中山大學
SUN YAT-SEN UNIVERSITY

# 《计算机视觉》

# 实验文档

## （实验五）

| | |
|---|---|
| 学 院 名 称 ： | 数据科学与计算机学院 |
| 专业（班级）： | 16 软件工程（数字媒体） |
| 学 生 姓 名 ： | 黎汛言 |
| 学 号 ： | 16340109 |
| 时 间 ： | 2018 年 11 月 18 日 |

# 实验五 ：Image Morphing

## 一. 实验目的

1. 了解Image Morphing的原理和实现方法。

## 二. 实验内容

1. 输入两幅人脸图像，根据Image Morphing的方法完成中间11帧的插值，得到一个过渡动画视频。

## 三. 实验环境

Windows 10 64位

## 四. 实验过程

1. 定义类Point和Triangle，记录点和三角形信息：

```
class Point {
public:
    int x, y;    // 点的x和y坐标值

    Point(): x(0), y(0) {}
    Point(int _x, int _y): x(_x), y(_y) {}
};
```

```
class Triangle {
public:
    Point p1, p2, p3;    // 三角形的三个顶点
    int index[3];        // 三个顶点在原图点集中的下标

    Triangle(Point _p1, Point _p2, Point _p3, int index1, int index2, int index3) {
        p1 = _p1;
        p2 = _p2;
        p3 = _p3;
        index[0] = index1;
        index[1] = index2;
        index[2] = index3;
    }
};
```

2. 定义类Morphing，将Morphing操作都封装在类中。使用两个CImg对象表示输入的两幅图像，使用两个vector<Point>来表示两幅图像的标记点的集合，使用两个vector<Triangle>来表示两幅图像中的三角划分，使用一个CimgList对象记录过

渡动画。

```
class Morphing {
private:
    CImg<unsigned char> source;              // 原图像
    CImg<unsigned char> target;              // 目标图像
    std::vector<Point> sourcePoint;          // 原图像的标记点
    std::vector<Point> targetPoint;          // 目标图像的标记点
    std::vector<Triangle> sourceTriangle;    // 原图像中的三角划分
    std::vector<Triangle> targetTriangle;    // 目标图像的三角划分
    CImgList<unsigned char> morphing;        // 过渡动画
```

3. Morphing类的成员函数如下，函数功能见注释。函数的具体实现见后。

```
public:
    Morphing(const char* sourcePath, const char* targetPath);
    void getPoints() ;                       // 获取图像标记点
    void getTriangles();                     // 根据标记点，获取图像的三角划分
    void calculateMiddleFrames();            // 计算所有过渡帧
    void save();                             // 保存过渡动画

    void selectPoints(CImg<unsigned char>& src,
        std::vector<Point>& points, const char* filename);          // 从文件中读取或手动标注标记点
    bool isPointInCircle(Point& P, Triangle& tri);                  // 判断一个点是否在三角形的外接圆内
    bool isPointInTriangle(Point& P, Triangle& tri);                // 判断一个点是否在三角形内
    CImg<double> calculateTransformMatrix(Triangle from, Triangle to);  // 给定变换前后的三角形，计算仿射变换的矩阵
```

4. 在构造函数中，根据路径读取图像；在getPoints()函数中，调用selectPoints函数分别获取两幅图像的标记点。

```
Morphing::Morphing(const char* sourcePath, const char* targetPath) {
    source.load(sourcePath);
    target.load(targetPath);
}

void Morphing::getPoints() {
    selectPoints(source, sourcePoint, "1.txt");
    selectPoints(target, targetPoint, "2.txt");
}
```

selectPoints()函数实现参见源代码。

5. 在getTriangles()函数中，利用遍历的方法得到原图的三角划分。然后根据原图的三角划分，在目标图中得到一一对应的三角划分。

```
void Morphing::getTriangles() {
    int size = sourcePoint.size();
    // 遍历所有三角形
    for (int i = 0; i < size - 2; i++) {
        for (int j = i + 1; j < size - 1; j++) {
            for (int k = j + 1; k < size; k++) {
                Point A = sourcePoint[i];
                Point B = sourcePoint[j];
                Point C = sourcePoint[k];
                Triangle tri(A, B, C, i, j, k);
                bool valid = true;

                // 遍历所有标记点
                for (int m = 0; m < size; m++) {
                    Point temp = sourcePoint[m];
                    if (m == i || m == j || m == k)
                        continue;
                    double k1 = (double)(A.y - B.y) / (A.x - B.x);
                    double k2 = (double)(A.y - C.y) / (A.x - C.x);
                    // 如果有标记点在三角形的外接圆内，或者三角形三点在一条直线上，则三角形不符合要求
                    if (isPointInCircle(temp, tri) || k1 == k2 || (A.x == B.x && B.x == C.x)) {
                        valid = false;
                        break;
                    }
                }
                // 若符合要求，则加入到集合中
                if (valid) {
                    sourceTriangle.push_back(tri);
                }
            }
        }
    }
    // 根据原图三角形顶点的下标，形成目标图的三角划分，以保证三角形一一对应
    for (int i = 0; i < sourceTriangle.size(); i++) {
        int index1 = sourceTriangle[i].index[0];
        int index2 = sourceTriangle[i].index[1];
        int index3 = sourceTriangle[i].index[2];
        Triangle tri(targetPoint[index1], targetPoint[index2], targetPoint[index3],
            index1, index2, index3);
        targetTriangle.push_back(tri);
    }
}
```

判断点是否在三角形外接圆内的实现参见源代码。

6. 中间帧插值的过程

每一帧需要如下的变量：

```
    double a = (double)i / (TOTAL_FRAME - 1);    // 线性变换的权重a
    CImg<unsigned char> middle(target.width(), target.height(), 1, 3, 0);    // 当前帧的图像
    std::vector<Triangle> middleTriangle;        // 当前帧的三角划分
    std::vector<CImg<double> > middleToSource;   // 当前帧的三角形到原图三角形的仿射变换的所有变换矩阵的集合
    std::vector<CImg<double> > middleToTarget;   // 当前帧的三角形到目标图三角形的仿射变换的所有变换矩阵的集合
```

首先得到中间帧的三角形划分：

```
// 根据权重a，得到当前帧的三角划分
for (int j = 0; j < sourceTriangle.size(); j++) {
    // 得到三角形顶点下标
    int index1 = sourceTriangle[j].index[0];
    int index2 = sourceTriangle[j].index[1];
    int index3 = sourceTriangle[j].index[2];
    // 计算三角形顶点坐标
    int mid_x1 = (1 - a) * sourcePoint[index1].x + a * targetPoint[index1].x + 0.5;
    int mid_y1 = (1 - a) * sourcePoint[index1].y + a * targetPoint[index1].y + 0.5;
    int mid_x2 = (1 - a) * sourcePoint[index2].x + a * targetPoint[index2].x + 0.5;
    int mid_y2 = (1 - a) * sourcePoint[index2].y + a * targetPoint[index2].y + 0.5;
    int mid_x3 = (1 - a) * sourcePoint[index3].x + a * targetPoint[index3].x + 0.5;
    int mid_y3 = (1 - a) * sourcePoint[index3].y + a * targetPoint[index3].y + 0.5;
    // 得到中间三角形
    Triangle tri(Point(mid_x1, mid_y1), Point(mid_x2, mid_y2), Point(mid_x3, mid_y3),
        index1, index2, index3);
    middleTriangle.push_back(tri);
}
```

然后计算每一个三角形到原图和目标图对应三角形的仿射变换的变换矩阵，存入对应的vector中：

```
// 得到每一个三角形到原图和目标图对应三角形的变换矩阵
for (int j = 0; j < middleTriangle.size(); j++) {
    CImg<double> toSource = calculateTransformMatrix(middleTriangle[j], sourceTriangle[j]);
    middleToSource.push_back(toSource);
    CImg<double> toTarget = calculateTransformMatrix(middleTriangle[j], targetTriangle[j]);
    middleToTarget.push_back(toTarget);
}
```

接下来，遍历中间帧的像素，判断像素属于哪个中间三角形。根据上面计算出来的变换矩阵，求出该像素对应到原图和目标图的像素。最后根据该帧权重a，完成该像素的插值。

```
// 生成中间帧
cimg_forXY(middle, x, y) {
    for (int j = 0; j < middleTriangle.size(); j++) {
        Triangle tri = middleTriangle[j];
        Point P(x, y);
        // 判断像素属于哪个中间三角形
        if (isPointInTriangle(P, tri)) {
            CImg<double> mid_xy(1, 3, 1, 1, 1);
            CImg<double> src_xy(1, 3, 1, 1, 1);
            CImg<double> tar_xy(1, 3, 1, 1, 1);
            mid_xy(0) = x;
            mid_xy(1) = y;

            // 使用变换矩阵求出对应原图和目标图的坐标
            src_xy = middleToSource[j] * mid_xy;
            tar_xy = middleToTarget[j] * mid_xy;

            // 根据权重a，求得过渡像素值
            cimg_forC(middle, c) {
                middle(x, y, c) = (1 - a) * source((int)(src_xy(0) + 0.5), (int)(src_xy(1) + 0.5), c) +
                                  a * target((int)(tar_xy(0) + 0.5), (int)(tar_xy(1) + 0.5), c);
            }
            break;
        }
    }
}
```

最后插入中间帧：

```
// 加入中间帧
morphing.push_back(middle);
```

计算仿射变换矩阵和判断点是否在三角形内部的实现参见源代码。

7. 使用CImg.save_gif_external函数保存过渡动画。

```
void Morphing::save() {
    morphing.save_gif_external("result.gif", 13);
}
```

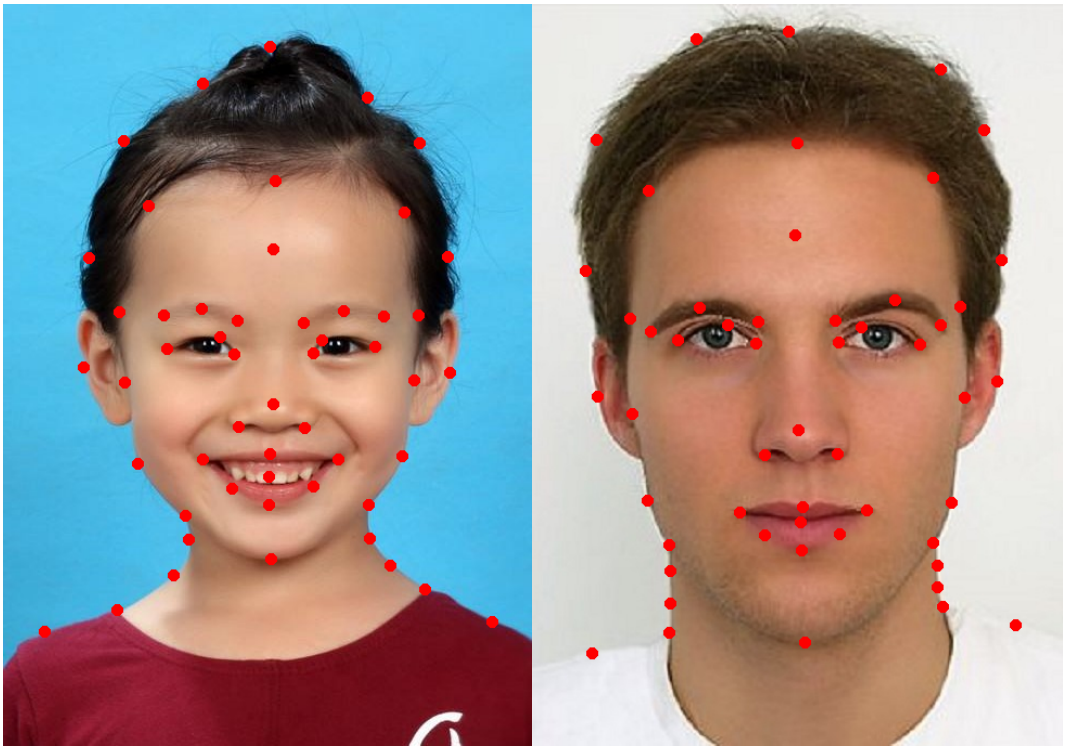
## 五. 测试过程及结果分析

测试函数：

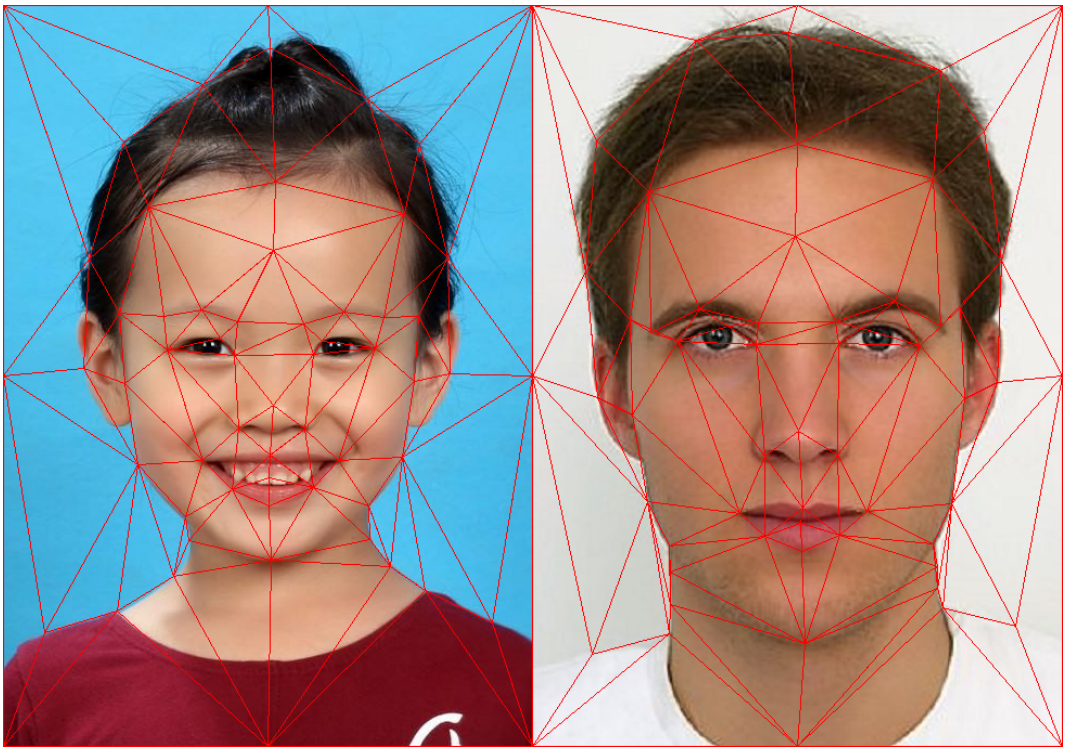
```
int main() {
    test("1.jpg", "2.bmp");
    return 0;
}
```

```
// 测试函数
void test(const char* sourcePath, const char* targetPath) {
    // 读入图像
    Morphing morphing = Morphing(sourcePath, targetPath);
    // 获取标记点
    morphing.getPoints();
    // 获取三角划分
    morphing.getTriangles();
    // 计算中间帧
    morphing.calculateMiddleFrames();
    // 输出结果
    morphing.save();
}
```

标记点如下：



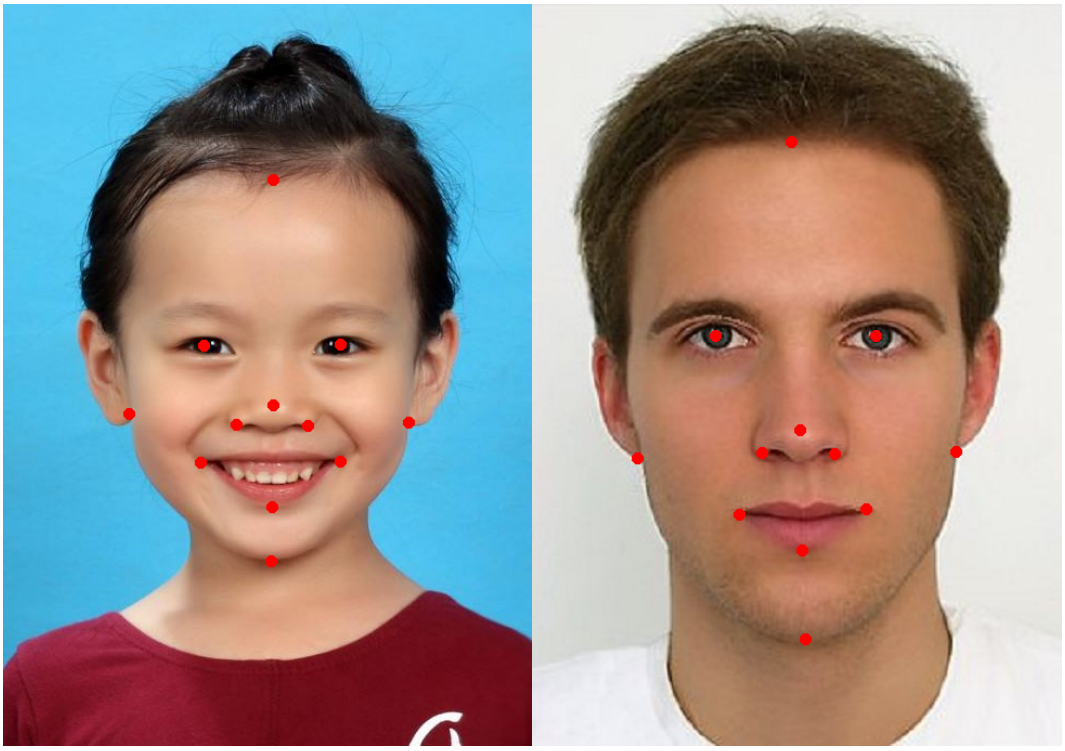
形成的三角划分如下：



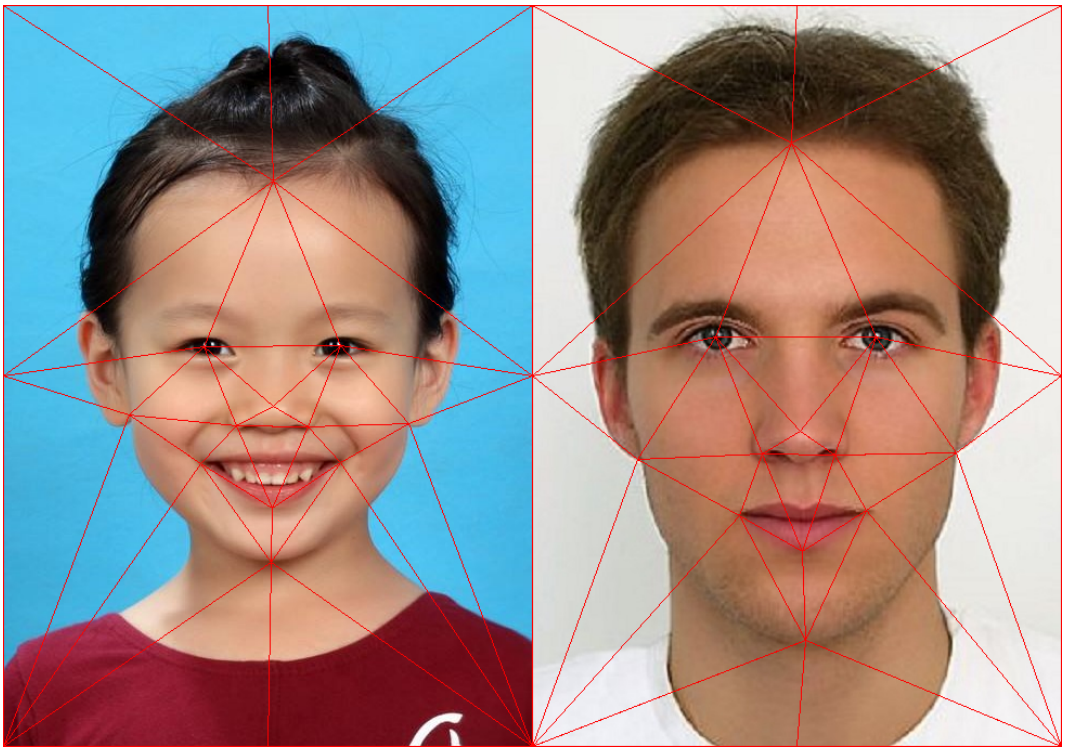
结果如下，动画参见result.gif。

可以看到，动画过渡十分自然。这与标记点的选择密切相关。下面来看一组对比测试。

**标记点：**



**形成的三角划分：**



**结果：**

**可以看到，这次测试的过渡效果明显不如前一次的好。原因在于标记点太少，导致形成的三角形太少，造成插值不准确。**