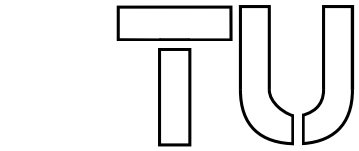


TECHNISCHE UNIVERSITÄT WIEN

Dissertation

Interactive Vegetation Rendering

ausgeführt
zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

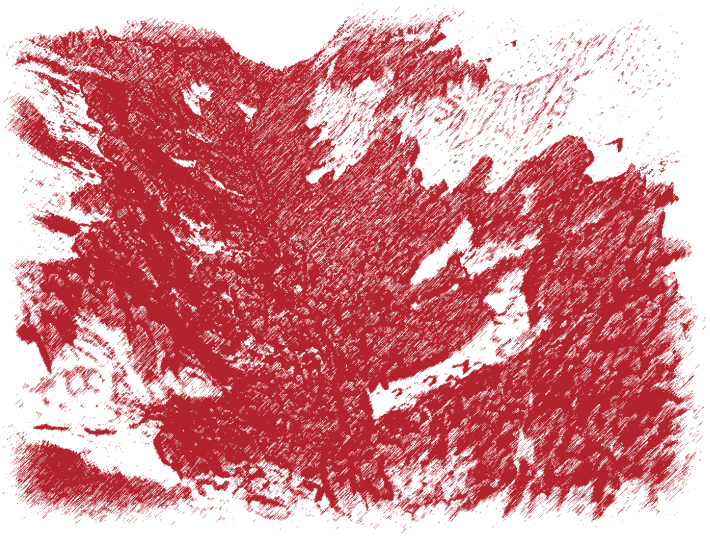
unter der Leitung von
Priv.-Doz. Dipl.-Ing. Dr.techn. Helwig Hauser
eingereicht
an der Technischen Universität Wien,
Fakultät für Technische Naturwissenschaften und Informatik,

von
Dipl.-Ing. Stephan Mantler,
Matrikelnummer 9325834,
Payergasse 1/17,
A-2340 Mödling, Österreich,
geboren am 1. Dezember 1974 in Wien.

Mödling, im März 2007.

INTERACTIVE VEGETATION RENDERING

STEPHAN MANTLER



MAY 8, 2007

If you want others to be happy,
practice compassion.
If you want to be happy,
practice compassion.

His Holiness the 14th Dalai Lama of Tibet

About the Author
Stephan Mantler, born 1974 in Vienna, received the degree of Dipl.-Ing. in Computer Science from the Vienna University of Technology in 1998. Before joining VRVis to pursue his doctorate, he has worked as a researcher and software developer for Imagination Computer Services GmbH, as a professional services specialist for Cable & Wireless Austria, and as an independent graphics software developer for numerous media and arts projects.

This work was financially supported by the Austrian Research Fund (FWF) contract no. P17260. The research was performed at VRVis, a research center partially funded by the Austrian Kplus program.

Parts of this thesis are based on peer reviewed publications written by the author in collaboration with colleagues from VRVis and the Institute of Computer Graphics at the Vienna University of Technology.

The following data sets were provided by other agencies and used with permission: SRTM elevation data courtesy of NASA/JPL. Landsat 7 global mosaic courtesy of University of Maryland/i-cubed. NPHT terrain and vegetation data courtesy of the Nationalpark Hohe Tauern administration. LIDAR scan data courtesy of Sorin Popescu, Texas A&M University Spatial Sciences Laboratory.

Abstract

Vegetation poses a significant problem to computer graphics because the geometric complexity of plants does not lend itself very well to traditional simplification approaches. This thesis presents new algorithms that address these issues at different scales, for rendering individual plants as well as entire landscapes.

For individual plants we introduce Displacement Mapped Billboard Clouds, an extended version of the *billboard cloud* extreme simplification approach. Billboard clouds have been successfully used to reduce the geometric complexity of highly detailed models to a few planes, however the resulting models are often unsuitable for viewing at closer distances. The presented extension exploits shaders to improve the visual quality of the resulting models.

Also, a method is introduced for quickly determining approximate visible sets for point clouds, which are often used for rendering individual plants. Approximate visible sets allow a significant reduction in the number of primitives to be rendered with only very little impact on visual quality.

For entire landscapes, displacement mapping shaders are explored to enhance existing terrain models with vegetation. We also address the issues involved with applying such techniques at a global scale, and present the integration of our method in the open source *World Wind* geospatial viewer.

Furthermore, we propose a way to enable early-Z acceleration methods on the GPU for shaders where this is not yet possible, and discuss the handling of level of detail validity and criteria for time-critical rendering of discrete and continuous levels of detail.

Kurzfassung

In der Computergrafik ist die interaktive Darstellung von Pflanzen nach wie vor ein bedeutendes Problem. Deren organische Struktur besitzt oftmals eine sehr große geometrische Komplexität, die nur schlecht mit herkömmlichen Verfahren reduziert werden kann. Diese Arbeit präsentiert neue Lösungsansätze sowohl für die Darstellung individueller Pflanzen als auch für ganze Landschaften.

Zur Darstellung einzelner Pflanzen wurde eine Erweiterung des *Billboard Cloud* Verfahrens zur extremen Vereinfachung von polygonalen Modellen entwickelt. Dieses Verfahren reduziert beliebig komplexe Objekte auf sehr wenige Polygone; allerdings ist das Ergebnis oft nur für die Betrachtung aus größeren Distanzen sinnvoll. Die von uns entwickelten *Displacement Mapped Billboard Clouds* erlauben eine stark verbesserte Darstellungsqualität, sodass die reduzierten Modelle über einen deutlich größeren Bereich angezeigt werden können.

Weiters werden für individuelle Pflanzen oft punktbasierte Darstellungsmethoden verwendet. Ein in dieser Arbeit vorgestelltes Verfahren erlaubt ein rasches, approximatives Erfassen der sichtbaren Punktmenge. In weiterer Folge kann diese reduzierte Punktmenge zur Darstellung verwendet werden, wodurch eine deutlich schnellere Darstellung bei nahezu gleichbleibender Bildqualität erzielt wird.

Für das interaktive Rendering ganzer Landschaften wurde ein Verfahren entwickelt, das auf *Displacement Mapping Shader* setzt um existierende Terrainmodelle mit Vegetation darzustellen. Zusätzlich zu dem eigentlichen Verfahren werden auch Probleme bei der Handhabung sehr großer Gebiete diskutiert und entsprechende Lösungsansätze vorgestellt. Als Beispiel für eine “global scale” Applikation wurde das Verfahren in dem von der NASA entwickelten Geoinformationssystem *World Wind* integriert.

Darüber hinaus wird in dieser Arbeit ein erweiterter Ansatz zur Verwendung von *early-Z* Beschleunigungsverfahren in Shadern präsentiert, für die dies derzeit nicht möglich ist. Darüber hinaus präsentieren wir ein zweistufiges Verfahren zur Gültigkeit von dynamisch erzeugten Impostors und Methoden zur zeitkritischen gemeinsamen Darstellung von diskreten und kontinuierlichen LOD-Modellen.

Acknowledgements

This thesis would not have been possible without the valuable input and feedback of my thesis advisor Helwig Hauser, who greatly contributed to my formation as a researcher.

Also, my colleagues both at the VRVis Research Center and at the Computer Graphics group of the Vienna University of Technology were immensely helpful, most notably Stefan Jeschke, Michael Wimmer, Gerd Hesina, Robert Tobler, Andreas Reichinger, Eike Umlauf and Toni Fuhrmann. They provided much of the knowledge and code libraries that I could build upon. I would also like to thank my students Wolfgang Berger and Bernd Leitner for their assistance with implementing my ideas, often from rather vague specifications.

Many other research groups provided feedback, code and data that were very helpful during the development of this work. Sorin Popescu from Texas A&M University provided interesting details on LIDAR data (and sample files!). I would also like to thank Oliver Deussen for several inspiring conversations. Gabriel Seitlinger from the Hohe Tauern National Park provided high resolution large scale data sets that became a key component (and a major stress test for many components) of this thesis and also gave much appreciated feedback on the results. The NASA Learning Technologies group originally created World Wind, which became one of the key frameworks of this thesis. I am indebted to them and the open source developers involved in the effort for supplying a great foundation for my work.

Finally, my friends and family deserve much praise for their assistance and understanding; they all have seen painfully little of me in the last months. In particular, Ivonne Lange and Nicole Kolar both did an outstanding job at keeping me (marginally) sane in rather stressful times.

kiwis don’t fly.

CONTENTS

CHAPTER 1

INTRODUCTION

> The beginning of knowledge is the discovery of something we do not understand.
>
> Frank Herbert

Computer graphics - the science of producing images with the help of computers - has evolved significantly since its beginnings, and continues to do so at a remarkable pace. It has become a very broad field and encompasses, for example, applications as diverse as scientific visualization, art and interactive computer games.

I this thesis, the term computer graphics is generally used for a more specific area – *interactive* (or real-time) computer graphics: the computer generated images are not static, but part of an interactive process such as navigating through a scene. Furthermore, (attempted) *realism* is typically implied in this context, such that the produced image resembles reality as closely as technologically possible. If an abstract, more symbolic depiction is desired the term *non-photorealistic* (NPR) rendering is usually used explicitly (but rarely implied; if an image is not explicitly described as an NPR rendition, it should be seen as the author's attempt at realism).

Today, even commodity personal computers come with graphics processing units (GPUs) that are easily as powerful as the graphics supercomputers of a decade ago[1].

[1] SGI Onyx2 RealityMonster (ca. 1997): 7.2GPixels/sec fill rate [Map02];

This massive processing power has lead to the emergence of new graphically intensive applications. Of course computer games have always been at the cutting edge of technology, but more 'serious' applications such as route planners and even operating systems now exploit the capabilities of modern graphics hardware.

1.1 Motivation

The availability of graphics processing power has lead to many new applications. Games continue to be a very important driving force in computer graphics today, but new applications such as geographic information tools, interactive maps and three-dimensional design applications are becoming more common.

The development of three-dimensional geographic information tools has been especially rapid, and has benefited from the increasing graphics power just as much as from the common availability of the high bandwidth required to access satellite imagery and other data, and of course from the growing CPU power for data processing.

It is therefore now possible to view the entire globe interactively, often at stunning detail. The satellite imagery used in Google Earth[2], NASA World Wind[3] and Microsoft's Virtual Earth[4] covers the entire globe at resolutions between 15m (Landsat 7 data) and up to 0.25m (USGS Urban Area Orthoimagery). Such extremely high detail enables the user to navigate much closer to the surface than previously possible, and it is quite easy to identify individual buildings and small scale features.

However, as soon as the view is tilted to a larger perspective, the flatness of the displayed data immediately becomes apparent (see Figure 1.1). Although satellite imagery is available at very high resolutions, elevation data is only processed at a much lower resolution: the widely used SRTM data has a resolution of 30-90 meters, and typical high-resolution elevation models are approximately 10 meters. On the other hand, satellite and aerial imagery is routinely available in resolutions of up to a few centimeters.

Recent versions of Google Earth do provide (rudimentary) 3D models in urban areas, but this is limited to areas where detailed information on buildings is available. Research projects are under way to automatically

NVIDIA GeForce 8800GTX GPU (2007): 13.8GPixels/sec [Pol06]

[2] http://earth.google.com/

[3] http://worldwind.arc.nasa.gov/

[4] http://local.live.com/

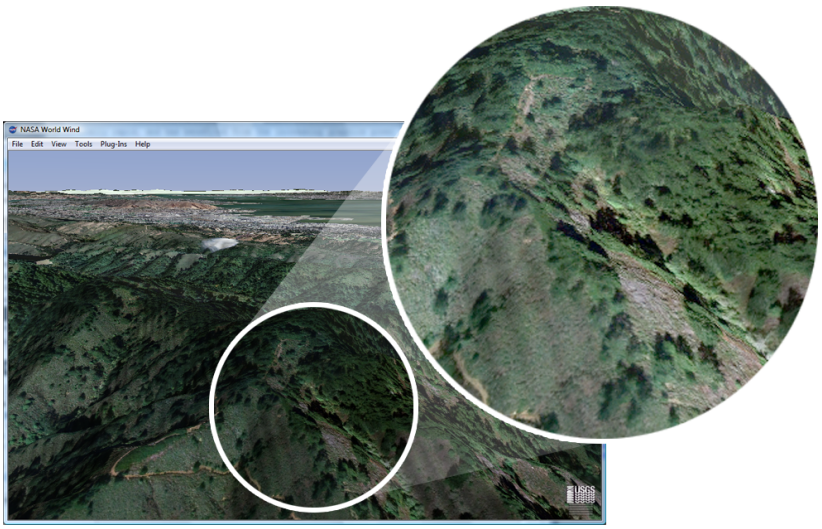


Figure 1.1: Orthophotographic maps projected onto a terrain model become apparently 'flat' when viewed at low angles.

derive such information on a larger scale [ZKHK03], but even these will be reasonably limited in scale.

The goal of this thesis, then, is to explore ways in which another crucial ingredient can be added to interactive 3D mapping applications: Vegetation.

1.2 Problems and Challenges

Since vegetation is a very broad term, this work specifically focuses on the visualization of landscapes and associated vegetation. This includes individual trees in urban environments as well as large large forested areas.

Rendering vegetation is a substantially different task to other, more 'geometric' objects. Many algorithms that work very well with 'technical' objects fail when it comes to organic structures, or modify them in an unrealistic way. Ultimately, computer graphics deals with (flat) polygons, and while we are used to flat surfaces in our everyday life, the human perception would immediately identify a hexagonal tree trunk as unnatural and odd.

Given that a large tree easily has hundreds of thousands of leaves, even a single detailed tree sufficiently detailed for closer inspection easily

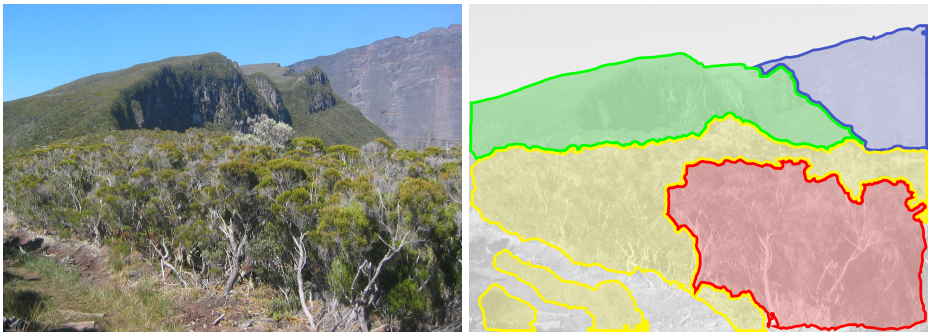


Figure 1.2: Comparison of applicable rendering function against view distance and visible detail. red: full geometry; yellow: impostors; green: shaders; blue: textures.

amounts to millions of polygons. Despite the huge performance of current graphics hardware, this is not a trivial task to render interactively. While a lot of research has been done on displaying individual plants and small groups (with very good results), these methods typically do not scale well to the thousands or even millions of trees present in a large forest.

Memory Requirements and Object Management

Even *managing* such large numbers of objects is problematic, since a forest contains 10.000-30.000 trees per square kilometer (assuming approximately $30m^2$ per tree).

To explicitly model a large forest with trees sufficiently detailed to inspect individual leaves, the required polygon count would grow well into the billions. This is not only far too much to be rendered directly, it also exceeds the memory capacity of most computer systems.

Therefore, a viable representation needs to be based on a lightweight representation that can be used either directly for rendering or as a basis for the generation of detailed geometric models (or, ideally, both).

Rendering Quality and Speed

Figure 1.2 compares various methods for vegetation rendering and their applicable view distance: some methods exist for rendering close objects, but they are only applicable to few objects and are therefore not suitable for more distant parts of the scene. Similarly, simplified objects that work well at a distance do not produce the detail required for near field ren-

dering, and their performance also deteriorates when a great number of objects must be displayed.

Ultimately, at the far end, features become small enough that a textured terrain is sufficient. To provide a seamless transition to simple textured meshes, algorithms are needed that are capable of rendering very large landscapes at a better quality than just a textured mesh. They need not work very well for medium or near distance (because there are already other algorithms that can be used there), but ideally it is possible to transition smoothly to other representations.

Integration

As is evident from Figure 1.2, any approach to rendering is ideally suited for a specific view range. It is therefore preferable to identify or design methods that are in some way 'compatible' with other approaches, in that a smooth transition from one method to the other is possible.

This is especially true for interactively rendering many objects. If two different acceleration methods are used - one for the near field, and one for the far field - and each method renders a perfect image of the original object, then a static image will be essentially flawless. However, for interactive exploration there must be a means to go from one acceleration method to the other. If this transition causes a slight blurring, this may not be immediately visible for a single object. But if an entire forest is displayed with such an approach, the blurred region of objects inside the 'transition zone' may become evident during navigation.

1.3 Contributions

This thesis addresses the presented challenges with a number of new approaches and improvements of existing techniques. Although the main body of the work focuses on landscape rendering, Part I also presents a number of methods for individual objects.

Near Field Vegetation Rendering

New methods for rendering plants and small scale vegetation at medium to close distances are presented in Part I. These chapters focus on point- and image-based rendering techniques suitable for accelerating the rendering of individual objects.

Far Field Vegetation Rendering

Part II focuses on the 'far field', rendering entire landscapes through the use of GPU based techniques. Chapter 6 introduces a new algorithm for augmenting existing landscape renderers using GPU based methods. It also demonstrates how the presented algorithms and future developments can be integrated in NASA World Wind, a large scale geospatial viewing application.

Impostor Validity, Levels of Detail and Optimizations

In a more general context, detailed discussions of impostor validity and level of detail rendering techniques have been included in Part III (Chapters 7 and 8). Finally, a detailed technical analysis of early-Z behavior of displacement mapping shaders, and a proposed hardware based optimization that exploits the specific characteristics of these shaders, is presented in Chapter 9.

CHAPTER 2

THE STATE OF THE ART IN REAL-TIME RENDERING OF VEGETATION

> Computers in the future may have only 1.000 vacuum tubes and perhaps weigh only 1 1/2 tons.
>
> Popular Mechanics, March 1949, p. 258

This chapter presents an overview of the current state of the art in real-time rendering of vegetation. Given our goal to increase the overall realism of computer generated scenes, the selected algorithms focus on *realistic* rendering of trees. There is of course research on achieving non-realistic effects such as sketch outlines, hatched shading, etc., but these methods are outside the scope of this summary (see, for example, Deussen's book for a good overview [Deu03]). Similar restrictions apply to approaches that cannot currently be rendered at interactive frame rates, or only with clusters of PCs, such as Dietrich's terrain guided renderer [DMS06], although we do include methods that may be feasible in the near future. In addition to presenting each algorithm, we will summarize the advantages and drawbacks in a separate section.

2.1 Overview

Vegetation can be rendered using a very large number of generic and specific algorithms. It is therefore necessary to impose certain restrictions. For example, since interaction is an important aspect of real-time computer graphics, we will disregard methods that put severe constraints on the viewpoints (or even keep it fixed) such as QuickTimeVR [Che95]. During the research of relevant papers, all publications were categorized according to three dimensions:

Specificity Is the method universally usable (eg. z-Buffer, Hierarchical Occlusion Maps, ...) or specific to vegetation rendering? This also includes how closely the rendering method is tied to a specific plant model (as opposed to only dealing with the resulting geometry).

Modeling vs. Rendering Does the publication deal mainly with modeling (ie. growth behavior, lighting simulation, ...) or rendering?

Realistic vs. Real-time Rendering Is the presented algorithm suitable for real-time rendering? Does it attempt to produce the best possible image, or does it trade quality for better performance?

Generally speaking, our main interest are methods that fall into the latter end of each of these scales. Of course this categorization is not always clearly possible, and there are certain interactions between the categories. For example, some rendering algorithms require a specific approach to modeling the object.

However, these dimensions allow us to somewhat narrow the subject. For example, we have chosen to omit 'pure' modeling papers that do not deal with rendering at all. Also, nonspecific acceleration methods such as the Hierarchical Z-Buffer [GK93] are typically well known in the computer graphics community and are therefore outside the scope of this report. We may include short references to these algorithms where applicable, but for a more general overview of rendering acceleration methods, we recommend eg. Moller's book [MH02]. Also, Deussen has published an excellent (although German) book on computer generated plants that presents a thorough overview of many aspects, including modeling, offline, real-time and non-photorealistic rendering of plants [Deu03].

Furthermore, Boudon et al. also published a detailed overview on the subject, which includes a discussion of various modeling as well as rendering approaches for trees [BMG06].

In the following sections, our categorization of the rendering algorithms is based on the main rendering primitive, and the presentation is approximately chronological. This distinction is not always clear, and hybrid methods will be described in the section of their main contribution. For example, practically all image based rendering methods require geometry to render the derived images (which may be a single plane, but can be significantly more complex).

2.2 Polygon-Based Algorithms

Polygonal and especially triangular models have traditionally been the predominant rendering primitive in computer graphics. Recent developments in hardware acceleration have also focused on triangular data; therefore this rendering primitive has a certain advantage when it comes to real-time rendering.

Many polygonal rendering methods for vegetation apply generic acceleration methods, such as triangle strips, to speed up rendering. Numerous other algorithms for complex polygonal data can also be applied [GK93, WFP+01, SS01]. Furthermore, see Moller's book [MH02] for an overview of such methods.

Since the foliage represents the majority of a tree's geometric complexity, groups of leaves or small branches can be approximated as a single, texture mapped polygon [Int02].

Simulation of Natural Scenes Using Textured Quadric Surfaces

An early approach to rendering natural scenes was introduced by Gardner in 1984 [Gar84], in which he discussed the inevitable tradeoff between rendering performance and image quality.

In this paper, textured quadric surfaces are presented as a suitable primitive. At the time of writing, scan conversion of primitives was performed in software, and quadrics provided the simplest possible curved surfaces without having to resort to piecewise linear approximations. Quadrics could be scan converted directly, without the additional overhead of converting them to polygonal approximations and rasterizing these. To allow for a greater variety of shapes, clipping planes may be used to truncate the quadrics. Rendering is performed entirely in software, and is based on the analysis of bound-

Figure 2.1: *Simulation of Natural Scenes using Textured Quadric Surfaces* Illustration from Gardner [Gar84].

ary curves to produce spans of constant visibility. Textures are created on the fly through an adapted Fourier expansion; the required parameters were found through manual experimentation.

Although far from interactive at the time of writing (up to several minutes per frame at 640x480 pixels), such a renderer could be expected to run at adequate frame rates on current hardware.

Real-Time Design and Animation of Fractal Plants and Trees

An early algorithm for real-time rendering of fractal plants and trees has been introduced by Oppenheimer in 1986 [Opp86]. Due to the early time of their publication, their paper discusses many aspects of fractal modeling and self similarity that can now be presumed.

Natural trees are not strictly self similar; there is always some deviation in the symmetry due to environmental differences. The greater the deviation of the tree parameters, the more random and gnarled the tree will appear. The author defines the resulting tree as *statistically* self-similar to represent this fact. The fractal model used in this system introduces a way to control the variance of these parameters.

The resulting models are rendered using bump mapped polygonal prisms. The branches emanating from a limb simply interpenetrate the limb; several prisms are combined to approximate curvilinear shapes. The resulting geometry is compiled into display lists to accelerate rendering.

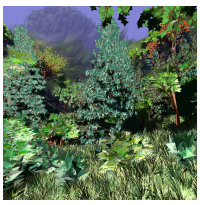
To further enhance the visual impression, a bump mapped texture is applied to the tree limbs. This texture is procedurally generated by adding fractal noise to a ramp, and then passing the result through a sawtooth function. The function used in the original paper also wraps seamlessly in u and v.

The rendering of leaves is not discussed in the paper; however sample screenshots in the paper by Oppenheimer do feature leaves and blossoms, so the algorithm can be adapted accordingly.

At the time of publication, the rendering of complex tree images could take several hours to render; obviously trying to design a desired tree at this rate is not very effective. The display list was therefore split into static geometry display lists and (variable) transformation matrices. These transformations could then be adjusted very efficiently to reflect changed parameters.

Obviously, a similar approach could be used to support dynamic effects, such as wind or other forces.

Multiresolution Rendering of Complex Botanical Scenes



In their 1997 publication, Marshall and Fussel have presented a system for rendering very large collections of randomly parameterized plants [MFC97]. Their multiresolution rendering system compiles plant models into a hierarchical volume approximation based on irregular tetrahedra. This partitioning creates a binary tree similar to BSP trees, which can be traversed quite efficiently.

The plant model used by Marshall and Fussel allows plant information to be stored at various levels of detail and memory usage. The generation of actual geometry for any subvolume can be delayed until it is needed. This drastically reduces memory consumption and initialization time, as the binary tree does not need to be built fully.

This compilation progress begins with a full tetrahedral volume as a first approximation to an object, which is then further refined recursively as needed to accommodate individual polygons. Depending on the exact intersection of a given polygon with a tetrahedral volume, the resulting subvolumes are typically not tetrahedra themselves. There is some

freedom in partitioning these into sub-tetrahedra; the method chosen by Marshall and Fussel chooses a subdivision depending on the aspect ratio of the resulting tetrahedra in order to avoid sliver subvolumes.

At runtime, this subdivision is performed depending on the view distance. For objects that are close to the viewer, explicit polygons are generated, while objects that are hidden or further away are rendered as groups of microsurfaces approximating the contents of the bounding tetrahedra.

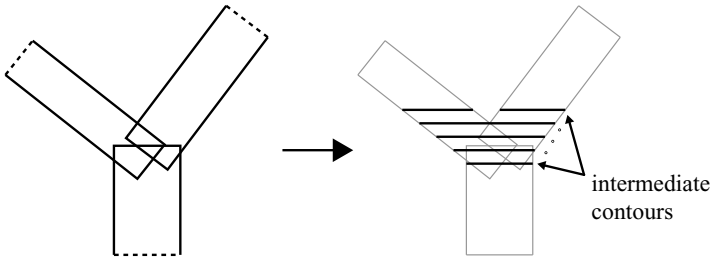
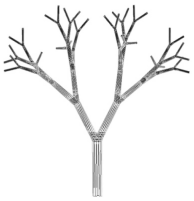


Figure 2.2: *Modeling of Branched Structures using a Single Polygonal Mesh*: intermediate contour generation. Illustration adapted from Lluch [LVF+01].

Modeling of Branched Structures using a Single Polygonal Mesh



Lluch et al. observe that one of the main issues of rendering polygonal trees is that many growth models produce disconnected meshes for each branch [LVF+01]; bifurcations are often simplified as the interpenetration of such meshes. If a single mesh could be obtained instead, this would facilitate the application of multiresolution and simplification methods.

The tree representation used in their proposal is based on sequences of elliptical (or circular) contours; a library created by the same research group is then used to obtain triangular meshes from two such contours.

However, bifurcations require special treatment, as they cannot be represented as elliptical structures. To handle these sections, they have developed an algorithm called *refinement by intervals*. Intermediate contours are generated at regular intervals over the branching section until

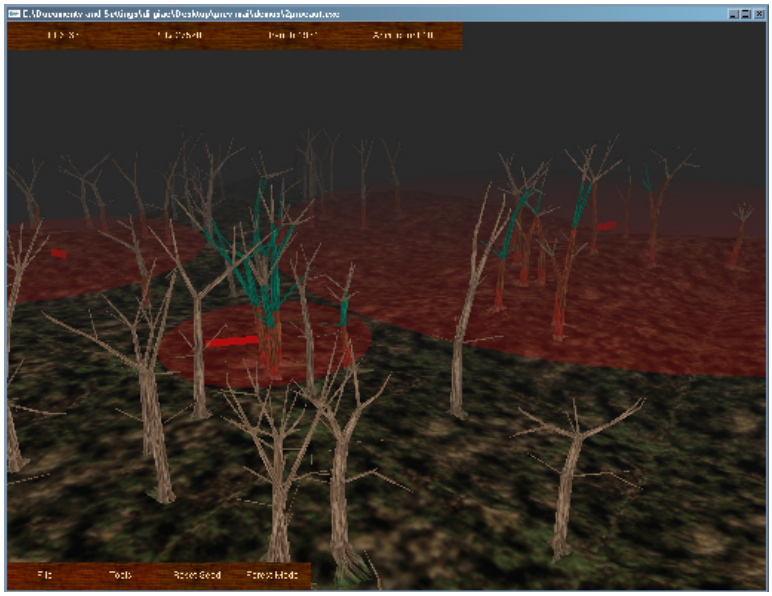
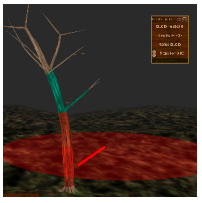


Figure 2.3: *An Interactive Forest* (Illustration from Giacomo [GCF01]. Red discs are wind influence objects; red branches are selected for animation; green ones transition between animated and static.

two separate elliptical sections have been reached. These intermediate contours can then be used to generate appropriate polygonal representations.

The authors note that even though refinement by intervals causes a significant increase in polygon count, the resulting continuous mesh can easily be reduced by a decimation algorithm.

An Interactive Forest

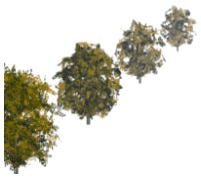


An approach that focuses more on interaction and physically based and procedural animation was presented by Giacomo et al. [GCF01]. Their system uses various heuristics to approximate wind forces and simplify calculations, and allows for levels of detail for calculation and rendering. For a given level of detail, branches below an associated threshold are considered solid and not included in the calculations.

Procedural animation is used for wind force estimation, and a physically based can be seamlessly added to account for user interaction. Figure 2.2 displays a sample scene with several wind influence objects.

The tree model is comprised of a topological representation *skeleton nodes* and a mesh that defines the actual geometry. Animation calculations are performed on the skeleton nodes and transferred to the mesh for rendering. Leaf geometry is not considered.

View-Dependent Multiresolution Model for Foliage



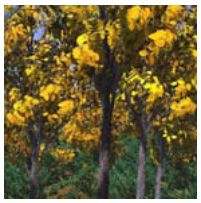
Since the leaf canopy of trees contributes a huge number of polygons in tree models, Remolar et al. have proposed a simplification method that specifically targets foliage [RCRB03, RCB+02].

Traditional geometry simplification methods are typically not applicable, since leaves consist of many individual polygons. Therefore, topology preserving algorithms will not succeed, and non-preserving methods typically introduce a significant change to the overall appearance.

The algorithm described in this paper succeeds in diminishing the number of polygons in the crown, while maintaining overall appearance. This is achieved by introducing a new method, the leaf collapse: Two leaves are replaced with a new one that preserves an area similar to that of the collapsed leaves.

In a preprocessing step, a multiresolution model is created from a sequence of leaf collapses. The resulting data structure is therefore created bottom-up as a binary tree, with a polygonal representation of individual leaves (the highest resolution) as the leaves, and the root nodes being the polygons required for a minimum representation. Therefore, the resulting data structure is a 'forest' of binary trees, ie. a list of disconnected trees (see Figure 2.2).

Multiresolution plant models with complex organs



A very similar algorithm has been proposed by Zhang et al. to allow a recursive simplification of more complex shapes [ZBJ06]. To build their *Hierarchical Union of [Plant] Organs* in a series of preprocessing steps, plant features are first grouped according to leaf phyllotaxis, flower anthotaxis and petal distribution. A hierarchical simplification is then performed by progressively merging two pairs of polygons within each cluster until a final, *representative quadrilateral* is found for each group. Finally, the process is repeated for these representative quadrilaterals.

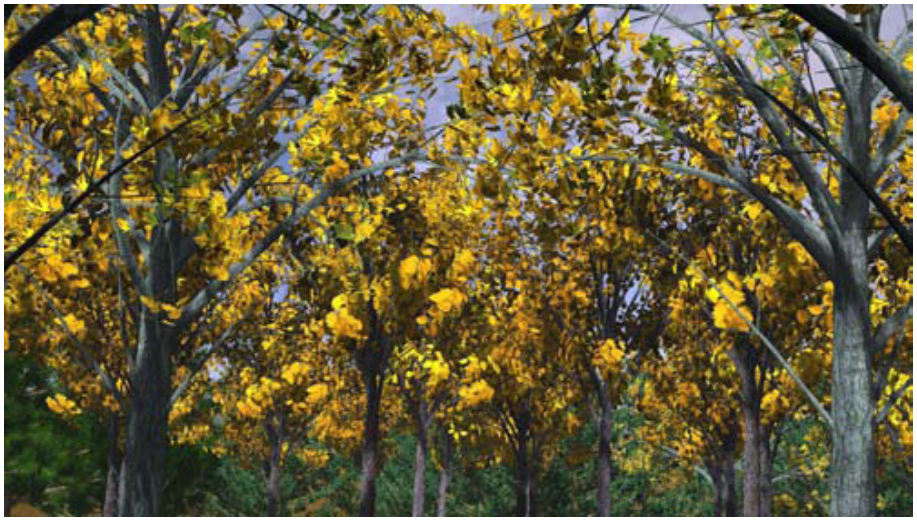
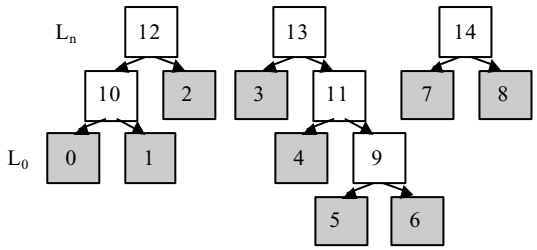


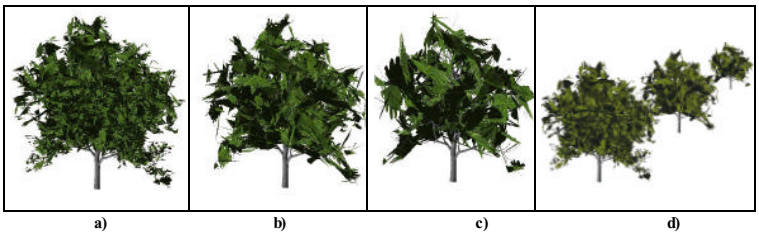
Figure 2.4: *Multiresolution plant models with complex organs* (Illustration from Zhang [ZBJ06]. Virtual garden consisting of 83 trees; original model has 9.5 million triangles, the rendered multiresolution model has 1.2 million. Rendering performance is 0.5 - 10fps.

At runtime, the desired pixel error is converted to a spatial error, which in turn is used to select an adequate level within this hierarchy.

Rendering of branches is not discussed. The authors refer to another paper 'in submission' which does not appear to be publicly available at the time of writing.



(a) Example of the data structure used to represent foliage as a 'forest' of binary trees. The top level nodes (12,13,14) represent the lowest detail, while the leaf nodes (grey) contain the highest resolution



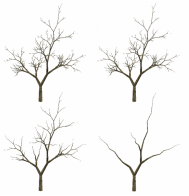
(b) Different uniform levels of detail of the same tree: (a) 13,420 polygons, (b) 1,558 polygons, (c) 472 polygons. In (d), they are shown depending on the distance to the viewer.

(c) View-dependent levels of detail: Interest area is determined by a plane; 18.406 polygons.

Figure 2.5: *View-Dependent Multiresolution Model for Foliage* (illustrations from Remolar [RCRB03].

Figure 2.6: *Procedural Multiresolution for Plant and Tree Rendering*: A tree generated at four different levels of detail (using 3252, 2103, 872 and 172 polygons; illustration from Lluch [LCV03]))

Procedural Multiresolution for Plant and Tree Rendering



Javier Lluch et al. propose another multiresolution method based on parametric L-systems [LCV03]. Their algorithm is based on a metric that quantifies the visual relevance of the branches of a tree. This paper focuses on the branch structures; leaves are not considered.

The level of detail algorithm operates on the underlying L-system, thus avoiding the generation of geometry that will not be rendered. To capture the relevance of individual chains generated from the L-system, an intermediate *weighted tree* data structure is created. From this data structure, the *multiresolution chain* can then be generated. In addition to the output of the L-system itself, the multiresolution chain supports two new instructions: *SAVE(id)* and *RESTORE(id)*. These can be used to store and restore the current state for some unique identifier.

This allows the weighted tree to be stored as a reordered chain that is sorted by the individual node weights. Higher weights (more important nodes) are stored first; finer LODs can be added to any point of the tree through the *RESTORE(id)* instructions.

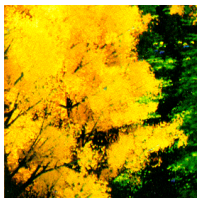
2.3 Point Based Algorithms

Levoy and Whitted have introduced points as an efficient display primitive in 1985 [LW85], and at the same time they were first used to render vegetation [RB85]. Point primitives and vegetation therefore share an interesting historical connection.

Until recently, further research on point based rendering has been somewhat sporadic. However, recent publications have presented some interesting general purpose algorithms [WFadH00, PZvBG00, ZSBP02],

and there have also been a number of more specialized methods that will be presented in this section.

Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems



In his 1985 publication, William T. Reeves describes a stochastic modeling system that has been used to render forest images [RB85]. At the time of publication it was clearly not a real-time system (five to ten hours of rendering per image on a VAX 11/750), however the performance of computer systems has increased dramatically in the past two decades.

Each tree is drawn as a set of particles, line segments and small circles, representing branches and leaves respectively. These particles are generated from a recursive representation in a preprocessing step.

To model self-shadowing, a probabilistic model based on the particle's position and orientation has been implemented. External shadows from other trees are also approximated through a probabilistic function.

Creation and Rendering of Realistic Trees

Weber and Penn's classic paper on modeling and rendering realistic trees also includes point based rendering [WP95]. Although their publication focuses mainly on the procedural modeling aspect, they make use of point and line primitives for leaves and branches, respectively. The representation created from their model is not explicitly converted to geometry, but interpreted at runtime.

When viewing the object at a close distance, full-resolution polygonal geometry is created. For larger distances this representation is changed to lines for stems and twigs, and points to render leaves. Heuristic equations are used to transition between these representations.

Interactive Visualization of Complex Plant Ecosystems

Deussen et al. have presented a system for interactively rendering large plant populations by using point and line primitives [DCSD02]. A hierarchical scene data structure is used to support a coarser representation of distant regions. Additionally, a visual importance factor can be manually assigned to objects, which allows certain objects to be rendered at a higher quality than others.

Point and line representations of the polygonal input data is generated semi-automatically. The user needs to choose the primitive to be used for each part of the plant and possibly assign the importance factor if required. Point and line data is then generated automatically and stored in display lists. Point and line representations and the respective polygonal data are reordered randomly (but in the same order for point primitive and polygons) to avoid popping artifacts. Through the random reordering, switching part of an object from polygonal to point or line representation will not be localized to some area of the object, but is distributed over the entire model (see Figure 2.3).

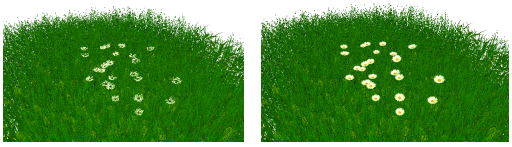
Rendering point data is performed by estimating the number of points required for a faithful representation (ie. no holes and correct coverage). Blending between polygonal and point data is supported by rendering only part of the polygonal display list, and displaying the remainder as point data. Since both lists are in the same order, the entire model will be covered. For line data, the area covered by the entire line set is calculated and compared to the triangle set it represents. Rendering then proceeds similar to the point data.

Sequential Point Trees

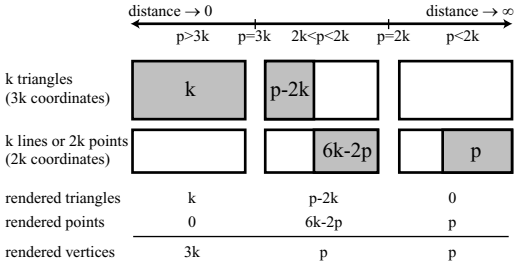
The Sequential Point Trees proposed by Dachsbacher et al. [DVS03] also provide a hybrid point and polygon based system, but using a different approach for selecting the primitives to be rendered.

Point samples are first generated regularly on the model; they are then grouped hierarchically according to a geometrical error metric that describes how well the parent disc approximates its child nodes. The authors note that with the right parameters their approach is equivalent to the QSplat algorithm [RL00].

After building this hierarchy, the nodes are converted to a sequential representation by sorting all nodes by the disc radius r_{max}. This allows



(a) Assigning importance factors. left: default values. right: setting a higher importance factor to the daisies, causing them to be rendered as polygons.



(b) Blending polygonal and point based rendering based on the available rendering budget.

Figure 2.7: *Interactive Visualization of Complex Plant Ecosystems* (illustration from Deussen [DCSD02].

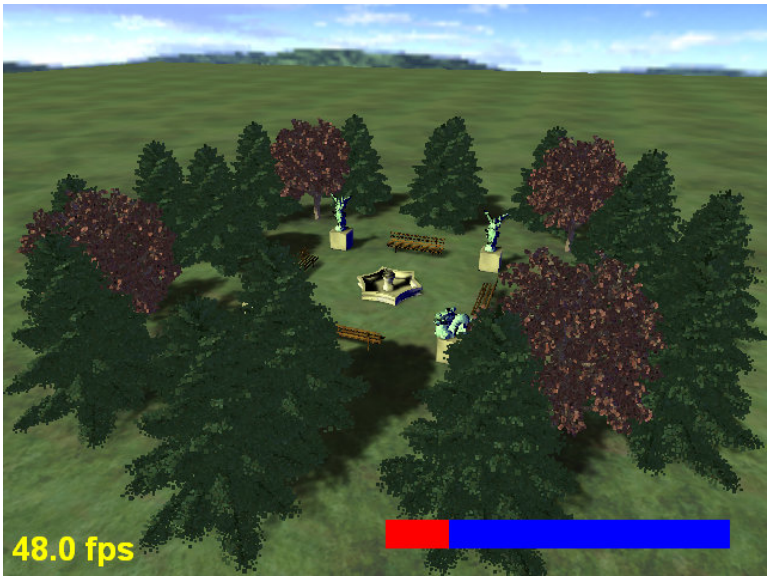


Figure 2.8: Sample scene rendered using *Sequential Point Trees* [DVS03]

all further calculations to be restricted to a prefix of the list determined by the view distance.

For a given view distance r, the nodes that actually need to be rendered vary between radii $min\{r\}$ and $max\{r\}$. This leads to the following rendering algorithm: All points up to r_{max} are sent to the vertex shader. This is very efficient, as a contiguous stream of vertices is sent. In the vertex shader, each point is then tested against the $[min\{r\}, max\{r\}]$ bounds and either passed to the fragment shader for rasterization or moved to infinity (and effectively culled).

The authors note that their algorithm can be extended to also support triangular primitives by regarding the triangle's longest edge as its radius. However, due to the necessary resorting of triangles according to their radius, triangle strips are torn apart and must be rendered as individual triangles. As a result, this approach bears some resemblance to the Randomized Z-Buffer introduced by Wand et al. [WFP+01].

Figure 2.9: Sample scene rendered using *Deferred Splatting* [GBP04]. The scene contains approximately 2300 visible trees consisting of approximately 750k polygons each. It is rendered at 11fps.

Deferred Splatting

Guennebaud et al. have developed Deferred Splatting, an algorithm that exploits various culling and level of detail methods to reduce the number of points to be splatted for highly complex geometry [GBP04]. Their technique is based on GPU-based EWA splatting [GP03], which uses a multi-pass approach to filter visible surface elements (*surfels*). At first, a *visibility splatting* pass is used to pre-fill the z-buffer; then the *EWA splatting* pass accumulates the filtered color values. Finally, a normalization pass divides the color values by the number of contributing splats, resulting in the final color output.

This method has been extended to exploit the coherence between the several passes as well as between consecutive frames. A low level point selection pass is introduced after visibility splatting: surfels are rendered with unique identifiers as color values; the resulting image is then read back in and used as a selection for the EWA splatting pass as well as the visibility pass of the next frame, thus effectively exploiting temporal coherence. Potential "holes" due to disocclusions between consecutive frames are filled by another visibility splatting pass only with surfels that are potentially visible in the current, but hidden in the previous pass. For efficiency, this decision is performed on groups of surfels rather than individually.

Point-Based Rendering of Trees

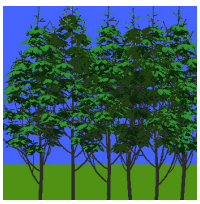
Gilet et al. have presented a hybrid point and polygon-based rendering approach that is based on a regular spatial subdivision [GMN05]. This approach is quite similar to Sequential Point Trees [DVS03] with an additional hierarchical subdivision. A block of vegetation (which can be either a single tree or a group) is subdivided into smaller cells. For each cell, a hierarchical clustering algorithm creates a binary tree representation of point approximations and triangles.

At runtime, cells are visited individually. In addition to the projected size of a cell, its view dependent position within the block is used to determine its level of detail for rendering. It is assumed that front cells mask those behind, and therefore cells closer to the viewer need to be rendered at a higher level of detail than the (partially occluded) blocks in the rear.

2.4 Image Based Algorithms

Due to the nature of image based algorithms, their performance is typically independent of object complexity and controlled by the output resolution alone. This makes them quite suitable for complex objects such as vegetation. Most image-based rendering methods have been designed as general purpose algorithms [CW93, DMBF96, Sch95, MB95, LH96, Sch98, GGSC96, SLS+96, DSV98, DSSD99], and no experimental results with vegetation are available. However many of these ideas have been transferred to more specific algorithms, and some even into commercial products [Bio03].

Rendering Trees from Precomputed Z-Buffer Views



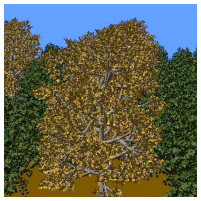
An algorithm proposed by Max in 1995 uses precomputed z-buffer views to approximate arbitrary viewpoints [MO95]. Their approach is similar to that of Chen and Williams [CW93], but with a few enhancements.

Precomputed views are acquired through parallel projection from a number of viewpoints generated through a simple longitude/latitude sphere partitioning scheme.

Since there is little coherence between leaves in a tree, the reconstruction for an arbitrary viewpoint is performed on a per-pixel basis. This typically leaves some pixels undefined where no information can be extracted from the available views. The authors have chosen to implement multiple z-buffer layers to reduce these artifacts.

Dynamic shading and shadowing is supported by storing compressed normal vector and material information for each pixel of the precomputed views. During the shading post-process, these values can be used to compute diffuse and Phong shading. Shadows can be found by reconstructing a z-buffer view for the light source and testing output pixels against this buffer. Since normal vector and material information is available, shading can be applied in a post-processing step once for each output pixel instead of each time pixel data is written to the output frame buffer.

Hierarchical Image-Based Rendering using Texture Mapping Hardware

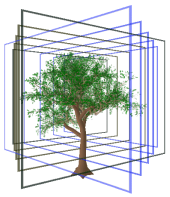


Max et al. combine a hierarchical tree model with an image based rendering method that supports hardware acceleration [MDK99]. Although their rendering times were not real-time at the time of publication, it may be feasible with current graphics hardware.

Their approach precomputes multi-layered depth images containing color and normal information using standard z-buffer hardware. Six orthogonal views are calculated for each level in the hierarchy. Multiple depth layers are computed by using hardware z clipping to partition the object into several slabs. To avoid excessive numbers of textures, the number of inequivalent sub-objects in the hierarchy must be limited.

Based on the object distance from the viewpoint, the hierarchical description is traversed until either the current level is a sufficient approximation, or actual polygons need to be generated. During rendering, the hierarchy is first traversed and a list of reprojection matrices accumulated for each of the textures. All visible instances of a texture are then rendered in order, thus significantly reducing texture swapping. Reprojection and rendering the depth images is performed similar to the method presented by Schaufler [Sch98].

Interactive Vegetation Rendering with Slicing and Blending



Jakulin combines traditional polygonal geometry rendering for the trunk and limbs of a tree with an image-based rendering system for the crown [Jak00]. The crown is rendered using multiple parallel layers (*slices*). The group of slices for a specific view direction is called a *slicing*.

During preprocessing, several sets of these slices are created from various viewpoints. For each slicing, the primitives (ie. individual leaves) are assigned to the closest slice (see Figure 2.4). Each slice is then rendered to an individual texture.

At runtime, the two slicings closest to the actual view direction are rendered simultaneously, using transparency and blending for transitions as the view direction changes.

The goal of this algorithm was to accommodate architectural walkthroughs and driving simulations, so viewing trees directly from above

or below the tree is not supported and leads to severe artifacts. Therefore, all slices are perpendicular to the ground plane, and blending between two sets provides sufficient coverage. However, this is not an inherent limitation of the method, and the authors speculate that blending three slicings would be appropriate for arbitrary viewpoints.

Image-Based Multiresolution Modeling for Real-Time Foliage Rendering

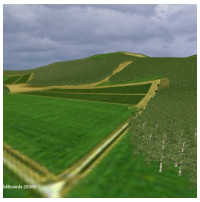
Lluch et al. have proposed an interesting image-based rendering approach for rendering foliage [LCV04]. Based on an L-system tree model, they create a hierarchical data structure that includes bounding boxes at each level.

All leaves within the bounding box are then projected to each of the bounding box planes, and stored as impostor textures. To increase visual realism, impostors are not only created for the usual $x = \pm 1$, $y = \pm 1$ and $z = \pm 1$ planes, but also for the main diagonals ($x \pm y = 0$ etc.). The bounding box, and therefore also its impostors, are oriented in the local coordinate system for each level.

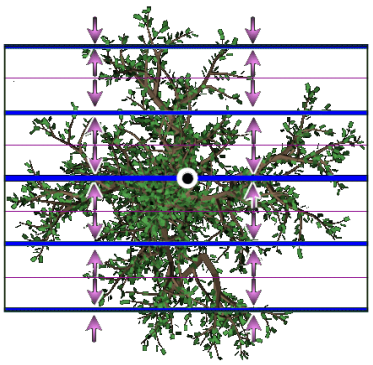
Not all possible levels of recursion are visited during impostor creation. To reduce the spatial cost of the model representation, a threshold based on the relative size of the branch (compared to the entire tree) can be set.

At run time, the hierarchy is traversed until a suitable (distance based) level of detail is reached. Appropriate viewing distances for each level of the hierarchy are precomputed for better performance. If no suitable impostor is available, the original (polygonal) geometry is used.

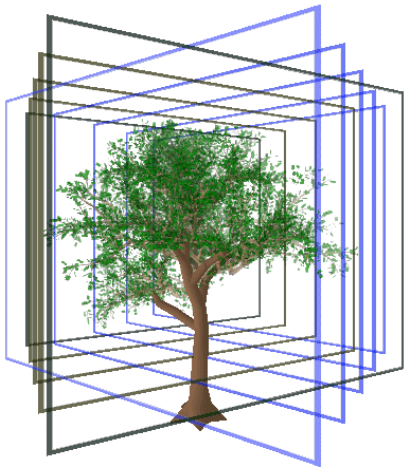
Drop and Resize of Billboards



Halper [Hal01] observes that when rendering large numbers of trees, the far field can be rendered similar to Remolar's multiresolution foliage [RCRB03]. In his case, trees are represented by billboards, and for the far field some of these billboards are dynamically dropped and the remainder resized accordingly. The author notes that although some artifacts are visible depending on the amount of simplification, the resulting image quality serves ell for interactive purposes.

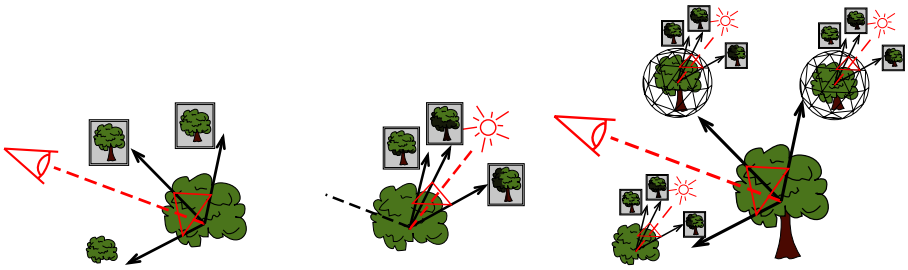


(a) Each primitive of the crown is assigned to the closest slice.



(b) Multiple slicings are blended to create a solid-looking rendering. Rectangular frames have been added to slice textures to aid visualization. Both slicings have a discrepancy angle of about 30 degrees.

Figure 2.10: *Interactive Vegetation Rendering with Slicing and Blending* (illustrations from Jakulin [Jak00].

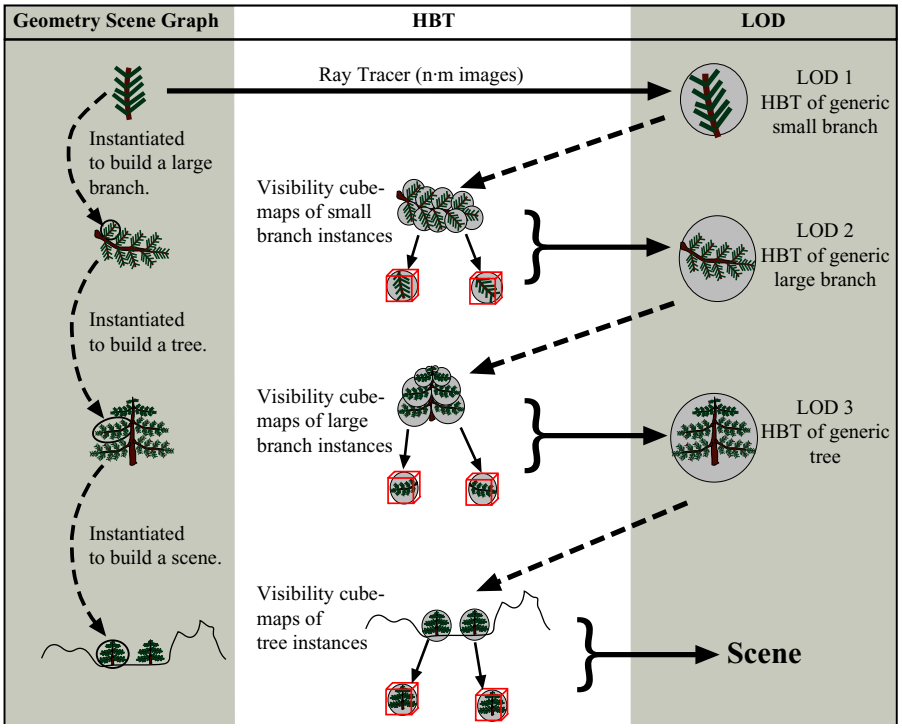


(a) A billboard is reconstructed from a given view direction by combining the 3 closest images stored in the sampled sphere of view directions.

(b) A billboard is reconstructed from a given light direction by combining the 3 closest images stored in the sampled sphere of light directions.

(c) The complete BTF allows the reconstruction of a billboard for given view and light directions by combining up to 9 stored images (in our implementation).

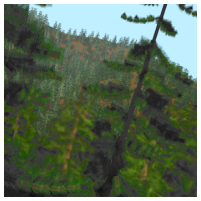
(a) Reconstructing billboards from view directions and light directions.



(b) Building the hierarchical data structures.

Figure 2.11: *Interactive Rendering of Trees with Shading and Shadowing* (illustrations from Meyer [MNP01]).

Interactive Rendering of Trees with Shading and Shadowing



The image-based rendering system proposed by Meyer et al. provides a framework for rendering trees with complex effects such as shading, self shadowing, and dynamic illumination [MNP01]. They combine a hierarchy of bidirectional textures (HBT) to provide billboards for each given observer and light directions with a hierarchical visibility structure for self-shadowing and cast shadows. This representation is efficient for trees, as it is hierarchical and instancing is used heavily.

Bidirectional texture functions (BTFs) are computed by associating a billboard representation with each pair of view and light directions (see Figure 2.4). Between 6 and 258 different view directions and light directions are used. During rendering, an arbitrary configuration can be approximated by interpolating 9 BTFs. These BTFs are associated to each level in the hierarchy either by creating a new, unique BTF or through instancing. During rendering, either BTF or the actual geometry is rendered depending on the distance.

To support dynamic illumination, approximate visibility cube-maps are computed for each level of the hierarchy. Since occlusion depends on the position within the hierarchy, separate cube-maps need to be generated for all instances. Shadowing can then be computed during rendering by traversing the hierarchy of visibility cubemaps. Casting shadows is supported through 'traditional' shadow maps by rendering from the light source.

Real-time Hardware Accelerated Rendering of Forests at Human Scale

This rendering method proposed by Szijártó and Koloszár uses a combination of geometry and image based (impostor) rendering for vegetation [SK04]. The trunks and branches are rendered as polygonal models, and impostors with 2.5D depth information are used to model the canopy.

The 2.5D data is used to provide correct depth in the near field, but the authors note that their approach requires modification of depth within the fragment shader, which incurs severe performance penalties on current graphics hardware. The authors therefore propose to switch to a

simple alpha blending approach for more distant objects, where they claim that resulting artifacts are visually indistinguishable, which is up to five times faster.

The paper also presents details on optimizing their method for the rendering of large forests, where individual impostors can be reused, data packed into a texture atlases, and rendered in correct order to minimize texture switching and similar state changes.

Rendering Forest Scenes in Real-Time

Decaudin and Neyret make use of volumetric textures to render forest scenes in real-time [DN04]. This is based on previous work by Neyret, where a similar volumetric approach was used for off-line rendering of landscapes.

The original landscape surface is replicated to a number of parallel slices, which are then rendered using volumetric textures. Aperiodic tiling is used to minimize repetition artifacts, and if the view direction is at grazing angles such that the slices are seen nearly edge-on, they are slanted towards the view direction.

In their approach, shading is precomputed and stored in the volumetric textures. This requires the tiles to be rendered at a specific orientation and also precludes detailed terrain specific shading (such as slope dependent), which is compensated by the authors by using an additional Lambertian term.

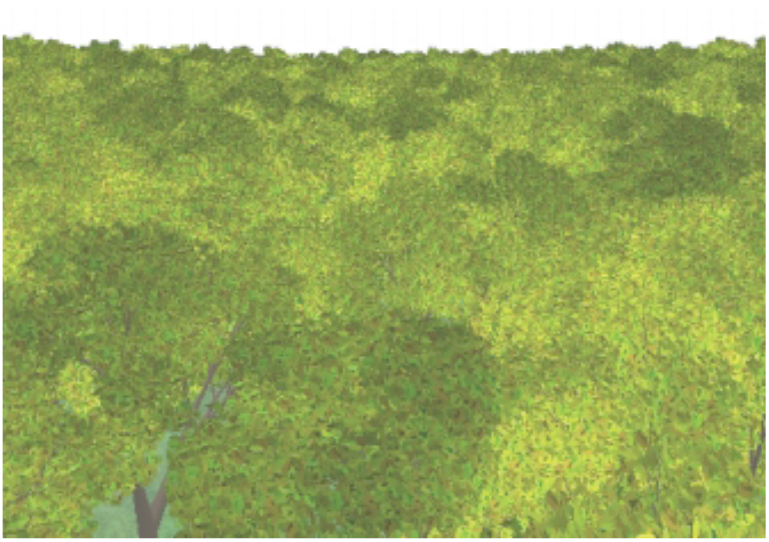
Note that this precomputation allows tiles only to be used in one specific orientation. Also, additional tiles must be used for border regions to avoid artifacts caused by trees overlapping the tile bounds.

Real-Time Rendering of Complex Photorealistic Landscapes using Hybrid Level-of-Detail Approaches

The use of *Billboard Clouds* was introduced by Decoret et al. [DDS03] and has been successfully adapted to rendering vegetation by Fuhrmann et al. [FUM05] as well as Colditz et al. [CCDH05]. We will discuss the latter approach, as it also includes realistic lighting.

In a similar paper, the authors extend the billboard cloud approach to use *shell textures* [DN04] for the far field [BCF+05]; this far field rendering method is further discussed in the Section on *Rendering Forest Scenes in Real-Time*.

(a) Inside the forest.



(b) Low altitude view.

Figure 2.12: *Real-time Hardware Accelerated Rendering of Forests at Human Scale* (illustrations from Szijarto [SK04]).

Figure 2.13: A forest scene of approximately 30000 trees, rendered in real time using the approach by Decaudin and Neyret [DN04]

Essentially, the billboard clouds approximate an arbitrary collection of individual polygons by a set of textured planes. In contrast to the original BBC approach by Decoret which explicitly avoided any topological information and considered triangles individually, Colditz et al. exploit the hierarchy information available in their plant models to find better local approximations. They also use k-means clustering instead of the original dual space approach.

Normal vector information is stored in a separate texture and used to approximate per-pixel illumination at runtime. Transitions between discrete levels of detail are performed through alpha blending.

Volumetric reconstruction and interactive rendering of trees from photographs

Reche-Martinez et al. observe that instead of explicitly modeling trees, photographs of existing trees could be used to create a volumetric representation which can then be used directly for rendering [RMMD04].

Photographs are captured such that alpha matting can be estimated to separate the tree from the background; this is performed through a semi-automatic process. Colored markers on the ground are used to calibrate the relative camera positions.

For volume rendering, the opacity of each voxel must be estimated. Color is treated separately. For each pixel of each input image, the alpha mask value is considered as the result of an accumulation of opacities in each voxel cell covered by the pixel. Using an absorption only

model, the transparency estimate of each voxel can be iteratively estimated. The final outcome of the estimation process is a recursive grid of low-frequency opacity values.

To capture fine detail of the tree, each volume cell is assigned a small (4x4 or 8x8 pixels) billboard for each camera direction. Color estimation for these billboards is non-trivial, since voxels may be partially occluded by other cells in the input images. The authors present an importance based heuristic that selects color values from the input images and removes selected values to avoid blurring artifacts.

Rendering itself is a straightforward process. The hierarchy of volume cells is traversed back to front, and the billboards rendered at each step: the two closest camera directions are selected and the associated billboards blended accordingly.

Real-time rendering of plant leaves

Wang et al. have presented a framework for rendering plant leaves with global illumination effects [WWD+05]. They use bidirectional reflectance functions (BRDFs) and bidirectional transmittance distribution functions (BTDFs) to capture the two main scattering behavior of plant leaves: rough surface scattering on the surface, and subsurface scattering inside the plant leaf. Their parametric BRDF and BTDF models are fitted to data measured from actual leaves.

Also, an extension to the precomputed radiance transfer (PRT) rendering algorithm is presented that also accounts for high-frequency sunlight. To achieve this, incident radiance is decomposed into direct and indirect components. Low-frequency indirect lighting is calculated through PRT. In a second pass, direct light is modeled through a light-visibility convolution, ie. a map that encodes the sun (modeled as a disc light source) as it is masked by components of the scene.

The authors note that other shadow mapping algorithms may be used, but state that it is hard to do so accurately due to the complicated self-occlusions involved in larger leaf assemblies.

2.5 Algorithms Summary and Conclusion

Table 2.1 summarizes the presented algorithms. It is certainly not possible to capture the intricate details of each method in one simple table, especially since factors such as quality and performance are difficult to

judge. Methods that did not run in real time a few years ago may be feasible with current hardware, and algorithms that were formerly limited eg. by texture memory size can now run with significantly higher quality.

The majority of algorithms is based on polygonal rendering, presumably because triangles are the best supported primitive in current graphics hardware. Image based methods also benefit from increased texture performance and typically use texture mapped polygons for rendering as well, rather than doing more work in GPU fragment programs. Point based systems have had some exposure since the early beginnings of real-time rendering, but were never as widely accepted and used as polygonal approaches. However, the works by Deussen [DCSD02], Dachsbacher [DVS03] and others show that they should not be completely ignored.

In Table 2.1, checkmarks indicate that a certain feature is explicitly supported by the algorithm. No checkmark denotes *omitted in the current state of development*' but should be generally feasible, possibly with slight adaptations) or unknown, and a dash indicates that a feature '*conflicts with the intentions of this method*' (because, for example, an expensive precalculation would become useless for animated geometry). For the performance and quality ratings, an empty circle represents the lowest rating, and full circles the highest. For older algorithms, we tried to at least roughly estimate how well they would perform on current computer hardware and how much they would benefit from its additional capabilities. Of course these are entirely subjective, but we feel that they still provide a good overview of how these algorithms perform. The same holds true for the depiction of applicable view distances, which was derived from considerations on achievable image quality (in the near field) and ability to render very large numbers of trees (in the far field).

supported geometry - Does the algorithm support full trees, or does it only render branches or leaves? Algorithms that only deal with one or the other need to be combined with suitable alternatives, which may lead to visual discrepancies in the way levels of detail are handled.

view dependent LOD - Is there integral support for distance or view direction based levels of detail, such as multiresolution representations?

animation - Does the algorithm support dynamic scenes, eg. movement of branches due to wind? Physical simulation is not required, but

it should be possible to add without major recalculations for each frame.

dynamic lighting - Does the algorithm support dynamic lighting? Static normal mapping is almost always possible, so we restrict this criterion to more complex effects like self shadowing.

quality - How good is the image quality in comparison to other algorithms with similar features?

performance - How fast is the algorithm?

memory requirements - How much memory does the algorithm consume? This includes global memory requirements (needed once for all instances, eg. impostor textures) as well as per-instance memory consumption (if instancing is available). The rating is subjectively based on a comparison with other algorithms.

Distance range - At which distances is the algorithm best used? At the extreme near field, leaf details such as veins and edge structure should be visible; the extreme far field requires the representation of a very large number (10^6 to 10^7) of trees.

State Of The Art Summary

In a perfect world, an algorithm would support all the features in table 2.1, at a very high quality and in real time, over the entire range of view distances. However, practically all current algorithms include some features at the cost of others.

For example, approaches that only support branches or foliage will be difficult to integrate if real-time animation is desired, and care must be taken to coordinate level of detail methods if branches and foliage are rendered separately. Image based approaches are also difficult to integrate with animation, since they typically involve extensive preprocessing and offline generation of textures which may need to be adjusted for dynamic lighting or animation. A notable exception is [MNP01], which does support dynamic lighting at the expense of significantly increased memory demands and rendering overhead.

Referring to the distance ranges depicted in Table 2.1, it is apparent that most rendering techniques focus on a 'moderate' distance range, where vegetation is both distant enough to allow a significant reduction in detail, but not far enough that a very large number of plants needs

	Description	Lea. / Br. / Full	View dep. LOD	Animation	Dyn. Illum.	Quality	Performance	Memory	Distance range (near – far)
	Plant Leaves [WWD+05]	L			✓	●	○	◐	
	Complex Botanical Scenes [MFC97]	F	✓	–	✓	◐	◐	○	
	Complex Plant Ecosystems [DCSD02]	F	✓	–		●	◐	◐	
	Plant Models with Complex Organs [ZBJ06]	F	✓	–	✓	●	◐	◐	
	Point-Based Trees [GMN05]	F	✓	–		◐	◐	◐	
	Procedural Multiresolution [LCV03]	B	✓		✓	●	◐	○	
	Complex Photorealistic Landscapes [CCDH05]	F	–	–	✓	◐	◐	○	
	Fractal Plants and Trees [Opp86]	B		✓	✓	○	○	●	
	Single Polygonal Mesh [LVF+01]	B			✓	●	◐	◐	
	Volumetric Reconstruction [RMMD04]	F	✓	–	–	●	○	●	
	Interactive Forest [GCF01]	B	✓	✓	✓	◐	◐	○	
	Forests at Human Scale [SK04]	F		–	–	◐	●	◐	
	Realistic Trees [WP95]	F	✓		✓	◐	◐	◐	
	Shading and Shadowing [MNP01]	F	✓	–	✓	●	◐	●	
	Hierarchical Image-Based Rendering [MDK99]	L	✓	–		◐	◐	●	
	Structured Particle Systems [RB85]	F		–	✓	◐	○	○	
	Image-Based Multiresolution [LCV04]	F	✓	–	–	◐	◐	◐	
	Sequential Point Trees [DVS03]	F	✓	–		◐	●	◐	
	Deferred Splatting [GBP04]	F	✓	–		◐	◐	◐	
	View-Dependent Multiresolution [RCRB03]	L	✓	–	✓	◐	●	◐	
	Forest Scenes in Real-Time [DN04]	F	–	–	–	●	◐	◐	
	Slicing and Blending [Jak00]	F		–		◐	●	○	
	Precomputed Z-Buffer Views [Max96]	L		–	✓	◐	◐	◐	
	Drop and Resize [Hal01]	F	✓	–	–	◐	●	◐	
	Textured Quadric Surfaces [Gar84]	F		–		○	●	○	

Table 2.1: Summary of the presented algorithms, roughly sorted by view distance range. Checkmarks indicate available features; Circles represent low (empty), medium (half filled) and high (filled) quality, performance and memory consumption.

to be displayed. The algorithms presented by Decaudin [DN04] and Halper [Hal01] extend their applicable range well beyond most other algorithms, but require special care to handle bordering regions ([DN04]) or do not support illumination ([Hal01]).

In contrast to rendering methods for other types of scenes, where graphics hardware fill-rate or bandwidth limitations are usually the performance bottleneck, many of the more realistic methods described above have significant CPU overheads. Obviously, this does not apply to all the algorithms, but the ratio of CPU limited approaches seems to be higher than in other areas.

Further Research Opportunities

Once again looking at Table 2.1, it is apparent that real-time animation is not easily implemented in most state of the art algorithms. However, "real" trees are hardly ever perfectly still since even a very light breeze will cause individual leaves or small twigs to move. This is often also visible at a larger scale if wind blows over the canopy of an entire forest. Therefore, such motion effects should be incorporated even for far field rendering methods.

Also, the handling of large scenes with very high detail quickly becomes problematic. If a multiresolution renderer requires a top-down approach (ie. lower levels of detail are generated from higher levels), that effectively results in the (temporary) creation of the full scale of the entire scene. A better approach would be bottom-up, such that the explicit generation of geometry or other data is avoided unless necessary for rendering.

Procedural modeling approaches are probably best suitable for such an approach (and partially incorporate it), but once the system needs to go beyond individual trees and collect them into larger groups, existing systems typically resort to tileable base data, which has its own problems. For example, border tiles and repetition artifacts are typically clearly visible, or require special attention.

At a closer scale, a variety of rendering methods exist but they are mostly self-contained and no obvious transitions exist between them. Therefore it is desirable to investigate how existing approaches could be adapted to allow seamless blending to other methods.

In summary, we believe that ultimately the goal should be an uniform approach that allows a seamless transition from viewing a forest canopy on the horizon to inspecting a single leaf.

PART I

NEAR FIELD VEGETATION RENDERING

CHAPTER 3

POINT BASED VEGETATION RENDERING

> This 'telephone' has too many shortcomings to be seriously considered as a means of communication. The device is inherently of no value to us.
>
> Western Union internal memo, 1876

3.1 Introduction

The natural structure of vegetation usually results in an immensely complex geometry. However, at typical viewing distances, leaves and other small parts are often smaller than a single pixel. Therefore, rendering these leaves as individual quadrilaterals (maybe even textured) is wasteful. Consequently clustering techniques (groups of leaves are approximated by a single polygon) or image based techniques with a similar strategy are usually employed [Int02, MNP01, Jak00]. Point based and hybrid techniques have been also explored [WP95, DCSD02].

The point based approach allows for nearly seamless transitions to other levels of detail and is therefore a good choice for intermediate rep-

resentations. However, since point sample clouds are a very large number of independent and disjoint samples, traditional occlusion based acceleration methods such as backface culling or spatial schemes can not be applied easily. Still, the large number of samples necessitates some means of simplification in order to achieve real-time rendering performance.

This chapter discusses a preprocessing step to determine the exact visible set for a number of views; at runtime the visible sets of the three closest matches to the actual view are combined for rendering. Point budgets can be assigned, so that distant objects are rendered with less detail.

Point Based Rendering of Vegetation

Rendering vegetation with point based approaches has several advantages over other methods. Points show no inherent connectivity, making it easier to approximate the intricate structure of a canopy. They can be easily managed within display lists and sorted by importance for adaptive levels of detail or blending with more detailed representations.

Hybrid point and polygon based approaches were first used for rendering vegetation by Weber and Penn [WP95]; later a much more flexible system was presented by Deussen et al. [DCSD02]. These approaches are preferable because polygonal data is useful at close distances, where purely point based approaches tend to 'fall apart' (see Figure 3.1).

Although these systems provide view *distance* based levels of detail, they do not use the view *direction* for further data reduction methods and rely on orientation independent methods for LOD generation. On the other hand, some point based rendering systems do exploit the current view direction for backface culling and splat size estimation [RL00]. However, when leaves are approximated through individual point samples, there is no continuous surface or solid, making backface culling inadequate.

Another method for rendering point based data is to use a randomized approach [Uni, WFadH00]. In this case, a (possibly view dependent) heuristic selects arbitrary samples from the data set. Special care must be taken to make sure that holes and artifacts do not occur, for example by using a poisson disc distribution function.

The presented approach uses a number of precalculated view dependent visible sets to create an approximate visible set for arbitrary view directions. These visible sets are created in a hardware accelerated preprocess and combined dynamically at runtime.

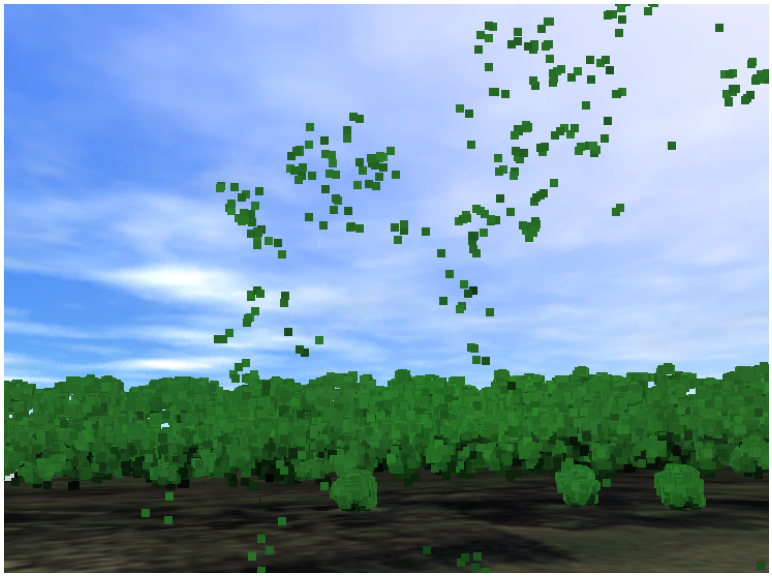


Figure 3.1: A point based tree falls apart when viewed from to close up.

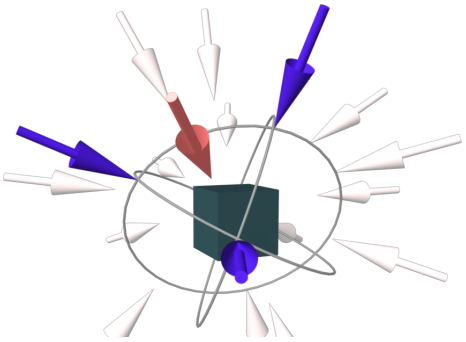


Figure 3.2: Identifying closest available view directions. For an arbitrary viewpoint (red arrow), the three closest precalculated directions (blue arrows) are identified and rendered. Light gray arrows symbolize additional precalculated directions not used for this viewpoint.

3.2 Preprocessing

Visible set determination can be performed very efficiently in hardware. First, a number of views is generated by selecting points on the unit sphere. Since trees cannot usually be viewed from directly underneath, this can also be constrained to a hemisphere. Our implementation employs a simple tetrahedron subdivision scheme to generate view directions.

The entire data set is then rendered from each view, but instead of using the original color information each leaf is rendered with a unique color. Individual leaves are rendered as single point primitives. To avoid unwanted artifacts, the data set is rendered without antialiasing, attenuation, or lighting. The frame buffer is then read back and each pixel's value is used to identify the visible sample at this point.

If occlusion queries are supported on the graphics hardware, reading back the frame buffer can be avoided: The data set is first rendered in full as before. It is then re-rendered in chunks of n samples with occlusion query enabled, and the depth test set to 'less or equal'. If no pixels have been updated, the entire chunk was invisible and can be discarded. Otherwise it is subdivided and processed recursively to identify all visible samples.

Sample Identification

To be correctly identified after the frame buffer has been read back in, each sample needs to be assigned an unique color. For simplicity, we use the sample's array index in the data set. Of course, 8-bit RGB colors effectively limit the maximum number of samples that can be identified in a single pass to $2^{24} - 1$.

However, this limit can be easily avoided by using multiple partitions of $2^{24} - 2$, reserving the value $2^{24} - 1$ for the background (no sample) and 0 for any samples not within the current partition. The preprocessing step then needs to render the entire set more than once, until all samples have been covered.

LOD Estimation

By rendering samples with more than one pixel and by re-rendering the same view from varying distances, a level of detail estimation can be obtained as follows: For each pixel that has been covered by a certain id, the contribution of the corresponding sample is increased by one. After all distances have been covered, samples with zero contribution are discarded, and the remaining samples are sorted by descending contribution. This way, leaves which are visible from all distances end up with the highest contribution count and will be rendered even if the total number of points to be rendered exceeds the allocated budget.

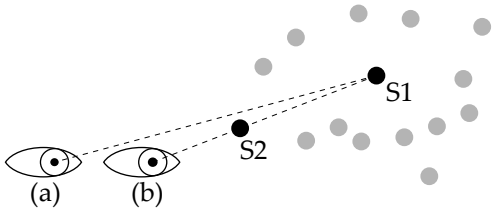


Figure 3.3: Perspective Occlusion. Splat S1, although visible from viewpoint (a), is occluded by splat S2 when viewed from viewpoint (b) as it is seen at a slightly different angle.

Alternative LOD Generation

Even though it is tempting to assume that samples which are visible at larger distances are also visible at a closer range, this is not always correct for perspective projection. Such samples may be obscured if perspective views are rendered from varying distances, rather than at different scales. Figure 3.3 illustrates this difference. However, a similar error is incurred from approximating the view direction; there may be samples that should be visible from the actual viewpoint but were not detected in any of the three views.

Stems and Branches

If desired, stems and branches can be included in this system. This is particularly useful if the tree has been generated automatically and includes a great number of small twigs. We have implemented this by assigning a separate range of identifier values to polygons; the sample identification process only requires minor changes to accommodate this. Although the resulting polygonal data does not lend itself very well to triangle striping or similar acceleration schemes, it greatly reduces the data to be rendered while still maintaining good visual quality.

Alternative LOD Method

As an alternative to sorting samples by contribution alone, they can be arranged by view distance first. The advantage of sorting samples by decreasing distance is precise control over how many points need to be rendered: an upper bound is the splat count for the next (closer) view (see Figure 3.4). Interpolation could be used for smooth transition between view distance steps, although ideally this should not be necessary, as the

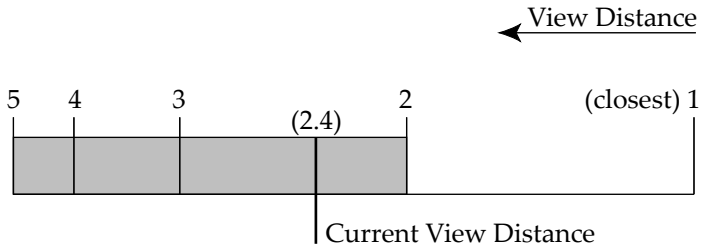


Figure 3.4: Sorting point samples by distance. For the current view distance, all point samples up to and including view distance '2' would be rendered.

closer view does already provide coverage for all visible samples. We have therefore chosen not to implement smooth transitions as they are already quite unobtrusive.

A straightforward implementation of this method would require storing additional information for each sample, or searching through the previously determined (greater) distances to determine if this sample has been seen already (obviously, each sample should be rendered only once). However, this time consuming step can be avoided:

Each view direction is rendered from various distances, beginning with the greatest distance. Samples that have already been visible at an earlier iteration are not assigned their original id, but simply rendered with the reserved background id. They will therefore still correctly obscure other samples, but remain invisible to the following sample identification process.

3.3 Rendering

During rendering, the three closest view directions are determined for each object. This can be done naively by finding the dot product of the actual view vector and the vector stored for each view direction. A more efficient method would be to precompute the three closest views in a cube map or similar lookup table.

Next, the number of samples to be rendered is found as a function of object distance and the total number of samples for this view direction. The samples can then be rendered efficiently as a single vertex array. For the original algorithm, since samples are ordered by contribution more important (ie. highly visible) data is guaranteed to be rendered even if only few samples are displayed. The alternative method always renders

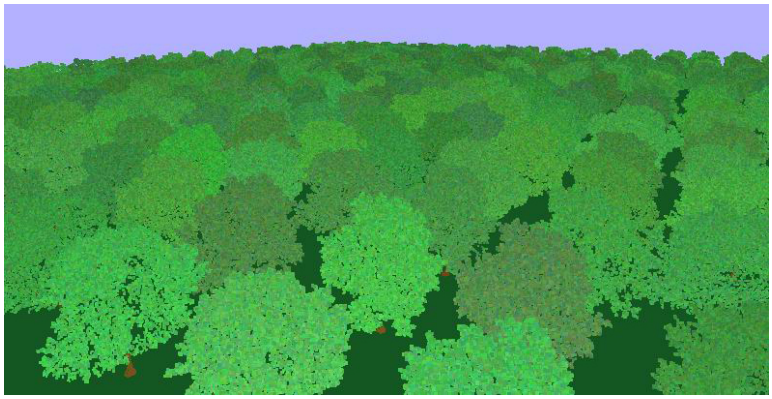


Figure 3.5: 400 instances of the reduced "Oak" model, interactively rendered from an arbitrary viewpoint with individual orientation and color bias for each instance.

a sufficient number of point samples through the use of a distance based lookup table for the appropriate sample count.

Rendering Multiple Instances

If a model is reused several times, for example to produce a group of trees, a slight variation in color and texture will greatly enhance the visual quality. However, this is not easily possible since sample data - including color information - is stored in vertex array for efficient rendering. For color variation, this array would either have to be copied and modified for each instance, or walked through "by hand" and each sample rendered individually. In both approaches, the advantage of using a vertex array is lost.

We circumvent this problem by reusing the same vertex array, including color information. The variation of appearance is achieved by adjusting light source color and intensity instead, which can be regarded as adding an individual per-instance bias (see Figure 3.5).

3.4 Results

We have implemented preprocessing and rendering of the leaves of various models of trees; the tree models were generated through an algorithm similar to the one proposed by Weber and Penn [WP95]. Lighting was precalculated by ray casting through a regular volume grid; the den-

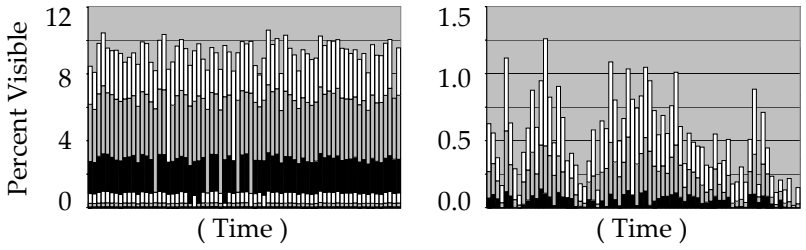


Figure 3.6: Percentages of visible leaves (point samples, left) and branches (ie. triangles, right) for tree "Oak" (total points 91054, triangles 52076). Note different scales on graphs.

sity of each cell was estimated through a heuristic function based on the number of enclosed leaves.

Figure 3.9 displays the preprocessing results for one particular tree model. 26 views were generated through a regular subdivision scheme, and the total and relative count of visible samples found with the original algorithm. This model only contains a dense set of leaves and no polygonal stems or branches. Loading the original data set (10 MB) and preprocessing took about 35 seconds on an 1.4GHz Intel Pentium 4 with GeForce4 graphics. The total number of samples for all views is 86265, each consisting of a vector and color information, for 15 bytes/sample and 1.3MB total per tree model. Data structure overhead is about 20 bytes per view direction. If polygonal data is stored as well, it can be estimated with an additional 45 bytes/triangle. For comparison, Meyer et al. report "a few tens of Megabytes" for their method, and a preprocessing time of about 75 minutes using an Onyx2 Infinite Reality.

A visual comparison of the original and reduced models is shown in Figure 3.10. Although different splats may be rendered for the full and reduced models (Figure 3.10c, the actual pixel value difference is quite small, as illustrated in Figure 3.10d.

In Figure 3.6, a complete tree model has been preprocessed with the alternative, distance-sorting algorithm. Each column represents a particular view direction, and the individual view distances are stacked from distant (bottom) to close-up (top). The left graph displays point samples (ie. leaves), and the right graph represents triangular data (ie. stem and branches). In this case, the total time for I/O and preprocessing was about three minutes. The total data file size is about 12MB.

The model, an oak-like tree, has a much less dense leaf cover than the "Balsam" model, and therefore also a higher percentage of visible

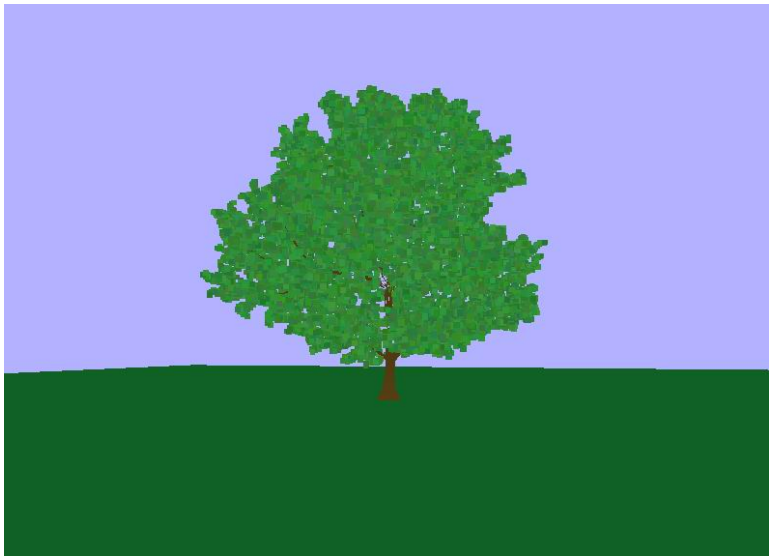


Figure 3.7: Reduced "Oak" model, interactively rendered from an arbitrary viewpoint using 3 out of 70 precalculated view directions.

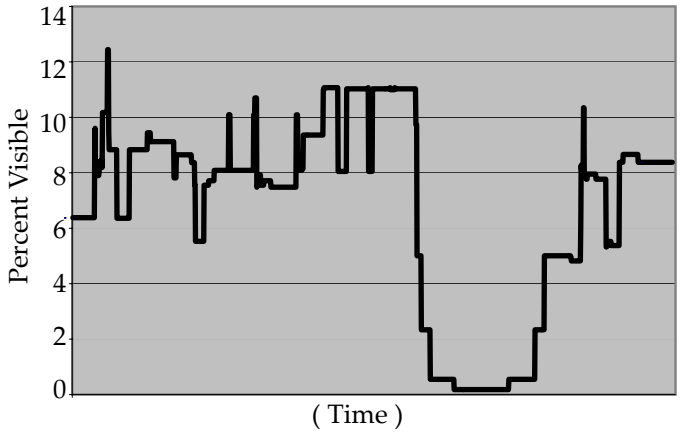


Figure 3.8: Percentage of total bandwidth used over time for a number of views.

leaves in comparison. A number of things can be seen in these graphs: For some view directions, there are distance steps that do not reveal any new samples. This may be in part due to the use of the OpenGL Point Parameter extension to adjust the point size according to distance.

Figure 3.7 shows the oak model rendered from an arbitrary viewpoint. Three out of 70 view directions have been rendered, resulting in

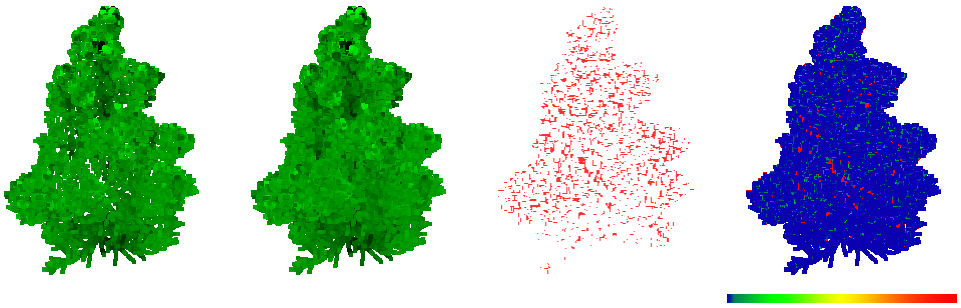
view	visible	pct.	view	visible	pct.
1	4236	2.19%	2	6882	3.56%
3	6606	3.41%	4	4197	2.17%
5	6665	3.44%	6	4324	2.23%
7	4310	2.23%	8	6560	3.39%
9	6099	3.15%	10	5520	2.85%
11	7275	3.76%	12	5213	2.69%
13	6451	3.33%	14	5592	2.89%
15	4103	2.12%	16	6015	3.11%
17	3608	1.86%	18	3594	1.86%
19	5282	2.73%	20	3845	1.99%
21	5630	2.91%	22	4304	2.22%
23	5891	3.04%	24	6489	3.35%
25	6815	3.52%	26	5482	2.83%
		total	26	140988	72.82%

Figure 3.9: Preprocessing results for tree "Sasafras" (total points 193608, splat size 4 pixels, viewport $600 * 600$).

a total of 22897 points (25.15%) and 763 polygons (1.47%). Assuming that triangles are three times as expensive as points to send to the graphics pipeline, the relative bandwidth usage can therefore be estimated at 10.19%. Figure 3.8 tracks this value over time as the viewpoint is moved about the model. As can be seen, it drops significantly - to less than 0.3% - as the view distance increases.

Due to the nature of our algorithm, overdraw can be almost completely avoided while still providing guaranteed bounds on visual fidelity. This mostly depending on how many view samples have been generated, as well as any other parameters such as point size. Consequentially, these cannot be changed dynamically and need to be chosen carefully at the preprocessing step.

Also, since our implementation only uses the view direction and position for visibility determination, the camera's field of view is one of these fixed parameters. While this may be an acceptable solution for most cases, a more flexible solution may be desirable to achieve better visual quality if these parameters do need to be changed dynamically.



(a) Reduced model (b) Full model (c) Splat diff. (d) Pixel value diff.

(a) Reduced model rendered from an arbitrary viewpoint with 19460 point samples (10.05% of the original size).

(b) Full model, from the same viewpoint (193608 point samples).

(c) Splat differences: Red pixels show where a different splat is rendered in front by the full model.

(d) Color coded pixel value differences between reduced and full models.

Figure 3.10: "Sasafras" tree model, comparison between reduced and full models.

3.5 Summary

We have presented a new algorithm for view direction based data reduction of disjoint point sample clouds. Our method is fast (using commonly available hardware acceleration), memory efficient, and easily adaptable to other data with similar properties. Preliminary results have been presented, illustrating a large savings in memory, preprocessing and rendering load.

The benefit of this method is that it can handle objects and static object groups where traditional occlusion based algorithms fail due to high complexity.

Obviously, this algorithm is not inherently restricted to point sample clouds and can be easily generalized to other data such as disjoint polygons or entire objects (which can be approximated as bounding spheres or bounding boxes for speed).

Furthermore, the polygonal stems could also be identified by the same process. This would lead to a fully usable tree rendering model. To improve near field rendering quality, the preprocessing step can be per-

formed with polygonal leaves. By counting the pixels covered by each leaf, an immediate metric is obtained for selecting the appropriate level of detail for each leaf: more than a few pixels: polygonal; only few pixels: point; zero: discard. Each view would then consist of several arrays: a stem list, a polygonal leaves list, and the original point sample list.

CHAPTER 4

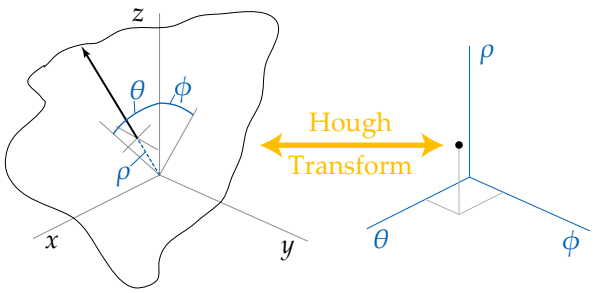
VEGETATION SPECIFIC BILLBOARD CLOUDS

> A life spent making mistakes is not only more honorable, but more useful than a life spent doing nothing.
>
> George Bernard Shaw

4.1 Introduction

For rendering vegetation at medium view distances, image based system are frequently used [Int02, DDS03, CCDH05]. In these systems, detailed geometry such as a group of leaves is represented as a single textured polygon. Therefore, these approaches provide a good tradeoff between rendering quality and performance as long as the flatness of the representative polygon does not become apparent.

One significant drawback of such approaches is the identification of suitable representative planes. Automatic placement methods are typically coupled with a procedural representation of the tree model [Int02], making them unsuitable in the general case where this information is not available.



(a) primal space (b) dual space

Figure 4.1: A plane in primal space (a) is transformed to a point in dual space (b) via the Hough transform.

However, the billboard clouds introduced by Decoret [DDS03] does not rely on such information. Dubbed an *extreme model simplification* method, an arbitrary polygonal input model is automatically simplified to a minimum set of planes that are automatically placed within the model. A similar approach by Andujar et al. [ABC+04] is difficult to apply to vegetation models, as it requires volume inside/outside tests, which are difficult to decide on non-manifold geometry.

4.2 The Original Billboard Cloud Algorithm

The goal of extreme model simplification is to find a minimal set of planes such that each polygon of the original model can be projected onto a *valid* plane. A plane is valid if it is closer to each vertex of the polygon than the error threshold ϵ; this threshold then represents the *validity domain* of a vertex.

In their billboard cloud generation algorithm, Decoret et al. [DDS03] formulate this simplification problem as a clustering problem in a dual space constructed by the *Hough transform* [DH72]. In this space, planes map to individual points in a three-dimensional coordinate system that represents the two plane angles θ, ϕ and the distance ρ from the origin of the plane (see Figure 4.1).

Conversely, a point in primal space can be defined as the intersection of all planes passing through that point, and maps to a sheet (or height field) in dual space. Also, the *validity domain* of a point – essentially a sphere of radius ϵ in primal space – is the same sheet translated up/down by ϵ along the ρ axis. Finally, the validity domain of a face is the intersection of the validity domains of its vertices.

For clustering, the dual space is represented as a discrete volume. The validity of each face of the original model is accumulated in this volume, and a greedy selection algorithm iteratively selects the best representative planes from the volume until all faces have been simplified to at least one plane.

As the final step of BBC generation, plane textures are generated by projecting the faces of the input model to their respective simplifying planes. Alternatively, faces can be projected to *all* planes within their validity region. This results in a more dense representation, which may be more desirable depending on the input model.

4.3 An Improved Simplification Algorithm

If this algorithm is directly applied to trees, the representation is often inadequate because the visual importance of the different features of a tree is not evident in the geometry.

For example, a large threshold is usually required to obtain sufficient data reduction rates but also leads to undesirable artifacts. One typical case is the entire trunk geometry being simplified to a single plane and thus becoming invisible if this plane is viewed at a grazing angle. Also, the context-free simplification results in artifacts such as loss of continuity and plane orientations that are mathematically correct, but visually unsatisfactory.

In addition to the methods presented below we have observed that much better results can be obtained by separating foliage and branches and simplifying them separately, using different parameters for each part.

Vertex Welding

During the simplification process, vertices are projected onto the supporting planes. This leads to discontinuities if polygons sharing a vertex are projected onto different planes, as the projected points usually do not coincide, leaving a visible gap (see Figure 4.2(b)).

These artifacts can be reduced (although not eliminated entirely) by a process called *vertex welding*. The first time a vertex is projected onto a simplifying plane, it is permanently moved to the projected location in world space, slightly distorting the other connected polygons. This effectively moves the projected locations on the various supporting planes of the other polygons closer together. Mathematically, the worst case world space distance between two projections onto simplifying planes is

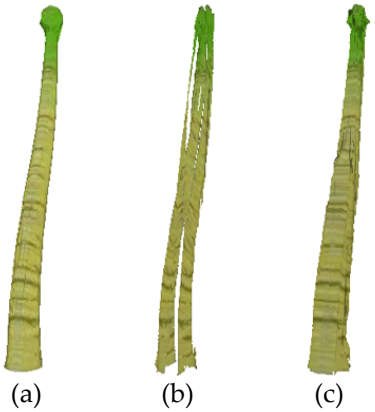


Figure 4.2: Palm tree stem: (a) polygonal model, (b) normal billboard cloud, (c) billboard cloud with vertex welding.

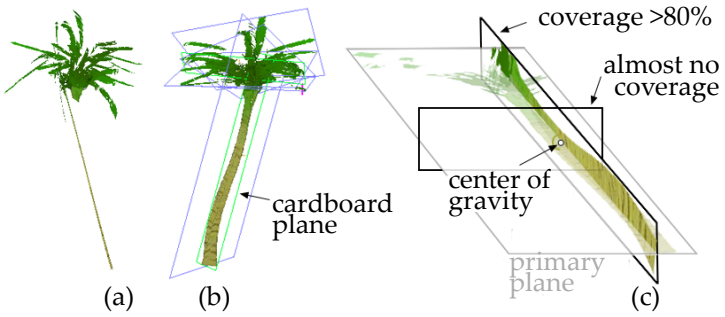


Figure 4.3: "Cardboard" plane construction to avoid edge-on artifacts of trunks.

2ϵ without vertex welding (if the projections are on opposite ends of the validity domain of the original vertex), and ϵ with vertex welding (since subsequent projections must lie within the validity domain of the relocated vertex). Figure 4.2 shows a Palm tree as the original model, and a comparison of 'plain' billboard clouds and with vertex welding enabled.

Cardboard Planes

For vegetation, relatively large values of ϵ (typically 10-15% of the bounding sphere radius) are required to simplify the mode to a few tens of polygons or less. The resulting models still retain a good visual representation of the original model, but the large ϵ do cause some problems

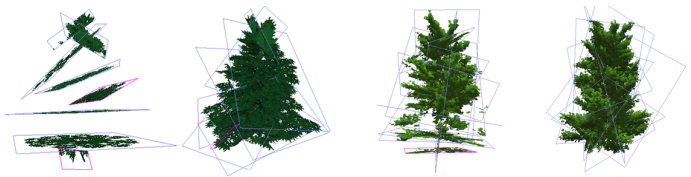


Figure 4.4: Tree models without (a, c) and with (b, d) view dependent penalty to avoid horizontal planes.

with branches. Figure 4.3(a) shows how the entire palm trunk is simplified to a single plane and becomes invisible when this plane is viewed edge-on.

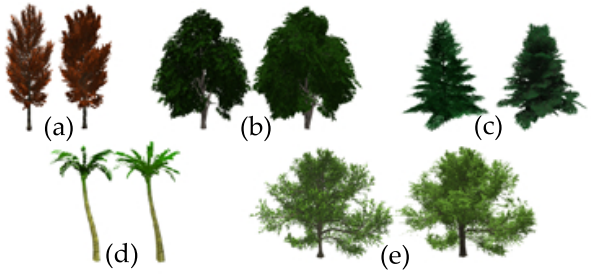
To avoid these artifacts, branches and leaves are simplified separately. Leaves are reduced with the traditional approach, and for branches an automatic *cardboard plane* construction is performed: one a simplifying plane has been found, orthogonal planes are tested for their coverage of the simplified geometry. If a plane is found that covers a significant amount of the simplified geometry, it is included in the constructed model, effectively constructing a crosswise impostor (see Figure 4.3).

View-dependent Penalty

Since one of the main applications of simplified vegetation is a walk-through scenario, the resulting models need to work well when viewed at eye height. However, due to the regular structure of trees the simplifying planes are often nearly horizontal. Figures 4.4(a,c) demonstrate the resulting artifacts.

To avoid these problems, a view direction dependent penalty can be assigned to simplifying planes. In our implementation, this penalty is weighted by the normal direction of the plane, making near horizontal planes less likely to be chosen during clustering and plane selection. However, other weighting schemes could easily be added, for example for models that are only viewed from a certain direction.

As demonstrated in Figure 4.4, the resulting models require slightly more planes (11 vs. 8) but are visually much more pleasing.



tree	ϵ	faces	billboards	preproc. time (s)	approx. trees / sec
a	10.0	108783	12	342	143000
b	12.0	159160	14	403	122000
c	12.5	20547	13	62	132000
d	6.5	7292	8	25	214000
e	6.5	169781	21	496	81000

Figure 4.5: Polygonal tree models and billboard clouds representations, together with creation and rendering statistics. In the above table, the error threshold ϵ is given as a percentage of the bounding sphere radius.

4.4 Results

Figure 4.5 presents a visual comparison of various tree models and their billboard cloud representations. In addition, the table in the same figure shows the error threshold used and various creation and runtime statistics for each of the models. The rendering performance (rightmost column) of this table clearly shows that billboard clouds are a viable approach to rendering large numbers of trees, supporting up to 200.000 rendered instances per second in a moderately optimized framework. This figure can very likely be further improved through the use of a texture atlas and similar techniques.

Furthermore, Figures 4.6 and 4.7 show how the models are integrated in a typical urban visualization framework. At the distances shown, the visual impression of the billboard cloud models is quite at its limits. As an abstract notion of vegetation is desired it is still sufficient, but in many cases a more realistic impression is desirable. One approach for rendering more detailed models that is directly linked to the presented billboard clouds will be described in the next chapter.

4.5 Summary

Billboard clouds are a very flexible approach to simplifying highly complex models. No information other than the input geometry is required. Therefore they are easily applicable to vegetation models, which are of-

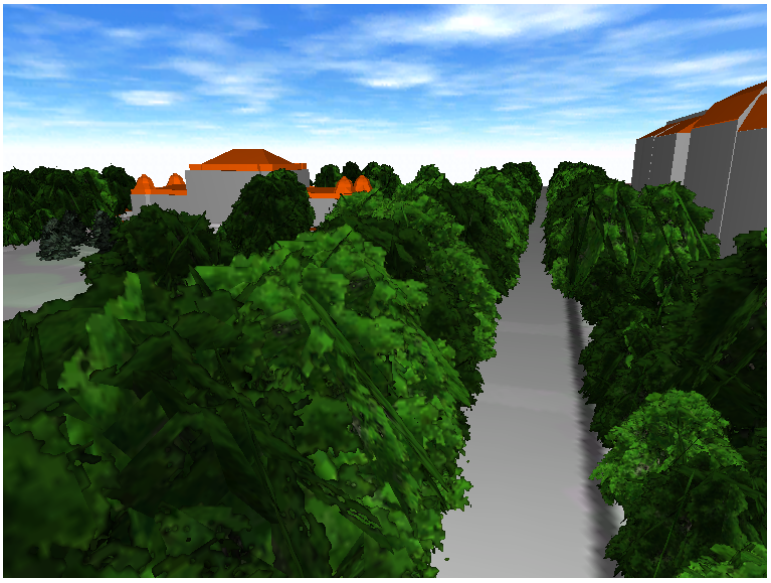


Figure 4.6: Improved billboard trees in an urban setting.

Figure 4.7: Walkthrough scenario with improved billboard trees.

ten created using third party tools where it would be very difficult to integrate additional information, or guarantee properties such as water tightness.

However, the original algorithm suffers several drawbacks when directly applied to vegetation. The growth pattern of trees results in a bias towards horizontal planes, which are undesirable in walkthrough applications, and small detail may be simplified to a single plane and therefore vanish for certain view directions.

We have presented several extensions to the original algorithm that address these issues. The resulting models can be rendered very efficiently, are visually pleasing at moderate view distances and have been successfully used in landscape and urban visualization applications.

CHAPTER 5

DISPLACEMENT MAPPED BILLBOARD CLOUDS

> The most exciting phrase to hear in science, the one that heralds new discoveries, is not "*Eureka!*" but "*That's funny...*".
>
> Isaac Asimov

5.1 Introduction

Based on the observations made in Chapter 4, it becomes obvious that a slightly more detailed representation of vegetation is desirable. Ideally, such a representation would be compatible with the simpler billboard clouds to allow a seamless transition depending on the current view distance. In this chapter, we will focus on an extension of the image-based *billboard clouds* approach that addresses these goals.

To reiterate, billboard clouds represent a part of a scene as a collection of arbitrarily placed and oriented texture mapped planes to which the original geometry is projected. Therefore, the represented part of a scene can be observed from all sides. But below a minimum viewing distance the visual quality deteriorates. While the quality of BBCs is sufficient for medium to distant parts of a scene, the "flatness" of the individual

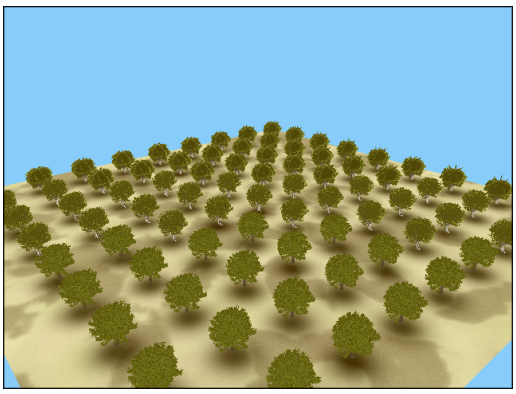


Figure 5.1: Screen shot from our interactive test application, simultaneously displaying full geometry (396.000 faces) for close trees, DMBBCs in the middle distance and BBCs for the far field.

planes becomes very noticeable when viewing them at closer distances or at grazing angles. This effect is independent of the texture resolution and cannot be easily avoided: switching to full geometry earlier defeats the purpose of BBCs as this leads to a much higher geometry load, while substituting a different BBC with a higher accuracy typically leads to noticeable popping artifacts. This restricts the usefulness of the BBC technique for scene parts close to the viewer.

We introduce *displacement mapped billboard clouds* (DMBBCs) to overcome these problems. The main idea of DMBBCs is to augment every plane of a BBC using a structure similar to a displacement map in order to better represent the original geometry. In essence, the BBC planes approximate the rough object structure, whereas the displacement map represents fine details, leading to a much lower geometric error. However, in contrast to surface detail typically approximated with displacement maps, the geometry projected to a BBC plane is not necessarily continuous. We therefore introduce *thick displacement maps* and a novel hardware ray tracing algorithm that can trace through holes in the representation, thus providing a more plausible approximation of the original object. The ray tracing algorithm makes use of several acceleration techniques that are new for hardware ray tracing.

Impostors based on DMBBCs allow for fast rendering of complex scenes with high image quality even for close objects. In particular, the artifacts typically associated with billboard clouds are avoided through the use of additional parallax and visual depth, while no additional geometric complexity is introduced. The new representation allows for

seamless blending with traditional billboard clouds, so that distant scene parts can be rendered as fast as possible. DMBBCs can be rendered entirely on graphics hardware.

In addition to being applicable to vegetation, DMBBCs are also suitable for objects containing curved surfaces, where traditional billboard clouds typically fail to provide a sufficiently high image quality.

5.2 Related Work

For a comprehensive overview of impostor techniques (including simple *planar impostors*, *layered depth images* and *textured depth meshes*), we refer the reader to a recent state-of-the-art report by Jeschke et al. [JWP05]. Since DMBBCs are based on the billboard cloud technique, we refer to the original work by Decoret [DDS03] as well as Chapter 4 for a detailed discussion of this method.

Texture-based surface descriptions

As an early approach, *bump mapping* [Bli78] solely relies on shading, providing no correct silhouettes nor parallax effects. Several more recent methods enrich surfaces with geometric details that are stored in texture maps. For instance, *parallax mapping* by Kaneko [KIK+01] (with its various extensions [Wel04, MM05, Tat06]) simulate the appearance of a height field by shifting the texture coordinates within a texture depending on the viewing angle. Note that these techniques are hardly applicable for our purpose because a billboard cloud typically contains empty areas in the texture which cannot be represented in this way.

Surfaces with a certain depth can also be represented using ray casting on the GPU. Hirche et al. [HEGD04] were the first who defined a volume on a surface. They extruded the triangles along the vertex normals which results in prisms. Every prism is decomposed into three tetrahedrons. The ray casting is then applied to every tetrahedron by calculating the entrance and exit point and linearly sampling and evaluating a heightfield between them. The first hit point with the heightfield determines the position, and the color is read from a color texture. This technique supports correct object silhouettes, self-occlusions, interpenetrations, and even shadowing. Later Dufort et al. [DLP05] extended the work to semi-transparent data and Porumbescu et al. [PBFJ05] discussed the mapping in a broader way. Note that the idea of Hirche et al. is most closely related to our technique: the ray casting step is imple-

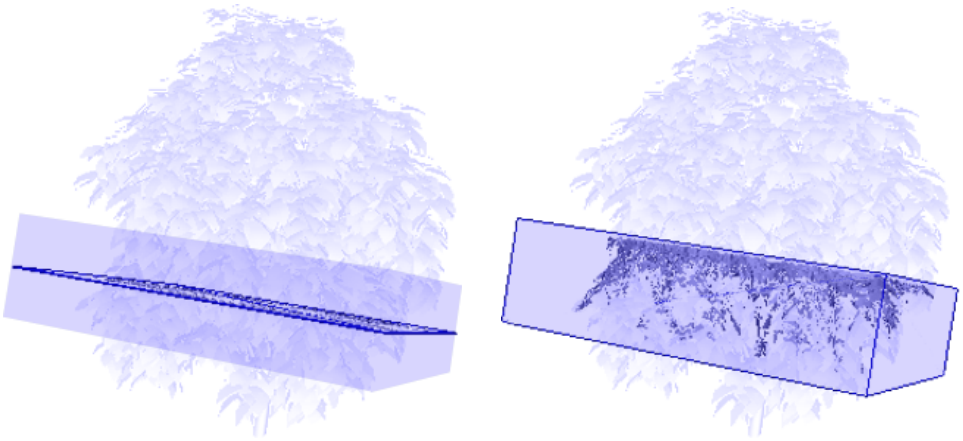


Figure 5.2: Left: billboard rectangle with its according validity region. Right: billboard box. Darker shaded regions show the rendered primitives together with the displayed contents. Note the difference between the flat rectangle and the volumetric contents of the box.

mented entirely in graphics hardware and allows for interactive frame rates, which makes the basic idea perfectly suitable for our DMBBCs. The main conceptual difference is that instead of prisms, DMBBC primitives are boxes which we can directly use for ray casting and we will allow the ray casting step to discover holes in the representation. We also accelerated the ray sampling process by using *sphere tracing* [PF05]. *Cone step mapping* [Dum06] or an enhanced version based on *safety zones* by Kolb et al. [KRS05] are further alternatives.

More recently, Policarpo et al. [POC05] applied ray casting to the original polygons of a mesh and implicitly defined a thick surface *inside* the object. While this works considerably faster than Hirche's method, silhouettes are still defined by the mesh geometry. They extended their work to better discover silhouettes and to support multiple height values per texel position [PO06]. However, their algorithm is based on the assumption of a continuous surface with well defined "inside" and "outside" values, which do not exist in the context of DMBBCs. The same applies to the work of Wang et al. [WWT+03a, WTL+04] who use massive preprocessing in order to reduce the cost for intersection searching at runtime. Please note that most of the presented techniques are explained in more detail in a book of Watt and Policarpo [WP05].

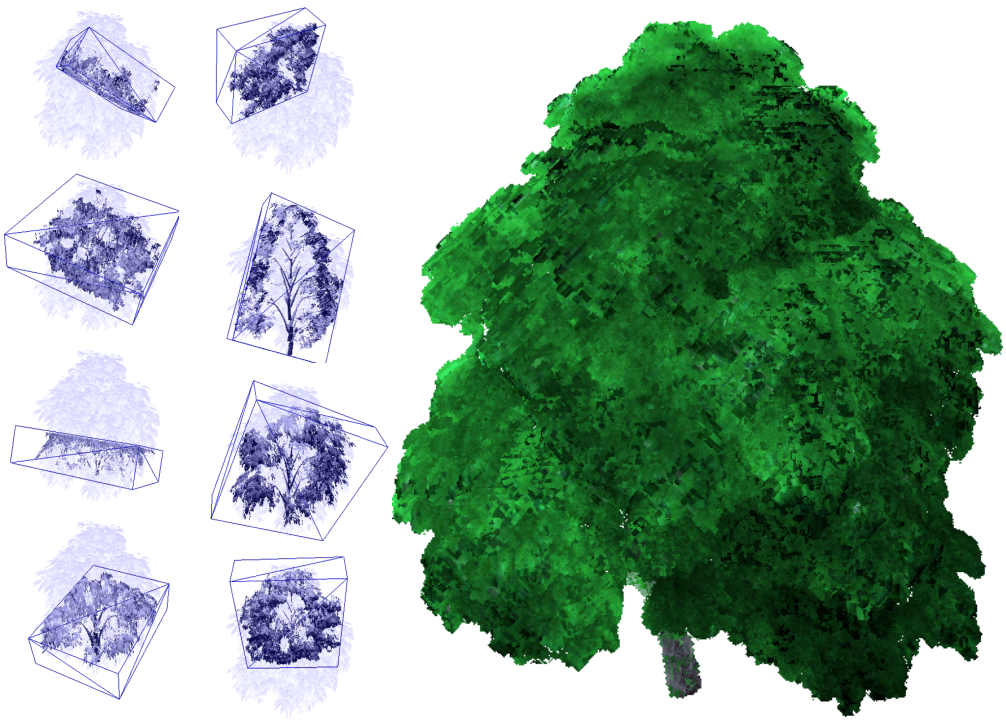


Figure 5.3: DMBBC representation of a chestnut tree consisting of 8 billboard boxes.

5.3 Displacement Mapped Billboard Clouds

DMBBC definition

As mentioned earlier, a *billboard rectangle* in a BBC is used to represent all model parts that are closer to the rectangle than a user-defined *validity threshold* $\pm\epsilon$. This validity threshold effectively represents a cuboid volume around the billboard rectangle, the *billboard box* (see Figure 5.2). The billboard box exactly defines the convex hull of the volume representing a DMBBC.

For a DMBBC, instead of simply projecting the model parts onto the rectangle and storing one color value, we store volumetric information, the so-called *volumetric displacement function*. At each texel (u, v), $f_{uv}(w)$ gives an opacity value for the height w along the rectangle normal, and an optional color value. For now, we assume that f is stored as a 3D texture with equidistant samples in w. Figure 5.3 shows the billboard boxes of a tree model.

Basic rendering algorithm

For displaying contents of a billboard box, we use a GPU ray-casting algorithm which determines the intersection of the viewing ray with the contained volumetric displacement function. The entry points of the viewing rays are interpolated from the corners of the billboard box by rendering the faces of the billboard box with interpolated (u, v, w) coordinates. The direction of the rays in texture space $\mathbf{v_{tangent}}$ can simply be calculated through an affine transformation of the world space view direction $\mathbf{v_{world}}$, defined by the local *tangent frame* $\mathbf{u}, \mathbf{v}, \mathbf{w}$:

$$\mathbf{v_{tangent}} = \begin{pmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z \\ \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z \\ \mathbf{w}_x & \mathbf{w}_y & \mathbf{w}_z \end{pmatrix} \times \mathbf{v_{world}}$$

The ray casting itself samples linearly along the ray in texture space (using a user-defined sampling rate) until an intersection is found, just like many previous methods described in Section 5.2. The search stops if either the ray leaves the box (which can be easily determined using the texture coordinates) so that the fragment can be discarded, or it intersects an opaque texel in the volumetric displacement function. In the latter case, the color value at the current position defines the output color. To optionally shade the current pixel, a normal can be read from an according map and used to apply an illumination model, as was already shown for BBCs [DDS03]. Note that the shader needs to output the correct depth value of the intersection point in order to correctly solve visibility between potentially overlapping billboard boxes. This is also straightforward and details are omitted here.

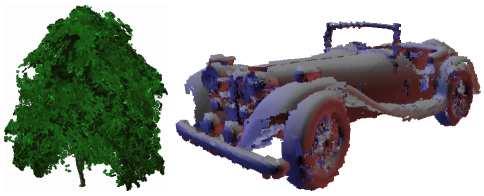
DMBBC generation

For generating the DMBBCs, we assume that the billboard rectangles together with the error value ϵ are already given. The rectangles can be obtained using any suitable billboard cloud algorithm; we have employed the variant described in Chapter 4. The extents of a billboard box are defined by a rectangle together with ϵ. The (u, v) texture resolution of the rectangle (defined by Decoret through the closest distance that the impostor should be valid for [DDS03]) is also valid for the box. Now given the original geometric model, we need to calculate and store the volumetric displacement function $f_{uv}(w)$. There are several ways how to represent this function. One obvious choice is to represent it volumetrically as a 3D texture. However, if the high memory consumption of this approach is

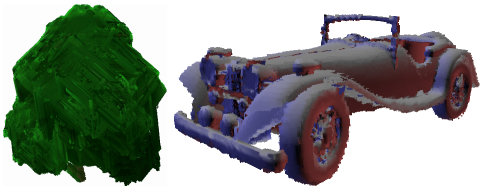
(a) Original meshes

(b) BBC models (tree: 16 rectangles; car: 31 rectangles)

(c) Volumetric DMBBCs: the volumetric data is stored as 3D texture, resulting in good visual quality but relatively high memory requirements.



(d) Shell DMBBCs: only one value per (u, v) position is stored. The tree still looks good except for the trunk, but the car shows many cracks and misses details at the front.



(e) Thick shell DMBBCs: two values per (u, v) define an interval. The tree is distorted by many incorrect "slabs", but the jaguar model looks better than with a shell DMBBC.

Figure 5.4: Original, BBC, and DMBBC renderings of two models.

not acceptable, the *Shells* and *Thick Shells* presented below are two other methods that significantly reduce the memory requirements while often providing a sufficiently high image quality.

Volumetric Representation

The highest DMBBC quality can be achieved by sampling the volumetric displacement function in a 3D texture. The resolution (sampling interval) of the texture in w-direction should be chosen so that the voxels are approximately cubic, which effectively generalizes the concept of tex-

ture resolution to 3D in the sense that a texel defines the smallest representable feature. The 3D texture is then filled by rasterizing the triangles of the original model into these voxels, and collecting for each voxel its normal vector and color information. If multiple polygons occupy the same voxel, this data can be either averaged with equal weights (this was used to generate Figure 5.4, c), or, if inside/outside information is available, the outside sample can be preferred. We currently do this step in software, but a GPU-based implementation (eg. by rendering the model into each of the volume slices) would be straightforward if preprocessing time becomes an issue. The software approach requires between several seconds and a few minutes per billboard box, depending on the size of the original mesh. At runtime, the ray casting algorithm has to test whether the 3D texture contains an object part at the current sampling position or not, indicated by either a special color or by using the alpha channel.

Although the obtained image quality is relatively high as also shown in Figure 5.4, c, the memory consumption is also high due to the typically large amount of empty texels. If the model should be dynamically illuminated at runtime, normal vectors can also be stored for every texel, again by averaging the normals of all surfaces that fall into a voxel or by selecting any of them. However, note that this further increases the required memory even though normals can be stored with lower texture resolution and/or compressed to only 2 bytes per normal at the expense of slightly reduced output image quality. However, if certain assumptions about the input model can be made, the following two sections present ways how memory can be saved with little impact on the quality.

Shell Representation

If we assume that the volumetric displacement function contains only one occupied region (voxel) for each (u, v) texel, then $f_{uv}(w)$ can be more efficiently represented by a single displacement value. This dramatically reduces memory requirements. We call this a *shell DMBBC*.

The displacement value represents the distance of the original sample to the billboard rectangle. Again, the same problem occurs as for the 3d texture case: if multiple model parts map to the same (u, v) position, one has to decide wether to average over all height values, color values and normals or to just select one of them. Note that for shell DMBBCs this issue is quite apparent since we deal with the whole thickness ϵ and not only with the thickness of one 3D texel as was the case for volumetric textures.

In contrast to pure displacement maps, which represent surfaces, the geometry contained in a billboard box is not guaranteed to be contiguous. Therefore, a ray needs to be able to trace *through* holes in the representation. On the other hand, continuous surfaces should not be pierced. We solve this by assigning a user-defined *thickness* (therefore "shell" representation) to the samples. If this thickness is chosen too small, holes will appear in continuous surfaces, if it's too large, the model will look very blocky. At runtime, the ray casting algorithm tests whether the current sampling position is within the thickness distance to the displaced sample so that it can be assumed to be hit by the ray. Currently we adapt the thickness manually using visual inspection. It might also be defined as the standard derivation over all acquired texel positions.

A shell DMBBC basically needs the same amount of memory as a BBC, except that along with the color information an additional displacement value per (u, v) position must be stored. On the other hand, the overall image quality is typically lower compared to the 3D texture approach. Figure 5.4 d shows an example for this where the introduced error is not too apparent for a largely unstructured model (here a tree) but becomes too obvious for an object with continuous surfaces and wrinkled details (here a car). The latter problem is caused by the global "thickness" value that on the one hand has to ensure that continuous surfaces appear closed but on the other hand should also preserve small details. The next section presents a method that overcomes this problem.

Thick Shell Representation

A more accurate representation of the volumetric displacement function is the *thick shell DMBBC* representation, which stores *two* height values per (u, v) position. The interval between the two values represents the part of the volume that is occupied by the model. We have simply stored the lowest and the highest displacement values for generating Figure 5.4 e, but other options might also be possible. Consequently, at runtime the ray caster tests if the current w value is within the stored interval at the current (u, v) position in order to determine if the model is hit.

There are many possibilities to balance image quality and memory requirements in the actual memory layout of a thick shell. It would for example be possible to store colors and normal vectors separately for the lower and higher interval boundaries and to linearly interpolate between the resulting output color values at runtime depending on the actual height. However, in order to save memory, for generating Figure 5.4 e we simply stored one additional height value compared to the shell

DMBBC approach. This works well for relatively thin objects such as the trunk of a tree where the color can typically be assumed to be identical for both sides. This also saves almost 50% memory. However, as Figure 5.4, c also shows, while the jaguar model looks considerably less cracky compared to shell DMBBCs (see Figure 5.4, d), the chestnut shows many incorrect "slabs" because neighboring leaves and branches are merged into one billboard box. This makes this technique much less useful for highly unstructured models like vegetation.

Rendering optimizations

Rendering Acceleration Using Distance Functions

GPU-based ray casting can become very costly since it is linear in the number of texels the ray projects to in each texture rectangle. In order to increase rendering speed without reducing the image quality we adapted an approach by Donnelly et al. [PF05], which is based on *distance functions*. Please note that it works similarly for all presented DMBBC variants.

In the preprocessing step, a so-called 3D *distance texture* is created. The (u, v) dimensions of this texture are the same as for the billboard boxes. The w dimension can be chosen as a tradeoff between fast intersection calculation and required memory. For every 3D position in the distance texture, the closest Euclidean distance to the next non-empty element within the billboard box (defined by the 3D texture, the shell, or the thick shell) is stored. Texels that lie within a non-empty region have a distance of zero. Note that the distances define spherical regions around every texel.

At runtime, instead of using uniform sampling, we read the distance texture for the current texel and either stop (if the value is 0) or move along the ray to the border of the distance region and repeat. Therefore, the distance function can be used to efficiently skip large empty parts in a billboard box without missing an intersection. Also, since the traversal automatically stops within occupied voxels, the main loop of the traversal is very efficient, only consisting of a texture lookup and two arithmetic operations. By using this technique we obtained speedup factors between 2 to 10 for typical objects.

Blending between Representations

One problem when rendering different representations for an object at different viewing distances is to stage a smooth transition between those representations to avoid noticeable popping artifacts which inevitably draws the observer's attention to any visual differences. Due to the construction of the underlying billboard cloud, it is possible to calculate at which distance the maximum displacement ϵ encoded in a DMBBC projects to less than one display pixel. Behind this distance rendering can simply be switched to regular BBCs for faster performance, without being too much noticeable.

However, if the transition should occur at closer distances (typically for performance reasons) a blending can be applied between the DMBBC and its corresponding BBC in the following way: the displacement (i.e., the height of the billboard box) is linearly reduced to zero before switching, thus flattening the billboard box to the according billboard rectangle, similar to the geomorphing approach for terrain rendering [Hop98]. This provides a seamless transition between the two representations and requires only to dynamically adjust the size of the box and the according world-to-texture-space-matrix.

Note that we do not blend between the original geometric model and the DMBBC, as visual artifacts should be quite reduced due to the similarity of the DMBBC with the geometric model.

5.4 Results

We have implemented the variants of the DMBBC algorithm described in the previous sections as HLSL Shader Model 3.0 pixel shader and tested them with a number of objects. All of the following tests were performed on an Pentium D PC running at 3.2GHz with 1GB of RAM and an ATI Radeon X1900 XT graphics card, running Windows XP.

Figure 5.4 provides a visual comparison of various models rendered as geometry, BBCs, and DMBBCs, and Table 5.1 provides details for these models. The dragon model was chosen because its billboard cloud representations are typically visually unpleasant due its curved surface, and the DMBBC representation offers a vastly improved representation. The chestnut tree is one of the target applications of the DMBBC algorithm. billboard cloud representations are frequently used for trees, and we show that DMBBCs can provide a higher quality by eliminating the edge-on artifacts which are quite visible for trees (e.g., see Figure 5.4).

Figure 5.5: Left: BBCs of the hippo, moped and Buddha. Right: the according DMBBCs.

model	faces original	ϵ	(DM)BBC rects/boxes	generation time (s) BBC	DMBBC
hippo	31583	10	10	130	303
moped	56882	5	22	834	1014
chestnut	159160	20	8	358	806
jaguar	188844	4	31	1129	2146
buddha	865792	10	8	1692	1794
dragon	871414	4	31	2799	2934

Table 5.1: Original and BBC/DMBBC figures for various models.

	memory requirements (MB)			
		volumetric	shell	thick shell
model	BBC	DMBBC	DMBBC	DMBBC
hippo	0.37	12.2	0.76	0.93
moped	0.42	18.2	0.59	0.76
chestnut	1.85	17.25	2.67	3.1
jaguar	0.62	100	2.43	3.27
buddha	0.59	3.22	0.89	1.15
dragon	1.76	10.3	3.18	3.88

Table 5.2: Continuation of Table 5.1: memory requirements for the different representations, including normal data and distance textures for rendering acceleration.

The comparison of various models rendered as BBC and DMBBC in Figure 5.5 illustrates how the DMBBC representation manages to preserve the general shape of the models more closely.

Concerning memory requirements, Table 5.2 shows statistics about the test models. While the volumetric DMBBC needs considerably more memory than the BBC, the shell and thick shell variants store the data more memory efficient. Also note that a complex BBC still does not reach the visual quality of the according DMBBC, although it occupies even more memory. In general the tradeoff between a high image quality and low memory consumption must be made for a particular application.

The rendering speed of DMBBCs is fully determined by the pixel rendering power of the graphics hardware. DMBBCs therefore trade CPU and/or vertex processing power as well as CPU/GPU bandwidth for pixel processing speed. For Figure 5.6, we created BBC and volumetric DMBBC representations for the willow tree model with an error thresh-

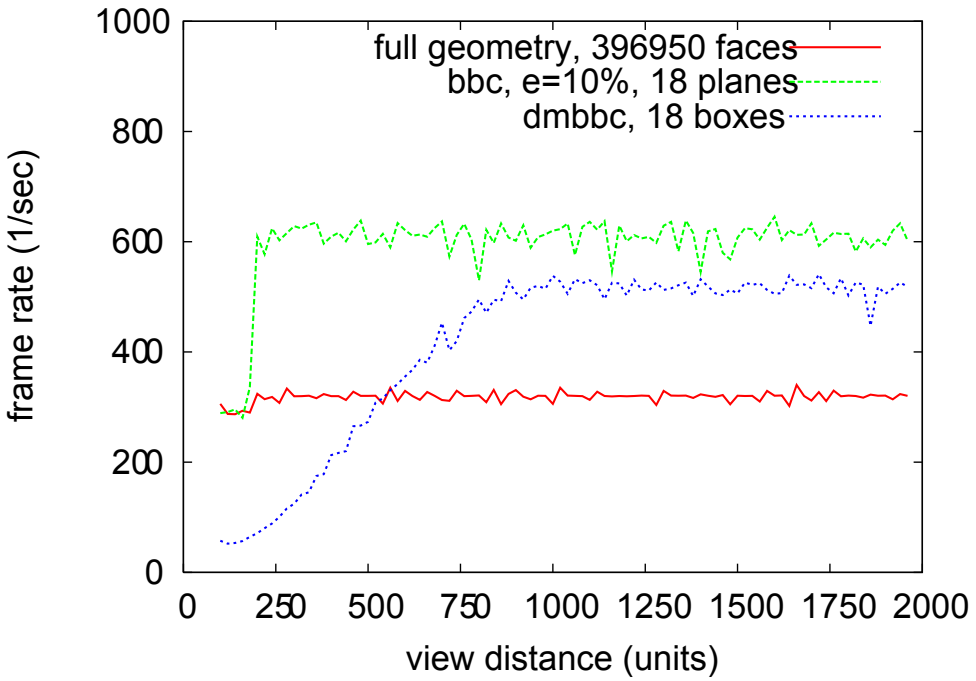


Figure 5.6: Frame rate vs. distance for full geometry, BBC and DMBBC representations of the willow tree model. Note that the top line (BBC) implies significant visual artifacts; rendering the BBC representation is therefore only useful once the visual error becomes sufficiently small (1 pixel at approximately 1300 units).

old of 20%. We then measured the average rendering time of a single such model for a number of viewpoints at distances from 500 to 5000 units in 20 unit increments. Furthermore, we performed the same test with the BBC representation of the same model, with the same error threshold as for the DMBBC, and the original geometry. Please note that the distance where the projected error threshold of the BBC becomes smaller than one pixel is at approximately 1300 units.

It can be observed that the frame rates for the full geometry and the BBC model are independent of the distance (and therefore independent of the screen size), leading to the conclusion that the GPU is certainly not fill rate limited for these models. The significantly lower performance of the visually equivalent BBC model is caused by the abundance of texture state changes required to render them—while the full geometry only uses two textures, the BBC representation requires a separate texture for each rectangle, causing a much larger overhead even if they are packed into

larger texture atlases.

On the other hand, the rendering performance of a DMBBC model is highly dependent on the viewing distance. The lower performance of the more complex DMBBCs is mostly caused by pixel shader overdraw—since the renderer modifies depth information, no early culling may be performed and the full shader must be evaluated for each of the billboard boxes. It can be seen that although the initial performance is very low, the DMBBC renderer surpasses the performance of the original geometry rather quickly. Figure 5.6 illustrates that the DMBBC representation is ideally suited for accelerating rendering at moderate distances, and provides a good transition from full geometry to BBC representations. Also note that the rendering speed is practically equal for the three DMBBC variants we have presented, because the ray casting loop is almost identical.

Furthermore, since the performance of pixel processing within the GPU is increasing rapidly with each generation of GPUs (and GPU bandwidth less quickly by far), it can be expected that the slope of the DMBBC curves in Figure 5.6 will become larger, shifting the break-even point closer to the viewer.

5.5 Summary

We have introduced displacement mapped billboard clouds, an image-based rendering primitive for complex objects. DMBBCs complement traditional billboard clouds for near to medium distances, where the visual shortcomings of BBCs are especially apparent and distracting. The main advantage of DMBBCs is that they allow an image-based representation to be used at closer distances than ordinary BBCs, leading to significantly higher rendering speeds than with the original geometry, while still providing good image quality. The improved image quality is due to the geometric detail added by image-based primitives, which also capture discontinuous geometry using so-called volumetric distance functions . The high rendering speed owes to the fact that a DMBBC is displayed entirely by modern graphics hardware with a fast ray casting algorithm. Conventional shading and shadowing techniques like normal maps and shadow mapping for realistic illumination can still be used with the new technique. In addition, different objects can intersect each other, while the representations look still correct.

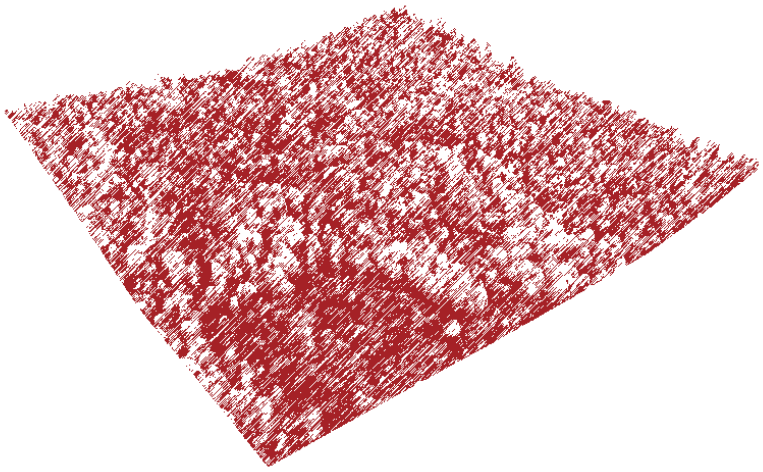
The scale of volumetric DMBBCs reaches from a purely *volumetric* representation (i.e., one billboard box for the whole model) to an al-

most purely *geometric* representation (large number of very thin billboard boxes). In other words, DMBBCs smoothly fill the gap between image-based and geometry-based representations.

In terms of future work, one interesting question is how to optimally simplify an object for displacement-mapped rendering. In our implementation we used the classical error metric of Decoret et al. [DDS03] based on the distance ϵ of a scene part to the rectangle it is mapped to. Although the displacement map practically eliminates the error measured by this metric, keeping the offsets small is still desirable in order to reduce overdraw in the output image. However, other metrics could adapt to the special characteristics of the DMBBC algorithm. For instance, while in shell DMBBCs, pixels can only adequately represent one surface piece, this does not hold true for the volumetric approach. New metrics can also be based on other optimization criteria, like using fewer boxes or making best use of memory by minimizing the number of empty pixels in every box.

PART II

FAR FIELD VEGETATION RENDERING



CHAPTER 6

INTERACTIVE LANDSCAPE RENDERING USING GPU BASED RAY CASTING

If at first the idea is not absurd,
then there is no hope for it

Albert Einstein

6.1 Introduction

Interactively exploring large landscapes has recently become quite popular. The most widely known examples for publicly available viewers are probably *Google Earth*[1] and *NASA World Wind*[2]; other well known applications are Microsoft's *Virtual Earth*[3], *Autodesk Map 3D*[4], *3DGeo*[5] and many more. These systems use images and height data obtained from laser range scanners, GPS data, and satellites to generate and render a

[1] http://earth.google.com/
[2] http://worldwind.arc.nasa.gov/
[3] http://local.live.com/
[4] http://www.autodesk.de/map
[5] http://www.3dgeo.de/

textured terrain mesh. Much work has been spent to also represent vertical structures like building facades in the representation, because such structures are typically not immediately available in the acquired data. On the other hand, to the author's best knowledge, vegetation has not been faithfully represented in such systems so far, although this obviously would contribute to a more realistic look.

We present a method for increasing the realism of vegetation rendering in landscape visualization systems. Since in most of the above systems the terrain is internally stored as height map, we directly render the landscape from this height map data using a GPU ray casting approach. We will demonstrate the following advantages of such a strategy compared to traditional mesh-based representations:

- Output sensitivity, i.e., performance depends on the number of rendered pixels rather than on the complexity of the landscape.
- It is possible to easily "refine" the landscape for a more realistic representation. This means that special surfaces in the landscape can be modeled with higher realism without the need for complex operations like remeshing the scene. An example for this is enhanced vegetation realism: given an input heightfield and an according color map, the technique extracts a special vegetation heightfield from the color data. During the terrain exploration this heightfield is rendered "on top" of the terrain thus providing parallax and occlusions like a real forest canopy. However, please note that height field data inherently becomes visible as such when viewed nearby, and the presented technique is merely thought for distant rendering only, as is typically the case for landscape explorations.
- Local and global illumination calculations like shadow mapping and ambient occlusion can be easily and quickly performed at runtime and changed dynamically. Furthermore, by exploiting natural implications about the data such as the appearance of trees, stones, glaciers etc., shadow behavior, and light interactions within a landscape, a fairly good image quality can be achieved.

6.2 Related Work

The presented work is most closely related to terrain rendering and here specifically height field rendering. For a review of vegetation rendering techniques, see Chapter 2.

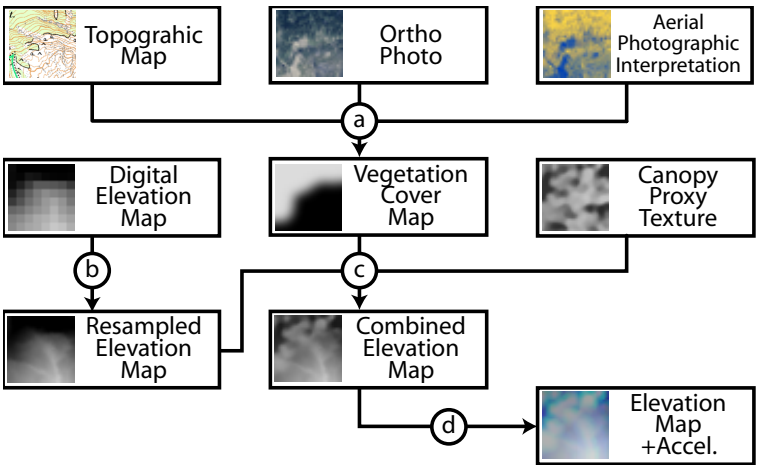
Height Field Rendering

Techniques such as bump and normal mapping have been in use to simulate the mesostructure of surfaces for many years [Bli78]. However, they lack parallax and self-occlusions within the surface. Consequently, enhanced techniques like parallax mapping [KIK+01] (optionally with offset limiting [Wel04]) and parallax occlusion mapping [Tat06] were developed that simulate the appearance of a heightfield by shifting the texture coordinates within a texture, depending on the view angle. Unfortunately, these techniques still cannot display correct object silhouettes in all cases.

Kajiya and Kay [KK89] (and later Neyret [Ney98]) first used ray casting of 3D texture maps in order to display fur and other complex surface structures. Recent advances in programmable shaders and increasing GPU performance have allowed implementations with various optimizations that run directly on graphics hardware. As an example, *relief mapping* proposed by Policarpo et al. [POC05] provides believable parallax effects within a surface, but does not render correct silhouettes. This is often not a problem for rendering object surface mesostructure. However, for landscape rendering, the undulating nature of the terrain would result in visual artifacts without correct silhouettes. To overcome this problem, Oliveira and Policarpo [OP05] proposed a method that uses quadrics to better approximate height fields on curved surfaces. This method relies on a preprocessing step that computes a quadric approximation of the surface for each vertex and stores its coefficients as additional vertex parameters. Unfortunately, the proposed approximation only works for smooth surfaces, i.e., it fails at sharp edges, and also requires the storage of additional per-vertex data, making it difficult to incorporate in existing systems.

Most existing methods that do produce correct silhouettes are based on rendering tetrahedra or prisms instead of the original surface to avoid visual artifacts. Examples for tetrahedra-based algorithms are the methods by Hirche [HEGD04], Porumbescu [PBFJ05], and Dufort, [DLP05]. Methods based on prisms have been proposed by Wang et al. [WTL+04]. Unfortunately, rendering prisms or tetrahedra significantly increases the polygon count and (even worse) overdraw.

A GPU implementation of a ray casting renderer for (terrain) height fields has also been proposed by Qu et al. [QQZ+03]. Although it lacks the acceleration methods proposed in other works, this approach is the most similar to our method.



a: Identification of Arboreous Regions and Vegetation Cover Map generation.
b: Elevation Data Resampling.
c: Height Map Texture Creation.
d: Calculation of Ray Casting Acceleration Data.

Figure 6.1: Overview of the preprocessing steps. See text for details.

6.3 Preprocess: Enhancing Landscape Detail

The goal of the preprocess component is to enhance the realism of a landscape model by adding surface detail in dependence on the represented surface.

There are several possible forms of input data. A DEM (digital elevation map) which contains the height values of the surface obtained, for instance, from SRTM (*Shuttle Radar Topography Mission*) or LIDAR (*Laser Imaging Detection And Ranging*, a time-of-flight based range laser scanning method) data. Along with the DEM there may either exist aerial photographic maps, topographic maps, and/or aerial photographic interpretation data. Note that we assume that these maps are already registered to the DEM.

The idea is to use the additional information to modify the elevation and/or photographic map in order to increase the realism in the scene. For the scope of this thesis, the focus is add detail for arboreous regions which constitute a special challenge due to their canopy structure. The goal of the preprocess is to create a *combined elevation map* from the various existing data sources (refer to Figure 6.1): by exploiting the available information, a *vegetation cover map* is generated that identifies arboreous regions. This map is then used to augment the original DEM with

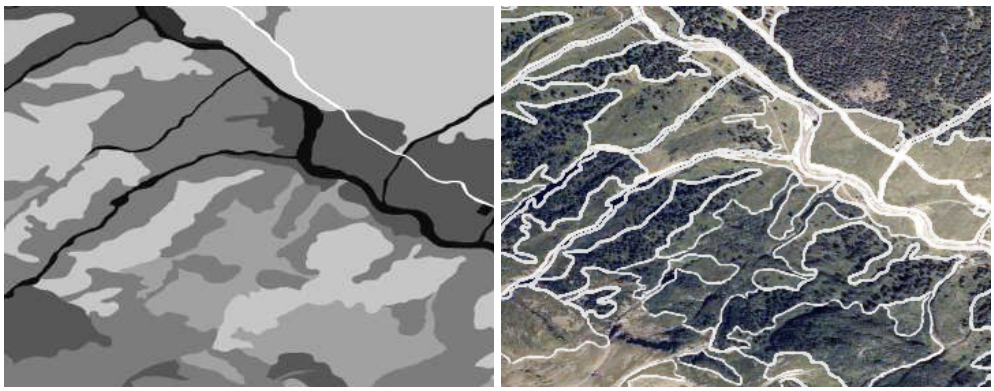


Figure 6.2: Left: Sample aerial image interpretation data. Individual shades of grey depict different land cover types (grass, forests, road, water, etc). Actual data is detailed enough to differentiate between dominant plant species. Right: Corresponding orthophotography, with surface type boundaries overlaid.

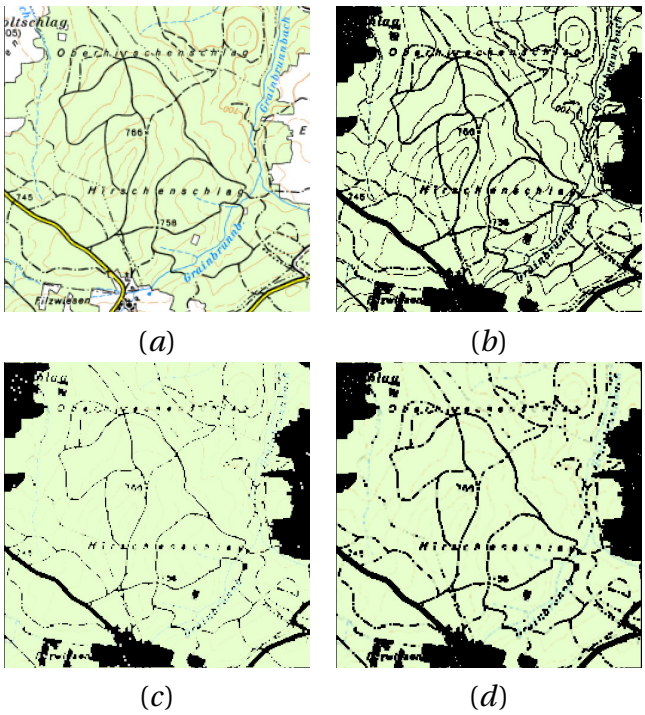
a generic displacement map (called *canopy proxy texture*) in order to geometrically enhance the forest canopy. The resulting *combined elevation map* represents both the terrain and its vegetation cover.

Please note that the basic idea to augment the original scene with a proxy appearance is also applicable to other surfaces like water, rocks, grass, glaciers and so on. Depending on the level of detail within the classification, this can be done generically (*water*, *grass*, *rock* or *trees*), or at a highly specific level (*mixed forest, dominant species larch, 60% closed*). In the latter case the proxy geometry can be varied at a much finer scale than if only a rough distinction is available.

The following sections describe the individual steps for generating the combined elevation map, including the identification of arboreous regions, resampling terrain elevation data to an appropriate level and combining terrain elevation with a suitable canopy proxy model to the combined elevation map. A schematic overview of these preprocessing steps is also depicted in Figure 6.1.

Identification of Arboreous Regions

Arboreous regions must be identified in order to generate the vegetation cover map. This can be done through a number of means. If aerial photographic interpretation or other GIS data is available, this typically already contains the required information. Each region within the infor-



(*a*) (*b*)

(*c*) (*d*)

x

Figure 6.3: Hardware accelerated vegetation coverage map creation. (a) original map; (b) vegetation areas selected; (c) after dilation; (d) dilated and eroded map. In this example, 2 passes of 3x3 filter kernels were used for the dilate and erode stages. A subsequent blur would greatly diminish the remaining artifacts.

mation database is assigned a classification ID, through which properties such as primary and secondary plant species, height and density can be queried. Figure 6.2 shows an example of aerial image interpretation data and the corresponding orthophotography from which the data was derived.

If no such data is available, the information can be extracted manually or (semi-) automatically from a given photographic and/or topographic map. For example, there are a number of methods for automatically deriving trees and other features [SH01, GL03]. If the input data is a topographic map instead of photographic material, the forested areas designated in the map can of course also be used in a similar fashion. Figure 6.6 shows an example of vegetation data derived purely from a 1:50,000 scale topographic map. In this example, the arboreous regions

were identified through color thresholding and smoothed through a morphological *close* (dilation followed by erosion) operation to combine regions separated eg. by height lines. Figure 6.3 illustrates this process.

This simplistic approach could be easily replaced by algorithms that also identify additional features from the map [AS99, MLK+04]. For example, this could be used to automatically place buildings in the model or select water surfaces for a differentiated surface shading model. In our case, the resulting *vegetation cover map* contains values between 0 for non arboreous regions and 1 for regions identified as forests.

Canopy Proxy Models and Height Map Texture Creation

Since the vegetation cover map only describes the presence of vegetation on the map but does not capture the undulating canopy surface typically seen in forests, we use a separate canopy proxy model to achieve this effect. This canopy proxy is essentially a generic, tileable patch of canopy height data based on a LIDAR scan of a deciduous forest. For large areas, we propose a tileable, non-repeating solution such as Wang tiles [CSHD03]. Although we only use one canopy proxy because our vegetation cover map does not identify individual species, the identification preprocess could be extended, allowing the use of several distinct canopy proxies for different surfaces, e.g., for deciduous or coniferous forests, or bushes.

In the ray casting step, the combined local height of the terrain and the canopy is required to calculate exact ray-canopy intersection points. To provide this information, the tileable canopy proxy model is multiplied with the vegetation cover map, and the resulting vegetation height added to the local terrain height and stored in the final texture.

An important issue in this context is the (vertical) map resolution. The DEM typically stores height values between 0 and 2000 meters (for high mountains even more), resulting in a vertical resolution of approximately 8 meters for an 8 bit map. Because this is clearly too coarse for modeling a forest canopy (for an example see Figure 6.4), the map is extended to 16 bit resolution before it is modulated with the proxy map.

Furthermore, since digital elevation data is typically available in resolutions of approximately 10-25 meters at best, it needs to be resampled at a higher (horizontal) resolution to capture individual trees in a canopy with sufficient detail. We have found 0.75 to 1.5 meters to be sufficient for this task. Note that bilinear filtering should be used for this step in order to preserve a smooth surface.

One might argue that we could have used two separate textures for

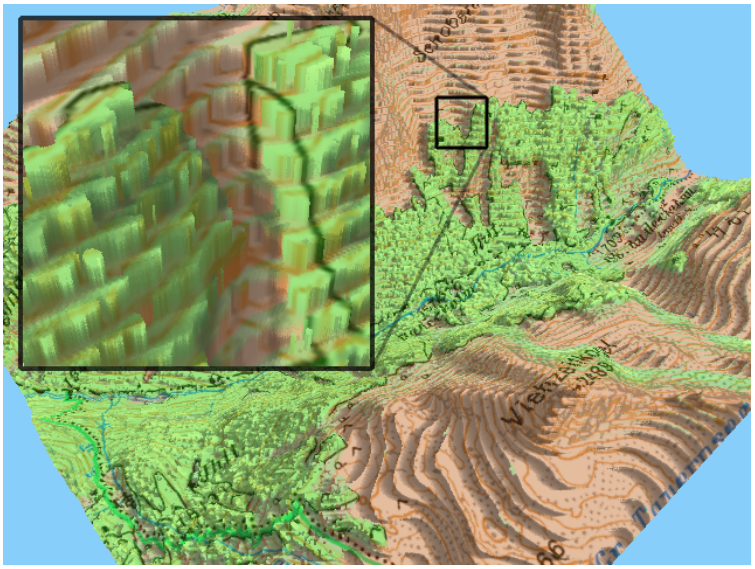


Figure 6.4: Visual artifacts as a result of insufficient vertical resolution.

the DEM and canopy proxy map and combine them in the fragment shader at runtime in order to save memory. In fact our first prototype system implemented this approach but the resulting shader was significantly (about 4 times) slower than using a single 16-bit texture. We attribute this to having to access two 8-bit textures per ray casting step. Although the total data per access is the same for two 8-bit textures as for one 16-bit texture, accessing a second texture apparently results in much worse cache coherence. Furthermore, the inner loop of the shader becomes more complex (2 texture + 12 arithmetic operations compared to 1 texture + 9 arithmetic operations, a 40% increase). Although this approach significantly increases the required memory, we do not expect this to be an issue for large scale terrain rendering, because such applications typically already support dynamic texture loading and unloading.

Hardware accelerated preprocessing

The identification of arboreous regions - especially from topographic maps - can be performed on the GPU very quickly. Given a topographic map and a color key that identifies the specific regions (typically a shade of green), the vegetation cover map can be calculated in a few passes, each rendered to an off-screen texture that is used as input for the next step. The individual steps are essentially the same as illustrated in Figure 6.3.

The dilate and erode passes may be performed multiple times to simulate a larger filter kernel that would exceed the maximum number of allowed texture accesses in a single pixel shader program. Similarly, the 3x3 filter kernel can be decomposed into two passes of a 3x1 and 1x3 filter, respectively. [6] Similarly, the combined elevation maps and acceleration data structure can also be created very efficiently on the GPU. Specifically, the safety zones introduced by Kolb et al. [KRS05] can easily be produced by a sequence of dilations operating on the individual color channels.

6.4 Runtime: Interactive Landscape Rendering

Given the DEM, enhanced with the canopy geometry as described in Section 6.3, the goal is to interactively display the landscape with realistic illumination. The rendering algorithm is based on per pixel ray casting using the GPU.

Due to its high resolution (at 0.5m resolution, a $1km^2$ area amounts to 2000x2000 samples, or roughly 8 million triangles) it is not practical to directly convert it to a triangular mesh for rendering. Of course any number of geometric reduction algorithms could be applied to create a less detailed representation. However, retaining full detail is typically desirable, eg., to identify outliers and systematic errors from the data acquisition process. We therefore propose to render a very coarse approximation as a convex hull, and to 'fill in' details as needed through an exact ray casting algorithm. Furthermore, the presented pixel shader approach allows an easy adaptation to different rendering styles and illumination methods (just by slightly changing the shader program). Note that this would be much more tedious with a fully geometric representation.

Choice of Rendering Primitives

At runtime a low-polygon approximation of the terrain is sent to the graphics card. The generated pixel fragments are used as starting points for ray casting within the pixel shader. The ray caster then uses the canopy map to derive exact visibility and optionally compute surface color and illumination.

[6] Note, however, that morphological operations are not commutative. For a dilation operation D and erosion operation E, the sequences D, D, E, E and D, E, D, E yield different results. The optimal combination depends on the input image and requires some experimentation.

Since the geometric representation of the terrain is only used to create the fragments for the pixel shader ray caster, it can be relatively coarse. In fact, even rendering a simple plane in front of the camera would be sufficient for this task. On the other hand, the geometry defines the starting point for the ray caster in the map. It is desirable to generate a starting point close to the landscape surface so that an intersection of the viewing ray with the surface can be found quickly. Many landscape exploration systems already use geometric levels of detail for the terrain, and fortunately these approximations always results in starting points close to the surface.

Since the geometry must encompass the terrain as well as the vegetation on top of it, an appropriate vertical offset is necessary. This can be easily performed in the vertex shader, which also allows a smooth transition to other rendering methods that are not based on displacement mapping by progressively reducing the vertical offset and thickness of the rendered vegetation according to the view distance. Once the vertical offset is zero, the ray casting step can be skipped and the surface is rendered using traditional texture mapping.

Basic Algorithm: Per Pixel Ray Casting

We use a ray casting step that intersects the viewing ray against the combined elevation map, quite similar to the algorithms described in previous work [QQZ+03, HEGD04, POC05]. Every vertex is assigned a 2D texture coordinate for the combined elevation map. The per fragment 2D texture coordinate (generated during the hardware rasterization) is the starting point for the ray casting algorithm in texture space. For simplicity, we define the landscape in world space over a (x, y) plane and z points up. The 2D direction in texture space is then simply defined by the (x, y) coordinates of the viewing ray in world space. Beginning with the starting point, the ray caster compares the value (the *height*) in the combined elevation map with the respective z value of the viewing ray at that position. If the ray is *lower* than stored height, an intersection with the heightfield occurred and the ray caster stops. Otherwise, the next position along the ray is evaluated using a user-defined distance from the current position. This process is repeated for a user-defined number of steps. If the surface has not been hit, the fragment is clipped in order to provide correct landscape silhouettes.

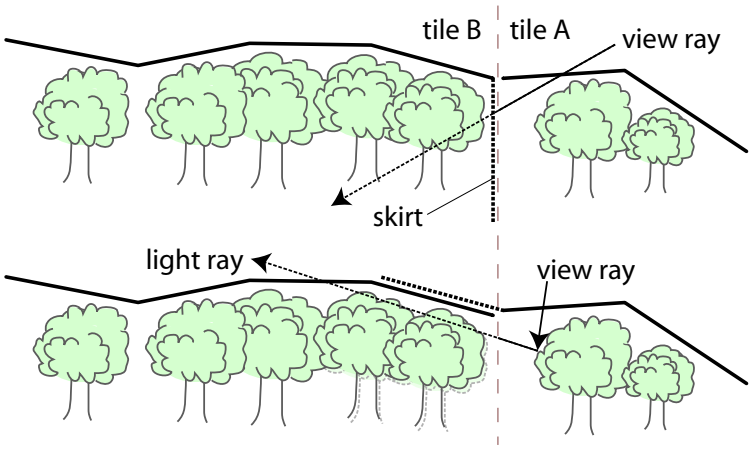


Figure 6.5: Avoiding artifacts at tile boundaries. Top: Vertical 'skirts' (dashed) must be added to account for view rays entering the tile below the top surface. Bottom: Some data (dashed) is replicated between tiles to allow for illumination calculations across tile boundaries.

Rendering Quality Considerations

Per-pixel displacement mapping algorithms are often associated with rendering artifacts such as texture distortion and/or incorrect silhouettes. For correct ray casting of a displacement map on arbitrary objects, the ray would need to be distorted appropriately to capture the curvature of the object. This problem is typically solved by using a piecewise linear (or, in the case of Oliveira's work [OP05], quadric) approximation for each individual triangle of the object. Triangles of the original mesh are rendered as prisms or tetrahedra to guarantee continuity and to avoid artifacts due to missed intersections. The use of these primitives also allows the computation of the exit point of a ray from the current prism or tetrahedron. By knowing both the start and exit point, the number of ray casting steps required for full coverage can be easily computed.

In our case, object curvature issues are obviated, because the displacement map stores absolute height information relative to a ground plane, i.e. the original mesh can be regarded as a plane with zero curvature. Therefore, no conversion to prisms or tetrahedra is required. However, given a very large terrain the number of ray casting steps required for full coverage of the terrain may be excessively large. To avoid this problem, we subdivide the terrain into tiles that are rendered individually. Each tile has a vertical boundary 'skirt' to avoid missed intersections, which can be directly compared to using prisms on a per-triangle level. Fortu-

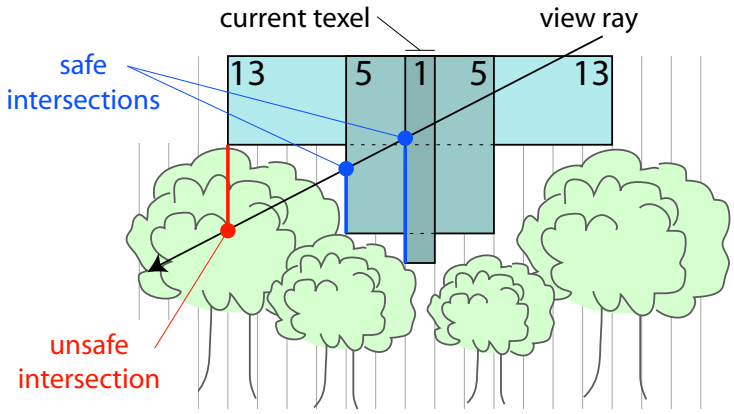


Figure 6.6: Ray casting acceleration. In addition to the local height (=1 texel safety zone), two safety zones (brighter green boxes) are stored for each texel. The maximum step size is determined by the largest box that is not intersected in its floor.

nately, similar skirts are also used in many tile based terrain rendering techniques to hide T vertices and other tiling artifacts [Ulr02, NAS], and such geometry could be re-used directly for our approach.

Also note that the distance between two ray casting steps as well as the overall number of steps provide simple means for balancing rendering speed and image quality.

Ray Casting Acceleration

Our per-pixel ray caster is based on the acceleration scheme proposed by Kolb et al. [KRS05]. For every texel, a so-called *safety zone* is defined, being the largest height value within a user-defined radius around that texel (see Figure 6.6). The safety zones for all texels are computed and stored in a preprocess. At runtime, the minimum height of a ray within the radius of the current texel is computed using some simple (and fast) math. If this minimum height of the ray within the radius is above the maximum height, it is guaranteed that the ray does not intersect the height field within the current safety zone and the next sampling point can be placed just outside the zone instead using a constant distance as described above. It is also possible to precompute *multiple* safety zones per pixel and to evaluate them in parallel at runtime by exploiting the vector arithmetic of graphics hardware. As an example, in Figure 6.6 the

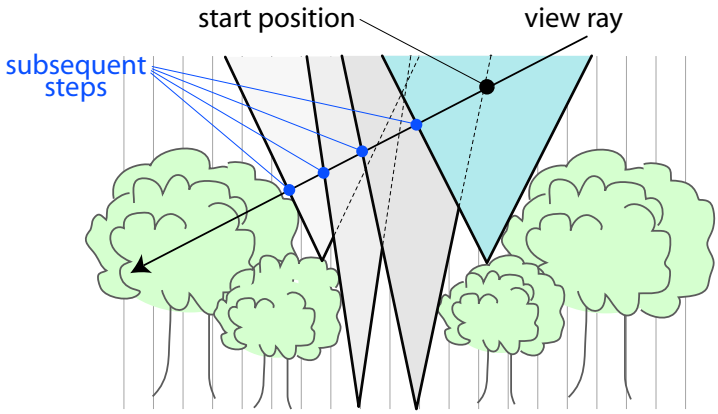


Figure 6.7: In cone step mapping, the ray step distance is determined by intersection with cones of free volume that are centered on the height field. If the height field includes narrow openings, the cone opening angles are small and performance decreases significantly. Compare Fig. 6.6.

ray intersects the two inner safety zones (the height of the current texel as well as the first (5 texel) safety zone) at their sides (blue dots, respectively) whereas the second (13 texel) safety zone is intersected through the floor (red dot). Therefore, the largest safe step size is determined by the medium box (left blue dot). Although Kolb et al. use three parallel safety zones, experiments showed that the performance gain of two safety zones is hardly lower compared to three zones, so we only used 4 and 32 texel safety zones. The last component was then used for the illumination calculation.

The choice of ray casting acceleration scheme is heavily dependent on the features of the elevation data. Since forests often include narrow paths and other gaps of open space, cone-based acceleration schemes (for example, Cone Step Mapping [Dum06]) may lead to bad performance in these cases (see Figure 6.7). This is especially true when viewing a scene at near horizontal angles.

Similarly, volumetric approaches such as Donnelly's Distance Functions [Don05] are either very coarse (if a low vertical resolution is used), leading to a small maximum step distance, or require very large amounts of texture memory as well as significant preprocessing.

Illumination Calculation

In some situations it might be desirable to relight the scene, for instance, when the sun should be simulated over a whole day. In order to get decent lighting, we first experimented with recovering local normal vector information and calculating diffuse illumination in addition to ambient occlusion and self shadowing. Unfortunately, this resulted in a quite unrealistic, 'plastic' look of the forest. We attribute this to the fact that the reflectance properties of forest canopies are hard to capture with a diffuse reflection model, and just using an ambient solution seems to be the best compromise if short of using a BRDF from measured data [MUN91]. Consequently, the color read from the orthophoto map (or, respectively, topographic map) at the intersection between view ray and elevation map is defined as base color.

In order to shadow the scene, the ray casting approach allows for a simple and efficient shadow computation which avoids implementing shadow mapping or shadow volumes [AMCH+04]. Beginning from the intersection between view ray and elevation, a ray is constructed to the light source (several light sources are of course also possible, requiring an individual ray to every light). The ray casting process is then basically performed as was described above for the visibility calculation. If the ray intersects the scene, the current pixel is in shadow. Otherwise, it is lit. Unfortunately, we have found that in the case of forests and trees, this produces very hard shadows that look not quite natural. Instead, we accumulate shadowing over several steps similar to ray casting of volumetric data [KH84] and in the spirit of the *deep shadow map* technique by Lokovic and Veach [LV00]. To also capture the contribution of distant scene parts (such as distant mountains), the step size is increased linearly, such that at close range, occluders are finely sampled and cast darker shadows whereas more distant occluders have a smaller contribution to the final shadowing. This results in visually pleasing shadowing, and the diminishing contribution can be likened to a pseudo ambient occlusion solution (where more distant occluders are outweighed by a dominating ambient illumination). See also Figure 6.14.

Also note that typical orthophotographic maps already contain shadows, which must be removed beforehand. Premoze et al. [PTS99] discuss efficient methods for this task. Figure 6.15 shows an example for such renderings.

When rendering tiled landscape data, care must be taken to avoid visual discontinuities at tile boundaries. This can be seen as the inverse problem of ray casting across tiles (which was solved by adding bound-

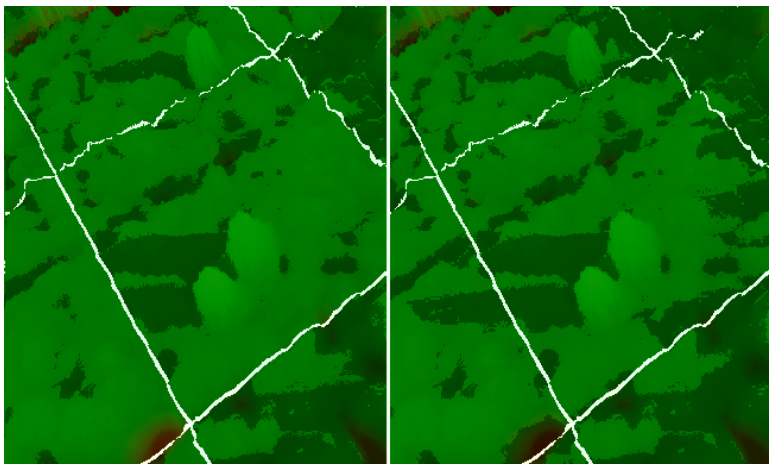


Figure 6.8: Artifacts from incorrect shadowing across tiles. Left: regular tiles (no overlap). Note that no shadows are cast across tile boundaries. Right: tiles overlap as described in the text to produce correct shadows. Very small tiles (64x64) have been used for illustration.

ary skirts to provide additional input data). Here, the intersection point is found on the local tile, but the light ray should sample a neighboring tile for shadowing (see Figure 6.8). To account for this, we propose to produce overlapping tiles such that local influences of neighboring tiles can still be captured (see Figure 6.5, bottom).

Tiling artifacts are not completely avoided by this approach, but they are greatly reduced since distant occluders already contribute less to our illumination heuristic.

In some cases, the additional boundary data could also be used to reduce the artifacts described in the discussion on rendering quality considerations, but typically vertical skirts must still be used to avoid gaps between tiles with different mesh resolutions.

Other Surface Types

The existing land cover information, or even simple topographic maps, can be used to differentiate between multiple surface types. We encode this surface type information in one channel of the combined elevation map. After the correct intersection point has been determined using the GPU ray caster, the local surface type can then be used to decide which kind of surface should be rendered. As an example, we use this information to switch between a "plain" renderer (directly using the topographic

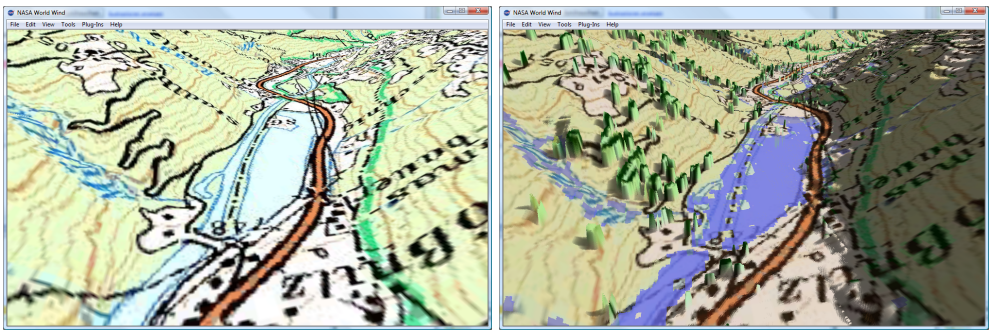


Figure 6.9: Land cover data derived from aerial photography is used to switch between different surface shading models. Left: plain topographic map. Right: Land cover dependent shading. Notice reflections on lake, surface color changes and sparse vegetation.

map), a "tree canopy" representation (adding a green tint to vegetation), and a "water shader" for rivers and lakes (which calculates reflections). See Figure 6.4 for a visual comparison of the original topographic map and a rendering based on different surface types.

Note that care must be taken to disable linear interpolation when reading the local surface type from a texture, or exact texel coordinates must be used. Otherwise the interpolation will likely result in incorrect return values.

6.5 World Wind Integration

To demonstrate how our method can be integrated in an existing geospatial viewing application, we have adapted the NASA World Wind viewer for this purpose.

NASA World Wind

The *World Wind* application was developed by the NASA Learning Technologies group and initially released in 2004. It is a geospatial viewer that enables the user to interactively explore Earth and other planets such as the Moon, Venus or Mars (see Figure 6.5).

World Wind was designed to render geospatial imagery from Blue Marble and Landsat datasets in combination with terrain data derived from the Shuttle Radar Topography Mission (SRTM). In addition, the standard viewer incorporates a series of additional layers such as USGS

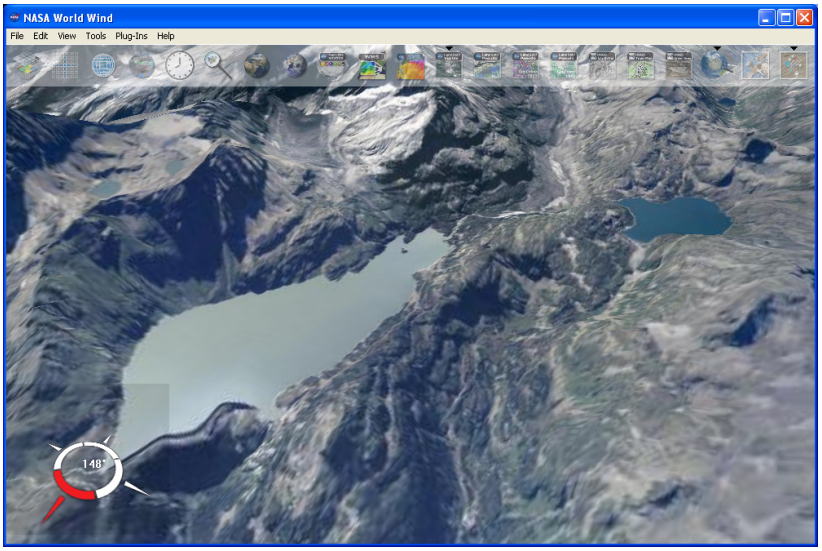


Figure 6.10: The World Wind application

topographic maps of the United States and aerial images down to 0.25m resolution for urban areas.

The development model of World Wind is extremely open; source code is freely available, and any developer is free to build upon it and create custom variants of the application. Naturally, *commit* access to the source code repository is restricted. Even without source code development tools, user defined layers can be easily incorporated via XML configuration files.

Geospatial Data in World Wind

Although World Wind excels at rendering geospatial raster data, support for vector information is available and can be used to draw boundaries and place names. Since our interest lies in terrain rendering, our discussion will focus mostly on how geospatial images are handled in World Wind, although any real-world application will of course incorporate vector features.

In World Wind, the globe is constructed from a small number of base tiles. Each of these tiles is dynamically subdivided in a quadtree hierarchy (dubbed *QuadTiles* in World Wind). The collection of such hierarchies and its individual nodes is called a *QuadTileSet* in World Wind. The subdivision level is determined at runtime to keep the screen space error below a given threshold (see Figure 6.5). The hierarchy supports out of

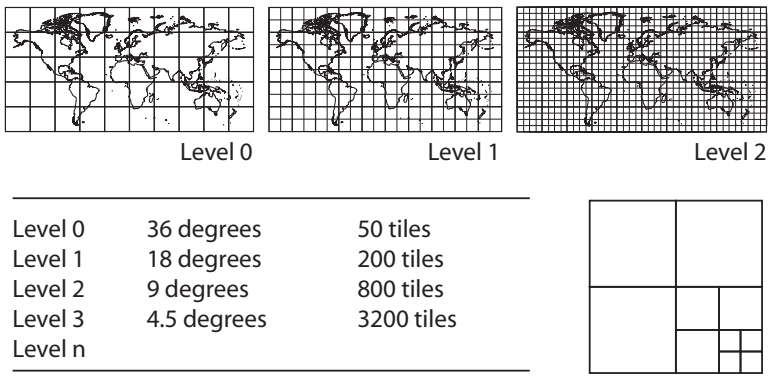


Figure 6.11: In World Wind, the globe is internally stored as a quadtree data structure

core data, i.e. higher detail subdivision levels are only loaded as needed for rendering. This applies to both image and elevation data.

The data can be fetched from disk or over the network via WMS (Web Mapping Service) requests. World Wind itself does not handle georeferenced data and instead relies on the data to be properly formatted to match the quadtree subdivision. This is implicit to the WMS requests, and external tools exist to prepare arbitrary image data accordingly.

At runtime, layers are rendered sequentially and may occlude each other. For each layer, the QuadTile hierarchy is traversed until either the screen space error is sufficiently small, or no data is available at the requested level. This may be either due to inadequate resolution of the available data, or because the required terrain and texture data has not yet been fully loaded.

The original World Wind renderer supports a few basic features such as layer opacity and dynamic illumination (and, naturally, texturing although only a single texture unit is typically used).

Only two adaptations were required to integrate our method: The rendering pipeline was extended to support programmable shaders, and the tile management system was changed to allow multiple textures per QuadTile such that the various maps could be made available to the shader.

Preprocessing via Proxy Servers

Since our goal is to provide detailed landscape rendering capabilities on a global scale, preprocessing and storage of the additional data required

for our renderer is not an adequate option. The data would consume several gigabytes of disk space and take several hours to preprocess. Therefore, we have chosen to implement a system that creates the required maps on demand through a proxy tile server. This minimizes the changes to the existing application, reduces the bandwidth required by the client application (since only the completed maps need to be downloaded), and allows multiple clients to share the preprocessing cache of the proxy tile server.

Since most servers do not have adequate GPU support for hardware accelerate preprocessing, our proxy server uses a purely CPU based approach. Depending on the server load, requests for uncached tiles require up to several seconds.

6.6 Results

We have implemented our method as a C# / Managed DirectX application using HLSL shader model 3.0 shaders. Screen shots and performance figures were taken on a 3.2GHz Pentium D PC with 1 GB RAM and an ATI Radeon X1900 XTX GPU with 512MB DDR3 memory.

For performance evaluation, we acquired sample data of a 3.5km x 3.5km section of the *Nationalpark Hohe Tauern* (Hohe Tauern national park, NPHT) in Austria. A digital elevation model, topographic map data and an aerial photography interpretation was available for the entire area, whereas the orthophotographic data at hand only covered the inner 2.5km x 2.5km subset. The original DEM data had a resolution of 10m (350x350 texels) and the orthophotographic data was provided at a resolution of 1.25m (2000x2000 texels).

Our canopy proxy texture was derived from a LIDAR scan of a 500m x 500m coniferous forest. The scan data was converted to an elevation map with 0.5m resolution (1000x1000 texels) and afterwards manually converted into to a large, tileable texture. The output resolution of our canopy proxy was 2048x2048 texels for the entire area (1.7m resolution), which was also the resolution of the combined elevation map.

The aerial photography interpretation included a detailed classification including vegetation density, dominant and subdominant species and underlying terrain structure. In theory this could be used to synthesize a highly detailed combined elevation map that also differentiates between various tree species, coppice, rocks and grassland. However since we only had a single canopy proxy texture we simplified the classification to only discern between arboreous and other areas.

Although aerial photography interpretation is based on orthophotographic data, the identified areas did not correspond *exactly* to the provided orthophoto. We attribute this to the interpretation being a partially manual process, such that small border details may be missed or consciously omitted. Similarly, the topographic map is an even stronger abstraction of reality where the correspondence of identified forests to 'reality' is an even coarser approximation.

For the evaluation of our integration in the World Wind viewer, we have chosen to integrate the entire data set for the Hohe Tauern national park. In total, the orthophotographic and aerial image interpretation data covers approximately $1700km^2$. In addition, we obtained the complete *ÖK50* $1:50000$ scale Austrian topographic map set, allowing very large scale testing of our automatic vegetation extraction methods.

Visual Quality

The main benefit of our rendering method is the visual detail provided by the forest canopy. Effects such as correct parallax are hard to capture in screen shots, however for example Figure 6.6 shows how a terrain textured with a topographic map can be visually enriched.

Figure 6.12 displays (from left to right) the same view rendered as a plain textured landscape, with textured with illumination, and with our approach. The benefits of adding vegetation to the rendering are clearly visible. In Figures 6.12 and 6.17, slight misregistration artifacts between the orthophoto and the forest canopy can be observed. This is a result of the approximate nature of the (manual) aerial photographic interpretation used for these models, which did not capture the exact forest boundaries.

In Figure 6.18, the orthophotographic information was embedded in a larger area and additional topographic data was used to create a canopy model that covers both the detailed orthophoto and the surrounding area. This is a versatile means of providing a meaningful setting for detailed landscape data.

Examples for dynamic illumination are also presented in Figures 6.14 and 6.15. For Figure 6.15, the original orthophoto contained illumination information which was removed manually. Of course this is not practical for larger areas, however automatic algorithms exist for this task [PTS99].

The possibility of adjusting the performance vs. quality tradeoff of our approach is illustrated in Figure 6.16. The left image is rendered with 30 ray casting steps and exhibits some artifacts near the horizon as depicted in the magnified inset, where no intersection could be detected

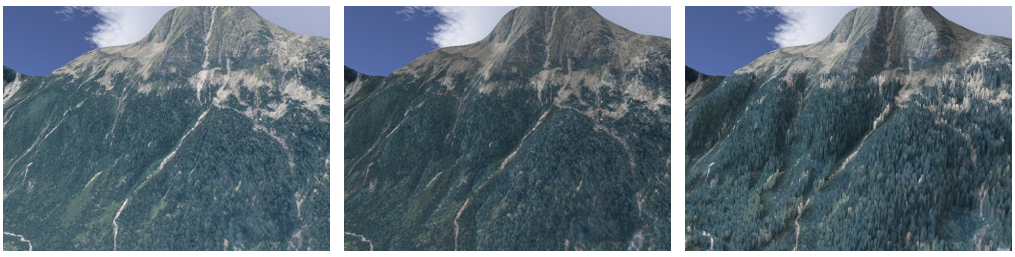


Figure 6.12: Rendering a landscape with orthophotographic data. Left: textured and unlit. Center: textured and lighted. Right: The new GPU ray casting approach.

with the limited number of intersection tests. On the right, a higher step count (70) is used, capturing the missing intersections and providing a much higher visual quality.

Performance Evaluation

To evaluate the rendering performance of the proposed method, we defined a camera path through the terrain and calculated average frame rates for various parameter values. Note that frame rate variance was low because the performance mostly depends on the number of rendered pixels which was constantly high. All frame rates were measured with the application running full screen at 1280x1024 pixels. Given the output sensitivity of our approach, smaller screen sizes (e.g. in a windowed application) have correspondingly higher performance. For example, we achieved an average frame rate of 15.4 frames per second (fps) in full screen mode for 90 view ray intersection steps + 30 shadow ray intersection steps, and in windowed mode (640x480 pixels) with the same parameters the application ran at 24fps.

The lowest setting that produced visually acceptable results in full screen mode used 30 steps for intersection calculation, and 10 steps for shadowing accumulation (see Table 6.1). With a 2048x2048 texels combined elevation map, we achieved an average frame rate of 35fps. In contrast, a very high setting (70 intersection steps) still ran at 30fps. Only after adding significantly more shadowing accumulation steps (70 + 30), the frame rate dropped to approximately 22fps.

We did also experiment with higher resolution canopy proxy maps and combined elevation maps (0.85m, 4096x4096 texels). However, this did hardly improve the visual quality while causing a significantly reduced rendering performance (see Table 6.1) due to an increased number

	2048x2048			4096x4096		
	10	20	30	10	20	30
30	35.4	29.0	24.1	25.3	20.7	19.1
50	32.2	27.0	22.6	23.1	19.1	16.2
70	30.1	26.0	21.9	22.1	18.4	15.5
90	29.8	25.1	21.7	21.5	18.0	15.4

(down : intersection steps. across : shadow steps)

Table 6.1: Average frames per second for 2048x2048 texels and 4096x4096 texels combined elevation maps with various values for ray casting intersection and shadow accumulation steps.

of necessary ray casting steps in order to calculate the view ray/elevation map intersection point. In addition, higher resolution elevation maps increase the time required for preprocessing (between 80 and 180 seconds). However, the required time can still be called fairly short, and our preprocessing code leaves much room for further optimizations if preprocessing time becomes an issue.

6.7 Summary

We have demonstrated an interactive technique for realistic rendering of landscape data. The work focussed on enhanced representations for vegetation without the need to model each plant individually. Instead, the approach combines a generic tileable canopy proxy model with landscape specific information to create a height map that can be rendered directly on the GPU. Although it may be possible to achieve a comparable approximation of vegetation by creating a purely geometric approximation (ie. a suitably fine, adaptively tesselated mesh), our solution has several advantages over such an approach. Its integration into existing terrain rendering frameworks is quite straightforward, only requiring one additional texture and the vertex and pixel shader. Changing the appearance of the rendition is very simple by adapting the shader program. Also, the presented approach requires a fairly short preprocessing time and the ray casting shader implementation is relatively simple.

In comparison to other existing methods for rendering vegetation on large landscapes, our approach requires little preprocessing, is mostly independent of the terrain rendering system, provides shading and shadowing, and achieves interactive frame rates even for large scenes.

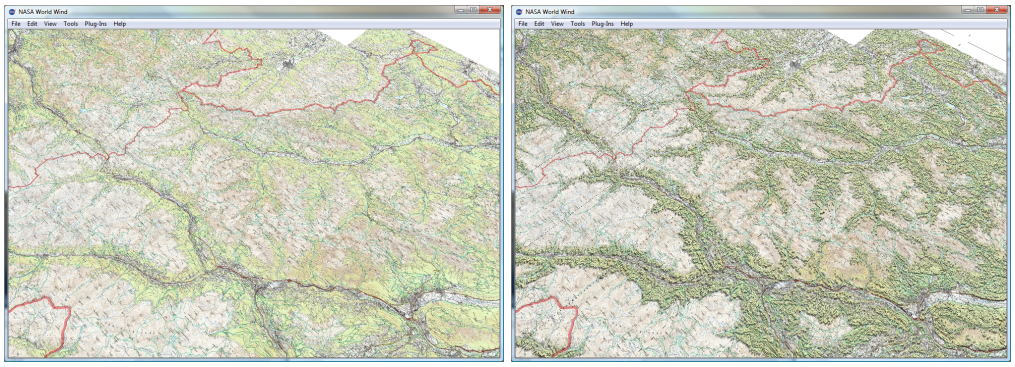


Figure 6.13: Topographical Maps as usually displayed in World Wind, compared to our rendering

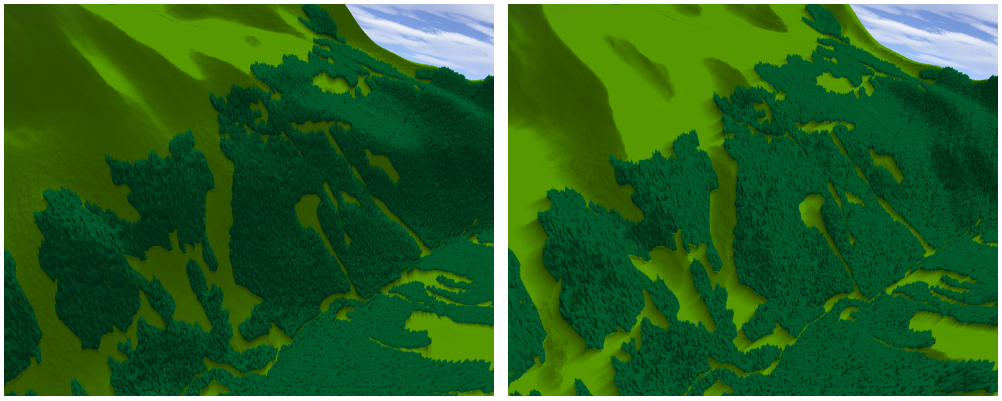


Figure 6.14: Example for dynamic coloring and illumination of a scene derived from a topographic map.

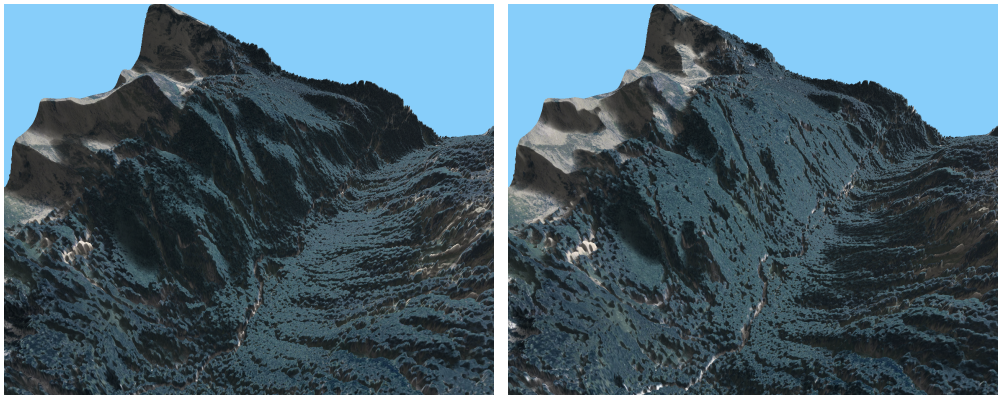


Figure 6.15: Dynamic illumination of orthophotographic data

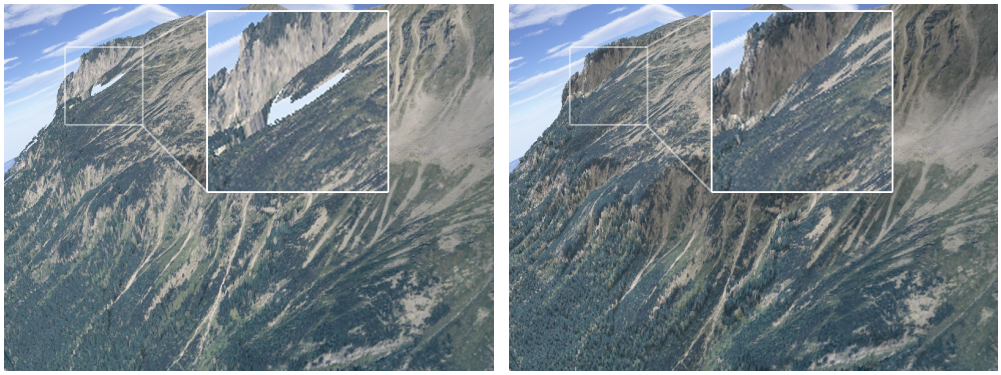


Figure 6.16: Artifacts caused by too few ray casting steps (left) and the same scene rendered with a higher number of steps (right)

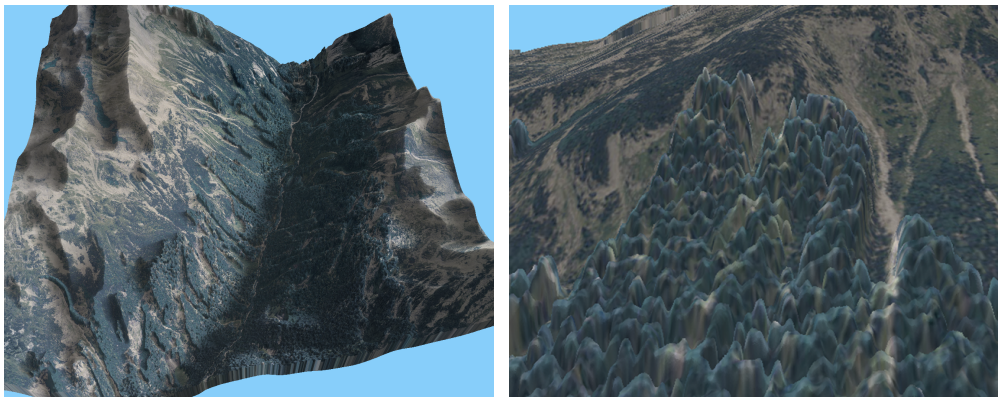


Figure 6.17: (left) Visualization derived from orthophotos, aerial image interpretation and digital elevation model. (right) close-up view.

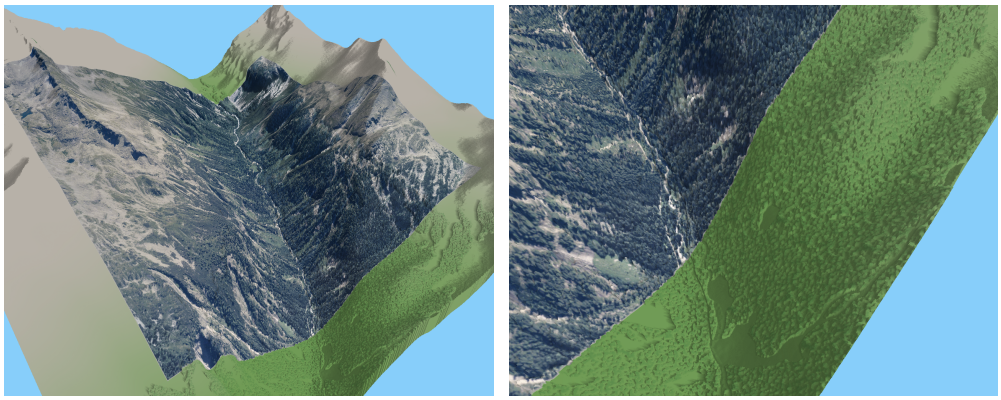
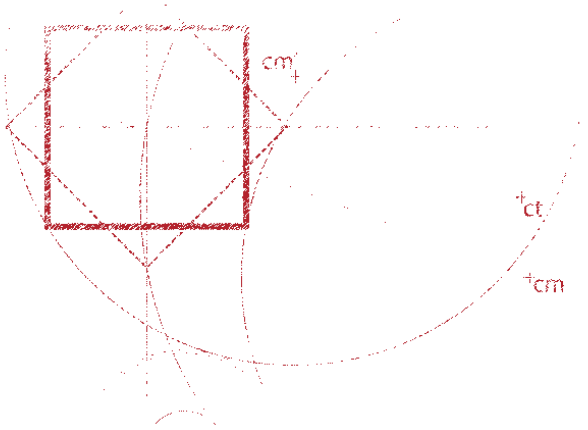


Figure 6.18: examples of an orthophotographic image embedded in a larger model, with continuous canopy information.

PART III

LEVELS OF DETAIL AND IMPOSTOR VALIDITY



CHAPTER 7

FAST AND PRECISE TESTING OF DYNAMIC IMPOSTOR VALIDITY USING A TWO-LEVEL CHECK

> All science is either physics or stamp collecting.
>
> Ernest Rutherford

7.1 Introduction

Impostors are becoming an integral part of many current real-time rendering systems. Complex objects are rendered to textures, which are then displayed instead of the original models. This often results in significant rendering speedups. A popular option is to generate impostors dynamically at runtime for the viewer location. If the viewer moves too far away from this location, the impostor becomes invalid (when the visual difference to the original model exceeds some threshold), and it is updated. Because rendering the object to a texture is a rather slow operation, it is highly desirable to update impostors as seldom as possible. The *validity* of the impostors is determined by an error metric, which typically estimates the geometric error introduced by the impostor compared to the original object [Sch95].

The *hierarchical image cache* that was concurrently presented by Schaufler and Stürzlinger [SS96] and Shade et al. [SLS+96] combines dynamic impostors with a hierarchical scene organization (for instance, an octree). A dynamic impostor is generated for hierarchy nodes where a speedup is expected. While the leaf nodes are generated directly from the original (geometric) representation, higher nodes are composed recursively from the impostors of their child nodes. The technique takes advantage of the fact that lower nodes are always valid longer, since parallax movement is higher for higher nodes. Consequently, for many output images, only the impostors for higher nodes need to be updated from their child impostors, which is much faster than generating them from the original geometry. Rendering the scene is done by traversing the hierarchy and displaying the highest available impostors the hierarchy, so that subtree traversal can be skipped. The hierarchical image cache provides fast rendering of very complex scenes, as long as the viewpoint is not changed too rapidly.

Having a correct and fast test whether an impostor is valid is crucial: on the one hand, it should be accurate in order to avoid unnecessary impostor updates (this is a costly operation). On the other hand, the computational cost for the test should be reasonably low. These demands apply especially for the hierarchical image cache, because its overall efficiency strongly depends on the number of impostor updates, and numerous impostors must be tested every frame. Therefore, we introduce a two-level validity test. The first stage provides a fast conservative estimate for impostor validity. If the first stage fails (i.e., the impostor cannot be classified as definitively valid), a second stage performs an exact validity test in order to avoid updates for valid impostors. In addition, for the hierarchical image cache, the first (conservative) stage can be used to quickly determine the validity of all impostors in a subtree, which may significantly increase the efficiency of the impostor update traversal. This means the advantage of the two-level test is that both the overhead required to check for validity and the number of impostor updates per frame are reduced.

7.2 Efficiency of Impostor Error Metrics

Evaluating the exact screen-space error of an impostor is prohibitively expensive, since every vertex of the original object would have to be projected onto the screen for each new viewpoint (which is ass slow as rendering the object directly). Thus, impostor error metrics typically use the

bounding box or sphere of the original object to provide a conservative, but faster estimate. In the following, we will analyze the costs of the two-level metric compared to the standard metric.

For a metric m, we define t_m as the time spent evaluating the metric, and i_m as the fraction of the total number of impostors it classifies as invalid. The total number of impostors in the scene is n, and t_u is the time required to update an impostor. We then define the *cost* c_m of a metric m as the time spent on evaluating the metric plus the time for the impostor updates:

$$c_m = n \cdot (t_m + i_m) \cdot t_u) \tag{7.1}$$

The cost of a metric can be reduced by making its evaluation cheaper, or by reducing the number of impostors it classifies as invalid. In contemporary systems, the time required to update an impostor is very large compared to the time spent on evaluating the metric ($t_u \gg t_m$). At the same time, the extent of common scenes is constantly growing, which means that the impostor metric has to be evaluated very often. Since far away impostors are updated less often than nearby ones, fewer impostors have to be updated with increasing scene extent. That means we require a metric that is both fast to evaluate and as precise as possible.

7.3 A Two-Level Impostor Validity Test

Instead of using a single metric, we propose to use a two-stage evaluation scheme. The first stage m_1 is a fast test that classifies the impostor as *valid* or *uncertain*. The second (more costly) stage m_2 uses a more precise error calculation to resolve uncertain cases. Ideally, most cases will be resolved by the first check alone. The performance of such a two-level metric can then be described as the time for evaluating m_1 plus the time for evaluating m_2 on the uncertain cases plus the time for updating the impostors:

$$c_{m_1+m_2} = n \cdot (t_{m_1} + i_{m_1} \cdot t_{m_2} + i_{m_2} \cdot t_u). \tag{7.2}$$

The cost difference between the two-level metric and using only the second stage alone is:

$$c_{m_2} - c_{m_1+m_2} = n \cdot (t_{m_2} \cdot (1 - i_{m_1}) - t_{m_1}) \tag{7.3}$$

Note that this *gain* in cost neither depends on the impostor update time nor on the total rendering time. The two-level scheme is desirable when the gain is positive, or:

$$t_{m_2} \cdot (1 - i_{m_1}) > t_{m_1} \tag{7.4}$$

This basically says that the two-level scheme is better if the first stage is faster on all impostors than the second stage on the certain impostors.

Construction of the Metric

For the first stage, we exploit the fact that changes of the viewpoint position are rather small in consecutive frames. This allows the establishment of a *Guaranteed Valid Volume (GVV)*, which is a simple volume (in our case a sphere) that contains those viewpoints the current impostor is guaranteed to be valid for. The main assumption is that once such a volume has been found, it is very likely that it will contain the viewpoint for the next few frames. The construction of this sphere is based on the error metric proposed by Schaufler [Sch95]. Since this metric is rotationally symmetric along the view direction, the construction can be done in 2D as is illustrated in Figure 7.1: translation and move-in error thresholds are constructed as the (circular) set of equiangular points defined by the maximum distance between the original and impostor model. The GVV radius can then be found from the distance of the current viewpoint to the closest of these circles. Note that due to the symmetry of the construction and since only the relative position of v' to v is important, only three radii need to be calculated.

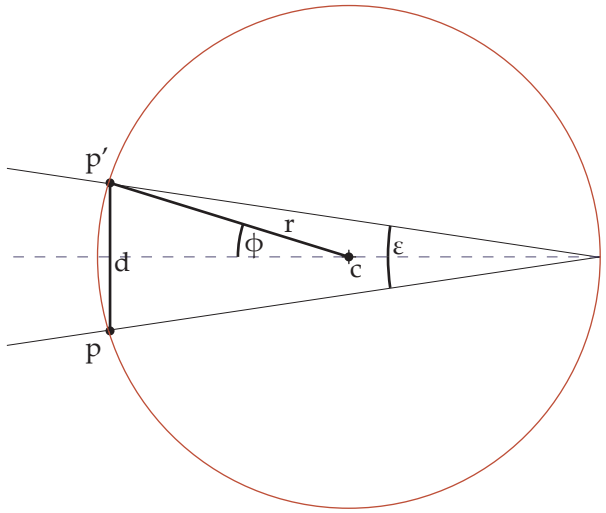
The radii of the translation and move-in error thresholds can be determined easily as depicted in Figure 7.1a. From the central angle theorem follows that $\phi = \epsilon$, and

$$r = \frac{d}{2 \sin \epsilon} \tag{7.5}$$

As second level of the metric, we simply use the error metric of Schaufler [Sch95]. When using the same construction scheme as described, parts of the calculations can be shared between the two metrics, which improves the overall efficiency.

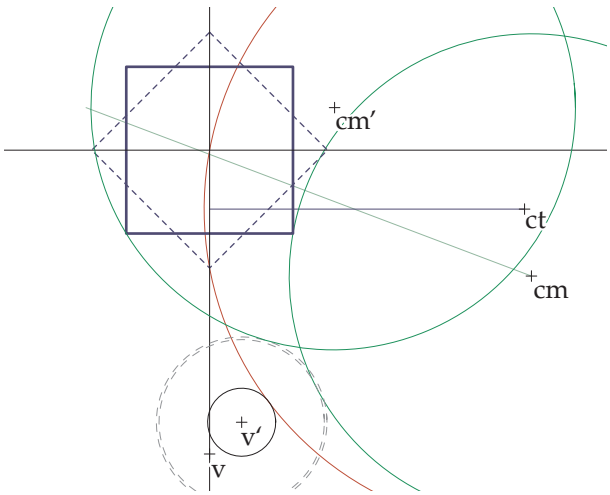
Metric Evaluation for Impostor Update

The scheme for runtime evaluation is straightforward: if the viewpoint is still in the GVV, the impostor is definitively valid. Otherwise, the second stage is performed to return a more accurate result. If the impostor is invalid, it is updated and the GVV is recalculated for the current viewpoint and the new impostor. If the result is a valid impostor, only the GVV is recalculated according to the new viewpoint, so that the first stage will



(a) Radius construction detail.

Figure 7.1: GVV radius construction



(a) GVV construction overview. v: Original (impostor) view point; v': current view point; cm: ; ct: translation error threshold; cm,cm': move-in error thresholds (compare Figures 5,6 in Schaufler [Sch95]).

```
Two_Level_Impostor_Update()
begin
    if object cannot be rendered from impostor then
        set to render geometry;
    else
        if viewpoint is inside the GVV then
            no update necessary;
        else
            if precise metric fails then
                Update_Impostor();
            end
            Update_GVV();
        end
    end
end
```

Algorithm 1: two-level impostor validity test.

hopefully be sufficient for the following frames. The steps are summarized in Algorithm 1.

The GVV can also be used efficiently for the hierarchical evaluation scheme used in the hierarchical image cache: the validity of each node depends on the GVV of its own impostor and on the GVV of its child nodes. A GVV of a node is then defined as a sphere that completely lies within the intersection of the GVVs of all child nodes and the GVV of the current node. Beginning with the leaf nodes, a GVV can easily be generated for each intermediate node from the respective child nodes.

During hierarchy traversal for the impostor update, the node GVV can be used to quickly decide whether all impostors of a given sub-tree are guaranteed to be valid or the child nodes have to be traversed. If the node GVV test fails, it is always possible to update it so that it contains the current viewpoint. Algorithm 2 summarizes the hierarchical version of the two-level impostor update.

7.4 Results

We have measured timings for both a single level evaluation (just using the precise error metric) and our two-level metric in a scene containing 65536 objects for 4500 frames of a pre-recorded walkthrough. On average, $i_{m_1} = 5.7\%$ of the impostors (3728) were classified as uncertain by the first stage and thus needed to be evaluated more precisely. From these

```
Hierarchical_Impostor_Update(node n)
begin
    if viewpoint is in node GVV then
        no update necessary;
    else
        for for all child nodes c do
            Hierachical_Impostor_Update(c)
        end
        if at least one child node updated then
            Update_Impostor(n);
        else
            if precise metric fails then
                Update_Impostor(n);
            end
        end
        Update_GVV(n);
    end
end
```

Algorithm 2: Evaluation for hierarchical impostors using the two-level error metric.

3728 impostors, the second stage classified only $i_{m_2} = 1.1\%$ (39 impostors) as actually requiring an update.

The two-level metric needs $t_{m_1} = 5.7ms$ per 10.000 impostors for the first stage, and $t_{m_2} = 8.6ms$ per 10.000 impostors that make it to the second stage. If only the second stage is used, it requires $t'_{m_2} = 15.9ms$ per 10.000 impostors. This is significantly higher because the second stage cannot reuse calculations from the first stage here (as was mentioned in Section 7.3). For this example, the average frame time including rendering was 124.1ms, and the same walkthrough with just the second stage enabled 160.0 ms, which is a factor of 1.29 (see Figure 7.2).

7.5 Summary

We presented a new two-level evaluation scheme for impostors that is both fast and precise. It can be efficiently used for individual impostors as well as for hierarchically organized impostors, namely the hierarchical image cache. Results showed a frame rate increase of 29% compared to using an exact error metric only, with exactly the same rendering results. A theoretical background on error metric performance was also

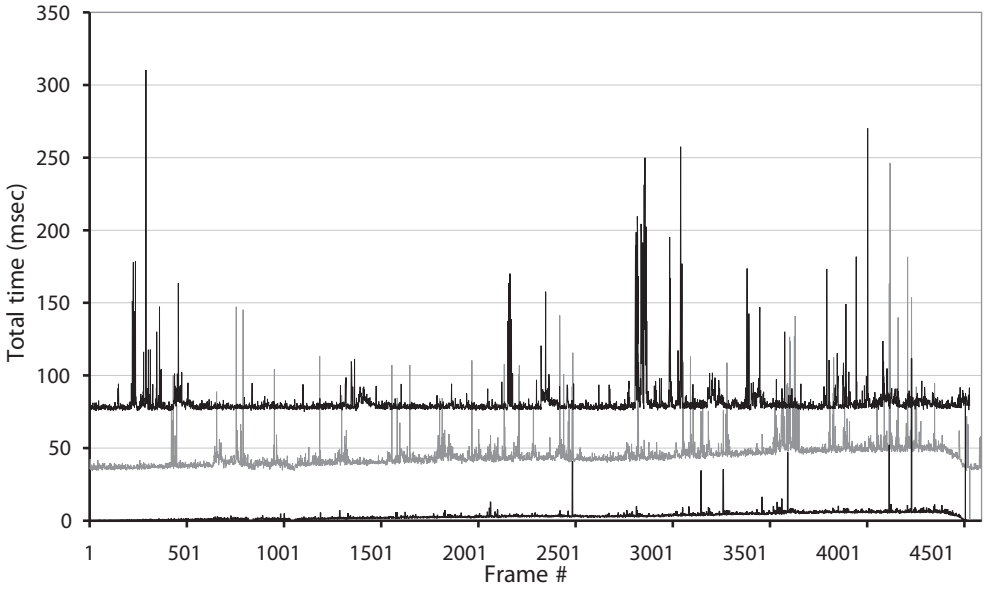


Figure 7.2: Times needed for evaluating 65536 impostors with different error metrics. From top to bottom: Schaufler's metric (second stage only), first stage of the two-level metric, second stage of the two-level metric.

presented, showing that the improvement gained by a two-level metric is independent of rendering systems or hardware specific characteristics. Only the relative performance of the error metrics and the quality of the result of the metric of the first level are relevant.

CHAPTER 8

TIME-CRITICAL RENDERING OF DISCRETE AND CONTINUOUS LEVELS OF DETAIL

> In My Egotistical Opinion, most people's C programs should be indented six feet downward and covered with dirt.
>
> Blair P. Houghton

8.1 Introduction

Time-critical rendering ensures guaranteed frame rates even for scenes with very high complexity. Therefore it provides a convenient framework for real-time rendering applications. Typically a time-critical framework mainly consists of a level of detail (LOD) selection method, that chooses the most valuable representation for visible objects not exceeding the available rendering budget.

We present a time-critical rendering approach that combines discrete and continuous LOD selection and we demonstrate its benefits in a ter-

rain flyover application. Our LOD selection method is based on an iterative optimization method over mixed (discrete and continuous) variables, which essentially combines previously used approaches for discrete and smooth LOD selection. In this context the notion of discrete level of detail is not restricted to static LOD representations, but extends to hierarchical or even view-dependent level of detail representations as well. Hence our approach addresses real-time rendering of environments with large 3D models (in terms of complexity and geometric extent) combined with many small, but complex objects in particular. The latter models are represented by view-independent levels of detail. Further we describe an extension to this basic LOD selection method to handle a huge number of objects with smooth representations.

The main motivation for this work is the real-time display of a terrain model with added dense vegetation. The terrain is rendered as a coarse view-dependent LOD, whereas each tree in the vegetation is displayed as smooth level of detail. We employ point based representations of procedurally generated trees for this task.

8.2 Previous Work

Most work on rendering acceleration techniques focuses on improving the frame rate while maintaining a controlled image quality. We aim on real-time rendering that *guarantees* a user given frame rate regardless of the complexity of the scene. The rendering process of every frame has to meet strict timing constraints and the goal of the rendering framework is to attain the best visual quality with the available rendering time. In the following sections, we will discuss previous work on time-critical rendering of various models and level of detail approaches.

Discrete Level of Detail Management

Most scene graph libraries use distance based or screen size based LOD switching to accelerate rendering of complex scenes. The rendering time for each frame depends on the scene content, the viewing parameters and the LOD switching threshold, but it can be arbitrarily large. In terms of optimization these methods maximize the frame rate subject to controlled image quality. Whenever a guaranteed interactive frame rate is needed, the role of frame rate and image quality must be exchanged: maximize the image quality that is achievable subject to controlled render time.

Funkhouser and Séquin formulated this optimization task as a multiple choice knapsack problem (MCKP) [FS93], which is known to be NP-hard, and used an approximation method to select the appropriate LOD for each object to be rendered. The importance of every object and the accuracy of every LOD depend on the viewing parameters, thus the MCKP must be solved for every frame.

Solving the MCKP needs a certain amount of time, which may affect the rendering performance on a uniprocessor computer. Funkhouser and Séquin used a dual processor solution for rendering: one processor selects the representations for the next frame and the other processor feeds the graphics pipeline.

This approach has one major disadvantage: visible objects may be completely missing if the allowed rendering time is not long enough to draw at least the coarsest representation of every visible object. Both Mason and Blake [MB97] and Maciel and Shirley [MS95] utilized a hierarchical level of detail approach and used variations of the MCKP to select appropriate representations for every frame. Mason and Blake's approximation method of their extended MCKP can be seen as a top down greedy traversal of the LOD hierarchy and guarantees half optimality, whereas Maciel and Shirley's heuristics don't provide such lower bounds. An extensive discussion of LOD management approaches can be found in Mason's Ph.D. thesis [Mas99].

Since in our implementation the selection of discrete levels of detail is based on Mason and Blake's work, we describe their idea briefly. Basically their LOD selection performs a sequence of so called increment steps as long as the rendering budget constraint is not violated. An increment step replaces a currently selected representation by the most valuable successor nodes in the LOD hierarchy. A converse decrement step is also introduced, which replaces least rated successor nodes with their parent. The root node (representing the entire scene at the coarsest level) is the initially selected representation. Additionally Mason and Blake exploited frame–to–frame coherence using the LOD assignment for the previous frame as starting point for a sequence of increment and decrement steps. Since our approach does currently not exploit frame–to–frame coherence and therefore starts from scratch at every frame, our method performs only increment steps.

Multiresolution Level of Detail Management

If the object representation allows smooth levels of detail, a similar optimization problem can be formulated, where the discrete variables are

replaced by continuous ones. Gobbetti and Bouvier [GB99, GB00] applied optimization techniques for solving non-linear constrained systems to time-critical rendering of multiresolution meshes. They partitioned the available frame time into the time available to the LOD selection procedure and the actual rendering time. Their LOD selection method proceeds incrementally by successively improving the current solution. Thus, this method can be interrupted at any time returning a suboptimal, but feasible solution.

Their LOD selection procedure is not applicable to scenes with a large number of multiresolution objects, since they assign one variable to every object.

Wimmer and Schmalstieg [WS98] describe a direct (non iterative) solution for smooth LOD selection using Lagrange multipliers, but they take only the rendering budget constraint into account and ignore potential constraints on maximal object resolutions. Therefore their approach works well only for objects with virtually unbounded resolution.

Schmalstieg and Fuhrmann [SF99] implemented a LOD selection policy for hierarchical and deformable multiresolution models and applied their method to character animation and terrain rendering.

Point based Rendering

The use of point based primitives for rendering was first introduced by Levoy and Whitted [LW85]. They observed that with the growing complexity of computer generated scenes, classical modeling primitives such as triangles become less appealing. Using points as a rendering primitive bears several advantages, such as being able to render arbitrarily complex geometry with a standardized rendering algorithm.

Pfister et al. [PZvBG00] and Rusinkiewicz et al. [RL00] almost simultaneously published new point based systems that used additional information, such as surface normals and texture data, for each point sample. However, the method introduced by Pfister was aimed at high fidelity, whereas Rusinkiewicz was more interested in handling huge amounts of data. The goal was to interactively visualize laser scans of up to 2 billion samples at interactive frame rates. Therefore, they designed a hierarchical data structure that allowed lower resolutions to be displayed even while additional data was still being read in.

Recently Cohen et al. [CAZ01] and Chen and Nguyen [CN01] combined polygonal and point based rendering by exploiting a hybrid representation of the entire scene. This is different to our approach, since we represent one object either point based or polygonal.

Terrain Visualization

Recent work on large scale terrain visualization with regular height-fields was done for instance by Hoppe [Hop98] and Lindstrom and Pascucci [LP01]. Both approaches employ a smooth level of detail framework and the required resolution is calculated from the allowed screen space error. This threshold can be adaptively refined during the animation to match the desired frame rate.

If the scene database consists of several LOD hierarchies (e.g. terrain augmented with vegetation), the adjustment of resolutions of different parts is no longer unique. Therefore we utilize a predictive model of the visual quality and rendering cost to calculate the appropriate resolution for each scene component.

Our terrain visualization system is based on the Styrian Flyover project (Kofler et al. [KGG98]), which uses a quad-tree representation of the digital elevation model and the ground texture. During rendering the determination of the required resolution is based on the distance to the virtual camera.

The quadtree representation can be regarded as a coarse variant of view dependent multiresolution meshes (e.g. view dependent progressive meshes, VDPM [Hop97]). Even with smoother view dependent progressive meshes our method described in Section 8.3 remains the same, since the current level of detail of a VDPM consists of a (discrete) set of nodes in the VDPM representation.

The chunked LOD approach [Ulr02] is conceptually similar to our method, but optionally uses a pre-simplified terrain mesh and hides visual artifacts near patch boundaries by rendering additional vertical triangles.

8.3 Mixed Level of Detail Selection

In every iteration, our algorithm performs either a greedy refinement step of discrete levels of detail or a gradient ascent step for smooth representations. The ratio of (estimated) visual benefit to rendering cost is used to select the step to perform. The iteration stops when the available rendering budget is exhausted, the highest level of detail of visible objects is attained or the time available to the optimization procedure itself is completely spent. In any case the obtained assignment of resolution for every object is feasible.

Since our application contains a huge number of smoothly represented

objects and therefore a very large set of continuous resolutions has to be determined, we utilize a hierarchical arrangement to accelerate the LOD selection procedure.

Problem Formulation

We consider the LOD selection problem for discrete and continuous variables. A set of representations $\{R_i\}$ of discrete objects and a set of objects $\{X_i\}$ with smooth levels of detail are given. For each discrete representation R_j we introduce a boolean variable r_j: we set $r_j = 1$, if the LOD selection chooses R_j for rendering in the next frame and $r_j = 0$ otherwise. Additionally we associate a continuous variable x_i with each X_i, that denotes the chosen resolution for rendering X_i.

For every representation a pair of functions is given: the first function (the *benefit*) estimates the effect on the visual quality if an object is rendered at a given resolution; the other one estimates the corresponding cost of drawing (i.e. rendering time) the object at the desired resolution. For discrete objects we denote these two functions by $benefit_d(R_i, r_i)$ and $cost_d(R_i, r_i)$ respectively. For objects with continuous resolutions these functions have the resolution x_i as the argument and are written as $benefit_c(X_i, x_i)$ and $cost_c(X_i, x_i)$. Obviously these functions also depend on the current viewing parameters, but for convenience we will not expose this dependency explicitly.

The task of the LOD selection method is to choose r_j and x_i:

$$\begin{aligned} \max \sum_j benefit_d(R_j, r_j) \quad &+ \quad \sum_i benefit_c(X_i, x_i) \text{ such that} \\ \sum_j cost_d(R_j, r_j) \quad &+ \quad \sum_i cost_c(X_i, x_i) \le T \\ r_j \quad &\in \quad \{0, 1\} \\ x_i \quad &\in \quad [0, 1] \end{aligned} \tag{8.1}$$

We restrict the resolution x_i to the range $[0, 1]$, where $x_i = 1$ represents the maximal resolution assigned to a multiresolution object. There are usually additional constraints related to r_j which encode the LOD hierarchy (details can be found in Mason and Blake [MB97]).

We aggregate the x_i into one vector $\vec{x}$ and denote the total cost caused by rendering smooth representations by

$$cost_c(\vec{x}) = \sum_i cost_c(X_i, x_i). \tag{8.2}$$

Similarly we define the total benefit of rendering the smooth levels of detail at resolution $\vec{x}$ as

$$benefit_c(\vec{x}) = \sum_i benefit_c(X_i, x_i). \tag{8.3}$$

Terminology

We require some additional notation to express the algorithm and its derivation in a compact manner. For any current assignment of smooth variables $\vec{x}$ and an arbitrary search direction $\vec{a}$ (with $\|\vec{a}\| = 1$) we define the directional cost function

$$\widehat{cost}_c(t) := cost_c(\vec{x} + t\,\vec{a}) \tag{8.4}$$

and (by the chain rule) we have

$$\frac{d\widehat{cost}_c}{dt} = (\vec{a}, \nabla_{\vec{x}} cost_c(\vec{x} + t\,\vec{a})) \tag{8.5}$$

(We denote the inner product of $\vec{a}$ and $\vec{b}$ by $(\vec{a}, \vec{b})$.) Similarly, we introduce the directional benefit function

$$\hat{b}_c(t) = benefit_c(\vec{x} + t\,\vec{a}). \tag{8.6}$$

Again, we will need the derivative in the following sections:

$$\frac{d\hat{b}_c}{dt} = (\vec{a}, \nabla_{\vec{x}} benefit_c(\vec{x} + t\,\vec{a})). \tag{8.7}$$

These functions depend on $\vec{x}$ and $\vec{a}$ as well; for clarity we omit these arguments since they are clear from the context.

Optimization Method

Our method is an interleaved combination of Mason and Blake's [MB97] increment steps for discrete LODs and gradient ascent steps for smooth representations. The optimization method works iteratively: Given a current assignment of discrete and continuous resolutions for each object, one of three cases may occur:

1. The rendering budget or the time available to LOD selection is exhausted and the procedure terminates with the current solution.

2. The discrete variables and therefore the total benefit and costs are increased.

3. The smooth resolutions are raised, thus increasing both the total benefit and cost.

In every iteration, the algorithm decides either to increase the discrete or continuous variables. If the discrete variables are already at the highest resolution or increasing every possible discrete representation causes violation of the constraint, the algorithm tries to enhance the smooth representations.

On the other hand, if the smooth levels are already at the highest resolution, only improving the discrete resolutions is considered by the algorithm.

If both improvement steps are feasible, then the algorithm compares the added benefit to added cost ratio of the best discrete improvement with the slope of the benefit function along the search direction. More formally we denote the added benefit obtained by the increment step as $\Delta benefit_d$ and the corresponding cost as $\Delta cost_d$. The quantities to compare are

$$slope_d = \frac{\Delta benefit_d}{\Delta cost_d} \tag{8.8}$$

and

$$slope_c = \frac{d\tilde{b}_c}{dcost}(cost_c(\vec{x}_i)), \tag{8.9}$$

where

$$\tilde{b}_c(cost) = \hat{b}_c(\widehat{cost}_c^{-1}(cost)). \tag{8.10}$$

Applying the chain rule we obtain

$$\frac{d\tilde{b}_c}{dcost}(cost_c(\vec{x}_i)) = \frac{d\hat{b}_c}{dt}(0)\,\frac{1}{(\vec{a}, \nabla_{\vec{x}} cost_c(\vec{x}_i))}. \tag{8.11}$$

(Recall that $\hat{b}_c$ implicitly depends on the current assignment $\vec{x}$ and the current search direction $\vec{a}$.)

If $slope_d > slope_c$, then incrementing the discrete LODs is certainly favorable. Otherwise the smooth variables are increased and the new solution $\vec{x}_{new}$ is set according to

$$\vec{x}_{new} = \vec{x} + t_{opt}\,\vec{a}. \tag{8.12}$$

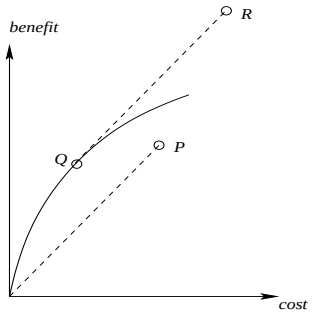


Figure 8.1: Determination of t_{opt}. The dashed line corresponds to a discrete increment step, whereas the solid curve illustrates $\tilde{b}_c$. P represents the location in the benefit/cost space after the discrete increment step, Q corresponds to the optimal increase of the continuous resolutions and R depicts the added result.

Selection of the Search Direction

Essentially the search direction $\vec{a}$ is the gradient of $b_c(\vec{x})$. If x_i is already saturated ($x_i = 1$), then $a_i = 0$. To avoid too small steps because of repeated saturation of smooth variables, approaching the boundary $x_i \leq 1$ can be penalized by a barrier term, e.g.:

$$a_i = \max\left\{0, \frac{\partial b}{\partial x_i}(\vec{x}) - \frac{\varepsilon}{1 - x_i}\right\}, \tag{8.13}$$

where ε is a small constant.

Line Search

Given the search direction $\vec{a}$, $\vec{x} + t\,\vec{a}$ leaves the feasible region for some $t_{max} > 0$. If $d\tilde{b}_c/dcost > slope_d$ at t_{max}, then $t_{opt} = t_{max}$, since raising the continuous resolutions along the search direction $\vec{a}$ gives always better values.

Otherwise we determine $t_{opt} \in (0, t_{max})$ such that

$$\frac{d\tilde{b}_c}{dcost}(\widehat{cost}_c(t_{opt})) = slope_d. \tag{8.14}$$

Thus, we increase the values of the smooth variables as long as this is preferable over the discrete incremental step. The graphical explanation is given in Figure 8.1.

Continuous Variable Hierarchy

The optimization procedure described above turns out to be not sufficiently scalable because the number of continuous variables is unnecessarily large for certain viewpoints. Although the time used for LOD selection can be amortized over several frames, there is an even better option available. We observe that (nearly) identical resolutions are often assigned to spatially close multiresolution objects. Therefore we augment view frustum culling for smooth objects with a hierarchical generation of continuous variables corresponding to multiresolution objects. We assume that smooth objects are organized in some spatial hierarchy (e.g. to speed to up view frustum culling).

Consider Equation 8.1 augmented with the following additional constraints:

$$x_1 = x_2 = \ldots = x_n \tag{8.15}$$

These constraints ensure that the continuous variables always have the same value. The optimal value of this tightened problem yields a non-optimal assignment of smooth LODs, but this loss in optimality can be predicted. If the estimated loss is smaller than some threshold θ, we replace the n smooth variables by only one, namely x, and set $x_i = x/n$. Additionally we replace the upper bounds on x_i with $x \leq n$. Note that the benefit and cost is still expressed in terms of X_i and x_i, e.g.

$$\sum_i benefit_c(X_i, x_i) = n\, benefit_c(X_i, x/n). \tag{8.16}$$

Otherwise we relax constraint 8.15 and replace it by one of the following four constraints:

$$\begin{aligned} x_1 &= \ldots = x_{n/4} \\ x_{n/4+1} &= \ldots = x_{n/2} \\ x_{n/2+1} &= \ldots = x_{3n/4} \\ x_{3n/4+1} &= \ldots = x_n \end{aligned} \tag{8.17}$$

These groups of variables correspond to child nodes in a quad-tree structure, in which the smooth objects are organized. For each of these constraints we estimate the loss in the corresponding optimization problem and compare it with an appropriate threshold $\theta/4$. If the loss is too large, the corresponding group of variables is recursively subdivided (i.e. the quad-tree node is replaced by its children) until the loss is small enough or the original variables are reached.

8.4 Test Application

Our test application consists of a terrain flyover scenario showing a large part of the Styrian province. The data set and terrain visualization system is taken from the Styrian flyover project (Kofler et al. [KGG98]). The complete height field consists of 2049×2049 samples covering a region of about $120 \times 120 km^2$. The corresponding texture has a resolution of 4096×4096. Both the height field and the associated texture fit entirely into main memory.

About 1.4 million trees are placed on a jittered grid into the terrain according to the forest regions found by image classification of the satellite image. The heights of these trees are exaggerated to have a closer match between the resolution of the ground texture and the vegetation. The type of tree was selected randomly, but spruces are more likely in higher terrains. In valleys and other lower parts the forest is a mixture of various conifers and broad-leafed trees.

The digital elevation model and the ground texture are organized in a quad-tree data structure as it is commonly used in terrain visualization systems (for example, the Styrian Flyover project [KGG98] or TerraVision II [RLIB99]). This quad-tree forms the discrete LOD hierarchy. Every triangulated patch in the hierarchy consists of 33×33 vertices and 64×64 texels. A repeating noise texture is combined with the ground texture to simulate small details on the ground.

The trees are grouped into sets of at most 64 spatially close, individual trees and the appropriate resolution is determined for these groups. A quad-tree hierarchy is employed again to speed up view frustum culling and to allow faster LOD selection.

Implementation Details

Tree Generation

In the current system, the tree generation itself is based on the algorithm by Weber and Penn [WP95]. Tree models are generated offline and stored on disk in an intermediate format, storing geometric information for each stem or branch as well as leaf locations. Upon startup of the test application, the desired trees are read in and the branch information is added as a list. The leaf positions are converted into a K-d-tree and reorganized as linear vertex and color array (using a top-down breadth first traversal). Each model is only read in once, and instances can be made by setting up the OpenGL transformations as required.

Point based Tree Rendering

The point based renderer is called separately for each tree, giving the rendering system fine control over the fidelity of each tree instance. Each tree is rendered with the assigned number of splats by drawing an appropriate fraction of all point samples generated for the template. Optionally the size of the rendered point primitives is adjusted to avoid holes in trees represented by a small number of splats.

Benefit and Cost Heuristics

Predicting the cost of rendering a representation is based on a model of the graphics pipeline. The cost of rendering a splatted representation consists of two parts: the setup time (e.g. to transform and scale the templates) and the variable time directly proportional to the number of rendered splats. Since the setup time becomes significant when at least one splat is assigned to an instantiated tree, the cost function is a piecewise linear graph:

$$cost_c(X_i, x_i) = \begin{cases} (c_{splat} + c_{tree})\, N_{splats}\, x_i & \text{if } n_i < N_{trees} \\ c_{splat}\, N_{splats}\, x_i + (c_{splat} + c_{tree})\, N_{trees} & \text{otherwise.} \end{cases} \tag{8.18}$$

Here c_{splat} denotes the rendering time for one splat and c_{tree} denotes the cost for individual tree setup, e.g. for transformation of the tree prototype to the final tree position. Since all objects are instances of the same model, the cost function $cost_c$ is actually independent of X_i. Both values are determined by a benchmarking procedure. On our hardware c_{splat} is $0.0775\mu s$ and c_{tree} is $1.385\mu s$. Our optimization procedure is able to handle this piecewise linear cost function correctly.

Since our polygonal objects (patches of a regular digital elevation model) consist of long triangle strips, we estimate the rendering cost as

$$cost_d(R_i, n) = c_{vertex}\, n, \tag{8.19}$$

where n is the number of sent vertices in the triangle strip and c_{vertex} is again an empirically measured combined cost for vertex and triangle processing (determined as $0.06\mu s$ on our hardware). Again, since our test application does not use models with different characteristics (with or without texturing, or with shaders of varying complexity) our cost function $cost_d$ is actually independent of the model R_i.

The benefit function for discrete and smooth objects is a product of an importance factor times an accuracy estimator. The importance is proportional to the projected screen area of the object. The benefit associated

with a group of N_{trees} trees is chosen as

$$benefit_c(X_i, x_i) = k\,S\,\frac{N_{trees}}{64}\sqrt{\frac{x_i}{N_{trees}}}, \tag{8.20}$$

where S is again the estimated screen size and x_i is the assigned resolution (in the range $[0, N_{trees}]$). Recall that $\frac{N_{trees}}{64}$ denotes the relative density of trees in one group containing at most 64 trees. k is a weighting constant currently set to 50.

The benefit assignment for terrain patches is essentially the same: For the quad-tree nodes at level d the benefit $benefit_d$ is set to

$$S\sqrt{4^d/4^{d_{max}}} = S\,2^{d-d_{max}}, \tag{8.21}$$

where S is the estimated screen size in pixels and d_{max} is the height of the terrain quad–tree. Note that $4^d/4^{d_{max}}$ is the ratio of texels displayed at level d to the number of texels is the full resolution terrain texture.

Estimating Benefit Loss

Let $X_i, i \in \{1,\ldots,n\}$ be a set of objects with smooth LODs. If x_i is the resolution assigned to X_i, and using the benefit functions defined in Equation 8.20, the total benefit is $\sum c_i\sqrt{x_i}$ for suitable c_i (essentially proportional to the estimated screen size). Our goal is the estimation of the maximal benefit loss caused by replacing the assigned resolutions x_i by the mean $\sum x_i/n$. Further we assume that $\sum x_i$ is bounded by the available budget X. Overall we search for the solution of

$$\begin{aligned} \max\sum c_i\sqrt{x_i} \;&-\; \sum c_i\sqrt{\frac{X}{n}} \text{ such that} \\ \sum x_i \;&=\; X \qquad \text{the budget is limited} \\ \sum c_i \;&=\; C \qquad \text{the screen extent is bounded} \\ x_i, c_i \;&\geq\; 0 \end{aligned} \tag{8.22}$$

The maximum is attained if $c_1 = C$ and $c_i = 0$ for $i > 1$. This implies that $x_1 = X$ and $x_i = 0$ for $i > 1$. In this case the error is

$$C\sqrt{X}\left(1 - \frac{1}{\sqrt{n}}\right). \tag{8.23}$$

This observation can be used to guide the process of splitting continuous variables. The recursive procedure to associate continuous variables

Function SplitVar
Input: $Node$, X, depth d, tree height d_{max}, a threshold ε
$S \leftarrow$ screen coverage of $Node$
if $k S \sqrt{X}\left(1 - \frac{1}{\sqrt{2^{d_{max}-d}}}\right) < \frac{\varepsilon}{2^{d_{max}-d}}$ **then**
{Estimated benefit loss is small (compare Equation 8.23).}
Associate a continuous variable with $Node$
else
{Solve the continuous LOD selection problem for the children $Child_i$ of $Node$, and obtain $X_1, \ldots, X_4$ as estimators for total resolution assigned to $Child_i$. The solution can be directly calculated using Lagrange multipliers.}
$S_i \leftarrow$ screen coverage of $Child_i$
$X_i \leftarrow X\, S_i^2 / \sum S_j^2$
for $i = 1$ to 4 **do**
Call SplitVar($Child_i$, X_i, $d+1$, ε) recursively
end for
end if

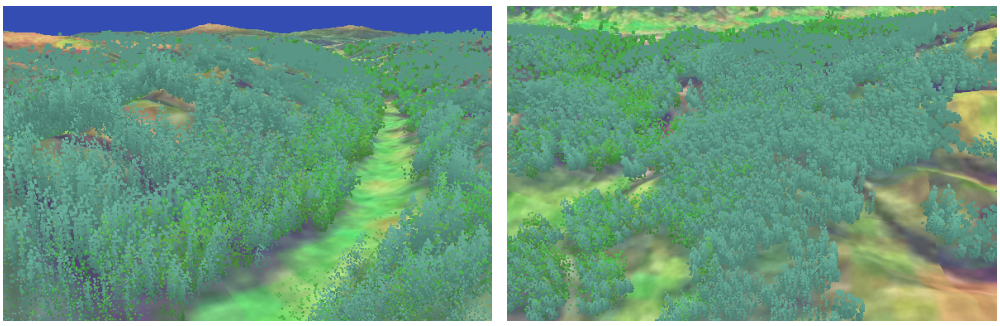
Algorithm 3: Determination of the required granularity of smooth variables

with groups of smooth LOD representations is given in Algorithm 3. Instead of using the same total resolution X (which can be computed from the available rendering budget), we estimate the assigned maximal resolution for every child node. Since this estimator is only an approximation, there is no guaranteed upper bound in the total loss.[1]

Occlusion Culling and Small Detail Culling

With the general availability of hardware supported occlusion tests it is reasonable to exploit this feature for large model visualization. In our framework a fixed portion (5%) of the frame budget is optionally used to render the terrain and subsequently the nodes in the vegetation hierarchy are tested for visibility. When the user navigates at very low altitudes, much of the vegetation in the far field is culled and higher resolutions can be assigned to trees in the near field.

[1] Strict error bounds can be derived for logarithmic benefit functions. We still use the square root, since we observed better visual quality with this benefit function in general.



(a) Viewing the far field. (b) A birds eye view.

Figure 8.2: Views of the terrain with a rendering budget of 40ms. The minimum size of point splats is 2 pixels.

In order to avoid expensive transformations to position the tree templates we perform small detail culling for groups of trees, which have a very small estimated screen size. This is performed in combination with view frustum culling. The remaining nodes (collections of trees according to the quadtree hierarchy) are passed to the LOD selection procedure.

If the point budget assigned to a group of trees is not sufficient to display each tree with at least one splat, only the corresponding fraction of the group is rendered. Thus, we obtain a smoother transition between culled vegetation and still displayed groups.

Results

We performed several measurements on a standard PC with 1.8GHz processor and an ATI Radeon 9700 graphics card. Figure 8.3 displays the actual rendering time and the time spent in LOD selection (with and without the continuous variable hierarchy acceleration technique). The allowed rendering budget was set to 40ms. With these settings our LOD selection procedure required about 20ms at maximum. If the rendering budget is increased to 80ms, less than 30ms are spent in LOD assignment. Figure 8.2 shows screen shots from our demonstration application with a constant rendering budget of 40ms.

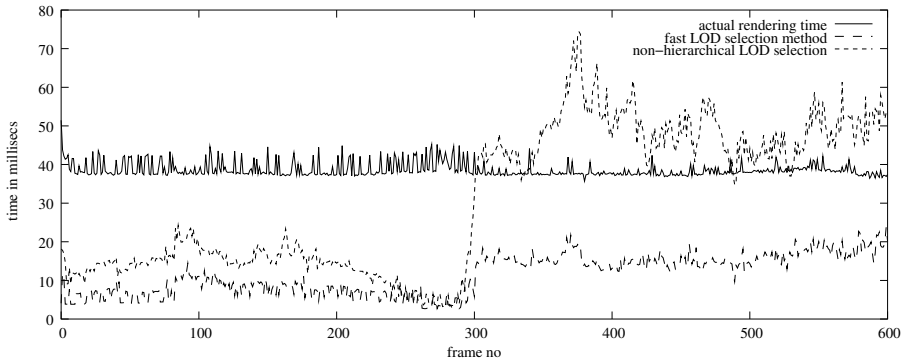


Figure 8.3: LOD selection and actual rendering times for a flyover path. The rendering budget was 40ms per frame. The solid line depicts actual frame times (combined hierarchical LOD selection and rendering). Note, that the frame times are close to the target frame time of 40ms. The dashed lines represent the time spent solely for LOD selection indicating the performance difference between hierarchical and non-hierarchical LOD selection.

8.5 Discussion

Polygonal Multiresolution Objects

Instead of a point-based representation of trees a polygonal multiresolution approach to display vegetation can be used [RCB+, RCRB03]. The multiresolution models are stored compactly as a sequence of mesh refinements (as in Bouvier and Gobbetti's Totally Ordered Meshes [BG01]) and the displayed mesh at the desired resolution is extracted on demand for rendering. Since mesh extraction can be performed in an incremental manner, the total time to generate actual models is proportional to the maximal requested resolution (assuming that one mesh refinement operation takes constant time).

Typically the metric to select the necessary resolution for polygonal multiresolution meshes is based on geometric deviation from the original model. In order to apply our level of detail selection method a continuous (and differentiable) approximation can be employed or an entirely heuristic metric for multiresolution meshes must be applied (similar to Gobbetti's Time-critical Multiresolution Scene Rendering [GB99]).

Terrain Representation

Our terrain rendering approach is a relatively coarse, view dependent method. Smoother representations are view-dependent simplification approaches [Pup98, LE97, Hop98] for irregular terrain meshes and restricted triangulations [DWS+97, Paj98, LP01] for regular heightfields. To avoid too many decision within our LOD selection procedure, a tradeoff between smooth appearance and computational complexity is necessary. The fine grained terrain representation can be reduced by collapsing several refinement operations into one larger mesh update.

Objective Function

Our level of detail selection procedure maximizes the sum of benefits for visible objects (with discrete or continuous levels of detail), such that the available budget is not exceeded [FS93]. If the benefit functions are not carefully chosen, the solution for this objective function is susceptible to the following artifacts:

- The budget is not distributed uniformly to all regions of the displayed image. The LOD assignment procedure sometimes compensates low resolution in one part of the image with high resolution in some other region.

- Changing the view parameters slightly may cause completely different resolutions, which disturbs the impression of a smooth animation.

Using smoother level of detail representations decreases those artifacts. The first item can be specifically addressed by choosing a different objective function: maximize the minimum accuracy (or alternatively minimize the maximum deviation) of all visible objects. Designing an efficient LOD selection procedure for this objective function and the evaluation of the resulting visual quality is future research.

Achieving smooth animations with visual coherence between successive frames is important for high visual quality. Using intra-frame LOD selection methods, temporal artifacts occur usually if an object represented by few discrete levels enters or leaves the viewing frustum. Restricting the change in resolution between successive frames implies a relaxation of the budget constraint, since both objectives cannot be satisfied simultaneously in general. Inter-frame LOD selection methods could be an option, but appear currently too expensive to be incorporated.

8.6 Summary

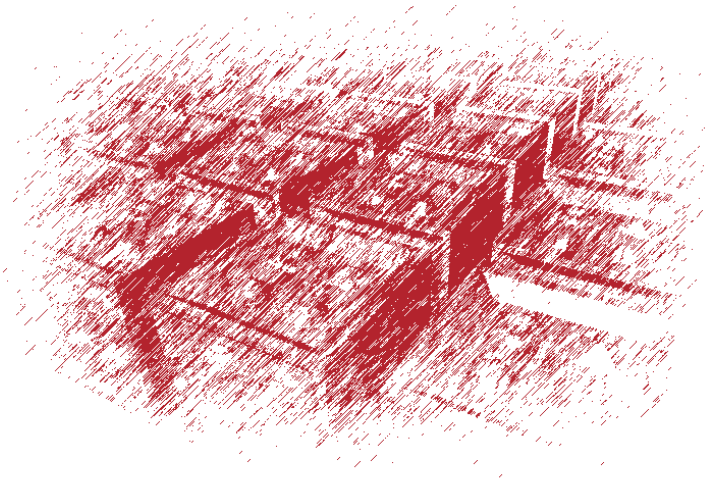
We presented a time-critical LOD management method that incorporates discrete and continuous levels of detail. Our approach combines approximation methods for discrete LOD selection with gradient ascent methods to choose an appropriate multiresolution representation. A terrain flyover application comprises the current test bed for our LOD management procedure.

In this work our main focus is the LOD selection method and we are aware that both terrain rendering and drawing of vegetation can be substantially enhanced. Especially our prototype of the point based vegetation renderer can be highly accelerated. Nevertheless our work demonstrates the prospects of mixed resolution objects within a time-critical framework.

We intent to utilize this approach in an urban visualization application, which aims on the real-time photorealistic presentation of an existing city. Further we will refine the terrain rendering engine and work on improvements on the drawing of vegetation. Additionally occlusion culling in terrain environments would significantly increase the accuracy of our LOD selection.

PART IV

TECHNICAL ASPECTS



CHAPTER 9

AN EARLY-Z OPTIMIZATION FOR DISPLACEMENT MAPPING SHADERS

In theory, there is no difference
between theory and practice.
In practice, there is.

Jan L.A. van de Snepscheut

9.1 Introduction

Early z-testing is an important optimization employed by current graphics hardware that tries to avoid the cost of executing expensive pixel shaders for invisible pixels. The early z-test is also widely employed to mask computations in general purpose GPU computations [HLB+05], and to stop iterative shader execution in GPU ray-casting [KW03].

Recently, approaches that modify the depth value on a per-pixel basis are becoming more and more common to ensure correct intersections if the pixel shader modifies the displayed geometry. Particularly well-known examples are per-pixel displacement mapping techniques such as relief mapping [POC05], parallax occlusion mapping [Tat06], or view-dependent displacement mapping [WWT+03b].

A general implementation problem of early z-testing is not only that the depth value of a pixel is not known before shader execution when it is modified by the shader, but also when and how the corresponding hierarchical z buffer [GK93] is updated. Therefore early z-testing is currently disabled when shaders modify depth values.

We propose a slight modification to current graphics hardware, or the respective drivers, that would allow depth-modifying pixel shaders to benefit from early z-testing and thus considerably increase performance. Although we use depth correct displacement mapping pixel shaders as an example, our proposed modification also applies to any other shaders that calculate depth values in a similar manner.

9.2 Z-Correct Per Pixel Displacement Mapping

Per-pixel displacement mapping is an enhancement of the bump mapping and normal mapping techniques for adding surface detail without creating additional geometry. It captures perspective effects such as self occlusion and motion parallax, which is desirable for modeling surface mesostructure with more apparent depth.

The main component of per-pixel displacement mapping techniques is a ray casting algorithm implemented in the pixel shader which adjusts texture coordinates to displace the intersection of the viewing ray with a hypothetical surface, often represented as a height map [POC05, Tat06, OBM00]. The surface represented by the height map can be either above the polygon face ("push up"), or below ("push down") as illustrated in Figure 9.1. Note that in the "push up" case, the hypothetical surface is always closer to the viewer than the actual geometry (ie. the rendered polygon), and conversely for the "push down" method: here the hypothetical surface is always further away than the geometry.

If the fragment shader does not modify the depth value according to the displacement, the interpolated depth value is used instead, which leads to artifacts in the case of intersections.

Although *conceptually* the pixel shader modifies an interpolated depth value, this value is only available to the shader in OpenGL (and not in DirectX), and the depth value returned by the pixel shader is typically calculated independently. We therefore use the term 'depth modifying' more generally to refer to pixel shaders that calculate depth in a way that is predictable from the interpolated depth value.

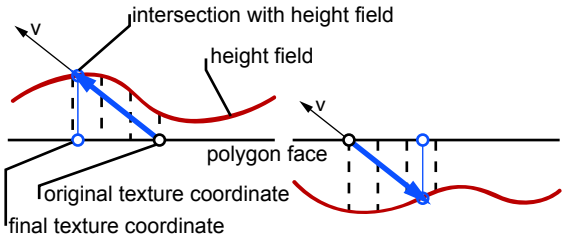
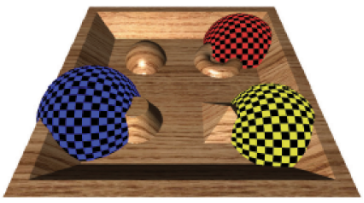


Figure 9.1: Basic Displacement Mapping. Left: "push up"; Right: "push down" approach.



(from Policarpo [POC05])

Figure 9.2: Writing correct depth values allows correct intersection of a displacement mapped polygon with other objects.

9.3 Early-Z Optimizations

Modern graphics hardware includes an initial test of the depth value of the current fragment against the contents of the Z buffer, possibly discarding occluded fragments before performing lighting and texturing calculations. For maximum effectiveness of this early culling acceleration it is generally recommended to draw a scene in front-to-back order [Hud05], or to perform an additional pass to mask out areas where shader evaluation is not needed [STM05].

Two variants of Early-Z optimizations are possible. It is generally preferable to perform the entire read-modify-write Z buffer update in parallel to the shader execution. However this is only possible if the fragment will not be discarded at a later stage to avoid invalidation of the Z buffer. In practice, this problem occurs with alpha testing and shaders that possibly discard fragments using texkill instructions [NVI05]. In this case a culling only test can still be performed, with the actual Z buffer update at a later stage.

The performance gain of early culling increases with the complexity of the shader, making it highly desirable for rendering complex shaders and large scenery. However, if the shader modifies the fragment depth

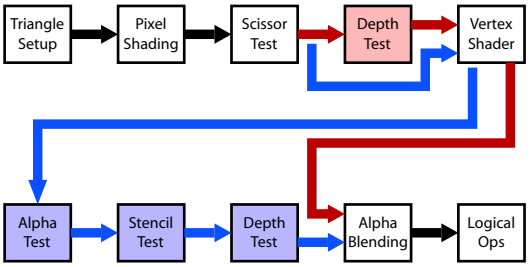


Figure 9.3: Schematic of the OpenGL shader pipeline with Early Depth Testing (after Anderson [AIK+05]). Red path is taken with early Z acceleration; the blue path is taken if early Z is disabled.

value, early-Z optimization is no longer possible as the final depth is not known before fragment shader execution. It must be evaluated first - executing the full shader and performing all texture lookups - and only then a decision can be made whether the fragment can be safely discarded. Since this is independent of the actual contents of the z buffer, it is roughly equivalent to the worst-case scenario of rendering back to front.

In the case of depth correct displacement mapping, the final depth is not yet known before shader execution. Therefore it is impossible to perform Z buffer *updates* in advance; our discussion thus focuses on enabling early Z-buffer *culling* for displacement mapping.

Other Hardware Considerations

For maximum performance, current GPU architectures include a wide range of optimization and acceleration techniques such as hierarchical Z-buffer schemes and parallel processing of multiple fragments. One important example is parallel processing, as modern GPUs typically process groups of four pixels simultaneously. Pixel processing pipelines are evaluated in lockstep: if one fragment is invalidated early (e.g. through texkill), the pipeline for this fragment cannot be refilled immediately and stalls until all pixels of the group have been processed [ATI05, Was05]. Therefore, early clipping only provides a significant speedup if the entire group of pixels can be discarded.

Comparison Function	**Fragment accepted if ...**
NEVER	(never)
LESS	$z_{fragment} < z_{buffer}$
LEQUAL	$z_{fragment} \leq z_{buffer}$
EQUAL	$z_{fragment} = z_{buffer}$
NOTEQUAL	$z_{fragment} \neq z_{buffer}$
GEQUAL	$z_{fragment} \geq z_{buffer}$
GREATER	$z_{fragment} > z_{buffer}$
ALWAYS	(always)

Table 9.1: DirectX and OpenGL Depth Comparison Functions (function names taken from OpenGL; DirectX uses nearly identical descriptions.)

9.4 Preserving the Validity of Early-Z Culling

To further complicate the problem, modern GPU hardware supports several depth comparison functions that can be used to accept or reject fragments [SA03, Mic05]; the available modes have been summarized in Table 9.1.

By observing how the "push up" and "push down" approaches to displacement mapping affect the depth value of a fragment, it becomes obvious that in some cases the depth comparison results for the initial and modified depth values will be the same. For example, the default depth comparison is LESS ($z_{fragment} < z_{buffer}$). Since a "push down" displacement mapping shader only increases the depth value, a fragment rejected with the initial depth value can never be a false negative. It can only turn out that the adjusted depth value is pushed 'behind' the z buffer value, i.e. $z_{zinitial} < z_{buffer} < z_{final}$. This can be captured if the depth comparison is repeated for the final depth value.

Similar considerations can be made for the other comparison functions and "push up" displacement mapping.

The EQUAL comparison function requires a slightly different change, as a failed test for equality does not sufficiently constrain the value range of the depth of a fragment to ensure that the adjusted value cannot possibly be accepted. To achieve this, it is necessary to perform a signed test for equality, i.e. to determine if the adjusted value can *potentially* be accepted; the GEQUAL and LEQUAL functions can be used to perform this test for push up and push down displacement mapping respectively.

	Early-Z reject possible for ...	
Comparison Function	**Push Up**	**Push Down**
LESS	⊙	•
LEQUAL	⊙	•
EQUAL	∘	∘
NOTEQUAL		
GEQUAL	•	⊙
GREATER	•	⊙

- • directly applicable
- ∘ requires modified depth test
- ⊙ must test against max displaced value

Table 9.2: Applicability of early-Z depth tests to displacement mapping functions

Testing for Maximum Displacement

So far, the presented tests are only based on the depth value of the base polygon. However, in most displacement mapping techniques the maximum displacement is also known. This provides an additional opportunity for early-Z culling for the cases that cannot be resolved from the base depth alone, eg. "push up" displacement mapping with LESS (ie. $z_{final} < z_{buffer} < z_{initial}$).

To achieve this, an additional API call would be required to specify the maximum displacement value. This could possibly be performed through the z biasing mechanism, depending on the exact implementation within the graphics hardware. However, in this case 'normal' z biasing would no longer be available, so a dedicated solution is preferable.

Table 9.2 summarizes how the depth comparison functions can support early-z culling of displacement mapping shaders. In Figure 9.3, our proposed change requires a path that passes through both depth tests with an optional arithmetic operation for testing against the maximum displacement. The first depth test would only either discard the current fragment or pass it on, without any updates to the depth buffer. The optional arithmetic operation should preferably be performed with the depth buffer value instead of the interpolated depth value, since this would preserve the interpolated value for use in the pixel shader. The remainder of the pipeline could be left unchanged.

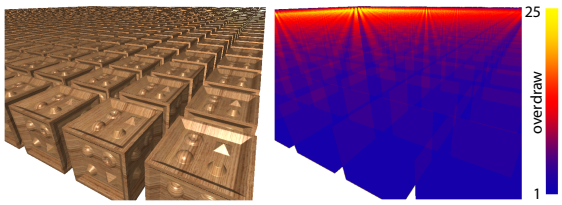


Figure 9.4: "Boxes" test scene (left) and resulting overdraw (right). In this image, 344318 pixels were drawn with 1351011 fragments (392% overdraw). Performance was 11.1fps rendering Front to Back, and 6.9fps Back to Front.

9.5 A Proof of Concept Simulation

Because the proposed early-z culling is not yet available in current GPU systems, we decided to implement a proof of concept simulation. To simulate the effect of early z-testing, it is necessary to discard fragments before the evaluation of the pixel shader, which is currently not possible with shaders that modify depth.

A simulation can still be performed by using a shader that does not modify depth. Because the disabled early z optimization is similar to the worst case (rendering back to front), one can render the same scene front-to-back and back-to-front and compare render times; the latter case effectively circumvents early z optimization and causes the fragment program to be evaluated for all objects, which is just what happens when a shader disables early z-testing. This could also be achieved by enabling stencil or alpha testing, but we decided to use the rendering order due to its ease of implementation in our framework.

This simulation is not fully accurate, because drawing back to front results in additional write operations that should not be performed, since occluded fragments are culled after shader evaluation. Additionally, the shader does not modify depth, such that Z-buffer writes can be optimized by the GPU. However, given the large cost of the displacement mapping fragment shader (which executes a few hundred instructions per fragment) the contribution of these operations to the total frame time is small. Performance comparisons showed that the difference introduced by the additional write operations was less than 1%.

Table 9.3 summarizes the frame times for various test scenes (see also Figure 9.4). The rendered scenes use a pixel shader based displacement mapping technique loosely based on Tatarchuk's Parallax Occlu-

Scene	Back to Front (no early-z)	Front to Back (with early-z)	Rel. Perf.
Boxes	6.9	11.1	160%
Spheres	5.7	7.6	133%
Hippos	3.7	4.7	127%
Buddha	8.8	9.8	111%
Dragon	1.5	1.8	120%

Table 9.3: Performance comparison for various test scenes (NVIDIA GeForce 7800GTX). Back to Front rendering simulates rendering without early-Z acceleration; Front to Back rendering simulates rendering with early-Z.

sion Mapping [Tat06] to display a number of complex objects. The performance gain depends on the shader evaluation cost and the overall depth complexity; both vary with the model and the viewpoint used for rendering, explaining the variation in Table 9.3. However, on average a 30% speedup was observed.

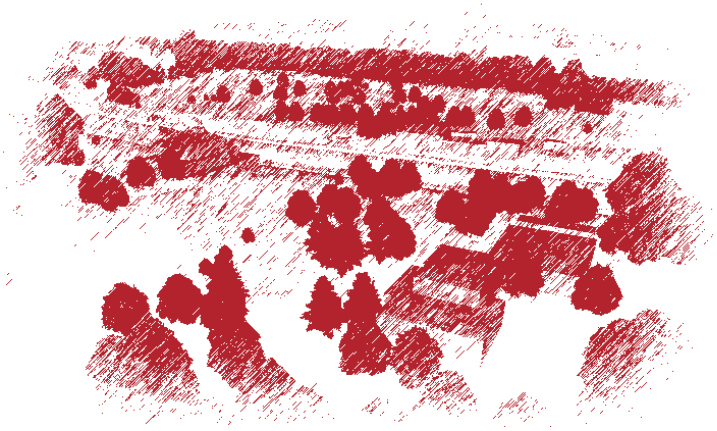
9.6 Summary

We have presented a possible adaptation to the current GPU rendering pipeline which would significantly increase performance for shaders which modify depth values in a predictable way. Given the increasing desire for realism, such shaders can be expected to become standard in modern applications and video games. The required change adds very little complexity to the rendering pipeline and could possibly be performed at the driver or firmware level. We have also demonstrated through a simulation that significant performance gains could be expected for this method.

Our understanding of the exact inner workings of current graphics hardware is based on the manufacturers' technical presentations. Unfortunately these often present a conceptual view and differ from the actual hardware implementation. We are currently working on additional tests to shed some more light on these aspects, such as the exact impact of pixel grouping on early culling performance.

PART V

SUMMARY AND CONCLUSIONS



CHAPTER 10

SUMMARY

> Research is to see what
> everybody else has seen,
> and to think what nobody else
> has thought.
>
> Albert Szent-Györgi

Due to the high complexity of vegetation and the necessity to span from close up views to very distant terrain, rendering landscapes is still a challenge. Despite the large increases in rendering performance with each new generation of graphics hardware, such scenes will remain too complex to be rendered in a straightforward manner for some time to come.

Even a single plant requires a very large amount of geometry to be displayed faithfully, and the structure of plants does not lend itself very well to the usual simplification and abstraction schemes. Therefore, "normal" geometric simplification algorithms such as Hoppe's Progressive Meshes [Hop96] or Garland and Heckbert's Quadric Error Metrics [GH97] often produce visually unsatisfactory results.

A number of specialized algorithms have been proposed that address the specific nature of plant models and that are often restricted to just rendering leaves or only the branches. These algorithms can be roughly categorized by the general method of approximation used: point based (reducing small detail to single points, often hierarchically), image based

(rendering planar impostors with precalculated views of the approximated geometry) and polygon based (using different geometry according to view distance).

An analysis of these algorithms is presented in Chapter 2; in particular, Table 2.1 gives a rough comparison of various features and applicable view distances. Most of the distance ranges are in the center range, such that they do not provide sufficient detail to be viewed at closer distances, and also run into problems when vast numbers of trees need to be displayed in the very far field.

10.1 Key Contributions

This thesis presents a number of approaches for rendering vegetation efficiently that are aimed at "filling the gap" between previously existing algorithms and facilitate the development of an environment that supports efficient rendering vegetation across a wide range of distances.

Recent developments in GPU architecture have resulted in much more per-fragment processing power, but traditional approaches to rendering vegetation have mostly relied on geometric simplification alone. As a result, these systems are mostly vertex bound - the resulting performance being mostly determined by the number of vertices to be processed - and the fragment processors was mostly unused.

Much of the presented work is designed to exploit these capabilities by offloading geometric complexity to the pixel shader. Ideally, this would lead to a system where the rendering load is equally distributed over all stages of the rendering pipeline – CPU processing, vertex transformation, and fragment processing.

The presented contributions can be categorized in three different areas: Near field rendering of individual plants, far field rendering of landscapes, and technical aspects not directly related to vegetation, but important for the algorithms used elsewhere in this thesis. In this thesis, the following contributions are made to these areas:

Billboard Clouds for Vegetation Rendering

The *Billboard Clouds* (BBCs) extreme model simplification method introduced by Decoret [DDS03] is a highly versatile approach for reducing arbitrary models. In this method, *supporting planes* are placed such that all geometry of the original model can be projected onto these planes under a guaranteed maximum error. The original method makes no assump-

tions regarding the structure of the model and allows arbitrary plane placements; when applied to vegetation this may lead to unsatisfactory results, mostly due to planes viewed at grazing angles.

We propose a number of improvements to the original algorithm that addresses these issues [FUM05]. For example, introducing a penalty for nearly horizontal planes leads to more upright plane placement, a better representation when viewing the model in walkthrough situations. Also, the automatic placement of orthogonal supporting planes captures stems and thick branches much better than if only a single plane was used, which would cause the stem to disappear entirely when viewed edge on.

Displacement Mapped Billboard Clouds

For the near field rendering of individual plants, *Displacement Mapped Billboard Clouds* (DMBBCs, Chapter 5) are ideally suited to transition between the commonly used billboard clouds [DDS03] and more detailed approaches. Billboard clouds are essentially a collection of textured planar surfaces that capture the entire model within a given error bound. This works quite well at a sufficient distance, but at closer ranges the planarity of the primitives used in this approach become quite apparent.

DMBBCs avoid these artifacts by rendering a volumetric representation instead of simple planes. This is performed on the GPU through a ray casting fragment program. This intermediate transition can be easily blended to 'regular' billboard clouds by reducing the thickness of the volume. Also, the volumetric rendering results in a more detailed representation that is acceptable for closer view distances than regular billboard clouds.

The creation of DMBBCs is a straightforward extension of the billboard cloud generation method; in addition to the 'normal' textures the vertical offset of each fragment from the supporting plane is stored in a separate plane. Various alternatives are available for handling multiple vertical offsets per position, such as a full volumetric representation, vertical spans or individual fragments.

GPU based Landscape Rendering

For adding vegetation to very large landscapes, it is desirable to use methods that have minimal interference with existing terrain rendering algorithms. Chapter 6 demonstrates a GPU based ray casting approach that is highly suitable approach for rendering vegetation detail at a very large scale, and can be readily incorporated into existing geospatial view-

ing applications. The integration into NASA World Wind, a widely used planet-scale geospatial viewing application, is also discussed in the same chapter.

This rendering method uses a *Combined Elevation Map* (CEM) that encodes both the terrain and vegetation data suitable for ray casting. The CEM is created in a preprocessing step, using an existing elevation model and arbitrary information on vegetation coverage. This vegetation data can be extracted semi-automatically from topographic maps, or georeferenced GIS data can be used if available.

At runtime, the rendered geometry is only used as a starting point for a GPU-based ray casting of the combined elevation map. Therefore, the terrain can be rendered at relatively coarse levels. After the correct intersection point has been found, local land coverage information can be used to choose the correct shading model. For example, the surface color can be modulated based on vegetation height, and ray cast reflections can be calculated for water surfaces.

Since large landscapes necessarily require tiling multiple textures, the presented method includes solutions for avoiding artifacts at tile boundaries. The correct initial intersection can be guaranteed by using vertical skirts at tile boundaries, but this is not sufficient if illumination or reflections should be cast as they may extend further into the neighboring tile. We propose to use overlapping textures to reduce these 'secondary' artifacts.

Fast Approximate Visible Sets

Point based representations are often used as an intermediate representations of geometric models [DCSD02], and for very dense clouds of point samples such as those obtained through laser range scans. Since point samples do not include any inherent visibility or occlusion information, it is usually necessary to render the entire data set even if this results in considerable overdraw.

In Chapter 3, we present an approach to quickly determine view dependent approximate visible set of points, which can be used to significantly reduce the amount of data to be rendered. Although these sets are only an approximation, errors are typically small and – especially for rendering vegetation – produce little visible artifacts.

In a preprocessing step, for each view direction a number of nearby views is rendered. Point samples are drawn with an unique identifier encoded in their color values. The resulting image can then be read back and scanned for the identifiers that remained visible. This information

is stored for all nearby views and combined to produce the approximate visible set for this view direction.

At runtime, the approximate visible sets for the closest available view directions are combined, such that smooth transitions between different sets are possible. Only a fraction of the total number of points is rendered for arbitrary view positions, resulting in a significant speedup.

Early-Z Optimization

GPU based fragment shaders are being increasingly used for rendering surface detail. In many cases of such shaders it is necessary to change the output depth (z) value in addition to color to produce correct intersections between objects.

However, this collides with *Early-Z* culling, an acceleration method employed in modern graphics hardware in which a fragment is discarded before executing the fragment program if it is hidden according to the z-buffer. Obviously, this is not possible if the fragment shader modifies the z value, since this may result in previously hidden (and thus culled) fragments becoming visible. Therefore, currently graphics drivers disable this acceleration if a shader is detected that modifies z values.

However, many shaders that modify z values in a very predictable way, such that this culling could still be employed. Most displacement mapping shaders use an 'outer shell' geometry to produce starting points for ray casting, such that the modified z values can be guaranteed to be larger than the original value. Therefore, previously occluded fragments are still hidden in all cases, and the Early-Z acceleration could remain enabled.

Also, if maximum displacement bounds are known (which is also often the case), the applicable range of Early-Z could be further extended to include other rendering modes and more general fragment programs.

In Chapter 9, we describe in detail how the various approaches to rendering displacement maps and the rendering modes of OpenGL and DirectX could be extended to provide Early-Z in as many cases as possible. Many of the proposed changes are very likely only driver changes, requiring no changes to GPU hardware and may therefore be also applicable to existing systems.

Contributions in Context

Table 10.1 summarizes the vegetation specific contributions of this thesis in the context of the state of the art algorithms presented in Chapter 2.

	Description	Lea. / Br. / Full	View dep. LOD	Animation	Dyn. Illum.	Quality	Performance	Memory	Distance range (near – far)
	Plant Leaves [WWD+05]	L			✓	●	○	◐	
	Complex Botanical Scenes [MFC97]	F	✓	–	✓	◐	◐	○	
	Complex Plant Ecosystems [DCSD02]	F	✓	–		●	◐	◐	
	Point Based Vegetation Rendering (Chp. 4)	F	✓	–		◐	●	◐	
	Plant Models with Complex Organs [ZBJ06]	F	✓	–	✓	●	◐	◐	
	Point-Based Trees [GMN05]	F	✓	–		◐	◐	◐	
	Procedural Multiresolution [LCV03]	B	✓		✓	●	◐	○	
	Complex Photorealistic Landscapes [CCDH05]	F	–	–	✓	◐	◐	○	
	Displacement Mapped Billboard Clouds (Chp. 5)	F	✓	–		◐	◐	◐	
	Vegetation Specific Billboard Clouds (Chp. 4)	F	✓	–		◐	●	◐	
	Fractal Plants and Trees [Opp86]	B		✓	✓	○	○	●	
	Single Polygonal Mesh [LVF+01]	B			✓	●	◐	◐	
	Volumetric Reconstruction [RMMD04]	F	✓	–	–	●	○	●	
	Interactive Forest [GCF01]	B	✓	✓	✓	◐	◐	○	
	Forests at Human Scale [SK04]	F		–	–	◐	●	◐	
	Realistic Trees [WP95]	F	✓		✓	◐	◐	◐	
	Shading and Shadowing [MNP01]	F	✓	–	✓	●	◐	●	
	Hierarchical Image-Based Rendering [MDK99]	L	✓	–		◐	◐	●	
	Structured Particle Systems [RB85]	F		–	✓	◐	○	○	
	Image-Based Multiresolution [LCV04]	F	✓	–	–	◐	◐	◐	
	Sequential Point Trees [DVS03]	F	✓	–		◐	●	◐	
	Deferred Splatting [GBP04]	F	✓	–		◐	◐	◐	
	View-Dependent Multiresolution [RCRB03]	L	✓	–	✓	◐	●	◐	
	Forest Scenes in Real-Time [DN04]	F	–	–	–	●	◐	◐	
	Slicing and Blending [Jak00]	F		–		◐	●	○	
	Precomputed Z-Buffer Views [Max96]	L		–	✓	◐	◐	◐	
	Drop and Resize [Hal01]	F	✓	–	–	◐	●	◐	
	Textured Quadric Surfaces [Gar84]	F		–		○	●	○	
	GPU Based Ray Casting (Chp. 6)	F	✓	–	✓	◐	◐	◐	

Table 10.1: Contributions of this thesis in the context of the previous state of the art. See also Table 2.1.

10.2 Research Outlook

Referring to Table 2.1 as well as the presented algorithms, it is apparent that algorithms exist to render vegetation over a very wide range of view distances. However, several aspects remain that deserve further investigation:

Data size Perhaps the most prominent issue is still the amount of data to be handled for large landscapes. Although algorithms exist that avoid the management of explicit plant positions for large areas, these data must be created dynamically as the viewer approaches. Also, the synthesized positions and vegetation features must correlate visually with the implicit representations, but still exhibit sufficient variation to avoid obvious repetitions.

Animation Only very few algorithms support animation of vegetation, an important aspect that greatly enhances the realism of natural scenes. This problem is further complicated because it is even more difficult to transition between rendering methods in a way that does not disturb animation.

Illumination The same problem can be said to exist for dynamic illumination, which is also only supported by a limited number of algorithms. Also, a better illumination model for very large areas would greatly enhance the visual quality of the far field. Bidirectional reflectance distribution functions of forest canopies have been measured for remote sensing purposes, and it would be a promising goal to integrate such models in an interactive rendering system.

In summary, interactive rendering of vegetation has improved significantly since its early days. However, additional research is required for creating dynamic scenes as well as building new approaches to managing very large areas of vegetation.

CHAPTER 11

CONCLUSIONS

I may not have gone
where I intended to go,
but I think I have ended up
where I needed to be.

Douglas Adams

With the increasing popularity of geospatial viewers such as Google Earth and World Wind, it can be expected that the inclusion of vegetation in these applications will remain an worthwhile topic in the near future. Most current applications that do include vegetation are typically limited to relatively small areas, and solutions that work well on a very large - or even global - scale have yet to be found. Although we believe that this work is one step in this direction, much remains to be done.

Solutions for very large scales are only feasible if generic models are used; however these must be converted to explicit geometry at some point to accommodate close up views. A visually unobtrusive transition is necessary between these representations, which is surprisingly difficult to achieve. For example, explicit instances must be placed at specific locations, and the geometry may include local variations. These features must blend well with the necessarily generic model used for the far field, such that its general shape is retained and trees do not appear to suddenly grow out of the canopy as the viewer comes closer. Non-repetitive tiling approaches (such as Wang tiles [CSHD03]) are helpful, but require

special care for boundary regions that do not coincide with tile boundaries.

This problem also includes illumination calculations. Although the reflectance behavior of a forest is mainly determined by its leaf canopy, this is extremely difficult to model. BRDF functions of various species of trees have been estimated for remote sensing purposes, but it is not known if this data has been incorporated in a rendering framework. Illumination is also problematic at the very near scale. Inside a forest, the complex interaction of scattering and translucency is quite hard to model. Methods exist for individual plant leaves and small plants [WWD+05], but the interaction within an entire canopy is fare more complex.

Additionally, the ground inside a forest typically abounds with leaves, broken twigs, and undergrowth. Such detail is often added manually in existing applications, but it would be preferable to create such detail algorithmically.

A successful solution will likely combine several approaches to rendering vegetation at different distances. Ultimately, we believe that such a solution will have to be tightly coupled with a plant growth simulation, such that generic representations for far field renderings can be easily transitioned to individual models for the near field.

In closing, we believe that it will remain impossible to beat a walk through a *real* forest for a long time to come.

List of Figures

List of Tables

BIBLIOGRAPHY

[ABC+04] Carlos Andújar, Pere Brunet, Antoni Chica, Isabel Navazo, Jarek Rossignac, and Alvar Vinacua. Computing maximal tiles and application to impostor-based simplification. *Computer Graphics Forum*, 23(3):401–410, September 2004. 52

[AIK+05] Michael Hugh Anderson, Ann Chris Irvine, Nidish Ramachandra Kamath, Chun Yu, Dan Minglun Chuang, Yshi Tian, and Yingyong Qi. Graphics pipeline and method having early depth detection. US Patent Application Publication No. US 2005/0195198 A1, 2005. 134

[AMCH+04] Tomas Akenine-Moeller, Eric Chan, Wolfgang Heidrich, Jan Kautz, Mark Kilgard, and Marc Stamminger. Real-time shadowing techniques. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, page 27, New York, NY, USA, 2004. ACM Press. 90

[AS99] Patrice Arrighi and Pierre Soille. From scanned topographic maps to digital elevation models. In D. Jongmans, E. Pirard, and P. Trefois, editors, *Proc. of Geovision'99: International Symposium on Imaging Applications in Geology*, pages 1–4. Université de Liège, Belgium, May 1999. 83

[ATI05] ATI Technologies Inc. Radeon X1800 shader architecture whitepaper. Technical report, ATI Technologies Inc., 2005. 134

[BCF+05] Stephan Behrendt, Carsten Colditz, Oliver Franzke, Johannes Kopf, and Oliver Deussen. Realistic real-time rendering of landscapes using billboard clouds. In *Computer Graphics Forum*, volume 24, pages 507–516, 2005. 29

[BG01] Eric Bouvier and Enrico Gobbetti. TOM: Totally ordered mesh a multiresolution structure for time critical graphics applications. *International Journal of Image and Graphics*, 1(1):115–134, 2001. 126

[Bio03] Bionatics. Bionatics homepage. web page, 2003. http://www.bionatics.com/. 23

[Bli78] James F. Blinn. Simulation of wrinkled surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 286–292, New York, NY, USA, 1978. ACM Press. 61, 79

[BMG06] Frédéric Boudon, Alexandre Meyer, and Christophe Godin. Survey on computer representations of trees for realistic and efficient rendering. Rapport de recherche 2301, LIRIS, Université Claude Bernard Lyon 1, 2006. 8

[CAZ01] Jonathan D. Cohen, Daniel G. Aliaga, and Weiqiang Zhang. Hybrid simplification: combining multi-resolution polygon and point rendering. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 37–44, Washington, DC, USA, 2001. IEEE Computer Society. 114

[CCDH05] Carsten Colditz, Liviu Coconu, Oliver Deussen, and Hans-Christian Hege. Real-time rendering of complex photorealistic landscapes using hybrid level-of-detail approaches. In E. Buhmann, P. Paar, I. Bishop, and E. Lange, editors, *Trends in Real-Time Landscape Visualization and Participation*, pages 97–106. Wichmann Verlag, 2005. 29, 35, 51, 146

[Che95] Shenchang Eric Chen. Quicktime VR - an image-based approach to virtual environment navigation. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 29–38. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995. 8

[CN01] Baoquan Chen and Minh Xuan Nguyen. Pop: a hybrid point and polygon rendering system for large data. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 45–52, Washington, DC, USA, 2001. IEEE Computer Society. 114

[CSHD03] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 287–294, New York, NY, USA, 2003. ACM Press. 83, 149

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In James T. Kajiya, editor, *SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, pages 279–288. ACM SIGGRAPH, Addison Wesley, August 1993. ISBN 0-201-51585-7. 23

[DCSD02] Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive visualization of complex plant ecosystems. In *IEEE Visualization '02*, October 2002. 19, 20, 33, 35, 39, 40, 144, 146

[DDS03] Xavier Decoret, Fredo Durand, and Francois X. Sillion. Billboard clouds. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, pages 376–376, New York, NY, USA, 2003. ACM Press. 29, 51, 52, 61, 64, 74, 142, 143

[Deu03] Oliver Deussen. *Computergenerierte Pflanzen*. Springer Verlag, 2003. 7, 8

[DH72] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, 1972. 52

[DLP05] Jean-François Dufort, Luc Leblanc, and Pierre Poulin. Interactive rendering of meso-structure surface details using semi-transparent 3d textures. In *Proc. Vision, Modeling, and Visualization 2005*, pages 399–406, November 2005. 61, 79

[DMBF96] William J. Dally, Leonard McMillan, Gary Bishop, and Henry Fuchs. The delta tree: An object-centered approach to image-based rendering. Technical Report AIM-1604, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1996. 23

[DMS06] Andreas Dietrich, Gerd Marmitt, and Philipp Slusallek. Terrain guided multi-level instancing of highly complex plant populations. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 169–176, September 2006. 7

[DN04] Philippe Decaudin and Fabrice Neyret. Rendering forest scenes in real-time. In *Rendering Techniques (Eurographics Symposium on Rendering - EGSR)*, pages 93–102, june 2004. 29, 31, 35, 36, 146

[Don05] William Donnelly. *Per-Pixel Displacement Mapping With Distance Functions*, pages 123–136. Addison-Wesley, 2005. 89

[DSSD99] Xavier Decoret, François Sillion, Gernot Schaufler, and Julie Dorsey. Multi-layered impostors for accelerated rendering. In *Computer Graphics Forum (Proc. Eurographics '99)*, pages 61–73. Eurographics, Blackwell Publishers, September 1999. ISSN 1067-7055. 23

[DSV98] Lucia Darsa, Bruno Costa Silva, and Amitabh Varshney. Walkthroughs of complex environments using image-based simplification. *Computers & Graphics*, 22(1):55–69, 1998. 23

[Dum06] Jonathan Dummer. Cone step mapping: An iterative ray-heightfield intersection algorithm. http://www.mganin.com/lonesock/ConeStepMapping pdf, 2006. 62, 89

[DVS03] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 657–662, New York, NY, USA, 2003. ACM Press. 19, 20, 22, 33, 35, 146

[DWS+97] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. In *IEEE Visualization '97*, October 1997. 127

[FS93] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH 93 Conference Proceedings*, pages 247–254, 1993. 113, 127

[FUM05] Anton L. Fuhrmann, Eike Umlauf, and Stephan Mantler. Extreme model simplification for forest rendering. In E. Galin and P. Poulin, editors, *Proceedings of the 2005 Eurographics Workshop on Natural Phenomena*. The Eurographics Association, 2005. 29, 143

[Gar84] Geoffrey Y. Gardner. Simulation of natural scenes using textured quadric surfaces. In *SIGGRAPH 1984, Conference Proceedings*. ACM Press / ACM SIGGRAPH, 1984. 9, 10, 35, 146

[GB99] Enrico Gobbetti and Eric Bouvier. Time-critical multiresolution scene rendering. In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 123–130, San Francisco, 1999. 114, 126

[GB00] E. Gobbetti and E. Bouvier. Time-critical multiresolution rendering of large complex models. In *Journal of Computer-Aided Design*, pages 785–803, 2000. 114

[GBP04] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Deferred splatting. *Comput. Graph. Forum*, 23(3):653–660, 2004. 21, 22, 35, 146

[GCF01] Thomas Di Giacomo, Stéphane Capo, and François Faure. An interactive forest. In Marie-Paule Cani, Nadia Magnenat-Thalmann, and Daniel Thalmann, editors, *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 65–74. Springer, sept. 2001. Manchester. 13, 35, 146

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. 23

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. 141

[GK93] Ned Greene and Michael Kass. Hierarchical Z-buffer visibility. In James T. Kajiya, editor, *SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, pages 231–238. ACM SIGGRAPH, Addison Wesley, August 1993. ISBN 0-201-51585-7. 8, 9, 132

[GL03] Francois Gougeon and Don Leckie. Forest information extraction from high spatial resolution images using an individual tree crown approach. PFC Information Report BC-X-396, Canadian Forest Service, 2003. 82

[GMN05] Guillaume Gilet, Alexandre Meyer, and Fabrice Neyret. Point-based rendering of trees. In E. Galin and P. Poulin, editors, *Eurographics Workshop on Natural Phenomena*, 2005. 22, 35, 146

[GP03] Gael Guennebaud and Mathias Paulin. Efficient screen space approach for Hardware Accelerated Surfel Rendering . In *Vision, Modeling and Visualization , Munich, 19/11/03-21/11/03*, pages 485–495. IEEE Signal Processing Society, novembre 2003. 22

[Hal01] Nick Halper. Drop and resize with billboards. Web page, 2001. http://isgwww.cs.uni-magdeburg.de/ nick/billboards/billboards.html. 25, 35, 36, 146

[HEGD04] Johannes Hirche, Alexander Ehlert, Stefan Guthe, and Michael Doggett. Hardware accelerated per-pixel displacement mapping. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 153–158, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society. 61, 79, 86

[HLB+05] Mark Harris, David Luebke, Ian Buck, Naga Govindaraju, Jens Krüger, Aaron Lefohn, Tim Purcell, and Cliff Woolley. A survey of general-purpose computation on graphics hardware. Siggraph 2005 Course Notes, 2005. 131

[Hop96] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. 141

[Hop97] Hugues Hoppe. View-dependent refinement of progressive meshes. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. 115

[Hop98] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 35–42, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press. 69, 115, 127

[Hud05] Richard Huddy. Directx graphics performance: Getting every bit you can. Technical report, ATI Technologies Inc., 2005. 133

[Int02] Interactive Data Visualization, Inc. Speedtree product homepage. web page, 2002. http://www.idvinc.com/speedtree/. 9, 39, 51

[Jak00] A. Jakulin. Interactive vegetation rendering with slicing and blending. In A. de Sousa and J.C. Torres, editors, *Proc. Eurographics 2000 (Short Presentations)*. Eurographics, August 2000. 24, 26, 35, 39, 146

[JWP05] Stefan Jeschke, Michael Wimmer, and Werner Purgathofer. Image-based representations for accelerated rendering of complex scenes. In Y. Chrysanthou and M.Magnor, editors, *EUROGRAPHICS 2005 State of the Art Reports*, pages 1–20. EUROGRAPHICS, The Eurographics Association and The Image Synthesis Group, August 2005. 61

[KGG98] Michael Kofler, Michael Gervautz, and Michael Gruber. The styria flyover - lod management for huge textured terrain models. In F.-E.Wolter and N.M.Patrikalakis, editors, *Computer Graphics International*, Hannover, Germany, June 1998. IEEE Computer Society. CGI'98. 115, 121

[KH84] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 165–174, New York, NY, USA, 1984. ACM Press. 90

[KIK+01] T. Kaneko, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda, T. Takahei, and S. Tachi. Detailed shape representation with parallax mapping. In *Proceedings of ICAT 2001*, pages 205–208, 2001. 61, 79

[KK89] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 271–280, New York, NY, USA, 1989. ACM Press. 79

[KRS05] Andreas Kolb and Christof Rezk-Salama. Efficient empty space skipping for per-pixel displacement maps. In *Proceedings of the VMV 2005 Conference*, 2005. 62, 85, 88

[KW03] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization 2003*, pages 287–292, 2003. 131

[LCV03] Javier Lluch, Emilio Camahort, and Roberto Vivó. Procedural multiresolution for plant and tree rendering. In *AFRIGRAPH '03: Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa*, pages 31–38, New York, NY, USA, 2003. ACM Press. 17, 35, 146

[LCV04] Javier Lluch, Emilio Camahort, and Roberto Vivó. An image-based multiresolution model for interactive foliage rendering. In *WSCG*, pages 507–514, 2004. 25, 35, 146

[LE97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 199–208. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7. 127

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 31–42. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. 23

[LP01] Peter Lindstrom and Valerio Pascucci. Visualization of large terrains made easy. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 363–371, Washington, DC, USA, 2001. IEEE Computer Society. 115, 127

[LV00] Tom Lokovic and Eric Veach. Deep shadow maps. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 385–392, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 90

[LVF+01] Javier Lluch, M.J. Vicent, S. Fernandez, C. Monserrat, and Roberto Vivo. Modelling of branched structures using a single polygonal mesh. In *IASTED International Conference on Visualization, Imaging, and Image Processing*, 2001. 12, 35, 146

[LW85] Marc Levoy and Turner Whitted. The use of points as display primitives. Technical Report TR85-022, Department of Computer Science, University of North Carolina - Chapel Hill, October 1 1985. 17, 114

[Map02] Ian Mapleson. Sgi graphics performance comparison tables. Web Page, 5 2002. 1

[Mas99] Ashton E. W. Mason. *Predictive Hierarchical Level of Detail Optimization*. PhD thesis, University of Cape Town, 1999. 113

[Max96] Nelson Max. Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In Xavier Pueyo and Peter Schröder, editors, *Rendering Techniques '96 (Proceedings of the Eurographics Workshop on Rendering 96)*, pages 165–174. Eurographics, Springer-Verlag Wien New York, June 1996. ISBN 3-211-82883-4. 35, 146

[MB95] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 39–46. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995. 23

[MB97] Ashton E. W. Mason and Edwin H. Blake. Automatic hierarchical level of detail optimization in computer animation. *Computer Graphics Forum*, 16(3):C191–C199, 1997. 113, 116, 117

[MDK99] Nelson Max, Oliver Deussen, and Brett Keating. Hierarchical image-based rendering using texture mapping hardware. In *Eurographics Workshop on Rendering 1999*, pages 57–62, 1999. 24, 35, 146

[MFC97] Dana Marshall, Donald Fussell, and A. T. Campbell III. Multiresolution rendering of complex botanical scenes. In Wayne A. Davis, Marilyn Mantei, and R. Victor Klassen, editors, *Graphics Interface '97*, pages 97–104. Canadian Human-Computer Communications Society, 1997. 11, 35, 146

[MH02] Tomas Möller and Eric Haines. *Real-Time Rendering, Second Edition*. A. K. Peters Limited, 2002. 8, 9

[Mic05] Microsoft Corporation. MSDN DirectX 9 documentation library, 2005. 135

[MLK+04] Takaharu Miyoshi, Weiqing Li, Kazufumi Kaneda, Hideo Yamashita, and Eihachiro Nakamae. Automatic extraction of buildings utilizing geometric features of a scanned topographic map. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3*, pages 626–629, Washington, DC, USA, 2004. IEEE Computer Society. 83

[MM05] Morgan McGuire and Max McGuire. Steep parallax mapping. In *I3D 2005 Poster*, 2005. 61

[MNP01] Alexandre Meyer, Fabrice Neyret, and Pierre Poulin. Interactive rendering of trees with shading and shadowing. In *Workshop on Rendering*, Eurographics. Springer-Verlag Wien New York, July 2001. 27, 28, 34, 35, 39, 146

[MO95] Nelson Max and Keiichi Ohsaki. Rendering trees from precomputed Z-buffer views. In *Proceedings of the 6th Eurographics Workshop on Rendering*, 1995. 23

[MS95] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 95–102. ACM SIGGRAPH, ACM Press, April 1995. ISBN 0-89791-736-7. 113

[MUN91] Scott N. Martens, Susan L. Ustin, and John M. Norman. Measurement of tree canopy architecture. *Journal of Remote Sensing*, 12:1525–1545, 1991. 90

[NAS] NASA. World wind homepage. http://worldwind.arc.nasa.gov/. 88

[Ney98] Fabrice Neyret. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):55–70, 1998. 79

[NVI05] NVIDIA Corporation. Nvidia gpu programming guide. Technical report, NVIDIA Corporation, 2005. 133

[OBM00] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief texture mapping. In Kurt Akeley, editor, *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 359–368. ACM SIGGRAPH, Addison Wesley, July 2000. 132

[OP05] Manuel Oliveira and Fabio Policarpo. An efficient representation for surface details. Technical Report RP-351, Instituto de Informática UFRGS, 2005. 79, 87

[Opp86] Peter Oppenheimer. Real time design and animation of fractal plants and trees. In David C. Evans and Rusell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, pages 55–64, August 1986. 10, 35, 146

[Paj98] Renato Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. *VIS '98: Proceedings of the conference on Visualization '98*, pages 19–26, 1998. 127

[PBFJ05] Serban D. Porumbescu, Brian Budge, Louis Feng, and Kenneth I. Joy. Shell maps. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 626–633, New York, NY, USA, 2005. ACM Press. 61, 79

[PF05] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005. 62, 68

[PO06] Fabio Policarpo and Manuel M. Oliveira. Relief mapping of non-height-field surface details. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 55–62, New York, NY, USA, 2006. ACM Press. 62

[POC05] Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 155–162, New York, NY, USA, 2005. ACM Press. 62, 79, 86, 131, 132, 133

[Pol06] Darren E. Polkowski. Geforce 8800: Here comes the DX10 boom. Web page, 11 2006. 2

[PTS99] Simon Premoze, William B. Thompson, and Peter Shirley. Geospecific rendering of alpine terrain. In Dani Lischinski and Gregory Ward Larson, editors, *Rendering Techniques '99, Proceedings of the Eurographics Workshop in Granada, Spain, June 21-23, 1999*, pages 107–118. Springer, 1999. 90, 96

[Pup98] Enrico Puppo. Variable resolution triangulations. In *WADS '97: Selected papers presented at the international workshop on Algorithms and data structure*, pages 219–238, New York, NY, USA, 1998. Elsevier Science Inc. 127

[PZvBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In Kurt Akeley, editor, *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 335–342. ACM SIGGRAPH, Addison Wesley, 2000. 17, 114

[QQZ+03] Huamin Qu, Feng Qiu, Nan Zhang, Arie Kaufman, and Ming Wan. Ray tracing height fields. In *Proceedings of Computer Graphics International 2003*, pages 202–209, Los Alamitos, CA, USA, 2003. IEEE Computer Society. 79, 86

[RB85] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *SIGGRAPH 85, Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, 1985. 17, 18, 35, 146

[RCB+] Inmaculada Remolar, Miguel Chover, Oscar Belmonte, José Ribelles, and Cristina Rebollo. Geometric simplification of foliage. 126

[RCB+02] Immaculada Remolar, Miguel Chover, Oscar Belmonte, Jose Ribelles, and Cristina Rebollo. Real-time tree rendering. Technical report, Departamento de Lenguajes y Sistemas Informaticos, Universitat Jaume I, Campus de Riu Sec, E-12080 Castellon, Spain, 2002. Informe Tecnico DLSI 1/3/2002. 14

[RCRB03] Immaculada Remolar, Miguel Chover, Jose Ribelles, and Oscar Belmonte. View-dependent multiresolution model for foliage. In *Journal of WSCG (WSCG'2003 Proceedings)*, February 2003. 14, 16, 25, 35, 126, 146

[RL00] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In Kurt Akeley, editor, *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 343–352. ACM SIGGRAPH, Addison Wesley, July 2000. 19, 40, 114

[RLIB99] Martin Reddy, Yvan Leclerc, Lee Iverson, and Nat Bletter. Terravision ii: Visualizing massive terrain databases in vrml. *IEEE Comput. Graph. Appl.*, 19(2):30–38, 1999. 121

[RMMD04] Alex Reche-Martinez, Ignacio Martin, and George Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 720–727, New York, NY, USA, 2004. ACM Press. 31, 35, 146

[SA03] Mark Segal and Kurt Akeley. The opengl graphics system: A specification. Technical report, Silicon Graphics, Inc., 2003. 135

[Sch95] Gernot Schaufler. Dynamically generated impostors. In Dieter W. Fellner, editor, *GI Workshop on Modeling, Virtual Worlds,*, pages 129–135, November 1995. 23, 103, 106, 107

[Sch98] Gernot Schaufler. Per-object image warping with layered impostors. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98 (Proceedings of the Eurographics Workshop on Rendering 98)*, pages 145–156. Springer-Verlag Wien New York, June 1998. 23, 24

[SF99] Dieter Schmalstieg and Anton L. Fuhrmann. Coarse view-dependent levels of detail for hierarchical and deformable models. Technical report, Vienna University of Technology, 1999. 114

[SH01] Bernd-Michael Straub and Christian Heipke. Automatic extraction of trees for 3d-city models from images and height data. *Automatic Extraction of Man-Made Objects from Aerial and Space Images*, III:267–277, 2001. 82

[SK04] Gabor Szijártó and József Koloszár. Real-time hardware accelerated rendering of forests at human scale. In *WSCG*, pages 443–450, 2004. 28, 30, 35, 146

[SLS+96] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996. 23, 104

[SS96] Gernot Schaufler and Wolfgang Stürzlinger. A three-dimensional image cache for virtual reality. *Computer Graphics Forum (Proc. Eurographics '96)*, 15(3):227–235, September 1996. ISSN 0167-7055. 104

[SS01] Randolf Schultz and Heidrun Schumann. Automatic Instancing of Hierarchically Organized Objects. In *Spring Conference on Computer Graphics 2001, Conference Proceedings*, Budmerice, Slovakia, 2001. 9

[STM05] Pedro V Sander, Natalya Tatarchuk, and Jason L. Mitchell. Explicit early-z culling for efficient fluid flow simulation and rendering. Technical report, ATI Application Research Group, 2005. 133

[Tat06] Natalya Tatarchuk. Dynamic parallax occlusion mapping with approximate soft shadows. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, pages 63–69, 2006. 61, 79, 131, 132, 138

[Ulr02] Thatcher Ulrich. Rendering massive terrains using chunked level of detail control. In *SIGGRAPH 2002 Course Notes CD-ROM*. Association for Computing Machinery, ACM SIGGRAPH, August 2002. Course 35. 88, 115

[Uni] Gevorg Grigoryan University. Probabilistic surfaces: Point based primitives to show surface uncertainty. 40

[Was05] Scott Wasson. Nvidia's geforce 7800 gtx graphics processor. Technical report, The Tech Report, LLC, 2005. 134

[Wel04] Terry Welsh. Parallax mapping with offset limiting: A per pixel approximation of uneven surfaces. Technical report, Infiscape Corporation, 2004. 61, 79

[WFadH00] Michael Wand, Matthias Fischer, and Friedhelm Meyer auf der Heide. Randomized point sampling for output-sensitive rendering of complex dynamic scenes, 2000. 17, 40

[WFP+01] 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 Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 361–370. ACM Press / ACM SIGGRAPH, 2001. 9, 21

[WP95] Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 119–128. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995. 18, 35, 39, 40, 45, 121, 146

[WP05] Alan Watt and Fabio Policarpo. *Advanced Game Development with Programmable Graphics Hardware*. A. K. Peters, Ltd., Natick, MA, USA, 2005. 62

[WS98] Michael Wimmer and Dieter Schmalstieg. Load balancing for smooth lods. Technical report, Vienna University of Technology, 1998. 114

[WTL+04] Xi Wang, Xin Tong, Stephen Lin, Shi-Min Hu, Baining Guo, and Heung-Yeung Shum. Generalized displacement maps. In Alexander Keller and Henrik Wann Jensen, editors, *Rendering Techniques*, pages 227–234. Eurographics Association, 2004. 62, 79

[WWD+05] Lifeng Wang, Wenle Wang, Julie Dorsey, Xu Yang, Baining Guo, and Heung-Yeung Shum. Real-time rendering of plant leaves. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 712–719, New York, NY, USA, 2005. ACM Press. 32, 35, 146, 150

[WWT+03a] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. View-dependent displacement mapping. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 334–339, New York, NY, USA, 2003. ACM Press. 62

[WWT+03b] Lifeng Wang, Xi Wang, Xin Tong, Steve Lin, Shimin Hu, Baining Guo, and Harry Shum. View-dependent displacement mapping. In *Proceedings of SIGGRAPH 2003*, pages 334–339, 2003. 131

[ZBJ06] Xiaopeng Zhang, Frédéric Blaise, and Marc Jaeger. Multiresolution plant models with complex organs. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 331–334, New York, NY, USA, 2006. ACM Press. 14, 15, 35, 146

[ZKHK03] Christopher Zach, Andreas Klaus, Markus Hadwiger, and Konrad Karner. Accurate dense stereo reconstruction using graphics hardware, 2003. 3

[ZSBP02] Wenting Zheng, Hanqiu Sun, Hujun Bao, and Qunsheng Peng. Rendering of virtual environments based on polygonal and point-based models. In *VRST'02, Conference Proceedings*, Hong Kong, 2002. ACM Press / ACM SIGGRAPH. 17

CURRICULUM VITAE

Name	Dipl.-Ing. Stephan Mantler
Date of birth	December 1, 1974 in Vienna, Austria
Address	Payergasse 1/17 A 2340 Mödling Austria step@stephanmantler.com

Education

1981-1985	Elementary school, VS Auhofstrasse, Vienna
1985-1989	High school, Goethe-Gymnasium, Astgasse, Vienna
1989-1993	High school, BRG XV, Diefenbachgasse, Vienna. Graduation with honors.
1993-1998	Studies of computer science at Vienna University of Technology. Masters Thesis *Dynamic Load Balancing in Distributed Virtual Environments*.
1999-2006	Studies of sports science at University of Vienna
2006	state certified swimming instructor (with distinction)

Employment History

1995-2001	numerous freelance projects
1998-1999	software developer, Imagination Computer Software GmbH
2000-2001	systems engineer, Cable & Wireless Austria
2001-2006	Austrian trade license for IT services and consulting
2002-2003	Junior researcher, VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH
2003-2004	Temporary research assistantship, Department of Robotics, University of Duisburg, Germany
2004-	back at VRVis
2005-	External lecturer, FH Technikum Wien, Vienna

Previous Publications

Stephan Mantler. Dynamic Load Balancing in Distributed Virtual Environments. Masters Thesis, 1998.

Paolo Petta, Alexander Staller, Robert Trappl, Stephan Mantler, Zsolt Szalavari, Thomas Psik, and Michael Gervautz. Towards Engaging Full-Body Interaction. In *Proceedings of the 8th International Conference on Human-Computer Interaction* (HCI International '99), 1999.

Kurt Hofstetter, Barbara Doser, Rainer Haslwandter, Stephan Mantler and others. The sunpendulum Book of Documents. 2002. Self published.

Christopher Zach, Stephan Mantler and Konrad Karner. Time-critical Rendering of Discrete and Continuous Levels of Detail. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (Hong Kong, China, November 11 - 13, 2002).

Stephan Mantler and Anton L. Fuhrmann. Fast Approximate Visible Set Determination for Point Sample Clouds. In *Proceedings of the Workshop on Virtual Environments 2003* (Zurich, Switzerland, May 22 - 23, 2003).

Stephan Mantler and Anton L. Fuhrmann. The State of The Art in Real-Time Rendering of Vegetation. VRVis Technical Report TR-2003-027. 2003.

Stephan Mantler and Anton L. Fuhrmann. Point Based Rendering of Massive Data Sets: A Case Study. In *Proceedings of the Computer Graphics international (Cgi'04) - Volume 00* (June 16 - 19, 2004).

Stephan Mantler, Gerd Hesina, Stefan Maierhofer, and Robert F. Tobler. Real-Time Rendering of Vegetation and Trees in Urban Environments. In *Proceedings of the CORP'2005 Conference* (Vienna, Austria, 2005).

Stephan. Mantler, Markus Hadwiger and Christian Sigg. Saving the Z-Cull Optimisation. In submission.

Stephan Mantler and Stefan Jeschke. Interactive Landscape Visualization Using GPU Ray Casting. In *Proceedings of the 4th international Conference on Computer Graphics and interactive Techniques in Australasia and Southeast Asia* (Kuala Lumpur, Malaysia, November 29 - December 02, 2006).

Stephan Mantler, Stefan Jeschke and Michael Wimmer. Displacement Mapped Billboard Clouds. 2006. VRVis Technical Report TR-2006-030; poster presented at I3D Symposium 2007.