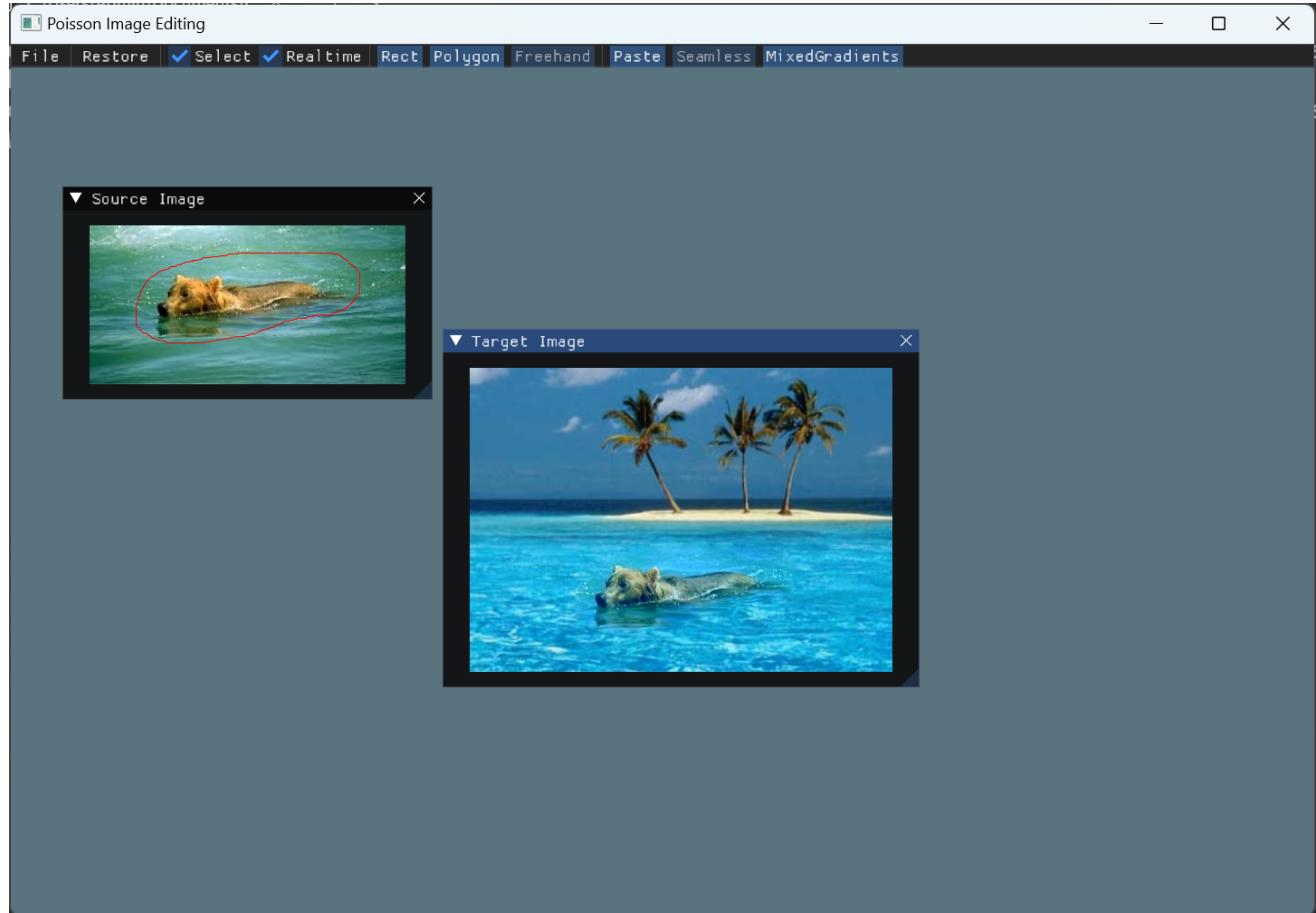


Homework 3 作业报告 by 76-朱雨田

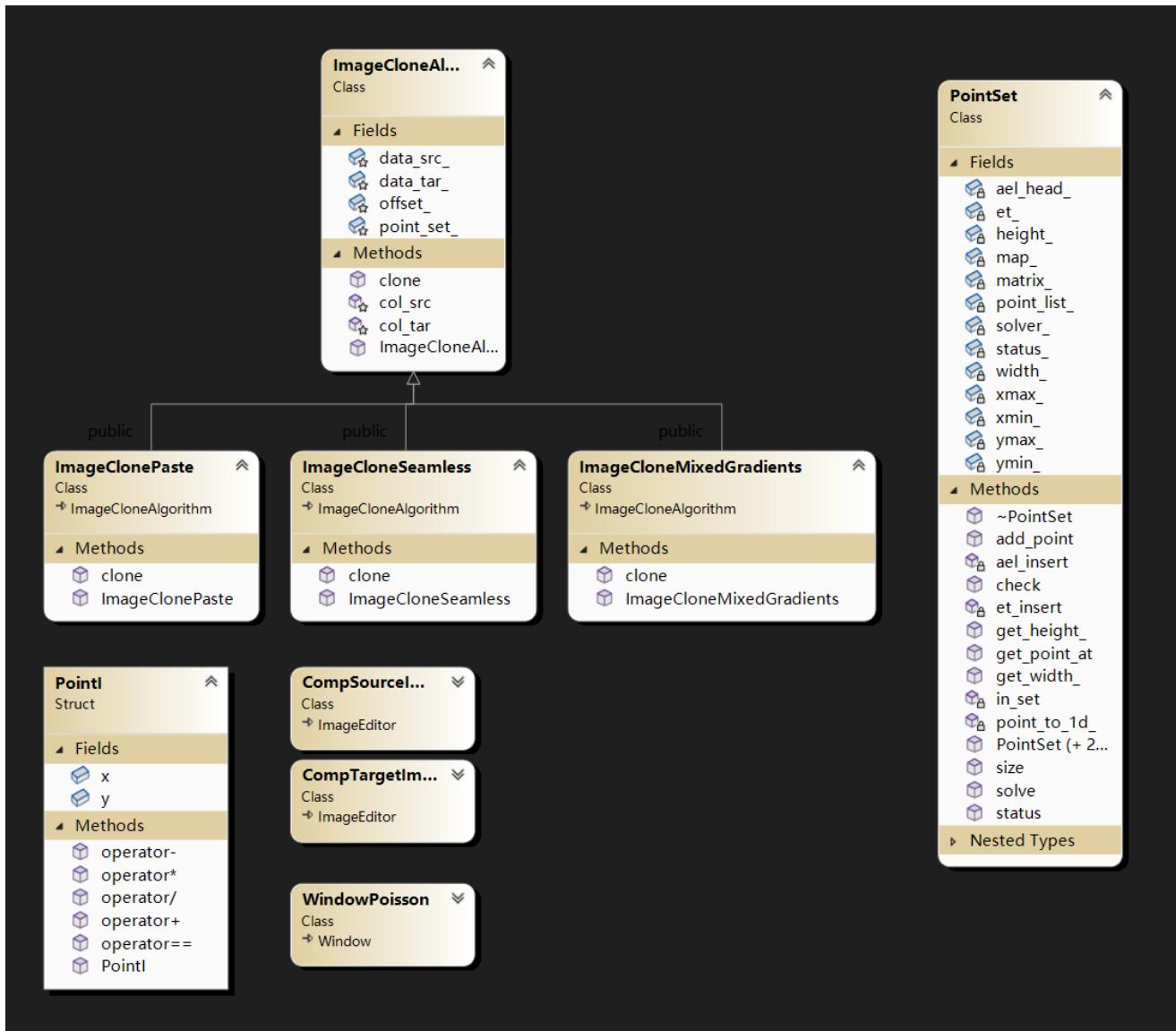
本次作业要求用求解泊松方程实现图像的无缝融合，并实现光栅化中的多边形扫描线算法。

我在本次作业中实现了一般泊松方程和梯度混合泊松方程两种图像融合算法，将其封装成了类的形式，并用有序边表算法实现了多边形的扫描线光栅化算法（封装在了整数坐标点集类中）。

效果展示如下：



本次作业的类图如下：



1. 泊松图像编辑

泊松图像编辑的目的是在保留源图像的纹理同时，让复制后的结果能与目标图像的边界颜色平滑过渡。

该算法同时实现以上两个目的的思路是：列泊松方程使得源图像的二阶导数不变（纹理不变），并为泊松方程设置 Dirichlet 边界条件使得与目标图像边界平滑过渡。

即：求解融合后的图像，满足以下条件：

$$\Delta f = \Delta g, f|_{\delta\Omega} = f^*|_{\delta\Omega}$$

其中 f 是融合后的图像（待求解量）， f^* 是目标图像， g 是源图像， $\delta\Omega$ 是多边形的边界。

为了求解该泊松方程，我们研究它的离散形式。注意到 Laplacian 算子有三种含义：多变量函数的二阶导数，梯度的散度，邻域平均值与中心点的差。第三种含义可以很好地应用在图像上，即：

$$(\Delta f)_{i,i} = f_{i-1,i} + f_{i+1,i} + f_{i,i-1} + f_{i,i+1} - 4f_{i,i}$$

我们将多边形中的每个点进行编号，就能将方程列出，并分别对三个颜色分量求解了。其中共有 n 个方程， n 个未知量：

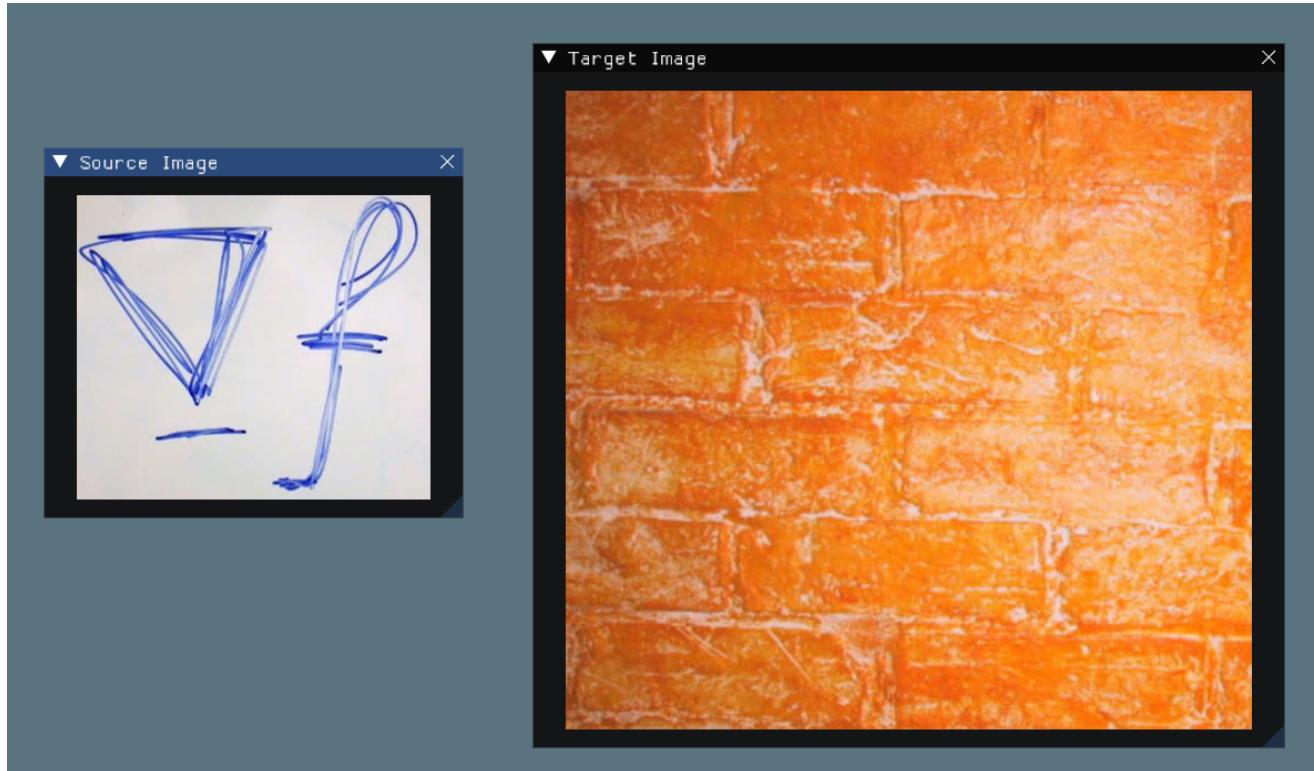
$$\begin{cases} (\Delta f)_i = (\Delta g)_i, & i \notin \delta\Omega \\ f_i = f_i^*, & i \in \delta\Omega \end{cases}$$

注意到将方程写为矩阵的形式后，系数矩阵**仅与多边形的形状有关**，而该算法的时间瓶颈就在矩阵的分解上。因此，我们可以在多边形选择完毕后进行一次矩阵预分解，这样在目标图像上就可以做到实时拖动多边形预览效果了。由于该矩阵是不对称的稀疏矩阵，我们采取 LU 分解。

效果如下（可做到实时拖动）：



观察以上方程，可以注意到该算法的一个问题是，目标图像的纹理没有参与任何计算。这会导致图像复制后目标图像的纹理明显损失，可能会造成不理想的结果，如下图：





可以注意到背景的砖纹变得模糊，这是明显不自然的融合结果。要保留目标图像的纹理，当然要让目标图像的信息也参与进方程。论文提出了一种 MixedGradients 算法改进，即：

比较目标图像与源图像在同一点处的梯度，选取梯度更大者参与方程求解。方程形式如下：

$$\begin{cases} (\Delta f)_i = (\Delta g)_i, & i \notin \delta\Omega, |\nabla g_i| \geq |\nabla f_i^*| \\ (\Delta f)_i = (\Delta f^*)_i, & i \notin \delta\Omega, |\nabla g_i| < |\nabla f_i^*| \\ f_i = f_i^*, & i \in \delta\Omega \end{cases}$$

注意一些编程容易出错的地方，如比较的是梯度，而不是 Laplacian；梯度有多个分量，应比较各个分量合成向量的模长后再决定应用哪个 Laplacian，而不是在比较的同时就进行 Laplacian 的计算。

效果如下，可以注意到砖块的纹理得到了很好的保留：



还有一个值得注意的问题是，当多边形在源图像或目标图像出界时，会报错或得到不正确的融合结果。这里在源图像的多边形绘制中限制了多边形顶点的坐标范围，且在粘贴算法基类中封装了对应坐标颜色获取的函数，对出界范围内的颜色也做出了定义，即最近的边界处的颜色。效果如下：



参考文章：[图像处理基础（九）泊松图像编辑 Possion Image Editing - 知乎\(zhihu.com\)](#)

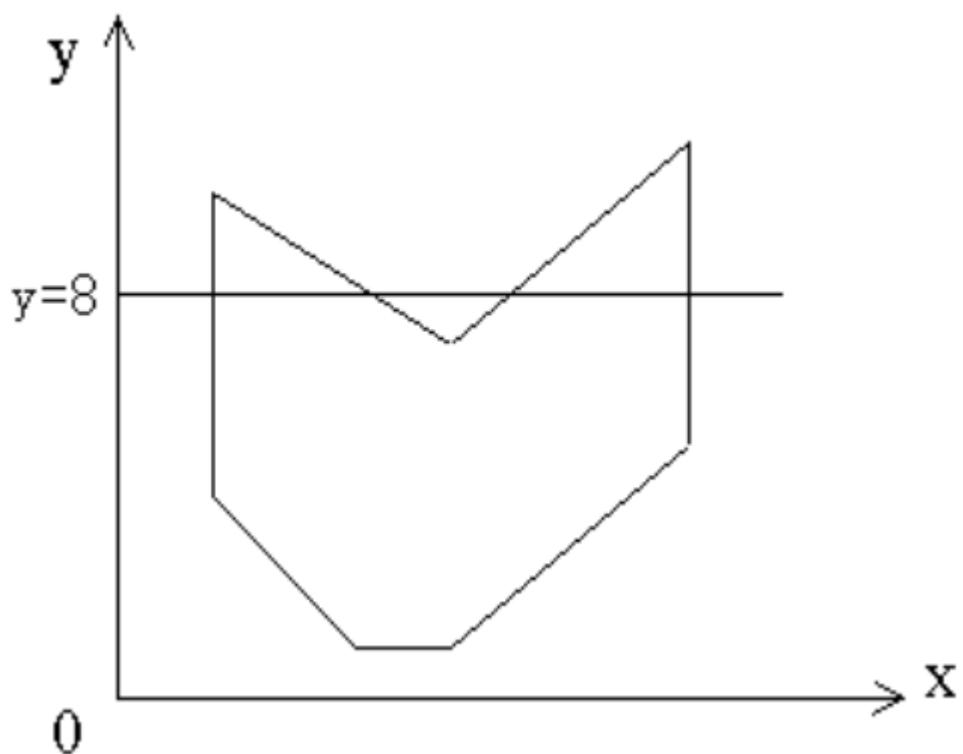
该文章对论文中的算法做出了很好的中文解释，并且扩展引申了不少新的应用。

2. 扫描线算法

要完成多边形选区的交互功能，只需要将 Homework 1 中的多边形绘制应用到这次的框架中即可。

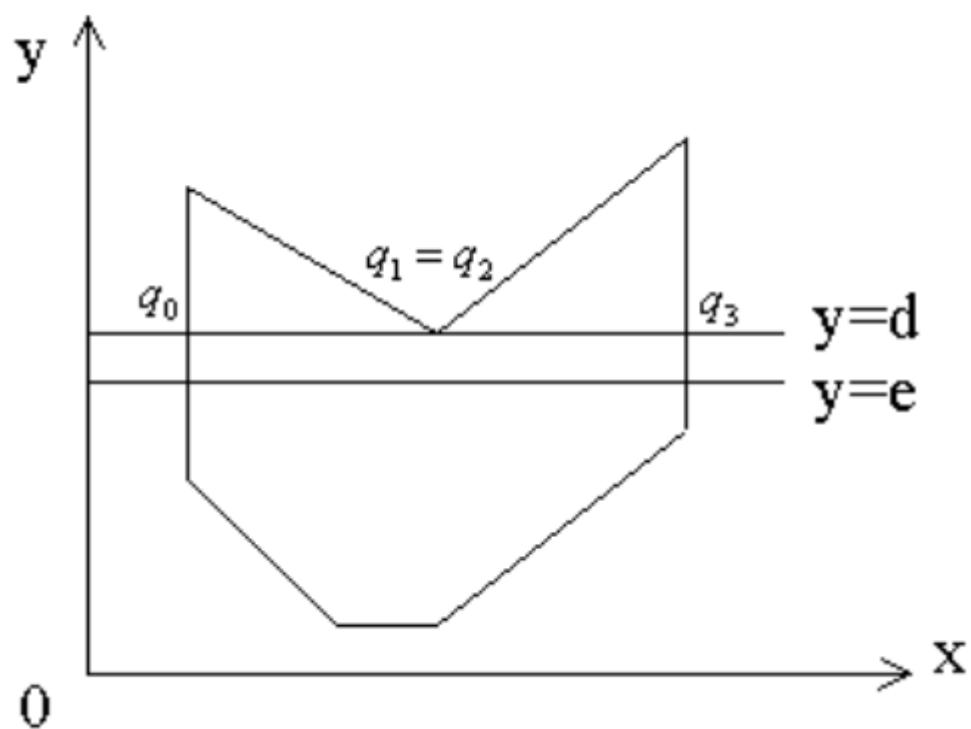
确定了多边形后，还需要确定**哪些像素是在多边形中的**，才能实现泊松方程的计算。这实际上和图形学渲染管线中的**光栅化**步骤是相同的。（第一次接触光栅化时的感觉时：“这也需要算？？”）

为了很好地处理各类多边形（凸多边形、凹多边形、非简单多边形...），我们采用扫描线的算法来确定哪些点在多边形的内部，如下图：



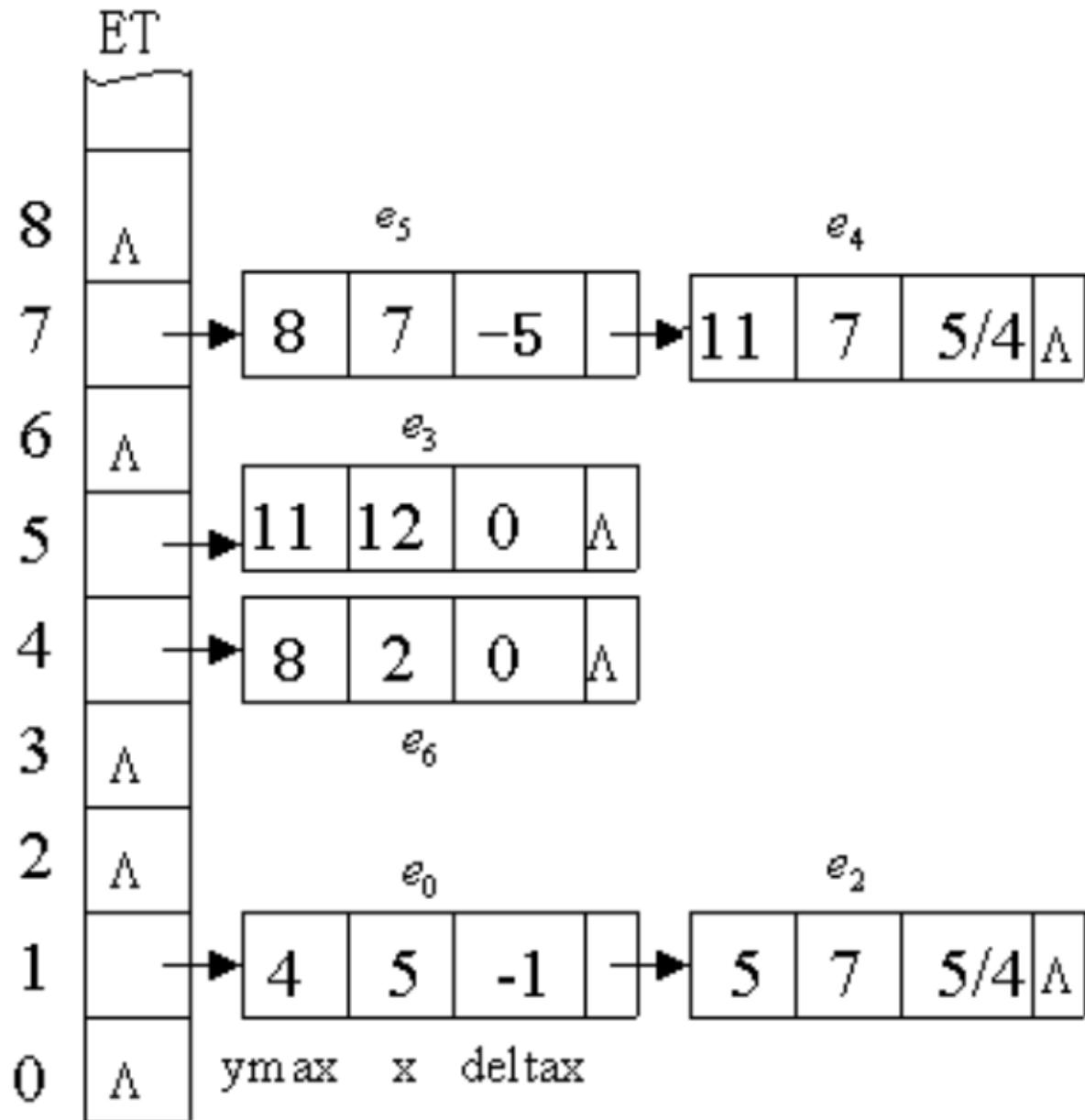
对每个纵坐标发射一条扫描线，初始时不填充；每与一条线相交，就将填充状态更改一次。这样，填充状态为 `true` 时经过的点，就是多边形中的点。

朴素的算法是对于每个 y 坐标都枚举一遍所有边并求交。我在第一次完成该作业时采用的就是朴素算法。这是一个 $O(n^2)$ 的算法，处理 Freehand 生成的多边形时在效率上并不理想，且会遇到与多边形顶点相交的特殊情况：

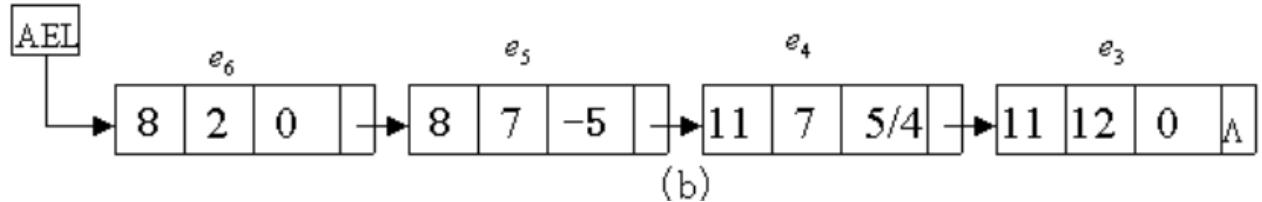
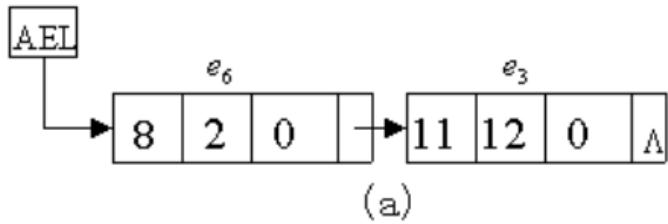


为了解决以上问题，我们忽略平行边，规定每条边下闭上开，并用有序边表的数据结构对多y边形进行维护。该算法维护两个系列链表，一个是对每个 y 坐标存边的 ET（分类表），一个是扫描过程中持续维护的有序的 AEL（活性边表）。每条边维护四个信息：int ymax (上端点的 y 坐标)，float x (下端点的x坐标)，float dx (斜率的倒数，便于计算且规避了竖直线条斜率不存在的情况)，ETElement* nxt (指向链表下一元素的指针)。

ET 表如下图所示，对每个 y 坐标存储一个指针，指向 y 端点为该坐标的边组成的链表头指针。

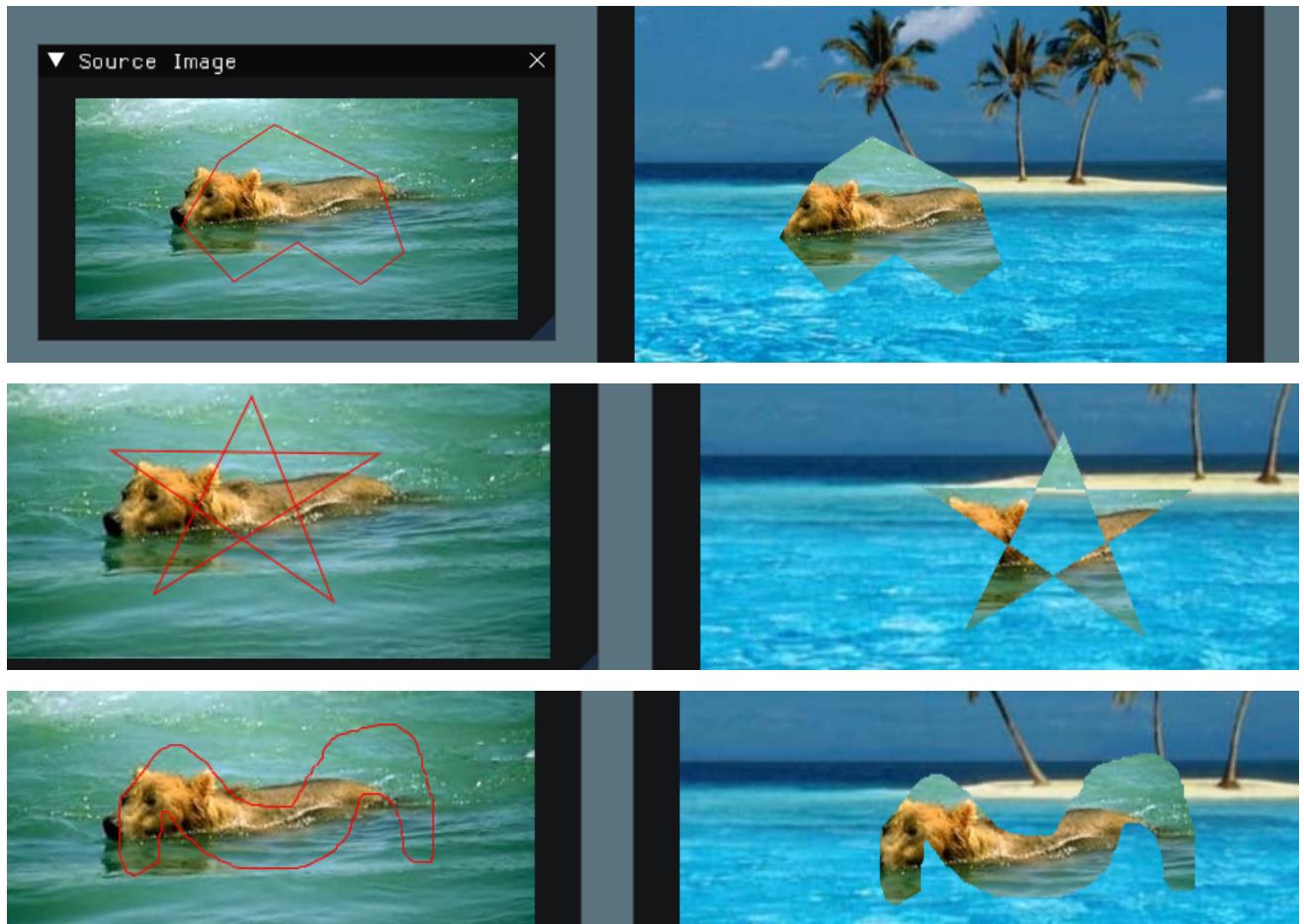


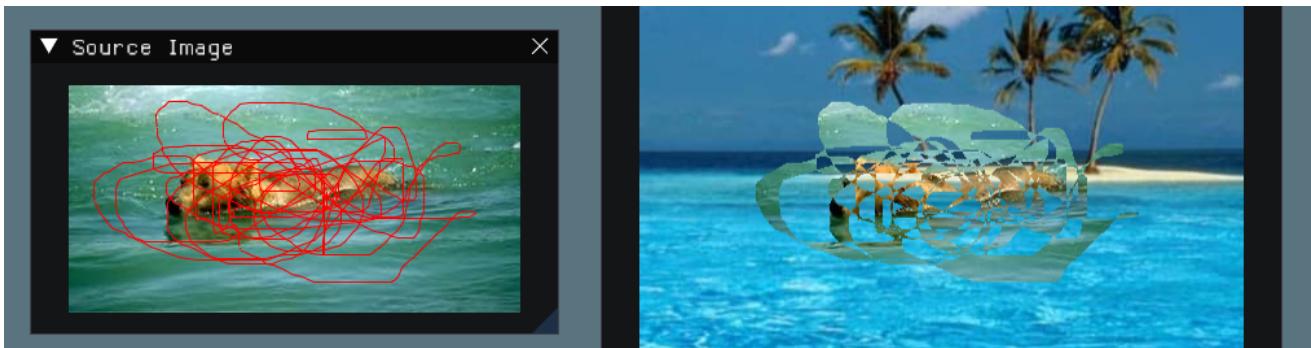
AEL 表存储与当前扫描线相交的边，并以 x 分量为比较依据保持有序（不要再想平衡树了！这个表的数据规模大部分时候都是0, 2和4, 每次更新时都进行一次插入排序就行）。



算法流程如下：每次将 y 更新自增 1 后，都删去 AEL 中符合 $\text{ymax} == y$ 的边（不再相交），将所有边的 x 坐标加上 dx （更新交点横坐标），再访问当前 y 坐标的 ET 表，加入新边，并重新对整个 AEL 进行排序（非简单多边形中，就算 AEL 不引入或删除新元素，边的顺序仍然有可能发生变化），扫描一次加入新的坐标点。

测试结果如下：





可以看到对非简单多边形也能做出很好的判断。

这里为了实现光栅化与矩阵预分解，我写了一个 `Pointset` 类，在构造函数中传入多边形的顶点列表，并在构造函数中完成了光栅化与矩阵预分解。粘贴时，只需要向对应的粘贴算法类传入点集指针，就能在粘贴算法类中调用对应的点集与预分解矩阵。

参考文章：[Microsoft PowerPoint - chp7-2.ppt \(ustc.edu.cn\)](http://ustc.edu.cn/)

本次报告关于算法的图片全部来自该 PPT 。

心得体会

本次作业在第一次完成的时候就已经完成了 MixedGradients、朴素光栅化等 optional 内容。这次加以实现了有序边表的光栅化，并将图像融合算法也封装成了类的形式（一周目是面向过程的）。

一周目时怎么也看不懂为什么“梯度的散度”能反映图像的纹理性质。后来看到了“邻域平均值减去中心值”的 Laplacian 算子描述，才将几个概念联系起来，豁然开朗，想明白代码该怎么写了。后面的数学比这里难得多，但我差不多在这里弄明白该怎么学习图形学中的数学概念了。

二周目的时候写代码犯了很多低级错误。感觉大部分时间都在调低级错误产生的 bug 上，果然还是需要先想清楚再去写啊。