Terrain Guided Multi-Level Instancing of Highly Complex Plant Populations

Andreas Dietrich Gerd Marmitt Philipp Slusallek

Computer Graphics Group, Saarland University *

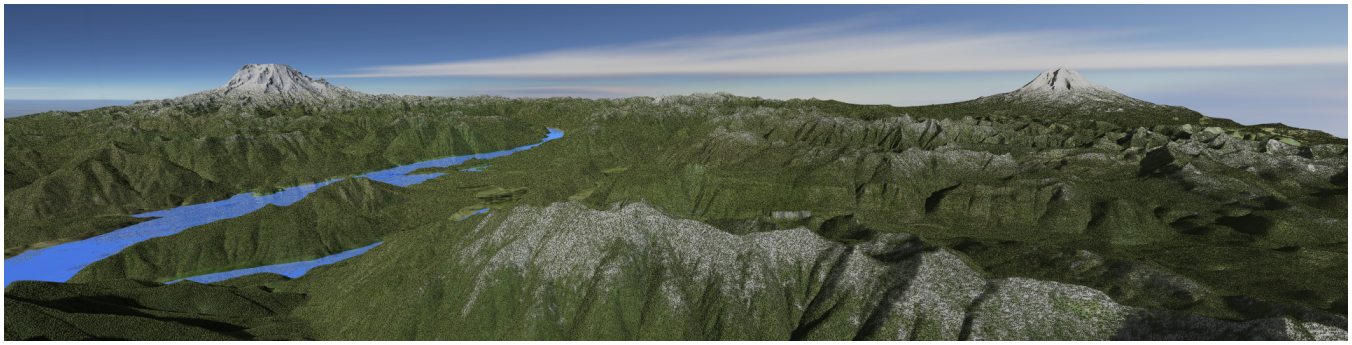


Figure 1: A panoramic view over a highly complex model of the Puget Sound Area. The ground terrain consists of 134 million triangles. It is covered with billions of plant instances, where each plant model is made up of several thousand polygons.

ABSTRACT

In this paper we demonstrate how todays ray tracing techniques can be applied to photo-realistically render extremely huge landscapes covered with trees and forests, where a user can freely choose between highly detailed close-up views or flyover scenarios. This is made possible by mapping a number of square sub-scenes onto a huge polygonal terrain during run-time. The full plant population results from the combination of these tiles, which are iterated over the terrain. This will be demonstrated at the example of a highly complex, plant covered ecosystem containing *trillions* of triangles.

CR Categories: I.3.3 [Computer Graphics]: Picture and Image Generation I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Ray Tracing I.6.2 [Simulation and Modeling]: Applications

Keywords: ray tracing, natural phenomena, complex scenes

1 INTRODUCTION

Natural landscapes contain an incredible amount of visual detail. For example, imagine an observer standing in a forest. Even for a limited field of view, hundreds of thousands of individual plants might be visible. Moreover, plants are made of highly complex structures themselves, e.g., countless leaves, complicated branchings, wrinkled bark, etc. And even if there is only a sparse vegetation, the underlying ground and rock formations can be equally rich in detail.

Although computer graphics has seen tremendous advances in the last two decades, *realistic* reproduction of such detail in synthetic images still remains a highly challenging problem, especially in an interactive context. This is because it is difficult to capture the complexity that can be perceived by the human eye in computer generated images. While it is already possible to produce convincing impressions for still images, this often requires considerable hand tweaking, and typically only allows for a small set of different camera positions without having to reorganize the scene according to the current viewing parameters. An observer can usually not go near plants or rocks and look a them closely, neither is it possible to wander through a complete fully-detailed landscape without having to make compromises regarding scene complexity.

In this paper we want to show how todays ray tracing techniques can be applied to get closer to the goal of photo-realistically rendering extremely huge landscapes covered with trees and forests, where a user can freely choose between highly detailed close-up views or flyover scenarios. This will be demonstrated at the example of a highly complex, plant populated terrain more than 80 $km \times 80$ km in size that is covered with *billions* of plants comprising *trillions* of triangle primitives.

2 DESCRIPTION OF COMPLEX LANDSCAPES

Explicitely storing a digital representation of a natural landscape with all possible visible detail would result in immense amounts of data. Even a model spanning only a few square meters could require dozens or hundreds of gigabytes of memory to represent all geometry, material and surface information. While an elaborate description of complex plant communities and terrains typically already exceeds available secondary storage space, keeping all data in main memory is hardly possible.

2.1 Implicit Model Description

One method to avoid such excessive memory consumption is to use *procedural* scene descriptions, which have the advantage that they can be evaluated during rendering. Common examples are fractal functions [22] for generating terrains, or L-Systems [24] to simulate plant growth. In addition to generating geometric information, plant population production algorithms can be applied to simulate a natural vegetation distribution, i.e., to describe individual plant positions [10, 11]. Unfortunately, such processes tend to be extremely compute intensive, and lead to unmanageable computing times that prevent rendering at interactive frame rates.

*e-mail: {dietrich, marmitt, slusallek}@cs.uni-sb.de

2.2 Approximate Instancing

However, it is not necessary to provide each single plant with an individual appearance. This is because the human eye can be fooled by sheer geometric complexity such that a scene can appear authentic even if all plants are absolutely identical. Empirical investigation has shown that individually bent and scaled copies of a few plant species are usually sufficient to generate a plausible impression of a complete landscape. This method of using many (slightly altered) plant copies from few representatives, which are placed throughout a scene is referred to as *approximate instancing* [10]. General instancing goes back to Sutherland [29]. An early example where it was used for the compact representation of complex plant scenes was given by Snyder and Barr [27]. Instancing for natural objects has since then been used quite frequently [13, 4, 10]. A detailed overview of related techniques for botanical models can be found in [11]. As we will see later on, instancing is a particularly powerful technique when it comes to rendering, especially for a ray tracer. Instances of geometric object classes can then be efficiently reused in the model over and over again with little memory overhead as only transformation matrices and object references need to be stored.

2.3 Multi-Level Instancing

Instancing can be applied hierarchically. Apart from reusing single plants, it is also possible to use this scheme within plant structures, e.g., for leaf instancing or recursive copies of branching complexes. While this already can significantly reduce geometric data, it is usually not sufficient for large scale scenes containing millions of plants.

A dramatic compression of geometry for such cases can be achieved by forming large plant populations by combining a number of equal base elements. This not only makes rendering easier, it additionally decreases the modeling effort. Here a terrain that is to be filled with plants is subdivided into (typically uniform) square regions called *tiles* [11]. A number of small square plant populations are created that serve as higher-level representatives. Tiles can then be arbitrarily assigned to these sub-scenes. The full plant population results from the combination of the tiles, which are iterated over the terrain. Aside from the base geometry used to define a few plant species, now only the positions of the individual sub-scenes need to be stored. Production of an authentically looking tiling, however, is not a trivial task. A popular method that allows for the production of plausible vegetation patterns by creating an *aperiodic tiling* are *Wang tiles* [5, 33, 34] , which will be introduced in more detail in Section 5.2.

3 Rendering of Complex Landscapes

Countless techniques have been proposed regarding the problem of visualizing complex natural scenes. Current landscape rendering approaches can be categorized into three principle domains: *level-of-detail* (LOD) techniques, *image-based* representations, and *ray tracing*. Most of todays systems typically use a combination of these methods and perform a tradeoff between interactivity and visual realism.

3.1 Level-Of-Detail

Level-of-detail has already been used in early computer animations. For example, in 1985 Reeves and Blau [25] represented trees using collections of disks in order to reduce the complexity of the foliage. Hand-tweaked colors and shadowing effects resulted in very esthetic images. Approximation of trees by points and lines was later proposed by Weber and Penn [35]. They represented the foliage by sets of points, which were placed inside agglomerated leaves. Tree skeletons were approximated by lines. This principle of replacing complicated polygonal geometry with simpler primitives like points has received a lot of attention, recent work in this area includes [28, 32, 9, 6].

Another common approach to realize level-of-detail are polygonal simplification algorithms, where the amount of geometry is reduced by decimating the number of polygons that represent a single object. Much research has been done in this field, and an extensive amount of literature exists. For an overview see, e.g., [18]. However, polygonal simplification algorithms are difficult to apply for plants because of their complex topology, which easily can lead to a change in overall appearance.

3.2 Image-Based Representations

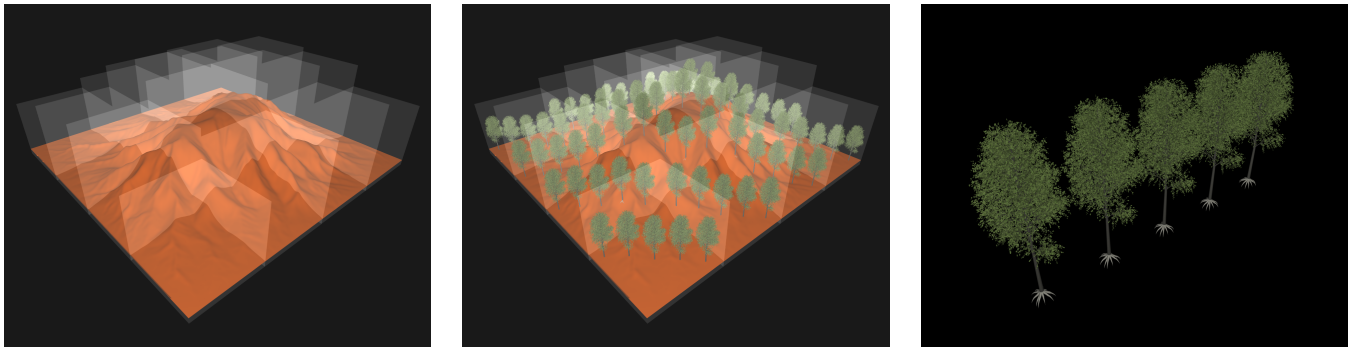
Image-based methods approximate complex geometry by using image-based entities, so-called *impostors*, as alternative representation. The simplest variant are *billboards*, where a number of textured, partly transparent polygons serve as impostors for distant objects. Consequently, they can only be used in the background, as otherwise parallax distortion would be noticeable. A better approximation can be achieved by using volumetric representations. For instance, sets of parallel billboards can represent trees, as done by Jakulin [14]. However, the parallel orientation results in visible artifacts. A related approach was followed by Neyret [20], who converted plants into volumetric textures, which were then ray traced. MIP maps were employed to reduce texture data, and to generate samples from distant trees. While high-quality images could be produced in a few minutes, the method was not applicable in an interactive context. In 2004 the system was adapted to graphics hardware [7]. However, an efficient level-of-detail mechanism was necessary to cope with the enormous amount of textures needed to describe large scenes.

More recently Behrendt et al. [2] presented a framework that makes use of a similar technique to render larger background parts of a scene. Here *shell textures* – a set of textured slices that are offset surfaces of the underlying terrain surface – approximate the plant layer. Trees visible at intermediate distances are displayed as *billboard clouds* [8], where a geometric model is represented by a set of arbitrarily oriented billboards. Behrendt et al. describe a clustering algorithm on the basis of the model vertices that allows for finding good, hierarchical billboard approximations. Moreover, their system incorporates spherical harmonics [26], which allow for realistic illumination. The system is able to render mid-sized scenes of a few ten thousand trees at interactive frame rates. However, accurate lighting can only be done locally per plant; global lighting effects like shadows from one object to another are not included.

3.3 Ray Tracing

Ray tracing of natural scenes has been done for over twenty years now, early examples include [15] and [27]. In order to handle scenes of a more realistic complexity, Pharr et al. [23] have proposed a caching and reordering mechanism that reorders the rays in a way that minimizes disk I/O. Though this allows for efficiently ray tracing datasets much larger than main memory, it is difficult to employ in an interactive context. A system specifically designed for high-quality rendering of natural ecosystems, presented by Deussen et al. [10], made use of a number of different techniques like approximated instancing, sub-scene compositing, and memory-coherent ray tracing. The system could render models with thousands of plants, built out of several million polygons. However, it has been exclusively used for offline rendering.

Interactive ray tracing has so far mainly found its application in visualization of highly complex CAD databases. The OpenRT real-time ray tracing engine [30, 31] has been shown to be capable



(a) (b) (c)

Figure 2: Populating a terrain with plant tiles. (a) Ground terrain subdivided into 4×4 cells. (b) Assigning a small plant sub-scene to each of the cells, where the plant z-position is defined by the underlying height field. (c) The used sub-scene containing five instances of one tree type.

of processing large outdoor scenes containing several million triangles in real-time. Additionally, it incorporates physically correct and global lighting, and features interactive placement of geometric parts. An extended framework was later shown to handle more realistically sized natural ecosystems with far more plants and vegetation layers [12]. The system incorporates advanced shading and lighting, including high dynamic range illumination from environment maps. It is capable of directly rendering scenes with hundreds of thousands of plants without simplification at interactive rates on a cluster of commodity PCs. However, the largest rendered model so far was still only 300 $m \times 300$ m in size.

4 Ray Tracing Tile-Based Landscapes

We will demonstrate in the following sections how to harness ray tracing techniques to render landscapes that contain almost five orders of magnitude more plants than shown in [12]. This can be done without any kind of model simplification due to the logarithmic time complexity of ray tracing with respect to scene size. No image-based representations will be involved except for textured plant leaves, a ground-layer texture, and an environment map for the sky.

The basic idea we follow is applying hierarchical multi-level instancing methods as described in Section 2, where a small number of fully-detailed sub-scenes are mapped onto a huge polygonal terrain. An illustration of the principle can be seen in Figure 2. The terrain displayed in Figure 2a is subdivided into 4×4 cells. These cells are then filled with tiles representing plant populations (Figure 2b). In this case the simple sub-scene depicted in Figure 2c is iterated over the terrain. The sub-scene itself is constructed out of five plant instances of one single tree model, which needs to be stored only once.

Basically, such a structure involves a hierarchical three-level spatial index, which is required to allow for fast ray traversal while avoiding unnecessary ray-polygon intersection tests. The next sections will provide a closer look into how this works.

4.1 Top-Level Terrain Traversal

Finding the cells that lie on a ray's path is a classical terrain rendering problem, as this is effectively a 2D traversal of a height field (i.e., a function $z = f(x,y)$). Although our cells are volumetric and can contain arbitrary geometry, it can for the targeted model type not happen that cells are above others. Therefore, a ray only needs to be tested for intersection with the top surface layer formed by the surfaces of the cells.

The cells can be seen as a 2D grid containing minimum and maximum z-values for each cell bounding box. A simple grid traverser only needs to step through the grid, performing some kind of 2D line drawing algorithm. In each step minimum and maximum z-value of the visited cell are compared against the z-value of the current ray. If both, the ray's entry and exit z-values lie above the box maximum z (or below the box minimum z) the cell is not pierced (as can, e.g., be seen in Figure 3 for cell 2, and entry and exit points z_2 and z_3).

Rather then using a 2D grid we implemented a 2D kd-tree [3], which allows for hierarchically skipping larger parts of the terrain. The leaves of the kd-tree simply contain the minimum and maximum z-values of the respective cells. The minimum and maximum z-values of the inner nodes are then the combined values from their children (dashed and dotted lines in Figure 3). This way, in a top-down kd-tree traversal, children can be skipped if a ray lies above (below) the top (bottom) bounding surface of an inner node. For example, in Figure 3 z_3 and z_5 are above the maximum z-value of cells 3 and 4. For sake of simplicity spatial subdivision, i.e., the placement of kd-tree splitting planes, is performed straightforward. During kd-tree construction bounding boxes are simply split in the middle of the longest dimension. It should, however, be easily possible to include more elaborate algorithms, like using *surface area heuristics* (SAH) [19] as cost prediction functions to determine good splitting plane positions. This has been done for the two remaining hierarchy levels (see below).

Before a kd-tree can be built, we need to determine the minimum and maximum z-values for each cell. Here we have to take the

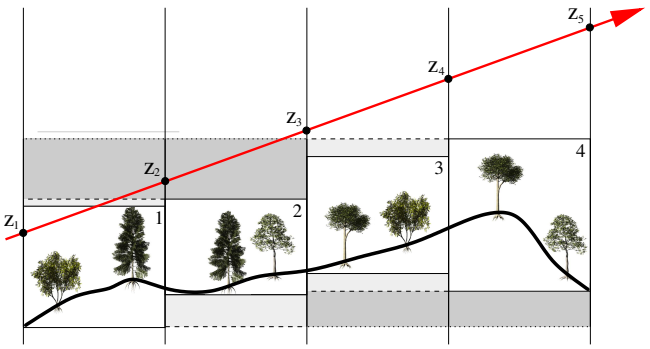


Figure 3: Top-level terrain kd-tree traversal. Dashed and dotted lines indicate how min/max z-values of kd-tree leaf nodes can be recursively combined to form min/max z-values for inner nodes.

extensions of the sub-scene into account that is to be mapped to the cell. This is illustrated in Figure 4. If the plants are aligned to the ground plane so that roots have negative coordinates, then the min/max z-values of the plant bounding box (z^p_{min} and z^p_{max}) simply have to be added to the min/max values (z^g_{min} and z^g_{max}) of the ground terrain in that cell, i.e.,

$$z_{min} = z^g_{min} + z^p_{min}, \quad z_{max} = z^g_{max} + z^p_{max}. \tag{1}$$

Obviously, this estimation can lead to over-conservative cell bounding boxes. This could be avoided by calculating z_{min} and z_{max} based on the actual plant z-positions in the respective cell. However, a correct calculation would require testing of potentially billions of plant instances. Of course, handling of negative coordinates can be neglected if it is guaranteed that the camera does not move below the ground.

This leaves us with the question how intersection tests inside a cell are being handled once a ray has been found to cross a terrain cell. Clearly, ground and the respective plant sub-scene that are contained within a cell have to be intersected separately. In our setup the ground terrain is a polygonal mesh generated from a 2D digital elevation map (DEM) since no fully-detailed geometric ground model was available. While it would have been possible to directly ray trace the original 2D height map, the intention behind using polygonal data is to allow for more complicated ground structures, e.g., overhanging rocks or ground covered with sand, stones, snow, etc.

The ground terrain itself is stored as a separate sub-scene and maintains its own three-dimensional SAH-based kd-tree to enable efficient ray-polygon intersection. Instead of splitting the ground into single patches that are stored independently inside the top-level cells, we use one large ground object. Before traversal of the top-level 2D kd-tree is started (see above), a ray is first tested against the complete ground object. If a hitpoint can be found, its distance from the ray origin is set as the new maximum ray distance. This way we can avoid having to switch between two traversal operations each time a ray enters a cell.

4.2 Lazy Two-Dimensional kd-Trees

As soon as a ray enters a top-level terrain cell, the plant tile assigned to the respective cell needs to be traversed. Each such a sub-scene incorporates a two-level kd-tree hierarchy. Here individual geometric objects (i.e, the plant species) maintain their own 3D SHA kd-tree, while the bounding boxes of the objects are themselves organized in a 2nd-level kd-tree. The 2nd-level kd-tree allows for instantiation of plants by simply keeping references to the geometric objects in conjunction with corresponding transformation matrices.

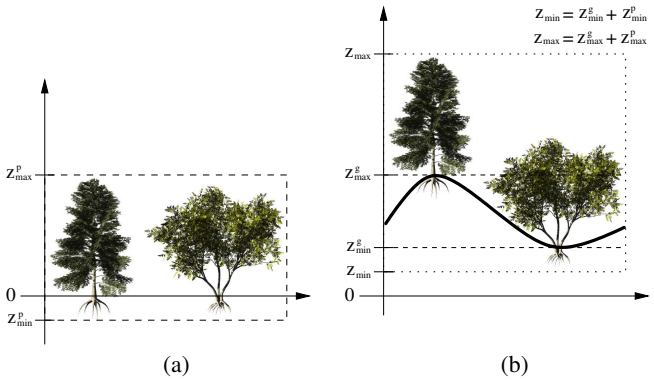


Figure 4: Approximate calculation of the min/max z-values of terrain bounding boxes. (a) Bounding box of a plant sub-scene. (b) The ground bounding box extended by the sub-scene bounding box.

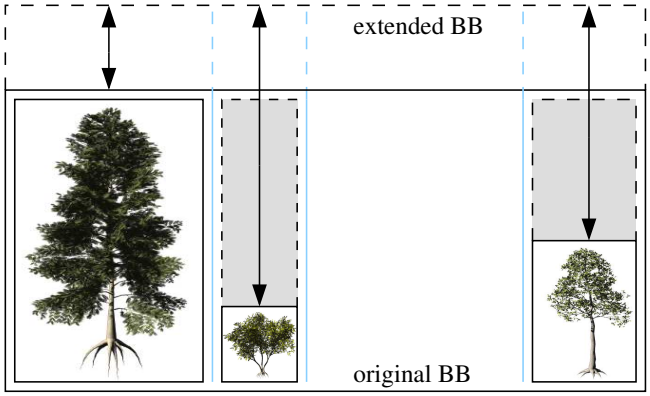


Figure 5: Sub-scene 2D kd-tree. Grey boxes indicate the extended object bounding boxes before kd-tree construction. During rendering the sub-scene bounding box is extended according to the terrain cell height. Trees can than be moved in z-direction within this range.

The problem that now arises is that the sub-scene needs to be mapped onto an uneven ground terrain. As a consequence, we cannot simply build a static three-dimensional 2nd-level kd-tree, since one and the same sub-scene can be used over and over again, and can potentially be mapped to any part of the terrain. Luckily, we can take advantage of the fact that all plants only need to be shifted in z-direction depending on the height map elevation. Thus, instead of building a full 3D 2nd-level kd-tree, we build a two-dimensional SAH tree by simply placing splitting planes only vertically, parallel to the x- and y-axes (see Figure 5). This can be easily accomplished by extending all plant bounding boxes in z-direction up to the upper sub-scene bounding box maximum before the 2nd-level kd-tree is constructed. During traversal we then only have to extend the upper bounding box z^p_{max} value by $z^g_{max} - z^g_{min}$, which will not affect the vertical splitting planes. When a ray enters a leaf voxel of the 2nd-level kd-tree, the boxes of the contained plant instances are then moved in z-direction according to the height map value at the plants' x-, y-positions. The ray is then sequentially transformed to each plant's object space, where it is tested against the object bounding box, and traversed through the object's 1st-level 3D kd-tree if necessary.

An optimization of this scheme would be to extend each plant bounding box by the value of its possible elevation range. This would allow for building a three-dimensional kd-tree including horizontal splitting planes. This, however, is only applicable if we place the sub-scene where the altitude of the ground terrain does not vary much. A disadvantage is that in such a case it is not possible to arbitrarily change the positions of plant tiles during run-time.

4.3 Adaptive Plant Density Reduction

Freely mapping a number of sub-scene tiles to a ground terrain allows for giving our landscape a natural look, but in practice this can require a great number of tiles. For a realistic appearance it is necessary to take into account how the plant density changes depending on local terrain conditions. For example, the population density decreases with increasing ground altitude. Instead of defining many small tiles with a large range of different densities and vegetation transitions, there is a much simpler way: During ray traversal of a sub-scene we already have to read the altitude for every plant from the height map. At this point we can calculate a certain probability, whether a plant can grow in this altitude or not, and can then simply exclude the respective instance from further intersection tests, thus effectively thinning out the vegetation.

More precisely, in the simplest variant we calculate a pseudo random value $p \in [0,1]$ for each instance, based on the current tile ID and instance ID, for example:

$$p = \frac{a \cdot (id_{inst} + id_{tile}) \mod m}{m-1}. \quad (2)$$

For our examples we chose $a = 47$ and $m = 256$. If p is larger than some density threshold $d \in [0,1]$, we just omit the instance, and do not intersect its object kd-tree. To determine d, we can use a linear attenuation between a lower and higher tree line:

$$d = d_{min} + \frac{(z - h_{min}) \cdot (d_{max} - d_{min})}{h_{max} - h_{min}}, \quad (3)$$

where z is the plant altitude, and d_{min} and d_{max} refer to the densities at a lower and higher tree line height (h_{min}, h_{max}), respectively. Additionally, we read the ground texture map at the plant's coordinates, and can decide from the color how the plant is stochastically skipped, e.g., in snow fields or desert areas.

5 Enhancing Realism

This section will sketch some techniques that were incorporated to make the visual appearance of the landscape more realistic. We will describe them only briefly, more information can be found in the indicated literature.

5.1 Environmental Lighting and Interleaved Sampling

In outdoor scenes illumination strongly depends on the incident light coming from the sky and distant environment. To take such effects into account, lighting can be based on high dynamic range environment maps. We perform shading and lighting calculations largely as described in [12]: A high dynamic range environment map is discretized, and the resulting regions are then approximated by a large but fixed set of virtual directional light sources in the spirit of [1, 17].

However, during camera motion, we only fire a single shadow ray towards the sun to allow for fast walkthroughs. As soon as the observer stops, we perform progressive improvement of image-quality by computing successive images from the same viewpoint with new random samples – using different virtual directional lights – and accumulate the resulting images.

A serious matter when dealing with scenes of the extreme complexity and sparse occlusion of outdoor environments is aliasing that results from the many small structures, like leaves or branches. To make best use of a given ray budget, it is advantageous to combine anti-aliasing of geometry and illumination by *interleaved sampling* [16]. Rather than using the same set of virtual directional lights for every primary ray, we can break them up into subsets, each of which applies to a different set of primary rays. Thus, the number of light samples can be increased while simultaneously improving pixel-anti-aliasing.

5.2 Aperiodic Tiling

Unfortunately, just iterating a number of sub-scenes over a large terrain will result in an artifical look, as the human eye is very sensitive in recognizing repeating patterns. In order to avoid repetition and aliasing artifacts (which are visible in grid-like structures), we used a Wang-tiling algorithm for the generation of our tiles. Wang tiles are square regions with color-coded edges. Tiles can only be connected to other tiles where adjacent edges share the same color. Placing tiles stochastically – while keeping colored edges matched – leads to a simple aperiodic tiling. Figure 6a and 6b show an example tile set and tiling. During the construction of the vegetation of a tile special care must be taken in a way, that plants that are placed near an edge must also be added to all matching tiles in case the plant extends over the edge. This means that sub-scenes must typically be defined larger than terrain cells in order to also include replicated plants (Figure 6c). For more details on how to construct plant populations using tiling algorithms see [5, 33, 34].

6 Results

As mentioned in the previous section, our landscape scene consists of a polygonal ground terrain, and a number of plant tiles that are mapped onto the terrain. As source data for the ground model we used a digital elevation map of the Puget Sound Area, obtained from the Georgia Institute of Technology's Large Geometric Models Archive [21]. From the original height map we extracted a subset of 8193×8193 samples. Assuming a 10 meter inter-pixel spacing, this results in a terrain spanning $81.92\ km \times 81.92\ km$. The elevation map was then regularly tessellated, yielding a polygon mesh consisting of 134,217,728 triangles. Including kd-tree information, the complete dataset for the ground model is roughly 21 GByte in size. In addition, a ground texture (8193×8193 texels, 193 MByte) was mapped onto the terrain.

For our test setup we created 18 different Wang tiles, i.e., sub-scenes with a complexity of approximately 82,000 plant instances per tile. The plant species themselves are highly complex alpha-textured polygonal objects, made of around 1,000 triangles for smaller plants, and up to 100,000 triangles for larger trees. On average, we have a total of around 454,000,000 triangles per tile. Due to the use of instances, the memory consumption for storing geometry data and kd-trees for the plant models can be kept low, and is only around 1 GByte of memory.

6.1 Rendering Performance

As hardware platform for all our experiments we utilized a shared-memory PC system, fitted with 8 dual-core 2.0 GHz AMD Opteron 870 CPUs, running Linux. A maximum of 64 GByte was available, which allowed us to fully keep all geometric and kd-tree data in-core.

An objective of this project was to test how scenes with excessive visual complexity affect ray tracing performance, and if a near interactive visualization is feasible with a moderate number of CPUs. We therefore subdivided the landscape into cells using a number of different resolutions ranging from 64×64 up to 512×512 terrain cells. At the highest resolution of 512×512, all terrain cells have an edge length of 160 m. With the above given plant density, this leads to a total amount of more than one hundred trillions potentially vis-

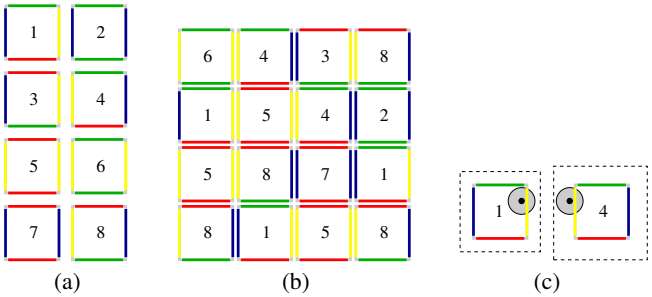


Figure 6: Wang tiling scheme used to cover the ground terrain with sub-scene plant tiles. (a) Example tile set with eight tiles and two horizontal and two vertical colors. (b) Resulting stochastic aperiodic example tiling. (c) Plants that extend over an edge need to be replicated for adjacent tiles.

ible triangles. If the adaptive plant density reduction mechanism is activated, roughly 24% of the plants (and triangles) are culled.

Table 1 lists the different cell grid resolutions and the corresponding scene complexity (considering adaptive plant culling). It is important to note that we did not create specific sub-scenes for lower grid resolutions. This means that for such cases the geometric size of terrain cells and sub-scenes are not conform, i.e., the plant to terrain relation is not correct. The frame time is measured for 640×480 pixels, with one primary ray per pixel and one shadow ray per primary ray (for the viewpoint of Figure 1). Please note that because of the alpha-textured leaves, often several secondary rays need to be spawned until a final hitpoint can be found.

Grid Resolution	Triangles	Frame Time (sec)
64×64	$1.4 \cdot 10^{12}$	19.85
128×128	$5.7 \cdot 10^{12}$	19.02
256×256	$22.6 \cdot 10^{12}$	18.75
512×512	$90.5 \cdot 10^{12}$	18.83

Table 1: Rendering performance for different terrain grid resolutions and scene complexities.

Interestingly, for such a high number of primitives, a linear scaling of scene size has almost no impact on rendering time, which is due to the logarithmic scaling property of the employed spatial index structures. That rendering gets actually faster sometimes is probably due to the different plant positions, but also because the trees are larger in relation to the ground for smaller grid sizes. Although we are not interactive at this resolution, it should be stressed that we are using a pure software implementation and only 16 CPUs. For smaller resolutions it is possible to quickly navigate to a desired position, and then render a high-quality image at a larger resolution.

7 Conclusions and Future Work

In this paper we have demonstrated how multi-level instancing in combination with ray tracing can be applied to photo-realistically render extremely huge, plant populated landscapes. By mapping a small number of individual sub-scenes repeatedly onto a large terrain, highly complex natural scenes can be formed, containing billions of single plants, formed by trillions of potentially visible triangle primitives. Parallel rendering on a shared-memory PC allows for near interactive frame rates, without any kind of model simplification or approximation, even under complex lighting conditions.

While we do not reach the same high frame rates as current state-of-the-art rasterization-based landscape rendering systems, e.g., [2], the presented framework is able to display scenes that are larger by several orders of magnitude, even with advanced lighting effects.

The shown methods should primarily seen as a motivation to further spur research in the combination of procedural run-time scene generation and photo-realistic rendering. A promising direction of future research will be to investigate how plants can be placed fully procedurally during run-time without having to rely on pre-constructed tiles. This problem is closely related to the fast construction of spatial acceleration structures, in particular for dynamic ray tracing applications. Another problem that needs to be addressed is the frequently appearing geometric aliasing artifacts caused by the large number of primitives and sparse occlusion in natural scenes. Standard level-of-detail methods are here difficult to employ because of the intensive use of object instances and the extreme scene size, which makes complicated preprocessing impractical.

Acknowledgments

We would like to thank Javor Kalojanov for his help as well as Carsten Colditz and Oliver Deussen for providing the plant models, and for help with related literature. The height map of the Puget Sound Area was provided by the Georgia Institute of Technology's Large Geometric Models Archive. All plant models were created at the University of Konstanz using the Xfrog modeling software.

References

[1] Sameer Agarwal, Ravi Ramamoorthi, Serge Belongie, and Henrik Wann Jensen. Structured Importance Sampling of Environment Maps. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, pages 605–612, 2003.

[2] Stephan Behrendt, Carsten Colditz, Oliver Franzke, Johannes Kopf, and Oliver Deussen. Realistic Real-Time Rendering of Landscapes Using Billboard Clouds. In *Computer Graphics Forum*, pages 507–516, 2005. (Proceedings of Eurographics).

[3] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] Andrew Brownbill. *Reducing the Storage Required to Render L-System Based Models*. PhD thesis, University of Calgary, 1996.

[5] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang Tiles for Image and Texture Generation. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, volume 22, pages 287–294, 2003.

[6] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential Point Trees. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, volume 22, pages 657–662, 2003.

[7] Philippe Decaudin and Fabrice Neyret. Rendering Forest Scenes in Real-Time. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering)*, pages 93–102, 2004.

[8] Xavier Décoret, Frédo Durand, François X. Sillion, and Julie Dorsey. Billboard Clouds for Extreme Model Simplification. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, volume 22, pages 689–696, 2003.

[9] Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive Visualization of Complex Plant Ecosystems. In *IEEE Visualization 2002*, pages 219–226, 2002.

[10] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomír Měch, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, pages 275–286, 1998.

[11] Oliver Deussen and Bernd Lintermann. *Digital Design of Nature – Computer Generated Plants and Organics*. Springer, 2005. ISBN 3540405917.

[12] Andreas Dietrich, Carsten Colditz, Oliver Deussen, and Philipp Slusallek. Realistic and Interactive Visualization of High-Density Plant Ecosystems. In *Natural Phenomena 2005, Proceedings of the Eurographics Workshop on Natural Phenomena*, pages 73–81, 2005.

[13] John C. Hart. The Object Instancing Paradigm for Linear Fractal Modeling. In *Proceedings of Graphics Interface '92*, pages 224–231, 1992.

[14] Aleks Jakulin. Interactive Vegetation Rendering with Slicing and Blending. In *Eurographics 2000 (Short Presentations)*, 2000.

[15] Timothy L. Kay and James T. Kajiya. Ray Tracing Complex Scenes. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, pages 269–278, 1986.

[16] Alexander Keller and Wolfgang Heidrich. Interleaved Sampling. In *Rendering Techniques 2001*, pages 269–276, 2001. (Proceedings of the 12th Eurographics Workshop on Rendering).

[17] Thomas Kollig and Alexander Keller. Efficient Illumination by High Dynamic Range Images. In *Rendering Techniques 2003*, pages 45–50, 2003. (Proceedings of the 14th Eurographics Workshop on Rendering).

[18] David P. Luebke. A Developer's Survey of Polygonal Simplification Algorithms. *IEEE Computer Graphics and Applications*, 21(3):24–35, 2001.

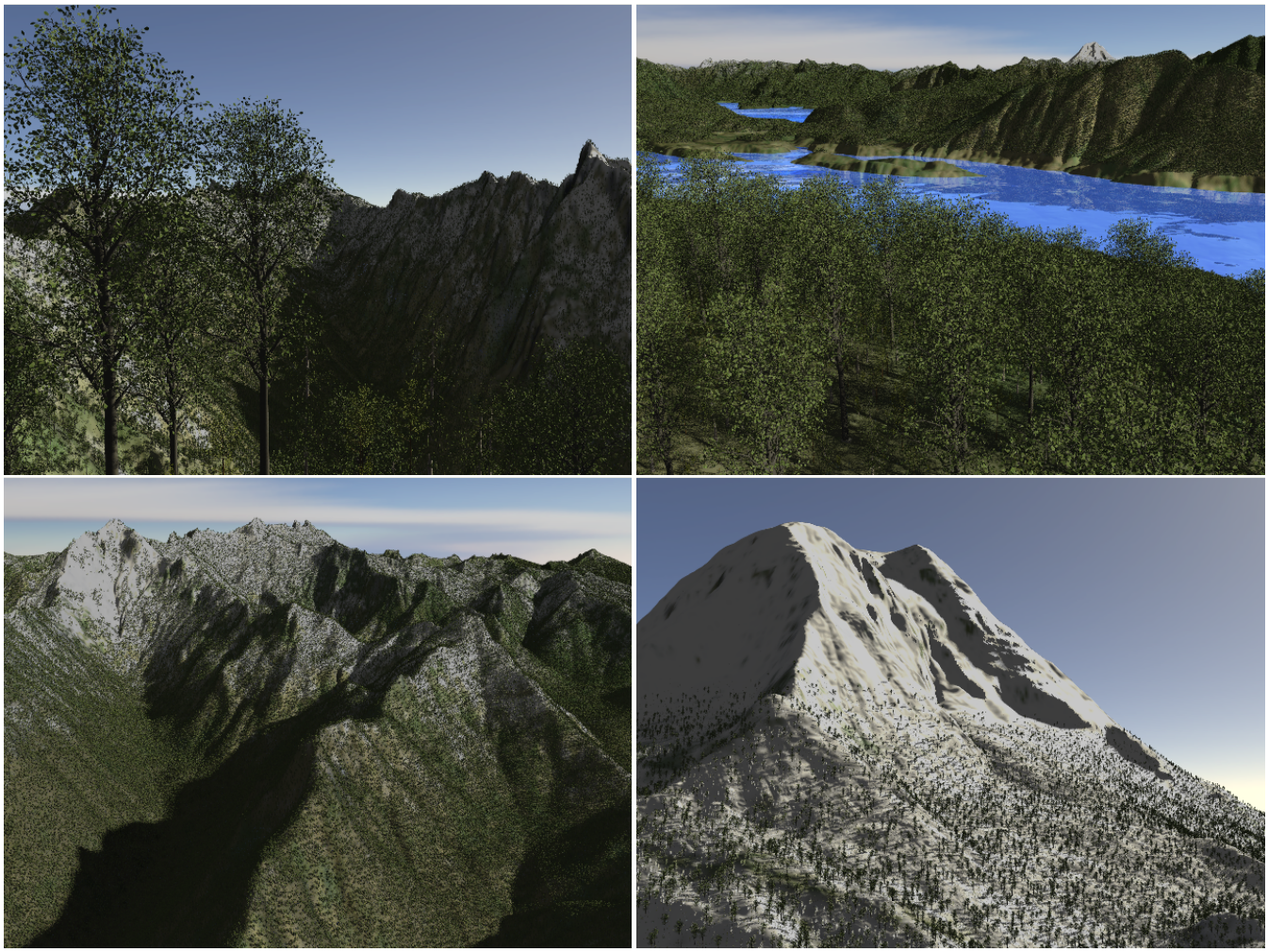


Figure 7: Some example screenshots from different areas of the landscape. Note the high geometric complexity that is visible for close-up views.

[19] J. David MacDonald and Kellogg S. Booth. Heuristics for Ray Tracing using Space Subdivision. In *Proceedings of Graphics Interface '89*, pages 152–163, 1989.

[20] Fabrice Neyret. Modeling Animating and Rendering Complex Scenes using Volumetric Textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):55–70, 1998.

[21] Georgia Institute of Technology. Large geometric models archive. http://www.cc.gatech.edu/projects/large_models.

[22] Heinz-Otto Peitgen and Dietmar Saupe (eds.). *The Science of Fractal Images*. Springer, New York, NY, USA, 1988.

[23] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering Complex Scenes with Memory-Coherent Ray Tracing. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, pages 101–108, 1997.

[24] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer, New York, NY, USA, 1990. ISBN 0387946764.

[25] William T. Reeves and Ricki Blau. Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, pages 313–322, 1985.

[26] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. In *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, pages 527–536, 2002.

[27] John M. Snyder and Alan H. Barr. Ray Tracing Complex Models Containing Surface Tessellations. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, pages 119–128, 1987.

[28] Marc Stamminger and George Drettakis. Interactive Sampling and Rendering for Complex and Procedural Geometry. In *Rendering Techniques 2001*, pages 151–162, 2001. (Proceedings of the 12th Eurographics Workshop on Rendering).

[29] Ivan E. Sutherland. Sketchpad – A Man-Machine Graphical Communication System. In *Proceedings of the Spring Joint Computer Conference (AFIPS)*, pages 328–346, 1963.

[30] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004. Available at http://www.mpi-sb.mpg.de/~wald/PhD/.

[31] Ingo Wald, Carsten Benthin, Andreas Dietrich, and Philipp Slusallek. Interactive Ray Tracing on Commodity PC Clusters – State of the Art and Practical Applications. In *Euro-Par 2003. Parallel Processing, 9th International Euro-Par Conference, 2003. Proceedings*, pages 499–508, 2003.

[32] 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 *Computer Graphics (Proceedings of ACM SIGGRAPH)*, pages 361–370, 2001.

[33] Hao Wang. Proving Theorems by Pattern Recognition. *Bell Systems Technical Journal*, 40:1–42, 1961.

[34] Hao Wang. Games, Logic, and Computers. *Scientific American*, pages 98–106, 1965.

[35] Jason Weber and Joseph Penn. Creation and Rendering of Realistic Trees. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, pages 119–128, 1995.

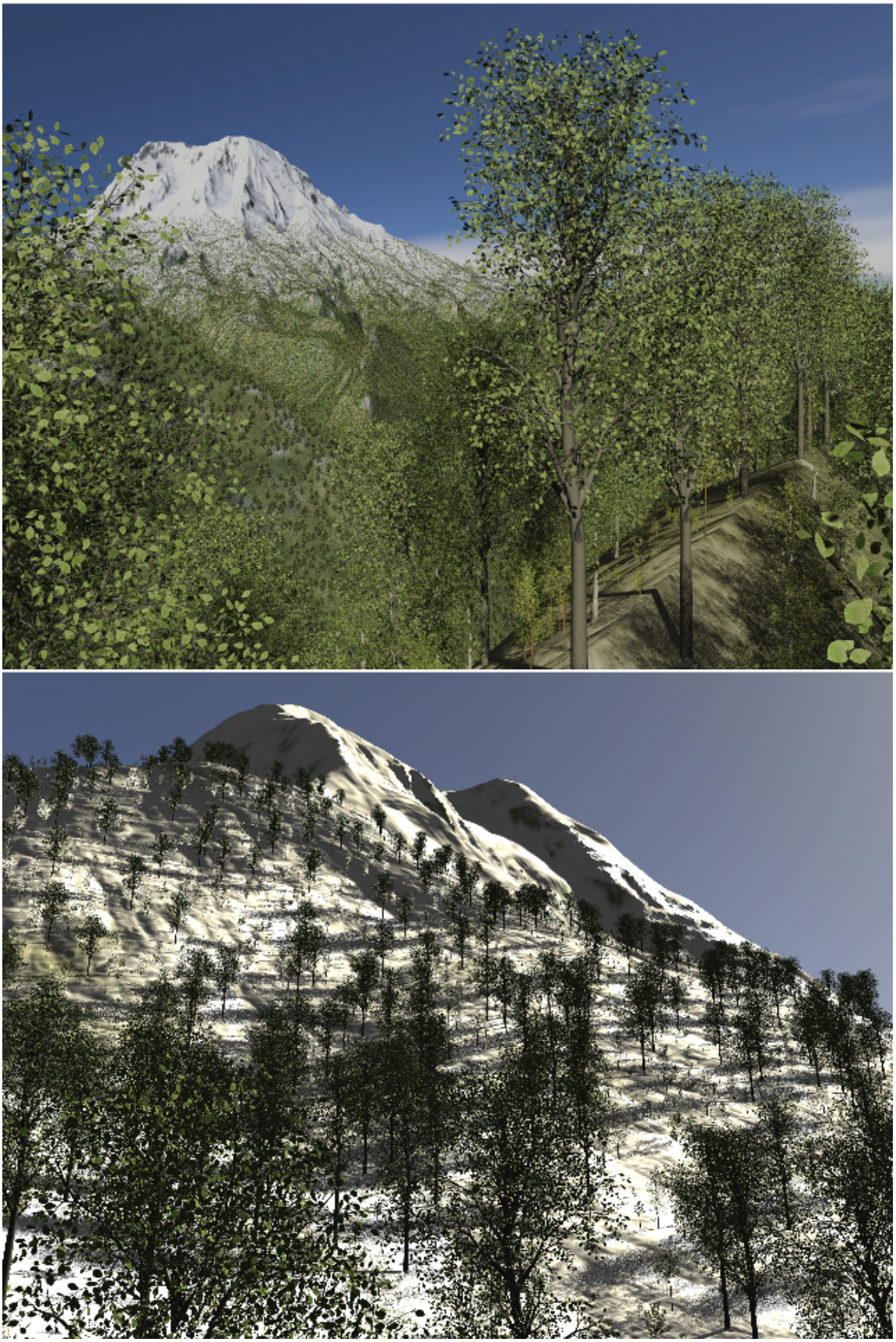


Figure 8: Example close-up views on some of the trees. All leaves are modeled as alpha-textured polygon meshes, which result in a high number of transparency rays. The scene is fully ray traced each frame, without any kind of precomputation, and without geometric simplification.