作业五验报告及测试文档

数据科学与计算机学院 梁育诚 16340133

一、实验内容

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。

二、实验原理

实验的重点是人脸变换，首先我们来讨论人脸。人脸是很复杂的图像，其信息非常多，在这次实验中我们关注的是人脸中的特征点。我们通过特征点来构建不同的三角形，通过三角形之间的变换得到变换矩阵，再将这个矩阵应用所有在三角形中的点，这样我们就可以将旧的三角变换到新的三角了。所以实验中关键的步骤有两个：一是获取人脸的特征点，因为当前还没有学习相关的算法，这里我直接找了网上一个博客给出的特征点数据，自己再稍微修改了一下直接使用了。二是实现三角形之间的仿射变换。这里重点解释一下三角形的仿射。从数学的角度看，仿射变换就是空间中，向量经过一次线性变换加一次平移变换，数学语言表示为：$\vec{q} = A\vec{p} + \vec{b}$，其中$\vec{p}$是变换前的原向量，$\vec{q}$是变换后的目标向量，A 称为线性变换矩阵，$\vec{b}$为平移变换向量。以上写成矩阵形式：

$$\begin{bmatrix}\vec{q}\\1\end{bmatrix} = \begin{bmatrix}A & \vec{b}\\0\cdots 0 & 1\end{bmatrix}\begin{bmatrix}\vec{p}\\1\end{bmatrix}$$

在这次实验中，我们操作的是平面图像，因此这里的$\vec{p} = \begin{bmatrix}x_1\\y_1\end{bmatrix}$，$\vec{q} = \begin{bmatrix}x_1'\\y_1'\end{bmatrix}$，应用到三角形中，就是$\begin{bmatrix}x_1' & x_2' & x_3'\\y_1' & y_2' & y_3'\\1 & 1 & 1\end{bmatrix} = \mathrm{M}\begin{bmatrix}x_1 & x_2 & x_3\\y_1 & y_2 & y_3\\1 & 1 & 1\end{bmatrix}$，M 就是我们要求的变换矩阵了，$\mathrm{M} = \begin{bmatrix}a & b & c\\d & e & f\\0 & 0 & 1\end{bmatrix}$。

问题转化为求解 M，看上去很复杂，其实有捷径。只需要将方程两边同时求转置，我们就可以通过求解线性方程组来求得M^T，最后再转回 M 即可。

三、实验过程

1. 首先检验测试图片的完整性。发现 2.bmp 无法正常读入，需要手动另存为 bmp，同时需要调整大小，保证两张图片的大小一致。
2. 实现读入数据部分，包括读入 source image 和 target image，以及手动输入的关于两张图片的特征点坐标文件，以及构成三角形的特征点数组下标。这里用到了 fstream 进行 txt 文件读入。

```cpp
void Morphing::readSource() {
  cout << "Read Source" << endl;
  source.load_bmp("./data/1.bmp");
}

void Morphing::readTarget() {
  cout << "Read Target" << endl;
  target.load_bmp("./data/2.bmp");
}
```

```cpp
// load source points
ifstream srcPointFile;
srcPointFile.open("./data/sourcepoints.txt");
if (srcPointFile.is_open()) {
  string s;
  int x, y;
  while (getline(srcPointFile, s)) {
    stringstream ss(s);
    ss >> x >> y;
    sourcePoints.emplace_back(Point(x, y));
  }
}
else {
  cout << "sourcepoints File not exist!" << endl;
}
```

3. 利用特征点构建三角形。我们已知两张图片的各自的特征点，以及用于构建三角形的特征点下标，因此我们可以利用插值的方法生成中间过程（11帧）的中间点以及对应的三角形。

```cpp
void Morphing::generateTrianglesFromPoints() {
  cout << "Begin Generate Triangles" << endl;
  // from keypoints to triangles
  for (int i = 0; i < index.size(); i++) {
    Triangle t1(sourcePoints[index[i][0]], sourcePoints[index[i][1]], sourcePoints[index[i][2]]);
    sourceTriangles.emplace_back(t1);

    Triangle t2(targetPoints[index[i][0]], targetPoints[index[i][1]], targetPoints[index[i][2]]);
    targetTriangles.emplace_back(t2);
  }

  // generate mid-process points
  for (int i = 0; i < frames; i++) {
    vector<Point> v;
    for (int j = 0; j < sourcePoints.size(); j++) {
      // Interpolation
      float x = float(sourcePoints[j].x) + float(i + 1) / (frames + 1) *
                (targetPoints[j].x - sourcePoints[j].x);
      float y = float(sourcePoints[j].y) + float(i + 1) / (frames + 1) *
                (targetPoints[j].y - sourcePoints[j].y);
      v.emplace_back(Point(x, y));
    }
    midPoints.emplace_back(v);
  }
```

```
  // generate mid-process triangles
  for (int i = 0; i < frames; i++) {
    vector<Triangle> v;
    for (int j = 0; j < index.size(); j++) {
      Triangle t(midPoints[i][index[j][0]], midPoints[i][index[j][1]], midPoints[i][index[j][2]]);
      v.emplace_back(t);
    }
    midTriangles.emplace_back(v);
  }
  cout << "End of Generate Triangles" << endl;
}
```

4. 构建仿射变换矩阵，基本的数学原理在第二部分已经分析，这里解释如何进行 coding。CImg 库给我们提供了 solve()函数来求解线性方程组。x = b.solve(A)求解的是 Ax=b 的方程（注意未知数 x 是在后面的）。但是我们求解的是$A^{'} = MA$。这里的未知数 M 是在 A 前面的，因此无法使用 solve()直接求解。这里利用矩阵转置的性质，交换位置，得：$(A^{'})^T = A^T M^T$。因为$A^{'}$矩阵和 M 矩阵都有一行是常数，因此我们不需要求他们的转置，而是分开两个向量来求就可以了，这样的计算量更简单。最后求得的两个向量求转置后拼接成 M 即可。

```
CImg<float> Morphing::generateMatrix(Triangle oldT, Triangle newT) {
  CImg<float> A(3, 3, 1, 1, 1);
  CImg<float> y1(1, 3, 1, 1, 0);
  CImg<float> y2(1, 3, 1, 1, 0);
  CImg<float> c1(1, 3, 1, 1, 0);
  CImg<float> c2(1, 3, 1, 1, 0);

  // A transposition
  A(0, 0) = oldT.point1.x;
  A(0, 1) = oldT.point2.x;
  A(0, 2) = oldT.point3.x;
  A(1, 0) = oldT.point1.y;
  A(1, 1) = oldT.point2.y;
  A(1, 2) = oldT.point3.y;

  // A' transposition
  y1(0, 0) = newT.point1.x;
  y1(0, 1) = newT.point2.x;
  y1(0, 2) = newT.point3.x;
  y2(0, 0) = newT.point1.y;
  y2(0, 1) = newT.point2.y;
  y2(0, 2) = newT.point3.y;

  c1 = y1.solve(A);
  c2 = y2.solve(A);
```

```
147     // construct M
148     CImg<float> M(3, 3, 1, 1, 0);
149     for (int i = 0; i < 3; i++) {
150       M(i, 0) = c1(0, i);
151       M(i, 1) = c2(0, i);
152     }
153     M(2, 2) = 1;
154     return M;
155   }
```

5. 判断一个点是否在给定的三角形中。第 3 步我们根据三个特征点求得一个三角形的仿射变换，那么我们需要对这个三角形内的所有点都做仿射变换，判断一个点是否在这个三角形中，网上查找后使用了比较常规的重心法。

```
157 bool Morphing::inTriangle(Point p, Triangle t) {
158   float x0 = t.point3.x - t.point1.x;
159   float y0 = t.point3.y - t.point1.y;
160   float x1 = t.point2.x - t.point1.x;
161   float y1 = t.point2.y - t.point1.y;
162   float x2 = p.x - t.point1.x;
163   float y2 = p.y - t.point1.y;
164
165   float dot00 = x0 * x0 + y0 * y0;
166   float dot01 = x0 * x1 + y0 * y1;
167   float dot02 = x0 * x2 + y0 * y2;
168   float dot11 = x1 * x1 + y1 * y1;
169   float dot12 = x1 * x2 + y1 * y2;
170
171   float inverDeno = 1.0 / (dot00 * dot11 - dot01 * dot01);
172   float u = (float)(dot11 * dot02 - dot01 * dot12) * inverDeno;
173   if (u < 0 || u > 1) {
174     return false;
175   }
176
177   float v = (dot00 * dot12 -dot01 * dot02) * inverDeno;
178   if (v < 0 || v > 1) {
179     return false;
180   }
181
182   return  u + v <= 1;
183 }
```

6. 实现 morphing。Morphing 的过程是逐帧操作的。每一帧都是对图片中的所有三角形做仿射变换，每一帧使用不同比重的线性插值（source 的权重越来越小，target 的权重越来越大），生成中间图像的像素点。最后使用一个 CImgList 来存储每一次的结果。

```
for (int i = 0; i < frames; i++) {
  CImg<float> mid(target._width, target._height, 1, 3, 1);
  cout << "In frame: " << i+1 << endl;
  cimg_forXY(mid, x, y) {
    CImg<float> x_(1, 3, 1, 1, 1);
    CImg<float> y1(1, 3, 1, 1, 1);
    CImg<float> y2(1, 3, 1, 1, 1);
    for (int m = 0; m < size; m++) {
      Point p(x, y);
      if (inTriangle(p, midTriangles[i][m])) {
        // transform each point in the triangle using Affix Transformation Matrix
        x_(0, 0) = x;
        x_(0, 1) = y;

        // from mid to source
        CImg<float> trans1 = generateMatrix(midTriangles[i][m], sourceTriangles[m]);
        y1 = trans1 * x_;
        // from mid to target
        CImg<float> trans2 = generateMatrix(midTriangles[i][m], targetTriangles[m]);

        y2 = trans2 * x_;

        // linear interpolation
        float a = float(i + 1) / (frames + 1);
        mid(x, y, 0) = (1 - a) * source(y1(0, 0), y1(0, 1), 0) + a * target(y2(0, 0), y2(0, 1), 0);
        mid(x, y, 1) = (1 - a) * source(y1(0, 0), y1(0, 1), 1) + a * target(y2(0, 0), y2(0, 1), 1);
```

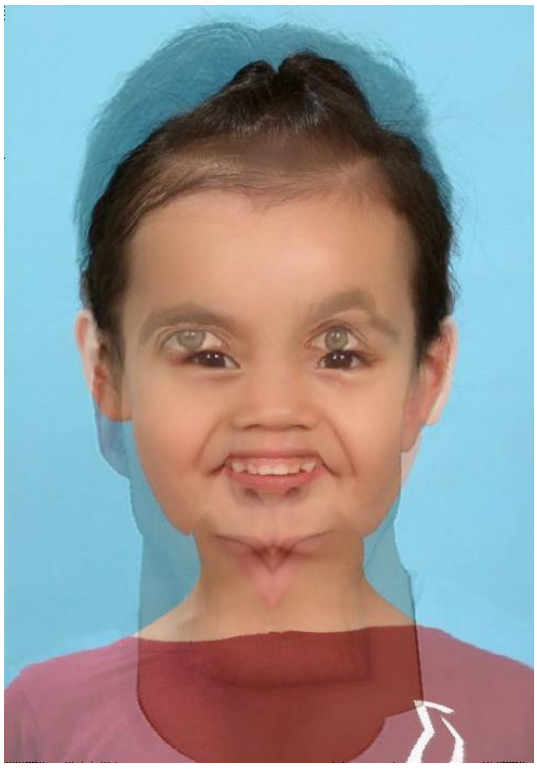
```
        mid(x, y, 2) = (1 - a) * source(y1(0, 0), y1(0, 1), 2) + a * target(y2(0, 0), y2(0, 1), 2);
        break;
      }
    }
  }
  result.push_back(mid);
}
```

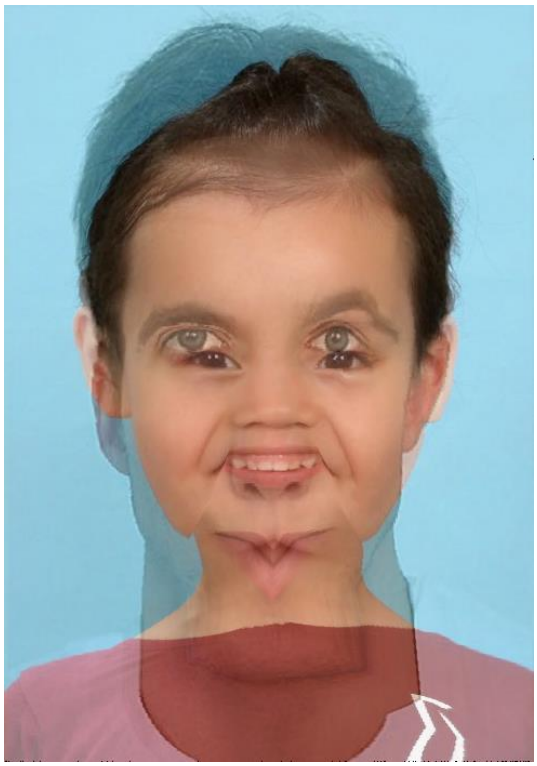
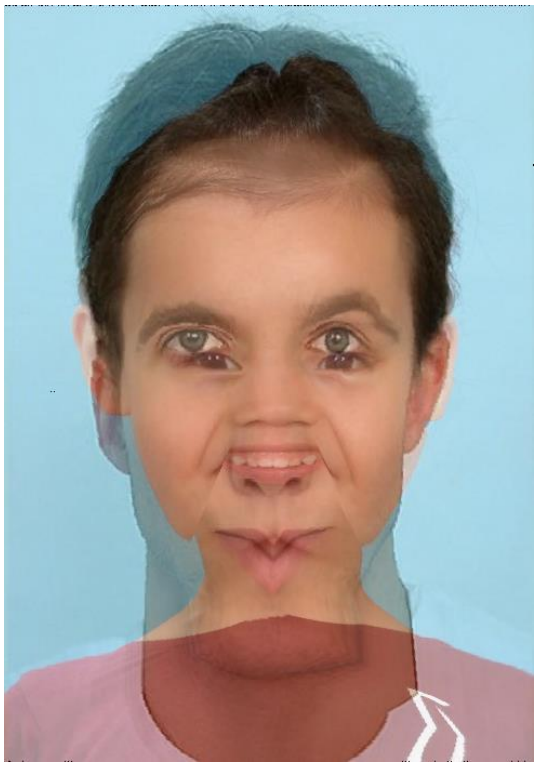
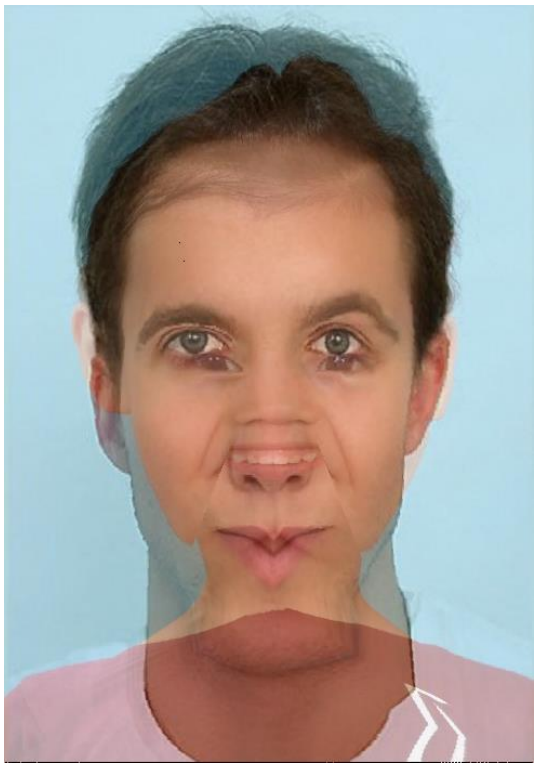
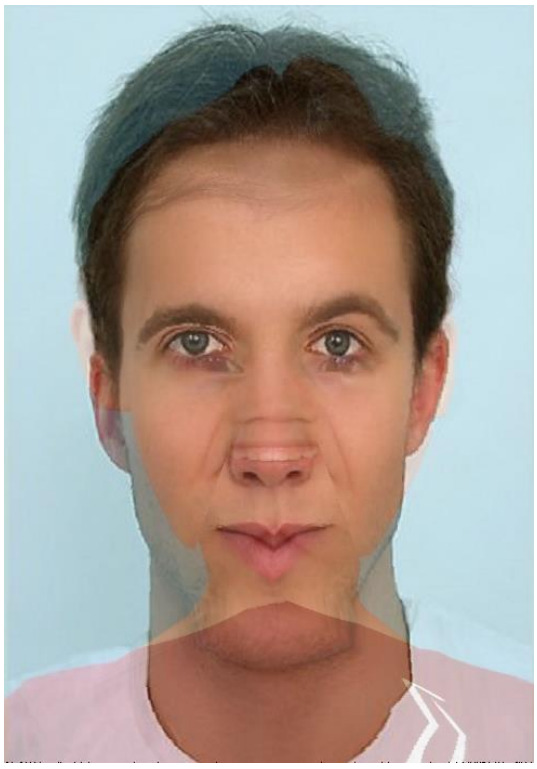
7. 将 13 张图片（1 张 source，1 张 target，11 张中间过程）整合成一个 gif 动图显示效果。

四、测试及分析

直接运行程序，将结果图用 PS 制作成 gif，观看效果。由于 word 文档中无法播放 gif 动图，这里给出 13 张图片分析，gif 在提交的作业包中有。



分析：经过这里的图片展示，和 gif 的显示可以看出，“变脸过程”还算是做的比较成功，基本的变形都做到。

五、难点、思考、总结与体会

难点：本次实验的难点在于计算三角形的仿射变换矩阵，我在这里花费了大量的时间进行演算。在 CImg 中求解方程组没有 Matlab 这么方便，因此为了利用 CImg

提供的 solve 函数，需要对原来的方程稍作变换，这也是利用了仿射变换的特性。同时，在进行线性插值的过程中要尤其注意数据类型，否则很容易出错。
思考：虽然这次实验难度不大，按照老师上课讲的思路做起来比较清晰，但是还是有很多可以优化的地方。第一是人脸特征点的检测。因为还没有学过，所以这里的特征点都是手动输入的，要自己对着图像一个一个找，这样的效果并不是很好。如果要优化，可以使用 dlib 来获取人像特征点。在插值方面，这里用的是线性插值，然而更好的方式是使用双线性插值，这样可以比较准确地反映中间图形的像素值。
总结：这次实验主要是依赖于仿真映射和线性插值两方面的数学知识，从 coding 的角度来说难度不算大，最后做出来的效果也可以接受，但还有优化的空间，总的来说我比较满意这次的实验结果。

六、参考资料

1. 特征点测试数据：
https://github.com/linjiafengyang/ComputerVision/tree/master/Ex4/ImageMorphing_Face/points
2. 仿射变换原理：https://en.wikipedia.org/wiki/Affine_transformation
3. 三角形的仿射变换：https://zhuanlan.zhihu.com/p/23199679
4. morphing 原理：https://blog.csdn.net/qq_31578409/article/details/70049516
5. 判断一个点是否在三角形中：
https://www.cnblogs.com/graphics/archive/2010/08/05/1793393.html