Grafiken

■ Einfache Grafiken aus Primitiven erstellen

▪ Ein Gitter

```
ClearAll["Global`*"]

GitterPunkte[P0_, dx_, dy_, nx_Integer, ny_Integer] :=
 Table[P0 + {i * dx, j * dy}, {i, 0, nx}, {j, 0, ny}]

g = GitterPunkte[{-.3, -.3}, .1, .1, 5, 7];
g // Chop // MatrixForm
```

$$\begin{pmatrix} \begin{pmatrix} -0.3 \\ -0.3 \end{pmatrix} & \begin{pmatrix} -0.3 \\ -0.2 \end{pmatrix} & \begin{pmatrix} -0.3 \\ -0.1 \end{pmatrix} & \begin{pmatrix} -0.3 \\ 0 \end{pmatrix} & \begin{pmatrix} -0.3 \\ 0.1 \end{pmatrix} & \begin{pmatrix} -0.3 \\ 0.2 \end{pmatrix} & \begin{pmatrix} -0.3 \\ 0.3 \end{pmatrix} & \begin{pmatrix} -0.3 \\ 0.4 \end{pmatrix} \\ \begin{pmatrix} -0.2 \\ -0.3 \end{pmatrix} & \begin{pmatrix} -0.2 \\ -0.2 \end{pmatrix} & \begin{pmatrix} -0.2 \\ -0.1 \end{pmatrix} & \begin{pmatrix} -0.2 \\ 0 \end{pmatrix} & \begin{pmatrix} -0.2 \\ 0.1 \end{pmatrix} & \begin{pmatrix} -0.2 \\ 0.2 \end{pmatrix} & \begin{pmatrix} -0.2 \\ 0.3 \end{pmatrix} & \begin{pmatrix} -0.2 \\ 0.4 \end{pmatrix} \\ \begin{pmatrix} -0.1 \\ -0.3 \end{pmatrix} & \begin{pmatrix} -0.1 \\ -0.2 \end{pmatrix} & \begin{pmatrix} -0.1 \\ -0.1 \end{pmatrix} & \begin{pmatrix} -0.1 \\ 0 \end{pmatrix} & \begin{pmatrix} -0.1 \\ 0.1 \end{pmatrix} & \begin{pmatrix} -0.1 \\ 0.2 \end{pmatrix} & \begin{pmatrix} -0.1 \\ 0.3 \end{pmatrix} & \begin{pmatrix} -0.1 \\ 0.4 \end{pmatrix} \\ \begin{pmatrix} 0 \\ -0.3 \end{pmatrix} & \begin{pmatrix} 0 \\ -0.2 \end{pmatrix} & \begin{pmatrix} 0 \\ -0.1 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0.1 \end{pmatrix} & \begin{pmatrix} 0 \\ 0.2 \end{pmatrix} & \begin{pmatrix} 0 \\ 0.3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0.4 \end{pmatrix} \\ \begin{pmatrix} 0.1 \\ -0.3 \end{pmatrix} & \begin{pmatrix} 0.1 \\ -0.2 \end{pmatrix} & \begin{pmatrix} 0.1 \\ -0.1 \end{pmatrix} & \begin{pmatrix} 0.1 \\ 0 \end{pmatrix} & \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix} & \begin{pmatrix} 0.1 \\ 0.2 \end{pmatrix} & \begin{pmatrix} 0.1 \\ 0.3 \end{pmatrix} & \begin{pmatrix} 0.1 \\ 0.4 \end{pmatrix} \\ \begin{pmatrix} 0.2 \\ -0.3 \end{pmatrix} & \begin{pmatrix} 0.2 \\ -0.2 \end{pmatrix} & \begin{pmatrix} 0.2 \\ -0.1 \end{pmatrix} & \begin{pmatrix} 0.2 \\ 0 \end{pmatrix} & \begin{pmatrix} 0.2 \\ 0.1 \end{pmatrix} & \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix} & \begin{pmatrix} 0.2 \\ 0.3 \end{pmatrix} & \begin{pmatrix} 0.2 \\ 0.4 \end{pmatrix} \end{pmatrix}$$

Polygon−Primitiv kann auch mit Liste von Punktlisten als Argument verwendet werden und ist in *Mathematica*−Code meist in dieser Form anzutreffen. Ist ein Design−Problem.. Besser wäre der Name **PolygonList** gewesen.

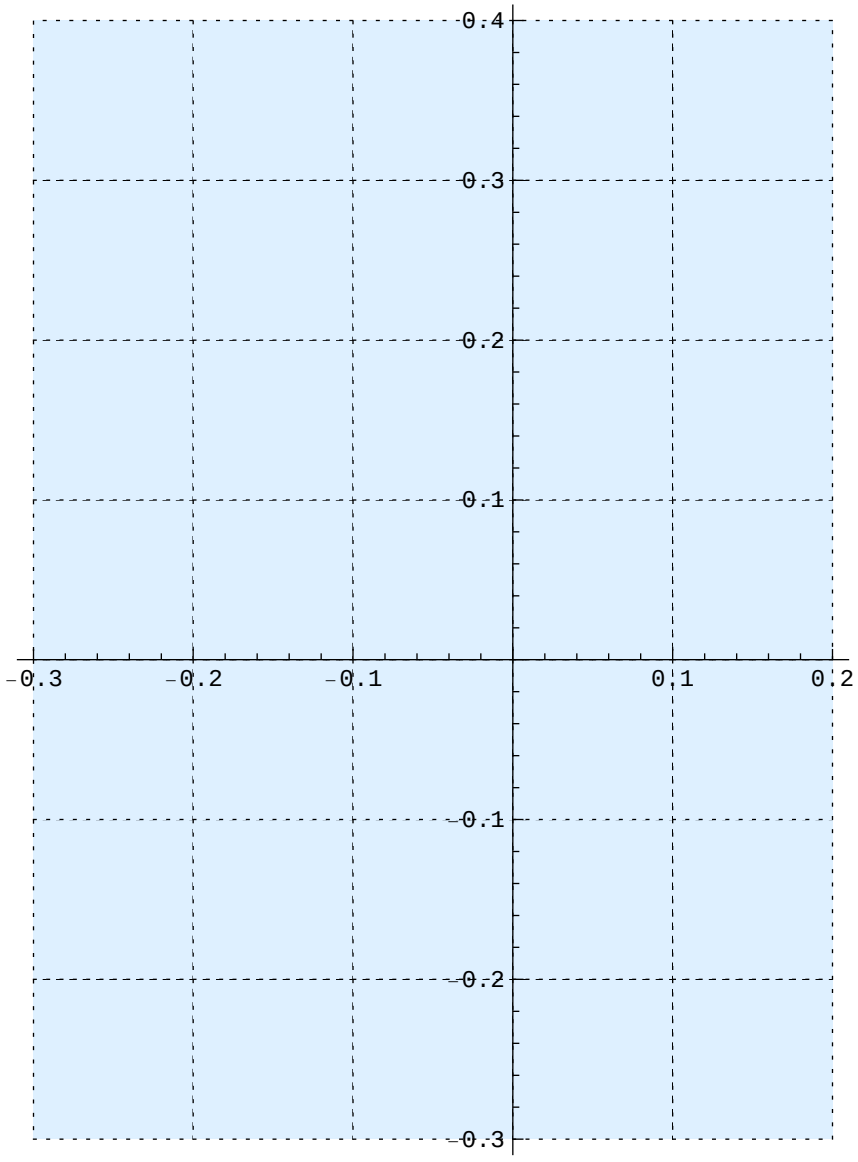
Wir verwenden deshalb nur einfache Polygone.

```
Gitter[g_] := Module[{d = Dimensions[g]},
  Join @@ Table[Polygon[{g[[i, j]], g[[i, j + 1]], g[[i + 1, j + 1]], g[[i + 1, j]]}],
    {i, d[[1]] - 1}, {j, d[[2]] - 1}]]

Short[(g1 = Gitter[g]), 7]
```

```
{Polygon[{{-0.3, -0.3}, {-0.3, -0.2}, {-0.2, -0.2}, {-0.2, -0.3}}],
 <<33>>, Polygon[{{0.1, 0.3}, {0.1, 0.4}, {0.2, 0.4}, {0.2, 0.3}}]}
```

Daraus eine 2D−Grafik erzeugen, mit grünen Flächen sowie kurz−gestrichelten schwarzen Kanten, gesteuert durch geeignete Grafikprimitive.

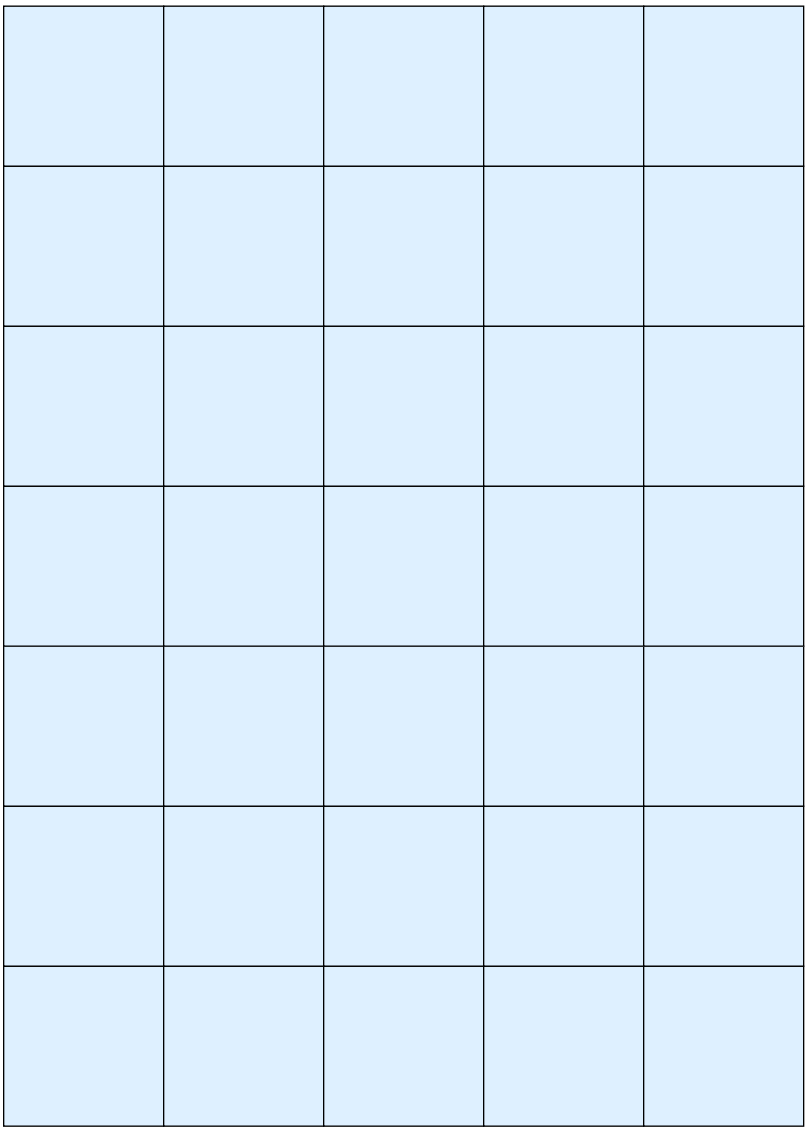
```
Graphics[{FaceForm[LightBlue], EdgeForm[{Dotted, Black}], g1}, Axes → True]
```



Dasselbe als **GraphicsComplex**−Objekt:

```
GitterComplex[P0_, dx_, dy_, nx_Integer, ny_Integer] :=
 GraphicsComplex[Join @@ Table[P0 + {i * dx, j * dy}, {i, 0, nx}, {j, 0, ny}] // Chop,
  Join @@ Table[Polygon[{(ny + 1) i + j + 1, (ny + 1) i + j + 2,
      (ny + 1) (i + 1) + j + 2, (ny + 1) (i + 1) + j + 1}], {i, 0, nx - 1}, {j, 0, ny - 1}]]
```

```
gc = GitterComplex[{-.3, -.3}, .1, .1, 5, 7];
Graphics[{FaceForm[LightBlue], EdgeForm[Black], gc}]
```



■ GitterPlot und ParametricPlot3D

g ist das Gitter der Parameterwerte, f: $R^2 \to R^3$ die darzustellende parametrisierte Fläche. **GitterPlot** erzeugt wieder ein komplexes Grafikprimitiv vom Typ **Polygon**.

Zu beachten ist, dass in unserer Anwendung f einstellig ist und als Argument einen Punkt $P \in R^2$ übergeben bekommt.

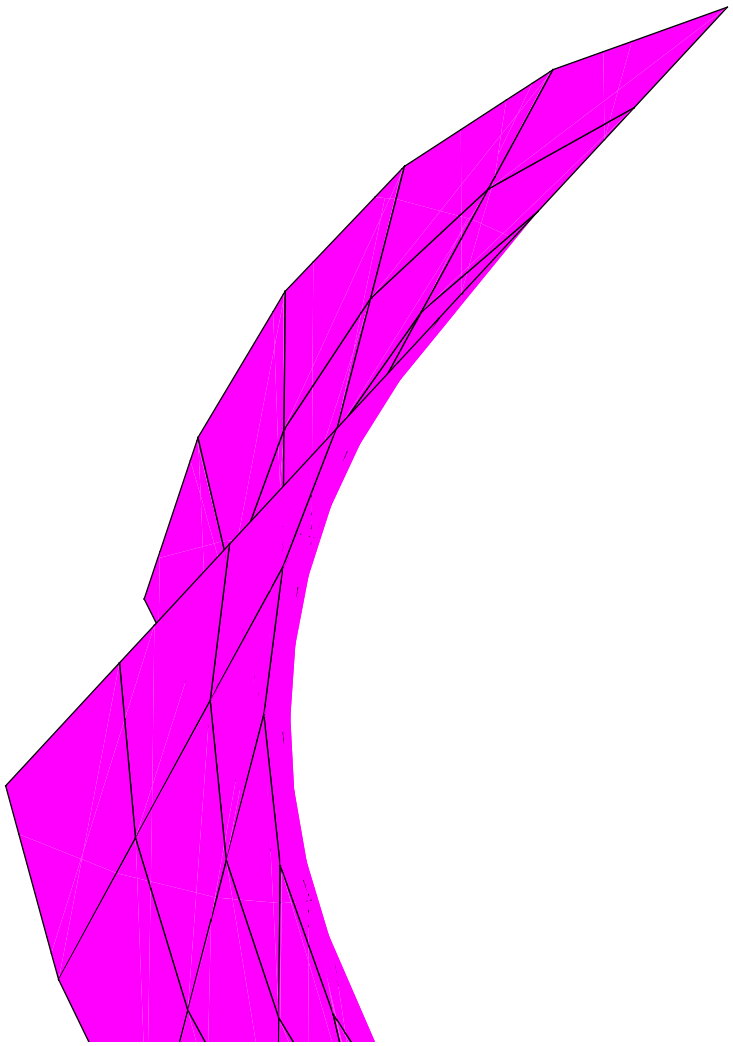
```
GitterPlot[f_, g_] := Module[{d = Dimensions[g]},
  Join @@ Table[Polygon[f /@ {g[[i, j]], g[[i, j + 1]], g[[i + 1, j + 1]], g[[i + 1, j]]}],
    {i, d[[1]] - 1}, {j, d[[2]] - 1}] ]
```

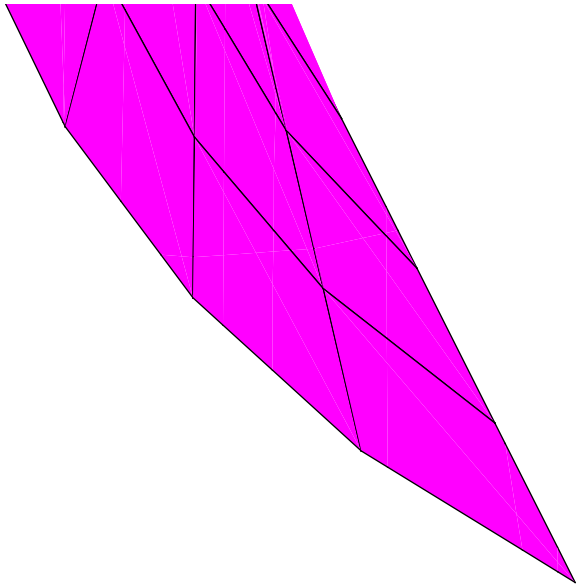
Anwendung : Die Tangentialfläche an die getwistete Kubik (t, t^2, t^3).

```
Clear[f]
f[{t_, u_}] := {t + u, t^2 + 2 t u, t^3 + 3 t^2 u}

g2 = GitterPlot[f, g];
```

Daraus wird mit entsprechenden Direktiven ein **Graphics3D**−Objekt erzeugt.
Hier eins, das in einem dunklen Raum in der Telekomfarbe glüht.

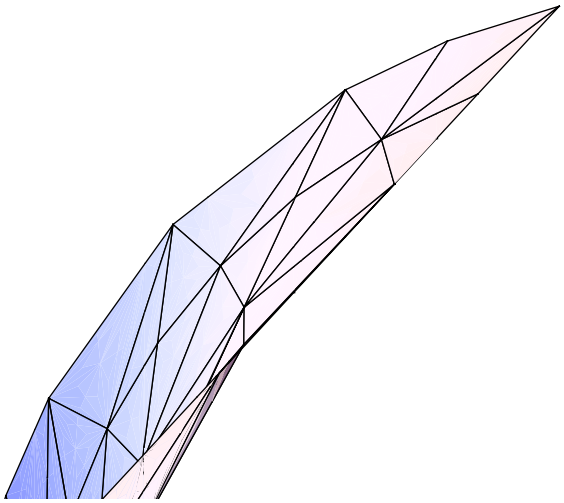
```
Graphics3D[{Glow[Magenta], g2},
 Lighting -> {Black}, Boxed -> False, ViewPoint → {2, 0, 3}]
```

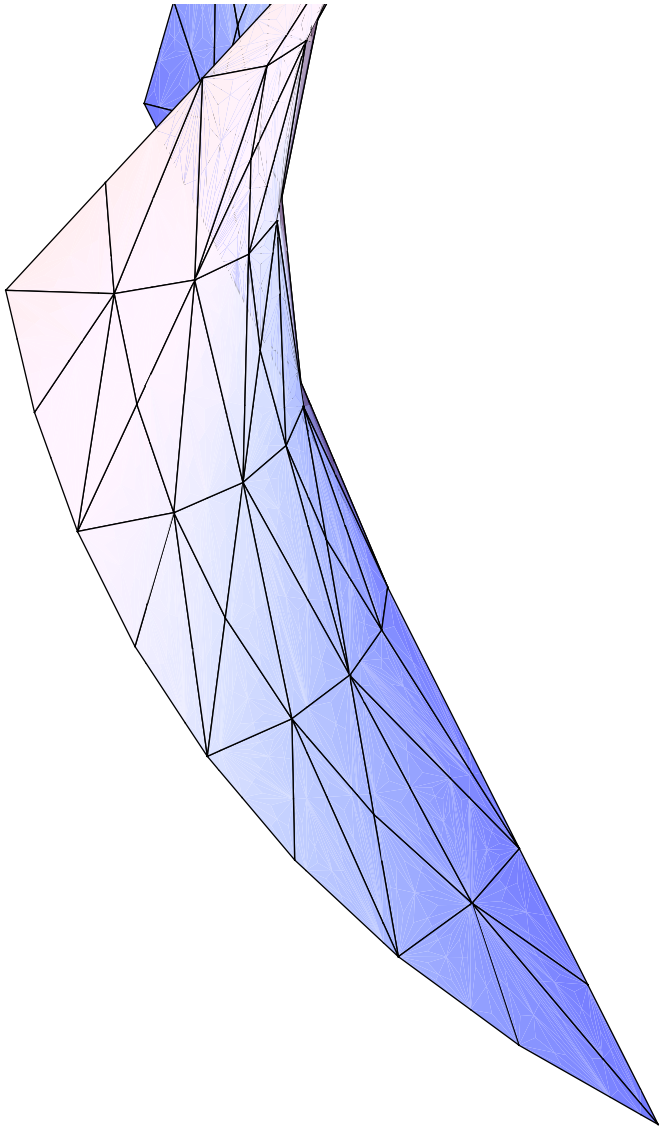




Und hier dasselbe Bild mit **ParametricPlot3D**.

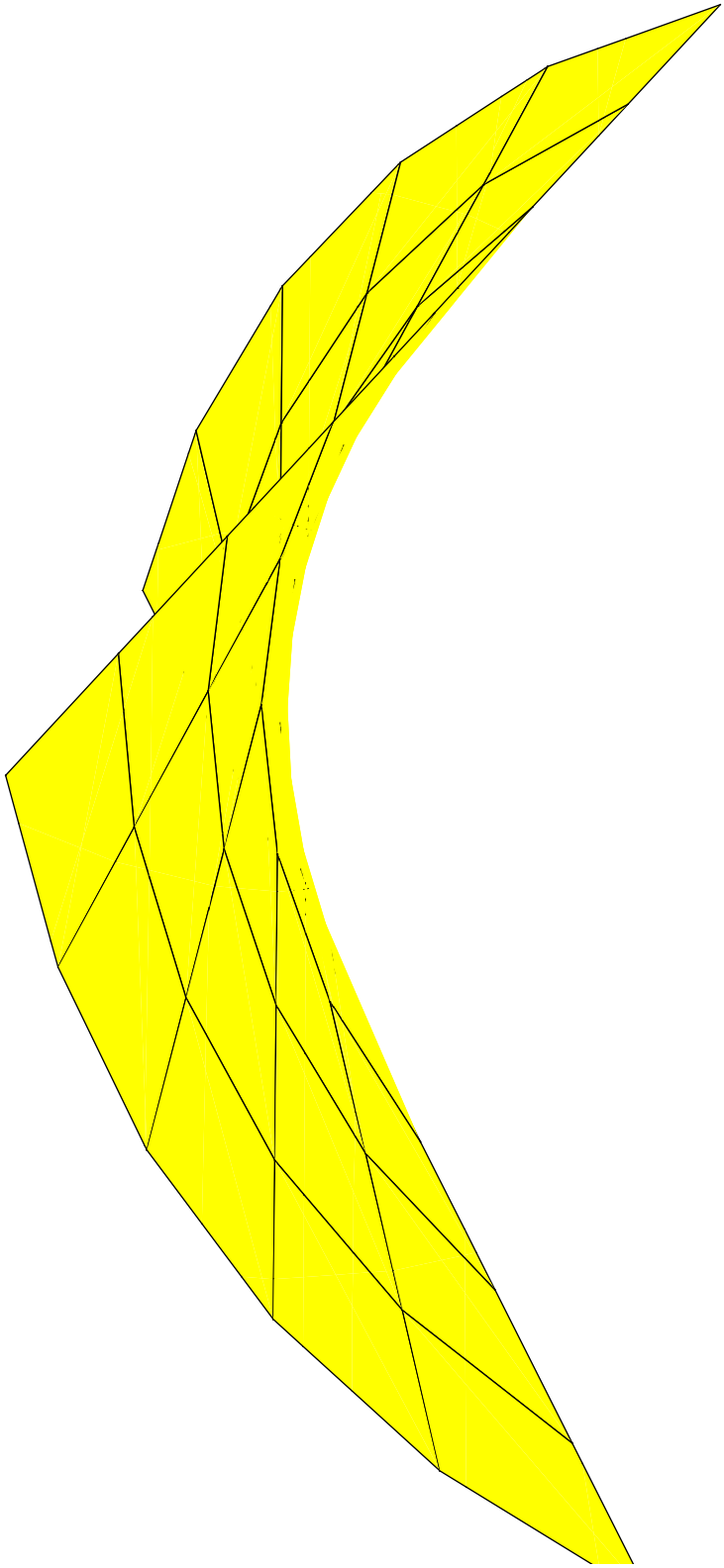
```
ParametricPlot3D[{t + u, t^2 + 2 t u, t^3 + 3 t^2 u}, {t, -.3, .2}, {u, -.3, .4}, PlotPoints -> 3,
 Mesh -> All, Boxed → False, Axes -> False, ViewPoint → {2, 0, 3}, PlotRange → All]
```





In einer Realisierung als **GraphicsComplex**−Objekt müssen nur die Punkte transformiert werden.
Sie brauchen weiter die Funktion **GitterComplex** aus dem letzten Abschnitt.

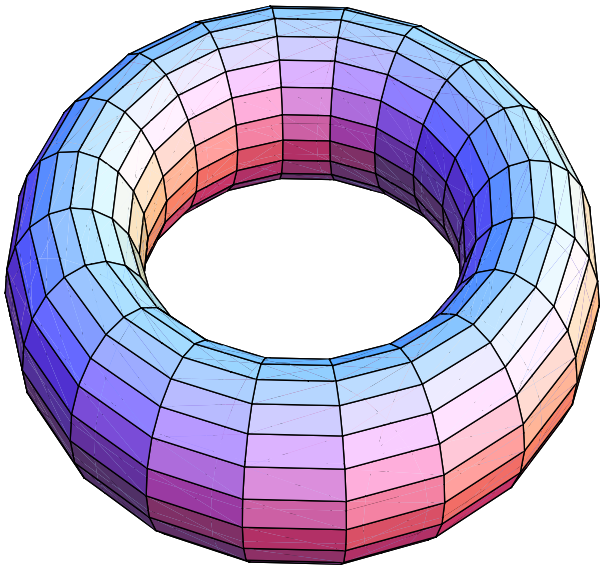
```
GitterComplexPlot[f_, P0_, dx_, dy_, nx_Integer, ny_Integer] :=
 With[{gc = GitterComplex[P0, dx, dy, nx, ny]}, GraphicsComplex[f /@ gc[[1]], gc[[2]]]]

g3 = GitterComplexPlot[f, {-.3, -.3}, .1, .1, 5, 7];
Graphics3D[{Glow[Yellow], g3},
 Lighting -> {Black}, Boxed -> False, ViewPoint → {2, 0, 3}]
```



Ein Torus

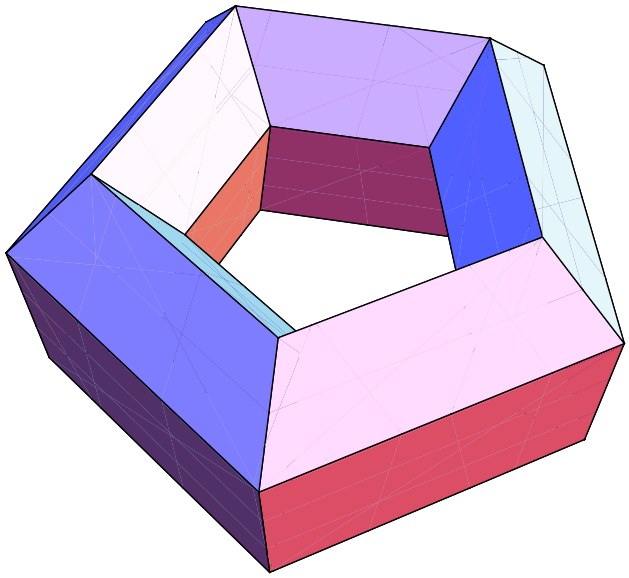
Ein Torus als GitterPlot.

```
Clear[tf, s, d]
tf[{u_, t_}] := Module[{r = {Cos[t], Sin[t], 0}, b = {0, 0, 1}},
  r + s r * Sin[u] + s b * Cos[u]]

d = 20; s = .3;
g4 = GitterPlot[tf, GitterPunkte[{0, 0}, 2 Pi / d, 2 Pi / d, d, d]];
Graphics3D[g4, Boxed → False, AspectRatio → 1]
```



Und als **GraphicsComplex**.

```
torusFunction[{t_, u_}, s_, d_] := Module[{r = {Cos[t], Sin[t], 0}, b = {0, 0, 1}},
  r + s r * Sin[u] + s b * Cos[u] // N]
Torus[s_, d_] :=
 GitterComplexPlot[torusFunction[#, s, d] &, {0, 0}, 2 Pi / d, 2 Pi / d, d, d]

g5 = Torus[.4, 5];
Graphics3D[g5, Boxed → False, AspectRatio → 1]
```

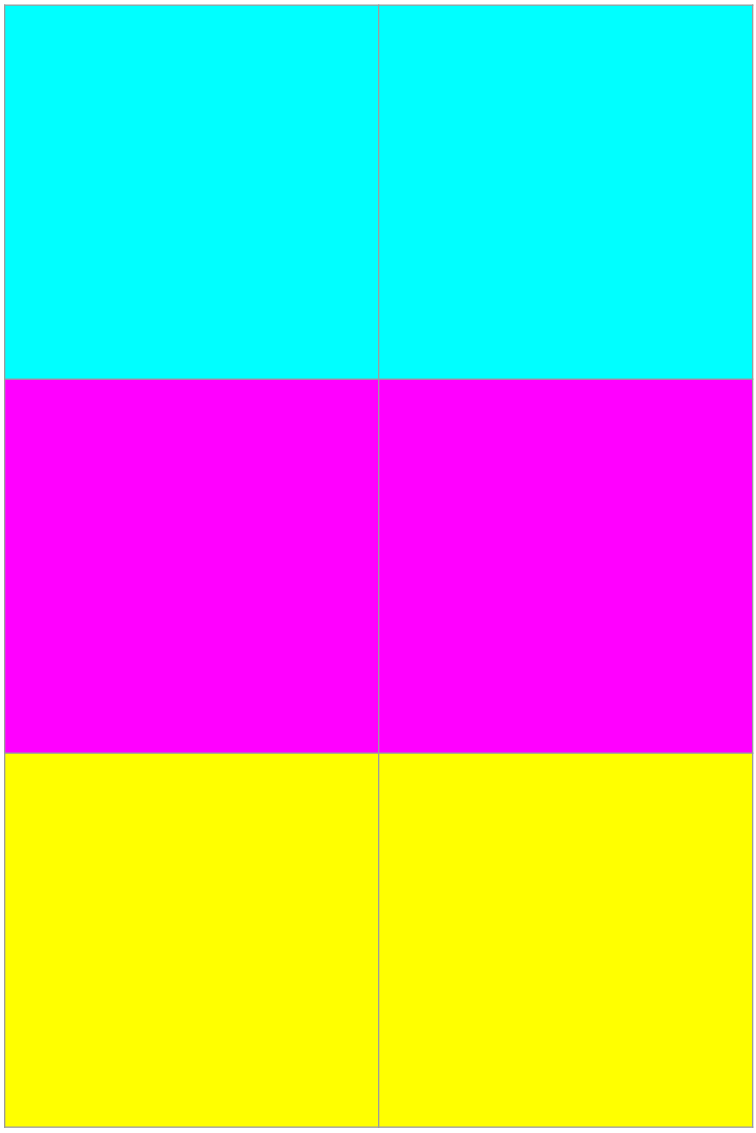


Die Farbsysteme von *Mathematica*

Die verschiedenen Farbsysteme

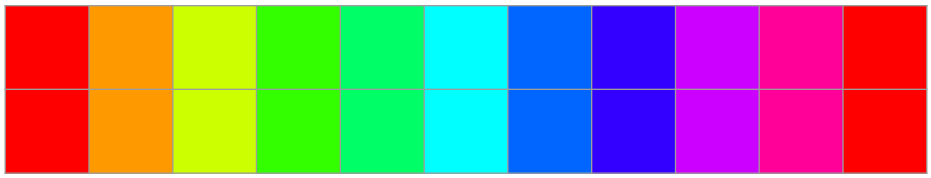
Die Farbsysteme RGB und CMYK entsprechen einander nicht genau. −− Nun doch.

```
ArrayPlot[{{Cyan, CMYKColor[1, 0, 0, 0]},
  {Magenta, CMYKColor[0, 1, 0, 0]}, {Yellow, CMYKColor[0, 0, 1, 0]}}, Mesh → True]
```



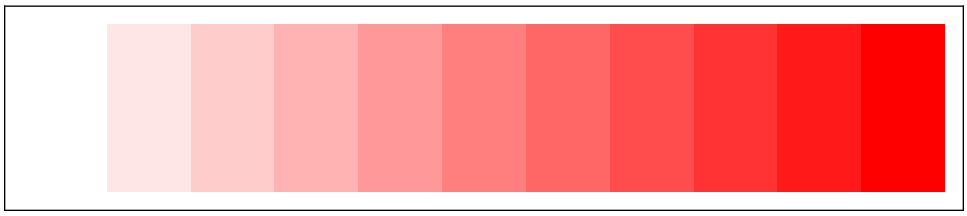
Der Zusammenhang zwischen **Hue** und RGB:

```
ArrayPlot[{Table[Hue[i], {i, 0, 1, .1}], Table[
   Blend[{Red, Yellow, Green, Cyan, Blue, Magenta, Red}, i], {i, 0, 1, .1}]}, Mesh → True]
```



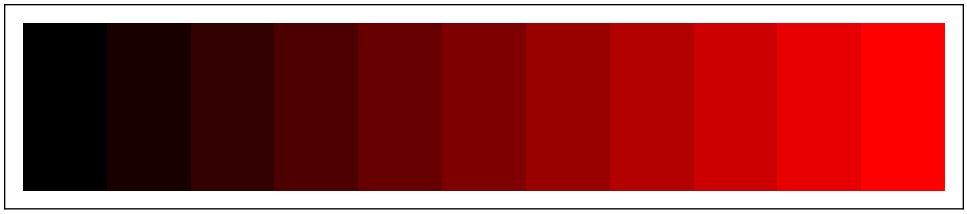
Hue mit dem Sättigungsparameter

```
ArrayPlot[{Table[Hue[1, i], {i, 0, 1, .1}],
  Table[Blend[{Red, White}, 1 - i], {i, 0, 1, .1}]}]
```



Hue mit dem Helligkeitsparameter.

```
ArrayPlot[{Table[Hue[1, 1, i], {i, 0, 1, .1}],
  Table[Blend[{Red, Black}, 1 - i], {i, 0, 1, .1}]}]
```



Mathematica stellt eine große Menge von Farbpaletten aus den verschiedensten Anwendungsgebieten bereit. Diese können aus der mitgelieferten **ColorData**−Sammlung über ein Schlüsselwort als **ColorDataFunction**−Objekt extrahiert werden. Eine solche **ColorDataFunction** ist eine Funktion aus einem meist als reelles Intervall gegebenen Argumentbereich in ein Farbspektrum spezieller, genauestens ausgewählter ("carefully chosen") RGB−Werte.

```
c = ColorData["Rainbow"]
ArrayPlot[{Table[c[i], {i, 0, 1, .01}]}, AspectRatio -> .1]
```

```
ColorDataFunction[{0, 1}, ]
```

```
ColorData[]
```

```
{Gradients, Indexed, Named, Physical}
```

```
ColorData["Physical"]
```

```
{BlackBodySpectrum, HypsometricTints, VisibleSpectrum}
```

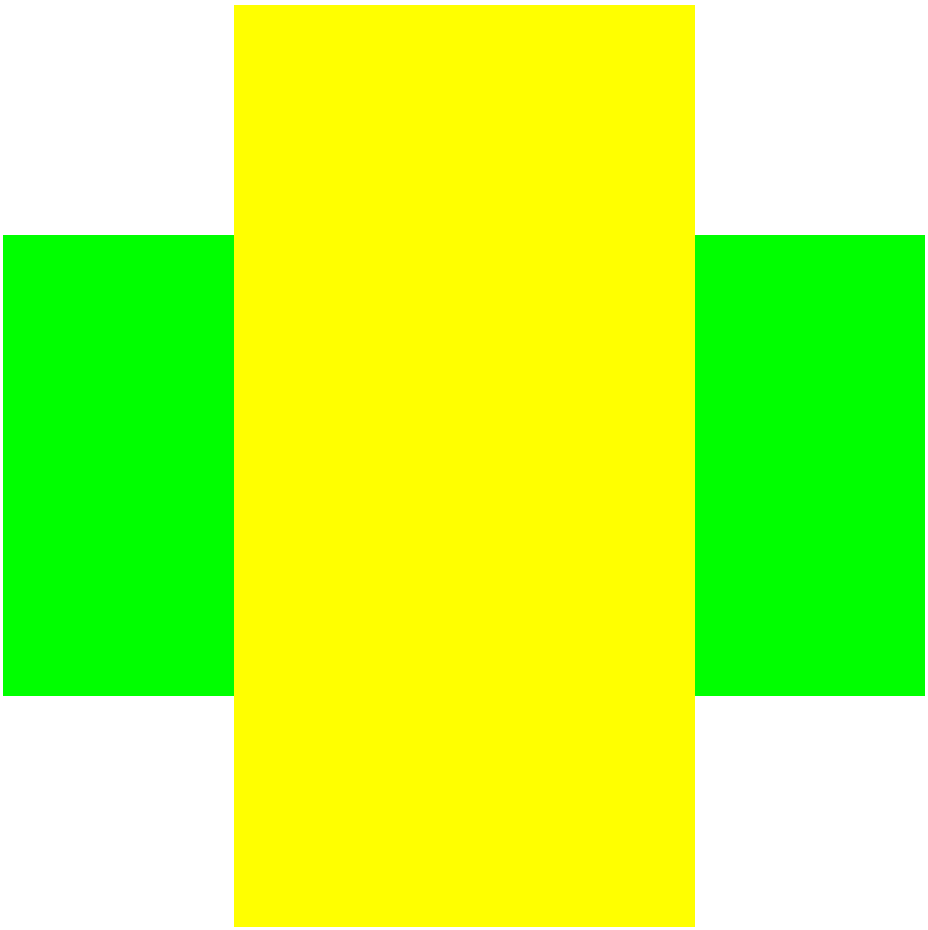
```
ColorData["VisibleSpectrum"]
```

```
ColorDataFunction[{380, 750}, ]
```

Uberdeckungen und Transparenz in 2D−Grafiken

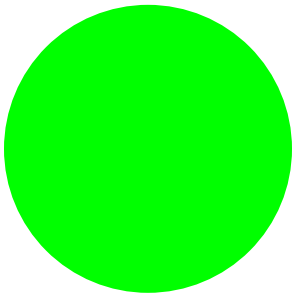
Zweidimensionale Flächen überdecken sich, was passiert? Hier ein Beispiel mit teilweiser Durchsichtigkeit.

```
r1 = Rectangle[{-2, -1}, {2, 1}];
r2 = Rectangle[{-1, -2}, {1, 2}];
g = Graphics[{Green, r1, Opacity[.7], Yellow, r2}]
```

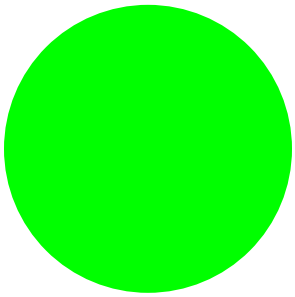


ColorOutput funktioniert nicht und ist als obsolet markiert, aber der Ersatz **ColorFunction** ist noch nicht einsatzbereit.
Der Punkt muss grau sein und nicht grün (und ist es auch in Version 5.1).

```
g = Graphics[{PointSize[.3], Green, Point[{0, 0}]}]
```

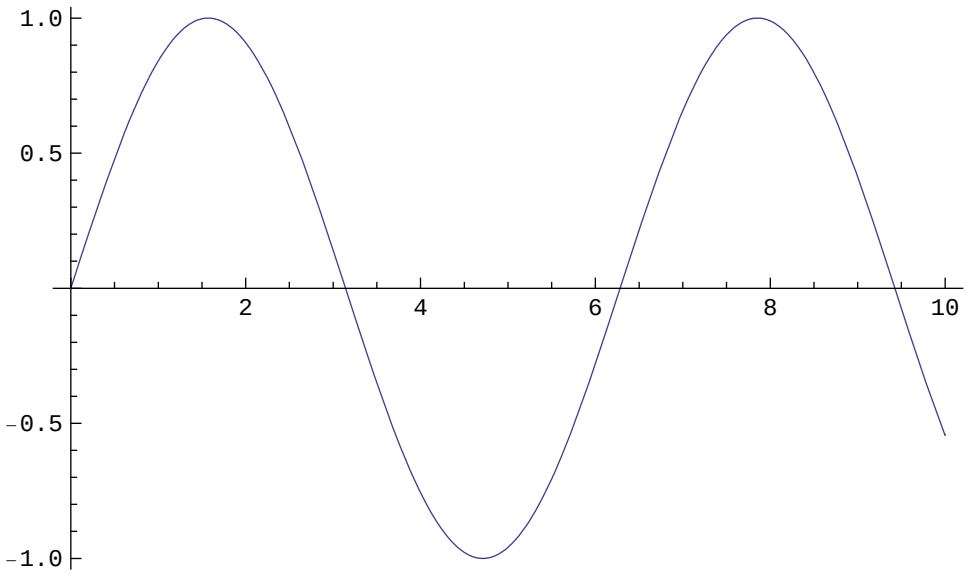


```
Show[g, ColorOutput → GrayLevel]
```

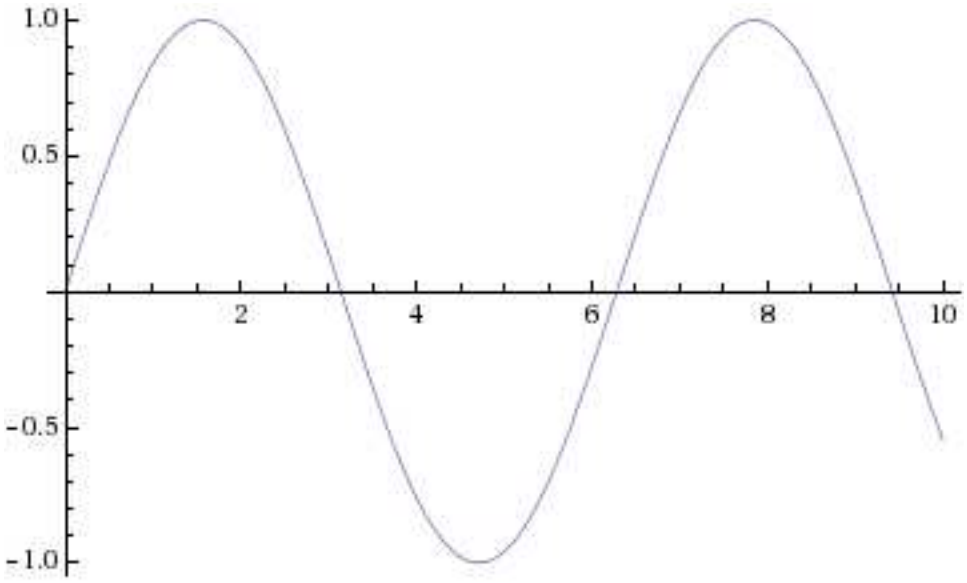


▪ Farbgebung und Rastergrafiken

```
gOut = Plot[Sin[x], {x, 0, 10}]
```




```
Export["test.gif", gOut];
gIn = Import["test.gif"]
```




```
ByteCount /@ {gOut, gIn}
```

```
{10 032, 321 584}
```

```
gIn // Head
```

```
Graphics
```

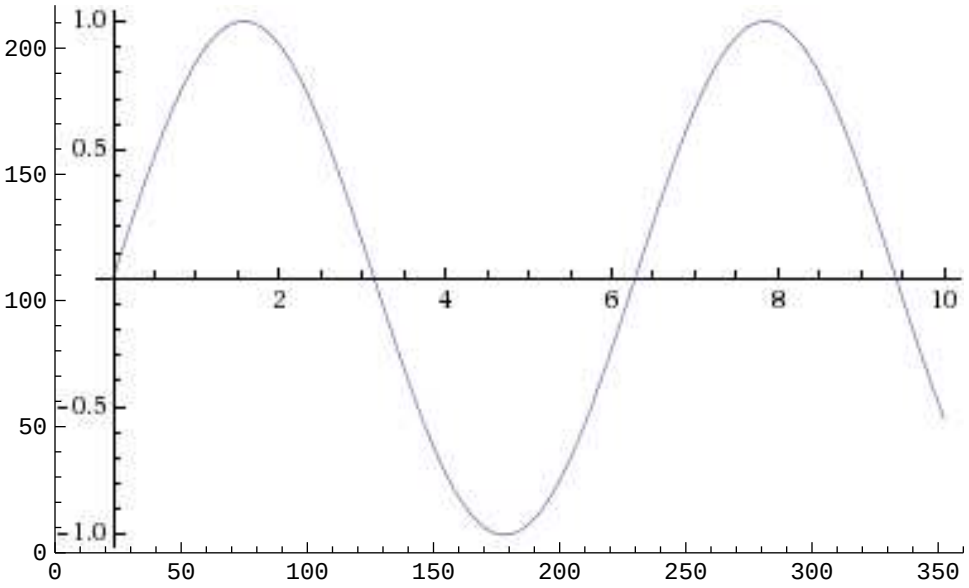
```
List @@ Head /@ gIn
```

```
{Raster, Rule, Rule, Rule}
```

```
gIn[[#]] & /@ {2, 3, 4}
```

```
{ImageSize → {360, 217}, PlotRange → {{0, 360}, {0, 217}}, Background → None}
```

```
Show[gIn, Axes -> True]
```




```
r = gIn[[1]];
List @@ Head /@ r
```

```
{List, List, Rule, Rule}
```

```
r[[1]] // Dimensions
```

```
{217, 360}
```

```
r[[2]]
```

```
{{0, 0}, {360, 217}}
```

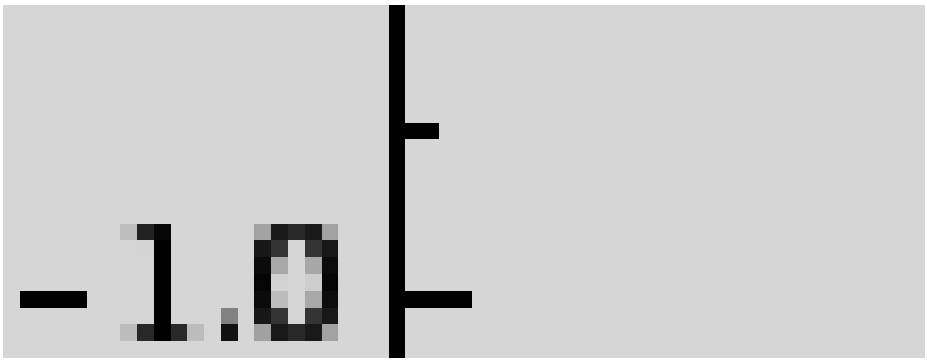
```
c = r[[4, 2]];
c // Short
```

```
RGBColor @@ {{0., 0., 0., 1.}, <<254>>, {0., 0., 0., 1.}}[[#1]] &
```

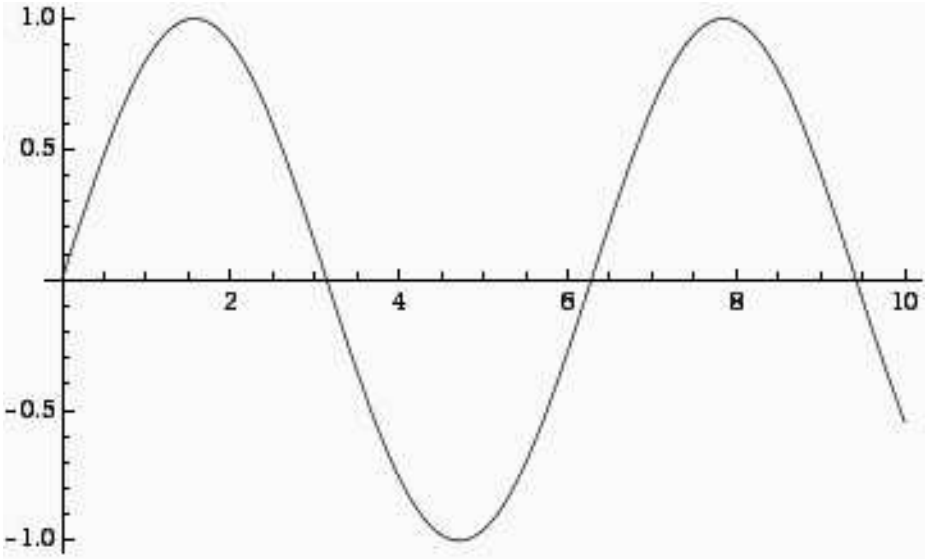
```
r[[1]][[Range[5, 9], Range[7, 12]]] // MatrixForm
```

$$\begin{pmatrix} 251 & 251 & 251 & 251 & 251 & 251 \\ 251 & 221 & 48 & 1 & 49 & 221 \\ 251 & 251 & 251 & 1 & 251 & 251 \\ 251 & 251 & 251 & 1 & 251 & 251 \\ 251 & 251 & 251 & 1 & 251 & 251 \end{pmatrix}$$

```
r1 = r[[1]][[Range[5, 25], Range[1, 55]]];
Graphics[Raster[r1 / 300]]
```

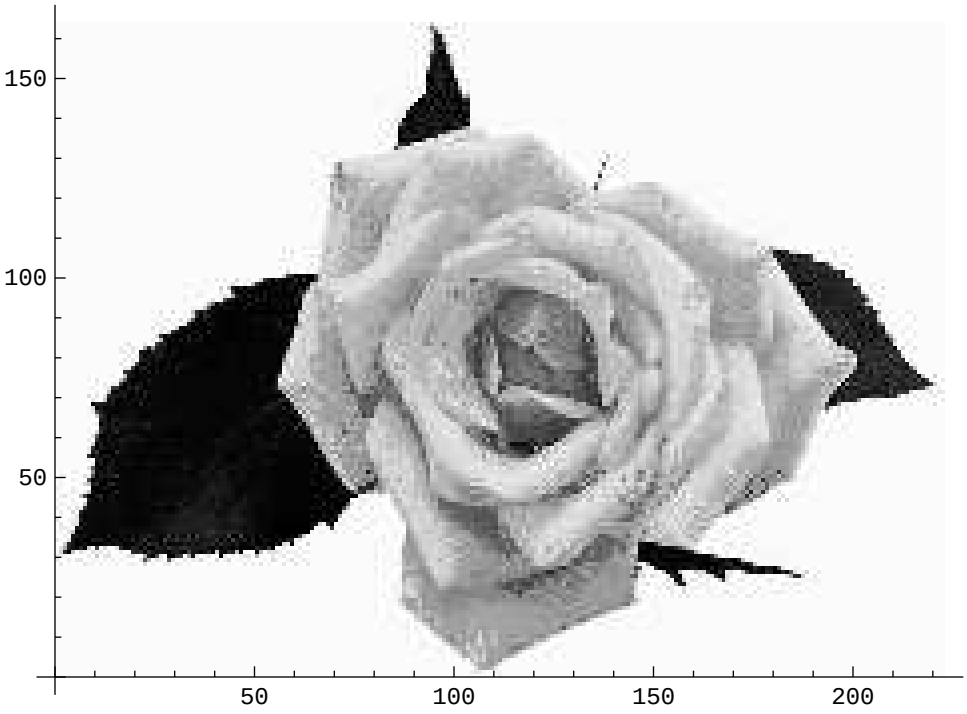


```
Graphics[Raster[r[[1]], ColorFunction -> (GrayLevel[# / 256] &)]]
```




```
gIn = Import["ExampleData/rose.gif"]
```

```
r = gIn[[1]];
Graphics[Raster[r[[1]], ColorFunction -> (GrayLevel[# / 256] &)], Axes -> True]
```

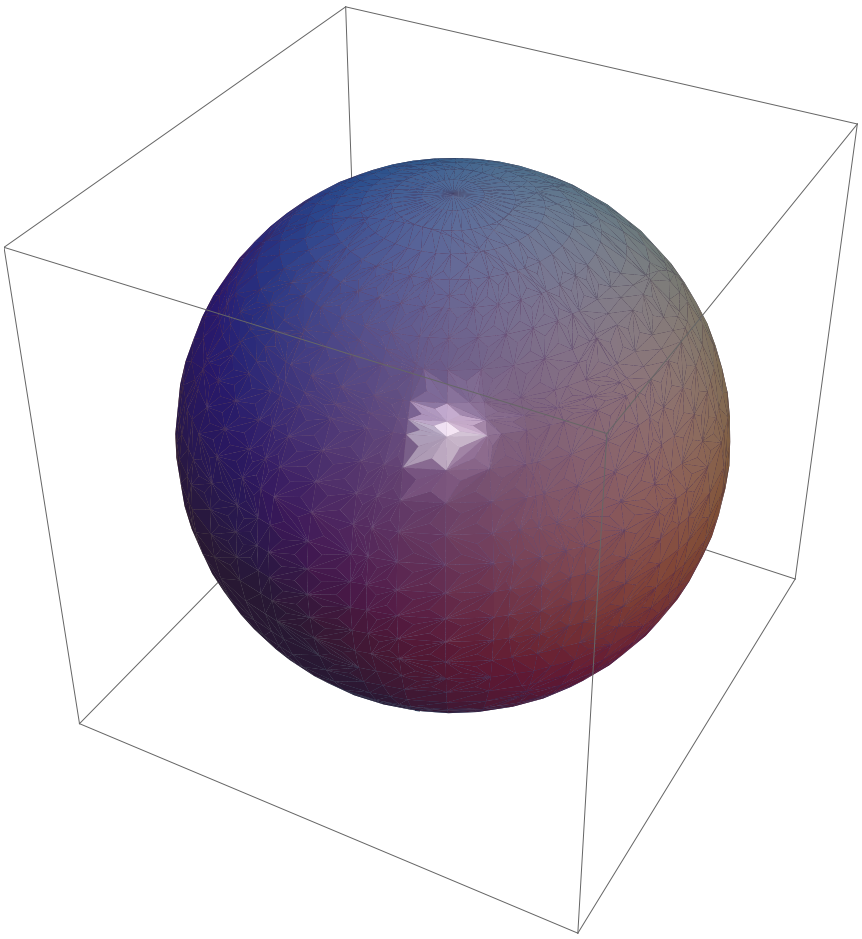


Das Beleuchtungsmodell

Visualisierung der Standardbeleuchtung

Standard−Lichtquellen auf einer stark spiegelnden Kugel: Ambientes Licht und drei (die Dokumentation behauptet vier) Lichtquellen in den Farben Rot, Grün, Blau, die mit der Kamera starr verbunden sind.

```
g = Graphics3D[{GrayLevel[.5], Specularity[1, 300], Sphere[]}]
```

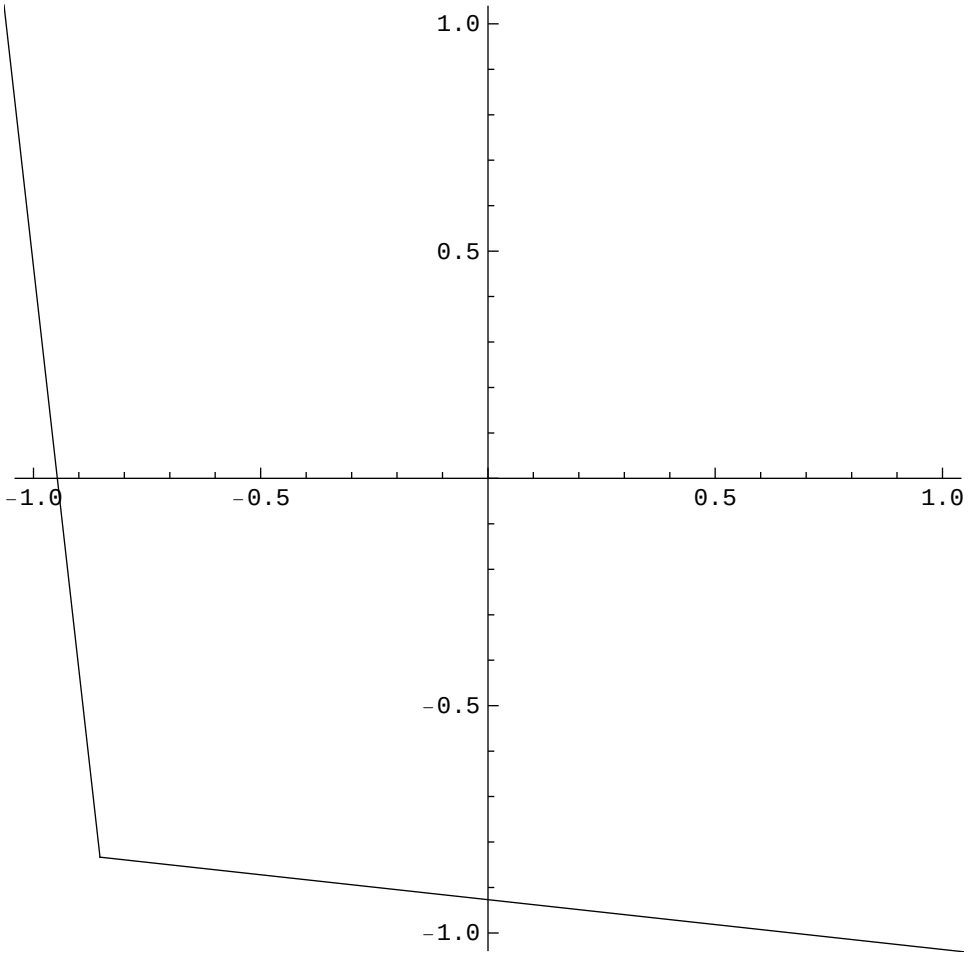


```
AbsoluteOptions[g, viewOptions = {ViewPoint, ViewCenter, ViewVertical}]
```

```
{ViewPoint → {1.3, -2.4, 2.}, ViewCenter → {0.5, 0.5, 0.5}, ViewVertical → {0., 0., 1.}}
```

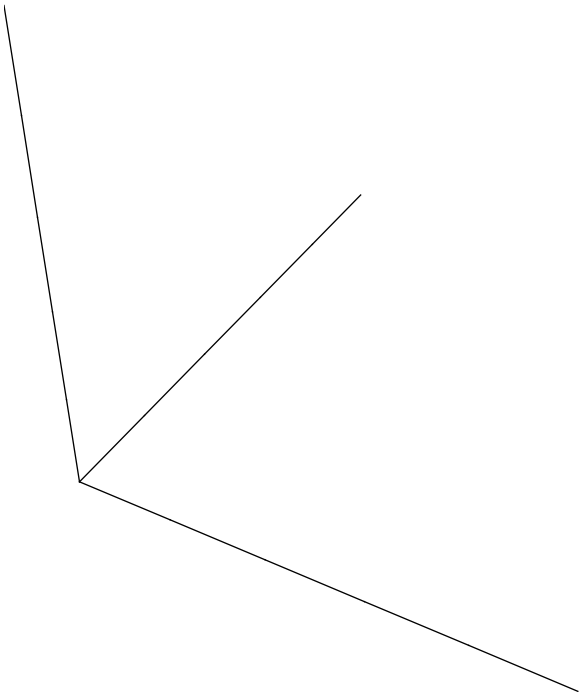
ImageScaled ist angeblich relativ zum **ImageSize**. Im 2D−Fall ist plausibel, was darunter zu verstehen ist, nicht aber im 3D−Fall.

```
v = {{1, 0}, {0, 1}};
Graphics[Line[ImageScaled /@ {#, {0.1, 0.1}}] & /@ v, Axes -> True]
```



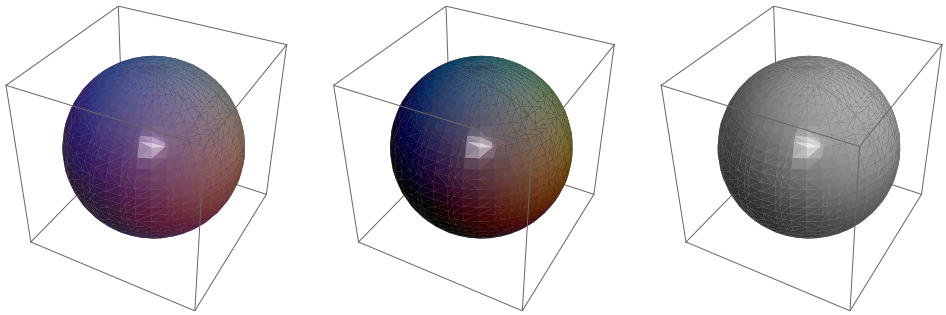
Obwohl **ImageScaled** ... und sie bewegt sich doch.

```
v = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
Graphics3D[Line[ImageScaled /@ {#, {0, 0, 0}}] & /@ v, Boxed -> False]
```



Früher gab es drei Lichtquellen. Ist auch jetzt noch so im Gegensatz zur Beschreibung im Hilfesystem. Allein ambientes Licht scheint noch eine weitere Portion dabei zu sein.
Der eps−Export funktioniert hier nicht richtig − die Spielegeffekte werden falsch dargestellt.

```
v = {{1, 0, 1}, {1, 1, 1}, {0, 1, 1}};
c = {Red, Green, Blue};
l0 = {"Ambient", GrayLevel[0.1]};
l = Table[{"Directional", c[[i]], ImageScaled /@ {v[[i]], {.5, .5, .5}}}, {i, 1, 3}];
u = GraphicsGrid[{{g, Show[g, Lighting → l], Show[g, Lighting → "Neutral"]}}]
```



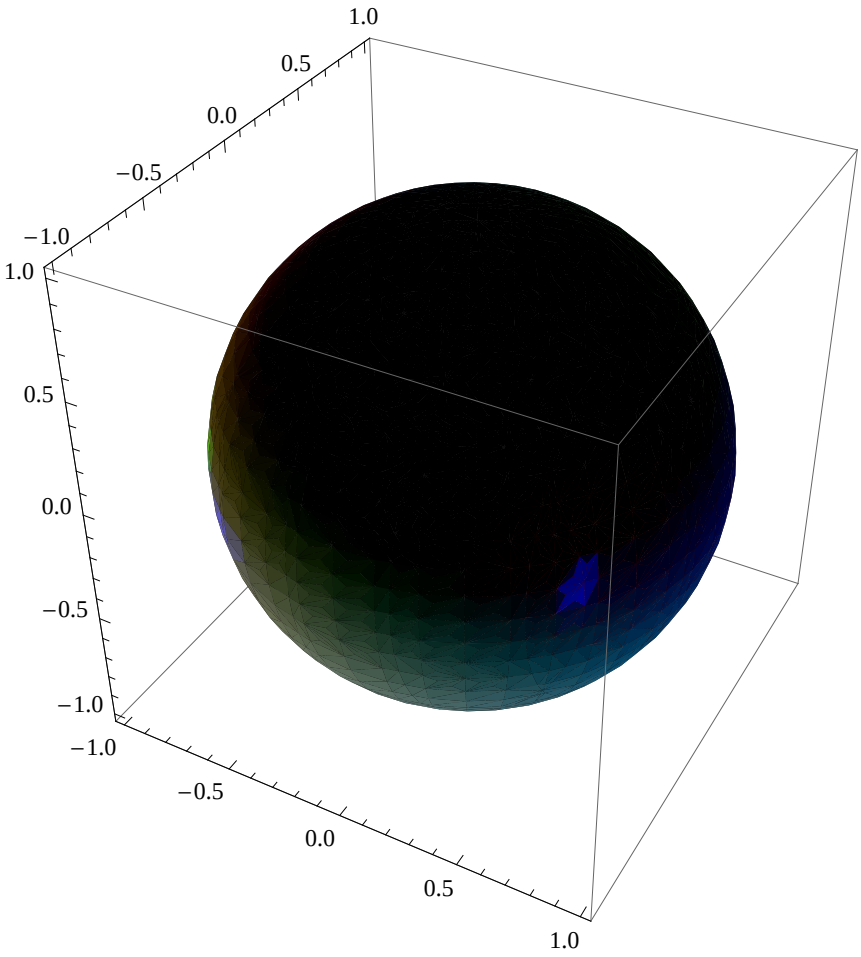
Leider funktioniert der eps−Export dieser Grafik nicht. Deshalb der Umweg über eine jpg−Grafik mit leidlicher Auflösung. Bei größeren Auflösungen hat auch der jpg−Export so seine Probleme.

```
Export["test.jpg", u, ImageSize -> 30 {60, 25}]
```

Probleme mit der Beleuchtung (1)

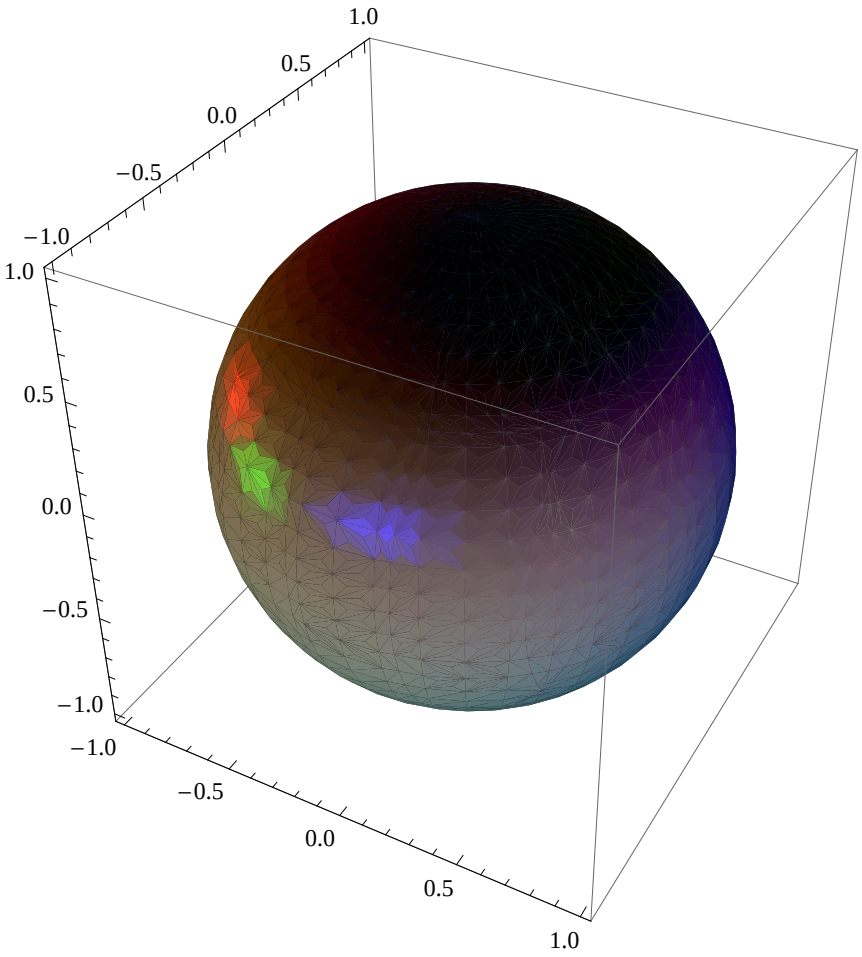
Obwohl **ImageScaled** bewegen sich die Lichtreflexe mit der Grafik und sind außerdem an der falschen Stelle.

```
l = Table[{"Point", c[[i]], ImageScaled[v[[i]]]}, {i, 1, 3}];
Show[g, Lighting → l, Axes -> True]
```



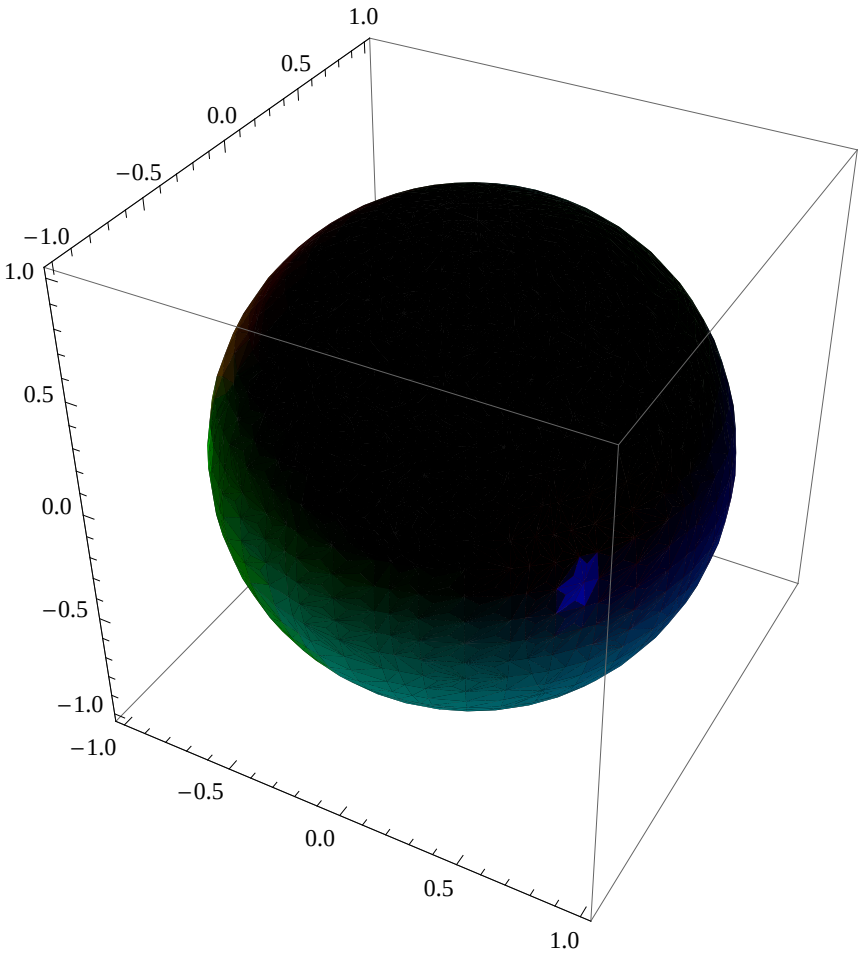
Hier werden absolute Koordinaten für die Lichtquellenpunkte verwendet und es sieht gut aus.

```
l = Table[{"Point", c[[i]], v[[i]]}, {i, 1, 3}];
Show[g, Lighting → l, Axes -> True]
```

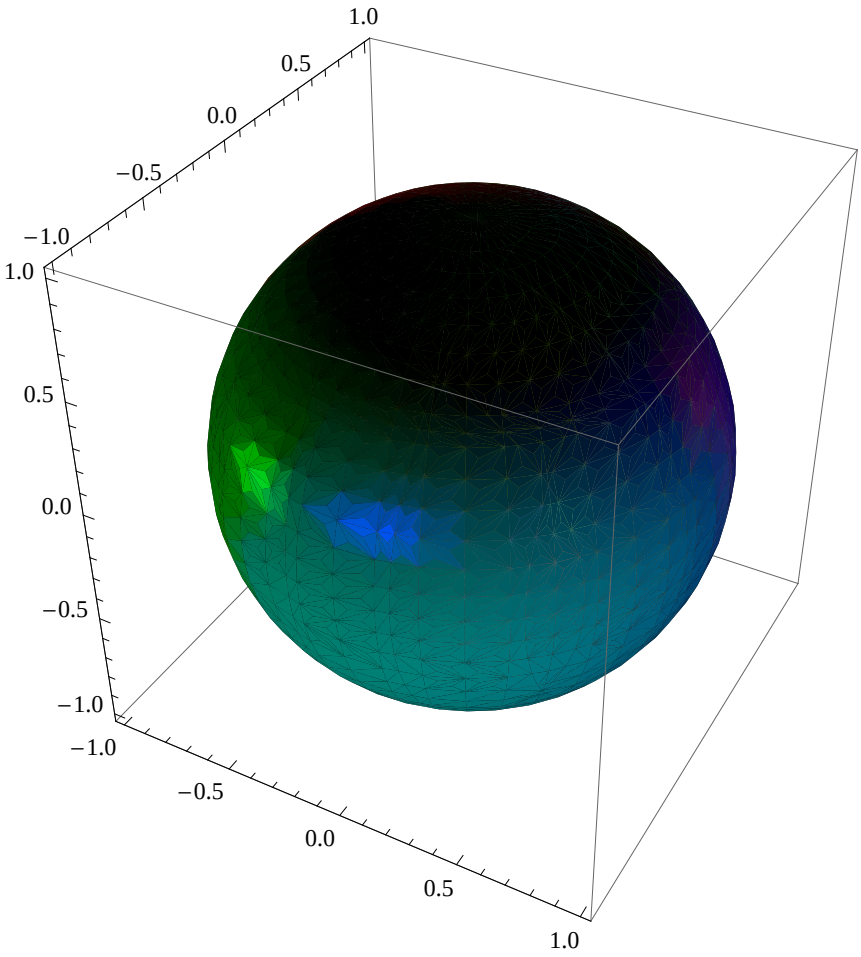


Mit Spot−Licht ist es nicht viel anders.

```
l = Table[{"Spot", c[[i]], ImageScaled /@ {v[[i]], {.5, .5, .5}}, 50 °}, {i, 1, 3}];
Show[g, Lighting → l, Axes -> True]
```



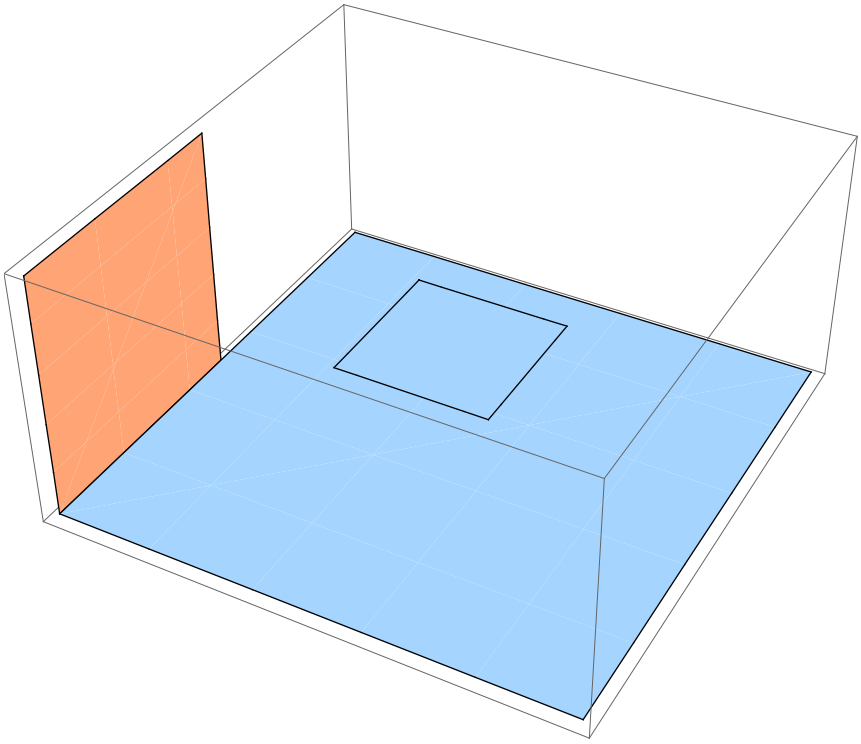
```
l = Table[{"Spot", c[[i]], {v[[i]], {.5, .5, .5}}, 50°}, {i, 1, 3}];
Show[g, Lighting → l, Axes -> True]
```



■ Probleme mit der Beleuchtung (2)

Kurz − mit der Beleuchtung scheint noch vieles im Argen zu liegen. Hier ein kleines Testbeispiel, an welchem die Beleuchtungverhältnisse überschaubar sind: zwei übereinander liegende Quadrate und eine einzige Lichtquelle. Wengistens den Schatten sollte man ja dann sehen.

```
v = {{0, 0, 0}, {0, 1, 0}, {1, 1, 0}, {1, 0, 0}};
g = Graphics3D[
  {Polygon[.5 RotateRight /@ v], Polygon[v], Scale[Polygon[v], .3, {.5, .5, .3}]}]
```



```
p1 = {"Point", White, {1, 1, 1}};
p2 = {"Point", White, Scaled[{1, 1, 1}]};
p3 = {"Point", White, ImageScaled[{1, 1, 1}]};
d1 = {"Directional", White, {{1, 1, 1}, {0, 0, 0}}};
d2 = {"Directional", White, Scaled /@ {{1, 1, 1}, {0, 0, 0}}};
d3 = {"Directional", White, ImageScaled /@ {{1, 1, 1}, {0, 0, 0}}};
s1 = {"Spot", White, {{1, 1, 1}, {0, 0, 0}}, 40 °};
s2 = {"Spot", White, Scaled /@ {{1, 1, 1}, {0, 0, 0}}, 40 °};
s3 = {"Spot", White, ImageScaled /@ {{1, 1, 1}, {0, 0, 0}}, 40 °};

Show[g, Lighting -> {p1}]
```

And this are the reults with the version of April 20:
p1 − no shadow, location plausible, moving with the picture
p2 − no shadow, location plausible, moving with the picture
p3 − no shadow, location wrong, moving with the picture
d1 − no shadow, location plausible, moving with the picture
d2 − no shadow, location plausible, moving with the picture
d3 − no shadow, location wrong, moving with the picture
s1 − small shadow, location plausible, moving with the picture
s2 − no shadow, location not plausible, moving with the picture
s3 − no shadow, location wrong, moving with the picture

CutOut (Durchsichtige 3D−Objekte)

Eine erste Lösung

Wir beginnen mit Grafikobjekten, die aus einfachen Primitiven zusammengesetzt sind.
Als Startpunkt verwenden wir unseren weiter oben definierten Torus.

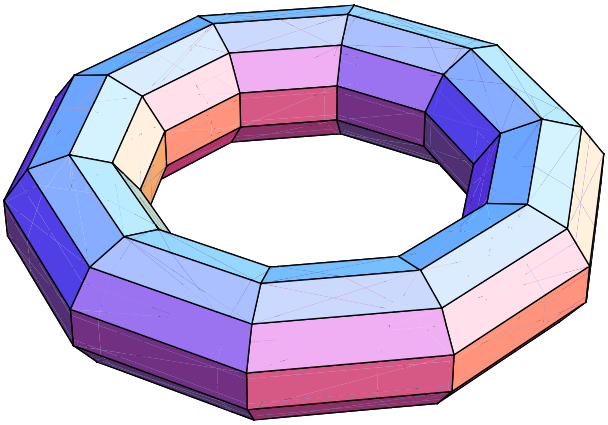
```
GitterComplex[P0_, dx_, dy_, nx_Integer, ny_Integer] :=
 GraphicsComplex[Join @@ Table[P0 + {i * dx, j * dy}, {i, 0, nx}, {j, 0, ny}] // Chop,
  Polygon[Join @@ Table[{(ny + 1) i + j + 1, (ny + 1) i + j + 2,
      (ny + 1) (i + 1) + j + 2, (ny + 1) (i + 1) + j + 1}, {i, 0, nx - 1}, {j, 0, ny - 1}]]]

GitterComplexPlot[f_, P0_, dx_, dy_, nx_Integer, ny_Integer] :=
 With[{gc = GitterComplex[P0, dx, dy, nx, ny]}, GraphicsComplex[f /@ gc[[1]], gc[[2]]]]

torusFunction[{t_, u_}, s_, d_] := Module[{r = {Cos[t], Sin[t], 0}, b = {0, 0, 1}},
  r + s r * Sin[u] + s b * Cos[u] // N]

Torus[s_, d_] := GitterComplexPlot[torusFunction[#, s, d] &, {0, 0}, 2 Pi / d, 2 Pi / d, d, d]
```

Ein solches **GraphicsComplex**−Objekt kann mit **Normal** in eine Liste einfacher Grafikprimitive transformiert werden. Dabei werden alls Abkürzungen aufgelöst, auch die, dass ein **Polygon**−Objekt mit einer Liste von Listen von Punkten als Liste von **Polygon**−Objekten interpretiert wird.

```
Short[Torus[.3, 10], 3]
Short[t = Torus[.3, 10] // Normal, 7]
```

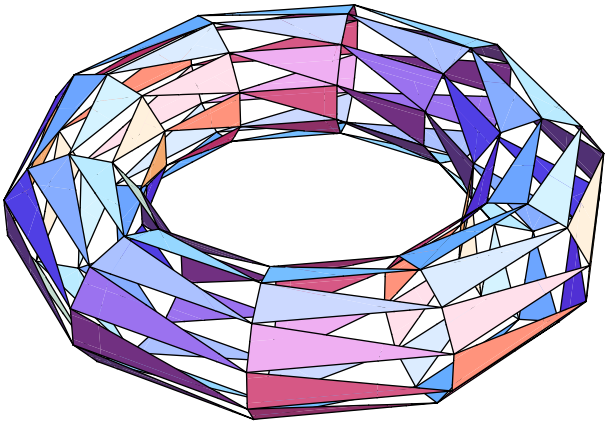
```
GraphicsComplex[{{1., 0., 0.3}, {1.17634, 0., 0.242705}, <<117>>,
  {0.823664, 0., 0.242705}, {1., 0., 0.3}}, Polygon[{{1, 2, 13, 12}, <<99>>}]]

{{Polygon[{{1., 0., 0.3}, {1.17634, 0., 0.242705},
    {0.951675, 0.691433, 0.242705}, {0.809017, 0.587785, 0.3}}], <<98>>,
  Polygon[{{0.666359, -0.484138, 0.242705}, {0.809017, -0.587785, 0.3},
    {1., 0., 0.3}, {0.823664, 0., 0.242705}}]}}
```

```
torus = Graphics3D[t, Boxed → False]
```



Ein so normalisierter Torus besteht aus Polygonen mit vier Eckpunkten, woraus das folgende Kommando Polygone mit drei Eckpunkten macht. Damit wird der Torus "durchsichtig".

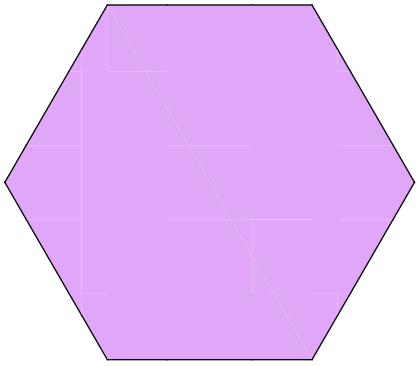
```
Clear[p1, p2, p3, p4];
torushohl = torus /.
  Polygon[{p1_, p2_, p3_, p4_}] → Polygon[{p1, p2, p3}]
```



CutOut − Löcher in Polygone schneiden

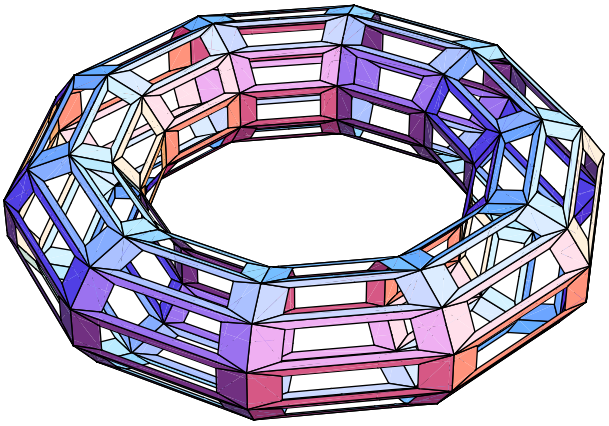
Reguläres Polygon über Punkte in der Gaußschen Zahlenebene als Grafikprimitiv konstruieren.

```
regularPolygon[n_Integer] /; n > 0 := Polygon[{Re[#], Im[#], 0} & /@
        Table[(-1)^(2 i / n), {i, n}] // N];

regularPolygon[6];
p = Graphics3D[%, Boxed → False, ViewPoint -> {0, 0, 1}]
```



```
cutout[p_List, α_] :=
     Module[{l, q},
      q = Map[((1 - α) Mean[p] + α #) &, p];
      l = Thread[{p, RotateLeft[p], RotateLeft[q], q}];
      Polygon /@ l
     ]
```

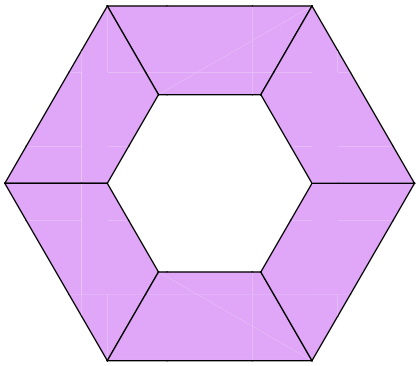
```
torus /. Polygon[x_] → cutout[x, .66]
```



```
CutOut[g_Graphics3D, α_?(0 < # < 1 &)] := g /. Polygon[x_] → cutout[x, α]

liste = {{1, 2, 3}, {4, 5, 6}, {3, 2, 1}};

cutout[liste, 0.5]
```

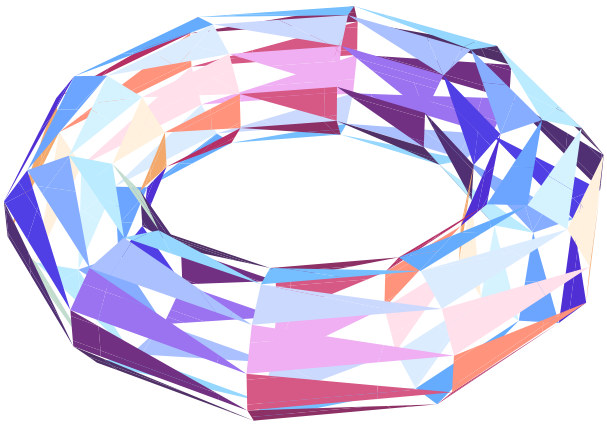
```
{Polygon[{{1, 2, 3}, {4, 5, 6}, {3.33333, 4., 4.66667}, {1.83333, 2.5, 3.16667}}],
 Polygon[{{4, 5, 6}, {3, 2, 1}, {2.83333, 2.5, 2.16667}, {3.33333, 4., 4.66667}}],
 Polygon[{{3, 2, 1}, {1, 2, 3}, {1.83333, 2.5, 3.16667}, {2.83333, 2.5, 2.16667}}]}
```

```
q = CutOut[p, 0.5]
```

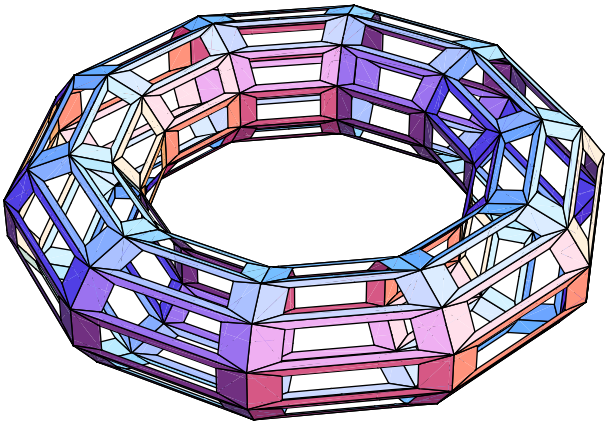


Das ist eine sehr simple Version zum Entfernen der Bildkanten, die nur mit einfachen **Graphics3D**−Objekten funktioniert. Hierzu wird in die Liste *a* der Grafikprimitive, aus denen das **Graphics3D**−Objekt aufgebaut ist, an erster Stelle die Grafikdirektive **EdgeForm[]** eingefügt. Diese bewirkt, dass alle folgenden Grafikprimitive ohne Kanten dargestellt werden.

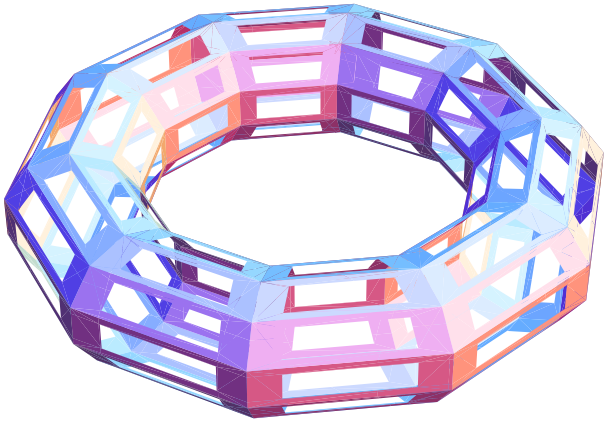
```
NoEdges[g_Graphics3D] := g /. Graphics3D[{a__}, b___] -> Graphics3D[{EdgeForm[], a}, b]

torushohl // NoEdges
```



```
CutOut[torus, .66]
```



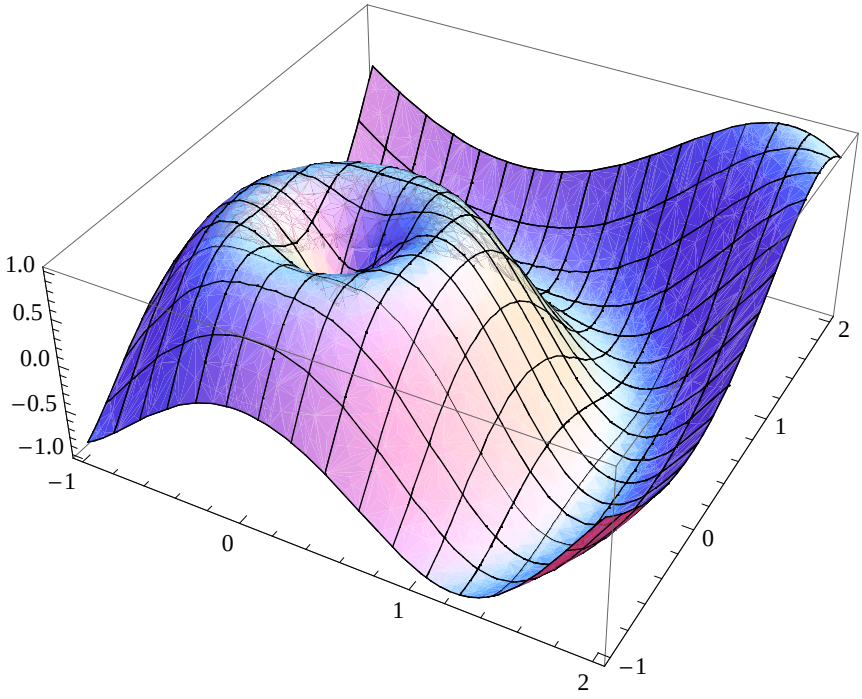
```
CutOut[torus, .66] // NoEdges
```



■ CutOut für komplexe Graphics3D−Objekte

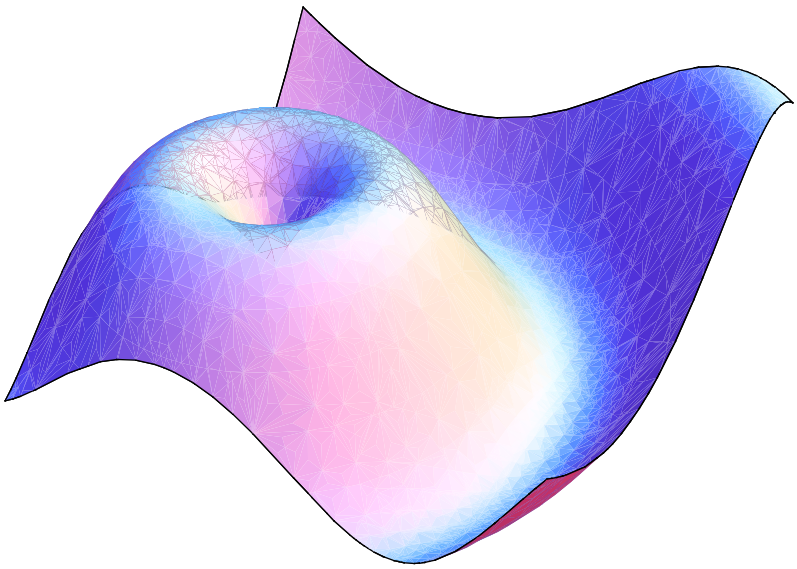
In der neuen Grafik von *Mathematica* haben die Gitterlinien eines 3D−Objekts nichts mehr mit dessen innerer Struktur zu tun, sondern sind erst in einem zweiten Arbeitsschritt als eindimensionale Linienzüge aufgebracht. Demzufolge muss mehr Aufwand getrieben werden, um auch komplexe Grafikobjekte zu "durchlöchern".

```
Plot3D[Sin[3 Sqrt[x^2 + y^2]], {x, -1, 2}, {y, -1, 2}]
```



Als erstes entfernen wir alle Linien, denn sie stören nur.

```
p =
 Plot3D[Sin[3 Sqrt[x^2 + y^2]], {x, -1, 2}, {y, -1, 2}, Mesh → None, Boxed → False, Axes → False]
```

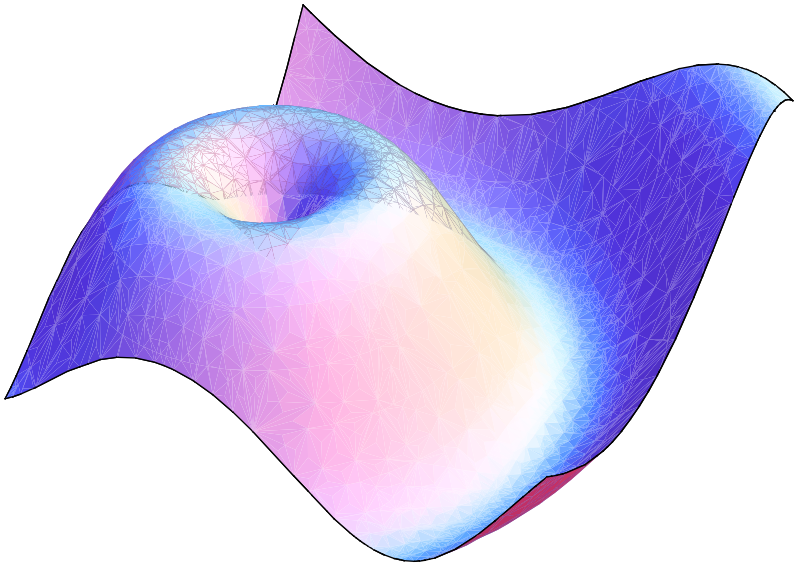


```
Short[p // InputForm, 5]
```

```
Graphics3D[GraphicsComplex[{{-0.9999997857142857, -0.9999997857142857,
  -0.8916818429467935}, {-0.7857141020408163, -0.9999997857142857,
  -0.6238468378008891}, {-0.5714284183673468, <<2>>}, {<<3>>}, <<1284>>, {<<3>>},
  {-0.6249998392857141, 1.8392855229591842, -0.4398768503698975}}, <<2>>], {<<5>>}]
```

p ist ein **Graphics3D**−Objekt. Da **CutOut** neue Stützpunkte generiert, muss diese kompakte Darstellung durch das Kommando **Normal** erst aufgelöst werden. Die Grafik wird natürlich genauso dargestellt wie die vorherige in ihrer kompakten Darstellung.

```
q = p // Normal
```



```
Depth /@ {p, q}
```

```
{11, 12}
```

```
ByteCount /@ {p, q}
```

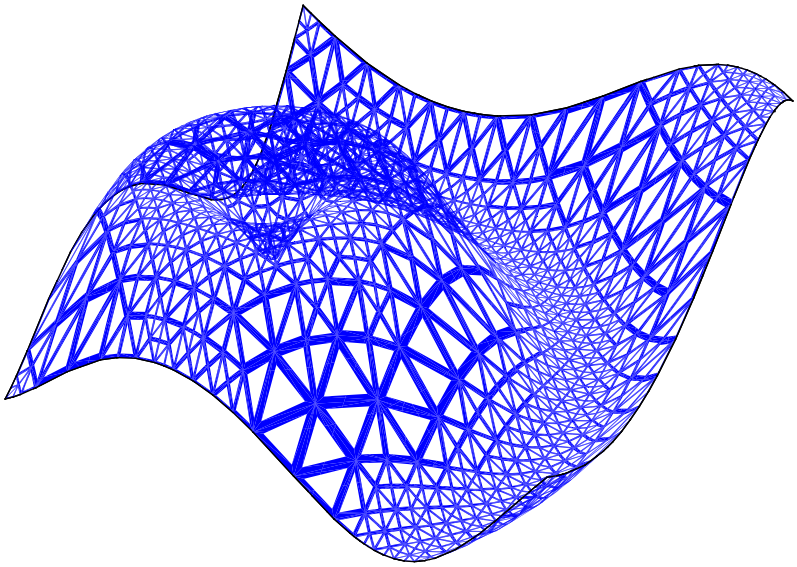
```
{93 480, 920 944}
```

Außerdem können **Polygon**−Objekte nun eine Liste von Optionen haben, was wir beim Anschreiben von **cutout** und **CutOut** berücksichtigen müssen. Die modifizierten Versionen haben folgendes Aussehen:

```
cutout[p_List, α_, opts___?OptionQ] :=
     Module[{l, q},
      q = Map[((1 - α) Mean[p] + α #) &, p];
      l = Thread[{p, RotateLeft[p], RotateLeft[q], q}];
      Polygon /@ l
     ]
CutOut[g_Graphics3D, α_?(0 < # < 1 &)] :=
 Normal[g] /. Polygon[x_, opts___?OptionQ] → cutout[x, α, opts]
```

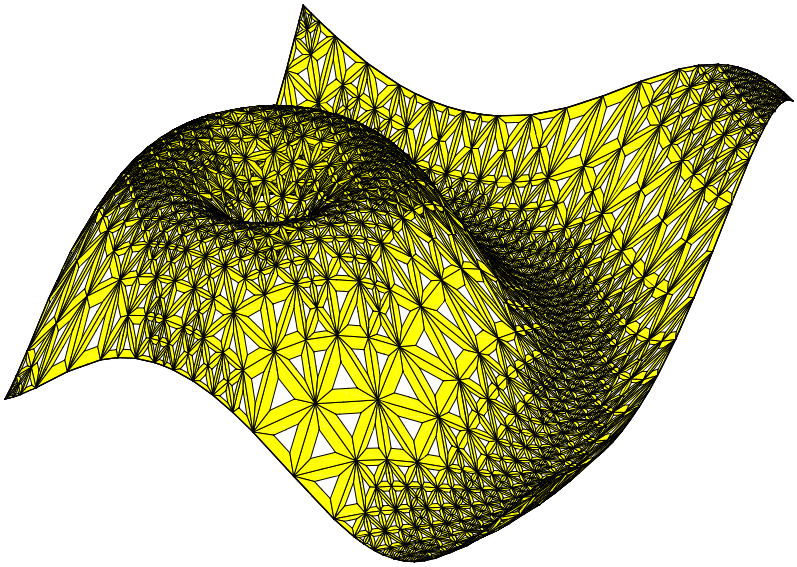
Nun funktioniert **CutOut** auch für **Plot3D**−Ausgaben.

```
CutOut[q, 0.7];
Show[%, Lighting -> {Blue}]
```



Den **Polygon**−Objekten ist die Grafikdirektive **EdgeForm[]** vorangestellt, welche das Zeichnen der Kanten der Polygone vermeidet. Dies kann mit dem folgenden Kommando verändert werden. Auf Grund der adaptiven Größe der Triangulierung ist das Bild aber nicht so eindrucksvoll wie bei einer Triangulierung mit konstanter Netzgröße.

```
CutOut[q, 0.4] /. EdgeForm[] → EdgeForm[{Thickness[0.0001], Black}];
Show[%, Lighting -> {Yellow}]
```



■ TubePlot3D − Röhrenkurven programmieren

Einige allgemein nützliche Funktionen.

```
ClearAll["Global‘*"]
TPlen[x_] := √(x[[1]]^2 + x[[2]]^2 + x[[3]]^2) ;
TPnorm[x_] := x / TPlen[x] // Together;
```

■ Erste Lösung: Die "mathematische", d.h. analytische über das begleitende Dreibein und ParametricPlot

```
Clear[TPDreibein];
TPDreibein[f_?VectorQ, t_Symbol] :=
 Module[{h, n, b, u},
  h = D[f, t] // TPnorm; (* Haupttangentenvektor *)
  u = D[f, t, t];
  n = If[u === {0, 0, 0},
    If[h[[1]] === 0, {0, h[[3]], -h[[2]]}, {h[[2]], -h[[1]], 0}], (u - (h . u) h)];
  n = n // TPnorm; (* Normalenvektor *)
  b = h × n // Together; (* Binormalenvektor *)
  {h, n, b}]
```

Message gibt Zwischenergebnisse aus. Das ist zum Debuggen der Funktion ebenso sinnvoll wie zur Demonstration, dass die analytische Lösung hier ungeeignet ist. Solche Zwischenausgaben können wie Systeminformationen mit **Off** abgeschaltet werden.

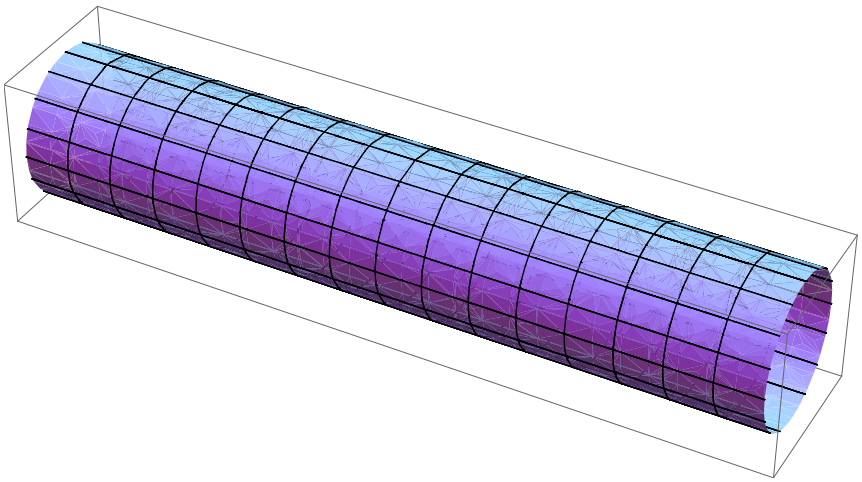
```
Clear[TubePlot3D];
TP::msg1 = "f ist ‘1‘";
TP::msg2 = "Begleitendes Dreibein ist ‘1‘";
TubePlot3D[f_?VectorQ, r_: 0.1, {t_Symbol, start_, end_}, opts___?OptionQ] :=
 Module[{h, n, b, u},
  Message[TP::msg1, f];
  {h, n, b} = TPDreibein[f, t];
  Message[TP::msg2, {h, n, b}];
  ParametricPlot3D[
   Evaluate[f + (Sin[u] n + Cos[u] b) r],
   {t, start, end}, {u, 0, 2 π},
   opts, Boxed → False, Axes → False]
 ]
```

Einige Beispiele.
Der Aufruf mit *u* statt *t* klappt auch, dank der lokal deklarierten Variablen in der Funktionsdefinition von **TubePlot3D**.

```
TubePlot3D[{u, 0, 0}, {u, 0, 1}, Boxed -> True]
```

```
TP::msg1 : f ist {u, 0, 0}
```

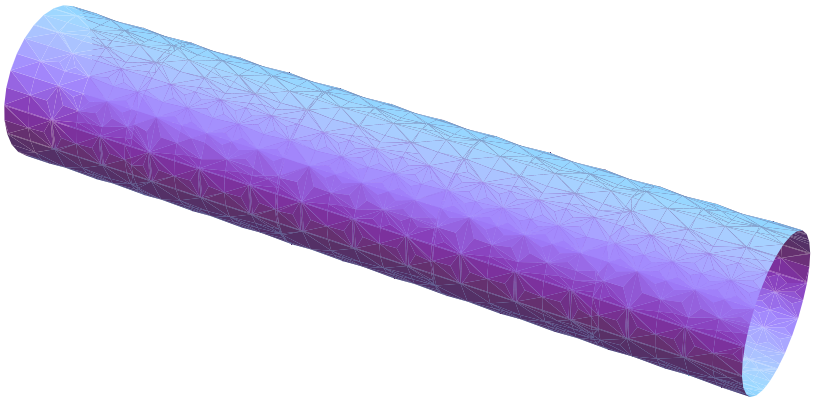
```
TP::msg2 : Begleitendes Dreibein ist {{1, 0, 0}, {0, -1, 0}, {0, 0, -1}}
```



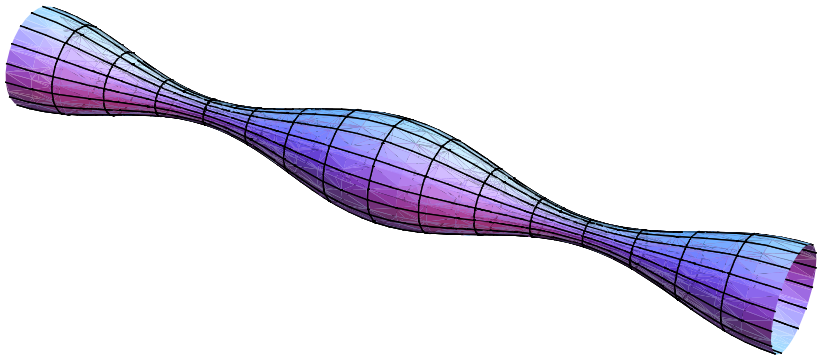
```
TubePlot3D[{u, 0, 0}, {u, 0, 1}, Mesh → None]
```

```
TP::msg1 : f ist {u, 0, 0}
```

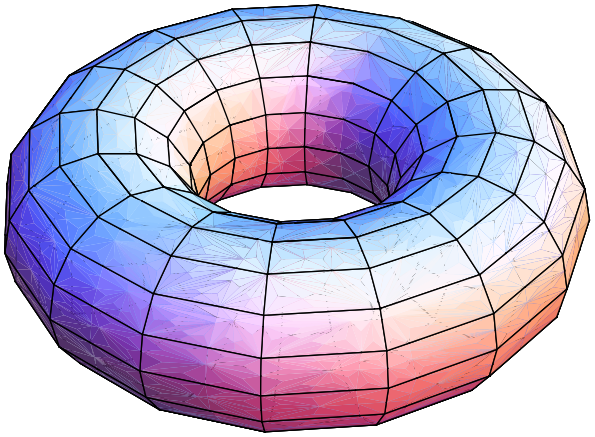
```
TP::msg2 : Begleitendes Dreibein ist {{1, 0, 0}, {0, -1, 0}, {0, 0, -1}}
```



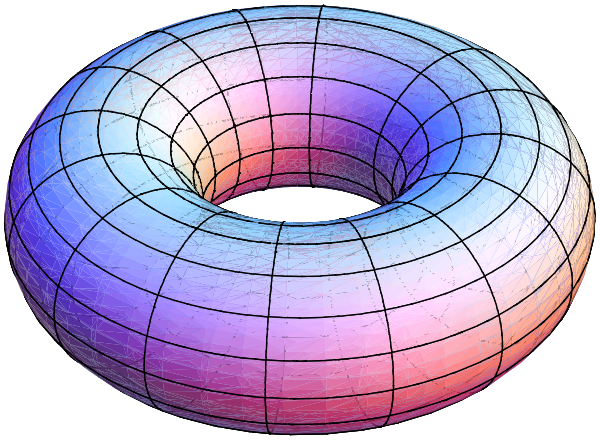
```
Off[TP::msg1]; Off[TP::msg2];
TubePlot3D[{t, 0, 0}, 0.5 + 0.3 Cos[t], {t, 0, 4 π}]
```



```
TubePlot3D[{Sin[t], Cos[t], 0}, 0.5, {t, 0, 2 π}, PlotPoints → 2]
```



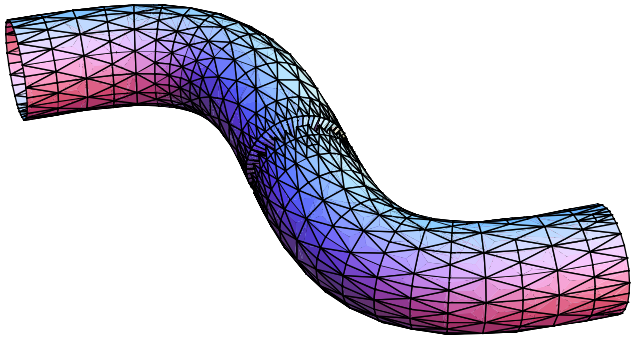
```
TubePlot3D[{Sin[t], Cos[t], 0}, 0.5, {t, 0, 2 π}, PlotPoints → 50]
```



An Wendepunkten der Mittenkurve treten Probleme auf, weil dann in den Koordinatenberechnungen des Dreibeins Ausdrücke der Form $\frac{0}{0}$ entstehen.

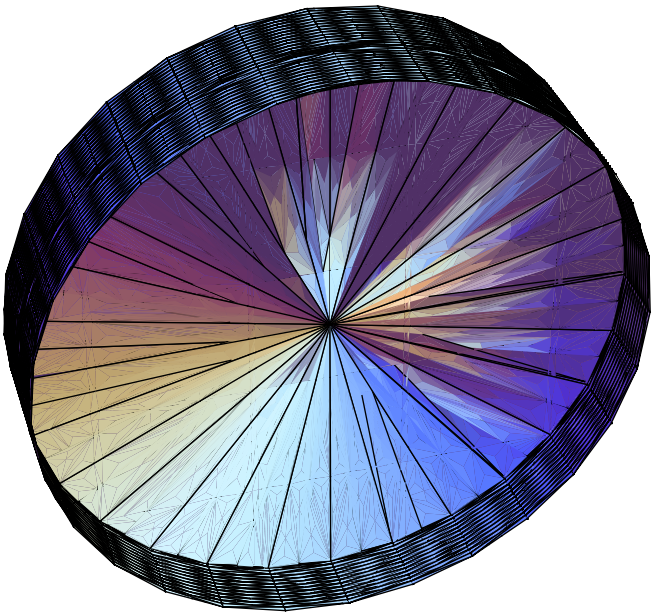
Deshalb eine Lücke in der Grafik.
Mit **Mesh → All** oder **Mesh → None** ist alles okay, weil dazu kein extra Netz auf die Figur aufgebracht wird.

```
TubePlot3D[{t, Sin[t], 0}, 0.5, {t, 0, 2 π}, Mesh → All]
```



Und hier etwas Code zum Studium des Verhaltens an der Unregelmäßigkeitsstelle.

```
u = .05;
TubePlot3D[{t, Sin[t], 0}, .3, {t, π - u, π + u}, Mesh → All]
```



▪ Zweite Lösung: Die "numerische", welche die Fläche unmittelbar aus Grafikprimitiven erzeugt

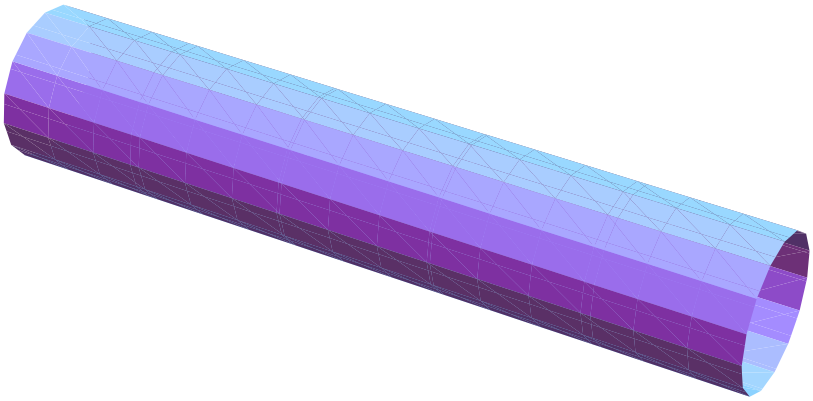
```
TPproject[n_?VectorQ, h_?VectorQ] := TPnorm[n - (h.n) / (h.h) h];
```

Und hier die Definition von **TubePlot3D**:

```
Clear[TubePlot3D];
TubePlot3D[func_?VectorQ, radiusfunc_: 0.1,
       {var_Symbol, start_, end_}, opts___?OptionQ] :=
Module[
   {f,  r, pList, hvList, nvList, bvList, rList, pts, u, delta, plotpts, nt, nr},
   plotpts = PlotPoints /. {opts} /. PlotPoints → {15, 15};
       Switch[plotpts,
            _List, {nt, nr} = plotpts[[{1, 2}]],
            _, nt = nr = plotpts];
        delta = N[(end - start)/nt];
        f[t_] := (func /. var → t);
        r[t_] := (radiusfunc /. var → t);
        pList = Table[f[i], {i, start, end, delta}];
        hvList = Table[TPnorm[f[i + delta/100] - f[i - delta/100]],
             {i, start, end, delta}];
       rList = Table[r[i], {i, start, end, delta}];

       (* Normalen- und Binormalenvektoren *)
       u = {1., 1., 1.};
       nvList = Map[Set[u, TPproject[u, #]] &, hvList];
       bvList = MapThread[TPnorm[Cross[#1, #2]] &, {hvList, nvList}];

       pts = MapThread[
         Table[#1 + #2 (Sin[u] #3 + Cos[u] #4), {u, 0., 2 π, 2 π/nr}] &,
             {pList, rList, nvList, bvList}];
   Graphics3D[Prepend[Table[
          With[{v = Join @@ pts[[{i, i + 1}, {j, j + 1}]]},
           {Polygon[v[[{1, 2, 3}]]], Polygon[v[[{2, 3, 4}]]]}],
         {i, nt}, {j, nr}], EdgeForm[]],
           DeleteCases[{opts}, PlotPoints → _], Boxed → False]];
(* Ende der Definition von TubePlot3D *)
```

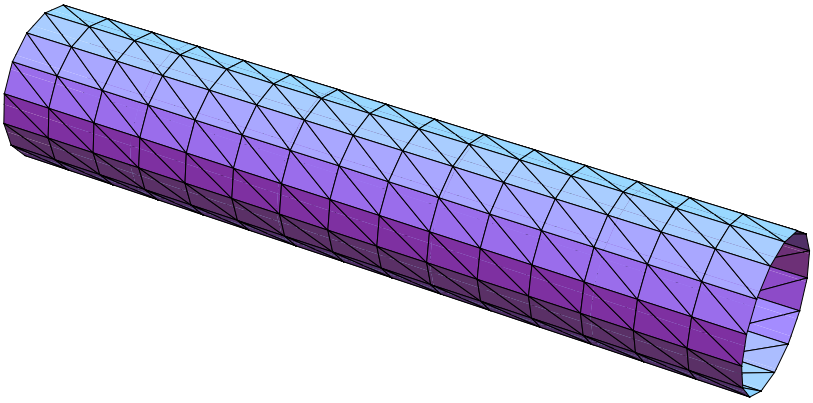
TubePlot3D anwenden

```
Clear[u]
p1 = TubePlot3D[{u, 0, 0}, {u, 0, 1}]
```

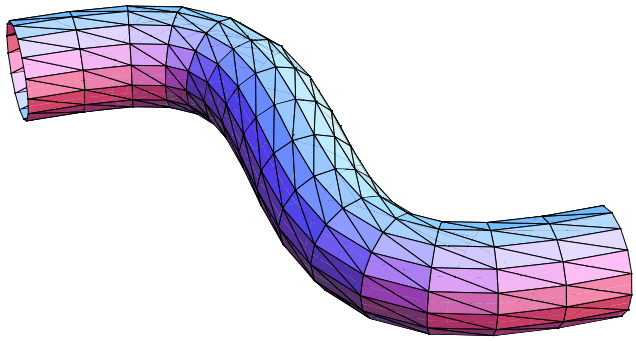


Durch das Einfügen von **EdgeForm[]** in die obige Definition werden die Polygone ohne Rand gezeichnet. Mit folgenden Kommandos kann der Rand nachträglich sichtbar gemacht oder auch wieder versteckt werden.

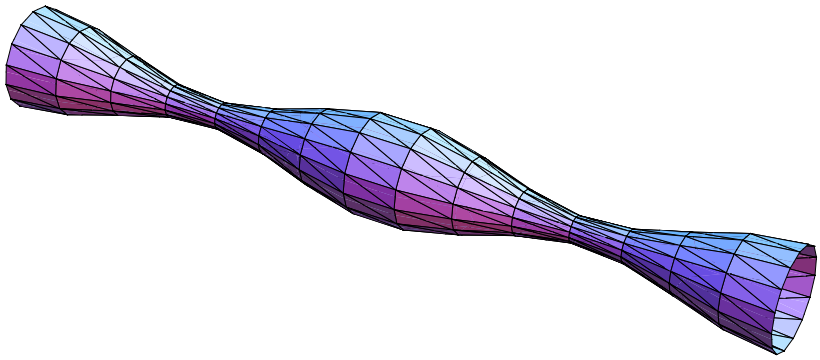
```
ShowEdges[g_Graphics3D] := g /. EdgeForm[] -> EdgeForm[Thickness[Tiny]]
NoEdges[g_Graphics3D] := g /. EdgeForm[_] -> EdgeForm[]

p1 // ShowEdges
```



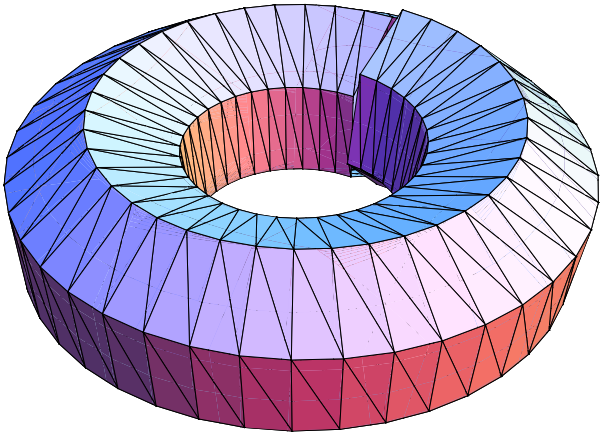
```
p2 = TubePlot3D[{t, Sin[t], 0}, 0.5, {t, 0, 2 π}] // ShowEdges
```



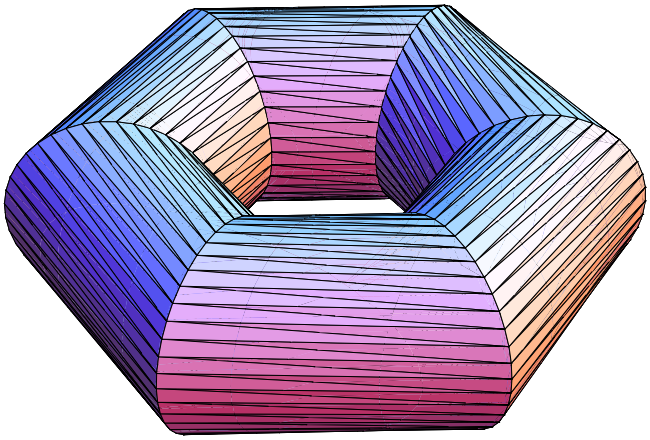
```
p3 = TubePlot3D[{t, 0, 0}, 0.5 + 0.3 Cos[t], {t, 0, 4 π}] // ShowEdges
```



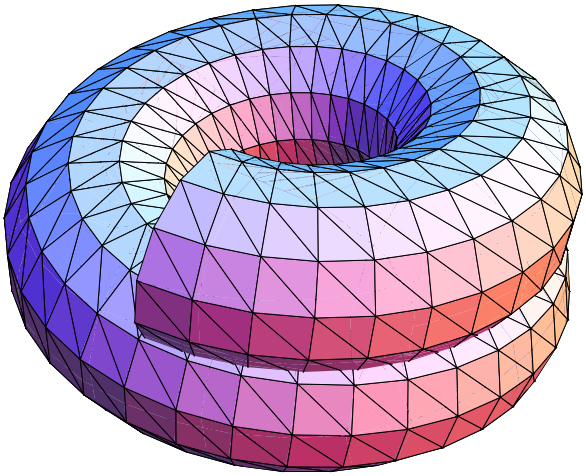
```
p4 = TubePlot3D[{Sin[t], Cos[t], 0}, 0.5, {t, 0, 2 π}, PlotPoints → {40, 6}] // ShowEdges
```



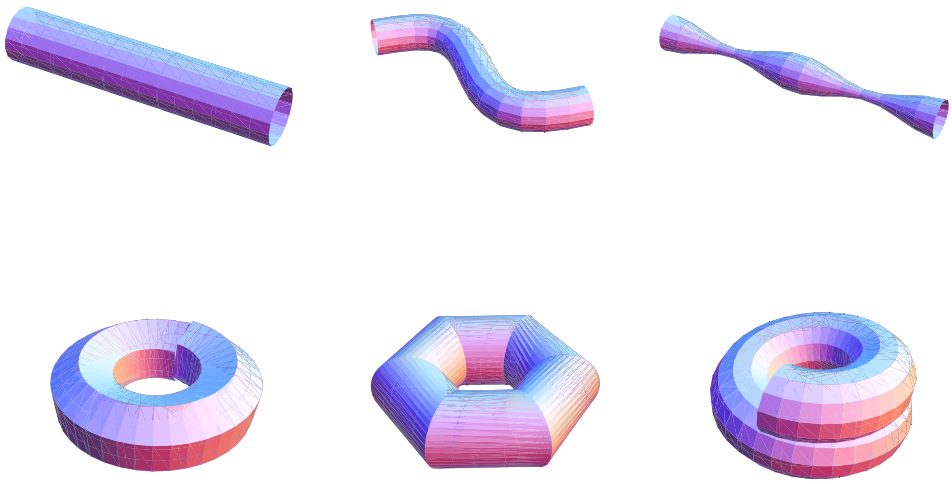
```
p5 = TubePlot3D[{Sin[t], Cos[t], 0}, 0.5, {t, 0, 2 π}, PlotPoints → {6, 40}] // ShowEdges
```



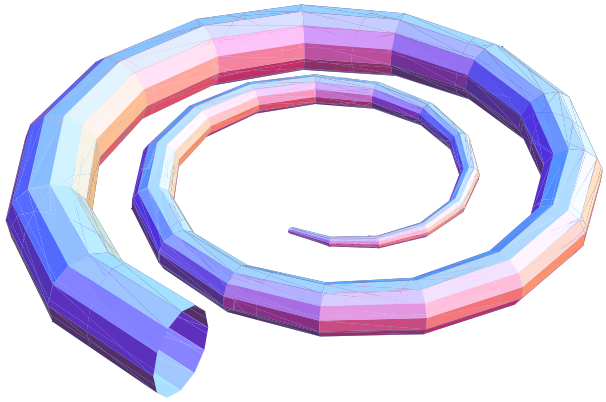
```
p6 =
 TubePlot3D[{Sin[t], Cos[t], t / 10}, 0.5, {t, 0, 3 π}, PlotPoints → {50, 12}] // ShowEdges
```



```
GraphicsGrid[{NoEdges /@ {p1, p2, p3}, NoEdges /@ {p4, p5, p6}}]
```



```
TubePlot3D[{(3 + t/2) Sin[t], -(3 + t/2) Cos[t], 0}, (1 + t)/10, {t, 0, 4 π}, PlotPoints → {30, 12}]
```



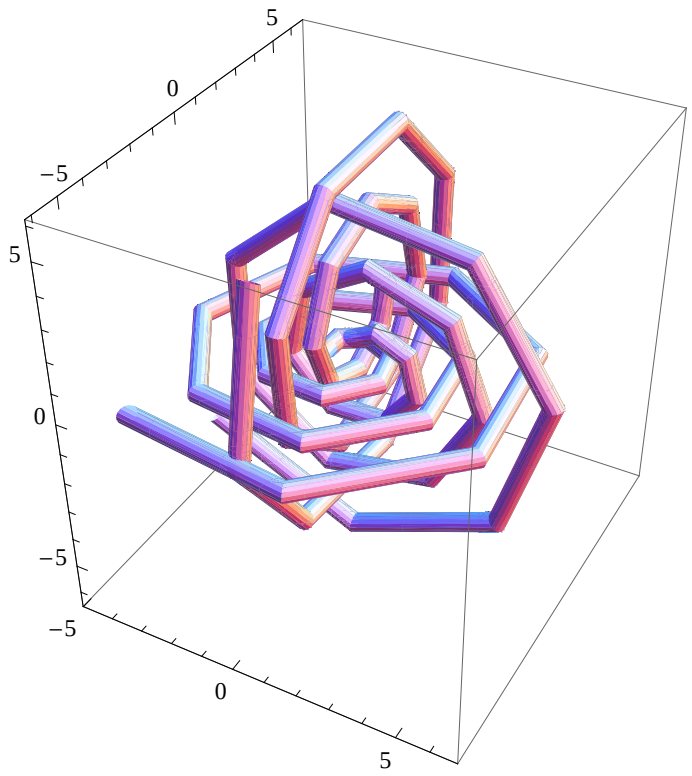
Titelbild

Auch das Titelbild des Buches wurde mit **TubePlot3D** erstellt. Hier zunächst die Einzelteile, aus denen der Knoten zusammengesetzt wurde.
In der Designphase arbeiten wir mit geringer Auflösung. Wenn alles zu unserer Zufriedenheit ausgefallen ist, werden alle Rechnungen noch einmal mit hoher Auflösung ausgeführt und das endgültige Bild erzeugt.
Dazu speichern wir die Auflösung in einer Variablen **res**, deren Wert über die Option **PlotPoints** die Auflösung steuert.

```
f1 = {(1 + t/π) Sin[t], (1 + t/π) Cos[t], 0};
f2 = {.8 + (1 + t/π) Sin[t], 0, (1 + t/π) Cos[t]};
f3 = {0, 1 + (1 + t/π) Sin[t], 1 + (1 + t/π) Cos[t]};

res = {20, 15};
p1 = TubePlot3D[f1, .3, {t, -0.5 π, 5.2 π}, PlotPoints → res];
p2 = TubePlot3D[f2, .3, {t, -.2 π, 5.3 π}, PlotPoints → res];
p3 = TubePlot3D[f3, .3, {t, π / 2, 5.7 π}, PlotPoints → res];
```

Der Knoten ohne Farben. Die Färbung kommt ausschließlich durch die Standardlichtquellen zustande.

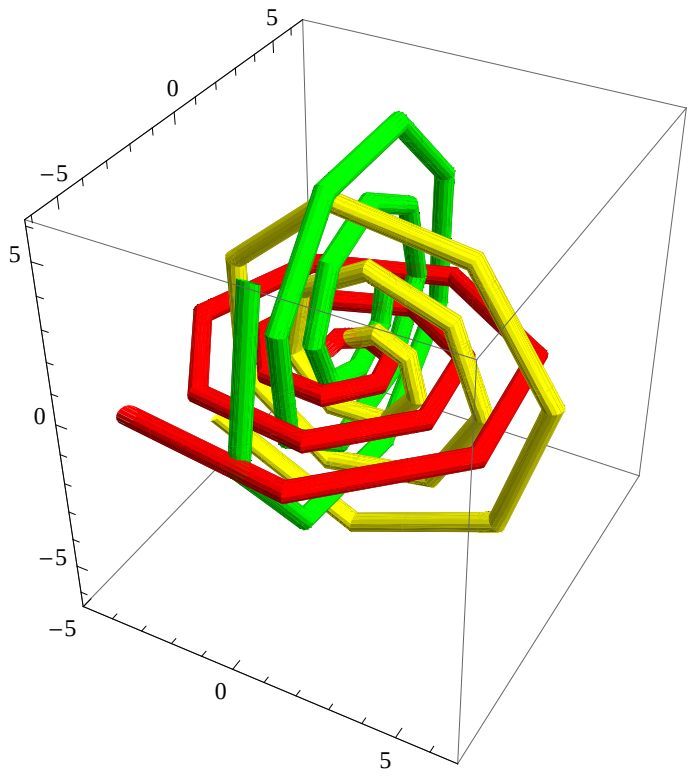
```
Show[{p1, p2, p3}, ViewAngle -> 30 °, Boxed -> True, Axes -> True]
```



Der Knoten mit Eigenfarben.

```
p1a = p1 /. EdgeForm[] → Directive[EdgeForm[], Red];
p2a = p2 /. EdgeForm[] → Directive[EdgeForm[], Yellow];
p3a = p3 /. EdgeForm[] → Directive[EdgeForm[], Green];
```

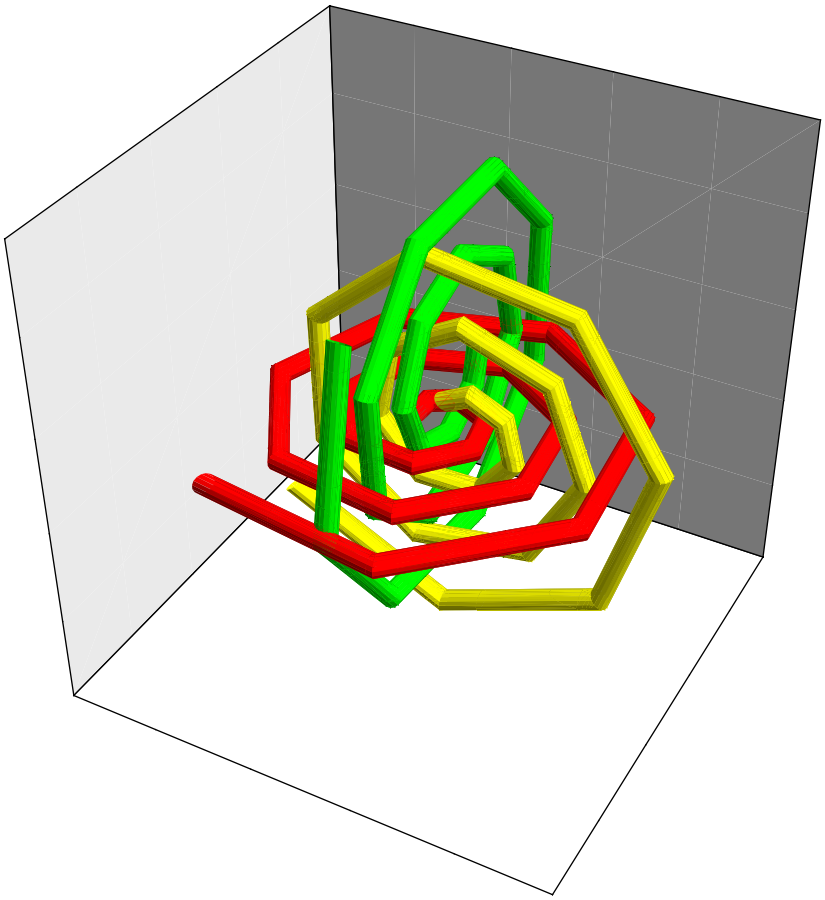
```
Show[{p1a, p2a, p3a}, Lighting -> "Neutral", ViewAngle -> 30 °, Boxed -> True, Axes -> True]
```



Mit der aktuellen Version von *Mathematica* lässt sich das Titelbild des Buches leider nciht erzeugen, weil Schattenwurf generell nicht berücksichtigt wird. Nicht einmal, wenn drei Seitenwände explizit eingezogen werden.

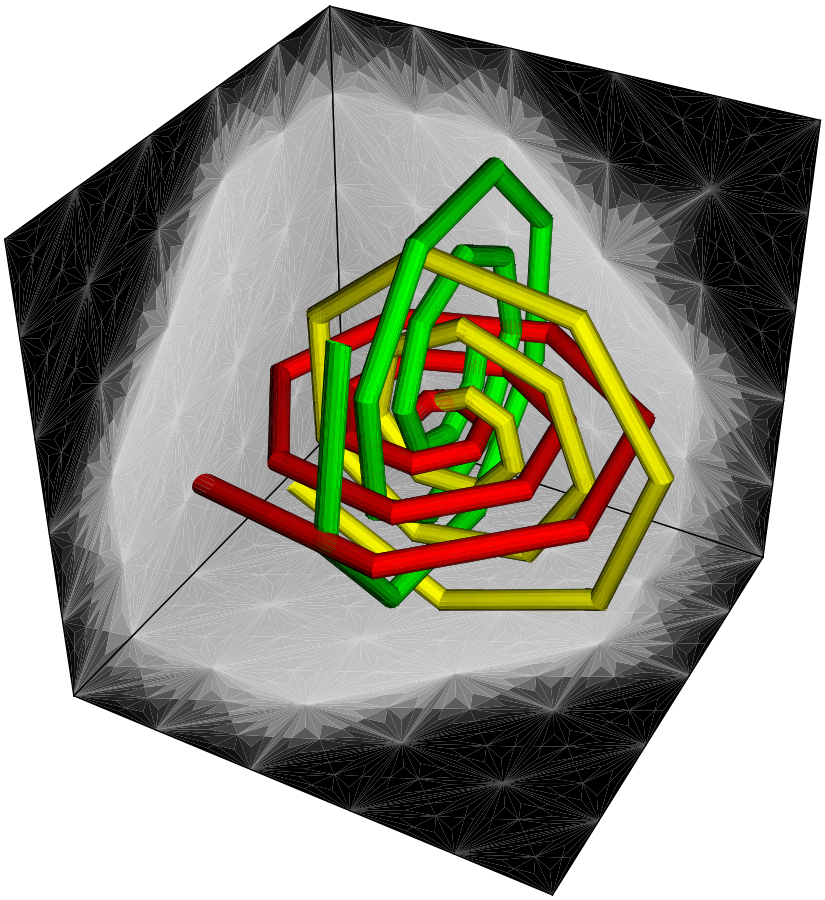
```
u = 7 {{1, 1, 1}, {1, 1, -1}, {1, -1, -1}, {1, -1, 1}, {1, 1, 1}};
box = {Polygon[-u], Polygon[-RotateLeft /@ u], Polygon[RotateRight /@ u]};
gbox = Graphics3D[box];
sbox = Graphics3D[{Specularity[1], box}];
```

```
g1 = Show[{gbox, p1a, p2a, p3a}, Lighting -> "Neutral", Boxed -> False]
```



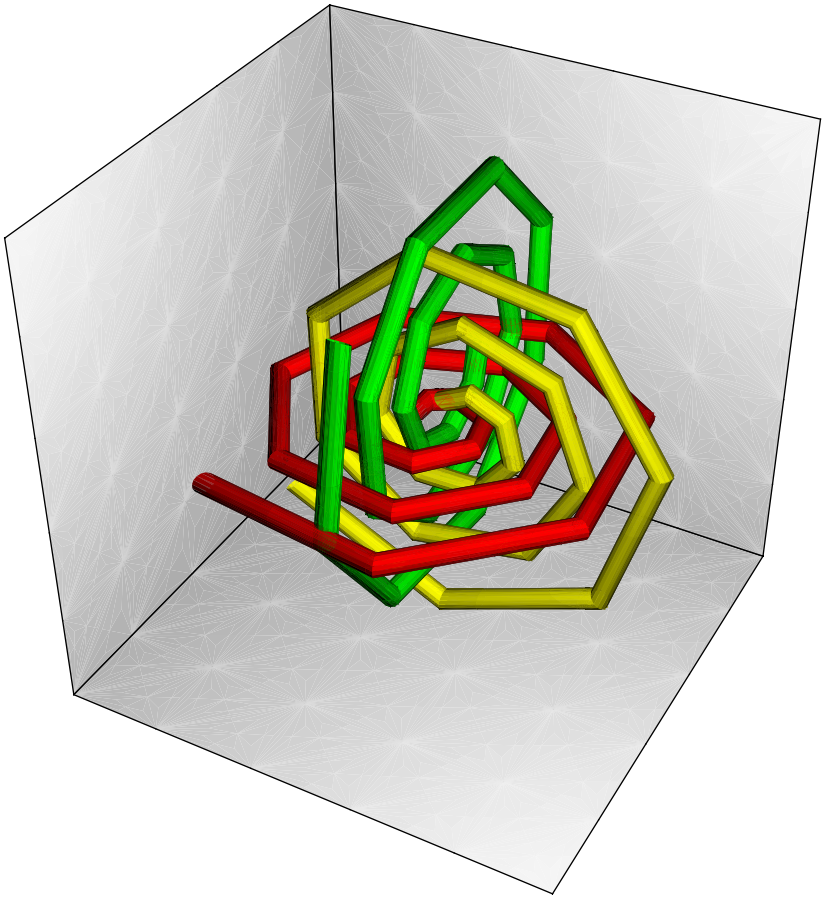
Das mit dem Spotlicht funktioniert bestens, wenn man einmal davon absieht, dass der runde Lichtkegel geradlinig begrenzte Hell−Dunkel−Bereiche auf den Seitenwänden erzeugt.

```
l = {{"Spot", White, {12 {1, -1, 1}, {0, 0, 0}}, 20 °}};
g2 = Show[{gbox, p1a, p2a, p3a}, Lighting -> l, Boxed -> False]
```

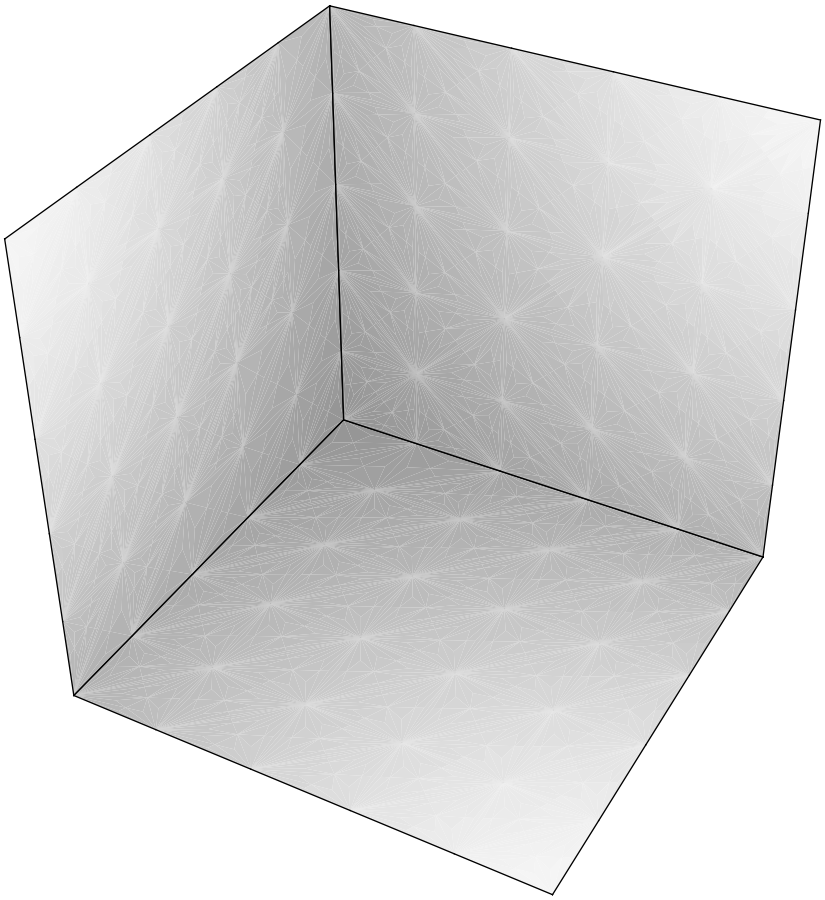


Auch bei einer punktförmigen Lichtquelle sieht das Bild nicht anders aus. Die diffusen Schatten auf den Seitenwänden stammen nicht von unserem Knoten, sondern sind Reflexe der Lichtquelle selbst, wie ein Bild ohne den Knoten zeigt.

```
l = {{"Point", White, 10 {1, -1, 1}}};
g3 = Show[{gbox, p1a, p2a, p3a}, Lighting -> l, Boxed -> False]
```



```
g4 = Show[{gbox}, Lighting -> l, Boxed -> False]
```



```
GraphicsGrid[{{g1, g2}, {g3, g4}}]
```

Das beste, was man derzeit erreichen kann, ist also das folgende Bild.

```
res = {150, 25};
p1 = TubePlot3D[f1, .3, {t, -0.5 π, 5.2 π}, PlotPoints → res];
p2 = TubePlot3D[f2, .3, {t, -.2 π, 5.3 π}, PlotPoints → res];
p3 = TubePlot3D[f3, .3, {t, π / 2, 5.7 π}, PlotPoints → res];
p1a = p1 /. EdgeForm[] → Directive[EdgeForm[], Red];
p2a = p2 /. EdgeForm[] → Directive[EdgeForm[], Yellow];
p3a = p3 /. EdgeForm[] → Directive[EdgeForm[], Green];

l = {{"Point", White, 10 {1, -1, 1}}};
p = Show[{p1a, p2a, p3a}, Lighting -> l, ViewAngle -> 20 °]

p = Show[{p1a, p2a, p3a}, Lighting -> "Neutral", ViewAngle -> 20 °]

p // ByteCount
```

```
7 076 544
```

Export in eine gif−Datei mit (standardmäßig) 360x360 Pixeln.

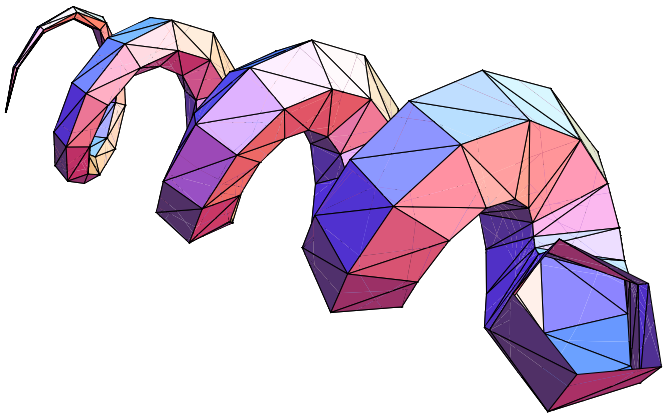
```
Export["cover.gif", p]

Export["cover.eps", p]
```

Und das war das alte Kommando; eine gif−Datei mit 3000x3000 Pixeln

```
Export["cover1.gif", p, ImageSize → {3000, 3000}];
```

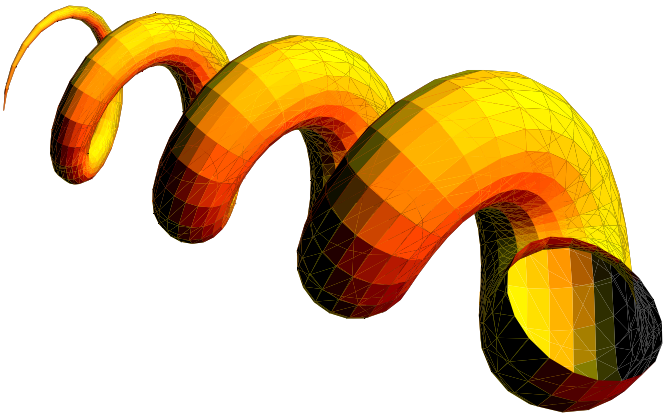
■ TubePlot3D und CutOut

```
ShowEdges[g_Graphics3D] := g /. EdgeForm[] -> EdgeForm[Thickness[Tiny]]
NoEdges[g_Graphics3D] := g /. EdgeForm[_] -> EdgeForm[]

p = TubePlot3D[{t^(3/2)/5, -3 Cos[t], 3 Sin[t]},
   t/8, {t, 0.1, 8 π}, PlotPoints → {40, 5}] // ShowEdges
```



```
p = TubePlot3D[{t^1.5/5, -3 Cos[t], 3 Sin[t]}, t/8, {t, 0.1, 8 π}, PlotPoints → {80, 20}];
```

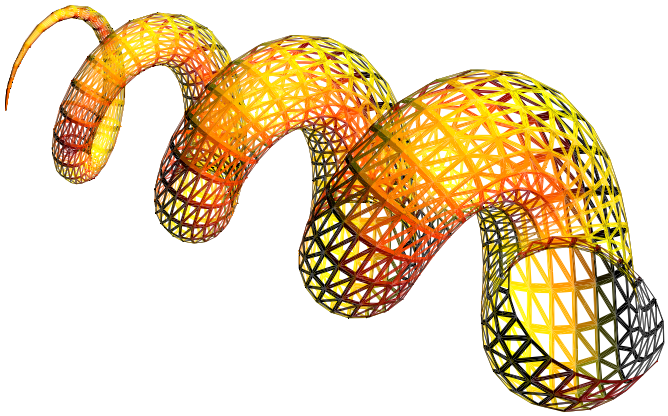
```
l = {{Red, ImageScaled[{1, 0, 1}]}, {Yellow, ImageScaled[{1, 1, 1}]}};
q = Show[p, Lighting → l]
```



Das folgende Kommando braucht die Definition von **CutOut** für komplexe Grafikobjekte aus dem Abschnitt "CutOut" dieses Notebooks.

```
cutout[p_List, α_, opts___?OptionQ] :=
     Module[{l, q},
      q = Map[((1 - α) Mean[p] + α #) &, p];
      l = Thread[{p, RotateLeft[p], RotateLeft[q], q}];
      Polygon /@ l
     ]
CutOut[g_Graphics3D, α_?(0 < # < 1 &)] :=
 Normal[g] /. Polygon[x_, opts___?OptionQ] → cutout[x, α, opts]
```

```
CutOut[q, 0.66]
```



■ Weitere Illustrationen

```
ClearAll["Global`*"]
TPlen[x_] := √(x[[1]]^2 + x[[2]]^2 + x[[3]]^2) ;
TPnorm[x_] := x / TPlen[x] // Together;
TPproject[n_?VectorQ, h_?VectorQ] := TPnorm[n - (h.n) / (h.h) h];
```

■ Einige Funktionen

Ebene durch P mit Normalenvektor h. Q dient der Ausrichtung des Bildes.

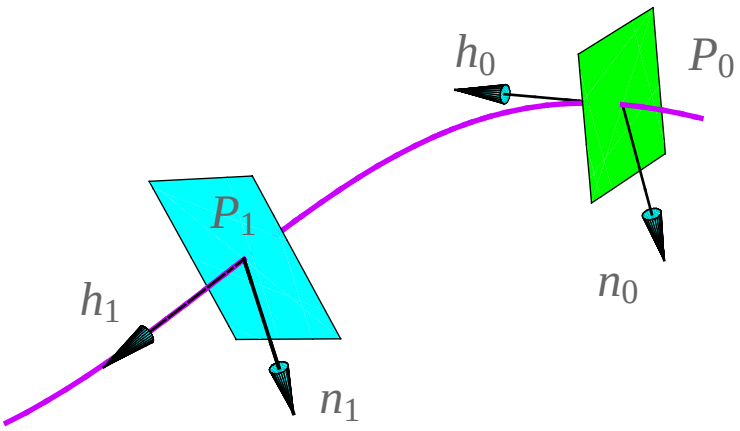
```
ebene[P_, Q_, h_] := Module[{n, b},
  n = Q - P;
  n = n - (h.n) / (h.h) h;
  b = h × n;
  Polygon[P + # & /@ {n, b, -n, -b}]
 ]

normale[h_] :=
 -If[h[[2]] === 0., {h[[3]], 0, -h[[1]]}, {0, h[[3]], -h[[2]]}] // TPnorm
```

Pfeil von P0 nach P1. a1 bestimmt die Länge, a2 die Dicke der Pfeilspitze.

```
pfeil[P0_, P1_, a1_, a2_] := Module[{v, n, b, Q, u},
  v = P1 - P0;
  Q = a1 P0 + (1 - a1) P1;
  n = normale[v]; b = v × n // TPnorm;
  b = Table[Q + a2 (Sin[u] n + Cos[u] b), {u, 0, 2 π, π / 12}];
  Append[
   Polygon /@ Append[Append[#, P1] & /@ Thread[{b, RotateLeft[b]}], b], Line[{P0, P1}]]]
```

Und nun die Bilder

```
mk[t_] := {t, 0, Sin[t]}; (* Mittenkurve *)
curve = Line[Table[mk[t], {t, 0.6, 4.2, .1}]];
t0 = 1.; t1 = 3.;
P0 = mk[t0]; h0 = mk'[t0] // TPnorm; n0 = normale[h0]; b0 = h0 × n0;
P1 = mk[t1]; h1 = mk'[t1] // TPnorm; n1 = TPproject[n0, h1];

Graphics3D[{
  {Hue[.8], AbsoluteThickness[2], curve},
  {FaceForm[Hue[0.3]], ebene[P0, {0, .5, 0}, h0]},
  {FaceForm[Hue[.5]], ebene[P1, {2., .3, 2}, h1]},
  {AbsoluteThickness[1], pfeil[P0, P0 + h0, .3, .05]},
  {AbsoluteThickness[1], pfeil[P0, P0 + n0, .3, .05]},
  {AbsoluteThickness[1], pfeil[P1, P1 + h1, .3, .05]},
  {AbsoluteThickness[1.4], pfeil[P1, P1 + n1, .3, .05]},
  Text["P0", P0 + {-.5, .1, .1}],
  Text["P1", P1 + {0, .2, .1}],
  Text["n0", P0 + {.3, -1.3, .3}],
  Text["h0", P0 + {.7, .3, .3}],
  Text["n1", P1 + {-.2, -1.2, .3}],
  Text["h1", P1 + {1.1, -1., 1.4}]
 }, Boxed → False, ViewPoint -> {-1., 4., -6.},
 TextStyle → {FontSize → 18}, Lighting → {RGBColor[0, 1, 1]}]
```



```
P0 = {0, 0, 0}; h = -{0, 0, 1.}; n = {3., 1., 7.} // TPnorm;
p = n - (h.n) / (h.h) h; p0 = p // TPnorm;

g = Graphics3D[{RGBColor[0., 0., 0.],
    {FaceForm[Orange], ebene[P0, {1, 1, 1}, h]},
    {AbsoluteThickness[1], pfeil[P0, P0 + h, .2, .03]},
    {AbsoluteThickness[1], pfeil[P0, P0 + n, .2, .03]},
    {AbsoluteThickness[1], pfeil[P0, P0 + p, .3, .03]},
    {AbsoluteThickness[1], pfeil[P0, P0 + p0, .2, .03]},
    {AbsoluteDashing[{5, 5}], AbsoluteThickness[1], Line[{n, p}]},
    Text["h", h + {-.1, .1, .1}],
    Text["n", n + {-.1, .1, 0}],
    Text["p", p - {0, .1, .1}],
    Text["p0", p0 - {.1, .1, .1}]
  }, Boxed -> False, ViewPoint -> {1., -12., 6.}, Lighting → {White}]
```