Ex5附加题：人脸渐变

14331098 黄建武 计应

- 实验环境
  - Linux Deepin 15.3
  - g++ 6.2.0
  - C++11
- 参考资料
  - 人脸特征点检测
  - 求三角形外接圆
  - 判断点是否在三角形内
  - Bowyer-Watson算法
- 实验要求

输入图像是两张人脸图像, 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。

- 实验方法
  - 分别对原图像和目标图像进行人脸特征点标记（使用dlib库）
  - 使用 $F = \alpha A + (1-\alpha)B$求出中间过渡图像的特征点对应的坐标
  - 对中间过渡图像的特征点进行三角剖分（Bowyer-Watson算法）
  - 根据剖分后的三角形，求出它在原图像和目标图像中对应的三角形（记录三角形顶点对应的下标）
  - 分别计算中间过渡图像的每个三角形到原图像和目标图像的变换矩阵
  - 对于中间图像中的每个点，求出它所在的三角形，使用对应的变换函数，将坐标变换到原图像中，插值得到像素值，最终得到一张变换后的图像a
  - 同理使用中间图像到目标图像的变换矩阵进行变换，得到变换后的图像b。
  - 使用 $F = \alpha A + (1-\alpha)B$对图像a、b的像素值进行加权，求出中间一帧的图像。
  - 从0到1增大α的值，重复以上步骤
  - 将原图像、目标图像和中间10帧的图像保存为gif
- 实验实现

使用dlib库对68个人脸特征点标记

```
void getFeaturePoints(const char* filename, std::vector<Point>& points) {
  CImg<float> src(filename);

  frontal_face_detector detector = get_frontal_face_detector(); // 人脸探测器
  array2d<rgb_pixel> img;
  load_image(img, filename);
  std::vector<rectangle> faces = detector(img);  // 返回探测到的所有人脸
  shape_predictor sp;
  deserialize("shape_predictor_68_face_landmarks.dat" ) >> sp;
  full_object_detection shape = sp(img, faces[0]);

  // 显示人脸特征点
  float red[] = {255, 0, 0};
```

```
  for (int i = 0; i < shape.num_parts(); ++i) {
    point p = shape.part(i);
    src.draw_circle(p.x(), p.y(), 2, red);
    points.push_back(Point(p.x(), p.y()));
  }
  src.display();
}
```

三角剖分

```
// 三角剖分
// Bowyer-Watson算法(推荐)
// 算法的基本步骤是:
// 1、构造一个超级三角形，包含所有散点，放入三角形链表。
// 2、将点集中的散点依次插入，在三角形链表中找出外接圆包含插入点的三角形(称为该点的影响三角形)
//     删除影响三角形的公共边，将插入点同影响三角形的全部顶点连接起来，
//     完成一个点在Delaunay三角形链表中的插入。
// 3、根据优化准则对局部新形成的三角形优化。将形成的三角形放入Delaunay三角形链表。
// 4、循环执行上述第2步，直到所有散点插入完毕。
void Delaunay::triangulate(vector<Point> &points) {
  // 求超级三角形
  float minX = points[0].x;
  float minY = points[0].y;
  float maxX = minX;
  float maxY = minY;

  for (Point p : points) {
    if (p.x < minX) minX = p.x;
    if (p.y < minY) minY = p.y;
    if (p.x > maxX) maxX = p.x;
    if (p.y > maxY) maxY = p.y;
  }

  float dx = maxX - minX, dy = maxY - minY;
  float deltaMax = max(dx, dy);
  float midx = (minX + maxX) / 2, midy = (minY + maxY) / 2;

  Point p1(midx - 20 * deltaMax, midy - deltaMax);
  Point p2(midx, midy + 20 * deltaMax);
  Point p3(midx + 20 * deltaMax, midy - deltaMax);

  // 将超级三角形加入三角形列表
  _triangles.push_back(Triangle(p1, p2, p3));

  for(Point p : points) {
    vector<Triangle> badTriangles;
    vector<Edge> polygon;

    for(Triangle t : _triangles) {
      if(t.circumCircleContains(p)) {
```

```
        badTriangles.push_back(t);
        polygon.push_back(t.e1);
        polygon.push_back(t.e2);
        polygon.push_back(t.e3);
      }
    }

    _triangles.erase(remove_if(begin(_triangles), end(_triangles),
                 [badTriangles](Triangle &t) {
      for(Triangle bt : badTriangles) {
        if (bt == t) return true;
      }
      return false;
    }), end(_triangles));

    vector<Edge> badEdges;
    int size = polygon.size();
    for(int i = 0; i < size; ++i) {
      for(int j = i + 1; j < size; ++j) {
        if (polygon[i] == polygon[j]) {
          badEdges.push_back(polygon[i]);
          badEdges.push_back(polygon[j]);
        }
      }
    }

    polygon.erase(remove_if(begin(polygon), end(polygon),
                [badEdges](Edge &e){
      for(Edge be : badEdges) {
        if (be == e) return true;
      }
      return false;
    }), end(polygon));

    for(Edge e : polygon)
      _triangles.push_back(Triangle(e.p1, e.p2, p));
  }


  _triangles.erase(remove_if(begin(_triangles), end(_triangles),
                [p1, p2, p3](Triangle &t){
    return t.containsVertex(p1)
         || t.containsVertex(p2)
         || t.containsVertex(p3);
  }), end(_triangles));
}
```

求出中间过渡特征点的坐标，需要注意的一点，α是从0到1，原图像的权重应该是从1到0减小，后面两幅图像像素点进行加权也是如此，这里被坑了

```
// 坐标变换
inline Point F(const Point& p1, const Point& p2, float alpha) {
  float x = (1 - alpha) * p1.x + alpha * p2.x;
  float y = (1 - alpha) * p1.y + alpha * p2.y;
  return Point(x, y);
}
```

计算两个三角形的变换矩阵，如$TA = B$ 则$T = B/A$ 可直接调用CImg的矩阵除法，这里需要使用两个三角形的顶点坐标构造矩阵AB

$$T\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} x_1' & x_2' & x_3' \\ y_1' & y_2' & y_3' \\ 1 & 1 & 1 \end{pmatrix}$$

需要注意的是，CImg里面声明的3x3的矩阵，下标是先y后x，写反了变换后的图像跟鬼一样，坑了好久

```
// 获取变换矩阵
// !!! CImg 下标是先y后x !!!
// 坑爹，以后要注意
CImg<float> getTransformMatrix(const Triangle& t1, const Triangle& t2) {
  CImg<float> m1(3, 3);
  m1(0, 0) = t1.p1.x;
  m1(1, 0) = t1.p2.x;
  m1(2, 0) = t1.p3.x;
  m1(0, 1) = t1.p1.y;
  m1(1, 1) = t1.p2.y;
  m1(2, 1) = t1.p3.y;
  m1(0, 2) = m1(1, 2) = m1(2, 2) = 1;

  CImg<float> m2(3, 3);
  m2(0, 0) = t2.p1.x;
  m2(1, 0) = t2.p2.x;
  m2(2, 0) = t2.p3.x;
  m2(0, 1) = t2.p1.y;
  m2(1, 1) = t2.p2.y;
  m2(2, 1) = t2.p3.y;
  m2(0, 2) = m2(1, 2) = m2(2, 2) = 1;

  return m2 / m1;
}
```

对原图像进行三角变换

```
//对原图像进行三角变换
CImg<float> target1 = src1;
cimg_forXY(target1, x, y) {
  bool flag = false;
  Point p(x, y);
```

```
  int size = triangles.size();
  for (int i = 0; i < size; ++i) {
    if (triangles[i].isPointInTriangle(p)) {
      // 计算中间图像对应于原图像的坐标
      // CImg 坐标先y后x
      float tx = x * mat1[i](0, 0) + y * mat1[i](1, 0) + mat1[i](2, 0);
      float ty = x * mat1[i](0, 1) + y * mat1[i](1, 1) + mat1[i](2, 1);
      int _x = (int)(tx), _y = (int)(ty);

      if (_x >= 0 && _x < width && _y >= 0 && _y < height) {
        cimg_forC(src1, c)
          target1(x, y, c) = src1.linear_atXY(_x, _y, c);
      }
      flag = true;
      break;
    }
  }
  if (!flag) {
    cimg_forC(src1, c)
      target1(x, y, c) = src1.linear_atXY(x, y, c);
  }
}
```

对生成的两幅图像像素值进行加权

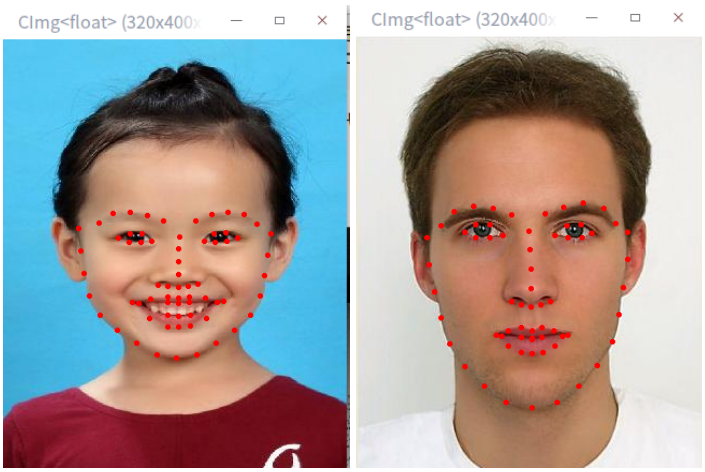
```
// 像素加权
CImg<float> result = src1;
cimg_forXYC(result, x, y, c)
  result(x, y, c) = alpha * target2(x, y, c) + (1 - alpha) * target1(x, y, c);

(target1, target2, result).display();
```

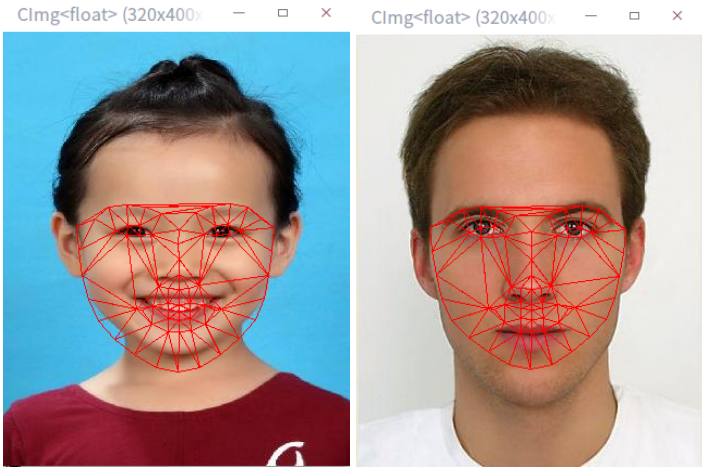
完整实现请看源代码。

- 实验结果

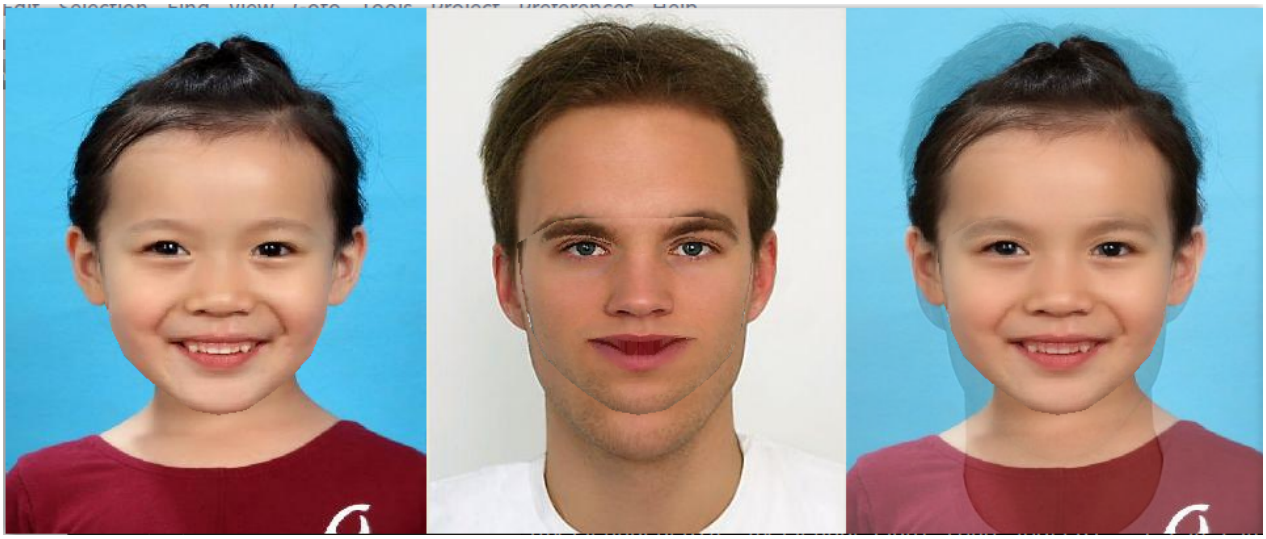
  人脸特征点



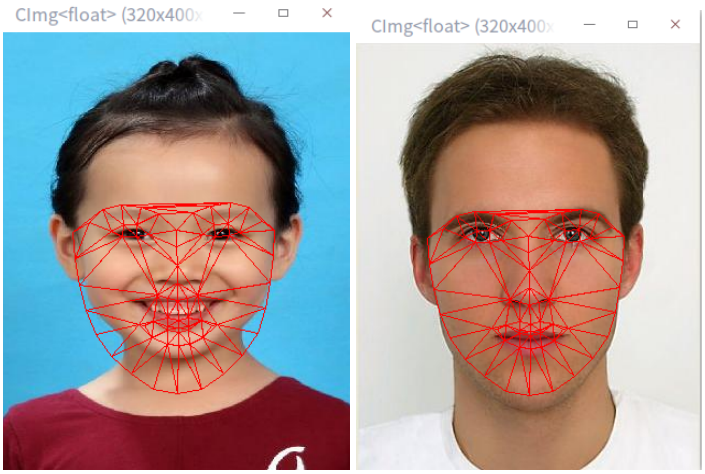
第3帧的三角剖分



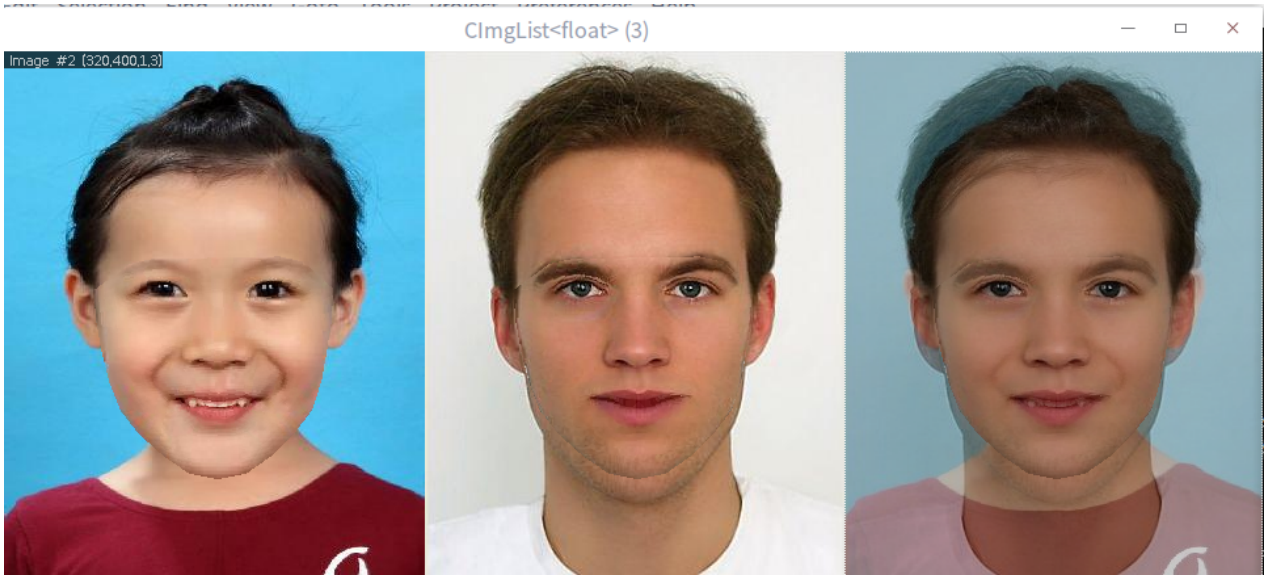
第3帧的图像(右边)，由左边和中间加权得到



第7帧的三角剖分



第7帧的图像(右边)，由左边和中间加权得到



实验问题

- 在进行三角剖分时，一开始使用的是暴力枚举所有的三角形，判断其他点是否在该三角形的外接圆内，可是最终剖分出来的结果还是有三角形重叠，特别是脸部特征点密集的区域，上网查阅后改用Bowyer-Watson算法，正确地对图像进行了剖分
- 一开始思路错，认为过渡三角形是通过原图像和目标图像对应的三角形求出来的，所以我先对原图像进行三角剖分，再对目标图像进行三角剖分，可是结果两张图像剖分的三角形个数不一样，无法对应。仔细想了之后才发现应该是通过原图像和目标图像的特征点坐标求出过渡图像的特征点坐标，对过渡图像进行三角剖分后反求出原图像和目标图像的三角形，这样形成的就是一一对应的关系。
- 计算三角形变换矩阵时，踩了CImg的大坑，构建了3x3的矩阵后，直接调用CImg重载的矩阵除法求出变换矩阵，当对图像进行处理时，处理的结果跟鬼一样，改了其他地方的小bug后确定其他地方都是对的后，居然怀疑起CImg的矩阵除法错了，看了源代码后发现 `cimg_forXY` 是相当于先y循环再x循环，3x3的矩阵里的索引也是先y后x，所以整个变换矩阵都反了，变换后的点的坐标很多为负数，导致很多个三角形畸形。
- 忘记对特征点的下标进行赋值，默认值都是-1，因此在从过渡三角形反推原图像和目标图像的三角形时，选取的点都越界了，可是c++运行没有报错，而且图像还能处理（这个时候我的变换矩阵错误的bug还没发现），很正常的显示，只是不是人脸渐变的效果，是淡入淡出的效果，原图像淡出，目标图像淡入，一度以为已经完成作业了，也是服气。