CIV-2802 – SISTEMAS GRÁFICOS PARA ENGENHARIA – 2011.1

Exercício sobre manipulação de visualização 3D com o OpenGL

Nome: ____

Leia o programa descrito nas próximas folhas e complete as linhas que estão faltando nos quadros abaixo.

(1)

(2)

(3)

(4)

(5)

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <stdlib.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include "iup.h"
#include "iupgl.h"

#define FRAMES 360
#define M_PI 3.141592654
#define MAX(a,b) (((a)>(b))?(a):(b))

/*
** ----------------------------------------------
** Variaveis:
*/
/* Objeto para o exemplo: um paralelepipedo com
 * lados iguais a 1, 2 e 3, e centrado em
 * (0.5, 1.0, 1.5).
 */
typedef struct _point {
 double x, y, z;
 } Point;
static Point obj[8] =
         { { 0.0, 0.0, 0.0 },
           { 1.0, 0.0, 0.0 },
           { 1.0, 2.0, 0.0 },
           { 0.0, 2.0, 0.0 },
           { 0.0, 0.0, 3.0 },
           { 1.0, 0.0, 3.0 },
           { 1.0, 2.0, 3.0 },
           { 0.0, 2.0, 3.0 } };

/* Caixa que envolve o objeto.
 */
static double xmin = 0.0, xmax = 1.0;
static double ymin = 0.0, ymax = 2.0;
static double zmin = 0.0, zmax = 3.0;

/* Definicao de cores.
 */
static double color[][3] =
         { { 0.0, 1.0, 1.0 }, /* CYAN */
           { 0.0, 1.0, 0.0 }, /* GREEN */
           { 1.0, 1.0, 0.0 }  /* YELLOW */
         };

enum {
      COLOR_CYAN,
      COLOR_GREEN,
      COLOR_YELLOW
    };

/* Parametros da camera.
 */
static double eyex, eyey, eyez;
static double refx, refy, refz;
static double vupx, vupy, vupz;

/* Angulo de rotacao e incremento.
 */
static double alpha = 0.0;
static double delta = 2*M_PI/FRAMES;

/* Handle para o canvas.
 */
static Ihandle *canvas;

/*
** ----------------------------------------------
** Funcoes:
*/
static void displaySolid( void );
static void adjustObj( int w, int h );
static int  rotateObj( void );
static int  redraw( Ihandle *self );
static int  resize( Ihandle *self, int w, int h
);

/* ============= displaySolid ============ */

static void displaySolid( void )
{
 glBegin( GL_QUADS );
 glColor3dv( color[COLOR_GREEN] );
 glNormal3d( -1.0, 0.0, 0.0 ); /* Face -x */
 glVertex3d( obj[0].x, obj[0].y, obj[0].z );
 glVertex3d( obj[4].x, obj[4].y, obj[4].z );
 glVertex3d( obj[7].x, obj[7].y, obj[7].z );
 glVertex3d( obj[3].x, obj[3].y, obj[3].z );

 glColor3dv( color[COLOR_CYAN] );
 glNormal3d( 0.0, -1.0, 0.0 ); /* Face -y */
 glVertex3d( obj[0].x, obj[0].y, obj[0].z );
 glVertex3d( obj[1].x, obj[1].y, obj[1].z );
 glVertex3d( obj[5].x, obj[5].y, obj[5].z );
 glVertex3d( obj[4].x, obj[4].y, obj[4].z );

 glColor3dv( color[COLOR_YELLOW] );
 glNormal3d( 0.0, 0.0, -1.0 ); /* Face -z */
 glVertex3d( obj[3].x, obj[3].y, obj[3].z );
 glVertex3d( obj[2].x, obj[2].y, obj[2].z );
 glVertex3d( obj[1].x, obj[1].y, obj[1].z );
 glVertex3d( obj[0].x, obj[0].y, obj[0].z );

 glColor3dv( color[COLOR_GREEN] );
 glNormal3d( 1.0, 0.0, 0.0 ); /* Face +x */
 glVertex3d( obj[1].x, obj[1].y, obj[1].z );
 glVertex3d( obj[2].x, obj[2].y, obj[2].z );
 glVertex3d( obj[6].x, obj[6].y, obj[6].z );
 glVertex3d( obj[5].x, obj[5].y, obj[5].z );

 glColor3dv( color[COLOR_CYAN] );
 glNormal3d( 0.0, 1.0, 0.0 ); /* Face +y */
 glVertex3d( obj[2].x, obj[2].y, obj[2].z );
 glVertex3d( obj[3].x, obj[3].y, obj[3].z );
 glVertex3d( obj[7].x, obj[7].y, obj[7].z );
 glVertex3d( obj[6].x, obj[6].y, obj[6].z );

 glColor3dv( color[COLOR_YELLOW] );
 glNormal3d( 0.0, 0.0, 1.0 ); /* Face +z */
 glVertex3d( obj[4].x, obj[4].y, obj[4].z );
 glVertex3d( obj[5].x, obj[5].y, obj[5].z );
 glVertex3d( obj[6].x, obj[6].y, obj[6].z );
 glVertex3d( obj[7].x, obj[7].y, obj[7].z );
 glEnd( );
}

/* =============== adjustObj ============= */

static void adjustObj( int w, int h )
{
 double sizex, sizey, sizez;
 double max_size;
 double eye2ref;
 double left, right, top, bottom;
 double front, back;
 double size_w, size_h;
 double ratio;

 alpha = 0.0;
 sizex = xmax - xmin;
 sizey = ymax - ymin;
 sizez = zmax - zmin;
 max_size = MAX( sizex, sizey );
 max_size = MAX( max_size, sizez );
/* Coloca o ponto de referencia no centro
 * da bounding box.
 */
/**** COMPLETE AQUI (1) ****/

/* Posiciona a camera (olho) quatro "max_sizes"
 * distante do pt. de ref. na dir. z e um
 * "max_size" na dir. y.
 */
 eye2ref = 4.0 * max_size;
/**** COMPLETE AQUI (2) ****/

/* Define o plano vertical da camera
 * perpendicular ao plano xz.
 */
/**** COMPLETE AQUI (3) ****/

/* Define os tamanhos da janela de visao de
 * forma que todo o objeto apareca no canvas
 * (com qualquer orientacao), mantendo a mesma
 * razao do retangulo do canvas.
 */
 size_w = size_h = max_size;
 ratio = (double)h / (double)w;
 if( ratio < 1.0 )
  size_w = size_h / ratio;
 else
  size_h = size_w * ratio;

/* Define os parametros do volume de visao,
 * sendo que:
 * (1) o centro da janela de visao fica no
 *      centro da tela;
 * (2) os planos de cerceamento anterior e
 *     posterior ficam a um "max_size" antes
 *     e depois do pt. ref. (visto da camera).
 */
/**** COMPLETE AQUI (4) ****/

/* Define o tipo de projeção (perpectiva) e
 * o volume de visao.
 */
 glMatrixMode( GL_PROJECTION );
 glLoadIdentity( );
 glFrustum( left, right, bottom, top,
           front, back );
}

/* ============== rotateObj ============== */

static int rotateObj( void )
{
 double radius;

/* Define raio de rotacao igual a projecao da
 * linha entre olho e pt. ref. no plano xz.
 */
 radius = sqrt( (eyex-refx)*(eyex-refx) +
               (eyez-refz)*(eyez-refz) );

/* Atualiza a posicao da camera, girando-a de
 * um angulo alpha em torno do eixo y, com
 * centro no pt. de ref.
 */
/**** COMPLETE AQUI (5) ****/

/* Incrementa o angulo alpha.
 */
 alpha += delta;

/* Atualiza a camera e redesenha objeto.
 */
 redraw( canvas );
 return( IUP_DEFAULT );
}

/* ================ redraw =============== */

static int redraw( Ihandle *self )
{
 float  light_pos[] = {0.0F, 0.0F, 1.0F, 0.0F};

/* Atualiza a camera e redesenha objeto.
 * Posiciona luz default branca na camera
 *(antes de aplicar a transformacao de camera).
 */
 glClear( GL_COLOR_BUFFER_BIT |
          GL_DEPTH_BUFFER_BIT );
 glMatrixMode( GL_MODELVIEW );
 glLoadIdentity( );
 glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
 gluLookAt( eyex, eyey, eyez, refx, refy, refz,
            vupx, vupy, vupz );
 displaySolid( );
 IupGLSwapBuffers( self );

 return( IUP_DEFAULT );
}

/* ================ resize =============== */

static int resize(Ihandle *self, int w, int h)
{
 IupGLMakeCurrent( self );
 glViewport( 0, 0, w, h );
 glEnable( GL_LIGHTING );
 glEnable( GL_LIGHT0 );
 glLightModeli( GL_LIGHT_MODEL_TWO_SIDE,
               GL_FALSE );
 glColorMaterial( GL_FRONT_AND_BACK,
                 GL_AMBIENT_AND_DIFFUSE );
 glEnable( GL_COLOR_MATERIAL );
 glEnable( GL_CULL_FACE );
 glDisable( GL_DEPTH_TEST );
 glClearColor( 1.0, 1.0, 1.0, 1.0 );
 adjustObj( w, h );
 return( IUP_DEFAULT );
}

/* =============== main =============== */

int main( int argc, char* argv[] )
{
 Ihandle *dialog;
 IupOpen( &argc, &argv );
 IupGLCanvasOpen( );
 canvas = IupGLCanvas( "redraw" );
 dialog = IupDialog( canvas );
 IupSetAttribute( canvas, IUP_RASTERSIZE,
                "300x200" );
 IupSetAttribute( canvas, IUP_BUFFER,
                 IUP_DOUBLE );
 IupSetAttribute( canvas, IUP_DEPTH_SIZE,
                 "16" );
 IupSetAttribute( canvas, IUP_RESIZE_CB,
                 "resize" );
 IupSetAttribute( dialog, IUP_TITLE,
                 "OpenGL camera demo" );
 IupSetFunction( "redraw", (Icallback)redraw );
 IupSetFunction( "resize", (Icallback)resize );
 IupSetFunction( IUP_IDLE_ACTION,
                (Icallback)rotateObj );
 IupShow( dialog );
 IupMainLoop( );
 IupClose( );
 return( 0 );
}
```