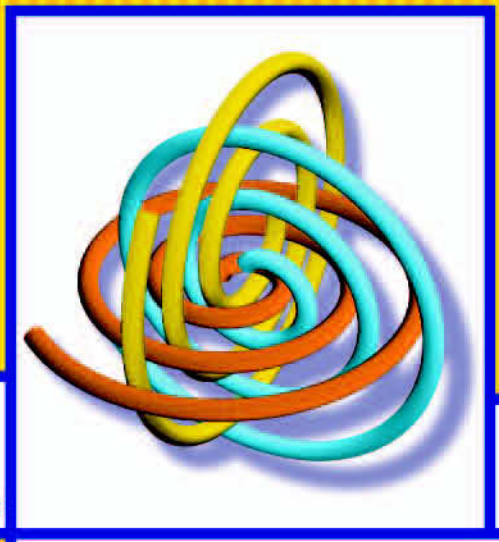
st
scientific tools



Hans-Gert Gräbe
Michael Kofler

Mathematica 6

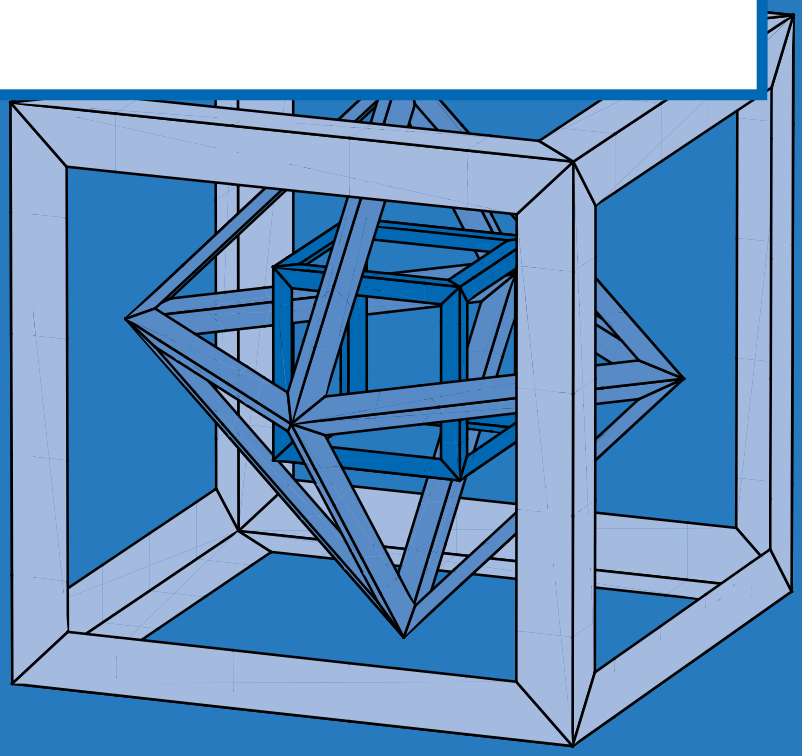
Einführung, Grundlagen, Beispiele

5., aktualisierte Auflage

PEARSON
Studium

TEIL I

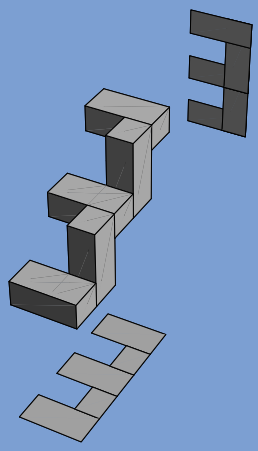
Mathematica kennenlernen



Mathematica als Visualisierungswerkzeug

3

ÜBERBLICK



» Neben der formelmäßigen Bearbeitung von Problemen mit verschiedenen algorithmischen Verfahren spielt die Visualisierung von Zusammenhängen und Ergebnissen zum besseren Verständnis realer Prozesse eine wichtige Rolle. Da die Erzeugung solcher Visualisierungen ein hochgradig algorithmischer Vorgang ist, können CAS auch hierbei ihre Stärken ausspielen. Dieses Zusammenspiel von Rechen-, Visualisierungs- und Präsentationsleistung ist der Kern dessen, weshalb sich *Mathematica* mit vollem Recht als „vollständig integrierte Umgebung für technisches Rechnen“ bezeichnet[1].

Mit Version 6 wurde das Grafikkonzept von *Mathematica* grundlegend überarbeitet, wobei vor allem die bisherige enge Anbindung an das Postscript-Format zugunsten eines eigenen Ausgabeformats für Grafiken aufgegeben wurde. Damit können dreidimensionale Grafiken endlich auch interaktiv manipuliert und Animationen nach einheitlichen Gesichtspunkten erstellt werden. Dabei sind viele neue Funktionen hinzugekommen und einge alte zugunsten einer Vereinheitlichung der Konzepte verschwunden. Letzteres bezieht sich vor allem auf die verfügbaren Optionen, mit denen die Darstellung modifiziert werden kann. Sie müssen natürlich nicht alles vergessen, was Sie über Grafiken wissen, wenn Sie schon mit früheren Versionen von *Mathematica* gearbeitet haben, denn viele Gestaltungselemente sind auch in der neuen Version verfügbar. Die Ausführungen zu Grafiken in diesem Buch beziehen sich allerdings ausschließlich auf Version 6.

Wir können nicht auf die Vielfalt der grafischen Darstellungsmöglichkeiten in systematischer Weise eingehen, denn das würde ein zweites Buch dieses Umfangs füllen. Beispiele, in denen Grafiken erstellt werden, finden Sie an verschiedenen Stellen unseres Buchs. In diesem Kapitel sind grundlegende Informationen zu Grafiken und deren Aufbau zusammengestellt, wobei wir uns einige wenige Vorgriffe auf sprachliche Mittel erlaubt haben, die erst später in diesem Buch besprochen werden. In Kapitel 8 nehmen wir den Faden noch einmal auf, erläutern das Farb- und Beleuchtungsmodell und zeigen, wie sich umfangreichere Grafiken mit denselben programmiersprachlichen Mitteln erstellen lassen, die auch in der Bearbeitung stärker computer-algebraisch orientierter Probleme zur Anwendung kommen. «

3.1 Einführung

Die Visualisierung von Zusammenhängen ist ein wichtiges Moment in der Analyse realer Prozesse. Ein Bild sagt oft mehr als zehn Formeln oder 1000 Daten. Mit leistungsfähigen Computern eröffnen sich auch hier vollkommen neue Möglichkeiten. Meist wird zur Visualisierung spezielle Grafik-Software und oft sogar spezielle Grafik-Hardware verwendet. Natürlich kann ein Allroundsystem wie *Mathematica* mit der Leistungsfähigkeit und Bildqualität solcher Systeme nicht mithalten. Der besondere Reiz des Einsatzes eines CAS auf diesem Gebiet besteht (neben dem Preisunterschied) in der Möglichkeit der engen Verknüpfung von analytischen und Visualisierungswerkzeugen, die es ohne großen Aufwand erlauben, die grafischen Auswirkungen von Modifikationen zu untersuchen.

1 „Mathematica is the world’s only fully integrated environment for technical computing“ lautete der erste Satz im Handbuch zu Version 4 (Handbuch, S. ix).

Die verfügbaren Kommandos und Optionen laden damit auch auf diesem Gebiet zum Experimentieren ein. Sie eignen sich nicht nur dazu, komplexe mathematische Zusammenhänge optisch darzustellen, sondern können und werden vielfach auch zur Herstellung von professionellen Abbildungen für Publikationen eingesetzt. Die Grafikkommandos von *Mathematica* lassen sich durch zahllose Optionen steuern, so dass die Darstellungsmöglichkeiten praktisch grenzenlos sind.

Mit dem Plot- oder Plot3D-Kommando lassen sich Grafiken besonders einfach erzeugen. Dem Kommando werden das darzustellende Objekt in Koordinatenform, die Koordinatenbereiche $\{x, x_s, x_e\}$ mit Bezeichner, Start- und Endwert sowie optional weitere Einstellungen übergeben:

```
Plot[f[x], {x, xs, xe} , Optionen]   oder
Plot3D[f[x, y], {x, xs, xe} , {y, ys, ye} , Optionen]
```

Im Gegensatz zu früheren *Mathematica*-Versionen wurde die Arbeitsteilung zwischen Frontend und Kernel beim Abarbeiten eines Plot-Kommandos verändert: Plot als Kernel-Prozedur erzeugt als Rückgabewert eine medienunabhängige textuelle Repräsentation des Grafikobjekts und gibt diese an das Frontend (oft in komprimierter Form) weiter. Erst dort wird das Grafikobjekt dargestellt. Die grafische Darstellung erfolgt damit im Gegensatz zu früheren *Mathematica*-Versionen nicht mehr als Seiteneffekt eines Prozeduraufrufs, sondern wird nun automatisch vom jeweiligen Frontend übernommen.

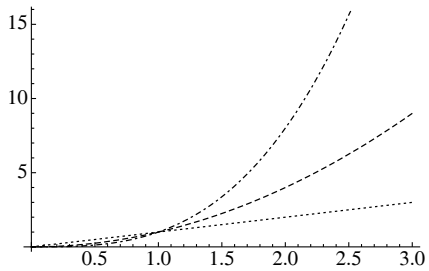
Der gravierendste Unterschied: Früher wurden Grafikkommandos meist durch einen Strichpunkt abgeschlossen, um zusätzliche Ausgaben zu unterdrücken. Mit Version 6 wird damit aber die Ausgabe des Grafikobjekts unterdrückt und das Frontend hat nichts darzustellen. Die Ausgabezelle bleibt leer!

> ***Hinweis:*** Wenn also *Mathematica* in der Version 6 trotz sorgfältiger Eingabe partout kein Bild erzeugen will – schauen Sie nach, ob da vielleicht ein unauffälliger Strichpunkt am Ende des Kommandos steht!

Das darzustellende Objekt kann auch aus einer Liste von Teilobjekten bestehen, wobei über entsprechende Listen von Optionen die Erscheinungsweise jedes dieser Teilobjekte separat gesteuert werden kann.

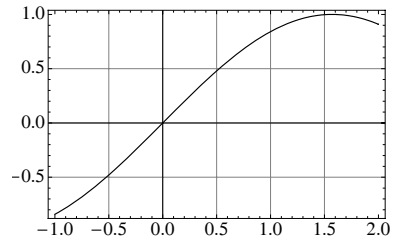
Diese Grafik vergleicht das Wachstumsverhalten verschiedener Potenzfunktionen. Die Funktionen x, x^2 und x^3 sind dank verschiedener Einstellungen für die Linienform in der Plot-Option PlotStyle leicht unterscheidbar.

```
Plot[{x, x^2, x^3}, {x, 0, 3}, PlotStyle →
  {Dotted, Dashed, DotDashed}]
```



Hier wurde von der Standarddarstellung abgewichen und statt der Koordinatenachsen ein Rahmen mit Gitterlinien verwendet.

```
Plot[Sin[x], {x, −1, 2},
    GridLines → Automatic,
    Frame → True]
```

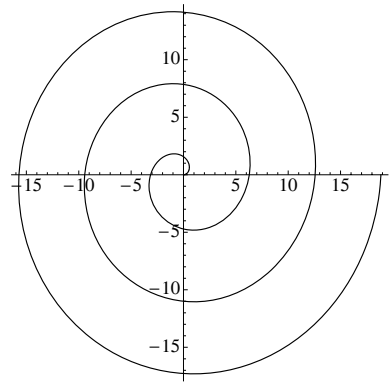


Neben gewöhnlichen Funktionen der Form $y = f(x)$ kommt *Mathematica* auch mit in Parameterform gegebenen Funktionen

$$\{(x = x(t), y = y(t)),\ t \in \mathbb{R}\}$$

zurecht. Die Option AspectRatio verhindert eine Verzerrung der Proportionen.

```
ParametricPlot[{t Cos[t], t Sin[t]},
    {t, 0, 6 π}, AspectRatio → 1]
```



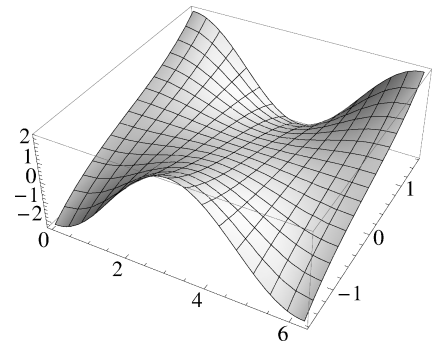
Beachten Sie, dass sich AspectRatio nicht auf die Maßeinteilung auf den Koordinatenachsen bezieht, sondern auf die Größenverhältnisse der 2D-Begrenzungsbox, in welcher das Bild dargestellt wird, und die Sie als orangefarbigen Rahmen angezeigt bekommen, wenn Sie die Grafik im Notebook markieren.

Mit AbsoluteOptions können Sie sich den verwendeten Wert von AspectRatio anzeigen lassen. Ohne die Option AspectRatio → 1 wäre in diesem Fall das Bild um den Faktor 0.91011 verzerrt worden.

```
ParametricPlot[
  {t Cos[t], t Sin[t]}, {t, 0, 6 π}];
AbsoluteOptions[%, AspectRatio]
  {AspectRatio → 0.91011}
```

Mindestens ebenso viele Gestaltungsmöglichkeiten gibt es bei dreidimensionalen Grafiken. Parametrische Funktionen können nicht nur in kartesischen Koordinaten, sondern auch in Zylinder- und Kugelkoordinaten gezeichnet werden. Alle 3D-Grafiken können beliebig gedreht und skaliert werden. Sowohl die Blickrichtung auf die Grafik als auch deren Beleuchtung durch verschiedene Lichtquellen sind frei einstellbar.

```
Plot3D[Im[Sin[x + I y]],
  {x, 0, 2 π}, {y, −π/2, π/2}];
```



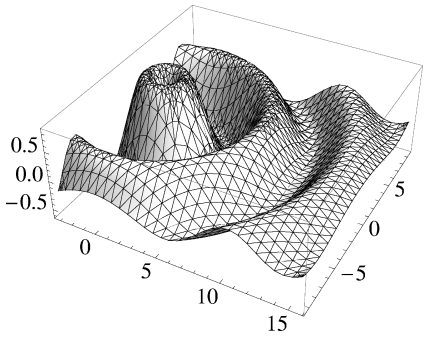
Diese Abbildung zeigt den Imaginärteil der komplexen Sinusfunktion als Fläche über der Gaußschen Zahlenebene.

Im Gegensatz zu früheren Versionen hat im neuen Grafikformat der Version 6 das auf der Fläche angezeigte Gitter nichts mehr mit der Triangulierung der Fläche zu tun, sondern es stellt ein separates Grafikobjekt dar, welches im Nachgang auf die Fläche aufgebracht wird.

Einstellungen wie die Option PlotPoints, die sich auf die Triangulierung beziehen, haben somit keine Auswirkung mehr auf dieses Gitter. Mit der Einstellung Mesh → All können Sie sich die wirkliche Triangulierung anzeigen lassen, die *Mathematica* verwendet.

```
Plot3D[
  Sin[√(x^2 + y^2)] Exp[-0.1 √(x^2 + y^2)],
  {x, -π, 5π}, {y, -3π, 3π},
  PlotPoints → 30, Mesh → All,
  Lighting → "Neutral"]
```

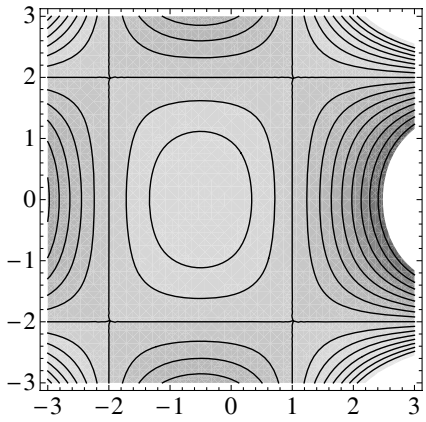
Diese Grafik ist außerdem ohne die sonst übliche farbliche Schattierung dargestellt, da über die Option Lighting das Ausleuchten der Szene mit weißem Licht eingestellt ist.



Schließlich gibt es noch die Möglichkeit, zweidimensionale Darstellungen von drei- oder höherdimensionalen Strukturen herzustellen, indem einzelne Dimensionen durch Schattierungen oder Farben repräsentiert werden.

DensityPlot und ContourPlot sind zwei solche Funktionen, wobei letztere Funktion eine Darstellung durch Höhenlinien erzeugt, in welcher der Funktionswert einer Höhenlinie als Tooltip angezeigt wird, wenn man mit der Maus über die Linie fährt. Die Zahl der Höhenlinien kann mit der Option Contours eingestellt werden. Hier werden 20 äquidistante Höhenlinien erzeugt.

```
f = (x - 1) (x + 2) (y - 2) (y + 2);
ContourPlot[f, {x, -3, 3}, {y, -3, 3},
    Contours → 20]
```

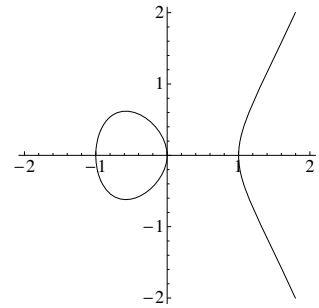


ContourPlot mit einem booleschen Ausdruck als erstem Parameter, etwa $f == 3$, zeichnet alle Punkte der untersuchten Region, in denen der Ausdruck wahr ist. Auf diese Weise lässt sich das Kommando verwenden, um einzelne Höhenlinien zu zeichnen, womit die Grafik eines implizit gegebenen funktionalen Zusammenhangs konstruiert werden kann[2]. Die Ergebnisse sind sowohl in zwei als auch in drei Dimensionen sehr gut.

2 Diese Variante löst das Kommando ImplicitPlot aus älteren *Mathematica*-Versionen ab.

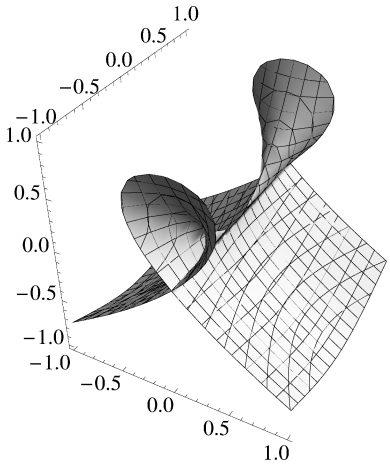
Als Beispiele betrachten wir die Darstellung der elliptischen Kurve $y^2 = x^3 - x$ ...

```
ContourPlot[y^2 == x^3 - x,
    {x, -2, 2}, {y, -2, 2},
    Frame → False, Axes → True]
```

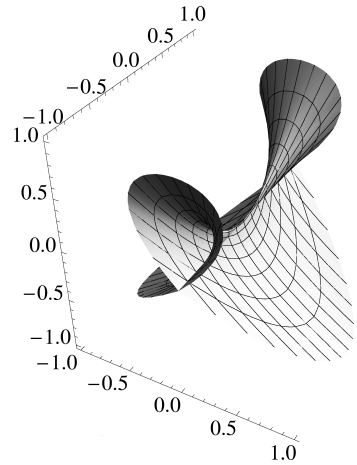


... sowie die auf der Frontseite des Buchs [Cox u.a. 1992] abgebildete „Schweinsohrfläche“ $x^2 - y^2 z^2 + z^3 = 0$. Diese Fläche hat die zum Erzeugen einer Grafik besser geeignete parametrische Darstellung $\{t(u^2 - t^2), u, u^2 - t^2\}$ [ebenda, S. 16]. Beide Darstellungen unterscheiden sich – im Gegensatz zu Grafiken, die für dieses Beispiel von anderen CAS produziert werden – nur unwesentlich.

```
ContourPlot3D[
  x^2 - y^2 z^2 + z^3 == 0,
    {x, -1, 1}, {y, -1, 1}, {z, -1, 1}]
```




```
ParametricPlot3D[
  {t (u^2 - t^2), u, u^2 - t^2},
    {t, -1, 1}, {u, -1, 1}]
```



3.2 Zum Aufbau von Grafiken

Grafiken werden vom *Mathematica*-Kern aus *Grafikobjekten* erzeugt. Grafiken sind symbolische textuelle Objekte wie andere *Mathematica*-Ausdrücke auch, die vom Frontend allerdings nicht in dieser textuellen Version ausgegeben, sondern in einer Grafikzelle als Bild dargestellt (gerendert) werden. Mit CELL|SHOW EXPRESSION oder dem Tastenkürzel Strg+Shift+E können Sie sich in einem Notebook den textuellen Inhalt einer Grafikzelle anschauen und den genauen Aufbau eines solchen Ausdrucks studieren. Dasselbe Kommando führt in den Ausgangszustand zurück. Auch

InputForm zeigt diese (oft sehr umfangreiche, deshalb sinnvollerweise mit Short oder Shallow zu kombinierende) Darstellung an.

Die kleinsten selbstständig bildlich darstellbaren Einheiten sind die *Grafiken*. Im Regelfall sind dies Graphics- oder Graphics3D-Konstrukte, je nachdem, ob es sich um zwei- oder dreidimensionale Grafiken handelt. Verschiedene Funktionen wie GraphicsGroup oder Inset erlauben es, aus solchen einfachen Grafiken komplexere zusammenzusetzen, wobei das Containerprinzip zur Anwendung kommt, also solche komplexen Objekte selbst wieder Grafiken sind und damit ihrerseits als Bausteine für noch komplexere Konstrukte verwendet werden können. Zusammen mit der Möglichkeit, Grafiken auch für interaktive Mausoperationen zu konditionieren und zu dynamisieren, lassen sich so mit *Mathematica* sehr komplexe Anwendungsszenarien umsetzen.

Auf diese vielfältigen Gestaltungsmöglichkeiten können wir in unserem Buch nicht eingehen. Einmal fehlt der Platz und zum anderen ist ein Buch nicht das geeignete Medium, mit dem sich interaktive Szenarien eindrucksvoll darstellen lassen. Wir beschränken uns deshalb auf die Erstellung *einfacher Grafiken*, bei deren Erzeugung interaktive Elemente, Gruppierungsfunktionen oder dynamische Aspekte nicht zur Anwendung kommen.

Der Aufbau solcher einfacher Grafiken folgt einem allgemeinen Schema und wird durch eines der Kommandos

```
Graphics[prim, opt]    oder    Graphics3D[prim, opt]
```

realisiert, wobei prim eine geschachtelte Liste aus Grafikprimitiven und -direktiven ist und opt weitere Optionen enthält. Wie die Namen schon vermuten lassen, sind Graphics 2D-Objekte und Graphics3D 3D-Objekte. Während zweidimensionale Grafiken im Wesentlichen nur skaliert werden können, lassen sich dreidimensionale Grafiken mit der Version 6 nun endlich auch interaktiv manipulieren.

In Kapitel 8 gehen wir detaillierter auf das Farb- und Beleuchtungsmodell ein, welches bei der Darstellung von *Mathematica*-Grafiken zur Anwendung kommt, und wenden uns komplexeren Aspekten der Grafikprogrammierung zu. In diesem Abschnitt besprechen wir zunächst nur den prinzipiellen Aufbau von Grafiken aus Grafikprimitiven und -direktiven, der zum Grundverständnis des Grafikkonzepts von *Mathematica* erforderlich ist.

3.2.1 Grafikprimitive und -direktiven

Wie bereits erläutert, ist jedes einfache Grafikobjekt aus *Grafikprimitiven* und *Grafikdirektiven* aufgebaut. Grafikprimitive sind die eigentlichen Bausteine der Grafik, die Grafikdirektiven legen die Art der Darstellung (Farbe, Durchsichtigkeit, Linienstärken usw.) dieser Bausteine fest. Innerhalb einer Liste wirken sich Grafikdirektiven auf alle nachfolgenden Grafikprimitive aus, können aber überschrieben werden. Der Gültigkeitsbereich von Grafikdirektiven ist auf die aktuelle Liste (und alle Unterlisten) beschränkt, so dass sich durch verschachtelte Listen von Direktiven und Primitiven komplexere Phänomene bereits auf der Ebene der Grafikbausteine realisieren lassen.

Die elementarsten Bausteine aller Grafiken sind jedoch *Punktkoordinaten*, also 2D- oder 3D-Vektoren mit reellen Koordinaten. Aus ihnen werden die einfachsten Punkt-, Linien-, Flächen- und Körperprimitive aufgebaut. Neben diesen Primitiven gibt es

neu in Version 6 noch Gruppierungsprimitive, von denen wir das GraphicsComplex-Primitiv auf Seite 94 besprechen.

Hier sind zunächst einmal die Punkt-, Linien-, Flächen- und Körperprimitive in der jeweils einfachsten Form zusammengestellt. Die Bezeichner $P, P_1, P_2, \ldots$ stehen für Punktkoordinaten.

Grafikprimitive

Nulldimensionale Primitive

Point[P]	Punkt (2D und 3D)

Eindimensionale Primitive

Line[{$P_1, P_2, \ldots, P_n$}]	Stückweise geradliniger Linienzug (2D und 3D)
Arrow[{$P_1, P_2, \ldots, P_n$}]	Linienzug mit Pfeilspitze(n) (2D)
Circle[P, r]	Kreislinie um P mit Radius r (2D)
Circle[P, r, {θ_1, θ_2}]	Bogenlinie um P mit Radius r im angegebenen Sektor (2D)

Zweidimensionale Primitive

Disk[P, r]	Kreisscheibe um P mit Radius r (2D)
Disk[P, r, {θ_1, θ_2}]	Kreissektor um P mit Radius r (2D)
Polygon[{$P_1, P_2, \ldots, P_n$}]	Von einem geschlossenen geradlinigen Linienzug berandetes Flächenstück (2D und 3D)
Rectangle[P_1, P_2]	Rechteck mit P_1 und P_2 als gegenüberliegenden Ecken (2D)
Sphere[P, r]	Sphäre mit Zentrum P und Radius r (3D)

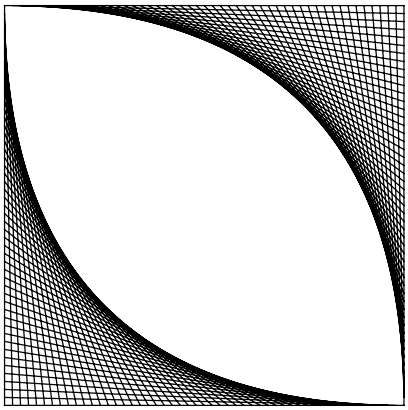
Dreidimensionale Primitive

Cuboid[P_1, P_2]	Quader mit P_1 und P_2 als gegenüberliegenden Ecken (3D)
Cylinder[{P_1, P_2}, r]	Zylinder mit Achse P_1P_2 und Radius r (3D)

Text

Text[text, P]	Text an der Position P in eine 2D- oder 3D-Grafik einbauen

Diese Grafik besteht aus Linien, die Punkte auf den Seiten eines Einheitsquadrats miteinander verbinden. Dazu werden in l_1 und l_2 mit Table erzeugte Listen von Line-Objekten (in diesem Fall Strecken) abgelegt, aus denen das Graphics-Objekt aufgebaut wird.



```
l1 = Table[Line[{{1, x}, {1 − x, 1}}],
    {x, 0, 1, 0.02}];
l2 = Table[Line[{{0, x}, {1 − x, 0}}],
    {x, 0, 1, 0.02}];
Graphics[{l1, l2}, AspectRatio → 1]
```

Die Option AspectRatio → 1 verhindert eine verzerrte Darstellung durch unterschiedliche Skalierung der Seiten der Begrenzungsbox, die sonst möglicherweise automatisch vorgenommen wird.

Grafikdirektiven können grob unterteilt werden in einfache Direktiven zur Steuerung von Punkt- und Strichstärken sowie zur Auswahl der Farbgebung und komplexe Direktiven, die das Aussehen zweidimensionaler Primitive steuern und dazu auf einfachen Direktiven aufbauen.

Zur Vereinbarung von Punkt- und Strichstärken können die folgenden Direktiven verwendet werden:

Grafikdirektiven für Punkt- und Linienstärken

AbsolutePointsize[d]	absolute Punktgröße
Pointsize[d]	relative Punktgröße
AbsoluteThickness[d]	absolute Linienstärke
Thickness[d]	relative Linienstärke
Dashing[...]	verschieden gestrichelte Linien

Relative oder absolute Angaben bewirken, dass die Punkte und Linien bei Skalierung der Grafik ihre Größe ändern oder nicht. In beiden Fällen ist d eine positive reelle Zahl ohne Einheit. Die relative Größe bezieht sich auf den Anteil an der Gesamtbreite der Grafik (womit nur $0 < d < 1$ sinnvolle Werte sind), die absolute Größe wird in der Einheit bp (Big Points, $\frac{1}{72}$ Zoll = 0.353 mm, einer im Buchdruck verbreiteten Einheit) interpretiert.

Für d kann eines der Worte Tiny, Small, Medium, Large stehen, Thin und Thick sind Kurzbezeichnungen für die Linienstärken Thickness[Tiny] und Thickness[Large].

Auf das Farben- und Farbgebungssystem für *Mathematica*-Grafiken kommen wir in Kapitel 8 genauer zu sprechen. An dieser Stelle seien nur die folgenden Farbdirektiven zur Festlegung von Farben und Farbintensitäten aufgeführt.

Grafikdirektiven für Farben

Hue[d]	Hue-Farbsystem
GrayLevel[d]	Graustufen
RGBColor[r, g, b]	RGB-Farbsystem

Im RGB-Farbsystem werden Farben durch direkte Angabe der Intensität des Rot-, Grün- und Blauanteils einer Lichtquelle beschrieben, so dass RGBColor[0, 0, 0] schwarzes (Black) und RGBColor[1, 1, 1] weißes (White) Licht ergibt. Das Hue-System entspricht der Interpolation über das Spektrum der Farben Rot, Gelb, Grün, Cyan, Blau, Magenta wieder zurück zu Rot, wenn der Parameter *d* im Intervall [0, 1] variiert, und kann eindeutig ins RGB-System transformiert werden. Details dazu finden Sie in Kapitel 8 auf Seite 225.

Für eine Reihe von Farben gibt es Konstanten als Bezeichner, die zum entsprechenden RGB-Wert auswerten. GrayLevel steht für Grauabstufungen, wobei GrayLevel[a] der Farbintensität RGBColor[a, a, a] entspricht, was zusammen mit anderen Farbquellen dann durchaus zu farbigen Darstellungen führen kann.

Zur dritten Gruppe von Direktiven gehören die Konstrukte EdgeForm und FaceForm, mit denen die Darstellung (Farben, Liniendicke, Durchsichtigkeit usw.) der Kanten und des Flächeninneren der 2D-Primitive Polygon, Disk und Rectangle gesteuert werden kann.

Im folgenden Beispiel werden vier Einheitsquadrate erzeugt, wozu wir dem Kommando Rectangle jeweils die Koordinaten des linken unteren Eckpunkts übergeben. Für drei von ihnen ist mit der Direktive EdgeForm eine spezielle Randgestaltung vereinbart. Die beiden Farbdirektiven Yellow und Red wirken sich auf mehrere der Quadrate aus. Beachten Sie die spezielle Struktur ineinandergeschachtelter Listen, um die Wirkung der Direktiven auf einzelne oder mehrere Primitive zu steuern.

```
g = Graphics[{Yellow, Rectangle[{0, 0}], {EdgeForm[Thick], Rectangle[{1, 1}]},
    Red, {EdgeForm[Dashed], Rectangle[{2, 0}]},
    {EdgeForm[{Thick, Dotted, Green}], Rectangle[{3, 1}]}}]
```

Über die Art und Weise, eine solche Farbgrafik in ein Schwarzweißformat zu überführen, sind sich die *Mathematica*-Entwickler bis zur Fertigstellung der Version 6 leider nicht einig geworden.

Die Option ColorOutput, über welche eine solche Transformation bisher immer gesteuert werden konnte, ist als *veraltet* markiert und weitgehend aus der Dokumentation herausgenommen worden, obwohl sie etwa beim Kommando Show noch präsent ist.

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

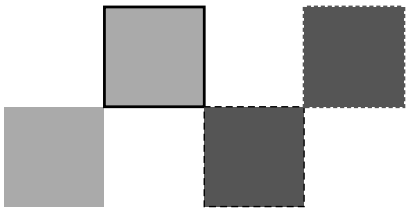
hat aber überhaupt keinen Einfluss auf die Ausgabe.

Unstimmigkeit besteht wohl darüber, ob und ggf. in welchem Umfang über die Option ColorFunction als Nachfolger von ColorOutput (Warnung: mit vollkommen anderer Semantik) global auf das Farbverhalten Einfluss genommen werden soll. Sie

können aber Schwarzweißausgaben immer erzwingen, wenn Sie die RGBColor- und Hue-Grafikdirektiven durch entsprechende GrayLevel-Einträge ersetzen.

Dieses Schwarzweißbild haben wir mit dem Kommando

```
g /. RGBColor[a__] :>
        GrayLevel[Mean[{a}]]
```



erzeugt, welches aus $a = (r, g, b)$ den Mittelwert (Mean) berechnet und die RGBColor-Direktiven durch GrayLevel mit dieser Sättigung ersetzt.

Beachten Sie die Verwendung von :>, denn Mean[{a}] soll ja nicht zur Definitionszeit mit dem Symbol a, sondern zur Anwendungszeit mit der dann unter a gespeicherten Sequenz aufgerufen werden.

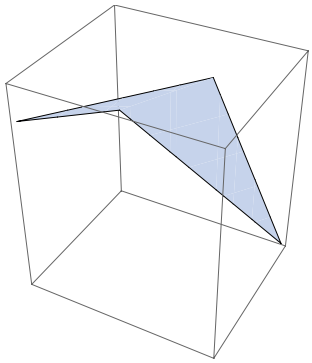
3.2.2 3D-Grafiken zusammenbauen

Während Disk und Rectangle 2D-Grafiken sind, kann Polygon sowohl für 2D- als auch 3D-Grafiken verwendet werden. Unklar ist zunächst, was unter einem 3D-Polygon zu verstehen ist, da selbst vier 3D-Punkte in der Regel nicht in einer Ebene liegen.

Wir wollen deshalb erforschen, was *Mathematica* in diesem Fall ausgibt, und dabei zugleich den komplexen Einsatz von Grafikprimitiven und -direktiven beispielhaft demonstrieren.

Zunächst definieren wir eine Funktion RP, mit welcher zufällige n-dimensionale Punktkoordinaten erzeugt werden können.

```
RP[n_] := Table[Random[ ], {n}]
```



Diese verwenden wir, um eine Liste p von vier 3D-Punktkoordinaten anzulegen, aus dieser ein Polygon-Primitiv und schließlich ein Graphics3D-Objekt zu erzeugen.

```
p = Table[RP[3], {4}];

Graphics3D[Polygon[p]]
```

Wir sehen schon einmal, was ein allgemeines 3D-Polygon mit den Ecken $P_1, P_2, \ldots, P_n$ für *Mathematica* ist – eine aus mehreren Teildreiecken zusammengefügte Fläche. 3D-Polygone bilden das Grundelement, aus welchem allgemeinere Flächen zusammengesetzt werden. Obwohl nicht ausdrücklich so eingeschränkt, werden dabei meist Dreiecke (Triangulierungen) oder – in seltenen Fällen – garantiert ebene Vielecke verwendet.

Unklar ist in unserem Beispiel, warum das Viereck längs der einen und nicht der anderen Diagonalen geknickt ist. Bevor wir dieser Frage im Detail auf den Grund gehen, wollen wir das so konstruierte Polygon drehen und genauer betrachten.

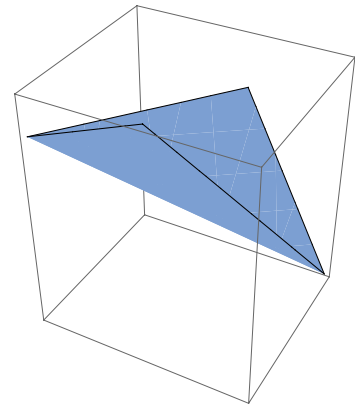
Beim Drehen wechselt es die Farbe, was damit zusammenhängt, dass wir uns nicht für die Farbgebung interessiert, sondern mit den Standardeinstellungen von *Mathematica* zufrieden gegeben haben. Mit der folgenden FaceForm-Direktive bekommt die Polygonfläche eine eigene Farbe. Die EdgeForm-Direktive zeichnet die Kanten des Polygons besonders dick. Die Sammlung aus Direktiven und Primitiven muss dem Graphics3D-Objekt als Liste im ersten (und hier einzigen) Parameter übergeben werden.

```
Graphics3D[{EdgeForm[Thick], FaceForm[LightBlue], Polygon[p]}]
```

Drehen und betrachten Sie dieses Objekt, so sehen Sie, dass der Knick keine Kante des Polygons ist. Die Farbe erscheint zwar schon homogener, wechselt aber noch immer beim Drehen. Das hängt damit zusammen, dass in *Mathematica* alle Grafiken standardmäßig mit vier verschiedenfarbigen Lichtquellen beleuchtet werden und sich so die Eigenfarbe und die Beleuchtungsfarben mischen. Die reine Eigenfarbe kommt nur bei weißem Licht zur Geltung, was mit der Option Lighting → "Neutral" oder Lighting → {White} eingestellt werden kann. Lighting → "Neutral" schaltet die Lichtquellen der Standardbeleuchtung auf weißes Licht, was noch gewisse Lichtaber keine Farbreflexe erhält. Lighting → {White} bewirkt eine uniforme ambiente Beleuchtung mit weißem Licht aus allen Richtungen, so dass wirklich nur noch die Eigenfarben zur Geltung kommen. Doch kehren wir zur Frage zurück, warum das Viereck gerade an dieser Diagonalen geknickt ist.

Dies hat sicher mit dem Startpunkt in der Liste *p* zu tun, was wir als Nächstes überprüfen wollen.

Dazu erzeugen wir mit RotateLeft die Liste *q*, in welcher die Punkte der Liste *p* um einen Platz nach links verschoben sind und der erste Punkt ans Ende gerutscht ist, und erzeugen daraus ein Grafikobjekt.



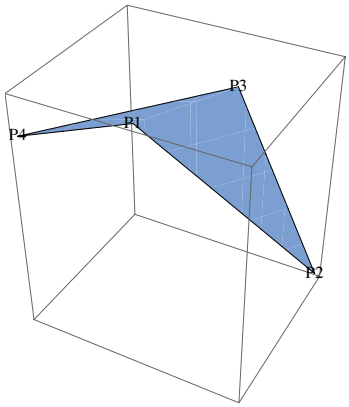
```
q = RotateLeft[p];

Graphics3D[
  {FaceForm[LightBlue], Polygon[q]},
  Lighting → {White}]
```

In der Tat ist die Fläche nun an der anderen Diagonalen geknickt. Den genauen Zusammenhang können wir bisher nicht studieren, da die Punkte bisher nicht benannt sind. Wir wollen deshalb die Grafik um Textprimitive mit den entsprechenden Bezeichnungen ergänzen. Wir definieren dazu eine Liste txt mit den Beschriftungen sowie eine Liste Beschriftung mit geeigneten Textprimitiven, so dass der folgende Funktionsaufruf, den wir mit := in symbolischer Form ohne Auswertung in der Variablen zeigeDasBild für spätere Aufrufe ablegen, die korrekt beschriftete Grafik erzeugt.

```
txt = {P1, P2, P3, P4};
zeigeDasBild :=
 Graphics3D[{Beschriftung, {FaceForm[LightBlue], Polygon[p]}}, Lighting → {White}];
```

Eine erste Definition könnte lauten

```
Beschriftung =
    Table[Text[txt[[i]], p[[i]]], {i, 1, 4}];
zeigeDasBild
```



Sie hat den Nachteil, dass der Text an den jeweiligen Eckpunkten ohne Abstand platziert ist.

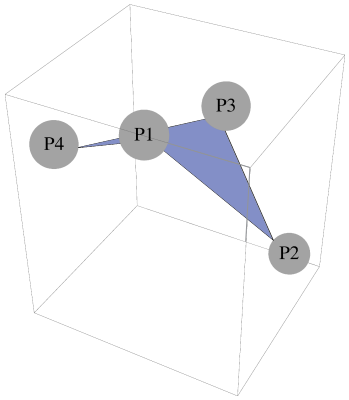
Um einen solchen Abstand herzustellen, müssen die Beschriftungen leicht umgesetzt werden, aber jede in eine andere Richtung. Das kann durch eine Verschiebung um einen festen Prozentsatz vom Schwerpunkt M des Vierecks weg erreicht werden.

Die Koordinaten von M können mit dem Kommando Mean berechnet werden. Punkte auf der Verbindungsgeraden MP haben die Koordinaten $\alpha M+(1-\alpha) P$, wobei der Wert $\alpha = -0.1$ eine günstige Wahl ist.

```
M = Mean[p];
Beschriftung = Table[
    Text[txt[[i]], 1.1 p[[i]] − 0.1 M, {i, 1, 4}];
zeigeDasBild
```

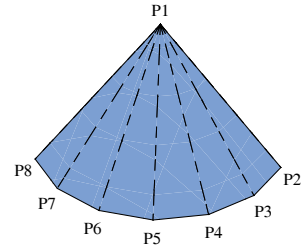
Zur Krönung des Ganzen packen wir jede Beschriftung noch in eine kleine orange Kreisscheibe, die mit dem Sphere-Primitiv erzeugt wird.

```
Beschriftung = Table[
  With[ {pos = 1.2 p[[i]] − 0.2 M},
    {Text[txt[[i]], pos],
      {Orange, Sphere[pos, 0.06]}}],
  {i, 1, 4} ];
zeigeDasBild
```



Dieses doch nicht mehr ganz so einfache Beispiel einer Beschriftung zeigt, auf welche Weise Sie grafische Effekte mit durchdacht geschachtelten Listen erzielten können, die aus Primitiven und Direktiven aufgebaut sind.

Natürlich sind wir noch ein Bild schuldig, welches die Ausgangsfrage beantwortet. Hier ist es. Wir haben einige Hilfslinien eingezeichnet, um zu verdeutlichen, dass die Polygonfläche Polygon[{P1, ..., Pn}] genau aus den Dreiecken $P_1P_iP_{i+1}$, $i = 2, \ldots, n-1$ zusammengesetzt wird.



Und hier folgt der Quelltext, mit dem dieses Bild erzeugt worden ist, ohne weiteren Kommentar:

```
p = Prepend[Table[{Sin[i], Cos[i], 0}, {i, 1.5, 4, .4}], {0, 0, 1}];
n = Length[p];
txt = Table["P" <> ToString[i], {i, 1, n}];
lines = Table[Line[{p[[1]], p[[i]]}], {i, 2, n}];
Beschriftung = Table[Text[txt[[i]], If[i > 1, 1.1p[[i]]−0.1p[[1]], {0, 0, 1.1}]], {i, 1, n}];
Graphics3D[{Beschriftung, {Dashing[.03], lines}, {FaceForm[Yellow], Polygon[p]}},
    Lighting → {White}, PlotRange → All, Boxed → False]
```

3.2.3 Das GraphicsComplex-Primitiv

Mit Version 6 stehen auch Gruppierungsprimitive zur Verfügung, von denen wir im Folgenden das GraphicsComplex-Primitiv vorstellen wollen.

Oft bestehen Grafiken oder Teile von Grafiken wie etwa Gitter oder Triangulierungen von Flächen aus einer größeren Zahl einfacher Primitive mit einer gemeinsamen Menge von Punktkoordinaten. Diese lassen sich kompakt als

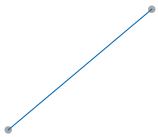
GraphicsComplex[{P1, ..., Pn}, data]

anschreiben. Das erste Argument enthält dabei eine Liste der Koordinaten der gemeinsamen Punkte, das zweite Argument eine geschachtelte Liste von Grafikprimitiven und -direktiven, in denen die Koordinaten von P_i durch i abgekürzt sind. $\{P_1, \ldots, P_n\}$ ist also eine Art Symboltabelle für die Interpretation der Primitive in data. Neben der Platzersparnis erlaubt ein solches Prinzip der eindeutigen Referenzierbarkeit der Punktkoordinaten auch, solche Gruppierungsprimitive auf einfache Weise zu transformieren, da nur die Punktkoordinaten transformiert werden müssen.

Im folgenden Beispiel besteht data aus zwei Punkten und der sie verbindenden Strecke. Die Punkte sind zusätzlich mit der Farbdirektive Red versehen, so dass sich die Farbdirektive Blue im Graphics-Objekt nur auf die Farbe der Strecke auswirkt.

```
gc = GraphicsComplex[{{1.2, 3.5}, {5.6, 7.2}}, {{Red, Point[1], Point[2]}Line[1, 2], }];
    Graphics[{Blue, PointSize[.05], Thick, gc}]
```

Mit Normal lässt sich ein solches GraphicsComplex-Primitiv in eine geschachtelte Liste von einfachen Primitiven und Direktiven expandieren, falls das erforderlich sein sollte.



GraphicsComplex-Primitive finden Sie in vielen Grafikobjekten, die *Mathematica* selbst zur Verfügung stellt. Wir wollen den Umgang mit diesem Primitiv am Beispiel des Studiums der Dualität der platonischen Körper demonstrieren.

Bekanntlich gibt es mit Tetraeder, Würfel, Oktaeder, Dodekaeder und Ikosaeder genau fünf verschiedene dreidimensionale reguläre Körper. Verbindet man die sechs Seitenmitten eines Würfels, so erhält man ein Oktaeder und umgekehrt. Beide Körper sind deshalb dual zueinander und wir wollen schrittweise eine Grafik erzeugen, in der dieser Sachverhalt visualisiert ist.

Als Rohmaterial können wir auf die Datenbank PolyhedronData[] zurückgreifen, in der Informationen über verschiedene reguläre und halbreguläre Körper gesammelt sind.

```
platonischeKoerper =
    PolyhedronData["Platonic"]
```

```
{"Cube", "Dodecahedron",
"Icosahedron", "Octahedron",
"Tetrahedron"}
```

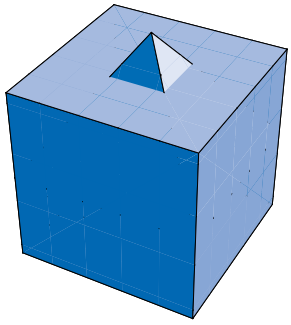
Auf einfache Weise kann dort die Liste der Schlüsselworte der etwa zu einem Würfel gespeicherten Informationen oder der Wert einer konkreten Eigenschaft eingesehen werden. Alle Werte beziehen sich auf Körper mit der Kantenlänge 1.

```
PolyhedronData["Cube", "Properties"]
```

```
{"AlternateNames", "Circumradius",
..., "Volume", "WythoffSymbol"}
```

Grafikobjekte oder -primitive der jeweiligen Körper sind unter den Schlüsselworten Edges (Kantenmodell-Primitiv), Faces (Flächenmodell-Primitiv) und Images (der Körper als Grafikobjekt) gespeichert.

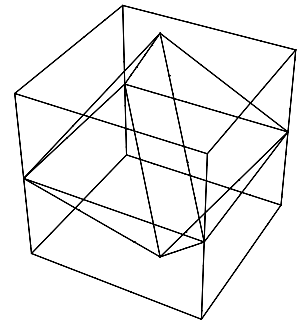
Wir verwenden als Ausgangspunkt unserer Experimente die Faces-Variante. Sie gibt je ein GraphicsComplex-Primitiv zurück. Diese müssen nun noch durch entsprechende Drehungen und Streckungen in eine passgerechte Lage transformiert werden.



```
cube = PolyhedronData[
                "Cube", "Faces"];
octahedron = PolyhedronData[
                "Octahedron", "Faces"];

Graphics3D [{cube, octahedron}]
```

Das Oktaeder ragt aus dem Würfel heraus, da wir es noch nicht skaliert haben. Komischerweise ragt es nur aus zwei Seitenflächen heraus. Mit

```
Graphics3D[
  {FaceForm[ ], cube, octahedron} ]
```



können wir die Seiten durchsichtig machen und erkennen am so entstehenden Drahtgittermodell, dass das Oktaeder auch noch „falsch herum“ im Würfel liegt, also um 45° um die *z*-Achse gedreht werden muss.

GraphicsComplex-Primitve lassen sich – wie gesagt – leicht geometrisch transformieren; wir müssen nur die Koordinaten der Punkte im ersten Argument mit der entsprechenden Transformationsmatrix multiplizieren und daraus das neue Primitiv nach denselben Anweisungen aufbauen. Hier ist schon mal eine Funktionsdefinition, die das erledigt.

```
gcTransform[r_, g_GraphicsComplex] := GraphicsComplex[(r . # &) /@ g[[1]], g[[2]]]
```

Sie wendet die namenlose Funktion (r . # &) (Multiplikation mit der Transformationsmatrix *r*) auf alle Elemente der Liste *g*[[1]], also die Punktkoordinaten, an, die dabei als 3D-Vektoren interpretiert werden.

Die erforderliche Transformation des Oktaeders können wir aus zwei Schritten zusammensetzen. *r* ist die Matrix, mit welcher eine Drehung in *z*-Richtung um 45° bewirkt wird. Nach einer Drehung des Oktaeders schaut dieses an allen Seiten auf gleiche Weise aus dem Würfel heraus und muss nur noch skaliert werden, so dass Inkugelradius des Würfels und Umkugelradius des Oktaeders übereinstimmen.

```
r = RotationMatrix[π/4, {0, 0, 1}];
r // MatrixForm
```

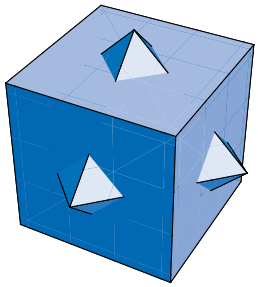
$$\begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
p = Graphics3D[
  {cube, gcTransform[r, octahedron]} ]
```

Ohne weitere Anpassungen ist rund um die (hier nicht abgedruckte) Grafik noch viel Platz. Das hat damit zu tun, dass die Mitten der Seitenflächen der 3D-Begrenzungsbox gerade die Oktaederecken sind, diese Box also um den Faktor $\sqrt{2}$ größer ist als der Würfel. Diese Begrenzungsbox wird ihrerseits standardmäßig vollständig ins 2D-Bildfenster eingepasst.

Mit der Option ViewAngle können Sie den dargestellten Ausschnitt beeinflussen.

```
Show[p, ViewAngle → 20°]
```



Der Skalierungsfaktor berechnet sich als Quotient aus Umkugelradius des Oktaeders und Inkugelradius des Würfels. Die erforderlichen Daten können wir wieder der Datenbank entnehmen.

```
radii = {#, PolyhedronData[#, "Inradius"], PolyhedronData[#, "Circumradius"]} &
                                                        /@ platonischeKoerper;
```

Sie sind für alle fünf platonischen Körper in der folgenden Tabelle zusammengestellt. Prepend ergänzt die Tabelle radii der Daten zunächst um eine Überschriftenzeile. Zur Formatierung haben wir das Kommado Grid mit den Optionen Dividers und Spacings verwendet, das bereits einmal auf Seite 56 zum Einsatz kam. Text@... verwandelt die ganze Tabelle in ein Objekt vom Typ Text, welches in der StandardForm mit einem gefälligeren Zeichensatz dargestellt wird.

```
Text@Grid[Prepend[radii, {"Körper", "Inkugelradius", "Umkugelradius"}],
    Dividers → {{{True}}, {True, True, {False}, True}}, Spacings → {1, 2}]
```

Körper	Inkugelradius	Umkugelradius
Cube	$\frac{1}{2}$	$\frac{1}{2}\sqrt{3}$
Dodecahedron	$\frac{1}{20}\sqrt{250+110\sqrt{5}}$	$\frac{1}{4}\sqrt{\sqrt{3}+\sqrt{15}}$
Icosahedron	$\frac{1}{12}\left(3\sqrt{3}+\sqrt{15}\right)$	$\frac{1}{4}\sqrt{\sqrt{10}+2\sqrt{5}}$
Octahedron	$\frac{1}{\sqrt{6}}$	$\frac{1}{\sqrt{2}}$
Tetrahedron	$\frac{1}{2\sqrt{6}}$	$\sqrt{\frac{3}{8}}$

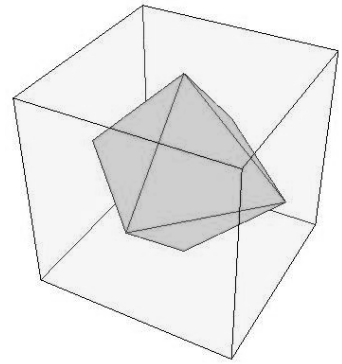
Mit dem folgenden Kommando können Sie also die gewünschte Grafik erzeugen.

```
Graphics3D[
    {{Yellow, Opacity[.3], cube}, {Green, gcTransform[r/√2, octahedron]}},
    PlotRange → All, Boxed → False, Lighting → {White}]
```

Ein paar Worte zur Erläuterung: Damit das Oktaeder auf die beschriebene Weise in den Würfel eingepasst wird, muss der Inkugelradius des Würfels mit dem Umkugelradius des Oktaeders übereinstimmen, Letzteres folglich um den Faktor $\sqrt{2}$ verkleinert werden. Das wird durch skalare Multiplikation der Matrix *r* mit dem Faktor $\frac{1}{\sqrt{2}}$ erreicht.

Die Szene ist mit weißem Licht ausgeleuchtet, um die Eigenfarben der beiden Körper unverfälscht zur Geltung zu bringen. Der Würfel ist außerdem leicht durchsichtig (Opacity-Direktive), um auch einen Blick ins Innere zu ermöglichen.

Die ausnahmsweise miserable Qualität des Abdrucks dieses Bilds hängt damit zusammen, dass wir als Vorlage eine `jpg`-Rastergrafik verwenden mussten und nicht wie sonst für die Abbildungen in diesem Buch den ansonsten sehr brauchbaren `eps`-Export verwenden konnten. Leider berücksichtigt *Mathematica* beim `eps`-Export die Opacity-Effekte nicht.



Dieselben Effekte können mit Transformationsfunktionen erzielt werden, die *Mathematica* für geometrische Transformationen zur Verfügung stellt. Dabei handelt es sich um Funktionsobjekte mit dem Kopfsymbol TransformationFunction, die auf geometrische Primitive und Objekte angewendet werden können. In unserem Fall ist rt die entsprechende Dreh-, st die Skalierungstransformation und ct deren Komposition.

```
rt = RotationTransform[π/4, {0, 0, 1}];
st = ScalingTransform[Table[1/√2, {3}]];
ct = Composition[st, rt]
```

```
TransformationFunction[ 1/2  -1/2   0    0 ]
                       [ 1/2   1/2   0    0 ]
                       [  0     0   1/√2  0 ]
                       [  0     0    0    1 ]
```

Sie sehen an der Ausgabe, dass Transformationsfunktionen intern durch eine vierreihige Transformationsmatrix auf einheitliche Weise dargestellt werden, die über einen Aufruf von GeometricTransformation ähnlich wie die von uns konstruierte Funktion gcTransform auf geometrische Objekte angewendet werden können. Der Unterschied besteht einzig darin, dass wir direkt mit dreireihigen Matrizen arbeiten und unser Ansatz nur für GraphicsComplex-Objekte funktioniert, während die *Mathematica*-Entwickler natürlich etwas weiter gedacht haben.

```
transformedOctahedron = GeometricTransformation[octahedron, ct]
Graphics3D[{ {Yellow, Opacity[.3], cube}, {Green, transformedOctahedron} }, ...]
```

transformedOctahedron ist dabei ein (literales) Grafikobjekt, welches durch Einbettung in Graphics3D zu einem grafisch darstellbaren Graphics3D-Objekt wird. Es kann auf die angegebene Weise in unserer Darstellung der Lagebeziehung dualer Polyeder verwendet werden.

3.3 Interaktive Grafiken und Animationen

Mit der Version 6 wurden die interaktiven Möglichkeiten von *Mathematica* einer grundlegenden Revision und Neukonzipierung unterworfen. Da die umfassende Nutzung dieser neuen Möglichkeiten weit über den Rahmen eines einführenden Buchs wie dieses hinausgeht, werden wir Ihnen an dieser Stelle nur die Kommandos Animate und Manipulate jeweils in ihrer Grundversion vorstellen. Da sich in einem Buch dynamische Szenen nicht sinnvoll darstellen lassen, haben wir auf den Abdruck von Ausgaben weitgehend verzichtet und verweisen auf das Notebook zu diesem Kapitel.

Beiden Kommandos liegt das Konzept einer *Szene* zu Grunde, welche wie im Film aus einer Folge von Momentaufnahmen (Rahmen, frames) eines sich dynamisch entwickelnden Sachverhalts besteht. Die Syntax beider Kommandos ist deshalb mit der des Table-Kommandos vergleichbar.

Animate[expr, {a, a_s, a_e}] und Manipulate[expr, {a, a_s, a_e}]

a ist dabei die unabhängige Zeit- oder Schrittvariable, über welche der Ablauf der Szene gesteuert wird, a_s und a_e bilden deren Start- und Endwert. Wie im Table-Kommando sind vierstellige Iteratoren möglich, wenn auch noch die Schrittweite angegeben werden soll. expr ist ein von a abhängender Ausdruck, der die Dynamik der Szene beschreibt.

Eine solche Szene ist meist ein zwei- oder dreidimensionales bewegtes Bild, kann aber auch einen Satz von Formeln umfassen.

So werden in der Hilfeseite zum Kommando Manipulate die Faktorzerlegungen des Polynoms x^n-1 für verschiedene n in einer solchen Szene zusammengefasst.

```
Manipulate[Factor[x^n − 1], {n, 2, 10, 1}]
```

Während Animate nur das automatische Abspielen einer solchen Szene ermöglicht, stellt Manipulate einen komplexeren Zusammenhang zwischen einem (oder mehreren) Entwicklungsparameter(n) und dem sich entwickelnden Sachverhalt her. So kann beim gerade betrachtenen Beispiel die Zahl n über einen Schieberegler eingestellt werden. Alternativ lässt sich über Knopfdruck ein Kontrollpanel öffnen. In einer Dialogbox finden sich dort Knöpfe, über welche die Szene automatisch oder im Schrittmodus abgespielt werden kann.

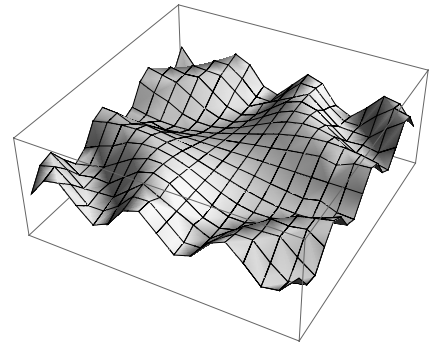
Dieses Kommando erzeugt eine Animation der Sin-Funktion im Intervall $[0, 10]$, die um verschiedene Phasenwinkel a verschoben ist.

```
Animate[
  Plot[Sin[x + a], {x, 0, 10}], {a, 0, 5}]
```

Die Knöpfe zur Steuerung einer Animation sind eine Teilmenge der Knöpfe im Manipulate-Kontrollpanel. Eine schrittweise Steuerung einer Animation ist nicht möglich. Auch lässt sich das Kontrollpanel nicht ausblenden. Wir konzentrieren die weiteren Ausführungen deshalb auf das Manipulate-Kommando.

Bei der Erstellung von Animationen sollten Sie den erforderlichen Rechenaufwand für das Erstellen der einzelnen Rahmen einer Szene berücksichtigen. Die Animation kann nicht schneller sein, als *Mathematica* diese berechnen kann.

Für dieses Plot3D-Objekt, welches uns als Ausgangspunkt der folgenden Animation dient, wurde deshalb mit der Option PlotPoints die Auflösung auf einen sehr kleinen Wert reduziert. Zur Veranschaulichung ist das Ausgangsbild hier wiedergegeben.



```
Plot3D[Sin[xy], {x, -π, π}, {y, -π, π},
    PlotPoints → 5, PlotRange → 3,
    Axes → None]
```

Wir erzeugen nun eine Szene, in welcher sich diese 3D-Grafik dreht. Allerdings bewegen wir dazu nicht die Grafik, sondern den Betrachter, indem wir über die Option ViewPoint eine Reihe von Betrachterstandpunkten auf einem Kreis mit dem Radius 3 in konstanter Höhe $z = 2$ um den Mittelpunkt $(0, 0, 2)$ „abfliegen".

Ein Problem bei dieser Animation ergibt sich aus dem Umstand, dass *Mathematica* für jedes einzelne Frame eine optimale Skalierung der 3D-Begrenzungsbox berechnet. Mit der Option PlotRange kann die Größe dieser Box explizit gesetzt und damit diese Skalierung verhindert werden. Wir haben das in obiger Grafik bereits berücksichtigt. Damit lassen sich leichte Größenänderungen zwischen verschiedenen Rahmen trotzdem nicht vermeiden, da diese 3D-Box konstanter Größe ja ihrerseits in das 2D-Bildfenster eingepasst wird und dabei aus unterschiedlichen Blickwinkeln unterschiedliche Größenverhältnisse (von Kantenlänge bis Flächendiagonale) wirksam sind. Das lässt sich hier durch einen festen Blickwinkel ViewAngle beheben, da wir uns die ganze Zeit in einer festen Entfernung von unserem Objekt befinden. Das folgende Kommando erzeugt die gewünschte Animation.

```
Manipulate[Plot3D[Sin[xy], {x, -π, π}, {y, -π, π}, PlotPoints → 5, PlotRange → 3,
    ViewPoint → {3 Sin[α], 3 Cos[α], 2}, ViewAngle → 25°, Axes → None],
  {α, 0., 2 π, π/12}]
```

Das Ergebnis der Animation ist ganz überzeugend, allein mit den Gitterlinien kommt *Mathematica* etwas durcheinander und „vergisst" sie während der Animation. Der Grund hierfür ist derselbe, wie auf Seite 231 für Bewegungen von Grafiken mit der Maus genauer erläutert wird: Um eine ausreichende Performanz zu erreichen, muss der Rechenaufwand während des Laufens der Animation reduziert und für Rechnungen möglichst auf Cachedaten zurückgegriffen werden. Deshalb wird in dieser Phase nur ein reduzierter Satz von Daten neu berechnet. Erst wenn Sie die Animation anhalten, wird wieder ein voller Satz Daten berechnet und das Ergebnis (mit einiger zeitlicher Verzögerung) im Detail neu gezeichnet. Dann werden auch die Gitterlinien wieder auf die Grafik aufgebracht.

Falls die Berechnung der Daten der einzelnen Rahmen für eine Echtzeitanimation zu aufwändig ist, können Sie die einzelnen Rahmen auch vorab berechnen, in einer Liste speichern und mit Manipulate oder ListAnimate danach abspielen. Der zweite Parameter von ListAnimate gibt die Abspielgeschwindigkeit in Frames pro Sekunde vor.

```
data = Table[Plot3D[Sin[x y], {x, -π, π}, {y, -π, π}, PlotPoints → 5, PlotRange → 3,
              ViewPoint → {3 Sin[α], 3 Cos[α], 2}, ViewAngle → 25°, Axes → None],
             {α, 0., 2 π, π/12}]
```

```
ListAnimate[data, 1]   oder   Manipulate[data[[i]], {i, 1, Length[data], 1}]
```

3.4 Export und Import

Mathematica-Grafiken lassen sich bequem auf bewährte und gegenüber älteren Versionen erweiterte Weise mit verschiedenen anderen Programmen austauschen. Dies ist sowohl (je nach Betriebssystem) über die Zwischenablage als auch über die Menüeinträge FILE|SAVE AS oder FILE|SAVE SELECTION AS möglich. Wir wollen hier nur kurz das Abspeichern und Importieren von Grafiken über die Befehle Export und Import besprechen.

Die Syntax der beiden Kommandos lautet

```
Export[dateiname, expr]   oder   Export[dateiname, expr, typ]
```

und

```
Import[dateiname]   oder   Import[dateiname, typ],
```

wobei das Importkommando im Erfolgsfall sein Ergebnis als *Mathematica*-Ausdruck zurückgibt. Das Abspeichern und Importieren von allgemeineren Daten funktioniert nach dem gleichen Prinzip.

Die unterstützten Austauschgrafikformate sind in den Systemvariablen $Export-Formats und $ImportFormats als Listen gespeichert. Die meisten Formate werden an der entsprechenden Dateiendung automatisch erkannt; sie können aber auch durch einen optionalen Parameter typ zur Ausgabe in einem fest vorgegebenen Format veranlasst werden.

Da *Mathematica*-Grafiken im Notebook in einem eigenen Grafikformat vorgehalten werden, in dem – besonders für 3D-Grafiken – auch die dynamischen Aspekte einer solchen Grafik beschrieben sind, ist ein Informationsverlust beim Abspeichern als statisches Bild prinzipiell nicht zu vermeiden. Beste Ergebnisse werden beim Abspeichern im `eps`-Vektorgrafikformat erreicht, da das *Mathematica*-interne Format ebenfalls ein Vektorgrafikformat ist.

Die klassischen Grafikformate wie `jpg`, `tiff`, `bmp`, `png` oder `gif` sind aber Rastergrafikformate, so dass entsprechende Grafiken auch nur noch als Raster reimportiert werden können. Eine akzeptable Bildqualität solcher Grafiken wird nur bei ausreichender Auflösung erreicht, was sich natürlich auf die Dateigröße auswirkt. Die Auflösung kann für Rastergrafiken über die Option ImageResolution des Export-Kommandos gesteuert werden.

3D-Grafiken lassen sich in einer großen Zahl gängiger 3D-Formate abspeichern und aus diesen wieder einlesen, da das auf GraphicsComplex-Primitiven basierende *Mathematica*-3D-Format mit Punkt- und Liniendaten, Triangulierungen und Vertex-Normalen alle wichtigen Sprachelemente der einschlägigen Formate abbilden kann. Dies wird in unserem Buch allerdings keine Rolle spielen, da die Feinheiten tiefere Kenntnisse der entsprechenden Grundlagen erfordern, die wir nicht voraussetzen

wollen. Im Hilfesystem finden Sie weitere detaillierte Informationen zu den einzelnen Formaten.

Auch verschiedene speziellere domänenspezifische Formate aus der Medizin, Biologie, Physik und Astronomie werden unterstützt. Animationen können in entsprechenden Formaten wie `avi`, animated `gif` oder `flv` (Adobe/Macromedia Flash) exportiert werden.

Beim Import von Grafiken hängt das Ergebnis wesentlich vom Grafikformat der Vorlage ab. Im `jpg`-, `png`- oder `gif`-Format vorliegende Rastergrafiken werden als Grafikobjekte auf der Basis des **Raster**-Primitivs eingelesen. Auf Seite 227 diskutieren wir die interne Präsentation einer so importierten Grafik an einem Beispiel.

Mathematica bietet in der Version 6 eine deutlich erweiterte Palette möglicher Ausgabeformate; allerdings bedarf es noch einiger Arbeit, bis wirklich alle Ecken und Kanten der entsprechenden Transformationsprozeduren ausgebügelt sind. Bei der Erstellung dieses Buchs spielte vor allem der `eps`-Export von Grafiken eine wichtige Rolle. Dieser Bereich wurde von den *Mathematica*-Entwicklern in der Endphase des Beta-Tests noch einmal gründlich durchgesehen, was erheblich positiven Einfluss auf Dateigrößen und Bildqualität hatte. Probleme mit abgeschnittenen Zeichen, ungenauen Begrenzungsboxen oder der adäquaten Berücksichtigung aller Grafikdirektiven sind aber auch in der von Wolfram Research ausgelieferten offiziellen Version vom Mai 2007 genügend enthalten, so dass *Mathematica* auf diesem Gebiet wohl trotz der erreichten beachtlichen Breite der Exportmöglichkeiten als „work in progress“ zu betrachten ist.

Dass ein komplexes Softwaresystem gar nicht anders entwickelt werden kann als auf diese Weise – durch Auslieferung gut konsolidierter „Schnappschüsse“ des Entwicklungsstands der Software –, dieses Problem teilt *Mathematica* mit anderen Systemen dieser Größenordnung. Jede Releasefreigabe erfordert einen angemessenen Kompromiss zwischen dem Wunsch der Nutzer, über neue und erweiterte Funktionalitäten möglichst rasch verfügen zu können, und dem Anspruch auf Seriosität der Entwickler, mit heißer Nadel gestrickte Lösungen zu vermeiden. Dass außerdem noch betriebswirtschaftliche, marketingstrategische und viele andere weniger funktionale Aspekte in solche Entscheidungen hineinregieren, sei nur der Vollständigkeit halber erwähnt.

Sie können also davon ausgehen, dass auch mit der Version 6 noch nicht das „Ende der Geschichte“ erreicht ist, nicht einmal das Ende der Erfolgsgeschichte von *Mathematica*, so dass es einer späteren Auflage dieses Buchs vorbehalten bleiben mag, über die dann hoffentlich weiter verbesserte Qualität dieser Exportfunktionalitäten ausgewogen zu urteilen.

3.5 Syntaxzusammenfassung

Plot-Kommandos

ContourPlot[f[x, y], {x, x_s, x_e}, {y, y_s, y_e}, Optionen]

2D-Darstellung der Funktionswerte von *f* durch Höhenlinien.

ContourPlot[f[x, y] == c, {x, x_s, x_e}, {y, y_s, y_e}, Optionen] oder
ContourPlot3D[f[x, y, z] == c, {x, x_s, x_e}, {y, y_s, y_e}, {z, z_s, z_e}, Optionen]

Darstellung eines implizit gegebenen funktionalen Zusammenhangs.

DensityPlot[f[x, y], {x, x_s, x_e}, {y, y_s, y_e}, Optionen]

2D-Darstellung der Funktionswerte von *f* durch Farben oder Grauwerte.

ParametricPlot[{x[u], y[u]}, {u, u_a, u_e}, Optionen] oder
ParametricPlot3D[{x[u, v], y[u, v], z[u, v]}, {u, u_a, u_e}, {v, v_a, v_e}, Optionen]

Darstellung einer in Parameterform gegebenen Kurve oder Fläche.

Plot[f[x], {x, x_a, x_e}, Optionen] oder
Plot3D[f[x, y], {x, x_a, x_e}, {y, y_a, y_e}, Optionen]

Funktionsgraphen der ein- bzw. zweistelligen Funktion *f* im angegebenen Bereich mit den angegebenen Optionen erstellen.

Für die Anzeige der Grafik darf das Kommando nicht mit einem Strichpunkt abgeschlossen werden!

Plot-Optionen

AspectRatio

Seitenverhältnisse der 2D-Begrenzungsbox einstellen.

Axes

Achsen darstellen.

Boxed

3D-Begrenzungsbox um 3D-Grafik darstellen.

Frame

Rahmen um 2D-Grafik darstellen. In dem Fall werden die Koordinatenticks auf diesem Rahmen angebracht.

Lighting

Lichtverhältnisse für eine 3D-Grafik einstellen.

Lighting → "Neutral"

Standardlichtquellen einer 3D-Grafik auf weißes Licht schalten. Die Farbgebung wird durch die Eigenfarben sowie Helligkeitsreflexe bestimmt.

Lighting → {White}

Beleuchtung einer 3D-Grafik durch ambientes weißes Licht. Die Farbgebung wird nur durch die Eigenfarben bestimmt.

Mesh → All

Die von *Mathematica* verwendete Triangulierung als Gitternetz auf einer 3D-Fläche anzeigen.

PlotRange

Größe der 2D- oder 3D-Begrenzungsbox explizit festlegen.

ViewAngle

Größe des Blickwinkels auf eine 3D-Szene, gemessen als Winkel zwischen Mittelachse und Seitenlinie des entsprechenden Kegels. Standard: so groß, dass die gesamte 3D-Begrenzungsbox zu sehen ist.

ViewPoint

Betrachterstandpunkt festlegen.

Export und Import

Export[dateiname, expr] oder Export[dateiname, expr, typ]

Wert des Ausdrucks expr – etwa eine Grafik – in der angegebenen Datei und dem angegebenen Ausgabeformat speichern. Ist das Ausgabeformat nicht explizit angegeben, so wird dieses aus der Dateiendung deduziert.

Import[dateiname] oder Import[dateiname, typ]

Daten – etwa eine Grafik – aus der angegebenen Datei einlesen. Ist das Format der Eingabedaten nicht explizit angegeben, so wird dieses aus der Dateiendung deduziert.