第14篇 (2D绘图) 涂鸦板

代码地址：https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/Painter03

一、实现涂鸦板

1．新建Qt Gui应用，项目名称为`pianter_3`，基类这次还用`QDialog`，类名保持`Dialog`不变即可。

2．到`dialog.h`文件中，先添加头文件包含：`#include <QMouseEvent>`

然后添加几个函数的声明：

```
protected:
    void paintEvent(QPaintEvent *);
    void mousePressEvent(QMouseEvent *);
    void mouseMoveEvent(QMouseEvent *);
    void mouseReleaseEvent(QMouseEvent *);
```

第一个是绘制事件处理函数，后面分别是鼠标按下、移动和释放事件的处理函数。

下面再添加几个`private`私有变量声明：

```
QPixmap pix;
QPoint lastPoint;
```

```
QPoint endPoint;
```

因为在函数里声明的`QPixmap`类对象是临时变量，不能存储以前的值，为了实现保留上次的绘画结果，我们需要将其设为全局变量。后面两个`QPoint`变量存储鼠标指针的两个坐标值，我们需要用这两个坐标值完成绘图。

2．到`dialog.cpp`文件中，先添加头文件包含：`#include <QPainter>`

然后在构造函数中添加如下初始代码：

```
resize(600, 500);    //窗口大小设置为600*500
pix = QPixmap(200, 200);
pix.fill(Qt::white);
```

下面添加几个函数的定义：

```
void Dialog::paintEvent(QPaintEvent *)
{
      QPainter pp(&pix);    // 根据鼠标指针前后两个位置就行绘制直线
      pp.drawLine(lastPoint, endPoint);    // 让前一个坐标值等于后一个坐标值，这样就能实现画出连续的线
      lastPoint = endPoint;
      QPainter painter(this);
      painter.drawPixmap(0, 0, pix);
}
```

这里使用了两个点来绘制线条，这两个点在下面的鼠标事件中获得。

```
void Dialog::mousePressEvent(QMouseEvent *event)
{
               if(event->button()==Qt::LeftButton) //鼠标左键按下
               lastPoint = event->pos();
}
```

当鼠标左键按下时获得开始点。

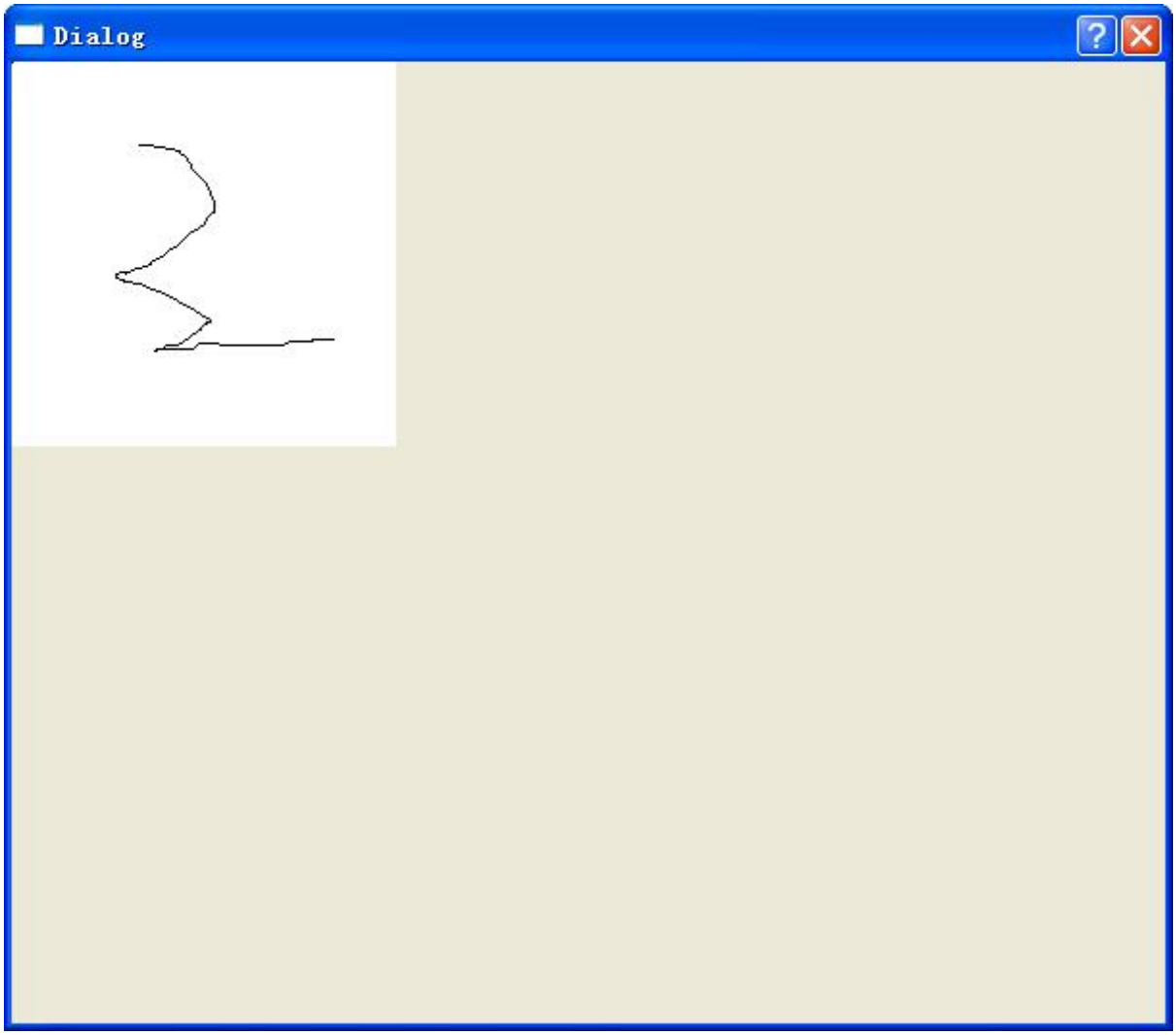
```
void Dialog::mouseMoveEvent(QMouseEvent *event)
{
         if(event->buttons()&Qt::LeftButton) //鼠标左键按下的同时移动鼠标
        {
                  endPoint = event->pos();
                  update(); //进行绘制
         }
}
```

当鼠标移动时获得结束点，并更新绘制。调用`update()`函数会执行`paintEvent()`函数进行重新绘制。

```
void Dialog::mouseReleaseEvent(QMouseEvent *event)
{
            if(event->button() == Qt::LeftButton) //鼠标左键释放
             {
                        endPoint = event->pos();
                        update();
             }
}
```

当鼠标按键释放时也进行重绘。

现在运行程序，使用鼠标在白色画布上进行绘制，发现已经实现了简单的涂鸦板功能，效果如下图所示。



二、实现放大功能

前面已经实现了简单的绘制功能，下面我们将实现放大功能，将画布放大后继续进行涂鸦。这里将使用两种方法来实现，也是对上一节坐标系统后面的问题的更进一步的应用实践。

1．添加放大按钮。到`dialog.h`文件中，先添加头文件：

```
#include <QPushButton>
```

然后添加下面`private`私有变量声明：

```
qreal scale;
QPushButton *button;
```

最后再添加一个私有槽声明：

```
private slots:
    void zoomIn();
```

2．到`dialog.cpp`文件中，先在构造函数中添加如下代码：

```
//设置初始放大倍数为1，即不放大
scale =1;
//新建按钮对象
button = new QPushButton(this);
//设置按钮显示文本
button->setText(tr("zoomIn"));
//设置按钮放置位置
button->move(500, 450);
//对按钮的单击事件和其槽函数进行关联
connect(button, SIGNAL(clicked()), this, SLOT(zoomIn()));
```

这里使用代码创建了一个按钮对象，并将其单击信号关联到了放大槽上，也就是说按下这个按钮，就会执行`zoomIn()`槽。

3．下面添加`zoomIn()`的定义：

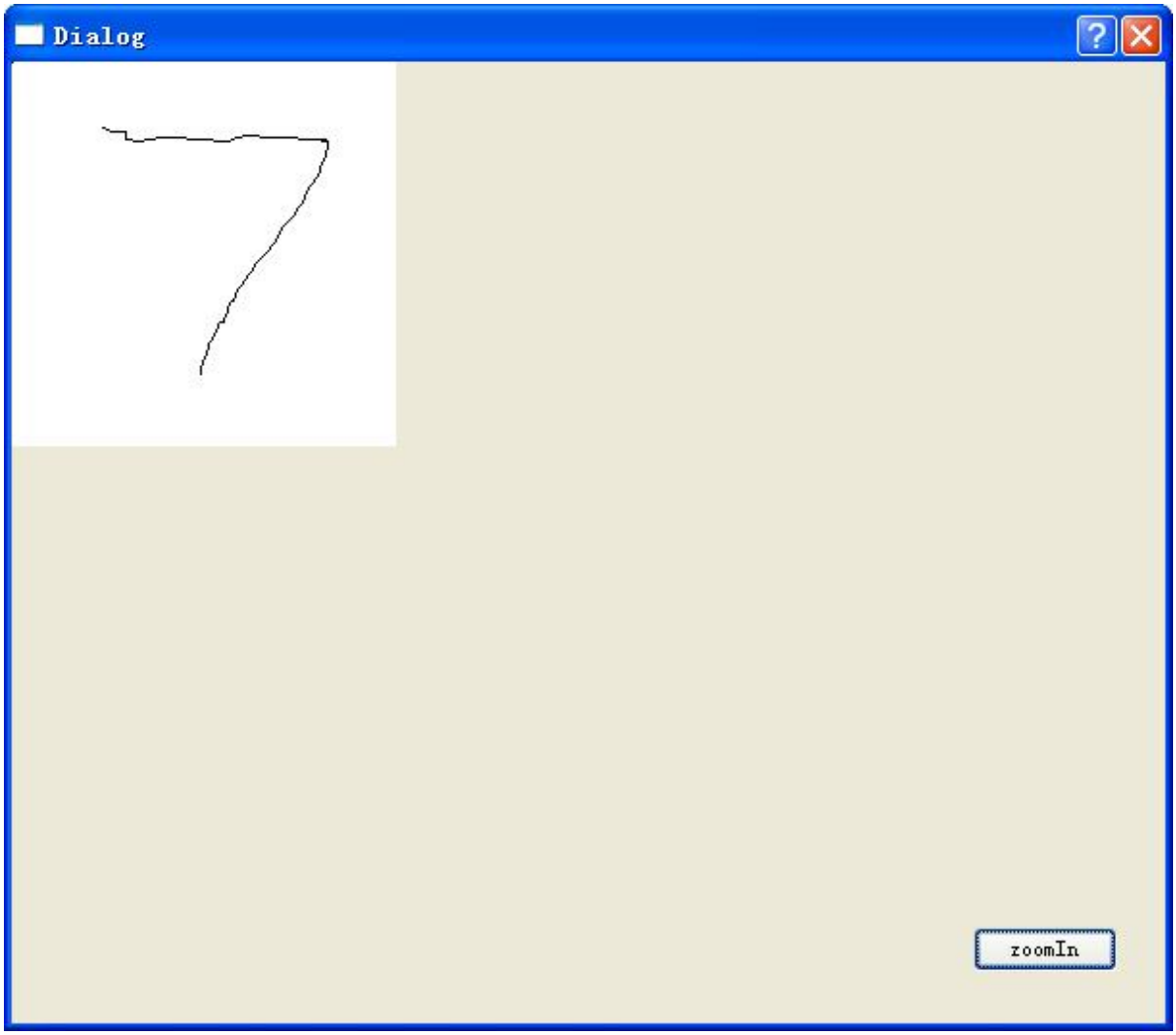
```
void Dialog::zoomIn()
{
    scale *=2;
    update();
}
```

这里我们让每按下这个按钮，放大值都扩大两倍。后面调用`update()`函数来更新显示。

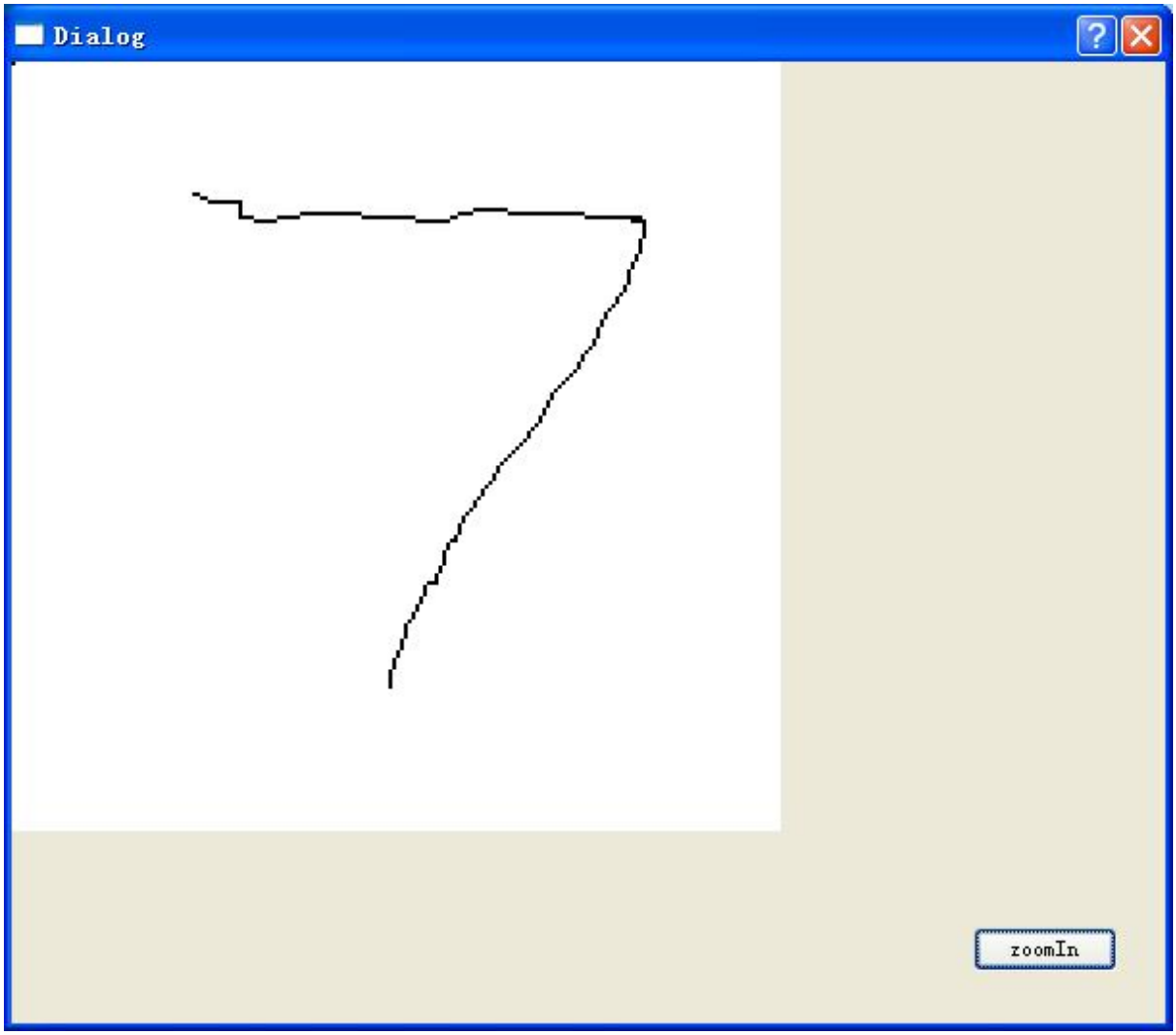
4．通过上一节的学习，我们应该已经知道想让画布的内容放大有两个办法，一个是直接放大画布的坐标系统，一个是放大窗口的坐标系统。下面我们先来放大窗口的坐标系统。更改`paintEvent()`函数如下：

```
void Dialog::paintEvent(QPaintEvent *)
{
    QPainter pp(&pix);
    pp.drawLine(lastPoint, endPoint);
    lastPoint = endPoint;
    QPainter painter(this);
    //进行放大操作
    painter.scale(scale, scale);
    painter.drawPixmap(0, 0, pix);
}
```

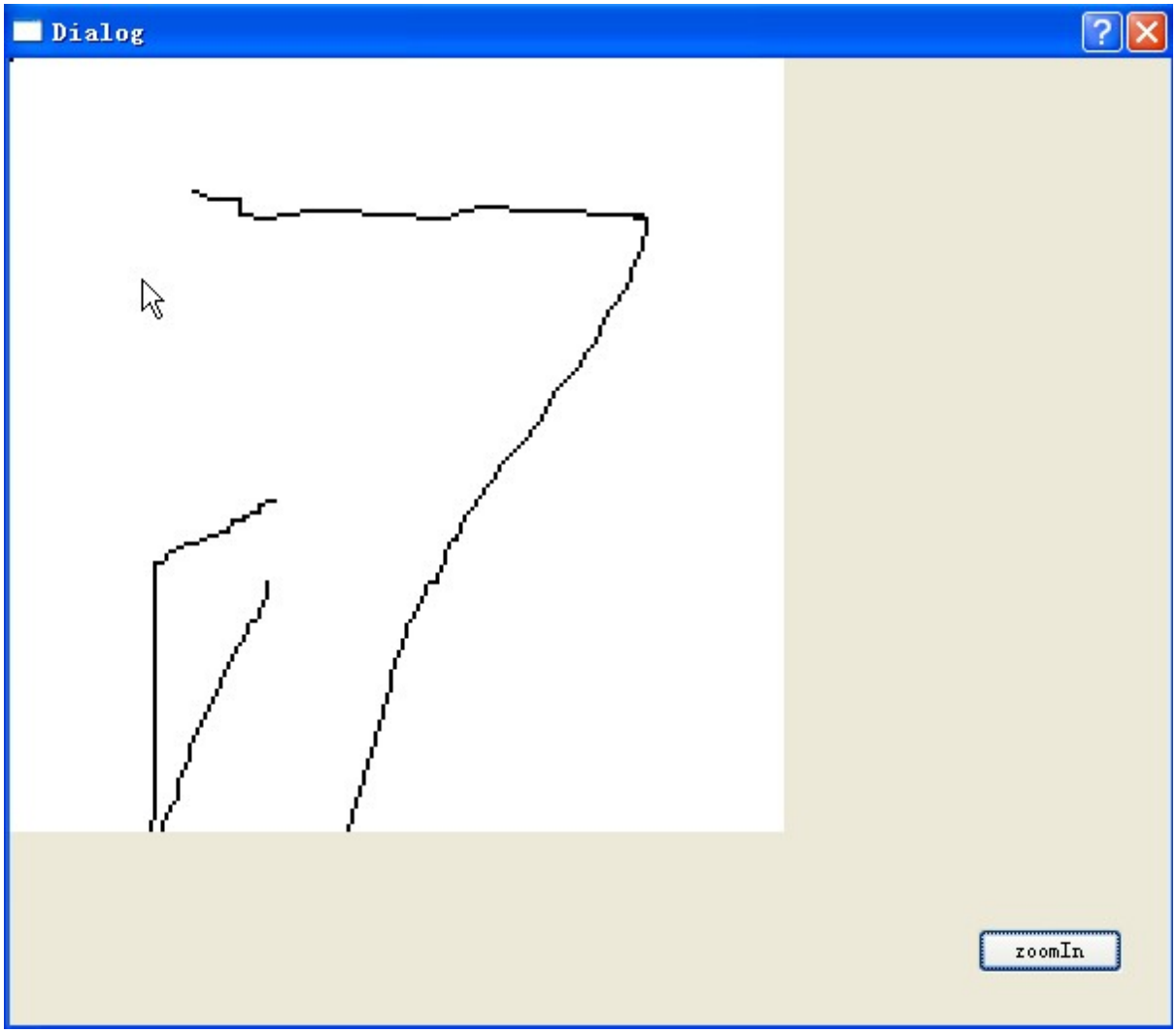
现在运行程序，先在白色画布上任意绘制一个图形，效果如下图所示。



然后按下`zoomIn`按钮，效果如下图所示。



现在再用鼠标进行绘制，发现图形已经不能和鼠标轨迹重合了，效果如下图所示。

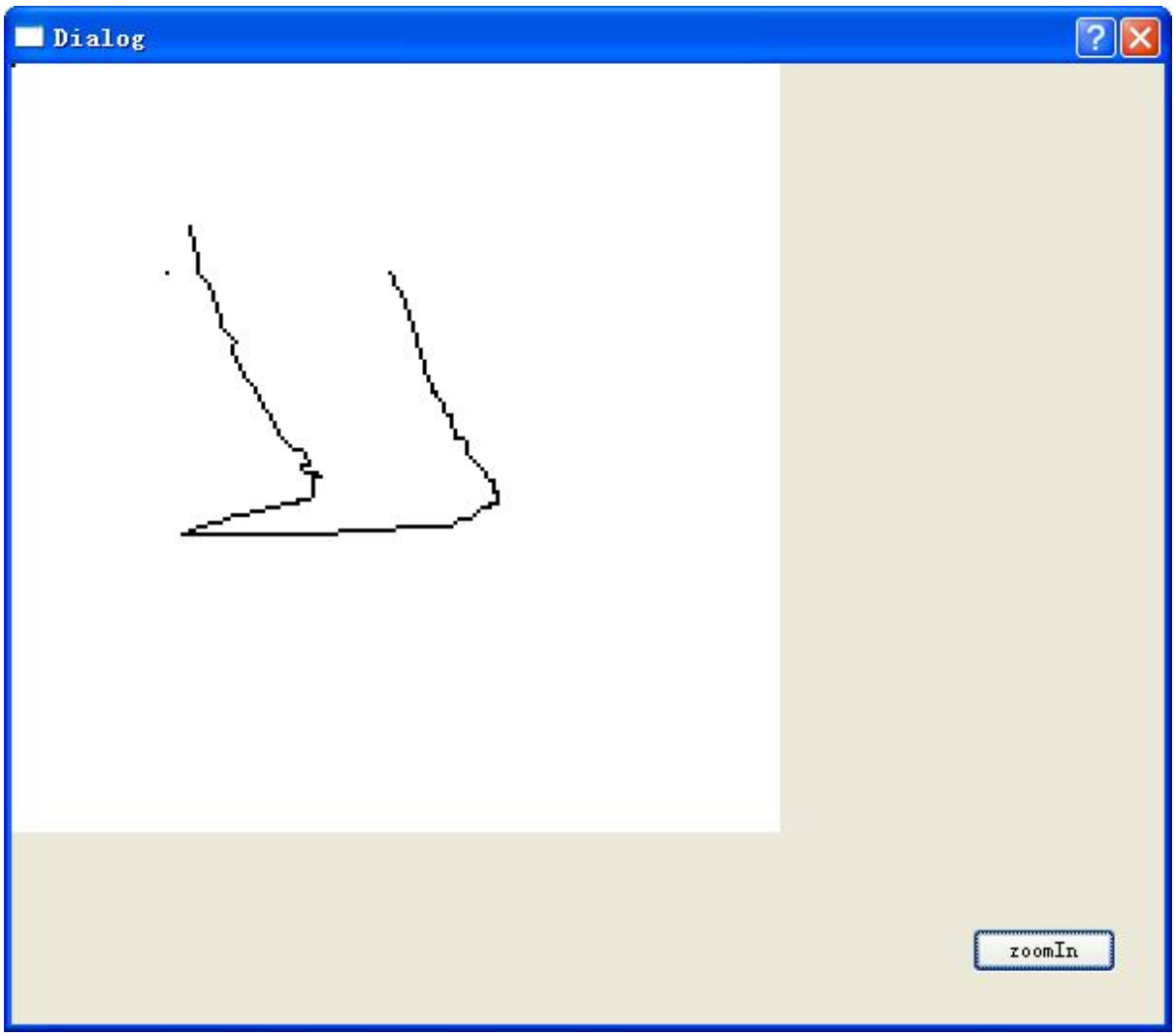


有了前面一节的知识，就不难理解出现这个问题的原因了。窗口的坐标扩大了，但是画布的坐标并没有扩大，而我们画图用的坐标值是鼠标指针的，鼠标指针又是获取的窗口的坐标值。现在窗口和画布的同一点的坐标并不相等，所以就出现了这样的问题。

其实解决办法很简单，窗口放大了多少倍，就将获得的鼠标指针的坐标值缩小多少倍就行了。我们将`paintEvent()`函数更改如下：

```
void Dialog::paintEvent(QPaintEvent *)
{
    QPainter pp(&pix);
    pp.drawLine(lastPoint/scale, endPoint/scale);
    lastPoint = endPoint;
    QPainter painter(this);
    painter.scale(scale, scale);
    painter.drawPixmap(0, 0, pix);
}
```

运行程序，效果如下图所示。可以看到，已经能够在放大以后继续绘图了。

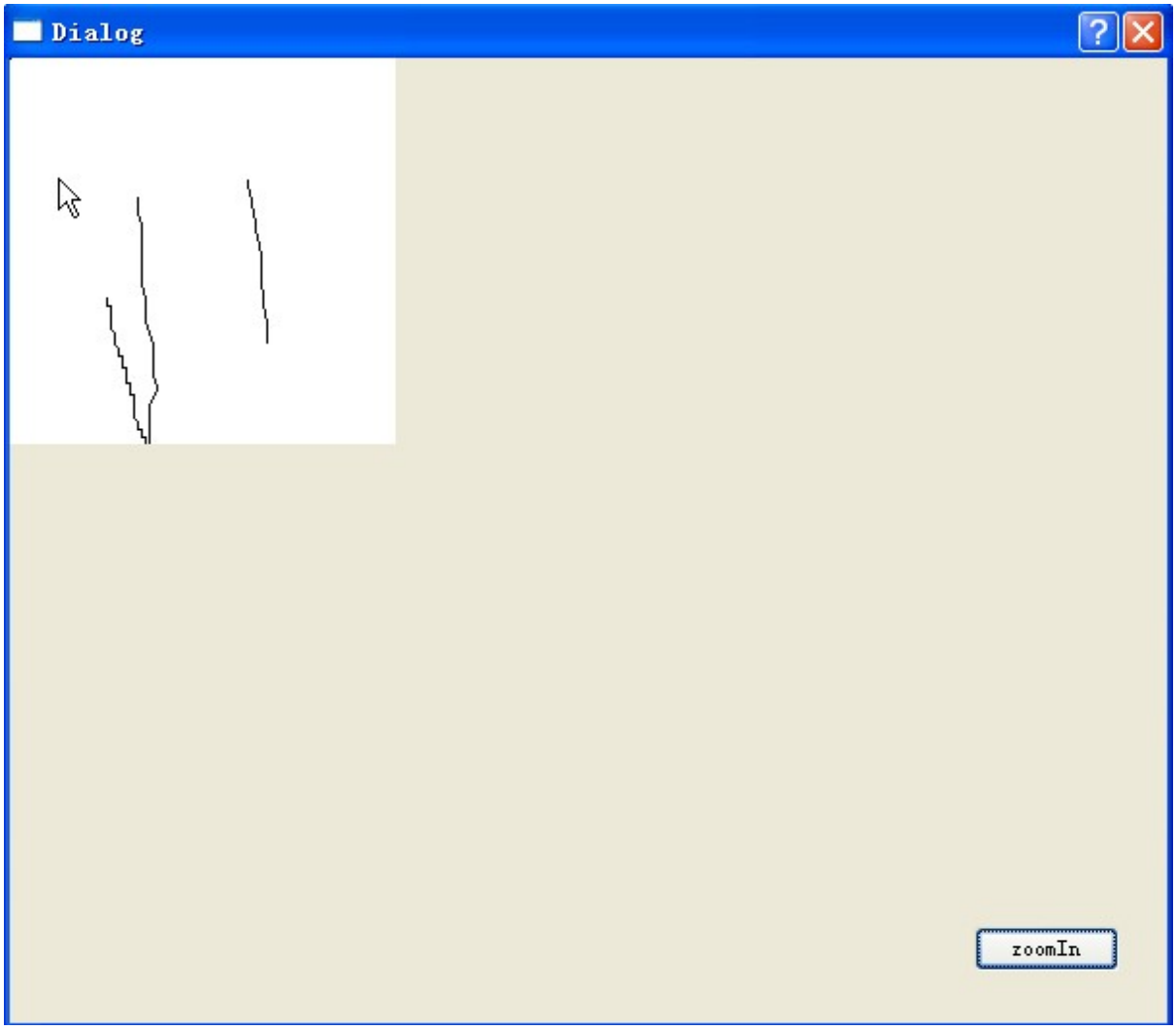


这种用改变窗口坐标大小来改变画布面积的方法，实际上是有损图片质量的。就像将一张位图放大一样，越放大越不清晰。原因就是，它的像素的个数没有变，如果将可视面积放大，那么单位面积里的像素个数就变少了，所以画质就差了。

5．方法二。扩大画布坐标系统。先将`paintEvent()`更改如下：

```
void Dialog::paintEvent(QPaintEvent *)
{
    QPainter pp(&pix);
    pp.scale(scale, scale);
    pp.drawLine(lastPoint,endPoint);
    lastPoint = endPoint;
    QPainter painter(this);
    painter.drawPixmap(0, 0, pix);
}
```

这时运行程序，先进行绘制，然后点击`zoomIn`按钮，发现以前的内容并没有放大，而当我们再次绘画时，发现鼠标指针和绘制的线条又不重合了。效果如下图所示。



这并不是我们想要的结果，为了实现按下放大按钮，画布和图形都进行放大，我们可以使用缓冲画布（就是一个辅助画布）来实现。将`paintEvent()`函数内容更改如下。

```
void Dialog::paintEvent(QPaintEvent *)
{
    if(scale!=1) //如果进行放大操作
    {
       //临时画布，大小变化了scale倍
       QPixmap copyPix(pix.size()*scale);
       QPainter pter(&copyPix);
       pter.scale(scale, scale);
       //将以前画布上的内容复制到现在的画布上
       pter.drawPixmap(0, 0, pix);
       //将放大后的内容再复制回原来的画布上
       pix = copyPix;
       //让scale重新置1
       scale =1;
```

```
    }
    QPainter pp(&pix);
    pp.scale(scale,scale);
    pp.drawLine(lastPoint/scale,endPoint/scale);
    lastPoint = endPoint;
    QPainter painter(this);
    painter.drawPixmap(0,0,pix);
}
```

现在运行程序，效果如下图所示。