**ASSIN1**

1文件输入（启动程序已完全实现）

2用户接口 主要是使用鼠标可以旋转和缩放模型 （启动程序已经完全实现）

3曲线 主要实现Bezier曲线和样条曲线 正确计算沿着曲线的局部坐标并显示出来 还应该使用不同的颜色单独渲染N,B,T向量，可在示例代码中的对应方法中实现；（提示：可以使用一个简单的变换来实现一种曲线向另一种曲线的转换）

4曲面 生成和显示旋转曲面（使用xy平面上的一条曲线 绕着y轴旋转）要用两种模式来显示，一个是现况模式，绘制曲面上的顶点法线，一个是顺滑的着色来显示曲面。

**疑问：**

为什么知道每个控制点的V,T,N,B向量就可以绘制曲线？

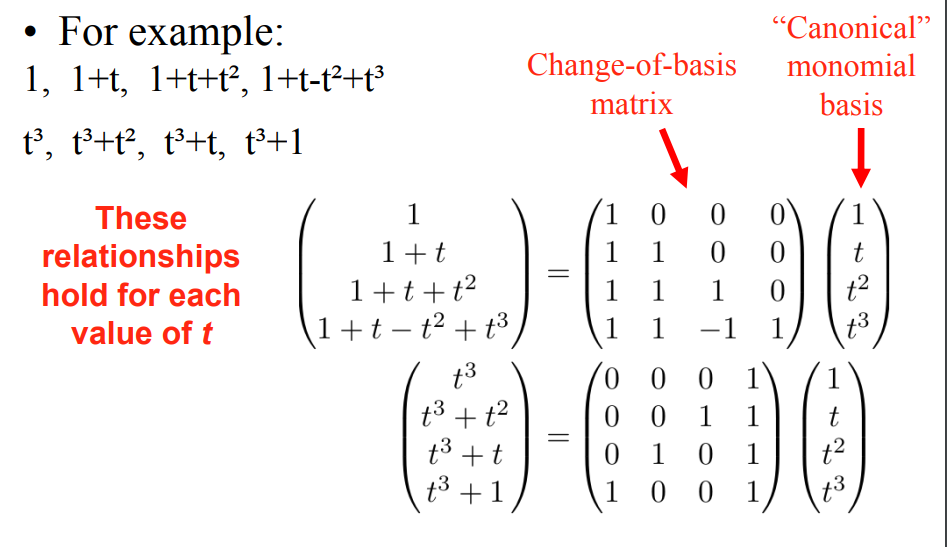
Cubic Bézier Curve4个控制点就可以定义唯一一条曲线 且曲线上的点满足一个方程：

P(t) = (1-t)³ P1+ 3t(1-t)² P2 + 3t²(1-t) P3 + t³ P4

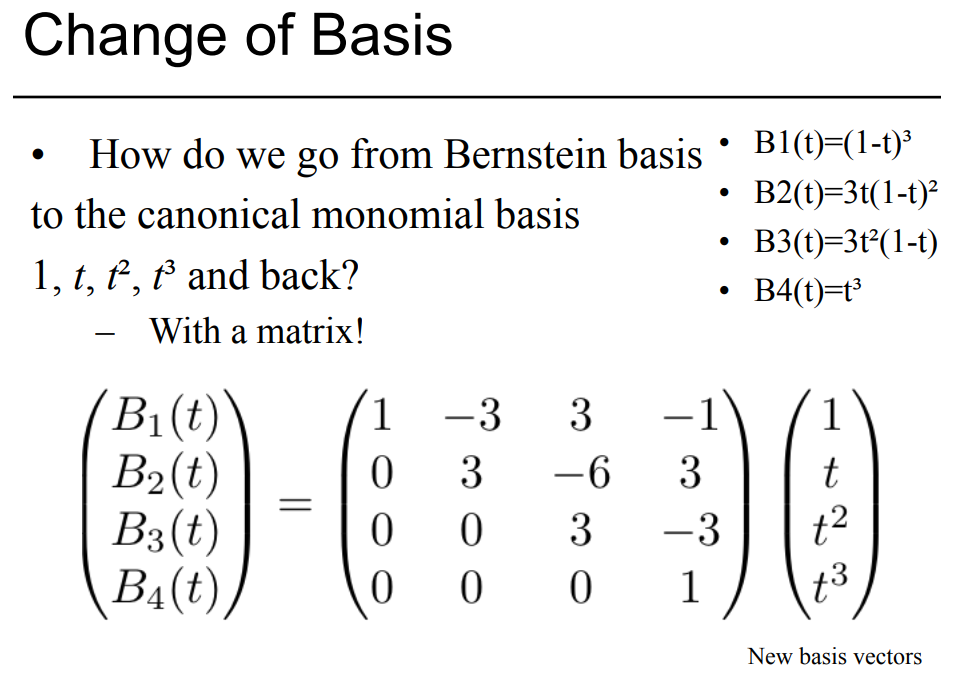
这是一个线性表达式，表示了每一个控制点的权重，0时刻表示P1控制点,1时刻表示P4控制点；

将其划分四个基部分： B1(t)=(1-t)³；B2(t)=3t(1-t)²；B3(t)=3t²(1-t)；B4(t)=t³

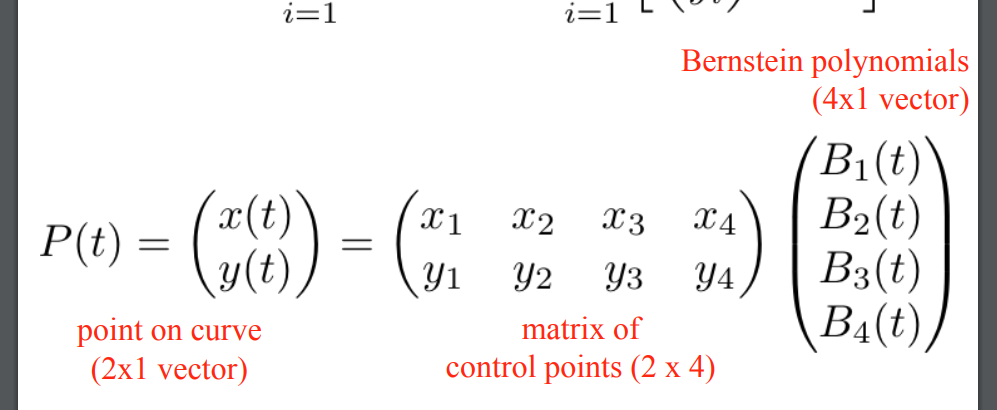
这也是一个三次方的多项式，可以写成(矩阵\*基)的形式，例如：



根据伯恩斯坦多项式，可以改写成如下表达式：



所以最后这个P(t)函数可以表达成如下形式：



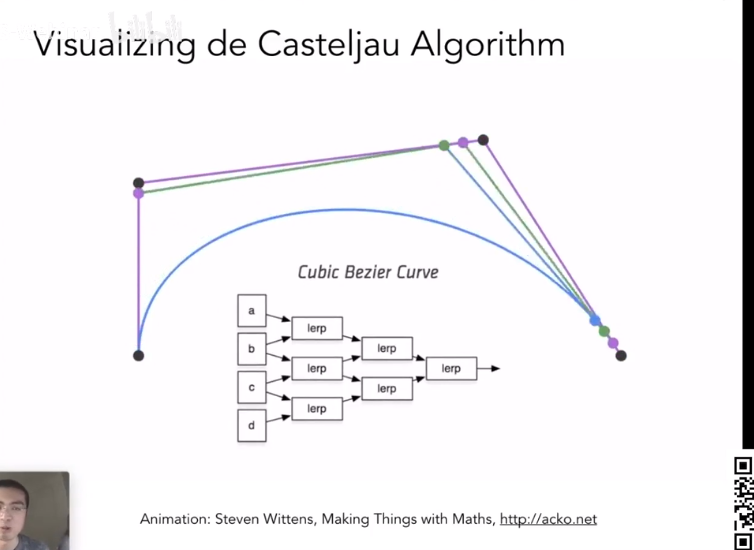
也就是X(t) = (1-t)³ X(P1)+ 3t(1-t)² X(P2) + 3t²(1-t) X(P3) + t³ X(P4)

Y(t) = … 类似

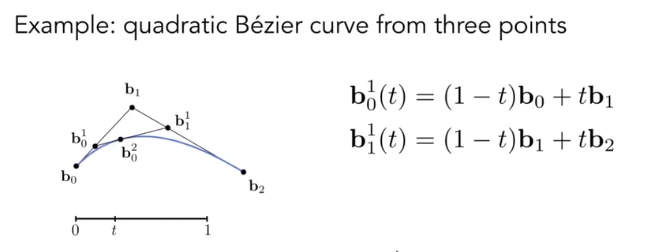
回到作业，给定n个控制点坐标和m的细分步骤(m段线段)，也就是m+1个新的点，然后recorder就可以根据相邻的两个点连成的线段，绘制出一条近似的曲线；

使用De Casteljau Construction算法可以获取4个控制点定义的cubic Bezier曲线的重点位置，如果仅仅是绘制曲线，使用算法计算各个点的位置已经足够，但是如果要实现动画模拟或者表面模型，这些信息不足以实现，所以需要计算V,T,N,B等向量；

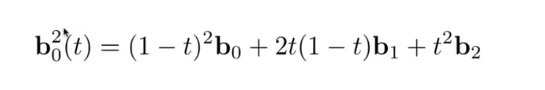
注意：使用De Casteljau Construction算法可以获取任意多个控制点定义曲线中任意时刻的点的位置，即使用这个算法可以绘制任意控制点定义的一条贝塞尔曲线，恰好作业不是传递了一个steps嘛，其实也就是贝塞尔曲线的精度；其实就是将时间划分为steps段，然后在每一个时刻t，去应用De Casteljau算法计算曲线上该时刻的点的位置，计算出steps



插值公式：

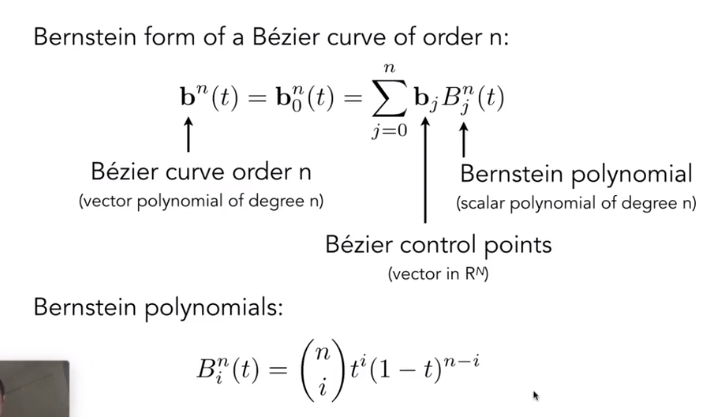


然后推导得出：

2层（即3个控制点）的贝塞尔曲线的多项式表达：  


3层（即4个控制点）的贝塞尔曲线的多项式表达：（即3次方贝塞尔曲线）：

总结，n阶的贝塞尔曲线(n+1)个控制点：



得出这个公式之后，就可以计算n个控制点的贝塞尔曲线，在每一个时刻t的位置，速度也可以直接对这个公式求一阶导数；二阶导数同理；

参考这个文档：

<https://www.jianshu.com/p/7c56103dcf63>

最后的计算方法：

V: 将每一项分成三部分，a,b,c，分别代表常数部分，(1-t)的幂次方和t的幂次方，从0项到n项，第i项的三个系数分别是：

a（使用二项展开式计算每一项）；b=(n-i)；c=i；

T: 和上面一样的方法，然后对t求导即可;



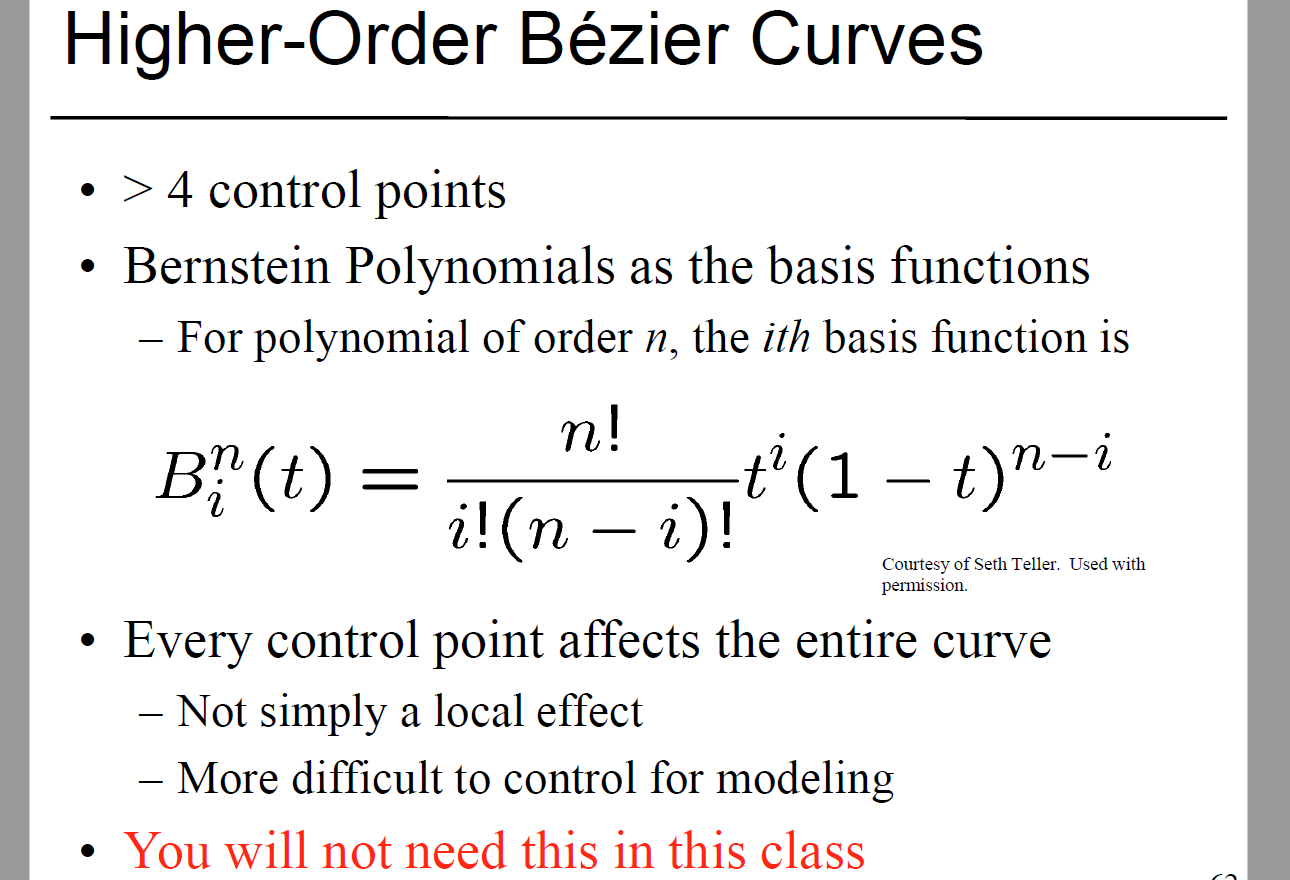
N和B采用特殊的技巧来计算，由于我们是离散化，细分的计算，所以可以递归的使用前一个点来计算当前点的N和B，即：



我们已经计算出当前点的Ti，也可以知道前一个点的B，即Bi-1，因为最起始的B可以任意指定，所以我们可以顺序递归计算；

**总结：**

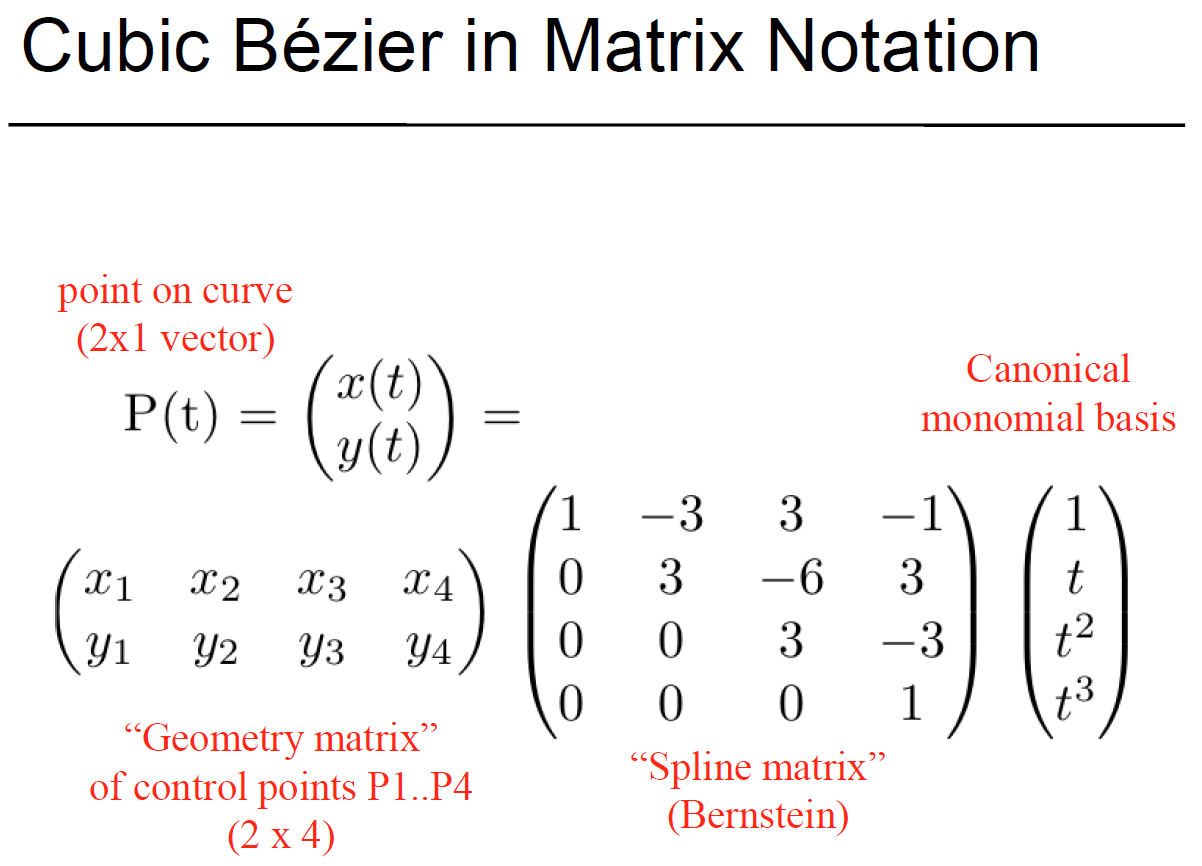
给定一系列控制点，要绘制一条贝塞尔曲线，如果仅仅是绘制曲线，其实可以直接使用伯恩斯坦多项式来计算，曲线上每一时刻的点的位置：



也可以使用De Casteljau Construction算法来递归地计算每一个时刻曲线点的位置；

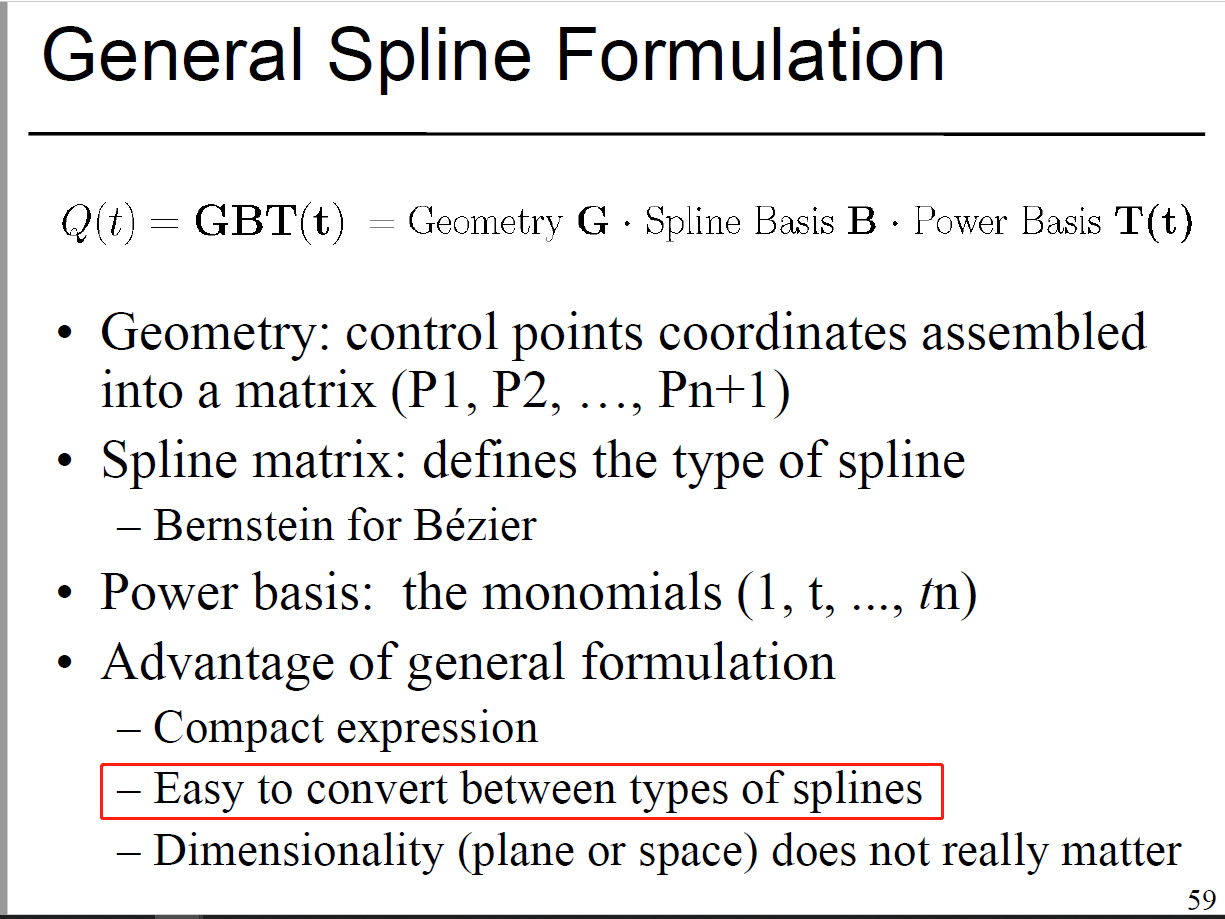
但是对于动画模拟等，光有曲线上的点还不够，我们还要知道曲线方向、方向趋势等信息，所以需要计算每一个点的切线向量，法线向量，副法线向量等，所以作业提出了要计算V, T, N, B；

对于另外一种曲线，即B样条曲线，可以根据课件里的矩阵表达式，来通过基矩阵的改变来转换：



因为将伯恩斯坦多项式拆开之后，可以写成t的线性组合的形式，也就是把这部分拆为规范基（1，t，t2，t3）和样条矩阵（即拆开后t的系数组成的矩阵）；最后再乘上几何矩阵部分，也就是控制点的坐标；

通用表达式：



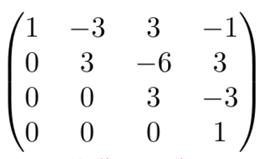
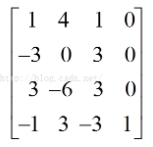
所以两种曲线的转换应该就是通过这个表达式，主要是改变样条矩阵；

推导：

G\*B\*T(t) = Bezier曲线上的一个点 -> funtion(P, steps) = Bezier曲线

G\*B’\*T(t) = Bspline曲线上的一个点 ->但是我们现在没有计算B样条曲线的方法

B和B’是已知的（3阶）

和

想到如果能用一种方式 将B样条曲线表现为Bezier曲线的方程，那么输入数据(变换后)也能得到一条贝塞尔曲线，但是这条贝塞尔曲线实际上是一条Bspline曲线

G \* B’\*T ⬄ G’\* B \* T ; 二者等价 那么funtion(P’, steps) = Bezier曲线’= Bspline

再观察，其实很简单，G \* B’\*T = G \* B’\* (B-1 \* B )\* T = ( G \* B’\* B-1)\* B \* T

* P’= G \* B’\* B-1只需要对输入的点坐标做这个变换就可以了

生成和显示旋转曲面：

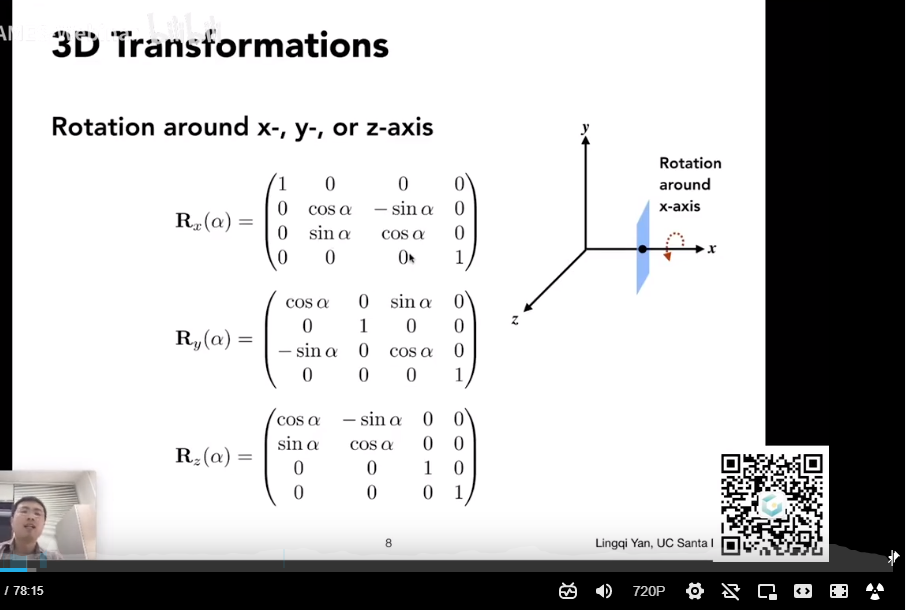
1.首先均匀采样旋转角度 即360°分多少个采样点 10°左右感觉差不多 然后计算曲线上点的相应位置；

2.顶点无法定义曲面，所以需要把法线也进行一个变换，但是记住我们此时使用的变换矩阵，要进行一个处理，也就是所谓的法线矩阵；用M的左上角3x3子矩阵的转置来变换；

复习一下转换那节课程的知识

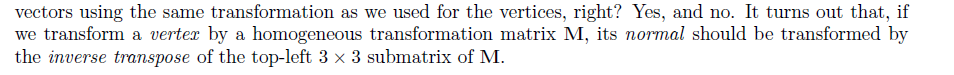
3.曲线的法线方向，需要进行反转，因为我们定义曲线上的法线方向，是始终指向沿着曲线行进方向的左边，那么也就是从上往下走的曲线，这样，当我们绕着y轴逆时针旋转曲线生成曲面的时候（旋转形成了一个圈），圈上的法线方向始终指向圆心，和我们定义的左手定则方向一致；所以这个时候我们计算出来的法线，就需要反转法向量的方向。

4.还需要生成三角形来表示平面，在相邻两条剖线之间，重复构建三角形；注意，在openGL中，指定三角形的顶点时，必须以逆时针来指定，否则会产生错误；



使用变换矩阵，进行旋转的变换；

但是法线的变换不能直接使用这个变换矩阵，要使用经过修正的法线变换矩阵：



反正经过推导，结果就是变换矩阵的逆矩阵的转置；又由于平移不影响法线，所以可以把齐次坐标中的平移那一列移除，只变换左上角3X3的旋转缩放部分；

但是由于我们本次作业是绕着y轴旋转的，并且没有位移和缩放（主要是没有缩放）,所以变换矩阵仅仅是旋转矩阵，而旋转矩阵本身是正交的(A\*AT=E)，所以左上角3x3矩阵取逆再取转置之后还是它本身(正交矩阵的逆=转置)，所以对于本作业而言不需要特殊处理。

**其他补充：**

目前有的是一条贝塞尔曲线 上面有n个点，包括其位置和法线信息；

steps传来的是90，意味着旋转细分为90个部分，也就是90条曲线，相邻曲线相隔4°；

（1）于是要将这条曲线绕着Y轴旋转90次，每次4°，然后记录这条旋转曲线的位置和法线信息；

（2）将所有的法线翻转；

（3）递归地记录相邻曲线4个点构成的两个三角形（逆时针）

通用圆柱体：

遍历扫掠曲线的每一个点，然后跟轮廓线第一个点的位置计算偏移值，然后把轮廓线上的所有点都位移到扫掠曲线的这个点上，相当于将这条轮廓线移动到扫掠曲线上的一个点，然后对下个扫掠曲线上的点也进行如此操作，知道扫掠曲线上的每一个点的位置都有一条轮廓线，然后对相邻轮廓线构造三角形网格即可！

贝塞尔公式：

<https://www.jianshu.com/p/7c56103dcf63>