1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA

deis departamento de engenharia informática 1995 – 2020

LEI - Computação Gráfica

prof. André Perrotta, prof. Hugo Amaro

Exame Normal

Nome: _____

Número: _____

Duração: 120min | 28 de Janeiro, 2025 | 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:

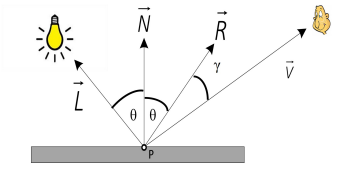
$$\vec{A} \times \vec{B} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = (a_2 b_3 - b_2 a_3)\hat{x} + (a_3 b_1 - b_3 a_1)\hat{y} + (a_1 b_2 - b_1 a_2)\hat{z}$$

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(θ)	0	$\frac{1}{2}$	$\frac{1}{\sqrt{2}}$	$\frac{\sqrt{3}}{2}$	1
cos(θ)	1	$\frac{\sqrt{3}}{2}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{2}$	0
tan(θ)	0	$\frac{1}{\sqrt{3}}$	1	$\sqrt{3}$	undefined

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

$I_{vertice} = I_{luz_{amb}} K_{mat_{amb}} + I_{luz_{dif}} K_{mat_{dif}} \cos\theta + I_{luz_{spec}} K_{mat_{spec}} \cos\gamma^{ns}$

$K_{mat_{amb}} = Vertex_{mat_{RGB}} * k_{amb}$

$K_{mat_{dif}} = Vertex_{mat_{RGB}} * k_{dif}$

$K_{mat_{spec}} = Vertex_{mat_{RGB}} * k_{spec}$

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();

//matriz coluna (colunated matrix)
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]);
}
```

Conceitos

Q1 (4 valores)

Em OpenGL, um modelo 3D é determinado por uma sequencia de informações relativas aos vértices que formam o modelo e a indicação de uma primitiva de desenho. Sobre a construção do modelo, responda às seguintes perguntas de forma clara, sucinta e objetiva:

a) (1 valor) Quais informações podem ser atribuídas aos vértices?

b) (1 valor) O que é determinado pela primitiva de desenho?

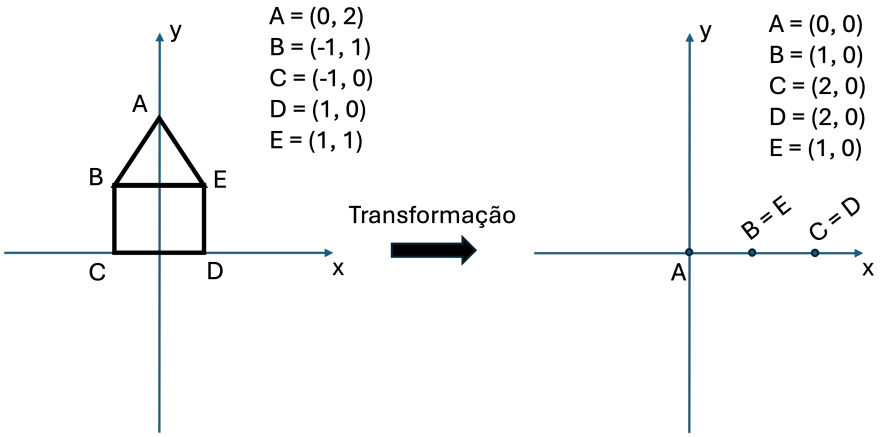
c) (1 valor) O que é definido pela ordem em que os vértices são ordenados?

d) (1 valor) Visto que a ordem dos vértices (”certa” ou ”errada”) não implica em erros de compilação, que recursos de OpenGL o desenvolvedor pode utilizar para avaliar se criou um modelo 3D com a ordem correta dos vértices?

Geometria e transformações

Q2 (4 valores)

Determine as transformações geométricas necessárias para transformar a forma $ABCDE$, conforme mostra a figura.



(a) (1 valor): Indicação das transformações com valores e ordem correta.

(ex: Translação de 10 em x pode ser escrita como T(10, 0))

(b) (2 valores): Representação da matriz final (M) resultante das transformações.

(c) (1 valor): Seria possível determinar uma matriz de transformação que retorne a forma para a condição inicial? Indique esta matriz de transformação ou justifique a impossibilidade.

Visualização, projeção e recorte

Q5 (4 valores)

Considere uma cena desenhada em Openframeworks/OpenGL, configurada para uma janela de 800x800 pixels. A cena está programada para desenhar um cubo unitário (origem em seu centro e lados de tamanho 1 unidade fig:1) e permitir a sua visualização ortográfica e perspectiva, de qualquer ponto de vista. O loop draw() está inicialmente configurado conforme o código a seguir, resultando na imagem mostrada na figura 2.

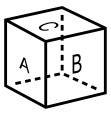


Figura 1: cubo unitário

```
void ofApp::draw(){
        glViewport(0, 0, 800, 800);
        glMatrixMode(GL_PERSPECTIVE);
        glLoadIdentity();
        glOrtho(-1, 1, -1, 1, -10, 10);
        lookat(0, 0, 5, 0, 0, 0, 0, 1, 0);
        glColor3f(0, 0, 0);
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        glPushMatrix();
        glScalef(sqrt(2), sqrt(2), sqrt(2));
        cube_unit();
        glPopMatrix();
}
```

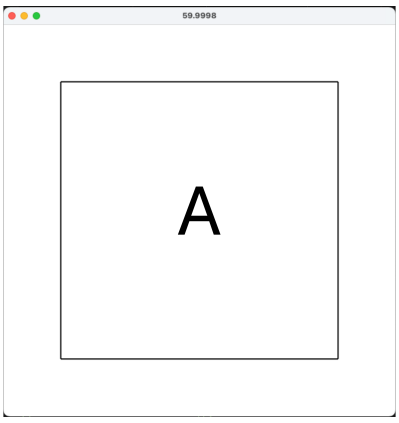


Figura 2: vista ortográfica frontal

(a)(1 valor) Como devemos configurar a função $lookat(\vec{pos}, \vec{target}, \vec{up})$ para obter a vista apresentada na figura 3?

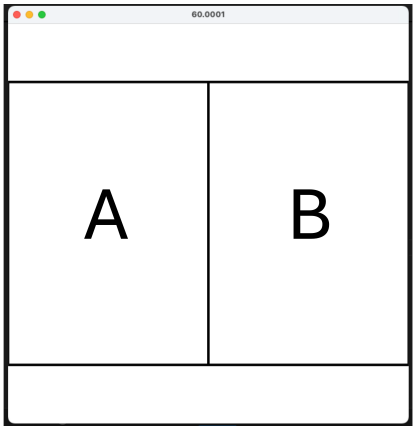


Figura 3

resp:

(b)(2 valor) Se quiséssemos substituir a função lookat(...) configurada na alínea anterior por uma matriz de transformação, como seria esta matriz?(justifique e defina a matriz com valores reais)

(c)(2 valor) Agora, substituímos a função glOrtho(...) pela função $perspective(90, 1000, 1000)$. Como devemos configurar a função $lookat(\vec{pos}, \vec{target}, \vec{up})$ para visualizar a face de cima (C) do cubo, de forma que ela fique centrada na tela e com tamanho de exatamente 200x200 pixels (figura 4) (justifique)?

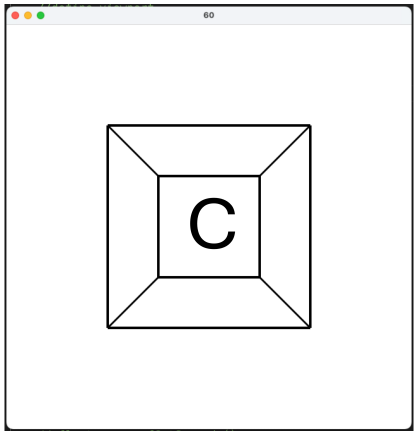


Figura 4

resp:

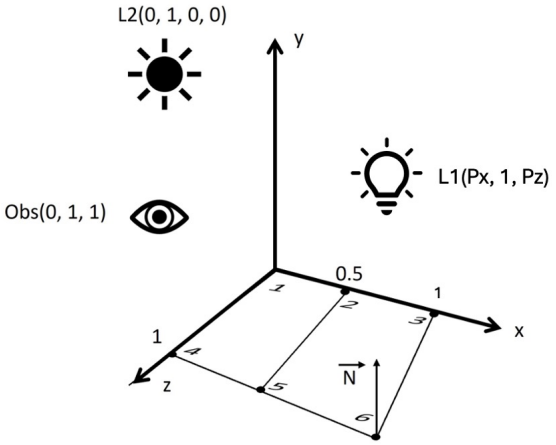
Iluminação

Q6 (4 valores)

A imagem abaixo mostra uma cena realizada em OpenGl formada por um malha de retângulos de resolução (2, 1), duas fontes de luz, L_1 e L_2, e um observador situado na posição $obs(0, 1, 1)$. O programa foi configurado para utilizar cores como materiais e a malha foi construida de forma a atribuir diferentes cores para cada vértice (ver tabela de cores). A normal $\vec{N}(0, 1, 0)$ é a mesma em todos os vértices. As fontes de luz estão configuradas conforme indicado, porém L1 pode movimentar-se livremente no plano $y = 1$.

$L_{1_{pos}} = (P_x, 1, P_z, 1)$
$L_{1_{amb}}(R, G, B) = (0, 0, 0)$
$L_{1_{dif}}(R, G, B) = (1, 0, 0)$
$L_{1_{spec}}(R, G, B) = (0, 1, 0)$

$L_{2_{pos}} = (0, 1, 0, 0)$
$L_{2_{amb}}(R, G, B) = (0, 0, 0)$
$L_{2_{dif}}(R, G, B) = (0, 0, 1)$
$L_{2_{spec}}(R, G, B) = (0, 1, 0)$



vertice	RGB
1	(1, 0, 0)
2	(1, 1, 0)
3	(1, 0, 1)
4	(0, 0, 1)
5	(0, 1, 1)
6	(1, 1, 1)

a)(1.5 valores) Qual deve ser uma possível posição de L1 para que o vértice 3 tenha sua cor final RGB = (0.5, 0, 1)? (justifique sua resposta)

b)(1.5 valores) Se fixarmos L1 na posição (0, 1, 0), qual será a cor final RGB do vértice 6? (justifique sua resposta)

Textura

Q3(4 valores):

Complete o pseudo-código com as coordenadas de textura e configuração adequada para obter os resultados conforme as imagens.
obs 1: os parâmetros de configuração podem ser em pseudo-código, mas devem ser claros e coerentes com as configurações reais possíveis.
obs 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.

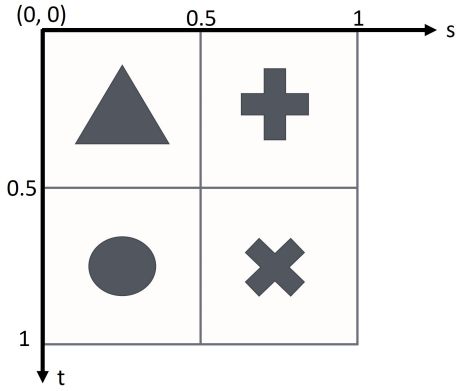
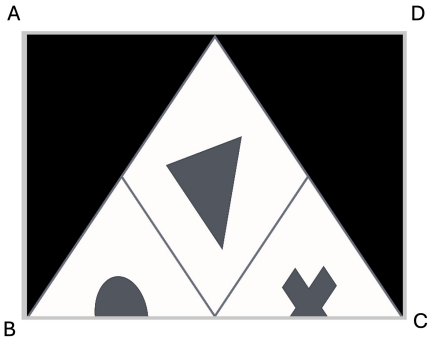


Imagem original

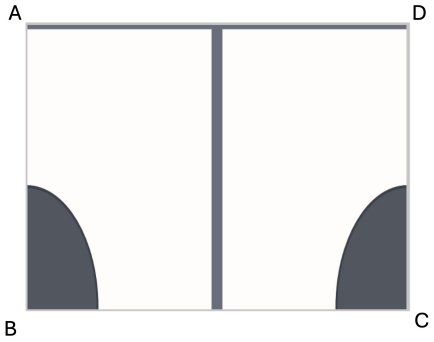
(a)(1 valor)

```
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();
```



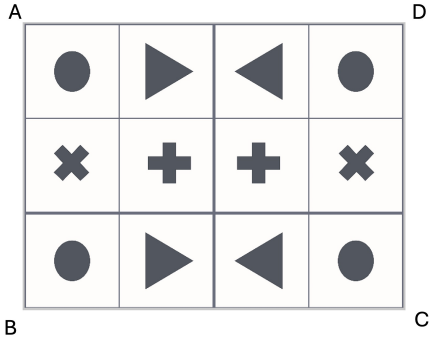
(b)(1 valor)

```
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)(1 valor)

```
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)(1 valor)

```
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();
```