GLSL

1 Introduction

Shaders are small programs that allow the user to interrupt the fixed functionality of the graphics pipeline in hardware. This functionality can be interrupted at the vertex-processing stage (world-to-eye transformations, lighting, textures) and/or at the fragment-processing stage (per-pixel operations). These vertex and fragment programs are targeted at specialized hardware and must therefore be compiled to machine code that the hardware (graphics card) can understand. The programming itself can be done in (card-vendor-specific) assembly language or in higher-level languages called shading languages, most of which adopt a C-like syntax.

This lab will focus on the use of GLSL (the OpenGL Shading Language) to perform vertex and fragment calculations. Your task is two-fold and you can perform the sub-tasks in any order:

1. Create a file with .vp as the extension containing a vertex program that applies some time-varying distortion to the teapot.

2. Create a file with .fp as the extension containing a fragment program that gives the teapot a procedurally-generated brick texture.

At the end of this lab you must be able to answer the following questions:

1. What are the advantages and disadvantages of shading languages?
2. What is GLSL? Do you know of any other shading languages? Why prefer GLSL? Why would you prefer others to GLSL?
3. How do you go about compiling a GLSL program and sending it to your graphics card?
4. How and where is this happening in your PA2 code?
5. What is a vertex processor? What is a fragment processor? How many of these can you typically find on state-of-the-art graphics cards?
6. What else is a modern graphics card good for besides graphics? (not a dumb question)
7. What is GLEW? Why could you possible be interested in it?
8. Why are we interested in your knowledge of the above?

2 Running the lab

The lab executable (lab5) accepts the base name of the shader file as its two command-line arguments. For example, invoking "lab5 vpshader fpshader" will expect vpshader.vp and fpshader.fp to reside in the working directory. At program initialization, these files are read and compiled by your running application into GPU-executable shader program objects. The files shader.h and shader.cpp (from your PA2) perform these tasks. You are encouraged to go through them to see how they trap errors and notify the user of failed compilation and linkage.

You can mix and match compatible vertex and fragment shaders by providing different names at the command line.

Pressing the 's' key toggles between using your shader programs and using the OpenGL fixed-function

pipeline. Initially, the program uses fixed-function.

You may write many combinations of vertex and fragment programs and test them with this lab implementation.

3 The Vertex Program

3.1 Minimal vertex program:

First, write a minimal vertex shader program to implement the OpenGL fixed-function . You can do this using gl_Vertex and gl_ModelViewProjectionMatrix to set gl_Position.[1]

You will need a minimal fragment shader so that your pixels are assigned some color. Create one that assigns all pixels the same color (choose that color).

3.2 Vertex morphing:

Now, write a slightly advanced vertex shader program to apply a time-varying distortion to the vertices in model coordinates:

1. Create a variable named period in your vertex program and set it to whatever time period you want your periodic disturbance to work in (e.g.: 3.0 for 3 seconds).
2. Create another variable named omega and set that to 2π/period.
3. Create a uniform variable named time in your vertex program. This will be set periodically by the application each time the screen is refreshed.
4. Scale the *x*- and *y*- model coordinates by 1.0 + 0.5*sin(omega*time) and 1.0+0.5*cos(omega*time) respectively. Then change them to clip coordinates and assign them to the relevant vertex shader output variable.
5. In main.cpp, create a global instance of the Timer class (see timer.h). This makes system calls to determine the time. Examine its methods to see how you can determine the elapsed time.
6. We would like the screen to be periodically refreshed automatically so that we see the animation of the distortion. Implement a GLUT callback that refreshes the screen when the program is idling. Let's call this function updateScreen().
7. In updateScreen(),
   - determine the location of the time uniform variable in the vertex program, and
   - set its value to the elapsed time as read from the Timer class methods.

Note that this is a procedural technique to morph vertices. Typical morphing techniques store mesh-keyframes and interpolate between them. There will be more on keyframe techniques when we talk about animation in class. There we will use spline curves to interpolate and approximate.

1 The built-in function ftransform() has been deprecated as of GLSL v1.3.

4 The Fragment Program

4.1 Minimal Fragment Shader:

Assign all pixels a fixed color. Choose this color. You may have already done this with the minimal vertex shader.

4.2 Procedural Brick Texture Shader:

1. Determine the GLSL input variable that gives you the pixel coordinates.
2. Find the remainder when the *x*-coord is divided by 20. Call this xmod.
3. Find the remainder when the *y*-coord is divided by 10. Call this ymod.
4. If xmod is between 5 and 15 and ymod is between 2 and 8, color the fragment red otherwise white.

4.3 GLSL Lighting Shader:

Note that GLSL provides no built-in function that implements the OpenGL fixed-function lighting. Implement the fixed-function lighting functionality. You will find the GLSL quick-reference document very helpful.

1. Implement Gouraud shading by computing per-vertex colors in the vertex shader and applying the OpenGL lighting math that we discussed in class. Apply interpolated colors in the fragment shader. With this your color should exactly match what the OpenGL fixed-function pipeline shows.
2. Interpolate the normal in the vertex shader so that you can implement Blinn-Phong shading. Perform lighting calculations per-fragment with the interpolated normals.
3. By extending the keyboard callback you can cycle between OpenGL fixed-function shading and your custom Gouraud and Blinn-Phong shading so that you can compare them and see the results first-hand.

5 References

- GLSL Appendix of your Red Book
- GLSL on Wikipedia - http://en.wikipedia.org/wiki/GLSL
- GLSL Specification - http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf
- Florian Rudolf, GLSL – an introduction
    - URL: http://nehe.gamedev.net/data/articles/article.asp?article=21
- LightHouse3D.com GLSL Tutorial
    - URL: http://www.lighthouse3d.com/opengl/glsl/index.php?intro
- GLSL Quick Reference
    - URL: http://www.opengl.org/sdk/libs/OpenSceneGraph/glsl_quickref.pdf