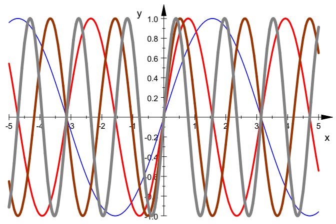
Mathematical graphics with MuPAD

The structure of MuPAD graphs

```
info(plot)
Library 'plot': graphical primitives and functions for two- and three-dimension\
al plots

-- Interface:
plot::AmbientLight,   plot::Arc2d,             plot::Arc3d,
plot::Arrow2d,        plot::Arrow3d,           plot::Bars2d,
plot::Bars3d,         plot::Box,               plot::Boxplot,
plot::Camera,         plot::Canvas,            plot::Circle2d,
plot::Circle3d,       plot::ClippingBox,       plot::Cone,
plot::Conformal,      plot::CoordinateSystem2d, plot::CoordinateSystem3d,
plot::Curve2d,        plot::Curve3d,           plot::Cylinder,
plot::Cylindrical,    plot::Density,           plot::DistantLight,
plot::Dodecahedron,   plot::Ellipse2d,         plot::Ellipse3d,
plot::Ellipsoid,      plot::Function2d,        plot::Function3d,
plot::Group2d,        plot::Group3d,           plot::Hatch,
plot::Hexahedron,     plot::Histogram2d,       plot::Icosahedron,
plot::Implicit2d,     plot::Implicit3d,        plot::Inequality,
plot::Integral,       plot::Iteration,         plot::Line2d,
plot::Line3d,         plot::Listplot,          plot::Lsys,
plot::Matrixplot,     plot::MuPADCube,         plot::Octahedron,
plot::Ode2d,          plot::Ode3d,             plot::Parallelogram2d,
plot::Parallelogram3d, plot::Piechart2d,        plot::Piechart3d,
plot::Plane,          plot::Point2d,           plot::Point3d,
plot::PointLight,     plot::PointList2d,       plot::PointList3d,
plot::Polar,          plot::Polygon2d,         plot::Polygon3d,
plot::Prism,          plot::Pyramid,           plot::QQplot,
plot::Raster,         plot::Rectangle,         plot::Reflect2d,
plot::Reflect3d,      plot::Rootlocus,         plot::Rotate2d,
plot::Rotate3d,       plot::Scale2d,           plot::Scale3d,
plot::Scatterplot,    plot::Scene2d,           plot::Scene3d,
plot::Sequence,       plot::SparseMatrixplot,  plot::Sphere,
plot::Spherical,      plot::SpotLight,         plot::Streamlines2d,
plot::Sum,            plot::Surface,           plot::SurfaceSTL,
plot::SurfaceSet,     plot::Sweep,             plot::Tetrahedron,
plot::Text2d,         plot::Text3d,            plot::Transform2d,
plot::Transform3d,    plot::Translate2d,       plot::Translate3d,
plot::Tube,           plot::Turtle,            plot::VectorField2d,
plot::VectorField3d,  plot::Waterman,          plot::XRotate,
plot::ZRotate,        plot::copy,              plot::delaunay,
plot::easy,           plot::getDefault,        plot::hull,
plot::modify,         plot::setDefault,
```

1 Function2d

```
G1 := plot::Function2d(sin(x), x=-5..5,
   LineColor=RGB::Blue,
   LineWidth=0.3*unit::mm
):
G2 := plot::Function2d(sin(2*x), x=-5..5,
   LineColor=RGB::Red,
   LineWidth=0.6*unit::mm
):
G3 := plot::Function2d(sin(3*x), x=-5..5,
   LineColor=RGB::Brown,
   LineWidth=0.8*unit::mm
):
G4 := plot::Function2d(sin(4*x), x=-5..5,
   LineColor=[0.5, 0.5, 0.5],
   LineWidth=1*unit::mm
):
plot(G1,G2,G3,G4) // here we plot objects G1,..,G4
```

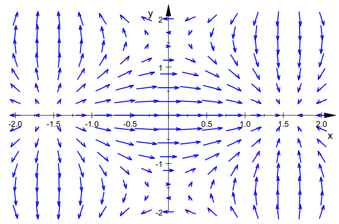


2 Vector field

```
z := (x,y)->sin(x)*cos(y)
```

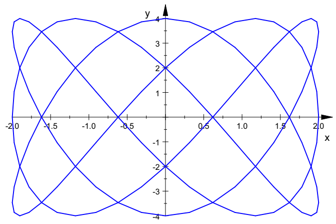
$(x, y) \rightarrow \sin(x)\cos(y)$

```
F := diff(z(x,y),x):
G := diff(z(x,y),y):
V := plot::VectorField2d([F,G], x=-2..2, y=-2..2,
   Mesh=[15,15]
):
plot(V)
```

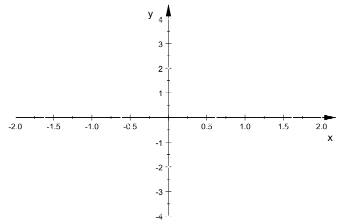


3 Curve2d

```
C1 := plot::Curve2d(
   [2*sin(3*t), 4*cos(5*t)], t=0..2*PI
):
plot(C1)
```

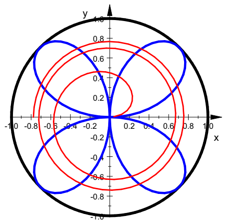



```
plot(plot::Curve2d([2*sin(3*x), 4*cos(5*x)], x=0..2*PI,
                      LineColorFunction = ((u, x, y, a) -> [(u-a)/5, (u-a)/5, 1]),
                      a = -5..5))
```



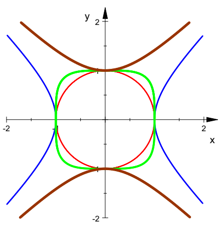
4 Polar coordinates

```
P1 := plot::Polar([1, alpha], alpha=0..2*PI,
   LineWidth=1, LineColor=RGB::Black
):
P2 := plot::Polar(
   [sin(2*alpha), alpha], alpha=0..2*PI,
   LineWidth=0.8, LineColor=RGB::Blue
):
P3 := plot::Polar([exp(-1/u^0.5),u], u=0.1..5*PI,
   LineWidth=0.5, LineColor=RGB::Red
):
plot(P1, P2, P3, Scaling=Constrained)
```



5 Implicit2d

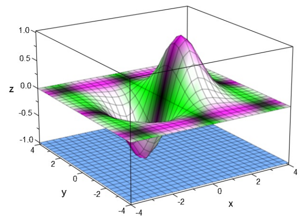
```
I1 := plot::Implicit2d(x^2+y^2=1, x=-2..2, y=-2..2,
   LineColor=RGB::Red
):
I2 := plot::Implicit2d(x^2-y^2=1, x=-2..2,y=-2..2,
   LineColor=RGB::Blue,
   LineWidth=0.5
):
I3 := plot::Implicit2d(x^4+y^4=1, x=-1..1, y=-1..1,
   LineColor=RGB::Green,
   LineWidth=0.7
):
I4 := plot::Implicit2d(-x^2+y^2=1, x=-2..2,y=-2..2,
   LineColor=RGB::Brown,
   LineWidth=0.9
):
plot(I1, I2, I3, I4, Scaling=Constrained)
```



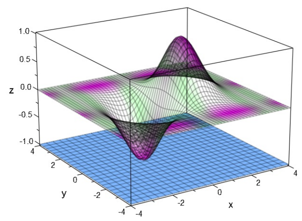
6 Function3d

```
K := (x,y,z)->[abs(sin(x)), abs(sin(y)), abs(sin(z))]:
```

```
F1 := plot::Function3d(
    1.7*x*exp(-1/2*(x^2+y^2)),
    x=-4..4, y=-4..4,
    FillColorFunction=K
):
F2 := plot::Function3d(-1, x=-4..4, y=-4..4):
plot(F1, F2)
```

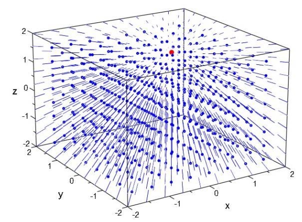


```
K := (x,y,z)->  [abs(sin(x)),abs(sin(y)),abs(sin(z)),abs(sin(z))]:
F1 := plot::Function3d(
    1.7*x*exp(-1/2*(x^2+y^2)),
    x=-4..4, y=-4..4,
    XMesh=50,
    YMesh=50,
    FillColorFunction=K
):
F2 := plot::Function3d(-1, x=-4..4, y=-4..4):
plot(F1, F2)
```



7 Vector field 3D

```
reset():
u := (x,y,z)->((x-1)^2+(y-1)^2+(z-1)^2):
F := diff(u(x,y,z),x):
G := diff(u(x,y,z),y):
H := diff(u(x,y,z),z):
V := plot::VectorField3d([F,G,H],
    x=-2..2, y=-2..2, z=-2..2,
    Mesh=[10,10,10],
    PointSize=1
):
P := plot::Point3d([1,1,1],
    PointColor=RGB::Red,
    PointSize=3,
    PointStyle=FilledCircles
):
plot(V,P)
```

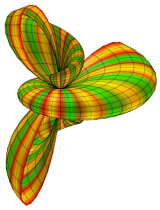


8 Surfaces

```
Cl := (u,v)->[u-trunc(u), v-trunc(v), 0]:
S1 := plot::Surface(
    [sin(u)*sin(v),
    sin(u)*cos(v),
    cos(u)*sin(v)],
    u=0..PI, v=0..PI,
    FillColorFunction=Cl
):
plot(S1)
Error: An arithmetical expression is expected. [sin]
```

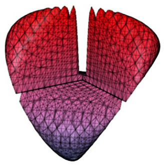
A threefold shell

```
h := (x,y,z,phi, theta)->[abs(1-sin(10*phi)),abs(sin(10*phi)),0]:
a := 1.3:
b := 3:
Shell := plot::Spherical(   [(a^phi)*sin(b*t),phi,t],    phi = -1..2*PI, t = 0..PI,    Mesh = [50,50],
FillColorFunction = h):
MyCam := plot::Camera(   [100,70,50],[0,0,-0.5], PI/45):
plot(   Shell, MyCam,    Scaling = Constrained,    Axes = None)
```



9 Implicit3d

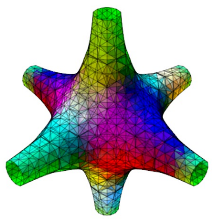
```
Steiner := plot::Implicit3d(
    y^2*z^2 + z^2*x^2 + x^2*y^2 + 2*x*y*z = 0,
    x=-1..1, y=-1..1, z=-1..1,
    MeshVisible=TRUE,
    XMesh=25, YMesh=25, ZMesh=25
):
plot(Steiner, Axes=None, Scaling=Constrained)
```



```
Cl := (u,v,x,y,z)->[abs(x),abs(y),abs(z)]:
SteinerParam:=plot::Surface(
    [sin(2*u)*(cos(v))^2,
    sin(u)*sin(2*v),
    cos(u)*sin(2*v)],
    u=-PI/2..PI/2,
    v=-PI/2..PI/2,
    FillColorFunction=Cl,
    Scaling=Constrained,
    UMesh=50,
    VMesh=50
):
plot(SteinerParam, Axes=None)
Error: An arithmetical expression is expected. [sin]
```

Kummer's surface

```
Kummer := x^2*y^2 + y^2*z^2 + z^2*x^2 - 1:
KummerColor := (x,y,z) -> [   max(0,sin(2*x)),    max(0,cos(2*y)),    max(0,sin(2*z))]:
KummerSurf := plot::Implicit3d(   Kummer, x = -3..3, y = -3..3, z = -3..3,    Mesh = [20,20,20],   FillColorFunction =
KummerColor,   MeshVisible = TRUE):
MyCam := plot::Camera([10,10,10],[0,0,0.3],PI/10):
plot(KummerSurf, MyCam,    Scaling = Constrained,    Axes = None)
```

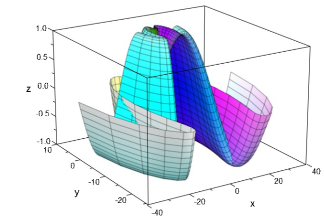


10 Spherical coordinates

```
p := x -> abs(5*x - trunc(5*x)):
Cl := (u,v,x,y,z)->[p(x*y),p(y*z),p(z*x)]:
Shell := plot::Spherical(
    [u*v,u,v],u=0..2*PI,v=0..PI,
    FillColorFunction=Cl
):
plot(
    Shell,
    Axes=Origin,
    AxesLineWidth=0.7,
    AxesTips=TRUE
)
Error: The argument '((x, y, z) -> (x - 1)^2 + (y - 1)^2 + (z - 1)^2) = 0..2*PI' is unexpected. [plot::Spherical::new]
```

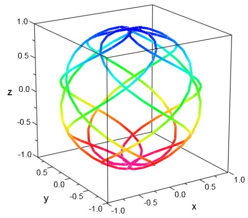
11 Cylindrical coordinates

```
Cl2:= (u,z) -> [abs(1-z/PI) ,abs(1-u/PI),abs(u/PI)]:
Hat := plot::Cylindrical(
    [t*v, sin(t), sin(v)],
    t=-2*PI..2*PI, v=0..2*PI,
    Scaling=Unconstrained,
    FillColorFunction=Cl2,
    UMesh=50,
    VMesh=50
):
plot(Hat)
```

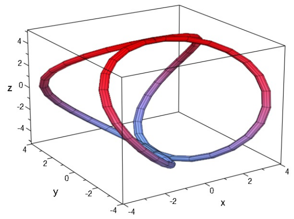


12 Curve3d and Tube

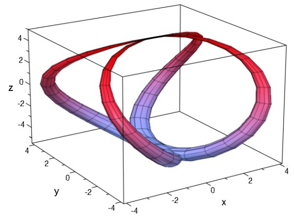
```
a := 5: b := 7: r := 1:
x := t -> r*sin(a*t)*cos(b*t):
y := t -> r*sin(a*t)*sin(b*t):
z := t -> r*cos(a*t):
C3D := plot::Curve3d([x(t),y(t),z(t)], t=0..2*PI, Mesh=200, LineWidth=0.8,LineColorType=Rainbow):
plot(C3D, Scaling = Constrained)
```



```
a := 7:
b := 3.5:
r := 5:
x := t -> r*sin(a*t)*cos(b*t):
y := t -> r*sin(a*t)*sin(b*t):
z := t -> r*cos(a*t):
T3D := plot::Tube([x(t),y(t),z(t)], 0.2, t=0..2*PI/b):
plot(T3D, Scaling=Unconstrained)
```



```
a := 7:
b := 3.5:
r := 5:
x := t -> r*sin(a*t)*cos(b*t):
y := t -> r*sin(a*t)*sin(b*t):
z := t -> r*cos(a*t):
T3D := plot::Tube([x(t),y(t),z(t)], sin(t*b)/2, t=0..2*PI/b):
plot(T3D, Scaling=Unconstrained)
```

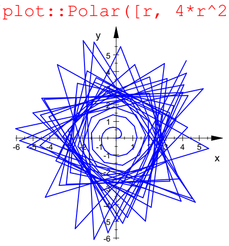


Refinement

In some cases, the default of 121 evaluations on the curve is not sufficient and causes visible artifacts:

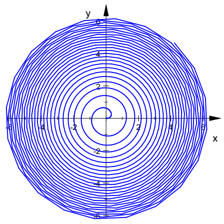
```
reset():
```

```
plot(plot::Polar([r, 4*r^2], r = 0..2*PI))
```



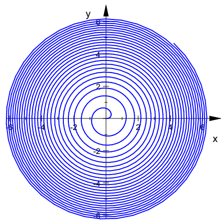
One remedy for this problem is to increase the number of evaluation points:

```
plot(plot::Polar([r, 4*r^2], r = 0..2*PI, Mesh = 1000))
```



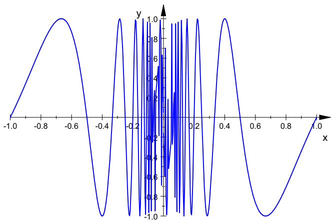
This method is, however, wasteful: Near the center, the initial density was perfectly sufficient, while on the outer edge still more points would be desirable. plot::Polar offers adaptive mesh refinement for exactly these situations. In the following example, we switch on adaptive mesh refinement with up to 2^4 = 16 points introduced between each two consecutive points of the initial mesh:

```
plot(plot::Polar([r, 4*r^2], r = 0..2*PI, AdaptiveMesh=4))
```



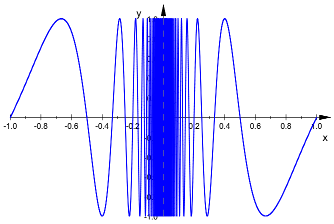
The standard mesh for the numerical evaluation of a function graph does not suffice to generate a satisfying graphics in the following case:

```
plot(plot::Function2d(sin(PI/x), x = -1 .. 1)):
```



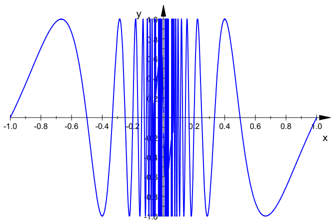
We increase the number of mesh points:

```
plot(plot::Function2d(sin(PI/x), x = -1 .. 1, XMesh = 10000)):
```



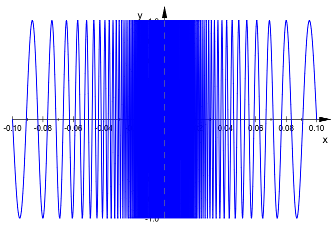
Alternatively, we enable adaptive sampling by setting AdaptiveMesh to some positive value:

```
plot(plot::Function2d(sin(PI/x), x = -1 .. 1, AdaptiveMesh = 10)):
```



Finally, we increase the XMesh value and use adaptive sampling:

```
plot(plot::Function2d(sin(PI/x), x = -0.1 .. 0.1, XMesh = 1000,AdaptiveMesh = 10)):
```

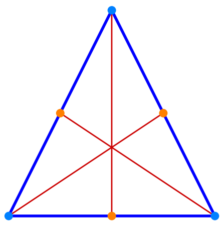


Plotting geometry models - Points, lines and polygons

```
// point coordinates
x1 := -1: x2 :=  1: x3 := 0:
y1 := -1: y2 := -1: y3 := 1:
```

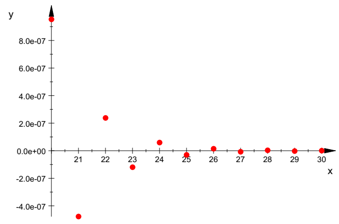
```
 // declarations of 2D points
point1 := plot::Point2d([x1,y1]):
```

```
point2 := plot::Point2d([x2,y2]):
point3 := plot::Point2d([x3,y3]):
points := plot::Group2d(
   point1, point2, point3,
   PointColor=[0,0.5,1],
   PointSize = 3*unit::mm
):
// sides of the triangle
side1 := plot::Line2d([x1,y1],[x2,y2]):
side2 := plot::Line2d([x2,y2],[x3,y3]):
side3 := plot::Line2d([x3,y3],[x1,y1]):
sides := plot::Group2d(
   side1, side2, side3, LineWidth=1
):
// midpoints of sides
midpoint1 := plot::Point2d([(x1+x2)/2,(y1+y2)/2]):
midpoint2 := plot::Point2d([(x2+x3)/2,(y2+y3)/2]):
midpoint3 := plot::Point2d([(x1+x3)/2,(y1+y3)/2]):
midpoints := plot::Group2d(
   midpoint1, midpoint2, midpoint3,
   PointColor=[1,0.5,0],
   PointSize = 3*unit::mm
):
// medians
median1 := plot::Line2d([x1,y1],[(x2+x3)/2,(y2+y3)/2]):
median2 := plot::Line2d([x2,y2],[(x1+x3)/2,(y1+y3)/2]):
median3 := plot::Line2d([x3,y3],[(x1+x2)/2,(y1+y2)/2]):
medians := plot::Group2d(
   median1, median2, median3,
   LineColor=[0.8,0,0],
   LineWidth=0.5
):
plot(sides, medians, midpoints,points,
   Scaling=Constrained,
   Axes=None
)
```

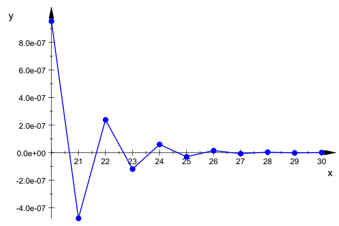


Points

```
// here we declare our sequence
data := [[n,1/(-2)^n] $ n=20..30]:
 // now we declare plot object for the sequence
sequence := plot::PointList2d(data,
   PointSize=2*unit::mm,
   PointColor=RGB::Red
):
 plot(sequence)
```



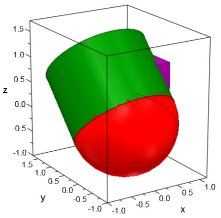

```
sequence2 := plot::Polygon2d(data,
  PointSize=2,
  PointsVisible=TRUE
):
 plot(sequence2)
```



Solids in MuPAD

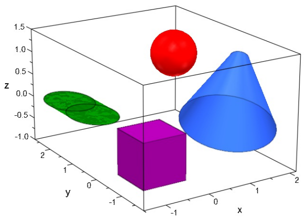
```
// declarations of four solids
RedSphere := plot::Sphere(1,[0,0,0],
   FillColor=[1,0,0]
):
```

```
GreenCylinder := plot::Cylinder(1,[0,0,0],[0,1,1],
   FillColor=[0,0.5,0]
):
PinkBox := plot::Box([0,0,0], [1,1,1],
   FillColor=[0.6, 0,0.6]
):
BlueCone := plot::Cone(1,[0,0,0], 0.2,[1,1,1]):
plot(RedSphere, GreenCylinder, BlueCone, PinkBox)
```

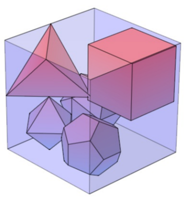


```
// transformations
A:=plot::Translate3d([1,1,1],
   plot::Scale3d([0.5,0.5,0.5],RedSphere)
):
B:=plot::Translate3d([-1,1,0],
   plot::Scale3d([0.5,1,0],GreenCylinder)
):
C:=plot::Translate3d([-1,-1,-1],PinkBox):
S:=plot::Translate3d([1,-1,0],BlueCone):
```

```
 // finally we plot translated solids
plot(A,B,C,S)
```

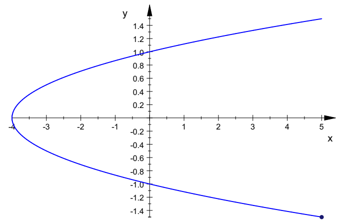


```
Bo := plot::Box([0,0,0],[4,4,4],
   FillColor=[0.7, 0.7, 1, 0.6]
):
He := plot::Hexahedron (
   Center = [2.9, 2.9, 2.9],
   Radius = 1
):
Te := plot::Tetrahedron (
   Center = [1, 1, 3],
   Radius = 1
):
Oc := plot::Octahedron (
   Center = [1, 1, 1],
   Radius = 1
):
Ic := plot::Icosahedron (
   Center = [1, 3, 1],
   Radius = 1
):
Do := plot::Dodecahedron(
  Center = [3, 1, 1],
  Radius = 1
):
Cam := plot::Camera(
  [28,-23,20],[1.5,1.5,1.5], PI/18
):
plot(Bo, He, Te, Oc, Ic, Do, Cam, Axes=None)
```

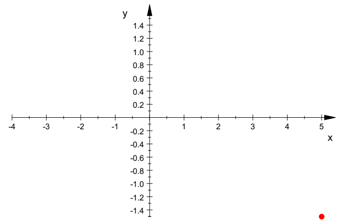


Animation step-by-step

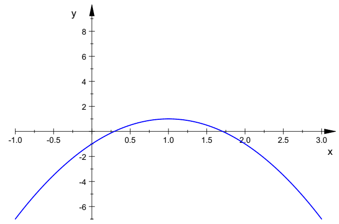
```
reset():
Curve := plot::Curve2d([t^2-4, t/2], t=-3..3):
Particle := plot::Point2d([a^2-4, a/2], a=-3..3):
plot(Curve, Particle)
```



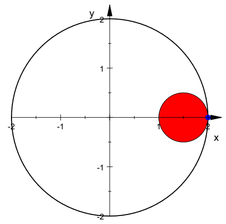

```
Curve := plot::Curve2d([t^2-4, t/2], t=-3..a, a=-3..3):
Particle := plot::Point2d([a^2-4, a/2], a=-3..3,
   PointSize=2,
   PointColor=RGB::Red
):
plot(Curve, Particle)
```



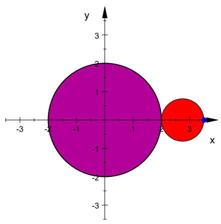

```
h := 1:
k := 1:
Parabola := plot::Function2d(
   a*(x-h)^2+k, x=-1..3, a=-2..2
):
plot(Parabola)
```




```
// radii of both circles
r := 0.5:
R := 2:

// coordinates of the particle
coordinates :=
   [(R-r)*cos(phi)+r*cos(((R-r)/r)*phi),
   (R-r)*sin(phi)-r*sin(((R-r)/r)*phi)]:

// static objects
LargeCircle := plot::Circle2d(R,
   LineColor=RGB::Black
):

// objects to animate
SmallCircle:= plot::Circle2d(
   r, [(R-r)*cos(a), (R-r)*sin(a)],
   a=0..2*PI,
   Color = [0.8,0,0],
   LineColor=RGB::Black,
   LineWidth=0.1,
   Filled=TRUE,
   FillPattern=Solid
):

Particle:=plot::Point2d(
   coordinates,
   phi = 0..2*PI,
   PointSize = 2,
   PointColor = [0,0,1]
):

Curve := plot::Curve2d(
   coordinates,
   phi =0..a, a=0..2*PI
):
// now we plot all together

plot(LargeCircle, SmallCircle, Particle, Curve,
   Frames=80,
   Scaling=Constrained,
   AxesInFront=TRUE
)
```

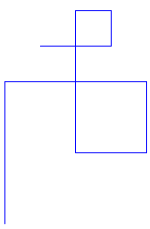


```
// define radius
r := 0.75:
R := 2:
// position of drawing point
Middle :=
   [(R+r)*cos(phi)+r*cos((1+R/r)*phi),
    (R+r)*sin(phi)+r*sin((1+R/r)*phi)]:
BigCircle := plot::Circle2d(R,
  LineColor=RGB::Black,
  Filled=TRUE,
  FillColor=[0.7,0,0.6],
  FillPattern=Solid
):
SmallCircle:= plot::Circle2d(
  r, [(R+r)*cos(phi), (R+r)*sin(phi)],
  phi=0..6*PI,
  Color = [0.8,0,0],
  LineColor=RGB::Black,
  LineWidth=0.1,
  Filled=TRUE,
  FillPattern=Solid
):
Point:=plot::Point2d(
  Middle,
  phi = 0..6*PI,
  PointSize = 2,
  PointColor = [0,0,1]
):
Curve := plot::Curve2d(
  Middle,
  phi = 0..a, a=0..6*PI
):
// plotting all together
plot(BigCircle, SmallCircle, Point, Curve,
  Frames=100,
  Scaling=Constrained,
  AxesInFront=TRUE
)
```

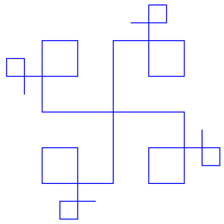


Turtle graphics and L-systems

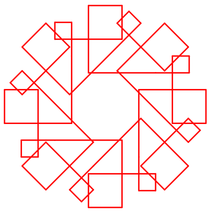
```
T:=plot::Turtle():
T::forward(100): T::right(PI/2):
T::forward(100): T::right(PI/2):
T::forward(50):  T::right(PI/2):
T::forward(50):  T::right(PI/2):
T::forward(100): T::right(PI/2):
T::forward(25):  T::right(PI/2):
T::forward(25):  T::right(PI/2):
T::forward(50):
plot(T)
```



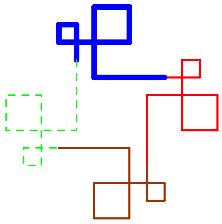
```
T2 := plot::Rotate2d(PI/2,[0,0],T):
T3 := plot::Rotate2d(PI,[0,0],T):
T4 := plot::Rotate2d(3*PI/2,[0,0],T):
plot(T,T2,T3,T4)
```



```
A1 := plot::Translate2d([50,-75], T):
A2 := plot::Rotate2d(  PI/4,[0,0],A1):
A3 := plot::Rotate2d(2*PI/4,[0,0],A1):
A4 := plot::Rotate2d(3*PI/4,[0,0],A1):
A5 := plot::Rotate2d(4*PI/4,[0,0],A1):
A6 := plot::Rotate2d(5*PI/4,[0,0],A1):
A7 := plot::Rotate2d(6*PI/4,[0,0],A1):
A8 := plot::Rotate2d(7*PI/4,[0,0],A1):
plot(A1,A2,A3,A4,A5,A6,A7,A8,
   LineColor=RGB::Red,
   LineWidth=0.5
)
```

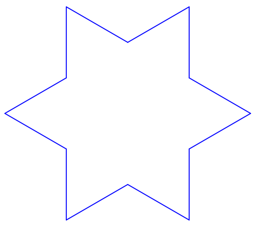


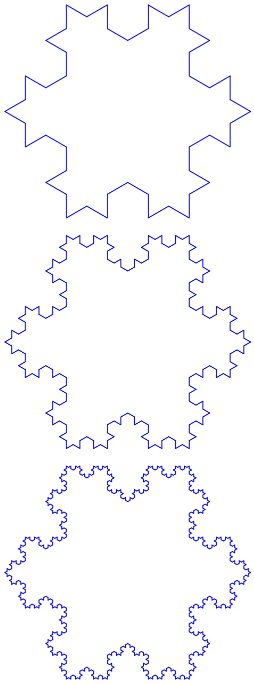
```
A1 := plot::modify(T, LineColor=RGB::Red):
A1 := plot::Translate2d([50,-75], A1):
A3 := plot::modify(T, LineColor=RGB::Blue, LineWidth=2):
A3 := plot::Translate2d([50,-75], A3):
A3 := plot::Rotate2d(2*PI/4,[0,0],A3):
A5 := plot::modify(T,
   LineColor=RGB::Green,
   LineStyle=Dashed,
   LineWidth=0.5
):
A5 := plot::Translate2d([50,-75], A5):
A5 := plot::Rotate2d(4*PI/4,[0,0],A5):
A7 := plot::modify(T, LineColor=RGB::Brown):
A7 := plot::Translate2d([50,-75], A7):
A7 := plot::Rotate2d(6*PI/4,[0,0],A7):
plot(A1,A3,A5,A7, LineWidth=0.75)
```



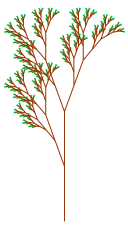
Lindenmayer systems

```
KochCurve := proc(n)
begin
   plot::Lsys( // start a new L-system
   PI/3,              // turtle must always turn PI/3
   "F++F++F",         // this is the seed
   "F"="F-F++F-F",    // iteration rule
   Generations=n      // number of generations
    ):
end:
plot(KochCurve(1)):      // now plot the turtle path
plot(KochCurve(2)):      // now plot the turtle path
plot(KochCurve(3)):      // now plot the turtle path
plot(KochCurve(4)):      // now plot the turtle path
```





```
plot(plot::Lsys(PI/9, "BL", "L" = "BR[+HL]BR[-GL]+HL",
                "R" = "RR", "L" = Line, "R" = Line,
                "B" = RGB::Brown, "H" = RGB::ForestGreen,
                "G" = RGB::SpringGreen, Generations = 6)):
```



```
plot(plot::Lsys(PI/3, "R", "L" = "R+L+R", "R" = "L-R-L",
                         "L" = Line, "R" = Line,
                Generations = 7)):
```