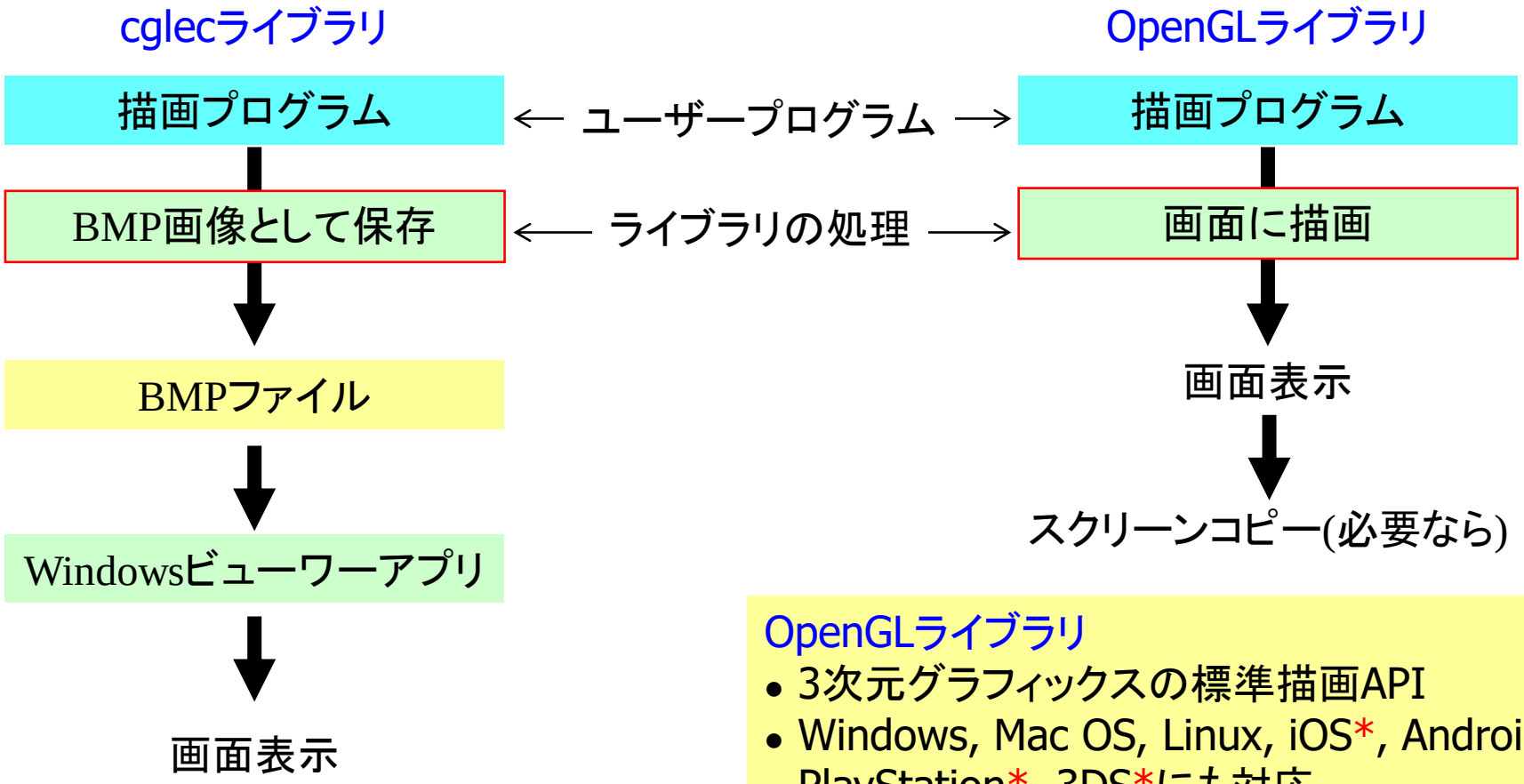
OpenGLライブラリ

復習



OpenGLライブラリ
- 3次元グラフィックスの標準描画API
- Windows, Mac OS, Linux, iOS*, Android*, PlayStation*, 3DS*にも対応
- 3次元CGの高度な描画機能
- C言語ライブラリ
- 参考書, 参考サイトがたくさんある

*正確にはOpenGL ESが使用されている

OpenGLを用いたプログラム

復習

Example9-1

モデルの形を決める

表示方法を決める

```
#include "glut.h"
#include <GL/gl.h>

void display(void)
{
    glClear( GL_COLOR_BUFFER_BIT );

    glColor3f(1.0, 1.0, 1.0);
    glBegin( GL_LINES );
        glVertex3f(-1.0, -1.0, 0.0);
        glVertex3f(+1.0, +1.0, 0.0);
    glEnd();

    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("初めてのOpenGLプログラム");
    glClearColor (0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);

    glutDisplayFunc(display);
    glutMainLoop();
}
```

画面に描画する関数. 重要

ウィンドウのクリア. CglSetAll()と同じ

描画する線の色の設定. ここでは白. R, G, Bの順

直線の**点座標**設定開始

始点の座標を設定

終点の座標を設定

xy平面への平行投影なのでz座標はあってもなくても同じ

点座標設定終了

描画処理！

main()関数はこう書くこと

ライブラリの初期化. おまじない

ウィンドウの最初の位置

ウィンドウの最初の大きさ. 400×400ピクセル.

ウィンドウのモード設定. おまじない.

ウィンドウのタイトル文字列

ウィンドウの最初の色. ここでは黒. **R**, **G**, **B**の順. 階調値は0.0～1.0の範囲　**(1.0, 1.0, 1.0)→白**

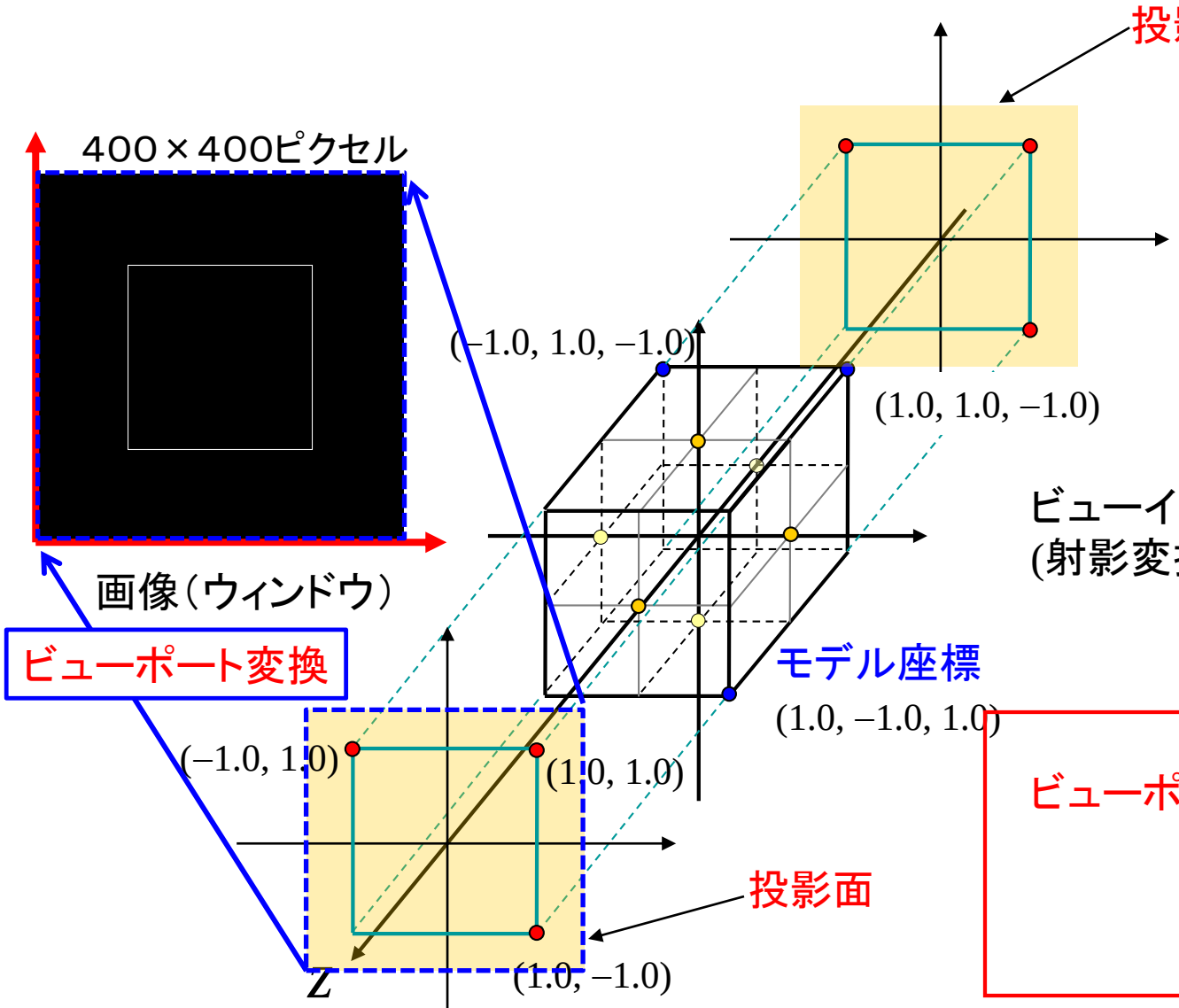
平行投影と投影座標の設定(次の順番)
*x*座標の最小値: –2, *x*座標の最大値: 2,
*y*座標の最小値: – 2, *y*座標の最大値: 2

この範囲でクリッピングされる

ウィンドウ制御. おまじない

モデル座標からCG画像までの変換(平行投影の例)

復習



モデル座標と
投影面座標
はほぼ同じ

モデル座標
(3次元実数座標)

ビューイング変換
(射影変換)
⇩
平行投影
透視投影

投影面座標
(2次元実数座標)

ビューポート変換
⇩
拡大と移動

画像の座標
(2次元整数座標)

OpenGLで自動化⇒投影面がぴったりウィンドウに入るようにビューポート変換される

復習

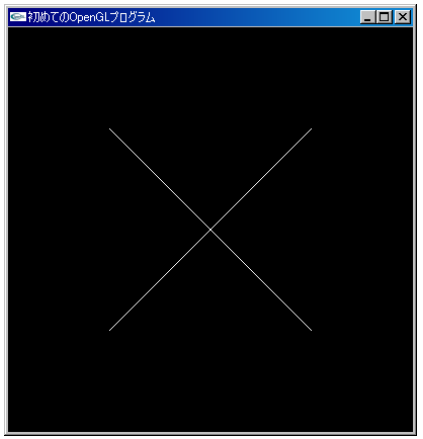
描画関数(1)

Vertex = 頂点

Example9-1a

```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );
    glBegin( GL_LINES );
        glVertex3f(-1.0, -1.0, 0.0);   //始点
        glVertex3f(+1.0, +1.0, 0.0);   //終点
        glVertex3f(-1.0, +1.0, 0.0);   //始点
        glVertex3f(+1.0, -1.0, 0.0);   //終点
    glEnd();
    glFlush();
}
```

GL_LINES
頂点座標を2点づつ指定して線分を描画する. 線分は何本あっても良い. DrawLines()と似た動作

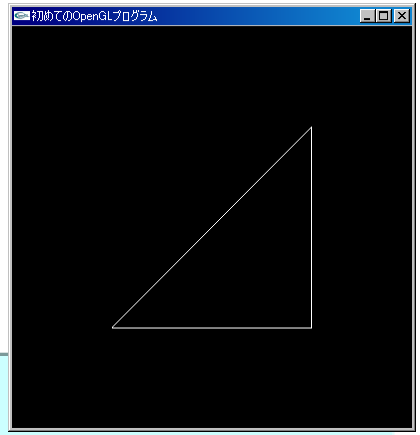


glVertex3f()関数
頂点座標を指定する

Example9-1b

```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );
    glBegin( GL_LINES );
        glVertex3f(-1.0, -1.0, 0.0);   //始点
        glVertex3f(+1.0, -1.0, 0.0);   //終点
        glVertex3f(+1.0, -1.0, 0.0);   //始点
        glVertex3f(+1.0, +1.0, 0.0);   //終点
        glVertex3f(+1.0, +1.0, 0.0);   //始点
        glVertex3f(-1.0, -1.0, 0.0);   //終点
    glEnd();
    glFlush();
}
```

必ず,
glBegin()関数と
glEnd()関数の間
で使うこと！



GL_LINE_LOOP
頂点座標が閉じた線図形の頂点であることを指定. 頂点は何点あってもよい.

Example9-1c

```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );
    glBegin( GL_LINE_LOOP );
        glVertex3f(-1.0, -1.0, 0.0);   //1点目
        glVertex3f(+1.0, -1.0, 0.0);   //2点目
        glVertex3f(+1.0, +1.0, 0.0);   //3点目
    glEnd();
    glFlush();
}
```

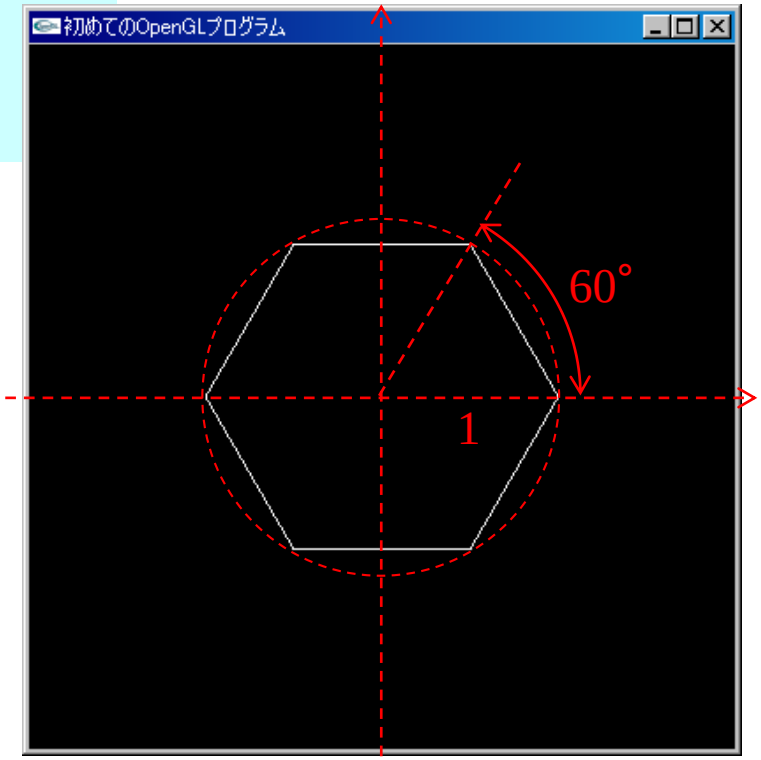
描画関数(2)

復習

Example9-2

```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );

    double angle = 2*3.1415/6; // 60度のラジアン値
    int i;

    glBegin( GL_LINE_LOOP );
        for (i = 0; i < 6; i++)
        {
            glVertex3f(cos(i*angle), sin(i*angle), 0.0);
        }
    glEnd();
    glFlush();
}
```

ループを用いて頂点座標を指定しても良い.



glBegin()関数とglEnd()関数の間で呼び出されたglVertex3f()関数がモデルの頂点を与える.

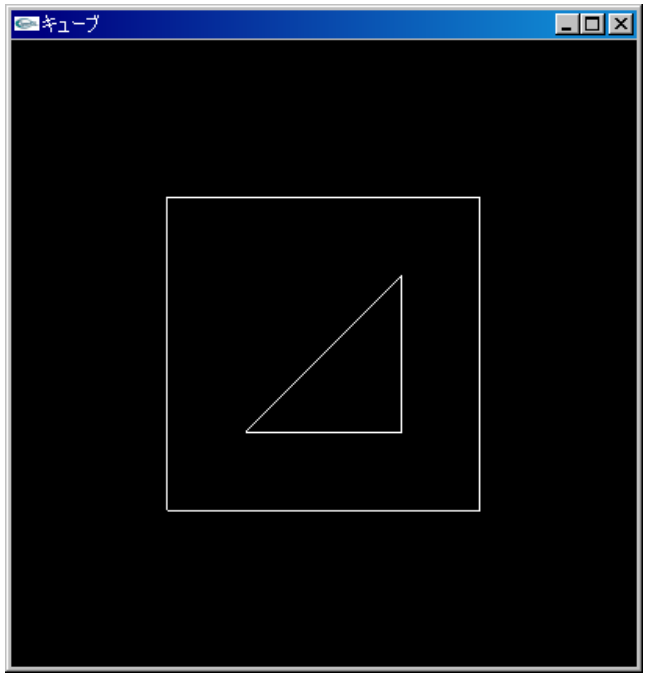
描画関数(3)

復習

Example9-3

```
struct Point3D
{
    double x;
    double y;
    double z;
};

void display( void )
{
    Point3D cube[] = {
    {1, -1, 1}, {1, -1, -1}, {1, -1, -1}, {-1, -1, -1}, //底面
    {-1, -1, -1}, {-1, -1, 1}, {-1, -1, 1},{1, -1, 1},
    {1, 1, 1}, {1, 1, -1}, {1, 1, -1}, {-1, 1, -1},      //上面
    {-1, 1, -1}, {-1, 1, 1}, {-1, 1, 1},{1, 1, 1},
    {1, -1, 1}, {1, 1, 1}, {1, -1, -1}, {1, 1, -1},      //縦線
    {-1, -1, -1}, {-1, 1, -1}, {-1, -1, 1}, {-1, 1, 1}
    };
    int N = 24;

    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );
    glBegin( GL_LINES );          //立方体の描画
        int i;
        for (i = 0; i < N; i++)
            glVertex3f(cube[i].x, cube[i].y, cube[i].z);
    glEnd();

    glBegin( GL_LINE_LOOP );     //平面三角形の描画
        glVertex3f(-0.5, -0.5, 1.0);  //1点目
        glVertex3f(+0.5, -0.5, 1.0);  //2点目
        glVertex3f(+0.5, +0.5, 1.0);  //3点目
    glEnd();
    glFlush();
}
```

基本課題8で作成した立方体モデル



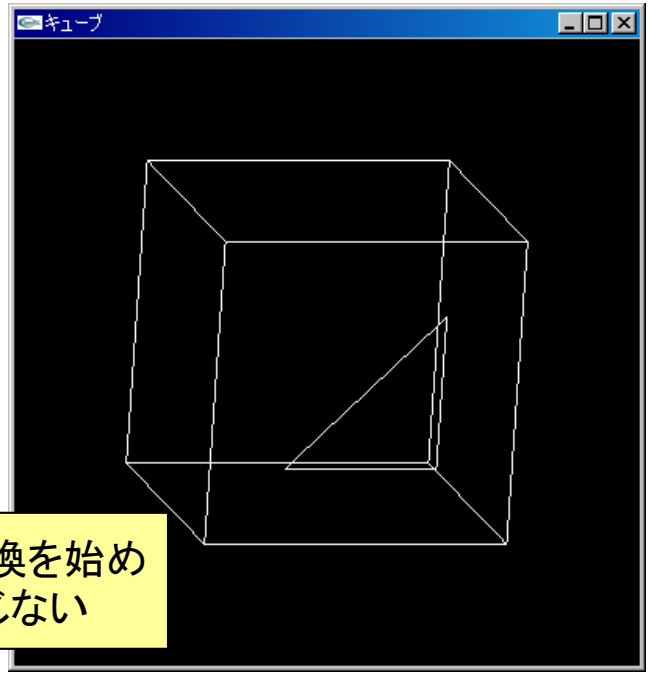
glBegin()・・・glEnd()は二つ以上いくつあってもよい

3次元アフィン変換

復習

Example9-4

```
// display()関数は前スライドと同じ

int main(int argc, char** argv)
{
    glutInit(&argc, argv);

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("キューブ");
    glClearColor (0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);

    glMatrixMode(GL_MODELVIEW);
    glRotatef(15, 0, 1, 0);  //y軸周りに15度回転
    glRotatef(15, 1, 0, 0);  //x軸周りに15度回転

    glutDisplayFunc(display);
    glutMainLoop();
}
```



アフィン変換を始めるおまじない

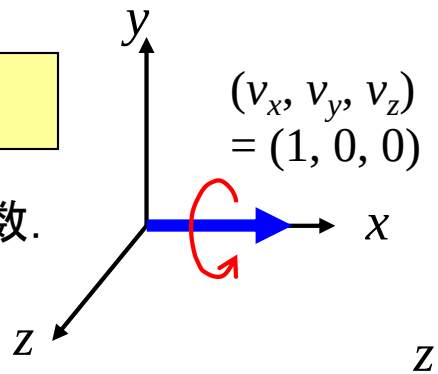
図形を回転するアフィン変換行列の設定

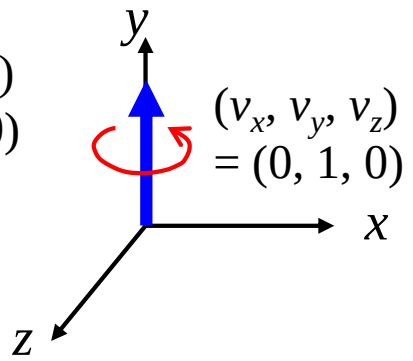
```
glRotatef(angle, vx, vy, vz);
```

(v_x, v_y, v_z)

角度angle[単位：度]の回転変換行列を乗算する関数.
vx, vy, vz：回転軸を表す単位ベクトルの成分

注意
C言語ライブラリではラジアン





基本課題9

復習

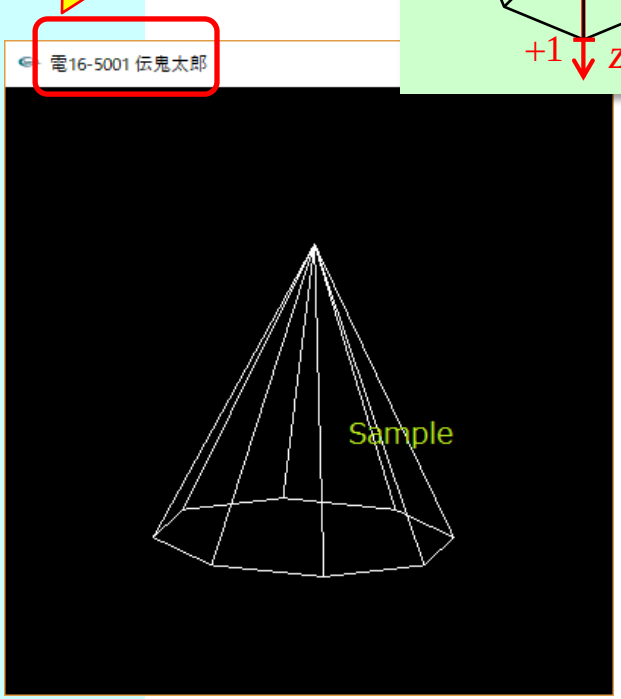
次のmain()をそのまま用いて，OpenGLの描画関数で次の八角錐をプログラムせよ．

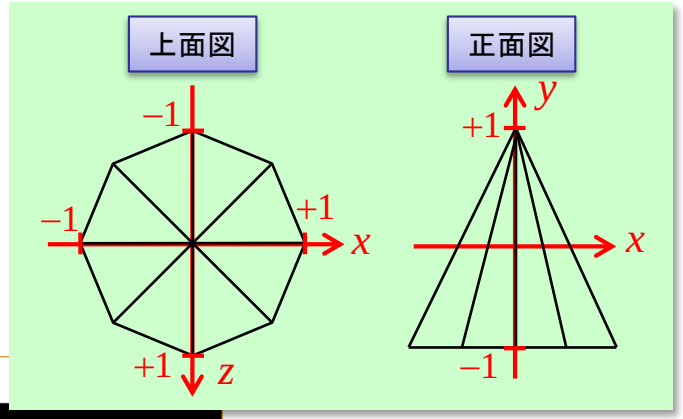
- 八角錐の底面は$(x, -1, z)$平面にある
- 底面の八角形の中心は$(0, -1, 0)$であり，八角形の対角線の長さは2である．
- 頂点の座標は$(0, +1, 0)$である．

Report9-1

```
#include "glut.h"
#include <GL/gl.h>
#include <math.h>

void display(void); // この関数をプログラムする

int main(int argc, char** argv)
{
    glutInit(&argc, argv);

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("電16-5001 伝鬼太郎");
    glClearColor (0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);

    glMatrixMode(GL_MODELVIEW);
    glRotatef(8, 0, 1, 0);
    glRotatef(15, 1, 0, 0);

    glutDisplayFunc(display);
    glutMainLoop();
}
```

ウィンドウのタイトルを学籍番号・氏名にすること





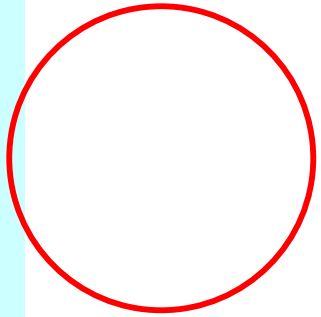
ソースプログラムとglutウィンドウのスクリーンコピーをレポートに貼り付けて提出

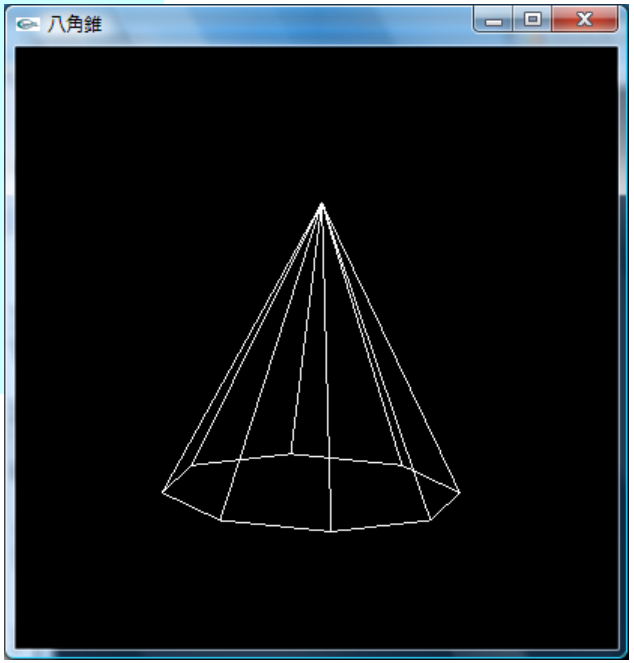
基本課題9 解答例

A君解答

```
void display(void)// この関数をプログラムする
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );

    double angle = 2*3.1415/8; //45度のラジアン値
    int i;

    glBegin( GL_LINES );
    for(i = 0; i < 8; i++)
    {
        glVertex3f(0.0, +1.0, 0.0);                    //始点（頂点）
        glVertex3f(cos(i*angle), -1.0, sin(i*angle));  //終点
    }
    glEnd();
    glFlush();

    glBegin( GL_LINE_LOOP );    //底面
    for(i = 0; i < 8; i++)
    {
        glVertex3f(cos(i*angle), -1.0, sin(i*angle));
    }
    glEnd();
    glFlush();
}
```

glFlush()は最後に
1度行えばよい

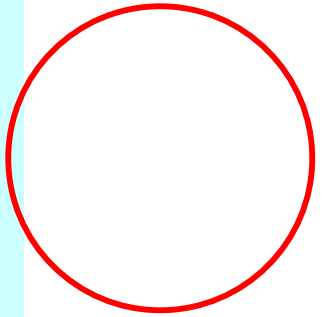


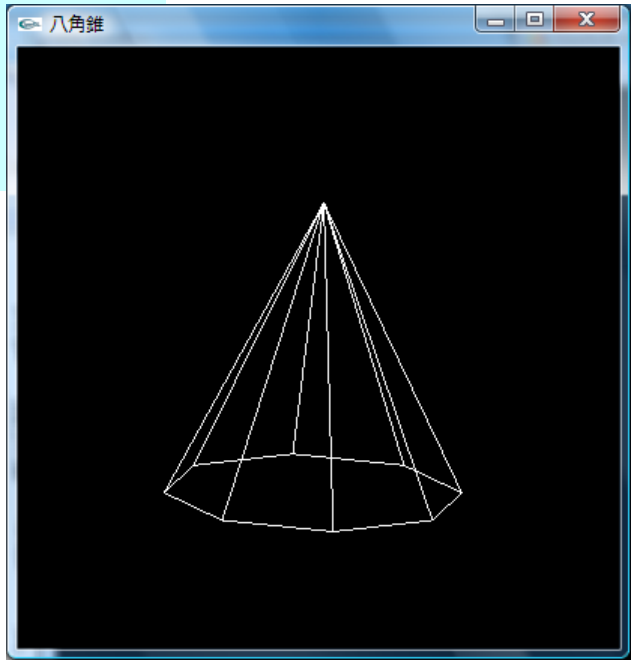


基本課題9 解答例

B君解答

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    int i;


    glBegin(GL_LINES);
    for (i = 0; i < 8; i++)
    {
        glVertex3f(0, 1.0, 0);
        glVertex3f(cos(i*angle), -1, sin(i*angle));
        glVertex3f(cos(i*angle), -1, sin(i*angle));
        glVertex3f(cos((i+1)*angle), -1, sin((i+1)*angle));
    }
    glEnd();
    glFinish();

}
```



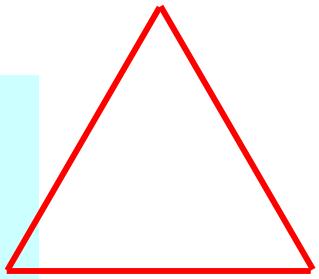


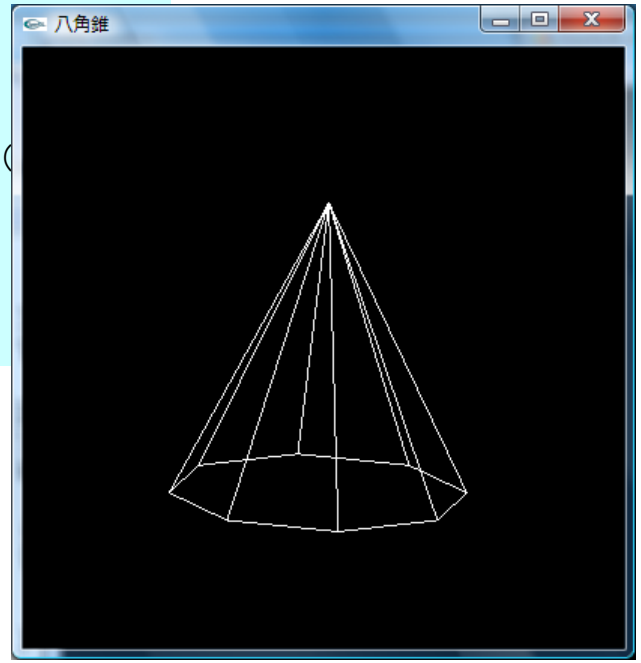
基本課題9 解答例

C君解答

```
void display(void)// この関数をプログラムする
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );

    double angle = 2*3.1415/8; //45度のラジアン値
    int i;

    glBegin( GL_LINE_LOOP );    //底面
    for(i = 0; i < 8; i++)
    {
        glVertex3f(cos(i*angle), -1.0, sin(i*angle));
    }
    glEnd();

    glBegin( GL_LINE_LOOP );
    for(i = 0; i < 8; i++)
    {
        glVertex3f(0.0, +1.0, 0.0);                          //始点 (
        glVertex3f(cos(i*angle), -1.0, sin(i*angle));   //終点
    }
    glEnd();
    glFlush();
}
```

なぜか，どちらも GL_LINE_LOOP??

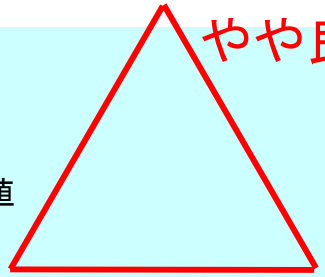




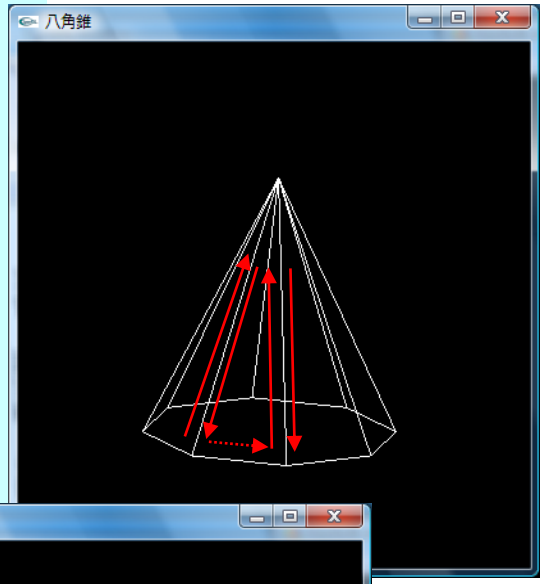
基本課題9 解答例

D君解答

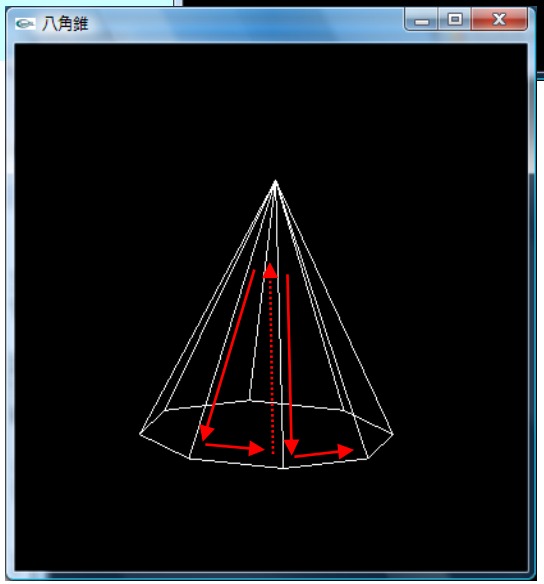
```
void display(void){

    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );
    double angle = 2*3.1415/8; // 45度のラジアン値
    int i;
    glBegin( GL_LINE_LOOP );
    for (i = 0; i < 8; i++)
        {
            glVertex3f(cos(i*angle),-1, sin(i*angle));
            glVertex3f(0.0,1.0,0);
            glVertex3f(cos(i*angle),-1, sin(i*angle));
        }

    glEnd();
    glFlush();

}
```



やや良い




```
for (i = 0; i < 8; i++)
{
    glVertex3f(0, +1, 0);
    glVertex3f(cos(i*angle),-1, sin(i*angle));
    glVertex3f(cos((i-1)*angle),-1, sin((i-1)*angle));
}
```



できるだけ無駄な描画を減らすこと！

基本課題9 解答例

E君解答

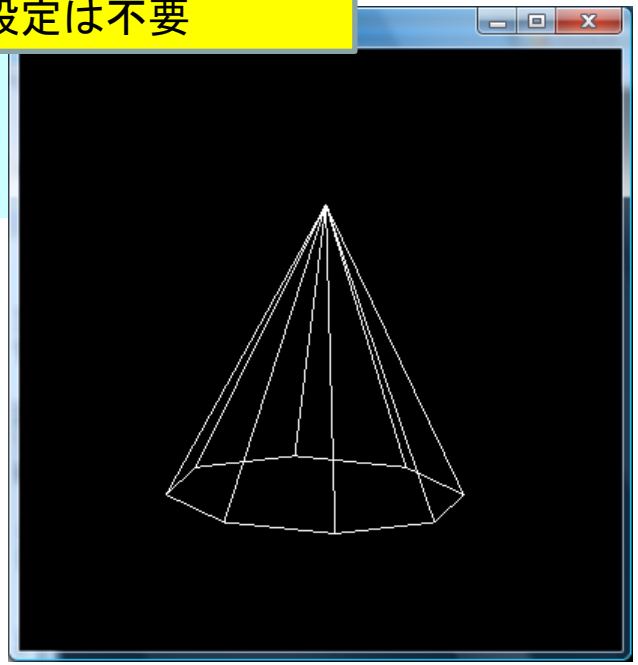
```
glBegin( GL_LINE_LOOP );
for (i = 0; i < 8; i++)
{
    glVertex3f(cos(i*angle), -1.0, sin(i*angle));
    glVertex3f(0, 1, 0);
}
glBegin( GL_LINE_LOOP );
for (i = 0; i < 8; i++)
{
    glVertex3f(cos(i*angle), -1.0, sin(i*angle));
}
glEnd();
glEnd();
glFlush();
```

原則　glBegin()とglEnd()の呼び出しは1対1に対応させること！

GL_LINE_LOOPモードで描画を続けるなら，glBegin()～glEnd()による再設定は不要

対応するglEnd() が呼び出され無いのに，glBegin()が無駄に繰り返し呼び出されている

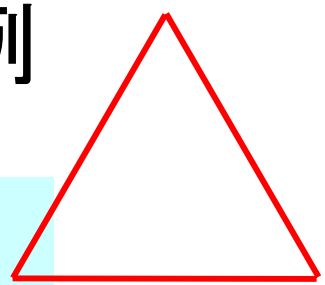
```
glBegin( GL_LINE_LOOP );
    for (i = 0; i < 8; i++)
    {
        glBegin( GL_LINE_LOOP );
        glVertex3f(0, 1, 0);
        glVertex3f(cos(i*angle), -1, sin(i*angle));
        glVertex3f(cos((i+1)*angle), -1, sin((i+1)*angle));
    }
glEnd();
glFlush();
```



解説

glBegin(GL_LINE_LOOP) ～ glEnd()は，その間で指定された頂点をLINE_LOOP(一筆書き)で描画することを意味する．従って，2重ループであろうと複数ループであろうとglBegin(GL_LINE_LOOP) ～ glEnd()は一対で良い

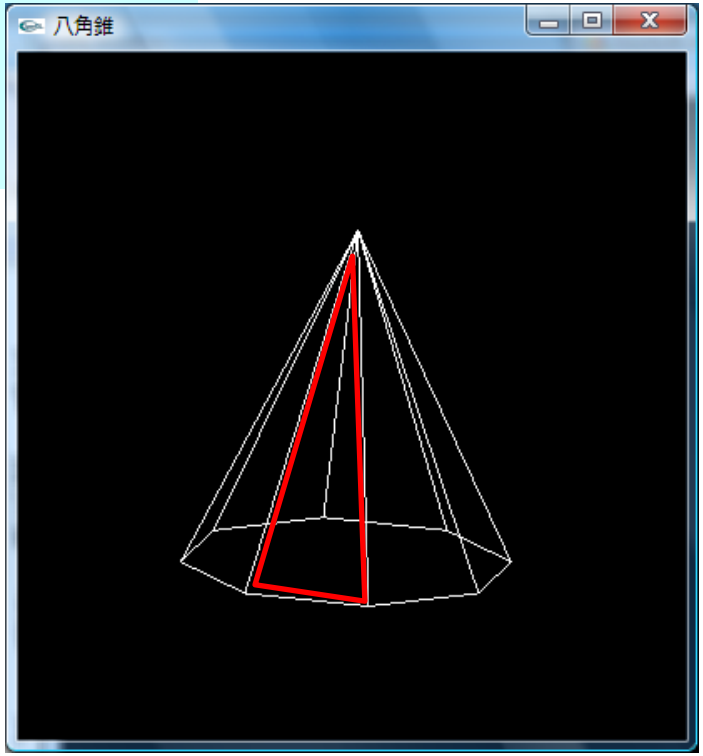
基本課題9 解答例



F君解答

```
double w = -2 * 3.1416 / 8;
for (int i = 0; i < 8; i++)
{
    glBegin(GL_LINE_LOOP);
    glVertex3f(0, 1, 0);
    glVertex3f(sin(i*w), -1, cos(i*w));
    glVertex3f(sin((i + 1)*w), -1, cos((i + 1)*w));
    glEnd();
}
```

3点を指定して，閉じた三角形を描画する



glBegin() ～glEnd()の位置によっては意味が変わる場合がある.

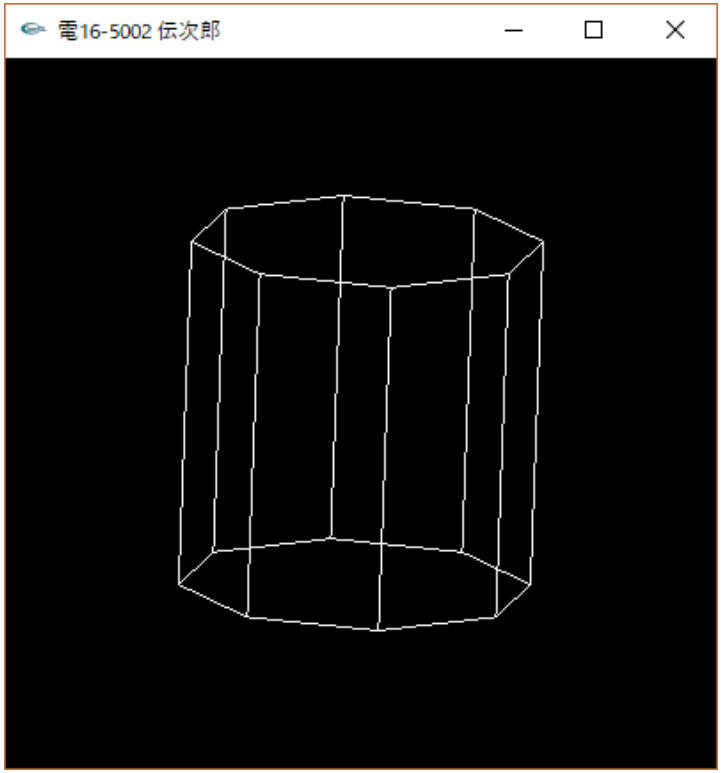
解説

glBegin(GL_LINE_LOOP) ～ glEnd()では，glEnd()を呼び出した時点で閉じた図形とみなされ，最初と最後の頂点の間に線が引かれる.

発展課題9 解答例

A君解答

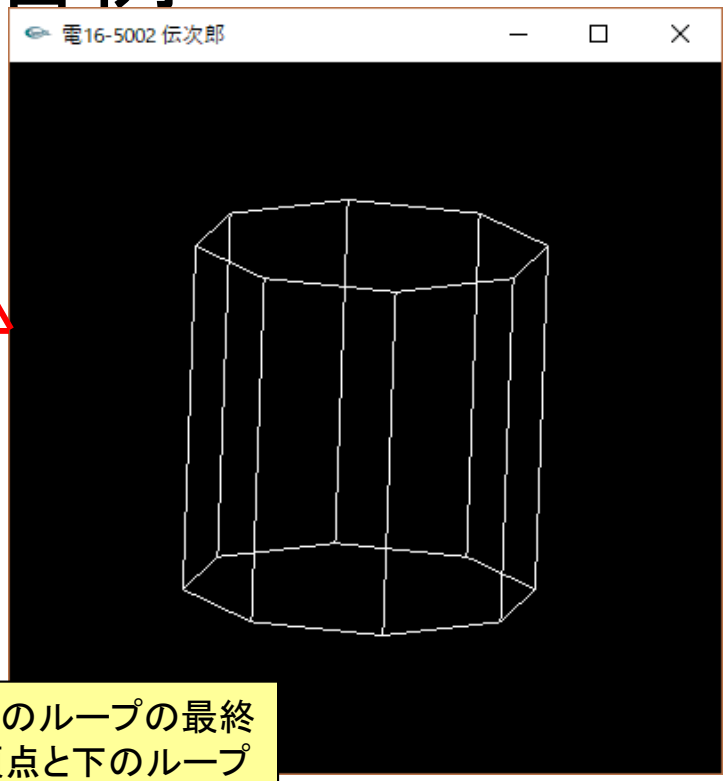
```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    int i, N;    N = 8;
    double angle = 2 * 3.1415 / N;       //45度のラジアン


    glBegin(GL_LINE_LOOP);       //底辺(８角形)の描画
    for (i = 0; i < N; i++) {
        glVertex3f(cos(i*angle), -1.0, sin(i*angle));
    }
    glEnd();

    glBegin(GL_LINE_LOOP);       //底辺(８角形)の描画
    for (i = 0; i < N; i++) {
        glVertex3f(cos(i*angle), +1.0, sin(i*angle));
    }
    glEnd();

    glBegin(GL_LINES);             //頂点へ伸びる縦線
    for (i = 0; i < N; i++) {
        glVertex3f(cos(i*angle), -1.0, sin(i*angle)); //始点
        glVertex3f(cos(i*angle), +1.0, sin(i*angle)); //終点
    }
    glEnd();
    glFlush();
}
```



発展課題9 解答例

B君解答

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);

    /*下面と下面の作成*/
    double angle = 2 * 3.1415 / 8;
    int i;

    glBegin(GL_LINE_LOOP);
        for (i = 0; i <= 8; i++)
        {
            glVertex3f(cos(i*angle), -1.0, sin(i*angle) );
        }

        for (i = 0; i <= 8; i++)
        {
            glVertex3f(cos(i*angle), 1.0, sin(i*angle));
        }

    glEnd();

    /*側面の作成*/
    glBegin(GL_LINES);
    for (i = 1; i < 8; i++)
    {
        glVertex3f(cos(i*angle), -1.0, sin(i*angle));
        glVertex3f(cos(i*angle), 1.0, sin(i*angle));
    }

    glEnd();
    glFlush();;
}
```



下のループの最終頂点と上のループの開始頂点で線が引かれる

上のループの最終頂点と下のループの開始頂点で線が引かれる

display()関数の疑問

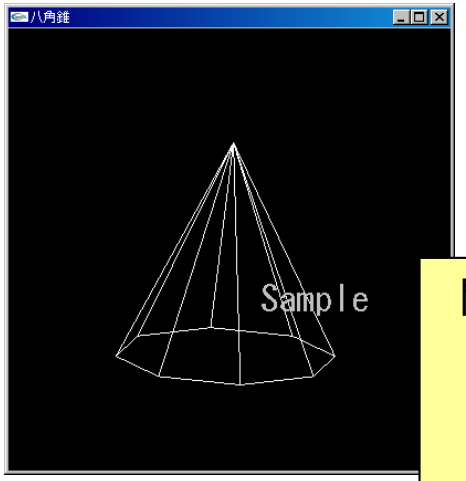
```
#include "glut.h"
#include <GL/gl.h>
#include <math.h>

void display(void)
{
    // 八角錐の頂点座標を設定
    // ・・・
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("八角錐");
    glClearColor (0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);

    glMatrixMode(GL_MODELVIEW);
    glRotatef(8, 0, 1, 0);
    glRotatef(15, 1, 0, 0);

    glutDisplayFunc(display);
    glutMainLoop();
}
```

関数は呼び出されて実行される

↓

display()関数が呼び出されて画面に図形が表示される

↓



関数呼び出しなら
display();
となるはず.

これはdisplay()関数の呼び出しか？

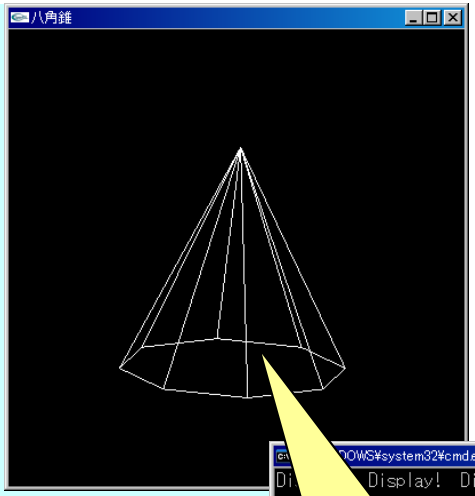
答え:NO！

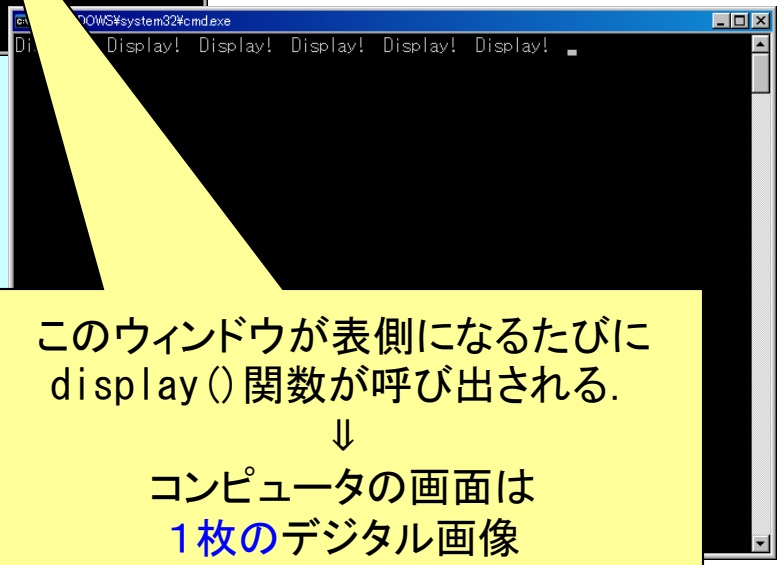
↓

では，これは何をしているのか？

display()関数はいつどこから呼び出されるのか？

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

void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );

    int N = 8;
    double angle = 2*3.1415/N;
    int i;
    glBegin( GL_LINE_LOOP );
        for (i = 0; i < N; i++)
            glVertex3f(cos(i*angle), -1.0, sin(i*angle));
    glEnd();
    glBegin( GL_LINES );
        for (i = 0; i < N; i++)
        {
            glVertex3f(0.0, 1.0, 0.0);
            glVertex3f(cos(i*angle), -1.0, sin(i*angle));
        }
    glEnd();
    glFlush();

    printf("Display!  ");
}
```





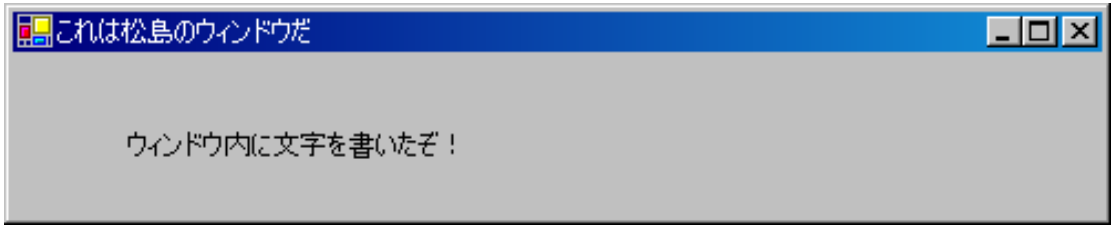
呼び出されるたびに
Display!
をコンソールに表示

このウィンドウが表側になるたびに
display()関数が呼び出される.
⇓
コンピュータの画面は
1枚のデジタル画像
⇓
display()関数は必要に応じてこの
ウィンドウ内の画像を描画している！
イベントドリブン方式

GUIプログラミングの基本＝イベント

イベントとは

ウィンドウやウィンドウ内のパーツ(ボタン・スクロールバーなど)に発生する様々な**出来事**・きっかけ



イベントの例

- マウスがウィンドウ上に入った
- マウスがウィンドウ上から出た
- マウスのボタンが押された
- キーボードのキーが押された
- ウィンドウが閉じられた
- ウィンドウが開かれた
- ウィンドウのサイズが変わった
- 他のウィンドウの下になっていたウィンドウ全体が前に出た
- 他のウィンドウに隠れていた一部分が前に出た

このタイプのイベントが発生したらプログラムはウィンドウ内の文字や図形を描かなければならない → Paint イベント発生

Paintハンドラがイベント処理！

ウィンドウが隠れて出たり，サイズが変わったりするたびにウィンドウ内を描画するのか？
答え： YES！　それがウィンドウプログラミングの基本

GUIプログラムの実行方式＝イベント駆動(イベントドリブン)

非GUIプログラム

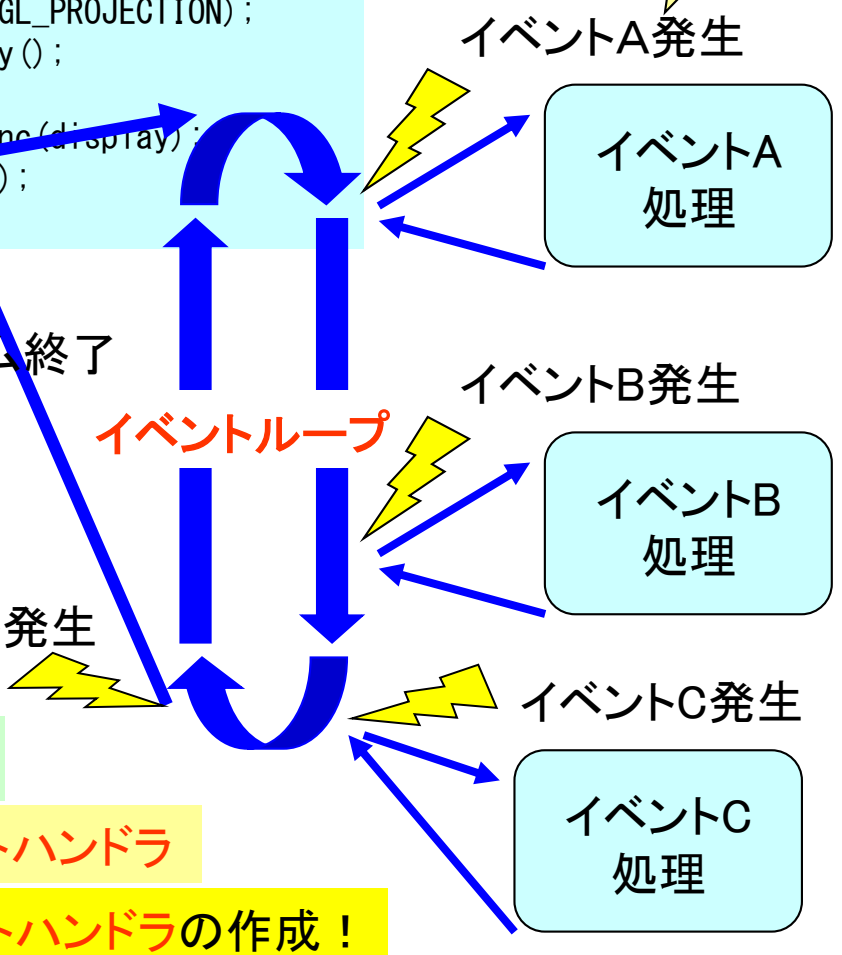
```
int main(void)
{
    int x,i ;
    x = x + 10;
    printf(・・・);
    printf(・・・);
    for(i = 0; i < 10; i++)
    {
        ・・・・
        ・・・・
    }
    if (x > 20)
        ・・・・
    ・・・・
    ・・・・


}
```

予測可能な一定の順序で処理が進む

GUIプログラム

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    ・・・
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    ・・・
    glutDisplayFunc(display);
    glutMainLoop();
}
```

イベント発生のタイミングも順序も予測不能



イベントを処理する関数 → イベントハンドラ

display()関数は何か？　→正解ペイントイベントハンドラ

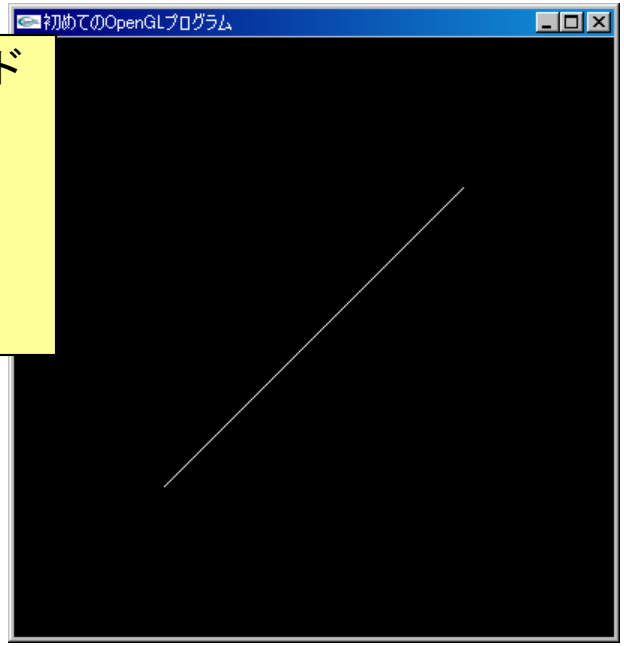
GUIではプログラマの仕事のほとんどはイベントハンドラの作成！

ペイントハンドラの登録

Example9-1

```
#include "glut.h"
#include <GL/gl.h>

void display(void)
{
    glClear( GL_COLOR_BUFFER_BIT );

    glColor3f(1.0, 1.0, 1.0);
    glBegin( GL_LINES );
        glVertex3f(-1.0, -1.0, 0.0);
        glVertex3f(+1.0, +1.0, 0.0);
    glEnd();

    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("初めてのOpenGLプログラム");
    glClearColor (0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);

    glutDisplayFunc(display);
    glutMainLoop();
}
```

GLUTでのペイントハンドラ関数の形式
void func(void)
⇓
この形式の関数なら
どんな名前でも良い



ペイントハンドラとしてdisplay
という名前の関数を登録する.
関数ポインタ

キーボードハンドラの登録

Example10-1

```
void KeyboardHandler(unsigned char key, int x, int y)
{
    // 押されたキーがスペース(空白)だったらプログラムを終了する
    if (key == ' ')
        exit(0);    //プログラムを終了する.
}

int main( int argc, char** argv )
{
    glutInit( &argc, argv );

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("八角錐");
    glClearColor (0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);

    glMatrixMode(GL_MODELVIEW);
    glRotatef(8, 0, 1, 0);
    glRotatef(15, 1, 0, 0);

    glutDisplayFunc(display);
    glutKeyboardFunc(KeyboardHandler);
    glutMainLoop();
}
```

GLUTでのキーボードイベントハンドラ関数の形式

```
void func(unsigned char key, int x, int y)
```

key　押されたキーの文字コード

x,y　キーボードが押されたときのカーソル位置

```
void exit(int status)
```

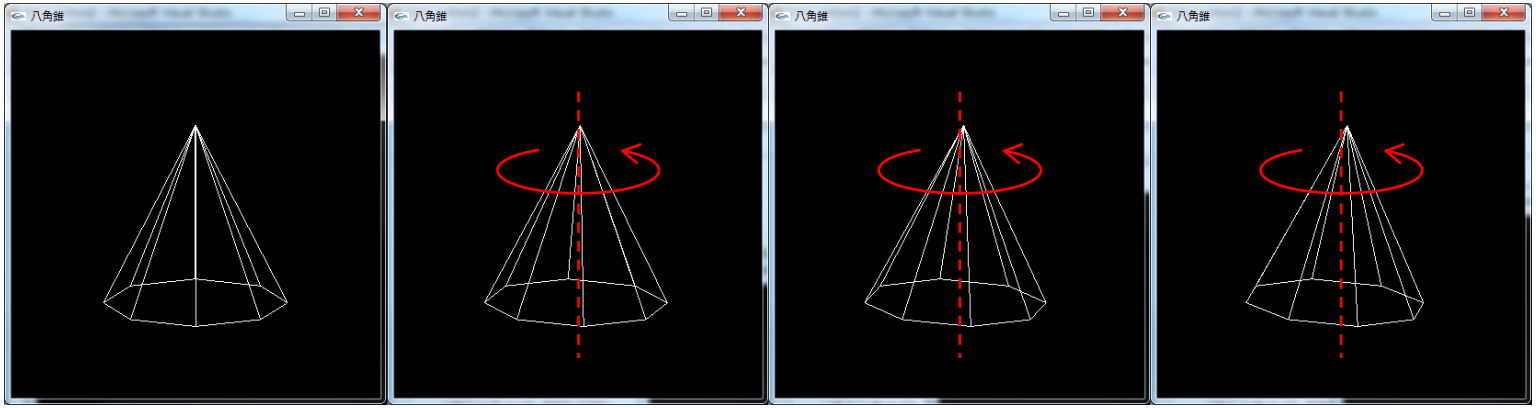
ヘッダファイル stdio.h

引数 status：状態

プログラムを終了する. OSにstatusを情報として伝える.

キーボードイベントハンドラとしてKeyboardHandler()関数を登録する

どうやってアニメーションを行うか？



y軸の周りで図形を少しずつ回転する

↓

glRotatef(t, 0, 1, 0)を用いて図形を少しずつ回転する

↓

変数tの値を少しずつ変更する

↓

どんなタイミングで変数tの値を変更するのか?
ペイントハンドラ内でループを回して変数tの値を増やす?
ペイントハンドラが呼び出されるたびに変数tの値を増やす?

↓

正解！　アイドルハンドラの利用

アイドルハンドラとアニメーション

Example10-2

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

void KeyboardHandler(unsigned char key, int x, int y);

double RotAngle = 0.0;      //グローバル(大域)変数

void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(RotAngle, 0, 1, 0);
    glRotatef(15, 1, 0, 0);

    int N = 8;
    double angle = 2*3.1415/N;
    int i;
    glBegin( GL_LINE_LOOP );
        for (i = 0; i < N; i++)
            glVertex3f(cos(i*angle), -1.0, sin(i*angle));
    glEnd();
    glBegin( GL_LINES );
        for (i = 0; i < N; i++)
        {
            glVertex3f(0.0, 1.0, 0.0);
            glVertex3f(cos(i*angle), -1.0, sin(i*angle));
        }
    glEnd();
    glFlush();
}
```

回転角度

変換行列を指定
- *x*軸周りで15度回転
- *y*軸周りでRotAngle度回転

アイドルハンドラ
- 回転角度を0.1度ずつ増加
- ウィンドウの再描画

```
void IncAngle(void)
{
    RotAngle = RotAngle + 0.1;
    if (RotAngle > 360.0)
        RotAngle = RotAngle - 360.0;
    glutPostRedisplay();
}

int main( int argc, char** argv )
{
    glutInit( &argc, argv );

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("八角錐");
    glClearColor (0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);

    glutDisplayFunc(display);
    glutKeyboardFunc(KeyboardHandler);
    glutIdleFunc(IncAngle);
    glutMainLoop();
}
```

回転速度を決める

ウィンドウの再描画

ペイントイベントの発生
↓
ペイントハンドラの呼び出し

Idleイベントは他にイベントが無い時に定期的に発生するイベント

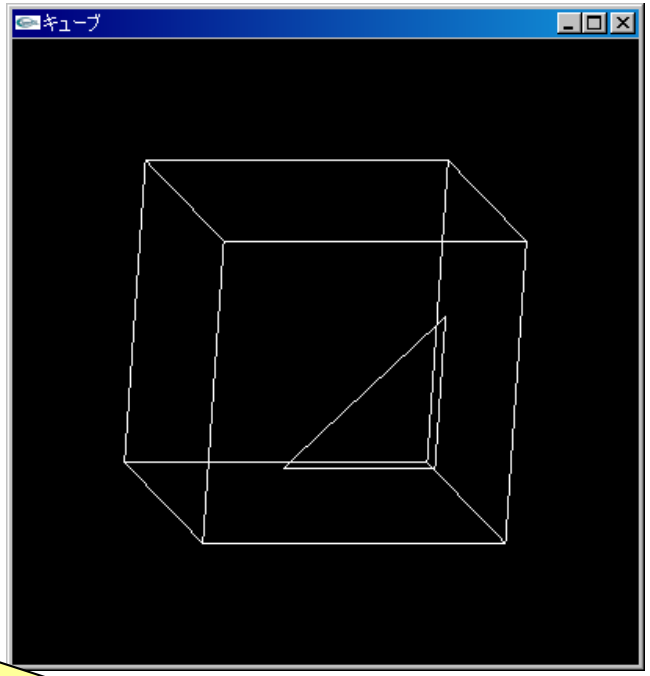
アイドルハンドラの登録
単語 idle = 遊んでいる. 暇な.

投影方法の設定

投影 = 3次元のモデルデータを2次元の図形にする処理

Example9-4

```
// display()関数は前スライドと同じ

int main(int argc, char** argv)
{
    glutInit(&argc, argv);

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("キューブ");
    glClearColor (0.0, 0.0, 0.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);

    glMatrixMode(GL_MODELVIEW);
    glRotatef(15, 0, 1, 0);  //y軸周りに15度回転
    glRotatef(15, 1, 0, 0);  //x軸周りに15度回転

    glutDisplayFunc(display);
    glutMainLoop();
}
```

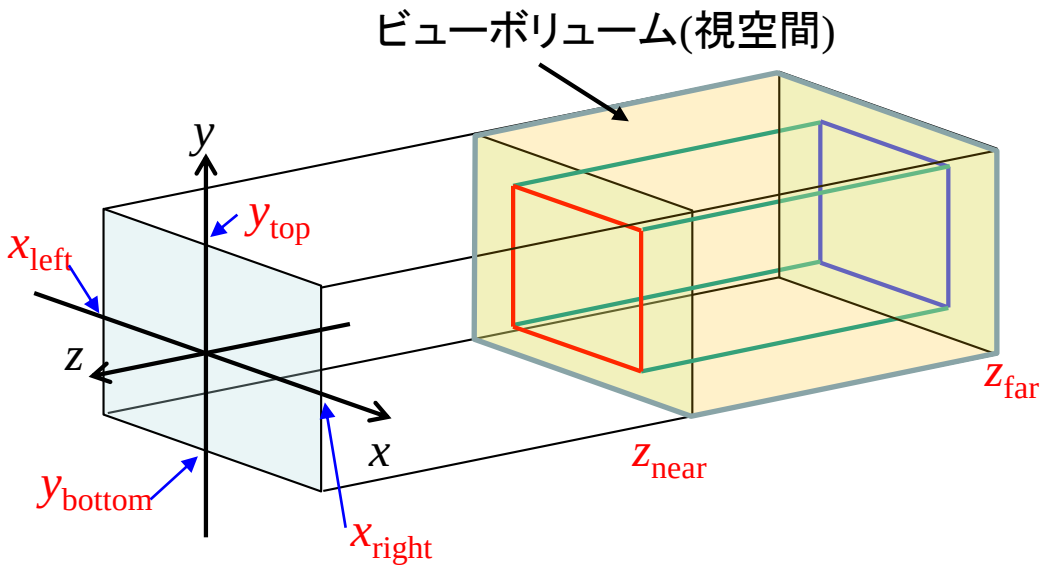


投影方法として正投影(平行投影)を指定し，ビューボリュームを設定している．

正投影(平行投影)とビューボリュームの設定

平行投影を投影変換行列として設定する

```
glOrtho(xleft, xright, ybottom, ytop, znear, zfar)
```



平行投影
の投影変換行列

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

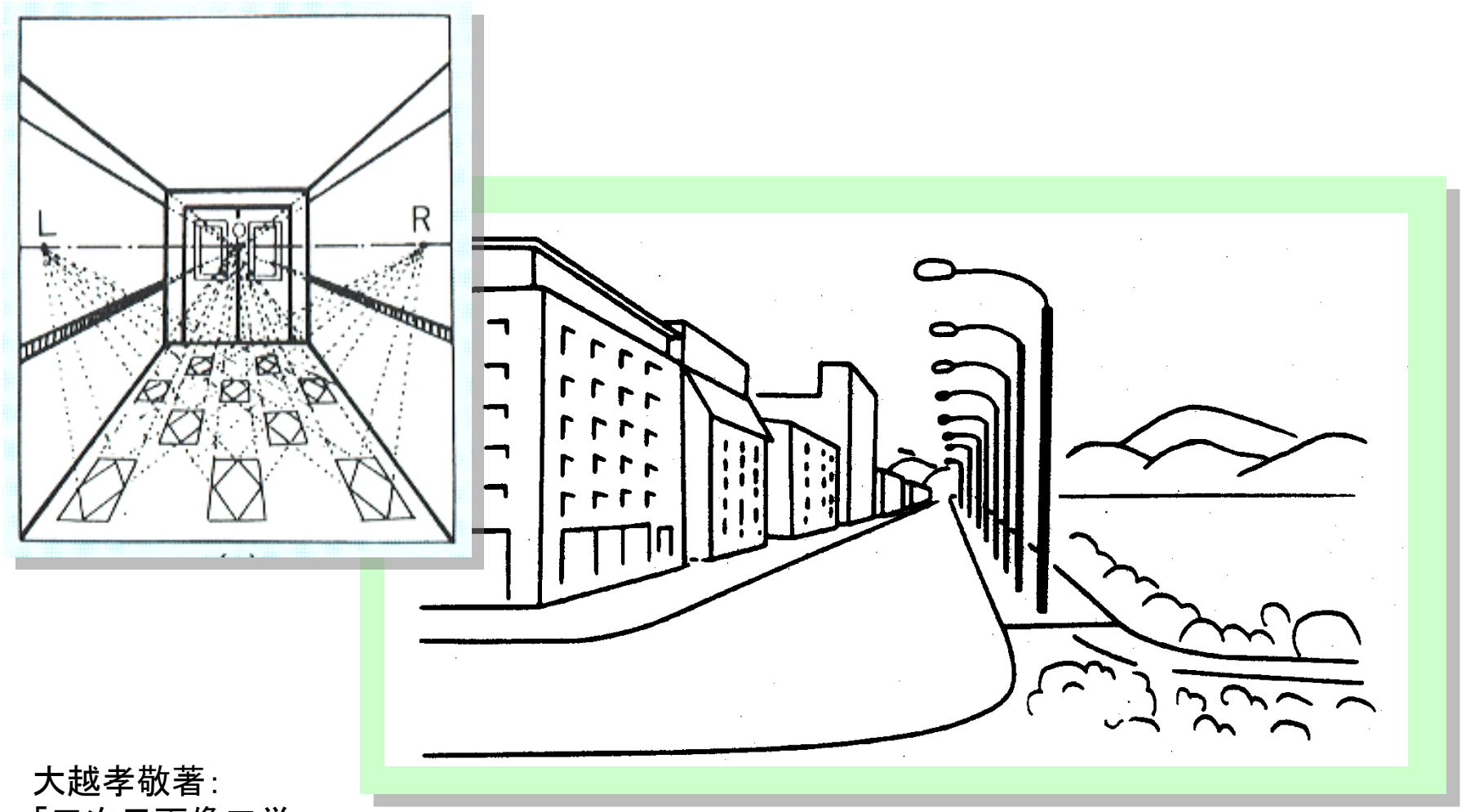
注意
ビューボリュームに
入っていない部分は
クリッピング処理により
表示されない

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);
```

投影変換行列の設定開始

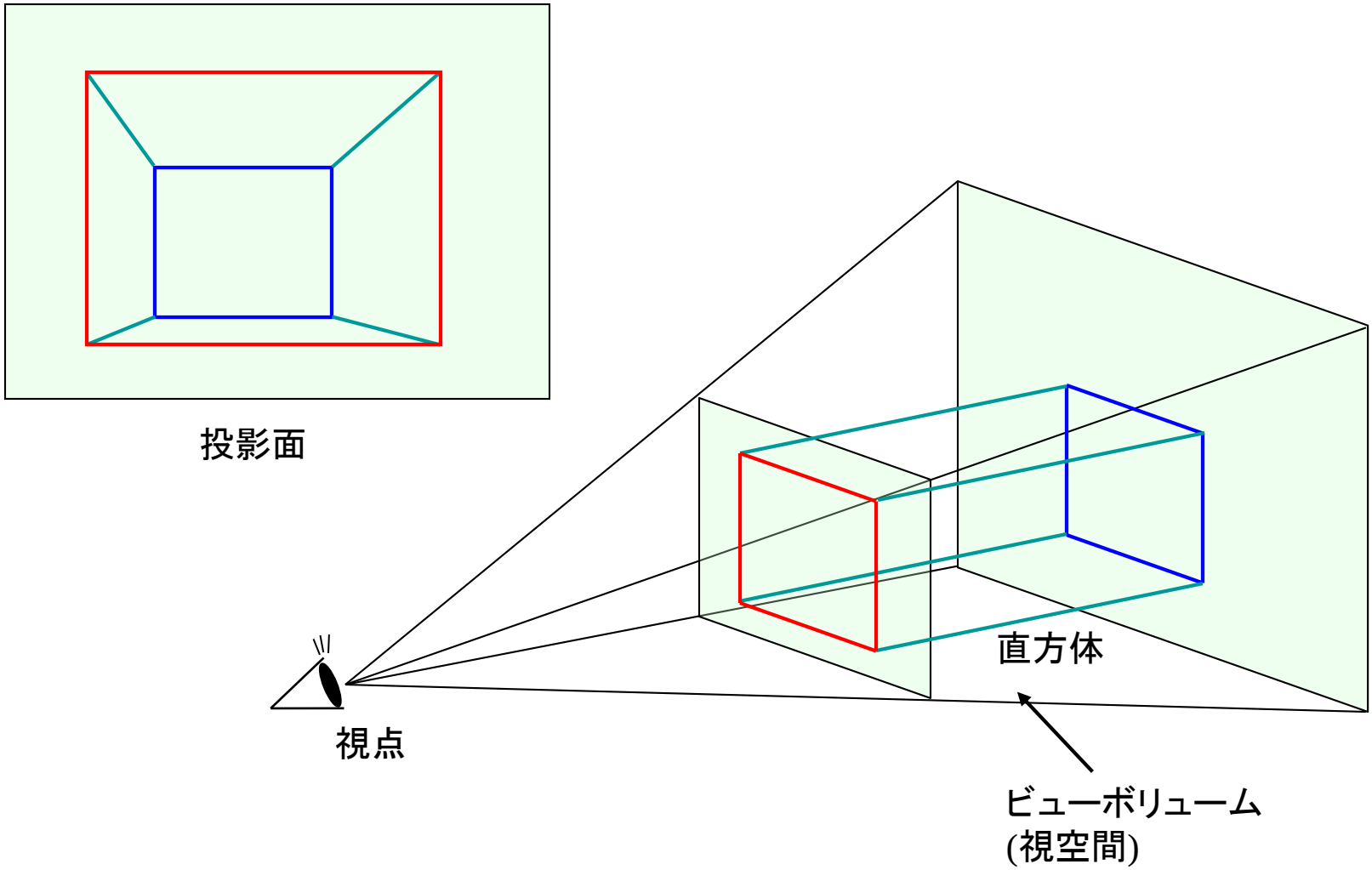
平行投影の行列を設定

透視投影(透視図法)



大越孝敬著:
「三次元画像工学」

透視投影とそのビューボリューム



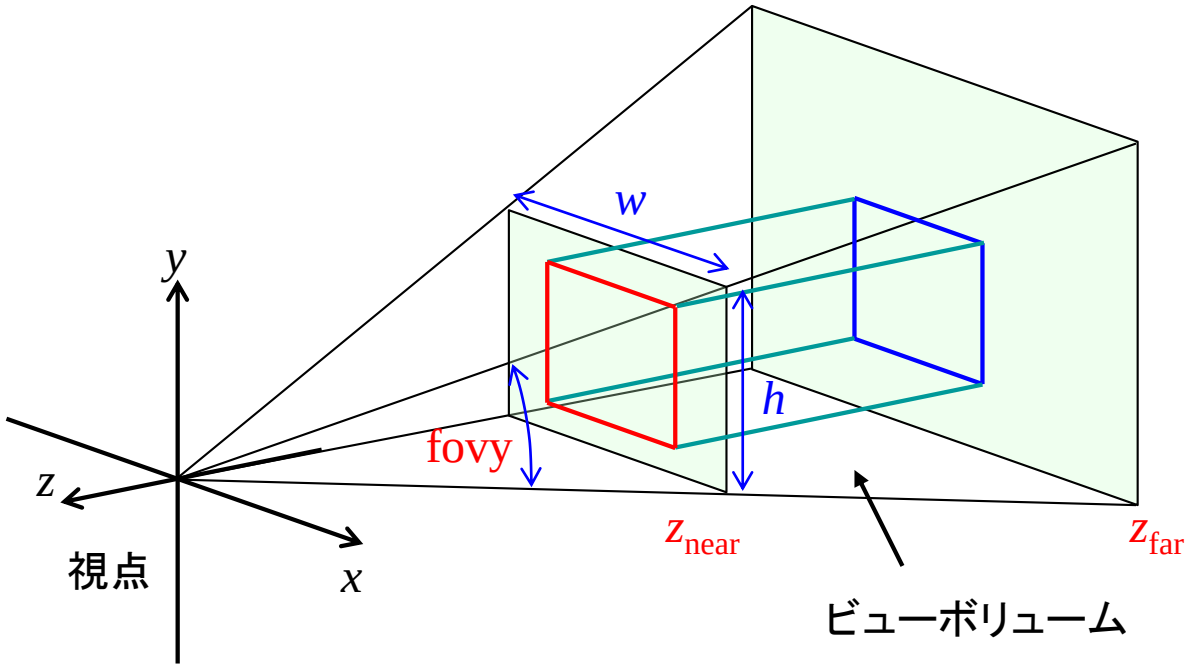
透視投影とそのビューボリュームの設定

透視投影を投影変換行列として設定する

```
gluPerspective(fovy, aspect, znear, zfar)
```

アスペクト比
aspect = w / h

注: znearとzfarは正値で設定する
(znearとzfarは原点からの距離を表す)



透視投影
の投影変換行列

$$\begin{bmatrix} \frac{1}{\tan\theta} & 0 & 0 & \cdots \\ 0 & \ddots & \ddots & \\ \vdots & & & \\ \vdots & & & \end{bmatrix}$$

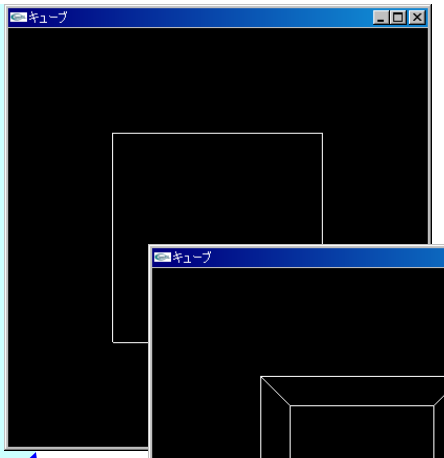
複雑なので省略

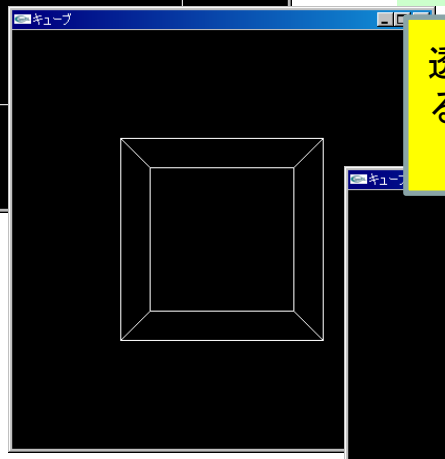
注意
ビューボリュームに入っていない部分はクリッピング処理により表示されない.

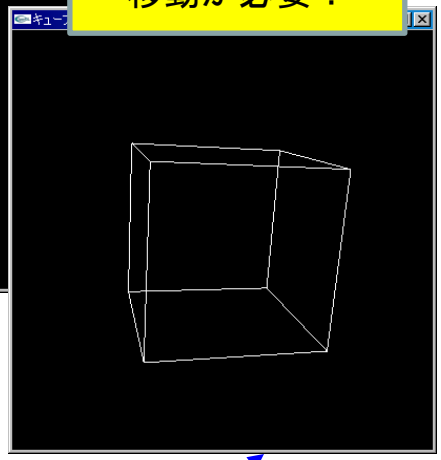
透視投影のプログラム

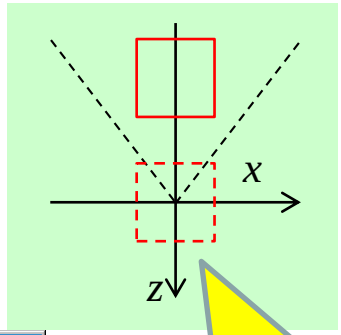
Example10-3

```
#include "glut.h"
#include <GL/gl.h>

void KeyboardHandler(unsigned char key, int x, int y);

void display( void )
{
    // 辺長２の立方体を描く
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("キューブ");
    glClearColor ( 0.0, 0.0, 0.0, 1.0 );

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho( -2.0, 2.0, -2.0, 2.0, -2.0, 2.0 );

    glutDisplayFunc(display);
    glutKeyboardFunc(KeyboardHandler);
    glutMainLoop();
}
```









透視投影で表示するためには物体の移動が必要！

平行投影

透視投影と移動

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45, 1.0, 0.0, 10.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0, 0.0, -6.0);
```

透視投影と移動と回転

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45, 1.0, 0.0, 10.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0, 0.0, -6.0);
glRotatef(15, 0, 1, 0);
glRotatef(15, 1, 0, 0);
```

投影方法(投影変換行列)の設定

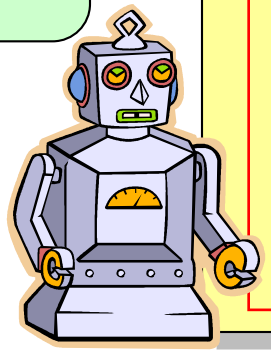
アフィン変換(幾何変換行列)の設定

状態マシンとしてのOpenGL

OpenGLは，現在の自分の状態に従って，頂点データを処理(描画)する

頂点データ
$v_1, v_2, v_3, \cdots$

OpenGL
状態
- 描く線の幅
- 描く線の色
- クリアで塗りつぶす色
- 幾何変換行列(GL_MODELVIEW) **M**
- 投影変換行列(GL_PROJECTION) **P**
- ・
- ・
- ・



$v'_1, v'_2, v'_3, \cdots$ で形成される図形の描画

拡大･回転など頂点座標(物体モデル)の操作

3次元物体モデルをどのように2次元図形に変換するかを決定

状態変更コマンド(関数)

OpenGLの現在の状態を変更する

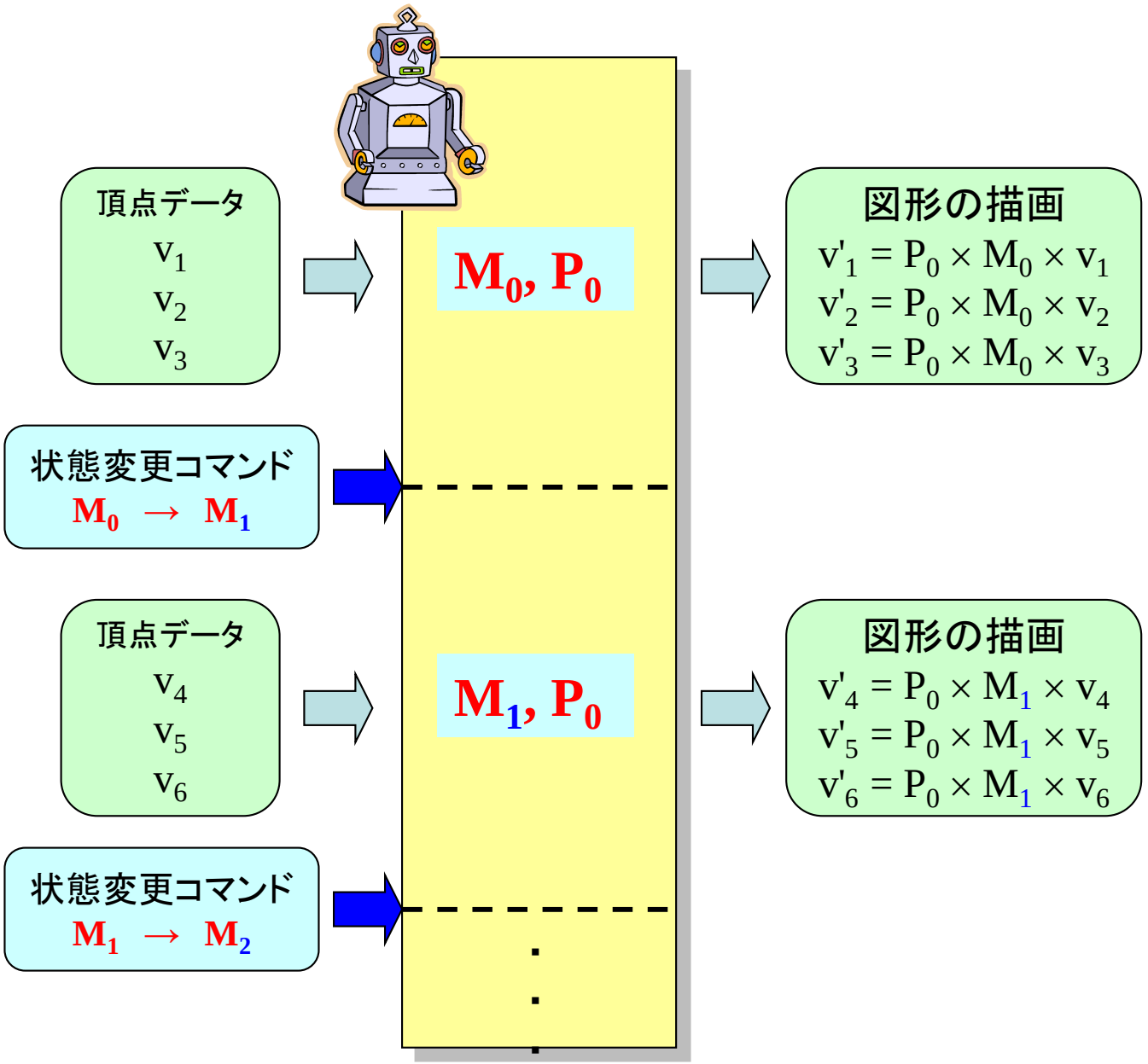
幾何変換行列Mと投影変換行列Pが頂点をアフィン変換してから図形を描画する

$$v' = P \times M \times v$$

$$M = M_1 M_2 M_3 \cdots$$

投影変換行列や幾何変換行列の設定関数は状態変更コマンド

OpenGL状態マシンの動作



幾何変換行列の設定(状態変更)

```
void glMatrixMode(GL_MODELVIEW)
```

幾何変換行列M(状態)の変更開始の指示

```
void glLoadIdentity(void)
```

Mとして単位行列 **I** を新たに設定する

単位行列に何を乗算しても同じ(行列をリセットする役割)

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
void glTranslatef(tx, ty, tz)
```

Mに, (tx, ty, tz)移動する変換行列 **T** を乗算する

```
void glSclalef(sx, sy, sz)
```

Mに, (sx, sy, sz)スケーリングする変換行列 **S** を乗算する

```
void glRotatef(angle, vx, vy, vz)
```

Mに, 回転軸ベクトル(vx, vy, vz)を中心に角度angle[度]回転する回転行列 **R** を乗算する

順序が問題！

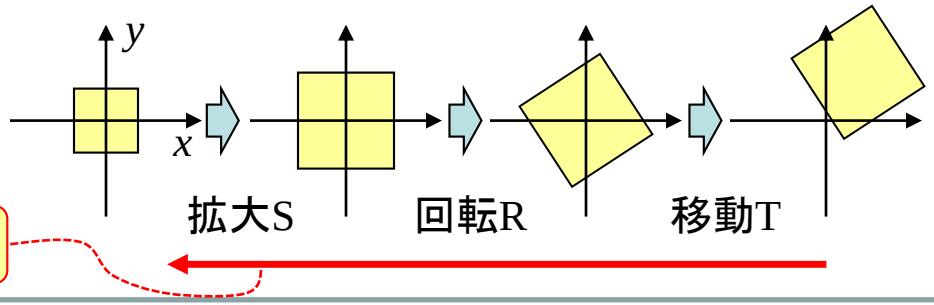
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();              // M=I
glTranslatef(2, 1, 0);         // M=IT
glRotatef(15, 0, 0, 1);        // M=ITR
glScalef(2, 2, 2);             // M=ITRS
```

M = ITRS

同じ

逆

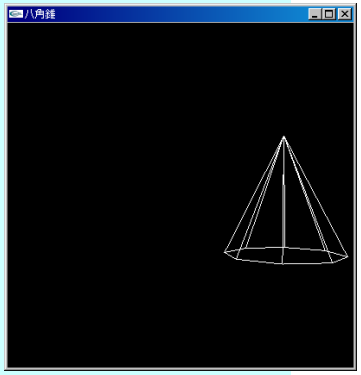
関数の実行順序と幾何変換操作の順序は逆になる

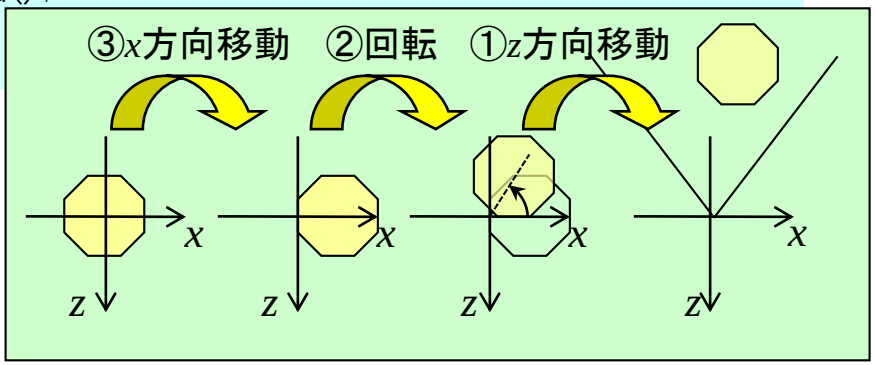


幾何変換と透視投影を用いたプログラム例

Example10-4

```
#include "glut.h"
#include <GL/gl.h>
#include <math.h>

void KeyboardHandler(unsigned char key, int x, int y)
{
    if (key == ' ')
        exit(0);
}

void OctPyramid(void)
{
    int N = 8;
    double angle = 2*3.1415/N;
    int i;
    glBegin( GL_LINE_LOOP );
        for (i = 0; i < N; i++)
            glVertex3f(cos(i*angle), -1.0, sin(i*angle));
    glEnd();
    glBegin( GL_LINES );
        for (i = 0; i < N; i++)
        {
            glVertex3f(0.0, 1.0, 0.0);
            glVertex3f(cos(i*angle), -1.0, sin(i*angle));
        }
    glEnd();
}
```





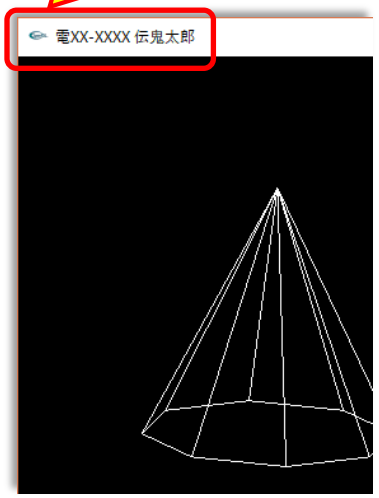

```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -6.0);  //①
    glRotatef(30, 0.0, 1.0, 0.0);  //②
    glTranslatef(+2.0, 0.0, 0.0);  //③

    OctPyramid();

    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);

    glutInitWindowPosition(0, 0);
    glutInitWindowSize(400, 400);
    glutInitDisplayMode(GLUT_RGBA);
    glutCreateWindow("八角錐");
    glClearColor ( 0.0, 0.0, 0.0, 1.0 );

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45, 1.0, 0.0, 10.0);

    glutDisplayFunc(display);
    glutKeyboardFunc(KeyboardHandler);
    glutMainLoop();
}
```

OpenGLの状態として幾何変換行列を登録

現在の状態で描画する頂点を登録

課題10

ウィンドウの名前を各自の学籍番号と氏名にすること



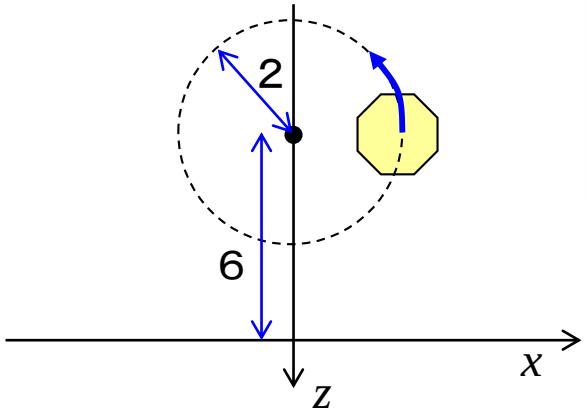
基本課題10

底面の直径が2で，高さが2の八角錐があり，その底面は$(x, -1, z)$平面にある．この八角錐の底面の中心が，$(0, -1, -6)$を中心とする$(x, -1, z)$平面上の半径2の円周上を回転するアニメーションを透視変換で作成せよ．但し，透視変換は上下開き角(fovy) が45度で，アスペクト比を1とする．

以下の二つのファイルをZIPファイルに入れて提出すること．

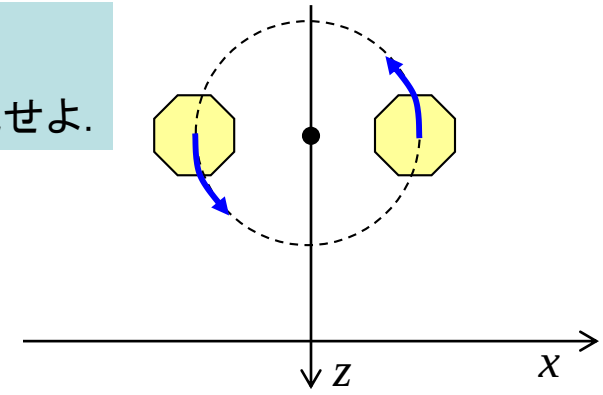
① Wordのレポート
- ソースプログラム
- glutウィンドウの画面コピー

② **実行プログラム(〇〇〇.exe)**



発展課題10

上の課題で八角錐の数を2個にしたアニメーションを作成せよ．

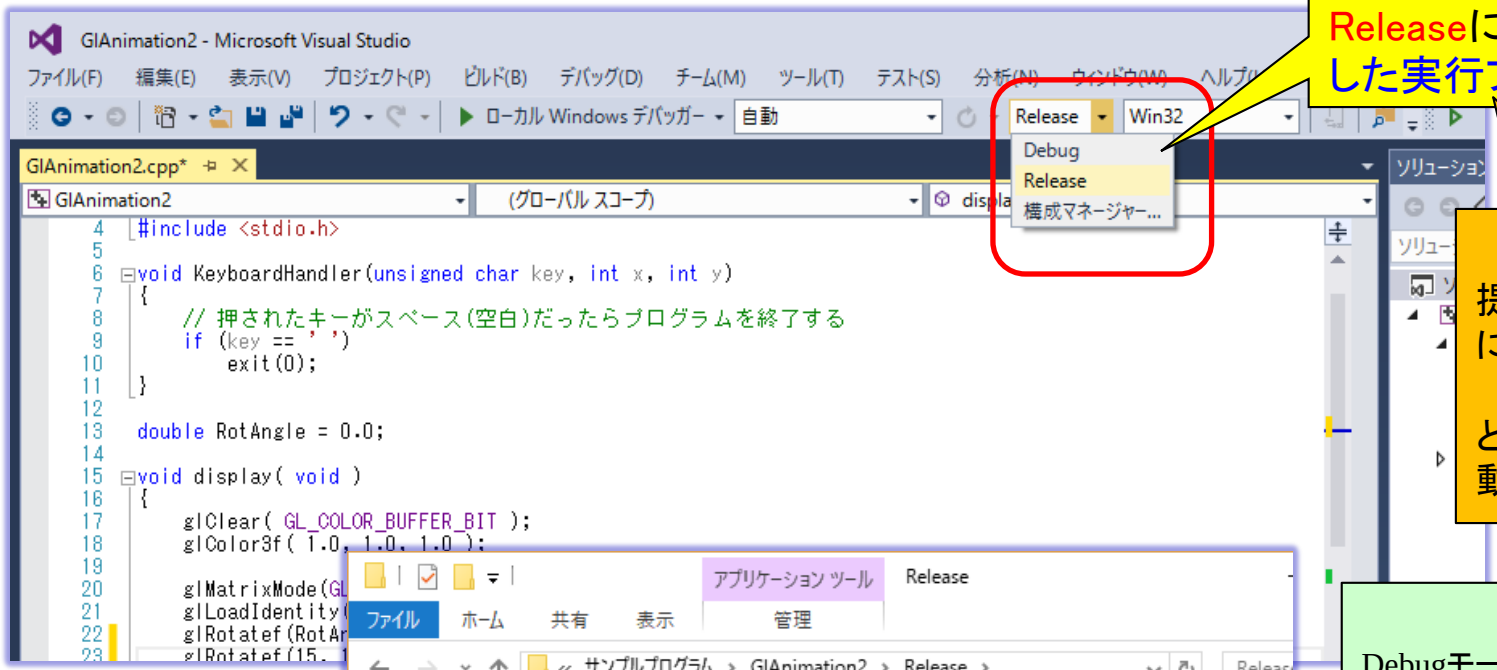


実行例

実行プログラム(Kadai10-1.exe, Kadai10-2.exe)をダウンロードして実行してみよ

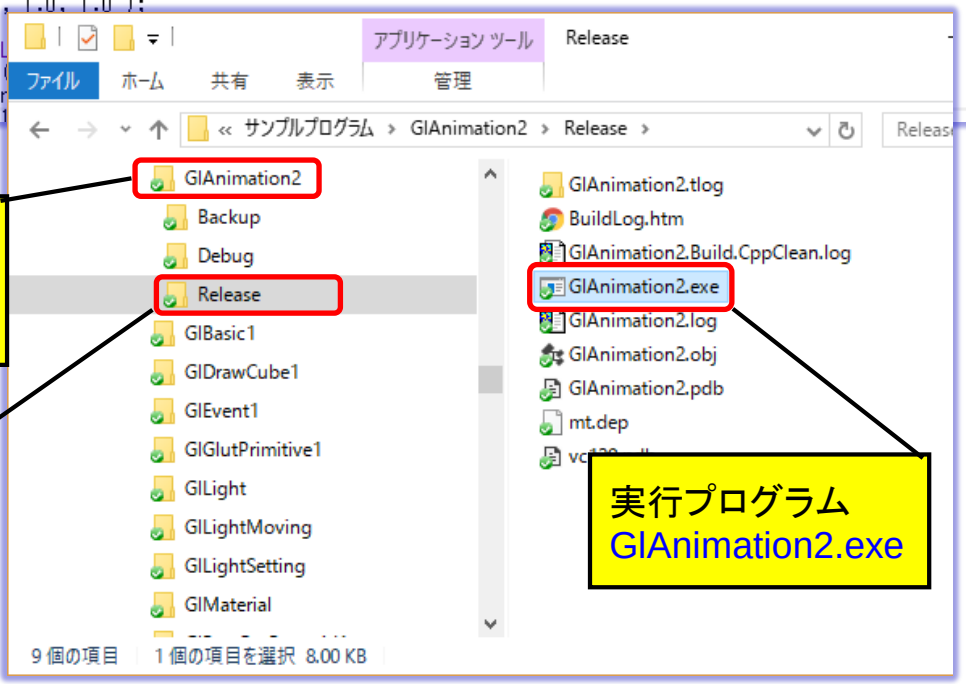
Wordに実行ファイルを添付することは禁止です!
(ウィルスとみなされることがあります) ⇒ 必ずzipファイルを利用

提出用の実行プログラム



[ソリューション構成]を Releaseに変更して，再ビルドした実行プログラムを提出．

【注意】
提出時以外はDebugにしておくこと．
ここがDebugでないと，デバッガが正常に動作しない．



プロジェクトフォルダ
プロジェクト名：GlAnimation2

Releaseフォルダ

実行プログラム GlAnimation2.exe

解説

Debugモードでは，ステップ実行などのために本来不要な部分が実行プログラム(exeファイル)に付け加えられてファイルサイズが増大し実行速度も低下する．

Releaseモードではそのような不要部分が取り除かれ，さらに最適化処理によって高速化し，ファイルサイズも小さくなる．そのため，他人に実行プログラムを渡す場合はReleaseにする方が良い．

ただし，Releaseモードではデバッガは正常に動作しない．