Extending METAPOST to 3D and 4D

L. Nobre G.

July 27, 2013

Abstract

Many authors have been working with three dimensions in METAPOST but, up to now, these works have not been integrated in the core of METAPOST. Since I am one of those authors, I would like to propose now a framing for that integration.

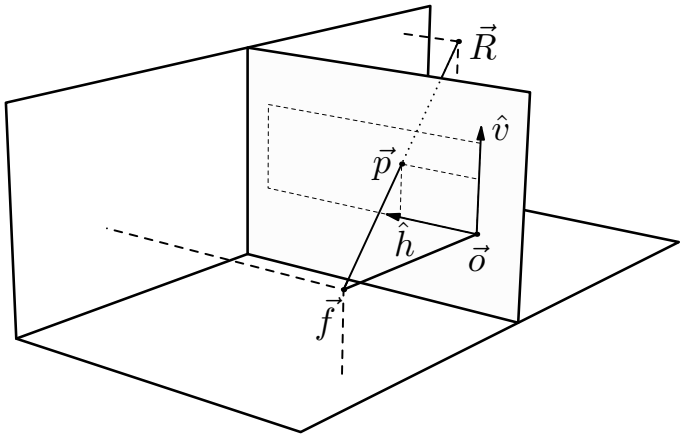
Contents

Since `colors` and `cmykcolors` were introduced in METAPOST, it became a simulation machine. Ordinary differential equations could be solved in 3D or 4D using the exact same Runge–Kutta code as for 2D.

But some way of projection had to be developed by the user in order to draw the results of the simulation.

1 Projecting from 3D into 2D

Many different kinds of projections exist, I tried a few and I think it is enough to project through a single point $\vec{f}$ onto a plane[1]. This kind of projection can take advantage of METAPOST's linear equations. Take, for instance, the following figure.



[1]When the point is at infinity, the projection is parallel and it defines only a direction.

Point $\vec{p}$ can be calculated from the following equations:

$$\begin{cases} \vec{R} + \lambda\left(\vec{R} - \vec{f}\right) = \vec{o} + \vec{p} \\ \left(\vec{h} \times \vec{v}\right) \cdot \vec{p} = 0 \end{cases} \tag{1}$$

which can be coded as below:

```
color p;
R + whatever*(R-f) = o + p;
(h crossproduct v) dotproduct p = 0;
```

where `R, f, o, h` and `v` are known `colors`. Of course, this requires that one defines `crossproduct`, the cross product of two vectors[2]. This kind of projection is called "perspective" and can easily be transformed into a parallel projection, like this:

```
color p;
R + whatever*(o-f) = o + p;
(h crossproduct v) dotproduct p = 0;
```

There are a few problems, though:

- Depending on the positions of $\vec{f}$ and $\vec{R}$ the linear equations may produce divisions by zero. But these equations are quite simple and can be solved in advance so that divisions by zero can be caught as is done in FEATPOST.
- $\vec{p}$ is a `color`, not a `pair`, so it must be converted. But this is very easy:

  ```
  x = p dotproduct h;
  y = p dotproduct v;
  ```

- if both $\vec{R}$ is a node or control point of a 3D path and the projection is a perspective then the path perspective is no longer a Bézier spline, it is a non-uniform rational basis spline (NURBS). The only way around this is to ignore the problem.

1.1 Exactly how?

Given that the actual way to draw a 3D path depends not only on the definition of that path but also on the kind of projection being used, I propose that the projection be done only at `shipout` time.

A perspective is not an affine transform and cannot be inverted by linear equations. A perspective is a new thing for METAPOST.

The parallel projection is an affine transform but METAPOST has not yet a framework neither for 3D nor for 4D `transforms`.

The transition from the actual METAPOST to a full–fledged 3D METAPOST depends critically on the support for smooth 3D and 4D paths.

Once METAPOST is beyond this transition what will certainly show up is the need for full path tansforms which may either be affine or not. I propose to call these "generic" transforms as `maps`. But note that these full path transforms cannot be perspectives because they only map each node and control point of a given path[3].

[2] I don't know how to define the cross product in 4D.

[3] A `map` may be regarded as an approximation of a perspective.

2 The whole package

2.1 Already on the Tracker

The next most basic 3D and 4D capability that needs to be added to METAPOST is already published as Tracker item # 104: both `abs` and `unitvector` must be expanded to accept `colors` and `cmykcolors` as arguments.

2.2 Cross product

A good `crossproduct` for METAPOST would accept `numerics`, `pairs` and `colors` as arguments.

`numerics` the result is `0`

`pairs A` **and** `B` the result is a `color` with all parts null except that the `bluepart=Ax*By-Ay*Bx`

`colors` the result is the standard cross product

2.3 angle

```
color A, B;
angle( A, B ) = angle( abs( A crossproduct B ), A dotproduct B );
```

2.4 Commands to be adapted

The following commands should be adapted to accept 3D (and 4D) points and paths:

```
draw undraw drawarrow drawdblarrow
fill unfill filldraw unfilldraw
reverse
precontrol postcontrol
arclength arctime
label dotlabel
slanted shifted rotated scaled xscaled yscaled
rotatedaround reflectedabout
bbox
subpath
path
transform transformed identity inverse
direction of
point of
```

And the following commands should be adapted to accept the new 3D and 4D `transforms`

```
xpart ypart xxpart xypart yxpart yypart
cyanpart magentapart yellowpart blackpart
```

2.5 New commands

`Colors` and `cmykcolors` should have analogues to `z` to be wiped–out on `beginfig`. I propose `C` for `colors` and `D` for `cmykcolors`. There should exist predefined names for unitary 4D vectors like `purecyan = (1,0,0,0); puremagenta = (0,1,0,0)`, etc.

```
zscaled tdtransform tdtransformed fdtransform fdtransformed
xzpart yzpart zzpart zpart zxpart zypart
ccpart cmpart cepart ckpart
mcpart mmpart mepart mkpart
```

```
ecpart empart eepart ekpart
kcpart kmpart kepart kkpart
mapped
```

This last one `mapped` is perhaps the most messy of all. It should work like this:

```
anypath mapped NameOfUserMap;
```

where `NameOfUserMap` is the name of a macro that uses as a single argument each node and control point of the `anypath`. It returns a new path that may have different dimensionality[4]

2.6 Commands that should not be touched

One of the problems of extending METAPOST is that somethings really cannot be extended. This is related with the smoothness of 2D paths. With the exception of cusp points, the `direction` is well–behaved on a 2D path. But on a 3D path (and specially on a 4D path) most directions cannot be found. Also, `intersections` become exceedingly hard to find.

[4]Standard affine `transforms` keep the dimensionality.