HASSO-PLATTNER-INSTITUT
Fachgebiet Computergrafische Systeme

High-Quality Rendering of Grass in Real-Time

Masterarbeit
zur Erlangung des akademischen Grades
"Master of Science"
(M.Sc.)
im Studiengang IT-Systems Engineering
des Hasso-Plattner-Instituts an der
Universität Potsdam

vorgelegt von
Stefan Lemme

Aufgabenstellung und Anleitung:
Prof. Dr. Jürgen Döllner
Juri Engel, M.Sc.

Potsdam,
September 12, 2012

Abstract

For photo-realistic renderings of outdoor scenes as they appear in virtual 3D city and landscape models, vegetation is an important parameter. An essential component of that vegetation is grass. Large terrain surfaces are covered by grass of different types, such as meadows, forests, golf courses or sport stadiums. The reliability of renderings for virtual 3D city models can be significantly enhanced by vegetation, especially by grass.

Due to human's wide range of visual experience with natural grass, a high-quality representation is indispensable, since even small discrepancies would be recognized by the viewer, at least subconsciously. In fact, a field of grass consists of thousands of millions of single grass blades, which is an immense geometric complexity asking for a highly specialized GPU-based rendering technique in order to achieve real-time performance.

This thesis describes a system for rendering grass in real-time applications. The grass rendering system utilizes three different representations of grass. For each of these representations a respective state-of-the-art rendering method is applied to efficiently create high-quality renderings of grass. The rendering techniques cover approaches from forward rendering of geometry, over ray tracing of volume data to an image-based 2D-texturing method. The benefits and drawbacks of these approaches as well as the consolidation into a level-of-detail concept are examined. Thereby, the management of the terrain surface covered with grass is considered as well. The implementation of several features such as the illumination, the imitation of natural variation and the handling of bumpy terrain surfaces are described in detail. Furthermore, a simple approach to generate the aforementioned utilized grass representations is explained.

In conclusion, this thesis evaluates the presented grass rendering system referring to the quality of renderings and its performance. The results are outlined and compared with respect to essential aspects.

Zusammenfassung

Für photorealistische Darstellungen von Außenszenen, wie sie in virtuellen 3D-Stadt- und Landschaftsmodellen vorkommen, spielt Vegetation eine wesentliche Rolle. Hierbei nimmt Gras als Kernanteil der Vegetation eine bedeutende Position ein. Große Geländeoberflächen, wie zum Beispiel Wiesen, Wälder, Golfplätze oder Sportstadien, sind mit Grasstrukturen unterschiedlichen Typs bedeckt. Aufgrund der umfangreichen Seherfahrung des Menschen in Bezug auf natürliche Beschaffenheiten und in diesem Zusammenhang besonders mit Blick auf die Struktur von Gras ist eine qualitativ hochwertige Repräsentation jener Vegetationsoberflächen unerlässlich, da bereits kleine Abweichungen dem Betrachter zumindest unterbewusst auffallen. Vegetation und insbesondere Gras können die Glaubwürdigkeit der Darstellung eines virtuellen 3D-Stadtmodells erheblich unterstützen.

Eine Rasenfläche besteht aus mehreren Millarden einzelner Grashalme. Die Repräsentation einer derartig hohen Anzahl bedeutet eine immense geometrische Komplexität, die die Nutzung hochspezialisierter, GPU-basierter Rendering-Verfahren notwendig macht, um interaktive Darstellungen erzeugen zu können.

Diese Arbeit präsentiert ein System zur Bildsynthese von Gras in Echtzeitanwendungen. In diesem Gras-Rendering-System werden drei verschiedene Repräsentationen für Gras eingesetzt. Für jede dieser Repräsentationen wird ein aktuelles Rendering-Verfahren verwendet, um in effizienter Weise qualitativ hochwertige Darstellungen des Grases zu erzeugen. Die Rendering-Verfahren umfassen Ansätze zum Forward-Rendering von Geometrie, Ray-Tracing für Volumendaten und eine bildbasierte Methode durch Texturierung. Die Vor- und Nachteile dieser Verfahren und deren Zusammenführung in einem Level-of-Detail-Konzept werden umfangreich beleuchtet. Dabei spielt auch das Management der mit Gras bedeckten Geländeoberflächen eine wichtige Rolle. Die Umsetzung einer Auswahl von Funktionalitäten (Features), wie der Beleuchtung, dem Nachahmen von natürlicher Variation und die Handhabung von unebenen Geländemodellen wird detailliert beschrieben. Desweiteren wird ein einfaches Verfahren zur Generierung der verwendeten Grasrepräsentationen erläutert.

Abschließend evaluiert diese Arbeit das vorgestellte Gras-Rendering-System im Hinblick auf die Darstellungsqualität des Grases und dem benötigten Rechenaufwand. Die Ergebnisse werden unter einigen wesentlichen Aspekten zusammengefasst und verglichen.

Contents

Chapter 1

Introduction

The image synthesis of photo-realistic renderings is a main topic in the field of computer graphics. In particular the video game industry encourages rendering techniques to enhance the photo-realism of images with the constraint of real-time rendering. The increasing hardware performance of today's consumer hardware together with the research done on efficient rendering techniques is capable of running applications in real-time with a high degree of photo-realism. In particular computer games and simulations profit from this development. Further, these capabilities are relevant for other interactive applications, such as geographic visualizations and applications of virtual reality.

For photo-realistic renderings of outdoor scenes as they appear in virtual 3D city and landscape models, vegetation is an important parameter. With regard to that vegetation an essential component is grass. Large terrain surfaces are covered by grass of different types, such as meadows, forests, golf courses or sport stadiums. The reliability of renderings for virtual 3D city models can be significantly enhanced by vegetation, especially by grass. Due to human's wide range of visual experience with natural grass, a high-quality representation is indispensable, since even small discrepancies would be recognized by the viewer, at least subconsciously. These circumstances set out a powerful argument for thorough research on that issue in order to improve today's rendering techniques, which have up to now merely produced poor results with regard to rendering grass in real-time applications.

1.1. Problem Statement

The performance requirements of the grass-rendering methods limit the potential for results with a high visual quality. As a matter of fact, fields of grass consist of thousands of millions of single grass blades, which is an immense geometric complexity that requires a highly specialized GPU-based rendering technique to accomplish real-time performance. Most rendering techniques only focus on a high performance for high frame rates and neglect visual quality of grass. In consequence, the rendered grass often appears to be flat or lack details.

We present a system for rendering grass in real-time while preserving a high visual quality. The system has to support several features to adapt the appearance of natural grass. This involves, for instance, a good parallax effect while moving through grass fields. In addition to that, the correct occlusion of other scene objects by grass blades is important as well. In combination with a plausible illumination model, these

features provide a basis for high visual quality and natural-like behaving grass. Another characteristic of nature scenes is the great variation of its components and the absence of repetitions, which also applies to grass - every single grass blade is unique. Accordingly, it is a major concern to achieve a maximum of individuality by means of a high amount of details to mimic realistic grass. Variations are provided by altering length, shape, color and texture of the virtual grass blades. When observing surfaces covered with grass it is also noticeable that the grass is distributed in a non-uniform manner. Thus, it is crucial to provide a density management to affect and alter the local density of grass.

Aiming at rendering grass in real-time, a level-of-detail scheme is used to preserve details and high visual quality for the viewer. Especially navigation near the ground with its broad range of distances to the grass covered surfaces benefits from that approach. Regarding these varieties in distance, three state-of-the-art grass rendering techniques are applied to different camera-distant zones so that distinctive quality performance rations provide a high-quality rendering system for grass in real-time.

1.2. Contributions

In this thesis we present a framework for enriching virtual 3D-scenes with grass. This framework can be easily integrated into existing rendering engines. Moreover, we elaborate several new approaches and improvements with reference to existing methods.

The framework supports real-time rendering of grass based on state-of-the-art techniques. Additionally, we describe a volume-based rendering approach embedded into a level-of-detail scheme to efficiently cover large portions of a terrain surface with grass of high visual quality.

Besides, an extensible grass-geometry-generation framework is also presented. This component generates proper geometrical representations of grass. In addition, we propose a method to improve the imitation of natural growth of grass increasing the reliability of renderings that employ the generated grass-representations.

The framework also provides interfaces to model and style the grass-layer for specific scenes. This contains a continuously variable grass density, a color modulation map, light maps (e.g. for ambient occlusion) and the support for bumpy terrain surfaces. We suggest a technique for the fine-granular displacement of grass on bumpy surfaces. This technique overcomes the limitation of adding grass to rough terrain surfaces in a coarse and large-scale manner.

1.3. Structure of the Thesis

The remaining part of this thesis is organized as follows: The next chapter provides a survey of existing work on techniques related to the representation of grass in computer graphics. In Chapter 3 our grass-rendering concept is briefly presented. It follows a chapter about the generation of natural-like grass geometry. Chapter 5 will give an in-depth description of the used rendering techniques. The next chapter explains the grass management on terrain surfaces in detail. Finally, Chapter 7 closes with a discussion

about our results including some performance measurements followed by conclusions and proposed future work in Chapter 8.

1.4. Fundamentals and Notations

In this thesis common terms, functions and notations are used from the field of computer graphics and real-time rendering. To clarify them to the unfamiliar reader, we explain the usual terms here.

Vectors and Matrices

Vectors are used in column vector notation. If not defined otherwise, the components of vectors are indexed as follows:

$$\vec{s} = \begin{bmatrix} s_x \\ s_y \end{bmatrix}, \vec{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}, \vec{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \\ u_w \end{bmatrix}$$

Matrices are denoted using upper-case letters, usually supplemented with an index note.

Coordinate Systems

The following coordinate systems are in use:

$WCS \subseteq \mathbb{R}^3$	The *world coordinate system* defines the world space.
$ECS \subseteq \mathbb{R}^3$	The *eye coordinate system* defines the camera space, also denoted as view space or eye space.
$CCS \subseteq \mathbb{R}^3$	The *clip coordinate system* defines the clip space.

Virtual Camera

The image synthesis requires a virtual camera. The virtual camera has a position in world space, a view direction, and an up-orientation. All scene objects are transformed from world space to view space with regard to these parameters. This is a substantial processing step in image synthesis. The view transformation matrix is denoted as C_{view} and defined by $WCS \xrightarrow{C_{view}} ECS$. Moreover, the projection transformation matrix is denoted as $C_{projection}$, which transforms the scene objects from eye space to clip space and is defined by $ECS \xrightarrow{C_{projection}} CCS$. The virtual camera is further called viewer or just camera.

Non-Standard Functions

Equations in this thesis often contain functions such as $\mathsf{clamp}(x, a, b)$, which are non-standard functions. They are not explicitly defined in the place of the equation due to the fact, that they are common in computer graphic applications and image processing. Instead, these functions are cohesively listed in Chapter A found in the appendix.

Chapter 2

Related Work

Numerous research was done on photo-realistic rendering of vegetation in real-time. Especially for virtual reality and computer games it is an important topic. In this chapter we briefly present work related to grass rendering. Every technique is commented with a short evaluation including the main advantages and drawbacks.

Grass in nature scenes is used in a wide range of distances. Thus, at ground level a very detailed representation of the grass blades is necessary for close-up views. In contrast to that, from bird's perspective and other high viewpoints a very fast technique is needed to cover the whole terrain with grass. Hence, multiple approaches are known to render grass but each existing method can only be applied in a context of specific view conditions (Habel, 2009). It follows that most of the methods are approximations to accelerate rendering due to the geometric complexity of grass. There are three fundamental categories of methods: geometry-based, image-based and volume-based rendering methods (Zhao et al., 2009). This classification is done by the model of grass and its computer graphical representation.

2.1. Geometry-based Rendering Methods

The most naive approach is handling grass with a geometry-based rendering method. The geometric representation is a close match to natural grass. Thereby, each grass blade is separately modeled and rendered resulting in a very high degree of realism and reliability. This method allows for an individual and correct illumination of every blade.

There is a multitude of ways how such a grass blade can be composed. Boulanger et al. (2006) use a fixed strip of quads textured with a semi-transparent scanned or hand-drawn image of a grass blade. Perbet & Cani (2001) use a few connected line segments to represent a grass blade colored in various shades of green. Similar to this approach, Guerraz et al. (2003) replace the line segments with a chain of billboards. The billboards are facing the camera and are textured with pieces of the green structure mimicking a grass blade surface.

The required memory to store the geometry for a relatively large terrain is immense. In addition to that, the related high computational effort when processing the huge amount of vertices while rendering is a major drawback of these methods. Today's processing power in consumer hardware is insufficient for rendering many thousands grass blades in real-time (Dietrich et al., 2005). Nevertheless, off-line rendering with geometry-based methods is common for motion pictures. In these contexts, clusters of

computers are utilized to provide the required processing power.

2.2. Image-based Rendering Methods

Generally, the image-based rendering methods are the most common approaches for vegetation due to the complex geometry. These methods utilize triangles or quads textured with a semi-transparent image to replace the complex geometry. That is because with far distances multiple rendering primitives only yield a few pixels in the final image so that the viewer does not recognize replacements with a simpler geometry. For that reason, several grass blades are arranged as groups into a single rendering primitive. This accelerates rendering due to the massively reduced amount of vertices to be processed.

Commonly used image-based techniques are billboards (Maciel & Shirley, 1995). The approaches differ in their orientation of the utilized rendering primitives. They can be statically arranged in parallel slices, crossed as stars or form clusters (Pelzer, 2004) or the rendering primitives are dynamically facing the camera (Whatley, 2005). The billboards are instanced over the terrain surface to fill large areas. This techniques are commonly used in actual software, probably due to their simple implementation and easy integration into existing rendering pipelines. However, the approach is fill-rate limited as the semi-transparency of the rendering primitives leads to a massive overdraw.

Another classical approach is 2D-texturing. A semi-transparent grass texture is mapped onto the existing geometry of the terrain surface (Maillot et al., 1993). The grass texture is blended with the ground texture - if existing - based on its alpha-channel. This approach lacks a parallax effect and the visible flatness appears unpleasant.

Both methods inherit a further weak point. They are limited by the resolution of the textures with regard to details, which is why in close-up views undesired artifacts are noticeable. As a consequence, image-based methods are merely suitable for grass in the distance.

Since the used rendering primitives are simplifications of complex geometry, they need a specialized illumination model to approximate the original illumination. One of these is the Spatial Bidirectional Reflectance Distribution Function (SBRDF), which store the spatial varying reflectance properties of mostly flat surfaces using textures (McAllister et al., 2002). The problem with these is that structured surfaces can be too rough for SBRDFs. In order to support them, Bidirectional Texture Functions (BTF) are used to retrieve the reflectance properties (Müller et al., 2004). Therefore, a set of images with different lighting and view conditions is stored using textures. While rendering, the final fragment color is an interpolation of several images. The limited sampling rate and the required memory to store the image set are the fundamental flaws of the BTFs approach. On account of this, compression schemes are applied to reduce the amount of required memory (Shah et al., 2005) or coarser approximations at lower sampling rates are used to accelerate rendering (Boulanger et al., 2009).

2.3. Volume-based Rendering Methods

A bounding volume full of grass is approximated and stored as volume data. Rendering this volume data produces a vivid and realistic presentation of the grass (Neyret, 1998). Volume rendering in general is computationally expensive. The individual approaches are distinct in their technique of accelerating rendering of volume data (Meyer & Neyret, 1998).

The volume-based grass rendering technique of Bakay et al. (2002) is similar to rendering fur with shells (Kajiya & Kay, 1989). Using this method, the grass blades are extruded from the ground texture. The rendering primitives are processed several times in various heights above the terrain surface with an increasing alpha-test threshold. Thereby, multiple textured disks occur on top of each other representing a grass blade. This approach is mainly used due to its easy technique of animating the grass (Bakay, 2003; Banisch & Wüthrich, 2006), although close-up views make the distinguished disks visible. Moreover, this method is fill-rate-limited because of the small amount of visible fragments per rendering primitive and the high degree of overdraw.

Another group of volume-based approaches utilize vertical slices. A slice aggregates a part of the volume data into a 2D-image. Thereby, each slice approximates a portion of the complex geometry enclosed in the volume (Perbet & Cani, 2001). The applied rendering primitives, such as quads or two adjacent triangles are axis-aligned and parallel. They are textured with the slices' images. While rendering, the orientation of the slices needs to be adjusted according to camera movements. Hence, the slices are arranged into a grid structure to avoid the view management. This makes any switching or blending depend on the camera obsolete (Boulanger et al., 2006).

Instead of forward rendering of the slices, ray tracing can be performed. By utilizing the fragment shader, the performance is significantly improved (Habel et al., 2007). This method allows for direct application to the geometry of the terrain surface so that no additional rendering primitives are necessary.

Vertical Slices match the direction of growth of the grass blades better in contrast to shells. This results in a more detailed representation of grass. However, it is limited by volume data resolution and makes magnification artifacts visible at low distances.

2.4. Level-of-Detail Methods

Most of the mentioned grass rendering methods are combined into level-of-detail schemes. By using different grass representations at different distances rather than using only one kind of representation, the grass rendering quality is optimized. Thus, mixing multiple representations yields to high-quality renderings of grass. All of these approaches generally use a geometry-based rendering method for close-up views. Being computationally expensive, this method is only applied to a small range around the camera and yields the best rendering results. For grass faraway from the camera, an efficient image-based rendering method is applied. The only difference lies in the mid-level rendering method for grass at moderate distances. Here, Guerraz et al. (2003) use parallel vertical slices, but this method requires special care regarding the view management of the camera.

This is one basic reason why Boulanger (2008) employs vertical slices in a grid so that the view management of the camera will be obsolete.

Chapter 3

Grass Rendering Concept

Our goal is to cover large terrain surfaces with grass in real-time. These grass fields consist of thousands of millions of grass blades. Storing the appearance of each individual blade as geometric representation requires lots of memory space. Thus, we only store a small rectangular piece of terrain surface filled with grass. This *grass patch* totally fits into the graphics memory and we render multiple instances of this patch onto the cells of a uniform grid to fill the whole terrain surface. Yet repeating the same patch over large areas produces undesired visible patterns. To break the visual repetitive structures, we apply an approach to use multiple versions of this base patch to introduce an aperiodic tiling scheme.

3.1. Grass Representations

For the purpose of achieving real-time performance, the computational effort for rendering is as relevant as the memory footprint. For large terrains a considerable number of patch instances needs to be rendered, which requires sophisticated rendering techniques to achieve real-time performance. We use three representations of grass (Figure 3.1) as

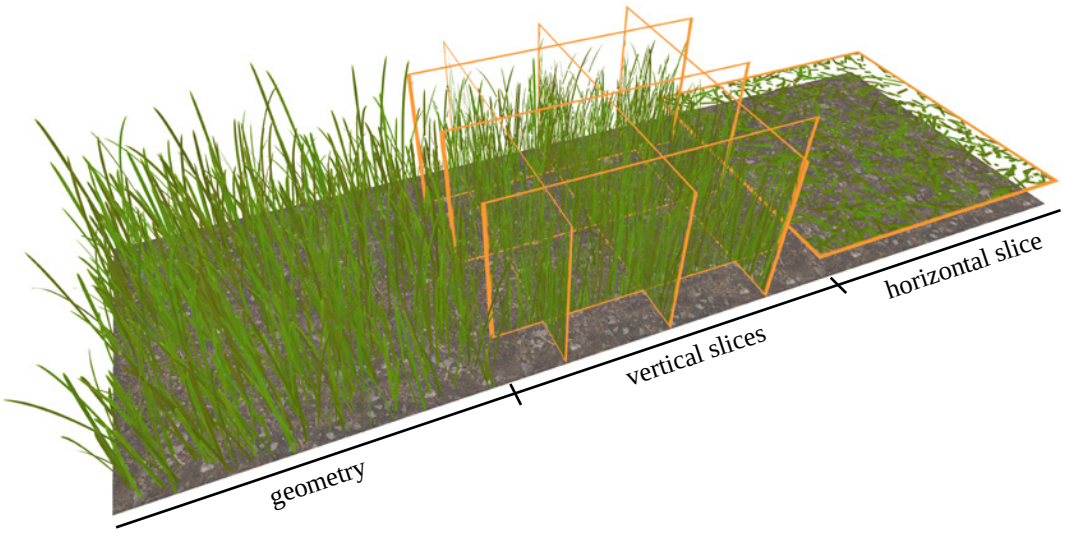

Figure 3.1.: *The three representations of grass used by our rendering method. The geometry-based patch consists of 24,000 triangles (800 grass blades, each 30 triangles).*

they differ in their visual quality and the required computational effort for a respective rendering method. Each representation depicts the same base patch of grass due to the consistency of renderings. We utilize a specialized rendering technique for each representation to draw the patch instances and finally fit the performance requirements.

Geometry-based Grass Representation

The base representation for grass is geometry. Therefore, each grass blade is modeled as a textured semi-transparent triangle strip. The correct and pixel-precise shape of the grass blade is obtained from the alpha mask, which allows for arbitrary formed edge structures of the grass blades. The grass blades themselves are uniformly distributed within the patch. Their root is always inside the rectangular piece of ground but they may reach outside with their tips, which helps to hide the edges between neighboring instances of the patch.

Geometry-based grass is the most accurate representation of the grass blades. Each blade is separately represented and obtains an almost smooth bending, depending on the number of subdivisions. Due to the suitable shape, the normals of the geometry surface provide the parameters of a plausible illumination. The grass blades are rich of details, such as fibers through 2D-texturing. Moving through geometry-based grass results in a correct parallax effect through reliable blade-blade occlusions. Further, the tips of the grass blades form the characteristic silhouette of a grass surface.

The amount of vertices required for the geometry of the grass blades depends on their count within the patch and the number of their subdivisions. Nonetheless, the whole patch totally fits into memory of today's graphics hardware.

Volume-based Grass Representation

As a second and more abstract representation of grass, we use a volume-based grass representation with multiple vertical axis-aligned slices. Every slice contains a portion of grass blades. These grass blades are stored in a semi-transparent texture mapped onto the slice. Thereby, the grass blades are fully opaque and the space between them is transparent. In contrast to Boulanger (2008), we do not distinct between the front and back face of a slice and apply the same texture, which is a simplification. However, this simplification is hardly visible in particular in the distance due to the homogeneous structure of grass.

The volume-based grass representation significantly reduces the geometric complexity of the grass. The geometry of several grass blades is consolidated into a textured, semi-transparent slice. Moving through volume-based grass results in a reliable but not correct parallax effect, which appears merely between the blades of separate slices rather than blades of the same slice. Additionally, due to the semi-transparency of the slices, the tips of the grass blades form the characteristic silhouette of a grass surface as the geometry-based representation does.

Since the slices are a flat structure, a specialized lighting model is required for a plausible illumination. Furthermore, the grid structure of the slices becomes noticeable with certain view angles. Thus, the number of slices per direction is adjustable to hide

the grid structure on the backdrop of an overhead of computational effort and required memory.

Image-based Grass Representation

For the third representation we use an image-based grass representation. This merely is a single horizontal slice short of the ground. A semi-transparent texture contains all grass blades of the patch shown from above and is mapped onto the horizontal slice.

This representation reduces the geometric complexity of grass to the largest extent. It aggregates the geometry of grass blades of a whole patch into one slice. Due to the usage of a mere horizontal slice, the represented grass shows a flat appearance. There is no parallax effect while moving through image-based grass. Similarly to the volume-based grass, the slice of the image-based grass representation has a flat structure as well, which requires a specialized lighting model to provide a plausible illumination.

3.2. Levels of Detail

In fact, we want to obtain a mostly accurate and reliable representation of grass while preserving real-time frame rates. Thus, we combine the aforementioned representations of grass into a level-of-detail scheme (Clark, 1976).

Since the overall appearance of geometry-based grass has the highest visual quality, this representation is utilized near the camera. It leads to reliable renderings of grass due to the high degree of details, the plausible illumination and the correct parallax effect. Due to the high geometrical complexity of geometry-based grass, the computational effort of a respective rendering method limits the application to a low range around the camera. Furthermore, each grass blade occupies only few pixels when rendered in the distance, which causes aliasing artifacts and reduced efficiency.

Hence, we employ the volume-based grass representation for moderate distances. This representation significantly reduces the geometric complexity. Thus, the application on a grand scale is still efficient compared to geometry-based grass. This representation provides a high visual quality just like the geometry-based ones do. The parallax effect is in particular reliable for moderate distances since there is no noticeable difference to geometry-based grass. Due to the fact that the grid structure of the vertical slices becomes visible with certain view angles, it is advisable to use multiple volume-based representations of grass with a decreasing number of vertical slices.

Since the parallax effect of grass in far distance is not even perceptible, we apply the more abstract image-based grass representation to that region. This representation further reduces the computational effort of rendering grass. In the far distance a huge amount of grass patch instances is required. Thus, we use the grass representation with the most efficient rendering technique.

We use different grass representations rather than only one kind of representation to optimize the grass rendering quality with regard to efficiency. This procedure results in a trade-off between visual quality of the rendering by using accurate representations of grass and performance by lowering the geometric complexity through abstract representations.

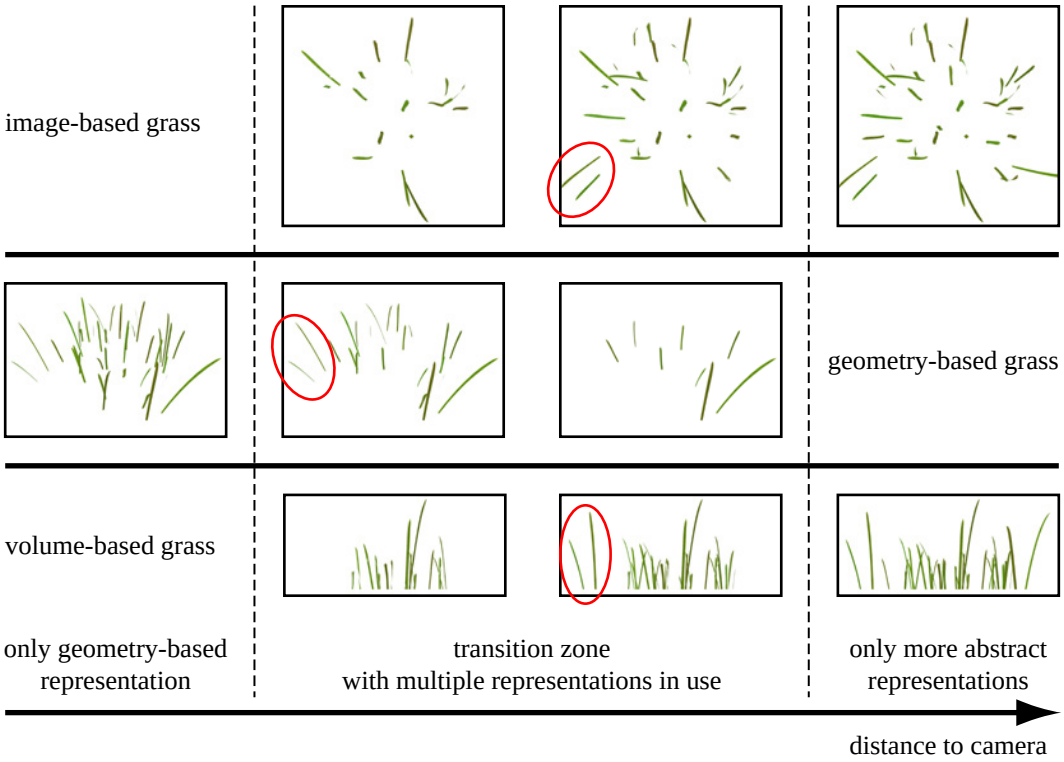

Figure 3.2.: *The grass blades move successively from one representation to another within the transition zone. Thus, the transition is seamless due to the simultaneous rendering of patch instances of both representations while transitioning.*

Thus, mixing multiple representations yields to high-quality renderings of grass while preserving real-time frame rates.

Nevertheless, transition zones still remain a substantial point with level-of-detail concepts since almost invisible and seamless transitions between the different levels of detail are very important for rendering a homogeneous grass structure. A linear blending between two representations of the grass patch would cause whole patch instances to be semi-transparent, which would look unpleasant. Instead we render the two representations of the grass patch simultaneously in place. When transitioning the grass patch between two representations, we move single grass blades successively from one representation to the other (Figure 3.2). This results in always entirely opaque grass blades, each rendered by only one representation. In avoidance of popping artifacts when a grass blade appears in the new representation, the opacity is additionally adjusted to smooth this crossover.

3.3. Density Management

Natural grass is distributed on a terrain surface in a non-uniform manner. Hence, it is desired to scale the density of a grass patch. We use a threshold-based approach as density management, which provides the possibility to continuously scale the density of a grass patch (Boulanger et al., 2006). Therefore, each grass blade within the patch

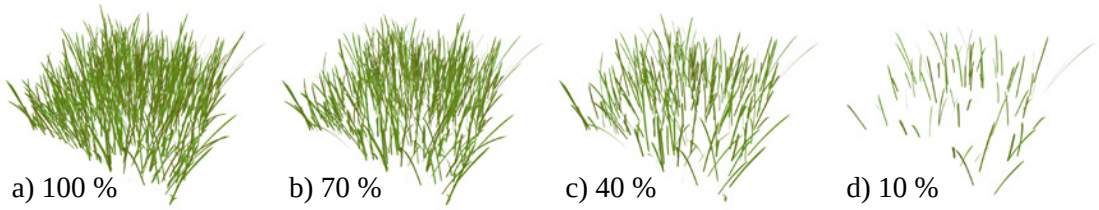

Figure 3.3.: *A geometry-based patch rendered with decreasing density from left to right. Due to the threshold-based approach, the density of a grass patch is adapted while rendering with almost no computational overhead.*

is a density threshold assigned. The threshold is in the bounded interval $(0, 1]$. While rendering, the grass blades with a threshold value larger than the desired local grass density are discarded. This provides the potential for a continuous thinning of the grass patch with almost no computational overhead at run-time (Figure 3.3). Thus, only one patch with the highest density is required rather than several patches with various grass densities. In this way, we can adapt the amount of rendered grass blades within a patch to the desired local grass density. This enables customized distributions of grass on the terrain surface. An artist, for instance, could design a distribution regarding the surrounding scenery to perfectly fit the grass and so achieve the desired result. This is explained in detail in Section 6.4.1.

Chapter 4

Grass Patch Generation

While rendering, we use a single in-memory grass patch and repeat this patch over the whole terrain surface. The base for grass representation is geometry. Hence, we initially generate a geometry-based patch for the desired grass type. The other representations are created with the base patch as its primary element.

For the generation of the geometry-based grass patch, various methods are known. Here, procedural approaches are the most common ones, which generate the geometry of grass with regard to user-defined parameters. Hand-made modeled patches by artists are as well quite common. However, the more exact methods are biological models considering environmental factors to generate the grass geometry.

Our grass-rendering framework provides a procedural algorithm to generate grass. The generation component is extensible to easily add new methods. The provided method provides grass patches for the most common sceneries like parks, football stadiums or meadows (Figure 4.1). With this framework, various grass parameters of the generated

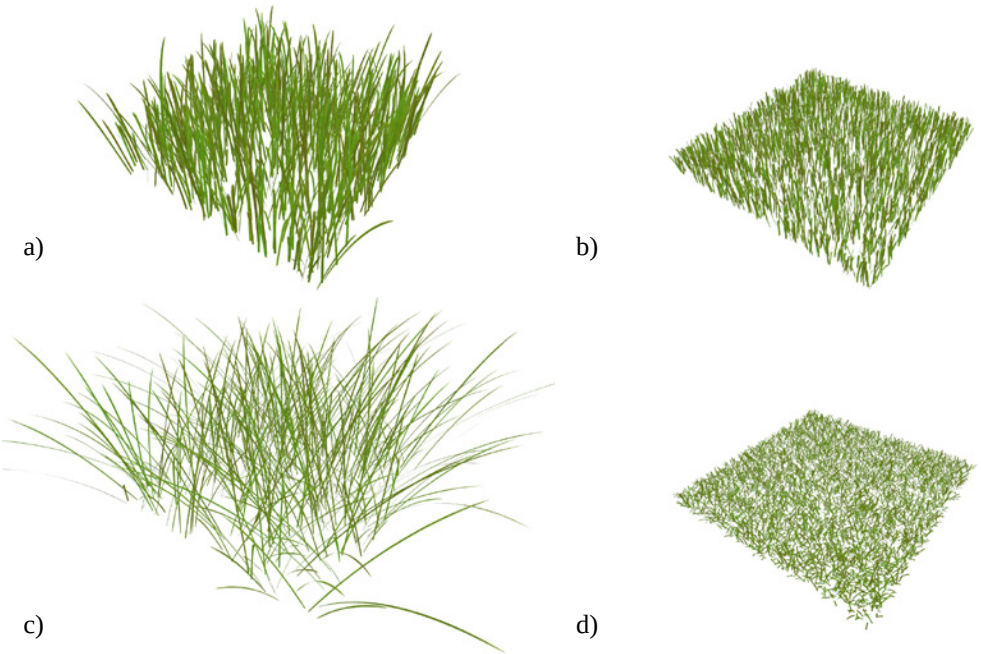

Figure 4.1.: *Generated grass patches with medium long grass (a), short-cut grass (b), long thin meadow grass (c) and utility lawn (d).*

patch are affected, e.g. the overall blade count, length range, tilt range and width range.

4.1. Geometry-based Patch

The geometry-based patch is bounded by a unit square of ground. We generate several individual grass blades onto this piece of ground. First, we select an appropriate root position. Secondly, we generate the geometry defining the shape of the grass blade. Thirdly, the appearance of the grass blade is modified by mimicking natural uniqueness.

The blade root position is randomly selected within the ground unit square. It is important to distribute the grass blades as uniform as possible. Otherwise, undesired visual structures appear when tiling the patch over the terrain surface. We use a random number generator to determine the position of a grass blade.

Stratified sampling is proposed by Boulanger (2008) as an improvement against the backdrop of theoretically infinite samples being necessary to achieve uniformity with a random number generator. Thereby, the patch is further divided into a fine uniform grid. Then a single grass blade is randomly placed in each of the grid cells. This results in almost isolated grass blades within the patch. Especially low dense patches appear unnatural in this way. Consequently, we still use a random number generator and do not apply stratified sampling. The implementation of the random number generator achieves a sufficient uniformity so that no visual structures are noticeable. Additionally, the bending and rotation of the grass blades mask these visual structures anyway.

4.1.1. Shape of the Grass Blade

In nature real grass blades are very thin. We represent them by a triangle strip with a two-sided material of zero thickness. Different parameters are used to define the range of variation such as length and tilt. At the root position of the grass blade a trajectory instance is calculated as follows:

$$\vec{p} = \vec{b} \cdot t + \vec{g} \cdot t^2, \ (0 \leq t) \tag{4.1}$$

The constant $\vec{b}$ is affected by the desired length, tilt and rotation of the grass blade whereas $\vec{g}$ describes gravity by default. This trajectory gives the grass blade its characteristic bending (4.1).

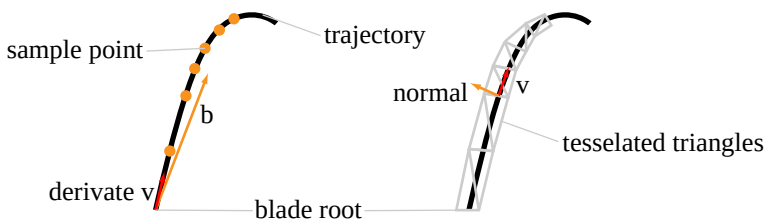

Figure 4.2.: *The tessellation of a grass blade. The triangles are created along with the sample points of the trajectory, which is affected by b according to the desired length, tilt, and rotation of the grass blade.*

A blade is subdivided into a quadrilateral strip with a fixed, pre-defined number of subdivisions (Figure 4.2). The overall amount of vertices of a patch can be estimated by the number of subdivisions with regard to the number of grass blades within the patch. This estimation can be utilized to optimize the patch referring to the usage of memory objects, such as vertex buffer objects. Each blade is tessellated with a fixed count of triangles due to sampling the trajectory by discrete steps of t in Equation (4.1).

The bending defined by the trajectory continuously increases. To preserve a steady angle between the tessellated triangles, their size needs to decrease towards the tip of a blade. Sampling in equal-size steps of t results automatically in smaller triangles at the more bended part due to the decreasing derivate $\vec{v}$ (4.2) of the trajectory (4.1).

$$\vec{v} = \vec{p}' = \vec{b} + \vec{g} \cdot 2t \tag{4.2}$$

The obtained triangle strip representing the grass blade is textured with a green structure, e.g. from the scanned image of a natural grass blade. The alpha mask of this texture determines the final shape. Finally, the vertex normals are calculated to be always orthogonal to $\vec{v}$.

4.1.2. Variation of the Appearance

The appearance of the grass varies in several aspects. Initially, the shape is modified according to the bending, tilt and rotation of a grass blade. Alternative alpha masks further increase this kind of variation and result in slightly different blade shapes.

Nonetheless, grass has no uniform surface. Various images for the green structure can be utilized or a procedural material, which generates an almost unique appearance.

Grass blades within a patch differ in their age and levels of degradation as well. We slightly modify the color of each blade in two ways to achieve such a variation. For every blade a color modulation is applied, by which they differ among themselves. Furthermore, a light color modulation is applied to every tessellated vertex and interpolated in between to raise the quality of imitating nature's variance.

4.1.3. Density Threshold Distribution

After having generated the geometry of the grass blades, we prepare them for the usage with the rendering component of the framework. The distribution-based density management of the grass discards a portion of blades according to the desired local density. The decision which blades are discarded is based on a threshold per blade. Each grass blade is a density threshold within the interval of $(0, 1]$ assigned. All grass blades of a patch with a density threshold above the local density are discarded.

The random and more or less uniform distribution of the density thresholds is important. There are two kinds of distribution that have to be cared about: the spatial distribution within the patch and the distribution of the values within the interval. Inappropriate spatial distributions of the density threshold cause visual structures. This imbalance introduces undesired visual patterns appearing when tiling a half-filled patch over a large area. The distribution of the values within the interval of $(0, 1]$ influences density scaling. A uniform distribution is mandatory to provide an almost

linear dependency between the desired grass density and the amount of rendered grass blades.

We present a naive approach of randomly distributing the density thresholds and introduce a more advanced one to improve the visual quality in a more natural-like manner.

Uniform Random Distribution

The naive approach is a simple random distribution of the density thresholds inside the patch. Thus, the relatively likelihood of a grass blade to take on a given density threshold is equal within the interval $(0, 1]$. A proper probability density function for a grass blade's density threshold is defined as follows:

$$f_{naive}(x) = \begin{cases} 1 & \text{if } x \in (0, 1] \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Due to the uniform distribution of grass blades inside the patch, the density thresholds are also homogeneously distributed over the patch. The thresholds in turn are uniformly distributed within its interval as well depending on the uniformity of the random number generator (Figure 4.5.a). This method causes a very consistent thinning, so that a low dense patch contains just a few isolated grass blades. The appearance of these patch instances seems unnatural. Especially long transitions from low to high density of grass or larger areas with a low density benefit from a more natural-like growth of grass.

As a matter of fact, in nature new grass blades grow first near already existing ones. Thus, to improve the appearance of that low dense patches in a more natural-like manner, we want to slightly cluster the grass blades (Figure 4.3).

Clustered Distribution

The proposed method does not change the geometry of the patch. The spatial distribution of density thresholds is merely modified, which only affects the visibility of grass blades at arbitrary density scales. A uniform distribution within the interval of $(0, 1]$ preserves almost the same amount of grass blades at any local density (Figure 4.5). This approach does only change the appearance of grass growth.

Figure 4.3.: *A generated grass patch at low density (10%) without clustering applied (a), with slightly clustering (sigma = 0.12) applied (b) and strong clustering (sigma = 0.0) applied (c).*

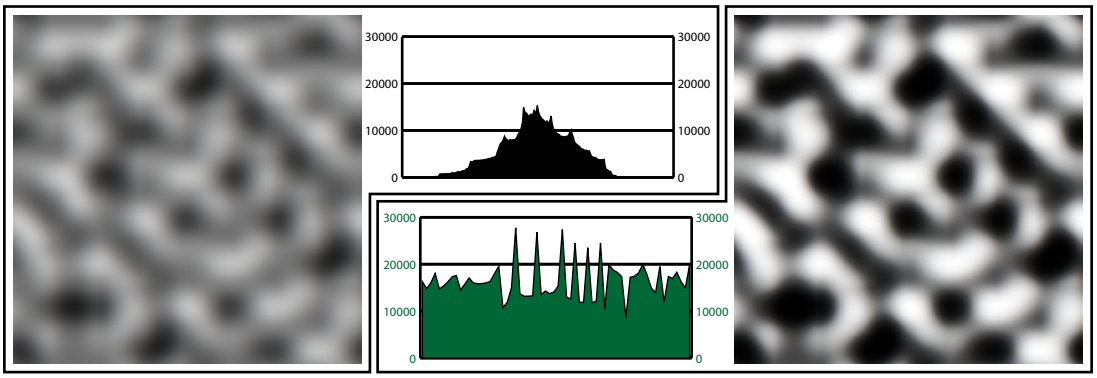

Figure 4.4.: *The 2D Perlin noise map applied to distribute the density threshold over the patch. The original image (left) heaps values around the mean (top histogram). The histogram equalize filter spreads the values (bottom histogram) trying to align their frequency with the result of the right image.*

We imitate this kind of growth by customizing the distribution of density threshold values. A low frequency two dimensional Perlin noise map (Perlin, 2002) is utilized to determine the density threshold of a grass blade. A clustering effect applied to the grass blades is noticeable. Hence, this method results in a patch with all grass blades discarded at low densities, such as 10% due to the values of the noise map, which form a normal distribution around its mean of 0.5 (Figure 4.4).

In avoidance of this behavior, we apply a level filter by rescaling the noise values to use the full interval $(0, 1]$. Indeed, this leads to a clustered patch but with a lower amount of blades below densities of 0.5 and a larger amount at densities above compared to the random distribution approach explained in Section 4.1.3. On the contrary, the threshold distribution within its interval is not uniform. Thus, we apply a histogram-equalize filter to the noise map (Jain, 1989) instead of the level filter. The histogram-equalize filter transforms the noise map to use the full range of available values, while trying to align their frequencies to be equal (Figure 4.4).

With regard to our procedural generation approach we want to automatically support clustering up to the desired results. Therefore, a parameter is desired to control the strength of clustering. However, different approaches to interpolate between the value of the noise map and a random value negatively affects the interval distribution of

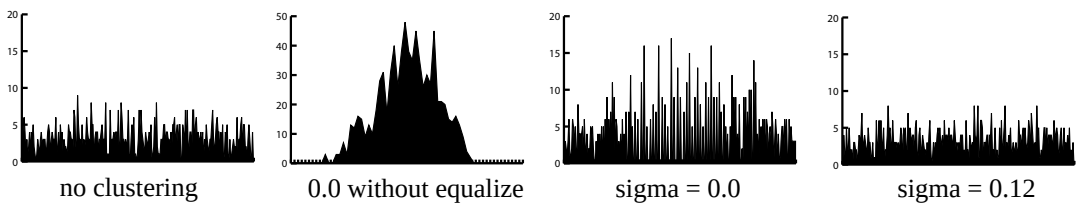

Figure 4.5.: *The blade density threshold distributions with different approaches and parameters. A random uniform distribution with no clustering (a), using only the noise map (b), with the equalize filter previously applied (c) and finally using the Gaussian distribution over the noise map (d).*

the density threshold towards a non-uniform distribution. For that purpose, we utilize a Gaussian distribution in order to determine the density threshold of a blade (4.4). The μ of a Gaussian distribution is retrieved from the equalized noise map at the root position of the blade (4.5). Due to an interval of $(0, 1]$ for the density threshold, the Gaussian distribution is limited to this interval resulting in a probability density function $f_{cluster}(x; \mu_{blade}, \sigma)$ as follows:

$$f_{Gaussian}(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{4.4}$$

$$\mu_{blade} = \mathsf{noise2D}_{Perlin}(blade_x, blade_y) \tag{4.5}$$

$$f_{cluster}(x; \mu_{blade}, \sigma) = \begin{cases} f_{Gaussian}(x; \mu_{blade}, \sigma) & \text{if } x \in (0, 1] \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

The scale σ of the Gaussian distribution was previously specified for this patch. It controls the falloff of the bell curve and thereby affects the importance of the noise map sample to the density threshold. Lower values of σ increase the influence of the noise map. Thus, the scale controls the strength of the grass clustering within a patch (Figure 4.3.b, c). High values of σ lead to similar results like the uniform random distribution of density thresholds as explained in Section 4.1.3.

This method yields to a more natural-like grass patch at low density with grass initially growing in clusters (Figure 4.3.b). As it is merely the spatial distribution that changed, this method does not affect the uniformity of the values within their interval. Neither the equalized noise map nor the Gaussian distribution built upon the noise map negatively affect the overall uniform distribution of the density thresholds (Figure 4.5).

4.2. Patch Conversion

The level-of-detail scheme of our grass-rendering framework uses three different representations of grass. The grass patch of the base representation is generated as described in Section 4.1. In a further preprocessing step, the patches of the other two representations are created.

Vertical and horizontal slices aggregate the geometry of the base patch. A consistency among patch representations is desired to mask the transition zones between the application of two representations. Each slice needs to store the color, normals and density threshold of the aggregated geometry. In order to approximately achieve the same appearance in every representation, the image-based and volume-based patches are

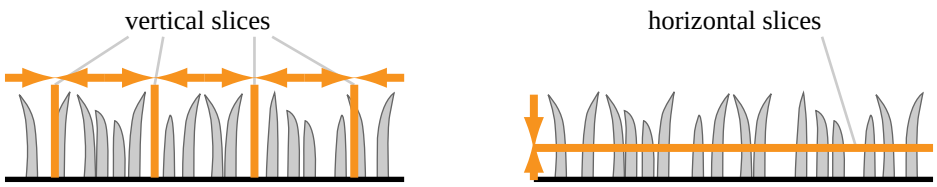

Figure 4.6.: *The vertical and horizontal slices aggregate the grass blades of the geometry-based representation to achieve the same appearance in every representation.*

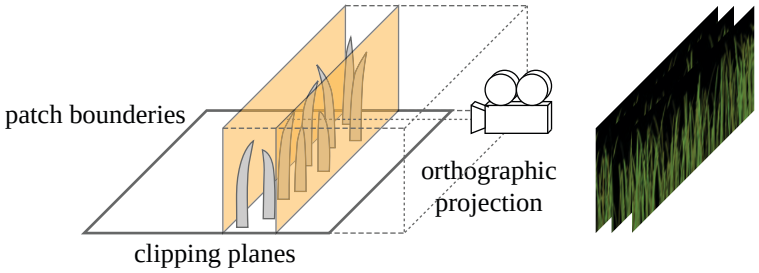

Figure 4.7.: *The generation of the vertical slices. The camera in front of the geometry patch uses an orthographic projection. Only the grass blades of the current slice are rendered due to the two clipping planes. The generated slices are arranged as texture arrays.*

conversions of the geometry-based patch. This requires the rendering of the geometry-based patch in a special setting (Figure 4.6). Moreover, we use render-to-texture to create the semi-transparent slices.

4.2.1. Render-to-Texture

The single horizontal slice contains all grass blades of the patch in an appropriate resolution. The patch is rendered with an orthographic camera from above. This slice is stored as one 2D-texture.

In contrast to that, an axis-aligned slice only contains a portion of grass blades. A section of the grass patch is defined by two parallel clipping planes. This section is rendered also with an orthographic camera along the direction of the slice's orientation (Figure 4.7). This process is done for each slice in the two directions. The resulting data is stored as a 2D-texture array instead of packing it into a single 2D-texture to simplify later access and filtering.

Additionally, we also store a normal map and a blade's density threshold map for every slice to perform dynamic lighting and continuous thinning of these patch representations while rendering.

4.2.2. Mip Maps

In avoidance of aliasing artifacts while rendering due to texture sampling, trilinear filtering is applied to the slices by using mip maps. Mip maps are prefiltered versions of a texture, which compress many pixels of the texture to a small place. When sampling the texture mip mapping supplements a third interpolation between two prefiltered versions (Williams, 1983).

Here, an alternative approach for the mip-map generation is necessary as the OpenGL box filter does not properly work with the semi-transparency of the slices (Figure 4.8). Be C_0, C_1, C_2, C_3 the color of the four texels to apply the filtering and A_0, A_1, A_2, A_3 their corresponding alpha values. It is then that the box filter calculates the texels of the next mipmap level by means of the color and opacity of the previous

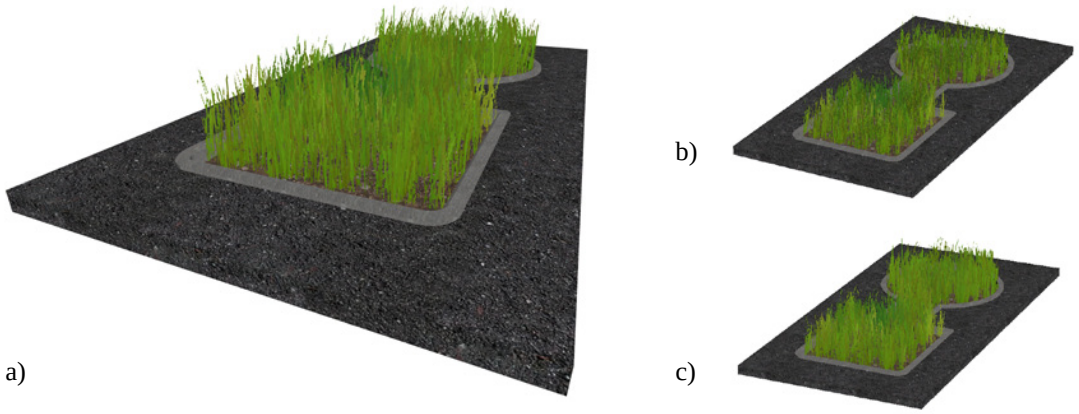

Figure 4.8.: *The vertical slices (a) partially disappear in the distance (b) when generated with the OpenGL box filter. The alpha-to-coverage approach preserves the visibility (c).*

level (4.7) (4.8).

$$box_{color} = \frac{C_0 + C_1 + C_2 + C_3}{4} \tag{4.7}$$

$$box_{alpha} = \frac{A_0 + A_1 + A_2 + A_3}{4} \tag{4.8}$$

Grass in reduced resolution levels seems to disappear. Thus, we need to handle the generation of mipmap pyramids on our own. On the one hand, we calculate the color channel weighted with their alpha values as follows:

$$s_a = \begin{cases} 1 & \text{if } A_0 + A_1 + A_2 + A_3 = 0 \\ A_0 + A_1 + A_2 + A_3 & \text{otherwise} \end{cases} \tag{4.9}$$

$$filter_{color} = \frac{A_0 \cdot C_0 + A_1 \cdot C_1 + A_2 \cdot C_2 + A_3 \cdot C_3}{s_a} \tag{4.10}$$

This leads to a more correct color while merging texels with different alpha values, in particular at the edges of grass blades.

On the other hand, we use an alpha-to-coverage approach to calculate the alpha channel (Castaño, 2010). Therefore, we perform an alpha-test to the original mipmap level of the texture with a high alpha reference value a_r and determine the amount of opaque texels out of the N overall texels, which succeeded in the alpha-test. The coverage is defined as:

$$coverage = \frac{\sum_{i}^{N} (a_i > a_r)}{N} \tag{4.11}$$

For the mipmap levels with a reduced resolution we want to ensure the same amount of opaque texels. First, we generate the alpha values for the next mipmap level taking the maximum value of the four texels from the previous level (4.12).

$$max_{alpha} = \mathsf{max}\,(A_0, A_1, A_2, A_3) \tag{4.12}$$

This guarantees opaque texels remaining visible (Boulanger, 2008). Depending on the ratio of opaque and transparent texels the approach lacks due to mipmap levels appear a little over or less opaque compared to the base level. The comparison is done by calculating the coverage of the new generated mipmap level as well. In the next step, we calculate an alpha rescale factor for this level. Utilizing a histogram of alpha values, we determine a new alpha-test reference value a'_r to obtain nearly the same ratio of opaque and transparent texels (4.15).

$$coverage' = \frac{\sum_i^{N'} (scale \cdot a'_i > a_r)}{N'} = \frac{\sum_i^{N'} (a'_i > a'_r)}{N'} \tag{4.13}$$

$$scale_{coverage} = \frac{a_r}{a'_r} \tag{4.14}$$

$$filter_{alpha} = scale_{coverage} \cdot \max(A_0, A_1, A_2, A_3) \tag{4.15}$$

Then every texel's alpha channel is corrected by the rescaling factor $scale_{coverage}$ (Castaño, 2010).

Now this mipmap level yields similar results performing the initial alpha-test like the base level. We iterate this process for each mipmap reduction step always comparing with the base level coverage.

4.2.3. Color Bleeding

Using the generated semi-transparent textures for slices with alpha-blending produces small undesired silhouettes around the opaque grass blades. Within a volume-based patch, this problem is observed as a gray veil.

Sampling the texture of a slice with bilinear filtering causes an interpolation of neighbored texels (Heckbert, 1986). With alpha transitions, in particular with hard blade edges, the texels differ significantly in their alpha values. This difference in alpha values is averaged to a partial opaque sample. Thus, the texels outside the blade are

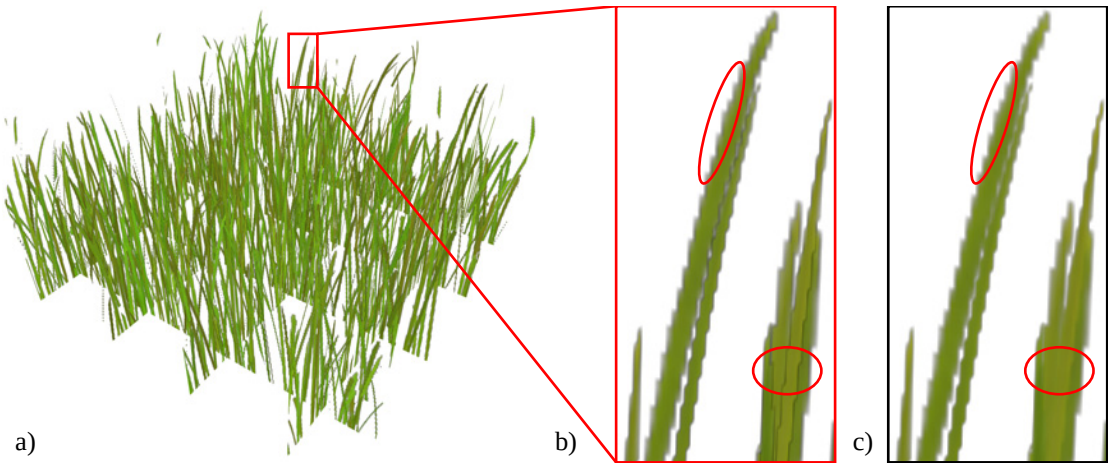

Figure 4.9.: *A volume-based patch (a) has undesired silhouettes around the grass blades (b). On account of our color bleeding approach, these silhouettes are hided (c).*

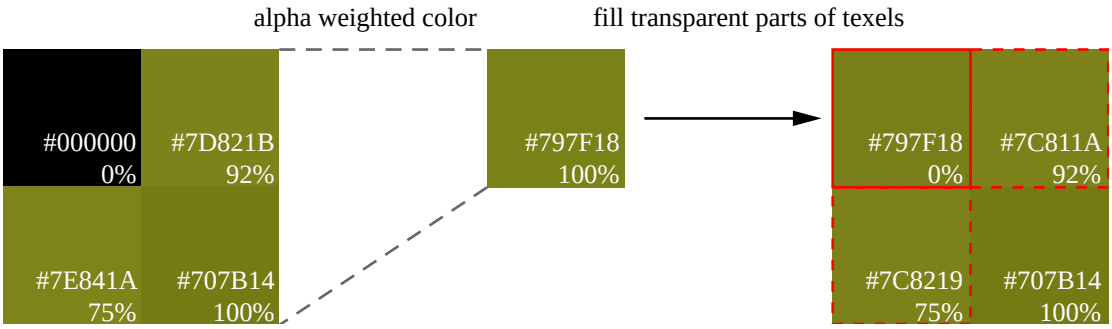

Figure 4.10.: *Our color bleeding approach fills the transparent parts of a texel (full transparency, partial transparency) with the average color from the surrounding opaque texels by sampling the next mipmap level in a full automatic manner. The color of a texel is given in hexadecimal values (RGB) and the opacity in percent.*

taken into account to determine the final color of the sample, when filtering at the edge of a blade. Hence, this color is partially blended with the background color of the slice This happens along the alpha mask edge of each blade and results in the mentioned silhouettes (Figure 4.9).

Enabling alpha-testing with a high alpha-test reference value solves the problem of silhouette artifacts. Due to the necessarily strong alpha-test, aliasing artifacts are introduced. They can be masked by using full-screen anti-aliasing with an increased computational overhead.

The same artifacts are observed with auto-generated images, such as scans of the green structure of the grass blade. The added alpha mask is insufficient in this respect and induces the undesired silhouettes as well. A manual modification of the image is necessary to apply a color bleeding along the alpha transition (Figure 4.11). This method hides the silhouettes and is the common solution for this kind of artifacts, although it is still tedious and impractical for a fully automated conversion pipeline.

We propose an alternative color bleeding approach by using the already generated mipmap levels. After having built the mipmap pyramid, we use an additional pass to fill

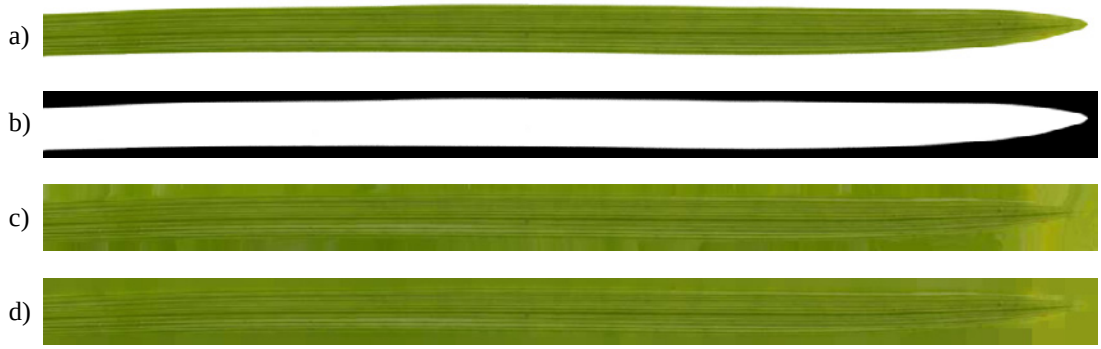

Figure 4.11.: *To avoid silhouette artifacts when alpha blending is enabled an artist usually applies a manual color bleeding (c) to the original image (a) according to its alpha mask (b). Our approach applies the color bleeding fully automated with a result image as shown in (d), which does not significantly differ from the manual modified one.*

the partial and fully transparent texels with a proper color. This method is based on an alpha-weighted mipmap filter, which excludes the OpenGL box filter. In Section 4.2.2 we presented alternative filters supporting alpha-weighting. The transparent amount of a texel in mipmap level i is filled with the mixed color of the neighbored, mostly opaque texels. Therefore, we sample the next mipmap level $i + 1$ to retrieve the mixed color (Figure 4.10).

$$C'_i = A_i \cdot C_i + (1 - A_i) \cdot C_{i+1} \tag{4.16}$$

In this case, we apply nearest filtering to sample the next mipmap level. This results in a spatially varying background color, which refines near blade edges. Hence, a sampling with bilinear interpolation across the blade edge is applied to four texels of a similar color, which leads to a slightly modified but almost correct color at the edges and avoids the undesired silhouettes.

Employing that method, best results are obtained by previously building the full mipmap pyramid and afterwards process the mipmap levels sequentially beginning with the highest.

Chapter 5

Rendering Techniques

Three representations of grass are utilized for the rendering concept. For each representation, a patch is either generated or converted from the geometry-based representation as described in Chapter 4. In this chapter, we describe the *renderers*, which implement rendering techniques to draw instances of those grass patches. A renderer supports exactly one representation of grass and performs three stages while rendering a frame:

1. **Prepare Pass.** In this stage the renderer setups its shader programs, binds required textures, and updates the probably modified camera configuration. The parameters that affect the rendering results are passed to the shader program here.

2. **Draw Pass.** In this stage all patch instances of the representation this renderer supports are drawn. This is usually the most time-consuming stage the renderer runs through.

3. **Finalize Pass.** In this stage the renderer cleans up, unbinds textures and terminates shader program usage.

Since we want to overcome the geometrical complexity of grass, we use abstract representations, such as the volume-based or image-based grass. The utilized rendering techniques with the respective renderers differ in their implementation and the required computational effort to draw a patch instance. Furthermore, the visual quality of the rendering results are very different between the rendering techniques. Due to the requirement of real-time performance, it is desired to approach a maximum of efficiency while rendering. Hence, we use state-of-the-art rendering techniques, which efficiently use the available hardware capabilities. All rendering techniques, we present in the remaining chapter, are highly specialized, GPU-based, and only limited through the graphics hardware.

5.1. Geometry-based Renderer

The geometry-based renderer usually performs regular forward rendering of the grass blade geometry (Akenine-Möller et al., 2008). That is to say, the renderer transforms the vertices of the geometry, which are stored in the patch and further processes the rasterized fragments with texturing and shading.

The grass blades are represented by geometry as textured quadrilateral stripes. These stripes are tessellated with triangles while the patch was generated. Quads are

Figure 5.1.: *An arbitrarily shaped beet filled with geometry-based grass.*

not used because they would be divided into triangles anyway by the graphics driver. The vertices of the triangles are in local patch coordinate system and stored in the graphics memory as a vertex buffer object. When rendering a patch instance of the geometry-based grass representation, a basic vertex shader is employed to transform each vertex of this vertex buffer object according to the camera's view and projection transformation C_{view}, $C_{projection}$ and the patch instance transformation $P_{instance}$ (5.1).

$$\vec{v}' = C_{projection} \cdot C_{view} \cdot P_{instance} \cdot \vec{v} \tag{5.1}$$

Afterwards, in fragment shader stage the blade fragments are discarded with regard to their density threshold and the local grass density. Furthermore, the fragments outside the opaque part of the blade's texture are discarded by an alpha-test with a medium threshold to shape the grass blade. The remaining fragments belong to the blade and will be textured with the color map and get a per-pixel illumination applied according to the utilized lighting model. Due to the relaxed alpha-test, the edges of the blades are anti-aliased and blended, which requires a sorted rendering order from back to front. Since it is impossible to efficiently sort the grass blade geometry in that manner, Boulanger (2008) propose the application of full-screen anti-aliasing by multisampling. We follow this approach and activate the hardware-accelerated alpha to coverage (NVIDIA Corporation, 2004), which converts the alpha value of a fragment to an appropriate sample mask. This method solves blending issues due to not noticeable artifacts and provides an order-independent rendering of grass blades with smooth edges.

5.2. Volume-based Renderer

We use a volume-rendering technique with semi-transparent textures for the vertical axis-aligned slices. One approach is to render each slice as geometry with the semi-transparent texture mapped to it. Each grass blade within the texture is fully opaque and the space between the blades is transparent. This way, an alpha-test with a high

reference value is applied to discard the transparent parts of a slice, which causes aliasing at the edges of the rendered grass blades. A lower alpha-test reference value reduces the appearance of aliasing artifacts due to the partial opacity, which arises within a slice at the edges of grass blades, especially when using a filtered version of the texture, such as a reduced mip-map level, sampling with a bilinear filter, or when the slices were generated with an anti-aliased renderer. Thus, alpha-blending is applied to the semi-transparent slices, which requires a correct rendering order. Usually, a back-to-front rendering order is employed to provide a correct blending result of the partial opaque fragments. Therefore, a sorting of the heavily overlapping slices is necessary, which cannot be efficiently realized. Furthermore, for intersecting rendering primitives, which apparently exist in an axis-aligned grid structure, there is no correct order to perform this rendering. Hence, Boulanger (2008) neglects an alpha-blending and proposes full-screen anti-aliasing as for the rendering of geometry-based patch instances.

For that reason, Habel et al. (2007) propose another method for a correct alpha blending. Instead of geometry for every slice a quad as carrier geometry is utilized, which contains virtual slices. In the fragment shader stage, a ray is cast into the volume of the carrier geometry and the intersections with the implicit slices are calculated. Since several slices are aggregated into one rendering primitive of the carrier geometry, this method reduces the massive overdraw of the slices. This approach works similar to relief mapping (Policarpo et al., 2005) with the difference that the intersections of the ray with the grid structure of the slices can be calculated through a ray-plane intersection rather than an appropriate search algorithm for a height field.

We extend the proposed method of Habel et al. (2007) by utilizing a cuboid as carrier geometry (Figure 5.2). This allows for viewing the volume-based grass patch from arbitrary angles of view, whereas the quad is limited to be viewed from above. Since three faces of the cuboid are always hidden, back-face culling reduces the computational

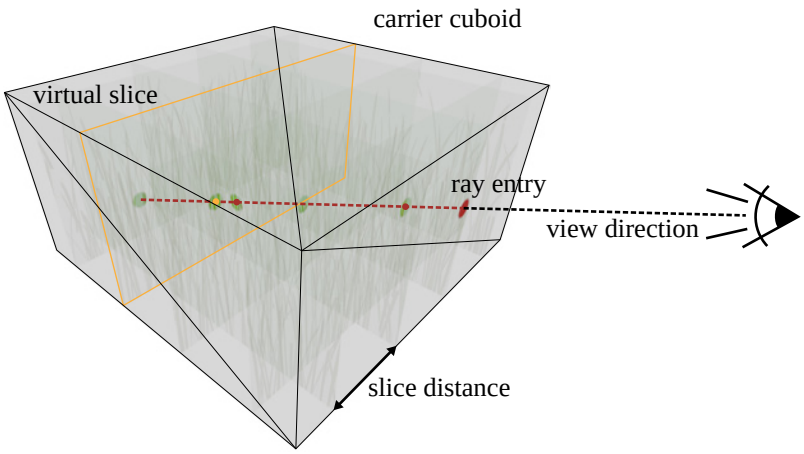

Figure 5.2.: *A ray is cast along the view direction into the carrier cuboid. The intersection with every hit slice is calculated (e.g. the orange) and the color of all intersection points with visible texels (red, orange) is used to determine the final color.*

effort to the three remaining faces. The dimensions of the carrier geometry match the bounding volume of the patch instances. Thus, the vertex of a corner $\vec{c_i}$ of an unit cube is transformed as follows:

$$\vec{c_i}' = C_{projection} \cdot C_{view} \cdot P_{instance} \cdot \left(\begin{bmatrix} 1 \\ patch_{height} \\ 1 \end{bmatrix} \cdot \vec{c_i} \right) \tag{5.2}$$

Thereby, the upper corners of the cube are scaled along the y-axis to deform the cube and shape a cuboid, which encloses the vertical slices according to the patch height. This cuboid is further transformed with regard to the camera's view and projection matrices C_{view}, $C_{projection}$ and the patch instance transformation $P_{instance}$.

Ray Entry and Direction

After the rasterization of the carrier geometry, a ray r is cast into the represented volume-based grass patch for each fragment. Such a fragment represents a point $\vec{p}$ on the surface of the carrier geometry. This surface point is the ray entry $\vec{e}$, which is determined by the interpolation of the corners of the cuboid. Further, the direction $\vec{d}$ of the ray is calculated by transforming the view direction $\vec{v}$ from the camera to the surface point $\vec{p}$ into local patch space. The patch space is defined by the patch instance transformation, which results in an equation for the ray r as follows:

$$r : \vec{x} = \vec{e} + \lambda \cdot \vec{d} \quad (0 \leq \lambda) \tag{5.3}$$

$$\vec{e} = \vec{p} \tag{5.4}$$

$$\vec{d} = \mathsf{normalize}\,(\vec{v} \cdot T_{VP}) \tag{5.5}$$

$$T_{VP} = \mathsf{mat3}\,(C_{view}) \cdot \mathsf{mat3}\,(P_{instance}) \tag{5.6}$$

Since the transformation matrix T_{VP} from view space to patch space is the same for all fragments of the carrier geometry, this matrix is calculated just once at vertex shader stage and shared for the respective fragments.

Selection of the First Intersected Slices

First, we declare a few constants used in the following calculations correlated to the number of planes p_{num} in each direction:

$$p_{iNum} = \frac{1}{p_{num}} \tag{5.7}$$

$$p_{iNum2} = 0.5 \cdot p_{iNum} \tag{5.8}$$

Since the vertical slices are parallel to the y-axis, the algorithm to find the intersected slices works independent of the y-component of the ray entry $\vec{e}$ and the ray direction $\vec{d}$. Thus, we consider the patch as a plane. Looking from above, the ray entry $\vec{e}$ is located inside a cell of the uniform grid structure of the vertical slices (Figure 5.3). This cell is bounded by up to four vertical slices. The lower left corner $\overrightarrow{pm}$ of this cell is determined as follows:

$$\overrightarrow{pm} = \mathsf{floor}\left(\begin{bmatrix} e_x - p_{iNum2} \\ e_z - p_{iNum2} \end{bmatrix} \cdot p_{num} \right) \cdot p_{iNum} + \begin{bmatrix} p_{iNum2} \\ p_{iNum2} \end{bmatrix} \tag{5.9}$$

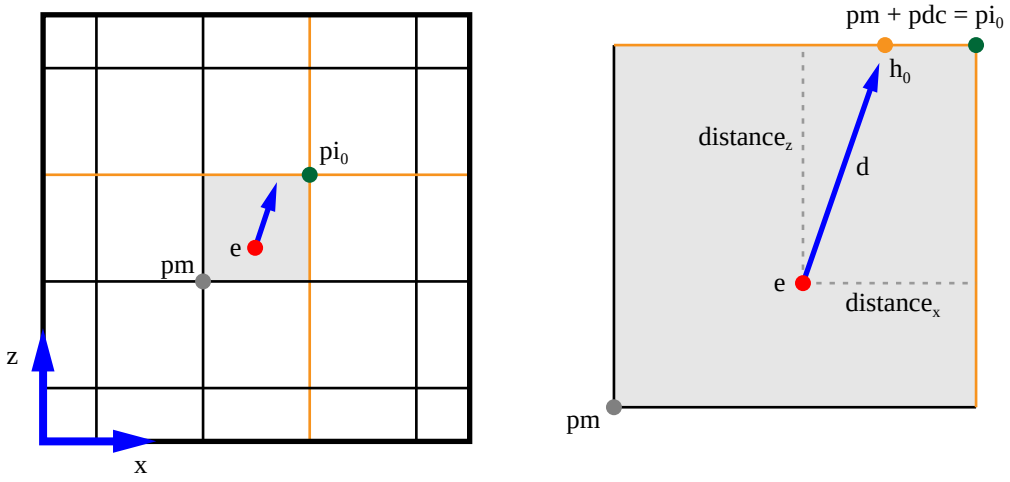

Figure 5.3.: *The volume-based patch from above. The ray entry e is located inside a cell bounded by up to four slices. The intersection point is determined by the direction d of the ray and the distance to the two next slices (orange).*

Subsequently, one corner of this cell is selected through an offset $\overrightarrow{pdc}$, which depends on the direction $\vec{d}$ of the ray (5.11).

$$\overrightarrow{pd} = \text{sign}\left(\begin{bmatrix} d_x \\ d_z \end{bmatrix}\right) \tag{5.10}$$

$$\overrightarrow{pdc} = (\overrightarrow{pd} + abs(\overrightarrow{pd})) \cdot p_{iNum2} \tag{5.11}$$

This offset is used to ensure an intersection of the same slice from both sides. Thus, the first intersected slices are at position $\overrightarrow{pi_0}$ and have the following plane definitions:

$$\overrightarrow{pi_0} = \begin{bmatrix} pi_{x_0} \\ pi_{z_0} \end{bmatrix} = \overrightarrow{pm} + \overrightarrow{pdc} \tag{5.12}$$

$$P_x : x = pi_{x_0} \tag{5.13}$$

$$P_z : z = pi_{z_0} \tag{5.14}$$

Ray-Plane Intersection

Knowing the position pi_i of the next slices, we calculate the distance of $\vec{e}$ to their planes (5.15). Since we consider the patch a plane and the slices axis-aligned, this calculation simplifies to the distance of a point to a line. Which slice is hit first by the ray further depends on the ray direction $\vec{d}$ and leads to a λ_i for the ray r as follow:

$$\overrightarrow{distance} = \begin{bmatrix} distance_x \\ distance_z \end{bmatrix} = \overrightarrow{pi_i} - \begin{bmatrix} e_x \\ e_z \end{bmatrix} \tag{5.15}$$

$$\vec{w} = \begin{bmatrix} w_x \\ w_z \end{bmatrix} = \begin{bmatrix} \frac{distance_x}{d_x} \\ \frac{distance_z}{d_z} \end{bmatrix} \tag{5.16}$$

$$\lambda_i = \text{min}\,(w_x, w_z) \tag{5.17}$$

Hence, the intersection point $\vec{h_i}$, which is the hit of the ray on the next slice, is calculated using (5.4) as follows:

$$\vec{h_i} = \vec{e} + \lambda_i \cdot \vec{d} \tag{5.18}$$

Ray-Tracing Loop

A volume-based grass patch consists of several slices in each direction. To each slice a semi-transparent texture is mapped, which contains opaque grass blades. Thus, the ray-plane intersection is repeated multiple times until the ray hit a grass blade within a slice. Therefore, we utilize a loop to trace the way of the ray, which performs the following steps:

1. Calculate an intersection point $\vec{h_i}$ with regard to the next slices $\overrightarrow{pi_i}$ as described above.

2. Sample the texture of the intersected slice and consider the sample for the final color. Skip this step, if the respective density threshold of the blade does not match the local grass density. Since the ray is cast into the volume, a rendering order from front-to-back is performed. Thus, we composite the final color as follows:

$$final'_{color} = final_{color} + (1.0 - final_{alpha}) \cdot h_{alpha} \cdot h_{color} \tag{5.19}$$
$$final'_{alpha} = final_{alpha} + (1.0 - final_{alpha}) \cdot h_{alpha} \tag{5.20}$$

3. Calculate $\overrightarrow{pi_{i+1}}$ according to the slice, which was intersected. Therefore, translate $\overrightarrow{pi_i}$ to the next cell corner as follows:

$$\overrightarrow{pi_{i+1}} = \overrightarrow{pi_i} + \vec{n_i} \cdot p_{iNum} \tag{5.21}$$

   Thereby, $\vec{n_i}$ is the normal of the intersected plane either $\vec{n_x}$ or $\vec{n_z}$.

This loop is repeated several times to process each slice that was not skipped by the initial selection of $\overrightarrow{pm}$ and $\overrightarrow{pdc}$. The execution of this loop is highly parallelized on the GPU due to the simultaneous processing of several fragments and quite fast, depending on the number of loop iterations.

On the one hand, the loop iterations will end, if the final color approaches sufficient opacity due to the hit of a grass blade in a slice. Besides the final color, the length of the ray is calculated as well and used for modifying the depth value of the fragment. This allows for a correct depth test with the rest of the scene, especially with objects located on the grass surface.

On the other hand, the loop will exit, if the intersection point $\vec{h_i}$ lies outside the volume of the carrier geometry, in particular when the ray leaves the cuboid on the backside. There is no slice anymore the ray can intersect with. In this case, the fragment is discarded as it does not represent a part of a grass blade of the volume-based grass patch.

5.3. Image-based Renderer

The grass patch represented as one horizontal slice is rendered as a semi-transparent texture onto a single quad. This quad consists of two triangles and is located at a low offset above the ground. The grass blades within the slice are opaque and the transparent part of the slice is blended with the previously rendered terrain surface. The fragments of hidden grass blades are discarded because of their density threshold value regarding the local grass density. Per-pixel illumination is applied to the visible grass blades.

5.4. Seamless Transitions

The transitions between the levels of detail are crucial. We use an approach for seamless and almost invisible transitions. Inside a transition zone a patch instance is rendered twice. One portion of the grass blades is rendered as one representation and the remaining part of blades as another. Therefore, the density management scheme is enhanced to perform this process (Boulanger, 2008).

Rendering a patch with the local density ld discards all grass blades with a density threshold value $bdth$ larger than the local grass density, no matter what representation is used. This could be expressed with a function for the opacity of grass blade depending on its density threshold.

$$\mathsf{opacity}_d\left(bdth\right) = \begin{cases} 1.0 & \text{if } bdth \leq ld \\ 0.0 & \text{otherwise} \end{cases} \tag{5.22}$$

Furthermore, a weighting function is defined for each level-of-detail zone, which depends on the blade's distance from the camera, e.g. $\mathsf{w}_1\left(d\right)$ and $\mathsf{w}_2\left(d\right)$ with a transition zone between e_1 and e_2 (Figure 5.4).

$$\mathsf{w}_1\left(d\right) = 1.0 - \mathsf{linearstep}\left(e_1, e_2, d\right), \left(e_1 < e_2\right) \tag{5.23}$$

$$\mathsf{w}_2\left(d\right) = 1.0 - \mathsf{w}_1\left(d\right) \tag{5.24}$$

In addition to this, we adapt the function for the opacity of the current grass blade.

$$\mathsf{opacity}_w\left(bdth, w\right) = \begin{cases} 1.0 & \text{if } bdth \leq w \cdot ld \\ 0.0 & \text{otherwise} \end{cases} \tag{5.25}$$

This results in a decreasing amount of grass blades, starting from the distance e_1, which is rendered as one representation. At e_2 no grass blade of this representation is visible

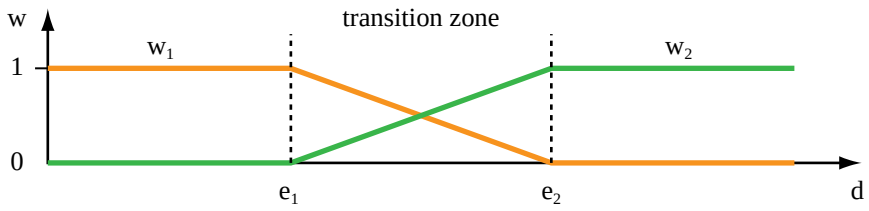

Figure 5.4.: *The weighting functions* $\mathsf{w}_1\left(d\right)$ *and* $\mathsf{w}_2\left(d\right)$ *for grass blades with a distance* d *to the camera defining a transition zone between* e_1 *and* e_2.

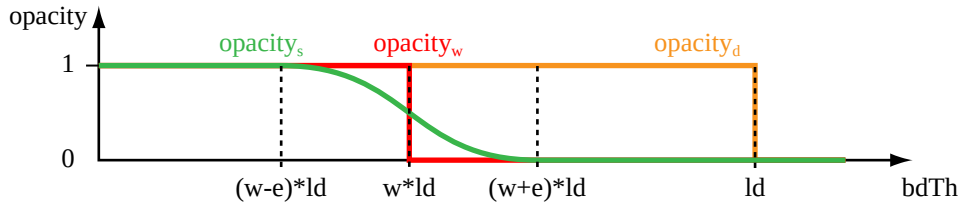

Figure 5.5.: *The graphs of the opacity depending on the blade's density threshold bdTh and the result of the weighting function w. The original function* $\mathsf{opacity}_d\,(bdTh)$ *discards only blades according to the local density ld. The modified version* $\mathsf{opacity}_w\,(bdTh, w)$ *takes w into account to shift grass blades between the representations. The function* $\mathsf{opacity}_s\,(bdTh, w)$ *utilizes a small window* $[w-\epsilon, w+\epsilon]$ *to perform a smooth blending when blades crossover the representations, thus avoid popping artifacts.*

anymore. The amount of grass blades of the other representation increases starting from e_1. Until e_2 is reached, all grass blades considering the local grass density are visible within the other representation.

This approach shifts individual blades between the levels of detail according to their density threshold. To avoid popping artifacts while a blade crossing the representations, we smooth the change of their opacity with a small range of blending. Boulanger (2008) uses a linear interpolation of the opacity within the blending range. We utilize instead the smoothstep function due to its continuity and the support through hardware instructions. The method is designed to have only a very low number of grass blades being semi-transparent at the same time.

$$\mathsf{opacity}_s\,(bdth, w) = 1.0 - \mathsf{smoothstep}\left((w-\epsilon), (w+\epsilon), \frac{bdth}{ld}\right) \tag{5.26}$$

Figure 5.5 shows the graphs of the different opacity functions.

Chapter 6

Terrain Management

The implemented grass-rendering system is able to cover large terrains with grass. The terrain is represented as a plane, which is defined by a base transformation. It has a rectangular size and is subdivided into uniform grid *cells*. The size of a cell matches the size of the grass patch (Figure 6.1). A data structure is utilized for a cell to provide, among other features, the cell transformation matrix to the renderers, which is required to draw a patch instance at the right place.

In order to fill the terrain with grass that is visible on screen, we render multiple instances of the grass patch located on cells of the terrain that are likely to be visible. Since it is inefficient to render a patch instance for every cell of the terrain, we need to organize them to determine the visible ones.

6.1. Quadtree Data Structure

A large terrain contains a huge amount of cells. The cells are spatially distributed over the plane due to the terrain's representation. We use a spatial hierarchical data structure

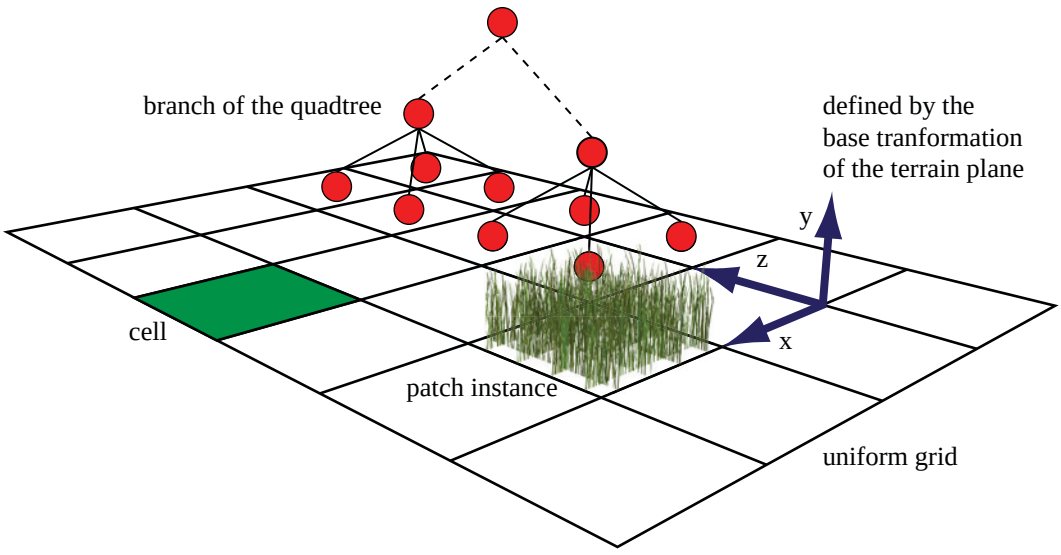

Figure 6.1.: *The terrain is represented as a plane oriented by means of a base transformation. It is subdivided into uniform grid cells and organized with a quadtree data structure. The size of a cell matches the size of a patch instance.*

to manage them, which means we utilize a quadtree (Samet, 1984).

The leave nodes of the quadtree contain single cells of the terrain (Figure 6.1). Furthermore, the nodes store additional cell data related to the quadtree like the bounding sphere of its content. The upper nodes refer to their up to four children and store the aggregated bounding volume in the form of a sphere, too. The overall size of the quadtree and thereby the size of the terrain is merely limited regarding its memory footprint.

6.1.1. Construction of the Quadtree

In a preprocessing step we build the tree data structure. The terrain has the arbitrary integral dimensions $columns \times rows$ cells. It is not required that the number of rows and columns are equal resulting in the terrain being squared. Furthermore, they are not necessarily a power of two. Given the dimensions of the terrain, the upper bounds for the depth of the quadtree and the number of nodes can be calculated as follows:

$$\begin{aligned} dim &= \mathsf{max}\left(columns, rows\right) \\ depth &= \mathsf{ceil}\left(\mathsf{log}_2\left(dim\right)\right) + 1 \\ nodes &= \mathsf{ceil}\left(\frac{4}{3} \cdot \left(\mathsf{nextPowerOfTwo}\left(dim\right)\right)^2\right) \end{aligned} \tag{6.1}$$

The maximum memory footprint for such a terrain is calculated by:

$$memory = nodes \times cellSize \tag{6.2}$$

The current cell data structure contains about 41 float values in matrices and vectors, which allocate approximately 164 bytes per cell depending on the machine platform. In one of our samples, we use a terrain with the dimensions of 380×532 cells to cover a whole football field with grass. As an upper limit for a balanced quadtree with squared and power-of-two dimensions enclosing the football field, an upper bound of the memory footprint is given by 219 MB. The actual required memory is 42 MB due to unused branches of the tree.

The generation of the quadtree is straightforward. A group of 2×2 neighbored leave nodes is aggregated to the next level node. This node is aggregated again together with three other neighbored nodes of the same level. Each node stores a bounding sphere that contains all child nodes.

While constructing the tree, we additionally utilize an axis-aligned bounding box, which exactly matches the content of a node, to calculate the stored bounding sphere rather than merging the bounding spheres of its children. This leads to usually smaller bounding spheres for upper nodes. The bounding sphere of the root node is surrounding the whole terrain.

In contrast to splitting the root-node area into four equal-sized parts and repeat this process, our bottom-up approach produces a non-balanced quad-tree. Certainly, this is rather undesired for spatial hierarchical data structures but it leads to nodes that almost always contain squared areas of the terrain. The utilized bounding sphere fits its content better in squared areas of the terrain than rectangular ones, which preserves the proportions of the root-node area that are likely to have huge differences between the long and the short edge (Figure 6.2).

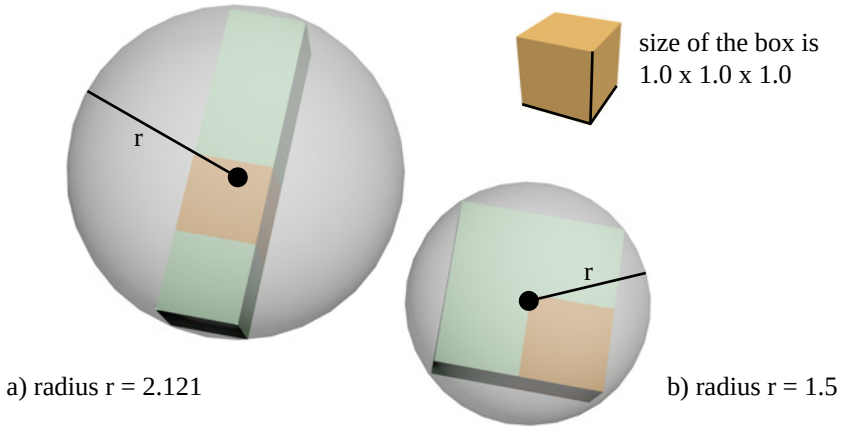

Figure 6.2.: *The bottom-up approach to build the quadtree data structure results in almost compact cell groups (b). The utilized bounding sphere fits its content in good approximation.*

6.1.2. Lod Tests

We use a level of abstraction to check whether to draw a grass patch instance for a cell or not. This level of abstraction is realized by *lod tests*. Multiple instances of lod tests - one for each level of detail - are attached to the terrain. These tests decide how to handle a node of the quadtree data structure with regard to their maintained level of detail:

- **Draw** - If the tested node is a leave and is located within the range of camera distances for the corresponding level of detail, a patch instance with the matching representation should be drawn.
- **Discard** - In this case, the the node is outside the range of camera distances for the corresponding level of detail so that the node is discarded and therefore also the complete branch of the quadtree.
- **Split** - If the node is at least partially located within the range of camera distances for the corresponding level of detail and the tested node is not a leave, the node is split and the four child nodes are processed.

We use additional types of lod tests to support different constraints in addition to the check whether a node is located within in the right distance of the camera. Thus, arbitrary conditions can be defined with this level of abstraction. In the remaining chapter we utilize this concept and present a few applications.

6.1.3. Traversal of the Quadtree

We traverse the nodes of the quadtree to collect the cells to be drawn with regard to the attached lod tests. While traversing, we try to discard a node early and together with that node all aggregated cells. Otherwise, we need to process its child nodes, which increases the depth of the traversal. First, we decrease the amount of cells by performing a view-frustum culling using the bounding sphere of the current node. We

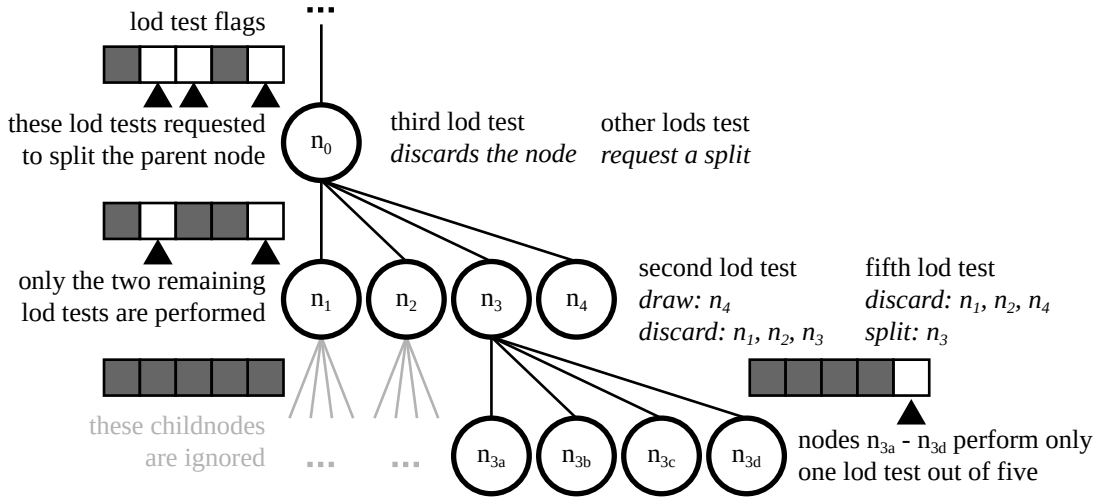

Figure 6.3.: *Flags are used to skip unnecessary lod tests for nodes of the quadtree. The nodes inherit the flags to their children.*

use a coarse but quite fast approximation to cull against the frustum, which is described in Section 6.1.4.

As a second step, we check the node against the lod tests. The result determines how to proceed the traversal of the quadtree. On Draw we collect the cell of the current node and proceed with a sibling node. On Discard we ignore the complete branch of the tree and proceed with a sibling node as well. Yet on Split, we need to go deeper in the tree and proceed with the children of the current handled node. We perform only a single traversal of the quadtree rather than one for every lod test attached to the terrain. Thus, a node is checked simultaneously against all lod tests, which will probably differ in their results. We need to subsequently process the child nodes as well as an lod test can discard the node, whereas another lod test requests a split. This leads to unnecessary checks of lod tests for these nodes, if the parent node was already discarded by one. So, we utilize flags to skip lod tests for a node and all child nodes to avoid the unnecessary checks.

Figure 6.3 illustrates this approach by a part of a branch of the quadtree. The node n_0 is only checked against three of the five lod tests attached to the terrain. Since the third lod test discards the node, its children need to perform only the two remaining lod tests. The flags which lod tests can be skipped are inherited by the parent nodes to their children. Finally, the leave nodes n_{3a} - n_{3d} perform only one lod test out of five.

6.1.4. View-Frustum Culling

We use a naive implementation of view-frustum culling to limit the amount of cells to those that are likely to be visible. Therefore, we utilize the bounding spheres stored within the quadtree nodes. The view frustum of the camera is approximated by a sphere and a cone (Figure 6.4). This approach is a coarse but quite fast approximation (Picco,

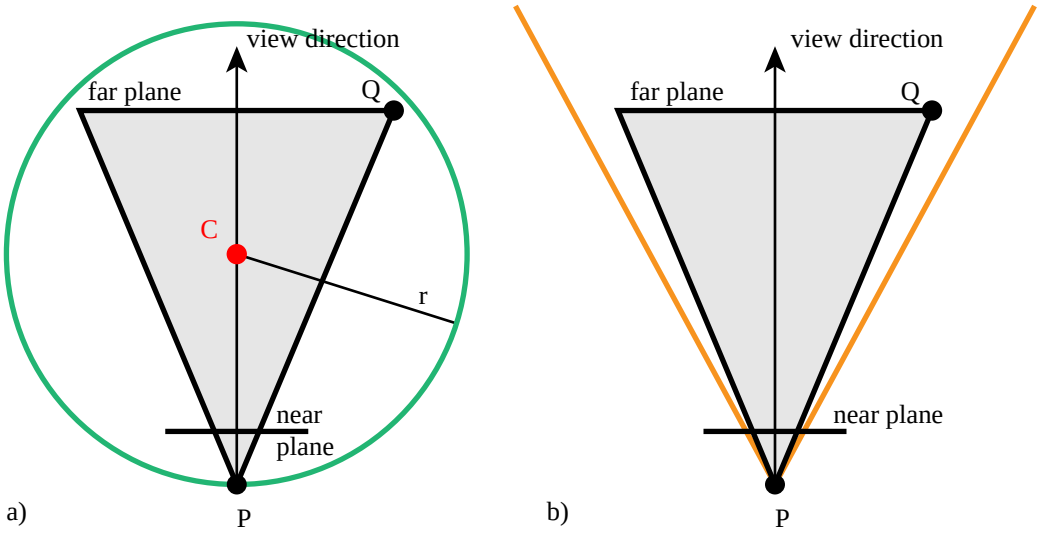

Figure 6.4.: *The view frustum (grey) of the camera at position P is approximated by a sphere (a) with its center at C and a cone (b) with its vertex at P and axis in view direction.*

2003).

A node is culled, if its bounding sphere does not intersect with the sphere that surrounds the view frustum. The center C of the sphere is half-way between the near and the far plane along the view direction. The radius r is according to the distance between C and the upper left corner of the view frustum on the far plane Q. Figure 6.4 shows a cut of the sphere in an orthographic projection from above, which results in a seeming difference between $\overline{CQ}$ and r. Many definitely invisible nodes are already culled after performing the efficient sphere-sphere-intersection test. However, surrounding the view frustum beyond the upper, lower, left and right clipping planes, few probably invisible nodes still remain.

A remaining node is culled, if its bounding sphere does not intersect with the cone enclosing the view frustum. The apex of the cone is at the camera position P with its axis in view direction. The angle of the cone is determined by the angle between the view direction and $\overline{PQ}$ to appropriately surround the frustum. Likewise, the sphere-cone intersection test is very efficient as well. A sophisticated implementation is presented by Eberly (2002).

With these two quite fast intersection tests almost all probably invisible cells are culled. The implementation of the view-frustum culling additionally benefits from the pre-calculation of few trigonometric values such as $\sin^2(cone_{angle})$ and $\cos^2(cone_{angle})$.

6.2. Macro-Cells

On large terrains with grass in the distance, the number of cells that are collected while traversal is excessive. Each cell requires the rendering of a patch instance. Boulanger (2008) proposes to aggregate multiple small cells to bigger ones, called *macro-cells*, to further reduce the overall number of patch instances (Figure 6.5).

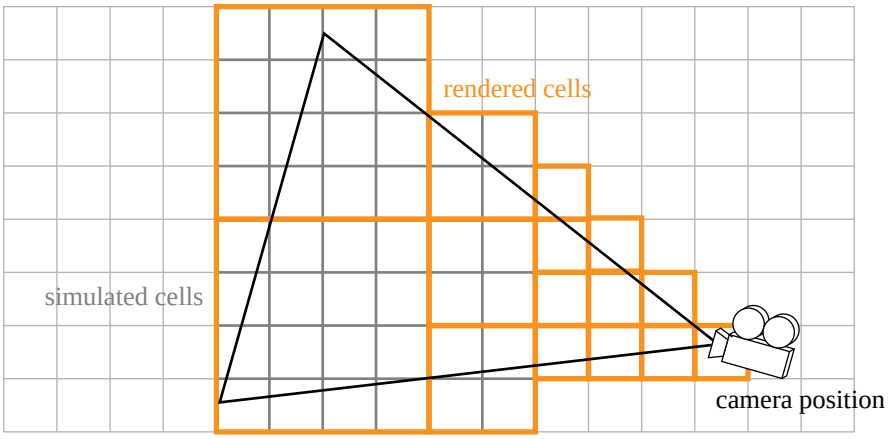

Figure 6.5.: *With an increasing distance of the camera, the cells are aggregated to macro-cells. These macro-cells in turn simulate the basic grid structure of the terrain. This hides the replacement of cells through macro-cells.*

Using such a macro-cell skips the processing of the following quadtree branch, which reduces the traversal depth. A macro-cell only requires the rendering of one patch instance that simulates the aggregated basic grid cells. This is provided by the volume-based and image-based approaches through texture repetition. The geometry-based approach does not support macro-cells. Due to its exclusive employment with low distances of the camera, it is unnecessary as well.

To utilize macro-cells, we adapt our lod tests with a new type that splits nodes until they reach a proper pre-defined size. Otherwise, this lod test triggers the drawing of a patch instance for the cell of the current node. The approach is optimized to discard nodes or draw their cells, which are almost macro-cells, as early as possible. Hence, the amount of checks of nodes against the lod tests is reduced. Furthermore, the overall traversal depth and therewith the number of processed nodes decrease significantly.

It is necessary to extend the cell data structure, which is used by the renderers to draw a patch instance. Beside the cell transformation matrix, we store the dimensions of the cell. The size of the cells in the leave nodes is 1×1, which matches the size of the basic patch instance (Figure 6.1). The upper nodes contain cells with dimensions $2n \times 2n$, while their children consist of cells with dimensions of $n \times n$. At the edges of the terrain surface the macro-cell dimensions are adjusted and be not necessarily equal.

6.3. Aperiodic Tiling

Repeating a small single grass patch over a large terrain surface shows undesired repetitive patterns (Figure 6.6.a). In Chapter 3 we mentioned the aperiodic tiling scheme to mask repetitive visual structures. This aperiodic tiling is not based on several versions of the grass patch as common for Wang tiles (Wang, 1960). These versions increase the required memory and complicate the patch management.

For that reason, Boulanger (2008) proposes the usage of only one patch and to randomly mirror its instances along the two axis while rendering. This results in four

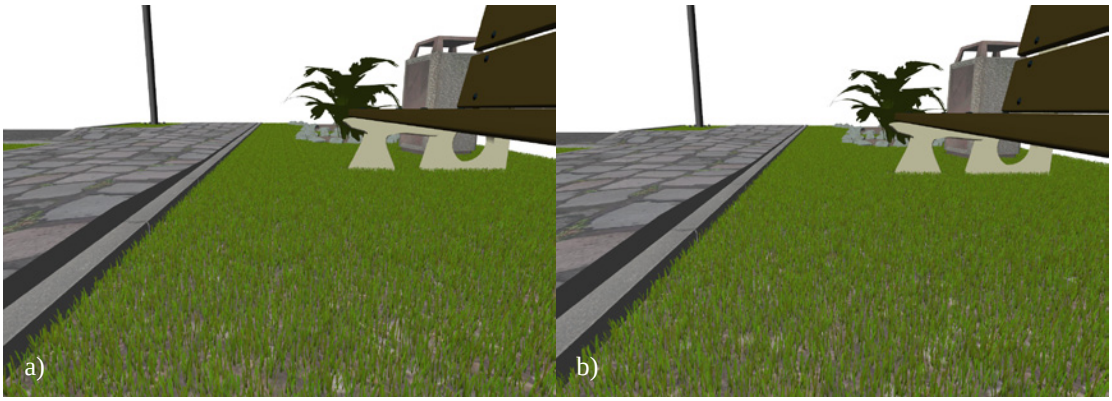

Figure 6.6.: *Repeating a small grass patch over the terrain surface causes undesired repetitive patterns (a). The aperiodic tiling scheme mask the repetitive visual structures (b).*

visual distinct patch instances with no additional memory required and only a small computational overhead. Which patch instances are mirrored is decided for each single cell using an orientation map. In the first channel (red) of this map the value -1 is stored, if the patch instance should be mirrored along the x-axis. Other than that, a 1 indicates no mirroring for this direction. In the second channel (green) of this orientation map the values -1 or 1 store the symmetry along the z-axis. Thus, the symmetry vector s_{om} for a cell with the normalized coordinates u, v is defined as:

$$s_{om} = \begin{bmatrix} s_x \\ s_z \end{bmatrix} = \begin{bmatrix} \mathsf{orientationMap}\,(u,v)\,.red \\ \mathsf{orientationMap}\,(u,v)\,.green \end{bmatrix} \tag{6.3}$$

We continue the approach of Boulanger (2008). Instead of the orientation map we utilize a general high frequency 2D-noise map yielding values within the interval $[0,1]$ to decide the orientation of the patch instances. Unlike the orientation map, the noise map is not sized for specific terrain dimensions. That simplifies the terrain management. The symmetry vector s is then determined by:

$$s = \begin{bmatrix} s_x \\ s_z \end{bmatrix} = 2 \cdot \mathsf{step}\left(0.5, \begin{bmatrix} \mathsf{noiseMap}\,(x,y)\,.red \\ \mathsf{noiseMap}\,(x,y)\,.green \end{bmatrix}\right) - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{6.4}$$

To provide a consistent lookup into the noise map within a patch instance, we use the transformed origin of the cell into world coordinate system as input modified by a constant matrix to control the sampling rate.

$$\begin{bmatrix} x \\ y \end{bmatrix} = sampleRate \cdot cellTf \cdot \begin{bmatrix} \mathsf{floor}\,(u) \\ 0 \\ \mathsf{floor}\,(v) \end{bmatrix} \tag{6.5}$$

With regard to the macro-cells, an arbitrary point in cell space is transformed to one of four possible destinations according to the right mirrored version of the patch instance as follows:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \mathsf{floor}\left(\begin{bmatrix} u \\ v \end{bmatrix}\right) + \mathsf{fract}\left(s \cdot \begin{bmatrix} u \\ v \end{bmatrix}\right) \tag{6.6}$$

The image-based and volume-based rendering methods apply equation (6.6) to determine the access of the slice textures. Since the grass blades in the geometry patch potentially reach outside the cell, this case is apart and the vertices of the grass blade geometry can not simply be transformed according to (6.6). Hence, the vertices of the grass blades are transformed by translating the patch to get it centered above its origin, after being mirrored along the axis reversing the translation:

$$\begin{bmatrix} x'_g \\ y'_g \\ z'_g \end{bmatrix} = \left(\begin{bmatrix} x \\ y \\ z \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \right) \cdot \begin{bmatrix} s_x \\ 1 \\ s_z \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \tag{6.7}$$

By means of the described approach, we randomly mirror the patch instances over the terrain. This aperiodic tiling scheme masks most of the unpleasant repetitive structures. By using a general 2D-noise map, we decouple the tiling from the cell-based orientation map simplifying the terrain management. The resolution of this noise map is independent from the dimensions of the terrain and the overall number of cells. For that purpose, we utilize only a small help texture, which could be used for other noise related cases as well.

6.4. Shape, Model, Design and Style the Grass

So far, we can create a large terrain covered with grass. Multiple instances of the grass patches based on the three kinds of representations are rendered onto the terrain surface. We are able to design a patch to simulate a specific kind of grass. Here, the number of grass blades for a patch is variable. The grass blades differ in their shape and color. We can continuously thin a grass patch. The crucial point is that all these customizations are applied to just one patch, which is tiled over the whole terrain. By means of all the provided capabilities, we can neither render a football field because of the marks nor adequate show other scene objects on the terrain interacting with the grass surface.

The customization of the overall terrain appearance is an integral feature and several use cases are common, which require a localized modification of the grass. One of the most important cases is the integration into an existing scene, so that the overall picture is the measurement for such an integration. Therefore, the grass needs to fit perfectly into

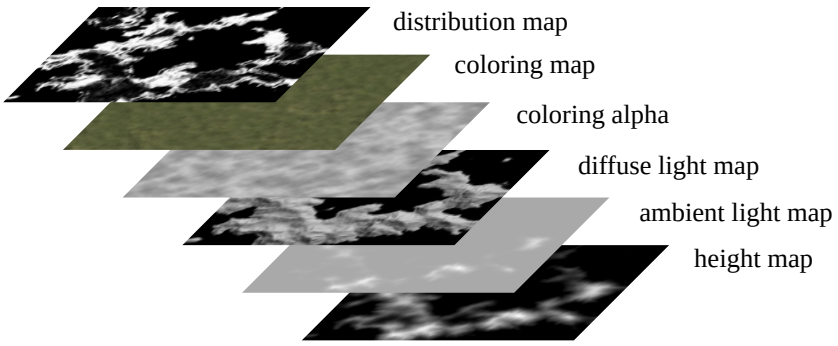

Figure 6.7.: *A stack of design maps to customize the appearance of the grass on the terrain surface.*

the desired result. A viewer can easily notice even small discrepancies in the rendering. Thus, the interaction between the grass and other objects of the scene requires a proper mapping. The surrounding scenery influences the grass with regard to its density, color and lighting.

For all these cases, an artist could previously design the grass on the terrain surface to reflect the desired properties. For that purpose, several design maps, like the grass-distribution map or the light maps, are available to modify the conditions affecting the grass rendering (Figure 6.7). The design maps are projected along the terrains up-axis and mapped on the terrain surface. In a preprocessing step, these maps are packed into a stack of images and stored as texture array. This simplifies the access due to a consistent lookup for a point on the terrain.

We utilize the cell data structure to store the relevant data, which contain the following informations:

- *cellTf* - The transformation of the cell in world space coordinate system. The renderer uses this matrix to draw the patch instance at the right place.
- *size* - The size of the cell is stored because of the usage of macro-cells. The volume-based and image-based renderers scale the patch instance to the according size.
- *configTf* - This matrix transforms the cell space into the design map space, which is the texture space of the design maps. It is used by the renderers to retrieve the relevant rendering configuration about parameters, such as local density or color modulation.

The explicit transformation into the design map space stored in the cell data structure provides the ability for a few features of the terrain. For scenes with multiple terrains a packing into one design-map stack is possible. This helps to further simplify the management of the renderer as they do not need to access different ones. Moreover, it is possible to replace the design transformation for a branch of the quadtree structure to map the cells into another area of the design map. This enables designing a detailed part of the terrain in contrast to the remaining part, which is designed in a coarser manner.

6.4.1. Non-Uniform Distribution of Grass

A terrain surface covered with natural grass is never uniformly covered. The local density of grass is affected by various environmental influences. In particular the type of ground and the structure of the terrain defines the distribution of grass, i.e. there is barely grass growing on paths or tracks. Furthermore, other objects in the scene laying on the grass like rocks or tree roots affect the local density. External strain affects the local grass density as well, a goal keeper for instance, who mainly moves short ways around his goal. So the degree of degradation at this place is higher resulting in a lower grass density. The density management enables the shaping of an arbitrary clump of grass as well. In our sample *beet* we realize a creative shape of the bed by utilizing the density management.

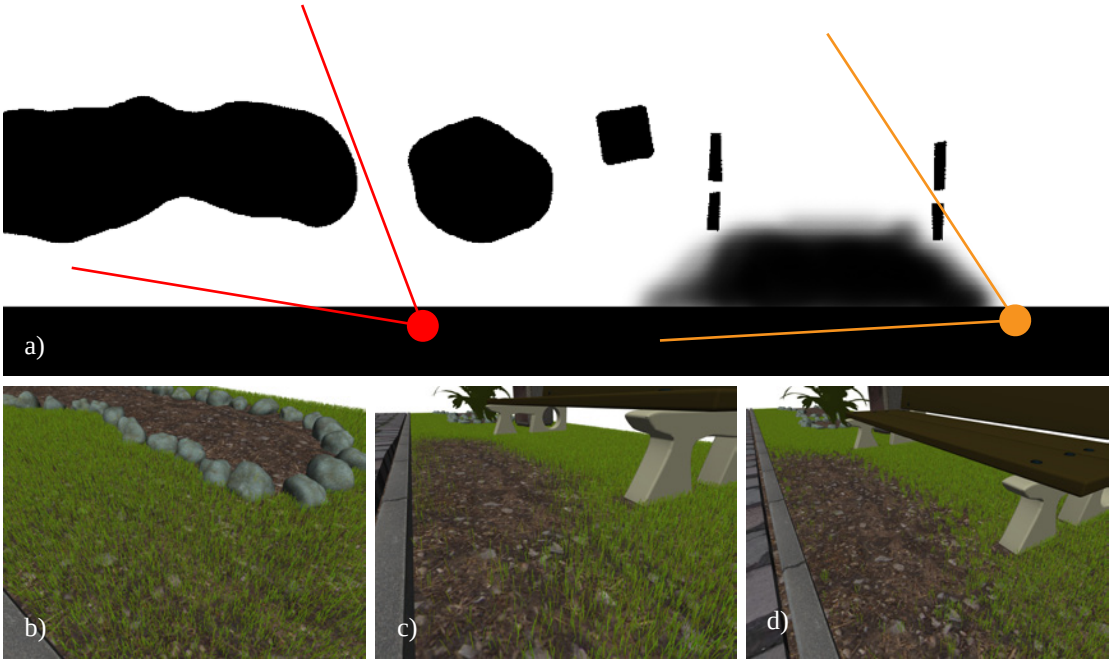

Figure 6.8.: *The distribution map (a) affects the grass density to exclude areas completely from grass (b) or thin the grass surface due to environmental influences. The appearance of grass in low density areas is configurable. It can be uniformly distributed (c) or more natural-like clustered (d).*

The distribution or so-called density map is a gray-scale image containing white texels for the highest grass density and black texels for a lack of grass (Figure 6.8). The density map is filtered bilinear to provide smooth transitions between areas with a high density and areas with lower densities. The appearance of the grass patch at low densities is configurable as described in Section 4.1.3.

6.4.2. Coloring the Grass

The color of grass blades is varied within the patch. Even the color of a single grass blade is individually modified to imitate nature's variance. However, due to tiling the same patch over the whole terrain, it still looks quite similar. The application of an aperiodic tiling scheme as described in Section 6.3 breaks merely the visual repetitive patterns but the overall appearance of the terrain surface is still homogeneous, especially when observed in the distance.

Therefore, we apply a color modulation of the grass on the terrain surface. A user-defined color modulation map is utilized to vary the color across patches spanning all over the terrain (Figure 6.9). This map contains colors provided with alpha values. The final color C for a point on the surface of a grass blade is determined by blending the grass color, which is modulated inside the patch by means of the overall color modulation map as follows:

$$C = A_c \cdot C_c + (1 - A_c) \cdot M_b \cdot C_b \tag{6.8}$$

The color C_c with its respective alpha value A_c is sampled by using bilinear filtering

Figure 6.9.: *The color modulation map (a) is applied to the grass surface (b) and and colorizes the blades (c).*

from the color modulation map. Hence, smooth transitions between different colored areas are provided. The color retrieved from the grass blade texture C_b is modulated by the blade's and vertex color modulation coefficients composited in M_b and then blended with the color sampled from the color modulation map.

We use this method in two ways. With high alpha values the grass blades are almost full-colored from the color modulation map. On a football field, for example, we provide the field marks in that fashion. With lower alpha values we slightly modify the color of the grass surface. Using an appropriate noise to modify the grass color this subtly enhances the variation across the patches and increases natural individuality. Different species of grass or the partial absence of water can be designed in that manner. Furthermore, a light color modulation breaks remaining visual pattern structures after applying the aperiodic tiling scheme.

6.4.3. Illumination

A plausible illumination given to the grass surface is essential for a photo-realistic portrayal of grass structures. There is usually one directional light source present in outdoor scenes, which mimics the sun. As we handle the surface of grass as a diffuse-only material without specular reflection, we apply the Lambert reflection model when illuminating the grass. An ambient lighting term is added with regard to blade-blade and scene-grass light transmission (Boulanger, 2008). The lighting equation for a surface point on the grass with the normal N is defined as follows:

$$light = I_{ambient} \cdot A_O \cdot ambient_{material} + I_{diffuse} \cdot material_{diffuse} \cdot \max(0, N \cdot L) \qquad (6.9)$$

The light direction L is globally static for the scenery. The normal N for a point on the grass blade's surface is determined with regard to the usage of a two-sided material on the geometry. The intensities of the light source $I_{ambient}$ and $I_{diffuse}$ are customized with light maps (Ramamoorthi & Hanrahan, 2001). These light maps are static and generated in a preprocessing step.

Thereby, the ambient light map contains the ambient intensity of the light source applied with the ambient occlusion factor from the scene. Other scene objects affect the light that is transmitted to the terrain surface covered with grass. Ambient occlusion is an approximation to simulate global illumination. It provides an occlusion factor based on a visibility function on a hemisphere over a surface point. This is constant for static

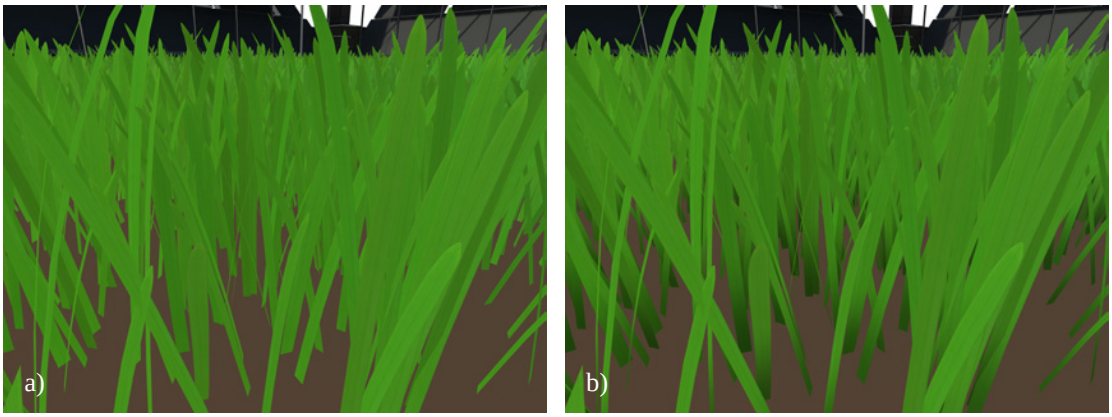

Figure 6.10.: *The blade-blade light transmission is simulated by decreasing the ambient light portion received by the surface of the grass blade. This results in brighter tips than roots of the grass blades (b) rather than uniform ambient light distribution (a).*

scenes. To simulate the blade-blade light transmission we adapt the ambient occlusion factor A_O as follows:

$$A_O = AO_{scene} \cdot AO_{blade} \tag{6.10}$$

It is split up into the portion AO_{scene} which is already applied to $I_{ambient}$ when sampling the ambient light map. The latter part AO_{blade} represents the occlusion by neighbored blades (Figure 6.10). It is approximated by using the height of the surface point P_y over the terrain and the local grass density (6.11).

$$AO_{blade} = 1 - B_{impact} \cdot density \cdot \mathsf{smoothstep}\,(B_{height}, 0.0, P_y) \tag{6.11}$$

Two parameters control the influence of this effect (B_{impact}) to the overall ambient occlusion and the effective height over the ground (B_{height}) of this effect.

The diffuse light map contains the varying diffuse light portion of the light source lit to the terrain surface. A shadowing effect being applied to the terrain surface as a variation of the diffuse light intensity received by the terrain leads to visual high quality umbras casted by the surrounding scenery.

Both light maps, the ambient and the diffuse map, are created in a preprocessing step. The usage of a high quality generator as an offline rendering pass is possible. The lighting equation in the framework is replaceable. Thus, the application of other illumination models to the grass is possible. The surrounding scene may utilize a different illumination model. To increase the visual integration, an adaption of the illumination model of the grass is necessary.

In contrast to Boulanger et al. (2006), we apply the same lighting model to the image-based and volume-based methods as to the geometry-based method. Due to the simplified geometry of the vertical and horizontal slices, we use normal maps to retrieve the correct normal for a surface point (Blinn, 1978). The normal maps store the normals in patch space, which makes a complex transformation obsolete. With the retrieved normal we can perform the lighting by using the same light equation as for the geometry. This results in a consistent illumination among the different representations of grass.

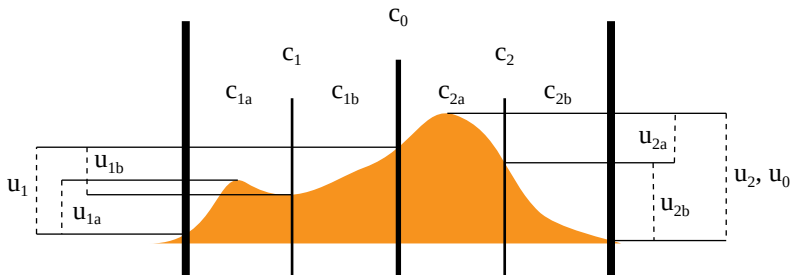

Figure 6.11.: *The terrain unplanarity u_i of the cell c_i is defined as the difference of the highest and the lowest altitude of the terrain within the cell.*

6.4.4. Grass Displacement

On flat terrains the cell transformation applied to a patch instance is only a translation in plane. This is a very efficient operation. Nonetheless, most terrain surfaces are not plain. The terrains may represent real-world elevation data or are an arbitrarily shaped surface. Most virtual 3D-city models contain small irregularities of the terrain surface. Other than that, landscape models have large differences in their altitude, if they contain hills or canyons. A displacement scheme for the grass is necessary to handle these use cases.

We utilize a gray scale image to represent the height of the grass covered terrain in a normalized manner. The values are stored relatively to the plane representing the terrain, whose orientation is defined by the base transformation. This height map is generated from terrain geometry in a preprocessing step. Therefore, the terrain is rendered from above with a camera using an orthographic projection. Alternatively, we could use the existent elevation data that had been the basis for the terrain geometry. This needs a proper conversion of values. Our height map data is normalized and scaled by a parameter to exactly match the terrain's altitude range.

Using the generated height map, we perform a displacement mapping on the GPU (Szirmay-Kalos & Umenhoffer, 2008). This leads to a irregular distortion of the terrain cells affecting the rendering of the patch instances through the respective renderer. Accordingly, we introduce a measurement of the *terrain unplanarity* for a cell. The terrain unplanarity represents the difference between the lowest and the highest altitude of the terrain within the cell (Figure 6.11). The bounding volume of cells increases with regard to their terrain unplanarity due to the larger extent of height. Recent approaches on displacing grass use the elevation data to sample the height of the four corners of a cell (Perbet & Cani, 2001; Boulanger et al., 2009). Then they calculate the height values for an arbitrary point P within the cell using a bilinear interpolation (6.12).

$$P_y = (1 - P_x) \cdot (1 - P_z) \cdot h_{00} + P_x \cdot (1 - P_z) \cdot h_{10} + (1 - P_x) \cdot P_z \cdot h_{01} + P_x \cdot P_z \cdot h_{11} \quad (6.12)$$

This method of displacement supports merely a low resolution of elevation data. On large terrain surfaces with even small rough structures, such as drainage ditches on golf courses or furrows on fallow fields, this results in an undesired flatness due to the low sampling frequency. However, reducing the patch size to increase the supported resolution of height-map data is not an alternative because of the increasing computational overhead.

We propose an alternative approach to draw grass on rough terrain surfaces. Like the approaches of Perbet & Cani (2001) and Boulanger et al. (2009) our displacement method is limited to the up-axis of the terrain. Nevertheless, this is sufficient due to the fact that grass growths up on skewed surfaces as well. The normal of the terrain surfaces is not relevant to the growth direction of grass.

Displacement of Geometry-based Patches

Our displacement method uses the capabilities of modern hardware to perform vertex displacement. In the vertex shader the height map data is sampled at the root position of the blade. By multiplying the sampled value with the height-map scale the exact height of the terrain surface at this point is calculated. Bilinear filtering is applied when sampling the texture data of the height map. Each vertex V of the blade is translated along the y-axis with regard to the determined height as follows:

$$V'_y = V_y + height_{scale} \cdot \mathsf{heightMap}\,(blade_x, blade_z) \tag{6.13}$$

The root position of a blade ($blade_x$ and $blade_z$) is attached to each one of its vertices to provide consistent lookups. Otherwise, a lookup based on the vertex position within the patch results in distinct displacement values depending on the bending of the blades. High-resolution height maps allow to exactly match the terrain surface of the scene. This is an important feature for the geometry-based patch due to the low distance from the camera.

Displacement of Volume-based Patches

The displace of the volume-based grass patch is far more complex. Nevertheless, its implementation is quite efficient in order to use it on large terrains. The volume-based patch is rendered by using a cuboid as carrier geometry. This method was described in Section 5.2. The vertices of the carrier geometry are displaced first. In consequence, we extend the cell data structure to store the minimal height that could be sampled while rendering this cell. Together with the terrain unplanarity, stored by the cell data structure as well, the corners V of the cuboid are displaced to generate an extruded carrier geometry as follows:

$$V'_y = \left(patch_{height} + cell_{unplanarity}\right) \cdot V_y + cell_{minHeight} \tag{6.14}$$

The extruded geometry encloses the uneven terrain surface. The vertical slices are extended as well by the vertical stretching of the cuboid. These implicit surfaces now reach from the lowest to the highest height of the terrain with regard to the cell dimensions.

The ray tracing is executed after rasterization stage, in the fragment shader. When the ray hits one of the vertical slices, its textures are sampled at the location s, t with $hit_{x,z}$ depending on the direction of the slice (6.15).

$$sliceLookup = \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} hit_{x,z} \\ patch_{invHeight} \cdot hit_y \end{bmatrix} \tag{6.15}$$

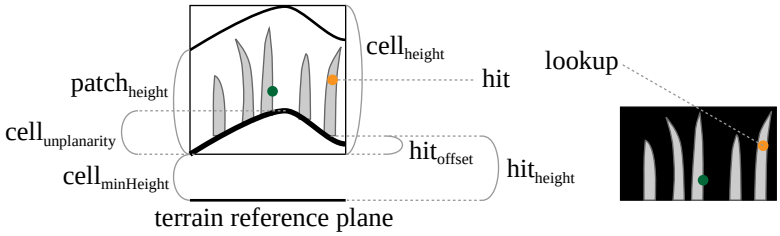

Figure 6.12.: *The lookup coordinates to the texture of the slices is shifted to support an arbitrary shaped terrain surface.*

To support displacement, a shift of the texture lookup coordinates is additionally applied depending on the height of the terrain (Figure 6.12). The sampling of the height map is performed by utilizing the hit point:

$$hit_{height} = height_{scale} \cdot \mathsf{heightMap}\,(hit_x, hit_z) \tag{6.16}$$

Due to the fixed position and the orientation of the slices inside the patch, the sampling result is consistent with regard to the vertical dimension. Then the offset is calculated to shift the hit point into the local distorted cell space (6.17).

$$hit_{offset} = hit_{height} - cell_{minHeight} \tag{6.17}$$

Thereby, any sampled value $slice_{height}$ in the cell is always larger than $cell_{minHeight}$. If the shifted value hit'_y is outside the interval $\left[0, patch_{height}\right]$ then it is not considered for the final color of the ray. The shifted slice lookup coordinates are calculated as follows:

$$shiftedSliceLookup = \begin{bmatrix} s \\ t' \end{bmatrix} = \begin{bmatrix} hit_{x,z} \\ patch_{invHeight} \cdot (hit_y - hit_{offset}) \end{bmatrix} \tag{6.18}$$

The distortion applied to the cell results in a distortion of the vertical slices. Subsequently, the aggregated grass blades are distorted similar to the approach of Boulanger (2008). However, for vertical slices with a sufficient resolution, there are no distortion artifacts visible due to the moderate distance from the camera.

Displacement of Image-based Patches

With the image-based patches, only a displacement of the four corners is applied. Inside the patch, the height of the grass is bilinear interpolated flattening the terrain surface. In order to preserve most of the terrain structure, the macro-cells, which are rendered with the image-based technique, are progressively split. This leads to further height sampling points due to more patch corners at costs of an overhead of image-based patch instances.

Displacement Integration

While building the quadtree structure, an additional sampling of the height-map data is necessary. For each cell including the macro-cells, the terrain unplanarity and the

minimal height are calculated and stored in the cell data structure. Furthermore, a new type of lod test is introduced, which splits cells until their terrain unplanarity fall below a pre-defined threshold. This restricts the excessive usage of macro-cells for image-based patch instances to almost flat ones (Boulanger, 2008) since otherwise popping artifacts appear at hills. By the means of the described displacement method, we support arbitrary formed terrain surfaces independent from the terrain grid subdivision decoupling the size of the grass patch from the height map data resolution.

6.4.5. Multiple Types of Grass

Typical golf courses are covered with one species of grass. The green keeper maintains the absence of other types of grass to ensure that the requirements for this sport are fully met. In contrast, on a general meadow, there are severals types of grass among flowers and other ground vegetation. Hence, it is desired to maintain multiple types of grass on the same terrain surface.

Therefore, we render multiple patch instances with different types of grass within a cell. Using multiple density maps enables different grass distributions on the terrain surface. This allows for clusters of grass types, for instance. By attaching additional lod tests to the terrain, the different grass types are able to use distinct transition zones and blend configurations so they share merely the terrain - and scene-specific design maps like the light maps.

Each grass type utilizes by default three lod tests - one for each level of detail with a respective representation. More lod tests can be attached to the terrain for applications that require more than three levels of detail. They could use multiple volume-based patches with different amounts of vertical slices, which would provide a great flexibility to adjust the transition zones of a level-of-detail scheme.

6.4.6. Silhouettes on the Top of Hills

On the top of hills natural grass shapes a characteristic silhouette of the terrain surface. An almost uniform background - atmosphere would serve as an example here - brings off this silhouette.

Grass on distant hills is probably rendered with an image-based representation leading to an undesired flatness at the top of hills. Thus, to preserve the characteristic silhouette shape, Boulanger (2008) proposes to locally use additional vertical slices. We follow this approach by introducing a new type of lod test to render additional volume-based patch instances at hilltops. This lod test collects cells starting in a distance, where generally no volume-based grass is present. Every cell that contains an image-based patch instance will additionally be rendered with the volume-based representation, if its bounding sphere reaches above the camera position. This implies that the cell is observed with a lateral view angle and constitutes a possible part of the silhouette.

Chapter 7

Results and Discussion

We applied the presented grass-rendering system in several sample applications to cover terrain surfaces with grass, for example a football field. The field has the dimensions of 380×532 cells and contains about one billion virtual grass blades. The generated geometry-based patch contains 5000 grass blades in the style of short-cut turf for sports fields. We apply five levels of detail to the terrain surface with three intermediate levels of volume-based grass representation. They differ in their number of vertical slices from 1 over 2 to 8 in each direction. The vertical slices use textures with a resolution of 256×64 texels. The image-based grass patch utilizes a texture with a resolution of 128×128 texels. The lod-blending is configured as follows:

level of detail	**representation**	**begin**	**end**	**macro-cells**
0	geometry-based	z_{near}	20.0	–
1	volume-based (2x 8 slices)	5.0	80.0	max. 40
2	volume-based (2x 2 slices)	50.0	160.0	max. 40
3	volume-based (2x 1 slice)	100.0	300.0	max. 40
4	image-based	200.0	z_{far}	max. 500

The football field is additionally surrounded by a stadium model. This model consists of about 100 thousand vertices and twice as much triangles and represents of further scene geometry, which affects the performance.

The sample application is run on a test system powered by an Intel i7-920 at 2.6 GHz with 6 GB of main memory and a NVIDIA Geforce 9800 GT graphics card. The demo runs smoothly at a resolution of 1024×768 pixels and 4x anti-aliasing. We achieve varying frame rates of 13 to 535 frames per second (fps) with an average of 56 fps. We take measurements of several selected viewpoints due to the broad range of frame rates.

The rendering results are shown in Figures 7.1 to 7.4. Besides the rendering results, the measured rendering times are broken down. Next to the rendering results the mid part of the respective image is additionally colorized to visualize which rendering technique is used: image-based method (blue), volume-based method (yellow) or geometry-based method (red). The rendering time of the whole image t_a leads to the respective frame rate (fps). It is compounded of the time to traverse the quadtree data structure (t_q), the rendering time for the geometry-based (t_g), volume-based (t_v) and image-based (t_i) patch instances and the rendering time for further scenery (t_m), such as the stadium. The number of cells per rendering method is cited as well (c_g, c_v, c_i) to estimate the amount of rendered patch instances. A small difference can be observed between the

Figure 7.1.: *The football stadium rendered from a high point of view.*

177	5.42 ms	0.47 ms	0.0 ms	0.79 ms	1.33 ms	3.02 ms	0	15	5	↑
fps	t_a	t_q	t_g	t_v	t_i	t_m	c_g	c_v	c_i	
15	68.12 ms	0.47 ms	41.44 ms	23.31 ms	1.23 ms	2.35 ms	106	54	5	↓

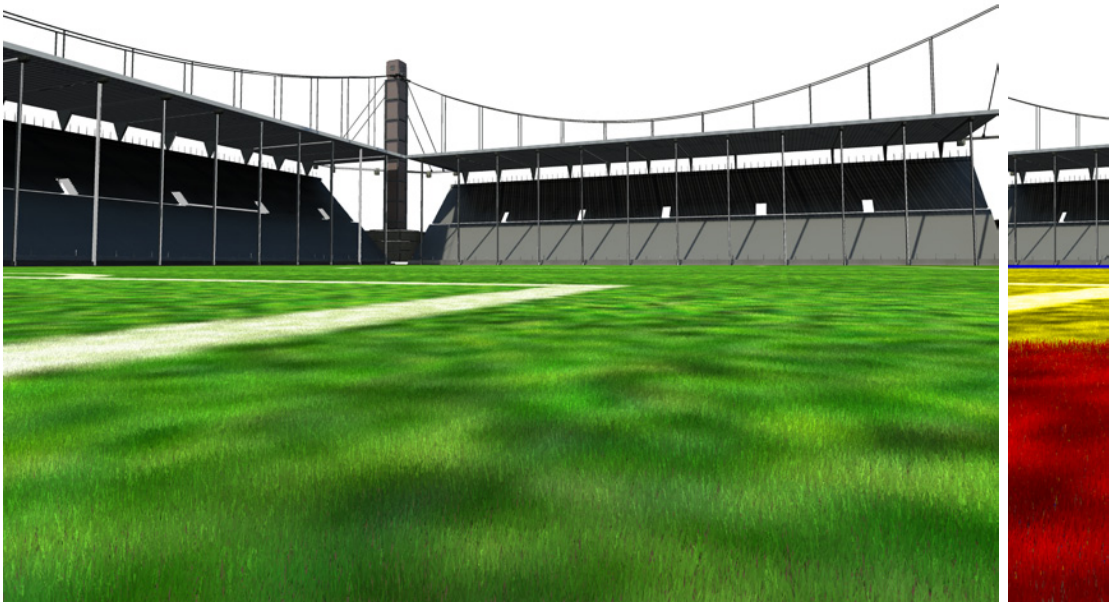

Figure 7.2.: *The football stadium rendered from a ground-near viewpoint at player's height.*

Figure 7.3.: *The football stadium rendered from a point of view within the grass.*

13	75.74 ms	0.48 ms	39.88 ms	33.07 ms	0.99 ms	2.08 ms	104	51	2	↑
fps	t_a	t_q	t_g	t_v	t_i	t_m	c_g	c_v	c_i	
21	46.70 ms	0.48 ms	19.39 ms	23.65 ms	1.23 ms	2.35 ms	46	59	5	↓

Figure 7.4.: *The football stadium rendered from a viewpoint within the grass.*

sum of the separated rendering times and the overall measured rendering time t_a, which is caused by an additional parallelization of the different rendering methods on the graphics hardware. Thus, we measured each rendering method separately to determine the actual computational effort.

The terrain surface, observed from a high point of view, is mainly covered with image-based grass due to the large distance to the camera (Figure 7.1). The usage of this very efficient rendering method leads to the high frame rate about 177 fps. Another common viewpoint to see the terrain surface is at player's height above the ground (Figure 7.2). The quality of this rendering result benefits from the very detailed geometry-based representation (red). The computational effort of this high-quality rendering method shows up as a significantly increased rendering time, which mainly results from the high rendering time t_g and t_v of the geometry-based and volume-based methods. This merely leads to a frame rate about 15 fps, which would just be sufficient for interactive applications. Figure 7.3 shows the grass from the perspective of an ant. The whole screen is covered with comparatively few geometry-based grass blades. Besides, the other grass representations in the background and the remaining part of the geometry-based representation are almost invisible but still drawn. There is a huge waste of unnecessary computations due to the dense geometry-based grass occupying with just a few blades the majority of the screen. Many fragments are processed for the final image, which are probable not visible. This results in a further increased rendering time with a low frame rate about 13 fps.

Since this is insufficient for real-time performance, we adjust the level-of-detail settings to match our hardware capabilities and reduce the range of the zone of the geometry-based grass representation from 20.0 down to 12.0. The change solely affects the quality of renderings from a ground-near viewpoint. Hence, we take a further measurement from the perspective of a football player (Figure 7.4). The result is still of high quality with a slightly decreased range of geometry-based grass representation but the amount of geometry-based patch instances is with 46 much more than halved. Thus, the rendering time t_g decreases from 41.44 ms to 19.39 ms resulting in an overall frame rate of 21 fps. A second measurement from a viewpoint directly above the ground, such as Figure 7.3, shows no difference in quality rather than an increased frame rate about 18 fps.

As an alternative, we decrease the zones of the volume-based grass representation in the same way. This leads to a marginal reduction of the rendering time t_v to 35.03 ms due to the fact that the covered screen area with this low-view angle nearly stays the same. Since the low view angle requires an intersection of many vertical slices, this results in a high amount of ray loop iterations in the fragment shader as described in Section 5.2. Thus, we rather reduce the upper bound of the macro-cell size from 40 to 10 than decrease the zone of volume-based grass with no change in visual quality. This significantly decreases the rendering time t_v for the volume-based grass representation from 39.88 ms to 26.52 ms for the ant's perspective due to the reduced maximum of loop iterations with an overall frame rate about 15 fps. However, macro-cells below a size of 10 increase the rendering time again with a drop of the frame rate due to the growing amount of patch instances with respective draw calls.

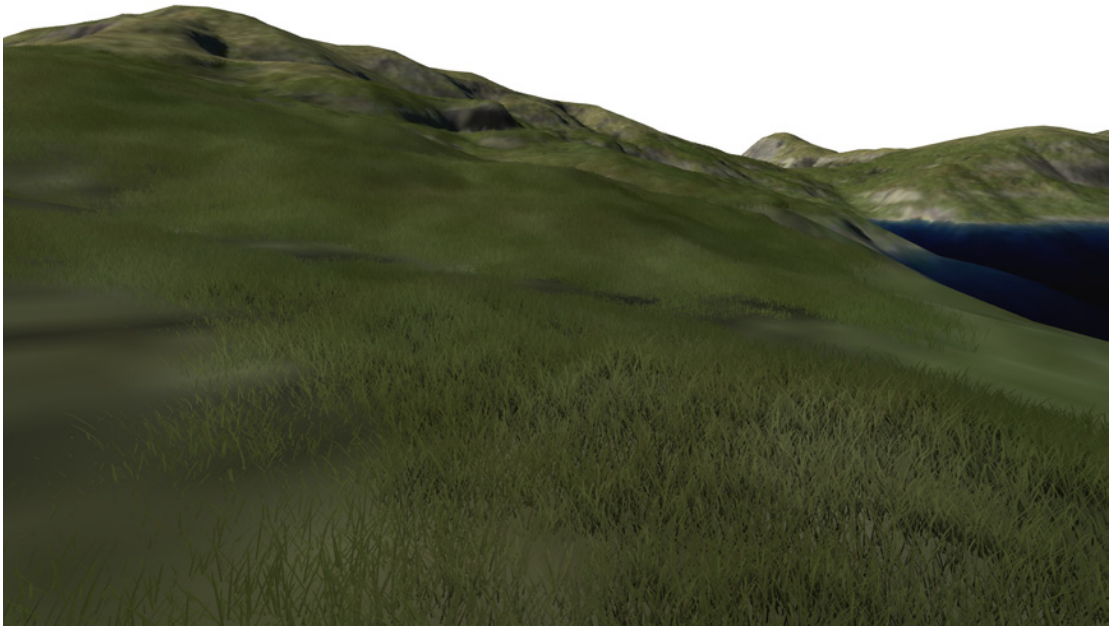

Figure 7.5.: *A landscape model covered with grass.*

Furthermore, we run a sample application to enrich a landscape model with grass. The terrain surface utilizes the excessive dimensions of 1024×1024 cells with a grass patch of 800 blades and is distorted with a large vertical height scale. Moreover, we omit the image-based rendering method due to the high terrain unevenness, which leads to a significantly decreasing of macro-cells and an immense amount of patch instances. Instead, the terrain surface has already an appropriate ground texture applied and we just want to improve the appearance for grass in moderate distance.

The demo application initially overcharges our test system and runs at a low frame rate about 4 fps. Due to the large vertical scale the grass-covered terrain occupies a large screen area, which requires much more fragment shader computations than flat terrain surfaces. Decreasing the range of volume-based grass representation achieves real-time performance with 15 fps on our test system through the reduced workload of the fragment shader. Moreover, we measured the landscape model with the full range of volume-based grass representation at a decreased vertical height scale to a quarter. This leads to more than a doubled frame rate of 10 fps and requires a smaller decrement of the range of volume-based grass representation.

Furthermore, we run the sample application with the full desired range of volume-based grass and the large vertical height scale using a more up-to-date graphics card. A NVIDIA 560 GTX powered our sample application at a frame rate about 21 fps showing 107 patch instances of geometry-based and 224 patch instances of volume-based grass representations (Figure 7.5). We confirm the performance increase due to the increased number of shader cores on this graphics hardware with a second measurement on this test system, which yields similar results (Figure 7.6).

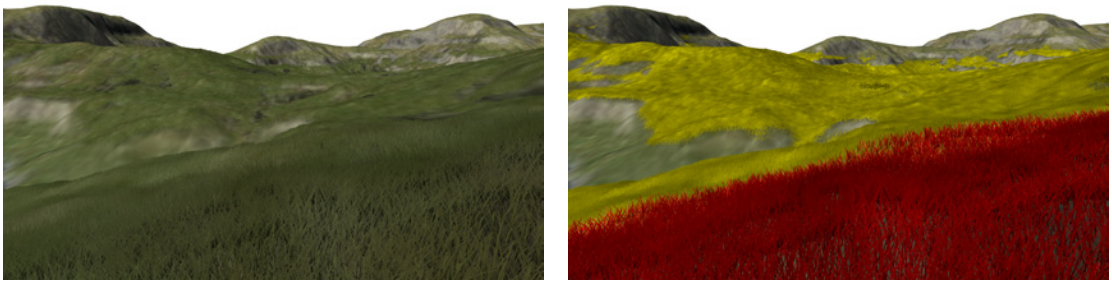

Figure 7.6.: *A landscape model covered with grass (left) and the utilized rendering techniques (right).*

7.1. Visual Quality Evaluation

We evaluated few aspects of visual quality of grass appearance. The most important are parallax effect, a plausible illumination, and the occlusion of other scene objects located on the terrain surface. All these parameters significantly affect the overall appearance of grass. A high visual quality of these aspects produces reliable renderings of grass.

7.1.1. Parallax Effect

When moving through grass fields, the parallax effect of the individual grass blades is important for the reliability of the renderings. This effect creates the illusion of depth to the coat of grass. The viewer receives the impression of a vivid material covering the terrain surface. Figure 7.7 shows the different representations of the grass from varying view points. It is noticeable that the parallax effect of the geometry-based

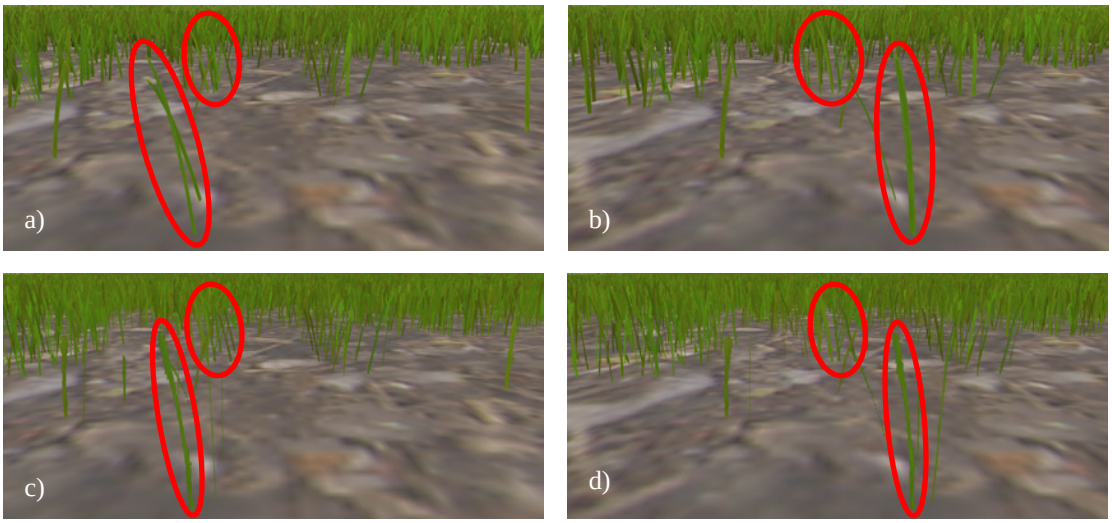

Figure 7.7.: *The grass is rendered from slightly varying viewpoints to show the parallax effect of the geometry-based representation (a,b) and the volume-based representation (c,d).*

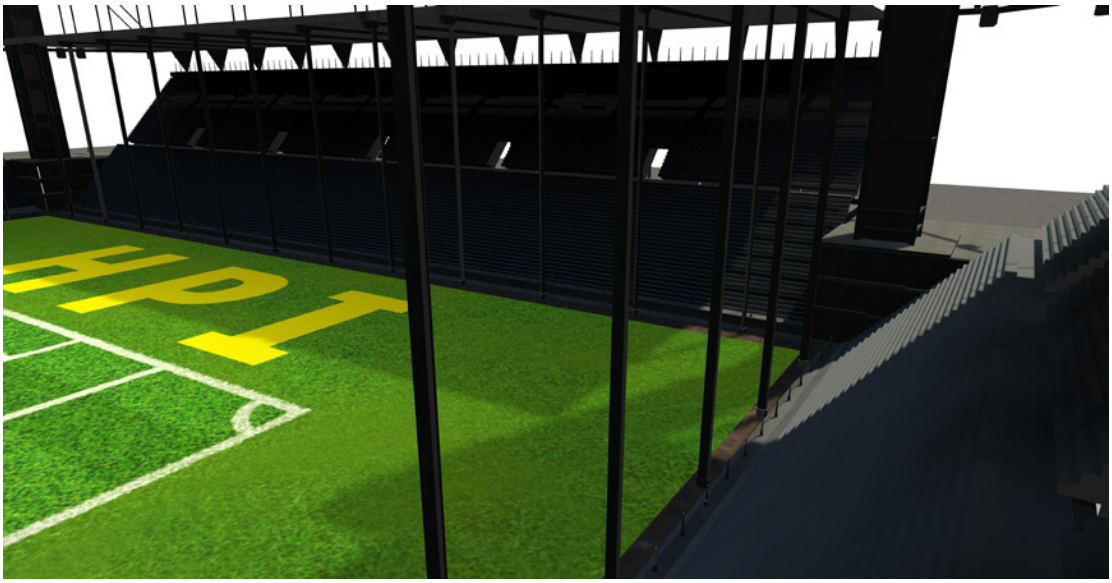

Figure 7.8.: *The shadow, which is casted by the stands of the stadium, affects the grass covered football field. This improves the integration of the rendered grass into the surrounding scenery.*

grass representation is best. The volume-based representation shows also a plausible parallax effect for grass blades in different slices. The image-based representation lacks the parallax effect due to the flatness of this representation.

7.1.2. Illumination

A plausible and consistent illumination of the grass-covered terrain, which is appropriate for the surrounding scenery, is essential for reliable, photo-realistic renderings of grass. Since the surrounding scenery affects the light transmission to the grass surface, we utilize light maps to customize the varying ambient and diffuse light intensities of the directional light source. This results in an illumination with a high visual quality with the ability to apply shadowing effects (Figure 7.8).

Each representation of grass is affected by the same illumination model. Therefore, normal maps are utilized in the volume-based and image-based rendering techniques because the vertical and horizontal slices differ from the aggregated grass blade geometry. This results in a very consistent illumination of grass and maintains the overall visual quality of a photo-realistic rendering.

7.1.3. Occlusion of Other Scene Objects

The three representations of grass create distinct results with regard to the occlusion of other scene objects (Figure 7.9). The geometry-based grass representation produces best results due to the separately modeled grass blades and their representation as quadrilateral strip. The volume-based grass representation leads to similar results as the geometry-based. In contrast to that, the image-based representation of grass does not support a reliable occlusion. Due to the flat structure of this representation, the edges to other scene objects are unpleasantly straight.

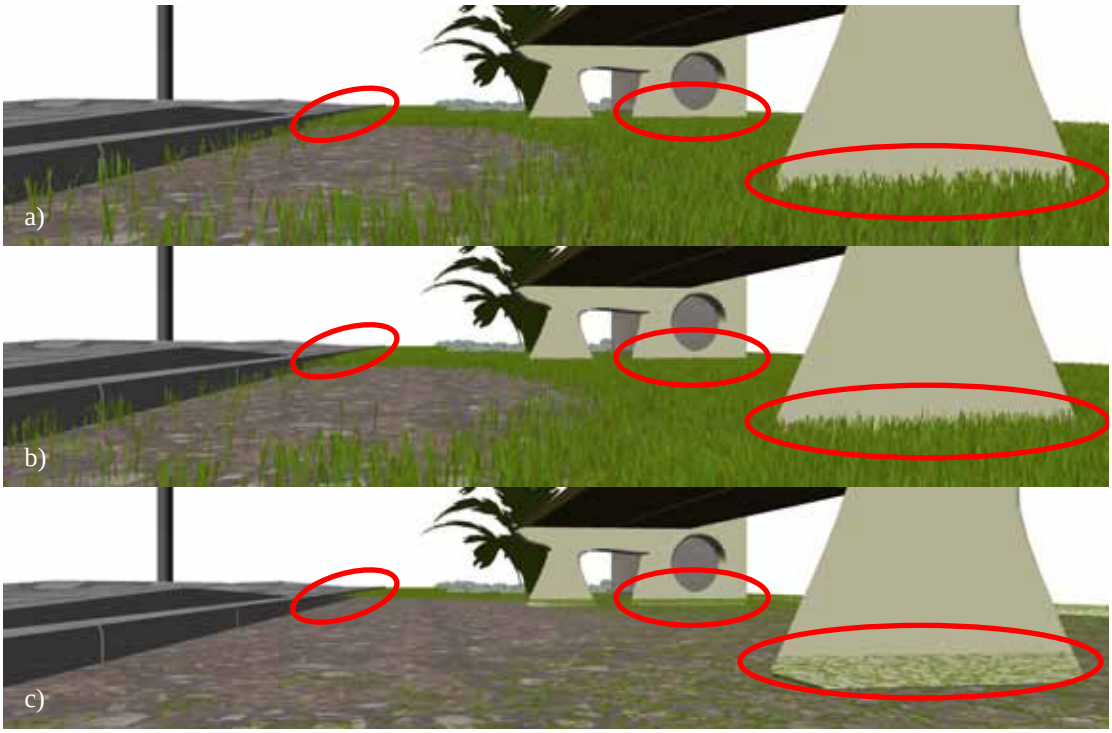

Figure 7.9.: *The occlusion of other scene objects is essential for the reliability of grass. The geometry-based (a) and volume-based (b) representations create plausible occlusions and a characteristic silhouette of grass. The image-based representation (c) lacks of plausible occlusion.*

Since we use a different rendering technique for the volume-based grass representation compared to Boulanger (2008), we are able to apply it in larger distances to the camera. This results in occlusions of objects in the distance with the same high visual quality as with the geometry-based representation.

7.2. Performance Evaluation

We already discussed the performance of the rendering system in two sample applications. Furthermore, we measured the computational effort of each rendering technique separately to estimate a ratio for the trade-off between quality and performance.

We utilize a scene setting to fill an equal-sized area with grass using the different rendering techniques (Figure 7.10). The terrain consists of 32×32 cells and uses aperiodic tiling, an illumination model and no displacement. The rendering time of the additional scene geometry and the traversal of the quad tree is measured with an average of 1.8 ms and excluded from the rendering times of the respective technique.

7.2.1. Comparison of the Rendering Techniques

We deactivated the usage of macro-cells. Thus, the terrain is filled with 1024 cells and the patch instance are all of basic size. The generated geometry patch contains 2400 grass blades with four subdivisions, which leads to 24 thousand triangles for one

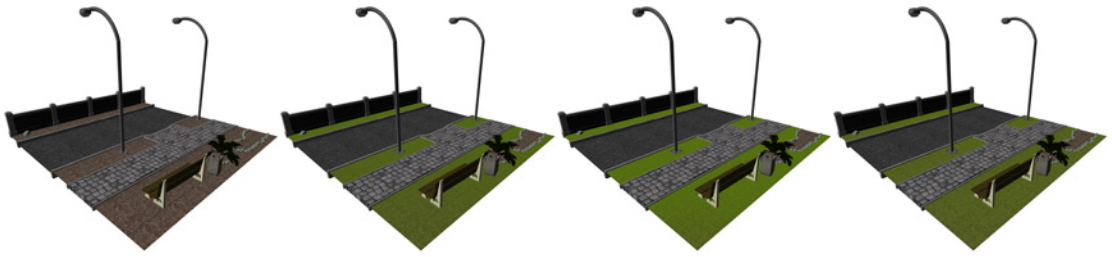

Figure 7.10.: *The terrain of the same scene (a) is covered with grass by the geometry-based rendering technique (b), the volume-based rendering technique (c) and the image-based rendering technique (d).*

geometry-based patch instance. The volume-based grass representation utilizes 16 slices in each direction with a texture resolution of 1024×128 texels and the image-based representation with a resolution of 1024×1024 texels. The results of the computational effort to fill the terrain with grass are shown in Table 7.1.

	Triangles	Rendering time
Geometry-based method	24.6 mio.	144.84 ms
Volume-based method	6144	22.45 ms
Image-based method	2048	2.24 ms

Table 7.1.: *Measurements of the computational effort for each rendering technique with the same amount of 1024 patch instances.*

The performance increase of the volume-based rendering technique over the geometry-based rendering technique is 545 percent with almost the same high visual quality at this distance. Furthermore, the performance increase of the image-based rendering method over the volume-based rendering technique is 900 percent. As a consequence, a shift of the level-of-detail zones to use less geometry-based grass representation and more volume-based grass representation or less volume-based grass representation and more image-based representation results in a significant performance improvement.

7.2.2. Macro-Cells

Since the volume-based and image-based rendering techniques allow the usage of macro-cells to reduce the amount of patch instance to fill the terrain, we measured the effect on the performance for different sizes of macro-cells. The results are shown in Table 7.2. As an additional reference, the rendering times for the patch instances of basic size are given.

Macro-cell size	1	2	4	8	16
Number of cells	1024	256	64	16	4
Volume-based method	22.45 ms	12.2 ms	8.65 ms	6.31 ms	6.02 ms
Image-based method	2.24 ms	0.94 ms	0.50 ms	0.42 ms	0.41 ms

Table 7.2.: *Measurements of the computational effort for the volume-based and image-based rendering technique with different sizes of macro-cells.*

The measurements confirm that in general the usage of macro-cells significantly improves the performance. In particular, the performance of the image-based rendering technique is always improved by larger macro-cells. The volume-based rendering technique benefits from macro-cells as long as the amount of patch instances is the bottle-neck. Since macro-cells reduce the number of patch instances, the performance of the volume-based rendering technique is improved as well. However, as mentioned earlier, the workload of the fragment shader increases with the size of macro-cells.

7.3. Limits of the Rendering Framework

A few problems related to the grass rendering framework are outstanding and may limit the application in some use cases.

We dismiss the translucency of grass. Our illumination model is a simplification and omits the translucency. We simulate the translucency of grass by an increased amount of ambient light. Since the illumination model is replaceable, the integration of an appropriate illumination model with regard to the translucency of grass is possible.

The support of the image-based rendering technique of rough terrain surfaces is limited due to a large vertical height scale causing a small size of the macro-cells. This results in a high amount of patch instances. A probable solution would be to shift horizontal slice upwards for cells with a higher terrain unplanarity.

Using the volume-based rendering technique for grass in the distance may cause aliasing artifacts due to a cell size of less than one pixel. This is solved by increasing the basic size of the cells or reducing the range of volume-based grass representation since the image-based one does not show any aliasing artifacts.

Additionally, particular angles of view and a low amount of vertical slices show the grid structure of the volume-based grass representation.

Chapter 8

Conclusions

This thesis presents a grass-rendering framework for real-time applications. The described rendering approach produces detailed renderings of grass offering a high visual quality. The rendering techniques utilize three different representations of grass composed in a level-of-detail scheme in order to achieve real-time performance. The presented system is able to cover large terrain surfaces with grass and supports several features to portray an almost natural-like behavior of grass, such as the non-uniform distribution on a terrain surface. Various customization possibilities are provided to adapt the appearance of the rendered grass with regard to an overall picture. The coloring of the grass and the distortion of the grass covered terrain surface serve as examples. These components enable the framework to enrich photo-realistic sceneries with grass of high visual quality, especially with regard to virtual 3D city and landscape models, which benefit from grass as an essential part of their vegetation. The reliability of such renderings increases significantly due to human's wide range of visual experience with natural grass.

Main Contributions

The presented rendering method overcomes the immense geometric complexity of grass. A football field with more than one billion virtual grass blades is rendered at interactive frame rates. Furthermore, the presented technique offers an approach to enhance the appearance of grass growth in a more natural-like manner, which is done by applying a clustering effect to imitate the behavior of new grass blades initially growing near existing ones. In this way, the reliability of areas on the terrain with a low grass density is significantly increased. Further, an alternative volume-rendering method for volume-based grass representation is presented as well. This method utilizes the capabilities of modern graphics hardware to perform ray tracing in the fragment shader stage of the rendering pipeline. Finally, the results of our approach are discussed with regard to quality and performance as well as the benefits and drawbacks of that method.

Implementation

The grass rendering framework is prototypically implemented and encapsulated in a software library. The implementation requires OpenGL support in the version 3.3 or higher. The framework consists of three modules to

- generate natural-like representations of grass,

- render them with state-of-the-art rendering techniques and
- customize the appearance through the design maps of the terrain management.

The framework provides the ability to integrate grass rendering support into other rendering engines. In addition to this, it offers a great extensibility so that new grass representations can be easily added to the framework and used in the level-of-detail scheme. Alternative renderer implementations can be added as well to examine the capabilities of uprising technology for the rendering of grass.

Future Work

There is a multitude of issues that can be addressed in future work. The essentials are briefly outlined here:

- **Animations.** As a matter of fact, grass is not static. Several environmental factors affect the movement of grass blades. Modeling the influence of wind using physical-based precomputations was already done by Perbet & Cani (2001). By utilizing an animation map, the effect of treading on grass was considered as well by Guerraz et al. (2003). Nonetheless, the quality of the visual presentation of the movements is still limited due to the support of rendering techniques. There is a lack of efficient rendering techniques representing animations in high visual quality.
- **Hardware capabilities.** The presented rendering techniques already take into account latest hardware capabilities. However, there are a few more to further optimize the implementations like the performance of the ray tracing implementation being bound to the work load of the fragment shader. Therefore, a shift of the computations to other stages in the rendering pipeline performing pre-calculations is desirable. Moreover, a renderer implementation that benefits from geometry instancing is possible. The lack of the image-based rendering technique on distorted terrain surfaces could be solved by using the features of hardware-accelerated tessellation.
- **Authoring.** The framework allows for the customization of grass to perfectly fit into the desired result. The customizations are provided by design maps that are previously created with 3rd-party tools. There are several use cases, where an almost automated adding of grass into the existing scene would be beneficial. A set of tools is imaginable to later modify and customize the grass inside the scenery. Procedural algorithms supporting the user with this task could be useful.
- **Ground-level Vegetation.** An extension of the proposed framework to support not only grass but rather all types of ground-level vegetation is feasible. This can be flowers common on meadows, such as dandelion or daisies, or other ground-cover plants, such as ivy.
- **Adaptive Rendering System.** The framework provides a broad range of performance screws to adapt the rendering speed to the hardware capabilities. Rather than manually tweak these parameters to achieve best quality on a given hardware

platform, an adaptive system is desirable to automatically adjust the rendering parameters for a stable, interactive frame rate.

Outlook

In near future, the software of the grass rendering framework will be publicly available with an appropriate open-source compatible license. A further development beyond this thesis will be oriented on the prospective requirements with regard to proposed future work. By now, the probation in various real-world application cases is outstanding to retrieve an external validation of the provided grass rendering framework.

Appendix A

Additional Non-Standard Functions

$$\mathsf{max}\,(a,b) = \begin{cases} a & \text{if } a \geq b \\ b & \text{otherwise} \end{cases} \tag{A.1}$$

$$\mathsf{min}\,(a,b) = \begin{cases} a & \text{if } a \leq b \\ b & \text{otherwise} \end{cases} \tag{A.2}$$

$$\mathsf{abs}\,(x) = |x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise} \end{cases} \tag{A.3}$$

$$\mathsf{clamp}\,(x,a,b) = \mathsf{max}\,(\mathsf{min}\,(x,b),a) \tag{A.4}$$

$$\mathsf{step}\,(e,x) = \begin{cases} 1 & \text{if } e \leq x \\ 0 & \text{otherwise} \end{cases} \tag{A.5}$$

$$\mathsf{linearstep}\,(a,b,x) = \mathsf{clamp}\left(\left(\frac{x-a}{b-a}\right),0,1\right) \tag{A.6}$$

$$\mathsf{smoothstep}\,(a,b,x) = 3 \cdot \mathsf{linearstep}\,(a,b,x)^2 - 2 \cdot \mathsf{linearstep}\,(a,b,x)^3 \tag{A.7}$$

$$\mathsf{ceil}\,(x) = \lceil x \rceil = \min\{k \in \mathbb{Z} | k \geq x\} \tag{A.8}$$

$$\mathsf{sign}\,(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ 0 & \text{otherwise} \end{cases} \tag{A.9}$$

All of the above defined functions support arguments of vectors as well. They are applied component-wise to vector-type arguments and return the result as a vector with the same dimension such as:

$$\mathsf{clamp}_{vec2}\left(\begin{bmatrix} x \\ y \end{bmatrix},a,b\right) = \begin{bmatrix} \mathsf{clamp}\,(x,a,b) \\ \mathsf{clamp}\,(y,a,b) \end{bmatrix} \tag{A.10}$$

$$\mathsf{nextPowerOfTwo}\,(x) = 2^{\mathsf{ceil}(\log_2 x)} \tag{A.11}$$

$$\mathsf{normalize}\,(\vec{x}) = \frac{1}{\|\vec{x}\|} \cdot \vec{x} \tag{A.12}$$

$$\mathsf{mat3}\left(\begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix}\right) = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix} \tag{A.13}$$

Appendix B

Software Framework

B.1. Packages

- **ng_core**
  The core library of the grass rendering framework. The grass representations, the renders and the terrain management is implemented here.
- **ng_generation**
  The generation library provides the algorithm to generate geometry-based grass patches.
- **viewer**
  This tool shows the three representations of grass in place to compare them.
- **designer**
  This tool is a grass patch designer. The parameters for several grass patches can be examined with the tool.

B.2. Dependencies

- CMake (2.8 or higher)
- Boost (1.46 or higher)
- GLM (0.9.2.4 or higher)
- GLEW (1.7.0 or higher)
- GLFW (2.7.2 or higher)
- Qt (optional, 4.6.2 or higher)
- CodeSynthesis XSD (3.3 or higher)
- Xerces (3.1.1 or higher)

List of Figures

Bibliography

Akenine-Möller, T., Haines, E., & Hoffman, N. (2008). *Real-time Rendering*. Natick, MA, USA: A. K. Peters, Ltd., 3nd edition.

Bakay, B. (2003). Animating and lighting grass in real-time. M. sc. thesis, University of British Columbia.

Bakay, B., Lalonde, P., & Heidrich, W. (2002). Real-time animated grass. In *Proceedings of Eurographics (short paper)*.

Banisch, S. & Wüthrich, C. A. (2006). Making grass and fur move.

Blinn, J. F. (1978). Simulation of wrinkled surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '78 (pp. 286–292). New York, NY, USA: ACM.

Boulanger, K. (2008). *Real-time Realistic Rendering of Nature Scenes with Dynamic Lighting*. PhD thesis, College of Engineering and Computer Science, University of Central Florida, Orlando, Florida.

Boulanger, K., Pattanaik, S., & Bouatouch, K. (2006). Rendering grass terrains in real-time with dynamic lighting. In *ACM SIGGRAPH 2006 Sketches*, SIGGRAPH '06 New York, NY, USA: ACM.

Boulanger, K., Pattanaik, S. N., & Bouatouch, K. (2009). Rendering grass in real time with dynamic lighting. *IEEE Computer Graphics and Applications*, 29, 32–41.

Castaño, I. (2010). Computing alpha mipmaps. http://the-witness.net/news/2010/09/computing-alpha-mipmaps/. [Online; accessed Sep-02-2012].

Clark, J. H. (1976). Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10), 547–554.

Dietrich, A., Colditz, C., Deussen, O., & Slusallek, P. (2005). Realistic and interactive visualization of high-density plant ecosystems. In P. Poulin & E. Galin (Eds.), *NPH* (pp. 73–81).: Eurographics Association.

Eberly, D. (2002). Intersection of a sphere and a cone. http://www.geometrictools.com/Documentation/IntersectionSphereCone.pdf. [Online; accessed Sep-02-2012].

Guerraz, S., Perbet, F., Raulo, D., Faure, F., & Cani, M.-P. (2003). A procedural approach to animate interactive natural sceneries. In *Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)*, CASA '03 (pp. 73–). Washington, DC, USA: IEEE Computer Society.

Habel, R. (2009). *Real-time Rendering and Animation of Vegetation.* PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria.

Habel, R., Wimmer, M., & Jeschke, S. (2007). Instant animated grass. *Journal of WSCG*, 15(1-3), 123–128. ISBN 978-80-86943-00-8.

Heckbert, P. S. (1986). Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11), 56–67.

Jain, A. K. (1989). *Fundamentals of digital image processing.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

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

Maciel, P. W. C. & Shirley, P. (1995). Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, I3D '95 (pp. 95–ff.). New York, NY, USA: ACM.

Maillot, J., Yahia, H., & Verroust, A. (1993). Interactive texture mapping. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (pp. 27–34). New York, NY, USA: ACM.

McAllister, D. K., Lastra, A. A., Cloward, B. P., & Heidrich, W. (2002). The spatial bi-directional reflectance distribution function. In *ACM SIGGRAPH 2002 conference abstracts and applications*, SIGGRAPH '02 (pp. 265–265). New York, NY, USA: ACM.

Meyer, A. & Neyret, F. (1998). Interactive Volumetric Textures. In G. Drettakis & N. Max (Eds.), *Eurographics Workshop on Rendering techniques (Rendering Techniques'98)* (pp. 157–168). Vienna, Autriche: Eurographics Springer Wein.

Müller, G., Meseth, J., Sattler, M., Sarlette, R., & Klein, R. (2004). Acquisition, synthesis and rendering of bidirectional texture functions. In C. Schlick & W. Purgathofer (Eds.), *Eurographics 2004, State of the Art Reports* (pp. 69–94).: INRIA and Eurographics Association.

Neyret, F. (1998). Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1), 55–70.

NVIDIA Corporation (2004). *Anti-Aliasing with Transparency.* Technical report, NVIDIA Corporation. http://http.download.nvidia.com/developer/SDK/Individual_Samples/DEMOS/Direct3D9/src/AntiAliasingWithTransparency/docs/AntiAliasingWithTransparency.pdf. [Online; accessed Sep-02-2012].

Pelzer, K. (2004). Rendering countless blades of waving grass. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics* (pp. 107–121). Amsterdam, NL: Addison-Wesley.

Perbet, F. & Cani, M.-P. (2001). Animating prairies in real-time. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, I3D '01 (pp. 103–110). New York, NY, USA: ACM.

Perlin, K. (2002). Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02 (pp. 681–682). New York, NY, USA: ACM.

Picco, D. (2003). Frustum culling. http://www.flipcode.com/archives/Frustum_Culling.shtml. [Online; accessed Sep-02-2012].

Policarpo, F., Oliveira, M. M., & Comba, J. a. L. D. (2005). Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05 (pp. 155–162). New York, NY, USA: ACM.

Ramamoorthi, R. & Hanrahan, P. (2001). An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01 (pp. 497–500). New York, NY, USA: ACM.

Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2), 187–260.

Shah, M. A., Kontinnen, J., & Pattanaik, S. (2005). Real-time rendering of realistic-looking grass. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '05 (pp. 77–82). New York, NY, USA: ACM.

Szirmay-Kalos, L. & Umenhoffer, T. (2008). Displacement mapping on the gpu - state of the art. *Computer Graphics Forum*, 27(6), 1567–1592.

Wang, H. (1960). Proving theorems by pattern recognition i. *Commun. ACM*, 3(4), 220–234.

Whatley, D. (2005). Toward photorealism in virtual botany. In *GPU Gems 2: Techniques for Graphics and Compute-Intensive Programming* (pp. 7–25). Amsterdam, NL: Addison-Wesley.

Williams, L. (1983). Pyramidal parametrics. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '83 (pp. 1–11). New York, NY, USA: ACM.

Zhao, X., Li, F., & Zhan, S. (2009). Real-time animating and rendering of large scale grass scenery on gpu. In *Proceedings of the 2009 International Conference on Information Technology and Computer Science - Volume 01*, ITCS '09 (pp. 601–604). Washington, DC, USA: IEEE Computer Society.