

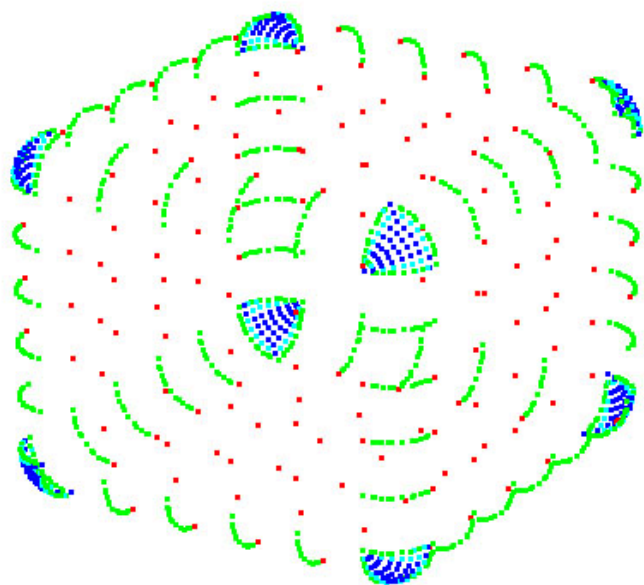
情報工学

2022年度後期 第5回 [11月2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



講義アウトライン [11月2日]

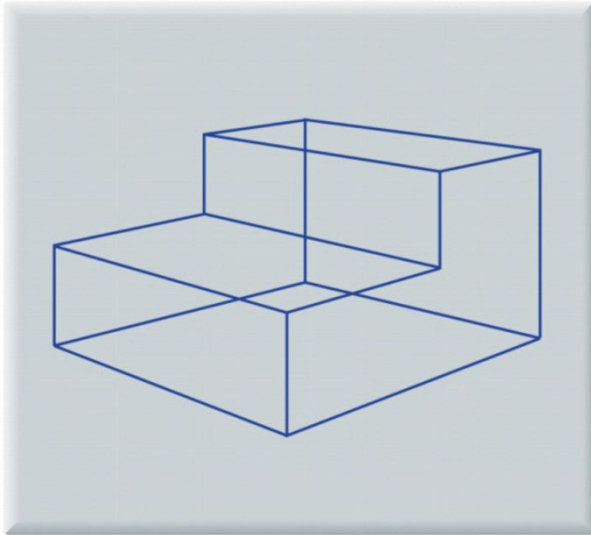
- ビジュアル情報処理
 - 2 モデリング
 - 2.1 形状モデル
 - 2.2 ソリッドモデルの形状表現
- OpenGL
 - 投影変換
 - メニューの作成

ビジュアル情報処理

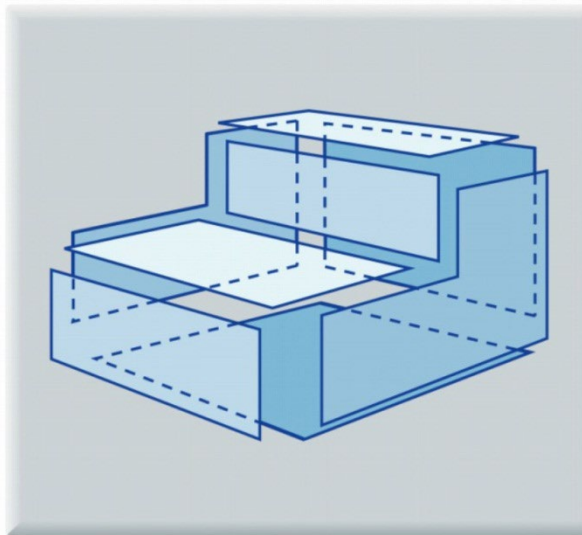
・2 モデリング

・2-1 形状モデル

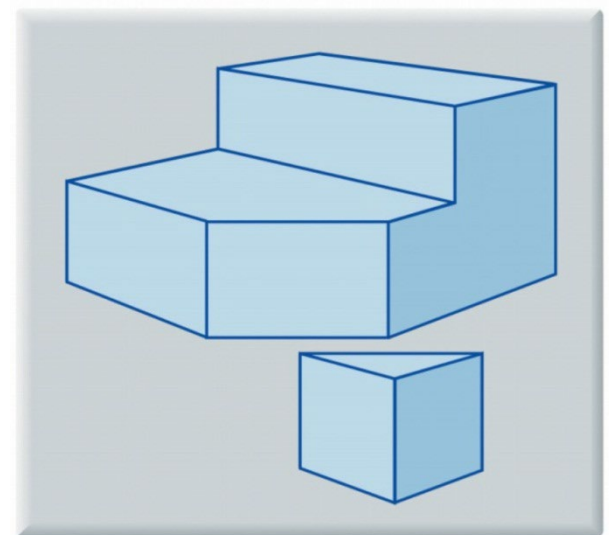
■図3.1—ワイヤフレームモデル



■図3.2—サーフェスモデル



■図3.3—ソリッドモデル

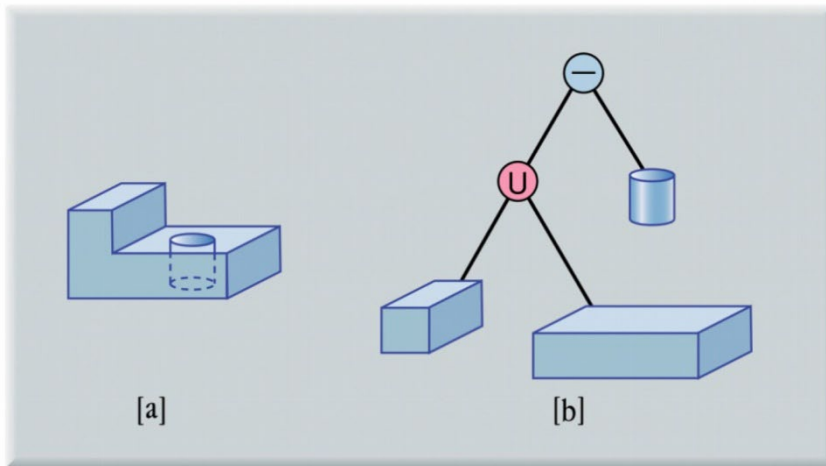


ビジュアル情報処理

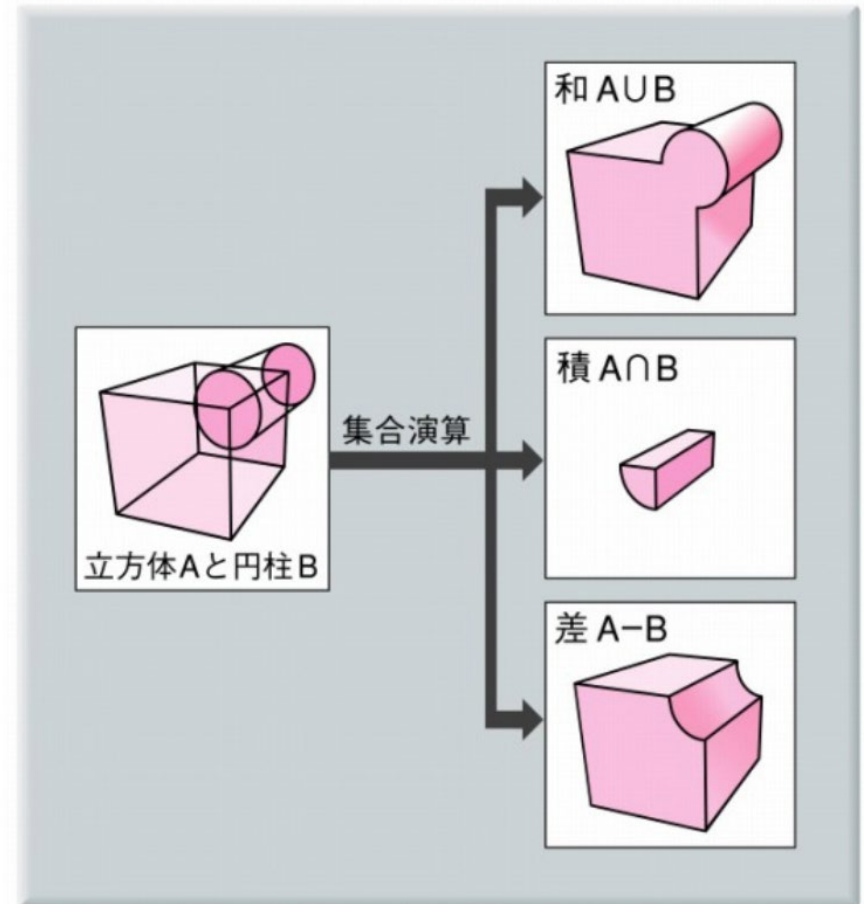
•2-2 ソリッドモデルの形状表現

•2-2-1 CSG表現

■図3.4——CSG表現の考え方



■図3.5——集合演算の例

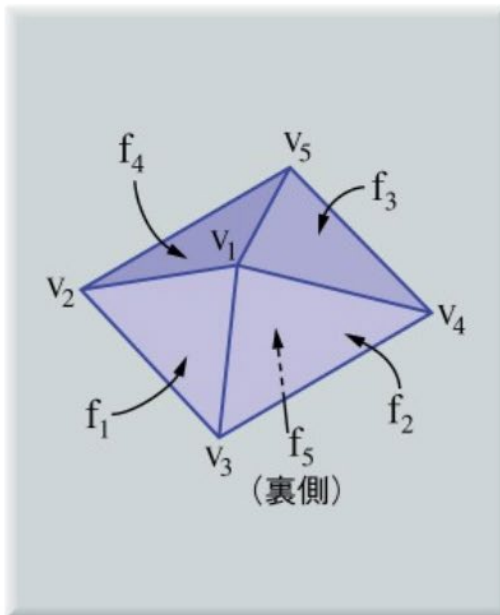


ビジュアル情報処理

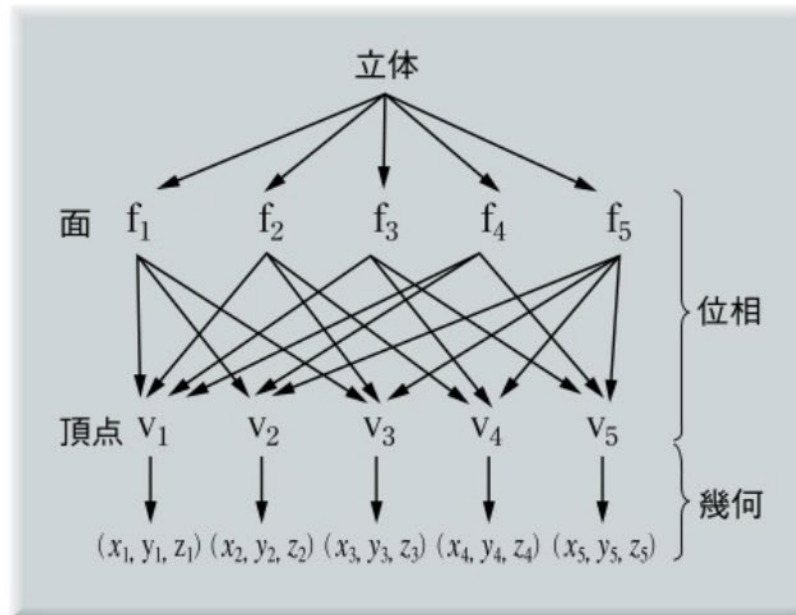
•2-2 ソリッドモデルの形状表現

•2-2-2 境界表現

■ 図3.6——境界表現



[a] 四角錐



[b] 形状表現の例

| | |
|-------|----------------------|
| f_1 | v_1, v_2, v_3 |
| f_2 | v_1, v_3, v_4 |
| f_3 | v_1, v_4, v_5 |
| f_4 | v_1, v_5, v_2 |
| f_5 | v_2, v_5, v_4, v_3 |
| v_1 | (x_1, y_1, z_1) |
| v_2 | (x_2, y_2, z_2) |
| v_3 | (x_3, y_3, z_3) |
| v_4 | (x_4, y_4, z_4) |
| v_5 | (x_5, y_5, z_5) |

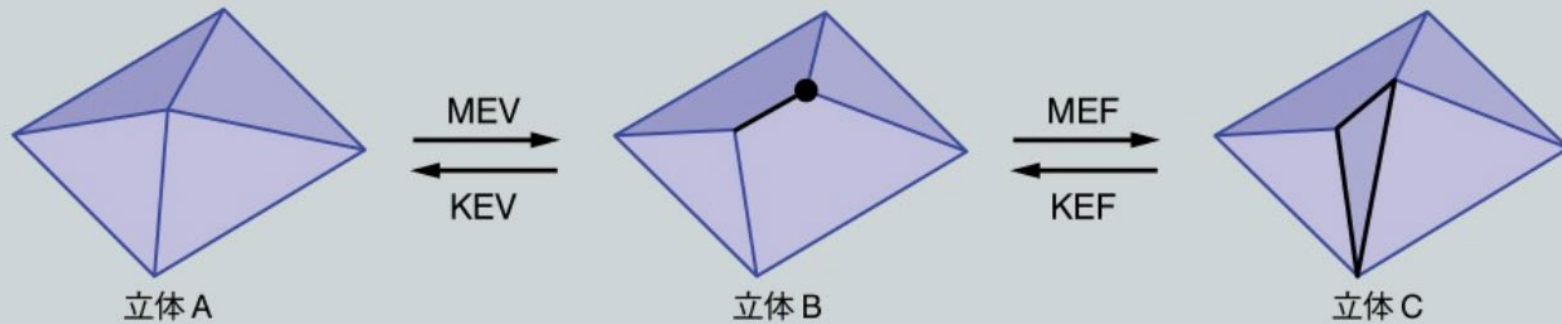
[c] コンピュータ上のデータ構造の例

ビジュアル情報処理

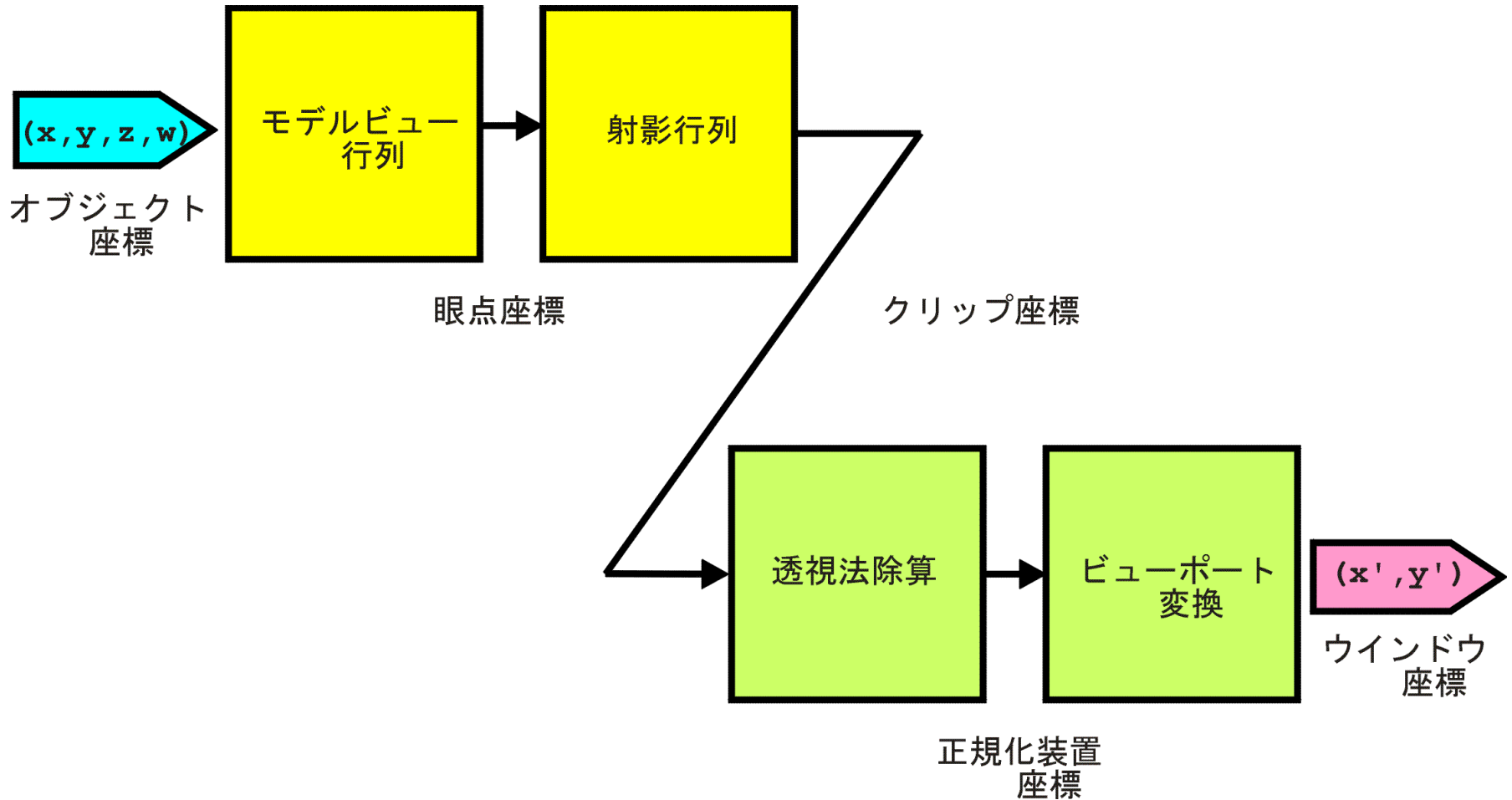
•2-2-2 境界表現

•オイラー操作 $v-e+f=2$

■図3.7——オイラー操作の例

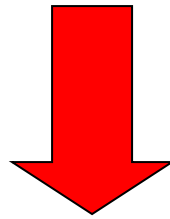


頂点変換の手順(復習)



射影変換の種類(復習)

3D図形をそのままの形で2Dディスプレイに表示することはできない。



3D図形を2D図形に変換 **射影変換**

「正射影変換 (orthographic projection)」

「透視変換 (perspective projection)」

正射影変換と透視変換(復習)

「正射影変換 (orthographic projection)」

1. 無限の位置から立体を見た場合に相当する射影変換
2. 視点と図形との相対的な位置関係とは無関係

「透視変換 (perspective projection)」

1. 視点から図形が離れれば離れるほど小さく変換
2. 見える領域(視体積)がピラミッド型

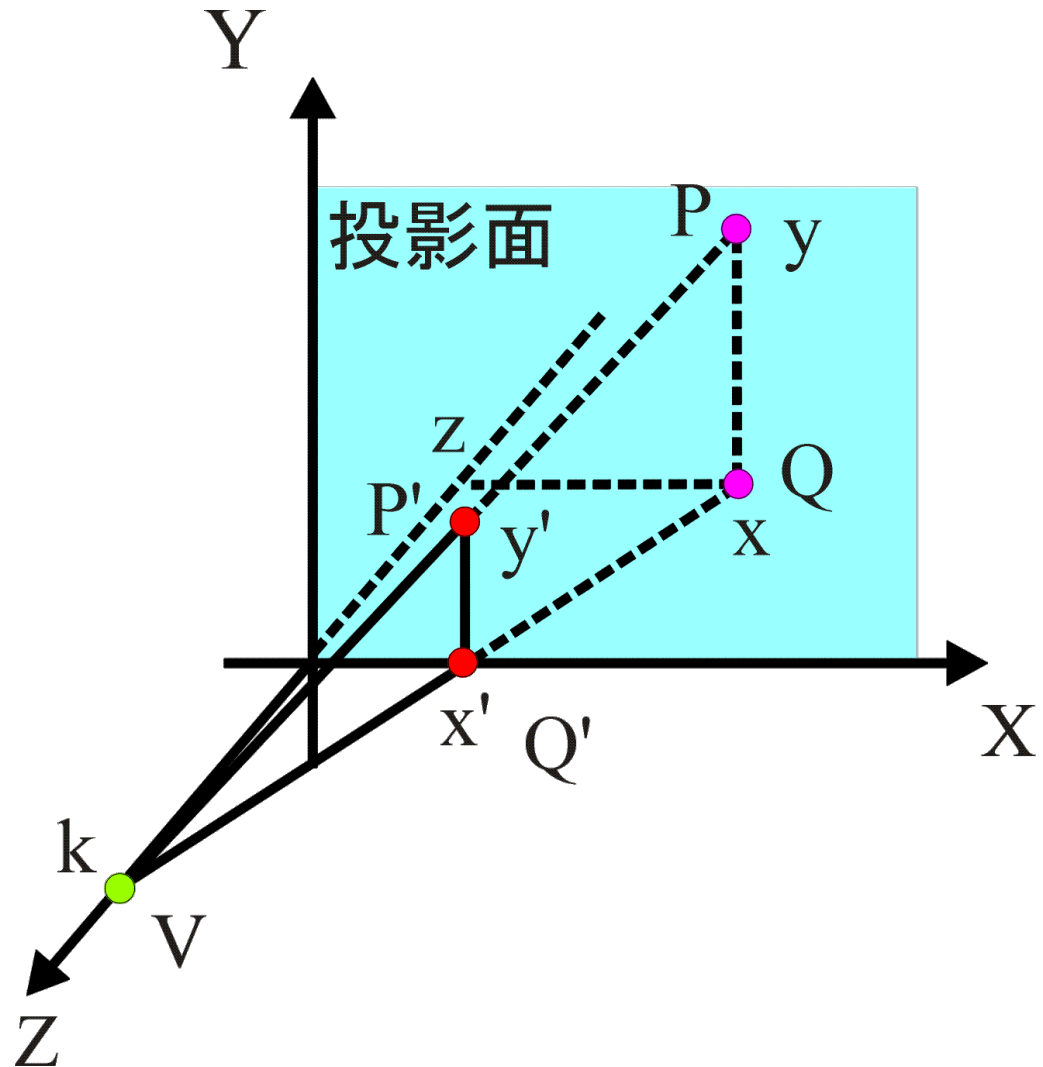
透視変換の計算(復習)

視点 $V=(0,0,k)$

$$\frac{x'}{k} = \frac{x}{-z+k}$$

$$x' = \frac{x}{1 - \frac{z}{k}}$$

$$y' = \frac{y}{1 - \frac{z}{k}}$$



透視変換を表す行列(復習)

視点 $V=(0,0,k)$

$$\frac{x'}{k} = \frac{x}{-z+k}$$

$$x' = \frac{x}{1 - \frac{z}{k}}$$

$$y' = \frac{y}{1 - \frac{z}{k}}$$

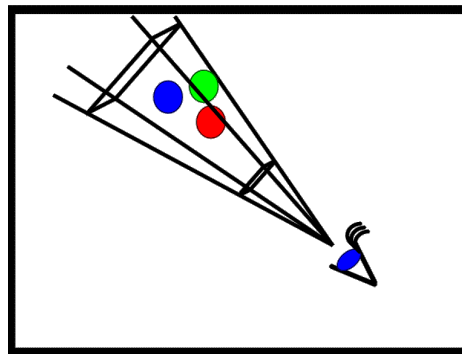
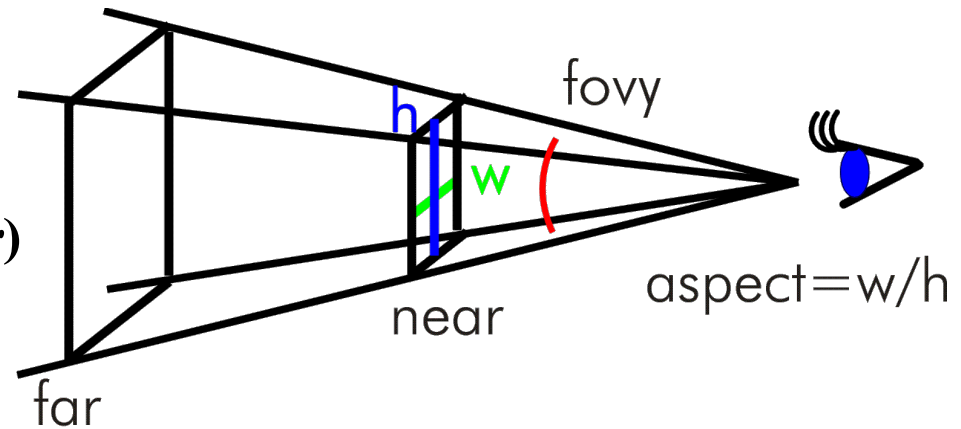
$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{k} & 1 \end{bmatrix}$$

透視変換:gluPerspective() (復習)

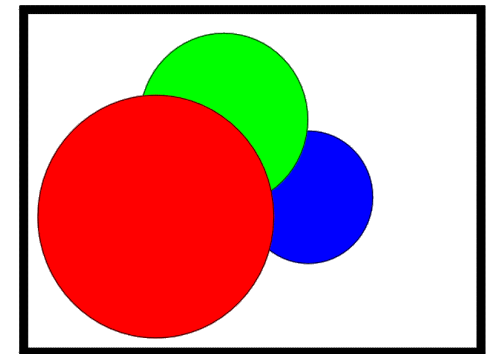
コマンド **gluPerspective()**

void

`gluPerspective(GLdouble fovy,`
`GLdouble aspect,`
`GLdouble near, GLdouble far)`



視体積の位置



視点からの図



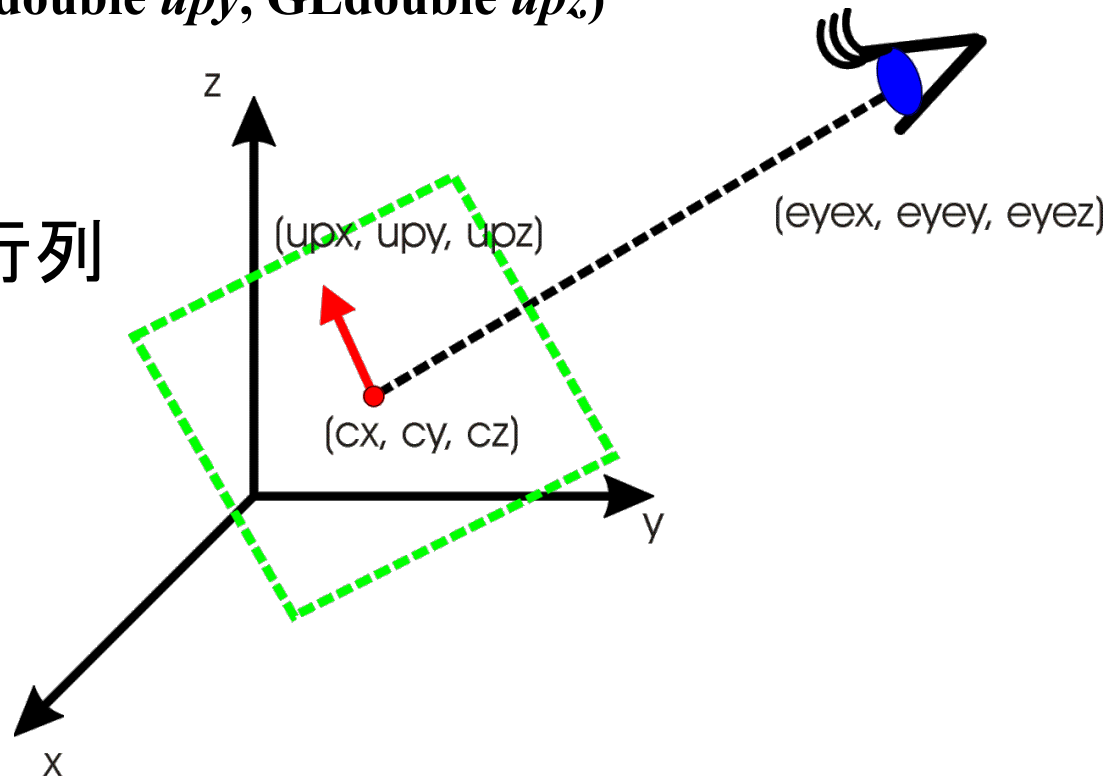
視界変換:gluLookAt() (復習)

コマンド **gluLookAt()**

void

`gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
GLdouble centerx, GLdouble centery, GLdouble centerz,
GLdouble upx, GLdouble upy, GLdouble upz)`

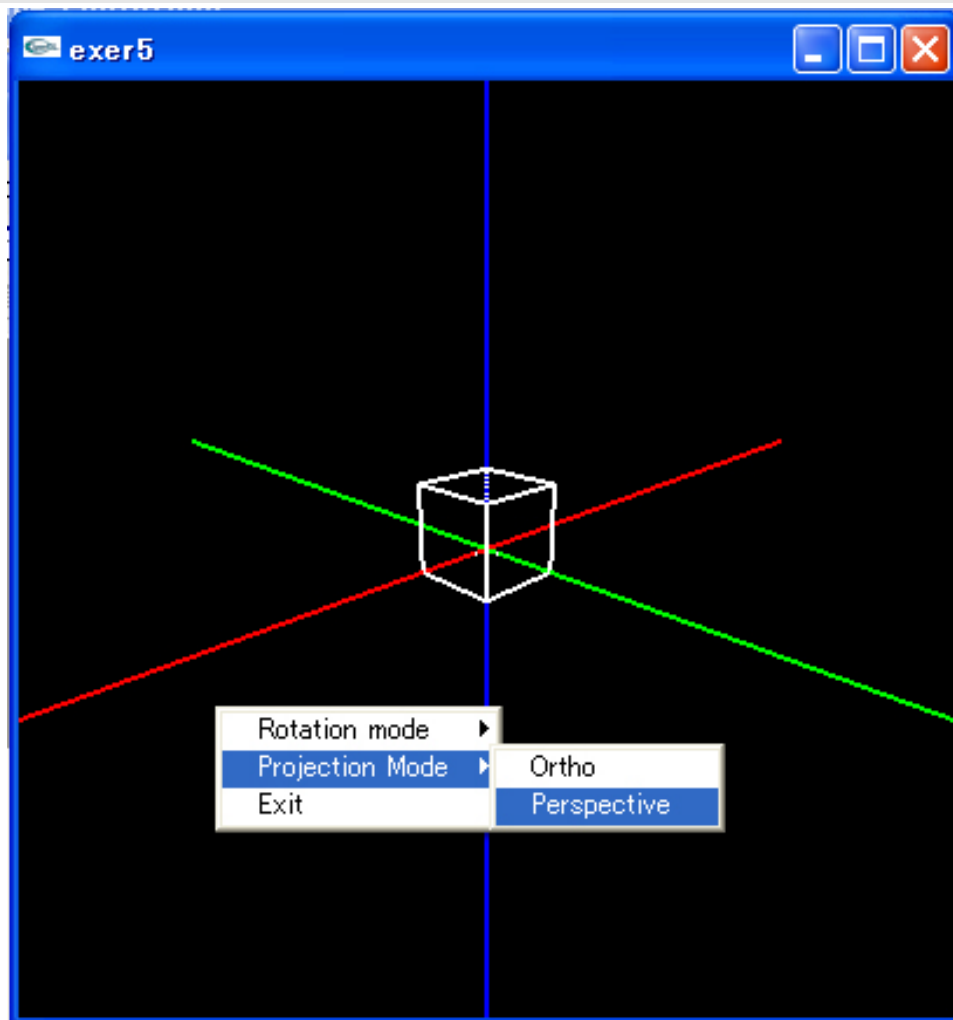
注意:モデルビュー行列
を変更する.



ウィンドウサイズの変更

```
int
void ourReshape(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(30.0,
                  (GLfloat)width / (GLfloat)height,
                  10.0, 1000.0);
}
```

課題5サンプル(実行結果)



ソースコードは授業用ホームページを参照すること。

第5回課題

```
#include <stdlib.h>
#include <math.h>
#include <GL/glut.h>
#include <stdio.h>

/* 視点と注視点のデータ */
static GLdouble eye[3] = { 45.0, 45.0, 25.0 }; /*視点の位置*/
static GLdouble center[3] = { 0.0, 0.0, 0.0 }; /*注視点*/
static GLdouble up[3]; /* ビューアップベクトル */

/* 視点の回転角 */
static int spin_eye = 0;
static int globalWidth;
static int globalHeight;

/* 視点の正方向への回転 */
void rotEyePlus(void) {
    spin_eye = ( spin_eye + 15 ) % 360;    /* 15° 加える */
}

/* 視点の負方向への回転 */
void rotEyeMinus(void) {
    spin_eye = ( spin_eye - 15 ) % 360;    /* 15° 差し引く */
}
```


第5回課題(続き)

```
static int projection=1;

/*立方体の描画*/
void drawCube() {
    float vertex[8][3]={{0.,0.,0.},{10.,0.,0.},{10.,10.,0.},{0.,10.,0.},
        {0.,0.,10.},{10.,0.,10.},{10.,10.,10.},{0.,10.,10.}};
    int i;
    glBegin(GL_LINE_LOOP);
    for(i=0;i<4;++i){
        glVertex3fv(vertex[i]);
    }
    glEnd();
    glBegin(GL_LINE_LOOP);
    for(i=0;i<4;++i){
        glVertex3fv(vertex[i+4]);
    }
    glEnd();
    glBegin(GL_LINES);
    for(i=0;i<4;++i){
        glVertex3fv(vertex[i]);
        glVertex3fv(vertex[i+4]);
    }
    glEnd();
}
```

第5回課題(続き2)

```
/* 座標軸の描画 */
void drawAxis() {
    /* x軸(レッド) */
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
        glVertex3f(-100.0f, 0.0f, 0.0f);
        glVertex3f(100.0f, 0.0f, 0.0f);
    glEnd();

    /* y軸(グリーン) */
    glColor3f(0.0f, 1.0f, 0.0f);
    glBegin(GL_LINES);
        glVertex3f(0.0f, -100.0f, 0.0f);
        glVertex3f(0.0f, 100.0f, 0.0f);
    glEnd();

    /* z軸(ブルー) */
    glColor3f(0.0f, 0.0f, 1.0f);
    glBegin(GL_LINES);
        glVertex3f(0.0f, 0.0f, -100.0f);
        glVertex3f(0.0f, 0.0f, 100.0f);
    glEnd();
}
```

第5回課題(続き3)

```
void ourDisplay(void) {

    /* バッファのクリア */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    /* モデルビュー行列 */
    glMatrixMode ( GL_MODELVIEW );
    glLoadIdentity ();
    /* 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]);
              /*視点の設定 (視点、注視点、方向) */

    /* 視点の回転移動 */
    glRotated ((GLdouble)spin_eye, 0.0, 0.0, 1.0 );/*z軸正の方向にspin_eye度回転*/

    glColor3f(1.,1.,1.);
    glLineWidth(2.);
    drawCube ();          /* drawCube() を呼び出す */

    /* x, y, z軸の描画 */
    drawAxis ();

    glFlush();
}
```

第5回課題(続き4)

```
void ourInit (void) {
    glClearColor(0.0, 0.0, 0.0, 1.0);           /*背景色の指定*/
    glDepthFunc ( GL_LESS );
    /*デプステストのための比較関数GL_LESS (より手前のフラグメントを描画) */
    glEnable ( GL_DEPTH_TEST );               /*デプステストを有効にする*/
}

/*
 * ウィンドウが最初にオープンした時やウィ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 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 */

    globalWidth=width;
    globalHeight=height;

    glViewport (0, 0, width, height);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();
```

第5回課題(続き5)

```
if(projection==2){ /* 透視変換 */
    gluPerspective(60.0, (GLfloat) width/(GLfloat) height, 0.1, 1000.0);
}
else{ /* 正射影変換 */
    glOrtho(-50.,50.,-50.*(GLfloat) height/(GLfloat) width,50.*(GLfloat)
        height/(GLfloat) width,-100.,1000.);
}

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;
}
```

第5回課題(続き6)

```
void main_menu(int value)                                /* メインメニュー */
{
    switch(value) {
        case 999:
            exit(1);
            break;
    }
    glutPostRedisplay();
}

void Rotate(int value)    /* 視点回転 */
{
    switch(value) {
        case 1:
            rotEyePlus();
            break;
        case 2:
            rotEyeMinus();
            break;
    }
    glutPostRedisplay();
}
```

第5回課題(続き7)

```
/* メイン */
int main(int argc, char **argv)
{
    int submenu1, submenu2;
    ... (省略)
    glutReshapeFunc(ourReshape);
    glutDisplayFunc(ourDisplay);

    submenu1=glutCreateMenu(Rotate);      /*Rotate eye のサブメニューの作成 */
    glutAddMenuEntry("positive",1);
    glutAddMenuEntry("negative",2);

    submenu2=glutCreateMenu(Projection); /* Projection のサブメニュー*/
    glutAddMenuEntry("Ortho",3);
    glutAddMenuEntry("Perspective",4);

    glutCreateMenu(main_menu);           /* メインメニュー */
    glutAddSubMenu("Rotation mode",submenu1);
    glutAddSubMenu("Projection Mode",submenu2);/* Projection Mode の作成 */
    glutAddMenuEntry("Exit",999);       /* Exit の作成 */
    glutAttachMenu(GLUT_RIGHT_BUTTON);/* マウス右クリックでポップアップメニュー */
    glutMainLoop();
    return 0;
}
```

課題5

課題A

四角錐(ピラミッド)を描画する関数`drawPyramid()`を作成し、それを`drawCube()`の代わりに呼び出せ。

課題B

サブメニューを追加して、立方体とピラミッドの描画を変更できるようにせよ。

Hint.

1. `main`関数に`int submenu3;`を追加する。
2. `main`関数に`submenu3=glutCreateMenu(Geometry);`を追加する。
3. メニューエントリーを追加する。
4. `Rotate()`や`Projection()`と同様に、関数`Geometry()`を追加する。
5. `glutAddSubMenu("Change geometry",submenu3);`

課題C (時間に余裕のある人のために)

立方体やピラミッドに対して、回転、平行移動、およびスケールを組み合わせて意味のある形状や模様を作成せよ。

まとめ

- ビジュアル情報処理

- 2 モデリング

- 2.1 多面体

- 2.2 ソリッドモデルの形状表現

- OpenGL

- 投影変換

- メニューの作成