GLSL Programming/Introduction

About GLSL

GLSL (OpenGL Shading Language) is one of several commonly used shading languages for real-time rendering (other examples are Cg and HLSL). These shading languages are used to program shaders (i.e. more or less small programs) that are executed on a GPU (graphics processing unit), i.e. the processor of the graphics system of a computer – as opposed to the CPU (central processing unit) of a computer.

GPUs are massively parallel processors, which are extremely powerful. Most of today's real-time graphics in games and other interactive graphical applications would not be possible without GPUs. However, to take full advantage of the performance of GPUs, it is necessary to program them directly. This means that small programs (i.e. shaders) have to be written that can be executed by GPUs. The programming languages to write these shaders are shading languages. GLSL is one of them. In fact, it is the shading language of several 3D graphics APIs (application programming interfaces), namely OpenGL, OpenGL ES 2.x, and WebGL. Therefore, GLSL is commonly used in applications for desktop computers, mobile devices, and the web.

About Learning GLSL

Learning GLSL is beneficial for several reasons:

- It provides insight into modern real-time graphics, i.e. in the way GPUs work. Most high-performance real-time graphics applications (such as games) rely in some way or another on computations on GPUs, i.e. on shader programs.
- It enables you to program GLSL shaders (and thus highly efficient graphics applications) and understand other shading languages (such as Cg and HLSL) since the differences are not too big.
- It also enables you to better understand and use high-level graphics development tools since they are usually also based on shaders.
- It might help you to find a job as skills in shader programming are an advantage for game programmers, graphics developers, technical artists, etc.

GLSL is not a particularly complicated programming language and typical programs (i.e. shaders) written in GLSL are rather small. Nonetheless, learning GLSL can be challenging for several reasons:

- GLSL is only part of other graphics APIs (such as OpenGL, OpenGL ES 2.x and WebGL); thus, GLSL is often learned together with a rather large graphics API, which is often necessary to set up the GLSL code and the required graphics data (e.g. meshes and images) for a graphics application.
- GLSL programming requires some knowledge about the programmable graphics pipeline that GPUs implement.
- Most GLSL shaders require matrix and vector operations, which require some understanding of vector and matrix arithmetics.
- GLSL can be hard to debug. Most often, graphics will just simply not be drawn, without any errors or crashes. Problems are best solved with solid understanding of GLSL fundamentals, as opposed to trying many small code changes or copying code snippets, as is common in other forms of programming.

Thus, if you like graphics but hate programming and mathematics, GLSL can be quite challenging.

About this Wikibook

This wikibook was started with students in mind, who like neither programming nor mathematics. The basic motivation for this book is the observation that students are much more motivated to learn programming environments, programming languages and APIs if they are working on specific projects. Such projects are usually developed on specific platforms and therefore the approach of this book is to present GLSL within specific platforms or frameworks, such as the game engine Unity, the game engine of Blender, OpenGL with GLUT, HTML5, the PowerVR SDK, the Android Java SDK, the iOS SDK, etc. Almost none of these platforms supports the latest version of GLSL and all of them have their specific traps and pitfalls.

Thus, instead of presenting the latest version of GLSL (which is likely to be quite irrelevant for learners who are actually trying to program in GLSL), this book tries to present specific versions of GLSL that are used on specific platforms. There is, of course, a large overlap between these versions of GLSL, but the differences are often subtle and can cause many problems. In fact, it is not a good idea to require a student to figure out how to adapt a textbook example of a GLSL shader to the requirements of a certain platform. Instead, specific examples are presented for each platform.

The parts of this wikibook about GLSL in Unity, Blender, OpenGL with GLUT, HTML5, etc. are independent of each other. However, there is a certain progression:

- Unity requires hardly any set-up code; importing meshes and textures is usually a matter of drag&drop and some mouse clicks.
- Blender requires some set-up code in Python; meshes and textures can be imported in Blender.
- HTML5 (WebGL) requires quite a lot of set-up code but runs in any modern web browser without the need to compile and link an application. (I haven't looked into importing meshes and images in WebGL.)
- OpenGL with GLUT requires libraries (OpenGL, GLUT, and usually GLEW or something similar to access all extensions), a compiler and a linker, and usually further libraries for importing images and meshes. And, of course, it requires quite a lot of set-up code.
- GLSL programming on mobile platforms is somewhat specific to either the operating system or the framework (e.g. the PowerVR SDK).

Thus, even if you do not plan to use GLSL within Unity nor within Blender, it might be a good idea to start programming GLSL in Unity or in Blender to quickly learn the basics without worrying about the annoying details of set-up code and import of meshes and images.

Each part of this wikibook is organized as a sequence of tutorials with working examples that produce certain effects. These examples determine which parts of the syntax of GLSL is discussed. A comprehensive description of the syntax of GLSL is provided in the specifications by the Khronos group: GLSL for OpenGL (http://www.opengl.org/documentation/glsl/) and GLSL for OpenGL ES 2.x and WebGL (http://www.khronos.org/registry/gles/).

It is tempting to present the OpenGL graphics pipeline stage by stage; however, this stage-by-stage approach tends to make it more difficult to understand how all the parts of the pipeline work together in various combinations to produce specific effects. Therefore, the discussion of each example also determines which parts of the graphics pipeline are discussed. Nonetheless, there are platform-independent sections to provide an overview of the graphics pipeline, an overview of the traditional matrix transformations, an overview of GLSL-specific vector and matrix operations, a discussion of how to apply matrix transformations to points, vectors, and normals, and some other general topics.

What this Wikibook is not about

Obviously, this wikibook is not about Cg, HLSL, or any other shading language than GLSL. (See the wikibook Cg Programming for an introduction to Cg, which is very similar to HLSL.) Also, this wikibook is not about GLSL for Cg and HLSL programmers. It's for everyone (including Cg and HLSL programmers). Furthermore, this wikibook is not about writing efficient GLSL shaders. It's about enabling the reader to write working shaders. Optimizations add a whole layer of additional tasks (introducing appropriate approximations, choosing appropriate precision modifiers, deciding whether to use look-up tables, avoiding branching by clever arithmetic expressions, etc.), which are beyond the scope of an introduction to GLSL. Of course, real life is different; however, you don't learn to drive by participating in a Formula One race.

About Contributions to this Book

Contributions by everyone are of course welcome. When adding a new example, it should be made sure that no copyrights are violated and that all concepts are either explained in the example itself or in previous examples of the same part.

Unless stated otherwise, all example source code in this wikibook is granted to the public domain. Therefore, you may modify it and relicense it under any license you please. Source code should be embedded in `<syntaxhighlight lang="..."> ... </syntaxhighlight>` in order to use syntax highlighting.

Other wikipages of the same part should be referenced by templates such as {{GLSL Programming Unity SectionRef}}, which should take the name of the wikipage, say X, as first parameter. The output should be something like `Section "X"`. In the print version, however, it should include a section number.

When referencing a figure either try to use a reference that works for any placement of the figure (e.g. “the corresponding figure”) or use templates to generate different references for the online version and the printed book (e.g. `the figure {{Hide in print|to the left}}{{Only in print|below}}`). Ignore problems with PDF versions since the online version is by far the most important one and many readers will have high expectations of the book version.

In order to make it easier to edit diagrams, the use of tables is encouraged. For example, use this:

```
{|cellpadding="5px" cellspacing="0" style="text-align:center; vertical-align:center;"
|-
|colspan="3" style="width:30%; background:#e0e0d0;"|'''first stage or input data'''
|style="width:1%; font-size:200%;" |←
|style="width:69%; background:#e0e0d0;"|additional input (optional, 1st style)
|-
|style="width:14%;"|
|style="width:1%; text-align:right; padding:5px 0 5px 0; font-size:200%;" |↓
|colspan="3" style="width:85%; text-align:left;"|communicated data or operation
|-
|colspan="3" style="width:30%; background:#e0e0d0;"|'''second stage or processed data'''
|style="width:1%;font-size:200%;" |←
|style="width:69%; text-align:left;"|additional input (optional, 2nd style)
|-
|style="width:14%;"|
|style="width:1%; text-align:right; padding:5px 0 5px 0; font-size:200%;" |↓
|colspan="3" style="width:85%; text-align:left; "|communicated data or operation
|-
|colspan="3" style="width:30%; background:#e0e0d0;"|'''third stage or processed data'''
|style="width:1%;font-size:200%;" |
|style="width:69%; text-align:left;"|description of stage or data (optional)
|-
|style="width:14%;"|
|style="width:1%; text-align:right; padding:5px 0 5px 0; font-size:200%;" |↓
|colspan="3" style="width:85%; text-align:left; "|communicated data or operation
|-
|colspan="3" style="width:30%; background:#e0e0d0;"|'''fourth stage or processed data'''
```

```
|style="width:1%;font-size:200%; " |→
|style="width:69%; background:#e0e0d0; "|additional output (optional, 1st style)
|-
|style="width:14%;"|
|style="width:1%; text-align:right; padding:5px 0 5px 0; font-size:200%;" |↓
|colspan="3" style="width:85%; text-align:left; "|communicated data or operation
|-
|colspan="3" style="width:30%; background:#e0e0d0;"|'''final stage or output data'''
|style="width:1%; font-size:200%;" |→
|style="width:69%; text-align:left; "|additional output (optional, 2nd style)
|}
```

To get this:

first stage or input data	←	additional input (optional, 1st style)
↓ communicated data or operation		
second stage or processed data	←	additional input (optional, 2nd style)
↓ communicated data or operation		
third stage or processed data		description of stage or data (optional)
↓ communicated data or operation		
fourth stage or processed data	→	additional output (optional, 1st style)
↓ communicated data or operation		
final stage or output data	→	additional output (optional, 2nd style)

< GLSL Programming