FEATPOST and a Review of 3D METAPOST Packages

L. N. Gonçalves
CFMC-UL, Av. Prof. Gama Pinto 2
1649-003 Lisboa
Portugal
nobre@lince.cii.fc.ul.pt
http://matagalatlante.org

Abstract

METAPOST is able to produce figures that look almost like ray-traced raster images but that remain vector-based. A small review of three-dimensional perspective implementations with METAPOST is presented. Special emphasis is given to the abilities of the author's implementation: FEATPOST.

1 Introduction

There are at least four METAPOST packages related to three-dimensional diagrams:

- GNU 3DLDF — `http://www.gnu.org/directory/graphics/3D/3DLDF.html`
- `3d/3dgeom` — `http://tug.org/tex-archive/graphics/metapost/macros/3d/`
- m3D — `http://www-math.univ-poitiers.fr/~phan/m3Dplain.html`
- FEATPOST — `http://matagalatlante.org/nobre/featpost/doc/featexamples.html`

All of these packages are individual and independent works "under construction". There has been neither collaboration nor competition among the authors. Each produces different kinds of diagrams and each uses a different graphic pipeline. The following sections of this document describe these packages, in a mainly independent way.

2 GNU 3DLDF

3DLDF is not a pure METAPOST package, as it is written in C++ using CWEB. Diagrams are also coded in C++ and are compiled together with the package. Nevertheless, this is, of all four, the package with the greatest promise for a future three-dimensional-capable METAPOST.

1. It outputs METAPOST.
2. Its syntax is similar to METAPOST.
3. It overcomes the arithmetic limitations inherent in METAPOST.
4. Both the affine transformations and the graphics pipeline are implemented through 4×4 matrices.
5. Its author, Laurence D. Finston, is actively improving and maintaining the package. His plan includes, among many other ideas, the development of an input routine (to allow interactive use) and the implementation of three-dimensional `path`s via NURBS.

Given the possible computational efficiency of this approach, one can foresee a system that merges the METAPOST language with the capabilities of standard ray-tracing software.

3 `3d/3dgeom`

This was the first documented extension of METAPOST into the third dimension—and also into the fourth dimension (time). Denis B. Roegel created, back in 1997, the `3d` package to produce animations of polyhedra. In 2003 he added the `3dgeom` "module" which is focused on space geometry. It remains the least computationally intensive package of those presented here.

1. Each component of a point or a vector is stored in a different `numeric` array. This eases control of a stack of points. Points are used to define planar polygons (faces of polyhedra) and the polygons are used to define *convex* polyhedra.
2. When defining a polygon, a sequence of points must be provided such that advancing on the sequence is the same as rotating clockwise on the polygon, when the polygon is visible. This means that, when a polyhedron is to be drawn, the selection of polygons to be drawn is very easy: only those whose points rotate clockwise (the visible ones). Hidden line removal is thus achieved without sorting the polygons.
3. Points can also be used to define other points according to rules that are common in the geometry of polyhedra or according to operations involving straight lines and/or planes and/or angles.

4. The author plans to release an updated version with the ability to graph parametric lines and surfaces.

4 m3D

Anthony Phan developed this very interesting package but has not yet written its documentation. Certainly, this is, of all four, the package that can produce the most complex and beautiful diagrams. It achieves this using, almost exclusively, four-sided polygons.

1. Complex objects can be defined and composed (see figure 1). For example, one of its many predefined objects is the fractal known as the "Menger Sponge".
2. It can render revolution surfaces defined from a standard METAPOST `path` (see figure 2).
3. Objects or groups of polygons can be sorted and drawn as if reflecting light from a punctual source and/or disappearing in a foggy environment.

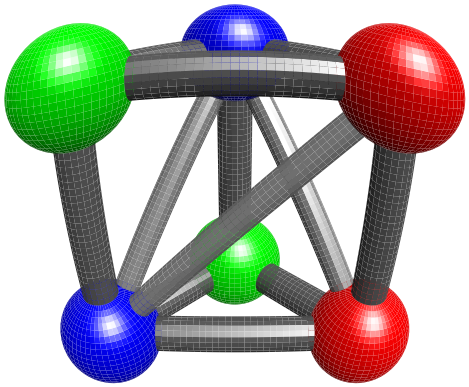

Figure 1: A diagram produced by m3D showing a single object, composed of spheres and cylindrical connections, under a spherical perspective.

5 FEATPOST

Geared towards the production of physics diagrams, FEATPOST sacrifices programming style and computational efficiency for a large feature set.

1. Besides the usual parallel and central perspectives it can make a sort of "spherical distortion" as if a diagram is observed through a fish-eye lens[1]. This kind of perspective is advantageous for animations as it allows the point of view to be inside or among the diagram objects. When using the central perspective, points that are as distant from the projection plane as the point of view get projected at infinity, and MetaPost overflows and crashes. The spherical projection is always finite.

[1] Also possible with m3D.

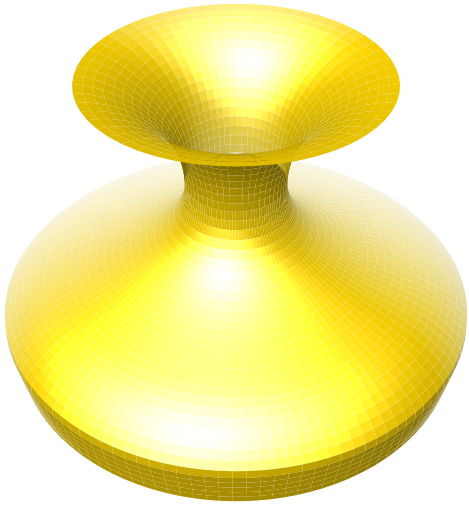

Figure 2: A diagram produced by m3D showing a revolution surface under a central perspective.

2. It can mark and measure angles in space.
3. It can produce shadows of some objects (see figure 9). Shadows are calculated in much the same way as perspectives. The perspective projection, from 3D into 2D, is a calculation of the intersection of a straight line and a plane. A shadow is also a projection from 3D into 2D, only the line and the plane are different. The shadow must be projected onto the paper page before the object that creates the shadow. Shadows are drawn after two projections, objects are drawn after one projection and after their shadows.
4. It can correctly draw intersecting polygons (see figure 12).
5. It knows how to perform hidden line removal on some curved surface objects. Imagine a solid cylinder. Now consider the part of the cylinder's base that is the farthest away. You only see a part of its edge. In order to draw that part, it is necessary to know the two points at which the edge becomes hidden. FEATPOST calculates this. Note that the edge is a circle, a curved line. FEATPOST does not use polygons to hide lines on some curved surface objects.
6. Supported objects include: dots, vectors, angles, ropes, circles, ellipses, cones, cylinders, globes, other curved surface objects, polygons, cuboids, polyhedra, functional and parametric

surface plots, direction fields, field lines and trajectories in conservative force fields.

Many of the drawable objects are not made of polygons, but rather of two-dimensional paths. FEATPOST does not attempt to draw surfaces of these objects, only their edges. This is partly because of the use of intrinsic METAPOST functions and partly because it eases the production of diagrams that combine space and planar (on paper) objects.

One of the intrinsic METAPOST functions that became fundamental for FEATPOST is the composition `makepath makepen`. As this converts a `path` into its convex form, it very much simplifies the determination of some edges.

Another important aspect of the problem is hidden line removal. Hidden line removal of a group of polygons can, in some cases, be performed by drawing the polygons by decreasing order of distance to the point of view. FEATPOST generally uses the Shell sorting method, although when the polygons are just the faces of one cuboid FEATPOST has a small specific trick. There is also a specific method for hidden line removal on cylinders and another for other curved surface objects.

5.1 Examples

Some of the FEATPOST macros are presented here. Detailed information is available at

- `http://matagalatlante.org/nobre/featpost/doc/macroMan.html`
- `http://www.ctan.org/tex-archive/graphics/metapost/macros/featpost/`

Each perspective depends on the point of view. FEATPOST uses the global variable `f`, of type `color`, to store the (X, Y, Z) coordinates of the point of view. Also important is the aim of view (global variable `viewcentr`). Both together define the line of view.

The perspective consists of a projection from space coordinates into planar (u, v) coordinates on the projection plane. FEATPOST uses a projection plane that is perpendicular to the line of view and contains the `viewcentr`. Furthermore, one of the projection plane axes is horizontal and the other is on the intersection of a vertical plane with the projection plane. "Horizontal" means parallel to the XY plane.

One consequence of this setup is that `f` and `viewcentr` must not be on the same vertical line (as long as the author avoids solving this problem, at least!). The three kinds of projection known to FEATPOST are schematized in figures 3, 4 and 5. The macro that actually does the projection is, in all cases, `rp`.

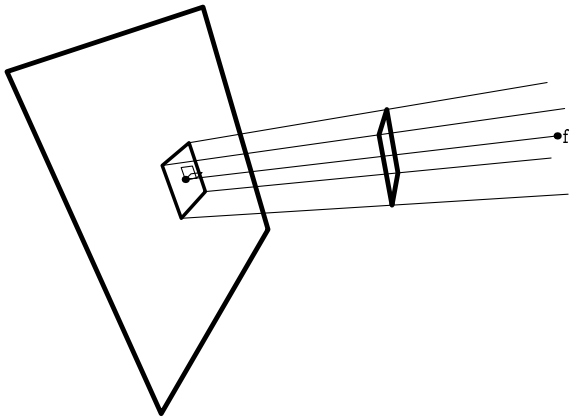

Figure 3: Parallel projection.

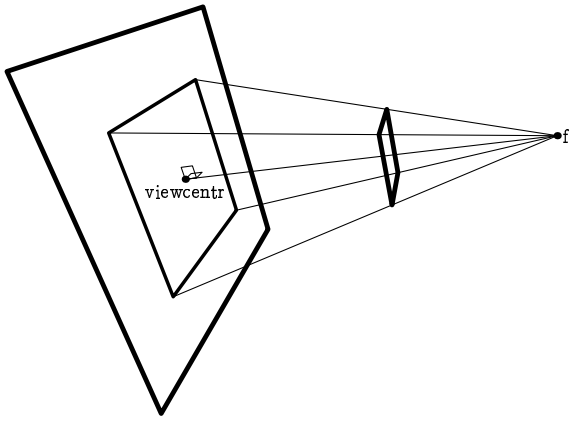

Figure 4: Central projection.

Physics problems often require defining angles, and diagrams are needed to visualize their meanings. The `angline` and `squareangline` macros (see figure 6 and the code below) support this.

```
f := (5,3.5,1);
beginfig(2);
  cartaxes(1,1,1);
  color va, vb, vc, vd;
  va = (0.29,0.7,1.0);
  vb = (X(va),Y(va),0);
  vc = N((-Y(va),X(va),0));
  vd = (0,Y(vc),0);
  drawarrow rp(black)--rp(va);
  draw rp(black)--rp(vb)--
               rp(va) dashed evenly;
  draw rp(vc)--rp(vd) dashed evenly;
  drawarrow rp(black)--rp(vc);
  squareangline( va, vc, black, 0.15 );
```

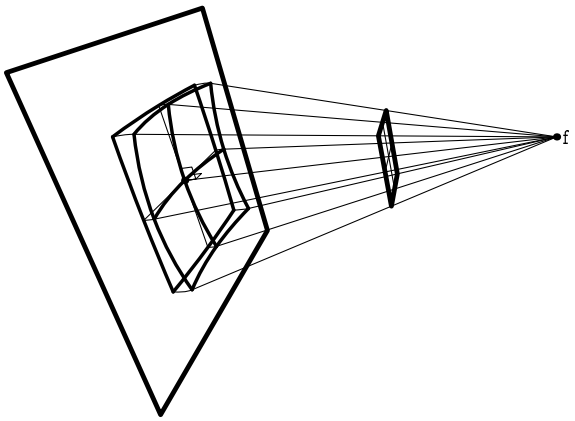

Figure 5: Spherical projection. The spherical projection is the composition of two operations: (i) there is a projection onto a sphere and (ii) the sphere is plaited onto the projection plane.

```
  angline(va,red,black,0.75,
        decimal getangle(va,red),lft);
endfig;
```

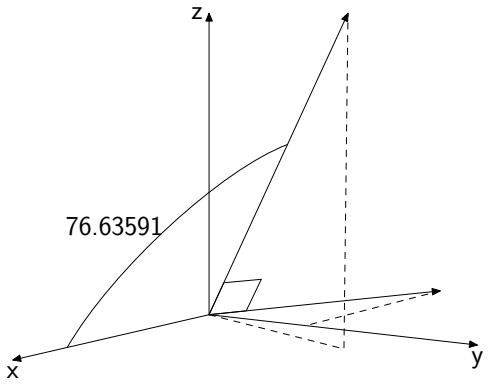

Figure 6: FEATPOST diagram using `angline`.

Visualizing parametric lines is another need of physicists. When two lines cross, one should be able to see which line is in front of the other. The macro `emptyline` can help here (see figure 7 and the code below).

```
f := (2,4,1.8);
def theline( expr TheVal ) =
  begingroup
    numeric cred, cgre, cblu, param;
    param = TheVal*(6*360);
    cred = -0.3*cosd( param );
    cblu = 0.3*sind( param );
    cgre = param/850;
    ( (cred,cgre,cblu) )
  endgroup
enddef;
beginfig(1);
  numeric axsize, zaxpos, zaxlen;
  color xbeg, xend, ybeg,
                  yend, zbeg, zend;
  axsize = 0.85;
  zaxpos = 0.55;
  zaxlen = 2.1;
  pickup pencircle scaled 1.5pt;
  xbeg = (axsize,0,0);
  xend = (-axsize,0,0);
  ybeg = (0,0,-axsize);
  yend = (0,0,axsize);
  zbeg = (zaxpos,-zaxpos,0);
  zend = (zaxpos,zaxlen,0);
  drawarrow rp( xbeg )--rp( xend );
  drawarrow rp( ybeg )--rp( yend );
  defaultscale := 1.95;
  label.rt( "A", rp( xend ) );
  label.lft( "B", rp( yend ) );
  emptyline(false,1,black,
     0.5black,1000,0.82,2,theline);
  drawarrow rp( zbeg )--rp( zend );
  label.bot( "C", rp( zend ) );
endfig;
```

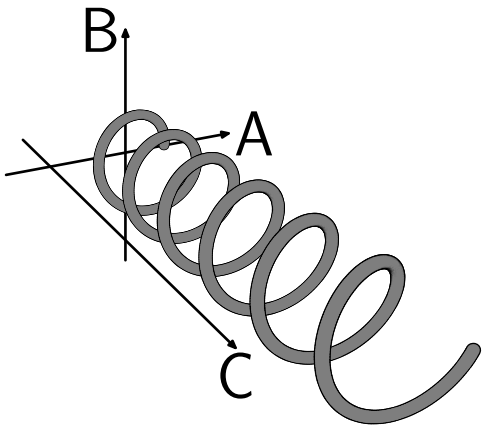

Figure 7: FEATPOST diagram using `emptyline`.

Cuboids and labels are always needed. The macros `kindofcube` and `labelinspace` fulfill this need (see figure 8 and the code below). The macro `labelinspace` does not project labels from 3D into 2D. It only `Transform`s the label in the same way as its bounding box, that is, the same way as two perpendicular sides of its bounding box. This is only exact for parallel perspectives.

```
f := (2,1,0.5);
ParallelProj := true;
verbatimtex
```

```
\documentclass{article}
\usepackage{beton,concmath,ccfonts}
\begin{document}
etex
beginfig(1);
  kindofcube(false,true,(0,-0.5,0),
                90,0,0,1.2,0.1,0.4);
  kindofcube(false,true,(0,0,0),
                 0,0,0,0.5,0.1,0.8);
  labelinspace(false,(0.45,0.1,0.65),
               (-0.4,0,0),(0,0,0.1),
          btex
            \framebox{\textsc{Label}}
          etex);
endfig;
verbatimtex \end{document} etex
```

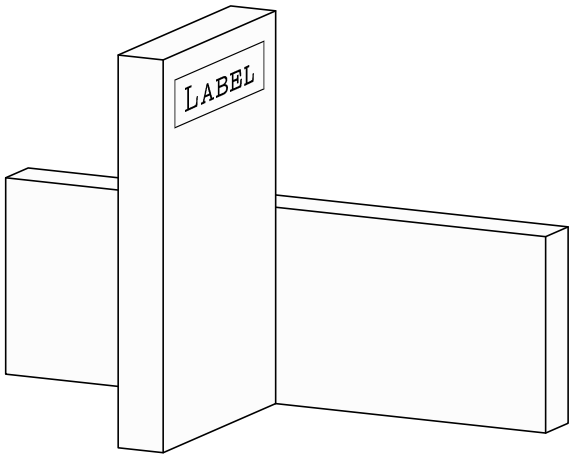

Figure 8: FEATPOST diagram using the macros `kindofcube` and `labelinspace`.

Some curved surface solid objects can be drawn with FEATPOST. Among them are cones (`verygoodcone`), cylinders (`rigorousdisc`) and globes (`tropicalglobe`). These can also cast their shadows on a horizontal plane (see figure 9 and the code below). The production of shadows involves the global variables `LightSource`, `ShadowOn` and `HoriZon`.

```
f := (13,6,4.5);   ShadowOn := true;
LightSource := 10*(4,-3,6);
beginfig(3);
  numeric reflen, frac, coordg;
  numeric fws, NumLines;
  path ella, ellb;
  color axe, cubevertex, conecenter,
     conevertex, allellaxe, ellaaxe,
                          pca, pcb;
  frac := 0.5;          wang := 60;
  axe := (0,cosd(90-wang),
            sind(90-wang));
  fws := 4;      reflen := 0.35*fws;
  coordg := frac*fws;
  NumLines := 45;
  HoriZon := -0.5*fws;
  setthestage(0.5*NumLines,3.3*fws);
  cubevertex = (0.3*fws,-0.5*fws,0);
  tropicalglobe( 7, cubevertex,
                    0.5*fws, axe );
  allellaxe:=reflen*(0.707,0.707,0);
  ellaaxe:= reflen*( 0.5, -0.5, 0 );
  pcb := ( -coordg, coordg, 0 );
  rigorousdisc( 0, true, pcb,
                0.5*fws, -ellaaxe );
  conecenter =
       ( coordg, coordg, -0.5*fws );
  conevertex = conecenter +
                  ( 0, 0, 0.9*fws );
  verygoodcone(false,conecenter,
            blue,reflen,conevertex);
endfig;
```

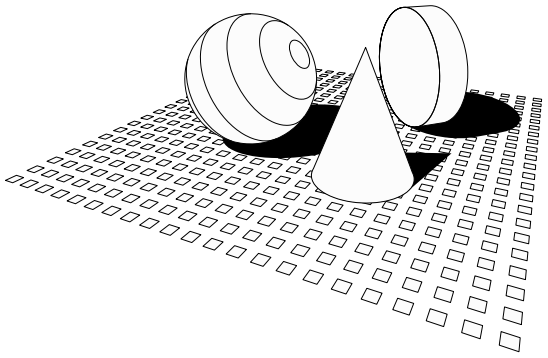

Figure 9: FEATPOST diagram using the macros `rigorousdisc`, `verygoodcone`, `tropicalglobe` and `setthestage`.

Another very common need is the plotting of functions, usually satisfied by software such as Gnuplot (`http://www.gnuplot.info/`). Nevertheless, there are always new plots to draw. One kind of FEATPOST plot that just became possible is the "triangular grid triangular domain surface" (see figure 10 and this code):

```
f := 16*(4,1,1);
LightSource := 10*(4,-3,4);
def zsu( expr xc, yc ) =
  cosd(xc*57)*cosd(yc*57)+
    4*mexp(-(xc**2+yc**2)*6.4) enddef;
beginfig(1);
  hexagonaltrimesh(false,52,15,zsu);
endfig;
```

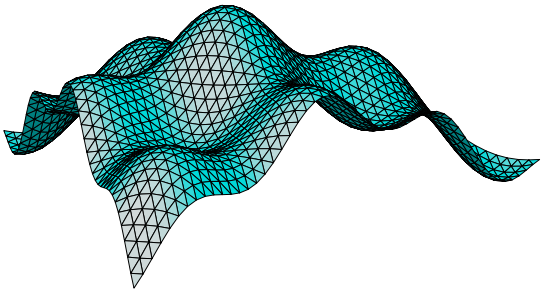

Figure 10: FEATPOST surface plot using the macro `hexagonaltrimesh`.

One feature that merges 2D and 3D involves what might be called "fat sticks". A fat stick resembles the Teflon magnets used to mix chemicals. They have volume but can be drawn like a small straight line segment stroked with a `pencircle`. Fat sticks may be used to represent direction fields (unitary vector fields without arrows). See figure 11 (the code is skipped from now on).

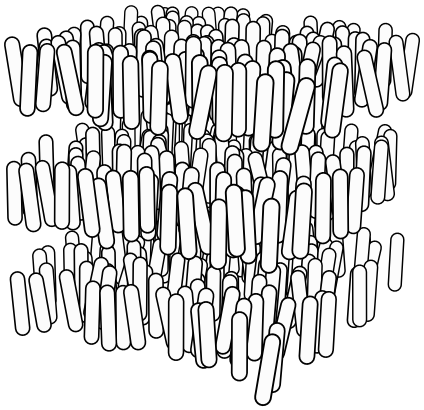

Figure 11: FEATPOST direction field macro `director_invisible` was used to produce this representation of the molecular structure of a Smectic A liquid crystal.

Finally, it is important to remember that some capabilities of FEATPOST, although usable, may be considered "buggy" or only partially implemented. These include the calculation of intersections among polygons, as in figure 12, and the drawing of toruses, as in figure 13. These two figures show "usable" situations but their code is skipped.

FEATPOST has many macros: some are specifically for physics diagrams, others may be useful for general purposes, some do not fit in this article and,

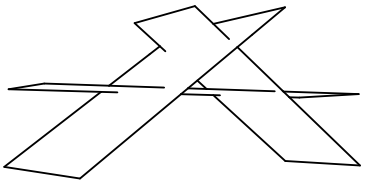

Figure 12: Intersecting polygons drawn with the macro `sharpraytrace`.

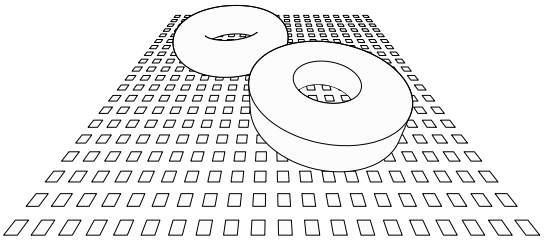

Figure 13: Final FEATPOST example containing a `smoothtorus` and a `rigorousdisc` with a hole. These macros may fail for some view points.

sadly, some are not anywhere documented. For instance, the tools for producing animations are not yet documented. (These tools are completely external to TEX: the control of an animation is done with a Python script, and Ghostscript and `netpbm` are used to produce MPEG videos.)

In summary, the collection of three-dimensional METAPOST software, such as the four reviewed packages, is large and growing in many independent directions. It constitutes an excellent resource for those desiring to produce good diagrams.

6 Acknowledgements

Many people have contributed to make FEATPOST what it is today. Perhaps it would have never come into being without the early intervention of Jorge Bárrios, providing access to his father's computer. Another fundamental moment happened when José Esteves first spoke about METAPOST.

More recently, the very accurate criticism of Cristian Barbarosie has significantly contributed to the improvement of these macros. Jens Schwaiger contributed new macros. Pedro Sebastião, João Dinis and Gonçalo Morais proposed challenging new features. The authors of the other packages graciously reviewed the paper, and Karl Berry actually entered new text into this document. They all have my deep thanks.