CMPUT 411/511

Assignment 4

©*Herb Yang*

October 15, 2021

Due date: **October 29, 2021** at 23:55 Total marks: 35

Important notes

- A late penalty of 10% per day applies to all late submissions. The maximum number of days late permitted is 1. After 1 day late, your assignment will be given a mark of **ZERO**.
- You are expected to complete the assignment on your own **without** collaboration with others. Discussions at the conceptual level but not at the coding level are permitted.
- You must write your own code. Using codes from the specified text or codes given in this course is permitted. Using any other codes is not permitted.
- Looking at other's codes or letting others to see your codes is not permitted.
- If your codes are used by another student, under the Code of Student Behaviour, you will be dealt with as participating in a plagiarism and cheating offence.
- If you hire a tutor, the tutor **cannot** give you step-by-step or code level instructions to solve the problem. This is defined as cheating.
- You are **not** permitted to upload this assignment to an online site to solicit solutions or to post your solutions to an online site for others to see. Violating this policy will result in severe penalty. These are defined as plagiarism and participation in a plagiarism offence, respectively.
- All your codes will be analyzed for potential plagiarism using MOSS.
- Do not wait until the last minute to work on your assignment. Start as soon as possible.
- Debugging programs and understanding of the materials take time. Debugging is part of the learning experience. You **must** not ask the TA to help you debug your programs.
- During lab, the TA is instructed to give directions but **not** solutions.
- For the programming part,
  - Your code should have sufficient comments to make it readable.
  - Include a README file to document features or bugs, if any, of your program.
- If there are questions, please ask the TA in the lab or via email.

Marking scheme:

- Full assigned marks - no observable bug
- Portions of the assigned marks - observable bugs. The TA has the discretion of assigning a fraction of the assigned marks depending on the number and the severity of the bugs.
- 0 marks - does not work at all

The objective of this assignment is to give you an opportunity to create shadow using WebGL. The shaders that you need to use are the following:

```
var VSHADER_SOURCE = `
  attribute vec4 a_Position;
  attribute vec4 a_Color;
  attribute vec4 a_Normal;
  uniform mat4 u_VpMatrix;       // view projection matrix
  uniform mat4 u_ModelMatrix;    // Model matrix
```

```
  uniform mat4 u_NormalMatrix;  // Transformation matrix of the normal
  varying vec4 v_Color;
  varying vec3 v_Normal;
  varying vec3 v_Position;
  void main() {
    gl_Position =  u_VpMatrix * u_ModelMatrix *  a_Position;
     // Calculate the vertex position in the world coordinate
    v_Position = vec3(u_ModelMatrix * a_Position);
    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));
    v_Color = a_Color;
  }
`;
// Herb Yang
// Aug. 20, 2021
// Fragment shader program
var FSHADER_SOURCE = `
  #ifdef GL_ES
  precision mediump float;
  #endif
  uniform vec3 u_LightColor;    // Light color
  uniform vec3 u_LightPosition;  // Position of the light source
  uniform vec3 u_AmbientLight;   // Ambient light color
  varying vec3 v_Normal;
  varying vec3 v_Position;
  varying vec4 v_Color;
  void main() {
     // Normalize the normal because it is interpolated and not 1.0 in length
        any more
    vec3 normal = normalize(v_Normal);
     // Calculate the light direction and make its length 1.
    vec3 lightDirection = normalize(u_LightPosition - v_Position);
     // The dot product of the light direction and the orientation of a
        surface (the normal)
    float nDotL = max(dot(lightDirection, normal), 0.0);
     // Calculate the final color from diffuse reflection and ambient
        reflection
    vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;
    vec3 ambient = u_AmbientLight * v_Color.rgb;
    gl_FragColor = vec4(diffuse + ambient, v_Color.a);
  }
`;
```

Also, you need to setup the light source first before you can see anything. The light source position can be set as

```
  var lightPosition = new Float32Array([10.0,10.0, 15.5]);
  // Set the light color (white)
  gl.uniform3f(u_LightColor, 1.0, 1.0, 1.0);
  // Set the light direction (in the world coordinate)
  gl.uniform3f(u_LightPosition, lightPosition[0], lightPosition[1],
     lightPosition[2]);
  // Set the ambient light
  gl.uniform3f(u_AmbientLight, 0.5, 0.3, 0.3);
```

After you have successfully setup all the required attribute and uniform variables in the shaders, then you can proceed to work on the following.

1. (1 mark) Define the `Plane` object. The input to the Plane constructor includes 4 vertices and their corresponding colours. Note, in your constructor, you need to initialize the normal, which should be computed based on the vertex information. Then use WebGL to create a blue plane with the following dimension and colour

```
// create a plane
  var planeVertices = new Float32Array([
    20.0, -3.0, 20.0,
    20.0, -3.0, -20.0,
    -20.0, -3.0, -20.0,
    -20.0, -3.0, 20.0
  ]);

  var planeColors = new Float32Array([
    0.3, 0.8, 1.0,
    0.3, 0.8, 1.0,
    0.3, 0.8, 1.0,
    0.3, 0.8, 1.0
  ]);
```

   The ground plane can be initialized using the following snippet:

```
groundPlane = new Plane(planeVertices, planeColors);// create the ground
    plane
```

   In the rest of this assignment, it is assumed that the view and projection matrix is set to:

```
setPerspective(45, canvas.width/canvas.height, 1, 100);
lookAt(8, 15, 20, 1, 0, -1, 0, 1, 0);
```

2. (2 marks) Define the `draw` method of the `Plane` object. Using the following code to draw the ground plane,

```
groundPlane.draw(); // draw the ground plane
```

   then you should see the following output (Fig. 1).

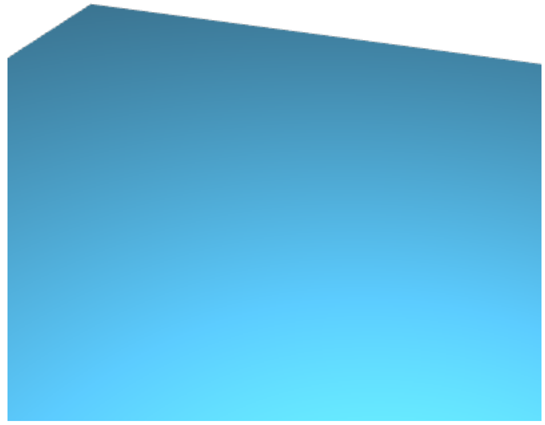

Figure 1: A light blue horizontal plane.

3. (5 marks) Define the `Cube` object. In your design, you **must** use the `Plane` object to construct the `Cube` object, i.e. a `Cube` object consists of 6 `Plane` objects. Note: No marks will be given for the rest of this assignment if you do not follow this restriction. The input to the constructor consists of 8 vertices and their corresponding colours.

4. (2 marks) Create a cube using the following vertices and colours.

```
// Create a cube
  //    v6----- v5
  //   /|      /|
  //  v1------v0|
  //  | |     | |
  //  | |v7---|-|v4
  //  |/      |/
  //  v2------v3
  // Coordinates
var cubeVertices = new Float32Array([
    1.0, 1.0, 1.0,  //v0
    -1.0, 1.0, 1.0, //v1
    -1.0, -1.0, 1.0,  //v2
    1.0, -1.0, 1.0, //v3
    1.0, -1.0, -1.0,  //v4
    1.0, 1.0, -1.0, //v5
    -1.0, 1.0, -1.0,  //v6
    -1.0, -1.0, -1.0//v7
  ]);
  var cubeColors = new Float32Array([
    1, 0, 0,  // v0
    1, 0, 0,  // v1
    1, 0, 0,  // v2
    1, 0, 0,    // v3
    1, 0, 0,  // v4
    1, 0, 0,    // v5
```

```
    1, 0, 0,  // v6
    1, 0, 0     // v7
  ]);
```

Create a cube and then draw it using the following code:

```
cube = new Cube(cubeVertices, cubeColors);
cube.draw(); // draw the cube
```

If everything works well, then you should see the following figure (Fig. 2).

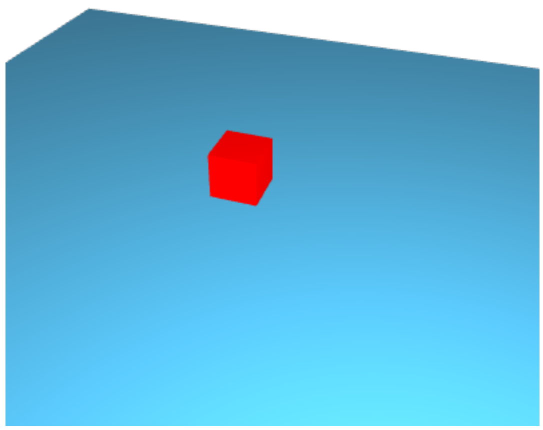

Figure 2: A red cube

5. (10 marks) Now, add shadow of the cube on the ground plane using the projection method (see Fig. 3). One possible design is to create the shadow in the `Plane` object, e.g. the ground plane. In other words, define a prototype method `shadow` of the `Plane` object. You need to think of the required inputs to the `shadow` method. Note that you need to derive the projection matrix for the shadow because the ground plane is at a position other than `y=0`. The color of the shadow is to reduce the color of the plane by some fraction. In the example shown, the reduction is 40%.

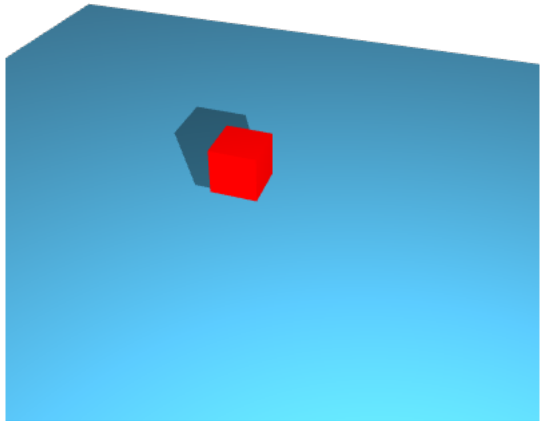

Figure 3: Casting shadow of a red cube with light source at (10.0, 10.0, 30.0)

The shadow as shown in the figure is rather short. You can make the object taller using scaling and translation.

6. (2 marks) Define a method to scale and a method to translate the `Cube` object. Note: Because the `Cube` object is made of the `Plane` objects, when you define a method for the `Cube` object, a similar method is required for the `Plane` object. You can see that the shadow is extended beyond the plane after scaling the cube by $(1, 3, 1)$ and translating it by $(1, 2, 1)$. You can see that the shadow extends beyond the ground plane (see Fig. 4)

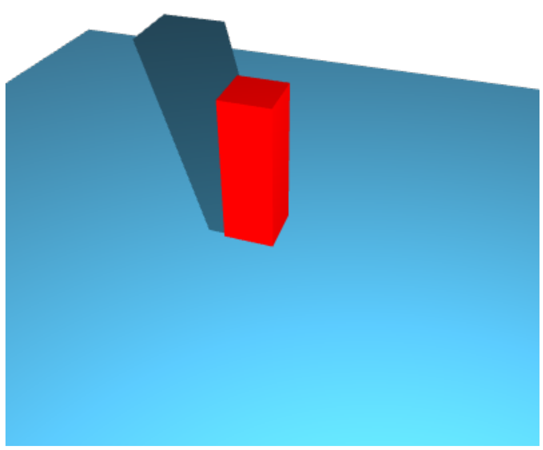

Figure 4: A shadow extended beyond the ground plane

7. (2 marks) One solution to the above problem is to use the stencil buffer. When the ground plane is drawn, the stencil buffer is also drawn. When the shadow is drawn, only areas in

the stencil buffer with non-zero values are available for drawing. In other words, anything drawn outside the ground plane will be clipped. Before you can use the stencil buffer, you must get the gl context with stencil buffer support using the following Javascript code:

```
var gl = canvas.getContext("webgl", {stencil:true});
```

To enable the stencil buffer, you do

```
gl.enable(gl.STENCIL_TEST);
gl.stencilOp(gl.REPLACE, gl.REPLACE, gl.REPLACE);
gl.stencilFunc(gl.ALWAYS,1,0xffffffff);
```

Before you draw the shadow, you need to disable depth test and set the appropriate stencil function. In this case, it is `gl.EQUAL`, which means that any pixel in the stencil buffer with a value of “1” is enabled for drawing. After the shadow is drawn, the stencil test is disabled and the depth test is enabled, see below.

```
gl.stencilFunc(gl.EQUAL,1,0xffffffff);
gl.stencilOp(gl.KEEP, gl.KEEP, gl.KEEP);
gl.disable(gl.DEPTH_TEST);
//draw the shadow
...
gl.disable(gl.STENCIL_TEST);
gl.enable(gl.DEPTH_TEST);
// draw the cube
...
```

If everything is done correctly, then you should see something similar to Fig. 5.

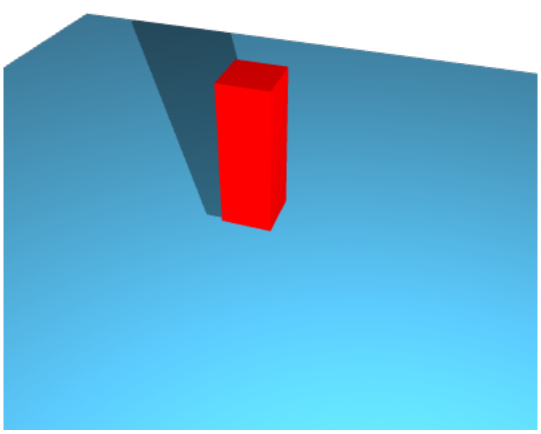

Figure 5: Result after using stencil buffer

8. (5 marks) Add rotation methods with respect to the x-axis and the y-axis to the `Cube` object. Fig. 6 shows an example of rotating the cube with respect to the y-axis by 30°.

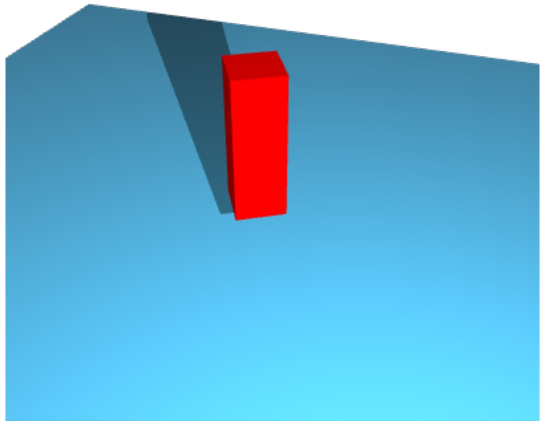

Figure 6: Result after rotating the object by 30° with respect to the y-axis

Fig. 7 shows an example of rotating the object with respect to the x-axis by 60°.

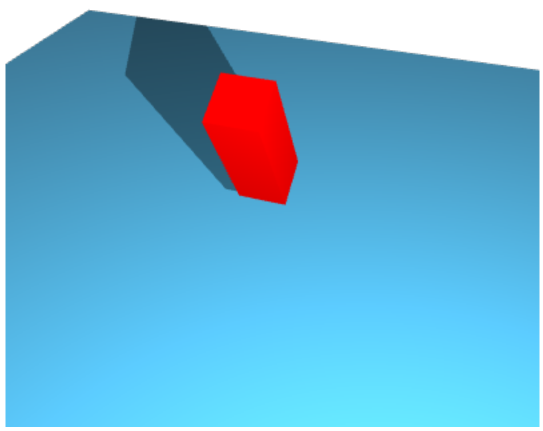

Figure 7: Result after rotating the object by 60° with respect to the x-axis

9. (5 marks) Add mouse interactions to rotate the object with respect to the x-axis and y-axis. Fig. 8 shows an example of rotating the object by dragging a mouse.

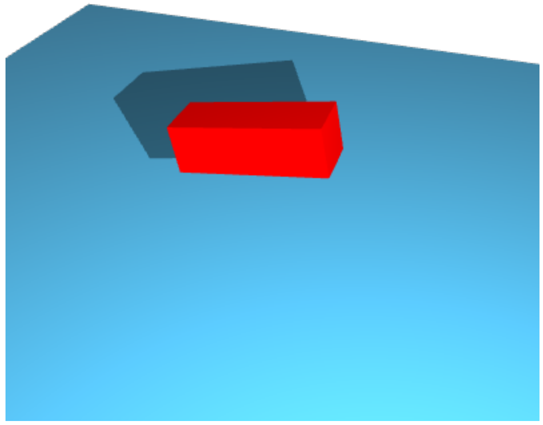

Figure 8: Use the mouse to rotate the object

10. (1 mark) Change the mouse interaction that controls the rotation with respect to the x-axis to control the height of the object (Fig. 9).

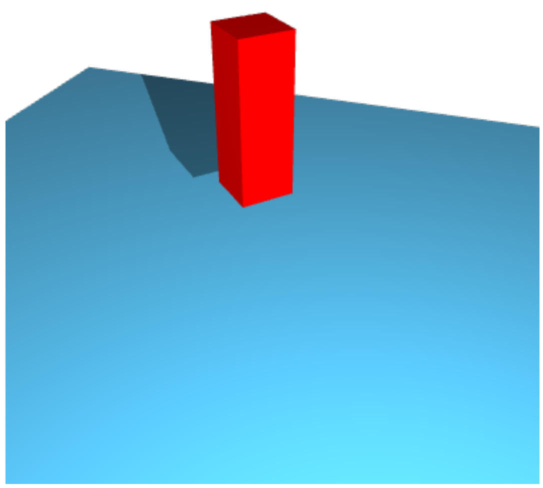

Figure 9: Use the mouse to move the object up and down

Submission information

- Organize the solution of each question in a separate folder, with a name that corresponds to each question, e.g. q1, q2,...
- Zip up all the folders in one single file
- Include a README file regarding your implementations, if needed
- Upload the zipped file to eClass