Animating Prairies in Real-Time

Frank Perbet and Marie-Paule Cani

iMAGIS†-GRAVIR, INRIA Rhône-Alpes
655 avenue de l'Europe, 38330 Montbonnot, France
Frank.Perbet/Marie-Paule.Cani@imag.fr http://www-imagis.imag.fr/Membres/Frank.Perbet/

Abstract

Generation of dynamic natural scenes is essential for real-time applications, such as simulators or video-games. This paper presents a method for animating and rendering a prairie in real time. The geometric model for the grass relies on three different levels of detail: 3D geometry, volumetric textures (called here 2.5D representation), and 2D textures. The animation of these LODs is controlled through procedural animation primitives that implement wind effects such as slight breeze, gust of wind, whirlwind, or blast of air due to a flying object. Smooth transitions between levels of detail are computed "on the fly" according to camera motion, without stopping the animation. We discuss real-time performance on two platforms: an SGI O2, and an ONYX 2 with an Infinite Reality board.

Categories and subject descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling–Curve, surface, solid and object representations, Geometric transformations; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism–Animation, Texture, Virtual reality.

Additional Key Words and Phrases: Real-time 3D graphics, Levels of detail, Natural phenomena.

1 Introduction

Human brain seems to have an insatiable need for complexity. This is particularly true when one looks at computer generated images: details, and if possible dynamic details, are essential to immerse oneself in a virtual world.

But this visual complexity is difficult to achieve for interactive applications such as video games. Owing to the the power of 3D boards and to the basic geometry of artificial (i.e. purely human made) environments, indoor scenes are easy to render and animate in real-time. Outdoor, the present models are not very attractive, especially for natural scenes. Indeed, because of their richness in animated details, they are very difficult to render in real-time. More-

---

over, little work has been carried out in this direction. This paper focuses on this new and exciting challenge.

We present a set of methods for the real-time display, animation and control of a very specific kind of natural scene: a prairie. Nevertheless, the methodology we develop can serve as a starting point for more general applications. As our main target applications are video games, the images need to be generated from the point of view of a walker (i.e. with a grazing angle). The prairie example is challenging because great portions of blades of grass continuously range from very close to quite far regions from the viewer (see figure 1). Our solution relies on the simultaneous use of three levels of detail (3D geometry, volumetric textures and 2D textures) to represent the grass. In addition to this representation, we present a method for animating the prairie under the influence of a variety of wind-effects, which can be interactively controlled by the user. Lastly, we describe transitions between animated LODs when the camera moves, without altering the perception of grass blown by the wind.

Figure 1: A real image of the kind of natural scene we would like to model and animate in real-time. . .

1.1 Related work

To the authors knowledge, no previous work has been done on the specific case of real-time animation of prairies. However, a number of previous works, each related to a specific aspect of our problem, were an inspiration to us. They are described next.

Object representations: The first obstacle for generating a virtual prairie is the number of geometric primitives that are needed. Rendering a geometric model for each blade of grass has been done using particle systems [10, 11]. But doing so with a sufficient frame rates and at the scale of a whole prairie is clearly impossible. Thus, coarser representations have to be generated. However, the prairie being a collection of many small objects rather than a single complex one, geometric methods such as polygons decimation cannot be used.

Volumetric textures, or "texels" [5, 9], is one of the best representations for capturing the complexity of fuzzy objects such as vegetation, fur, or short hair. Real-time implementations of texels, based on layers of polygons mapped with semi-transparent textures, have been recently developed [8, 6]. In particular, Lengyel [6] uses real-time texels for representing fur, which is quite similar to grass. His work allows one to span the viewing distance from close-ups to far views, thanks to the use of several LODs. The latter include fine to coarse versions of the texel representation, textures being more or less filtered. In addition, a method for generating real-time soft shadows is described.

This work is not directly applicable in our case: first, Lengyel maps the semi-transparent textures modeling fur onto concentric shells over the support surface. Although convenient when the predominant viewing direction is perpendicular to this surface, this method would not give good results for computing a walker's view in a prairie, shell polygons being viewed edge-on. Secondly, Lengyel's method is not designed for allowing animation: although animation methods had been developed for previous texel representations [9], the way Lengyel pre-filters the semi-transparent textures forbids any animation of hair strands. Only camera motions can be generated, and the motionlessness of the scene considerably simplifies the generation of smooth transitions between LODs.

Our work also relies on real-time texels for one of the LODs, but does it in a slightly different way, more convenient for generating animations. Although we did not implement it, Lengyel's solution for soft shadowing would certainly be a good way of enhancing the visual realism of our prairies.

Animation methods: A first method for generating a stochastic animation of fields of grass blowing in the wind is proposed by Reeves in the mid-eighties [11], as an application of the particle-based modeling approach. Particles evolving in a 2D space are first used for modeling complex gusts of wind subject to random local variations of intensity. The resulting 2D wind maps are stored, for each animation step. Then, procedural trajectories are generated for particles modeling blades of grass by letting them bend proportionally to the intensity stored in the map, according to a given wind direction.

Still using a procedural approach, Neyret [9] animates breeze in a meadow by applying a sine wave motion to the control points of ray-traced texels modeling grass. This method generates the same deformations as if blades were embedded into an animated FFD volume [12]. Therefore, it results into a very homogeneous animation.

Generating grass response to various winds can be done using physically-based simulation. Wejchert [15] simulates the motion of falling leaves in velocity fields created by combining a few wind primitives. Shinya [13] simulates vegetation such as trees and grass blown by complex wind fields. The main contributions are an accurate modeling of the stochastic properties of winds, which are modeled in Fourier space and then converted into time varying-force fields, and the idea of using physically-based simulation of the vegetation. However, although this idea is applied to tree branches (using the modal analysis of uniform beams), the physical approach is considered too time consuming for being applied to grass: procedural particle animation is used as in [11], but with initial conditions that vary with the local value of the force field.

Still close to grass models, physically-based modeling has been applied to hair. Anjyo [1] relies on chains of rigid sticks, with 1D oscillators at hinges, for simulating hair motion under the influence of wind. To increase efficiency, Dalegan [3] only applies physically-based modeling to few "strands guides", interpolated motion being used for others strands. However, even this simplified method would hardly reach real-time performances if applied

to a whole prairie, where the view goes from close blades of grass to hills in the distance.

Closer to our concern for efficiency, Stam [14] generates the animation of trees under turbulent winds in real-time. Similarly to Shinya [13], he relies on modal analysis of tree dynamics and on the stochastic modeling of winds in Fourier space. However, the key idea here is to combine pre-computed vibration modes for animating the tree branches rather than integrating dynamical equations over time. Our solution to prairie animation inspires from Stam's ideas: we rely on physically-based precomputations during a real-time animation session, where the effects, rather than the causes, are animated. However, we are not meaning to perform a proper simulation, so we use much simpler models: we animate the action of procedural wind primitives — similar to those in [15], but with an additional time-varying stochastic component — rather than accurately modeling turbulent winds. Moreover, our pre-computations only simulate simplified physically-based models (as those used in hair modeling techniques), which are sufficient to easily obtain an acceptable visual realism.

Concerning animation techniques in a LOD framework, the simultaneous use of several LODs has already been applied in both procedural and physically-based animation [2, 4, 7]. The basic principle is to generate different approximations of the desired motion, according to the number of degrees of freedom that are currently active. Our animation method can also be interpreted this way. However, the prairie case exacerbates a difficulty that has not received much attention in the past: the problem of generating seamless transitions between LODs during animation. Indeed, there is no skin (contrary to [4]) for hiding the switches between LODs, and classical cross-dissolve techniques would result in vanishing blades of grass reappearing at different locations.

1.2 Overview

This paper presents an integrated set of methods for animating prairies in real-time. The first contribution, described in Section 2, is a representation of grass that includes three different LODs and yields real-time visualization. The animation algorithm, introduced in Section 3, relies on precomputations for enabling the animation of wind primitives over the prairie. Interactive control of wind effects is illustrated through the animation of a flying object that produces a blast of air. Section 4 introduces an original technique to achieve seamless transitions between LODs These transitions depends on the camera motions. They allow the different LODs to adequately coexist during the animation of the prairie and the movement of the walker. Performances are discussed in Section 5. Section 6 concludes and presents future work.

2 Real-time Display of a Prairie

Let us consider the case of a standard walker's view of a meadow, with a viewing direction almost parallel to the terrain. Then, both close and distant parts of the field of grass will appear in the same image. To obtain a good visual quality with a high frame rate, we combine different representations of grass, each of them being used at a given distance range. In addition, to allow camera motion, the prairie model must support an adaptive tiling of the terrain into regions specific to a given LOD. The next sections discuss these two points.

2.1 LOD representation of grass

While 3D blades of grass are the only acceptable solution to generate convincing rendering and animation close to the viewer, tex-

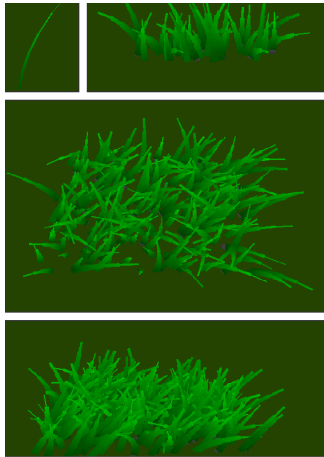tured polygons are used to solve both aliasing and efficiency problems in distant regions.

Figure 2: First level of detail: the 3D representation. Top left: a blade of grass. Top right: 3D blades located in a slice of a patch. Middle: a patch of grass viewed from above. Bottom: the same patch from the walker's point of view.

We rely on three levels of detail to represent grass.

- a 3D representation, depicted in Figure 2, is used for blades of grass near the viewer. The blade is represented as a chain of line-segment primitives.

- an interactive representation of a volumetric texture, inspired from Meyer's work [8], is used at mid-distance. Rather than using semi-transparent layers made of one polygon, we use layers of vertical polygon strips, like the one depicted in Figure 3, in order to allow animation. Each polygon strip is covered by a semi-transparent texture representing blades of grass (see Figure 4). There are two possible and perpendicular orientations for each polygon strip. We choose the one which prevent the viewing direction from being parallel to the polygon strip. In the remainder of this paper, we call this model the "2.5D" representation.

- a 2D texture is mapped on the terrain for representing grass in distant regions.

These LODs can easily be tuned from fine to coarse versions by changing the number of geometric primitives in the 3D grass model, the number of polygons and the resolution of textures in the 2.5D representation, and the resolution of the texture in the 2D model. We will develop this point in Section 5.
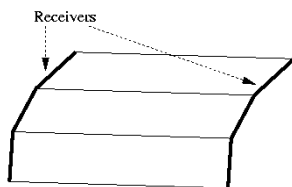

Figure 3: In the 2.5D representation, the two vertical edges of a polygon strip can be animated.

2.2 Coating the terrain with patches of grass

The terrain model is an elevation map. To allow the joint use of different LODs for the grass, we tile the map into a number of square
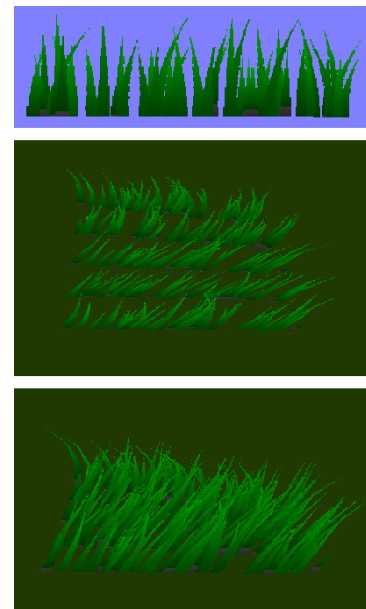

Figure 4: Second LOD: 2.5D representation. Top: a polygon strip mapped with a semi-transparent texture. Middle: a view from above showing the layers of textured polygons. Bottom: the walker's view of the grass-patch.

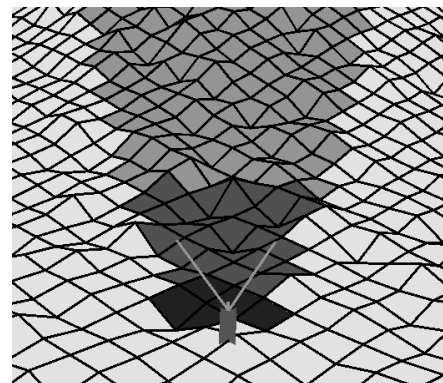elements, called "patches of grass", as shown in Figure 5.

Figure 5: The terrain, tiled into patches of grass, is viewed from above. The walker's position and his view cone are represented. No grass representation is associated with the lightest patches, which are culled. The remaining patches are displayed in different colors according to the grass LOD that is applied to them.

Since the walker's view is supposed to be parallel to the ground, we rely on a basic 2D culling to quickly get rid of most patches of grass that will not appear in the image (see Figure 5). Remaining patches of grass are provided with a given grass representation according to their distance to the viewer. We use a 2D discrete distance, where patches sharing an edge or a vertex with the viewer patch are set to be at distance one. Regions associated with a specific LOD are defined by a given range of distance (for instance, the 2.5D representation is used from distance 3 to 6 in Figure 5). A discussion on the tuning of distance thresholds will be held in Section 5.

Generating a transition between LODs when the camera moves requires the ability to represent consistently *the same patch of grass* using any of the three representations. To do so, we use a procedural method to build stochastic patches of grass. This is done from the finest to the coarsest representation: we generate the 3D

blades of grass first, using random numbers to set the orientation of each blade segment. Then, the semi-transparent textures used in the 2.5D representations are generated by computing 2D images of the 3D blades included in a slice of the 3D patch of grass. Finally, the 2D textures are computed as coarse images of the 3D patch of grass taken from different viewing angles. Mip-mapping is used to filter these textures according to the current viewing distance. Since we use OpenGL alpha-channel for transparencies, the scene is rendered from back to front. To achieve that, we just need to process the prairie in the right order, without having to sort any polygons.

In practice, transitions will take place during a real-time animation session, where grass undulates under the influence of wind. After a description of our animation algorithm in Section 3, Section 4 will present our solution for the generation of seamless transitions during motion.

3   Interactive Animation of Grass

3.1   Procedural control of wind effects

We are looking for an animation algorithm that

- produces a convincing deformation of grass blowing in the wind,

- is efficient enough to be apply to a whole prairie at interactive rates,

- and provides the user with an interactive control of wind effects.

These interactions are essential in applications like video games where characters should cut their path through wild grass, may throw bombs, or land onto the prairie with flying vehicles such as helicopters. Other effects, such as breeze or gusts of wind, will have to be stochastically generated in aim of increasing realism.

As in [15] we animate the prairie by computing the combined action of several wind primitives moving over the terrain. However, contrary to this previous work, we defend the idea of modeling directly wind primitives through *their consequences* on grass motion, rather than modeling the force or velocity field that causes it. This allows us to use procedural animation instead of physically-based simulation of grass blown by the wind which would have been impossible at the scale of a whole prairie.

3.2   The receiver concept

A wind primitive should be able to act upon a patch of grass whatever its current LOD. To do so, wind primitives send informations to grass through *receivers*. Every animate object has an associated receiver. At each frame, all the receivers receive information from wind primitives. Then, all the animated objects are drawn according to the informations stored in their associated receiver. A receiver can store information sent by several wind primitives.

In the 3D model, receivers are associated to the 3D blades of grass. In the 2.5D model, they are attached to the vertical edges of the semi-transparent polygon strips. Thus 2.5D texture can be animated by moving the two vertical edge as you would have done for blade of grasses (see Figure 3). We did not provide the 2D model with receivers because we have not animate it. That one one the main future improvement of this algorithm (we expect to use light maps to model the change of light reflection when grass bends).

To reduce the time needed to spread information over the prairie, the size of the information which is sent by wind primitives to receivers must be as compact as possible. We have seen that both the 3D and 2.5D receivers can be considered to be attached to a blade

of grass. That is why we choose to sent to receivers a direction and an index. The direction is the curve direction of the blade of grass, and the index describe how much the blade of grass is bent.

In fact, there are some differents kind of grass with specific size and stiffness. For each kind of grass, we have precomputed the range of possible bent postures. This can be done by using a physically-based simulator which look at the successive 2D postures of the blade when a constant wind starts blowing from a given direction. A given number of characteristic postures, indexed from 1.0 to -1.0, where 0 is the blade of grass's rest state, are extracted from these simulations (see Figure 6).

Each primitive uses the precomputed postures in its own and specific way, and can change the direction and the bent of all the blade of grasses which are under its influence. This choice will allow the same wind primitive to have different effects on the regions of the prairie planted with different kind of grass.
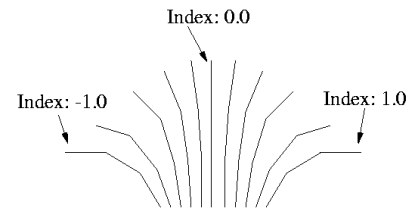

Figure 6: Precomputed postures resulting from a physically-based simulation. It is use by receivers to translate a given index into spatial positions.

3.3   Design of wind primitives

We want to control both the spatial extent and the action of a wind primitive onto the grass. To do this, we define a wind primitive as:

- *a 2D mask* that represents the spatial extent of its action, and which may move over the terrain during the animation. In practice, the mask is discretized over the patches of grass that tile the terrain. See figure 7.

- *an action* is a procedure that send information which describe the consequence (i.e. the posture index and the direction) of the wind primitive on any blade in the mask via the receivers.

All the receivers which are in the 2D mask of a given primitive store the direction and the index of deformations that can be applied to their attached blade of grass. Depending on the type of wind primitive, both posture index and direction can be function of time. In consequence, an action returns a time-varying vector, directed along the bending direction, and whose norm is the posture index.

Figure 7: Masks used for the gust of wind and whirlwind primitives, seen from above. These masks move over time.

Let us take the example of the gust of wind primitive. A gust of wind is defined by a strip shaped mask that translates over the terrain in an orthogonal direction to its main axis (see Figure 7). A blade of grass receiving the gust should bend in that direction, and
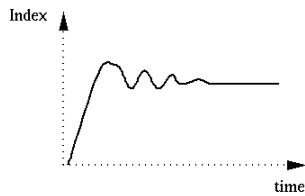
Figure 8: Physically-based precomputations give the posture index to use for blades of grass over time. The function above represents the variation of posture index when a constant wind starts blowing.

then oscillate back and forth before coming back to its rest state. During the physically-based precomputations, we have measured the parameters of these oscillations, i.e. the time variation of the blade of grass's posture index when a constant wind reaches the blade of grass (see Figure 8), and when this wind suddenly stops. These variations can also be interpreted as *spatial variations* of the posture index, at a given time step, over the primitive's mask. Indeed, as the gust of wind translates over the prairie, blades of grass that are at the front of the gust are just starting their motion, while those at its far end are just finishing their oscillation, back at their rest position.

In consequence, the functions giving time variations of indexes are also used for representing the procedural action of wind primitives. For instance, at each time step, a gust of wind sent posture and direction information to every receivers, according to their position in the prairie. In order to obtain a non-uniform result (as those generated by stochastic winds), it is necessary to add small oscillation of random direction and intensity around the prescribed position. A resulting image is shown in Figure 9.
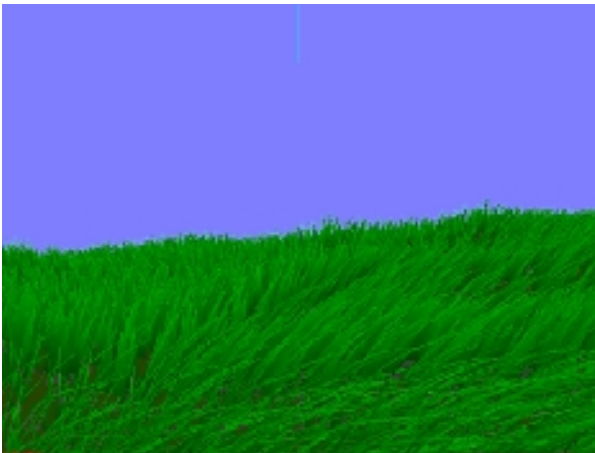

Figure 9: A gust of wind passing over a prairie.

Implementing a variety of other wind primitive can easily be done using the same approach. In addition to the gust of wind primitives, our system already includes gentle breeze, whirlwind, and blasts of air due to flying objects (see Figure 10).

The gentle breeze is characterized by an infinite mask. Our model for its action combines motion at two different scales: each blade experiences a succession of small stochastic oscillations, of random angles and intensity, around rest angles and bent positions that slowly vary over time. Here, a solution relying on the stochastic properties of turbulent winds, as those in [13, 14], would generate much more accurate results.

The whirlwind primitive uses a circular mask, the bending direction at each point being tangent to the circle, and the amount of bending being larger near the center of the circle (see Figure 11).
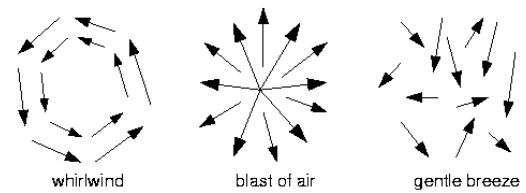

Figure 10: Every wind primitives can be describe as a vectors field. The direction of a vector is the bending direction and its norm is the posture index.

The prescribed posture index slightly changes over time in a circular way around the center of the mask. As in the gust primitive, stochastic oscillations around these prescribed positions are added to produce a non-uniform motion.
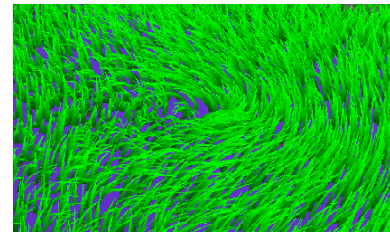

Figure 11: A whirlwind.

The blast of air primitive models a repelling blow around a flying object. The mask may be of any shape. It translates and rotates onto the prairie according to the motion of an object under the user's control. Here, the main bending directions are radial to the mask contour. As usual, stochastic small oscillations are added to achieve less regular grass deformations. A screenshot from an animation illustrating the use of this primitive are shown in Figure 12. Here, the user controls the motion of a flying saucer (here, a tea-pot). The latter always stays two meters above the prairie, generating a blast of air, as an helicopter would do.
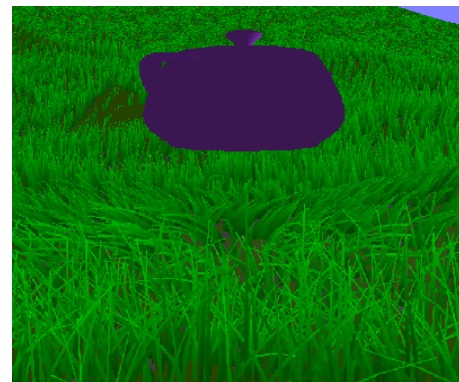

Figure 12: Interactive control of the blast of air primitive, through the displacement of a flying object. In this image, both some 3D grass patches and some 2.5D grass patches intersect the primitive's mask.

In practice several primitives may be active at the same time over the prairie, and pass through the same location. Due to the very nature of the primitives we implemented, we choosed to set a priority order between them. The primitive of highest priority simply cancels the effect of other primitives applied at the same location. Indeed, the blast of air and whirlwind will overcome the effect of gusts of winds, which will themselves mask the effect of gentle breeze. A more general solution would consist in computing, for

each receiver, a weighted sum of the action vectors resulting from the different primitives applied at this location. Nevertheless, mixing the influence of several wind primitives in a realistic is not easy at all. Actually, these is one of the biggest limitation of the procedural approach.

4 On The Fly Transitions between LODs

A standard walker's view of a prairie depicts regions represented with different levels of detail: the 3D representation is used close to the viewer, the 2.5D model is set up next, and the 2D texture is applied in the background. As the walker — and thus the camera — moves, adequate transitions between LODs have to take place. Generating seamless transition is made more difficult by the fact that patches of grass are animated. Transitions should not affect the visual perception of grass motion. This section describes our solution to this problem.

4.1 Scheduling a transition

The criteria for starting a transition between LODs are those presented in Section 2: if a patch of grass that was already visible at the previous time step changes status, i.e. if its discrete distance to the camera reaches a limit value, a transition to another LOD has to be started. Slightly different limit values are chosen for going back and forth from a LOD to the next one, so that a grass-patch will not stay forever in a transition state if the walker stands at the limit distance.

To be smooth, the transition will have to last within a few time steps. In practice, the duration of the transition is chosen smaller than the time taken by the walker to cross a patch of grass, to avoid generating too many active transitions at the same time. For instance, for a walker's speed of one meter per second, and patches of grass of 2.5×2.5 meters, we set the duration of a transition to twenty iterations, i.e. about one second.

4.2 Transitions between 3D and 2.5D LODs

A first remark is that applying a mere switch, or a fade in - fade out, between the 3D and 2.5D representations would give particularly poor results, due to the specific structure of grass: there is extremely little chance that the animated blades of grass of the two representations appear *exactly* at the same location from the walker's point of view, so this solution would result into blades of grass disappearing somewhere and reappearing elsewhere, which is unacceptable. An alternate solution for generating transition is to apply a morphing technique. However, standard metamorphosis does not apply to our case: Due to the specific nature of the LODs we use, morphing between 3D geometry and *a mapped texture* has to be defined.

Computing morphing between 3D blades of grass and the blades painted onto the semi-transparent polygons of the 2.5D representation is made possible by the way we construct patches of grass, described in Section 2: each representation uses the same number of blades, since each semi-transparent texture is the image of the 3D grass that lie between two given planes (see figure 13). However, we still have to define a morphing method that works during motion.

Our solution is to store, for each blade of grass painted on the texture, its relative coordinates in the texture space, as illustrated in Figure 14. This painted blade will serve as a target for a 3D blade of grass, which will move and deform to fit its coordinates, whatever the current motion and deformation of the semi-transparent polygons. More precisely, when a patch of grass is in the transition state, both its 3D and 2.5D representations are active at the same time. At each transition step, their individual animation are computed first,
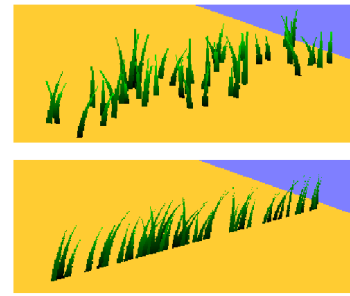

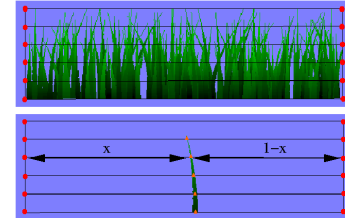Figure 13: 3D geometry and texture to be morphed.

Figure 14: The position of each control point of a blade drawn into the 2D texture is precomputed and stored in texture-space coordinates.

exactly as usual. Then, each 3D blade of grass is displayed at an intermediate position between the positions prescribed by the 3D representation, and its 2D target position. We use a mere linear interpolation for computing the intermediate position, the coefficients varying with the index of the transition step. For instance, if we are computing a transition from 2.5D to 3D, the 3D blade of grass is displayed first at its target position on the texture, and then reaches, within a few time steps, the position computed by the 3D grass-patch model. The opposite transition is used for switching from 3D to 2.5D. Although we use linear interpolation, the motion of a blade of grass is non-planar, due to the fact that each of the two LOD is animated during the transition (see Figure 4.2).

One may notice that the root of a blade of grass moves during transitions. Although this choice may seem unrealistic, it proves good in practice: being quite smaller than the motion at the tip, the root's motion seems almost un-noticeable when looking at an animated prairie for the walker's view point.

A noticeable artifact remains when we just switch from one representation to another. This creates a slight flash because the shading and the width of the blade of grasses are not exactly the same in the two representations. Adding an additional fade in - fade out is sufficient to fix the problem, since blades of grass are already
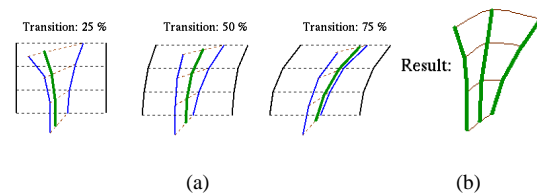

Figure 15: (a): Computing a smooth transition between the 3D and the 2.5D representations is done through a progressive linear interpolation, during motion, between a 3D blade of grass and its target blade in the 2.5D texture. (b): The resulting motion of the displayed blade is non-planar

located at the same place.

4.3 Transitions between 2.5D and 2D LODs

Here, the aim is to generate smooth transitions between the 2.5D representation, based on textured strips of polygons, and a 2D texture, which is not animated. Using a mere cross dissolve of the two representations is often sufficient, since the 2D representation is only used in distant regions. However, it gives very poor results at the outline of hills, where blades of grass of the 2.5D representation are silhouetted against the sky. Making them fade in and out of the sky color is really unrealistic.

To provide a better solution in this case, we have defined a more geometrical metamorphosis between the two representations. Firstly, instead of directly texturing the terrain in the 2D representation, we rather texture an offset of it, located at an average distance between the tip and the top of the grass. This helps displaying the outlines of hills at approximately the same location in the two representations. Then, we generate a transition by making the 2.5D polygons progressively grow (or un-grow) from (respectively into) the ground. Meanwhile, we let the 2D texture progressively disappear (respectively appear). This results in still noticeable, but not too annoying transitions.

5 Results

5.1 Performances

Many parameters of our implementation (for instance the size of the grass patches, the number of blades of grass they include, the number of line segments primitives per blade in the 3D representation, the number of polygons in the 2.5D representation, and the distance ranges for which a given LOD is used) are strongly related to both the quality of the results, and the performances of the system. These parameters should be tuned according to the power of the workstation on which the application runs.

If the size of the patches are small enough (about 3×3 meters), our application runs at a quasi-constant frame rate. Indeed, the number of polygons to render and animate varies slightly. For instance, if the 3D representation is only used at discrete distances from the viewer ranging from 0 to 2, the number of 3D patches of grass cannot exceed 8 for a fixed viewing angle of 40°. Each of the 3D grass patches takes the same amount of time to be rendered and animated, since it has a given number of blades of grass, rendered using a given number of polygons. Then, the amount of computation required by the 3D model can be easily bounded. Similar calculation can be done for the 2.5D model.

However, if the size of the patches is too large, many 3D blades of grass will not be visible because of the basic culling we use. A simple solution would be to reduce the size of the patches, but it increases the rendering time of 2.5D grass. In consequence, it could be a great improvement to do a more precise culling to decrease the waste of time due to blades of 3D grass which are out of the field of vision. This can be done using a quadtree for the tiling of the terrain.

In order to find out efficient instances of our system, we have tried to avoid bottlenecks by setting parameters in such a way that similar amounts of computation are required for treating the 3D and the 2.5D representations. We did not include the 2D representation in these considerations since it requires no animation time and almost no rendering time.

To give a more precise idea of the way of adapting our system, some sets of acceptable parameters are detailed below, together with the resulting performances on two different platforms: an SGI O2 and an ONYX 2 Infinite Reality. The table below compares three different implementation of the system, ranging from low to

high quality and using the same terrain with a size patches equal to 2.5 meters. The time needed to compute the animation is insignificant in comparison with the rendering time. The space needed for all the precomputings is about 20 megabytes. All the measures have been made on the same terrain of 100 m × 100 m. Figure 16 gives an idea of the respective visual qualities of the three implementations.
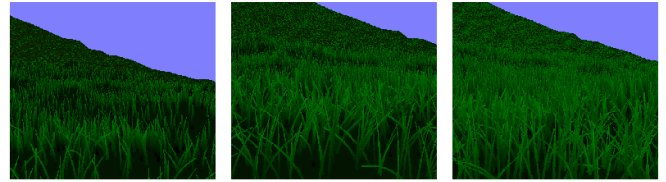

Figure 16: Views of a prairie respectively taken from the low quality (left), medium quality (middle), and high quality (right) instances of the system.

Quality	Low	Medium	High
Nb. of blades per patch	160	320	500
2.5D distance range	3-8	2-12	3-20
Nb. seg. per blade of grass	3	4	8
Approx. nb. blades per image	100,000	500,000	1,000,000
Frame rate on an SGI O2	5 Hz	4 Hz	2 Hz
Frame rate on an ONYX	25 Hz	12.5 Hz	8 Hz

Even if these measures were not made on standard PCs, we believe that they still demonstrate the applicability of the approach: new graphical PC workstations are already faster than SGI O2, and there performances are similar to ONYX.

5.2 Examples of animation sequences

A few frames from a walking session in the prairie are depicted in Figure 17. During this sequence, some artefacts are still visible during the transitions, especially when the 2.5D grass "grown" in front of the background (see the borderline between the sky and the prairie in the second and third pictures of Figure 17). Nevertheless, the animation is globally fluid and pretty, even if the vegetation is monotonous (see section 6).

6 Conclusion

We have presented a complete solution to the problem of prairie animation in real-time. Offering real-time control of such a natural scene is difficult because of the number of blades of grass to animate and display. Our solution is based on the use of three different LODs for representing grass: 3D geometry, 2.5D representation (i.e. volumetric texture), and 2D texture. Procedural wind primitives are used to animate these representations, through associated receivers that control their degrees of freedom. An algorithm to generate smooth transitions between LODs during motion has been described. This method handles continuous changes of camera position during a prairie animation sequence, and interactive control of wind effects. Lastly, this algorithm is applicable on various platforms by giving different ways of compromising between quality and efficiency.

Our model could be improved by integrating a series of sublevels of detail into each of the representations. As we stated earlier, several parameters, such as the number of line segments primitives per blade of 3D grass or the number of polygons in the 2.5D representation, can be modified. We think that tuning those parameters on the fly according to the discrete distance between a patch of grass and the camera could greatly increase the frame rate.

We are also working on a more complex finite state automata for switching from one representation to another. Our aim is to take into account the angle from which a patch of grass is viewed when choosing its representation. For instance, a view from above, which would be useful for viewing the prairie from the sky and for generating better quality images when the walker is at the top of a hill, should use shell texture layers over the terrain rather than vertical ones for implementing the volumetric texture representation.

Finally, we also plan to improve the aesthetic aspect of the prairie by extending our method to other vegetation primitives like bushes and trees. They could be modeled and animated in the same way, using a pre-computed physically-based animation for helping parameter setting, and by generating smooth transitions between animated 3D models and volumetric textures. It could also be very interesting to add some wind primitives. In fact, the procedural approach we use for animating wind could be generalized to model other effects such as grass interaction with humans or animals running through the prairie, or changes of illumination due to the sun position.

Although our method was developed for a specific application, as the approach we have defined is easy to generalize, we hope it will be applied to other type of landscape like desert or ocean for instance.
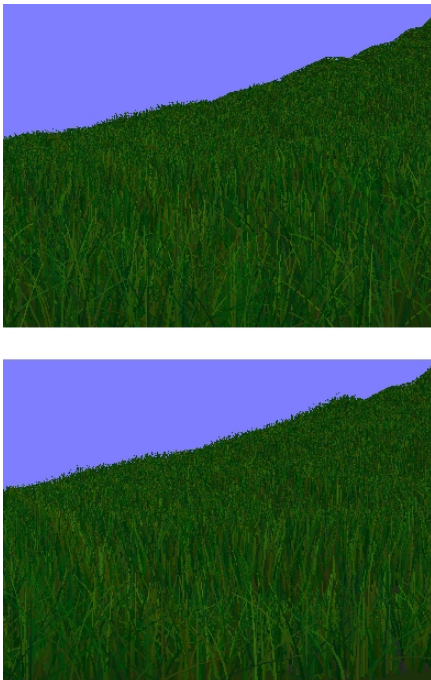

Figure 17: Two frames from "A walk through a prairie"

References

[1] Kenichi Anjyo, Yoshiaki Usami, and Tsuneya Kurihara. A simple method for extracting the natural beauty of hair. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 111–120, July 1992.

[2] Deborah Carlson and Jessica Hodgins. Simulation level of details for real-time animation. In *Graphics Interface'97*, pages 1–8, June 1997.

[3] Agnes Daldegan, Nadia Magnenat Thalmann, Tsuneya Kurihara, and Daniel Thalmann. An integrated system for modeling, animating and rendering hair. *Computer Graphics Forum. Proceedings of Eurographics '93*, 12(3):211–221, 1993.

[4] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan Barr. Adaptive simulation of soft bodies in real-time. In *Computer Animation 2000*, May 2000.

[5] James T. Kajiya and Timothy L. Kay. Rendering fur with three dimensional textures. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 271–280, July 1989.

[6] Jerome Lengyel. Real-time hair. In B. Péroche and H. Rushmeier, editors, *Eurographics Workshop on Rendering*, pages 243–256. Eurographics, June 2000.

[7] Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Loring S. Holden, J. D. Northrup, and John F. Hughes. Art-based rendering with continuous levels of detail. *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, pages 59–66, June 2000.

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

[9] Fabrice Neyret. Modeling, animating and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1), January–March 1998. ISSN 1077-2626.

[10] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[11] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 313–322, July 1985.

[12] Thomas W. Sedeberg and Scott R. Parry. Free-form deformations of solid geometric models. *Computer Graphics*, 20(4):151–160, 1986.

[13] Mikio Shinya and Alain Fournier. Stochastic motion - motion under the influence of wind. In A.C. Kilgour and L. Kjelldahl, editors, *Computer Graphics Forum (Eurographics '92)*, volume 11(3), pages 119–128, September 1992.

[14] Jos Stam. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Computer Graphics Forum*, 16:159–164, August 1997. Proceedings of Eurographics '97. ISSN 1067-7055.

[15] Jakub Wejchert and David Haumann. Animation aerodynamics. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 19–22, July 1991.