GLSL Programming/Unity/Minimal Shader

This tutorial covers the basic steps to create a minimal GLSL shader in Unity.

Starting Unity and Creating a New Project

After downloading and starting Unity (Windows users have to use the command-line argument `-force-opengl`), you might see an empty project. If not, you should create a new project by choosing **File** > **New Project...** from the menu. For this tutorial, you don't need to import any packages but some of the more advanced tutorials require the scripts and skyboxes packages. After creating a new project on Windows, Unity might start without OpenGL support; thus, Windows users should always quit Unity and restart it (with the command-line argument `-force-opengl`) after creating a new project. Then you can open the new project with **File** > **Open Project...** from the menu.

If you are not familiar with Unity's Scene View, Hierarchy View, Project View and Inspector View, now would be a good time to read the first two (or three) sections ("Unity Basics" and "Building Scenes") of the Unity User Guide (http://unity3d.com/support/documentation/Manual/User%20Guide.html).

Creating a Shader

Creating a GLSL shader is not complicated: In the **Project View**, click on **Create** and choose **Shader**. A new file named "NewShader" should appear in the Project View. Double-click it to open it (or right-click and choose **Open**). An editor with the default shader in Cg should appear. Delete all the text and copy & paste the following shader into this file:

```
Shader "GLSL basic shader" { // defines the name of the shader
   SubShader { // Unity chooses the subshader that fits the GPU best
      Pass { // some shaders require multiple passes
         GLSLPROGRAM // here begins the part in Unity's GLSL

         #ifdef VERTEX // here begins the vertex shader

         void main() // all vertex shaders define a main() function
         {
            gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
               // this line transforms the predefined attribute
               // gl_Vertex of type vec4 with the predefined
               // uniform gl_ModelViewProjectionMatrix of type mat4
               // and stores the result in the predefined output
               // variable gl_Position of type vec4.
         }

         #endif // here ends the definition of the vertex shader


         #ifdef FRAGMENT // here begins the fragment shader

         void main() // all fragment shaders define a main() function
         {
            gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
               // this fragment shader just sets the output color
               // to opaque red (red = 1.0, green = 0.0, blue = 0.0,
               // alpha = 1.0)
         }

         #endif // here ends the definition of the fragment shader

         ENDGLSL // here ends the part in GLSL
```

```
        }
    }
}
```

Save the shader (by clicking the save icon or choosing **File** > **Save** from the editor's menu).

Congratulations, you have just created a shader in Unity. If you want, you can rename the shader file in the Project View by clicking the name, typing a new name, and pressing Return. (After renaming, reopen the shader in the editor to make sure that you are editing the correct file.)

Unfortunately, there isn't anything to see until the shader is attached to a material.

Creating a Material and Attaching a Shader

To create a material, go back to Unity and create a new material by clicking **Create** in the **Project View** and choosing **Material**. A new material called “New Material” should appear in the Project View. (You can rename it just like the shader.) If it isn't selected, select it by clicking. Details about the material appear now in the Inspector View. In order to set the shader to this material, you can either

- drag & drop the shader in the **Project View** over the material or
- select the material in the **Project View** and then in the **Inspector View** choose the shader (in this case “GLSL basic shader” as specified in the shader code above) from the drop-down list labeled **Shader**.

In either case, the Preview in the Inspector View of the material should now show a red sphere. If it doesn't and an error message is displayed at the bottom of the Unity window, you should reopen the shader and check in the editor whether the text is the same as given above. Windows users should make sure that OpenGL is supported by restarting Unity with the command-line argument `-force-opengl`.

Interactively Editing Shaders

This would be a good time to play with the shader; in particular, you can easily change the computed fragment color. Try neon green by opening the shader and replacing the fragment shader with this code:

```
        #ifdef FRAGMENT

        void main()
        {
           gl_FragColor = vec4(0.6, 1.0, 0.0, 1.0);
              // red = 0.6, green = 1.0, blue = 0.0, alpha = 1.0
        }

        #endif
```

You have to save the code in the editor and activate the Unity window again to apply the new shader. If you select the material in the Project View, the sphere in the Inspector View should now be green. You could also try to modify the red, green, and blue components to find the warmest orange or the darkest blue. (Actually, there is a movie about finding the warmest orange and another about dark blue that is almost black.)

You could also play with the vertex shader, e.g. try this vertex shader:

```
        #ifdef VERTEX

        void main()
```

```
        {
           gl_Position = gl_ModelViewProjectionMatrix
           * (vec4(1.0, 0.1, 1.0, 1.0) * gl_Vertex);
        }

        #endif
```

This flattens any input geometry by multiplying the y coordinate with 0.1. (This is a component-wise vector product; for more information on vectors and matrices in GLSL see the discussion in Section “Vector and Matrix Operations”.)

In case the shader does not compile, Unity displays an error message at the bottom of the Unity window and displays the material as bright magenta. In order to see all error messages and warnings, you should select the shader in the **Project View** and read the messages in the **Inspector View**, which also include line numbers, which you can display in the text editor by choosing **View > Line Numbers** in the text editor menu. You could also open the Console View by choosing **Window > Console** from the menu, but this will not display all error messages and therefore the crucial error is often not reported.

Attaching a Material to a Game Object

We still have one important step to go: attaching the new material to a triangle mesh. To this end, create a sphere (which is one of the predefined game objects of Unity) by choosing **GameObject > Create Other > Sphere** from the menu. A sphere should appear in the Scene View and the label “Sphere” should appear in the Hierarchy View. (If it doesn't appear in the Scene View, click it in the Hierarchy View, move (without clicking) the mouse over the Scene View and press “f”. The sphere should now appear centered in the Scene View.)

To attach the material to the new sphere, you can:

- drag & drop the material from the **Project View** over the sphere in the **Hierarchy View** or
- drag & drop the material from the **Project View** over the sphere in the **Scene View** or
- select the sphere in the **Hierarchy View**, locate the **Mesh Renderer** component in the **Inspector View** (and open it by clicking the title if it isn't open), open the **Materials** setting of the Mesh Renderer by clicking it. Change the “Default-Diffuse” material to the new material by clicking the dotted circle icon to the right of the material name and choosing the new material from the pop-up window.

In any case, the sphere in the Scene View should now have the same color as the preview in the Inspector View of the material. Changing the shader should (after saving and switching to Unity) change the appearance of the sphere in the Scene View.

Saving Your Work in a Scene

There is one more thing: you should save you work in a “scene” (which often corresponds to a game level). Choose **File > Save Scene** (or **File > Save Scene As...**) and choose a file name in the “Assets” directory of your project. The scene file should then appear in the Project View and will be available the next time you open the project.

One More Note about Terminology

It might be good to clarify the terminology. In GLSL, a “shader” is either a vertex shader or a fragment shader. The combination of both is called a “program”.

Unfortunately, Unity refers to this kind of program as a “shader”, while in Unity a vertex shader is called a “vertex program” and a fragment shader is called a “fragment program”.

To make the confusion perfect, I'm going to use Unity's word “shader” for a GLSL program, i.e. the combination of a vertex and a fragment shader. However, I will use the GLSL terms “vertex shader” and “fragment shader” instead of “vertex program” and “fragment program”.

Summary

Congratulations, you have reached the end of this tutorial. A few of the things you have seen are:

- How to create a shader.
- How to define a GLSL vertex and fragment shader in Unity.
- How to create a material and attach a shader to the material.
- How to manipulate the ouput color `gl_FragColor` in the fragment shader.
- How to transform the input attribute `gl_Vertex` in the vertex shader.
- How to create a game object and attach a material to it.

Actually, this was quite a lot of stuff.

Further Reading

If you still want to know more

- about vertex and fragment shaders in general, you should read the description in Section “OpenGL ES 2.0 Pipeline”.
- about the vertex transformations such as `gl_ModelViewProjectionMatrix`, you should read Section “Vertex Transformations”.
- about handling vectors (e.g. the `vec4` type) and matrices in GLSL, you should read Section “Vector and Matrix Operations”.
- about how to apply vertex transformations such as `gl_ModelViewProjectionMatrix`, you should read Section “Applying Matrix Transformations”.
- about Unity's ShaderLab language for specifying shaders, you should read Unity's ShaderLab reference (http://unity3d.com/support/documentation/Components/SL-Reference.html).

< GLSL Programming/Unity