Wolfgang K. Seiler

Geometrische Vorbereitungen

Unterlagen zur Sommerschule Computational Visualization
Universität Mannheim, 31. Juli – 9. August 2002

KAPITEL I: GEOMETRISCHE ABBILDUNGEN 1

§1: Abbildungen des Raums in die Ebene 1
a) Axonometrien 2
b) Perspektivische Projektionen und Kameramodell 5

§2: Bewegungen im Raum 10
a) Homogene Koordinaten 10
b) Verschiebungen, Drehungen, Streckungen 13
c) Matrixstacks und z-Puffer 15

KAPITEL II: GEOMETRISCHE OBJEKTE 17

§1: Polyeder und Flächen 17
a) Polyeder 17
b) Parametrische Flächen 20
c) Implizite Flächen 22

§2: CSG-Darstellung eines dreidimensionalen Objekts 24
a) Primitive Objekte 24
1) Solid modeling 24
2) Kinematische Erzeugung 25
3) Algebraische Halbraumdarstellung 26
4) Normteile 26
b) Geometrische Manipulation primitiver Objekte 28
c) Regularisierte Boolesche Operationen 28
d) Der CSG-Baum eines Objekts 30
e) Manipulation von CSG-Bäumen 32

§3: Krümmungen und Verzerrungen 37
a) Krümmung und geometrische Stetigkeit 37
b) Das Theorema egregium 41
c) Bewußte Verzerrungen 43

Kapitel 1
Geometrische Abbildungen

§ 1: Abbildungen des Raums in die Ebene

In der Visualisierung wollen wir Objekte aus einem höherdimensionalen Raum verstehen, indem wir sie auf einem zweidimensionalen Bildschirm dargestellen. Im einfachsten Fall geht es um dreidimensionale Objekte aus unserer Erfahrungswelt, oft müssen aber auch Phänomene visualisiert werden, deren Phasenraum eine erheblich größere Dimension hat. (Unter dem Phasenraum versteht man einen Raum, dessen Koordinaten den für das Phänomen interessanten Größen entsprechen; für die Bewegung eines punktförmigen Teilchens etwa wäre der Phasenraum der $\mathbb{R}^6$, dessen erste drei Koordinaten die Position des Teilchens angeben, während die restlichen den drei Komponenten des Geschwindigkeitsvektors entsprechen.)

Wir werden im Verlauf dieser Sommerschule viele Techniken kennenlernen, um diese Aufgabe mit möglichst geringem Informationsverlust zu lösen. Da der Raum unserer Anschauung dreidimensional ist, beruhen viele davon darauf, zunächst ein dreidimensionales Objekt zu erzeugen und dieses dann auf dem Bildschirm darzustellen.

Aus diesem Grund wollen wir uns heute, am ersten Tag der Sommerschule, zunächst damit beschäftigen, wie man die dreidimensionale Welt auf den Bildschirm bringt.

Tatsächlich wollen wir meist nicht die gesamte dreidimensionale Welt abbilden, sondern nur einen gewissen Ausschnitt – genau wie ja auch der Bildschirm (oder ein darauf befindliches Fenster) als Ausschnitt

eines $\mathbb{R}^2$ betrachtet werden kann. Der Übersichtlichkeit halber entkoppeln wir das Problem der Ausschnittbildung von der eigentlichen Abbildungsvorschrift und betrachten Abbildungen des gesamten $\mathbb{R}^3$ (oder zumindest einer möglichst großen Teilmenge davon) nach $\mathbb{R}^2$.

(Mit Abbildungsvorschriften wie

$$\varphi\colon \begin{cases} \mathbb{R}^3 \to \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \times \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \\ (x,y,z) \mapsto \left(\arctan\left(-\frac{\sqrt{3}}{2}x + \frac{\sqrt{3}}{2}y\right), \arctan\left(z - \frac{x+y}{2}\right)\right) \end{cases}$$

könnte auch der gesamte $\mathbb{R}^3$ auf ein Rechteck abgebildet werden; wegen der zum Rand hin immer größer werdenden Verzerrungen, die eine solche Abbildung mit sich bringt, werden solche Abbildungen aber nur selten und nur für sehr spezielle Probleme verwendet.)

Für die Konstruktion von Abbildungen von $\mathbb{R}^3$ nach $\mathbb{R}^2$ sind vor allem zwei Strategien gebräuchlich:

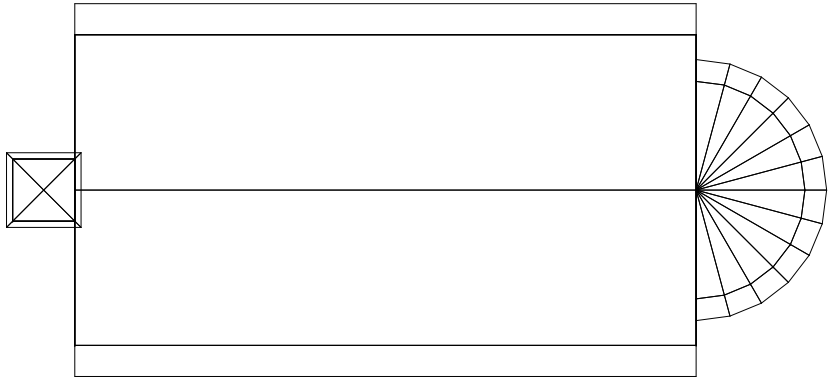
a) Axonometrien

Hier geht es um man lineare Abbildungen von $\mathbb{R}^3$ nach $\mathbb{R}^2$. Eine solche Abbildung ist gegeben durch die Bilder $\vec{a}_1, \vec{a}_2, \vec{a}_3$ der drei Koordinateneinheitsvektoren des $\mathbb{R}^3$; sie hat die Form

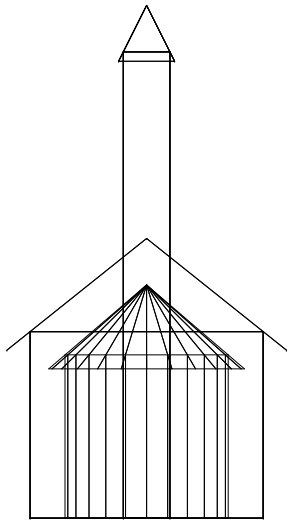
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto x\vec{a}_1 + y\vec{a}_2 + z\vec{a}_3 \,.$$

Diese Art von Abbildung wird vor allem für Konstruktionszeichnungen benutzt; die bekanntesten Beispiele sind Grundriß, Aufriß und Kreuzriß, bei denen jeweils $\vec{a}_3, \vec{a}_1$ *bzw.* $\vec{a}_2$ gleich dem Nullvektor sind und die beiden übrigen $\vec{a}_i$ die Einheitsvektoren des $\mathbb{R}^2$; konkret handelt es sich also um die Abbildungen

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix}, \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} y \\ z \end{pmatrix} \quad \text{und} \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ z \end{pmatrix}.$$

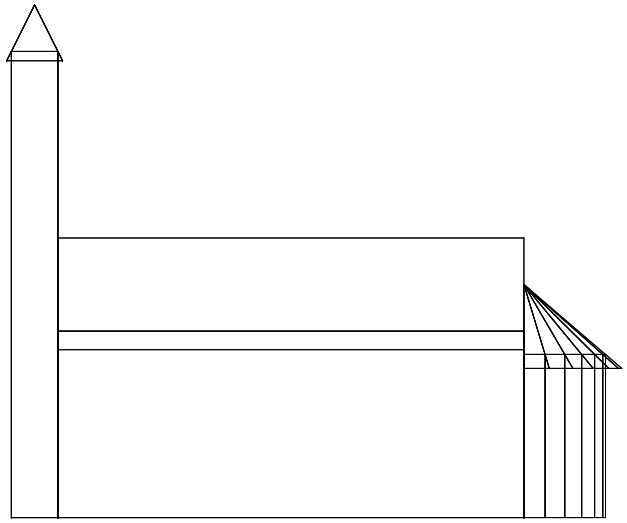


Grundriß



Aufriß

Einen „dreidimensionaleren“ Eindruck vermittelt die Isometrie nach DIN 5 (inzwischen aufgegangen in ISO 5456-3), bei der alle drei Vektoren Länge eins haben und zwei aufeinanderfolgende jeweils einen



Kreuzriß

Winkel von 120° einschließen; konkret ist

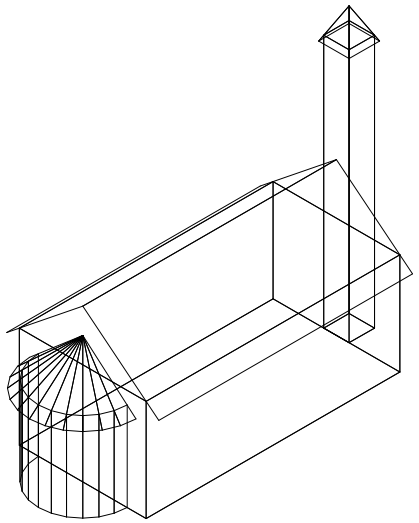
$$\vec{a}_1 = \begin{pmatrix} -\frac{1}{2}\sqrt{3} \\ -\frac{1}{2} \end{pmatrix}, \quad \vec{a}_2 = \begin{pmatrix} \frac{1}{2}\sqrt{3} \\ -\frac{1}{2} \end{pmatrix} \quad \text{und} \quad \vec{a}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Eine Variante davon ist die *isometrische Normalprojektion*, bei der diese Vektoren jeweils mit $\sqrt{2/3}$ multipliziert werden.

Ein Kompromiss zwischen Isometrien und Aufriß ermöglichen die *Dimetrien*, bei denen zwei Vektoren die gleiche Länge haben, der dritte aber nur halb so groß ist. Auf diese Weise wird die Vorderansicht des Objekts betont, ohne daß der räumliche Eindruck verloren geht. Gebräuchlich sind die *Ingenieursaxonometrie* mit

$$\vec{a}_1 = \begin{pmatrix} -\frac{3}{4} \\ -\frac{1}{4}\sqrt{7} \end{pmatrix}, \quad \vec{a}_2 = \begin{pmatrix} \frac{3}{4}\sqrt{7} \\ -\frac{1}{4} \end{pmatrix} \quad \text{und} \quad \vec{a}_3 = \begin{pmatrix} 0 \\ 2 \end{pmatrix},$$

sowie die Dimetrie nach DIN 5, bei der die drei Vektoren genau halb so groß sind. Eine Demonstration der genannten Abbildungen ist im Maple worksheet über Axonometrien zu finden, das (wie die weiter



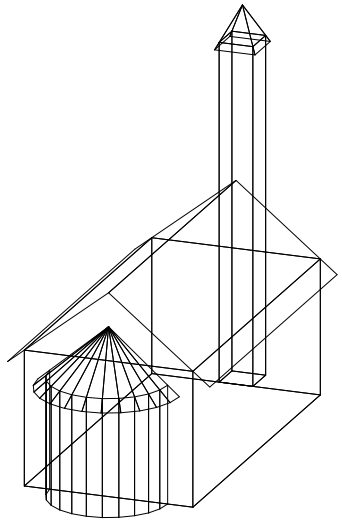
Isometrische Normalprojektion

hinten erwähnten worksheets) über die home page der Sommerschule erreicht werden kann. (Bei den meisten Rechnern wird man es wohl zunächst abspeichern müssen und dann Maple aufrufen.)

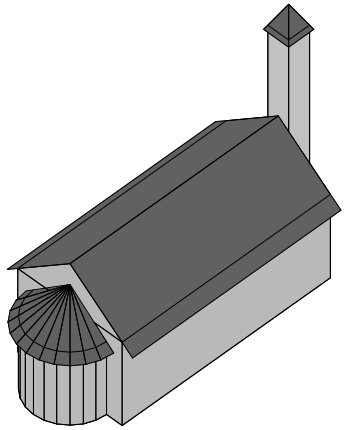
Anschaulicher, wenn auch in mancher Hinsicht weniger informativ, ist ein Bild, das an Stelle von Kanten Flächen projeziert und dabei auch auf verdeckte Linien achtet. Genau deshalb sieht ein solches Bild natürlicher aus, sagt dafür aber etwas weniger aus über die Geometrie. Ein Vorteil der Flächendarstellung ist natürlich die Möglichkeit Farben zu benutzen; seinen vollen Nutzen entfaltet ein solches Bild dann, wenn man es am Bildschirm von allen Seiten betrachten kann.

b) Perspektivische Projektionen und Kameramodell

Die zweite Strategie, die eher für anschauliche bis photorealistische Darstellungen benutzt wird, orientiert sich an einem Kameramodell: Wir stellen eine Kamera in den $\mathbb{R}^3$ und identifizieren den Bildschirm (oder das betrachtete Fenster) mit dem Stück Film, das diese belichtet. Als Kamera nehmen wir ein altbewährtes Modell, das bereits im fünften



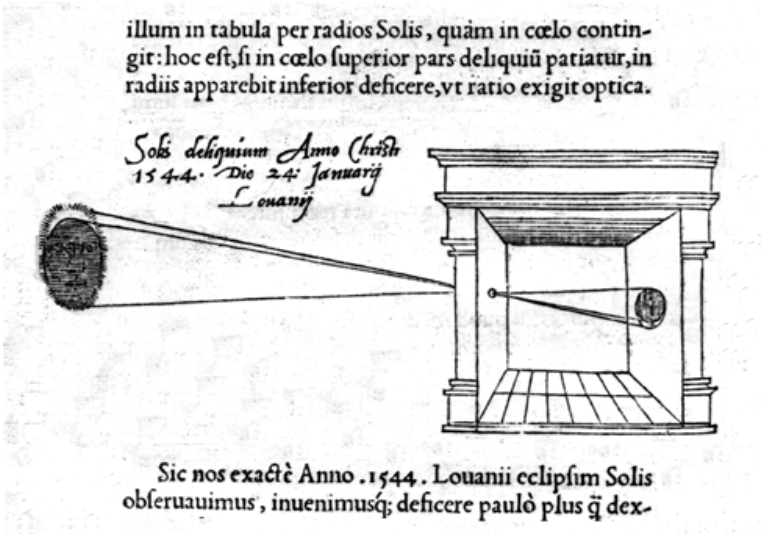
Ingenieursaxonometrie



Isometrische Projektion mit Flächen

vorchristlichen Jahrhundert von dem chinesischen Philosophen MO-TI oder MO TZU (470–391) beschrieben wurde und mit dem im darauf-

folgenden Jahrhundert auch ARISTOTELES (384–322) recht gute Erfahrungen machte: die *Camera obscura*.



Ihr großer Vorteil gegenüber moderneren Kamaras ist, daß sie anstelle eines Objektivs nur eine Lochblende hat, so daß es keinerlei Probleme mit der Tiefenschärfe gibt (für die wir rechnerisch aufwendige Faltungsintegrale bilden müßten). Natürlich kommt es immer wieder einmal vor, daß wir einen unscharfen Hintergrund brauchen, aber da können wir genausgut mit rechnerisch erheblich billigerem Nebel arbeiten.

Die *Camera obscura* definiert allerdings keine Abbildung von $\mathbb{R}^3$ nach $\mathbb{R}^2$: Zunächst einmal bildet sie nur Objekte ab, die sich *vor* der Kamera befinden. Diese Unzulänglichkeit der Realität ist mathematisch natürlich überhaupt kein Problem: Wir *definieren* einfach, daß auch bei Punkten hinter der Kamera der Bildpunkt gleich dem Schnittpunkt der Gerade zwischen Punkt und Lochblende mit der Filmebene sein soll. Ob diese Verbesserung der Natur wirklich wünschenswert ist, hängt von der Anwendung ab; statt darüber zu spekulieren, wollen wir lieber sehen, *wie* wir unsere Kamera in OpenGL im $\mathbb{R}^3$ aufstellen können.

Der einfachste Befehl dazu kommt aus der `glu`-Bibliothek *(GL utilities)* und hat die Form

`gluLookAt(` $x_0, y_0, z_0,\quad x_1, y_1, z_1,\quad v_1, v_2, v_3$ `);`

Dabei ist $A = (x_0, y_0, z_0)$ die Position der Lochblende, der Augpunkt also, $Z = (x_1, y_1, z_1)$ ist das *Zentrum*, auf das die Kamera gerichtet ist, und v_1, v_2, v_3 sind die Komponenten eines Vektors $\vec{v}$, der senkrecht auf der Oberseite der Kamera steht und somit deren Drehposition um die optische Achse AZ festlegt. Der Übergang vom Quer- zum Hochformat etwa entspricht der Drehung des Vektors $\vec{v}$ um 90°. Zum Einfluß des Zentrum auf das Aussehen des Bilds ist auf der home page der Sommerschule ein Maple worksheet zu finden.

Die übliche Konvention ist, daß man die Kamera auf der positiven z-Achse positioniert und auf den Nullpunkt schauen läßt; der Vektor $\vec{v}$ ist üblicherweise der Einheitsvektor der y-Achse.

Als nächstes muß festgelegt werden, welchen Ausschnitt des $\mathbb{R}^3$ wir tatsächlich abbilden wollen; bei einer modernen Kamera entspricht dies der Wahl eines Objektivs, bei der *Camera obskura* beschreibt es die Geometrie der Kamera.

OpenGL legt alle diese Dinge fest durch die Angabe des abzubildenden Bereichs. Dieser ist, wenn wir ein rechteckiges Bild wollen, ein Pyramidenstumpf, der in OpenGL zum Beispiel beschrieben werden kann durch

`glFrustum(` $x_1, x_2, y_1, y_2, z_1, z_2$ `);`

Dabei wird ein Koordinatensystem fest mit der Kamera verbunden, dessen Ursprung im Augpunkt liegt; die z-Achse ist die optische Achse AZ, die y-Achse geht in Richtung von $\vec{v}$, und die x-Achse bildet mit beiden ein kartesisches Rechtssystem. In diesem System hat dann das vordere Rechteck des Pyramidenstumps die Eckpunkte (x_i, y_j, z_1), und im hinteren Rechteck ist $z = z_2$. (Die Eckpunkte des hinteren Rechtecks sind damit nach dem Strahlensatz festgelegt.)

Man beachte, daß der Pyramidenstumpf nicht symmetrisch sein muß; man kann also auch Perspektiven erzeugen, die nur mit professionellen

Kameras möglich sind, bei denen die Filmebene und die Frontebene gegeneinander verschieb- und verschwenkbar sind. Die Hauptanwendung solcher Verschwenkungen, die Schärfedehnung nach SCHEIMPFLUG, spielt bei uns natürlich keine Rolle: Unsere Camera obscura bildet jeden Punkt im festgelegten Volumen perfekt und scharf ab; uns geht es nur um die Perspektive.

In der `glu`-Bibliothek gibt es für den häufig nützlichen Fall symmetrischer Pyramidenstümpfe die Alternative

`gluPerspective(` ω, r, z_1, z_2 `);`

die an Stelle der Ecken den Öffnungswinkel ω in der von AZ und $\vec{v}$ aufgespannten Ebene verwendet; dabei entspricht $\omega = 180$ der Fischaugenperspektive, und kleine Werte von ω entsprechen Teleperspektiven. z_1 *bzw.* z_2 sind wie oben, so daß

$$y_{1/2} = \pm z_1 \tan \frac{\omega}{2}$$

ist. Die noch fehlenden x-Koordinaten können nun leicht aus dem Verhältnis $r = x_1/y_1 = x_2/y_2$ berechnet werden.

OpenGL bietet noch eine dritte Alternative, die nicht wirklich kompatibel mit dem Kameramodell ist: Der Fall, daß der Pyramidenstumpf zu einem Quader degeneriert. Dies würde anschaulich bedeuten, daß die Kamera unendlich weit von der Szene entfernt ist; wir betrachten also beispielsweise den von der Sonne verursachten Schattenwurf der Szene. (Vergleichen mit den Dimensionen realer Objekte hier im Hörsaal kann die Entfernung zur Sonne für alle optischen Zwecke als unendlich angenommen werden.) Die Abbildung ist dann eine sogenannte orthogonale Projektion wie schon der oben betrachtete Grundriß, Aufriß und Kreuzriß. Der OpenGL-Befehl für diese Situation ist

`glOrtho(` $x_1, x_2, y_1, y_2, z_1, z_2$ `);`

wobei die Argumente genau dieselbe Bedeutung haben wie bei `glFrustum`, nur daß die hintere Begrenzungsfläche hier nicht durch einer perspektivischen Streckung, sondern durch eine orthogonale Parallelverschiebung aus der vorderen hervorgeht. z_1 und z_2 dienen hier nur zur

Positionierung des abzubildenden Quaders; die Lochblende muß als unendlich weit entfernt angenommen werden.

Übung im Umgang mit diesen Befehlen bekommt man mit dem Tutorium projection von NATE ROBINS, das (wie auch die im folgenden erwähnten anderen seiner Tutorien) über die home page der Sommerschule erreicht werden kann.

§ 2: Bewegungen im Raum

Wir wissen nun, wie ein dreidimensionales Volumen auf einen zweidimensionalen Bildschirm abgebildet werden kann, und *im Prinzip* können wir damit auch jede Perspektive realisieren. Tatsächlich aber wird das oftmals recht umständlich und auch unnatürlich sein:

Wenn wir etwa ein hinreichend kleines dreidimensionales Objekt von allen Seiten betrachten wollen, nehmen wir es in die Hand und drehen es; für jede Position des Objekts in der Hand muß dazu nach unserem bisherigen Ansatz eine neue Projektion betrachtet werden. Dies erscheint unnatürlich: Besser wäre es, von einer festen Abbildung auszugehen, und das Objekt im $\mathbb{R}^3$ jeweils neu zu positionieren.

Vom Resultat her sind beide Ansätze natürlich identisch: Niemand kann einem Film ansehen, ob die Kamera um einen Gegenstand herumgefahren ist oder sich der Gegenstand vor der Kamera gedreht hat – wenn man auch bei manchen Objekten wie etwa dem Mannheimer Wasserturm die zweite Alternative mit relativ hoher Wahrscheinlichkeit ausschließen kann.

a) Homogene Koordinaten

Bewegungen im $\mathbb{R}^3$ werden bekanntlich beschrieben durch Abbildungen der Form

$$\vec{v} \mapsto A\vec{v} + \vec{b}\,,$$

wobei A eine (orthogonale) 3×3-Matrix ist und $\vec{b}$ ein Vektor, der Verschiebungsvektor.

In der Computergraphik faßt man die Matrix A und den Vektor $\vec{b}$ zusammen zu einer 4×4-Matrix A^*, indem man zunächst den Vektor $\vec{b}$ als vierte Spalte an die Matrix anhängt und dann als vierte Zeile lauter Nullen einsetzt mit Ausnahme einer Eins an der letzten Stelle:

$$A^* = \begin{pmatrix} A & \vec{b} \\ \mathbf{0} & 1 \end{pmatrix},$$

wobei die fette Null für drei gewöhnliche Nullen steht. Mit dieser Bezeichnung ist dann, wie man leicht nachrechnet,

$$A^* \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} \quad \text{mit} \quad \begin{pmatrix} u \\ v \\ w \end{pmatrix} = A \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \vec{b}\,.$$

Die Darstellung links ist zwar rechnerisch etwas aufwendiger als die rechte, dafür aber etwas kompakter und hat vor allem den Vorteil, daß sich die Hintereinanderausführung zweier Abbildungen einfach durch ein Matrixprodukt beschreiben läßt.

Als Nachteil erscheint auf den ersten Blick, daß nun jeder Punkt *vier* Koordinaten hat. Dies hat aber durchaus auch Vorteile, so daß OpenGL (wie auch die meisten anderen Computgergraphiksysteme) tatsächlich stets mit allen vier Koordinaten rechnet, obwohl die vierte *fast* immer eins ist.

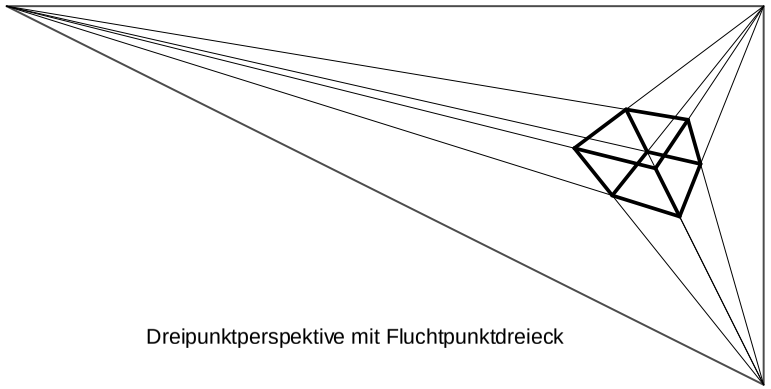
Wenn sie *nicht* eins ist, gilt die Konvention, daß es nur auf die *Verhältnisse* zwischen den Koordinaten ankommen soll; der Punkt mit homogenen Koordinaten $(x : y : z : u)$ entspricht also in klassischen kartesischen Koordinaten dem Punkt $\left(\frac{x}{u}, \frac{y}{u}, \frac{z}{u}\right)$ – falls $u \neq 0$ ist.

Wirklich interessant wird es, wenn $u = 0$ wird: Der Punkt $(1 : 0 : 0 : u)$ etwa hat für $u \neq 0$ die kartesische Darstellung $(1/u, 0, 0)$, ist also der Punkt auf der x-Achse mit Koordinate $1/u$. Geht u gegen null, wandert der Punkt also (je nach Vorzeichen links oder rechts) ins Unendliche; $(1 : 0 : 0 : 0)$ kann also als *unendlich ferner Punkt* der x-Achse aufgefaßt werden, und genauso ist allgemein $(x : y : z : 0)$ der unendlich ferne Punkt auf der vom Vektor (x, y, z) aufgespannten Geraden durch den Nullpunkt.

Solche unendlich ferne Punkte sind beispielsweise interessant für die Positinierung von Lichtquellen: Wenn wir etwa eine Szene mit Sonnenlicht beleuchten wollen, wäre es unsinnig, die genaue Position der Sonne in kartesischen Koordinaten anzugeben; es reicht völlig, wenn wir die Richtung angeben. Damit teilen wir dem System auch mit, daß nun für die Lichtintensität kein $1/r^2$-Gesetz mehr gilt, sondern daß die Intensität unabhängig von der Entfernung ist.

Im Tutorium lightposition von NATE ROBINS kann man die Unterschiede zwischen den beiden Fällen demonstrieren.

Gelegentlich können wir unendlich ferne Punkte auch auf dem Bildschirm sehen: Falls wir an einem Würfel die Geraden durch zwei parallele Kanten betrachten, schneiden diese sich natürlich nirgends. Ihre Bilder unter einer perspektivischen Projektion dagegen können durchaus einen Schnittpunkt haben: Parallelen werden schließlich nicht auf Parallelen abgebildet. Diesem Schnittpunkt entspricht dann im Dreidimensionalen der unendlich ferne Punkt in Richtung der betrachteten Kantenrichtung. In manchen Fällen, wie bei der untenstehenden Abbildung, sind auch alle drei dieser Punkte zu sehen; sie bilden ein Dreieck, dessen Kantenurbilder den $\mathbb{R}^3$ im Unendlichen umrunden.



Dreipunktperspektive mit Fluchtpunktdreieck

Schließlich sind unendlich ferne Punkte auch nützlich, um den Definitionsbereich von Abbildungen zu erweitern und dadurch Fallunterscheidungen zu vermeiden: Beim Kameramodell war bislang aus gutem

Grund nur von Punkten vor oder hinter der Frontebene die Rede. Punkte die *auf* der Frontebenen liegen, aber vom Augpunkt verschieden sind, haben mit diesem eine Verbindungsgerade, die ganz in der Frontebene liegt und somit die Filmebene in keinem endlichen Punkt schneidet. Gemäß der üblichen Konvention kann man aber sagen, daß der unendlich ferne Punkt der Verbindungsgeraden gleichzeitig unendlich ferner Punkt der Filmebene ist, so daß sich dieser Punkt als Bildpunkt anbietet. Nachrechnen zeigt, daß dies in der Tat genau der Punkt ist, auf den auch die in homogenen Koordinaten ausgedrückte Abbildungsformel führt – sofern man das Ergebnis so erweitert, daß es keine Nenner mehr enthält. Somit kann man für alle Punkte des $\mathbb{R}^3$ mit Ausnahme des Augpunkts selbst einen Bildpunkt definieren, der in homogenen Koordinaten durch eine lineare Abbildung beschrieben werden kann. (Der Augpunkt würde unter dieser Formel auf den Punkt mit homogenen Koordinaten $(0:0:0:0)$ abgebildet, und so einen Punkt gibt es nicht.)

b) Verschiebungen, Drehungen, Streckungen

Um 4×4-Matrizen in Aktion zu sehen, wollen uns einige gängige Transformationen anschauen. Am einfachsten sind die Translationen: Bei einer Verschiebung um den Vektor $\vec{b}$ mit Komponenten x_0, y_0, z_0 ist die Matrix A gleich der Einheitsmatrix, die 4×4-Matrix ist also

$$A^* = \begin{pmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix} .$$

OpenGL läßt uns diese Matrix auch einfacher erzeugen durch den Befehl

`glTranslatef(` x_0, y_0, z_0 `);`

wobei das `f` an Ende nur anzeigt, daß wir Gleitkommazahlen als Argumente verwenden; es gibt auch Versionen für Vektoren, ganze Zahlen, *usw.*

Drehungen sind etwas komplizierter: Zwar ist der Verschiebungsanteil stets gleich dem Nullvektor, da die Drehachse punktweise fest bleibt, aber dafür kann der Matrixanteil ziemlich komplex werden. Beginnen wir daher mit dem einfachsten Fall einer Drehung um die z-Achse.

Hier bleibt die z-Koordinate fest und für die (x, y)-Ebene haben wir das wohlbekannte Sinus-Cosinus-Drehkästchen; insgesamt erhalten wir also die Matrix

$$A^* = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} .$$

Ist die Drehachse eine beliebige Gerade durch den Nullpunkt, so können wir die Drehmatrix durch Basiswechsel aus der obigen erhalten, indem wir den Richtungsvektor der Drehachse als dritten Basisvektor nehmen, einen darauf senkrecht stehenden als ersten und das Kreuzprodukt dieser beiden als zweiten. Im neuen Koordinatensystem haben wir dann eine Drehung um die z-Achse; wir müssen also einfach, wie im ersten Studienjahr vielfach geübt, eine Abbildungsmatrix bezüglich einer neuen Basis darstellen. Dazu gehört insbesondere auch die Inversion der Matrix des Basiswechsels; hier können wir also wirklich froh sein, daß uns OpenGL die Arbeit abnimmt mit dem Befehl

`glRotatef(` φ, x_0, y_0, z_0 `);`

wobei φ den Drehwinkel (in Grad) angibt und x_0, y_0, z_0 die Komponenten eines Richtungsvektors der Drehachse sind.

Leider ist aber auch damit noch nicht die allgemeinste Rotation beschrieben: Wir müssen wir auch um Geraden rotieren können, die nicht durch den Nullpunkt gehen. Das läßt sich am einfachsten dadurch bewerkstelligen, daß wir zunächst um irgendeinen Vektor $\vec{v}$, der irgendeinen Punkt der Drehachse auf den Nullpunkt abbildet, *verschieben*, dann um die verschobene Drehachse rotieren, und schließlich noch um den Vektor $-\vec{v}$ verschieben.

Eine dritte, oft nützliche Transformation, ist die *Reskalierung*, die die Maßstäbe auf den drei Achsen unabhängig voneinander neu festlegt. Dies ist zwar keine Bewegung, aber doch oft nützlich: Beispielsweise kann man so Code zum Zeichnen einer Kugel wiederverwenden, um auch beliebige Ellipsoide zu zeichnen.

Das Tutorium transformation von NATE ROBINS demonstriert diese Befehle.

c) Matrixstacks und z-Puffer

Gerade in Animationen wollen wir oft mehrere Bewegungen miteinander kombinieren, und auch bei Rotationen um eine beliebige Achse haben wir gerade gesehen, daß man diese am einfachsten als Hintereinanderausführung einer Verschiebung, einer Drehung und dann der inversen Verschiebung realisiert. OpenGL trägt dem dadurch Rechnung, daß es jede neu erzeugte Matrix auf einem Stack ablegt, und für die Berechung der tatsächlichen Abbildung werden alle Matrizen auf dem Stack miteinander multipliziert.

Tatsächlich gibt es mehrere Stacks: Wie oben erwähnt, kann man dieselbe Projektion sowohl durch eine Kamerabewegung wie auch durch eine Bewegung Objekts realisieren. OpenGL hat daher für beides je einen Stack, und tatsächlich auch noch einen dritten für Texturen.

Um die Kamerapositionierung kümmert sich der Bewegungsstack, allerdings produziert auch `gluLookAt` eine Matrix, die auf eine eventuell dort bereits vorhandene draufmultipliziert wird. Da das in diesem Fall nur selten gewünscht wird, sollte man den Stack vorher löschen, d.h. irgendwo ganz am Anfang des Programms sollten Befehle der Art

```
glLoadIdentity();
gluLookAt( x0, y0, z0,   x1, y1, z1,   v1, v2, v3 );
```

stehen. (Zu Programmbeginn ist der Bewegungsstack automatisch ausgewählt.)

Befehle zur Kameraeinstellung beziehen sich auf den Projektionsstack, allerdings sorgt OpenGL nicht automatisch dafür, daß die entsprechende Vierermatrix dort landet, und es sorgt auch nicht dafür, daß dieser Stack vorher geleert wird. Beides muß der Anwender explizit fordern durch

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
```

Erst danach können Befehle folgen wie

```
glFrustum( x1, x2, y1, y2, z1, z2 );
```

Danach sollte man gleich den Befehl


```
glMatrixMode( GL_MODELVIEW );
```

geben, um auf den Bewegungsstack zurückzuschalten; ansonsten führen Anweisungen wie `glTranslatef`, `glRotatef` zu unerwarteten Ergebnissen.

Ein aufmerksamer Leser wird sich wahrscheinlich schon seit langem fragen, warum wir für eine Abbildung von $\mathbb{R}^3$ nach $\mathbb{R}^2$ einen Stack von 4×4-Matrizen betrachten: Der Bildschirm ist schließlich nur zweidimensional, und selbst wenn wir auch dort homogene Koordinaten einführen, gibt es nur drei davon, so daß eine 4×3-Matrix ausreichen sollte.

Tatsächlich sorgen seit einiger Zeit die Graphikkarten dafür, daß mittlerweile selbst die Bildschirme billiger Computer *dreidimensional* sind: Bei jeder Projektion geht natürlich Information verloren, und viele Punkte des Raums werden auf denselben Punkt der Ebenen abgebildet. Auf dem Bildschirm wollen wir aber nur einen davon sehen, nämlich – genau wie auch in der Realität – den mit dem geringsten Abstand zum Augpunkt. Diesen Punkt zu finden kann, abstrakt betrachtet, äußerst schwierig sein.

Tatsächlich aber gibt es in der Computergraphik immer nur endlich viele Objekte, die abgebildet werden sollen, und so konnte sich ein zwar wenig eleganter, dafür aber sehr effizienter Rambo-Algorithmus durchsetzen: Jedes Objekt wird einzeln in die Bildschirmebene projeziert, aber zusätzlich zu den zwei kartesischen Koordinaten des Bildschirms wird auch noch der Abstand z von der Frontebene der Kamera (oder vom der Bezugsebenen im Falle einer orthogonalen Projektion) mit berechnet. Im Bildschirmspeicher der Graphikkarte werden nicht nur die drei Farbkoordinaten des Urbilds gespeichert, sondern auch seine z-Koordinate. Sobald nun ein weiteres Objekt einen Bildpunkt liefert, der auf dasselbe Pixel abgebildet wird, werden die z-Koordinaten verglichen, und nur wenn die z-Koordinate des neuen Punkts keiner ist als die bislang eingetragene, ersetzt seine Farb- und z-Koordinaten-Information den gespeicherten Wert. So ist sichergestellt, daß nach Abarbeiten der gesamten Szene nur die dem Augpunkt am nächsten liegenden Objektteile gezeigt werden.

Kapitel 2
Geometrische Objekte

§ 1: Polyeder und Flächen

Nachdem wir nun wissen, wie geometrische Objekte auf den Bildschirm abgebildet werden, sollten wir uns auch damit befassen, was geometrische Objekte eigentlich sind. Der Vielfalt möglicher Anwendungen entsprechend gibt es eine ganze Reihe von Möglichkeiten.

a) Polyeder

Das ist die einfachste Variante, in die im Endeffekt allerdings alle anderen Varianten übersetzt werden: Die Bausteine sind konvexe ebene Polygone, die durch Angabe ihrer Ecken spezifiziert werden. Dazu kann noch für *jede* Ecke deren Farbe festgelegt werden; bei verschiedenfarbigen Ecken werden die Farben im Innern des Polyeders interpoliert.

Obwohl Polygone in OpenGL eben sein müssen, kann man doch für jede Ecke einen eigenen Normalenvektor festlegen, der nichts mit dem tatsächlichen Normalenvektor zu tun haben muß.

Normalenvektoren haben vor allem zwei Funktionen: Zum ersten legen sie fest, welcher Seite der Fläche innen und welche außen sein soll. Standardmäßig wird nur die Außenseite dargestellt, jedoch läßt sich diese Einstellung auch überschreiben.

Zum zweiten sind Normalenvektoren wichtig für die Beleuchtung: Bei allen Lichtquellen, die eine Richtung haben, bestimmt der Normalenvektor in einem Punkt, wie der Strahl reflektiert wird. Ein falscher Normalenvektor kann daher dazu führen, daß auch ein Objekt, das sich

im abzubildenden Volumen befindet, unsichtbar bleibt, da die dem Auge zugewandte Seite entweder fälschlicherweise als Innenseite interpretiert wird oder aber das vom Objekt zurückgeworfene Licht den Augpunkt nicht erreicht.

Wenn alles korrekt definiert ist, können „falsche“ Normalenvektoren dazu beitragen, via GOURAUD-Shading Kanten wegzuretouchieren: In der `glut`-Bibliothek beispielsweise werden Kugeln, Tori und auch die ehemalige des Computer Science Departments von Utah (mittlerweile im Computermuseum in Boston) durch Polyeder angenähert, deren Normalenvektoren nicht nach den Richtungen am Polyeder, sondern nach denen am realen Objekt ausgerichtet sind. Zwecks Modellierung einer Kugel beschreibt man also der Kugel ein Polyeder ein, und nimmt als Normalenvektor einer jeden Ecke den Radiusvektor an dieser Stelle.

Die primitivste und universellste Art der Erzeugung von Polygonen ist die Anweisungsfolge für ein N-Eck:

```
glBegin( GL_POLYGON );
    glColor3f( r1, g1, b1 );
    glNormal3f( nx1, ny1, nz1 );
    glVertex3f( x1, y1, z1 );
    glColor3f( r2, g2, b2 );
    glNormal3f( nx2, ny2, nz2 );
    glVertex3f( x2, y2, z2 );
        ...
    glColor3f( rN, gN, bN );
    glNormal3f( nxN, nyN, nzN );
    glVertex3f( xN, yN, zN );
glEnd();
```

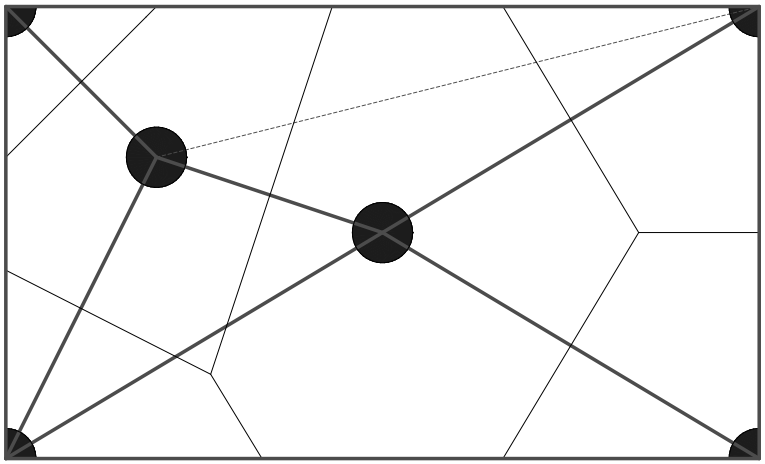
Die Befehle zur Festlegung der Farbe und des Normalenvektors können auch ganz oder teilweise entfallen; in diesen Fällen wird die bisherige Farbe weiterverwendet und, falls es mindestens einen gibt, der bisherige Normalenvektor. Normalenvektoren sollten Einheitsvektoren sein.

Für häufig benutzte Polygone wie Dreiecke, Rechtecke und Folgen davon gibt es auch kompaktere Befehle; einige davon sind im Tutorium shapes von NATE ROBINS zu finden.

Gelegentlich will man Normalenvektoren nicht nur an den Eckpunkten eines Polygons festlegen, sondern auch an gewissen Punkten im Innern. Dazu bietet OpenGL keine direkte Möglichkeit; man muß stattdessen das Polygon so zerteilen, daß die fraglichen Punkte zu Ecken werden. Ein populärer Algorithmus für eine solche Zerteilung ist die DELAUNAY-Triangulierung, benannt nach dem russischen Mathematiker Борис Николаевич Делоне (1890–1980).

Als erstes wird zu jedem der betrachteten Punkte P_i einschließlich der Ecken des Polygons die *Elementarzelle* konstruiert. Diese heißt auch *Wirkungsbereich*, VORONOI-*Bereich* oder WIGNER-SEITZ-*Zelle;* gemeint ist jeweils die Menge aller Punkte, deren Abstand zu P_i kleiner ist als der zu jedem anderen P_j.

Als nächstes verbindet man zwei Punkte P_i und P_j genau dann durch eine Strecke, wenn ihre Elementarzellen benachbart sind, deren Abschlüsse also eine Kante gemeinsam haben (die dann auf der Mittelsenkrechten der Verbindungsstrecke liegt.). Dies zerlegt das Polygon in kleinere Polygone, wobei nun alle P_i als Ecken auftreten.



DELAUNAY-Triangulierung eines Rechtecks mit zwei inneren Punkten

Wenn die Punkte im Innern keine sehr spezielle Lage haben, werden die entstehenden Polygone Dreiecke sein; andernfalls muß man, falls man wirklich eine DELAUNAY-*Triangulierung* möchte, Polygone mit mehr als drei Ecken weiter zerlegen, indem man Verbindungsstrecken zwischen nicht benachbarten Kanten einzieht. Echte Triangulierungen sind beispielsweise wichtig als Definitionsbereiche für einige der im nächsten Abschnitt betrachteten parametrischen Flächen.

b) Parametrische Flächen

In der Frühzeit der Computergraphik war dies die einzige Möglichkeit, Flächen darzustellen: Eine Fläche wird zusammengesetzt aus *Flächenstücken,* die als Einbettungen einfacher Gebiete der Ebene in den $\mathbb{R}^3$ definiert sind.

Ein Flächenstück soll demnach gegeben sein durch eine Funktion

$$f\colon B \to \mathbb{R}^3; \quad (u,v) \mapsto f(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}$$

auf einer Teilmenge $B \subset \mathbb{R}^2$; um gut damit rechnen zu können, wollen wir f als stetig differenzierbar voraussetzen in einer offenen Umgebung von B. In der Computergraphik ist B meist ein Dreieck oder ein Rechteck, und $x(u,v)$ sowie $y(u,v)$ sind rationale Funktionen wenn nicht gar Polynome in u und v.

Als Graphik noch per Kugelschreiber auf Flachbett- und Trommelplottern dargestellt werden mußte, hatten diese Flächenstücke den großen Vorteil, sich einfach zeichnen zu lassen: Man wählte einfach Strecken in B und stellt deren Bild dar: Zunächst abstrakt als Kure im $\mathbb{R}^3$, und dann, via Projektion, als ebenes Kurvenstück. Oft nimmt man als Strecken sogenannte *Parameterlinien,* die dadurch entstehen, daß man in obiger Abbildungsvorschrift eine der Koordinaten festhält und nur die andere laufen läßt.

Heute spielen diese Drahtgittermodelle keine Rolle mehr; heute geht es darum, die Fläche zweidimensional darzustellen, d.h. sie durch Polygone anzunähern, die gegebenenfalls beleuchtet und in jedem Fall auf den Bildschirm projeziert werden.

Zumindest im Beleuchtungsfall muß man die Normalenvektoren kennen (die dann den Polygonecken an Stelle des „echten“ Normalenvektors des Polygons zugeordnet werden), aber das ist recht einfach: Der Normalenvektor steht senkrecht auf der Tangentialebene, also berechnen wir zunächst diese. Auch dazu dienen die Parameterlinien: Diese sind Kurvenstücke im folgenden Sinne:

Definition: Ein *Kurvenstück* γ ist eine stetig differenzierbare Abbildung

$$\gamma\colon [a,b] \to \mathbb{R}^3$$

eines Intervalls in den $\mathbb{R}^3$.

Für ein Kurvenstück können wir den zugehörigen Tangentenvektor im Punkt $\gamma(t_0)$ leicht bestimmen: Er ist einfach die Ableitung $\dot\gamma(t_0)$, der die bestmögliche Linearisierung von γ in der Umgebung des betrachteten Punkts liefert.

Für ein Flächenstück betrachten wir im Punkt $f(u_0,v_0)$ die beiden Parameterlinien

$$\gamma_1\colon \begin{cases} (u_0-\varepsilon, u_0+\varepsilon) \to \mathbb{R}^3 \\ t \mapsto f(t,v_0) \end{cases} \text{ und } \gamma_2\colon \begin{cases} (v_0-\varepsilon, v_0+\varepsilon) \to \mathbb{R}^3 \\ t \mapsto f(u_0,t) \end{cases}$$

sowie ihre Tangentenvektoren $\dot\gamma_1(0)$ und $\dot\gamma_2(0)$ im Punkt

$$f(u_0,v_0) = \gamma_1(0) = \gamma_2(0)\,.$$

Falls diese beiden Vektoren nicht linear unabhängig sind, haben wir ein Problem: Dann ist das Flächenstück, zumindest in linearer Näherung, im betrachteten Punkt nur eindimensional. Für visuell zweidimensionale Flächenstücke sollte das bei einer guten Parametrisierung nicht vorkommen; wir fordern deshalb, daß diese beiden Vektoren in jedem Punkt linear unabhängig sein sollen. (Man sagt dann auch, das Flächenstück sei *regulär*.)

Alsdann ist das Kreuzprodukt

$$\vec n(u_0,v_0) \underset{\text{def}}{=} \dot\gamma_1(0) \times \dot\gamma_2(0)$$

nicht der Nullvektor und steht senkrecht sowohl auf $\gamma_1(0)$ als auch auf $\gamma_2(0)$, also auf der gesamten Tangentialebene, und damit ist $\vec n(u_0,v_0)$ ein Normalenvektor an die Fläche im Punkt $f(u_0,v_0)$.

c) Implizite Flächen

Die Darstellung als parametrische Fläche ist nicht immer die natürlichste: Bei der Kugel

$$x^2+y^2+z^2 = r^2$$

etwa wäre die einfachste parametrische Darstellung die durch Längen- und Breitengrade, also z.B.

$$f\colon \begin{cases} [0,\,2\pi)^2 \to \mathbb{R}^3 \\ (u,v) \mapsto \begin{pmatrix} r\cos u\cos v \\ r\sin u\cos v \\ r\sin u \end{pmatrix} \end{cases}.$$

Eine auch nur stückweise Darstellung durch Polynome kann es offensichtlich (wie auch bereits beim Kreis) nicht geben, denn die Summe zweier oder dreier Polynomquadrate kann nur dann konstant sein, wenn alle Polynome konstant sind. Eine rationale Darstellung ist möglich, wenn auch nicht ganz einfach: Nachdem man sich durch Ausmultiplizieren davon überzeigt hat, daß

$$\gamma\colon \mathbb{R} \to \mathbb{R}^2;\quad t \mapsto \left(r\cdot\frac{t^2-1}{t^2+1},\; r\cdot\frac{2t}{t^2+1}\right)$$

die Kreislinie mit Ausnahme des Punkts $(r,\,0)$ parametrisiert (für letztere bräuchten wir den unendlich fernen Punkt ∞ von $\mathbb{R}$), folgt auch leicht, daß

$$f\colon \mathbb{R}^2 \to \mathbb{R}^3;\quad (u,v) \mapsto r\cdot\begin{pmatrix} \frac{(u^2-1)(v^2-1)}{(u^2+1)(v^2+1)} \\ \frac{2u(v^2-1)}{(u^2+1)(v^2+1)} \\ \frac{2u}{u^2+1} \end{pmatrix}$$

die Kugeloberfläche mit Ausnahme des Punkts $(r,0,0)$ parametrisiert. Verglichen mit der harmlosen Kugelgleichung ist diese Darstellung erheblich aufwendiger und rechnerisch unangenehmer.

Genau wie man Flächen aus parametrischen Flächenstücken zusammensetzen kann, kann man sie auch aus impliziten Flächenstücken zusammensetzen; eine Methode dafür, die sogenannten *A-patches* mit A für *algebraisch* wurden von CHANDRAJIT BAJAJ, dem Hauptredner dieser

Sommerschule, entwickelt. Tatsächlich sind implizite Flächen sogar allgemeiner, denn während sich jede rational parametrisierte Fläche auch als implizite algebraische Fläche darstellen läßt, mit f als Polynom also, ist das umgekehrte nur in wenigen sehr speziellen Ausnahmefällen möglich.

Implizite Flächen sind natürlich nur dann nützlich, wenn man sie erstens durch Polygone annähern und so auf den Bildschirm bringen kann – dafür gibt es, insbesondere für A-patches, effiziente Algorithmen. Als zweites müssen wir die Normalenvektoren berechnen können, und das ist hier deutlich einfacher als im Fall der parametrischen Flächen: Für eine differenzierbare Funktion f, einen Punkt $(x_0, y_0, z_0) \in \mathbb{R}^3$ mit $f(x_0, y_0, z_0) = 0$ und einen Vektor mit Komponenten a, b, c ist

$$\begin{aligned} & f(x_0 + ha, y_0 + hb, z_0 + hc) \\ &= f(x_0, y_0, z_0) + h \begin{pmatrix} a \\ b \\ c \end{pmatrix} \nabla f(x_0, y_0, z_0) + o(h) \\ &= h \begin{pmatrix} a \\ b \\ c \end{pmatrix} f(x_0, y_0, z_0) + o(h)\,, \end{aligned}$$

die Tangentialebene steht also senkrecht auf dem Gradienten, der somit gleich dem Normalenvektor ist. Falls man Tangentenvektoren braucht, die die Ebene aufspannen, reicht es, Lösungen der linearen Ebenengleichung hinzuschreiben.

Als Beispiel können wir den Normalenvektor im Punkt (x_0, y_0, z_0) an die Kugel mit Radius r berechnen: Der Gradient von

$$f(x, y, z) = x^2 + y^2 + z^2 - r^2$$

im Punkt (x_0, y_0, z_0) ist

$$\nabla f(x_0, y_0, z_0) = 2 \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}\,,$$

der Normalenvektor hat also, was hoffentlich niemanden überrascht, die gleiche Richtung wie der Vektor vom Mittelpunkt der Kugel zum Punkt (x_0, y_0, z_0). Wer möchte, kann dieses Ergebnis zum Vergleich auch aus den beiden obigen Parameterdarstellungen herleiten.

§2: CSG-Darstellung eines dreidimensionalen Objekts

CSG steht für *computational solid geometry*, wobei *solid geometry* das englische Wort für *dreidimensionale Geometrie* oder *Stereometrie* ist. Der wesentliche Punkt der CSG-Darstellung ist die Synthese komplexer dreidimensionaler Objekte oder Szenen aus (im allgemeinen) wenigen *primitiven* Objekten mittels einiger weniger Verknüpfungsregeln. Die notwendige Flexibilität wird dadurch erreicht, daß sowohl die primitiven Objekte als auch die Verknüpfungsregeln im allgemeinen von Parametern abhängen.

a) Primitive Objekte

Die „primitiven Objekte“ können je nach Ansatz von völlig verschiedener Natur sein. Über ihre exakte Implementierung wollen wir uns hier keine Gedanken machen; dafür gibt es Graphiksysteme wie OpenGL, CAD-Systeme und so weiter. Hier sollen nur einige populäre Wahlen für solche Objekte aufgeführt werden, die sich möglichst gut an die aus §1 gekannten anlehnen.

1) Solid modeling: Das Wort *solid* in diesem Zusammenhang ein Substantiv mit der deutschen Übersetzung „(geometrischer) Körper“. Bei dieser ältesten Variante der geometrischen Modellierung, die an die klassische Darstellende Geometrie anknüpft, sind die primitiven Objekte einfache Objekte der Elementargeometrie wie Quader, Dreiecksprismen, Zylinder, Kegel sowie (manchmal) Kugeln und Tori. Ihre geometrische Gestalt wird durch Parameter festgelegt; ihre Lage und Orientierung im Raum ist zunächst in einer festen Standardposition.

So könnte beispielsweise das geometrische Objekt

$$\texttt{Quader}(a, b, c)$$

ein achsenparalleler Quader sein, der in x-Richtung die Kantenlänge a hat, in y- und z-Richtung entsprechend b und c. Seine Lage kann beispielsweise dadurch festgelegt sein, daß die linke hintere untere Ecke gleich dem Nullpunkt ist.

Entsprechend können wir ein Objekt

$$\texttt{Zylinder}(r, h)$$

definieren, das einen Zylinder mit Höhe h und Radius r beschreibt, dessen Achse die z-Achse zwischen $z = 0$ und $z = h$ ist. In der `glu`-Bibliothek beispielsweise könnte dieser erzeugt werden durch den Aufruf

```
GLUquadricObj *zyl;
zyl = gluNewQuadric();
gluCylinder( zyl, r,r,h,n,m );
```

Hier steht der Radius zweimal, da die gleiche Routine auch Kegel und Kegelstümpfe zeichnen kann, für die zunächst der untere und dann der obere Radius angegeben wird. n und m geben an, durch wie viele Scheiben und „Schnitze“ der Zylinder polygonal approximiert werden soll.

Aufgrund der Einfachheit der primitiven Objekte lassen sich sowohl diese selbst als auch die daraus zusammengesetzten Objekte gut manipulieren; auch photorealistische Darstellungen sind problemlos.

2) Kinematische Erzeugung: Bei dieser Methode wird ein primitives Objekt dadurch definiert, daß eine Fläche wie beispielsweise eine Kreisscheibe entlang einer Raumkurve geführt wird; das Objekt besteht aus allen Punkten, die dabei überstrichen werden.

Um diese Definition eindeutig zu machen, muß natürlich der Punkt auf der Fläche festgelegt werden, der sich wirklich auf der Kurve bewegt. Außerdem braucht man einen von diesem Punkt ausgehenden Vektor, der die Orientierung der Fläche im Raum festlegt, beispielsweise so, daß er stets parallel zum Tangentenvektor der Raumkurve sein soll.

Als Beispiel betrachten wir eine Kreisscheibe mit dem Mittelpunkt als ausgezeichnetem Punkt; der Vektor dort sei ein Normalenvektor, stehe also senkrecht auf der Kreisscheibe. Bewegt sich diese Fläche nun entlang einer Strecke, so entsteht ein Zylinder; bewegt sie sich entlang eines Kreises mit hinreichend großem Radius, entsteht ein Torus.

Bei komplizierten Flächen und Raumkurven kann es recht aufwendig sein, beispielsweise für Beleuchtungseffekte, die Randfläche des so definierten Körpers zu bestimmen; wenn es in erster Linie um photorealistische Graphik geht, ist diese Art von primitiven Objekten also nicht zu empfehlen. Andererseits lassen sich Objekte, die auf einer numerisch gesteuerten Werkzeugmaschine mit Bohrern, Fräsen und ähnlichem hergestellt werden sollen, hervorragend auf diese Weise modellieren, und die geometrische Beschreibung des Objekts kann leicht in Steuerdaten für die Werkzeugmaschine übersetzt werden.

3) Algebraische Halbraumdarstellung: Hier sind die primitiven Objekte algebraische Halbräume, d.h. Mengen der Form

$$\left\{(x, y, z) \in \mathbb{R}^3 \;\middle|\; f(x, y, z) \leq 0\right\},$$

wobei f ein irreduzibles Polynom in x, y und z ist[*)].

Für das Polynom

$$f(x, y, z) = x^2 + y^2 + z^2 - R^2$$

etwa erhalten wir eine Kugel, für

$$g(x, y, z) = x$$

den gesamten Halbraum, in dem die x-Koordinate nicht positiv ist. Insbesondere gibt es also bei diesem Ansatz also auch primitive Objekte, die nicht endlich sind; sie sind nützlich zum Schneiden mit anderen Objekten. Beispielsweise ist der Durchschnitt der beiden gerade definierten algebraischen Halbräume eine Halbkugel.

Das Arbeiten mit Polynomen hohen Grades kann recht aufwendig sein; deshalb erlauben Systeme, die mit algebraischen Halbräumen arbeiten, gelegentlich nur Polynome vom Grad höchstens drei oder gar nur zwei. Dadurch vereinfachen sich die Algorithmen beträchtlich, allerdings verliert man auch Flexibilität und muß dies eventuell durch eine erheblich größere Anzahl von Halbräumen zur Darstellung eines gewünschten Objekts erkaufen.

[*)] Ein Polynom heißt irreduzibel, wenn es nicht als Produkt zweier nichtkonstanter Faktoren geschrieben werden kann.

4) Normteile: Zumindest aus Sicht des Anwenders kann man auch deutlich kompliziertere Objekte als *primitiv* betrachten: Im Maschinenbau etwa gibt es eine Vielzahl von Teilen, die immer wieder verwendet werden: Das sind einmal Normteile wie Schrauben, Nieten, Scheiben, Bolzen, Wälzlager, ..., die in einer Vielzahl von DIN-Normen, europäischen Normen und ISO-Normen beschrieben sind[*)], dann aber auch Katalogteile von Zulieferfirmen und mehrfach verwendbare Eigenentwicklungen.

Von der Implementierung her sind Norm- und Katalogteile natürlich alles andere als primitive Objekte; sie sind nach irgendeinem der hier oder im nächsten Paragraphen behandelten Prinzipien konstruiert. Allerdings sind sie oft unabhängig vom Rest des Systems implementiert und mit diesem nur über eine Schnittstelle verbunden:

Auf Initiative der Anwendert gründete hier das Deutsche Institut für Normung (DIN) zusammen mit dem Verband der Automobilindustrie (VDA), dem Verband Deutscher Maschinen und Anlagenbau (VDMA) und dem Zentralverband der Elektrotechnik und Elektronikindustrie (ZVEI) im April 1988 die DIN Software GmbH, die nicht nur Dateien mit CAD-Normteiltabellen (nach DIN V 4001), sondern auch Programme mit genormter Schnittstelle (DIN V 66304) für CAD-Systeme vertreibt.

Bei den Anbietern von CAD-Systemen stieß dies naturgemäß auf sehr geringe Begeisterung; trotzdem gibt es zumindest im (aus Sicht der geometrischen Modellierung steinzeitlichen) zweidimensionalen CAD praktisch kein marktrelevantes CAD-System mehr, das seine eigenen Normteile anbietet. Stattdessen ist ein Modul zur Realisierung der Schnittstelle nach DIN V 66304 integriert, über den die CAD-Geometrieprogramme der DIN Software AG sowie auch die für Zulieferund Werksteile unabhängig vom CAD-System integriert werden können. Für den Endanwender, den Konstrukteur also, sind diese Teile primitive Objekte, deren innerer Aufbau ihm nicht bekannt sein muß und

*) Allein die CAD-Normteildatei des DIN enthält rund 1000 Normen

möglicherweise auch nur wenig mit der Designphilosophie seines eigenen CAD-Systems zu tun hat.

b) Geometrische Manipulation primitiver Objekte

Die bislang definierten primitiven Objekte haben eine feste Position und Orientierung im Raum; bevor sie für eine geometrische Konstruktion verwendet werden können, müssen sie also noch in die richtige Position gebracht werden. Dies geschieht durch Verschiebungen und Drehungen, durch die klassischen Bewegungen der EUKLIDischen Geometrie also, die wir aus dem ersten Kapitel kennen.

c) Regularisierte Boolesche Operationen

Um primitive Objekte zu einer interessanten Konstruktion oder Szene zusammenzustellen, brauchen wir Regeln, mit denen diese miteinander kombiniert werden können. Das sind im wesentlichen die üblichen mengentheoretischen Operationen, allerdings nicht ganz:

Im Rahmen der geometrischen Modellierung wäre es unsinnig, zwischen der offenen Kugel

$$\left\{(x,y,z) \in \mathbb{R}^3 \;\middle|\; x^2 + y^2 + z^2 < 1\right\}$$

und der geschlossenen Kugel

$$\left\{(x,y,z) \in \mathbb{R}^3 \;\middle|\; x^2 + y^2 + z^2 \leq 1\right\}$$

zu unterscheiden: Ein Unterschied zwischen den beiden Mengen läßt sich weder visuell erkennen noch physikalisch konstruieren. Da der Rand eines Objekts für die graphische Darstellung sehr wichtig ist, wollen wir vereinbaren, daß wir grundsätzlich nur abgeschlossene Teilmengen von $\mathbb{R}^3$ betrachten.

Außerdem interessieren in der geometrischen Modellierung nur *dreidimensionale* Objekte; das zweidimensionale Quadrat

$$\left\{(x,y,z) \in \mathbb{R}^3 \;\middle|\; |x| \leq 1, |y| \leq 1, z = 0\right\}$$

ist also kein Objekt einer sinnvollen geometrischen Modellierung: Solche Mengen werden erst interessant, wenn wir aus dem dreidimensionalen geometrischen Modell zweidimensionale Ansichten erzeugen.

Wir erstetzen daher die üblichen mengentheoretischen Operationen durch sogenannte *regularisierte* Versionen

$$A \cup^* B \underset{\text{def}}{=} \overline{A^\circ \cup B^\circ}, \quad A \cap^* B \underset{\text{def}}{=} \overline{A^\circ \cap B^\circ}, \quad \text{und} \quad A \setminus^* B \underset{\text{def}}{=} \overline{A^\circ \setminus \overline{B}}.$$

Dabei steht M° wie üblich für das Innere einer Menge und $\overline{M}$ für den Abschluß. Da wir zur geometrischen Modellierung nur abgeschlossene Teilmengen des $\mathbb{R}^3$ betrachten, ist die Bildung des Abschlusses von B vor der Berechnung der Differenz nicht wirklich notwendig; sie ist nur erwähnt um auch typographisch klarzumachen, daß wir zunächst nur die inneren Punkte des Resultats berechnen.

Geometrisch betrachtet bedeuten diese Definitionen, daß wir Randpunkte zunächst außer acht lassen, wenn wir die Mengen vereinigen, schneiden oder ihre Differenz bilden; danach berechnen wir zusammen mit dem Abschluß auch den neuen Rand für die entstehende Mengen.

Der Unterschied zu den klassischen Operationen wird am besten anhand einiger Beispiele klar:

Seien A und B zwei Quader, die sich nur am Rand berühren. Dann haben A und B keine inneren Punkte gemeinsam, $A^\circ \cap B^\circ$ ist also die leere Menge, und damit ist auch $A \cap^* B = \emptyset$.

Die reguläre Vereinigung unterscheidet sich nicht von der gewöhnlichen; in beiden Fällen werden (fast alle) gemeinsame Randpunkte zu inneren Punkten, genauso wie man es für die meisten Anwendungen auch haben möchte: Ein L-förmiges Werkstück, das für die geometrische Modellierung aus zwei Quadern zusammengesetzt wird, sollte die (von der Geometrie des Werkstücks her nicht eindeutig definierte) Grenzfläche der beiden Quadern nicht als ausgezeichnete Fläche enthalten, schon gar nicht, wenn wir die (Rand-)Flächen für die Berechnung von Beleuchtungseffekten verwenden.

Es gibt allerdings auch Fälle, in denen gemeinsame Randpunkte nicht verschwinden sollten: Falls ein konstruiertes Objekt anschließend für Simulationen von Eigenschaften wie Elastizität, Bruchfestigkeit oder Umströmungsverhalten weiterverarbeitet werden soll, ist es sehr nützlich, ein in Zellen unterteiltes Objekt zu haben, denn das übliche Verfahren zur numerischen Integration partieller Differentialgleichungen,

die Methode der finiten Elemente, arbeitet mit solchen Zerlegungen, und für krummlinig begrenzte Objekte kann es recht schwierig sein, eine solche Zerlegung zu finden. Man sollte daher in solchen Fällen die „verschwindenden“ Randflächen nicht vergessen, sondern nur notieren, welche Flächen zum Rand der entstehenden Gesamtfigur gehören und welche nicht. Dazu können auch weitere Unterteilungen notwendig sein, etwa wenn sich zwei Quader nur in Teilen einer Randfläche schneiden wie bei einem aus zwei Quadern zusammengesetzten L- oder T-Stück.

Regularisierte Differenzen sind vor allem dann interessant, wenn die zweite Menge eine Teilmenge der ersten ist; ist $B \subset A$ etwa ein Zylinder im Quader A, so ist $A \setminus^* B$ das Resultat einer Bohrung.

d) Der CSG-Baum eines Objekts

Ein geometrisches Objekt in CSG-Darstellung wird üblicherweise dargestellt durch einen Baum. Die Wurzel dieses Baumes repräsentiert das Objekt selbst, die Blätter sind die primitiven Objekte, aus denen es zusammengesetzt ist.

Die restlichen Knoten stehen entweder für geometrische Transformationen oder für regularisierte BOOLEsche Operationen. Im ersten Fall geht vom entsprechenden Knoten genau eine Kante aus, an der ein Baum hängt, der das Objekt beschreibt, auf das die Transformation angewandt wird. Im zweiten Fall gibt es zwei Kanten, an denen Bäume für die beiden Objekte hängen, auf die die BOOLEsche Operation angewandt wird. (Es ist Geschmacksache, ob man bei der regularisierten Vereinigung und beim regularisierten Durchschnitt, die ja beide kommutativ und assoziativ sind, eventuell auch mehrere Operanden zulassen will.)

Als Beispiel betrachten wir ein L-Stück, dessen lange Seite eine zylindrische Bohrung aufweise. Das L läßt sich auf zwei verschiedene Weisen als Vereinigung zweier Quader definieren; wir nehmen etwa an, der erste habe als linke untere hintere Ecke den Punkt $(0, 0, 0)$ und als rechte obere vordere Ecke den Punkt $(7, 3, 4)$. Wenn wir beide Quader als achsenparallel voraussetzen, ist der erste Quader dadurch eindeutig festgelegt. Beim zweiten Quader sei die linke untere hintere Ecke $(7, 0, 0)$ und die rechte obere vordere sei $(9, 10, 4)$. Die zylindrische Bohrung sei ein

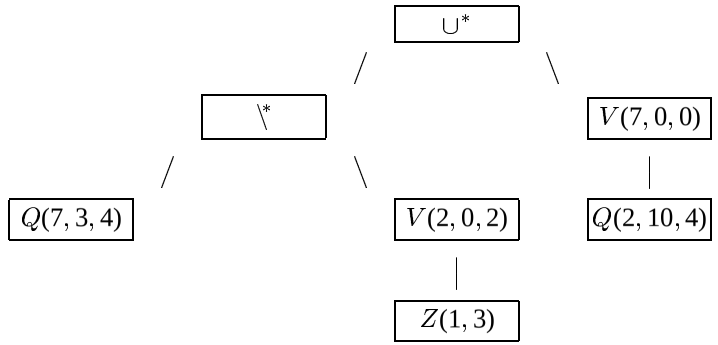
Zylinder, dessen Mittellinie vom Punkt $(2, 3/2, 4)$ zum Punkt $(2, 3/2, 1)$ gehe und der den Radius eins habe.

Der erste Quader liegt dann bereits so, wie wir es oben für primitive Quader vorausgesetzt haben; in der dort eingeführten Terminologie ist er also das primitive Objekt `Quader`$(7, 3, 4)$.

Der zweite Quader hat keine Ecke im Nullpunkt, ist aber immerhin noch achsenparallel. Er kann daher ohne Drehung nur durch Verschiebung aus einem primitiven Quader erzeugt werden, wobei allerdings auch das auf verschiedene Weise geschehen kann. Am einfachsten ist es wohl, ihn durch Verschiebung um den Vektor $(7, 0, 0)$ aus `Quader`$(2, 10, 4)$ zu erzeugen.

Bleibt noch der Zylinder. Er hat Höhe drei und Radius eins, ist also kongruent zu `Zylinder`$(1, 3)$. Dieser Zylinder hat als Mittelachse den Teil der z-Achse zwischen dem Nullpunkt und $(0, 3, 0)$; um daraus den gewünschten Zylinder zu erhalten, können wir uns also wieder mit einer Verschiebung begnügen, diesmal um den Vektor (2, 1,5, 1).

Der CSG-Baum ist auch mit diesen Festlegungen noch nicht eindeutig bestimmt: Dazu müssen wir noch festlegen, ob wir zuerst bohren wollen oder zuerst die beiden Quader zusammensetzen. Falls wir zuerst bohren ergibt sich der Baum



in dem `Quader`(a, b, c) der Einfachheit halber durch $Q(a, b, c)$ abgekürzt ist und `Zylinder`(r, h) durch $Z(r, h)$; entsprechend steht $V(x, y, z)$ für die Verschiebung um den Vektor (x, y, z).

e) Manipulation von CSG-Bäumen

Wie schon das sehr einfache Beispiel aus dem letzten Abschnitt zeigt, ist der CSG-Baum eines geometrischen Objekts alles andere als eindeutig. Dies hat den Vorteil, daß geometrische Operationen und regularisierte BOOLEsche Operationen für Objekte, die durch CSG-Bäume definiert sind, sehr einfach ausführbar sind: Man erzeugt einfach eine neue Baumwurzel und hängt das oder die Argumente der Operation an.

Es kann im Einzelfall zu sehr ineffizienten Darstellungen führen, etwa wenn dasselbe Objekt mehrfach als Blatt vorkommt oder wenn ein Objekt und eine echte Teilmenge davon Blätter desselben CSG-Baums sind. Mit graphentheoretischen Algorithmen läßt sich dieses Problem teilweise abmildern, jedoch möchte ich darauf nicht näher eingehen.

Zumindest anhand einiger weniger Beispiele sollten wir uns aber doch einen Eindruck vom Umgang mit CSG-Bäumen verschaffen. Eine wichtige Aufgabe in der geometrischen Modellierung ist beispielsweise die Klassifikation eines Punkts bezüglich eines CSG-Objekts

Damit ist folgendes gemeint: Gegeben sei ein Punkt $P = (x, y, z) \in \mathbb{R}^3$. Für die Lage dieses Punkts bezüglich eines durch einen CSG-Baum definierten Objekts sind für uns drei Fälle interessant: P kann innerer Punkt sein, er kann auf dem Rand liegen oder außerhalb. Die Entscheidung darüber, welcher dieser drei Fälle vorliegt, bezeichnet man als *Klassifikation* des Punkts bezüglich des CSG-Objekts.

Mit unseren bisherigen Kenntnissen können wir nicht unbedingt unterscheiden, welcher der drei Fälle vorliegt: Zumindest bei einigen der Beispiele für primitive geoometrische Objekte ist es alles andere als trivial, die Lage eines Punkts bezüglich dieses Objekts zu klassifizieren. Für andere wie etwa die algebraische Halbraumdarstellung ist das Problem dagegen trivial: Für

$$M = \{(x, y, z) \in \mathbb{R}^3 \mid f(x, y, z) \leq 0\}$$

ist P innerer Punkt, falls $f(x, y, z) < 0$ ist und Randpunkt für $f(x, y, z) = 0$; andernfalls ist P äußerer Punkt.

Wir wollen davon ausgehen, daß es zu jedem primitiven Objekt einen Algorithmus gibt, der die Frage entscheidet; falls wir Normteile und

ähnliches als primitive Objekte zulassen, müssen wir uns (bei effizientem Einsatz dieser Dateien und Programme) ohnehin auf das verlassen, was uns der Anbieter zur Verfügung stellt.

Im Augenblick geht es also nur darum, die Klassifikation eines Punkts bezüglich eines CSG-Objekts zurückzuführen auf seine Klassifikation bezüglich der primitiven Objekte, aus denen dieses zusammengesetzt ist.

Dazu müssen wir denn CSG-Baum zunächst von der Wurzel hin zu den Blättern traversieren, um das Klassifikationsproblem auf Klassifikationsprobleme für die primitiven Objekte zurückzuführen. Dies bezeichnen wir, unabhängig davon, wie sich Botaniker einen Baum vorstellen, als Traversieren von oben nach unten.

Bei diesem Abstieg nach unten ignorieren wir zunächst die regularisierten BOOLEschen Operationen und geben an jedem Knoten, der einer solchen Operation entspricht, die Frage einfach an beide (oder alle, falls es mehr als zwei gibt) darunterhängende Bäume weiter.

An den Knoten, die geometrischen Transformationen entsprechen, ist allerdings Handlungsbedarf gegeben: Ein Punkt liegt sicherlich nicht genau dann in einem gedrehten oder verschobenen Körper, wenn er im ursprünglichen Körper liegt. Um vom transformierten Körper zurückzukommen zum ursprünglichen, müssen wir zunächst die Transformation rückgängig machen. Bei jedem solchen Knoten muß also der zu klassifizierende Punkt der *inversen* Transformation unterworfen werden.

Wenn wir auf diese Weise unten angekommen sind, müssen die Klassifikationsprobleme bezüglich der primitiven Objekte in den Blättern gelöst werden; ein Problem, mit dem wir uns, wie gesagt, im Augenblick noch nicht beschäftigen wollen. Tatsächlich wird sich gleich herausstellen, daß wir gelegentlich mehr tun müssen, als den Punkt einfach zu klassifizieren; davon gleich mehr.

Danach müssen wir den Baum wieder von den Blättern aus bis zur Wurzel hochklettern. Die Knoten, die geometrischen Transformationen

entsprechen, können wir dabei ignorieren, da sie bereits beim Abstieg berücksichtigt wurden.

Die Knoten zu den regularisierten BOOLEschen Operationen dagegen haben wir beim Abstieg praktisch ignoriert; wir haben einfach die Frage an die Zweige weitergegeben. Jetzt, beim Aufstieg, müssen deren Antworten zu einer Gesamtantwort kombiniert werden.

Solange es nur darauf ankommt, ob ein Element in einer Menge liegt oder nicht, lassen sich die klassischen mengentheoretischen Operationen Vereinigung, Durchschnitt und Differenz leicht durch logische Operationen auf Aussagen über die Zugehörigkeit zu den Komponenten zurückführen. Bei der Klassifikation, für die wir uns interessieren, haben wir demgegenüber gleich zwei Komplikationen: Erstens haben an Stelle der klassischen Operationen der Mengenlehre die *regularisierten* BOOLEschen Operationen, und zweitens geht es uns nicht nur darum, ob es Punkt in der Menge liegt oder nicht, sondern wir interessieren uns zusätzlich auch noch dafür, ob er innerer Punkt oder Randpunkt ist.

Die klassischen Operationen einer (extensionalen) Logik werden üblicherweise durch Wahrheitstafeln definiert; entsprechend können wir versuchen, Tafeln für die Klassifikation eines Punkts bezüglich eines geometrischen Objekts aufzustellen. Da kompakte Tabellen übersichtlicher sind, seien die drei Möglichkeiten *inneren Punkt*, *Randpunkt* und *äußerer Punkt* abkürzen durch *in*, *auf* und *aus*.

Wenn wir damit eine Tafel für die regularisierte Vereinigung aufstellen, sind acht der neun Einträge trivial: Ein innerer Punkt einer der beiden Mengen ist auch innerer Punkt der regularisierten Vereinigung; damit sind fünf Einträge erledigt. Ebenfalls völlig klar ist, daß ein äußerer Punkt einer der beiden Mengen bezüglich der regularisierten Vereinigung in derselben Weise klassifiziert wird wie bezüglich der anderen Menge; auch dies erledigt fünf Fälle, von denen allerdings zwei bereits mit der vorigen Regel erledigt wurden; insgesamt sind also erst acht Fälle erledigt.

Der noch verbleibende Fall besteht darin, daß wir einen Randpunkt *beider* Mengen haben. Dieser *kann* zum inneren Punkt der Vereinigung werden; ein Beispiel dafür sind zwei Quader, die eine Seitenfläche

gemeinsam haben; jeder innere Punkt dieser gemeinsamen Seitenfläche ist innerer Punkt der (regularisierten) Vereinigung. Die Randpunkte der gemeinsamen Seitenfläche sind allerdings auch Randpunkte der Vereinigung; hier ist das Ergebnis also nicht eindeutig bestimmt durch die Ergebnisse bei den Komponenten.

Damit erhalten wir folgende Verknüpfungstafel:

$\cup^*$	*in*	*auf*	*aus*
in	*in*	*in*	*in*
auf	*in*	**?**	*auf*
aus	*in*	*auf*	*aus*

Das fette Fragezeichen steht für *auf* oder *in*.

Entsprechend ist es beim Durchschnitt: Ein Punkt, der äußerer Punkt einer der beiden Komponenten ist, muß auch äußerer Punkt des Durchschnitts sein; damit sind fünf Fälle erledigt. Entsprechend wird ein innerer Punkt einer der beiden Komponenten bezüglich des Durchschnitts genauso klassifiziert wie bezüglich der anderen Komponente; auch das erledigt wieder fünf Fälle, von denen zwei bereits geklärt sind. Übrig bleibt auch hier wieder der Fall, daß wir einen Randpunkt bezüglich beider Komponenten haben. Im Fall zweier Quader, die nur eine Randfläche (oder Teile von jeweiligen Randflächen) gemeinsam haben, ist der Punkt dann nicht im regularisierten Durchscnitt enthalten, ist also äußerer Punkt. Andererseits gibt es auch den Fall, daß der Punkt als Grenzwert einer Folge innerer Punkte aus dem Durchschnitt beider Komponenten dargestellt werden kann; beispielsweise im Falle zweier Quader, die einen dreidimensionalen Durchschnitt haben, zu dem auch ein Teil des Randes gehört. (Einfachster Fall wären zwei Quader, von denen der eine im anderen liegt und eine Randfläche hat, die ebenfalls in einer Randfläche des anderen enthalten ist.)

Hier ergibt sich also folgende Verknüpfungstafel:

$\cap^*$	*in*	*auf*	*aus*
in	*in*	*auf*	*aus*
auf	*auf*	**?**	*aus*
aus	*aus*	*aus*	*aus*

Hier steht das fette Fragezeichen hier für *auf* oder *aus*.

Genauso sieht es aus bei der regularisierten Differenz: Die Formel

$$A \setminus B = A \cap \left(\mathbb{R}^3 \setminus B\right),$$

mit der sich die gewöhnliche Mengendifferenz auf Durchschnitt und Komplement zurückführen läßt, gilt ganz entsprechend auch als

$$A \setminus^* B = A \cap^* \left(\mathbb{R}^3 \setminus^* B\right)$$

für die regularisierten Operationen. Da beim regularisierten wie beim gewöhnlichen Komplement innere Punkte zu äußeren werden, während Randpunkte Randpunkte bleiben, erhalten wir die Tabelle für die regularisierte Differenz aus der für den regularisierten Durchschnitt, indem wir einfach die erste mit der dritten Spalte vertauschen:

$\setminus^*$	*in*	*auf*	*aus*
in	*aus*	*auf*	*in*
auf	*aus*	**?**	*auf*
aus	*aus*	*aus*	*aus*

Auch hier steht das fette Fragezeichen natürlich für *auf* oder *aus*.

Bei allen drei Operationen reicht also im Falle eines gemeinsamen Randpunkts beider Komponenten die Klassifikation des Punkts in Bezug auf die beiden Komponenten nicht aus, um den Punkt bezüglich des Ergebnisses der regularisierten BOOLEschen Operation zu klassifizieren; hier müssen wir mehr Information über die Lage bezüglich der Komponenten haben.

Diese Mehrinformation verschaffen wir uns anhand der für die meisten *Anwendungen* der Mathematik nur wenig relevante Definition eines inneren *bzw.* äußeren Punkts: Ein Punkt ist bekanntlich inneren Punkt einer

Menge $A \subseteq \mathbb{R}^3$, wenn es irgendeine Kugel um diesen Punkt gibt, die ganz in der Menge liegt; er ist äußerer Punkt, falls es eine entsprechende Kugel gibt, die ganz außerhalb der Menge liegt.

Für einen Randpunkt enthält demnach jede noch so kleine Kugel um diesen Punkt sowohl Punkte, die in der Menge liegen, als auch solche, die außerhalb liegen. Zumindest für die einfacheren Arten von primitiven Objekten, die wir in §1 betrachteten, können wir den Durchschnitt zwischen einer kleinen Kugel und einer primitiven Menge explizit berechnen: Bei einem Quader Beispielsweise ist das für einen *inneren* Punkt einer Randfläche eine Halbkugel; für einen inneren Punkt einer *Kante* ist es eine Viertelkugel und für einen Eckpunkt schließlich eine Achtelkugel.

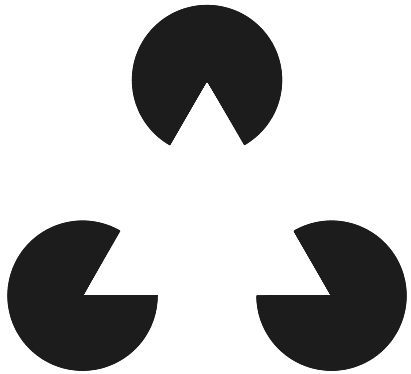
Falls man diese Durchschnitte für die primitiven Objekte bestimmen kann, ist es auch kein Problem, sie beim Aufstieg mit den jeweiligen regularisierten BOOLEschen Operationen zu bearbeiten. Falls dabei nach Ankunft an der Wurzel eine Vollkugel herauskommt, hat man einen inneren Punkt des Objekts; falls die leere Menge herauskommt, handelt es sich um einen äußeren Punkt. In allen anderen Fällen hat man es mit einem Randpunkt zu tun.

§3: Krümmungen und Verzerrungen

a) Krümmung und geometrische Stetigkeit

Wenn wir Flächenstücke zu einer Fläche zusammenfügen, möchten wir dort, wo sie sich berühren, zwar manchmal wirklich einen Knick oder eine Kante sehen, oft aber sollen die Flächenstücke so zusammenpassen, daß der Betrachter nicht sieht, wo das eine in das andere übergeht.

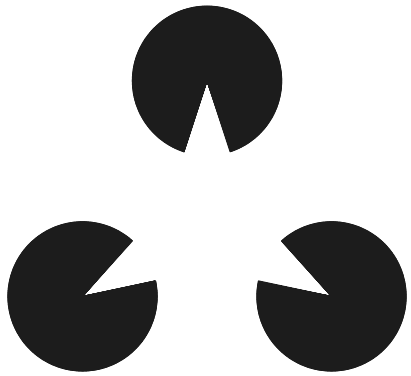
Dazu müssen insbesondere an allen Grenzpunkten die Tangentialebenen übereinstimmen, denn sonst gibt es definitif einen Knick (den man natürlich zur Not auch mit GOURAUD-Shading wegretouchieren kann). Das reicht allerdings noch nicht um unangenehme Effekte zu vermeiden, denn unser Auge reagiert auch sehr sensitiv auf Krümmungen.



Drei Kreissektoren mit Dreieck

Als erstes Beispiel, das mit dem Thema des heutigen Vortrags noch nichts zu tun hat, betrachten wir die hier dargestellten drei Kreissektoren:

Obwohl hier außer den ausgefüllten Kreissektoren nichts gezeichnet ist, glaubt man, in der Mitte ein weißes Dreieck zu sehen, dessen Kanten die Sektorengrenzen verbinden.



Drei Kreissektoren mit „krummem“ Dreieck

In der nächsten Abbildung ist der freie Sektor geschrumpft, so daß es keine geradlininge Verbindung zwischen den Sektorengrenzen mehr gibt. Trotzdem *sehen* wir im Innern ein krummlinig begrenztes Dreieck.

Dieses Dreieck, das uns unsere Augen vorgaukelt (und für das man zumindest bei Affen auch schon die feuernden Neuronen gemessen hat), ist dadurch charakterisiert, daß seine Kanten minimale Gesamtkrümmung haben – ein klarer Hinweis darauf, daß unsere Augen auf Krümmung ansprechen. Auch im computergestützten Konstruieren sind Krümmungen wichtig, den wie sich Zeit, sehen unserer Augen die Unstetigkeitslinien der Krümmung als unangenehme Artefakte, die sogenannten *hard lines*.

Bevor wir über Stetigkeit der Krümmung diskutieren können, müssen wir allerdings zunächst wissen, was Krümmung überhaupt ist.

Beginnen wir mit einem Kurvenstück $\gamma\colon [a,b] \to \mathbb{R}^3$. Seine Länge zwischen dem Anfangspunkt $\gamma(a)$ und dem Zwischenpunkt $\gamma(t)$ ist gleich dem Integral

$$s(t) = \int_a^t |\dot\gamma(t)|\ dt\,,$$

und offensichtlich ist s auf $[a,b]$ eine monoton steigende Funktion.

Sie hat daher eine Umkehrfunktion $\varphi\colon [0,s(b)] \to [a,b]$, zu der wir die Zusammensetzung

$$\delta = \gamma \circ \phi\colon \begin{cases} [0,s(b)] \to \mathbb{R}^3 \\ \qquad s \mapsto \gamma\big(\varphi(s)\big) \end{cases}$$

betrachten können. Diese Abbildung parametrisiert das Kurvenstück durch seine Bogenlänge; man spricht auch von der *natürlichen Parameterdarstellung* des Kurvenstücks.

Da für jedes s gilt

$$s = \int_0^s \left|\dot\delta(s)\right|\ ds\,,$$

ist $\dot\delta(s)$ stets ein Einheitsvektor; seine zweite Ableitung $\ddot\delta(s)$ bezeichnen wir als *Krümmungsvektor*, und den Betrag davon als Krümmung.

Als Beispiel betrachten wir die Kreislinie

$$\gamma\colon \begin{cases} [0,2\pi) \to \mathbb{R}^2 \\ \qquad t \mapsto (r\cos t, r\sin t) \end{cases} \quad \text{mit} \quad \dot\gamma(t) = \begin{pmatrix} -r\sin t \\ r\cos t \end{pmatrix}.$$

Hier hat der Tangentenvektor also überall den Betrag r, d.h. $s(b) = rb$, wie es ja auch der Definition des Bogenmaßes entspricht. Die natürliche Parametrisierung ist damit leicht angebbar als

$$\delta\colon \begin{cases} [0,2\pi r) \to \mathbb{R}^2 \\ \qquad t \mapsto (r\cos\frac{t}{r}, r\sin\frac{t}{r}) \end{cases} \quad \text{mit} \quad \dot\delta(t) = \begin{pmatrix} -\sin t \\ \cos t \end{pmatrix}$$

erwartungsgemäß vom Betrag eins. Die Ableitung

$$\ddot\delta(t) = \frac{1}{r}\begin{pmatrix} -\cos t \\ -\sin t \end{pmatrix}$$

hat überall Betrag $1/r$, d.h. die Krümmung der Kreislinie hat in jedem Punkt diesen Wert.

Bei komplizierteren Kurven läßt sich die natürliche Parametrisierung oft nicht explizit hinschreiben, der Vektor $\dot\delta(s)$ ist aber immer einfach berechenbar als

$$\dot\delta(s) = \frac{1}{|\dot\gamma(t)|}\dot\gamma(t)\,,$$

und daraus läßt sich mit einer weiteren Ableitung auch die Krümmung berechnen.

Auf einer Fläche können wir viele Kurven durch einen gegebenen Punkt legen, die viele verschiedene Krümmungen haben. Für die Krümmung der Fläche selbst sind Krümmungen *innerhalb* der Fläche nicht relevant: Eine Ebene wird nicht krumm, weil es dort Kreise und andere gekrümmte Kurven gibt.

Deshalb zerlegen wir den Krümmungsvektor einer Kurve in zwei Komponenten: Eine Komponente in der Tangentialebene (die sogenannte *geodätische Krümmung*) und eine in Normalenrichtung, die *Normalenkrümmung*. Nur letztere sagt etwas aus über die Krümmung der Fläche selbst.

Abgesehen von speziellen Flächen wie der Kugel werden auch die Normalenkrümmung im betrachteten Punkt im allgemeinen von der Kurve abhängen, sie läßt sich im Falle eines parametrischen Flächenstücks f stets durch die Tangentialrichtung der Kurve und die zweiten partiellen Ableitungen von f ausdrücken, so daß stetige Krümmung äquivalent dazu sind, daß die zweite Ableitung von f auch an Grenzpunkten stetig ist.

Bei impliziten Flächenstücken $f = 0$ folgt dann sofort aus dem Satz über implizite Funktionen, daß auch hier genau dann alle (Normalen-)Krümmungen stetig sind, wenn an den Grenzpunkten die zweiten Ableitungen der definierenden Funktionen übereinstimmen.

Falls dies an allen Grenzpunkten zwischen zwei Flächenstücken erfüllt ist, reden wir von *visueller* oder *geometrischer* Stetigkeit der zusammengesetzten Fläche.

b) Das Theorema egregium

Krümmungen haben auch einen engen Bezug zu *Verzerrungen*, wie sie beispielsweise bei der Projektion in die Ebene auftreten können.

Um letztere zu definieren, betrachten wir wieder zunächst den Kurvenfall und gehen aus von einem Kurvenstück $\gamma\colon [a,b] \to \mathbb{R}^3$, von dem wir annehmen, daß es im abzubildenden Volumen V liegt.

Dieses Kurvenstück projezieren wir, wie alle Objekte in V, über eine Abbildung $\pi\colon V \to \mathbb{R}^2$ in die Ebene; das Bild ist offensichtlich das ebene Kurvenstück

$$\delta = \pi \circ \gamma\colon [a,b] \to \mathbb{R}^2; \quad t \mapsto \pi\big(\gamma(t)\big)\,.$$

Als *Verzerrung* im Punkt $\gamma(t)$ definieren wir das Verhältnis

$$\frac{\big|\dot\delta(t)\big|}{\big|\dot\gamma(t)\big|}$$

der Längen der Tangentenvektoren; anschaulich betrachtet ist dies also ein lokaler Abbildungsmaßstab.

Wenn wir an Stelle des Kurvenstücks ein Flächenstück betrachten, gibt es dort wieder viele Kurven durch jeden Flächenpunkt, und im allgemeinen werden diese Kurven verschieden stark verzerrt werden. Wenn alle diese Verzerrungen gleich sind, sagen wir, die Abbildung sei im betrachteten Punkt der Fläche *konform* oder *lokal winkeltreu*. Wenn sie überall konform ist und darüber hinaus die Verzerrung sogar in allen Punkten gleich ist, reden wir von einer *maßstabstreuen* oder unverzerrten Abbildung.

In vielen Anwendungen, beispielsweise bei der kartographischen Abbildung (eines Teils) der Erdoberfläche, sind solche unverzerrten Abbildungen genau das, was man möchte. Leider konnte aber GAUSS gerade in Zusammenhang mit diesem Beispiel zeigen, daß es solche Abbildungen nicht geben kann.

Ausgangspunkt ist wieder die (Normalen-)Krümmung. In einem gegebenen Flächenpunkt wird diese von der Kurve abhängen, tatsächlich aber nur von der Richtung des Tangentenvektors an die Kurve. Wir haben daher für jeden Winkel eine Krümmung, und da die Kreislinie kompakt ist, wird sowohl das Minimum κ_1 als auch das Maximum κ_2 angenommen. Ihr Produkt heißt *Gaußsche Krümmung*.

Nach dem *Theorema egregium* („erhabenes Theorem“) von GAUSS bleibt dieses Produkt bei allen verzerrungsfreien Abbildungen erhalten.

Unsere Abbildungen gehen in die Ebene; dort liegen alle Krümmungsvektoren ebener Kurven in der Ebene selbst, die Normalenkrümmung und damit die GAUSSsche Krümmung ist also überall null. Somit können nur Flächen mit GAUSSscher Krümmung null unverzerrt in die Ebene abgebildet werden.

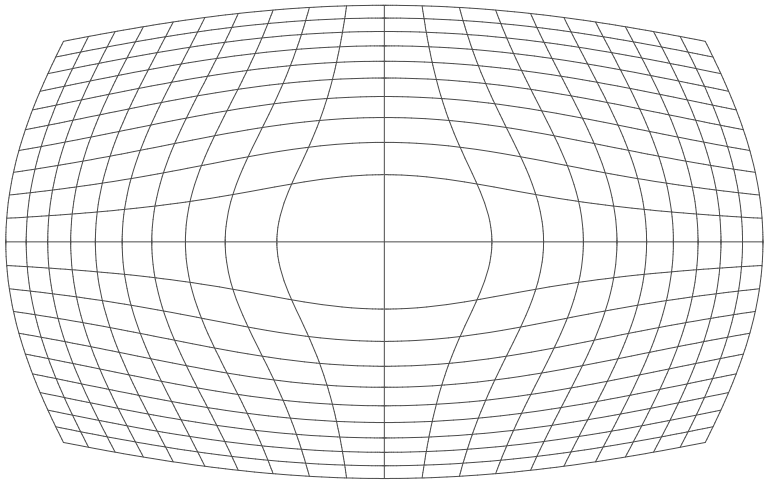
Beispiele solcher Flächen sind etwa der Zylinder, dessen Hauptkrümmungen in Richtung der Mantellinien und der Kreislinien auf dem Mantel gehen, d.h. $\kappa_1 = 0$ und $\kappa_2 = 1/r$, wobei r den Radius bezeichnet. Genauso verhält es sich auch mit dem Kegel, und in der Tat lassen sich beide verzerrungsfrei in die Ebene abrollen.

Bei der Kugel allerdings haben wir überall und in allen Richtungen Normalenkrümmung $1/r$, also GAUSSsche Krümmung $1/r^2$, und deshalb

kann kein noch so kleines Stück der Kugel verzerrungsfrei in die Ebene abgebildet werden; genauso auch bei Ellipsoiden. In der Kartographie gibt es somit keine wirklich maßstabstreuen Abbildungen; man kann nur abgeschwächte Bedingungen wie etwa Winkeltreue *oder* Flächentreue realisieren.

c) Bewußte Verzerrungen

Gelegentlich ist man aber auch in der Kartographie gar nicht interessiert an maßstabtreuen Abbildungen: Bei Stadtplänen beispielsweise wird oft die Innenstadt größer dargestellt als die Randbezirke, und auch bei sonstigen Visualisierungsaufgaben kann es oft zweckmäßig sein, interessante Regionen größer darzustellen als den Rest. Die Abbildung unten zeigt, wie man etwa ein Rechteckgitter stetig und differenzierbar so verzerren kann, daß im Innern ein größerer Maßstab realisiert wird als am Rand. Eines der Maple worksheets zum Vortrag zeigt, wie man diese Verzerrung kontinuierlich erzeugen kann.



Das Rechteckgitter zu einer verzerrten Karte

WOLFGANG K. SEILER
Mathematisches Institut der
Universität Mannheim
Seminargebäude A5, C 201
68131 Mannheim

Tel. 181-2515
seiler@math.uni-mannheim.de
http://hilbert.math.uni-mannheim.de/∼seiler/