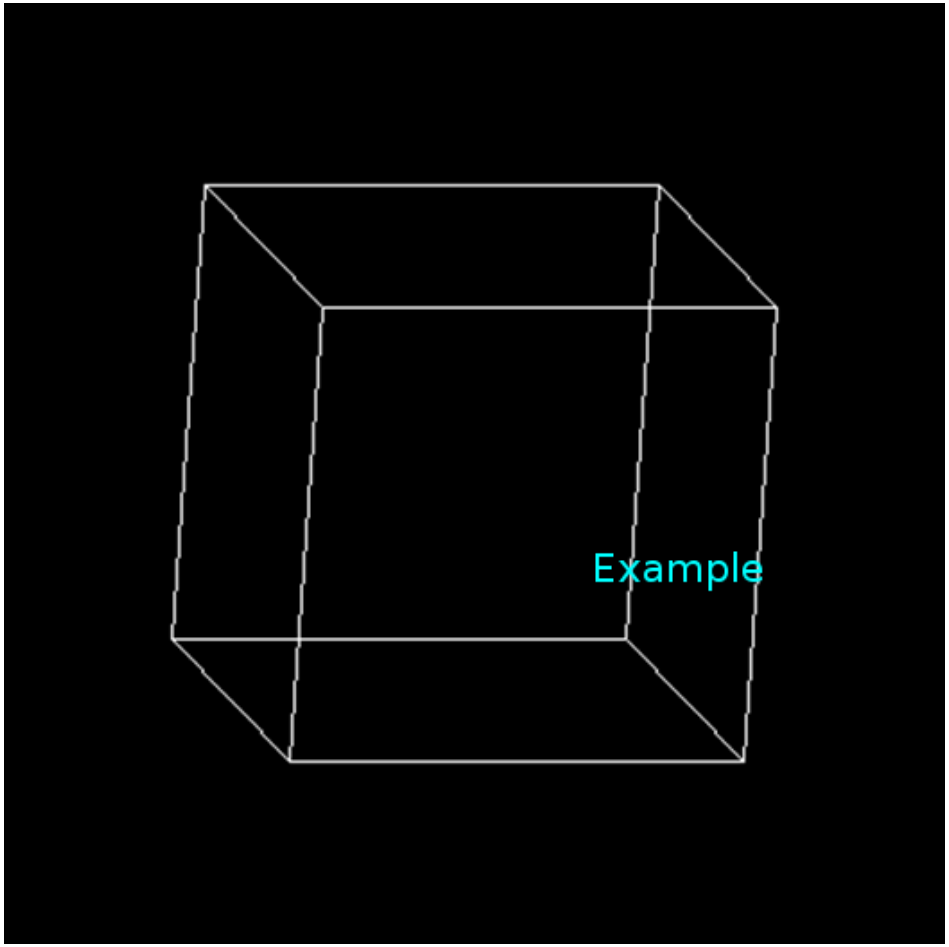
今日の目標

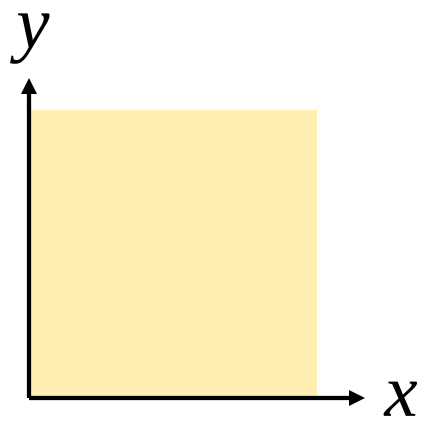
復習



3次元座標系

復習

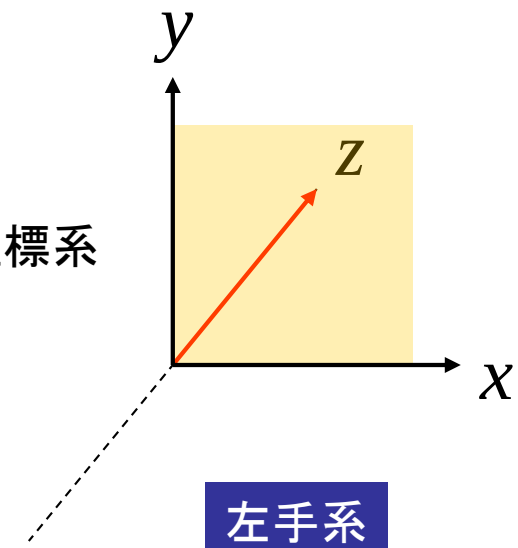
2次元座標系



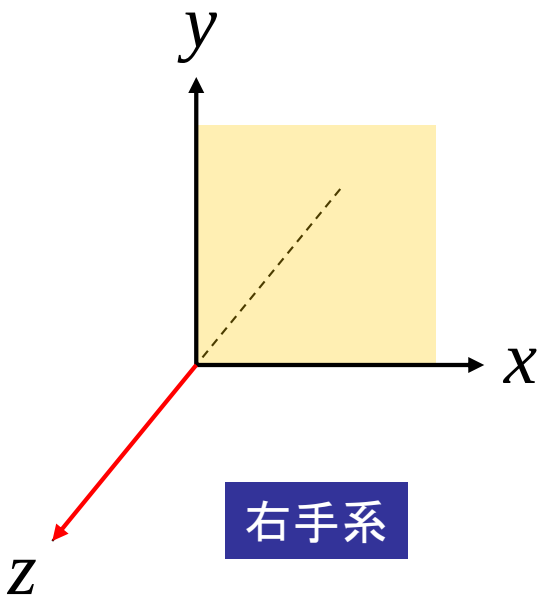
x軸：親指
y軸：人差し指
z軸：中指

右手でこの方向が指差せたら右手系！
左手でこの方向が指差せたら左手系！

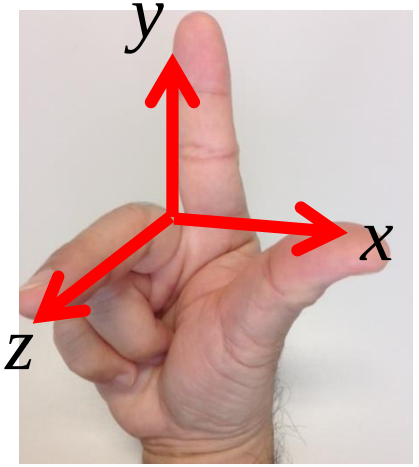
3次元座標系



左手系



右手系



3次元アフィン変換

復習

$$\begin{aligned} x' &= ax + by + cz + s \\ y' &= dx + ey + fz + t \\ z' &= gx + hy + iz + u \end{aligned}$$

$$\Leftarrow$$

$$\begin{aligned} x' &= ax + by + c \\ y' &= dx + ey + f \end{aligned}$$

$$\Downarrow$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} s \\ t \\ u \end{bmatrix} \Rightarrow \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & s \\ d & e & f & t \\ g & h & i & u \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

同次座標系

$\mathbf{v}' = \mathbf{M}\mathbf{v}$

$$\mathbf{v}' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}, \quad \mathbf{M} = \begin{bmatrix} a & b & c & s \\ d & e & f & t \\ g & h & i & u \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

変換前の座標

変換後の座標

3次元幾何変換

復習

拡大/縮小(スケーリング)

$$\mathbf{S}=\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} x' &= S_x x = S_x x + 0 \times y + 0 \times z + 0 \\ y' &= S_y y = 0 \times x + S_y y + 0 \times z + 0 \\ z' &= S_z z = 0 \times x + 0 \times y + S_z z + 0 \end{aligned}$$

移動

$$\mathbf{T}=\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} x' &= x' + t_x = 1 \times x + 0 \times y + 0 \times z + t_x \\ y' &= y' + t_y = 0 \times x + 1 \times y + 0 \times z + t_y \\ z' &= z' + t_z = 0 \times x + 0 \times y + 1 \times z + t_z \end{aligned}$$

3次元の回転(典型的な例)

復習

x軸周りの回転(反時計回り)

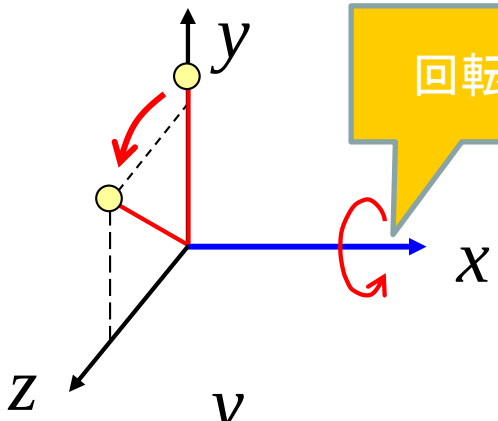
$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

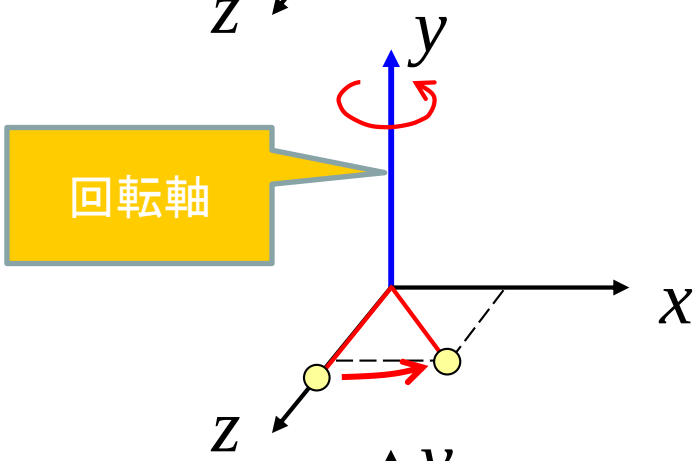
y軸周りの回転(反時計回り)

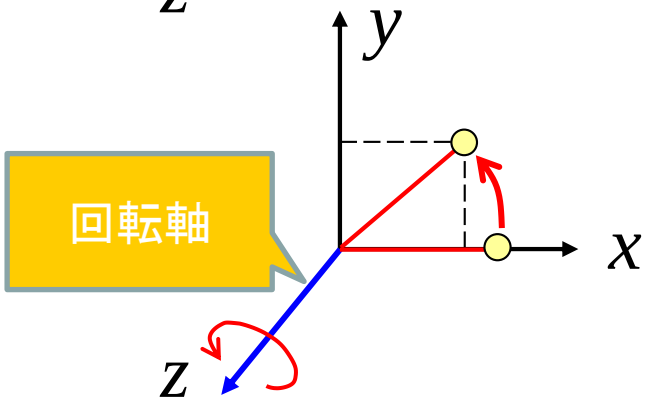
$$\mathbf{R}_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

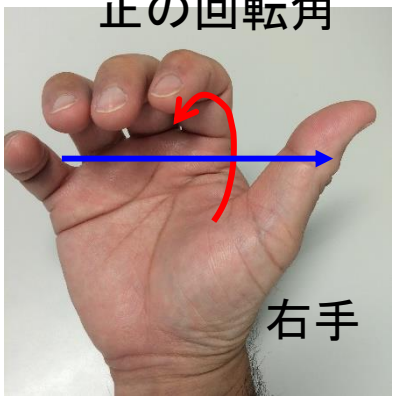
z軸周りの回転(反時計回り)

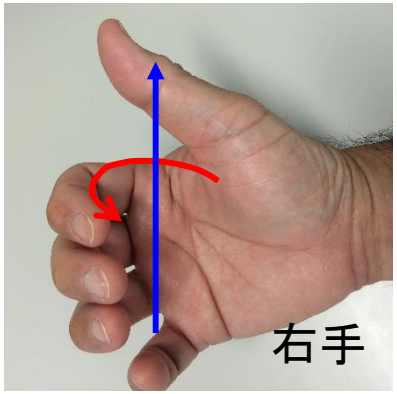
$$\mathbf{R}_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$





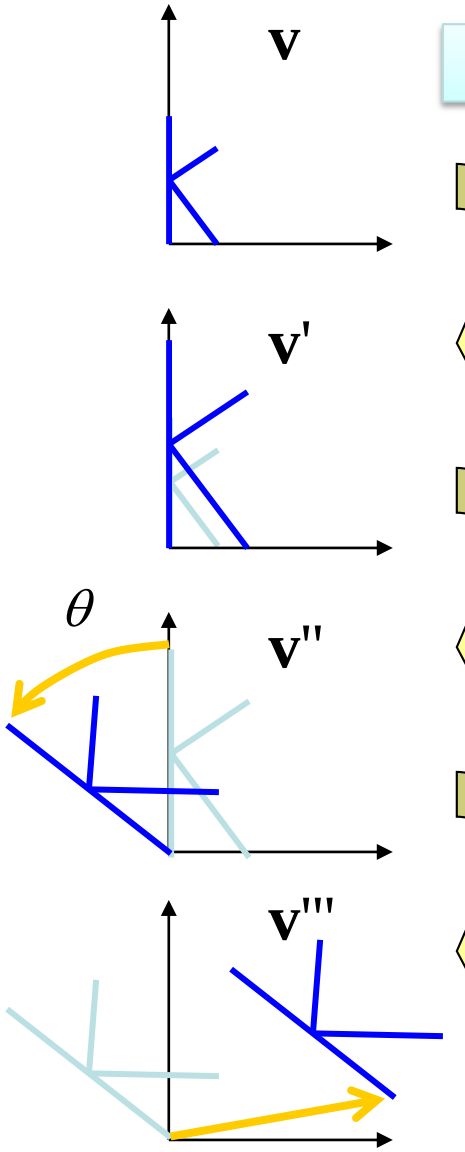






幾何変換の組合せとアフィン変換行列

復習



幾何変換操作

拡大 S

回転 R

移動 T

$\mathbf{v}' = \mathbf{S}\mathbf{v}$

$\mathbf{v}'' = \mathbf{R}\mathbf{v}'$

$\mathbf{v}''' = \mathbf{T}\mathbf{v}''$

$\mathbf{v}''' = \mathbf{TRSv}$ ⇒ $\mathbf{v}''' = \mathbf{Mv}$

アフィン変換行列

$\mathbf{M} = \mathbf{TRS}$

移動 T

回転 R

拡大 S

アフィン変換行列の並び順は
幾何変換操作と逆の順番！

基本課題8

復習

[1] 次の順番どおりに3次元図形を幾何変換した時の総合アフィン変換行列を**M**として，

$$\mathbf{M} = \begin{bmatrix} \cdot & \cdots & & \\ \vdots & \ddots & & \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

の数式を示しなさい．

レポートでは，Wordの数式機能を用いてきれいに書くこと

途中経過があれば部分点がつくかも

変換操作の順序と数式は逆順になることに注意

例えば，こんな感じの解答

$$\mathbf{M} = \begin{bmatrix} M\cos\theta_z & \cdots & \cdots \\ 0 & M\sin\theta_z\cos\theta_x & \\ -M\cos\theta_y & M & \ddots \\ 0 & 0 & 0 \quad 1 \end{bmatrix}$$

変換操作の順序

(1) x軸の周りに角度 θ_x回転する
(2) y軸の周りに角度 θ_y回転する
(3) x, y, z全ての座標値をM倍する
(4) x方向にt_x, y方向にt_y, z方向にt_zだけ移動する

[2] 次の値を代入したときのアフィン変換行列の値を電卓等を用いて計算し，変換行列**M**を数値で示しなさい．

$\theta_x = \theta_y = 15$ [度]
$M = 100$
$t_x = t_y = t_z = 200$

例えば，こんな感じの解答

$$\mathbf{M} = \begin{bmatrix} 90.0 & 5.5 & \cdots & \\ 0 & -3.1 & & \\ -20.3 & 25.2 & \ddots & \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

基本課題8［3］が3ページ後にあり

基本課題8 解答例

A君解答

(1) x軸の周りに角度 θ_x 回転する　R_x
(2) y軸の周りに角度 θ_y 回転する　R_y
(3) x, y, z全ての座標値をM 倍する　S
(4) x方向にt_x, y方向にt_y, z方向にt_zだけ移動する　T

[1]

（１）　$Rx = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta x & -\sin\theta x & 0 \\ 0 & \sin\theta x & \cos\theta x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

（２）　$Ry = \begin{bmatrix} \cos\theta y & 0 & \sin\theta y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta y & 0 & \cos\theta y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

（３）　$S = \begin{bmatrix} M & 0 & 0 & 0 \\ 0 & M & 0 & 0 \\ 0 & 0 & M & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

（４）　$T = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$

変換行列 M は

$$M = T\,S\,Ry\,Rx = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} M & 0 & 0 & 0 \\ 0 & M & 0 & 0 \\ 0 & 0 & M & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} Ry\,Rx$$

$$= \begin{bmatrix} M & 0 & 0 & tx \\ 0 & M & 0 & ty \\ 0 & 0 & M & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta y & 0 & \sin\theta y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta y & 0 & \cos\theta y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} Rx$$

$$= \begin{bmatrix} M\cos\theta y & 0 & M\sin\theta y & tx \\ 0 & M & 0 & ty \\ -M\sin\theta y & 0 & M\cos\theta y & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta x & -\sin\theta x & 0 \\ 0 & \sin\theta x & \cos\theta x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} M\cos\theta y & M\sin\theta x\sin\theta y & M\cos\theta x\sin\theta y & tx \\ 0 & M\cos\theta x & -M\sin\theta x & ty \\ -M\sin\theta y & M\sin\theta x\cos\theta x & M\cos\theta x\cos\theta y & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

基本課題8 解答例

A君解答

[2]

[1]で求めた M に　$\theta x = \theta y = 15$[度] , $M = 100$, $tx = ty = tz = 200$　を代入する。

$\sin 15 \fallingdotseq 0.259$　,　$\cos 15 \fallingdotseq 0.966$ なので、

$$M = \begin{bmatrix} 100\cos 15^\circ & 100\sin 15^\circ \sin 15^\circ & 100\cos 15^\circ \sin 15^\circ & 200 \\ 0 & 100\cos 15^\circ & -100\sin 15^\circ & 200 \\ -100\sin 15^\circ & 100\sin 15^\circ \cos 15^\circ & 100\cos 15^\circ \cos 15^\circ & 200 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 96.6 & 6.7081 & 25.0194 & 200 \\ 0 & 96.6 & -25.9 & 200 \\ -25.9 & 25.0194 & 93.3156 & 200 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

有効数字に注意！

となる。

アフィン変換を行う関数

復習

Example8-1

$$\mathbf{M} = \begin{bmatrix} m[0][0] & m[0][1] & m[0][2] & m[0][3] \\ m[1][0] & m[1][1] & m[1][2] & m[1][3] \\ m[2][0] & m[2][1] & m[2][2] & m[2][3] \\ m[3][0] & m[3][1] & m[3][2] & m[3][3] \end{bmatrix}$$

```
struct Point3D  //3次元の座標点
{
    double x;
    double y;
    double z;
};

void AffineTransform(Point3D p[], int n, double m[][4]) //n個の座標点p[]をアフィン行列mで変換
{
    int i;
    for (i = 0; i < n; i++)
    {
        double x = p[i].x, y = p[i].y, z = p[i].z;
        p[i].x = m[0][0]*x + m[0][1]*y + m[0][2]*z + m[0][3];
        p[i].y = m[1][0]*x + m[1][1]*y + m[1][2]*z + m[1][3];
        p[i].z = m[2][0]*x + m[2][1]*y + m[2][2]*z + m[2][3];
    }
}

int main(void)      // 以下はAffineTransform()の使い方の例. 意味はない.
{
    Point3D p3[] = { {10, 20, 30}, {15, 30, 20}, {20, 20, 20} };  //3次元の三つの点
    double mat[][4] = { {1.0,  0.0, 0.0, 10.0},// y方向に20倍拡大し, x方向に10移動する変換行列
                        {0.0, 20.0, 0.0,  0.0},
                        {0.0,  0.0, 1.0,  0.0},
                        {0.0,  0.0, 0.0,  1.0}  }
    AffineTransform(p3, 3, mat);        // p3の各点を変換行列matで変換
}
```

注) C言語では2次元配列を引数にする場合は二つ目のインデックスの要素数を明示しなければならない

復習

投影

3次元空間で与えられた図形を2次元平面の図形に変換すること
(3次元図形をコンピュータで表示するためにはこれが必要)

主な投影法

i. 正投影
ii. 斜投影
iii. 透視投影

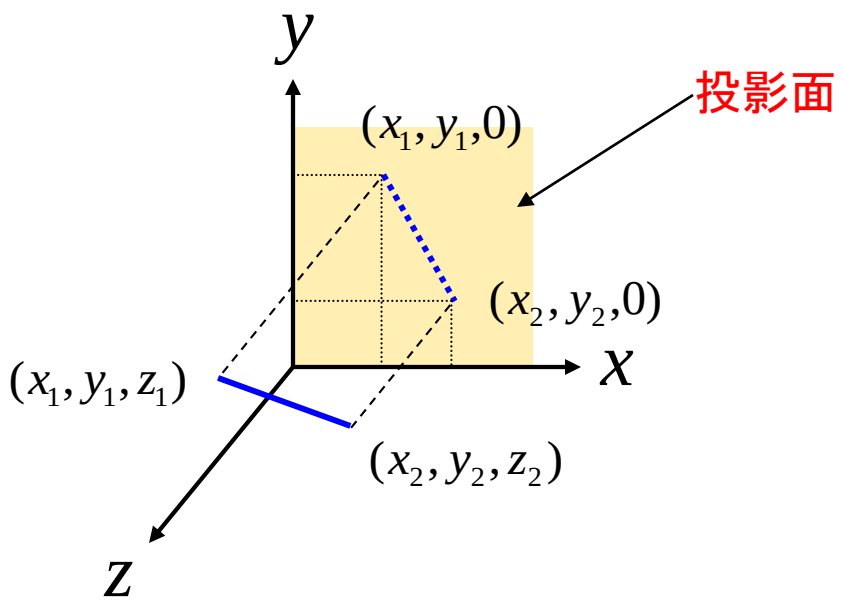
・・・他にもたくさんある

xy平面への正投影
(z座標値の除去)

もしも正投影を変換行列として書いたら. .

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

実際にはz値を無視するだけなので変換行列を用いる必要は無い.



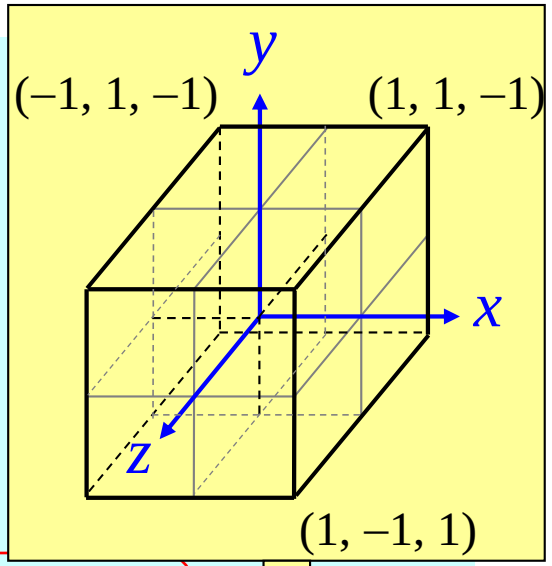
基本課題8[3]

ソースと実行 復習

以下は辺の長さが2である立方体の図形を，アフィン変換・正投影して描くプログラムである．このプログラムを完成せよ．

Report8-1

```
#include  <stdlib.h>
#include  <stdio.h>
#include  <math.h>
#include  "cglec.h"
// struct Point, CglDrawLine(), CglDrawLines()は"cglec.h"で定義済み
// struct Point3D, AffineTransform()はこれ以前で定義・宣言しておく
#define WIDTH 400
#define HEIGHT 400
int main(void)
{
    unsigned char data[WIDTH][HEIGHT];
    Image img = { (unsigned char*) data, WIDTH, HEIGHT };

    Point3D cube[] = {                                          //立方体の線図形データ
    {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;     // 点データの数は24個

    // ============== ここで基本課題8[2]で求めたアフィン変換行列を定義する
    double mat[][4] = {   // 行列の数値を入れる
                          //   ・・・
                       };  //
```



モデル

各自でプログラムする内容

i) 基本課題8[2]で計算した数値をここに書き込む

ソースは次のページに続く

基本課題8[3]続き

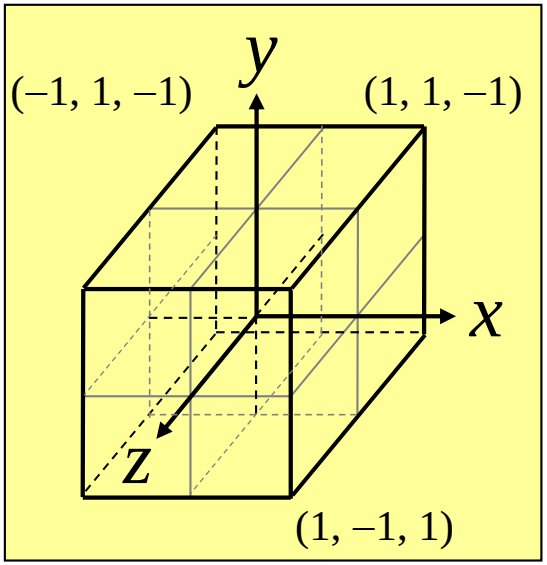
復習

```
    CglSetAll(img, 0);
    AffineTransform(cube, N, mat)     // アフィン変換を実行

    Point p[24];    // 正投影後の点データを入れる2次元の配列
    // ============== ここからプログラムする
    // アフィン変換したcubeを正投影する. cube[]→p[]
    // CglDrawLines()を用いて画像imgに2次元図形p[]を描く

    CglSaveGrayBMP(img, "cube.bmp");
}
```

各自でプログラムする内容

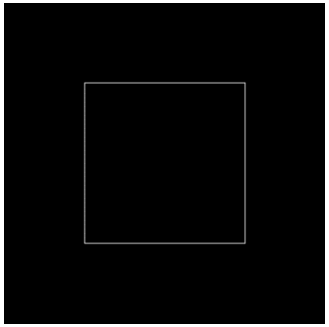
ii) Point3D型配列の3次元点データcubeを正投影(z値を無視)してPoint型配列の点データpに代入

iii) 変換した点データpとCglDrawLines()関数を用いて2次元画像を描画



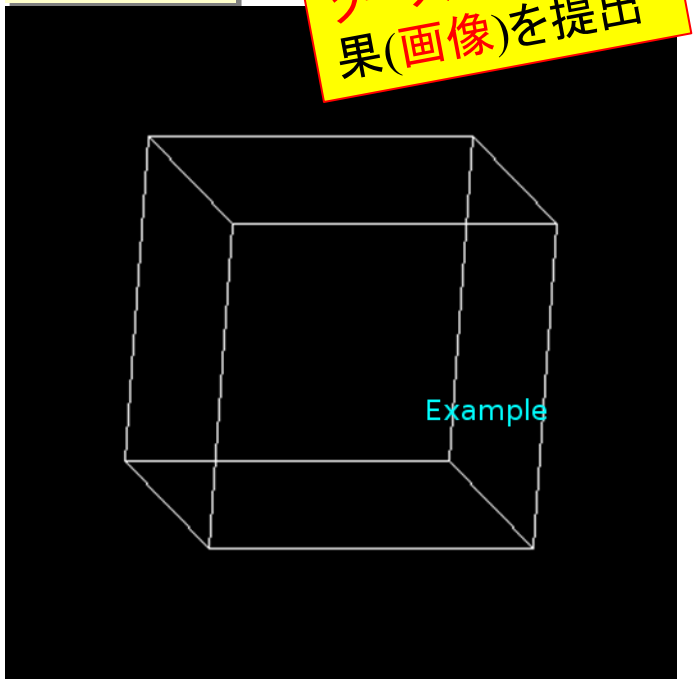
アフィン変換前の立方体データ

仮に回転せずに移動・拡大だけで正投影したら・・・



実行結果

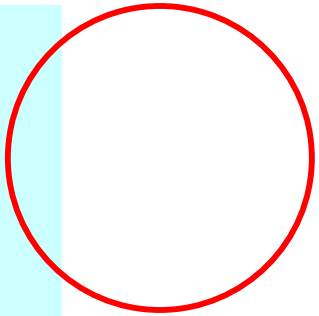
ソースと実行結果(画像)を提出



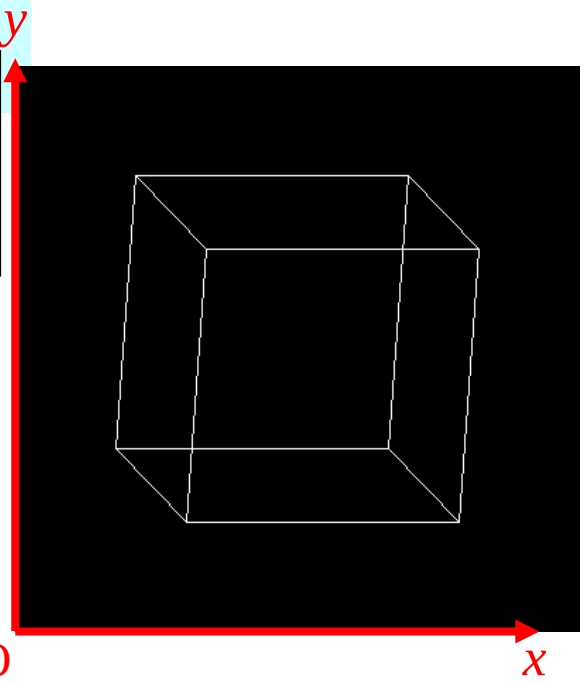
基本課題8[3] 解答例

A君解答

```
// ============== ここで課題-1で求めたアフィン変換行列を定義する
double mat[][4] = { { 96.6,   6.7081,  25.0194,  200},
                    {    0,     96.6,    -25.9,  200},
                    {-25.9,  25.0194,  93.3156,  200},
                    {    0,        0,        0,    1}};

CglSetAll(img, 0);
AffineTransform(cube, N, mat);// アフィン変換を実行
// ============== ここからプログラムする
// アフィン変換したcubeを正投影してimg上に図形を描く
int i;
for(i = 0;i < N;i++){       //Point p に  Point3D cube を入れる
    p[i].x = (int)(cube[i].x + 0.5);
    p[i].y = (int)(cube[i].y + 0.5);
}
CglDrawLines(img, p, N, 255);
```



(注)この問題の場合，アフィン変換後の図形データはすべて正の数(そうでなければそもそも画像として描画できない)




```
//3次元の座標点
struct Point3D
{
    double x;
    double y;
    double z;
};
```

実数座標値
負の数もOK

```
//2次元の座標点
struct Point
{
    int x;
    int y;
};
```

整数座標値
負の数はダメ

発展課題8

復習

基本課題8で行った3次元図形のアフィン変換と投影は，[1]～[2]の行例の手計算がいささか面倒であるので，基本課題と同じ図形を次の方法で描画せよ．

1. 基本課題の図形座標データCube[]とAffineTransform()関数はそのまま利用する．
2. 基本課題の四つのアフィン変換行列を数値で求める．
   - $\mathbf{R}_x$　x軸の周りに15度回転
   - $\mathbf{R}_y$　y軸の周りに15度回転
   - $\mathbf{S}$　100倍に拡大
   - $\mathbf{T}$　x, y, z方向に200ピクセル移動
3. これらの四つのアフィン変換行列を用いてAffineTransform()関数を正しい順番で４回実行し，図形データCube[]の座標を変換する．
4. 基本課題と同様に，変換後の座標から正投影により２次元図形を描いて保存する．

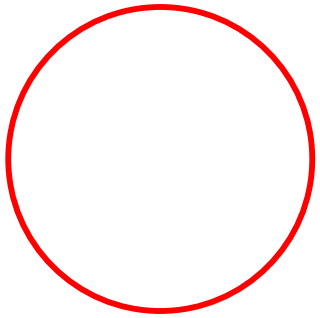
発展課題 解答例

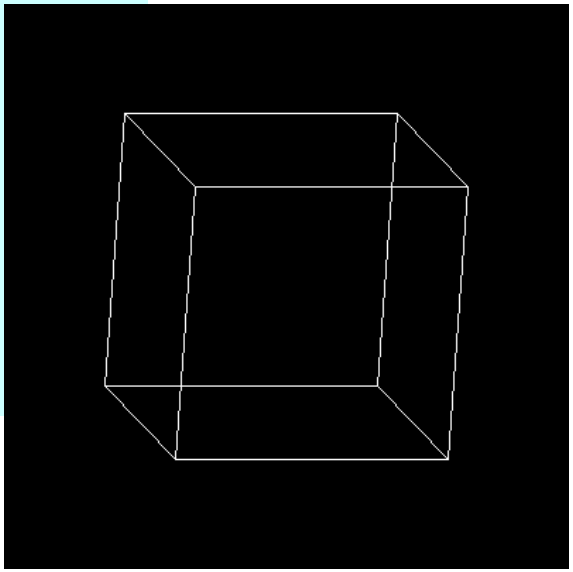
A君解答

```
double Rx[][4] = {
        {1,  0,  0,  0},
        {0,  0.9659,  -0.2588,  0},
        {0,  0.2588,  0.96590,  0},
        {0,  0,  0,  1}
    };
    double Ry[][4] = {
        {0.96590,  0,  0.2588,  0},
        {0,  1,  0,  0},
        {-0.2588,  0,  0.96590,  0},
        {0,  0,  0,  1}
    };
    double S[][4] = {
        {100,0,0,0},
        {0,100,0,0},
        {0,0,100,0},
        {0,0,0,1}
    };
    double T[][4] = {
        {1,0,0,200},
        {0,1,0,200},
        {0,0,1,200},
        {0,0,0,1}
    };

    CglSetAll(img,  0);

    AffineTransform(cube,  N,  Rx);
    AffineTransform(cube,  N,  Ry);
    AffineTransform(cube,  N,  S);
    AffineTransform(cube,  N,  T);
```

B君解答

```
double w=1.0/12.0*3.14;
double rx[][4] = {  {1,0,0,0},
                    {0,cos(w),-sin(w),0},
                    {0,sin(w),cos(w),0},
                    {0,0,0,1}
                  };
double ry[][4] = {  {cos(w),0,sin(w),0},
                    {0,1,0,0},
                    {0,sin(w),cos(w),0},
                    {0,0,0,1}
                  };
double s[][4] ={  {100,0,0,0},
                  {0,100,0,0},
                  {0,0,100,0},
                  {0,0,0,100}
                 };
double t[][4] = {{1,0,0,200},
                  {0,1,0,200},
                  {0,0,1,200},
                  {0,0,0,1}
                 };
CglSetAll(img,  0);
AffineTransform(cube,  N,  rx);
AffineTransform(cube,  N,  ry);
AffineTransform(cube,  N,  s);
AffineTransform(cube,  N,  t);
```

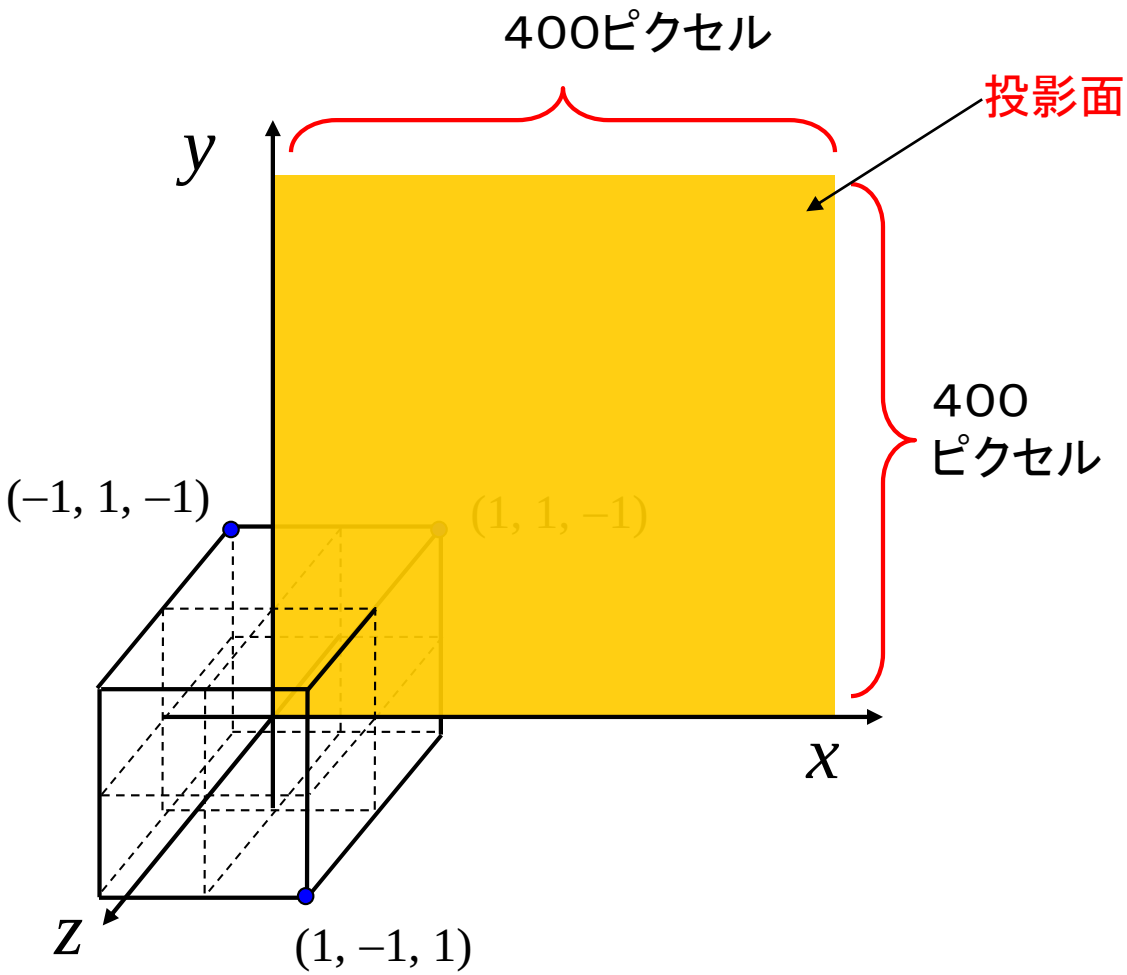




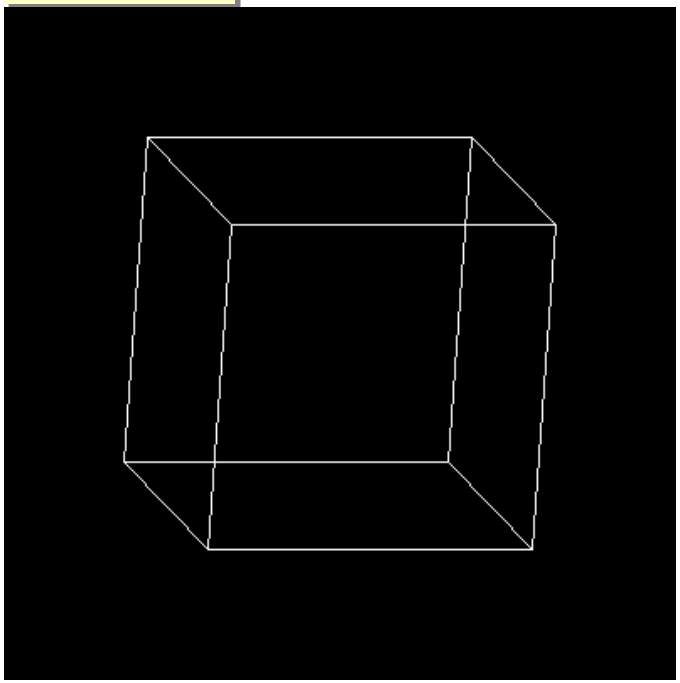
幾何変換(アフィン変換)が必要な理由

(1) 回転 ・・・理由：回転しないと立体ぽく見えないので
(2) 拡大 ・・・理由：400×400ピクセルの画像に正投影するので
(3) 移動 ・・・理由：元データは負の座標値を含むので

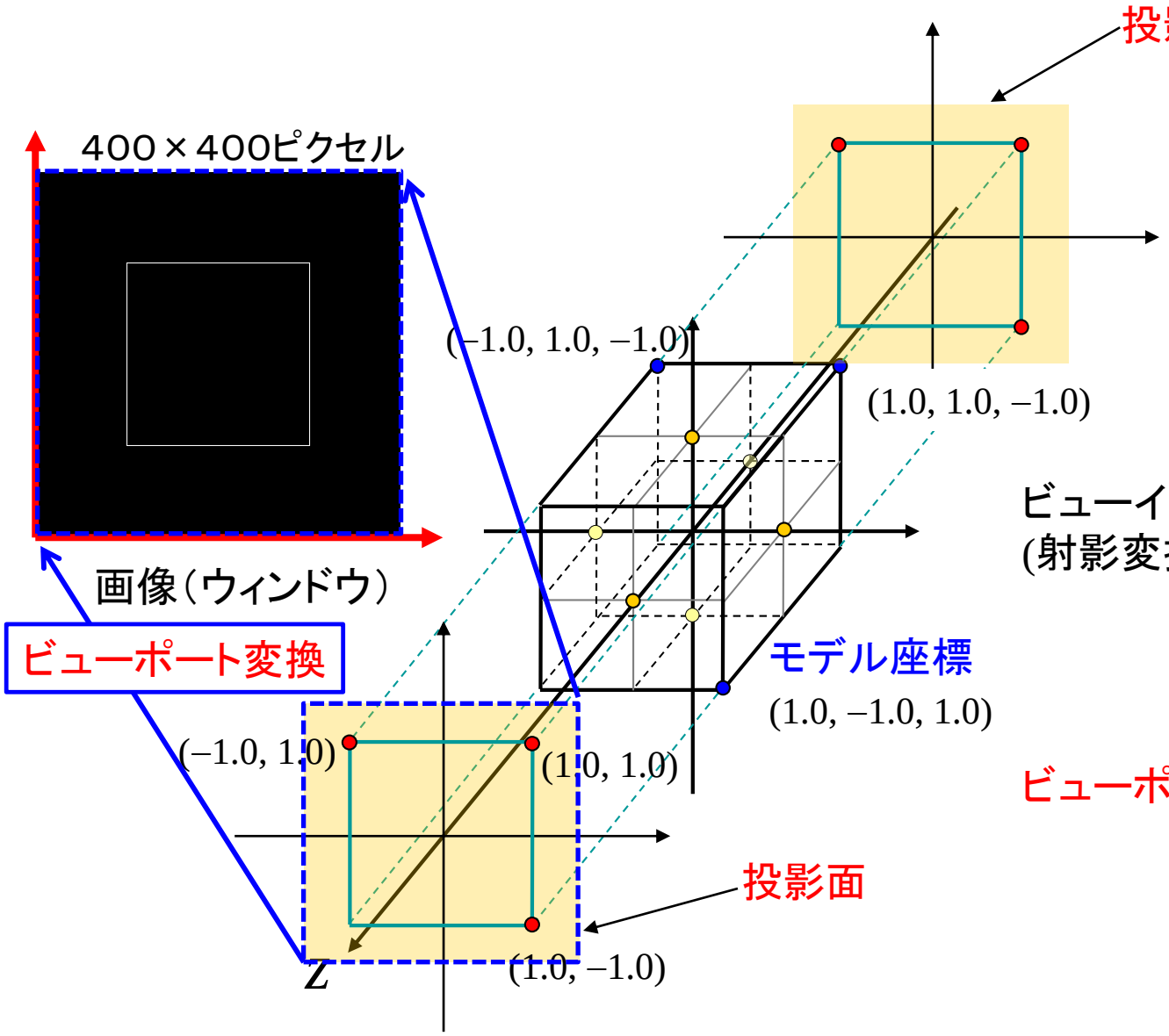
元の図形データ(モデル)を簡単にデザインしたい！



実行結果



モデル座標からCG画像までの変換(平行投影の例)



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)

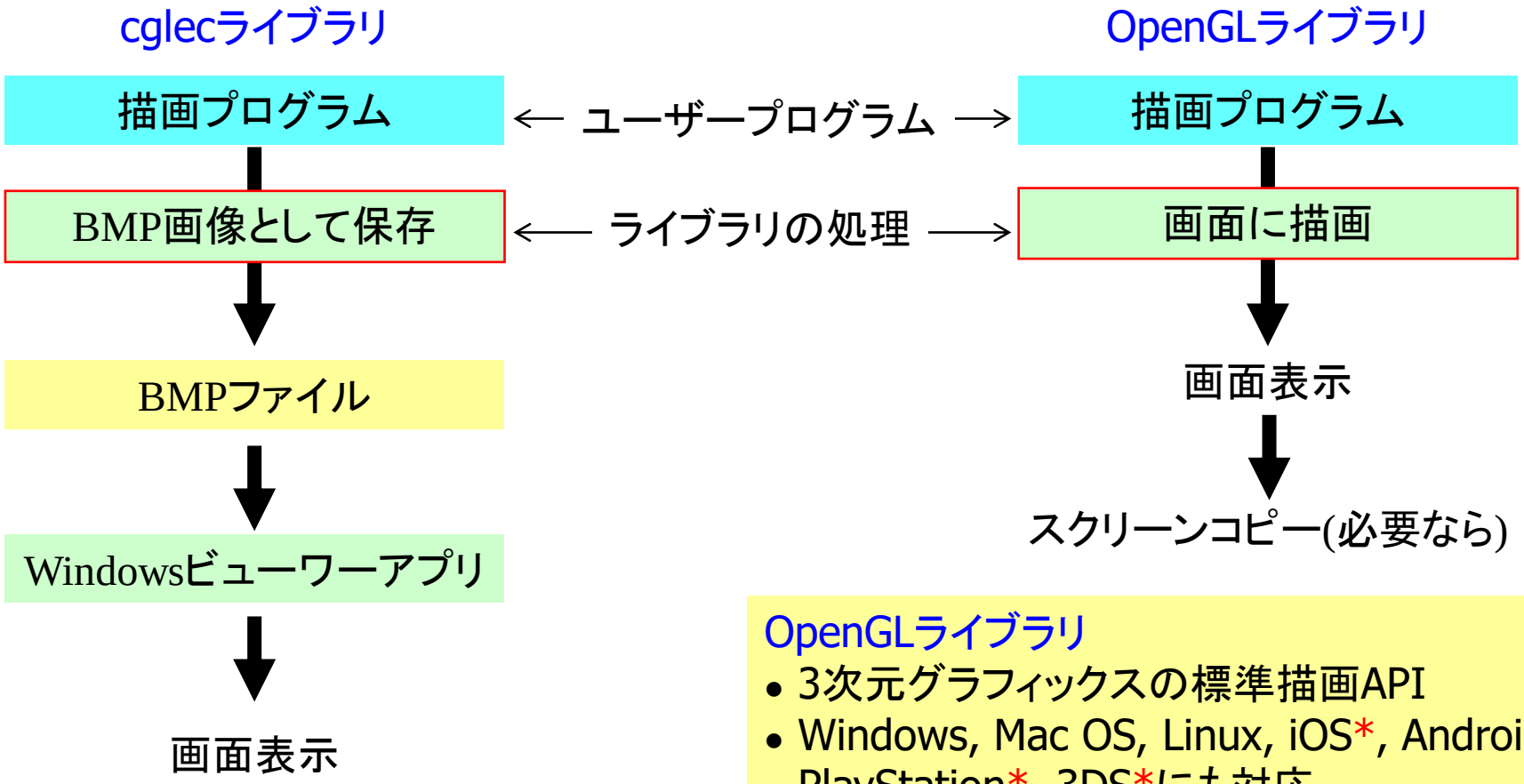
ビューイング変換
(射影変換)
⇩
平行投影
透視投影

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ライブラリ

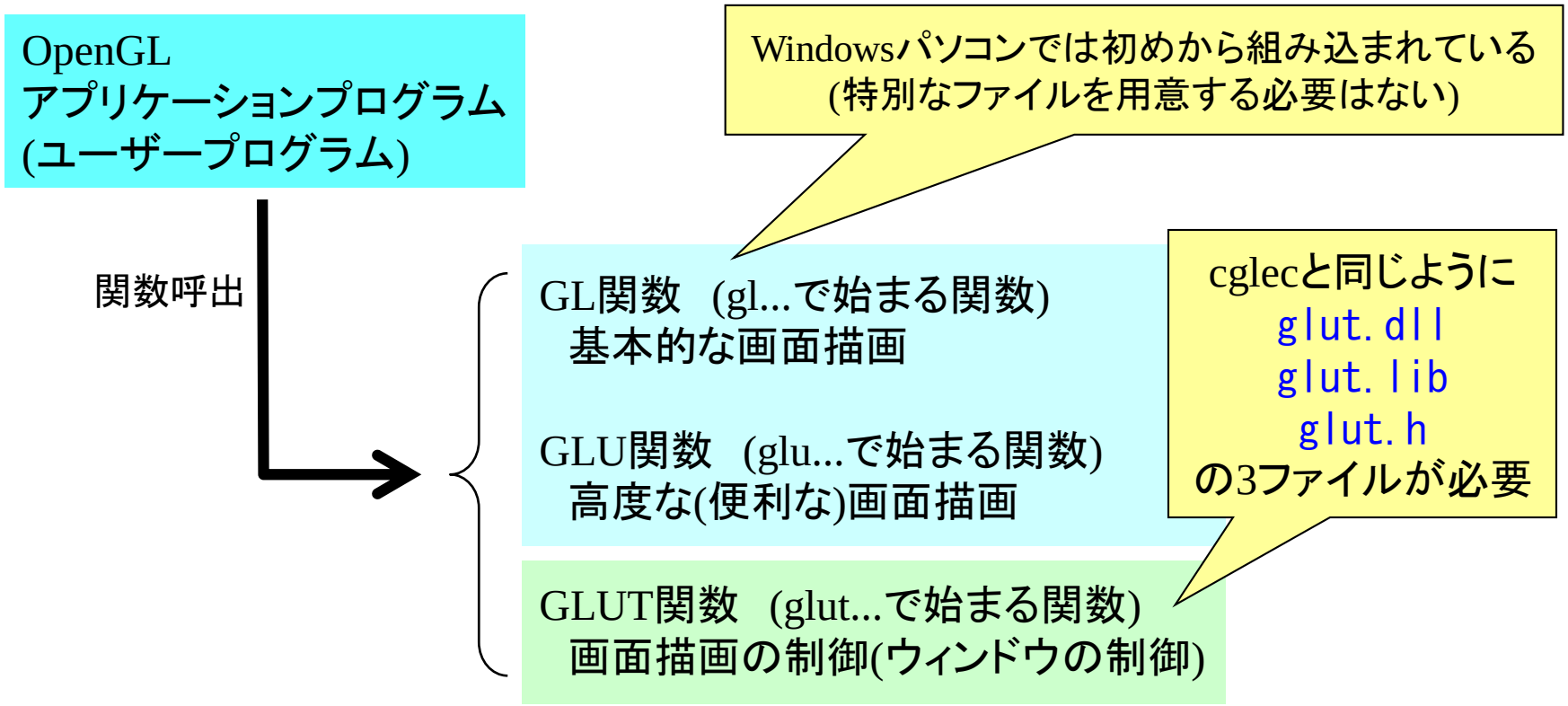


OpenGLライブラリ

- 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の利用



GLUT関数を使うには

関大LMSから，GLUTライブラリ(glut-376-bin.zip)をダウンロードし，それを解凍して中にある次の3つのファイルをVisual Studioのプロジェクトフォルダーに入れておく

- glut.h
- glut.lib　⇒ この2つはコンパイル(ビルド)時にのみ必要なファイル
- glut.dll　⇒ プログラム実行時に必要なファイル

注) 新たにプロジェクトを作成するたびにそのプロジェクトフォルダーに3つのファイルをコピーしておく必要がある.

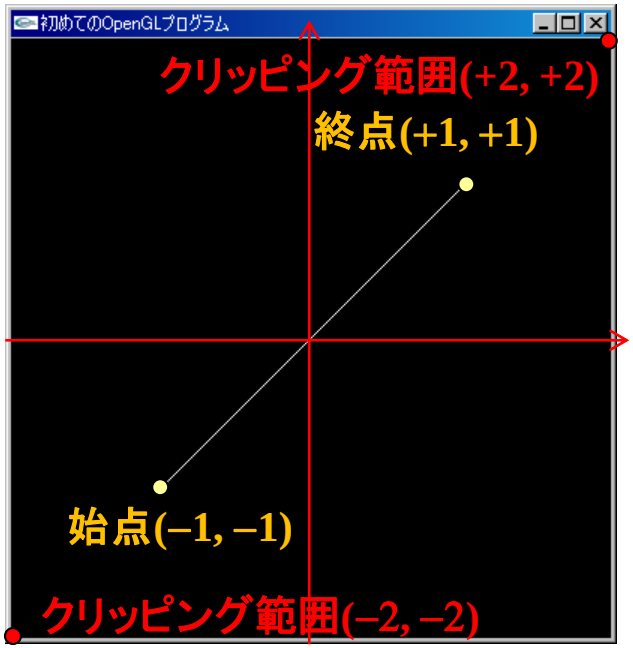
OpenGLを用いたプログラム

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();
}
```

モデルの形を決める

表示方法を決める



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

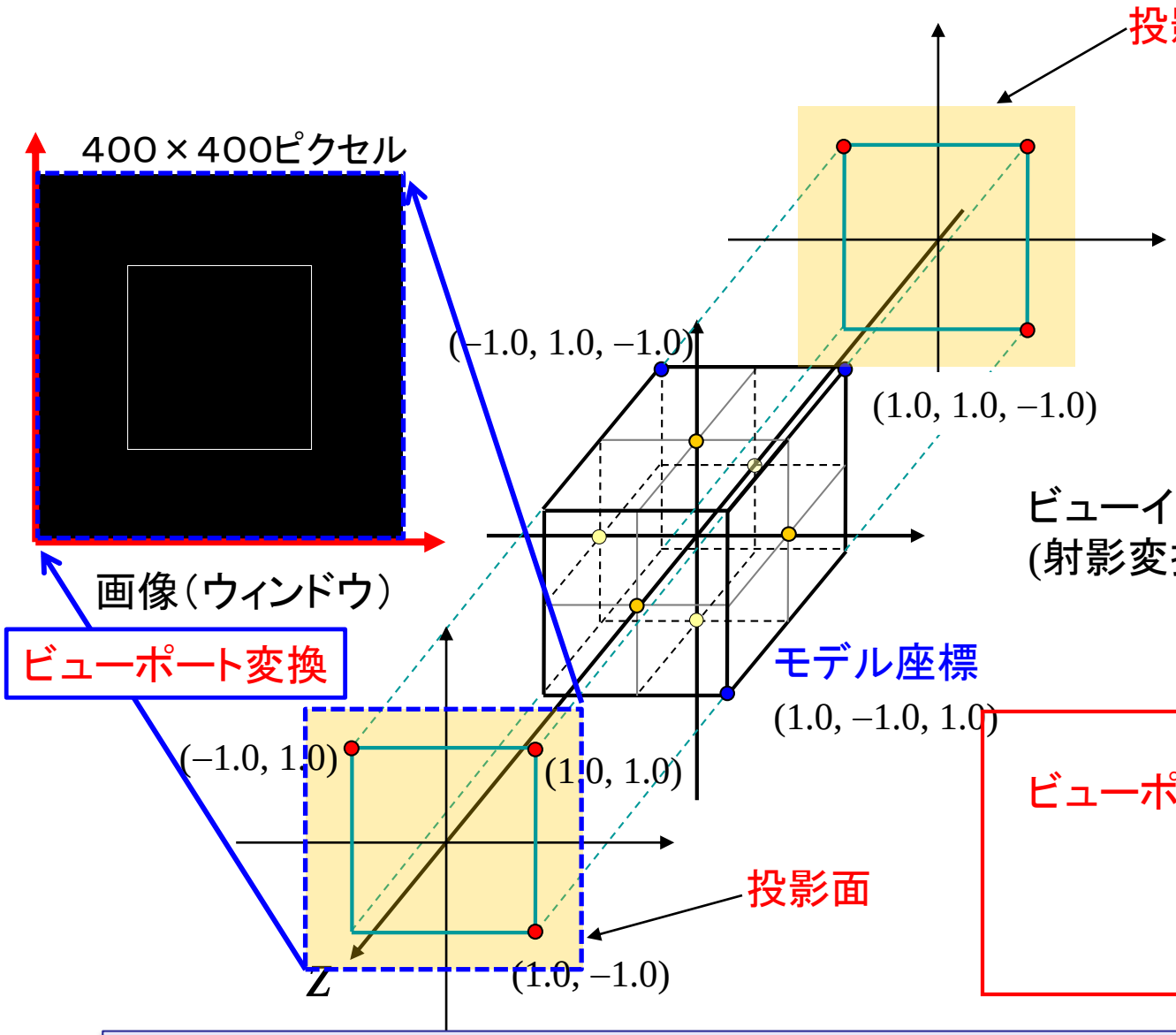
ウィンドウの最初の色．ここでは黒．R, G, Bの順．
階調値は0.0～1.0の範囲　(1.0, 1.0, 1.0)→白

平行投影と投影面座標(モデル座標)の設定
投影面座標がビューポート変換される範囲
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)



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

OpenGLを用いたプログラム

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**

平行投影と投影面座標(モデル座標)の設定
投影面座標がビューポート変換される範囲
*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

400ピクセル

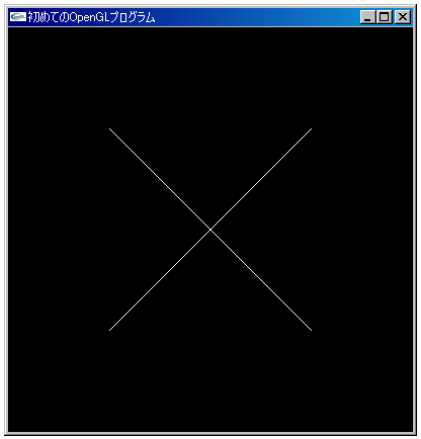
描画関数(1)

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

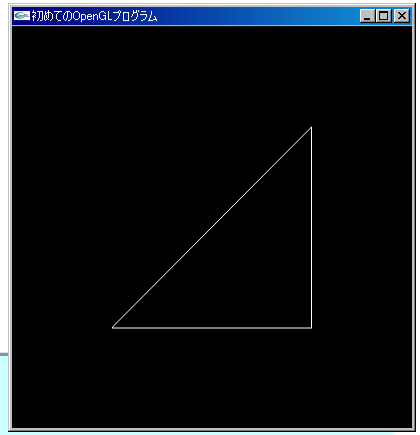


glVertex3f()関数
頂点座標を指定する

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

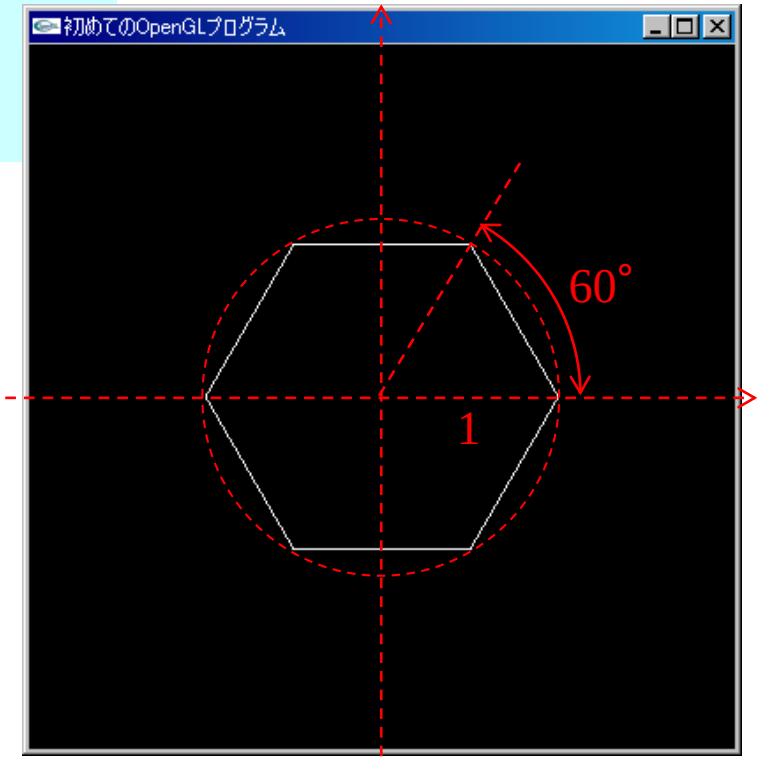
```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT );
    glColor3f( 1.0, 1.0, 1.0 );
    glBegin( GL_LINE_LOOP );
        glVertex3f(-1.0, -1.0, 0.0);   //1点目
        glVertex3f(+1.0, -1.0, 0.0);   //2点目
        glVertex3f(+1.0, +1.0, 0.0);   //3点目
    glEnd();
    glFlush();
}
```

描画関数(2)

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.



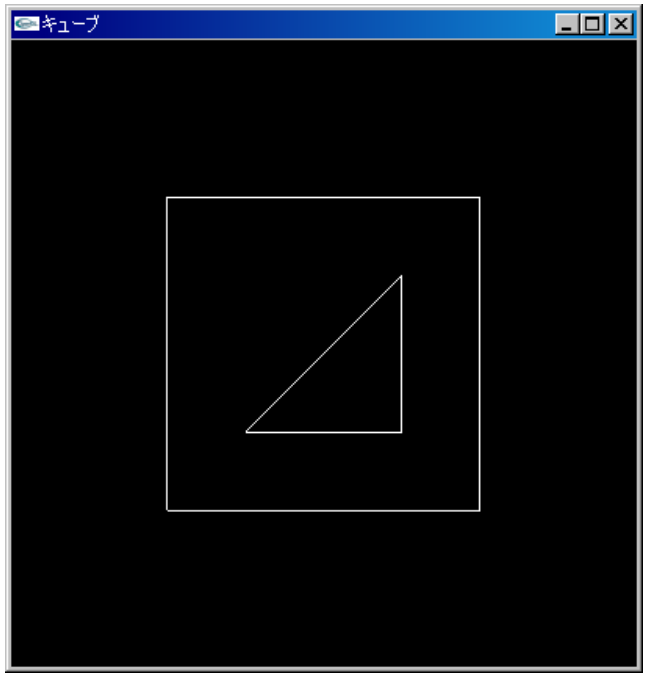
glBegin()関数とglEnd()関数の間で呼び出されたglVertex3f()関数がモデルの頂点を与える.

描画関数(3)

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

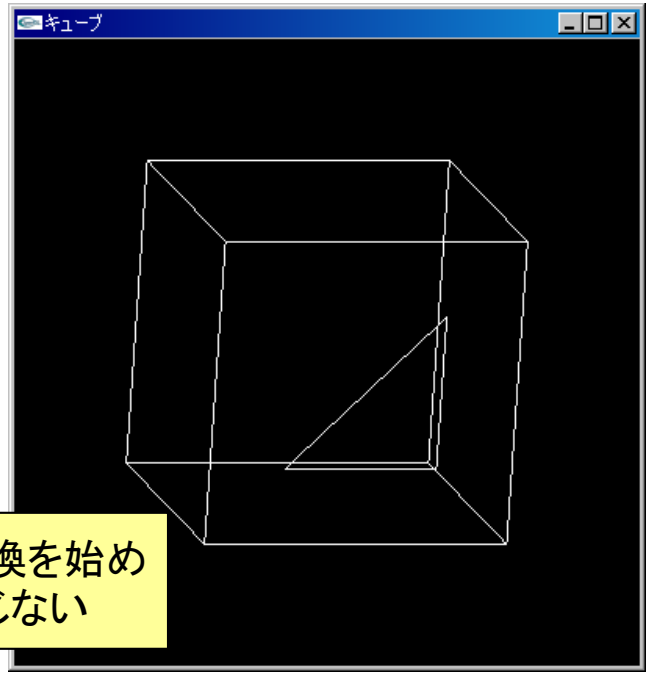


glBegin()・・・glEnd()は二つ以上いくつあってもよい

3次元アフィン変換

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

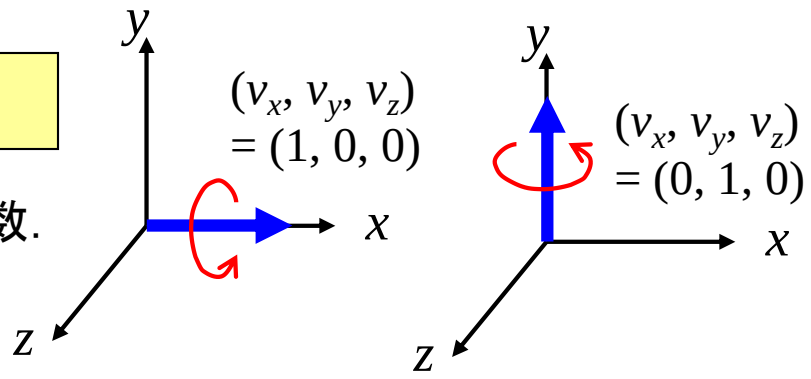
図形を回転するアフィン変換行列の設定

```
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

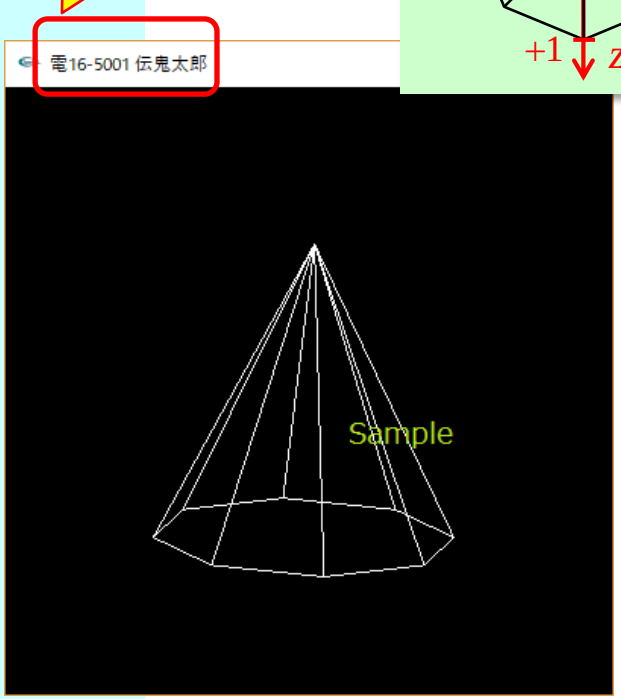
次のmain()をそのまま用いて，OpenGLの描画関数で次の八角錐をプログラムせよ．

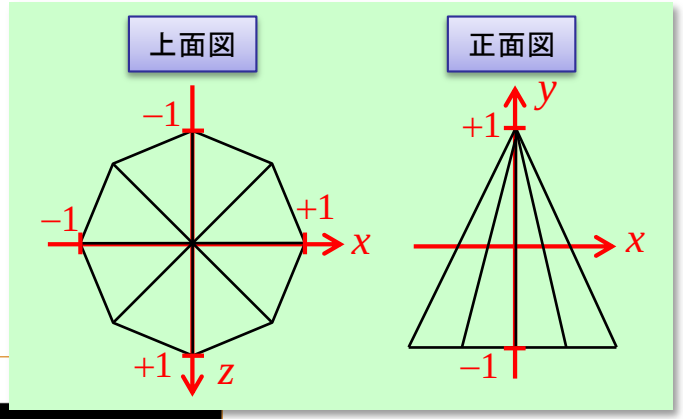
- 八角錐の底面は$(x, -1, z)$平面にある
- 底面の八角形の中心は$(0, -1, 0)$であり，八角形の対角線の長さは2である．
- 頂点の座標は$(0, +1, 0)$である．

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

発展課題9

基本課題9のmain()をそのまま用いて，OpenGLの描画関数で次の八角柱をプログラムせよ．

- 八角柱の上面は$(x, +1, z)$平面にある
- 八角柱の下面は$(x, -1, z)$平面にある
- 上面・下面の八角形の中心は$(0, \pm 1, 0)$であり，八角形の対角線の長さは2である．