第15篇 (2D绘图) 双缓冲绘图

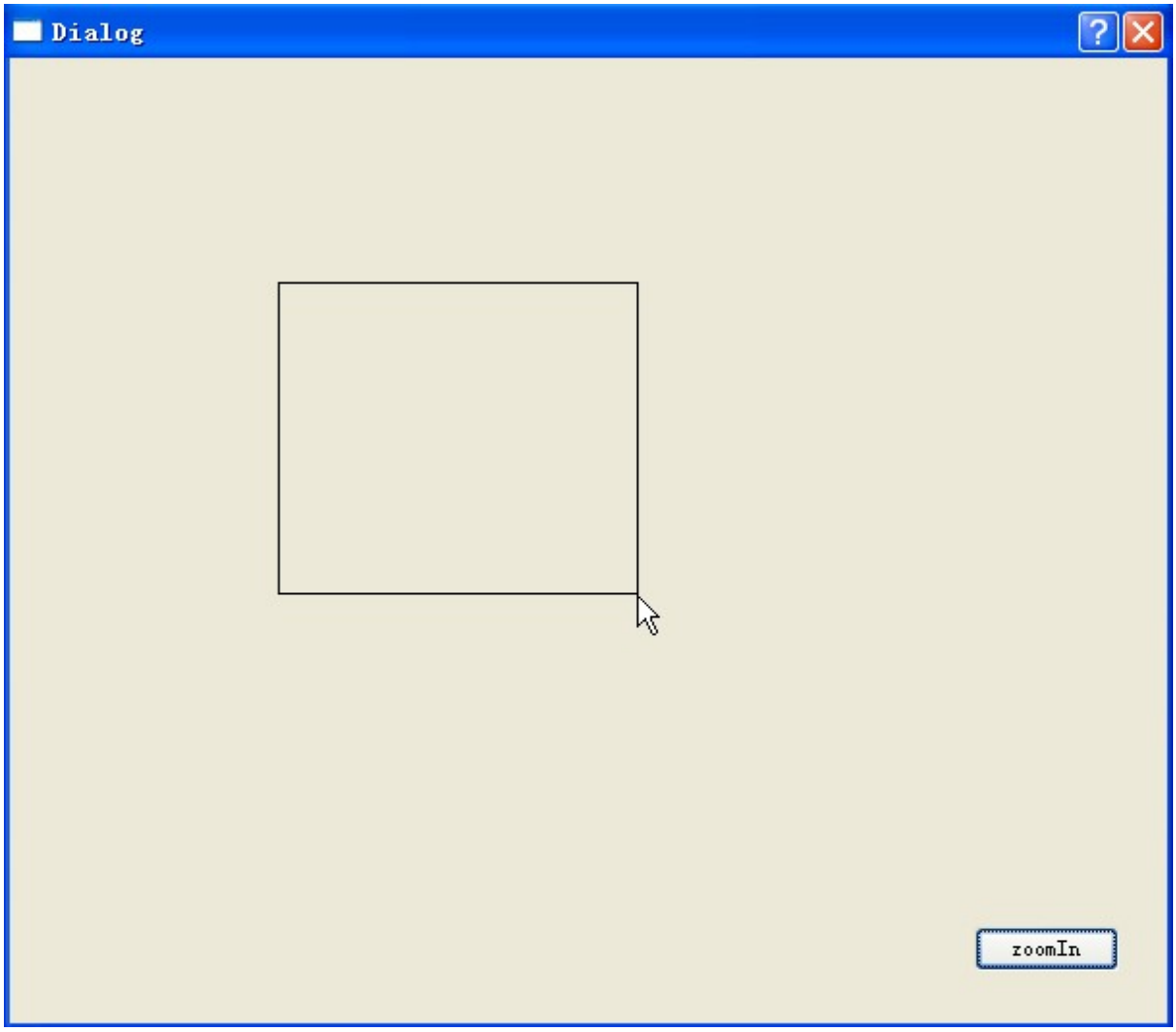
代码地址：https://github.com/Lornatang/QtStartQuicklyTutorial/tree/main/Painter03

一、绘制矩形

1．我们仍然在前面程序的基础上进行修改，先更改`painEvent()`函数：

```
void Dialog::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    int x,y,w,h;
    x = lastPoint.x();
    y = lastPoint.y();
    w = endPoint.x() - x;
    h = endPoint.y() - y;
    painter.drawRect(x, y, w, h);
}
```

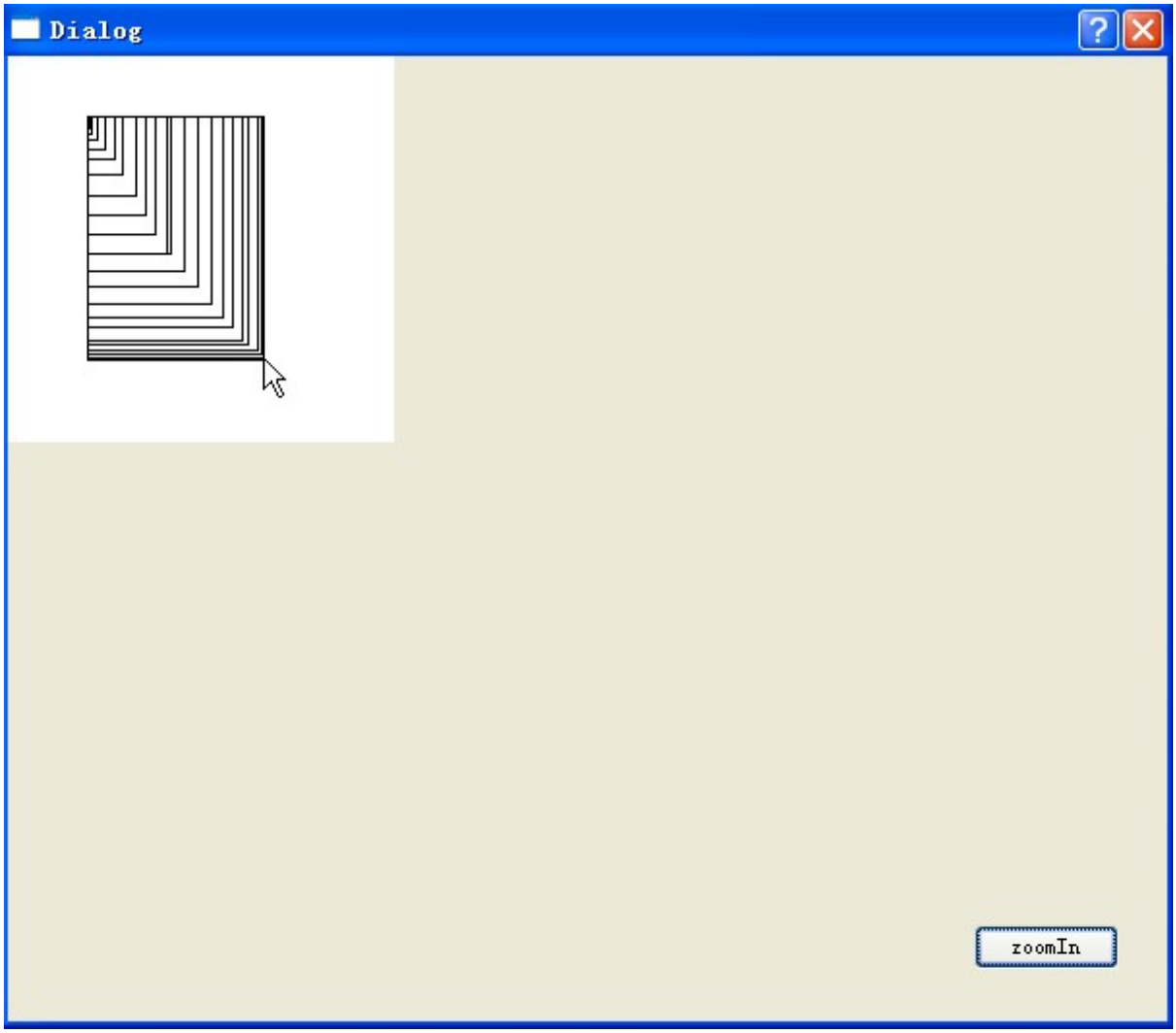
这里就是通过lastPoint和endPoint两个点来确定要绘制的矩形的起点、宽和高的。运行程序，用鼠标拖出一个矩形，效果如下图所示。



2．上面已经可以拖出一个矩形了，但是这样直接在窗口上绘图，以前画的矩形是不能保存下来的。所以我们下面加入画布，在画布上进行绘图。将`paintEvent()`函数更改如下：

```
void Dialog::paintEvent(QPaintEvent *)
{
    int x,y,w,h;
    x = lastPoint.x();
    y = lastPoint.y();
    w = endPoint.x() - x;
    h = endPoint.y() - y;
    QPainter pp(&pix);
    pp.drawRect(x, y, w, h);
    QPainter painter(this);
    painter.drawPixmap(0, 0, pix);
}
```

这里就是将图形先绘制在了画布上，然后将画布绘制到窗口上。我们运行程序，然后使用鼠标拖出一个矩形，发现出现了很多重影，效果如下图所示。



为什么会出现这种现象呢？大家可以尝试分别快速拖动鼠标和慢速拖动鼠标来绘制矩形，结果会发现，拖动速度越快，重影越少。其实，在我们拖动鼠标的过程中，屏幕已经刷新了很多次，也可以理解为`paintEvent()`函数执行了多次，每执行一次就会绘制一个矩形。知道了原因，就有方法来避免这个问题发生了。

二、双缓冲绘图

1．我们再添加一个辅助画布，如果正在绘图，也就是鼠标按键还没有释放的时候，就在这个辅助画布上绘图，只有当鼠标按键释放的时候，才在真正的画布上绘图。

首先在`dialog.h`文件中添加两个私有变量：

```
QPixmap tempPix; //辅助画布
bool isDrawing;   //标志是否正在绘图
```

然后到`dialog.cpp`的构造函数中对变量进行初始化：

```
isDrawing = false;
```

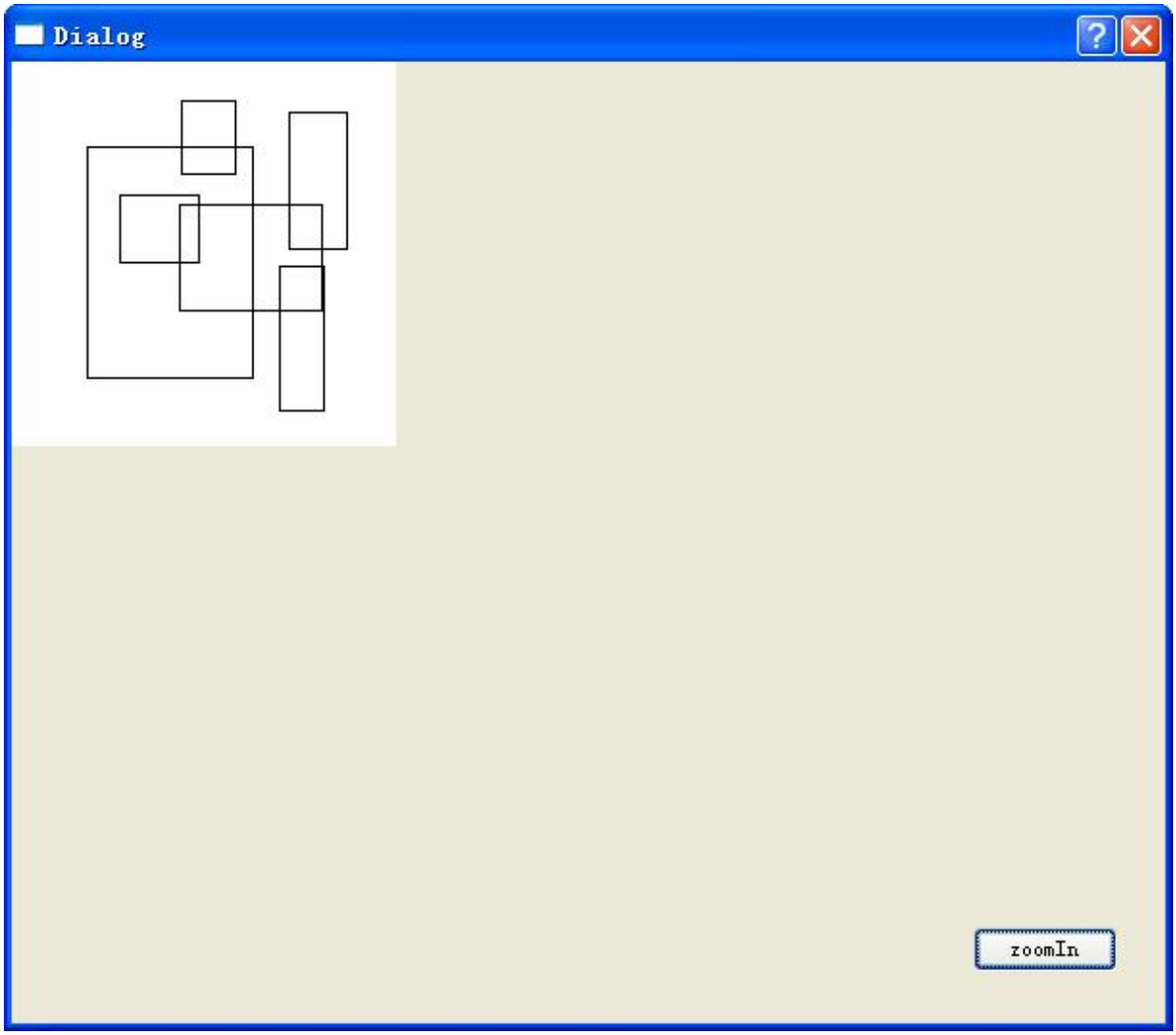
下面再更改`paintEvent()`函数：

```
void Dialog::paintEvent(QPaintEvent *)
{
    int x,y,w,h;
    x = lastPoint.x();
    y = lastPoint.y();
    w = endPoint.x() - x;
    h = endPoint.y() - y;
    QPainter painter(this);
    if(isDrawing) //如果正在绘图，就在辅助画布上绘制
    {
       //将以前pix中的内容复制到tempPix中，保证以前的内容不消失
       tempPix = pix;
       QPainter pp(&tempPix);
       pp.drawRect(x,y,w,h);
       painter.drawPixmap(0, 0, tempPix);
    } else {
       QPainter pp(&pix);
       pp.drawRect(x,y,w,h);
       painter.drawPixmap(0,0,pix);
    }
}
```

下面还需要更改鼠标按下事件处理函数和鼠标释放事件处理函数的内容：

```
void Dialog::mousePressEvent(QMouseEvent *event)
{
    if(event->button()==Qt::LeftButton) //鼠标左键按下
    {
       lastPoint = event->pos();
       isDrawing = true;   //正在绘图
    }
}
void Dialog::mouseReleaseEvent(QMouseEvent *event)
{
    if(event->button() == Qt::LeftButton) //鼠标左键释放
    {
       endPoint = event->pos();
       isDrawing = false;    //结束绘图
       update();
    }
}
```

当鼠标左键按下时我们开始标记正在绘图，当按键释放时我们取消正在绘图的标记。现在运行程序，已经可以实现正常的绘图了。效果如下图所示。



2．双缓冲绘图

根据这个例子所使用的技巧，我们引出所谓的双缓冲绘图的概念。双缓冲（double-buffers）绘图，就是在进行绘制时，先将所有内容都绘制到一个绘图设备（如`QPixmap`）上，然后再将整个图像绘制到部件上显示出来。使用双缓冲绘图可以避免显示时的闪烁现象。从Qt 4.0开始，`QWidget`部件的所有绘制都自动使用了双缓冲，所以一般没有必要在`paintEvent()`函数中使用双缓冲代码来避免闪烁。

虽然在一般的绘图中无需手动使用双缓冲绘图，不过要想实现一些绘图效果，还是要借助于双缓冲的概念。比如这个程序里，我们要实现使用鼠标在界面上绘制一个任意大小的矩形。这里需要两张画布，它们都是`QPixmap`实例，其中一个`tempPix`用来作为临时缓冲区，当鼠标正在拖动矩形进行绘制时，将内容先绘制到`tempPix`上，然后将`tempPix`绘制到界面上；而另一个`pix`作为缓冲区，用来保存已经完成的绘制。当松开鼠标完成矩形的绘制后，则将`tempPix`的内容复制到pix上。为了绘制时不显示拖影，而且保证以前绘制的内容不消失，那么在移动鼠标过程中，每绘制一次，都要在绘制这个矩形的原来的图像上进行绘制，所以需要在每次绘制`tempPix`之前，先将pix的内

容复制到`tempPix`上。因为这里有两个`QPixmap`对象，也可以说有两个缓冲区，所以称之为双缓冲绘图。