你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+λ_{1} b_{1}+\cdots+λ_{k} b_{k}$。

在这个表达式中，$(λ_{1}, \cdots, λ_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、

b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

> 友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array} {cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array} {llll}
1 & 0 & 0 & x \\
0 & 1 & 0 & y \\

0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array} {llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array} {cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array} {cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array} {cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array} {llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array} {llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array} {cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

> 友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+λ_{1} b_{1}+\cdots+λ_{k} b_{k}$。

在这个表达式中，$(λ_{1}, \cdots, λ_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\
0 & 1 & 0 & y \\
0 & 0 & 1 & z \\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\

0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我们要讲的内容是“仿射空间”。

一听到仿射空间，你也许会觉得很奇怪，之前我们说过了线性空间，现在怎么又来一个空间？特别是“仿射”这个词，它有什么含义？它和线性空间的区别和联系又是什么呢？这我们就要从线性空间开始说起了。

我们知道，线性空间中有向量和标量两个对象，而仿射空间与线性空间的区别就在于它又加了一个对象，那就是“点”，而且它们的运算规则也不相同。比如，在仿射空间中，点和标量之间没有定义运算；向量和点之间有加法，运算结果是点；点和点之间有减法，运算结果是向量。

所以，仿射空间可以说是点和向量的空间，而且可以被看成是一个没有原点的线性空间。那你有没有想过，我们为什么要研究仿射空间呢？

那是因为仿射空间在计算机图形处理中有着极其重要的地位。在线性空间中，我们可以用矩阵乘向量的方法表示各种线性变换。但是，有一种常用的变换却不能用线性变换的方式表示，那就是**平移**，一个图形的平移是非线性的。为了表示平移，以及方便现实世界的描述，就需要使用仿射空间。

仿射子空间

和向量子空间一样，我们现在需要把注意力转移到更有实践意义的仿射子空间上。仿射子空间在计算机科学中的运用主要体现在**计算机图形处理**中，比如：图形的平移、缩放和旋转等等。

如果我们把Ax看成是线性，那么$Ax+b$就是仿射，其实就是多了一个平移。也就是说，我们完全可以把仿射子空间看成是线性子空间的平移。

这样理解很容易，不过，我们还是要看看数学上对仿射子空间的严格定义：V是一个向量空间，U是V的一个向量子空间，x_{0}是V中的元素，那仿射子空间L就等于：$L=x_{0}+U:=\left\{x_{0}+u: u \in U\right\}$。

这里的U叫做方向，x_{0}叫做支撑点。仿射子空间在实数三维R^{3}中有点、线和面，它们在坐标系中都是不过原点的。

仿射子空间也经常由参数来描述。因为，我们能通过参数，把表达式组合成方程形式，这样更有助于计算。假设，有一个k维的仿射子空间L，它可以表示成：$L=x_{0}+U$。如果U有一个有序基$(b_{1}, \cdots, b_{k})$，那么，每一个属于仿射子空间L的元素x，都能够表示成：$x=x_{0}+\lambda_{1} b_{1}+\cdots+\lambda_{k} b_{k}$。

在这个表达式中，$(\lambda_{1}, \cdots, \lambda_{k})$是实数参数，我们把这个表达式叫做“参数方程”，而$(b_{1}, \cdots, b_{k})$就是方向向量。

现在我们来看看，仿射子空间在不同维度中的几个例子，让你能从几何角度更了解仿射子空间。

一维仿射子空间，也叫做“线”，参数方程是：$y=x_{0}+\lambda b_{1}$。也就是说，一条线是由一个支撑点x_{0}和一个方向向量b_{1}定义的。

二维仿射子空间，也叫做“平面”，参数方程是：$y=x_{0}+\lambda_{1} b_{1}+\lambda_{2} b_{2}$。也就是说，一个平面是由一个支撑点$x_{0}$和两个线性独立的向量$b_{1}$和$b_{2}$定义的。

$n-1$维仿射子空间，也叫做“超平面”，参数方程如下。

$$y=x_{0}+\sum_{i=1}^{n-1} \lambda_{i} b_{i}$$

在这个参数方程中，$(b_{1}, \cdots, b_{n-1})$是$n-1$维子空间的一个基。也就是说，超平面是由一个支撑点x_{0}和$n-1$个线性独立的向量$(b_{1}, \cdots, b_{n-1})$所定义的。

仿射映射

空间说完，必定会来到动态部分，对应到今天的内容就是仿射映射了。仿射映射和向量空间之间的线性映射是类似的，很多线性映射的特性也能使用在仿射映射上。现在，我们试着来定义两个仿射空间之间的仿射映射。

现在我们有两个向量空间V和W，一个V到W的线性映射ϕ，以及一个属于向量空间W的向量a。

$$
\begin{aligned}
\varphi: & \ \rightarrow W \\\
& x \rightarrow a+\phi(x)
\end{aligned}
$$

上面这样的映射就是V到W的仿射映射，其中，你可以看到x元素是通过线性映射函数ϕ和平移向量a进行仿射变换的。

事情就是那么简单，看上去就是一个线性映射加上一个向量，或者从几何角度说，就是做了一次线性变换后，进行了一次平移操作。而更专业一点的说法就是，每个V到W的仿射映射，都是“一个V到W的线性映射”和“一个W到W平移”的组合。

从这里你也可以判断出，仿射映射是保持了原几何结构和维度不变的。

接下来，我们再来看看三维世界中物体的仿射映射，这就包括物体的平移、缩放和旋转了。在几何空间中，物体的平移、旋转、放大缩小，这类操作如果从局部坐标系来看，就是在局部坐标系定义的点，或者向量x经过变换后，得到点或向量y。这类变换可以用公式$y=Ax+v$来表示，其中A就是$3×3$矩阵，v就是三维向量。

当然，我们还可以用矩阵来表示这类变换，也就是仿射变换矩阵和向量乘，这和公式$y=Ax+v$达到的效果是一样的。不过，由于仿射变换矩阵是在实践中经常用的方式，所以我们要来具体看看仿射变换矩阵。

仿射变换矩阵

在三维世界中，物体的平移、缩放和旋转，这些操作其实都可以放到一个$4×4$的矩阵中，并且统一用这个矩阵与向量的乘操作来进行物体的变换，或者说向量的空间变换。这个$4×4$仿射变换矩阵是下面这样的。

$$
A^{\prime}=\left[\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\\
a_{21} & a_{22} & a_{23} & a_{24} \\\
a_{31} & a_{32} & a_{33} & a_{34} \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

其中，a_{11}到a_{33}的$3×3$矩阵就是变换公式$y=Ax+v$中的A矩阵，表示的是线性变换。而右上角的a_{14}, a_{24}, a_{34}表示的是平移变换，也就是变换公式中的v向量。右下角的数字表示的则是整体缩放，现在它是1，也就意味着不进行整体缩放。

在计算机图形图像处理中，仿射变换尤其重要，那是因为它能保持变换后的共线或共面性。也就是说，线段变换到线段，还是一条直线，变换前的线段中心点就是变换后的线段中心点。同样，三角形变换后，原三角形的重心还是变换后新三角形的重心。

现在，我们拿一个三角形来举例，因为三角形从计算机图形图像上来说是最基础的图形，是组合成其它多边形的基础。如果我们要变换一个三角形，只要对三个定点a、b、c进行仿射变换就行了，对于三角形边上的其它点，变换后还是一样会在边上，这样计算量会极大地降低，变换效率就提高了。

接下来，我们把具体的这几个仿射变换矩阵：平移、缩放和旋转，都单独拿出来，看看它们长什么样。

平移矩阵

我们先来看看$4×4$的平移仿射变换矩阵$A^{\prime}$，你有没有注意到矩阵中的x、y、z？它们就像公式里写的那样固定在矩阵的右边，定义了矩阵在三个轴方向上的平移距离。

$$
A^{\prime}=\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

缩放矩阵

接下来，我们再来看看缩放仿射变换矩阵$A^{\prime}$，这里x、y、z的位置产生了变化，固定在了矩阵的对角线上，定义了矩阵在三个轴方向上的缩放大小。

$$
A^{\prime}=\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

旋转矩阵

最后，我们来看看旋转仿射变换矩阵$A^{\prime}$。这里有三种情况：绕x轴旋转、绕y轴旋转、绕z轴旋转。

绕x轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕y轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & 0 & \sin \theta & 0 \\\
0 & 1 & 0 & 0 \\\
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

绕z轴旋转的矩阵：

$$
A^{\prime}=\left[\begin{array}{cccc}
\cos \theta & -\sin \theta & 0 & 0 \\\
\sin \theta & \cos \theta & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在3D图形实践中，一般仿射变换矩阵上的操作都是可以叠加的。也就是说，我们可以通过连续的矩阵乘，来完成一系列的对象变换。比如，如果我们的对象要先平移，再缩放，最后再绕x轴旋转，那么我们就可以通过矩阵的三连乘来表达这个变换。

$$
\left[\begin{array}{llll}
1 & 0 & 0 & x \\\
0 & 1 & 0 & y \\\
0 & 0 & 1 & z \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{llll}
x & 0 & 1 & 0 \\\
0 & y & 0 & 0 \\\
0 & 0 & z & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
$$

在这里，我也给你推荐一个数学库作为研究使用。因为工作的缘故，我们做的事情和WebGL有关，所以我推荐的是一个前端TS实现的数学库TSM，你可以访问GitHub来了解。TSM很好地封装了仿射映射，刚才的仿射变换矩阵的叠加操作也是封装好的，比如：平移（translate(vector:vec3):mat4）；缩放（scale(vector:vec3):mat4）；旋转（rotate(angle:number,u:vec3):mat4）。

本节小结

这一节课的重点是仿射空间和仿射映射。有关仿射空间，你一定要掌握的是仿射子空间在不同维度中的几个例子，特别是$n-1$维仿射子空间，也叫做“超平面”。因为超平面在机器学习的分类算法中很重要，比如SVM支持向量机的二分类算法就会用到它。

而在仿射映射中，仿射变换矩阵是重点。因为在3D计算机图形图像处理中，它能够被用来进行物体的平移、缩放和旋转。而且，在计算性能方面，仿射变换矩阵可以极大地降低计算机的运算量，提高变换效率。

线性代数练习场

好，今天的练习时刻到了，不过今天的练习会有一些特别。刚刚我推荐了前端TS实现的数学库TSM，这里我再推荐另一个在Python中广泛使用的库OpenCV。

我希望你能够使用它对一张JPG图片做一个简单的仿射变换：平移，平移$(50, 20)$。JPG素材如下图所示。

友情提示：你可以使用CV2来读图片，NumPy的shape来获取图片的行和列数据，再使用NumPy的float32产生仿射变换矩阵，最后使用CV2的warpAffine来完成图片的平移操作。我贴了部分代码在下面，其中的仿射变换矩阵的产生和仿射变换这两行代码是需要你来完成的。

```
import cv2
import numpy as np

//读取图片
img = cv2.imread('09.jpg',0)
rows,cols = img.shape

//这里是你要完成的代码

//显示原图片和仿射变换后的图片
cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindow
```

当然，你也可以用其它的库来完成，比如TSM，这些都没问题，你可以自由发挥。

欢迎在留言区贴出你的代码和最后的输出结果，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。