Computer Graphics Practical 1

Michaelmas Term 2020

Overview

The aim of this practical is to give you a chance to experiment with writing shaders, allowing you to implement the mathematics and algorithms seen during the course and see them in action.

The objective of the practical is to create a 3D scene (using Three.js) and then write fragment and vertex shaders for some of the objects in your scene. Three.js is a common rendering library that uses WebGL. Three.js is written in JavaScript and shaders are written in the GLSL language.

You should aim to complete this practical by the end of the second practical session, but it will be signed off in the final session.

Tasks

Task 1 - Setting up your Three.js scene

You are provided with the skeleton of some code to set up a Three.js scene. You should begin by looking over the code in `main.js`'s `initialise_scene` function and understanding the key parts. The `load_resources` function is less important to understand – it loads resources in for you so you can use images, models and shaders from external files.

Once you have read through the code, try and add some objects to your scene. The easiest way to do this is to consult the online documentation for Three.js, look at some examples, or follow an introductory tutorial.

You should then experiment with your scene by adding some more objects and adding some simple controls.

Task 2 - Writing a shader

Your second task is to write a shader for some of the objects in your scene. You should begin by replacing one of the materials in your scene with a `THREE.ShaderMaterial`. This will enable you to use your own shader definition on the objects that use that material.

You should load the shader in from an external file (using the RESOURCES array at the top of the code). You will need to include both a fragment and a vertex shader.

`default.frag` and `default.vert` are provided in the `shaders` folder to give you a starting point. Note that the fragment shader has a uniform, which you will need to pass during the construction of your `ShaderMaterial`.

Once you have successfully loaded the default shaders in, you should modify them to customise how your objects are rendered. You could implement a simple lighting model, modify the displayed geometry using the vertex shader, or implement any other shaders you can think of.

Further Task - Writing an advanced shader

To continue further, you should extend your shader to display more complex behaviour. For example, you might consider one of the following:

- Implementing Phong shading
- Adding a shading effect such as toon shading
- Making your texture reflective, refractive, or translucent
- Making your shader vary over time, to appear animated
- Adding a texture to your shader
- Adding a mapping technique, such as bump mapping, to your texture

You could also write a post-processing shader (i.e. a shader that modifies the output of the rendering process). To do this, you will need to modify the JavaScript code to add a post-processing step (using `THREE.EffectComposer` and `THREE.ShaderPass`).

Assessment

This practical will be assessed as follows:

- For the S grade, you should set up a 3D scene, consisting of multiple objects with different materials. You should implement some form of controls so the camera can be moved around your scene. You should implement a simple shader, other than the default one provided.
- For the S+ grade, you should meet the requirements of the S grade and extend your shader to more complex behaviour. For examples, see the list provided above.

If you are unsure whether your planned shader will meet the requirements for the S+ grade, please agree it with the demonstrator before beginning.

Notes

There are lots of examples and tutorials available for Three.js to help you get started.

Writing in GLSL can be fiddly; make sure you put semicolons at the end of lines, and be careful about mixing up integers and floats – always add a decimal point when expressing floats (write 5.0 instead of 5, for instance).

Make sure you know the difference between `varying`, `uniform` and `attributes`. `uniforms`, which are defined in the JavaScript code, can be many different types, including entire textures.

Your vertex shader will (often) not do much other than calculating vectors for you to use in your fragment shader. Remember that `varying` variables are interpolated for you when they are passed to the fragment shader.

ThreeJS supports many kinds of inbuilt shaders, such as Phong and texturing; because the point of the exercise is for you to write your own shaders and understand how they work, you should implement your shaders using `ShaderMaterial`.