Introduction to Computer Graphic

Professor Mark Pauly

Project Report

Grass Rendering

Authors:
Maëlle COLUSSI (205612)
Raphaël DAVID (203712)
Lucas VANDROUX (202343)

Sections:
Computer Science
Communication Systems

December 6, 2012

Motivation

Grass rendering is used everywhere nature is needed. In all animation movies or video games we can always observe a field of grass at one time or another. That's why we decided it would be a good idea for a first project to implement a solid and realistic grass rendering system in real time.
Raphaël had tried before to create some grass using Blender and was motivated to do more in the field. He succeeded to convince the entire team. Moreover, the presentation of beautiful landscapes in the presentation made by the two guys from pixar really motivated all of us in doing a realistic rendering of a natural scene.

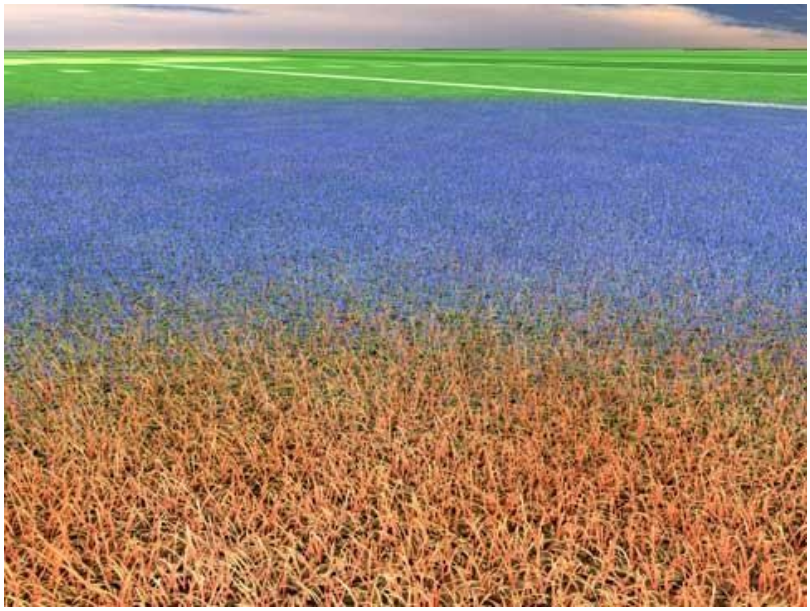
Crysis 3 - Electronic Arts - Crytek - CryENGINE® 3
http://www.jeuxvideo.com - December 2012

Brave - Disney - Pixar
http://www.pixar-planet.fr - December 2012

Related Work

Modeling the grass

One of the main project that inspired us is the work of Kevin Boulanger (http://www.kevinboulanger.net/grass.html) where a scene is decomposed in three parts: “real” grass objects in the foreground of the image, texture in the background and billboards in between. In our project, we only deals with the billboards because the time to realize our project was not long enough for us to work on the 3 parts. Besides, billboards offer a good compromise in term of realism and performance.



Levels of details of the grass rendering algorithm
http://www.kevinboulanger.net/grass.html - December 2012

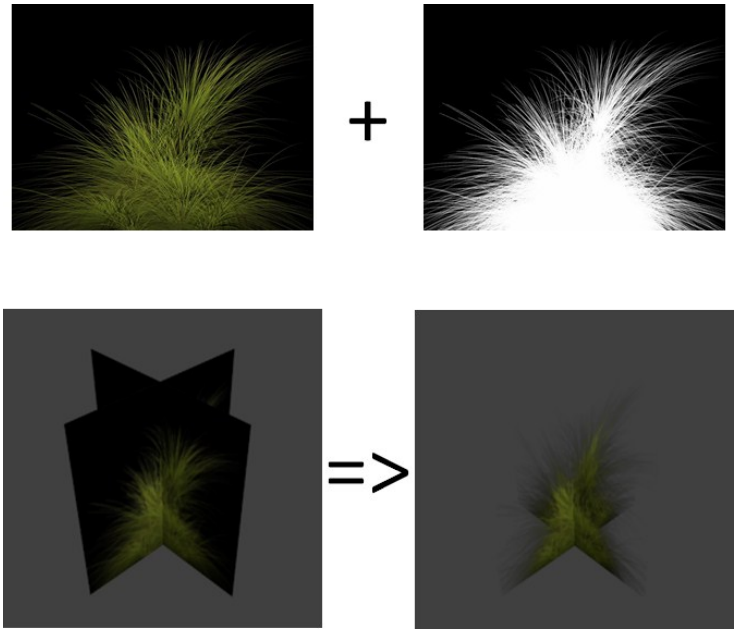
Generation of points

About the generation of particles where blades of grass will be drawn, we used a variant of the blue noise described in this paper: http://johanneskopf.de/publications/blue_noise/tilesets/index.html. Our method uses a unique pattern of points and translates it to generate the particles. The paper explains that it is more realistic if we use more than one pattern and choose randomly each one of them to use in order to avoid the pattern to be noticeable on the terrain. A random rotation of the terrain is also encouraged. We chose to use only one pattern because the repetition of the pattern cannot be detected with the human eye on our scene and because we wanted to keep our particles system easy to implement and its code easy to read.

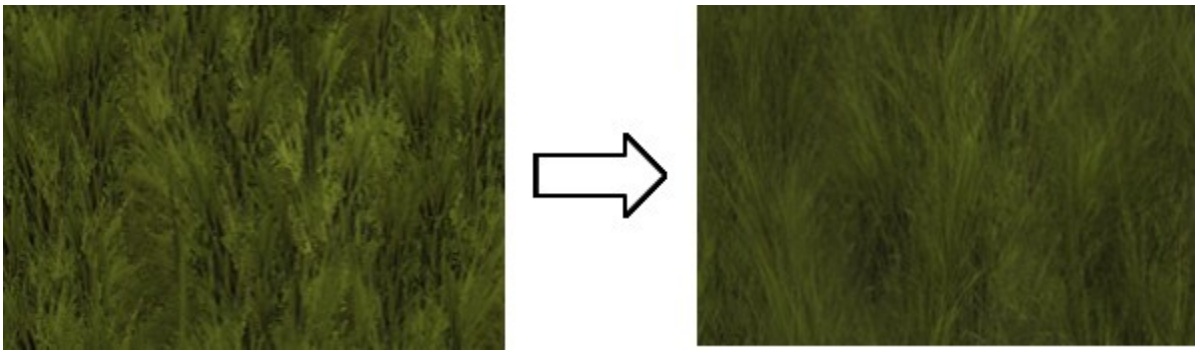
Method

Blades of grass

For the grass itself, we use cross-billboards with an alpha texture, and we apply alpha-to-coverage in order to reduce the flickering. This flickering is due to the fact that the multi-sampling has only effects on the edges of primitives and cannot deal with simple textures.



Cross-billboards with an alpha texture used to draw grass
Personal project - November 2012



Comparison between grass before and after applying alpha-to-coverage
Personal project - November 2012

Particles generation

For the generation of particles, we use a small pattern of points (no more than 50) that is translated until the entire plane of the field is covered (coordinates (x; 0; z)). At this point of the algorithm, we have the x and z coordinates for each particle. In order to draw grass on a bumpy terrain, we must find their y coordinate. This can be done by using the uv coordinates of the terrain that we generated with Blender. We rescale these coordinates in order to obtain the projection of our terrain on the plane where we draw the particles. Then we can find the triangles of the terrain where each particle belongs by looking for the three points of the rescaled texture coordinates that are the nearest for each particle. This can produce artifacts, but we only have to discard the incorrect points. Finally we can compute the barycentric coordinates of each particle point and use them with the 3D corresponding triangles of the terrain to find the final position of our particle.

Grass density

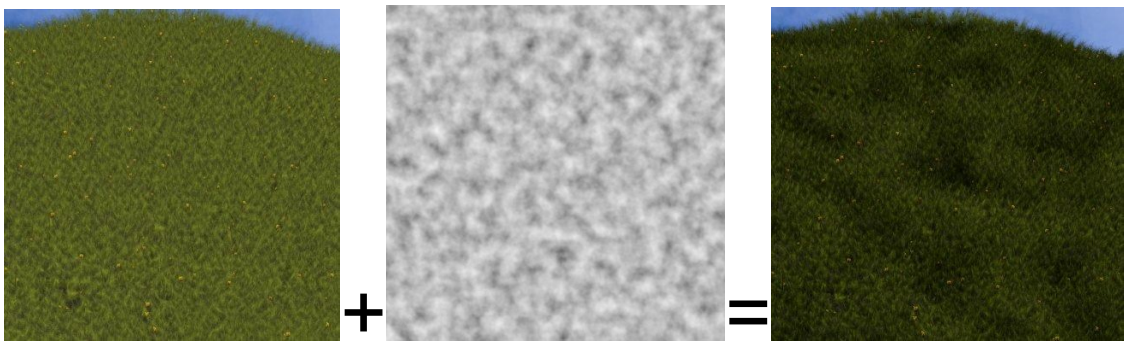
In order to have a variation of density in our grass, we must create a black and white texture that we use in order to decide if a blade of grass should be drawn for a given particle or not. That means that for each particle, we read the pixel value of the density texture that corresponds to the position of our particle. This value will modify the probability for a blade of grass to be drawn. That means that we generate a random value between 0 (black) and 1(white) and we only draw the blade of grass if this number is smaller than the diffuse value. We also use the diffuse texture to modify the random size of the grass cross-billboards, so that the smaller blades of grass stand on the smallest density values. The density affects also the alpha value of the blades of grass (a small density involves a small alpha value).

An exemple of diffuse texture
Personal project - November 2012

Result of the use of the diffuse texture
Personal project - November 2012

Color variation

In order to avoid that our grass looks too much homogeneous, we decided to implement a color variation by the use of a cloudy texture. The principle is the same as with the grass density except that now the value of the texture modify the final color of our grass in the pixel shader.



Result of the color variation
Personal project - December 2012

Wind simulation

Our wind simulation is a simple translation of the higher points of the blades of grass. There is two translations: one along the x axis and one along the z axis. We decided to implement our function in the vertex shader in order to improve the performance of our program. As our wind simulation needs to be fluid we decided to use a periodic function. Concretely, we use the sinus of an imaginary “angle” that we increase during the simulation to produce the movement. Note that we do not want to have a completely noisy wind. It should rather move by blow. That’s why we do not give the same initial angle value to our vertices. We rather use a wind_pattern_distance which corresponds to the distance in which a period of our sinus function happens (concretely 2*Pi). So the initial value of the angle for a given vertex is computed according to its position modulo the wind_pattern_distance. Doing so will avoid each blade of grass to have the same movement, but will avoid too a completely noisy wind.

Results

Final result of our grass rendering function
Personal project - Decembre 2012

Final result of our grass rendering function 2
Personal project - Decembre 2012

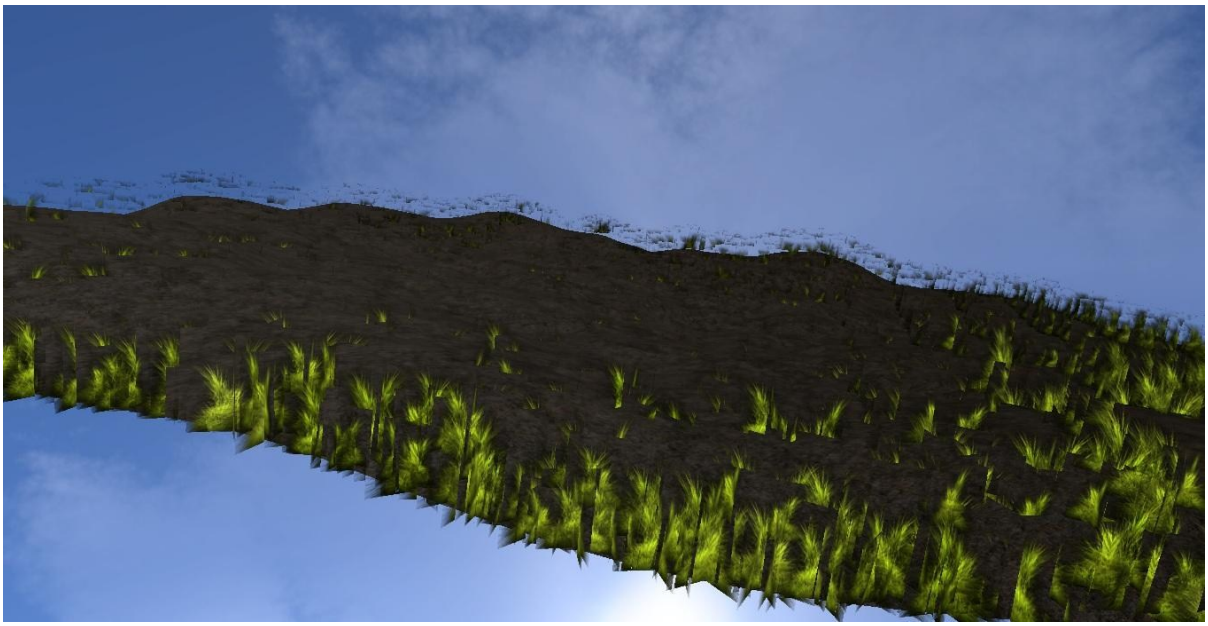
Final result of our grass rendering function 3
Personal project - Decembre 2012

The initialization of the project takes one minute or two, then the simulation is quite fluid, even with the wind.
Here is the list of the different shortcuts for the simulation:
g : enable or disable the grass presence
t : enable or disable the use of textures
z : enable or disable the transparency
a : enable or disable the alpha to coverage
c : enable or disable the color variation
h : show the help in the console

Discussion

Compatibility

The main problem with openGL is to assure a compatibility for every graphics cards. As you can see below, not all the computers can apply alpha_to_coverage. Those sorts of problems would not have appear if we had chosen to implement it as a ray tracer, because it would have been cpu-compatible. However, we would not have been able to render a simulation in real time.



Alpha-to-coverage activated on a Acer Aspire One 772 with a Radeon HD Graphics Computer of Maëlle - December 2012

Possible extensions

An idea for a further extension to add to our work would be to implement the collisions. Doing so would allow us to have a flexible grass where we could roll a ball.
Another thing that would be interesting would be to use our particles system to render other elements such as a forest or a city.

Ball rolling in grass
http://www.youtube.com - December 2012