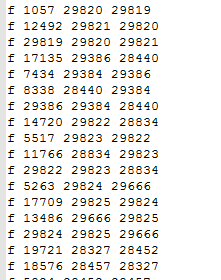
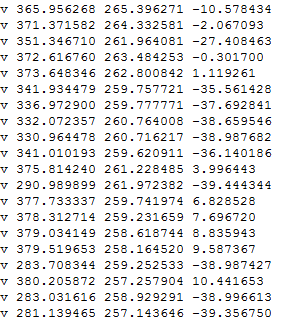
三角网格变形传递算法的研究和实现

之前有机会接触到一篇关于动画模型动作迁移的实现论文，这篇论文被引用很多次，其中的算法使用到了大量线性代数中矩阵的运算。第一次接触到一个纯粹的数学方法能够应用在解决实际问题上，颇感惊奇，所以尝试了研究和实现。

论文的题目为Deformation Transfer for Triangle Meshes，作者为Robert W. Sumner和 Jovan Popovi´c。2004年入选SIGGRAPH的一篇论文。算法的输入给定三个文件s0.obj、s1.obj和t0.obj，描述了三个个由大量三角网格组成的人物面部模型。其中s0和s1是同一个人物的面部不同表情，该算法需要将s0到s1的转换应用到t0上，得到相似变换的t1文件。Obj文件的格式如图。



V代表点，后面接三个代表该点的空间xyz坐标值，f代表小三角面片，后面接三个点的序号。Obj文件可以用meshlab打开，这样由大量点和面组成的面部模型如下图：

s0 s1 t0

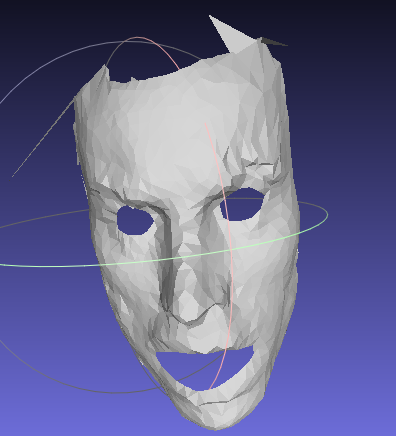
这三张从左到右分别代表s0、s1、t0，s0到s1是同一个人的平静和惊讶的表情，目标是将这样的惊讶应用到t0表情上，得到t0的惊讶表情。每一个面片都涉及大量的数据，由此形成的矩阵是巨大的，利用编程语言进行计算的时候需要花费大量的时间和内存空间。这个算法最吸引人的地方在于，它可以将一个动作应用在许多不同的模型上，可以省略大量重复的工作。并且，在算法中，动作的变换是由纯数学方法（矩阵）描述的，因此一个动作的变换可以抽象出一个矩阵，这一个矩阵又可以应用在不同的模型上，得到不同的结果。

前面说到，一个面部模型由大量的三角面片组成，因此每个三角面片的点数越多，模型越细腻，而且点数和面数有数量上的关系，即面数约等于点数的两倍。简单证明如下，一个三角形每个角平均是60度，一个点周围一圈共360度，所以一个点周围平均可以有6个面。意思是同一个点由6个面共享，而一个三角形有3个点，那么每个面片平均可以得到3/6个点，即半个点。所以三角面片的数量约等于点的数量的2倍。接下来我们研究算法的具体内容：

每个顶点有xyz三个坐标值，组成一个向量，一个三角形有三个向量v1、v2、v3，在这3个点上构造第四个向量v4，计算方法如下：



其实第四个向量v4就是以v1为起点，垂直于三角面片的单位法向量。每个三角面片都有一个v4，v4的个数等于面的个数。可以认为v4就代表了这个三角面。根据仿射变换，将平移的向量d消去，得到，其中大V的计算方法为：，所以可以得到仿射变换矩阵Q的计算方式为。可以根据给出的s0.obj、s1.obj算出每个三角片的Q，而如果将此时的Q直接用于t0的每个三角片上，再加上s0到s1的平移向量d的话，得到的模型一定是不连续的。如下图所示：



这是因为模型中的每个点都是由许多三角面片共享的，如果只根据其中一个三角形的仿射变换就确定该点的坐标，对于其他的三角片来说，这个点的坐标很可能错位。于是出现了上图所示的情况，很多点飞出去了导致模型不光滑。所以算法给了一个约束：



p(vi)是所有共享vi点的三角形的集合。意思是对于同一个点来说，每一个和他相关的三角形应用仿射变换都能到达同一个坐标，这就规定了该模型的光滑。那么如何计算这些点呢？

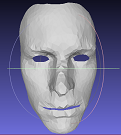
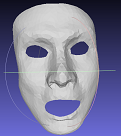
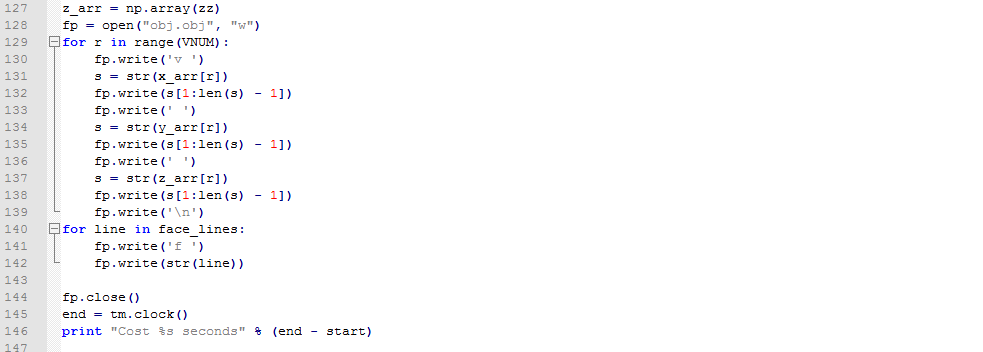
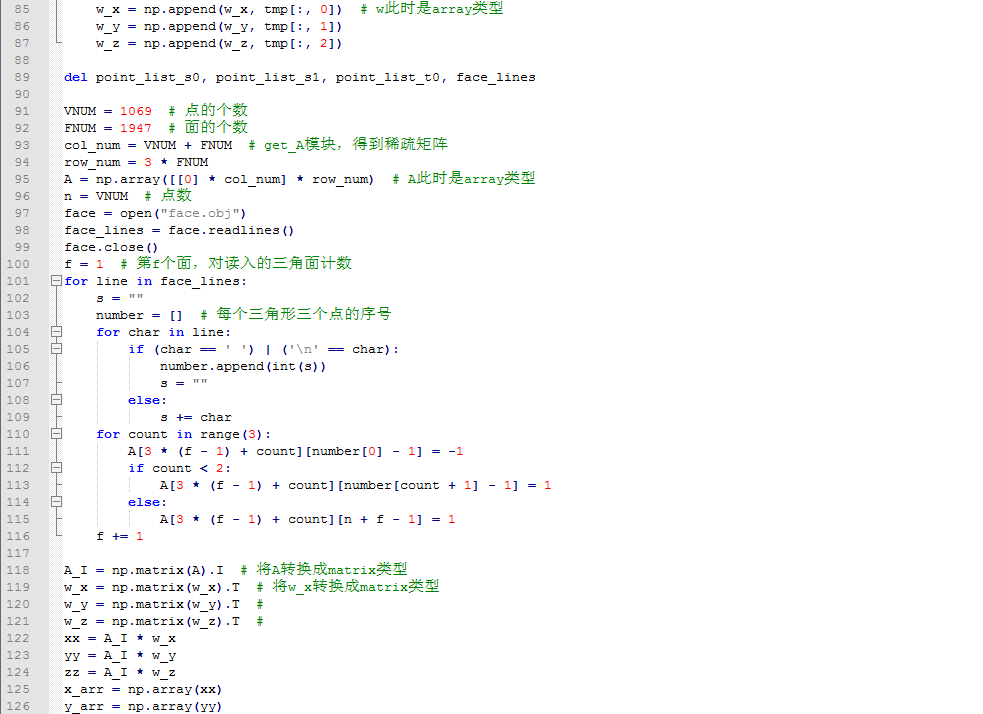
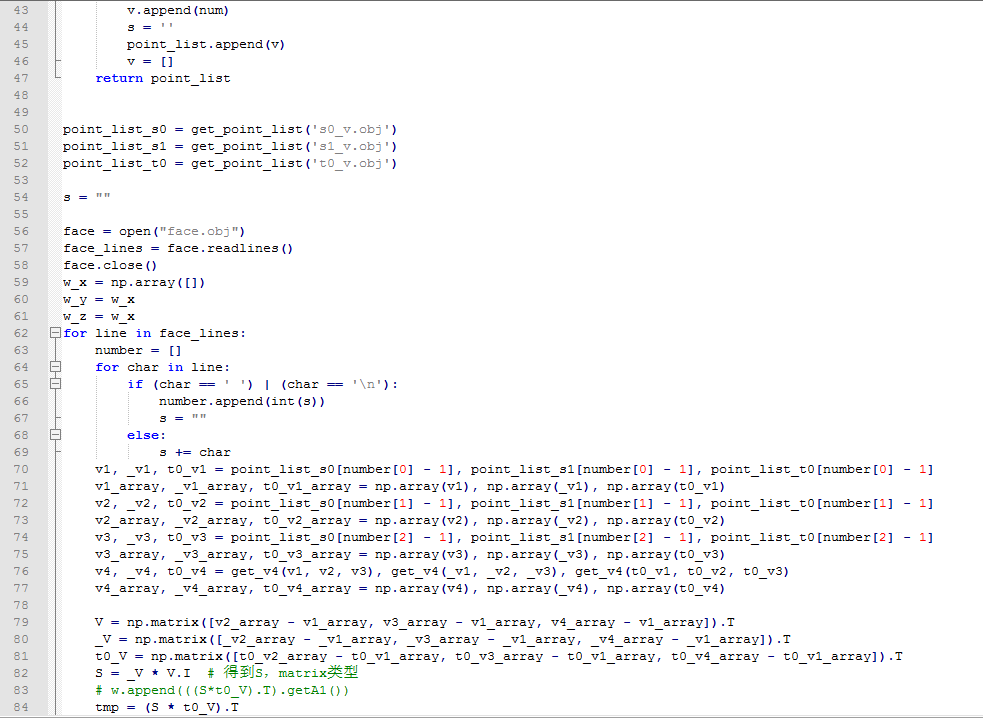
算法给出了一个关键的稀疏矩阵A，并且要求如下式子最小，即等于0：



C是一个向量，由原来的变换运用到目标模型后的结果组成的向量，要求出x~，就必须先求出c向量。将每一个仿射变换的Q应用在t0上，分别抽取每一个三角面片的v2-v1，v3-v1，v4-v1，组成向量即为大向量c。所以有三个大向量，分别由每面片的三个小向量组成。最后根据公式Ax~ = c得出x~的值。所以关键在于这个稀疏矩阵A的给出。

规定A的行数为3 \* faces，列数为vertices + faces，每三行代表一个面。每个三行里面，列数等于这个面的第一个点的序号数的位置为-1。第一行，列数等于第二个点序号数的位置为1；第二行，列数等于第三个点序号数的位置为1；第三行，列数等于面序号数加上总共点数的位置为1。

上述规则有点绕口，其实观察一下就能发现，每三行里面，第一行代表v2-v1，第二行代表v3-v1，第三行代表v4-v1。所以v4在算法中代表的就是整个三角面。我使用python语言实现的，由于原模型中数据量太大，造成计算矩阵的逆的时候，内存空间不够，所以我用了下采样的数据，即相同模型，较少点和面的数据进行计算。点的数量为1069，面的数量为1947。一次计算下来花费的时间为85s左右。最后给出我的代码实现以及实现结果。



s0 s1 t0 t1