Real Time Rendering Using Modern GPU Techniques

Jonatan Olofsson [jonol402]
Mikael Silvén [miksi016]

2012-05-13

1 Description

This paper describes the project made by Jonatan Olofsson and Mikael Silvén in the course *TSBK07 - Computer Graphics* at Linköping University, spring 2012. The project aimed to create real-time rendered grass, and to demonstrate and test a few uses of modern GPU algorithms.
Basic ideas and inspiration was taken from a previous Master's Thesis on the subject of real-time grass rendering[1].

2 Grass

The grass rendering was the first part of this project to be finalized. The grass rendering uses a OpenGL shader pipeline with the following steps.

Vertex Shader Calculates points in the terrain where grass strains should be rendered

Geometry Shader Generates the grass strains

Fragment Shader Paints the grass strains

Most of the work on generating the grass is done in the geometry shader. The geometry shader is fed with the position of the grass seed, which is extruded to the full grass strain. As the grass strain is generated - from the bottom-up - a modeled wind force is applied to bend the grass-strain with the wind.
To give an even more realistic appearance, each grass strain is associated a random phase offset from the applied wind sinusoid. This causes each strain to move independently, without removing the sense that all strains are moved by a common wind.
Depending on the distance between the generated grass strain and the camera, different level of details are selected for the grass, to reduce computational load on grass strains far away. The higher the level of detail, the more vertices are used to build the strain. Each strain is built of two parallell triangle strips in an angle to create a 3D-effect on each grass strain. The texture applied to each strain is also selected to enhance this 3D-effect, for instance adding shadow in the lower parts of the strain.

[1] http://illogictree.com/blog/masters-thesis/

3 Frustum

To reduce the computational load, a technique called *frustum culling* was studied and implemented. The idea is to render only the parts of the world that are inside the visible frustum. In this project, the focus of the frustum culling was the terrain and grass. Since both these are present on the ground level, we can limit the culling to this plane. By imposing certain restrictions on the orientation of the viewing frustum, we can map the corners of the frustum-intersecting ground plane to world coordinates. Since we know the ground plane in world coordinates and the frustum bounding box in frustum coordinates, we need to solve the multi-dimensional two-point boundary-value problem in (1).

$$\begin{pmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ * & * & * & * \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} * & * & * & * \\ y_{10} & y_{11} & y_{12} & y_{13} \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \tag{1}$$

The system of equations in (1) is easily solved numerically, yielding world coordinates which can be mapped to the terrain, as visualized in 1.

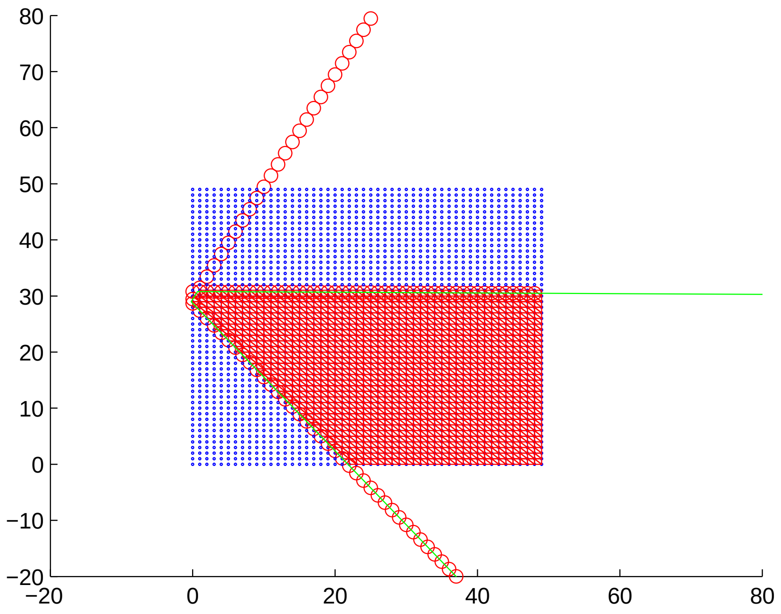

Figure 1: Frustum culling visualized from above.

The extracted corners were then used to do a single-dimension sweep in the x-axis; from the leftmost corner of the visible terrain, to the rightmost. For each x-coordinate, an OpenCL step was applied to extract all the visible indices with that coordinate. Doing this in OpenGL memory-space ment that a minimal amount of data had to be copied before the final rendering stage, when a single OpenGL call could render the full visible world, thanks to the inherent sorting of the OpenCL pre-processing stage.

4 Post-processing Effects

5 Conclusions

In a project like this, there is always work left to do when the project is due. Thus, further work would include bugfixes and further optimization of the code as well as extensions to for instance reduce far away grass to mere textures.
We have however, during the course of the project, demonstrated advanced real-time rendering of grass, utilizing both an OpenCL pre-processing stage on the GPU as well as advanced use of the OpenGL shader pipeline.