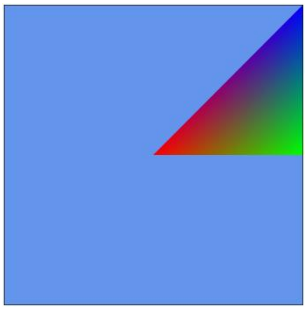
02561 COMPUTER GRAPHICS DTU COMPUTE

Exercise 12: WebGL Introduction

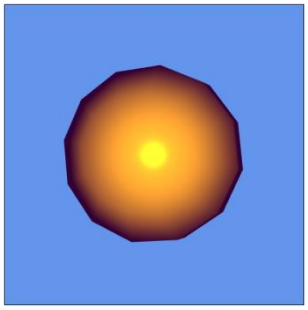
Reading	
Purpose	The purpose of the exercise is to get an overview of how to setup and work with WebGL. WebGL is based on OpenGL ES 2.0, which is a subset of the OpenGL API we have been working with so far. We get access to the OpenGL API through a WebGL context object that gives us the naming convention we are used to but without the gl-prefix. For example in OpenGL we write *glClear(GL_COLOR_BUFFER_BIT)* and in WebGL we write *gl.clear(gl.COLOR_BUFFER_BIT).* We will see how to create a WebGL context, draw triangles, draw models and how to use shaders and textures. A handy WebGL quick reference card can be downloaded here: https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf
Part 1	Ex01 shows the basic setup of a WebGL application. All code is in the HTML document. • Study and understand the code/script
Part 2	To make things a bit easier we will use some helper libraries: - initShaders.js: loads and compile shader code (Angel) - MV.js: very basic math library (Angel) - webgl-utils.js: a collection of handy utility code (Google) Ex02 shows the same basic WebGL setup as in Ex01, but this time we make use of the helper libraries and we separate our JavaScript code from the HTML document to make things a bit cleaner. • Study and understand the code/script

Exercise 12: WebGL Introduction

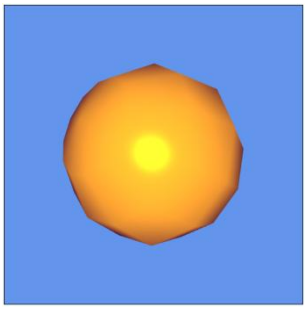
Part 3 	In Ex03 we will actually draw something on the canvas and we will need to write a shader. We will extend the program to include vertex colors. We use the, Angel supplied, method *initShaders* which will load, compile and link vertex and fragment shader code embedded in their respective script tags in the HTML document. a) Study and understand the code/script b) Create a color array with one color for each vertex c) Create and bind a color buffer d) Extend the vertex shader and associate the color buffer to the new vertex shader attribute e) Pass the vertex color to the fragment shader and use the interpolated color as fragment color

Exercise 12: WebGL Introduction

Part 4



Point light



Directional light

In Ex04 we will load a mesh and render it using Phong shading with both a point light and a directional light (just like in the Gouraud/Phong exercise).

In *createVertexBufferObject* we interleave the loaded mesh data into a single buffer to avoid managing and uploading multiple buffers (similar to how we have used a vertex data structure in previous exercises). The actual mesh data is read from a script tag in the HTML document and is embedded using the JSON format (a high-res sphere JSON file is included in the exercise folder if you prefer).

To generate the needed matrices, take a look at the helper methods in the MV.js library. Note: when sending an Angel *mat4* matrix to the GPU using *glUniform*, you must use the method *flatten()* to flatten the matrix into an array. In addition to this, also note that for the *glUniformMatrix* method, the *transpose* parameter must be set to *false* (see the quick reference card for details).

a) In the *render* method
   - Create a perspective projection matrix using the supplied parameters
   - Create a view matrix with the eye in (0,0,3) looking at the origin
   - Create a model matrix as a rotation around the axis (0,1,1) with angle of θ, where θ is animated to make the mesh spin
   - Upload the modelView and projection matrices to the GPU

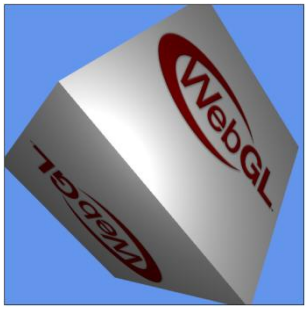
b) Implement Phong shading in the vertex and fragment shaders
   - In the vertex shader, compute the vertex position. Then compute the normal, eye position and light direction and pass them to the fragment shader. In this version of GLSL you use the keyword *varying* instead of *in/out* when passing variables between shader stages
   - In the fragment shader, compute the diffuse and specular reflectance coefficients

c) Use the w component of the LightPosition to toggle between point light and directional light (1 is point light, 0 is directional light). Modify the shader to support this. The key ‘L’ is already setup to toggle the w component.

Tip: *When working with shaders always remember what space your vectors are in and don’t let vectors from different spaces operate on each other!*

Exercise 12: WebGL Introduction

Part 5



In Ex05 we will extend Ex04 to also include a texture. The texture image data is base64 encoded and embedded in a script tag in the HTML document using JSON formatting.

- Study and understand the loadTexture method.

a) In the *render* method, create the projection, view and model matrices and upload the modelView and projection matrices to the GPU as in Ex04.
b) In the *bindAttributes* method, extend the vertex declaration to include texture coordinates (uv).
c) In the *createVertexBufferObject* method modify the mesh data loading process to also load the texture coordinates (look at the data property names in the model data 'boxData' embedded in a script tag in the HTML document).
d) Implement Phong shading as in Ex04 but extend the vertex shader to include texture coordinates (uv attribute) and pass that to fragment shader. In the fragment shader use the texture coordinates to sample the texture and use that as the diffuse color.

Tip: *Be vigilant when you copy and paste from Ex04 as we have added a few extra lines of code in Ex05 to help you along.*