INSTITUTO SUPERIOR TÉCNICO

# Animação e Visualização Tridimensional

Mestrado em Engenharia Informática e de Computadores
*Alameda*

**1st Mini-Test Repetition**
16th December 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 three buffers with vertices' attributes (position, normal and texture coordinates) of an object by using glBufferData(). 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?

   4 VBO (3 VBOs for the vertices' attributes and one for the indices)

   b) **(1.5 points)** The object is a mesh with 16 quads. How many elements do you have in the index buffer?

   The index VBO (type GL_ELEMENT_ARRAY_BUFFER) should have 16 quads * 2 triangles/quad * 3 vertices/triangle = 96 elements.

2. **(1 point)** Consider the following OpenGL code sample. One function is wrongly placed in the code. Tell why and place it correctly.

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

Aluno: _____ _____

The line code
`glLinkProgram(p);`
should be placed after the last `glBindAttribLocation()` command

**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, -1.0, 1.0, 1.0, -1.0, 0.0, 0.0, 1.0)
pushMatrix(MODEL);
translate(MODEL, 0.5f, 1.5f, 1.0f);
scale (MODEL, 2.0f, 0.5f, 1.0f);
send_matrices(); //send the matrices PROJECTION, VIEW and MODEL to GLSL
draw_obj1();
pushMatrix(MODEL);
translate (MODEL, 1.5f, 1.5f, 1.5f);
send_matrices(); //send the matrices PROJECTION, VIEW and MODEL to GLSL
draw_obj2();
popMatrix(MODEL);
send_matrices(); //send the matrices PROJECTION, VIEW and MODEL to GLSL
draw_obj3();
popMatrix(MODEL);
……
```

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

$$VRP = [eye_x \quad eye_y \quad eye_z] = [0 \quad 0 \quad -1]$$
$$VPN = [center_x - eye_x \quad center_y - eye_y \quad center_z - eye_z] = [1 \quad 1 \quad 0]$$
$$VUV' = [up_x \quad up_y \quad up_z] = [0 \quad 0 \quad 1]$$
$$VUV = VUV' - VPN(VPN \cdot VUV') = [0 \quad 0 \quad 1] - [1 \quad 1 \quad 0] * ([1 \quad 1 \quad 0] \cdot [0 \quad 0 \quad 1])$$
$$= [0 \quad 0 \quad 1] - [1 \quad 1 \quad 0] * 0 = [0 \quad 0 \quad 1]$$

$$\vec{n} = \frac{VPN}{\|VPN\|} = \frac{[1 \quad 1 \quad 0]}{2} = \left[{}^{1}/_{\sqrt{2}} \quad {}^{1}/_{\sqrt{2}} \quad 0\right]$$
$$\vec{v} = \frac{VUV}{\|VUV\|} = \frac{[0 \quad 0 \quad 1]}{1} = [0 \quad 0 \quad 1]$$
$$\vec{u} = \vec{n} \times \vec{v} = \left[{}^{1}/_{\sqrt{2}} \quad {}^{1}/_{\sqrt{2}} \quad 0\right] \times [0 \quad 0 \quad 1] = \left[{}^{1}/_{\sqrt{2}} \quad {}^{-1}/_{\sqrt{2}} \quad 0\right]$$
$$M_{View@LookAt} = \begin{bmatrix} u_x & u_y & u_z & -\vec{u} \cdot VRP \\ v_x & v_y & v_z & -\vec{v} \cdot VRP \\ -n_x & -n_y & -n_z & \vec{n} \cdot VRP \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^{1}/_{\sqrt{2}} & {}^{-1}/_{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 1 \\ {}^{-1}/_{\sqrt{2}} & {}^{-1}/_{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**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 $*$
$$\begin{matrix}1&0&0&0.5\\0&1&0&1.5\\0&0&1&1\\0&0&0&1\end{matrix} * \begin{matrix}2.0&0&0&0\\0&0.5&0&0\\0&0&1.0&0\\0&0&0&1\end{matrix}$$

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

Objecto 3 = Objecto 1

**4.** Analyze, in the last sheet 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.

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

Phong shading. Blinn-Phong color is calculated in the fragment shader, thus at fragment level, by using the interpolated normal calculated in the vertex shader

**b)** **(1 point)** Identify the type of source light used in this program as well as the coordinates' space where it is defined? Justify.

Point light because the 4th component of array lightPos is 1. The light is defined in eye coordinates since it is sent to the shaders **without any transformation**:

```
loc = glGetUniformLocation(p,"l_pos");
glUniform4fv(loc, 1, lightPos);
```

**c)** **(2 points)** Identify the coordinate's space of the GLSL variables *n*, *e* and *l* in the fragment shader? Justify.

Their values come from the interpolated DataIn variable which is in eye coordinates

**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 )** Why the normal stored in *DataIn.normal* is normalized in the fragment shader since it was already normalized in the vertex shader?

The normal should be normalized in the vertex shader in order to guarantee that the interpolation yields a vector with the correct direction. But its length can be different from one, so it is necessary to normalize it again in the fragment shader

**f) (2 points)** Assume, for a particular fragment, that the angle between *n* and *l*, and the angle between *e* and *n*, are both 60 degree. Calculate the value of *colorOut*.
(cos 30º = 0.866, cos 45º = 0.707, cos 60º = 0.5)

O half-vector h = l + e
logo h faz zero graus com a normal o que significa que dot(h,n) = 1
cos 60 = 0.5
intensity*diffuse = [ 0 0.3 0.3 1]
spec = [0 0 0.5 1].
intensity*diffuse + spec= [0.0 0.3 0.8 1.0]
colorOut vai escolher o max deste vector e do ambiente pelo que
colorOut = [ 0.2 0.3 0.8 1.0]

```
GLfloat lightPos[4] = {4.0f, 6.0f, 2.0f, 1.0f};
GLfloat mat_ambient[] = { 0.2 0.2, 0.5, 1.0 };
GLfloat mat_diffuse[] = { 0.0, 0.6, 0.6, 1.0 };
GLfloat mat_specular[] = { 0.0, 0.0, 0.5, 1.0 };
GLfloat mat_shininess= 80.0f;
loc = glGetUniformLocation(p,”l_pos”);
glUniform4fv(loc, 1, lightPos);
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
uniform vec4 l_pos;

in vec4 position;
in vec4 normal;

out Data { vec3 normal, eye, lightDir;} DataOut;

void main () {

        vec4 pos = m_viewModel * position;
```

Aluno: _____ _____

```
        DataOut.normal = normalize(m_normal * normal.xyz);
        DataOut.lightDir = vec3(l_pos - pos);
        DataOut.eye = vec3(-pos);
        gl_Position = m_pvm * position;
}

------fragment shader

out vec4 colorOut;
struct Materials {
        vec4 diffuse, ambient, specular, emissive;
        float shininess; };
uniform Materials mat;

in Data { vec3 normal, eye, lightDir;} DataIn;

void main() {

        vec4 spec = vec4(0.0);
        vec3 n = normalize(DataIn.normal);
        vec3 l = normalize(DataIn.lightDir);
        vec3 e = normalize(DataIn.eye);

        float intensity = max(dot(n,l), 0.0);

        if (intensity > 0.0) {

                vec3 h = normalize(l + e);
                float intSpec = max(dot(h,n), 0.0);
                spec = mat.specular * pow(intSpec, mat.shininess);

        colorOut = max(intensity*mat.diffuse + spec, mat.ambient);
}
```