COMPUTER GRAPHICS – WEBGL

by vzbf32

Question (a)

[4 marks]

Describe how an index buffer object improves the use of a vertex buffer object in order to represent a 3D mesh.

An index buffer object tells the GPU which vertices (from the vertex buffer object) to join together to draw a triangle. Using an index buffer allows us to define a complete 3D mesh just by its vertices and avoids the need to repeat any of these vertex definitions. If we didn't have an index buffer object, then we would have to repeat a vertex definition if we used the same vertex as part of a different triangle. Using the index buffer object reduces redundancy.

Question (b)

[8 marks]

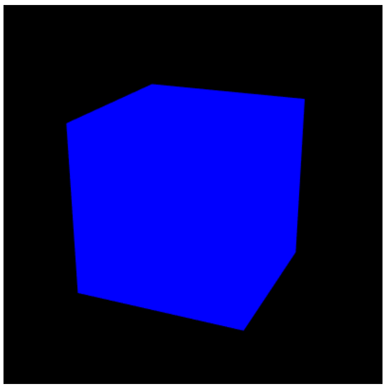
Given a fragment shader program as follows:

```
void main() { gl_FragColor = vec4(0.0, 0.0, 1.0, 1.0); }
```

Describe the functionality of this fragment shader program. If this fragment shader program is adopted by a WebGL program, which was supposed to support point lighting for mesh rendering, explain whether this fragment shader program will perform this job correctly. If yes, describe the rendering result produced. Otherwise, describe how you modify the program to support point lighting.

`gl_FragColor` is a built-in variable available only in a fragment shader. It controls the colour of a fragment, specified in RGBA format. In this case, the colour given is equivalent to `0x0000FF`, or blue.

The program does not correctly support point lighting. Applying it to a cube with a single point light, we get the following:

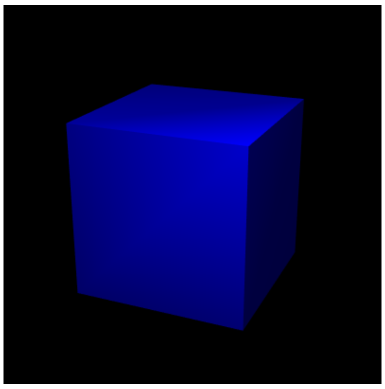


Instead, we should add a `varying vec4 v_Color` declaration and update the body of the `main()` function to `void main() { gl_FracColor = v_Color; }`.

We also need to add some information to the vertex shader to correctly calculate the lighting:

```
vec3 normal = normalize(vec3(u_NormalMatrix * a_Normal));
// Calculate world coordinate of vertex
vec4 vertexPosition = u_ModelMatrix * a_Position;
// Calculate the light direction and make it 1.0 in length
vec3 lightDirection = normalize(u_LightPosition - vec3(vertexPosition));
// The dot product of the light direction and the normal
float nDotL = max(dot(lightDirection, normal), 0.0);
// Calculate the color due to diffuse reflection
vec3 diffuse = u_LightColor * a_Color.rgb * nDotL;
// Calculate the color due to ambient reflection
vec3 ambient = u_AmbientLight * a_Color.rgb;
//  Add the surface colors due to diffuse reflection and ambient reflection
v_Color = vec4(diffuse + ambient, a_Color.a);
```

This computes the point lighting by combining the diffuse and the ambient lighting, and gives us the correctly shaded cube as follows:



Source: https://github.com/huningxin/webglbook_examples/blob/master/cho8/PointLightedCube.js, retrieved 23/03/2018

Question (c) [8 marks]

Describe the main functionality of a normal vector in terms of 3D mesh rendering. Suppose in a WebGL program, the initial normal vectors of a 3D mesh have been pre-computed. If this 3D mesh is then being rotated in the program, explain how you will update its normal vectors accordingly.

The normal vectors are used to calculate lighting. The calculation performed by the vertex shader is:

$$\begin{aligned}&\langle\text{surface colour by diffuse reflection}\rangle\\ &\qquad = \langle\text{light colour}\rangle \times \langle\text{base colour of surface}\rangle\\ &\qquad \times \left(\langle\text{light direction}\rangle \cdot \langle\text{orientation of a surface}\rangle\right).\end{aligned}$$

The orientation of the surface is defined by the normal vector, as the normal is always perpendicularly oriented to the surface. The vertex shader can use this to calculate the surface colour by the diffuse reflection, as given by the equation above.

If a 3D mesh is being rotated, its normal vectors will also change. To calculate the new normal vectors, the rule is to multiply the normal by the inverse transpose of the model matrix. Practically, this can be done with the `Matrix4` object. For example, if we rotate 90 degrees clockwise around the z-axis and then translate 0.9 units along the y-axis:

```
var modelMatrix = new Matrix4();  // Model matrix
var normalMatrix = new Matrix4(); // Transformation matrix for normal

// Calculate the model matrix
modelMatrix.setTranslate(0, 1, 0); // Translate to y-axis direction
modelMatrix.rotate(90, 0, 0, 1);   // Rotate around the z-axis

// ...

// Calculate matrix to transform normal based on the model matrix
normalMatrix.setInverseOf(modelMatrix);
normalMatrix.transpose();
```

Question (d)

(Reference: WebGL Prog. Guide Chapter 4, Table 4.1)

Suppose `drawBox(m)` is a function to draw a transformed matrix `m`. That is, if `m` is a rotation matrix, the function will draw a rotated box.

Part (i)

[7 marks]

Explain the meaning of the following code segment and state the result obtained:

```
m.setTranslate(20.0, 0.0, -30.0);
m.rotate(angle, 2.0, 0.0, 0.0);
drawBox(m);
```

The first line sets the matrix `m` to the translation matrix, which translates 20 units in the direction of the x-axis, no units in the direction of the y-axis and -30 units in the direction of the z-axis.

The second line rotates the matrix stored in `m` (currently the translation matrix described above) by 2 degrees around the x-axis.

The third line draws the box.

As transformations are applied in reverse order due to the way matrix multiplication works, the box that is drawn will have been rotated by 2 degrees around the x-axis, then translated by the vector:

$$\begin{bmatrix} 20 \\ 0 \\ -30 \end{bmatrix}.$$

Part (ii)

[3 marks]

Explain whether you will get the same result if `m.rotate()` has been replaced by `m.setRotate()`.

The `set` prefix calculates a transformation matrix and then writes it to `m`, the Matrix4 object. In contrast, omitting the `set` prefix multiplies the calculated transformation matrix with the matrix already stored in `m` and stores the result back in `m`.

This means that, if we were to use `m.setRotate()` instead of `m.rotate()`, we would essentially be discarding the result of the `m.setTranslate()` command, and the box that is drawn would only be rotated, not translated.