

2019年度コンピュータグラフィックス 中間小テスト

場所 : 3201 教室
(3号館2階, 屋上にテニスコート)
日付 : 5月31日(金)
時間 : 14:40から15~20分程度

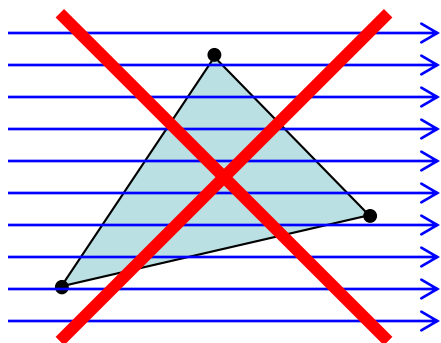
C言語ソースファイルの穴埋め問題
持込み・参照は一切不可

受験には学生証が必要！ 座席指定！ 遅刻に注意！

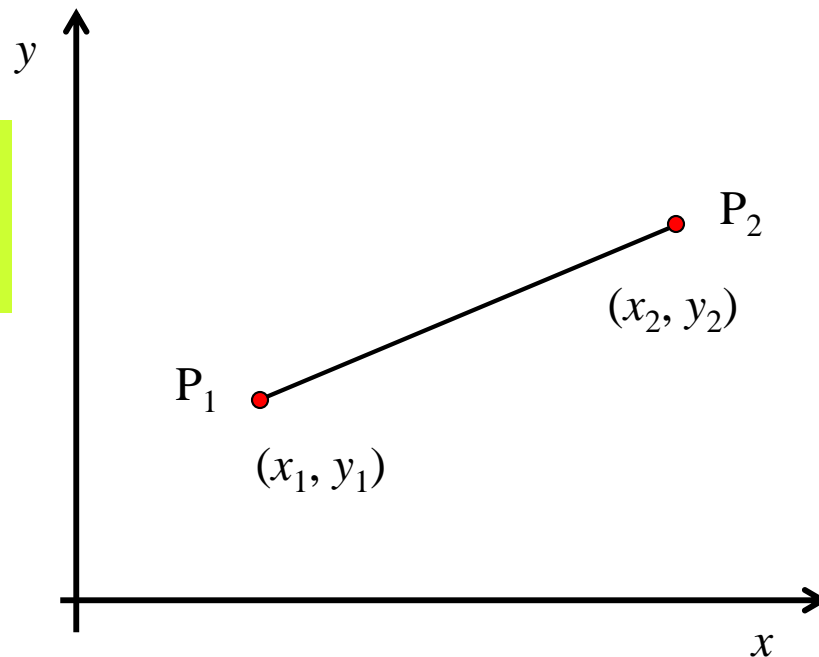
小テスト終了後にOD2教室に移動して通常の授業をします。

直線の描画

2点 P_1, P_2, P_3 の座標を与えたとき、
この2点を結ぶ線分を描画する関
数を考える

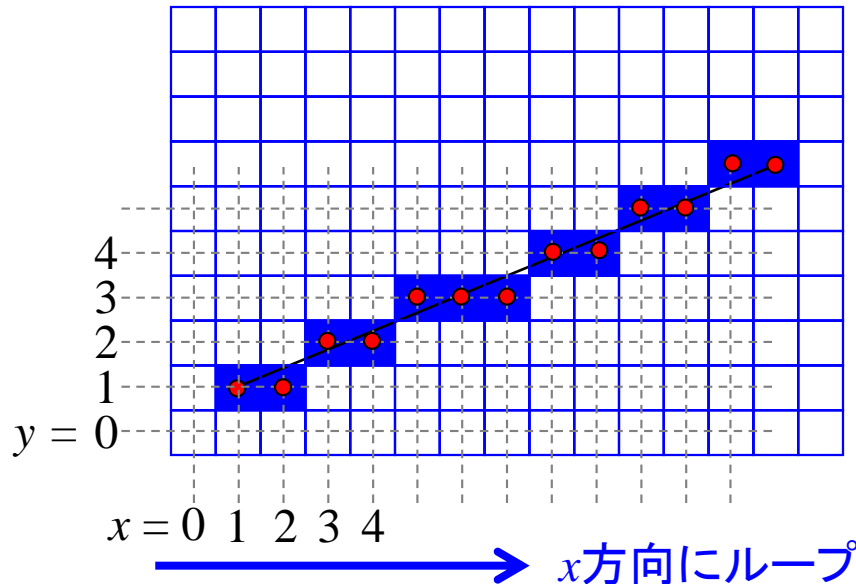


ラスタスキャン



$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

直線の描画(2): 単純なアルゴリズム



$$y = m(x - x_1) + y_1$$

$$m \equiv \frac{y_2 - y_1}{x_2 - x_1}$$

傾きは整数とは限らない

y' は整数とは限らない

- (1) x を1増やす
- (2) $y' = m(x - x_1) + y_1$ を求める
- (3) y' を四捨五入して y を求める
- (4) (x, y) のピクセルのグレイレベルを変える

Example6-1

```
void DrawLine0(Image img, int x1, int y1, int x2, int y2, int g)
{
    // とりあえずクリッピングやパラメータチェックはやらない
    double m = (double) (y2 - y1) / (x2 - x1); // 直線の傾き.
    int x, y;
    for (x = x1; x <= x2; x++)
    {
        y = (int) (m * (x - x1) + y1 + 0.5); // 四捨五入して整数に変換
        *(img.Data + x*img.Ny + y) = g;
    }
}
```

(double) は double型へのキャスト. 割り算で少数以下が切り捨てられないようにする.

$x_1 == x_2$?
ゼロ除算

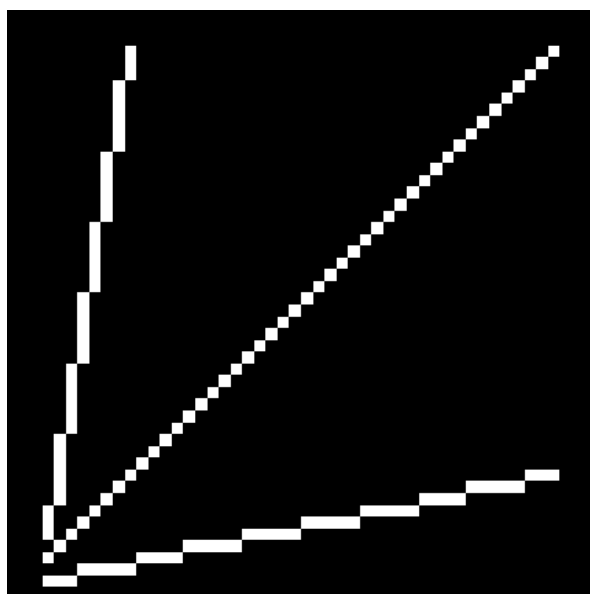
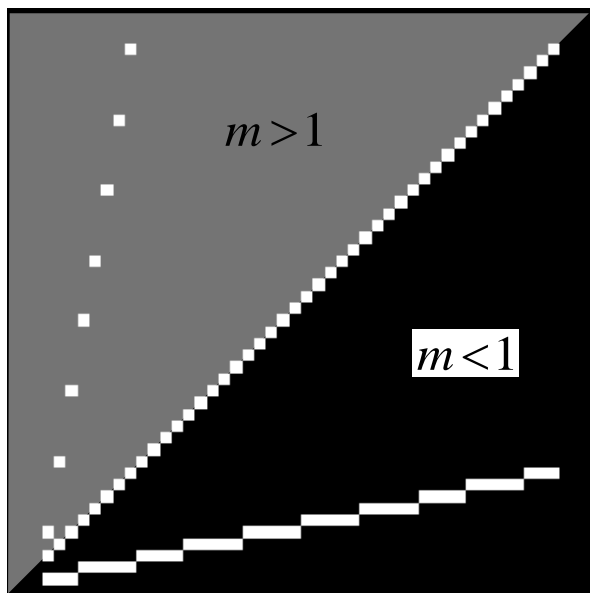
(int) はint型へのキャスト. 少数以下は切り捨てられる

0.5を足して少数以下を切り捨てると, 四捨五入

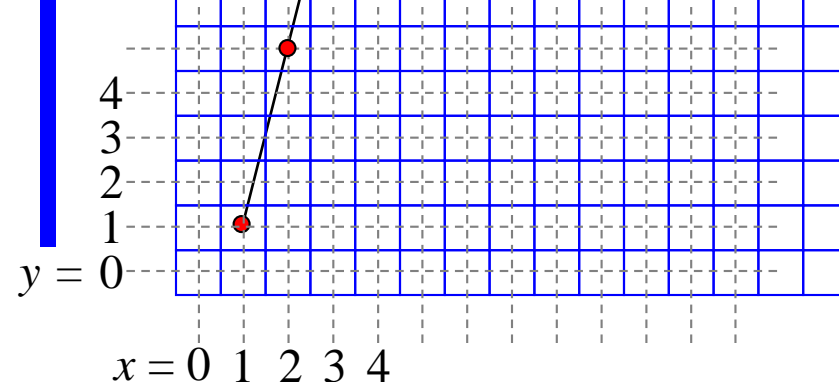
基礎プログラミング
講習第12回

直線の描画: 単純なアルゴリズムの結果

復習

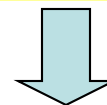


y方向に
ループ



対処法

$m > 1$ と $m < 1$ では別に扱う



$m > 1$ の場合は x と y の役割を逆転させる

$$x = (y - y_1) / m + x_1$$

直線描画アルゴリズムのまとめ

傾き m をチェック

m の計算時にゼロ除算
が発生しないように！

$$m \equiv \frac{y_2 - y_1}{x_2 - x_1}$$

◆ $|m| \leq 1$ であれば

- x_1 と x_2 を比較
- $x_1 > x_2$ なら P_1 と P_2 の座標値を入れ替え
- x_1 から x_2 までループする

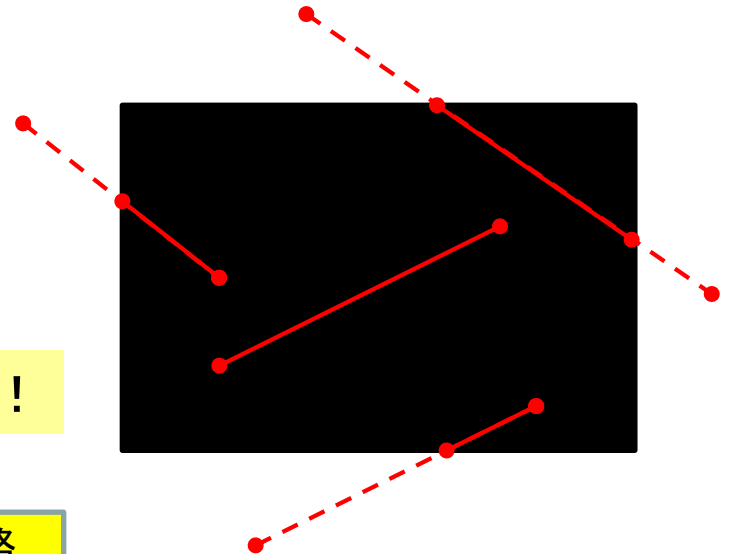
もしも $x_1 = x_2$ だったら, 垂
直方向の直線

◆ $|m| > 1$ であれば,

- y_1 と y_2 を比較
- $y_1 > y_2$ なら P_1 と P_2 の座標値を入れ替え
- y_1 から y_2 までループする

注) 直線のクリッピングは複雑！

この授業では省略



基本課題6

復習

点 $P_1(x_1, y_1)$ と点 $P_2(x_2, y_2)$ の間にグレイレベル g の線分を描画する`DrawLine()`関数を作成せよ。作成した関数と下記の`main()`を組み合わせる実行結果と同じ画像を得よ。なお、この関数ではクリッピングは考慮しなくて良い。
(点 P_1 と点 P_2 は常に画像の範囲内にあるものとする) ただし、ゼロ除算を発生させないこと。

Report6-1

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cglec.h"

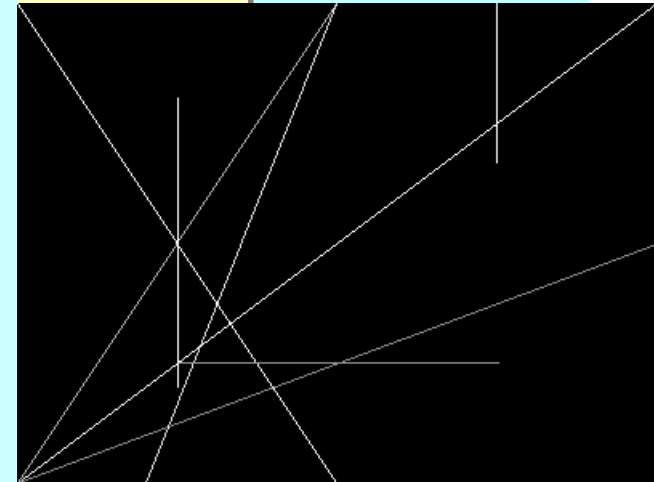
void DrawLine(Image img, int x1, int y1, int x2, int y2, int g)
{ /* この部分をプログラム */ }

int main(void)
{   int Nx, Ny;
    printf("画像の横方向ピクセル数は? "); scanf("%d", &Nx);
    printf("画像の縦方向ピクセル数は? "); scanf("%d", &Ny);
    unsigned char* data = (unsigned char*)
        malloc(sizeof(unsigned char) * Nx * Ny);

    if (data == NULL)
    {   printf("メモリエラー!!");   exit(0);   }
    Image img = { (unsigned char*) data, Nx, Ny };
    CglSetAll(img, 0); // imgをグレイレベル0でクリアする
    DrawLine(img, 0, 0, Nx - 1, Ny - 1, 255);
    DrawLine(img, 0, 0, Nx - 1, Ny / 2 - 1, 70);
    DrawLine(img, 0, 0, Nx / 2 - 1, Ny - 1, 128);
    DrawLine(img, 0, Ny - 1, Nx / 2 - 1, 0, 255);
    DrawLine(img, Nx / 2 - 1, Ny - 1, Nx / 5, 0, 200);
    DrawLine(img, 3 * Nx / 4 - 1, Ny - 1, 3 * Nx / 4 - 1, 2 * Ny / 3, 255);
    DrawLine(img, Nx / 4, Ny / 4, 3 * Nx / 4, Ny / 4, 80);
    DrawLine(img, Nx / 4, Ny / 5, Nx / 4, 4 * Ny / 5, 255);
    free(data);
}
```

- ・ コンソール画面と画像の両方を提出！
- ・ 実行例を二つ提出！

実行結果



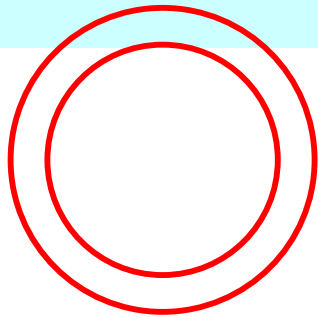
画像の横方向ピクセル数は? 400
画像の縦方向ピクセル数は? 300
続行するには何かキーを押してください . . .

基本課題6 解答例

A君解答

```
void DrawLine(Image img, int x1, int y1,
int x2, int y2, int g)
{ /* この部分をプログラム*/
    int x, y, c, ys=y1, ye=y2;

    double m;
    if(x1==x2) { //傾き∞の回避
        if(y1>y2) {
            ys=y2;
            ye=y1;
        }
        for (y=ys; y<=ye; y++) {
            *(img.Data + x1*img.Ny + y) = g;
        }
    }
    else{
```



```
        m=((double)(y2-y1)/(double)(x2-x1)); //傾き算出
        if(m > -1 && m < 1) { //xループ条件
            if(x1 > x2) {
                c=x1;
                x1=x2;
                x2=c;

                c=y1;
                y1=y2;
                y2=c;
            }
            for (x=x1; x<=x2; x++) {
                y = (int) (m * (x-x1) + y1 + 0.5);
                *(img.Data + x*img.Ny + y) = g;
            }
        }
        else{ //yループ条件
            if(y1>y2) {
                c=x1;
                x1=x2;
                x2=c;

                c=y1;
                y1=y2;
                y2=c;
            }
            for (y=y1; y<=y2; y++) {
                x = (int) ((y-y1)/m + x1 + 0.5);
                *(img.Data + x*img.Ny + y) = g;
            }
        }
    }
}
```

基本課題6 解答例

B君解答

```
void DrawLine(Image img, int x1, int y1, int x2, int y2, int g)
{
    double m = (double) (y2 - y1) / (x2 - x1);
    int x, y;

    if (x1 == x2)
    {
        for (y = y1; y <= y2; y++)
        {
            x = x1; //x軸をd固定し、mの発散を阻止する
            *(img.Data + x*img.Ny + y) = g;
        }
    }
    else
    {
        .
        .
        .
    }
}
```

この行の実行でゼロ
除算が発生！！

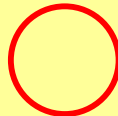
プログラムの目的や、コンパイラ、システム等によってはゼロ除算が問題にならないこともある(実際この課題では大きな問題は生じない)。しかし、場合によっては重大な問題につながることもあるため、基本的に**ゼロ除算を回避**すること。

if文の基本がわかっていない人！

```
if (-1 < m < 1)
{
    . . .
}
```



```
if (-1 < m && m < 1)
{
    . . .
}
```



ゼロ除算処理に問題がある例

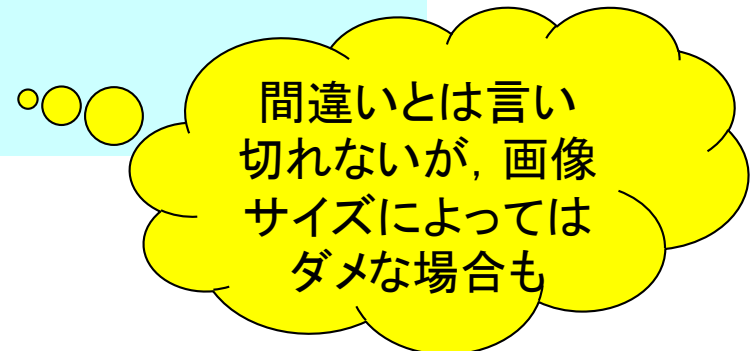
```
if (x1 == x2)
{
    m = (double) (y2 - y1);
}
else
{
    m = (double) (y2 - y1) / (x2 - x1);
}
```



```
double m;
if (x2 == x1)
    m = 3 * (y2 - y1);
else
    m = (double) (y2 - y1) / (x2 - x1);
```



```
if (x2 != x1)
    m = (double) (y2 - y1) / (x2 - x1);
else
    m = 100.0;
```



発展課題6

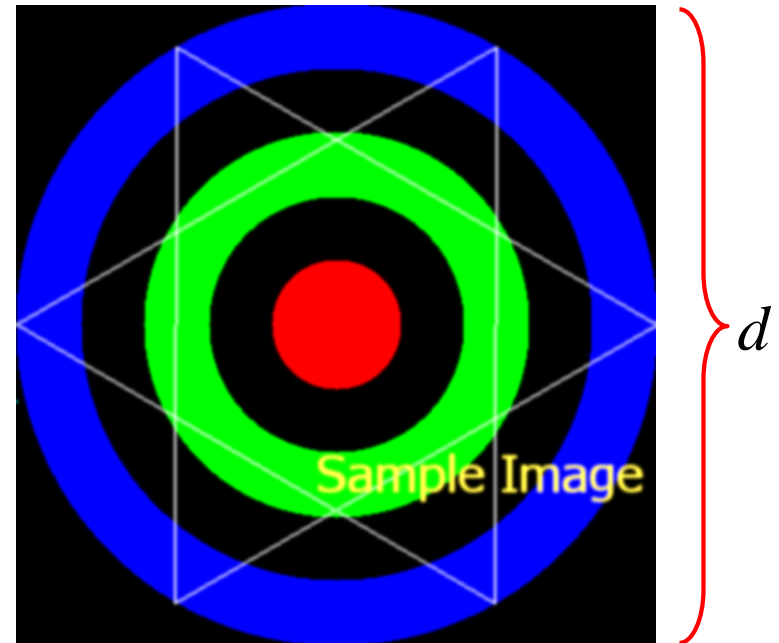
復習

DrawLine()関数とPaintCircle()関数を用いて、次のような画像を作り出すプログラムを作成せよ。

注) PaintCircle()関数の代わりにcglecに組み込まれているCglPaintCircle()を用いても良い

画像の縦と横方向ピクセル数は？ 400
続行するには何かキーを押してください . . .

- 画像の縦と横のピクセル数は同じ
- 画像の一边 d として,
 - 青円の外径は d , 内径は $4d/5$
 - 緑円の外径は $3d/5$, 内径は $2d/5$
 - 赤円の直径は $d/5$
- 星型六角形(六芒星)の半径は $d/2$ で, 線の色は白色.
- 「Sample Image」の文字は不要



発展課題6 解答例

F君解答

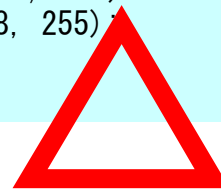
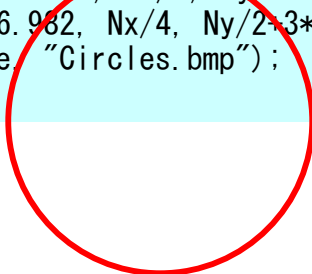
```
PaintCircle(blue, Nx/2, Ny/2, N/2, 255);  
PaintCircle(blue, Nx/2, Ny/2, 2*N/5, 0);  
PaintCircle(green, Nx/2, Ny/2, 3*N/10, 255);  
PaintCircle(green, Nx/2, Ny/2, N/5, 0);  
PaintCircle(red, Nx/2, Ny/2, N/10, 255);
```

青いリングを描くために、まず青プレーンに直径 $N/2$ で明度255の円を描き、次に直径 $2*N/5$ で明度0(黒)の円を描く

```
DrawLine(red, 0, Ny/2, 3*Nx/4, Ny/2+3*Ny/6.928, 255);  
DrawLine(green, 0, Ny/2, 3*Nx/4, Ny/2+3*Ny/6.928, 255);  
DrawLine(blue, 0, Ny/2, 3*Nx/4, Ny/2+3*Ny/6.928, 255);  
DrawLine(red, 3*Nx/4, Ny/2+3*Ny/6.928, 3*Nx/4, Ny/2-3*Ny/6.982, 255);  
DrawLine(green, 3*Nx/4, Ny/2+3*Ny/6.928, 3*Nx/4, Ny/2-3*Ny/6.982, 255);  
DrawLine(blue, 3*Nx/4, Ny/2+3*Ny/6.928, 3*Nx/4, Ny/2-3*Ny/6.982, 255);  
DrawLine(red, 3*Nx/4, Ny/2-3*Ny/6.982, 0, Ny/2, 255);  
DrawLine(green, 3*Nx/4, Ny/2-3*Ny/6.982, 0, Ny/2, 255);  
DrawLine(blue, 3*Nx/4, Ny/2-3*Ny/6.982, 0, Ny/2, 255);  
DrawLine(red, Nx/4, Ny/2+3*Ny/6.928, Nx, Ny/2, 255);  
DrawLine(green, Nx/4, Ny/2+3*Ny/6.928, Nx, Ny/2, 255);  
DrawLine(blue, Nx/4, Ny/2+3*Ny/6.928, Nx, Ny/2, 255);  
DrawLine(red, Nx, Ny/2, Nx/4, Ny/2-3*Ny/6.982, 255);  
DrawLine(green, Nx, Ny/2, Nx/4, Ny/2-3*Ny/6.982, 255);  
DrawLine(blue, Nx, Ny/2, Nx/4, Ny/2-3*Ny/6.982, 255);  
DrawLine(red, Nx/4, Ny/2-3*Ny/6.982, Nx/4, Ny/2+3*Ny/6.928, 255);  
DrawLine(green, Nx/4, Ny/2-3*Ny/6.982, Nx/4, Ny/2+3*Ny/6.928, 255);  
DrawLine(blue, Nx/4, Ny/2-3*Ny/6.982, Nx/4, Ny/2+3*Ny/6.928, 255);  
GglSaveColorBMP(red, green, blue, "Circles.bmp");
```

白線を描くために、全プレーンに同じ明度255の線を描く

力技タイプ



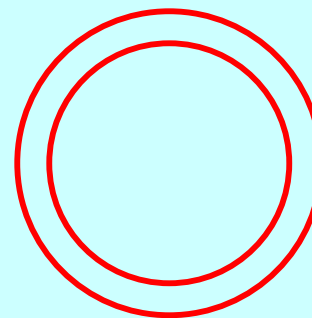
発展課題6 解答例

```
unsigned char* red = (unsigned char*)malloc(sizeof(unsigned char) * Nx * Nx);
unsigned char* gre = (unsigned char*)malloc(sizeof(unsigned char) * Nx * Nx);
unsigned char* blu = (unsigned char*)malloc(sizeof(unsigned char) * Nx * Nx);
if ((red == NULL) || (blu == NULL) || (gre == NULL))
{
    printf("メモリエラー!!");
    exit(0);
}
Image data[3]={{ (unsigned char*) red, Nx, Nx }, { (unsigned char*) gre, Nx, Nx }, { (unsigned char*)
blu, Nx, Nx }};
int i;
for(i=0;i<3;i++)
{
    CglSetAll(data[i], 0); // imgをグレイレベルでクリアする
}
CglPaintCircle(data[2], Nx/2, Nx/2, Nx/2, 255);
CglPaintCircle(data[2], Nx/2, Nx/2, Nx*2/5, 0);
CglPaintCircle(data[1], Nx/2, Nx/2, Nx*3/10, 255);
CglPaintCircle(data[1], Nx/2, Nx/2, Nx*1/5, 0);
CglPaintCircle(data[0], Nx/2, Nx/2, Nx*1/10, 255);

int Lx=(int)((Nx*3)/4+0.5);
int Ly=(int)(((float)(Nx*sqrt(3.0)))/4+0.5);

for(i=0;i<3;i++)
{
    DrawLine(data[i], 0, Nx/2, Lx, (Nx/2)+Ly, 255);
    DrawLine(data[i], 0, Nx/2, Lx, (Nx/2)-Ly, 255);
    DrawLine(data[i], Lx, (Nx/2)+Ly, Lx, (Nx/2)-Ly, 255);
    DrawLine(data[i], Nx-1, Nx/2, Nx-Lx, (Nx/2)+Ly, 255);
    DrawLine(data[i], Nx-1, Nx/2, Nx-Lx, (Nx/2)-Ly, 255);
    DrawLine(data[i], Nx-Lx, (Nx/2)+Ly, Nx-Lx, (Nx/2)-Ly, 255);
}

CglSaveColorBMP(data[0], data[1], data[2], "Drawing. bmp");
```

ループタイプ &
技巧派

発展課題6 解答例

関数派

H君解答

```
void star(Image img, int d)
{
    DrawLine( img, 0, d/2, (3*d)/4, ((d * 1.732)/4)+d/2, 255);
    DrawLine( img, 0, d/2, (3*d)/4, d/2-((d * 1.732)/4), 255);
    DrawLine( img, (3*d)/4, ((d * 1.732)/4)+d/2, (3*d)/4, d/2-((d * 1.732)/4), 255);

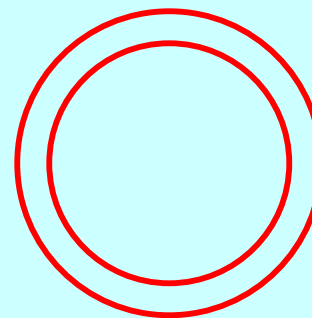
    DrawLine( img, d-1, d/2, d/4, ((d * 1.732)/4)+d/2, 255);
    DrawLine( img, d-1, d/2, d/4, d/2-((d * 1.732)/4), 255);
    DrawLine( img, d/4, ((d * 1.732)/4)+d/2, d/4, d/2-((d * 1.732)/4), 255);
}
```

```
CglPaintCircle(img_blue, d/2, d/2, d/2, 255); //青の外径
CglPaintCircle(img_blue, d/2, d/2, 4*d/10, 0); //青の内径
star( img_blue, d );
```

```
CglPaintCircle(img_green, d/2, d/2, 3*d/10, 255); //緑の外径
CglPaintCircle(img_green, d/2, d/2, 2*d/10, 0); //緑の内径
star( img_green, d );
```

```
CglPaintCircle(img_red, d/2, d/2, d/10, 255); //赤
star( img_red, d );
```

```
CglSaveColorBMP(img_red, img_green, img_blue, "Star_Circles.bmp");
```



線図形の描画

Example7-1

```
struct Point
{
    int x;
    int y;
};
```

点の座標を表す構造体




CglDrawLine()はcglecに定義されている。基本課題6のDrawLine()と全く同じ機能を持った関数

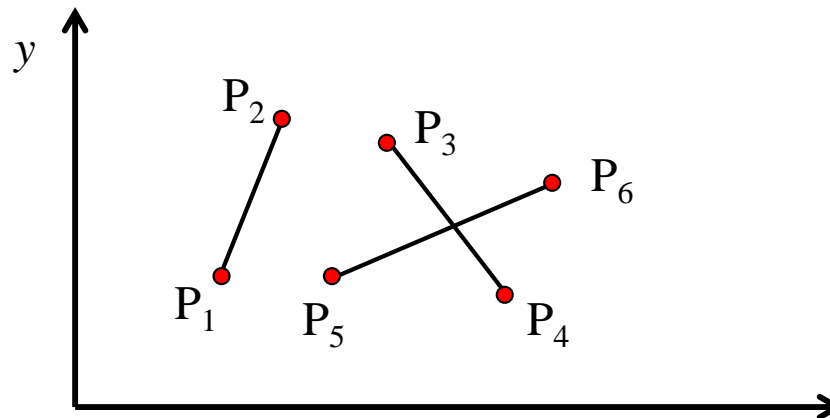
```
void CglDrawLine(Image img, int x1, int y1, int x2, int y2, int g);
```

```
void DrawLines(Image img, Point p[], int n, int g)
```

```
{
    int i;
    if (n <= 1 || n%2 != 0) return; //2点以上の偶数でなければ描けない
    for (i = 0; i < n; i += 2)
    {
        CglDrawLine(img, p[i].x, p[i].y, p[i + 1].x, p[i + 1].y, g);
    }
}
```

Point構造体の配列

画像に、 n 個の点列からなるグレーレベルの線分を描画する。



線図形の描画

Example7-2

```
// struct Point, CglDrawLine() はcglec.hで定義されている.  
// DrawLines() はここより前で定義または宣言されているものとする.
```

```
#define WIDTH 300  
#define HEIGHT 300
```

```
int main(void)  
{
```

```
    unsigned char data[WIDTH][HEIGHT];
```

```
    Image img = { (unsigned char*) data, WIDTH, HEIGHT };
```

```
    CglSetAll(img, 0);
```

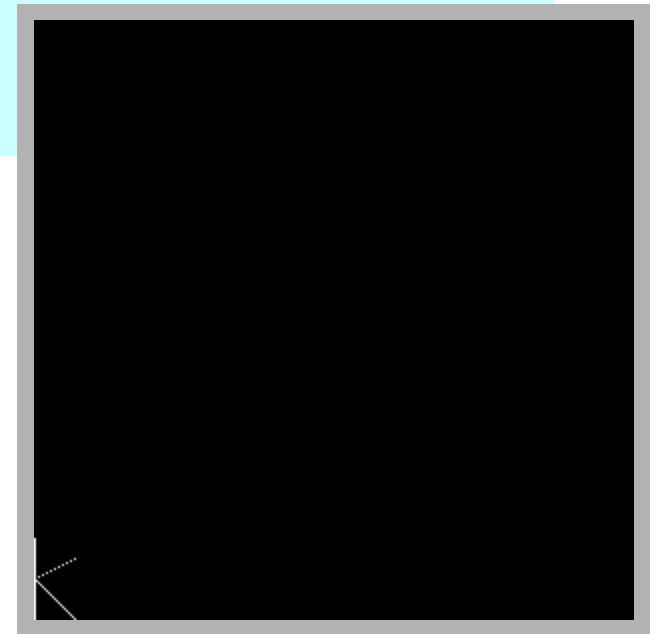
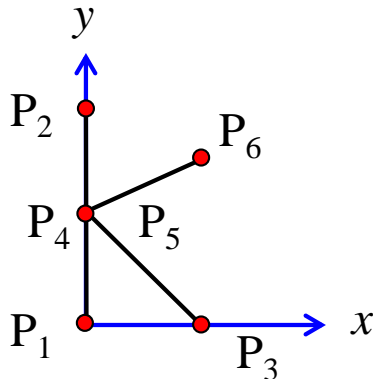
```
    Point moji_k[] = {{0, 0}, {0, 40}, {20, 0}, {0, 20}, {0, 20}, {20, 30}};  
    int N = 6;
```

```
    DrawLines(img, moji_k, N, 255);
```

```
    CglSaveGrayBMP(img, "moji_k.bmp");
```

```
}
```

図形データ
(モデル)



図形の拡大

全点の座標をM倍
に拡大する関数

座標位置をM倍に拡大する

$$x' = Mx$$

$$y' = My$$

Example7-3

```
void Kakudai(Point p[], int n, double M)
{
    int i;
    for (i = 0; i < n; i++)
    {
        p[i].x = (int) (p[i].x * M + 0.5);
        p[i].y = (int) (p[i].y * M + 0.5);
    }
}
```

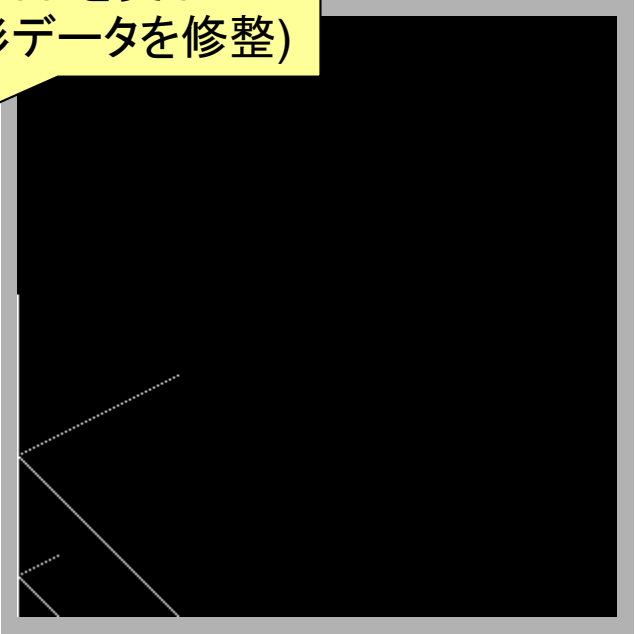
main()関数を少し変更

```
DrawLines(img, moji_k, N, 255); //元の図形を描画
```

```
Kakudai(moji_k, N, 4); //4倍に拡大
DrawLines(img, moji_k, N, 255); //拡大図形を描画
```

図形を変形
(図形データを修整)

注) 関数内で引数に代入しても、通常は呼び出し側の変数値に影響を与えることはできない(「値渡し」, 基礎プロ講習11回). しかし、引数が配列の場合には呼び出し側の配列を変化できる(「参照渡し」の機能).



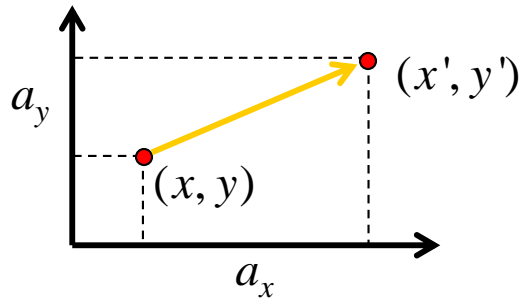
図形の移動

全点を x 方向に ax ,
 y 方向に ay 移動する関数

座標位置を, x 方向に a_x ,
 y 方向に a_y 移動する

$$x' = x + a_x$$

$$y' = y + a_y$$



Example7-3

```
void Idou(Point p[], int n, int ax, int ay)
{
    int i;
    for (i = 0; i < n; i++)
    {
        p[i].x = p[i].x + ax;
        p[i].y = p[i].y + ay;
    }
}
```

main()関数を少し変更

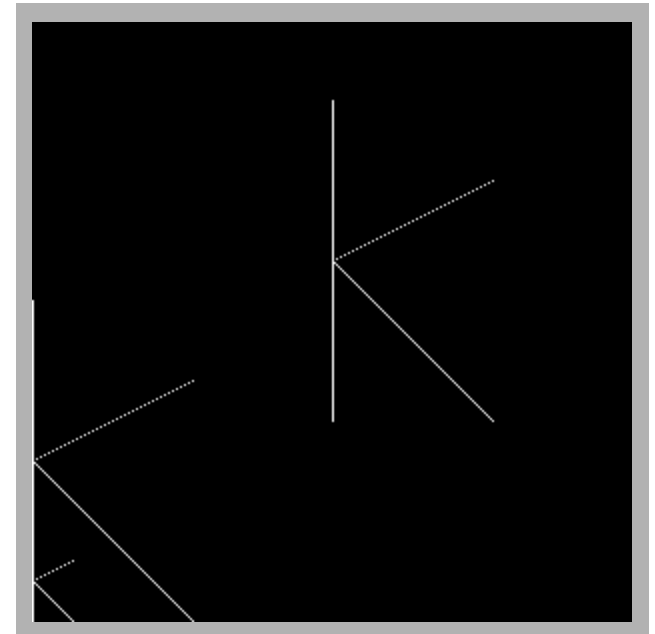
```
DrawLines(img, moji_k, N, 255); //元の図形を描画
```

```
Kakudai(moji_k, N, 4); //4倍に拡大
```

```
DrawLines(img, moji_k, N, 255); //拡大図形を描画
```

```
Idou(moji_k, N, 150, 100); //移動
```

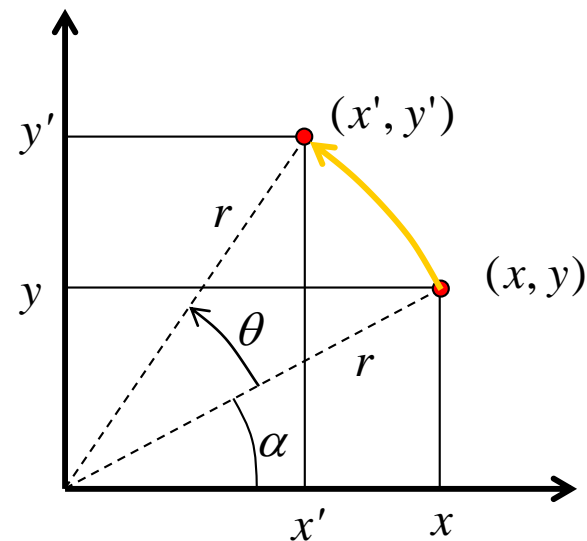
```
DrawLines(img, moji_k, N, 255); //移動図形を描画
```



図形の回転

座標位置を角度 θ 回転する

$$\begin{aligned}x' &= r \cos(\alpha + \theta) \\&= r \cos \alpha \cos \theta - r \sin \alpha \sin \theta \\&= x \cos \theta - y \sin \theta \\y' &= r \sin(\alpha + \theta) \\&= r \sin \alpha \cos \theta + r \cos \alpha \sin \theta \\&= y \cos \theta + x \sin \theta\end{aligned}$$



```
void Kaiten(Point p[], int n, double q)
```

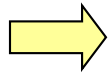
全点を原点の周りに
角度 q [度]
回転する関数

2次元図形の幾何変換

図形を M 倍に拡大する

$$x' = Mx$$

$$y' = My$$



図形を x 方向に S_x 倍, y 方向に S_y 倍拡大する

$$x' = S_x x$$

$$y' = S_y y$$

$$x' = S_x x + 0 \times y + 0$$

$$y' = 0 \times x + S_y y + 0$$

図形を x 方向に t_x , y 方向に t_y 移動する

$$x' = x + t_x$$

$$y' = y + t_y$$

$$x' = 1 \times x + 0 \times y + t_x$$

$$y' = 0 \times x + 1 \times y + t_y$$

$$x' = ax + by + c$$

$$y' = dx + ey + f$$

アフィン変換

図形を角度 θ 回転する(反時計回り)

$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$


$$x' = \cos \theta \times x - \sin \theta \times y + 0$$


$$y' = \sin \theta \times x + \cos \theta \times y + 0$$

アフィン変換の行列表示

$$x' = ax + by + c$$

$$y' = dx + ey + f$$


$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix}$$


$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

アフィン変換
の係数が一つ
にまとまる！

同次座標系

変換前の
座標

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

$$\mathbf{v}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}, \mathbf{M} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

変換後の座標

拡大/縮小(スケーリング)

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

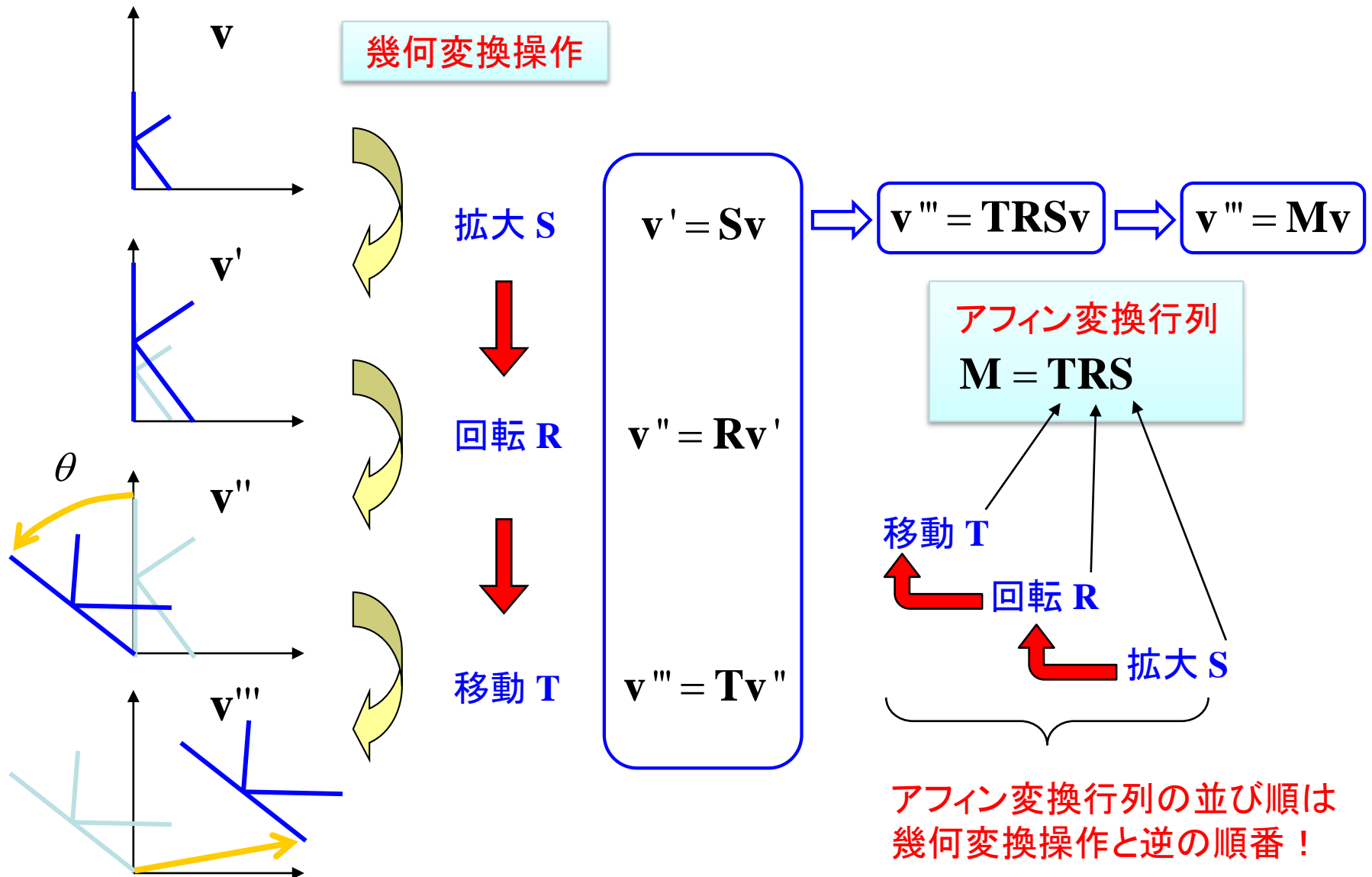
移動

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

回転(反時計回り)

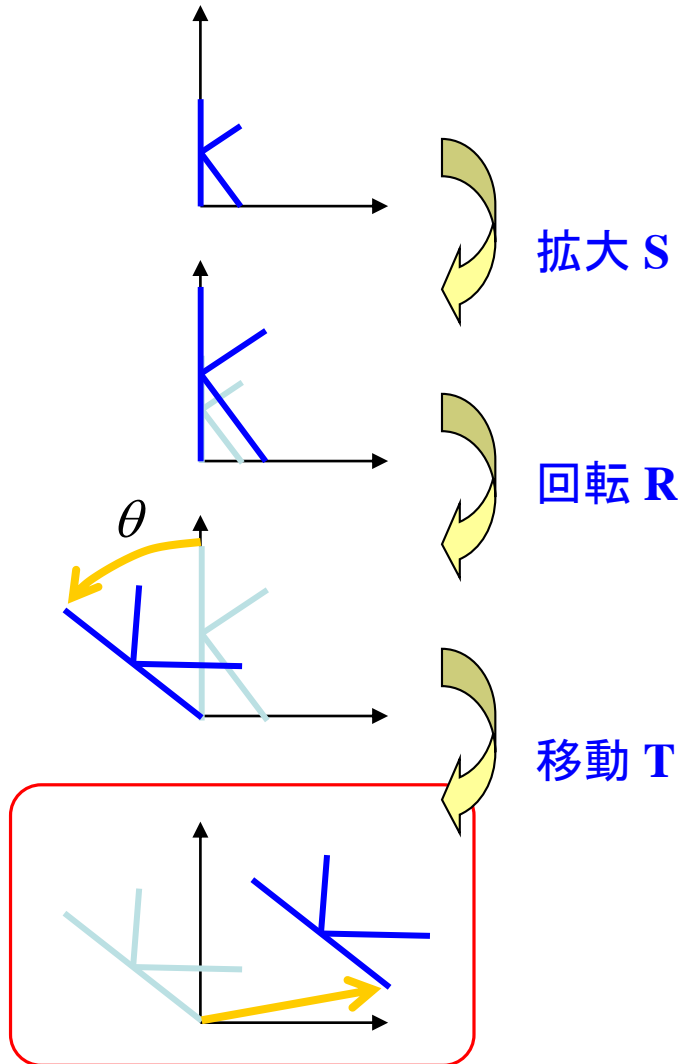
$$\mathbf{R} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

幾何変換の組合せとアフィン変換行列

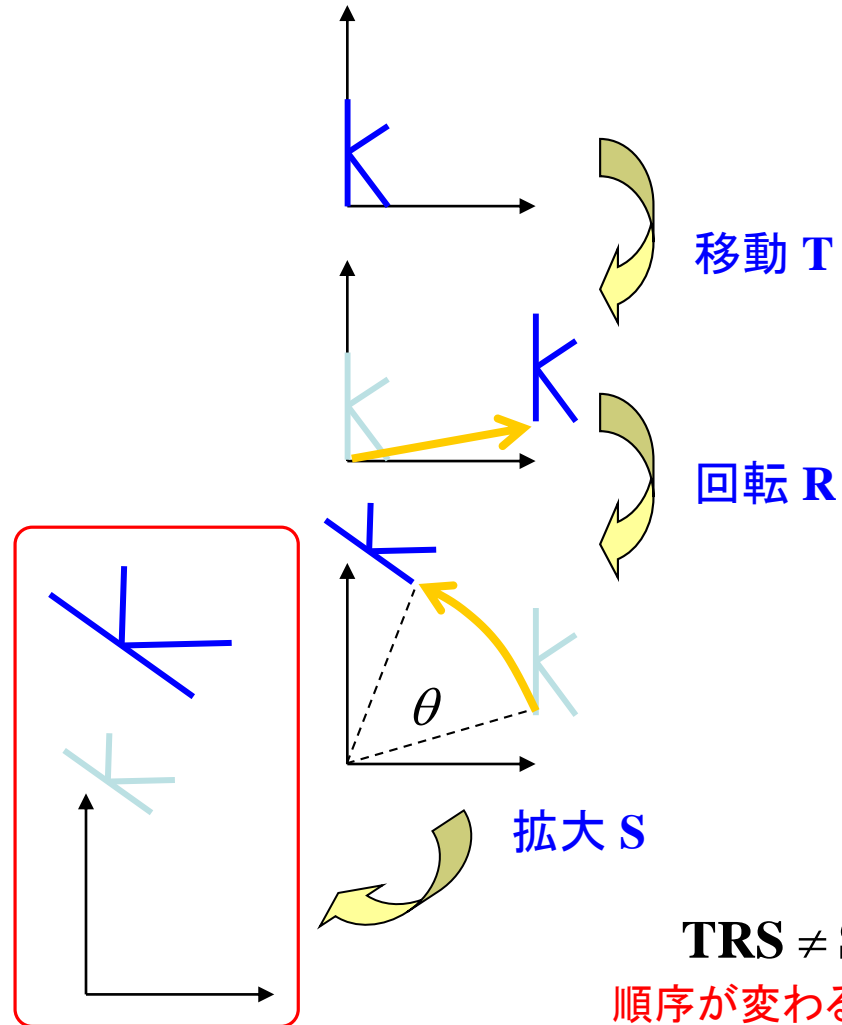


幾何変換の順序

変換行列 $M = \mathbf{TRS}$



変換行列 $M = \mathbf{SRT}$



$\mathbf{TRS} \neq \mathbf{SRT}$

順序が変わると異なった変換になることがある

アフィン変換の注意

書籍によっては次のような定義を用いてる場合もある.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \Rightarrow \quad [x' \quad y' \quad 1] = [x \quad y \quad 1] \begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{bmatrix}$$

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

$$\mathbf{v}' = [x' \quad y' \quad 1], \quad \mathbf{M} = \begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{bmatrix}, \quad \mathbf{v} = [x \quad y \quad 1]$$

この場合, 変換行列は元の行列の転置行列となる

基本課題7

Report7-1

全点を原点の周りに角度 q [度]だけ回転する関数を作成し、以下のmain()関数を用いて実行せよ。

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cglec.h"
```

```
// struct Point, CglDrawLine()は,"cglec.h"で定義されている
// DrawLines(), Kakudai(), Idou()は、これ以前の行で定義または宣言しておくこと
```

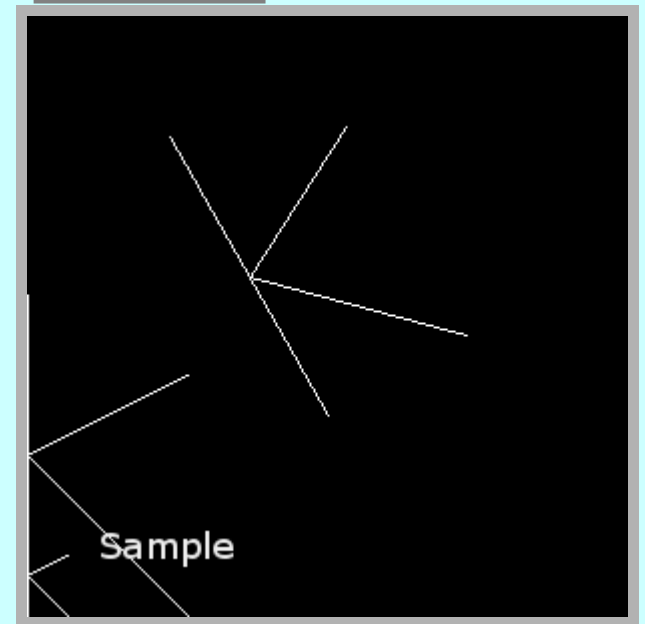
```
void Kaiten(Point point[], int n, double q)
{ /* この部分をプログラミング */ }
```

```
#define WIDTH 300
#define HEIGHT 300
int main(void)
{
    unsigned char data[WIDTH][HEIGHT];
    Image img = { (unsigned char*) data, WIDTH, HEIGHT };
    CglSetAll(img, 0);
    Point moji_k[] = {{0, 0}, {0, 40}, {20, 0},
                     {0, 20}, {0, 20}, {20, 30}};

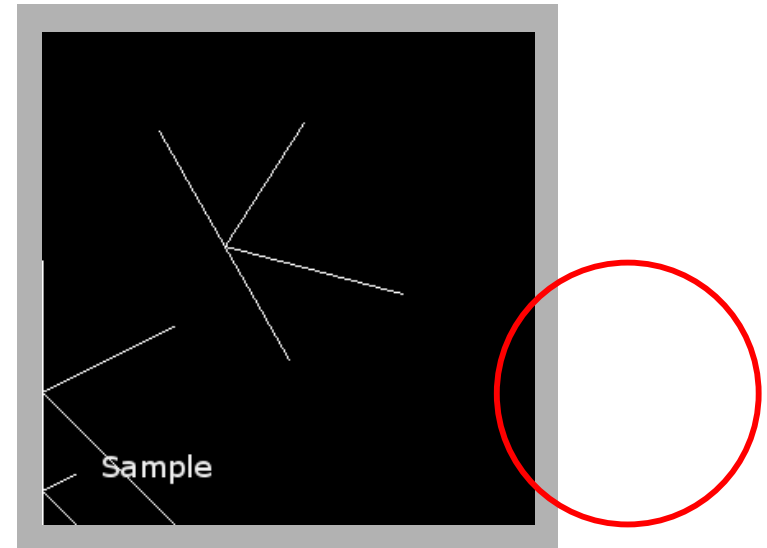
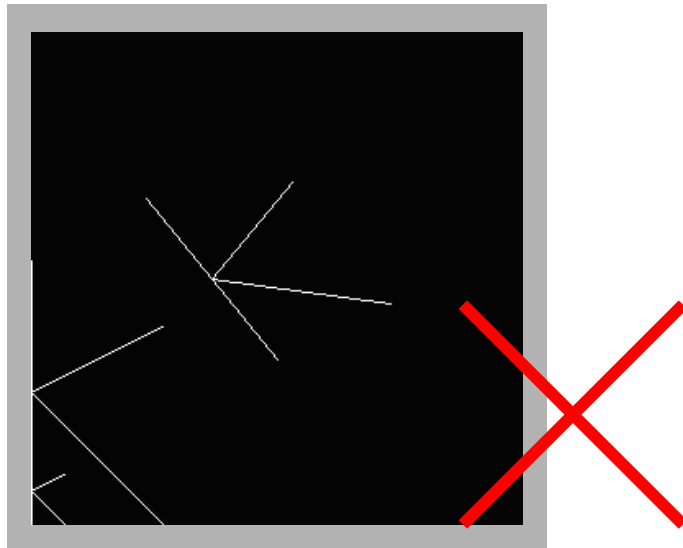
    int N = 6;
    DrawLines(img, moji_k, N, 255);
    Kakudai(moji_k, N, 4);
    DrawLines(img, moji_k, N, 255);
    Kaiten(moji_k, N, 30);
    Idou(moji_k, N, 150, 100);
    DrawLines(img, moji_k, N, 255);
    CglSaveGrayBMP(img, "moji_k.bmp");
}
```

ソースと画像を提出

実行結果



基本課題7の正解と不正解



今回は発展課題はありません