「情報工学」ＣＧ 課題７

令和２年１１月１８日（水曜日）

提出締め切り 令和２年１１月２４日（火曜日）２４：００提出

提出物：提出物は各課題（課題１）の最後の状態のプログラムのソース
　　コードとする。

提出：takechi.kentaro.16@shizuoka.ac.jp（三浦研武智健太郎君）
件名：情報工学課題３　学籍番号　名前
提出方法：添付ファイル、本文にも　TA が管理しやすくなるので、学籍番号、名前を必ず記入すること。

スタック処理

グラフィック

- Push と Pop

サンプルプログラム

```
/*
 * robotModel.c
 * 簡易モデラを使ったロボットシミュレータ
 *
 * Date   : November 14, 2020
 * Author : Kenjiro T. MIURA, Shizuoka University, Japan
 *
 */

/* ロボットシミュレータ */
#include "GL/glut.h"
#include <math.h>
#include <stdio.h>
// #include <stdlib.h>
#include <string.h>
#include "robotModel.h"

body   Body[MAX_NO_BODY];
face   Face[MAX_NO_FACE];
edge   Edge[MAX_NO_EDGE];
vertex Vertex[MAX_NO_VERTEX];

int No_Body; /* ロボットを構成する部品の数 */

static int base_angle = 0, arm0_angle = 0, arm1_angle = 0;

/* ロボットの形状データの読み込み */
void readRobotData( char *filename )
{
    int i, j; /* カウンタ */
```

```
FILE *fp;
int no_face, no_edge, no_vertex; /* 面,稜線,頂点の総数 */
int no_f, no_e, no_v; /* 個々の部品の面,稜線,頂点の数 */
int index0, index1, index2, index3;
    errno_t error;

if ( ( error = fopen_s ( &fp, filename, "r" ) ) != 0 ) {
    printf ( "cannot open %s!¥n", filename );
    exit (1);
}

no_face = no_edge = no_vertex = 0; /* 総数の初期化 */

fscanf_s ( fp, "%d", &No_Body ); /* ロボットの部品の数 */

for ( i=0; i<No_Body; ++i ) {

    /* 部品に属する面 */
    fscanf_s ( fp, "%d", &index0 );
    Body[i].face = &Face[no_face+index0];

    fscanf_s ( fp, "%d", &no_f ); /* 面の数の読み込み */

    for ( j=0; j<no_f; ++j ) {
        /* 面に属する稜線, 次の面 */
        fscanf_s ( fp, "%d %d", &index0, &index1 );
        Face[no_face+j].edge = &Edge[no_edge+index0];
        if ( index1 == -1 ) Face[no_face+j].next = NULL;
        else Face[no_face+j].next = &Face[no_face+index1];
    }

    fscanf_s ( fp, "%d", &no_e ); /* 稜線の数の読み込み */

    for ( j=0; j<no_e; ++j ) {
        /* 稜線に属する頂点 */
        fscanf_s ( fp, "%d %d", &index0, &index1 );
        Edge[no_edge+j].start_vertex = &Vertex[no_vertex+index0];
        Edge[no_edge+j].end_vertex = &Vertex[no_vertex+index1];
        /* 右稜線,左稜線,右面,左面 */
        fscanf_s ( fp, "%d %d %d %d", &index0, &index1, &index2, &index3 );
        Edge[no_edge+j].right_edge = &Edge[no_edge+index0];
        Edge[no_edge+j].left_edge  = &Edge[no_edge+index1];
        Edge[no_edge+j].right_face = &Face[no_face+index2];
        Edge[no_edge+j].left_face =  &Face[no_face+index3];
    }

    fscanf_s ( fp, "%d", &no_v ); /* 頂点の数の読み込み */

    for ( j=0; j<no_v; ++j ) { /* 頂点データの読み込み */
        fscanf_s ( fp, "%f %f %f", &Vertex[no_vertex+j].coord[0],
                                   &Vertex[no_vertex+j].coord[1],
                                  &Vertex[no_vertex+j].coord[2] );
    }
```

```
        no_face += no_f; no_edge += no_e; no_vertex += no_v; /* 総数の更新 */
    } /* iループの終了 */
} /* readRobotDataの終了 */

/* 視点と注視点のデータ */
static GLdouble eye[3] = { 200.0, 200.0, 100.0 };
static GLdouble center[3] = { 0.0, 0.0, 0.0 };

void addBase(void)
{
    base_angle = (base_angle + 10) % 360;
}

void subtractBase(void)
{
    base_angle = (base_angle - 10) % 360;
}

void addArm0(void)
{
    arm0_angle = (arm0_angle + 10) % 360;
}

void subtractArm0(void)
{
    arm0_angle = (arm0_angle - 10) % 360;
}

void addArm1(void)
{
    arm1_angle = (arm1_angle + 10) % 360;
}

void subtractArm1(void)
{
    arm1_angle = (arm1_angle - 10) % 360;
}

/* 面に属する頂点の検索 */
void getVertexData ( face *pface, int *no_vertex,
                     GLfloat vertex_data[MAX_NO_VERTEX][3] )
{
    int i; /* カウンタ */
    edge *sedge, *nedge; /* 最初の稜線,次の稜線 */
    face *left_face; /* 左の面 */

    *no_vertex = 0; /* 初期化 */
    sedge = nedge = pface->edge;
    left_face = nedge->left_face;

    do {
        if( left_face == pface ) {
                for ( i=0; i<3; ++i ) {
                vertex_data[*no_vertex][i] = nedge->end_vertex->coord[i];
```

```
            }
            nedge = nedge->left_edge;
        }
        else {
            for ( i=0; i<3; ++i ) {
                vertex_data[*no_vertex][i] = nedge->start_vertex->coord[i];
            }
            nedge = nedge->right_edge;
        }
        ++*no_vertex;
        left_face = nedge->left_face;
    } while ( nedge != sedge ); /* 次の稜線が最初の稜線に一致したら終了 */

} /* getVertexDataの終了 */

/* ロボットの腕の描画 */
void drawArmShading ( body bdy )
{
    int i; /* カウンター */
    int no_vertex; /* 頂点の数 */
    face *pface; /* 面へのポインタ */
    GLfloat vertex_data[MAX_NO_VERTEX][3]; /* 頂点データ */
    GLfloat normal[3]; /* 法線ベクトル */
    double norm; /* ベクトルのノルム */
    GLfloat vec0[3], vec1[3];

    pface = bdy.face; /* 物体に属する面 */

    while ( pface != NULL ) {
        getVertexData( pface, &no_vertex, vertex_data );
        /* 法線ベクトル */
        for ( i=0; i<3; ++i ) {
            vec0[i] = vertex_data[2][i] - vertex_data[1][i];
            vec1[i] = vertex_data[0][i] - vertex_data[1][i];
        }
        normal[0] = vec0[1] * vec1[2] - vec0[2] * vec1[1];
        normal[1] = vec0[2] * vec1[0] - vec0[0] * vec1[2];
        normal[2] = vec0[0] * vec1[1] - vec0[1] * vec1[0];
        norm = normal[0] * normal[0] + normal[1] * normal[1] +
               normal[2] * normal[2];
        norm = sqrt ( norm );
        for ( i=0; i<3; ++i ) normal[i] /= norm;

        glBegin(GL_POLYGON); /* 多角形の描画開始 */
            glNormal3f( normal[0], normal[1], normal[2] );
            for ( i=0; i<no_vertex; i++ ){
                    glVertex3f(vertex_data[i][0], vertex_data[i][1],
                               vertex_data[i][2]);
            } /* jループの終了 */
        glEnd(); /* 描画終了 */
        pface = pface->next; /* 次の面 */
    } /* whileの終了*/
}
```

```
/*  スクリーンをクリアーする。カレントカラーを白にセットする。 *
 *  ロボットの腕を面塗り描画する。                              */
void ourDisplay (void)
{
    GLfloat material_color0[4] = { 1.0, 0.0, 0.0, 1.0 }; /* 拡散反射成分 */
    GLfloat material_color1[4] = { 0.0, 0.0, 1.0, 1.0 }; /* 拡散反射成分 */
    GLfloat material_color2[4] = { 0.0, 1.0, 0.0, 1.0 }; /* 拡散反射成分 */
    GLfloat material_specular[4] = { 0.2, 0.2, 0.2, 1.0 }; /* 鏡面反射成分 */

    /* バッファーのクリアー*/
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* 鏡面反射成分のセット */
    glMaterialfv(GL_FRONT, GL_SPECULAR, material_specular);

    /* ロボットのベースの回転 */
    glPushMatrix();
        /* ロボットベースの回転 */
        glRotatef ( (GLfloat) base_angle, 0.0, 0.0, 1.0 );
        /* レッド */
        glMaterialfv(GL_FRONT, GL_DIFFUSE, material_color0);
        glMaterialf(GL_FRONT, GL_SHININESS, 10.0);
        drawArmShading(Body[0]); /*  ロボットのベースの描画 */

        glPushMatrix();
            /* ロボットの腕No.0の回転 */
            glRotatef ( (GLfloat) arm0_angle, 0.0, -1.0, 0.0 );
            /* ブルー */
            glMaterialfv(GL_FRONT, GL_DIFFUSE, material_color1);
            drawArmShading(Body[1]);      /* ロボットの腕No.0の描画 */
            /* ロボットの腕No.1の回転 */
            glTranslatef ( 40.0, 0.0, 0.0 );
            glRotatef ( (GLfloat) arm1_angle, 0.0, -1.0, 0.0 );
            glTranslatef ( -40.0, 0.0, 0.0 );
            /* ブルー */
            glMaterialfv(GL_FRONT, GL_DIFFUSE, material_color2);
            drawArmShading(Body[2]);      /* ロボットの腕No.1の描画 */
        glPopMatrix();

    glPopMatrix();
    glFlush();
}

/* OpenGLの初期化 */
void ourInit (void)
{
    GLfloat light0_position[] = { 0.0, 0.0, 1.0, 0.0 }; /* 照明の位置 */
    GLfloat light1_position[] = { 100.0, 100.0, 0.0, 1.0 }; /* 照明の位置 */

    GLfloat light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 }; /* 拡散成分 */
    GLfloat light1_diffuse[] = { 0.5, 0.5, 0.5, 1.0 }; /* 拡散成分 */

    GLfloat light_specular[] = { 0.3, 0.3, 0.3, 1.0 }; /* 鏡面成分 */
```

```
    GLfloat lmodel_ambient[] = { 0.1, 0.1, 0.1, 1.0 }; /* 周囲光 */

    /* 照明No.0 */
    glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);

    /* 照明No.1 */
    glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
    glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular);

    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHT1);
    glDepthFunc(GL_LEQUAL);
    glEnable(GL_DEPTH_TEST);
}

/*  ウインドウが最初にオープンした時やウインドウが移動やリサイズされた時
 *  呼ばれる。 */
void ourReshape (int width, int height)
{
    int i; /* カウンター */
    GLdouble up[3]; /* ビューアップベクトル */
    GLdouble vector[3]; /* ビューアップベクトル計算用ベクトル */
    GLdouble norm; /* ベクトルのノルム */

    glViewport (0, 0, width, height);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(30.0, (GLfloat) width/(GLfloat) height, 10.0, 1000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    /* ビューアップベクトルを計算する。*/
    for ( i=0; i<3; i++ ){
        vector[i] = center[i] - eye[i];
    }
    up[0] = - vector[0] * vector[2];
    up[1] = - vector[1] * vector[2];
    up[2] = vector[0] * vector[0] + vector[1] * vector[1];
    norm = up[0] * up[0] + up[1] * up[1] + up[2] * up[2];
    norm = sqrt ( norm );

    for ( i=0; i<3; ++i ) up[i] /= norm;

    /* ビュー行列をセットする。*/
    gluLookAt(eye[0], eye[1], eye[2], center[0], center[1], center[2],
              up[0], up[1], up[2]);
}
```

```
/* マウスによる移動 */
void ourMouse(int button, int state, int x, int y)
{
    if (state != GLUT_DOWN)
        return;

    switch(button) {
    case GLUT_LEFT_BUTTON:
        addArm1();
        break;
    case GLUT_MIDDLE_BUTTON:
        addBase();
        break;
    case GLUT_RIGHT_BUTTON:
        subtractArm1();
    }
    glutPostRedisplay();
}

/* キーによる移動 */
void ourKeyboard(unsigned char key, int x, int y)
{
    switch(key) {
    case 27:
        exit(0);
        break;
    case 'l':
        subtractBase();
        break;
    case 'r':
        addBase();
        break;
    case 'u':
        addArm0();
        break;
    case 'd':
        subtractArm0();
        break;
    default:
        break;
    }
    glutPostRedisplay();
}

/*  メインループ                                *
 *  初期値でウインドウをイニシャライズする. */
int main(int argc, char **argv)
{
    char filename[30];

    strcpy_s ( filename, "robot.dat" );

    /* ロボット幾何形状データの読み込み */
    readRobotData ( filename );
```

```
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(400, 400);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("robotModelGlut");
    ourInit();
    glutReshapeFunc(ourReshape);
    glutKeyboardFunc(ourKeyboard);
    glutMouseFunc(ourMouse);
    glutDisplayFunc(ourDisplay);
    glutMainLoop();
    return 0;
}
```

課題１

ロボットシミュレータを目的としたプログラム robotModel.cpp の

ourDisplay() 関数を

1) ‘a’（正方向），‘A’（負方向）の入力でベース部（赤）をｚ軸回り
   に回転
2) ‘b’（負方向），‘B’（正方向）の入力でアーム0部（青）をｙ軸回り
   に回転
3) ‘c’（負方向），‘C’（正方向）の入力でアーム1部（緑）をｙ軸回り
   に回転

と動作するように glPushMatrix(), glPopMatrix() を利用して変更を加えよ.