实验名称：用 CImg 实现 Image Morphing

实验要求：

所有的图像读写、数据处理只使用 CImg 库

实验步骤：

对两张输入图像根据 Image Morphing 的方法完成中间 11 帧的差值，得到一个 Image Morphing 的动画视频，图像：

实验过程

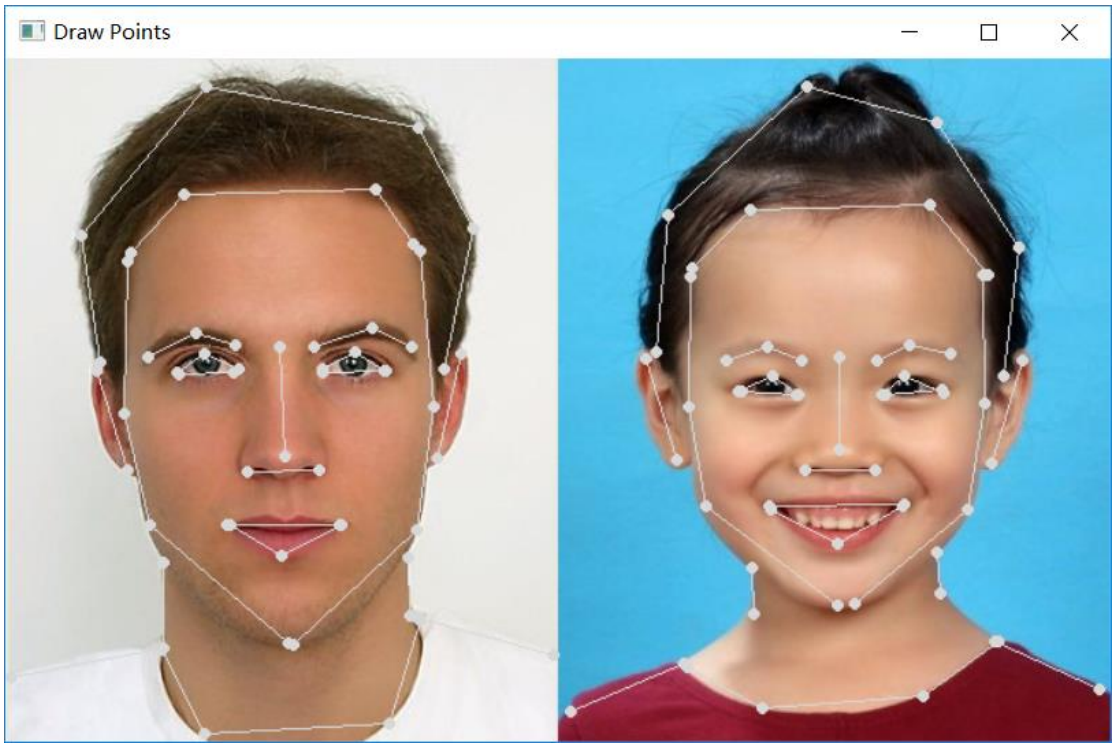
这次实验是要实现 Image Morphing，但实际上，Image Morphing 只是对两张图片的颜色做平均而已，如果直接这样做会出现什么情况呢？可以想见会出现的严重的错位，这样就只是单纯的从一张图片变成另一张图片，而非将一个人平滑的转变成另一个人了。为了达到这一步，必须先对图片进行 warping。

所谓 Warping 就是对图像进行像素位置的转换，使得错位的那些部分平滑的转到正确

位置上，再进行 Morphing。

Warping有两种实现方法，一种是将先将图像上分割为三角网格(用到Delaunay算法)，再对每个三角形进行处理；另一种是描出图像上的特征线，两个图像间的特征线需对应，并通过像素与特征线的映射关系进行 warping；这里选择的是后一种方法。

为此，在这里要先对图片进行描点，好在 cimg 已经提供了 cimg_display，使用它可以检测鼠标事件，来满足我们的描点需求，描点效果如下图（两图的点需对应）：



描完点后将这些点存储为线，这里使用一个 Line struct 来存储：

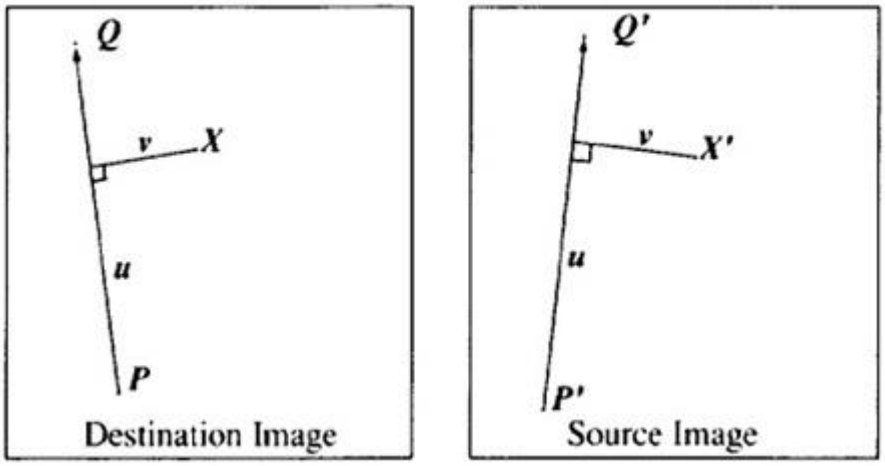
```
struct Line{
    pair<int, int> P;
    pair<int, int> Q;
    double len;
    Line(){}
    Line(pair<int, int> p, pair<int, int> q){
        P = p;
        Q = q;
        len = sqrt((p.first - q.first) * (p.first - q.first) + (p.second -
q.second) * (p.second - q.second));
    }
    pair<double, double> GetCorrPoint(double, double);
    double GetLen();
    double GetDist(double, double, pair<int, int>);
    double Getu(pair<int, int> x);
    double Getv(pair<int, int> x);
};
```

相应的，原图与目标图间的特征线需要对应，这里使用一个 LinePair struct 来存储对应的线段：

```
struct LinePair{
    Line leftLine;
    Line rightLine;
    vector<Line> warpLine;
    void GetWarpLine();
    LinePair(Line l, Line r){
        leftLine = l;
        rightLine = r;
        GetWarpLine();
    }
};
```

注意到这里的 vector 类型的 warpLine，这是为了后面生成多个帧。

得到向量之后就要对图像进行 warping 了，warping 采用的算法是 Beier-Neely 算法，是对目标图的每个像素，找到它对于每条特征线的 u 和 v，再根据这个 u 和 v 找到原图中对应的特征线算出一个新的点X'，目标图上 X 的色彩值便取自原图的X'，对应关系如下图：



其中 u、v，以及新点X'的计算公式为：

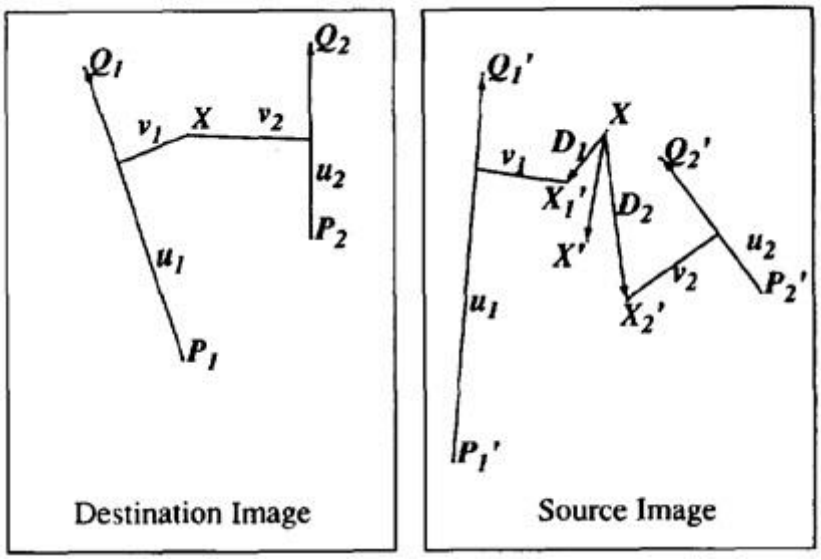
$$u = \frac{(X-P)\cdot(Q-P)}{\|Q-P\|^2} \quad (1)$$

$$v = \frac{(X-P)\cdot Perpendicular(Q-P)}{\|Q-P\|} \quad (2)$$

$$X' = P' + u\cdot(Q'-P') + \frac{v\cdot Perpendicular(Q'-P')}{\|Q'-P'\|} \quad (3)$$

然而，在实际处理中我们会描上很多条特征线，此时就要将这些根据特征线计算出来的

新点整合到一个点上，这一步通过计算每个点的 weight 来实现。如图，多条特征线时：



而此时，weight 的计算公式为：

$$weight[i]=\left(\frac{length[i]^{p}}{a+dist[i]}\right)^{b}$$

其中 a、p、b 这几个参数需要根据图像手动设置，而 length 是特征线的长度，唯一需要注意的是 dist，它的算法如下：

- 如果 0 < u < 1，dist 的值为 abs(v)
- 如果 u < 0，dist 的值就是X'到P'的距离
- 如果 u > 1，dist 的值就是X'到Q'的距离

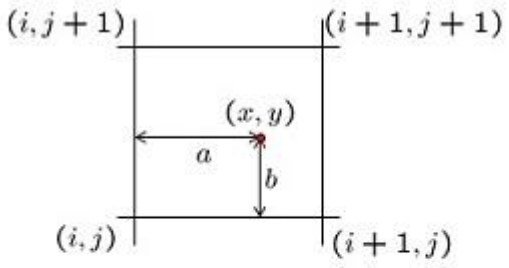
这几个数值的计算都作为前面 line struct 的成员函数而实现了。

在 LinePair 中有一个 vector<line>类型的成员 warpline，正如之前所说，它是用来存储两条对应特征线之间的 warpLine 的，这里总共需要生成 11 个中间帧，那么就需要 11 条 warpline，其生成过程为：

```
for(int i = 0; i < frameCount; ++i){
	double ratio = (double)(i + 1) / (frameCount + 1);
	Line l;
	l.P.first = (1 - ratio) * leftLine.P.first + ratio *
rightLine.P.first;
	l.P.second = (1 - ratio) * leftLine.P.second + ratio *
rightLine.P.second;
	l.Q.first = (1 - ratio) * leftLine.Q.first + ratio *
rightLine.Q.first;
	l.Q.second = (1 - ratio) * leftLine.Q.second + ratio *
rightLine.Q.second;
```

```
        l.len = sqrt((l.P.first - l.Q.first) * (l.P.first - l.Q.first) +
(l.P.second - l.Q.second) * (l.P.second - l.Q.second));
        warpLine.push_back(l);
}
```

Morph 类中 WarpAndMorph 函数是处理的主函数，它对每一帧设置一个渐进的 ratio，并取出对应这一帧的 warpLine，LinePair 中的 leftLine（原图中）对 warpLine 进行 warping，rightLine（目标图）也对 warpLine 进行 warping，得到相应的两个X'。此时由于之前进行的浮点运算，这两个点的坐标有可能不是整数，所以取色彩值时需要进行 Bilinear 操作：



$$
\begin{aligned}
f(x,y) = \quad & (1-a)(1-b) \quad f[i,j] \\
& +a(1-b) \quad f[i+1,j] \\
& +ab \quad f[i+1,j+1] \\
& +(1-a)b \quad f[i,j+1]
\end{aligned}
$$

得到两个色彩值后，再根据 ratio 赋给这一帧的结果图像：

```
res(x, y, 0) = ratio * colorL(0, 0, 0) + (1 - ratio) * colorR(0, 0, 0);
res(x, y, 1) = ratio * colorL(0, 0, 1) + (1 - ratio) * colorR(0, 0, 1);
res(x, y, 2) = ratio * colorL(0, 0, 2) + (1 - ratio) * colorR(0, 0, 2);
```

这样一帧一帧的处理，最后就能够得到 11 个中间帧了。

原图、中间结果、目标图对应：

最后使用 ffmpeg 就能够将原来的两幅图+11 个中间帧变为 gif 图像。

在生成过程中，a、p、b 这三个参数对于结果的影响是很大的，如图：

a=1 p=0 b=1

a=1 p=0 b=2

a=2 p=0 b=2

a=1 p=1 b=2

可见 a=1，p=0，b=1 时，效果并不好，但仅把 b 变为 2 后，效果就好了很多。此时把 a 变为 2，没什么变化，而若是把 p 增加为 1，效果又变差了许多。测试发现 a=1，p=0，b=2 这组参数的效果是最好的。

实现的其他一些测试图像：