

授業のおおまかな予定

4/05	ガイダンス	2次元グラフィックス 四角形, 円形, 直線の描画 カラー画像 座標変換
4/12		
4/19		
4/26		
5/10		
5/17		

5/24 休講
5/27(月) 6限 補講(OD2)

5/31 (3201教室?) ← 中間試験(小テスト)

6/07	3次元グラフィックス 物体形状の表現 シェーディング 隠面消去 OpenGL CG作品
6/14	
6/21	
6/28 休講	
7/01(月) 6限 補講(OD2)	
7/05 休講	
7/12	
7/19	

試験期間中 ← 期末試験

レポート	10%
出席	10%
修了作品	20%
中間試験	20%
期末試験	40%

レポート
≡ C言語プログラム + α

受講するためには
→ 「基礎プログラミング」程度
のC言語の知識が必須

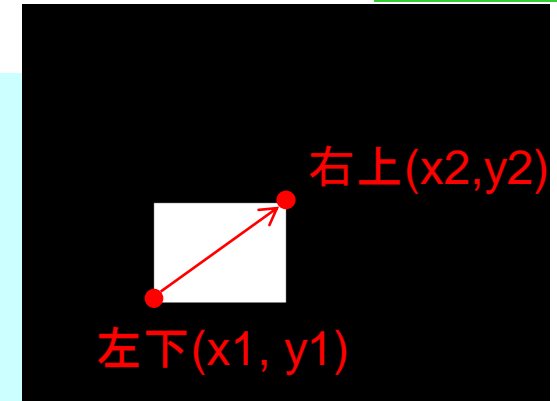
関数を作ろう！

Example4-1

```
#include <stdio.h>
#include <stdlib.h>
#include "cglec.h"
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;
    data = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    if (data == NULL) //メモリ割当に失敗か?
    {
        printf("メモリエラー!!");
        exit(0); //プログラムを終了する
    }

    Image img = { (unsigned char*) data, Nx, Ny };
    GglSetAll(img, 0);
    int x, y;
    for (y = Ny/4; y < Ny/2; y++)
        for (x = Nx/4; x < Nx/2; x++)
        {
            *(data + x*Ny + y) = 255;
        }
    GglSaveGrayBMP(img, "Rei3-1.bmp");
    free(data); //メモリ解放
}
```



四角形の描画はよく利用するので、関数にしておこう！

左下の点座標

右上の点座標

PaintRect(img, x1, y1, x2, y2, 255);

Image型構造体変数
↓
描画する「キャンバス」
を指定する

四角形描画関数

Example4-2

```
void PaintRect(Image img, int x1, int y1, int x2, int y2, int level)
```

```
{
    int temp;
    if (x1 > x2)    // x1 < x2 になるように入れ替え
    {
        temp = x1;  x1 = x2;    x2 = temp;
    }
    if (y1 > y2)    // y1 < y2 になるように入れ替え
    {
        temp = y1;  y1 = y2;    y2 = temp;
    }
}
```

// パラメータが範囲外の場合は修正


```
if (x1 < 0)        x1 = 0;
if (y1 < 0)        y1 = 0;
if (x2 > img.Nx)   x2 = img.Nx;
if (y2 > img.Ny)   y2 = img.Ny;
if (level < 0)     level = 0;
if (level > 255)   level = 255;
```

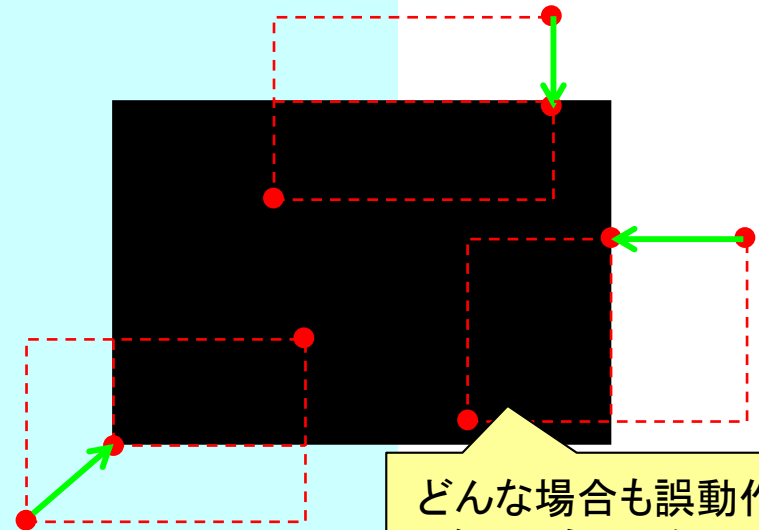
// 四角形描画

```
int x, y;
for (y = y1; y < y2; y++)
    for (x = x1; x < x2; x++)
    {
        *(img.Data + x*img.Ny + y) = level;
    }
}
```

構造体変数imgの画像に対する描画なので、画像の幅や高さはimgのメンバー変数を用いなければならない

機能

画像において、二点(x1, y1), (x2, y2)を対角点とする四角形をグレースケールlevelで塗りつぶす



どんな場合も誤動作しないようにしたい！

```
struct Image
{
    unsigned char* Data; // データ領域へのポインタ
    int Nx;              // 横方向ピクセル数
    int Ny;              // 縦方向ピクセル数
};
```

基本課題4

復習

中心座標(x_0, y_0)と半径 r , グレーレベル g を引数として, 円を塗りつぶす関数を作成せよ.
この時, 引数が適切な範囲で無い場合も正常に動作すること. 作成した関数と下記の
main()を組み合わせて実行例と同じ画像を得よ.

Report4-1

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

void PaintCircle(Image img, int x0, int y0, int r, int g)
{
    // この関数を作成する.  x0, y0 : 中心座標,  r : 半径,  g : グレーレベル
}

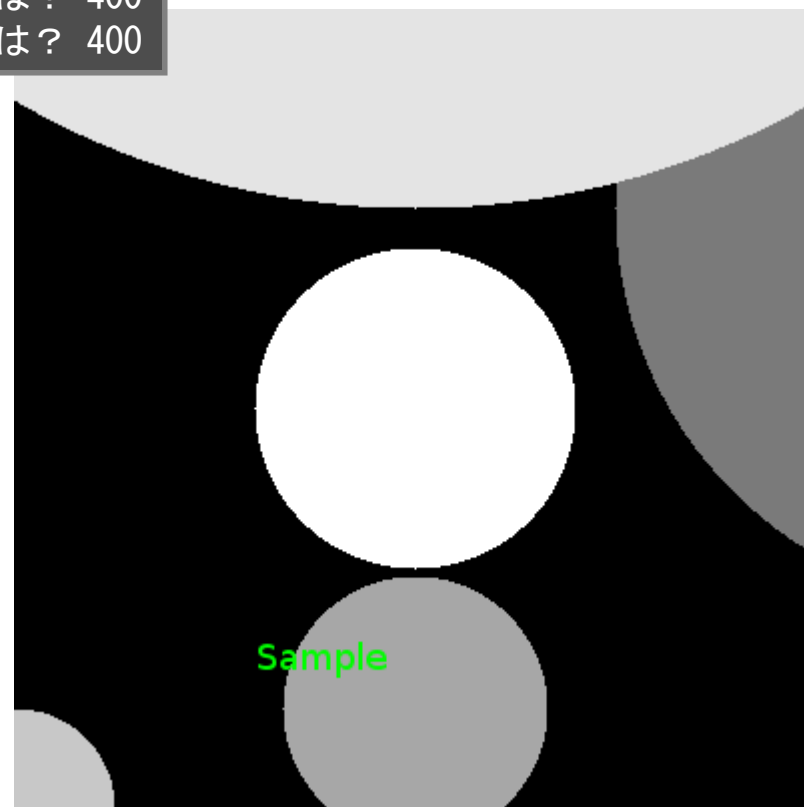
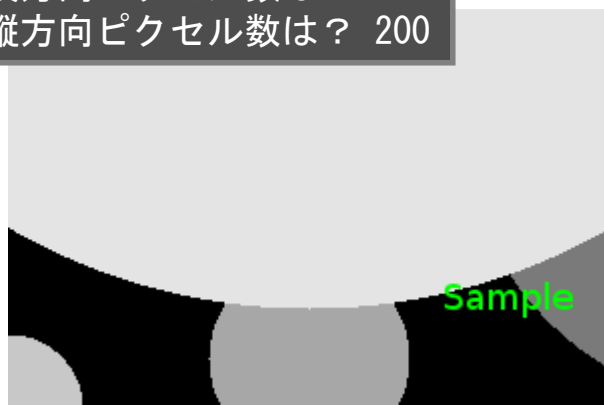
int main(void)    // main() 関数はこのまま使う. 修正しないこと !
{
    int Nx, Ny;
    printf("画像の横方向ピクセル数は? "); scanf("%d", &Nx);
    printf("画像の縦方向ピクセル数は? "); scanf("%d", &Ny);
    unsigned char* data = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    if (data == NULL) // メモリ割当成功か?
    {
        printf("メモリエラー!!"); exit(0); } // メモリ割当に失敗したらプログラムを終了する
    Image img = { (unsigned char*) data, Nx, Ny };
    CglSetAll(img, 0);
    PaintCircle(img, Nx/2, Ny/2, Nx/5, 255);
    PaintCircle(img, 0, 0, Nx/8, 150);
    PaintCircle(img, Nx/2, Ny/8, Nx/6, 100);
    PaintCircle(img, 5*Nx/4, 3*Ny/4, Nx/2, 50);
    PaintCircle(img, Nx/2, 7*Ny/4, Nx, 200);
    CglSaveGrayBMP(img, "Circles.bmp");
    free(data); // メモリ解放
}
```

基本課題4の実行例

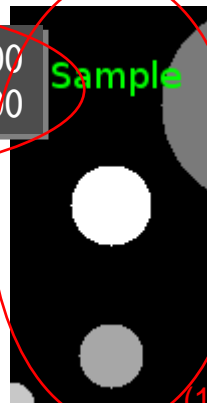
注) 文字「Sample」は描画不要

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？ 400

画像の横方向ピクセル数は？ 300
画像の縦方向ピクセル数は？ 200



画像の横方向ピクセル数は？ 100
画像の縦方向ピクセル数は？ 200



(2) 実行結果 (画面コピー)

(1) 実行結果(BMPファイル)

- ・ ソースを提出
- ・ (1)と(2)の両方を提出！
- ・ 実行例を2種類以上提出！

発展課題3 ヒント

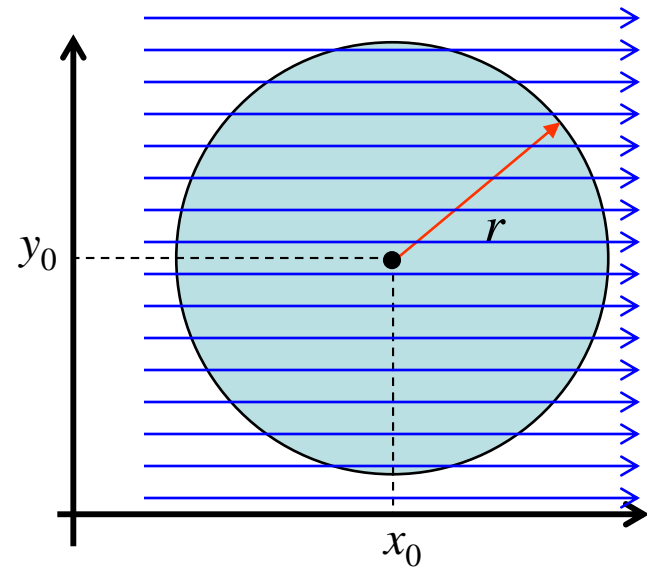
(x_0, y_0) を中心とする半径 r の円の方程式は

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

である. 従って, この円の内側の領域は

$$(x - x_0)^2 + (y - y_0)^2 \leq r^2$$

で表される.



ラスタースキャン

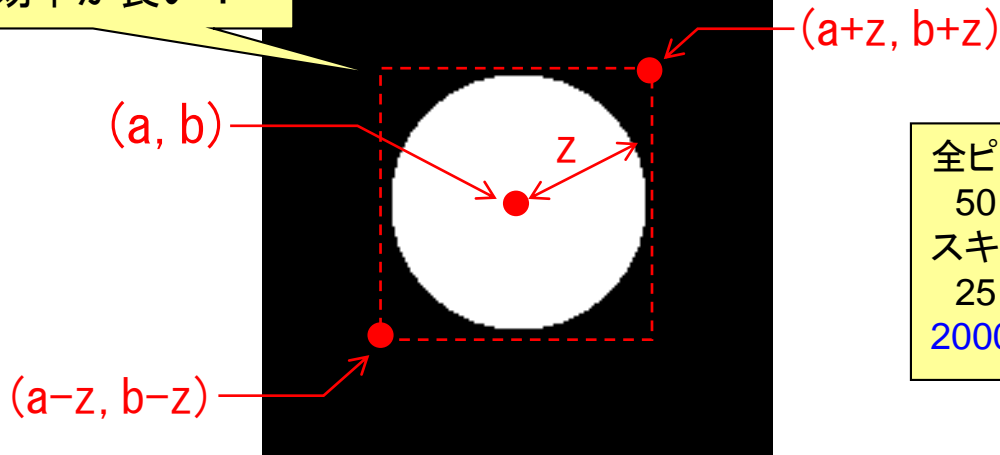
発展課題3 解答例

E君解答

```
CglSetAll(img, 0);
int a=Nx/2, b=Ny/2, x, y, z;
if (Nx<Ny)
    z=a/2;
else
    z=b/2;
for (x=a-z; x<a+z; x++)
    {
        for (y=b-z; y<b+z; y++)
        {
            if ((x-a)*(x-a)+(y-b)*(y-b)<=z*z)
                *(data + x*Ny+ y) = 255;
        }
    }
CglSaveGrayBMP(img, "Rei3-1.bmp");
```

효율적인 코드

良くできました!

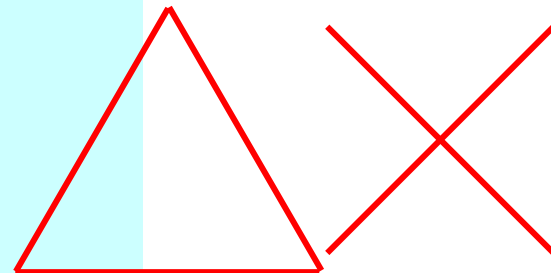
 この範囲だけをスキャン
するので効率が良い!


全ピクセル数
 $50 \times 400 = 20,000$
 スキャン範囲
 $25 \times 25 = 625$
 $20000/625 = 32$ 倍!

基本課題4 解答例

A君解答

```
void PaintCircle(Image img, int x0, int y0, int r, int g)
{
    int x, y;
    for(y = 0; y < img.Ny; y++)
    {
        for(x = 0; x < img.Nx; x++)
        {
            if( (x-x0)*(x-x0) + (y-y0)*(y-y0) <= r*r)
                *(img.Data + x*img.Ny + y) = g;
        }
    }
}
```



あまりにも無駄が多い

例えば

画像サイズ 400×200ピクセル
円の半径 5ピクセル

$$400 \times 200 \div (2 \times 5)^2 = 800$$

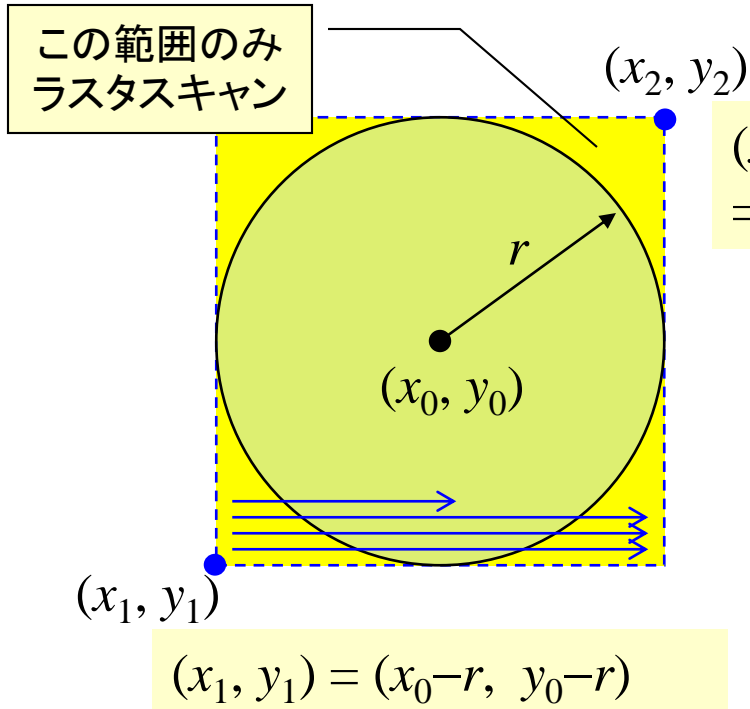
800倍程度遅いプログラム！

しかも関数は何度も呼び出される・・・



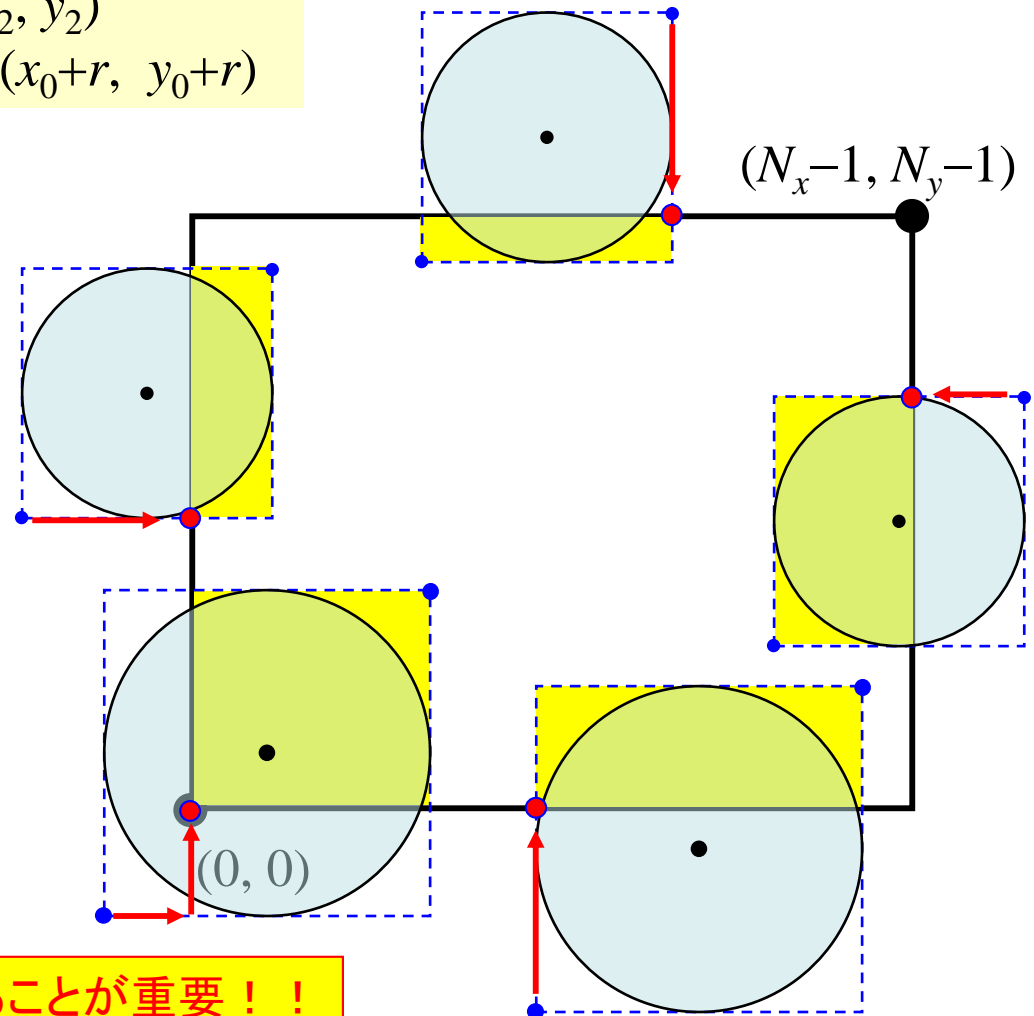
円塗りつぶし関数の考え方

범위를 지정하기
クリッピング
= 描画に必要な領域だけに
限定すること



$x_1 < 0$ なら,	$x_1 = 0$
$x_2 > N_x - 1$ なら,	$x_2 = N_x - 1$
$y_1 < 0$ なら,	$y_1 = 0$
$y_2 > N_y - 1$ なら,	$y_2 = N_y - 1$

プログラムを始める前に良く考えることが重要！！



危険な不正アクセス！

```
#define WIDTH 8 // 画像の幅(ピクセル数)
#define HEIGHT 6 // 画像の高さ(ピクセル数)
配列 image[WIDTH][HEIGHT]
```

i = WIDTH ↓

	i=0	1	2	3	4	5	6	7
j=0	1	0	0	0	0	0	0	0
j=1	0	1	0	0	0	0	0	0
j=2	0	0	1	0	0	0	0	0
j=3	0	0	0	1	0	0	0	0
j=4	0	0	0	0	1	0	0	0
j=5	0	0	0	0	0	1	0	0

j = HEIGHT →

存在しない配列要素！
確保していないメモリ！

不正ア
クセス

運が良い時 ⇒ 何も問題は起きない
(ただし正解と一致したのは偶然)

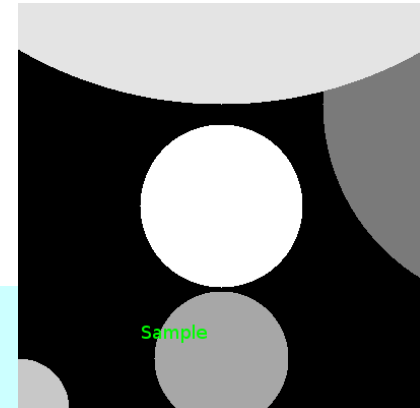
問題が起きるときの症状

- ✓ 途中でプログラムが停止しているが、一見正常終了
- ✓ 不正アクセス等のエラーメッセージで終了
- ✓ 突然、コンソールウィンドウ(黒ウィンドウ)が閉じる

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

```
void PaintCircle(Image img, int x0, int y0, int r,
{
    // この関数を作成する. x0,y0 : 中心座標, r :
```

```
int main(void) // main()関数はこのまま使う. 修正
{
    int Nx, Ny;
    printf("画像の横方向ピクセル数は? "); scanf("
    printf("画像の縦方向ピクセル数は? "); scanf("
    unsigned char* data = (unsigned char*) malloc(s
    if (data == NULL) //
    { printf("メモリエラー!!"); exit(0); } //
    Image img = { (unsigned char*) data, Nx, Ny };
    CglSetAll(img, 0);
    PaintCircle(img, Nx/2, Ny/2, Nx/5, 255);
    PaintCircle(img, 0, 0, Nx/8, 150);
    PaintCircle(img, Nx/2, Ny/8, Nx/6, 100);
    PaintCircle(img, 5*Nx/4, 3*Ny/4, Nx/2, 50);
    PaintCircle(img, Nx/2, 7*Ny/4, Nx, 200);
    CglSaveGrayBMP(img, "Circles.bmp");
    free(data); //メモリ解放
}
```



円塗りつぶし関数

B君解答

```
void PaintCircle(Image img, int x0, int y0, int r, int g)
{
    int x, y, xs = x0 - r, ys = y0 - r, xe = x0 + r, ye = y0 + r;

    if (xs < 0)        xs = 0;
    if (ys < 0)        ys = 0;
    if (xe > img.Nx)    xe = img.Nx;
    if (ye > img.Ny)    ye = img.Ny;
    if (g < 0)         g = 0;
    if (g > 255)        g = 255;

    for (y = ys; y < ye; y++)
    {
        for (x = xs; x < xe; x++)
        {
            if ((x - x0)*(x - x0) + (y - y0)*(y - y0) <= r*r)
            {
                *(img.Data + x*img.Ny + y) = g;
            }
        }
    }
}
```

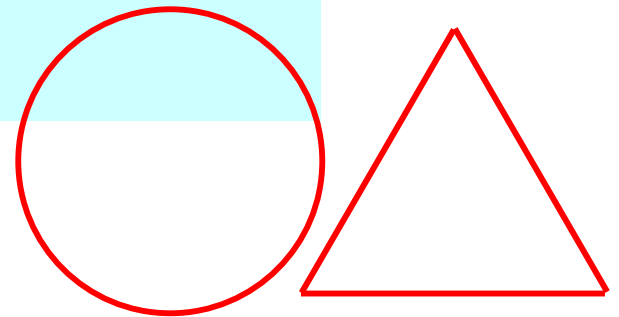
ここがないと正常実行できない。
※そのようなバグはデバッグですぐに発見できる

良くできました!

いろいろな解答

C君解答

```
void PaintCircle(Image img, int x0, int y0, int r, int g)
{
    int x, y;
    if (g<0)    g = 0;
    if (g>255)  g = 255;
    for (x = x0 - r; x<x0 + r; x++)
    {
        for (y = y0 - r; y<y0 + r; y++)
        {
            if ((x - x0)*(x - x0) + (y - y0)*(y - y0) <= r*r
                && 0<x&&img.Nx>x && 0<y&&img.Ny>y)
            {
                *(img.Data + x*img.Ny + y) = g;
            }
        }
    }
}
```



効率的なプログラムの条件

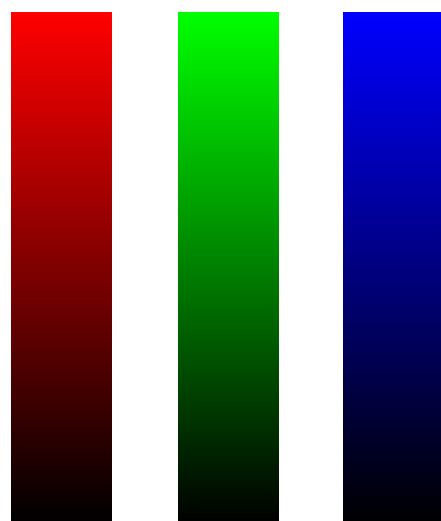
- 1) ループを実行する前にしっかりと事前処理する
パラメータチェック！
- 2) ループ内での処理をできるだけ簡単化する
条件判断(if文)がループに入るとスピードダウン！
- 3) ループの回数をできるだけ減らす
アルゴリズムの検討！

コンピュータにおけるカラー表現



光のスペクトル*

* Wikipediaからの引用



Red

Green

Blue

255

⋮

0

0, 1, ..., 255: 明度

256階調

RGB各色が8ビット階調の明度を持ち、その組み合わせでカラーを表現
→ 1ピクセルの表現に3バイト必要

(R, G, B)

= (0, 255, 0)

→ 緑

= (255, 255, 255)

→ 白

= (255, 255, 0)

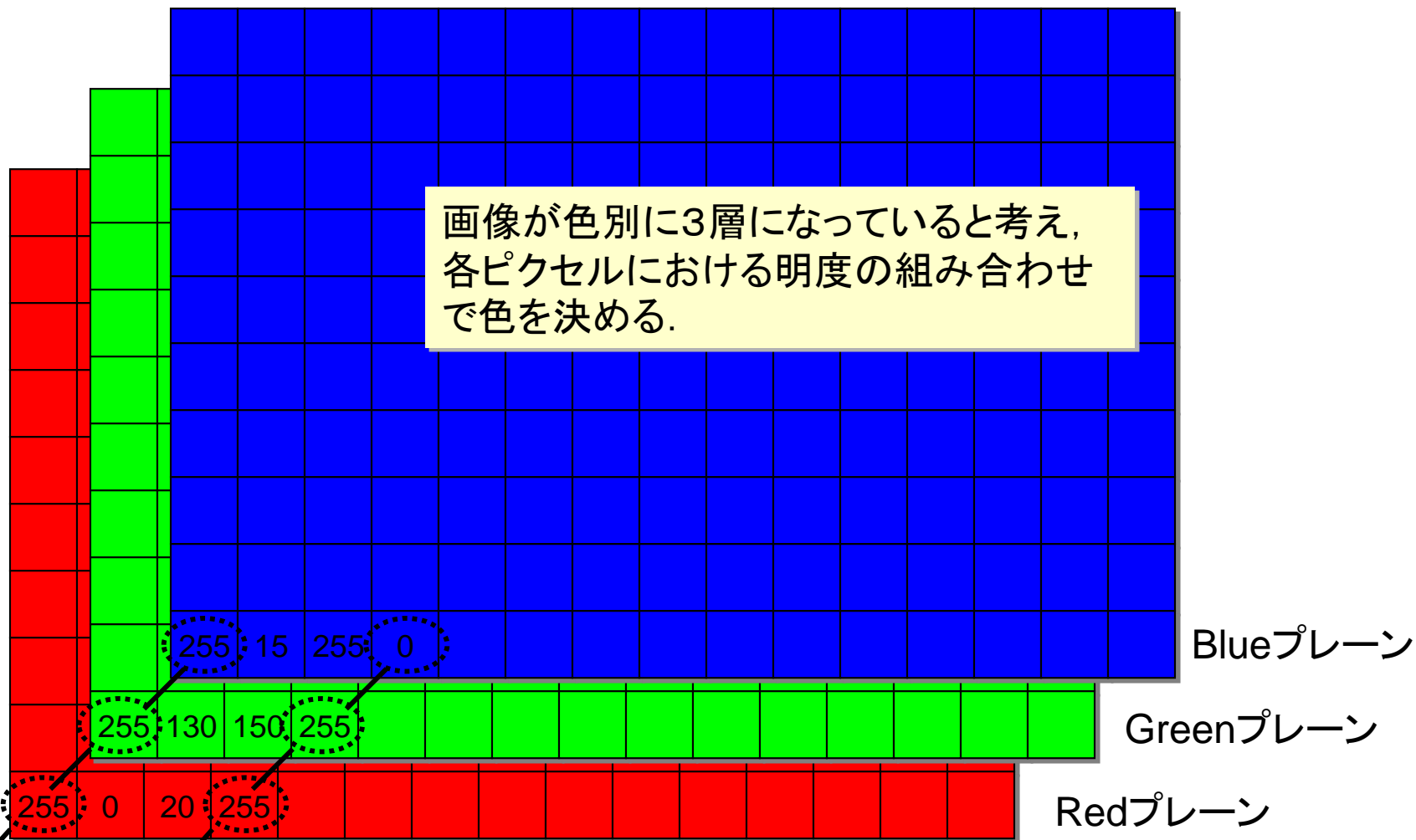
→ 黄

これ以外にも、コンピュータでカラー画像を現す方式は種々あるが、
ここでは深く立ち入らない

プログラムでのカラー画像の取り扱い

復習

画像が色別に3層になっていると考え、
各ピクセルにおける明度の組み合わせ
で色を決める。



(255, 255, 0) → 黄色

(255, 255, 255) → 白色

カラー画像作成プログラムのソース例

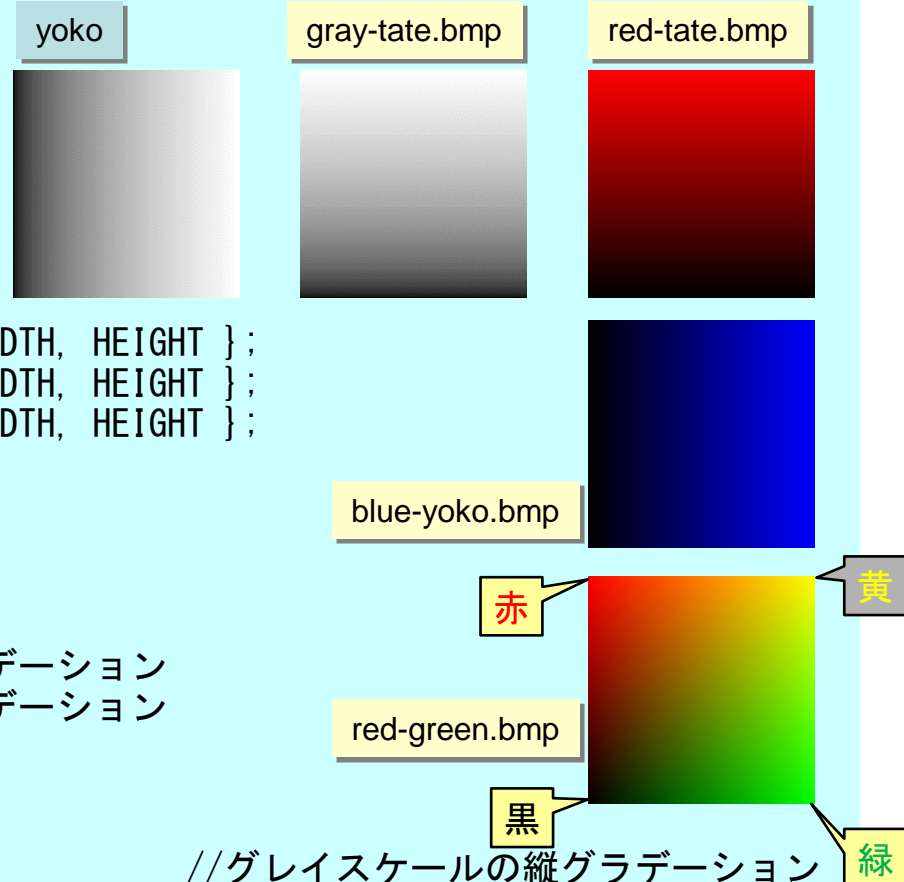
Example4-3

```
#include "cglec.h"
#define WIDTH 256
#define HEIGHT 256

int main(void)
{
    unsigned char tate[WIDTH][HEIGHT];
    unsigned char yoko[WIDTH][HEIGHT];
    unsigned char kuro[WIDTH][HEIGHT];
    Image img_tate = { (unsigned char*) tate, WIDTH, HEIGHT };
    Image img_yoko = { (unsigned char*) yoko, WIDTH, HEIGHT };
    Image img_kuro = { (unsigned char*) kuro, WIDTH, HEIGHT };

    int i, j;
    for (i = 0; i < WIDTH; i++)
        for (j = 0; j < HEIGHT; j++)
        {
            tate[i][j] = j; // 縦方向のグラデーション
            yoko[i][j] = i; // 横方向のグラデーション
        }
    CglSetAll(img_kuro, 0);

    CglSaveGrayBMP(img_tate, "gray-tate.bmp");
    CglSaveColorBMP(img_tate, img_kuro, img_kuro, "red-tate.bmp"); // 赤色の縦グラデーション
    CglSaveColorBMP(img_kuro, img_kuro, img_yoko, "blue-yoko.bmp"); // 青色の横グラデーション
    CglSaveColorBMP(img_tate, img_yoko, img_kuro, "red-green.bmp"); // 赤と緑の縦横グラデーション
}
```

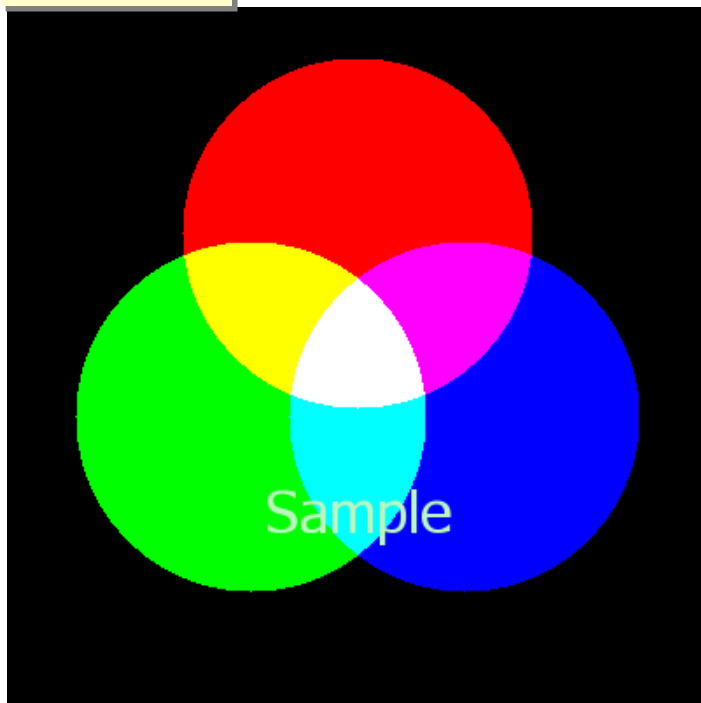


↑ Red ↑ Green ↑ Blue

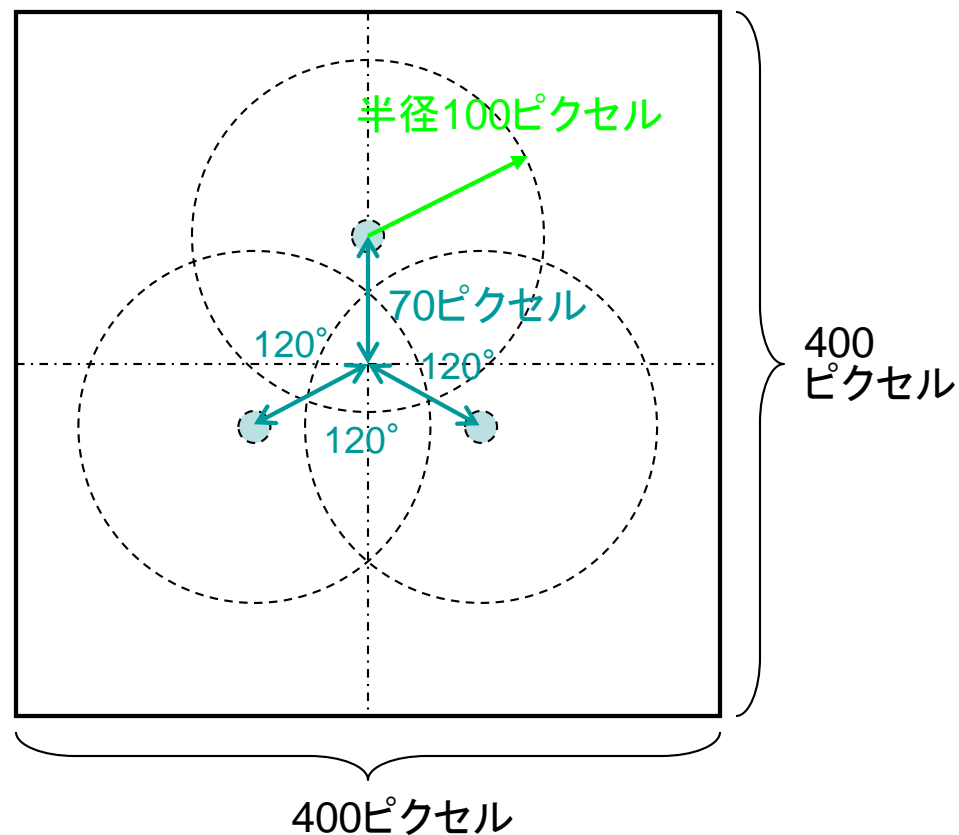
発展課題4

基本課題4で作成したPaintCircle()関数を再び用いて、下記の実行結果と同じ加法混色を示すカラー画像を作成するプログラムを作成しなさい。

実行結果



注) 図中の「Sample」の文字は、
解答には必要ない



発展課題4解答例

E君解答

```
#include "cglc.h"
#include <math.h>
#define WIDTH 400
#define HEIGHT 400
#define PI 3.1415 //円周率
#define si_ta PI/6 //横軸に対する角度(傾き)

void PaintCircle(Image img, int x0, int y0, int r, int g);

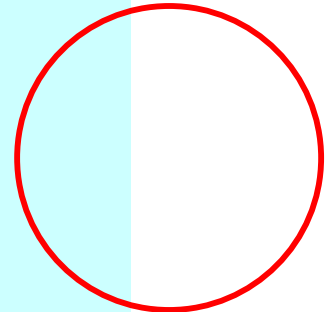
int main(void)
{
    /* 画像データ次元配列の宣言*/
    unsigned char red[WIDTH][HEIGHT];
    unsigned char green[WIDTH][HEIGHT];
    unsigned char blue[WIDTH][HEIGHT];

    /* Image型変数のimg_red, img_green, img_blueを宣言*/
    Image img_red = { (unsigned char*)red, WIDTH, HEIGHT };
    Image img_green = { (unsigned char*)green, WIDTH, HEIGHT };
    Image img_blue = { (unsigned char*)blue, WIDTH, HEIGHT };

    /* imgを初期化*/
    CglSetAll(img_red, 0);
    CglSetAll(img_green, 0);
    CglSetAll(img_blue, 0);

    /*座標と円の大きさとグレーレベルを指定し、円の内部をカラー化*/
    PaintCircle(img_red, 200, 270, 100, 255);
    PaintCircle(img_green, 200 - 70 * cos(si_ta), 200 - 70 * sin(si_ta), 100, 255);
    PaintCircle(img_blue, 200 + 70 * cos(si_ta), 200 - 70 * sin(si_ta), 100, 255);

    /*描画したものを拡張子.bmpで保存する*/
    CglSaveColorBMP(img_red, img_green, img_blue, "ColorPaint-Circles.bmp");
}
```



発展課題4 珍解答

F君解答

```
int main(void)
{
    int Nx=400, Ny=400;
    unsigned char* data = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    unsigned char* img = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    unsigned char* en1 = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    unsigned char* en2 = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);
    unsigned char* en3 = (unsigned char*) malloc(sizeof(unsigned char) * Nx * Ny);

    if (data == NULL) //か？
    {
        printf("メモリエラー!!");
        exit(0);
    }
    Image img_img = { (unsigned char*) img, 400, 400 };
    Image img_en1 = { (unsigned char*) en1, 400, 400 };
    Image img_en2 = { (unsigned char*) en2, 400, 400 };
    Image img_en3 = { (unsigned char*) en3, 400, 400 };

    CglSetAll(img_img, 0);
    PaintCircle(img_en1, 200, 270, 100, 255);
    PaintCircle(img_en2, 139, 165, 100, 255);
    PaintCircle(img_en3, 261, 165, 100, 255);

    CglSaveColorBMP(img_en1, img_en2, img_en3, "kansei4-2. bmp");
}
```

なぜか5カ所のメモリ領域を確保??

確保したメモリの内容は0になっていない！
不定な値になっている！

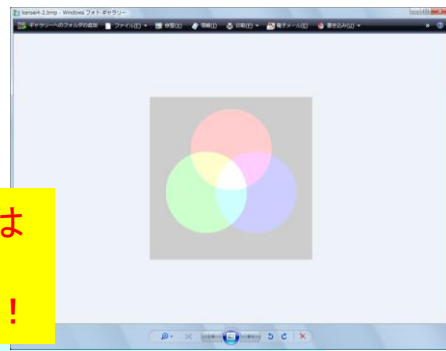
一つのメモリ領域だけを確認??
残り4つは大丈夫なの？

確認したメモリ領域は**使わず**、未確認のメモリ領域だけを画像描画に使う!?

画像img_imgのみを0でクリア。
残りの3つはクリアしないの??

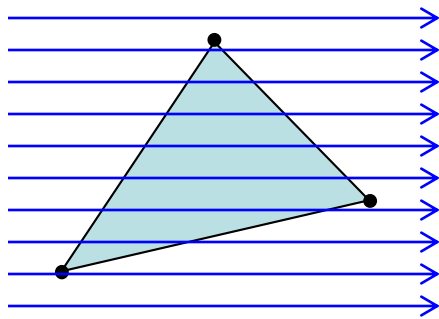
クリアしていない画像に
対して円を描画!?

クリアしていない画像を使ってカラー出力!?



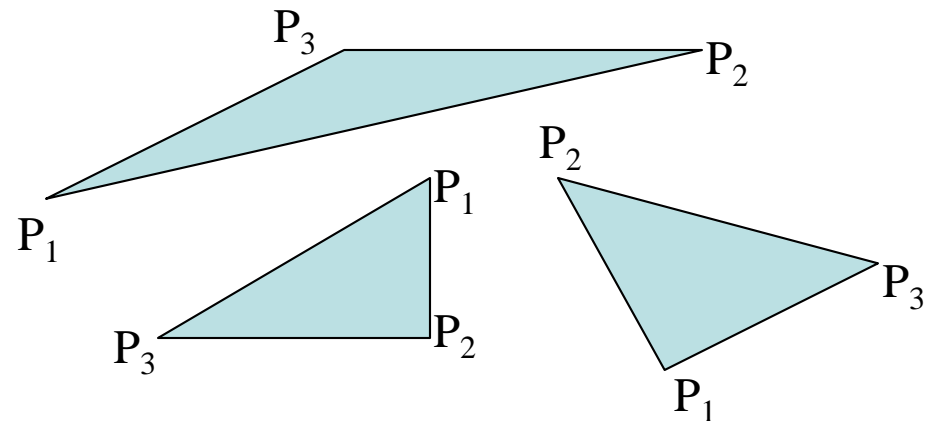
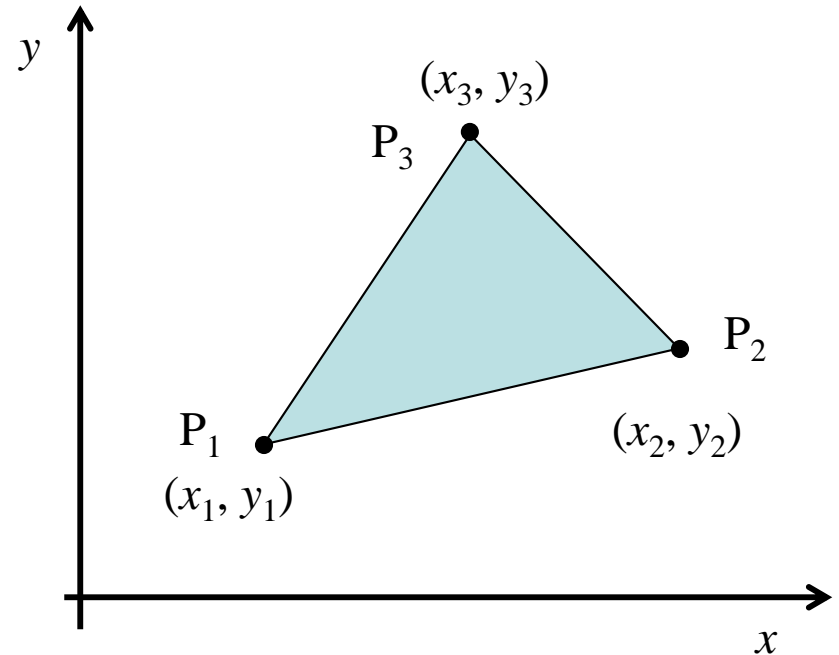
三角形の塗りつぶし(1)

3点 P_1 , P_2 , P_3 の座標を与えたとき,
この3点を頂点とする三角形を塗り
つぶす関数を考える



ラスタスキャンを行う

- ✓ 内側の判定をどうするか？
- ✓ クリッピングをどうするか？



三角形の塗りつぶし(2) — 陰関数表示

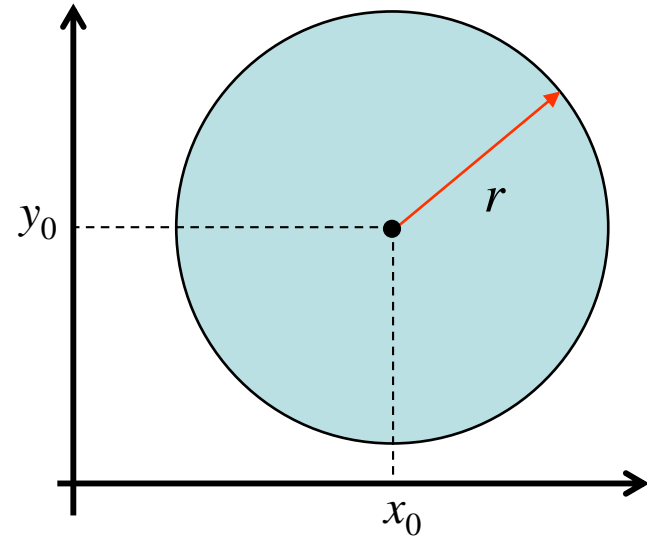
円の方程式 $(x - x_0)^2 + (y - y_0)^2 = r^2$

$(x - x_0)^2 + (y - y_0)^2 \leq r^2 \quad \rightarrow \quad \text{内側}$

陰関数による円の方程式

$$f(x, y) = (x - x_0)^2 + (y - y_0)^2 - r^2 \\ = 0$$

$f(x, y) < 0 \quad \rightarrow \quad \text{内側}$



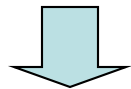
三角形の塗りつぶし(3) — 正領域・負領域

2点を通る直線の方程式(陽関数)

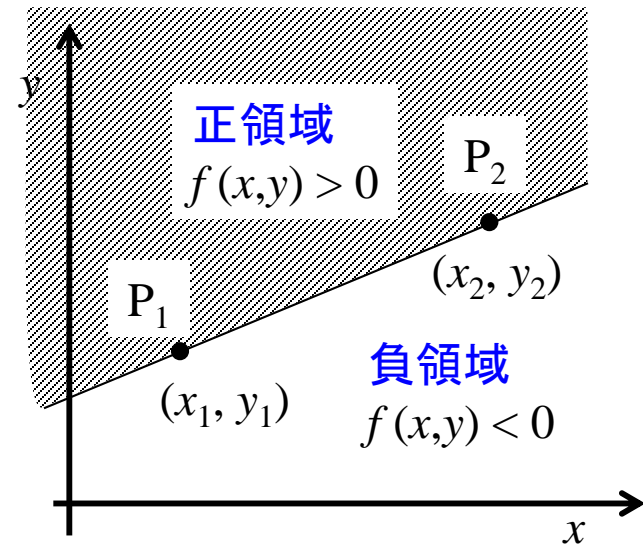
$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

2点を通る直線の方程式(陰関数)

$$f(x, y) = (y - y_1) - \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \\ = 0$$



$x_2 = x_1$ で発散

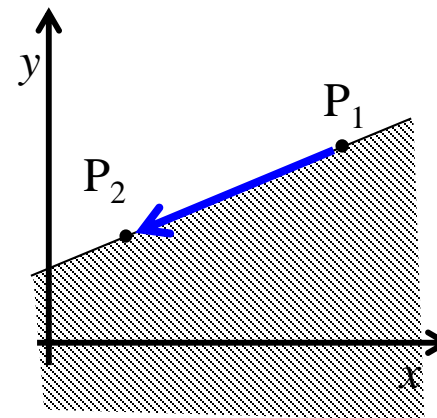
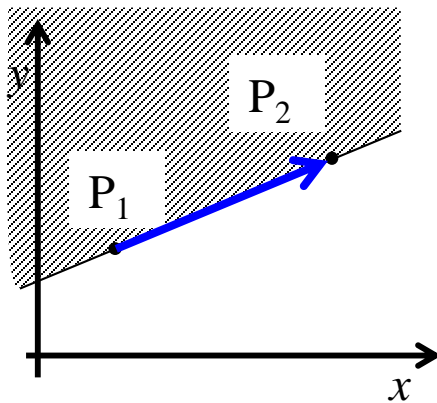
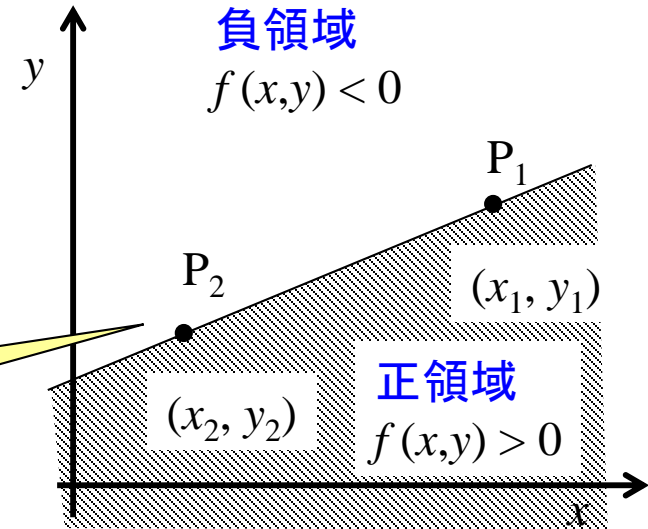


$$f(x, y) = (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)$$

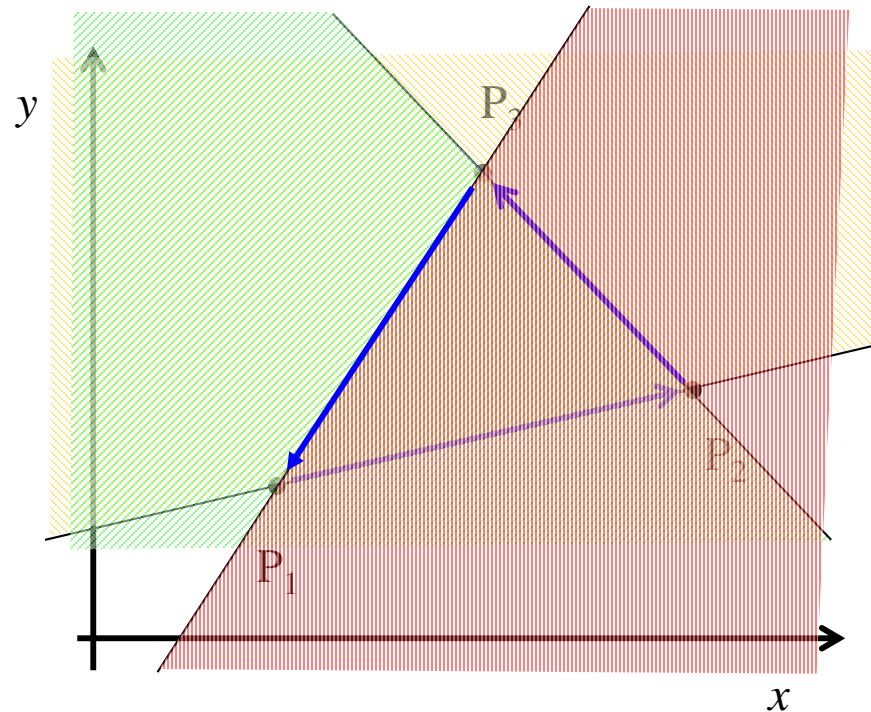
三角形の塗りつぶし(4) — 向きによる変化

$$f(x, y) = (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)$$

P_1 と P_2 を逆転



三角形の塗りつぶし(6) — 内側の条件

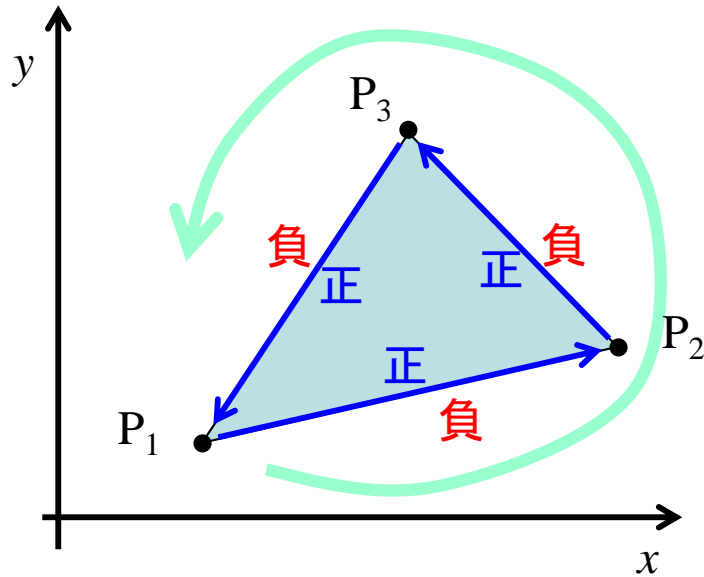


三角形の内側の条件

- ✓ 直線 P_1P_2 について正領域
- ✓ 直線 P_2P_3 について正領域
- ✓ 直線 P_3P_1 について正領域

3条件が全て満たされる場合が
三角形の内側

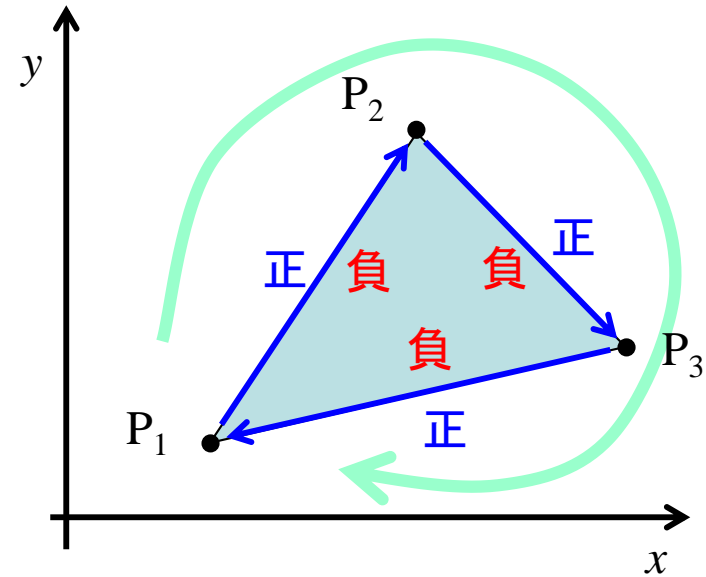
三角形の塗りつぶし(7) — 順序による差異



- ✓ 直線 P_1P_2 について正領域
- ✓ 直線 P_2P_3 について正領域
- ✓ 直線 P_3P_1 について正領域

パターン I

反時計回り

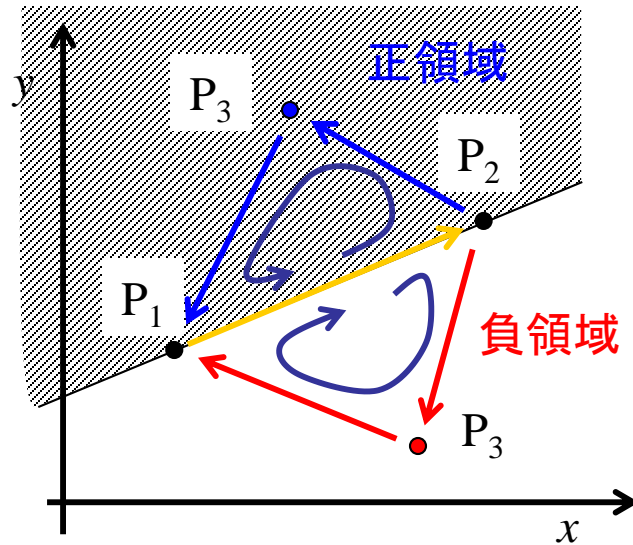


- ✓ 直線 P_1P_2 について負領域
- ✓ 直線 P_2P_3 について負領域
- ✓ 直線 P_3P_1 について負領域

パターン II

時計回り

三角形の塗りつぶし(8) 一回る方向の判別



直線 P_1P_2 に対して

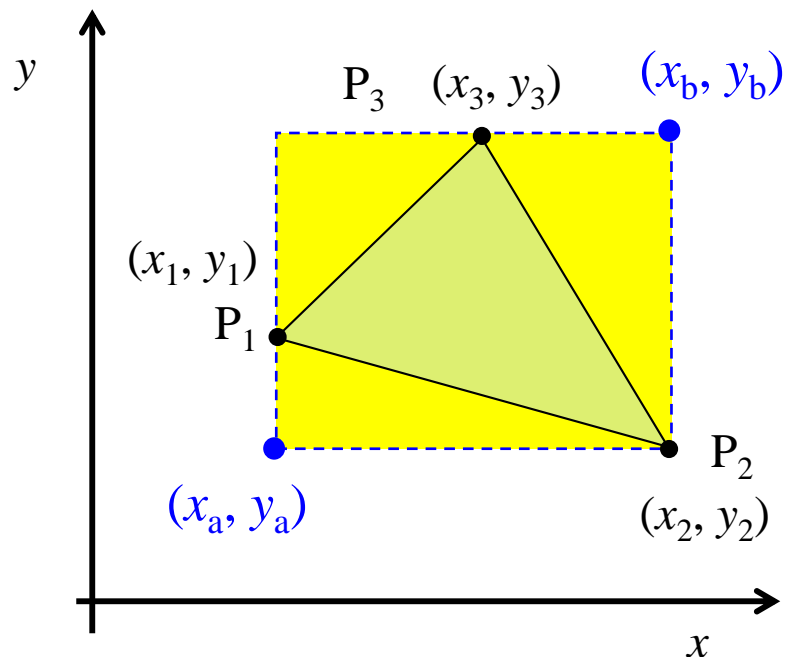
もし点 P_3 が**正領域**にあれば, パターン I

→ 3辺に対して全て**正領域**の条件で塗りつぶし

もし点 P_3 が**負領域**にあれば, パターン II

→ 3辺に対して全て**負領域**の条件で塗りつぶし

三角形の塗りつぶし(9) – クリッピング

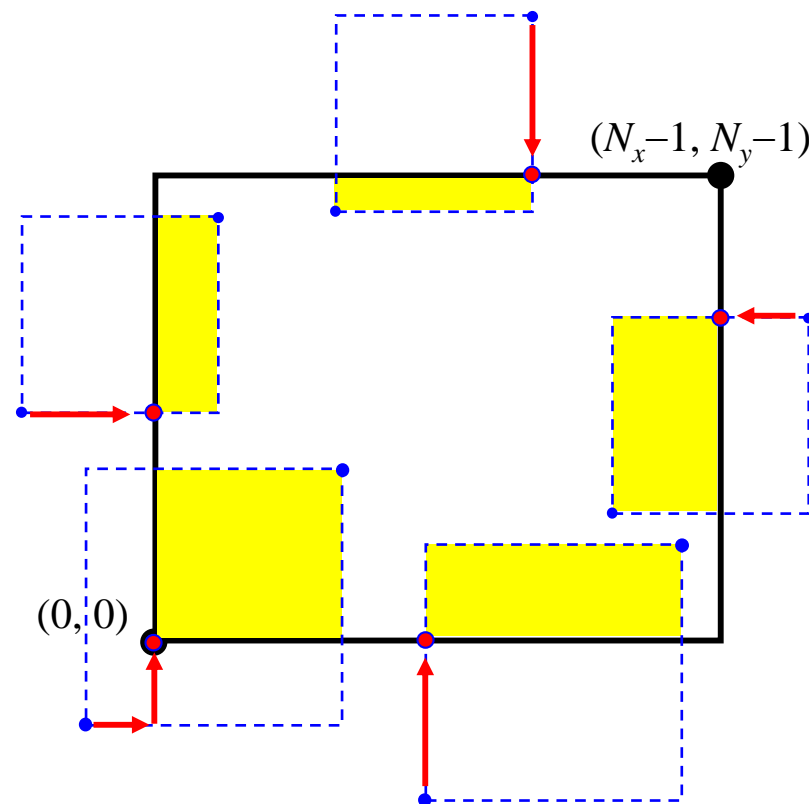


$x_a = [x_1, x_2, x_3 \text{ の中で最も小さな値}]$

$y_a = [y_1, y_2, y_3 \text{ の中で最も小さな値}]$

$x_b = [x_1, x_2, x_3 \text{ の中で最も大きな値}]$

$y_b = [y_1, y_2, y_3 \text{ の中で最も大きな値}]$



基本課題5

3点 (x_1, y_1) , (x_2, y_2) , (x_3, y_3) を頂点とする三角形をグレーレベル g で塗りつぶす関数 `PaintTriangle()` を作成せよ. 作成した関数と下記の`main()`を組み合わせることで実行例と同じ画像を得よ. なお関数`LineFunc()`を有効に活用すること.

Report5-1

$$f(x, y) = (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)$$

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

```
int LineFunc(int x, int y, int x1, int y1, int x2, int y2) //点(x,y)が正領域なら正值,
{ return (x2 - x1)*(y - y1) - (y2 - y1)*(x - x1); } //負領域なら負値を返す関数
```

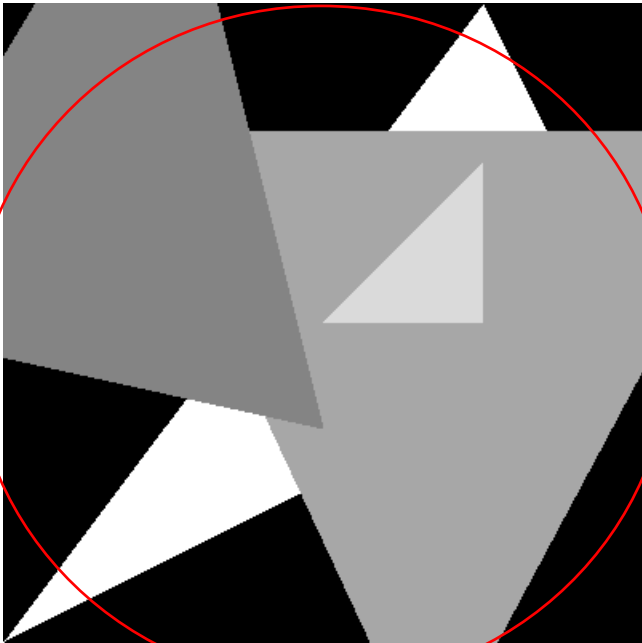
```
void PaintTriangle(Image img, int x1, int y1, int x2, int y2, int x3, int y3, int g)
{ /* この関数を作成せよ */ }
```

```
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);
    PaintTriangle(img, 0, 0, Nx - 1, Ny / 2, 3 * Nx / 4, Ny - 1, 255);
    PaintTriangle(img, Nx / 5, 4 * Ny / 5, Nx / 5 + Nx, 4 * Ny / 5, 2 * Nx / 3, -Ny / 5, 100);
    PaintTriangle(img, Nx / 2, Ny / 2, 3 * Nx / 4, 3 * Ny / 4, 3 * Nx / 4, Ny / 2, 180);
    PaintTriangle(img, Nx / 2, Ny / 3, Nx / 4, 4 * Ny / 3, -Nx / 4, Ny / 2, 60);
    CglSaveGrayBMP(img, "Triangles.bmp");
}
```

基本課題5 実行例

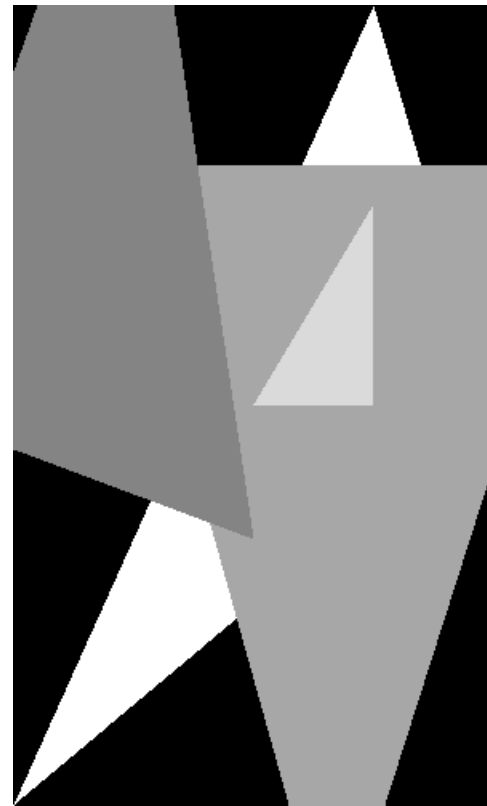
(1) 実行結果 (画面コピー)

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？ 400
続行するには何かキーを押してください . . .



(2) 実行結果(BMPファイル)

画像の横方向ピクセル数は？ 300
画像の縦方向ピクセル数は？ 500
続行するには何かキーを押してください . . .

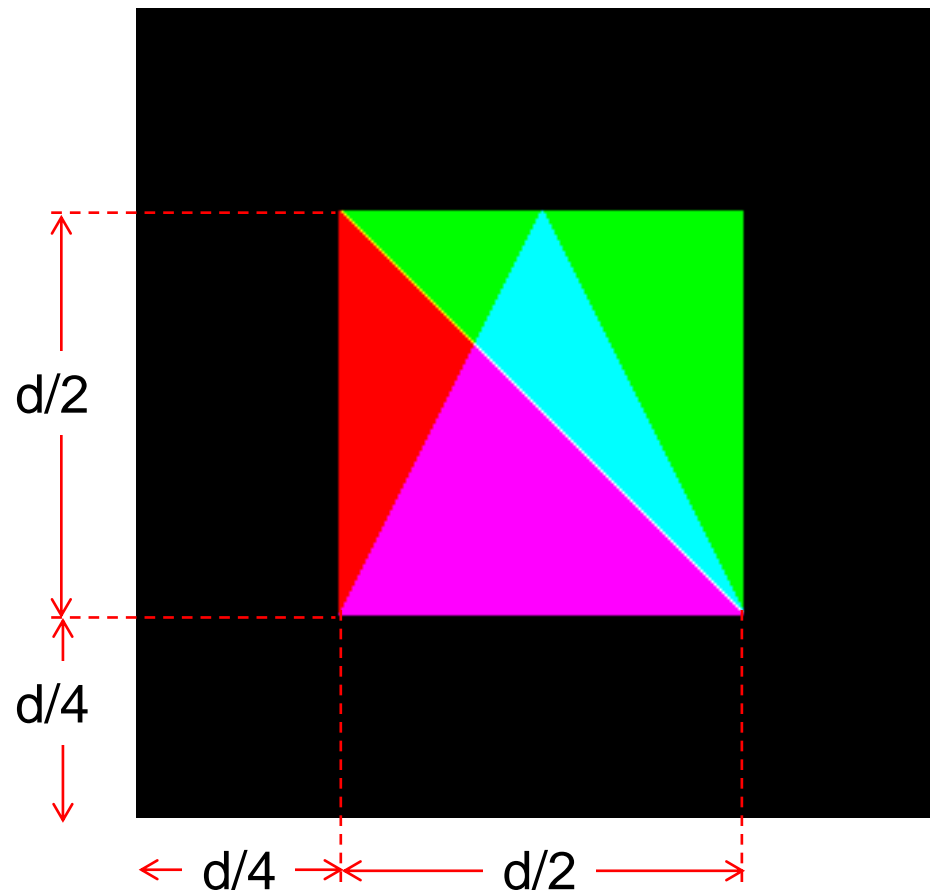


- ・ (1)と(2)の両方を提出！
- ・ 実行例を二つ以上提出！

発展課題5

PaintTriangle()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.

画像の横方向ピクセル数は? 300
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 . . .



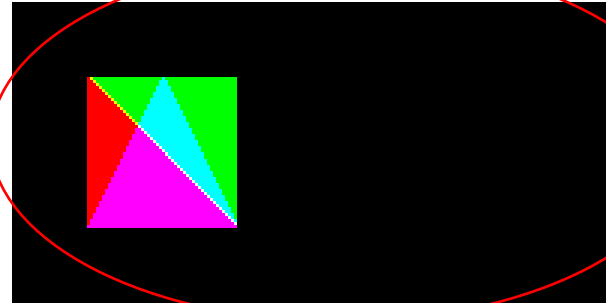
d: 短辺

発展課題5 実行例

PaintTriangle()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。

画像の横方向ピクセル数は？ 200
画像の縦方向ピクセル数は？ 100
続行するには何かキーを押してください . . .

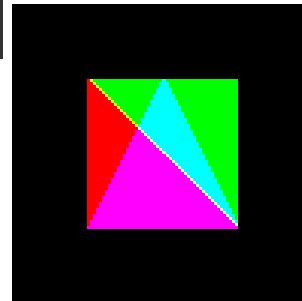
(1) 実行結果 (画面コピー)



(2) 実行結果(BMPファイル)

画像の横方向ピクセル数は？ 50
画像の縦方向ピクセル数は？ 200
続行するには何かキーを押してください . . .

画像の横方向ピクセル数は？ 100
画像の縦方向ピクセル数は？ 100
続行するには何かキーを押してください . . .



- ・ (1)と(2)の両方を提出！
- ・ 実行例を二つ以上提出！

三角形の塗りつぶし — まとめ

Step 1 三角形を囲む領域(x_a, y_a), (x_b, y_b)を求める

Step 2 三角形を囲む領域が画像の外にはみ出している場合のクリッピング処理を行う.

Step 3 3点の順番がパターン I (反時計回り)かパターン II (時計周り)かを判定する

Step 4 ラスタスキャンをおこなう. ただし,

(i) 3点の順番がパターン I の場合は, 3辺について正領域を塗りつぶす

(ii) 3点の順番がパターン II の場合は, 3辺について負領域を塗りつぶす