FEATPOST manual

L. Nobre G.

0.8.8

Abstract

FEATPOST is an extension of the METAPOST language that has a fairly large set of features to facilitate the production of schematic diagrams, both in three dimensions (3D) and in two dimensions (2D).

These schematic diagrams are vectorial and focus on the representation of edges (unlike ray-traced raster images that focus on surfaces).

Getting started

```
input featpost3Dplus2D;
```

First taste of FEATPOST

Each perspective depends on the point of view. FEATPOST uses the global variable `f`, of `color` type, to store the (X, Y, Z) space coordinates of the point of view. Also important is the aim of view (global variable `viewcentr`). This pair of points defines 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 perpendicular and on 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. The three kinds of projection known to FEATPOST are schematized in figures 1, 2 and 3. The macro that actually does the projection is, in all cases, `rp`.

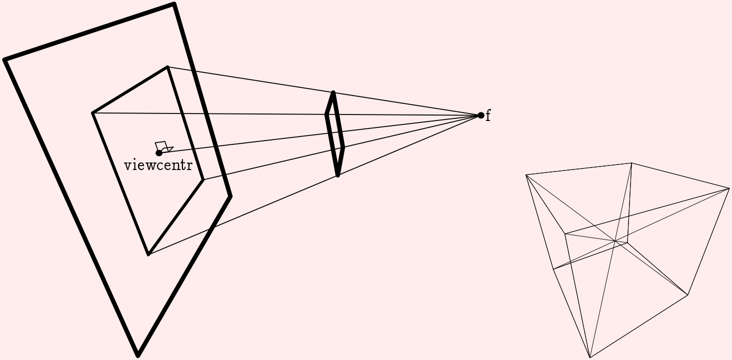


Figure: Central projection (default).

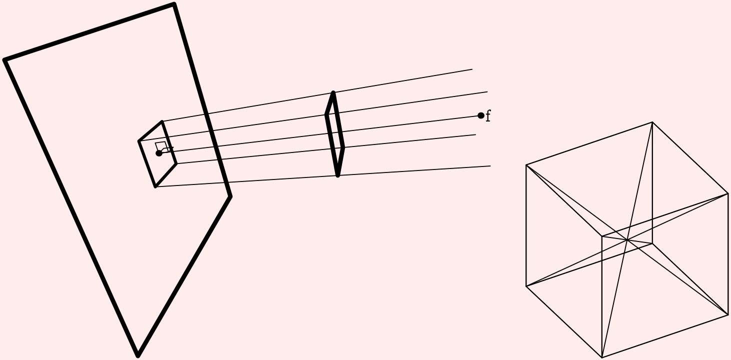


Figure: Parallel projection.

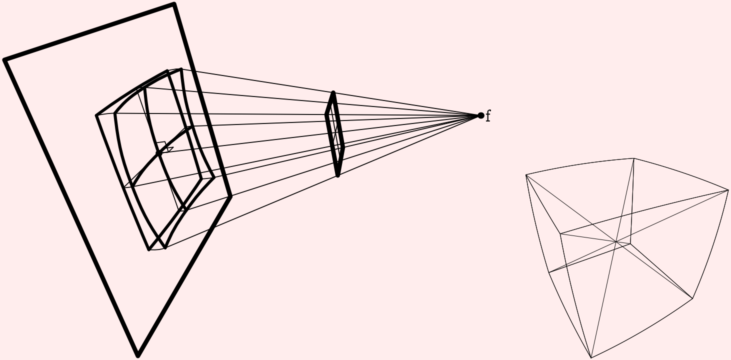


Figure: Spherical projection. The spherical projection is the composition of two operations: (i) there is a projection onto a sphere and (ii) the sphere is plaited onto the projection plane.

Angles

Some problems often require defining angles, and diagrams are needed to visualize their meanings. The `angline` and `squareangline` macros support this (see figure 4).

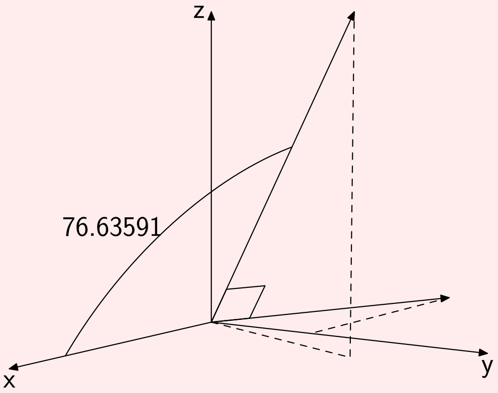


Figure: Example that uses `cartaxes`, `squareangline`, `angline` and `getangle`.

Parametric lines

Visualizing parametric lines is another need. 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 5).

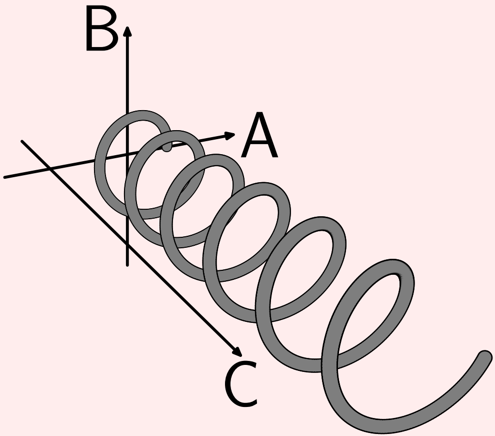


Figure: FEATPOST diagram using `emptyline`.

Curved solids

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 6). The production of shadows involves the global variables `LightSource`, `ShadowOn` and `HoriZon`.

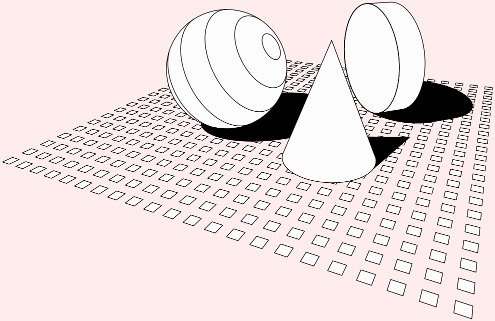


Figure: **FEATPOST** diagram using the macros `rigorousdisc`, `verygoodcone`, `tropicalglobe` and `setthestage`.

Fat sticks

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 big `pencircle`. Fat sticks may be used to represent direction fields (unitary vector fields without arrows). See figure 7.

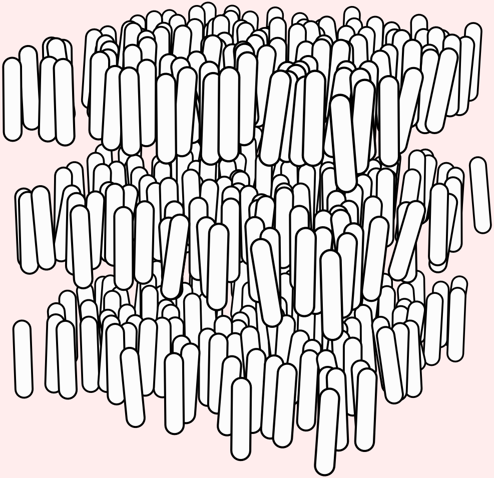


Figure: **FEATPOST** direction field macro `director_invisible` was used to produce this representation of the molecular structure of a Smectic A liquid crystal.

Bugs

Finaly, it is important to remember that some capabilities of FEATPOST, although usable, may be considered “buggy” or only partially implemented. These include the drawing of cylinders with holes, as in figure 8.

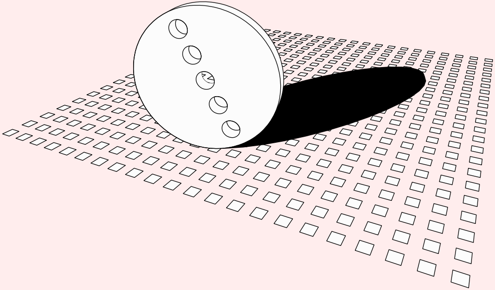


Figure: FEATPOST example containing a `rigorousdisc` with five holes, four of which are fake.

Moving on

It is highly beneficial to be able to understand and cope with METAPOST error messages as FEATPOST has no protection against mistaken inputs. One probable cause of errors is the use of variables with the name of procedures (macros), like

`X, Y, Z, W, N, rp, cb, ps`

All other procedure names have six or more characters.

The user must be aware that METAPOST has a limited arithmetic power and that the author has limited programming skills, which may lead to unperfect 3D figures, very long processing time or shear bugs. It's advisable not to try very complex diagrams at first and it's recommended to keep 3D coordinates near order 1 (default METAPOST units).

All three-dimensional FEATPOST macros are build apon the METAPOST `color` variable type. It looks like this:

```
(red,green,blue)
```

Its components may, nevertheless, be arbtitrary numbers, like:

```
(X,Y,Z)
```

So, the `color` type is adequate to define not only colors but also 3D points and vectors.

Hello world

One very minimalistic example program could be:

```
input featpost3Dplus2D; beginfig(1); cartaxes(1,1,1);
endfig; end.
```

where `cartaxes` is a FEATPOST macro that produces the Cartesian reference frame with axis labels.

The main variable of any three-dimensional figure is the point of view. FEATPOST uses the variable `f` as the point of view. `Spread` is another global variable that controls the size of the projection.

Therefore the minimalistic program above should be, at least, like this:

```
input featpost3Dplus2D; f:=(6,1,3); Spread:=40; beginfi
cartaxes(1,1,1); endfig; end;
```

Why FEATPOST?

FEATPOST is good enough to produce scientific diagrams:

- Figure 1 of *Phys. Rev. E*, **60**, 2985-2989 (1999).
- Figures 4, 6 and 8 of *Eur. Phys. J. E*, **2**, 351-358 (2000).
- Figures 8 and 12 of *Eur. Phys. J. E*, **20**, 55-61 (2006).

From 3D to 2D

The most important macro is `rp` that converts 3D points to two-dimensional (2D) rigorous, orthogonal or fish-eye projections. To draw a line in 3D-space try

```
    draw rp(a)--rp(b);
```

where `a` and `b` are points in space (of `color` type).

“Straight lines”

But if you’re going for fish-eye it’s better to

```
    draw pathofstraightline(a,b);
```

If you don’t know, leave it as

```
    drawsegment(a,b);
```

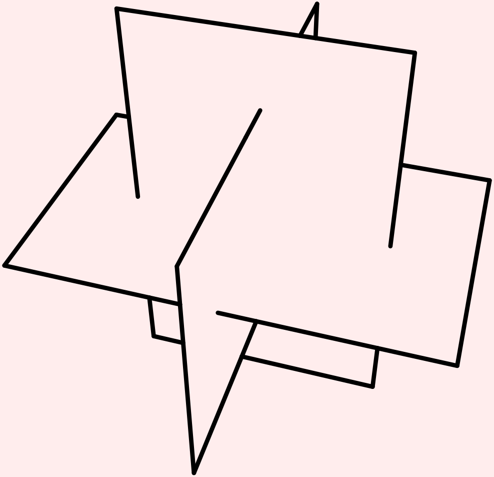


Figure: Intersecting polygons drawn with the macro `sharpraytrace`.

Coming back to 3D from 2D

It is possible to do an "automatic perspective tuning" with the aid of macro `photoreverse`. Please, refer both to example `photoreverse.mp` (see figure 10) and to the following web page: FeatPost Deeper Technicalities.

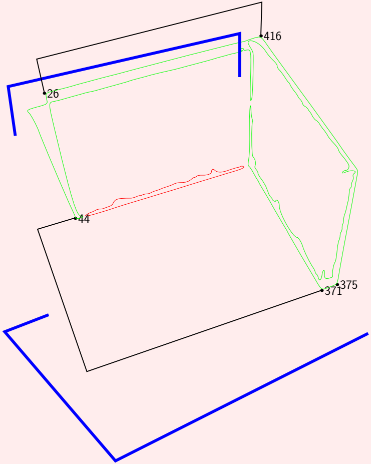


Figure: Example that uses `photoreverse`. It may not work when vertical lines are not vertical in average on the photo.

Coming back to 3D from 1D

Using almost the same algorithm as `photoreverse`, the macro `improvertex` allows one to approximate a point in 3D-space with given distances d from three other points (an initial guess $\vec{i}$ is required).

$$\texttt{point := improvertex(} \; \vec{a},\, d_a,\, \vec{b},\, d_b,\, \vec{c},\, d_c,\, \vec{i} \; \texttt{);}$$

`ultraimprovertex`

Approximating a point in 3D-space with given distances from three other points is the same as calculating the intersection of three spheres. And the method to do that is the same as the method to calculate the intersection of a plane, a cylinder and a spheroid (see figure 11).

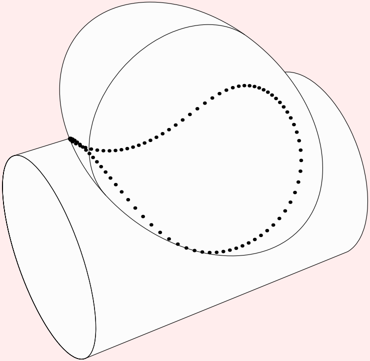


Figure: Example that uses `ultraimprovertex`.

Scalar function minimization

Macro `minimizestep` is a minimization routine for scalar functions like $y = f(x)$ where an initial triplet (x_1, x_2, x_3) with $x_1 < x_2 < x_3$ is given as a parabolic squeleton that provides a way to search for the smallest value of y (if iterated).

`point := minimizestep(` $\vec{x}$ `)(` f `);`

The features

3D dots, vectors, flat and curved arrows, angles, parametric lines, circles and ellipses, cuboids, cones, cylinders, cylindric holes, parts of cylindrical surfaces, spheres and spheroids, globes, hemispheres, tora, elliptical frusta, revolution paraboloids, polygons, polyhedra, functional and parametric surfaces, direction fields, field lines and trajectories in vector fields (differential equations), schematic automobiles, schematic electric charges, automatic perspective tuning, 2D representation of ropes, reference horizontal surfaces, hexagonal plots, schematic 2D springs, zig–zag lines, irregular circles, selective intersection of two circles, 2D detection of tangency, paths for laser cutting machines, minimization of scalar functions, intersection of 2D areas, intersection point of three spheres, intersection point of a plane, a cylinder and a spheroid, intersection of a straight line and a spheroid, intersection of a straight line and a torus.
And much more.

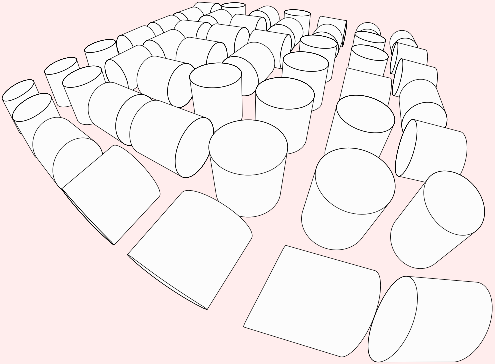


Figure: Figure that uses `SphericalDistortion:=true` and `rigorousdisc`.

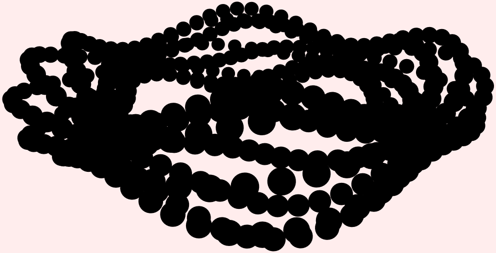


Figure: Figure that uses `signalvertex`.

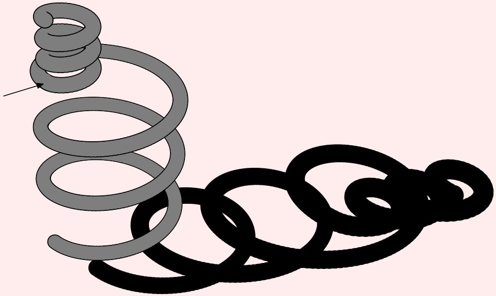


Figure: Figure that uses `emptyline`. The junction point of two different lines is indicated by an arrow.

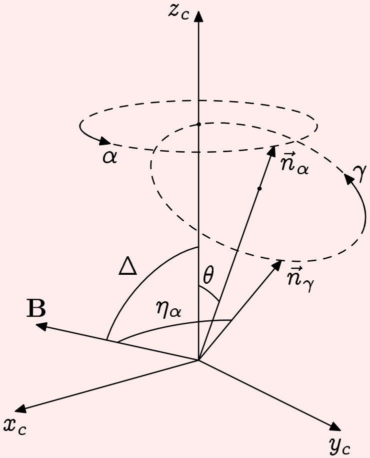


Figure: Figure that uses `anglinen` and `rigorouscircle`.

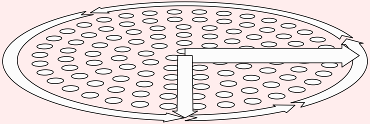


Figure: Figure that uses `tdarrow` and `tdcircarrow`.

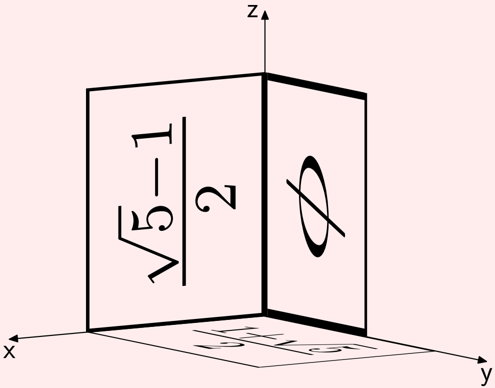


Figure: Example that uses `labelinspace`.

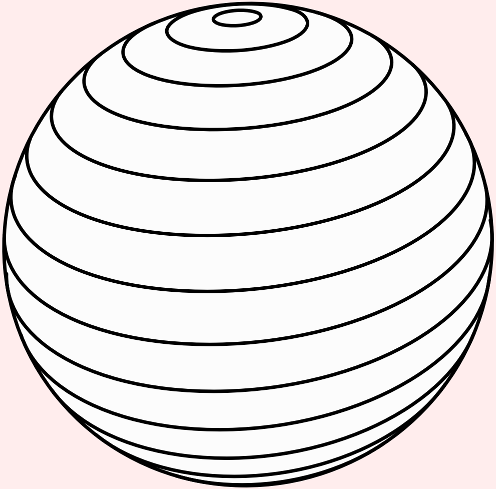


Figure: Figure that uses `tropicalglobe`.

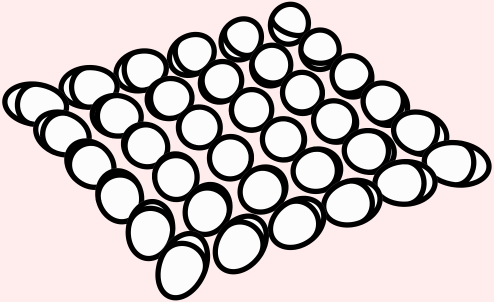


Figure: Figure that uses `spheroid`.

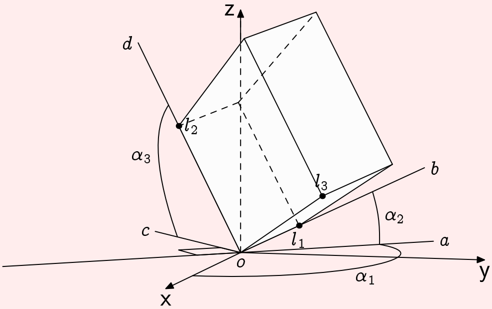


Figure: Figure that uses and explains `kindofcube`. Note that the three indicated angles may be used as arguments of `eulerrotation`.

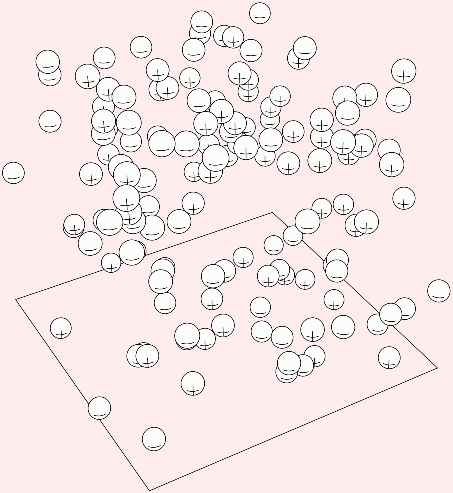


Figure: Figure that uses `positivecharge`, `getready` and `doitnow`.

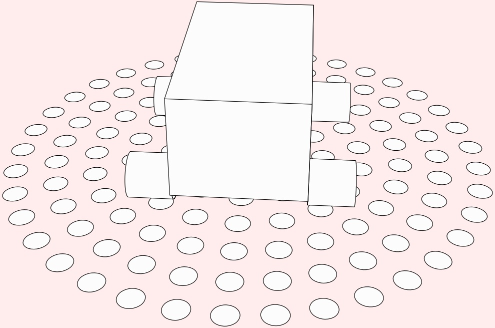


Figure: Figure that uses setthearena and simplecar.

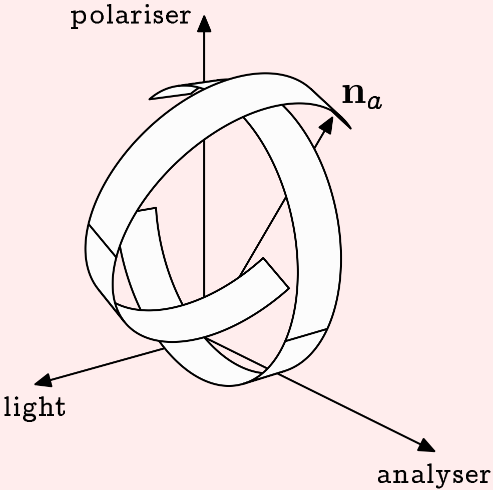


Figure: Figure that uses banana.

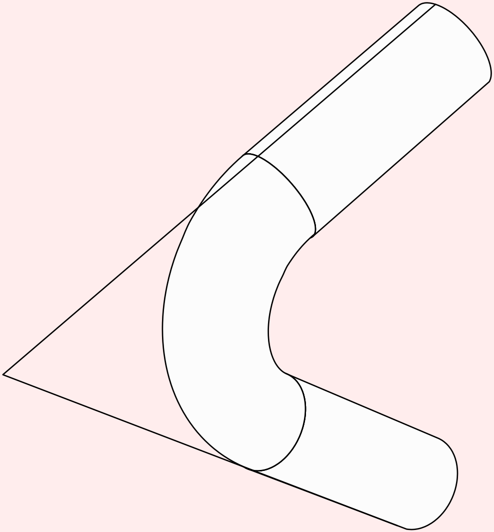


Figure: Figure that uses `quartertorus`.

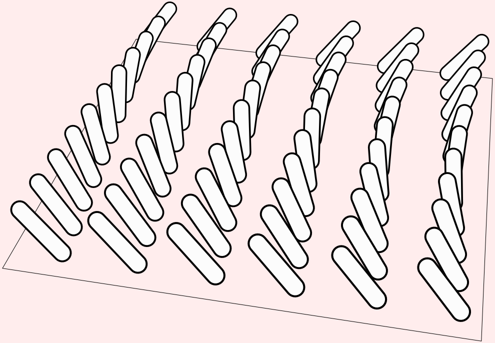


Figure: Figure that uses `director_invisible` and `generatedirline`.

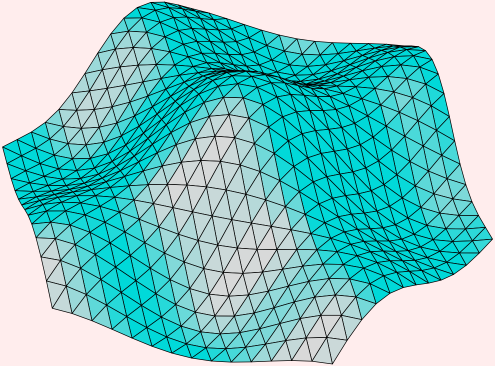


Figure: Figure that uses `hexagonaltrimesh`.

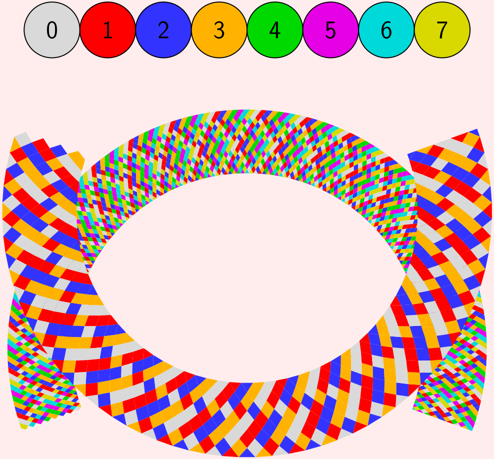
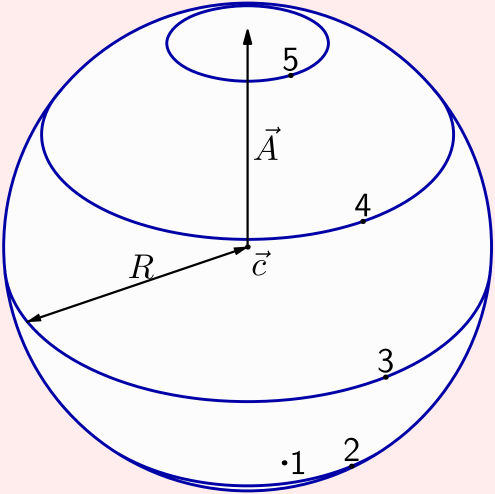


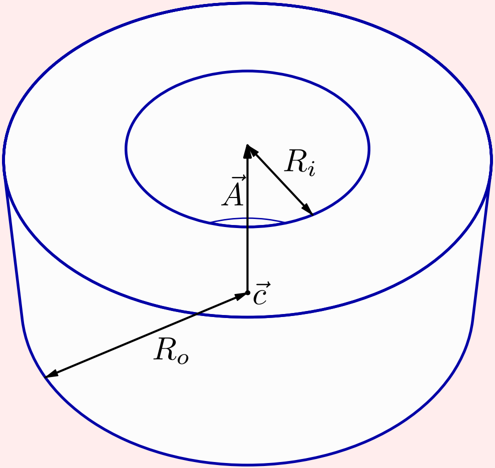
Figure: Figure that uses `ropepattern`.

`tropicalglobe(` N, $\vec{c}$, R, $\vec{A}$ `)`



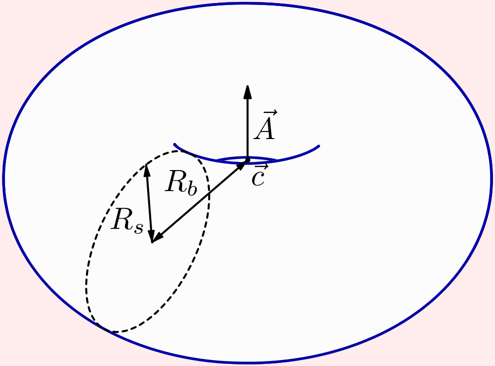
```
tropicalglobe( 5, black, 1, blue );
```

rigorousdisc(R_i, bool, $\vec{c}$, R_o, $\vec{A}$)



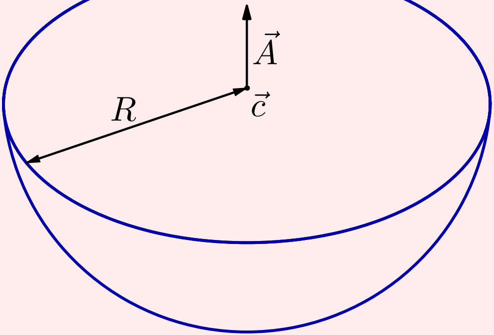
```
rigorousdisc( 0.5, true, black, 1, 0.85blue );
```

`smoothtorus(` $\vec{c}$, $\vec{A}$, R_b, R_s `)`



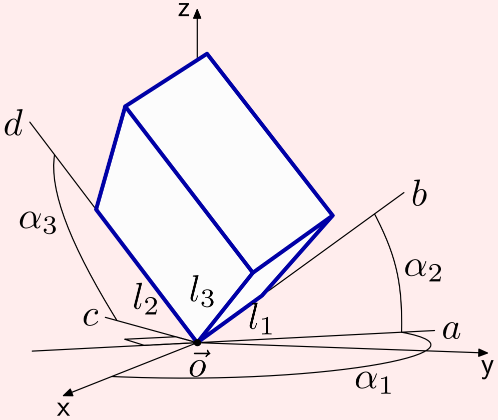
```
smoothtorus( black, blue, 0.7, 0.4 );
```

spatialhalfsfear($\vec{c}$, $\vec{A}$, R)



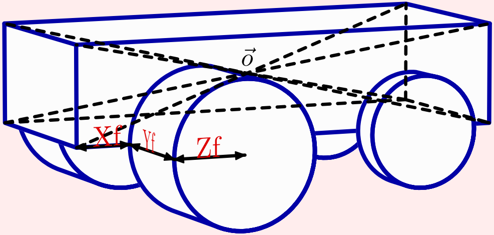
```
spatialhalfsfear( black, blue, 1 );
```

`kindofcube(bool,bool,`$\vec{o}, \alpha_1, \alpha_2, \alpha_3, l_1, l_2, l_3$`)`



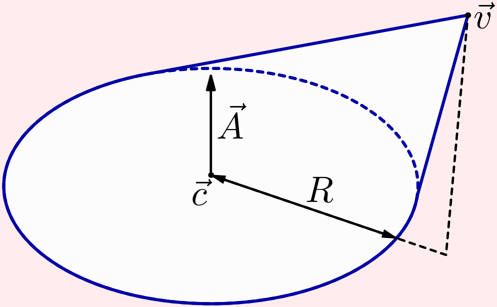
```
kindofcube( false, true, black, 130, 32, 67, 0.3,
0.6, 0.9 );
```

simplecar($\vec{o}$,$(\alpha_1, \alpha_2, \alpha_3)$, (l_1, l_2, l_3), (Xf,Yf,Zf), (Xr,Yr,Zr))



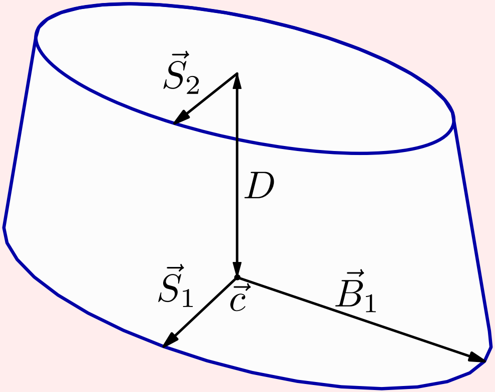
```
simplecar( black, black, (0.8,0.35,0.18),
  (0.1,0.2,0.132), (0.06,0.06,0.1) );
```

verygoodcone(bool, $\vec{c}$, $\vec{A}$, R, $\vec{v}$)



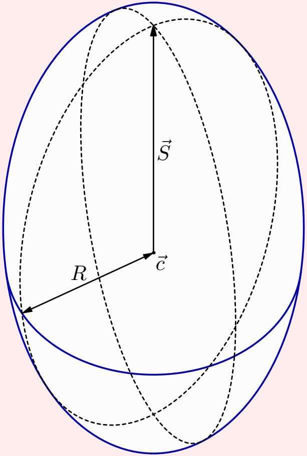
```
verygoodcone( true, black, blue, 0.8, blue+green );
```

`whatisthis(` $\vec{c}$, $\vec{S}_1$, $\vec{B}_1$, D, $||\vec{S}_2||/||\vec{S}_1||$ `)`



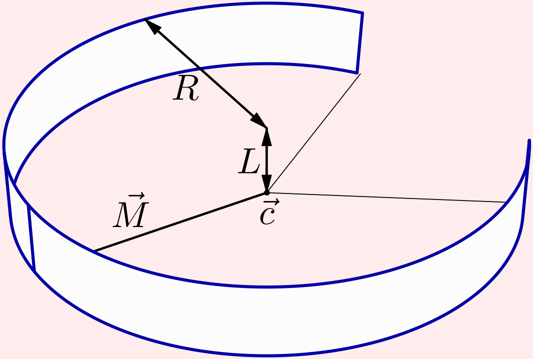
```
whatisthis( black, 0.5red, green, 0.85, 0.8 );
```

spheroid($\vec{c}$, $\vec{S}$, R)



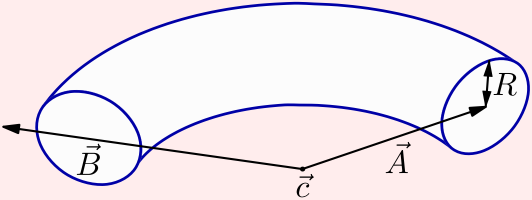
```
spheroid( black, 2*blue, 1 );
```

$\mathsf{banana}(\ \vec{c},\ R,\ (\alpha_M, \beta_M, \gamma_M),\ L,\ \theta\ )$



```
banana( black, 1, black, 0.3, 145 );
```

`quartertorus(` $\vec{c}$, $\vec{A}$, $\vec{B}$, R `)`



```
quartertorus( black, -red, red-green, 0.25 );
```

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 in 1986. Another important moment happened when José Esteves first spoke about METAPOST sometime in the late nineties.

Also, the very accurate criticism of Cristian Barbarosie has significantly contributed to fundamental improvements. Jens Schwaiger contributed new macros. Pedro Sebastião, João Dinis and Gonçalo Morais proposed challenging new features.