Parametric Curve Interrogations

topics

Differential Geometry
Basic Interrogations (the API)
Graphics
Incidence Interrogations

Differential Geometry

In standard differential geometry it is always assumed that a curve is parametrised with respect to arc length. In CAGD it is necessary to face the reality that arc length parametrisations are actually hard to achieve and expensive to implement, so that we cannot assume that the parameter is arc length[1]

A useful mental image is to interpret t as time, and think of a point moving along the curve.

What is the derivative of a curve with respect to t ?

The fundamental definition of a derivative is as the limit of the ratio of differences

$$dP/dt = Limit_{dt \to 0}(P(t + dt) - P(t))/dt$$

Now $(P(t+dt) - P(t))$ is just a chord. As dt shrinks, $P(t+dt)$ moves around the curve towards $P(t)$ and in the limit dP/dt becomes a tangent. From this definition we see that it is a displacement.

If t is time, then $P'(t)$ is just the velocity vector, whereas in classical differential geometry the tangent is a unit vector.

---

[1] **Arc Length Parametrizations**: There are some very lovely recent results (look up Rida Farouki at UC Davis) which show that it is impossible for any parametric polynomial apart from the straight line to have arc length parametrization.

However, it is possible for a parametric polynomial to have an arc length which is a rational polynomial in terms of parameter. These 'rational speed curves' have a number of degrees of freedom the same as the ordinary rational parametric polynomials of just over half the degree. The theory ties together Pythagorean triangles whose sides are polynomials, not numbers, (the formula $x = u^2 - v^2; y = 2uv; z = u^2 + v^2$ still works when u, v, x, y, z are all functions) and the squaring of complex numbers.

The interesting, but unexplained, fact is that aesthetically they are rather more beautiful than their rational equivalents.

Because of this complication CAGD people have invented the distinction between **parametric continuity** and **geometric continuity**. If two pieces of curve have the same value of dP/dt where they meet the junction is said to be **C1**: if they merely have the same direction and sense of tangent vector, but different magnitudes of $|dP/dt|$, the junction is said to be **G1**.

A G1 junction can always be turned into a C1 one by just rescaling the parameter on one side of it. The actual shape of a curve does not depend on the rate at which it is traversed.

What about second derivatives ?

If the first derivative is velocity, the second derivative is acceleration.

This is closely lnked to the curvature of a curve. In fact, the component of acceleration perpendicular to the curve is proportional (at a constant speed) to the curvature. The component of acceleration along the curve is not linked to the curvature at all.

We draw a similar distinction between **C2** and **G2** continuity. In G2 continuity the actual curvature of the curve is the same on both sides of a junction, while in C2 the second derivative vector is the same. A good intuitive mental image of a G2 junction is a car driving round a constant radius track, where the driver suddenly puts his foot down on the accelerator. The acceleration vector changes, but the curvature of the path does not.

Basic Interrogations (the API)

In software design it is considered good practice to restrict dependence on the detail of the representations of objects to a small set of kernel methods related to each object.

In the case of parametric curves we can indeed do this, limiting detailed knowledge of the basis functions and the coefficients to just a few routines:

Evaluation: real $\rightarrow$ point

This routine takes the value of t as argument and returns the position of the corresponding point. The result is always with respect to the same coordinate system as that in which the coefficients are represented.

In the examples of interrogation using this routine it will just be denoted by $P(t)$

Derivative evaluation: real $\rightarrow$ derivs

In principle, derivatives could be evaluated by differences, and so this routine is not strictly necessary, but it is frequently much more efficient to evaluate derivatives with respect to t at the same time as evaluating the actual position. There are also some robustness issues which make it sensible. For formal purposes we say that derivatives of any degree are accessible through this method.

In the examples of interrogation using this routine it will just be denoted by $dP(t)$, or $d^2P(t)$ depending on which derivative is being evaluated.

Bounds data: $\rightarrow$ real x real

We have defined curves so far as always being maps from [0..1] to the segment of curve. In fact it is possible that curves in a practical CAD system might be **trimmed**, and so any enquiry might need to know the parametric interval which is mapped into the valid piece of curve.

In the examples of interrogation using this routine it will just be denoted by the use of the variables t_{min} and t_{max}, so that the actual extraction of this data is implicit.

Graphics: Drawing a Curve

There are two stages in this process. One is to calculate a sequence of points along the curve, which will then be joined by straight line segments by the graphics device. The other is to apply an appropriate transformation so that the required view of the curve is seen.

Because of the coordinate system invariance of the curves that we actually use, we can carry out these two parts of the process in either sequence, either transforming the representation to camera coordinates first and then rendering, or else calculating points and then mapping them to camera coordinates afterwards. For drawing curves the former is often more efficient, but for other interrogations the balance may favour the other approach.

Here we consider only the rendering process, of finding a sequence of points along the curve. There are three approaches:

The first and simplest is just to evaluate points at a very large number of values of parameter.

The problem here is to know how many are needed. The inefficiency comes from the fact that the necessary densities for the appearance of smoothness usually varies from one part of the curve to another. Note that for good efficiency we want to minimise the number of points evaluated, but even more important is to minimise the numer of segments issued through the graphics system. The cost of turning a short segment into illuminated pixels is probably quite a lot larger than that of evaluating a parametric curve.

For graphical purposes a good measure of density is the **chord height error**, the distance from the point at the middle of a chord to the local curve. A conservative estimator of this is the distance from the point at the middle of the chord to the curve point at a parameter value halfway between those at the ends of the chord.

If we have upper bounds on the magnitude of the second derivative vector, these can be used to determine the necessary density.

The second approach is to start at one end of the curve, and work along to the other, choosing the step at each stage to give about the optimum chord height error.

The magnitude of the first step can be guessed by looking at the first and second derivatives at the start point. Subsequently it is more efficient to use a bit of the history of the process. Adjust the step taken depending on the actual chord height error of the previous step.

The third approach is the optimistic one; assume that the curve is almost straight, and only add points into the interior of the sequence when evidence is found that there is actually a bend in it.

This approach was developed by Douglas and Peucker for reducing the scale of maps, where the available data was a dense string of points and a sparser one was required for a less detailed map.

We call the second method **marching** and the third **refinement** or **homotopy**.

Incidence Interrogations

We use two examples of interrogations, although there are many more. The ones we elaborate are

Curve-Plane

Here we wish to find the point in which the curve passes through a given plane.

Closest Approach

Here we wish to find the points, one on each of two curves, which minimise the distance between them.

Curve-Plane Intersection

If we use the implicit plane equation $f(P) = 0$, we can in principle substitute the parametric equation of the point into it, giving an equation in terms of the scalar t for which we want to find the roots.

$$f(F(t)) = 0$$

Actually doing the algebraic substitution is generally very little help, as it violates the principle of separation of levels of functionality, and while very good root-finders are available for polynomials, there are rather few for spline functions.

It does help, however, to view the intersection process as a special-purpose root-finder, and steal ideas from the root-finders in the standard libraries.

In this context the two main ones are

– Newton iteration to home in on a root

– Bracketting, to narrow the range in which a root lies.

Newton iteration is a classical process for improving an estimate of the position of a root. Suppose that we have an estimate t_0 for the position of a root, we can evaluate both the value and the derivative of the function there.

$$\begin{aligned}
P_0 &:= P(t_0) \\
f_0 &:= [P_0 - Q].N \\
P'_0 &:= dP(t_0) \\
f'_0 &:= P'_0.N
\end{aligned}$$

From these we can estimate a better position for the root

$$t_1 := t_0 - f_0/f'_0$$

This in turn can be used as a starting point for a further improvement, and the process repeated until the value of f is small enough.

This is a very fast process: under normal circumstances it converges quadratically, which means that the number of correct digits in t doubles at every step. If, as is reasonable, we start with two or three binary digits correct, we can get down to double precision floating point precision in 5 steps.

However, there are a few problems

* We need to find an initial estimate from which to start. The iteration is not guaranteed to converge at all.

Indeed, it is easy to set up counterexamples where it definitely diverges.

* Near a double root the process slows down, only adding one binary digit to the precision at each step.

* It finds only one root. There may be many intersections, and in some contexts we need to find them all.

Current practice is pretty ad hoc. We can address the initial estimate question by taking a spatter of points along the line, but who can say how dense that spatter needs to be ? We can address divergence by trapping when the process gives an estimate outside [0..1], but what should we do then ? We can address multiple roots by iterating to a solution from each of the points of the spatter, but we then need to weed out duplicate roots which have resulted from converging to the same place from different starting points. If f' is small, they may still have different values of t when f says that they have both converged. CAD systems have ways of doing something sensible, but these are not appropriate to be taught in a maths course.

One technique which I have not seen in the CAGD literature, but which is used in standard root-finders, is **deflation**. If we are solving for the roots of a polynomial, we can ensure that any second root found will not be a duplicate by setting up a new function, consisting of the old one divided by the distance of t from the known root. In the case of a polynomial, we can modify the representation to give a new polynomial explicitly of lower degree. In our more general context we cannot do that, but we can do the division numerically every time f and f' are evaluated. This would not work well in the presence of multiple roots, but those can be tested for by looking at the derivatives at any root actually found.

Bracketting, also called **Regula Falsi** or **Iterated Inverse Linear Interpolation** is a more robust process, though it is not usually as fast. In this process we scan along the curve to find a change of sign of f, and as soon as we have two values of t for which the signs of f are different, we set up the interval between them as $[t_l \dots t_u]$.

Now find the value t_n at which the chord from (t_l, f_l) to (t_u, f_u) cuts $f = 0$. This must lie within our interval, because the values of f at the ends have different signs.

Evaluate f_n at t_n. This may be zero, in which case the root is found, or it may be positive or it may be negative. If it is not zero, replace as a bound of the current interval whichever of t_l, t_u has the same sign of f. The width of the interval gets strictly smaller at each iteration. When it is narrow enough the iteration can stop anyway.

This process is only linearly convergent, adding a fixed number of correct digits to the precision of t at each stage. That number depends on the second derivative of the curve over the interval. (For almost straight curves it is obviously very fast.)

There are counter-examples where the speed of convergence is very low, but these can be countered by ad-hoc variation of the algorithm, or by fitting something other than a straight line (a rational function, linear divided by quadratic, is suggested in the literature, chosen using the derivatives at the end of the current interval).

Bracketting is much more robust than Newton, because it cannot diverge. It has a problem with double roots, because there is no sign-change to be exploited there. The finding of many intersections, rather than just one, is also a somewhat ad hoc process.

The main limitation of Bracketting, however, is that it does not generalise to multiple dimensions. Newton is therefore the principal paradigm in practical use.

Closest approach of two curves

Consider the case where we have two curves $P(t)$ and $Q(s)$, where s and t are independent parameters.

We want to find the values of s and t which minimize the value of $|P(t) - Q(s)|$.

It is easiest to deal with the square of this distance

$$w = [P(t) - Q(s)].[P(t) - Q(s)]$$

Focussing on the 'usual' case, where this minimum occurs in the interior of both curves (not at an end point), and where there is only a single local minimum, the problem becomes the finding of points at which

$$f(s,t) = dw/ds = -2dQ/ds.[P(t) - Q(s)] = 0 < /i >$$
$$g(s,t) = dw/dt = 2dP/dt.[P(t) - Q(s)] = 0 < /i >$$

If we have an estimate of the pair (s_0, t_0) (This is the same thing as a pair of estimates, but I want to think of the pair of parameters as a single object) we can evaluate f, g, df/ds, df/dt, dg/ds, dg/dt there.

Setting up a bivariate Taylor series about (s_0, t_0) gives

$$\begin{matrix} f(s+ds,t+dt) \\ g(s+ds,t+dt) \end{matrix} \approx \begin{matrix} df/ds & df/dt & ds \\ dg/ds & dg/dt & dt \end{matrix} = \begin{matrix} 0 \\ 0 \end{matrix}$$

The right hand side of this can be solved for ds, dt which can be added to the original estimate to give us a new iteration. This is essentially Newton's method in a multivariate setting. It works just as well / just as badly as in the univariate case.