Real-Time Continuum Grass

Kan Chen[*]
Nanyang Technological University, TQ Global Ltd

Henry Johan[†]
Nanyang Technological University

ABSTRACT

Simulating grass field in real-time has many applications, such as in virtual reality and games. Modeling accurate grass-grass, grass-object and grass-wind interactions requires a high computational cost. In this paper, we present a method to simulate grass field in real-time by considering grass field as a two dimensional grid-based continuum and shifting the complex interactions to the dynamics of continuum. We adopt the wave simulation as the numerical model for the dynamics of continuum which represents grass-grass interaction. We propose a procedural approach to handle grass-object and grass-wind interactions as external force that updates the wave simulation. The proposed method can be efficiently implemented on a GPU. As a result, massive amounts of grass can interact with moving objects and wind in real-time.

Keywords: Real-time animation, grass, continuum simulation.

Index Terms: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation, Virtual reality

1 INTRODUCTION

In recent years, there has been an increasing demand in interactive virtual environments. Real-time simulation of a large scale dynamic and believable virtual environment provides users with an immersive feeling. Grass field is an important component of outdoor environment and it is required in many applications, such as virtual reality, games and simulators. Using texture mapping and modeling techniques, we can realize the visual complexity of a grass field. However, in many real-time applications, grass field is still lack of interactivity. For example, in many of the recent games, grass still cannot interact with objects and penetrate through the objects.

A grass field will look more believable 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. 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, however 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 real-time solution for such application.

In this paper, we propose a method to simulate the interactions and natural effects of a large scale grass filed in real-time for virtual reality and games. Based on the grass in real-life, we focus on simulating the essential effects and behaviors of grass, such as bending and recovery. Grass should bend and recover in response to passing objects and wind, and the bending effect propagates to the neighboring grass. This propagation is mainly caused by grass-grass collision. Moreover, grass is hard to break and stretch but free to move in all directions. Other than the physical properties of grass, the behaviors of grass also depend on how objects and wind interact with them. For example, 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 behaviors are considered in the design of our model.

In many real-time applications, the realization of the effects of grass is more important than the physical accuracy. Thus, we propose a novel real-time method that combines a simple dynamic simulation and a procedural method instead of performing an explicit physics simulation. Our method to simulate grass integrates four types of interaction: grass-grass, grass-terrain, grass-object and grass-wind. The proposed method has the following features:

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

(2) We propose a procedural method to handle the grass 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) Animators can control the simulation effects by setting the parameters of the wave propagation and the behaviors of the objects and wind.

All computations 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 10.

2 RELATED WORK

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

2.1 Modeling of grass

Modeling of grass has been widely investigated. The major techniques to model a grass scene are as follows.

Particle system Reeves and Blau [19] used particles that move along parabolic trajectories to draw grass. The rendering cost is too high to be suitable for real-time applications.

Volumetric representation Kajiya and Kay introduced 3D texture mapping called ”texel” to model furry surface [11]. Meyer and Neyret [12, 13] 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. [10], Bakay and Heidrich [1], Banisch and Wuthric [2]. The shell-based method is based on geometry layers mapped with semi-transparent textures and sliced parallel to the terrain. Pelzer [16] 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 [5] proposed to use volumetric textures and aperiodic tiling. The forest is represented using a set of edge-compatible prisms containing forest samples which are aperiodically mapped

[*]e-mail: chenkan@pmail.ntu.edu.sg
[†]e-mail: henryjohan@ntu.edu.sg

Table 1: Comparison with other grass animation methods.

Method	Wind	Object	Grass	Terrain	Summary
Perbet and Cani [17]	Yes	No	No	No	Control using wind primitives.
Bakay and Heidrich [1]	Yes	No	No	No	Control using wind vectors.
Guerraz et al. [7]	Yes	Yes	No	No	Animate blade by blade independently for nearby grass using control primitives and store the information of all bended grass.
Ramraj [18]	Yes	No	No	No	Animate using a simple wave simulation.
Wang et al. [20]	Yes	No	Yes	No	Perform free-form deformation and explicit collision detection.
Banisch and Wuthric [2]	Yes	No	Yes	No	Simulate using a mass-spring model.
Habel et al. [8]	Yes	No	No	No	Distort grass texture lookup based on a procedural or hand-crafted texture.
Orthmann et al. [15]	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.

onto the ground. This method can also be used to model grass. In general, the volumetric representation is efficient in rendering but not easy to perform animation.

3D geometric representation Wang et al. [20] 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 work using the level of detail scheme to combine 3D geometry representation, volumetric representation and 2D texture of grass. Perbet and Cani [17] 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. [7] used many billboards for 3D representation of grass and semi-transparent vertical textures of different orientations for volumetric representation of grass. Boulanger et al. [4] 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. The 3D geometric representation of grass can model grass in details, however the computational cost for rendering and animation is high.

Other methods Habel et al. [8] and J. Orthmann et al. [15] used only the conventional billboards to model grass. Deussen et al.[14] used lines and points to represent grass.

2.2 Animation of grass

There are several work for modeling grass-wind interaction. Perbet and Cani [17] proposed a procedural animation controlled by wind primitives. Bakay and Heidrich [1] created an animation by translating the vertices of the shell grass according to a wind vector. Wang et al. [20] combined free form deformation and explicit grass-grass collision detections. Endo et al. [6] proposed a precomputed physical model using dynamic constraints. Banisch and Wuthric [2] used a mass-spring system where the masses and springs are attached to the shell layers. Habel et al. [8] achieved a texture based animation by distorting the texture lookups. The conventional way to generate wind effects is by using the combination of sin and cos wave plus noise as in Pelzer [16] and Zioma [22]. Ramraj [18] simulated the wind effects using a simple water wave.

Most grass animation techniques can achieve visually pleasing results, but they only addressed grass-wind interaction. For grass-object interaction, Guerraz et al. [7] presented a method to simulate the interaction between a vegetation and a moving object. Most recently Orthmann et al. [15] proposed a GPU-based approach to model responsive grass. They handle grass-object collision by passing the vertices of grass to the geometry shader and performing the simulation using a spring model.

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 and we address all the grass-grass, grass-object, grass-terrain and grass-wind interactions. Table 1 shows the comparison of our method with other methods.

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. [3] gave an overview of hair simulation. Hadap and Thalmann [9] 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 fluid dynamics.

3 Modeling a grass field

To model a grass field, we have to model a grass and determine how to distribute grass in the terrain.

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 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 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 1(b)). This representation can model more complex deformation of grass. To make our method more efficient, we also consider the level-of-details (LOD) for grass. That is, for grass far away from objects and camera, we use the one-segment billboard. Based on the simulation results, we bend the segments of the billboard (Figure 1). The simulation process will be discussed in details in Section 4. The height of 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.

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. 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.

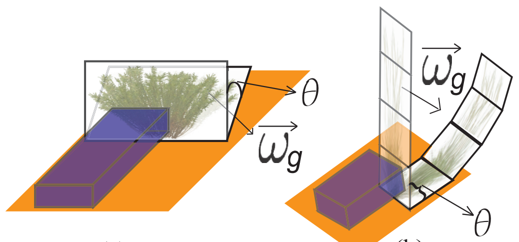

Figure 1: Object interacts with (a) a one-segment billboard grass (stiff grass and short grass) and (b) a multiple-segment billboard grass (soft grass and tall grass).

4 Real-time grass simulation

In this section, 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 is explained in Section 5.

4.1 Basic idea and simulation steps

We want to achieve the following effects which are essential to a grass field: When moving objects and wind enter a grass field, they collide and bend the grass. The grass that collides with the objects and wind will bend first. The grass-grass interaction occurs. The bending is then propagated to other grass that is away from the objects and wind. After the objects and wind passed, a track will be left on the field. The bended grass will eventually stop bending and start to recover (it may sway) and finally recover to a rest pose.

To achieve the above effects, we propose to model a grass field as a continuum. The interaction between grass is actually the transfer of energy, hence the grass-grass interactions are modeled as the energy flow in a continuum and the grass-object and grass-wind interactions are modeled as the external force to update the continuum simulation (details in Section 5). We adopt the wave simulation as the numerical model for the dynamics of continuum. Furthermore, those effects of grass field (refer to the above effects of grass) have similarities with the behaviors of wave: The initial amplitudes of wave are set at some locations. The amplitudes are then propagated to nearby locations. During the propagation, the amplitudes will eventually stop increasing and start to decrease (the wave may oscillate), and the wave will finally recover to a rest state. Thus, we adopt the wave model and mimic those essential effects of grass field using the wave simulation.

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 at the bottom segment) and *bending velocity* (the angular velocity $\vec{\omega}_g$) (Figure 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).

The simulation is performed as follows:

(1) Simulation region management: Manage the patches of grass where the continuum simulation is performed (Section 4.2).

(2) Interaction with objects and wind: This process includes collision detection and response. We detect the grass that collides with objects and wind. Then, we update the bending velocity $\vec{\omega}_g$ of the grass. This step will be discussed in details in Section 5.

(3) Direct bending process: This step processes the bending effect caused by grass-object and grass-wind interaction. We bend the clusters of grass colliding with objects and wind by updating the bending angle θ (Section 4.3).

(4) Propagation process: This step simulates the bending effect caused by 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 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 rendering (Section 4.5).

4.2 Simulation region management

Simulation regions are patches of grass in which 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.

(1) 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.

(2) 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.

(3) If the memory consumption reaches the specified upper bound (in our experiment: 225 patches, 180 feet by 180 feet per patch), 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.

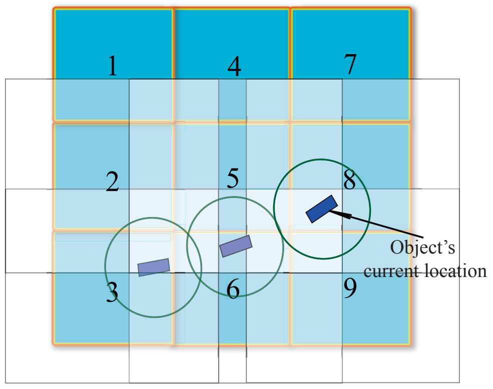

Figure 2: The dark small rectangles represent the moving object at some instances of time. We allocate the memory for the patches of grass no. 2, 3, 5, 6, 8 and 9. The circles represent the affected area caused by the object at current location.

We increase the size of the 2D cells of 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 2). This is to ensure that when an object is crossing to the neighboring patches of grass, the simulation is continuous at the boundary. The size of the extended 2D cells of one patch $(m_a \times n_a)$ should be sufficient enough to record all potential interactions on the patch including the interactions which are caused by objects outside the original patch $(m_g \times n_g)$.

In Figure 2, we can easily see that the object is currently located inside the extended 2D cells of patches no. 5, 6, 8 and 9. Thus, we allocate memory and perform simulation for patches no. 5, 6, 8 and 9. The patch-by-patch simulation can be done in any order. Note that the object is not in the original patches no. 5, 6 and 9, however 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 patches no. 5, 6 and 9. The patches no. 2 and 3 are the patches of grass where the object has passed. We continue to perform the simulation on these patches since the effects of grass-object interaction might still be noticeable in these patches.

Inside the simulation regions, besides grass-object interactions, we also perform grass-wind simulation for all the patches that are near to the camera (based on the same distance checking using d mentioned previously). For other patches that are far away from the camera (maybe inside or outside the simulation regions) we use only one patch of grass-wind simulation result to tile these patches. The grass-wind simulation will be discussed in details in Section 5.

4.3 Direct bending process

After the collisions are handled (details in Section 5), we have a new bending velocity $\vec{\omega}_g$ for the clusters of grass colliding with objects 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 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 objects and wind is

$$\theta(t_s+\Delta t_s) = \theta(t_s) + \|\vec{\omega}_g\|\Delta t_s. \tag{1}$$

To avoid the intersection between the bottom segment of 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. In other words, we propagate the energy in continuum.

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 each cell, the bending angle and the bending velocity, as the amplitude of 2D waves. The wave functions we used are the same as in [21]. For each simulation time t_s, $a_\theta(t_s)$ and $a_{\vec{\omega}_g}(t_s)$ are the accelerations of the wave propagation:

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

$$a_{\vec{\omega}_g}(t_s) = \frac{\partial^2\vec{\omega}_g}{\partial t_s^2} = c_{\vec{\omega}_g}(t_s)^2\left(\frac{\partial^2\vec{\omega}_g}{\partial x^2} + \frac{\partial^2\vec{\omega}_g}{\partial y^2}\right). \tag{3}$$

c_θ and $c_{\vec{\omega}_g}$ are the wave's propagation speed. 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 $\|\vec{\omega}_g(t_s)\|$ at t_s:

$$c_\theta(t_s) = c_0 + \mu\,\|\vec{\omega}_g(t_s)\|. \tag{4}$$

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, 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 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:

$$\begin{aligned}\theta(t_s+\Delta t_s) &= (2-\lambda_\theta)\theta(t_s) - (1-\lambda_\theta)\theta(t_s-\Delta t_s)\\ &+ a_\theta(t_s)(\Delta t_s^2),\end{aligned} \tag{5}$$

$$\begin{aligned}\vec{\omega}_g(t_s+\Delta t_s) &= (2-\lambda_{\vec{\omega}_g})\vec{\omega}_g(t_s) - (1-\lambda_{\vec{\omega}_g})\vec{\omega}_g(t_s-\Delta t_s)\\ &+ a_{\vec{\omega}_g}(t_s)(\Delta t_s^2),\end{aligned} \tag{6}$$

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 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 cells, we bend or recover the billboards 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 representation of the grass and we assume that the object hits the bottom segment of the billboard.

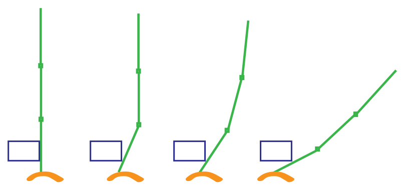
Figure 3: The bending process of one cluster of grass. Grass bends from the bottom to top.

Based on the observation that when the collision happens at the bottom of grass, the bottom bends while dragging the top to bend (Figure 3). 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 a cluster of grass with n segments $\{L_0 \ldots L_{n-1}\}$ at cell P, the bending angles of the segments at rendering time t_r are $\{\theta_0(t_r)_P \ldots \theta_{n-1}(t_r)_P\}$:

$$\theta_i(t_r)_P = \theta_0(t_r - \gamma_i)_P, \quad (0 \le i \le n-1) \tag{7}$$

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.

If memory is not an issue, we store the bending angles θ_0 at several time steps and use them to evaluate Equation 7. 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 grass and only pass the latest result of the continuum simulation to the rendering process. We then perform an approximated computation as follows.

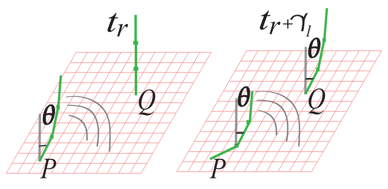

Figure 4: During time γ_i, the θ value propagates from P to Q. The θ value also propagates from the bottom to the upper segment at P.

Consider the wave propagation from cell P to cell Q during time γ_i. The θ value stored in the cell Q is actually the θ value of the grass at P at γ_i time ago (Figure 4). The direction of the 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

$$\theta_i(t_r)_P = \theta_0(t_r - \gamma_i)_P \approx \theta_0(t_r)_Q, \tag{8}$$

$$Q = P + \gamma_i c_\theta(t_r)_P \frac{\vec{\omega}_o}{\|\vec{\omega}_o\|} \tag{9}$$

where $c_\theta(t_r)_P$ is the wave propagation speed at cell P and is computed using Equation 4. $\gamma_i c_\theta(t_r)_P$ 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 4.4), 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 objects or wind interacts with the grass at the cell Q, then we may not get the correct previous θ value. However, the experimental results show that no visual defects are encountered.

The bending of billboard is in 3D. Using the forward kinematics (Figure 5), we move the 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.

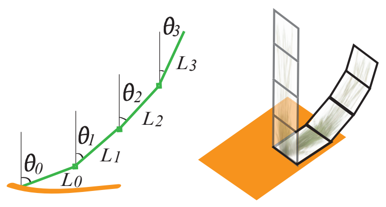

Figure 5: Based on θ and L (the height of the segment), apply the forward kinematics to bend a cluster of grass.

For grass' recovery, we perform the similar computation as grass' bending. The lower parts recover first while dragging the upper parts to recover. Generally, the recovery of the upper parts may not be delayed much, therefore in the recovery process, we use smaller values for γ_i in Equation 7.

5 Interaction with Objects and Wind

The grass-object and grass-wind collision directly update the bending velocity of the grass that collides with objects and wind. First, we describe grass interacting with moving objects, such as moving cars (Sections 5.1, 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 6). The extrusion represents the area of grass that directly collides with the object during the simulation time step. Thus, 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.

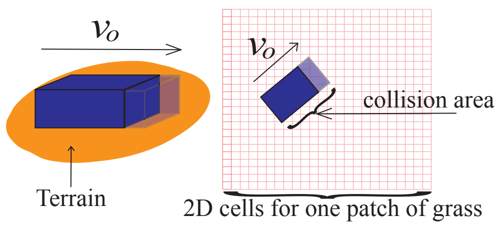

Figure 6: Determine the collision area.

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 Section 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 7). We add them to determine the bending direction.

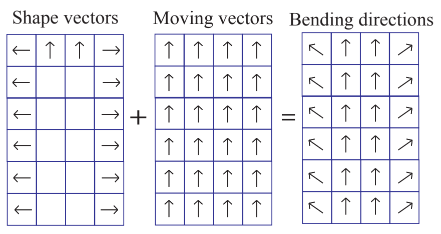

Figure 7: Assume that the object has a rectangular shape. The calculation of the bending directions for the grass in the collision area.

The shape vectors are created based on the shape of collision area (Figure 8). Suppose the position of the cell in the collision area is P. The shape vector $\vec{M}_P$ is defined as follows.

$$\vec{M}_P = \begin{cases} \vec{N}_P & \text{if } \vec{N}_P \cdot \vec{v}_o \geq 0 \text{ and P is at boundary,} \\ \vec{0} & \text{if } \vec{N}_P \cdot \vec{v}_o < 0 \text{ or P is not at boundary,} \end{cases} \tag{10}$$

where $\vec{v_o}$ is the object's velocity and $\vec{N_P}$ is the normal vector at cell P.

The moving vectors are created based on the object's moving behavior. For example, the object bends the grass in its moving direction, therefore the grass in the collision area has moving vector of $\frac{\vec{v_o}}{\|\vec{v_o}\|}$ (Figure 7). For the cell at P, the moving vector is $\vec{T_P}$, which is normalized, thus the bending direction $\vec{d_{gP}}$ is:

$$\vec{d_{gP}} = \frac{\vec{M_P} + \vec{T_P}}{\left\|\vec{M_P} + \vec{T_P}\right\|}. \tag{11}$$

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 8. 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 P, we can compute the grass's tangential speed s_g and the bending velocity $\vec{\omega_{gP}}$ from the object's horizontal speed s_o by

$$s_g = s_o cos\theta, \quad \vec{\omega_{gP}} = \vec{d_{gP}}\left(s_g/\left(h/cos\theta\right)\right), \tag{12}$$

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.

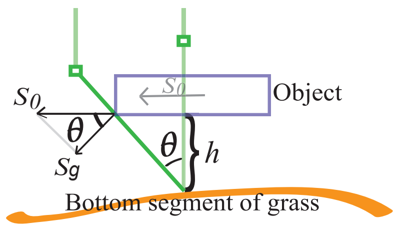

Figure 8: The calculation of the bending velocity of the grass in the collision area.

5.3 Interaction with wind

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 11(a) and 11(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 wind and how the wind bends the grass (bending directions). 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 directions. Figures 9(a) and 9(b) show two examples of the shape of the collision area and the bending directions designed by an animator.

The second approach simulates the effect caused by natural wind that is similar to the method proposed by Ramraj [18]. We perturb the values $(\theta, \vec{\omega_g})$ 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.

6 GPU IMPLEMENTATION

All the simulation steps of our method are performed in GPU. The states of the grass are stored as textures in GPU memory. The simulation is implemented using standard texture manipulation techniques, such as rendering to texture and texture fetching. If the shape of the object is concave, we divide it into several convex

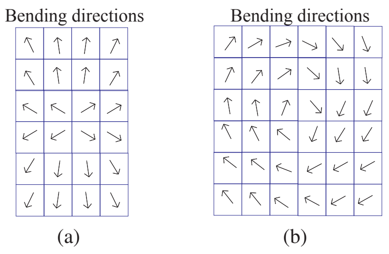

Figure 9: Two examples of the shape of the collision area and the bending directions designed by an animator.

shapes. We pass the object's position, it's convex shapes 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 a simulation texture in pixels is the same as the resolution of the extended 2D cells in a patch. Each pixel of the 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 determine the collision area based on the object's velocity and it's convex shapes. This calculation is done in vertex shader when 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. Computing each pixel's $\vec{\omega_g}$ in pixel shader for the whole collision area is expensive. Therefore, the $\vec{\omega_g}$ is computed at the vertices of the convex 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 [21]. 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.

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 grass-terrain interaction, we compare the height of the joints of grass billboards with the height of terrain, which is also stored as a texture. If the height of a joint is less that the height of terrain (intersection happens), the joint of the grass segment will use the height of the terrain as its height.

7 RESULTS AND DISCUSSIONS

Figures 10, 11, 12 and 13 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.

Comparisons We compare our method with Guerraz et al. [7] and Orthmann et al. [15] because their methods simulate grass-object interaction. Compared with [7] and [15], our method can simulate larger area of grass in real-time. Guerraz et al. [7] performed grass-object interaction simulation blade by blade and stored the information of all the affected grass. Thus, the computational cost is high, especially when multiple objects are moving in a dense grass field leaving a lot of tracks. We can simulate up to millions of clusters of grass in real-time (Note that not all of the simulated clusters of grass are visible at one point of time), while Orthmann et al. [15] simulates in the order of 60000 clusters of grass.

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 Conclusions and future work

In this paper, we have presented a method for real-time simulation of grass for games and virtual reality applications such as outdoor training simulations, driving simulators and flight simulators. We proposed to model grass field as a continuum. To efficiently model grass-object, grass-wind interactions and to propagate the effects (grass-grass interaction) in a grass field, we proposed a procedural collision detection, response and wave simulation, respectively. Our method is suitable for GPU implementation. In addition, our method allows the user to control the simulation effects by setting several intuitive parameters. 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.

Acknowledgments

We gratefully thank the reviewers for their constructive comments. We also thank the coordinator of our submission for the help during the preparation of the camera-ready version.

References

[1] B. Bakay and W. Heidrich. Real-time animated grass. In *Proceedings of Eurographics (short paper) 2002*, 2002.

[2] S. Banisch and C. A. Wuthric. Making grass and fur move. *Journal of WSCG 2006*, 14:25–32, 2006.

[3] F. Bertails, S. Hadap, M.-P. Cani, M. Lin, T.-Y. Kim, S. Marschner, K. Ward, and Z. Kačić-Alesić. Realistic hair simulation: animation and rendering. In *SIGGRAPH 2008: ACM SIGGRAPH 2008 Classes*, pages 1–154, New York, NY, USA, 2008.

[4] K. Boulanger, S. N. Pattanaik, and K. Bouatouch. Rendering grass in real time with dynamic lighting. *IEEE Computer Graphics and Applications*, 29(1):32–41, 2009.

[5] P. Decaudin and F. Neyret. Rendering forest scenes in real-time. In *Rendering Techniques 2004: Proceedings of Eurographics Symposium on Rendering*, pages 93–102, Norrkoping, Sweden, 2004.

[6] L. Endo, C. Morimoto, and A. Fabris. Real-time animation of underbrush. *Journal of WSCG 2003 (short paper)*, 11, 2003.

[7] S. Guerraz, F. Perbet, D. Raulo, F. Faure, and M.-P. Cani. A procedural approach to animate interactive natural sceneries. In *CASA 2003: Proceedings of the 16th International Conference on Computer Animation and Social Agents*, pages 73–78, Washington, DC, USA, 2003.

[8] R. Habel, M. Wimmer, and S. Jeschke. Instant animated grass. *Journal of WSCG 2007*, 15:123–128, 2007.

[9] S. Hadap and N. Magnenat-Thalmann. Modeling dynamic hair as a continuum. *Computer Graphics Forum*, 20(3):329–338, 2001.

[10] A. F. Jerome Lengyel, Emil Praun and H. Hoppe. Real-time fur over arbitrary surfaces. In *I3D 2001: Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 227–232, New York, NY, USA, 2001.

[11] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, 23(3):271–280, 1989.

[12] A. Meyer and F. Neyret. Interactive volumetric textures. In *Rendering Techniques 98: Proceedings of Eurographics Symposium on Rendering*, pages 157–168, New York, NY, USA, 1998. Springer.

[13] F. Neyret. Synthesizing verdant landscapes using volumetric textures. *RAPPORT DE RECHERCHE-INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE*, 1996.

[14] P. H. Oliver Deussen, R. M. Bernd Lintermann, M. Pharr, and P. 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, New York, NY, USA, 1998.

[15] J. Orthmann, C. Rezk-Salama, and A. Kolb. GPU-based Responsive Grass. In *Journal of WSCG 2009*, 2009.

[16] K. Pelzer. Rendering countless blades of waving grass. *GPU Gems*, pages 107–121, 2004.

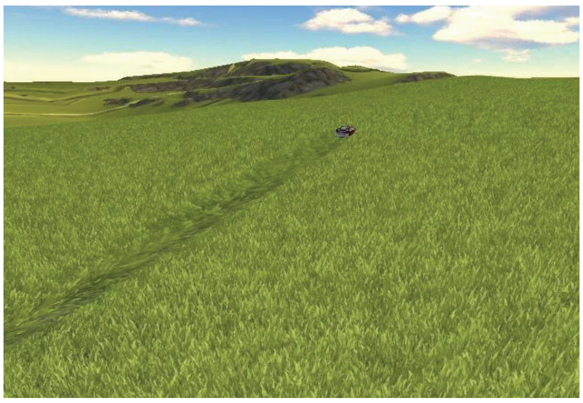

Figure 10: Example of real-time grass. The track appeared as the result of the interaction between car and grass.

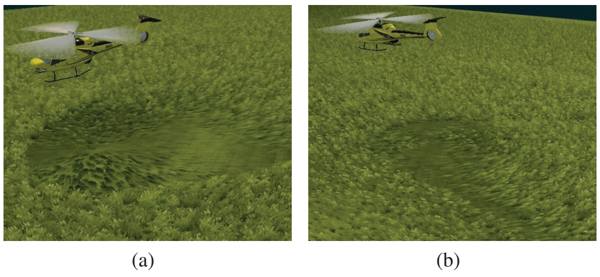

Figure 11: A helicopter scene using the bending directions in (a) Figure 9(a) and (b) Figure 9(b).

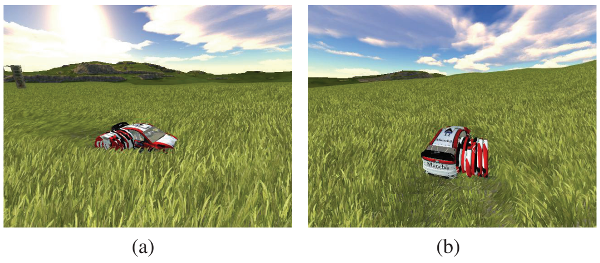

Figure 12: Interaction with a concave object.

[17] F. Perbet and M.-P. Cani. Animating prairies in real-time. In *I3D 2001: Proceedings of the 2001 Symposium on Interactive 3D graphics*, pages 103–110, New York, NY, USA, 2001.

[18] R. Ramraj. Dynamic grass simulation and other natural effects. *Game Programming Gems 5*, pages 411–419, 2005.

[19] W. T. Reeves and R. 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, New York, NY, USA, 1985.

[20] C. Wang, Z. Wang, Q. Zhou, C. Song, Y. Guan, and Q. 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.

[21] J. Zelsnack. Vertex texture fetch water. *NVIDIA SDK White Paper*, 2004.

[22] R. Zioma. GPU-generated procedural wind animation for trees. *GPU Gems 3*, pages 105–120, 2007.

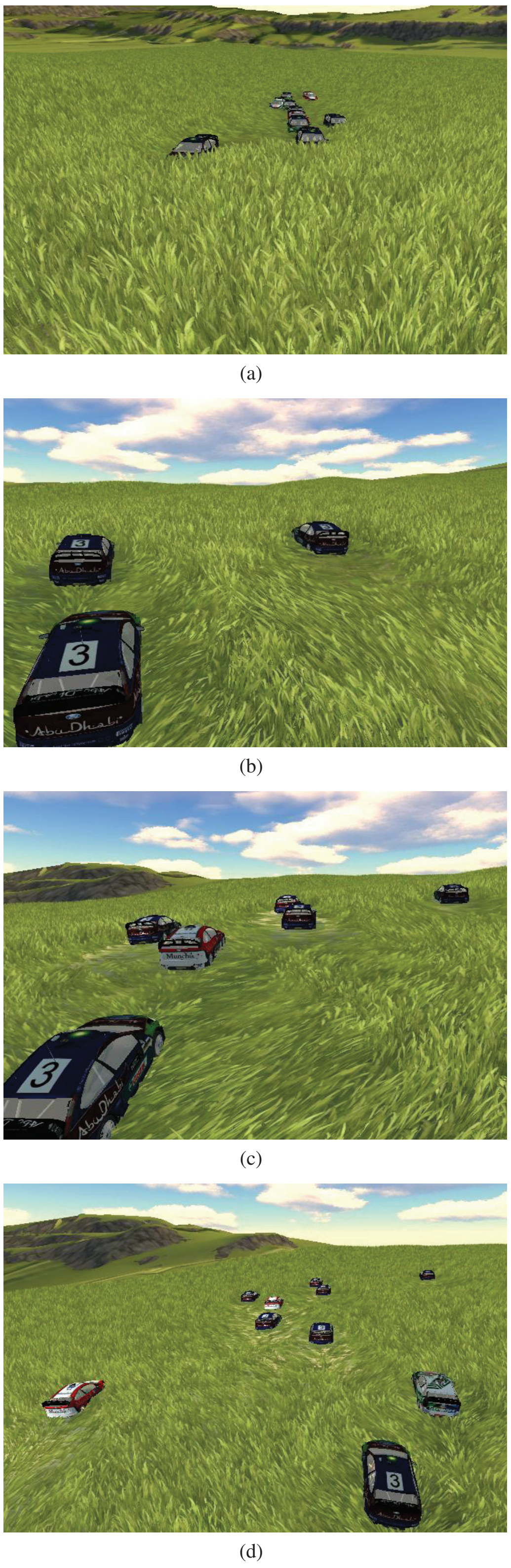

Figure 13: Animation sequence with multiple cars.