Animação e Visualização Tridimensional

Mestrado em Engenharia Informática e de Computadores
Alameda

INSTITUTO SUPERIOR TÉCNICO

1º mini-teste
21 de Outubro de 2015

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 VBO with all vertices' attributes 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?

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 12 quads. How many elements do you have in the index buffer?

The index VBO (type GL_ELEMENT_ARRAY_BUFFER) should have 12 quads * 2 triangles/quad * 3 vertices/triangle = 72 elements.

2. **(1 point)** Consider the following OpenGL code sample. Indicate the location (index) of the *normal* variable in the GLSL program referenced by *p*?

```
enum AttribType { NORMAL_ATTRIB, TEXTURE_COORD, VERTEX_COORD};
glBindFragDataLocation(p, 0,"colorOut");
glBindAttribLocation(p, VERTEX_COORD, "position");
glBindAttribLocation(p, NORMAL_ATTRIB, "normal");
glBindAttribLocation(p, TEXTURE_COORD, "texCoord");
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");` Esta instrução impõe que o índice (location) da variável "normal" é 0 (posição de NORMAL_COORD no enum)

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, 2.0, 2.0, -2.0, 0.0, 0.0, 1.0)
pushMatrix(MODEL);
translate(MODEL, 2.0f, 2.5.0f, 1.0f);
send_matrices(); //send the matrices PROJECTION, VIEW and MODEL to GLSL
draw_obj1();
pushMatrix(MODEL);
scale (MODEL, 1.5f, 1.5f, 1.5f);
send_matrices(); //send the matrices PROJECTION, VIEW and MODEL to GLSL
draw_obj2();
popMatrix(MODEL);
popMatrix(MODEL);
send_matrices(); //send the matrices PROJECTION, VIEW and MODEL to GLSL
draw_obj3();
……
```

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

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

$$VPN = [center_x - eye_x \quad center_y - eye_y \quad center_z - eye_z] = [2 \quad 2 \quad 0]$$

$$VUV' = [up_x \quad up_y \quad up_z] = [0 \quad 0 \quad 1]$$

$$VUV = VUV' - VPN(VPN \cdot VUV') = [0 \quad 0 \quad 1] - [2 \quad 2 \quad 0] * ([2 \quad 2 \quad 0] \cdot [0 \quad 0 \quad 1])$$
$$= [0 \quad 0 \quad 1] - [2 \quad 2 \quad 0] * 0 = [0 \quad 0 \quad 1]$$

$$\vec{n} = \frac{VPN}{\|VPN\|} = \frac{[2 \quad 2 \quad 0]}{\sqrt{8}} = \left[{}^{1}/_{\sqrt{2}} \quad {}^{1}/_{\sqrt{2}} \quad 0\right]$$

$$\vec{v} = \frac{VUV}{\|VUV\|} = \frac{[0 \quad 0 \quad 1]}{1} = [0 \quad 0 \quad 1]$$

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

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

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

Objecto 1: MODEL: Identity * $\begin{matrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2.5 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{matrix}$

Objecto 2: MODEL: Identity * $\begin{matrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2.5 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{matrix} * \begin{matrix} 1.5 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} = \begin{matrix} 1.5 & 0 & 0 & 2 \\ 0 & 1.5 & 0 & 2.5 \\ 0 & 0 & 1.5 & 1 \\ 0 & 0 & 0 & 1 \end{matrix}$

Objecto 3: Identity matrix

4. Analyze, in the last section of the mini-test, the OpenGL 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 and its position is described in the Eye space.

a) (2 valores) 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 at each fragment and received in the corresponding fragment shader that only displays the received color.

b) (1 point) Identify the type of source light used in this program? Justify.
Directional light because the 4th component of array lightPos is 0

c) (2 points) Identify the coordinate's space of the GLSL variables *n*, *e* and *l* in the vertex shader? Justify.
Eye coordinates: n = input normal multiplied by m_normal), e= (0, 0, 0)- (input vertex position multiplied by viewModel) and l is related with the l_pos which receives from the OpenGL application the light direction in eye space (multMatrixPoint(VIEW, lightPos, result)

d) (2 points) What is the relationship between the mat3 *m_normal* and mat4 *m_viewModel*?
Get the submatrix 3x3 from *m_viewModel* (get ride the last column and the last row): *m_viewModel3x3.* Then:

$m\_normal = ([m\_viewModel3x3]^T)^{-1}$

Aluno: _________ ____

e) **(2 points)** Identify the space where the built-in GLSL variable *gl_Position* is described.

Clip Coordinates. To be used in the Clipping stage.

f) Assume, for a particular vertex, that the angle between *n* and *l*, and the angle between *e* and *n*, are respectively 60 degree and 30 degrees.

i. **(1 point)** Write the numerical expression to calculate the specular color.

half-vector h = l + e

dot(h,n) = cos 15º

spec = [0, 0, 0.5* pow (cos 15, 100), 1]

ii. **(1 point)** Just to simplify the computation (since you are not allowed to use a calculator), consider that dot(h,n)=1. Calculate the value of DataOut.color.

(cos 30º = 0.866, cos 45º = 0.707, cos 60º = 0.5)

dot(h,n) = 1

spec = [0, 0, 0.5, 1]

dot (l,n) = cos 60º=0,5

intensity*diffuse = [0 0.4 0.4 1]

spec = [0 0 0.5 1].

intensity*diffuse + spec= [0.0 0.4 0.9 1.0]

DataOut.color = [0.1 0.4 0.9 1.0]

```
GLfloat lightPos[4] = {4.0f, 6.0f, 2.0f, 0.0f};
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 Data { vec4 color;} DataOut;

void main () {
       vec3 l = normalize(l_pos.xyz);
       vec3 n = normalize(m_normal * normal.xyz);
       vec4 spec = vec4(0.0);

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

------fragment shader

out vec4 colorOut;
in Data { vec4 color;} DataIn;

void main() {

       colorOut = DataIn.color;
}
```