# Praktikum 'Anwendungen der Computergraphik'

Hans-Joachim Bungartz

20. Juli 1999

# Inhaltsverzeichnis


**1 Einführung, Grundlagen** **5**

1.1 Historie . . . . . 5

1.2 Gliederung und Überblick . . . . . 6

1.3 Grundlagen . . . . . 7

1.4 Literatur . . . . . 7

**2 Geometrische Modellierung** **9**

2.1 Darstellungsarten dreidimensionaler Körper . . . . . 9

2.2 Freiformkurven und -flächen . . . . . 9

2.2.1 Freiformkurven . . . . . 10

2.2.2 Freiformflächen . . . . . 18

2.2.3 Glatter Übergang an den Nahtstellen . . . . . 20

**3 Animation** **21**

**4 Projektion und Sichtbarkeit** **23**

**5 Beleuchtung und Reflexion** **25**

**6 Shading** **27**

**7 Anmerkungen zu OpenGL** **29**

7.1 Grundlegendes . . . . . 29

7.2 Viewing . . . . . 31

**8 Visualisierung** **35**

8.1 Methoden der Bildverarbeitung . . . . . 36

8.2 Visualisierung raum- und zeitaufgelöster Simulationsdaten . . . . . 38

8.3 Vom Bild zum Film . . . . . 42

8.4 Bunte Bildchen über alles . . . . . 43

**9 Farben und Farbmodelle** **45**

**10 Globale Beleuchtung** **47**

10.1 Einleitung . . . . . 47

10.2 Das Problem der globalen Beleuchtung . . . . . 47

10.2.1 Größen aus der Radiometrie . . . . . 47

10.2.2 BRDFs . . . . . 49

10.2.3 Die Rendering-Gleichung . . . . . 50

10.3 Techniken zur globalen Beleuchtung . . . . . 51

10.3.1 Klassifikation . . . . . 51

10.3.2 Ray Tracing . . . . . 52

10.3.3 Path Tracing . . . . . 53

10.3.4 Radiosity . . . . . 54

10.3.5 Light Ray Tracing . . . . . 56

10.3.6 Monte Carlo Ray Tracing mit Photon Maps . . . . . 56

**11 Raumbildsehen** **59**

# Kapitel 1

# Einführung, Grundlagen

## 1.1 Historie

Warum ein Praktikum über Anwendungen der Computergraphik und keines über Computergraphik? Erstens, weil die Basics der Computergraphik heute in einer Vielzahl von Programmen schon integriert sind, man sich um sie also nicht mehr kümmern muss. Zweitens, weil die heutzutage spannenden ur-graphischen Fragestellungen zumeist sehr ins Detail gehen somit eine intensivere Beschäftigung mit der Materie erfordern, als dies im Rahmen eines Praktikums möglich ist. Drittens, weil die Computergraphik inzwischen nicht mehr nur die Sache einer einer kleinen Gruppe von Spezialisten ist, sondern sehr breit angewandt wird. Damit hat sich auch die Aufgabenstellung für den Graphiknutzer radikal geändert: Man *programmiert* Graphikbibliotheken nicht mehr, man *nutzt* sie, man bastelt sich seine Objekte nicht mehr durch explizite Programmierung, sondern man greift auf komfortable Modellierprogramme zurück.

Dies bedeutet nicht, dass das Leben leichter geworden ist. Die Anforderungen an die maschinennahen Kenntnisse des Anwenders sind heute sicher wesentlich geringer als früher, dafür steht er bei der heute üblichen Graphiksoftware leicht Manualen enzyklopädischen Umfangs gegenüber.

Um diese Entwicklung zu veranschaulichen: Im Graphikpraktikum des Jahres 1988 waren typische Aufgabenstellungen:

- Programmierung von Koordinatentransformationen,
- Zeichnen von Kurven,
- abstrakte Datentypen für Punkte und Transformationen,
- interaktive Definition affiner Transformationen,
- Editor für Punktsequenzen,
- Zeichnen von Bézierkurven,
- interaktiver Farbeditor,
- Berechnung und Darstellung konvexer Hüllen,
- Definition graphischer Grundprimitive in Rand- und Quadtree-Darstellung.

Im Praktikum 1990 wurde dann die Interaktivität durch das Fenstersystem X erleichtert, man konnte jetzt die Programmierung einfacher graphischer Benutzeroberflächen angehen. Dem Nachfolger ging's wieder besser: Jetzt wurde mit Interviews programmiert. Was damit alles möglich ist, zeigt der auf Interviews aufbauende Graphikeditor idraw – heftig benutzt, aber auch nicht mehr das Neueste. Das Ganze hat natürlich nicht nur eine technische Dimension, sondern auch eine finanzielle: Selbst wenn gewisse Dinge zu einer Zeit schon möglich waren, so waren die erforderliche Hardware oder die betreffende Software deswegen noch lange nicht erschwinglich.

Heute ist das anders: Hardware-z-Buffer oder Hardware-Texture-Mapping sind nicht mehr Großrechnern vorbehalten, auch oder gerade Graphikkarten für PCs leisten Erstaunliches, getrieben von Spiele- und Multimedia-besessenen Nutzerscharen. Auch die Software steht da nicht zurück: Eine Floating-Lizenz des state-of-the-art Modellier-, Animations- und Renderingprogramms Maya kostet die Hochschule – nach Abzug des (allerdings nicht unerheblichen) Rabatts für Forschung und Lehre – knapp 3000 DM (März 1999).

Dieser Entwicklung Rechnung tragend, soll in der vorliegenden, völlig neu gestalteten Auflage des Graphikpraktikums vor allem der effiziente, kreative und das angebotene Potential auch ausschöpfende Umgang mit moderner Graphiksoftware erlernt werden.

## 1.2 Gliederung und Überblick

Im Praktikum werden vier große Themenbereiche angesprochen: Die *geometrische Modellierung (geometric modelling)*, die *graphische Darstellung (rendering)*, die *Graphikprogrammierung* und die *(wissenschaftliche) Visualisierung*:

- Die geometrische Modellierung befasst sich mit der Modellierung von darzustellenden Objekten und Szenen, d. h. mit deren Nachbildung bzw. Approximation im Rechner. Sie ist damit die Grundlage graphischer Applikationen.
- Aufgabe der graphischen Darstellung ist es, aus einer gegebenen (dreidimensionalen) Szene ein möglichst realistisches (zweidimensionales) Bild zu erzeugen. Zentrales Themen sind dabei Perspektive und Beleuchtung.
- Während geometrische Modellierung und graphische Darstellung heute in der Regel mittels komfortabler Programmpakete durchgeführt werden, setzt die Graphikprogrammierung eine Ebene tiefer an. Hier wird jeder Schritt mit Hilfe von Bibliotheksfunktionen ausprogrammiert. Diese Vorgehensweise ist mächtiger (weil nicht vom Angebot eines speziellen Softwarepakets abhängig) und effizienter (da die Bibliotheksfunktionen sehr hardwarenah realisiert sind), sie erfordert aber ein tieferes Verständnis der Zusammenhänge, als es für das Bedienen einer komfortablen Benutzeroberfläche erforderlich ist. Insbesondere für Echtzeit-Anwendungen ist die durch Graphikprogrammierung mögliche Effizienzsteigerung unerlässlich.
- Die Visualisierung wissenschaftlicher Datensätze ist ein wichtiges Anwendungsfeld von Methoden der Computergraphik, bei der gemessene (CT, Mikroskopie, Satelliten) oder am Rechner simulierte Daten zur besseren Interpretierbarkeit visuell aufbereitet werden.

Jedem dieser vier Bereiche sind eine einführende Aufgabe, die von allen Teilnehmern in ca. zwei Wochen zu bearbeiten ist, und ein weiterführendes Projekt, für das vier Wochen Zeit zur Verfügung stehen (nur eines zu bearbeiten), gewidmet.

Die obigen Themenkreise spiegeln sich auch in der jeweils einzusetzenden Software wider:

- Das Programmpaket Maya von Alias Wavefront ist ein mächtiges Werkzeug für die Aufgaben 3 D Modellierung, Animation und Rendering.
- Die Graphikbibliothek OpenGL wurde aus der Graphics Library GL entwickelt, die auf SGI-Systemen verfügbar und im Lieferumfang von IRIX enthalten war, und stellt heute einen auf allen gängigen Plattformen erhältlichen Standard in Sachen Graphikprogrammierung dar. Auch Maya setzt auf OpenGL auf.

- Das Visualisierungssystem AVS bzw. AVS/Express bietet eine Vielzahl von Möglichkeiten zur Visualisierung zwei- und dreidimensionaler Datensätze, inklusive Techniken der Bildverarbeitung und des Volume Rendering.

Die begleitende zweistündige Vorlesung zum Praktikum 'Anwendungen der Computergraphik' setzt sich zusammen aus einer einstündigen Vorlesung zu den erforderlichen theoretischen Grundlagen der Computergraphik und aus einer einstündigen Einführung in die zu bearbeitenden Aufgaben sowie in die hierfür einzusetzenden Programmpakete. Das vorliegende Skript ist eine Zusammenfassung des theoretischen Stoffs. Zum Teil wird auf die entsprechenden Abschnitte in *Bungartz, Griebel, Zenger: Einführung in die Computergraphik, Vieweg 1996* verwiesen, an manchen Stellen wird ergänzendes Material bereitgestellt.

Folgende Themen werden in der Vorlesung behandelt:

1. Grundlagen
   Arbeitsschritte, Koordinatensysteme, Transformationen, Primitive etc.

2a. Geometrische Modellierung I
   Darstellungsarten dreidimensionaler Körper

2b. Geometrische Modellierung II
   Freiformkurven und -flächen

3. Animation

4. Projektion und Sichtbarkeit
   Parallel- und Zentralprojektion, Clipping, Rückseitenentfernung, $z$-Buffer, Ray Casting

5. Beleuchtung und Reflexion
   ambientes Licht, diffuse Reflexion, spiegelnde Reflexion

6. Shading
   konstant, Gouraud-Shading, Phong-Shading

7. Grundlegendes zu OpenGL

8. Visualisierung

9. Farben und Farbmodelle

10a. Globale Beleuchtung I
   physikalische Grundlagen, BRDFs, Rendering-Gleichung

10b. Globale Beleuchtung II
   Ray Tracing, Path Tracing, Radiosity, Light Ray Tracing, Photon Tracing

11. Raumbildsehen

## 1.3 Grundlagen

Siehe im Buch die Abschnitte 1.1 bis 1.5.

## 1.4 Literatur

Abgesehen vom bereits erwähnten Graphikbuch sei auf folgende Literatur verwiesen:

- J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes:
  Computer Graphics: Principles and Practice
  Addison-Wesley, Reading, 1990
  *(die Bibel)*
- G. Farin:
  Kurven und Flächen im Computer Aided Geometric Design
  Vieweg, Braunschweig, 1994
  *(die Referenz für Freiformkurven und -flächen)*

# Kapitel 2

# Geometrische Modellierung

## 2.1 Darstellungsarten dreidimensionaler Körper

Siehe im Buch die Abschnitte 2.1 und 2.2.

## 2.2 Freiformkurven und -flächen

Für viele Anwendungen sind die indirekten Darstellungsschemata, die Starrkörper nicht über ihr Volumen, sondern über ihre Oberfläche beschreiben, von herausragender Bedeutung. Man denke etwa an den Bereich des CAM (*Computer Aided Manufacturing*) mit den dort verbreiteten computergesteuerten Fräs- und Stanzmaschinen. Auch für den mit dem CAD (*Computer Aided Design*) befassten Ingenieur ist das Arbeiten mit bzw. das Manipulieren von Flächen intuitiver als der Umweg über Volumenmodelle. Der einfachste Zugang einer *Triangulierung* des Randes von Starrkörpern, also der Erzeugung eines Netzes aus Dreiecken, deren Eckpunkte auf der Oberfläche liegen, ist dabei in aller Regel nicht genau genug. Man braucht mehr als Streckensegmente und planare Flächenstücke. Deshalb wurde in der Geburtsstunde des CAGD (*Computer Aided Geometric Design*) die Verwendung parametrisierter Kurven und Flächen angeregt. Als bahnbrechende Entwicklungen sind dabei einerseits *Bézierkurven* und *-flächen* sowie andererseits der *Coons Patch* bzw. die *transfinite Interpolation* hervorzuheben. Bezeichnenderweise stammen alle diese in den sechziger Jahren erfolgten Entwicklungen aus dem Automobilbereich: Bézierkurven und -flächen wurden unabhängig von P. de Casteljau bei Citroën und P. Bézier bei Renault entwickelt[1]. S. Coons beriet Ford, und der Urheber der transfiniten Interpolation, W. Gordon, war für General Motors tätig.

Trotz der erheblich gestiegenen Beschreibungsmöglichkeiten mittels parametrisierter Kurven und Flächen, die die Modellierung sog. *Freiformkurven und -flächen* gestatten, ist das der Triangulierung zugrunde liegende Prinzip der Partitionierung der Oberfläche bzw. ihrer Zerlegung in einzelne *Patches* nach wie vor unverzichtbarer Bestandteil der Oberflächenmodellierung komplizierter Objekte wie beispielsweise eines Autos. Die Oberfläche des Objekts wird *topologisch* in Eckpunkte, Kanten und Flächen partitioniert. Wie diese konkret *geometrisch* aussehen, wird zunächst nicht festgelegt. Somit ergeben sich folgende zentrale Probleme bzw. Aufgaben:

1. Wie kann die (topologische) Oberfläche von Objekten, die sich aus einzelnen Flächenstücken zusammensetzt, effizient beschrieben werden?

[1] De Casteljaus Arbeiten waren etwas früher, wurden aber aus Geheimhaltungsgründen nie veröffentlicht.

2. Wie kann sichergestellt werden, dass eine solche topologische Beschreibung auch tatsächlich einer Oberfläche eines Starrkörpers entspricht und nicht etwa eine Kleinsche Flasche liefert?
3. Wie können die einzelnen topologischen Kanten oder Flächen mit effizient zu berechnenden, leicht zu manipulierenden und zu ästhetisch zufriedenstellenden Ergebnissen führenden Kurven und Flächen attributiert werden?
4. Wie werden die resultierenden (geometrischen) Patches zum Gesamtmodell der Oberfläche verklebt?

Die Fragen 1 und 2 sind eher von theoretischem Interesse und sollen deshalb an dieser Stelle nicht weiter verfolgt werden (siehe dazu im Buch den Abschnitt 2.3). Die Fragen 3 und 4 schauen wir uns aber näher an.

Die einfachste Form der geometrischen Attributierung haben wir oben bereits erwähnt: Man reduziere die polygonalen topologischen Patches durch wiederholte Unterteilung auf drei- oder viereckige und ordne den topologischen Punkten geometrische Punkte (Punkte im Raum) und den topologischen Kanten geometrische Kanten (Strecken) zu. Dreieckigen Patches wird dann die planare Dreiecksfläche zugeordnet (Triangulierung), bei viereckigen Patches wird bilinear interpoliert (siehe Abbildung 2.1). Diese Vorgehensweise liefert aber noch keine zufriedenstellenden Ergebnisse (zu einfache Patch-Geometrie, Knicke an den Patch-Grenzen). Deshalb diskutieren wir kurz, wie

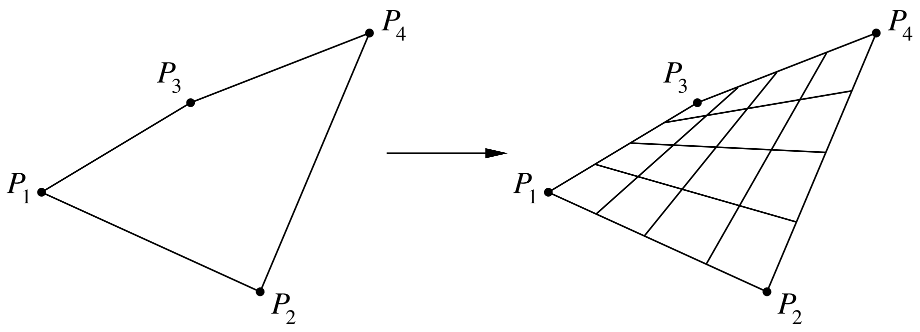


Abbildung 2.1: Bilineare Interpolation bei viereckigen Flächenstücken

es besser gemacht werden kann, zunächst bei Kurven, dann bei Flächen. Dabei sollte man immer im Hinterkopf behalten, dass es um ganz unterschiedliche Aufgabenstellungen gehen kann: Bei der *Interpolationsaufgabe* sind vorgegebene Punkte geeignet zu verbinden, bei der *Approximationsaufgabe* dagegen ist eine vorgegebene Kurve bzw. Fläche möglichst gut anzunähern, wobei es keine gemeinsamen Punkte von Original und Näherung geben muss. Interpolation und Approximation sind Probleme der *Rekonstruktion*, im CAGD hingegen ist oftmals der *Entwurf* von Kurven und Flächen gefragt (ohne konkretes Vorbild). Gedanklich starten wir jedoch zunächst von der Rekonstruktionsaufgabe aus.

### 2.2.1 Freiformkurven

Gegeben sei eine Raumkurve, entweder analytisch als parametrisierte Kurve oder in Form diskreter (Mess-) Punkte $(t_i, \mathbf{x}_i(t_i))$, $i = 0, \ldots, n$, wobei $\mathbf{x}_i(t_i) := (x_i(t_i), y_i(t_i), z_i(t_i))$ einen Raumpunkt bezeichnet. Herkömmliche Polynominterpolation im Lagrangeschen Sinne scheidet in der Graphik fast immer aufgrund ihrer aus der Numerik bekannten schlechten Eigenschaften bei wachsendem Polynomgrad aus. So ist das Interpolationsproblem für große Werte $n$ im Allgemeinen schlecht konditioniert, und kleine (lokale) Änderungen in den Stützwerten $\mathbf{x}_i, i = 0, \ldots, n$, können große (globale) Änderungen im Kurvenverlauf nach sich ziehen. Zudem treten oftmals starke Oszillationen auf. Die exakte Interpolation der Werte ist dagegen gar nicht so wichtig, gegen eine gute Approximation (bei festgelegtem Start- und Zielpunkt freilich) ist nichts einzuwenden. Worauf muss also in der Computergraphik und im CAGD Wert gelegt werden? Gefordert werden *Kontrollierbarkeit* (die (interaktive) Manipulation des Kurvenverlaufs durch die dem Benutzer zur Verfügung gestellten Kontrollparameter soll intuitiv und vorhersagbar sein) und *Lokalität* (lokale Veränderungen der Eingabedaten dürfen allenfalls kleine globale Auswirkungen haben, besser ist ein begrenzter Einflussbereich).

Um diesen beiden Zielen der Kontrollierbarkeit und der Lokalität möglichst nahezukommen, sind eine Reihe von Ansätzen zur Freiformkurvenmodellierung entwickelt worden. Die drei wohl wichtigsten Konzepte, *Bézierkurven*, *B-Splines* und *NURBS* sollen im Folgenden kurz angesprochen werden. Für Details sei auf das eingangs erwähnte Standardwerk von Farin verwiesen.

Bézierkurven:

Zur Definition von Bézierkurven benötigen wir zunächst den Begriff der *Bernsteinpolynome*. Bernsteinpolynome vom Grad $n$ sind definiert als

$$B_i^n(t) \;:=\; \binom{n}{i} \cdot (1-t)^{n-i} \cdot t^i\,, \quad t \in [0,1]\,, \quad i = 0, \ldots, n\,. \tag{2.1}$$

Wie man leicht zeigen kann, gilt $B_i^n(t) \in [0,1]$ für alle $n \in \mathbb{N},\ 0 \le i \le n,\ t \in [0,1]$. Ferner hat

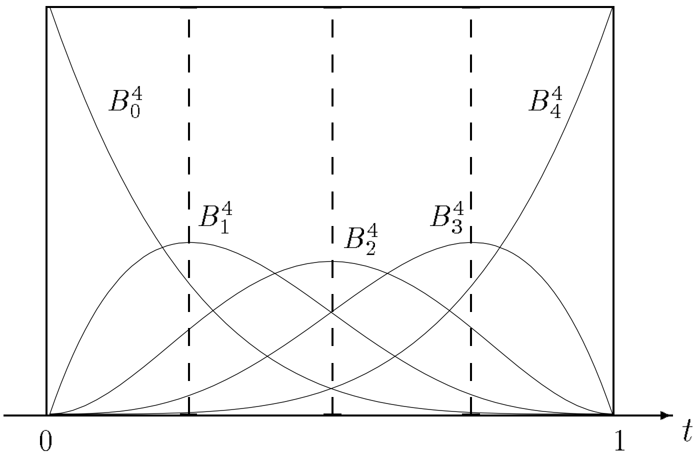


Abbildung 2.2: Bernsteinpolynome vom Grad 4

$B_i^n(t)$ eine $i$-fache Nullstelle in $t = 0$ sowie eine $(n-i)$-fache Nullstelle in $t = 1$. Außerdem gilt

$$\sum_{i=0}^{n} B_i^n(t) \;=\; 1 \qquad \forall t \in [0,1]\,, \tag{2.2}$$

da

$$\sum_{i=0}^{n} B_i^n(t) \;=\; \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \;=\; \Big((1-t)+t\Big)^n \;=\; 1\,. \tag{2.3}$$

Die Kurve

$$\mathbf{x}(t) \;:=\; \sum_{i=0}^{n} \mathbf{b}_i \cdot B_i^n(t) \quad \text{mit} \quad t \in [0,1] \quad \text{und} \quad \mathbf{b}_i \in \mathbb{R}^3 \tag{2.4}$$

heißt *Bézierkurve* vom Grad $n$ zu den *Bézierpunkten* oder *Kontrollpunkten* $\mathbf{b}_0, \ldots, \mathbf{b}_n$. Letztere spannen mit ihrer konvexen Hülle das *Kontrollpolygon* der Kurve auf.

$\mathbf{x}(t)$ besitzt eine Reihe wichtiger Eigenschaften:

- Wegen (2.2) und (2.4) sind die Werte $B_i^n(t)$ *baryzentrische Koordinaten*[2] von $\mathbf{x}(t)$ zum Kontrollpolygon. Deshalb sind Bézierkurven invariant unter affinen Abbildungen (Verschiebungen, Drehungen, Skalierungen, Scherungen).
- Aufgrund von (2.3) verläuft die Bézierkurve im Kontrollpolygon.
- Die Endpunkte werden interpoliert, also $\mathbf{x}(0) = \mathbf{b}_0$ und $\mathbf{x}(1) = \mathbf{b}_n$, die anderen Kontrollpunkte jedoch i. a. nicht.
- Bézierkurven sind symmetrisch in den Kontrollpunkten (für die Anordnungen $\mathbf{b}_0, \ldots, \mathbf{b}_n$ und $\mathbf{b}_n, \ldots, \mathbf{b}_0$ ergibt sich dieselbe Kurve).
- Für zwei Zahlen $\alpha$ und $\beta$ mit $\alpha + \beta = 1$ und zwei Sätze $\mathbf{b}_j$ bzw. $\mathbf{c}_j$ von Kontrollpunkten gilt $(0 \leq j \leq n)$: Die Kurve zu den Kontrollpunkten $\alpha\mathbf{b}_j + \beta\mathbf{c}_j$ stimmt mit dem entsprechend gewichteten Mittel der beiden Einzelkurven überein.
- Die Tangente in $t = 0$ ist $n \cdot (\mathbf{b}_1 - \mathbf{b}_0)$, die in $t = 1$ ist $n \cdot (\mathbf{b}_n - \mathbf{b}_{n-1})$.
- Liegen alle Bézierpunkte auf einer Geraden, so wird die Bézierkurve zu einer Strecke.
- Parabelbögen lassen sich exakt erzeugen, Kreisbögen sind gut approximierbar.
- Bézierkurven sind *formerhaltend* (*shape-preserving*). Ein nicht negativer (monotoner, konvexer) Datensatz führt zu einer nicht negativen (monotonen, konvexen) Kurve:

$$\begin{aligned} \text{alle } \mathbf{b}_i \geq 0 &\implies \mathbf{x}(t) \geq 0\,, \\ \text{alle } \mathbf{b}_{i+1} - \mathbf{b}_i \geq 0 &\implies \dot{\mathbf{x}}(t) \geq 0\,, \\ \text{alle } \mathbf{b}_{i+1} - 2\mathbf{b}_i + \mathbf{b}_{i-1} \geq 0 &\implies \ddot{\mathbf{x}}(t) \geq 0\,. \end{aligned} \tag{2.5}$$

- Doppelpunkte ($\mathbf{x}(t_1) = \mathbf{x}(t_2)$, obwohl $t_1 \neq t_2$) können ohne weitere Einschränkungen an die Kontrollpunkte nicht ausgeschlossen werden.
- Lokale Änderungen in den Daten (Kontrollpunkten) wirken sich zwar global aus, ihr Einfluss ist jedoch nur lokal von Bedeutung:

$$\mathbf{b}_i \mapsto \mathbf{b}_i + \mathbf{e}_i \implies \mathbf{x}(t) \mapsto \mathbf{x}(t) + \mathbf{e}_i \cdot \varepsilon_i(t) \tag{2.6}$$

  mit

$$\varepsilon_i(t) := \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i} \in [0,1]\,. \tag{2.7}$$

  Dabei ist die Änderung $\mathbf{e}_i \cdot \varepsilon_i(t)$ nur nahe bei $\mathbf{b}_i$ signifikant (siehe Abb. 2.2). Bzgl. unserer beiden Ziele *Lokalität* und *Kontrollierbarkeit* ist die Situation also akzeptabel, aber noch nicht optimal.

Von großer Bedeutung ist die effiziente Auswertung von $\mathbf{x}(t)$ für einen Parameterwert $t$, also die Bestimmung eines Kurvenpunkts. Hierzu gibt es verschiedene Techniken:

- rekursiv mittels

$$\begin{aligned} B_i^{r+1}(t) &:= (1-t) \cdot B_i^r(t) + t \cdot B_{i-1}^r(t)\,, \\ B_i^0(t) &:= \begin{cases} 1 & : \ i = 0\,, \\ 0 & : \ \text{sonst}, \end{cases} \end{aligned}$$

[2] Ein baryzentrisches Koordinatensystem im $n$-dimensionalen affinen Raum $\mathcal{A}_n$ besteht aus $n+1$ Punkten $P_0, \ldots, P_n$, die nicht alle gemeinsam in einer Hyperebene des $\mathcal{A}_n$ liegen. Baryzentrische Koordinaten eines Punktes $P$ sind dann Gewichte $\omega_0, \ldots, \omega_n$ mit $P = \sum_{i=0}^n \omega_i \cdot P_i$. Oft werden die Gewichte $\omega_i$ eingeschränkt auf $0 \leq \omega_i \leq 1$ und $\sum_{i=0}^n \omega_i = 1$. Dann sind alle Punkte in der konvexen Hülle der $P_i$ eindeutig darstellbar.

- oder durch fortgesetzte lineare Interpolation mit dem *Algorithmus von de Casteljau* (der eigentlichen Initialzündung der Bézierkurven):

$$\begin{array}{l}
\mathbf{i = 0, \ldots, n:} \\
\qquad\qquad\qquad \mathbf{b}_i^0 := \mathbf{b}_i; \\
\mathbf{k = 1, \ldots, n:} \\
\qquad \mathbf{i = k, \ldots, n:} \\
\qquad\qquad\qquad \mathbf{b}_i^k := (1-t) \cdot \mathbf{b}_{i-1}^{k-1} + t \cdot \mathbf{b}_i^{k-1};
\end{array}$$

Man erhält schließlich $\mathbf{x}(t) = \mathbf{b}_n^n$. Hierbei ist zu beachten, dass der Parameter $t$ nicht auf $[0,1]$ begrenzt ist, sondern eine beliebige reelle Zahl sein kann. Zur Veranschaulichung zeigt Abbildung 2.3 das Schema, das dem Algorithmus von de Casteljau zugrunde liegt (vgl. hiermit das Schema von Aitken und Neville zur effizienten Auswertung von Polynom-Interpolanten).

$$\begin{tikzcd}
\mathbf{b}_0 = \mathbf{b}_0^0 \arrow[dr] \\
\mathbf{b}_1 = \mathbf{b}_1^0 \arrow[r] \arrow[dr] & \mathbf{b}_1^1 \arrow[dr] \\
\mathbf{b}_2 = \mathbf{b}_2^0 \arrow[r] \arrow[dr] & \mathbf{b}_2^1 \arrow[r] \arrow[dr] & \mathbf{b}_2^2 \ddots \\
\vdots & \vdots & \vdots & \\
\mathbf{b}_{n-1} = \mathbf{b}_{n-1}^0 \arrow[r] \arrow[dr] & \mathbf{b}_{n-1}^1 \arrow[r] \arrow[dr] & \mathbf{b}_{n-1}^2 \cdots & \mathbf{b}_{n-1}^{n-1} \arrow[dr] \\
\mathbf{b}_n = \mathbf{b}_n^0 \arrow[r] & \mathbf{b}_n^1 \arrow[r] & \mathbf{b}_n^2 \cdots & \mathbf{b}_n^{n-1} \arrow[r] & \mathbf{b}_n^n = \mathbf{x}(t)
\end{tikzcd}$$

Abbildung 2.3: Berechnung von $\mathbf{x}(t)$ nach de Casteljau

In Abb. 2.4 ist für den Fall $n = 3$ ein Schema zur geometrischen Konstruktion von $\mathbf{b}_3^3 = \mathbf{x}(t)$ angegeben, das die Bedeutung der $\mathbf{b}_i^k$ im Algorithmus von de Casteljau verdeutlicht.

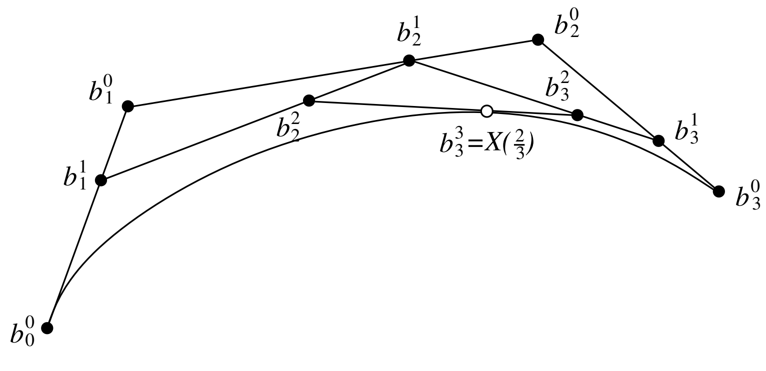


Abbildung 2.4: Geometrische Konstruktion nach de Casteljau für $n = 3$ und $t = 2/3$

Der Aufwand für die Auswertung liegt in beiden Fällen insgesamt bei $O(n^2)$, da $n + (n-1) + \ldots + 1 = n(n+1)/2$ Berechnungsschritte nötig sind, wobei bei der rekursiven Berechnung der Bernsteinpolynome zusätzlich noch die Summation zu $\mathbf{x}(t)$ gemäß (2.4) auszuführen ist.

B-Splines:

Bézierkurven sind jedoch nicht der Weisheit letzter Schluss. Lokalität und Kontrollierbarkeit sind verbesserungsfähig. Wenn ferner die zu modellierende Kurve eine sehr komplizierte Gestalt hat, dann führt die große Zahl der erforderlichen Kontrollpunkte zu einem sehr hohen Polynomgrad $n$. Spätestens ab $n = 10$ ist man damit in der Praxis aus dem Rennen. Deshalb geht man über zu *zusammengesetzten* oder *stückweisen* Bézierkurven $\mathbf{s}(u)$. Dieses Vorgehen ist ganz analog zur *Polynominterpolation* in der Numerik, wo man die einfache Lagrange-Interpolation auch zugunsten etwa stückweise kubischer Splines aufgibt. Im Gegensatz zur Interpolationsaufgabe „kleben" wir jetzt allerdings lokal definierte Bézierkurven $\mathbf{s}_i(t)$ mit $t = (u - u_i)/(u_{i+1} - u_i)$ aneinander. Als Funktion von $u$ lebt $\mathbf{s}_i(t(u))$ somit auf $[u_i, u_{i+1}]$. Die Nahtstellen $u_i$ werden dabei *Knoten* genannt, die zugehörigen Kontrollpunkte $\mathbf{s}(u_i)$ heißen *Knotenpunkte*, die restlichen Kontrollpunkte *innere Bézierpunkte*. Durch die Knotenpunkte geht die Spline aufgrund der Eigenschaften von Bézierkurven durch. Wie wir später sehen werden, ist die Lage der Kontrollpunkte Design-Kriterien unterworfen (z. B. Glattheitsbedingungen an den Nahtstellen), sie sind *nicht* mehr das Steuerinstrument. Vielmehr werden diese Rolle zusätzliche Punkte $\mathbf{d}_i$ übernehmen.

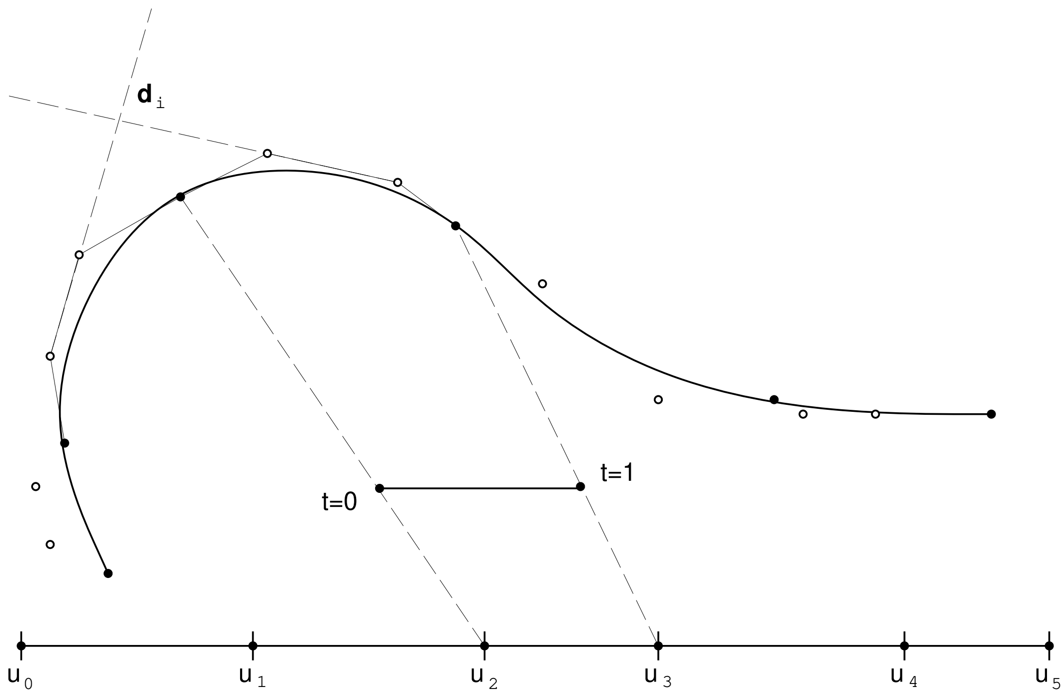


Abbildung 2.5: Béziersplines: Knotenpunkte (schwarz) und innere Bézierpukte (weiß)

Der glatte Übergang an den Knoten $u_i$ ist natürlich der zentrale Punkt bei den Splines – schließlich soll trotz des reduzierten lokalen Polynomgrads ein Höchstmaß an globaler Glattheit ermöglicht werden können. Nehmen wir an, zwei Bézierkurven mit den Kontrollpunkten $\mathbf{b}_0, \ldots, \mathbf{b}_n$ bzw. $\mathbf{b}_n, \ldots, \mathbf{b}_{2n}$, die über $[u_0, u_1]$ bzw. $[u_1, u_2]$ parametrisiert seien, sollen in $u_1$ glatt verklebt werden. Man kann zeigen, dass die „globale" Kurve auf $[u_0, u_2]$ in der Nahtstelle $u_1$ genau dann $r$-mal stetig differenzierbar ist, wenn

$$\mathbf{b}_{n+i} \;=\; \mathbf{b}^i_{n-i}(t)\,, \qquad i = 0, \ldots, r\,, \tag{2.8}$$

gilt mit $t = (u_2 - u_0)/(u_1 - u_0)$. Die $\mathbf{b}^i_{n-i}(t)$ sind dabei die Zwischengrößen aus dem de-Casteljau-Schema (siehe Abb. 2.3).

Man kann sich auch die Ableitungen an der Nahtstelle von links und rechts direkt anschauen. Will man etwa $\mathcal{C}^1$-stetigen Übergang an den Knoten $u_i$ haben, so ist

$$\left.\frac{\mathrm{d}\mathbf{s}(u)}{\mathrm{d}u}\right|_{u\uparrow u_i} = \left.\frac{\mathrm{d}\mathbf{s}(u)}{\mathrm{d}u}\right|_{u\downarrow u_i}, \tag{2.9}$$

also

$$\frac{1}{u_i - u_{i-1}} \cdot \frac{\mathrm{d}\mathbf{s}_{i-1}(t)}{\mathrm{d}t} = \frac{1}{u_{i+1} - u_i} \cdot \frac{\mathrm{d}\mathbf{s}_i(t)}{\mathrm{d}t} \tag{2.10}$$

sicherzustellen. Aufgrund der oben erwähnten Eigenschaften von Bézierkurven (Tangenten in den Endpunkten) erfordert dies die Kollinearität der zugehörigen drei Kontrollpunkte sowie ein bestimmtes Abstandsverhältnis der Knoten $u_i$ (also eine geeignete globale Parametrisierung der Kurve $\mathbf{s}(u)$). Will man bei dem in Abb. 2.5 angedeuteten Fall *kubischer Splines* $\mathcal{C}^2$-Stetigkeit haben, so erfordern die resultierenden Glattheitsbedingungen die eindeutige Existenz zusätzlicher Hilfspunkte $\mathbf{d}_i$, die im Schnittpunkt der Geraden durch $\mathbf{b}_{3i-2}$ und $\mathbf{b}_{3i-1}$ mit der Geraden durch $\mathbf{b}_{3i+2}$ und $\mathbf{b}_{3i+1}$ liegen, wobei jeweils gleiche Längenverhältnisse gelten müssen (siehe Abb. 2.6).

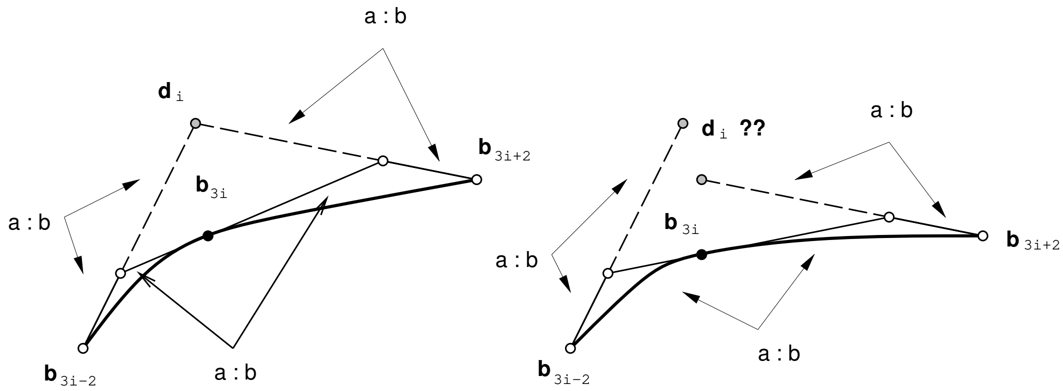


Abbildung 2.6: Erfüllte (links) und nicht erfüllte (rechts) $\mathcal{C}^2$-Bedingung

Bei allgemeinem Grad $n$ schließlich ist offensichtlich $\mathcal{C}^{n-1}$-Stetigkeit die maximal mögliche Glattheit.

Soll die Bézier-Spline die maximale Glattheit $\mathcal{C}^{n-1}$ haben, ist sie durch die Wahl der Knoten $u_i$ und durch die Punkte $\mathbf{d}_i$ des sog. *B-Spline-Polygons* eindeutig bestimmt. Für $n = 2$ sind die $\mathbf{d}_i$ gerade die Randkontrollpunkte $\mathbf{b}_0$ und $\mathbf{b}_n$ sowie die inneren Bézierpunkte (d.h. die $\mathbf{b}_{2i-1}$), im kubischen Fall ($n = 3$) sind auf jeder Seite die beiden äußersten Kontrollpunkte $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_{n-1}$ und $\mathbf{b}_n$ sowie die oben eingeführten Hilfspunkte $\mathbf{d}_i$ die Punkte des B-Spline-Polygons. Aus den Knoten und dem B-Spline-Polygon ergeben sich die jeweiligen Kontrollpunkte $\mathbf{b}_i$ und ihre zugehörigen Parameterabszissen dann alleine aus den Glattheitsbedingungen. Deshalb werden solche Bézier-Splines auch B-Splines genannt. Man startet jetzt also von den Knoten und dem B-Spline-Polygon und konstruiert sich dazu die B-Spline inklusive der Kontrollpunkte.

Die B-Splines erben viele Eigenschaften von den Bézierkurven:

- Die B-Spline liegt in der konvexen Hülle der Punkte des B-Spline-Polygons; schärfer: sie liegt in der Vereinigung der konvexen Hüllen der lokalen Kontrollpolygone.
- B-Splines sind invariant unter affinen Abbildungen.
- Es gilt die Symmetrie in den Punkten des B-Spline-Polygons.
- Wieder haben wir Endpunkt-Interpolation.

Woher kommt das „B“ in den Bezeichnungen B-Spline und B-Spline-Polygon? Das B rührt daher, dass für B-Splines eine *Basis*darstellung aus *Minimum-Support-Splines* $N_i^n(u)$ als Basisfunktionen existiert,

$$\mathbf{s}(u) := \sum_{i=0}^{N+n-1} \mathbf{d}_i \cdot N_i^n(u)\,, \tag{2.11}$$

wobei die $\mathbf{d}_i$ die Punkte des B-Spline-Polygons sind. Im Gegensatz zu den nur lokal definierten aneinandergeklebten Beźierkurven sind die einzelnen Minimum-Support-Splines $N_i^n(u)$ *global* definiert, d. h. über dem ganzen Parameterbereich von $u$, und zwar so, dass $N_i^n(u_i) = 1$ gilt, dass $\mathcal{C}^{n-1}$-Glattheit sichergestellt ist und dass der *Träger* (*support*) von $N_i^n(u)$ (der $u$-Bereich, über dem $N_i^n(u)$ nicht verschwindet) eine minimale Anzahl von Knoten umfasst. Im Falle linearer B-Splines ($n = 1$) mit globaler Stetigkeit ($\mathcal{C}^0$) sind die $N_i^n(u)$ also die bekannten Hutfunktionen (siehe Abb. 2.7). Diese Konstruktion ist ganz analog zur aus der Numerik (vielleicht) bekannten

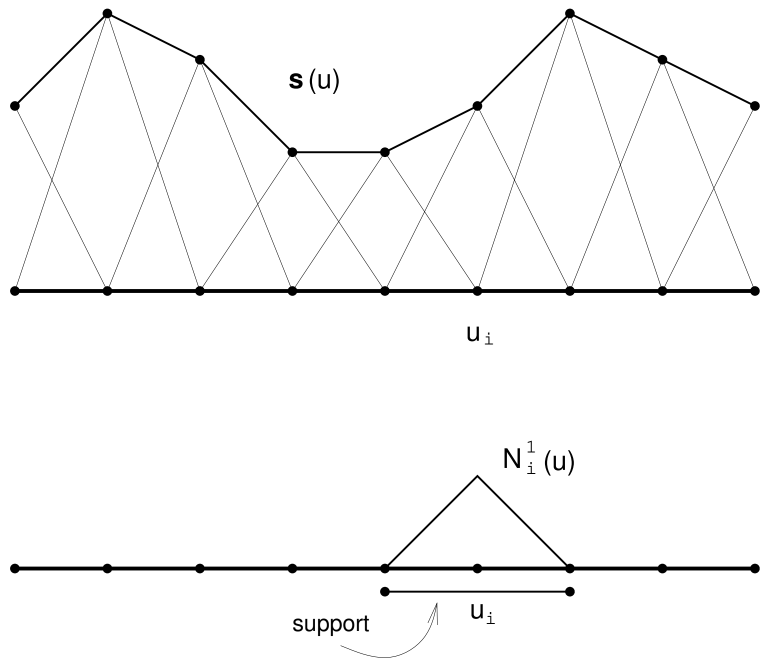


Abbildung 2.7: Lineare B-Spline mit zugehöriger Basisfunktion $N_i^1(u)$

Spline-Interpolation. Dort werden Splines zunächst als stückweise Polynomkurven lokal definiert und verklebt, bevor dann mit Hilfe von Minimum-Support-Splines eine globale Basisdarstellung angegeben wird.

Der entscheidende Vorteil des Spline-Ansatzes ist die Lokalität: Jetzt sind lokale Änderungen in einem Kontrollpunkt wirklich lokal begrenzt (auf die lokale Bézierkurve und – glattheitsabhängig – einen Teil der beiden benachbarten Bézierkurven). Bzgl. Lokalität und Kontrollierbarkeit sind wir jetzt zufrieden.

NURBS:

Ein Problem bleibt: Die bisher studierten Polynomkurven sind zwar invariant unter affinen Abbildungen, nicht jedoch unter *projektiven*. Solche projektiven Abbildungen oder kurz Projektionen sind aber in der Computergraphik wichtig: Schließlich ist i. A. eine dreidimensionale Szene mittels einer perspektivischen Projektion auf eine Ebene abzubilden. Wie man vielleicht von *Kegelschnitten* (Projektionen von im Raum gegebenen Parabeln auf eine Ebene) weiß, entstehen bei der Projektion im Raum gegebener polynomialer Kurven auf eine Ebene *rationale* Kurven in der Ebene (Polynomausdrücke in Zähler und Nenner). Unsere zum Modellieren verwendeten Polynomkurven ergeben also auf dem Bildschirm rationale Kurven. Dann kann man auch gleich die größere Mächtigkeit des rationalen Ansatzes ausnutzen und mit *rationalen Bézierkurven* oder *rationalen Splinekurven* im Raum modellieren – die sind nämlich abgeschlossen unter projektiven Abbildungen. Rationale B-Splines werden auch *NURBS* genannt (*non uniform rational B-Splines*) und sind heute Standard bei der Kurven- und Flächenmodellierung.

Zunächst zu den dreidimensionalen rationalen Bézierkurven vom Grad $n$, die wir uns als Projektion von rationalen Bézierkurven vom Grad $n$ im $\mathbb{R}^4$ mit den Punkten $(x, y, z, w)$ auf die Hyperebene $w = 1$ vorstellen. Man kann zeigen, dass eine solche rationale Bézierkurve durch

$$\mathbf{x}(t) \ := \ \frac{\sum_{i=0}^{n} w_i \cdot \mathbf{b}_i \cdot B_i^n(t)}{w_i \cdot B_i^n(t)}\,, \qquad \mathbf{x}(t), \mathbf{b}_i \in \mathbb{R}^3\,, \tag{2.12}$$

mit *Gewichten* $w_i$ gegeben ist. Wie im polynomialen Fall bilden die $\mathbf{b}_i$ das Kontrollpolygon, das jetzt gerade die Projektion des vierdimensionalen Kontrollpolygons der Punkte $(w_i \mathbf{b}_i^T, w_i)$ des polynomialen Urbilds von $\mathbf{x}(t)$ ist. Falls alle Gewichte gleich eins sind, so liefert (2.12) gerade die gewöhnliche polynomiale Bézierkurve. Üblicherweise lässt man nur nichtnegative Gewichte zu, um Singularitäten zu vermeiden. Rationale Bézierkurven besitzen alle wesentlichen Eigenschaften ihrer polynomialen Pendants (affine Invarianz etc.). Die Gewichte $w_i$ werden üblicherweise als *Formparameter* verwendet. Ein vergrößertes Gewicht $w_i$ erhöht den Einfluss des Kontrollpunkts $\mathbf{b}_i$. Der Effekt einer Veränderung des Gewichts unterscheidet sich deutlich vom Effekt des Verschiebens des Kontrollpunkts. Die Gewichte bringen also zusätzliche Modellierfreiheit ins Spiel (siehe Abb. 2.8).

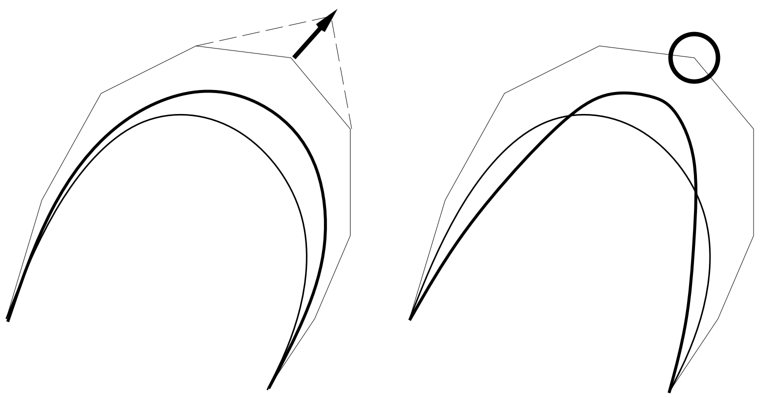

Abbildung 2.8: Einfluss der Kontrollpunktverschiebung (links) vs. Einfluss der Gewichtserhöhung (qualitativ)

Ganz analog sind dreidimensionale rationale B-Splines $n$-ten Grades Projektionen vierdimensionaler rationaler B-Splines auf die Hyperebene $w = 1$. Sie lassen sich somit darstellen als

$$\mathbf{s}(u) \ := \ \frac{\sum_{i=0}^{N+n-1} w_i \cdot \mathbf{d}_i \cdot N_i^n(u)}{\sum_{i=0}^{N+n-1} w_i \cdot N_i^n(u)}\,, \tag{2.13}$$

wobei die $N_i^n(u)$ wie zuvor die (polynomialen) Basis-Splines und die $\mathbf{d}_i$ die Punkte des B-Spline-Polygons bezeichnen. Wiederum sind die $\mathbf{d}_i$ die Projektionen der Punkte des entsprechenden vierdimensionalen B-Spline-Polygons.

Das Modellieren mit rationalen Splines unterscheidet sich nicht besonders vom Modellieren mit ihren polynomialen Pendants. Die Gewichte bringen zusätzliche Flexibilität, die Auswirkung ihrer Modifikation ist lokal begrenzt.

### 2.2.2 Freiformflächen

Ausgehend von den verschiedenen Ansätzen zur Freiformkurvenmodellierung im vorigen Abschnitt kann man sich nun an die Modellierung von Freiformflächen machen. Die naheliegendste Vorgehensweise ist dabei der *Tensorproduktansatz*. Den linearen Fall hat Abb. 2.1 mit der *bilinearen Interpolation* gezeigt. Die so definierte Fläche $\mathbf{x}(s,t)$, $s,t \in [0,1]$ – gewissermaßen die einfachste Möglichkeit der Definition einer Fläche zwischen vier Punkten – kann interpretiert werden als die überstrichene Fläche, die durch Bewegung der beiden Punkte $P_1$ und $P_2$ auf geradem Weg nach $P_3$ bzw. $P_4$ unter Mitführung der jeweiligen geraden Verbindung der Endpunkte entsteht:

$$\mathbf{x}(s,t) \;:=\; (1-s)\cdot(1-t)\cdot P_1 \;+\; s\cdot(1-t)\cdot P_2 \;+\; (1-s)\cdot t\cdot P_3 \;+\; s\cdot t\cdot P_4\,, \qquad (2.14)$$

wobei für die Parameter $s,t \in [0,1]$ gelte. Allgemein, d. h. unter Verzicht auf die Linearität, gelangt man zu der intuitiven Definition einer Tensorproduktfläche als dem geometrischen Ort einer Kurve, die sich durch den Raum bewegt und dabei ihre Gestalt ändert.

Doch eines nach dem anderen. Wir starten vom bilinearen Patch in Abb. 2.1, lassen jetzt aber allgemeine Kurven $\mathbf{x}(s,0) := \gamma_0(s)$ von $P_1$ nach $P_2$ (also $\gamma_0(0) = P_1$, $\gamma_0(1) = P_2$) und $\mathbf{x}(s,1) := \gamma_1(s)$ von $P_3$ nach $P_4$ (also $\gamma_1(0) = P_3$, $\gamma_1(1) = P_4$) zu, $s \in [0,1]$. Anschließend lassen wir wieder $P_1$ und $P_2$ auf Geraden nach $P_3$ bzw. $P_4$ wandern und erhalten eine sog. *Regelfläche*:

$$\mathbf{x}(s,t) \;:=\; (1-t)\cdot\mathbf{x}(s,0) \;+\; t\cdot\mathbf{x}(s,1)\,, \qquad (2.15)$$

wobei wiederum $s,t \in [0,1]$ gelte. In $t$-Richtung wird also nach wie vor linear interpoliert – allerdings nicht mehr diskrete Punkte, sondern ganze Kurven (siehe Abb. 2.9). Dieser Vorgang wird auch *transfinite Interpolation* genannt.

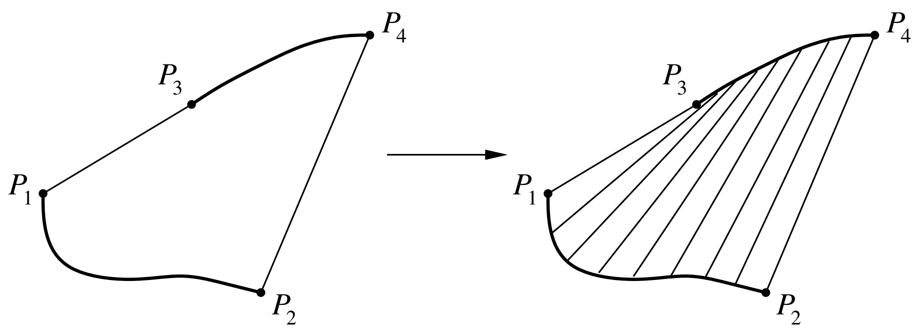


Abbildung 2.9: Regelfläche (transfinite Interpolation in einer Richtung)

Noch einen Schritt weiter geht der *Coons Patch*. Jetzt können alle vier Randkurven $\gamma_0(s)$ von $P_1$ nach $P_2$, $\gamma_1(s)$ von $P_3$ nach $P_4$, $\delta_0(t)$ von $P_1$ nach $P_3$ sowie $\delta_1(t)$ von $P_2$ nach $P_4$ beliebig vorgegeben werden:

$$\begin{aligned} \mathbf{x}(s,0) &:= \gamma_0(s)\,, & \mathbf{x}(s,1) &:= \gamma_1(s)\,, \\ \mathbf{x}(0,t) &:= \delta_0(t)\,, & \mathbf{x}(1,t) &:= \delta_1(t)\,, \end{aligned} \qquad (2.16)$$

wieder mit $s,t \in [0,1]$. Darauf aufbauend konstruieren wir die zwei Regelflächen $\mathbf{x}_s(s,t)$ und $\mathbf{x}_t(s,t)$:

$$\begin{aligned} \mathbf{x}_s(s,t) &:= (1-t)\cdot\mathbf{x}(s,0) \;+\; t\cdot\mathbf{x}(s,1)\,, \\ \mathbf{x}_t(s,t) &:= (1-s)\cdot\mathbf{x}(0,t) \;+\; s\cdot\mathbf{x}(1,t)\,, \end{aligned} \qquad (2.17)$$

$s, t \in [0,1]$. Offensichtlich interpoliert $\mathbf{x}_s$ die $\gamma$-Kurven, kann aber die $\delta$-Kurven nicht reproduzieren. Für $\mathbf{x}_t$ gilt das Umgekehrte. Jede der beiden Regelflächen verhält sich also an zwei Seiten korrekt, an zwei Seiten jedoch falsch, genauer linear. Die Summe $\mathbf{x}_s + \mathbf{x}_t$ enthält demnach – zumindest am Rand – genau die linearen Anteile zuviel. Subtraktion des bilinearen Interpolanten $\mathbf{x}_{st}$, definiert wie in (2.14), liefert demzufolge am Rand das gewünschte Ergebnis. Als *Coons Patch* wird deshalb definiert

$$\mathbf{x}(s,t) \;:=\; \mathbf{x}_s(s,t) \;+\; \mathbf{x}_t(s,t) \;-\; \mathbf{x}_{st}(s,t)\,, \tag{2.18}$$

$s, t \in [0,1]$. Auch für (2.18) ist der Name *transfinite Interpolation* gebräuchlich.

Die nächste Verallgemeinerung liegt auf der Hand: Anstelle der linearen Faktoren $1-s, s, 1-t, t$ in (2.14) – (2.17) können auch allgemeinere Interpolationsfunktionen $f_0(s)$ und $f_1(s)$ bzw. $g_0(t)$ und $g_1(t)$ verwendet werden, sofern sie

$$\begin{aligned} f_0(s) + f_1(s) &= 1 \quad \forall s \in [0,1]\,, \\ g_0(t) + g_1(t) &= 1 \quad \forall t \in [0,1]\,, \\ f_0(0) = f_1(1) &= 1\,, \\ g_0(0) = g_1(1) &= 1 \end{aligned} \tag{2.19}$$

erfüllen. Oder man lässt gleich Bernsteinpolynome zu und gelangt somit zu viereckigen oder Tensorprodukt-*Bézierflächen* vom Grad $(m,n)$:

$$\mathbf{x}(s,t) \;:=\; \sum_{i=0}^{n}\sum_{k=0}^{m} \mathbf{b}_{ik} \cdot B_i^n(s) \cdot B_k^m(t)\,, \qquad s,t \in [0,1]\,. \tag{2.20}$$

Dabei sind die $\mathbf{b}_{ik} \in \mathbb{R}^3$ die *Bézierpunkte* der Fläche, sie spannen (analog zum Kontrollpolygon bei Bézierkurven) jetzt ein *Kontrollnetz* für die Fläche auf (siehe Abb. 2.10).

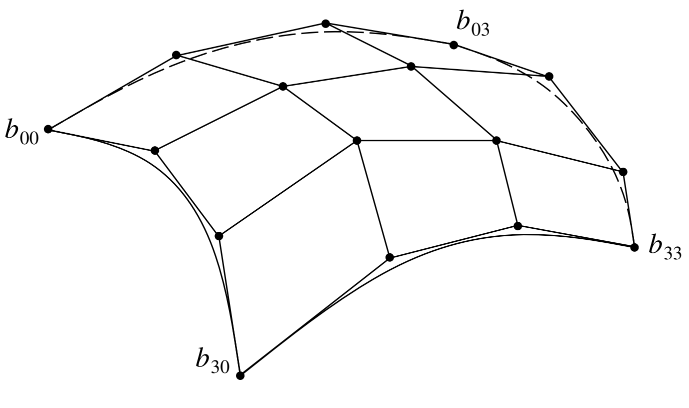


Abbildung 2.10: Bikubische Bézierfläche mit zugehörigem Kontrollnetz

An Eigenschaften halten wir fest:

- Die vier Eckpunkte $\mathbf{b}_{00}$, $\mathbf{b}_{0m}$, $\mathbf{b}_{n0}$ und $\mathbf{b}_{nm}$ liegen auf der Fläche, die anderen Punkte im Allgemeinen nicht.
- Die Kurven mit konstantem $t_2$ sind Bézierkurven bezüglich $t_1$ mit den Bézierpunkten

  $$b_i(t_2) \;:=\; \sum_{k=0}^{m} \mathbf{b}_{ik} B_k^m(t_2)\,.$$

  Für die Kurven mit konstantem $t_1$ gilt das Entsprechende. Insbesondere sind also die vier Randkurven der Bézierfläche Bézierkurven, die durch die Kontrollpunkte $\mathbf{b}_{0k}$ bzw. $\mathbf{b}_{i0}$ bzw. $\mathbf{b}_{nk}$ bzw. $\mathbf{b}_{im}$ $(0 \le k \le m, 0 \le i \le n)$ beschrieben werden.

- Die Bézierfläche liegt in der konvexen Hülle ihres Kontrollnetzes.

Für die effiziente Berechnung von Punkten $\mathbf{x}(s, t)$ auf Bézierflächen sowie für dreieckige Bézierflächen sei auf Abschnitt 2.4.5 im Buch verwiesen.

### 2.2.3 Glatter Übergang an den Nahtstellen

Spline-Flächen (auf B-Spline- oder NURBS-Basis) erlauben die Definition größerer Flächen auch über Patch-Grenzen hinweg, da sie mit festem Grad arbeiten und Glattheit an den (inneren) Patch-Grenzen per definitionem garantieren. Anders ist dies bei Bézierflächen bzw. bei Spline-Flächen an den (äußeren) Patch-Grenzen. Hier sind explizite Vorkehrungen zur Erzwingung von Glattheit (Vermeidung von Knicken bspw.) zu treffen (siehe den Abschnitt 2.4.6 im Buch).

# Kapitel 3

# Animation

Siehe im Buch den Abschnitt 4.7.

Eine kurze Anmerkung zum Begriff der *Kinematik*. Im Zusammenhang mit Animation versteht man hierunter die Spezifikation bzw. Steuerung der Bewegung (eines Skeletts, z. B.). Man unterscheidet hierbei zwischen *Vorwärts-Kinematik* einerseits und *Rückwärts-* bzw. *inverser Kinematik* andererseits. Zur Veranschaulichung betrachte man ein Modell eines Roboterarms oder das „Skelett" eines menschlichen Arms, bestehend aus Stäben und Gelenken. Start und Zielpunkt des Greifarms bzw. der Hand werden nun für eine konkrete Bewegung vorgegeben.

Bei der Vorwärtskinematik wird die erforderliche Bewegung jedes Einzelteils explizit angegeben, beginnend mit der „Wurzel" des Modells (im Sinne einer hierarchischen Modellierung). Offensichtlich sind damit sehr präzise Steuerungen möglich, das Ganze kann aber auch sehr aufwendig werden (insb. bei komplizierten Modellen). Zudem kommt die Intuition etwas kurz: Wer kann schon auf Anhieb sagen, wie alle Gelenke des Arms zu bewegen sind, um die Hand an eine vorgegebene Stelle zu führen.

Intuitiver ist die zielgetriebene inverse Kinematik. Hier wird nur der gewünschte Positionswechsel des Greifarms bzw. der Hand vorgegeben, die hierzu erforderlichen Bewegungen der Einzelglieder werden automatisch bestimmt und durchgeführt. Voraussetzung hierfür ist natürlich, dass alle von einer Bewegung betroffenen Glieder schnell identifiziert werden können (in Maya mittels *IK Handles*) und dass die erforderlichen Einzelbewegungen dann auch schnell berechnet werden können (in Maya mittels *IK Solvers*).

## Kapitel 4

# Projektion und Sichtbarkeit

Siehe im Buch die Abschnitte 3.1 und 3.2.

# Kapitel 5

# Beleuchtung und Reflexion

Siehe im Buch den Abschnitt 3.3.

# Kapitel 6

# Shading

Siehe im Buch den Abschnitt 3.4.

# Kapitel 7

# Anmerkungen zu OpenGL

## 7.1 Grundlegendes

Die für die meisten gängigen höheren Programmiersprachen zur Verfügung stehenden *Graphikbibliotheken* enthalten Prozeduren, die es ermöglichen, graphische Grundfunktionen in Programmen zu benutzen. Dies erstreckt sich von einfachen Routinen für graphische Primitive bis hin zu Funktionen, die bei der Erstellung von Animationen benötigt werden. Die verschiedenen Prozeduren sind meist sehr hardwarenah realisiert, was zu einer hohen, für Echtzeitfähigkeit unerlässlichen Effizienz führt. Das Programmieren in einer höheren Sprache ermöglicht größtmögliche Flexibilität, jedoch ist ein grundsätzliches Verständnis von graphischer Programmierung notwendig.

Als Beispiele für Graphikbibliotheken seien genannt:

- Starbase (für HP-Systeme, im Lieferumfang von HP-UX enthalten, programmierbar in C oder FORTRAN; Ray Tracing, Radiosity)
- GL (für SGI-Systeme, früher im Lieferumfang von IRIX enthalten, programmierbar in C oder FORTRAN, inzwischen ersetzt durch OpenGL)
- OpenGL (aus GL entwickelter und 1992 eingeführter Standard, im Lieferumfang von IRIX enthalten, verfügbar für nahezu alle gängigen Plattformen

Von einer bestimmten Programmiersprache unabhängig ist die frei verfügbare Algorithmensammlung *Graphics Gems*, die sowohl die grundlegenden Algorithmen als auch sehr problemspezifische Funktionen enthält. Diese müssen allerdings erst an die jeweils verwendete Programmiersprache angepasst werden.

Doch nun zu OpenGL, das im Praktikum als Grundlage zur (interaktiven) dreidimensionalen Graphikprogrammierung dienen soll. OpenGL hat sich als Standard für graphische APIs (Application Programming Interfaces) weitgehend durchgesetzt. Es ist also zunächst eine Schnittstelle zwischen Graphik-Software und graphikfähiger Hardware. Im Kern besteht OpenGL aus rund 120 verschiedenen Kommandos, die alle völlig hardware-unabhängig sind und nur rudimentäre Funktionen erbringen. Befehle, die das Fenstersystem, Benutzereingabe oder die Modellierung kompliziert geformter Objekte betreffen, sind zusätzlich bereitzustellen.

Damit das nicht alles dem Anwender überlassen bleibt, gibt es eine Reihe assoziierter Bibliotheken:

- Die *OpenGL Utility Library* (GLU) enthält eine Reihe von Routinen, die auf der Grundlage der einfachen OpenGL-Programme komplexere Funktionen bereitstellen. Die GLU ist Bestandteil einer OpenGL-Implementierung.

- Die *OpenGL Extension to the X Window System* (GLX) stellt die Verbindung der OpenGL-Basisroutinen mit dem X Fenstersystem her. Die GLX ist ebenfalls Bestandteil einer OpenGL-Implementierung.
- *Open Inventor* ist eine objektorientierte Programmsammlung, die Objekte und Methoden für aufwendigere interaktive 3 D Graphikapplikationen bereitstellt. Open Inventor wird von SGI vertrieben.

Mit den Erweiterungsbibliotheken umfasst das Anwendungsspektrum von OpenGL 3 D Modellierung und CAD, echtzeitfähige 3 D Animation, Rendering, Texture Mapping, Spezialeffekte, Visualisierungsfunktionen und vieles mehr. Zahlreiche Anwendungsprogramme bauen auf OpenGL auf, so etwa Maya, die CAD/CAM Systeme CATIA von Dassault und I-DEAS von SDRC oder das Visualisierungssystem AVS/Express (siehe das folgende Kapitel). Die Protagonisten rühmen folgende Vorteile:

- breite Industrieunterstützung, offen und herstellerunabhängig
- beständig und aufwärtskompatibel,
- Portierbarkeit der Anwendungen,
- verfügbar auf der ganzen Palette der Architekturen, vom Laptop zum Supercomputer (bspw. UNIX, Windows 95, Windows NT, Mac OS, Linux, OS/2 etc.),
- benutzerfreundlich (was sich hoffentlich erweisen wird!),
- verwendbar aus einer Vielzahl von Sprachen (bspw. Fortran, C, C++, Java etc.),
- dokumentiert.

Eine Auswahl der Dinge, die OpenGL kann:

- *accumulation buffer*: mehrfach berechnete Frames können zur Erzielung bestimmter Effekte (verschwommenes Bild, Tiefenschärfe, Nachziehen bei Bewegung (*motion blur*)) überlagert bzw. kombiniert werden;
- *alpha blending*: Möglichkeit zur Realisierung stufenloser Transparenz (der Parameter $\alpha \in [0, 1]$ gibt die Lichtdurchläsigkeit einer Fläche bzw. eines Objekts an; realisiert als sog. RGBA-Wert: rot, grün, blau und $\alpha$);
- *anti-aliasing*: lokaler Glättungsoperator;
- *double buffering*: zur Verhinderung des Flackerns, das bei Animationssequenzen durch das ständige Löschen, Neuberechnen und Laden des Bildschirminhalts entsteht, wird ein zweiter Frame-Buffer eingesetzt, so dass aufeinanderfolgende Bilder in unterschiedlichen Bildschirmspeichern bereitgestellt werden können;
- *Gouraud shading*: lineare Interpolation der Licht- bzw. Farb-Intensitätswerte in den Polygonecken auf das ganze Polygon zur Erzielung glatter Übergänge an den Polygonkanten;
- *Materialeigenschaften*: diese können in die Berechnung der Licht- bzw. Farb-Intensitätswerte auf Oberflächen einfließen;
- *Pixel Operationen*: speichern, abbilden, zoomen;
- *texture mapping*: Projektion einer Textur auf die Oberfläche eines Objekts der Szene;
- *Transformationen*: Möglichkeit der expliziten Durchführung von Transformationen (Spiegelungen, Drehungen, Streckungen, Ansichtswechsel) durch Angabe der Transformationsmatrizen;
- *z-buffering*: zum Sichtbarkeitsentscheid;
- *3 D texturing*: zum Hardware-beschleunigten Volume Rendering;

- *Glanzeffekte*: auch nach der Texturierung anbringbar;
- Kontrolle des *Level-of-detail*: Texturen liegen in verschiedenen Auflösungen vor (*mipmap textures*); die verwendete Auflösung richtet sich nach der Entfernung des zu texturierenden Objekts vom Betrachter.

## 7.2 Viewing

Unter *Viewing* versteht man die Aufgabe, (dreidimensionale) Modelle im Raum (in der *Welt*) zu einer Szene anzuordnen, diese von einer bestimmten Position aus zu betrachten und aus dem Resultat ein zweidimensionales Bild auf dem Ausgabegerät zu erzeugen[1]. Da dies in OpenGL alles explizit zu programmieren ist, wollen wir hierauf etwas näher eingehen. Folgendes ist also zu tun:

1. Modelle von Objekten im Raum anordnen und ausrichten mit Hilfe geeigneter Transformationen (*modelling transformations*);
2. den Betrachterstandpunkt im Raum wählen und ausrichten mit Hilfe ähnlicher Transformationen (*viewing transformations*; defaultmäßig steht der Betrachter im Ursprung und blickt in Richtung der negativen z-Achse);
3. den abzubildenden Ausschnitt der Welt (*viewing volume*) festlegen mit Hilfe von sechs *Clipping*-Ebenen (vorne, hinten, links, rechts, oben, unten) und auf 2 D projizieren (*projection transformations*);
4. Abbildung des Resultats auf den vorgesehenen Bereich im Ausgabegerät (*viewport transformation*).

Man kann sich die einzelnen Schritte ganz einfach anhand des Beispiels des Fotographierens veranschaulichen: Die aufzunehmenden Personen und Objekte werden geeignet angeordnet (1.), die Kamera wird positioniert und auf einem Stativ justiert (2.), der Bildausschnitt wird durch Zooming bzw. durch Auswahl des Objektivs festgelegt (3.), und zuletzt wird entschieden, in welchem Format das Foto ausgedruckt werden soll (4.).

Jede Transformation bildet dabei ein gegebenes Koordinatensystem auf ein neues ab. Arbeitet man mit *homogenen Koordinaten* ($(x, y, z, w)$ mit $w = 1$ in 3 D und zusätzlich $z = 0$ in 2 D[2]), so lassen sich sowohl *affine Abbildungen* (Verschiebung, Streckung, Drehung, Spiegelung, Scherung) als auch *Zentral-* und *Parallelprojektion* als Multiplikation des jeweiligen Koordinatenvektors mit einer $4 \times 4$-Matrix darstellen. Ein Objektpunkt als vierdimensionaler Vektor ist also mehreren Matrix-Vektor-Multiplikationen unterworfen, bevor er sich als Pixel auf dem Bildschirm wiederfindet. Wie das in OpenGL aussieht, ist in Abbildung 7.1 illustriert.

Die Modelview Matrix in Abbildung 7.1 ist das Produkt mehrerer Matrizen zur Durchführung aller Viewing und Modelling Transformations. Diese einheitliche Betrachtung ist auch sinnvoll, da beide eng miteinander verwoben sind: Die Kamera von der Szene zu enfernen im Sinne einer Viewing Transformation ist offensichtlich gleichbedeutend mit dem Entfernen der Szene von der Kamera im Sinne einer Modelling Transformation. OpenGL wendet dabei die Matrix einer weiteren Transformation immer automatisch auf die bislang durch Produktbildung akkumulierte an bzw. multipliziert sie mit dieser. Wenn ein neues Objekt transformiert werden soll (ein zweites Rad wird z. B. an ein Auto montiert), dann muss zunächst immer die Identität explizit geladen werden, um den Speicher zu löschen. Da die Matrix für eine Folgetransformation immer von rechts auf die aktuelle Modelview Matrix multipliziert wird, wird de facto die zuletzt aufgeschriebene Transformation zuerst auf den Koordinatenvektor angewendet:

$$M_4\,, \quad M_3\,, \quad M_2\,, \quad M_1 \qquad \text{entspricht} \qquad M_4 \cdot M_3 \cdot M_2 \cdot M_1 \cdot x\,. \tag{7.1}$$

[1] Dies ist die OpenGL-Sehweise der Dinge: Viewing im engeren Sinne geht bereits von einer fertig modellierten Szene aus, die dann abgebildet werden soll.

[2] Die z-Koordinate wird mitgenommen, um zu jeder Zeit Tiefeninformation zum Sichtbarkeitsentscheid zu haben.

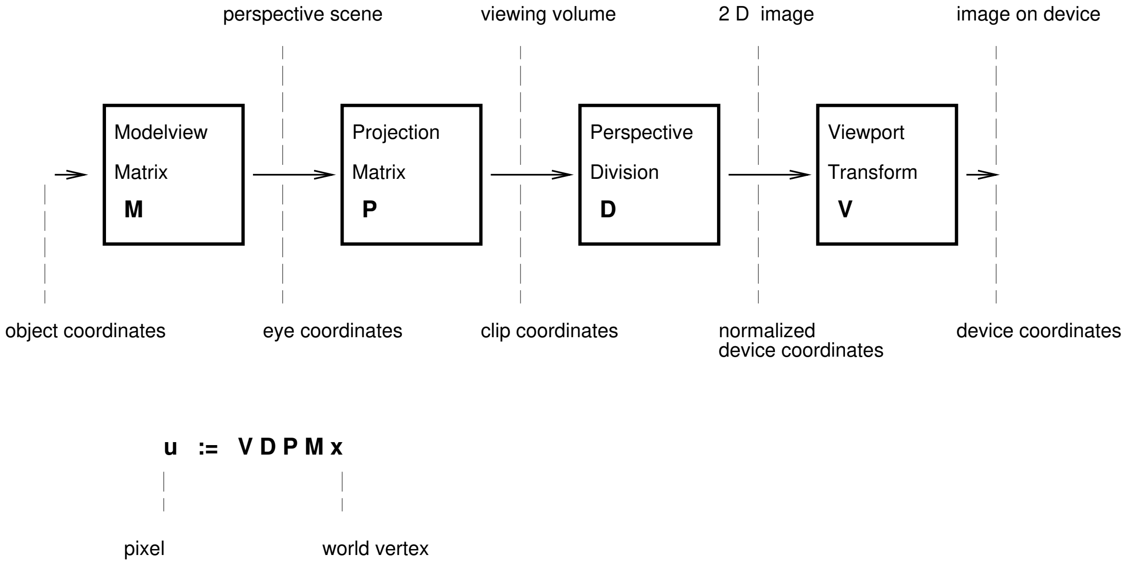


Abbildung 7.1: Viewing in OpenGL: Einzelne Transformationsschritte, zugehörige Matrizen und jeweilige Koordinatensysteme

Weil ferner in OpenGL zuerst die Szene aufgebaut wird und anschließend die Kamera positioniert wird, müssen die Viewing Transformations in OpenGL im Code immer vor den Modelling Transformations stehen.

Als Modelling Transformations stehen die standardmäßigen affinen Abbildungen zur Verfügung: Translation, Rotation sowie Skalierung mit den Spezialfällen Zoom (Streckung) und Spiegelung. Viewing Transformations können aufgrund der oben erwähnten Symmetrie mittels Modelling Transformations realisiert werden, es gibt aber auch spezielle komfortablere Kommandos.

Die Projection Transformation legt den abzubildenden Bereich (das *Viewing Volume*) fest, definiert dadurch das *Clipping* (alles außerhalb des Viewing Volume wird nicht gezeichnet) und führt die Projektion dieses dreidimensionalen Bereichs auf eine zweidimensionale Fläche durch. Für diese Aufgaben gibt es die üblichen zwei Verdächtigen, die *(orthogonale) Parallelprojektion* (*orthographic projection*) und die *Zentralprojektion* (*perspective projection*). Abb. 7.2 zeigt die Definition von Viewing Volume in beiden Fällen mittels vorderer und hinterer *Clipping-Ebene* (bzw. mittels Festlegung des vorderen Clipping-Rechtecks und der Tiefe der hinteren Clipping-Ebene). Voreinstellungsmäßig ist die Projektionsrichtung jeweils die z-Achse, bei der Zentralprojektion liegt das Projektionszentrum (der Betrachter) im Ursprung. Als Resultat ergibt sich jeweils ein durch sechs Clipping-Ebenen begrenzter darzustellender Bereich – ein Quader bzw. ein Parallelepiped bei der orthogonalen bzw. allgemeinen Parallelprojektion, ein Pyramidenstumpf bei der Zentralprojektion. Die Projektionsebene ist parallel zur vorderen und hinteren Clipping-Ebene und wird üblicherweise als zwischen diesen beiden liegend angenommen.

Bleibt noch die Viewport Transformation zu besprechen. Die Bereitstellung eines Fensters zur Ausgabe fällt in den Aufgabenbereich des Window Managers. Der Viewport, d. h. der Bereich im Ausgabegerät, auf dem das Bild ausgegeben werden soll, hat die gesamte Fenstergröße als Voreinstellung, geeignete Ausschnitte davon können aber spezifiziert werden. Um Verzerrungen zu vermeiden, sollte das Aspektverhältnis des Viewports mit dem des Viewing Volume bzw. des Clipping-Rechtecks übereinstimmen. Während der Viewport Transformation wird der z-Wert auf das Intervall $[0, 1]$ normiert und gespeichert. Somit ist Tiefeninformation für den Sichtbarkeitsentscheid nach wie vor verfügbar.

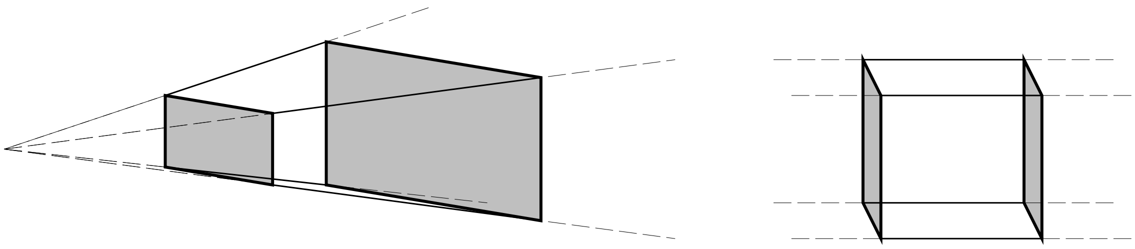

Abbildung 7.2: Viewing Volume bei Zentral- (links) und Parallelprojektion (rechts)

Abschließend sei noch auf eine weitere OpenGL-Spezialität hingewiesen, den *Matrix Stack*. Wie bereits erwähnt, sind die Matrizen aus Abb. 7.1 ihrerseits im Allgemeinen Produkte mehrerer Transformationsmatrizen. Dies gilt insb. für die Modelview Matrix. Man stelle sich eine Raumszene mit einem Tisch und vier Stühlen vor. Das Objekt Stuhl und das Objekt Tisch sind als Modelle vorhanden, jeweils beispielsweise im Ursprung positioniert (Objektkoordinaten) und in einer Vorzugsrichtung orientiert. Der Aufbau von Szene und Bild kann dann wie folgt ablaufen:

- belasse den Tisch an seinem Ort und zeichne ihn;
- speichere die aktuelle Position, rotiere das Objekt Stuhl, führe eine Translation zum Platz des ersten Stuhles durch, zeichne den Stuhl;
- kehre zurück zum Ausgangspunkt, merke diesen, rotiere das Objekt Stuhl, führe eine Translation zum Platz des zweiten Stuhls durch, zeichne diesen Stuhl;
- etc.

Der *Keller* (*stack*) ist die natürliche Struktur zur Verwaltung solcher Abläufe. Bei jeder neuen Transformation wird die um einen weiteren Faktor ergänzte Modelview Matrix gekellert (push-Operation), nach Abschluss des Zeichnens eines Objekts werden die hierzu verwendeten Transformationen durch Entfernen der entsprechenden Matrizen aus dem Keller (pop-Operationen) wieder rückgängig gemacht, und man landet automatisch wieder am Ausgangspunkt der letzten Operation zum Hinzufügen eines Objekts. Dadurch lassen sich hierarchische Objekte (die Szene besteht aus Tisch und Stühlen, Tisch und Stühle bestehen aus Beinen und Platten etc.) elegant platzieren und zeichnen. Da beim Aufbau der Modelview Matrix der Stack in der Regel heftig gebraucht wird, ist hier eine Schachtelungstiefe von 32 standardmäßig vorgesehen. Initialisiert wird der Keller mit der Identität. Demgegenüber besteht der Keller bei der Projection Matrix im Allgemeinen nur aus zwei Matrizen, eben der Projektionsmatrix und der Identität zur Initialisierung. Es gibt aber durchaus auch Fälle mit einer größeren Tiefe, wenn etwa in eine perspektivisch projizierte dreidimensionale Szene ein (sicher nicht perspektivisch zu projizierendes) Help-Fenster mit Text eingeblendet werden soll. Hierzu kann dann temporär auf Parallelprojektion umgeschaltet werden.

# Kapitel 8

# Visualisierung

Bisher haben wir uns mit der geometrischen Modellierung künstlicher Szenen und deren realistischer Darstellung auf dem Bildschirm beschäftigt. Eine andere wichtige Anwendung von Methoden aus der Computergraphik ist die *wissenschaftliche Visualisierung (scientific visualization)*. Diese ist als Teilaufgabe des *wissenschaftlichen Rechnens (scientific computing)* zu sehen, das sich mit der numerischen Simulation von Abläufen, wie sie in Technik und Naturwissenschaften studiert werden, befasst.

Nach einer Simulationsrechnung, also am Ende der numerischen Lösung eines mathematischen Modells im Rechner, liegt das Ergebnis als (i. a. sehr großer) Datensatz vor: die Temperaturverteilung in einem Reaktor, das Geschwindigkeitsfeld um ein Auto im Windkanal oder die Geschwindigkeiten, Beschleunigungen und Kräfte eines Autos beim Elchtest. Wenn auch einige der zentralen Fragestellungen (maximale Temperatur im Reaktor, erliegt das Auto dem A-Klassen-Syndrom?) oftmals durch bloße Bearbeitung der Ergebnisdaten geklärt werden können, so ist doch deren Aussagekraft eher gering. Hier kann die Visualisierung helfen, die visuelle bzw. optische Darstellung der Ergebnisse. Ihre Bedeutung liegt dabei in der Tatsache begründet, dass erstaunlich große Anteile des menschlichen Gehirns dem Sehvorgang und der natürlichen Bildverarbeitung gewidmet sind und dass die Kapazität oder Bandbreite der Informationsverarbeitung unseres visuellen Systems größer als bei allen anderen Sinnen ist. Dadurch können Bilder mit (noch wahrnehmbarer) Information geradezu vollgepackt werden: Sie bieten einen globale Ansicht des Ganzen und erlauben zudem die Kodierung lokaler Phänomene via Größe, Formen, Farben, Musterung, Helligkeit etc.

Um den gesamten Prozess der natürlichen Bildverarbeitung zu verstehen und damit Visualisierungen optimal gestalten zu können, sind detaillierte Kenntnisse aus Physik, Medizin und Psychologie erforderlich. So weit wollen wir hier nicht gehen, sondern uns auf den technischen Teil der Bilderzeugung konzentrieren. Je nach konkreter Aufgabenstellung sind dabei ganz unterschiedliche Methoden gefragt:

- Bei der Visualisierung medizinischer Datensätze, wie sie beim Einsatz bildgebender Verfahren wie Ultraschall, CT oder NMR (MRI) entstehen, sind insbesondere Methoden der *Bildverarbeitung* und des *Volume Rendering* erforderlich.
- Für die Visualisierung von raumauflösenden Simulationsdatensätzen, also bspw. des Geschwindigkeitsfelds und der Temperaturverteilung einer Strömung, spielt die effiziente Generierung von Isoflächen, Höhenlinien, Schnittbildern oder Streichlinien (s. u.) eine herausragende Rolle (also auch hier wieder klassische Aufgaben des *Volume Rendering*).
- Oftmals sind grundlegende Techniken aus der Computergraphik wie die geometrische Modellierung dreidimensionaler Szenen, Beleuchtungsmodelle oder Sichtbarkeitsentscheide von

Bedeutung. Man denke etwa die Simulation der Fahrdynamik eines PKW auf einer Teststrecke, oder an simulierte Bewegungen eines Industrieroboters.

- Schließlich fließen auch Methoden der Virtual Reality ein, wenn etwa ein Architekturmodell visualisiert werden soll, um einen virtuellen Spaziergang durch das Gebäude zu gestatten, oder bei synthetischen Geländemodellen für Flugsimulatoren etc.

Im Folgenden befassen wir uns kurz mit den ersten beiden Themenkreisen.

## 8.1 Methoden der Bildverarbeitung

Die Visualisierung streift auch den Bereich bildgebender Verfahren, bei denen Messwerte (Radar-Aufnahmen der Satelliten ERS-1 und ERS-2, Computertomogramme, Aufnahmen von Rasterelektronenmikroskopen etc.) in Bilder im herkömmlichen Sinne umgewandelt werden sollen. Unter einem Bild ist dabei ein diskreter Datensatz zu verstehen, genauer ein zwei- oder (in einem verallgemeinerten Sinne auch) dreidimensionales Feld von Graustufen- oder Farbwerten. Im Gegensatz zur Visualisierung von Simulationsdaten, bei der aus künstlich errechneten Daten Bilder erzeugt werden sollen, handelt es sich hier also eher um eine optische Rekonstruktion. Hierbei finden Techniken aus der Bildverarbeitung starke Anwendung.

In einer groben Einteilung kann man fünf Kategorien bildverarbeitender Operationen unterscheiden:

- *Geometrische Verarbeitung (geometric processing)*: Veränderung der Form eines Bildes bzw. von Teilen davon; solche *geometrischen Operationen* hängen von der Position des Pixels ab, nicht jedoch von den Grauwerten im Pixel und in seinen Nachbarpunkten.
  Beispiele:
  - Vergrößerung/Verkleinerung,
  - Drehung,
  - Verzerrung.
- *Punktweise Bearbeitung (point-to-point processing)*: Modifikation des Bildes durch einfache lokale Veränderungen in den Pixelwerten; solche *Punktoperationen* hängen i. A. vom lokalen Grau- bzw. Farbwert ab, nicht jedoch von Position oder Umgebung.
  Beispiele:
  - *Konstantenaddition*: Modifikation aller Pixelwerte um eine Konstante;
  - *contrast stretching*: Aufblähen der Helligkeitswerte auf den vollen darstellbaren Bereich (Erhöhung des Kontrasts, Aufspüren von Details);
  - *Falschfarbendarstellung*: unterschiedlichen Graubereichen werden verschiedene Farbtöne zugewiesen, wobei jede Farbe den gesamten Intensitätsbereich durchläuft; große Unterschiede in den Pixelwerten äußern sich damit als Farbunterschiede, kleinere wie zuvor in der Intensität (weitere Erhöhung des Kontrasts);
  - *Exponentialtransformation*: die Belichtung von Filmen erfolgt nicht proportional, sondern logarithmisch zur Lichtstärke; eine Exponentialtransformation kann die ursp. Intensitätswerte restaurieren.
- *Lokale Bearbeitung (local-to-point processing)*: Die Umgebungsgrauwerte fließen in den neuen lokalen Wert ein.
  Beispiele:
  - *lokale gewichtete Mittelungsoperatoren*: Auswertung einer lokalen Umgebung (Rechteck, Kreis, ...), gewichtete Mittelung (*Faltung (convolution)*, allg. Glättungsoperatoren);

- *lokale Rangoperatoren*: Auswertung einer lokalen Umgebung, Sortieren der auftretenden Werte, Auswahl bzgl. der sortierten Werte (Median, Mittel der Extremwerte etc.);
- *Segmentierung*: Zuordnung zu Klassen via Schwellwerte, *Filterung*;
- *Flanken- bzw. Kantendetektion*: Pixelwerte in glatten Bereichen werden abgeschwächt oder bleiben unverändert, Unterschiede werden zur Konturierung hervorgehoben.
- *Merkmalsextraktion*: Lokalisierung bestimmter Merkmale (Schwellwertüberschreitung, Flanken, Oszillationen, Muster, Texturen etc.).

• *Vergleichende Verarbeitung (ensemble processing)*: Auswertung mehrfacher Bilder zur Ableitung eines neuen Bilds.
Beispiele:

- *Bewegungsdetektion*: Erkennen von Veränderungen oder Bewegungen durch Vergleich zweier zu verschiedenen Zeitpunkten aufgenommener Bilder, Anwendungen in Medizin und bei Satellitenaufnahmen;
- *Differenzenbilder*: Anwendungen wiederum in der Medizin (Visualisierung des Blutkreislaufs über Vergleich vor und nach der Injektion von Kontrastmittel, da Blut, Blutbahnen und das umgebende Gewebe nahezu dieselben Röntgen-Absorptionseigenschaften haben und somit schlecht sichtbar gemacht werden können) und bei Satellitenaufnahmen (Aufnahmen des Landsat-Satelliten als Kombination der Antworten auf vier Frequenzbänder zur Ermittlung von Vegetation oder Aufspürung von Bodenschätzen);

• *Globale Verarbeitung (domain processing)*: komplizierte Pixelwertmodifikationen auf der Grundlage lokaler und globaler Information (Fourier- und verwandte Transformationen); diese Verfahren stellen die aufwendigsten und mächtigsten Methoden dar und sind vielseitig einsetzbar, etwa zur Bildschärfung, Flankenhervorhebung, Kontrasterhöhung, Detailrestaurierung sowie zur Komprimierung.
Beispiele zur Kompression:

- *Cosinustransformation*: auf ihr beruht der JPEG-Standard (Joint Photographic Experts Group) zur Komprimierung digitaler Bilddaten;
- *Wavelet-Transformation*: verwandt mit der Fourier-Transformation, aber mit in Ort/Zeit *und* Frequenz lokalisierten Basisfunktionen (*Wavelets*); hohe Kompressionsraten, effiziente Algorithmen;

Am Anfang (medizinischer) bildgebender Verfahren standen Röntgenaufnahmen. Diese stellen Summationsbilder dar, die häufig wegen der gegenseitigen Überlagerung Details nicht erkennen lassen. Mit der *Schichtuntersuchung* (*Tomographie*) können morphologische Einzelheiten in einer wählbaren Schichttiefe scharf abgebildet werden. Die darüber- und darunterliegenden Schichten werden verwischt. Aus den einzelnen Schichtbildern können dann dreidimensionale Ansichten generiert werden. Man unterscheidet Verfahren der *Transmissionstomographie*, bei denen die Passage von Strahlen durch das abzubildende Objekt betrachtet wird, und Verfahren der *Emissionstomographie*, bei denen von inkorporierten Radionukliden ausgesandte Strahlen erfasst werden. Beispiele für erstere sind etwa:

- *Röntgenbasierte Computertomographie (CT)*: Sie basiert mathematisch auf der *Radon-Transformation*. Anhand der Abschwächung von gebündelten Röntgenstrahlen beim Durchgang durch das zu untersuchende Gewebe werden einzelne Schnittbilder erzeugt, die die Dichte des Gewebes visualisieren.
- *Kernspintomographie (NMR, MRI)*: Hier wird ein starkes Magnetfeld anstelle von Röntgenstrahlen zur Generierung der Schnittbilder eingesetzt. Visualisiert wird wiederum die Dichte der Materie. Vorteile gg. CT sind u. a. die höhere Auflösung, die größere Empfindlichkeit gegenüber weichem Gewebe sowie die geringere Belastung des Patienten.

– *Confokale Laser-Scanning-Mikroskopie (CLSM)*: Hier werden statt Röntgen- Laserstrahlen durchgeschossen. Verwendung findet die CLSM vor allem in der Biologie (Präparate, nicht am Menschen).

Beispiele für Emissionstomographie sind:

– *Positronen-Emissions-Tomographie (PET)*: Gemessen wird hier die beim Positronenzerfall entstehende Strahlung. Sie dient somit der Darstellung Positronen-emittierender Radionuklide. Das Verfahren ist sehr aufwendig, da die Positronen in einem Zyklotron erzeugt und dann innerhalb kurzer Zeit (Halbwertszeit 60 bis 120 Minuten) eingesetzt werden müssen.

– *Single Photon Emission Computed Tomography (SPECT)*: Hier werden Gammaemitter verwendet. Die SPECT ist der PET hinsichtlich Qualität unterlegen, aber weniger aufwendig und für den Patienten weniger belastend, billiger und oftmals hinreichend gut.

Erfasst wird bei der Emissionstomographie durch Injektion schwach radioaktiv markierter Substrate deren metabolische (d. h. Stoffwechel-) Aktivität (also biologische statt physikalischer Struktur). Beispielsweise werden Zucker oder Aminosäuren von tumorbefallenem Gewebe heftiger abgebaut als von gesundem, so dass diese Substrate die Radionuklide an die richtige Stelle transportieren. Beim Zerfall der Positronen (PET) bzw. durch unmittelbare Emission (SPECT) werden dann Paare von Gammaquanten frei, die sich in exakt entgegengesetzter Richtung vom Ort ihrer Freisetzung entfernen und die als Strahlung erfasst werden können.

In der Medizin kommen heute auch Kombinationen von Transmissions- und Emissionstomographien zum Einsatz, etwa als Überlagerungen einer Kernspintomographie mit einer SPECT. Diese hybride Vorgehensweise bietet den Vorteil, dass sowohl strukturelle als auch metabolische Information sichtbar gemacht und zueinander in Bezug gesetzt werden kann. Das CT-Prinzip kann auch noch allgemeiner eingesetzt werden, etwa für Visualisierungen der inneren Struktur der Erde auf der Grundlage der Ausbreitung mechanischer (seismischer) Wellen.

## 8.2 Visualisierung raum- und zeitaufgelöster Simulationsdaten

Gegeben sei das (in Form einer großen Menge von Zahlen vorliegende) Ergebnis einer numerischen Simulation, also i. A. eine skalare oder vektorwertige Funktion $f : \mathbb{R}^d \supset \Omega \to \mathbb{R}^m$, $m \in \mathbb{N}$, über einem $d$-dimensionalen Feld. Im Folgenden diskutieren wir kurz einige Techniken bzw. Möglicheiten zur optischen Darstellung solcher Datensätze. Der uns vor allem interessierende Fall ist dabei $d = 3$ bzw., bei Zeitabhängigkeit, $d = 4$.

- **Auswahl und Darstellung von (niedrigdimensionalen) Teilbereichen:**
  Die direkte Visualisierung von im drei- oder höherdimensionalen Raum gegebenen Funktionen ist schwierig. Deshalb beschränkt man sich oft auf die Darstellung bzw. Hervorhebung zwei- oder eindimensionaler Teilbereiche (*Dimensionsreduktion*). Die populärsten hiervon sind:

  – *Schnittebenen (Slices, siehe Abb. 8.1)*: Man betrachtet die interessierenden Größen nicht im Grundgebiet $\Omega$, sondern im Schnitt einer Ebene $E$ mit $\Omega$. Auf der Schnittebene können dann zweidimensionale Techniken angewandt werden.

  – *Isoflächen (Isosurfaces, siehe Abb. 8.2)*: Dargestellt wird die Hyperfläche, auf der eine der berechneten skalaren ($m = 1$) Funktionen $f$ einen fest vorgegebenen Wert annimmt:

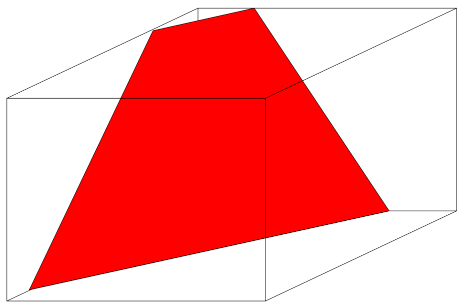

Abbildung 8.1: Beispiel einer Schnittebene

$I(c) := \{\mathbf{x} \in \Omega : f(\mathbf{x}) = c\}$. Will man Isoflächen interaktiv erzeugen und verschieben (Änderung der Konstante $c$), so bedarf es einerseits effizienter Algorithmen zur schnellen Bestimmung der Isoflächen (etwa der Marching-Cubes-Algorithmus), andererseits einer hinreichend leistungsfähigen Graphik-Hardware (bei einem typischen Beispiel einer Simulation von Strömung und Transport in einer komplizierten Hindernisgeometrie (Auflösung $512^3$ Zellen) besteht eine Isofläche bspw. aus 4 Millionen Polygonen). Auf der Isofläche lassen sich zweidimensionale Techniken zur Visualisierung weiterer Größen einsetzen. Das Pendant zu Isoflächen in zweidimensionalen Fall sind *Isolinien* bzw. *Höhenlinien*.

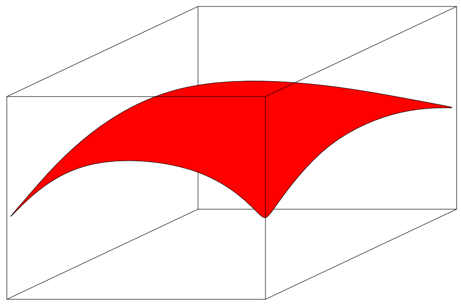

Abbildung 8.2: Beispiel einer Isosurface

– *Stromlinien (Streamlines)*: Stromlinien dienen zur Visualisierung von Vektorfeldern ($m > 1$), d. h. von vektorwertigen Daten. Sie sind definiert als Kurven in $\Omega$ mit in jedem Punkt zum jeweiligen Ortsvektor des Feldes paralleler Tangente. Die Auswahl der Stromlinie erfolgt durch die Auswahl eines Startpunkts.

– *Bahnlinien (Particle Tracing)*: Bahnlinien werden zur Visualisierung zeitabhängiger Vektorfelder eingesetzt. Ein virtuelles Teilchen wird an einem Startpunkt in das Gebiet $\Omega$ eingebracht und folgt nun in der Zeit dem jeweiligen Ortsvektor des Vektorfeldes. Die Bahn des Teilchens (theoretisch eine kontinuierliche Kurve, praktisch eine Folge diskreter Positionen zu diskreten Zeitpunkten) definiert dabei die Bahnlinie.

– *Streichlinien (Streaklines)*: Auch Streichlinien dienen der Visualisierung zeitabhängiger Vektorfelder. Ab einem Startzeitpunkt werden an einem Startpunkt (theoretisch kontinuierlich, praktisch in festen Abständen) Partikel injiziert, die sich alle den jeweiligen Ortsvektoren des Feldes folgend weiterbewegen. Die dadurch zu einer vorgegebenen Zeit entstandene Linie wird Streichlinie genannt. Bahnlinien geben also die Bewegung *eines* Partikels über ein Zeitintervall wieder, Streichlinien dagegen die Position *vieler* vom gleichen Ausgangspunkt startenden Partikel zu einem festen Zeitpunkt. Im stationären Fall stimmen Streichlinien und Bahnlinien mit Stromlinien überein.

Neben Streichlinien sind auch *Streichbänder* gebräuchlich. Hier werden die Partikel nicht an einem einzigen Punkt, sondern längs eines Intervalls injiziert. Im zeitlichen Fortlauf entstehen somit keine Kurven, sondern breite Bänder. Vorteile von Streichbändern sind die Möglichkeit zur Darstellung weiterer Information auf ihnen sowie das Sichtbarmachen von Rotationen. Zur Verbesserung des dreidimensionalen Eindrucks werden auch *Streichröhren* eingesetzt.

- **Farbgebung**:
Die Intensität einer (skalaren) Größe kann durch den Farbton oder durch die Intensität ausgedrückt werden. Dabei sollte die Visualisierung die Intuition unterstützen (im Falle der Temperatur z. B. blau für kalt und rot für warm und nicht umgekehrt, bei einer Stoffkonzentration schwache Intensität für geringe Werte etc.). Außerdem können verschiedene Farben für verschiedene Größen benutzt werden (rote Töne für die Temperatur, Blautöne für die Stoffkonzentration). Hierbei setzt die Wahrnehmungsfähigkeit des Menschen allerdings Grenzen: Bei mehr als fünf Farben nimmt die parallele Verarbeitungsfähigkeit ab. Wichtig ist in jedem Fall die passende Wahl der Skala: Deckt diese einen zu großen Bereich ab, verschwinden kleinere Unterschiede in den Daten, ist der darstellbare Bereich zu klein, erscheinen marginale Unterschiede als gewaltig.

- **Dimensionserhöhung**:
Eine auf einem zweidimensionalen Gebiet gegebene Größe kann in einem dreidimensionalen Bild als Funktion über dem Grundgebiet aufgetragen werden (*Funktionsplot*). Solche *Surface Plots* machen allerdings nur in 2 D Sinn.

- **Verwendung von Darstellungsprimitiven**:
Die meisten Visualisierungstechniken stellen die ausgewählten niedrigdimensionalen Teilbereiche als Kurven oder Flächen dar. Alternativ dazu können auch bestimmte Darstellungsprimitive aufgebracht werden. Ein Beispiel hierfür sind Pfeile im Falle eines Geschwindigkeitsfelds, die via Größe und Farbe dann noch weitere Informationen tragen können. Kugeln oder Bälle sind eine andere gebräuchliche Primitivenart, die vor allem bei Bahn- oder Streichlinien eingesetzt wird und die einzelnen virtuellen Teilchen darstellen soll. Werden sie in festem Zeitabstand in die Szene eingebracht, ist ihr Abstand zugleich ein Maß für die absolute Geschwindigkeit im Fluid.

- **Unterstützung realistischer Darstellung**:
Projektionen (Zentral- oder Parallelprojektion) ergeben einen perspektivischen Eindruck und unterstützen die räumliche Wahrnehmung. Beleuchtungstechniken (Lichtquellen, Schatten, Spiegelungen, globale Beleuchtung) können zu realistischeren Bildern führen. Die realistische Darstellung spielt bei der Visualisierung von Datensätzen kaum eine Rolle, bei Aufgaben der Visualisierung in Virtual-Reality-Anwendungen kommt ihr allerdings eine entscheidende Bedeutung zu.

- **Verwendung von Texturen**:
Iso- oder Schnittflächen bzw. Bänder können zur Wiedergabe zusätzlicher Information mit einer Textur versehen werden, d. h., ein Muster wird auf die Fläche aufgetragen. Dies kann ein fest vorgegebenes Muster zur Identifizierung der Fläche oder ein durch eine bestimmte Problemgröße definiertes Muster sein (Isofläche der Stoffkonzentration, Norm bzw. Absolutbetrag der Geschwindigkeit via Farbe, Visualisierung eines Feldes über eine Vektorfeldprojektion auf die Fläche, vergleichbar Eisenfeilspänen in einem Magnetfeld). Die Textur kann aber auch die Beschaffenheit des Grundgebiets visualisieren (Versickerung von Öl im Erdreich: auf einer Isofläche der Ölkonzentration wird die Charakteristik des Erdbodens (Sand, Lehm, Kies etc.) dargestellt).

- **Beschriftung und Legende**:
Ohne Legende sind Visualisierungen oftmals nicht zu verstehen oder können leicht missverstanden werden (beispielsweise aufgrund einer Fehlinterpretation der Skalen). Beschriftungen im Bild können hilfreich sein, sie können aber auch den Blick des Betrachters von der

eigentlichen Information ablenken.

Die Visualisierung stützt sich auf die diskreten Ergebnisdaten eines Simulationslaufs. Damit erbt sie in natürlicher Weise auch die zugrunde liegende diskretisierte Geometrie, also das Gitter (strukturiert oder unstrukturiert, regulär oder adaptiv verfeinert). Kann ein Visualisierungsalgorithmus die ererbte Geometrie direkt behandeln, dann können Berechnung und graphische Darstellung auf derselben Gitterstruktur erfolgen. Andernfalls müssen die berechneten Daten auf ein geeignetes Gitter abgebildet (d. h. interpoliert) werden.

Zur Beschreibung von Datensätzen verwenden Visualisierungssysteme wie z. B. AVS/Express eine Komponentenhierarchie (vgl. Abb. 8.3).

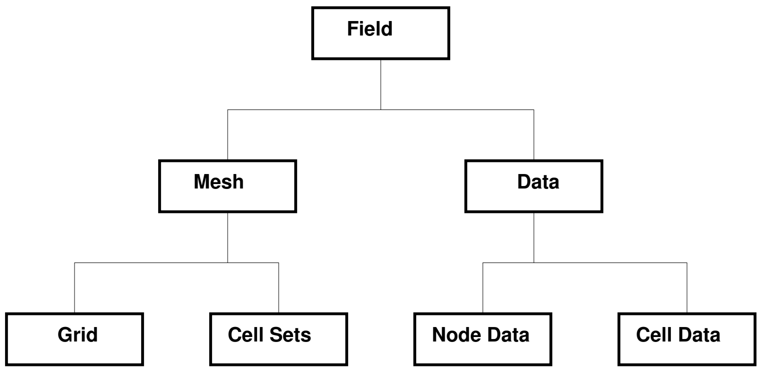


Abbildung 8.3: Komponentenhierarchie in AVS

- **Field**: Das Feld, die oberste Komponente in der Hierarchie, repräsentiert die gesamte Information (in voller Allgemeinheit). Es setzt sich zusammen aus einer Netz- und einer Datenkomponente.
- **Mesh**: Die Netz-Komponente liefert die geometrische Beschreibung des zugrunde liegenden Gebiets (eigentliche und faktische Dimensionalität des Gebiets[1], Typ des Netzes: unstrukturiert, strukturiert, rechtwinklig, uniform, vgl. Abb. 8.4) und definiert, wo und wie die Daten organisiert sind. Die entsprechenden Informationen werden mittels einer Gitter- und einer Zellen-Komponente gehalten.
- **Grid**: Die Gitter-Komponente enthält die Positionen der Gitterpunkte. Auch hier wird zwischen unstrukturierten und strukturierten Gittern unterschieden. Je mehr Struktur ein Netz bzw. ein Gitter trägt, umso weniger Information muss explizit gehalten werden. Im uniformen Fall etwa liegen Positionen und Nachbarschaftsrelationen der Gitterpunkte implizit fest.
- **Cells**: Eine Menge von Punkten mit zugeordneten Werten reicht zur Erzeugung eines Bildes (einer Isosurface etwa) noch nicht aus. Die Frage der Nachbarschaft bzw. des Zusammenhangs (*connectivity*) muss beantwortet werden. Dies leistet die Zellen-Komponente. Zellen stellen dabei allgemeine geometrische Einheiten dar und können Punkte, Kanten, Flächen (Dreiecke, Vierecke) oder Volumen (Tetraeder, Hexaeder) sein. Bei strukturierten Gittern ergibt sich das topologische Beziehungsgeflecht (wer grenzt an wen) automatisch aus der Information der Gitter-Komponente, bei unstrukturierten Gittern muss es separat gespeichert werden.
- **Data**: Die Daten-Komponente repräsentiert die Daten des Gitters (also die berechneten bzw. gespeicherten bzw. darzustellenden Funktionen.
- **Node Data**: Knotendaten sind diejenigen Daten, die den Knoten (Gitterpunkten) im Gitter zugeordnet sind.

[1] Eine Isosurface ist ein eigentlich zweidimensionales Gebilde, lebt faktisch aber im dreidimensionalen Raum.

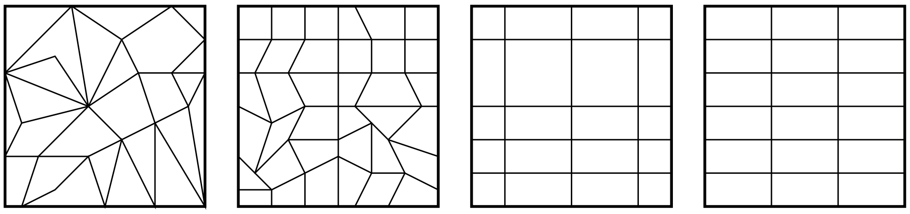

Abbildung 8.4: Netz- bzw. Gittertypen am Beispiel AVS: unstrukturiert (links) und strukturiert (konform, rechtwinklig, uniform; v. l. n. r.). Im unstrukturierten Fall müssen alle Gitterpunkte und die Zusammenhangsinformationen explizit gespeichert werden. Im konformen Fall reichen die Positionen der Punkte, im rechtwinkligen Fall die Vektoren der Maschenweiten, und bei uniformen Gittern reichen die konstanten Maschenweiten.

- **Cell Data**: Zelldaten sind die Daten, die den Zellen zugeordnet sind. In der Strömungsmechanik arbeitet man bspw. oftmals mit *versetzten Gittern (staggered grids)*, bei denen die Druckwerte den Mittelpunkten und die Geschwindigkeitswerte den Seitenmitten der quaderförmigen Zellen zugeordnet werden.

Die unterschiedliche Zuordnung der jeweiligen Größen zu Punkten oder Zellen ist für die Visualisierung durchaus von Bedeutung: Im ersten Fall erhält nur der Punkt einen entsprechenden Wert bzw. eine entsprechende Farbe, dazwischen wird interpoliert. Sind Größen Zellen zugeordnet, werden die ganzen Zellen den Werten entsprechend konstant eingefärbt (was aus ästhetischen Gesichtspunkten natürlich unerwünscht ist, da es zu harten Übergängen führt).

Ein weiterer zentraler Punkt ist die Schnittstelle zwischen Simulations- und Visualisierungsdaten. Moderne Diskretisierungstechniken reduzieren den Datenaufwand bei der Berechnung zum Teil erheblich (hierarchische Strukturen, adaptive Verfeinerung), Visualisierungssysteme setzen oftmals auf wesentlich einfacheren und reguläreren Strukturen auf und verschenken somit viel an Effizienz. Zum Beispiel ist es mehr als ärgerlich, wenn zu Visualisierungszwecken die Daten eines adaptiven Gitters erst auf ein reguläres interpoliert werden müssen, oder wenn die Beschreibungsart geändert werden muss. Maßgeschneiderte Schnittstellen helfen hier, Reibungsverluste zu vermeiden.

Schließlich sei darauf hingewiesen, dass auch bei der in diesem Abschnitt behandelten Visualisierung von raum- und zeitaufgelösten Simulationsdaten Techniken der Bildverarbeitung hilfreich sein können, beispielsweise die Merkmalsextraktion zum Aufspüren bestimmter Details.

## 8.3 Vom Bild zum Film

Bei zeitabhängigen Simulationsdaten befriedigt das Einzelbild noch nicht alle Wünsche, da immer nur ein isolierter Zeitpunkt betrachtet werden kann. Sequenzen von Einzelbildern erhöhen die Aussagekraft bereits, Ziel bei zeitabhängigen Problemen muss jedoch der Film sein. „Ich könnte mir sogar vorstellen, dass man diesen Bildern das Laufen beibringen kann, indem man, was auch sonst, viele Bilder aneinanderhängt“ (Zitat eines Mathematikers, Februar 1999). In der Tat funktioniert's so, genau genommen mit 25 bzw. 30 Bildern pro Sekunde (Videobildfrequenz), oft reichen aber auch weniger. Allerdings sind einige Dinge zusätzlich zu beachten. Die Folge der Einzelbilder muss natürlich so gewählt werden, dass weder ruckartige Übergänge noch zu langsames Fortschreiten den Eindruck stören. Solche Rucke können auch durch das Aufeinandertreffen unterschiedlicher Farbgebungen auf zwei aufeinanderfolgenden Einzelbildern entstehen. Da die Übernahme der Skala des ersten Bildes für alle folgenden Bilder aber ebenfalls Risiken birgt (werden die Unterschiede im Lauf des Films kleiner, ist später nichts mehr zu erkennen, werden sie größer, fallen immer mehr Werte aus dem darstellbaren Bereich), müssen im Vorfeld über alle Bilder globale Minima und Maxima ermittelt werden, um eine einheitliche und aussagekräftige Einfärbung sicherzustellen.

Für Filme geeignet sind prinzipiell alle Einzelbild-Visualisierungstechniken, die die Verhältnisse zu einem festen Zeitpunkt beschreiben. Welche Darstellungsmöglichkeit die Dynamik am besten wiedergibt, hängt vom speziellen Problem ab.

Was aber tun, wenn die Erzeugung einer solchen Vielzahl von Einzelbildern zu aufwendig ist? Bei dreidimensionalen Strömungssimulationen liegt die Rechenzeit für einen Zeitschritt leicht im Stundenbereich. Für einen einminütigen Film mit 1800 Bildern bedeutet dies somit einen nicht unerheblichen Rechenaufwand. Da die Zeitschrittweite bei der Simulation numerischen Zwängen unterworfen ist (Stabilitätsbedingungen), fällt eine gewisse Anzahl von einzelnen Bildern (Frames) ohnehin an. Man kann aber u. U. auf die visualisierungsbedingte Verkleinerung der Schrittweite verzichten und stattdessen weitere Bilder (Frames) durch Interpolation erzeugen (*Inbetweening*). Man muss sich aber der Tatsache bewusst sein, dass das Produkt dann keine Visualisierung von Simulationsdaten im engeren Sinne mehr ist (siehe auch Regel 6 im nachfolgenden Abschnitt).

## 8.4 Bunte Bildchen über alles

Dass die Visualisierung wichtig, ja für das moderne wissenschaftliche Rechnen unverzichtbar ist, haben wir jetzt gelernt. Etwas Skepsis kann aber dennoch nie schaden.

Ein wichtiger Aspekt ist zunächst, dass Visualisierungen nicht „besser“ als die ihnen zugrunde liegenden Daten sein können. Ein Bild sagt beispielsweise per se nichts über die Qualität des verwendeten Modells aus, auch wenn Falsifizierungen manchmal möglich sind. Außerdem führt auch der denkbar langsamste numerische Algorithmus (so er denn korrekt ist) zum richtigen Ergebnis und somit zum hübschen Bild, egal, wie lange der Rechner dafür schwitzen musste.

Ein zweiter Problempunkt sind *optische Artefakte*: Spiegelt das, was ich sehe, tatsächlich die Wirklichkeit wider, oder handelt es sich nur um eine optische Täuschung?

Drittens muss das Problem der fehlerhaften Interpretation beachtet werden. Deute ich die Abbildungen richtig (ein Problem, das vor allem in der Frühphase der Computertomographie allgegenwärtig war, als Tumoren oftmals schlichtweg übersehen wurden), handelt es sich bei dem interpretierten Phänomen wirklich um eine Eigenschaft der Daten, oder liegt es vielleicht nur an der Wahl der Parameter (Farbskala, Intensität, Kontrast etc.)? Trotz aller Suggestivkraft guter Visualisierungen sollte man von der Disziplin bzw. dem Phänomen, welches visualisiert wurde, ein gewisses Maß an Ahnung haben, bevor man sich auf Interpretationen einlässt.

Schließlich sind die Manipulationsmöglichkeiten bei der Visualisierung nahezu unbegrenzt. In diesem Zusammenhang sei die Arbeit *13 Ways to Say Nothing with Scientific Visualization* (A. Globus, E. Raible, NASA Ames Research Center, Report RNR-92-006) erwähnt, in der dem angehenden Visualisierer folgende Ratschläge erteilt werden:

1. *Never include a colour legend*:
   Auf die Optik kommt's an, Legenden lenken den Betrachter nur ab.

2. *Avoid annotation*:
   Erklärungen unterminieren bloß die visuelle Verwirrung.

3. *Never mention error characteristics*:
   Wenn ein Bild gut aussieht, muss es einfach auch korrekt sein.

4. *When in doubt, smooth*:
   Störende Peaks immer herausglätten, denn: Schönheit und rasche Publizierbarkeit sind allemal wichtiger als Genauigkeit.

5. *Avoid providing performance data*:
   Nur Kleingeister scheren sich darum, wieviel Zeit die Visualisierung in Anspruch nimmt.

6. *Quietly use stop-frame video techniques*:
   Warum für eine Animation unnötig viele Einzelbilder (Frames) produzieren, wenn einzelne wiederholt werden können oder zwischen einzelnen interpoliert werden kann?

7. *Never learn anything about the data or scientific discipline*:
   Wer nicht weiß, was er im Bild erwartet, muss sich ggfs. auch nicht auf Fehlersuche begeben – es lebe die Unvoreingenommenheit.

8. *Never compare your results with other visualization techniques*:
   Die Bilder der anderen könnten besser aussehen.

9. *Avoid visualization systems*:
   Warum den Mist von anderen benutzen (vgl. Regel 8), wenn man selbst hübsche Sachen erfinden kann?

10. *Never cite references for the data*:
    Das wäre nur eine Einladung zum Nachprüfen und Nörgeln.

11. *Claim generality, but show results from a single data set*:
    Wozu gibt's 'o. B. d. A.'?

12. *Use viewing angle to hide blemishes*:
    Warum 'Hidden Surface Removal' nicht zum 'Disturbing Detail Removal' benutzen?

13. *'This is easily extended to 3 D'*:
    siehe Regel 11.

Damit die Visualisierung aber nicht alles abbekommt, sei auch auf die *Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers* (D. Bailey, Supercomputer Review, Aug. 1991, pp. 54–55) hingewiesen: Getrickst wird schließlich überall.

## Kapitel 9

# Farben und Farbmodelle

Siehe im Buch den Abschnitt 1.6.

# Kapitel 10

# Globale Beleuchtung

## 10.1 Einleitung

Photorealismus, d. h. die Erzeugung synthetischer Bilder im Rechner, die von Photographien im Idealfall nicht zu unterscheiden sind, ist wohl das große Ziel der Computergraphik schlechthin. Hier ist zunächst die geometrische Modellierung gefordert, die möglichst realitätsnahe Beschreibungen der Szenen inkl. geometrischer und (materialabhängiger) optischer Eigenschaften liefern muss. Dafür, was dann aus dieser Szenenbeschreibung gemacht wird, ist der *Renderer* verantwortlich, der im Wesentlichen die Aufgaben Perspektive, Sichtbarkeit und Beleuchtung auszuführen hat. Die Modellierung der *globalen Beleuchtung*, also der Licht-Wechselwirkungen aller Objekte der Szene miteinander, ist dabei von herausragender Bedeutung und soll im Folgenden diskutiert werden.

Der erste Ansatz, das sog. *Ray Tracing*, geht auf Whitted (1980) und Appel (1968) zurück. Es kann spiegelnde Reflexion perfekt wiedergeben, diffuse Beleuchtung bzw. ambientes Licht dagegen überhaupt nicht. Die resultierenden Bilder wirken synthetisch und zu perfekt. Gewissermaßen das Gegenstück, *Radiosity*, wurde 1984 von Goral et al. vorgestellt und gibt diffuse Beleuchtung perfekt wieder. Spiegelnde Reflexionen sind allerdings nicht möglich. Die resultierenden Bilder wirken natürlicher, aber noch nicht realistisch – es fehlen eben Spiegelungs- und Glanzeffekte. In der Folge wurden zahlreiche Ansätze zur Kombination von Ray Tracing und Radiosity entwickelt, ausgehend von der einfachen Hintereinanderschaltung bis hin zu komplizierteren Verfahren. Trotzdem bleiben Probleme bei der Wiedergabe indirekter spiegelnder Beleuchtungen (etwa über einen Spiegel), sog. *Caustics*. Weitere Entwicklungsstufen waren *Path Tracing* (Kajiya 1986) und *Light* oder *Backward Ray Tracing* (Arvo 1986). Beide sind prinzipiell in der Lage, das globale Beleuchtungsproblem zu lösen, haben jedoch Probleme hinsichtlich Aufwand bzw. Einschränkungen bei der Geometriebeschreibung. Den (derzeitigen) Stand der Dinge stellt *Monte Carlo Ray Tracing mit Photon Maps* dar (Jensen 1995), das auch Caustics in befriedigender Qualität effizient wiedergeben kann.

## 10.2 Das Problem der globalen Beleuchtung

### 10.2.1 Größen aus der Radiometrie

Radiometrie ist die Lehre von der (physikalsichen) Messung elektromagnetischer Energie. Einige wichtige Größen hieraus sind für das Folgende von Bedeutung. Bezugsgröße ist dabei eine Oberfläche $A$ in der Szene. $x \in A$ bezeichne einen Punkt auf der betrachteten Oberfläche, $\omega \in S^2$ mit der Einheitssphäre $S^2$ bzw. $\omega \in H^2$ mit der Einheitshemisphäre $H^2$ eine Strahlrichtung. Ferner führen wir das Konzept des *Raumwinkels* (*solid angle*) ein, der i. A. über die Fläche auf der Oberfläche von $H^2$ gemessen wird, maximal also den Wert $2\pi$ annimmt, bzw. $2\pi sr$ mit der künstlichen Einheit *Steradiant* (*steradian*). Zu einem Flächenstück auf einer Hemisphäre mit beliebigem Radius $r$ ergibt sich der Raumwinkel einfach durch Division des Flächeninhalts durch $r^2$. Der Winkel $\theta = \theta(x, \omega)$ bezeichne schließlich den Winkel zwischen Flächennormale in $x$ und der Strahlrichtung $\omega$ (siehe Abb. 10.1). Doch jetzt zu den radiometrischen Größen:

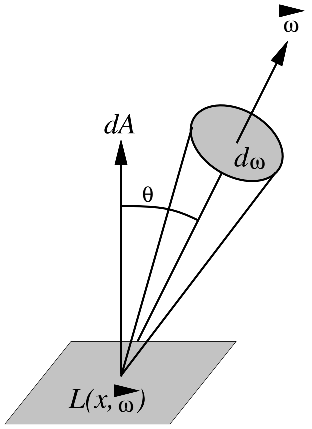


Abbildung 10.1: Radiometrische Größen im Überblick

- *radiant energy* $Q$: elektromagnetische Energie bzw. Lichtenergie (Einheit Joule);
- *radiant flux* $\Phi$: Energiefluss, -eintritt oder -ausgang pro Zeit (Einheit Watt; üblicherweise wird nur der statische Fall betrachtet, und die Begriffe Radiant Energy und Radiant Flux werden synonym verwandt);

$$\Phi := \frac{\partial Q}{\partial t} \tag{10.1}$$

- *radiance* $L(x, \omega)$: Radiant Flux in einem infinitesimal dünnen Strahl bzw. Flux pro Flächeneinheit $\mathrm{d}A \cos\theta$ senkrecht zum Strahl und pro Raumwinkel $\mathrm{d}\omega$ in Strahlrichtung $\omega$ (Einheit $W/m^2 sr$);
  $L_i$, $L_o$, $L_r$, $L_e$: eintreffende, ausgehende, reflektierte und emittierte Radiance;
- *irradiance* $E$: auftreffender Radiant Flux pro Flächeneinheit (Einheit $W/m^2$, Summation bzw. Integration über alle eingehenden Richtungen $\omega_i$);

$$\begin{aligned} \mathrm{d}E &:= L_i(x, \omega_i) \cos\theta_i \, \mathrm{d}\omega_i \, , \\ E(x) &:= \int_{H^2} L_i(x, \omega_i) \cos\theta_i \, \mathrm{d}\omega_i \end{aligned} \tag{10.2}$$

- *radiosity* $B$: ausgehender Radiant Flux pro Flächeneinheit (Einheit $W/m^2$, Summation bzw. Integration über alle ausgehenden Richtungen $\omega_o$);

$$\begin{aligned} \mathrm{d}B &:= L_o(x, \omega_o) \cos\theta_o \, \mathrm{d}\omega_o \, , \\ B(x) &:= \int_{H^2} L_o(x, \omega_o) \cos\theta_o \, \mathrm{d}\omega_o \end{aligned} \tag{10.3}$$

- *radiant intensity* $I$: ausgehender Radiant Flux pro Raumwinkel, d. h. pro Richtung im Raum; (Einheit $W/sr$, Summation bzw. Integration über alle Flächenelemente);

$$\begin{aligned} \mathrm{d}I &:= L_o(x,\omega_o)\cos\theta_o\,\mathrm{d}A\,, \\ I(\omega_o) &:= \int_A L_o(x,\omega_o)\cos\theta_o\,\mathrm{d}A \end{aligned} \tag{10.4}$$

- nochmalige Integration liefert dann den Radiant Flux:

$$\begin{aligned} \Phi_o &= \int_{H^2} I(\omega_o)\,\mathrm{d}\omega_o = \int_A B(x)\,\mathrm{d}A = \int_A \int_{H^2} L_o(x,\omega_o)\cos\theta_o\,\mathrm{d}\omega_o\,\mathrm{d}A\,, \\ \Phi_i &= \int_A E(x)\,\mathrm{d}A = \int_A \int_{H^2} L_i(x,\omega_i)\cos\theta_i\,\mathrm{d}\omega_i\,\mathrm{d}A \end{aligned} \tag{10.5}$$

  im Falle vollständiger Reflexion (keine Transparenz/Brechung, keine Absorption) gilt für eine Oberfläche $A$ $\Phi_o = \Phi_i$;

Radiance hat zwei wichtige Eigenschaften: Sie ist konstant entlang eines Strahls, solange dieser auf keine Oberfläche trifft, und sie ist die ausschlaggebende Größe für die Antwort eines lichtempfindlichen Sensors (Kameras, Auge).

Schließlich hängen alle genannte Größen von der *Wellenlänge* des Lichts ab. Theoretisch tritt also ein weiteres Integral über die Wellenlänge hinzu, praktisch beschränkt man sich auf den üblichen dreidimensionalen Vektor $(R, G, B)$ für die Grundfarben Rot, Grün und Blau.

### 10.2.2 BRDFs

Entscheidend für die Modellierung der optischen Erscheinung von Flächen ist die Beschreibung der *Reflexion*. Diese hängt ab von der Richtung des einfallenden und des ausgehenden Lichts, der Wellenlänge des Lichts sowie dem jeweiligen Punkt auf der Oberfläche. Wichtigstes Beschreibungsmittel in diesem Zusammenhang sind die *Bidirectional Reflection Distribution Functions (BRDF)* $f_r(x, \omega_i \to \omega_r)$, die die Abhängigkeit des ausgehenden (reflektierten) Lichts von der einfallenden Lichtstärke angeben:

$$\mathrm{d}L_r = f_r(x,\omega_i \to \omega_r)\,\mathrm{d}E = f_r(x,\omega_i \to \omega_r)L_i(x,\omega_i)\cos\theta_i\,\mathrm{d}\omega_i\,. \tag{10.6}$$

Man beachte, dass die reflektierte Radiance mit der eingehenden Irradiance in Bezug gesetzt wird, nicht mit der eingehenden Radiance (dafür gibt's auch eine schlaue Begründung, siehe z. B. Foley). Im Allgemeinen sind BRDFs anisotrop, ein Drehen der Oberfläche kann (bei unveränderten Ein- und Ausgangsrichtungen) zu einer veränderten reflektierten Lichtmenge führen.

Aus (10.6) kann die sog. *Reflectance-Gleichung* abgeleitet werden:

$$L_r(x,\omega_r) = \int_{H^2} f_r(x,\omega_i \to \omega_r)L_i(x,\omega_i)\cos\theta_i\,\mathrm{d}\omega_i\,. \tag{10.7}$$

BRDFs stellen eine sehr allgemeine Beschreibungsmöglichkeit der Reflexion dar. Aus Komplexitätsgründen werden in der Praxis einfache Reflexionsmodelle für die unterschiedlichen Klassen von BRDFs verwendet, die vor allem die Zahl der Parameter reduzieren (siehe auch Abb. 10.2):

- Bei der *idealen spiegelnden Reflexion* wird ein einfallender Strahl in genau eine Richtung reflektiert, gemäß dem Reflexionsgesetz von Snellius. Die Modellierung der BRDFs erfolgt mit Hilfe Diracscher $\delta$-Funktionen.
- Bei der *praktischen spiegelnden Reflexion* wird Licht einer Verteilung gehorchend reflektiert, die sich stark um die ideale Reflexionsrichtung konzentriert.

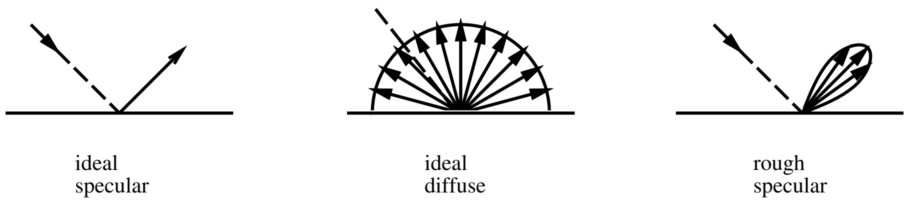


Abbildung 10.2: Verschiedene Typen von BRDFs

- *Lambertsche* oder *diffuse* Reflexion reflektiert einfallendes Licht in alle Richtungen gleichmäßig, die zugehörige Verteilung ist also eine Gleichverteilung.

Allgemeine Reflexions- bzw. Beleuchtungsmodelle kombinieren oft alle Ansätze (vgl. die bekannten Modelle von Torrance-Sparrow, Blinn, Cook-Torrance und Phong). Hiermit wird aber in jedem Fall nur beschrieben, wie Licht einfällt und was mit einfallendem Licht geschieht. Zur Beantwortung der Frage, wie das jeweils einfallende Licht zustande kommt, muss ein weiterer Schritt getan werden.

Transparenz und Beugung können leicht eingebaut werden, indem man als mögliche „Reflexionsrichtungen“ auch die untere bzw. innere Hemisphäre zulässt. Im folgenden werden alle Integrale der Einfachheit halber bzgl. $H^2$ formuliert.

### 10.2.3 Die Rendering-Gleichung

Gleichung (10.7) liefert den (lokalen) Zusammenhang zwischen einfallendem und reflektiertem Licht. Darauf aufbauend, muss jetzt eine Verteilung des einfallenden Lichts, ein *Beleuchtungsmodell* hergeleitet werden.

Wir betrachten zunächst den *lokalen* Fall: Eine Punktlichtquelle in $x_s$ habe die Intensität $I_s$. Die Radiance $L_i$ in einem Punkt $x$ der Szene bestimmt sich dann gemäß

$$L_i(x,\omega) \;=\; \begin{cases} I_s \cdot |x - x_s|^{-2} \text{ für } \omega = \omega_s := x - x_s\,, \\ 0 \text{ sonst }\;. \end{cases} \tag{10.8}$$

Das Integral (10.7) degeneriert dann zu dem Ausdruck (formal über Dirac-Stoß)

$$L_r(x,\omega_r) \;=\; \frac{I_s}{|x - x_s|^2} f_r(x, \omega_s \to \omega_r) \cos\theta_s\,, \tag{10.9}$$

bzw. zu einer Summe über $n$ solche Terme bei $n$ Punktlichtquellen. Dies ist ein denkbar primitives Beleuchtungsmodell, ohne indirekte Beleuchtung und ohne Schatten. Es berücksichtigt also die Umgebung des Punktes $x$ nicht und wird deshalb *lokal* genannt.

*Globale* Beleuchtung ist aber mehr als die lokale Betrachtung in allen Punkten der Szene. Das Wechselspiel der einzelnen Flächen muss korrekt wiedergegeben werden. Dieses Wechselspiel ist nichts anderes als Energietransport: Lichtquellen emittieren Energie, die in der Szene reflektiert, gebrochen oder absorbiert wird. Berechnet werden muss nun die Lichtmenge, die schließlich das Auge oder die Kamera erreicht. Dazu muss für jedes Bildpixel die Radiance über die entsprechenden sichtbaren Flächen integriert werden. Der resultierende Radiant Flux definiert dann Helligkeit und Farbe des Pixels. Die Radiance, die einen Punkt $x$ in Richtung $\omega_o$ verlässt, setzt sich aus (selbst-) emittierter und reflektierter Radiance zusammen:

$$L_o(x,\omega_o) \;=\; L_e(x,\omega_o) \;+\; L_r(x,\omega_o)\,. \tag{10.10}$$

Mit (10.7) erhält man daraus die *Rendering-Gleichung*:

$$L_o(x,\omega_o) \;=\; L_e(x,\omega_o) \;+\; \int_{H^2} f_r(x,\omega_i \to \omega_o) L_i(x,\omega_i) \cos\theta_i \,\mathrm{d}\omega_i\,. \tag{10.11}$$

Die Geometrie der Szene, die BRDFs der Oberflächen und die Lichtquellen (interpretiert als rein selbst-emittierende Flächen) beschreiben die Lichtverteilung vollständig.

Eine andere Schreibweise der Rendering-Gleichung benutzt die *Sichtbarkeitsfunktion* $V(x,y)$,

$$V(x,y) \;:=\; \begin{cases} 1 & (x \text{ sieht } y)\,, \\ 0 & (\text{sonst})\,. \end{cases} \tag{10.12}$$

Dabei seien $x \in A_x$ und $y \in A_y$ zwei Punkte auf zwei Oberflächen $A_x$ und $A_y$ der Szene. Mit $V$ gilt

$$L_i(x,\omega_i) \;=\; L_o(y,\omega_i)\,V(x,y)\,. \tag{10.13}$$

Um das Integrationsgebiet in (10.11) zu ändern, setzen wir $\mathrm{d}\omega$ in Bezug zu dem Flächenstück $\mathrm{d}A_y$, wo das Licht herkommt:

$$\mathrm{d}\omega_i \;=\; \frac{\cos\theta_y\,\mathrm{d}A_y}{|x-y|^2}\,. \tag{10.14}$$

Mit der Definition

$$G(x,y) \;:=\; V(x,y)\cdot\frac{\cos\theta_i\cos\theta_y}{|x-y|^2} \tag{10.15}$$

erhält man dan die zweite Form der Rendering-Gleichung:

$$L_o(x,\omega_o) \;=\; L_e(x,\omega_o) \;+\; \int_{A_y} f_r(x,\omega_i\to\omega_o)L_o(y,\omega_i)G(x,y)\,\mathrm{d}A_y\,. \tag{10.16}$$

Mit dem Integraloperator

$$(Tf)(x,\omega_o) \;:=\; \int_{A_y} f_r(x,\omega_i\to\omega_o)f(y,\omega_i)G(x,y)\,\mathrm{d}A_y \tag{10.17}$$

vereinfacht sich (10.16) zu

$$L_o \;=\; L_e \;+\; TL_o \qquad \text{bzw. kurz} \qquad L \;=\; L_e \;+\; TL\,. \tag{10.18}$$

Der *Lichttransportoperator* $T$ wandelt also die Radiance auf $A_y$ um in die Radiance auf $A_x$ (nach einer Reflexion). Wendet man (10.18) rekursiv an, so erhält man

$$L \;=\; \sum_{k=0}^{\infty} T^k L_e\,, \tag{10.19}$$

wobei $L_e$ für emittiertes Licht, $TL_e$ für direkte Beleuchtung und $T^kL_e$, $k > 1$, für indirekte Beleuchtung nach $k-1$ Reflexionen stehen.

Die Integralgleichungen (10.11) bzw. (10.16) sind zur Bestimmung der globalen Beleuchtung zu lösen.

## 10.3 Techniken zur globalen Beleuchtung

### 10.3.1 Klassifikation

Alle Verfahren, die das globale Beleuchtungsproblem lösen wollen, können anhand ihrer Mächtigkeit zur Berücksichtigung verschiedener Typen von Licht-Interaktion (charakterisiert mittels einer Folge von Reflexionen auf dem Weg von der Lichtquelle zum Auge) kategorisiert werden. Mit den

Bezeichnungen $L$ für Lichtquelle, $E$ für das Auge, $D$ für eine diffuse Reflexion und $S$ für eine spiegelnde Reflexion muss ein optimales Verfahren alle Folgen vom Typ des regulären Ausdrucks

$$L\,(\,D\,|\,S\,)^*\,E \tag{10.20}$$

wiedergeben können (siehe Abb. 10.3). Da man auch bei Transparenz zwischen „spiegelnder“ Transparenz (*transparent*, z. B. Glas) und diffuser Transparenz (*translucent*, z. B. Milchglas) un-

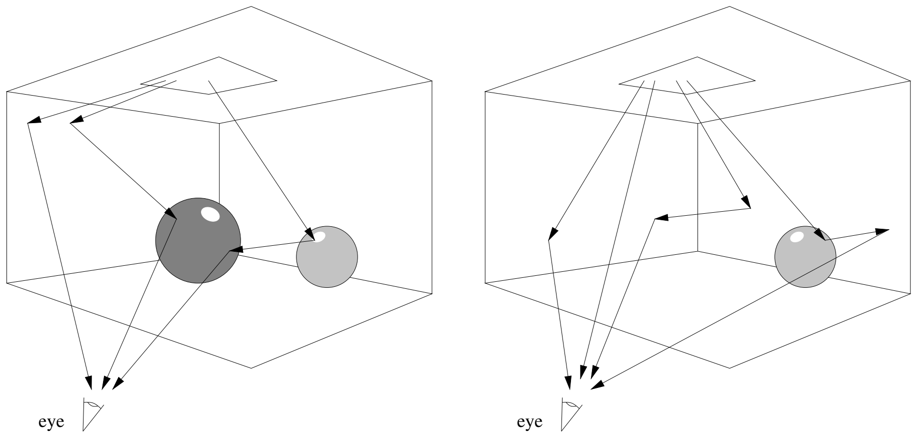


Abbildung 10.3: Unterschiedliche Lichtpfade: $LDE$, $LDSE$, $LSSE$ und $LDE$, $LE$, $LDDE$, $LSDE$ (von links nach rechts)

terscheiden kann, sind mit (10.20) wirklich alle Fälle erschlagen. Bezeichnet ferner $X$ einen Punkt der Szene, so lassen sich für ihn folgende Beleuchtungstypen unterscheiden:

$$\begin{array}{rl} L\,X & \text{direkte Beleuchtung}\,, \\ L\,(\,D\,|\,S\,)^+\,X & \text{indirekte Beleuchtung}\,, \\ L\,D^+\,X & \text{rein diffuse indirekte Beleuchtung}\,, \\ L\,S^+X & \text{rein spiegelnde indirekte Beleuchtung}\,. \end{array} \tag{10.21}$$

Der letzte Fall ist besonders „ätzend“ und wird deshalb *Caustics* genannt.

### 10.3.2 Ray Tracing

*Ray Tracing* ist das älteste und wohl einfachste Verfahren zur globalen Beleuchtung. Schattenwurf sowie ideale spiegelnde Reflexion und Brechung werden erfasst, diffuse Beleuchtung dagegen nicht.

Grundlage des Algorithmus' sind einzelne Lichtstrahlen, die von einem definierten Startpunkt ausgehen und beim ersten Treffer mit einem Objekt der Szene oder ggfs. nie enden. Energie wird in Form von Radiance transportiert.

Der rechentechnische Hauptaufwand entsteht beim Ray Tracing bei der Suche nach dem Endpunkt der Strahlen. Hierzu müssen Schnittpunkte eines Strahls mit allen Objekten der Szene berechnet werden. Je nachdem, wie die Szene geometrische modelliert ist, ist das mit mehr oder weniger Aufwand verbunden. In jedem Fall dominiert die Schnittpunktberechnung den Rechenaufwand, und der Effizienzsteigerung über Partitionierung oder hierarchisches Clustering kommt eine große Bedeutung zu.

Zum Algorithmus: Vom Betrachter aus wird durch jedes Bildpixel ein Strahl in die Szene geschossen (*Primärstrahlen*). Trifft ein Strahl kein Objekt, erhält das Pixel bspw. die Hintergrundfarbe. Andernfalls starten vom nächstgelegenen Schnittpunkt $x$ dreierlei Typen von *Sekundärstrahlen*:

der perfekt reflektierte Strahl (bei nicht völlig mattem Material), der perfekt gebrochene Strahl (bei lichtdurchlässigem Material) sowie die sog. Schattenstrahlen zu allen Lichtquellen. Jede Lichtquelle, die vom entsprechenden Schattenstrahl ohne Hindernis (Objekt in der Szene) erreicht wird, beleuchtet $x$ direkt. Die von $x$ zum Beobachter ausgehende Radiance wird dann rekursiv ermittelt aus den von allen Sekundärstrahlen eingehenden Radiance-Werten, den jeweiligen Richtungen und der BRDF in $x$ gemäß dem lokalen Beleuchtungsmodell nach (10.9). Zum Abbruch der Rekursion muss eine maximale Rekursionstiefe oder eine Mindest-Radiance festgelegt werden. Die fehlende Berücksichtigung diffuser Reflexion wird schließlich durch einen konstant Term für ambientes Licht modelliert.

Gemäß der Klassifizierung von (10.20) gibt Ray Tracing alle Pfade vom Typ

$$L\,D\,S^*\,E \tag{10.22}$$

wieder. Das $D$ in (10.22) kommt daher, dass die Schattenstrahlen von der Lichtquelle aus gesehen beliebige und von keiner „Einfallsrichtung“ abhängende Richtungen annehmen können. Pfade vom Typ $L\,S^*\,E$ können nicht dargestellt werden, da Lichtquellen nicht als Objekte behandelt werden und folglich nur von Schattenstrahlen getroffen werden können.

Die Bestimmung der zum Beobachter ausgehenden Radiance in einem Schnittpunkt $x$ erfolgt wie gesagt auf der Grundlage eines lokalen Beleuchtungsmodells (10.9). Beim ursprünglichen Ray Tracing wurde die eintreffende Radiance einfach mit einem materialabhängigen Reflexions- bzw. Brechungs-Koeffizienten multipliziert, unabhängig von eintreffender oder ausgehender Richtung. Analog zur Rendering-Gleichung in der Form (10.19) können wir die Funktion von Ray Tracing jedoch auch in seiner allgemeinen Form beschreiben als

$$L \;=\; L_0 \;+\; \sum_{k=1}^{\infty} T_0^k L_e\,, \tag{10.23}$$

wobei $L_0$ den ambienten Term bezeichnet. Der Term $L_e$ entfällt, weil Punktlichtquellen nicht als emittierende Objekte behandelt werden. Im Gegensatz zu $T_0$ ist $T$ kein Integral-, sondern ein einfacher Summenoperator, der über die drei Terme direkte Beleuchtung, reflektiertes Licht und gebrochenes Licht summiert. Aus der Sicht der zu lösenden Rendering-Gleichung betrachtet heißt das, dass wir anstelle des Integrals über *alle* eingehenden Richtungen ein paar fest vorgegebene Richtungen samplen, nämlich die perfekt reflektierte, die perfekt gebrochene und die zu den Lichtquellen.

### 10.3.3 Path Tracing

*Path Tracing* (auch *Monte Carlo Ray Tracing* oder *Monte Carlo Path Tracing* genannt) wurde 1986 von Kajiya vorgestellt. Die Idee ist, die Rendering-Gleichung (10.11) direkt anzugehen und mittels Monte-Carlo-Integration zu lösen. Wie beim Ray Tracing ersetzen wir also das Integral über alle Richtungen durch Samples in eine endliche Zahl von Richtungen. Die Richtungen zu den Lichtquellen bleiben erhalten (mittels zufälliger Strahlen kann man Punktlichtquellen nicht treffen, obwohl sie sehr wichtig sind), der perfekt reflektierte und der perfekt gebrochene Strahl werden jedoch durch einen einzigen Strahl ersetzt, dessen Richtung zufällig bestimmt wird (daher der Namensteil *Monte Carlo*). Dadurch ergibt sich eine lineare Rekursion, ein Pfad von Strahlen (daher der Namensteil *Path*). Die Rekursionstiefe kann fest vorgegeben, adaptiv gewählt oder durch *Russisches Roulette* bestimmt werden, wobei jedesmal zufällig ermittelt wird, ob der Pfad enden oder weitergeführt werden soll (bis zur Maximaltiefe – eben wie beim Russischen Roulette).

Anstelle der Gleichverteilung über alle Richtungen kann auch eine auf der BRDF basierende gewichtete Verteilung benutzt werden, die die aus der BRDF resultierenden wichtigeren Richtungen bei der Auswahl bevorzugt. Mittels

$$\rho(x,\omega_o) \;:=\; \int_{H^2} f_r(x,\omega_i \to \omega_o)\cos\theta_i \,\mathrm{d}\omega_i \tag{10.24}$$

wird

$$\frac{f_r(x, \omega_i \to \omega_o) \cos\theta_i}{\rho(x, \omega_o)} \tag{10.25}$$

zur Verteilung auf $H^2$, als Schätzer für den Integralwert in (10.11) aus einer gemäß dieser Verteilung ermittelten Richtung $\omega_i$ ergibt sich dann

$$\rho(x, \omega_o) \cdot L_i(x, \omega_i)\,. \tag{10.26}$$

Die Einführung einer Verteilung wie in (10.25) ist wichtig, da nur über eine Bevorzugung der Richtungen „in der Nähe" des perfekt reflektierten Strahls wirksam zwischen diffuser und spiegelnder Reflexion unterschieden werden kann.

Da man beim Path Tracing nur eine einzige Richtung als Sample für die Bestimmung des Integralwerts benutzt, entsteht ein erhebliches Rauschen. Weil die lineare (Pfad-) Rekursion aber aus Effizienzgründen unangetastet bleiben soll, schießt man anstelle eines einzigen Strahls jetzt mehrere Strahlen durch ein Pixel in die Szene und mittelt das Resultat.

Was die Klassifizierung von (10.20) angeht, so kann Path Tracing theoretisch alle gewünschten Pfade

$$L\,D\,(\,D\,|\,S\,)^*\,E \tag{10.27}$$

wiedergeben. Das erste $D$ muss aus denselben Gründen wie beim Ray Tracing aufscheinen. Ansonsten exisitert für Path Tracing aufgrund der zufälligen Richtungswahl der Unterschied zwischen spiegelnder und diffuser Reflexion nicht (bei der Wahl einer Verteilung wie in (10.25) vorgeschlagen werden die Richtungen des perfekt reflektierten bzw. perfekt gebrochenen Strahls nur wahrscheinlicher). Trotzdem sind zwei wesentliche Nachteile festzuhalten. Zunächst ist Path Tracing extrem teuer, da bei typischen Szenen zwischen 25 und 100 Strahlen pro Pixel abgefeuert werden müssen, wenn man ein zufriedenstellendes Ergebnis erzielen will. Zum zweiten bleiben Probleme bei der Wiedergabe von Caustics. Gemäß (10.21) waren diese charakterisiert als $L\,S^+\,X$ bzw. $L\,S^+\,D\,E$. Betrachten wir das Beispiel eines Spiegels, der eine Oberfläche indirekt beleuchtet. Das direkte Licht von einer Lichtquelle zur Oberfläche wird über die Schattenstrahlen separat berücksichtigt, das Licht, das über den Spiegel ankommt (der jetzt gewissermaßen als sekundäre Lichtquelle fungiert), jedoch nicht. Hier muss König Zufall die Berücksichtigung im Rahmen des Monte-Carlo-Prozesses sicherstellen, was aber nur mit einer bestimmten Wahrscheinlichkeit geschieht.

### 10.3.4 Radiosity

Radiosity ist sowohl der Name der in (10.3) eingeführten Größe als auch eines Verfahrens zur globalen Beleuchtung. Dieses wure 1984 von Goral et al. vorgestellt und wählt einen grundlegend anderen Ansatz als den strahlorientierten.

Radiosity verzichtet auf Spiegelungen und geht von ausschließlich diffusen Flächen aus. In diesem Fall hängen alle BRDFs nicht von der Richtung des einfallenden oder des austretenden Lichts ab. Somit kann man aus der Rendering-Gleichung in der Form (10.16) die BRDF vor das Integral ziehen:

$$\begin{aligned}
L_o(x,\omega_o) &= L_e(x,\omega_o) + \int_{A_y} f_r(x,\omega_i \to \omega_o) L_o(y,\omega_i) G(x,y)\,\mathrm{d}A_y \\
&= L_e(x,\omega_o) + \int_{A_y} f_r(x) L_o(y,\omega_i) G(x,y)\,\mathrm{d}A_y \\
&= L_e(x,\omega_o) + f_r(x) \cdot \int_{A_y} L_o(y,\omega_i) G(x,y)\,\mathrm{d}A_y\,.
\end{aligned} \tag{10.28}$$

Weil die ausgehende Radiance einer diffusen Oberfläche keine Richtungsabhängigkeit aufweist, kann man (10.28) auch mittels $B(x)$ formulieren und erhält die sogenannte *Radiosity-Gleichung*

$$B(x) \;=\; B_e(x) \;+\; \frac{\rho(x)}{\pi} \cdot \int_{A_y} B(y) G(x,y) \,\mathrm{d}A_y \,. \tag{10.29}$$

Hierbei gilt für die *Reflectance* $\rho(x)$

$$\rho(x) \;:=\; \int_{H^2} f_r(x) \cos\theta_i \,\mathrm{d}\omega_i \;=\; f_r(x) \cdot \int_{H^2} \cos\theta_i \,\mathrm{d}\omega_i \;=\; \pi \cdot f_r(x) \tag{10.30}$$

(vgl. auch (10.24)).

Radiosity ist ansichtsunabhängig, die globale Beleuchtung wird also einmal für die ganze Szene berechnet und nicht nur für eine bestimmte Betrachterposition. Zur Lösung wird die Radiosity-Gleichung (10.30) im Finit-Element-Sinne diskretisiert. Man überzieht alle Oberflächen der Szene mit einem Netz von ebenen Oberflächenstücken $f_i$ mit Flächeninhalt $A_i$, $1 \leq i \leq n$, wobei nicht mehr zwischen Lichtquellen und eigentlichen Objekten unterschieden wird. Zur Beschreibung des Lichttransports von $f_i$ nach $f_j$ werden *Formfaktoren* $F_{ij}$ eingeführt. $F_{ij}$ gibt dabei den Anteil des Radiant Flux $A_i \cdot B_i$ (der $f_i$ verlassende Flux), der bei $f_j$ ankommt. Offensichtlich gilt also $0 \leq F_{ij} \leq 1$. De facto entsprechen die Formfaktoren im Wesentlichen der Funktion $G$ aus (10.15). Somit wird (10.29) zu

$$B_i \;=\; B_{e,i} \;+\; \rho_i \cdot \sum_{j=1}^{n} \frac{B_j \cdot A_j \cdot F_{ji}}{A_i} \qquad \forall\, 1 \leq i \leq n \,. \tag{10.31}$$

Der Faktor $\pi^{-1}$ ist dabei in die Formfaktoren gewandert. Aufgrund der Beziehung

$$A_i \cdot F_{ij} \;=\; A_j \cdot F_{ji} \tag{10.32}$$

läßt sich (10.31) vereinfachen, und man erhält die *diskrete Radiosity-Gleichung*:

$$B_i \;=\; B_{e,i} \;+\; \rho_i \cdot \sum_{j=1}^{n} B_j \cdot F_{ij} \qquad \forall\, 1 \leq i \leq n \,. \tag{10.33}$$

Aus (10.33) ergeben sich die zwei wesentlichen Hauptaufgaben bei Radiosity-Verfahren: Erstens müssen die Formfaktoren berechnet werden, zweitens muss das entstehende System linearer Gleichungen numerisch gelöst werden.

Radiosity betrachtet nur diffuse Oberflächen. Deshalb können nur Lichtpfade vom Typ

$$L\, D^*\, E \tag{10.34}$$

simuliert werden. Der direkte Pfad $L\,E$ ist möglich, da Lichtquellen und Oberflächen gleich behandelt werden. Im Gegensatz zum Ray Tracing besteht das Ergebnis nicht aus einem Rasterbild für eine bestimmte Ansicht der Szene, sondern aus den Radiosity-Werten aller (diskreter) Oberflächen $f_i$. Nach der zeitintensiven Radiosity-Berechnung kann deshalb ein (Hardware-beschleunigtes) $z$-Buffer Verfahren zur Generierung bestimmter Ansichten benutzt werden, was beispielsweise Echtzeit-Flüge durch die Szene gestattet.

Aber auch Ray Tracing kann zur Ansichtsgenerierung verwendet werden. Eine solche Hintereinanderschaltung von Ray Tracing und Radiosity ermöglicht Lichtpfade vom Typ $L\,D^+\,S^*\,E$. Dies ist besser als jede Einzellösung, Caustics gehen aber noch immer nicht. Deshalb wurden noch weitere Verallgemeinerungen entwickelt (*extended form factors*), die auch Caustics integrieren sollen. Das bleibende Problem ist aber, dass Radiosity immer mit ebenen Flächenstücken arbeitet, womit viele Caustics nicht erschlagen werden können.

### 10.3.5 Light Ray Tracing

*Light Ray Tracing* oder *Backward Ray Tracing* geht auf Arvo 1986 zurück und wurde speziell für die Simulation indirekter Beleuchtung entwickelt.

In einem ersten Path Tracing Schritt werden Strahlen von den Lichtquellen in die Szene geschossen. Trifft ein solcher Strahl auf eine zumindest anteilig diffuse Oberfläche, wird seine Radiance nach der Reflexion reduziert, und der Differenzbetrag an Energie wird der Oberfläche gutgeschrieben und in einer *Illumination Map* gespeichert. Das Konzept ist ganz ähnlich wie bei Texture Maps. Im zweiten Schritt wird dan ein gewöhnliches Ray Tracing durchgeführt, wobei die Illumination Maps der Oberflächen in die Beleuchtungsberechnung einfließen.

Light Ray Tracing ist das erste der bisher diskutierten Verfahren, das mit Caustics umgehen kann. Allerdings ist der Aufwand exterm hoch, da die Auflösung der Illumination Maps hoch und die Anzahl der Strahlen im ersten Durchlauf folglich sehr groß sein muss. Außerdem limitiert die Notwendigkeit der Parametrisierung der Oberflächen über die Illumination Map die Gestalt der Oberflächen. Schließlich liegt über die Illumination Map die Energie an einer Stelle vor, aber nicht die Richtung, aus der das Licht einfiel, was für den zweiten Schritt (die Berechnung der ausgehenden Radiance) von Nachteil sein kann.

### 10.3.6 Monte Carlo Ray Tracing mit Photon Maps

*Monte Carlo Ray Tracing mit Photon Maps* oder kurz *Photon Tracing* wurde 1995/1996 von Jensen vorgestellt. Es stellt die derzeit vielleicht beste Methode zur globalen Beleuchtung dar, und zwar in folgender Hinsicht:

- **Qualität**: Grundbaustein ist Path Tracing, eine gute Lösung des Beleuchtungsproblems. Zur Wiedergabe von Caustics wird ein Light Ray Tracing Schritt hinzugefügt. Gespeichert wird die Information aus diesem Schritt in *Photon Maps*, ähnlich den *Illumination Maps*.
- **Flexibilität**: Photon Tracing baut nur auf Ray Tracing Techniken auf. Weder eine Beschränkung auf polygonale Objekte (wie bei Radiositiy) noch eine Parametrisierung der Oberflächen (wie bei den Illumination Maps) sind erforderlich. Sogar fraktale Objekte sind verwendbar.
- **Geschwindigkeit**: Der zweite Schritt benutzt einen optimierten Path Tracer, dessen Laufzeit durch die Information aus der Photon Map reduziert werden kann.
- **Parallelisierbarkeit**: Wie alle Ray Tracing Verfahren kann auch Photon Tracing einfach und effizient parallelisiert werden.

Ähnlich wie beim Light Ray Tracing besteht der erste Schritt aus einem Path Tracing von den Lichtquellen aus. *Photonen*, d. h. energiebehaftete Partikel, werden von den Lichtquellen aus in die Szene geschossen. Trifft ein Photon eine Oberfläche, werden sowohl seine Energie als auch seine Einfallsrichtung in die Photon Map eingetragen. Dann wird das Photon zufällig reflektiert, oder es endet (wird absorbiert). Nach Abschluss dieser Phase gibt die Photon Map eine Approximation der Lichtverhältnisse in der Szene: Je höher die Photonendichte, desto mehr Licht. Man beachte den entscheidenden Unterschied zwischen Illumination Maps und der Photon Map: Erstere sind lokal, assoziiert zu jeder Oberfläche und benötigen eine Parametrisierung, die ein zweidimensionales Array auf diese Fläche abbildet. Letztere ist global und lebt in dreidimensionalen Weltkoordinaten. Außerdem werden sowohl die Energie als auch die Einfallsrichtung gespeichert. Somit ist die Photon Map eine riesige gestreute Datenmenge in 3 D.

Im zweiten Schritt wird ein optimierter Path Tracer benutzt. Nach nur wenigen Rekursionsstufen wird der Pfad abgebrochen, und die Beleuchtungsverhältnisse werden angenähert durch Schätzung der Dichteverteilung der Photonen in der Umgebung des betreffenden Punkts. Damit kann die Photon Map auch zur Simulation on Caustics benutzt werden.

**<u>Erster Pass:</u>**

Zunächst muss die Aussendung der Photonen festgelegt werden. Dabei werden die Lichtquellen zufällig ausgewählt, unter Berücksichtigung ihrer Leistung. Bei Punktlichtquellen ist der Startpunkt trivial, bei anderen müssen Größe und sonstige Charakteristik der Lichtquelle einfließen. Die Strahlrichtung ist völlig gleichverteilt bei Punktlichtquellen, bei realistischeren Lampen etc. muss jedoch die Abstrahlcharakteristik der Quelle berücksichtigt werden. Die Leistung aller Photonen ist konstant, manchaml lässt man allerdings Variationen in der RGB-Zusammensetzung zu.

Die „Weiterreise" des Photons kann entweder enden (völlige Absorption, Abbruch des Pfades), oder sie geht in einer zufällig bestimmten Richtung weiter, wobei eine BRDF-basierte Verteilung zugrunde gelegt wird. Die Energie des Photons bleibt je nach Modell entweder erhalten, oder sie wird um einen konstanten bzw. richtungsabhängigen Anteil reduziert.

Mit der Photon Map will man zwei Ziele erreichen: erstens die Lichtverhältnisse approximieren, zweitens Caustics simulieren. Für Ersteres genügt eine geringe Genauigkeit, bei Caustics ist dagegen Präzision angesagt. Deshalb implementiert man i. d. R. zwei Maps: die *Global Photon Map* für die indirekte Beleuchtung und die *Caustics Photon Map* für Caustics. In letzterer werden alle Photonen aufgesammelt, die bislang ausschließlich auf spiegelnde Oberflächen getroffen waren, alle anderen werden in die Global Photon Map eingetragen. Um die erhöhte Genauigkeit für Caustics zu erreichen, müssen zudem in Gebiete mit spiegelnden Flächen gezielt mehr Photonen geschossen werden.

**<u>Zweiter Pass:</u>**

Der zweite Schritt ist ein Path Tracer. An zwei Hebeln setzt man zur Verbesserung an. Zunächst wird zur Beschleunigung in jedem Punkt, in dem die ausgehende Radiance zu bestimmen ist, entschieden, ob man sie möglichst genau berechnen oder approximieren will. Genähert wird üblicherweise, sobald der Pfad schon eine diffuse Reflexion hinter sich hat. Außerdem gibt es eine Höchstzahl spiegelnder Reflexionen, ab der ebenfalls genähert wird. „Nähern" heißt in diesem Fall, dass man die Information der Global Photon Map benutzt, „genau berechnen" heißt den Pfad weiterverfolgen. Der zweite Hebel dient zur Qualitätsverbesserung. An allen Punkten, an denen man die Radiance genau haben möchte, addiert man die sich aus der Caustics Photon Map ergebende Radiance dazu.

Zwei Bemerkungen zum Schluss:

- An mehreren Stellen wurde zwischen diffuser und spiegelnder Reflexion unterschieden. Für einen normalen Path Tracer gibt es diese Unterscheidung (explizit) ja gar nicht, beim Photon Tracing wird sie jedoch protokolliert.
- Zur Speicherung der Photonen im dreidimensionalen Raum sowie zur schnellen Auffindung von Nachbarn sind spezielle Datenstrukturen erforderlich, etwa kd-Bäume, die Koordinaten-Bisektion realisieren (siehe Abb. 10.4).

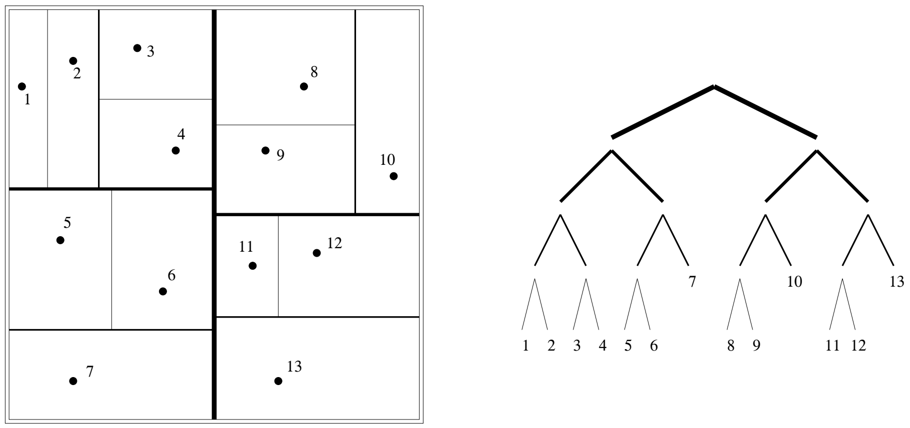


Abbildung 10.4: Raumpartitionierung mit kd-Bäumen

# Kapitel 11

# Raumbildsehen

Bei den meisten im Kapitel zur Visualisierung angesprochenen Techniken bleibt die Problematik, dass sich dreidimensionale Daten nur schwer direkt darstellen lassen. Auch wenn ein aufwendiges Rendering den Raumbildeindruck unterstützen kann, auch wenn mittels Iso- oder Schnittflächen nahezu beliebige Ausschnitte ausgewählt werden können, die Betrachtung bleibt im Wesentlichen zweidimensional. Echt dreidimensionale Phänomene lassen sich aber nur mittels stereoskopischem Raumbildsehen voll erfassen. Kann man etwa virtuell ein durchströmtes poröses Medium durchwandern und dabei geometrische Details, Wirbelstrukturen o. Ä. plastisch wahrnehmen, so erhöht sich der für den Betrachter verwertbare Informationsgehalt der Visualisierung beträchtlich. Dasselbe gilt für die Computergraphik ganz allgemein: Raumbildsehen ist einer der entscheidenden Schritte auf dem Weg zu echtem Photorealismus, ohne echte Dreidimensionalität fehlt etwas.

Stereoskopie erfordert, analog zum natürlichen Sehvorgang, zwei getrennte Bilder für das linke und das rechte Auge. Durch die stereoskopische Überlagerung entsteht dann der räumliche Eindruck. Zur Erzeugung der beiden getrennten Bilder gibt es eine Vielzahl von Lösungsansätzen, die grob in zwei Klassen eingeteilt werden können. Bei den sogenannten *feld-sequentiellen* oder *zeit-multiplexen* Verfahren werden die Bilder für das linke und das rechte Auge abwechselnd auf dem Bildschirm dargestellt. Damit jedes Auge nun auch tatsächlich nur „seine“ Bilder sieht, muss ein synchronisiertes Verschlusssystem eingesetzt werden, das jeweils ein Auge abdeckt. Man unterscheidet hier zwischen *aktiven Systemen*, bei denen der Verschlussmechanismus in eine Brille integriert ist (*Shutter-Brille*, teuer), und *passiven Systemen*, bei denen der Verschluss als abwechselnd polarisierender Filter vor dem Bildschirm realisiert ist, wobei der Betrachter dann eine Brille mit entsprechenden Polarisierungsfiltern (billig) tragen muss.

Bei den sogenannten *zeit-parallelen* Verfahren werden dagegen die beiden Ansichten nicht abwechselnd, sondern gleichzeitig gezeigt. Dies erfordert natürlich optische Sehhilfen für die korrekte Trennung der beiden Bilder. Bei der Methode der *Anaglyphen*, einem seit den fünfziger Jahren verbreitete Technik zur Erzeugung von Raumbildeindruck, werden beide Ansichten in Komplementärfarben (Rot und Grün oder Rot und Cyan) erzeugt. Farbdarstellungen sind nicht (bzw. nur sehr eingeschränkt) möglich. Im Falle eines hellen Hintergrunds werden Überlagerungspunkte meist schwarz, vor einem dunklen Hintergrund gelb gezeichnet. Der Betrachter trägt dabei eine (billige) Brille mit zwei entsprechenden Farbfiltern (Rot in der Regel links), die im Idealfall dafür sorgt, dass das linke Auge nur das grüne Bild, das rechte Auge nur das rote Bild sieht. Die Versetzung der beiden Bilder auf dem Bildschirm kommt schließlich durch die zwei unterschiedlichen Projektionszentren für Rot und Grün zustande (vgl. Abb. 11.1). Allerdings ist das Verfahren mit zahlreichen Nachteilen behaftet: Der Betrachter ist an eine feste Position gebunden, es kommt zu Restbildern bei nicht perfekter Überlappung, zu Trübungen des Raumbildeindrucks bei nicht exakter Positionierung sowie zu Aliasing-Effekten.

Eine alternative zeit-parallele Möglichkeit ist die Aufeinanderprojektion zweier unterschiedlich

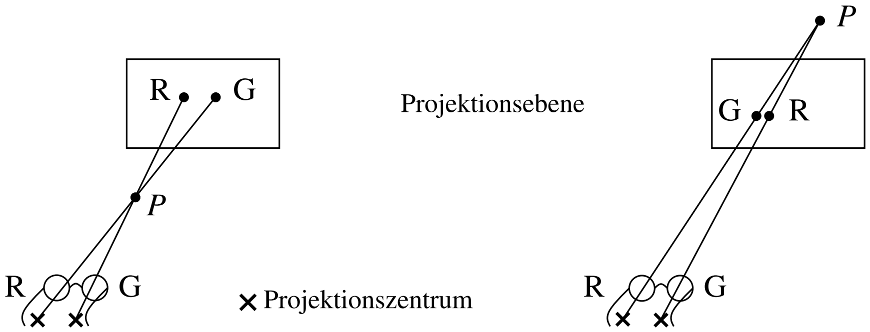


Abbildung 11.1: Erzeugung des roten und des grünen Bildes im Fall von Punkten $P$ vor und hinter der Projektionsebene

polarisierter Bilder. Der Betrachter trägt eine billige Polarisationsbrille und ist in seiner Position nicht gebunden. Die Qualität des Raumbildeindrucks ist hier nahezu perfekt, auch Farben und Bildsequenzen stellen kein Problem dar. Im letzteren Fall muss natürlich eine Synchronisation der beiden Bilder erfolgen, am einfachsten über die Überlagerung von Zwillingsbildern (siehe Abb. 11.2). Akzeptiert man eine aufwendigere Technologie, so können die beiden Bilder auch mittels eines Head-Mounted Displays unmittelbar vor den Augen gezeigt werden.

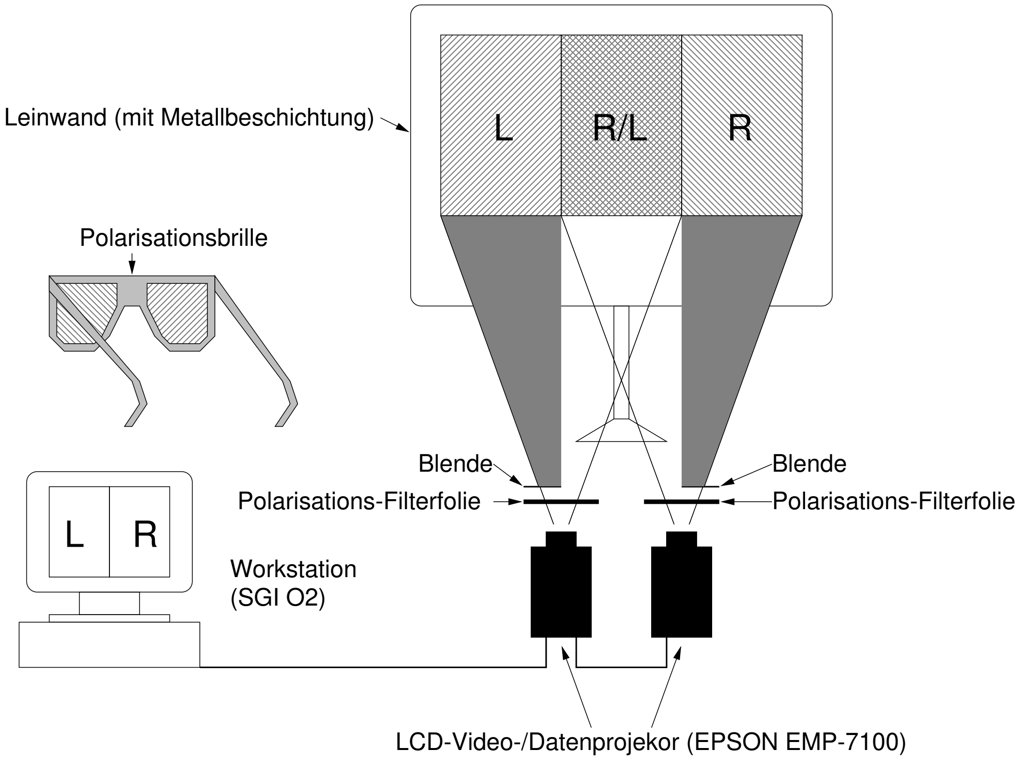


Abbildung 11.2: Stereoprojektion mittels zweier LCD-Beamer

Man beachte, dass die beschriebenen Verfahren der Stereoskopie zwar realistischen Raumeindruck vermitteln, sich aber dennoch auf (zwei) rein zweidimensionale Abbildungen derselben Szene abstützen. Das (derzeit) einzige *echt* dreidimensionale Verfahren zur Bildaufzeichnung und -rekonstruktion stellt die *Holographie* dar, bei der auf *einem* Bild der Raumeindruck festgehalten wird. Allerdings ist die Holographie ein rein photographisches Verfahren und somit kein Verfahren zur Visualisierung in unserem Sinne (es gibt keine digitalen, synthetischen Hologramme). Bei der Holographie wird die Interferenz von Lichtwellen ausgenutzt, wobei zur Erzielung perfekter Darstellungen phasenreines Licht (also z. B. Laserlicht) erforderlich ist. Grob gesprochen, beruht das Prinzip darauf, dass das Interferenzmuster des von der Szene reflektierten Lichts mit einem Referenzstrahl auf Film aufgezeichnet wird. Durch Projektion des Referenzstrahls im Idealfall kann das reflektierte Licht und damit das dreidiemnsionale Bild rekonstrukiert werden, und zwar unabhängig von der Perspektive des Betrachters. Glücklicherweise braucht man nicht immer das exakt gleiche Laserlicht und manchmal sogar überhaupt keinen Laser.