



LEI - Computação Gráfica
prof. André Perrotta, prof. Evgheni Polisciuc

Melhoria mini-testes

Nome:

Número:

Duração: 60min

31 de Janeiro, 2024

valor max: 20

Formulário

sejam os vetores $\vec{A}(a_1, a_2, a_3)$ e $\vec{B}(b_1, b_2, b_3)$
produto escalar:

$$\vec{A} \bullet \vec{B} = \sum_{i=1}^3 a_i b_i = |\vec{A}| |\vec{B}| \cos \theta$$

produto vetorial:

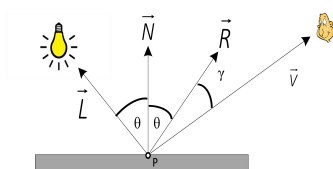
$$\vec{A} \times \vec{B} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = (a_2 b_3 - b_2 a_3) \hat{x} + (a_3 b_1 - b_3 a_1) \hat{y} + (a_1 b_2 - b_1 a_2) \hat{z}$$

transformações geométricas:

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Proj_{perspectiva_{openGl}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \quad Proj_{ortogonal} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Modelo de Phong para iluminação:



	0°	30°	45°	60°	90°
$\sin(\theta)$	0	$\frac{1}{2}$	$\frac{1}{\sqrt{2}}$	$\frac{\sqrt{3}}{2}$	1
$\cos(\theta)$	1	$\frac{\sqrt{3}}{2}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{2}$	0
$\tan(\theta)$	0	$\frac{1}{\sqrt{3}}$	1	$\sqrt{3}$	undefined

$$\vec{R} = 2(\vec{L} \bullet \vec{N})\vec{N} - \vec{L}$$

$$I_{vertex} = I_{lu_{amb}} K_{mat_{amb}} + I_{lu_{dif}} K_{mat_{dif}} \cos \theta + I_{lu_{spec}} K_{mat_{spec}} \cos \gamma^{n_s}$$

Implementação das funções *perspective(...)* e *lookat(...)* (implementação utilizada nas aulas PL)

```
void perspective(
    GLfloat theta,
    GLfloat alpha,
    GLfloat beta,
    bool invertX = false,
    bool invertY = false
){
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    GLfloat tan = tanf(theta*0.5 * PI / 180.0);
    GLfloat d = (gh() / 2.0) / tan;
    GLfloat nearClip = d / alpha;
    GLfloat farClip = d * beta;
    GLfloat ymax = nearClip * tan;
    GLfloat xmax = (gw() / gh()) * ymax;
    if (invertX) {
        xmax = -xmax;
    }
    if (invertY) {
        ymax = -ymax;
    }
    glFrustum(-xmax, xmax, -ymax, ymax, nearClip, farClip);
}
```

```
void lookat(
    GLfloat camX,
    GLfloat camY,
    GLfloat camZ,
    GLfloat targetX,
    GLfloat targetY,
    GLfloat targetZ,
    GLfloat upX,
    GLfloat upY,
    GLfloat upZ
){
    ofVec3f cam = ofVec3f(camX, camY, camZ);
    ofVec3f target = ofVec3f(targetX, targetY, targetZ);
    ofVec3f up = ofVec3f(upX, upY, upZ);

    ofVec3f N = cam - target;
    N = N.normalized();
    ofVec3f U = cross(up, N);
    U = U.normalized();
    ofVec3f V = cross(N, U);
    V = V.normalized();

    GLfloat camTransformMatrix[4][4] = {
        {U.x, V.x, N.x, 0},
        {U.y, V.y, N.y, 0},
        {U.z, V.z, N.z, 0},
        {-U.dot(cam), -V.dot(cam), -N.dot(cam), 1}
    };

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glMultMatrixf(&camTransformMatrix[0][0]);
}
```

Geometria e transformações

Q1 (4 valores)

A cena a seguir foi desenhada com OpenFrameworks/OpenGL. Os valores de coordenadas e tamanhos foram adicionados em pós-processamento da imagem gerada pelo código, e estão em valores de "coordenadas mundo". A aplicação foi configurada para uma janela de 1024x1024 pixels, definida em main.cpp.

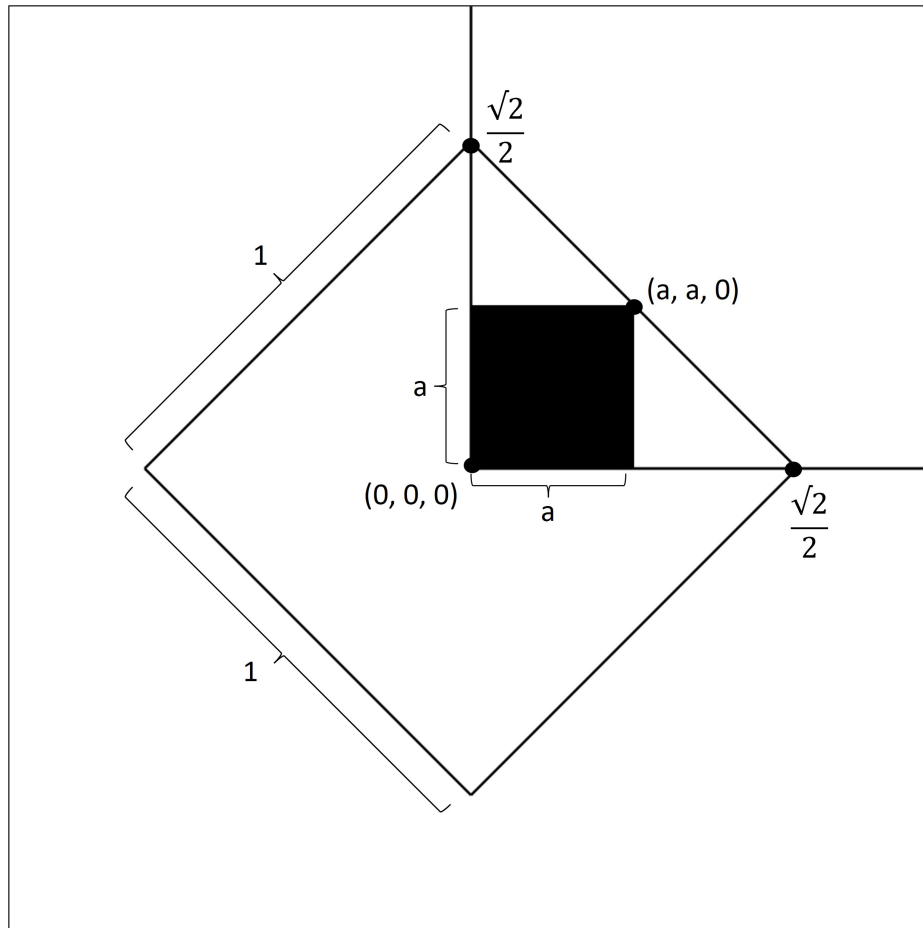


imagem gerada pela aplicação OF/OpenGL

Funções utilizadas

Para a construção do desenho foram utilizadas as seguintes funções customizadas:

```
void rect(){
    glBegin(GL_QUADS);
    glVertex3f(-0.5, 0.5, 0.);
    glVertex3f(-0.5, -0.5, 0.);
    glVertex3f(0.5, -0.5, 0.);
    glVertex3f(0.5, 0.5, 0.);
    glEnd();
}
```

```
void axis() {
    glBegin(GL_LINES);
    glVertex3f(0, 0, 0);
    glVertex3f(1, 0, 0);
    glVertex3f(0, 0, 0);
    glVertex3f(0, 1, 0);
    glVertex3f(0, 0, 0);
    glVertex3f(0, 0, 1);
    glEnd();
}
```

Complete o código com os valores necessários para a gerar a imagem apresentada anteriormente. Se achar pertinente, coloque `"/"` (comments) nas transformações que julgar desnecessárias.

```
void draw(){

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1, 1, -1, 1, -2, 2);
    glMatrixMode(GL_MODELVIEW);
    lookat(0, 0, 1, 0, 0, 0, 0, 1, 0);

    glPushMatrix();
        glTranslatef(---, ---, ---);
        glRotatef(---, ---, ---, ---);
        glScalef(---, ---, ---);

    glPushMatrix();
        glColor3f(0, 0, 0);
        glTranslatef(---, ---, ---);
        glScalef(---, ---, ---);
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        rect();
    glPopMatrix();

    glPushMatrix();
        glColor3f(0, 0, 0);
        glTranslatef(---, ---, ---);
        glRotatef(---, ---, ---, ---);
        glScalef(---, ---, ---);
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        rect();
    glPopMatrix();

    glPopMatrix();

    glPushMatrix();
        axis();
    glPopMatrix();

}
```

Camera, projeção e visualização

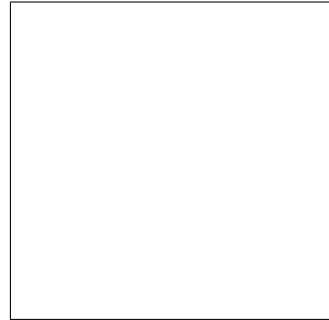
Q2 (4 valores)

Para cada trecho de código apresentado (Openframeworks/OpenGL), faça um esboço do desenho realizado pela função *axis()*, identificando o “nome” e sentido de cada eixo (x,y,z) quando visíveis. A aplicação foi configurada com uma janela de $(w, h) = (1024, 1024)$ pixels.

(observação: a função *axis()* é a mesma utilizada no exercício 1.)

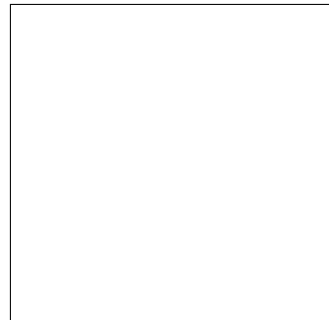
(a)(1 valor):

```
void draw(){
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1, 1, -1, 1, -2, 2);
    lookat(0, 0, 1, 0, 0, 0, 0, -1, 0);
    axis();
}
```



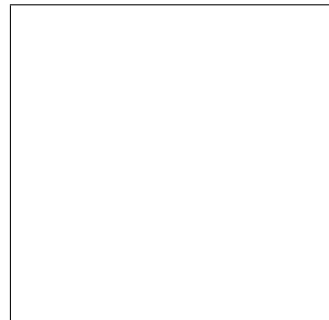
(b)(1 valor):

```
void draw(){
    glViewport(0, 0, w*0.5, h*0.5);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(1, -1, -1, 1, -2, 2);
    lookat(0, 0, -1, 0, 0, 0, 0, -1, 0);
    axis();
}
```



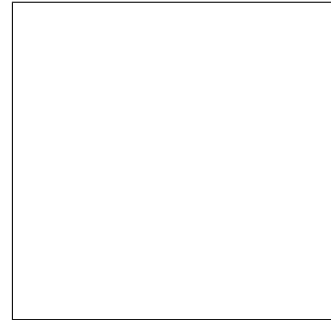
(c)(1 valor):

```
void draw(){
    glViewport(w*0.5, 0, w*0.5, h*0.5);
    perspective(60, 100, 1000);
    lookat(0, h*0.5/tan(PI/6.), 0, 0, 0, 0, 0, 0, -1);
    glScalef(w, w, w);
    axis();
}
```



(d)(1 valor):

```
void draw(){
    glViewport(0, 0, w, h);
    perspective(60, 100, 1000);
    lookat(w*0.5, h*0.5, w*0.5, 0, 0, 0, 0, 1, 0);
    glScalef(w, w, w);
    axis();
}
```



Textura

Q3 (8 valores)

Complete o pseudo-código com as coordenadas de textura e configuração adequada para obter os resultados conforme as imagens.

observação 1: os parâmetros de configuração podem ser em pseudo-código, mas devem ser claros e coerentes com as configurações reais possíveis.

observação 2: A imagem está em espaço de coordenadas de textura com dimensão normalizada, eixo t orientado para baixo, eixo s orientado para a direita e origem no topo-esquerdo da imagem.

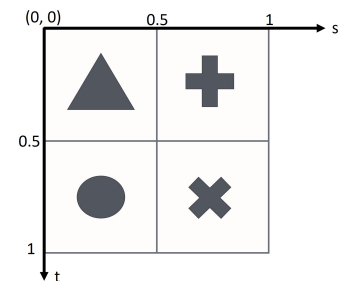
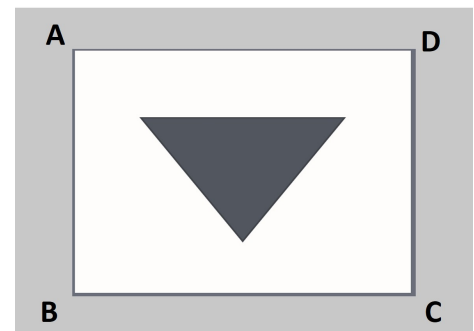


Figura 1: imagem original

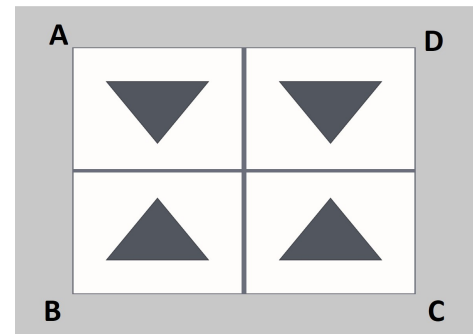
(a)(2 valores):

```
texParameter(GL_TEXTURE_WRAP_S, -----);
texParameter(GL_TEXTURE_WRAP_T, -----);
glBegin(GL_QUADS);
texCoord(---, ---); vertex_A(-0.5, 0.5, 0);
texCoord(---, ---); vertex_B(-0.5, -0.5, 0);
texCoord(---, ---); vertex_C(0.5, -0.5, 0);
texCoord(---, ---); vertex_D(0.5, 0.5, 0);
glEnd();
```



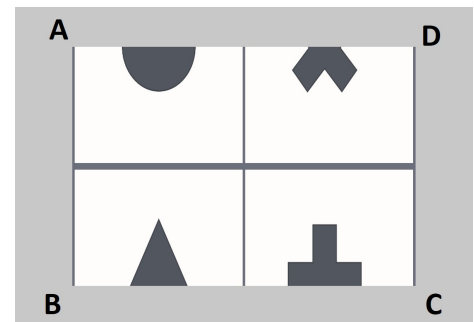
(b)(2 valores):

```
texParameter(GL_TEXTURE_WRAP_S, -----);  
texParameter(GL_TEXTURE_WRAP_T, -----);  
glBegin(GL_QUADS);  
texCoord(---, ---); vertex_A(-0.5, 0.5, 0);  
texCoord(---, ---); vertex_B(-0.5, -0.5, 0);  
texCoord(---, ---); vertex_C(0.5, -0.5, 0);  
texCoord(---, ---); vertex_D(0.5, 0.5, 0);  
glEnd();
```



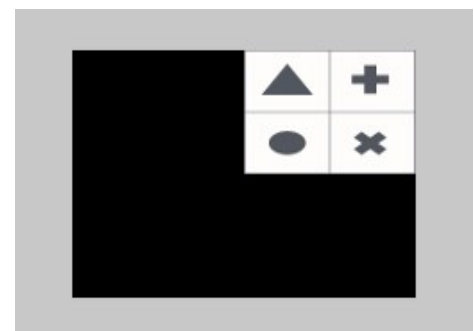
(c)(2 valores):

```
texParameter(GL_TEXTURE_WRAP_S, -----);  
texParameter(GL_TEXTURE_WRAP_T, -----);  
glBegin(GL_QUADS);  
texCoord(---, ---); vertex_A(-0.5, 0.5, 0);  
texCoord(---, ---); vertex_B(-0.5, -0.5, 0);  
texCoord(---, ---); vertex_C(0.5, -0.5, 0);  
texCoord(---, ---); vertex_D(0.5, 0.5, 0);  
glEnd();
```



(d)(2 valores):

```
texParameter(GL_TEXTURE_WRAP_S, -----);  
texParameter(GL_TEXTURE_WRAP_T, -----);  
glBegin(GL_QUADS);  
texCoord(---, ---); vertex_A(-0.5, 0.5, 0);  
texCoord(---, ---); vertex_B(-0.5, -0.5, 0);  
texCoord(---, ---); vertex_C(0.5, -0.5, 0);  
texCoord(---, ---); vertex_D(0.5, 0.5, 0);  
glEnd();
```



1 Iluminação

Q4 (4 valores):

O código apresentado implementa uma aplicação Openframeworks/OpenGL onde é desenhado um retângulo e a cor final é calculada por iluminação. Utilize o modelo teórico de Phong para calcular a cor e intensidade de luz, em valores (R, G, B) normalizados ([0, 1]), no centro do retângulo (coordenadas mundo) (justifique sua resposta).

observação: a função utilizada para desenhar o retângulo é a função *rect()* utilizada no exercício 1.

```
void draw(){
    glViewport(0, 0, gw(), gh());
    perspective(60, 100, 1000);
    lookat(0, 0, gh()*0.5/tan(PI/6.), 0, 0, 0, 0, 1, 0);

    glShadeModel(GL_FLAT);
    glEnable(GL_LIGHTING);
    glEnable(GL_NORMALIZE);
    glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, true);

    GLfloat sceneAmb[] = { 0.5, 0.5, 0.5, 1. };
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, sceneAmb);

    GLfloat pos[] = { 0., 0.5, sqrt(3.) * 0.5, 0 };
    GLfloat amb[] = { 0, 0, 0, 1 };
    GLfloat dif[] = { 1, 1, 1, 1 };
    GLfloat spec[] = { 0, 0, 0, 1 };
    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, dif);
    glLightfv(GL_LIGHT0, GL_SPECULAR, spec);

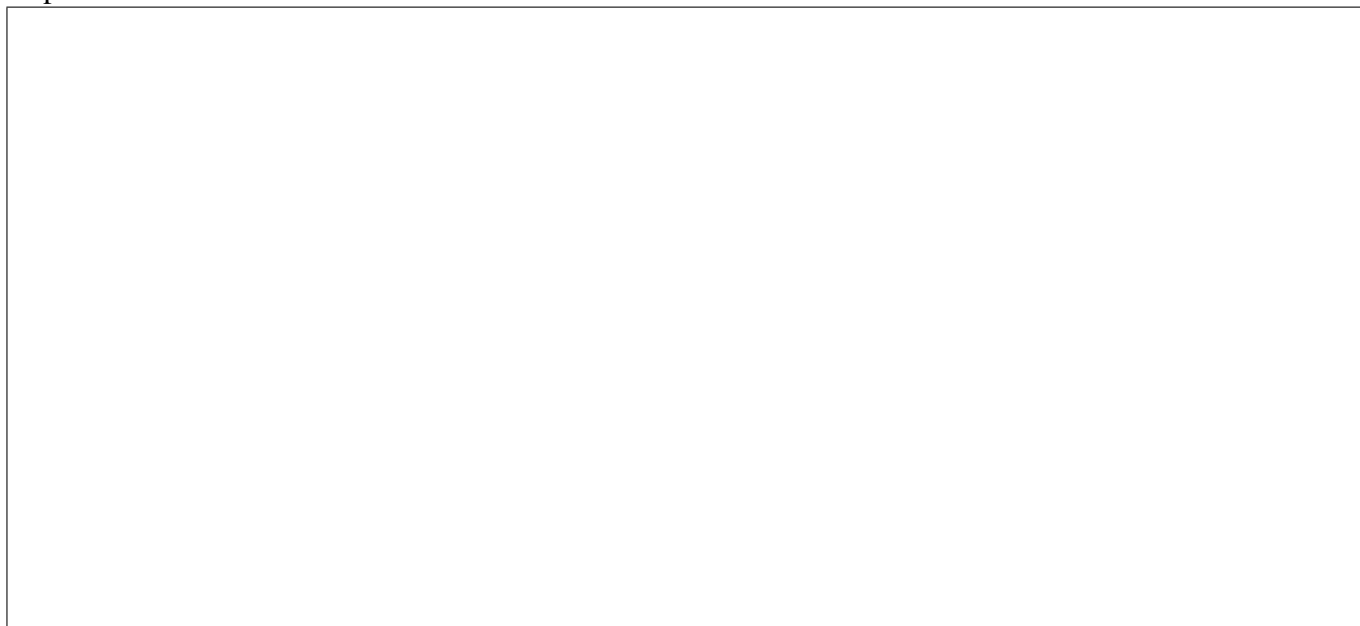
    glEnable(GL_LIGHT0);

    GLfloat matAmb[] = { 1, 0, 0, 1 };
    GLfloat matDif[] = { 0, 1, 0, 1 };
    GLfloat matSpec[] = { 0, 0, 1, 1 };
    GLfloat matCoef = 0.5 * 128;
    glMaterialfv(GL_FRONT, GL_AMBIENT, matAmb);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, matDif);
    glMaterialfv(GL_FRONT, GL_SPECULAR, matSpec);
    glMaterialf(GL_FRONT, GL_SHININESS, matCoef);

    glPushMatrix();
    glScalef(gw()*0.5, gh()*0.5, 1.);
    glNormal3f(0, 0, 1);
    rect();
    glPopMatrix();
}
```

responder na próxima página.

resp:

A large, empty rectangular box with a thin black border, intended for a response.