第3章 光栅图形学

课程目标

- 圆的基本概念
- 普通圆的生成算法
- 中点画圆法
- Bresenham 画圆算法
- 正负法

圆的八方向对称性

圆心在 (x_c, y_c)，半径为 R 圆的方程是：

$$(x-x_c)^2+(y-y_c)^2=R^2$$

圆关于坐标轴、和对角线对称

在对圆进行扫描转换时，只需计算 0^0 到 45^0 之间的一段圆弧就能得到整个圆。

圆的八方向对称性

对于圆心为 (x_c, y_c) 的圆，用程序 CirclePoints 显示 8 个相对称的点。

```
void CirclePoints (int xc, int yc, int x, int y, int Color)
{
    PutPixel (xc + x, yc + y, Color);
    PutPixel (xc + x, yc − y, Color);
    PutPixel (xc − x, yc + y, Color);
    PutPixel (xc − x, yc − y, Color);
    PutPixel (xc + y, yc + x, Color);
    PutPixel (xc + y, yc − x, Color);
    PutPixel (xc − y, yc + x, Color);
    PutPixel (xc − y, yc − x, Color);
}
```

圆的生成算法1--使用二次多项式

圆心在 (x_c, y_c)，半径为 R 圆的方程是：

$$(x-x_c)^2+(y-y_c)^2=R^2$$

利用该方程沿 x 轴从 x_c-R 到 x_c+R 以单位步长步进计算出对应的 y 值，即可得到圆周上每点的 y 值：

$$y=y_c\pm\sqrt{R^2-(x-x_c)^2}$$

缺点：每步包含大量的计算，所画像素位置的间距是不一致。

圆的生成算法2--使用极坐标

另一种消除不等间距的方法是使用圆的极坐标方程：

$$x = x_c + R\cos\theta,\ y = y_c + R\sin\theta$$

缺点：涉及到三角运算，同样增加了计算量。

最为常用的生成圆的有效算法有以下三种：

1. 中点画圆法
2. Bresenham 画圆算法
3. 正负法

圆的生成算法3--中点画圆法

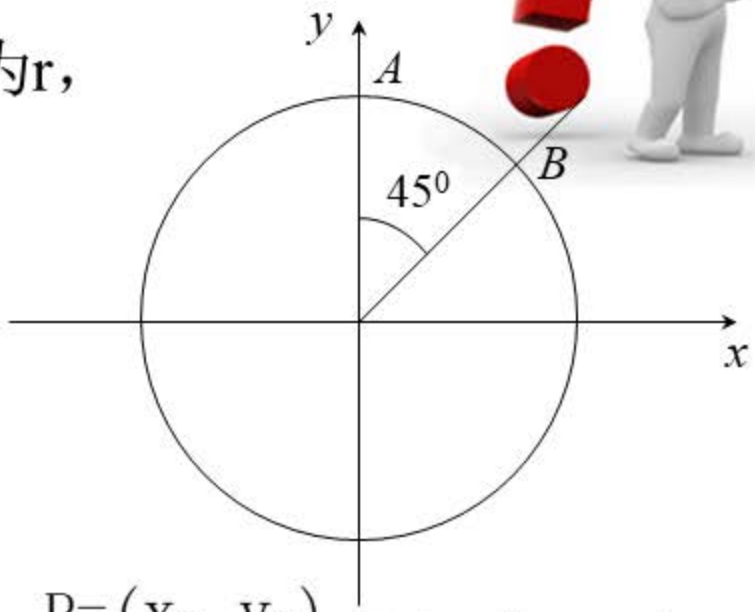
考虑中心在原点，半径为R的半径为r，

从x=0到x=y的1/8圆

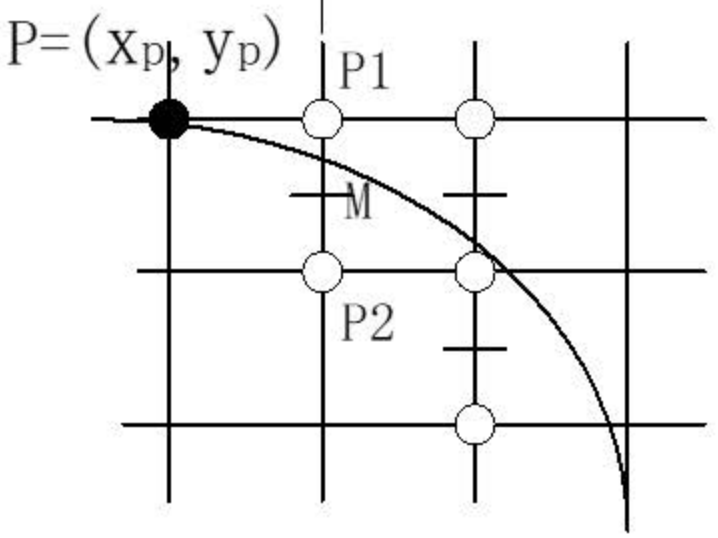
$$x_{i+1}=x_i+1$$

相应的y则在两种可能中选择：

$y=y_i$，或者 $y=y_i-1$

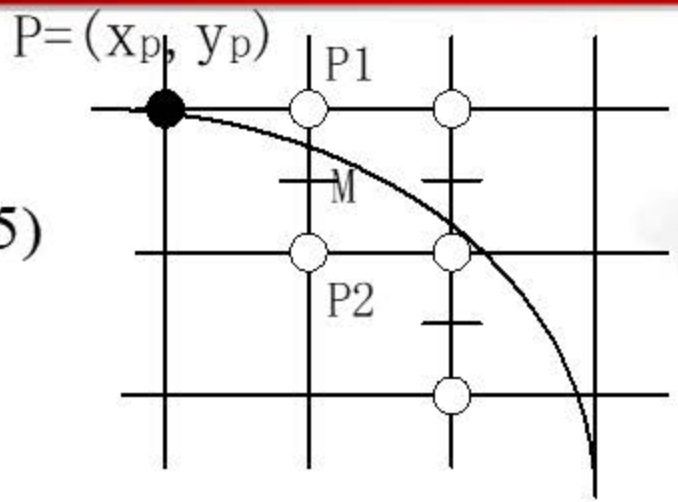


选择的原则是在2个像素之间的中点处给出一判定函数，并据此在2个像素中选择最靠近圆的那个像素。



圆的生成算法3--中点画圆法

构造判别式（圆方程）



$$d = F(M) = F(x_p + 1, y_p - 0.5)$$
$$= (x_p + 1)^2 + (y_p - 0.5)^2 - R^2$$

若 d<0, 则取P1为下一象素，而且再下一象素的判别式为

$$d' = F(x_p + 2, y_p - 0.5) = (x_p + 2)^2 + (y_p - 0.5)^2 - R^2 = d + 2x_p + 3$$

若d>=0, 则取P2为下一象素，而且再下一象素的判别式为

$$d' = F(x_p + 2, y_p - 1.5) = (x_p + 2)^2 + (y_p - 1.5)^2 - R^2 = d + 2(x_p - y_p) + 5$$

第一个象素是（0,R），判别式d的初始值为

$$d_0 = F(1, R - 0.5) = 1.25 - R$$

圆的生成算法3--中点画圆法

中点圆扫描转换算法

```
void MidpointCircle(int r, int Color)
{
        int x, y;
        float d;
        x = 0; y = r; d = 1.25 - r;
        CirclePoints(0, 0, x, y, Color);
        while(x < y) {
            if (d < 0) {d += x*2.0 + 3; }
            else{d += (x - y)*2.0 + 5; y --;}
            x ++;
           CirclePoints(0, 0, x, y, Color);
        }
}
```

圆的生成算法3--中点画圆法

由于判定函数 d 的初值是浮点数，因此算法必须做浮点数的算术运算。

希望有一个效率更高的只进行整数运算的算法。

为此，定义一个新的判定函数：$h = d - 0.25$

这样在初始化时，$h = 1-R$，而条件 $d<0$ 变成了 $h<-0.25$。

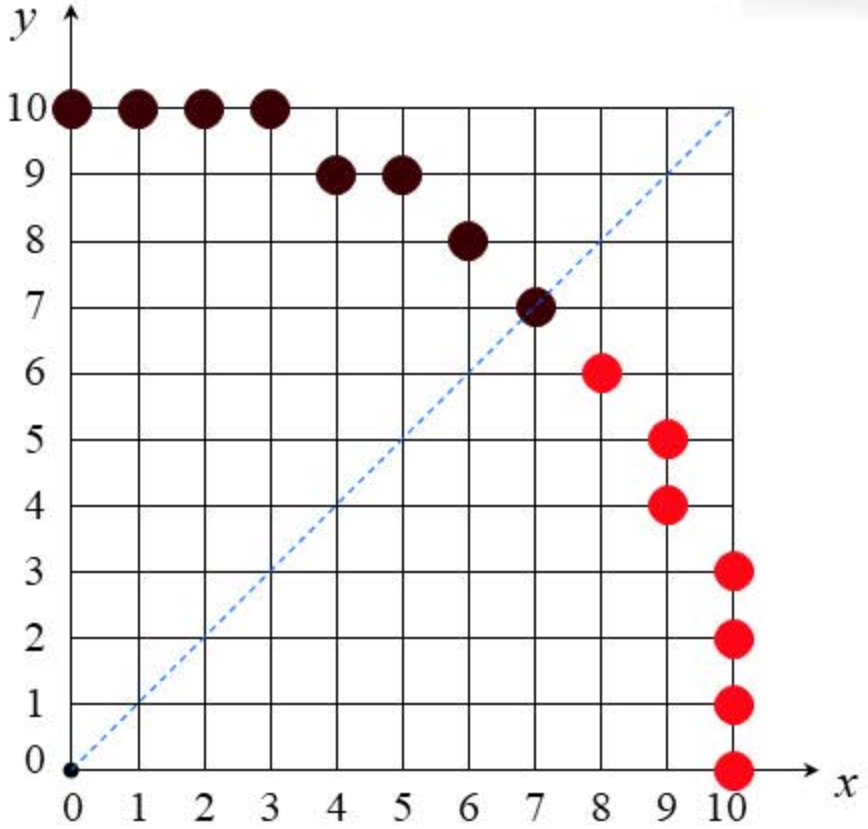
由于 h 的初值是整数，且其相应的增量也是整数，因此可将条件 $h<-0.25$ 改为 $h<0$ 。

这样算法就变成了关于 h 的只进行整数运算的形式。

圆的生成算法3--中点画圆法

例：用中点算法画圆心为 $O(0, 0)$，半径为 $R = 10$ 的八分之一圆弧。

d	x	y
–9	0	10
–6	1	10
–1	2	10
6	3	10
–3	4	9
8	5	9
5	6	8
	7	7



圆的生成算法3--中点画圆法

```
void MidpointCircle(int r, int Color)
{
    int x, y, d;
    x = 0; y = r; d = 1 - r;
    CirclePoints (0, 0, x, y, Color);
    while( x < y){
      if (d < 0) { d += x*2 + 3; }
      else { d += (x - y)*2 + 5;  y --; }
      x ++;
      CirclePoints (0, 0, x, y, Color);
    }
}
```

Circle.cpp

圆的生成算法4-- Bresenham 画圆法

Bresenham 算法是画圆的最有效的算法之一。

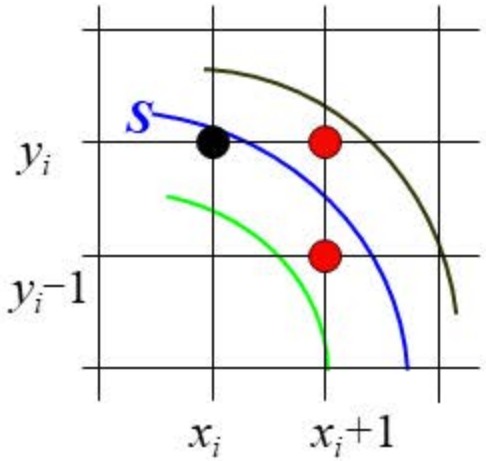
设要画的圆的圆心在坐标原点 (0, 0)，半径为 R。

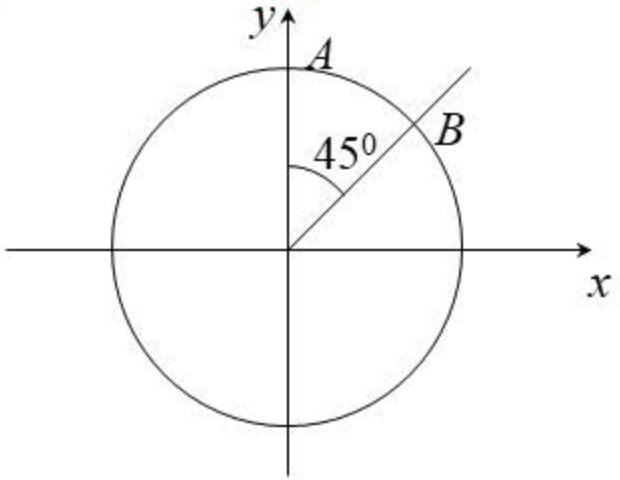
考虑第一象限的上半部的八分之一圆弧的生成过程

在这种情况下，x 每步增加 1，相应的 y_{i+1} 则有两种选择：

$y_{i+1}=y_i$ 或 y_i-1。

选择的原则是考察精确值 y 是靠近 y_i 还是靠近 y_i-1。





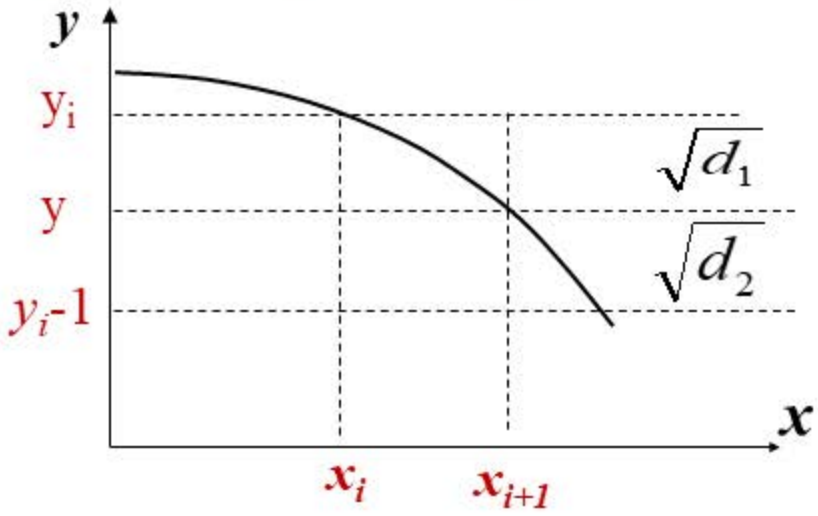
圆的生成算法4-- Bresenham 画圆法

Bresenham画圆算法采用一个决策值来确定到底是选择y_i还是y_i-1。

在$x=x_i+1$位置上，用d_1和d_2来标识两个候选像素的y值与圆弧上理想y值的差值，则：$y^2=r^2-(x_i+1)^2$

$$d_1=y_i^2-y^2=y_i^2-r^2+(x_i+1)^2$$

$$d_2=y^2-(y_i-1)^2=r^2-(x_i+1)^2-(y_i-1)^2$$



圆的生成算法4-- Bresenham 画圆法

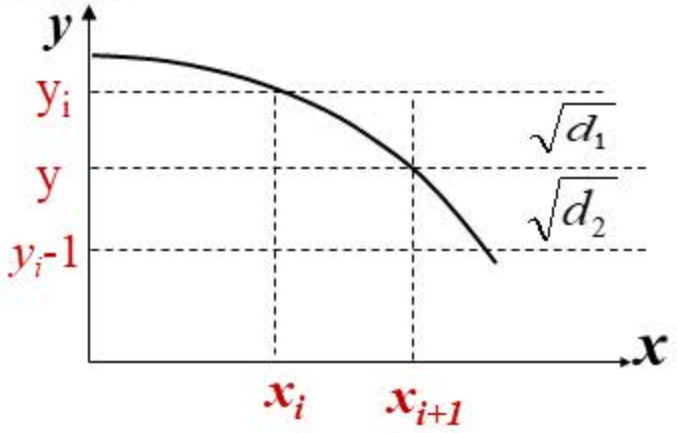
令$d_i=d_1-d_2$，并代入d_1、d_2，则有：

$$d_i=2(x_i+1)^2+y_i^2+(y_i-1)^2-2r^2$$

这里d_i就是Bresenham画圆算法的第i步决策值。

如果$d_i<0$，则$y_{i+1}=y_i$；

如果$d_i>=0$，则$y_{i+1}=y_i-1$。



圆的生成算法4-- Bresenham 画圆法

余下的问题是如何快速地计算判定函数 d_i ？

$$d_i=2(x_i+1)^2+y_i^2+(y_i-1)^2-2r^2$$

在i+1步，d_{i+1}为：$d_{i+1}=2(x_{i+1}+1)^2+y_{i+1}^2+(y_{i+1}-1)^2-2r^2$

已知$x_{i+1}=x_i+1$，因而得到：$d_{i+1}=2(x_i+1+1)^2+y_{i+1}^2+(y_{i+1}-1)^2-2r^2$

若$d_i<0$，取右方像素，$y_{i+1}=y_i$，则：

$$d_{i+1}=2(x_i+1+1)^2+y_i^2+(y_i-1)^2-2r^2=d_i+4x_i+6$$

若$d_i>=0$，取右下方像素，$y_{i+1}=y_i-1$，则：

$$d_{i+1}=2(x_i+1+1)^2+(y_i-1)^2+(y_i-1-1)^2-2r^2=d_i+4(x_i-y_i)+10$$

而决策值的初值d_0由x=0,y=r代入前面公式，得：

$$d_0=2(0+1)^2+r^2+(r-1)^2-2r^2=3-2r$$

圆的生成算法4-- Bresenham 画圆法

基于上述推导，生成圆的 Bresenham 算法之步骤如下：

Step 1. 初始化：$x=0$，$y=R$，$d=3-2R$;

Step 2. 当 $x<y$ 时，执行以下操作：

2.1 画像素点 (x, y);

2.2 求下一个像素点：

$$x=x+1,\quad y=\begin{cases} y, & d<0 \\ y-1, & d\geq 0 \end{cases}$$

2.3 计算下一个判定函数：

$$d=\begin{cases} d+4x+6, & d<0 \\ d+4(x-y)+10, & d\geq 0 \end{cases}$$

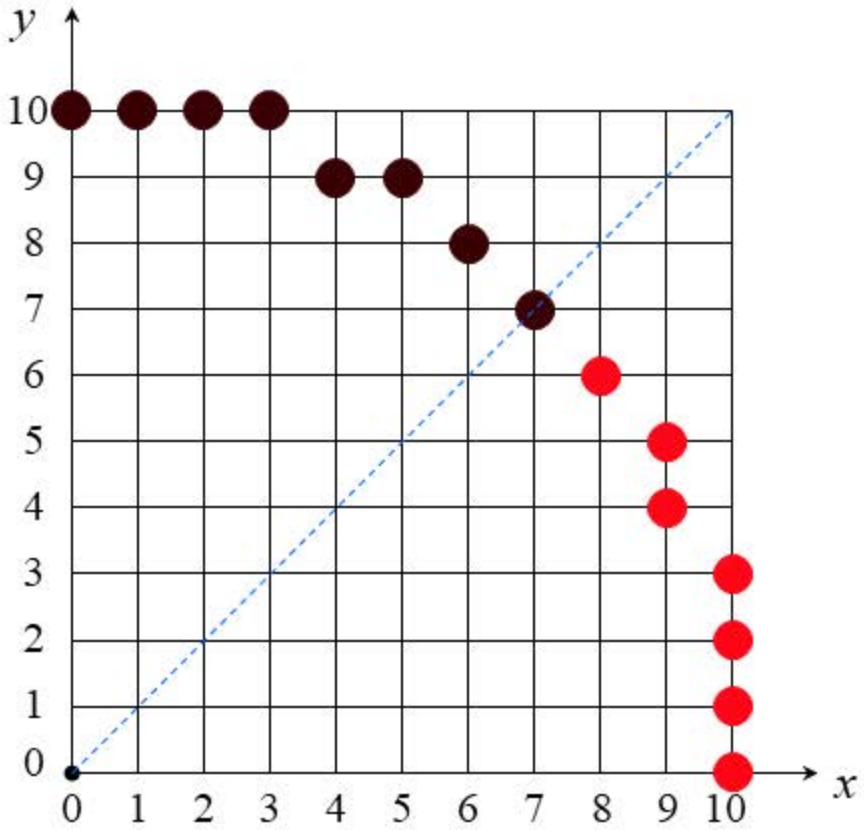
Step 3. 若 $x=y$，画像素点 (x, y);

Step 4. 算法结束。

圆的生成算法4-- Bresenham 画圆法

例 用 Bresenham 算法画圆心为 $O(0, 0)$，半径为 $R = 10$ 的八分之一圆弧。

d	x	y
−17	0	10
−11	1	10
−1	2	10
13	3	10
−5	4	9
17	5	9
11	6	8
	7	7



圆的生成算法4-- Bresenham 画圆法

圆 Bresenham 扫描转换算法

```
void Circle_BRES ( int xc, int yc, int R, int Color )
{
    int x = 0, y = R, d = 3 − 2*R;
    while (x < y) {
        CirclePoints (xc, yc, x, y, Color);
        if (d >= 0) {
            d += 4 − 4*y; y−−;
        }
        d += 4*x + 6;  x ++;
    }
    if (x == y)
        CirclePoints (xc, yc, x, y, Color);
}
```

Circle.cpp

圆的生成算法5--正负法

设要绘制的圆的圆心在 (x_c, y_c) 半径为 R，令：

$$F_{circle}(x，\ y) = (x - x_c)^2 + (y - y_c)^2 - R^2$$

则圆将平面分为三个区域：

$$F_{circle}(x, y)\begin{cases} <0 \ ，\ (x, y) \text{ 位于圆周内} \\ =0 \ ，\ (x, y) \text{ 位于圆周上} \\ >0 \ ，\ (x, y) \text{ 位于圆周外} \end{cases}$$

以第一象限的四分之一圆弧为例，说明正负法的原理。

圆的生成算法5--正负法

取$P_0(x_0, y_0) = (x_c, y_c + R)$，求得点$P_i(x_i, y_i)$后，寻找$P_{i+1}(x_{i+1}, y_{i+1})$的原则是：

当$F_{circle}(x_i, y_i) \le 0$时，$x_{i+1} = x_i + 1$，$y_{i+1} = y_i$；

当$F_{circle}(x_i, y_i) > 0$时，$x_{i+1} = x_i$，$y_{i+1} = y_i - 1$。

即，当点P_i在圆周内或圆周上时，向右走一步，而当点P_i在圆周外时，向下走一步。这样用于逼近圆弧的像素点均在圆周的附近，且使判定函数时正时负，这就是正负法名称的由来。

圆的生成算法5--正负法

每找一点都要计算一次判定函数 $F_{circle}(x, y)$。

如果直接计算，由于要进行乘法，不仅耗时，而且也不易硬件实现。

因此，应建立计算判定函数 $F_{circle}(x, y)$ 的高效算法以避免乘法。

圆的生成算法5--正负法

$F_{circle}(x_0, y_0) = 0$。若 $F_{circle}(x_i, y_i)$ 已经求出，

那么$F_{circle}(x_{i+1}, y_{i+1})$ 的计算分两种情形：

情形1．$F_{circle}(x_i, y_i) \le 0$. 此时，$x_{i+1} = x_i + 1$，$y_{i+1} = y_i$，那么

$$F_{circle}(x_{i+1}, y_{i+1}) = (x_{i+1} - x_c)^2 + (y_{i+1} - y_c)^2 - R^2$$

$$= (x_i + 1 - x_c)^2 + (y_i - y_c)^2 - R^2$$

$$= F_{circle}(x_i, y_i) + 2(x_i - x_c) + 1$$

圆的生成算法5--正负法

情形2．$F_{circle}(x_i, y_i) > 0.$

此时，$x_{i+1} = x_i + 1$，$y_{i+1} = y_i - 1$，那么

$$\begin{aligned} F_{circle}(x_{i+1},\ y_{i+1}) &= (x_{i+1} - x_c)^2 + (y_{i+1} - y_c)^2 - R^2 \\ &= (x_i - x_c)^2 + (y_i - 1 - y_c)^2 - R^2 \\ &= F_{circle}(x_i,\ y_i) - 2(y_i - y_c) + 1 \end{aligned}$$

因此，由 (x_i, y_i) 求下一点判定函数的递推公式为：

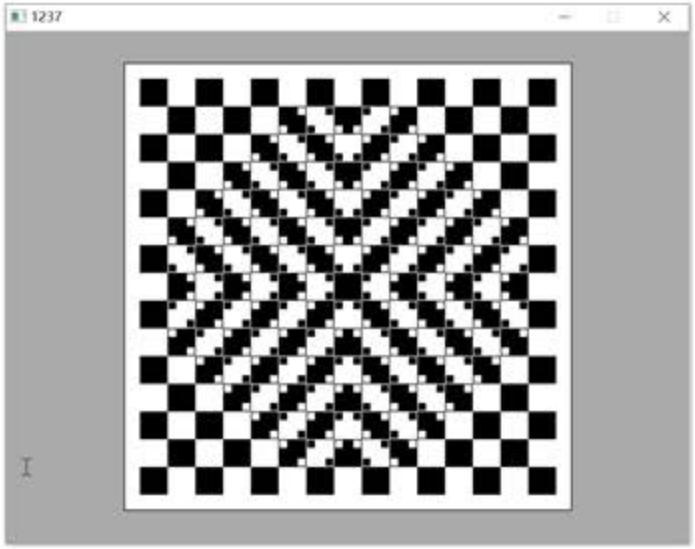
$$F_{circle}(x_{i+1},\ y_{i+1}) = \begin{cases} F_{circle}(x_i,\ y_i) + 2(x_i - x_c) + 1, & F_{circle}(x_i,\ y_i) \le 0 \\ F_{circle}(x_i,\ y_i) - 2(y_i - y_c) + 1, & F_{circle}(x_i,\ y_i) > 0 \end{cases}$$

圆的生成算法5--正负法

```
void circle_PNM (int xCenter, int yCenter, int Radius, int Color)
    {
      int x = xCenter, y = yCenter + Radius;
      int f = 0;
      while (y > yCenter) {
          PutPixel (x, y, Color);
          if (f > 0) {f += 2*(yCenter − y) + 1; y −−;  }
          else { f += 2*(x - xCenter) + 1;  x ++; }
       }
      if (y == yCenter) PutPixel (x, y, Color)
    }
```

Circle.cpp

视觉错觉艺术图片



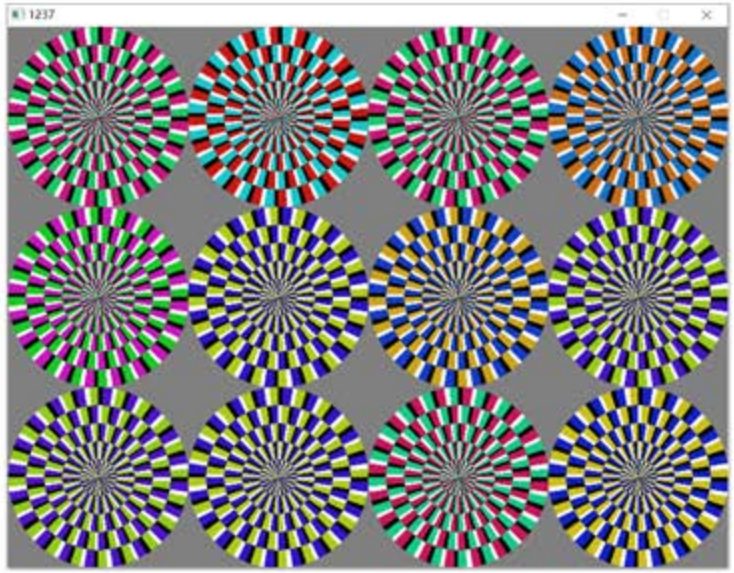
- 这是网上经常见的视觉错觉艺术图片，可以用程序生成。

视觉错觉艺术图片

```
#include <graphics.h>
#include <conio.h>


// 定义回调
void (*callback)(int x, int y);


// 画方块上的小方块
void DrawSmallBox(int x, int y, bool lt, bool rt, bool lb, bool rb)
{
    int nx = x * 26 - 13;
    int ny = y * 26 - 13;
    if (lt) solidrectangle(nx + 1,  ny + 1,  nx + 1 + 6, ny + 1 + 6);
    if (rt) solidrectangle(nx + 24, ny + 1, nx + 24 - 6, ny + 1 + 6);
    if (lb) solidrectangle(nx + 1,  ny + 24, nx + 1 + 6, ny + 24 - 6);
    if (rb) solidrectangle(nx + 24, ny + 24, nx + 24 - 6, ny + 24 - 6);
}


// 圆中的每个点(回调函数)
void CirclePoints(int x, int y)
{
    if (x == 0 && y < 0)
        DrawSmallBox(x, y, false, false, true, true);
    else if (x == 0 && y > 0)
        DrawSmallBox(x, y, true, true, false, false);
    else if (x < 0 && y == 0)
        DrawSmallBox(x, y, false, true, false, true);
    else if (x > 0 && y == 0)
        DrawSmallBox(x, y, true, false, true, false);
    else if (x == 0 && y == 0)
        ;
    else if ((x < 0 && y < 0) || (x > 0 && y > 0))
        DrawSmallBox(x, y, false, true, true, false);
    else
        DrawSmallBox(x, y, true, false, false, true);
}


// 基于 Bresenham 算法画填充圆
void FillCircle_Bresenham(int x, int y, int r)
{
    int tx = 0, ty = r, d = 3 - 2 * r, i;

    while( tx < ty)
    {
        // 画水平两点连线(<45度)
        for (i = x - ty; i <= x + ty; i++)
        {
            CirclePoints(i, y - tx);
            if (tx != 0)    // 防止水平线重复绘制
                CirclePoints(i, y + tx);
        }

        if (d < 0)          // 取上面的点
            d += 4 * tx + 6;
        else                // 取下面的点
        {
            // 画水平两点连线(>45度)
            for (i = x - tx; i <= x + tx; i++)
            {
                CirclePoints(i, y - ty);
                CirclePoints(i, y + ty);
            }

            d += 4 * (tx - ty) + 10, ty--;
        }

        tx++;
    }

    if (tx == ty)           // 画水平两点连线(=45度)
        for (i = x - ty; i <= x + ty; i++)
        {
            CirclePoints(i, y - tx);
            CirclePoints(i, y + tx);
        }
}


// 主函数
void main()
{
    // 创建绘图窗口
    initgraph(640, 480);
    setbkcolor(LIGHTGRAY);
    cleardevice();
    setorigin(320, 240);

    // 画 15 x 15 的间隔黑白块，每块 26 x 26，共 390 x 390
    setlinecolor(BLACK);
    setfillcolor(WHITE);
    fillrectangle(-210, -210, 209, 209);
    COLORREF fc = WHITE;
    for (int x = -195; x < 195; x += 26)
        for (int y = -195; y < 195; y += 26)
        {
            fc = (~fc) & 0xffffff;
            setfillcolor(fc);
            solidrectangle(x, y, x + 25, y + 25);
        }

    // 在填充圆内的方块内画小方块
    setrop2(R2_XORPEN);
    setfillcolor(WHITE);
    FillCircle_Bresenham(0, 0, 6);

    // 按任意键退出
    getch();
    closegraph();
}
```

VisualPhoto.cpp

旋转圆



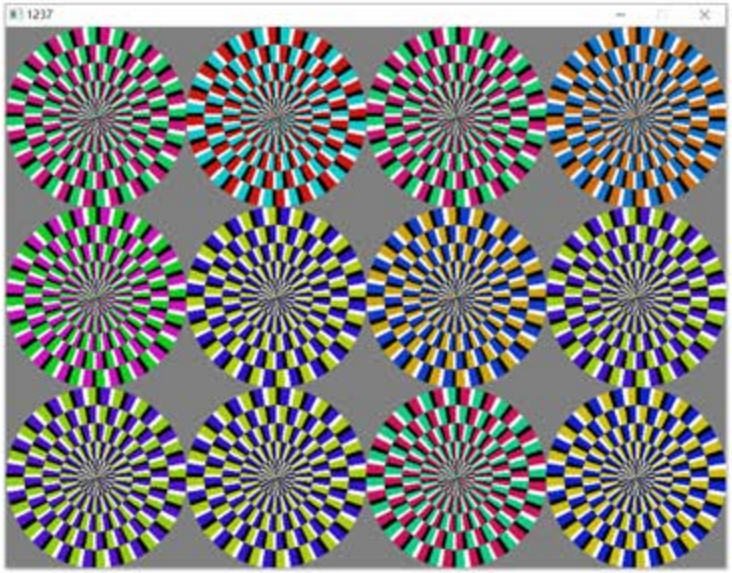
绘制过程

绘制非常神奇的错觉图片，静止的圆盘看起来却有在转动的错觉。

首先学习绘制扇形函数和RGB颜色模型，绘制了一个基本单元；然后学习了for循环语句和循环的嵌套，实现了旋转圆的绘制；最后学习了HSV颜色模型，并利用随机函数和按键切换，实现了丰富多变的旋转圆错觉图案。

代码分析

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <time.h>
int main()
{
    float Pi = 3.14159;  // 圆周率Pi
    initgraph(800,600); // 打开一个窗口
    setbkcolor(RGB(128,128,128)); // 设置背景颜色为灰色
    cleardevice();  // 以背景颜色清空画布
    srand(time(0)); // 随机种子函数

    int centerX,centerY; // 圆心坐标
    int radius; // 圆半径
    int i;
    float offset;  // 同一半径各组扇形之间的角度偏移量
    float totalOffset; // 不同半径之间的角度偏移量
    while(1)  // 重复执行
    {
        for (centerX = 100;centerX<800;centerX=centerX+200) // 对圆心x坐标循环
        {
            for (centerY = 100;centerY<600;centerY=centerY+200) // 对圆心y坐标循环
            {
                totalOffset = 0; // 同一半径各组扇形之间的角度偏移量
                float h = rand()%180; // 随机色调
                COLORREF  color1 = HSVtoRGB(h,0.9,0.8); // 色调1生成的颜色1
                COLORREF  color2 = HSVtoRGB(h+180,0.9,0.8); // 色调2生成的颜色2
                for (radius=100;radius>0;radius=radius-20) // 半径从大到小绘制
                {
                    int left = centerX - radius; // 圆外切矩形左上角x坐标
                    int top = centerY - radius; // 圆外切矩形左上角y坐标
                    int right = centerX + radius; // 圆外切矩形右下角x坐标
                    int bottom = centerY + radius; // 圆外切矩形右下角y坐标
                    for (i=0;i<20;i++) // 绕着旋转一周，绘制扇形区域
                    {
                        offset = i*Pi/10 + totalOffset; // 各组扇形之间偏移的角度
                        setfillcolor(color1); // 色调1生成的颜色1
                        solidpie(left,top,right,bottom,offset,2*Pi/60+offset);
                        setfillcolor(RGB(255,255,255));// 设置填充颜色为白色
                        solidpie(left,top,right,bottom,2*Pi/60+offset,3*Pi/60+offset);
                        setfillcolor(color2); // 色调2生成的颜色2
                        solidpie(left,top,right,bottom,3*Pi/60+offset,5*Pi/60+offset);
                        setfillcolor(RGB(0,0,0));// 设置填充颜色为黑色
                        solidpie(left,top,right,bottom,5*Pi/60+offset,6*Pi/60+offset);
                    }
                    totalOffset = totalOffset + Pi/20; // 不同半径间角度偏移量为Pi/20
                }
            }
        }
        _getch();  // 暂停，等待按键输入
    }
    return 0;
}
```



rotationcircle.cpp

冰墩墩

将冰墩墩简化成多个椭圆，并且按照不同部位，分步用函数分别绘制。

适合初学者借鉴学习与加以细化，例如可以细化一下冰墩墩的腿部和手部细节，将它肚子上的文字改成冬奥LOGO，改变一下眼睛的颜色等等，还可以画一只雪容融。

可以借鉴的地方

本程序改变了原有的坐标原点，通过观察，我们发现冰墩墩大部分身体都是对称的，因此，可以通过改变它的坐标原点，只需获取一半的坐标点，通过对称绘制另一半，减少获取坐标需要的时间。

代码分析

```
#include<graphics.h>
#include<conio.h>
#include<math.h>

#define PI acos(-1.0)
#define WIDTH 800
#define HEIGHT 800
double th = PI / 180;

void DrawBack();                                    // 绘制背景
void DrawEar();                                     // 绘制耳朵
void DrawLeg();                                     // 绘制腿
void DrawArm();                                     // 绘制胳膊
void DrawBody();                                    // 绘制身体
void DrawEye();                                     // 绘制眼睛
void DrawNose();                                    // 绘制鼻子
void DrawMouth();                                   // 绘制嘴
void DrawColour();                                  // 绘制彩带
void DrawLogo();                                    // 绘制标志
void heart(int x0, int y0, int size, COLORREF C);                          // 绘制心
void DrawEllipse(int x0, int y0, int a, int b, int k, COLORREF color);     // 绘制倾斜椭圆

int main()
{
    initgraph(WIDTH, HEIGHT);
    DrawBack();
    setorigin(WIDTH / 2, HEIGHT / 2);               // 设置坐标系
    DrawEar();                                      // 绘制耳朵
    DrawLeg();                                      // 绘制腿
    DrawArm();                                      // 绘制胳膊
    DrawBody();                                     // 绘制身体
    DrawEye();                                      // 绘制眼睛
    DrawNose();                                     // 绘制鼻子
    DrawMouth();                                    // 绘制嘴
    DrawColour();                                   // 绘制彩带
    DrawLogo();                                     // 绘制标志
    heart(330, -120, 10, RED);
    setfillcolor(RED);
    floodfill(330, -100, RED);
    _getch();
    return 0;
}
```

代码分析

```
void DrawBack()
{
    float H = 190;       // 色相
    float S = 1;         // 饱和度
    float L = 0.7f;      // 亮度
    for (int y = 0; y < HEIGHT; y++)
    {
        L += 0.0002f;
        setlinecolor(HSLtoRGB(H, S, L));
        line(0, y, WIDTH - 1, y);
    }
}

void DrawEar()
{
    setfillcolor(BLACK);
    solidcircle(172, -300, 62);
    solidcircle(-172, -300, 62);
}

void DrawLeg()
{
    setfillcolor(BLACK);
    solidellipse(44, 155, 168, 348);
    solidellipse(-44, 155, -168, 348);
}

void DrawArm()
{
    DrawEllipse(-267, 50, 100, 60, 50, BLACK);
    DrawEllipse(297, -60, 100, 60, 50, BLACK);
    setfillcolor(BLACK);
    floodfill(-267, 50, BLACK);
    floodfill(297, -60, BLACK);
}

void DrawBody()
{
    setlinecolor(BLACK);
    setlinestyle(PS_SOLID, 8);
    setfillcolor(WHITE);
    fillellipse(-270, -354, 270, 260);
}
```

```
void DrawEye()
{
    DrawEllipse(109, -131, 84, 60, 314, BLACK);
    DrawEllipse(-109, -131, 84, 60, -314, BLACK);
    setfillcolor(BLACK);
    floodfill(109, -131, BLACK);
    floodfill(-109, -131, BLACK);
    setfillcolor(WHITE);
    setlinestyle(PS_SOLID, 1);
    solidcircle(92, -137, 30);
    solidcircle(-92, -137, 30);
    setfillcolor(BLACK);
    solidcircle(90, -137, 26);
    solidcircle(-90, -137, 26);
    setfillcolor(WHITE);
    solidcircle(81, -146, 9);
    solidcircle(-81, -146, 9);
}

void DrawNose()
{
    setlinestyle(PS_SOLID, 1);
    setfillcolor(BLACK);
    solidellipse(-26, -106, 26, -63);
}

void DrawMouth()
{
    setlinecolor(BLACK);
    setlinestyle(PS_SOLID, 8);
    arc(-43, -75, 43, -7, PI, 0);
}

void DrawColour()
{
    setlinecolor(BLACK);
    setlinestyle(PS_SOLID, 8);
    setlinecolor(RGB(91, 198, 250));
    ellipse(-205, -265, 205, 74);
    setlinecolor(RGB(119, 216, 113));
    ellipse(-215, -275, 215, 84);

    setlinecolor(RGB(254, 122, 185));
    ellipse(-225, -285, 225, 94);
}
```

代码分析

```
void DrawLogo()
{
    RECT r = { -116, 100, 116, 175 };
    settextcolor(BLACK);
    setbkmode(TRANSPARENT);
    settextstyle(60, 0, _T("黑体"));
    drawtext(_T("BeiJing"), &r, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    RECT r1 = { -116, 175, 116, 250 };
    drawtext(_T("2022"), &r1, DT_CENTER | DT_VCENTER | DT_SINGLELINE);
}

void DrawEllipse(int x0, int y0, int a, int b, int k, COLORREF color)
{
    double i;
    double x, y, tx, ty;
    for (i = -180; i <= 180; i = i + 0.5)
    {
        x = a * cos(i * th);
        y = b * sin(i * th);
        tx = x;
        ty = y;
        x = tx * cos(k * th) - ty * sin(k * th) + x0;
        y = y0 - (ty * cos(k * th) + tx * sin(k * th));
        setfillcolor(color);
        solidcircle((int)x, (int)y, 2);
    }
}

void heart(int x0, int y0, int size, COLORREF C)
{
    double m, n, x, y;
    double i;
    for (i = 0; i <= 2 * size; i = i + 0.01)
    {
        // 产生极坐标点
        m = i;
        n = -size * (((sin(i) * sqrt(fabs(cos(i)))) / (sin(i) + 1.4142)) - 2 * sin(i) + 2);
        // 转换为笛卡尔坐标
        x = n * cos(m) + x0;
        y = n * sin(m) + y0;
        setfillcolor(C);
        solidcircle((int)x, (int)y, 1);
    }
}
```

bingdwendwen.cpp

实验

1. 实现中点画圆算法。
2. Bresenham画圆算法。
3. 正负法画圆。
4. 绘制作品