Instituto Federal de Brasília
Campus Taguatinga

Lista de Exercícios 4.1 (18/11/2024)

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

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

1. Parte I

Código 1: parte1.c

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

void initialize(void){
    glMatrixMode(GL_PROJECTION);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(15,15);
    glutCreateWindow("Tales Lima Oliveira");

    glutDisplayFunc(displayCallback);
    glutKeyboardFunc(keyboardCallback);
    glutSpecialFunc(specialKeysCallback);
    glutReshapeFunc(reshapeCallback);

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

Código 2.1: input.h

```c
#ifndef INPUT_H
#define INPUT_H

void keyboardCallback(unsigned char key, int x, int y);
void specialKeysCallback(int key, int x, int y);
void reshapeCallback(GLsizei w, GLsizei h);

#endif
```

Código 2.2: drawing.c

```c
#include <GL/glut.h>
#include "drawing.h"

double tx = 0, ty = 0, angulo = 0, px = 0, py = 0;
double left = -1.0, right = 1.0, bot = -1.0, top = 1.0;
double zoom = 50.0;

void drawHouse() {
    glBegin(GL_TRIANGLES);
        glColor3f(0, 0, 1); glVertex2f(-0.25f, 0.1f);
        glColor3f(1, 0, 0); glVertex2f(0.0, 0.25);
        glColor3f(0, 0, 1); glVertex2f(0.25f, 0.1f);
    glEnd();

    glColor3f(1.0f, 1.0f, 1.0f);
    glLineWidth(2);

    glBegin(GL_LINE_LOOP);
        glVertex2f(-0.25, -0.2);
        glVertex2f(-0.25,  0.1);
        glVertex2f( 0.25,  0.1);
        glVertex2f( 0.25, -0.2);
    glEnd();
}

void drawCross() {
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0);

    glColor3f(0, 0, 0);
    glLineWidth(1);

    glBegin(GL_LINES);
        glVertex2f(0, 1); glVertex2f(0, -1);
        glVertex2f(-1, 0); glVertex2f(1, 0);
    glEnd();
}

void displayCallback() {
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();

    glPushMatrix();
        gluPerspective(zoom, 1.0, 1.0, 100.0);
        gluOrtho2D(left+px, right+px, bot+py, top+py);
        glTranslatef(tx, ty, 0);
        glRotatef(angulo, 0, 0, 1);
        drawHouse();
    glPopMatrix();

    drawCross();
    glFlush();
}
```

Código 3.1: input.h

```h
#ifndef DRAWING_H
#define DRAWING_H

extern double tx, ty, angulo, px, py;
extern double left, right, bot, top;
extern double zoom;

void drawHouse();
void drawCross();
void displayCallback();

#endif
```

Código 3.2: input.c

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

// Define os limites para o movimento
const double moveLimitX = 1.0f;
const double moveLimitY = 1.0f;

// Limites do zoom
const double zoomLimitMin = 10.0f;
const double zoomLimitMax = 100.0f;

void keyboardCallback(unsigned char key, int x, int y) {
    switch (key) {
        case 27: // ESC
            exit(0);
            break;

        case 'q': // Movimento diagonal (superior esquerda)
            if (px + 0.25 < moveLimitX && py - 0.25 > -moveLimitY) {
                px += 0.25;
                py -= 0.25;
            }
            break;
        case 'a': // Movimento diagonal (inferior esquerda)
            if (px + 0.25 < moveLimitX && py + 0.25 < moveLimitY) {
                px += 0.25;
                py += 0.25;
            }
            break;
        case 'e': // Movimento diagonal (superior direita)
            if (px - 0.25 > -moveLimitX && py - 0.25 > -moveLimitY) {
                px -= 0.25;
                py -= 0.25;
            }
            break;
        case 'd': // Movimento diagonal (inferior direita)
```

```
            if (px - 0.25 > -moveLimitX && py + 0.25 < moveLimitY) {
                px -= 0.25;
                py += 0.25;
            }
            break;
    }
    glutPostRedisplay();
}

void specialKeysCallback(int key, int x, int y) {
    switch (key) {
        case GLUT_KEY_PAGE_UP: // Rotação para a esquerda
            angulo += 0.5;
            break;
        case GLUT_KEY_PAGE_DOWN: // Rotação para a direita
            angulo -= 0.5;
            break;

        case GLUT_KEY_UP: // Movimento para cima
            if (ty + 0.2 < moveLimitY) {
                ty += 0.2;
            }
            break;
        case GLUT_KEY_DOWN: // Movimento para baixo
            if (ty - 0.2 > -moveLimitY) {
                ty -= 0.2;
            }
            break;
        case GLUT_KEY_RIGHT: // Movimento para a direita
            if (tx + 0.2 < moveLimitX) {
                tx += 0.2;
            }
            break;
        case GLUT_KEY_LEFT: // Movimento para a esquerda
            if (tx - 0.2 > -moveLimitX) {
                tx -= 0.2;
            }
            break;

       case GLUT_KEY_HOME: // Zoom out
            zoom -= 5.0f;
            if (zoom < zoomLimitMin) zoom = zoomLimitMin;
            break;

        case GLUT_KEY_END: // Zoom in
            zoom += 5.0f;
            if (zoom > zoomLimitMax) zoom = zoomLimitMax;
            break;

        case GLUT_KEY_INSERT: // Reset para o centro
            tx = ty = px = py = angulo = 0;
            left = bot = -1.0;
            right = top = 1.0;
            break;
    }
    glutPostRedisplay();
```

```
94  }
95
96  void reshapeCallback(GLsizei w, GLsizei h) {
97      if (h == 0) h = 1;
98      glViewport(0, 0, w, h);
99      glLoadIdentity();
100
101     if (w <= h)
102         gluOrtho2D(left, right, bot, bot + (right - left) * h / w);
103     else
104         gluOrtho2D(left, left + (right - left) * w / h, bot, top);
105 }
```

2. Compilação e Execução do Código

Este projeto utiliza um `Makefile` para simplificar o processo de compilação, execução e limpeza dos programas.

- Para **compilar** todos os programas, utilize o comando:
  - `make all`
- Para **executar** os programas, utilize o comando:
  - `make run`
- Para a **limpeza** dos arquivos binarios, utilize o comando:
  - `make clean`