A Simple OpenGL Shader Example

eryar@163.com

Abstract. OpenGL Shading Language, the high-level programming language defined to allow application writers to write programs that execute on the programmable processors defined within OpenGL. Informally the language is sometimes referred to as GLSL. The GLSL has been made part of the OpenGL standard as of OpenGL2.0. The paper focus on a simple example of OpenGL Shader, which can be used as a guide of GLSL.

Key Words. OpenGL, OpenGL Shading Language, GLSL, Shader, Qt

1. Introduction

很早之前，从网上下载到这么一本书《OpenGL Shading Language》，翻看了几遍，终不得要领。后来看到一本由仇德元编写的《GPGPU 编程技术—从 GLSL、CUDA 到 OpenCL》，对 GPU 的了新的认识，对其性能刮目相看。书中给出的一个简单例子，也便于对 Shader 的入门。

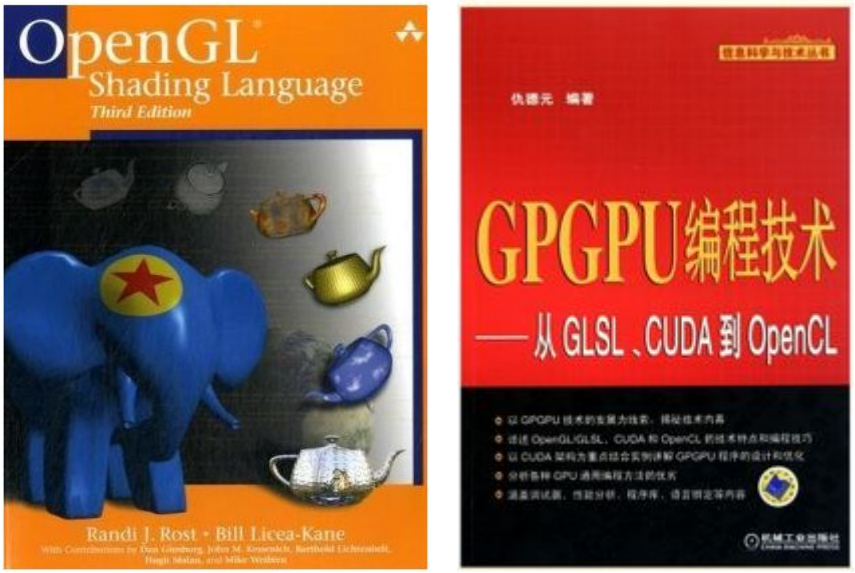

Figure 1.1 OpenGL Shading Language and GPGPU

从 OpenGL Shading Language 的出现可以发现程序员的一个特点，那就是不喜欢一成不变的东西，希望自己有更多的控制权，有个性，向往自由。如果要有个性，那就要引入这个新的东西 GLSL 了，增加了学习成本。不过从 OpenGL2.0 之后，Shader 已经成了 OpenGL 标准的一部分，新版的 OpenGL 的书籍中都少不了 shader 的内容。Shader 成了真实感图形、高性能计算中不可或缺的技术，学习掌握新的工具是为了生活更轻松。

为了让例子简单，本文在 Qt 中实现了一个简单的 Shader 示例，通过这个例子入门后，再结合《OpenGL Shading Language》去实现更炫的效果。再者就是去理解 OpenCASCADE 中的 shader 应用。

2. OpenGL operation pipeline

原来有个问题一直困扰着我，那就是三维的模型怎么在二维的计算机屏幕上显示的。现在明白了这个就其实是显卡的主要工作，并按一定的流水线来实现的。图形流水线（pipeline）是GPU 工作的通用模型，它以某种形式表示的三维场景为输入，输出二维的光栅图像（raster images），也就是位图。这样三维的模型就可以在二维的屏幕上进行显示了。下面依次解释流水线中的关键步骤：

- 图形流水线的起点是一个三维模型。这个三维模型可以是用软件设计出的三维游戏人物，也可以是逆向工程（Reverse Engineering）中用激光扫描仪（Laser Scanner）设备采集的顶点，也可以是几何造型内核（如 OpenCASCADE）将模型网格化生成的顶点等。不论何种模型，在计算机处理之前都一定要经过采样而得到有限的离散顶点，每个顶点都可以被一个向量描述为一个三维坐标系里的点。这些可用来描述三维顶点组成了点云（Point Cloud）。如果采样频率足够高，得到的顶点就可以细致地描述模型的表面。点云中的点可以由一个列表表示，列表中每一项是某点的三维坐标值。同时，列表中每一点都带有该点的颜色信息。这个顶点列表即是流水线的输入数据，从起点进入流水线。
- 顶点可以用来形成多边形，从而拟合出近似的表面。由顶点形成多边形最常用的一种方法是三角化（triangulation），即每相邻的三个点组成一个三角形。接下来每个顶点要经过一系列的逐顶点操作（per-vertex operation），比如计算每个顶点的光照、坐标变换等。
- 由于显示输出的需要，用户会定义一个视口（viewport），即观察模型的位置和角度。然后，模型被投影到与视口观察方向垂直的平面上。这个投影变换也是硬件加速的。根据视口的大小，投影结果有可能被裁剪（clipping）掉一部分。
- 接受模型投影的平面是一个帧缓存（frame buffer），它是一个由像素（pixels）定义的光栅化平面。光栅化（rasterization）的过程，实际上就是决定帧缓存上的哪些像素该取怎么样的值。通过采样和插值，光栅化器（rasterizer）会决定一幅最接近原投影图像的位图。
- 这些像素或者由像素连成的片段还须经历一些逐片段操作（per-fragment operation），也就是说它们的颜色也可以根据算法改变。另外纹理映射（texturing or texture mapping）在这一阶段也会覆盖某些像素的值。对于投影和光栅化的结果，还要判断片段的可见性，也就是遮挡探测（occlusion detection）。
- 最后帧缓存里面的结果被刷新到显示器上。该过程以较高的频率重复，因为人的视频延迟，感觉到的就是连续的。

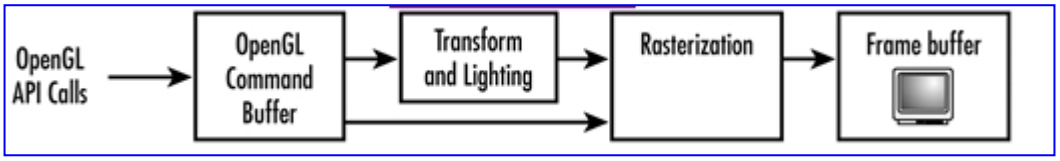

Figure 2.1 A simplified version of the OpenGL pipeline

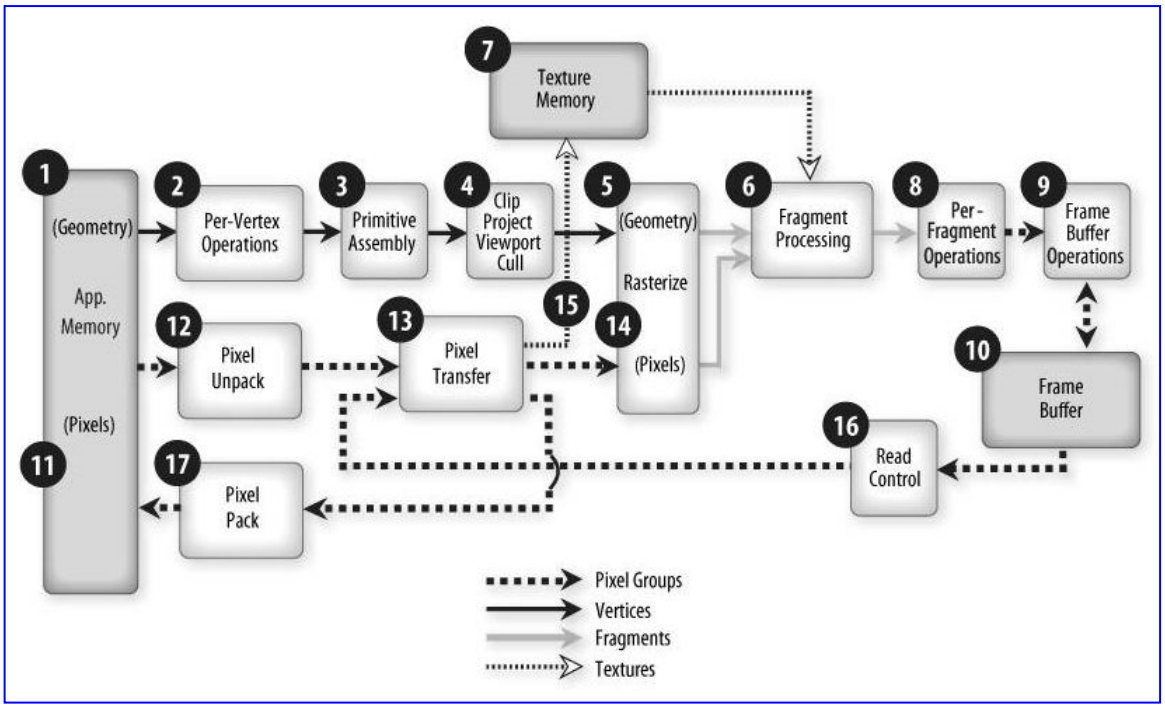

Figure 2.2 Overview of OpenGL operation

可以将上述 OpenGL 的渲染管线想成有两个机器来完成主要的工作：一个机器处理顶点；一个处理片段。对于早期的 OpenGL 而言，只是两个机器是内置的，程序员不能改变他们的工作方式。

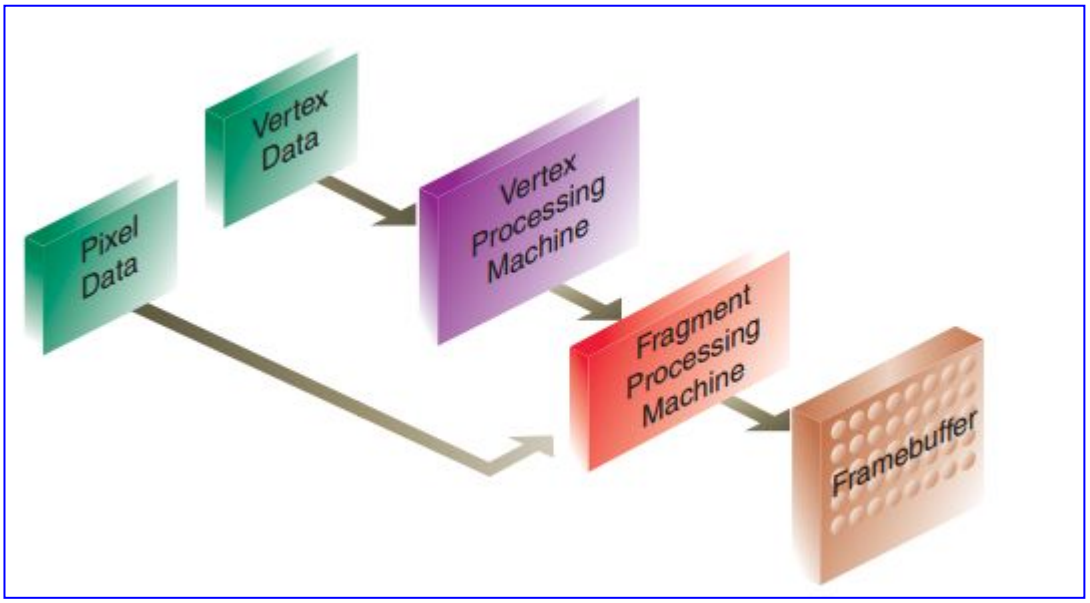

Figure 2.3 The OpenGL Fixed-Function Pipeline[3]

然而可编程的 Shader（programmable shader）则是可以对这两个机器的工作进行干预。

3. **Using GLSL Shaders**

当你想在程序中使用一个顶点 Shader 或片段 Shader 时，需要按如下步骤进行：

- 创建一个 Shader 对象；
- 将 Shader 的源文件编译到这个对象；
- 验证源文件是否编译成功；

然后将这些 shader 链接到一个 Shader 程序：

- 创建一个 Shader 程序；
- 将创建的 Shader 对象绑定到这个 Shader 程序；
- 链接 Shader 程序；
- 验证链接是否成功；
- 将 shader 对象应用到顶点及片段的处理；

如下图所示：

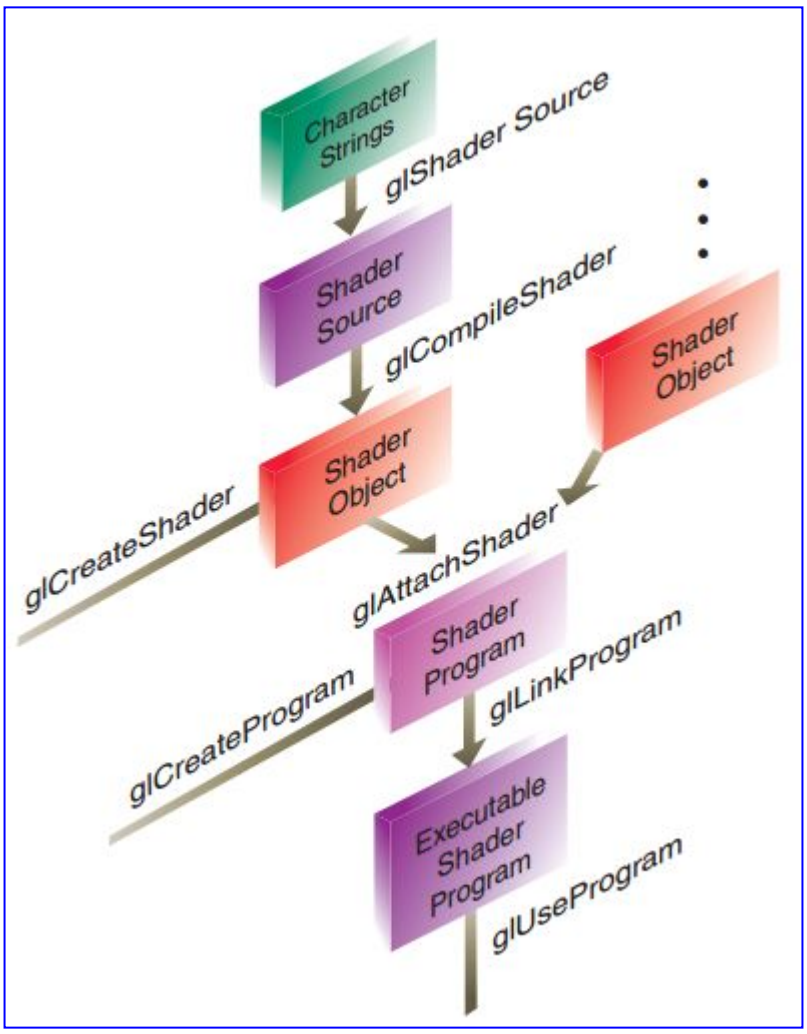

Figure 3.1 Shader Creation Flowchart

4. **The simplest Shader Example**

本来想用 glut 来写个简单的例子的，但是识别不了 glCreateShader 这种函数，发现要下载一些第三方库才可以，如 glew。看到 Qt 中已经有封装好的 QGLShader 和 QGLShaderProgram，所以还是决定用 Qt 来写个简单的例子，从而来对 OpenGL 的 shader 有个感性的认识。其中关于 Shader 部分的主要是这个函数：

```
void ShaderWidget::setShader()
{
    if (!isValid())
    {
        return;
    }

    const QGLContext* aGlContext = context();

    QGLShaderProgram* aShaderProgram = new QGLShaderProgram(aGlContext);

    aShaderProgram->addShaderFromSourceFile(QGLShader::Vertex, "vertex.vert");
    aShaderProgram->addShaderFromSourceFile(QGLShader::Fragment, "fragment.frag");

    aShaderProgram->link();
    aShaderProgram->bind();
    QString aLog = aShaderProgram->log();
}
```

先看一下没有使用 Shader 之前的程序的效果图：

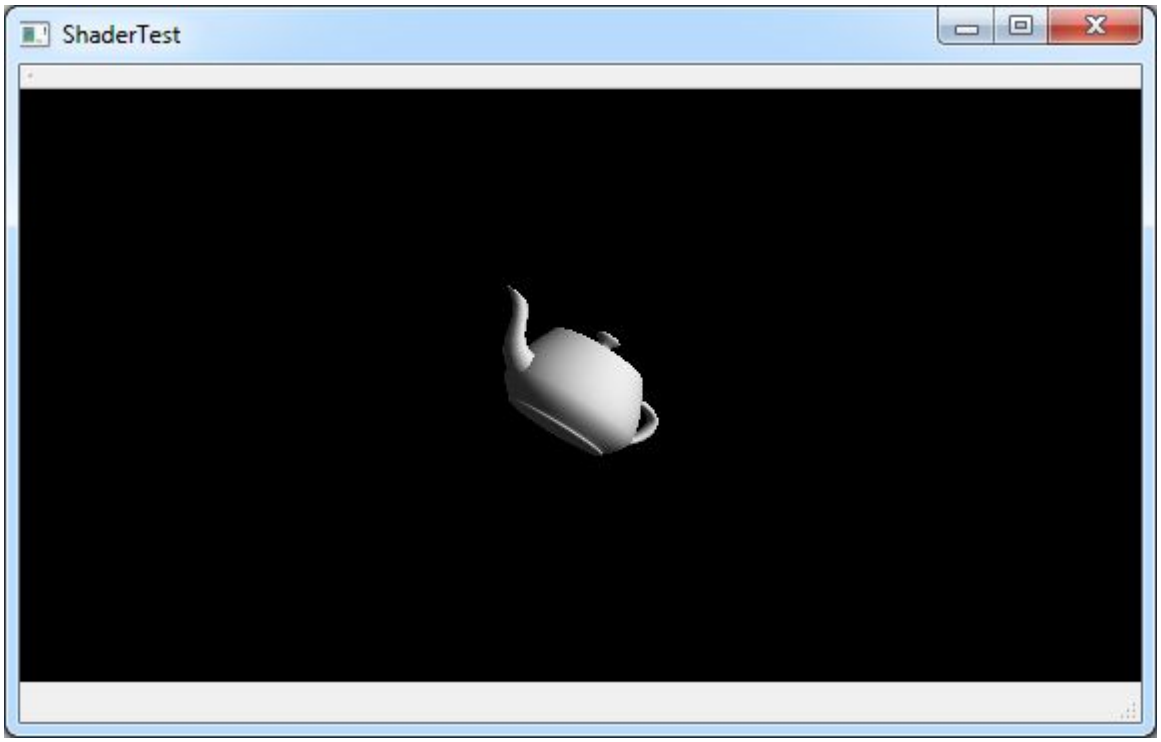

Figure 4.1 A teapot model without shader

其中添加两个 shader，一个是 vertex shader: vertex.vert，一个是 fragment shader:fragment.frag。在 vertex shader 中对顶点的坐标进行变换，代码如下所示：

```
void main()
{
    vec4 a = gl_ModelViewProjectionMatrix * gl_Vertex;
    gl_Position.x = 0.4 * a.x;
    gl_Position.y = 0.1 * a.y;
}
```

OpenGL 内置变量 gl_Position 保存了当前顶点的位置信息，上面的顶点着色器修改了每个顶点的 X 坐标和 Y 坐标，使得输出了一个扭曲的 teapot。

片段着色器当前片段的颜色改成紫色，片段着色器代码如下：

```
void main()
{
    gl_FragColor = vec4(0.627, 0.125, 0.941, 1.0);
}
```

为了验证程序是使用了着色器，运行程序得到如下图所示：

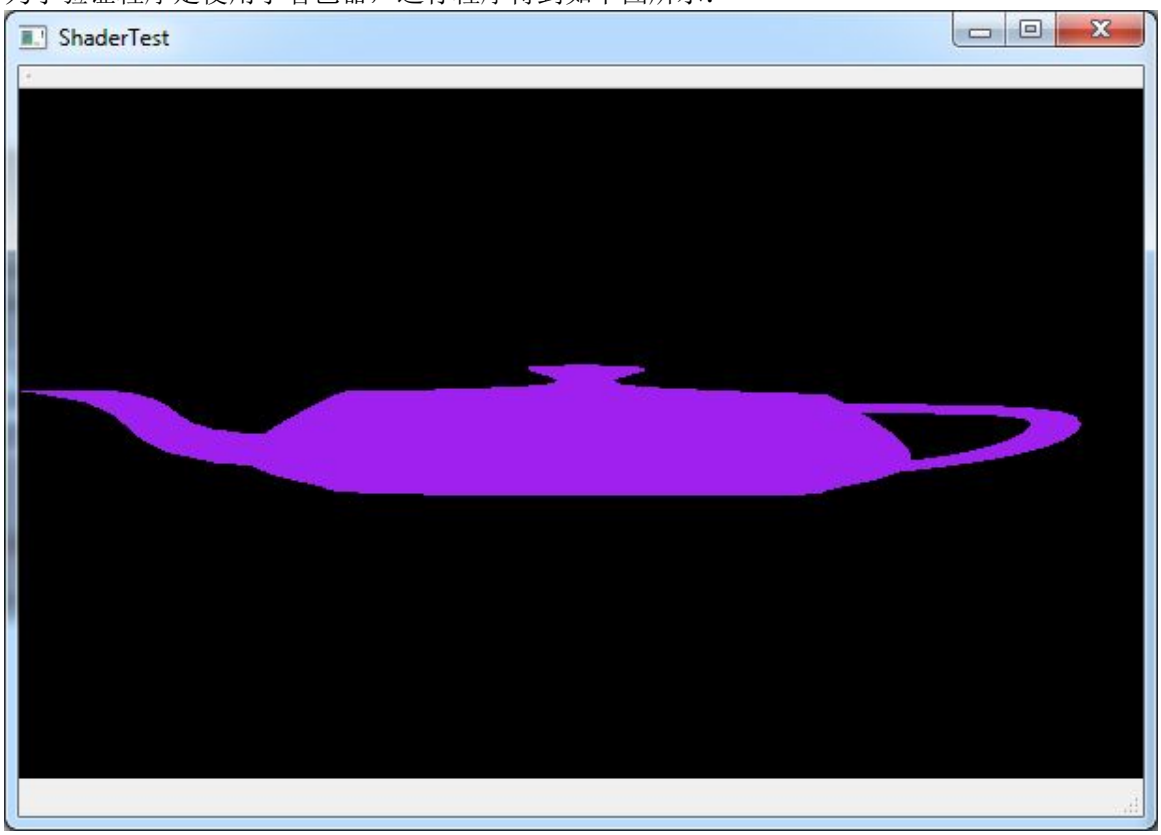

Figure 4.2 A Teapot with shader

在不重新编译程序的情况下，只修改片段着色器中的代码，改成如下所示内容：

```
void main()
{
    gl_FragColor = vec4(0.627, 0.125, 0.0, 1.0);
}
```

保存片段着色器代码后直接运行程序，可得到如下图所示的红色 teapot：

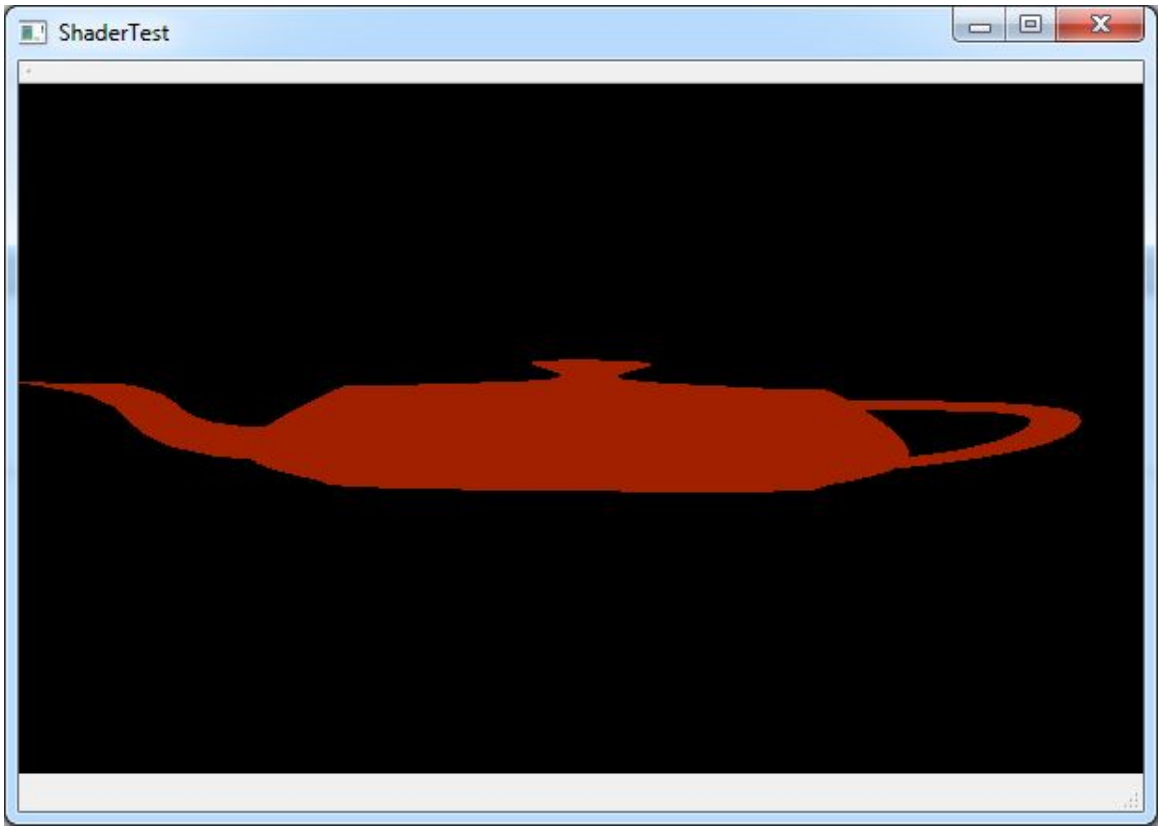

Figure 4.3 Change Fragment Shader

可以看到 shader 的确是起了作用。本文最后将给出程序的完整代码及 shader 的代码，便于测试。

5. Conclusion

OpenGL 的 Shader 给了程序员对 OpenGL 的更多的控制权，可对其顶点处理和片段处理进行更个性化的配置以达到炫酷的效果。

Shader 的使用步骤是先创建 shader 对象，再将源码编译到 shader 对象。最后通过 shader 程序，将 shader 添加并编译、链接和使用。

最后在 Qt 中以一个简单的例子来验证了 shader 的效果，入门之后便于理解 GLSL 更详细的功能，以使自己的可视化程序具有更高的性能，更酷的效果。

6. References

1. 仇德元. GPGPU 编程技术—从 GLSL、CUDA 到 OpenCL. 机械工业出版社. 2012
2. Randi J. Rost. OpenGL Shading Language. Addison Wesley. 2006
3. Dave Shreiner. OpenGL Programming Guide(7th). Addison-Wesley. 2009