CAM Part I: Tangents and Normals

We have concentrated on the popular mathematical techniques for modeling of the various kinds of shapes that can occur in the design of mechanical components. Now we shall look at the essential mathematics required for the computation of important manufacturing data using the CAD information.
As before, we shall concentrate on the geometric aspects of the computations. There is other information required to complete the manufacturing plans. This includes the following:

Determination of the manufacturing process
Selection of the machine tool that will be used
The tools that will be used
Determination of 'which tool will machine which region'
The fixtures that will locate and hold the part while it is being machined,
The sequencing of the machining operations,
The sequencing of the setups (one setup => one fixture)
The cutting conditions (speed, feed rate, depth of cut, coolant)
The cutting motion and the motion plan

All the above decisions, taken together, constitute what is known as Process Planning. If they are mostly done using computers, the software used is a Computer-Aided Process Planning (CAPP) system. At this time, there is no industrially useful CAPP system. Optimal CAPP decision-making is complex. In fact, several of the functions mentioned above cannot be solved optimally; further, CAPP decision making requires forming the optimum combination of the outcome(s) of the individual functions, which is also algorithmically intractable. We shall look only at those issues which involve tool motion planning. The essential problems here are geometric in nature, and are fairly well defined.

There are two essential computations that will form the basis of most of the cutter path planning: computations of offsets, and intersections.
Offsets allow us to compute the tool location with respect to the surface we need to generate; it requires the computations of tangents and normals, which in turn require computing the derivatives of the curve equations.
Intersections allow us to plan the tool motion along paths that are 'good' from a machining point of view. They require solving systems of equations simultaneously, and may require numerical techniques for most practical surfaces.

Derivatives of NURBS Curves

In several manufacturing operations, it is important to compute the tangent(s) and normal to curves or surfaces that define the geometry of the part. For 2D shapes, given a point on a curve, the first derivative gives us the tangent, from which we can compute the normal. For surfaces, we need the first and second partial derivatives of the parametric representations, as discussed in the earlier lectures.

As we have seen earlier, Bezier and B-Splines are merely special cases of NURBS. Hence we shall directly attempt finding derivatives of NURBS curves/surface. We can always use the same formulae for the other two forms.

Recall that NURBS are rational functions – and from our high-school calculus, higher order derivatives of rational functions are very messy because of the terms in the denominator. We shall avoid some of this complication by the use of a very useful transformation. In the process of doing so we get an introduction to projective geometry, and in particular perspective projections. We shall handle these projections through the use of homogeneous coordinates.

Projective Geometry

In order to get a decent understanding of the mathematics used in computing derivatives of NURBS, one needs to study a little bit the subject called projective geometry. Recall from the appendix in our first lesson that affine geometry is the study of properties that remain invariant under parallel projections (the distance between any two points of an object is equal to the distance between the corresponding points on the image of the object obtained by a projection). Likewise, projective geometry is the study of properties that remain invariant under projective transformations. Projective transformations are obtained by a process of central projection (you may think of this process as follows: affine projections mean that parallel projection lines are used to find the image of any point on the object; in projective transformations, all projection lines originate from a fixed point, which we may call the center of projection). The shadow of object when the light source is the sun (nearly parallel rays) is an affine projection; the shadow of this object when the light source is a nearby street-lamp is a projective projection.
Some additional notes with an introduction to projective geometry will be made available to you, but to remain focused on our task, we just begin assuming some background.

An interesting concept arising out of projective geometry (and particularly the work of mathematicians such as Klein, Moebius and Plucker) was that of homogeneous coordinates, which have many beautiful applications.
The basic implementation of homogeneous coordinates is simple, and since homogeneous coordinates are abstract, but we use them to represent points in Euclidean 2D or 3D space (denoted E2, or E3) which are easier to picture in our minds, so we learn the correspondence between these two types of representations.

We shall represent a point in E2 by a 2-tuple (x, y).
The corresponding homogeneous coordinate is a 3-tuple (kx, ky, k), where $k \neq 0$.
In other words, homogeneous coordinates are n-tuples of real numbers, but the entity (0, 0, …, 0) is not defined.

The first interesting thing about homogeneous coordinates is that scaling does not change a point – thus $(x, y, 1)$ is the same as (kx, ky, k) for any real number k.

The correspondence between Euclidean and homogeneous coordinates is defined as follows: *(x, y)* ≡ *(x, y, 1)*.
Therefore, for any homogeneous point (*a, b, r*), we can get the corresponding point in E2 as *(a/r, b/r, r/r) = (a/r, b/r, 1)*. such a mapping from the projective space to Euclidean space, which sends the last element to 1, is called a perspective mapping (this is explained in the additional notes), and is often denoted as *H*. So we say *H: (a, b, r)* → *(a/r, b/r, 1)* ≡ *(a/r, b/r)* in E2.

These definitions can easily be extended to a point (*x, y, z*) in E3, corresponding to which we get the homogeneous points *(kx, ky, kz, k), k* ≠ *0*.
And correspondingly, if we have a point, (*x/w, y/w, z/w*) in E3, an equivalent homogeneous representation could be (*x, y, z, w*). This correspondence is important – and we shall use it in the following section.

Derivatives of Rational Curves

Earlier, we had derived the formula for derivative of B-Spline basis functions, $N'_{i,p}(u)$, which can be used to write the derivatives of a given B-spline curve: if $\boldsymbol{C}(u) = \sum N_{i,p}(u)\,\boldsymbol{P}_i$, then $\boldsymbol{C'}(u) = \sum N'_{i,p}(u)\,\boldsymbol{P}_i$. We now do some more algebra with those derivations, to get a format useful in differentiation of rational B-splines (NURBS).

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u)\,P_i, \quad for\ a\ knot-vector\ U = \{0,\ldots p+1 times\ldots,0,\ldots,1,\ldots p+1 times\ldots,1\}$$

$$\begin{aligned}
C'(u) &= \sum_{i=0}^{n} N'_{i,p}(u)\,P_i \\
&= \sum_{i=0}^{n}\left(\frac{p}{u_{i+p}-u_i}N_{i,p-1}(u) - \frac{p}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}(u)\right)P_i \\
&= \left(p\sum_{i=-1}^{n} N_{i+1,p-1}(u)\frac{P_{i+1}}{u_{i+p+1}-u_{i+1}}\right) - \left(p\sum_{i=0}^{n} N_{i+1,p-1}(u)\frac{P_i}{u_{i+p+1}-u_{i+1}}\right) \\
&= p\frac{N_{0,p-1}(u)\,P_0}{u_p-u_0} + p\sum_{i=0}^{n-1} N_{i+1,p-1}(u)\frac{P_{i+1}-P_i}{u_{i+p+1}-u_{i+1}} - p\frac{N_{n+1,p-1}(u)\,P_n}{u_{n+p+1}-u_{n+1}} \\
&= p\sum_{i=0}^{n-1} N_{i+1,p-1}(u)\frac{P_{i+1}-P_i}{u_{i+p+1}-u_{i+1}} \quad [\sin ce\ first\ and\ last\ terms\ are\ \frac{0}{0}=0] \\
&= \sum_{i=0}^{n-1} N_{i+1,p-1}(u)\,Q_i \qquad\qquad where
\end{aligned}$$

$$Q_i = p\frac{P_{i+1}-P_i}{u_{i+p+1}-u_{i+1}}$$

If we let U' be a knot vector obtained by deleting the first 0 and last 1 of the knot vector U, that is,

U' = { 0, … p times, 0, u_{p+1}, …, u_{m-p-1}, 1, … p times …, 1} {note: U' has m knots], then it can be verified that each of Ni+1,p-1(u) computed over U is the same as Ni,p-1(u) computed over U'. That is,

$$C'(u) = \sum_{i=0}^{n-1} N_{i,p-1}(u) Q_i$$

Thus the derivative of a degree-p B-spline is also a B-spline, of degree-(p-1), over the knot vector U', and with control points derived from the difference of consecutive control points of the original curve !

Let us not return to the topic we love – how to find derivatives of rational functions, such as those used to describe NURBS. we have earlier seen that the formulas (and corresponding algorithms) for derivatives of non-rational B-splines are relatively easy to derive, using some basic calculus and lots of algebra. Let's see the steps in understanding the use of homogeneous coordinates.

Step 1.. Recall that for each NURBS curve, associated with each Control point $\boldsymbol{P}_i$ is a weight, w_i. We map the control points into the 4-dimensional perspective space as follows: $\boldsymbol{P}_i^w = (w_i x_i, w_i y_i, w_i z_i, w_i)$.

Step 2. Let's try to define a 4-dimensional B-spline curve, using the 4D $\boldsymbol{P}_i^w$'s:

$$C^w(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i^w$$

Step 3. Let us now try to project this 4D curve onto the 3D Euclidean space using our usual transformation (dividing each of the first three elements of each vector by the 4th vector). Notice that the 4th element of $\boldsymbol{C}^w(u)$ can be written as: $\Sigma\ N_{i,p}(u)\ w_i$. Thus projection of the 4D curve $\boldsymbol{C}^w(u)$ to Euclidean 3D space using the mapping above (divide each of the first three elements by the fourth) gives us a curve:

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i}.$$

[Why do we have the w_i's in the numerator ?]

But this is precisely the representation of a NURBS curve in E3. Hence, a rational B-spline in 3D can be represented equivalently by a non-rational curve in a 4D projective space.

Why do we do all this ?

We set out to find derivatives of NURBS. Recall from your high-school calculus that derivatives of functions of the form $f(u)/g(u)$, with respect to u, are not pretty – since they will involve a term with $[g(u)]^2$ in the denominator. And the second derivative will have $[g(u)]^3$ in the denominator, and so on..

On the other hand, derivatives of non-rational polynomials are easy to work with.

Since $\boldsymbol{C}^w(u)$ is non-rational, polynomial, we can easily find all of its derivatives easily, just as we do for our usual 3D B-spline curves.

Thus we shall now derive the relation between derivatives of a NURBS with the derivatives of $\boldsymbol{C}^w(u)$ – and then use this relation to compute derivatives of any NURBS curve $\boldsymbol{C}(u)$.

We start by defining a function, $w(u) = \sum_{i=0}^{n} N_{i,p}(u)\, w_i$ (which is the denominator of the NURBS equation.

$\boldsymbol{C}(u) = w(u)\boldsymbol{C}(u)/w(u) = \boldsymbol{A}(u)/w(u)$

Thus A(u) is a vector valued function with three scalars, which are basically the first three terms of $\boldsymbol{C}^w(u)$. Differentiating the above,

$$C'(u) = \frac{w(u)A'(u) - w'(u)A(u)}{w(u)^2} = \frac{A'(u) - w'(u)C(u)}{w(u)}$$

Since we can write $\boldsymbol{C}^w(u) = [\, \boldsymbol{A}(u)\, w(u)\,]$, so we compute their derivatives just as we find derivatives of a non-parametric B-spline curve.

Thus, computing $\boldsymbol{C}'(u)$ at a given point, u_0, requires us to (a) compute the derivatives of the equivalent homogeneous form and evaluate them at u_0, (b) compute the curve vector at $u = u_0$, (c) compute $w(u)$ at u_0, and (d) put it all in the formula for $\boldsymbol{C}'(u)$ above.

In fact, using this idea, the formula for the k-th derivative of the curve can be derived:

$$A^{(k)}(u) = (w(u)C(u))^{(k)} = \sum_{i=0}^{k} \binom{k}{i} w^{(i)}(u) C^{(k-i)}(u)$$
$$= w(u)C^{(k)}(u) + \sum_{i=1}^{k} \binom{k}{i} w^{(i)}(u) C^{(k-i)}(u)$$

and with some more algebra, we can finally obtain:

$$C^{(k)}(u) = \frac{A^{(k)}(u) - \sum_{i=1}^{k} \binom{k}{i} w^{(i)}(u) C^{(k-i)}(u)}{w(u)}$$

What does this formula tell us ? That we can compute the *k*-th derivative of a NURBS curve, ***C(u)***, if we know how to compute the following:
(a) *k*-th derivative of ***A(u)***,
(b) First through (*k-1*)th derivatives of *w(u)* and ***C(u)***
(c) *w(u)*

From our discussion above, all the derivatives of ***A(u)*** and w(u) are computed by the same formulas as for non-parametric B-Spline curves using homogeneous coordinates; *w(u)* can directly be computed since it only needs computing the basis functions and the weight of each control point; finally, we need all the derivatives (form *0*- to (*k-1*) of ***C(u)*** – these are computed iteratively, so we only need to know how to compute the zero-derivative, i.e. a point on the curve ***C(u)***.

Exercise:
We are mostly be interested only in the first and the second derivatives. Write the formula for the second derivative explicitly from the equation above (replace *k* by 2, expand the summations, and replace the *w(u)*'s by the $N_{i,p}(u)$'s and w_i's.)

Once we can compute the first and second derivatives of a NURBS curve, we can derive tangent (and unit tangent), and the normal (and unit normal).

Surface Derivatives

As we saw in the case of determination of surface points for B-Splines and NURBS, it is easy to derive formulas for surfaces once we know the corresponding formulas for curves – this advantage arises out of our choice to restrict ourselves to tensor product surfaces, where the *u*-functions and the *v*-functions are (a) of the same form, and (b) factorized out.

Recall that a normal vector to the surface **S**(u,v) is given in terms of the partial derivatives $\mathbf{S}_u$(u,v) and $\mathbf{S}_v$(u,v), as $\mathbf{n} = \mathbf{S}_u \times \mathbf{S}_v$. Thus the tangent plane at point **p** is (**x**-**p**).**n**

We first look at the partial derivatives of B-Splines, and then use homogeneous coordinates to compute the partial derivatives of NURBS.

Derivatives of B-Spline Surfaces

$$S_u(u,v) = \sum_{j=0}^{m} N_{j,q}(v) \left(\frac{\partial}{\partial u} \sum_{i=0}^{n} N_{i,p}(u)\ P_{i,j} \right)$$

As you can see, the term inside the parenthesis is basically the derivative of a B-spline curve.

$$S_u(u,v) = \sum_{j=0}^{m} \sum_{i=0}^{n-1} N_{i,p-1}(u)\, N_{j,q}(v)\, P_{i,j}^{(1,0)} \qquad where$$

$$P_{i,j}^{(1,0)} = p \frac{P_{i+1,j} - P_{i,j}}{u_{i+p+1} - u_{i+1}} \qquad over\ the\ knot-vectors$$

$$U^{(1)} = \{0,...p\ times...,0, u_{p+1,} ..., u_{r-p-1}, 1,...p\ times...,1\}$$

$$V^{(0)} = V$$

and similarly:

$$S_v(u,v) = \sum_{j=0}^{m-1} \sum_{i=0}^{n} N_{i,p}(u)\, N_{j,q-1}(v)\, P_{i,j}^{(0,1)} \qquad where$$

$$P_{i,j}^{(0,1)} = q \frac{P_{i,j+1} - P_{i,j}}{v_{j+q+1} - v_{j+1}} \qquad over\ the\ knot-vectors$$

$$U^{(0)} = U,$$

$$V^{(1)} = \{0,...q\ times...,0, v_{q+1,} ..., u_{s-q-1}, 1,...q\ times...,1\}$$

To find subsequent higher order derivatives, just repeatedly apply the above forms (we can do so, since each subsequent partial derivative is also a B-Spline, as we saw before).

In general:

$$\frac{\partial^{k+l}}{\partial^k u\, \partial^l v} S(u,v) = \sum_{i=0}^{n-k} \sum_{j=0}^{m-l} N_{i,p-k}(u)\, N_{j,q-l}(v)\ P_{i,j}^{(k,l)} \qquad where$$

$$P_{i,j}^{(k,l)} = (q-l-1) \frac{P_{i,j+1}^{(k,l-1)} - P_{i,j}^{(k,l-1)}}{v_{j+q+1} - v_{j+l}}$$

The above results are useful in obtaining derivatives of NURBS surfaces – since, as in the case of NURBS curves, we shall transform the surface into an equivalent one, $\mathbf{S}^w(u,v)$ in the 4D homogeneous space; derivatives of $\mathbf{S}^w$ are computed exactly the same way as above for B-Spline surfaces. We look at expressing the derivatives of $\mathbf{S}(u,v)$ in terms of those of $\mathbf{S}^w(u,v)$. I will skip the details, which are in the NURBS book (Piegl and Tiller). Writing S(u,v) as below,

$$S(u,v) = \frac{w(u,v)\,S(u,v)}{w(u,v)} = \frac{A(u,v)}{w(u,v)}$$

we can establish that:

$$S_u = \frac{A_u - w_u S}{w}$$

$$S_v = \frac{A_v - w_v S}{w}$$

$$S_{uv} = \frac{A_{uv} - w_{uv} S - w_u S_v - w_v S_u}{w}$$

$$S_{uu} = \frac{A_{uu} - 2w_u S_u - w_{uu} S}{w}$$

$$S_{vv} = \frac{A_{vv} - 2w_v S_v - w_{vv} S}{w}$$

This information is sufficient for us to perform most operations related to CAM of surfaces, described by any of our usual means – Bezier, B-Splines, or NURBS. Note that to find the cutter location data for machining a surface, we need to generate an offset surface to a given surface (the offset distance is related to the cutter geometry – for a ball-end mill, it is equal to the radius of the ball-end of the mill, in a direction along the surface normal).
Note that the offset surface (by a distance d) to a NURBS (or B-Spline) surface is given by:

$$S^o(u,v) = S(u,v) + d\,n(u,v), \quad \textit{where } n(u,v) = \frac{S_u \times S_v}{\left\| S_u \times S_v \right\|}$$

Since the denominator in the expression for the normal vector, n(u,v) has a term with a square-root, it cannot, in general, be a polynomial. Hence the Offset surface to any polynomial form cannot be a rational polynomial. In other words, the offset of a NURBS surface is not a NURBS. In fact, even the offset to a Bezier or B-Spline is not a NURBS. Therefore, when forming offset surfaces, we shall just use approximations. A typical method is to construct the offset surface out of a mesh of points, each of which is the offset to a corresponding mesh point on the given surface. If required, a NURBS surface can be fit into the offset mesh.

In the next set of notes, we shall look at the basic techniques needed for machining applications – surface-surface intersections, and its special case, surface-plane intersections.

--
Source:
The NURBS Book, Piegl and Tiller.