Hans Havlicek

Visualisierung

Vorlesung mit Übung im Wintersemester 2015/2016

Autor:
Hans Havlicek
Institut für Diskrete Mathematik und Geometrie
Forschungsgruppe Differentialgeometrie und Geometrische Strukturen
Technische Universität Wien
Wiedner Hauptstraße 8-10/104
A-1040 Wien

Dieses Skriptum ist für doppelseitigen Ausdruck eingerichtet.

Version: 10. Oktober 2015

Inhaltsverzeichnis

Vorwort

Schau'n Sie sich das an.
Karl Farkas,
österreichischer Kabarettist

In dieser Lehrveranstaltung geht es um die bildhafte Darstellung mathematischer Inhalte. Es ist klar, dass sich weite Teile der Mathematik einer unmittelbaren Veranschaulichung entziehen. Diese Tatsache darf aber nicht als Plädoyer für eine Mathematik ohne Bilder missverstanden werden. Ganz im Gegenteil: *Dort, wo Mathematik veranschaulicht werden kann, sollte dies auch geschehen.* Ob das mit einer einfachen Handskizze, einer Zeichnung mit Zirkel und Lineal oder mit einer am Computer erstellten Figur geschieht, ist letztlich zweitrangig.

Der Stoff der Vorlesung zerfällt zwar inhaltlich in drei Teile, diese sind aber eng verwoben und sollten im Gesamtkontext gesehen werden:

Erstens soll dargestellt werden, wie durch Projektionen definierte Bilder entstehen, welche Eigenschaften erfüllt sind und welche nicht. Die hier vermittelten Inhalte sind zeitlos und unabhängig von den eingesetzten Medien. Der Einsatz des Computers legt es nahe, die Sprache der Linearen Algebra zu verwenden. Dass dabei auch die Projektive Geometrie ins Spiel kommt, liegt auf der Hand. Eine der Wurzeln dieses Teilgebietes der Mathematik ist ja – wie schon der Name sagt – die Projektion aus einem Zentrum auf eine Ebene.

Zweitens spielt bei der Darstellung von mathematischen Objekten die Differentialgeometrie eine große Rolle, werden doch in den vielen Fällen nicht bloß durch Ebenen begrenzte Körper sondern auch Kurven und Flächen als Bildinhalt auftreten. Wir stellen daher einige differentialgeometrische Grundlagen zusammen und beschreiben auch, worauf bei der Darstellung von Kurven und Flächen zu achten ist. Letztes geht über den gängigen Stoff einer Vorlesung über klassische Differentialgeometrie hinaus.

Der dritte Punkt betrifft das in den Übungen einzusetzende Handwerkszeug. Hier führt bei anspruchsvollen Problemstellungen gegenwärtig kein Weg am Computer vorbei. Wir beschränken uns bewusst auf ein einziges Programm, nämlich *POV-Ray*. Es ist für verschiedene Betriebssysteme frei verfügbar, unterstützt zahlreiche mathematische Befehle und liefert mit geringem Aufwand Bilder von erstklassiger technischer Qualität. Es geht in der Lehrveranstaltung vor allem darum, auf Stärken und Schwächen der Software hinzuweisen sowie an Hand von Beispielen verschiedene Einsatzmöglichkeiten in der Mathematik aufzuzeigen. Viele der Figuren dieses Skriptums wurden mit POV-Ray erstellt. Dabei wurden bewusst verschiedenste stilistische Darstellungsweisen gewählt, um Anschauungsmaterial zur Verfügung zu stellen.

Weitere Anregungen zum Thema „Visualisierung in der Mathematik“ liefert beispielsweise das Buch von G. Glaeser und K. Polthier [6]. Die bestechenden Bilder dieses Werkes wurden mit dem Programm *Open Geometry* erstellt. Lohnenswert ist auch ein Blick in das Buch von

T. Arens et al. [1] mit seinen rund 1400 farbigen Illustrationen. Aber auch aus Lehrbüchern der Darstellenden Geometrie, etwa von H. Brauner, [5], F. Hohenberg [9] und W. Wunderlich [11], [12] kann heute noch gelernt werden, wie eindrucksvolle und das wesentliche klar hervorhebende Illustrationen aussehen, obgleich die Zeiten von mit der Reissfeder angefertigten Tuschezeichnungen schon länger vorbei ist.

Zum Schluss möchte ich mich bei allen herzlich bedanken, die mich bei der Erstellung dieses Skriptums unterstützt haben. Dies sind zunächst einmal zahlreiche Studierende der Technischen Universität Wien. Sie halfen mit mit ihren kritischen Anmerkungen zu früheren Versionen dieses Skriptums, die Anzahl der Fehler zu vermindern und so manche Unzulänglichkeit auszumerzen. Georg Glaeser (Universität für Angewandte Kunst, Wien) und Tomáš Pajdla (České Vysoké Učení Technické, Prag) stellten freundlicherweise Fotos zur Verfügung. Mein besonderer Dank gilt aber Boris Odehnal (Technische Universität Dresden und Technische Universität Wien). Viele Illustrationen basieren auf seinen Entwürfen. Ebenso konnte ich für Teile des Textes auf seine umfangreichen Unterlagen zurückgreifen.

Wien, im Februar 2012 Hans Havlicek

Vorwort zur Ausgabe Wintersemester 2015/2016

Software ist eingefrorene Theorie
und als solche stets veraltet.
Gerhard Geise,
Professor an der Technischen Universität Dresden

Die vorliegende Version des Skriptums unterscheidet sich nur geringfügig von früheren Ausgaben. Selbstverständlich wurden alle bekannten Fehler korrigiert. Text und Beispiele wurden an POV-Ray 3.7 angepasst. Dies ist die derzeit aktuelle Version des Programms. Im Literaturverzeichnis konnten mit [2] und [7] zwei Neuauflagen berücksichtigt werden.

Wien, im Oktober 2015 Hans Havlicek

Kapitel 1

Parallelprojektion

1.1 Koordinatensysteme

1.1.1 Wenn im Folgenden vom *Raum* die Rede ist, so meinen wir den *Anschauungsraum* $\mathcal{A}$, das ist der nicht näher präzisierte Raum in dem wir leben, wobei wir uns vorstellen, dass die im Kleinen erlebte Geometrie auch im Großen gilt. Geben wir im Raum ein *kartesisches Koordinatensystem* (U, P_1, P_2, P_3) vor, so können wir jeden Punkt durch seine drei Koordinaten beschreiben, wobei wir diese als *Zeilenvektor* schreiben. Es dient uns dann der euklidische Raum $\mathbb{R}^3$ als mathematisches Modell für den Anschauungsraum. Durch die Vorgabe eines Koordinatensystems ist eine *Längenmessung* mitbestimmt: Die Einheitsstrecken auf den Koordinatenachsen haben die Länge 1.

Der Raum $\mathcal{A}$ ist bekanntlich orientierbar. Wir verwenden ausnahmslos *kartesische Rechtssysteme*. Die drei Koordinatenachsen sind also so angeordnet wie *Daumen*, *Zeigefinger* und *Mittelfinger* der *rechten Hand*. Dabei kann die folgende Merkregel hilfreich sein (vgl. Abbildung 1.1):

- Die erste Koordinatenachse weist nach vorne.
- Die zweite Koordinatenachse weist nach rechts.
- Die dritte Koordinatenachse weist nach oben.

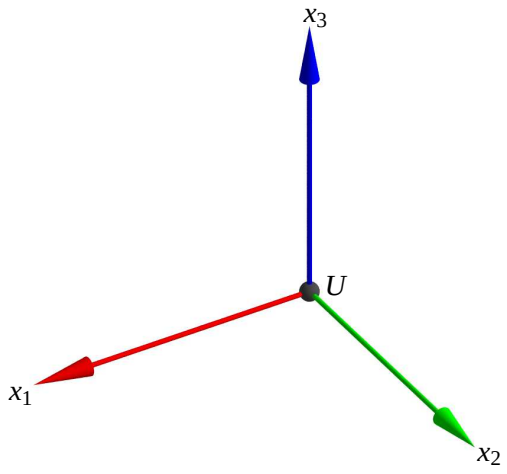


Abbildung 1.1: Rechtssystem

Da wir ein Koordinatensystem aber drehen und schieben können, wird diese Koppelung mit den anschaulichen Begriffen *vorne*, *rechts* und *oben* nicht immer gegeben sein.

1.1.2 Jede Ebene und jede Gerade des Raumes ist ebenfalls (auf genau zwei Arten) orientierbar. Durch die gegebene Orientierung im Raum wird aber aber keine der beiden möglichen Orientierungen einer Ebene oder Geraden ausgezeichnet. Wir werden darauf später noch zurückkommen.

1.2 Definition und Eigenschaften einer Parallelprojektion

1.2.1 Wir geben im Raum eine Bildebene π und eine nicht in π liegende Gerade s vor. Die *Parallelprojektion* p in Richtung s auf die Ebene π ist wie folgt erklärt:

$$p : \mathcal{A} \to \pi : X \mapsto s_X \cap \pi, \tag{1.1}$$

wobei s_X die (eindeutig bestimmte) zu s parallele Gerade durch X bezeichnet (vgl. Abbildung 1.2). Die zu s parallelen Geraden werden die *Sehgeraden* der Parallelprojektion genannt. Zwei Punkte haben offenbar genau dann denselben Bildpunkt, wenn sie derselben Sehgeraden angehören.

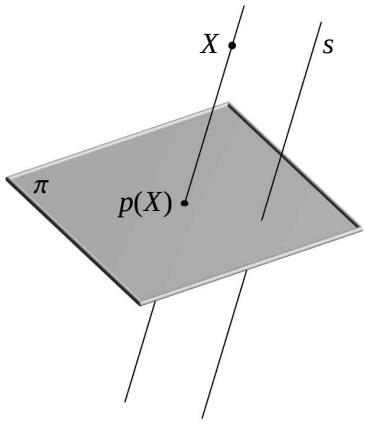


Abbildung 1.2: Parallelprojektion

1.2.2 Wählen wir den Ursprung U eines räumlichen Koordinatensystems in der Bildebene π, so wird p durch eine Projektion $\mathbb{R}^3 \to \mathbb{R}^3$ im Sinne der Linearen Algebra beschrieben.[1] Diese Projektion wird durch eine reelle 3×3 Matrix C mit den beiden Eigenschaften

$$\operatorname{rg} C = 2, \tag{1.2}$$

$$C \cdot C = C \tag{1.3}$$

festgelegt. Die Sehgerade durch den Ursprung entspricht dabei dem Kern von C, der Bildraum (Zeilenraum) von C bestimmt die Bildebene π. Umgekehrt legt auch jede Matrix C mit diesen beiden Eigenschaften eine Parallelprojektion auf eine Ebene durch U fest.

[1] Die im Folgenden verwendeten Ergebnisse aus der Linearen Algebra können etwa in [8] nachgeschlagen werden.

Die Matrix C lässt sich leicht angeben, wenn s und π spezielle Lage zum Koordinatensystem haben. Die Parallelprojektion auf die Ebene $x_3 = 0$ in Richtung des Vektors $(c_1, c_2, -1)$ hat die Matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ c_1 & c_2 & 0 \end{pmatrix}. \tag{1.4}$$

Die Parallelprojektion in Richtung der x_3-Achse auf die Ebene $c_1x_1 + c_2x_2 - x_3 = 0$ hat die Matrix

$$\begin{pmatrix} 1 & 0 & c_1 \\ 0 & 1 & c_2 \\ 0 & 0 & 0 \end{pmatrix}. \tag{1.5}$$

Aus der Beschreibung von p mit Hilfe einer Matrix C lassen sich die folgenden Eigenschaften leicht nachrechnen:

Satz 1.2.3 (Abbildung von Geraden und Ebenen) *Sei $p : \mathcal{A} \to \pi$ die Parallelprojektion in Richtung einer Geraden s auf eine Ebene π. Dann gelten die folgenden Eigenschaften:*

(a) *Jede Sehgerade wird unter p auf einen Punkt abgebildet.*

(b) *Jede nicht zu s parallele Gerade wird unter p bijektiv und affin auf eine Gerade von π abgebildet. Die Einschränkung auf eine solche Gerade ist daher teilverhältnistreu.*

(c) *Jede zu s parallele Ebene wird unter p auf eine Gerade abgebildet.*

(d) *Jede nicht zu s parallele Ebene wird unter p bijektiv und affin auf die Ebene π abgebildet.*

Die zu s parallelen Geraden und Ebenen werden auch *projizierend* genannt. Nach (d) gehen unter p nicht projizierende parallele Geraden des Raumes in parallele Geraden der Ebene π über.

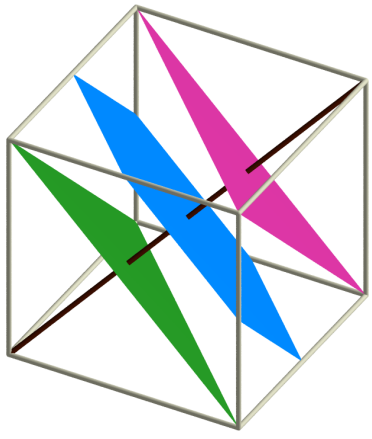


Abbildung 1.3: Drei ebene Schnitte eines Würfels

Zur Illustration von Satz 1.2.3 kann Abbildung 1.3 dienen. Sie zeigt eine Parallelprojektion eines Würfels, der mit drei Ebenen geschnitten wird. Die drei Ebenen sind zu einer der Raumdiagonalen orthogonal. Die grüne und die violette Ebene verlaufen durch drei Ecken, die blaue Ebene verbindet sechs Mittelpunkte von Kanten. Aus dem Bild ist nicht unmittelbar ersichtlich, dass als Schnittfiguren zwei gleichseitige Dreiecke und ein regelmäßiges Sechseck auftreten.

Hingegen können die im Raum auftretenden Teilverhältnisse (etwa auf der Raumdiagonalen) und die Parallelität gewisser Geraden aus dem Bild unmittelbar abgelesen werden.

Satz 1.2.4 (Längentreue) *Unter einer Parallelprojektion $p : \mathcal{A} \to \pi$ besitzt eine Strecke (X, Y) mit $X \neq Y$ genau dann dieselbe Länge wie ihr Bild $(p(X), p(Y))$, falls die Gerade XY eine der nachfolgenden Eigenschaften besitzt:*

(a) *XY ist zur Bildebene π parallel.*

(b) *Wird XY an einer zu den Sehgeraden orthogonalen Ebene gespiegelt, so ist diese gespiegelte Strecke zu π parallel.*

Beweis. Sei eine Strecke (X, Y) mit $X \neq Y$ gegeben. Wir unterscheiden zwei Fälle:

Fall 1: XY ist eine Sehgerade. Hier gilt $\overline{XY} \neq 0 = \overline{p(X)p(Y)}$. Als Sehgerade ist XY zur Bildebene nicht parallel. Unter einer Spiegelung an einer zu den Sehgeraden orthogonalen Ebene bleibt XY fest. Die Gerade XY erfüllt daher weder (a) noch (b).

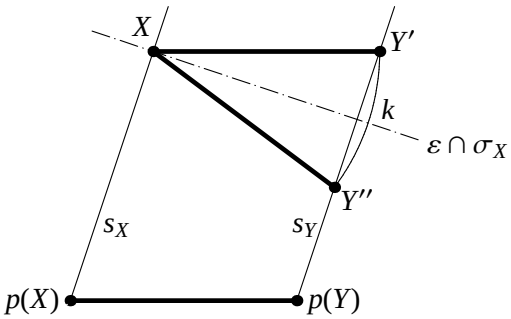


Abbildung 1.4: Längentreue

Fall 2: XY ist keine Sehgerade. Die Sehgeraden $s_X \ni X$ und $s_Y \ni Y$ spannen eine Ebene ε auf. Unter der Schiebung τ mit $p(X) \mapsto X$ geht die Bildstrecke $(p(X), p(Y))$ in eine gleich lange Strecke (X, Y') mit $Y' \in s_Y$ über. Wir betrachten den Kreis k in ε mit dem Mittelpunkt X durch Y' und setzen $k \cap s_Y =: \{Y', Y''\}$, wobei $Y' = Y''$ möglich ist. Mit Hilfe der Abstandsformel für zwei Punkte folgt leicht

$$\begin{aligned}
&\overline{XZ} < \overline{p(X)p(Y)} && \text{für alle Punkte } Z \in s_Y \text{ zwischen } Y' \text{ und } Y'',\\
&\overline{XZ} = \overline{p(X)p(Y)} && \text{für } Z = Y', Y'',\\
&\overline{XZ} > \overline{p(X)p(Y)} && \text{für alle anderen Punkte } Z \in s_Y.
\end{aligned}$$

Daher gilt

$$\overline{XY} = \overline{p(X)p(Y)}$$

genau dann, wenn $Y = Y'$ oder $Y = Y''$ erfüllt ist (vgl. Abbildung 1.4) . Ersteres ist zur Bedingung (a) und letzteres zur Bedingung (b) äquivalent, da Y' aus Y'' durch die Spiegelung an jener Ebene σ_X durch X übergeht, welche zu den Sehstrahlen orthogonal steht. □

Der obige Beweis zeigt, dass es unter einer Parallelprojektion immer Strecken gibt, die im Bild verkürzt dargestellt werden. Genau dann, wenn s zu π nicht orthogonal ist, gibt es auch Strecken, die im Bild verlängert erscheinen.[2]

[2] Sprichwort: Wenn die Sonne tief steht, werfen auch Zwerge lange Schatten.

Satz 1.2.5 (Differenzierbarkeit) *Die durch C beschriebene Projektion $\mathbb{R}^3 \to \mathbb{R}^3$ ist beliebig oft stetig differenzierbar. In jedem Punkt von $\mathbb{R}^3$ ist die Matrix C ihre totale Ableitung.*[3]

Beweis. Nachrechnen. □

Satz 1.2.5 stellt sicher, dass eine Parallelprojektion ein „anschauliches“ Bild des Raumes liefert. Dies trifft freilich auch auf das *Panoramabild* in Abbildung 1.5 zu, der keine Parallelprojektion zu Grunde liegen kann. Das zeigen die krummlinigen Bilder von Geraden.

Abbildung 1.5: Panoramabild (Technisches Museum Prag), Photo: Tomáš Pajdla

1.3 Definition und Eigenschaften einer Orthogonalprojektion

1.3.1 Sind die Sehgeraden einer Parallelprojektion p zur Bildebene orthogonal ($s \perp \pi$), so wird von einer *Orthogonalprojektion* oder *Normalprojektion* gesprochen.

Orthogonalprojektionen haben gegenüber beliebigen Parallelprojektionen einige zusätzliche Eigenschaften, die wir in diesem Abschnitt zusammenstellen.

1.3.2 Wir schließen an 1.2.2 an. Eine reelle 3×3 Matrix C beschreibt genau dann eine Orthogonalprojektion in $\mathbb{R}^3$, wenn sie neben den Eigenschaften (1.2) und (1.3) noch die Eigenschaft

$$C = C^{\mathrm{T}} \tag{1.6}$$

besitzt, d. h. C muss *symmetrisch* sein. (Vgl. Lineare Algebra: Erfüllt C die Eigenschaften (1.2) und (1.3), so trifft das auch auf die transponierte Matrix C^{T} zu. Die Matrix C^{T} beschreibt die Projektion auf eine zu s orthogonale Ebene durch den Ursprung U in Richtung der zu π orthogonalen Geraden. Die Matrizen in (1.4) und (1.5) illustrieren diesen Sachverhalt.)

[3] Die beiden Eigenschaften gelten sinngemäß für jede lineare Abbildung $\mathbb{R}^n \to \mathbb{R}^m$.

Die Matrix einer Orthogonalprojektion lässt sich etwa wie folgt bestimmen: Sei $\boldsymbol{s} \in \mathbb{R}^3$ ein normierter Richtungsvektor der Sehgeraden. Ist $\boldsymbol{x} \in \mathbb{R}^3$ der Koordinatenvektor eines Punktes, so hat sein Bildpunkt den Koordinatenvektor

$$\boldsymbol{x} - (\boldsymbol{s} \cdot \boldsymbol{x})\boldsymbol{s}. \tag{1.7}$$

Vgl. dazu Abbildung 1.6. Indem für $\boldsymbol{x}$ der Reihe nach die Vektoren der kanonischen Basis von $\mathbb{R}^3$ eingesetzt werden, ergeben sich die Zeilen der zugehörigen Matrix in richtiger Reihenfolge.

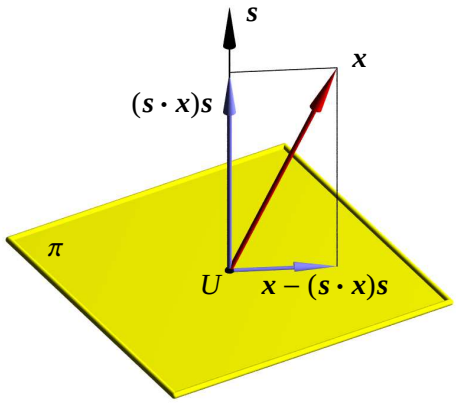


Abbildung 1.6: Orthogonalprojektion eines Vektors

Aus dem Beweis zu Satz 1.2.4 lesen wir ab:

Satz 1.3.3 (Längenverzerrung) *Eine Strecke (X, Y) mit $X \neq Y$ besitzt unter einer Orthogonalprojektion genau dann dieselbe Länge wie ihr Bild $(p(X), p(Y))$ falls die Gerade XY zur Bildebene parallel ist. Das Bild einer Strecke unter einer Orthogonalprojektion ist niemals länger als die Strecke selbst.*

Es gibt eine weitere Eigenschaft, die eine Orthogonalprojektion unter den Parallelprojektionen auszeichnet.

Satz 1.3.4 (Satz vom rechten Winkel) *Es seien $p : \mathcal{A} \to \pi$ eine Orthogonalprojektion und g, h zwei orthogonale Geraden[4], aber keine Sehgeraden von p. Dann gilt*

$$p(g) \perp p(h) \tag{1.8}$$

genau dann, falls mindestens eine der Geraden g, h zur Bildebene von p parallel ist.

Beweis. Es seien $\boldsymbol{s}$ ein normierter Richtungsvektor der Sehgeraden von p und $\boldsymbol{a}$ bzw. $\boldsymbol{b}$ Richtungsvektoren der Geraden g bzw. h. Die Bildgeraden $p(g)$ bzw. $p(h)$ haben nach (1.7) die Richtungsvektoren $\boldsymbol{a} - (\boldsymbol{s} \cdot \boldsymbol{a})\boldsymbol{s} \neq \boldsymbol{o}$ bzw. $\boldsymbol{b} - (\boldsymbol{s} \cdot \boldsymbol{b})\boldsymbol{s} \neq \boldsymbol{o}$. Wir berechnen

$$\begin{aligned}(\boldsymbol{a} - (\boldsymbol{s} \cdot \boldsymbol{a})\boldsymbol{s}) \cdot (\boldsymbol{b} - (\boldsymbol{s} \cdot \boldsymbol{b})\boldsymbol{s}) &= \underbrace{\boldsymbol{a} \cdot \boldsymbol{b}}_{=0} - (\boldsymbol{s} \cdot \boldsymbol{b})(\boldsymbol{a} \cdot \boldsymbol{s}) - (\boldsymbol{s} \cdot \boldsymbol{a})(\boldsymbol{s} \cdot \boldsymbol{b}) + (\boldsymbol{s} \cdot \boldsymbol{a})(\boldsymbol{s} \cdot \boldsymbol{b}) \\ &= -(\boldsymbol{s} \cdot \boldsymbol{a})(\boldsymbol{s} \cdot \boldsymbol{b}).\end{aligned}$$

[4]Orthogonale Geraden sind durch orthogonale Richtungsvektoren gekennzeichnet. Sie brauchen keinen gemeinsamen Punkt zu besitzen!

Die Bildgeraden sind daher genau dann orthogonal, wenn mindestens eine der Geraden g, h zu den Sehgeraden orthogonal steht oder – in anderen Worten – zur Bildebene parallel ist. □

Es sei an dieser Stelle daran erinnert, dass bei einer Orthogonalprojektion genau die zur Bildebene parallelen Geraden unverzerrt erscheinen.

Zur Illustration der Sätze 1.3.3 und 1.3.3 zeigt Abbildung 1.7 ein kartesisches Koordinatensystem sowie einen Kreis in der x_1x_2-Ebene unter einer Orthogonalprojektion. Das Bild des Kreises ist eine Ellipse. Im Bild gilt folgende Eigenschaft: *Die Hauptachse der Ellipse ist zum Bild der x_3-Achse orthogonal.*

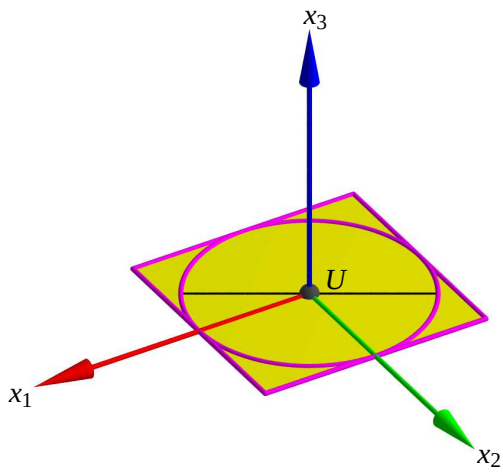


Abbildung 1.7: Kreis in der x_1x_2-Ebene bei Orthogonalprojektion

Zum Beweis betrachten wir das Urbild der Hauptachse in der x_1x_2-Ebene. Diese Gerade g ist nach Satz 1.3.3 zur Bildebene parallel, da sie im Bild den längsten Durchmesser der Ellipse trägt. So wie alle Geraden der x_1x_2-Ebene Ebene, ist g zur x_3-Achse orthogonal. Nach Satz 1.3.4 erscheint dieser rechte Winkel im Bild unverzerrt.

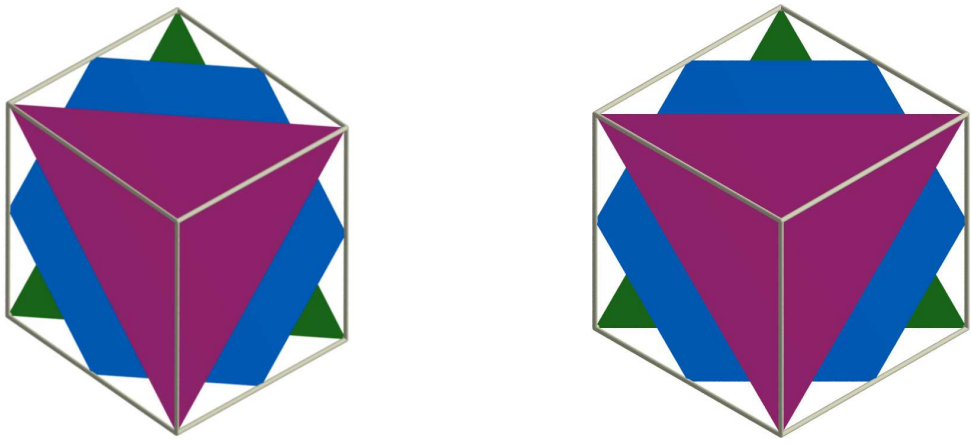


Abbildung 1.8: Drei ebene Schnitte eines Würfels in Schräg- und Orthogonalprojektion

Als weitere Beispiele betrachten wir nochmals den Würfel aus Abbildung 1.3. Die beiden Figuren in Abbildung 1.8 zeigen den Würfel unter Parallelprojektionen, bei denen die zu den Schnittebenen orthogonale Raumdiagonale projizierend ist. Links liegt keine Orthogonalprojektion vor, was daran zu erkennen ist, dass die beiden gleichseitigen Dreiecke und das regel-

mäßige Sechseck verzerrt erscheinen. Das Bild rechts zeigt eine Orthogonalprojektion, sodass die Dreiecke und das Sechseck im Raum zu ihren Bilder kongruent sind.

1.4 Abbildung in eine allgemeine Ebene

1.4.1 Der Begriff der Parallelprojektion ist für unsere Zwecke noch zu speziell, da die Lage der Bildebene π im Raum wesentlich eingeht. Jeder Punkt von π ist ja sein eigener Bildpunkt. Wir betrachten im Folgenden zusätzlich eine beliebige Ebene ψ, deren Lage im Raum nicht von Interesse ist, und Abbildungen $\mathcal{A} \to \psi$, die sich wie folgt zusammensetzen:

- Zuerst wird mittels einer Parallelprojektion p auf eine Ebene π (im Raum) projiziert.
- Dann wird die Ebene π auf ψ unter einer Kongruenzabbildung β, einer Ähnlichkeitsabbildung β oder einer Affinität β abgebildet.

Zur Unterscheidung nennen wir ab nun π die *Zwischenbildebene* und ψ die *Bildebene* der Abbildung

$$\alpha := \beta \circ p.$$

Die Ebene ψ entspricht dabei etwa einer Buchseite oder einem Bildschirm. Abbildung 1.9 zeigt

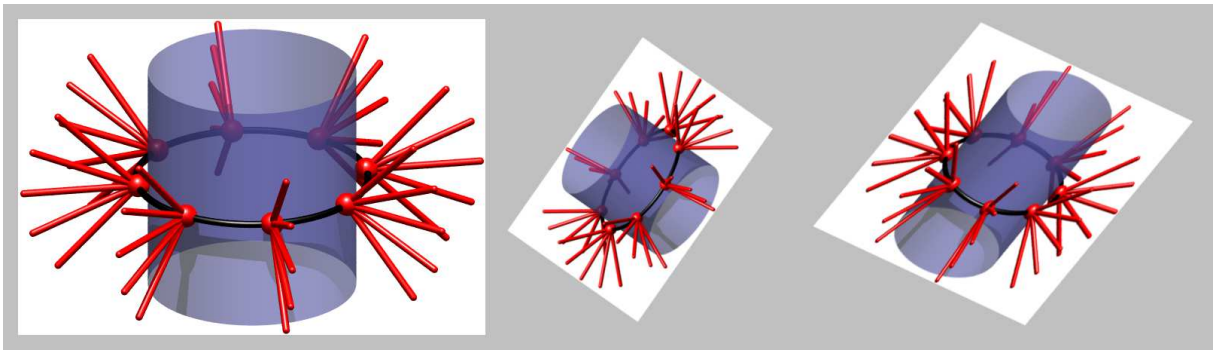


Abbildung 1.9: Drei Bilder in der Ebene ψ zur selben Orthogonalprojektion

drei Bilder eines Zylinders und einiger Geraden. Allen Abbildungen liegt dieselbe Orthogonalprojektion zu Grunde. Es wurde links eine Kongruenzabbildung (Maßstab $1:1$), in der Mitte eine Ähnlichkeitsabbildung (Verkleinerung) und rechts eine Affinität verwendet, um die Bilder in ψ zu erzeugen.

1.4.2 Wir wählen nun in der Bildebene ψ ein kartesisches Koordinatensystem (U', P_1', P_2'), dessen Ursprung mit dem α-Bild des räumlichen Ursprungs U übereinstimmt. Dann besitzt α als Koordinatendarstellung eine 3×2 Matrix C mit Rang 2. (Wäre $\alpha(U) \neq U'$, so müssten wir zusätzlich eine Schiebung von $\mathbb{R}^2$ berücksichtigen.) Umgekehrt bestimmt auch jede derartige 3×2 Matrix eine lineare Abbildung $\mathbb{R}^3 \to \mathbb{R}^2$, die wir als Koordinatendarstellung einer surjektiven affinen Abbildung des Raumes auf eine Ebene interpretieren können. Eine einfache Aussage enthält der folgende Satz:

Satz 1.4.3 *Jede surjektive affine Abbildung $\alpha : \mathcal{A} \to \psi$ lässt sich in eine Parallelprojektion p auf eine Ebene π und eine Affinität $\beta : \pi \to \psi$ zerlegen.*

Beweis. Sei C eine Koordinatenmatrix von α. Die Matrix C hat nach der Rangformel einen eindimensionalen Kern. Wir wählen einen beliebigen (2-dimensionalen) Unterraum $\boldsymbol{W} \subset \mathbb{R}^3$, der zum Kern komplementär liegt. Bezüglich unseres räumlichen Koordinatensystems entspricht $\ker C$ einer Geraden s durch den Ursprung U und $\boldsymbol{W}$ einer Ebene π durch den Ursprung. Wir bezeichnen die Parallelprojektion $\mathcal{A} \to \pi$ in Richtung s mit p und setzen $\beta := \alpha|\pi$; vgl. Abbildung 1.10. Dann ist β in der Tat eine Affinität. Da zwei verschiedene Punkte genau dann dasselbe Bild unter α besitzen, wenn ihre Verbindungsgerade zu s parallel ist, hat jeder Raumpunkt X dasselbe α-Bild wie $p(X)$. Daraus folgt

$$\alpha = (\alpha|\pi) \circ p = \beta \circ p,$$

also eine Zerlegung der oben beschriebenen Art. □

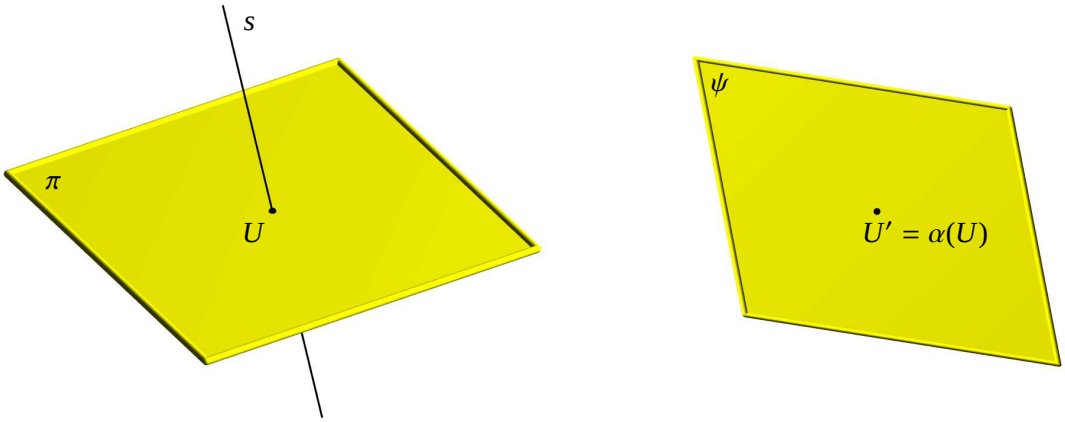


Abbildung 1.10: Definition von β

Im obigen Satz geht nicht ein, dass $\mathcal{A}$ ein euklidischer Raum ist. Die Zwischenbildebene π ist auch nicht eindeutig bestimmt. Vielmehr kann jede nicht projizierende Ebene, egal ob sie durch U geht oder nicht, als Zwischenbildebene verwendet werden.

Eine schärfere Aussage liefert der folgende Satz.

Satz 1.4.4 (Satz von Pohlke) *Jede surjektive affine Abbildung $\alpha : \mathcal{A} \to \psi$ lässt sich in eine Parallelprojektion p auf eine Ebene π und eine Ähnlichkeitsabbildung $\beta : \pi \to \psi$ zerlegen.*

Beweis. Sei wieder C eine Koordinatenmatrix von α. Ferner seien $w_1 \geq w_2 > 0$ die Singulärwerte[5] von C. Nach dem Satz über die Singulärwertezerlegung gibt es eine ONB $(\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3)$ von $\mathbb{R}^3$ so, dass gilt:

- $\|\boldsymbol{b}_i \cdot C\| = w_i$ für $i \in \{1, 2\}$.
- $\boldsymbol{b}_1 \cdot C \perp \boldsymbol{b}_2 \cdot C$.
- $\boldsymbol{b}_3 \cdot C = \boldsymbol{o}$.

[5] Die Singulärwerte einer reellen Matrix C sind bekanntlich die Quadratwurzeln aus den nicht verschwindenden Eigenwerten der symmetrischen Matrix CC^{T}.

Wir betrachten das zu dieser Basis gehörende Koordinatensystem (U, Q_1, Q_2, Q_3). Die Gerade UQ_3 ist eine Sehgerade, die beiden anderen Koordinatenachsen liegen in einer zu den Sehgeraden orthogonalen Ebene durch den Ursprung U. Es gibt mindestens eine Ebene π durch die Koordinatenachse UQ_2, die mit der Ebene $\sigma := UQ_1Q_2$ einen Winkel γ mit

$$\cos\gamma = \frac{w_2}{w_1} \tag{1.9}$$

einschließt. Projizieren wir Q_1 in Richtung UQ_3 auf die Ebene π, so erhalten wir einen Punkt $\widetilde{Q}_1$. Es folgt

$$\overline{U\widetilde{Q}_1} = \frac{w_1}{w_2}. \tag{1.10}$$

Die Einschränkung $\alpha|\pi$ ist eine Ähnlichkeitsabbildung zum Faktor w_2, da das rechtwinkelige

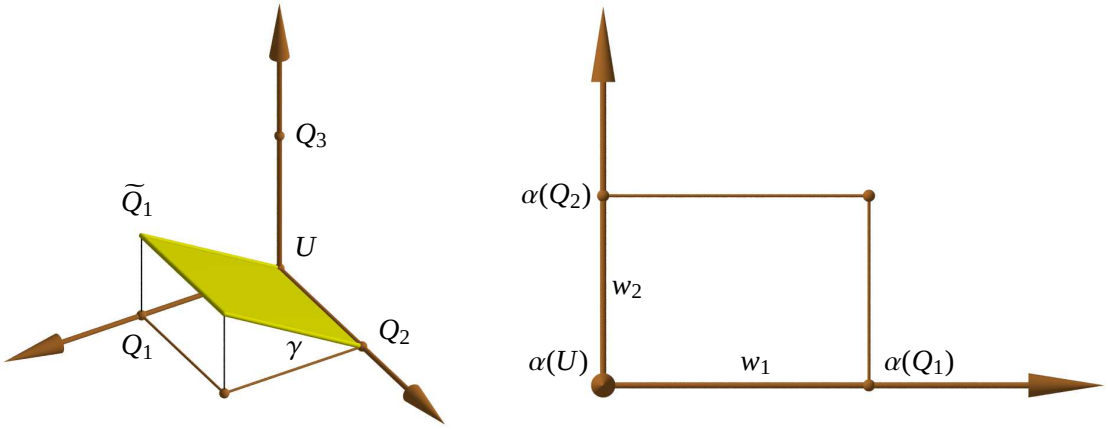


Abbildung 1.11: Definition der Ebene π

Dreieck $\widetilde{Q}_1, U, Q_2$ in das rechtwinkelige Dreieck $\alpha(\widetilde{Q}_1) = \alpha(Q_1), \alpha(U), \alpha(Q_2)$ übergeht und die Längen zugeordneter Katheten sich um den Faktor w_2 unterscheiden; vgl. Abbildung 1.11, wo aber dieses Dreieck zu einem Rechteck ergänzt wurde. □

Der obige Beweis zeigt, wie sich aus den beiden Singulärwerten w_1 und w_2 der Maßstabsfaktor von α und der Winkel γ einer geeigneten Zwischenbildebene gegen eine zu den Sehgeraden orthogonale Ebene σ berechnen lässt. Wir übergehen hier den einfachen Beweis, dass die Zwischenbildebene bis auf Translationen und Spiegelung an σ eindeutig bestimmt ist.

Daraus ergibt sich, dass α genau dann in eine Orthogonalprojektion und eine Ähnlichkeitsabbildung (zum Faktor w_2) faktorisiert werden kann, falls

$$w_1 = w_2. \tag{1.11}$$

Denn genau in diesem Fall ist $\cos\gamma = 1$ bzw. $\gamma = 0$.

Die obigen Ergebnisse liefern nebenbei auch den mathematischen Hintergrund für *Handskizzen* räumlicher Objekte. Zeichnen wir auf einem Blatt Papier (in einer Ebene ψ) vier Punkte U', Q_1', Q_2' und Q_3', die nicht derselben Geraden angehören, so gibt es genau eine surjektive affine Abbildung $\alpha : \mathcal{A} \to \psi$, welche ein gegebenes räumliches kartesisches Koordinatensystem (U, Q_1, Q_2, Q_3) der Reihe nach auf die gegebenen Punkte abbildet. Unter Beachtung der Eigenschaften dieser Abbildung können dann weitere Bildpunkte ergänzt werden (etwa das Bild des Einheitswürfels). Bei dieser Vorgangsweise ergibt sich nach dem Satz von Pohlke ein zu einer Parallelprojektion ähnliches Bild. Die Projektionsrichtung und den Ähnlichkeitsfaktor

braucht man dazu nicht zu kennen. Dieses Verfahren wird in der Darstellenden Geometrie als *Axonometrie* bezeichnet.[6]

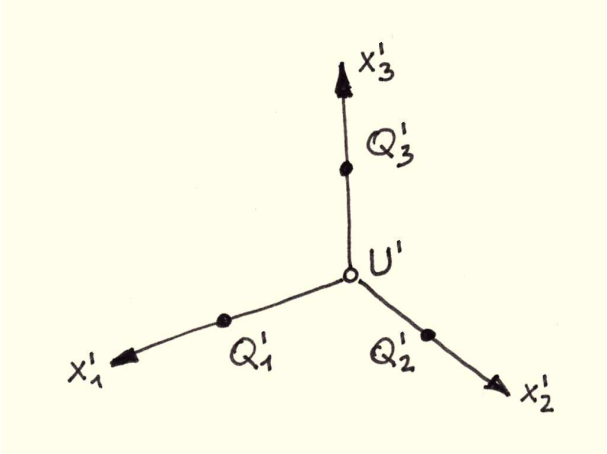


Abbildung 1.12: Handskizze (Axonometrie)

1.5 Sichtbarkeit und Orientierung der Bildebene

1.5.1 Bei unseren bisherigen Untersuchungen haben wir noch nicht berücksichtigt, dass ein Punkt einen anderen Punkt verdecken kann. Dazu brauchen wir als zusätzliche Information eine *Orientierung* der Sehgeraden.

Wir orientieren zunächst eine Sehgerade s_0 einer Parallelprojektion durch Auszeichnung eines Punktepaares (X_0, Y_0) mit $X_0, Y_0 \in s_0$ und $X_0 \neq Y_0$. Ein beliebiges Punktepaar (X, Y) mit $X, Y \in s_0$ und $X \neq Y$ heißt zu (X_0, Y_0) *gleichorientiert*, falls

$$\overrightarrow{XY} = c\overrightarrow{X_0Y_0} \text{ für eine reelle Zahl } c > 0. \tag{1.12}$$

Diese Definition bleibt sinnvoll, wenn wir (X, Y) auf einer beliebigen Sehgeraden wählen. Aus den Sehgeraden werden auf diese Weise daher *Sehstrahlen* (orientierte Sehgeraden).

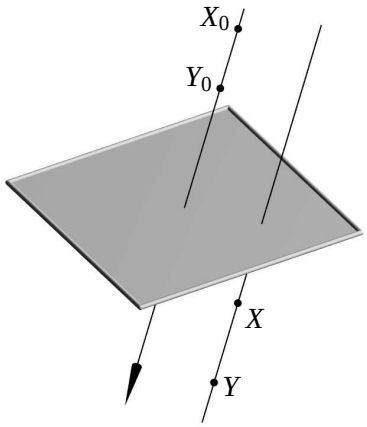


Abbildung 1.13: Sichtbarkeit

[6] Vgl. [5], [9] oder [12].

Anschaulich formuliert: Wir stellen wir uns vor, dass wir „in Richtung des Vektors $\overrightarrow{X_0Y_0}$ projizieren“: Sind X und Y zwei verschiedene *undurchsichtige Massepunkte* mit demselben Bildpunkt und gilt

$$\overrightarrow{XY} = c\overrightarrow{X_0Y_0} \text{ für eine reelle Zahl } c > 0, \tag{1.13}$$

so sei im Bild der Punkt X *sichtbar*. Der Punkt Y hingegen gilt als unsichtbar, da er von X *verdeckt* wird. Die Lage der Zwischenbildebene geht hier nicht ein!

1.5.2 Mathematisch gesehen zerlegt eine *Ebene* die Menge aller nicht in ihr liegenden Punkte zwar in zwei *Seiten* (*Halbräume*), diese sind aber völlig gleichberechtigt.

Durch die Orientierung der Sehgeraden einer Parallelprojektion und die Orientierung des Raumes ist aber eine *Orientierung der Zwischenbildebene* π mitbestimmt: Ein kartesisches Koordinatensystem (Q, Q_1, Q_2) der Bildebene sei ein Rechtssystem, wenn es so zu einem räumlichen kartesischen Rechts-Koordinatensystem (Q, Q_1, Q_2, Q_3) ergänzt werden kann, dass $(Q_3, p(Q_3))$ zur Orientierung der Sehstrahlen gleichorientiert ist. Im Falle einer Orthogonalprojektion verdeckt also Q_3 den Ursprung Q.

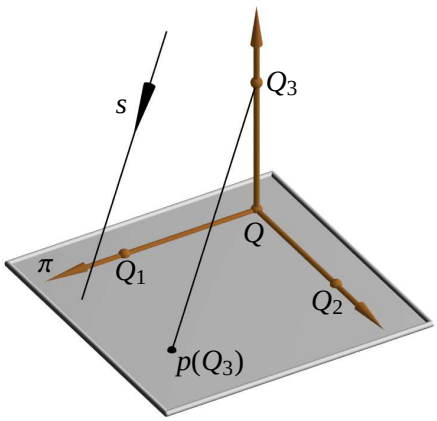


Abbildung 1.14: Orientierung von π

Damit ist jene Seite von π ausgezeichnet, in welcher der Punkt Q_3 liegt. Die Punkte dieses Halbraumes verdecken die Punkte von π. Die Punkte des anderen Halbraumes werden von π verdeckt.

1.5.3 Die Koordinatendarstellung von β stützt sich nun auf das schon bisher verwendete Rechtskoordinatensystem (U, P_1, P_2, P_3) im Raum, ein oben beschriebenes Rechtskoordinatensystem (Q, Q_1, Q_2) in der Zwischenbildebene π mit $Q = U$ und ein Rechtskoordinatensystem U', P_1', P_2' in der Bildebene ψ mit $\alpha(U) = U'$. Dabei beruht die *Orientierung der Bildebene* ψ nicht auf einer mathematischen sondern nur auf einer anschaulich-physikalischen Begründung, welche im Folgenden dargelegt wird.

Es ist in der Regel jener Halbraum von ψ ausgezeichnet, von dem aus das Bild gesehen werden *kann*, also etwa die Vorderseite eines Monitors oder die bedruckte Seite eines Plakats. In manchen Fällen, etwa bei einem Glasmosaik oder bei einer transparenten Folie für einen Overheadprojektor, muss anders vorgegangen werden. Hier ist jener Halbraum auszuwählen, von dem aus das Bild gesehen werden *soll*. Nach diesen Vorbereitungen können wir nun festsetzen:

Ein Rechtskoordinatensystem von ψ liege dann vor, wenn es so zu einem räumlichen Rechtskoordinatensystem erweitert werden kann, dass der dritte Einheitspunkt im ausgezeichneten Halbraum liegt.

Für eine betrachtende Person im ausgezeichneten Halbraum geht dann P'_1 durch eine Drehung um U' gegen den Uhrzeigersinn in P'_2 über.

1.5.4 Nachdem wir nun die Ebenen π und ψ orientiert haben, können wir die folgende Voraussetzung treffen:

Die Abbildung β soll gleichsinnig sein.

Damit schließen wir aus, dass in der Bildebene π (aus der Sicht einer betrachtenden Person) ein *spiegelverkehrtes Bild* entsteht.

Kapitel 2

Einführung in POV-Ray

2.1 Einleitung

2.1.1 *POV-Ray* ist eine Abkürzung für *persistance of vision raytracer*. Der Begriff *Raytracing* bedeutet auf Deutsch etwa *Strahlenrückverfolgung*, die Phrase *persistance of vision* kann mit *Beständigkeit des Sehens* übersetzt werden.

Das gleichnamige Programm rechnet vereinfacht gesagt Pixel für Pixel eines Bildes so aus, wie es in der Realität auf der Netzhaut eines unserer Augen entstehen würde. Die Einsatzgebiete reichen von der Architektur über Physik, Chemie bis hin zur Mathematik.

2.1.2 Das Programm POV-Ray steht im Internet unter der Adresse

`http://www.povray.org`

kostenlos zur Verfügung (Versionen für Windows, Linux und MacOS). Dort gibt es auch eine umfangreiche *Online-Hilfe*. Ferner sind ein *Handbuch* und die *technische Referenz* verfügbar. Es ist nicht unser Ziel, auf alle Feinheiten und Möglichkeiten des Programms POV-Ray einzugehen. Für einen enzyklopädischen Überblick aller Befehle sei mit Nachdruck auf die soeben genannten Quellen verwiesen.

Eine empfehlenswerte deutschsprachige Einführung ist unter

`http://www.f-lohmueller.de`

zu finden. Beeindruckende Bilder zeigt die *Hall of Fame*:

`http://hof.povray.org`

2.1.3 Wir stellen einige Eigenschaften von POV-Ray zusammen:

- Bilder können mittels Orthogonalprojektion, Zentralprojektion und anderen (nichtlinearen) Abbildungsverfahren erstellt werden.
- Eine beliebige Anzahl von Lichtquellen ist einstellbar.
- Es werden zahlreiche Grundkörper zur Verfügung gestellt, aus denen kompliziertere Szenen zusammen gestellt werden können.
- Sowohl parametrisierte als auch durch Gleichungen bestimmte Flächen sind darstellbar. Dies trifft mit Einschränkungen auch auf parametrisierte Kurven zu.

- Boolesche Operationen (Durchschnitt, Vereinigung, . . .) und Transformationen (Drehungen, Schiebungen, Affinitäten, . . .) werden unterstützt.
- Logische Operationen ermöglichen es, Verzweigungen und Schleifen zu programmieren.
- Es werden eine Reihe von mathematischen Funktionen zur Verfügung gestellt.
- Es ist in einfacher Weise möglich, Daten aus anderen Programmen zu importieren.
- Es sind eine Vielzahl an Texturen vorhanden, Farbgebung und Oberflächenbeschaffenheit werden unterstützt.
- Sichtbarkeit, Schatten und Reflexionen werden numerisch berechnet.
- Bilderfolgen können einfach erstellt werden. Daraus lassen sich anschließend Animationen anfertigen.

2.1.4 Es folgen einige Bilder, die mit POV-Ray erstellt wurden.

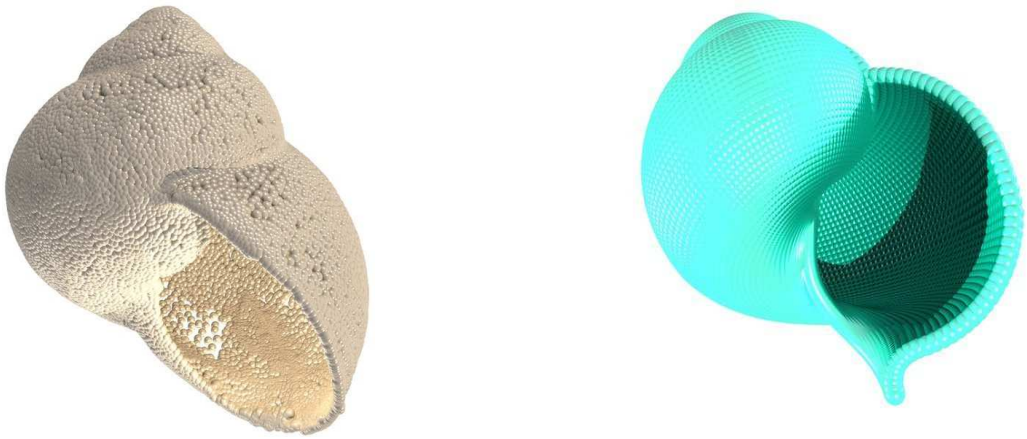


Abbildung 2.1: Weinbergschnecke (*helix pomatia*)

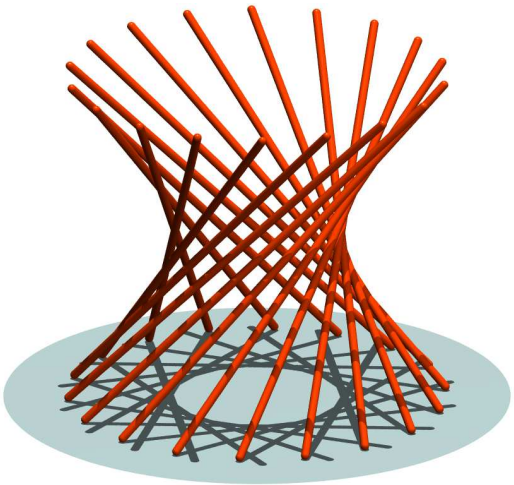


Abbildung 2.2: Geradenschar am Hyperboloid

Abbildung 2.1 zeigt links die Rekonstruktion einer *Weinbergschnecke*, deren Haus mit Hilfe eines 3D-Scanners digitalisiert wurde. Das Datenmodell besteht aus einzelnen gescannten Punkten. Zur Darstellung dienten kleine Sphären, deren Mittelpunkte diese Punkte sind. Im

rechten Bild wurde die gleiche Darstellungsweise gewählt, wobei aber die Mittelpunkte einer mathematisch definierten *Spiralfläche* angehören.

Die Abbildung 2.2 zeigt eine der beiden Geradenscharen auf einem *einschaligen Drehyperboloid* samt ihrem Schatten auf einer Ebene. Das Licht fällt dabei orthogonal zur Ebene ein. Da Geraden in POV-Ray nicht darstellbar sind, wurden sie als dünne Drehzylinder mit kleinen Kugeln an den beiden Enden modelliert.

2.2 Grundstruktur und Ausführung einer POV-Ray-Datei

2.2.1 *POV-Ray-Dateien* sind Textdateien, die vom Programm verarbeitet werden und als Ergebnis ein digitales Bild ergeben. Die Beschreibung einer *Szene* (Kamera, Lichtquellen, Objekte, Transformationen usw.) erfolgt mittels Schlüsselworten und Zahlenangaben. Groß- und Kleinschreibung werden unterschieden, überzählige Leerzeichen und leere Zeilen spielen keine Rolle. Deutsche Umlaute und andere Sonderzeichen dürfen in POV-Ray-Programmen nicht verwendet werden. Eine Ausnahme stellen Kommentare dar; vgl. dazu 2.2.3. Das Programm bearbeitet den Text von der ersten Zeile beginnend bis zur letzten Zeile.

2.2.2 Gleich zu Beginn jeder POV-Ray-Datei, also vor jedem anderen Befehl, sollte die verwendete *Version* der Programmsprache von POV-Ray stehen:

```
#version 3.7;
```

Wird diese Zeile weggelassen, so erfolgt eine Warnung. Außerdem wird POV-Ray in diesem Fall in einem *Kompatibilitätsmodus* ausgeführt, der die Verwendung von (inzwischen obsoleten) Befehlen aus früheren Programmversionen ermöglicht. Da wir ausschließlich auf die Version 3.7 zurückgreifen, sollte diese Anweisung keinesfalls fehlen.

In POV-Ray können *globale Parameter* für die Ausführung neu definiert werden. Ab Version 3.7 muss der Wert für die *Gammakorrektur* explizit festgelegt werden. Die zweite Anweisung in einer POV-Ray-Datei sollte daher lauten:

```
global_settings { assumed_gamma 1.0 }
```

Damit wird die reelle Zahl 1.0 als Wert für die Gammakorrektur definiert. An die Stelle von 1.0 kann im Prinzip jede positive reelle Zahl treten; vgl. 2.4.1 für die Schreibweise reeller Zahlen in POV-Ray. Vereinfacht gesagt gilt: Werte größer als 1.0 liefern kräftigere Farben und Werte keiner als 1.0 blasse Farben. Es besteht aber in der Regel keine Notwendigkeit vom Standardwert 1.0 abzuweichen. Falls die Anweisung fehlt, erfolgt bei der Ausführung eine Warnung.

Für einige globale Parameter gibt es Voreinstellungen, die sich bei Bedarf in ähnlicher Weise verändern lassen. Siehe dazu auch 2.2.7.

2.2.3 Hinter den Zeichen `//` können einzeilige *Kommentare* (auch mit deutschen Umlauten) von der Ausführung ausgeschlossen werden. Längere Kommentare über mehrere Zeilen können mit `/*` und `*/` eingeschlossen werden.

2.2.4 In eine POV-Ray-Datei können andere Dateien *inkludiert* werden. So wird etwa mit dem Befehl

```
#include "colors.inc"
```

eine zum POV-Ray-Paket gehörende Datei mit Definitionen verschiedener Farben eingelesen. Vgl. dazu 2.9.2.

2.2.5 In der *Windows*-Version kommt das Programm POV-Ray mit einem integrierten Editor samt graphischer Benutzeroberfläche.[1]

Der Editor bietet die üblichen Funktionen zur Erstellung und Modifikation von Texten. Etwas versteckt gibt es unter dem Menüpunkt Search den Unterpunkt Match Brace. Damit kann zu einer Klammer leicht die *zugeordnete Klammer* gefunden werden. Das funktioniert aber nicht für die beiden Zeichen „größer“ und „kleiner“, mit denen die Koordinaten von Vektoren (vgl. 2.4.2) eingeschlossen werden. Bei Eingabe der Tastenkombination Strg-Leertaste erscheint ein *Pop-Up-Menü*, aus dem Schlüsselworte ausgewählt werden können.

2.2.6 Um eine (zuvor geladene oder erstellte) POV-Ray-Datei *auszuführen*, wird zuerst aus dem Menü rechts oben ein *Bildformat* samt *Anti-Aliasing*-Einstellung gewählt. Die Ausführung erfolgt dann mit dem Befehl Run. Im *Messages-Fenster* finden sich alle Details dazu sowie allfällige Warnungen oder Fehlermeldungen.

POV-Ray kann auch unter Windows bequem über die *Kommandozeile* oder ein *Befehlsskript* ausgeführt werden. Es kann etwa mit einem Editor eine Textdatei `alles.cmd` erstellt werden, welche aus der Befehlszeile[2]

```
for %%I IN (*.pov) DO
  "C:\Programme\POV-Ray\v3.67\bin\pvengine.exe"
  /RENDER "%%I" /EXIT
```

besteht. Durch den Aufruf von `alles.cmd` werden alle POV-Ray-Dateien im aktuellen Verzeichnis ausgeführt.

2.2.7 In der *Initialisierungs-Datei* `quickres.ini` sind die wichtigsten Voreinstellungen gespeichert. Sie kann über den Menüpunkt Ini leicht editiert werden. POV-Ray erstellt Bilder standardmäßig im Format PNG (Portable Network Graphics, verlustfrei komprimiert). Diese Einstellung sollte beibehalten werden.

2.3 Syntaxprobleme

2.3.1 Als *Trennzeichen* zwischen Schlüsselworten, Zahlen oder Vektoren genügt in der Regel ein *Leerzeichen* oder ein *Zeilenwechsel*. Zusätzliche Leerzeichen und leere Zeilen haben keine Wirkung. Damit lassen sich POV-Ray-Dateien übersichtlich gliedern und durch gezielte Einrückungen von Textblöcken optisch gut strukturieren. Es ist egal, ob zwischen einem Schlüsselwort und einer nachfolgenden Klammer ein Leerzeichen steht oder nicht.

Auf einige Fälle, in denen ein Leerzeichen als Trennzeichen nicht ausreicht, wird im Folgenden kurz eingegangen.

[1] Wir beschreiben im Folgenden nur die Version für Windows. Unter *Linux* erfolgt die Steuerung des Programms POV-Ray über die Kommandozeile.

[2] Nur aus Platzgründen wurde der nachfolgende Text in drei Zeilen geschrieben. Tatsächlich muss aber alles in *in einer einzigen Zeile* stehen!

2.3.2 Bei vielen Schlüsselworten ist zusätzlich die Angabe mehrerer Zahlen oder Vektoren nötig, wobei diese meist innerhalb eines Paares von Klammern auftreten. Hier ist in der Regel zwischen je zwei Eingaben ein *Komma* zu setzen. Es ist aber in vielen Fällen auch möglich (und bequem), auf diese Kommata zu verzichten.

Um die Sache noch ein wenig undurchsichtiger zu machen, funktionieren manche Schlüsselworte ohne jedes trennendes Komma völlig problemlos, wenn nur konkrete Zahlen oder Vektoren übergeben werden. Sobald aber eine zuvor definierte Variable übergeben wird, führt das Fehlen eines Kommas zu einer Fehlermeldung. Beispiele dazu sind in 2.7.3 und 2.10.1 angegeben.

Innerhalb von Fließkommazahlen wird als Trennzeichen nach der Einerstelle ein Punkt und kein Komma geschrieben. Bei den Komponenten eines Vektors darf auf die trennenden Kommata nicht verzichtet werden. Vgl. dazu 2.4.1 und 2.4.2.

2.3.3 An einigen Stellen ist ein *Semikolon* als Trennzeichen zwingend vorgeschrieben, insbesondere bei `#version` und manchmal auch bei `#declare`. Siehe dazu 2.2.2 und 2.10.1. Fehlt dieses Trennzeichen, so kommt es immer zu einer Fehlermeldung.

2.3.4 Einige Schlüsselworte, wie etwa `#declare`, beginnen mit dem *Raute-Zeichen* `#`. Fehlt dieses Zeichen, so zeigt sich POV-Ray gelegentlich großzügig und begnügt sich mit einer Warnung an Stelle einer Fehlermeldung. Wird hingegen beim Schlüsselwort `#version` (vgl. 2.2.2) das Rautezeichen weggelassen, so ergibt das (ohne Fehlermeldung) ein schwarzes Bild.

2.3.5 In 2.4.4 wird beschrieben, wie das Produkt eines Vektors mit einer reellen Zahl aufzuschreiben ist. Dabei ist die *Reihenfolge der Faktoren* in der Regel egal, erfahrungsgemäß funktioniert aber in manchen Fällen nur eine der beiden Schreibweisen.

2.4 Mathematik

2.4.1 *Reelle Zahlen* werden in POV-Ray wie üblich in Dezimalschreibweise als Fließkommazahlen angegeben. Als Trennzeichen ist bei Bedarf ein *Dezimalpunkt* zu verwenden, etwa `4.46`, `-1.12` oder `433`. Als vordefinierte Konstante steht die *Zahl* π zur Verfügung; sie wird als `pi` eingegeben und hat den Näherungswert 3.14159265358979323846.

2.4.2 *Vektoren* werden in gespitzte Klammern eingeschlossen, wobei die Koordinaten durch Kommata getrennt werden. Zum Beispiel ist

```
<1,4,3>
```

ein Vektor aus $\mathbb{R}^3$. Für einige Vektoren gibt es reservierte Namen, die nicht verändert werden können. Die Vektoren der *kanonischen Basis* von $\mathbb{R}^2$ heißen `u` und `v`. In $\mathbb{R}^3$ sind `x`, `y`, `z` als Namen für die kanonischen Basisvektoren reserviert. Vektoren aus $\mathbb{R}^4$ kommen als Daten für diverse Einstellungen in Frage. In diesem Fall ist die kanonische Basis mit `x`, `y`, `z`, `t` ansprechbar. An Hand der Zuordnung entscheidet POV-Ray selbst, ob `x` für `<1,0,0>` oder für `<1,0,0,0>` steht. Für den *Nullvektor* kann unabhängig von der Dimension des Raumes kurz `0` geschrieben werden.

2.4.3 In 2.10.1 wird erklärt, wie *Variablen* mit Hilfe des Schlüsselwortes `#declare` mit reellen Zahlen, Vektoren und anderen Größen belegt werden können. Die Verwendung von Variablen wird nachdrücklich empfohlen. Sie tragen einerseits zur besseren Lesbarkeit von POV-Ray-Dateien bei und ermögliche andererseits die einfache Modifikation von Angabedaten.

2.4.4 Wir stellen die Symbole für die wichtigsten *Rechenoperationen* und einige *Funktionen* zusammen:

- Grundrechnungsarten: `A+B, A-B, A*B, A:B` oder `A/B`
- Potenzen, Wurzeln: `pow(A,B)` (ergibt A^B)
- Elementare Funktionen: `sin(A)`, `ln(A)`, `asin(A)`, `exp(A)`, `cosh(A)`, ...
- Addition und Subtraktion von Vektoren: `<1,2,1>+x, <3,4.8>-<5,3>`
- Skalar mal Vektor: `3*<1,2,3>` oder[3] `<1,2,3>*3`
- Länge (euklidische Norm) eines Vektors: `vlength(V)`
- Normieren eines Vektors: `vnormalize(V)`
- Skalarprodukt von Vektoren: `vdot(V,W)`
- Kreuzprodukt von Vektoren aus $\mathbb{R}^3$: `vcross(V,W)`

2.4.5 Die zur kanonischen Basis von $\mathbb{R}^2$ *duale Basis*, also die zwei *Koordinatenformen*, werden mit `.u`, `.v` bezeichnet. Um etwa die zweite Koordinate eines Vektors aus $\mathbb{R}^2$ zu erhalten kann

```
<1.21,4.18>.v
```

geschrieben werden. Als Ergebnis tritt die reelle Zahl 4.18 auf. Beachte, dass das Funktionssymbol hier nach dem Vektor geschrieben werden muss.[4] In $\mathbb{R}^3$ heißen die Koordinatenformen `.x`, `.y` und `.z`. Für Vektoren aus $\mathbb{R}^4$ können ebenfalls `.x`, `.y` und `.z` verwendet werden; die vierte Koordinate eines Vektors wird mit `.t` erhalten.

2.5 Grundobjekte

2.5.1 Wir stellen einige wichtige *Grundobjekte* vor. Aus diesen können dann kompliziertere Objekte zusammengesetzt werden. Zur Illustration der Syntax werden hier immer konkrete Vektoren und Zahlen eingesetzt. An der durch Punkte markierte Stelle können noch Farbe, Oberflächenstruktur oder eine anzuwendende Transformation hinzugefügt werden.

- *Quader:* `box {<0,0,0>, <3,2,5> ...}`
  Die beiden Punkte bestimmen gegenüberliegende Ecken eines achsenparallelen Quaders.
- *Sphäre:* `sphere {<3,2,-5>, 5 ...}`
  Der Punkt ist die Mitte, die reelle Zahl der Radius.

[3] Vgl. den Hinweis in 2.3.5.

[4] Es wird also die sogenannte *umgekehrte polnische Notation* verwendet.

- *Drehzylinder:* `cylinder {<1,2,4>, <4,-2,3>, 2 ...}`
  Die beiden Punkte geben die Mittelpunkte von Basis- und Deckkreis an, die reelle Zahl den Radius.
  Der Basis- und der Deckkreis werden als Kreisscheiben ausgeführt. Der Zusatz `open` liefert einen offenen Zylinder.

- *Drehkegel(stumpf):* `cone {<1,2,4>, 3, <3,3,1>, 4 ...}`
  Die Angabe erfolgt wie beim Drehzylinder, wobei aber nach jedem Kreismittelpunkt der betreffende Radius kommt. Wird einer der Radien gleich null gesetzt, so entsteht die Kegelspitze.
  Der Zusatz `open` liefert einen offenen Kegel(stumpf).

- *Torus:* `torus {8, 2 ...}`
  Die erste Zahl ist der Radius des Mittenkreises. Das ist jener Kreis, welcher von den Mittelpunkten aller Meridiankreise gebildet wird. Die zweite Zahl gibt den Radius der Meridiankreise an. Die Drehachse des Torus ist die y-Achse, der Mittelpunkt des Torus ist der Ursprung.[5]

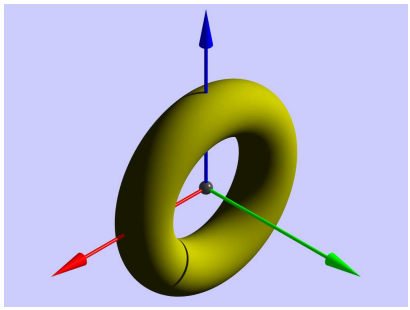


Abbildung 2.3: Torus mit Koordinatensystem

- *Prisma:* `prism {-1, 4, 3, <0,1>, <5,0>, <6,8> ...}`
  Das Basis- und das Deckpolygon liegen in Ebenen y = const. Die ersten beiden Zahlen sind die diese beiden Konstanten. Es folgt die Anzahl der Punkte am Basispolygon gefolgt von den Punkten des Basispolygons, wobei nur die x und z-Koordinaten angegeben werden. Das Polygon wird automatisch geschlossen. Die Prismenkanten sind parallel zur y-Achse.
  Der Zusatz `open` liefert ein offenes Prisma.

2.5.2 Die Liste in 2.5.1 umfasst nur räumliche Objekte. Nachfolgend geben wir eine Auswahl an *ebenen Objekten* an:

- *Dreieck:* `triangle {<1,2,0>, <1,0,2>, <1,1,3> ... }`
  Die drei Punkte sind die Ecken des gefüllten Dreiecks.

- *Polygon:* `polygon {4, <0,0>, <0,2>, <1.5,2>, <1,0> ... }`
  Die erste Zahl gibt die Anzahl der Ecken des gefüllten Polygons an. Es folgen die Eckpunkte selbst, wobei diese in der Ebene $z = 0$ liegen und nur die ersten beiden Koordinaten angegeben werden.

[5] In Abbildung 2.3 werfen die y- und die z-Achse ihren Schatten auf den Torus.

- *Ebene:* `plane {<2,1,1>, 2 ... }`
  Der Vektor steht auf die Ebene orthogonal, die reelle Zahl gibt den orientierten Abstand (im Sinne des gegebenen Vektors) vom Ursprung an.
  Ebenen sind unbegrenzt!

2.5.3 Zum Abschluss soll noch auf das Schlüsselwort

```
object { ... }
```

hingewiesen werden. Es ist eine Art *Verpackung* für jenes Objekt, das innerhalb der Klammern auftritt. Beispiele, in denen `object{ ... }` verwendet werden muss, geben wir in 2.10.1 und 5.1.4.

2.6 Boolesche Operationen

2.6.1 Aus den in 2.5 definierten Grundobjekte können durch *Boolesche Operationen* neue Objekte generiert werden (Abbildung 2.4):

- *Vereinigung:* `union {sphere { ... } cylinder { ... } ... }`
  Die angegebenen Objekte werden vereinigt. Es können auch gemeinsame Eigenschaften, etwa eine Farbe hinzugefügt werden.
- *Differenz:* `difference {sphere { ... } cylinder { ... } ... }`
  Das zweite Objekt wird aus dem ersten ausgeschnitten. Die Schnittfläche dabei wird geschlossen!
- *Durchschnitt:* `intersection {sphere { ... } cylinder { ... } ... }`
  Die angegebenen Objekte werden geschnitten.

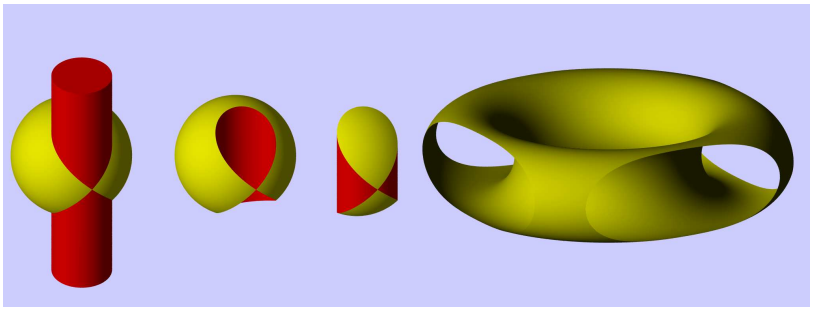


Abbildung 2.4: Vereinigung, Differenz, Durchschnitt und Clipping

2.6.2 Zu erwähnen ist hier auch der *Clipping*-Befehl. Vgl. Abbildung 2.4 ganz rechts. So ermöglicht etwa

```
union { ... clipped_by {sphere {<0,0,3>,1} } }
```

das Abschneiden eines komplizierten Objekts, das als Vereinigung gegeben ist, mit einer Kugel (ohne Farbe und Oberflächenstruktur). Analog kann mit einem Prisma oder anderen Grundkörpern abgeschnitten werden. Beim Clipping mit einer Ebene gilt diese als Grenze jenes Halbraumes, in welches der Normalvektor der Ebene *nicht* zeigt. So liefert etwa

```
sphere {0, 1 clipped_by {plane {<0,0,-1>,0} }
```

den im Halbraum $z \geq 0$ liegenden Teil der Einheitssphäre. Beim Clipping wird das abgeschnittene Objekt entlang des Schnitts nicht geschlossen!

2.7 Transformationen

2.7.1 Objekte können bei ihrer Definition durch *Transformationen* verändert werden. Wir erklären das an Hand von Beispielen.

- *Translation:* `torus {4,1 translate <2,3,-1>}`
  Der Torus wird aus seiner Standardlage unter der Translation zum Schiebvektor $(2, 3, -1)$ verschoben.
- *Drehung:* `box {<1,2,3>, <4,5,6> rotate <20,-90,30>}`
  Der Quader wird zuerst um die x-Achse um 20°, dann um die y-Achse um −90° und schließlich um die z-Achse um 30° gedreht.[6] Das Ergebnis ist dann eine Drehung um eine gewisse Gerade durch den Ursprung um einen gewissen Winkel. Genaueres lässt sich meist nur mit Hilfe der zugehörigen Drehmatrix sagen.
  Die oben auftretenden Drehwinkel sind nach folgender Regel mit einem Vorzeichen versehen: Unter einer Drehung um die x-Achse durch einen Winkel von 90° geht die positive y-Achse in die positive z-Achse über. Gleiches gilt bei zyklischer Vertauschung von x, y, z. Soll etwa nur um die z-Achse um 45° gedreht werden, so kann das als `rotate 45*z` geschrieben werden.
- *Skalierung längs der Koordinatenachsen:* `sphere { ... scale <2,4,3>}`
  Die Sphäre wird unter einer linearen Bijektion transformiert, welche die kanonische Basis als Eigenvektoren besitzt. Die Koordinaten des Vektors geben der Reihe nach die drei Eigenwerte (Verzerrungen) an. Aus einer Sphäre entsteht so ein *Ellipsoid*.
- *Affinität:* Eine Abbildung
  $$\alpha : \mathbb{R}^3 \to \mathbb{R}^3 : \boldsymbol{x} \mapsto \boldsymbol{x} \cdot A + \boldsymbol{b}$$
  mit $A \in \mathrm{GL}_3(\mathbb{R})$ und $\boldsymbol{b} \in \mathbb{R}^3$ wird durch

```
transform { matrix <A_{11},A_{12},A_{13},
                    A_{21},A_{22},A_{23},
                    A_{31},A_{32},A_{33},
                    b_1,b_2,b_3}> }
```

  beschrieben. Die ersten neun Einträge der Matrix ergeben (zu je dreien gelesen) die Bilder der kanonischen Basisvektoren, die letzten drei Zahlen ergeben den Vektor $\boldsymbol{b}$ der Schiebung.

2.7.2 Mehrere Transformationen lassen sich *zusammensetzen*, indem die Befehle hintereinander geschrieben werden. Bei der Zusammensetzung wird von links nach rechts vorgegangen.

Allgemein kann `transform { ... }` eine beliebige (endliche) Folge von Transformationen aufnehmen. Damit lässt sich etwa eine Transformation mit `#declare` (vgl. 2.10.1) definieren. Das Schlüsselwort `invers` gestattet es, die *inverse* Transformation anzuwenden.

[6] Die Abfolge dieser Drehungen ist wichtig, da die Gruppe SO_3 der Drehungen um den Ursprung nicht kommutativ ist.

2.7.3 Mit dem Befehl

```
#include "transforms.inc"
```

können weitere Transformationen geladen werden. Außerdem lassen sich dann Transformationen auch auf Vektoren und nicht nur auf Objekte anwenden. So wird etwa durch

```
vtransform( <3,5,4>, transform {rotate x*30} )
```

der Vektor $(3, 5, 4)$ um die x-Achse durch $30°$ gedreht.

Zwischen `<3,5,4>` und `transform rotate x*30` müsste kein *Komma* gesetzt werden. In manchen Situationen darf auf dieses Komma aber nicht verzichtet werden. Vgl. dazu die Hinweise in 2.3.2.

2.8 Kamera, Lichtquellen und Hintergrundfarbe

2.8.1 Die Festlegung der Art der *Projektion* erfolgt mit dem obligatorischen Schlüsselwort `camera`. Wir besprechen hier zunächst nur die Angabe einer Orthogonalprojektion. Durch

```
camera { orthographic location <3,2,6>  look_at <0,0,0> }
```

wird die Verbindung der Punkte $(3, 2, 6)$ und $(0, 0, 0)$ als Sehstrahl definiert, wobei der erste Punkt den zweiten verdeckt. Die Lage der Zwischenbildebene ist belanglos. Jede zum gegebenen Sehstrahl orthogonale Ebene liefert ja bis auf Verschiebung dasselbe Zwischenbild im Raum. Auf Grund verschiedener interner Vorgaben, die sich verändern ließen, ist auch schon mitbestimmt, wie das Zwischenbild im endgültigen Bild (am Bildschirm) erscheint.

- Der durch `look_at` beschriebene Punkt H erscheint in der Mitte des Bildes.
- Der Abstand des durch `location` bestimmten Punktes O vom Punkt H regelt die *Vergrößerung*. Je größer der Abstand ist, desto kleiner kleiner erscheinen die Objekte im endgültigen Bild.
- Sofern die y-Achse nicht projizierend ist, weist sie im Bild nach oben. Bei projizierender y-Achse zeigt die z-Achse nach oben.

Wird das Schlüsselwort `orthographic` weggelassen, so wird eine *Zentralprojektion* erstellt, welche den Punkt O als Augpunkt und den Punkt H als Hauptpunkt besitzt. Dies wird in Kapitel 3 ausführlich besprochen.

2.8.2 Bei der Darstellung von Objekten unter einer Orthogonalprojektion treten in POV-Ray gewisse Einschränkungen auf, die sich aus der Wahl der Punkte O und H ergeben. Sei π_v die zur Geraden OH orthogonale Ebene durch den Punkt O.

- Es werden nur jene Teile von Objekten dargestellt, die auf derselben Seite von π_v wie der Punkt H liegen.[7]

[7] Es kommt gelegentlich auch vor, dass Objekte, die der Ebene π_v zu nahe kommen, gar nicht dargestellt werden. Vom mathematischen Standpunkt ist die Ebene π_v für eine Orthogonalprojektion irrelevant. Sie wird aber von POV-Ray bei der Berechnung des Bildes verwendet. Vgl. dazu auch 3.2.3, wo die (ebenfalls mit π_v bezeichnete) Verschwindungsebene einer Zentralprojektion beschrieben wird.

- Es werden nur jene Teile von Objekten dargestellt, die innerhalb eines vierkantigen Prismas liegen, dessen Kanten Sehgeraden sind. Die Basis dieses Prismas ist jenes Rechteck in π, das den Rand des elektronischen Bildes ergibt.

Die erste Einschränkung lässt sich umgehen, indem die Punkte O und H gleichzeitig entlang der Geraden OH verschoben werden. Auf diese Weise ändert sich die Orthogonalprojektion nicht.

2.8.3 Es ist zu beachten, dass POV-Ray (entgegen den einschlägigen Normen) im Raum ein *kartesisches Linkssystem* verwendet: Die x-Achse zeigt nach rechts, die y-Achse nach oben und die z-Achse nach hinten. Solange nur in $\mathbb{R}^3$ gerechnet wird, ist davon nichts zu merken. Sobald wir aber mit dem Programm POV-Ray ein Bild erstellen, werden die oben genannten Vorgaben unmittelbar sichtbar.

Um möglichst einfach zu der in Kapitel 1 genannten Situation zu kommen, kann folgende Strategie eingeschlagen werden.

- Am Beginn der POV-Ray-Datei wird folgende Transformation definiert:

```
#declare Trafo =
transform { matrix <1,0,0, 0,0,1, 0,1,0, 0,0,0>} }
```

  Das ist die Spiegelung an einer Ebene durch die x-Achse, welche die y-Achse mit der z-Achse vertauscht. Mehr zum Befehl `#declare` ist in 2.10.1 zu finden.

- Soll eine Kamera etwa durch die beiden Punkte $(3,5,4)$ und $(1,2,3)$ festgelegt werden, so wird die Angabe in folgender Weise modifiziert:

```
camera { orthographic location vtransform( <3,5,4>, Trafo )
                      look_at vtransform( <1,2,3>, Trafo )
                      transform Trafo }
```

  Der Befehl `transform Trafo` am Ende der `camera` Anweisung ergibt eine Kamera, die (aus der Sicht von POV-Ray) *spiegelverkehrte Bilder* liefert. Dabei ist es notwendig, die *Spiegelbilder* der Punkte $(3,5,4)$ und $(1,2,3)$ als Angabe bei `location` und `look_at` zu verwenden. Dies geschieht durch die beiden `vtransform` Befehle. Insgesamt kommen die gespiegelten Punkte durch die anschließende Anweisung `transform Trafo` wieder an die richtige Position.[8]

Im digitalen Bild ist von alledem nichts zu bemerken. *Es sieht einfach so aus, als würde POV-Ray ein Rechtssystem verwenden, so wie es in Kapitel* 1 *betrachtet wurde.* In der Beschreibung des Koordinatensystems aus 2.8.1 tauschen nun die Bilder der y- und der z-Achse ihre Rollen. Alle weiteren Angabestücke der Szene werden anschließend einfach in (x,y,z)-Koordinaten beschrieben.[9]

[8] Hier geht ein, dass eine Spiegelung ihre eigene inverse Abbildung ist.

[9] Wird eine Kamera einer Transformation unterworfen und damit ein Objekt abgebildet, so entsteht ein gewisses Bild. Dasselbe Bild lässt sich auch so gewinnen: Die Kamera wird nicht transformiert. Dafür wird das Objekt der *inversen Transformation* unterworfen.

2.8.4 *Lichtquellen* und *Lichtstrahlen* sind in POV-Ray immer unsichtbar. Erst wenn Licht auf eine Oberfläche auftrifft, kann dies zu einem visuell erkennbaren Effekt führen. Dabei kommt es sowohl auf die Beschaffenheit der Lichtquelle als auch auf die Beschaffenheit der Oberfläche an. Mehr zum Thema Oberflächen findet sich in Abschnitt 2.9.

Ohne weiteres Zutun ist ein diffuses rein weißes *Umgebungslicht* durch

```
global_settings { ambient_light <1,1,1> }
```

bereits vorgegeben. Wird diese Zeile mit einem anderen Farbvektor (vgl. 2.9.2) in eine POV-Ray-Datei eingefügt, so wird der Standardwert überschrieben. Dieses Umgebungslicht wirkt zusammen mit dem Oberflächenparameter `ambient` (vgl. 2.9.5).

Eine *punktförmige Lichtquelle* dient der *direkten Beleuchtung*. Sie kann etwa durch

```
light_source { <-50, 50, 50>  color White shadowless }
```

erklärt werden. Der Vektor gibt jenen Punkt an, wo sich die Lichtquelle befindet. Es wird weißes Licht in alle Richtungen gleichmäßig emittiert, wobei die Beleuchtungsstärke mit der Entfernung *nicht abnimmt*. Der Farbwert `White` ist in der Datei `colors.inc` definiert, die zuvor gemäß 2.2.4 geladen werden muss. Für die Definition von Farben wird auf 2.9.2 verwiesen. Der Zusatz `shadowless` kann auch weggelassen werden. Er bewirkt nur, dass *schattenloses Licht* entsteht. Es gibt daneben noch eine Fülle von Parametern, mit denen die Beschaffenheit von Lichtquellen beeinflusst werden kann.

In vielen Fällen ist es zweckmäßig, mehrere Lichtquellen zu definieren. So wurden etwa in Abbildung 4.4 auf Seite 60 zwei verschiedenfarbige Lichtquellen verwendet, um einen plastischen Eindruck zu bewirken.

2.8.5 Trotz Lichtquelle erscheint der Hintergrund in POV-Ray schwarz, sofern nicht eine andere *Hintergrundfarbe* definiert wird. Vgl. Abbildung 2.5. Durch

```
background { color Yellow }
```

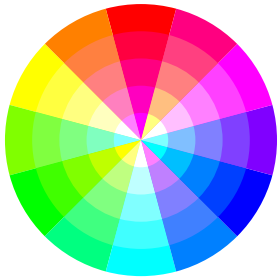
wird die Hintergrundfarbe zum Beispiel auf gelb gesetzt. Dabei greifen wir auch hier auf einen in `colors.inc` definierten Farbwert zurück. Vgl. dazu 2.9.2.

2.9 Eigenschaften von Oberflächen

2.9.1 Die Oberflächen von in POV-Ray definierten Objekten können mit Eigenschaften versehen werden, die nicht mathematischer Natur sind, sondern physikalischen Eigenschaften realer Objekte entsprechen. Bei der Berechnung des Bildes eines Objekts spielen dessen Oberflächeneigenschaften, die Oberflächeneigenschaften anderer Objekte, die gewählten Lichtquellen und anderes mehr zusammen.

2.9.2 *Farben* werden in POV-Ray additiv aus den Grundfarben Rot, Grün, Blau (r,g,b) zusammengesetzt. Die Angabe erfolgt in Form eines Vektors, etwa als `color rgb <0.3,1,0.2>`. Die Koordinaten eines Farbvektors können auch größer als 1 sein. Um die zuvor definierte Farbe heller zu machen, kann `color rgb 1.5*<0.3,1,0.2>` geschrieben werden. (Etwa für eine starke Lichtquelle.) Wird für ein Objekt keine Farbe definiert, so erscheint es in der *Standardfarbe* schwarz.

0	0	0	schwarz	schwarz
1	0	0	rot	rot
0	1	0	grün	grün
0	0	1	blau	Blau
1	1	0	gelb	gelb
1	0	1	violett	violett
0	1	1	türkis	türkis
1	1	1	weiß	



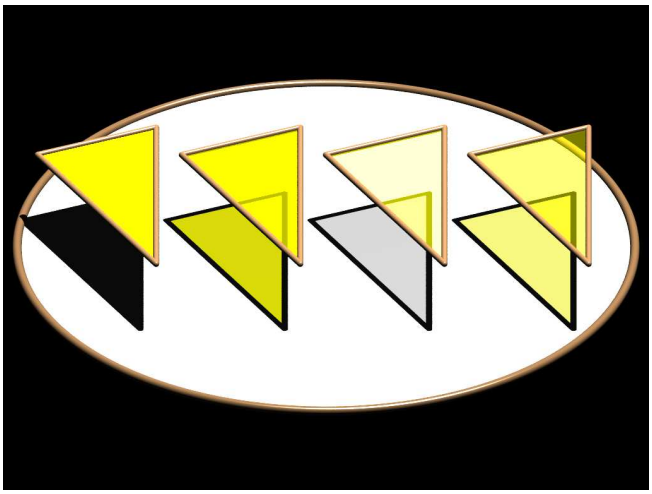
Es ist empfehlenswert, die Datei `colors.inc` gemäß 2.2.4 einzulesen. In dieser Datei sind eine Reihe von fein abgestimmten Farben definiert, die dann über Schlüsselworte angesprochen werden können. Die Verwendung der Grundfarben, etwa rot (`color rgb <1,0,0>`), führt für große Flächen meist zu einer wenig ansprechenden Farbgebung. Das zeigen die Abbildungen 4.23 und 4.24.

2.9.3 Die Definition der *Farbeigenschaften* eines Objekts erfolgt über das Schlüsselwort `pigment`. Hier ist ein einfaches Beispiel:

```
sphere { 3*y, 2 pigment {color Yellow transmit 0.5} }
```

Es wird eine Sphäre mit Mittelpunkt $(0, 3, 0)$ und Radius 2 festgelegt, deren Oberfläche gelb ist und eine *Transparenz ohne Farbfilter* von 50% aufweist.

2.9.4 Es gibt neben `color rgb <.,.,.>` noch eine andere Möglichkeit zur verfeinerten Definition von Farbe und *Lichtdurchlässigkeit*. Vgl. Abbildung 2.5.



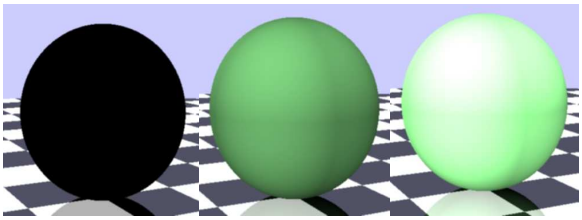
`rgb <1,1,0>`, `rgbf <1,1,0,0.7>`, `rgbt <1,1,0,0.7>`, `rgbft <1,1,0,0.4,0.4>`
Abbildung 2.5: Farbe und Lichtdurchlässigkeit

- *Transparenz mit Farbfilter:* `color rgbf <.,.,.,.,.>`
  Die vierte Koordinate des Vektors gibt in Prozenten an, wie viel Licht gefiltert durch die Oberfläche hindurchgeht. Alternativ kann auch das Schlüsselwort `filter` mit `color rgb <.,.,.>` verwendet werden.

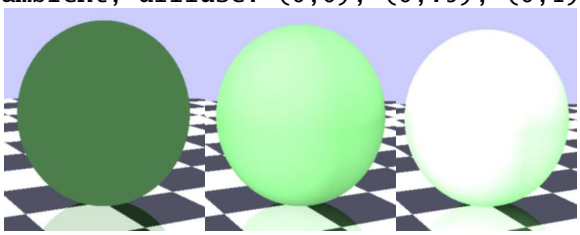
- *Transparenz ohne Farbfilter:* `color rgbt <.,.,.,.>`
  Die vierte Koordinate des Vektors gibt in Prozenten an, wie viel Licht ungefiltert durch die Oberfläche hindurchgeht. Sie entspricht dem zuvor besprochenen Wert bei `transmit`.

- *Transparenz mit und ohne Farbfilter:* `color rgbft <.,.,.,.,.>`
  Damit lassen sich beide Arten von Transparenz mischen.

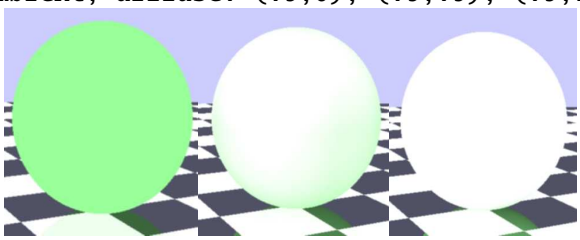
2.9.5 Wird nur ein Teil eines Objekts durch die Lichtquellen ausgeleuchtet, so gibt es eine Licht- und eine Schattenseite. Es ist aber meist unrealistisch, den unbeleuchtete Teil als völlig schwarz darzustellen. POV-Ray gestattet es, die *indirekte Beleuchtung* eines Objekts zu steuern. Wir erklären die zugehörigen Befehle an Hand von Beispielen und den Figuren in Abbildung 2.6.



`ambient, diffuse: (0,0), (0,.5), (0,1)`



`ambient, diffuse: (.5,0), (.5,.5), (.5,1)`



`ambient, diffuse: (1,0), (1,.5), (1,1)`

Abbildung 2.6: Verschiedene Werte für `ambient, diffuse`

- *Helligkeit durch Umgebungslicht:* `ambient 0.6`
  In POV-Ray gibt es standardmäßig ein weißes Umgebungslicht, das von den gewählten Lichtquellen und anderen Objekten *nicht* abhängt. Vgl. dazu 2.8.4. Durch `ambient 0.6` wird erreicht, dass eine Oberfläche überall 60% des Umgebungslichts reflektiert. Damit lassen sich Schatten aufhellen.

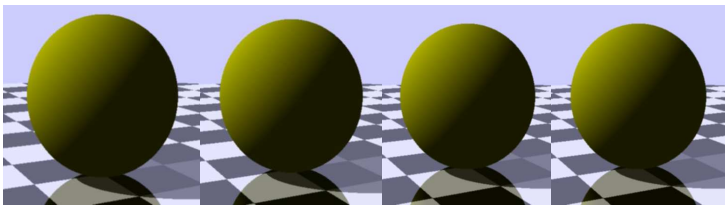
- *Farbe durch Umgebungslicht:* `ambient color rgb <0.6,0.4,0>`
  Alternativ kann der Farbton des reflektierten Umgebungslichts festgesetzt werden. Statt `ambient 0.6` könnte auch `ambient color rgb <0.6,0.6,0.6>` geschrieben werden.

- *Helligkeit durch direkte Beleuchtung:* `diffuse 0.6`
  Dieser Prozentsatz wirkt nur in den durch die Lichtquellen beleuchteten Teilen der Oberfläche.

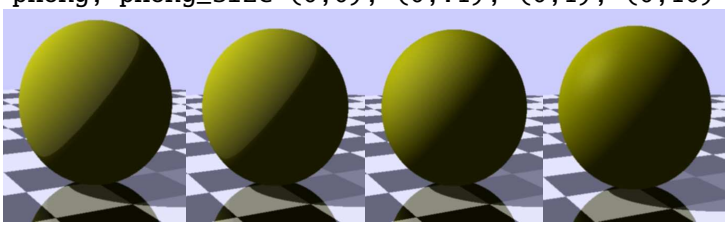
Werden die obigen Werte angegeben, so überschreiben diese gewisse Standardwerte von POV-Ray. Diese sind – im Gegensatz zur Standardfarbe schwarz – nicht alle gleich 0.0.

2.9.6 Bei der Beleuchtung von Objekten können *Glanzstellen* auftreten, die sich folgendermaßen steuern lassen (Abbildung 2.7):

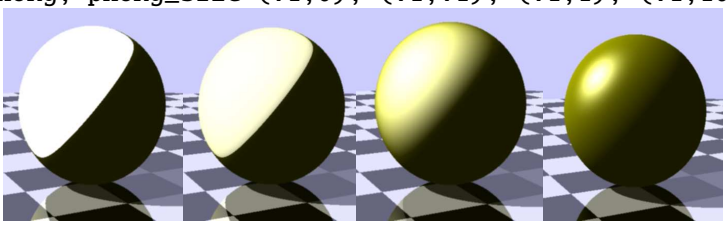
- *Stärke der Glanzstellen:* `phong 0.9`
  Die Berechnung der Stärke von Glanzpunkten erfolgt in Abhängigkeit vom angegebenen Faktor: Je größer der Wert ist, desto leuchtender wird der Glanzpunkt.
- *Größe der Glanzstellen:* `phong_size 20`
  Große Werte liefern kleine Glanzstellen.
- Glanzpunkte lassen sich auch mit `specular` erzeugen; dies funktioniert ähnlich wie `phong`.



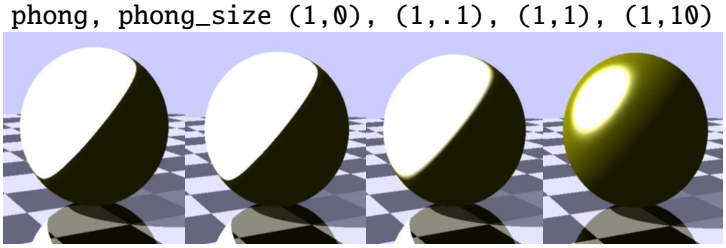
`phong, phong_size (0,0), (0,.1), (0,1), (0,10)`



`phong, phong_size (.1,0), (.1,.1), (.1,1), (.1,10)`



`phong, phong_size (1,0), (1,.1), (1,1), (1,10)`



`phong, phong_size (10,0), (10,.1), (10,1), (10,10)`

Abbildung 2.7: Verschiedene Werte für `phong, phong_size`

Glanzpunkte treten nur bei Lichtquellen auf, die nicht als `shadowless` deklariert wurden. Falls Schatten eher störend wirken und dennoch Glanzpunkte auftreten sollen, empfiehlt sich die Definition einer einzigen Lichtquelle mit Schattenwurf und mehrerer schattenloser Lichtquellen. Letztere hellen die auftretenden Schatten wieder auf. Alternativ kann auch eine Lichtquelle mit Schattenwurf so definiert werden, dass die Lichtstrahlen mit den Sehstrahlen übereinstimmen.

2.9.7 Wir geben noch einige weitere Möglichkeiten zur Gestaltung von Oberflächen an. Vgl. dazu die Abbildungen 2.8, 2.9 und 2.10.

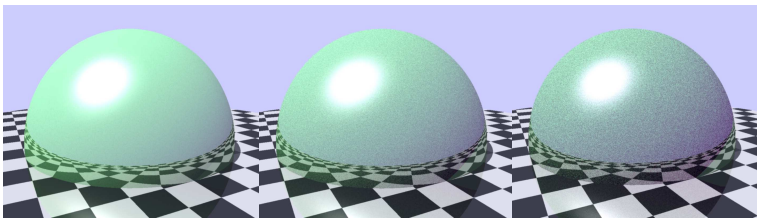


Abbildung 2.8: Rauhe Oberflächen

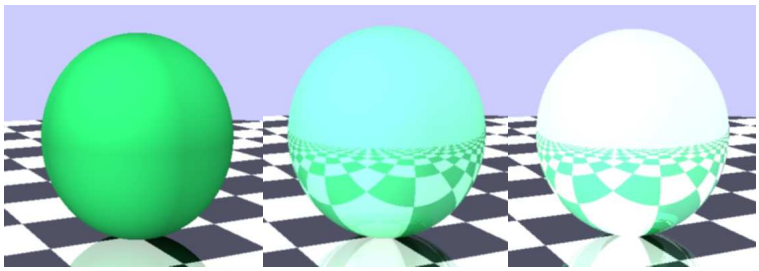


Abbildung 2.9: Reflektierende Oberflächen

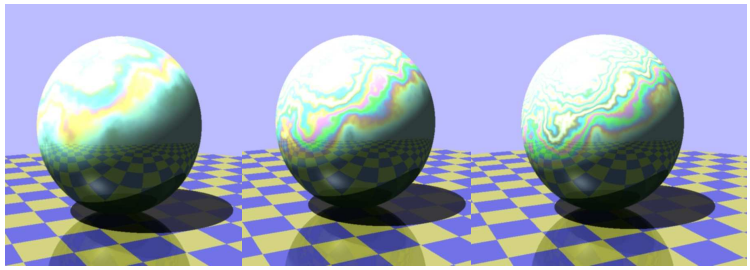


Abbildung 2.10: Irideszenz

- *Körnigkeit:* `crand 0.5`
  So können zufällig strukturierte Oberflächen wie Beton oder Sand simuliert werden. Der Befehl stammt aus früheren Versionen von POV-Ray und gilt inzwischen als veraltet. Von einer Verwendung wird daher abgeraten.
- *Verspiegelungsgrad:* `reflection 0.2`
  Je höher der Wert ist, desto stärker wird die Reflexion. Der Standardwert ist 0; bei 1 liegt ein perfekter Spiegel vor.
- *Metallischer Glanz:* `metallic`
  Dieses Schlüsselwort nimmt keinen Parameter auf. Es wirkt nur gemeinsam mit `phong` oder `specular` und gibt den Glanzpunkten ein metallisches Aussehen.

- *Brillanz:* `brilliance 5`
  Das Schlüsselwort regelt den Übergang zwischen Licht und Schatten. Höhere Werte vergrößern die hellen Teile und vermitteln so mehr Glanz.

- *Irideszenz:* `irid {0.25 thickness 0.5 turbulence 2 }`
  Damit kann die die Interferenz an dünnen Schichten (Perlmutt, Fette, Öle, Seifenblasen) simuliert werden. Der erste Wert ist eine Prozentzahl, der nächste regelt die Dicke des „Ölfilms“ und mit dem dritten Wert kann die Stärke der Turbulenzen geregelt werden.

2.9.8 Die Definition der zuvor beschriebenen Oberflächeneigenschaften eines Objekts erfolgt über das Schlüsselwort `finish`. Hier ist ein einfaches Beispiel:

```
sphere { 3*y, 2 pigment {color Yellow transmit 0.5}
                finish {ambient 0.3 phong 0.6} }
```

Die schon in 2.9.3 betrachtete Sphäre wird zusätzlich mit Oberflächeneigenschaften versehen.

2.9.9 Es gibt noch eine Reihe von Gestaltungsmöglichkeiten für Oberflächen. Auf einige davon sei hier wenigstens hingewiesen; siehe auch Abbildung 2.11:

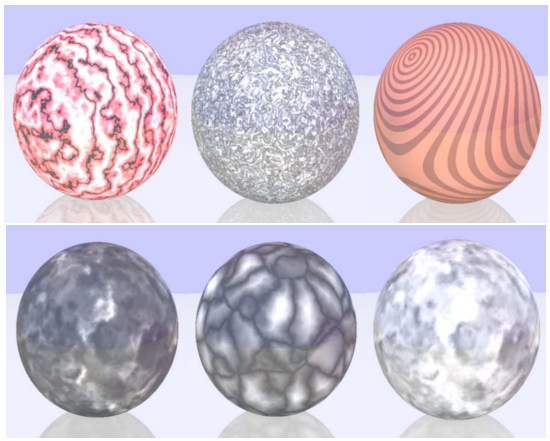


Abbildung 2.11: Texturen und andere Oberflächenstrukturen

- *Textur:* `texture { ...}`
  Nach dem Laden der betreffenden Datei mit dem Befehl `#include "textures.inc"` stehen zahlreiche vordefinierte Oberflächenstrukturen zur Verfügung: `marble, granite, wood, dents, crackle, bumps, ...`

- *Schwankungen der Flächennormalen:* `normal { ...}`
  Damit können Schwankungen der Oberfläche simuliert werden, die aber nur bei der Berechnung des Bildes eingehen. Die Geometrie der Fläche bleibt gleich.

Beispiel 2.9.10 Mit der folgenden POV-Ray-Datei wurde Abbildung 1.1 erstellt. Dabei wurde nicht versucht, alles so kurz wie möglich zu schreiben. Für die `#declare`-Anweisung wird auf 2.10.1 verwiesen.

```
// Hans Havlicek Visualisierung

#version 3.7;
global_settings { assumed_gamma 1.0 }

#include "colors.inc"        // Vordefinierte Farben laden
#include "transforms.inc"    // Vordefinierte Transformationen laden

// Eigene Definitionen xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

// Dicke der Koordinatenachsen
#declare dicke = 0.04;
// Länge der Pfeile
#declare laenge = 4;
// Länge der Pfeilspitzen
#declare pfeil = 1;
// Dicke der Pfeile an der Basis und Dicke des Ursprungs
#declare basis = 4*dicke;
// Trafo auf Rechtssystem
#declare Trafo = transform {matrix <1,0,0, 0,0,1, 0,1,0, 0,0,0>}
// eine sehr einfache Oberflächenstruktur, phong wirkt nicht, da
// Lichtquelle shadowless
#declare MeinFinish = finish { phong 0.9 ambient 0.5 }

// Grundeinstellungen xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
// Kamera
camera {orthographic location vtransform( <3,5,4> Trafo )
                     look_at  vtransform( <0,0,0> Trafo )
                     transform Trafo }
// Lichtquelle
light_source { <-50, 50, 50> color White shadowless }

// Hintergrundfarbe des Bildes
background { color White }

// Das Gesamtobjekt, in gleichfarbige Objekte gegliedert
// Jeder Pfeil besteht aus einem Zylinder und einem Kegel,
// der Ursprung ist eine Sphäre.
union { // Koordinatenachsen
        union { cylinder {<0,0,0>,<laenge,0,0>, dicke}
        cone {<laenge,0,0>, basis , <laenge+pfeil,0,0>, 0}
        pigment {color Red } finish {MeinFinish} }
        union { cylinder {<0,0,0>,<0,laenge,0>, dicke}
        cone {<0,laenge,0>, basis, <0,laenge+pfeil,0>, 0}
        pigment {color Green } finish {MeinFinish} }
```

```
        union { cylinder {<0,0,0>,<0,0,laenge>, dicke}
                cone {<0,0,laenge>, basis, <0,0,laenge+pfeil>, 0}
                pigment {color Blue} finish {MeinFinish} }
        // Ursprung
        sphere {<0,0,0> basis pigment {color rgb 0.3}
                              finish {MeinFinish} }

        // Transformationen, die der Reihe nach angewendet werden
        // Verschiebung in y-Richtung, damit das Bild
        //      etwa in der Mitte ist
        translate <0,1.6,0>

       } // Ende der Vereinigung
// Ende der Datei xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

2.10 Programmieren

2.10.1 Reelle Konstanten, Vektoren, Objekte, Transformationen, Farben, u.a. können mit `#declare` definiert werden.

- `#declare MeinViolett = pigment {color rgb <0.9,0.01,0.7>}`
- `#declare Pfeil = union {cylinder { ... } cone { ... } }`
- `#declare Polyeder = object {box { ... } clipped_by { ... } }`
- `#declare Mittelpunkt = <3,4,5>;`
- `#declare Radius = 5.02;`

Die Definitionen von reellen Konstanten und Vektoren müssen stets mit einem *Semikolon* beendet werden.

Der Aufruf erfolgt dann etwa so:

- `pigment{ MeinViolett }`
- `object{ Pfeil }`
- `object{ Polyeder rotate <20,30,40> }`
- `sphere{ Mittelpunkt, Radius ... }`

Vgl. dazu die Hinweise zum Schlüsselwort `object` in 2.5.3 und zur Schreibweise mit und ohne Komma in 2.3.2. Im letzten Beispiel darf das *Komma* zwischen den Worten `Mittelpunkt` und `Radius` nämlich nicht weggelassen werden.

2.10.2 Zur Verarbeitung von Daten ist es oft zweckmäßig, diese in *Datenfeldern* abzulegen. So definiert etwa

```
#declare Punkt = array[5]
```

ein Datenfeld mit 5 Elementen, wobei die *Indizes von* 0 *bis* 4 laufen. Die einzelnen Werte sind aber noch nicht belegt. Dies kann etwa durch

```
#declare Punkt[3] = <1,2,4>;
```

geschehen. Wichtig: Alle Einträge in einem Datenfeld müssen vom selben Typ (hier Vektoren aus $\mathbb{R}^3$) sein. Andererseits wird durch

```
#declare Wert = array[3][2] { {7.1,6.3}, {1.3,2.0}, {0.1,-1.1} }
```

nicht nur ein (3×2)-Datenfeld definiert, sondern auch gleich wie angegeben mit reellen Zahlen belegt. Zahlreiche Manipulationen mit Datenfeldern sind in `arrays.inc` definiert.

2.10.3 In POV-Ray können `#for`-*Schleifen* definiert werden:

- `#for(A,Aanf,Aend,Astufe)` *Anweisung(en)* `#end`
- Die Variable `A` stimmt zu Beginn mit dem Startwert `Aanf` überein. Bei jedem Durchlauf wird die positive (oder negative) Schrittweite `Astufe` zu `A` addiert. Sobald der Endwert `Aend` echt überschritten (bzw. unterschritten) wird, bricht die Verarbeitung ab. Der Wert für `Astufe` kann weggelassen werden. In diesem Fall wird die Schrittweite gleich 1 gesetzt.
- *Anweisungen* sind Rechnungen, Graphikbefehle, Definitionen usw.

Die Syntax von `#while`-*Schleifen* ist analog:

- `#while (`*Bedingung*`)` *Anweisung(en)* `#end`
- Die *Bedingung* wird in runde Klammern `(...)` gesetzt. Mehrere Bedingungen können durch logische Operatoren (vgl. 2.10.4) verknüpft werden.

Verzweigungen und *bedingte Anweisungen* werden ganz ähnlich geschrieben:

- `#if (`*Bedingung*`)` *Anweisung(en)* `#else` *Anweisung(en)* `#end`
- `#if (`*Bedingung*`)` *Anweisung(en)* `#end`

2.10.4 Die Notation für *logische Operationen* ist folgendermaßen:

- nicht A: `!A`
- A und B: `(A & B)`
- A oder B: `(A | B)`

Ferner kann überprüft werden, ob gewisse Relationen für reelle Zahlen zutreffen oder nicht:

- A kleiner B: `(A < B)`
- A kleiner oder gleich B: `(A <= B)`
- A gleich B (bis auf $|A - B| < \varepsilon$): `(A = B)`
- A ungleich B (bis auf $|A - B| >= \varepsilon$): `(A != B)`
- A größer oder gleich B: `(A >= B)`
- A größer B: `(A > B)`

Das Ergebnis ist in jedem Fall entweder 0 (false) oder 1 (true). Beim Zusammensetzen von Bedingungen werden Ausdrücke in runden Klammern zuerst berechnet.

2.10.5 In POV-Ray können *Macros* definiert werden. Die Syntax lautet zum Beispiel

```
#macro MeinTorus(A,B) ... #end
```

Dabei sind `A,B` die Namen von zwei übergebenen Variablen. Werden keine Variablen übergeben, so sind dennoch die beiden Klammern zu schreiben. Die drei Punkte stehen stellvertretend für die auszuführenden Anweisungen. Mehr zur Verwendung von Macros und *lokalen Variablen* findet sich in den Beispielen 4.3.3 und 4.3.5.

Beispiel 2.10.6 Ein Würfel wird um die z-Achse kontinuierlich verschraubt. Es sollen mehrere Positionen des bewegten Würfels dargestellt werden, um den Bewegungsvorgang im Raum zu illustrieren. Vgl. dazu Abbildung 2.12.

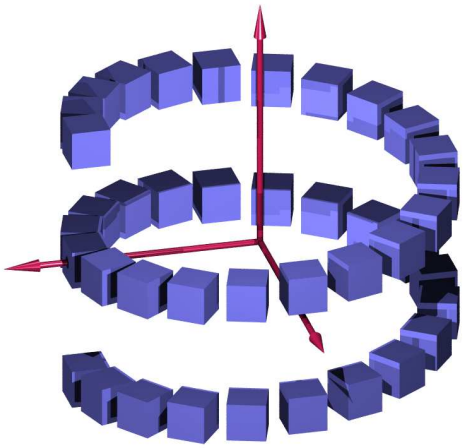


Abbildung 2.12: Verschraubung eines Würfels

In POV-Ray können die einzelnen Positionen des Würfels etwa so programmiert werden:

```
union { // Laufvariable tt, Anfangs- und Endwert
        #for (tt,-20,20)
         box {<2,-.25,-.25>,<2.5,.25,.25> // ein fester Wuerfel
         scale 1.5 // wird zuerst skaliert
         rotate <0,0,360/20*tt> translate <0,0,tt/8>}
                  // und dann verschraubt um z-Achse
        #end            // Ende der Schleife
        pigment {Blau}
        finish {phong .9} // Farbe und Finish fuer alle gleich
}
```

Dabei wurde schon zuvor mit

```
#declare Blau =pigment {color rgb <.5,0.5,1>}
```

die Farbe `Blau` definiert.

Kapitel 3

Zentralprojektion

3.1 Der projektiv abgeschlossene Anschauungsraum

3.1.1 Im Gegensatz zu einer Parallelprojektion werden bei einer *Zentralprojektion* Sehgeraden durch einen festen Punkt zur Abbildung auf eine Bildebene verwendet. Wir werden das später präzisieren und betrachten zunächst ein einfaches Beispiel:

Ein *Mengerschwamm* entsteht aus einem Würfel mit der Kantenlänge a durch folgenden rekursiven Prozess:

- Der Würfel wird in 27 gleich große Teilwürfel mit der Kantenlänge $\frac{a}{3}$ zerlegt.
- Jene sieben Teilwürfel, die an keine Kante des Ausgangswürfels anliegen, werden entfernt.
- Es verbleiben 20 Teilwürfel, mit denen der Prozess wiederholt wird.

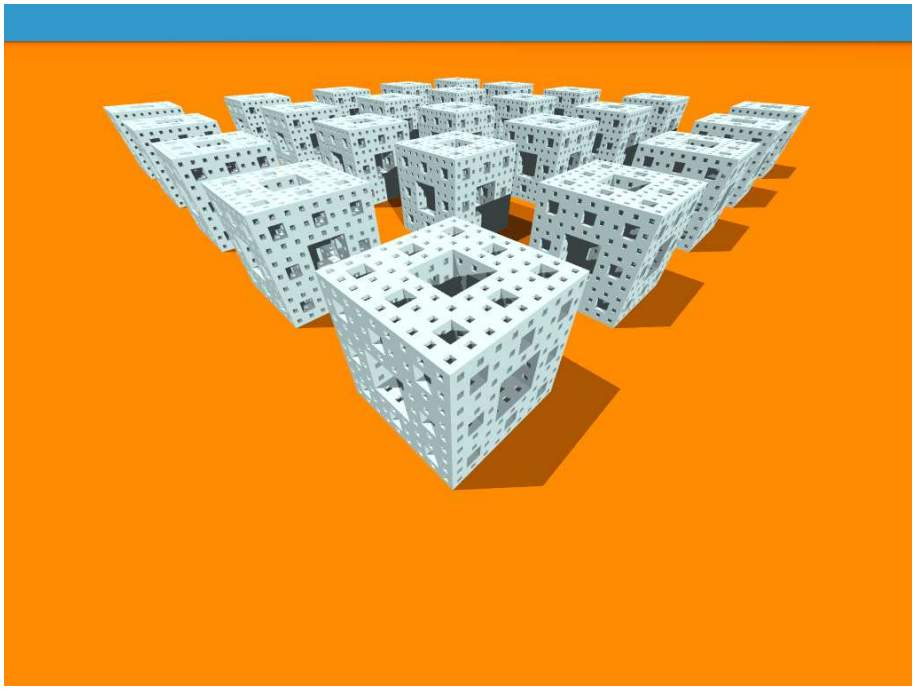


Abbildung 3.1: Zentralprojektion von 25 Exemplaren eines Mengerschwammes

Um einen Mengerschwamm zu illustrieren oder zu bauen, muss der Prozess nach einigen Schritten abgebrochen werden. Wir bezeichnen ein solches *Näherungspolyeder* eines Mengerschwammes im Folgenden ebenfalls als Mengerschwamm.[1] In Abbildung 3.1 sind 25 Exemplare eines Mengerschwammes unter einer Zentralprojektion abgebildet. Die einzelnen Exemplare sind dabei an einem kartesischen Koordinatenraster ausgerichtet.

In Abbildung 3.2 wurden im Bild nachträglich die Bilder einzelner paralleler Strecken verlängert, um zu illustrieren, dass die Bilder paralleler Geraden in einem Punkt zusammenlaufen können. Ein solcher Punkt wird ein *Fluchtpunkt* genannt. Es wurden insgesamt sechs Fluchtpunkte hervorgehoben. Jeder Fluchtpunkt gehört zu einer Klasse paralleler Geraden. Dabei liegen drei dieser Fluchtpunkte immer dann auf derselben Geraden der Bildebene, falls die betreffenden Strecken (bzw. deren Trägergeraden) im Raum zur selben Ebene parallel sind. Die an den Mengerschwämmen auftretenden Quadrate (bzw. deren Trägerebenen) bestimmen auf diese Weise im Bild drei *Fluchtgeraden*. Dies wird besonders deutlich für die zur hellbraunen „horizontalen Standebene“ parallelen Geraden. Ihre Fluchtpunkte liegen alle am *Horizont*, das ist jene Fluchtgerade, wo scheinbar Himmel und Erde zusammentreffen.

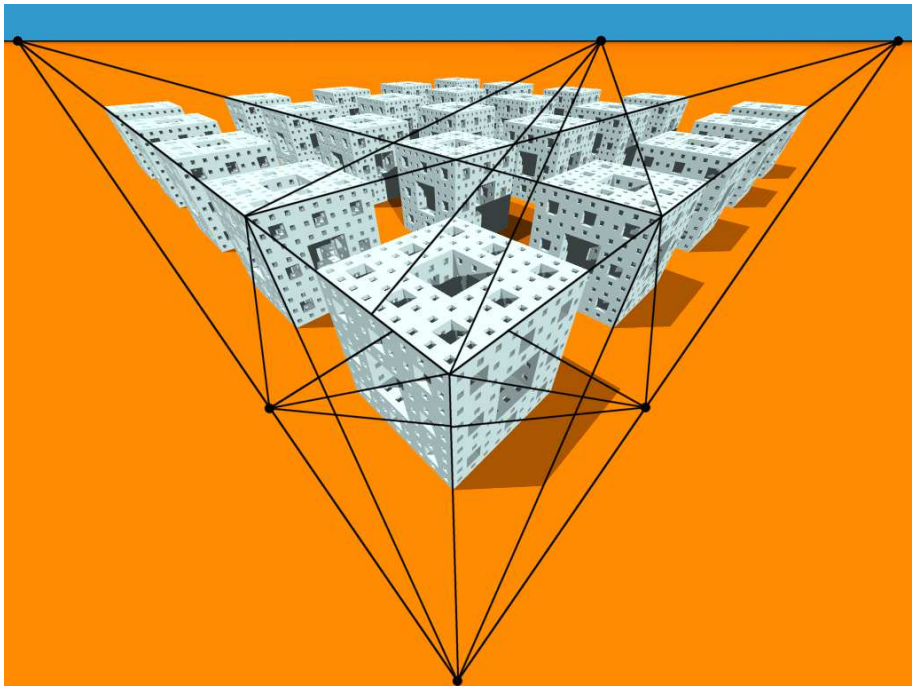


Abbildung 3.2: Fluchtpunkte

3.1.2 Zur Beschreibung einer Zentralprojektion ist es zweckmäßig, dem Anschauungsraum $\mathcal{A}$ neue Punkte hinzuzufügen. Diese werden *Fernpunkte* genannt. Die in 3.1.1 vorgestellten Fluchtpunkte können dann als Bilder dieser Fernpunkte interpretiert werden. Wer mag, kann sich daher unter einem Fernpunkt den „unendlich fernen Schnittpunkt“ paralleler Geraden vorstellen. Es ist aber oft besser, sich unter einem Fernpunkt gar nichts vorzustellen. Wichtig ist nur, nach welchen Regeln mit Fernpunkten zu operieren ist. Als Motivation für die nachfolgend eingeführten *Ferngeraden* können die in Abbildung 3.2 eingezeichneten Fluchtgeraden dienen. Sie

[1] Ein an der TU Dresden mit Hilfe eines 3D-Druckers erstelltes Modell ist an unserem Institut im 8. Stock ausgestellt.

sind die perspektiven Bilder von Ferngeraden.

Definition 3.1.3 Aus dem Anschauungsraum entsteht folgendermaßen der *projektiv abgeschlossene Anschauungsraum*:

1. Jede Gerade $g \subset \mathcal{A}$ wird um genau einen weiteren Punkt $F \notin \mathcal{A}$ (*Fernpunkt*) erweitert; es entsteht eine *projektiv abgeschlossene Gerade*[2] $g \cup \{F\}$.
2. Zu Geraden $g_1, g_2 \subset \mathcal{A}$ wird genau dann derselbe Fernpunkt hinzugefügt, falls sie parallel sind.
3. Die Menge aller so auftretenden Fernpunkte wird mit ω bezeichnet. Die *Punktmenge* des projektiv abgeschlossenen Anschauungsraumes sei $\mathcal{P} := \mathcal{A} \cup \omega$.
4. Eine Teilmenge $f \subset \omega$ wird eine *Ferngerade* genannt, falls sie aus den Fernpunkten aller Geraden einer festen Ebene ε entsteht.
5. Eine *projektive Gerade* ist entweder eine projektiv abgeschlossene Gerade oder eine Ferngerade.

Der projektiv abgeschlossene Anschauungsraum ist strukturell betrachtet die Menge $\mathcal{P}$ mit einem System ausgezeichneter Teilmengen, nämlich der Menge aller projektiven Geraden.[3]

3.1.4 Jeder Fernpunkt F wird durch eine Gerade g repräsentiert. In einer Skizze lässt sich ein Fernpunkt so *illustrieren*: Es wird eine in g enthaltene Strecke gezeichnet, wobei an einem Ende der Strecke der Buchstabe F geschrieben wird. Welches Ende gewählt wird, ist dabei belanglos. An die Stelle von g kann auch jede zu g parallele Gerade treten.[4] In manchen Büchern wird am Ende der Strecke zusätzlich ein Pfeil angebracht, der andeuten soll: „In dieser Richtung geht es zum Fernpunkt“. Allerdings geht es in der entgegengesetzten Richtung auch zum selben Fernpunkt. Das liegt im Wesen der projektiven Erweiterung des Anschauungsraumes.

Jede Ferngerade f wird durch eine Ebene ε repräsentiert. Zwei Ebenen ε_1 und ε_2 legen genau dann dieselbe Ferngerade fest, falls sie parallel sind. Denn genau in diesem Fall gibt es zu jeder Geraden in ε_1 eine parallele Gerade in ε_2 und umgekehrt.

In einer Skizze lässt sich eine Ferngerade nicht gut illustrieren. Wir werden immer nur Begleittext anmerken, dass etwa f die Ferngerade einer gegebenen Ebene ε ist.

3.1.5 Es ist zweckmäßig, jede Ebene zu einer sogenannten *projektiv abgeschlossenen Ebene* zu erweitern. Dazu fügen wir zur Punktmenge der Ebene alle Fernpunkte jener Ferngeraden hinzu, welche durch die Ebene repräsentiert wird. Im Sinne dieser Festsetzung werden daher parallele Ebenen durch dieselbe Ferngerade abgeschlossen, während zu nicht parallelen Ebenen verschiedene Ferngeraden dazukommen. Eine elementare Diskussion zeigt:

- Je zwei verschiedene projektive Geraden derselben projektiv abgeschlossenen Ebene haben genau einen Punkt gemeinsam.

[2] Im Gegensatz zur reellen Analysis, wo die Zahlengerade $\mathbb{R}$ durch zwei verschiedene Punkte $+\infty$ und $-\infty$ ergänzt wird, kommt hier pro Gerade nur ein einziger Punkt hinzu.

[3] Vgl. zu diesem Thema [3].

[4] Ein Fernpunkt kann auch als eine Klasse paralleler Geraden definiert werden. Dieses mengentheoretische Modell für die Menge aller Fernpunkte zeigt, dass die projektive Erweiterung tatsächlich möglich ist.

- Je zwei verschiedene Ferngeraden haben genau einen Fernpunkt gemeinsam.[5]

Die Menge ω verhält sich also in Bezug auf die in ihr liegenden Geraden so wie eine projektiv abgeschlossene Ebene. Aus diesem Grund wird die Menge ω als *Fernebene* bezeichnet.

Eine *projektive Ebene* sei im Folgenden entweder eine projektive abgeschlossenen Ebene oder die (einzige) Fernebene.

3.1.6 Zur Vereinfachung der Sprechweise nennen wir $\mathcal{P}$ auch kurz den *projektiven Raum*. Die Punkte von $\mathcal{A}$ werden *eigentlich* genannt. Wenn von Geraden $g_1, g_2, \ldots$ oder Ebenen $\varepsilon_1, \varepsilon_2, \ldots$ die Rede ist, meinen wir immer eine projektive Gerade oder eine projektive Ebene (einschließlich ihrer Fernpunkte). Das Adjektiv *eigentlich* kennzeichnet jene Geraden und Ebenen, die nicht nur aus Fernpunkten bestehen, also die projektiv abgeschlossenen Geraden und Ebenen. Wir fassen die wesentliche Eigenschaften des projektiven Raums $\mathcal{P}$ zusammen.

- Je zwei verschiedene Punkte haben genau eine Verbindungsgerade.
- Je zwei verschiedene Ebenen haben genau eine Schnittgerade.
- Zwei verschiedene Geraden haben genau dann einen Schnittpunkt, falls sie komplanar sind (d. h. gemeinsam einer Ebene angehören).
- Jede Gerade g und jede Ebene ε mit $g \not\subset \varepsilon$ haben genau einen gemeinsamen Punkt.

Durch die projektive Erweiterung des Anschauungsraumes $\mathcal{A}$ wird also die Sonderrolle paralleler Geraden und Ebenen eliminiert.

3.1.7 Die oben erklärte Erweiterung des Anschauungsraumes kann auch mit Methoden der linearen Algebra beschrieben werden. Für jeden Punkt $(p_1, p_2, p_3) \in \mathbb{R}^3$ ist $(1, p_1, p_2, p_3) \in \mathbb{R}^4$ vom Nullvektor verschieden und bestimmt den eindimensionalen Unterraum

$$\mathbb{R}(1, p_1, p_2, p_3) \subset \mathbb{R}^4.$$

Für jeden Fernpunkt wählen wir eine beliebige repräsentierende Gerade

$$(a_1, a_2, a_3) + \mathbb{R}(r_1, r_2, r_3)$$

mit dem Aufpunkt $(a_1, a_2, a_3) \in \mathbb{R}^3$ und dem Richtungsvektor $(r_1, r_2, r_3) \neq (0, 0, 0)$. Dem so festgelegten Fernpunkt wird der eindimensionale Unterraum

$$\mathbb{R}(0, r_1, r_2, r_3) \subset \mathbb{R}^4$$

zugeordnet. Dieser ist von der Auswahl des Aufpunktes und des Richtungsvektors unabhängig.[6] Insgesamt ergibt sich eine Bijektion der Menge aller Punkte von $\mathcal{P}$ auf die Menge der eindimensionalen Unterräume von $\mathbb{R}^4$.

Die Geraden bzw. Ebenen von $\mathcal{P}$ entsprechen genau den zwei- bzw. dreidimensionalen Unterräumen von $\mathbb{R}^4$; dabei ist jeder solche Unterraum von $\mathbb{R}^4$ als Menge der in ihm enthaltenen eindimensionalen Unterräume zu interpretieren.

In der linearen Geometrie wird allgemein die Punktmenge des projektiven Raumes über einem Vektorraum V als die Menge der eindimensionalen Unterräume von V definiert. Vgl. dazu etwa [8]. Wir benötigen dies aber im Folgenden nicht.

[5] Genauer gilt: Die verschiedenen Ferngeraden werden durch nicht parallele Ebenen repräsentiert. Die Schnittgerade dieser beiden Ebenen ist ein Repräsentant des einzigen gemeinsamen Fernpunktes.

[6] Wir haben die vierte Koordinate hier den Tripeln aus $\mathbb{R}^3$ vorangestellt. In der Literatur wird sie oft auch ans Ende gestellt.

3.2 Definition und Eigenschaften einer Zentralprojektion

3.2.1 Wir geben im projektiven Raum $\mathcal{P}$ eine eigentliche *Bildebene* π und einen eigentlichen *Augpunkt* O vor, der nicht in π liegt. Die *Zentralprojektion* (oder *Perspektive*) c mit Zentrum O auf die Ebene π ist wie folgt erklärt:[7]

$$c : \mathcal{P} \setminus \{O\} \to \pi : X \mapsto OX \cap \pi, \tag{3.1}$$

wobei OX die (eindeutig bestimmte) Verbindungsgerade von X und O bezeichnet (vgl. Abbildung 3.3). Diese Vorschrift versagt nur für den Punkt O, da der Augpunkt O nicht mit sich selbst zu einer Geraden verbunden werden kann. Die Zentralprojektion c ist daher im Augpunkt O nicht definiert.

Die durch O verlaufenden Geraden werden die *Sehgeraden* der Zentralprojektion genannt. Zwei Punkte $\neq O$ haben offenbar genau dann denselben Bildpunkt, wenn sie derselben Sehgeraden angehören.

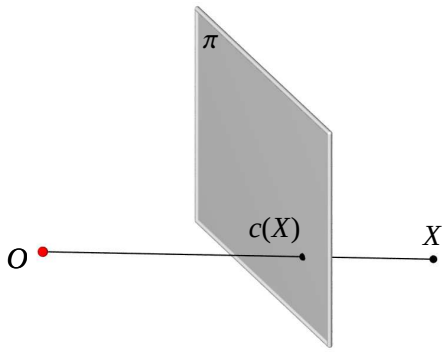


Abbildung 3.3: Zentralprojektion

Die Zentralprojektion ist dem *einäugigen Sehen* beim Menschen nachempfunden. Ebenso realisiert jeder *Photoapparat* (von Verzerrungen durch das Linsensystem abgesehen) eine Zentralprojektion: Der Augpunkt entspricht dem optischen Mittelpunkt des Objektivs, die Bildebene entspricht der Ebene des Films. Eine andere Realisierung ist die bekannte *Lochkamera*.

3.2.2 Im Gegensatz zu einer Parallelprojektion gibt es in der Bildebene einer Zentralprojektion einen ausgezeichneten Punkt: Legen wir durch O die zu π orthogonale Gerade, so schneidet diese π in einem Punkt H, welcher der *Hauptpunkt* der Zentralprojektion genannt wird. Der Abstand $\overline{OH}$ wird die *Distanz* genannt. Die Gerade OH heißt die *Blickachse*. Vgl. Abbildung 3.4.

3.2.3 Das perspektive Bild $c(V)$ eines Punktes $V \neq O$ ist genau dann ein Fernpunkt, falls die Gerade OV zu π parallel ist. Ein solcher Punkt V wird ein *Verschwindungspunkt* genannt. Der Punkt V „verschwindet unter c im Unendlichen“, sein Bild $c(V)$ liegt auf der Ferngeraden $\pi \cap \omega$. Zur Veranschaulichung von $c(V)$ können einige parallele Strecken der Ebene π dienen, deren Trägergeraden den Punkt $c(V)$ enthalten. Vgl. Abbildung 3.4.

[7] Der Buchstabe c erinnert an die einst übliche Schreibweise *Centralprojektion*.

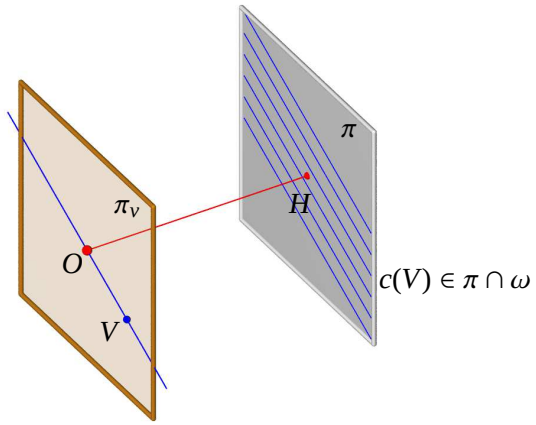


Abbildung 3.4: Hauptpunkt und Verschwindungsebene

Die zur Bildebene π parallele Ebene π_v durch den Augpunkt O besteht aus allen Verschwindungspunkten sowie aus dem Punkt O. Diese Ebene heißt daher die *Verschwindungsebene* π_v. Würden wir keine Fernpunkte zum Anschauungsraum $\mathcal{A}$ hinzufügen, so hätten alle Punkte der Verschwindungsebene keine Bildpunkte.

Die Verschwindungsebene definiert zwei offene Halbräume innerhalb der Menge $\mathcal{A}$ der eigentlichen Punkte. Jener Halbraum, in dem die Bildebene liegt, wird der *Sehraum* genannt, der andere Halbraum heißt der *virtuelle Raum*.[8] Die Punkte der Fernebene ω werden keinem der beiden Halbräume zugerechnet. Bei anschaulichen Bildern werden in der Regel nur Punkte des Sehraumes abgebildet.

Ist F ein Fernpunkt, so wird sein Bildpunkt $c(F)$ als *Fluchtpunkt* bezeichnet. Dabei interessieren vor allem jene Fluchtpunkte, die eigentlich sind, da in ihnen die Bilder paralleler Geraden zusammenlaufen. Vgl. Abbildung 3.2.

Die Punkte der Ferngeraden $\pi \cap \pi_v$ sind die einzigen Punkte, die zugleich Verschwindungspunkte und Fluchtpunkte sind.

3.2.4 Unter Verwendung kartesischer Koordinaten lassen sich die Abbildungsgleichungen einer Zentralprojektion c leicht angeben, falls der Augpunkt O im Ursprung des Koordinatensystems gewählt wird und die Bildebene durch $x_1 = 1$ gegeben ist. Die Gleichung der Verschwindungsebene lautet daher $x_1 = 0$. Aus der Definition der Zentralprojektion folgt unmittelbar

$$(x_1, x_2, x_3) \overset{c}{\mapsto} \left(1, \frac{x_2}{x_1}, \frac{x_3}{x_1}\right), \quad \text{falls} \quad x_1 \neq 0. \tag{3.2}$$

Damit lässt sich für jeden eigentlichen Punkt, der nicht in der Verschwindungsebene liegt, der Bildpunkt berechnen. Sollen auch die Bilder von Verschwindungs- und Fernpunkten berechnet werden, so muss auf die Beschreibung der Punkte von $\mathcal{P}$ mit Hilfe der eindimensionalen Unterräume von $\mathbb{R}^4$ zurückgegriffen werden. Wir gehen darauf nicht ein.

Satz 3.2.5 (Abbildung von Geraden und Ebenen) *Sei $c : \mathcal{P} \setminus \{O\} \to \pi$ die Zentralprojektion mit Augpunkt O auf eine Ebene π. Dann gelten die folgenden Eigenschaften:*

(a) *Jede Sehgerade wird unter c auf einen Punkt abgebildet.*

[8] Bei einem Photoapparat liegt eine andere Anordnung vor: Die Bildebene befindet sich im virtuellen Raum.

(b) *Jede nicht durch O verlaufende Gerade wird unter c bijektiv auf eine Gerade von π abgebildet.*

(c) *Jede Ebene durch O wird wird unter c auf eine Gerade abgebildet.*

(d) *Jede nicht durch O verlaufende Ebene wird unter c bijektiv und kollinear*[9] *auf die Ebene π abgebildet.*

Beweis. Alle Behauptungen sind mit Hilfe der in 3.1.6 genannten Eigenschaften von (projektiven!) Geraden und Ebenen leicht zu zeigen. □

Die durch O verlaufenden Geraden und Ebenen werden auch *projizierend* genannt.

3.2.6 Die Abbildung 3.5 zeigt, wie das perspektive Bild einer eigentlichen Geraden $g \nparallel \pi$ entsteht. Auf der Geraden g gibt es zwei ausgezeichnete Punkte: Den Verschwindungspunkt $V := g \cap \pi_v$ und den Fernpunkt $F := g \cap \omega$. Dementsprechend sind auf der Bildgeraden $c(g)$ der Fernpunkt $c(V)$ und der Fluchtpunkt $c(F)$ ausgezeichnet.

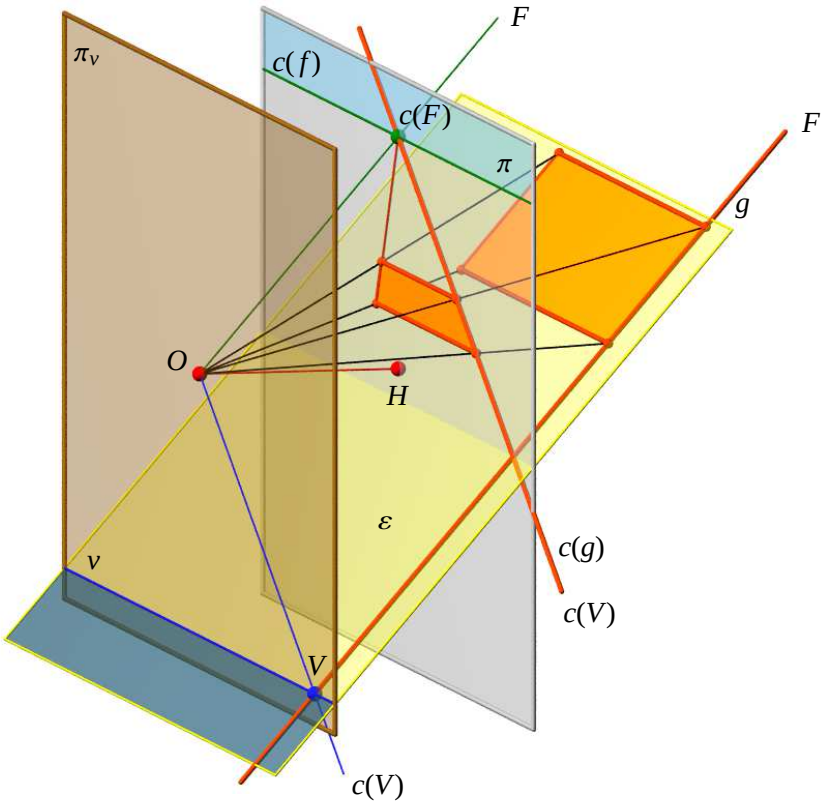


Abbildung 3.5: Perspektives Bild einer Geraden und einer Ebene

Der Fluchtpunkt $c(F)$ teilt die Gerade $c(g) \cap \mathcal{A}$ in zwei offene Halbgeraden. Die eine Halbgerade besteht aus den Bildern der Punkte von g im Sehraum, die andere Halbgerade aus den Bildern der Punkte von g im virtuellen Raum. Der Fernpunkt $c(V)$ wird keiner der beiden offenen Halbgeraden zugerechnet.

[9] Das heißt: Die Bilder kollinearer Punkte der Ebene sind ebenfalls kollinear. Es liegt hier eine *Kollineation* vor. Das ist eine bijektive Abbildung einer projektiven Ebene in eine projektive Ebene, bei der je drei kollineare Punkte Bildpunkte besitzen, die ebenfalls kollinear sind. Jede Kollineation bildet Geraden auf Geraden ab.

Jede eigentliche Gerade h, die zu π parallel ist und nicht in der Verschwindungsebene π_v liegt, ist zu ihrer Bildgeraden $c(h)$ parallel. Die Einschränkung von c auf eine solche Gerade ist nach dem Strahlensatz eine Ähnlichkeit und damit teilverhältnistreu. Die Gerade erscheint im Bild also ähnlich vergrößert oder verkleinert. Der Verschwindungspunkt $h \cap \pi_v = h \cap \pi$ ist zugleich ihr Fluchtpunkt. Vgl. dazu die beiden zu π parallelen Seiten des orange hervorgehobenen Rechtecks in Abbildung 3.5.

Wir notieren noch eine sehr einfache Folgerung:

Satz 3.2.7 *Die Bilder paralleler Geraden gehen durch denselben Fluchtpunkt.*

Beweis. Parallele Gerade haben definitionsgemäß denselben Fernpunkt. □

Insbesondere ist der Hauptpunkt H der Fluchtpunkt der perspektiven Bilder aller zur Bildebene π orthogonalen Geraden.

3.2.8 Die Abbildung 3.5 zeigt, wie das perspektive Bild einer eigentlichen Ebene $\varepsilon \nparallel \pi$ entsteht. In der Ebene ε gibt es zwei ausgezeichnete Geraden: Die *Verschwindungsgerade* $v = \varepsilon \cap \pi_v$ und die Ferngerade $f = \varepsilon \cap \omega$. Dementsprechend sind im Bild der Ebene $c(\varepsilon) = \pi$ die Ferngerade $c(v) = \pi \cap \omega$ und die *Fluchtgerade* $c(f)$ ausgezeichnet.

Die Fluchtgerade $c(f)$ teilt die Ebene $c(\varepsilon) \cap \mathcal{A} = \pi \cap \mathcal{A}$ in zwei offene Halbebenen. Die eine Halbebene besteht aus den Bildern der Punkte von ε im Sehraum, die andere Halbebene aus den Bildern der Punkte von ε im virtuellen Raum (in der Figur blau hervorgehoben). Die Ferngerade $c(v)$ wird keiner der beiden offenen Halbebenen zugerechnet. Außerdem ist dargestellt, wie ein Rechteck in der Ebene ε perspektiv abgebildet wird. Da zwei Seiten des Rechtecks zur Bildebene parallel gewählt wurden, tritt als Bild ein Trapez in π auf. Die Bilder der beiden anderen Seiten haben einen (in der Figur grün markierten) Fluchtpunkt auf $c(f)$ gemeinsam. Die Verbindungsgerade dieses Fluchtpunkts mit dem Augpunkt O ist zu den betreffenden Seiten des Rechtecks parallel.

Für jede eigentliche Ebene $\varepsilon \neq \pi_v$, die zu π parallel ist, ist die Einschränkung von c eine Ähnlichkeit und damit teilverhältnistreu. Die Ebene erscheint im Bild also ähnlich vergrößert oder verkleinert. Die Verschwindungsgerade $\varepsilon \cap \pi_v = \varepsilon \cap \pi$ ist zugleich ihre Fluchtgerade.

Wir notieren eine weitere sehr einfache Folgerung:

Satz 3.2.9 *Die Bilder paralleler Ebenen haben dieselbe Fluchtgerade.*

Beweis. Parallele Ebenen haben definitionsgemäß dieselbe Ferngerade. □

Insbesondere verlaufen die Fluchtgeraden der zur Bildebene π orthogonalen Ebenen alle durch den Hauptpunkt.

3.2.10 Verschwindungspunkte können im perspektiven Bild nicht unmittelbar „gesehen“ werden. In Abbildung 3.6 ist ein Polarkoordinatenraster in einer horizontalen Ebene abgebildet. Da eine Orthogonalprojektion auf diese Ebene verwendet wurde, erscheint die gesamte Figur unverzerrt.

Wir stellen dieselbe Szene jetzt in mehreren Perspektiven dar. Abbildung 3.7 links liegt der Ursprung liegt im Sehraum. Dann wurde das Polarkoordinatensystem so verschoben, dass der Ursprung ein Verschwindungspunkt wird. Die Lage von Augpunkt und Bildebene wurde nicht

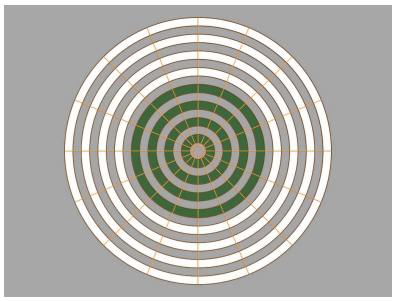


Abbildung 3.6: Polarkoordinatenraster

verändert. In der mittleren Abbildung erscheinen daher die radialen Geraden durch den Ursprung parallel. In Abbildung 3.7 rechts wurde der Ursprung schließlich in den virtuellen Raum verschoben. Die Bilder der radialen Geraden laufen jetzt durch einen Punkt oberhalb des *Horizonts*; dieser nur anschaulich begründete Begriff bezeichnet die Fluchtgerade aller horizontalen Ebenen.

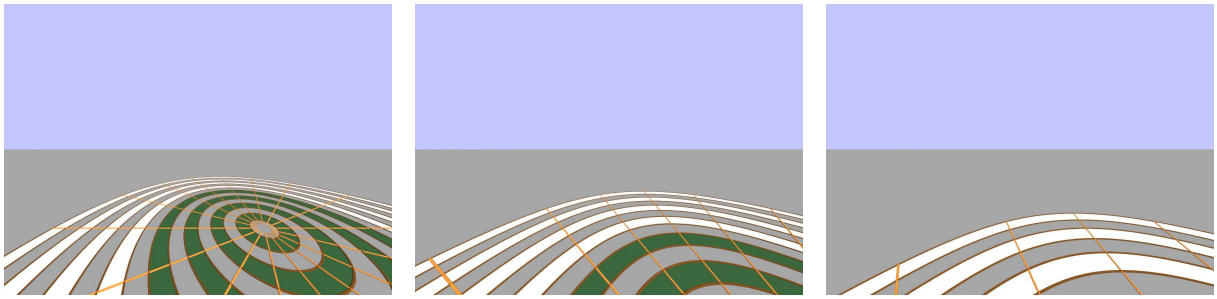


Abbildung 3.7: Ursprung im Sehraum, in der Verschwindungsebene, im virtuellen Raum

3.2.11 Wir erinnern an den Begriff des *Teilverhältnisses*: Sind B und C zwei verschiedene eigentliche Punkte einer Geraden g, so gibt es für jeden eigentlichen Punkt A der Geraden g genau eine reelle Zahl x mit

$$\overrightarrow{CA} = x\overrightarrow{CB}.$$

Diese Zahl x wird bekanntlich als Teilverhältnis $\mathrm{TV}(A, B, C)$ bezeichnet. Gilt etwa $\mathrm{TV}(A, B, C) = -1$, so ist C der Mittelpunkt von A und B.

Unter einer Zentralprojektion bleiben Teilverhältnisse im Allgemeinen nicht erhalten. Das zeigt schon Abbildung 3.1: Das Bild des Mittelpunkts einer (beliebigen) Kante eines der Würfel fällt nicht in den Mittelpunkt des perspektiven Bildes der Kante.

Sind B, C und D drei verschiedene eigentliche Punkte einer Geraden g, so wird für jeden eigentlichen Punkt $A \neq D$ der Geraden g der Quotient

$$\frac{\mathrm{TV}(A, B, C)}{\mathrm{TV}(A, B, D)} =: \mathrm{DV}(A, B, C, D)$$

das *Doppelverhältnis* der vier Punkte genannt. Lässt man in dieser Formel auch $A = D$ zu, so tritt im Nenner eine Null auf; das Doppelverhältnis wird dann gleich ∞ gesetzt.

Von dieser Definition ausgehend kann gezeigt werden, dass sich das Doppelverhältnis unter einer Zentralprojektion nicht ändert, sofern nur g nicht projizierend ist und alle vier Bildpunkte $c(A), c(B), c(D), c(D)$ eigentlich sind. Es gilt also

$$\mathrm{DV}(c(A), c(B), c(D), c(D)) = \mathrm{DV}(A, B, C, D).$$

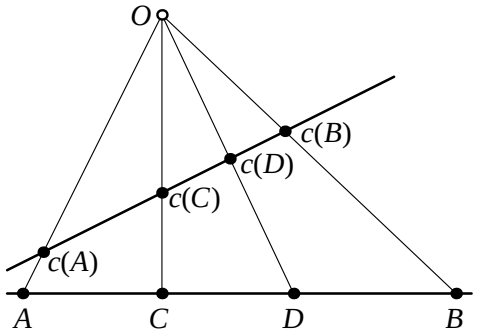


Abbildung 3.8: Doppelverhältnistreue

Sobald dieses Ergebnis zur Hand ist, lässt sich der Begriff des Doppelverhältnisses erweitern: Es seien B, C, D drei verschiedene Punkte einer Geraden $g \subset \mathcal{P}$ und A ein weiterer Punkt von g. Wenn unter den gegebenen Punkten wenigstens ein Fernpunkt vorkommt, werden die vier Punkte aus einem beliebigen Zentrum Z so auf eine geeignete Hilfsgerade g' projiziert, dass alle vier Bildpunkte A', B', C', D' eigentlich sind. Das Doppelverhältnis $\mathrm{DV}(A, B, C, D)$ wird dann als $\mathrm{DV}(A', B', C', D')$ *definiert*. Vgl. Abbildung 3.9. In der linken Figur ist B ein Fernpunkt, in der rechten Figur sind A, B, C, D vier Fernpunkte. Mit Hilfe der zuvor bewiesenen Doppelver-

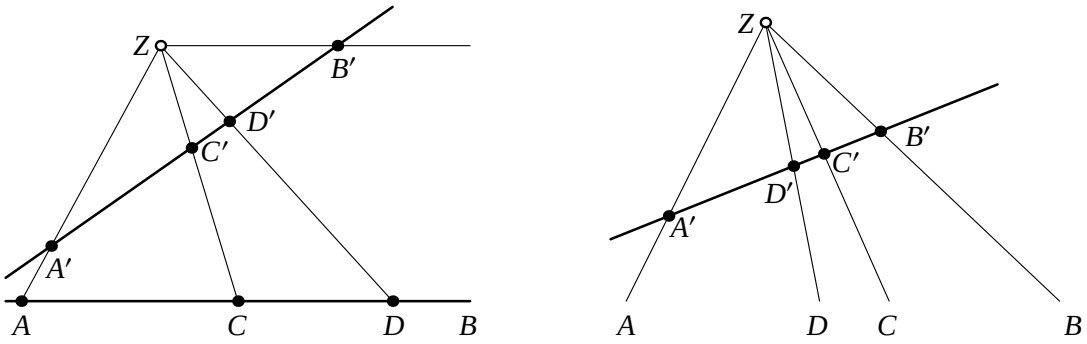


Abbildung 3.9: Definition des Doppelverhältnisses

hältnistreue gelingt der Nachweis, dass diese Definition nicht von der Auswahl des Punktes Z und der Geraden g' abhängt.[10] Wir erwähnen noch den Sonderfall, dass D der Fernpunkt einer eigentlichen Geraden g ist. Dann gilt einfach

$$\mathrm{DV}(A, B, C, D) = \mathrm{TV}(A, B, C).$$

In Ergänzung von Satz 3.2.5 gilt somit:

Satz 3.2.12 *Sei $c : \mathcal{P} \setminus \{O\} \to \pi$ die Zentralprojektion mit Augpunkt O auf eine Ebene π. Dann ist die Einschränkung von c auf jede nicht projizierende Gerade doppelverhältnistreu.*

Diese Doppelverhältnistreue ist unter anderem für die *Rekonstuktion* von Objekten aus perspektiven Bildern (etwa mehreren photographischen Aufnahmen) von zentraler Bedeutung.

Bemerkung 3.2.13 Die Diskussion der *Verzerrung* von Strecken ist für eine Zentralprojektion zwar möglich, bringt aber wenig nützliche Informationen. Im Gegensatz zu einer Parallelprojektion ändert sich die Verzerrung einer Strecke nämlich, wenn sie im Raum beliebig verschoben

[10] Vgl. dazu etwa [5, 98ff.]. Weniger anschaulich, aber weitaus eleganter, ist es, das Doppelverhältnis gleich für alle Punkte mit Hilfe des projektiven Raumes über $\mathbb{R}^4$ zu erklären.

wird. Wegen der im Allgemeinen nicht gegebenen Teilverhältnistreue ändert sich die Verzerrung einer Strecke auch, falls sie verlängert oder verkürzt wird.

Unter Verwendung der Abbildungsgleichungen aus (3.2) folgt unmittelbar, dass die Zentralprojektion aus dem Ursprung O auf die Ebene $x_1 = 1$ in einer passenden Umgebung jedes eigentlichen Punktes $P \notin \pi_v$ stetig differenzierbar ist.[11] Hat P die Koordinaten $(p_1, p_2, p_3) \in \mathbb{R}^3$, so liefert das totale Differential von c als erste *Taylor-Näherung* von c an der Stelle (p_1, p_2, p_3) die Abbildung

$$(x_1, x_2, x_3) \mapsto \left(1, \frac{p_2}{p_1}, \frac{p_3}{p_1}\right) + (x_1 - p_1, x_2 - p_2, x_3 - p_3) \cdot \begin{pmatrix} 0 & -\frac{p_2}{p_1^2} & -\frac{p_3}{p_1^2} \\ 0 & \frac{1}{p_1} & 0 \\ 0 & 0 & \frac{1}{p_1} \end{pmatrix}. \tag{3.3}$$

Wenn wir beachten, dass (p_1, p_2, p_3) im Kern der auftretenden 3×3 Matrix liegt und passend herausheben, lässt sich die Taylor-Näherung auch in der Form

$$(x_1, x_2, x_3) \mapsto \left(1, \frac{p_2}{p_1}, \frac{p_3}{p_1}\right) + \frac{1}{p_1}(x_1, x_2, x_3) \cdot \begin{pmatrix} 0 & -\frac{p_2}{p_1} & -\frac{p_3}{p_1} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

schreiben. Die Zentralprojektion c verhält sich daher in einer Umgebung von $P \notin \pi_v$ abgesehen vom Ähnlichkeitsfaktor $1/p_1$ in erster Näherung wie die Parallelprojektion auf π in Richtung der Geraden OP. Vgl. dazu Formel (1.4) in 1.2.2. Genau für den Hauptpunkt H ist die Taylor-Näherung aus (3.4) die Orthogonalprojektion auf die Bildebene π.

Bemerkung 3.2.14 Analog zu Abschnitt 1.4 lassen sich Abbildungen untersuchen, die sich aus einer Zentralprojektion auf eine *Zwischenbildebene* π (im Raum) und einer weiteren Abbildung β von π auf eine Ebene ψ zusammensetzen. Wählt man für β eine Kongruenzabbildung oder eine Ähnlichkeitsabbildung, so bleiben die wesentlichen Eigenschaften einer Zentralprojektion erhalten. Es gilt aber keine analoge Aussage zum Satz von Pohlke: Das Produkt einer Zentralprojektion und einer Affinität kann „stärker verzerrt“ sein als eine Zentralprojektion. Vgl. dazu Abbildung 3.10: Sie zeigt links eine Photographie und rechts eine nachträglich affin verzerrte Kopie des Originals. Dieses Bild lässt sich mit einem Photoapparat alleine nicht erzeugen, egal wie man die Kamera dreht und wendet.

Eine noch stärkere Verzerrung kann sich ergeben, wenn eine Zentralprojektion mit einer Kollineation zusammensetzt wird. Dies tritt etwa dann ein, wenn man ein Photo mit Hilfe einer Kamera reproduziert und die optische Achse der Kamera bei der Aufnahme nicht orthogonal zur Ebene des Photos ist.

[11] Das Ergebnis über die Differenzierbarkeit von c lässt sich auf alle Punkte von $\mathcal{P} \setminus \{O\}$ ausdehnen, wenn der projektive Raum $\mathcal{P}$ in geeigneter Weise als *differenzierbare Mannigfaltigkeit* aufgefasst wird.


Abbildung 3.10: Photo und affin verzerrtes Photo (Schloss Eggenberg, Graz)

3.3 Sichtbarkeit und Orientierung der Bildebene

3.3.1 Eine Zentralprojektion liefert nur dann ein mit unserer Seherfahrung übereinstimmendes Bild, falls ausschließlich Punkte des Sehraumes dargestellt werden. (Wir können auch nicht gleichzeitig nach vorne und nach hinten sehen.) Außerdem erschienen Objekte, die sich im virtuellen Raum befinden, im perspektiven Bild „am Kopf stehend“. Die Punkte im virtuellen Raum, die Verschwindungspunkte und die Fernpunkte sind vor allem für die theoretische Beschreibung einer Zentralprojektion wichtig.[12]

3.3.2 Wir treffen nun die folgenden Festsetzungen:

1. Sichtbarkeit wird nur für Punkte im Sehraum diskutiert.
2. Jede nicht in der Verschwindungsebene liegende Sehgerade (ohne ihren Fernpunkt) wird durch das Punktepaar (O, S) orientiert, wobei S den Schnittpunkt der Sehgeraden mit der Bildebene bezeichnet.

Damit lassen sich dann die Überlegungen zur Sichtbarkeit sinngemäß von den Parallelprojektionen übertragen. Die Sehstrahlen sind immer vom Auge zur Bildebene hin orientiert.

3.3.3 Auch bei einer Zentralprojektion ist eine Orientierung der Bildebene π mitbestimmt: Es ist jener Halbraum bezüglich π ausgezeichnet, in dem der Augpunkt liegt. Damit kann dann so wie in Abschnitt 1.5 beschrieben werden, was es bedeuten soll, dass die Abbildung $\beta : \pi \to \psi$ (vgl. 3.2.13) gleichsinnig ist. Damit wird auch hier sicher gestellt, dass keine spiegelverkehrten Bilder in ψ entstehen.

3.4 Zentralprojektion in POV-Ray

3.4.1 In POV-Ray wird eine Zentralprojektion durch den Augpunkt O und den Hauptpunkt H bestimmt. So wird etwa durch

```
camera { location <3,2,6>  look_at <0,0,0> }
```

[12] Es kann sich aber zum Beispiel eine Lichtquelle in einem Punkt des virtuellen Raumes oder in einem Fernpunkt befinden (Rückenlicht, Parallelbeleuchtung).

der Punkt $(3, 2, 6)$ als Augpunkt und der Punkt $(0, 0, 0)$ als Hauptpunkt definiert. Die Zwischenbildebene π ist dann schon mitbestimmt. Auf Grund interner Einstellungen wird dann ein passend vergrößertes oder verkleinertes elektronisches Bild erzeugt, welches in der Ebene ψ aus 3.3.3 liegt.

- Der durch `look_at` beschriebene Hauptpunkt H erscheint in der Mitte des rechteckigen Bildes.
- Die Distanz $\overline{OH}$ regelt den *Maßstab* der Ähnlichkeitsabbildung $\pi \to \psi$. Je größer der Abstand ist, desto kleiner erscheinen die Objekte im endgültigen Bild.
- Sofern die y-Achse nicht projizierend ist, weist sie im Bild nach oben. Bei projizierender y-Achse zeigt die z-Achse nach oben.

Die letztgenannte Eigenschaft ändert sich, sobald wir mit `trafo` gemäß 2.8.3 (vgl. auch Beispiel 2.9.10) im Bild auf ein Rechtssystem umrechnen. Die y-Achse und die z-Achse tauschen dann ihre Rolle.

3.4.2 Bei der Darstellung von Objekten unter einer Zentralprojektion gelten in POV-Ray folgende Einschränkungen:

- Es werden nur jene Teile von Objekten dargestellt, die innerhalb des Sehraumes liegen.
- Es werden nur jene Teile von Objekten dargestellt, die innerhalb einer vierkantigen Pyramide mit Spitze O liegen. Die Basis dieser Pyramide ist jenes Rechteck in π, das den Rand des elektronischen Bildes ergibt.[13]

POV-Ray unterscheidet zwar endliche Objekte (etwa Quader) und unendliche Objekte (etwa Ebenen), beide Arten von Objekten bestehen aber nur aus eigentlichen Punkten. Bei mit POV-Ray erstellten Zentralprojektionen treten Fluchtpunkte nur in der Form von Randpunkten des perspektiven Bildes eines unendlichen Objekts auf.

3.4.3 Der rechteckige Bildausschnitt (und damit der Ähnlichkeitsfaktor der Abbildung der Zwischenbildebene π auf die Bildebene ψ) lassen sich auch durch das Schlüsselwort `angle` steuern. Zum Beispiel ergibt

```
camera { angle 110 location <3,2,6>  look_at <0,0,0> }
```

einen *horizontalen Öffnungswinkel* von 110° für die in 3.4.2 erwähnten Pyramide, was einem starken Weitwinkelobjektiv entspricht.

Die Abbildungen 3.11 bis 3.17 illustrieren, wie sich das Bild der schon zuvor betrachteten 25 Exemplare eines Mengerschwammes je nach Wahl von Augpunkt, Hauptpunkt und Öffnungswinkel verändert. Die *Aughöhe* ist dabei der orientierte Abstand des Auges von der waagrechten Ebene, auf der alle Schwämme aufliegen.

Die vier Bilder in einer Zeile unterscheiden sich immer nur im horizontalen Öffnungswinkel. Die ersten drei Bilder in jeder Zeile stellen immer nur Ausschnittsvergrößerungen des letzten Bildes dar.

[13] Es kommt gelegentlich auch vor, dass Objekte, die dem Auge zu nahe kommen, gar nicht dargestellt werden. Theoretisch ist die Zentralprojektion ja nur in O nicht definiert. Aus numerischen Gründen muss offensichtlich eine Umgebung von O ausgenommen werden.

- Für das erste Bild (ganz links) wurde `angle 30` gesetzt. Das entspricht einem starken Teleobjektiv.
- Für das zweite Bild von links wurde kein `angle` Wert angegeben.
- Das dritte Bild von links gehört zum Wert `angle 90`.
- Das vierte Bild (ganz rechts) wurde mit `angle 110` erstellt.

Nun beschreiben wir der Reihe nach die restlichen Einstellungen. Sie sind innerhalb einer Zeile immer gleich:

- In Abbildung 3.11 zeigt die Blickachse nach oben. Daher liegt der Fluchtpunkt der vertikalen Kanten über dem Horizont. Dieser Effekt ist in der Photographie unter dem Namen *stürzende Linien* bekannt.
- In Abbildung 3.12 ist die Blickachse waagrecht, womit die vertikalen Kanten parallele Bilder haben. Der Augpunkt ist derselbe wie in Abbildung 3.11.
- Für die Bilder in Abbildung 3.13 wurde der Augpunkt weiterhin nicht verändert, die Blickachse weist nun aber abwärts. Daher laufen die Bilder der vertikalen Kanten in einem Punkt unterhalb des Horizonts zusammen.
- Abbildung 3.14 zeigt eine sogenannte *Froschperspektive*. Hier ist die Aughöhe (relativ zur Größe der Würfel) sehr klein. Die Blickachse weist nach oben.
- Abbildung 3.15 gibt erneut eine Froschperspektive zum selben Augpunkt wie zuvor wieder, aber jetzt mit waagrechter Blickachse.
- Nur der Vollständigkeit halber wurde Abbildung 3.16 erstellt. Hier zeigt die Blickachse der Froschperspektive nach unten.
- Schließlich wurde für Abbildung 3.17 die Aughöhe negativ gewählt. Damit das Bild nicht nur die waagrechte Ebene von unten zeigt, wurde die Ebene transparent gestaltet. Die Blickachse dieser *Maulwurfsperspektive* weist nach oben.
  Beachte, dass im vierten Bild (ganz rechts) der Horizont und *darunter* ein Stück des Himmels zu sehen ist. Ein solches Bild ließe sich auch mit einem Photoapparat machen, indem 25 Modelle eines Mengerschwammes auf eine Glasplatte gestellt würden.
  Selbstverständlich ließen sich auch für diese Maulwurfsperspektive weitere Ansichten mit horizontaler und nach unten gerichteter Blickachse machen.

Abbildung 3.11: Blickachse nach oben, Aughöhe etwa auf halber Höhe der Würfel

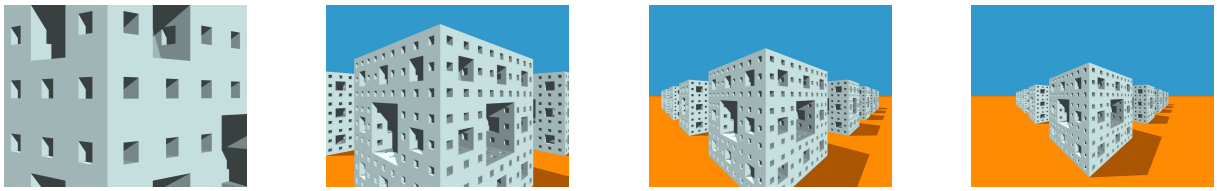


Abbildung 3.12: Blickachse waagrecht, Aughöhe etwa auf halber Höhe der Würfel

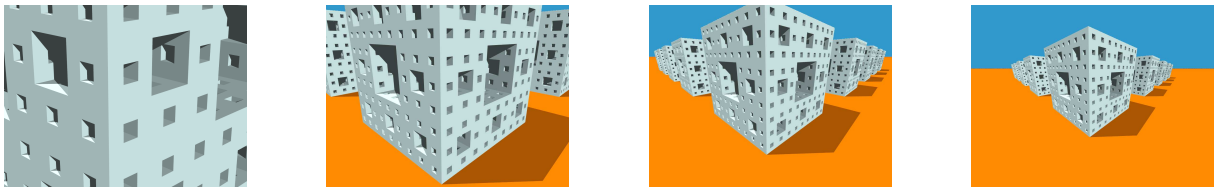


Abbildung 3.13: Blickachse nach unten, Aughöhe etwa auf halber Höhe der Würfel

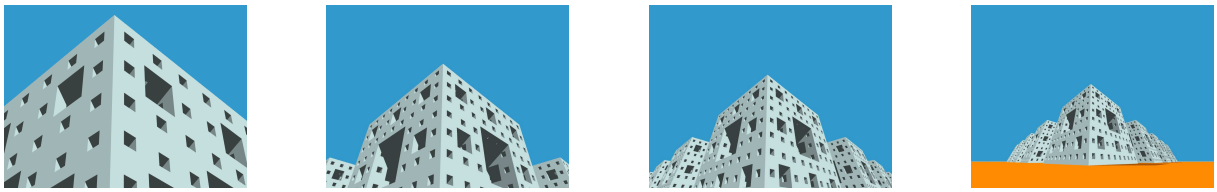


Abbildung 3.14: Blickachse nach oben, Froschperspektive

Abbildung 3.15: Blickachse waagrecht, Froschperspektive

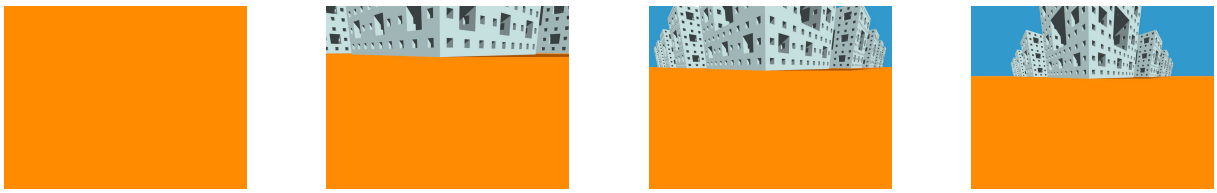


Abbildung 3.16: Blickachse nach unten, Froschperspektive

Abbildung 3.17: Blickachse nach oben, Maulwurfsperspektive

Kapitel 4

Darstellung von Kurven und Flächen mit POV-Ray

4.1 Kurven

4.1.1 Unter einer *parametrisierten Kurve* im $\mathbb{R}^3$ verstehen wir die Bildmenge einer stetig differenzierbaren Abbildung

$$k : I \to \mathbb{R}^3 : t \mapsto k(t) = (k_1(t), k_2(t), k_3(t)). \tag{4.1}$$

Dabei bezeichnet $I \neq \emptyset$ ein offenes Intervall in $\mathbb{R}$. Eine derartige Abbildung k wird auch als *Weg* oder *Parametrisierung* bezeichnet. Anschaulich lassen sich t als Zeitparameter und der Weg k als Bewegungsgleichungen interpretieren: Die Kurve entsteht als Bahn eines bewegten Punktes, der sich zum Zeitpunkt t an der Stelle $k(t)$ befindet.

Eine parametrisierte Kurve kann auf viele verschiedene Arten dargestellt werden. Zwei Wege $k : I \to \mathbb{R}^3$ und $\overline{k} : \overline{I} \to \mathbb{R}^3$ heißen *äquivalent*, falls sie sich nur um einen *zulässigen Parameterwechsel* unterscheiden, das heißt, es muss eine bijektive und in beiden Richtungen stetig differenzierbare Abbildung

$$\varphi : \overline{I} \to I$$

so geben, dass

$$\overline{k} = k \circ \varphi \quad \text{und} \quad \frac{\mathrm{d}\varphi}{\mathrm{d}\overline{t}} \neq 0 \quad \text{in ganz} \quad \overline{I}.$$

Eine *Kurve* ist dann eine Äquivalenzklasse von Wegen. In der Differentialgeometrie werden Eigenschaften von Kurven untersucht, eine konkrete Parametrisierung ist dabei nur Mittel zum Zweck.[1]

4.1.2 Manche Begriffe für Kurven lassen sich nur unter der Voraussetzung formulieren, dass wenigstens ein beschreibender Weg r-mal stetig differenzierbar ist mit $r \geq 2$. In diesem Fall muss die obige Definition äquivalenter Wege so verschärft werden, dass nur r-mal stetig differenzierbare Wege zugelassen werden. Ferner muss auch jeder zulässige Parameterwechsel r-mal stetig differenzierbar sein.

[1] Die hier angegebenen Ergebnisse aus der Differentialgeometrie können etwa in den Lehrbüchern [4] oder [10] nachgelesen werden.

4.1.3 Ist für $t_0 \in I$ die Bedingung

$$\dot{k}(t_0) = \frac{\mathrm{d}k}{\mathrm{d}t}(t_0) \neq 0$$

erfüllt, so wird $k(t_0)$ als ein *regulärer Punkt*. Die Gerade

$$k(t_0) + \mathbb{R}\dot{k}(t_0)$$

heißt die *Tangente* der Kurve im Punkt $k(t_0)$. Bei einem zulässigen Parameterwechsel kann sich zwar der Richtungsvektor $\dot{k}(t_0)$ der Tangente ändern, nicht jedoch die Tangente selbst.

4.1.4 Sind für $t_0 \in I$ die ersten beiden Ableitungsvektoren

$$\dot{k}(t_0), \quad \ddot{k}(t_0)$$

linear unabhängig, so wird die Ebene

$$k(t_0) + \mathbb{R}\dot{k}(t_0) + \mathbb{R}\ddot{k}(t_0)$$

als die *Schmiegebene* der Kurve im Punkt $k(t_0)$ bezeichnet. Bei einem zulässigen Parameterwechsel können sich zwar die Richtungsvektoren $\dot{k}(t_0)$ und $\ddot{k}(t_0)$ der Schmiegebene ändern, nicht jedoch die Schmiegebene selbst.

Sind hingegen $\dot{k}(t_0), \ddot{k}(t_0)$ linear abhängig und ist $\dot{k}(t_0) \neq 0$, so wird $k(t_0)$ ein *Wendepunkt* der Kurve genannt.

Die Gestalt einer regulär parametrisierten Kurve, die keine Wendepunkte besitzt, wird (bis auf gleichsinnige Kongruenztransformationen) durch zwei skalare Funktionen völlig bestimmt: Die *Krümmung* $\kappa : I \to \mathbb{R}$ ist durch

$$\kappa(t) = \frac{\|\dot{k}(t) \times \ddot{k}(t)\|}{\|\dot{k}(t)\|^3} > 0.$$

erklärt. Sie misst, vereinfacht ausgedrückt, wie sehr die Kurve von einer Geraden abweicht. Die Kurven mit in ganz I verschwindender Krümmung sind genau die Geraden. Die *Torsion* $\tau : I \to \mathbb{R}$ ist durch

$$\tau(t) = \frac{\det(\dot{k}(t), \ddot{k}(t), \dddot{k}(t))}{\|\dot{k}(t) \times \ddot{k}(t)\|^2}.$$

definiert. Sie misst, vereinfacht ausgedrückt, wie sehr die Kurve von einer ebenen Kurve abweicht. Eine Kurve hat genau dann in ganz I verschwindende Torsion, wenn sie in einer Ebene enthalten ist. Krümmung und Torsion ändern sich bei einem zulässigen Parameterwechsel nicht.

Bemerkung 4.1.5 In praktischen Anwendungen hat man es oft mit stetigen, aber nur *stückweise differenzierbaren* Wegen zu tun. Hier kann es in einzelnen Punkten eine *rechtsseitige Tangente* und eine davon verschiedene *linksseitige Tangente* geben. Sind diese beiden Tangenten gleich, so kann dennoch eine *rechtsseitige Schmiegebene* und eine davon verschiedene *linksseitige Schmiegebene* existieren. Die Gleichheit von Tangente und Schmiegebene in einem solchen Punkt garantiert übrigens noch nicht, dass der Übergang *krümmungsstetig* erfolgt.

4.2 Bilder von Raumkurven unter Projektionen

4.2.1 Wir betrachten im Folgenden eine Zentralprojektion

$$c : \mathcal{P} \setminus \{O\} \to \pi : X \mapsto OX \cap \pi,$$

und – mit den Bezeichnungen aus (4.1) – eine parametrisierte Kurve $k(I) \subset \mathbb{R}^3$, die keine Verschwindungspunkte enthält. Alle Ergebnisse dieses Abschnitts gelten sinngemäß auch für jede Parallelprojektion p, wobei die Einschränkung hinsichtlich der Verschwindungspunkte wegfällt.

Die zusammengesetzte Abbildung $c \circ k$ liefert eine parametrisierte Kurve $(c \circ k)(I)$ der Bildebene. Ist k ein r-mal stetig differenzierbarer Weg, so trifft dies auch auf $c \circ k$ zu, da c in einer Umgebung jedes Kurvenpunktes beliebig oft partiell differenzierbar ist. Wir notieren eine einfache Folgerung:

Satz 4.2.2 *Ist $k(t_0)$ ein regulärer Kurvenpunkt, so ist $(c\circ k)(t_0)$ ein regulärer Punkt der Bildkurve $(c \circ k)(I)$, sofern nur die Tangente im Punkt $k(t_0)$ nicht projizierend ist. In diesem Fall stimmt das perspektive Bild der Tangente im Punkt $k(t_0)$ mit der Tangente an das perspektive Bild der Kurve überein.*

Beweis. Unter dem totalen Differential von c an der Stelle $k(t_0)$ geht der Ableitungsvektor $\dot{k}(t_0)$ in den Ableitungsvektor $(c \circ k)^{\cdot}(t_0)$ über. Da $\dot{k}(t_0) \neq 0$ nach 3.2.13 nicht im Kern des totalen Differentials liegt, folgt $(c \circ k)^{\cdot}(t_0) \neq 0$. Die Aussage über die Tangenten ergibt sich daraus durch eine einfache Rechnung. □

Falls $k(t_0)$ ein regulärer Kurvenpunkt mit projizierender Tangente ist, so ist der Weg $c \circ k$ an der Stelle t_0 nicht regulär. Über den Verlauf der Bildkurve in der Nähe von $(c \circ k)(t_0)$ lässt sich dann keine allgemein gültige Aussage machen. In vielen Fällen wird die Bildkurve eine *Spitze* besitzen. Vgl. dazu Abbildung 4.1. Sie zeigt den Schatten[2] einer Raumkurve in einer Ebene, wobei sich die Lichtquelle auf der blau hervorgehobenen Tangente der Kurve befindet. Die im Bild hinzugefügte *Spitzentangente* tritt dabei als Schatten der Schmiegebene im Punkt $k(t_0)$ auf.

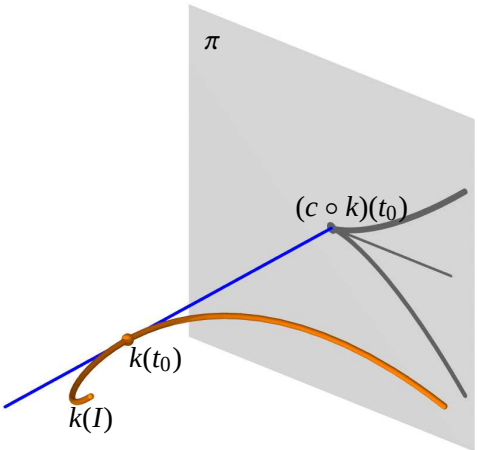


Abbildung 4.1: Spitze der Schattenkurve

[2] Für eine punktförmige Lichtquelle L entspricht der Schattenwurf auf eine Ebene $\varepsilon \not\ni L$ einer Zentralprojektion.

4.3 Kurven und POV-Ray

4.3.1 Um eine *parametrisierte Kurve* in POV-Ray darzustellen, wird auf ein *Sehnenpolygon* zurückgegriffen. Dazu werden im Parameterintervall I (oder gegebenfalls auch am Rand von I) Werte $a, b \in \mathbb{R}$ mit $a < b$ gewählt. Das abgeschlossene Intervall $[a, b]$ wird in $w \geq 1$ Teilintervalle mit den Randpunkten

$$a = t_0 < t_1 < \cdots < t_w = b$$

zerlegt. Dann wird das durch die Punkte $k(t_i)$ beschriebene Sehnenpolygon dargestellt. An die Stelle einer Strecke tritt in POV-Ray eine Sphäre um $k(t_i)$ und ein offener Zylinder von $k(t_i)$ nach $k(t_{i+1})$. Am Schluss kann noch eine Sphäre um $k(b)$ ergänzt werden. Alle diese Sphären und Zylinder haben denselben (kleinen) Radius. Vgl. dazu Abbildung 4.2 links. Sofern nur benachbarte Punkte $k(t_i), k(t_{i+1})$ hinreichend nahe beieinander liegen, ergibt sich im Bild der Eindruck einer glatten Kurve so wie Abbildung 4.2 rechts.

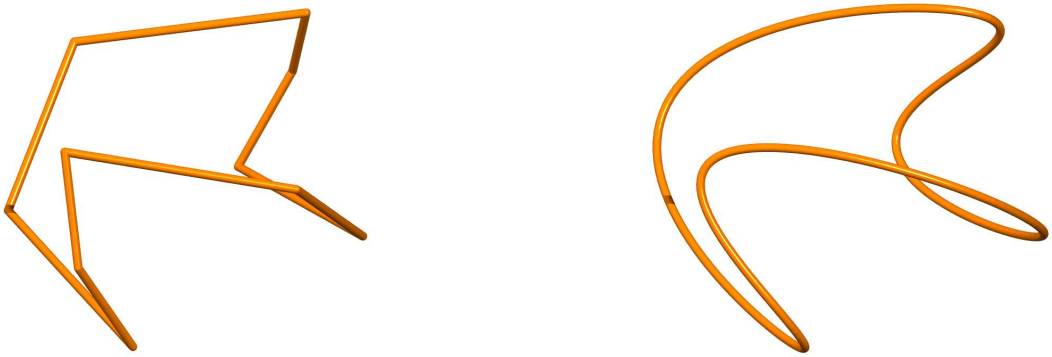


Abbildung 4.2: Approximation einer Kurve durch ein Sehnenpolygon

Die Wahl des Radius der Zylinder und Sphären ist dabei Geschmackssache. Wird ein sehr kleiner Radius gewählt, so entspricht dies eher dem Idealbild einer Strecke bzw. eines Punktes. Es werden dann aber Farbe und Oberfläche nur schlecht erkennbar sein. Wird ein größerer Radius gewählt, so besteht die Gefahr, dass das Bild eher an ein Rohr als an eine Kurve erinnert. Ein weiteres Problem tritt bei einer Zentralprojektion auf, da dann der „Radius“ der Bildkurve nicht konstant zu sein braucht. Vgl. dazu etwa Abbildung 3.7.

4.3.2 Eine in vielen Fällen ausreichende Strategie ist es, das Parameterintervall $[a, b]$ in w gleich lange Teile zu zerlegen, wobei w hinreichend groß ist. Bei dieser *äquidistanten Unterteilung* gilt also

$$t_i = a + \frac{i}{w}(b - a) \quad \text{für} \quad i \in \{0, 1, \ldots, w\}.$$

Diese Approximation liefert jedoch dann keine gute Näherung, falls die Strecken des Sehnenpolygons stark unterschiedliche Längen besitzen.

Beispiel 4.3.3 Mit der folgenden POV-Ray-Datei wurde Abbildung 4.2 rechts erstellt. Das Beispiel zeigt auch, wie die Parameterdarstellung einer Kurve mit Hilfe eines einfachen *Macros* in POV-Ray implementiert werden kann.

```
// Hans Havlicek Visualisierung

#include "colors.inc"
#include "transforms.inc"

// Trafo auf Rechtssystem
#declare Trafo = transform {matrix <1,0,0, 0,0,1, 0,1,0, 0,0,0>}


camera { orthographic location vtransform(<-4, 7, 5> Trafo)
         look_at vtransform(<0, 0, 0>  Trafo) transform Trafo}

light_source { <-50, 30, 50> color rgb 1  }
light_source { < 50, 60, 70> color rgb 0.8  shadowless}

background { color White }

#declare MeinFinish = finish { phong 0.9 ambient 0.3 }
#declare MeinPigment = pigment{ color Orange }

// Macro für die Parameterdarstellung der Kurve
// t kann nicht verwendet werden, da reservierte Bezeichnung
#macro Kurvenpunkt(T) 5*<sin(T)*cos(2*T),cos(T),sin(T)*cos(T)> #end

// Grundeinstellungen
#declare Teile=200; // Anzahl der Teilstrecken
#declare Tanf=0;    // Anfangsparameter
#declare Tend=2*pi; // Endparameter
#declare S=0;       // Anfangswert des Zählers S
#declare Dicke=0.1; // halbe Dicke der Kurve (Rohrfläche)

// Objekt erstellen
#while (S<Teile) // beachte das kleiner-Zeichen
        // Parameter zum Anfangs- und Endpunkt der aktuellen Strecke
        #declare T1 = Tanf+S/Teile*(Tend-Tanf);
        #declare T2 = Tanf+(S+1)/Teile*(Tend-Tanf);
        union{
                // Anfangspunkt und Strecke
                sphere{Kurvenpunkt(T1) Dicke}
                cylinder{Kurvenpunkt(T1) Kurvenpunkt(T2) Dicke open}
                pigment{MeinPigment} finish{MeinFinish}
                }
        #declare S = S+1; // Zähler erhöhen
#end // Ende der while-Schleife
```

```
// Bei Bedarf eine Kugel zum Abschluss ergänzen
```

4.3.4 Ein anderes Verfahren ist die *Unterteilung durch sukzessive Intervallhalbierung*. Sie geht von zwei Parameterwerten $t_1 < t_2$ aus. Ist der Abstand der Kurvenpunkte $k(t_1)$ und $k(t_2)$ kleiner oder gleich einer vorgegebenen Schranke $\varepsilon > 0$, so werden die Punkte mit einer Sehne verbunden. Ist das nicht der Fall, so wird das Intervall halbiert, also der Mittelpunkt

$$t_3 := \frac{1}{2}(t_1 + t_2)$$

berechnet. Dann werden die Überlegungen mit t_1, t_3 und t_3, t_2 wiederholt. Die Gesamtzahl $w+1$ der Teilungspunkte ergibt sich dabei in Abhängigkeit von ε. Zur Illustration wurden in Abbildung 4.3 links die Zylinder weggelassen, während rechts sowohl Kugeln als auch Zylinder verwendet wurden.

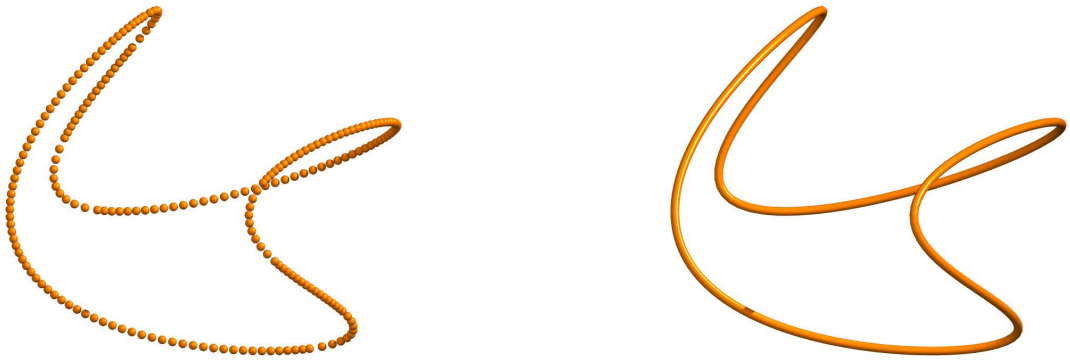


Abbildung 4.3: Approximation durch sukzessive Intervallhalbierung

Beispiel 4.3.5 Mit der folgenden POV-Ray-Datei wurde Abbildung 4.3 rechts erstellt. Dabei ist es wichtig, dass die *lokale Variable* `T3` innerhalb des Macros `Strecke( , )` nicht mit dem Schlüsselwort `#declare` sondern mit `#local` definiert werden muss. Sonst würde der rekursive Aufruf dieses Macros nicht funktionieren.

```
// Hans Havlicek Visualisierung

#include "colors.inc"
#include "transforms.inc"

// Trafo auf Rechtssystem
#declare Trafo = transform {matrix <1,0,0, 0,0,1, 0,1,0, 0,0,0>}


camera { orthographic location vtransform(<4, 7, 5> Trafo)
look_at vtransform(<0, 0, 0>  Trafo) transform Trafo}

light_source { <-50, 30, 50> color rgb 1  }
light_source { < 50, 60, 70> color rgb 0.8  shadowless}
```

```
background { color White }

#declare MeinFinish = finish { phong 0.9 ambient 0.3 }
#declare MeinPigment = pigment{ color Orange }

// Macro für die Parameterdarstellung der Kurve
// t kann nicht verwendet werden, da reservierte Bezeichnung
#macro Kurvenpunkt(T) 5*<sin(T)*cos(2*T),cos(T),sin(T)*cos(T)> #end


// Macro zum Darstellen der Kurve
#macro Strecke(T1,T2) // T1, T2 sind zwei Parameterwerte
        #if ( vlength(Kurvenpunkt(T1)-Kurvenpunkt(T2)) <= Epsilon )
                // falls Punkte zu T1 und T2 nahe genug:
                // Kugel im ersten Punkt und verbindender Drehzylinder
                union{    sphere{Kurvenpunkt(T1) Dicke }
                          cylinder{Kurvenpunkt(T1) Kurvenpunkt(T2)
                Dicke open }
                          pigment{MeinPigment} finish{MeinFinish}
                     }
        #else
                // halbiere das Intervall und rufe das Macro mit beiden
                // Teilstrecken auf
                #local T3 = 1/2*(T1+T2); // Halbierungspunkt
                                         // als lokale (!) Variable
                Strecke(T1,T3) // erster Aufruf
                Strecke(T3,T2) // zweiter Aufruf
        #end // End if
#end // Ende macro

// Grundeinstellungen
#declare Epsilon=0.3; // max Abstand für Sehnen
#declare Dicke=.1;  // halbe Dicke der Kurve (Rohrfläche)

// Objekt erstellen
// Aufruf des Macros in zwei Schritten, da geschlossene Kurve
Strecke(0,pi)
Strecke(pi,2*pi)

// Bei Bedarf eine Kugel zum Abschluss ergänzen
```

Bemerkung 4.3.6 Eine parametrisierte Kurve kann Verschwindungspunkte einer Zentralprojektion enthalten. (Bei einer Parallelprojektion tritt dieser Fall nicht auf.) Wir diskutieren die dabei auftretenden Möglichkeiten hier nicht systematisch.

Exemplarisch sei aber ohne Beweis der Fall eines Kreises in einer nicht projizierenden Ebene erwähnt. Auf sein perspektives Bild trifft genau einer der folgenden Fälle zu:

- Der Kreis enthält keinen Verschwindungspunkt. Das perspektive Bild des Kreises ist eine Ellipse.
- Der Kreis enthält genau einen Verschwindungspunkt V. Das perspektive Bild des Kreises ist eine Parabel. Die Achse dieser Parabel enthält den Fernpunkt $c(V)$.
- Der Kreis enthält genau zwei Verschwindungspunkte V_1, V_2. Das perspektive Bild des Kreises ist eine Hyperbel. Die perspektiven Bilder der Tangenten in den beiden Verschwindungspunkten sind die Asymptoten der Hyperbel. Die beiden Fernpunkte der Asymptoten sind die Punkte $c(V_1)$ und $c(V_2)$.

Vgl. dazu Abbildung 3.7 links: Die Bilder der Randkreise der grünen Kreisringe sind Ellipsen, bei den weißen Kreisringen treten aber Hyperbeln auf. Dazwischen könnte noch genau ein Kreis ergänzt werden, dessen perspektives Bild eine Parabel ist. Dieser Kreis ist aber in Abbildung 3.7 nicht dargestellt.

Falls eine Kurve Verschwindungspunkte oder Punkte im virtuellen Raum einer Zentralprojektion enthält, so wird sie unter POV-Ray nur teilweise oder (schlimmstenfalls) überhaupt nicht dargestellt. Wenn etwa das perspektive Bild eines Kreises eine Hyperbel ist, so wird in POV-Ray bestenfalls einer ihrer beiden Äste im Bild zu sehen sein. Vgl. dazu Abbildung 3.7.

4.4 Flächen

4.4.1 Es seien $M \neq \emptyset$ ein einfach zusammenhängendes Gebiet[3] im $\mathbb{R}^3$,

$$F : M \to \mathbb{R} : \boldsymbol{x} \mapsto F(\boldsymbol{x})$$

eine stetig differenzierbare Funktion und $c \in \mathbb{R}$ eine Konstante. Dann heißt

$$F^{-1}(c) = \{\boldsymbol{x} \in M \mid F(\boldsymbol{x}) = c\} \tag{4.2}$$

die *Niveaufläche* von F zum Wert c. Falls c nicht im Wertevorrat der Funktion F liegt, gilt $F^{-1}(c) = \emptyset$. Dieser Fall ist nicht von Interesse und wird im Folgenden ausgeschlossen.

Indem wir F durch die Funktion $\boldsymbol{x} \mapsto F(\boldsymbol{x}) - c$ ersetzen, können wir uns o. B. d. A. auf Niveauflächen zum Wert 0 beschränken.

4.4.2 Ist $F(\boldsymbol{p}) = 0$ und

$$(\operatorname{grad} F)|_{\boldsymbol{p}} = \left(\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \frac{\partial F}{\partial x_3}\right)\Bigg|_{\boldsymbol{p}} \neq 0,$$

so wird $\boldsymbol{p}$ ein *regulärer Punkt* der Niveaufläche $F^{-1}(0)$ genannt. Die Nullstellenmenge der Funktion

$$\mathbb{R}^3 \to \mathbb{R} : \boldsymbol{x} \mapsto (\operatorname{grad} F)|_{\boldsymbol{p}} \cdot (\boldsymbol{x} - \boldsymbol{p}) \tag{4.3}$$

ist die *Tangentialebene* der Niveaufläche im Punkt $\boldsymbol{p}$. Der Gradient $(\operatorname{grad} F)|_{\boldsymbol{p}}$ steht also auf die Tangentialebene orthogonal.

Nach dem Hauptsatz über implizite Funktionen lässt sich F in einer Umgebung von $\boldsymbol{p}$ nach einer der drei Koordinaten auflösen. Das bedeutet, dass sich der Durchschnitt der Niveaufläche

[3] Diese Voraussetzung reicht für unsere Zwecke.

$F^{-1}(0)$ mit einer passenden Umgebung von $\boldsymbol{p}$ als Graph einer Funktion darstellen lässt. Vgl. dazu 4.4.7.

4.4.3 Ist $F(\boldsymbol{p}) = 0$ und

$$(\operatorname{grad} F)|_{\boldsymbol{p}} = 0$$

so wird $\boldsymbol{p}$ ein *singulärer Punkt* der Niveaufläche $F^{-1}(0)$ genannt. Über den Verlauf der Niveaufläche in einer Umgebung von $\boldsymbol{p}$ lässt sich dann keine allgemein gültige Aussage machen. Unter Umständen ist $\boldsymbol{p}$ sogar ein isolierter Punkt; d. h. in einer Umgebung von $\boldsymbol{p}$ liegt kein anderer Punkt der Niveaufläche $F^{-1}(0)$. Letzteres trifft etwa für $F(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2$ im Ursprung zu.

4.4.4 Unter einer *parametrisierten Fläche* im $\mathbb{R}^3$ verstehen wir die Bildmenge einer stetig differenzierbaren Abbildung

$$f : U \to \mathbb{R}^3 : (u, v) \mapsto f(u, v) = (f_1(u, v), f_2(u, v), f_3(u, v)). \tag{4.4}$$

Dabei bezeichnet $U \neq \emptyset$ ein einfach zusammenhängendes Gebiet in $\mathbb{R}^2$. Eine derartige Abbildung f wird auch als *Weg* oder *Parametrisierung* bezeichnet.

Eine parametrisierte Fläche kann auf viele verschiedene Arten dargestellt werden. Zwei Wege $f : U \to \mathbb{R}^3$ und $\overline{f} : \overline{U} \to \mathbb{R}^3$ heißen *äquivalent*, falls sie sich nur um einen *zulässigen Parameterwechsel* unterscheiden, das heißt, es muss eine bijektive und in beiden Richtungen stetig differenzierbare Abbildung

$$\varphi : \overline{U} \to U$$

so geben, dass

$$\overline{f} = f \circ \varphi \quad \text{und} \quad \det J_\varphi \neq 0 \quad \text{in ganz} \quad \overline{U};$$

dabei bezeichnet $\det J_\varphi$ die Funktionaldeterminante von φ, also die Determinante der Jacobi-Matrix (Matrix der partiellen Ableitungen) von φ. Eine *Fläche* ist dann eine Äquivalenzklasse von Wegen. In der Differentialgeometrie werden Eigenschaften von Flächen untersucht, eine konkrete Parametrisierung ist dabei nur Mittel zum Zweck.

4.4.5 Manche Begriffe für Flächen lassen sich nur unter der Voraussetzung formulieren, dass wenigstens ein beschreibender Weg r-mal stetig differenzierbar ist mit $r \geq 2$. In diesem Fall muss die obige Definition äquivalenter Wege so verschärft werden, dass nur r-mal stetig differenzierbare Wege zugelassen werden. Ferner muss auch jeder zulässige Parameterwechsel r-mal stetig differenzierbar sein.

4.4.6 Sind für $(u_0, v_0) \in U$ die partiellen Ableitungsvektoren

$$\frac{\partial f}{\partial u}(u_0, v_0), \ \frac{\partial f}{\partial v}(u_0, v_0) \ \text{ linear unabhängig,}$$

so wird $f(u_0, v_0)$ als *regulärer Punkt* an der Stelle (u_0, v_0) bezeichnet. Die Ebene

$$f(u_0, v_0) + \mathbb{R}\,\frac{\partial f}{\partial u}(u_0, v_0) + \mathbb{R}\,\frac{\partial f}{\partial v}(u_0, v_0) \tag{4.5}$$

heißt die *Tangentialebene* der Fläche im Punkt $f(u_0, v_0)$. Der *Normalvektor*

$$\frac{\partial f}{\partial u}(u_0, v_0) \times \frac{\partial f}{\partial v}(u_0, v_0) \neq 0 \tag{4.6}$$

steht auf die Tangentialebene orthogonal. Bei einem zulässigen Parameterwechsel können sich zwar die Richtungsvektoren der Tangentialebene ändern, nicht jedoch die Tangentialebene selbst.

Ist $\boldsymbol{p} = f(u_0, v_0)$ ein regulärer Punkt, so gibt es eine in U enthaltene Umgebung U_0 von (u_0, v_0) derart, dass sich $f(U_0)$ als Graph einer Funktion darstellen lässt. Vgl. dazu 4.4.7.

4.4.7 Ein Sonderfall einer parametrisierten Fläche ist der *Graph* einer stetig differenzierbaren Funktion $z : U \to \mathbb{R}$, wobei $U \neq \emptyset$ ein zusammenhängendes Gebiet in $\mathbb{R}^2$ ist. Die Abbildung f aus (4.4) hat hier die spezielle Bauart[4]

$$f : U \to \mathbb{R}^3 : (u, v) \mapsto f(u, v) = (u, v, z(u, v)). \tag{4.7}$$

Es liegt daher nahe, $\mathbb{R}^2$ mit dem Unterraum $x_3 = 0$ von $\mathbb{R}^3$ vermöge $(u, v) = (u, v, 0)$ zu identifizieren. In diesem Sinne ist dann U ein Teil der Ebene $x_3 = 0$ und die Punkte der Fläche $f(U)$ erscheinen als Graph der Höhenfunktion z über den Punkten von U (Funktionsgebirge). Die Parametrisierung f ist für alle $(u_0, v_0) \in U$ regulär, da

$$\frac{\partial f}{\partial u}(u_0, v_0) = \left(1, 0, \frac{\partial z}{\partial u}(u_0, v_0)\right), \quad \frac{\partial f}{\partial v}(u_0, v_0) = \left(0, 1, \frac{\partial z}{\partial v}(u_0, v_0)\right)$$

offensichtlich linear unabhängig sind.

Die parametrisierte Fläche $f(U)$ stimmt mit der Niveaufläche $F^{-1}(0)$ der Funktion

$$F : U \times \mathbb{R} : (x_1, x_2, x_3) \mapsto z(x_1, x_2) - x_3 \tag{4.8}$$

überein; dabei wurden die Argumente von z nicht mit u, v sondern mit x_1, x_2 bezeichnet. Auch bei dieser Darstellung ist jeder Punkt der Niveaufläche $F^{-1}(0)$ regulär, da

$$\operatorname{grad} F = \left(\frac{\partial z}{\partial x_1}, \frac{\partial z}{\partial x_2}, -1\right) \neq 0$$

sogar in ganz $U \times \mathbb{R}$ erfüllt ist. Für einen Punkt $\boldsymbol{p} = (u_0, v_0, z(u_0, v_0))$ der Fläche $f(U) = F^{-1}(0)$ stimmt der Gradient von F an der Stelle $\boldsymbol{p}$ bis auf einen Vorzeichenfaktor mit den Kreuzprodukt (4.6) der partiellen Ableitungen von f an der Stelle (u_0, v_0) überein. Die beiden Beschreibungen der Tangentialebene aus (4.3) und (4.5) liefern also dieselbe Ebene.

4.4.8 Wie bereits zuvor erwähnt wurde, kann jede reguläre Niveaufläche und jede reguläre parametrisierte Fläche stückweise als Graph einer Funktion dargestellt werden. Beim Studium von lokalen Eigenschaften brauchen daher nur Funktionsgraphen betrachtet zu werden. Global gesehen reichen Funktionsgraphen jedoch nicht aus, um Flächen allgemeinerer Art zu beschreiben.

Bemerkung 4.4.9 In praktische Anwendungen hat man es oft mit stetigen, aber nur *stückweise differenzierbaren* Flächen und Wegen zu tun. Hier kann es in einzelnen Punkten mehrere Tangentialebenen geben.

[4] In (4.7) ist die dritte Koordinate ausgezeichnet, da dort die Funktion z auftritt. Analog könnte eine andere Koordinate ausgezeichnet werden.

4.5 Bilder von Flächen unter Projektionen

4.5.1 Wir betrachten im Folgenden eine Zentralprojektion

$$c : \mathcal{P} \setminus \{O\} \to \pi : X \mapsto OX \cap \pi,$$

und – mit den Bezeichnungen aus (4.4) – eine parametrisierte Fläche $f(U) \subset \mathbb{R}^3$, die keine Verschwindungspunkte enthält. Alle Ergebnisse dieses Abschnitts gelten lokal auch für Niveauflächen. Ferner lassen sich alle Ergebnisse dieses Abschnitts sinngemäß auf Parallelprojektionen übertragen, wobei die Einschränkung hinsichtlich der Verschwindungspunkte wegfällt.

Die zusammengesetzte Abbildung $c \circ f$ liefert eine parametrisierte Fläche $(c \circ f)(U)$ der Bildebene. Ist f ein r-mal stetig differenzierbarer Weg, so trifft dies auch auf $c \circ f$ zu, da c in einer Umgebung jedes Kurvenpunktes beliebig oft partiell differenzierbar ist.

4.5.2 Unter einer Zentralprojektion sind jene regulären Flächenpunkte $\boldsymbol{p} = f(u_0, v_0)$ ausgezeichnet, die eine projizierende Tangentialebene besitzen. Derartige Punkte werden *Umrisspunkte* oder *Konturpunkte* genannt. Die zusammengesetzte Abbildung $c \circ f$ ist an solchen Stellen (u_0, v_0) nicht regulär. Die Umrisspunkte einer Fläche bilden in vielen Fällen eine oder mehrere Kurven; bei manchen Flächen können aber auch isolierte Umrisspunkte auftreten. Die Bilder der Umrisspunkte sind im Allgemeinen Randpunkte der Bildfläche $(c \circ f)(U)$. Da die Bilder allfälliger Umrisskurven unter POV-Ray „automatisch“ entstehen, verzichten wir hier auf eine genauere Diskussion.

Beispiel 4.5.3 Abbildung 4.4 zeigt das Bild eines Torus unter verschiedenen Blickrichtungen. Als Umriss treten in allen drei Figuren zwei Kurven am Torus auf, die sich regulär parametrisieren lassen. In der linken Figur erinnern die Bilder der Umrisskurven an Ellipsen. Eine genauere Betrachtung zeigt aber, dass es sich nicht um Ellipsen handelt. In der mittleren Figur hat das

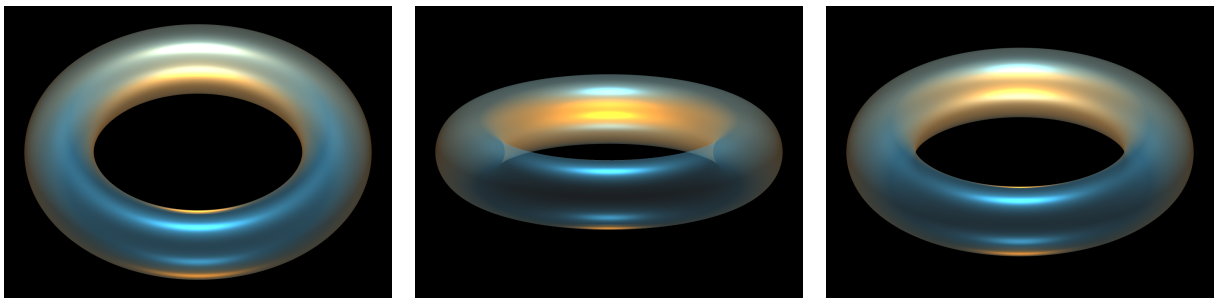


Abbildung 4.4: Umriss eines Torus bei verschiedenen Blickrichtungen

Bild der „inneren“ Umrisskurve vier Spitzen. In den betreffenden Punkten der Umrisskurve ist die Tangente eine Sehgerade, die Umrisskurve selbst hat dort einen regulären Punkt. Für das rechte Bild wurde die Blickrichtung so gewählt, dass die „innere“ Umrisskurve im Bild genau zwei Spitzen höherer Ordnung hat.[5] Der Torus wurde in allen drei Ansichten mit leichter Transparenz dargestellt, damit in allen Fällen die gesamten Umrisskurven gesehen werden können.

[5] Im Bild sieht es so aus, als würde die Kurve links und rechts je eine Ecke mit zwei verschiedenen Tangenten besitzen. Das ist nicht aber nicht richtig. Die Kurve verhält sich in jedem dieser Punkte wie die parametrisierte Kurve $t \mapsto (t^4, t^3)$ an der Stelle $t_0 = 0$: Für $t_0 = 0$ lässt sich durch stetiges Ergänzen genau eine „Ersatztangente“ definieren. Deren Richtungsvektor ist $(0, 1)$.

4.6 Niveauflächen und POV-Ray

4.6.1 Die Niveauflächen linearer Funktionen sind Ebenen. Deren Angabe wurde schon in 2.5.2 besprochen. *Quadriken* sind die Niveauflächen $F^{-1}(0)$ quadratischer Funktionen

$$F : \mathbb{R}^3 \to \mathbb{R} : (x,y,z) \mapsto Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J$$

Die Definition erfolgt als

```
quadric { <A,B,C>, <D,E,F>, <G,H,I>, J pigment {...} }
```

und kann von einer Clipping-Anweisung (etwa `clipped_by{ box ...}`) begleitet werden. Vgl. dazu Abbildung 4.5. Dort sind auch die Koordinatenachsen dargestellt, wobei diese ihren Schatten auf die Quadrik werfen.

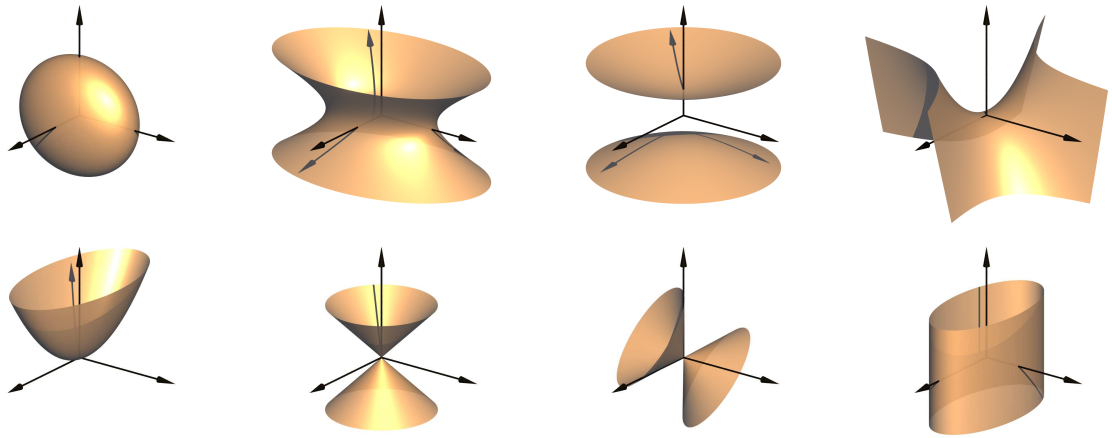


Abbildung 4.5: Quadriken

4.6.2 *Algebraische Flächen* werden durch eine Polynomfunktion F vom Grad n beschrieben. In POV-Ray werden alle Gradzahlen $n \leq 15$ unterstützt. Die Definition kann so erfolgen:

```
poly { n, <.,.,  ...  ,.,.> pigment {...}}
```

Dabei ist n der Grad des Polynoms und `<.,.,  ...  ,.,.>` die Koeffizientenliste bei den auftretenden Monomen. Etwa für kubische Flächen sind die Monome so geordnet:

$$x^3, x^2y, x^2z, x^2, xy^2, xyz, xy, xz^2, xz, x, y^3, y^2z, y^2, yz^2, yz, y, z^3, z^2, z, 1$$

Für Polynome vom Grad 3 und 4 gibt es auch die Schlüsselworte `cubic` und `quartic`. Deren Syntax ist so wie oben, bloß fällt die Gradzahl n weg. Alle mit `poly` erklärten Niveauflächen sind unbegrenzt.

Als Alternative gibt es den Befehl `polynomial`. Hier können die Koeffizienten in beliebiger Reihenfolge angegeben werden. Um etwa die kubische Fläche mit der Gleichung

$$x^3 + 5x^2y - 3yz + 1 = 0$$

festzulegen, wird geschrieben:

```
polynomial { 3, xyz(3,0,0):1, xyz(2,1,0):5, xyz(0,1,1):-3,
                               xyz(0,0,0):1  pigment { ... } }
```

Die erste Zahl ist der Grad des Polynoms, die drei Zahlen in der Klammer nach `xyz` sind die Exponenten i, j, k beim Term $x^i y^j z^k$, die Zahl nach dem Doppelpunkt ist der zugehörige Koeffizient. Alle Koeffizienten mit dem Wert 0 können weggelassen werden. Bei mehrfacher Belegung eines Koeffizienten gilt immer die letzte Definition.

Beispiel 4.6.3 Wir betrachten die Niveaufläche mit der Gleichung

$$xy + xz + yz - xyz = 0.$$

Sie enthält neben den drei Koordinatenachsen noch drei weitere Geraden. Die Definition in POV-Ray zur Erstellung von Abbildung 4.6 erfolgte so:

```
poly { 3, <0,0,0,0,0, -1,1,0,1,0, 0,0,0,0,1, 0,0,0,0,0>
clipped_by {box {<-1,-1,-1>*6, <1,1,1>*6}}
pigment {color Wheat }
finish {  phong 0.9 ambient 0.3}
}
```

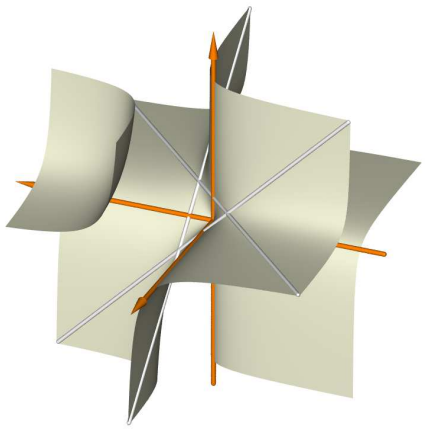


Abbildung 4.6: Kubische Fläche

Zur Ausleuchtung wurden mehrere schattenlose Lichtquellen verwendet. Daher kommt es zu keinen Glanzpunkten, obwohl `phong 0.9` gesetzt wurde.

Beispiel 4.6.4 Wir betrachten die Niveaufläche mit der Gleichung

$$x^4 + y^4 + z^4 - 6(x^2 + y^2 + z^2) + 17 = 0$$

(Abbildung 4.7). Die Definition in POV-Ray erfolgte so:

```
poly { 4, <1,0,0,0, 0,0,0,0, 0,-6,0,0, 0,0,0,0, 0,0,0,0,
1,0,0,0, 0,-6,0,0, 0,0,1,0, -6,0,17>
pigment {checker color rgb 0.7 color DarkBrown scale .25}
finish {phong .9 diffuse 0.9 }
```

Dabei wurden einen starke Lichtquelle im Ursprung und drei weitere (schwächere) Lichtquellen gewählt.

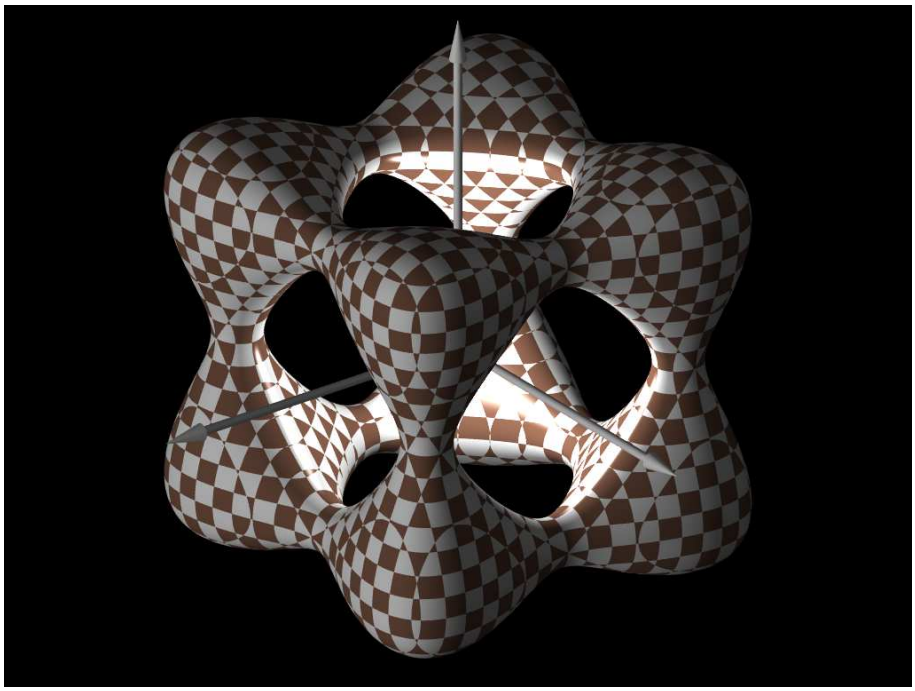


Abbildung 4.7: Fläche 4. Ordnung

4.6.5 *Niveauflächen* allgemeiner Art können über den Befehl `isosurface` erklärt werden. Im Gegensatz zu `poly` und `polynomial` kann die Funktion in ganz einfacher Weise angegeben werden. Wir erklären dies an einem Beispiel.

```
isosurface{
    function {sin(x)*sin(x)+sin(y)*sin(y)+sin(z)*sin(z)-1}
    open
    contained_by {box{ 1.9*(-x-y-z) 1.9*(x+y+z) } }
    max_gradient 1.631
    accuracy 0.001
    pigment{ Orange }
    finish{ phong 0.3 diffuse 0.6 ambient 0.2 }
    scale 2.8
          }
```

- Hier ist `function` die definierende Funktion F.
- Der Zusatz `open` verhindert, dass der Rand der Menge M (siehe unten) dargestellt wird.
- Nach Flächenpunkten wird nur innerhalb der in `contained_by` erklärten Punktmenge M gesucht, wobei hier ausschließlich `box` und `sphere` zugelassen sind. Wird diese Angabe weggelassen, tritt eine Standardeinstellung in Kraft, die meist nicht brauchbar sein wird.
- Die Zahl `max_gradient` kann auch weggelassen werden. Sie entspricht der maximalen Länge der Gradienten von F innerhalb von M. Diese Angabe kann aber die Berechnung beschleunigen. Bei jeder Berechnung des Bildes wird der angegebene Wert überprüft. Allfällige Abweichungen werden im *Messages-Fenster* angegeben und können in die POV-Ray-Datei übernommen werden.

- Der Wert für `accuracy` gibt die Genauigkeit der (ohnehin nur näherungsweisen) Berechnung der Flächenpunkte an. In der Testphase kann dieser Wert auch deutlich größer gewählt werden.

Vgl. dazu die Abbildung 4.8. Dort sind auch gut einige *Singularitäten* (nicht reguläre Punkte) zu sehen, in denen sich die Niveaufläche näherungsweise wie ein Kegel verhält.

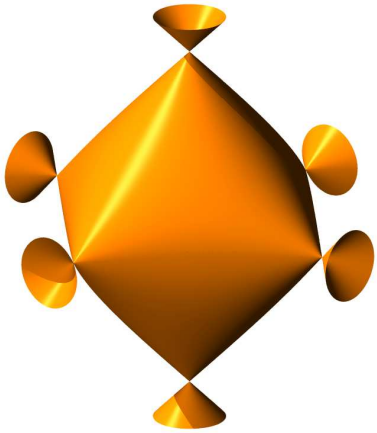


Abbildung 4.8: Niveaufläche mit Singularitäten

4.7 Parametrisierte Flächen und POV-Ray

4.7.1 Für die Darstellung *parametrisierten Flächen* stellt POV-Ray den Befehl `parametric` zur Verfügung. Die Berechnung von parametrisierten Flächen läuft damit aber erfahrungsgemäß *sehr langsam* ab. Ein weiterer Nachteil ist auch, dass es nicht möglich ist, die Fläche in Abhängigkeit von den Parametern (u, v) zu färben.

Wir stellen daher in Abschnitt 4.9 andere Möglichkeiten für die Darstellung von parametrisierten Flächen mit POV-Ray vor.

4.7.2 Obwohl wir die Verwendung des Befehls `parametric` nicht empfehlen, soll er dennoch an Hand einer einer Wendelfläche (vgl. Abbildung 4.9) besprochen werden:

```
parametric {
    function { u*cos(v) }, function { u*sin(v) }, function { 0.5*v }
    <-1,-pi/2>, <1,pi/2>
    contained_by{box{-2 2}}
    // accuracy 0.01 grober Wert in der Testphase
    // max_gradient 2
    pigment{ Orange }
    finish{ phong 0.3 diffuse 0.6 ambient 0.2 }
          }
```

- Die drei unter `function` angegebenen Funktionen legen die drei Koordinaten der Bildpunkte $f(u, v)$ fest.

- Das Parametergebiet ist immer ein achsenparalleles Rechteck der (u, v)-Ebene. Die beiden Vektoren legen die gegenüberliegende Ecken zu den Werten $(u_{\min}, v_{\min})$ und $(u_{\max}, v_{\max})$ fest.
- Flächenpunkte werden nur innerhalb der in `contained_by` erklärten Punktmenge M ermittelt, wobei hier ausschließlich `box` und `sphere` zugelassen sind. Wird diese Angabe weggelassen, tritt eine Standardeinstellung in Kraft, die meist nicht brauchbar sein wird.
- Die Zahl `max_gradient` kann auch weggelassen werden. Sie entspricht dem maximalen Betrag der partiellen Ableitungen der drei Koordinatenfunktionen von f nach u und v. Diese Angabe kann die Berechnung beschleunigen.
- Der Wert für `accuracy` gibt die Genauigkeit der (nur näherungsweisen) Berechnung der Flächenpunkte an. In der Testphase kann dieser Wert auch deutlich größer gewählt werden.

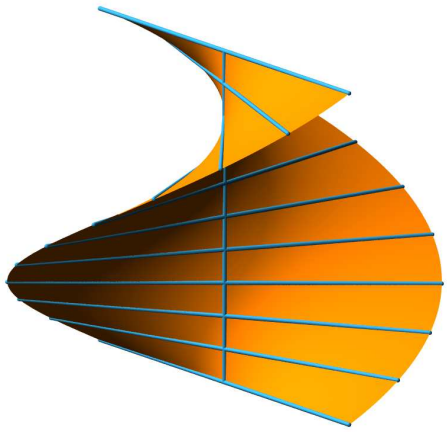


Abbildung 4.9: Wendelfläche

4.7.3 Bei der Darstellung von parametrisierten Flächen bietet es sich an, auch *Parameterlinien* zu ergänzen. So wurden in Abbildung 4.9 einige *u-Linien* (v = const) sowie die *v-Linie* zum Parameter $u = 0$ hinzugefügt. Dies steigert in vielen Fällen die Lesbarkeit einer Figur, wie etwa die Kurven in Abbildung 4.18 zeigen.

4.8 Übergänge zwischen Flächen

4.8.1 Wir diskutieren in diesem Abschnitt stetige, aber zumindest stückweise differenzierbare Flächen. Da wir keine Kenntnisse aus der Krümmungstheorie der Flächen voraussetzen, können wir die Ergebnisse nur an Hand von Beispielen illustrieren.

4.8.2 Wir behandeln zunächst die Reflexion an einer parametrisierten Fläche $f(U)$ relativ zu einer Zentralprojektion mit Augpunkt O. Dabei sei der Weg $f : U \to \mathbb{R}^3$ wenigstens einmal stetig differenzierbar und in ganz U regulär. Ferner sei O kein Punkt der Fläche. Die folgenden Überlegungen gelten sinngemäß auch für Niveauflächen und auch relativ zu einer Parallelprojektion.

Sei $k(I)$ eine parametrisierte Kurve. In den Punkten der *Reflexionslinie* $r \subset f(U)$ der Punktmenge $k(I)$ treffen jene Sehgeraden auf die Fläche $f(U)$, die nach erfolgter Spiegelung an $f(U)$ mindestens einen Punkt von $k(I)$ enthalten. Dabei ist die Spiegelung an $f(U)$ im Punkt $f(u_0, v_0)$ als Spiegelung an dessen Tangentialebene erklärt. Die *Flächennormale* in $f(u_0, v_0)$ hat den Richtungsvektor

$$\frac{\partial f}{\partial u}(u_0, v_0) \times \frac{\partial f}{\partial v}(u_0, v_0) \neq 0.$$

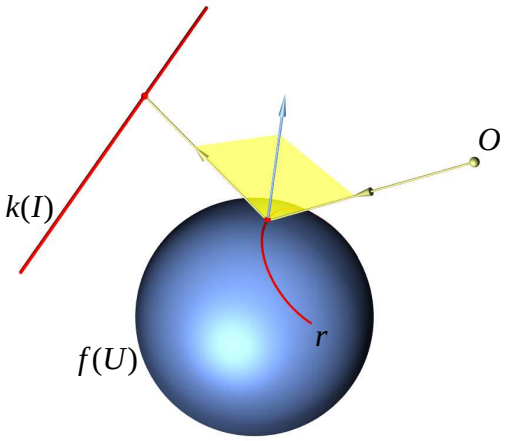


Abbildung 4.10: Reflexion einer Kurve an einer Fläche

Die Sehgerade durch $f(u_0, v_0)$ und ihre gespiegelte Gerade liegen mit der Flächennormalen in $f(u_0, v_0)$ in derselben Ebene und schließen mit dieser Flächennormalen denselben Winkel ein (Einfallswinkel = Ausfallswinkel). Vgl. Abbildung 4.10, wo $k(I)$ als Strecke gewählt wurde.[6]

In POV-Ray lassen sich Reflexionslinien nur bezüglich der gewählten Projektion leicht realisieren und nicht bezüglich eines vom Augpunkt verschiedenen Punktes. Das liegt daran, dass POV-Ray nur das *perspektive Bild* der Reflexionslinie bestimmt, aber keinen Zugriff auf die eigentliche Reflexionslinie (im Raum) gestattet.

Das nachfolgende Beispiel soll zeigen, wie mit Hilfe der Bilder von Reflexionslinien die Art des Übergangs zwischen Flächen sichtbar gemacht werden kann.

Beispiel 4.8.3 Die in Abbildung 4.11 dargestellte Drehfläche ist – von oben nach unten gesehen – aus einer kubischen Fläche (gelb), einem Kegelstumpf (violett), einer Kugelzone (blau), einem weiteren Kegelstumpf (orange) und einem Stück eines Torus (grün) zusammengesetzt. In der Fläche spiegeln sich sechs Strecken (dünne Drehzylinder), die selbst im Bild nicht zu sehen sind. An Hand der auftretenden Reflexionen lässt sich gut erkennen, wie die Übergänge zwischen den einzelnen Teilen der Fläche gestaltet wurden:

- Der Übergang von der kubische Fläche auf den anschließenden Kegelstumpf erfolgt *krümmungsstetig*. Die beiden Flächen haben in jedem Punkt des Übergangskreises nicht nur dieselbe Tangentialebene sondern auch dieselben Hauptkrümmungsrichtungen und in jeder dieser Richtungen dieselbe Hauptkrümmung.[7] Dies hat zur Folge, dass sich die zusammengesetzte Fläche in einer Umgebung jedes Punktes des Übergangskreises als

[6] Die Reflexionslinie r wurde mit Hilfe von Maple berechnet.

[7] Anders formuliert: Die beiden Weingarten-Abbildungen stimmen an dieser Stelle überein.

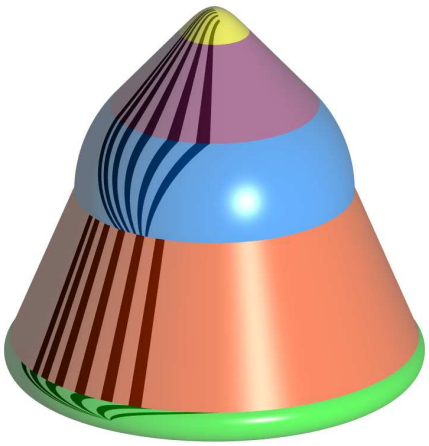


Abbildung 4.11: Reflexionslinien an einer Drehfläche

Graph einer *zweimal stetig differenzierbaren Funktion* beschreiben lässt. Zur Berechnung der Reflexionslinien muss diese Funktion *einmal* differenziert werden, da die Normalvektoren in die Rechnung eingehen. Das bewirkt, dass die Reflexionslinien *einmal stetig differenzierbar* sind. Sie verlaufen hier ohne Knick von der einen auf die andere Fläche.

- Der Kegelstumpf geht nur *berührend* in die Kugelzone über. Die beiden Flächen haben in jedem Punkt des Übergangskreises zwar dieselbe Tangentialebene, aber nicht dieselben Hauptkrümmungen[8]. Das bewirkt, dass sich die Fläche in einer Umgebung jedes Punktes des Übergangskreises nur als Graph einer *einmal stetig differenzierbaren Funktion* beschreiben lässt. Dementsprechend sind die Reflexionslinien in den Übergangspunkten nur mehr stetig. Sie gehen in der Tat mit einem Knick von der einen auf die andere Fläche über.

- Die Kugelzone schließt nur *stetig* an den nächsten Kegelstumpf an. Die beiden Flächen haben in jedem Punkt des Übergangskreises verschiedene Tangentialebenen. Das bewirkt, dass sich die Fläche in einer Umgebung jedes Punktes des Übergangskreises nur als Graph einer *stetigen Funktion* beschreiben lässt. Dementsprechend sind die Reflexionslinien hier nicht mehr stetig. Sie gehen mit einem Sprung von der einen auf die andere Fläche über.

- Der Übergang vom Kegel auf den Torus ist ebenfalls nur *stetig* und führt so wie zuvor auf Sprungstellen der Reflexionslinien.

An einer Sprungstelle kann es auch sein, dass ein *Endpunkt einer Reflexionslinie* vorliegt, da es nur in einer der beiden Flächen zu einer Spiegelung kommt.

4.8.4 Wir gehen von einem Beleuchtungsmodell mit einer punktförmigen Lichtquelle L aus, bei dem sich die Beleuchtungsstärke mit der Entfernung nicht ändert. Diese Annahme ist immer dann gerechtfertigt, falls eine *Parallelbeleuchtung* vorliegt (L ist also ein Fernpunkt) oder die Lichtquelle „weit entfernt“ ist. Wir betrachten eine parametrisierte Fläche $f(U)$. Dabei sei der Weg $f : U \to \mathbb{R}^3$ wenigstens einmal stetig differenzierbar, in ganz U regulär und L kein Punkt der Fläche. Die folgenden Überlegungen gelten sinngemäß auch für Niveauflächen.

[8] Eine Hauptkrümmung des Kegels ist 0, während die Hauptkrümmungen der Sphäre von 0 verschieden sind.

Die *Helligkeitsfunktion* längs des Weges f relativ zur Lichtquelle L ist die Funktion

$$H : U \to [0,1] : (u,v) \mapsto \left| \cos \sphericalangle \left(\frac{\partial f}{\partial u}(u,v) \times \frac{\partial f}{\partial v}(u,v), l(u,v) \right) \right|,$$

wobei $l(u,v)$ einen Richtungsvektor des Lichtstrahls durch den Flächenpunkt $f(u,v)$ bezeichnet. Für jedes $w \in [0,1]$ heißt

$$f(H^{-1}(w)) = \{ f(u,v) \mid (u,v) \in U,\ H(u,v) = w \}$$

die *Isophote* von $f(U)$ zur Helligkeit w. Isophoten können aus Flächenkurven, einzelnen Flächenpunkten oder ganzen Flächenstücken bestehen; letzteres ist genau dann der Fall, falls diese Flächenstücke in einer Ebene liegen.[9]

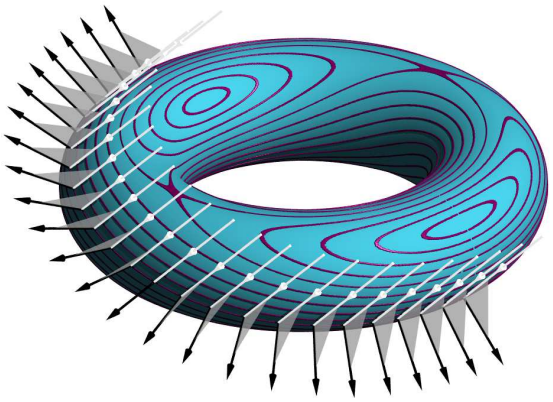


Abbildung 4.12: Isophote am Torus mit Lichtstrahlen und Normalen

Die Abbildung 4.12 zeigt Isophoten auf einem Torus.[10] Dabei wurden längs einer Isophote Richtungsvektoren für die Lichtstrahlen und für die Flächennormalen eingetragen. Da sich der Torus durch einen beliebig oft stetig differenzierbaren Weg parametrisieren lässt, sind die Isophoten (zumindest lokal) als Bildmengen beliebig oft differenzierbare Wege erfassbar. Im Bild sind auch vier Punkte erkennbar, in denen die Helligkeitsfunktion stationär ist: In zwei der Punkte ergibt sich ein (sogar absolutes) Maximum, da der Lichtstrahl dort orthogonal auf den Torus auftrifft. Die anderen beiden Punkte sind Sattelpunkte der Helligkeitsfunktion.

Eine spezielle Isophote ist die *Eigenschattengrenze* von $f(U)$. Sie gehört zur Helligkeit $w = 0$. Die betreffenden Flächenpunkte befinden sich im *Streiflicht* bezüglich L. Die Eigenschattengrenze ist genau der Umriss der Fläche bezüglich Projektion aus der Lichtquelle. In Abbildung 4.13 ist die Eigenschattengrenze der Rand der Mondsichel.

POV-Ray stellt Oberflächen gemäß einem Beleuchtungsmodell dar, welches über die oben beschriebene Helligkeitsfunktion H weit hinausgeht. Es gestattet aber keinen Zugriff auf die einzelnen Isophoten.

Kurvenförmige Isophoten verhalten sich beim Übergang von einer Fläche auf eine andere Fläche so wie Reflexionslinien (ohne Knick, mit Knick, mit Sprungstelle). Daher können

[9] Die hier beschriebene Helligkeitsfunktion bedarf in der Praxis noch einiger Modifikationen. Sie berücksichtigt etwa nicht, ob das von der Lichtquelle ausgesandte Licht einen Flächenpunkt auch tatsächlich erreicht.

[10] Die Punkte der Isophoten wurden numerisch mit MATLAB berechnet, in eine Datei geschrieben und zur Darstellung in POV-Ray aus dieser Datei eingelesen.

Abbildung 4.13: Eigenschattengrenze am Mond (Photo: Georg Glaeser)

Isophoten dazu dienen, die Übergänge zwischen Flächen besser sichtbar zu machen. Die nachfolgenden Beispiele illustrieren das an Hand von Eigenschattengrenzen.

Beispiel 4.8.5 Wir betrachten bei Parallelbeleuchtung einen Drehkegel, an den ein Drehzylinder anschließt. Der Übergang zwischen den Flächen ist nur stetig. Daher hat die aus vier Strecken bestehende Schattengrenze Sprungstellen. In Abbildung 4.14 sind nur zwei dieser vier Strecken zu sehen. Die beiden anderen Schattengrenzen befinden sich auf der Rückseite der Flächen.

In Abbildung 4.15 ist schematisch dargestellt, wie diese Schattengrenze aussieht, wenn sie in Richtung der Lichtstrahlen auf eine Ebene projiziert wird. Statt von Schattengrenzen könnte dann auch von Umrissen gesprochen werden. Die Projektion der Schattengrenze am Kegel ist eine Strecke der Bildebene. Diese geht berührend in die Projektion des Übergangskreises (eine Ellipse der Bildebene) über. Im Raum berühren einander die Schattengrenze des Kegels und der Übergangskreis aber nicht. Die Projektion *glättet* diesen Übergang im Bild! In gleicher Weise schließt die Projektion der Schattengrenze am Zylinder berührend an die Ellipse an, obwohl im Raum keine Berührung vorliegt. Insgesamt fällt der Sprung zwischen den beiden Schattengrenzen im Bild kaum auf, da er durch einen Bogen des Übergangskreises geschlossen wird.[11]

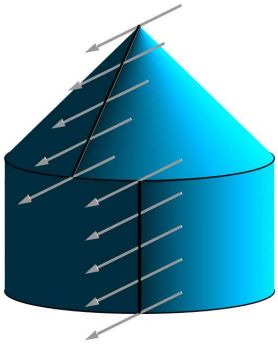


Abbildung 4.14: Schattengrenze mit Sprungstelle

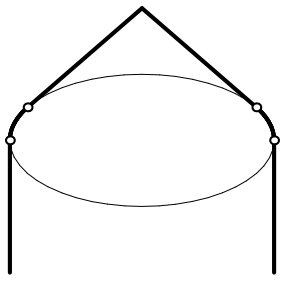


Abbildung 4.15: Umriss mit Sprungstellen

[11] Abbildung 4.15 ist eigentlich überflüssig. Der Umriss in Abbildung 4.14 zeigt nämlich ohnehin dieselbe Situation. Bloß sind die Bilder der Sprungstellen dort nicht hervorgehoben.

Beispiel 4.8.6 Wir betrachten bei Parallelbeleuchtung eine Halbkugel, an die ein Drehzylinder berührend anschließt. Der Übergang zwischen den Flächen ist nur einmal stetig differenzierbar. Daher hat die aus einem Halbkreis und zwei Strecken bestehende Schattengrenze einen Knick. In Abbildung 4.16 sind nur ein Teil des Halbkreises und eine dieser Strecken zu sehen. Wird diese Schattengrenze in Richtung der Lichtstrahlen orthogonal projiziert, so entsteht in der Bildebene ein Halbkreis, an den zwei Strecken berührend anschließen. Dazu gibt es keine eigene Abbildung. Vielmehr sei auf das Bild des Umrisses in Abbildung 4.16 verwiesen. Dieser spiegelt genau diese Situation wider. Auch hier gilt: Die Projektion in Richtung der Lichtstrahlen glättet diesen Übergang im Bild.

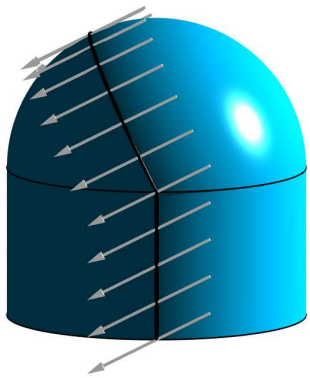


Abbildung 4.16: Schattengrenze mit Knick

4.9 Triangulierte Flächen und POV-Ray

4.9.1 Wir gehen von einem rechteckigen Parameterbereich $U = [a_u, b_u] \times [a_v, b_v]$ und einem stetig differenzierbaren Weg

$$f : U \to \mathbb{R}^3$$

aus, der in ganz U regulär ist. Zur Darstellung der parametrisierten Fläche $f(U)$ in POV-Ray wird die Fläche durch einbeschriebene Dreiecke approximiert. Dazu werden die abgeschlosse-

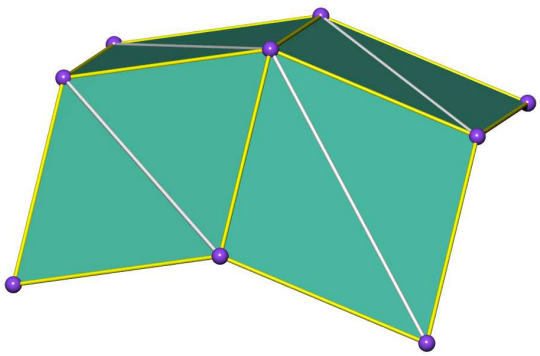


Abbildung 4.17: Triangulierung einer parametrisierten Fläche

nen Intervalle $[a_u, b_u]$ und $[a_v, b_v]$ in Teilintervalle mit den Randpunkten

$$a_u = u_0 < u_1 < \cdots < u_r = b_u \quad (r \geq 1)$$
$$a_v = v_0 < v_1 < \cdots < v_s = b_v \quad (s \geq 1)$$

zerlegt, was zu einer Zerlegung von U in Rechtecke führt. Jedes dieser Rechtecke wird dann längs einer Diagonalen in zwei Dreiecke geteilt. Wir erhalten so eine *Triangulierung* von U. Die f-Bilder der Ecken dieser Dreiecke liefern dann die gesuchte Approximation von $f(U)$ durch Dreiecksflächen, also eine *triangulierte Fläche*. In Abbildung 4.17 sind acht derartige Dreiecke dargestellt. Dabei wurde jedes Rechteck

$$(u_i, v_j),\ (u_{i+1}, v_j),\ (u_{i+1}, v_{j+1}),\ (u_i, v_{j+1}),$$

in gleicher Weise durch die Diagonale von (u_{i+1}, v_j) nach (u_i, v_{j+1}) geteilt.

4.9.2 Die oben beschriebene Approximation durch Dreiecke ist sehr einfach umzusetzen, hat aber einen ganz entscheidenden Nachteil: Da die Dreiecke nur stetig aneinander anschließen, wird im Allgemeinen an jeder Dreiecksseite eine *Unstetigkeitsstelle* der Helligkeitsfunktion auftreten. Wir illustrieren das an einem Beispiel:

In Abbildung 4.18 ist eine triangulierte *Dupinsche Zyklide* dargestellt. Die Fläche hat mit einem Torus viele Eigenschaften gemeinsam, da sie aus diesem durch eine Inversion an einer Sphäre entsteht. Obwohl eine sehr feine Unterteilung in $2 \cdot 120 \times 240$ Dreiecke gewählt wurde, sind die Unstetigkeitsstellen der Helligkeitsfunktion gut erkennbar.

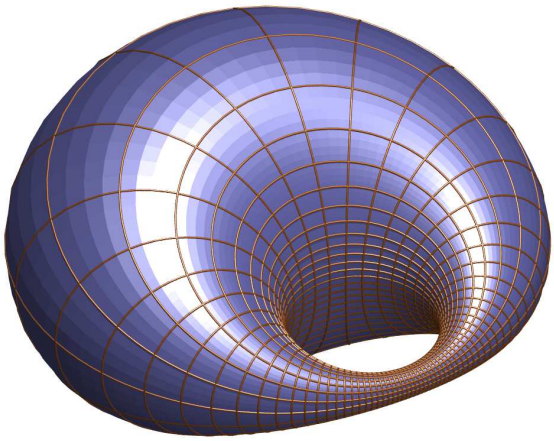


Abbildung 4.18: Dupinsche Zyklide mit Parameterlinien

4.9.3 Ein *Dreieck* lässt sich in POV-Ray mit dem Befehl

```
triangle { <...>, <...>, <...> pigment {...} finish {...}}
```

definieren, wobei an den ersten drei Stellen die Ecken des Dreiecks, also Vektoren aus $\mathbb{R}^3$ einzusetzen sind. Bei parametrisierten Flächen werden diese Ecken zweckmäßig in zwei verschachtelten Schleifen berechnet. Vgl. dazu Beispiel 4.9.6.

Als Alternative zu `triangle` bietet POV-Ray den Datentyp `smooth_triangle` an. Die Syntax lautet

```
smooth_triangle { <...>, <...>, <...>, <...>, <...>, <...>
                  pigment {...} finish {...}},
```

wobei an den ersten sechs Stellen Vektoren aus $\mathbb{R}^3$ einzusetzen sind.

- Der erste, dritte und fünfte Vektor bestimmen die Ecken P_1, P_2, P_3 des Dreiecks.
- Der zweite, vierte und sechste Vektor sind *fiktive Normalvektoren* $\boldsymbol{n}_1, \boldsymbol{n}_2, \boldsymbol{n}_3 \neq 0$ in den Ecken P_1, P_2, P_3.

Ein fiktiver Normalvektor kann beliebig gewählt werden, der Nullvektor muss aber unbedingt vermieden werden. Die Länge des Vektors ist belanglos, nur die Richtung ist relevant. POV-Ray stellt die Helligkeit eines solchen *geglätteten Dreiecks* dann in stetiger Weise relativ zu den fiktiven Normalvektoren dar. Das Dreieck wird also so behandelt, als ob es eine krumme Fläche wäre.

In Abbildung 4.19 sind (von links nach rechts gesehen) drei geglättete Dreiecke und ein gewöhnliches Dreieck zu sehen. Ganz links wurden die fiktiven Normalvektoren linear unabhängig gewählt, was zu einer nicht konstanten, aber stetigen Helligkeit führt. Dann wurden die fiktiven Normalvektoren parallel, aber nicht orthogonal zum Dreieck gewählt. Das ergibt ein Dreieck mit konstanter, aber „falscher“ Helligkeit. Schließlich wurde in jeder Ecke tatsächlich ein Normalvektor des Dreiecks gewählt. Das ergibt ein Dreieck mit der richtigen (konstanten) Helligkeit. Derselbe Effekt wurde im vierten Bild einfacher erreicht: Das Dreieck wurde ohne fiktive Normalvektoren mit `triangle` dargestellt.

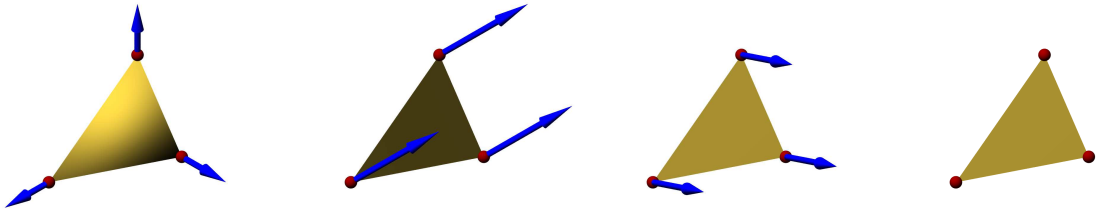


Abbildung 4.19: Dreiecke mit und ohne Normalvektoren zur Farbglättung

4.9.4 Bei einer durch Dreiecke approximierten Fläche kann `smooth_triangle` dazu verwendet werden, Sprünge in der Helligkeit zu vermeiden. Dazu wird in jeder Ecke $f(u_i, v_j)$ eines Dreiecks der Normalvektor

$$\frac{\partial f}{\partial u}(u_i, v_j) \times \frac{\partial f}{\partial v}(u_i, v_j) \neq 0$$

der Fläche verwendet. Sind die Normalvektoren nicht oder nur schwierig berechenbar, so kann auch auf den Vektor

$$(f(u_i, v_j) - f(u_{i+1}, v_j)) \times (f(u_i, v_j) - f(u_i, v_{j+1}))$$

als Näherung zurückgegriffen werden. Dabei muss über den Rand des Parameterrechtecks hinaus gerechnet werden. Falls dies nicht möglich ist, muss für Punkte am Rand des Parameterrechtecks eine andere Näherung gewählt werden. Wichtig ist nur, dass jedem Punkt $f(u_i, v_j)$ immer nur ein einziger fiktiver Normalvektor zugeordnet wird.

In Abbildung 4.20 zeigt nochmals die triangulierte *Dupinsche Zyklide*, jetzt aber mit geglätteten Dreiecken dargestellt. Die Unterteilung stimmt dabei mit jener aus Abbildung 4.20 überein.

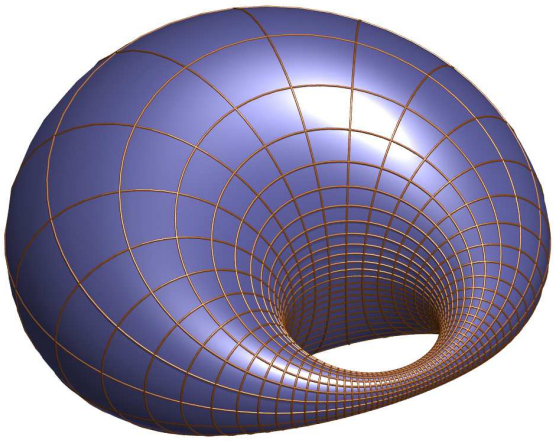


Abbildung 4.20: Dupinsche Zyklide mit Parameterlinien (geglättete Dreiecke)

4.9.5 Eine in vielen Fällen ausreichende Strategie zur Darstellung einer parametrisierten Fläche ist es, die Intervalle $[a_u, b_u]$ und $[a_v, b_v]$ in $r \geq 1$ bzw. $s \geq 1$ gleich lange Teile zu teilen, wobei r und s hinreichend groß sind (*äquidistante Unterteilung*). Es gilt also

$$u_i = a_u + \frac{i}{r}(b_u - a_u) \quad \text{für} \quad i \in \{0, 1, \ldots, r\}$$

und analog

$$v_j = a_v + \frac{j}{s}(b_v - a_v) \quad \text{für} \quad j \in \{0, 1, \ldots, s\}.$$

Diese Approximation liefert jedoch dann keine gute Näherung, falls die Seiten der einbeschriebenen Dreiecke stark unterschiedliche Längen besitzen. Letzteres ist zum Beispiel in den Abbildungen 4.18 und 4.20 der Fall.

Beispiel 4.9.6 Gegeben sei die Funktion

$$F : \mathbb{R}^3 \to R : (x_1, x_2, x_3) \mapsto x_3(x_1^2 + x_2^2) - x_1.$$

Die Niveaufläche $F^{-1}(0)$ wird als *Müllersche Fläche*[12] bezeichnet. Wir erfassen einen Teil der Niveaufläche durch die Parametrisierung

$$f : \left[\frac{1}{4}, 5\right] \times [0, 2\pi] \to \mathbb{R}^3 : (u, v) \mapsto \left(\frac{u}{2}(1 + \cos v), \frac{u}{2}(1 + \sin v), \frac{1}{u}\right)$$

und einen weiteren Teil der Fläche durch

$$\tilde{f} : \left[-\frac{1}{4}, -5\right] \times [0, 2\pi] \to \mathbb{R}^3 : (u, v) \mapsto \left(\frac{u}{2}(1 + \cos v), \frac{u}{2}(1 + \sin v), \frac{1}{u}\right).$$

Die POV-Ray-Datei zur Erstellung dieses Bildes ist am Ende dieses Beispiels samt Kommentaren vollständig wiedergegeben, wobei zwei Varianten zur Färbung eingebaut wurden. In Variante 1 wird die Fläche in einer einzigen Farbe gefärbt und mit Hilfe von `smooth_triangle` dargestellt. Die zweite Variante (mit dem Farbenkreis) wird in Beispiel 4.9.7 beschrieben. Wir fassen hier die wichtigsten Punkte zusammen.

- Zur Triangulierung der Fläche gehen wir wie in 4.9.5 vor.

[12] Emil Müller (1861–1927).

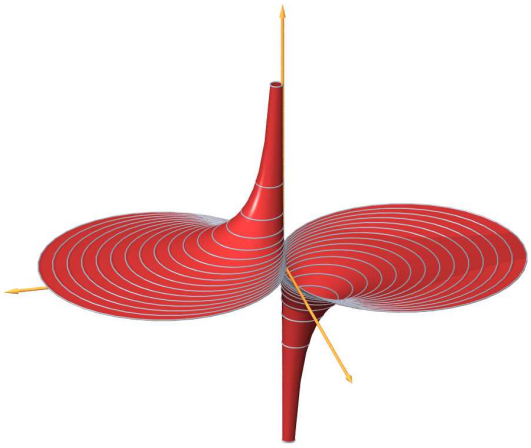


Abbildung 4.21: Müllersche Fläche

- Die Parametrisierung der Fläche und die Berechnung der Normalvektoren erfolgt mit Hilfe von Macros.
- Die Berechnung der einzelnen Dreiecke samt den Normalvektoren findet in zwei verschachtelten Schleifen mit Hilfe der zuvor beschriebenen Macros statt.
- Alle geglätteten Dreiecke werden durch `union`-Befehle zusammengefasst.
- Zusätzlich wurden einige v-Linien ergänzt. Hier läuft alles wie in Beispiel 4.3.3 ab. Dabei wurde für die v-Linien die Teilung von U in v-Richtung übernommen, damit die Approximation der Kurven mit der Approximation der Fläche zusammenpasst.
- Es wird auch das Koordinatensystem dargestellt.

Um die Lesbarkeit der Datei zu erhöhen, besteht der erste Teil der Datei nur aus Deklarationen. Die eigentliche Szene findet sich ganz am Ende der Datei. Dabei wird der durch $\tilde{f}$ beschriebene Teil der Fläche durch die Spiegelung am Ursprung aus dem durch f beschriebenen Teil gewonnen.

```
// Hans Havlicek Visualisierung
// Nach einer Vorlage von Boris Odehnal

#version 3.7;
global_settings { assumed_gamma 2.0 }

#include "transforms.inc"
#include "colors.inc"

#declare Trafo = transform {matrix <1,0,0, 0,0,1, 0,1,0, 0,0,0>}

// Kamera
camera {orthographic location vtransform(<2,8,4> Trafo)
        look_at <0,0,0> transform{Trafo}
        }

// Hintergrund
```

```
background {color White}

// Lichtquellen, nur eine mit Schatten
light_source {<100,0,20> color rgb 0.7}
light_source {<0,100,20> color rgb .5 shadowless}
light_source {<2,4,8>*100 color rgb 1.5 shadowless}

// Koordinatensystem Einstellungen
#declare Laenge = 5.5;
#declare Dicke = .03;
#declare Pfeil = .3;
#declare Basis = 2*Dicke;
#declare AchsenFarbe = pigment {color Gold}
#declare AchsenFinish = finish {phong 0.9 ambient 0.5}

// Koordinatensystem definieren
#declare Koordinatensystem =
union{
// Kegel und Zylinder
        cylinder {0 Laenge*x Dicke}
        cylinder {0 Laenge*y Dicke}
        cylinder {0 Laenge*z Dicke}
cone {Laenge*x Basis (Laenge+Pfeil)*x 0}
cone {Laenge*y Basis (Laenge+Pfeil)*y 0}
cone {Laenge*z Basis (Laenge+Pfeil)*z 0}
        sphere {0 Dicke}
pigment {AchsenFarbe} finish {AchsenFinish}
      }

// Fläche Einstellungen
// U = [Uanf,Uend] x [Vanf,Vend]
#declare Uanf = 1/4;
#declare Uend = 5;
#declare Vanf = 0;
#declare Vend = 2*pi;

// Parameterdarstellung
#macro Flaeche(U,V)
  < U/2*(1+cos(V)), U/2*sin(V), 1/U>
#end

// Parameterdarstellung des Normalenfeldes (nicht normiert)
#macro Normale(U,V) vcross(<1/2*(1+cos(V)),1/2*sin(V), -1/(U*U)>,
                           <      -sin(V) ,    cos(V),        0>)
#end
```

```
// Diskretisierung: Teilung und Inkrement in U- und V-Richtung
#declare UTeile = 40;
#declare VTeile = 40;
#declare DeltaU = (Uend-Uanf)/UTeile;
#declare DeltaV = (Vend-Vanf)/VTeile;

// Farbe und Finish
#declare FlaechenFarbe = pigment {color Brown}
#declare FlaechenFinish = finish {phong 0.9 ambient 0.3}

// Farbenkreis
#macro FarbenKreis(V)
 color rgb
 1/4*<1+2*cos(V), 1-sqrt(3)*sin(V)-cos(V), 1+sqrt(3)*sin(V)-cos(V)>
#end

// Fläche definieren
#declare MuellerFlaeche =
union {
// äussere Schleife von 0 bis UTeile-1, wegen p2 und p3
#for (USchritt,0,UTeile-1)
     #declare U = Uanf+USchritt*DeltaU;
// innere Schleife 0 bis VTeile-1, wegen p3 und p4
#for (VSchritt,0,VTeile-1)
        #declare V = Vanf+VSchritt*DeltaV;
                //Flächenpunkte und Normalen
#declare p1 = Flaeche(U,V);
#declare n1 = Normale(U,V);
                #declare p2 = Flaeche(U+ DeltaU,V);
#declare n2 = Normale(U+ DeltaU,V);
#declare p3 = Flaeche(U+ DeltaU,V+ DeltaV);
#declare n3 = Normale(U+ DeltaU,V+ DeltaV);
#declare p4 = Flaeche(U,V+DeltaV);
#declare n4 = Normale(U,V+DeltaV);
// Zwei geglättete Dreiecke
union { smooth_triangle {p1,n1,p2,n2,p3,n3}
smooth_triangle {p3,n3,p4,n4,p1,n1}
// Variante 1
pigment {FlaechenFarbe}
// Variante 2
//pigment{FarbenKreis(V+DeltaV/2)}
finish {FlaechenFinish}
              }
#end // Ende VSchritt
```

```
#end // Ende USchritt
} // Ende union


// Kreise in den Ebenen z = const. Einstellungen
// (Anzahl der Kreise weniger 1) soll Teiler von UTeile sein
#declare KreisSchritt = 16;
#declare KreisDicke = .015;

// DeltaV von oben, damit die Kurven auf der
// diskretisierten Fläche liegen
#declare KreisFarbe = pigment {color Silver*0.5}
#declare KreisFinish = finish {phong 0.9 ambient 0.5}

// Kreise definieren
#declare MuellerKreise =
union {
// USchritt von 0 bis KreisSchritt,
// damit der letzte Kreis auch gezeichnet wird
        #for (USchritt,0, KreisSchritt)
        #declare U = Uanf+USchritt*(Uend-Uanf)/KreisSchritt;
// Kurve geschlossen, daher kleiner-Zeichen
        #for (VSchritt,0,2*VTeile-1)
        #declare V = Vanf+VSchritt*DeltaV;
// Berechne zwei Punkte der Kurve und verbinde diese
#declare p1 = Flaeche(U,V);
#declare p2 = Flaeche(U,V + DeltaV);
union { cylinder {p1,p2,KreisDicke}
        sphere   {p1,KreisDicke}
pigment {KreisFarbe}
finish  {KreisFinish}}
        #end
        #end
}
// Ende der Deklarationen

// Szene
union{
        object{Koordinatensystem}
        object{MuellerFlaeche}
        object{MuellerFlaeche scale -1} // zweiter Teil
        object{MuellerKreise}
        object{MuellerKreise scale -1}  // zweiter Teil
        translate <0,0,-0.9>
}
```

Beispiel 4.9.7 Wir schließen an Beispiel 4.9.6 an und färben jeden Flächenpunkt entsprechend seinem v-Wert ein. Da die v-Linien ($u = \text{const}$) Kreise sind, bietet sich eine periodische Färbungsfunktion an. Eine der vielen Möglichkeiten ist die Funktion

$$k : [0, 2\pi] \to \mathbb{R}^3 : v \mapsto \frac{1}{3}\left(1 + 2\cos v, 1 - \sqrt{3}\sin v - \cos v, 1 + \sqrt{3}\sin v - \cos v\right).$$

Sie liefert im RGB-Farbraum den Umkreis des Dreiecks mit den Ecken $(1, 0, 0)$ = rot, $(0, 1, 0)$ = grün und $(0, 0, 1)$ = blau. Diese drei Grundfarben gehören zu den Parametern $v = 0$, $v = 2\pi/3$ und $v = 4\pi/3$. In Abbildung 4.22 wurde aus Gründen der Farbgebung die Funktion k mit dem Faktor $3/4$ skaliert. Diese Färbungsfunktion wurde durch ein Macro (`FarbenKreis`) implementiert.

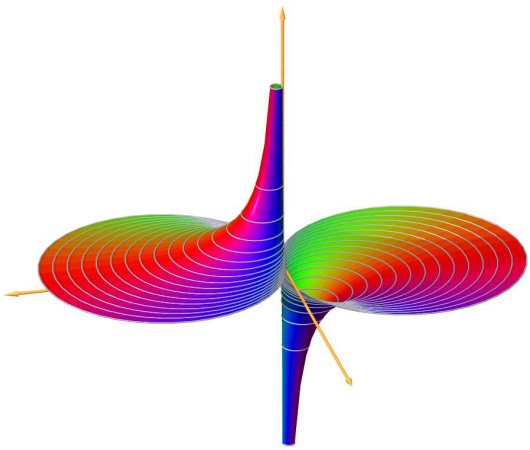


Abbildung 4.22: Müllersche Fläche nach v gefärbt

4.9.8 Ein anderes Verfahren zur Triangulierung einer parametrisierten Fläche $f(U)$ geht von einem achsenparallelen Rechteck von U aus, das die gegenüberliegenden Ecken

$$(u_1, v_1),\ (u_2, v_2)$$

mit $u_1 < u_2$ und $v_1 < v_2$ besitzt. Ein solches Rechteck bestimmt zwei der Fläche eineschriebene Dreiecke. Es gibt nun in Abhängigkeit von einer zuvor festgelegten Schranke $\varepsilon > 0$ folgende Möglichkeiten:

- Gilt $u_2 - u_1 \le \varepsilon$ und $v_2 - v_1 \le \varepsilon$, so werden die beiden Dreiecke dargestellt.
- Andernfalls unterscheiden wir zwei Unterfälle:
  - Gilt $u_2 - u_1 > \varepsilon$ so wird das Rechteck in zwei Teile geteilt mit

    $$u_3 := \frac{1}{2}(u_1 + u_2)$$

    als u-Wert der neuen Teilungspunkte. Dann werden die Überlegungen mit $(u_1, v_1), (u_3, v_2)$ und $(u_3, v_1), (u_2, v_2)$ wiederholt.
  - Tritt der obige Unterfall nicht ein, so muss $v_2 - v_1 > \varepsilon$ gelten. Hier wird das Rechteck in zwei Teile geteilt mit

    $$v_3 := \frac{1}{2}(v_1 + v_2)$$

als v-Wert der neuen Teilungspunkte. Dann werden die Überlegungen mit $(u_1, v_1), (u_2, v_3)$ und $(u_1, v_3), (u_2, v_2)$ wiederholt.

Die Umsetzung dieser *Unterteilung durch sukzessive Intervallhalbierung* kann in POV-Ray wie in Beispiel 4.3.5 erfolgen.

4.9.9 Das Schlüsselwort `mesh` bietet in POV-Ray eine andere Möglichkeit, Dreiecke zu einer Gesamtstruktur zusammenzufassen. Dabei sind aber gewisse Einschränkungen zu beachten, die bei Verwendung von `union` nicht auftreten. Wir gehen daher darauf nicht näher ein, obwohl `mesh` gegenüber `union` bei großen Datenmengen eine raschere Verarbeitung verspricht. Ferner sei auch noch auf den verwandten Befehl `mesh2` hingewiesen.

4.10 Import von Graphiken

4.10.1 Das Mathematikpaket *Maple*[13] unterstützt den Export von Graphiken nach POV-Ray. Die folgenden Bemerkungen stellen keine Einführung in Maple dar. Vielmehr wird nur gezeigt, wie ein Export von Maple nach POV-Ray erfolgen kann. Ferner soll hervorgehoben werden, worauf dabei in Maple und POV-Ray zu achten ist. Text in roter Farbe bezieht sich immer auf Maple.

Beispiel 4.10.2 Zur Illustration sei ein (sehr einfaches) Objekt gewählt: Ein grüner Würfel im Ursprung, ein blauer Quader entlang der positiven x-Achse, ein roter Würfel entlang der positiven y-Achse und ein grau-weißer Würfel entlang der positiven z-Achse. In Maple kann die Definition in Form einer Liste von Quadern so erfolgen:

```
> Quaderliste:=[cuboid([0,0,0],[2,2,2],colour=green),
> cuboid([2,0,0],[4,1,1],colour=blue),
> cuboid([0,2,0],[1,3,1],colour=red),
> cuboid([0,0,2],[1,1,3],colour=grey)
> ]:
```

Jedem dieser vier Quader entspricht in Maple eine Plotstruktur, die aus sechs Polygonen (einem Polygon je Seitenfläche) aufgebaut ist. Wir übernehmen alle Standardeinstellungen (Beleuchtung, Blickrichtung, ...) von Maple und stellen den Quader unter einer unverzerrten Orthogonalprojektion dar.

```
> display(Quaderliste,scaling=constrained,axes=normal,labels=[x,y,z]);
```

Es entsteht die Figur 4.23. Dieses Bild kann in der Windows-Version über das Kontext-Menü (rechte Maustaste) als POV-Ray-Datei exportiert werden. Die entsprechende Datei ist in Beispiel 4.10.5 wiedergegeben; sie liefert ohne Nachbearbeitung die Figur 4.24.

Alternativ zum Export über das Menü kann die Graphikausgabe vor dem `display`-Befehl in eine POV-Ray-Datei umgeleitet und anschließend wieder auf die Standardausgabe zurückgesetzt werden.

[13] Alle Angaben beziehen sich auf die Version 10.

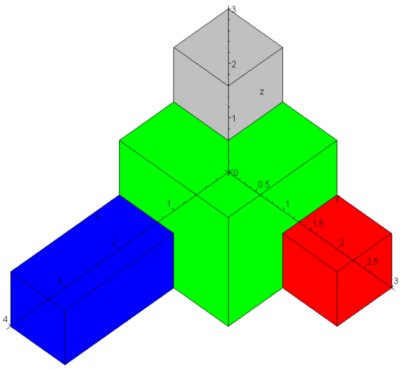


Abbildung 4.23: Darstellung in Maple

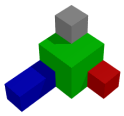


Abbildung 4.24: Darstellung in POV-Ray

```
> plotsetup(pov,plotoutput='test.pov', plotoptions='phi=23,theta=122');
    # Angabe des Ausgabetyps, des Dateinamens und der Blickrichtung
> display(Quaderliste);
> plotsetup(default); # Standardausgabe
```

Die Angabe der Blickrichtung muss bei dieser Vorgehensweise als `plot`-Option erfolgen, wobei die Werte für ϑ und φ die in Maple übliche Bedeutung haben. Die entsprechende `display`-Option (also hier: `orientation=[23,122]`) funktioniert beim Export nach POV-Ray nicht bzw. nicht richtig. Hingegen wird beim Export über das Menü die jeweils aktuelle Blickrichtung offenbar korrekt übernommen.

4.10.3 Beim Export sind noch in Maple folgende Einschränkungen zu beachten: Es werden nur die Blickrichtung und die Objektfarben berücksichtigt. Nicht unterstützt werden insbesondere Lichteinstellungen, Angaben wie `style=patchcontour`, Angaben zum Koordinatensystem (etwa `axes=boxed`) und Einstellungen über eine Zentralprojektion (`projection`). Es werden auch keine mit `spacecurve` dargestellten Raumkurven exportiert. Daher sollte `tubeplot` verwendet werden, um Raumkurven darzustellen.

4.10.4 Wir wechseln jetzt von Maple nach POV-Ray. Die von Maple erstellte POV-Ray-Datei liefert unter POV-Ray ein Bild, das nur in den seltensten Fällen so aussieht, wie das unter Maple erstellte Bild. Der Grund dafür ist, dass Maple beim Export gewisse einfache Standardeinstellungen vornimmt, die sich nicht verändern lassen.

- Es wird immer eine Zentralprojektion mit dem Augpunkt `<0,0,-20>` und dem Hauptpunkt `<0,0,0>` erstellt. Die Drehwinkel ϑ und φ für die Blickrichtung in Maple werden dabei durch eine Drehung der Kamera berücksichtigt:

  `[theta,phi]` in Maple ergibt `rotate<-phi,0,theta>` in POV-Ray.

  Achtung! Wenn das abzubildende Objekt weit vom Ursprung entfernt liegt, wird es in POV-Ray nicht oder nur teilweise zu sehen sein. (Unter Maple wird immer das gesamte Objekt dargestellt, wobei das Bild des Ursprungs nicht in der Mitte des rechteckigen Bildes zu liegen braucht.)

- Es wird eine Lichtquelle (weißes Licht) im Punkt `<50,50,-50>` definiert. Dieser Punkt wird so wie die Kamera gedreht.

- Die Hintergrundfarbe wird auf weiß gesetzt.

- Daran schließt die Definition einer Oberflächenstruktur `MaplePlotFinish` an.

Diese Standardeinstellungen dienen offensichtlich nur dazu, eine lauffähige POV-Ray-Datei zu erstellen. Sie werden in den meisten Fällen modifiziert werden müssen. An diesen Vorspann schließt dann die Beschreibung des Objekts an:

- Sämtliche Maple-Plotstrukturen werden in eine Vereinigung von Dreiecken übersetzt:

  `union{ triangle {...} triangle {...} ...}`

  Jedes einzelne Dreieck wird entsprechend den Farbeinstellungen in Maple gefärbt und erhält die zuvor definierte Oberflächenstruktur.

- Da Maple im Gegensatz zu POV-Ray ein Rechts-Koordinatensystem verwendet, werden die Maple-Koordinaten beim Export der Dreiecksmaschen umgerechnet:

  `[x,y,z]` in Maple geht über in `<y,-x,-z>` in POV-Ray.

Beispiel 4.10.5 Der Export der Graphik aus Beispiel 4.10.2 über das Menü ergibt (bis auf den Zeilenumbruch) die folgende POV-Ray-Datei.

```
// Maple POV-Ray Export

camera { location <0, 0, -20> look_at <0, 0, 0> rotate <-45, 0, 45> }

light_source { <50, 50, -50> color red 1.0 green 1.0 blue 1.0  rotate <-45, 0, 45> }

background { color red 1.0 green 1.0 blue 1.0 }

#declare MaplePlotFinish = finish { specular 0.4 ambient 0.5 diffuse 0.2 }

union
{
    triangle { <1.0000, -1.0000, -2.0000>, <1.0000, 0.0000, -3.0000>, <1.0000, -1.0000, -3.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <1.0000, -1.0000, -2.0000>, <1.0000, 0.0000, -2.0000>, <1.0000, 0.0000, -3.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, 0.0000, -3.0000>, <1.0000, -1.0000, -3.0000>, <0.0000, -1.0000, -3.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, 0.0000, -3.0000>, <1.0000, 0.0000, -3.0000>, <1.0000, -1.0000, -3.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, -1.0000, -2.0000>, <1.0000, -1.0000, -3.0000>, <1.0000, -1.0000, -2.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, -1.0000, -2.0000>, <0.0000, -1.0000, -3.0000>, <1.0000, -1.0000, -3.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, 0.0000, -2.0000>, <0.0000, -1.0000, -3.0000>, <0.0000, -1.0000, -2.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, 0.0000, -2.0000>, <0.0000, 0.0000, -3.0000>, <0.0000, -1.0000, -3.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, 0.0000, -2.0000>, <1.0000, 0.0000, -3.0000>, <1.0000, 0.0000, -2.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, 0.0000, -2.0000>, <0.0000, 0.0000, -3.0000>, <1.0000, 0.0000, -3.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, 0.0000, -2.0000>, <1.0000, -1.0000, -2.0000>, <0.0000, -1.0000, -2.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <0.0000, 0.0000, -2.0000>, <1.0000, 0.0000, -2.0000>, <1.0000, -1.0000, -2.0000>
      texture { pigment { color red 0.753 green 0.753 blue 0.753 } finish { MaplePlotFinish } } }
    triangle { <3.0000, -1.0000, 0.0000>, <3.0000, 0.0000, -1.0000>, <3.0000, -1.0000, -1.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <3.0000, -1.0000, 0.0000>, <3.0000, 0.0000, 0.0000>, <3.0000, 0.0000, -1.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, 0.0000, -1.0000>, <3.0000, -1.0000, -1.0000>, <2.0000, -1.0000, -1.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, 0.0000, -1.0000>, <3.0000, 0.0000, -1.0000>, <3.0000, -1.0000, -1.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, -1.0000, 0.0000>, <3.0000, -1.0000, -1.0000>, <3.0000, -1.0000, 0.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, -1.0000, 0.0000>, <2.0000, -1.0000, -1.0000>, <3.0000, -1.0000, -1.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, 0.0000, 0.0000>, <2.0000, -1.0000, -1.0000>, <2.0000, -1.0000, 0.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, 0.0000, 0.0000>, <2.0000, 0.0000, -1.0000>, <2.0000, -1.0000, -1.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, 0.0000, 0.0000>, <3.0000, 0.0000, -1.0000>, <3.0000, 0.0000, 0.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, 0.0000, 0.0000>, <2.0000, 0.0000, -1.0000>, <3.0000, 0.0000, -1.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, 0.0000, 0.0000>, <3.0000, -1.0000, 0.0000>, <2.0000, -1.0000, 0.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <2.0000, 0.0000, 0.0000>, <3.0000, 0.0000, 0.0000>, <3.0000, -1.0000, 0.0000>
      texture { pigment { color red 1.000 green 0.000 blue 0.000 } finish { MaplePlotFinish } } }
    triangle { <1.0000, -4.0000, 0.0000>, <1.0000, -2.0000, -1.0000>, <1.0000, -4.0000, -1.0000>
```

```
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <1.0000, -4.0000, 0.0000>, <1.0000, -2.0000, 0.0000>, <1.0000, -2.0000, -1.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, -1.0000>, <1.0000, -4.0000, -1.0000>, <0.0000, -4.0000, -1.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, -1.0000>, <1.0000, -2.0000, -1.0000>, <1.0000, -4.0000, -1.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -4.0000, 0.0000>, <1.0000, -4.0000, -1.0000>, <1.0000, -4.0000, 0.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -4.0000, 0.0000>, <0.0000, -4.0000, -1.0000>, <1.0000, -4.0000, -1.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, 0.0000>, <0.0000, -4.0000, -1.0000>, <0.0000, -4.0000, 0.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, 0.0000>, <0.0000, -2.0000, -1.0000>, <0.0000, -4.0000, -1.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, 0.0000>, <1.0000, -2.0000, -1.0000>, <1.0000, -2.0000, 0.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, 0.0000>, <0.0000, -2.0000, -1.0000>, <1.0000, -2.0000, -1.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, 0.0000>, <1.0000, -4.0000, 0.0000>, <0.0000, -4.0000, 0.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, 0.0000>, <1.0000, -2.0000, 0.0000>, <1.0000, -4.0000, 0.0000>
    texture { pigment { color red 0.000 green 0.000 blue 1.000 } finish { MaplePlotFinish } } }
  triangle { <2.0000, -2.0000, 0.0000>, <2.0000, 0.0000, -2.0000>, <2.0000, -2.0000, -2.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <2.0000, -2.0000, 0.0000>, <2.0000, 0.0000, 0.0000>, <2.0000, 0.0000, -2.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, 0.0000, -2.0000>, <2.0000, -2.0000, -2.0000>, <0.0000, -2.0000, -2.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, 0.0000, -2.0000>, <2.0000, 0.0000, -2.0000>, <2.0000, -2.0000, -2.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, 0.0000>, <2.0000, -2.0000, -2.0000>, <2.0000, -2.0000, 0.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, -2.0000, 0.0000>, <0.0000, -2.0000, -2.0000>, <2.0000, -2.0000, -2.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, 0.0000, 0.0000>, <0.0000, -2.0000, -2.0000>, <0.0000, -2.0000, 0.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, 0.0000, 0.0000>, <0.0000, 0.0000, -2.0000>, <0.0000, -2.0000, -2.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, 0.0000, 0.0000>, <2.0000, 0.0000, -2.0000>, <2.0000, 0.0000, 0.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, 0.0000, 0.0000>, <0.0000, 0.0000, -2.0000>, <2.0000, 0.0000, -2.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, 0.0000, 0.0000>, <2.0000, -2.0000, 0.0000>, <0.0000, -2.0000, 0.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
  triangle { <0.0000, 0.0000, 0.0000>, <2.0000, 0.0000, 0.0000>, <2.0000, -2.0000, 0.0000>
    texture { pigment { color red 0.000 green 1.000 blue 0.000 } finish { MaplePlotFinish } } }
}
```

Diese Datei wurde nur eingefügt, um zu illustrieren, dass beim Export nach POV-Ray selbst für sehr einfache Objekte recht umfangreiche POV-Ray-Dateien entstehen.

Bemerkung 4.10.6 Als Alternative bietet es sich an, auf den direkten Import aus Maple gänzlich zu verzichten und die benötigten Daten gleich in einer solchen Form in eine Datei zu schreiben, dass diese Daten dann in POV-Ray als Include-Datei direkt weiterverarbeitet werden können. In gleicher Weise kann auch ein Import von Daten aus *MATLAB* und anderen Programmen erfolgen.

Kapitel 5

Animationen

5.1 Bilderfolgen in POV-Ray

5.1.1 Gleich vorweg: Mit POV-Ray alleine können keine *Animationen* erstellt werden. Es ist vielmehr nur möglich, *Folgen von Standbildern* zu generieren. Dabei wird jedes einzelne Standbild[1] unabhängig von den anderen Bildern berechnet. Das bedeutet zwar unter Umständen einen großen Rechenaufwand, hat aber den Vorteil einer hohen Flexibilität.

5.1.2 Damit POV-Ray beispielsweise eine Folge von fünfzig Bildern berechnet, sind in die (aktuell verwendete) Initialisierungs-Datei folgende beiden Zeilen einzutragen:

```
Initial_Frame=1
Final_Frame=50
```

Beim Ausführen einer POV-Ray-Datei, etwa mit dem Namen `meine_anim.pov`, werden dann der Reihe nach die Bilder

```
meine_anim01.png, meine_anim02.png, ..., meine_anim50.png
```

angelegt. Dabei richtet sich die Anzahl der führenden Nullen in den Dateinamen nach dem Wert von `Final_Frame`. Wird dieser Wert zum Beispiel auf 150 verändert, so werden Dateien mit den Namen

```
meine_anim001.png, meine_anim002.png, ..., meine_anim150.png
```

generiert. Diese Einstellungen lassen sich auch über die Kommandozeile steuern.

Ohne weiteres Zutun liefert die oben beschriebene Vorgangsweise lauter identische Bilder. Wie der sich der Inhalt der einzelnen Bilder beeinflussen lässt, wird in 5.1.4 beschrieben.

5.1.3 Soll nur eine *Teilfolge* der oben beschriebenen Bilderfolge berechnet werden, so sind in der Initialisierungs-Datei zusätzlich zu den Angaben von `Initial_Frame` und `Final_Frame` etwa noch die Zeilen

```
Subset_Start_Frame=21
Subset_End_Frame=40
```

[1] Wir sprechen im Folgenden auch kurz von einem Bild.

zu ergänzen. Es wäre falsch, zu diesem Zweck in der Initialisierungs-Datei die Eintragung `Initial_Frame=21, Final_Frame=40` zu machen. Damit würde die gesamte ursprüngliche Bildersequenz auf zwanzig neue Bilder

```
meine_anim21.png, meine_anim22.png, ..., meine_anim40.png
```

umgerechnet.

5.1.4 Zur Programmierung des Inhalts von Bilderfolgen dient in POV-Ray das Schlüsselwort `clock`, welches einem Zeitparameter entspricht. Der Wert dieser reservierten Variablen ist eine reelle Zahl, die nur über die Initialisierungs-Datei oder die Kommandozeile beeinflusst werden kann. In einer POV-Ray-Datei selbst ist keine Veränderung möglich. Wir besprechen hier nur die Steuerung über die Initialisierungs-Datei. Wird dort etwa

```
Initial_Clock=2
Final_Clock=4
```

eingetragen, so tritt folgendes ein:

- Die Variable `clock` hat während der gesamten Berechnung des ersten Bildes einer Bilderfolge den Wert 2.
- Falls die Bilderfolge aus mehr als einem Bild besteht, nimmt `clock` beim letzten Bild den Wert 4 an. Für die Bilder dazwischen wird `clock` durch lineare Interpolation zwischen dem Anfangs- und Endwert berechnet.

Werden in der Initialisierungs-Datei keine Angaben gemacht, so bleiben die Standardwerte

```
Initial_Clock=0
Final_Clock=1
```

in Kraft. Wir nehmen im Folgenden stets an, dass `clock` von 0 bis 1 läuft.

Mit Hilfe von `clock` kann nun in der POV-Ray-Datei der Bildinhalt gesteuert werden. So bewirkt etwa eine Anweisung

```
object{ ... rotate z*180*clock }
```

die Drehung eines Objekts um die dritte Koordinatenachse in Abhängigkeit von der Zeit.

Der Vorteil von `clock` liegt darin, dass sich die POV-Ray-Datei unabhängig von der Anzahl der zu erstellenden Bilder schreiben lässt. Die Schrittweite, mit der `clock` dann bei der Berechnung einer Bilderfolge verändert wird, ergibt sich alleine aus den Einstellungen für `Initial_Frame` und `Final_Frame`.

5.1.5 Soll eine Bilderfolge später eine *zyklisch ablaufende Animation* liefern, so muss die POV-Ray-Datei derart gestaltet werden, dass für `clock=0` und `clock=1` dasselbe Bild entsteht. So bleibt die Unabhängigkeit von der Anzahl der Bilder gewahrt. Das letzte Bild wird allerdings gar nicht benötigt. Durch eine Abfrage der Form

```
# if (clock < 1) ... #end
```

kann erreicht werden, dass das (überflüssige) letzte Bild rascher (mit leerem Bildinhalt) berechnet wird.

5.2 Weiterverarbeitung

5.2.1 Mehrere elektronische Einzelbilder lassen sich mit einschlägiger Software in Video-Animationen überführen. Wir können darauf hier nicht eingehen. Es sei aber betont, dass derartige Programme in der Regel folgende Möglichkeiten bieten:

- Berechnung von Bildübergängen, also das Einfügen neuer (interpolierter) Bilder zwischen die vorhandenen Bilder.
- Individuelle oder globale Steuerung der Anzeigedauer von Einzelbildern.

Durch nachträglich erstellte Übergänge kann die Anzahl der zu berechnenden Bilder unter Umständen verkleinert werden. Gemeinsam mit der Steuerung der Anzeigedauer ist es auch möglich, im Nachhinein die Geschwindigkeit der Animation zu beeinflussen.

5.2.2 Wir werden im Folgenden keine *Weiterverarbeitung* berücksichtigen. Bei der Berechnung einer Bilderfolge mit POV-Ray gehen wir daher von folgenden Annahmen aus:

- Es werden nachträglich keine Bilder mehr eingefügt.
- Jedes Bild wird beim Betrachten für eine konstante Zeitspanne gesehen, bevor das nächste Bild erscheint.

Im Sinne dieser Festsetzungen entspricht jeder Wert der Variablen `clock` bis auf einen positiven Proportionalitätsfaktor tatsächlich dem Zeitpunkt, zu dem ein Bild im Laufe einer Präsentation erscheint.

Nur bei einer zyklisch ablaufenden Präsentation tritt eine Abweichung ein. Hier lassen wir das letzte von POV-Ray berechnete Bild einfach weg. Vgl. dazu die Anmerkungen in 5.1.5.

5.3 Wiedergabe

5.3.1 Wir verwenden zur Wiedergabe unserer mit POV-Ray erstellten Bilderfolgen eine HTML-Datei mit einem eingebundenem Skript[2]. Diese Datei steht unter dem Namen

`muster_anim.html`

in TUWEL bereit. Wir beschreiben im Folgenden die nötigen Modifikationen.

5.3.2 Im Konfigurationsbereich des Skripts brauchen nur die folgenden (vorhandenen) Eintragungen angepasst werden:

```
image_name = "kubik_anim";
image_type = "png";

first_image = 1;
last_image = 20;

animation_height = 480;
animation_width = 640;
```

[2] jsImagePlayer 1.0, Copyright © 1996–98 by Martin Holečko, ČVUT Praha. Das in JavaScript geschriebene Original wurde verändert. Quelle: `http://sgi.felk.cvut.cz/~xholecko/player` (abgerufen 11. Juli 2001).

```
play_mode = 1;
```

Dabei muss jede Zeile mit einem *Semikolon* enden. In der Vorlage sind noch kurze Kommentare hinzugefügt, die (so wie in POV-Ray) hinter der Zeichenfolge // versteckt sind. Wir beschreiben hier die Bedeutung der einzelnen Einstellungen:

- `image_name`: Der gemeinsame erste Teil des Namens aller Bilddateien. (Bitte nur Kleinbuchstaben verwenden.)
- `image_type`: Der gemeinsame Dateityp der Namens aller Bilddateien. (Bitte `png` verwenden.)
- `first_image`: Laufende Nummer des ersten Bildes. Führende Nullen spielen bei der Eingabe keine Rolle. Sie werden in den Dateinamen automatisch passend ergänzt.
- `last_image`: Laufende Nummer des letzten Bildes.
- `animation_height`: Höhe der animierten Graphik in Pixel. (Sie sollte mit der Originalhöhe übereinstimmen.)
- `animation_width`: Breite der animierten Graphik in Pixel. (Sie sollte mit der Originalbreite übereinstimmen.)
- `play_mode`: Folgende Werte sind zulässig:
  0: Die Animation läuft einmal ab und endet mit dem letzten Bild.
  1: Die Animation läuft beliebig lange zyklisch ab.
  2: Die Animation läuft beliebig lange vor und zurück.

Ganz am Ende der HTML-Datei steht dieser Aufruf:

```
<script type="text/javascript">
<!--
animation();
//-->
</script>
```

Damit wird die Animation samt Steuerungsoberfläche an genau dieser Stelle in das HTML-Dokument eingefügt. Der nachfolgende Hinweis

```
<noscript>
Java Script not supported
</noscript>
```

dient nur zur Information. Er erscheint immer dann, wenn JavaScript im verwendeten Web-Browser nicht aktiviert wurde.

5.3.3 Nach dem Öffnen der oben beschriebenen HTML-Datei mit einem Web-Browser erscheint unterhalb des ersten Bildes der Animation eine einfache Benutzeroberfläche: Hier lässt

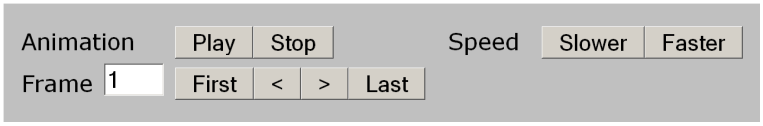


Abbildung 5.1: Benutzeroberfläche

sich die Animation starten und stoppen. Ferner kann die die Geschwindigkeit reguliert werden. In der zweiten Zeile erscheint die laufende Nummer des Bildes. Durch Eintragen einer Zahl springt die Darstellung zum betreffenden Bild.[3] Darüber hinaus kann zum ersten [First], zum vorhergehenden [<], zum nachfolgenden [>] und zum letzten Bild [Last] gesprungen werden. Mit [Play] startet die Animation wieder beim aktuellen Bild.

Falls die Animation auf einmaligen Ablauf eingestellt ist und das letzte Bild erreicht wurde, muss zuerst [First] und dann [Play] gedrückt werden, um die Animation neu zu starten.

5.4 Gestaltung von Animationen

5.4.1 Wenn sich in einer Animation alles wie wild dreht und bewegt, dann ist das wenig ansprechend. Eine gute Animation sollte mit den Möglichkeiten bewegter Bilder sparsam umgehen. Wir stellen im Folgenden einige Ideen zusammen, die aber nur als Anregung dienen sollen.

5.4.2 Soll die *stetige Bewegung* eines Objekts dargestellt werden, so ist zu beachten, dass jeder Bewegungsvorgang relativ zu einem „ruhenden Bezugssystem“ zu sehen ist. Bei einer solchen Animation sollte daher die Kamera nicht auch noch bewegt werden. Der ruhende Raum kann durch einen unbewegten Hintergrund (eine Kulisse) suggeriert werden. Das bewegte Objekt befindet sich im Idealfall im Vordergrund der Szene. Bei der Gestaltung der Bewegung sollte ferner berücksichtigt werden, ob konstante oder variable Geschwindigkeit vorliegen soll.

Bei einer reinen *stetigen Drehung* eines Objekts um eine Achse kann auf einen Hintergrund verzichtet werden, falls es beim Betrachten der Animation klar ist, wo sich die Drehachse ungefähr befindet.

Beispiel 5.4.3 Wir erstellen eine Animation einer *kubischen Raumkurve*. Diese Kurve liegt auf einem parabolischen Zylinder. Ferner wird ein durch zwei Punkte P und Q der Kurve bestimm-

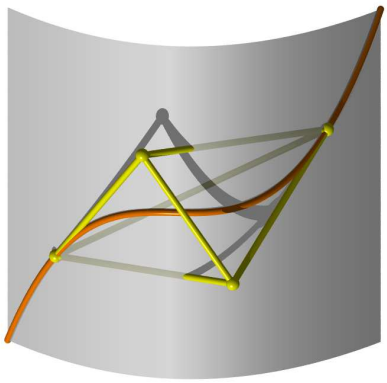


Abbildung 5.2: Ein Bild aus der Animation einer kubischen Raumkurve

tes Schmiegtetraeder dargestellt. Zwei der Ecken des Tetraeders sind die Punkte P und Q, die beiden anderen Ecken sind die Schnittpunkte der Tangente in P (bzw. Q) mit der Schmiegebene in Q (bzw. P).

[3] Ein in der Dokumentation zitierter *known bug* wurde von Peter Regner behoben. Es erscheint nunmehr auch nach Abschluss der Eingabe mit der Enter-Taste das betreffende Bild.

Das gesamte Objekt rotiert stetig um die zur z-Achse parallele Gerade durch den Punkt $(0, 1, 0)$ (zyklische Animation). Die gesamte Animation (HTML-Datei und Bilder) kann in TUWEL unter dem Namen

kubik_anim.zip

gefunden werden. Alle darin enthaltenen Dateien sind in dasselbe Verzeichnis zu entpacken. Die Animation wurde folgendermaßen erstellt:

```
// Visualisierung Hans Havlicek

// In der Ini-Datei einfügen oder über die Befehlszeile steuern
// Initial_Frame = 1
// Final_Frame = 101

// Drehung! Frame 101 = Frame001 wird später nicht verwendet

#include "colors.inc"
#include "transforms.inc"

// Trafo auf Rechtssystem
#declare Trafo = transform {matrix <1,0,0, 0,0,1, 0,1,0, 0,0,0>}


camera { orthographic location vtransform(<5, 15, 3>,Trafo)
 look_at vtransform(<0, 0 , 0>,Trafo) transform Trafo}

#if (clock < 1) // letztes Bild leer

light_source { <50, 50, -50> color White parallel }

background { color red 1.0 green 1.0 blue 1.0 }

#declare MeinFinish = finish { phong 0.9 ambient 0.3 }
#declare MeinPigment = pigment{ color Orange }


// Grundeinstellungen
#declare Teile=200; // Anzahl der Teilstrecken
#declare Tanf=-5.5; // Anfangsparameter
#declare Tend= 5.5; // Endparameter
#declare S=0;       // Anfangswert des Zählers S
#declare Dicke=0.1; // halbe Dicke der Kurve (Rohrfläche)

//Gestaltparameter für die Kurve
#declare A=0.2;
#declare B=0.03;
```

```
#declare Eps=1.5;

// Macro für die Parameterdarstellung der Kurve
#macro Kurvenpunkt(T)
        <T, A*T*T, B*T*T*T>
#end

// Punkte auf Tangenten aus Nebenrechnung übernommen
#declare TangP = <-1.333333333, -1.066666668, 1.920000000>;
#declare TangQ = <1.333333333, -1.066666668, -1.920000000>;


// Objekt erstellen
union{
#while (S<Teile) // kleiner
        // Parameter zum Anfangs- und Endpunkt der aktuellen Strecke
        #declare T0 = Tanf+S/Teile*(Tend-Tanf);
        #declare T1 = Tanf+(S+1)/Teile*(Tend-Tanf);
        union{
                // Anfangspunkt und Strecke
                sphere{Kurvenpunkt(T0) Dicke}
                cylinder{Kurvenpunkt(T0) Kurvenpunkt(T1) Dicke open}
                pigment{MeinPigment} finish{MeinFinish}
                }
        #declare S = S+1; // Zähler erhöhen
#end // Ende der while-Schleife

// Kugel zum Abschluss ergänzen
sphere{Kurvenpunkt(Tend) 1*Dicke pigment{MeinPigment}
                                  finish{MeinFinish}}


// Schmiegtetraeder aus Kugeln und Zylindern
union{
sphere{Kurvenpunkt(Tanf+Eps) 2*Dicke }
sphere{Kurvenpunkt(Tend-Eps) 2*Dicke }
sphere{TangP 2*Dicke }
sphere{TangQ 2*Dicke }

cylinder{Kurvenpunkt(Tanf+Eps) TangP Dicke open }
cylinder{Kurvenpunkt(Tend-Eps) TangQ Dicke open }
cylinder{Kurvenpunkt(Tend-Eps) TangP Dicke open }
cylinder{Kurvenpunkt(Tanf+Eps) TangQ Dicke open }
cylinder{TangP TangQ Dicke open }
cylinder{Kurvenpunkt(Tanf+Eps) Kurvenpunkt(Tend-Eps) Dicke open }
```

```
pigment{color Yellow} finish{MeinFinish}
}

// parabolischer Zylinder
quadric {<A,0,0>,<0,0,0>,<0,-1,0>,0
 pigment{color White transmit 0.2} finish{MeinFinish}
 clipped_by{box{< Kurvenpunkt(Tanf).x,Kurvenpunkt(0).y,
                                    Kurvenpunkt(Tanf).z>,
                Kurvenpunkt(Tend)}} }

// Schieben, da um die z-Achse gedreht wird. Anpassen an Bildformat

translate -y

scale 1.2
// Steuerung der Animation mit clock
rotate  z*360*clock

} // Ende union

#end // Ende if clock
```

5.4.4 Bei einer *Kamerafahrt* sollte das Objekt nicht stark verändert werden. Die Verwendung einer Zentralprojektion ist hier dringend anzuraten, da damit das Projektionszentrum tatsächlich an einen beliebigen Punkt des Raumes gebracht werden kann. Beim Betrachten der Animation spielt es keine Rolle, ob bei der Berechnung die Kamera oder das gesamte Objekt bewegt wurde. Diese beiden Bewegungsvorgänge sind aber invers zueinander.

5.4.5 Beim *Zoomen* empfiehlt sich die Verwendung einer Zentralprojektion. Beachte, dass der Augpunkt hier gleich bleibt. Es wird nur der Hauptpunkt der Perspektive auf der Blickachse verschoben.

5.4.6 Das Wechselspiel von Licht und Schatten kann durch eine *stetig bewegte Lichtquelle* unterstrichen werden. Ebenso können Änderungen an Oberflächen zeitabhängig gestaltet werden. Mehr dazu findet sich unter dem Schlüsselwort `phase` in der Online-Hilfe zu POV-Ray.

Beispiel 5.4.7 Die folgende Animation (HTML-Datei und Bilder) kann in TUWEL unter dem Namen

`baum_anim.zip`

gefunden werden. Sie zeigt eine Animation des Schattens eines Baumes zum Äquinoktium (Zeitpunkt der Tag-und-Nacht-Gleiche) auf 50° nördlicher Breite.

Der Lauf und die variable Helligkeit der Sonne wurden so implementiert:

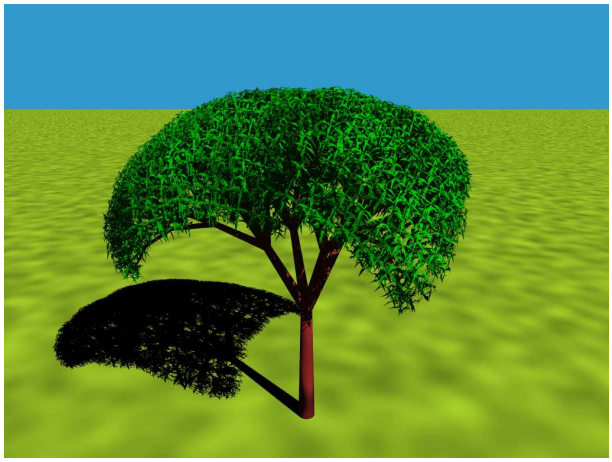


Abbildung 5.3: Ein Bild aus der Animation des Schattens eines Baumes

```
light_source {<-440, 0, 0> color rgb 1.8*pow(sin(clock*pi),1/30)
              parallel
              rotate y*(-10+clock*200)
              rotate -x*50
             }
```

Der Winkel der oben beschriebenen Drehung um die y-Achse sei im Folgenden mit φ bezeichnet. Wir schreiben außerdem $t \in [0, 1]$ für den Zeitparameter `clock`. Die Animation beginnt für $t = 0$ mit $\varphi = -10°$ noch vor Sonnenaufgang. Für $\varphi = 0°$ geht die Sonne auf und für $\varphi = 180°$ wieder unter. Das letzte Bild entspricht einem Winkel $\varphi = 190°$ zum Zeitpunkt $t = 1$. Der Winkel für die Drehung um die x-Achse ergibt sich aus der geographischen Breite. Die Helligkeit des Sonnenlichts wird durch

$$1.8 \cdot \sqrt[30]{\sin(t\pi)}$$

geregelt. Die Helligkeit beginnt also bei 0, steigt dann sehr rasch an und fällt schließlich ebenso rasch wieder auf 0. Zusätzlich gibt es ein Morgen- und Abendrot, das durch

```
#declare Morgenrot=Red*0;
#if (clock<1/4) #declare Morgenrot=Red*0.4*sin(4*clock*pi);   #end
#if (clock>3/4) #declare Morgenrot=Red*0.4*sin(4*(1-clock)*pi); #end

light_source {<-440, 0, 0> color 2*Morgenrot parallel
rotate y*(-10+clock*200) rotate -x*50  shadowless }
```

gesteuert wird. Die Helligkeit dieses zusätzlichen Rotanteils am Sonnenlicht wird durch

$$0.4 \cdot \sin(4t\pi)$$

bestimmt, wobei t im Intervall $[0, 1/4]$ läuft. Das Abendrot wird analog für $t \in [3/4, 1]$ definiert. Schließlich wird auch die Farbe des Himmels zeitabhängig gesteuert. Dazu dient die folgende Anweisung, welche die Farbe mit dem Faktor

$$2\sqrt{t}$$

für $t \in [0, 1/4]$ (am Morgen) und analog für $t \in [3/4, 1]$ (am Abend) verändert.

```
#declare Himmel=SkyBlue;

#if (clock < 1/4) #declare Himmel=SkyBlue*2*sqrt(clock);   #end
#if (clock > 3/4) #declare Himmel=SkyBlue*2*sqrt(1-clock); #end

background { color Himmel+0.5*Morgenrot}
```

5.4.8 Als letztes seien *veränderliche Objekte* genannt. Dabei können intuitiv zwei Arten von Prozessen unterschieden werden.

Bei einer *Deformation* ändert etwa eine parametrisierte Fläche ihre Gestalt, es kommen aber weder Punkte dazu noch fallen Punkte weg. Das kann etwa dadurch realisiert werden, dass bei festem Parameterbereich nur die Parametrisierung zeitabhängig verändert wird.

Ein *Wachstumsprozess* liegt etwa vor, falls illustriert wird, wie eine Lösungskurve einer Differentialgleichung von einer gewissen Anfangsbedingung aus im Laufe der Zeit durch Integration (als Funktion der oberen Grenze) berechnet werden kann. Hier liegt letztlich eine feste Parameterdarstellung einer Kurve vor, bloß werden für die einzelnen Bilder nur Teile des Parameterintervalls verwendet.

Literaturverzeichnis

[1] Arens, T., F. Hettlich, C. Karpfinger, U. Kockelkorn, K. Lichtenegger und H. Stachel: *Mathematik*. Spektrum Akademischer Verlag, Heidelberg, 2. Auflage, 2011.

[2] Arens, T., F. Hettlich, C. Karpfinger, U. Kockelkorn, K. Lichtenegger und H. Stachel: *Mathematik*. Springer Spektrum, Heidelberg, 3. Auflage, 2015.

[3] Beutelspacher, A. und U. Rosenbaum: *Projektive Geometrie. Von den Grundlagen bis zu den Anwendungen*. Vieweg, Braunschweig, 2. Auflage, 2004.

[4] Brauner, H.: *Differentialgeometrie*. Vieweg, Braunschweig, 1981.

[5] Brauner, H.: *Lehrbuch der konstruktiven Geometrie*. Springer, Wien, 1986.

[6] Glaeser, G. und K. Polthier: *Bilder der Mathematik*. Spektrum Akademischer Verlag, Heidelberg, 2009.

[7] Glaeser, G. und K. Polthier: *Bilder der Mathematik*. Springer Spektrum, Heidelberg, 2. Auflage, 2014.

[8] Havlicek, H.: *Lineare Algebra für Technische Mathematiker*. Heldermann, Lemgo, 2. Auflage, 2008.

[9] Hohenberg, F.: *Konstruktive Geometrie in der Technik*. Springer, Wien, 2. Auflage, 1961.

[10] Kühnel, W.: *Differentialgeometrie. Kurven – Flächen – Mannigfaltigkeiten*. Vieweg, Wiesbaden, 5. Auflage, 2010.

[11] Wunderlich, W.: *Darstellende Geometrie I*. Bibliographisches Institut, Mannheim, 1966.

[12] Wunderlich, W.: *Darstellende Geometrie II*. Bibliographisches Institut, Mannheim, 1967.

Sachverzeichnis

Abbildungsverzeichnis