**1 曲线和曲面基础**

**1.1 隐式和参数表示**

两种曲线和曲面的表示方法：隐式表示和参数表示方法。

**（1）参数表示方法不唯一**

比如有



那么如果令



因为



所以原式也可以写为



上式的*u*和*t*的取值范围不需要限定，倍角公式等都是适用于定义域在实数域上的三角函数。

**将不同的参数选为时间参数**，以求取速度与加速度，最终得到的结果可能会不一样，比如说将*u*看做时间参数



此时的速度方向变化，但是速度的大小不变，这种情况被成为均匀参数化，但如果将*t*看做时间参数



此时的速度方向变化，**速度的大小也会发生改变，此时的参数化是非均匀的。**

**（2）一些比较重要的问题**

**①参数化引起的奇异性**

考虑一个球上的点，用与*z*轴夹角*u*和与*x*轴夹角*v*来表示有



其偏导矢为



对于球面的南北极，也就是*u*为0或者*pi*的位置有



那么此时用切矢叉乘来获得法向矢量的方法是不管用的了，因为计算出来为**0**向量，但球的南北极的法向又是确实存在的，而且在几何上来说，球面的南北极和其他点本身并无不同，是因为坐标的选取和参数化的定义才引起的奇异性。

**②参数化表示和隐式表示的优缺点**

只说明一点，用参数化表达方式来判断一个点是否在面或者线上，是不方便的。

**1.2 幂基曲线**

表示为



由幂函数的性质，可以得到



利用Horner方法进行幂基曲线的求解，写出一个幂基曲线的展开表达式有



先提出一个*u*能够得到



继续提出*u*得到

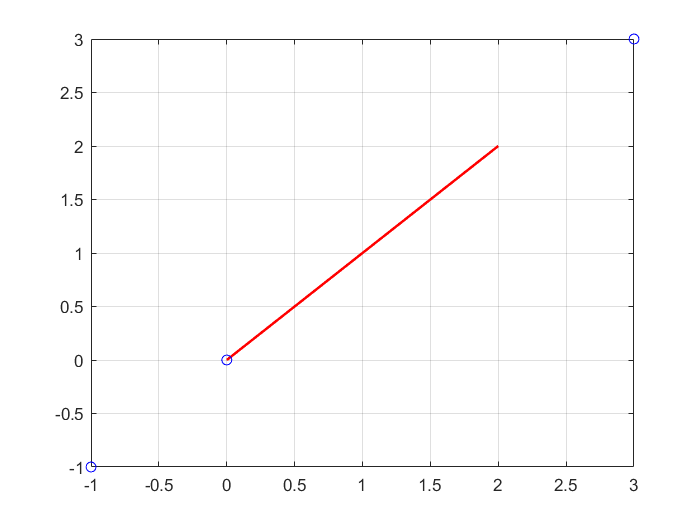


比如说对一个四次幂基曲线，其可以写为



那么容易看出，使用这种算法，对一个*n*次的幂基曲线，**其一共含有*n*+1个控制点，一共需要计算*n*次乘法和*n*次加法。**

幂基曲线的导数（任意阶数）相对来说是容易求取的。下图是想说明P6中的一个案例的问题，即可能在质点沿幂基曲线的运动过程中，**其没有速度为0的时候。**



**算法. 根据上述过程，Horner方法可以用伪代码写为**

Point CurveByHorner( b, n, u )

{

Point C=b[n];

for(int i=n-1;i>=0;i--)

{

C=C\*u+b[i];

}

return C;

}

**1.3 贝塞尔曲线**

*n*次的Bezier曲线表达式为



表示的是 ***i* 位置（对应于第 *i* 个控制点）*n*阶（应该是次，阶数 = 次数 + 1）**的**贝尔斯坦基（Bernstein）**函数，其表达式为



对贝尔斯坦函数求导，得到



**需要分开讨论的是**



可以看到，对于对应于**第一个控制点（0）的贝尔斯坦函数**，其是单调递减的，且在*u*为0的时候，其值为1，**而对于对应于最后一个控制点（*n*）的贝尔斯坦函数**，其是单调递增的，且在*u*为1的时候，其值为1。

而对于一般情况，其导数符号取决于



可以求出



可以看到，**非首末位置的贝尔斯坦函数**的导数符号取决于一个**线性函数**，导致这些位置的贝尔斯坦函数是**先增后减**的，且**越往后的贝尔斯坦函数，达到最大值的位置越晚。**

**1.3.1 Berstein函数的求导规则**

首先可以明确：



那么如果给出



那么就有



可以直接写出



进一步化简得到



对于第一个贝尔斯坦基函数的导数，其表示为



但之前也说过，第一个和最后一个贝尔斯坦基函数的导数要单独讨论，有



所以令



对于最后一个贝尔斯坦基函数的导数，其表示为



以之前写出过的表达式，写为



所以令



**1.3.2 Berstein函数的递推关系**

可以写出



先把前面的组合数拆开有



最终得到



可以看到，上面拆成的组合数，刚好对应于



那么把原来的贝尔斯坦基函数分开写来有



最终得到结论



**记忆方法：递推和求导公式都是一项两个负号，且分解开的贝尔斯坦基函数的阶数都降低1。求导才会得到系数为次幂的表达，故多的系数为*n*相关的是递推公式。**

另外，由贝尔斯坦基函数的定义，容易看出



**算法1. 求取*i*位置*n*阶贝尔斯坦基函数，由于对于贝尔斯坦基函数，有**



所以无论是求哪一个贝尔斯坦基函数，都可以通过若干递推来获取，这种问题，最好的方式是通过特殊值来书写通用的算法，比如说现在想要求



又因为



同时又因为



同时又因为



看出几个规律：

**①*n*为次数，有多少次，就需要进行多少次递归降阶,反之，如果提前知道了所有的0阶贝尔斯坦基函数，就需要*n*次的递推升阶来获得最终需要的贝尔斯坦基函数；**

**②每一次降阶，低一阶的需要的贝尔斯坦基函数的个数就要增加一个，拿这个问题来说，一共降阶4次，那么最终需要5个0阶的贝尔斯坦基函数来求目标4阶的贝尔斯坦基函数；**

**③每一次降阶，对应的贝尔斯坦基函数的最小的*i*都要降低1，比如这个问题中，最初的*i*为3，那经过4次递归，最终的0阶贝尔斯坦基函数的最小的*i*为-1，通过这个我们知道，在撰写算法的时候，数组中的第二个贝尔斯坦基函数的值为1，其他的贝尔斯坦基函数都为0，那么写成通用的形式也就是在第*n*-*i*+1的位置，则数组的索引如果是从0开始，就是第*n*-*i*个位置。**

基于上述的分析，可以以下述算法来进行贝尔斯坦基函数的求解

BersteinFunction(int n,int i,float u)

{

float\* tempV=new float[n+1];

for(int i=0;i<n+1;i++) *n+*1个0阶贝尔斯坦基函数

tempV[i]=0f;

tempV[n-i]=1f;

1du=1f-u;

for(int i=0;i<n;i++) *n*次递推，只用来计算次数，跟下标无关

{

for(int j=0;j<n-i;j++)

{

tempV[j]=u\*tempV[j]+1du\*tempV[j+1];

}

}

}

return tempV[0];

}

**算法2. 求取*n*阶的所有贝尔斯坦基函数，此时与算法1不同，要充分利用除了**为1，其他0阶贝尔斯坦基函数都为0的特点，那么从0阶开始递推，结合相邻的两个0，也就是和，一共3个0阶基函数，可以推出两个1阶基函数，并结合得到的1阶基函数的相邻的两个0，也就是和，一共4个0阶基函数，可以推出3个2阶基函数……

综上所述，0实际上并不需要存储，同样，对于最终目标是*n*阶的贝尔斯坦基函数，需要**做*n*次递推**，而最终也能够得到***n*+1个贝尔斯塔基函数**，且在**第*i*次递推升阶的时候，需要推导出*i*+1个贝尔斯坦基函数。**

基于上述的分析，可以以下述算法来进行贝尔斯坦基函数的求解，一个比较关键的问题是，如果利用算法1的思路，1个*n*阶的贝尔斯坦基函数需要*n*+1个0阶基函数，那么*n*阶的贝尔斯坦基函数共*n*+1个，那么总共就需要2*n*+1个0阶贝尔斯坦基函数，这样做的好处是比较方便，可以在整个处理过程中，都将首尾元素进行一般处理，而不是单独拿出来讨论，但是这个方法的问题在于其比较占用存储空间。

而算法2的思路在于，**充分利用在递推的过程中，非首尾贝尔斯坦基函数会被复用的特点（对于同*i*的高阶贝尔斯坦基函数，其乘上一个1-*u*，对于*i*大1的高阶贝尔斯坦基函数，其乘上一个*u*）**

float\* AllBersteinFunciton(int n. float u)

{

float tempV = new float[n+1]; *n*阶的贝尔斯坦基函数一共*n*+1个

tempV[0]=1f;

for(int i=0;i<n;i++)

{

float saved=0f;

for(int j=0;j<i+1;j++)

{

temp=tempV[j];

tempV[j]=saved+temp\*(1-u);

saved=temp\*u;

}

tempV[i+1]=saved;

}

}

**算法3. 计算Bezier曲线上的点**

利用算法2计算得到的结果即可**（下面是伪代码）**

PointOnBezierCurve( n,u )

{

float \*B=AllBersteinFunction( n,u );

Point C;

for(int i=0;i<=n;i++) *n*阶Bezier曲线一共需要*n*+1个控制点和基函数

{

C=C+B[i]\*p[i];

}

}

**1.3.3 Beizer曲线的求导**

基于**1.3.2中**得到的贝尔斯坦基函数的求导公式，得到Bezier曲线的一阶导数是



拆开来得到



舍去为0的基函数得到



同理可以得到



将其写为一般形式有



然后来探索一下贝塞尔曲线的首尾性质，容易得到



同理也有



同理可以写出端点处**任意阶导数**为



**可以看到在端点处的*l*阶导矢只和端点处的*l* + 1个控制点有关。**

**1.3.4 德卡斯特里奥（deCasteljau）算法**

简单来说，高阶的贝塞尔曲线可以通过控制点之间**不断构造一阶贝塞尔曲线得到**，*n*阶的贝塞尔曲线，要求得上面的某一点，需要**构造*n*次**一阶插值函数（每次构造都是在上一次构造的一阶插值的基础上），而每一次构造一阶插值的时候，又需要构造*n*-*i*次一阶插值，*i*表示的是已经构造了的次数（或者当前实现的插值对应的Bezier曲线的阶数）。

deCasteljau( P,u,n,C ) 输出一个点C

{

Q=P; 这一步是因为我们不要去破坏原来的控制点

for(int i=0;i<n;i++)

{

for(int j=0;j<n-i;j++)

{

Q [j]= Q [j+1]\*u+ Q [j]\*(1-u);

}

}

C=Q[0];

}

**1.3.5 实际的曲线求导方法**

因为*n*次的Bezier曲线求导后是*n* – 1次的Bezier曲线，所以不妨先转换控制点，然后通过deCasteljau算法推导矢量。而不同次的控制点的推导也可以采用递推的关系，比如有原本的控制点



求出来之后再用deCasteljau算法求解。

**1.4 有理Bezier曲线**

Bezier曲线和幂基曲线很明显都是多项式曲线，然后多项式曲线并不能精确表示圆、椭圆、双曲线等。故需要利用有理曲线。

**曲线有理**，即用有理函数（多项式相除）来作为基函数。而包括原在内的所有二次曲线，都可以由有理函数来表示。

基于上述因素，可以写出*n*次有理Bezier曲线的定义为



也可以表示为



式中



式中*B*表示的是**贝尔斯坦基函数**。

**1.4.1 比较重要的几个性质**

**（1）规范性**



**（2）端点插值性，即**



这个结论是容易看出来的。

**（3）端点导数性质，对有理 Berstein基函数求导**



太麻烦了，懒得求了，总而言之，其第一个点处的曲线切矢方向为



而在末端点处的切矢方向为



**此外**，曲线在*u*为0或1处的*l*阶导矢只与和端点相邻的*l*+1个控制点有关。

**例：现在让求一个在第一个象限内的1/4圆对应的有理Bezier曲线表达**

已经给出的已知条件是圆的半径为1，,那么由有理Bezier曲线的端点性质（过首末端点以及端点切矢的性质）可以得到二次有理Bezier曲线的3个控制点为



由于已知**1/4圆**的参数表达为



那么可以写出



展开来得到



由于最终的目标是要求上式对*u*的取值范围内的所有值都成立，所以得到



由于是对整个区间上都成立，所以直接取3个点上的特殊值也可以，这是**多项式之间的一一对应关系（对一个二次多项式，3点就可以将其完全确定）。**

由上述过程就可以确定，构造出来的曲线一定是圆，这样来说明：

令圆的参数表达式为



再令构造的有理Bezier曲线的表达式为



由于构造的是二次曲线，圆也为二次有理曲线，故只需要为*x*方向和*y*方向各找到3个对等关系，就能够证明两者表示的曲线相同，而又由于首末端点重合，这里直接提供了2个对等关系，最后一个通过导数为0来提供，有



以及



此时就说明了构造的曲线一定是圆。

**1.4.2 齐次坐标及其内涵**

对于各个坐标分量的表达式**都含有相同分母**的有理曲线，可以将欧几里得空间中的坐标表示到四维空间中有



并将四维空间中的点向三维欧氏空间中进行映射，将这个以原点为投影中心的透视投影定义为



式中，direction说明此时的坐标表示的是一个方向，而不是具体的点。

另外，齐次坐标的形式不唯一，只要在物理空间中的映射是一样的就可以。

**1.4.3 齐次坐标表示Bezier曲线**

那么有理Bezier曲线实际上就是基于齐次坐标（四维）形式的Bezier曲线，得到目标点后，再将其投影回三维空间，则在齐次坐标下的Bezier曲线表示为



式中



**意思是说，**先将给出的控制点三维坐标，通过一个权系数**映射到四维空间中（不是直接当做齐次坐标的前三个分量）**成为齐次坐标，再进行Bezier曲线的构造。那么这种情况下，最终点的第四个分量表示为



**并且有**



将其映射回三维物理空间得到目标的有理Bezier曲线的表达式为



写成矢量形式有



**那么总的来说，构造有理Bezier曲线的过程就可以通过齐次坐标的形式进行：**

**STEP 1：**拿到给出的控制点，将其转换为齐次坐标；

**STEP 2：**利用德卡斯特里奥算法，构造齐次坐标下的Bezier曲线；

**STEP 3：**将得到的高维空间中的点投影回物理空间。

**例：就以1.4.1节中的圆为例，现在已经得到控制点和权系数为**



**将其转换为齐次坐标**有



**比如此时**想要求*u*为0.5时候的有理Bezier曲线对应点，那么可以**在齐次坐标下利用decasteljau算法构造参数为0.5的Bezier曲线坐标**为



**将其投影回物理空间有**



还可以注意的一个问题是，参数为0.5的时候，此时圆点并不在圆弧对称轴位置，说明这种参数化方式**并非均匀参数化。**

**1.5 张量积曲面**

如果说参数曲线是直线段向二维或者三位空间中的变形或映射，那么参数曲面就是二维平面区域向三维空间中的映射。

张量积是一种表示参数曲面的形式，其通过**在两个方向上采用曲线的处理方法来构造曲面**，表示为



**1.5.1 幂基曲面**

以幂基曲面为例，其表示为



如果将*u*固定得到



此时得到的是幂基曲面上的一条幂基曲线，其中



并称其为沿着*v*方向的等参曲线（或*v*曲线），也就是**谁变那么就是沿着那个方向的等参曲线。**

对于幂基曲面，可以通过两个层次的Horner算法来计算其上的点，用矩阵的形式比较好理解这个问题，原幂基曲面可以写为



有两种顺序进行计算，用书上的计算顺序来举例有



式中，对***Temp***向量中的每一个元素，都要使用一次Horner算法来求值**（共*n*次，因为是*n*行）**，且每个位置上传入的控制点是对应一整行的控制点，然后在进行***u***和***Temp***之间的点乘运算的时候，还需要执行一次Horner算法。

**算法.** 利用上述逻辑，求解幂基曲面的算法可以写为

SurfByHorner( a, n, m, u, v )

{

b;

for(int i=0;i<=**n**;i++)

{

b[i]=CurveByHorner( a[i][],**m**,v );

}

CurveByHorner( b,**n**,u );

}

对于幂基曲面，其偏导数的求法为



用得到的偏导矢做叉乘就能得到某个点上的法向矢量。

**1.5.2 非有理Bezier曲面**

即容易通过将多项式Bezier曲线的表达进行推广，得到



其特性需要注意一点，其没有变差减少性，也就是说，与曲面相交的次数是有可能超过与控制多边形相交的次数的。

对于曲面***S***上某一点的计算，同样可以采用两个层次的Bezier曲线的构造方法，比如首先按照*i*方向进行Bezier曲线的构造，也就是



令



那么第二个层次的Bezier曲线的构造可以表示为



是这样的，可以先把控制点的点阵列给写出来，表示为



那么按照上述的过程，此时在外层循环是*j*的情况下，确定一个*j*，选取一整列的控制点，然后可以将这些点作为**控制点（而非型值点）**生成*m*+1条Bezier曲线，当然，需要具体取到某一个点的时候，给定一个值*u*，此时这*m*+1条Bezier曲线上，都会确定到一个点***Q***，那么将这*m*+1个点作为**控制点（而非型值点）**可以在第二个层次上生成一条贝塞尔曲线，那么此时再给定一个值*v*，就可以确定到Bezier曲面上的点。

综上所述，可以看出，在第一个层次求Bezier曲线的时候，仅有**上下两边求出的型值点**和**给出的控制点**重合，而在第二个层次求Bezier曲线的时候，又仅有**左右两边求出的型值点**和**求出的控制点（上一层次）**重合，故最终仅有四个角点上的型值点和控制点重合。

然后来讨论一下，执行线性插值的次数，先说上面这种顺序的情况下：

**第一个层次**，需要执行*m*+1次deCasteljau算法，而每一次执行deCasteljau算法，都是在构造一个*n*阶的Bezier曲线，那么对于一个*n*阶的Bezier曲线的构造，需要从线性插值开始，递推*n*次，且在第*i*次的递推过程中，*n*+1-*i*次线性插值，那么可以计算在第一个层次中需要执行的线性插值次数为



**第二个层次**，执行一次*m*阶的deCasteljau算法即可，即需要执行的线性插值次数为



则最终得到，**先按列取出控制点**进行构造的方法**（构造横向的*v*线，再给定*v*值）**，共需要执行线性插值的次数为



简而言之，对一个*n*次的Bezier曲线，deCasteljau算法需要执行次线性插值来求出目标点，比如对2次Bezier曲线，需要执行3次线性插值。

同理得到，**先按行取出控制点**进行构造的方法**（构造纵向的*u*线，再给定*u*值）**，共需要执行线性插值的次数为



两者相减有



如果*n*比较大，那么**先取列（先构造纵向*u*线，最后得到一条*v*线）的算法**就会计算**更多次**的线性插值，如果*m*比较大，那么先取行的算法就会计算更多次的线性插值。**即*n*大的情况，计算*u*线（先横向取控制点构造若干横向的*v*线，再构造纵向的*u*线，最后给定*u*值）比较好；*m*大的情况，计算*v*线比较好（先纵向取控制点，再构造横向的*v*线，最后给定*v*值）。**

**注：一定要分清楚*u*和*i*，*v*和*j*对应的方向。从点矩阵来看，*u*线是沿着控制点纵向的，*v*线是沿着控制点横向的**

**算法. 按照先按列取出控制点的方法书写伪代码**

deCasteljau( P,u,n,C ) ->看一下变量

URationalBezierSurf( P,u,v,n,m,S )

{

Point C[m+1];

if(n>=m)

{

for(int i=0;i<=n;i++)

deCasteljau( P[i][],v,m,C[i] );

deCasteljau( C,u,n,S );

}

else

{

for(int j=0;j<=m;j++)

deCasteljau( P[][j],u,n,C[j] );

deCasteljau( C,v,m,S );

}

}

最后还需要说明一个问题，再看到Bezier曲面的表达式



先取列，就是说，把*j*放置不变，那么当然，内层就是*i*变，此时对应的就是第一层次***u*变*v*不变**，第二层次就是*u*已经确定，然后*v*变。

**1.5.3 有理Bezier曲面**

其实同样是先在齐次坐标系中进行Bezier曲面的构造，表示为



式中



那么容易看出，将其投影回物理空间的方法为



式中



值得说明的是，**物理空间中**的有理Bezier曲面并非是张量积曲面，其面基函数不是由曲线基函数的乘积构成，但在**齐次坐标下**的Bezier曲面构造方式，构造的是一个张量积曲面。

最后还需要注意一点：

一定要搞清楚**控制点、基函数和参数**之间的对应关系，比如说



在控制点矩阵中，按列取控制点，此时*j*不变*i*变，那么对应的基函数下标为*i*，而这个基函数对应的参数又是*u*，则这种情况下取出来的曲线肯定是**沿着*u*方向变化而*v*令其固定不变的曲线。**

**2 B样条基函数**

先说明一个问题，这个是**理解参数连续和几何连续**的关键：就拿圆的参数表达式来说，其有两种，可以表示为

或者

那么对两种不同的参数表示方法都求导数，而此时得到的是在曲线上某一点处的切向矢量（瞬时的变化矢量），而**如果两者表示的是同一点**，那么肯定，此时**得到的切矢的指向一定是相同的**，但是**切矢的模是有可能不同的**，而这种不同，并非几何上的不同，而是由于参数化处理才带来的不同。

那从我的理解来看，达到几何连续就已经完全足够了。

**2.1 B样条基函数的定义和性质**

先说一下Bezier曲线的缺陷，只说比较重要的：

首先是**单段Bezier曲线**不适合用于形状交互设计，一个是因为如果描述的控制点比较多，那Bezier曲线的次数会非常高，还有就是对其控制不具有局部性，因为其基函数在整个参数区间上都起作用，所以修改一个控制点，整个曲线段都会受到影响；

**然后**是**分段Bezier曲线**，最主要的问题就是在于**为了保证连接点处的连续性，其对控制点本身的位置是有要求的**，这就比较不好了，最好的是找到一个方式，使得曲线的连续性仅仅和基函数有关，而不是控制点。

**2.1.1 B样条基函数的定义**

采用**递推方法**来定义B样条基函数，令是一个单调不减的实数序列，即，其中*u*称为节点，*U*成为节点矢量，用表示第*i*个*p*阶B样条基函数，其被定义为（Cox-deBoor公式）



为了**可以熟悉B样条基函数的表达，**先推导一下*p*为2的情况下的表达式，由于整个过程是递推上去的，先写出此时的节点矢量为



那么可以知道需要用到的**0阶B样条基函数为**





利用三个0阶的B样条基函数，可以计算两个1阶的B样条基函数为





利用两个1阶的B样条基函数，可以计算需要的2阶B样条基函数为

**总结几个规律和性质：**

**（1）定义第*i*个区间****上的*p*阶的B样条基函数需要从第*i*个区间数起的连续*p*+1个区间共*p*+2个节点信息（只需要横坐标，不需要函数值）；**

**（2）定义（1）中的*p*+1个区间****为B样条基函数的支撑区间（组成这个区间的节点定义了对应的基函数，反过来该基函数也只能影响这个区间内的B样条曲线，在这个区间之外的其他取值，B样条基函数的取值为0）；**

**（3）基函数的个数取决于给定的控制点的个数，比如给出的控制点索引为0, 1, 2,..., *n*（那么控制点的个数为*n*+1，自然需要），那么可以确定同样索引的基函数，而确定的这*n*+1个基函数的支撑区间所含节点的并集*U*容易确定为（最后一个基函数（索引为*n*）的支撑区间为）**



***U*被称为这一组B样条基函数的节点矢量。**

**（4）在任意给定的区间上，最多只有*p*+1个B样条基函数的值不为0，这个很好理解，因为所有的B样条基函数都是由0阶基函数**



**递推得到的，那么想要在区间不为0，那么这个基函数必须要基于**递推得到，而**能够使用到这个基函数的*p*阶基函数**，是从往前面数到一共*p*+1个*p*阶基函数。

**（5）在任意给定的区间上，有**，这个结论要对应于**（4）**中的结论，即把在一个区间不为0的*p*+1个*p*阶的B样条基函数加起来，其值为1，证明如下：



然后将两个式子分开来处理，**先看第一个式子**，由于此时降阶了，而*p*-1阶的B样条基函数最多到不为0，那么第一式的第一项就为0，将其舍去，那么第一式就可以写为



**再来看第二式**，通过令



那么原式可以被化为



而上式中，明显有最后一项为0，故第二项最终可以被化简为



**将前式和后式加起来得到**



通过这个规律继续往下写有



**得证，不得不说，真的很有意思。**

**（6）除了*p*为0的时候，其他阶数的B样条基函数严格地达到最大值一次。**

**2.1.2 重节点对基函数连续性的影响**

首先来几个定义，定义重复度为两种情况：

**（1）**节点相对于节点矢量的重复度，即节点在一个节点矢量中重复出现的次数，比如有



那么0和1的重复度都为3，0.5的重复度为1。

**（2）**节点相对于基函数的重复度，因为影响基函数的节点只有到共*p*+2个，故需要具体讨论一个节点在基函数的支撑区间内出现的次数，比如对于一个2次的B样条基函数，有



那么0对该B样条基函数的重复度记为3，设重复度的标志为*k*。

**现在重点来了，**对于B样条基函数，其是一个分段的多项式，而在每一个小段区间内，其是无限次连续的，但是在分段节点上，其连续性为



**其中*k*最小取为1**，如果对*k*为0的情况，说明该节点并不在基函数的支撑区间之内，所以该基函数在这个节点的值以及各阶导数肯定是0，其连续性不受该节点的影响。

再说明一下说明求了一次导数之后，得到的导函数依然连续，说明位置连续，求导得到的导函数不连续，其不可导，位置都不连续。

综上所述，基函数在某个节点处的连续性主要看这个节点在基函数支撑区间内**（支撑区间的确定是看下标不是看数值）**的重复度。另外，B样条基函数的次数越高，**在所有节点处的连续性就越高**；在某个节点处的重复度越高，**在这个节点处的连续性就越低**。

且重节点减少了非0区间的个数，如果一个基函数的支撑区间内，**没有任何重复度超过1的节点，那么非0的区间数为*p*+1**，**而如果有节点的重复度超过了1，那么将所有重复次数超过1的元素的超过次数加起来得到*m***，最终的非0区间数就为*p*+1-m。

**比如，**有一个5阶的B样条基函数，其支撑区间对应的节点为



那么1的重复度为2，超过1的部分为1次，2的重复度为3，超过1的部分为2次，那么*m*为1+2=3，则不为0的区间个数为

*p*+1-*m*=5+1-3=3

一个小技巧归纳：



发现了一个神奇的现象，**当分母相减为0的时候**，按理来说这种情况是有错误的，但是当哪一项低阶的B样条基函数前面的系数分母为0，**这个低阶B样条基函数本身肯定恒为0。**

**这个情况是这样的，对于**，其支撑区间为



**那如果**，说明在该基函数的支撑区间内，*p*+2个也就是*b*+1个节点全部重合，也就是*p*+2个重复度，那么在其*p*+1个节点区间（看下标）内，非0区间的个数为*p*+1-(p+2-1)=0个，这是**2.1.2**的结论。

**同理，**对，其支撑区间为



**但**，所以其同样在整个支撑区间上都为0。

**2.2 B样条基函数的导数**

**先给出结论**



**不证明了，太麻烦了。**

然后给出一个一般的求导公式为



反复对上式两侧求导可以获得*k*阶的B样条基函数的导数公式为



**还能得到一个通过*p*-*k*阶基函数来求解*p*阶基函数的*k*阶导数的公式为**



**式中**，表示对应于求*k*阶导数下，第*j*个位置的系数，表示为



**使用条件：**

**①*k*不应该超过*p*；**

**②节点分母相减为0的情况，令其得到的商为0，这个在理论上是没问题的，可以见上一节的分析。**

**我估计，对B样条基函数，其区间内部，用上面的公式是没问题的，然后对节点，只要是在连续性条件有保障的情况下（大概没有连续性保障的情况下，求出来的结果是阶跃的导函数），应该也是可以用的。**

**另外，式中的*a*可以用一个三角矩阵来表示**



**还有其他公式就不研究了，不一定很重要（知道*p*-1阶基函数的*k*阶导数，求*p*阶基函数的*k*阶导数）。**

**算法1. 计算非0B样条基函数及其导数**

为了搞明白，它到底是怎么进行的，我们来推导一些实例。

**例：**设*p*为2，节点矢量为



此时要求***u*为2.5的时候**，计算和。

写出通式表达为



依照文中所述，现在先构造一个表，将B样条基函数和节点矢量的差给放进去有



注意，此时*j*表示当前递推进行的次数，递推*j*次说明当前构造的是*j*阶的B样条曲线，*r*表示在一次递推进行过程中，需要进行**遍历**求解的B样条基函数，而且特别注意，对于right和left的存放，只需要存放对应基函数不为0的右式即可。

另外，对right+left来说，行表示的是递推的次数，也就是构造B样条函数的阶数，列表示的是一次递推中当前遍历的索引次数；而对B样条基函数来说正好反过来，每一列表示的是递推的次数。

然后根据公式来看



**式中**，表示对应于求*k*阶导数下，第*j*个位置的系数，表示为



其实，现在比较重要的问题在于，找到上述表达式中，节点矢量的分母对应于创建的矩阵中的位置。

想要求，那么来看原式，需要的元素是



而因为为0，故只需要计算，同理对于



而此时两个1阶B样条基函数都不为0，所以要同时计算和。同理对于



再来看二阶导



以及



以及



这其实说明了一个问题，实际上如果要计算在某一个区间上，不为0的所有B样条基函数的导数，那么对应的*a*是都需要求解的，具体来说是



可以看到，对应的阶数越多，需要求解的*a*也就越多。

**我放弃了，太特么的蛋疼了。**

**2.3 B样条基函数进一步的性质**

**非周期节点矢量具有以下形式**



式中，*a*和*b*的重复度都为*p*+1。而最后一个*b*的下标为*n*+*p*+1，如果按书上的规则设节点个数为*m*+1=*n*+*p*+2，那么可以得到

*m*=*n*+*p*+1->*n*=*m*-*p*-1

式中，*n*+1则表示了控制点也是基函数的个数。

而从写出来的节点矢量，可以明显看出



那么现在来看



由于明显的，并且，所以进一步写为



接着往下写有



总结这个规律，可以写出



**而此时就有**



**同理可以写出**



那么再**由B样条基函数的规范性**，可以得到



以及



另外，以下述节点矢量定义的B样条基函数其实就是贝尔斯坦基函数



**上述的节点矢量，0和1的重复度都是*p*+1，则一共可以定义*p*+1个贝尔斯坦基函数，也就是说此时的控制点也是*p*+1个，那么此时定义的曲线是*p*阶的Bezier曲线。**

**2.4 B样条基函数的计算**

**算法1. 用于查找参数*u*处于节点矢量的哪一个区间之内，当然，此时研究的节点矢量依然是**



int FindSpan( n,p,u,U )

{

if( abs(u-U[n+1]<1e-8) ) return n;

**这一步是因为B样条基函数的定义中，在所有分段区间之上，都是左闭右开的，而其实在节点矢量的最后一个点（在这种情况下，是下标为*n*+1的参数节点），其值实际上是需要取到的，而也正是由于判断参数在哪一个区间上的条件是左闭右闭开的，所以必须要考虑参数点位于下标为*n*+1的节点的情况，并将其作为第*n*个区间的参数。**

int low=p, high=n+1;

int mid=(low+high)/2;

while( u<U[mid]||u>=U[mid+1] ) **最终目的是找到U[mid]<=u<U[mid+1]**

{

if( U[mid]<u ) low=mid;

else high=mid;

mid=(low+high)/2;

}

return mid;

}

**算法2. 计算**。这个问题是这样的，考虑一个区间，那么在这个区间上不为0的基函数就是上述的那些，然后，实际上这些非0的高阶基函数都是基于唯一0阶基函数



推导得到的。

**例：**设*p*为2，节点矢量为



此时要求***u*为2.5的时候，不为0的B样条基函数**，首先，先按照算法1确定该参数点所在的区间范围，现在直接看出来是在



并且还可以知道，只有基函数



是不为0的，且是基于推导出来的以上的2阶B样条基函数，且推导关系可以表示为



具体写出来有



把值带入进去，首先求出1阶的B样条基函数的值为



然后求出2阶的B样条基函数为



为了设计一个更好的算法，我们来找找规律，首先看对于1阶的基函数的情况



**上述表达中，**有两个可以在意的地方：

①第一式的左项和最后一式的右项为0；

②前一式的右项和后一式的左项有很大一部分重合的地方，不必重复运算。

基于上述的问题描述，继续写出2阶的B样条基函数的推导



**可以看到**，此时的表达仍然有前述1阶表达式的规律。并且通式还可以表达为



**式中**，*r*从0取到*j***（或者说从*p*取到0吧，都一样）**。那么如果令



那么之前的通式可以表达为

**式中，*j*表示外层循环，也就是当前进行递推的次数，其索引从1开始，最后到*p*，*r*表示内层循环，也就是要遍历*i*-*j*到*i*的整个过程。**

写出1阶的表达式为（*j*=1，*r*=0和1）



同理写出2阶的表达式为（*j*=2，*r*=0、1和2）



可以看到，由于第一项的左项和最后一项的右项本身为0，所以不需要计算，那么在一个大的阶与阶之间的递推过程中，**right和left的下标不会超过当前的递推次数。通过之前的通式，也可以说明这个问题了，因为在通式中，各个通式在循环过程中的最大值为**



**可以看到，在第一个算式的左式和最后一个算式的右式，都超出了当前外层索引，即为*j*+1，但是由于这两个式子本身为0，不用计入，所以保证了所有索引都在可行范围内。**

**则算法可以用伪代码写为**

BasisFunc( i, u, p, U, N ) 有*i*说明已经找到了*u*所在的区间

{

N=new float[p+1];

N[0]=1f;

for( int j=1;j<=p;j++ ) 要递推*p*次

{

float saved=0f;

left[j]=u-U[i+1-j];

right[j]=U[i+j]-u;

for( int r=0;r<j;r++ ) 比如第一次递推，要算1次，那自然要加个1

{

temp=N[r]/( right[r+1]+left[j-r] );

N[r]=saved+temp\*right[r+1];

saved=temp\*left[j-r];

}

N[j]=saved;

}

}

**求导数的算法先不管了。**

**算法3. 现在目的是要求出某一个B样条基函数的值（可以通过这个函数来生成B样条基函数的曲线），先写出问题的通式**

**第一次降阶**



**第二次降阶**



……**第*k*次降阶**



我勾巴服了，还是用案例来说明吧，设*p*为2，节点矢量为



此时要求***u*为2.5的时候，不为0的B样条基函数**，但现在只求其中一个，我们将三个不为0的B样条基函数的求解递推关系都写出来



写出表达式有



以及



写出表达式有



以及



写出表达式有



总结来说，设计这种比较复杂的递推关系的算法，必要重要的几点是：

**（1）**弄清循环变量对于表达式的影响，**是正向的，还是逆向的（特别注意，不然凑出来肯定是错的）**，需不需要+-之类；

**（2）**初始情况和结束情况的处理；

**（3）**任意情况的处理；

**（4）**其他特殊情况。

现在就来具体分析：

OneBasisFunc( p, m, U, i, u, n ) **注：i是基函数的下标，不是搜索区间下标**

{

float Nip;

if( (i==0&&u==U[0])||(i==n&&u==U[n+p+1]) ) return 1.0f;

**特殊情况处理1（4），**主要是为了处理最后一个区间为右开的情况

if(u<U[i]||u>=U[i+p+1]) return 0.0f;

**特殊情况处理2（4），**处理的目的不用多说了，但必须要注意，这个特殊处理必须放在**特殊情况处理1（4）**之后，否则如果此时如果对应的i为n，且u取到U[n+p+1]，此时会返回0.0f，那就错了。

现在已经确定了，所求的u在对应基函数的支撑区间内，且特殊情况也被排除，则只需要按照区间左闭右开的形式进行计算即可。

float \*N=new float[p+1];

for( int j=0; j<p+1;j++ )

{

if( u>=U[i+j]&&u<U[i+j+1] ) N[j]=1.0f;

else N[j]=0.0f;

}

有一个很有意思的事实在于，无论求的基函数是多少阶的，**最终能够递推得到它的0阶基函数中，只有一个基函数的值不为0。**

for(int k=1;j<=p;k++) 要进行p次递推

{

if( U[0]==0.0f ) saved=0.0f;

else saved=(u-U[i])\*N[0]/(U[i+k]-U[i]);

for(int j=0;j<p-k+1;j++)

{

if( N[j+1]==0.0f )

{

N[j]=saved;

saved=0.0f;

}

else

{

left=U[i+j+k+1];

right=U[i+j+1];

temp= N[j+1]/(left-right);

N[j]=saved+(left-u)\* temp;

saved=(u-right) \* temp;

}

}

}

return N[0];

}

简而言之，在上述的算法中，saved用于存储每次递推过程中的左项（整个一项），其实很关键的一个技巧就是找准下标的影响因素，然后对应着凑就完事了。

**3 B样条曲线曲面**

**3.1 B样条曲线的定义和性质**

*p*次B样条函数的定义为



对于固定的*u*值，计算B样条曲线上的对应点需要三步：

**（1）**找到*u*所在的节点区间；

**（2）**求取对应节点区间内值不为0的B样条基函数；

**（3）**计算非0基函数与控制点的乘积并求和。

**算法. 求取B样条曲线上的点**

这就比较简单了

CurvePoint( n, p, U, P, u )

{

Point C;

int span=FindSpan( n, p, u, U );

BasisFuncs( span, u , p, U, N );

C=0;

for( int i=0;i<=p;i++ )

C=C+N[i]\*P[span-p+i];

因为找出来的区间对应下标为span，那么从span-p开始到span基函数都不为0，所以要使用到对应的控制点P。

}

**B样条曲线的一些性质：**

**（1）**表示求了*r*次导数之后，导函数连续；

**（2）** B样条曲线具有强凸包性，也就是说B样条曲线中在一段参数区间上的部分必须会在对应控制多边形的凸包中。而且必须要注意的是，曲线和控制点的对应关系，比如说在参数区间上的曲线段，其对应的不为0的基函数为



那么对应于这个参数区间曲线的控制点也就是（每个参数区间段曲线对应*p*+1个控制点）



而且很牛皮的一点在于，这个真的太契合于节点矢量的设计。



对上述的节点矢量，第一个有效曲线段（真正会画的出来的）对应的参数区间为，那么此时刚好（不多不少）用上了之前所有的控制点，也就是



**（3）对于节点不连续的B样条基函数，**通过调整控制点的位置，是有可能使得到的B样条曲线具有一定的连续性的。

**3.2 B样条曲线的导矢**

之前导矢的算法咱们就不管了，然后主要是知道，得到了各阶导数的值，放在一个矩阵里



式中，*n*表示求导的最高阶数（不超过*p*），*p*表示基函数的阶数，*p*+1表示在某个区间上仅有*p*+1个B样条基函数不为0。

先利用以下的公式来实现一个例子：



**例：**设*p*为2，节点矢量为



此时要求***u*为2.5的时候**，计算、和。那么分别写出





实际上关系还是很明确的，上面的表是递推关系中需要用到的所有数据，同时也是求导关系中所有用到的数据**（是求一阶导的情况）**。

而根据B样条曲线的定义式，容易写出任意阶数的B样条导矢的表达式为



利用上式写出算法为

CurveDerivsAlg( n, p, U, P, u, d, CK )

{

用CK来返回B样条曲线的所有阶数的导矢

int du=min(n, p); 因为最大就只能求到p，更大就为0了

for( int k=p+1;k<=d;k++ ) CK[k]=0.0; 最大阶数为d，但后面的导矢都为0，所以先初始化好。

int span=FindSpan( n, p, u, U );

DersBasisFuncs( span, u, p du, U, nders ); 得到任意阶B样基函数导数

for( int k=0;k<=du;k++ )

{

CK[k]=0.0;

for(int j=0;j<=p;j++)

{

CK[k]=CK[k]+nders[k][j]\*P[span-p+j];

}

}

}

再来看，先写出一般形式B样条曲线的表达式为



对上式求导得到



将上式分解为



对左式，令



那么上式可以化简为



而和必然为0，所以原式可以化简为



那么如果令



那么原式可以表示为



如果此时用*n*个控制点***Q***和*p*-1阶B样条基函数的要求构造一个节点矢量**（实际上就是在原来节点矢量的基础上，在前后部分各去掉一个重复的节点）**，可以得到



此时的节点矢量的节点数目为*n*-1+*p*-1+2=*n*+*p*个。而在原节点矢量上，从*i*为1开始取，这个时候不为0，而在新的构造的节点矢量上，其可以从0开始取，也就是说**在新构造的节点矢量上**，B样条曲线的导数可以表示为



**例：给出一条B样条曲线为**



其对应的节点矢量为



求其对应的一阶导矢。

首先构造出来我们要的节点矢量为



然后求出点



那么可以推出



总的来说，在求导之后，*p*阶的B样条曲线会被降阶为*p*-1阶B样条曲线，并且转换后的控制点个数也会减少1个。

另外，对B样条曲线的端点处，容易看出有



**注：需要注意的是在计算B样条曲线导矢的基函数的过程中，其使用的节点矢量是构造的新的节点矢量，但是对于B样条曲线导矢的控制点的求取的过程中，其是利用原来的节点矢量的节点进行控制点构造的。**

那么现在想要求B样条曲线的任意阶导矢就是比较简单的了：



现在有两个目的：

**（1）构造新的控制点；**

**（2）构造新的节点矢量。**

而在构造新的控制点的过程中，最好始终使用原来的节点矢量，将这个过程和第二个过程分开，那么有



那么观察可以得到通用的表达式为



然后构造节点矢量为



以上述的表达方式，中间的参数**直接用的原来节点矢量的节点和下标**，但个数是变了的，需要注意。

**算法. PK[k][i]用来返回*k*阶导曲线的第*i*个控制点，这里**，如果，那最终的结果是返回**求导后B样条曲线的所有控制点**。

CurveDerivCpts( n, p, U, P, d, r1, r2, PK )

{

r=r2-r1;

for(int i=0;i<=r;i++) 将原来的需要的控制点全部读进来

{

PK[0][i]=P[r1+i]; 先把0阶的控制点读进来

}

for(int k=1;k<=d;k++) 这个循环是代表当前求导的阶数

{

temp=p-k+1; 求解系数

for(int i=0;i<r-k;i++) 每次求导，产生的控制点的个数就减少1

{

PK[k][i]=temp\*(PK[k-1][i+1]-PK[k-1][i])/(U[r1+i+p+1]-U[r1+i+k]);

**注意，当前r1+i才是逻辑上的i，返回的控制点是从下标r1开始的。**

}

}

}

让我们来计算一下B样条曲线端点的二阶导矢



进一步化简得到



最终化简得到



然后是



进一步化简得到



最终化简得到



总结一个更易于理解的公式吧



**算法. 重新给出一个基于上边方法的求B样条曲线导矢的方法**

CurveDerivsAlg( n, p, U, P, d, u, d, CK )

{

du=min(d,p);

for(int k=p+1;k<d;k++) CK[k]=0.0f; 到下标为p，即p阶导时为常数

span=FindSpan( n, p, u, U );

AllBasisFuncs( span, u, p, U, N );

最后求出来是这样的



注意，上述的下标全部都是全部都是基于当前的节点矢量来说的，也就是先不要修改节点矢量，就完全按照当前节点矢量来求解低阶的B样条基函数，而且很巧妙的一点在于，在求解高阶B样条基函数的过程中，所有在区间*i*上不为0的低阶B样条基函数都会被计算出来，在之前的算法之上，只需要把向量改成矩阵进行存储即可。

CurveDerivCpts( n, p, U, P, du, span-p, span, PK );

for( int k=0;k<=du;k++ ) 表示一共能够进行的求导次数

{

CK[k]=0.0f;

for(int j=0;j<p-k;j++)

CK[k]=CK[k]+N[j][p-k]\*PK[k][j];

}

}

**3.3 B样条曲面的定义和性质**

**3.3.1 B样条曲面的定义**

B样条曲面由两个方向的控制点网格、两个节点矢量和单变量B样条基函数的乘积来定义，表示为



节点矢量为



给出参数，计算B样条曲面上的点一共需要5个步骤：

**STEP 1：先对参数*u***，查找其在节点矢量***U***中的区间位置；

**STEP 2：计算在对应区间上的非0基函数**；

**STEP 3：再对参数*v***，查找其在节点矢量***V***中的区间位置；

**STEP 4：计算在对应区间上的非0基函数**；

**STEP 5：利用B样条曲面的定义式求解点：**



或者写成矩阵的形式为



**算法. 用来计算B样条曲面上的点**

SurfacePoint( n, p, U, m, q, V, P, u, v, S )

{

uspan=FindSpan( n, p, u, U );

BasisFuncs( uspan, u, p, U, Nu );

vspan=FindSpan( m, q, v, V );

BasisFuncs( vspan, v, q, V, Nv );

uind=uspan-p;

for( int l=0;l<=q;l++ ) 先计算行和矩阵的值，一共得到的列数为后面列的行数，所以一共需要计算*q*+1次

{

temp[l]=0.0f;

vind=vspan-q+l; 确定当前计算的是哪一列

for(k=0;k<=p;k++)

temp[l]=temp[l]+Nu[k]\*P[uind+k][vind];

}

S=0.0f;

for( int l=0;l<=q;l++ )

S=S+Nv[l]\*temp[l];

}

**3.3.2 B样条曲面的性质**

只说明几个比较重要的性质：

**（1）局部性：**如果选定的参数(*u*, *v*)在矩形，那么



**超出了支撑区间的范围；**

**（2）另外一种局部性：**在参数矩形至多只有



共(*p+*1)(*q*+1)个基函数乘积非0；

**（3）单极值性：**如果*p>*0且*q*>0，那么精确地达到一次最

大值；

**（4）连续性：**因为是曲面，所以此时讨论的基本上是在某一点处的偏导矢，对基函数乘积来说，在区间内部，其是无限阶连续可导的（多项式乘积），但是在节点处，沿着*u*或*v*方向的偏导数的连续可微阶数为和；

**（5）强凸包性：**如果(*u*, *v*)在矩形内，此时B样条曲面在控制点组成的控制多边形的凸包内；

**（6）控制点：**可以通过调整控制点来使连续性不高的B样条基函数实现一定连续性的B样条曲面；同样也可以通过调整控制点来使得本来连续的曲面达到视觉上不连续的效果。

跟Bezier曲面一样，B样条曲线也存在等参数线**（需要注意，等参数线是在B样条曲面之上的，其形成的方法是将曲面求法分解为两次曲线求解的第二层次）**。

先来看沿*v*方向的等参数曲线，即先确定*u*，那么原式可以写为



简单思考就能够知道，等*u*参数得到的曲面上的曲线，是沿着控制点*i*不变，*j*改变的方向的。将上式化简一下有



式中



那么等*u*曲线，也就是*v*向曲线，是一条*q*阶的B样条曲线，其节点矢量为***V***；同理等*v*曲线，也就是*u*向曲线，是一条*p*阶的B样条曲线，其节点矢量为***U***。

**3.4 B样条曲面的偏导矢**

**3.4.1 基于B样条基函数导数的求法**

现在给出一种计算B样条曲面上所有直到*d*阶的偏导矢，即



**注意：上述说的所有是指**



那么用来存放导数的变量应当是下图所示的形状



上图表示的是*p*和*q*都大于*d*的情况，**此时在整个导数矩阵中的数据都需要计算，此外，还有其他几种情况**





**如上图所示，表示的情况分别为**



**红色的部分就是需要命为0**的部分，这个如何实现之后的算法中所有体现。那么用来存放导数信息的应当是一个矩阵



实际上，B样条曲面的偏导矢可以直接通过基函数的导数来得到，即有



**算法. 来实现上述的过程，实际上是对之前实现的算法的应用**

SurfaceDerivsAlg( n, p, U, m, q, V, P, u, v, d, SKL )

{

du=min( d, p );

for( int k=p+1;k<=d;k++ )

for( int l=0;l<=d-k;l++ ) SKL[k][l]=0.0f;

for( int l=q+1;l<=d;l++ )

for( int k=0;k<=d-l;k++ ) SKL[k][l]=0.0f;

uspan=FindSpan( n, p, u, U );

DersBasisFuncs( uspan, u, p, du, U, Nu );

vspan=FindSpan( m, q, v, V );

DersBasisFuncs( vspan, v, q, dv, V, Nv );

for( int k=0;k<=du;k++ )

{

for( int s=0;s<=q;s++ ) 最终得到的列数为V基函数的行数

{

temp[s]=0.0f;

for( int r=0;r<=p;r++ )

temp[s]=temp[s]+Nu[k][r]\*P[uspan-p+r][vspan-q+s];

}

dd=min( d-k, dv );

可以去观察上面画出的图，实际上需要进行填充的是重叠的部分**（红色的部分已经全部填充为0）**，而求出来的dv实际上是在第0行的标识的左边的那个点，而在按行循环的过程中，需要填充的最右边的那个点有可能是d-k。

for( int l=0;l<=dd;l++ )

{

SKL[k][l]=0.0f;

for( int s=0;s<=q;s++ )

SKL[k][l]=SKL[k][l]+temp[s]\*Nv[l][s];

}

}

}

**3.4.2 基于对B样条曲线求偏导的方法**

对***S***求偏导得到



进一步化简有



式中



而上式是一个B样条曲线的表达式，其表示的是固定*j*的情况下，拿出一列控制点，以*u*为参数的B样条曲线。

而对B样条曲线的导矢已经有过研究，将结论带入有



式中



类似地对于*v*有



式中



那么如果先对*u*求一阶偏导（这个过程已经得到了），然后再对*v*求一阶偏导，就容易得到



式中



新的节点矢量的定义与之前一致。

通用的形式有



式中



前式一个很好的应用在于计算角点的偏导矢，比如有



那么可以得到



再次写出B样条曲面的表达式



式中



如果此时令*u*为0，那么



那么原B样条曲面退化为



同理如果令*v*为0，那么



那么有



得到的结果可以与前面的式子相互验证。

**算法. 该算法用于计算曲面的所有直到**阶导对应的曲面控制点（r和s用来规定需要计算哪些控制点），该算法输出的结果为PKL[k][l][i][j]，表示对*u*求*k*次偏导对*v*求*l*次偏导的情况下，第(*i*, *j*)个位置上面的控制点。

SurfaceDerivCpts( n, p, U, m, q, V, P, d, r1, r2, s1, s2, PKL )

{

du=min(d, p); dv=min(d, q);

r=r2-r1; s=s2-s1; 与要求出来的控制点的个数有关系，表示的是涉及到要求的控制点的下标，且每求一次导数，控制点个数就减小1，**最后一个控制点的下标也减小1。**

for( int j=s1;j<=s2;j++ )

{

CurveDerivCpts( n, p, P[][j], du, r1, r2, temp );

固定*j*，也就是拿出一列控制点，去求出该列控制点沿着*i*方向变化的所有阶数的对应控制点，相当于求出了对*u*的所有阶数而对*v*为0阶的偏导数对应的控制点。

for( int k=0;k<=du;k++ )

**实际上，最多只能计算到du，对于d>p的情况，计算到p阶的时候，就只剩一个控制点了，不能再继续往下算，对于p>d的情况，再多算也不需要了，因为只要到d阶。**

{

for( int i=0;i<=r-k;i++ )

**最多只能到r-k，i代表了u方向上能够存在的控制点的个数，那在k为0，也就是0阶导数的情况下，自然是r+1个控制点无疑，但是每多求一阶导数，尾部就会减少一个控制点。**

PKL[k][0][i][j-s1]=temp[k][i];

}

}

for( int k=0;k<=du;k++ )

{

**注意：控制点的个数只与求导的阶数有关，0阶时整个控制点矩阵为(*r*+1)(*s*+1)的，当对*u*求导，每求一阶导，对应的控制点矩阵就减少一排，即对应于*k*阶导数的情况，此时的控制点矩阵为(*r*+1-*k*)(*s*+1)阶的，同理，每次对*v*求导，每求一阶导，对应的控制点矩阵就减少一列，即对应于*l*阶导数的情况，此时的控制点矩阵为(*r*+1-*k*)(*s*+1-*l*)阶的。**

**此外，还有一个问题在于，哪些控制点需要求，哪些控制点不需要求，这个其实和之前的求偏导矢的算法类似，见之前画出来的图，有些导矢是不需要求，恒为0的。**

for( int i=0;i<=r-k;i++ )

{

dd=min(d-k,dv);

CurveDerivCpts( m, q, V, PKL[k][0][i][], dd, s1, s2, temp );

for( int l=1;l<=dd;l++ )

for( int j=0;j<=s-l;j++ )

PKL[k][l][i][j]=temp[l][j];

}

}

}

**算法. 基于上述的控制点求法，现在给出求解B样条曲面偏导矢的求法，该算法可以给出B样条曲面上的任意阶的偏导矢**

SurfaceDerivsAlg( n, p, U, m, q, V, P, u, v, d, SKL )

{

du=min( d, p );

for( int k=p+1;k<=d;k++ )

for( int l=0;l<=d-k;l++ ) SKL[k][l]=0.0f;

for( int l=q+1;l<=d;l++ )

for( int k=0;k<=d-l;k++ ) SKL[k][l]=0.0f;

uspan=FindSpan( n, p, u, U );

AllBasisFuncs( uspan, u, p, U, Nu );

vspan=FindSpan( m, q, v, V );

AllBasisFuncs( vspan, v, q, V, Nv );

SurfaceDerivCpts( n, p, U, m, q, V, P, d, uspan-p, uspan, vspan-q, vspan, PKL );

for( int k=0;k<=du;k++ )

{

dd=min( dv,d-k );

for( int l=0;l<=dd;l++ )

{

SKL[k][l]=0.0f;

for( int j=0;j<=q-l;j++ )

对应的是*j*也就是*v*方向，有效的控制点为*q*-*l*列

{

temp=0.0f;

for( int i=0;i<=p-k;i++ ) **先计算行向量乘矩阵**

temp=temp+Nu[i][p-k]\*PKL[k][l][i][j];

SKL[k][l]=Skl[k][l]+temp[j]\*Nv[j][q-l];

}

}

}

}

**4 有理B样条曲线曲面**

**4.1 NURBS曲线的定义和性质**

**4.1.1 NURBS曲线的定义**

一条*p*次NURBS曲线定义为



*N*就是B样条基函数，其定义在节点矢量**（规定所有的权系数都大于0）**



令



那么原式可以表示为



式中，*R*成为有理基函数，它们是[0, 1]上的分段有理函数。

**4.1.2 NURBS曲线的性质**

只标注几个比较重要的性质：

**（1）**；

**（2）**在分段区间内部，有理基函数是无限次连续可微的，而在重复度为*k*的节点，有理基函数的是*p*-*k*连续可微的；

**（3）通用性：**当权系数全部为1，有理基函数**退化为B样条基函数**；若节点矢量无内部节点，即节点矢量由*p*+1个0和*p*+1个1组成，此时代表的是*p*阶的**有理Bezier曲线**，且控制点个数为*p*+1，在此基础上，如果所有的权系数都为1，那么此时曲线进一步退化为**非有理Bezier曲线**；

**（4）局部修改性：**主要注意，当去修改和的时候，受到影响的只有支撑区间部分的曲线部分。

**（5）权因子对曲线的影响：**对权因子，其越大，曲线越被拉向，其越小，曲线越被推离。而且如果*u*不变，仅修改，此时得到的是过的直线，这个结论是容易得到的，实际上此时的直线表达式为



则上式表示的就是一条直线（不是按线性变化，但是从图形上来看是一条直线），并且当趋于无穷，此时点就更加趋近于。

而NURBS曲线的计算同样可以利用齐次坐标，即首先将给出的控制点变为加权的齐次坐标为



那么在四维空间中定义B样条曲线为



然后将其投影回物理空间得到最初的NURBS曲线表达式。

**算法.** 用齐次空间中的坐标来求取NURBS曲线，且需要注意，在本书中，C=Cw/w用于表示将齐次坐标投影回三维物理空间。

CurvePoint( n, p, U, pw, u, C )

{

span=FindSpan( n, p, u, U );

BasisFuncs( span, u, p, U, N );

Cw=0.0f;

for( int j=0;j<=p;j++ )

Cw=Cw+N[j]\*Pw[span-p+j];

C=Cw/w;

}

**4.2 NURBS曲线的导矢**

现在令齐次坐标为



那么采用前面B样条曲线求导矢的方法，就可以求出



则在拿到控制点后，先把所有控制点构造为需要的齐次形式，然后代入到之前的算法中，就可以求出两个分量。

**4.2.1 一阶导矢的计算方法**

然后根据这个来看NURBS曲线的求导方法，首先有



求导得到



化简得到



那么只需要先根据之前的方法，求出齐次坐标下的导矢，然后根据得到导矢的分量就可以计算NURBS曲线的导矢。

**4.2.2 高阶导矢的计算方法**

根据莱布尼兹公式，有



化简得到



进一步得到



此时就得到了一个递推公式，由于***A***和的任意阶导数都是容易获得的，所以可以从1阶开始往上不断递推，比如说有



NURBS曲线在端点处的导矢的计算方法，由之前B样条曲线端点导矢的公式，即



可以得到



以及



那么NURBS曲线在端点处的导矢为



化简得到



**注：是不具有规范性的，但有理基函数*R*具有规范性。**

同理有



化简得到



然后，实际上B样条的所有操作都可以转换到NURBS曲线上面来，现在给出下面的例子：

**例：**令一条NURBS曲线基于节点矢量



控制点表示为



以及权系数为



那么可以先利用之前写出的NURBS曲线的端点导矢的计算公式有



那么再由B样条曲线的端点二阶导矢公式



化简得到

再来看权系数



再带入NURBS曲线的公式有



*u*取1的情况不谈了，是一样的**（上面明显为0所以没有计算）**。

**算法. 通过之前给出的递推方法来计算NURBS曲线的任意阶导数**

RatCurveDerivs( Aders, wders, d, CK )

{

现在设通过之前的B样条曲线求导矢的方法，已经求出了齐次坐标下的B样条的任意阶导矢，并将分量***A***放在了Aders中，将放在了wders中。

for( int k=0;k<=d;k++ )

不需要对d和p进行讨论了，因为在B样条曲线求导矢的过程中这个部分就被完成了

{

v=Ader[k];

for( int i=1;i<=k;i++ )

v=v-Bin[k][i]\*wders\*CK[k-i];

CK[k]=v/wders[0];

**Bin[k][i]表示**

}

}

**4.3 NURBS曲面的定义和性质**

**4.3.1 NURBS曲面的定义**

一张在*u*方向*p*次，*v*方向*q*次的NURBS曲面是具有以下形式的双变量分段有理矢值函数：



式中



引入分段有理基函数



**那么原式可以写为**



**4.3.2 NURBS曲面的性质**

**只列举几个比较重要的性质**

**（1）规范性：**之前一直没领会到一个含义，对所有的，有



上述表达含义在于，一定是对在区间下标这个范围内的区间参数，才能够具有上述的规范性；

**（2）端点基函数：**；

**其他的性质和B样条曲面基本一致，在此不赘述。**

和NURBS曲线一样，为了达到修改曲面的目的，可以通过调整控制点，同样也可以通过调整权系数来修改曲面，而且当保持*u*和*v*不变，此时修改，得到的同样是一条直线，因为有



同样可以通过齐次坐标的形式来定义NURBS曲面



**算法.** 只需要对前文的B样条曲面计算方法稍作修改就能够用来计算NURBS曲面

SurfacePoint( n, p, U, m, q, V, Pw, u, v, S )

{

uspan=FindSpan( n, p, u ,U );

BasisFuncs( uspan, u, p ,U, Nu );

vspan=FindSpan( m, q, v, V );

BasisFuncs( vspan, v, q, V, Nv );

for( int j=0;j<=q;j++ )

{

temp[j]=0.0f;

for( int i=0;i<=p;i++ )

temp[j]=temp[j]+Nu[i]\*Pw[uspan-p+i][vspan-q+j];

}

Sw=0.0f;

for( int j=0;j<=q;j++ )

Sw=Sw+temp[j]\*Nv[j];

S=Sw/w;

}

**4.4 NURBS曲面的偏导矢**

采用与NURBS曲线推导导矢类似的方法，有



那么可以推导得到



**注：包括之前B样条曲面在内，求偏导的时候都设求偏导的顺序不影响最终的求导结果。**

**那么有**



**同理将0阶的量提出来有**



**那么可以得到**



**算法.** 用于求NURBS曲面上任意阶的偏导矢，需要注意的是，此时令Aders为求出的，wders为求出的，这两个量是通过将原控制点转换为齐次坐标形式，然后进行B样条曲面的求偏导过程，然后齐次坐标的前三个分量变为***A***，最后一个分量变为。

RatSurfaceDerivs( Aders, wders, d, SKL )

{

for( int k=0;k<=d;k++ )

{

for( int l=0;l<=d-k;l++ )

{

v1=Aders[k][l];

for( int j=1;j<=l;j++ )

v1=v1-Bin[l][j]\*wders[0][j]\*SKL[k][l-j];

for( int i=1;i<=k;i++ )

{

v1=v1-Bin[k][i]\*wders[i][0]\*SKL[k-i][l];

v2=0.0f;

for( int j=1;j<=l;j++ )

v2=v2+Bin[l][j]\*wders[i][j]\*SKL[k-i][l-j];

v1=v1-v2\*Bin[k][i];

}

SKL[k][l]=v1/**wders[0][0]**;

}

}

}

**5 基本几何算法**

**5.1 节点插入**