Realtime Rendering

Assignment 2

Assessment: 33%

Deadline: Fri 15/9/2017 9pm (end week 8)

Changes and Clarifications

Base Code

The following base code is made available as a starting point. It is a procedural sine wave program, which uses OpenGL Mathematics (GLM) for transformations rather than (deprecated) OpenGL transformations.

Aims

This assignment is intended to have students learn about shaders for performing transform and lighting calculations, another key aspect of modern graphics programming, along with providing data using buffers.

Individuals or Pairs

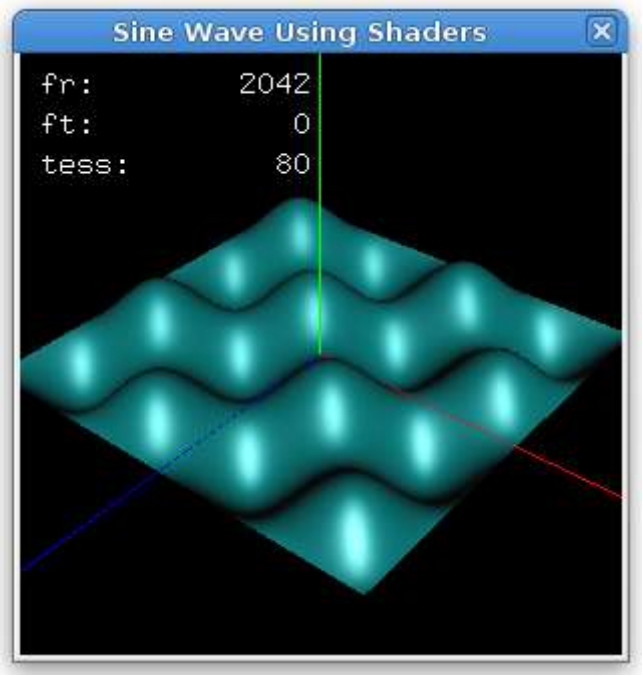
The assignment may be done in pairs.

Problem Description

This assignment is intended to allow students to gain experience in writing shaders and using GLM for transformations instead of OpenGL. It has four main parts:

- Implementing lighting using shaders and comparing per-vertex and per-pixel shading in terms of both performance and image quality.
- Using GLM in conjunction with shaders for transformations.
- Performing per-vertex animation using vertex shaders, i.e. using the GPU rather than the CPU.
- Using buffers for providing data.

The assignment is to use a procedurally generated wave. The images below show a wave rendered using shaders using Blinn-Phong lighting (same as in the OpenGL fixed pipeline) and Phong lighting.



The assignment is in part based on using the shaders and code from the shader tutorials week 5, week 6, week 7, the brick example from the Orange Book and the Lighthouse lighting (and other) tutorials.

Start with the sine wave generator program (linked above) which uses GLM for transformations and can switch between fixed pipeline OpenGL lighting and CPU based lighting.

- As in the first assignment, there should be a performance meter, with the ability to switch output of frame rate, frame time and other state data as text to the console (stdout) instead of on screen. Use a rate of 1 update/second.
- As for assignment 1 buffers should be used for data, although that can (and should) be done after getting shaders working.
- When using shaders to perform lighting calculations use uniform variables to control global settings.
- The shaders should give the same results as the original sine wave program when using either fixed pipeline lighting (with no shaders) or CPU based lighting calculations - all three approaches are using the same calculations, but differ in where and how they are being done.
- The following interative controls:
  - a - toggle animation
  - b - toggle smooth, flat shading
  - c - console
  - d - toggle directional, positional lighting
  - f - toggle gpu (fixed)/cpu lighting calculation mode
  - g - toggle shaders
  - H/h - increase/decrease shininess
  - l - lighting
  - m - Blinn-Phong or Phong specular lighting model
  - n - render normals: as line segments in fixed pipeline mode but as colours in programable pipeline i.e. shader mode
  - o - cycle through OSD options
  - p - per (vertex, pixel) lighting
  - s - cycle through shapes (grid, wave)
  - v - toggle VBOs
  - 4 - toggle single/multi view

    - +/- - increase/decrease tesselation
    - w - wireframe (line) or fill mode
    - 'z' - 2D/3D wave
- A set of axes showing the x, y and z directions (which can use fixed pipeline immediate mode).
- Interactive changing of the tesselation, using doubling/halving with a base of 8x8.
- Fixed pipeline lighting consisting of a single directional light source at (0.5, 0.5, 0.5, 0) with default ambient, diffuse and specular values (which can be looked up in the OpenGL reference pages for shader calculations). For the wave's material use default ambient, cyan diffuse (0.0, 0.5, 0.5) and light grey specular (0.8. 0.8, 0.8) with shininess of 20.

Use shaders for geometry generation, lighting calculations and animation:

- Calculate wave vertex coordinates and normals directly in the vertex shader from parametric coordinates instead of performing the calculations on the CPU and storing the values. This means they are recalculated every frame, but on the GPU.
- Perform modelling and viewing transformations by passing in from the CPU program model view and normal matriced to the vertex shader as uniforms. (So don't use gl_ModelViewMatrix and gl_NormalMatrix as they are deprecated, although they still work).
- Per-vertex and per-pixel lighting using Blinn-Phong lighting, and then Phong lighting. Use a single light source as for the fixed pipeline.
- Provide the ability to switch between per-vertex and per-pixel shading as well as fixed pipeline lighting and shading.

Either glut or SDL can be used.

Visuals and Performance - answer five questions (in a readme.txt file, no more than than a paragraph each):

- Is there any visual difference between fixed pipeline and shader lighting? Should there be?
- Is there any performance difference between fixed pipeline and using shader lighting?
- What overhead/slowdown factor is there for performing animation compared to static geometry using the vertex shader? For static i.e. geometry is it worth pre-computing and storing the sine wave values in buffers?
- Is there any difference in performance between per vertex and per pixel shader based lighting?
- What is the main visual difference between Phong and Blinn-Phong lighting?

Marking guide

Subject to (minor) change:

- 25 -- Functionality (implementation approach and quality is taken into account)
    - PA: shader with per-vertex ADS calculations
    - CR: use of VBOs
    - DI: shader based wave calculations
    - HD: per pixel lighting, detailed lighting calculations, e.g. positional versus directional light, shininess control
- 3 -- Code
    - Code well structured, documented, consistent naming, indentation, style, etc
- 5 -- Q & A