Instituto Federal de Brasília
Campus Taguatinga

Lista de Exercícios 3 (11/11/2024)

Computação Gráfica - 2024/2
Dr. Prof. Raimundo C. S. Vasconcelos

Tales Lima de Oliveira
tales.oliveira@estudante.ifb.edu.br

1. Cite aplicações de Computação Gráfica.

- **Entretenimento e Mídia**: Criação de imagens e animações realistas para filmes, jogos e outras formas de mídia visual.
- **Design e Engenharia**: Desenvolvimento de protótipos e modelos digitais para produtos, arquitetura e engenharia.
- **Medicina**: Produção de imagens médicas para diagnósticos e simulações de procedimentos médicos.
- **Realidade Virtual e Aumentada**: Construção de ambientes imersivos e interativos para uso em simulações, treinamento e entretenimento.
- **Visão Computacional**: Análise e interpretação de imagens para reconhecimento de padrões e automação.
- **Ciência e Pesquisa**: Visualização de dados complexos em áreas como física, química e meteorologia, facilitando a análise e compreensão.

2. Qual a diferença entre resolução e quantização?

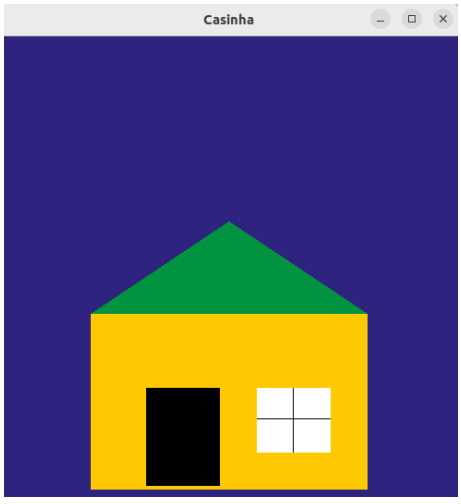
- **Resolução**: Refere-se ao número de pixels que compõem a imagem. Em geral, é expressa como o produto da quantidade de pixels na largura e na altura da imagem, como em 1920×1080 pixels. Quanto maior a resolução, mais detalhes a imagem pode exibir, pois há um número maior de pontos (pixels) para representar o conteúdo visual.
- **Quantização**: Está relacionada à profundidade de cor de cada pixel, ou seja, ao número de níveis de cor que cada pixel pode representar. Esse número depende da quantidade de bits atribuídos para armazenar cada componente de cor. Por exemplo, com 8 bits por canal (vermelho, verde e azul) em um sistema RGB, temos $2^8 = 256$ níveis de intensidade para cada cor, resultando em $256^3 = 16.777.216$ combinações de cores possíveis para cada pixel. Uma quantização baixa limita o número de cores, o que pode causar a presença de *banding* (faixas de cor) e a perda de suavidade nas transições de cor.

Portanto, a **resolução** está ligada ao detalhamento espacial da imagem (tamanho), enquanto a **quantização** está relacionada ao detalhamento de cor de cada pixel.

3. Quais são as principais áreas da Computação Gráfica?

- **Renderização**: Estudo e desenvolvimento de algoritmos para geração de imagens a partir de descrições geométricas. Envolve técnicas como *Ray Tracing* (traçado de raios) e *Rasterização*.
- **Modelagem Geométrica**: Focada na criação e representação de objetos em 3D. Inclui métodos para construção de formas, curvas e superfícies, além de manipulação de malhas (*meshes*).
- **Animação**: Área que se ocupa da criação de movimentos realistas para personagens, objetos e cenários. Envolve a aplicação de técnicas como cinemática, dinâmica, e simulação física.
- **Processamento de Imagens**: Concentra-se na manipulação e melhoria de imagens já existentes, incluindo técnicas de filtragem, reconhecimento de padrões e compressão de imagens.
- **Realidade Virtual e Aumentada**: Utiliza técnicas de computação gráfica para criar ambientes imersivos e interativos, seja para simulação de cenários virtuais ou sobreposição de informações no mundo real.
- **Interação e Interfaces Gráficas (UI/UX)**: Desenvolvimento de interfaces visuais e interativas para usuários, incluindo design e experiência de uso.
- **Visualização de Dados**: Focada na representação gráfica de informações e dados complexos para facilitar a compreensão e análise visual.
- **Computação Gráfica em Tempo Real**: Utilizada em áreas como jogos e simulações, onde é necessário gerar gráficos em tempo real, garantindo uma experiência visual fluida e responsiva.

4. Escreva um programa para desenhar a casinha abaixo.



Código 4: Criando a casa - casinha4.c

```c
#include <GL/glut.h>
#include <stdlib.h>

int tx = 0, ty = 0, angulo = 0, ex = 1, ey = 1;

```

```
6  // Função para configurar a cor de fundo
7  void Inicializa(void) {
8      // Cor de fundo azul
9      glClearColor(0.18f, 0.14f, 0.5f, 1.0f);
10     glOrtho(0,10,0,10,-1,1);
11
12 }
13
14 // Função chamada quando o tamanho da janela é alterado
15 void AlteraTamanhoJanela(GLsizei w, GLsizei h) {
16     if (h == 0) h = 1; // Evita divisão por zero
17
18     // Define as dimensões da área de visualização
19     glViewport(0, 0, w, h);
20
21     // Configura a projeção p/ manter a proporção
22     glMatrixMode(GL_PROJECTION);
23     glLoadIdentity();
24     if (w <= h)
25         gluOrtho2D(0.0f, 250.0f, 0.0f, 250.0f * h / w);
26     else
27         gluOrtho2D(0.0f, 250.0f * w / h, 0.0f, 250.0f);
28 }
29
30 // Função de desenho
31 void Desenha(void) {
32     // Limpa a tela com a cor de fundo
33     glClear(GL_COLOR_BUFFER_BIT);
34
35     // Transformações de translação, escala e rotação
36     glMatrixMode(GL_MODELVIEW);
37     glLoadIdentity();
38     glTranslatef(tx, ty, 0);
39     glScalef(ex, ey, 1);
40     glRotatef(angulo, 0, 0, 1);
41
42
43     // ROOF
44     glBegin(GL_TRIANGLES);
45         glColor3f(0.0f, 0.58f, 0.25f);  // GREEN
46         glVertex2i(50, 100);            // TOP LEFT HOUSE
47         glVertex2i(125, 150);           // MIDDLE ROOF
48         glVertex2i(200, 100);           // TOP RIGHT HOUSE
49     glEnd();
50
51     // HOUSE
52     glBegin(GL_QUADS);
53         glColor3f(1.0f, 0.79f, 0.0f);   // YELLOW
54         glVertex2i(50, 5);              // BOT LEFT
55         glVertex2i(50, 100);            // TOP LEFT
56         glVertex2i(200, 100);           // TOP RIGHT
57         glVertex2i(200, 5);             // BOT RIGHT
58     glEnd();
59
60
61     // DOOR
```

```
62      glBegin(GL_QUADS);
63          glColor3f(0.0f, 0.0f, 0.0f);    // BLACK
64          glVertex2i(80, 7);              // BOT LEFT
65          glVertex2i(80, 60);             // TOP LEFT
66          glVertex2i(120, 60);            // TOP RIGHT
67          glVertex2i(120, 7);             // BOT RIGHT
68      glEnd();
69
70      // WINDOWS
71      glBegin(GL_QUADS);
72          glColor3f(1.0f, 1.0f, 1.0f);    // WHITE
73          glVertex2i(140, 25);            // BOT LEFT
74          glVertex2i(140, 60);            // TOP LEFT
75          glVertex2i(180, 60);            // TOP RIGHT
76          glVertex2i(180, 25);            // BOT RIGHT
77      glEnd();
78
79      // BARS
80      glBegin(GL_LINES);
81          glColor3f(0.0f, 0.0f, 0.0f);    // BLACK
82          glVertex2i(160, 60);            // H TOP
83          glVertex2i(160, 25);            // H BOT
84      glEnd();
85      glBegin(GL_LINES);
86          glVertex2i(140, 43);            // V LEFT
87          glVertex2i(180, 43);            // V RIGHT
88      glEnd();
89
90      // Processa os comandos OpenGL
91      glFlush();
92  }
93
94
95  // Função principal
96  int main(int argc, char **argv) {
97      // Inicializa o ambiente GLUT e configura a janela
98      glutInit(&argc, argv);
99      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
100     glutInitWindowSize(500, 500);
101     glutInitWindowPosition(25, 25);
102     glutCreateWindow("Casinha");
103
104     // Registra as funções de callback
105     // Função de desenho
106     glutDisplayFunc(Desenha);
107     // Função de redimensionamento da janela
108     glutReshapeFunc(AlteraTamanhoJanela);
109
110     // Configura a cor de fundo e inicia o loop principal
111     Inicializa();
112     glutMainLoop();
113
114     return 0;
115 }
```

5. Crie um programa para desenhar a casinha apresentada no exercício anterior, mas possibilitando que o ponto superior do telhado seja informado pelo usuário via chamada do programa.

Código 5: Adicionando `user_input` ao código - casinha5.c

```c
#include <GL/glut.h>
#include <stdlib.h>

// Declarar como global
int user_input = 0;


// Função de desenho
void Desenha() {
    // ... codigo ...

    // ROOF
    glBegin(GL_TRIANGLES);
        glColor3f(0.0f, 0.58f, 0.25f);      // GREEN
        glVertex2i(50, 100);                // TOP LEFT HOUSE
        glVertex2i(125, 110 + user_input);  // MIDDLE ROOF
        glVertex2i(200, 100);               // TOP RIGHT HOUSE
    glEnd();

    // ... codigo ...
}


// Função principal
int main(int argc, char **argv) {

    // Verifica se tem argumento de chamada
    if(argc != 2) return 1;
    user_input = atoi(argv[1]);

    // ... codigo ...
}
```

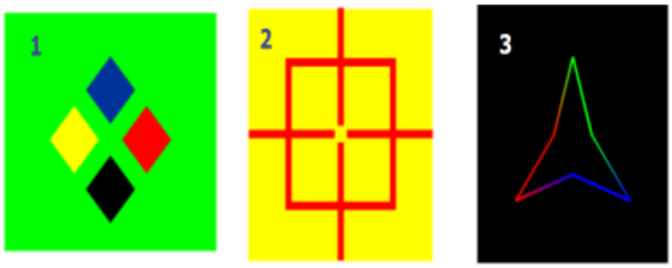
6. Rotacione a casa.

Código 6: Teclas para rotação, escala e translação - casinha6.c

`PAGE_UP` = Rota para direita.
`PAGE_DOWN` = Rota para esquerda.
`HOME` = Aumenta a escala no eixo X.
`END` = Aumenta a escala no eixo Y.
`SETA_CIMA` = Move para cima.
`SETA_BAIXO` = Move para baixo.
`SETA_DIREITA` = Move para direita.
`SETA_ESQUEDA` = Move para esquerda.

```
// Função de teclas especiais
void TeclasEspeciais(int key, int x, int y) {
    switch (key) {
        case GLUT_KEY_PAGE_UP:    // Rota para a direita
            angulo += 1; break;
        case GLUT_KEY_PAGE_DOWN:  // Rota para a esquerda
            angulo -= 1; break;
        case GLUT_KEY_HOME:       // Aumenta a escala no eixo X
            ex += 1; break;
        case GLUT_KEY_END:        // Diminui a escala no eixo X
            ex -= 1; break;
        case GLUT_KEY_UP:         // Move para cima
            ty += 1; break;
        case GLUT_KEY_DOWN:       // Move para baixo
            ty -= 1; break;
        case GLUT_KEY_RIGHT:      // Move para a direita
            tx += 1; break;
        case GLUT_KEY_LEFT:       // Move para a esquerda
            tx -= 1; break;
    }

    // Redesenha a tela com as novas transformações
    glutPostRedisplay();
}

// Função principal
int main(int argc, char **argv) {
    // ... código ...

    // Função de teclas especiais
    glutSpecialFunc(TeclasEspeciais);

    Inicializa();
    glutMainLoop();

    return 0;
}
```

7. Desenvolva código usando OpenGL para desenhar as seguintes imagens.



Código 7.1: Imagem 1 - ouros71.c

```c
#include <GL/glut.h>
#include <stdlib.h>

int tx = 0, ty = 0, angulo = 0, ex = 1, ey = 1;

// INIT OPENGL WINDOW
void init() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);       // BLACK
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);   // Config ortogonal
}

// DRAW DIAMOND SUIT
void createDiamond(float x, float y) {
    glPushMatrix();
    glTranslatef(x, y, 0.0f);       // POSITION
    glBegin(GL_POLYGON);
        glVertex2f(0.0f, 0.5f);     // TOP
        glVertex2f(0.5f, 0.0f);     // RIGHT
        glVertex2f(0.0f, -0.5f);    // BOT
        glVertex2f(-0.5f, 0.0f);    // LEFT
    glEnd();
    glPopMatrix();
}

// DRAW SCENE
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // RESHAPE CONFIG
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(tx, ty, 0);
    glScalef(ex, ey, 1);
    glRotatef(angulo, 0, 0, 1);

    glColor3f(1.0f, 0.0f, 0.0f);    // RED
    createDiamond(0.0f, 0.6f);      // TOP

    glColor3f(0.0f, 1.0f, 0.0f);    // GREEN
    createDiamond(-0.5f, 0.0f);     // MID LEFT

    glColor3f(0.0f, 0.0f, 1.0f);    // BLUE
    createDiamond(0.5f, 0.0f);      // MID RIGHT
```

```
44
45      glColor3f(1.0f, 1.0f, 0.0f);     // YELLOW
46      createDiamond(0.0f, -0.6f);      // BOTTOM
47
48      glFlush();
49  }
50
51  // RESHAPE WINDOW
52  void reshape(GLsizei w, GLsizei h) {
53      if (h == 0) h = 1;
54      glViewport(0, 0, w, h);
55
56      glMatrixMode(GL_PROJECTION);
57      glLoadIdentity();
58      if (w <= h)
59          gluOrtho2D(0.0f, 250.0f, 0.0f, 250.0f * h / w);
60      else
61          gluOrtho2D(0.0f, 250.0f * w / h, 0.0f, 250.0f);
62  }
63
64
65
66  // MAIN FUNTION
67  int main(int argc, char** argv) {
68      glutInit(&argc, argv);
69      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
70      glutInitWindowSize(600, 600);
71      glutInitWindowPosition(25, 25);
72      glutCreateWindow("Losangos Coloridos - Naipe de Ouros");
73
74      glutDisplayFunc(display);
75      glutReshapeFunc(reshape);
76
77      init();
78      glutMainLoop();
79
80      return 0;
81  }
```

Código 7.2: Imagem 2 - crosshair72.c

```
1   #include <GL/glut.h>
2   #include <stdlib.h>
3
4   int tx = 0, ty = 0, angulo = 0, ex = 1, ey = 1;
5
6   void init() {
7       glClearColor(0.0, 0.0, 0.0, 1.0);  // BLACK
8   }
9
10  void display() {
11      glClear(GL_COLOR_BUFFER_BIT);
12
13      // RESHAPE CONFIG
14      glMatrixMode(GL_MODELVIEW);
```

```
15      glLoadIdentity();
16      glTranslatef(tx, ty, 0);
17      glScalef(ex, ey, 1);
18      glRotatef(angulo, 0, 0, 1);
19
20      glColor3f(1.0, 1.0, 1.0);     // WHITE
21
22      // CENTER SQUARE
23      float squareSize = 0.2f;
24      glBegin(GL_LINE_LOOP);
25          glVertex2f(-squareSize, -squareSize);
26          glVertex2f( squareSize, -squareSize);
27          glVertex2f( squareSize,  squareSize);
28          glVertex2f(-squareSize,  squareSize);
29      glEnd();
30
31      // VERTICAL LINE
32      glBegin(GL_LINES);
33          glVertex2f(0.0, -0.5);          // BOT TO TOP, OUTSIDE SQUARE
34          glVertex2f(0.0, -squareSize);   // BOT SQUARE
35          glVertex2f(0.0, squareSize);    // TOP SQUARE
36          glVertex2f(0.0, 0.5);           // TOP TO BOT, OUTSIDE SQUARE
37      glEnd();
38
39      // HORIZONTAL LINE
40      glBegin(GL_LINES);
41          glVertex2f(-0.5, 0.0);          // LEFT TO RIGHT, OUTSIDE SQUARE
42          glVertex2f(-squareSize, 0.0);   // LEFT SQUARE
43          glVertex2f(squareSize, 0.0);    // RIGHT SQUARE
44          glVertex2f(0.5, 0.0);           // RIGHT TO LEFT, OUTSIDE SQUARE
45      glEnd();
46
47      glFlush();
48  }
49
50  // RESHAPE WINDOW
51  void reshape(GLsizei w, GLsizei h) {
52      if (h == 0) h = 1;
53      glViewport(0, 0, w, h);
54
55      glMatrixMode(GL_PROJECTION);
56      glLoadIdentity();
57      if (w <= h)
58          gluOrtho2D(0.0f, 250.0f, 0.0f, 250.0f * h / w);
59      else
60          gluOrtho2D(0.0f, 250.0f * w / h, 0.0f, 250.0f);
61  }
62
63
64  int main(int argc, char** argv) {
65      glutInit(&argc, argv);
66      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
67      glutInitWindowSize(500, 500);
68      glutInitWindowPosition(25, 25);
69      glutCreateWindow("Mira do aspas");
70
```

```
71      glutDisplayFunc(display);
72      glutReshapeFunc(reshape);
73
74      init();
75      glutMainLoop();
76
77      return 0;
78  }
```

Código 7.3: Imagem 3 - shuriken73.c

```
1   #include <GL/glut.h>
2   #include <stdlib.h>
3   #include <math.h>
4
5   int tx = 0, ty = 0, angulo = 0, ex = 1, ey = 1;
6
7   // OPENGL CONFIGS
8   void init() {
9       glClearColor(0.0, 0.0, 0.0, 1.0);  // BLACK
10      glMatrixMode(GL_PROJECTION);
11      glLoadIdentity();
12      gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
13  }
14
15  // DRAW SHURIKEN
16  void drawShuriken() {
17      glBegin(GL_LINE_LOOP);
18
19      // Definindo coordenadas dos vértices
20      GLfloat vertices[6][2] = {
21          {0.0, 0.5},         // Ponta superior (vermelho)
22          {-0.5, -0.3},       // Interior inferior esquerdo
23          {0.0, -0.1},        // Ponta inferior esquerda (verde)
24          {0.5, -0.3},        // Interior inferior direito
25          {0.0, -0.5},        // Ponta inferior direita (azul)
26          {-0.5, -0.3}        // Fechando o loop
27      };
28
29      // Definindo cores das pontas
30      GLfloat cores[3][3] = {
31          {1.0, 0.0, 0.0},  // Vermelho para a primeira ponta
32          {0.0, 1.0, 0.0},  // Verde para a segunda ponta
33          {0.0, 0.0, 1.0}   // Azul para a terceira ponta
34      };
35
36      // Desenha com gradiente
37      for (int i = 0; i < 6; i++) {
38          if (i % 2 == 0) {
39              glColor3f(cores[i / 2][0], cores[i / 2][1], cores[i / 2][2]);  //
                ↪  Alterna entre as cores
40          } else {
41              glColor3f(0.5, 0.5, 0.5);  // Cinza claro para os pontos internos
42          }
43          glVertex2f(vertices[i][0], vertices[i][1]);
```

```
    }

    glEnd();
}

// Função para desenhar a cena
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // RESHAPE CONFIG
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(tx, ty, 0);
    glScalef(ex, ey, 1);
    glRotatef(angulo, 0, 0, 1);

    drawShuriken();
    glFlush();
}


// RESHAPE WINDOW
void reshape(GLsizei w, GLsizei h) {
    if (h == 0) h = 1;
    glViewport(0, 0, w, h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        gluOrtho2D(0.0f, 250.0f, 0.0f, 250.0f * h / w);
    else
        gluOrtho2D(0.0f, 250.0f * w / h, 0.0f, 250.0f);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(25, 25);
    glutCreateWindow("Shuriken com Efeito Gradiente");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);

    init();
    glutMainLoop();
    return 0;
}
```