Prof. G. Zachmann
P. Dittmann

Universität Bremen
Fachbereich 3
Arbeitsgruppe CGVR
20.07.2019

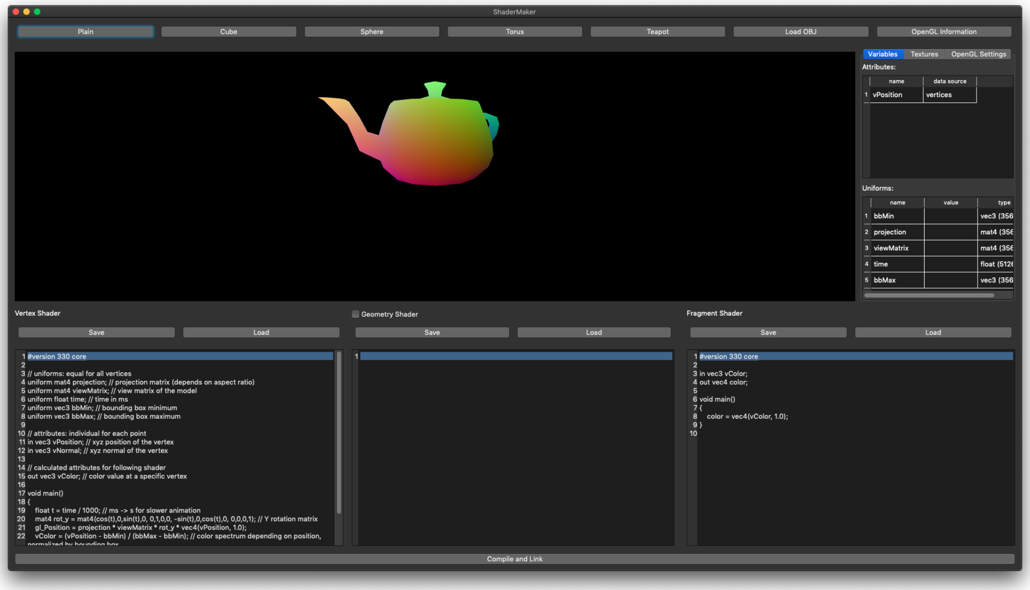
Summer Semester 2019

Assignment on Advanced Computer Graphics - Sheet 5

Due date 01.07.2019

<u>Remember to add me to your GitLab project:</u>
Account: dittmann
E-Mail: dittmann@cs.uni-bremen.de

Download the new version of ShaderMaker at http://cgvr.cs.uni-bremen.de/research/shader_maker and get familiar with the OpenGL Shader Language 3.3[1] (GLSL).



Each shader needs '#version 330 core' in it's first line. Uniforms are like read only, global variables. Attributes contain information about each vertex like position or normal. 'in'-attributes are input of a shader and 'out'-attributes will be send only to the <u>next</u> shader. The order is: vertex shader, geometry shader (optional), fragment shader. The fragment shader has exactly one 'out'-attribute, which contains the result color.

[1] https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.3.30.pdf

Exercise 1 (A basic OpenGL shader, *1 Points*)

An OpenGL shader for rendering consists at least of a 'vertex shader' and a 'fragment shader'. Modify the example fragment shader, that the color given by vertex shader is inverted and save both shaders as 'exercise01.{vert|frag}'. Color values should always be between 0 and 1. Values outside of this range will be clamped.

Exercise 2 (Color animation, *1 Points*)

Restart the application to restore the example shaders. Modify the example fragment shader, that the color of the object is animates. Use the **uniform float time**, which contains program uptime in milliseconds for that. You should down-scale the value for smooth animations. Math functions like sin and cos might be helpful. Save both shaders as 'exercise02.{vert|frag}'.

Exercise 3 (Movement animation, *1 Points*)

Restart the application to restore the example shaders. Modify the example vertex shader, that the objects changes it's direction from time to time. Use the **uniform float time**, which contains program uptime in milliseconds for that. You should down-scale the value for smooth animations. Math functions like *sin* and *cos* might be helpful. Save both shaders as 'exercise03.{vert|frag}'.

Exercise 4 (Pattern, *5 Points*)

Restart the application to restore the example shaders. Modify the example vertex and fragment shader, that the objects gets a pattern like chessboard or circles, depending on the vPosition attribute. Textures are not supported, yet. Save both shaders as 'exercise04.{vert|frag}'.

Exercise 5 (Geometry Shader, *2 Points*)

Modify the given geometry shader from exercise page[2], the all points starts scattering depending on **uniform float time**. Use a local variable for the maximum allowed distance. Save both shaders as 'exercise05.{vert|geom|frag}'.

Exercise 6 (Bonus, *2 Points*)

Write another 'fancy' shader called 'bonus.{vert|geom|frag}'.

[2] http://cgvr.cs.uni-bremen.de/teaching/cg2_19/