CS112 Programming Assignment 2

The deadline is 11.55 PM, Thursday Feb 15. Please submit the zip file to EEE.

1. Introduction

In this assignment, we will get familiar with lighting which is also fundamental to creating any scene, using WebGL. In this assignment, you would need to write some code (maybe dozens of lines) and it will take some time to understand on how the lighting equations work, so **please start early**.

Software and hardware requirement: WebGL runs within the browser, so is independent of the operating and window systems. You may finish the assignment using any operating system you like, e.g. Windows, OSX or Linux. **Programming language**: The assignment will be implemented in JavaScript. As we will minimize the use of obscure Javascript language features, it should not be too difficult for anyone with experience in a dynamic language like Python or familiar with the principles of Object Oriented Programming (like C++ or Java) to get a handle on Javascript syntax by reading through some of the code in code skeleton. For a more formal introduction to Javascript, checkout the nice tutorial from https://javascript.info/.

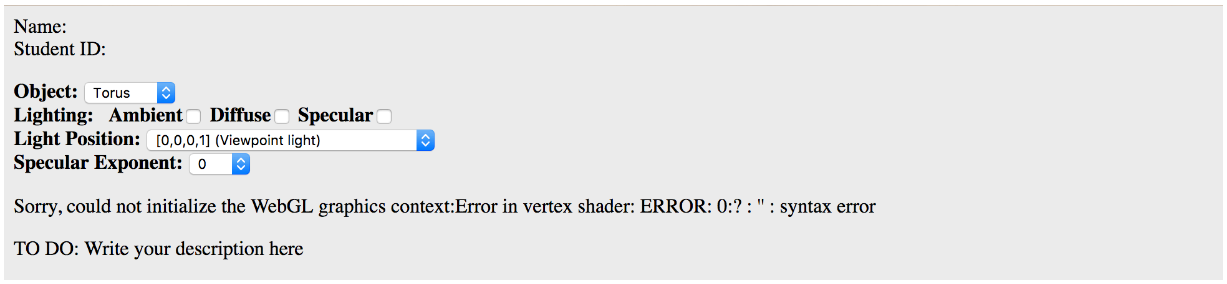
Cooperation and third-party code: This is an individual programming assignment, and you should implement the code **by your own**. You may not share final solutions, and you must write up your own work, expressing it in your own words/notation. Third party codes are not allowed unless with professor's permission.

2. Getting started with the code skeleton.

1) Download pa2.zip and extract it any folder you like. You should have five files in the folder:

a.	pa2.html	: html file which shows the WebGL canvas.
b.	pa2.js	: functions used to draw the scene.
c.	gl-matrix-min.js	: utility functions for operating matrices
d.	models.js	: file describing the models in IFS format.
e.	trackball-rotator.js	: utility functions for rotating the scene using the cursor.

2) Open pa2.html in the extracted folder with Chrome. You will see an error message right now as the vertex and fragment shaders are not implemented. The vertex and fragment shaders are responsible for drawing the scene.



Name:
Student ID:

Object: Torus
Lighting: **Ambient** ☐ **Diffuse** ☐ **Specular** ☐
Light Position: [0,0,0,1] (Viewpoint light)
Specular Exponent: 0

Sorry, could not initialize the WebGL graphics context:Error in vertex shader: ERROR: 0:? : '' : syntax error

TO DO: Write your description here

3) Note, **for this project, you will be modifying two files pa2.html and pa2.js.** But try to read and understand what other functions are doing as well as it would help you better implement things. There are lots of comments throughout the codebase and the functions have similar structure as programming assignment 1 for better/easy understanding.

4) Using the *object tab*, a user can select different models (torus, cylinder or sphere). Using the *lighting checkmarks*, a user can toggle between different lighting conditions. A user can also adjust the *lighting position* and *exponent* used for computing specular highlights.

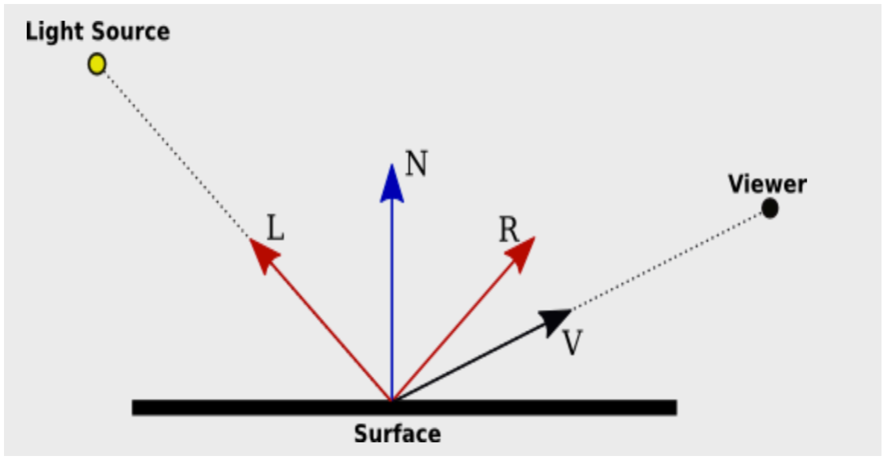
There are two tasks which need to be completed in this programming assignment. They are as follows.

3. Implement the lighting function.

TASK 1: You will be implementing 3 different kind of lighting, namely ambient, diffuse and specular using the Phong shading model. You will complete both vertex and fragment shader in pa2.html in order to draw object.

1. Before starting the project, you need to first know how to compute the light at a single point. Below is a schematic of a point on a surface which is being illuminated by a point light source where,

   L = Light is the direction of the light
   N = Normal of the point on the surface
   R = Reflection of L about N.
   V = viewing direction



Light at a point = Ambient Light + Diffuse Light + Specular Light.

Ambient Light = $K_a * \text{diffuse\_color}$
Diffuse Light = $K_d * \text{diffuse\_color} * \max((N.L), 0)$
Specular Light = $K_s * \text{specular\_color} * \max((R.V), 0)^n$
$R = 2N(N.L) - L$

Use K_a, K_d, K_s 0.1, 0.4 and 0.4 respectively. *n* is the specular coefficient, which the user can change

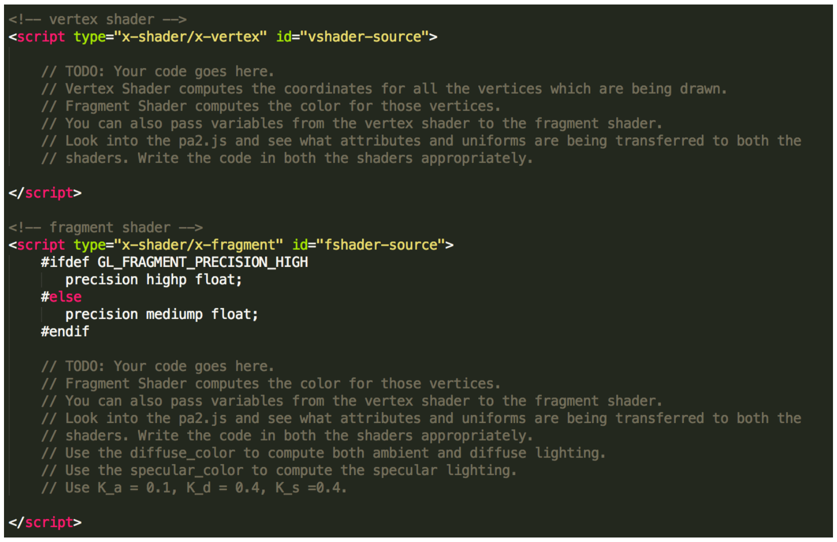
Note, although the above light equation is not complete as it does not account for the distance from light source (light intensity decreases as we move away from the light source), to keep things simple we would only implement above equation in this programming assignment.

2. The next thing you need to know is what are Shaders. Shaders (Vertex, Fragment, etc) are small programs which are executed on the GPU. They are mainly responsible for determining the color of each object. However, they can be used for many different tasks. In this assignment we will be using two shaders namely *vertex* and *fragment* shaders to compute the lighting of our scene.

3. To write in any shader, you must first know the basics of GLSL. GSLS is a short term for OpenGL Shading

Language. This is used to write code into any shader. It is similar to C/C++. Check out the references #1 and #4 to understand more on how to write code in a shader.

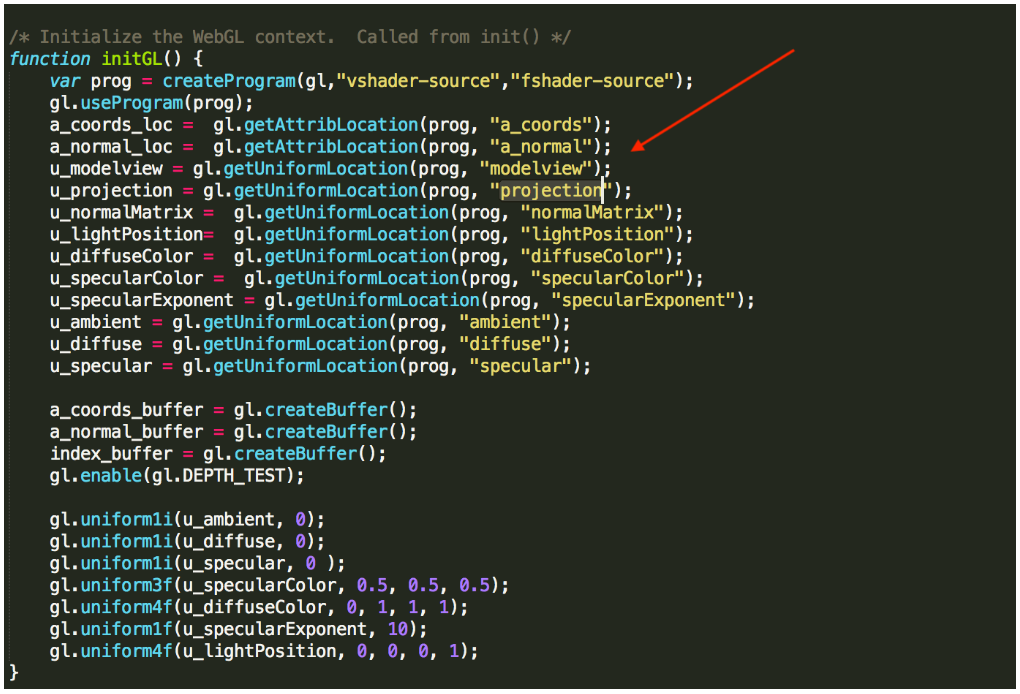
4. Once you are familiar with GSLS, you can start writing code in the vertex and fragment shader in pa2.html.



```
<!-- vertex shader -->
<script type="x-shader/x-vertex" id="vshader-source">

    // TODO: Your code goes here.
    // Vertex Shader computes the coordinates for all the vertices which are being drawn.
    // Fragment Shader computes the color for those vertices.
    // You can also pass variables from the vertex shader to the fragment shader.
    // Look into the pa2.js and see what attributes and uniforms are being transferred to both the
    // shaders. Write the code in both the shaders appropriately.

</script>

<!-- fragment shader -->
<script type="x-shader/x-fragment" id="fshader-source">
    #ifdef GL_FRAGMENT_PRECISION_HIGH
       precision highp float;
    #else
       precision mediump float;
    #endif

    // TODO: Your code goes here.
    // Fragment Shader computes the color for those vertices.
    // You can also pass variables from the vertex shader to the fragment shader.
    // Look into the pa2.js and see what attributes and uniforms are being transferred to both the
    // shaders. Write the code in both the shaders appropriately.
    // Use the diffuse_color to compute both ambient and diffuse lighting.
    // Use the specular_color to compute the specular lighting.
    // Use K_a = 0.1, K_d = 0.4, K_s =0.4.

</script>
```

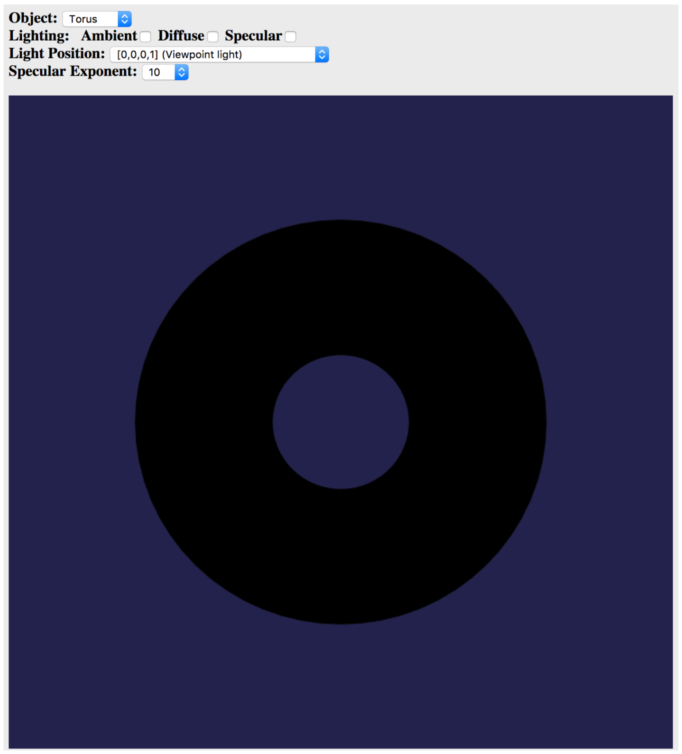
However, before writing, I would recommend checking out the function *initGL()* in pa2.js to see what all variables you will be creating in the shaders. Note, you will be using some of these variables in the vertex shader and some in the fragment shader. Hence, it's important to understand

i. What vertex and fragment shaders do.
ii. What these variables are and what you can compute from them.

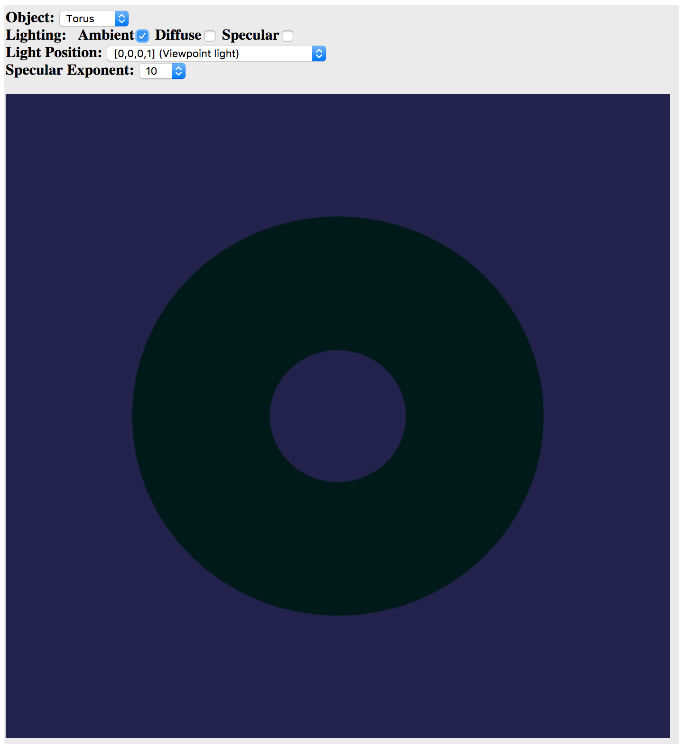


```
/* Initialize the WebGL context.  Called from init() */
function initGL() {
    var prog = createProgram(gl,"vshader-source","fshader-source");
    gl.useProgram(prog);
    a_coords_loc =  gl.getAttribLocation(prog, "a_coords");
    a_normal_loc =  gl.getAttribLocation(prog, "a_normal");
    u_modelview = gl.getUniformLocation(prog, "modelview");
    u_projection = gl.getUniformLocation(prog, "projection");
    u_normalMatrix =  gl.getUniformLocation(prog, "normalMatrix");
    u_lightPosition=  gl.getUniformLocation(prog, "lightPosition");
    u_diffuseColor =  gl.getUniformLocation(prog, "diffuseColor");
    u_specularColor =  gl.getUniformLocation(prog, "specularColor");
    u_specularExponent = gl.getUniformLocation(prog, "specularExponent");
    u_ambient = gl.getUniformLocation(prog, "ambient");
    u_diffuse = gl.getUniformLocation(prog, "diffuse");
    u_specular = gl.getUniformLocation(prog, "specular");

    a_coords_buffer = gl.createBuffer();
    a_normal_buffer = gl.createBuffer();
    index_buffer = gl.createBuffer();
    gl.enable(gl.DEPTH_TEST);

    gl.uniform1i(u_ambient, 0);
    gl.uniform1i(u_diffuse, 0);
    gl.uniform1i(u_specular, 0 );
    gl.uniform3f(u_specularColor, 0.5, 0.5, 0.5);
    gl.uniform4f(u_diffuseColor, 0, 1, 1, 1);
    gl.uniform1f(u_specularExponent, 10);
    gl.uniform4f(u_lightPosition, 0, 0, 0, 1);
}
```

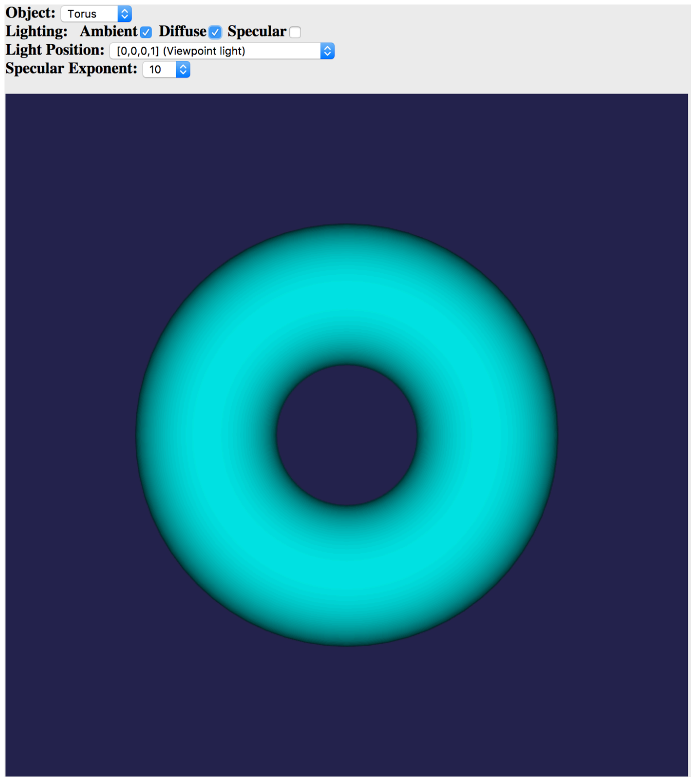
5. You will need to implement each of the lighting separately such that when the user checks ambient or diffuse or specular only those are being implemented. If everything works well, you should be getting results similar to the below images.


Object: Torus
Lighting: Ambient Diffuse Specular
Light Position: [0,0,0,1] (Viewpoint light)
Specular Exponent: 10

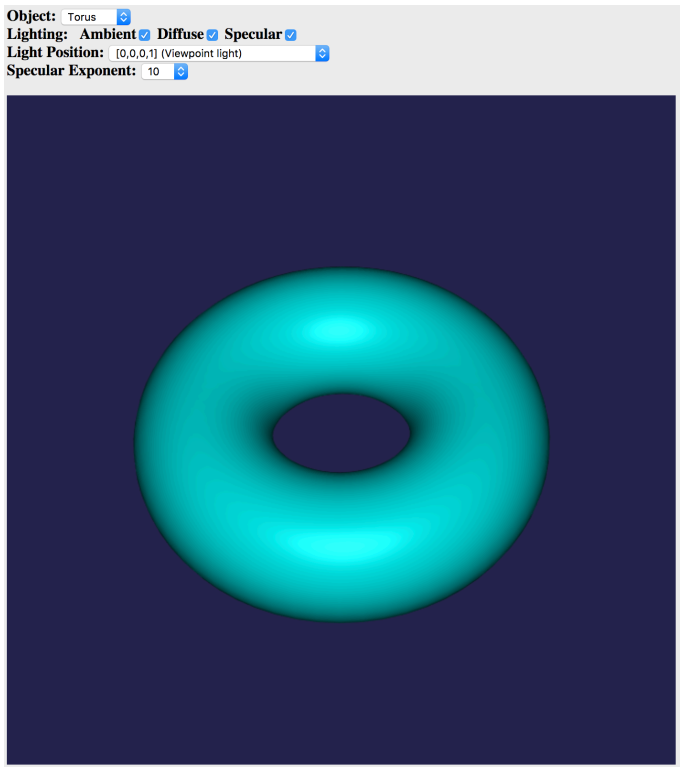
There is no light. Hence you see a black torus


Object: Torus
Lighting: [✓ Ambient] Diffuse Specular
Light Position: [0,0,0,1] (Viewpoint light)
Specular Exponent: 10

Torus with only ambient light. Cyan colored torus is seen.


Object: Torus
Lighting: [✓ Ambient] [✓ Diffuse] Specular
Light Position: [0,0,0,1] (Viewpoint light)
Specular Exponent: 10

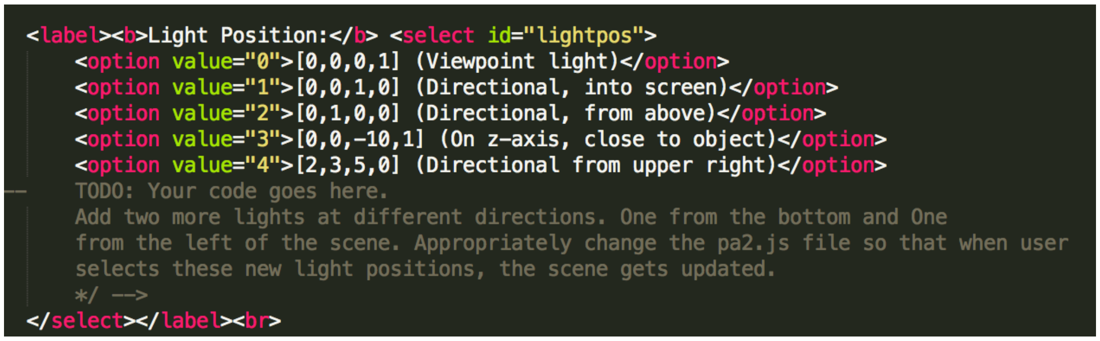
Torus with ambient and diffuse lighting.


Object: Torus
Lighting: [✓ Ambient] [✓ Diffuse] [✓ Specular]
Light Position: [0,0,0,1] (Viewpoint light)
Specular Exponent: 10

Torus with ambient and diffuse and specular lighting.

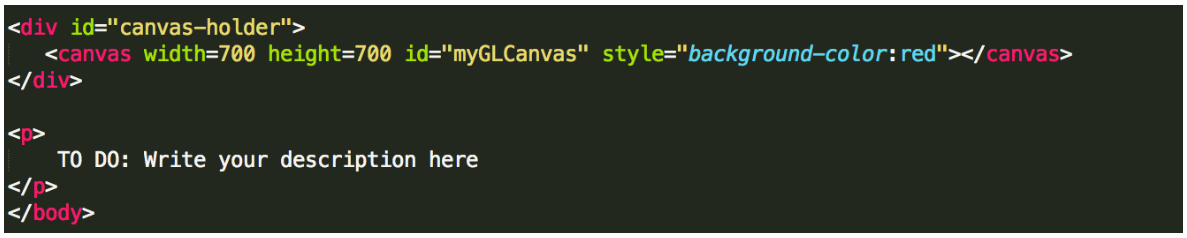
<u>TASK 2:</u> Once you have implement all the three lighting with Phong shading, add two more light positions, one at the bottom of the object and the other on the left of the object.

1. Add two more light positions for the user to select. When the user selects this light position, the scene should be rendered appropriately.



```
<label><b>Light Position:</b> <select id="lightpos">
    <option value="0">[0,0,0,1] (Viewpoint light)</option>
    <option value="1">[0,0,1,0] (Directional, into screen)</option>
    <option value="2">[0,1,0,0] (Directional, from above)</option>
    <option value="3">[0,0,-10,1] (On z-axis, close to object)</option>
    <option value="4">[2,3,5,0] (Directional from upper right)</option>
    TODO: Your code goes here.
    Add two more lights at different directions. One from the bottom and One
    from the left of the scene. Appropriately change the pa2.js file so that when user
    selects these new light positions, the scene gets updated.
    */ -->
</select></label><br>
```

2. What is happening when the user changes the specular exponent value? Why is this happening. Describe this no more than 3 lines below the scene in the pa2.html file. (Feel free to play around with different objects, lighting and light positions to get a sense as to what is happening)



```
<div id="canvas-holder">
   <canvas width=700 height=700 id="myGLCanvas" style="background-color:red"></canvas>
</div>

<p>
    TO DO: Write your description here
</p>
</body>
```

4. Submission

1) Make sure your name and ID is filled above the canvas, in pa2.html.
2) You will need to submit the following files on EEE in **a zip archive**, please DO NOT submit individual files.
    a. pa2.html
    b. p2.js
    c. model.js
    d. gl-matrix-min.js
    e. trackball-rotator.js

5. Grading

1) Your program should be able to correctly light up all the three objects for all the three lighting scenarios (ambient, diffuse, specular) using Phong shading. The user should be able to manipulate these scenarios using the lighting checkboxes.
2) You should be successfully able to add two new lighting positions (one at the bottom and one on the left) to the existing codebase. When selected, they should light the objects appropriately.
3) In your own words you should also describe how is the specular exponent value affecting the lighting.

6. Useful references

1) WebGL tutorial:
   http://learningwebgl.com/blog/?page_id=1217
2) JavaScript
   https://developer.mozilla.org/en-US/docs/Web/JavaScript
3) JS style guide:
   https://google.github.io/styleguide/javascriptguide.xml?showone=Comments#Comments
4) WebGL Shaders and GLSL (GL Shading Language)
   https://webglfundamentals.org/webgl/lessons/webgl-shaders-and-glsl.html