Real-Time Animated Grass

Brook Bakay[1,2], Paul Lalonde[2], Wolfgang Heidrich[1]

1) The University of British Columbia
2) Electronic Arts Canada
{bbakay,heidrich}@cs.ubc.ca, lalonde@ea.com

Abstract

We present a simple method to render fields of grass, animated in the wind, in real-time. The technique employs vertex shaders to render displacement maps with Russian-doll style transparent shells. Animation is achieved by translating the surface according to a local wind vector while preserving the length of the blades of grass. This technique achieves convincing results on current consumer graphics hardware and can be applied to other similar surfaces such as hair and fur.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

The realistic representation of outdoor scenes is a continuing problem in computer graphics. Real-time computer graphics has often relied on being "indoors" – using large occluding walls to facilitate detailed renderings. Outdoor applications have traditionally used very sparse geometry to describe the landscape, often greatly impacting its believability. In these systems, a field of grass could be reduced to a single texture.

High performance consumer computer graphics hardware has allowed for the display of complex detailed natural phenomena, such as the fur on a bunny[5], at interactive frame rates. However, this work has not yet been able to effectively animate the hair or grass displayed. Our work continues in this tradition, while using displacement maps and vertex shaders to leverage current consumer level computer graphics hardware for the animation of complex natural phenomena.

Our grass is composed of transparent shells, layered above the landscape. The vertices of these shells are moved in real-time to create the animation. Control of the animation direction, in response to a simulated wind field, is maintained at the vertex level. At each time step, each vertex is moved in accordance with the local wind direction a distance determined by a global intensity function. The wind direction is stored at the vertex level allowing for arbitrarily accurate wind movement over a landscape, and arbitrarily complex wind patterns. Vertices are divided into groups that use different intensity values facilitating advanced global wind effects such as waves across the landscape, attenuation in wind intensity or even whirlwinds.

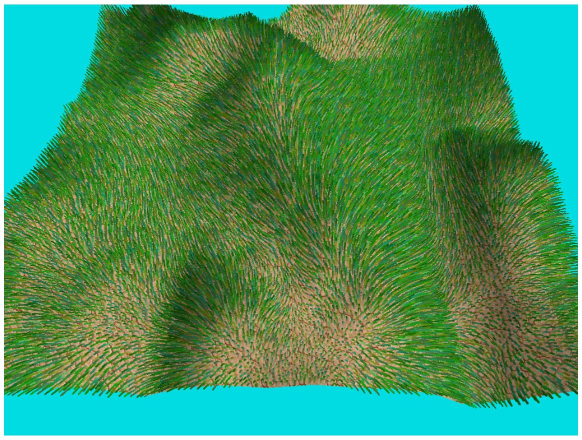
Figure 1: *A grassy knoll.*

2. Related Work

Volume rendering techniques have been applied to the problem of displaying finely detailed surfaces for many years. Kajiya and Kay[2] proposed using volumetric textures to render furry surfaces. Adding complex geometries such as grass or fur to a scene can greatly improve its visual impact, but much realism is lost if these geometries do not move in an appropriate way. Neyret[8] produced excellent results by animating the texture volumes, instead of the geometries themselves. This work was aimed at improving the performance of ray-tracing programs, and not directly applicable to interactive applications.

Now that hardware exists to render volumetric textures in real-time, even at a consumer level, work has been done with respect to rendering complex phenomena such as grass or fur in real-time. Lengyel[5] produced a very convincing furry bunny using several levels of detail, some of which included volumetric textures. However, this technique was very memory intensive and not suitable for animation. Subsequent work[6] alleviated the memory requirements somewhat, but animation has not been addressed.

Meyer and Neyret[7] discuss volume visualization using transparent slices composed of textured polygons, and achieve interactive frame rates. They mention that the animation techniques described by Neyret[8] could be applied as well. In their system, a separate texture is stored for each slice, which can lead to memory issues.

The displacement maps we use are patterned after those Kautz and Seidel[3] who built upon the work of Dietrich[1]. This technique is extremely memory efficient because it generates a texture volume from a single two-dimensional texture. Regrettably, their technique to generate shells in arbitrary slicing directions is not applicable here due to the high frequency data contained in the grass textures. We are limited to shells parallel to the model's surface. Shells perpendicular to the surface sample the base texture in one pixel wide strips and would miss a large proportion of the blades of grass. Kautz and Seidel do not address the animation of the volume.

Perbet and Cani[9] do discuss animation of volumetric textures to produce realistic grass in the wind. Their slices are perpendicular to the ground's surface, and thus their system is better suited to low views, close to the ground. They precompute a number of postures for each type of grass and send information to each blade regarding which direction to face, and which posture to assume. The two-dimensional textures for the slices are then computed from this information. Having data to control the motion of each blade of grass allows for some very detailed animations, but results in a performance penalty. This algorithm is dependent on the number of blades of grass in a scene. It would seem that a large amount of texture memory is also required, as each slice uses a unique texture.

Our technique is suitable for viewing from above, as in a flight simulator, or for walking in relatively short grass. It is fast, and because animation data is interpolated between vertices of the base mesh, fields of grass can be arbitrarily dense. Our technique is memory efficient, as all the shell textures are generated as required from one base texture. Lastly, our technique incorporates vertex shaders available on current consumer grade graphics hardware to further increase speed.

3. Approach

The grass is rendered through Russian-doll style transparent shells. Several copies of the base terrain mesh are "grown out" by displacing the vertices along their associated surface normals in a vertex shader. The shells are transparent except where a blade of grass intersects them. At these points, a cross section of the blade is contained within the texture. A single texture is used to generate all the shell textures, encoding the "height" of the grass in the alpha channel. The ground has no height and thus all ground texels in the texture have an alpha value of zero. Texels representing grass have non-zero alpha values depending on their respective heights, up to a maximum of 255. As we see in Figure 3, the white "dots" in the alpha channel of the texture map correspond to the green dots representing a cross-section of a blade of grass.

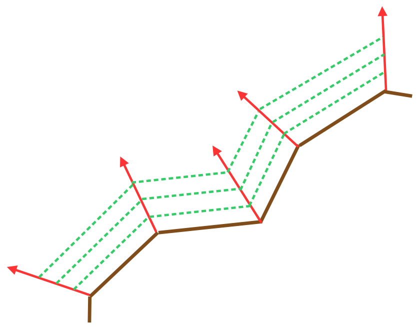
Figure 2: *Shells are extruded along the normals above the base mesh.*

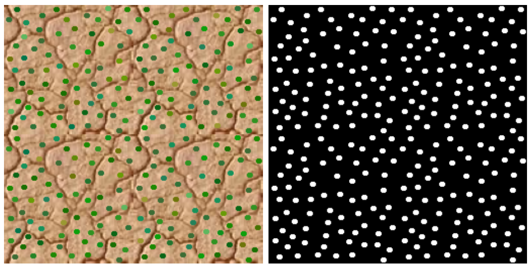
Figure 3: *The Grass Texture Map with Alpha Channel*

The shells are rendered in bottom to top order. We enable

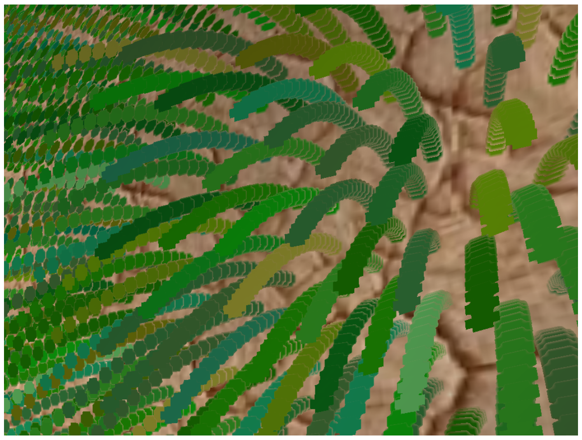

Figure 4: *Close view of grass rendered with 16 shells.*

the alpha test and set the alpha compare value to the shell's height before drawing each shell. The first shell, the base ground mesh, is composed of all the texels in the texture, so the alpha compare value is set to zero (with the alpha test method set to greater than or equal). Before the rendering of each subsequent shell the alpha compare value is set to a higher value. For example, if 10 shells were being rendered, the first shell above the base mesh would be rendered with an alpha compare value of 255/10 = 25. Any blade of grass with a height greater than one tenth of the maximum height would have a cross section included in that shell's texture. As Dietrich[1] points out, a state change could be avoided using the alpha channel of the diffuse colour to store the current shell height instead of repeatedly changing the alpha compare value.

Animation is implemented by moving each vertex along its "wind vector" – a vector stored with each vertex. Wind vectors are computed in a preprocessing step.

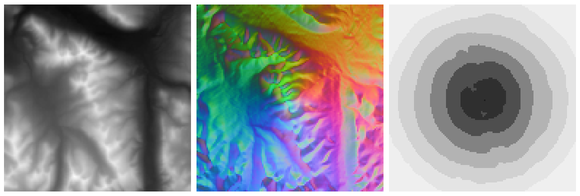

Figure 5: *Images Representing Wind Vectors. The wind source is roughly in the centre of the images. The original height map is shown at left. The X,Y and Z components of the vector are mapped to the red, green and blue components, respectively, of the centre image. The animation step to which each vertex belongs, in this case representing its distance from the wind source, is shown at right.*

There are several ways of generating appropriate wind vectors, ranging from heuristics to artist painting to a proper fluid dynamics simulation of wind moving over a landscape. Our actual animation algorithm will only consider the contribution of this wind vector that is perpendicular to the local surface normal. This restriction would be easy to overcome with additional per-vertex data, which would, however, degrade the performance slightly.

For the examples in this paper we have used a simple point source for the wind. To create the wind vectors, we project the vector from each vertex to the wind source onto the plane perpendicular to the normal vector by subtracting from the vector its projection onto the normal. The wind vector is given by:

$$\vec{V_w} = \vec{W} - \left(\vec{W} \cdot \vec{N}\right) \vec{N}$$

Where V_w is the wind vector at the vertex, W is the vector from the vertex to the wind source and N is the normal vector. All vectors are assumed to be normalized.

Given a wind direction, every vertex is moved along its normal vector and along its wind vector (perpendicular to the normal vector) every frame. The amount moved along these two vectors preserves the inter-shell distance, and thus the length of each blade of grass. In the absence of wind, the distance between shell vertices (along the normal) is a constant equal to the maximum height of the grass divided by the number of shells rendered. With the addition of wind, and thus vertex movement in a direction perpendicular to the normal, we must preserve this constant inter-shell distance or blades of grass will appear to grow and shrink as they animate. Each blade of grass is composed of segments of this constant length that are tilted appropriately in the wind. A windless moment would have all the segments "tilted" at zero degrees, and a moment of maximum wind would have the final segment tilted at 90 degrees. As we move along the blade from bottom to top, the tilt angle increases from zero degrees to a maximum of 90 degrees representing a segment moved into alignment with the wind – parallel to the surface of the landscape. We increment the tilt angle a constant amount between shells. This need not be so, and one could vary the "stiffness" of the grass by changing this increment. "Floppy" grass would do almost all of its bending in the first few shells, whereas stiff grass would do its bending further up the stalk. As we see in Figure 6, each segment forms the hypotenuse of a right angle triangle. Therefore, for each shell, the amount to move each vertex along its normal is given by:

$$n_i = \sum_{j=0}^{i} S \cdot \cos\left(\frac{i}{N} \cdot \frac{\pi}{2} \cdot I\right)$$

where i is the current shell, I is the current wind intensity at this vertex, S is the inter-shell distance and N is the total number of shells being rendered.

Similarly the amount to move each vertex along its wind

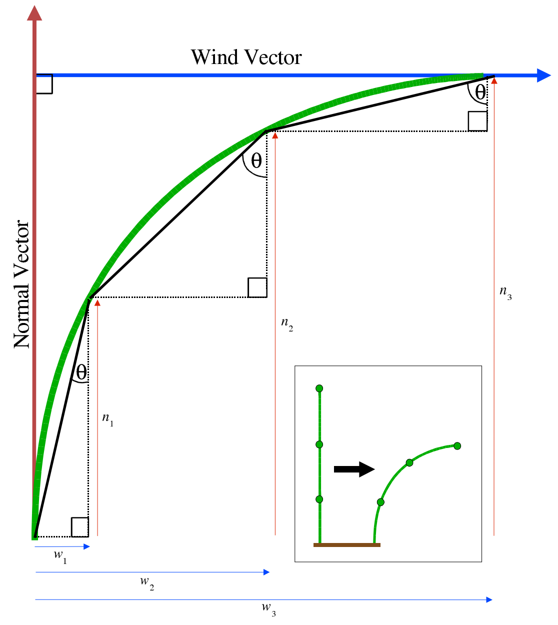

Figure 6: *The Length-Preserving Function to Bend a Blade of Grass. This blade of grass is rendered with four shells, including the ground. The blade is composed of segments of constant length (the hypotenueses of the right angle triangles) which are tilted in response to the wind intensity. A shell is created by moving each vertex a distance along its normal (n_i) and along its wind vector (w_i).*

vector is:

$$w_i = \sum_{j=0}^{i} S \cdot \sin\left(\frac{i}{N} \cdot \frac{\pi}{2} \cdot I\right)$$

Since the shells are drawn in bottom to top order, a running total is kept and the sums need not be recalculated for each shell. Each blade of grass, then, is the sum of its segments.

The wind intensity can be represented by any function the user chooses. For best results it should be periodic, continuous and vary between -1.0 and 1.0. Values outside this range would cause "over-bending" in the grass, which may actually be a desired effect.

The final piece of data sent along with each vertex is an integer representing its animation "step". We divide the world into segments based on their distance from the wind source. Blades of grass close to the source will experience the effects of a sudden spike in wind intensity before distant blades. This allows for a more realistic animation – including "waves" moving across the field, or attenuation of wind intensity.

The vertices are moved in a vertex shader. Values representing the amount of movement along the wind vector and the normal are stored in registers for each of the animation steps present. The shader references the values using the integer sent with each vertex, multiplies the appropriate vector with each value and and adds the result to the vertex position. This process is repeated for every shell rendered. The movement values must change as we travel up a blade of grass – the tip will be affected by the wind more than the base near the root.

Variable	Type	Frequency
S	float	once
AnimStep	int	pre-process
Normal	4-vec	pre-process
Wind Vector	4-vec	pre-process
Vertex Pos	4-vec	each frame, each shell, each vertex
n_i,w_i,θ	float	each frame, each AnimStep, each shell
I	float	each frame, each AnimStep

Table 1: *Computation Frequencies*

4. Results

The test machine for the following results has a 1.7 Ghz Intel Pentium processor with 512 Megabytes of memory. The video card is a 32 Megabyte NVidia Geforce3. All tests were done in 32 bit colour with a 16 bit Z-buffer. For the rendering, we used EAGL, a proprietary graphics API of Electronic Arts, Inc., as described by Lalonde and Schenk[4].

Framerates in this application vary depending on the coverage of the landscape on the screen. These numbers represent near worst-case values. Framerate increases as one pulls away from the landscape, indicating a fill-rate limitation to the method. However, when close to the landscape, frame rate also increases when some geometry can be culled. The screen coverage of the landscape for the tests can be seen in Figure 7.

Even at this stage, results are very encouraging. Although we have made no attempt at optimization, we are achieving real-time frame rates with convincing animation and visual quality. This technique has already proven itself useful for its target market, consumer games.

		Resolution	
Shells	Polygons	640 x 480	1024 x 768
1	512	293.5	130.8
9	4608	47.0	22.8
17	8704	25.6	12.4
33	16896	13.4	6.5
65	33280	6.9	3.3

Table 2: *Results in Frames Per Second*

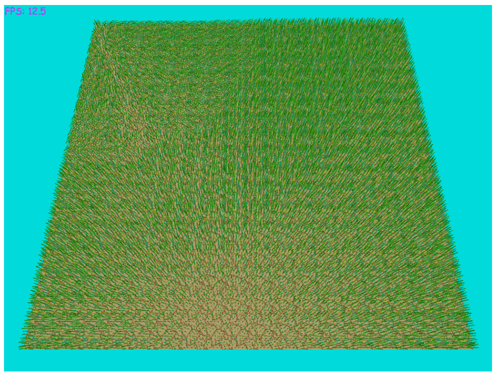

Figure 7: *Screen Coverage for Benchmarking*

Not surprisingly, this technique is limited by the fill rate of the graphics hardware. Adding shells, or increasing resolution, has a larger negative impact on performance than adding geometry to the scene. Adding shells does, of course, add geometry as well, and we noted frame rate increases when some of the geometry can be culled. As Lengyel noted[5], a covering of grass allows for a much less detailed base mesh than would otherwise be needed.

5. Future Work

We have not yet addressed proper lighting of the grass. Grass is an anisotropic surface, and significant gains in realism could be attained if it were lit as such. We are currently working on an extension of our algorithm that would allow us to simulate this effect.

Currently the volume is sampled (or sliced) at regular intervals, but this is an arbitrary decision. Future work will include concentrating the shells where they are needed most. Additionally, it may be possible to adapt Kautz' work, allowing for shells perpendicular to the mesh surface for viewing from grazing angles. Similarly, we use a linear relationship between the alpha value in the texture and the height of the grass and this need not necessarily be so. In fact, in the case of a field of grass, it might make more sense to have finer control at the upper end of the height scale.

Additionally, the two-dimensional textures themselves warrant further consideration. Lower level MIPmaps cannot be generated from the textures automatically, as the high frequency details of the grass will become lost. Custom MIPmaps, likely retaining representative blades from level to level, should improve visual quality at a distance. Further, in a multiple level of detail setup, it would be adventageous to reduce the number of shells for far off portions of the landscape, however doing so also reduces the density of the grass – the landscape becomes less green. This effect may be diminished by making the grass more dense on textures that will use fewer shells.

6. Conclusion

Recent advances in consumer graphics hardware have allowed for very realistic rendering of natural phenomena. Computer games are starting to emerge from the sewers into a lush, natural environment. We have presented a method that can animate detailed, believable fields of grass in real-time on current devices. Our method is fast, memory efficient and requires no special volume rendering hardware. Fields of grass may be arbitrarily dense and wind patterns may be arbitrarily complex.

Acknowledgements

The authors would like to gratefully acknowledge the support of the Tools and Libraries group at Electronic Arts, Canada, especially Patrick Martin who was invaluable in sorting out the technical details of EAGL. We would also like to thank the Imager group at the University of British Columbia.

This work was partially funded by a scholarship from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Dietrich, Sim, "Elevation Maps", *Technical Report* NVIDIA Corporation, 2000.
2. Kajiya, James T. and Timothy L. Kay, "Rendering Fur With Three Dimensional Textures", *Computer Graphics* **23**(3), pp. 271–280 (July 1989).
3. Kautz, Jan and Hans-Peter Seidel, "Hardware Accelerated Displacement Mapping for Image Based Rendering", *Graphics Interface 2001* pp. 61-70 (June 2001).
4. Lalonde, Paul and Eric Schenk, "Shader-Driven Compilation of Rendering Assets", *SIGGRAPH 2002* To appear.
5. Lengyel, Jerome, "Real-Time Fur", *Eurographics Workshop On Rendering* pp. 243-256 (June 2000).
6. Lengyel, Jerome, Emil Praun, Adam Finkelstein and Hugues Hoppe, "Real-Time Fur Over Arbitrary Surfaces", *Proceedings of the ACM Symposium on Interactive 3D Graphics* (March 2001).
7. Meyer, Alexandre and Fabrice Neyret, "Interactive Volume Textures", *Eurographics Workshop on Rendering '98* pp. 157–168 (July 1998).
8. Neyret, Fabrice, "Animated Texels", *Eurographics Workshop on Rendering '96* pp. 97–103 (September 1995).
9. Perbet, Frank and Marie-Paule Cani, "Animating Prairies in Real-Time", *ACM Interactive 3D Graphics Proceedings* (March 2001).

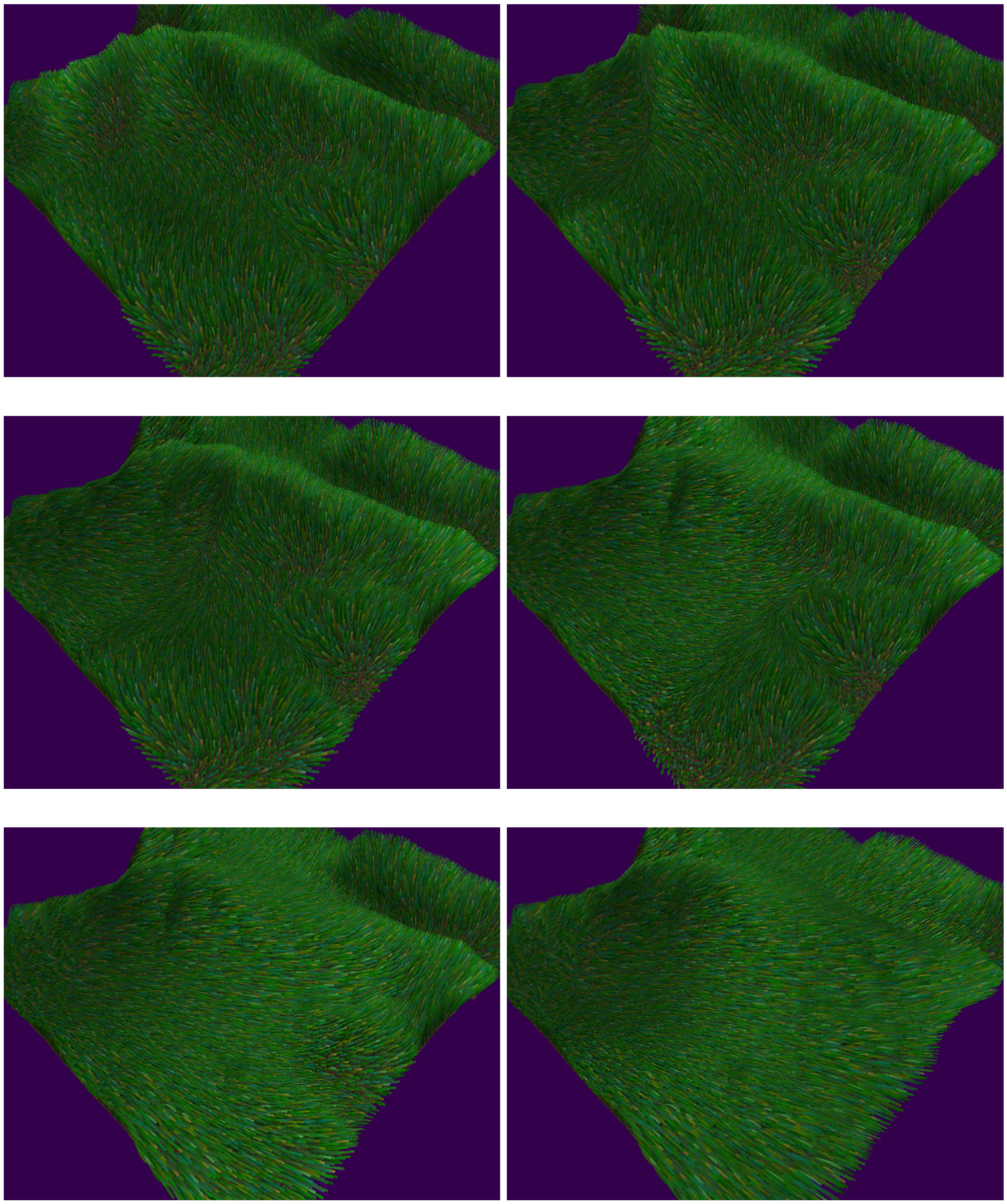

Figure 8: *A gust of wind across the prairie. These frames show a wind gust originating on the left side of the screen and moving toward the right side of the screen. Frame order: Top Left, Top Right, Middle Left, Middle Right, Bottom Left, Bottom Right.*