6.2 Drawing Three-Dimensional Objects

2015 – 04 – 21

```
$Version
```

```
10.0 for Mac OS X x86 (64-bit) (September 10, 2014)
```

1. Curves in space may be drawn in perspective. **(ParametricPlot3D,ListPointPlot3D).**

2. **2-dimensional surfaces** extending through 3-dimensional space may be represented in several ways.

2.1 They may be drawn in perspective. **(ParametricPlot3D, Plot3D, ListSurfacePlot3D)**

2.2 Lines of equal height may be projected on a plane **(ContourPlot).**

3.2 The height of each point above a plane upon which the surface is projected is indicated by a corresponding shade of gray (or of colour) **(DensityPlot).**

6.2.1 Plotting Curves in Parametric Representation in 3-Dimensional Space

The general form of the statement is:

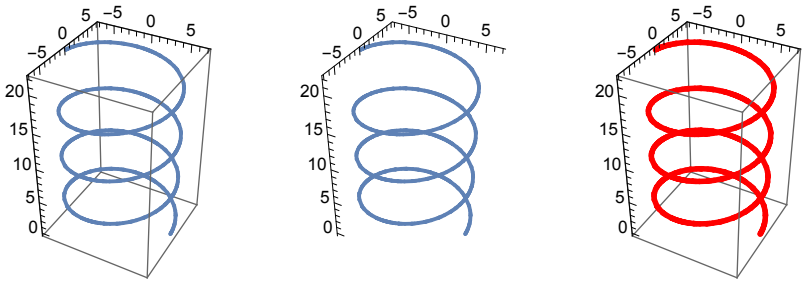
```
ParametricPlot3D[ {x[t],y[t],z[t]}, {t, tmin, tmax},options ]
```

The options are discussed in § 6.2.6.

```
? ParametricPlot3D
```

ParametricPlot3D[{f_x, f_y, f_z}, {u, u_{min}, u_{max}}] produces a three-dimensional space curve parametrized by a variable u which runs from u_{min} to u_{max}.
ParametricPlot3D[{f_x, f_y, f_z}, {u, u_{min}, u_{max}}, {v, v_{min}, v_{max}}] produces a three-dimensional surface parametrized by u and v.
ParametricPlot3D[{{f_x, f_y, f_z}, {g_x, g_y, g_z} ...} ...] plots several objects together. ≫

```
x = 7 Cos[t]; y = 7 Sin[t]; z = t;
p1 = ParametricPlot3D[{x, y, z}, {t, 0, 7 π}];
p2 = Show[p1, Boxed → False];
p3 = ParametricPlot3D[{x, y, z}, {t, 0, 7 π}, PlotStyle → {Red, Thickness[0.03`]}];
Show[GraphicsRow[{p1, p2, p3}], ImageSize → 450]
```



6.2.2 Plotting Curves Given in 3-Dimensional Space by Points: ListPointPlot3D[]

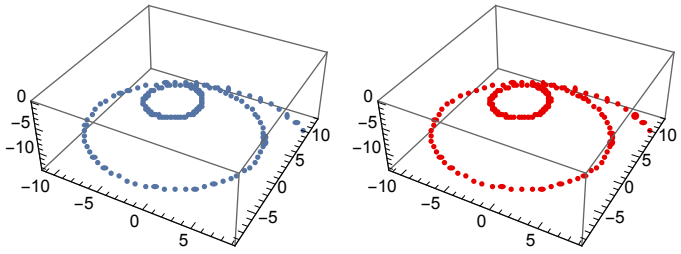
Curves may be given by lists of points. They may be drawn by the comand

```
ListPointPlot3D[list]
```

```
? ListPointPlot3D
```

ListPointPlot3D[{{x_1, y_1, z_1}, {x_2, y_2, z_2}, ...}] generates a 3D scatter plot of points with coordinates {x_i, y_i, z_i}.
ListPointPlot3D[*array*] generates a 3D scatter plot of points with a 2D array of height values.
ListPointPlot3D[{$data_1$, $data_2$, ...}] plots several collections of points, by default in different colors. ≫

```
lt = Table[t {Sin[t], Cos[t], -1}, {t, 0, 13.3`, 0.1`}];
p1 = ListPointPlot3D[lt];
p2 = ListPointPlot3D[lt, PlotStyle → RGBColor[0.9`, 0, 0]];
Show[GraphicsRow[{p1, p2}], ImageSize → 350]
```



6.2.3 Plotting Surfaces in 3-dimensional Space in Perspective: ParametricPlot3D[], Plot3D[]

Surfaces may be represented in various mathematical forms:

1. The coordinates are given as functions of two parameters:

```
x = x[u,v], y = y[u,v],  z = z[u,v].
```

The form of the corresponding statement is:

```
ParametricPlot3D[ {x[u,v],y[u,v],z[u,v]},
          {u, umin, umax}, {v, vmin, vmax}, Options ]
```

```
a = 3 + Cos[u]; x = a Cos[v]; y = a Sin[v]; z = Sin[u];
p1 = ParametricPlot3D[{x, y, z}, {u, 0, 2 π}, {v, 0, 2 π}];
```

2. The third coordinate is given as a function of the two other coordinates:

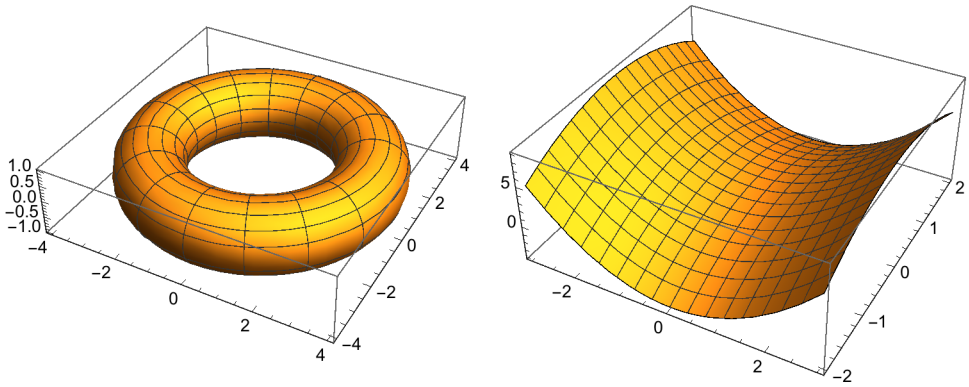
```
z = f[x,y]
```

The general form of the corresponding statement is:

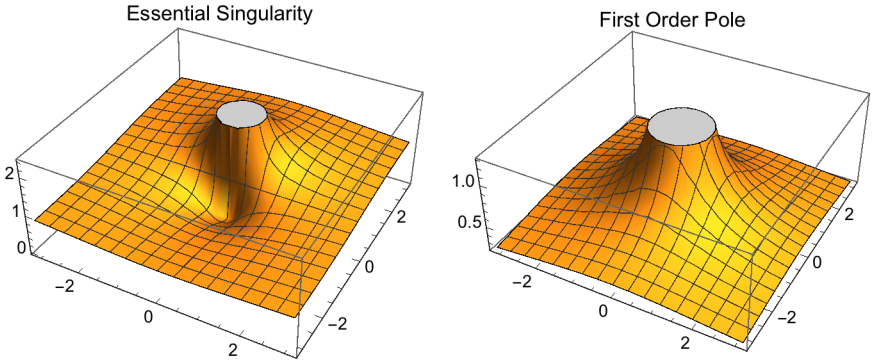
```
Plot3D[ f[x,y], {x, xmin, xmax}, {y, ymin, ymax}, Options]
```

```
Clear[x, y, f];     f = x^2 - y^2;
p2 = Plot3D[ f, {x, -3, 3}, {y, -2, 2} ];
```

```
Show[GraphicsRow[{p1, p2}], ImageSize → 500]
```



```
p1 = Plot3D[Abs[Exp[ⅈ/(x + ⅈ y)]], {x, -π, π},
    {y, -π, π}, PlotLabel → "Essential Singularity"];
p2 = Plot3D[Abs[ⅈ/(x + ⅈ y)], {x, -π, π}, {y, -π, π}, PlotLabel → "First Order Pole"];
Show[GraphicsRow[{p1, p2}], ImageSize → 450]
```



The preceeding plots show the relief (= the surface of the absolute value of the function) of an essential singularitly and of a pole.

6.2.3.1 Plotting a surface from points: ListSurfacePlot3D[]

```
? ListSurfacePlot3D
```

ListSurfacePlot3D[{{x_1, y_1, z_1}, {x_2, y_2, z_2}, ...}]
plots a three-dimensional surface constructed to fit the specified points. ≫

```
p1 = ListSurfacePlot3D[Flatten[Table[
      {Cos[ϕ] Sin[θ], Sin[θ] Sin[ϕ], Cos[θ]}, {ϕ, -π, π, .2}, {θ, 0, π, .2}], 1]];
p2 = ListSurfacePlot3D[Flatten[Table[{x, y, Sin[x y]},
      {x, 0, 3, 0.1}, {y, 0, 3, 0.1}], 1]] ;
p3 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}];
GraphicsRow[{p1, p2, p3}, ImageSize → 500]
```

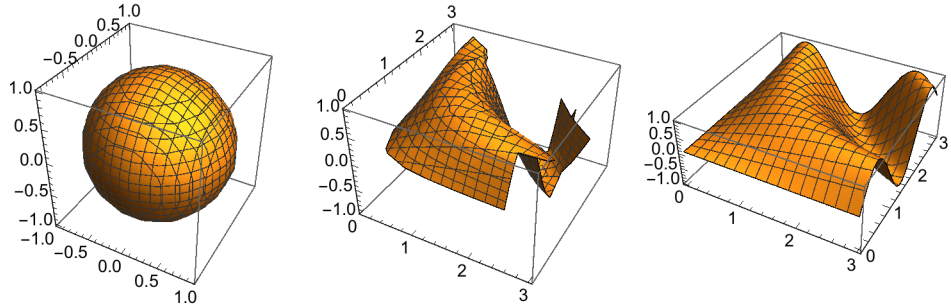


Figure **p2** (the middle figure) above shows that **`ListSurfacePlot3D[]`** may yield unsatisfactory images as compared to the output of **`Plot3D[]`**. Try this again for a step width of 0.05 !

6.2.4 Contour Plots: Plotting Contour Lines of Surfaces: ContourPlot[]

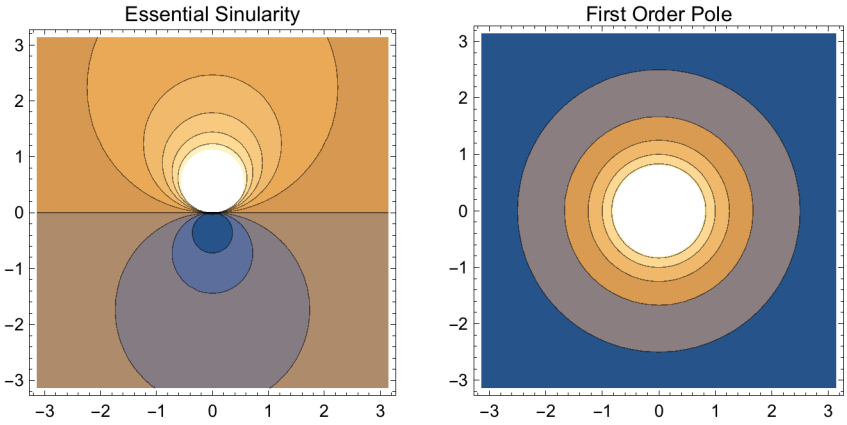
```
ContourPlot[  f[x,y], {x, xmin, xmax}, {y, ymin, ymax} ]
```

The comand produces contour lines as in a (geographical) map of the surface z = f(x,y) . The shade of the area between two curves on the map is the lighter the higher the level of the corresponsing slice of the surface.

```
? ContourPlot
```

ContourPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a contour plot of f as a function of x and y.
ContourPlot[f == g, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots contour lines for which f = g.
ContourPlot[{f_1 == g_1, f_2 == g_2, …}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several contour lines. »

```
p1 = ContourPlot[Abs[Exp[i/(x + i y)]], {x, -π, π},
    {y, -π, π}, PlotLabel → "Essential Sinularity"];
p2 = ContourPlot[Abs[i/(x + i y)], {x, -π, π}, {y, -π, π},
    PlotLabel → "First Order Pole"];
Show[GraphicsRow[{p1, p2}], ImageSize → 450]
```



6.2.5 Density Plots: Expressing the Level as Gray Density or Color: DensityPlot[]

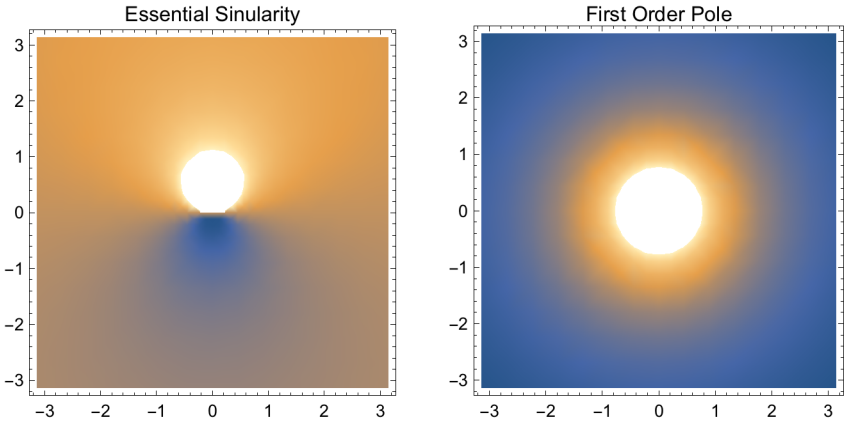
```
DensityPlot[  f[x,y], {x, xmin, xmax}, {y, ymin, ymax} ]
```

The shade of a pixel is related to the level of the corresponding surface element above it. The shade is the lighter the higher the level.

```
? DensityPlot
```

DensityPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] makes a density plot of f as a function of x and y. ≫

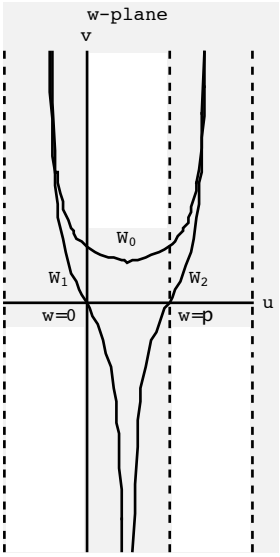
```
p1 = DensityPlot[Abs[Exp[i/(x + i y)]], {x, -π, π},
    {y, -π, π}, PlotLabel → "Essential Sinularity"];
p2 = DensityPlot[Abs[i/(x + i y)], {x, -π, π}, {y, -π, π},
    PlotLabel → "First Order Pole"];
Show[GraphicsRow[{p1, p2}], ImageSize → 450]
```



6.2.6 Comparing Diagrams

6.2.6.1 Integral Representations of Bessel Functions. The Method of Steepest Descent

Integral representations of different Bessel functions. The integrand is always the same.
Different integration paths lead to different functions.



$$Z_n(r) = (1/2\pi i) \int_{Wi} e^{ir\,Cos(w)}\, e^{in\,(w - \pi/2)}\, dw$$

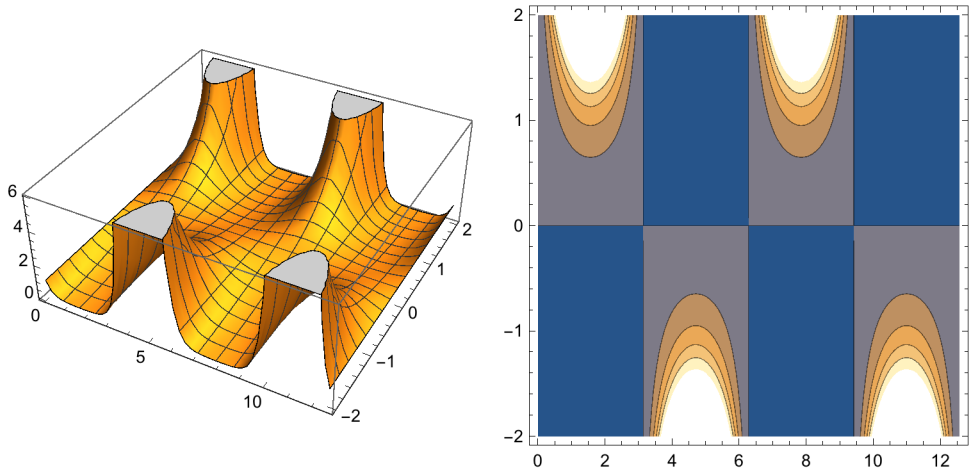
W_0 : $J_n(r)$ Bessel function of the first kind

$W_{1,2}$: $H_n^{(1,2)}(r)$ Hankel function of the 1 st, 2 nd kind.

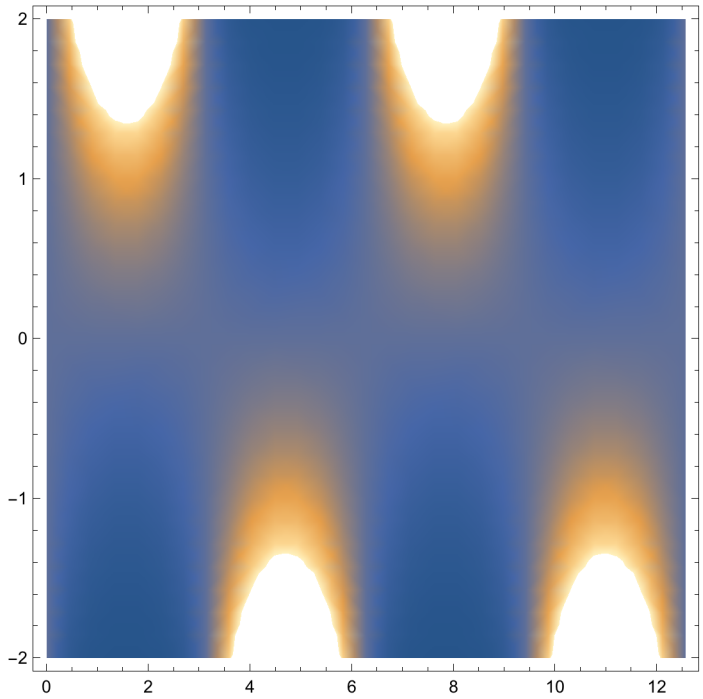
A. Sommerfeld: Partielle Differentialgleichungen der Physik.
Akademische Verlagsgesellschaft, Leipzig 1958. § 19, Fig.18.

Order n = 0, r = real: Integrand = $e^{ir\cos(z)}$, z = x + I y.

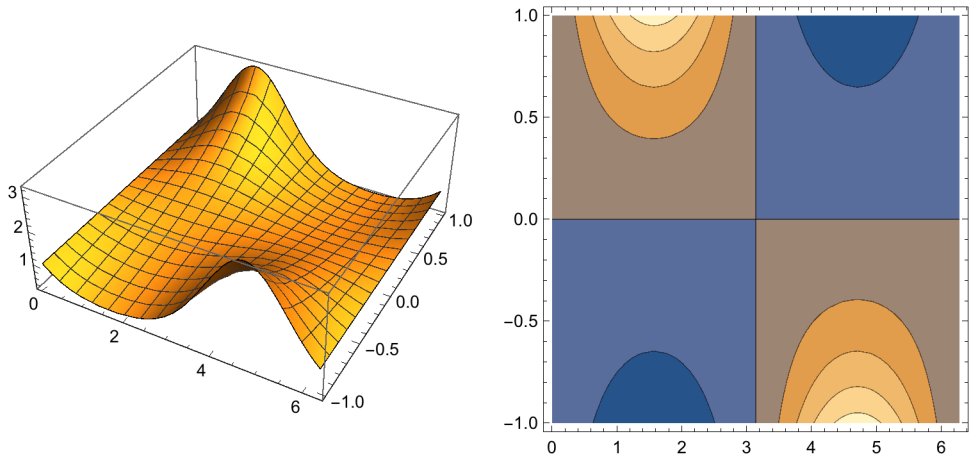
```
Clear[f, r, x, y, z]
f[x_, y_, r_] = Abs[Exp[ I r Cos[x + I y] ] ];
s1 = Plot3D[ f[x, y, 1], {x, 0, 4 Pi}, {y, -2, 2}];
c1 = ContourPlot[ f[x, y, 1], {x, 0, 4 Pi}, {y, -2, 2}] ;
Show[GraphicsRow[{s1, c1}], ImageSize → 500]
```




```
d1 = DensityPlot[ f[x, y, 1], {x, 0, 4 Pi}, {y, -2, 2}]
```

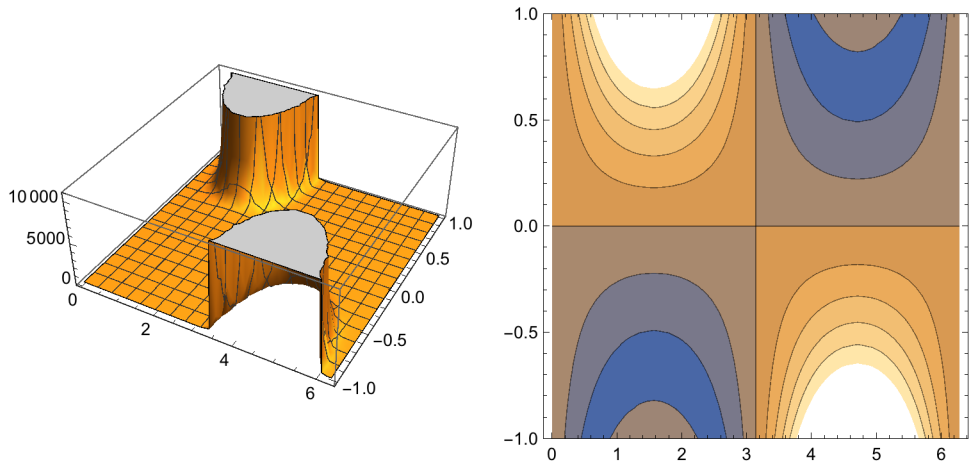


```
s2 = Plot3D[f[x, y, 1], {x, 0, 2 π}, {y, -1, 1}];
c2 = ContourPlot[f[x, y, 1], {x, 0, 2 π}, {y, -1, 1}];
Show[GraphicsRow[{s2, c2}], ImageSize → 500]
```

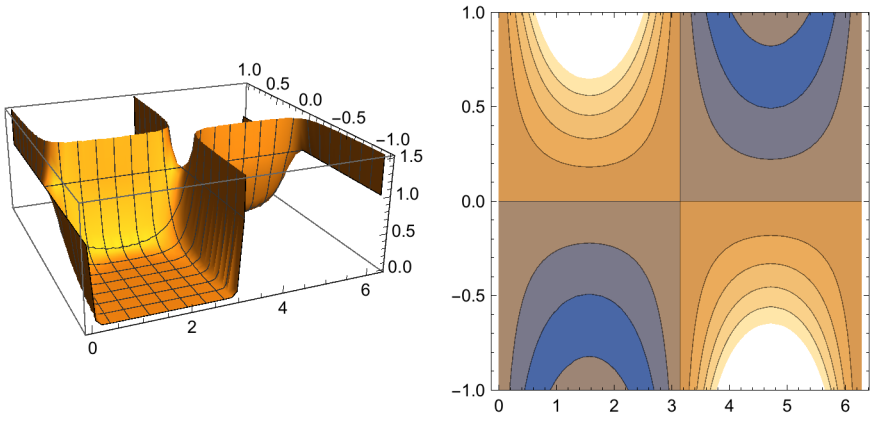


The working of the method of steepest descent (Sattelpunktsmethode) applied to the integrals above for large argument **r** can be understood in looking at the surfaces representing the modulus of the integrand.

```
s20 = Plot3D[f[x, y, 20], {x, 0, 2 π}, {y, -1, 1}];
s21 = ContourPlot[f[x, y, 1], {x, 0, 2 π}, {y, -1, 1}, PlotRange → {0, 2}];
Show[GraphicsRow[{s20, s21}], ImageSize → 500]
```

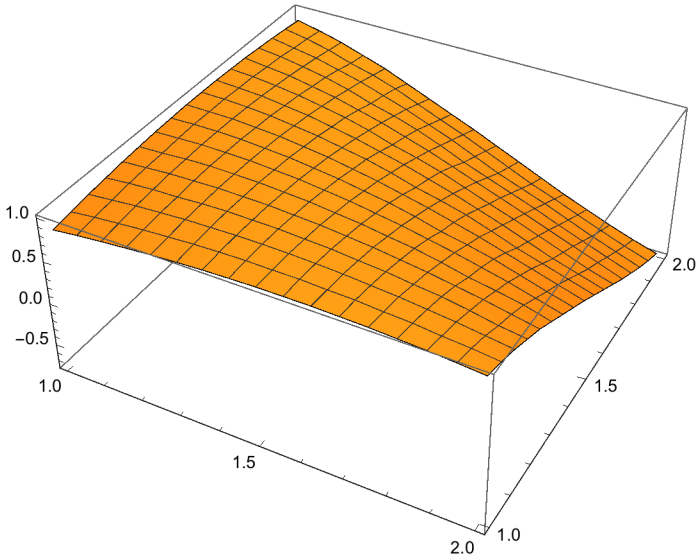


```
s22 = Show[s20, PlotRange → {0, 1.5}, ViewPoint → {-1., -2.4, 1.}];
s23 = Show[s21, AspectRatio → 1];
Show[GraphicsRow[{s22, s23}], ImageSize → 450]
```



6.2.6.2 Comparison between Plot3D[] and ParametricPlot3D[]

```
pp = Plot3D[Sin[x y], {x, 1, 2}, {y, 1, 2}, PlotPoints → 5]
```

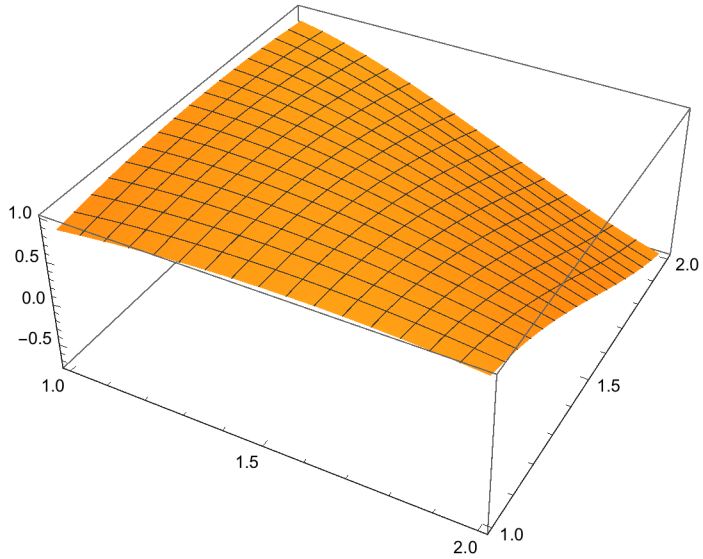



```
FullForm[pp]
```

```
Graphics3D[
 GraphicsComplex[List[List[1.00000025`, 1.00000025`, 0.841471254958978`],
   List[1.2500001250000001`, 1.00000025`, 0.9489847573090389`], ··· 861 ···,
   List[1.2500001250000004`, 1.0625002187500001`, 0.9677387947245041`],
   List[1.3750000625000005`, 1.0625002187500001`, 0.9939716260641457`]],
  ··· 1 ···, Rule[ ··· 1 ···]], ··· 1 ···]
```

large output | show less | show more | show all | set size limit...

```
pa = ParametricPlot3D[{x, y, Sin[x y]},
  {x, 1, 2}, {y, 1, 2}, PlotPoints → 4, BoxRatios → {1, 1, 0.4`}]
```




```
FullForm[pa]
```

```
Graphics3D[GraphicsComplex[
  List[List[1.0000003333333334`, 1.0000003333333334`, 0.8414713450093068`],
   List[1.3333334444444445`, 1.0000003333333334`, 0.9719380320507122`],
   ··· 793 ···, List[1.1875002083333337`, 1.7499998333333346`,
    0.874045392136197`]], List[List[ ··· 1 ··· ], ··· 1 ···],
  Rule[VertexNormals, List[ ··· 1 ··· ]]], ··· 1 ···]
```

large output | show less | show more | show all | set size limit...

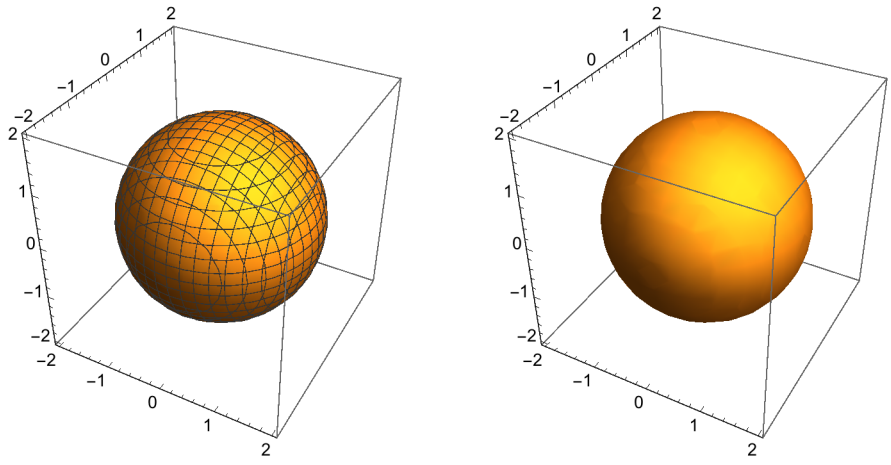
6.2.7 RegionPlot3D

```
?? RegionPlot3D
```

RegionPlot3D[*pred*, {*x*, x_{min}, x_{max}}, {*y*, y_{min}, y_{max}}, {*z*, z_{min}, z_{max}}]
makes a plot showing the three-dimensional region in which *pred* is True. ≫

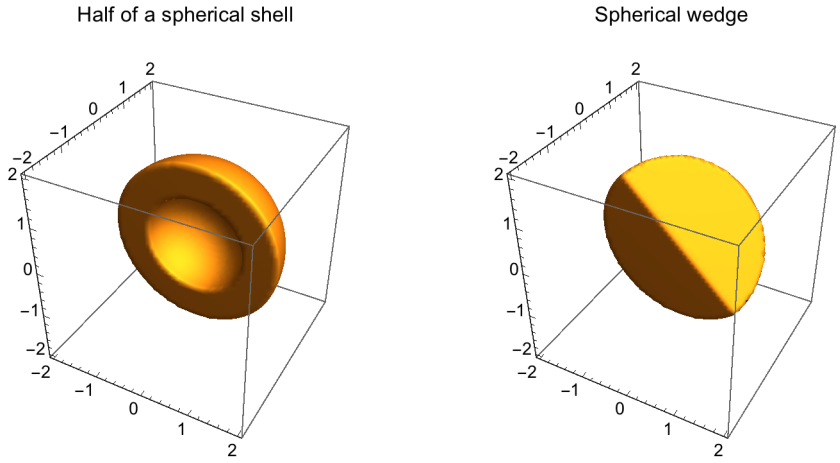
```
Attributes[RegionPlot3D] = {HoldAll, Protected, ReadProtected}

Options[RegionPlot3D] =
 {AlignmentPoint → Center, AspectRatio → Automatic, AutomaticImageSize → False,
  Axes → True, AxesEdge → Automatic, AxesLabel → None, AxesOrigin → Automatic,
  AxesStyle → {}, Background → None, BaselinePosition → Automatic, BaseStyle → {},
  BoundaryStyle → GrayLevel[0], Boxed → True, BoxRatios → {1, 1, 1}, BoxStyle → {},
  ClipPlanes → None, ClipPlanesStyle → Automatic, ColorFunction → Automatic,
  ColorFunctionScaling → True, ColorOutput → Automatic, ContentSelectable → Automatic,
  ControllerLinking → Automatic, ControllerMethod → Automatic,
  ControllerPath → Automatic, CoordinatesToolOptions → Automatic,
  DisplayFunction :→ $DisplayFunction, Epilog → {}, Evaluated → Automatic,
  EvaluationMonitor → None, FaceGrids → None, FaceGridsStyle → {},
  FormatType :→ TraditionalForm, ImageMargins → 0., ImagePadding → All,
  ImageSize → Automatic, ImageSizeRaw → Automatic, LabelStyle → {},
  Lighting → Automatic, MaxRecursion → Automatic, Mesh → Automatic,
  MeshFunctions → {#1 &, #2 &, #3 &}, MeshShading → None, MeshStyle → Automatic,
  Method → Automatic, NormalsFunction → Automatic, PerformanceGoal :→ $PerformanceGoal,
  PlotLabel → None, PlotLegends → None, PlotPoints → Automatic, PlotRange → Full,
  PlotRangePadding → Automatic, PlotRegion → Automatic, PlotStyle → Automatic,
  PlotTheme :→ $PlotTheme, PreserveImageOptions → Automatic, Prolog → {},
  RotationAction → Fit, SphericalRegion → False, TargetUnits → Automatic,
  TextureCoordinateFunction → Automatic, TextureCoordinateScaling → Automatic,
  Ticks → Automatic, TicksStyle → {}, TouchscreenAutoZoom → False,
  ViewAngle → Automatic, ViewCenter → Automatic, ViewMatrix → Automatic,
  ViewPoint → {1.3, -2.4, 2.}, ViewRange → All, ViewVector → Automatic,
  ViewVertical → {0, 0, 1}, WorkingPrecision → MachinePrecision}
```

```
p1 = RegionPlot3D[x^2 + y^2 + z^2 ≤ 3, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}];

p2 =
  RegionPlot3D[x^2 + y^2 + z^2 ≤ 3, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}, Mesh → None];

Show[GraphicsRow[{p1, p2}], ImageSize → 490]
```




```
p1 = RegionPlot3D[1 ≤ x^2 + y^2 + z^2 ≤ 3 && y ≥ 0, {x, -2, 2}, {y, -2, 2}, {z, -2, 2},
    Mesh → None, PlotPoints → 50, PlotLabel → "Half of a spherical shell\n"];

p2 = RegionPlot3D[x^2 + y^2 + z^2 ≤ 3 && y ≥ 0 && y ≥ x + z, {x, -2, 2}, {y, -2, 2},
    {z, -2, 2}, Mesh → None, PlotPoints → 50, PlotLabel → "Spherical wedge\n"];
```

```
Show[GraphicsRow[{p1, p2}], ImageSize → 490]
```



6.2.8 Options for Perspectivic Plots

```
curve = {x[t], y[t], z[t]};
ParametricPlot3D[ curve, {t, tmin, tmax} ]
surfa = {x[u,v], y[u,v], z[u,v]};
ParametricPlot3D[ surfa, {u, umin, umax}, {v, vmin, vmax} ]
z = z[x,y];
Plot3D[ z, {x, xmin, xmax}, {y, ymin, ymax} ]
```

```
??Plot3D
```

Plot3D[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a three-dimensional plot of f as a function of x and y.
Plot3D[{f_1, f_2, ...}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several functions. ≫

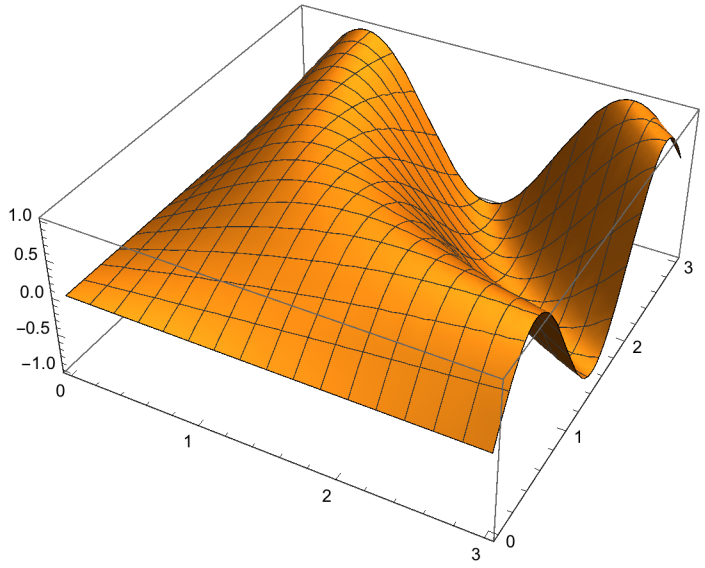
```
Attributes[Plot3D] = {HoldAll, Protected, ReadProtected}

Options[Plot3D] = {AlignmentPoint → Center, AspectRatio → Automatic,
  AutomaticImageSize → False, Axes → True, AxesEdge → Automatic,
  AxesLabel → None, AxesOrigin → Automatic, AxesStyle → {}, Background → None,
  BaselinePosition → Automatic, BaseStyle → {}, BoundaryStyle → GrayLevel[0],
  Boxed → True, BoxRatios → {1, 1, 0.4}, BoxStyle → {}, ClippingStyle → Automatic,
  ClipPlanes → None, ClipPlanesStyle → Automatic, ColorFunction → Automatic,
  ColorFunctionScaling → True, ColorOutput → Automatic, ContentSelectable → Automatic,
  ControllerLinking → Automatic, ControllerMethod → Automatic,
  ControllerPath → Automatic, CoordinatesToolOptions → Automatic,
  DisplayFunction :→ $DisplayFunction, Epilog → {}, Evaluated → Automatic,
  EvaluationMonitor → None, Exclusions → Automatic, ExclusionsStyle → None,
  FaceGrids → None, FaceGridsStyle → {}, Filling → None, FillingStyle → Opacity[0.5],
  FormatType :→ TraditionalForm, ImageMargins → 0., ImagePadding → All,
  ImageSize → Automatic, ImageSizeRaw → Automatic, LabelStyle → {}, Lighting → Automatic,
  MaxRecursion → Automatic, Mesh → Automatic, MeshFunctions → {#1 &, #2 &},
  MeshShading → None, MeshStyle → Automatic, Method → Automatic,
  NormalsFunction → Automatic, PerformanceGoal :→ $PerformanceGoal, PlotLabel → None,
  PlotLegends → None, PlotPoints → Automatic, PlotRange → {Full, Full, Automatic},
  PlotRangePadding → Automatic, PlotRegion → Automatic, PlotStyle → Automatic,
  PlotTheme :→ $PlotTheme, PreserveImageOptions → Automatic, Prolog → {},
  RegionFunction → (True &), RotationAction → Fit, SphericalRegion → False,
  TargetUnits → Automatic, TextureCoordinateFunction → Automatic,
  TextureCoordinateScaling → Automatic, Ticks → Automatic, TicksStyle → {},
  TouchscreenAutoZoom → False, ViewAngle → Automatic, ViewCenter → Automatic,
  ViewMatrix → Automatic, ViewPoint → {1.3, -2.4, 2.}, ViewRange → All,
  ViewVector → Automatic, ViewVertical → {0, 0, 1}, WorkingPrecision → MachinePrecision}
```

```
?ParametricPlot3D
```

ParametricPlot3D[{f_x, f_y, f_z}, {u, u_{min}, u_{max}}] produces a
three-dimensional space curve parametrized by a variable u which runs from u_{min} to u_{max}.
ParametricPlot3D[{f_x, f_y, f_z}, {u, u_{min}, u_{max}}, {v, v_{min}, v_{max}}] produces a
three-dimensional surface parametrized by u and v.
ParametricPlot3D[{{f_x, f_y, f_z}, {g_x, g_y, g_z} ...} ...] plots several objects together. ≫

The options of ParametricPlot3D are nearly the same as those of Plot3D

```
Clear[x, y, z]
z = Sin[x y];
pp = Plot3D[z, {x, 0, 3}, {y, 0, 3}]
```



```
? AspectRatio
```

AspectRatio is an option for Graphics and
related functions that specifies the ratio of height to width for a plot. ≫

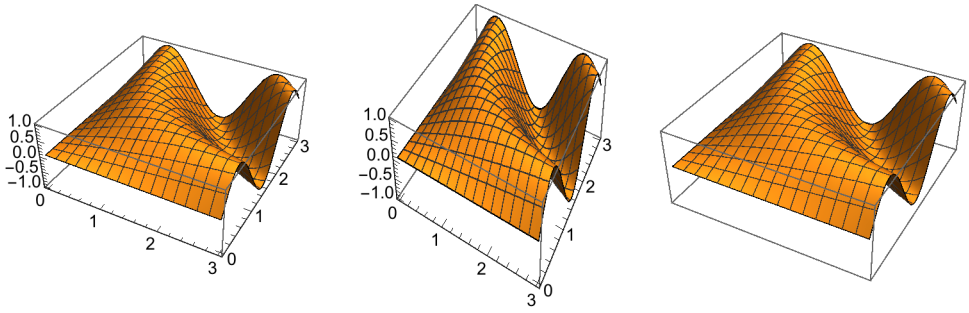
See also **BoxRatios** .

```
p1 = Show[pp, AspectRatio → Automatic];
p2 = Show[pp, AspectRatio → 1.3`];
```

```
? Axes
```

Axes is an option for graphics functions that specifies whether axes should be drawn. ≫

```
p3 = Show[pp, Axes → None]; Show[GraphicsRow[{p1, p2, p3}], ImageSize → 500]
```



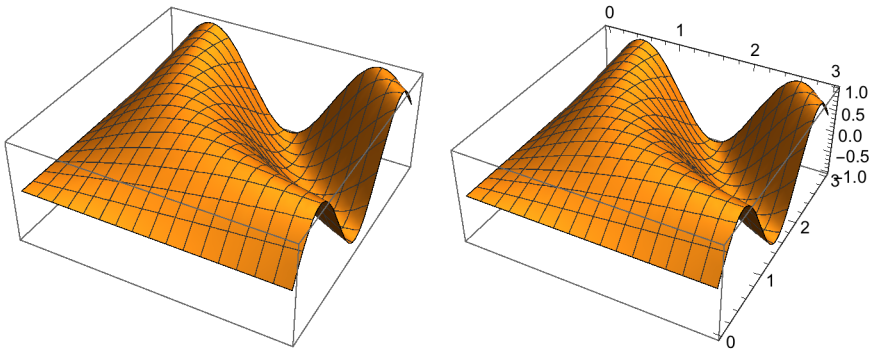
```
? AxesEdge
```

AxesEdge is an option for three-dimensional graphics functions
that specifies on which edges of the bounding box axes should be drawn. ≫

```
AxesEdge -> {{diry,dirz}, {dirx,dirz}, {dirx,diry}}
dirk = + 1 or - 1.
```

Any pair may be replaced with Automatic or None. The entire list may be replaced with a single Automatic or a single None.

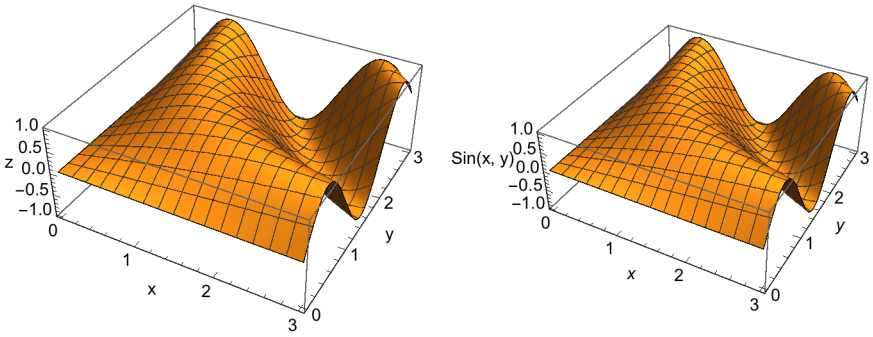
```
p1 = Show[pp, AxesEdge → None];
p2 = Show[pp, AxesEdge → {{1, 1}, {1, -1}, {1, 1}}];
Show[GraphicsRow[{p1, p2}], ImageSize → 450]
```



```
? AxesLabel
```

AxesLabel is an option for graphics functions that specifies labels for axes. ≫

```
z = "Sin(x, y)  ";
p1 = Show[pp, AxesLabel → {"x", "y", "z   "}];
p2 = Show[pp, AxesLabel → {x, y, z}]; Show[GraphicsRow[{p1, p2}], ImageSize → 450]
```




```
? AxesOrigin
```

AxesOrigin is an option for graphics functions that specifies where any axes drawn should cross. ≫

See same option in 2D case, 6.1.9 .

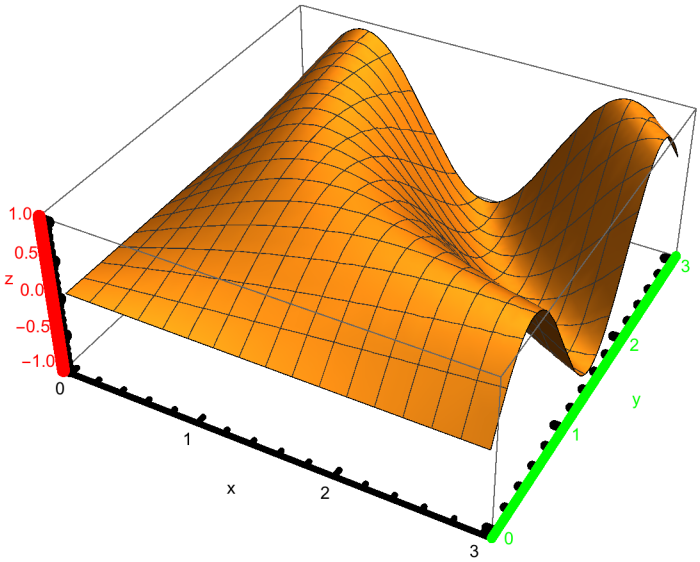
```
? AxesStyle
```

AxesStyle is an option for graphics functions that specifies how axes should be rendered. ≫

AxesStyle -> style All axes are rendered according to same style directive.
AxesStyle -> {{xstyle}, {ystyle}, {zstyle}}

Style directives: **Dashing[], Thickness[], AbsoluteThickness[];** see § 6.5.2.
Hue; RGBColor, CMYKColor; see § 6.5.3.

```
Show[pp, AxesStyle →
  {{Thickness[0.01`]}, {Thickness[0.015`], Green}, {Thickness[0.02`], Red}},
 AxesLabel → {"x", "y", "z"}]
```



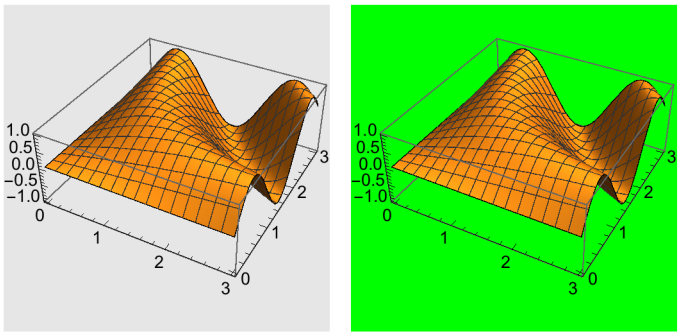
```
? Background
```

Background is an option that specifies what background color to use. ≫

```
Background is an option which specifies the background color to use. More...
```

Style directives: **GrayLevel[], Hue[]; RGBColor[], CMYKColor[].**

```
p1 = Show[pp, Background → GrayLevel[0.9`]];
p2 = Show[pp, Background → RGBColor[0, 1, 0]];
Show[GraphicsRow[{p1, p2}], ImageSize → 350]
```

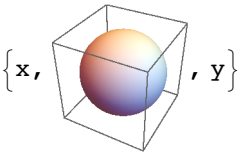



```
?? BaselinePosition
```

BaselinePosition is an option that specifies where the baseline of an object is
considered to be for purposes of alignment with surrounding text or other expressions. ≫

```
Attributes[BaselinePosition] = {Protected}
```

```
{x, Graphics3D[Sphere[], BaselinePosition → Center,
  BaselinePosition → Center, ImageSize → 70], y}
```

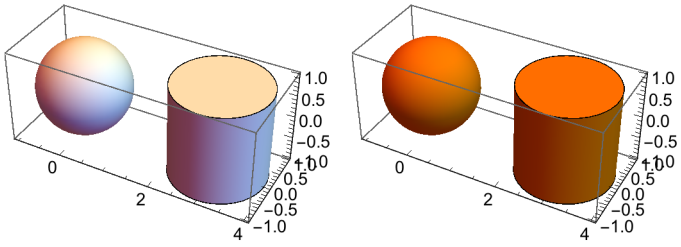


Confer the same command, **BaselinePosition**, in the 2D-case in 6.1.9.

```
? BaseStyle
```

BaseStyle is an option for formatting and related constructs that specifies the base style to use for them. ≫

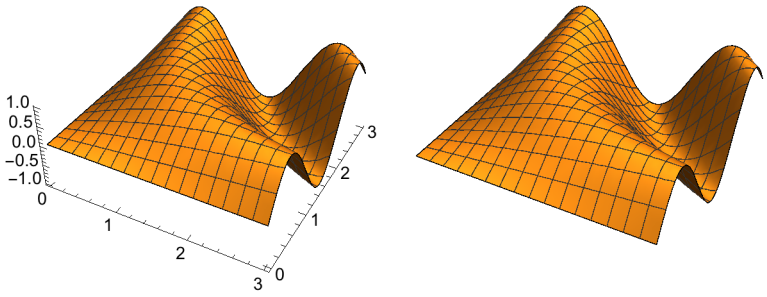
```
p1 = Graphics3D[{Sphere[], Cylinder[{{3, 0, -1}, {3, 0, 1}}]}, Axes → True];
p2 = Graphics3D[{Sphere[], Cylinder[{{3, 0, -1}, {3, 0, 1}}]},
    Axes → True, BaseStyle → Orange];
Show[GraphicsRow[{p1, p2}], ImageSize → 350]
```



```
? Boxed
```

Boxed is an option for Graphics3D that specifies whether
to draw the edges of the bounding box in a three-dimensional picture. ≫

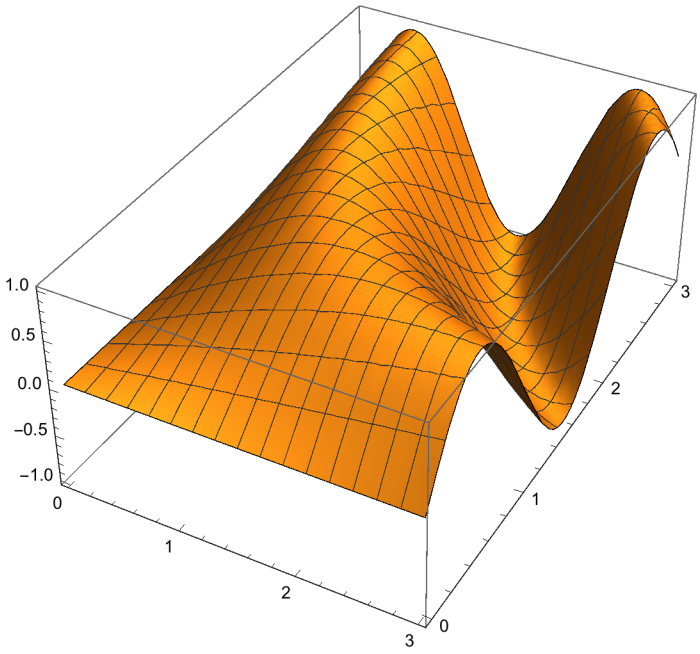
```
p1 = Show[pp, Boxed → False];
p2 = Show[pp, Axes → None, Boxed → False];
Show[GraphicsRow[{p1, p2}], ImageSize → 400]
```



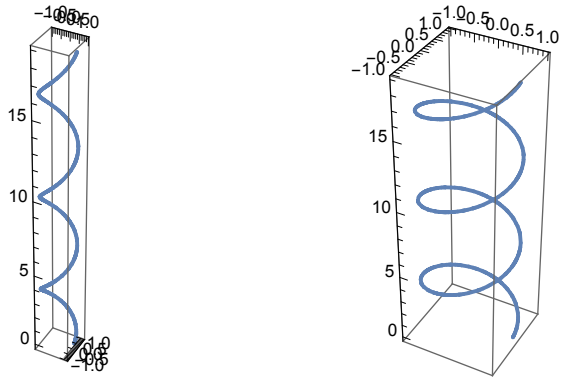

```
? BoxRatios
```

BoxRatios is an option for Graphics3D that gives the
ratios of side lengths for the bounding box of the three-dimensional picture. ≫

```
Show[pp, BoxRatios → {1, 1.5, 0.6`}]
```



```
curve = {Cos[t], Sin[t], t};
p1 = ParametricPlot3D[Evaluate[curve], {t, 0, 6 π}, ImageSize → 100];
p2 = Show[p1, BoxRatios → {1, 1, 3}, ImageSize →  200];
Show[GraphicsRow[{p1, p2}], ImageSize → 400]
```

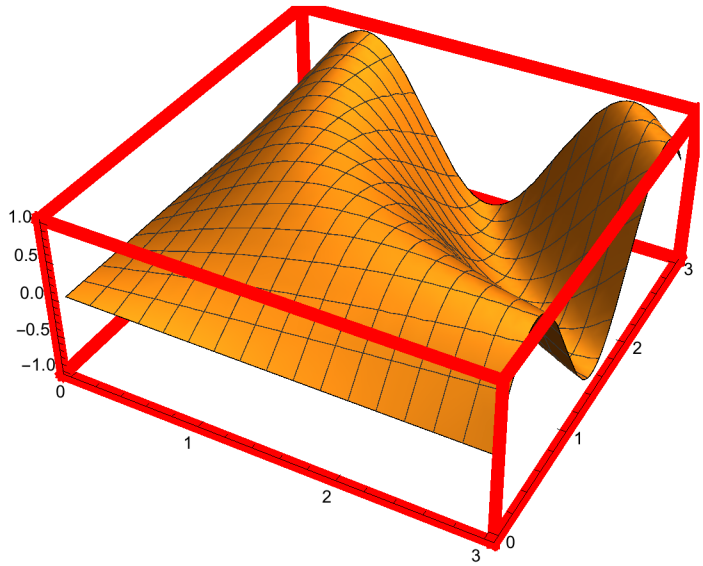



```
? BoxStyle
```

BoxStyle is an option for three-dimensional graphics
functions that specifies how the bounding box should be rendered. ≫

Examples of settings are: `Dashing[], Thickness[]; GrayLevel[], Hue[]; RGBColor[], CMYKColor []`.

```
plc = Show[pp, BoxStyle → {Thickness[0.02`], RGBColor[1, 0, 0]}]
```



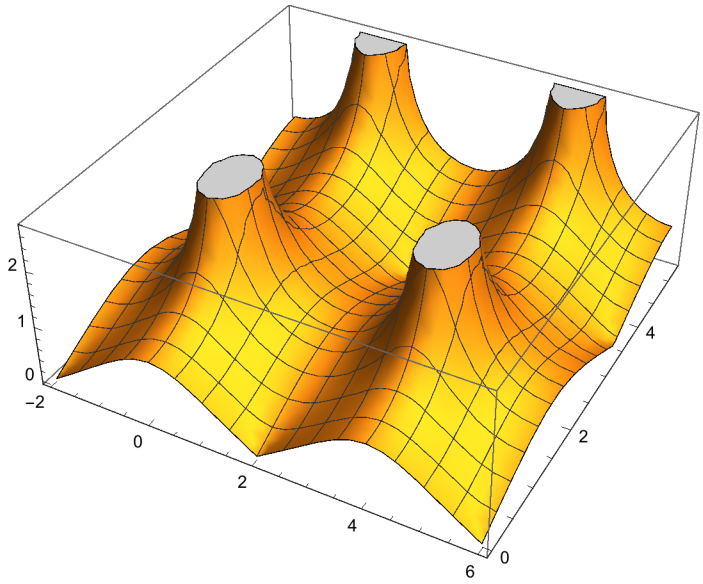
```
? ClippingStyle
```

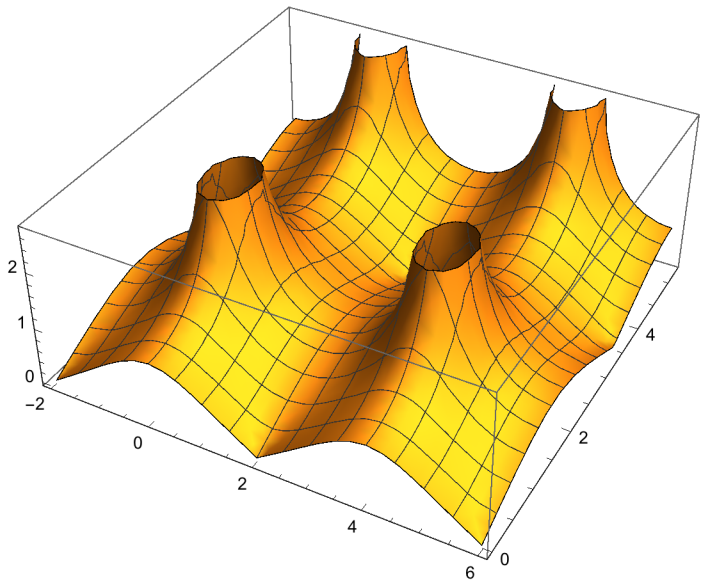
ClippingStyle is an option for plotting functions that specifies the style of what should be drawn when curves or surfaces would extend beyond the plot range. ≫

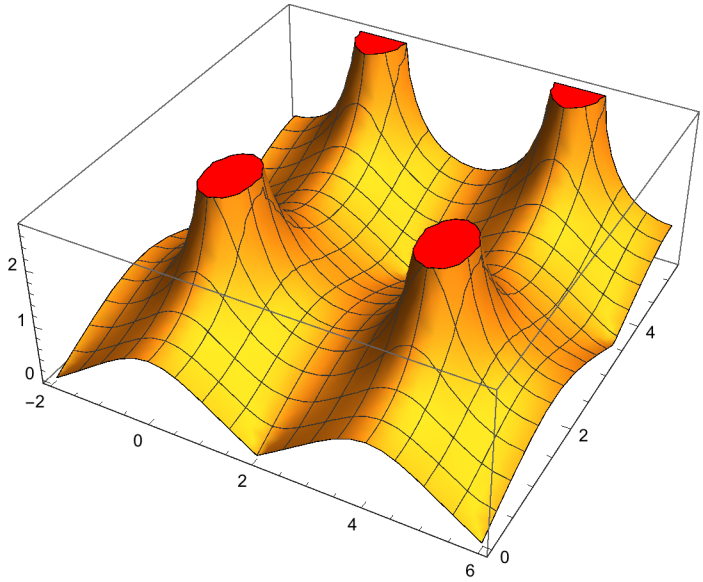
For **Plot3D[]** only ! Possible Settings are:

Automatic:	Show clipped area like the rest of the surface.
None:	Make holes where the surface would be clipped.
Color:	Show clipped areas with a particular color.
{bottom, top}:	Use different specifications for bottom, top clipped areas.

```
mm = 0.8`^2; m1 = 1 - mm;
cna = Abs[JacobiCN[x + i y, mm]];
per = EllipticK[mm]; pep = EllipticK[m1];
plcn = Plot3D[cna, {x, -per, 3 per}, {y, 0, 3 pep}]
p2 = Plot3D[cna, {x, -per, 3 per}, {y, 0, 3 pep}, ClippingStyle → None]
p3 = Plot3D[cna, {x, -per, 3 per}, {y, 0, 3 pep}, ClippingStyle → Red]
```






2
1
0
−2
0
2
4
6
0
2
4

```
? ColorFunctionScaling
```

ColorFunctionScaling is an option for graphics functions that specifies whether
arguments supplied to a color function should be scaled to lie between 0 and 1. ≫

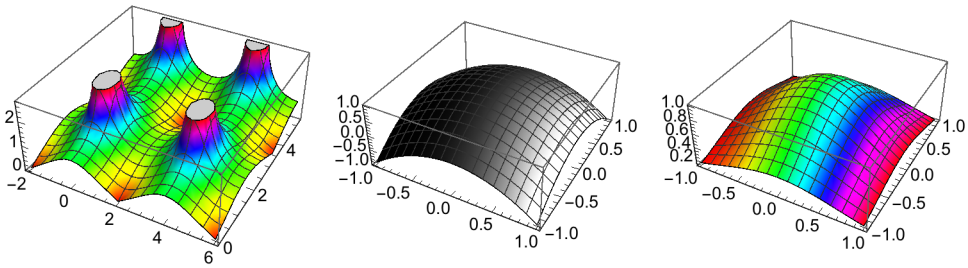
```
? ColorFunction
```

ColorFunction is an option for graphics functions
that specifies a function to apply to determine colors of elements. ≫

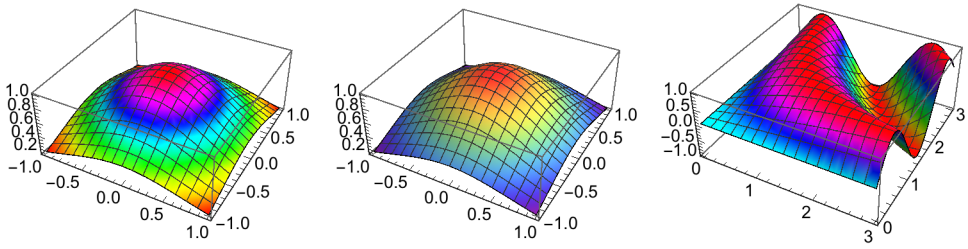
The option must specify a **GrayLevel[], Hue[], RGBColor[]** or **CMYKColor[]** directive. Note, however, the option specifying a curve drawn by **ParametricPlot3D[]** must be included as the fourth argument of the first list (cf. § 6.2.1).

ColorFunction -> Automatic: yields a range of gray levels or colors.
ColorFunction -> Hue: yields another range of colors.

```
p1 = Plot3D[cna, {x, -per, 3 per}, {y, 0, 3 pep}, ColorFunction → Hue];
p2 = Plot3D[1 - (x^2 + y^2), {x, -1, 1}, {y, -1, 1}, ColorFunction → (GrayLevel[#1^3] &)];
p3 = Plot3D[ Exp[-(x^2 + y^2)], {x, -1, 1}, {y, -1, 1}, ColorFunction → (Hue[#1^(3/2)] &)];
Show[GraphicsRow[{p1, p2, p3}], ImageSize → 500]
```



```
p1 = Plot3D[Exp[-(x^2 + y^2)], {x, -1, 1}, {y, -1, 1}, ColorFunction → Hue];
p2 = Plot3D[Exp[-(x^2 + y^2)], {x, -1, 1}, {y, -1, 1}, ColorFunction → "Rainbow"];
p3 = Plot3D[Sin[x y], {x, 0, 3},
    {y, 0, 3}, ColorFunction → Function[{x, y, z}, Hue[z]]];
Show[GraphicsRow[{p1, p2, p3}], ImageSize → 500]
```



```
? ColorOutput
```

ColorOutput is an option for graphics functions that specifies the type of color output to produce. ≫

Superseded by ColorFunction

? Epilog

Epilog is an option for graphics functions that gives a list of
graphics primitives to be rendered after the main part of the graphics is rendered. ≫

See 6.1.11.2

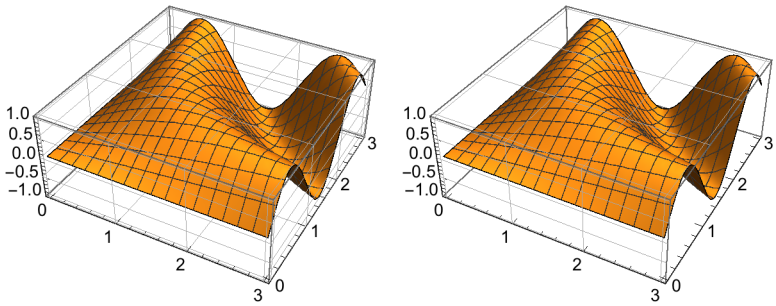
```
? FaceGrids
```

FaceGrids is an option for three-dimensional graphics functions that specifies grid lines to draw on the faces of the bounding box. ≫

None	No grid lines drawn.
All	Grid lines drawn on all faces.
{face$_1$,face$_2$,...}	Grid lines drawn on faces listed

face$_i$ = {dir$_x$,dir$_y$,dir$_z$} specifies the face where two **dir$_i$** must be 0; while the third one must be either -1 or 1. (As direction cosines)

```
pp = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}];
p1 = Show[pp, FaceGrids → All];
p2 = Show[pp, FaceGrids → {{0, 0, 1}, {0, -1, 0}}];
Show[GraphicsRow[{p1, p2}], ImageSize → 400]
```

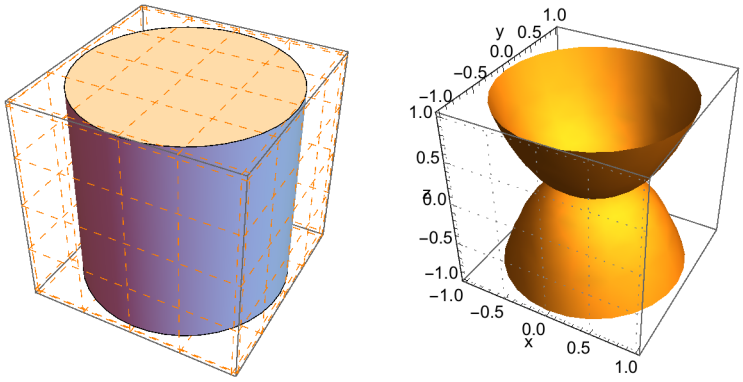



```
?? FaceGridsStyle
```

FaceGridsStyle is an option for 3D graphics functions that specifies how face grids should be rendered. ≫

```
Attributes[FaceGridsStyle] = {Protected}
```

```
p1 = Graphics3D[Cylinder[], FaceGrids → All,
    FaceGridsStyle → Directive[Orange, Dashed]];
p2 = ParametricPlot3D[{v Cos[u], v Sin[u], v^3}, {u, 0, 2 Pi},
    {v, -1, 1}, Mesh → None, FaceGrids → {{0, -1, 0}},
    AxesLabel → {"x", "y", "z"}, FaceGridsStyle → Directive[Gray, Dotted]];
Show[GraphicsRow[{p1, p2}], ImageSize → 400]
```



```
?? Filling                    (* Plot3D  only  *)
```

Filling is an option for ListPlot, Plot, Plot3D, and related functions that specifies what filling to add under points, curves, and surfaces. ≫

```
Attributes[Filling] = {Protected}
```

Information::nomatch : No symbol matching (* found. ≫

Plot3D[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a three-dimensional plot of f as a function of x and y.
Plot3D[{f_1, f_2, ...}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several functions. ≫

```
Attributes[Plot3D] = {HoldAll, Protected, ReadProtected}

Options[Plot3D] = {AlignmentPoint → Center, AspectRatio → Automatic,
  AutomaticImageSize → False, Axes → True, AxesEdge → Automatic,
  AxesLabel → None, AxesOrigin → Automatic, AxesStyle → {}, Background → None,
  BaselinePosition → Automatic, BaseStyle → {}, BoundaryStyle → GrayLevel[0],
  Boxed → True, BoxRatios → {1, 1, 0.4}, BoxStyle → {}, ClippingStyle → Automatic,
  ClipPlanes → None, ClipPlanesStyle → Automatic, ColorFunction → Automatic,
  ColorFunctionScaling → True, ColorOutput → Automatic, ContentSelectable → Automatic,
  ControllerLinking → Automatic, ControllerMethod → Automatic,
  ControllerPath → Automatic, CoordinatesToolOptions → Automatic,
  DisplayFunction :→ $DisplayFunction, Epilog → {}, Evaluated → Automatic,
  EvaluationMonitor → None, Exclusions → Automatic, ExclusionsStyle → None,
  FaceGrids → None, FaceGridsStyle → {}, Filling → None, FillingStyle → Opacity[0.5],
  FormatType :→ TraditionalForm, ImageMargins → 0., ImagePadding → All,
  ImageSize → Automatic, ImageSizeRaw → Automatic, LabelStyle → {}, Lighting → Automatic,
  MaxRecursion → Automatic, Mesh → Automatic, MeshFunctions → {#1 &, #2 &},
  MeshShading → None, MeshStyle → Automatic, Method → Automatic,
  NormalsFunction → Automatic, PerformanceGoal :→ $PerformanceGoal, PlotLabel → None,
  PlotLegends → None, PlotPoints → Automatic, PlotRange → {Full, Full, Automatic},
  PlotRangePadding → Automatic, PlotRegion → Automatic, PlotStyle → Automatic,
  PlotTheme :→ $PlotTheme, PreserveImageOptions → Automatic, Prolog → {},
  RegionFunction → (True &), RotationAction → Fit, SphericalRegion → False,
  TargetUnits → Automatic, TextureCoordinateFunction → Automatic,
  TextureCoordinateScaling → Automatic, Ticks → Automatic, TicksStyle → {},
  TouchscreenAutoZoom → False, ViewAngle → Automatic, ViewCenter → Automatic,
  ViewMatrix → Automatic, ViewPoint → {1.3, -2.4, 2.}, ViewRange → All,
  ViewVector → Automatic, ViewVertical → {0, 0, 1}, WorkingPrecision → MachinePrecision}
```

Information::notfound : Symbol only not found. ≫

Information::nomatch : No symbol matching *) found. ≫

```
?? FillingStyle               (* Plot3D  only  *)
```

FillingStyle is an option for ListPlot, Plot, Plot3D, and related functions that specifies the default style of filling to be used. ≫

```
Attributes[FillingStyle] = {Protected}
```

Information::nomatch : No symbol matching (* found. ≫

Plot3D[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a three-dimensional plot of f as a function of x and y.
Plot3D[{f_1, f_2, ...}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several functions. ≫

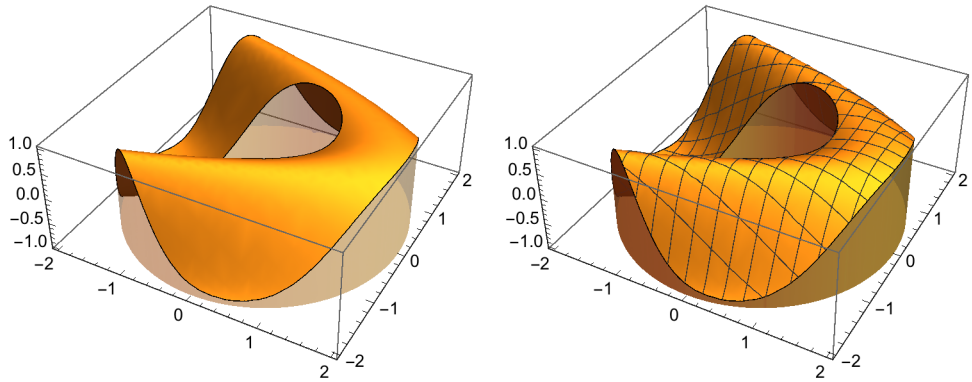
```
Attributes[Plot3D] = {HoldAll, Protected, ReadProtected}
```

```
Options[Plot3D] = {AlignmentPoint → Center, AspectRatio → Automatic,
  AutomaticImageSize → False, Axes → True, AxesEdge → Automatic,
  AxesLabel → None, AxesOrigin → Automatic, AxesStyle → {}, Background → None,
  BaselinePosition → Automatic, BaseStyle → {}, BoundaryStyle → GrayLevel[0],
  Boxed → True, BoxRatios → {1, 1, 0.4}, BoxStyle → {}, ClippingStyle → Automatic,
  ClipPlanes → None, ClipPlanesStyle → Automatic, ColorFunction → Automatic,
  ColorFunctionScaling → True, ColorOutput → Automatic, ContentSelectable → Automatic,
  ControllerLinking → Automatic, ControllerMethod → Automatic,
  ControllerPath → Automatic, CoordinatesToolOptions → Automatic,
  DisplayFunction :→ $DisplayFunction, Epilog → {}, Evaluated → Automatic,
  EvaluationMonitor → None, Exclusions → Automatic, ExclusionsStyle → None,
  FaceGrids → None, FaceGridsStyle → {}, Filling → None, FillingStyle → Opacity[0.5],
  FormatType :→ TraditionalForm, ImageMargins → 0., ImagePadding → All,
  ImageSize → Automatic, ImageSizeRaw → Automatic, LabelStyle → {}, Lighting → Automatic,
  MaxRecursion → Automatic, Mesh → Automatic, MeshFunctions → {#1 &, #2 &},
  MeshShading → None, MeshStyle → Automatic, Method → Automatic,
  NormalsFunction → Automatic, PerformanceGoal :→ $PerformanceGoal, PlotLabel → None,
  PlotLegends → None, PlotPoints → Automatic, PlotRange → {Full, Full, Automatic},
  PlotRangePadding → Automatic, PlotRegion → Automatic, PlotStyle → Automatic,
  PlotTheme :→ $PlotTheme, PreserveImageOptions → Automatic, Prolog → {},
  RegionFunction → (True &), RotationAction → Fit, SphericalRegion → False,
  TargetUnits → Automatic, TextureCoordinateFunction → Automatic,
  TextureCoordinateScaling → Automatic, Ticks → Automatic, TicksStyle → {},
  TouchscreenAutoZoom → False, ViewAngle → Automatic, ViewCenter → Automatic,
  ViewMatrix → Automatic, ViewPoint → {1.3, -2.4, 2.}, ViewRange → All,
  ViewVector → Automatic, ViewVertical → {0, 0, 1}, WorkingPrecision → MachinePrecision}
```

Information::notfound : Symbol only not found. ≫

Information::nomatch : No symbol matching *) found. ≫

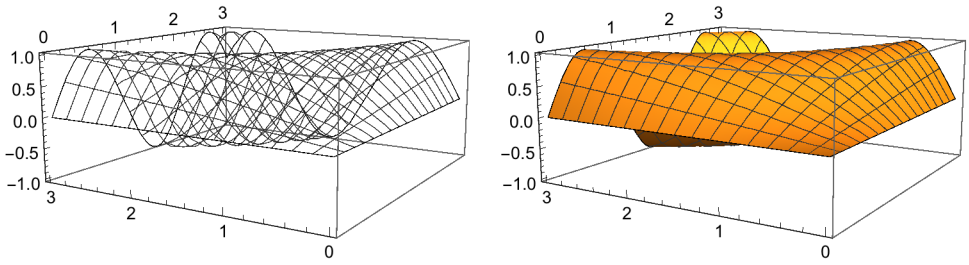
```
p1 = Plot3D[Sin[x + y^2], {x, -2, 2},
    {y, -2, 2}, RegionFunction -> (1 < #1^2 + #2^2 < 4 &),
    Filling → Bottom, FillingStyle → Opacity[0.3], Mesh → None];
p2 = Plot3D[Sin[x + y^2], {x, -2, 2}, {y, -2, 2}, RegionFunction ->
     (1 < #1^2 + #2^2 < 4 &), Filling → Bottom, FillingStyle → Opacity[0.7]];
Show[GraphicsRow[{p1, p2}], ImageSize → 500]
```



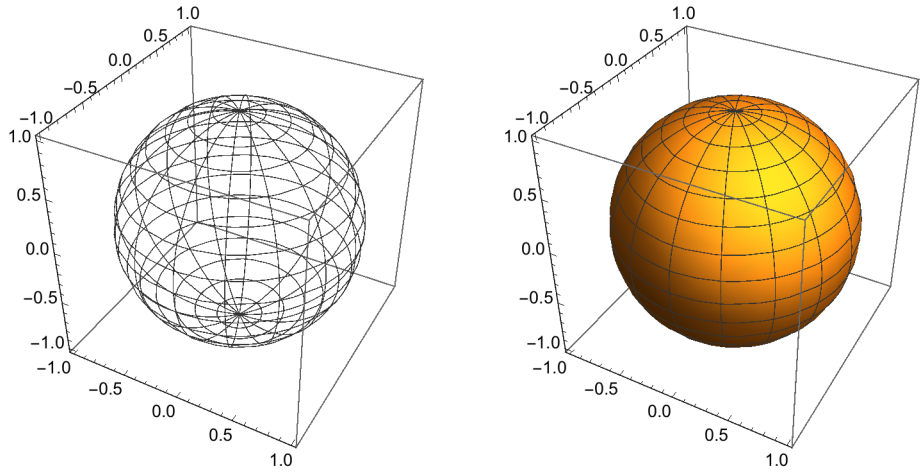
The surface may be rendered transparent by the option **PlotStyle → FaceForm[]** for both **Plot3D** and

ParametricPlot3D .

```
p1 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3},
    PlotStyle → FaceForm[], ViewPoint → {-2.281`, -1.373`, 0.5`}];
p2 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, ViewPoint → {-2.281`, -1.373`, 0.5`}];
Show[GraphicsRow[{p1, p2}], ImageSize → 500]
```




```
p1 =
  ParametricPlot3D[{Sin[t] Sin[p], Sin[t] Cos[p], Cos[t]}, {t, 0, π}, {p, 0, 2 π},
   PlotStyle → FaceForm[]];
p2 =
  ParametricPlot3D[{Sin[t] Sin[p], Sin[t] Cos[p], Cos[t]}, {t, 0, π}, {p, 0, 2 π}];
Show[GraphicsRow[{p1, p2}], ImageSize → 500]
```

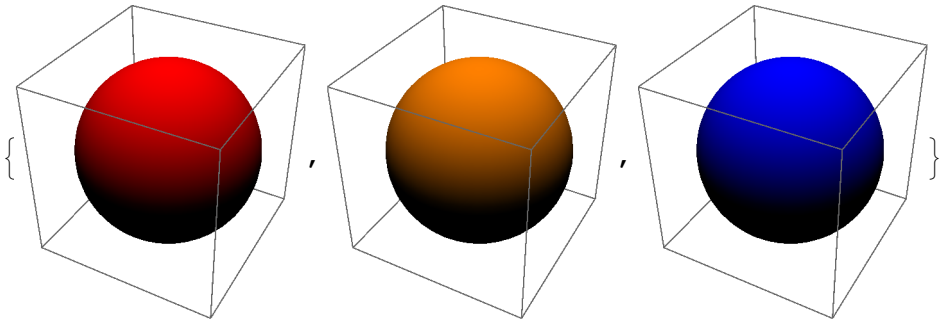


```
? Lighting
```

Lighting is an option for Graphics3D and related
functions that specifies what simulated lighting to use in coloring 3D surfaces. ≫

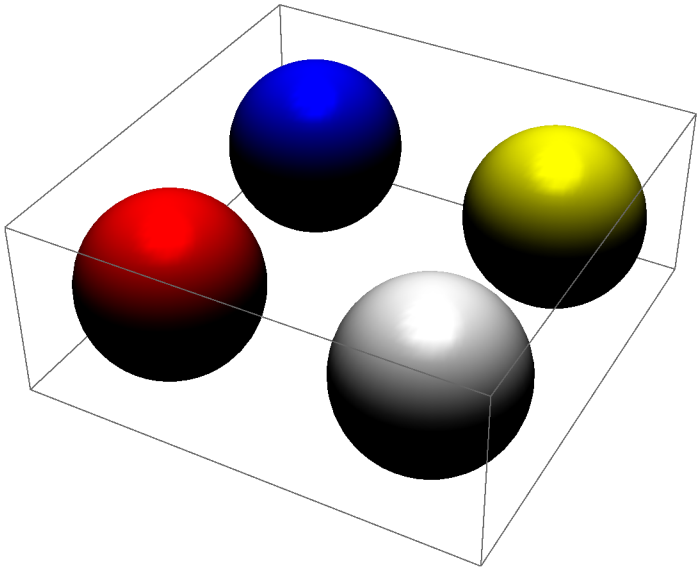
This is a rather complex command. Here only a simple example is shown.

```
Table[Graphics3D[{White, Sphere[]}, ImageSize → 150,
  Lighting → {{"Directional", c, {{0, 0, 1}, {0, 0, 0}}}}], {c, {Red, Orange, Blue}}]
```



Oben wird die Richtung der Lichtstraheln angegeben; unten die Position eines farbigen Scheinwerfers.

```
Graphics3D[{Specularity[White, 50], Lighting → {{"Point", Red, {0, 0, 5}}},
  Sphere[], Lighting → {{"Point", White, {3, 0, 5}}}, Sphere[{3, 0, 0}],
  Lighting → {{"Point", Blue, {0, 3, 5}}}, Sphere[{0, 3, 0}],
  Lighting → {{"Point", Yellow, {3, 3, 5}}}, Sphere[{3, 3, 0}]}]
```

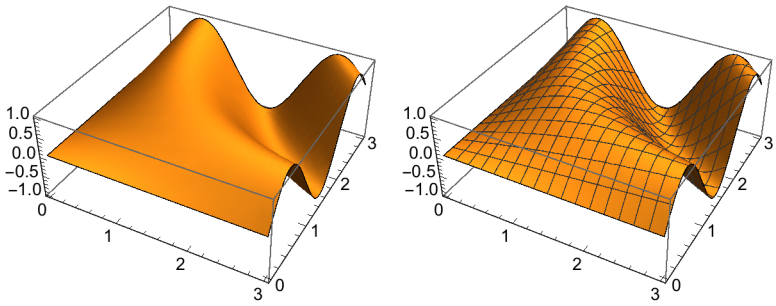


```
? Mesh
```

Mesh is an option for Plot3D, DensityPlot,
and other plotting functions that specifies what mesh should be drawn. ≫

Mesh -> True : A mesh is drawn (default value).
Mesh -> False: No mesh.

```
p1 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, Mesh → False];
p2 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, Mesh → True];
Show[GraphicsRow[{p1, p2}], ImageSize → 400]
```

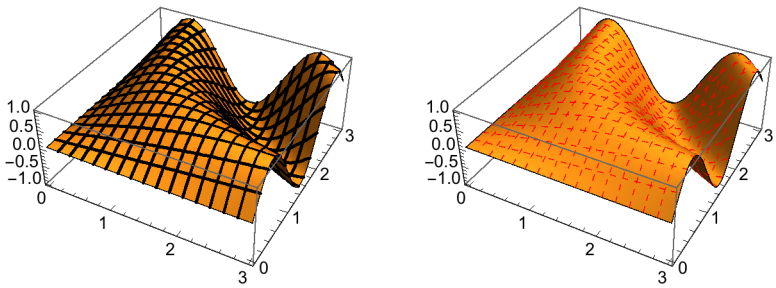



```
? MeshStyle
```

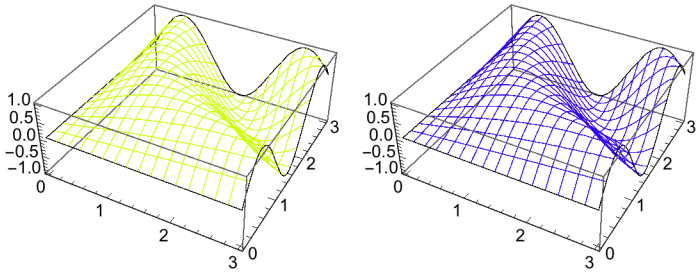
MeshStyle is an option for Plot3D, DensityPlot, and
other plotting functions that specifies the style in which to draw a mesh. ≫

Graphics directives include: **Dashing[], Thickness[]; GrayLevel[], Hue[]; RGBColor[], CMYKColor[].**

```
p1 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, MeshStyle → Thickness[0.01`]];
p2 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, MeshStyle → {{Dashing[{0.02`}], Red}}];
Show[GraphicsRow[{p1, p2}, Spacings → {Scaled[0.3`], Scaled[0]}], ImageSize → 400]
```



```
p1 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3},
   PlotStyle → FaceForm[], MeshStyle → Hue[0.2`]];
p2 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, PlotStyle → FaceForm[],
   MeshStyle → Hue[0.7`]]; Show[GraphicsRow[{p1, p2}]]
```




```
? PlotLabel
```

PlotLabel is an option for graphics functions that specifies an overall label for a plot. ≫

```
? PlotPoints
```

PlotPoints is an option for plotting functions that specifies how many initial sample points to use. ≫

With more than one variable, **PlotPoints -> n** specifies that n initial points should be used in each direction.

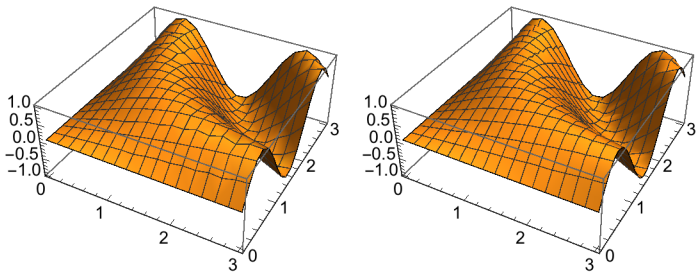
With a single variable, **PlotPoints -> n** specifies the total number of initial sample points to use.

PlotPoints ->
{n_1, n_2, …} specifies different numbers of initial sample points for each successive direction.

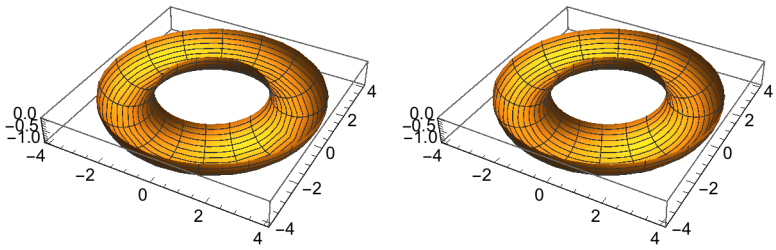
The initial sample points are usually equally spaced.

Adaptive procedures controlled by **MaxRecursion[]** are used to choose more sample points.

```
z = Sin[x y];
p1 = Plot3D[z, {x, 0, 3}, {y, 0, 3}, PlotPoints → 5];
p2 = Plot3D[z, {x, 0, 3}, {y, 0, 3}, PlotPoints → {5, 10}];
Show[GraphicsRow[{p1, p2}]]
```



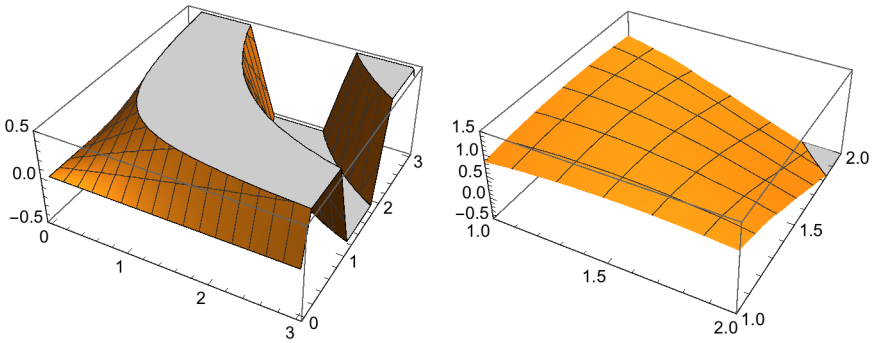
```
a = 3 + Cos[u]; x = a Cos[v]; y = a Sin[v]; z = Sin[u];
p1 = ParametricPlot3D[{x, y, z}, {u, π, 2 π}, {v, 0, 2 π}];
a = 3 + Cos[u]; x = a Cos[v]; y = a Sin[v]; z = Sin[u];
p2 = ParametricPlot3D[{x, y, z}, {u, π, 2 π}, {v, 0, 2 π}, PlotPoints → {7, 10}];
Show[GraphicsRow[{p1, p2}], ImageSize →  400]
```



```
?PlotRange
```

PlotRange is an option for graphics functions that specifies what range of coordinates to include in a plot. ≫

```
p1 = Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, PlotRange → {-0.5`, 0.5`}]; p2 =
  Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, PlotRange → {{1, 2}, {1, 2}, {-0.5`, 1.5`}}];
Show[GraphicsRow[{p1, p2}], ImageSize →  450]
```

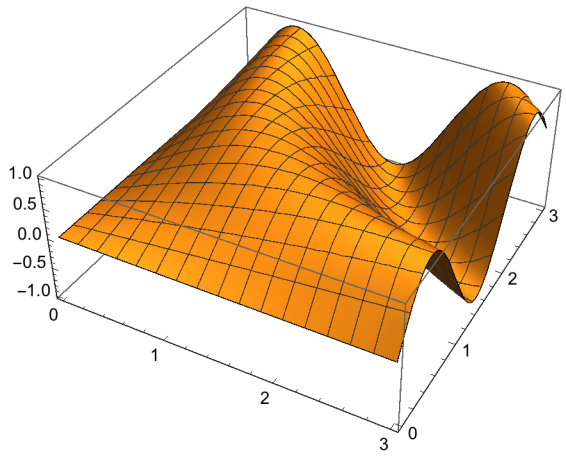


```
?PlotRegion
```

PlotRegion is an option for graphics functions
that specifies what region of the final display area a plot should fill. ≫

For Plot3D only !

```
Plot3D[Sin[x y], {x, 0, 3}, {y, 0, 3}, PlotRegion -> {{.1,.9}, {.3,.8}}]
```



Clicking at the picture shows your the frame sourring the area reserved for the image and the image compressed.

```
? PlotStyle
```

PlotStyle is an option for plotting and related functions that specifies styles in which objects are to be drawn. ≫

See explanations in § 6.1.11.2

?Prolog

Prolog is an option for graphics functions which gives a list of graphics
primitives to be rendered before the main part of the graphics is rendered. ≫

See explanations in § 6.1.11.2

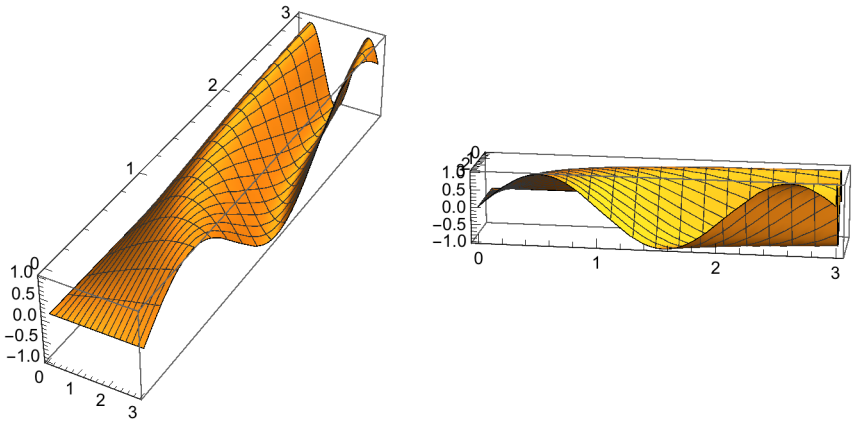
```
?SphericalRegion
```

SphericalRegion is an option for three-dimensional graphics functions
that specifies whether the final image should be scaled so that a sphere drawn around
the three-dimensional bounding box would fit in the display area specified. ≫

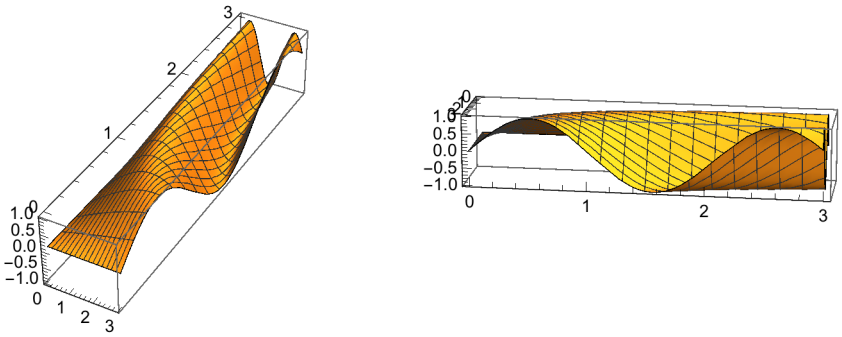
SphericalRegion -> False (default value)
scales 3-dimensional images to be as large as possible within the given display area.

SphericalRegion -> True
scales 3-dimensional images so that a sphere drawn around the 3-dimensional bounding box always fits into the display area specified. So the image of the object remains consistent in size, regardless of the orientation of the object.

```
pld = Show[pp, BoxRatios → {1, 5, 1}]; ple = Show[pld, ViewPoint → {7, 1, 2}];
Show[GraphicsRow[{pld, ple}], ImageSize → 450]
```



```
p1 = Show[pld, SphericalRegion → True]; p2 = Show[ple, SphericalRegion → True];
Show[GraphicsRow[{p1, p2}], ImageSize → 450]
```



```
? Ticks
```

Ticks is an option for graphics functions that specifies tick marks for axes. ≫

see 6.1.11.2

```
? TicksStyle
```

TicksStyle is an option for graphics functions which specifies how ticks should be rendered. ≫

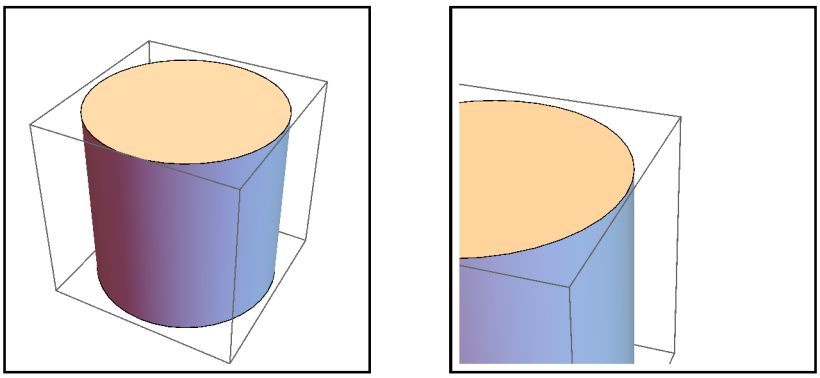
see 6.1.11.2

```
?ViewCenter
```

ViewCenter is an option for Graphics3D and related functions which gives the scaled coordinates of the point which should appear at the center of the final image. ≫

- With the default setting `ViewCenter -> Automatic`, the whole bounding box is centered in the final image.
- `ViewCenter -> Automatic` is equivalent to `ViewCenter -> {1 / 2, 1 / 2, 1 / 2}`, which places the center of the three-dimensional bounding box at the center of the final image.
- The setting for `ViewCenter` is given in scaled coordinates, which run from 0 to 1 across each dimension of the bounding box.
- `ViewCenter -> {{x, y, z}, {px, py}}` makes the 3D point with scaled coordinates (x, y, z) appear at scaled coordinate position $(p\,x, p\,y)$ in the final 2D image.
- The default is `ViewCenter -> {{x, y, z}, ImageScaled[{1, 1} / 2]}`.
- In a notebook front end, rotating a 3D object by dragging the mouse does not change the setting for `ViewCenter`. Panning changes the second element in the `ViewCenter` list.
- Explicit settings for `ViewVector` override settings of the form `ViewCenter -> {x, y, z}`.
- Explicit settings for `ViewMatrix` always override settings for `ViewCenter`.

```
p1 = Framed[
    Graphics3D[Cylinder[], ViewCenter → Automatic, SphericalRegion → True]];
p2 = Framed[Graphics3D[Cylinder[], ViewCenter → {1, .5, 1},
     SphericalRegion → True]];
Show[GraphicsRow[{p1, p2}], ImageSize →  450]
```



The command for ViewCentre puts the top-right corner of the objects at the center of the final image **(p2)**.

```
?ViewPoint
```

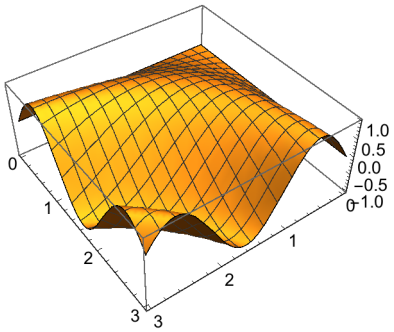
ViewPoint is an option for Graphics3D and related functions which gives the point in space from which three-dimensional objects are to be viewed. ≫

Default: **`ViewPoint -> {1.3, -2.4, 2.}`**

A new view point may be selected in several ways:

1. **The simplest method** is adding **an appropriate option.**

```
pp = Plot3D[Sin[x y], {x, 0, 3},
   {y, 0, 3}, ImageSize → 200, ViewPoint → {1.3`, 1.8`, 2}]
```

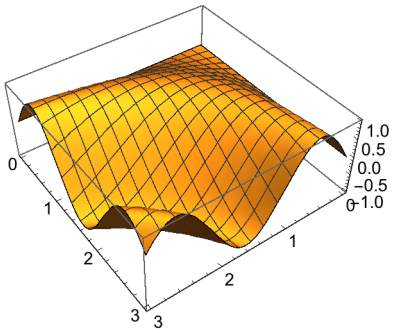


`Show[pp, ViewPoint -> {1.3,1.8,2} ]` renders the same figure but with different size.

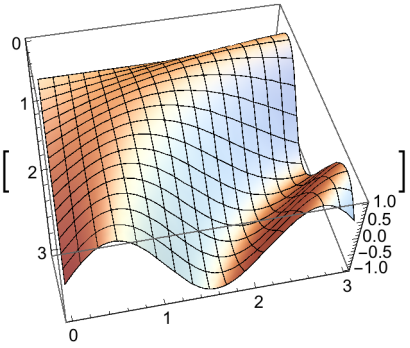
2. Dragging the picture with the mouse you may draw the picture in all directions

After you have found a convenient position of the object you write Options[] and copy the object as argument.
After "Retrun|" you get a list of options including that for the new viewpoint.

```
Show[pp]
```



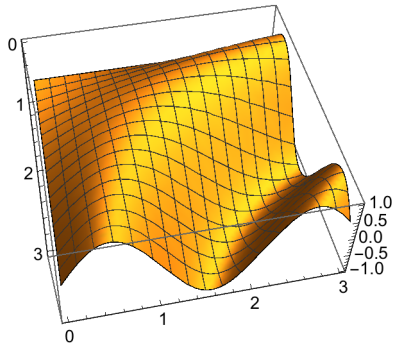

```
lo = Options[
```




```
]
```

```
{Axes → True, BoxRatios → {1, 1, 0.4},
 ImageSize → 200, Method → {RotationControl → Globe},
 PlotRange → {{0, 3}, {0, 3}, {-0.999996, 0.999993}},
 PlotRangePadding → {Scaled[0.02], Scaled[0.02], Scaled[0.02]},
 ViewPoint → {1.91523, -0.396313, 2.25939}, ViewVertical → {0., 0., 1.}}
```

```
Show[pp, lo[[-2]]]
```



3. Dragging with the mouse while pressing Ctrl, Alt or Option zooms in or out, changing the value of `ViewAngle`, but keeping `ViewPoint` fixed.

4. More information on directives for the option ViewPoint.

- `ViewPoint -> {x, y, z}` gives the position of the view point relative to the center of the three-dimensional box that contains the objects.
- The view point is given in a special scaled coordinate system in which the longest side of the bounding box has length 1. The center of the bounding box is taken to have coordinates `{0, 0, 0}`.
- Common settings for `ViewPoint` are:

`{1.3, -2.4, 2}`	default setting
`{0, -2, 0}`	directly in front
`{0, -2, 2}`	in front and up
`{0, -2, -2}`	in front and down
`{-2, -2, 0}`	left-hand corner
`{2, -2, 0}`	right-hand corner
`{0, 0, 2}`	directly above

- The following symbolic forms can also be used: »

`Above`	above, along the positive z direction
`Below`	below, along the negative z direction
`Front`	in front, along the negative y direction
`Back`	at back, along the positive y direction
`Left`	left, along the negative x direction
`Right`	right, along the positive x direction
`{Left, Top}`, etc.	corners

- Choosing a `ViewPoint` farther away from the object reduces the distortion associated with perspective.
- Infinite coordinates can be used to specify orthographic views: »

`{0, 0, Infinity}`	view from above (plan view)
`{0, 0, -Infinity}`	view from below
`{0, -Infinity, 0}`	view from the front (front elevation)
`{0, Infinity, 0}`	view from the back
`{-Infinity, 0, 0}`	view from the left
`{Infinity, 0, 0}`	view from the right

- The coordinates of the corners of the bounding box in the special coordinate system used for `ViewPoint` are determined by the setting for the `BoxRatios` option.

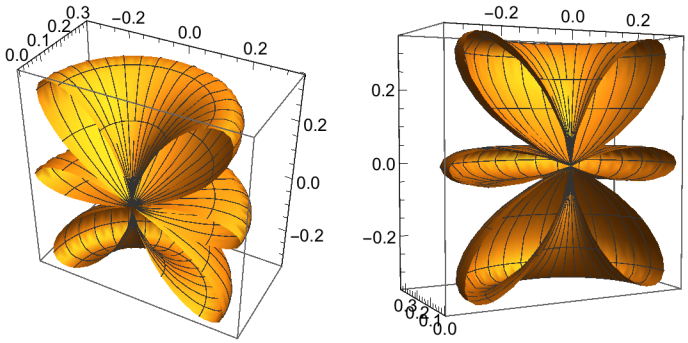
- In a notebook front end, dragging with the mouse rotates a 3D object, by changing the azimuthal components of `ViewPoint`, as well as the setting for `ViewVertical`.
- Dragging with the mouse while pressing Ctrl, Alt or Option zooms in or out, changing the value of `ViewAngle`, but keeping `ViewPoint` fixed.
- Explicit settings for `ViewVector` or `ViewMatrix` override settings for `ViewPoint`.

The spherical harmonics

$$Y_{lm}(\vartheta,\varphi) = N_{lm} P_l^m(\cos\vartheta)\, e^{im\varphi}$$

are very important functions in quantum mechanics.

```
Clear[x, y, z]
psa = Abs[SphericalHarmonicY[4, 2, th, ph]];
x = psa Sin[th] Cos[ph];
y = psa Sin[th] Sin[ph];
z = psa Cos[th];
plwa = ParametricPlot3D[{x, y, z}, {th, 0, π}, {ph, 0, π}];
p2 = Show[plwa, ViewPoint → {-1.509`, -2.91`, 0.18`}];
Show[GraphicsRow[{plwa, p2}]]
```



6.2.9 Options for ContourPlot and DensityPlot

```
?? ContourPlot
```

ContourPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a contour plot of f as a function of x and y.
ContourPlot[f == g, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots contour lines for which f = g.
ContourPlot[{f_1 == g_1, f_2 == g_2, ...}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several contour lines. ≫

```
Attributes[ContourPlot] = {HoldAll, Protected, ReadProtected}
```

```
Options[ContourPlot] = {AlignmentPoint → Center, AspectRatio → 1,
  Axes → False, AxesLabel → None, AxesOrigin → Automatic, AxesStyle → {},
  Background → None, BaselinePosition → Automatic, BaseStyle → {},
  BoundaryStyle → None, BoxRatios → Automatic, ClippingStyle → None,
  ColorFunction → Automatic, ColorFunctionScaling → True, ColorOutput → Automatic,
  ContentSelectable → Automatic, ContourLabels → Automatic, ContourLines → True,
  Contours → Automatic, ContourShading → Automatic, ContourStyle → Automatic,
  CoordinatesToolOptions → Automatic, DisplayFunction :→ $DisplayFunction,
  Epilog → {}, Evaluated → Automatic, EvaluationMonitor → None, Exclusions → Automatic,
  ExclusionsStyle → None, FormatType :→ TraditionalForm, Frame → True,
  FrameLabel → None, FrameStyle → {}, FrameTicks → Automatic, FrameTicksStyle → {},
  GridLines → None, GridLinesStyle → {}, ImageMargins → 0., ImagePadding → All,
  ImageSize → Automatic, ImageSizeRaw → Automatic, LabelStyle → {}, LightingAngle → None,
  MaxRecursion → Automatic, Mesh → None, MeshFunctions → {}, MeshStyle → Automatic,
  Method → Automatic, PerformanceGoal :→ $PerformanceGoal, PlotLabel → None,
  PlotLegends → None, PlotPoints → Automatic, PlotRange → {Full, Full, Automatic},
  PlotRangeClipping → True, PlotRangePadding → Automatic, PlotRegion → Automatic,
  PlotTheme :→ $PlotTheme, PreserveImageOptions → Automatic, Prolog → {},
  RegionFunction → (True &), RotateLabel → True, TargetUnits → Automatic,
  Ticks → Automatic, TicksStyle → {}, WorkingPrecision → MachinePrecision}
```

?? DensityPlot

DensityPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] makes a density plot of f as a function of x and y. ≫

```
Attributes[DensityPlot] = {HoldAll, Protected, ReadProtected}
```

```
Options[DensityPlot] = {AlignmentPoint → Center, AspectRatio → 1, Axes → False,
  AxesLabel → None, AxesOrigin → Automatic, AxesStyle → {}, Background → None,
  BaselinePosition → Automatic, BaseStyle → {}, BoundaryStyle → None,
  BoxRatios → Automatic, ClippingStyle → None, ColorFunction → Automatic,
  ColorFunctionScaling → True, ColorOutput → Automatic, ContentSelectable → Automatic,
  CoordinatesToolOptions → Automatic, DisplayFunction :→ $DisplayFunction,
  Epilog → {}, Evaluated → Automatic, EvaluationMonitor → None,
  Exclusions → Automatic, ExclusionsStyle → None, FormatType :→ TraditionalForm,
  Frame → True, FrameLabel → None, FrameStyle → {}, FrameTicks → Automatic,
  FrameTicksStyle → {}, GridLines → None, GridLinesStyle → {}, ImageMargins → 0.,
  ImagePadding → All, ImageSize → Automatic, ImageSizeRaw → Automatic,
  LabelStyle → {}, LightingAngle → None, MaxRecursion → Automatic, Mesh → None,
  MeshFunctions → {#1 &, #2 &}, MeshShading → None, MeshStyle → Automatic,
  Method → Automatic, PerformanceGoal :→ $PerformanceGoal, PlotLabel → None,
  PlotLegends → None, PlotPoints → Automatic, PlotRange → {Full, Full, Automatic},
  PlotRangeClipping → True, PlotRangePadding → Automatic, PlotRegion → Automatic,
  PlotTheme :→ $PlotTheme, PreserveImageOptions → Automatic, Prolog → {},
  RegionFunction → (True &), RotateLabel → True, TargetUnits → Automatic,
  Ticks → Automatic, TicksStyle → {}, WorkingPrecision → MachinePrecision}
```

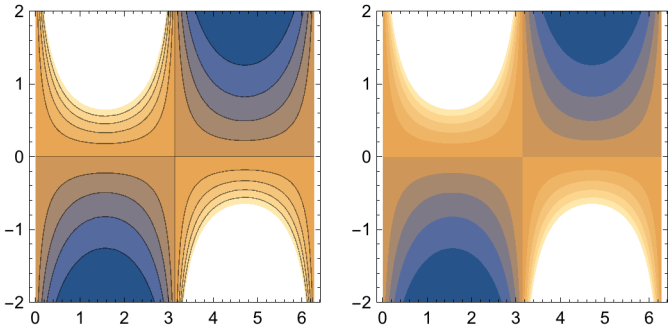
For the options:
AspectRatio, Axes, AxesLabel, AxesOrigin, AxesStyle, Background, ColorFunction, ColorOutput, Compiled cf. § 6.2.7

?ContourLines

ContourLines is an option for contour plots that specifies whether to draw explicit contour lines. ≫

ContourLines -> True (Default); or
ContourLines -> False.

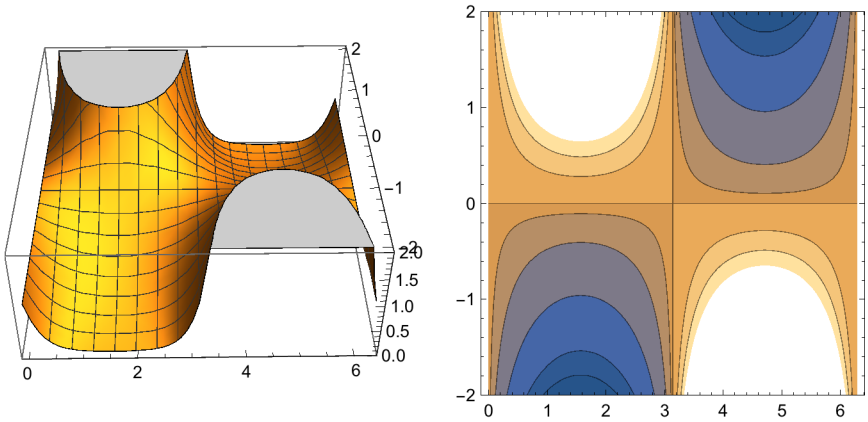
```
Clear[x,y,r]; f[x_,y_,r_] = Abs[Exp[ I r Cos[x + I y] ] ];

p1 = ContourPlot[f[x, y, 1], {x, 0, 2 π},
   {y, -2, 2}, PlotRange → {0, 2}, ContourStyle → Automatic];
p2 = ContourPlot[f[x, y, 1], {x, 0, 2 π}, {y, -2, 2},
   PlotRange → {0, 2}, ContourStyle → None];
Show[GraphicsRow[{p1, p2}], ImageSize → 350]
```



Contours -> n specifies to plot **n** contours, whose z values are spaced equally between the minimun and maximum z values. Default value n = 10.

Contours -> {z1,z2,....} specifies the z-values to use.

```
p1 = Plot3D[f[x, y, 1], {x, 0, 2 π}, {y, -2, 2},
   PlotRange → {0, 2}, ViewPoint → {-0.03`, -2.4`, 2.`}];
p2 = ContourPlot[f[x, y, 1], {x, 0, 2 π}, {y, -2, 2}, PlotRange → {0, 2},
   Contours → {0.055`, 0.11`, 0.33`, 0.66`, 0.9`, 1, 1.33`, 1.66`, 2.`}];
Show[GraphicsRow[{p1, p2}], ImageSize → 450]
```

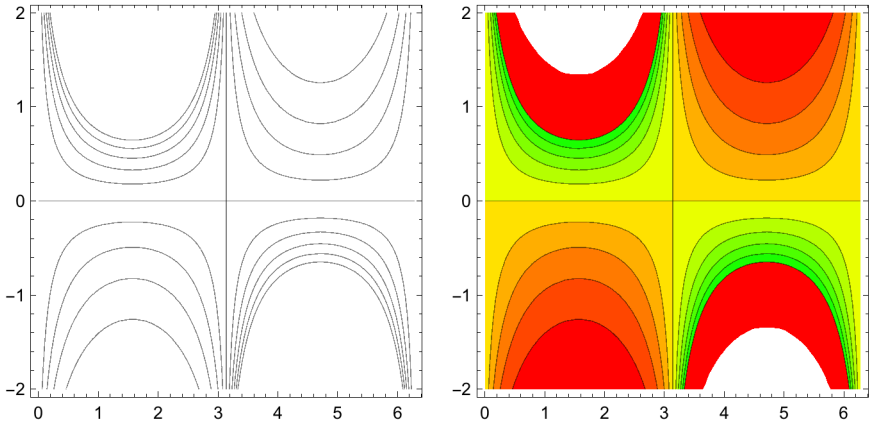


```
?ContourShading
```

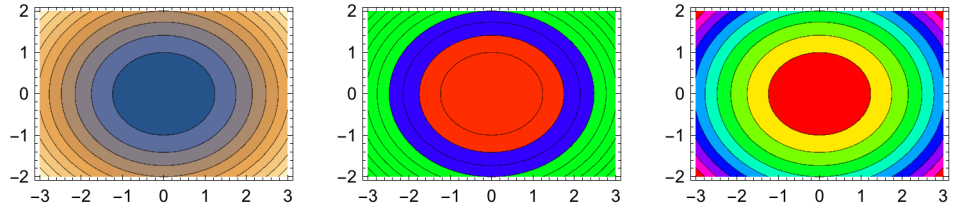
ContourShading is an option for contour plots
that specifies how the regions between contour lines should be shaded. ≫

ContourShading -> False : No shading
ContourShading -> True : Shadings of gray increasing from black to white with increasing height.
ContourShading -> ColourFunction

```
p1 = ContourPlot[f[x, y, 1], {x, 0, 2 π}, {y, -2, 2},
   Contours → {0.2`, 0.4`, 0.6`, 0.8`, 1.`, 1.2`, 1.4`, 1.6`, 1.8`, 2.`},
   ContourShading → False]; p2 = ContourPlot[f[x, y, 1], {x, 0, 2 π}, {y, -2, 2},
   Contours → {0.2`, 0.4`, 0.6`, 0.8`, 1.`, 1.2`, 1.4`, 1.6`, 1.8`, 2.`},
   ContourShading → True, ColorFunction → Hue];
Show[GraphicsRow[{p1, p2}], ImageSize → 450]
```



```
cc[xx_] := Which[xx ≥ 1, Hue[.35], xx > 0.5, Hue[.7], True, Hue[.03]]
p1 = ContourPlot[0.65` x^2 + y^2, {x, -3, 3}, {y, -2, 2}, AspectRatio → Automatic];
p2 = ContourPlot[0.65` x^2 + y^2, {x, -3, 3}, {y, -2, 2}, ColorFunctionScaling → True,
    ColorFunction → (cc[2.5 #1] &), AspectRatio → Automatic];
p3 = ContourPlot[0.65` x^2 + y^2, {x, -3, 3}, {y, -2, 2},
    ColorFunction → Hue , AspectRatio → Automatic];
Show[GraphicsRow[{p1, p2, p3}], ImageSize → 500]
```

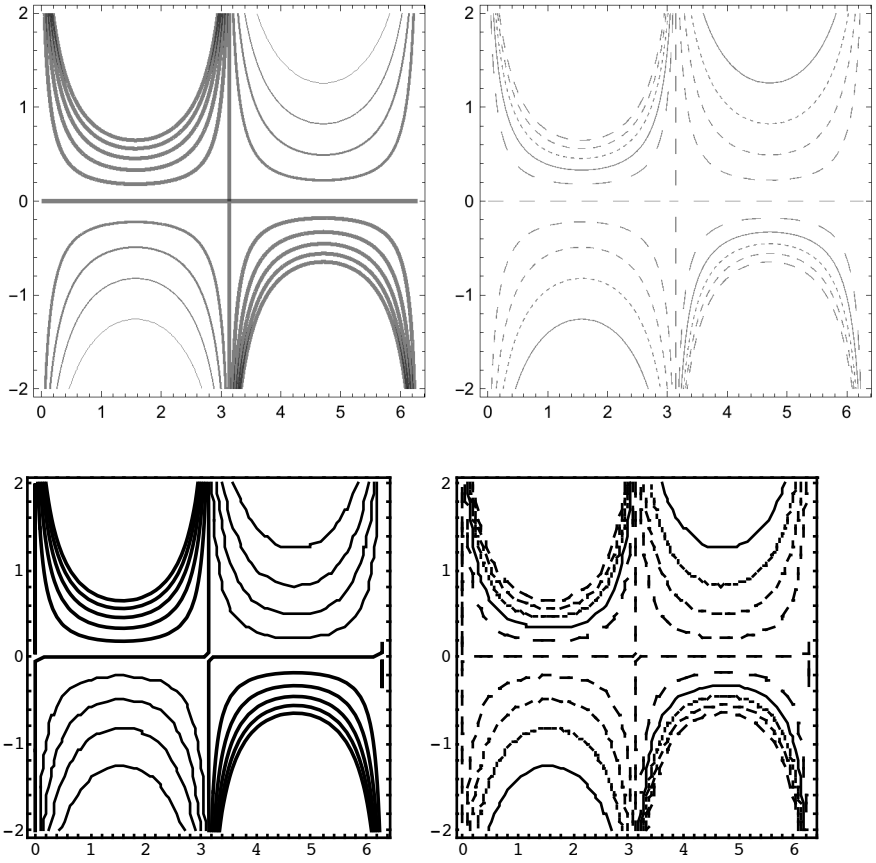


```
?ContourStyle
```

ContourStyle is an option for contour plots that
specifies the style in which contour lines or surfaces should be drawn. ≫

ContourStyle -> {{style1},{style2}, ...}
specifies that successive contour lines should be drawn according to style1, ...
Style directives must be lists, perhaps of length 1.
Corresponding Graphics Directives are: Dashing[], Thickness[]; Hue; RGBColor[], ...

```
p1 = ContourPlot[f[x, y, 1], {x, 0, 2 π}, {y, -2, 2},
   Contours → {0.2`, 0.4`, 0.6`, 0.8`, 1.`, 1.2`, 1.4`, 1.6`, 1.8`, 2.`},
   ContourShading → False, ContourStyle → {{Thickness[0.001`]},
     {Thickness[0.003`]}, {Thickness[0.005`]}, {Thickness[0.007`]},
     {Thickness[0.01`]}, {Thickness[0.01`]}, {Thickness[0.01`]},
     {Thickness[0.01`]}, {Thickness[0.01`]}, {Thickness[0.01`]}}];
p2 = ContourPlot[f[x, y, 1], {x, 0, 2 π}, {y, -2, 2},
   Contours → {0.2`, 0.4`, 0.6`, 0.8`, 1.`, 1.2`, 1.4`, 1.6`, 1.8`, 2.`},
   ContourShading → False,
   ContourStyle → {{Dashing[{}]}, {Dashing[{0.01`}]}, {Dashing[{0.02`}]},
     {Dashing[{0.03`}]}, {Dashing[{0.04`}]}, {Dashing[{0.05`}]}}];
Show[GraphicsRow[{p1, p2}], ImageSize → 450]
```



For the options:
DefaultColor, Epilog, Frame, Framlabel, FrameStyle, FrameTicks, PlotLabel, PlotPoints, PlotRange, PlotRegion, Prolog, RotateLabel, Ticks, DisplayFunction cf. § 6.1.9.

6.2.9.1 The Potential and the Equipotentials of the Restricted Three-Body System

In the restricted 3-body problem the motion of 3 masses under their mutual attraction is treated under the
assumption that one of them (the zero mass, e.g. a planetoid) is so small that it does not influence the
motion of the other two (the finite masses, e.g. Sun and Jupiter). The finite masses move with the same
angular velocity n on circles around the common centre of mass, i.e. their is equilibrium between the

mutual attraction and the centrifugal force.The finite masses have relative values $\mu \le 1/2$ and $1 - \mu$. In a frame rotating with angular velocity n the finite masses are located on the x-axis, at positions x1 = $-\mu$ a and x2 = $(1 - \mu)$ a; their mutual distance is chosen as unit of length, so a = 1. The rotation
period is taken as unit of time, so n = 1. The zero mass is attracted by the two masses and experiences
the centrifugal force. So its potential is:

```
Clear[fu, fx, fy, x, y]
```

$$\text{fu} = -\frac{1}{2}\left(x^2 + y^2\right) - \frac{\mu}{\sqrt{y^2 + (-1 + x + \mu)^2}} + \frac{-1 + \mu}{\sqrt{y^2 + (x + \mu)^2}};$$

The corresponding force components are:

```
fx = - D[fu, x]
```

$$x - \frac{\mu\,(-1 + x + \mu)}{\left(y^2 + (-1 + x + \mu)^2\right)^{3/2}} + \frac{(-1 + \mu)\,(x + \mu)}{\left(y^2 + (x + \mu)^2\right)^{3/2}}$$

```
fy = - D[fu, y]
```

$$y - \frac{y\,\mu}{\left(y^2 + (-1 + x + \mu)^2\right)^{3/2}} + \frac{y\,(-1 + \mu)}{\left(y^2 + (x + \mu)^2\right)^{3/2}}$$

The libration (= equilibrium) points of the zero mass are those points where fx = fy = 0. These are called
L_1, L_2, L_3, L_5,L_4.Here these are computed for the particular value μ = 1/4.

```
sm = μ → 1 / 4 ;
son = RotateRight[NSolve[Thread[{fx, fy} == {0, 0}] /. sm, {x, y}], 1]
```

```
{{y → 0, x → 0.360743}, {y → 0, x → 1.26586}, {y → 0, x → -1.10317},
 {y → -0.866025, x → 0.25}, {y → 0.866025, x → 0.25}}
```

Their positions and potential values are:

```
um = fu /. sm;
vli = {x, y, um} /. son
```

```
{{0.360743, 0, -1.93533}, {1.26586, 0, -1.7806}, {-1.10317, 0, -1.62247},
 {0.25, -0.866025, -1.40625}, {0.25, 0.866025, -1.40625}}
```

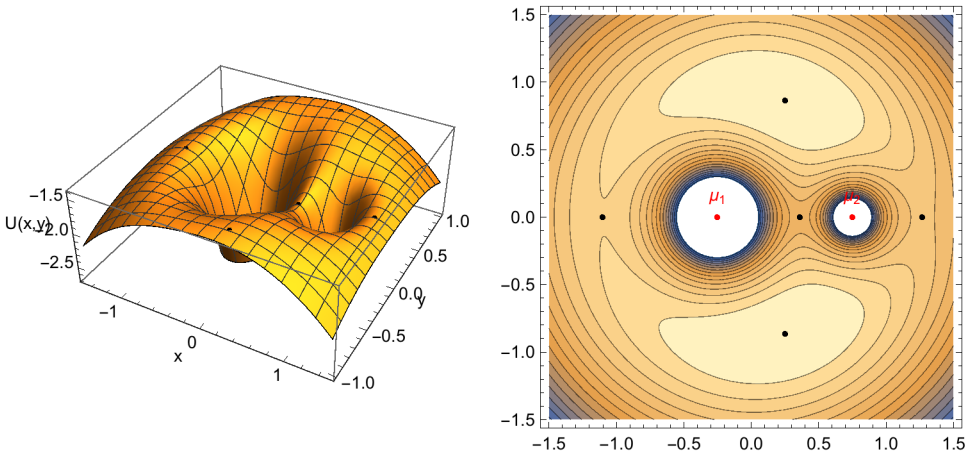
The surface corresponding to this potential is shown below:

```
pp = Plot3D[um, {x, -1.5`, 1.5`}, {y, -1, 1}, PlotPoints → 40];

pv = Show[pp, Graphics3D[Point /@ vli],
    ImageSize → 500, AxesLabel → {"x", "y", "U(x,y)"}];
```

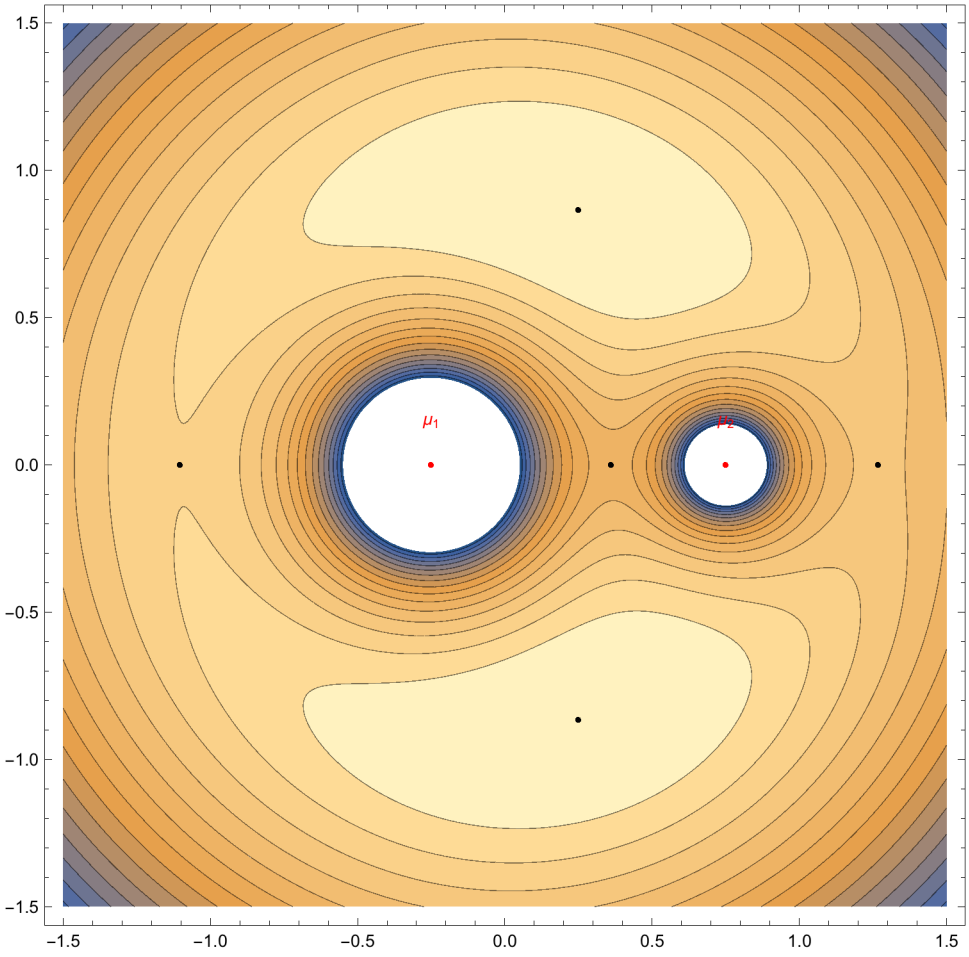
The equipotentials are computed by:

```
cp = ContourPlot[um, {x, -1.5`, 1.5`},
    {y, -1.5`, 1.5`}, Contours → 15, PlotPoints → 100];

pm = { Point[{ - μ, 0}], Point[{1 - μ, 0}]} /. sm ; (* masses *)
pe = Point[{x, y}] /. son ;           (* libration points *)
su = .15;
pms = {{- μ, su}, {1 - μ, su}} /. sm ; (* positions mass labels *)
tm =
  {Text[Subscript["μ", 1], pms[[1]]], Text[Subscript["μ", 2], pms[[2]]]} /. sm;
```

```
pc = Show[cp, Epilog → {pe, Red, pm, tm}, ImageSize → 500];

Show[GraphicsRow[{pv, pc}], ImageSize → 500]
```




```
Show[pc, ImageSize → 500]
```

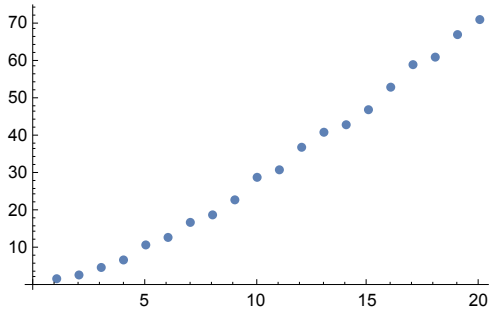


Pointing the pointer to a contour line in the picture above makes appear a tag indicating the corresponding height.

6.3 Graphics Elements in 2 and 3 Dimensions. Plotting Points and Lines. Texts in Graphics

Mathematica presents all graphics in terms of graphics primitives; these present elements of an image and can be used to compose a drawing or to insert some additional elments into a drawing by the option **Epilog.** **Prime[n]**, the command used below, renders the n-th prime number.

```
lp = ListPlot[Table[Prime[n], {n, 20}], ImageSize → 250]
```




```
InputForm[lp]
```

```
Graphics[{{}, {{{}, {}, {Hue[0.67, 0.6, 0.6], Directive[PointSize[0.019444444444444445],
      RGBColor[0.368417, 0.506779, 0.709798], AbsoluteThickness[1.6]],
     Point[{{1., 2.}, {2., 3.}, {3., 5.}, {4., 7.}, {5., 11.}, {6., 13.}, {7., 17.},
       {8., 19.}, {9., 23.}, {10., 29.}, {11., 31.}, {12., 37.}, {13., 41.}, {14., 43.},
       {15., 47.}, {16., 53.}, {17., 59.}, {18., 61.}, {19., 67.}, {20., 71.}}]}, {}}},
  {}}, {DisplayFunction -> Identity, PlotRangePadding ->
   {{Scaled[0.02], Scaled[0.02]}, {Scaled[0.02], Scaled[0.05]}}, AxesOrigin -> {0, 0},
  PlotRange -> {{0., 20.}, {0, 71.}}, DisplayFunction -> Identity,
  AspectRatio -> GoldenRatio^(-1), Axes -> {True, True}, AxesLabel -> {None, None},
  AxesOrigin -> {0, 0}, DisplayFunction :> Identity,
  Frame -> {{False, False}, {False, False}}, FrameLabel -> {{None, None}, {None, None}}
  FrameTicks -> {{Automatic, Automatic}, {Automatic, Automatic}},
  GridLines -> {None, None}, GridLinesStyle -> Directive[GrayLevel[0.5, 0.4]],
  ImageSize -> 250, Method -> {}, PlotRange -> {{0., 20.}, {0, 71.}},
  PlotRangeClipping -> True, PlotRangePadding -> {{Scaled[0.02], Scaled[0.02]},
    {Scaled[0.02], Scaled[0.05]}}, Ticks -> {Automatic, Automatic}}]
```

The output above shows that the information on the points is the first element of the Graphics list. This may be used
in another graphic:

```
lp[[1]]
```

```
{{}, {{{}, {■, Directive[PointSize[0.0194444], ■, AbsoluteThickness[1.6]],
     Point[{{1., 2.}, {2., 3.}, {3., 5.}, {4., 7.}, {5., 11.},
       {6., 13.}, {7., 17.}, {8., 19.}, {9., 23.}, {10., 29.}, {11., 31.},
       {12., 37.}, {13., 41.}, {14., 43.}, {15., 47.}, {16., 53.},
       {17., 59.}, {18., 61.}, {19., 67.}, {20., 71.}}]}, {}}}, {}}
```

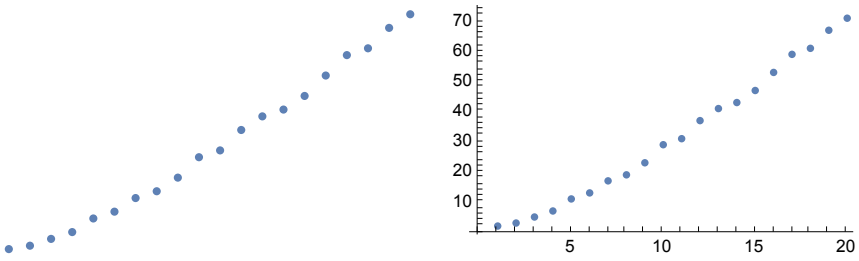
```
lp[[2]]
```

```
{DisplayFunction → Identity,
 PlotRangePadding → {{Scaled[0.02], Scaled[0.02]}, {Scaled[0.02], Scaled[0.05]}},
 AxesOrigin → {0, 0}, PlotRange → {{0., 20.}, {0, 71.}},
 DisplayFunction → Identity, AspectRatio → 1/GoldenRatio,
 Axes → {True, True}, AxesLabel → {None, None}, AxesOrigin → {0, 0},
 DisplayFunction :→ Identity, Frame → {{False, False}, {False, False}},
 FrameLabel → {{None, None}, {None, None}},
 FrameTicks → {{Automatic, Automatic}, {Automatic, Automatic}},
 GridLines → {None, None}, GridLinesStyle → Directive[ ], ImageSize → 250,
 Method → {}, PlotRange → {{0., 20.}, {0, 71.}}, PlotRangeClipping → True,
 PlotRangePadding → {{Scaled[0.02], Scaled[0.02]}, {Scaled[0.02], Scaled[0.05]}},
 Ticks → {Automatic, Automatic}}
```

```
rp = ReplacePart[lp[[1]], Red, 1]
```

```
{ , {{{}, { , Directive[PointSize[0.0194444], , AbsoluteThickness[1.6]],
     Point[{{1., 2.}, {2., 3.}, {3., 5.}, {4., 7.}, {5., 11.},
       {6., 13.}, {7., 17.}, {8., 19.}, {9., 23.}, {10., 29.}, {11., 31.},
       {12., 37.}, {13., 41.}, {14., 43.}, {15., 47.}, {16., 53.},
       {17., 59.}, {18., 61.}, {19., 67.}, {20., 71.}}]}, {}}}, {}}
```

```
p1 = Show[Graphics[{PointSize → 0.02, rp}], AspectRatio → 0.6];
p2 = Show[Graphics[{PointSize → 0.02, lp[[1]]}, lp[[2]]], AspectRatio → 0.6];
GraphicsRow[{p1, p2}, ImageSize → 450]
```



6.3.1 Two-Dimensional Graphics Elements

```
?? Graphics
```

Graphics[*primitives*, *options*] represents a two-dimensional graphical image. >>

```
Attributes[Graphics] = {Protected, ReadProtected}
```

```
Options[Graphics] = {AlignmentPoint → Center, AspectRatio → Automatic, Axes → False,
  AxesLabel → None, AxesOrigin → Automatic, AxesStyle → {}, Background → None,
  BaselinePosition → Automatic, BaseStyle → {}, ColorOutput → Automatic,
  ContentSelectable → Automatic, CoordinatesToolOptions → Automatic,
  DisplayFunction :→ $DisplayFunction, Epilog → {}, FormatType :→ TraditionalForm,
  Frame → False, FrameLabel → None, FrameStyle → {}, FrameTicks → Automatic,
  FrameTicksStyle → {}, GridLines → None, GridLinesStyle → {}, ImageMargins → 0.,
  ImagePadding → All, ImageSize → Automatic, ImageSizeRaw → Automatic,
  LabelStyle → {}, Method → Automatic, PlotLabel → None, PlotRange → All,
  PlotRangeClipping → False, PlotRangePadding → Automatic,
  PlotRegion → Automatic, PreserveImageOptions → Automatic,
  Prolog → {}, RotateLabel → True, Ticks → Automatic, TicksStyle → {}}
```

Point[{x,y}]	point at position {x,y}

Line[{{x1,y1},{x2,y2},....}]	line through points {x1,y1}, {x2,y2},....
Circle[{x,y},r]	circle with radius r centered at {x,y}
Disk[{x,y},r]	filled disk of radius r centered at {x,y}
Text[*expr*, {x,y}]	a list of the text of *expr*, centered at {x,y}

In 2 dimensions some results may be obtained in two ways; either by the graphics elements **Point[]** or by **ListPlot[]**; either by **Line[]** or by **ListPlot[]** with the option **PlotJoined -> True.** The graphics elements give drawings only by the command **Show[Graphics[]]** with the graphics elements as arguments.

? Point

Point[p] is a graphics and geometry primitive that represents a point at p.
Point[{p_1, p_2, ...}] represents a collection of points. ≫

? Line

Line[{p_1, p_2, ...}] represents the line segments joining a sequence for points p_i.
Line[{{p_{11}, p_{12}, ...}, {p_{21}, ...}, ...}] represents a collection of lines. ≫

? Circle

Circle[{x, y}, r] represents a circle of radius r centered at {x, y}.
Circle[{x, y}] gives a circle of radius 1.
Circle[{x, y}, {r_x, r_y}] gives an axis-aligned ellipse with semi-axes lengths r_x and r_y.
Circle[{x, y}, ..., {θ_1, θ_2}] gives a circular or ellipse arc from angle θ_1 to θ_2. ≫

? Disk

Disk[{x, y}, r] represents a disk of radius r centered at {x, y}.
Disk[{x, y}] gives a disk of radius 1.
Disk[{x, y}, {r_x, r_y}] gives an axis-aligned elliptical disk with semiaxes lengths r_x and r_y.
Disk[{x, y}, ..., {θ_1, θ_2}] gives a sector of a disk from angle θ_1 to θ_2. ≫

? Text

Text[*expr*] displays with *expr* in plain text format.
Text[*expr*, *coords*] is a graphics primitive that
displays the textual form of *expr* centered at the point specified by *coords*. ≫

? Arrow

Arrow[{pt_1, pt_2}] is a graphics primitive that represents an arrow from pt_1 to pt_2.
Arrow[{pt_1, pt_2}, s] represents an arrow with its ends set back from pt_1 and pt_2 by a distance s.
Arrow[{pt_1, pt_2}, {s_1, s_2}] sets back by s_1 from pt_1 and s_2 from pt_2.
Arrow[*curve*, ...] represents an arrow following the specified *curve*. ≫

?? Grid

Grid[{{$expr_{11}$, $expr_{12}$, ...}, {$expr_{21}$, $expr_{22}$, ...}, ...}] is
an object that formats with the $expr_{ij}$ arranged in a two-dimensional grid. >>

```
Attributes[Grid] = {Protected, ReadProtected}

Options[Grid] = {Alignment → {Center, Baseline}, AllowedDimensions → Automatic,
  AllowScriptLevelChange → True, AutoDelete → False, Background → None,
  BaselinePosition → Automatic, BaseStyle → {}, DefaultBaseStyle → Grid,
  DefaultElement → □, DeleteWithContents → True, Dividers → {},
  Editable → Automatic, Frame → None, FrameStyle → Automatic, ItemSize → Automatic,
  ItemStyle → None, Selectable → Automatic, Spacings → Automatic}
```

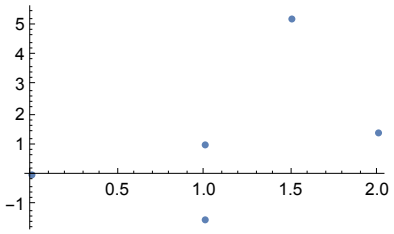
```
? Framed
```

Framed[*expr*] displays a framed version of *expr*. >>

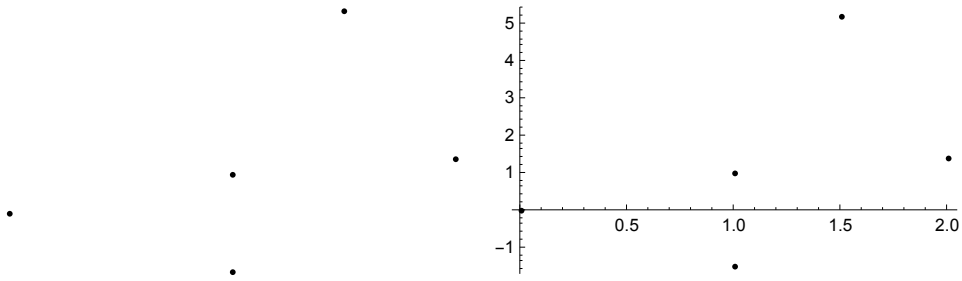
More details may be found at the Wolfram Website. Go to Help, then Documention Center, In the Search Window type **Graphics Objects**, click >>, then click **Graphics Objects** .

In 2 dimensions some results may be obtained in two ways; either by the graphics elements **Point[]** or by **ListPlot[]**; either by **Line[]** or by **ListPlot[]** with the option **PlotJoined -> True.** Sometimes graphics elements give drawings only by the command **Show[Graphics[]]** with the graphics elements as arguments.

```
li = {{0, 0}, {1, 1}, {1.5`, 5.2`}, {2, 1.4`}, {1, -1.5`}};
ListPlot[li, Prolog → PointSize[0.015`], ImageSize → 200]
```

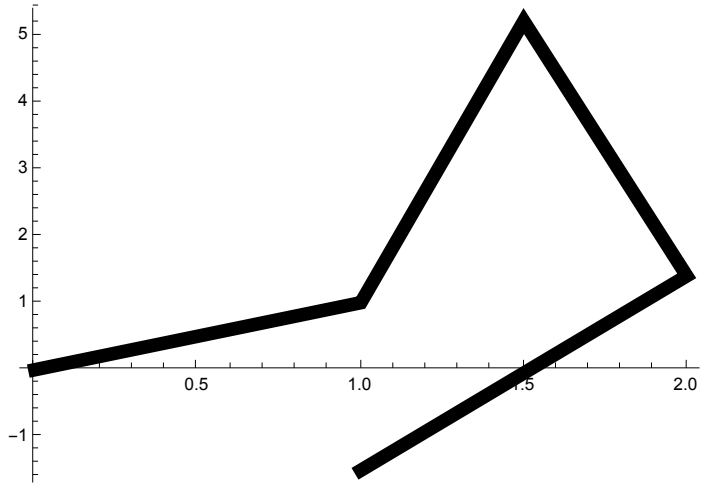



```
pt = Table[Graphics[Point[ li[[k]] ] ], {k,Length[li]} ] ;

as = 0.6;
p1 = Show[pt, AspectRatio → as];
p2 = Show[pt, Axes → True, AspectRatio → as];
Show[GraphicsRow[{p1, p2}], ImageSize → 500]
```

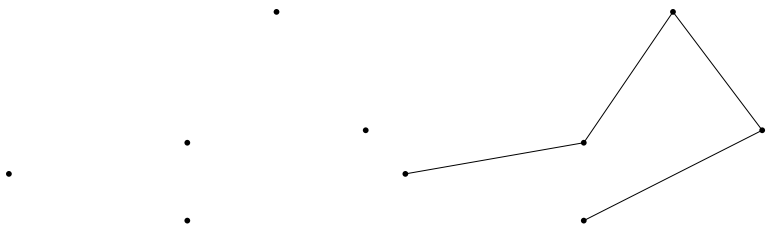


Style options must be included at the very first position of the list for a graphics element.

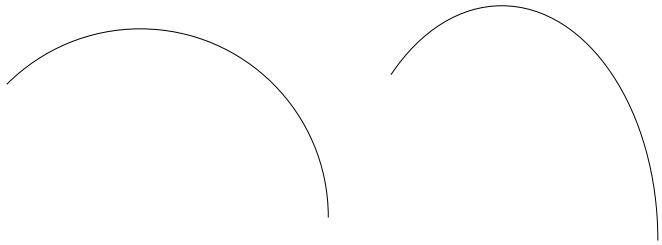
```
Show[Graphics[{Thickness[0.02`], Line[li]}], Axes → True, AspectRatio → 0.7]
```



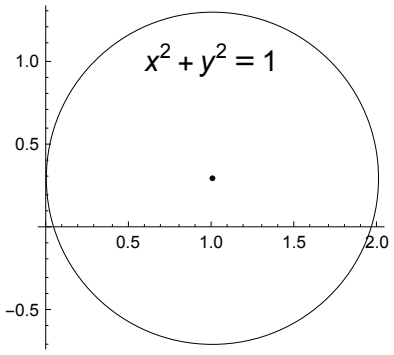

```
p1 = Show[pt, Prolog → PointSize[0.015`], AspectRatio → as];
p2 = Show[pt, Graphics[Line[li]], Prolog → PointSize[0.015`], AspectRatio → as];
Show[GraphicsRow[{p1, p2}], Prolog → PointSize[0.02`], ImageSize → 400]
```



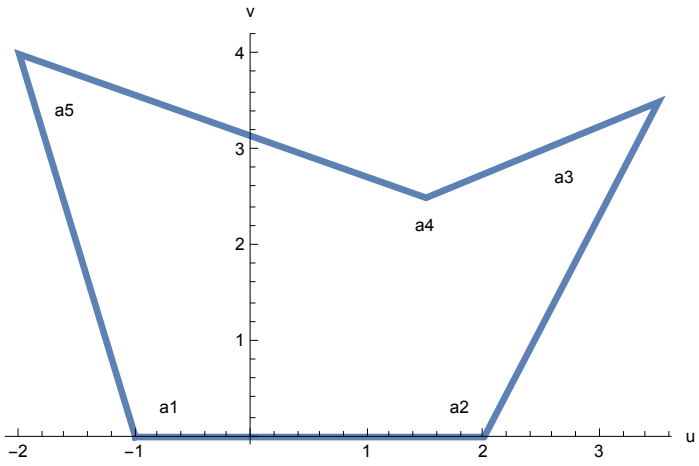
```
c1 = Circle[{0, 0}, 2, {0, 3 π / 4}];
p1 = Show[Graphics[c1], AspectRatio → Automatic];
c2 = Circle[{0, 0}, {2, 3}, {0, 3 π / 4}];
p2 = Show[Graphics[c2], AspectRatio → Automatic];
Show[GraphicsRow[{p1, p2}]]
```



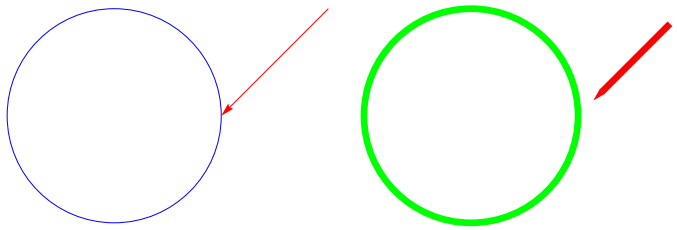
```
p1 = Graphics[{Point[{1, 0.3}], Circle[{1, 0.3}, 1],
   Text[Style[x^2 + y^2 == 1, 15], {1, 1}]}, Axes → True, ImageSize → 200]
```



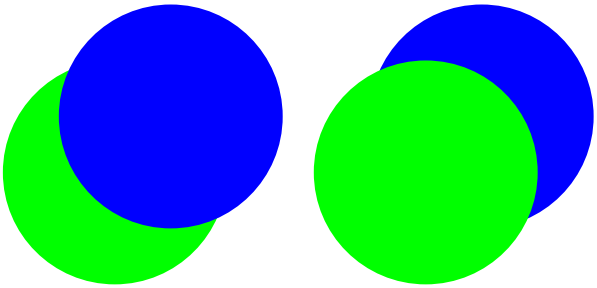

```
la = {{-1, 0}, {2, 0}, {3.5`, 3.5`}, {1.5`, 2.5`}, {-2, 4}, {-1, 0}};
ListPlot[la, Joined → True, PlotStyle → Thickness[0.01`], AxesLabel → {"u", "v"},
 Epilog → {Text["a1", {-0.7`, 0.3`}], Text["a2", {1.8`, 0.3`}],
   Text["a3", {2.7`, 2.7`}], Text["a4", {1.5`, 2.2`}], Text["a5", {-1.6`, 3.4`}]}]
```




```
p1 = Graphics[{Blue, Circle[{0, 0}], Red, Arrow[{{2, 1}, {1, 0}}]}];
p2 = Graphics[
   {Green, Thickness[0.02], Circle[{0, 0}], Red, Arrow[{{2, 1}, {1, 0}}, 0.2]}];
GraphicsRow[{p1, p2}]
```



```
p1 = Graphics[{RGBColor[0, 1, 0], Disk[{0, 0}, 2]}, ImageSize → 150];
p2 = Graphics[{RGBColor[0, 0, 1], Disk[{1, 1}, 2]}, ImageSize → 150];
p3 = Show[p1, p2, AspectRatio → Automatic];
p4 = Show[p2, p1, AspectRatio → Automatic];
GraphicsRow[{p3, p4}]
```



```
p1 = Grid[{{Graphics[Disk[], ImageSize → 30, BaselinePosition → Bottom], abc},
    {ead, Graphics[Rectangle[], ImageSize → 30, BaselinePosition → Top]}}, Frame →
    All];
p2 = Framed[p1];
GraphicsRow[{p1, p2}]
```


```
p1 = Grid[Table[x, {3}, {3}], Frame → All, Spacings → 2];
p2 = Grid[Table[x, {3}, {3}], Frame → All, Spacings → {2, 0}];
GraphicsRow[{p1, p2}]
```

6.3.1.1 Arrows

Arrow

Arrow

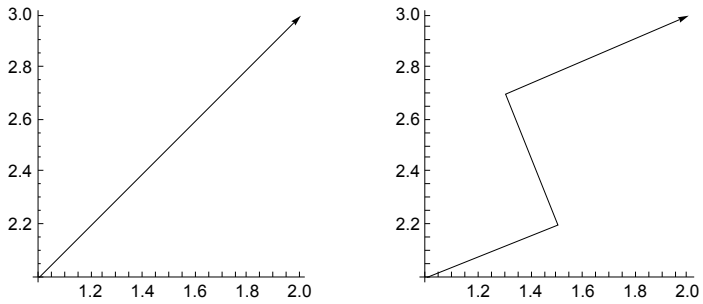
`Arrow[{pt1, pt2}]`
is a graphics primitive which represents an arrow from pt_1 to pt_2.

`Arrow[{pt1, pt2}, s]`
represents an arrow with its ends set back from pt_1 and pt_2 by a distance s.

`Arrow[{pt1, pt2}, {s1, s2}]`
sets back by s_1 from pt_1 and s_2 from pt_2.

`Arrow[{pt1, pt2}]` is drawn by default with its tail at pt_1 and its head at pt_2.
`Arrow[{pt1, pt2, ...}]` represents an arrow whose shaft passes through the sequence of points pt_i.

```
p1 = Graphics[Arrow[{{1, 2}, {2, 3}}], Axes -> True];
p2 = Graphics[Arrow[{{1, 2}, {1.5, 2.2}, {1.3, 2.7}, {2, 3}}], Axes -> True];
GraphicsRow[{p1, p2}, Spacings → {100, 0}]
```



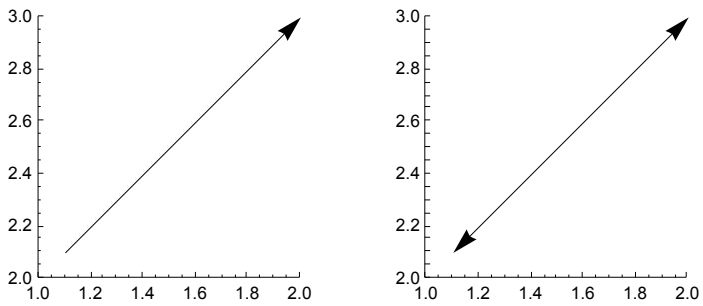

```
? Arrowheads
```

Arrowheads[*spec*] is a graphics directive specifying that arrows that
follow should have arrowheads with sizes, positions, and forms specified by *spec*. ≫

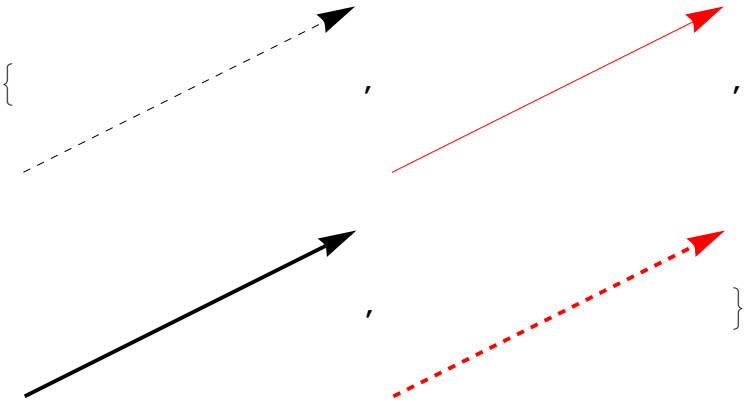
`Arrowheads[s]` specifies that arrowheads should have a length that is a fraction s of the total width of the graphic. The default is 0.04. »
`Arrowheads[{-s, s}]` gives double-headed arrows.

The symbolic values `Tiny`, `Small`, `Medium` and `Large` can be used for s. With these values, the size of the arrowhead is independent of the total width of the graphic. »

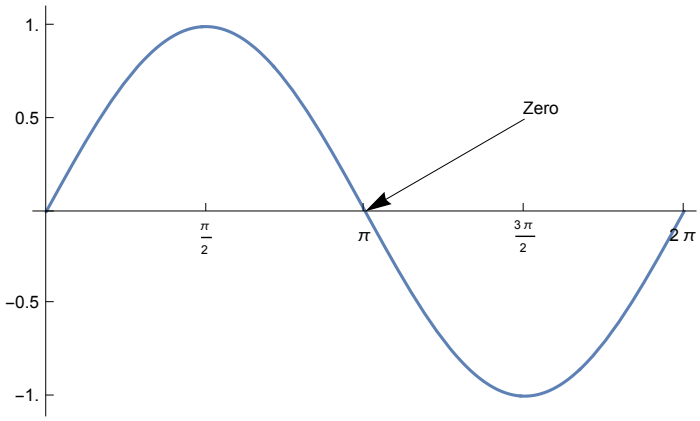
```
p1 = Graphics[{Arrowheads[0.1], Arrow[{{1.1, 2.1}, {2, 3}}]},
    Axes -> True, PlotRange -> {{1, 2}, {2, 3}}];
p2 = Graphics[{Arrowheads[{-0.1, 0.1}], Arrow[{{1.1, 2.1}, {2, 3}}]},
    Axes -> True, PlotRange -> {{1, 2}, {2, 3}}];
GraphicsRow[{p1, p2}, Spacings → {100, 0}]
```



```
a = {Arrowheads[Large], Arrow[{{0, 0}, {1, .5}}]};

{Graphics[{Dashed, a}], Graphics[{Red, a}],
 Graphics[{Thick, a}], Graphics[{Thick, Dashed, Red, a}]}
```



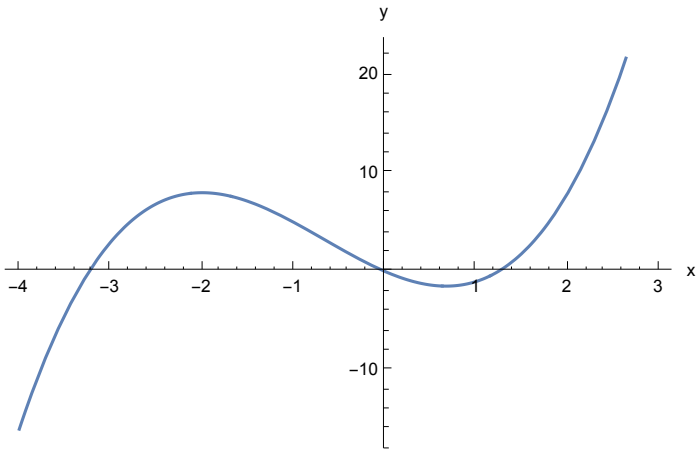
```
Plot[Sin[x], {x, 0, 2 Pi}, Ticks → {π {1 / 2, 1, 3 / 2, 2}, Range[-1, 1, 0.5]}, Epilog →
  {Arrow[{{3 Pi / 2, 1 / 2}, {Pi, 0}}], Text["Zero", {3 Pi / 2, 1 / 2}, {-1, -1}]}]
```



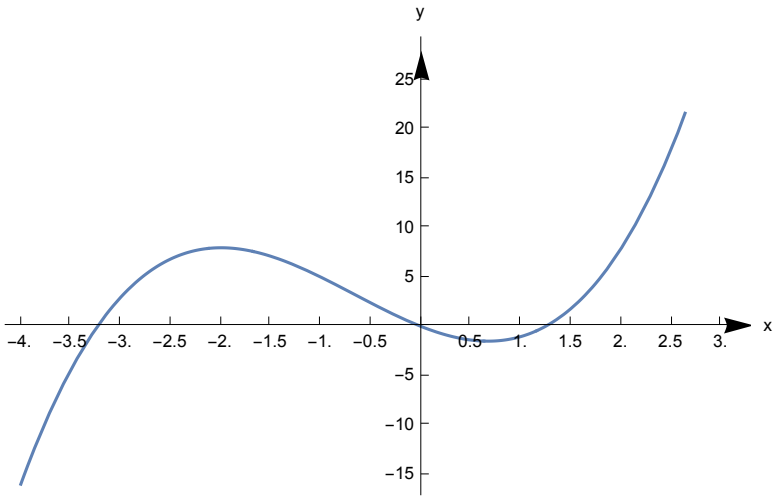
6.3.1.2 Arrows applied to the coordinate axes

When applying arrows to the coordinate axes, it is necessary to choose the lengths of the arrows, the Arrowhead,
the PlotRange and the Ticks very carefully by trial and error in accordance with the ImageSize.

```
pl = Plot[x^3 + 2 x^2 - 4 x, {x, -4, 3}, AxesLabel → {"x", "y"}]
```



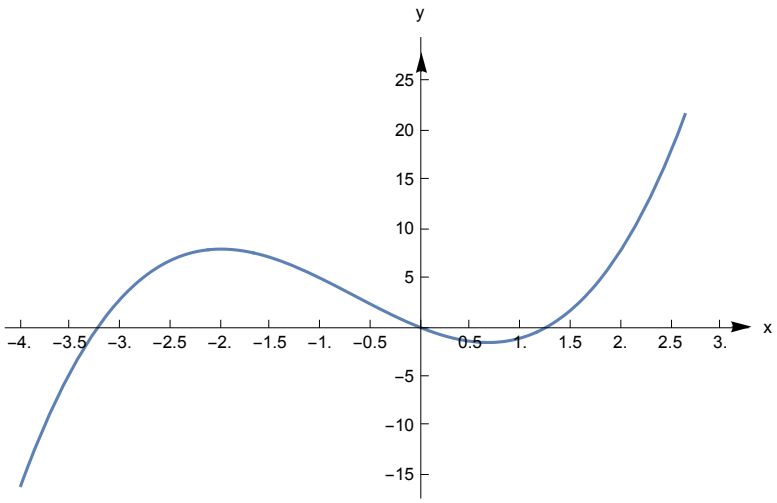
```
Show[pl,
 Graphics[{Arrow[{{3.15, 0.12}, {3.32, 0.12}}], Arrow[{{0, 26}, {0, 28}}]}],
 PlotRange → {{-4, 3.13}, {-15, 27}},
 Ticks → {Range[-4, 3, 0.5], Range[-30, 30, 5]}, ImageSize → 400]
```




```
Show[pl,
  Epilog → {Arrow[{{3.15, 0.12}, {3.32, 0.12}}], Arrow[{{0, 26}, {0, 28}}]},
  PlotRange → {{-4, 3.13}, {-15, 27}},
  Ticks → {Range[-4, 3, 0.5], Range[-30, 30, 5]}, ImageSize → 600];
```

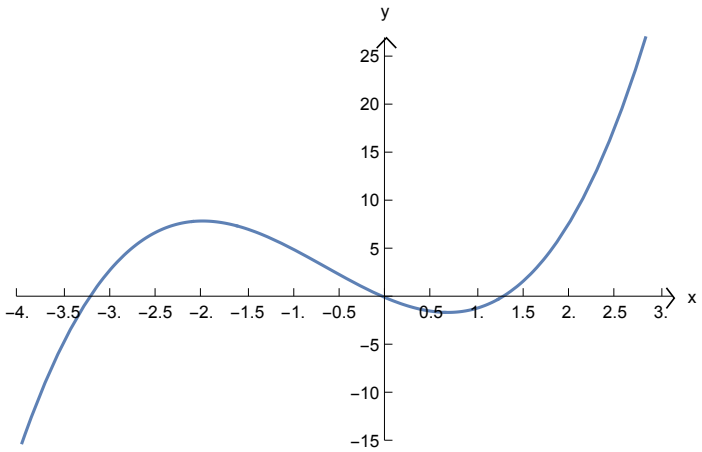
gives the same picture as above.

```
Show[pl, Graphics[{Arrowheads[0.028],
   Arrow[{{3.1, 0.12}, {3.3, 0.12}}], Arrow[{{0, 26}, {0, 28}}]}],
 PlotRange → {{-4, 3.13}, {-15, 27}},
 Ticks → {Range[-4, 3, 0.5], Range[-30, 30, 5]}, ImageSize → 400]
```



Another method is to construct arrow from lines (see the Epilog).

```
Plot[x^3 + 2 x^2 - 4 x, {x, -4, 3},
 AxesLabel → {"x", "y"}, PlotRange → {{-4, 3.125}, {-15, 27}},
 Ticks → {Range[-4, 3, 0.5], Range[-30, 30, 5]},
 Epilog → {Black, Thickness[0.0025], Line[{{3.05, 1}, {3.13, 0}, {3.05, -1}}],
   Line[{{0.1, 26}, {0, 27}, {-0.1, 26}}]}]
```



6.3.2 Three-Dimensional Graphics Elements

```
?? Graphics3D
```

Graphics3D[*primitives*, *options*] represents a three-dimensional graphical image. ≫

```
Attributes[Graphics3D] = {Protected, ReadProtected}

Options[Graphics3D] =
 {AlignmentPoint → Center, AspectRatio → Automatic, AutomaticImageSize → False,
  Axes → False, AxesEdge → Automatic, AxesLabel → None, AxesOrigin → Automatic,
  AxesStyle → {}, Background → None, BaselinePosition → Automatic, BaseStyle → {},
  Boxed → True, BoxRatios → Automatic, BoxStyle → {}, ClipPlanes → None,
  ClipPlanesStyle → Automatic, ColorOutput → Automatic, ContentSelectable → Automatic,
  ControllerLinking → Automatic, ControllerMethod → Automatic,
  ControllerPath → Automatic, CoordinatesToolOptions → Automatic,
  DisplayFunction :→ $DisplayFunction, Epilog → {}, FaceGrids → None, FaceGridsStyle → {},
  FormatType :→ TraditionalForm, ImageMargins → 0., ImagePadding → All,
  ImageSize → Automatic, ImageSizeRaw → Automatic, LabelStyle → {}, Lighting → Automatic,
  Method → Automatic, PlotLabel → None, PlotRange → All, PlotRangePadding → Automatic,
  PlotRegion → Automatic, PreserveImageOptions → Automatic, Prolog → {},
  RotationAction → Fit, SphericalRegion → False, Ticks → Automatic,
  TicksStyle → {}, TouchscreenAutoZoom → False, ViewAngle → Automatic,
  ViewCenter → Automatic, ViewMatrix → Automatic, ViewPoint → {1.3, -2.4, 2.},
  ViewRange → All, ViewVector → Automatic, ViewVertical → {0, 0, 1}}
```

Point[{x,y,z}]	point at position **{x,y,z}**
Line[*listp*]	line through points given in ***listp*** with
***listp* = {x1,y1,z1},{x2,y2,z2},..}**	
Text[*expr*, {x,y,z}]	the text of ***expr***, centered at **{x,y,z}**

? Point

Point[p] is a graphics and geometry primitive that represents a point at p.
Point[{p_1, p_2, ...}] represents a collection of points. ≫

? Line

Line[{p_1, p_2, ...}] represents the line segments joining a sequence for points p_i.
Line[{{p_{11}, p_{12}, ...}, {p_{21}, ...}, ...}] represents a collection of lines. ≫

? Text

Text[*expr*] displays with *expr* in plain text format.
Text[*expr*, *coords*] is a graphics primitive that
displays the textual form of *expr* centered at the point specified by *coords*. ≫

? Framed

Framed[*expr*] displays a framed version of *expr*. ≫

The graphics elements give drawings by the command **Show[Graphics3D[]]** with the graphics elements as arguments. Style options must be included at the very first position of the list for a graphics element.

? Arrow

Arrow[{pt_1, pt_2}] is a graphics primitive that represents an arrow from pt_1 to pt_2.
Arrow[{pt_1, pt_2}, s] represents an arrow with its ends set back from pt_1 and pt_2 by a distance s.
Arrow[{pt_1, pt_2}, {s_1, s_2}] sets back by s_1 from pt_1 and s_2 from pt_2.
Arrow[*curve*, ...] represents an arrow following the specified *curve*. ≫

? Cylinder

Cylinder[{{x_1, y_1, z_1}, {x_2, y_2, z_2}}, r] represents a cylinder of radius r around the line from (x_1, y_1, z_1) to (x_2, y_2, z_2).
Cylinder[{{x_1, y_1, z_1}, {x_2, y_2, z_2}}] represents a cylinder of radius 1. ≫

? Sphere

Sphere[p] represents a unit sphere centered at the point p.
Sphere[p, r] represents a sphere of radius r centered at the point p.
Sphere[{p_1, p_2, ...}, r] represents a collection of spheres of radius r. ≫

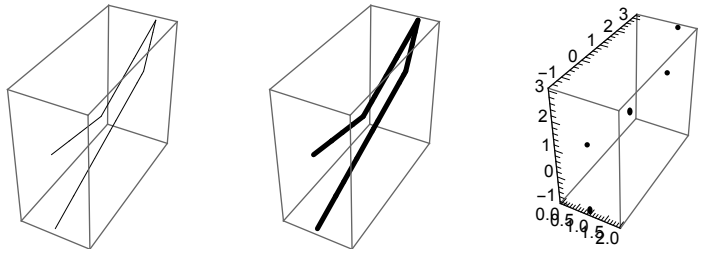
? Cuboid

Cuboid[p_{min}] represents a unit hypercube with its lower corner at p_{min}.
Cuboid[p_{min}, p_{max}] represents an axis-aligned filled cuboid with lower corner p_{min} and upper corner p_{max}. ≫

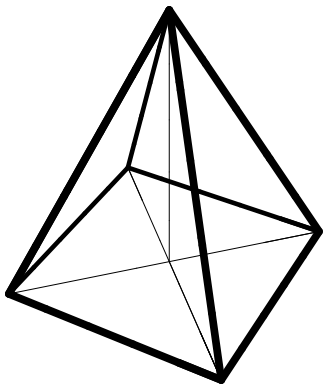
? Tube

Tube[{{x_1, y_1, z_1}, {x_2, y_2, z_2}, ...}] represents a 3D tube around the line joining a sequence of points.
Tube[{pt_1, pt_2, ...}, r] represents a tube of radius r.
Tube[{{pt_{11}, pt_{12}, ...}, {pt_{21}, ...}, ...}, ...] represents a collection of tubes.
Tube[*curve*, ...] represents a tube around the specified 3D curve. ≫

```
l3 = {{0, 0, 0}, {1, 1, 1}, {1.5`, 3.2`, 3}, {2, 1.4`, 2.5`}, {1, -1.5`, -1}};
p1 = Show[Graphics3D[Line[l3]]];
p2 = Show[Graphics3D[{Thickness[0.03`], Line[l3]}]];
l4 = Table[Graphics3D[{PointSize[0.04`], Point[l3[[k]]]}], {k, Length[l3]}];
p3 = Show[l4, Axes → True];
Show[GraphicsRow[{p1, p2, p3}], ImageSize → 400]
```



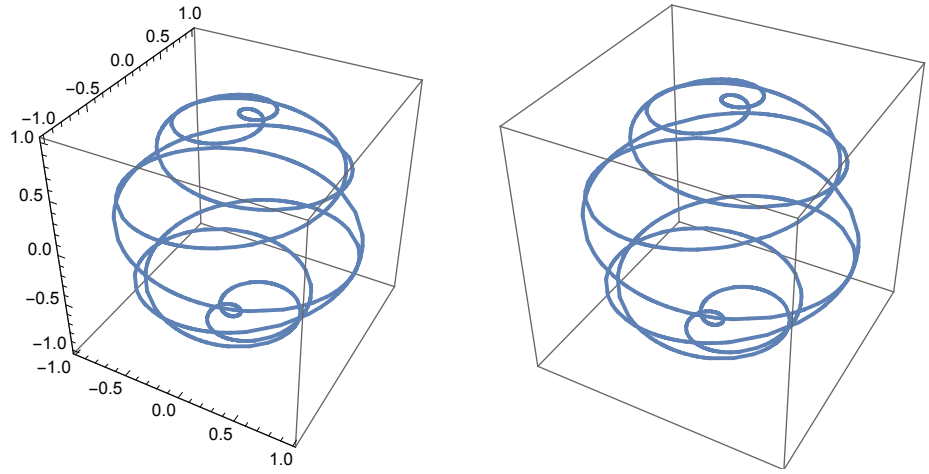

```
Clear[a, z0, b, e, bx, x0]
a = 4;
z0 = 5;
b[1] = {0, 0, 0};
b[2] = {a, 0, 0};
b[3] = {a, a, 0};
b[4] = {0, a, 0};
x0 = a/2; e = {x0, x0, z0}; bx = {x0, x0, 0};
lp1 = {Thickness[0.02`], Line[{b[2], b[3], e, b[1], e, b[2], b[1]}]};
lp2 =
  {Thickness[0.012`], Line[{b[1], b[4]}], Line[{b[4], b[3]}], Line[{e, b[4]}]};
lp3 = {Line[{b[1], b[3]}], Line[{b[4], b[2], bx, e}]};
Show[Graphics3D[lp1], Graphics3D[lp2],
 Graphics3D[lp3], Boxed → False, ImageSize → 200]
```



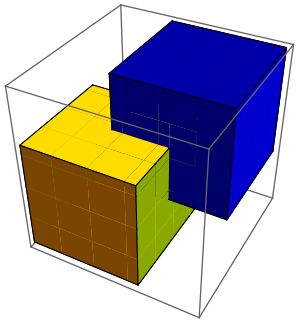
```
p1 = ParametricPlot3D[
    {Sin[8 u] Sin[u], Cos[8 u] Sin[u], Cos[u]}, {u, 0, 2 π}, PlotPoints → 200];
```

The first element of the Graphics3D output contains the information on the curve(s. beginning of 6.3);
this may be used for another graphic where the layout of the curve even may be modified.

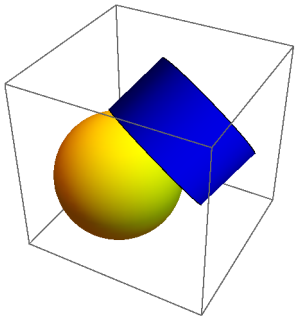
```
p2 = Show[Graphics3D[{Thickness[0.015`], First[p1]}]];
Show[GraphicsRow[{p1, p2}], ImageSize → 500]
```



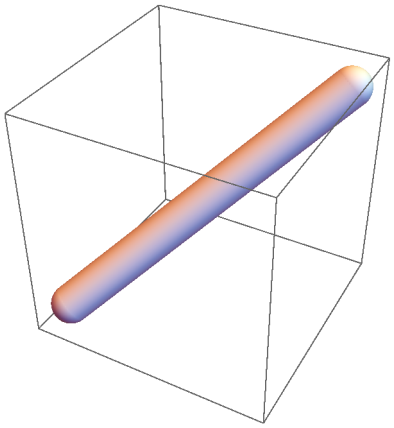

```
Graphics3D[{Yellow,Cuboid[{0,0,0}],Blue,Cuboid[{0.5,0.5,0.5}]}, ImageSize ->  150]
```



```
Graphics3D[{Yellow,Sphere[{0,0,0}],Blue,Cylinder[{{0.5,0.25,0.36},{1,1,1}}]}, ImageSize
```



```
Graphics3D[Tube[{{0, 0, 0}, {1, 1, 1}}, .1], ImageSize → 200]
```



6.3.2.3 I Arrows

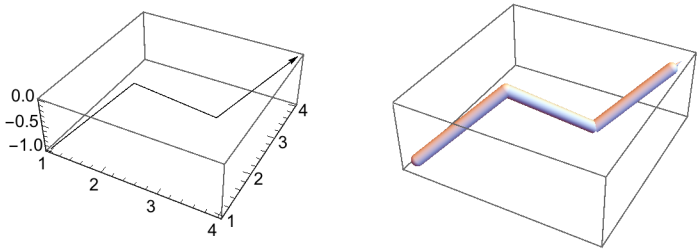
Arrow

`Arrow[{`pt_1`, `pt_2`}]`
is a graphics primitive which represents an arrow from pt_1 to pt_2.

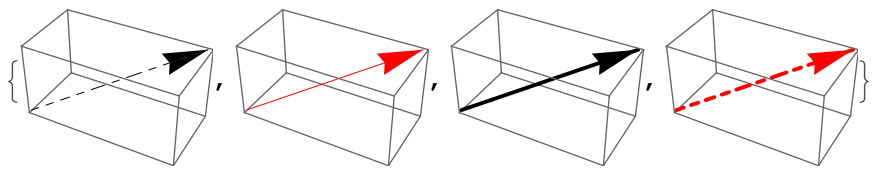
`Arrow[{`pt_1`, `pt_2`}, `s`]`
represents an arrow with its ends set back from pt_1 and pt_2 by a distance s.

`Arrow[{`pt_1`, `pt_2`}, {`s_1`, `s_2`}]`
sets back by s_1 from pt_1 and s_2 from pt_2.

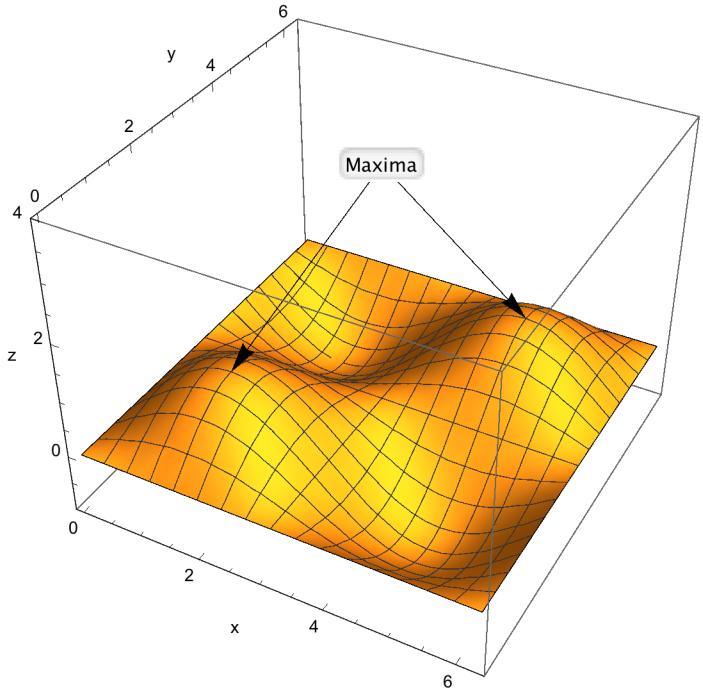
```
p1 =
  Graphics3D[Arrow[{{1, 1, -1}, {2, 2, 0}, {3, 3, -1}, {4, 4, 0}}], Axes → True];
p2 = Graphics3D[Arrow[Tube[{{1, 1, -1}, {2, 2, 0}, {3, 3, -1}, {4, 4, 0}}, .1]]] ;
GraphicsRow[{p1, p2}, Spacings → 100]
```




```
a = {Arrowheads[Large], Arrow[{{0, 0, 0}, {2, 1, 1}}]};
{Graphics3D[{Dashed, a}, ImageSize → 100],
 Graphics3D[{Red, a}, ImageSize → 100], Graphics3D[{Thick, a}, ImageSize → 100],
 Graphics3D[{Thick, Dashed, Red, a}, ImageSize → 100]}
```



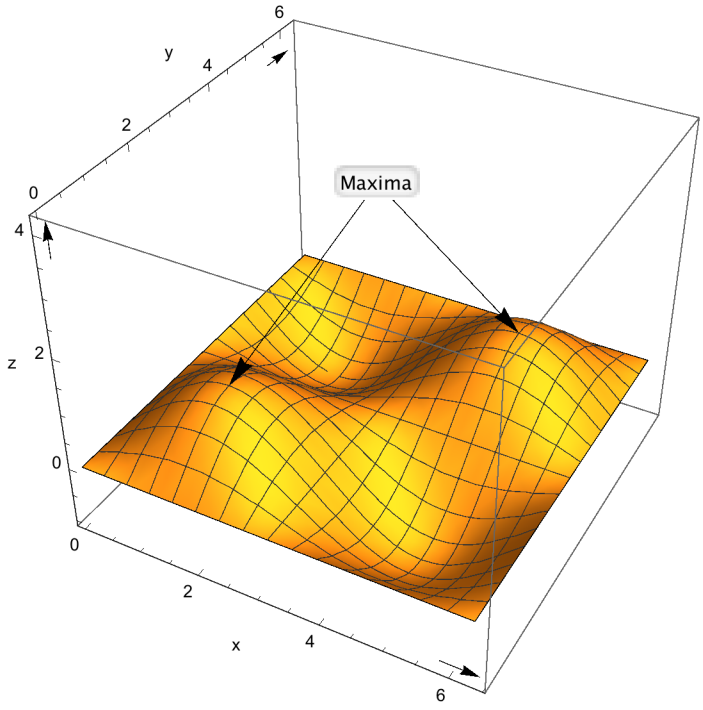
```
pl = Show[Plot3D[Sin[x] Sin[y], {x, 0, 2 Pi},
    {y, 0, 2 Pi}, PlotRange → {-1, 4}, BoxRatios → Automatic,
    AxesLabel → {"x", "y", "z"}], Graphics3D[
    {Arrow[{{Pi, Pi, 4}, {Pi / 2, Pi / 2, 1}}],
     Arrow[{{Pi, Pi, 4}, {3 Pi / 2, 3 Pi / 2, 1}}],
     Text[Panel["Maxima", FrameMargins → 0], {Pi, Pi, 4}]}]]
```

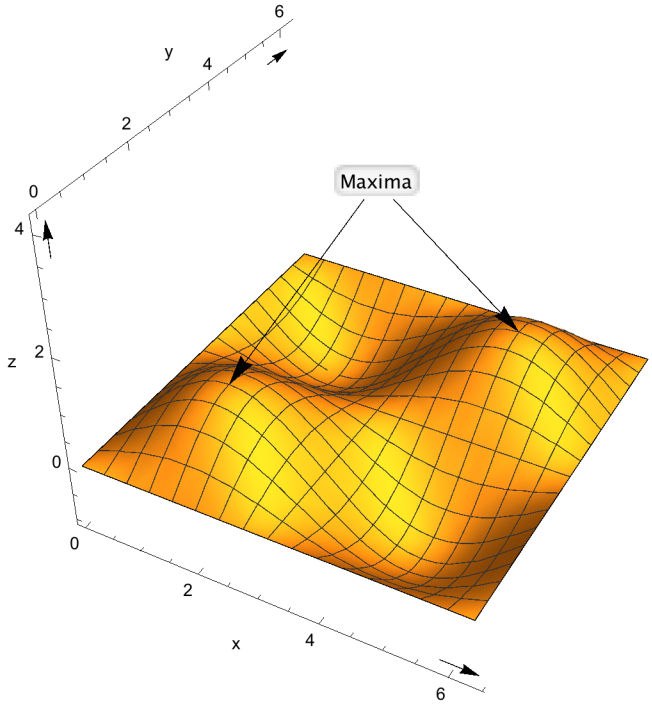


6.3.2.2 Arrows applied to the coordinate axes

I did not succed in drawing the arrqws as an extension of the lines supporting the ticks.

```
aa = Graphics3D[{Arrowheads[0.025],
    Arrow[{{5.8, 0, -1.}, {6.4, 0, -1.}}], Arrow[{{0, 0, 3.6}, {0, 0, 4.2}}],
    Arrow[{{0, 5.4, 4}, {0, 6., 4}}]}];
p1 = Show[pl, aa, PlotRange → All]
p2 = Show[pl, aa, Boxed → False, PlotRange → All]
```





6.3.3 Exercises

3D Graphics

Not all exercises are obligatory; Obligatory ones have a cross (+) after the number.

6.20+ Plot the space curve x = t cos(t), y = t sin(t), z = 2 t for 0 <= t <= 4π.

6.21 a) Plot the ellipsoid $(x/2)^2 + y^2 + (z/3)^2 = 1$.

b) Show contour lines for the upper half of this surface, i.e. for z > 0.

Hint: Use spherical coordinates for x/2, y, z/3.

6.22 Plot the modulus of sin z for z taken from the strip of length 3π and width 2 around the origin. Show also the contour lines.

6.23+ Show the modulus(= absolute value) of the integrand of Sommerfeld's integral representation of the Bessel function $J_1(r)$ for r = 1 and r = 10.
Display the saddle point as a surface and by contour lines.

6.24 Draw a red screw with blue box and green axes; these should have labels x,y,z.
The background should be yellow.

6.25 a) Show the surface displaying the modulus of the function sin(x + i y) for
$-\pi$ 1.2 <= x <= π 1.2, -1.1 <= y <= 1.1.
The faces facing the viewer should contain gridlines. The x -axis should have labels x, $-\pi$, π, the y-axis y, -1,1.

b) Show also the surface with a blue mesh. The other parts of the drawing should be in red.

c) Limit the above surface to the range (0,1.1). "Wounds" resulting from removals should be coloured in yellow.

6.26+ a) Plot a hollow cylinder (x = r cos(u), y = r sin(u), z = v; v = -h/2, h/2; h = 5, r = 1).

b) Besides the picture resulting from the default values, show a view head-on to the mantle, a top view, a view from an excentric point chosen such that the opposite generator and a bit of the floor are visible.

c) Show the 4 pictures designed such that the real magnitude of the cylinder is the same in each of them and arrange them again in a 2 x 2 array.

6.27 Draw a circular cone. The visible part of the basis and the two generators defining the outline should be 1.2 mm thick. Draw the height and a radius; attach the letters h, r respectively to the height, the radius respectively. Show also two other thin generators and two semi-circles at heights h/3 and 2h/3 in the visible part of the lateral surface; in the back these circles should be closed by dashed semi-circles; there should be two dashed generates corresponding to those drawn by continuous lines. There should be a letter A (= apex) above the tip of the cone.

6.29 Using the list **OldColorNames** determine which color names *Mathematica* recognizes as long as the list has not been introduced into the kernel. The command should prepare a drawing (a disk or a square or a rectangle) colored by the color and have a plotlabel giving the name of the color. For the colors not implemented into the kernel one gets a black figure. Note that the color names given in the list are strings. They can be changed into expressions by the command **ToExpression[].**