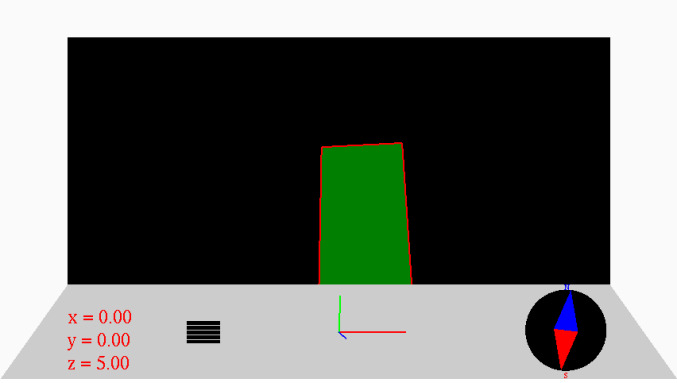
Редослед рендеровања. Приказивање текста у моделу. Статични део модела.

Опис проблема

Направити поједностављену симулацију летења авионом:

1. Направити оквир на екрану који ће представљати изглед кабине авиона као на слици 1.



Слика 1. Кабина авиона

Табла кабине треба да садржи:

- Тренутну позицију авиона представљену координатама.
- Брзиномер са четири правоугаоника где ће број црвених правоугаоника представљати тренутну брзину авиона (остали су црни). Повећавање и смањивање брзине се врши притиском на слова ***U*** и ***J*** на тастатури.
- Координатни систем који показује тренутну оријентацију авиона.
- Компас.

2. Нацртати један квадар у простору.

Решење

Редослед рендеровања.

Да би се реализовао добијени задатак потребно је научити како се дефинише редослед исцртавања, тако да се избегне утицај параметара камере и пројекције на део модела нацртан у координатама екрана.

Како би се избегао тај утицај на неки објекат, потребно је да се камера и пројекција поставе тек након рендеровања датог објекта. То се постиже тако што се GL_MODELVIEW и GL_PROJECTION поставе на јединичне матрице, а потом се изврши рендеровање објеката. Тек након тога се извршава постављање камере и дефинисање пројекције као и рендеровање осталог дела модела.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  drawPlanePanel();
  drawKompas();
  DrawStrelice();
  RenderString(-0.8, -0.7, GLUT_BITMAP_TIMES_ROMAN_24, 1.0, 0.0, 0.0);
  drawMeracBrzine();
  drawCoordinates();
  gluPerspective(alpha, ratio, 0.1f, 20.0f);

  gluLookAt(CameraPosition.m_x,CameraPosition.m_y,CameraPosition.m_z,
            LookAt_vector.m_x,LookAt_vector.m_y,LookAt_vector.m_z,
            LookUp_vector.m_x,LookUp_vector.m_y,LookUp_vector.m_z);

  drawWiredFrameCube();
```

Још је остало на неки делови модела увек буде исцртани на екрану испред остатка модела. Уз помоћ **GL_DEPTH_TEST-**а, ***OpenGL*** одлучује шта ће бити исрцртано на екрану, а шта не у зависност од података о положају тачака у односу на камеру. Ти подаци се чувају у **GL_DEPTH_BUFFER**-у. Да би се вршило приказивање модела на екрану у зависности од редоследа навођења објеката, а не од реалне поставке модела (самих координата објеката), потребно је онемогућити извршавање **GL_DEPTH_TEST**-а наредбом *glDisable(GL_DEPTH_TEST)*. У задатку је потребно да се кабина нађе испред осталог дела модела. То се решава тако што се прво сви остали делови модела рендерују са омогућеним атрибутом **GL_DEPTH_TEST**, а тек онда кабина са онемогућеним атрибутом **GL_DEPTH_TEST**. Наводећи делове кабине за рендеровање мора да се води рачуна о редоследу јер сада **OpenGL** не води рачуна о томе шта је испред чега.

```
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear color and
depth buffers

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluPerspective(alpha, ratio, 0.1f, 20.0f);

  gluLookAt(CameraPosition.m_x,CameraPosition.m_y,CameraPosition.m_z,
            LookAt_vector.m_x,LookAt_vector.m_y,LookAt_vector.m_z,
            LookUp_vector.m_x,LookUp_vector.m_y,LookUp_vector.m_z);

  drawWiredFrameCube();

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  glDisable(GL_DEPTH_TEST);
  drawPlanePanel();
  drawKompas();
```

```
  DrawStrelice();
  RenderString(-0.8, -0.7, GLUT_BITMAP_TIMES_ROMAN_24, 1.0, 0.0, 0.0);
  drawMeracBrzine();
  drawCoordinates();

  glEnable(GL_DEPTH_TEST);

  glutSwapBuffers();  // Swap the front and back frame buffers (double
buffering)
}
```

Цртање карактера по екрану у OpenGL – у.

Да би се одређени низ карактера исписао на екрану потребно је позвати три методе:

- ***glColor3f(0.0, 0.0, 1.0)*** – Постављање боја слова.
- ***glRasterPos2f(double x, double y)*** – Постављање позиције на екрану у пикселима на којој ће се извршити писање.
- ***glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_10, (const unsigned char *)toWrite)*** – Исписивање задатог низа карактера ***toWrite*** фонтом који се задаје као први параметар.

```
 glColor3f(0.0, 0.0, 1.0);
 glRasterPos2f(0.665, -0.525);
 glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_10, (const unsigned char *)"N");
 glColor3f(1.0, 0.0, 0.0);
 glRasterPos2f(0.665, -0.99);
 glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_10, (const unsigned char *)"S");
```

Комплетно решење задатка

```
#include <GL/glut.h>
#include <math.h>
#include <sstream>
#include <iomanip>
#include "vector4d.h"
#include "matrix4x4.h"
#include <vector>
#include <GL/freeglut.h>

using namespace std;

char title[] = "Plane";

#define MOVING_CONST 0.1
#define ROTATION_CONST 3.14/180.0

Vector3D CameraPosition(0.0, 0.0, 5.0);
#define LOOK_MOVEMENT_CONST 0.1
Vector3D LookAt_vector(0.0,0.0,0.0);
Vector3D LookUp_vector(0.0,1.0,0.0);

vector<Vector3D> cube;

vector<Vector3D> coordinateSystem;

vector<Vector3D> planePanel;
double alpha = 90;

double visina = 480;
double ratio = 16.0/9.0;

vector<Vector3D> kompasOsnova;
vector<Vector3D> kompasStrelica;
vector< vector<Vector3D> > meracBrzine;

double r_kompas = 1.0;
double r_kompas_real = 5.0;
int circle_points_kompas = 50;

int speedIndicator = 0;
double streliceRotate = 0.0;
double rotationUpDown = 0.0;

void Transform(vector<Vector3D> &a, Matrix4x4 &T)
{
    for(int i = 0; i < (int)a.size(); i++)
        a[i] = T.Transform(a[i]);
}

void createCoordinates()
{
    coordinateSystem.resize(4);
    coordinateSystem[0] = Vector3D(0.0, 0.0, 0.0);
    coordinateSystem[1] = Vector3D::AxisX;
    coordinateSystem[2] = Vector3D::AxisY;
    coordinateSystem[3] = Vector3D::AxisZ;

    coordinateSystem[1] *= 0.2;
    coordinateSystem[2] *= 0.2;
    coordinateSystem[3] *= 0.2;

    Matrix4x4 MT;
    MT.loadTranslate(0, -0.75, 0.0);
    Transform(coordinateSystem,MT);
```

```
}

void drawCoordinates()
{
    glLineWidth (2.0);

    glBegin(GL_LINES);
        // X axis
        glColor3f(1.0f, 0.0f, 0.0f);     // Red
        glVertex3d(coordinateSystem[0].m_x, coordinateSystem[0].m_y, coordinateSystem[0].m_z);
        glVertex3d(coordinateSystem[1].m_x, coordinateSystem[1].m_y, coordinateSystem[1].m_z);

        // Y axis
        glColor3f(0.0f, 1.0f, 0.0f);     // Green
        glVertex3d(coordinateSystem[0].m_x, coordinateSystem[0].m_y, coordinateSystem[0].m_z);
        glVertex3d(coordinateSystem[2].m_x, coordinateSystem[2].m_y, coordinateSystem[2].m_z);

        // Z axis
        glColor3f(0.0f, 0.0f, 1.0f);     // Blue
        glVertex3d(coordinateSystem[0].m_x, coordinateSystem[0].m_y, coordinateSystem[0].m_z);
        glVertex3d(coordinateSystem[3].m_x, coordinateSystem[3].m_y, coordinateSystem[3].m_z);
    glEnd();
}

void createMeracBrzine()
{
    int i;
    meracBrzine.resize(5);
    for(i = 0; i < 2; i++)
        meracBrzine[i].resize(4);

    meracBrzine[0][0] = Vector3D(-0.05, -0.01, 0.0);
    meracBrzine[0][1] = Vector3D( 0.05, -0.01, 0.0);
    meracBrzine[0][2] = Vector3D( 0.05,  0.01, 0.0);
    meracBrzine[0][3] = Vector3D(-0.05,  0.01, 0.0);

    Matrix4x4 MT;
    MT.loadTranslate(0.0, 0.025, 0.0);
    for(i = 1; i < 5; i++)
    {
        meracBrzine[i] = meracBrzine[i-1];
        Transform(meracBrzine[i], MT);
    }

    MT.loadTranslate(-0.4, -0.8, 0.0);
    for(i = 0; i < 5; i++)
        Transform(meracBrzine[i], MT);
}

void drawMeracBrzine()
{
    glBegin(GL_QUADS);
    for(int i = 0; i < 5; i++)
    {
        if(i < speedIndicator)
            glColor3d(1.0,0.0,0.0);
        else
            glColor3d(0.0,0.0,0.0);
        glVertex3f( meracBrzine[i][0].m_x, meracBrzine[i][0].m_y, meracBrzine[i][0].m_z);
        glVertex3f( meracBrzine[i][1].m_x, meracBrzine[i][1].m_y, meracBrzine[i][1].m_z);
        glVertex3f( meracBrzine[i][2].m_x, meracBrzine[i][2].m_y, meracBrzine[i][2].m_z);
        glVertex3f( meracBrzine[i][3].m_x, meracBrzine[i][3].m_y, meracBrzine[i][3].m_z);
    }
    glEnd();
}
void RenderString(float x, float y, void *font, double r, double g, double b)
{
```

```
  glColor3f(r, g, b);
  glRasterPos2f(x, y);

  char s[100];
  sprintf(s, "x = %.2lf\ny = %.2lf\nz = %.2lf", CameraPosition.m_x, CameraPosition.m_y,
CameraPosition.m_z);
  glutBitmapString(font, (const unsigned char *)s);
}

void Translate(vector<Vector3D> &a, double tx, double ty, double tz)
{
    for(int i = 0; i < (int)a.size(); i++)
        a[i].Translate(tx,ty,tz);
}

void createCubeCoordinates(double a)
{
    double half_a = a * 0.5;

    cube.resize(8);
    cube[0] = (Vector3D(-half_a, -half_a,  half_a));
    cube[1] = (Vector3D( half_a, -half_a,  half_a));
    cube[2] = (Vector3D( half_a, -half_a, -half_a));
    cube[3] = (Vector3D(-half_a, -half_a, -half_a));
    cube[4] = (Vector3D(-half_a, half_a,  half_a));
    cube[5] = (Vector3D( half_a, half_a,  half_a));
    cube[6] = (Vector3D( half_a, half_a, -half_a));
    cube[7] = (Vector3D(-half_a, half_a, -half_a));

    Matrix4x4 MT, MT1, MT2, MT3;
    MT.identity();
    MT1.identity();
    MT2.identity();
    MT3.identity();
    MT1.loadScale(1.0,2.0,0.5);

    MT = MT3*MT2*MT1;
    Transform(cube, MT);
}

void drawCube()
{
    glBegin(GL_QUADS);
        // bottom
    glVertex3f( cube[0].m_x, cube[0].m_y, cube[0].m_z);
    glVertex3f( cube[3].m_x, cube[3].m_y, cube[3].m_z);
    glVertex3f( cube[2].m_x, cube[2].m_y, cube[2].m_z);
    glVertex3f( cube[1].m_x, cube[1].m_y, cube[1].m_z);

        // far
    glVertex3f( cube[7].m_x, cube[7].m_y, cube[7].m_z);
    glVertex3f( cube[6].m_x, cube[6].m_y, cube[6].m_z);
    glVertex3f( cube[2].m_x, cube[2].m_y, cube[2].m_z);
    glVertex3f( cube[3].m_x, cube[3].m_y, cube[3].m_z);

        // left
    glVertex3f( cube[7].m_x, cube[7].m_y, cube[7].m_z);
    glVertex3f( cube[3].m_x, cube[3].m_y, cube[3].m_z);
    glVertex3f( cube[0].m_x, cube[0].m_y, cube[0].m_z);
    glVertex3f( cube[4].m_x, cube[4].m_y, cube[4].m_z);

        // right
    glVertex3f( cube[5].m_x, cube[5].m_y, cube[5].m_z);
    glVertex3f( cube[1].m_x, cube[1].m_y, cube[1].m_z);
    glVertex3f( cube[2].m_x, cube[2].m_y, cube[2].m_z);
    glVertex3f( cube[6].m_x, cube[6].m_y, cube[6].m_z);
```

```
        // top
    glVertex3f( cube[5].m_x, cube[5].m_y, cube[5].m_z);
    glVertex3f( cube[6].m_x, cube[6].m_y, cube[6].m_z);
    glVertex3f( cube[7].m_x, cube[7].m_y, cube[7].m_z);
    glVertex3f( cube[4].m_x, cube[4].m_y, cube[4].m_z);

        // near
    glVertex3f( cube[5].m_x, cube[5].m_y, cube[5].m_z);
    glVertex3f( cube[4].m_x, cube[4].m_y, cube[4].m_z);
    glVertex3f( cube[0].m_x, cube[0].m_y, cube[0].m_z);
    glVertex3f( cube[1].m_x, cube[1].m_y, cube[1].m_z);
    glEnd();
}

void drawWiredFrameCube()
{
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glEnable(GL_POLYGON_OFFSET_FILL);
        glPolygonOffset(1.0,1.0);
        glColor3f(0.0, 0.5, 0.0);
        drawCube();
    glDisable(GL_POLYGON_OFFSET_FILL);

    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glColor3f(1.0, 0.0, 0.0);
    drawCube();
}

void createPlanePanel()
{
    planePanel.resize(12);

    double ymax = 1;
    double xmax = 1;

    double side = 20; //%
    double bootom = 50; //%

    planePanel[0] = Vector3D(-xmax, -ymax, 0.0);
    planePanel[1] = Vector3D(xmax, -ymax, 0.0);
    planePanel[2] = Vector3D(xmax,  ymax, 0.0);
    planePanel[3] = Vector3D(-xmax,  ymax, 0.0);

    planePanel[4] = Vector3D(-xmax + xmax*side/100, -ymax + ymax*bootom/100, 0.0);
    planePanel[5] = Vector3D(xmax - xmax*side/100,  -ymax + ymax*bootom/100, 0.0);
    planePanel[6] = Vector3D(xmax - xmax*side/100,  ymax - ymax*side/100, 0.0);
    planePanel[7] = Vector3D(-xmax + xmax*side/100, ymax - ymax*side/100, 0.0);
}

void drawPlanePanel()
{
    glPolygonMode(GL_FRONT, GL_FILL);

    glBegin(GL_QUADS);
        glColor3f(0.80f, 0.80f, 0.80f);
        glVertex3f( planePanel[0].m_x, planePanel[0].m_y, planePanel[0].m_z);
        glVertex3f( planePanel[1].m_x, planePanel[1].m_y, planePanel[1].m_z);
        glVertex3f( planePanel[5].m_x, planePanel[5].m_y, planePanel[5].m_z);
        glVertex3f( planePanel[4].m_x, planePanel[4].m_y, planePanel[4].m_z);

        glColor3f(0.98f, 0.98f, 0.98f);
        glVertex3f( planePanel[1].m_x, planePanel[1].m_y, planePanel[1].m_z);
        glVertex3f( planePanel[2].m_x, planePanel[2].m_y, planePanel[2].m_z);
        glVertex3f( planePanel[6].m_x, planePanel[6].m_y, planePanel[6].m_z);
        glVertex3f( planePanel[5].m_x, planePanel[5].m_y, planePanel[5].m_z);

        glVertex3f( planePanel[7].m_x, planePanel[7].m_y, planePanel[7].m_z);
```

```
        glVertex3f( planePanel[6].m_x, planePanel[6].m_y, planePanel[6].m_z);
        glVertex3f( planePanel[2].m_x, planePanel[2].m_y, planePanel[2].m_z);
        glVertex3f( planePanel[3].m_x, planePanel[3].m_y, planePanel[3].m_z);

        glVertex3f( planePanel[0].m_x, planePanel[0].m_y, planePanel[0].m_z);
        glVertex3f( planePanel[4].m_x, planePanel[4].m_y, planePanel[4].m_z);
        glVertex3f( planePanel[7].m_x, planePanel[7].m_y, planePanel[7].m_z);
        glVertex3f( planePanel[3].m_x, planePanel[3].m_y, planePanel[3].m_z);
    glEnd();
}

void createKompas()
{
    kompasOsnova.resize(circle_points_kompas);

    r_kompas = 0.12;

    for (int i = 0; i < circle_points_kompas; i++)
    {
        double angle = 2*PI*i/circle_points_kompas;
        kompasOsnova[i] = Vector3D(r_kompas*cos(angle), r_kompas*sin(angle)*ratio, 0.0);
    }

    Matrix4x4 MT;
    MT.loadTranslate(0.67, -0.74, 0.0);
    Transform(kompasOsnova, MT);
}

void createStrelice(double phi)
{
    kompasStrelica.resize(5);

    r_kompas = 0.12;

    kompasStrelica[0] = Vector3D(0.0, r_kompas, 0.0);
    kompasStrelica[1] = Vector3D(-r_kompas*0.3, 0.0, 0.0);
    kompasStrelica[2] = Vector3D(r_kompas*0.3, 0.0, 0.0);
    kompasStrelica[3] = Vector3D(0.0, -r_kompas, 0.0);
    kompasStrelica[4] = Vector3D(0.0, 0.0, 0.0);

    Matrix4x4 MR;
    MR.loadRotateZ(-phi);
    Transform(kompasStrelica,MR);
}

void DrawStrelice()
{
    createStrelice(streliceRotate);

    kompasStrelica[0].m_y *= ratio;
    kompasStrelica[1].m_y *= ratio;
    kompasStrelica[2].m_y *= ratio;
    kompasStrelica[3].m_y *= ratio;

    Matrix4x4 MT;
    MT.loadTranslate(0.67, -0.74, 0.0);
    Transform(kompasStrelica, MT);

    glPolygonMode(GL_FRONT, GL_FILL);

    glColor3f (0.0, 0.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f( kompasStrelica[0].m_x, kompasStrelica[0].m_y, kompasStrelica[0].m_z);
        glVertex3f( kompasStrelica[1].m_x, kompasStrelica[1].m_y, kompasStrelica[1].m_z);
        glVertex3f( kompasStrelica[2].m_x, kompasStrelica[2].m_y, kompasStrelica[2].m_z);
    glEnd();
```

```
    glColor3f (1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
        glVertex3f( kompasStrelica[1].m_x, kompasStrelica[1].m_y, kompasStrelica[1].m_z);
        glVertex3f( kompasStrelica[3].m_x, kompasStrelica[3].m_y, kompasStrelica[3].m_z);
        glVertex3f( kompasStrelica[2].m_x, kompasStrelica[2].m_y, kompasStrelica[2].m_z);
    glEnd();
}

void drawKompas()
{
    glPolygonMode(GL_FRONT, GL_FILL);

    glColor3f (0.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    for(unsigned int i = 0; i < kompasOsnova.size(); i++)
        glVertex3f( kompasOsnova[i].m_x, kompasOsnova[i].m_y, kompasOsnova[i].m_z);
    glEnd();

    glColor3f(0.0, 0.0, 1.0);
    glRasterPos2f(0.665, -0.525);
    glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_10, (const unsigned char *)"N");
    glColor3f(1.0, 0.0, 0.0);
    glRasterPos2f(0.665, -0.99);
    glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_10, (const unsigned char *)"S");
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear color and depth buffers

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluPerspective(alpha, ratio, 0.1f, 20.0f);

    gluLookAt(CameraPosition.m_x,CameraPosition.m_y,CameraPosition.m_z,
              LookAt_vector.m_x,LookAt_vector.m_y,LookAt_vector.m_z,
              LookUp_vector.m_x,LookUp_vector.m_y,LookUp_vector.m_z);

    drawWiredFrameCube();


    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glDisable(GL_DEPTH_TEST);
    drawPlanePanel();
    drawKompas();
    DrawStrelice();
    RenderString(-0.8, -0.7, GLUT_BITMAP_TIMES_ROMAN_24, 1.0, 0.0, 0.0);
    drawMeracBrzine();
    drawCoordinates();

    glEnable(GL_DEPTH_TEST);

    glutSwapBuffers();  // Swap the front and back frame buffers (double buffering)
}

void reshape(GLsizei width, GLsizei height)
{
    if(height * ratio <= width)
        width = ratio * height;
    else
```

```
        height = width / ratio;

   glViewport(0, 0, width, height);
}

void moveForward()
{
    Matrix4x4 MT;
    Vector3D V;
    Vector3D L, T;

    V = LookAt_vector - CameraPosition;
    V.m_y = 0.0;
    V.Normalize();
    V = V*(speedIndicator*MOVING_CONST);
    MT.loadTranslate(V.m_x, V.m_y, V.m_z);

    CameraPosition = MT.Transform(CameraPosition);
    LookAt_vector  = MT.Transform(LookAt_vector);
}

void moveBackward()
{
    Matrix4x4 MT;
    Vector3D V;
    Vector3D L, T;

    V = LookAt_vector - CameraPosition;
    V.Normalize();
    V.m_y = 0.0;
    V = -V*(speedIndicator*MOVING_CONST);
    MT.loadTranslate(V.m_x, V.m_y, V.m_z);

    CameraPosition = MT.Transform(CameraPosition);
    LookAt_vector  = MT.Transform(LookAt_vector);
}

void speedUp()
{
    speedIndicator++;
    if(speedIndicator == 6)
        speedIndicator--;
}

void speedDown()
{
    speedIndicator--;
    if(speedIndicator == -1)
        speedIndicator++;
}

void turnLeft()
{
    Matrix4x4 MT;
    Matrix4x4 Mr,Mtr1,Mtr2;

    Mtr1.loadTranslate(-CameraPosition.m_x, -CameraPosition.m_y, -CameraPosition.m_z);
    Mtr2.loadTranslate(CameraPosition.m_x, CameraPosition.m_y, CameraPosition.m_z);
    Mr.loadRotateY(ROTATION_CONST);

    MT = Mtr2 * Mr * Mtr1;

    LookAt_vector = MT.Transform(LookAt_vector);

    Vector4D LookUp_vector4d = LookUp_vector;
    LookUp_vector4d.m_w = 0.0;
    LookUp_vector4d = MT.Transform(LookUp_vector4d);
```

```
    LookUp_vector = LookUp_vector4d;

    streliceRotate += ROTATION_CONST;

    Mtr1.loadTranslate(-coordinateSystem[0].m_x, -coordinateSystem[0].m_y, -
coordinateSystem[0].m_z);
    Mtr2.loadTranslate(coordinateSystem[0].m_x, coordinateSystem[0].m_y,
coordinateSystem[0].m_z);
    Mr.loadRotateY(ROTATION_CONST);

    MT = Mtr2 * Mr * Mtr1;

    Transform(coordinateSystem, MT);
}

void turnRight()
{
    Matrix4x4 MT;
    Matrix4x4 Mr,Mtr1,Mtr2;

    Mtr1.loadTranslate(-CameraPosition.m_x, -CameraPosition.m_y, -CameraPosition.m_z);
    Mtr2.loadTranslate(CameraPosition.m_x, CameraPosition.m_y, CameraPosition.m_z);
    Mr.loadRotateY(-ROTATION_CONST);

    MT = Mtr2 * Mr * Mtr1;

    LookAt_vector = MT.Transform(LookAt_vector);

    Vector4D LookUp_vector4d = LookUp_vector;
    LookUp_vector4d.m_w = 0.0;
    LookUp_vector4d = MT.Transform(LookUp_vector4d);
    LookUp_vector = LookUp_vector4d;

    streliceRotate -= ROTATION_CONST;

    Mtr1.loadTranslate(-coordinateSystem[0].m_x, -coordinateSystem[0].m_y, -
coordinateSystem[0].m_z);
    Mtr2.loadTranslate(coordinateSystem[0].m_x, coordinateSystem[0].m_y,
coordinateSystem[0].m_z);
    Mr.loadRotateY(-ROTATION_CONST);

    MT = Mtr2 * Mr * Mtr1;

    Transform(coordinateSystem, MT);
}

void lookUp()
{
    Matrix4x4 MT;
    Vector3D f = LookAt_vector - CameraPosition;
    Vector3D w = LookUp_vector.Cross(f);

    w.Normalize();

    if(rotationUpDown + ROTATION_CONST < 0.5*M_PI)
    {
        MT.loadRotate(CameraPosition, w, -ROTATION_CONST);

        LookAt_vector = MT.Transform(LookAt_vector);

        Vector4D LookUp_vector4d = LookUp_vector;
        LookUp_vector4d.m_w = 0.0;
        LookUp_vector4d = MT.Transform(LookUp_vector4d);
        LookUp_vector = LookUp_vector4d;

        rotationUpDown += ROTATION_CONST;
```

```
        Matrix4x4 Mr;

        Vector3D axis = coordinateSystem[1] - coordinateSystem[0];
        Mr.loadRotate(coordinateSystem[0],axis, ROTATION_CONST);

        Transform(coordinateSystem, Mr);
    }
}

void lookDown()
{
    Matrix4x4 MT;
    Vector3D f = LookAt_vector - CameraPosition;
    Vector3D w = LookUp_vector.Cross(f);

    w.Normalize();

    if(rotationUpDown + ROTATION_CONST > -0.5*M_PI)
    {
        MT.loadRotate(CameraPosition, w, ROTATION_CONST);

        LookAt_vector = MT.Transform(LookAt_vector);

        Vector4D LookUp_vector4d = LookUp_vector;
        LookUp_vector4d.m_w = 0.0;
        LookUp_vector4d = MT.Transform(LookUp_vector4d);
        LookUp_vector = LookUp_vector4d;

        rotationUpDown -= ROTATION_CONST;

        Matrix4x4 Mr;

        Vector3D axis = coordinateSystem[1] - coordinateSystem[0];
        Mr.loadRotate(coordinateSystem[0],axis, -ROTATION_CONST);

        Transform(coordinateSystem, Mr);
    }
}

void KeyboardKeyPressed(unsigned char key, int x, int y)
{
    switch(key)
    {
    case 27:    //ESC key
        exit(0);
        break;
    case 'w':
        cout << "w pressed -> moving forward" << endl;
        moveForward();
        break;
    case 's':
        cout << "s pressed -> moving backward" << endl;
        moveBackward();
        break;
    case 'u':
        cout << "u pressed -> SpeedUp" << endl;
        speedUp();
        break;
    case 'j':
        cout << "j pressed -> SpeedDown" << endl;
        speedDown();
        break;
    case 52:
        cout << "4 pressed -> turning left" << endl;
        turnLeft();
        break;
    case 54:
```

```
        cout << "6 pressed -> turning right" << endl;
        turnRight();
        break;
    case 50:
        cout << "2 pressed -> look up" << endl;
        lookDown();
        break;
    case 56:
        cout << "8 pressed -> look down" << endl;
        lookUp();
        break;
    }

    glutPostRedisplay();
}

/* Initialize OpenGL Graphics */
void initGL()
{
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
   glShadeModel(GL_FLAT);

   glEnable(GL_DEPTH_TEST);   // Enable depth testing for z-culling
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);            // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_STENCIL); // Enable double buffered
mode
    glutInitWindowSize(ratio * visina, visina);   // Set the window's initial width & height
    glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
    glutCreateWindow(title);          // Create window with the given title

    createCubeCoordinates(2.0);
    createPlanePanel();
    createKompas();
    createStrelice(streliceRotate);
    createMeracBrzine();
    createCoordinates();

    glutDisplayFunc(display);       // Register callback handler for window re-paint event
    glutReshapeFunc(reshape);       // Register callback handler for window re-size event

    glutKeyboardFunc(KeyboardKeyPressed);

    initGL();                       // Our own OpenGL initialization
    glutMainLoop();                 // Enter the infinite event-processing loop
    return 0;
}
```