Computação Gráfica I - MAB122 (2020-1)
Professor: João Vitor de Oliveira Silva
Lista

Você deve fazer 5 das 7 questões a seguir. Pode também ser realizado em duplas.

1. Considere os seguintes algoritmos:

```
Algoritmo 1:
Entrada: IMAGE imagem[width, height], SCENE cena, CAMERA cam
Variável: RAY raio, OBJECT3D objeto, OBJECT3D obj_closest, FLOAT t_hit,
          FLOAT t_closest, PIXEL pixel
início
    para cada pixel em imagem faça
        raio = cam.GENERATERAYTO(pixel.x, pixel.y)
        t_closest = +∞
        obj_closest = null
        para cada objeto em cena faça
            se raio.INTERSECT(objeto) então
                t_hit = raio.GETINTERSECTIONHITTIME()
                se t_hit < t_closest e t_hit >= 0 então
                    t_closest = t_hit
                    obj_closest = objeto
                fim
            fim
        fim
        se obj_closest ≠ null então
            imagem[pixel.y, pixel.x] = COMPUTECOLOR(objeto, t_closest)
        fim
    fim
fim
```

Algoritmo 2:

```
Entrada: IMAGE imagem[width, height], SCENE cena, CAMERA cam
Variável: OBJECT3D objeto, PIXEL pixel, OBJECT2D objeto_proj
início
    para cada pixel em imagem faça
        depth[pixel.y, pixel.x] = +∞
    fim
    para cada objeto em cena faça
        objeto_proj = cam.PROJECT(objeto)
        para cada pixel em imagem faça
            se pixel.inside(objeto_proj ) então
                imagem[pixel.y, pixel.x] = COMPUTERCOLOR(objeto_proj, pixel,
                  depth)
            fim
        fim
    fim
fim
```

Estes algoritmos são versões bastante simplificadas de algoritmos famosos de renderização. Nomeie cada um deles, explique o funcionamento de ambos e indique suas diferenças.

2. Um filtro de imagens famoso, chamado de máscara de nitidez (em inglês, *unsharp masking*), tem um efeito contrário ao de borramento. O mesmo torna a obra mais nitida, realçando as bordas. Seu cálculo segue o seguinte princípio:

   (i) Aplique um filtro de borramento sobre sua imagem I, obtendo uma imagem I_{blur}:
   $$I_{\text{blur}} = G * I$$
   Aqui, vamos considerar G como o filtro Gaussiano 3×3, ou seja
   $$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

   (ii) Subtraia a imagem borrada I_{blur} da imagem original I. O resultado dessa diferença é chamada da máscara.
   $$I_{\text{mask}} = I - I_{\text{blur}}$$

   (iii) Adicione a máscara sobre a imagem original (com um peso $0 \leq \alpha \leq 1$):
   $$I_{\text{sharp}} = I + \alpha I_{\text{mask}}$$

Um exemplo de aplicação deste filtro pode ser configurado na Figura 1.

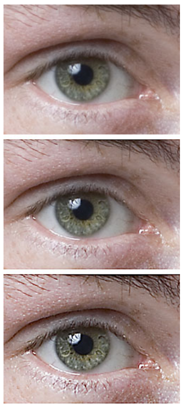
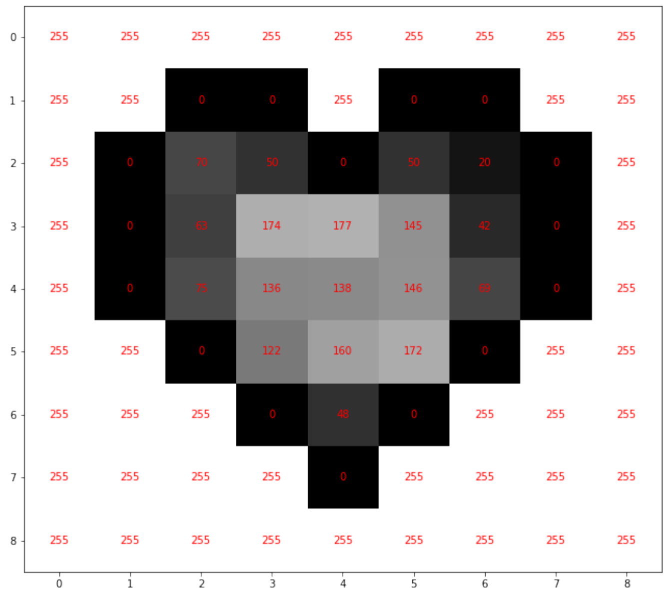


Figura 1: Exemplo do filtro máscara de nitidez: no topo a imagem original, no meio o resultado para um α_1, na parte de baixo o resultado para $\alpha_2 > \alpha_1$.

A respeito desse filtro, pede-se que faça:

a) Usando $\alpha = \frac{1}{4}$, aplique este filtro na imagem a seguir:



Há um arquivo `.csv` *no Google Classroom com os dados da imagem.*

b) Deduza a matriz S_α associada a este filtro, ou seja, que faz diretamente

$$I_{\text{sharp}} = S_\alpha * I$$

3. Considere o braço mecânico tridimensional abaixo, composto de três partes: antebraço, braço e mão. Usando transformações locais e mudança de referenciais, determine as coordenadas do ponto **p** no referencial global.

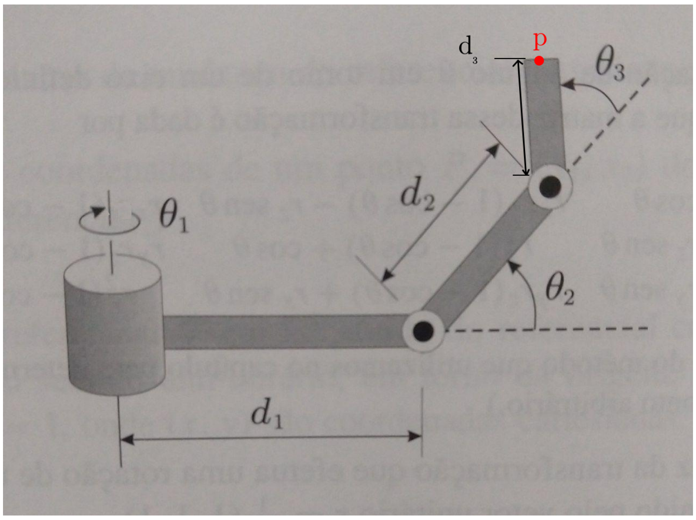


Figura 2: Braço mecânico tridimensional.

4. No ThreeJS, todos os objetos são descritos por uma `position`, um `quaternion` e uma `scale`. A cada transformação que um objeto sofre, o mesmo executa um método `updateMatrix` que atualiza a matriz de transformação do objeto com um método chamado `compose`. Veja abaixo a implementação em código deste método:

```
compose( position, quaternion, scale ) {
        // te is an array containing the matrix values
        const te = this.elements;
        // getting some values
        const x = quaternion._x, y = quaternion._y, z = quaternion._z;
        const w = quaternion._w;
        const x2 = x + x, y2 = y + y, z2 = z + z;
        const xx = x * x2, xy = x * y2, xz = x * z2;
        const yy = y * y2, yz = y * z2, zz = z * z2;
        const wx = w * x2, wy = w * y2, wz = w * z2;
        const sx = scale.x, sy = scale.y, sz = scale.z;
        // Assigning the first column
        te[ 0 ] = ( 1 - ( yy + zz ) ) * sx;
        te[ 1 ] = ( xy + wz ) * sx;
        te[ 2 ] = ( xz - wy ) * sx;
        te[ 3 ] = 0;
        // Assigning the second column
        te[ 4 ] = ( xy - wz ) * sy;
```

```
        te[ 5 ] = ( 1 - ( xx + zz ) ) * sy;
        te[ 6 ] = ( yz + wx ) * sy;
        te[ 7 ] = 0;
        // Assigning the third column
        te[ 8 ] = ( xz + wy ) * sz;
        te[ 9 ] = ( yz - wx ) * sz;
        te[ 10 ] = ( 1 - ( xx + yy ) ) * sz;
        te[ 11 ] = 0;
        // Assigning the fourth column
        te[ 12 ] = position.x;
        te[ 13 ] = position.y;
        te[ 14 ] = position.z;
        te[ 15 ] = 1;
}
```

Explique o funcionamento dessa função, destacando o significado das atribuições realizadas. *Lembre-se que internamente o* ThreeJS *armazena as matrizes de forma column-major.*

5. Nessa questão, iremos realizar todas as etapas estudadas do pipeline gráfico. Suponha que temos o ponto $\mathbf{q}_{local} = [1, 2, 0, 1]^T$ em coordenadas locais.

   a) Obtenha as coordenadas globais $\mathbf{q}_{global}$, considerando que a *model matrix* é dada pela sequência das seguintes transformações:

      i. Escalonamento em x de 2 unidades.

      ii. Rotação em torno do eixo y de 30^o.

      iii. Reflexão em torno do espelho $\mathbf{n} = [0, 1, 0]^T$.

      iv. Rotação em torno do eixo x de 45^o.

   b) Obtenha as coordenadas no referencial da câmera do ponto, ou seja, $\mathbf{q}_{camera}$. A câmera virtual está posicionada em $\mathbf{p}_{from} = [5, 10, 5, 1]^T$, apontada para $\mathbf{p}_{to} = [0, 0, 0, 1]^T$.

   c) Encontre as coordenadas no *clipping space* do ponto $\mathbf{q}_{clip}$, usando que a câmera virtual realiza projeção perspectiva com os seguintes parâmetros:

      - fov: 90^o
      - aspect: $\frac{16}{9}$
      - near: 1
      - far: 50

      Diga também se o ponto é visível ou não pela câmera.

   d) Caso o ponto seja visível, realize a etapa de divisão perspectiva, encontrando as coordenadas do ponto no espaço normalizado $\mathbf{q}_{ndc}$.

e) Caso o ponto seja visível, usando que a largura w é igual a 960 e que a altura h é igual a 540, obtenha as coordenas do ponto no espaço de tela (*screen space*). Para realizar este procedimento, faça

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} \frac{w}{2}(1 + x_{ndc}) \\ \frac{h}{2}(1 + y_{ndc}) \end{bmatrix}.$$

6. Considere um caminho (*path*) descrito por 2 segmentos de Bézier cúbicos bidimensionais. Os pontos de controle são $\mathbf{p}_1 = [10, -6, 1]$, $\mathbf{p}_2 = [4, -10, 1]$, $\mathbf{p}_3 = [5, -2, 1]$, $\mathbf{p}_4 = [10, 0, 1]$, $\mathbf{p}_5 = [15, -2, 1]$, $\mathbf{p}_6 = [16, -10, 1]$, $\mathbf{p}_7 = \mathbf{p}_1$.

   a) Usando o algoritmo de Casteljau, indique os ponto associados a $t = 0.3$ no primeiro segmento e $t = 0.6$ no segundo segmento.

   b) Faça um esboço deste caminho Bézier (ou gere um gráfico em seu computador).

7. Considere um cilindro centrado no eixo z, de altura h e raio r. Suponha que desejamos mapear uma textura de dimensões 256×256 quatro vezes ao longo do cilindro, como indicado na Figura 3:

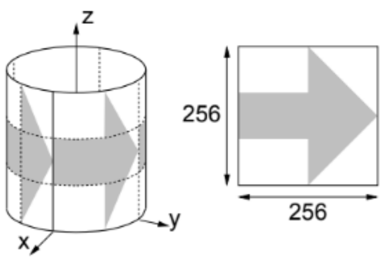


Figura 3: Cilindro e textura.

Pede-se que faça:

   a) Escreva uma parametrização do cilindro. Não esqueça de indicar os intervalos dos parâmetros (u, v).

   b) Encontre o mapeamento paramétrico inverso.

   c) Encontre a relação entre as coordenadas (u, v) e as coordenadas normalizadas de textura (s, t).

   d) Sabendo (s, t), indique como obter o texel correspondente desta imagem 256×256.