Instant Animated Grass

Ralf Habel Michael Wimmer Stefan Jeschke
Institute of Computer Graphics and Algorithms
Vienna University of Technology, Austria
{habel,wimmer,jeschke}@cg.tuwien.ac.at

ABSTRACT

This paper introduces a technique for rendering animated grass in real time. The technique uses front-to-back compositing of implicitly defined grass slices in a fragment shader and therefore significantly reduces the overhead associated with common vegetation rendering systems. We also introduce a texture-based animation scheme that combines global wind movements with local turbulences. Since the technique is confined to a fragment shader, it can be easily integrated into any rendering system and used as a material in existing scenes.

Keywords
Real-time Rendering, Natural Phenomena, Natural Scene Rendering, GPU Programming

1. INTRODUCTION

Interactive rendering of vegetation in natural scenes plays an important role in virtual reality and computer games where grass is an essential part of most natural scenes. Unfortunately, grass is also very complex: modeling each individual blade of grass would require a huge amount of geometry, making it impossible to render in real time. Common acceleration techniques represent grass using billboards [Pel04]. However, even these simplified billboards lead to massive overdraw in realistically modeled scenes. Furthermore, placing grass into a scene using geometry requires a significant storage and modeling effort.

In this paper, we present a new method to render animated grass in real time that exhibits all important visual characteristics of grass, namely parallax and occlusion effects when the viewpoint moves, as well as animation due to wind. It is efficient to render and easy to incorporate into existing rendering systems. The main target are video games with a first person viewpoint where grass is mostly seen at grazing angles. Grass is represented using implicitly defined textured billboards perpendicular to the terrain geometry which are ray traced in a fragment shader and is therefore suited for short and dense grass such as lawns or meadows. While the billboards could also be defined using conventional geometry, the advantage of an implicit definition is that no extra geometry has to be generated, only the terrain geometry and its associated tangent space is required as input. This makes it easy to apply the grass to different scenes and the output speed merely depends on the number of pixels covered by grass rather than on the density of the grass or the extent of the terrain. Standard lighting techniques such as dynamic lighting and shadowing, including light maps or precomputed radiance transfer can be used without modification. Furthermore, we animate grass using a texture based approach that incorporates both low-frequency phenomena like gusts of wind and high-frequency phenomena like small turbulences, leading to a very realistic appearance of the rendered grass.

Figure 1: A terrain textured with animated grass with moderate grass density and height.

2. RELATED WORK

In order to display complex volumetric effects such as fur and short hair, Kajiya and Kay [KK89] introduced volumetric textures, called "texels". In this context, texels are representations of a three-dimensional material by a cubic reference volume that is mapped onto a surface repeatedly. A texel itself is a three-dimensional array approximating the visual properties of a micro-surface. They were created to solve the problem of spatial aliasing of ray-traced complex geometries. Rendering a texel involves front-to-back compositing along a ray, which is also used in our method. An extension and application to natural scenes of volumetric textures was presented by Neyret [Ney98]. The typical real-time implementation of texels uses stacks of polygons, mapped with semi-transparent textures [BH02] [Len00] [LPFH01] [MN98] . However, slices that are parallel to a terrain geometry are not optimal for viewing positions typical for walkthroughs, with objectionable artifacts at grazing angles.

The most common way to represent grass (also used in many current games) are billboards mapped with a texture of several grass blades [Pel04]. The billboard vertices are usually animated analytically. However, a massive amount of polygons is required to densely cover a terrain, and analytical animation looks very uniform.

Perbet and Cani [PC01] combine different grass representations at different distances to render and animate prairies in real time. In the nearest level of detail, grass blades are modeled individually, which makes it difficult to combine various types of grass and flowers.

The proposed algorithm is closely related to real-time relief mapping [POaLDC05] [OP05] [WWT$^+$03]. In contrast to those methods, which are based on ray tracing a height field within a shell on the surface of an object, the presented method ray traces a regular grid.

3. RENDERING GRASS

Motivation

We model grass as a collection of textured billboards, and arrange them in a regular grid.

Typically, a grass texture is fully transparent between the individual grass blades and fully opaque within the blades. However, partial opacity arises at the edges of the grass blades if the grass texture is a filtered version of a higher resolution texture, or if it has been generated using an anti-aliased renderer in the first place. Therefore, the colors and opacities of billboards overlapping in screen space need to be correctly composited. Just as in volume rendering, this can be done either in back-to-front or front-to-back fashion [LL94].

Back-to-front compositing corresponds to standard transparency alpha blending used when rendering the billboards as geometry. However, back-to-front compositing can be very inefficient because all slices have to be traversed *in order* to get a correct result. Furthermore, if the billboards intersect each other, a consistent back-to-front order does not exist. The popular alternative of using alpha testing instead of alpha blending leads to noticeable aliasing artifacts especially at the edges of the grass blades.

Front-to-back compositing, on the other hand, is typically used with ray tracing and allows for early ray termination when the accumulated opacity is sufficiently high. We exploit this fact for grass rendering in the following way: Instead of rendering the textured grass billboards using polygons, we define them implicitly on a "carrier polygon" and ray trace these "virtual billboards"' in the fragment shader using front-to-back compositing (also known as the "over"-operator [PD84]). This allows exiting the fragment shader when the opacity reaches a user-defined threshold. Furthermore, intersecting billboards are handled automatically, always giving correct compositing results. We have found that the illusion of grass can be perfectly maintained even when doing a small, fixed number of iterations, which is more amenable to current graphics hardware.

The setup of the ray tracing step is very similar to relief mapping [POaLDC05], where a height map, defined in a shell carried by polygons is ray traced in the fragment shader. As with relief mapping, the regular grid of grass billboards therefore seem to reside inside the carrier polygon (see Figure 2).

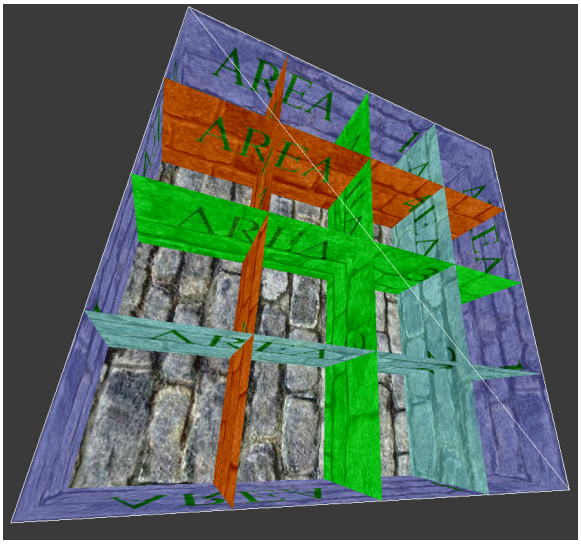

Figure 2: A quad patch (wireframe overlay) rendered with fully opaque textures. The grid structure is generated in the fragment shader.

The most significant advantage of rendering grass in the fragment shader may be the ease of modeling and

integration into existing rendering systems. Grass can be defined as a material and does not require any other change in the scene definition (whereas in polygonal rendering, each grass billboard has to be placed either by hand or automatically). Furthermore, we will show how to animate the ray traced grass in high quality.

Grass Ray Tracer

This section describes the grass ray tracer in more detail. A basic grass patch consists of the ground texture and a texture containing one subtexture for each virtual billboard (or slice) in the patch. We currently use the same set of billboard textures for both axes of the regular grid. For current graphics hardware, we allow the raytacing loop to exit before full opacity has been reached. The remaining opacity can be filled using a constant color or an additional, fully opaque grass slice (Figure 4). The fragment shader casts rays into a shell defined by the carrier polygon at the top, and a virtual ground polygon at the bottom that is offset by a user-defined distance along the negative tangent-space w axes at the vertices (i.e., the inverted normal vectors).

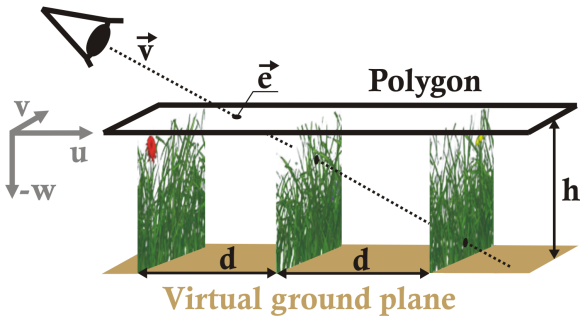

Figure 3: A ray is cast from the viewing point through grass slices.

Grass can be applied to any mesh in a scene that has a tangent space (u,v,w) defined. A basic grass patch is tiled onto the whole mesh. Note that this leads to similar restrictions as with relief mapping, where silhouettes are difficult to define. Additionally, analogous to relief mapping, the viewpoint cannot move into the grass.

The fragment shader takes as input the interpolated tangent space vectors, the view vector $\vec{v}$ in tangent space (interpolated from $\vec{p}-\vec{s}$ at each vertex $\vec{p}$ and viewpoint $\vec{s}$), and the interpolated texture coordinates (which give the ray entry point $\vec{e}$ see Figure 3). The user also has to provide the parameters $d_{u,v}$ for the distance between the slices in tangent space and the depth of the ground plane h. The shader executes the following steps:

1. Calculate for both u and v a texture offset to select the initial grass slices.

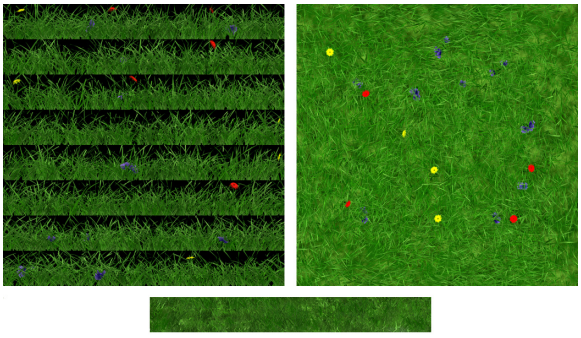

Figure 4: A grass data set consisting of grass blades (left), a ground texture (right) and a fully opaque grass slice (bottom). Note that as in any texture packing method, a one-texel border needs to be observed between grass slices.

2. Adjust this offset depending on the sign of the view vector so the same slice is seen from both sides.

3. Calculate the positions $p_{u,v}$ of the first planes to be ray traced in both u and v directions according to $d_{u,v}$ using a `floor()` operator.

4. Enter the raytacing loop.

The inner ray tracing loop consists of the following steps:

1. Calculate the intersections with the next slice in u and v direction. Since the slices are axis aligned, the ray-plane intersection

$$\vec{x} = \vec{e} + \vec{v} \cdot \frac{\vec{n}_p \cdot (\vec{p} - \vec{e})}{\vec{n}_p \cdot \vec{v}}, \qquad (1)$$

where $\vec{n}_p$ is the normal vector and $\vec{p}$ is an arbitrary point on the plane, simplifies to

$$\vec{x} = \vec{e} + \vec{v} \cdot \frac{p_{u,v,w} - e_{u,v,w}}{v_{u,v,w}}, \qquad (2)$$

depending on which axis is used.

2. Choose the closer intersection point and increment (or decrement, depending on the sign of v) the corresponding slice by $d_{u,v}$.

3. Test intersection point against the virtual ground polygon. If the intersection is outside the shell, intersect the ray with the ground polygon using equation 2.

4. Composit the current color $\vec{c}$ with the color of the slice $\vec{c}_i$ (with associated alpha values α and α_i) using the standard "over" blending function,

which assumes that colors are premultiplied with their corresponding opacity values:

$$\begin{aligned} \vec{c} &= \vec{c} + (1-\alpha) \cdot \vec{c}_i \\ \alpha &= \alpha + (1-\alpha) \cdot \alpha_i \end{aligned} \quad (3)$$

After the raytracing loop, the remaining transparency is filled with a texture lookup from the fully opaque grass slice or the average color of the grass data set. A single grass patch rendered with the data set of Figure 4 using 16 slices for both u and v axes can be seen in Figure 5. We have found that a very low number (4 was used in the images shown) of ray casting iterations is sufficient for high image quality. This helps to keep the number of required texture reads low.

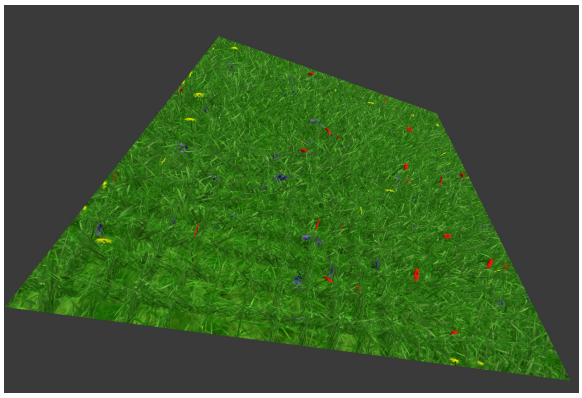

Figure 5: A quad patch rendered with the data set of Figure 4. The grid structure is apparent at perpendicular angles but vanishes at more grazing angles.

Especially for higher grass, if the grass patch is expected to be viewed at perpendicular angles, the grid structure becomes apparent. This can be mostly avoided by adding a horizontal grass slice (in the middle of the shell) which is ray cast just like the vertical slices (Figure 6).

Visibility Interactions

If the grass is to interact with the rest of the scene, visibility with scene objects has to be resolved. Otherwise, the objects will be clipped against the top of the grass (Figure 7). The correct solution would be to render the opaque objects first and generate an offscreen buffer with the corresponding depth information (for example using the multiple render target functionality found in current graphics hardware). When rendering the grass, the depth value at which to terminate a ray can be read from this buffer.

However, this method requires a non-trivial modification of the rendering pipeline, and therefore we opted for a simpler solution. Instead of testing the ray against the current depth buffer, we generate a depth value directly in the fragment shader by calculating the depth

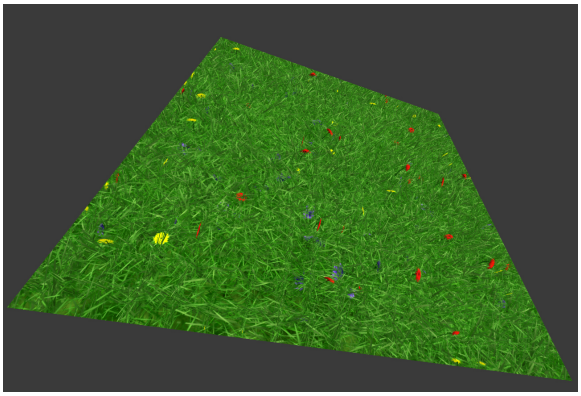

Figure 6: A quad patch with the same data set as in Figure 5 but with an additional horizontal plane at half the ground depth. The grid structure is not dominant even at perpendicular angles.

Figure 7: A grass patch with (left) and without (right) correct visibility.

when a threshold opacity has been reached. Depending on the hardware used, it may prove to be efficient to terminate the ray casting loop through an early out if the threshold opacity is reached. The fragment shader then outputs a depth value determined from the slice distance. This does not require any modification of the rendering pipeline and gives correct occlusion for the fully opaque parts of grass blades. The semi-transparent parts of grass blades are not handled exactly, but the introduced errors are unnoticeable in practice.

4. ANIMATING GRASS

The perceived realism of rendered grass depends greatly on whether it is animated or not. Previous methods to animate grass relied on analytic functions (usually combinations of sines and random perturbations) applied to billboard vertices, which results in fairly simple grass blade movement. A realistic simulation of grass movement has to take two components into account. On the one hand, gusts of wind cause relatively large areas of grass to bend in the same di-

rection. On the other hand, high frequency wind turbulences near the ground cause smaller, but more random movements of grass blades.

In this paper we propose a texture-based animation scheme for grass billboards. Instead of animating the billboard vertices, we distort the texture lookups of the grass billboard horizontally in u or v direction, depending on the billboard orientation. This offset is looked up in a separate noise map that covers the whole mesh and not only an individual grass patch. The offset is scaled with the height above the ground plane of the grass so that at the bottom of the grass billboards stay fixed. The noise map is translated each frame to define the overall wind direction. Although this is a shear operation, with sufficiently small pertubations the impression of moving grass can be maintained. Note that this animation technique works both for standard polygonal billboards as well as for our raytraced virtual billboards.

The advantage of texture-based animation is that any procedural or hand-crafted texture can be used, while the animation over the whole mesh will always remain consistent. The noise texture map used in this paper is a combination of two Perlin noise functions [Per85]. A low-frequency noise function with higher amplitudes simulates gusts of wind, and a high-frequency noise map with lower amplitudes introduces more erratic movements to the grass blades.

5. RESULTS

The proposed method was implemented on a 3.2 GHz Pentium 4 and a GeForce 7900 GT, using DirectX HLSL Shader Model 3.0 and the OGRE [OGR] open source graphics engine. To generate grass slices, the commercial 3D software Maya and its PaintFX features were used (Figure 4). The accompanying video shows a scene with 8×8 grass patches, where each patch contains 16 slices in u and v direction (Figure 1) and a second, denser and shorter grass data set with 32×32 slices in u and v (Figure 8). The grass is animated using the noise map described in the previous section, and visibility with shown polygonal objects is resolved correctly.

The performance of the algorithm depends on the number of pixels covered and on the ray-casting iteration depth. The camera path shown in the video, rendered at a resolution of 1024×768 and an iteration depth of 4, results in an average frame rate of 140 frames per second.

For comparison, we generated a simple scene with grass billboards represented by hand-placed billboard polygons using standard alpha blending. With 32×2 slices per grass patch and 8×8 patches, an average frame rate of 90 frames per second can be obtained. Compared to the billboard implementation, our method incorporates correct alpha blending, texture based animation and does not require the geometry to be modeled by hand. The much higher performance of our method can be explained by the reduced overdraw and the fact that current hardware is fill-rate optimized.

A HLSL implementation of the grass shader and the textures used in this paper can be found at [Hab].

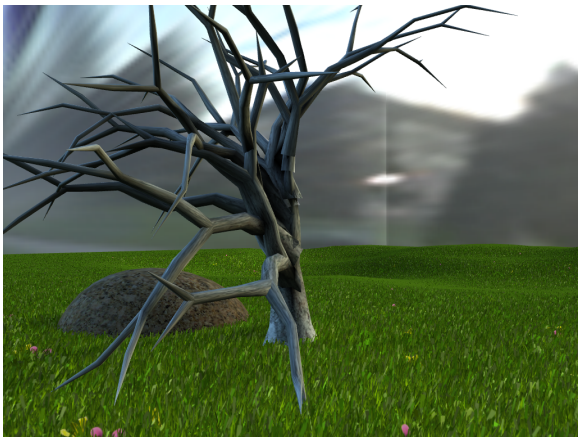

Figure 8: A terrain textured with short, dense grass.

6. CONCLUSION AND FUTURE WORK

This paper has introduced a new method to render animated grass in real time. Instead of rendering polygonal billboards, the technique uses front-to-back compositing of implicitly defined grass slices in a fragment shader and therefore significantly reduces the overhead associated with rendering dense vegetation scenes. One of the main advantages of the method is its ease of integration: as all operations are refined to a single fragment shader, grass can simply be incorporated as a material into any existing scene and renderer that supports hardware shading, thus avoiding the tedious modeling effort and storage costs of geometric grass billboards. Furthermore, we have shown a texture-based animation technique that combines consistent global wind motion with small, high-frequency perturbation, leading to a much more natural impression than previous analytic methods.

The performance of the shader is independent of the density of the grass, so a massive amount of slices can be rendered. Standard filtering techniques and levels of detail can be used with the proposed method. We are currently investigating methods to display silhouettes of grass by adapting higher order surface approximations to the presented algorithm and ways to move the camera into the grass consistently. Furthermore, it should be easy to break up the regularity of the grass patches using Wang tiling [CSHD03]. Finally, we are

investigating ways to incorporate advanced lighting techniques like self shadowing to even further increase realism.

7. ACKNOWLEDGEMENTS

This research was funded by the Austrian Science Fund (FWF) under contract no. P17261-N04. The authors would like to thank Oliver Mattausch for very helpful discussions and comments.

REFERENCES

[BH02] Brook Bakay and Wolfgang Heidrich. Real-time animated grass. In *Proceedings of Eurographics (short paper)*, 2002.

[CSHD03] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics*, 22(3):287–294, 2003.

[Hab] www.cg.tuwien.ac.at/research/publications/2007/Habel_2007_IAG/.

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

[Len00] Jerome Edward Lengyel. Real-time hair. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 243–256, London, UK, 2000. Springer-Verlag.

[LL94] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 451–458, New York, NY, USA, 1994. ACM Press.

[LPFH01] Jerome E. Lengyel, Emil Praun, Adam Finkelstein, and Hugues Hoppe. Real-time fur over arbitrary surfaces. In *2001 ACM Symposium on Interactive 3D Graphics*, pages 227–232, March 2001.

[MN98] Alexandre Meyer and Fabrice Neyret. Interactive volumetric textures. In George Drettakis and Nelson Max, editors, *Eurographics Rendering Workshop 1998*, pages 157–168, New York City, NY, July 1998. Eurographics, Springer Wien. ISBN 3-211-83213-0.

[Ney98] Fabrice Neyret. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):55–70, 1998.

[OGR] OGRE Graphics Engine www.ogre3d.org.

[OP05] Manuel M. Oliveira and Fabio Policarpo. An efficient representation for surface details. *UFRGS Technical Report RP-351*, 2005.

[PC01] Frank Perbet and Maric-Paule Cani. Animating prairies in real-time. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 103–110, New York, NY, USA, 2001. ACM Press.

[PD84] Thomas Porter and Tom Duff. Compositing digital images. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 253–259, New York, NY, USA, 1984. ACM Press.

[Pel04] Kurt Pelzer. *GPUGems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, chapter 7 Rendering Countless Blades of Waving Grass, pages 107–121. Addison-Wesley, 2004.

[Per85] Ken Perlin. An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA, 1985. ACM Press.

[POaLDC05] Fábio Policarpo, Manuel M. Oliveira, and Jo ao L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. *ACM Trans. Graph.*, 24(3):935–935, 2005.

[WWT+03] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. View-dependent displacement mapping. *ACM Trans. Graph.*, 22(3):334–339, 2003.