M. Stamminger, C. Weber *Erlangen,* ***Monday, 03.11.2014***

Computer Graphics - Programming Exercises

Assignment 3 [6 Points] (Sun Shader)

Put your lighting shaders from the previous assignment aside for a while, you will now create a special effect shader for the sun. The vertex positions and the fragment colors should oscillate in a certain way to create an illusion of a fireball (see the reference implementation in the solution folder). The exact tasks of the vertex and the fragment shader are as follows:

Vertex shader:
The vertex shader should alter the position P of each vertex along its normal vector $\mathbf{n}$. Use the sin-function to let the vertex oscillate smoothly between $P_{min} = \mathbf{p_0}$ and $P_{max} = \mathbf{p_0} + a\mathbf{n}$ where $\mathbf{p_0}$ is the original position and a is an individual maximum amplitude of the vertex. The maximum amplitude should be as random as possible within a certain (reasonable) range. It is also very important that the phase of the sine-wave of each vertex is randomly shifted, otherwise the entire sphere would just get bigger and smaller.
Thus, you need two pseudo-random numbers for each vertex. Find a way to generate such numbers in the vertex shader from the information you get from built-in vertex attributes.

To make the smooth vertex-motion possible, there is a global time parameter `t` in the main program which is incremented each frame and available in the shader as variable `Time`.

The vertex shader should provide the following varying variables to the fragment shader: The current phase of the vertex, its normal vector and the viewing direction.

Fragment Shader
The fragment shader takes the interpolated phase to determine a base color C_{base} of the fragment. The color should be yellow at the maximum phase (corresponding to a vertex position P_{max}) and red at the minimum phase (vertex position P_{min}). In between, it should smoothly blend between these two color values. In the RGB color model, you can achieve this effect by altering the green component while red=1 and blue=0.

To let the sun appear more yellow at the edge, add a constant yellow color C_{yellow} to the base color, weighted by a parameter α:.

$$C_{final} = C_{base} + \alpha \cdot C_{yellow}$$

The weight $\alpha \in [0, 1]$ controls the amount of yellow added to the base color and should be the bigger, the flatter the viewer looks onto the surface point of the sphere. It should take the maximum value when the viewing direction and the surface normal are perpendicular and the minimum if they are parallel.

Additional Information

- You only have to edit the shader code in the files `sun_VS.glsl` and `sun_FS.glsl`. There is no need to edit the files of the main program.
- As you should already know, the individual RGB color components are in the range $[0, 1]$ (e.g. $(1, 0, 0)$ is pure red). However, you do not need to clamp the values manually to this range, openGL automatically does that for you. If, for example, the color value $(2, 2, 0)$ is assigned as final color, it will automatically be clamped to $(1, 1, 0)$.
- The glsl-function `noise` is not an option for the generation of a pseudo random number since it doesn't work on most GPUs (it will always return the value 1)!
- Since the speed of the effect highly depends on the hardware (and on the fact whether you operate on a local or a remote machine), the increment of the time-parameter can be increased and decreased in the program by pressing +/-.

Implementation Guidelines This assignment will be the last with the old OpenGL fixed function pipeline. Starting with assignment 4, we will abandon the matrix stack and the old, state machine like lighting model. That means shaders will become an essential part of every future assignment. So take your time and study the math behind lighting and have a look at the code responsible for managing the shader.

Good Luck!

Your source code will be copied from your handin directory on:
Monday, 10.11.2014 14:00 pm
all subsequent changes cannot be taken into account!