Grass project report

Ewen Randriamanohisoa*

Lund University
Sweden

Abstract

The goal of this project was to make grass the center of studies. It is usually just part of the scenery and therefore made just realistic enough to have the user not complain about it while focused on something else, but here, the focus was put on grass.

1 Introduction

This project tried to create grass that is realistic thanks to grass attributes. Lighting, post-processing and things of the same kind were not implemented in that project because they are generic methods to make anything look realistic. What this project aims to do is recreate the behaviour of grass using few lines of codes in shaders but still relying a lot on shaders in order to potentially be able to implement realistic grass in a project where it isn't the focus but still isn't left behind.

2 Algorithms or Application

The first thing we need to start working is some geometry. Grass strands have a pointy tip so the first thing one can think about as a model is a triangle. It turned out it was not realistic enough. What is needed is a function that stays globally the same for a certain amount of time before finishing in sharp tip. Seems like an exponential function was the best choice, after changing some parameters. See Figure 1.

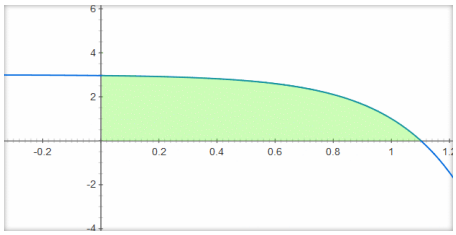

Figure 1: Curve $y = (1 - e^{(x-1)*4}) * 2 + 1$

But once we get that shape, we need it not to stay vertical and stiff. It needs to fall like a grass strand would. To do this, we simply fire a particle with a certain initial velocity and under a certain gravity, and we put the previous shape on the path of that particle. The equation of that path is just that of a parabola. Only the parameters for it are to be tuned.

Since we plan on adding wind and we want performance, we actually code this part in the shader. The geometry we created earlier is only composed of vertex stashes, as if the strand had a length of zero. Then the shader expands that length by putting the vertices on the path of the particle.

However, if we always use a "vertical" gravity, it makes grass always fall towards the same direction when on a slope (towards the downhill direction). But we want something more as in Figure 2, where the strand can also fall towards the uphill direction. For this reason, gravity is an attribute that is different for each strand and follows the normal to the terrain (or rather its opposite).

*e-mail: ewen.randriamanohisoa@ensimag.grenoble-inp.fr

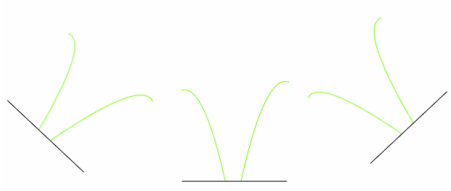

Figure 2: Grass on a slope can fall towards the uphill direction

Then, we need to add wind. The main program manages a wind vector that randomly but still smoothly changes its direction, and a wind force that changes the same way. This wind vector is then passed to the vertex shader, multiplied with a sine to add oscillation and simply added to the forces that affect the particle the strand follows.

But if we do this, it will actually just stretch the strand towards the direction of the wind. All parts of the strand will still face the direction its root is facing even though its tip will be moved. We need the strand to actually turn towards the direction of the wind. For this, we use derivatives of the path of the particle to compute a moving base and we place the vertices according to this base. Now, the whole strand turns towards the wind, which isn't what we wanted either, its root shouldn't move. Using simple linear interpolation between the two methods, we can have a root that always faces the same direction, a tip that turns towards the wind, and a coherent shape in between.

Shaders also use simple functions often based on sinus to give colors to objects. The cubemap mixes yellow and pink using sines in a pattern which repetition doesn't strike too much and grass strands have lines on them when zoomed in. These last lines are however faded out when the camera is farther away in order to avoid artifacts. The same kind of combination of sines is also used to shape the terrain, giving it an almost random feel but still forcing the creation of a crease in order to check the behaviour of strands in slopes.

3 Results

Code-wise, the result is just a few lines, and only simple ones. But even though it seems trivial when reading it, it wasa lot of trial and error and rewrite before getting to this stage of coherent code that produces this realistic behaviour. But due to its simpleness, it also gets honorable performance as shown on the FPS counter on previous screenshots. Performance can obviously be tweaked by changing parameters here and there, such as attributing less triangles to grass strands.

Here are some screenshots showing what grass and its environment look like.

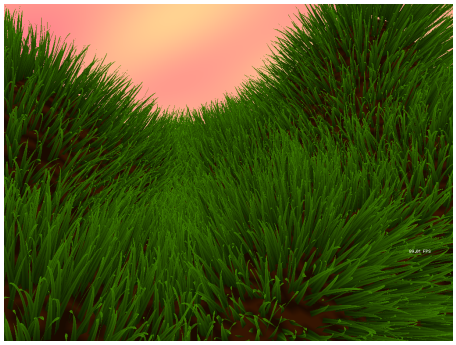

Figure 3: Grass behaves well on slopes

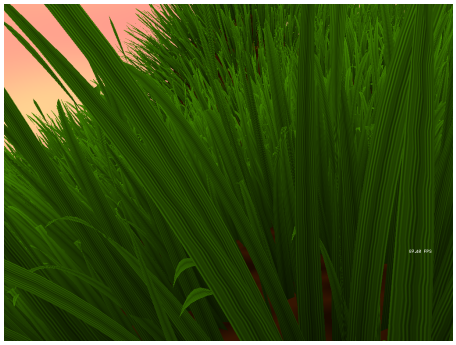

Figure 4: Lines on near strands, not on distant ones, and light artifacts inbetween

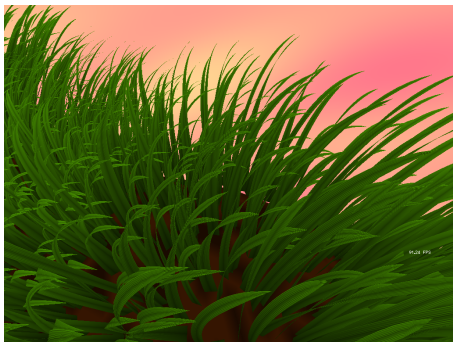

Figure 5: Wind affects tips but not roots

Fur was also implemented, as a variation of grass since they share most of their properties, as shown on Figure 6.

4 Discussion

Perlin noise was attempted for the ground under the grass but it doesn't seem to work well. It seems like vertices only have two colors and only the interpolation from the shader gives a somewhat smooth look. But the noise was supposed to be smoother itself. Since that was only under the grass and not very visible, nothing was attempted to try and fix it.

Something else that didn't see a lot of effort to improve was geometry generation. It was only written in order to be easily modifiable to add features or change its behaviour (such as the moment

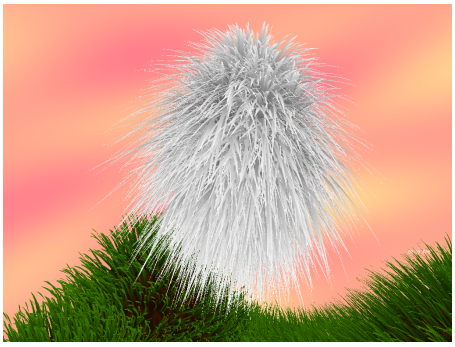

Figure 6: A furry ball using grass code

when it was decided geometry would be stashes and not direct strands). Which means it could probably use some optimization, both speed-wise and space-wise. But size doesn't often matter in a project, especially in one without any kind of images, and speed isn't so important when generating geometry, it's only a one time step at the beginning. If speed was really needed at that point, it would probably be a good idea to store pre-computed data and never have to generate grass again at launch.

Since generating fur and grass is so similar, they could also share their code, but really share functions instead of using duplicated code with very few lines changed. Code was duplicated when grass was working and fur was to be added, in order to implement fur without breaking grass. But now that they both work, it would be a much cleaner piece of code if it were a single function with different parameters for example.

And of course, things that can be implemented to increase general realism are generic methods like lighting and shadows, but as written in the introduction, that was not the purpose of this project.