INSTITUTO SUPERIOR TÉCNICO

Animação e Visualização Tridimensional

Mestrado em Engenharia Informática e de Computadores
Alameda/Tagus

1º mini-teste
16 Outubro 2019

The mini-test has a maximum duration of 45 minutes. Answer with black or blue pen to the following questions and **justify in detail** all the answers. If necessary you can use the back of the respective sheet to complete the answer. Calculators, cell phones or other mobile devices are not allowed. Identify all the sheets of your mini-test.

Good luck!

Synopsis of some commands used in the code samples of the mini-test:

```
void lookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ, GLdouble centerX,
       GLdouble centerY, GLdouble centerZ, GLdouble upX, GLdouble upY, GLdouble upZ);
```

1. Assume that you have set up one buffer with vertices' attributes (position, normal, tangent and texture coordinates) of an object by using glBufferData() together with glBufferSubData(..). You will draw that object with the call glDrawElements(GL_TRIANGLES,……).
   a) **(1.5 points)** What VBOs should be bound to the object's VAO?

   Since we are using glBufferData() with glBufferSubData(..) we have two VBOs: The VBO with vertices' attributes (type GL_ARRAY_BUFFER) and the VBO with the indices of the vertices per triangle (type GL_ELEMENT_ARRAY_BUFFER).

   b) **(1.5 points)** The object is a mesh with 5 quads. How many elements do you have in the index buffer?

   5 x 2 triangles/quad x 3 vertices= 30 elements

2. **(1.5 points)** Consider the following OpenGL code sample. Indicate the location (index) bound to the *normal* attribute variable in the GLSL *p* program?

```
enum AttribType {VERTEX_COORD, TEXTURE_COORD, TANGENT_ATTRIB, NORMAL_ATTRIB};
glBindFragDataLocation(p, 0,"colorOut");
glBindAttribLocation(p, VERTEX_COORD, "position");
glBindAttribLocation(p, NORMAL_ATTRIB, "normal");
glBindAttribLocation(p, TEXTURE_COORD, "texCoord");
glBindAttribLocation(p, TANGENT_ATTRIB, "tangent");
glLinkProgram(p);
pvm_Id = glGetUniformLocation(p, "m_pvm");
vm_uniformId = glGetUniformLocation(p, "m_viewModel");
normal_uniformId = glGetUniformLocation(p, "m_normal");
```

`glBindAttribLocation(p, NORMAL_ATTRIB, "normal");` imposes location 3 (fourth position in the enum) for the GLSL normal GLSL variable

Aluno: _____ _____

3. Consider the following excerpt of code in OpenGL 3.3. Assume that a stack mechanism was implemented for all three types of matrices MODEL, VIEW and PROJECTION used by the Geometric Transform stage of the OpenGL pipeline.

```
void renderScene(void) {
….
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
loadIdentity(VIEW);
loadIdentity(MODEL);
lookAt(0.0, 0.0, 2.0, 1.0, 1.0, 2.0, 1.0, 0.0, 0.0)
translate(MODEL, 0.5f, 1.5f, 1.0f);
send_matrices(); //send the matrices PROJECTION, VIEW and MODEL to GLSL
draw_obj1();
pushMatrix(MODEL);
scale (MODEL, 2.0f, 0.5f, 1.0f);
send_matrices(); //send the matrices PROJECTION, VIEW and MODEL to GLSL
draw_obj2();
popMatrix(MODEL);
translate (MODEL, 1.5f, 1.5f, 1.5f);
send_matrices(); //send the matrices PROJECTION, VIEW and MODEL to GLSL
draw_obj3();
……
```

a) **(2.5 points)** Calculate the last column of the matrix VIEW sent to the GLSL in order to draw each of the three objects.

$$VRP = [eye_x \quad eye_y \quad eye_z] = [0 \quad 0 \quad 2]$$

$$VPN = [center_x - eye_x \quad center_y - eye_y \quad center_z - eye_z] = [1 \quad 1 \quad 0]$$

$$\overrightarrow{up} = [up_x \quad up_y \quad up_z] = [1 \quad 0 \quad 0]$$

$$\vec{n} = \frac{VPN}{\|VPN\|} = \frac{[1 \quad 1 \quad 0]}{2} = \left[{}^{1}/_{\sqrt{2}} \quad {}^{1}/_{\sqrt{2}} \quad 0\right]$$

$$\vec{u} = \frac{\vec{n} \times \overrightarrow{up}}{\|\vec{n} \times \overrightarrow{up}\|} = \frac{\left[{}^{1}/_{\sqrt{2}} \quad {}^{1}/_{\sqrt{2}} \quad 0\right] \times [1 \quad 0 \quad 0]}{\left\|\left[{}^{1}/_{\sqrt{2}} \quad {}^{1}/_{\sqrt{2}} \quad 0\right]\right\|} = \frac{\left[0 \quad 0 \quad {}^{-1}/_{\sqrt{2}}\right]}{\left\|\left[0 \quad 0 \quad {}^{-1}/_{\sqrt{2}}\right]\right\|}$$
$$= [0 \quad 0 \quad -1]$$

$$\vec{v} = \vec{u} \times \vec{n} = [0 \quad 0 \quad -1] \times \left[{}^{1}/_{\sqrt{2}} \quad {}^{1}/_{\sqrt{2}} \quad 0\right] = \left[{}^{1}/_{\sqrt{2}} \quad {}^{-1}/_{\sqrt{2}} \quad 0\right]$$

$$M_{View} = \begin{bmatrix} u_x & u_y & u_z & -\vec{u} \cdot VRP \\ v_x & v_y & v_z & -\vec{v} \cdot VRP \\ -n_x & -n_y & -n_z & \vec{n} \cdot VRP \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 & 2 \\ {}^{1}/_{\sqrt{2}} & {}^{-1}/_{\sqrt{2}} & 0 & 0 \\ {}^{-1}/_{\sqrt{2}} & {}^{-1}/_{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b) **(2.5 points)** Calculate the matrix MODEL sent to the GLSL in order to draw each of the three objects.

I x T[0.5 1.5 1.0]

Aluno: _____ _____

I x T[0.5 1.5 1.0] x S[2.0 0.5 1.0]
I x T[0.5 1.5 1.0] x T[1.5 1.5 1.5]

4. Analyze, in the last sheet of the mini-test, the OpenGL(OGL) code snippet as well as the GLSL version 330 of both *vertex shader* and *fragment shader*. The RGB channels of the three components of the source light have unit values. The reflection model is the Blinn-Phong.

a) (1.5 points) Consider the variable l*ightPos* used in this OGL program Identify the type of source light used as well as the space coordinates of its position. Justify.
Point light since the last coordinate of *ligthPos* is 1.
lightPos is multiplied by the VIEW matrix before to be sent to the GLSL program which means that it was described in World coordinates

b) (2.5 points) Write the GLSL code to calculate, in Eye space, the following entities:
n (normal), ***l*** (light direction), ***pos*** (vertex position), ***e*** (eye direction) and ***h*** (half-vector)

```
vec4 pos = m_viewModel * position;
vec3 l = normalize(vec3(l_pos – pos))
vec3 n = normalize(m_normal * normal.xyz);
vec3 e = normalize(vec3(-pos));
vec3 h = normalize(l + e);
```

c) (2 points) What shading technique is being used to draw the scene? Why?

Gouraud shading. Blinn-Phong color is calculated in the vertex shader, thus at vertex level. Then the color is **interpolated** by the rasterizer and received in the corresponding fragment shader that only displays the received color.

d) (1.5 points) Identify the space where the built-in GLSL variable *gl_Position* is described.

Clipping Coordinates space

e) (3 points) Assume, for a particular vertex, that the angle between *n* and *l*, and the angle between *e* and *n*, are both 60º. Calculate the value of *colorOut*. You should indicate how to calculate both diffuse and specular components.
(cos 30º = 0.866, cos 45º = 0.707, cos 60º = 0.5)

H between l and e which means an angle half of 2*60=60. Vector e, in this case coincides the mirror vector. This means an angle 0 between h and n
Intensity_diffuse= dot (n, l)= cos 60
Intensity_spec= dot(h,n) = cos 0 = 1
Max {[0 0.8 0.8]*0.5 + [0 0 0.5]* 1 * exp100; [0.1 0.2 0.2]} = [0.1 0.4 0.9]

Aluno: _____ _____

```
GLfloat lightPos[4] = {4.0f, 6.0f, 2.0f, 1.0f};
GLfloat mat_ambient[] = { 0.1 0.2, 0.2, 1.0 };
GLfloat mat_diffuse[] = { 0.0, 0.8, 0.8, 1.0 };
GLfloat mat_specular[] = { 0.0, 0.0, 0.5, 1.0 };
GLfloat mat_shininess= 100.0f;
multMatrixPoint(VIEW, lightPos, result);
loc = glGetUniformLocation(p,”l_pos”);
glUniform4fv(loc, 1, result);
loc = glGetUniformLocation(p, "mat.ambient");
glUniform4fv(loc, 1, mat_ambient);
loc = glGetUniformLocation(p, "mat.diffuse");
glUniform4fv(loc, 1, mat_diffuse);
glGetUniformLocation(p, "mat.specular");
glUniform4fv(loc, 1, mat_specular);
loc = glGetUniformLocation(p, "mat.shininess");
glUniform1f(loc,mat_shininess);

    ------Vertex shader

    uniform mat4 m_pvm;  // proj * view * model
    uniform mat4 m_viewModel; // view * model
    uniform mat3 m_normal;  // normal matrix
    struct Materials {
           vec4 diffuse, ambient, specular, emissive;
           float shininess; };
    uniform Materials mat;
    uniform vec4 l_pos;

    in vec4 position;
    in vec4 normal;
    out vec4 v_color;

    void main () {
           vec4 pos = _____________________________;
           vec3 l = ____________________________________________;
           vec3 n = ____________________________________________;
           vec4 spec = vec4(0.0);

           float intensity = max(dot(n,l), 0.0);
           if (intensity > 0.0) {
               vec3 e = ____________________________________________;
               vec3 h = ____________________________________________;
                  float intSpec = max(dot(h,n), 0.0);
                  spec = mat.specular * pow(intSpec, mat.shinines);
           }
           v_color = max(intensity*mat.diffuse+spec,mat.ambient);
           gl_Position = m_pvm * position;
    }

    ------fragment shader

    out vec4 colorOut;
    in vec4 v_color;

    void main() {

           colorOut = v_color;
    }
```