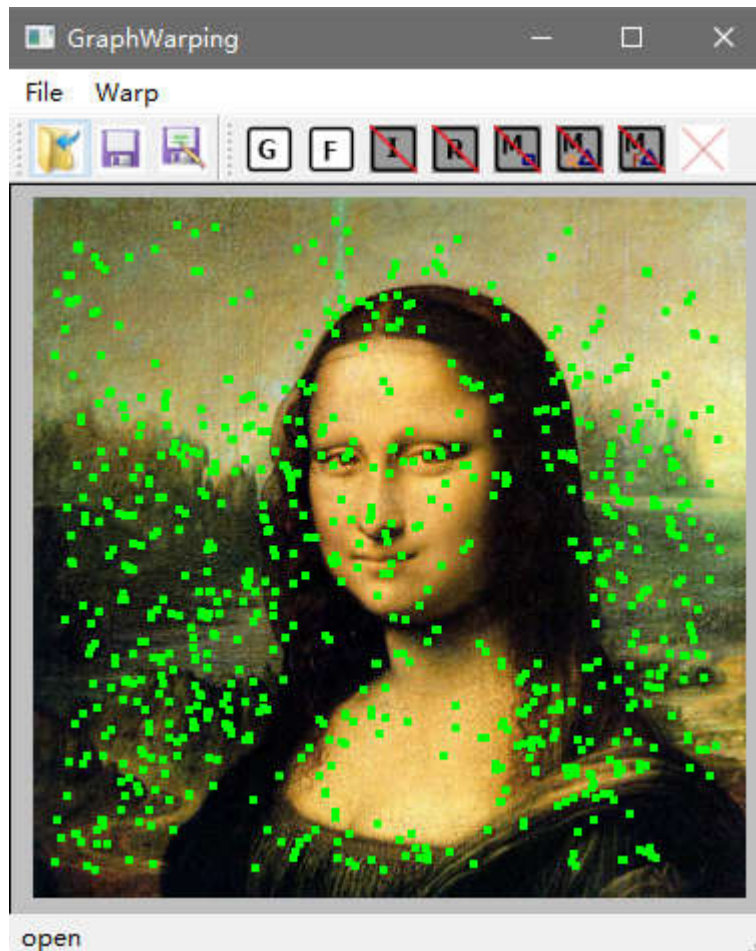


作业 03 Mesh based Image Warping

庄涛 PB15111679, ID : 85
计算机科学与技术系, 215 院 011 系 01 班
2018/3/18

一、要求

- 1) 实现基于**四边形网格**和**三角形网格**的 Image warping 算法。
- 2) 学习使用 **OpenCV 图像库** (见下面说明) ;
- 3) 将作业 2 中的 3 个 warping 类 (几乎不用改动) 移植到 OpenCV 的框架中 ;
- 4) 实现扫描线算法 :对于**三角形 (或任意多边形边界)** 的封闭区域, 要得到它的内部像素, 需要使用**扫描线算法**。
- 5) 使用**重心坐标**插值多边形或三角形内部的像素。
- 6) 继续巩固 OOP 编程思想。



二、环境

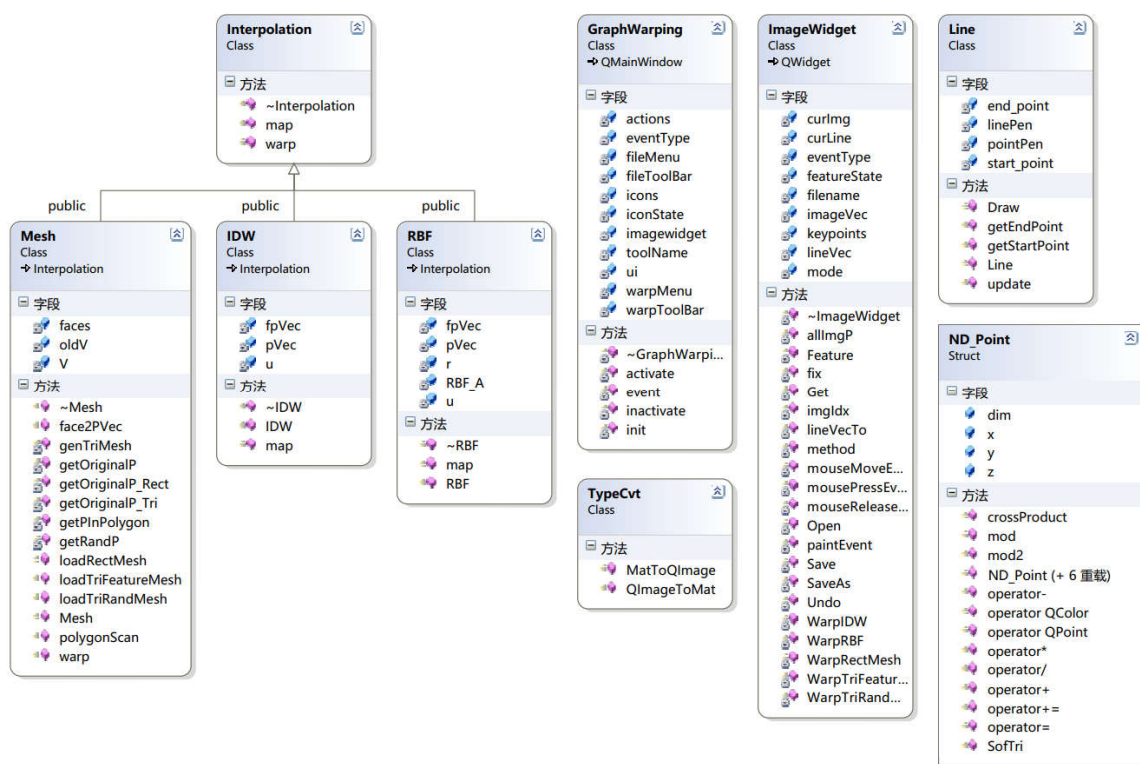
系统 : windows10 x64

IDE : Visual Studio 2010-debug-win32

外部依赖库 : QT-5.5, eigen-3.3.2, OpenCV-2.4.8-vs2010-x86

三、过程

3.1 类关系图



3.2 采用统一的数据结构 ND_Point

为了加速运算和代码复用性，并没有去使用 QT 提供的 QPoint/QColor，而是用统一的结构 ND_Point 来表示二维（对应 QPoint）或者三维（对应 QColor）的数据。为了实现这点，对各种函数点的逻辑进行了大量的简化修改，还给 ND_Point 重载了各种操作符，使得其能在各处无缝衔接，另外操作符的实现使用了性能最好的形式，省略了各种关于维度等的检查。最终简化了各种函数接口的实现，实现了数据结构的统一。

3.3 各类 Warp 的接口设计

根据老师的说法，IDW 和 RBF 继承于 Warp，然后再各自实现自己的 warp 算法。然而通过观察发现，各类的 warp 算法是有共性的，即都要映射一个点集（warp）。不同的只是对一个点的具体映射算法（map）以及需要映射的点集（points）。故将映射点集（warp）和映射算法（map）作为虚函数放到类 Interpolation 中，而 IDW 和 RBF 只需简单修改 map 函数，不过 mesh 要修改 warp。

3.4 Icon 的动态显示

为了提高程序的鲁棒性，需要对一些不合理的操作进行限制。比如，在未打开文件前不允许进行 save, saveas, get, IDW, RBF 等操作；还有，在没有数据输入之前，不允许进行 IDW, RBF 等操作。简单的实现方法可以是在相应的函数里边进行判断，如果不合理就不做出反应或者用一定的方式提醒用户其为错误操作，如弹出对话框。

但这些我认为不够好，我的想法是动态的改变 action 的 icon 和连接性。当操作可行时，相应的 icon 正常显示，当操作不可行时，相应的 icon 显示禁止状态，用户即使点击也没有反应。如下



在 ImageWidget 中可以通过 `parent()->findChild<QToolBar*>("name")` 来获取到 GraphWarping 相应的工具栏，再对其中的 action 进行改变 icon 和连接性。

但是我在 GraphWarping 中使用了 `activate(int id)` 和 `inactivate(int id)` 来统一管理 action。而 `parent()` 是 `QObject *`，不能使用 GraphWarping 的函数。而且又不能将其强制类型转换为 `GraphWarping *` 从而调用上述函数，因为 GraphWarping 的定义中使用了 `ImageWidget`，如果我们还打算在 `ImageWidget` 的定义中使用 GraphWarping，就会引入循环定义问题。

解决的办法是使用 `QObject` 提供的一个泛用的虚函数接口 `virtual bool event (QEvent *)`，在 GraphWarping 中重定义该函数来调用 `activate` 和 `inactive`，而在 `ImageWidget` 中使用 `parent()->event(&QEvent(...))` 来触发这个事件。

```
bool QObject::event(QEvent *e) [virtual]
```

This virtual function receives events to an object and should return true if the event *e* was recognized and processed

The event() function can be reimplemented to customize the behavior of an object.

Make sure you call the parent event class implementation for all the events you did not handle.

3.5 OpenCV 架构的引入

原本的代码图像的结构使用的是 QImage，这次实验涉及了 OpenCV，在 OpenCV 中，数据的结构为 Mat。老师的建议是将程序的架构完全转移成 OpenCV。但我认为其实并没有必要，比较合理的做法是设计一个专门用来类型转换的工具类 TypeCvt，调用其中的静态方法就可以进行 QImage 与 Mat 的相互转换，这样能够提升代码的质量。



3.6 Mesh 算法

Mesh 的主要流程可以分为以下 3 个步骤

1. **生成网格**：四边形网格（规则撒点）loadRectMesh，三角形网格（随机撒点）loadTriRandMesh，特征撒点 loadTriFeatureMesh，三角化 genTriMesh)
2. **网格顶点映射**：调用 Interpolation 的 warp 即可
3. **网格内点映射**：扫描线算法 polygonScan，根据重心坐标获取原位置 getOriginalP

生成三角网格的方法使用了 triangle 的函数 triangulate。

生成特征点的方法使用了 OpenCV 的函数 SurfFeatureDetector::detect。

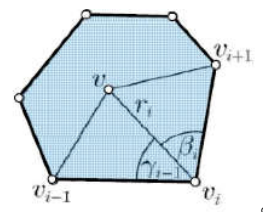
扫描线算法的实现来自[\[简书\]扫描线算法完全解析](#)，进行了简单的优化。

重心坐标的计算采用方法如下

- 四边形

■ Wachspress (WP) coordinates

$$w_i = \frac{\cot \gamma_{i-1} + \cot \beta_i}{r_i^2}$$



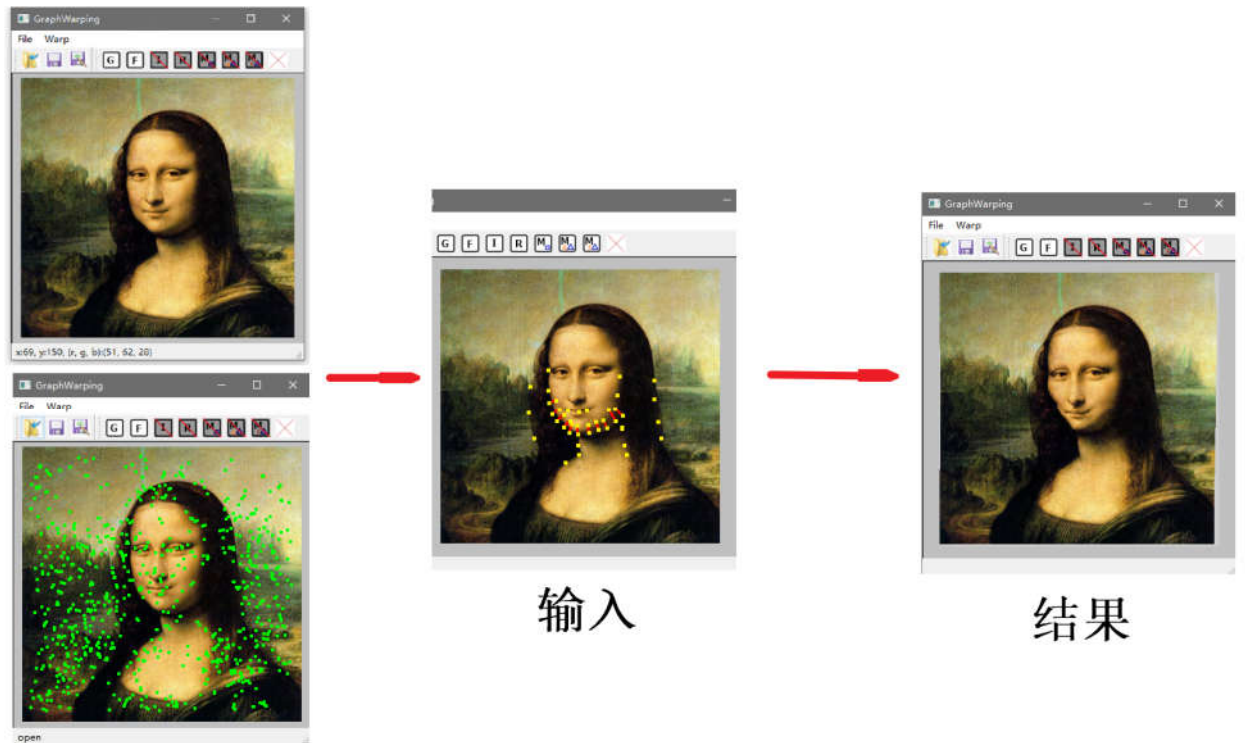
- 三角形

$$p = \lambda_1 t_1 + \lambda_2 t_2 + \lambda_3 t_3, \lambda_1 + \lambda_2 + \lambda_3 = 1$$

$$\lambda_1 = \frac{[PT_2T_3]}{[T_1T_2T_3]}, \lambda_2 = \frac{[T_1PT_3]}{[T_1T_2T_3]}, \lambda_3 = \frac{[T_1T_2P]}{[T_1T_2T_3]}$$

用[PQR]表示有向 ΔPQR 的面积(有正负)

四、结果演示（三角特征网格化算法）












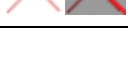
原图+特征点

实际上速度并没有加快多少，复杂度的优化只体现在数据点 n ，而实际上 n 一般不会很大，这样性能提升的不明显，准备工作撒点，三角化，扫描线算法耗费的时间也是比较大的。若真要体现巨大差异，就加大数据点个数。

五、总结

实验 2 只是草草完成，没有精细地对类接口，数据结构等进行设计，可扩展性差，鲁棒性交叉。这次花大量时间对两者进行了弥补，有了很大的提高。

附录

	Open
	Save
	Save As
	Get Data Point
	IDW
	RBF
	Rectangle Mesh
	Triangle Random Mesh
	Triangle Feature Mesh
	Undo