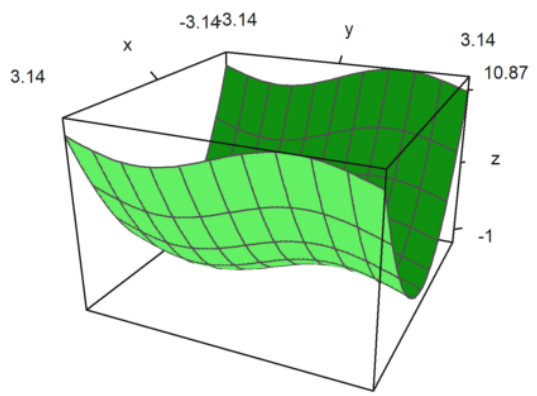
Menggambar Plot 3D dengan EMT

This is an introduction to 3D plots in Euler. We need a 3D plot to visualize a function of two variables.
Euler draws such functions using a sorting algorithm to hide parts in the background. In general, Euler uses a central projection. The default is from the positive x-y quadrant towards the origin x=y=z=0, but angle=0° looks from into the direction of the y-axis. The view angle and height can be changed.
Euler can plot
- surfaces with shading and level lines or level ranges,
- clouds of points,
- parametric curves,
- implicit surfaces.

A 3D plot of a function uses plot3d. The easiest way is to plot an expression in x and y. The parameter r set the range of the plot around (0,0).

```
>aspect(1.5); plot3d("x^2+sin(y)",r=pi):
```



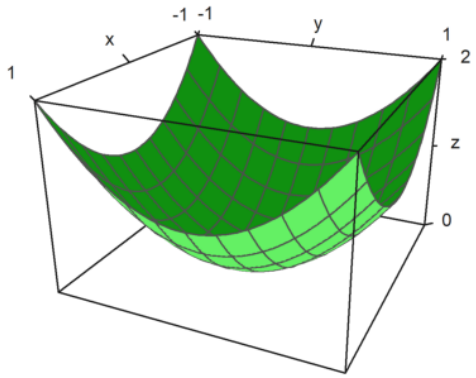
Functions of two Variables

For the graph of a function, use
- a simple expression in x and y,
- the name of a function of two variablesl
- or data matrices.

The default is a filled wire grid with different colors on both sides. Note that the default number of grid intervals is 10, but the plot uses the default number of 40x40 rectangles to construct the surface. This can be changed.
- n=40, n=[40,40]: number of grid lines in each direction
- grid=10, grid=[10,10]: number of grid lines in each direction.

We use the default n=40 and grid=10.

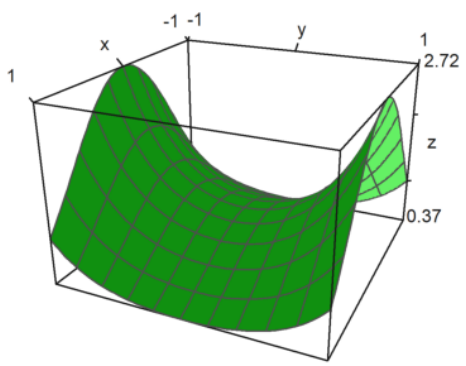
```
>plot3d("x^2+y^2"):
```



User interaction is possible with the >user parameter. The user can press the following keys.
- left,right,up,down: turn the viewing angle
- +,-: zoom in or out
- a: produce an anaglyph (see below)
- l: toggle turning the light source (see below)
- space: reset to default
- return: end interaction

```
>plot3d("exp(-x^2+y^2)",>user, ...
>  title="Turn with the vector keys (press return to finish)":
```

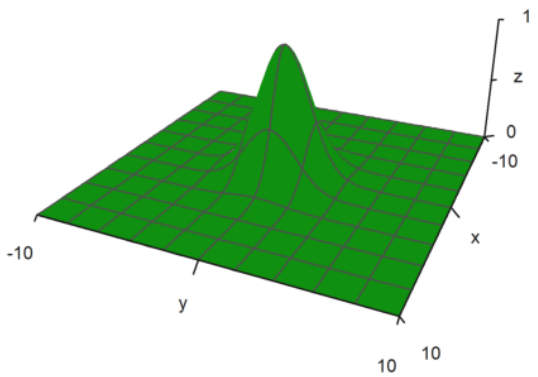
Turn with the vector keys (press return to finish)



The plot range for functions can be specified with
- a,b: the x-range
- c,d: the y-range
- r: a symmetric square around (0,0).
- n: number of subintervals for the plot.

There are some parameters to scale the function or change the look of the graph.
fscale: scales to function values (default is <fscale).
scale: number or 1x2 vector to scale into x- and y-direction.
frame: type of frame (default 1).

```
>plot3d("exp(-(x^2+y^2)/5)",r=10,n=80,fscale=4,scale=1.2,frame=3):
```



The view can be changed in many different ways.
- distance: the viewing distance to the plot.
- zoom: the zoom value.
- angle: the angle to the negative y-axis in radians.
- height: the height of the view in radians.

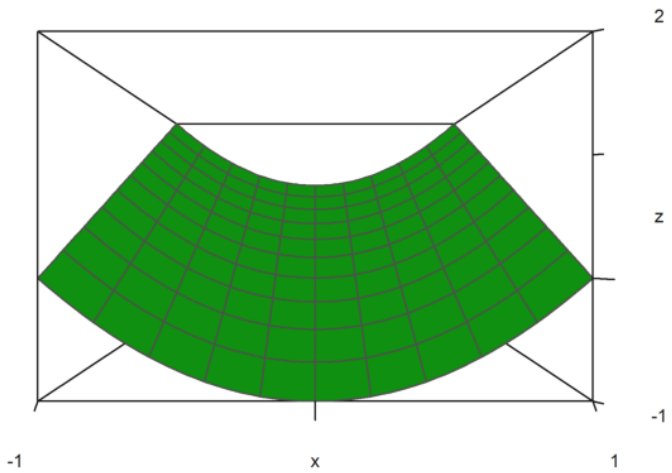
The default values can be inspected or changed with the function view(). It returns the parameters in the order above.

```
>view
```

```
  [5,  2.6,  2,  0.4]
```

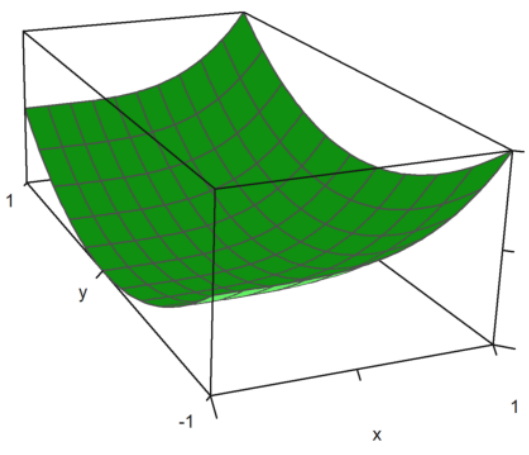
A closer distance needs less zoom. The effect is more like a wide angle lens.
In the following example, angle=0 and height=0 look from the negative y-axis. The axis labels for y are hidden in this case.

```
>plot3d("x^2+y",distance=3,zoom=2,angle=0,height=0):
```



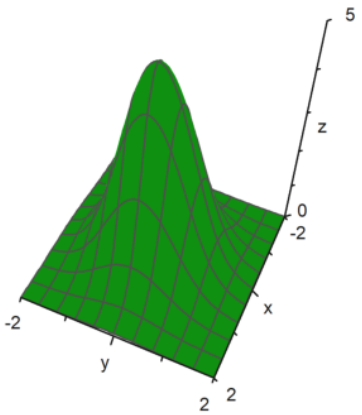
The plot looks always to the center of the plot cube. You can move the center with the center parameter.

```
>plot3d("x^4+y^2",a=0,b=1,c=-1,d=1,angle=-20°,height=20°, ...
>  center=[0.4,0,0],zoom=5):
```



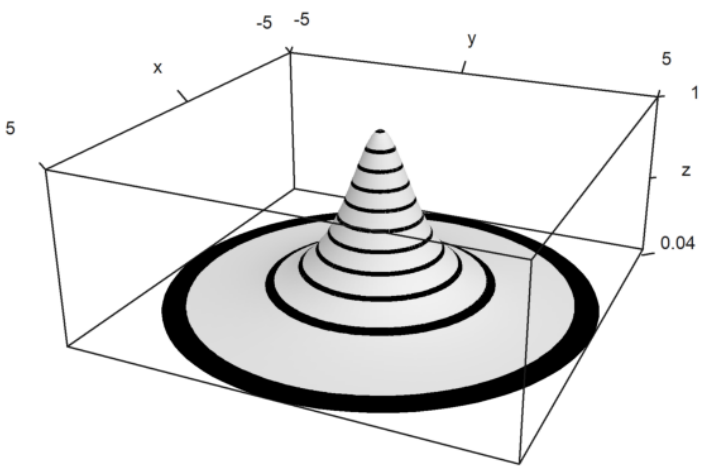
The plot is scaled to fit into a unit cube for viewing. So there is no need to change the distance or zoom depending on the size of the plot. The labels refer to the actual size, however.
If you turn this off with scale=false, you need to take care, that the plot still fits into the plotting window, by changing the viewing distance or zoom, and moving the center.

```
>plot3d("5*exp(-x^2-y^2)",r=2,<fscale,<scale,distance=13,height=50°, ...
>  center=[0,0,-2],frame=3):
```

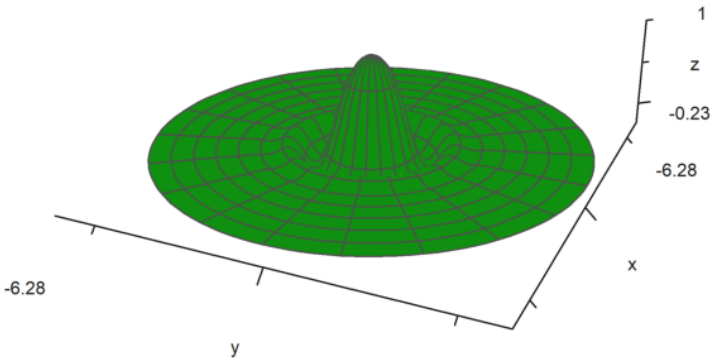


A polar plot is also available. The parameter polar=true draws a polar plot. The function must still be a function of x and y. The parameter "fscale" scales the function with an own scale. Otherwise the function is scaled to fit into a cube.

```
>plot3d("1/(x^2+y^2+1)",r=5,>polar, ...
>fscale=2,>hue,n=100,zoom=4,>contour,color=gray):
```

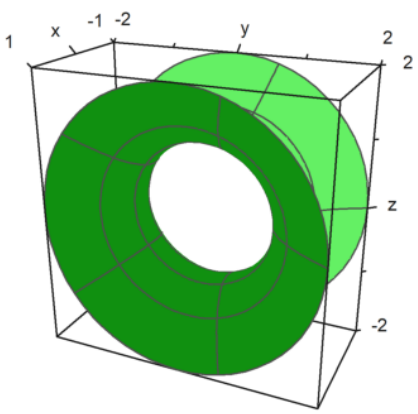



```
>function f(r) := exp(-r/2)*cos(r); ...
>plot3d("f(x^2+y^2)",>polar,scale=[1,1,0.4],r=2pi,frame=3,zoom=4):
```



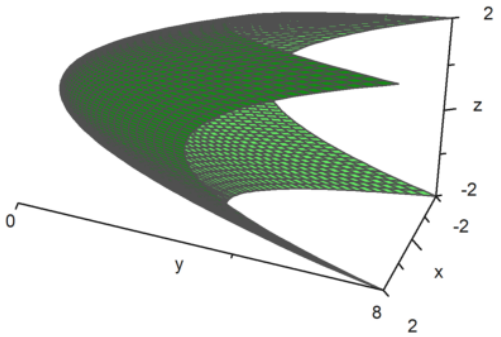
The parameter rotate rotates a function in x around the x-axis.
- rotate=1: Uses the x-axis
- rotate=2: Uses the z-axis

```
>plot3d("x^2+1",a=-1,b=1,rotate=true,grid=5):
```



Here is a plot with three functions.

```
>plot3d("x","x^2+y^2","y",r=2,zoom=3.5,frame=3):
```



Contour Plots

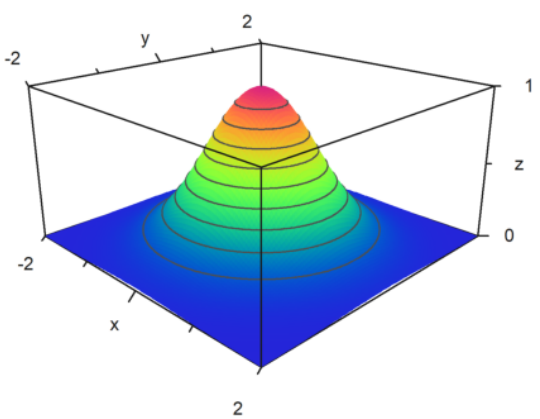
For the plot, Euler adds grid lines. Instead it is possible to use level lines and a one-color hue or a spectral colored hue. Euler can draw the heights of functions on a plot with shading. In all 3D plots Euler can produce red/cyan anaglyphs.

- >hue: Turns on light shading instead of wires.
- >contour: Plots automatic contour lines on a plot.
- level=... (or levels): A vector of values for the contour lines.

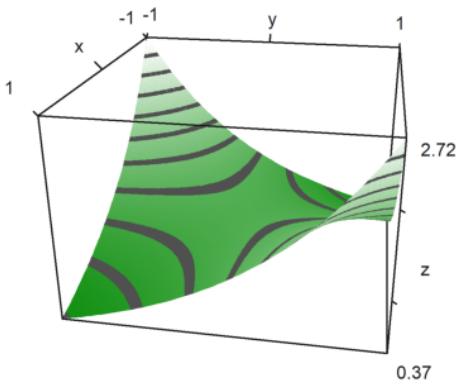
The default is level="auto", which computes some level lines automatically. As you see in the plot, the levels are in fact ranges of levels.

The default style can be changed. For the following contour plot, we use a finer grid fo 100x100 points, scale the function and the plot, and use different angle of view.

```
>plot3d("exp(-x^2-y^2)",r=2,n=100,level="thin", ...
> >contour,>spectral,fscale=1,scale=1.1,angle=45°,height=20°):
```

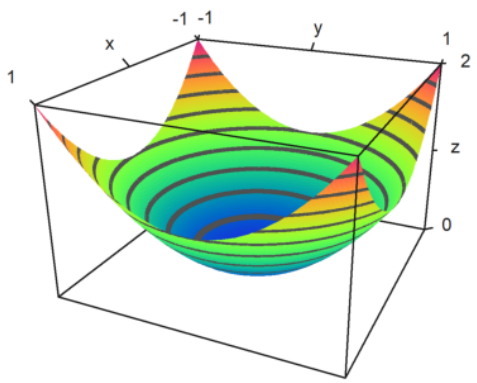


```
>plot3d("exp(x*y)",angle=100°,>contour,color=green):
```



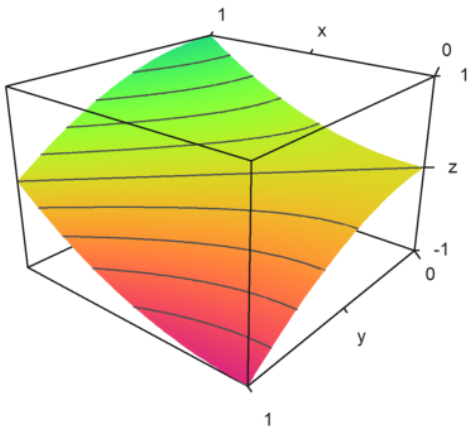
The default shading uses a gray color. But a spectral range of colors is also available.
- >spectral: Used the default spectral scheme
- color=...: Uses special colors or spectral schemes
For the following plot, we use the default spectral scheme and increase the number of points to get a very smooth look.

```
>plot3d("x^2+y^2",>spectral,>contour,n=100):
```



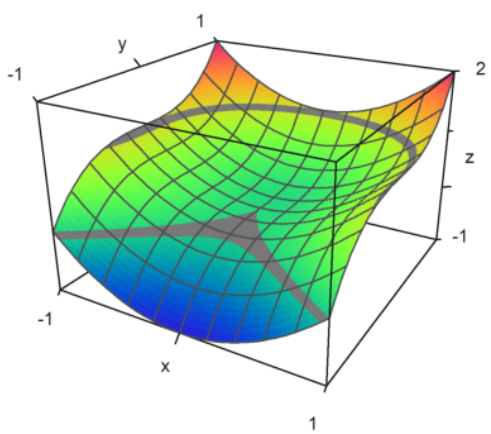
Instead of automatic level lines, we can also set values of the level lines. This will produce thin level lines instead of ranges of levels.

```
>plot3d("x^2-y^2",0,1,0,1,angle=220°,level=-1:0.2:1,color=redgreen):
```



In the following plot, we use two very broad level bands from -0.1 to 1, and from 0.9 to 1. This is entered as a matrix with level bounds as columns.
Moreover, we overlay a grid with 10 intervals in each direction.

```
>plot3d("x^2+y^3",level=[-0.1,0.9;0,1], ...
>  >spectral,angle=30°,grid=10,contourcolor=gray):
```

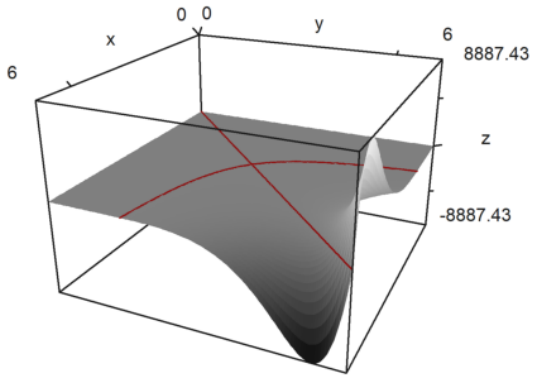


In the following example, we plot the set, where

$$f(x,y) = x^y - y^x = 0$$

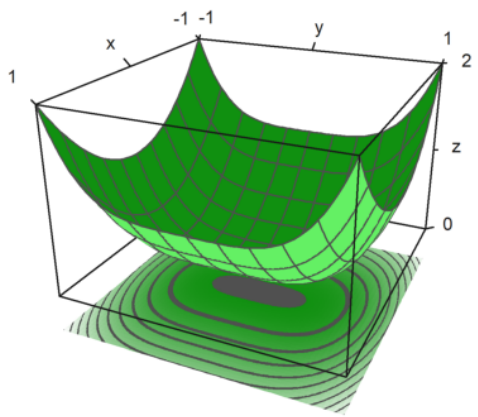
We use a single thin line for the level line.

```
>plot3d("x^y-y^x",level=0,a=0,b=6,c=0,d=6,contourcolor=red,n=100):
```



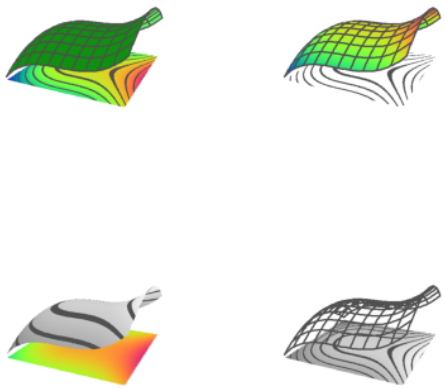
It is possible to show a contour plane below the plot. A color and distance to the plot can be specified.

```
>plot3d("x^2+y^4",>cp,cpcolor=green,cpdelta=0.2):
```



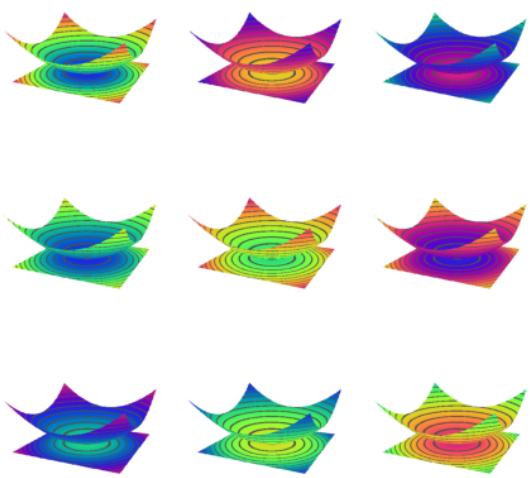
Here are a few more styles. We always turn off the frame, and use various color schemes for the plot and the grid.

```
>figure(2,2); ...
>expr="y^3-x^2"; ...
>figure(1);  ...
>  plot3d(expr,<frame,>cp,cpcolor=spectral); ...
>figure(2);  ...
>  plot3d(expr,<frame,>spectral,grid=10,cp=2); ...
>figure(3);  ...
>  plot3d(expr,<frame,>contour,color=gray,nc=5,cp=3,cpcolor=greenred); ...
>figure(4);  ...
>  plot3d(expr,<frame,>hue,grid=10,>transparent,>cp,cpcolor=gray); ...
>figure(0):
```



There are some other spectral schemes, numbered from 1 to 9. But you can also use the color=value, where value
- spectral: for a range from blue to red
- white: for a fainter range
- yellowblue,purplegreen,blueyellow,greenred
- blueyellow, greenpurple,yellowblue,redgreen

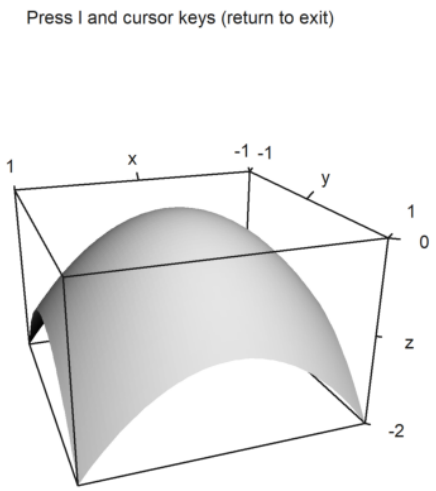
```
>figure(3,3); ...
>for i=1:9;  ...
>  figure(i); plot3d("x^2+y^2",spectral=i,>contour,>cp,<frame,zoom=4);  ...
>end; ...
>figure(0):
```



The light source can be changed with l and the cursor keys during the user interaction. It can also be set with parameters.
- light: a direction for the light
- amb: ambient light between 0 and 1

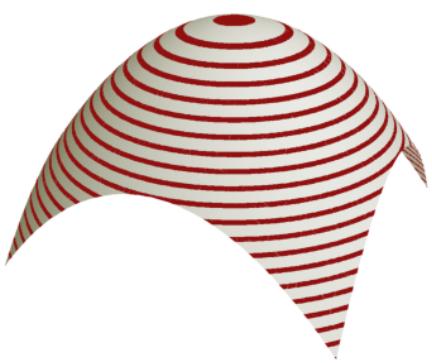
Note that the program does not make a difference between the sides of the plot. There are no shadows. For this you would need Povray.

```
>plot3d("-x^2-y^2", ...
>  hue=true,light=[0,1,1],amb=0,user=true, ...
>  title="Press l and cursor keys (return to exit)"):
```



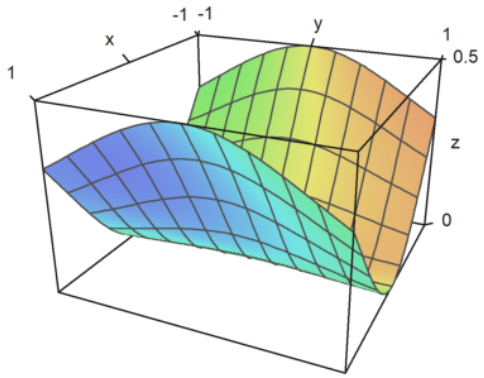
The color parameter changes the color of the surface. The color of the level lines can also be changed.

```
>plot3d("-x^2-y^2",color=rgb(0.2,0.2,0),hue=true,frame=false, ...
>  zoom=3,contourcolor=red,level=-2:0.1:1,dl=0.01):
```



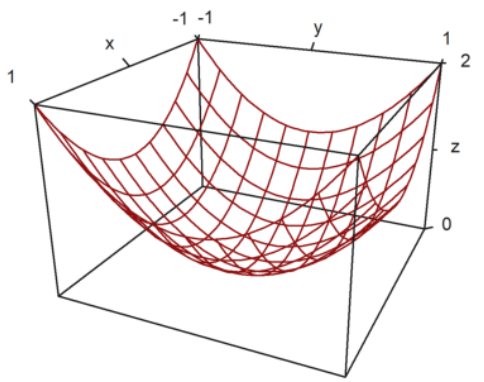
The color 0 gives a special rainbow effect.

```
>plot3d("x^2/(x^2+y^2+1)",color=0,hue=true,grid=10):
```



The surface can also be transparent.

```
>plot3d("x^2+y^2",>transparent,grid=10,wirecolor=red):
```



Implicit Plots

There are also implicit plots in three dimensions. Euler generates cuts through the objects. The features of plot3d include implicit plots. These plots show the zero set of a function in three variables.
The solutions of

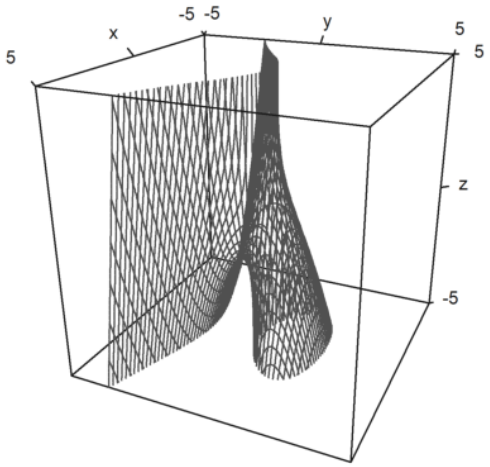
$$f(x,y,z)=0$$

can be visualized in cuts parallel to the x-y-, the x-z- and the y-z-plane.
- implicit=1: cut parallel to the y-z-plane
- implicit=2: cut parallel to the x-z-plane
- implicit=4: cut parallel to the x-y-plane

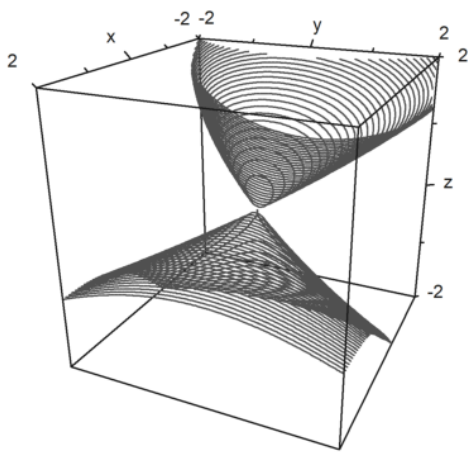
Add these values, if you like. In the example we plot

$$M=\{(x,y,z):x^2+y^3+zy=1\}$$

```
>plot3d("x^2+y^3+z*y-1",r=5,implicit=3):
```




```
>plot3d("x^2+y^2+4*x*z+z^3",>implicit,r=2,zoom=2.5):
```



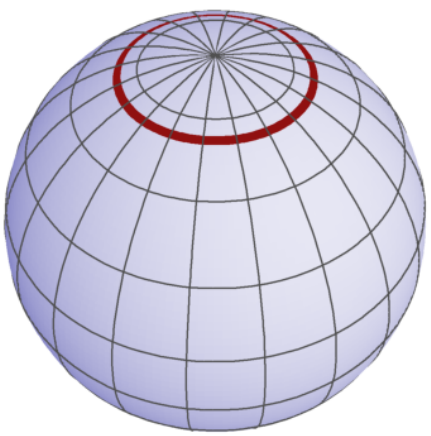
Plotting 3D Data

Just as plot2d, plot3d accepts data. For 3D objects, you need to provide a matrix of x-, y- and z-values, or three functions or expressions fx(x,y), fy(x,y), fz(x,y).

$$\gamma(t,s) = (x(t,s), y(t,s), z(t,s))$$

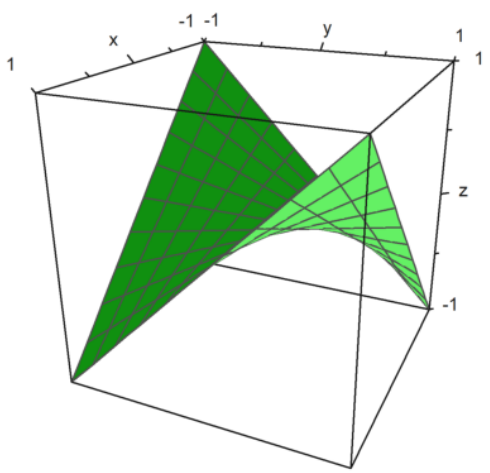
Since x,y,z are matrices, we assume that (t,s) run through a square grid. As a result, you can plot images of rectangles in space.
You can use the Euler matrix language to produce the coordinates effectively.
In the following example, we use a vector of t values and a column vector of s values to parameterize the surface of the ball. In the drawing we can mark regions, in our case the polar region.

```
>t=linspace(0,2pi,180); s=linspace(-pi/2,pi/2,90)'; ...
>x=cos(s)*cos(t); y=cos(s)*sin(t); z=sin(s); ...
>plot3d(x,y,z,>hue, ...
>color=blue,<frame,grid=[10,20], ...
>values=s,contourcolor=red,level=[90°-24°;90°-22°], ...
>scale=1.4,height=50°):
```



Here is an example, which is the graph of a function.

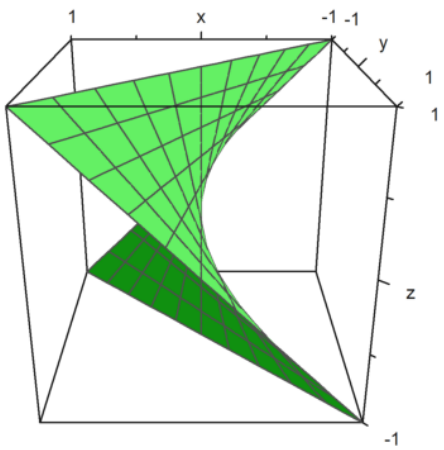
```
>t=-1:0.1:1; s=(-1:0.1:1)'; plot3d(t,s,t*s,grid=10):
```



However, we can make all sorts of surfaces. Here is the same surface as a function

$$x = y\,z$$

```
>plot3d(t*s,t,s,angle=180°,grid=10):
```



With more effort, we can produce many surfaces.
In the following example we make a shaded view of a distorted ball. The usual coordinates for the ball are

$$\gamma(t,s) = (\cos(t)\cos(s), \sin(t)\sin(s), \cos(s))$$

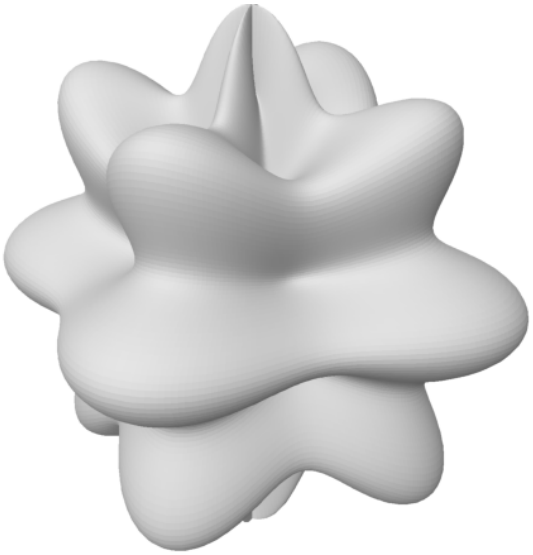
with

$$0 \le t \le 2\pi, \quad \frac{-\pi}{2} \le s \le \frac{\pi}{2}.$$

We distored this with a factor

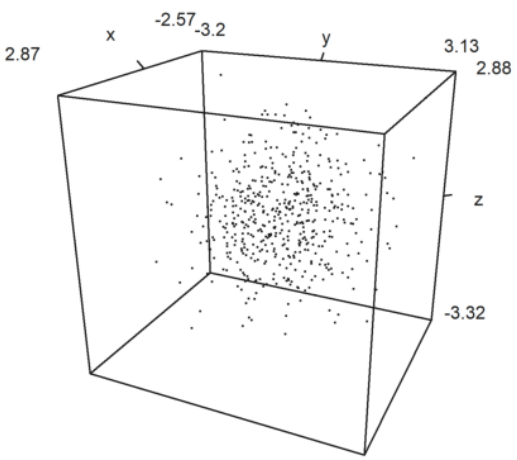
$$d(t,s) = \frac{\cos(4t)+\cos(8s)}{4}.$$

```
>t=linspace(0,2pi,320); s=linspace(-pi/2,pi/2,160)'; ...
>d=1+0.2*(cos(4*t)+cos(8*s)); ...
>plot3d(cos(t)*cos(s)*d,sin(t)*cos(s)*d,sin(s)*d,hue=1, ...
>  light=[1,0,1],frame=0,zoom=5):
```



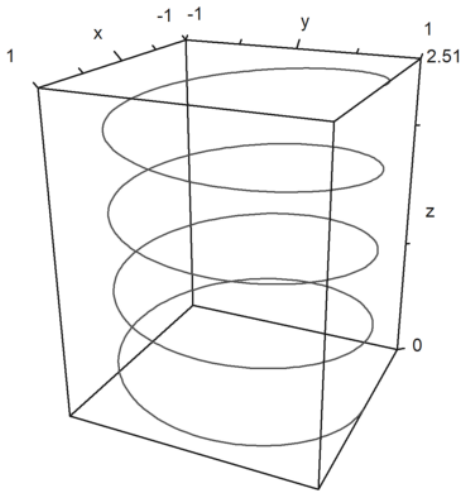
Of course, a point cloud is also possible. To plot point data in the space, we need three vectors for the coordinates of the points.
The styles are just as in plot2d with points=true;

```
>n=500;  ...
>  plot3d(normal(1,n),normal(1,n),normal(1,n),points=true,style="."):
```

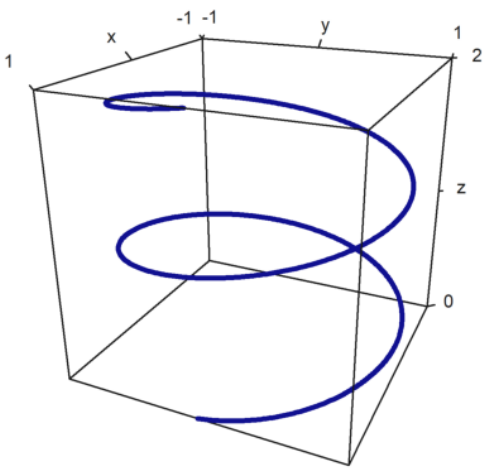


It is also possible to plot a curve in 3D. In this case, it is easier to precompute the points of the curve. For curves in the plane we use a sequence of coordinates and the parameter wire=true.

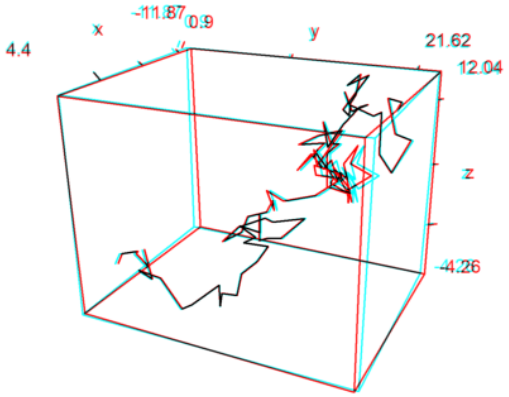
```
>t=linspace(0,8pi,500); ...
>plot3d(sin(t),cos(t),t/10,>wire,zoom=3):
```




```
>t=linspace(0,4pi,1000); plot3d(cos(t),sin(t),t/2pi,>wire, ...
>linewidth=3,wirecolor=blue):
```

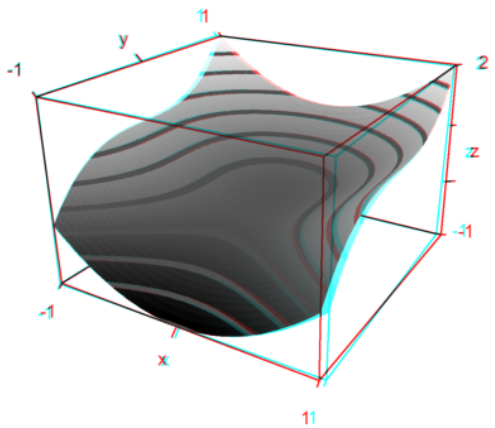



```
>X=cumsum(normal(3,100)); ...
> plot3d(X[1],X[2],X[3],>anaglyph,>wire):
```



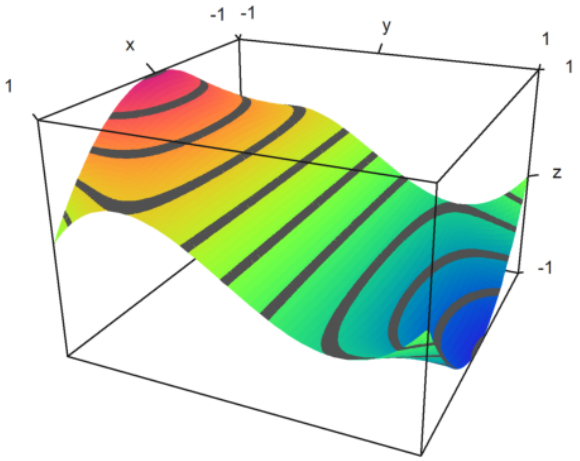
EMT can also plot in anaglyph mode. To view such a plot, you need red/cyan glasses.

```
> plot3d("x^2+y^3",>anaglyph,>contour,angle=30°):
```



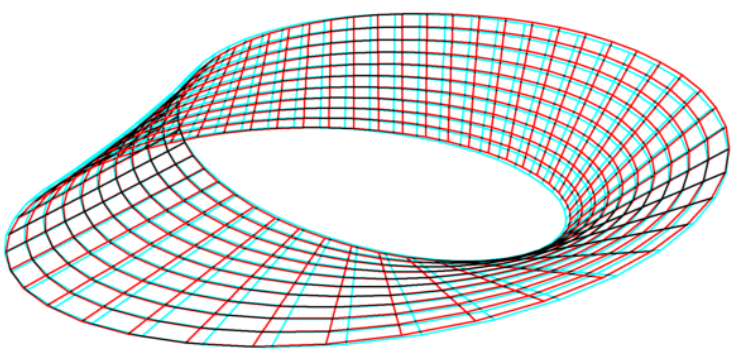
Often, a spectral color scheme is used for plots. This emphasizes the heights of the function.

```
>plot3d("x^2*y^3-y",>spectral,>contour,zoom=3.2):
```



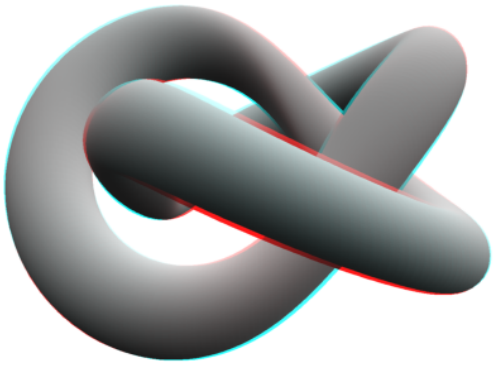
Euler can plot parameterized surfaces too, when the parameters are the x-, y-, and z-values of an image of a rectangular grid in the space.
For the following demo, we setup u- and v- parameters, and generate space coordinates from these.

```
>u=linspace(-1,1,10); v=linspace(0,2*pi,50)'; ...
>X=(3+u*cos(v/2))*cos(v); Y=(3+u*cos(v/2))*sin(v); Z=u*sin(v/2); ...
>plot3d(X,Y,Z,>anaglyph,<frame,>wire,scale=2.3):
```



Here is a more complicated example, which is majestic with red/cyan glasses.

```
>u:=linspace(-pi,pi,160); v:=linspace(-pi,pi,400)';  ...
>x:=(4*(1+.25*sin(3*v))+cos(u))*cos(2*v); ...
>y:=(4*(1+.25*sin(3*v))+cos(u))*sin(2*v); ...
> z=sin(u)+2*cos(3*v); ...
>plot3d(x,y,z,frame=0,scale=1.5,hue=1,light=[1,0,-1],zoom=2.8,>anaglyph):
```

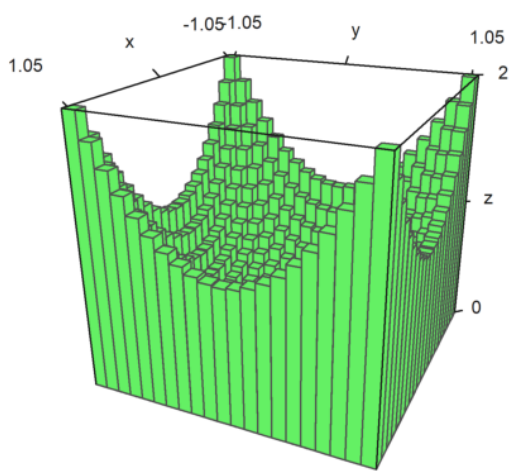


Statistical PLots

Bar plots are possible too. For this, we have to provide
- x: row vector with n+1 elements
- y: column vector with n+1 elements
- z: nxn matrix of values.

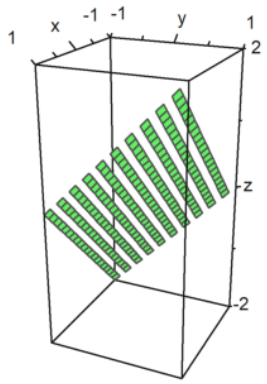
z can be larger, but only nxn values will be used.
In the example, we first compute the values. Then we adjust x and y, so that the vectors center at the values used.

```
>x=-1:0.1:1; y=x'; z=x^2+y^2; ...
>xa=(x|1.1)-0.05; ya=(y_1.1)-0.05; ...
>plot3d(xa,ya,z,bar=true):
```



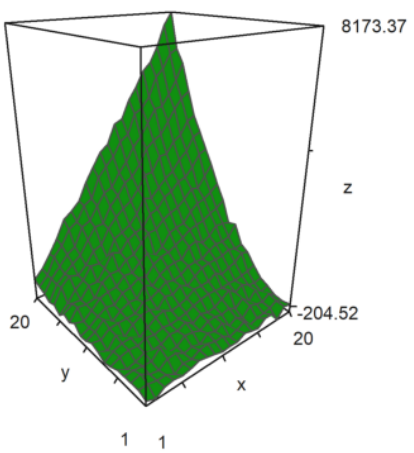
It is possible to split the plot of a surface in two or more parts.

```
>x=-1:0.1:1; y=x'; z=x+y; d=zeros(size(x)); ...
>plot3d(x,y,z,disconnect=2:2:20):
```

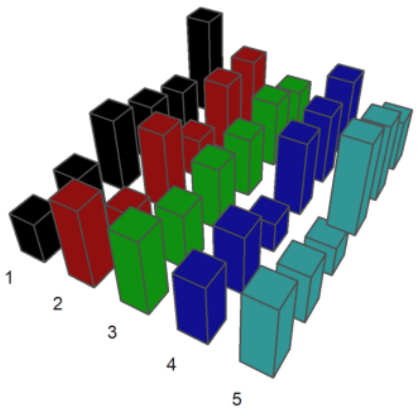


If load or generate a data matrix M from a file and need to plot it in 3D you can either scale the matrix to [-1,1] with scale(M), or scale the matrix with >zscale. This can be combined with individual scaling factors which are applied additionally.

```
>i=1:20; j=i'; ...
>plot3d(i*j^2+100*normal(20,20),>zscale,scale=[1,1,1.5],angle=-40°,zoom=1.8):
```

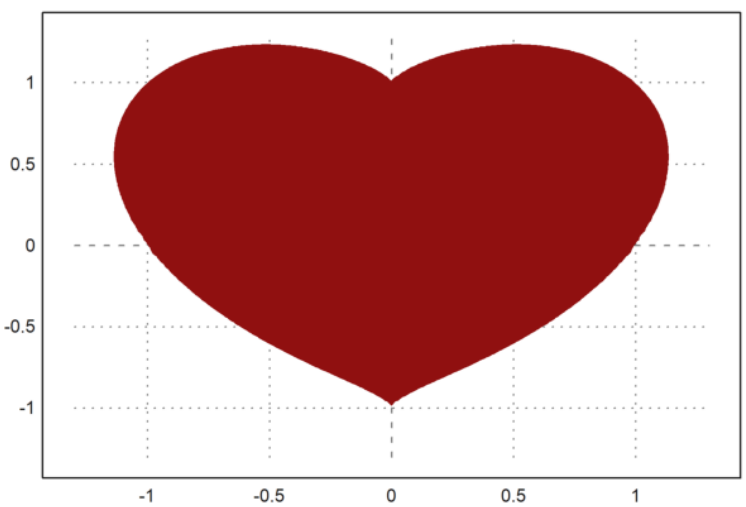



```
>Z=intrandom(5,100,6); v=zeros(5,6); ...
>loop 1 to 5; v[#]=getmultiplicities(1:6,Z[#]); end; ...
>columnsplot3d(v',scols=1:5,ccols=[1:5]):
```



Permukaan Benda Putar

```
>plot2d("(x^2+y^2-1)^3-x^2*y^3",r=1.3, ...
>style="#",color=red,<outline, ...
>level=[-2;0],n=100):
```




```
>ekspresi &= (x^2+y^2-1)^3-x^2*y^3; $ekspresi
```

$$\left(y^2+x^2-1\right)^3-x^2\,y^3$$

We wish to turn the heart curve around the y-axis. Here is the expression, which defines the heart:

$$f(x,y)=(x^2+y^2-1)^3-x^2.y^3.$$

Next we set

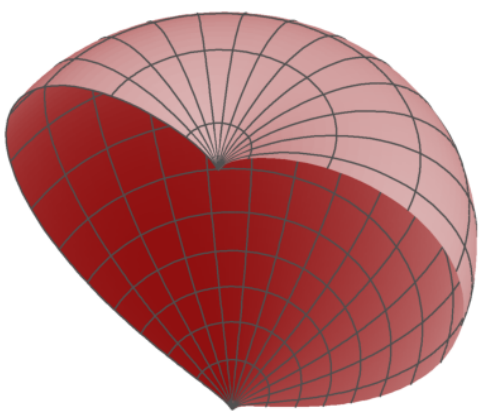
$$x = r.cos(a), \quad y = r.sin(a).$$

```
>function fr(r,a) &= ekspresi with [x=r*cos(a),y=r*sin(a)] | trigreduce; $fr(r,a)
```

$$\left(r^2-1\right)^3+\frac{\left(\sin\left(5\,a\right)-\sin\left(3\,a\right)-2\,\sin a\right)\,r^5}{16}$$

This allows to define a numerical function, which solves for r, if a is given. With that function we can plot the turned heart as a parametric surface.

```
>function map f(a) := bisect("fr",0,2;a); ...
>t=linspace(-pi/2,pi/2,100); r=f(t);  ...
>s=linspace(pi,2pi,100)'; ...
>plot3d(r*cos(t)*sin(s),r*cos(t)*cos(s),r*sin(t), ...
>>hue,<frame,color=red,zoom=4,amb=0,max=0.7,grid=12,height=50°):
```

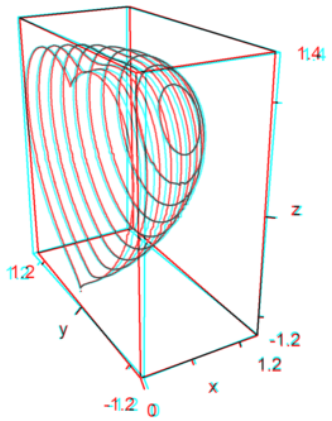


The following is a 3D plot of the figure above rotated around the z-axis. We define the function, which describes the object.

```
>function f(x,y,z) ...
```

```
  r=x^2+y^2;
  return (r+z^2-1)^3-r*z^3;
   endfunction
```

```
>plot3d("f(x,y,z)", ...
>xmin=0,xmax=1.2,ymin=-1.2,ymax=1.2,zmin=-1.2,zmax=1.4, ...
>implicit=1,angle=-30°,zoom=2.5,n=[10,60,60],>anaglyph):
```



Special 3D Plots

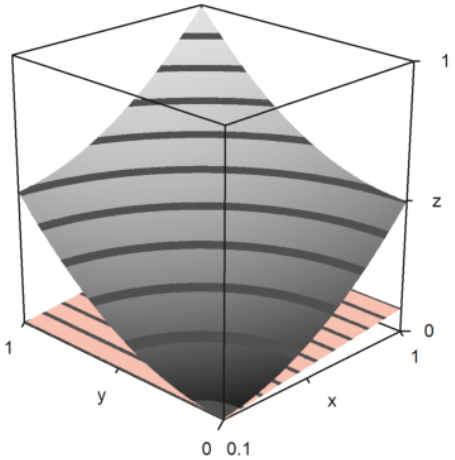
The plot3d function is nice to have, but it does not satisfy all needs. Besides more basic routines, it is possible to get a framed plot of any object you like.
Though Euler is not a 3D program, it can combine some basic objects. We try to visualize a paraboloid and its tangent.

```
>function myplot ...
```

```
    y=0:0.01:1; x=(0.1:0.01:1)';
    plot3d(x,y,0.2*(x-0.1)/2,<scale,<frame,>hue, ..
      hues=0.5,>contour,color=orange);
    h=holding(1);
    plot3d(x,y,(x^2+y^2)/2,<scale,<frame,>contour,>hue);
    holding(h);
  endfunction
```

Now framedplot() provides the frames, and sets the views.

```
>framedplot("myplot",[0.1,1,0,1,0,1],angle=-45°, ...
>  center=[0,0,-0.7],zoom=6):
```

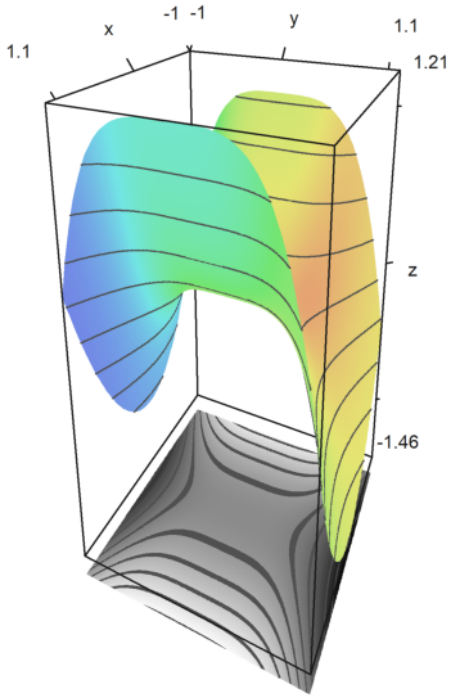


In the same way, you can plot the contour plane manually. Note that plot3d() sets the window to fullwindow() by default, but plotcontourplane() assumes that.

```
>x=-1:0.02:1.1; y=x'; z=x^2-y^4;
>function myplot (x,y,z) ...
```

```
    zoom(2);
    wi=fullwindow();
    plotcontourplane(x,y,z,level="auto",<scale);
    plot3d(x,y,z,>hue,<scale,>add,color=white,level="thin");
    window(wi);
    reset();
  endfunction
```

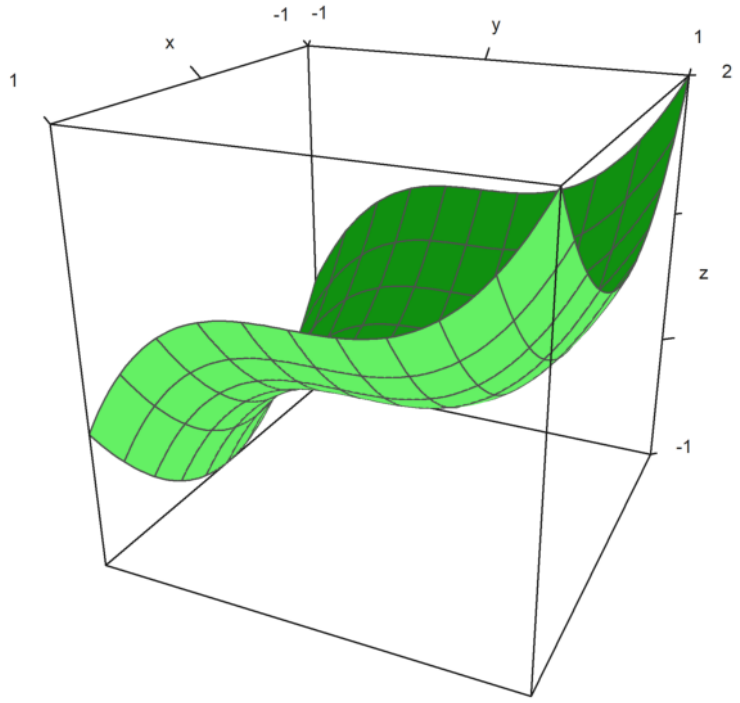
```
>myplot(x,y,z):
```



Animation

Euler can use frames to pre-compute the animation.
One function, which makes use of this technique is rotate. It can change the angle of view and redraw a 3D plot. The function calls addpage() for each new plot. Finally it animates the plots.
Please study the source of rotate to see more details.

```
>function testplot () := plot3d("x^2+y^3"); ...
>rotate("testplot"); testplot():
```



Menggambar Povray

With the help of the Euler file povray.e, Euler can generate Povray files. The results are very nice to look at.
You need to install Povray (32bit or 64bit) from http://www.povray.org/, and put the sub-directory "bin" of Povray into the environment path, or set the variable "defaultpovray" with full path pointing to "pvengine.exe".
The Povray interface of Euler generates Povray files in the home directory of the user, and calls Povray to parse these files. The default file name is current.pov, and the default directory is eulerhome(), usually c:\Users\Username\Euler. Povray generates a PNG file, which can be loaded by Euler into a notebook. To clean up these files, use povclear().
The pov3d function is in the same spirit as plot3d. It can generate the graph of a function f(x,y), or a surface with coordinates X,Y,Z in matrices, including optional level lines. This function starts the raytracer automatically, and loads the scene into the Euler notebook.
Besides pov3d(), there are many functions, which generate Povray objects. These functions return strings, containing the Povray code for the objects. To use these functions, start the Povray file with povstart(). Then use writeln(...) to write the objects to the scene file. Finally, end the file with povend(). By default, the raytracer will start, and the PNG will be inserted into the Euler notebook.
The object functions have a parameter called "look", which needs a string with Povray code for the texture and the finish of the object. The function povlook() can be used to produce this string. It has parameters for the color, the transparency, Phong Shading etc.
Note that the Povray universe has another coordinate system. This interface translates all coordinates to the Povray system. So you can keep thinking in the Euler coordinate system with z pointing vertically upwards,a nd x,y,z axes in right hand sense.
You need to load the povray file.

```
>load povray;
```

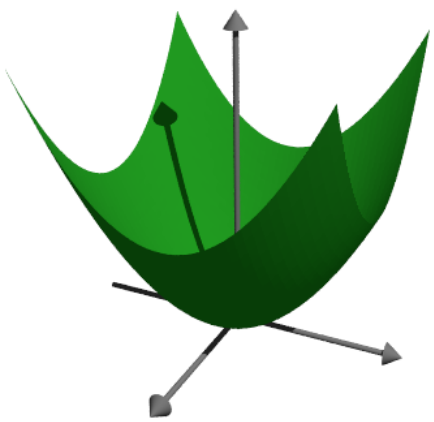
Make sure, the Povray bin directory is in the path. If it is not edit the following variable so that it contains the path to the povray executable.

```
>defaultpovray="C:\Program Files\POV-Ray\v3.7\bin\pvengine.exe"
```

```
C:\Program Files\POV-Ray\v3.7\bin\pvengine.exe
```

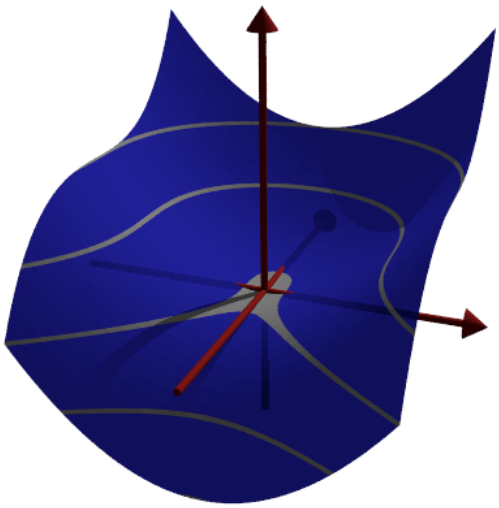
For a first impression, we plot a simple function. The following command generates a povray file in your user directory, and runs Povray for ray tracing this file.
If you start the following command, the Povray GUI should open, run the file, and close automatically. Due to security reasons, you will be asked, if you want to allow the exe file to run. You can press cancel to stop further questions. You may have to press OK in the Povray window to acknowledge the start-up dialog of Povray.

```
>pov3d("x^2+y^2",zoom=3);
```



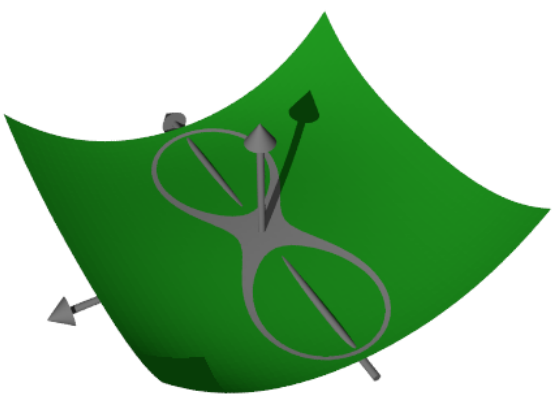
We can make the function transparent and add another finish. We can also add level lines to the function plot.

```
>pov3d("x^2+y^3",axiscolor=red,angle=20°, ...
>  look=povlook(blue,0.2),level=-1:0.5:1,zoom=3.8);
```



Sometimes it is necessary to prevent the scaling of the function, and scale the function by hand.
We plot the set of points in the complex plane, where the product of the distances to 1 and -1 is equal to 1.

```
>pov3d("((x-1)^2+y^2)*((x+1)^2+y^2)/40",r=1.5, ...
>  angle=-120°,level=1/40,dlevel=0.005,light=[-1,1,1],height=45°,n=50, ...
>  <fscale,zoom=3.8);
```



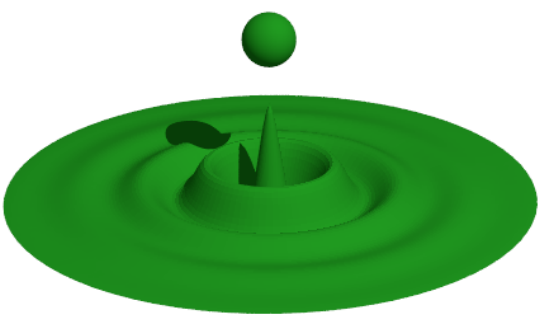
Plotting with Coordinates

Instead of functions, we can plot with coordinates. As in plot3d, we need three matrices to define the object.
In the example we turn a function around the z-axis.

```
>function f(x) := x^3-x+1; ...
>x=-1:0.01:1; t=linspace(0,2pi,8)'; ...
>Z=x; X=cos(t)*f(x); Y=sin(t)*f(x); ...
>pov3d(X,Y,Z,angle=40°,height=20°,axis=0,zoom=4,light=[10,-5,5]);
```

In the following example, we plot a damped wave. We generate the wave with the matrix language of Euler. We also show, how an additional object can be added to a pov3d scene. For the generation of objects, see the following examples. Note that plot3d scales the plot, so that it fits into the unit cube.

```
>r=linspace(0,1,80); phi=linspace(0,2pi,80)'; ...
>x=r*cos(phi); y=r*sin(phi); z=exp(-5*r)*cos(8*pi*r)/3;  ...
>pov3d(x,y,z,zoom=5,axis=0,add=povsphere([0,0,0.5],0.1,povlook(green)), ...
>  w=500,h=300);
```



With the advanced shading method of Povray, very few points can produce very smooth surfaces. Only at the boundaries and in shadows the trick might become obvious.
For this, we need to add normal vectors in each matrix point.

```
>Z &= x^2*y^3
```

$$x^2 y^3$$

The equation of the surface is [x,y,Z]. We compute the two derivatives to x and y of this and take the cross product as the normal.

```
>dx &= diff([x,y,Z],x); dy &= diff([x,y,Z],y);
```

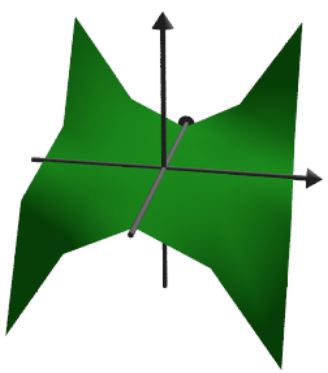
We define the normal as the cross product of these derivatives, and define coordinate functions.

```
>N &= crossproduct(dx,dy); NX &= N[1]; NY &= N[2]; NZ &= N[3]; N,
```

$$\left[-2\,x\,y,\ -3\,x^2\,y^2,\ 1\right]$$

We use only 25 points.

```
>x=-1:0.5:1; y=x';
>pov3d(x,y,Z(x,y),angle=10°, ...
>  xv=NX(x,y),yv=NY(x,y),zv=NZ(x,y),<shadow);
```



The following is the Trefoil knot done by A. Busser in Povray. There is an improved version of this in the examples.
See: Examples\Trefoil Knot | Trefoil Knot
For a good look with not too many points, we add normal vectors here. We use Maxima to compute the normals for us. First, the three functions for the coordinates as symbolic expressions.

```
>X &= ((4+sin(3*y))+cos(x))*cos(2*y); ...
>Y &= ((4+sin(3*y))+cos(x))*sin(2*y); ...
>Z &= sin(x)+2*cos(3*y);
```

Then the two derivative vectors to x and y.

```
>dx &= diff([X,Y,Z],x); dy &= diff([X,Y,Z],y);
```

Now the normal, which is the cross product of the two derivatives.

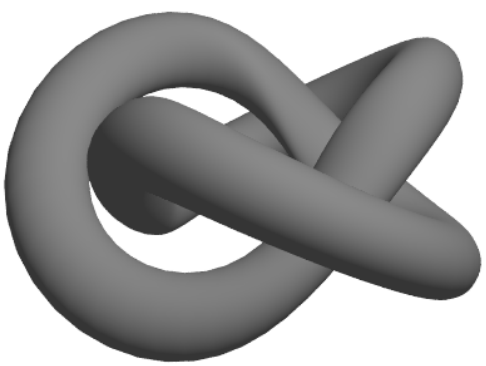
```
>dn &= crossproduct(dx,dy);
```

We now evaluate all this numerically.

```
>x:=linspace(-%pi,%pi,40); y:=linspace(-%pi,%pi,100)';
```

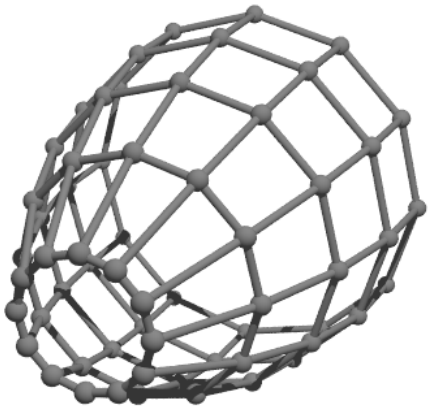
The normal vectors are evaluations of the symbolic expressions dn[i] for i=1,2,3. The syntax for this is &"expression"(parameters). This is an alternative to the method in the previous example, where we defined symbolic expressions NX, NY, NZ first.

```
>pov3d(X(x,y),Y(x,y),Z(x,y),axis=0,zoom=5,w=450,h=350, ...
>  <shadow,look=povlook(gray), ...
>  xv=&"dn[1]"(x,y), yv=&"dn[2]"(x,y), zv=&"dn[3]"(x,y));
```



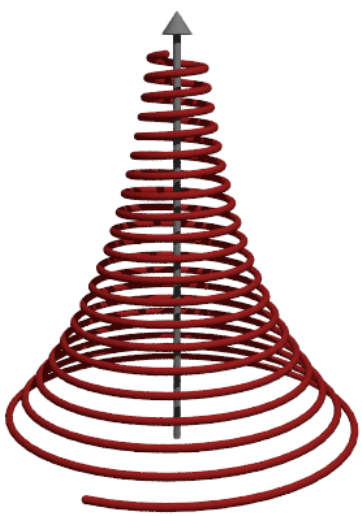
We can also generate a grid in 3D.

```
>povstart(zoom=4); ...
>x=-1:0.5:1; r=1-(x+1)^2/6; ...
>t=(0°:30°:360°)'; y=r*cos(t); z=r*sin(t); ...
>writeln(povgrid(x,y,z,d=0.02,dballs=0.05)); ...
>povend();
```



With povgrid(), curves are possible.

```
>povstart(center=[0,0,1],zoom=3.6); ...
>t=linspace(0,2,1000); r=exp(-t); ...
>x=cos(2*pi*10*t)*r; y=sin(2*pi*10*t)*r; z=t; ...
>writeln(povgrid(x,y,z,povlook(red))); ...
>writeAxis(0,2,axis=3); ...
>povend();
```



Povray Objects

Above, we used pov3d to plot surfaces. The povray interface in Euler can also generate Povray objects. These objects are stored as strings in Euler, and need to be written to a Povray file.
We start the output with povstart().

```
>povstart(zoom=4);
```

First we define the three cylinders, and store them in strings in Euler.
The functions povx() etc. simply returns the vector [1,0,0], which could be used instead.

```
>c1=povcylinder(-povx,povx,1,povlook(red)); ...
>c2=povcylinder(-povy,povy,1,povlook(green)); ...
>c3=povcylinder(-povz,povz,1,povlook(blue)); ...
```

The strings contain Povray code, which we need not understand at that point.

```
>c1
```

```
  cylinder { <-1,0,0>, <1,0,0>, 1
   texture { pigment { color rgb <0.564706,0.0627451,0.0627451> }  }
   finish { ambient 0.2 }
   }
```

As you see, we added texture to the objects in three different colors.
That is done by povlook(), which returns a string with the relevant Povray code. We can use the default Euler colors, or define our own color. We can also add transparency, or change the ambient light.

```
>povlook(rgb(0.1,0.2,0.3),0.1,0.5)
```

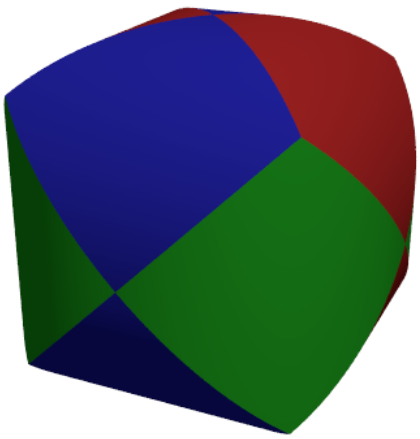
```
   texture { pigment { color rgbf <0.101961,0.2,0.301961,0.1> }  }
   finish { ambient 0.5 }
```

Now we define an intersection object, and write the result to the file.

```
>writeln(povintersection([c1,c2,c3]));
```

The intersection of three cylinders is hard to visualize, if you never saw it before.

```
>povend;
```

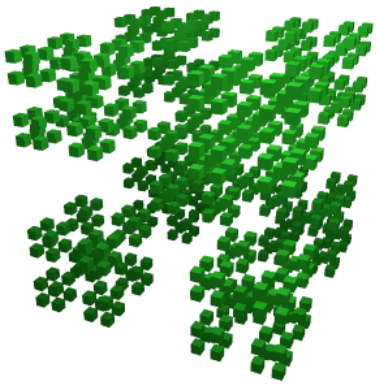


The following functions generate a fractal recursively.
The first function shows, how Euler handles simple Povray objects. The function povbox() returns a string, containing the box coordinates, the texture and the finish.

```
>function onebox(x,y,z,d) := povbox([x,y,z],[x+d,y+d,z+d],povlook());
>function fractal (x,y,z,h,n) ...
```

```
  if n==1 then writeln(onebox(x,y,z,h));
  else
    h=h/3;
    fractal(x,y,z,h,n-1);
    fractal(x+2*h,y,z,h,n-1);
    fractal(x,y+2*h,z,h,n-1);
    fractal(x,y,z+2*h,h,n-1);
    fractal(x+2*h,y+2*h,z,h,n-1);
    fractal(x+2*h,y,z+2*h,h,n-1);
    fractal(x,y+2*h,z+2*h,h,n-1);
    fractal(x+2*h,y+2*h,z+2*h,h,n-1);
    fractal(x+h,y+h,z+h,h,n-1);
  endif;
endfunction
```

```
>povstart(fade=10,<shadow);
>fractal(-1,-1,-1,2,4);
>povend();
```



Differences allow cutting off one object from another. Like intersections, there are part of the CSG objects of Povray.

```
>povstart(light=[5,-5,5],fade=10);
```

For this demonstration, we define an object in Povray, instead of using a string in Euler. Definitions are written to the file immediately.
A box coordinate of -1 just means [-1,-1,-1].

```
>povdefine("mycube",povbox(-1,1));
```

We can use this object in povobject(), which returns a string as usual.

```
>c1=povobject("mycube",povlook(red));
```

We generate a second cube, and rotate and scale it a bit.

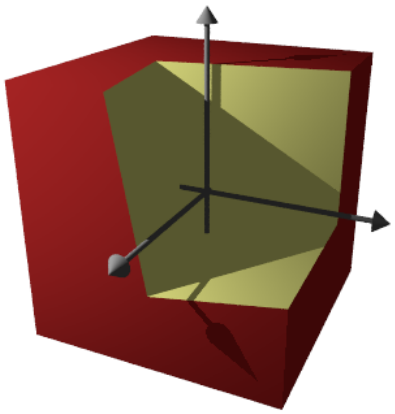
```
>c2=povobject("mycube",povlook(yellow),translate=[1,1,1], ...
>  rotate=xrotate(10°)+yrotate(10°), scale=1.2);
```

Then we take the difference of the two objects.

```
>writeln(povdifference(c1,c2));
```

Now add three axes.

```
>writeAxis(-1.2,1.2,axis=1); ...
>writeAxis(-1.2,1.2,axis=2); ...
>writeAxis(-1.2,1.2,axis=4); ...
>povend();
```



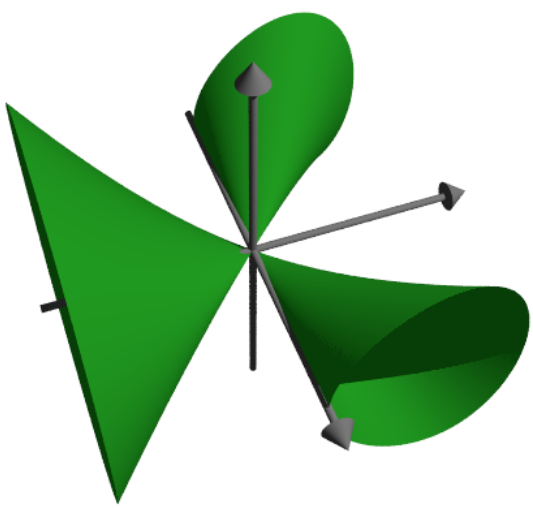
Implicit Functions

Povray can plot the set where f(x,y,z)=0, just like the implicit parameter in plot3d. The results looks much better, however.
The syntax for the functions is a bit different. You cannot use the output of Maxima or Euler expressions.

```
>povstart(angle=70°,height=50°,zoom=4);
```

Create the implicit surface. Note the different syntax in the expression.

```
>writeln(povsurface("pow(x,2)*y-pow(y,3)-pow(z,2)",povlook(green))); ...
>writeAxes(); ...
>povend();
```



Mesh Object

In this example, we show how to create a mesh object, and draw it with additional information.
We like to maximize xy under the condition x+y=1 and demonstrate the tangential touching of the level lines.

```
>povstart(angle=-10°,center=[0.5,0.5,0.5],zoom=7);
```

We cannot store the object in a string as before, since is too large. So we define the object in a Povray file using declare. The function povtriangle() does this automatically. It can accept normal vectors just like pov3d().
The following defines the mesh object, and writes it immediately into the file.

```
>x=0:0.02:1; y=x'; z=x*y; vx=-y; vy=-x; vz=1;
>mesh=povtriangles(x,y,z,"",vx,vy,vz);
```

Now we define two discs, which will be intersected with the surface.

```
>cl=povdisc([0.5,0.5,0],[1,1,0],2); ...
>ll=povdisc([0,0,1/4],[0,0,1],2);
```

Write the surface minus the two discs.

```
>writeln(povdifference(mesh,povunion([cl,ll]),povlook(green)));
```

Write the two intersections.

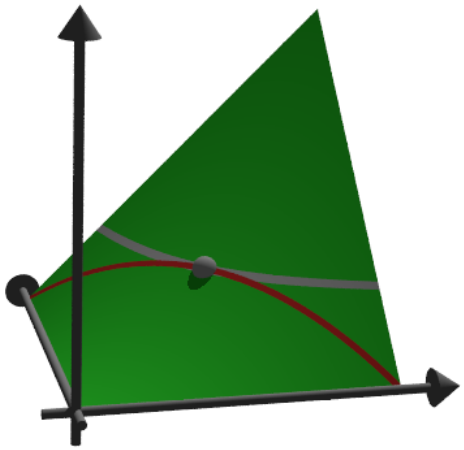
```
>writeln(povintersection([mesh,cl],povlook(red)); ...
>writeln(povintersection([mesh,ll],povlook(gray)));
```

Write a point at the maximum.

```
>writeln(povpoint([1/2,1/2,1/4],povlook(gray),size=2*defaultpointsize));
```

Add axes and finish.

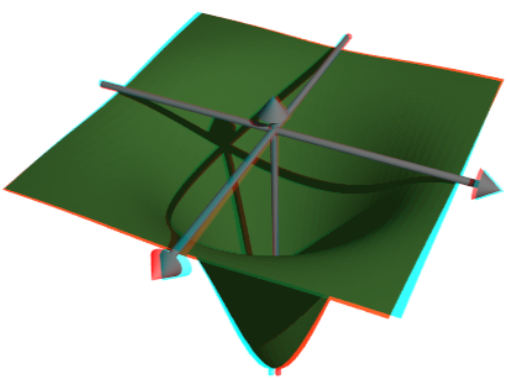
```
>writeAxes(0,1,0,1,0,1,d=0.015); ...
>povend();
```



Anaglyphs in Povray

To generate an anaglyph for a red/cyan glasses, Povray must run twice from different camera positions. It generates two Povray files and two PNG files, which are loaded with the function loadanaglyph().
Of course, you need red/cyan glasses to view the following examples properly.
The function pov3d() has a simple switch to generate anaglyphs.

```
>pov3d("-exp(-x^2-y^2)/2",r=2,height=45°,>anaglyph, ...
>  center=[0,0,0.5],zoom=3.5);
```



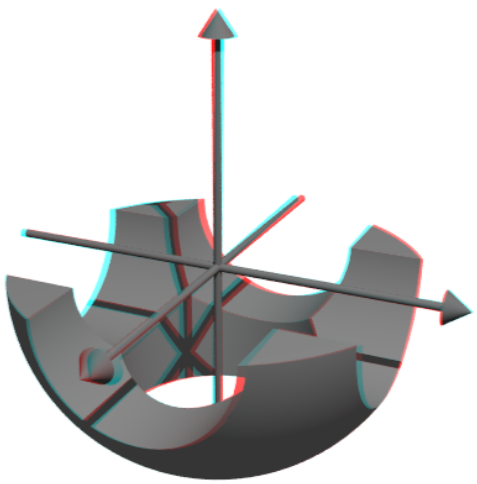
If you create a scene with objects, you need to put the generation of the scene into a function, and run it twice with different values for the anaglyph parameter.

```
>function myscene ...
```

```
    s=povsphere(povc,1);
    cl=povcylinder(-povz,povz,0.5);
    clx=povobject(cl,rotate=xrotate(90°));
    cly=povobject(cl,rotate=yrotate(90°));
    c=povbox([-1,-1,0],1);
    un=povunion([cl,clx,cly,c]);
    obj=povdifference(s,un,povlook(red));
    writeln(obj);
    writeAxes();
  endfunction
```

The function povanaglyph() does all this. The parameters are like in povstart() and povend() combined.

```
>povanaglyph("myscene",zoom=4.5);
```



Defining own Objects

The povray interface of Euler contains a lot of objects. But you are not restricted to these. You can create own objects, which combine other objects, or are completely new objects.
We demonstrate a torus. The Povray command for this is "torus". So we return a string with this command and its parameters. Note that the torus is always centered at the origin.

```
>function povdonat (r1,r2,look="") ...
```

```
    return "torus {"+r1+","+r2+look+"}";
  endfunction
```

Here is our first torus.

```
>t1=povdonat(0.8,0.2)
```

```
  torus {0.8,0.2}
```

Let us use this object to create a second torus, translated and rotated.

```
>t2=povobject(t1,rotate=xrotate(90°),translate=[0.8,0,0])
```

```
  object { torus {0.8,0.2}
   rotate 90 *x
   translate <0.8,0,0>
   }
```

Now we place these objects into a scene. For the look, we use Phong Shading.

```
>povstart(center=[0.4,0,0],angle=0°,zoom=3.8,aspect=1.5); ...
>writeln(povobject(t1,povlook(green,phong=1))); ...
>writeln(povobject(t2,povlook(green,phong=1))); ...
```

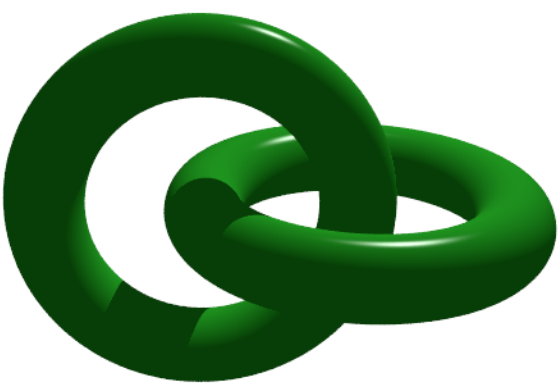
```
 >povend();
```

calls the Povray program. However, in case of errors, it does not display the error. You should therefore use

```
 >povend(<exit);
```

if anything did not work. This will leave the Povray window open.

```
>povend(h=320,w=480);
```



Here is a more elaborate example. We solve

$$Ax \le b, \quad x \ge 0, \quad c.x \to \text{Max}.$$

and show the feasible points and the optimum in a 3D plot.

```
>A=[10,8,4;5,6,8;6,3,2;9,5,6];
>b=[10,10,10,10]';
>c=[1,1,1];
```

First, let us check, if this example has a solution at all.

```
>x=simplex(A,b,c,>max,>check)'
```

```
  [0,  1,  0.5]
```

Yes, it has.
Next we define two objects. The first is the plane

$$a \cdot x \le b$$

```
>function oneplane (a,b,look="") ...
```

```
    return povplane(a,b,look)
  endfunction
```

Then we define the intersection of all half spaces and a cube.

```
>function adm (A, b, r, look="") ...
```

```
    ol=[];
    loop 1 to rows(A); ol=ol|oneplane(A[#],b[#]); end;
    ol=ol|povbox([0,0,0],[r,r,r]);
    return povintersection(ol,look);
  endfunction
```

We can now plot the scene.

```
>povstart(angle=120°,center=[0.5,0.5,0.5],zoom=3.5); ...
>writeln(adm(A,b,2,povlook(green,0.4))); ...
>writeAxes(0,1.3,0,1.6,0,1.5); ...
```

The following is a circle around the optimum.

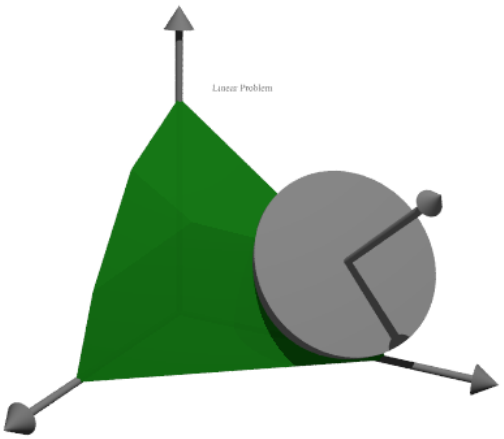
```
>writeln(povintersection([povsphere(x,0.5),povplane(c,c.x')], ...
>  povlook(red,0.9)));
```

And an error in the direction of the optimum.

```
>writeln(povarrow(x,c*0.5,povlook(red)));
```

We add text to the screen. Text is just a 3D object. We need to place and turn it according to our view.

```
>writeln(povtext("Linear Problem",[0,0.2,1.3],size=0.05,rotate=125°)); ...
>povend();
```



More Examples

You can find some more examples for Povray in Euler in the following files.
See: Examples/Dandelin Spheres
See: Examples/Donat Math
See: Examples/Trefoil Knot
See: Examples/Optimization by Affine Scaling