Solid Modeling – Die Geometrie der Polyeder

von Wolfgang K. Seiler

Als fünftes Ausbildungsfach für den künftigen Philosophenkönig sah Platon die Stereometrie vor, obwohl weder Sokrates noch Glaukon viel dazu sagen konnten. Entsprechend sehen auch die Bildungspläne für die Gymnasien in Baden-Württemberg in mehreren Jahrgangsstufen vor, daß man sich mit dreidimensionaler Geometrie beschäftigt, und trotzdem kann man bei vielen Abiturienten und leider sogar Absolventen mathematischer Studiengänge nicht unbedingt voraussetzen, daß sie wissen, was ein Tetraeder ist – von einem Dodekaeder oder Ikosaeder ganz zu schweigen. Das Gebiet wurde und wird in der Mathematikausbildung extrem vernachlässigt.

An fehlenden Anwendungen kann es nicht liegen: Hier erobert sich die dreidimensionale Geometrie ständig neue Gebiete, darunter auch solche, an die bislang niemand gedacht hatte: Die Neuentwicklung eines Autos oder gar Flugzeugs ist nur deshalb in akzeptabler Zeit möglich, weil zunächst dreidimensionale Modelle im Computer entworfen und dort einer Vielzahl von Simulationen ausgesetzt werden, bevor das erste Modell eines Prototyps physikalisch getestet wird. Auch Werkstücke und künstliche Hüftgelenke entstehen heute zunächst als dreidimensionale Geometrie im Computer – in einigen Zahnarztpraxen sogar der anhand photographischer Aufnahmen digitalisierte und vom Zahnarzt im Computer nachbearbeitete Zahnersatz, der dann gleich, während der Patient wartet, computergesteuert geschliffen und gefertigt wird. Selbst in der chemischen Forschung werden neue Moleküle zunehmend zunächst als dreidimensionale geometrische Modelle im Computer entworfen und vorgetestet, bevor sie im Labor synthetisiert werden.

Im Film sind Computeranimationen erheblich älter älter als die ersten Oskars dafür: Schon aus Kostengründen arbeitet man dort nicht nur für *special effects* mit virtueller Realität: Bekannt wurde beispielsweise der Einsatz fraktaler Geometrie in der Folge „Der Zorn des Khan“ der Serie „Raumschiff Enterprise“. Auch sonst spielt die dreidimensionale Geometrie in den bildenden Künsten eine zunehmende Rolle: An der James Madison University etwa gibt es bereits einen Curriculum, das Kunststudenten Entwurfstechniken mit Polyedern und Splines vermittelt, insbesondere auch für die künstlerische Darstellung von Personen (*siehe z.B.* [**Ra**]); in Italien wurde gerade der erste Wettbewerb „Miss Digital World“ ausgeschrieben (www.missdigitalworld.com); für Designer und Architekten, insbesondere solchen, die städtebauliche oder sonstige Akzente setzen wollen, sind photorealistische Modelle als Hilfsmittel sowohl bei der Planung als auch beim Versuch, den Auftraggeber für einen bestimmten Entwurf zu gewinnen, unabdingbar.

Wenn die dreidimensionale Geometrie trotzdem eher unbeliebt ist in der Mathematikausbildung, liegt dies sicherlich zu einem großen Teil am Problem, daß Zeichnungen im Dreidimensionalen nicht mehr möglich sind und die Darstellung einer zweidimensionalen Projektion, insbesondere wenn sie frei Hand und einigermaßen schnell an der Tafel durchgeführt werden muß, die Fähigkeiten der meisten von uns Mathematikern übersteigt. Auch gibt selbst eine technisch perfekte Zeichnung nur ein unvollkommenes Bild eines dreidimensionalen Gegenstands: Zwar bleibt die Erkenntnis, Geometrie sei die Kunst, aus falschen Zeichnungen richtige Schlüsse zu ziehen, auch im Computerzeitalter richtig; sie hilft aber weder dem Schüler oder Studenten noch dem Lehrer dabei, sich dreidimensionale Geometrie plastisch vorzustellen.

Für die Überwindung dieser Schwierigkeit können Computer eine große Hilfe sein: Zwar können auch sie (schon auf Grund der Beschränktheit ihrer Pixelgraphik) keine im mathematischen Sinne korrekten Zeichnungen erstellen, aber sie bieten den großen Vorteil, daß ein einmal definiertes Objekt unter verschiedensten Blickwinkel dargestellt und vor den Augen der Schüler in alle Richtungen gedreht werden kann, was die räumliche Vorstellung erheblich erleichtert. Auch kann ein einmal definiertes Objekt jederzeit innerhalb weniger Sekunden wieder auf dem Bildschirm erscheinen – ein fast unschätzbarer Vorteil angesichts der Tatsache, daß jeder erfolgreiche Unterricht von Wiederholungen lebt.

Natürlich bringen die Limitierungen der Computergraphik auch neue Probleme. Diese dürften zwar für die Visualisierungsprobleme, mit denen ein Lehrer im Mathematikunterricht konfrontiert ist, keine Rolle spielen; in der Praxis der geometrischen Modellierung muß man sie aber beachten.

Die Methoden, mit denen diese Probleme gemeistert werden können, entstammen größtenteils der klassischen Geometrie; auf einige davon soll am Ende dieses Vortrags kurz eingegangen werden.

1. Zweidimensionale Darstellung dreidimensionaler Körper

Schon der Bildungsplan für Klasse fünf sieht vor, einfache dreidimensionale Körper und deren Schrägbilder zu behandeln.

Das Konstruktionsprinzip eines Schrägbilds ist einfach: Jedem der drei Einheitsvektoren eines räumlichen kartesischen Koordinatensystems wird ein ebener Vektor zugeordnet, wobei beim klassischen Schrägbild wohl meist die ersten beiden Einheitsvektoren auf sich selbst abgebildet werden.

In der darstellenden Geometrie gibt es diese Beschränkung nicht: Hier geht es einfach darum, jedem der drei Koordinateneinheitsvektoren des dreidimensionalen Raums *irgendeinen* zweidimensionalen Vektor zuzuordnen. Unter der hier mehr oder weniger selbstverständlichen Nebenbedingung, daß die Abbildung linear sein soll, ist damit eine Projektion $\mathbb{R}^3 \to \mathbb{R}^2$ eindeutig festgelegt. Da die Festlegung darauf basiert, daß für jede der drei Koordinatenachsen des dreidimensionalen Raums eine Achse in $\mathbb{R}^2$ festgelegt wird, sprechen darstellenden Geometer und Ingenieure bei dieser Art der zweidimensionalen Darstellung von einer *Axonometrie.*

a) Erste Schritte mit Würfeln

Natürlich kann jedes Schrägbild problemlos als zweidimensionale Graphik programmiert werden, aber das wäre eine Verschwendung der Möglichkeiten eines Computeralgebrasystems: Dieses erlaubt den Aufbau dreidimensionaler geometrischer Objekte und kümmert sich selbst um deren Projektion in die Ebene, wobei die Blickrichtung interaktiv geändert werden kann.

▶ *Laden Sie Maple worksheet* `polyeder.mws` ◀

Betrachten wir als erstes Beispiel eine Würfel! Ein derart einfacher und immer wieder gebrauchter Körper ist Maple natürlich bekannt, allerdings nur nach Laden des Pakets `plottools`. Im Befehl

`hexahedron([x,y,z], r,` *Optionen*`)`

sind alle Parameter optional; die einfachste Variante `hexahedron()` zeichnet einen Würfel mit Ecken $(\pm 1, \pm 1, \pm 1)$. Falls ein Mittelpunkt (x, y, z) angegeben ist, wird dieser Würfel entsprechend verschoben, und falls r angegeben ist, wird er anschließend noch mit den Faktor r skaliert. Ergebnis der Ausführung dieses Befehls ist allerdings keine Zeichnung, sondern eine

Datenstruktur; erst das Kommando `display` macht daraus eine Zeichnung. Die Befehlsfolge

```
with(plots): with(plottools):
display(hexahedron(), scaling=constrained);
```

bringt also einen Würfel auf den Bildschirm, wobei die erste Zeile nur einmal vor der erstmaligen Verwendung der Befehle (sowie gegebenenfalls nach einem `restart`) notwendig ist, und die Option `scaling=constrained` angibt, daß der Würfel nicht verzerrt werden soll, um – wie dies standardmäßig geschieht – die Fläche des Fensters optimal auszunutzen.

Zu weiteren *Optionen* gehört wie beim zweidimensionalen Zeichnen z.B. die Farbangabe, die Art, wie die Achsen gezeichnet werden, sowie der Plotstil. Achsen werden im Dreidimensionalen standardmäßig nicht gezeichnet; die üblichen kartesischen Koordinatenachsen bekommt man mit

`axes = normal` , während `axes = boxed`

einen Quader mit Achsbeschriftungen um das Bild zeichnet.

Der Stil ist standardmäßig `patch`, d.h. alle Polygone werden ausgefüllt und mit Kanten gezeichnet. Mit

`style = line` *bzw.* `style = patchnogrid`

kann man auch nur Kanten *bzw.* nur Flächen zeichnen.

`style = hidden`

zeichnet ebenfalls nur Kanten, eliminiert dabei aber alle (unter der jeweils aktiven Projektion) unsichtbaren.

Fährt man mit gedrückter linker Maustaste über die Zeichnung, läßt sich das Bild beliebig drehen; mit den verschiedenen Ikonen in der Leiste über dem *worksheet* lassen sich auch nachträglich noch Stil und Koordinatenachsen ändern – sofern diese nicht bereits bei der Erstellung der Zeichnung festgelegt wurden.

Natürlich kann eine Zeichnung mehr als nur einen Würfel enthalten; möchte man mit Würfeln als Bauklötzen einen Turm bauen, geht das z.B. mit

```
display(hexahedron([0,0,0],     1, color=red),
        hexahedron([0,0,1.8], 0.8, color=yellow),
        hexahedron([0,0,3.2], 0.6, color=green),
        hexahedron([0,0,4.2], 0.4, color=cyan),
        hexahedron([0,0,4.8], 0.2, color=blue),
        scaling=constrained);
```

b) Konstruktion einer Kapelle

Auch einfache Gebäude, anhand derer sich Projektionen, Schrägrisse und ähnliches demonstrieren lassen, können einfach konstruiert werden. Bauen wir etwa eine kleine Kapelle!

Die von Maple standardmäßig zur Verfügung gestellten Farben wie rot, grün, blau, ... sind dazu etwas zu grell; definieren wir also drei neue:

```
macro(lehm   = COLOR(RGB, 0.8,0.8,0.1)):
macro(schilf = COLOR(RGB, 0.65, 0.7, 0.4)):
macro(ziegel = COLOR(RGB, 0.8,0.2,0.2)):
```

Diese Farben sind durch ihre RGB-Werte, d.h. den Rot-, Grün- und Blauanteil festgelegt; `lehm` ist also im wesentlichen ein Gelbton, `schilf` etwas dunkler und mit leichter Betonung des Grünanteils, und `ziegel` ist ein abgedunkeltes Rot.

Der voluminöseste Teil einer Kirche ist das Schiff; als grobe Näherung können wir einen Quader nehmen:

```
Schiff := cuboid([0,-3,0],[14,3,4], color=lehm):
```

Der Befehl `cuboid(P, Q,` *Optionen*`)` produziert einen achsenparallelen Quader mit Ecken P und Q, die einander auf einer Raumdiagonalen gegenüberliegen.

Auch der Kirchturm ist in erster Näherung ein Quader:

```
Turm := cuboid([-2,-1,0],[0,1,15], color=schilf):
```

Das Dach ist etwas komplizierter: Seine schrägen Flächen sollen ziegelrot sein, die senkrechten Begrenzungsflächen am Anfang und Ende des Schiffs aber lehmfarben. Somit können wir hier keinen Körper als Bauklotz verwenden, sondern müssen die vier Begrenzungsflächen einzeln zeichnen. Dazu dient der Befehl

`polygon([`P_1`, ...,` P_n`],` *Optionen*`)` ,

der ein (ebenes)Polygon mit Ecken P_1 bis P_n zeichnet. Die erste Dachfläche kann also beispielsweise durch

```
Dach1 := polygon([[0,-3.6,3.25], [0,0,8.5], [14,0,8.5],
                  [14,-3.6,3.25]], color = ziegel):
```

erzeugt werden, wobei sich die Zahlen natürlich der Konstrukteur überlegen muß. Die zweite ist symmetrisch dazu, könnte also, da die x-Achse Symmetrieachse des Kirchenschiffs ist, als

```
Dach2 := polygon([[0,+3.6,3.25], [0,0,8.5], [14,0,8.5],
                  [14,+3.6,3.25]], color = ziegel):
```

erzeugt werden, allerdings ist das erstens viel Schreibarbeit und zweitens muß man, wenn man später das Endprodukt ändern möchte, jede Änderung doppelt machen. Besser ist es, die zweite Dachhälfte einfach als Spiegelung der ersten an der (x, z)-Ebene zu definieren. Ebenen werden im Spiegelungsbefehl durch drei darin liegende Punkte charakterisiert, im Falle der (x, z)-Ebene also zum Beispiel $(0, 0, 0)$, $(1, 0, 0)$ und $(0, 0, 1)$. Eine bessere Alternative ist also

```
Dach2 := reflect(Dach1, [[0,0,0], [1,0,0], [0,0,1]]):
```

Was noch fehlt, sind die seitlichen Begrenzungsdreiecke, an der Turmseite also

```
Dach3 := polygon([[0,-3,4], [0,0,8.5], [0,3,4]],
                  color=lehm):
```

und am anderen Ende dasselbe Dreieck, verschoben um den Vektor mit Komponenten $(14, 0, 0)$ entsprechend der Schiffslänge 14:

```
Dach4 := translate(Dach3, 14,0,0):
```

Genauso bekommt der Turm ein Dach aus vier roten Dreiecken, die auseinander hervorgehen durch Rotation um die Mittelachse des Turms, d.h. der Geraden durch die beiden Punkte $(-1, 0, 0)$ und $(-1, 0, 1)$:

```
TDach1 := polygon([[-2.35,-1.35,14.5], [-1,0,17],
                   [0.35,-1.35,14.5]], color=ziegel):
TDach2 := rotate(TDach1, Pi/2, [[-1,0,0], [-1,0,1]]):
TDach3 := rotate(TDach1, Pi, [[-1,0,0], [-1,0,1]]):
TDach4 := rotate(TDach1, -Pi/2, [[-1,0,0], [-1,0,1]]):
```

Um sie besser vom Rathaus des Dorfes unterscheiden zu können, sollten wir der Kapelle noch eine Apsis spendieren, etwa eine durch acht Strecken approximierte Halbellipse. Mittelpunkt der Ellipse ist der mittlere Punkt am Ende des Schiffs, also $(14, 0, 0)$; die neun Endpunkte der Strecken sind also, wenn wir Halbachsen der Länge drei und zwei nehmen, die Punkte

$$P_k = (14 + 3\cos(-\tfrac{\pi}{2} + \tfrac{k\pi}{8}), 2\sin(-\tfrac{\pi}{2} + \tfrac{k\pi}{8}, 0), \quad k = 0, \ldots, 8$$

auf dem Grund. Über jeder dieser Strecken errichten wir ein Rechteck der Höhe 4,5, und darüber als Dach ein Dreieck:

```
ApsisBoden := polygon([seq(
[14+3*cos(-Pi/2 + k*Pi/8), 2*sin(-Pi/2 + k*Pi/8), 0],
k=0..8)], color=lehm):
ApsisWaende := seq(polygon([
[14+3*cos(-Pi/2+(k-1)*Pi/8), 2*sin(-Pi/2+(k-1)*Pi/8), 0],
[14+3*cos(-Pi/2 + k*Pi/8), 2*sin(-Pi/2 + k*Pi/8), 0],
```

```
[14+3*cos(-Pi/2 + k*Pi/8), 2*sin(-Pi/2 + k*Pi/8), 4.5],
[14+3*cos(-Pi/2+(k-1)*Pi/8),
 2*sin(-Pi/2+(k-1)*Pi/8),4.5]],
color=lehm), k=1..8):
ApsisDach := seq(polygon([
[14+3*cos(-Pi/2 + k*Pi/8), 2*sin(-Pi/2 + k*Pi/8), 4.5],
[14+3*cos(-Pi/2 + (k-1)*Pi/8),
 2*sin(-Pi/2 + (k-1)*Pi/8),4.5],
[14,0,7]], color=ziegel), k=1..8):
```

Definieren wir die Kapelle als Liste

```
Kapelle := [Schiff, Turm, Dach1, Dach2, Dach3, Dach4,
            TDach1, TDach2, TDach3, TDach4,
            ApsisBoden, ApsisWaende, ApsisDach]:
```

der so definierten Körper und Flächen, erhalten wir das fertige Bild durch

```
display(Kapelle, scaling=constrained);
```

c) Darstellungsformen dreidimensionaler Objekte

Diese Kapelle soll nun als Demonstration für verschiedene zeichnerische Darstellungen dienen. Ein Schrägbild beispielsweise erhalten wir einfach durch die zusätzliche Option `style=line`; falls das Ergebnis nicht am Bildschirm betrachtet, sondern via Beamer bei Tageslicht auf eine Leinwand projiziert werden soll, empfiehlt es sich, durch die zusätzliche Option `thickness=2` (oder mehr) die Sichtbarkeit der Linien zu verbessern.

Das Ergebnis läßt sich natürlich mit der Maus oder durch Manipulation der Felder ϑ und φ über dem *worksheet* beliebig drehen, so daß man problemlos den Einfluß des Betrachtungswinkels demonstrieren kann. Falls nur eine Blickrichtung interessiert, kann man diese auch durch die Option

```
orientation = [ϑ, φ]
```

als Anfangsrichtung festlegen; andernfalls verwendet Maple die voreingestellten Werte $\vartheta = \varphi = 45°$. So erhält man etwa durch

```
display(Kapelle, style=line, thickness=2,
orientation=[-90, 0], scaling=constrained,
title="Grundriß", titlefont=[HELVETICA, 18]);
```

den Grundriß der Kapelle, die Orientierung `[-90, 90]` führt zum Aufriß und `[180, 90]` zum Kreuzriß.

Alternativ können wir auch eine Ebene im Raum durch drei Punkte P_1, P_2 und P_3 festlegen und die Kapelle (senkrecht und parallel) darauf projizieren; die Plotdatenstruktur des projizierten Bilds wird erzeugt durch

```
project(Objekt, [P1, P2, P3])
```

und kann mit display gezeichnet werden. Durch

```
display(project(Kapelle, [[0,0,0], [1,0,0], [0,1,0]]));
```

erhalten wir also die Projektion der Kapelle auf die (x, y)-Ebene, d.h. den Grundriß. Er unterscheidet sich dadurch vom durch Wahl der Orientierung erhaltenen Grundriß, daß wir nun ein zweidimensionales Bild haben: Wie wir auch den Blickwinkel verändern, wir schauen immer (wenn auch in wechselnder Perspektive) auf den Grundriß.

Einen „echten" zweidimensionalen Grundriß erhalten wir, wenn wir alles in den $\mathbb{R}^2$ projizieren und dort zeichnen. Auch das ist mit Maple möglich: Das Kommando

```
transform((x,y,z) -> [f(x,y,z), g(x,y,z)])
```

erzeugt eine Prozedur, die auf einen dreidimensionalen Plot punktweise die Abbildung

$$\mathbb{R}^3 \to \mathbb{R}^2; \qquad (x, y, z) \mapsto \big(f(x, y, z),\, g(x, y, z)\big)$$

anwendet. Mit

```
Grundriss := transform((x,y,z) -> [x, y]):
```

wird also eine Abbildung definiert, die jedem dreidimensionalen Plot dessen Grundriß zuordnet. Wenn wir den Grundriß unverzerrt und nur mit Linien statt mit Flächen sehen wollen, müssen wir noch entsprechende Optionen angeben, die wir besten ein für allemal in einer Variablen zusammenfassen:

```
OPTIONEN := axes=none, scaling=constrained,
            style=line, thickness=2:
display(Grundriss(Kapelle), OPTIONEN);
```

(Die Option axes=none ist notwendig, weil bei zweidimensionalen Plots im Gegensatz zum dreidimensionalen Fall die Koordinatenachsen standardmäßig gezeichnet werden, obwohl sie im Falle von Konstruktionszeichnungen meist unerwünscht sind.)

Zum perfekten Grundriß stört jetzt nur noch eines: Leider haben die Linien jeweils die Farbe der Fläche, die sie begrenzen; üblich für einen Grundriß wäre, sie schwarz zu zeichnen.

Das ist eine Eigenart von der dreidimensionalen Graphik von Maple: Bei einer dreidimensionalen Graphik mit style=patch bezieht sich eine eventuell vorhandene Farboption stets auf die Farbe, mit der die Polygone ausgefüllt werden; ihre Kanten werden schwarz gezeichnet. (Durch Wahl von style=patchnogrid werden diese schwarzen Kanten unterdrückt.)

Zeichnet man aber mit `style=line` oder `style=hidden`, beziehen sich dieselben Farbangaben auf die Kanten. Möchte man also etwa einen grünen Würfel mit roten Kanten zeichnen, so geht das nicht mit einem einzigen Aufruf von `hexahedron`, sondern nur mit zwei:

```
display(hexahedron([0,0,0], 1, style=patch, color=green),
        hexahedron([0,0,0], 1, style=hidden, color=red));
```

Entsprechend funktioniert auch das Zeichnen eines schwarzen Grundrisses nur dann, wenn wir in der Definition von Kapelle entweder alle Farbangaben auf schwarz setzen oder aber überhaupt keine Farbangaben machen, um diese dann erst im äußeren `display`-Kommando für den Grundriß nachzuschieben: Farbangaben in den Optionen eines `display`-Kommandos beziehen sich nur auf Objekte, die nicht bereits von einem weiter innen stehenden Kommando eine Farbe bekommen haben. Allerdings können wir auf der Ebene eines äußeren `display`-Kommandos nicht mehr unterscheiden zwischen der Farbe des Kirchenschiffs, des Turms oder des Dachs; eine Farbe, die erst dort festgelegt wird, gilt für alle Polygone, die nicht schon auf Grund eines weiter innen stehenden Kommandos eingefärbt sind.

Möchten wir daher einerseits ein farbiges Bild der Kapelle, andererseits aber einfarbige Projektionszeichnungen, so müssen wir die Farben bereits in der Definition der Kapelle als variabel vorsehen. Eine solche variabel eingefärbte Kapelle können wir als eine Prozedur `FKapelle(f1, f2, f3)` mit drei Farbparameter definieren, indem wir einfach alle bisherigen Plotbefehle mit variabel gemachten Farbangaben zusammenfassen.

Dann liefert

```
display(FKapelle(lehm, schilf, ziegel), scaling=constrained);
```

die dreidimensionale farbige Ansicht, während

```
display(Grundriss(FKapelle(black, black, black), OPTIONEN));
```

einen Grundriß mit schwarzen Linien zeichnet.

Die so gezeichnete dreidimensionale Kapelle sieht allerdings im Vergleich zur Kapelle oben betrachteten sehr viel weniger plastisch aus. Der Grund liegt darin, daß wir hier nur mit der Option `scaling=constrained` arbeiten, während für das obige Bild noch eine ganze Reihe anderer Optionen das Aussehen beeinflußten.

```
FKapelle := proc(FSchiff, FTurm, FDach)
local Schiff, Turm, Dach1, Dach2, Dach3, Dach4,
      TDach1, TDach2, TDach3, TDach4,
      ApsisBoden, ApsisWaende, ApsisDach, k;
Schiff:= cuboid([0,-3,0],[14,3,4], color=FSchiff):
Turm  := cuboid([-2,-1,0],[0,1,15], color=FTurm):
Dach1 := polygon([[0,-3.6,3.25],[0,0,8.5],[14,0,8.5],
                  [14,-3.6,3.25]], color = FDach):
Dach2 := reflect(Dach1,[[0,0,0],[1,0,0],[0,0,1]]):
Dach3 := polygon([[0,-3,4],[0,0,8.5],[0,3,4]],
                 color=FSchiff):
Dach4 := translate(Dach3, 14,0,0):
TDach1 := polygon([[-2.35,-1.35,14.5],[-1,0,17],
                   [0.35,-1.35,14.5]], color=FDach):
TDach2 := rotate(TDach1, Pi/2, [[-1,0,0],[-1,0,1]]):
TDach3 := rotate(TDach1, Pi, [[-1,0,0],[-1,0,1]]):
TDach4 := rotate(TDach1, -Pi/2, [[-1,0,0],[-1,0,1]]):
ApsisBoden := polygon([seq(
[14+3*cos(-Pi/2 + k*Pi/8), 2*sin(-Pi/2 + k*Pi/8), 0],
k=0..8)], color=FSchiff):
ApsisWaende := seq(polygon([
[14+3*cos(-Pi/2+(k-1)*Pi/8), 2*sin(-Pi/2+(k-1)*Pi/8), 0],
[14+3*cos(-Pi/2+k*Pi/8), 2*sin(-Pi/2+k*Pi/8), 0],
[14+3*cos(-Pi/2+k*Pi/8), 2*sin(-Pi/2+k*Pi/8), 4.5],
[14+3*cos(-Pi/2+(k-1)*Pi/8),
                2*sin(-Pi/2+(k-1)*Pi/8), 4.5]],
color=FSchiff), k=1..8):
ApsisDach := seq(polygon([
[14+3*cos(-Pi/2+k*Pi/8), 2*sin(-Pi/2+k*Pi/8), 4.5],
[14+3*cos(-Pi/2+(k-1)*Pi/8),
                2*sin(-Pi/2+(k-1)*Pi/8), 4.5],
[14,0,7]], color=FDach), k=1..8):
display([Schiff, Turm, Dach1, Dach2, Dach3, Dach4,
         TDach1, TDach2, TDach3, TDach4,
         ApsisBoden, ApsisWaende, ApsisDach],
scaling=constrained);
end;
```

d) Lichteffekte

Wenn kein Stil angegeben ist, wählt Maple bei einer dreidimensionalen Graphik `style=patch`, d.h. Polygone werden flächig in Farbe und mit schwarzen Kanten gezeichnet. Für viele Zwecke gerade des Geometrieunterrichts ist das eine sinnvolle Wahl: Der Betrachter bekommt einerseits einen plastischen Eindruck, sieht aber andererseits auch die Begrenzungslinien der mathematischen Objekte, aus denen das Ganze zusammengesetzt ist.

In der Natur freilich sind solche Linien selten zu sehen; falls ein einigermaßen realistischer Eindruck erwünscht wird, sollte man also auf sie verzichten. Technisch ist das einfach: Die Option `style=patchnogrid` sorgt genau dafür.

Das Resultat ist allerdings meist ziemlich enttäuschend: Offenbar geben die schwarzen Kanten unseren Augen erheblich mehr Hinweise, als uns bewußt ist.

Trotzdem erkennen wir dreidimensionale Gegenstände in der Natur problemlos auch ohne schwarz eingezeichnete Kanten, und das keinesfalls nur beim räumlichen Sehen mit zwei Augen: Auch mit nur einem Auge, etwa durch ein Fernrohr oder den Sucher einer Kamera, sieht eine Kapelle nie so flächig aus wie das, was uns ein Plotkommando mit `style=patchnogrid` auf den Bildschirm bringt – höchstens vielleicht in einem sehr dichten Nebel.

Tatsächlich erkennen wir Kanten vor allem daran, daß sich die reflektierte Helligkeit in ihrer Umgebung stark ändert. Diese Änderung fällt aber natürlich weg, wenn wir Polygone mit homogener Farbgebung auf den Bildschirm bringen.

Ein einigermaßen realistisch aussehendes Bild muß daher auch Lichteffekte mitberücksichtigen. Computergraphiksysteme bieten dazu eine Vielzahl von Möglichkeiten, deren wichtigste Gegenstand des vierten Vortrag dieses Seminars sein werden. Hauptaufgabe eines Computeralgebrasystems ist allerdings die Unterstützung des Benutzers bei der mathematischen Arbeit, und dabei spielt photorealistische Darstellung meist nur eine sehr untergeordnete Rolle.

Dementsprechend sind hier die Möglichkeiten von Maple eher rudimentär: Während Licht in der Realität von praktisch allen Seiten kommt mit jeweils verschiedener Intensität und Farbe, erlaubt Maple nur die explizite Positionierung einzelner Lichtquellen, von denen nur festgelegt werden kann, aus welcher Richtung und in welcher Farbe sie strahlen. Insbesondere ist es also nicht möglich, Effekte darzustellen, die von der Entfernung zur Lichtquelle abhängen und davon, ob die beleuchtete Fläche eher spiegelnd oder eher matt reflektiert – das können nur Graphikprogramme.

Eine Lichtquelle wird plaziert durch eine Option

```
light = [φ, ϑ, r, g, b] ,
```

wobei φ und ϑ die Kugelkoordinaten der Richtung sind, aus der das Licht kommen soll, und r, g, b seine Farbwerte bezeichnen. Soll die Szene von mehreren Lichtquellen beleuchtet werden, verwendet man einfach mehrere dieser Optionen.

Das Ergebnis einer so beleuchteten Szene ist selten realistisch: Wenn Licht ausschließlich außer einer Richtung (oder aus einigen wenigen Richtungen) kommt, führt dies zu unnatürlich scharfen Übergängen zwischen Licht und Schatten, wir bekommen also gerade das gegenteilige Problem verglichen mit der Situation ohne Lichtquellen: Dort verschwinden praktisch alle Konturen wegen nicht erkennbarer Übergänge.

Als Lösung bietet sich eine Kombination beider Methoden an: Mit der Option

```
ambientlight = [r, g, b]
```

wird ein von allen Richtungen kommendes Umgebungslicht mit Farbwerten r, g, b definiert, das auch die Flächen aufhellt, die nicht im Strahl der Lichtquellen liegen. Die Angabe

```
ambientlight = [1, 1, 1]
```

ohne zusätzliche Lichtquellen führt zum selben Ergebnis wie überhaupt keine Beleuchtungsoption, also zur standardmäßigen Darstellung durch Maple. Hintergrundlicht zusätzlich zu explizit definierten Lichtquellen muß natürlich schwächer sein; hier wird man mit

```
ambientlight = [0.3, 0.3, 0.3]
```

oder Parametern einer ähnlichen Größenordnung arbeiten. (Beleuchtung mit farbigem Licht ist bei der Visualisierung von Geometrie nur selten nützlich.)

Die bloße Kombination von Hintergrundlicht mit ein oder zwei Lichtquellen führt selbstverständlich noch nicht zu einem photorealistischen Bild; was man bestenfalls erreichen kann ist der Eindruck einer nicht sonderlich professionellen Sachphotographie, wie man sie gelegentlich in Katalogen sieht. Verglichen mit der Standarddarstellung allerdings ist die plastischere Wirkung einer solchen Zeichnung unverkennbar.

2. Platonische Körper

Als Beispiel zur Nützlichkeit von computerunterstützter Visualisierung in der dreidimimensionalen Geometrie möchte ich die fünf platonischen Körper vorstellen, so wie man sie – natürlich in deutlich ausführlicherer Darstellung – wohl auch einem Schüler vermitteln kann.