图像变形及背后的仿射变换知识

熊章智 2023533146

摘要：本文从实际图像变形案例（原始问题）出发，深入分析图像变形及其背后的仿射变换知识。除此之外，文章将重点聚焦于**还原我的思考里程与理解并进行总结**。

关键词：图像变形　仿射变换　线性代数

1 从原始性问题出发

图像变形已经出现在生活的方方面面中，而图像的变形离不开线性代数。这一点我们都认同，但是我在此想提出一个触及灵魂的问题——为什么变形离不开线性代数，或者说仿射变换？本文中，我想从老师给出的实际案例出发，引出后续的理论分析和进一步推广。

1.1 原始问题

在 cv 领域中，最为出名的便是 Lena 头像照片了。在照片中，我想让 Lena 的头稍微再昂起来一些，显得更为高挑美丽。那么如何实现呢？

Photoshop 便是不错的选择，我选的是 2024 版的 Adobe Photoshop。

首先导入照片（图一），然后 ctrl+T 进入自由变换模式，之后用 alt 键加入节点（图二）。加完三角形节点之后，用 shift+鼠标左键进行拖动与旋转，最终完成了图像的变形（图三）。

图一　　图二　　图三

可见效果还是不错的，至少达到了目的；在上述操作中，主要使用了旋转（局部），加上少量的平移，使得 Lena 有了昂头耸肩的效果。可惜的是帽子部分一直难以调试到最佳状态，可能与我的三角形画点有关。

1.2 启示

在上述案例中，利用老师提示的“三角形节点方式”，我们实现了图形变换，那么为什么是三角形呢？在自由变换模式中鼠标的拖拽、旋转，背后启示究竟发生了什么？为什么一个图形的变化，直觉上面积变大或者变小了，但是为什么图像依然“完整”而非“混乱”？三角形顶点（vertex point）的选取、三角形的划分有什么讲究吗？

这些是我开始思考问题后首先想到的，多半是出于直觉和对线性变化的似懂非懂。接下来将深入分析。

2 深入分析原始问题

通过信导课的信号处理（signal processing）部分，我们知道图像其实是一个包含很多个像素点的矩阵，每一个矩阵元素都有各自的信息，比如说 gray-level（简单来说就是亮度）和 RGB 三原色的亮度。因此想要对图像进行变化，就是要对这些点进行变化。

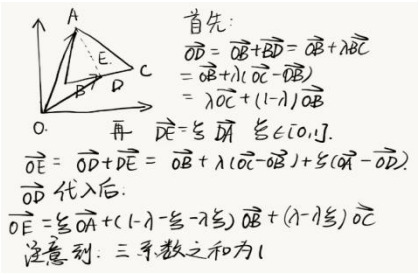
2.1 为什么是三角形？

在老师的提示中，着重论述了仿射变化能由三点处的变化完全决定的事实。

设三角形三个顶点的对应向量分别为$v_1v_2v_3$，则三角形内每一点可以写成唯一（uniquely expressed）的凸组合：

$$v = \lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3$$

这貌似是个显而易见的结论，但是还是要严谨证明一下：



因此证明了：三点围成的三角形之内的任意一个点，都可以通过三个顶点向量表示；或者联系线性代数知识，任意一个三角形内的点的向量，都可以被三个定点向量线性表达。

而如果进行仿射变换，实质上只要对三个定点向量进行变换：

$$w_i = M\, v_i + b$$

因为线性表达的系数是固定的，或者联系线性代数知识，这个映射是双射（Bijective），因此三角形内的每一个点最后都“各自搬到了新家”，这也就解释了变化后的图像为什么不会“混乱”。这将是仿射变换中最为重要的基础性质，仿射变换可以将一个向量组的凸组合（convex combination）映射（map）到另一个向量组的凸组合。

2.2 一定要是三角形吗？

这个问题是十分自然的：为什么我们的划分要是三角形？不能是长方形吗？全能扫面描王上面都是长方形边框让你进行图形变换！其实，选择三角形作为划分的基本形状，是因为在一个长方形中（一般图片都是长方形），只要标出了有限个点，总能够轻松地划分出很多个三角形出来。而且凸组合的唯一性（uniqueness）优点也保存了下来。

2.3 怎么划分三角形？

注意到，如果是矩阵中取 n 个点，然后划分三角形，那么划分的规则是什么？不应该划分有很多种吗（这是很容易想象到的）？

事实上，划分三角形也应该是有规则的：

①：每一条线段都应该属于是三角形的一边

②：线段不能交叉

③：每个顶点至少隶属于两个三角形

④：划分出来的三角形数量应该是最大值（尽可能划分出来三角形）

如此观之，原先我的三角形划分其实是有点问题的，大概率划分数量并没有达到最大数量！不知道这一点对图像结果产生了多大的干扰，但是没关系，毕竟一开始的时候并不知道三角形的划分原来是有讲究的，不是随便乱划分的。

那么问题来了，划分的三角形数量最大是多少？首先，我们应该意识到：最外边的矩阵上的点和矩阵内部的点是分开的，在公式的自变量设置中应该将两类点区分开来；其次，我们假设数量和点的数量关系是呈线性的（不一定是线性关系，但是应该是正比关系）。

了解了上面两点，我们心中有数，开始假设自变量：n 是矩阵内部的点的数量，m 是矩阵边上的点的数量（包括四个顶点），f(n,m)代表最大的三角形数量，那么我不禁开始大胆思考：

如果内部增加了一个点，那么这个增加的点肯定会落在有且仅有的一个三角形上（根据上述规则，不允许内部的点共线），那么原本一个的三角形将被划分为三个三角形，多出来了两个，因此 n 的系数（coefficient）应该是 2；而边上加一个点，也只能落在有且仅有的一个三角形上，将原来的一个三角形划分为两个三角形，因此 m 前面的系数应该是 1。我们又能知道：

$$f(0,4) = 2 \quad f(1,4) = 4$$

因此，我们大胆猜出公式：

$$f(n,m) = 2n + m - 2$$

而事实上，这个公式是正确的，可以利用归纳法证明！

那么对于仿射变换来说有什么要求呢？仿射变换后的一堆三角形，首先是至少应该仍然全部位于长方形的内部，其次不能有三角形重叠的现象。为什么这十分重要？我认为：仿射变换实质是一个可逆矩阵（invertible matrix），作用在坐标向量（coordinate vector）之后实现变换；那么对于之后的状态，我们应该能够作逆操作，那么这时候我们便希望之后状态的三角形划分是符合规则的。

2.4 渐变（morph）

那么仿射变换是一个是双射，那么从 A 状态到 B 状态的整个过程中是不是一个渐变的呢？答案是肯定的。我们规定：

$$u_i = (1-t)v_i + tw_i$$

因为变化过程中，点的移动轨迹是直线（linear），所以任何一个中间状态的点的向量可以利用线段向量表达公式的，也就是上述的公式，其中最大的特点是两个系数之和为 1。

用这种表达是就能实现渐变的过程，如下图（老师提供的照片）：

$t = 0.00$ $t = 0.25$ $t = 0.50$ $t = 0.75$ $t = 1.00$

3 二维仿射变换

3.1 仿射变换的定义

在上面的过程中，我们已经知道了图形变换的核心：顶点向量的映射

$$w_i = M\ v_i + b$$

注意到 M 是一个 2*2 的矩阵，w、v 和 b 都是坐标的向量形式，因此我们能够借此得出真正的仿射变换的定义：

仿射变换是R^2到R^2的一个映射，它的形式是：

$$T(\begin{bmatrix} x \\ y \end{bmatrix}) = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

那么形式已经定义出来了，首先肉眼可见的是：b 向量的功能是实现平移的

那么我们不禁好奇：a b c d 各自的功能是什么？

3.2 各种变换（M_{2*2}）

①：平移（translation）

最直接、最直观的操作，通过 e f 实现。再次不过多赘述

②：缩放（scale）

当我们 b c 不看的时候，不难发现，a d 事实上是对图形进行缩放。其中 a 影响图像横向（x 轴）的缩放，a>1，便是放大，反之则是缩小；d 影响图像纵向（y 轴）的缩放，a>1，便是放大，反之则是缩小。

③：错切（shear）

那么 b c 是干什么的呢？我们不妨从计算过程中发现灵感：

首先矩阵第一行和 v 向量点乘，得到 $ax + by$，这个值代表了之后的向量的新 x 值；根据之前的发现，ax 实现了 x 的缩放，那么 by 呢？由于不同或者相同的 x 值下对应的 y 值不同，因此，不同的 x 会得到额外一种线性的效果。y 值越大的点，能够得到的 by 就越多，反之同理，不难发现，这其实就是错切。理解这一点，c 的作用便是类似了。

总结：b c 实现图形的错切。b 越大，错切程度越大，反之同理；c 亦同理

④：旋转

课堂上学过了 rotation standard matrix 的我们会自信地说：我知道逆时针旋转 θ 的变换矩阵是：

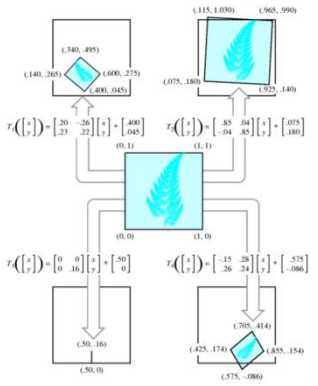
$$\begin{bmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{bmatrix}$$

原本这个公式的推导十分的折磨，那么我们不妨思考一下：这四个三角函数其实对应的都是 a b c d，为什么这四个参数的如此关系能够产生旋转效果呢？这种思考其实反而就是在暗示我们：旋转的实质其实就是缩放和错切的组合！如果这么思考，其实直观上我们能够十分容易认同这个观点。先错切，然后因为略放大的原因，要进行缩小。

3.3 矩阵的可逆性

我们 a b c d 四个参数看似当然可以自由设置，但是事实是这样吗？其实我们考虑一个仿射变换是否“合乎常理”，是不是会把“我能不能变回来”作为一个十分重要的衡量标准？因此，我们要求映射是双射的，要求矩阵 M 可逆。

下面是图像处理先驱 Michael Barnsley 教授著名的仿射变换的例子：



明显注意到：

当矩阵不可逆的时候（第三个例子），变换之后的图像明显不对劲了。这明显不是一个双射，而是一个单射（injective）但不满射（surjective）。我们不能根据变换之后的图反推原来的图是什么。

3.4 M_{3*3}？

有了 $T([\begin{matrix}x\\y\end{matrix}]) = [\begin{matrix}a & b\\c & d\end{matrix}][\begin{matrix}x\\y\end{matrix}] + [\begin{matrix}e\\f\end{matrix}]$ 公式已经很不错了！但是毕竟还是有一个 b 向量后面，我们更希望一个矩阵乘上去，直接实现所有变换。但是通过刚才的分析，很明显，我们无法独立地（换而言之：不是类似于错切那样）对 x、y 值进行改变。

那么怎么办呢？我们不妨升维，实现“降维打击”。我们大胆定义一个公式：

$$T(\begin{matrix}x\\[y]\\1\end{matrix}) = \begin{matrix}a & b & e\\[c & d & f]\\g & h & i\end{matrix}\begin{matrix}x\\[y]\\1\end{matrix}$$

这是一个十分大胆的操作，我们甚至输入的变量都改变了！但是，核心思想是：我只关注 x y，而不关注 1；这个“1”带来的一系列变化就方便我们对 x y 值进行独立的变化了！

就比如第一行和坐标向量点乘：ax + by + e，除了放缩和错切，发现 e 参数恰恰能够帮助我们实现 x 方向的平移；同理，f 参数恰恰能够帮助我们实现 y 方向的平移。至于第三行和坐标向量的点乘结果，我根本不在乎！因此 g h i 的功能在这个场景下，我根本不必关心，因为我们已经实现了基本目标：一个矩阵乘法，实现所有的二维仿射变换。

通常来说为了方便，g h 取 0，i 取 1，使得 gx+hy+i = 1，符合输入坐标向量的格式。

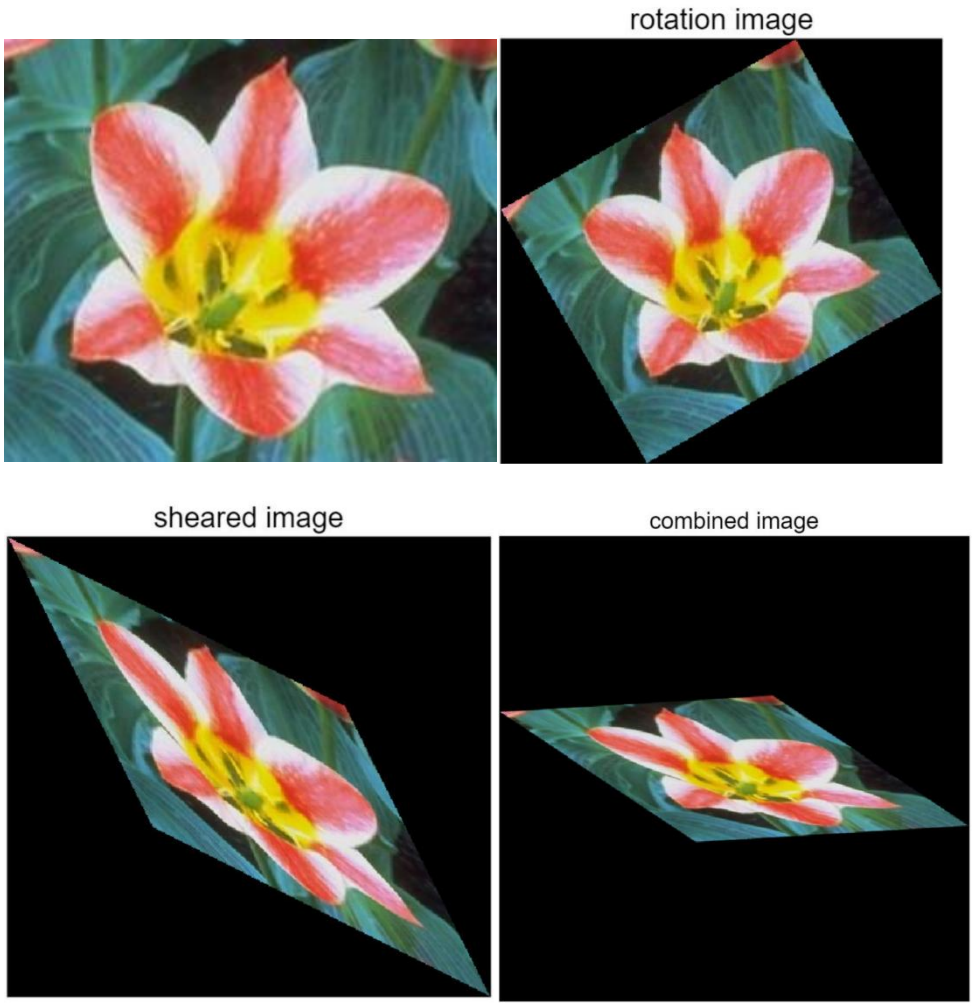
3.5 MATLAB 实现二维仿射变换

在 MATLAB 中，读入的照片本质上是一个矩阵。对矩阵进行乘法就能实现仿射变换。

```
flower = im2double(imread('flower.png'));
theta = 30;
matrix1 = affine2d([cos(theta) -sin(theta) 0; sin(theta) cosd(theta) 0; 0 0 1]);
rotated_image = imwarp(flower, matrix1);
figure
imshow(rotated_image);
title('rotation image');
dx = 0.5;dy = 0.5;
```

```
matrix2 = affine2d([1 dx 0;dy 1 0;0 0 1]);
sheared_image = imwarp(flower, matrix2);
figure
imshow(sheared_image);
title('sheared image');
combined = imwarp(sheared_image, matrix1);
figure
imshow(combined);
title('combined image');
```

我们能够得到旋转（逆时针三十度）、错切、旋转加错切的照片。当然，这个案例是 signal processing homework3 中的题目，但是不妨迁用至此。



4 三维仿射变换

人类向往的总是海阔天空，略显拘束的二维仿射变换也远远不能满足我们的胃口。我们不妨思考：三维中的仿射变换应该如何表达？

4.1 依葫芦画瓢——M_{3*3}

依然，$w_i = M\,v_i + b$ 公式是绝对的核心。那么我们也依葫芦画瓢大胆设出公式：

$$T([y]) = \begin{matrix} x \\ z \end{matrix} \begin{bmatrix} a & b & e \\ c & d & f \\ g & h & i \end{bmatrix} \begin{matrix} x \\ y \\ z \end{matrix} + \begin{matrix} j \\ [k] \\ l \end{matrix}$$

其中 a b c d e f g h i 均是参数，但是这个公式明显更恐怖，让人望而生畏。虽然我们其实是分析过前面这个三阶方阵的，但是明显，这个情况和之前的那一个并不相同，因为引入的坐标向量中，z 不再是 1 了，而是一个真正独立的变量。

但是我们还是抽丝剥茧，慢慢入手：

首先很明显能思考到的是：a d i 这三个值很容易理解。当各自对应的一行和坐标向量进行点乘的时候，ax dy zi 表明 a d i 三个变量就是在 x y z 轴方向上进行放缩。

但是 b e f c g h 呢？看看第一行和坐标向量的点乘结果：ax+by+ez，发现 by ez 代表 x 轴上关于 y、z 周的错切！因此，b e f c g h 的实质是对应坐标轴的错切！注意到一个轴的错切和另外两个轴的错切都有关系，因此有两个参数对应一个轴关于另外两个轴的错切变换就十分有道理了。数学，妙不可言。

那么旋转一般来说沿着的是坐标轴旋转的，因此我们依葫芦画瓢，带着“旋转 = 错切 + 放缩”的思想（有了这个很容易判断对正负号），写出沿着 x y z 轴旋转的 standard matrix：

$$Rotation_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix}$$

$$Rotation_y = \begin{bmatrix} cos\alpha & 0 & sin\alpha \\ 0 & 1 & 0 \\ -sin\alpha & 0 & cos\alpha \end{bmatrix}$$

$$Rotation_z = \begin{bmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

我们还发现了一个小惊喜：沿着 z 轴旋转的公式正好就是二维仿射变换中绕着原点进行旋转的公式。但是我突然想到：为什么二维旋转只有绕着原点旋转，但是三维中可以绕着三个轴转呢？其实，旋转最重要的是抓面！二维欧氏空间（Euclidean Space）中基（basis）只有两个线性无关（Linearly Independent）的向量，在两个基向量相互垂直（orthogonal）的时候，只能形成一个面；但是三维中就能形成三个面。那么我们不妨大胆猜测一下，四维中有几个“旋转公式”？我猜是 4*3/2 = 6 个（四个基，任取两个作面）。当然这只是猜测，不一定正确。

最后就是 j k l 了，很明显是 x y z 轴的平移。

4.2 复杂的旋转

在三维空间中的旋转明显是更为复杂的，但是上述的旋转却只有围绕 x y z 三个轴进行旋转的情景。明显，这个轴可以更为复杂。接下来，我们将进一步探讨围绕过原点的轴旋转的情景：

假设过原点的轴的单位向量表达是（a,b,c），围绕这个轴逆时针转 θ 角。思路是什么呢？我们考虑能不能将（a,b,c）这个向量转到与 z 轴重合，然后将转后的向量绕着 z 轴逆时针转，最后再转回来，恢复（a,b,c）。

那么可以考虑：首先绕着 z 轴转 α 角，然后绕 y 轴转 β 角。其中：

$$\alpha = \tan^{-1} b/a\,;\ \beta = \tan^{-1} c/\sqrt{a^2+b^2}$$

则每一步操作都有：

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix}$$

$$R_z(\vartheta) = \begin{bmatrix} cos\vartheta & -sin\vartheta & 0 \\ sin\vartheta & cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

则应该：$R(\vartheta) = R_z(-\alpha)R_y(-\beta)R_z(\vartheta)R_y(\beta)R_z(\alpha)$

最后所有的式子带入，则仿射变换的矩阵便是：

$$\begin{bmatrix} a^2(1-\cos\vartheta)+\cos\vartheta & ab(1-\cos\vartheta)-c\sin\vartheta & ac(1-\cos\vartheta)+b\sin\vartheta \\ ab(1-\cos\vartheta)+c\sin\vartheta & b^2(1-\cos\vartheta)+\cos\vartheta & bc(1-\cos\vartheta)-a\sin\vartheta \\ ac(1-\cos\vartheta)-b\sin\vartheta & bc(1-\cos\vartheta)+a\sin\vartheta & c^2(1-\cos\vartheta)+\cos\vartheta \end{bmatrix}$$

4.3 大胆拓展——M_{4*4}

能不能把 j k l 舍弃掉呢？还是运用二维仿射变换的灵感，我们设：

$$T(\begin{bmatrix} x \\ y \\ z \end{bmatrix}) = \begin{bmatrix} a & b & e & j \\ c & d & f & k \\ g & h & i & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

惊喜地发现，这种解决方案完全可行。类似地，j k l 直接代表 x y z 轴方向的平移，而 m n o p 四个变量我根本不在乎。通常来说，为了方便，m n o 通常取 0，p 取 1，使得 mx+ny+oz+p = 1，符合输入的坐标向量的格式。

这个矩阵已经十分完美的实现了目标：通过一个矩阵乘法，实现了放缩、平移、错切、旋转的仿射变换。

5 展望高维

在已经了解了三维仿射变换公式的导出之后，那么其实 n 维仿射变换公式其实我们也可以做出一些 ambitious 的猜想：

5.1 参数含义

n 维仿射变换的矩阵（一次矩阵乘法+一次矩阵加法版本）可以建立在 n-1 维的仿射变换矩阵（只有一次矩阵乘法版本）之上；矩阵中M_{i*i}参数代表 x y z 轴的放缩，其余元素均代表对应轴与其他轴的错切关系，“b”向量中的 n 个参数代表各自轴的平移。

5.2 升维矩阵表示仿射变换

n 维仿射变换的矩阵（只有一次矩阵乘法版本）由“一次矩阵乘法+一次矩阵加法版本”得到，最右列加入的 n 个参数代表各自轴的平移（最右列的最下面的一个除外），最后一行用 m 个 0 和 1 个 1 代替；输入的向量坐标最下方加上一个 1（为了方便）。

6 总结

在此次研究中，我从老师提供的实际问题出发，进一步深入思考，并且进行扩展思考。这种研究模式十分值得我学习！在日后的学术道路中，我们可能会遇到一个个实际问题，但是我们不能仅仅是表面上解决它们，而是要不断深入研究，发现问题背后的一系列真理。很感激老师能够提供一个这样的机会，让我体会到学术研究的快乐！

通过本文的论述，我们理解了三角形顶点划分的必要性，以此为出发点，理解了仿射变换的实质；在推导仿射变换的过程中，我们理解了矩阵参数的含义，理解了升维的意义，理解了输入坐标变量底下补个 1 的意义；我们还受此启发，发现了升维前后两个矩阵的联系，大胆猜想了高维中推广的结论。

在仿射变换相关理论的演算时，我结合进了大量课本中的知识点，并且尝试用我自己的思考去理解每一个公式乃至每一个参数的含义，而不仅仅是让我把它们背下来。这在日后的学习中也是十分重要的！死记硬背永远不是硬道理，理解公式才是真本事！理解了公式背后的原理，记忆便只是水到渠成的事情了。

在二维推广至三维、三维推广至无限维的过程中，“推广”扮演了十分重要的角色。大胆推广结论，再小心严谨证明，这种思考模式也值得我用心学习。很多理论不仅仅能用于某一场景，也许这个理论在其他领域、或者更深层次的领域中同样有着深远的影响。

在学习仿射变换知识的过程中，我的搜寻信息能力也进一步提升。网络将海量知识呈现在我们眼前，我们要加以辨别，取其精华、去其糟粕。在本次研究中，我主要使用的资料来自 Google（风评很好），youtube（B 站上一搜仿射变换，全是讲高中椭圆的“仿射变换秒杀高中椭圆大题”，真是令人唏嘘！），以及 https://www.algorithm-archive.org/网站，这个网站的动画对于我理解参数含义真的提供了非常多的帮助！

最后的最后，我想感谢老师能够提供这次带有学术研究性质的机会，让我们得以享受“独立思考，不断钻研”的快乐！在此也借这个机会，感谢助教 TA 们的辛勤付出，感谢老师课堂上无微不至的讲解与答疑解惑！我也要努力汲取这次 project 的经验，为学术道路之后的一次次学术研究不断沉淀！