

情報工学

2021年度後期 第7週 [11月24日]

静岡大学

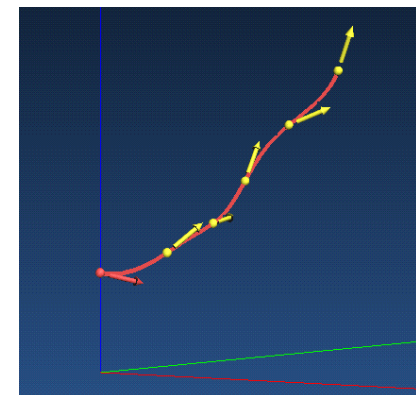
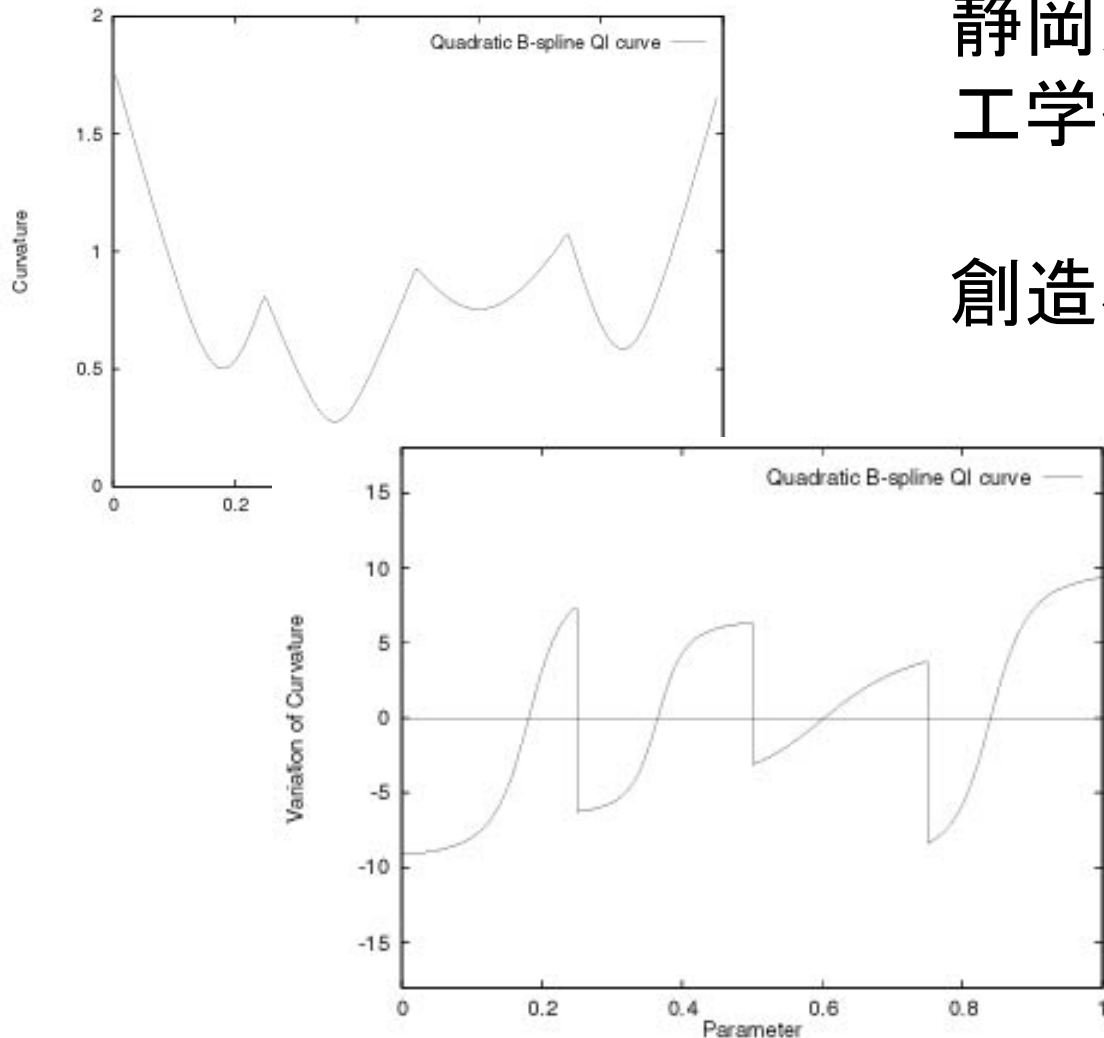
工学研究科機械工学専攻

ロボット・計測情報講座

創造科学技術大学院

情報科学専攻

三浦 憲二郎



情報工学CGパート試験

- 日時 12月1日(水) 午前10:00～午後5時
- 場所 工学部1号館210号室三浦居室
- 試験問題を配布
- 12月8日(水) 午後5時までに三浦に提出

講義アウトライン [11月24日]

- OpenGL
 - 複数パーツから構成されるモデルの動作
 - 行列のプッシュとポップ

行列のプッシュとポップ

ロボットシミュレータ

ロボットアーム全体の動き

各関節ごとの動き

簡単に様々な座標系が利用できることが望ましい

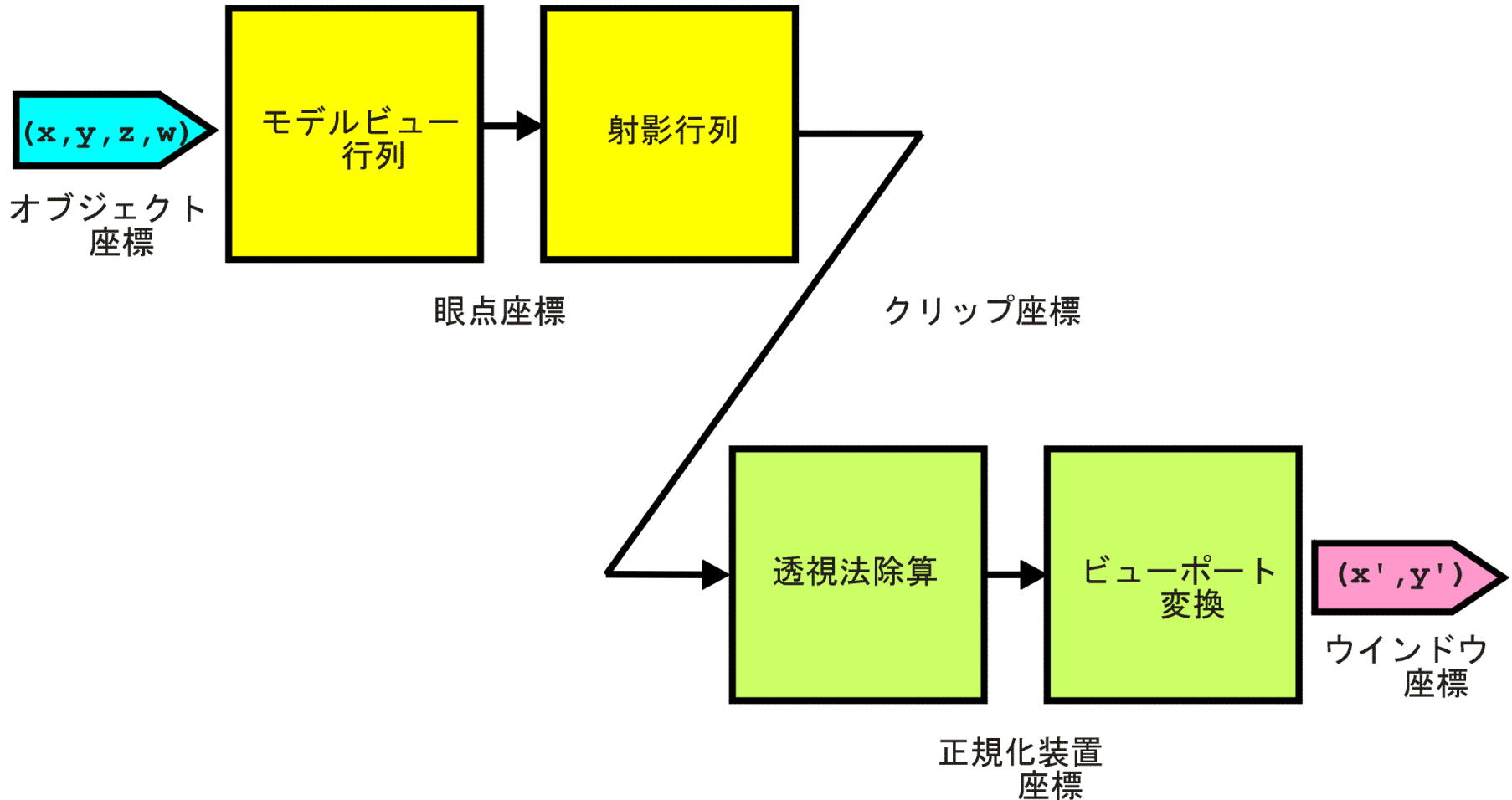
OpenGLはスタックマシンなので,

Rotate, Translate

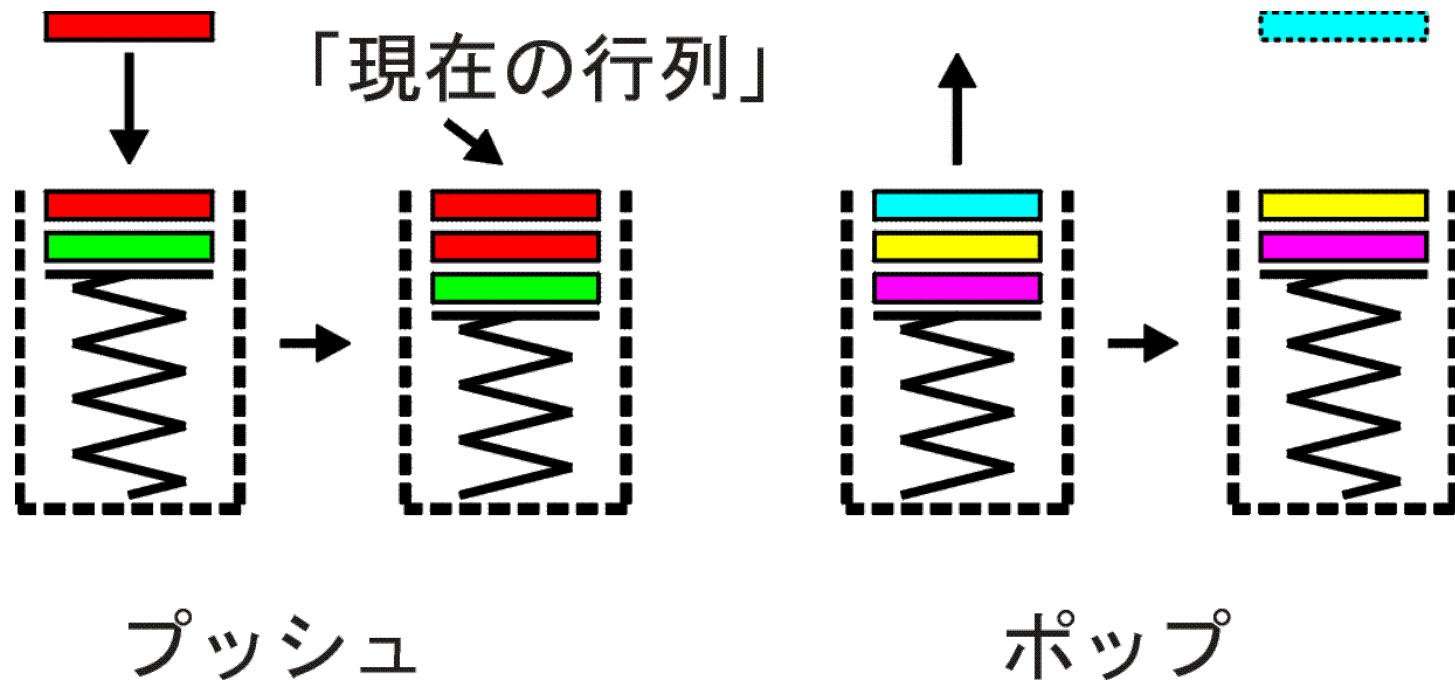
により座標系が移動



頂点変換の手順



行列スタック



FILO (First In, Last Out)

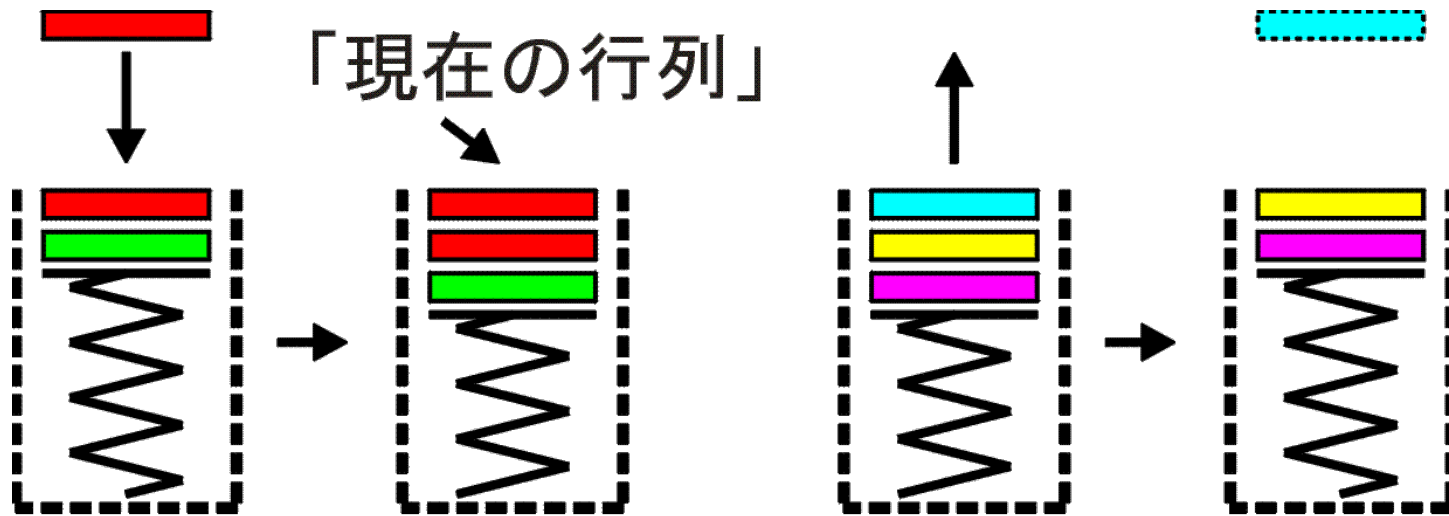


行列スタックのプッシュ

コマンド **glPushMatrix()**

void

```
glPushMatrix( void ); /* モデルビュー行列スタック 最低32段 */  
/* 射影行列スタック 最低2段 */
```



プッシュ

ポップ

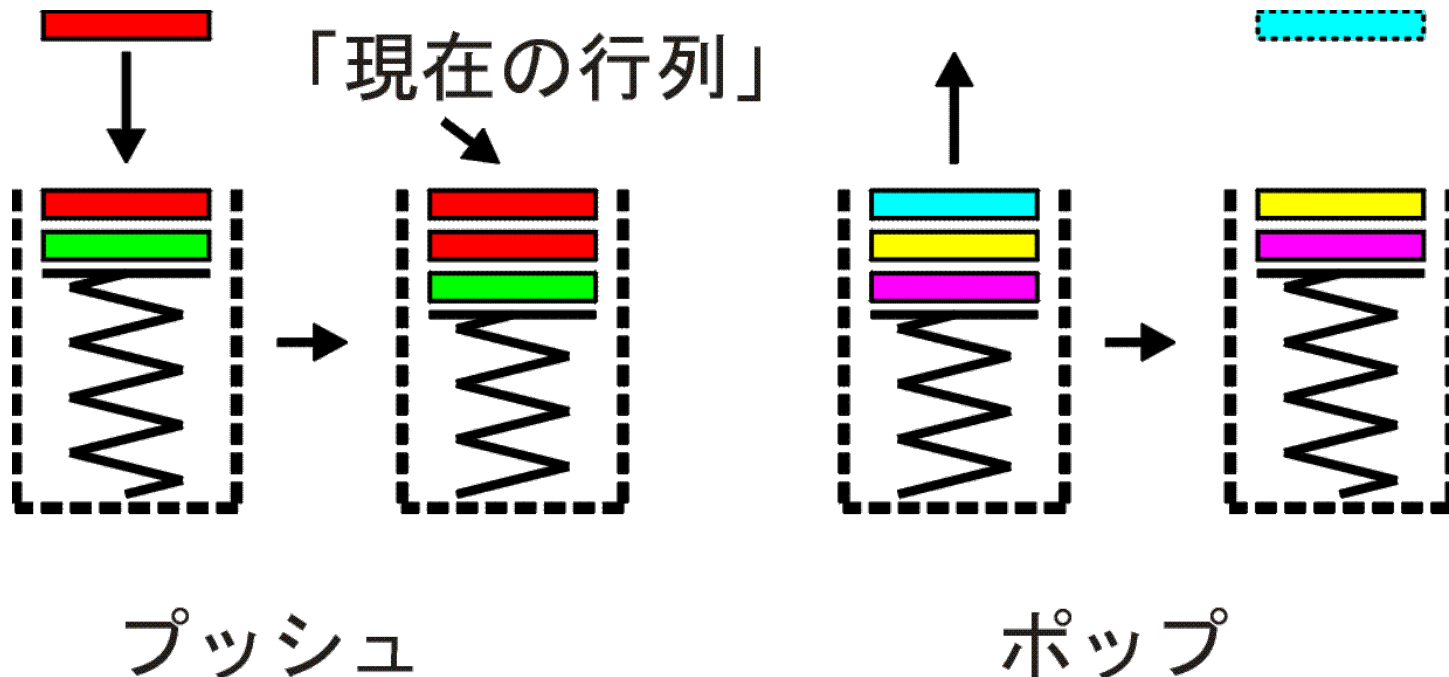


行列スタックのポップ

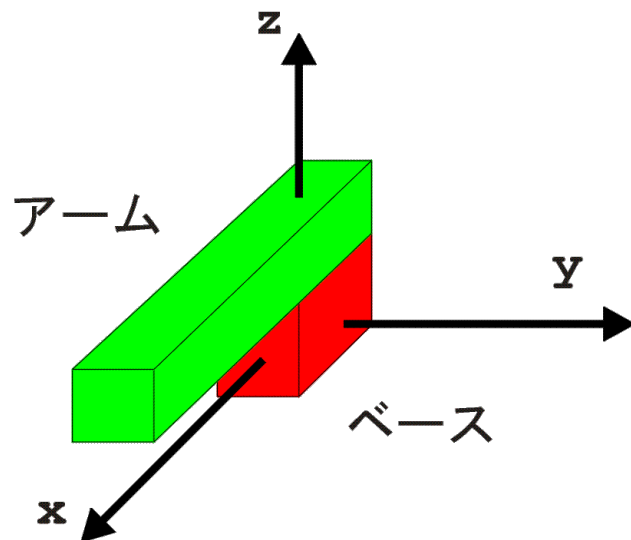
コマンド **glPopMatrix()**

void

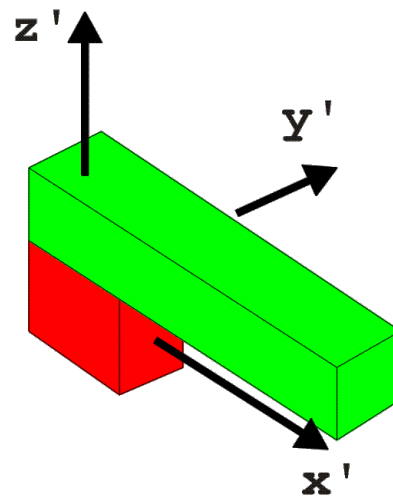
glPopMatrix(void); /* スタックに行列が1つしかない場合はエラー */



プッシュとポップを使わない場合



z軸回りに回転

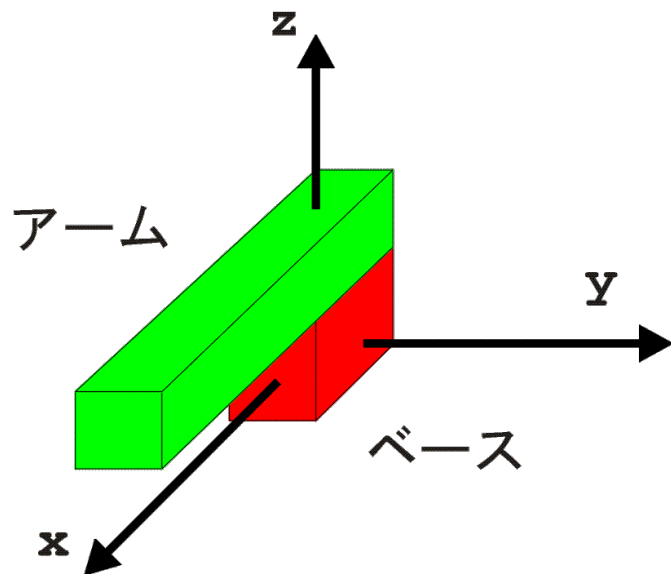


ベースとアームが回転

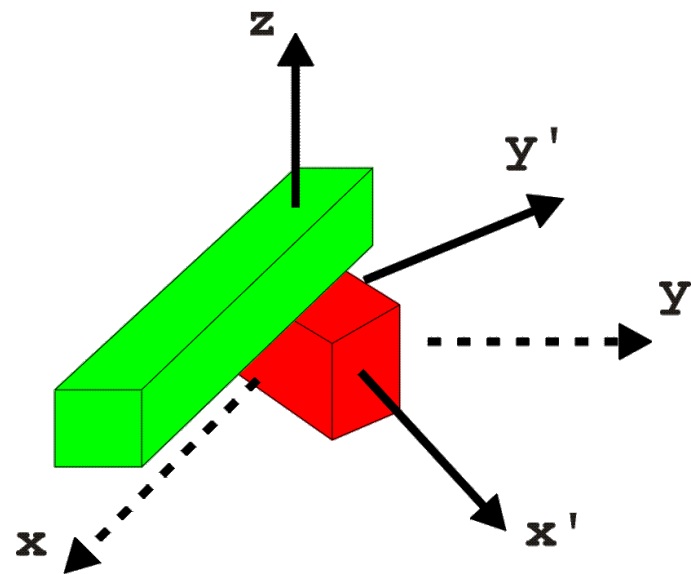
```
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);  
drawBase();  
drawArm();
```



プッシュとポップの効果



z軸回りに回転

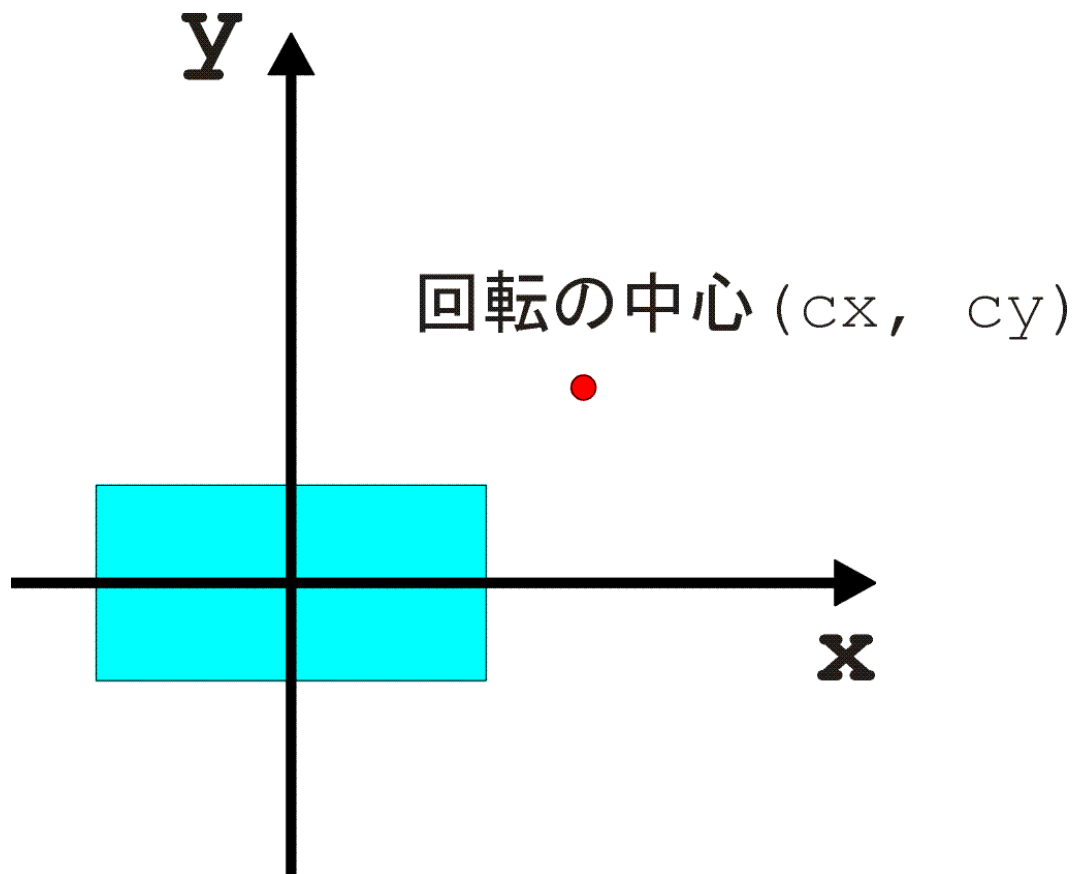


ベースだけが回転

```
glPushMatrix();  
    glRotatef(45.0f, 0.0f, 0.0f, 1.0f);  
    drawBase();  
glPopMatrix();  
drawArm();
```



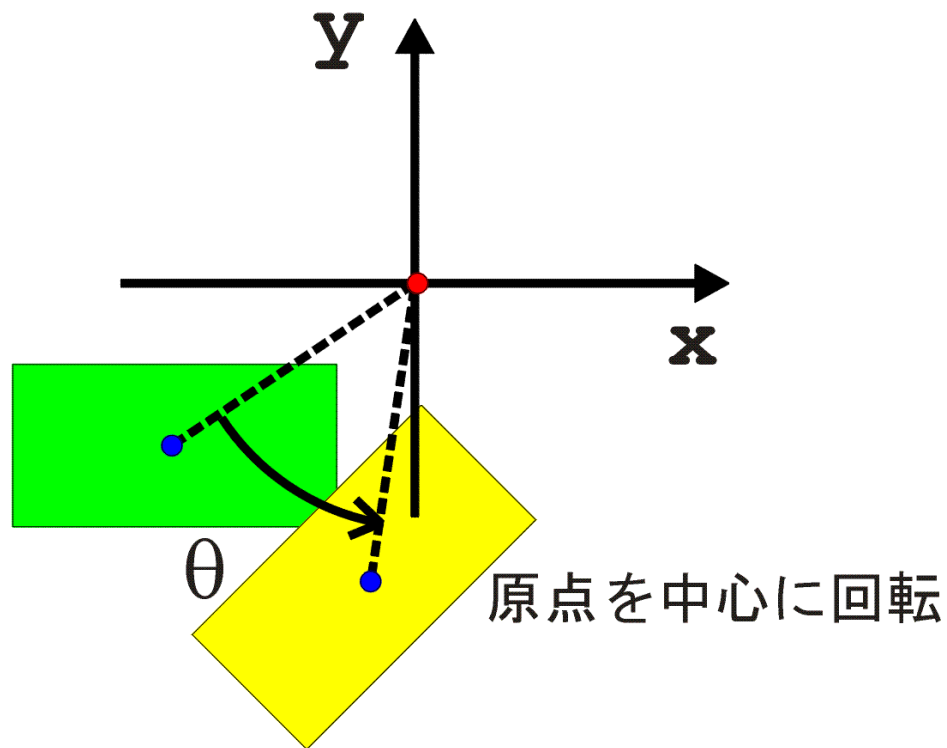
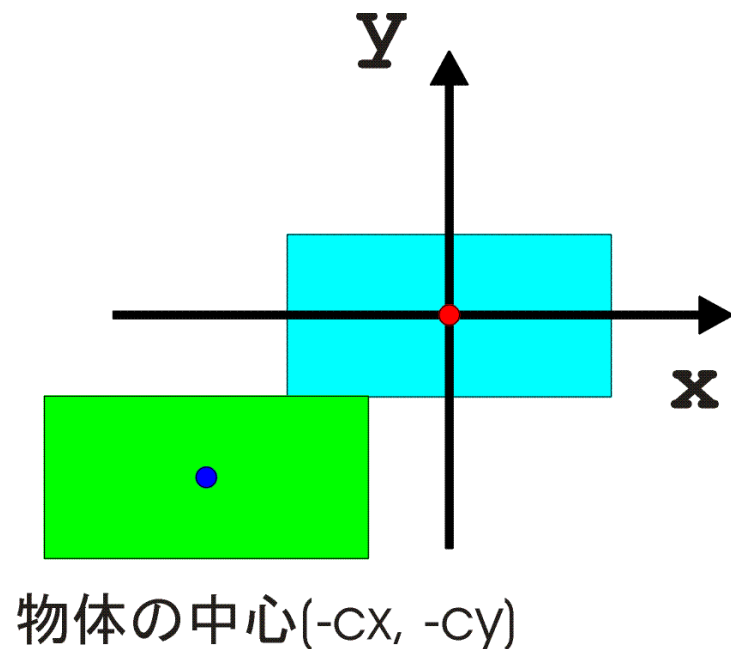
ローカル座標系における回転



「回転の中心(c_x, c_y)」回りに物を回転させたい.



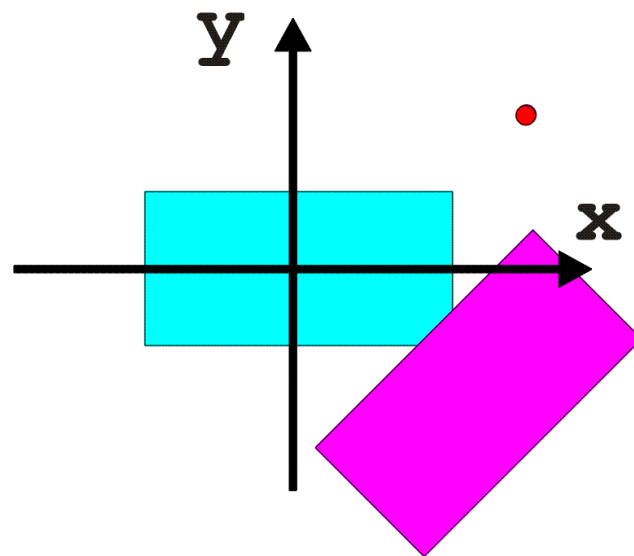
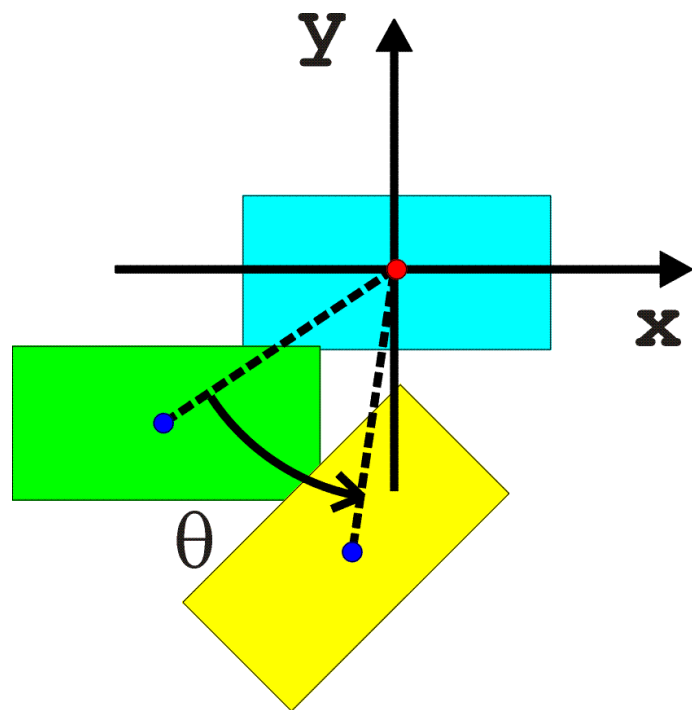
グローバル座標系における回転(その1)



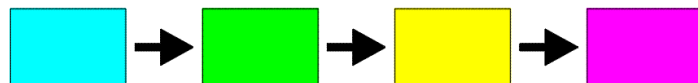
1. $(-cx, -cy)$ 平行移動
2. 原点回りに回転



グローバル座標系における回転(その2)



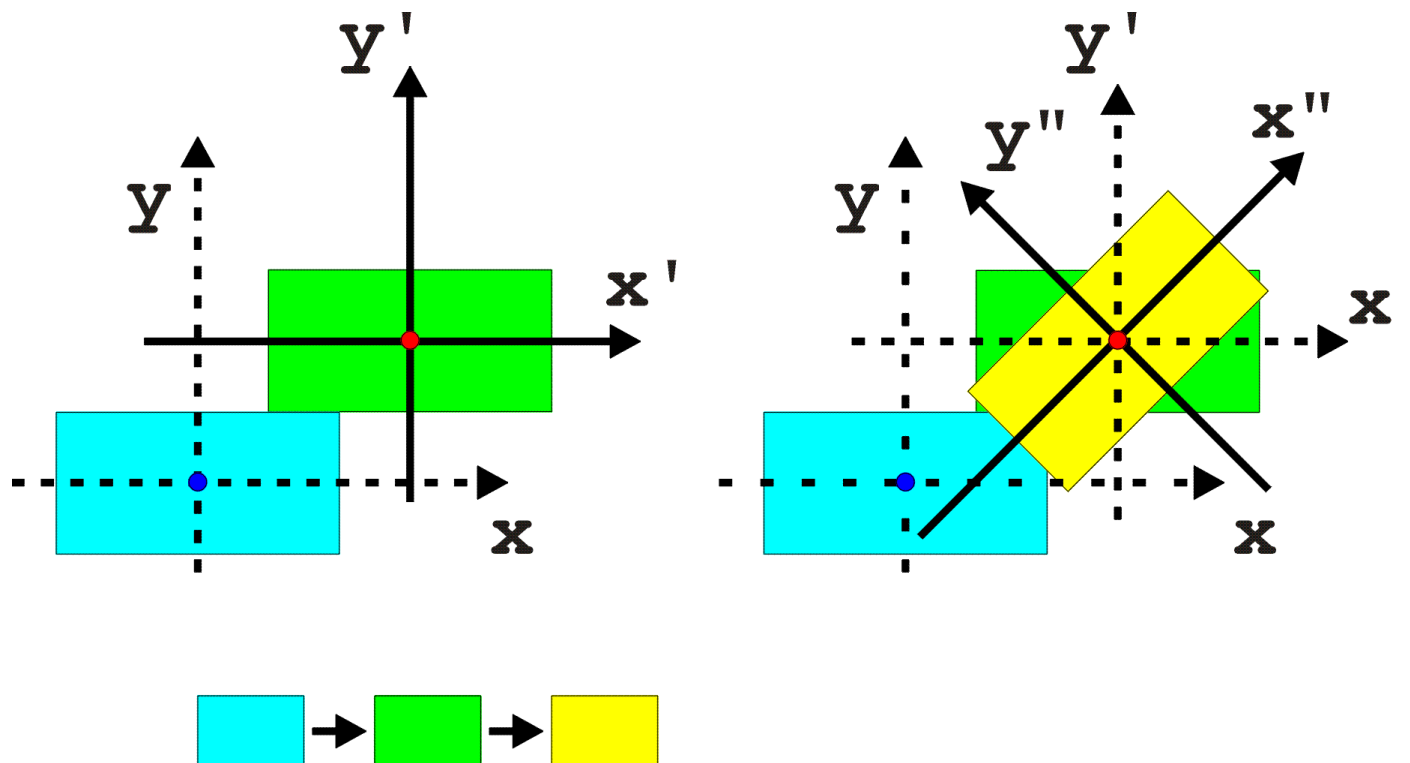
(cx, cy) 物体を平行移動



3. (cx, cy) 平行移動



ローカル座標系における回転(その1)

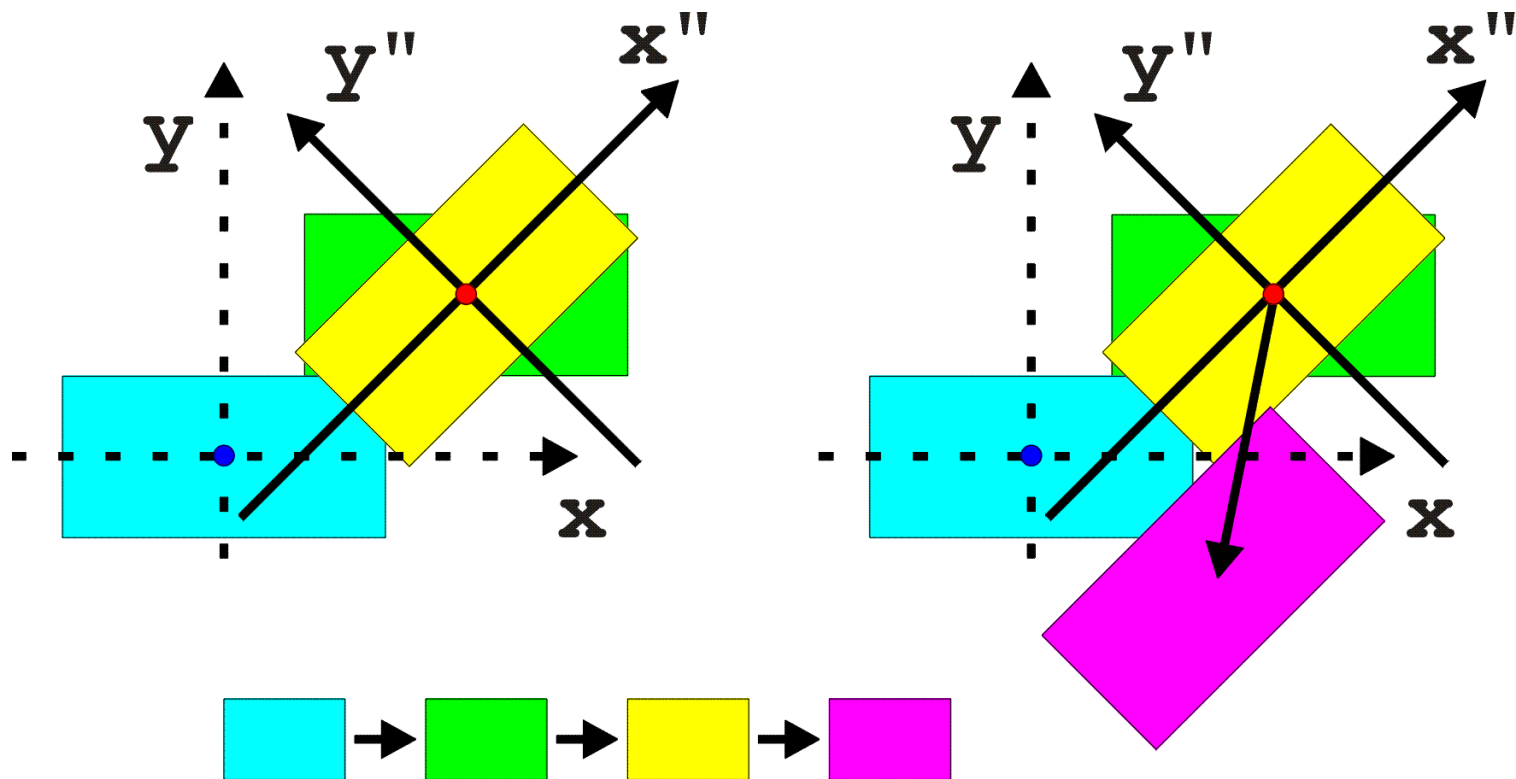


グローバル座標系の場合の処理の順序を入れ換え,

1. (cx, cy) 平行移動
2. 原点回りに回転



ローカル座標系における回転(その2)



3. $(-cx, -cy)$ 平行移動



グローバル・ローカル座標系の関係

グローバル座標系 -> ローカル座標系



行列の掛ける順序を入れ変える.

$$M_{global} = (M_1)_{global} (M_0)_{global}$$

$$M_{local} = (M_0)_{local} (M_1)_{local}$$



グローバル・ローカル座標系の行列

$$M_{global} = (M_1)_{global} (M_0)_{global}$$

$$\begin{aligned} M_{local} &= ((M_0)_{local} (M_1)_{local} (M_0)_{local}^{-1}) (M_0)_{local} \\ &= (M_0)_{local} (M_1)_{local} \end{aligned}$$



行列のプッシュポップに関する基本課題

課題A

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.



robotModel.h

```
#define PAI                3.1415926

typedef struct s_vertex { /* 頂点の構造体の定義 */
    GLfloat coord[3]; /* (x,y,z)座標 */
} vertex;

typedef struct s_edge { /* 稜線の構造体の定義 */
    vertex *start_vertex; /* 始点 */
    vertex *end_vertex; /* 終点 */
    struct s_edge *right_edge; /* 右側稜線 */
    struct s_edge *left_edge; /* 左側稜線 */
    struct s_face *right_face; /* 右側稜線 */
    struct s_face *left_face; /* 左側稜線 */
} edge;

typedef struct s_face { /* 面の構造体の定義 */
    edge *edge; /* 面に属する稜線 */
    struct s_face *next; /* 次の面へのポインタ */
} face;

typedef struct s_body { /* 物体の構造体の定義 */
    face *face; /* 物体に属する面 */
} body;

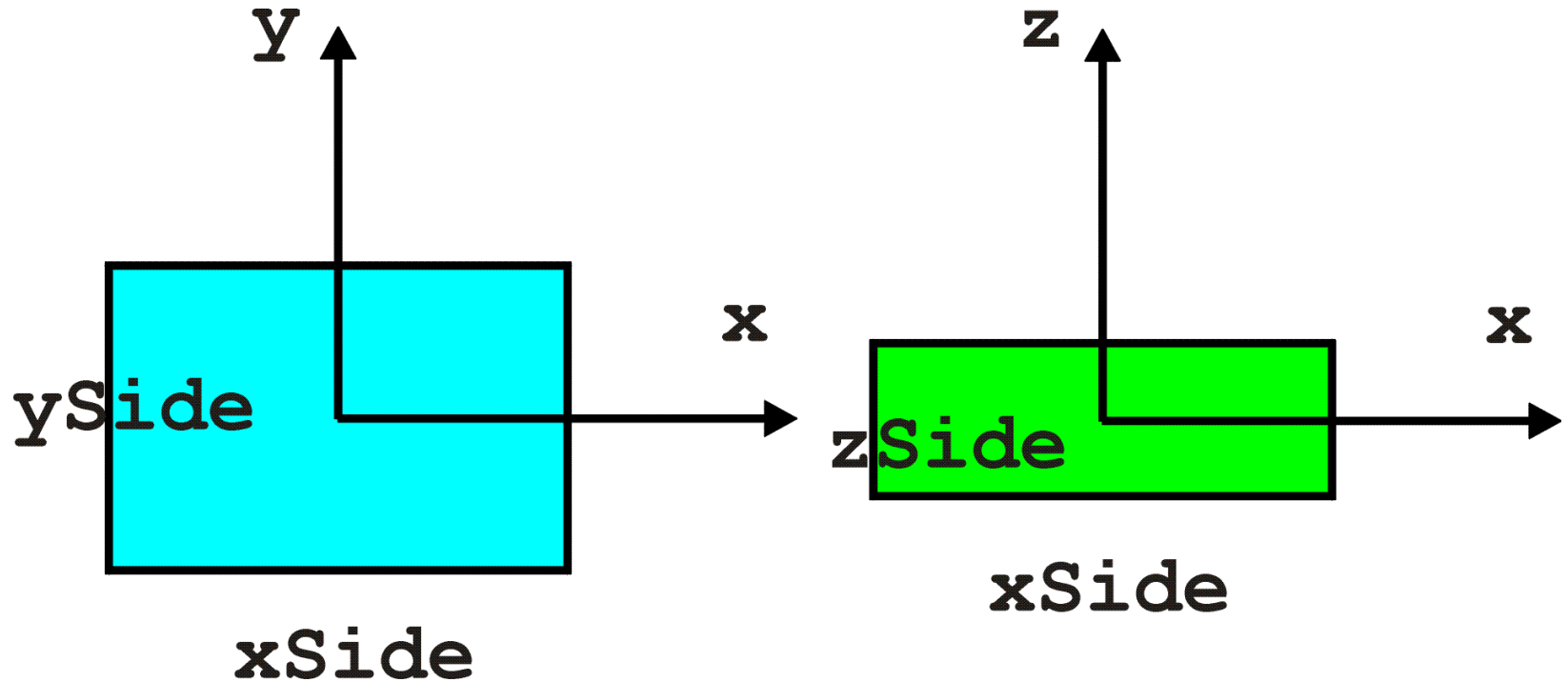
#define MAX_NO_BODY      10
#define MAX_NO_FACE      50
#define MAX_NO_EDGE      100
#define MAX_NO_VERTEX    100

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

extern int       No_Body; /* 物体の数 */
```



直方体の座標系



直方体の中心(重心)を原点

回転中心は直方体の座標系で指定



readRobotData()

```
void
readRobotData(void)
{
    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 */

    ...
}
```



ourDisplay()

```
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 */
    /* バッファのクリア */
    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();
}
```



drawArmShading()

```
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 ) {
        glVertexData( 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*/
}
```

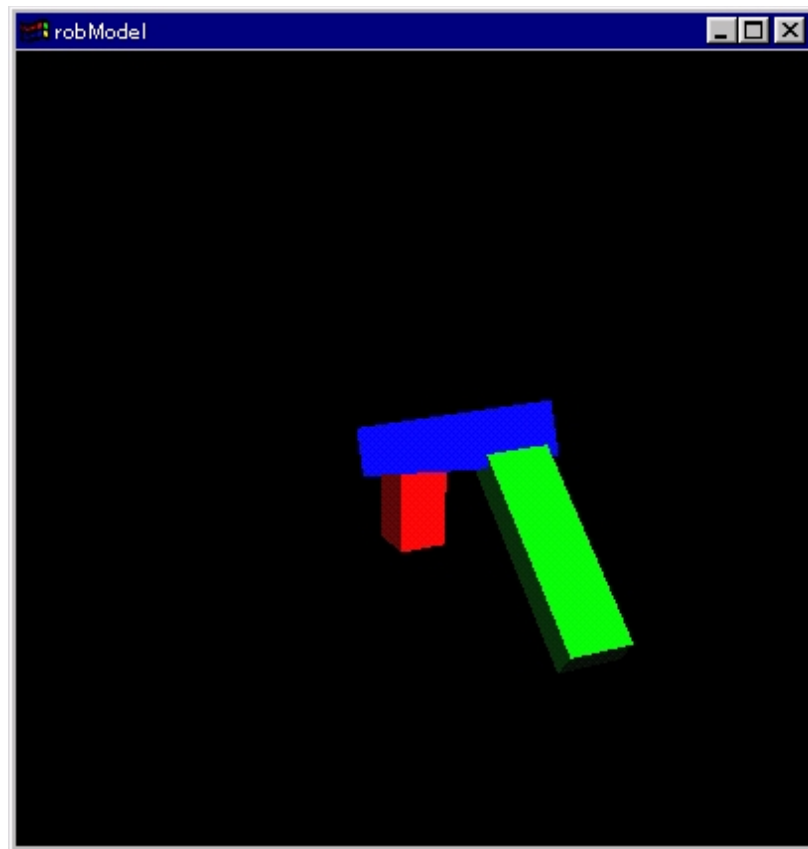


ourKeyboard()

```
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();
}
```



実行例



まとめ

- OpenGL
 - 複数パーツから構成されるモデルの動作
 - 行列のプッシュとポップ

