Real-time animation of grass

Chen, Kan

2009

Chen, K. (2009). Real-time animation of grass. Master' s thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/36294>

<https://doi.org/10.32657/10356/36294>

Real-Time Animation of Grass

by

Chen Kan

School of Computer Engineering

Nanyang Technological University

2009

Real-Time Animation of Grass

by

Chen Kan

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Engineering

in the

School of Computer Engineering

Nanyang Technological University

2009

Abstract

Real-time grass has many applications, such as in games, simulators and virtual reality. Physically-based simulation of grass-grass, grass-object, grass-terrain and grass-wind interactions requires a high computational cost. In this thesis, we present a method to animate grass in real-time by considering a grass field as a 2D continuum and shifting the complex interactions to the dynamics of continuum. We adopt a wave simulation as the numerical model for the dynamics of continuum which represents grass-grass interaction. Using the wave simulation, we modify the bending angle and the bending velocity of grass. We propose a procedural approach to handle grass-object and grass-wind interactions which directly updates the bending angle and the bending velocity of the grass that collides with the object and wind. Then, we propagate the bending angle and bending velocity to the neighboring grass using the wave simulation. To avoid the intersection between grass and the terrain, the maximum bending angles are calculated based on the slope of the terrain and the bending direction of the grass. The proposed method can be efficiently implemented on a GPU. As a result, massive amount of grass can interact with moving objects and wind in real-time.

Acknowledgements

I would like to thank my supervisor, Dr. Henry Johan, for his aspiring guidance and encouragement throughout this project. I also would like to thank TQ Global, for providing an excellent infrastructure and an ideal work environment for this research. Furthermore, I would like to thank all my friends who have given me their valuable pointers on improving this project. Finally and above all I would like to express my deep gratitude to my parents Chen Feiqiang and Wang Jian and my girlfriend Han Jing for their love and extraordinary emotional support.

Contents

Abstract	ii
Acknowledgements	iii
Table of Contents	iv
List of Figures	v
List of Tables	vi
1 Introduction	1
1.1 Background	1
1.2 Objectives	4
1.3 Contributions	5
1.4 Thesis organization	6
2 Related Work	7
2.1 Simulation of nature scene	7
2.1.1 Fluid	8
2.1.2 Cloud	9
2.1.3 Terrain	10
2.1.4 Wind	11
2.1.5 Animation of trees	11
2.1.6 GPU usage in simulation of nature scene	12
2.2 Grass modeling and animation	13
2.2.1 Modeling of grass	13
2.2.2 Animation of grass	16
2.2.3 Hair and fur	21
3 Modeling Field of Grass	22
3.1 Modeling of grass	22
3.2 Distribution of grass	23

4	Real-Time Grass Simulation	25
4.1	Basic idea and simulation steps	25
4.2	Simulation region management	28
4.3	Direct bending process	30
4.4	Propagation process	31
4.5	Animating the cluster of grass	34
5	Interaction with Objects and Wind	38
5.1	Collision detection	38
5.2	Collision response	39
5.3	Interaction with wind	41
6	GPU Implementation	44
6.1	GPU programming	44
6.1.1	Introduction	44
6.1.2	GPU programming concepts	46
6.2	GPU implementation of simulation steps	47
7	Results	50
8	Conclusion and Future Work	61
8.1	Conclusion	61
8.2	Limitations	62
8.3	Future work	63
A	Publications	64
	Bibliography	65

List of Figures

1.1	Screen shots of games (a) Call of Juarez 2: Bound In Blood (2009) [Ubi09] and (b) Arcania: A Gothic Tale (2009) [Sof09].	2
1.2	Grass is an important component of nature scene.	2
1.3	Screen shot of game Flatout Ultimate Carnage (2008) [Int08]. Grass penetrates through the car.	3
1.4	Real-time grass in game. The track appeared as the result of the interaction between the car and the grass.	4
2.1	Particle system [RB85].	13
2.2	Shell grass [BH02].	14
2.3	A grass field with three levels of details [BPB06].	15
2.4	A screen shot of the game Crysis (2008) [EA08] using billboards for vegetation modeling.	16
2.5	Procedural grass-wind interaction [PC01]. (a) and (b) are the results, (c) is the wind primitives.	17
2.6	Grass-wind interaction based on free form deformation and explicit grass-grass collision detection [WWZ ⁺ 05].	17
2.7	Shell grass with a mass-spring system [BW06].	18
2.8	Texture based animation [HWJ07].	18
2.9	Wind effects using a simple water wave [Ram05].	18
2.10	Object interacts with grass [GPR ⁺ 03].	19
2.11	Object interacts with grass [JSK09].	19
3.1	Use a billboard to model one cluster of grass. (a) is a one-segment billboard grass (stiff grass and short grass) and (b) is a multiple-segment billboard grass with four segments (soft grass and tall grass).	24
3.2	Example of a grass field. This grass field is divided into four patches, each patch has nine cells and each cell contains one cluster of grass.	24
4.1	Object interacts with (a) a one-segment billboard grass (stiff grass and short grass) and (b) a multiple-segment billboard grass (soft grass and tall grass).	26
4.2	Simulation steps.	27
4.3	The dark small rectangles represent the moving object. We allocate the memory for the patches of grass no. 2, 3, 5, 6, 8 and 9. The circles represent the affected area of the object.	29
4.4	Wave propagation.	32

4.5	Central difference approximations.	32
4.6	The bending process of one cluster of grass. Grass bends from the bottom to the top.	35
4.7	During time γ_1 , the θ value propagates from P to Q . The θ value also propagates from the bottom to the upper segment at P	36
4.8	Based on θ and L (the height of the segment), apply the forward kinematics.	37
5.1	Calculate the collision area.	39
5.2	Assume that the object has a rectangular shape. The calculation of the bending direction for the grass in collision area.	40
5.3	The calculation of the bending velocity of the grass in the collision area.	41
5.4	Two examples of the shape of the collision area and the bending direction designed by an animator.	42
5.5	A helicopter scene using the bending directions in (a) Figure 5.4(a) and (b) Figure 5.4(b).	42
5.6	Grass field in natural wind.	43
6.1	Programmable graphics pipeline [DR08].	45
6.2	Switch the simulation textures.	49
7.1	Animation sequence with one car and one-segment billboard grass.	52
7.2	Animation sequence with one car and multiple-segment billboard grass.	53
7.3	Concave object.	53
7.4	Animation sequence with multiple cars.	54
7.5	Animation sequence with helicopter with the bending direction as in Figure 5.4(b).	55
7.6	Control the propagation area. (a, c, e) Smaller λ : large area. (b, d, f) Greater λ : small area.	56
7.7	Control the propagation area (closer view). (a, c, e) Smaller λ : large area. (b, d, f) Greater λ : small area.	57
7.8	Control the propagation speed. (a, c, e) Smaller c : slow propagation. (b, d, f) Greater c : fast propagation.	58
7.9	Control the propagation speed (closer view). (a, c, e) Smaller c : slow propagation. (b, d, f) Greater c : fast propagation.	59
7.10	Comparison with Guerraz et al. [GPR ⁺ 03]. (a) Images from [GPR ⁺ 03]. (b) Our result.	60
7.11	Comparison with Orthmann et al. [JSK09]. (a) Images from [JSK09]. (b) Our result.	60

List of Tables

2.1	Comparison with other grass animation methods.	20
7.1	Performance of the game with different settings on GPU-1 and GPU-2. .	52

Chapter 1

Introduction

Wild grasses spread over ancient plain; With spring and
fall they come and go. Fire tries to burn them up in
vain; They rise again when spring winds blow.

Bai Juyi

1.1 Background

Modeling nature scene is essential for computer graphics. It has various applications, such as digital games, virtual environments and etc. Real-time simulation of a large scale dynamic and believable nature scene provides users with an immersive feeling (Figure 1.1). Moreover, simulating nature scene and natural phenomena covers a wide range of research areas. The popular examples are the simulations of water, fire, smoke, terrain, cloud, vegetation and etc. Grass is the vegetation that is very abundant on earth. It can be found in meadows, prairies, forests, mountains, savanna, football stadiums and etc.,

FIGURE 1.1: Screen shots of games (a) Call of Juarez 2: Bound In Blood (2009) [Ubi09] and (b) Arcania: A Gothic Tale (2009) [Sof09].

FIGURE 1.2: Grass is an important component of nature scene.

thus grass field is an essential component of nature scene (Figure 1.2).

Using texture mapping and modeling techniques, we can realize the visual complexity of a grass field (Figure 1.1). Nevertheless, static scenery is not enough for presenting users with a rich view into an artificial world. One of the principles of animation is that the viewer should never be unintentionally surprised by the motion or lack of it in a scene. Nowadays, the static sceneries have been replaced by the dynamic environments that can interact with the users in real-time throughout the gaming or simulating process. The interactive environment is becoming an essential part of digital games and virtual reality: when objects pass and wind blows, the water will be perturbed, the branches and leaves

FIGURE 1.3: Screen shot of game Flatout Ultimate Carnage (2008) [Int08]. Grass penetrates through the car.

of trees will be responsive, a track may be left on the sand and etc. The better the natural behavior is modeled, the more immersive the users will feel. However, in many real-time applications, nature scene (especially the grass field) is still lack of interactivity. For example, in the game Flatout Ultimate Carnage [Int08] (Figure 1.3), the grass cannot interact with object and the grass penetrates through the car. This effect will cause the user feel uncomfortable and ruin the overall effect of the nature scene. The grass field will look more believable and the user will be more satisfied if we also simulate the countless blades of grass interacting with objects and wind. For instance, grass should bend and recover in response to the passing car and wind. In addition to being visually more believable, the interactive grass may also deliver important information that help the user to understand the scene. For example, Figure 1.4 shows an image of a car moving across a grass field. The presence of the track makes it clear that the grass is soft rather than hard shrub.

FIGURE 1.4: Real-time grass in game. The track appeared as the result of the interaction between the car and the grass.

It is obvious that grass field is an important component of outdoor environment and it is required in many applications, such as virtual reality, games and simulators. Grass rendering and modeling techniques are more and more popular. However, because of the huge number of grass, the dynamics and the complex physical interactions remain challenges for grass animation. Performing an explicit physics simulation of the grass can produce realistic animation, but the computational cost is extremely high. For example, to simulate an object moving in a dense grass field, we have to perform a large number of grass-object and grass-grass collision detections and responses. Up to now, there has not been much work addressing a real-time solution for such application.

1.2 Objectives

In this thesis, we propose a method to achieve real-time animation of grass. The requirements of the method includes:

- (1) **Grass interactivity:** This method should simulate the interactions and natural

effects that are considered typical for games and simulators, such as grass-car, grass-helicopter and grass-wind interactions. The method should be robust enough to model these complex interactions and effects for a large scale grass field at interactive frame rates.

(2) **Grass properties:** The method models the properties of grass, such as bending and recovery. Grass should bend and recover in response to the passing car and wind, and the bending grass may cause other grass to bend. Furthermore, different types of grass have different properties that react differently upon collisions. For example, the shrub like grass is hard to bend while the normal grass is softer. Moreover, grass is hard to break and stretch but free to move in all directions. Other than the grass' physical properties, the behaviors of grass also depend on how objects and wind interact with them. For example, the grass will likely bend according to the moving direction of the objects during the interaction. The grass' properties as well as the objects and wind's behaviors should be considered in the design of the model.

(3) **User control:** The users have control over the grass to a certain degree.

1.3 Contributions

We propose a novel method that combines a simple dynamic simulation and a procedural method. Our method to simulate grass integrates four types of interaction: grass-grass, grass-object, grass-terrain and grass-wind. The proposed method has the following features:

(1) Grass field is treated as a 2D continuum, which stores the grass' status (bending angle and bending velocity). The flow inside the continuum is computed based on the

wave simulation. The propagation property of the wave represents the grass-grass interaction. We formulate a forward kinematics method to bend and recover the grass. The grass-terrain collision is tested during this bending process.

(2) We propose a procedural method to handle the interactions with objects and wind. This method updates the continuum simulation (wave simulation) by directly setting the status of grass colliding with the objects and wind.

(3) Our model is animator friendly. Animators can easily control the simulation effects by setting the parameters for the wave propagation and the behaviors of the objects and wind.

All steps of our method are performed in GPU. Our proposed method provides a simple and effective framework that enables a real-time animation of grass field. The simplicity and generality of our method allowed it to be a practical solution for grass simulation. The proposed method has been integrated into a game engine and one of the results is shown in Figure 1.4.

1.4 Thesis organization

The remaining chapters of the thesis are organized as follows:

Some of the previous works about the simulation of nature scene and the grass animation are discussed in Chapter 2. Chapter 3 describes the modeling of grass and the way we distribute grass over the terrain. Chapters 4 and 5 describe the algorithms we proposed to animate grass in real-time. Chapter 6 describes the GPU implementation of each simulation step. Simulation results are presented in Chapter 7. Conclusion and directions for future research are given in Chapter 8.

Chapter 2

Related Work

Rest is not idleness, and to lie sometimes on the grass
under the trees on a summer's day, listening to the
murmur of water, or watching the clouds float across the
sky, is by no means a waste of time

John Lubbock

2.1 Simulation of nature scene

Nature scene is composed of many components, such as water, cloud, terrain, plant and etc. If we failed to model any one of them, the overall effect of the nature scene may be ruined and the users will not feel immersive. Moreover, the components of nature scene are usually related to each other. Some techniques used in modeling one component of the nature scene or one natural phenomena can be applied to modeling others. For example, the followings are based on the fluid simulation, Krištof et al. [KBKv09] modeled the

erosion effect of terrain and Miyazaki et al. [MDN02] simulated cloud. Grass is an important part of the nature scene. Knowing the techniques of simulating various nature scene and natural phenomena will be helpful in developing algorithm to animate grass. As such, we review some related work in modeling nature scene in this section.

2.1.1 Fluid

Fluid simulation is an active research area and an increasingly popular tool in computer graphics for generating realistic animations of water, smoke, fire and related natural phenomena. Fluid is volumetric phenomena. When each part of the volume interacting locally, it produces streams and rotational effects such as vortices and eddies. The well known Navier-Stokes equations can capture this behavior. It was introduced to computer graphics by Kayija and Herzen [KH84], which describes how a velocity field changes over time while applying pressure and forces. Foster and Metaxas were the earliest to apply the Navier-Stokes equations in full 3D [FM96]. Stam developed a semi-Lagrangian advection technique to provide unconditionally stable behavior of fluid [Sta99]. This method enables much larger time steps and thus increase the simulation speed. This technique was extended to handle complex 3D water simulations by Foster and Fedkiw [FF01] and by Enright et al. [EMF02] and Selle et al. [SFK⁺08]. Other most common methods are smoothed particle hydrodynamics methods [MCG03, HKK07, SP], vorticity-based methods [FSJ01], and Lattice Boltzmann methods [LWK03].

Smoke and fire are also fluid phenomena. Foster and Metaxas applied a solver of Navier-Stokes equations to approximate the motion of a hot, turbulent gas [FM97]. Fedkiw et al. [FSJ01] used the inviscid Euler equations which are usually more appropriate

for gas modeling and less computationally intensive than the Navier-Stokes equations used by others. Nguyen et al. [NFJ02] simulated the thin flames using the incompressible Navier-Stokes equations to independently model both vaporized fuel and hot gaseous products. For volumetric combustion, Melek and Keyser [MK02] used fluid simulation to model the motion of the fuel gases, oxidizing gases, exhaust gases, and heat that are necessary components of a flame simulation. Feldman et al. [FOA03] used the incompressible fluid equations and the partial system to model the motion of air and hot gases. Particles immersed in the fluid track the motion of particulate fuel and soot as they are advected by the fluid. Combustion is modeled using a process governed by the particle and fluid systems. Bridson [Bri08] gave an overview of fluid simulation in computer graphics.

2.1.2 Cloud

The conventional way to model cloud is using billboards. However, explicit physics simulation of cloud can be complicated. Dobashi et al. [DKY⁺00] proposed a procedural and computationally inexpensive method for animation of cloud. Miyazaki et al. [MYND01] modeled clouds using a method called the coupled map lattice (CML), which is developed based on the cellular automaton. Harris et al. [HBSL03] proposed flat 3D textures to simulate cloud on GPU. Miyazaki et al. [MDN02] simulated realistic clouds based on computational fluid dynamics. Dobashi et al. [DKNY08] extended Miyazaki et al.'s work [MDN02] to control the simulation and generation of realistic clouds. Realistic cloud is very important for flight simulation. Harris and Lastra [HL01] proposed a cloud shading algorithm that suitable for flight simulation and games. They approximated the multiple forward scattering at the pre-process time and used the first order anisotropic scattering

at the runtime. Wang [Wan03] developed an efficient and effective techniques for cloud simulation in Microsoft's Flight Simulator 2004.

2.1.3 Terrain

There are two main approaches to model terrain: fractal landscape modeling and physical erosion simulation. Variety of stochastic subdivision techniques have been introduced to fractal landscape modeling. To generate fractal terrain, Miller [Mil86] proposed a square-square subdivision scheme. Lewis [Lew87] proposed generalized stochastic subdivision. Szeliski and Terzopoulos [ST89] proposed to combine deterministic splines and stochastic fractals into constrained fractals. The fractal based approaches are reviewed in [EMP⁺02] and [Dac06].

Erosion based simulation is another approach to model terrain. This approach is based on stream erosion from the geomorphology community. It is often applied after a rough height field is generated. Kelley et al. [KMN88] proposed a method to simulate natural terrain by approximating the erosion of stream networks. Musgrave et al. [MKM89] proposed to combine the fractal modeling and erosion simulation approaches into a single system. Benes et al. [BTHB06] used Navier-Stokes equations to simulate a full 3D erosion. A wide variety of phenomena, such as concavities or receding waterfalls, can be modeled. Kristof et. al [KBKv09] simulated the erosion based on Smoothed Particle Hydrodynamics (SPH).

Terrain rendering provides methods to render the surface of earth. For out-door scene, the terrain usually is modeled using millions of triangles. As a result, the level

of detail(LOD) [Whi08], regular grid algorithms [LH04, AH04] and triangulated irregular networks [Hop98] algorithms are frequently employed to render large scale of terrain. Pajarola and Gobbetti [PG07] gave a survey of all the various semi-regular terrain LOD approaches.

2.1.4 Wind

To fully and physically simulate wind, Akagi et al. [AK06] applied a full fluid dynamics simulation, solving the Navier-Stokes equation. The cost is expensive. Some speed up calculations were applied to achieve real-time results. To avoid the fluid dynamics simulation, Shinya et al. [SF92] and Zhang et al. [ZST⁺06] proposed a stochastic approach (FFT based) from wind engineering that modeled wind as a velocity field. Stam [Sta97] developed a similar model by fluttering uncorrelated random velocity vectors for each branch in the frequency domain to achieve a correlation of loads on nearby branches. In the game of Crysis [EA08], with the SpeedTree engine, the wind animation system is entirely artist-controlled [Sou07]. The movement information of every single vertex on a plant was painted by hand which is tedious, however the results are impressive. There are other methods to model wind, such as a global wind direction [OFT⁺03], noise functions [Zio07, HKW09] and wind projection based [JD09].

2.1.5 Animation of trees

Besides grass, trees are also an important member of the vegetation family. However, trees are different from grass in terms of the material, geometric structure and shape.

For example, trees is usually modeled with joints and branches. The animation schemes of grass and trees are different.

Considering joints as structural elements, the explicit or simplified integration of the equations of motion can be applied directly [AK06, SF92, GCF01, SO99]. The modal analysis is a successful approach to create the animation of trees. Stam [Sta97] proposed to carry out the simulation in the frequency domain, by applying spectral methods to reduce the computational cost. Chuang et al. [CGZ⁺05] used spectral methods to model the animation of plants and trees. Haevre et al. [HFR06] applied a technique similar to motion graphs by pre-calculating a set of motion samples with a dynamic simulation and re-sequence the results at run time. Diener et al. [JD09] applied a finite element method to model the dynamics of trees. Ota et al. [OFT⁺03] and Sousa [Sou07] used noise functions to drive the animation. Habel et al. [HKW09] proposed a physics guided procedural animation.

2.1.6 GPU usage in simulation of nature scene

With conventional CPU-based algorithms, the simulation costs of natural phenomena effects are often too high to permit dynamic interactions. With the rapidly increasing computation power of modern GPU, more and more complex calculations to create realistic images can be done interactively. Computer games are the main driving force behind this development. Nowadays, almost every computer and game console are equipped with a dedicated GPU, thus the users can enjoy a high-performance, interactive and visually rich 3D scene. Much attention has been directed to use GPU to achieve interactive nature scene. The examples are the GPU applications in simulating fluid [HKK07],

cloud [JSK09], terrain [HBSL03], trees [AH04] and grass [HKW09]. More about the GPU programming will be discussed in Chapter 6.

2.2 Grass modeling and animation

In this section, we review some related work in grass modeling and animation.

2.2.1 Modeling of grass

Modeling of grass has been widely investigated. The major techniques to model a grass scene are particle system, volumetric representation and billboard representation.

FIGURE 2.1: Particle system [RB85].

Particle system

Reeves and Blau [RB85] used particles that move along parabolic trajectories to draw grass (Figure 2.1). The rendering cost is too high to be suitable for real-time applications.

FIGURE 2.2: Shell grass [BH02].

Volumetric representation

Kajiya and Kay introduced 3D texture mapping called "texel" to model furry surface [KK89]. Meyer and Neyret [MN98, Ney96] extended this work to use the z-buffer techniques to render 3D geometry that is sliced into a series of thin layers. A further extension called shell-based method used specifically for rendering fur and grass has been proposed by Lengyel et al. [LPFH01], Bakay and Heidrich [BH02], Banisch and Wuthric [BW06] (Figure 2.2). The shell-based method is based on geometry layers mapped with semi-transparent textures and sliced parallel to the terrain. There are other work using volumetric representation. Pelzer [Pel04] proposed to model grass by crossing several polygons with the configuration that looks like stars. This configuration can ensure good visual quality independent of the current line of sight. Decaudin and Neyret [DN04] proposed to use the volumetric textures and aperiodic tiling. The forest is represented using a set of edge-compatible prisms containing forest samples which are aperiodically mapped onto the ground. In general, the volumetric representation is efficient in rendering.

FIGURE 2.3: A grass field with three levels of details [BPB06].

3D geometric representation

Wang et al. [WWZ⁺05] used skeleton-lines to model grass blades. The number of skeleton-lines per blade dynamically decreases when the distance from the viewer increases. There are some works using the level of detail scheme to combine 3D geometry representation, volumetric representation and 2D texture of grass. Perbet and Cani [PC01] used chain of line-segmented polygons for 3D representation of grass and semi-transparent vertical textures of same orientation for volumetric representation of grass. Guerraz et al. [GPR⁺03] used many billboards for 3D representation of grass and semi-transparent vertical textures of different orientations for volumetric representation of grass. Boulanger et al. [BPB06] used a textured semi-transparent quadrilateral strips for 3D representation of grass and semi-transparent horizontal and vertical slices of textures for volumetric representation of grass (Figure 2.3).

FIGURE 2.4: A screen shot of the game Crysis (2008) [EA08] using billboards for vegetation modeling.

Other methods

Habel et al. [HWJ07] and J. Orthmann et al. [JSK09] and the game Crysis [EA08] (Figure 2.4) adopted the conventional billboards to model grass. There are other ways to model grass such as parametric representation and L-System by Prusinkiewicz et al. [PPH⁺96] which included the botanical factors. Deussen et al. [DHBL⁺98] used lines and points to represent grass.

2.2.2 Animation of grass

There are several work for modeling grass-wind interaction. Perbet and Cani [PC01] proposed a procedural animation controlled by wind primitives (Figure 2.5). Bakay and Heidrich [BH02] created an animation by translating the vertices of the shell grass according to a wind vector. Wang et al. [WWZ⁺05] combined the free form deformation and the explicit grass-grass collision detections (Figure 2.6). Endo et al. [EMF03] proposed a

FIGURE 2.5: Procedural grass-wind interaction [PC01]. (a) and (b) are the results, (c) is the wind primitives.

FIGURE 2.6: Grass-wind interaction based on free form deformation and explicit grass-grass collision detection [WWZ⁺05].

pre-computed physical model using dynamic constraints. Banisch and Wuthric [BW06] used a mass-spring system where the masses and springs are attached to the shell layers (Figure 2.7). Habel et al. [HWJ07] achieved a texture based animation by distorting the texture lookups (Figure 2.8). The conventional way to generate wind effects is by using the combination of sin and cos wave plus noise as in Pelzer [Pel04] and Zioma [Zio07]. Ramraj [Ram05] simulated the wind effects using a simple water wave (Figure 2.9).

FIGURE 2.7: Shell grass with a mass-spring system [BW06].

FIGURE 2.8: Texture based animation [HWJ07].

FIGURE 2.9: Wind effects using a simple water wave [Ram05].

Most animation techniques only addressed the grass-wind interaction. For the grass-object interaction, Guerraz et al. [GPR⁺03] presented a method to simulate the interaction between a vegetation and a moving object (Figure 2.10). This method can simulate a grass field of one million blades of grass at 2 fps with a SGI O2 platform. Most recently

Orthmann et al. [JSK09] proposed an GPU-based approach to model responsive grass. They handle the grass-object collisions by passing the vertices of grass to the geometry shader and using a cloth model based on spring constraints (Figure 2.11). This method can simulate 60000 billboards of grass at 40-80 fps with a platform of a AMD Athlon 64 3500+ 2.2 GHz processor and a GeForce 8800 GTX graphics card with 768 MB DDR3 memory. However, both these two methods did not clearly address the problem of the grass-grass interactions.

FIGURE 2.10: Object interacts with grass [GPR⁺03].

Most animation approaches do not consider that different types of grass have different properties, different types of objects and wind will create different impact on grass. We take into account all these factors. We integrate all interactions into a single model instead of the previous approaches that model wind and object interactions separately. Compared with other methods, our continuum grass can simulate larger scale of grass field (up to millions of clusters of grass at 40 fps) and we address all the grass-grass, grass-object, grass-terrain and grass-wind interactions. Table 2.1 shows the comparison of our method with other methods.

FIGURE 2.11: Object interacts with grass [JSK09].

TABLE 2.1: Comparison with other grass animation methods.

Method	Wind	Object	Grass	Terrain	Summary
Perbet and Cani [PC01]	Yes	No	No	No	Control using wind primitives.
Bakay and Heidrich [BH02]	Yes	No	No	No	Control using wind vectors.
Guerraz et al. [GPR ⁺ 03]	Yes	Yes	No	No	Animate blade by blade independently for nearby grass using control primitives and store the information of all bended grass.
Ramraj [Ram05]	Yes	No	No	No	Animate using a simple wave simulation.
Wang et al. [WWZ ⁺ 05]	Yes	No	Yes	No	Perform free-form deformation and explicit collision detection.
Banisch and Wuthric [BW06]	Yes	No	Yes	No	Simulate using a mass-spring model.
Habel et al. [HWJ07]	Yes	No	No	No	Distort grass texture lookup based on a procedural or hand-crafted texture.
Orthmann et al. [JSK09]	Yes	Yes	Yes	No	Simulate using a spring model. Refine the simulation mesh of every possibly affected grass during runtime.
Our method	Yes	Yes	Yes	Yes	Animate using continuum simulation and procedural method.

2.2.3 Hair and fur

There are similarities between hair, fur and grass. The techniques such as mass-spring, free form deformation and particle system have been used in simulating hair. Bertails et al. [BHC⁺08] gave an overview of the hair simulation. Hadap and Thalmann [HMT01] introduced a novel method to model dynamic hair as a continuum by performing a particle-based fluid simulation for lateral hair movement. Particles and hair strands are embedded in a fluid continuum. The hair-hair, hair-body and hair-air interactions are handled by using the fluid dynamics.

Chapter 3

Modeling Field of Grass

Until man duplicates a blade of grass, nature can laugh
at his so called scientific knowledge.

Thomas Edison

To model a grass field, we have to model a grass (Section 3.1) and determine how to distribute the grass in the terrain (Section 3.2).

3.1 Modeling of grass

There are two approaches to model grass: modeling a single blade of grass and a group of grass. For a single blade of grass, the conventional geometric models are cylinders and polylines. For a group of grass, the conventional way is to use billboards as the geometric model. Based on the observation that grass always forms clusters, we treat a cluster of grass as one simulation element and adopt the billboard representation (Figure 3.1).

By doing so, the computational cost can be reduced, which is important for real-time applications.

Our billboard grass is composed of different number of segments to represent different types of grass. For stiff grass (shrub-like grass) and short grass, one-segment billboard (Figure 3.1(a)) is used. This is because the shape of the grass does not deform much when bending. On the contrary, for soft grass and tall grass, multiple-segment billboard is used (Figure 3.1(b)). This representation can model more complex deformation of grass. To make our method more efficient, we also consider the level-of-details for grass. That is, for grass far away from the objects and camera, we use the one-segment billboard. Based on the simulation results, we bend the segments of the billboard. The simulation process will be discussed in detail in Chapter 4. The height of the grass (billboard) is determined based on the types of grass. We generate a random height offset for each grass to make the grass field looks natural. Each grass blade is slightly different in contrast and color to mimic the effect that the grass blades are occluded by each other and the lighting results will be slightly different.

3.2 Distribution of grass

It is not efficient to consider the entire terrain as one huge simulation region. Therefore, we divide the whole terrain into $m_p \times n_p$ patches of grass. The simulation is performed patch by patch. Each patch is further divided into $m_g \times n_g$ cells and we randomly place one cluster of grass (one billboard) within each cell (Figure 3.2). To increase the realism, we initialize the state of grass by randomly bend it. The randomness is based on the Perlin noise which is often used to model natural effects.

FIGURE 3.1: Use a billboard to model one cluster of grass. (a) is a one-segment billboard grass (stiff grass and short grass) and (b) is a multiple-segment billboard grass with four segments (soft grass and tall grass).

FIGURE 3.2: Example of a grass field. This grass field is divided into four patches, each patch has nine cells and each cell contains one cluster of grass.

Chapter 4

Real-Time Grass Simulation

Every blade of grass has it's Angel that bends over it
and whispers, "grow, grow."

Talmud

In this chapter, we present a method to simulate grass-grass interaction. The details on how to incorporate the grass-object and grass-wind interactions into the simulation will be explained in Chapter 5.

4.1 Basic idea and simulation steps

We treat a patch of grass (Section 3.2) as a continuum and perform the simulation patch by patch. That is, we perform the simulation on 2D cells. Each cell corresponds to one cluster of grass and stores two types of data: *bending angle* (the angle θ) and *bending velocity* (the angular velocity $\vec{\omega}_g$) (Figure 4.1). The bending angle θ is a scalar value that shows how much the cluster of grass has bended. Increasing θ means the cluster of grass

is bending. When θ is approaching to zero, it means that the grass is recovering. The bending velocity $\vec{\omega}_g$ is a 2D vector. The magnitude of $\vec{\omega}_g$ is the bending angular speed of this cluster of grass. The direction of $\vec{\omega}_g$ is its bending direction (this is 2D with respect to the 2D plane of the simulation cells).

FIGURE 4.1: Object interacts with (a) a one-segment billboard grass (stiff grass and short grass) and (b) a multiple-segment billboard grass (soft grass and tall grass).

The followings are the effects which we want to simulate: when moving objects and wind enter the grass field, they collide and bend the grass. The grass that collides with the objects and wind will bend first, then grass-grass collision occurs. Due to the collision, the bending is then propagated to other grass that is away from the objects. After the objects and wind passed, a track may be left on the field and the bended grass on the track will eventually recover.

We achieve the above effects by performing the following simulation steps (Figure 4.2):

FIGURE 4.2: Simulation steps.

1. Simulation region management: Manage the patches of grass where the continuum simulation is performed (Section 4.2).
2. Interaction with object and wind: This process includes collision detection and response. We detect the grass that collides with the object and wind. Then, we update the bending velocity $\vec{\omega}_g$ of the grass. This step will be discussed in details in Chapter 5.
3. Direct bending process: This step processes the bending effect caused by the grass-object and grass-wind interaction. We bend the clusters of grass colliding with the objects and wind by updating the bending angle θ (Section 4.3).

4. Propagation process: This step simulates the bending effect caused by the grass-grass interaction and the recovery effect of grass. We achieve these effects by simulating the propagation of values $(\theta, \vec{\omega}_g)$ inside the 2D cells (Section 4.4).
5. Animating the cluster of grass: This step illustrates how we use the values $(\theta, \vec{\omega}_g)$ to bend and recover one single cluster of grass during the rendering (Section 4.5).

We discuss these steps in details in the following sections and the GPU implementation in Chapter 6.

4.2 Simulation region management

Simulation regions are the patches of grass where we allocate memory and perform simulation. In some real-time applications such as games, the amount of memory to simulate a grass field might be limited. To cope with this issue, we propose the following method to manage the simulation regions.

- When an object entered the 2D cells of a patch of grass and this patch of grass is not in the simulation regions, we add this patch into the simulation regions.
- When the distance from the current position of the camera to the center of one patch of grass is less than a threshold d and this patch of grass is not in the simulation regions, we add this patch into the simulation regions. The threshold d can be set based on the camera's settings.

- If the memory consumption reaches the specified upper bound (225 patches in our experiment), we release the memory of the patch which is furthest from the current location of the camera and allocate it to the newly entered patch. Hence, the interactions that are far away from the camera may not be considered. In practice, this normally does not cause any problems, since these kind of interactions are not noticeable.

FIGURE 4.3: The dark small rectangles represent the moving object. We allocate the memory for the patches of grass no. 2, 3, 5, 6, 8 and 9. The circles represent the affected area of the object.

We increase the size of the 2D cells for one patch as $m_a \times n_a$, where $m_a \geq m_g$ and $n_a \geq n_g$, to overlap each other (Figure 4.3). This is to solve the problem that when the object is crossing to other patches of grass, the simulation is not continuous at the boundary. The size of the extended 2D cells of one patch ($m_a \times n_a$) should be large enough to record all potential interactions on this patch, even the interactions which are caused by objects outside the original patch ($m_g \times n_g$). However, when we make use of

the simulation result, we cut and use the center part (the original patch, $m_g \times n_g$) of the extended 2D cells of every patches of grass in the simulation regions.

In Figure 4.3, we can easily find that the object is currently located inside the extended 2D cells of patches no. 5, 8, 6 and 9. Then we consider allocating memory for patches no. 5, 8, 6 and 9 and simulate the interactions of the object on patches no. 5, 8, 6 and 9. The patch-by-patch simulation can be done in any order. Note that even the object is not in the original patches no. 5, 6 and 9, but we still simulate the interaction of the object on patches no. 5, 6 and 9. By doing so, the interaction is propagated from patch no. 8 to no. 5, 6 and 9.

Inside the simulation regions, we perform the natural wind simulation for all the patches that are near to the camera (based on the same distance checking using d in the previous part). For other patches that are far away from the camera (maybe inside or outside the simulation regions) we use only one patch of wind simulation result to tile these patches. The wind simulation will be discussed in details in Chapter 5.

4.3 Direct bending process

After the collisions are handled (details in Section 4.5), we have the new bending velocity $\vec{\omega}_g$ for the clusters of grass colliding with the object or wind. For simplicity, we omit the index of the cells. The direct bending process increases the bending angle θ of each colliding grass by the magnitude of the bending velocity $\|\vec{\omega}_g\|$. This new bending angle is later used to bend the bottom segment of grass. We denote the grass' bending angle at the simulation time t_s as $\theta(t_s)$, so for each simulation time step Δt_s , the new bending

angle of one cluster of grass that collides with the object and wind is

$$\theta(t_s + \Delta t_s) = \theta(t_s) + \|\vec{\omega}_g\| \Delta t_s. \quad (4.1)$$

To avoid the intersection between grass and the terrain, a maximum bending angle is calculated based on the height and slope of the terrain and the bending direction of the grass.

4.4 Propagation process

The grass-grass interaction (the bending and recovery of grass) is computed as a wave propagation inside a continuum (2D cells). Instead of performing the collision detections among grass, we propagate the values $(\theta, \vec{\omega}_g)$ stored in each cell to achieve the consecutive bending and recovery of grass. The values $(\theta, \vec{\omega}_g)$ are propagated from one cell to another cell. This implies that a cluster of grass is bending another cluster of grass. The values stored in cells are indeed energies. So in other words, we propagate the energy in continuum (Figure 4.4).

The energy of the continuum gained during the grass-object and grass-wind interactions will eventually go to zero after the energy flows inside the continuum. This energy damping effect of wave mimics the real world phenomenon. In real world, the friction between grass-grass, grass-air and grass-terrain makes the grass eventually stop moving and recover to its original state.

We treat the values stored in the cells, the bending angle and the bending velocity, as the amplitude of 2D waves. The wave functions we used are the same as in [Zel04]. For

FIGURE 4.4: Wave propagation.

FIGURE 4.5: Central difference approximations.

each simulation time t_s , $a_\theta(t_s)$ and $a_{\vec{\omega}_g}(t_s)$ are the accelerations of the wave propagation:

$$a_\theta(t_s) = \frac{\partial^2 \theta}{\partial t_s^2} = c_\theta(t_s)^2 \left(\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2} \right), \quad (4.2)$$

$$a_{\vec{\omega}_g}(t_s) = \frac{\partial^2 \vec{\omega}_g}{\partial t_s^2} = c_{\vec{\omega}_g}(t_s)^2 \left(\frac{\partial^2 \vec{\omega}_g}{\partial x^2} + \frac{\partial^2 \vec{\omega}_g}{\partial y^2} \right). \quad (4.3)$$

c_θ and $c_{\vec{\omega}_g}$ are the wave's propagation speed. After apply the central difference approximations (Figure 4.5):

$$\begin{aligned} a_\theta(t_s) = c_\theta(t_s)^2 \left(\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2} \right) &\approx (\theta_{(-1,0)} + \theta_{(1,0)} - 2\theta_{(0,0)}) \\ &+ (\theta_{(0,-1)} + \theta_{(0,1)} - 2\theta_{(0,0)}) \end{aligned} \quad (4.4)$$

$$\begin{aligned} a_{\vec{\omega}_g}(t_s) = c_{\vec{\omega}_g}(t_s)^2 \left(\frac{\partial^2 \vec{\omega}_g}{\partial x^2} + \frac{\partial^2 \vec{\omega}_g}{\partial y^2} \right) &\approx (\vec{\omega}_{g(-1,0)} + \vec{\omega}_{g(1,0)} - 2\vec{\omega}_{g(0,0)}) \\ &+ (\vec{\omega}_{g(0,-1)} + \vec{\omega}_{g(0,1)} - 2\vec{\omega}_{g(0,0)}) \end{aligned} \quad (4.5)$$

For one cluster of grass, the c values control the propagation speed of its θ and $\vec{\omega}_g$ to its neighbors. For example, a stiff grass will cause the neighboring grass to react faster so its c_θ and $c_{\vec{\omega}_g}$ are large. c_θ is also set based on the grass' bending speed $\|\omega_g(t_s)\|$ at t_s :

$$c_\theta(t_s) = c_0 + \mu \|\vec{\omega}_g(t_s)\|. \quad (4.6)$$

c_0 is the base wave propagation speed for θ . When one cluster of grass has the bending velocity of $\vec{\omega}_g(t)$, μ controls how its velocity contribute to the changing and propagation of its θ . For example, the grass with faster bending velocity propagates θ faster to its neighbors.

Based on $a_\theta(t_s)$ and $a_{\vec{\omega}_g}(t_s)$, we apply the verlet integration method to compute the new status of the grass. This method is simple to implement, accurate and stable to model continuum dynamics. We denote the values $(\theta, \vec{\omega}_g)$ at each simulation time t_s as

$(\theta(t_s), \vec{\omega}_g(t_s))$, for each simulation time step Δt_s , we have:

$$\begin{aligned}\theta(t_s + \Delta t_s) &= (2 - \lambda_\theta)\theta(t_s) - (1 - \lambda_\theta)\theta(t_s - \Delta t_s) \\ &+ a_\theta(t_s)(\Delta t_s^2),\end{aligned}\tag{4.7}$$

$$\begin{aligned}\vec{\omega}_g(t_s + \Delta t_s) &= (2 - \lambda_{\vec{\omega}_g})\vec{\omega}_g(t_s) - (1 - \lambda_{\vec{\omega}_g})\vec{\omega}_g(t_s - \Delta t_s) \\ &+ a_{\vec{\omega}_g}(t_s)(\Delta t_s^2),\end{aligned}\tag{4.8}$$

where λ_θ and $\lambda_{\vec{\omega}_g}$ are the damping parameters to control how much the amplitude of wave is decreased when the wave propagates. If λ is zero, it means the wave propagation will never stop.

By using the wave simulation, we provide a convenient way to control the grass animation. We can tune the affected area of grass-grass interaction by setting the λ (smaller λ : large area, greater λ : small area) and control how fast a grass propagates the bending by setting the c (smaller c : slow propagation, greater c : fast propagation). In our experiments, c_0 is in the range $[1, 10]$, μ is in the range $[0, 1.5]$, $c_{\vec{\omega}_g}$ is in the range of $[1, 10]$, λ_θ and $\lambda_{\vec{\omega}_g}$ are in the range of $[0.01, 1]$.

4.5 Animating the cluster of grass

The animation for a single cluster of grass are the bending and the recovery process. In the above continuum simulation step, the bending angle θ and the bending direction $\vec{\omega}_g$ have been calculated. Based on the values stored in the 2D cells, we bend or recover the billboard of cluster of grass when we render it. We explain our method by using tall grass as an example. Since the grass is tall, we adopt the multiple-segment billboard

FIGURE 4.6: The bending process of one cluster of grass. Grass bends from the bottom to the top.

representation of the grass and we assume that the object hits the bottom segment of the billboard.

Based on the observation that when the collision happens at the bottom of grass, the bottom bends while dragging the top to bend (Figure 4.6). A physics-based simulation has been done to verify this effect. We use θ as the bending angle for the bottom segment of the billboard. Then, we propagate θ from the bottom segment to the upper segments of grass. In other words, the upper segments bend later than the bottom and the bending angles of the upper segments can be considered as the previous θ of the bottom segment. For n segments $\{L_0 \dots L_{n-1}\}$ of grass at position P , the angles at each rendering time t_r are $\{\theta_0(t_r)_P \dots \theta_{n-1}(t_r)_P\}$:

$$\theta_i(t_r)_P = \theta_0(t_r - \gamma_i)_P, \quad (0 \leq i \leq n-1) \quad (4.9)$$

where γ_i is the factor to control the time of the upper segments to start to bend relative to the time when the bottom segment bends. γ for the bottom segment (γ_0) is zero. If the segment is closer to the top, it has greater γ .

For applications whose memory consumption is not an important issue, we store the

bending angles at several time steps and use them to evaluate Equation 4.9. However, for applications such as simulators or games, less memory consumption is important. Therefore, we also propose the following alternative. We do not store many previous states of the grass and only pass the latest result of the continuum simulation to the rendering process. We then perform the following approximated computation.

FIGURE 4.7: During time γ_1 , the θ value propagates from P to Q . The θ value also propagates from the bottom to the upper segment at P .

Consider the wave propagation from the position P to the position Q during time γ_i . The θ value stored in the position Q is actually the θ value of the grass at P at γ_i time ago (Figure 4.7). The direction of wave propagation at P is the bending direction $(\frac{\vec{\omega}_o}{\|\vec{\omega}_o\|})$ of the grass at P . For the i -th segment of the grass at P , $\theta_i(t_r)_P$ is approximated as

$$\theta_i(t_r)_P = \theta_0(t_r - \gamma_i)_P \approx \theta_0(t_r)_Q, \quad (4.10)$$

$$Q = P + \gamma_i c_\theta(t_r)_P \frac{\vec{\omega}_o}{\|\vec{\omega}_o\|} \quad (4.11)$$

where $c_\theta(t_r)_P$ is the wave propagation speed at cell P and is computed using Equation 4.7. $\gamma_i c_\theta(t_r)_P$ approximates the distance the wave travels during γ_i . In our experiments, γ_i is in the range $[0, \frac{\pi}{2\|\vec{\omega}_o\|}]$. Since θ is damped during the wave propagation (Section

4.5), we get the damped value of $\theta_0(t_r - \gamma_i)_P$. This gives the side effect of the damping of energy when bending the grass from bottom to top, due to frictions between grass-grass and grass-air. The limitation of this approach is that when other object or wind interacts with the grass at the position Q , then we may not get the correct previous θ value. However, from experimental results, we can create grass animation without any visual defects.

FIGURE 4.8: Based on θ and L (the height of the segment), apply the forward kinematics.

The bending of billboard is in 3D. Using the forward kinematics (Figure 4.8), we move vertices of the segment of the billboard along the bending direction ($\frac{\vec{\omega}_o}{\|\vec{\omega}_o\|}$). If the θ value is less than zero, we bend the cluster of grass in the opposite direction of $\frac{\vec{\omega}_o}{\|\vec{\omega}_o\|}$ with angle $\|\theta\|$. To avoid the upper segments of the grass colliding with the terrain, we compare the height of the segment with the height of the terrain.

For grass' recovery, we perform the similar computation as grass' bending. The lower parts recover first while dragging the upper parts' to recover. Based on the observation, the recovery of the upper parts may not be delayed much, so in the recovery process, we use smaller values for γ_i in Equation 4.9.

Chapter 5

Interaction with Objects and Wind

Grass is the forgiveness of nature - her constant
benediction. Forests decay, harvests perish, flowers
vanish, but grass is immortal.

Brian Ingalls

The grass-object and grass-wind collision directly update the bending velocity of the grass that collides with the object and wind. First, we describe grass interacting with moving objects, such as moving cars (Sections 5.1 and 5.2). Then, we introduce grass interacting with wind (Section 5.3).

5.1 Collision detection

This process performs the grass-object collision detection. First, we extrude the shape of the object along its moving direction according to the object's speed (Figure 5.1). The

amount of extrusion for each simulation time step Δt_s is $\|\vec{v}_o\| \Delta t_s$, where $\vec{v}_o$ is the object's velocity. Then, we project the extruded object in world space into its local position in 2D cells of the simulation region to determine the collision area.

FIGURE 5.1: Calculate the collision area.

5.2 Collision response

In this process, we set the bending velocity ($\vec{\omega}_g$) for clusters of grass in the collision area. With the new $\vec{\omega}_g$, we can update the continuum simulation in Chapter 4.

To compute the bending direction, we take into account the shape of the collision area and the behavior of the object. For example, the grass on the right side of the object will bend toward the right. To model this property, we define two types of vectors for each cell in the collision area, namely *shape vector* ($\vec{M}_P$) and *moving vector* ($\vec{T}_P$) (Figure 5.2). We add them to determine the bending direction.

The shape vectors are created based on the shape of collision area (Figure 5.2). Suppose the position of the cell in the collision area is P . The shape vector $\vec{M}_P$ is defined as

FIGURE 5.2: Assume that the object has a rectangular shape. The calculation of the bending direction for the grass in collision area.

follows.

$$\vec{M}_P = \begin{cases} \vec{N}_P & \text{if } \vec{N}_P \cdot \vec{v}_o \geq 0 \text{ and } P \text{ is at boundary,} \\ \vec{0} & \text{if } \vec{N}_P \cdot \vec{v}_o < 0 \text{ or } P \text{ is not at boundary,} \end{cases} \quad (5.1)$$

where $\vec{v}_o$ is the object's velocity and $\vec{N}_P$ is the normal vector at the point P .

The moving vectors are created based on the object's moving behaviors. For example, the object bends the grass in its moving direction, therefore grass in the collision area has the moving vector of $\frac{\vec{v}_o}{\|\vec{v}_o\|}$ (Figure 5.2). For the cell at position P , the moving vector is $\vec{T}_P$, which is normalized, thus the bending direction $\vec{d}_{gP}$ is:

$$\vec{d}_{gP} = \frac{\vec{M}_P + \vec{T}_P}{\|\vec{M}_P + \vec{T}_P\|}. \quad (5.2)$$

The bending speed is computed as follows. We consider the vertical plane in the bending direction of the grass and illustrate our calculation of angular speed in Figure 5.3. The grass's horizontal speed is the same as the projected object's horizontal speed onto the bending direction s_o . θ is the bending angle of the grass. So, for the cell at position P , we can compute the grass's tangential speed s_g and the bending velocity $\omega_{gP}^{\vec{}}$

FIGURE 5.3: The calculation of the bending velocity of the grass in the collision area.

from the object's horizontal speed s_o by

$$s_g = s_o \cos \theta, \quad \omega_{gP}^{\rightarrow} = d_{gP}^{\rightarrow} \frac{s_g}{h / \cos \theta}, \quad (5.3)$$

where h is the height from the ground where the collision takes place. The value of h is set by the animator for each object.

5.3 Interaction with wind

In this thesis, we use two approaches to simulate the wind effects. The first approach simulates wind such as cyclone or wind caused by an object in the air, such as helicopter (Figures 5.5(a) and 5.5(b)). We treat this type of wind as an object moving in the grass field. To update the continuum simulation, we set the location, the shape and the cell values of the collision area based on the location of the source of wind, the strength of the wind and how the wind bends the grass (bending direction). For example, as a helicopter moves upwards, the wind is weaker, the size of the collision area and the bending speed becomes smaller. To model various types of wind, animators can design different shape of collision area, the strength of the wind and the bending direction. Figures 5.4(a) and

FIGURE 5.4: Two examples of the shape of the collision area and the bending direction designed by an animator.

FIGURE 5.5: A helicopter scene using the bending directions in (a) Figure 5.4(a) and (b) Figure 5.4(b).

5.4(b) show two examples of the shape of the collision area and the bending direction designed by an animator.

The second approach simulates the effect caused by natural wind (Figure 5.6) that is similar to the method proposed by Ramraj [Ram05]. We perform the natural wind simulation for the patches of grass that are near to the camera. We perturb the values

FIGURE 5.6: Grass field in natural wind.

$(\theta, \vec{\omega}_g)$ of the grass, then we use the wave simulation to propagate the effects. The amount of perturbation can be controlled using a Perlin noise or any user defined noise. For efficiency, we simulate the natural wind effect on one patch of grass and use the result for the patches that are far from the camera.

Chapter 6

GPU Implementation

Any activity becomes creative when the doer cares
about doing it right, or doing it better.

John Updike

In Section 6.1, some basic and related GPU techniques are introduced and in Section 6.2 our implementation of all the simulation steps is presented.

6.1 GPU programming

6.1.1 Introduction

Graphic Processing Units (GPU) is placed on the graphics card as a data-parallel computing device to accelerate 3D applications. GPU has major advantages compared to current Central Processing Unit (CPU) whenever:

FIGURE 6.1: Programmable graphics pipeline [DR08].

- Compute-intensive floating point arithmetics,
- Massively parallel,
- Flow control mechanisms are less frequently used.

These are exactly what are required for calculating the graphics on the screen. GPU functionality has, traditionally, been very limited. Starting from Microsoft's DirectX 8.0 introducing Shader Model 1.1 to the current Shader Model 4.0 and from OpenGL 1.5 introducing OpenGL Shading Language (GLSL) to the current GLSL 1.40, each shader model and GLSL model increased the programming model flexibilities and capabilities. Figure 6.1 illustrates the programmable graphics pipeline. Vertex shaders allow the programmer to alter per-vertex attributes, such as position, color, texture coordinates and normal vector. Pixel (Fragment) shaders are used to calculate the color of a pixel. Programmable vertex and pixel shaders were added to the graphics pipeline to enable

programmers to generate even more realistic effects, such as lens effects, displacement mapping and depth of field.

With the the advances in the development of GPU, general purpose programming on graphics processing units (GPGPU) receives a lot of attention. GPGPU is the technique of using a GPU to perform computation in applications conventionally handled by the CPU. Software developers can use stream processing on non-graphics data, because the programmable stages were added to the graphics pipeline and the current GPU can handle higher precision floating point arithmetic than before.

6.1.2 GPU programming concepts

GPU is a stream processor that can operate in parallel by running a kernel on many vertices or pixels in a stream at once. A stream is simply a set of vertices or pixels that require similar computation. Streams provide data parallelism. Kernels are the shader operations that are applied to each vertex or pixel in the stream. The computational resources on GPU are:

- Programmable processors: vertex, primitive and pixel pipelines allow programmer to perform kernel on streams of data.
- Rasterizer: creates pixels and interpolates per-vertex constants such as normals, texture coordinates and color.
- Texture Unit: read-only graphical memory interface.
- Framebuffer: write-only graphical memory interface.

Actually, a write-only texture can be substituted for output instead of the framebuffer. This is accomplished either through Render-To-Texture, Render-To-Backbuffer-Copy-To-Texture (RTBCTT), or the more recent stream-out. For GPU, the common form of a stream is a texture (2D grids) because this fits naturally with the rendering model built into GPU. The texture fetchings are used as memory read because the textures are used as memory. Commonly used operations in shader are highly optimized for graphics processing, such as vector and matrix functions, interpolation, texture functions and etc. The flow control structures also have recently been added to GPU.

6.2 GPU implementation of simulation steps

All the simulation steps are performed in GPU. The states of the grass is stored as textures in GPU memory. The simulation is implemented using the standard texture manipulation techniques, such as rendering to texture and texture fetching. We only pass the object's position, shape and velocity from CPU to GPU.

Simulation region's representation

2D cells for each patch of grass is represented as one texture named as *simulation texture*. Each cell corresponds to one pixel in this texture. The resolution of simulation texture in pixels is at least the same as the resolution of the 2D cells. Lower resolution will reduce the quality of simulation. Each pixel of simulation texture contains three channels, the first channel stores the cluster of grass's bending angle θ and the other two channels store its bending velocity $\vec{\omega}_g$. All these values are floating point numbers.

Collision detection

When the object moves one time step, we calculate the collision area based on the objects' velocity and shape. Then, we render the collision area to the simulation textures. The pixels inside the collision area represent the clusters of grass colliding with the object.

Collision response

Collision response is to set the $\vec{\omega}_g$ in the collision area. We implement this process as shading the pixels when rendering the collision area to the simulation textures. Specifying each pixel's $\vec{\omega}_g$ in pixel shader for the whole collision area is expensive. Therefore, we divide the collision area into small regular shapes. The $\vec{\omega}_g$ is then computed only at the vertices of these small regular shapes and the Gouraud shading (interpolation) is performed to shade the remaining pixels.

Direct bending process

This process is to add the magnitude of the 2D vectors represented by the second and third channels ($\vec{\omega}_g$) to the first channel (θ) in the pixel shader.

Propagation process

The implementation of the wave simulation is based on [Zel04]. The wave simulation uses three textures. We perform texture fetching and rendering to texture techniques in pixel shader. The wave function is discretized using the central difference approximation. To calculate the next state of the simulation $c(t + \Delta t)$, we need to sample the previous state of simulation $c(t - \Delta t)$ and the current state of simulation $c(t)$. In every time step, we perform the wave simulation and then switch the textures by changing their indices.

Then, we clear $c(t + \Delta t)$ and set $c(t + \Delta t)$ as the render target. This process is described in Figure 6.2.

FIGURE 6.2: Switch the simulation textures.

Animating the cluster of grass

Animating the billboard of a cluster of grass is performed in the vertex shader during the rendering. We fetch the values $(\theta, \vec{\omega}_g)$ from the simulation texture and use them to bend or recover the billboard by moving the vertices of each segment of the billboard. To examine the grass-terrain interaction, we compare the heights of the grass's segments of billboards with the height of terrain, which is also stored as a texture.

Chapter 7

Results

However beautiful the strategy, you should occasionally look at the results.

Winston Churchill

Figures 1.4, 5.5, 5.6 and 7.1-7.9, show various simulation results. We perform our experiments on two GPUs, one is NVIDIA GeForce 8600GTS with 256M DDR3 Memory (GPU-1) and the other one is ATI Radeon HD4800 with 512M DDR3 Memory (GPU-2) in order to measure the performance of our method on different classes of GPUs. We integrated our method in a racing game which was developed on the DirectX 9.0c platform.

The size of the terrain in the experiment is 15840 feet by 15840 feet and is divided into 88 by 88 patches. Each patch contains 120 by 120 cells. In total, there are more than 100 million clusters of grass in the terrain. We maintain simulation regions that consist of 225 patches of grass. Each patch (extended 2D cells) of grass is represented as one simulation texture with the resolution of 128 by 128 pixels. Each pixel contains

three channels (R,G,B) and each channel is 16 bits. Each patch is overlapped with the neighboring patches in 4 cells length (6 feet) in each direction. For applications that require a very fast and far propagation (very small λ and very large c), a larger area of overlapping among patches needs to be set. Each continuum simulation needs 3 such textures. All together, we use 675 textures for simulation, which is around 63 M of the GPU memory.

Maintaining 225 patches (180 feet by 180 feet per patch) as the simulation regions is usually sufficient to simulate a large interactive grass field. In general, the number of patches maintained for simulation is decided based on the number of objects in the simulation and the settings of camera, for example, if the number of objects in the simulation or the size of the viewing frustum of the camera is small, less patches are required for simulation.

In the simulation regions, maintaining 4 patches (360 feet by 360 feet) around the camera to perform the natural wind simulation is usually sufficient. The reason is because the grass outside this area (360 feet by 360 feet) is usually occluded by nearby grass, and in the case of "bird-view", the camera can see a large area of grass, however the camera is also far from the terrain. As we mentioned in Section 4.2, for the patches of grass that are far from the camera, we use one patch of wind simulation result to tile these patches.

In the game, with these settings, we can simulate grass-grass, grass-terrain, grass-object and grass-wind interactions on the grass field with a frame rate around 40 fps on GPU-1 and 100 fps on GPU-2. Note that the bending effect in our results due to grass-grass interaction cannot be achieved by simply randomly bending and recovering the grass. The main factors that affect the simulation speed are the number and resolution of patches maintained for the simulation, the number of objects in the simulation and

the number of segments in the segmented billboard. Table 7.1 presents how these factors affect the performance of the game on GPU-1 and GPU-2. The first row in the table shows the default settings and in the rest of the rows, (*) show the difference in parameter settings compared to the default one.

TABLE 7.1: Performance of the game with different settings on GPU-1 and GPU-2.

Number of patches	Resolution of patches (pixels)	Number of objects	Number of segments	GPU-1	GPU-2
225	128 by 128	10	4	40	100
400 (*)	128 by 128	10	4	15	40
225	256 by 256 (*)	10	4	15	40
225	128 by 128	20 (*)	4	20	50
225	128 by 128	10	5 (*)	35	80

(a)

(b)

(c)

(d)

FIGURE 7.1: Animation sequence with one car and one-segment billboard grass.

FIGURE 7.2: Animation sequence with one car and multiple-segment billboard grass.

FIGURE 7.3: Concave object.

One-segment billboard grass One-segment billboard is adopted to model the short and stiff (shrub-like) grass (Figure 7.1).

Multiple-segment billboard grass Multiple-segment billboard is adopted to model

FIGURE 7.4: Animation sequence with multiple cars.

the tall and soft grass (Figure 7.2). For multiple-segment billboard grass, the bottom will drag the top to bend.

Concave object As we discussed in Chapter 5, the shape vectors are created based on the shape of collision area. For complex or concave object, we may divide its collision area into small regular shapes. Figure 7.3 shows the animation result of a concave object moving in the grass field.

Multiple objects The grass field can interact with multiple objects (Figure 7.4).

Helicopter To simulate a helicopter interacts with the grass field is another important application of our algorithm (Figure 7.5).

Controlling the simulation The simulation parameters are set and tuned in the experiments. By tuning the parameters, we can achieve many types of interactions to

FIGURE 7.5: Animation sequence with helicopter with the bending direction as in Figure 5.4(b).

match different needs or artists' visions in the animation results. For example, we can tune the affected area of grass-grass interaction by setting the λ (smaller λ : large area, greater λ : small area) (Figures 7.6 and 7.7) and control how fast a grass propagates the bending by setting the c (smaller c : slow propagation, greater c : fast propagation) (Figures 7.8 and 7.9).

FIGURE 7.6: Control the propagation area. (a, c, e) Smaller λ : large area. (b, d, f) Greater λ : small area.

Comparison with the previous methods Compared with Guerraz et al. [GPR⁺03] and Orthmann et al. [JSK09], our method presents many types of interactions, while they only showed the grass-wind and grass-object interactions. They did not show clearly

FIGURE 7.7: Control the propagation area (closer view). (a, c, e) Smaller λ : large area. (b, d, f) Greater λ : small area.

the grass-grass and grass-multiple objects interactions. Furthermore, we consider that different types of grass have different properties and different types of objects and wind will create different impact on grass and demonstrate how to control the grass simulation. However, they did not mention these. Guerraz et al. [GPR⁺03] performed the simulation blade by blade and stored the information for all the bended grass (Figure 7.10 and the submitted video). The computational cost is very high, especially in the case when multiple objects are moving in a dense grass field leaving a lot of tracks. This method may not be suitable for real-time applications such as games. Compared with the images

FIGURE 7.8: Control the propagation speed. (a, c, e) Smaller c : slow propagation. (b, d, f) Greater c : fast propagation.

from Orthmann et al.'s responsive grass [JSK09] (We contacted the authors, however they did not provide us with the video.), in our opinion, the bending and recovery of our grass look smoother (Figure 7.11). In their images, the grass-object interaction is not clear.

FIGURE 7.9: Control the propagation speed (closer view). (a, c, e) Smaller c : slow propagation. (b, d, f) Greater c : fast propagation.

Moreover, we can simulate much larger fields of grass. We can simulate up to millions of clusters of grass (Note that not all of the simulated clusters of grass are visible at one point of time), while [JSK09] can simulate 60000 billboards of grass.

(a)

(b)

FIGURE 7.10: Comparison with Guerraz et al. [GPR⁺03]. (a) Images from [GPR⁺03].
(b) Our result.

(a)

(b)

FIGURE 7.11: Comparison with Orthmann et al. [JSK09]. (a) Images from [JSK09].
(b) Our result.

Chapter 8

Conclusion and Future Work

Basic research is what I am doing when I don't know
what I am doing.

Wernher von Braun

8.1 Conclusion

There is an increasingly demand for modeling large and interactive natural scene in real-time. However, as an important component of natural scene, grass field is still lack of interactivity. In this thesis, we have presented a method for real-time simulation of grass for games and real-time applications such as outdoor training simulations, driving simulators and flight simulators. Our method to simulate grass integrates four types of interaction: grass-grass, grass-object, grass-terrain and grass-wind. We proposed a continuum representation for grass. The grass-grass interaction is considered as the dynamics of the continuum. To efficiently model the grass-object, grass-wind interactions and to

propagate the effects in a grass field, we proposed the procedural collision detection, response and the wave simulation, respectively. We formulate a forward kinematics method to bend and recover the grass. The grass-terrain collision is handled during this bending process. Our method is suitable for GPU implementation. In addition, our method allows the user to control the simulation effects by setting several intuitive parameters. Our results shows that it is possible to efficiently animate large scale of grass field with millions of blades of grass. Compared with other methods, our method addresses all the grass-grass, grass-object, grass-terrain and grass-wind interactions, our method considers the grass' properties as well as the objects and wind's behaviors and our method can animate larger scale of grass field. We believe the presented technique is well adapted to real-time applications such as computer games and simulators.

8.2 Limitations

Since our method does not perform an explicit collision detection among the grass, in a highly dense grass field, we may see the billboards intersecting each other. Usually this problem can be alleviated by tuning the c , λ and γ values. Currently, we determine these values experimentally. Since our method does not perform an explicit complex physics simulation, the computed grass bending angles and velocities are not accurate. However, the proposed method can simulate the essential effects of grass field and this is generally sufficient for real-time applications. Since we use billboards in the rendering process, the grass field does not look nice from the top view.

8.3 Future work

In the future, we are interested in implementing our method using CUDA and investigating the performance. Some particle and motion blur effects can be added to enhance the rendering effects. We also want to determine the optimal values of the parameters (c , λ and γ values) based on the botanical property of the grass. Another possible future work is to explore an alternative modeling approach for grass (rather than billboards) that can be combined with our current simulation method. Self-shadowing can improve the lighting effect. However, incorporating self-shadowing for a large number of blades of grass usually requires high computational cost. For example, to simulate a large number of blades of grass moving and casting shadows on each other, a large number of dynamic shadow calculations have to be performed. We plan to develop an efficient self-shadowing algorithm for a large scale grass field.

Appendix A

Publications

Kan Chen and Henry Johan. Real-Time Animation of Grass. In *I3D '09: Proceedings of the 2009 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (poster)*, 2009.

Kan Chen and Henry Johan. Real-time Continuum Grass. In *VR '10: Proceedings of the IEEE Virtual Reality 2010 (full paper)*, 2010.

Bibliography

- [AH04] Arul Asirvatham and Hugues Hoppe. Terrain rendering using GPU-based geometry clipmaps. *GPU Gems 2*, pages 37–45, Matt Pharr, 2004.
- [AK06] Yasuhiro Akagi and Katsuhiro Kitajima. Computer animation of swaying trees based on physical simulation. *Computers and Graphics*, 30(4):529–539, 2006.
- [BH02] Brook Bakay and Wolfgang Heidrich. Real-time animated grass. In *Proceedings of Eurographics (short paper) 2002*, 2002.
- [BHC⁺08] Florence Bertails, Sunil Hadap, Marie-Paule Cani, Ming Lin, Tae-Yong Kim, Steve Marschner, Kelly Ward, and Zoran Kačić-Alesić. Realistic hair simulation: animation and rendering. In *SIGGRAPH '08: ACM SIGGRAPH 2008 Classes*, pages 1–154, 2008.
- [BPB06] Kévin Boulanger, Sumanta Pattanaik, and Kadi Bouatouch. Rendering grass terrains in real-time with dynamic lighting. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 46, 2006.
- [Bri08] Robert Bridson. *Fluid simulation for computer graphics*. A K Peters, 2008.

- [BTHB06] Bedřich Beneš, Václav Těšínský, Jan Hornyš, and Sanjiv K. Bhatia. Hydraulic erosion: Research articles. *Computer Animation and Virtual Worlds*, 17(2):99–108, 2006.
- [BW06] Sven Banisch and Charles A Wuthric. Making grass and fur move. *Journal of WSCG 2006*, 14:25–32, 2006.
- [CGZ⁺05] Yung-Yu Chuang, Dan B Goldman, Ke Colin Zheng, Brian Curless, David H. Salesin, and Richard Szeliski. Animating pictures with stochastic motion textures. *ACM Transactions on Graphics*, 24(3):853–860, 2005.
- [Dac06] Carsten Dachsbacher. *Interactive terrain rendering: Towards realism with procedural models and graphics hardware*. PhD thesis, Universitätsbibliothek, 2006.
- [DHBL⁺98] Oliver Deussen, Pat Hanrahan, Radomír Měch Bernd Lintermann, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 275–286, 1998.
- [DKNY08] Yoshinori Dobashi, Katsutoshi Kusumoto, Tomoyuki Nishita, and Tsuyoshi Yamamoto. Feedback control of cumuliform cloud formation based on computational fluid dynamics. *ACM Transactions on Graphics*, 27(3):1–8, 2008.
- [DKY⁺00] Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. A simple, efficient method for realistic animation of

- clouds. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 19–28, 2000.
- [DN04] Philippe Decaudin and Fabrice Neyret. Rendering forest scenes in real-time. In *Rendering Techniques '04: Proceedings of Eurographics Symposium on Rendering*, pages 93–102, 2004.
- [DR08] Antonio Deledda and Florian Ries. The graphics processing unit, http://www-micro.deis.unibo.it/~guerrier/Dida01/Seminari_2008/seminario_GPU.pdf. Seminar, 2008.
- [EA08] EA. Crysis, 2008.
- [EMF02] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Transactions on Graphics*, 21(3):736–744, 2002.
- [EMF03] Luis Carlos Yano Endo, Carlos Hitoshi Morimoto, and Antonio Elias Fabris. Real-time animation of underbrush. *Journal of WSCG 2003 (short paper)*, 11, 2003.
- [EMP⁺02] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and modeling: a procedural approach*. Morgan Kaufmann, 2002.
- [FF01] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer graphics and Interactive Techniques*, pages 23–30, 2001.

- [FM96] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. In *GI '96: Proceedings of the Conference on Graphics Interface '96*, pages 204–212, 1996.
- [FM97] Nick Foster and Dimitris Metaxas. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 181–188, 1997.
- [FOA03] Bryan Eric Feldman, James F. O'Brien, and Okan Arikan. Animating suspended particle explosions. In *SIGGRAPH '03: Proceedings of the 30th Annual Conference on Computer Graphics and Interactive Techniques*, pages 708–715, 2003.
- [FSJ01] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 15–22, 2001.
- [GCF01] Thomas Di Giacomo, Stéphane Capo, and François Faure. An interactive forest. In *Computer Animation and Simulation 2001: Proceedings of the Eurographics Workshop*, page 65, 2001.
- [GPR⁺03] Sylvain Guerraz, Frank Perbet, David Raulo, François Faure, and Marie-Paule Cani. A procedural approach to animate interactive natural sceneries. In *CASA '03: Proceedings of the 16th International Conference on Computer Animation and Social Agents*, page 73, 2003.
- [HBSL03] Mark Jason Harris, William Valentine Baxter, Thorsten Scheuermann, and Anselmo Lastra. Simulation of cloud dynamics on graphics hardware. In

- HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 92–101, 2003.
- [HFR06] William Van Haevre, Fabian Di Fiore, and Frank Van Reeth. Physically-based driven tree animations. In *Eurographics Workshop on Natural Phenomena*, 2006.
- [HKK07] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed particle hydrodynamics on GPUs. In *CGI '07: Proceedings of Computer Graphics International 2007*, pages 63–70, 2007.
- [HKW09] Ralf Habel, Alexander Kusternig, and Michael Wimmer. Physically guided animation of trees. In *Computer Graphics Forum*, volume 28, pages 523–532, 2009.
- [HL01] Mark J. Harris and Anselmo Lastra. Real-time cloud rendering. *Computer Graphics Forum*, 20(3):76–84, 2001.
- [HMT01] Sunil Hadap and Nadia Magnenat-Thalmann. Modeling dynamic hair as a continuum. *Computer Graphics Forum*, 20(3):329–338, 2001.
- [Hop98] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *VIS '98: Proceedings of the Conference on Visualization '98*, pages 35–42, 1998.
- [HWJ07] Ralf Habel, Michael Wimmer, and Stefan Jeschke. Instant animated grass. *Journal of WSCG 2007*, 15:123–128, 2007.
- [Int08] Empire Interactive. Flatout ultimate carnage, 2008.

- [JD09] Lionel Baboud Lionel Reveret Julien Diener, Mathieu Rodriguez. Wind projection basis for real-time animation of trees. In *Computer Graphics Forum*, volume 28, 2009.
- [JSK09] Orthmann Jensn, Christof Rezk Salama, and Andreas Kolb. GPU-based responsive grass. In *Journal of WSCG 2009*, 2009.
- [KBKv09] Peter Krištof, Bedřich Beneš, Jaroslav Křivánek, and Ondřej Šťava. Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum*, 28(2):219–228, 2009.
- [KH84] James Thomas Kajiya and Brian P Von Herzen. Ray tracing volume densities. volume 18, pages 165–174, 1984.
- [KK89] James Thomas Kajiya and Timothy Kay. Rendering fur with three dimensional textures. *Computer Graphics (Proceedings of ACM SIGGRAPH '89)*, 23(3):271–280, 1989.
- [KMN88] Alex D Kelley, Michael C Malin, and Gregory M Nielson. Terrain simulation using a model of stream erosion. In *SIGGRAPH '88: Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, pages 263–268, 1988.
- [Lew87] J. P. Lewis. Generalized stochastic subdivision. *ACM Transactions on Graphics*, 6(3):167–190, 1987.
- [LH04] Frank Losasso and Hugues Hoppe. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics*, 23(3):769–776, 2004.

- [LPFH01] Jerome Lengyel, Emil Praun, Adam Finkelstein, and Hugues Hoppe. Real-time fur over arbitrary surfaces. In *I3D '01: Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 227–232, 2001.
- [LWK03] Wei Li, Xiaoming Wei, and Arie Kaufman. Implementing lattice boltzmann computation on graphics hardware. *The Visual Computer*, 19(7-8):444–456, 2003.
- [MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159, 2003.
- [MDN02] Ryo Miyazaki, Yoshinori Dobashi, and Tomoyuki Nishita. Simulation of cumuliiform clouds based on computational fluid dynamics. In *Proceedings of Eurographics (short paper) 2002*, pages 405–410, 2002.
- [Mil86] Gavin S P Miller. The definition and rendering of terrain maps. In *SIGGRAPH '86: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 39–48, 1986.
- [MK02] Zeki Melek and John Keyser. Interactive simulation of fire. In *PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 431, 2002.
- [MKM89] Forest Kenton Musgrave, Craig E Kolb, and R S Mace. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the*

- 16th Annual Conference on Computer Graphics and Interactive Techniques*, pages 41–50, 1989.
- [MN98] Alexandre Meyer and Fabrice Neyret. Interactive volumetric textures. In *Rendering Techniques '98: Proceedings of Eurographics Symposium on Rendering*, pages 157–168, 1998.
- [MYND01] Ryo Miyazaki, Satoru Yoshida, Tomoyuki Nishita, and Yoshinori Dobashi. A method for modeling clouds based on atmospheric fluid dynamics. In *PG '01: Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, page 363, 2001.
- [Ney96] Fabrice Neyret. Synthesizing verdant landscapes using volumetric textures. *RAPPORT DE RECHERCHE-INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE*, 1996.
- [NFJ02] Duc Quang Nguyen, Ronald Fedkiw, and Henrik Wann Jensen. Physically based modeling and animation of fire. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 721–728, 2002.
- [OFT⁺03] Shin Ota, Tadahiro Fujimoto, Machiko Tamura, Kazunobu Muraoka, Kunihiro Fujita, and Norishige Chiba. $1/f$ β noise-based real-time animation of trees swaying in wind fields. In *CGI '01: Proceedings of Computer Graphics International, 2003.*, pages 52–59, 2003.

- [PC01] Frank Perbet and Marie-Paule Cani. Animating prairies in real-time. In *I3D '01: Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 103–110, 2001.
- [Pel04] Kurt Pelzer. Rendering countless blades of waving grass. *GPU Gems*, pages 107–121, Randima Femando, 2004.
- [PG07] Renato Pajarola and Enrico Gobbetti. Survey of semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, 23(8):583–605, 2007.
- [PPH⁺96] Przemyslaw Prusinkiewicz, Przemyslaw Prusinkiewicz, Jim Hanan, Jim Hananz, Mark Hammel, Mark Hammely, and Radomir Mech. L-systems: from the theory to visual models of plants. *Machine Graphics and Vision*, 5(1/2):365–392, 1996.
- [Ram05] Rishi Ramraj. Dynamic grass simulation and other natural effects. *Game Programming Gems 5*, pages 411–419, Kim Pallister, 2005.
- [RB85] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *SIGGRAPH '85: Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, pages 313–322, 1985.
- [SF92] Mikio Shinya and Alain Fournier. Stochastic motion-motion under the influence of wind. In *Computer Graphics Forum*, volume 11, pages 119–128, 1992.

- [SFK⁺08] Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2-3):350–371, 2008.
- [SO99] Tatsumi Sakaguchi and Jun Ohya. Modeling and animation of botanical trees for interactive virtual environments. In *Proceedings of the ACM symposium on Virtual Reality Software and Technology*, pages 139–146, 1999.
- [Sof09] JoWooD Productions Software. Arcania: A gothic tale, 2009.
- [Sou07] Tiago Sousa. Vegetation procedural animation and shading in Crysis. *GPU Gems 3*, pages 373–385, Hubert Nguyen, 2007.
- [SP] Barbara Solenthaler and Renato Pajarola. Predictive-corrective incompressible sph. In *SIGGRAPH '09: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*.
- [ST89] Richard Szeliski and Demetri Terzopoulos. From splines to fractals. *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, 23(3):51–60, 1989.
- [Sta97] Jos Stam. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. In *Computer Graphics Forum*, volume 16, pages 159–164, 1997.
- [Sta99] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 121–128, 1999.
- [Ubi09] Ubisoft. Call of juarez 2: Bound in blood, 2009.

- [Wan03] Niniane Wang. Realistic and fast cloud rendering in computer games. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Sketches*, pages 1–1, 2003.
- [Whi08] Matthew White. Real-time optimally adapting meshes: terrain visualization in games. *International Journal of Computer Games Technology*, 2008(2):1–7, 2008.
- [WWZ⁺05] Changbo Wang, Zhangye Wang, Qi Zhou, Chengfang Song, Yu Guan, and Qunsheng Peng. Dynamic modeling and rendering of grass wagging in wind: Natural phenomena and special effects. *Computer Animation and Virtual Worlds*, 16(3-4):377–389, 2005.
- [Zel04] Jeremy Zelnack. Vertex texture fetch water. *NVIDIA SDK White Paper*, 2004.
- [Zio07] Renaldas Zioma. GPU-generated procedural wind animation for trees. *GPU Gems 3*, pages 105–120, Hubert Nguyen, 2007.
- [ZST⁺06] Long Zhang, Chengfang Song, Qifeng Tan, Wei Chen, and Qunsheng Peng. Quasi-physical simulation of large-scale dynamic forest scenes. *Lecture Notes in Computer Science*, 4035:735–742, 2006.