GPU-based Responsive Grass

Orthmann Jens
Computer Graphics Group
University of Siegen
56076, Siegen, Germany
orthmann@fb12.uni-siegen.de

Christof Rezk Salama
Computer Graphics Group
University of Siegen
56076, Siegen, Germany
rezk@fb12.uni-siegen.de

Andreas Kolb
Computer Graphics Group
University of Siegen
56076, Siegen, Germany
kolb@fb12.uni-siegen.de

ABSTRACT

Large natural environments are often essential for todays computer games. Interaction with the environment is widely implemented in order to satisfy the player's expectations of a living scenery and to help increasing the immersion of the player. Within this context our work describes an efficient way to simulate a responsive grass layer with todays graphics cards in real-time. Clumps of grass are approximated by two billboard representations. GPU-based distance maps of scene objects are employed to test for penetrations and for resolving them. Adaptive refinement is necessary to preserve the shape of deformed billboards. A recovering process is applied after the deformation which restores the original that is to say the undeformed and efficient shape. The primitives of each billboard are assembled during the rendering process. Their vertices are dynamically lit within an ambient occlusion based irradiance volume. Alpha-to-Coverage completes the illusion as it is used to simulate the semitransparent nature of grass.

Keywords: Grass Simulation, Interactive Environments, GPU-based Collision Handling

1 INTRODUCTION

State-of-the-art 3D games and realtime simulations demonstrate the power of currently available graphics hardware for rendering exciting natural sceneries in real-time. As nature scenes often include a lot of plants (blades of grass, shrubs, trees etc.) the rendering of a large number of them is still challenging. Furthermore, they cannot be displayed with complex geometry in real time. Many of the approaches make use of billboard representations to preserve the real-time constraint while leaving out user interaction.

In general, static level design is more and more replaced by dynamic environments that can be modified in real-time throughout the gaming process. Due to the fact that natural phenomena are better approximated in the game, the player feels a higher immersion while playing [McM03]. Consequently, the dynamic environment is becoming a part of the game logic: Trees are chopped to clear the path and objects need to be moved in order to fulfill quests. The more the realism of the scene is enhanced the more of the player's expectations are satisfied.

Following this trend, our paper takes dynamic environments one step further by integrating responsive real-time simulation of ground vegetation. We propose a highly efficient technique for GPU-based simulation of responsive grass billboards. Our implementation targets Shader Model 4 graphics boards, including geometry shaders and stream output. The collision detection with dynamic scene objects, the response and the recovering are directly simulated on the GPU. An adaptive geometrical representation of the grass guarantes a pleasing visual rendering in conjunction with a high performance. Thus, the responsive grass approach has the potential to significantly improve the challenges in game play of modern games and may lead to a better perception of interactive environments.

The structure of this paper is as follows: in Section 2, an overview of the related work on grass simulation is given, followed by a overview of the responsive grass system in Section 3. Section 4 proposes the procedural generation process of the grass layer. In Section 5, the realization of the collision system is described. The rendering of the grass layer is presented in Section 6 and the results and performance of our technique is discussed in Section 7. Finally, Section 8 concludes the presented responsive grass approach.

2 RELATED WORK

In recent years, most research applied to natural sceneries focuses on the rendering and animation of a great number of plants. For volumetric representations, as proposed in [BCF+05, BPB06], collision detection and reaction is awkward to handle. Guerraz et al. [GPR+03], however, presented an approach which allows an object to tramp on the grass layer. A primitive is moved along the character's trajectory while affecting the procedural animation process of the grass. Nevertheless there still is no possibility to react to collision, based upon the object's geometry. The reuse of grass tiles amplifies the problem of collision

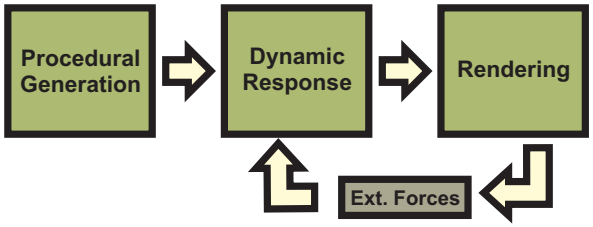

Figure 1: The system for responsive grass.

response. Billboards which represent a number of grass blades as a semi-transparent 2D texture are more suitable in that case. The billboard representations stored in a single vertex buffer [Wha05] are efficiently animated [Pel04, Bot06, Sou07] and rendered [BCF+05] on the GPU.

As the collision detection for grass is less explored, related algorithms on a wider range are examined. In case large dynamic geometry is stored and processed completely on the graphics memory, image-based techniques [VSC01, KP03, HTG04, KLRS04, GLM05, Sat06] are proven to solve the collision tests very fast. Kolb et al. [KLRS04] offered an approach to collision detection using distance maps which are fully generated and accessed on the GPU. A lookup into each distance map is used to decide whether a vertex lies inside or outside of a object. Using the normal information the vertex can be translated in the direction of the shortest way out of the object. Their approach fits best in case all computations, including the collision reaction, are done on the GPU.

Cloth models [Pro95, FGL03, Zel07] are applied in order to overcome the problems in the context of the collision reaction. Fuhrmann et al. [FGL03] replace the cloth forces [Pro95] by several length constraints along the connection of two particles in order to avoid problems which are caused by large time steps. Zellner [Zel07] entirely offloads the model to the GPU and handles the recursion via the stream output stage.

Regarding high quality rendering of massive material scenery a precomputed irradiance volume is employed [Oat06, CL07]. The volume stores the irradiance information of the whole static scene. Interpolation within the volume allows us to dynamically lit the grass billboards at runtime similar to the two-sided lighting proposed by Kharlamov et al. [KCS07]. The Alpha-To-Coverage feature of todays graphic cards [Mye06] avoids expensive depth-sorting of the semi-transparent billboards while maintaining a consistent visual appearance similar to David Whatleys procedure [Wha05].

3 SYSTEM OVERVIEW

The pipeline for responsive grass comprises the following components as shown in Figure 1:

- **Procedural Generation:**
  For a given terrain mesh, a geometry shader automatically generates billboards for grass blades. This geometry shader is executed once for each tile of terrain, and the results are stored in local video memory using the stream-out capabilities. We describe the process in detail in Section 4.

- **Dynamic Response:**
  A CPU-based broad phase working on the spatial organized grass tiles and a GPU-based narrow phase working on the generated grass billboards constitute the responsive component. During this stage the grass layer will be adapted whenever external forces like colliding scene objects make it necessary. This process which is implemented within the collision system is outlined in Section 5.

- **Rendering:**
  Deformed or undeformed billboards are rendered based on the output of the collision system. Precomputed occlusion volumes respectively irradiance volumes may be employed to integrate ground vegetation into a dynamic global lighting environment. We adapt such techniques for realistic rendering of dynamic ground vegetation as described in Section 6.

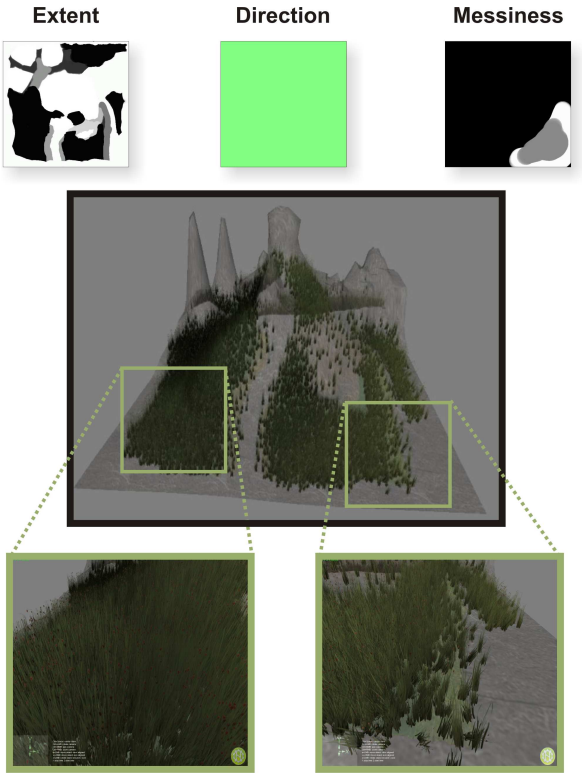

Figure 2: The top row shows the texture images for the extent, direction and messiness and underneath the resulting plant cover is displayed.

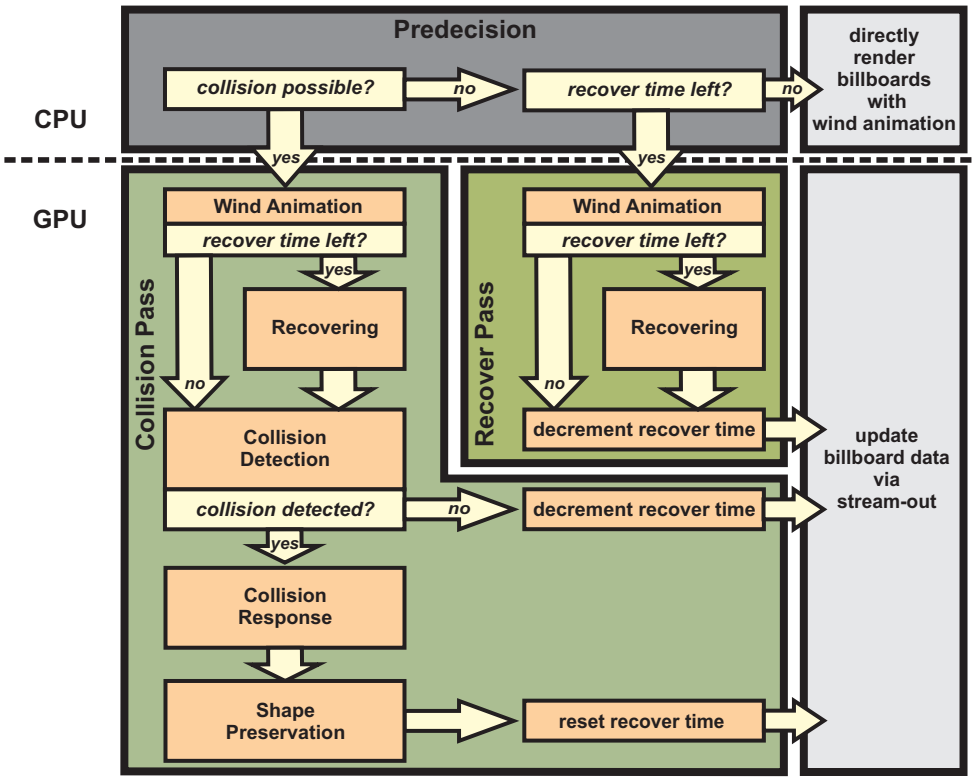

Figure 3: Flow diagram for collision detection, reaction and recovering.

4 PROCEDURAL GENERATION OF THE BILLBOARDS

A clump of grass is represented by a semi-transparent textured quad. The individual billboards are created in a pre-processing step performed by the GPU. A set of texture images provides the information of the global layout of the grass layer as shown in Figure 2. These textures are in detail:

- a grayscale texture map which defines the regions of the plant cover (extent),
- a RGB texture which defines the direction to which the grass blades grow (for simplicity the direction is chosen to be the same for all grass blades),
- a grayscale *messiness* texture which defines the amount of randomness for the blades.

The geometry shader creates a randomized set of billboards representing the grass blades. Each billboard stores an orientation, a position, a collision state, and a texture index, addressing a 2D texture array, which stores different semi-transparent images of grass clumps. Each billboard is passed through the pipeline as a point primitive, which allows the different geometry shaders to handle its information en bloc during the collision handling and rendering. When the billboards are generated, they are streamed to one large vertex buffer [Wha05] to minimize subsequent render calls. For a coarse collision detection on the CPU, the terrain mesh is used to divide the set of billboards into an octree hierarchy. Each leaf node of the octree stores a range of indices into the vertex buffer of the billboards and state information described throughout the next section.

5 COLLISION SYSTEM

The pipeline of the collision system is outlined in Figure 3. Without collision, the billboard quads can directly be rendered. The upper two vertices of each billboard quad are transformed with a procedural wind animation based on a weighted sum of trigonometric functions with different frequencies [Pel04, Wha05, Bot06, Sou07]. The collision system is split into a CPU-based coarse handling and two GPU-based procedures, one for executing the collision test and response and one for performing the recovering. The different steps of the GPU-based collision handling are outlined in Figure 4.

5.1 Coarse Handling on the CPU

At each frame, the bounding volumes of all dynamic collision meshes are tested for collision with the axis aligned bounding boxes (AABB) of the octree containing the grass blades. According to the current state information and the results of the collision test, each node is marked as either *possibly colliding*, *non-colliding* or *recovering*. The geometry assigned to each octree node

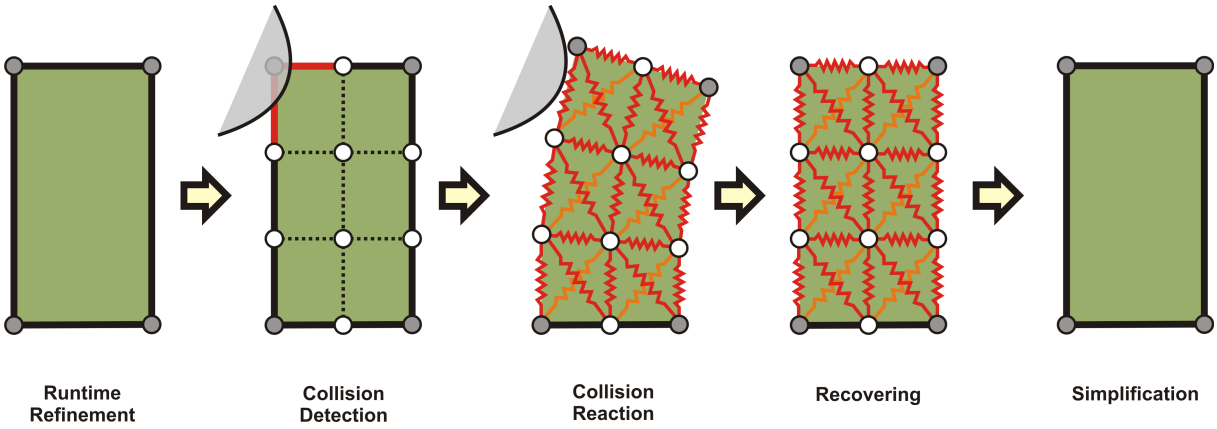

Figure 4: Collision detection, reaction and recovering for a single billboard.

marked as *possibly colliding* is streamed through a collision pass on the GPU. A tile marked as *recovering* will stay active for a fixed amount of time after the collision when the object has left the AABB of the octree node. During that time a separate geometry shader recovers the original shape of the grass blades.

5.2 Collision Detection

In the collision pass, a geometry shader receives all the vertex information of a potentially colliding billboard at once. The geometry shader computes the bounding sphere of the animated billboard and performs a collision test against the bounding spheres of the dynamic objects. These bounding spheres are managed in a dynamic texture resource, which is updated every frame. If the collision test is passed on the bounding sphere level, a second and more exact collision test is performed on a subdivided mesh of the billboard. The geometry shader determines for each vertex whether it lies inside or outside the dynamic collision object by performing lookups into the depth cube map [KLRS04] of the object. Dynamic objects capable of colliding with the plant cover are represented by depth cube maps for efficiency. These cube maps are computed by projecting the object's mesh onto the faces of a bounding cube and store the distance to the cube plane and the respective object normal in the four texture components. They are updated for each frame to account for animated objects. Thus, to perform the test each vertex $\mathbf{v} = (v_x, v_y, v_z, 1)^T$ is transformed to each of the six projection spaces:

$$\mathbf{v}^i = \mathbf{T}^i_{OC \to DM} \ \mathbf{v}\,, i = 1,..,6\,, \tag{1}$$

where $\mathbf{v}^i = (v^i_x, v^i_y, v^i_z, 1)^T$ is the transformed vertex of the billboard. $\mathbf{T}^i_{OC \to DM}$ is a transformation from the object coordinate space to the i-th projection space from where the current distance map was computed. Along the projection direction the vertex lies within the object if

$$d^i(\mathbf{v}) = dm^i(v^i_x, v^i_y) - v^i_z < 0\,, \tag{2}$$

where $dm^i(x,y)$ is the distance looked up within the distance map dm^i at pixel position (x,y). If the distances for all the six faces i of the cube do not yield a distance value $d^i(\mathbf{v})$ less than zero a collision with the billboard has been detected. Identifying the distance $d(\mathbf{v})$ between the closest surface point in the corresponding depth cube face for a given point $\mathbf{v}$, the following formula is used (for details see [KLRS04]):

$$d(\mathbf{v}) = \begin{cases} \max\{d^i(\mathbf{v})\} & \text{if } d^i(\mathbf{v}) < 0 \ \forall i \\ \min\{d^i(\mathbf{v}) \ : \ d^i(\mathbf{v}) > 0\} & \text{else} \end{cases} \tag{3}$$

5.3 Collision Response

If a collision has been detected, the vertex is moved out of the object's shape. Its position is translated along a normal vector $\mathbf{n}$ obtained from the depth cube map:

$$\mathbf{v} \leftarrow \mathbf{v} + s\,\mathbf{n}\,, \tag{4}$$

where $\mathbf{n}$ is taken from the depth cube map face dm^i providing the smallest distance. s is the reaction strength that is to say the surface normal $\mathbf{n}$ multiplied with the smallest distances to the surface $d(\mathbf{v})$:

$$s = d(\mathbf{v}) \ \frac{\mathbf{n}}{\|\mathbf{n}\|}\,. \tag{5}$$

In order to remember the collision, the data-structure of the billboard is expanded by an additional value storing its recover time. In case of a collision the recover time is reset.

5.4 Shape Preservation

As the separate processing of individual vertices may lead to visually unpleasant distortions, a cloth model based on spring constraints [Pro95, FGL03, Zel07], is applied to preserve the overall shape of the grass clump. A network of structural and shear springs takes care of the billboard mesh. Whenever such a spring is compressed or stretched, which means the connected vertices diverge or converge, the resulting spring force translates the connected vertices.

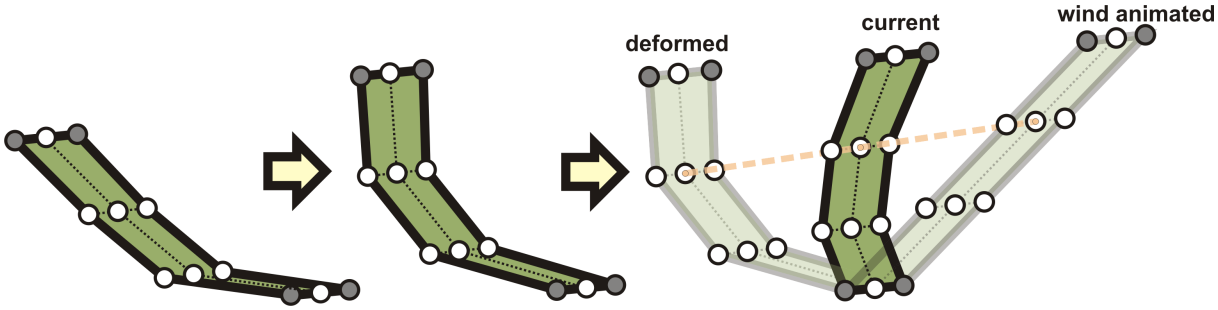

Figure 5: The interpolation between the vertices of the current mesh (deformed mesh) and the vertices solely affected by the wind animation results in a smooth recovering over time.

Referring to Provot et al. [Pro95], a spring force $\mathbf{f} \in R^3$ between two billboard vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ is defined as:

$$\mathbf{f} = k(\|\mathbf{l}\| - l_0)\,\frac{\mathbf{l}}{\|\mathbf{l}\|}\,, \tag{6}$$

where $\mathbf{l} = \mathbf{v}_1 - \mathbf{v}_2$ is the direction of the connection between both vertices. l_0 is the initial length of the spring and $k \in [0,1]$ is the stiffness of the spring. A stiffness of 1 results in a conservative spring in contrast to a value of 0 which has no effect. Each spring force directly affects the two connected vertices [FGL03, Zel07]:

$$\begin{aligned} \mathbf{v}_1 &\leftarrow \mathbf{v}_1 - r_1\,\Delta t\,\mathbf{f} \\ \mathbf{v}_2 &\leftarrow \mathbf{v}_2 + r_2\,\Delta t\,\mathbf{f}\,, \end{aligned} \tag{7}$$

where r_1 is the responsiveness for vertex $\mathbf{v}_1$ and r_2 is the responsiveness for vertex $\mathbf{v}_2$ with $r_1 + r_2 = 1$. We added the responsiveness in order to distinguish between fixed ground vertices and movable vertices. As a fixed vertex should not be moved, the responsiveness is set to zero whereas the other vertex then is completely responsive. If both vertices are not fixed they are equal responsive and thus $r_1 = r_2 = 0.5$.

As the relaxation of one spring affects the neighbouring springs as well, in general more iterations over all springs have to be applied to get a good result. In our case two iterations yield visually pleasant results due to the small number of vertices.

5.5 Recovering

The recovering is processed on each billboard that has some recover time left. Since the animation is a stateless process, solely based upon the position of the fixed ground vertices and the current time [Sou07], it is possible to compute the original shape defined by the wind without considering the current collision state. The linear interpolation between the deformed vertex and its original position, with respect to the recover time left, results in the current shape of the grass clump as shown in Figure 5:

$$\mathbf{v} \leftarrow\ (1-t^3)\mathbf{w} + t^3\mathbf{v}\,, \tag{8}$$

where $t \in [0,1]$ is the recover time left, $\mathbf{w}$ is the vertex position obtained by the wind function and $\mathbf{v}$ is the current respectively last recovered vertex position.

Collision tests are required in case that there are still collision objects inside the AABB of the respective octree node. At every time without any collision, the recover time will be decreased. After the recover time has elapsed, the billboards will be handled again as simple quads. However, the recovering does not preserve the length of the billboards.

6 RENDERING

On the CPU level, grass tiles which previously have been streamed and others that have not been affected by neither collisions nor wind exist. The tiles run through separate render passes: Collided billboards are rendered using their current refined mesh whereas the unaffected ones are animated and rendered using their simple quad-representation. Furthermore, to overcome problems caused by too much render calls, only batches of visible tiles, which have not been culled by view or occlusion queries, are rendered.

6.1 Global Illumination

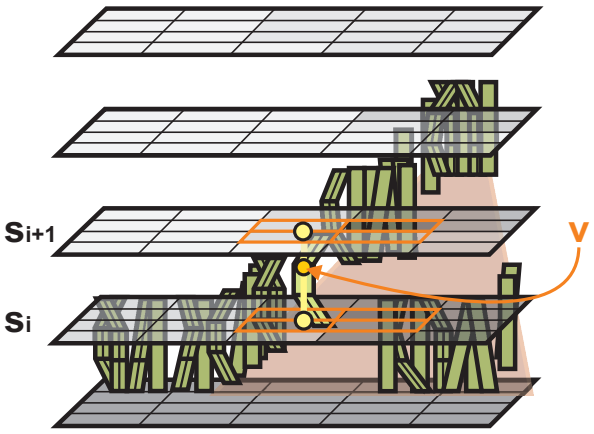

Figure 6: The irradiance for each vertex $\mathbf{v}$ of the billboard is interpolated within the two closest texture slices s_i and s_{i+1}.

Dynamic global illumination is achieved by pre-computing a volume, with each voxel storing ambient occlusion information for its location in the scene [CL07]. The whole volume is then stored

Figure 7: The result of the collision handling in a dense field of grass.

as an 2D texture array to allow linear interpolation based on mip-mapping. In addition, a second volume which covers the same space provides pre-computed irradiance information for each point. The irradiance is determined by sampling an environment map by using the previously computed ambient occlusion information [PG04].

The texture coordinate for the volume texture can easily be obtained from the billboard's vertex positions in the geometry shader. Ambient occlusion and irradiance information is trilinear interpolated between adjacent texture slices, and the incident light is evaluated per vertex during the geometry shader process as illustrated in Figure 6. Finally, the pixel shader uses the texture index into the semi-transparent texture array to receive the decal color and transparency of the grass clump. Multiplying this decal value with the incident two-sided light [KCS07] results in the final semi-transparent pixel color.

6.2 Alpha-To-Coverage

Since grass has a semi-transparent nature a feature of modern cards, so-called Alpha-to-Coverage, is used to blend the billboards without the necessity to perform expensive depth-sorting. The alpha value is used to determine the number of subpixels, that will be filled with the current pixel color. Then, blending between the subpixels is performed while resolving the multisample resolution to the final image resolution [Mye06].

7 RESULTS AND PERFORMANCE

Achieving a high performance is one of the major aims to real-time applications. All components concerning the grass layer are designed to reduce the workload of the CPU as much as possible. Thus, the simulation is almost completely shifted to the GPU. All the tests are performed on an AMD Athlon 64 3500+ 2.2 GHz processor including a GeForce 8800 GTX graphics card with 768 MB DDR3 memory. Figure 7 shows the response of the grass after the scene object has moved through the meadow. The scene, presented in Figure 2, is running at 40-80 frames per second by using DirectX 10 and fourfold multi sampling anti aliasing (4xMSAA). The grass layer contains 60000 grass billboards requiring 12 MByte of graphics memory. All invisible grass tiles are culled. The grass is pushed to the side or is stamped down on the line of movement. The object has left a clearly noticeable imprint on the grass. We analyzed the performance of the scene with the aid of the NVidia PerfHUD tool. In Figure 8 the number of colliding grass tiles (red boxes) respectively recovering grass tiles (green boxes) increases from top down. The lower left overlay displayed in each image shows the workload balancing of the programmable render pipeline stages: The unified streaming processors are utilized to work on pixels with about 50 to 60 percent (the blue bar) whereas the geometry shader unit of the pipeline is active by approximately ten percent (the green bar). The remaining workload is caused by frame buffer operations. Approximately 16 million pixels are processed within the fragment shader resulting in many read as well as write accesses to the frame buffer. Those are amplified by the Alpha-to-Coverage feature which in that case requires a multisample resolution that is four times higher than the image resolution. The diagrams located at the right hand side of each image in Figure 8 present the amount of time which is consumed within each GPU pass: Please note that the time spent within the recover process (R) and the collision pass (C) varies only by small amounts. In contrast, the more grass billboards are deformed the more time is spent rendering the collided and recovering grass tiles (RA). This performance loss is caused by the primitive generation as well as the rendering of the high number of primitives. Referring to the utilization graph and the time measurements the performance of the system depends on the number of assembled primitives which are passed through the rasterizer back-end. Thus, both the memory operations as well as the workload shifted to the fragment shader stage, are influenced by the number of colliding grass billboards. Consequently, it is necessary to set up a low recover time and to provide a low multi-sampling rate for the Alpha-to-Coverage process to preserve the overall performance. In con-

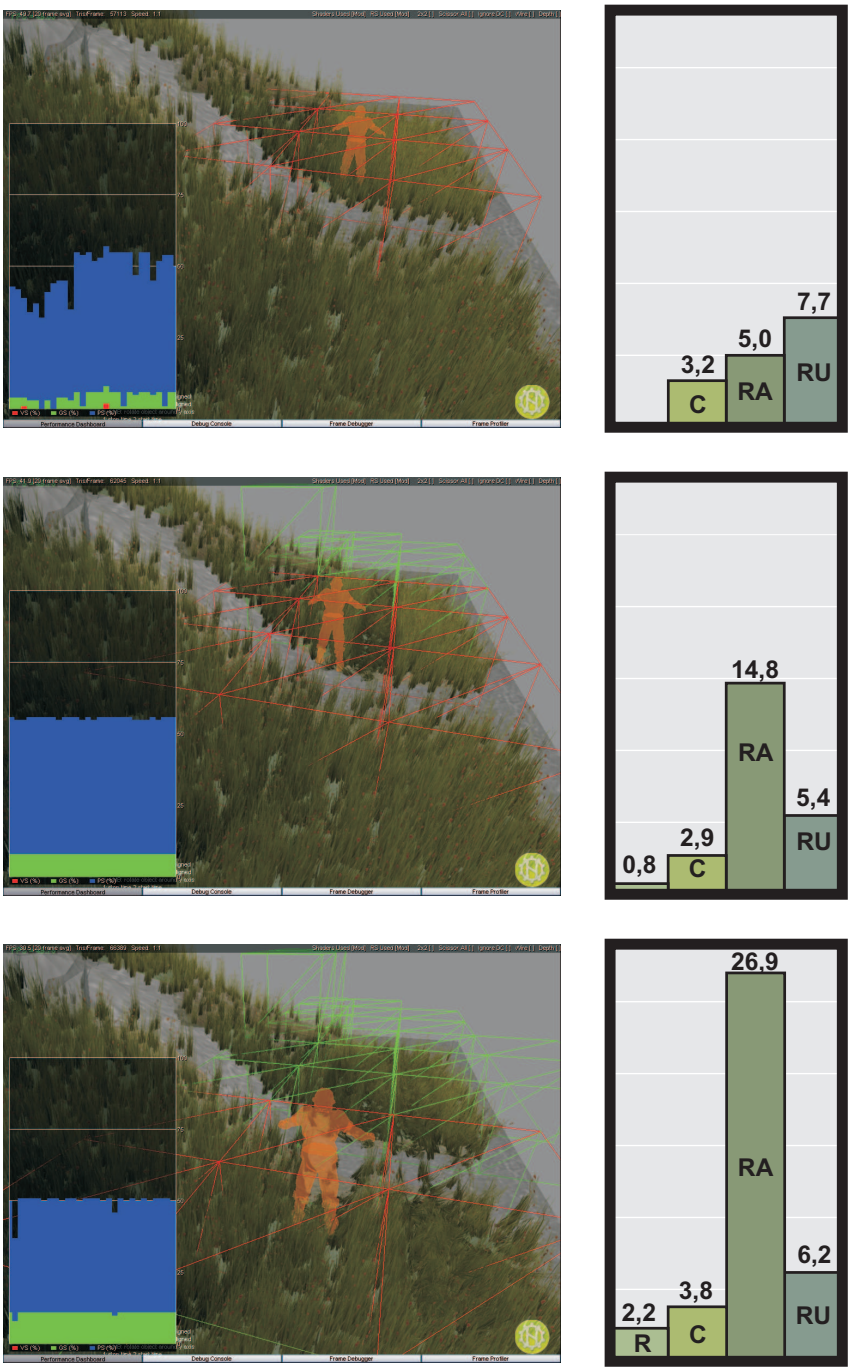

Figure 8: A scene which contains more and more deformed grass billboards. The time spent within each GPU pass is displayed in milliseconds on the right-hand side of the corresponding image. The measured passes are: The recovering pass (R), the pass performing the collision handling (C), the rendering of the possibly affected grass tiles (RA), and the rendering of the unaffected grass tiles (RU).

trast, the time spent within the collision handling depends mainly on the number of scene objects moving through the grass layer.

8 CONCLUSION AND FUTURE WORK

In the past thousands of billboards were successfully used to create an illusion of dense grass vegetation. In combination with wind animation nice visual results were achieved. But the visual perception was often compromised by lack of interactivity: Objects are moving through the grass without leaving a trace. Due to prior hardware constraints a visually pleasing collision reaction for a large area of grass was unachievable. The visual quality of dense vegetation and the good performance give a proof of the great suitability of our imple-

mentation strategies for large responsive grass layers in todays real-time applications.

The results are demonstrating that collision response works fine for regions where the flat structure of the grass billboards is hardly recognized. However, in areas where grass is planted sparsely, for example at the borders of the grass layer, due to the coarse mesh of the billboards the visual impression could be improved. Two different approaches might be promising when trying to solve this problem: On the one hand the collision handling for each billboard could be distributed over several streaming passes which allows the spring constraints to work on a higher subdivided mesh. On the other hand the displayed primitives could be assembled by a higher order interpolation during rendering.

REFERENCES

[BCF+05] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen. Realistic real-time rendering of landscapes using billboard clouds. *Comput. Graph. Forum*, 24(3):507–516, 2005.

[Bot06] A. Botorabi. *Shader X5*, chapter Animating Vegetation Using GPU Programs, pages 141 – 175. Charles River Media, 2006.

[BPB06] K. Boulanger, S. Pattanaik, and K. Bouatouch. Rendering grass in real-time with dynamic light sources and shadows, 2006.

[CL07] G. Cadet and B. Lécussan. Fast approximate ambient occlusion. In *SIGGRAPH '07: ACM SIGGRAPH 2007 posters*, page 191. ACM, 2007.

[FGL03] A. Fuhrmann, C. Groß, and V. Luckas. Interactive animation of cloth including self collision detection. In *WSCG*, 2003.

[GLM05] N. K. Govindaraju, M. C. Lin, and D. Manocha. Quick-cullide: Fast inter- and intra-object collision culling using graphics hardware. In *VR '05: Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, pages 59–66, 319, Washington, DC, USA, 2005. IEEE Computer Society.

[GPR+03] S. Guerraz, F. Perbet, D. Raulo, F. Faure, and M.-P. Cani, editors. *A Procedural Approach to Animate Interactive Natural Sceneries*. IEEE Computer Society, 2003.

[HTG04] B. Heidelberger, M. Teschner, and M. H. Gross. Detection of collisions and self-collisions using image-space techniques. In *WSCG*, pages 145–152, 2004.

[KCS07] A. Kharlamov, I. Cantlay, and Y. Stepanenko. *GPU Gems 3*, chapter Next-Generation SpeedTree Rendering, pages 69–92. Addison-Wesley, 2007.

[KLRS04] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 123–131, New York, NY, USA, 2004. ACM.

[KP03] D. Knott and D. K. Pai. CIndeR: Collision and interference detection in real-time using graphics hardware. In *Graphics Interface*, pages 73–80, 2003.

[McM03] A. McMahan. *Immersion, Engagement, and Presence - A Method for Analysing 3-D Video Games*, chapter 3, pages 67–85. Routledge Taylor & Francis Group, 2003.

[Mye06] K. Myers. *Shader X5*, chapter Alpha-to-Coverage in Depth, pages 69 – 74. Charles River Media, 2006.

[Oat06] C. Oat. *Shader X5*, chapter Irradiance Volumes for Real-time Rendering, pages 333 – 357. Charles River Media, 2006.

[Pel04] K. Pelzer. *GPU Gems*, chapter Rendering Countless Blades of Waving Grass, pages 107 – 121. Addison-Wesley, 2004.

[PG04] M. Pharr and S. Green. *GPU Gems*, chapter Ambient Occlusion, pages 279 – 292. Addison-Wesley, 2004.

[Pro95] X. Provot. Deformation constraints in a mass–spring model to describe rigid cloth behavior. In *Graphics Interface '95*, pages 147–154, May 1995.

[Sat06] R. Sathe. *Shader X5*, chapter Collision Detection Shader Using Cube-Maps, pages 533 – 542. Charles River Media, 2006.

[Sou07] T. Sousa. *GPU Gems 3*, chapter Vegetation Procedural Animation and Shading in Crysis, pages 373 – 407. Addison-Wesley, 2007.

[VSC01] T. I. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. *Comput. Graph. Forum*, 20(3), 2001.

[Wha05] D. Whatley. *GPU Gems 2*, chapter Toward Photorealism in Virtual Botany, pages 7–25. Addison-Wesley, 2005.

[Zel07] C. Zeller. Cloth simulation. White paper, NVidia, 2007.