计算机图形学个人作业

本作业是在 Ubuntu 16.04 + QT5.9 环境下完成的。完成了图元、样条曲线、分形曲线三种类型图形的绘制。主界面较为简洁，如下图一，只添加了按钮，没有加入鼠标点击、键盘输入等用户交互功能。整体构建采用继承于 QMainWindow 大类的 MainWindow 类，再通过调用 MainWindow 类中的 paintEvent 函数进行绘图操作。在 Qt Creator 的 IDE 自带的 UI 绘制组件 Qt Designer 中添加按钮 PushButton，同时启用“可按下”选项。利用 QT 的信号槽功能将按钮未按下与按下的信号通过槽传到 MainWindow 中，由条件语句判断是否启用绘制相关图形的算法。

由于操作系统的内核不同，所以本程序需要运行在 Linux 环境下，在终端下执行$ ./zjq_hw 即可。

图一

一、图元的生成

1. 直线

   直线的生成采用的是 Breenham 画线算法。其原理为：

   i) 输入线段的两个端点，并将左端点存储在(x_0, y_0)中；

   ii) 将装入帧缓冲器，画出第一个点；

   iii) 计算常量Δx、Δy、$2\Delta y$和$2\Delta y-2\Delta x$，并得到决策参数的第一个值：$p_0=2\Delta y-\Delta x$

iv)　从 k=0 开始，在沿线路径的每个x_k处，进行下列检测：
如果$p_k < 0$，下一个要绘制的点是(x_k+1, y_k)，并且

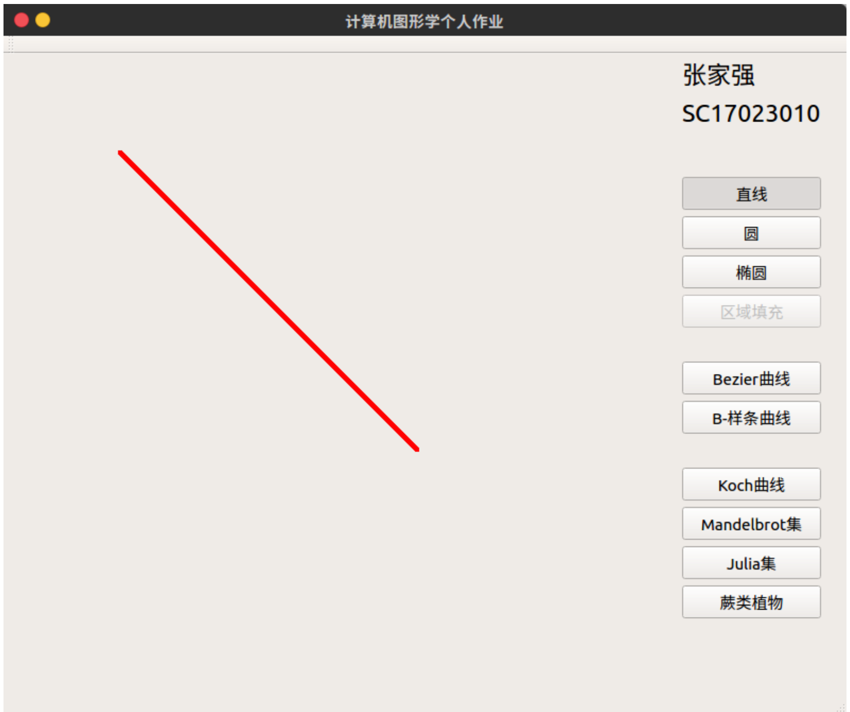
$$p_{k+1} = p_k + 2\Delta y$$

否则下一个绘制的点是(x_k+1, y_k+1)，并且

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

v)　重复步骤 iv）共Δx次

结果如下图二。



图二

2. 圆

圆的生成采用的是 Breenham 画圆算法。其原理为：

i)　输入圆的半径 r 和圆心(x_c, y_c),t 得到第一个点（0，r）；

ii)　计算决策参数的初始值：$p_0 = \frac{5}{4} - r$

iii)　利用 Bresenhm 算法画出八分之一圆

iv)　利用圆的对称性画出整个圆

代码如下：

```
//Bresenham画圆 中点画圆
if(if_drawcircle == true)
{
    QPainter painter(this);
//     painter.drawLine(QPointF(1,1), QPointF(100,100));


    //画刷，线宽，画笔风格，画笔端点，画笔连接风格
    QPen pen(Qt::red, 5, Qt::DotLine, Qt::RoundCap, Qt::RoundJoin);
    painter.setPen(pen);
    int xCenter = 250;//圆心
    int yCenter = 250;
    int radius = 200;//半径
```

```
int x = 0;
int y = radius;
int p = 1 - radius;
painter.drawPoint(QPoint(250, 450));

//painter.drawPoint(QPoint(252, 450));

while (x < y) {
    x++;
    if(p < 0)
    {
        p += 2*x + 1;
        painter.drawPoint(QPoint(xCenter + x, yCenter + y));//画点
        painter.drawPoint(QPoint(xCenter - x, yCenter + y));
        painter.drawPoint(QPoint(xCenter - x, yCenter - y));
        painter.drawPoint(QPoint(xCenter + x, yCenter - y));
        painter.drawPoint(QPoint(xCenter + y, yCenter + x));
        painter.drawPoint(QPoint(xCenter - y, yCenter + x));
        painter.drawPoint(QPoint(xCenter - y, yCenter - x));
        painter.drawPoint(QPoint(xCenter + y, yCenter - x));
    }
    else {
        y--;
        p += 2*(x - y) + 1;
        painter.drawPoint(QPoint(xCenter + y, yCenter + x));//画点
        painter.drawPoint(QPoint(xCenter - y, yCenter + x));
        painter.drawPoint(QPoint(xCenter - y, yCenter - x));
        painter.drawPoint(QPoint(xCenter + y, yCenter - x));
        painter.drawPoint(QPoint(xCenter + x, yCenter + y));
        painter.drawPoint(QPoint(xCenter - x, yCenter + y));
        painter.drawPoint(QPoint(xCenter - x, yCenter - y));
        painter.drawPoint(QPoint(xCenter + x, yCenter - y));
    }
}
qDebug()<<"C";
     }
```

结果如下图三

图三

3. 椭圆

椭圆生成采用中点算法。生成原理与圆较为相似，但是由于椭圆是半对称，而圆是全对称的，所以椭圆是先画出四分之一的椭圆。代码如下：

```
 //椭圆中点算法
  if(if_drawellipse == true)
  {
      int Rx = 150;//顶点
      int Ry = 200;
      int xCenter = 250; //椭圆中心
      int yCenter = 250; //
      int Rx2 = Rx * Rx;
      int Ry2 = Ry * Ry;
      int twoRx2 = 2 * Rx2;
      int twoRy2 = 2 * Ry2;
      int p;
      int x = 0;
      int y = Ry;
      int px = 0;
      int py = twoRx2 * y;
      QPainter painter(this);
      //画刷，线宽，画笔风格，画笔端点，画笔连接风格
      QPen pen(Qt::red, 5, Qt::DotLine, Qt::RoundCap, Qt::RoundJoin);
      painter.setPen(pen);
      p = ROUND(Ry2 - (Rx2 * Ry) + (0.25 *Rx2));

//          painter.drawPoint(QPoint(300, 300));

      //Region 1
      while (px < py) {
          x++;
```

```
        px += twoRy2;
        if(p < 0)
        {
            p += Ry2 + px;
            painter.drawPoint(QPoint(xCenter + x, yCenter + y));//画点
            painter.drawPoint(QPoint(xCenter - x, yCenter + y));
            painter.drawPoint(QPoint(xCenter + x, yCenter - y));
            painter.drawPoint(QPoint(xCenter - x, yCenter - y));
        }
        else {
            y--;
            py -= twoRx2;
            p += Ry2 + px - py;
            painter.drawPoint(QPoint(xCenter + x, yCenter + y));//画点
            painter.drawPoint(QPoint(xCenter - x, yCenter + y));
            painter.drawPoint(QPoint(xCenter + x, yCenter - y));
            painter.drawPoint(QPoint(xCenter - x, yCenter - y));
        }
    }

    //Region 2
    p = ROUND(Ry2 * (x + 0.5) * (x + 0.5) + Rx2 * (y - 1) * (y - 1) - Rx2 * Ry2);
    while (y > 0) {
        y--;
        py -= twoRx2;
        if(p > 0)
        {
            p += Rx2 - py;
            painter.drawPoint(QPoint(xCenter + x, yCenter + y));//画点
            painter.drawPoint(QPoint(xCenter - x, yCenter + y));
            painter.drawPoint(QPoint(xCenter + x, yCenter - y));
            painter.drawPoint(QPoint(xCenter - x, yCenter - y));
        }
        else {
            x++;
            px += twoRy2;
            p += Rx2 - py + px;
            painter.drawPoint(QPoint(xCenter + x, yCenter + y));//画点
            painter.drawPoint(QPoint(xCenter - x, yCenter + y));
            painter.drawPoint(QPoint(xCenter + x, yCenter - y));
            painter.drawPoint(QPoint(xCenter - x, yCenter - y));
        }
    }
    qDebug()<<"E";
        }
```

结果如下图四

4. 区域填充
   区域填充没有最终实现，在 Debug 过程中，点击区域填充的按钮后，主界面会直接跳出循环报错，并且界面产生无响应的反应。

二、样条曲线的生成

1. Bezier 曲线的生成

由于我没有对画线算法进行封装，所以在绘制 Bezier 样条曲线及其它一些图形时调用了系统的 drawLine 库函数，但并不影响算法的实现。我的算法可以进行 n 次 Bezier 曲线的绘制，结果的图中设置了三个控制点，即是二次 Bezier 曲线。

$$B(t)=\sum_{i=0}^{n}\binom{n}{i}Pi(1-t)^{n-i}t^{i}=\binom{n}{0}P0(1-t)^{n}t^{0}+\binom{n}{1}P1(1-t)^{n-1}t^{1}+\ldots+\binom{n}{n-1}Pn-1(1-t)^{1}t^{n-1}+\binom{n}{n}Pn(1-t)^{0}t^{n},t\in[0,1]$$

Bezier 曲线是通过控制点来生成的，当 t 变化时，就能得到曲线的坐标。

结果如图六

算法部分代码如下：

```
void Bezier<Point>::AllBernstein(int n, double u, double *B)
{
    B[0] = 1.0;
    double u1 = 1.0-u;
    for(int j=1; j<=n;j++)
    {
        double saved  = 0.0;
        for(int k=0; k<j; k++)
        {
            double temp = B[k];
            B[k] = saved+u1*temp;
            saved = u*temp;
        }
        B[j] = saved;
    }
     }
```

画图部分代码如下：

```
    //Bezier曲线
    if(if_bezier == true)
    {
        //QPainter painter(this);
        QPainter *painter = new QPainter(this);
        //画刷，线宽，画笔风格，画笔端点，画笔连接风格
        QPen pen(Qt::red, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);
        QPen pen1(Qt::darkBlue, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);
        //painter.setPen(pen1);
        Bezier<QPointF>  *bezier;// = new Bezier<QPointF>;
```

```
        bezier = new Bezier<QPointF>();
        QPointF p[3];
        p[0] = QPointF(100,100);
        p[1] = QPointF(500,400);
        p[2]  = QPointF(100,500);
        painter->setPen(pen);
        painter->drawLine(p[0],p[1]);
        painter->drawLine(p[1],p[2]);
        bezier->appendPoint(p[0]);
        bezier->appendPoint(p[1]);
        bezier->appendPoint(p[2]);

        QPainterPath path = bezier->getPainterPath();
        painter->setPen(pen1);
        painter->drawPath(path);
        delete painter;

        qDebug()<<"Be";
    }
```

2. B 样条曲线的生成

B 样条多项式次数是独立于控制点数目的，同时可以局部控制曲线，但 B 样条较 Bezier 曲线更复杂。

n 次样条基为

$$\mathbf{S}(t) = \sum_{i=0}^{m} \mathbf{P}_i b_{i,n}(t),\ t \in [0,1].$$

m+1 个 n 次 B 样条基可以用 Cox-de Boor 递归公式 定义：

$$b_{j,0}(t) := \begin{cases} 1 & t_j < t < t_{j+1} \\ 0 & \cdots \end{cases}$$

$$b_{j,n}(t) := \frac{t - t_j}{t_{j+n} - t_j} b_{j,n-1}(t) + \frac{t_{j+n+1} - t}{t_{j+n+1} - t_{j+1}} b_{j+1,n-1}(t).$$

当节点等距，称 B 样条为均匀(uniform)否则为非均匀(non-uniform)。

相关画图代码如下：

```
//B样条
    if(if_b == true)
    {
        QPainter painter(this);
        int ctrlpoint_num = 6; //控制点数
        int points_num = 100; //曲线上点数
        int px[] = {30, 140, 380, 460, 540, 550};//控制点
        int py[] = {350, 240, 150, 260, 450, 150};
```

```
//画刷，线宽，画笔风格，画笔端点，画笔连接风格
QPen pen(Qt::red, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);
//QPen pen1(Qt::darkBlue, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);

QPainterPath path;
path.moveTo(px[0], py[0]);
for(int i = 0; i < ctrlpoint_num; i++)
{
    path.lineTo(px[i], py[i]);
    painter.drawPath(path);
    painter.setPen(pen);
}

float T[ctrlpoint_num + 4];
for(int i = 0; i < ctrlpoint_num + 4; i++)
    T[i] = i;
float pointsX[ctrlpoint_num + 3][4], pointsY[ctrlpoint_num + 3][4];
bool first = true;
for(int i = 3; i < ctrlpoint_num; i++)
{
    float delta = (T[i + 1] - T[i])/points_num;
    for(float t=T[i]; t<=T[i+1]; t+=delta)
    {
        for(int k=0; k<=3; k++)
        {
            for(int j=i-3+k; j<=i; j++)
            {
            if(k==0)
            {
                pointsX[j][k] = px[j];
                pointsY[j][k] = py[j];
            }
            else
            {
                float alpha = (t-T[j]) / (T[j+4-k]-T[j]);
                pointsX[j][k] = (1.0 - alpha) * pointsX[j-1][k-1] + alpha * pointsX[j][k-1];
                pointsY[j][k] = (1.0 - alpha) * pointsY[j-1][k-1] + alpha * pointsY[j][k-1];
                if(j == i && k == 3)
                {
                    if(first)
                    {
                        path.moveTo(pointsX[j][3], pointsY[j][3]);
                        painter.setPen(pen);
                        painter.drawPath(path);
                    }
                    else
                    {
                        path.lineTo(pointsX[j][3], pointsY[j][3]);
                        painter.setPen(pen);
                        painter.drawPath(path);
                    }
                    first = false;
                }
            }
            }
```

```
            }
        }
    }

    qDebug()<<"B_s";
      }
```

结果如图七

三、分形图形的生成

1. Koch 曲线

Koch 曲线的算法为给定线段 AB，科赫曲线可以由以下步骤生成：

i) 将线段分成三等份（AC,CD,DB）

ii) 以 CD 为底，向外（内外随意）画一个等边三角形 DMC

iii) 将线段 CD 移去

iv) 分别对 AC,CM,MD,DB 重复 1~3。

我的 Koch 选取了两个点（40，400）（600，400）作为第一次迭代的起始位置。相关代码如下：

```
//Koch 曲线
```

```
if(if_koch == true)
{
    QPainter painter(this);

    QPen pen(Qt::red, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);
    painter.setPen(pen);
    koch.ctrlPoints.push_back(QPointF(40,400));
    koch.ctrlPoints.push_back(QPointF(600,400));
    for (int i = 0; i < koch.ctrlPoints.size()-1; i++){
        koch.generateKoch(koch.ctrlPoints[i], koch.ctrlPoints[i + 1]);

        //koch.curvePoints1.clear();
        koch.k = 0;
    }
    for (int i = 0; i<koch.curveLine.size(); i++)
    {
        painter.drawLine(koch.curveLine[i]);
    }

        }
```

结果如图八

2. Mandelbrot 集

曼德博集合可以用复二次多项式来定义，c 为一个复数参数

$$f_c(z) = z^2 + c$$

从 $z=0$ 开始对 $f_c(z)$ 进行迭代:

$$z_{n+1}=z_n^2+c, n=0,1,2,\ldots$$

$$z_0=0$$

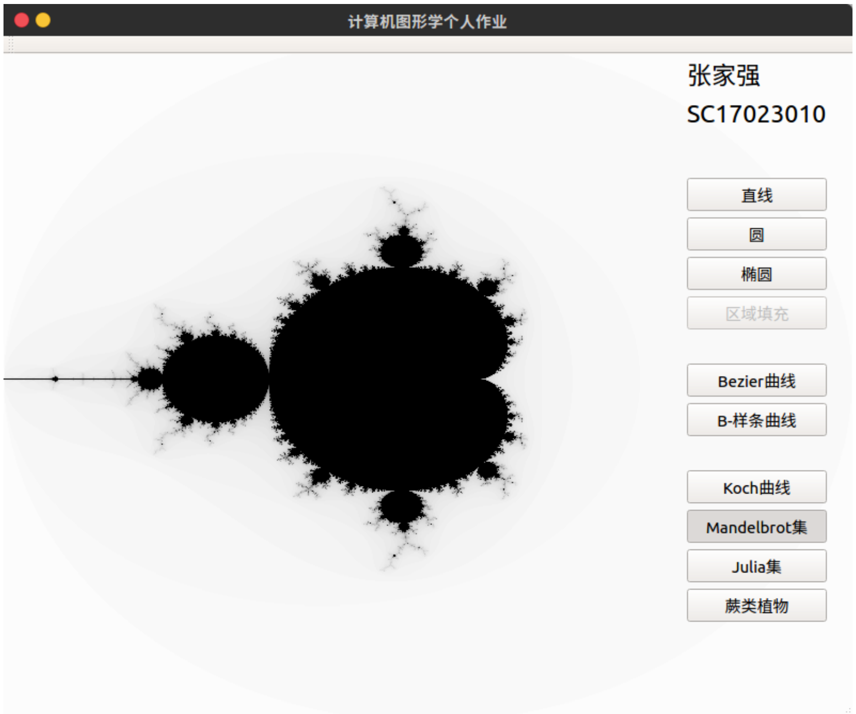
$$z_1=z_0^2+c=c$$

$$z_2=z_1^2+c=c^2+c$$

我定义了一个复数类 idComplex 方便实现 Mandelbrot 集的绘制，绘图代码如下

```
//Mandelbrot Set
if(if_mandelbrot == true)
{
    QSize size = this->size();
    double width = size.width()/2;//横向位置
    double height = size.height()/2; //纵向位置
    double scale = 2; //放缩倍数，越小图越大
    int const max_time = 100;
    QPainter painter(this);
    QPen pen1;
    for(int a =- width; a <= width; a++)
    {
        for(int b =- height; b <= height; b++ )
        {
            int times = max_time;
            idComplex c0(a/width*scale, b/height*scale); //Mandelbrot 原
            idComplex c1(0, 0);// = c0;

            while(times--)
            {
                  c1 = c1 * c1 + c0;
                  if(c1.Operatmod() > 2)
                      break;
            }
            times = times < 0 ? 0 : times;
            //if(times>99){pen1.setColor(QColor(255, 255, 255, 255));}
            //else if(times>95){pen1.setColor(QColor(255, 255, 255, 255));}

            int gray = 255*times/max_time;

            //画刷，线宽，画笔风格，画笔端点，画笔连接风格
            painter.setPen(QPen(QColor(gray, gray, gray, 255)));

            int      x = a + width;
            int      y = b + height;
            painter.drawPoint(x,y);
        }
   }
   qDebug()<<"M"; //疯狂输出 已解决，原因为没有初始化
}
```

结果如图九



图九

3. Julia 集

Julia 集与 Mandelbrot 集 的算法完全一样，只不过起始点不同

以下为 Julia 集与 Mandelbrot 集不同的部分

```
        idComplex c1(a/width*scale, b/height*scale); //Julia
//          idComplex c0(0.285,0.001);
//          idComplex c0(-0.8,0.156);
//          idComplex c0(-0.7,-0.38);
//          idComplex c0(0.45,-0.14);
            idComplex c0(0.4,0.3);
```

通过改变 idComplex 的值能得到不同的 Julia 集图形。

结果如图十

图十

4. 蕨类植物

蕨类是巴恩斯利蕨，这是通过概率分布的不同经过若干次迭代过程，实现分型，最终得到图形。

如下为绘图代码

```
//蕨类植物
if(if_fern == true)
{
    QPainter painter(this);
    QSize size = this->size();
    double width = size.width();
    double height  = size.height();
    unsigned long iter = 500000;// 迭代次数
    double x0 = 0, y0 = 0, x1, y1;
    int xp = 70;//x方向缩放参数
    int yp = 50;//y方向缩放参数
    int diceThrow;
    //timer_t t;
    //srand((unsigned)time(&t));
    //画刷，线宽，画笔风格，画笔端点，画笔连接风格
    QPen pen(Qt::darkGreen, 1.5, Qt::DotLine, Qt::RoundCap, Qt::RoundJoin);
    painter.setPen(pen);

    while (iter > 0) {
        diceThrow = rand()%100;

        if(diceThrow == 0)
```

```
        {
            x1 = 0;
            y1 = 0.16 * y0;
        }
        else if(diceThrow >= 1 && diceThrow <= 7)
        {
            x1 = -0.15 * x0 + 0.28 * y0;
            y1 = 0.26 * x0 + 0.24 * y0 + 0.44;
        }
        else if(diceThrow>=8 && diceThrow<=15){
            x1 = 0.2 * x0 - 0.26 * y0;
            y1 = 0.23 * x0 + 0.22 * y0 + 1.6;
        }

        else{
            x1 = 0.85 * x0 + 0.04 * y0;
            y1 = -0.04 * x0 + 0.85 * y0 + 1.6;
        }

        x0 = x1;
        y0 = y1;
        iter--;
        painter.drawPoint(QPoint(xp * x1 + width/3, yp * y1 + height/6));

    }

}
    }
```

结果如图十一

图十一