Mathematics 308—February 12, 1996

Drawing parametrized paths

Here is the main routine we shall use from now on for drawing parametrized paths. It is a bit complicated, and in these notes I shall explain how it works as well as how to use it.

```
% stack:  [parameters] /f t0 t1 N

/mkpath {
16 dict begin
/N exch def
/t1 exch def
/t0 exch def
/f exch cvx def
/pars exch def

/h t1 t0 sub N div def
/h3 h 0.333333 mul def

/currentloc pars t0
f def
pars t0 f 0 get
aload pop

thereisacurrentpoint
{
lineto
}
{
moveto
}
ifelse

N { % x y = currentpoint

currentloc 0 get 0 get % x0 dx0
currentloc 1 get 0 get
h3 mul
add

currentloc 0 get 1 get
currentloc 1 get 1 get
h3 mul
add

/t0 t0 h add def
/currentloc pars t0 f def
currentloc 0 get 0 get
currentloc 1 get 0 get
h3 mul sub
currentloc 0 get 1 get
```

```
currentloc 1 get 1 get
h3 mul sub
currentloc 0 get 0 get
currentloc 0 get 1 get
curveto
} repeat
end
} def
```

How to use it

The input to the procedure consists of five items on the stack. First comes an array of parameters, which I shall say something more about in a moment. Then follows the name of a routine specifying the parametrization in PostScript. Third and fourth are the initial and final values of the parameter associated to the segment of the path to be draw. Fifth is the number of Bezier segments to be drawn.

The most important item here is the parametrization routine. It must have as input two items, an array of parameters which may be empty and a value for the path parameter t. Its output (left on the stack) will be a 2×2 matrix written according to my conventions as an array of two $2D$ arrays.

For example, suppose we want to draw circles of varying radii, centred at the origin. The parametrization of such a circle is

$$t \mapsto (R\cos t, R\sin t)$$

and the velocity vector of this parametrization is if t is in radians. Then the variable t is the parameter which traverse the path, while R is a parameter specifying which of several possible circles are to be drawn. Thus the input to the circle drawing routine will be a pair `[R] t` and output will be `[xt yt][dxt dyt]]`.

Here is a more explicit block of PostScript code (assuming π defined elsewhere). Note that PostScript uses degrees instead of radians, but that the formula for the velocity vector assumes radians, and we must convert from radians to degrees.

```
/circle {
1 dict begin
/t exch def
/pars exch def
/R pars 0 get def
/t t 180 mul pi div def
[
[ t cos R mul t sin R mul ]
[ t sin neg R mul t cos R mul ]
]
end
} def

(mkpath.inc) run

newpath
[2] /circle 0 2 pi mul 8 mkpath
closepath
stroke
```

This example shows among other things that the routine `mkpath` does not actuall draw a path, but builds it just as the PostScript `curveto` does, and allows the programmer to decide what to do with it — to stroke it or fill it, for example, or even clip to it. In fact, the routine `mkpath` does something a bit more complicated;

it adds to any path already drawn, by beginning with a straight line from the last point drawn to to the start of the path segment about to be built. This is the way the PostScript `arc` behaves, and it seems a good model to follow. At any rate, it is for this reason that the routine tests whether or not there already exists a current point. The overall effect is to allow you to build up a series of paths so as to make a rather complicated one. You might even draw several successive segments of the same path, for example, using a different value of the integer N (the fifth argument) to allow for parts of the path being straighter or more curvy.

Another possibly interesting point is how to pass a procedure to another procedure. What we do in fact is to pass the *name* of the procedure (begun with `/`) to our routine, and the routine converts the name to a procedure with the command `cvx`.

The array of parameters can be of any size you want. It can even be empty if in fact you just want to draw one of a kind. But you will probably find that most paths you want to draw are just part of a larger family.

The file `mkpath.inc`, which contains what you need for using this routine, is available on the course home page.

How it works

The basic idea is simply to automate the procedure you used in the last assignment to draw the hyperbola. The point is that if we are given a parametrization of a path we can draw an approximation by Bezier curves using the velocity vectors associated to the parametrization in order to construct control points. The routine is fairly straightforward, except that it calls a procedure `thereisacurrentpoint` to tell whether the path being drawn is the beginning of a new path or the continuation of an old one. You don't have to know the details of the procedure called — it operates in a very simple manner at a somewhat low level of PostScript.

This routine all by itself is very useful, and is capable of making interesting pictures. In combination with a programmed version of the chain rule from second year calculus it is capable of producing spectacular ones.