# CTM仿射变换矩阵分析

# 一、变换矩阵概念

变换矩阵（Transformation Matrix）是线性代数中的一个重要概念，

用于描述对向量或坐标进行几何变换（如平移、旋转、缩放、反射等）的数学工具。

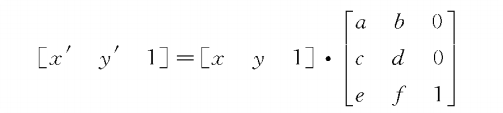
在计算机图形学、机器人学、物理学和工程学中，变换矩阵被广泛应用于处理二维或三维空间中的对象变换。

## 1、 通用矩阵形式‌

* 变换矩阵通常是一个方阵（如 2×2、3×3 或 4×4 矩阵），用于对向量进行线性变换。

以 3×3 矩阵变换为例：

设变换前的坐标是(x, y),变换后的坐标是(x', y'),那么满足公式:



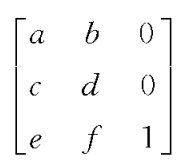
使用代码计算，公式则如下：

x′ = ax + cy + e

y′ = bx + dy + f

## 2、OFD 的矩阵形式

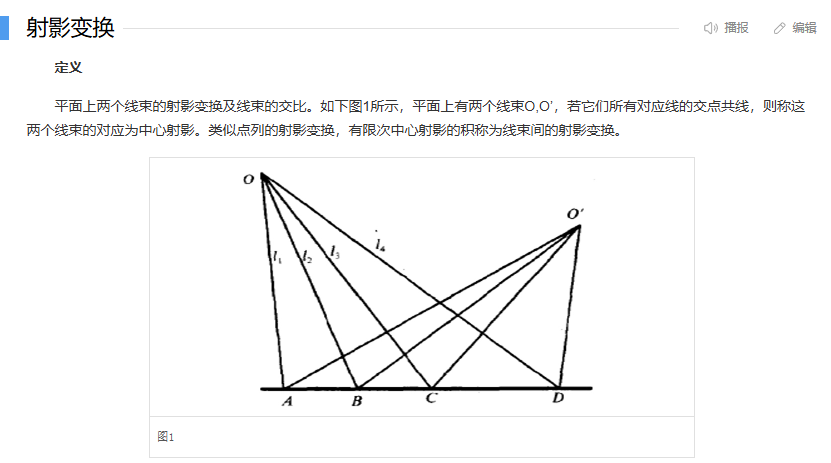
通常是一个 ‌**3×3 矩阵**‌，形式如下：

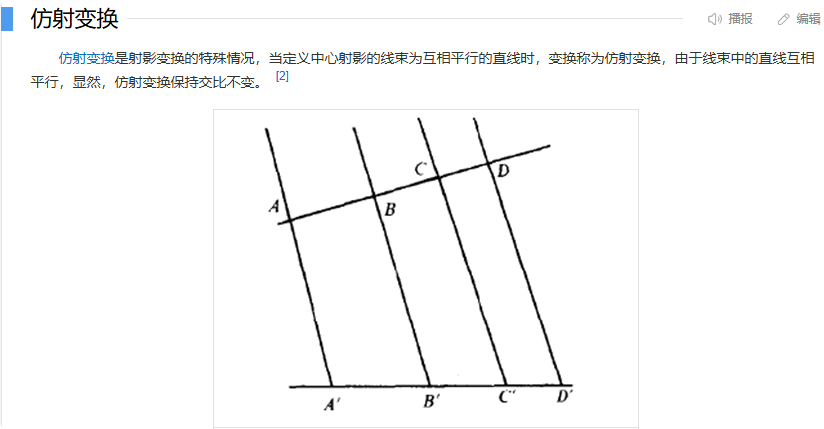


* ‌**前两列**‌（a, b, c, d）：定义旋转、缩放、倾斜等线性变换。
* ‌**第三行**‌（e, f）：定义平移（e 为 x 方向平移，f 为 y 方向平移）。
* **第三列**‌（0, 0, 1）：固定为 0, 0, 1，用于保持齐次坐标的兼容性。

## 3、关于 OFD 矩阵第三列的解释

这涉及到两个概念：**仿射变换矩阵、透视变换矩阵**





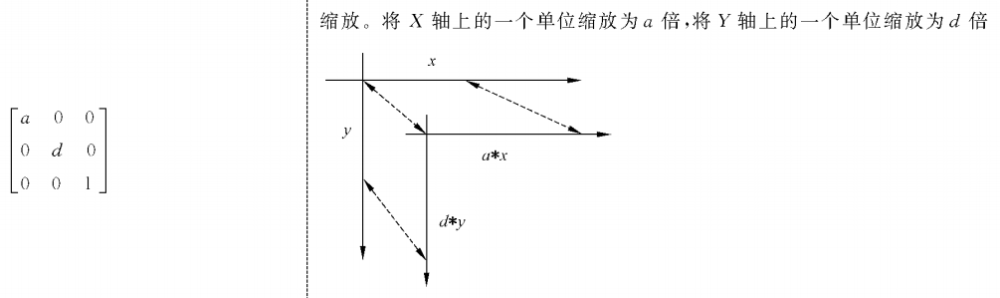
其中第三行固定为 0 0 1，原因如下：

1. **保持仿射性质**这一固定形式确保了变换属于仿射变换，仿射变换的特点是保持直线和平行关系。如果第三行发生变化，那么变换将进入射影变换（3D投影变换）的范畴，这会破坏仿射变换的一些基本性质，如平行性。
2. **齐次坐标的约定**在二维齐次坐标中，一个点通常表示为 [x,y,1]。通过固定第三行为 0 0 1，矩阵乘法后的结果依然保持最后一维为 1，这样可以正确地将齐次坐标转换回二维坐标。如果第三行不是 0 0 1，则需要对结果进行额外的归一化操作，且可能会导致非线性（射影）变换效果。
3. **项目需求**OFD 中的变换只需要实现平移、缩放、旋转和剪切这些常见的变换，因此使用仿射矩阵已经足够。让第三行固定为 0 0 1 是一种常规做法，保证变换的简单性和一致性。

# 二、矩阵变换说明

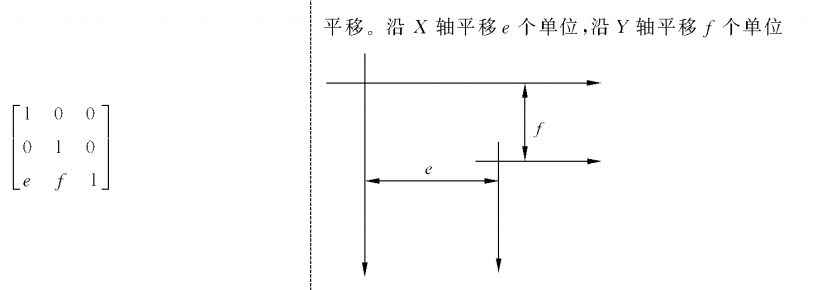
## 1、缩放

a 和 d 的值，是用于控制文字的横纵坐标缩放的，如图：



此时，文字基于原点是不移动的，只是文字或图形变大或变小了

## 2、平移

e 和 f 的值，是用于控制矩阵平移的，如图：

此时，文字从原来的坐标（x,y）,变成了（x+e, y+f）

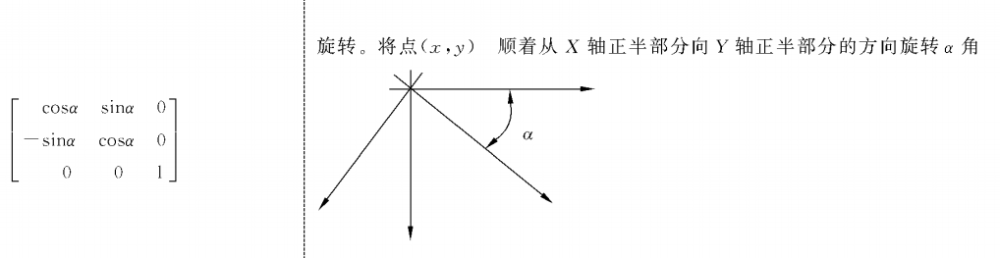
## 3、仅旋转

判断是否是仅旋转，需要满足公式：

| a b | = | cosθ -sinθ |

| c d | | sinθ cosθ |

当 a、b、c、d 不满足上面这种严格关系时，就说明除了旋转之外，还有其它变换成分。



现从真实的 OFD 文档中取一个变换矩阵为例，计算旋转角度：

CTM="0 1 -1 0 3.3084 0"

从矩阵 CTM = "0 1 -1 0 3.3084 0" 来看，前四个数值构成了一个 2×2 的子矩阵，即：

| a b | 实际上这里 a=0, c=-1， b=1, d=0  
| c d |

对于仅含旋转（不含缩放、剪切）的情况，这个子矩阵应满足：

* a = cosθ
* b = sinθ
* c = -sinθ
* d = cosθ

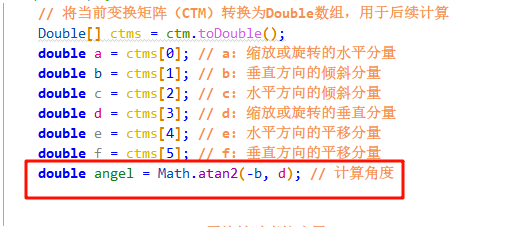
将提供的值带入：

* cosθ = 0
* sinθ = 1

这刚好对应于 θ = 90°（也可以通过计算 arctan2(b, a) 得到，即 arctan2(1, 0) ≈ π/2）。

因此，可以判断该 CTM 矩阵中包含了一个 90° 的旋转，此外矩阵的后两个值（3.3084 和 0）表示平移量。

在 ofdrw 项目中，正是借助Math.atan2(-b, d)公式，看结果是否为 0，来判断是否文本有旋转



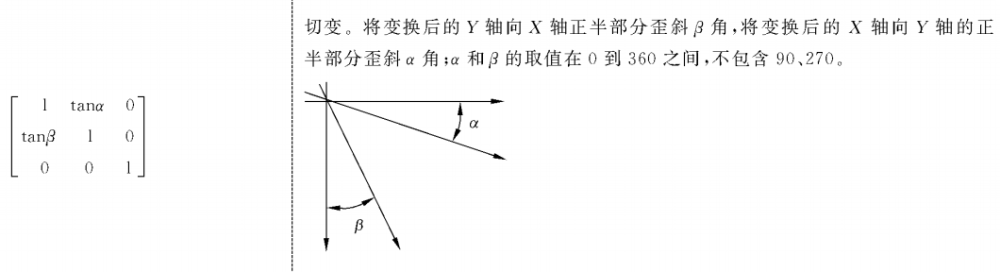
然后代码中，有判断a == 0 && d == 0 的情况,其实就是在考虑当文字旋转90度的排版情况

## 4、切变（剪切、倾斜）

当矩阵中存在剪切时，会导致在旋转和缩放之外，还存在额外的变形。剪切可以理解为一种将形状倾斜的操作，通常计算剪切因子的方法是：

double shear = (a \* c + b \* d) / (scaleX \* scaleX);

如果 shear 不为 0，则说明存在剪切。

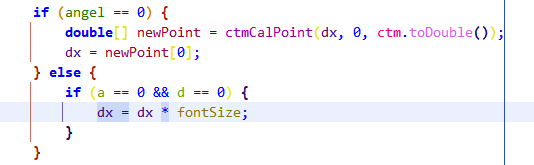


# 三、探究当文本发生旋转时，dx 的取值

观察下图代码，当 angel 不等于 0，证明当前文本发生了角度旋转；

继续判断（a == 0 && d == 0），

其实是计算 cosθ = 0，可得 θ = 90 度；此时可知满足条件时，文本旋转角度为 90 度



也就是，当旋转角度为 90 度，ofdrw 的做法是通过 dx \* fontSize，得到最终的偏移量；

但这种做法有点简单粗暴，且毫无逻辑，事实证明最终的渲染结果也确实不对。

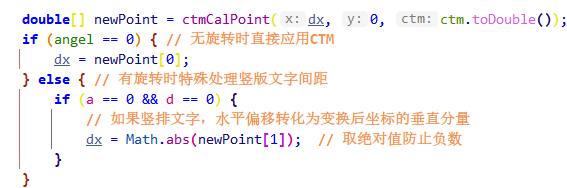
**那当旋转角度为 90 度时，应该怎么得知文字之间的偏移量呢？**

当未发生旋转时，我们可知，可以通过 dx 值与仿射矩阵进行相乘，得到 x 轴方向的偏移量，此时的偏移量就是文字之间的间距；

当旋转角度为 90 时，实际上就是 x 轴与 y 轴发生了映射交换；

此时 x 轴方向的偏移量应该赋值为 y 轴的偏移量;

由此尝试，可修改代码为：



经过验证，字体排版无误。