render part of the object by triangles, and the rest by points. In this manner, we can blend between polygonal and point representations smoothly, largely avoiding popping artefacts, as we will describe later.

4.2 Line Representations

Thin and long structures like twigs or several types of leaves are better represented by a line than by a set of points [24]. We generate a line set for these object parts already in the modelling phase, when all information about the object structure is available. The line set is also randomly reordered, and stored in a list. In the same manner as for the points, for each view only a prefix of this list is rendered, according to the current viewing distance. We will discuss this issue further in the rendering section.

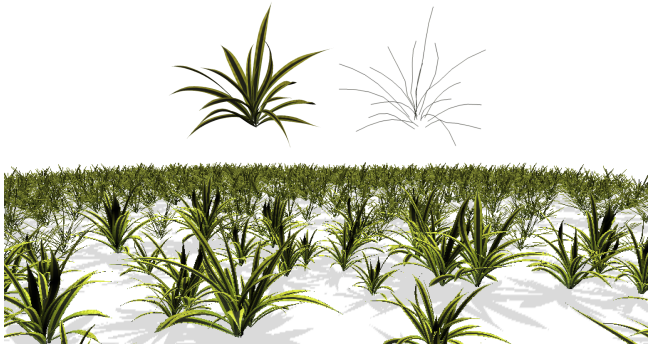
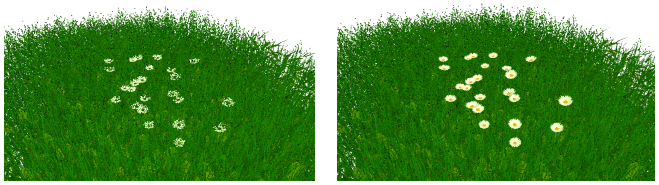


Figure 3: Representing a small palm by lines. Upper row: polygonal model and representation with 106 lines. Lower row: 1,500 plants distributed on a plane, original scene size is 12M triangles.

4.3 Importance Reduction

Best visual results can be achieved if the plants are not uniformly reduced. Visual important parts like petals or other plant elements with special colors should be reduced more slowly than than the rest of the plant. In Figure 4 an example is shown. The daisies in the meadow are visually important even though their geometry is small in comparison to the rest. If all geometry is reduced uniformly, the reduction is more noticeable, and the perceived quality much worse than if these elements are handled separately.

Currently, an importance factor is set interactively by the user. In the future this might be automated by color analysis of the geometric primitives: in most cases objects seem to be important if their color is clearly distinguishable from the majority of element colors in a plant.



(a) (b)

Figure 4: Importance reduction: a) uniform reduction; b) importance reduction, the visually important daisies are represented polygonally.

4.4 Memory Requirements

The additional point and line data roughly doubles the memory requirements for storing our models. This is why we will try to fully automate the entire pre-processing step. In this case the plant models can be stored as procedural plant descriptions in the form of

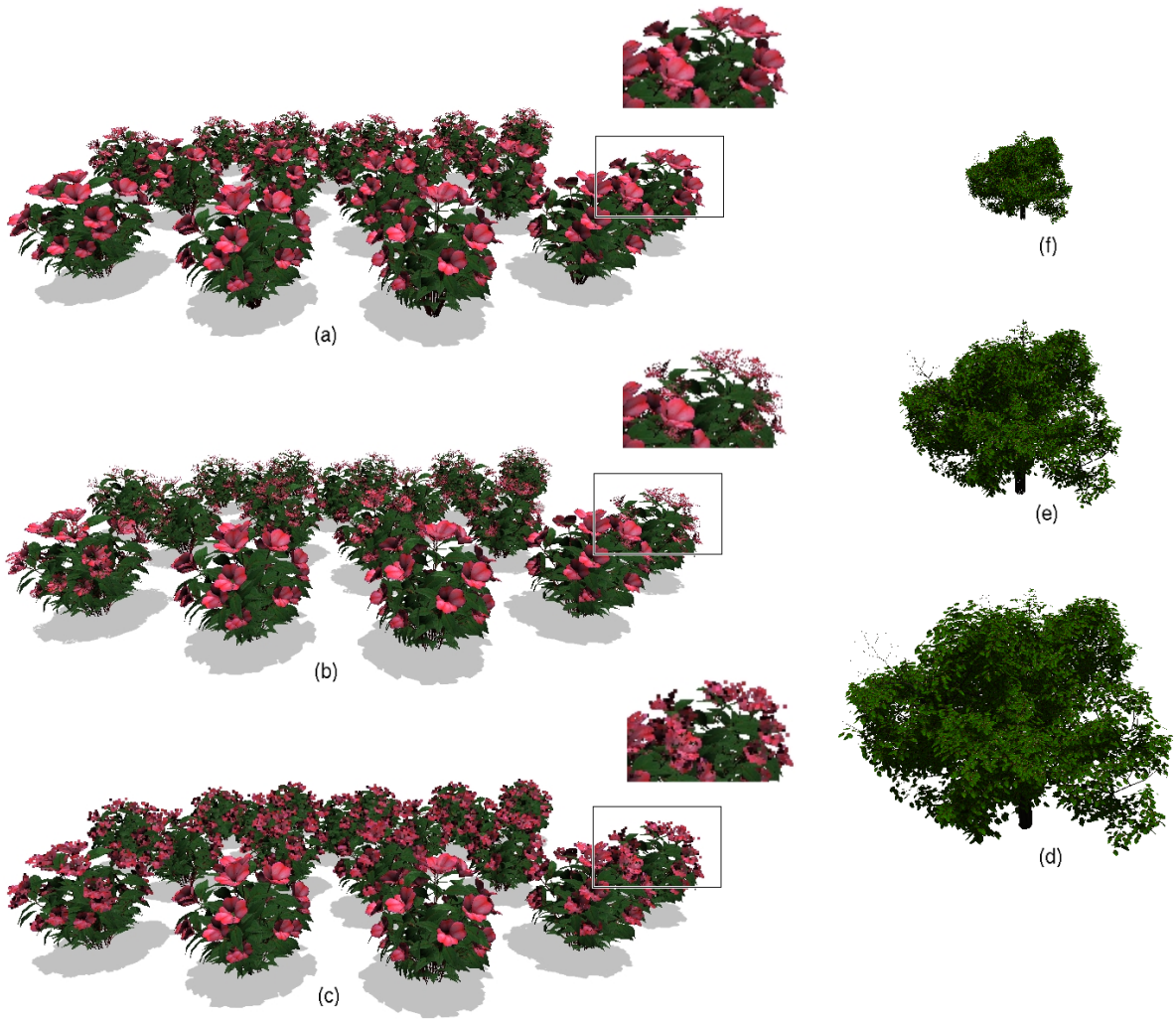


Figure 5: Left column: Influence of sampling distance d' and splat factor c_p: a) Sampling distance of about one pixel. Plants in the front are drawn by polygons, those in the back by points. b) With a larger sampling distance, more plants are rendered by points. An insufficient splat factor leads to holes in the rendering. c) Same sample distance as in b), but a larger splat factor leads to bigger splats, such that holes are reduced. Right column: Mixed representation of plants. d) a tree rendered by triangles, e) a mix of triangles and points, f) points only.

L-Systems or xfrog-files with very low memory needs. The extraction and approximation process then has to be done for each run in a pre-processing step. However, the memory needs for our plant models should are not very high, due to the approximate instancing mechanism in which a complex scene needs about several dozens of different models requiring usually less than a hundred megabytes of memory. For the scenes shown on the accompanying video, the entire geometric data was compact enough to be stored in AGP memory, which increases the rendering throughput by a factor of almost two.

5 RENDERING PLANT MODELS

After eco files and plant models are input into the rendering system, the geometry (polygons, points and lines) is stored in vertex arrays. During display, for each part of a plant its representation—triangles or points and lines—and the number of displayed primitives is determined dynamically.

Before rendering an object, visibility culling is applied. First, the culling is performed on each of the eco files, then on the plant model instances. If eco files are organized in a hierarchy, the culling is also applied hierarchically.

In the following, we assume that the original plant model consists of n triangles $T_1, .., T_n$ with area A in world space. Some of these triangles are approximated by n_l lines $L_1, .., L_{n_l}$ with overall length l, others are approximated by n_p points $P_1, .., P_{n_p}$. We now split the set of triangles $T_1, .., T_n$ of the model description into two sets, one set $TP = T_1, .., T_k$ that is approximated by points and the set $TL = T_{k+1}, .., T_n$ approximated by lines. The set TL has an area A_l, the set TP has an area A_p in world space. Clearly we have: $A = A_p + A_l$.

5.1 Rendering Points

In the rendering stage, point and line representations are handled separately: For the point approximation the number of points is calculated that is required to render the current object faithfully,

i.e. without holes and with correct coverage. Here each point of the approximation is represented by a small subimage, a so called splat, usually one pixel in diameter.

The user controls the speed versus quality tradeoff by setting a sample distance parameter d', that defines the average distance between two neighboring point samples on the image plane. A large value of d' results in a larger distance and thus fewer samples and vice versa. Consequently, the required point splatting area is $A'_{sp} = d'^2$ (in practice, we will use slightly bigger splat areas to avoid holes; see below).

The number of points that is needed for representing the plant faithfully depends on the surface A_p, as well as its average distance to the virtual camera r. In order to avoid scaling factors, we define the image plane to be at distance one from the camera. The approximate projected area A'_p of the triangle set TP on the image plane and the required point number p can be calculated by (see also [21]):

$$A'_p = \frac{1}{2}\frac{A_p}{r^2} \qquad\qquad p = c_p \frac{A'_p}{A'_{sp}} n_p \qquad (1)$$

In this formula it is assumed that the triangles are double sided, i.e., a leaf is modelled by a single surface and not two separate surfaces for front and back. The factor 1/2 accounts for leaf orientation under the assumption that all leaves are randomly distributed. $1/r^2$ corresponds to perspective foreshortening. The splat factor c_p allows us to increase the splat size in order to avoid gaps that appear due to the random sampling. Values from 1.2 to 1.5 are usually sufficient. If the number of required points $p > n_p$ the model part is rendered polygonally, otherwise by its point approximation using p points.

5.2 Rendering Lines

For the plant part that is to be approximated by lines the approach is slightly different. Here we have two requirements for the approximation: the projected area needed to represent the object faithfully and the geometric approximation error of the lines. This is due to the fact that two line sets with equal projected area can approximate the object differently.

Let the line set $L_1, .., L_{n_l}$ approximate the triangle set TL with a maximum distance ϵ_l in world space. In image space the value is compared with the sampling distance. If $\epsilon_l/r \geq d'$ we render the set by triangles, otherwise – as in the point case – we compute the image area if all lines were drawn. Doing so we need the projected area A'_l of the polygonal representation of corresponding triangle set TL. This is determined by projecting it similarly to Eq. (1). If l is the length of all lines in world space, the lines will have length $l/2r$ in the image plane, where again $1/2$ accounts for the (random) orientation, and $1/r$ for the perspective foreshortening ($1/r$ instead of $1/r^2$, because we project length and not area!). The image plane area covered by the lines drawn with line width d' is thus

$$A'_{sl} = l'\, d' = \frac{l d'}{2r}, \qquad (2)$$

We are now able to determine the ratio between projected area of TL and its line approximation $q_l = A'_l/A'_{sl}$. If $q_l \leq 1$ we draw $q_l\, n_l$ lines, otherwise the model part is drawn using polygons. Note that this assumes, that all line segments have about the same length, which is the case for the lines generated by our modelling tool.

5.3 Blending Representations

In the above section we described how and when to switch between a polygonal description and its approximation by points or lines. Depending on the selected sample distance d', this switch in representations may become visible (popping). However, this effect can easily be avoided by not switching between point and triangle representations immediately, but instead continuously replacing more and more triangles by points or vice versa.

As described above, for certain objects, we determined two random points on each triangle for the point representation. The point list is scrambled in the same way as the triangles, so that the first $2n$ points always correspond to the first n triangles. The selected level of detail depends on p, the number of requested points (see Eq. (1) and also Fig. 6).

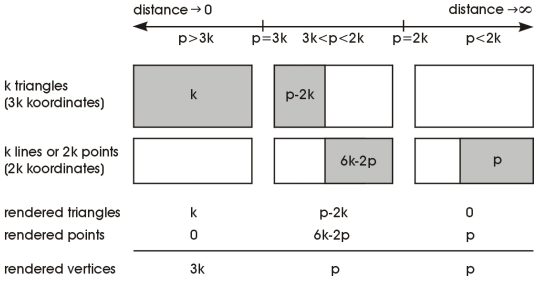


Figure 6: Rendering a plant in mixed polygon/line/point representations

If p is larger than three times the number of triangles k, that form the part of the plant that is to be approximated by points, the full triangle set is rendered. On the other hand, if $p < 2k$, we render the first p points of the point array. For $2k \leq p < 3k$, we blend between both representations by rendering the first $p-2k$ triangles, and the points corresponding to the remaining triangles. Due to the ordering enforced during the pre-process, these points can be rendered efficiently as vertex arrays.

If we count the number of rendered vertices (number of rendered points plus three times the number of triangles), we see that p vertices are always rendered, but never more than $3k$, so the rendering time is roughly linear in the number of pixels covered by the object, and essentially independent of object complexity.

The blending between polygons and their approximation works particulary well for the isolated surfaces which constitute plants. In Figure 5(d)-(f) the representation change is presented. While subfigure (d) shows the polygonal model, in (e) a mixed representation is given and in (f) a point approximation.

5.4 Implementation Issues

Image display can be altered by two parameters: the sampling distance d' and the splat factor c_p in Eq. 1. Fig. 5(a)-(c) demonstrates the influence of both parameters. If the sampling distance is increased, elements are converted into points earlier, resulting in fewer, but larger points. If the splat size is enlarged, resolution is decreased but the larger area of the splats allows a stronger reduction.

A noteworthy side effect of calculating the representations in the described way is that the sampling distance d' can be varied dynamically to ensure a desired vertex complexity. This enables animations with nearly constant frame rates.

Another nice aspect of our rendering method is that due to the fixed

size of the display elements a point representation often causes fewer aliasing artefacts than a polygonal representation with very small polygons. In this sense, our point representation causes aliasing due to its discrete structure, but due to the stable display of points in animations the point approximations of plants are visually more stable than their polygonal representation.

6 OPTIMIZATION

As mentioned above, in our system vegetation is represented by plant models and eco files with instancing information. In order to deal with the huge complexity of nature, we introduced hierarchical data structures in two ways: for the efficient rendering of complex individual plants and for the representation of very large numbers of small plants.

Since our rendering system is tightly coupled to the modeller, we directly work on its output, and need to get fast visual feedback. Resorting the objects, building completely new scene hierarchies, or prefiltering of the point sets is thus not possible. We essentially have to use the structure of the scene as provided by the modeller, and enhance it with lightweight hierarchical information. In the following we describe our solution, which adds hierarchical information even to complex scenes in a few seconds.

6.1 Representation of Complex Plants

Eqs. 1 and 2 provide a global mechanism for representing a plant geometry polygonally or by points and lines. This is in contrast to the local schemes presented in [3, 2]. The advantage of the global decision is that it is very efficient. Nonetheless, this approach has a drawback for large and complex plant models: If the viewer is close to a single leaf of a complex tree, due to Eq. (1) all the leaves are displayed polygonally because of the large projected size of the nearby leaves. This is not optimal for the distant leaves of the same plant that might be better represented by points or lines.

To reduce this effect, large and complex objects are spatially subdivided by an octree of small depth. All the triangles in a cell are represented by a polygon set and a point and line set. If the camera comes closer, the content of nearby cells is shown polygonally while distant cells can be approximated. The maple in Figure 7(b) is a typical example of a tree that benefits from such a structure.

6.2 Grouping of Plants

Rendering large scenes requires the storage and display of millions of plants. In this case, the iteration over all plants tends to be the limiting factor even if each plant is displayed by only a single point or line.

To deal with plant models of such size, vast plant populations are stored in a set of quadratic tiles that fit together. This results in a local grouping of plants. If a single tile has too many instances, it is further subdivided into sub-tiles, until a given number of instances per tile is achieved.

Each tile is represented by an eco file. For each eco file a rough point representation for the geometry of the entire tile is computed by randomly collecting points from the geometry approximations of the plant instances. Additionally, eco files can be organized hierarchically i.e., an eco file contains several eco files with positional information. If this is the case, the rough point approximations of the children are merged to form a point approximation of the overall content of the eco file. Doing so, we are able to display large areas.

In contrast to a quad-tree data structure this model-oriented spatial data structure gives us more freedom to organize different plant populations by several eco files that, for instance, obtain different importance parameters. This is why we prefer this data organization. Nevertheless, it is relatively easy to use a quad-tree data structure for the same purpose.

6.3 Caching

For performance reasons it is very important to load as much data as possible into memory which is quickly accessible by the graphics processor. Free AGP memory is usually quite limited, e.g., about 18MB on our GeForce3. Therefore, we determine how often a plant model is used in the current scene. The geometric data of frequently used models is primarily loaded into AGP memory. The previously described instancing scheme that limits the number of different models in our scenes helps in this respect by minimizing the amount of geometry needed. We are thus able to achieve a vertex rate of about eleven million vertices per second for rendering our scenes on a GeForce3.

6.4 Shadow Computation

Shadows are very important for correct light interaction in outdoor scenes. In combination with our interactive system we implemented two shadow map mechanisms. The first is to obtain a standard shadow map in a pre-processing step. This can be used efficiently during rendering but does not allow any dynamic objects and results in pixelisation artifacts for the large scenes we want to show.

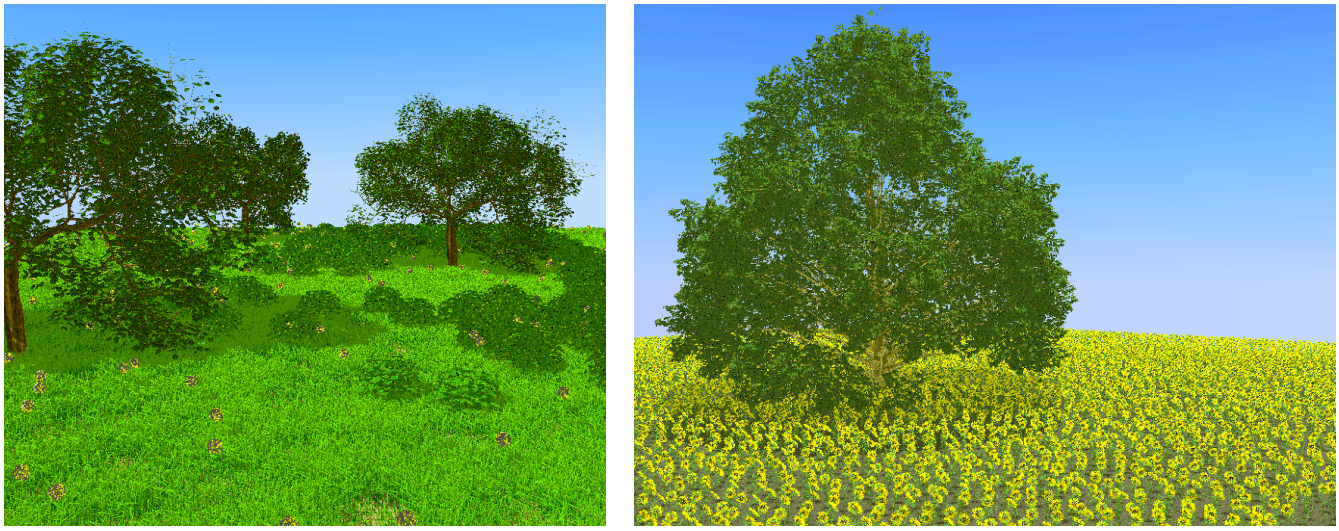
The second possibility is to use the perspective shadow map proposed by Stamminger and Drettakis recently [22]. In this case, the map is obtained for each image in a way that the pixel resolution is concentrated to areas where it is needed. This is done by computing the map after the perspective projection of the scene. The approach allows us to use a relatively small map without visible artefacts. On the other hand, the map has to be obtained for each frame, which reduces performance. When computing a conventional high resolution shadow map the scene has to be used without any level-of-detail approximation –a procedure that might last for some seconds– the perspective shadow map can be generated in level-of-detail mode seen from the camera. The frame rate is thus about 50% of the frame rate without shadow computation.

7 RESULTS AND FUTURE WORK

We have applied the method to a number of plant models and eco systems. With modern graphics hardware based on PC's –we used a 1,2 GHz Pentium with NVidia Geforce3 graphics processor– over eleven million vertices per second can be drawn. The vertex rate is enough for interactively displaying a number of fairly complex plant scenes as can be seen in the accompanying video.

In Figure 7(a) an outdoor scene is shown, that consists of over 120 million polygons. The original scene generated by Deussen et al. [5] is a part of what we use in our system. Their conventional renderer needed about 75 minutes for image generation of 30 million triangles computing 9 samples per pixel. In our system we achieve eight to ten frames per second for the much larger scene albeit without anti-aliasing.

Figure 7(b) shows another scene with a large tree and 13 thousand sunflowers, 70 million triangles in total. This scene is displayed typically at three to four frames per second. The relatively low performance stems from the models that allow no strict reduction



(a) (b)

Figure 7: a) Outdoor scene, rendered with typically 8-10 Hz, original model size: 120 million triangles, image: 795x700 ; b) sunflower field, rendered with 3-4 Hz average, original model size: 70 million triangles, image: 729x536

without visual artefacts. The line-approximated scene of Figure 3 consists of 12 million triangles and is typically displayed with five to ten frames. In Table 1 some values are given.

Table 1: Complexity of different scenes and count of elements in average during interactive rendering. Typical frame rates; if camera is very close smaller rates can result.

Model	Size (triang)	LOD Triang. (av.)	LOD Lines (av.)	LOD Points (av.)	Frame rate (av.)
640 trees as in Fig. 2	10M	500K	100K	400K	9-16 Hz
Fig. 3	12M	560K	40K	0	12-20 Hz
Fig. 7(a)	120M	460K	400K	300K	8-10 Hz
Fig. 7(b)	70M	450K	220K	400K	3-4 Hz

Future work will include the automatic texture synthesis for distant eco files and a simple approach for simulating wind. Using the ingredients described, a variety of users from fields like such architecture, landscaping, simulators, and game design can use scenes with rich vegetation for a real virtual reality.

8 ACKNOWLEDGEMENTS

Marc Stamminger has been supported by a Marie-Curie postdoctoral fellowship while doing this work.

References

[1] Iris performer programmer's guide, 1995.

[2] B. Chen and M. Nguyen. Pop: A hybrid point and polygon rendering system for large data. In *IEEE Visualization 2001*. IEEE, 2001.

[3] J. Cohen, D. Aliaga, and W. Zhang. Hybrid simplification: Combining multi-resolution polygon and point rendering. In *IEEE Visualization 2001*. IEEE, 2001.

[4] P. de Reffye, C. Edelin, J. Francon, M. Jaeger, and C. Puech. Plant models faithful to botanical structure and development. In J. Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 151–158. ACM SIGGRAPH, August 1988.

[5] O. Deussen, P. Hanrahan, M. Pharr, B. Lintermann, R. Měch, and P. Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. *Computer Graphics*, 32(3), SIGGRAPH 98 Conference Proceedings.

[6] M. Duchaineau, M. Wolinsky, D. Sigeti, M. Miller, C. Aldrich, and M. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. *IEEE Visualization 1997*, 1997.

[7] H. Hoppe. Progressive meshes. In Holly Rushmeier, editor, *Proceedings of SIGGRAPH 96*, pages 99–108, August 1996.

[8] Hugues Hoppe. View-dependent refinement of progressive meshes. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, pages 189–198. ACM SIGGRAPH, Addison Wesley, August 1997.

[9] M. Levoy and T. Whitted. The use of points as display primitives. Technical Report TR 85-022, Univ. of North Carolina at Chapel Hill, 1985.

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

[11] N. Max, O. Deussen, and B. Keating. Hierarchical image-based rendering using texture mapping hardware. *Eurographics Rendering Workshop 1999*, pages 57–62, June 1999.

[12] N. Max and K. Ohsaki. Rendering trees from precomputed Z-buffer views. In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.

[13] F. Perbert and M. Cani. Animating prairies in real-time. In *2001 ACM Symposion on interacive 3D Graphics*, pages 103–110, March 2001.

[14] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *SIGGRAPH 2000 Conference Proceedings*, pages 335–242.

[15] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.

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

[17] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T.L. Kunii, editors, *Geometric Modeling in Computer Graphics*, pages 455–465. Springer Verlag, Genova, Italy, 1993.

[18] S. Rusinkiewicz and M. Levoy. Qsplat: A mulitresolution point rendering system for large meshes. In *SIGGRAPH 2000 Conference Proceedings*, pages 343–352.

[19] Gernot Schaufler and Wolfgang Strzlinger. A three dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):227–236, August 1996. ISSN 1067-7055.

[20] J.W. Shade, S.J. Gortler, L. He, and R. Szeliski. Layered depth images. *Proceedings of SIGGRAPH 98*, pages 231–242, July 1998.

[21] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In S. Gortler and C. Myszkowski, editors, *Rendering Techniques 2001*, pages 151–162. Eurographics, Springer-Verlag, Vienna, 2001.

[22] M. Stamminger and G. Drettakis. Perspective shadow maps. In *SIGGRAPHH 2002 Conference Proceedings*. ACM Siggraph, 2002.

[23] Michael Wand, Matthias Fischer, Ingmar Peter, Friedhelm Meyer auf der Heide, and Wolfgang Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *SIGGRAPH 2001 Conference Proceedings*, pages 361–370.

[24] J. Weber and J. Penn. Creation and rendering of realistic trees. In R. Cook, editor, *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 119–128. ACM SIGGRAPH, August 1995.