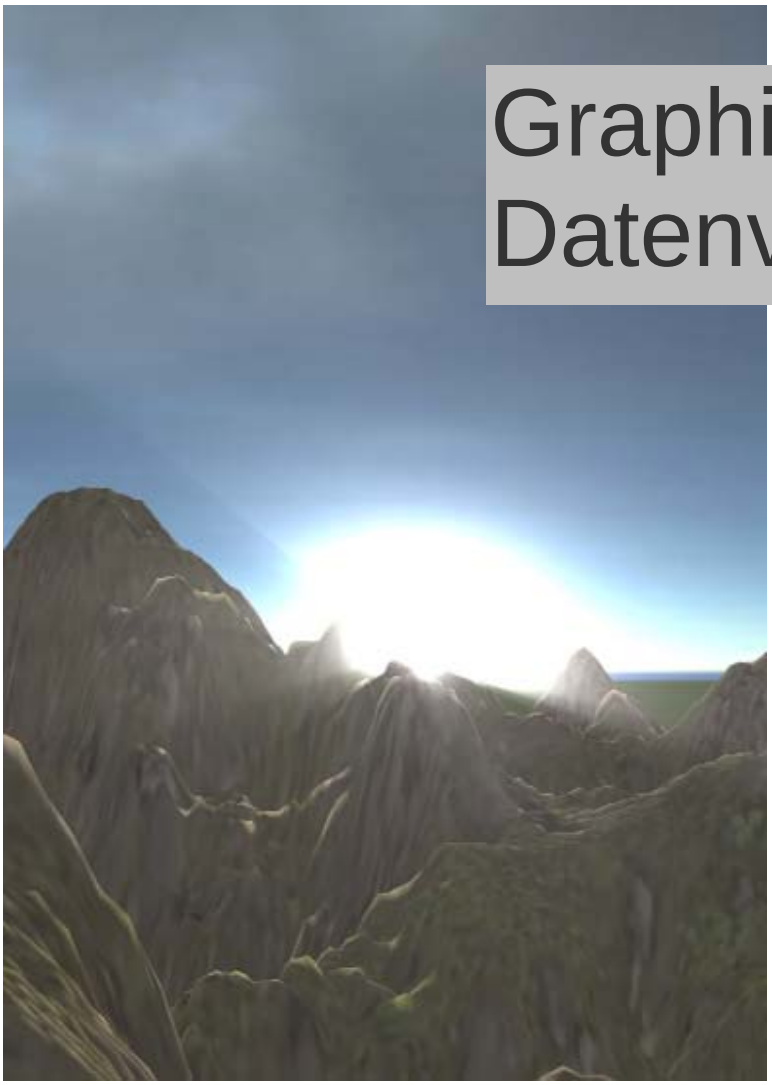
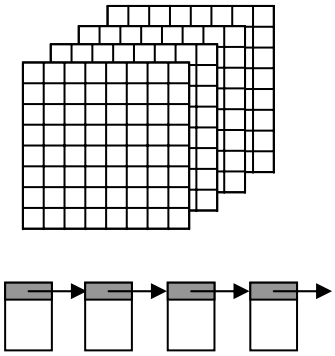
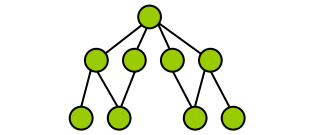
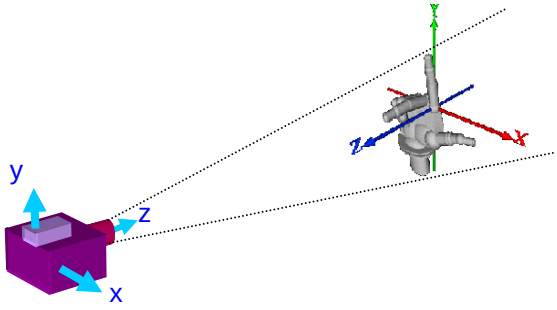
Graphische Datenverarbeitung

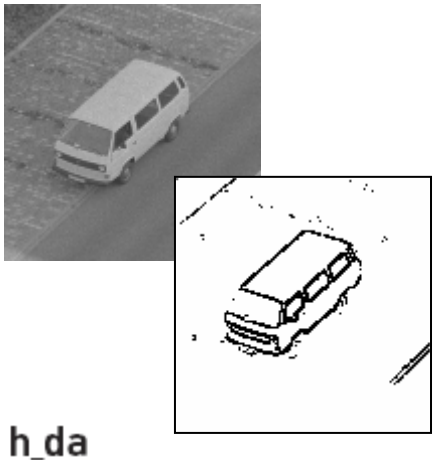


B. Amend und Ph. Hammer; erstellt in:
„Vertiefung aktueller Themen in der Computer Graphik“









Prof. Dr. W.-D. Groch
Prof. Dr. E. Hergenröther

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Inhaltsübersicht: Graphische Datenverarbeitung

Seitenangaben bzgl. Gesamt-PDF:

Inhaltsübersicht 2

1 Einführung und Überblick 4
1.1 Ein Überblick über das Fachgebiet der Graphischen Datenverarbeitung 4
1.1.1 Computer Graphik (engl. Computer Graphics) 4
1.1.2 Digitale Bildverarbeitung (engl. Digital Image Processing) 6
1.1.3 Rechnergestütztes Konstruieren (engl. Computer Aided Design, kurz CAD) 7
1.1.4 Computer Geometrie 7
1.2 Entwicklung und Zukunft des Fachgebietes 7

2 Graphische Datenverarbeitung - eine Disziplin der Informatik 9
2.1 Das Profil des „Computer Graphikers“ 9
2.2 Graphische Daten und einige ihrer Besonderheiten 9
2.2.1 Aliasing und Antialiasing 10
2.2.2 Pixel, Vektoren, Hierarchien 10

3 Graphische Objekte 12
3.1 Graphische Objekte und ihre Erzeugung 12
3.1.1 Koordinaten und Attribute 12
3.1.2 Objekthierarchien 14
3.1.3 Virtuelle Kamera 15
3.1.4 Szenegraphen 17
3.2 Geometrische Modelle 17
3.2.1 Eine grobe Klassifikation geometrischer Modelle 17
3.2.2 3D-Kantenmodell 18
3.2.3 Flächenmodell 19
3.2.4 Volumenmodell 20

4 Graphische Programmierung 23
4.1 OpenGL 23
4.1.1 Definition von Primitiven 23
4.1.2 Transformationen 24
4.1.3 Entkoppelung von Transformationen in einer Hierarchie 25
4.1.4 Kamera 27
4.2 VRML / X3D 28

5 Visualisierungstechniken 32
5.1 Visualisierung geometrischer Modelle 32
5.1.1 Visualisierung ohne Berücksichtigung von Licht 32
5.1.1.1 Lineare Interpolation 32 32
5.1.1.2 Bilineare Interpolation 33 33
5.1.2 Rendering 33
5.1.2.1 Lokale Verfahren 34
5.1.2.2 Globale Verfahren 36
5.2 Visualisierung nicht-geometrischer Modelle 38

6 Grundlegende mathematische Sachverhalte 40
6.1 Einfache geometrische Grundobjekte 40
6.1.1 Punkt 40
6.1.2 Gerade, Strahl, Strecke 40
6.2 2D-Koordinatentransformationen 42
6.2.1 2D-Elementar-Transformationen (konventionell) 42
6.2.2 2D-Elementar-Transformationen mit homogenen Koordinaten 43
6.2.3 Zusammengesetzte 2D-Transformationen 44
6.3 3D-Koordinatentransformationen mit homogenen Koordinaten 45
6.3.1 Translation (Verschiebung) 46
6.3.2 Skalierung (Größenänderung) 46
6.3.3 Rotationen (Drehungen) 46

Seitenangaben bzgl. Gesamt-PDF:

7 Digitale Bilder; Farbe 48
7.1 Geometrische Aspete 48
7.1.1 Bildpunkte (Pixel) und digitale Bilder 48
7.1.2 Vektor- und Rastertechnik 48
7.2 Photometrische Aspekte 50
7.2.1 Schwarz-Weiß- und Grauwertbilder 50
7.2.2 Farbbilder 51
7.2.2.1 RGB- und CMY-Farbmodell 51
7.2.2.2 Weitere Farbmodelle (YUV, HSV) 52
7.2.2.3 Rechnerinterne Repräsentation von Farbbildern 55

8 Bildbearbeitung für Rasterbilder 56
8.1 Photometrische Bildmanipulationen 56
8.1.1 Punkt-Operatoren 57
8.1.1.1 Lineare Grauwerttransformation 57
8.1.1.2 Gamma-Korrektur 59
8.1.1.3 Reduktion der Anzahl verschiedener Grauwerte 60
8.1.2 Lokale Bildoperatoren 61
8.1.2.1 Faltung 61
8.1.2.2 Rangfolge-Operatoren 64
8.2 Einführung in kontur- und flächenorientierte Segmentierung 66
8.2.1 Binarisierung 67
8.2.2 Markierung von Zusammenhangskomponenten (ZHK-Markierung) 67
8.3 Geometrische Bildmanipulationen 68

9 Bildkompression und Dateiformate 72
9.1 Kompressionstechniken für Rasterbilddaten 72
9.1.1 Lauflängencodierung (Run Length Encoding RLE) 73
9.1.2 LZW-Codierung 73
9.1.3 Hufman-Codierung 73
9.1.4 Fraktale Bildkompression 73
9.1.5 Diskrete Cosinus Transformation mit Frequenzfilterung 74
9.1.6 Wavelet-Transformation mit Quantisierung 77
9.2 Bilddatenformate für Rasterbilder 79
9.2.1 BMP 79
9.2.2 GIF 79
9.2.3 PNG 80
9.2.4 FIF / FCI 80
9.2.5 JPEG 81
9.2.6 JPEG-2000 / WI 82
9.2.7 MPEG 83
9.2.8 Schlussbemerkungen zu Kompressionstechniken für Rasterbilder 85
9.3 Bildbeschreibende Dateiformate 85

10 Gewinnung digitaler Bilder 88
10.1 Sensoren 88
10.2 Scanner 89
10.3 Prozedurale Erzeugung synthetischer Bilder 91
10.4 Deklarative Erzeugung synthetischer Bilder 91

11 Ausgabe digitaler Bilder 93
11.1 Ausgabe digitaler Bilder auf einem Bildschirm 93
11.1.1 Bildspeicher (Frame Buffer) 93
11.1.2 Bilderzeugungs-Komponente 93
11.1.3 Bilddarstellungs-Komponente 94
11.1.4 Bildschirm 94
11.2 Ausgabe digitaler Bilder auf einem Drucker 96

Literatur 97
Index 98

Ein Bild sagt mehr als tausend Worte
(Konfuzius)

1 Einführung und Überblick

Das menschliche Gehirn nimmt visuelle Eindrücke um ein Vielfaches schneller auf, als beispielsweise Sprache, und daher fällt es der Mehrheit der Menschen leichter, sich Inhalte zu merken, wenn ihnen diese in Form von Graphiken oder Bildern veranschaulicht werden. Allgemein kann man sagen, dass die Informationsaufnahme beim Menschen zu etwa 80 Prozent visuell, also über das Auge, und nur zu etwa 10 Prozent über das Gehör erfolgt. Die anderen Sinne, wie der Geruchs-, der Tast- und der Geschmackssinn spielen eine untergeordnete Rolle. Daher entstand bereits in der Frühzeit der elektronischen Datenverarbeitung der Wunsch, Informationen (wie z.B. Berechnungsergebnisse) auch graphisch darzustellen, möglichst realistisch wirkende Abbildungen zu erzeugen und reale Bilder im Rechner speichern und manipulieren zu können. Um dies alles realisieren zu können, entwickelte sich die Graphische Datenverarbeitung als ein Teilgebiet der Informatik. Einige der wesentlichen Bereiche, die diesem Fachgebiet zugeordnet werden, werden im Folgenden beschrieben.

1.1 Ein Überblick über das Fachgebiet der Graphischen Datenverarbeitung

Meist werden die Bezeichnungen ***Graphische Datenverarbeitung*** (engl. ***graphical data processing***) und ***Computer Graphik*** (engl. ***computer graphics***) synonym verwendet. Neben der Computer Graphik gehören die Digitale Bildverarbeitung, das Rechnergestützte Konstruieren, die Computer Geometrie und der große Themenkomplex Multimedia zu den Bereichen mit der größten Industrierelevanz. Unter ***Multimedia*** versteht man die Integration von Texten, Bildern und Graphiken, Tonsequenzen, Videoclips und Animationen innerhalb interaktiver Systeme. Da Multimedia Bestandteil anderer Bachelorveranstaltungen ist, werden im Folgenden nur die übrigen vier Bereiche kurz vorgestellt, die auch die Basis für die Veranstaltung „Graphische Datenverarbeitung“ bilden.

1.1.1 Computer Graphik (engl. Computer Graphics)

Ein großer Teil der Kommunikation mit dem Rechner erfolgt heute über ***graphische Benutzeroberflächen*** (engl. ***Graphical User Interface***, kurz ***GUI***). Statt umständlicher Texteingaben werden Aktionen durch „Draufzeigen“ und „Anklicken“ ausgelöst. Dies gilt sowohl für die Bedienung von Betriebssystemen, als auch für Anwendungsprogramme jeder Art. Trotz der heute relativ komfortabel zu bedienenden Oberflächen ist die Mensch-Maschine-Kommunikation (engl. Human Computer Interaction, kurz HCI) bis heute ein sehr aktives Forschungsgebiet. Immer wieder werden neue, intuitiver zu bedienende Eingabegeräte und Eingabemethoden entwickelt, wie z.B. die Kinect, die wii-mode und der iPod zeigen.

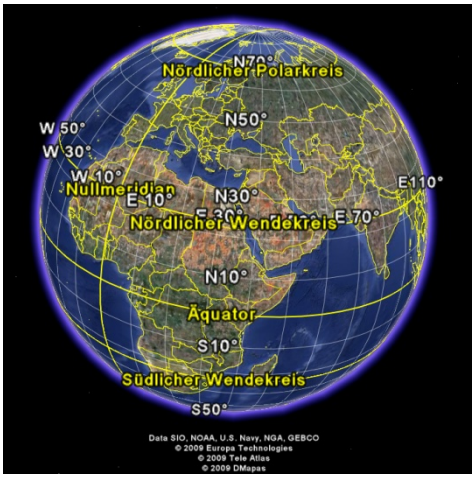


Abb. 1.1: Google Earth

Neben der Eingabe spielt in der Computer Graphik vor allem die Ausgabe, also die Visualisierung der Daten eine wichtige Rolle. Während noch vor ein paar Jahren die zweidimensionale Darstellung von Daten Standard war, werden heute überwiegend dreidimensionale Darstellungsformen genutzt. Selbst bei der Darstellung von Kartenmaterial, einer der klassischen 2D-Anwendungen, nutzt man heute Methoden zur dreidimensionalen Darstellung, wie die perspektivische Verzerrung und Schattierung der dreidimensionalen Objekte. Beispiele hierfür sind Google Earth (siehe Abbildung 1.1.) und die Graphikausgabe der neusten Generation von Navigationsgeräten.

Abb. 1.2: Prozedural erzeugte Gebäude; aus der Masterarbeit von S. Janusch

Bei der Visualisierung von Karten kann man auf bekannte Daten zurückgreifen. Etwas schwieriger wird es, wenn Sachverhalte dargestellt werden sollen, für die noch keine Daten vorhanden sind. Umgebungen und Charaktere in Computerspielen oder das Design und die Konstruktion neuer technischer Produkte sind typische Beispiele hierfür. In diesen Fällen muss die zu visualisierende Geometrie zunächst erzeugt werden. Das geschieht entweder durch die Nutzung so genannter Modellierungstools (siehe hierzu auch das Kapitel 1.1.3 „Rechnergestütztes Konstruieren“) oder man erzeugt das Gebilde algorithmisch z.B. mit Methoden der ***fraktalen Geometrie***, die ein Teilgebiet der ***Chaostheorie*** ist. Typische Beispiele von algorithmisch erzeugten Objekten sind Pflanzen, Wolken, Landschaften und Gebäude (siehe Abbildung 1.2).

Um die Räumlichkeit und die Oberflächenstruktur der künstlichen Objekte auf dem Bildschirm erkennbar zu machen, sind sowohl perspektivische Darstellung als auch Beleuchtung der Objekte sehr hilfreich. Erst durch die Abschattung der vom Licht abgewandten Seiten und durch Hinzufügen des Schattenwurfs wird die dreidimensionale Gestaltung der Geometrie und ihre Lage im Raum erkennbar. Daher sind sowohl die Projektionsberechnung, mit der man 3D-Körper auf eine Ebene abbildet, als auch die Beleuchtungs- und die Schattenberechnung wichtige Gebiete in der Computer Graphik.

Abb. 1.3: Kleid mit Samtshader; visualisiert von C. Weber und R. Giera in „Vertiefung aktueller Themen in der Computer Graphik“

Neben gegenständlichen Objekten werden in der Computer Graphik auch nicht-geometrische Sachverhalte visualisiert. Darunter versteht man z.B. Statistiken, die man beispielsweise als Balken- und Kuchendiagramme visualisieren kann, oder auch die Darstellung von Simulationsergebnissen, wie beispielsweise meteorologischen Vorhersagedaten. In diesem Zusammenhang spricht man häufig von ***wissenschaftlich-technischer Visualisierung (Scientific Visualization)***. Dabei will man die Qualität der Interpretation wissenschaftlicher Daten erhöhen, und so zu einem besseren Verständnis und zu neuen Einblicken gelangen.

Spezielle Anwendungsgebiete der Computer Graphik, die in den letzten Jahren zunehmend an Bedeutung gewonnen haben, sind die „***Virtual Reality***“ ***(VR; virtuelle Realität)*** und die zwischen „echter“ und virtueller Realität angesiedelten Abstufungen der „***Mixed Reality***“ ***(MR; vermischte Realität)***; siehe Abbildung 1.4.

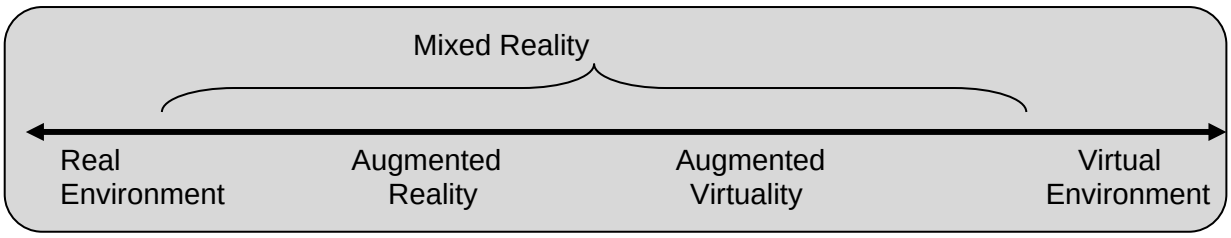


Abb. 1.4: Virtuelles Kontinuum. (Nach: Milgram, Paul; H. Takemura, A. Utsumi, F. Kishino (1994). "Augmented Reality: A class of displays on the reality-virtuality continuum"; Proceedings of Telemanipulator and Telepresence Technologies: 2351-34).

Bei der virtuellen Realität (ganz rechts) wird mit Hilfe des Rechners eine vollständige virtuelle Welt (Bild und z.B. Sound und Haptik) geschaffen, in die der Benutzer unter Verwendung spezieller Ein- und Ausgabe-Geräte (z.B. 3D-Maus und VR-Helm) vollständig „eintaucht“ (Immersion).

Bei der Mixed Reality werden reale und virtuelle Objekte gleichzeitig verwendet. So sind z.B. bei der „***Augmented Virtuality***“ (***AV; erweiterte Virtualität***; Mitte rechts) in einer virtuellen Szene auch reale Objekte enthalten; (z.B. „realer“ Moderator in virtuellem Sendestudio). Und bei der „***Augmented Reality***“ (***AR; angereicherte Realität***; Mitte links) enthält eine reale Szene zusätzliche virtuelle Objekte; (z.B. virtuelle Wegweiser, die in das Blickfeld des Benutzers eingeblendet werden).

Mixed Reality erfordert ein gut funktionierendes Zusammenspiel zwischen Computer Graphik und Bildverarbeitung, damit die virtuellen Objekte wirklichkeitsgetreu hinsichtlich Position, Orientierung, Beleuchtung und Schattenwurf usw. in die reale Umgebung eingeblendet werden können.

1.1.2 Digitale Bildverarbeitung (engl. Digital Image Processing)

Digitale Bildverarbeitung ist ein Sammelbegriff, der alle Verarbeitungsschritte vom Bildeinzug (z.B. mit Hilfe einer Kamera), über die Bildbearbeitung, die Bildsegmentierung und die Klassifizierung bis hin zur Bildinterpretation (maschinelles Sehen, Computer Vision) umfasst.

Die Digitale Bildverarbeitung stellt von der Vorgehensweise her die Umkehrung der graphischen Datenverarbeitung dar; (siehe Abbildung 1.5): während bei der graphischen Datenverarbeitung i.d.R. aus Beschreibungen synthetische Bilder erzeugt werden, geht man bei der digitalen Bildverarbeitung von (realen) Bildern aus, analysiert diese und versucht, z.B. eine Beschreibung des Bildinhaltes abzuleiten. Beide Ansätze müssen bei anspruchsvollen Aufgaben über zusätzliches Wissen verfügen: Weltmodell, Reflektionsmodell, Beleuchtungsmodell und Kameramodell; siehe Kapitel 10.

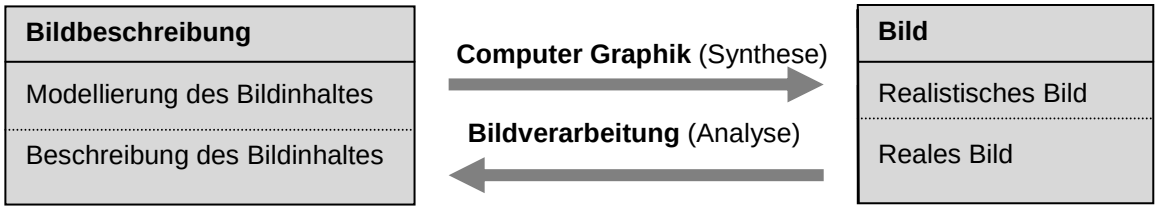


Abb. 1.5: Prinzipieller Zusammenhang zwischen Computer Graphik und Bildverarbeitung.

Die ***Bildbearbeitung*** (siehe Kapitel 8) umfasst Methoden zur geometrischen und photometrischen Manipulation von Bildern; z.B. Vergrößerungen/Verkleinerungen, Helligkeits-, Farb- und Kontraständerungen, Störungseliminationen und Hervorhebungen bestimmter Sachverhalte. Die Bildbearbeitung erzeugt dabei aus einem Bild wieder ein Bild.

Das Ziel einer ***Bildsegmentierung*** ist es, ein Bild in seine wesentlichen Bestandteile (Objekte oder Objektteile) zu zerlegen und diese zu charakterisieren (z.B. Größe und Form). Die Bildsegmentierung erzeugt aus einem Bild eine symbolische Beschreibung des Bildes (siehe Abbildung 1.6).

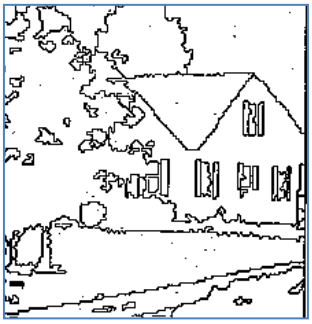


Abb. 1.6:Beispiel für eine Bildsegmentierung

Die so beschriebenen Bildteile können anschließend an Hand ihrer Merkmale klassifiziert und benannt werden. Diese ***Klassifizierung*** geschieht in der Regel unter Verwendung zusätzlichen Wissens. Das Wissen kann dabei von allgemeiner Natur sein (z.B. physikalische Modelle), oder spezielle Eigenschaften der Szene umfassen (Vorgabe von Prototypen, arttypischen Beschreibungen etc.). Die Aufgabenstellung kann auch darin bestehen, das Vorhandensein bestimmter Objekte, deren Anzahl, Formen, Positionen, Orientierungen usw. festzustellen.

Eine der schwierigsten Aufgaben besteht darin, beliebiges Bildmaterial (etwa eine mit einer Videokamera aufgenommene Bildfolge) möglichst in Echtzeit vollständig zu interpretieren. (Dies bedeutet, dass eine Beschreibung der in der Szene abgebildeten Objekte und ihrer gegenseitigen Beziehungen abgeleitet werden muss). Diese ***Bildinterpretations***- bzw. ***Com-***

puter-Vision-Aufgabe ist außer für Spezialanwendungen immer noch weitgehend ungelöst und ist Gegenstand intensiver Forschungsarbeiten.

1.1.3 Rechnergestütztes Konstruieren (engl. Computer Aided Design, kurz CAD)

Unter ***Rechnergestütztem Konstruieren*** (***CAD***, **C**omputer **A**ided **D**esign) versteht man die Unterstützung des Konstruktionsvorganges eines industriellen Produktes durch ein EDV-System.

Das Spektrum reicht dabei von einfachen 2D-CAD-Systemen zum computergestützten technischen Zeichnen bis hin zu 3D-CAD-Systemen, die die Konzipierung, Entwicklung und Konstruktion eines technischen Produktes unterstützen. Hierzu gehört die rechnerische Modellierung technischer Objekte (siehe Kapitel 3), ihre Visualisierung (siehe Kapitel 5), die gesamte Datenhaltung, die Durchführung umfangreicher Berechnungen, die Gewährleistung der Kommunikation usw..

Neben der Form und Farbe des Produktes werden dabei auch z.B. die Materialeigenschaften, sowie die Funktion und Ergonomie des Produktes berücksichtigt. Die Zielsetzung von CAD ist, schneller zu besseren und preiswerteren Produkten zu kommen.

CAD wird häufig als Bestandteil von ***CIM*** (**C**omputer **I**ntegrated **M**anufactoring; Computerintegrierte Fertigung) gesehen. Dabei sollen alle technischen und betriebswirtschaftlichen Fertigungsaspekte auf der Basis eines gemeinsamen Datenbestandes zusammengeführt werden.

1.1.4 Computer Geometrie

Die ***Computer Geometrie*** hat sich als eigenständiges Forschungsgebiet etabliert, das in hohem Maße z.B. in den Bereichen CAD, Computer Graphik und Robotik Anwendung findet. Die Computer Geometrie beschäftigt sich im weitesten Sinne mit der Behandlung geometrischer Sachverhalte mit Hilfe des Rechners.

Hierzu gehören Fragestellungen der analytischen Geometrie (siehe Kapitel 6), die z.B. in der Graphik bei der Objektmodellierung und –darstellung benötigt werden, wie z.B. räumliche Transformationen und Schnittpunktpunkt-Berechnungen zwischen Linien und/oder Flächen und auch die in Kapitel 1.1.1 „Computer Graphik“ erwähnten Berechnungen zur perspektivischen Darstellung.

Weitere Standardthemen sind Probleme der geometrischen Suche (z.B. "liegt-in"-Fragestellungen, Suche nach dem nächsten Nachbarn), Triangulationen von Punktmengen, spezielle geometrische Konstruktionen und Kollisionsuntersuchungen z.B. für die Robotik und Virtuelle Realität. Auch die Modellierung von möglichst glatt verlaufenden kurvigen Geometrien, wie man sie beispielsweise im Karosseriebau nutzt, ist Thema der Computer Geometrie.

1.2 Entwicklung und Zukunft des Fachgebietes

Die Entwicklung und die Zukunft des Fachgebietes „Graphische Datenverarbeitung“ sollen hier nur stichwortartig aufgelistet werden:

Ab 1950; Kathodenstrahlröhre zur graphischen Ausgabe am Rechner (Whirlwind, SAGE)
Kommando-Eingabe
Bewegung einzelner Punkte
Plotterausgaben
passive Graphische Datenverarbeitung

1963: Sketchpad (I. Sutherland)
Konstruieren mit ligthpen
"Geburtsstunde" des CAD
interaktive Graphische Datenverarbeitung

ca. 1965: Erstes allgemein verfügbares Display (Modell 2250 - IBM)

bildwiederholendes Vektorsichtgerät [vector-refresh-CRT (Cathode Ray Tube): sehr kostspielig
Speicherröhre [DVST (direct view storage tube), Tektronix]: preiswerte Technologie

1967: Hochleistungsdisplays mit 3D-Transformationshardware (z.B. Adage, Evans and Sutherland)
realistische Bilder, Dynamik
sehr kostspielig

ab 1975: Graphikfähige PCs (bildwiederholende Rastersichtgeräte)
"Graphik für Jedermann"
Standardisierung (GKS, PHIGS, OpenGL, CGM, CGI, ...)
Leistungsfähige Graphikworkstations

ab 1987: Window-Systeme (Mac und MS-Windows), Graphik in Netzwerken

ab 2000: „nahezu“ Photorealismus, Echtzeit-Animation,
VR, AR und MR etabliert,
haptische Ein- und Ausgaben (z.B. Kraftrückkopplung bei Interaktionen und Kollisionen) durch Roboterarme möglich .

Heute: Nachfolgend nur einige Beispiele, die in Bezug zur Vorlesung GDV stehen:
„nahezu“ photorealistische Echtzeitdarstellungen durch Shadertechnik,
Darstellungsverfahren, die über den Photorealismus hinaus gehen (HDR),
intuitive Eingabetechniken für den Massenmarkt (Kinect, wii-mode, iPod)

Abb. 1.7:
Beispiel für eine über den Photorealismus hinausgehende Echtzeitdarstellung - realisiert mittels High Dynamic Range Rendering (HDRR) und Deferred Shading ; erstellt von M. Horsch in; „Vertiefung aktueller Themen in der Computer Graphik“

Zukunft: Geforscht wird an noch intuitiveren Eingabetechniken, die möglichst ohne Geräte, die in der Hand gehalten werden müssen, auskommen. Dabei soll der Anwender sich frei vor dem Monitor oder der Projektion bewegen können (Gestenerkennung).

Natürlich wird die realistische Darstellung künstlicher Objekte, nicht nur für Echtzeitanwendungen, weiterhin verbessert werden und auch die dargestellten Szenen werden immer größer und komplexer werden.

Vielleicht wird es in absehbarer Zukunft auch bessere Geräte zur haptische Ein- und Ausgabe geben.

Was die Zukunft wirklich bringen wird, weiß natürlich keiner, aber dass es aus Sicht der Computer Graphik spannend bleibt, dessen sind wir uns sicher. ☺

2. Allgemeine Grundlagen der Graphischen Datenverarbeitung

Nachdem Sie bereits in Kapitel 1 einen Überblick über die vielfältigen Aufgabengebiete der Graphischen Datenverarbeitung bekommen haben, wollen wir Sie in diesem Kapitel mit den Anforderungen (siehe: Kapitel 2.1 „Das Profil des Computer Graphikers“) und mit einigen charakteristischen Eigenschaften der Graphischen Datenverarbeitung (siehe: Kapitel 2.2. „Graphische Daten und einige ihrer Besonderheiten“) vertraut machen.

2.1 Das Profil des "Computer Graphikers"

Die Graphische Datenverarbeitung ist in jeder Beziehung ein buntes Fachgebiet. So gehört zum Aufgabengebiet eines Computer Graphikers nicht nur die Hard- und Software-Entwicklung von Mensch-Maschine-Schnittstellen, die photorealistische oder gerade nicht photorealistische Darstellung einer Szene, die Veränderung eines Bildes oder die Extraktion bestimmter Informationen daraus, sondern noch viel, viel mehr.

Folglich sind auch die Anforderungen an einen „Computer Graphiker“, also eine Spezialistin oder einen Spezialisten auf dem Gebiet der Graphischen Datenverarbeitung, mannigfaltig. Tatsächlich gibt es nur wenige Anwendungsgebiete der Informatik, in denen Techniken der Graphischen Datenverarbeitung nicht sinnvoll eingesetzt werden können und umgekehrt kann der Computer Graphiker ohne das Wissen anderer Informatikdisziplinen nicht auskommen. So sind beispielsweise Softwaretechnik, Datenbanken, Netzwerktechnik und auch theoretische Informatik Gebiete, die dem Graphiker bei der Entwicklung und Realisierung von Lösungen hilfreich und bisweilen unentbehrlich sind.

Reduziert man die Fragestellung auf die Spezialistinnen bzw. die Spezialisten, die sich auf dem Gebiet der "Kern-GDV" betätigen möchte, so verschärft sich das Anforderungsprofil. Ein kurzer Blick in die Literatur zeigt, dass die Graphische Datenverarbeitung aufs Engste mit der Entwicklung von Algorithmen und Programmen verknüpft ist. Die Programmiersprachen C, C++ und inzwischen auch Java, spielen eine herausragende Rolle und gerade im Zusammenhang mit der Shadertechnologie, aber nicht nur hier, sind teilweise detaillierte Kenntnisse der entsprechenden Hardware unabdingbar. Früh werden Sie merken, dass grundlegende Verfahren und Algorithmen ohne solide Mathematikkenntnisse nicht entwickelt werden können. Dabei ist der souveräne Umgang mit homogenen Koordinaten (siehe Kapitel 6 „Mathematische Grundlagen“) eine Selbstverständlichkeit. Da in der Graphik alles Denkbare modelliert wird (Geometrien, Materialien, Farbe, Beleuchtungen, Reflexionseigenschaften, usw.) sollte ein "Computer Graphiker" nicht zuletzt auch ein gewisses Interesse für elementare physikalische Gesetzmäßigkeiten mitbringen.

Nur eins ist der „Computer Graphiker“ **nicht**: er ist **kein** künstlerisch Schaffender, obwohl ein Gefühl für Ästhetik dem Graphiker bei der Arbeit durchaus hilfreich sein kann.

2.2 Graphische Daten und einige ihrer Besonderheiten

In der "Steinzeit" der Informatik waren Programmierer und Anwender sowohl bei der Programmerstellung, als auch bei den Laufzeiten von Programmen daran gewöhnt, sich in Geduld zu üben. Im Gegensatz dazu war in bestimmten Anwendungen der Graphischen Datenverarbeitung schon frühzeitig eine Programmausführung in Echtzeit gefordert. Die Forderung nach hohen Geschwindigkeiten machte die Graphik damals wie heute (Spieleindustrie, ***Virtual Reality (VR)***, ***CAD***, usw.) zum Motor technischer Innovationen. Beinahe nirgendwo gilt das Motto "Der Appetit kommt mit dem Essen" so sehr, wie in der Graphischen Datenverarbeitung. Mit anderen Worten: Je schneller die Hardware und die Algorithmen, umso höher die Anforderungen an Geschwindigkeit, Realitätstreue und das zu verarbeitende Datenvolumen (Bildauflösung, Echtfarben, Texturen, 3D); siehe Abbildung 2.1.

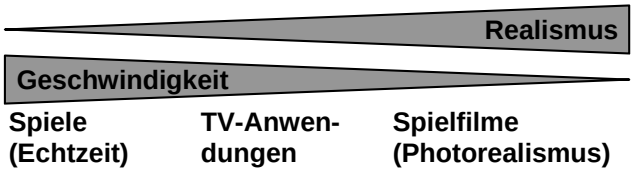


Abb 2.1: Realismus contra Geschwindigkeit

Die Bearbeitung der breiten Palette von Aufgabenstellungen erfordert neben schneller Hardware und ausreichendem

Speicherplatz auch die Verwendung der unterschiedlichsten Datenstrukturen und Algorithmen. In diesem Kapitel soll nun ein erster Eindruck davon vermittelt werden, mit welchen typischen Problemen und Besonderheiten beim Umgang mit graphischen Daten zu rechnen ist. Ein ganz kennzeichnendes Problem der Computer Graphik ist der in Bildern immer wieder sichtbare Treppeneffekt. Er entsteht, wenn schräg verlaufende Linien dargestellt werden (siehe hierzu: Kapitel 2.2.1 „Aliasing und Antialiasing“).

Weiterhin soll gezeigt werden, dass Datenstrukturen wie Arrays, Listen und Bäume in der Graphischen Datenverarbeitung eine breite Anwendung haben (siehe Kapitel 2.2.2 „Pixel, Vektoren und Hierarchien“). Diese Datenstrukturen und die zugehörigen Algorithmen sind nur ein Beispiel dafür, dass man in der Graphischen Datenverarbeitung die typischen Lösungsstrategien der Informatik bestens anwenden kann, ja sogar anwenden muss, um eine praktikable Lösung entwickeln zu können.

2.2.1 Aliasing und Antialiasing

Abb. 2.2: Aliasing

Abb. 2.3: Antialiasing

Wer sich schon mal digitale Bilder in einer niedrigen Auflösung angeschaut hat, kennt die unerwünschten sogenannten ***Treppeneffekte*** (siehe Abb. 2.2 bis 2.4. Dieser Effekt ist einer von vielen gleichartigen, die unter dem Begriff ***Aliasing*** bekannt sind. Die Ursache dafür ist, dass analoge Daten für den Rechner digitalisiert werden müssen. Soll also eine Strecke mit leichter Steigung auf dem Bildschirm dargestellt werden, dann muss diese Strecke in eine endliche Anzahl diskreter Bildpunkte, sogenannter ***Pixel***, aufgeteilt werden. Diesen Vorgang nennt man ***Vektor-Raster-Konvertierung***. Dabei entstehen zwangsläufig Fehler, die wir dann als Treppen- bzw. Aliasingeffekte wahrnehmen (siehe Abb. 2.2 „Aliasing“). Mit diesem Effekt ist bei jeder Umwandlung von analogen in diskrete Daten zu rechnen. In der Graphik wird er nur besonders "schön" sichtbar. Die Maßnahmen zur Behebung dieses Problems nennt man ***Antialiasing***. Ein Beispiel dafür sehen Sie in Abb. 2.3. Aber auch Abb. 2.4 stellt ein mit Antialiasing geglättetes Bild dar.

2.2.2 Pixel, Vektoren und Hierarchien

Abb. 2.4: „Bildausschnitt "

Da eine Bildmatrix aus Pixeln (siehe Abb. 2.4 „Bildausschnitt“) kein Kontinuum darstellt, spricht man von einer ***Rasterrepräsentation*** der Bilddaten und folglich spricht man von Rastertechniken, wenn es sich um den Umgang mit Rasterdaten (Veränderung, Speicherung, Ein- und Ausgabe) handelt. Eine sehr einfache Bildbearbeitungsaufgabe[1] könnte darin bestehen, den weißen Pixeln in Abb. 2.4 einen einheitlichen Grauwert zuzuweisen. Diese Aufgabe kann z.B. mit einem "einfachen" Malprogramm bewältigt werden. Eine anspruchsvollere Aufgabe der Bildverarbeitung wäre es, automatisch zu erkennen, dass hier der Schriftzug „50^0“ steht.

Die Interpretation des Bildinhaltes entspricht im Prinzip der Umkehrung der Vektor-Rasterkonvertierung. Bei der Vektor-Rasterkonvertierung werden übrigens nicht nur Strecken gerastert, sondern ganz beliebige graphische Objekte, wie beispielsweise auch der Schriftzug „50^0“. Beim umgekehrten Fall müssen die Pixel zu einer sinnvollen geometrischen Einheit zusammengefügt werden, um somit anschließend die richtig Bedeutung ermitteln zu können. Man nennt diesen Vorgang daher ***Raster-Vektorkonvertierung***.

Während die Vektor-Rasterkonvertierung ein Vorgang ist, der standardmäßig bei jeder Graphikanwendung durchgeführt wird und i. a. hardwareunterstützt ist, sind für die Raster-Vektorkonvertierung ausgeklügelte Algorithmen entwickelt worden. Man kann sich vorstellen,

[1] Die hier beschriebene Bildbearbeitungsaufgabe ist ein Beispiel für eine Rastertechnik

dass es eine wesentlich anspruchsvollere Aufgabe ist, aus Bruchstücken ein Ganzes aufzubauen, als ein bestehendes Objekt in kleine Einheiten zu zerlegen.

Rastertechniken und Raster-Vektorkonvertierung sind typische Aufgaben der ***Digitalen Bildverarbeitung***. In der ***Computer Graphik*** hingegen erzeugt man das Bild, das dann mittels Digitaler Bildverarbeitung bearbeitet wird, also beispielsweise antialiased wird. Dabei geht man standardmäßig so vor: man definiert zunächst eine Szene, die aus graphischen Objekten besteht und nimmt dann diese Szene aus einem bestimmten Blickwinkel mit einer virtuellen Kamera auf. Die einfachsten Formen graphischer Objekts sind Punkte, Strecken und Dreiecke. Aus mehreren Dreiecken kann man dann beispielsweise, wie in Abb 2.7 dargestellt die Füße der Pinguine aus Abb. 2.5 definieren. Aber nicht nur die Füße bestehen aus Dreiecken sondern die gesamten Pinguine wurden aus Dreiecken zusammengebaut.

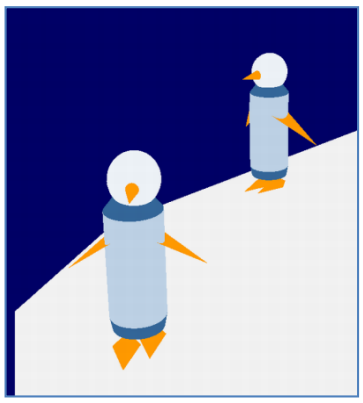


Abb. 2.5: Pinguine von D. Paulus (GDV Prakt. SoSe07)

Um die Konstruktion komplexer Geometrien oder den Aufbau einer 3D-Szene zu erleichtern, stellen die meisten graphischen Anwendungen ***hierarchische Datenstrukturen***, wie Bäume, oder häufiger gerichteten, zyklenfreie Graphen zur Verfügung. Man nennt diese Struktur ***Szenengraph***. Ein Beispiel hierfür sehen Sie in Abbildung 2.6[2]. Vereinfacht kann man sagen, dass in den Blättern die geometrischen Daten einzelner Teile der Szene oder der komplexen Geometrie gespeichert werden und in den Knoten wird die Zusammensetzung der Einzelteile definiert. Im nachfolgenden Kapitel 3 „Graphische Objekte“ wird hierauf genauer eingegangen.

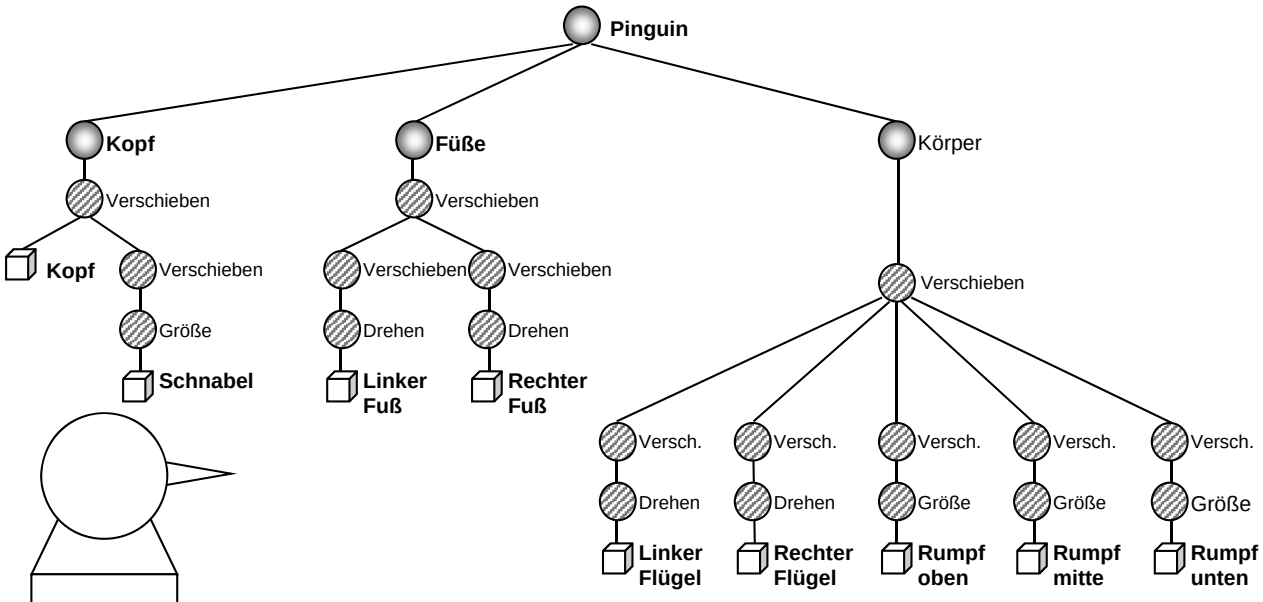
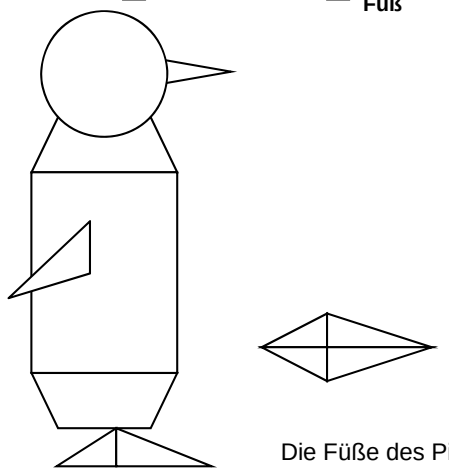


Abb. 2.6: Szenengraph des Pinguins (nach D. Paulus)



Die Füße des Pinguins

Abb. 2.7: Schematischer Aufbau des Pinguins (nach D. Paulus)

[2] „Größe“ im Szenengraph bedeutet: Die Höhe, Breite und Tiefe der Geometrie wird geändert.

3. Graphische Objekte

Im folgenden Kapitel wird zunächst beschrieben, wie graphische Objekte definiert werden können und was man bei ihrer Definition beachten muss (siehe Kapitel 3.1 „Graphische Objekte und ihre Erzeugung“). Nach der Erzeugung der graphischen Daten im ersten, wird im zweiten Teil die Speicherung der Daten, also ihre rechnerinterne Repräsentation, behandelt (siehe Kapitel 3.2 „Geometrische Modelle“).

3.1 Graphische Objekte und ihre Erzeugung

Um ein graphisches Objekt auf einem Ausgabegerät darzustellen, ist eine Reihe von Schritten erforderlich, die in diesem Kapitel besprochen werden sollen. (Die Mathematik hierzu folgt in Kapitel 6). Dies sei an einem Beispiel erläutert:

Die Aufgabe soll darin bestehen, eine Szene (z.B. einen Raum mit Mobiliar, wie ein Hörsaal) zu erstellen und auszugeben (zu ***rendern***).

- In einem ersten Schritt muss das Mobiliar definiert werden. Ein Tisch beispielsweise besitzt eine bestimmte Form und Größe (geometrische Daten) und Attribute wie die Oberflächenbeschaffenheit (Abb. 3.1.1a).
- Der Tisch muss, so wie die übrigen Möbelstücke, im Raum platziert werden (Abb. 3.1.1b).
- Es muss festgelegt werden, welche Ansicht bzw. welcher Ausschnitt des Raums auf dem Ausgabegerät erscheinen soll (Definition einer Sicht, hier im gewählten Beispiel die komplette Szene von oben).
- Die gewählte Sicht muss in geeigneter Größe und an der richtigen Stelle der Ausgabefläche dargestellt werden (Abb. 3.1.1c).

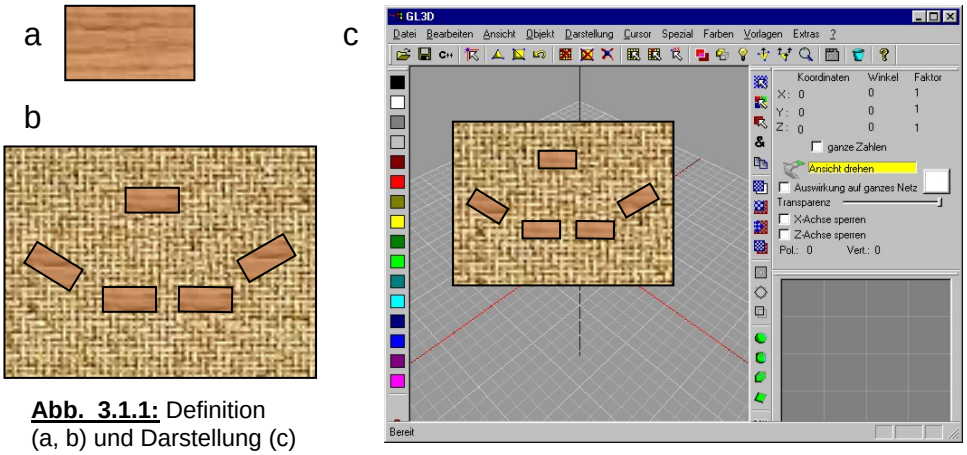


Abb. 3.1.1: Definition (a, b) und Darstellung (c)

3.1.1 Koordinaten und Attribute

Um die Form und Größe der Grundobjekte festzulegen, benötigen wir zunächst ein Koordinatensystem in dem das Objekt definiert werden kann. Am bekanntesten sind ***kartesische Koordinatensysteme***. Zu Modellierungszwecken werden dabei i.a. die rechtshändigen Varianten verwendet (s. Abb. 3.1.2). Objekte wie der Tisch im obigen Beispiel können hierin sehr einfach beschrieben werden (s. Abb. 3.1.3).

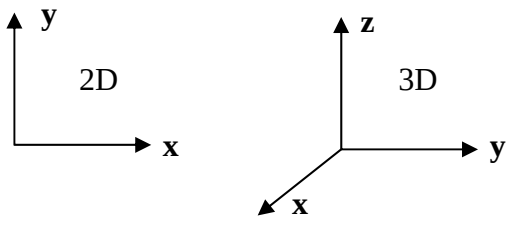


Abb. 3.1.2: Kartesische Koordinatensysteme

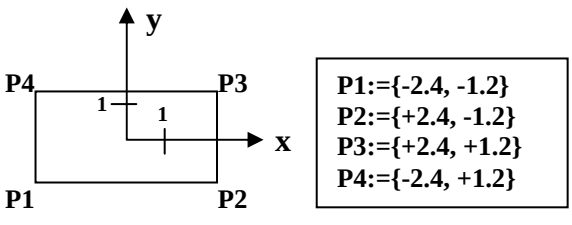


Abb. 3.1.3: Definition eines Rechtecks in kartesischen Koordinaten

Beachten Sie, dass die Koordinaten hier nicht ganzzahlig (also in Pixel) angegeben werden. Koordinatensysteme, in denen die Grundobjekte modelliert werden, bezeichnet man als ***Modellierungskoordinatensysteme*** oder auch "körpereigene Koordinatensysteme". Jedes Objekt, das modelliert wird, besitzt sein eigenes (daher "körpereigenes") Koordinatensystem. Die Koordinaten können i.a. beliebig als Integer- oder Gleitkommagrößen angegeben werden.

Sind die geometrischen Größen festgelegt, können dem Objekt nun ***graphische Attribute*** zugeordnet werden. Beispiele hierfür sind in Abb. 3.1.4 zu sehen.

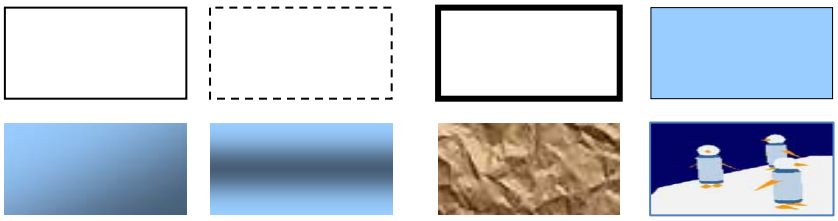


Abb. 3.1.4: Beispiele für graphische Attribute

Von links oben nach rechts unten sind dies:

- Fläche mit einfachem Rand (z.B. als Drahtgitter, sodass darunter liegende Objekte zu sehen sind.
- Rand gestrichelt
- Randdicke erhöht
- Fläche gefüllt
- Eckpunkte mit unterschiedlicher Farbe, die dann über die Fläche interpoliert wird.
- Fläche mit Reflexionseigenschaften
- Synthetische Textur auf Fläche gemappt (="geklebt")
- JPEG-Bild auf Fläche gemappt (wie man sieht, hier verzerrt).

Wie hätte man vorzugehen, wenn der Tisch nicht rechteckig, sondern rund (kreisförmig) wäre? Wir erinnern uns, dass es für einen Kreis eine Abhängigkeit der y-Koordinate von der x-Koordinate gibt. Diese spiegelt sich in den beiden Funktionen $y = \pm\sqrt{R^2 - x^2}$ wider.

Ein Kreis kann sehr einfach durch 2 Angaben beschrieben werden: Radius und Mittelpunkt. Legen wir das körpereigene Koordinatensystem für den Kreis so fest, dass der Mittelpunkt des Kreises im Ursprung liegt, kann der Kreis allein durch seinen Radius beschrieben werden. Der Kreis kann in einem Modell zwar mit dieser Angabe gespeichert werden, zur Ausgabe muss er jedoch in ein regelmäßiges Polygon umgewandelt werden. Dies gilt für alle gekrümmtem Kurven. Eine hohe Dichte der aufeinanderfolgenden Eckpunkte des Polygons gaukelt dem Betrachter eine glatte runde Form vor.

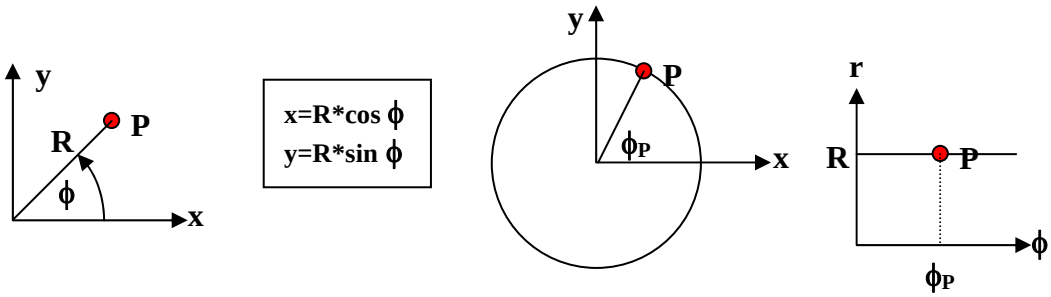
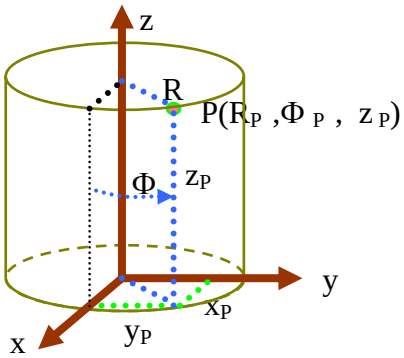


Abb. 3.1.5: Kreisdefinition in Polarkoordinaten

Die Form $y = \pm\sqrt{R^2 - x^2}$ ist zur Berechnung der Stützpunkte ungeeignet. Wir verwenden stattdessen ***Polarkoordinaten*** (Abb. 3.1.5). Hierbei können die x- und y-Koordinaten der Eckpunkte durch gleichmäßige Veränderung des Winkels ϕ gewonnen werden. Polarkoordinaten werden immer dann verwendet, wenn die einzelnen Punkte des Objekts

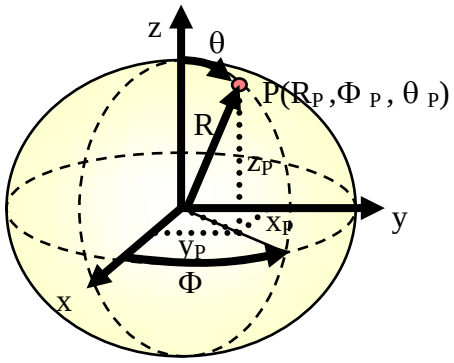
durch Drehung um eine Achse gewonnen werden können. Dies ist z.B. auch bei der Definition eines spiralförmigen Objekts der Fall.

In 3D kommen noch 2 weitere Koordinatensysteme hinzu, die für Objektmodellierungen verwendet werden. Dieses sind die ***Zylinderkoordinaten*** und die ***Kugelkoordinaten***. Will man ein Objekt wie eine Dose, eine Tonne, einen Kegel etc., ganz allgemein einen Körper mit Rotationssymmetrie um eine Achse, modellieren, dann verwendet man Zylinderkoordinaten. Diese entstehen aus Polarkoordinaten durch Hinzufügen eines z- Wertes.



x = R * cos Φ **y = R * sin Φ** **z = z**

Abb. 3.1.6: Zylinderkoordinaten



x = R * sin θ * cos Φ
y = R * sin θ * sin Φ
z = R * cos θ

Abb. 3.1.7: Kugelkoordinaten

Will man dagegen Körper mit Rotationssymmetrie bezügliche eines Punktes beschreiben (kugelförmige Gebilde), so verwendet man Kugelkoordinaten. Der Zusammenhang zwischen diesen beiden Koordinatensystemen und kartesischen Koordinaten ist in den Abbildungen 3.1.6 und 3.1.7 dargestellt.

3.1.2 Objekthierarchien

Nachdem nun die einzelnen Objekte in ihren körpereigenen Modellierungskoordinaten definiert worden sind, ist die nächste Aufgabe, sie in größeren Gruppen zusammenzufassen. Dies ist am Beispiel der Abb. 3.1.8 zu sehen. Bezüglich des übergeordneten Modellierungskoordinatensystems der Gruppe werden die elementaren Objekte positioniert und orientiert. Wir wollen uns diesen Vorgang veranschaulichen.

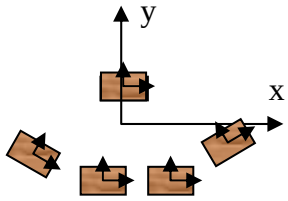


Abb. 3.1.8: Modelierungskoordinatensystem einer Gruppe

Abbildung 3.1.9 zeigt das Ergebnis einer Objekt***verschiebung*** (***Translation***). Zu den Koordinaten des Objekts werden entsprechend der Größe der Verschiebung Werte hinzuaddiert. Hier in y-Richtung ein positiver Wert, in x-Richtung der Wert 0.

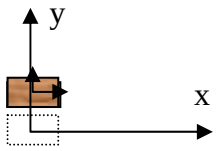


Abb. 3.1.9: Translation

Ein interessanteres Beispiel ist die ***Drehung*** (***Rotation***) eines Objekts um einen beliebigen Punkt. In der Regel sind Objekte in einem ***körpereigenen Koordinatensystem*** definiert (s.o.). Um beim Beispiel zu bleiben: Die Form des Mobiliars (Tisch, Stuhl) soll natürlich von seiner späteren Position im Raum unabhängig sein. Abbildung 3.1.10 zeigt diesen Sachverhalt (gestricheltes Rechteck mit Mittelpunkt im Ursprung). Das Objekt soll nun an anderer Position um den Winkel α gedreht werden.

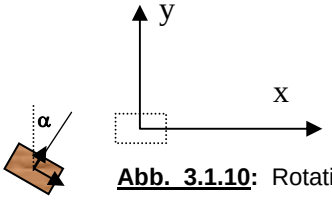


Abb. 3.1.10: Rotation

Im Prinzip erfolgt diese ***Transformation*** stets in 2 Schritten. Diese sind eine Drehung des Objekts um den Ursprung und eine Verschiebung zur gewünschten Position. Die Abbildungen 3.1.11 und 3.1.12 zeigen, dass es bei der Durchführung entscheidend auf die Reihenfolge ankommt, in der die Transformationen durchgeführt werden.

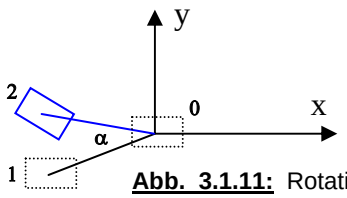


Abb. 3.1.11: Rotation (Translation+Rotation)

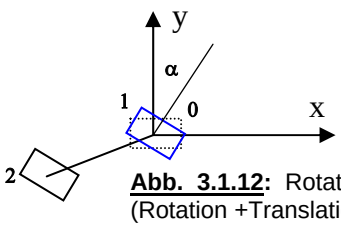


Abb. 3.1.12: Rotation (Rotation +Translation)

Wird das Objekt zunächst in die Position 1 verschoben und anschließend eine Drehung (um den Ursprung!) um den Winkel α durchgeführt, so ergibt sich das (fehlerhafte) Ergebnis 2 Abbildung 3.1.11). Wie aus Abbildung 3.1.12 ersichtlich ist, muss zur Erzielung des korrekten Ergebnisses zuerst die Drehung (1) und erst danach die Verschiebung durchgeführt werden.

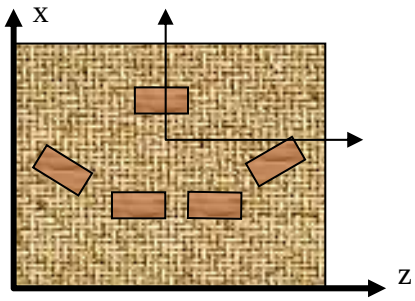


Abb. 3.1.13: Einbettung in WKS

Schlussfolgerung: Bei Transformationen, die in mehreren Schritten durchgeführt werden, ist das Ergebnis i.a. von der Reihenfolge der einzelnen Transformationen abhängig. Wegen einer mathematischen Begründung müssen Sie sich bis zum Kapitel 6 gedulden.

Durch die Zusammenfassung einzelner Objekte in einem neuen Koordinatensystem ist ein ***Teilmodell*** entstanden. Derartige Teilmodelle können erneut transformiert und zu komplexeren Einheiten zusammengefasst werden. Hierbei kann eine vielstufige ***Objekthierarchie*** entstehen. Am Ende, also in der obersten Ebene der Hierarchie, werden die Teilmodelle in die ***Szene*** (Miniwelt) eingebettet. Das Koordinatensystem der obersten Ebene der Hierarchie wird üblicherweise als ***Weltkoordinatensystem*** (WKS) bezeichnet. Im Beispiel der Abb. 3.1.13 ist zur Einbettung des Teilmodells in das WKS eine Rotation und eine Translation erforderlich (Beachten Sie, dass das gewählte Beispiel 3D sein soll und die WKS- y-Koordinaten die Höheninformation der Szene beinhalten).

3.1.3 Virtuelle Kamera

Die Modellierung einer Szene kann nicht Selbstzweck sein. Ein Modell kann dazu dienen, Berechnungen damit durchzuführen. Im Falle der Modellierung graphischer Objekte ist das Ziel in der Regel eine Visualisierung der Szene. Den Vorgang der Erzeugung einer (möglichst realistischen) Darstellung des Modells auf einem graphischen Ausgabegerät bezeichnet man als ***Rendering***. Um eine Szene erfolgreich zu rendern sind Angaben darüber erforderlich

- von wo aus (Position des "Betrachters“=Kamera) und unter welchem Winkel (Orientierung des "Betrachters"), (= ***äußere Kameraparameter***)
- welcher Ausschnitt der Szene ("Gesichtsfeld" des "Betrachters", Art des Kameraobjektivs) (= ***innere Kameraparameter***)

aufgenommen werden soll.

Dies wird als Definition einer ***View***, einer ***Sicht***, bezeichnet. Zur Definition einer View wird eine ***virtuelle Kamera*** mit sehr einfachen Eigenschaften (Lochkamera) verwendet. Diese Kamera besitzt ihr eigenes ***Kamerakoordinatensystem***. Im Gegensatz zu den Modellierungskoordinatensystemen ist dies üblicherweise ein Linkssystem. Dies bedeutet, dass die z-Achsen von Modellierungskoordinaten- und Kamerakoordinatensystem entgegengesetzt gerichtet sind.

Abbildung 3.1.14 zeigt das Einbringen einer Kamera in die Szene. Im linken Bild ist die Lage und Orientierung des Kamerakoordinatensystems (***äußere Kameraparameter***) bezüglich des WKS dargestellt. Rechts ist der Ausschnitt der Szene zu sehen, der von der Kamera gesehen wird. Der schraffierte Bereich ist (wie in der realen Welt) 3-dimensional und hat (bei perspektivischen Projektionen) die Form einer Pyramide, deren Spitze im Brennpunkt der Kamera liegt. Je spitzer die Pyramide ist, desto mehr entspricht dies einem Teleobjektiv. Eine flache Pyramide entspricht einem Weitwinkel(!)-Objektiv und führt zu starken perspektivischen Verzerrungen der "Aufnahme".

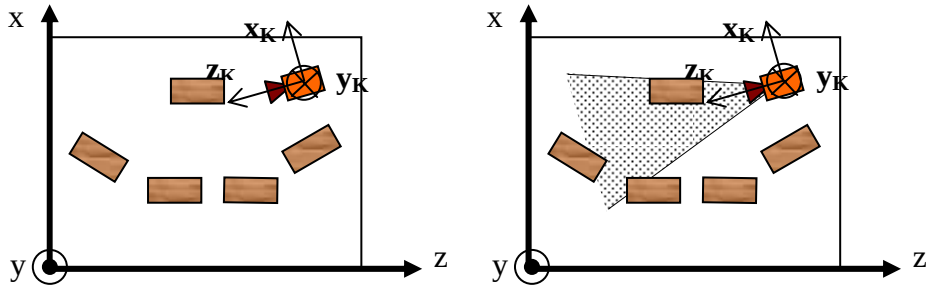


Abb. 3.1.14: Kamera: Ort, Orientierung und Ausschnitt

Eine Besonderheit der virtuellen Kamera besteht darin, dass der Sichtbereich nicht nur seitlich, sondern auch vorne und hinten begrenzt wird. Hierdurch wird der Pyramide zum Einen die Spitze abgeschnitten und zum Anderen wird die Pyramide in ihrer Gesamthöhe begrenzt. Der für die Kamera sichtbare Bereich entspricht geometrisch damit einem Pyramidenstumpf, auch ***Frustum*** (Abb. 3.1.15) genannt.

- **1** ***Brennpunkt*** (***COP***)
- **2** vordere ***Clipping-Ebene***
- **3** hintere Clipping-Ebene
- **4** seitliche Clipping-Ebenen

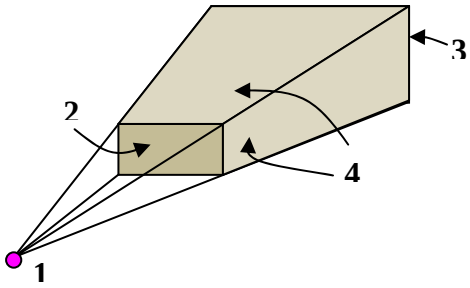


Abb. 3.1.15: Frustum

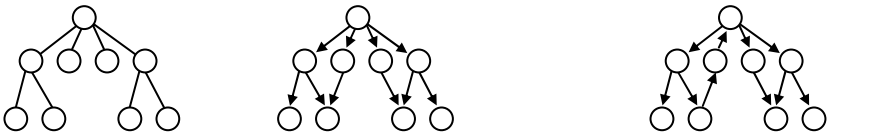
Der Vorgang des Ausblendens (=Abschneidens) nicht sichtbarer Teile der Szene wird als ***Clipping*** bezeichnet. Die Festlegung der geometrischen Abmessungen des Frustums entspricht dem Vorgang der Einstellung einer (einfachen) Kamera und damit der Festlegung der sog**. *inneren Kameraparameter***.

Um eine Szene zu rendern, müssen also die Weltkoordinaten noch

- in das Koordinatensystem der Kamera überführt (transformiert) werden (= ***Definition einer View***)
- entsprechend den Einstellungen der Kamera verzerrt und anschließend nach 2D projiziert werden (= ***Definition einer Projektion***).
- Hierauf folgt noch der Schritt der Umwandlung in ***Bildkoordinaten*** (= ***Vektor-Rasterkonvertierung***)

3.1.4 Szenengraphen

Hierarchien, wie sie im vorausgehenden Abschnitt beschrieben wurden, können in Datenstrukturen gespeichert werden, die als ***gerichtete, zyklenfreie (schleifenfreie) Graphen*** bezeichnet werden.



Baum | gerichteter zyklenfreier Graph | gerichteter nicht-zyklenfreier Graph

Abb. 3.1.16: Unterschied Baum & Graph

Im Unterschied zu einem Baum können bei einem gerichteten schleifenfreien Graphen auf einen untergeordneten Knoten mehrere Verweise existieren (Abb. 3.1.16). Wie bei einem Baum wird der oberste Knoten der Hierarchie als ***Wurzel*** (= ***root***) bezeichnet. Die Knoten der untersten Ebene, jene die keine Kinder haben, heißen ***Blätter*** (= ***Endknoten, terminale Knoten***). Die Wurzel repräsentiert die gesamte Szene. Die Blätter enthalten die Grundelemente und damit die Geometrie, die letztendlich gerendert wird. Die übrigen Knoten enthalten neben den Verweisen auf ihre Kinder möglicherweise die Transformationen, die auf alle ihre Kinder angewendet werden. In der graphischen Datenverarbeitung wird eine derartige Hierarchie häufig als ***Szenengraph*** bezeichnet.

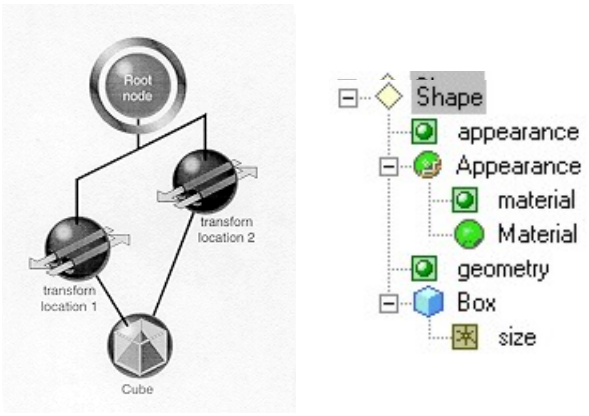


Abb. 3.1.17 : Bsp. 2 einfache Szenengraphen

Szenengraphen werden sehr unterschiedlich dokumentiert. Zwei einfache Beispiele hierfür sind in Abb. 3.1.17 zu sehen.

Das linke Beispiel zeigt eine einfache Szene, die einen Würfel an 2 verschiedenen Positionen enthält. Die rechte Szene enthält einen Würfel, dessen Materialeigenschaften abweichend von Standardwerten definiert wurden. Zur Erzeugung von Szenengraphen s. Kapitel 4.

3.2 Geometrische Modelle

In diesem Abschnitt soll von geometrischen Modellen und ihren rechnerinternen Repräsentationen die Rede sein. Derartige Modelle sind sowohl in der graphischen Datenverarbeitung als auch im CAD-Bereich von großer Bedeutung. Zunächst soll eine grobe Klassifizierung (Einteilung) vorgenommen werden. Eine Klassifikation nach verschiedenen Gesichtspunkten soll uns die wesentlichen Unterschiede verdeutlichen. Zur Erinnerung: ***Geometrische Modelle*** beschreiben im Gegensatz zu graphischen Modellen (Datenstrukturen wie Szenengraphen) nur ***geometrische*** und ***topologische Eigenschaften*** der modellierten Objekte und beinhalten keinerlei Information, die sich auf die Darstellung (Visualisierung) derselben bezieht. Mit anderen Worten: Kamerastandorte, Attribute von Linien, etc. sind nicht Bestandteil eines reinen geometrischen Modells.

3.2.1 Eine grobe Klassifikation geometrischer Modelle

Eine sehr grobe Unterscheidung können wir treffen, wenn wir von 2D- und 3D-Modellierung sprechen. Wir wollen später nur noch 3D-Modelle berücksichtigen. Anfang der 70er Jahre wurde in CAD-Modellen die Form eines zu konstruierenden Gegenstandes durch seine ***(technische) Zeichnung*** repräsentiert. Dies entspricht genau der herkömmlichen

Methode ein 3D-Objekt durch mehrere Ansichten darzustellen. Ein derartiges Modellierungsschema ist sehr robust. So ist es vielfach möglich aus 2 Ansichten eine 3. Ansicht abzuleiten. Dennoch ist es möglich bei der Konstruktion Fehler zu machen und "unsinnige" Objekte zu erzeugen. Vorteile, die sich aus dem Einsatz eines Computers ergaben, waren vor allem die Möglichkeit zur selektiven Änderung an einer technischen Zeichnung und die Ausgabe der Ergebnisse. Für echte 3D-Modelle ist die Einteilung entsprechend Abb.3.2.1 üblich und sinnvoll.

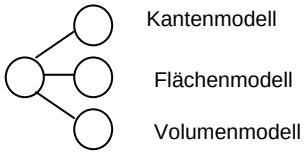


Abb. 3.2.1: Geometrische Modelle

3.2.2 3D-Kantenmodell (Drahtmodell, wire frame model)

Diesem Repräsentationsschema (***3D-Kantenmodell, Drahtmodell, wire frame model***) liegt die Vorstellung zugrunde, dass ein 3D-Körper durch seine Oberfläche begrenzt wird. Diese besteht möglicherweise aus mehreren Teilflächen (z.B. 6 Flächen bei einem Würfel). Die Teilflächen (auch Facetten genannt) stoßen an Kanten aneinander. Die Kanten sind sichtbar (wegen der Unstetigkeit beim Übergang von einer Fläche zur anderen) und werden daher im Modell repräsentiert. Die Beschreibung enthält also ***Kanten*** (Linien) und die sie begrenzenden ***Ecken*** (Endpunkte).

In Abb. 3.2.2. ist die ***rechnerinterne Darstellung*** (***RID***) als Graph (Prinzip) dargestellt. Beachten Sie den hierarchischen Aufbau des Modells.

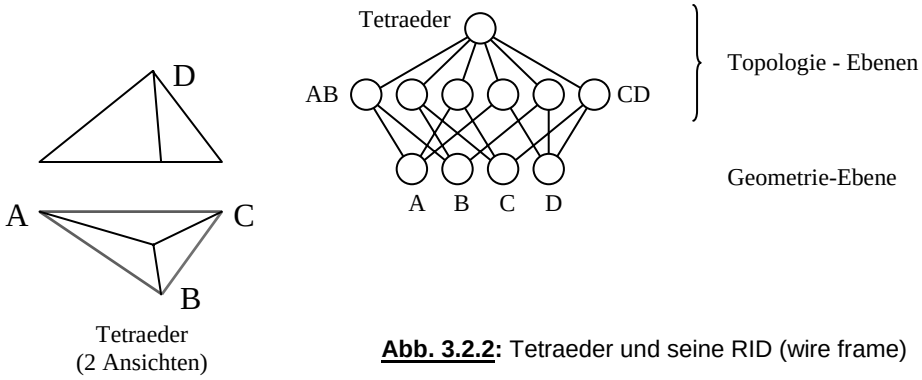


Abb. 3.2.2: Tetraeder und seine RID (wire frame)

Das Beispielsmodell enthält 4 Eckpunkte A bis D, die paarweise durch Kanten verbunden sind. Die Menge aller Kanten bildet das Gesamtobjekt, den Tetraeder. Hieraus ergibt sich eine allgemeingültige Definition für eine Kante: Eine Kante ist eine (nicht notwendigerweise geradlinige) Verbindung, die stets in einem Punkt beginnt und in einem (möglicherweise demselben) Punkt endet.

Abb. 3.2.2 stellt einen schleifenfreien gerichteten Graph dar. Die unterste Ebene enthält die geometrische Information des Objekts, die Koordinaten der Eckpunkte, während die Ebenen darüber die sog. Topologie (z.B. die Verbindungsvorschriften und Zusammenhänge) des Objekts beschreiben. Die Trennung von ***Geometrie*** und ***Topologie*** ist eine grundlegende Forderung bei der Erstellung geometrischer Modelle. Dies hat zahlreiche Vorteile; z.B.:

1. Änderungen an der Geometrie (Verschiebung eines Punktes) lassen die Topologie unverändert.
2. Wird z.B. in Abb. 3.2.2 der Punkt D gelöscht, entfallen damit automatisch alle zu D führenden Kanten. Es bleibt nur das Dreieck ABC übrig.
3. Entfällt eine Kante (z.B. die Verbindungslinie zwischen den beiden Kreisen in Abb. 3.2.3), so ändert sich an der Geometrie nichts, obwohl das Objekt in 2 getrennte Kreise zerfällt.

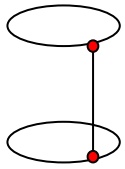


Abb. 3.2.3: Gültiges Kantenmodell

Abb. 3.2.3. zeigt ein Drahtmodell, das aus 3 Kanten und 2 Punkten besteht. Beachten Sie bei diesem Beispiel folgendes: Sollte der "Konstrukteur" die Absicht gehabt haben ein Drahtgerüst, bestehend aus 2 Ringen und einer Verbindung dazwischen, zu modellieren, so ist dieses Modell eindeutig (und damit vollständig). Sollte er beabsichtigt haben einen zylindrischen Vollkörper oder eine (vielleicht offene) zylindrische Fläche (=geöffnete Konservendose) zu modellieren, so ist ihm dieses nicht eindeutig gelungen.

Allgemeine Schlussfolgerung: Flächenhafte Objekte und Vollkörper lassen sich mit Hilfe von Drahtmodellen nicht **eindeutig** (=**vollständig**) modellieren.

Unabhängig von einem speziellen Beispiel lässt sich auch allgemeiner formulieren, welche Information ein 3D-Drahtmodell enthalten muss:

1. Ein Objekt enthält eine Menge von Kanten.
2. Jede Kante ist durch ein Punktpaar (Anfangs- und Endpunkt) definiert. Beachte, dass die Reihenfolge der beiden Punkte keine Rolle spielt.

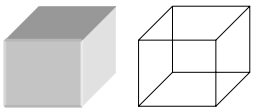


Abb. 3.2.5: Visualisierung verdeckter Teile

3. Jeder Punkt ist durch ein geordnetes Tripel von Koordinatenwerten definiert.

Verwendung von Drahtmodellen:

- Historisch wegen geringer Rendering-Zeiten,
- Ausgabe auf vektoriellen Ausgabegeräten,
- zur Visualisierung sonst verdeckter Teile (Abb. 3.2.5).

3.2.3 Flächenmodell (surface model)

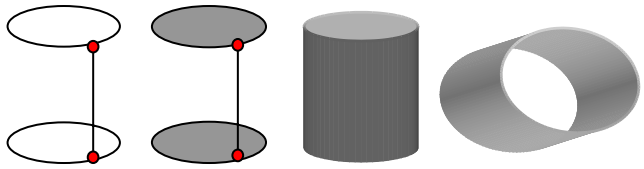


Abb. 3.2.6: Kantenmodell und 3 verschiedene Flächenmodelle

Wie wir festgestellt haben, lässt sich ein flächenhaftes Objekt (s. offene Konservendose) durch ein Drahtgittermodell nicht eindeutig repräsentieren. Dem Modell muss Information hinzugefügt werden, die aussagt, an welchen Stellen das Objekt mit einer Fläche "überzogen" ist. Man erhält dann ein ***Flächenmodell (surface model)***. In Abb. 3.2.6 besteht das Modell einmal aus 2 Kreisscheiben, die durch eine Kante zusammengehalten werden. Daneben sollte eine geöffnete Blechbüchse und ganz rechts ein offenes Ofenrohr modelliert werden. Alle 3 Modelle wären als Drahtgittermodell möglicherweise wie in Abb 3.2.6 links repräsentiert worden.

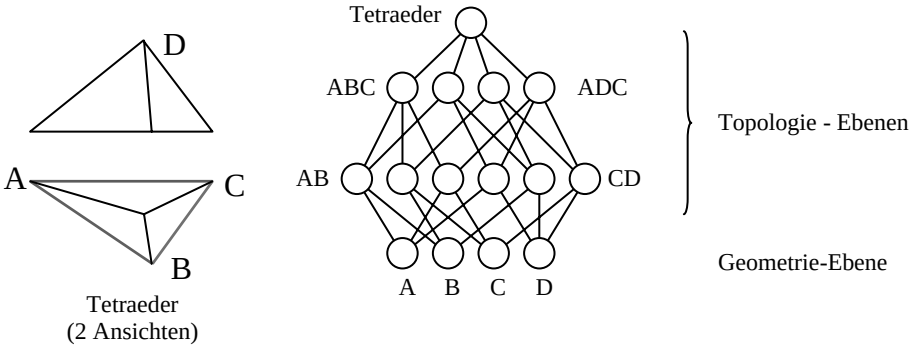


Abb. 3.2.7: Tetraeder und seine RID (surface model)

Gegenüber dem Drahtgittermodell enthält die RID eine zusätzliche Topologie-Ebene, die eine Folge von Kanten (= geschlossene Kontur) enthält. Die Kontur ist die Begrenzung einer Teilfläche (=***Facette***) des Objekts. In der Graphik und im CAD-Bereich sind Facetten sehr häufig Dreiecke. Dies hat u.a. folgende Gründe: 3 Punkte (im Gegensatz zu 4 Punkten) liegen grundsätzlich in einer Ebene. Dreiecke sind immer konvex. Letzteres vereinfacht viele Algorithmen (s. z.B. Vektor/Rasterkonvertierung).

Neben der Verwendung von Dreiecken gibt es noch einen weiteren Gesichtspunkt bei der Verwendung von Facetten: Eine Facette sollte i.a. eine Richtung besitzen, die es ermöglicht, Vorder- und Rückseite zu unterscheiden. Dies ist z.B. erforderlich, wenn bei der Visualisierung die beiden Seiten unterschiedlich eingefärbt werden sollen, wenn Licht auf die Facette fällt (die Helligkeit einer Fläche ist u.a. von der Richtung der Flächennormalen abhängig), oder wenn bei einer geschlossenen Fläche der Innen- vom Außenraum unterschieden werden soll.

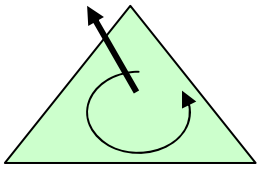


Abb. 3.2.8: Flächennormale aus Umlaufsinn

Die Facette wird als geordnete Folge von Kanten repräsentiert. Damit ist der ***Umlaufsinn*** implizit festgelegt. Aus dem Umlaufsinn kann nun die Richtung der ***Flächennormalen*** ermittelt werden (s. Abb. 3.2.8).

3.2.4 Volumenmodell (solid model)

Die durch Flächenmodelle repräsentierten Objekte können offene Raumflächen (wie ein fliegender Teppich) oder geschlossene Oberflächen (wie eine Coladose) sein. Will man wirklich Körper, die über einen Innen- und Außenraum verfügen, definieren, so ist es erforderlich, dem Modell noch weitere Information hinzuzufügen, damit die Repräsentation eindeutig wird. Hierdurch (***Volumenmodell, solid model***) wird es möglich für jeden Punkt des Raumes zu entscheiden, ob er im Inneren des Körpers liegt oder außerhalb. Damit ist auch der Begriff Volumenmodell definiert: Ein Modell, das zur eindeutigen Repräsentation eines Körpers geeignet ist, d.h. bei dem für jeden Raumpunkt entscheidbar ist ob er sich im Inneren des Körpers befindet oder nicht. Weiter unten werden die heute gebräuchlichen Repräsentationsschemata für Volumenmodelle genannt.

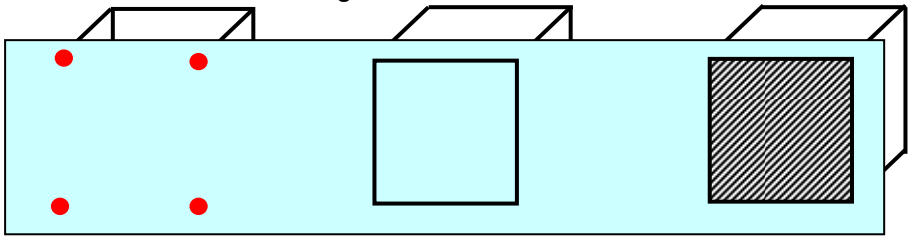


Abb. 3.2.9: Ergebnisse eines Schnitts durch ein Draht-, ein Flächen- und ein Volumenmodell

Abb. 3.2.9 zeigt in einer Gegenüberstellung die verschiedenen Ergebnisse, die beim Schnitt einer Ebene mit einem Kanten-, Flächen- und Volumenmodell entstehen können. Es sind dies Punkte, Linien bzw. Flächen.

Zusatzinformation zu den Volumenmodellen

Das Volumenmodell stellt für Körper das einzige wirklich eindeutige Modell dar. Für jeden Punkt des Raumes ist entscheidbar, ob dieser zum Inneren des Körpers gehört, oder nicht. Schnittflächen können vollautomatisch schraffiert werden. Verdeckte Kanten können stets automatisch entfernt werden. Volumen- und Massenberechnungen sind möglich. In der Literatur werden die Techniken der Volumenmodellierung in 6 Gruppen eingeteilt. Diese werden im Folgenden kurz beschrieben.

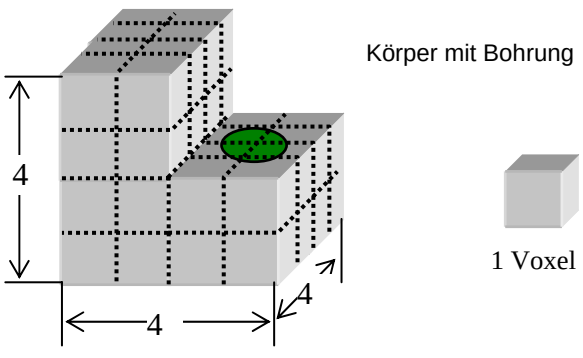


Abb. 3.2.10: Körper durch Voxel approximiert

Enumerationsverfahren ('spatial occupancy matrix')

Prinzip: Bei ***Enumerationsverfahren (spatial occupancy matrix)*** wird der Raum in eine Menge disjunkter Teile (Zellen) unterteilt.

Das Objekt wird dadurch beschrieben, dass angegeben wird, welche dieser Zellen belegt sind.
Hierbei kann ein fest vordefiniertes Raumgitter verwendet werden (Abb. 3.2.10). Alle Zellen besitzen dann dieselbe Form und Größe. Der Aufwand für eine Repräsentation kann gewaltig sein.

Anwendung: Z.B. in der Biomedizin (Computertomographie). Die ***Raumzellen*** werden auch **'voxel'** = volume element genannt.

Zellzerlegung ('cell decomposition')

Prinzip (***Zellzerlegung, 'cell decomposition'***):
Der Körper wird aus einfachen (parametrisierten) Bausteinen (Zellen) zusammengesetzt (Abb. 3.2.11). In einem Repräsentationsschema wird eine Menge solcher Zellen (parametrisierte Primitive) definiert. Die Zellen können auch gekrümmte Oberflächen aufweisen.

Anwendung: Bei der Finite Elemente Methode (FEM). Die Erzeugung dieser Repräsentation erfolgt häufig aus anderen Darstellungsformen (Repräsentationsschemata) heraus.

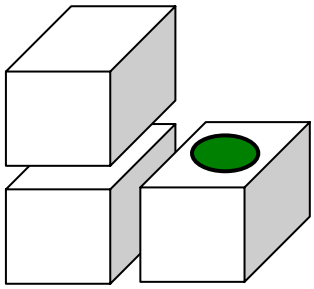
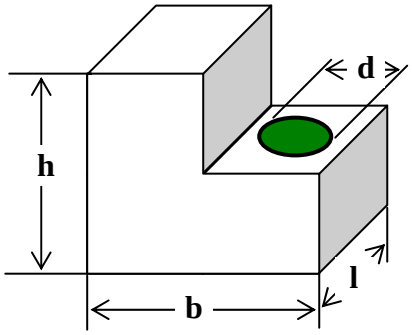


Abb. 3.2.11: Zellzerlegung

Parametrisierung ('primitive instancing')

Prinzip (***Parametrisierung, 'primitive instancing'***):
Mit Hilfe variabler Größen und Abmessungen werden ***Teilefamilien*** **(*Varianten*)** und Baugruppen beschrieben. Jedes Element wird durch eine feste Anzahl von Parametern beschrieben (Abb. 3.2.12). Ein parametrisiertes primitives Element wird als Teilefamilie (Prototyp) angesehen. Die Mitglieder der Familie unterscheiden sich nur durch die Parameterwerte.



	l	b	h	d
Variante 1	100	80	80	20
Variante 2	100	100	100	20
Variante 3	....	....	....	....

Abb. 3.2.12: Körper als Mitglied einer Teilefamilie

Anwendung: Bei der Konstruktion komplexer Gebilde, da die Mitglieder der Teilefamilie sehr einfach beschrieben werden können. Das Schema begünstigt die Standardisierung, ist aber unflexibel.

Sweeping

Prinzip (***Sweeping***):

Eine Fläche (oder ein Volumen) wird im Raum bewegt. Der dabei überstrichene Bereich gehört zum Inneren des Körpers. Bei diesem sehr gebräuchlichen Verfahren können einige Sonderfälle unterschieden werden.

Sonderfall 1: ***Profilkörper (=Translationskörper)***
Der Übergang 2D ➔ 3D erfolgt durch Hinzufügen einer Tiefe ("2.5 D"). Die Leitgerade (das ist die Gerade entlang der die Bewegung der Fläche erfolgt) kann senkrecht oder schräg auf der Fläche stehen (Abb. 3.2.13).

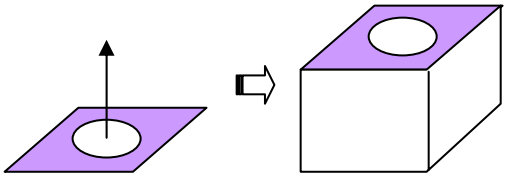


Abb. 3.2.13: Beispiel Profilkörper

Sonderfall 2: ***Rotationskörper***
Der Übergang 2D ➔ 3D erfolgt durch eine Drehung (Abb. 3.2.14)

Allgemeines Sweeping:
Eine beliebige Fläche (oder Volumen) wird entlang einer beliebigen Bahnkurve bewegt (Abb. 3.2.15). Dies ist die allgemeinste Form des Sweeping.

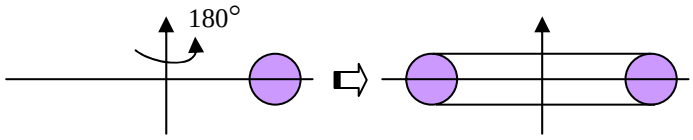


Abb. 3.2.14: Beispiel Rotationskörper

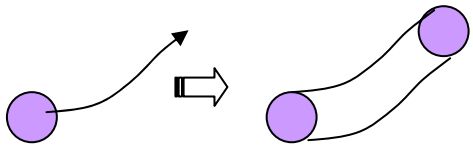


Abb. 3.2.15: Beispiel Allgemeines Sweeping

Anwendung: Die Sonderfälle (Translations- und Rotationskörper) kommen häufig vor. Die Beschreibung ist einfach. Das Schema wird oft zur Eingabe, aber weniger häufig zur internen Repräsentation verwendet. Es taucht oft im Zusammenhang mit anderen Repräsentationsschemata auf.

Flächenbegrenzungsdarstellung ('boundary representation')

Prinzip (***Flächenbegrenzungsdarstellung, 'boundary representation, BREP'***):

Die (geschlossene) Oberfläche eines Körpers teilt den Raum in 2 Teile, von denen der eine zum Inneren, der andere zum Äußeren des Körpers gehört.

Die Oberfläche kann durch Facetten beschrieben werden (s. Flächenmodell). Man spricht dann auch hier von einem Facettenmodell. Die Facetten sind grundsätzlich orientiert, d.h. sie besitzen eine Außen- und eine Innenseite. Sie werden durch Kanten begrenzt.

Im Beispiel der Abb. 3.2.16 wird der Körper durch 8 ebene Flächen (2 davon mit Loch) und durch einen Zylindermantel begrenzt.

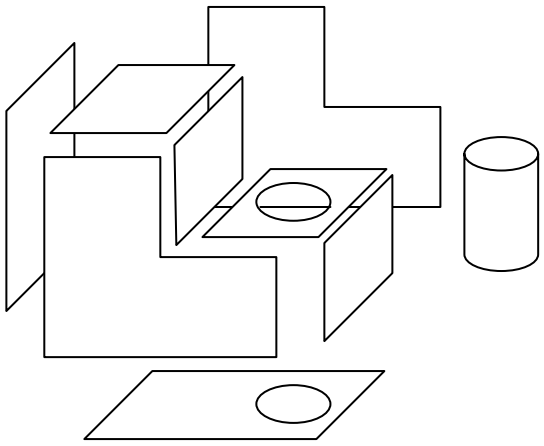


Abb. 3.2.16: Beispiel BREP

Anwendung: Die BREP ist das heute gebräuchlichste primäre Repräsentationsschema. Die Beschreibung eines Körpers bei der Eingabe durch alle seine Flächenteile und ihre Beziehungen zueinander ist allerdings häufig zu aufwendig. Die Eingabe erfolgt daher häufig in anderer Form (z.B. Sweeping oder CSG).

Festkörperdarstellung ('Constructive Solid Geometry)

Prinzip (***Festkörperdarstellung, 'Constructive Solid Geometry'***):

Das Objekt wird aus (einfachen) Grundkörpernaufgebaut. Diese werden als Mengen von Punkten aufgefasst. Die Verknüpfung zu komplexen Körpern erfolgt durch ***Mengenoperationen*.**

Abb. 3.2.17 zeigt eine CSG-Hierarchie. Der Körper wird durch Vereinigung zweier Quader und anschließender Differenzbildung mit einem Zylinder (alle 3 Grundelemente sind Körper) gebildet.

Anwendung: Die CSG-Repräsentation ist das modernste Repräsentationsschema.

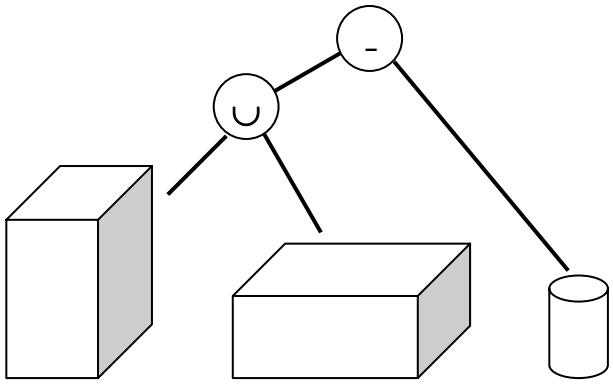


Abb. 3.2.17: Beispiel CSG

4. Graphische Programmierung

Um eine einfache 3D - Szene auf einem Ausgabegerät darzustellen, sind eine Reihe von Schritten erforderlich, die in diesem Kapitel besprochen werden sollen. Die Erstellung des Modells und die Definition von views werden am Beispiel der 3D - Graphik - API OpenGL und dem Dateiformat VRML erläutert.

4.1 OpenGL

OpenGL (Open Graphics Library) ist eine Programmbibliothek (API) für 3D-Graphik. Es ist aus der GL von Silicon Graphics (SGI) weiterentwickelt worden. OpenGL ist heute ein de facto Standard. Die OpenGL Bibliothek enthält etwa 120 Funktionen (die in Variationen verwendbar sind). OpenGL ist dazu geeignet, realistische Graphiken in Echtzeit zu erzeugen.

OpenGL ist Plattform-unabhängig (unabhängig von einem bestimmten Window System). Es ist so konzipiert, dass auch in einer Netzumgebung effizientes Arbeiten möglich ist. Ein Anwendungsprogramm (Client, z.B. PC) kann auf einem Rechner ausgeführt, und die 3D - Graphik auf einer anderen Maschine (Server, z.B. Hochleistungs-Graphik Workstation) ausgegeben werden.

Da OpenGL kein Modellierungssystem ist, müssen komplexe Modelle aus einfachen graphischen Primitiven aufgebaut werden. Stattdessen ist OpenGL bezüglich seiner Performance optimiert. Aus diesem Grund und wegen der fehlenden Eingabemöglichkeiten wird OpenGL auch als Rendering System bezeichnet. OpenGL - Funktionen werden "sofort" ausgeführt. Diese Arbeitsweise bezeichnet man als "***immediate mode***" (immediate = sofort). Vergleiche hierzu die Arbeitweise von VRML (Kap. 4.2), wo die graphische Information in einer Datenstruktur abgelegt wird und diese "irgend wann" ausgeführt (interpretiert) wird (***retained mode***; retain = bewahren, festhalten). OpenGL ist nicht objektorientiert konzipiert.

4.1.1 Definition von Primitiven

Die in OpenGL verwendeten ***graphischen Primitive***, sind in Abb. 4.1 dargestellt.

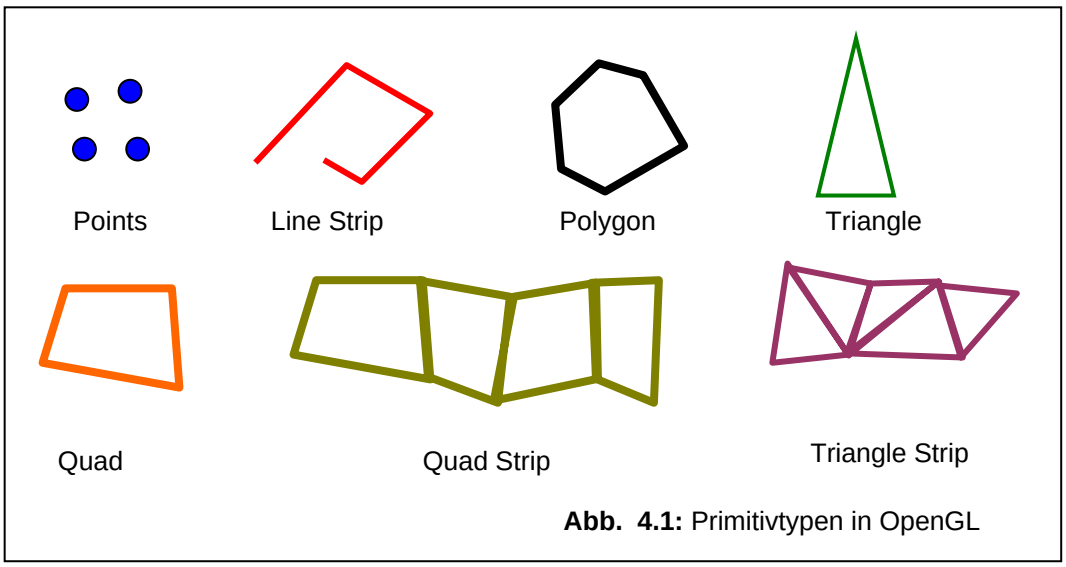


Abb. 4.1: Primitivtypen in OpenGL

Die Definition eines Quadrats, das als Polygon definiert wird, soll die Anwendung veranschaulichen.

Abb. 4.2 zeigt links den hierzu erforderlichen OpenGL -Code. Das Quadrat erhält die Kantenlänge 100. Der Ursprung des Modellierungskoordinatensystems liegt im Mittelpunkt der Fläche. Die ***Eckpunkte*** (x,y,z) der 4 Eckpunkte (***Vertices***) werden in den Funktionen **glVertex3f** definiert. Jede Ecke erhält im Beispiel eine eigene Farbe. Ziel dieses Beispiels ist es, im Folgenden die Hülle eines RGB - Würfels zu erstellen.

```
...
GLfloat KL=100.0f;
...
glBegin ( GL_POLYGON );  //Vorderseite
  glColor4f ( 1.0f, 0.0f, 0.0f, 1.0f ); //ROT
  glVertex3f ( -KL/2.0f, -KL/2.0f, +KL/2.0f );
  glColor4f ( 1.0f, 1.0f, 0.0f, 1.0f );   //GELB
  glVertex3f ( +KL/2.0f, -KL/2.0f, +KL/2.0f );
  glColor4f ( 1.0f, 1.0f, 1.0f, 1.0f );   //WEISS
  glVertex3f ( +KL/2.0f, +KL/2.0f, +KL/2.0f );
  glColor4f ( 1.0f, 0.0f, 1.0f, 1.0f );   //MAGENTA
  glVertex3f ( -KL/2.0f, +KL/2.0f, +KL/2.0f );
glEnd ( );
```

Abb. 4.2: Definition eines Primi-

Wir wollen nun annehmen, dass in der Funktion Wuerfel() auch die übrigen 5 Seiten des RGB-Würfels auf dieselbe Weise wie die Vorderseite definiert wurden. Beim Aufruf dieser Funktion beim Rendern würde sich keine Veränderung der Ausgabe gegenüber Abb 4.2 ergeben, da wir den Würfel noch von vorne sehen.

4.1.2 Transformationen

Abb. 4.3: Drehung um die y-Achse

Damit wir sehen, dass der Würfel tatsächlich mehrere Seiten besitzt, drehen wir ihn um 30 Grad um seine y-Achse. Im OpenGL - Programm stellen wir dem Wuerfel - Aufruf eine Rotation voraus:

```
glRotatef(30., 0., 1., 0.);
Wuerfel ( );
```

Die Parameter 0.0, 1.0, 0.1 im Rotationsaufruf sagen aus, dass die Drehung um einen Vektor mit diesen Koordinaten, also hier um die y-Achse, erfolgen soll. Die ***Transformationshierarchie*** ist in Abb. 4.3 rechts neben dem gerenderten Ergebnis dargestellt.

Abb. 4.4: zusätzliche Drehung um die x-Achse um 20°

Da aus der Darstellung immer noch nicht hervorgeht, dass es sich bei dem Würfel um ein 3D- Objekt handelt, fügen wir eine weitere Drehung um die x-Achse um 20° (Abb. 4.4) hinzu:

```
glRotatef(20., 1., 0., 0.);
glRotatef(30., 0., 1., 0.);
Wuerfel ( );
```

Beachten Sie, dass aus der Transformationshierarchie in Abb. 4.4 klar hervorgeht, dass die Drehung um die x-Achse tatsächlich auf das bereits um die y-Achse rotierte Objekt ausgeführt wird. In OpenGL müssen Sie das wie gezeigt lösen. Sie "lesen" dann den Code am besten entweder wie folgt:

- "rotiere um die x-Achse alles was folgt....", oder
- "rotiere um die x-Achse, davor aber um die y-Achse...."

Im nächsten Schritt soll aus dem Würfel ein Quader entstehen, denn am Ende soll ein einfacher Tisch definiert werden. Die Kantenlänge des Würfels war mit 100 ziemlich willkürlich gewählt. Wir wollen dem Tischbein durch Skalierung die Maße 4x4x70 geben. Natürlich hätten wir auch unser Primitiv gleich mit diesen Maßen definieren können.

Abb. 4.5: Gedrehter Quader


```
glRotatef ( 20., 1., 0., 0. );
glRotatef ( 30., 0., 1., 0. );
glScalef ( 0.04, 0.7, 0.04 );
Wuerfel ( );
```

Beachten Sie, dass unser Teilmodell nun der (skalierte) Quader (= Tischbein) ist, und dieses Teilmodell um die beiden Achsen (wie gehabt) gedreht wird (Abb. 4.5). Im nächsten Schritt müssen 3 weitere Tischbeine erzeugt und zusammen mit dem ersten im Teilmodell "Tisch" richtig positioniert werden.

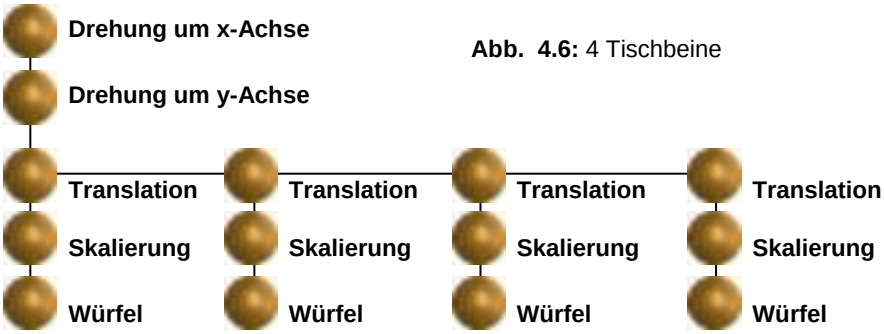
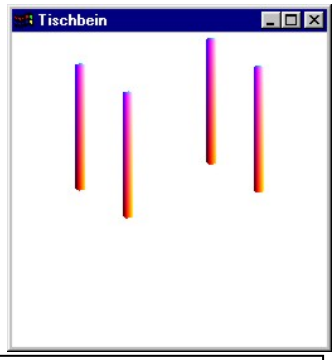


Abb. 4.6: 4 Tischbeine



4.1.3 Entkopplung von Transformationen in einer Hierarchie

Nebenstehend ist der entsprechende Auszug aus dem OpenGL - Programm (mit der ***Entkopplung von Transformationen***) gelistet. Folgendes kann diesem Codebeispiel entnommen werden (Vergleichen Sie dazu die Transformationshierarchie in Abb 4.6):

- Die beiden Rotationen werden auf das gesamte Teilmodell "4 Tischbeine" angewendet.
- Jeder der 4 skalierten Quader wird durch eine Translation an die richtige Position verschoben. In y-Richtung wird um die halbe Tischbeinhöhe, nämlich 35 verschoben, damit das Teilmodell schon auf dem gedachten Boden in der Höhe y=0 steht.
- Die Funktion glPushMatrix() "rettet" die Drehung um die beiden Achsen auf den sogenannten Matrix - Stack

```
glRotatef (20.0f, 1.0f, 0.0f, 0.0f );
glRotatef (30.0f, 0.0f, 1.0f, 0.0f );
glPushMatrix ( );
glTranslatef (-40.0f, 35.0f, 25.0f );
glScalef (0.04f,  0.7f,  0.04f );
Wuerfel ( );
glPopMatrix ( );
glPushMatrix ( );
glTranslatef (+40.0f, 35.0f, 25.0f );
glScalef (0.04f,  0.7f,  0.04f );
Wuerfel ( );
glPopMatrix ( );
glPushMatrix ( );
glTranslatef (+40.0f, 35.0f, -25.0f );
glScalef (0.04f,  0.7f,  0.04f );
Wuerfel ( );
glPopMatrix ( );
glPushMatrix ( );
glTranslatef (-40.0f, 35.0f, -25.0f );
glScalef (0.04f,  0.7f,  0.04f );
Wuerfel ( );
glPopMatrix ( );
```

- Die Funktion glPopMatrix() holt die zuletzt auf dem ***Matrix-Stack*** abgelegte ***Transformationsmatrix*** (s. Kap.10) zurück und macht sie wieder "wirksam".
- Ohne die Anwendung der beiden Funktionen **glPushMatrix()** und **glPopMatrix()** würden sich alle Translationen und Skalierungen "aufaddieren", so wie das ja schon beim einzelnen Tischbein der Fall war! In der Transformationshierarchie können Sie das Zusammenwirken mehrerer Transformationen daran ablesen, dass die Knoten untereinander stehen. Das Aufsetzen auf früheren Zuständen (Entkoppelung) erkennen Sie an der Anordnung der Knoten in waagrechter Richtung.

Nun soll der Tisch auch noch eine geeignete Deckplatte erhalten, oder wie sollte man sonst daran arbeiten? Wir verwenden nochmals den Würfel, skalieren ihn und verschieben ihn an die richtige Stelle. Der Graph für die Transformationshierarchie ändert sich nur dadurch, dass noch 3 weitere Knoten Translation, Skalierung und Würfel rechts angefügt werden. Entsprechend erweitert sich der Code wie folgt:

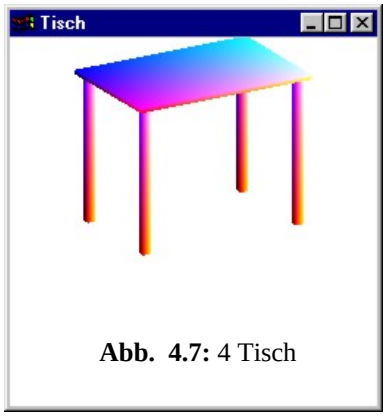


Abb. 4.7: 4 Tisch

```
glPushMatrix ( );
glTranslatef (0.0f, 71.0f, 0.0f );
glScalef (0.9f,  0.02f,  0.6f );
Wuerfel ( );
glPopMatrix ( );
```

Da wir nun mehrere Tische erzeugen wollen, fassen wir den obigen Code, bis auf die beiden obersten Rotationen in der Funktion **Tisch ( )** zusammen. Der Aufruf lautet nun:

```
glRotatef (20.0f, 1.0f, 0.0f, 0.0f );
glRotatef (30.0f, 0.0f, 1.0f, 0.0f );
Tisch ( );
```

An der Darstellung ändert sich nichts.

```
glRotatef (90.0f, 1.0f, 0.0f, 0.0f );
glScalef (0.25f,  0.25f,  0.25f );

glPushMatrix ( );
glTranslatef (-60.0f, 0.0f, 80.0f );
Tisch ( );
glPopMatrix ( );

glPushMatrix ( );
glTranslatef (+60.0f, 0.0f, 80.0f );
Tisch ( );
glPopMatrix ( );

glPushMatrix ( );
glTranslatef (-180.0f, 0.0f, 0.0f );
glRotatef (-45.0f, 0.0f, 1.0f, 0.0f );
Tisch ( );
glPopMatrix ( );

glPushMatrix ( );
glTranslatef (+180.0f, 0.0f, 0.0f );
glRotatef (+45.0f, 0.0f, 1.0f, 0.0f );
Tisch ( );
glPopMatrix ( );

glPushMatrix ( );
glTranslatef (0.0f, 0.0f, -80.0f );
Tisch ( );
glPopMatrix ( );
```

Um die übrigen Tische hinzuzufügen und dies optisch kontrollieren zu können, verkleinern wir die Tische (Skalierung der Szene auf 1/4 der Größe in allen Richtungen) und sehen uns die Szene von oben an (Rotation um die x-Achse um 90 Grad). Darauf folgen die bereits bekannten Sequenzen Translation und Aufruf des Teilmodells, eingerahmt von Push und Pop. Das Ergebnis ist in Abb. 4.8 zu sehen. Die Transformationshierarchie ist in Abb. 4.9 dargestellt, wobei sich hinter "Tisch" die bereits bekannte Hierarchie verbirgt.

Abb. 4.8: Tischgruppe von oben

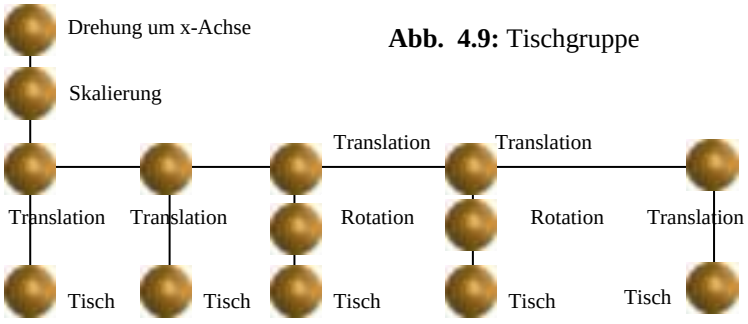


Abb. 4.9: Tischgruppe

4.1.4 Kamera

Skalierung und Rotation, die in Abb. 4.8 den Blick von oben auf die Tischgruppe ermöglichen, stellen logisch gesehen eine Sicht (View) auf das Modell (die Szene) dar. Sie sind ja eigentlich nicht zur Modelldefinition erforderlich. OpenGL stellt Funktionen zur Verfügung, die die Definition einer View und eines Frustums besser unterstützen. Die Anwendung derartiger Funktionen soll im Folgenden demonstriert werden.

Abb. 4.10: Kamera von vorne

Werden die beiden erwähnten Rotationen aus dem Code entfernt, so erhält man eine Darstellung der Szene wie in Abbildung 4.10 dargestellt. Dies kommt dadurch zustande, dass im Programm bereits eine Kamera verwendet wurde, ohne dass wir darüber gesprochen haben. Die Einstellung dieser Kamera ***(innere Kameraparameter)*** beschreibt ein ***Frustum*** und wird über die Funktion ***glOrtho*** vorgenommen. "Ortho" soll aussagen, dass es sich bei der Projektion um eine ***orthographische Projektion*** (=senkrechte Parallelprojektion) handelt. Diese Projektionsart verzerrt im Gegensatz zu einer ***perspektivischen Projektion*** Objekte nicht. Sie entspricht einer unendlich großen Kamerabrennweite. Das Frustum hat dann die Form eines Quaders, dessen Abmessungen (xlinks, xrechts, yunten, yoben, zvorne, zhinten) sind. In unserem Fall lautet der Aufruf:

```
glOrtho ( -KL*2.5, +KL*2.5, -KL*2.5, +KL*2.5, -40*KL, +40*KL );
```

Durch dieses Frustum wird also ein Ausschnitt der Welt beschrieben, der einen Querschnitt von (5*KL)x(5*KL) Einheiten besitzt und damit die gesamte Szene umfassen kann. Die Tiefe des Frustums ist 80*KL. Für die Orientierung der Kamera wählen wir die Defaulteinstellung von OpenGL. (Orientierung in Richtung der negativen z-Achse des WKS).

Nun soll die Kameraposition und -Orientierung (***äußere Kameraparameter***) also die ***view*** verändert werden. Hierfür können wir die Funktion **gluLookAt** verwenden. Sie besitzt 9 Parameter. Die Gruppe der ersten 3 definiert die (x,y,z)-Position der Kamera (COP). Die Gruppe der nächsten 3 Parameter definiert einen Punkt in der Szene auf den die Kamera gerichtet ist (***Center of Interest*** =***CoI***). Die 3 folgenden Parameter stellen einen Vektor dar, der die Aufwärtsrichtung der Kamera bezügliche des WKS festlegt. Hierzu ein kleines Beispiel: Stellen Sie sich die Beispielszene in der Realität vor: Ein Raum mit 5 Tischen. Die x- und z-Achsen liegen in der Bodenebene des Raumes. Die y-Achse weist nach oben. Um ein Normalphoto im Querformat zu erhalten, müsste die Kameraaufwärtsrichtung also z.B. durch (0.0, 1.0, 0.0) definiert werden. Soll die Aufnahme auf dem Kopf stehen lautet der Vektor (0.0, -1.0, 0.0), wollen Sie ein Photo im Hochformat, so sollte der Vektor z.B. (1.0, 0.0, 0.0) lauten. Die Funktion gluLookAt wird genau an der Stelle des Programms eingefügt, wo zuvor die Skalierung und die Rotation die view definierten.

Mit `gluLookAt (0., 100., 0., 0., 0., 0., 0., 0., -1.);  //von oben`

erhalten wir eine Darstellung wie in Abb 4.11 gezeigt. Die Kamera befindet sich 100 Einheiten über der Tischgruppe.und blickt in den Ursprung des WKS.

gluLookAt (0., 180., -110., 0., 80., 80., 0., 1., 0.); //von schräg oben

versetzt den Betrachter (Kamera) hinter den Einzeltisch (-110), führt eine Augenhöhe (180) ein, so dass die Szene von schräg oben aus der Gegenrichtung betrachtet werden kann (Abb. 4.12).

Abb. 4.11: Kamera von oben

Abb. 4.12: Kamera von schräg oben

Abb. 4.13: Weitwinkelobjektiv

Abb. 4.14: Normalobjektiv

In einem letzten Schritt soll die orthographische Kamera nun durch eine perspektivische ersetzt werden. Hierzu ersetzen wir die Funktion glOrtho durch gluPerspective. Durch diese wird ein (echtes) Frustum in Form eines Pyramidenstumpfs definiert. Die Funktion besitzt 4 Parameter. Diese sind der Reihe nach: Ein Winkel (Öffnung von der Spitze aus nach oben gemessen), das Seitenverhältnis (aspect ratio) des Pyramidenquerschnitts, den Abstand der vorderen Clipping - Ebene vom COP (ungleich =0!) und der Abstand der hinteren Clipping - Ebene vom COP.

gluPerspective (90.0, 1.0, 1.5*KL, 40*KL);

liefert Abb. 4.13, in Abb. 4.14 wurde ein kleinerer Öffnungswinkel (60 Grad) gewählt. Bei den beiden letzten Bildern wurde der Betrachterstandpunkt leicht erhöht. Deutlich sind die stärkeren perspektivischen Verzerrungen in Abb. 4.13 gegenüber 4.14 zu erkennen.

4.2 VRML / X3D

VRML (***V****irtual* ***R****eality* ***M****odeling* ***L****anguage*) verwendet WEB Technologie und wurde entwickelt um 3D Graphik in das Worldwide Web einzubringen. VRML basiert auf der Open Inventor Klassenbibliothek. Die Beschreibung der Szene erfolgt in einem hierarchisch aufgebauten ***Szenengraph***. In der hier verwendeten Notation erscheinen untergeordnete Kindknoten rechts. So sind in dem Szenegraph der Abbildung 4.15 dem Shape - Knoten ein Appearance - Knoten und ein Box - Knoten untergeordnet. Eine Ebene unterhalb des Appearance - Knotens existiert ein Kindknoten Material. Das dabei definierte Objekt wird also durch sein Aussehen (appearance) und seine spezielle Form (box = Quader) definiert. Die einem Knoten zugeordneten Werte werden in so genannten Feldern (fields) definiert. Diese sind in den Szenegraphen hier nicht aufgeführt. Diese Felder können z.B. direkte Werte enthalten (im Beispiel Abb. 4.15: size mit den Abmessungen 0.04, 07 und 0.04 in x-, y- und z-Richtung) oder sie enthalten Knoten wie das Feld material den Material - Knoten.

```
#VRML V2.0 utf8
# Szene01: Das Tischbein weiß
Shape {
  appearance Appearance {
    material Material { }
  }
  geometry Box {size 0.04 0.7 0.04}
}
```

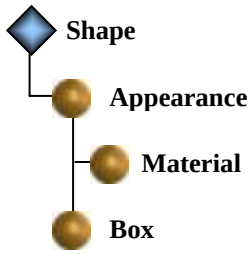


Abb. 4.15: Tischbein

Um einen ersten Eindruck zu vermitteln wie eine 3D - Szene in VRML beschrieben werden kann, soll noch einmal das in Kapitel 4.1 verwendete Beispiel dienen.

Die Konstruktion der Tischgruppe soll wieder bottom - up erfolgen. Die Definition des Tischbeins in den richtigen Abmessungen erfolgt in einem nahezu minimalen VRML - file. Der Szenegraph (Abb. 4.15) enthält lediglich einen Shape - Knoten (shape = Form). Das Feld geometry enthält einen Quader (= Box) mit den gewünschten Abmessungen. Eine Farbe ist nicht definiert. Der vordefinierte Standardwert ist ein helles grau (0.8 0.8 0.8).

Soll das Tischbein die Farbe ROT erhalten, wird das Feld diffuseColor des Materialknotens wie folgt gesetzt: material Material { diffuseColor 1.0 0.0 0.0 }.

```
#VRML V2.0 utf8
#  4 rote Tischbeine

Transform  {
 children [
   Transform {
     translation   -0.4 0.35 0.25
     children
     DEF Tischbein Shape {
       appearance Appearance {
         material Material { diffuseColor 1.0 0.0 0.0}
       }
       geometry Box {size 0.04 0.7 0.04}
     }
   }

 Transform {
  translation    0.4 0.35 0.25
  children
  USE Tischbein
 }

 Transform {
  translation    0.4 0.35 -0.25
  children
   USE Tischbein
 }

 Transform {
   translation   -0.4 0.35 -0.25
   children
   USE Tischbein
 }

 ]
}
```

Im nächsten Schritt sollen nun 4 Tischbeine in geeigneten Positionen angeordnet werden. Hierzu werden 4 Translationen benötigt, die in jeweils einem Tranform - Knoten definiert werden (s. translation - Feld nebenstehend).

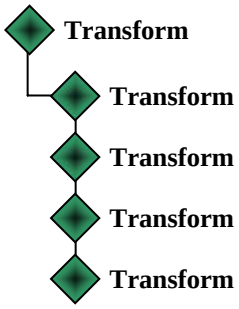


Abb. 4.16: 4 Tischbeine

Der zugehörige Szenegraph ist in Abb. 4.16 dargestellt. Es ist nicht erforderlich das (jeweils gleich aussehende) Tischbein in jedem Transform - Knoten erneut zu definieren. Die Definition (DEF) erfolgt einmalig im ersten Knoten und wird mit einem Namen (hier: "Tischbein") versehen. In den darauf folgenden Knoten kann nun mit Hilfe von USE auf diese Definition zurückgegriffen werden (Exemplare von "Tischbein"). Der oberste Transform - Knoten hält die untergeordneten 4 Transform - Knoten auf einer Ebene mit Hilfe von children [...] zusammen.

DEF und USE können nur verwendet werden, wenn alle Exemplare identisches Aussehen besitzen, in unserem Fall, wenn sie in der Größe und Farbe übereinstimmen. Im Folgenden soll gezeigt werden, wie vorgegangen werden kann, wenn diese Voraussetzung nicht gegeben ist. In unserem Beispiel sollen die Tischbeine verschiedene Farben erhalten. VRML stellt für solche Fälle den sog. Prototyp (PROTO - Knoten) zur Verfügung. Dort werden Felder festgelegt, deren Werte erst beim Aufruf des Prototyps definiert werden. Hier im Beispiel ist dies das Feld für die Farbe, das den Namen "tischbeinFarbe" und den Standardwert ROT erhält:

```
PROTO Tischbein   [ field SFColor tischbeinFarbe 1.0 0.0 0.0 ]  #Default-Wert
```

```
#VRML V2.0 utf8
#  Einzeltisch:

PROTO Tischbein   [
        field SFColor tischbeinFarbe 1.0 0.0 0.0
] {
  Shape {
    appearance Appearance {
      material Material { diffuseColor IS tischbein
Farbe }
}
geometry Box {size 0.04 0.7 0.04 }
}
}
```

Der gesamte Prototyp - Knoten, der am Anfang des files eingefügt wurde, sieht dann wie nebenstehend aus.

Der Aufruf des Prototyps findet wie die Verwendung von USE im Transform - Knoten statt:

```
Transform {
   translation  0.4 0.35 0.25
   children
     Tischbein {tischbeinFarbe
                0.0 1.0 0.0
     }
}
```

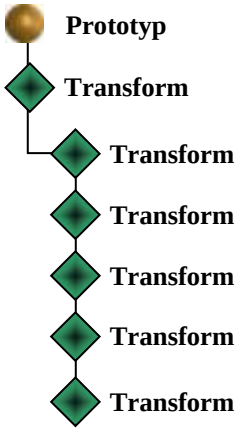


Abb. 4.17: Einzeltisch

Abbildung 4.17 zeigt nun den Szenegraph des kompletten Tisches. Durch Hinzufügen der Tischplatte ist ein weiterer Transform-Knoten hinzugekommen. Dieser muss untergeordnet einen Shape - Knoten enthalten, da die Abmessungen des Quaders im Prototyp nicht parametrisiert wurden. 4 der Transform - Knoten enthalten Aufrufe des Prototyps.

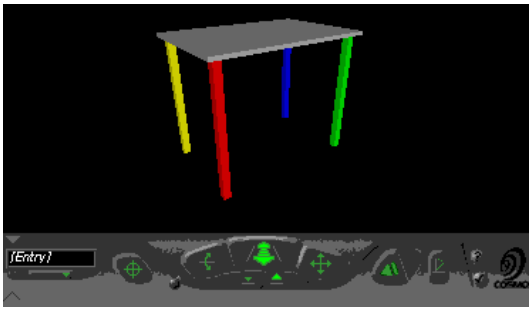


Abb. 4.18: Tisch

Abb. 4.18. zeigt den Tisch (gedreht und vergrößert) wie er durch einen Browser (retained mode) dargestellt wird. Das 4. Bein ist (da es im Original blau ist) über dem schwarzen Hintergrund im Druck kaum sichtbar. Beachten Sie die perspektivische Verzerrung.

Es bleiben nun noch 2 Aufgaben zu erledigen. Erstens müssen 5 gleiche Tische in der wie im vorigen Kapitel gezeigten Anordnung positioniert werden. In einem (hier) letzten Schritt sollen verschiedene Views auf die Tischgruppe definiert werden.

Der zuvor definierte Einzeltisch wurde in dem VRML - file "Einzeltisch.wrl" abgespeichert. Auf diesen file kann nun ein link (über das url - Feld eines Inline - Knotens) definiert werden. Wie die Bezeichnung "url" vermuten lässt kann statt eines lokalen files auch eine beliebige Adresse im Internet stehen.

```
Transform     {
        translation -0.6 0 0.8
        children  [Inline {
          url ["Einzeltisch.wrl"]

        }]
}
```

```
#VRML V2.0 utf8

#  Tischgruppe mit views und Hintergrundfarbe

Background      {
        skyColor            [1 1 1 ]
}

Viewpoint {
        position  0 0 3
        orientation  0 0 1 0
        fieldOfView 0.785398
        description "Von vorne"
}

Viewpoint {
        position  0 10 0
        orientation 1 0 0 -1.571
        fieldOfView 0.785398
        description "Von oben"
}

Viewpoint {
        position  0 0 -10
        orientation  0 1 0  3.14159265359
        fieldOfView  0.785398
        description "von hinten"
}

.....
```

Nebenstehend wurden dem VRML- file nun noch 4 verschiedene Views hinzugefügt. Auf die genaue Bedeutung der Felder im Viewpoint - Knoten wird hier nicht eingegangen. Browser erlauben in der Regel einen Wechsel zwischen den unterschiedlichen Ansichten.

Wegen der Druckdarstellung wurde die Hintergrundfarbe weiss gewählt.

Die Ergebnisse der 4 views sind in der Abbildung 4.19 dargestellt. Es wurden dabei jeweils Ausschnitte aus dem Browserfenster gebildet. Betrachterposition und Blickrichtung wurden jedoch nicht geändert.

von vorne

von oben

von hinten

Abb. 4.19: verschiedene views

5. Visualisierungstechniken

"***Visualisieren***" bedeutet soviel wie "sichtbar machen", oder auch "darstellen". Die Aufgabe der graphischen Datenverarbeitung beschränkt sich dabei jedoch nicht (wie die eines Photoapparates) auf die Sichtbarmachung von Objekten, die in unserer Alltagswelt materiell vorhanden sind bzw. sein könnten (vgl. Kapitel 1). Das folgende Kapitel beschäftigt sich zunächst mit der Visualisierung von Objekten dieser Art, und geht anschließend auf einige Techniken ein, die die graphische Darstellung von Objekten behandelt, die keinerlei oder nur in geringfügigem Maße geometrische Informationen beinhalten.

5.1. Visualisierung geometrischer Modelle

Wenn wir davon ausgehen, dass die Modelle die darzustellen sind 3-dimensionaler Natur sind, können wir zunächst eine einfache Unterscheidung treffen: Sollen bei der Visualisierung der Objekte die Gesetze der Optik (also die Einwirkung von Licht (Art, Farbe, Position der Lichtquelle) und die Eigenschaften des Materials (spiegelnd, brechend, transparent) eine Rolle spielen oder nicht. Lassen wir diese Einflüsse außer Betracht, wird auch eine komplexe Szene schneller darzustellen sein, als wenn Lichteinwirkungen eine Rolle spielen.

5.1.1. Visualisierung ohne Berücksichtigung von Licht

Diese Visualisierungsmethode wurde bereits im Abschnitt 9.1 verwendet. Das Prinzip ist sehr einfach: Für eine Objektoberfläche (repräsentiert durch graphische Primitive) werden eine oder mehrere Farben definiert. Dies geschieht im Allgemeinen durch Angabe der RGB-Werte. In OpenGL erfolgt die Zuordnung der Farben zu Eckpunkten (vertices) (s. Abb. 5.1):

```
Beginn der Primitivdefinition
  Setzen einer aktuellen Farbe
  Definition des 1. Vertex
  .....
  Definition des n-ten Vertex
Ende der Primitivdefinition
```

Abb. 5.1: OpenGL - Primitiv mit einheitlicher Farbe

Handelt es sich bei dem Primitiv in Abb. 5.1 um GL_LINE_STRIP, so hat der gesamte Streckenzug eine einheitliche Farbe. Handelt es sich um GL_POINTS, so erhalten alle Punkte der Punktmenge dieselbe Farbe. Ist das Primitiv z.B. GL_POLYGON, so wird die gesamte Fläche mit der einheitlichen Farbe ausgefüllt.

Nun wird die aktuelle Farbe aber den nachfolgenden Eckpunkten zugeordnet (s. oben). Was also, wenn das Primitiv zwei Vertices enthält, dem ersten davon die Farbe ROT, dem 2. die Farbe BLAU zugeordnet wird?

5.1.1.1. Lineare Interpolation

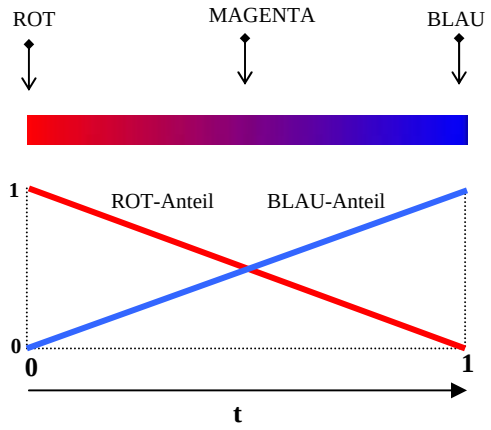


Abb. 5.2: Lineare Interpolation der Farbe

Die Antwort, die OpenGL auf die obige Frage gibt, lautet: ***Lineare Interpolation***. Dieses Prinzip soll am Beispiel der Abbildung 5.2 erläutert werden. Ausgehend vom 1. Eckpunkt nimmt der Rot-Anteil der Farbe zum 2. Eckpunkt hin linear ab. Gleichzeitig steigt der Blau-Anteil der Linie (von 0 beginnend) linear auf seinen Endwert 1.

Mathematisch wird dieser Sachverhalt wie folgt formuliert:

$$F(t) = (1-t) * F(0) + t * F(1)$$

Hierbei ist F(t) der Wert, der sich für eine Farbe an der Stelle t ergibt. Der Parameter t steht für die normierte Länge der Strecke und geht von t=0 bis t=1.

Beispiel: Die Farbe F_0 im ersten Punkt sei [R_0=1, G_0=0, B_0=0] und F_1 des 2. Punktes [R_1=0, G_1=0, B_1=1] (vgl. Abb. 5.2), so ergibt sich

wie zu erwarten für den Startpunkt (t=0): F(0) = (1-0)*[1 0 0] + 0*[0 0 1] = [1 0 0] (rot),

ebenso für den Endpunkt: (t=1): F(1) = (1-1)*[1 0 0] + 1*[0 0 1] = [0 0 1] (blau),

und für die Mitte der Strecke: (t=½): F(½) = (1-½)*[1 0 0] + ½*[0 0 1] = [½ 0 ½]
(Dies ist ein dunkles Magenta.)

Beispiel: Die Farbe im ersten Punkt sei [R_0=1, G_0=1, B_0=0] und die des 2. Punktes (R_1=0, G_1=1, B_1=1) (vgl. Abb. 5.2), so ergibt sich

wie zu erwarten für den Startpunkt (t=0): F(0) = (1-0)*[1 1 0] + 0*[0 1 1] = [1 1 0] (gelb),

ebenso für den Endpunkt: (t=1): F(1) = (1-1)*[1 1 0] + 1*[0 1 1] = [0 1 1] (cyan)

und für ein Viertel der Strecke: (t=¼): F(¼) = (1-¼)*[1 1 0]+ ¼*[0 1 1] = [¾ 1 ¼]
Dies ist ein ... ja was wohl? Probieren Sie es aus!)

5.1.1.2. Bilineare Interpolation

Als nächstes stellt sich die Frage, wie die Farbe im Inneren eines Polygons ermittelt wird, wenn sich die Farben der Eckpunkte unterscheiden. Erinnern wir uns daran, dass Graphiksysteme Flächen zum Zweck der Darstellung i. A. in Dreiecke zerlegen. Wir werden daher die Lösung des Problems am Beispiel eines Dreiecks betrachten. Hierbei ist es nützlich sich an die Vektor-Rasterkonvertierung von Dreiecken zu erinnern.

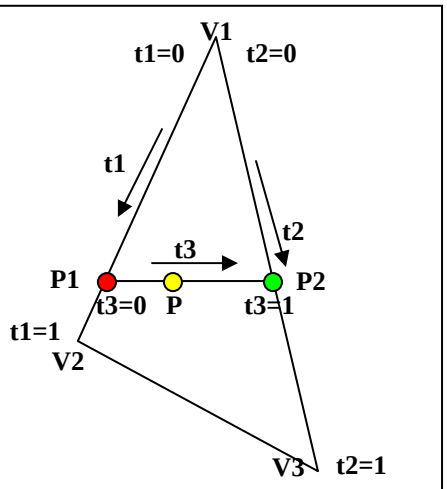


Abb. 5.3: Bilineare Interpolation der Farbe

Die Berechnung der Farbe eines Pixels im Dreieck erfolgt in 2 Stufen (Abb. 5.3):

1. Stufe: Bei der Konvertierung wird das Dreieck zeilenweise (beginnend mit kleinen y-Werten) bearbeitet. Durch lineare Interpolation wird die Farbe in den Punkten P1 (Parameter t1 zwischen den Vertices V1 und V2) und P2 (Parameter t2) ermittelt (s. o.). Hierzu sind also 2 lineare Interpolationen entlang zweier Dreiecksseiten erforderlich.

2. Stufe: P1 und P2 (deren Farben ja nun bekannt sind) werden linear interpoliert (Parameter t3). Hieraus ergibt sich die Farbe jedes Punktes P auf der Scan-Zeile.

Man nennt dies eine ***bilineare Interpolation***, da nach der Interpolation auf der Kontur (lineare Interpolation) ein Übergang in das Innere der Fläche erfolgt.

5.1.2. Rendering

Im Folgenden soll kurz über Techniken gesprochen werden, bei denen die Beleuchtungsverhältnisse und die Art und Weise wie Materialien darauf reagieren eine Rolle spielen. Die (möglichst realistisch wirkende) Visualisierung einer derartigen Szene wird als ***Rendern*** bezeichnet. Unter einem ***Renderer*** versteht man dann die Software oder Hardware, die die Ausführung übernimmt. Die Erstellung qualitativ hochwertiger Bilder für die Produktion kann bis zu mehreren Stunden dauern. Echtzeit-Renderer erzeugen Bilder in Bruchteilen von Sekunden. Als ***Rendering Pipeline*** bezeichnet man die Folge der Einzelschritte, die erforderlich ist, um ein realistisches Bild zu erzeugen. Dies beinhaltet die Umwandlung von Weltkoordinaten in Bildschirmkoordinaten, die Berechnung der Beleuchtung, Aufbringung von Texturen und die Vektor/Rasterkonvertierung.

Der Eindruck, den ein Betrachter vom Aussehen einer bestimmten Stelle einer Objektoberfläche gewinnt, hängt von den verschiedensten Faktoren ab. Zum einen spielt dabei die relative

Lage und Orientierung von Betrachter und Objekt eine Rolle, zum anderen ist die Art der Beleuchtung und die Beschaffenheit der Objektoberfläche entscheidend. Um realistische Darstellungen zu erzielen, müssen diese Gegebenheiten modelliert werden (optische Modellierung). Weiterhin spielt für den visuellen Eindruck auf der Bildschirmoberfläche das Verfahren eine Rolle, das zum Rendern der Objekte eingesetzt wird.

In diesem Rahmen kann über optische Modellierung nicht gesprochen werden. Dennoch ist es möglich, die wichtigsten Rendering-Techniken in ihrer prinzipiellen Arbeitsweise vorzustellen. Um von einem Objekt - im Folgenden ist damit immer ein Teil seiner Oberfläche (i. A. eine dreieckige ***Facette***) gemeint - einen realistischen Eindruck zu erzeugen, wollen wir voraussetzen, dass folgende Einflussgrößen bekannt sind:

Objekt: Position, Ausrichtung, Materialeigenschaften,...
Lichtquellen: Lage, Strahlrichtung, Lichtfarbe und Intensität,...
Betrachter: Position, Blickrichtung

Man kann sich vorstellen, dass die Vielfalt dieser Daten, in teilweise umfangreichen Formeln miteinander verknüpft, zu aufwendigen Berechnungen führen kann - ein Grund, weshalb trotz schneller Hardware, nach wie vor nach neuen effizienten Verfahren zum Rendern komplexer Szenen gesucht wird. Die erste Frage beim Rendern sollte daher immer sein: Welches Verfahren ist anwendbar, um mit dem geringsten Aufwand ein qualitativ akzeptables Ergebnis zu erzielen? Eine Möglichkeit den Aufwand von vornherein zu begrenzen besteht in der Anwendung so genannter ***lokaler Verfahren***:

Diese Verfahren berücksichtigen bei der Berechnung des von einer Objektoberfläche reflektierten Lichtes ausschließlich a) das von den definierten Lichtquellen einfallende Licht und b) die Eigenschaften dieser Oberfläche. Die einzelnen Objekte in der Szene stehen hierbei nicht in Wechselwirkung miteinander. Dies bedeutet, dass keine gegenseitige Beeinflussung der Objekte durch Spiegelung, Schattenwurf und Lichtreflexionen stattfindet.

Im Gegensatz hierzu stehen die ***globalen Verfahren***, bei denen sich die Objekte in der Szene gegenseitig beeinflussen. Hier können Spiegelungen, Schattenwurf und Reflexionen von Licht direkt umgesetzt werden. Man kann diesen Sachverhalt auch dadurch ausdrücken, dass neben den so genannten ***primären Lichtquellen*** (direktes Licht) auch ***sekundäre Lichtquellen*** (also durch Reflexionen von Licht an anderen Objekten verursacht) berücksichtigt werden. Sollen bei Verwendung lokaler Verfahren derartige Effekte erzielt werden, ist dies allerdings über "Tricks" und Umwege auch möglich.

Die lokalen Verfahren sind historisch älter, da sie weniger rechenaufwendig sind. Zu ihnen gehören Schattierungsverfahren wie Gouraud Shading und Phong Shading. Zu den globalen Verfahren zählt das rekursive Ray Tracing und das Radiosity Verfahren.

5.1.2.1. Lokale Verfahren

Im Folgenden wird von ***Shading-Verfahren*** die Rede sein. Der Begriff Shading wird im Deutschen mit Schattierung übersetzt (also: ***Schattierungsverfahren***) und hat nichts mit dem ähnlich klingenden Begriff shadow = Schatten zu tun. Mit Schattierung meint man das Einfärben einer Facettenoberfläche entsprechend den Material-, Beleuchtungs- und Betrachtungsverhältnissen.

Flat Shading

Flat Shading, auch ***Constant Shading*** genannt**,** ist das einfachste Verfahren dieser Art. Die Facette erhält dabei eine einheitliche Farbe. Diese wird z.B. für eine Ecke der Facette ermittelt und dann auf alle Punkte des Inneren der Facette übertragen. Da gekrümmte Oberflächen i. A. durch ebene Facetten approximiert werden (s. Abb. 5.4) liefert dieses einfache Verfahren nur bedingt qualitativ befriedigende Ergebnisse. Die Voraussetzungen für brauchbare Ergebnisse sind gegeben, wenn

- die Lichtquelle sehr weit entfernt ist,
- die Betrachterin von der Facette weit entfernt ist,
- die Oberfläche eben ist.

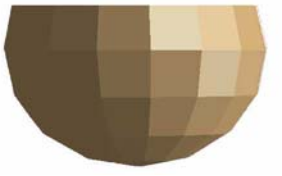


Abb. 5.4: Flat Shading

Gouraud Shading

Das weit verbreitete ***Gouraud Shading*** (sprich: Gur'ooh), ist nicht viel aufwendiger als Flat Shading, liefert jedoch schon erheblich bessere Resultate (s. Abb. 5.5).

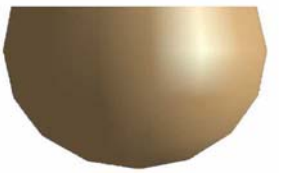


Abb. 5.5: Gouraud Shading

Das Verfahren arbeitet wie folgt:
Die Farbe (Helligkeit) wird in den Eckpunkten der Facette (des Dreiecks) berechnet. Dies erfolgt (wie beim Flat Shading) unter Anwendung der Formeln der optischen Modellierung. Der einzige Unterschied besteht darin, dass statt eines (richtigen) Wertes für die Facette jetzt an 3 Stellen (bei einem Dreieck) "exakte" Werte errechnet werden. Damit diese Aufgabe erfüllbar ist, müssen in den Ecken Normalenvektoren vorgegeben werden. Man bezeichnet diese als ***Eckennormalen***. Zum Unterschied zwischen Eckennormalen und einer ***Flächennormalen*** siehe Abbildung 5.6.

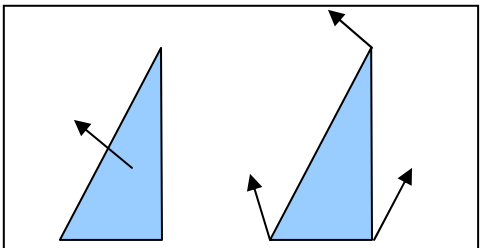


Abb. 5.6: Ecken- und Flächennormalen

Sind die Farben (Helligkeiten) in den Ecken berechnet, so werden die Werte für die übrigen Pixel durch bilineare Interpolation errechnet. Nehmen die Eckennormalen unterschiedliche Werte an (sinnvollerweise sind das die Normalen der Tangentialebenen in diesen Punkten, vgl. Abb. 5.7.), so ergibt sich ein erstaunlich gut glättender Effekt (s. Abb. 5.5.) Die Geometrie weist ja wirklich nur eine geringe Anzahl von Facetten auf - vgl. Abb. 5.4). Beachten Sie, dass die Geometrie unverändert bleibt, also nicht geglättet wird.

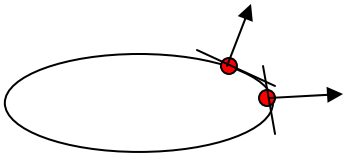


Abb. 5.7: Definition von Eckennormalen

Die Grenzen des Gouraud Shadings werden klar, wenn man sich. überlegt welchen Effekt z.B. ein stark gebündeltes Licht, das auf das Innere der Facette gerichtet ist, hervorruft. Wenn die 3 Ecken kein Licht erhalten ist ihre Helligkeit 0. Werden diese 3 Helligkeiten interpoliert, so bleibt natürlich auch das Innere der Facette dunkel. Eine übliche Abhilfe ist, die gesamte Objektoberfläche feiner zu unterteilen. Damit steigt natürlich auch entsprechend der Rechenaufwand.

Phong Shading

Ein wesentlich rechenaufwendigeres Verfahren stellt das ***Phong Shading*** dar. Der Unterschied zum Gouraud Shading besteht darin, dass dabei für jedes Pixel der Facette die optischen Gesetze Anwendung finden. Hierzu ist es erforderlich, für jedes der Pixel die Normale zu kennen. Diese wird aus den Eckennormalen der Facette ermittelt - wer hätte das für möglich gehalten? Dies geschieht durch bilineare Interpolation der Eckennormalen! Mit diesem Verfahren sind auch so genannte Glanzlichter darstellbar.
OpenGL unterstützt Flat und Gouraud Shading.

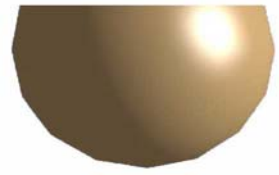


Abb. 5.8: Phong Shading

5.1.2.2. Globale Verfahren

Strahlverfolgung (Ray Tracing)

Mit diesem Verfahren können 'relativ einfach' der Schattenwurf von Objekten, sowie spiegelnd reflektierende und transparente Oberflächen behandelt werden. Die Strahlverfolgung stellt eine aufwendige 'Geradeausmethode' zur Erzeugung sehr realistisch wirkender Darstellungen dar.

Die Grundidee hierbei ist, den Wegen der Lichtstrahlen zu folgen, die die Szene erhellen. Dabei wird berechnet, wie die Strahlen an Oberflächen absorbiert, reflektiert und gebrochen werden. Einige dieser Lichtstrahlen werden am Ende am Blickpunkt (COP) eintreffen und das endgültige Bild erzeugen. Da die meisten Lichtstrahlen die Betrachterin niemals erreichen, wäre es ein unnötig hoher Berechnungsaufwand, die physikalischen Strahlen von der Lichtquelle aus zum Auge hin zu verfolgen (***Forward Ray Tracing***). In der Praxis verfolgen wir in umgekehrter Richtung, ausgehend vom COP, den Sehstrahl auf seinem Weg durch jedes Pixel der Projektionsfläche hindurch. Irgendwann trifft dieser Strahl auf das erste Objekt oder er verschwindet im Hintergrund (***Backward Ray Tracing***).

Abb. 5.9 zeigt dieses Prinzip am Beispiel dreier vom COP ausgehender Sehstrahlen. Beachte, dass hier noch keine gegenseitige Beeinflussung der Objekte statt findet. Andererseits wird das Problem der gegenseitigen Verdeckung von Objekten bereits auf sehr natürliche Art gelöst. Der vom Auge ausgehende Sehstrahl schneidet zwar möglicherweise mehrere Objekte in der Szene, maßgeblich für die Sichtbarkeit ist jedoch (bei nichttransparenten Objekten) ausschließlich der erste dieser Schnittpunkte. Mathematisch kann dieser sehr einfach ermittelt werden.

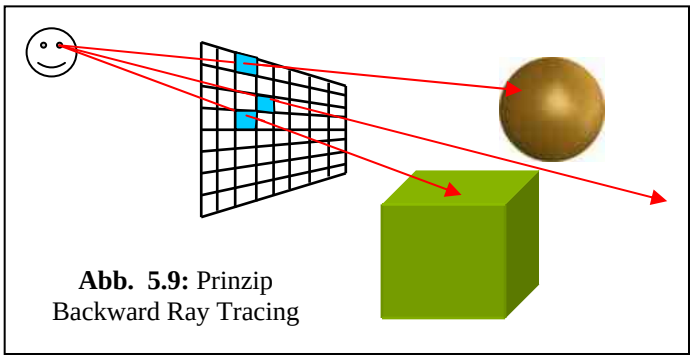


Abb. 5.9: Prinzip Backward Ray Tracing

Die Helligkeit eines Oberflächenpunktes (Schnittpunkt des Sehstrahls mit Objekt) hängt nicht nur von direkten Lichtquellen ab, die diesen Punkt beleuchten, sondern auch von anderen Objekten, die sich möglicherweise in diesem Punkt spiegeln oder, bei Transparenz des Objekts, durchscheinen. Reflektierte bzw. gebrochene Strahlen werden im Gegensatz zu den vom COP ausgehenden ***Primärstrahlen*** als ***Sekundärstrahlen*** bezeichnet. Die Verfolgung von Sekundärstrahlen führt zum sog. ***rekursiven***

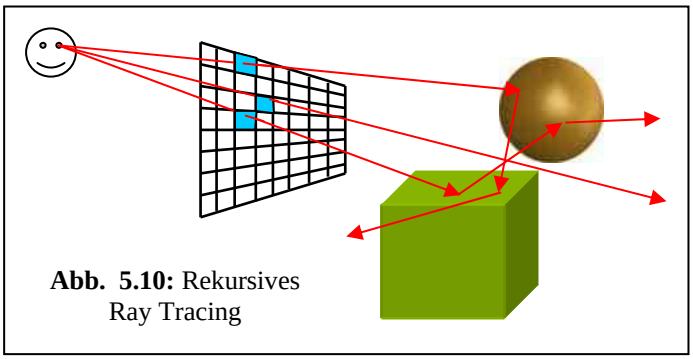


Abb. 5.10: Rekursives Ray Tracing

Ray Tracing. Abb. 5.10 zeigt den prinzipiellen Unterschied zum so genannten ***Ray Casting***, dem nicht rekursiv arbeitenden Ray Tracing (Abb. 5.9).

Gegenüber den (lokalen) Shading Verfahren lassen sich folgende Vorteile des Verfahrens ausmachen:

- Lichteffekte können fast alle realistisch simuliert werden.
- Das Verdeckungsproblem wird elegant (nebenbei) gelöst. Dies gilt auch in Fällen, wo dies bei den Schattierungs-Algorithmen nur mit Schwierigkeiten möglich ist.

Folgende Nachteile des Verfahrens bestehen gegenüber den (lokalen) Shading Verfahren:
Es ergeben sich (vor allem für das rekursive Ray Tracing) hohe Rechenzeiten, die abhängig sind von

- der Bildauflösung (es handelt sich um ein bildraumorientiertes Verfahren – vgl. Abb. 5.9!)
- der Komplexität der Szene,
- der gewünschten Bildqualität

Der durchschnittliche Aufwand für eine einfache 3D-Szene verteilt sich etwa wie folgt:

- Verwaltung: 13 %
- Schnittpunktberechnung: 75 % (bei komplexen Szenen bis 95 %)
- Helligkeitsberechnungen: 12 %

Eine Optimierung setzt daher sinnvoll bei der Schnittpunktberechnung an. Das bedeutet u. a., die Anzahl der erforderlichen Schnittpunktberechnungen zu minimieren und Schnittpunktberechnungen mit komplizierten Oberflächen durch solche mit einfachen Oberflächen zu ersetzen. Hierzu dient z.B. die Definition geeigneter Begrenzungskörper (***bounding boxes***) und entsprechender Hierarchien.

Radiosity

Das ***Radiosity*** Verfahren zur Schattierung von Objekten ist noch jünger als das Ray Tracing. Im ursprünglichen Ansatz konnten nur diffus reflektierende Oberflächen bearbeitet werden. Heute sind hybride Verfahren entwickelt worden, die Ray Tracing und Radiosity verbinden.

Prinzip: Als Annahme und Einschränkung gilt, dass alle Oberflächen ideale diffuse Reflektoren sind. Das bedeutet, dass die reflektierte Lichtintensität in allen Richtungen gleich ist. Das physikalische Verhalten des Lichtes wird mit Hilfe der Gesetze der Thermodynamik beschrieben (Energietransport und Energieerhaltung). Damit wird die Interaktion von Licht zwischen diffusen Oberflächen berücksichtigt.

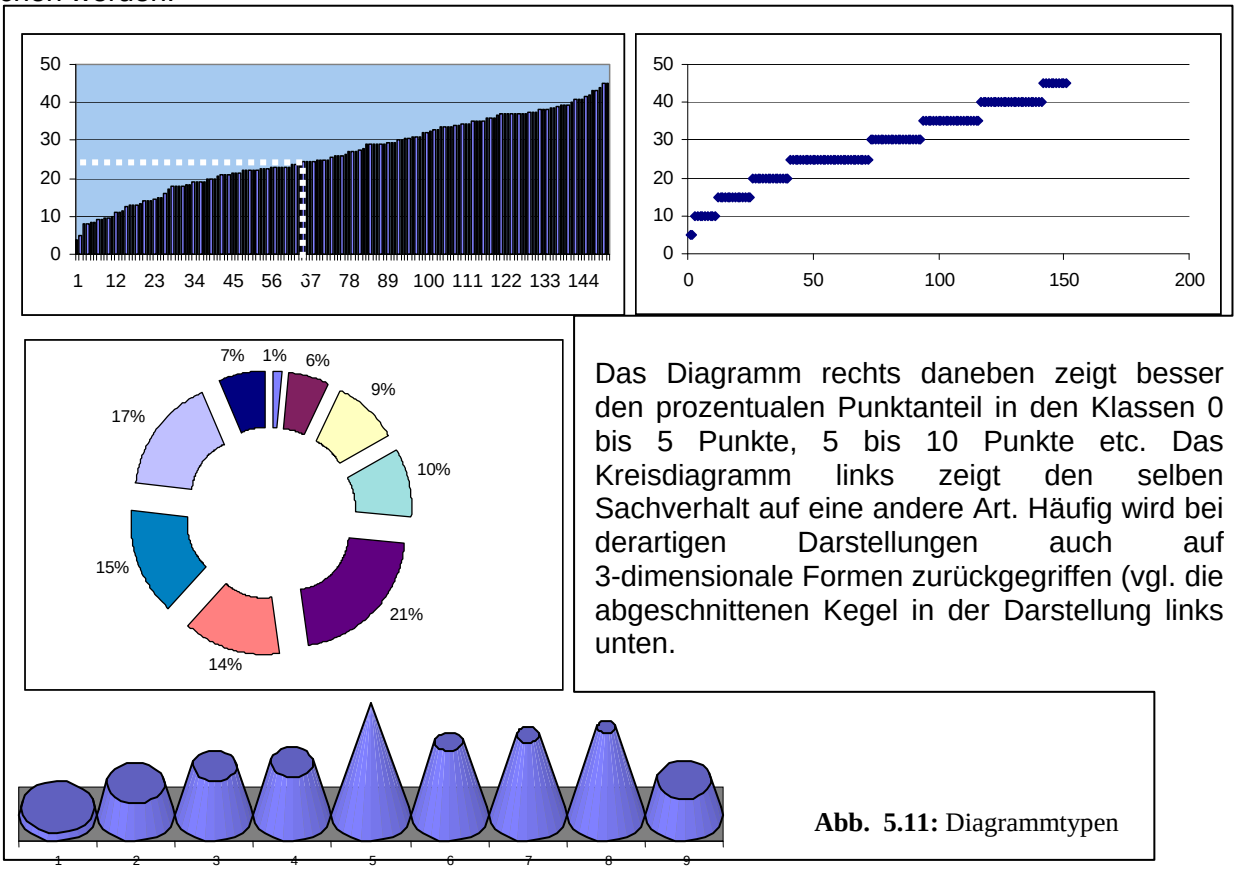
Soll die auf eine Oberfläche auftreffende Lichtenergie berechnet werden, so muss hierzu die Strahlung aus allen Richtungen des Raumes berücksichtigt werden, die die Oberfläche erreicht. Jede Fläche wird in Teilflächen zerlegt, die sich bzgl. ihrer Eigenschaften einheitlich verhalten:

- die Teilfläche ist ein idealer diffuser Reflektor, oder
- die Teilfläche ist eine ideale diffuse Lichtquelle, oder
- die Teilfläche ist eine Kombination aus beiden.

Bei Verwendung einer direkten Lichtquelle werden die Flächen bestimmt, die durch diese beleuchtet werden. Das durch die Flächen reflektierte Licht wird als diffuse Lichtquelle behandelt (sekundäre Lichtquelle).

5.2. Visualisierung nichtgeometrischer Modelle

Dieser Abschnitt zur ***Visualisierung*** nichtgeometrischer Modelle soll mit einem einfachen Beispiel beginnen. Abbildung 5.11 zeigt die Statistik über die erzielten Punkte im GDV-Kurs für Bachelor-Studierende im WS 2010/2011. An der Klausur haben 152 Studierende teilgenommen. Es konnten 45 Punkte erworben werden. Die zu Grunde liegenden Daten beinhalten keinerlei geometrische Informationen. Gleichzeitig zeigt das Beispiel den Vorteil einer graphischen Visualisierung. Auf einen Blick kann dem Diagramm links oben entnommen werden, dass etwa 66 der 152 Studierenden ihr Ziel, einen Schein zu erhalten vermutlich nicht erreichen werden.



Das Diagramm rechts daneben zeigt besser den prozentualen Punktanteil in den Klassen 0 bis 5 Punkte, 5 bis 10 Punkte etc. Das Kreisdiagramm links zeigt den selben Sachverhalt auf eine andere Art. Häufig wird bei derartigen Darstellungen auch auf 3-dimensionale Formen zurückgegriffen (vgl. die abgeschnittenen Kegel in der Darstellung links unten.

Abb. 5.11: Diagrammtypen

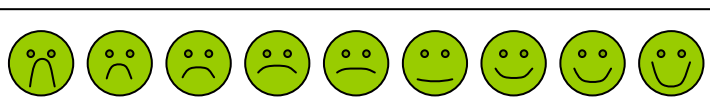


Abb. 5.12: Gesichtsikone

Abbildung 5.12 zeigt den selben Sachverhalt auf andere Art. Man spricht hier von einer ikonenbasierten Visualisierung von Daten. Hierunter versteht man die Abbildung von Datenwerten auf graphische Primitive. Letztere bezeichnet man als "***Ikonen***" oder "***Glyphen***". Wie aus der Darstellung hervorgeht, werden dabei die zu visualisierenden Werte auf einzelne Elememte des Primitivs abgebildet. Im obigen Fall sind es die "Wertigkeit" der einzelnen Klassen auf die Form des Mundes).

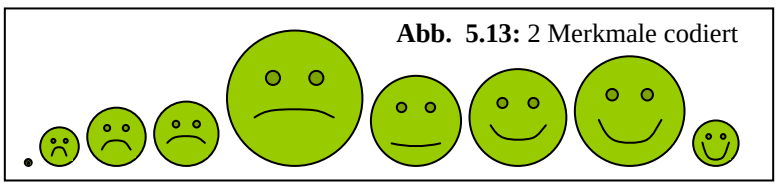


Abb. 5.13: 2 Merkmale codiert

Ikonenbasierte Techniken eignen sich auch zur Visualisierung mehrerer Parameterwerte auf einmal. Dies ist am Beispiel der Abbildung 5.13 zu ersehen. Hier wurde die Anzahl der Elemete pro Klasse in die Größe der Ikone codiert. Offensichtlich eignet sich eine derartige Darstellung recht gut, um Zusammenhänge zu verdeutlichen. Vergleiche hierzu die Aussagekraft der Darstellungen in Abbildung 5.11 und 5.13.

Multiparameterdaten (also mehr als ein Parameter) werden auch als ***mehrdimensionale Daten*** oder auch ***multivariate Daten*** bezeichnet. Die geeignete, also anschauliche Visualisierung solcher Daten ist ein eigener Wissenschaftszweig. Ohne auf die einzelnen Techniken einzugehen, folgen an dieser Stelle noch einige Beispiele:

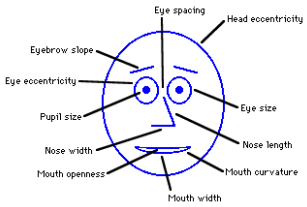


Abb. 5.14: Chernoff - Ikone mit 11 Merkmalen

Abb. 5.14 zeigt eine sog. ***Chernoff-Ikone***. Dies ist ein häufig verwendeter Glyph in dem bis zu 12 Merkmale eines menschlichen Gesichts verwendet und codiert werden.

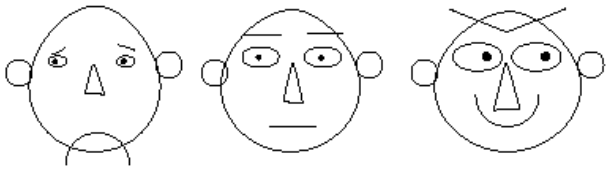


Abb. 5.15: Chernoff Gesichter mit 4 Merkmalen

Abb. 5.15 zeigt dazu ein Beispiel, bei dem 4 verschiedene Merkmale codiert wurden. Die Abbildungen 5.16 bis 5.19 zeigen abschließend noch 4 Beispiele von Visualisierungstechniken aus dem Bereich "Cybergeographie"

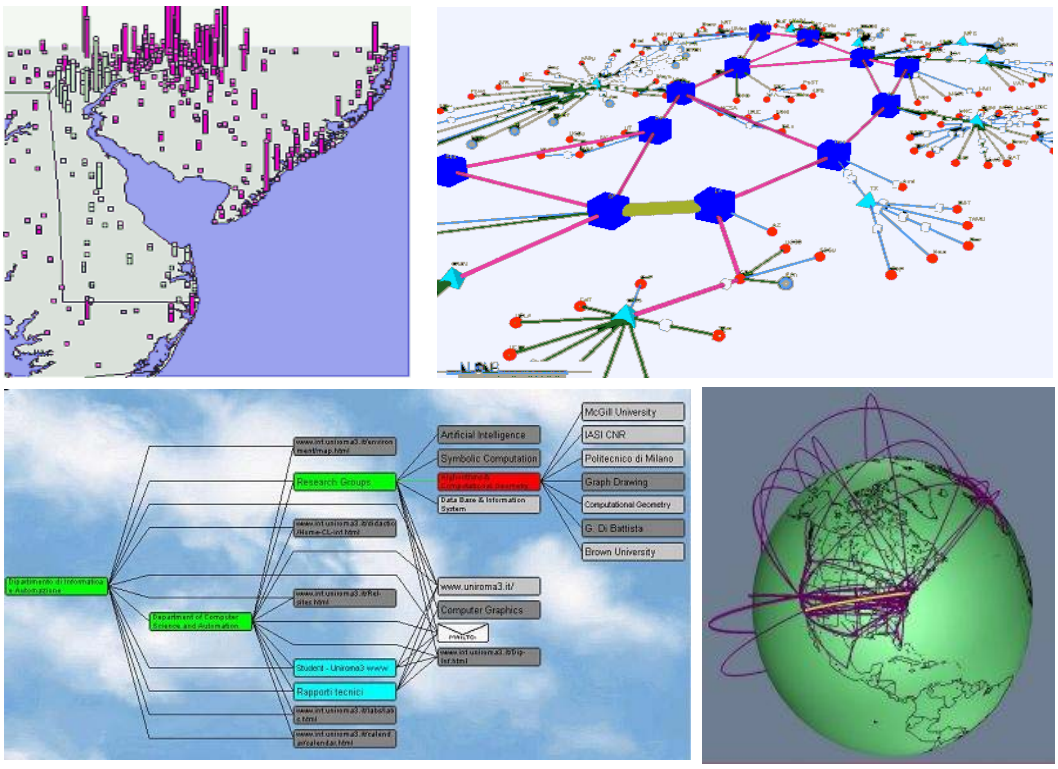


Abb. 5.16-20 Beispiele: Cybergeographie

6 Grundlegende mathematische Sachverhalte

In der Computer Graphik ist es üblich, zweidimensionale Objekte durch ihren Konturpolygonzug, d.h. den Streckenzug durch die Eckpunkte des Objektes, zu beschreiben. Bei dreidimensionalen Objekten wird die Objektoberfläche häufig in ebene Facetten (z.B. Dreiecke oder Vierecke) zerlegt, wobei die Facetten wiederum über die Eckpunkte ihres Konturpolygonzuges beschrieben sind. Grundlegende geometrische Objekte sind also Punkte und die durch diese Punkte definierten Verbindungsstrecken.

In den folgenden Teilkapiteln werden diese und weitere geometrische Grundobjekte definiert. Weiterhin werden einfache geometrische Transformationen eingeführt, mit deren Hilfe Objekte geometrisch transformiert (z.B. verschoben, gedreht und vergrößert bzw. verkleinert) werden können, indem man die Eckpunkte der Objekte transformiert.

6.1 Einfache geometrische Grundobjekte

Alle weiteren Betrachtungen dieses Teilkapitels beschränken sich auf die zweidimensionale Zeichenebene. In dieser werden Punkte als nulldimensionale Objekte, und Strecken, Strahlen und Geraden als eindimensionale Objekte betrachtet.

6.1.1 Punkt

Die ***Punkte*** der Zeichenebene werden durch ihre x- und ihre y-Koordinaten festgelegt. Punkte werden im Weiteren als Spaltenvektoren geschrieben:

(6-1)

$$\vec{P} = \begin{bmatrix} x_P \\ y_P \end{bmatrix} = [x_P, y_P]^t \quad \text{(mit } ^t \text{ für transponiert)}$$

6.1.2 Gerade, Strahl, Strecke

Strecken, Strahlen und Geraden werden im Folgenden in parametrischer Form eingeführt. Dies hat unter anderem den Vorteil, dass man mit Hilfe des Parameters jeden Punkt der Objekte gezielt ansteuern kann.

6.1.2.1 Paramerterform (vektorielle Schreibweise)

Die (gerichtete) ***Strecke*** zwischen zwei Punkten $\vec{P}_0$ und $\vec{P}_1$ (siehe Abbildung 6.1) ist die Menge aller Punkte $\vec{X}$, für die gilt:

(6-2)

$\vec{X}(u) = \vec{P}_0 + u * (\vec{P}_1 - \vec{P}_0) = \vec{P}_0 + u * \vec{R}$ mit $u \in [0, 1]$. ($\vec{R}$ ist der Richtungsvektor von $\vec{P}_0$ nach $\vec{P}_1$).

Diese Form wird als ***vektorielle Schreibweise der Parameterform*** bezeichnet.

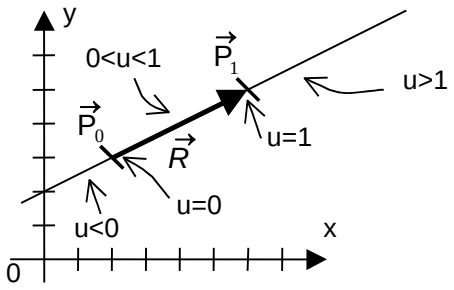


Abb. 6.1: Beispiel für die parametrische Form einer Strecke bzw. eines Strahls oder einer Geraden.

Für u=0 bzw. u=1 erhält man die Punkte $\vec{P}_0$ und $\vec{P}_1$, und für 0<u<1 erhält man alle Zwischenpunkte auf der Strecke zwischen $\vec{P}_0$ und $\vec{P}_1$.

Der ***Strahl***, der von $\vec{P}_0$ ausgeht, und über $\vec{P}_1$ hinaus nach Unendlich verläuft (siehe Abbildung 6.1), wird durch die Menge aller Punkte $\vec{X}$ gebildet, für die gilt:

(6-3)

$$\vec{X}(u) = \vec{P}_0 + u * (\vec{P}_1 - \vec{P}_0) = \vec{P}_0 + u * \vec{R} \qquad \text{mit } u \geq 0.$$

Die (gerichtete) ***Gerade*** durch die Punkte $\vec{P}_0$ und $\vec{P}_1$ besteht aus allen Punkten $\vec{X}$ (siehe Abbildung 6.1), für die gilt:

(6-4)

$$\vec{X}(u) = \vec{P}_0 + u * (\vec{P}_1 - \vec{P}_0) = \vec{P}_0 + u * \vec{R} \qquad \text{mit } u \in \mathbb{R} \text{ (reelle Zahl).}$$

6.1.2.2 Paramerterform (Matrix-Schreibweise)

Durch Umformung der vektoriellen Form erhält man die ***Matrix-Schreibweise der Parameterform***:

$$\vec{X}(u) = \vec{P}_0 + u * (\vec{P}_1 - \vec{P}_0) = \; = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + u * \begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \end{bmatrix} = \begin{bmatrix} x_0 - u * x_0 + u * x_1 \\ y_0 - u * y_0 + u * y_1 \end{bmatrix} = \begin{bmatrix} x_0 * (1-u) + u * x_1 \\ y_0 * (1-u) + u * y_1 \end{bmatrix}$$

und damit : (6-5)

$$\vec{X}(u) = \begin{bmatrix} x(u) \\ y(u) \end{bmatrix} = \begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix} * \begin{bmatrix} 1-u \\ u \end{bmatrix}$$

Mit diesem Produkt aus einer Matrix und einem Spaltenvektor erhält man entsprechend zu oben

- mit $u \in [0,1]$ eine Strecke,
- mit $u \geq 0$ einen Strahl und
- mit $u \in \mathbb{R}$ (reelle Zahl) eine Gerade.

Schlußbemerkungen: Die Verwendung der Parameterform erlaubt auch die besonders einfache und effiziente Berechnung von Schnittpunkten. Dies ist z.B. im CAD-Bereich bei der Konstruktion von Teilen und in der Computer Graphik beispielsweise beim Raytracing (Strahlverfolgung) zur Erzeugung photorealistischer Darstellungen eine ständig wiederkehrende Aufgabe.

Beispiel: `p[0], p[1], p[2]` und `p[3]` seien vier im Quadrat angeordnete Punkte. Die Seiten des Quadrates sind Strecken, die in vektorieller Parameterform gegeben sind durch: `x = p[0] + u*(p[1]-p[0])` bis `x = p[3] + u*(p[0]-p[3])` mit $u \in [0,1]$. Für einen festen Wert $u \in [0,1]$ erhält man auf jeder Seite des Quadrates einen Zwischenpunkt `zp[i]`, wobei alle Quadratseiten im selben Verhältnis unterteilt werden. Verbindet man diese Zwischenpunkte miteinander, so erhält man wieder ein Quadrat, auf dessen Seiten wiederum Zwischenpunkte ermittelt und verbunden werden, usw.. Das folgende Pseudocode-Programmstück liefert für u = 0.08 das in Abbildung 6.2 gezeigte Bild.

```
// Eingaben
Parameter u und Punkte p[0] bis p[3] eingeben;
// 20 Quadrate zeichnen
for (int j=0; j<20; j++) {
    p[4]= p[0];
    // Ein Quadrat zeichnen
    for (int i=0; i<4; i++) {
        Zeichne Linie von p[i] nach p[i+1]);
        //  neuen Zwischenpunkt berechnen
        zp[i] = p[i] + u*(p[i+1]-p[i]);
    }
    // Zwischenpunkte werden neue Eckpunkte
    for (i=0; i<4; i++)
        p[i] = zp[i];
}
```

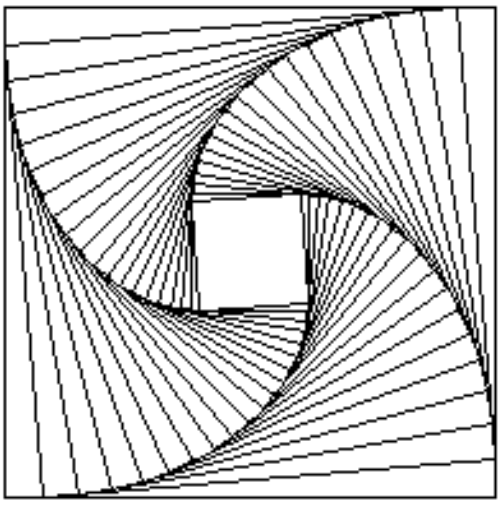


Abb. 6.2: Konstruktion von 20 Quadraten unter Verwendung der Parameterform.

6.2 2D-Koordinatentransformationen

Mit Hilfe von geometrischen Koordinatentransformationen können Objekte in der Zeichenebene positioniert, orientiert und skaliert werden. Auch Bewegungen, wie sie z.B. im Rahmen einer Animation auftreten, lassen sich mit geometrischen Transformationen durchführen. Weiterhin kann man bei der Modellierung komplexe Objekte aus mehreren geeignet positionierten, orientierten und skalierten Einzelteilen zusammensetzen.

6.2.1 2D-Elementar-Transformationen (konventionell)

Als Beispiele geometrischer ***2D-Elementar-Transformationen*** werden im Weiteren die Translation, die Skalierung und die Rotation vorgestellt.

6.2.1.1 Translation (Verschiebung)

Die ***Translation (Verschiebung)*** T_T eines Punktes $\vec{P} = [x_P, y_P]^t$ um dx in x- und dy in y-Richtung liefert den Punkt $\vec{P}' = [x_P', y_P']^t$ mit:

(6-6)

$$\begin{aligned} x_P' &= dx + x_P \\ y_P' &= dy + y_P \end{aligned} \quad \text{d.h.} \quad \begin{bmatrix} x_P' \\ y_P' \end{bmatrix} = \begin{bmatrix} dx \\ dy \end{bmatrix} + \begin{bmatrix} x_P \\ y_P \end{bmatrix} \quad \text{bzw.} \quad \vec{P}' = T_T + \vec{P}.$$

Beispiel:

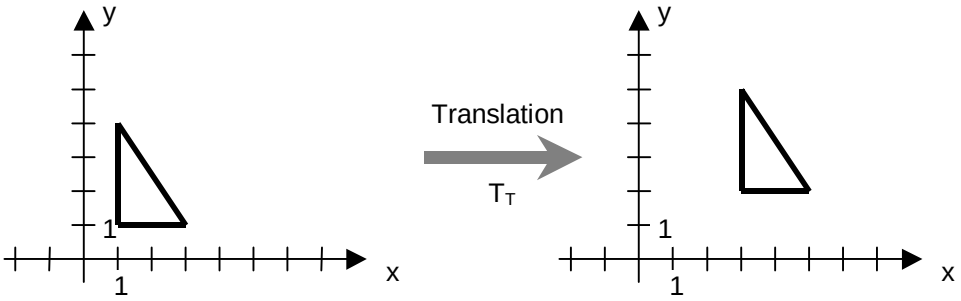


Abb. 6.3: Beispiel für eine Translation um dx = 2 und dy = 1.

6.2.1.2 Skalierung (Größenänderung)

Die ***Skalierung (Größenänderung)*** T_S eines Punktes $\vec{P} = [x_P, y_P]^t$ mit s_x in x- und s_y in y-Richtung bezüglich des Ursprungs liefert den Punkt $\vec{P}' = [x_P', y_P']^t$ mit:

(6-7)

$$\begin{aligned} x_P' &= s_x * x_P \\ y_P' &= s_y * y_P \end{aligned} \quad \text{d.h.} \quad \begin{bmatrix} x_P' \\ y_P' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \end{bmatrix} \quad \text{bzw.} \quad \vec{P}' = T_S * \vec{P}.$$

(Natürlich ändert sich bei einer Skalierung nicht die Größe, sondern nur die Lage eines Punktes.)

Beispiel:

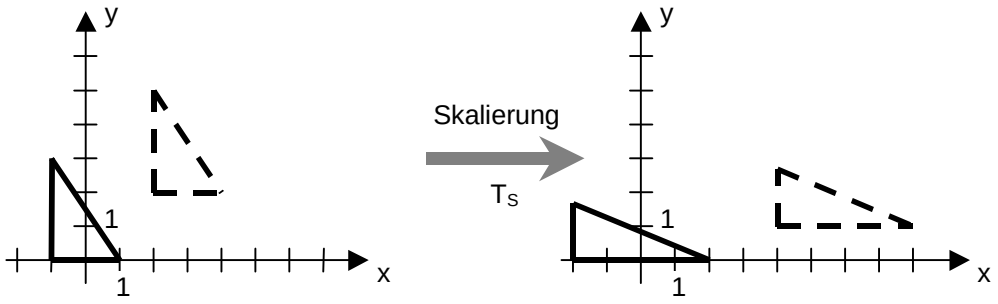


Abb. 6.4: Beispiele für eine Skalierung mit $s_x = 2$ und $s_y = ½$ bezüglich des Ursprungs.

Das Beispiel zeigt, dass keineswegs gelten muss: $s_x = s_y$.

6.2.1.3 Rotation (Drehung)

Die ***Rotation (Drehung)*** T_R eines Punktes $\vec{P} = [x_P, y_P]^t$ um den Winkel φ bezüglich des Ursprungs liefert den Punkt $\vec{P}' = [x_P', y_P']^t$ mit:

(6-8)

$$x_P' = \cos(\varphi) * x_P - \sin(\varphi) * y_P \qquad y_P' = \sin(\varphi) * x_P + \cos(\varphi) * y_P$$

$$\text{d.h. } \begin{bmatrix} x_P' \\ y_P' \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \end{bmatrix} \quad \text{bzw. } \vec{P}' = T_R * \vec{P}.$$

Beispiel:

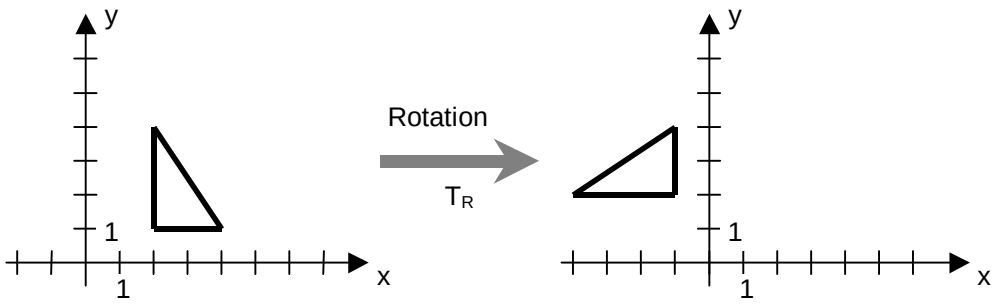


Abb. 6.5: Beispiel für eine Rotation um den Winkel $\varphi = 90°$ bezüglich des Ursprungs.

6.2.1.4 Kombination von 2D-Elementar-Transformationen

Im Allgemeinen besteht eine gewünschte Transformation nicht nur aus einer einzelnen Elementar-Transformation, sondern es müssen verschiedene Elementar-Transformationen miteinander kombiniert werden. Da die Skalierung (6-7) und die Rotation (6-8) als Matrixmultiplikationen, die Translation (6-6) aber als Vektoraddition eingeführt wurde, lassen sich Kombinationen von Elementar-Transformationen in der Regel nicht in einer einzigen Transformationsmatrix zusammenfassen. Das nachfolgende Kapitel zeigt, wie sich dieser Nachteil durch die Verwendung homogener Koordinaten vermeiden lässt.

6.2.2 2D-Elementar-Transformationen mit homogenen Koordinaten

Homogene Koordinaten stellen eine Erweiterung der „normalen" Koordinaten um eine weitere, sogenannte ***homogene Komponente*** dar. Ein Punkt $\vec{P} = [x_P, y_P]^t$ wird in homogenen Koordinaten geschrieben als $\vec{P} = [x_P, y_P, 1]^t$. Die homogene Komponente hat im Weiteren bei Punkten immer den Wert 1.

Unter Verwendung von homogenen Koordinaten können alle drei eingeführten Transformationen als Matrix-Multiplikationen ausgedrückt werden.

6.2.2.1 Translation (Verschiebung)

Die ***Translation (Verschiebung)*** T_T eines Punktes $\vec{P} = [x_P, y_P, 1]^t$ um dx in x- und dy in y-Richtung liefert den Punkt $\vec{P}' = [x_P', y_P', 1]^t$ mit:

(6-9)

$$\begin{bmatrix} x_P' \\ y_P' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \\ 1 \end{bmatrix} \quad \text{bzw. } \vec{P}' = T_T * \vec{P}.$$

Die inverse Transformation zu einer Translation um dx und dy ist die Translation um –dx und –dy.

Bei einer Translation mit $dx \neq 0$ oder $dy \neq 0$ werden alle Punkte der Zeichenebene verschoben, so dass eine Translation keinen ***Fixpunkt*** besitzt.

6.2.2.2 Skalierung (Größenänderung)

Die ***Skalierung (Größenänderung)*** T_S eines Punktes $\vec{P} = [x_P, y_P, 1]^t$ mit s_x in x- und s_y in y-Richtung bezüglich des Ursprungs liefert den Punkt $\vec{P}' = [x_P', y_P', 1]^t$ mit:

(6-10)

$$\begin{bmatrix} x_P' \\ y_P' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \\ 1 \end{bmatrix} \quad \text{bzw. } \vec{P}' = T_S * \vec{P}\,.$$

Die inverse Transformation zu einer Skalierung mit s_x und s_y ist die Skalierung $1/s_x$ und $1/s_y$.

Da der Ursprung sich bei einer Skalierung (bezüglich des Ursprungs) nicht ändert, ist der Ursprung der Fixpunkt dieser Skalierung.

Spezielle Skalierungen

1. Wenn genau einer der Skalierungsfaktoren s_x oder s_y gleich Null ist, so ergibt sich eine ***Projektion***. Für s_x=0 erhält man eine Projektion auf die y-Achse, und für s_y=0 erhält man eine Projektion auf die x-Achse. Wenn der von Null verschiedene Faktor ungleich Eins ist, so ergibt sich zusätzlich eine Größenänderung.
2. Wenn einer oder beide Skalierungsfaktoren negativ sind, so erhält man eine ***Spiegelung***:
   - mit s_x=**-**1 und s_y=1 erhält man eine Geraden-Spiegelung an der y-Achse.
   - mit s_x=1 und s_y=**-**1 erhält man eine Geraden-Spiegelung an der x-Achse.
   - mit s_x=**-**1 und s_y=**-**1 erhält man eine Punkt-Spiegelung am Ursprung.

   (Wenn einer oder beide Faktoren betragsmäßig ungleich Eins sind, so ergibt sich zusätzlich eine Größenänderung).

6.2.2.3 Rotation (Drehung)

Die ***Rotation (Drehung)*** T_R eines Punktes $\vec{P} = [x_P, y_P, 1]^t$ um den Winkel φ bezüglich des Ursprungs liefert den Punkt $\vec{P}' = [x_P', y_P', 1]^t$ mit:

(6-11)

$$\begin{bmatrix} x_P' \\ y_P' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \\ 1 \end{bmatrix} \quad \text{bzw. } \vec{P}' = T_R * \vec{P}\,.$$

Die inverse Transformation zu einer Rotation um den Winkel φ ist die Rotation um den Winkel $-\varphi$

Da der Ursprung sich bei einer Rotation (bezüglich des Ursprungs) nicht ändert, ist der Ursprung der Fixpunkt dieser Rotation.

6.2.3 Zusammengesetzte 2D-Transformationen

Mit Hilfe von homogenen Koordinaten lassen sich die Elementar-Transformationen (Translation, Skalierung und Rotation) eines Punktes als Multiplikation einer 3*3-Matrix mit dem Spaltenvektor des zu transformierenden Punktes darstellen. Dies hat zur Folge, dass komplexe Transformationen, die sich als ***Hintereinanderausführung (Verkettung)*** mehrerer Elementar-Transformationen schreiben lassen, zu einer einzigen ***Gesamttransformation*** T_G zusammengefasst werden können.

Eine Gesamttransformation T_G, die sich aus

- einer Skalierung T_S mit s_x und s_y bezüglich eines Punktes $\vec{P}_0 = [x_0,y_0,1]^t$,
- einer Rotation T_R um den Winkel φ bezüglich desselben Punktes und
- einer Translation T_T um dx und dy

zusammensetzt, wird vereinbarungsgemäß wie folgt zerlegt:

(6-12)

$$T_G = T_T'' * T_R * T_S * T_T'$$

Dabei ist T_T' die Hilfstranslation um $-x_0$ und $-y_0$, die den Punkt $\vec{P}_0 = [x_0,y_0,1]^t$ in den Ursprung verschiebt. Die Skalierung T_S und die Rotation T_R können nun wie oben beschrieben bezüglich des

Ursprungs durchgeführt werden. Die abschließende Hilfstranslation $T_{T''}$ macht die erste Hilfstranslation $T_{T'}$ rückgängig und führt zusätzlich die gewünschte Translation um dx und dy durch.

Ausmultiplizieren von (6-12) liefert:

(6-13)

$$T_G = \begin{bmatrix} 1 & 0 & x_0 + dx \\ 0 & 1 & y_0 + dy \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} s_x * \cos(\varphi) & -s_y * \sin(\varphi) & -x_0 * s_x * \cos(\varphi) + y_0 * s_y * \sin(\varphi) + x_0 + dx \\ s_x * \sin(\varphi) & s_y * \cos(\varphi) & -x_0 * s_x * \sin(\varphi) - y_0 * s_y * \cos(\varphi) + y_0 + dy \\ 0 & 0 & 1 \end{bmatrix}$$

Die Transformation eines allgemeinen Punktes $\vec{P} = [x_P, y_P, 1]^t$ ergibt sich also zu:

(6-14)

$$\vec{P} = T_{T''} * T_R * T_S * T_{T'} * \vec{P} = T_G * \vec{P}$$

Beispiel: Eine zusammengesetzte Transformation soll bezüglich eines Punktes $\vec{P}_0 = [2,1,1]^t$ eine Skalierung mit s_x=-1 und s_y=2, sowie eine Rotation um 270° und außerdem eine Translation um dx=3 und dy=0 durchführen. Die Gesamttransformation T_G ergibt sich zu: $T_G = \begin{bmatrix} 0 & 2 & 3 \\ 1 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}$.

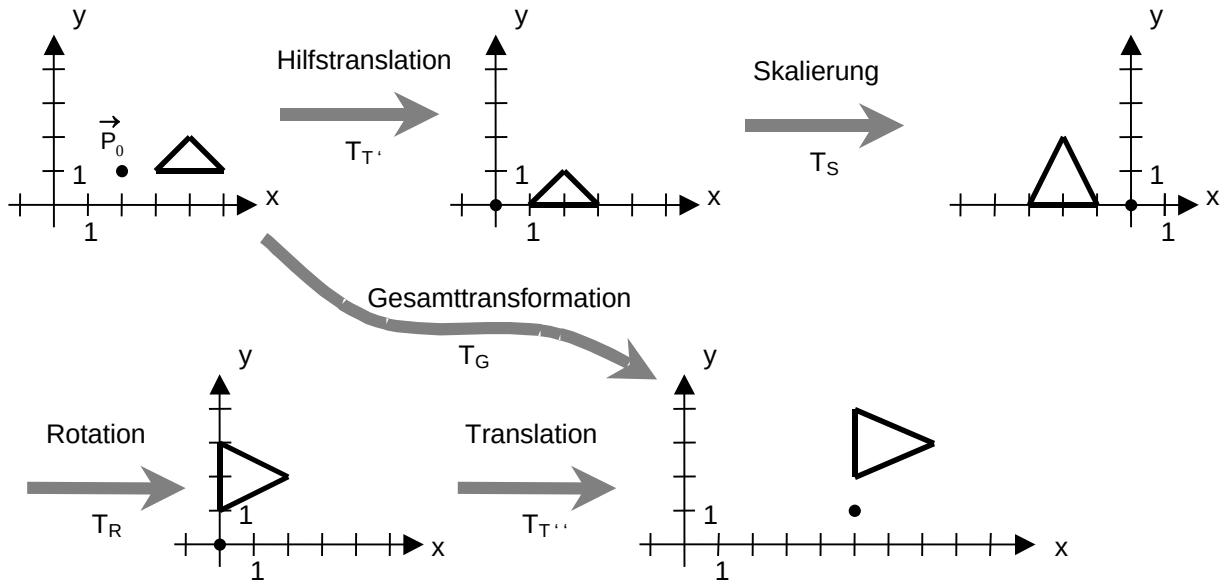


Abb. 6.6: Beispiel für eine zusammengesetzte Transformation.

Mit Hilfe der ***Fixpunkt-Bedingung***

(6-15)

$$\vec{P}_F = T * \vec{P}_F$$

kann man feststellen, ob eine Transformation T einen Fixpunkt $\vec{P}_F$ hat, also einen Punkt, der sich bei der Transformation nicht ändert.

6.3 3D-Koordinatentransformationen mit homogenen Koordinaten

Ein 3D-Punkt wird in ***homogenen Koordinaten*** geschrieben als $\vec{P} = [x_P, y_P, z_P, 1]^t$. Unter Verwendung von homogenen Koordinaten können unter anderem die ***3D-Elementar-Transformationen*** Translation, Skalierung und Rotation als Matrix-Multiplikationen ausgedrückt werden. (Alternative Möglichkeiten, wie z.B. die perspektivische Projektion werden hier nicht betrachtet).

6.3.1 Translation (Verschiebung)

Die ***Translation (Verschiebung)*** T_T eines Punktes $\vec{P} = [x_P, y_P, z_P, 1]^t$ um dx in x-Richtung, dy in y-Richtung und dz in z-Richtung liefert den Punkt $\vec{P}' = [x_P', y_P', z_P',1]^t$ mit:

(6-16)

$$\begin{bmatrix} x_P' \\ y_P' \\ z_P' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix} \quad \text{bzw. } \vec{P}' = T_T * \vec{P}.$$

6.3.2 Skalierung (Größenänderung)

Die ***Skalierung (Größenänderung)*** T_S eines Punktes $\vec{P} = [x_P, y_P, z_P, 1]^t$ mit s_x in x-Richtung, s_y in y-Richtung und s_z in z-Richtung bzgl. des Ursprungs liefert den Punkt $\vec{P}' = [x_P', y_P', z_P',1]^t$ mit:

(6-17)

$$\begin{bmatrix} x_P' \\ y_P' \\ z_P' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix} \quad \text{bzw. } \vec{P}' = T_S * \vec{P}.$$

6.3.3 Rotationen (Drehungen)

Die ***Rotation (Drehung)*** T_{Rz} eines Punktes $\vec{P} = [x_P, y_P, z_P, 1]^t$ um den Winkel φ um die z-Achse liefert den Punkt $\vec{P}' = [x_P', y_P', z_P',1]^t$ mit:

(6-18)

$$\begin{bmatrix} x_P' \\ y_P' \\ z_P' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix} \quad \text{bzw. } \vec{P}' = T_{Rz} * \vec{P}.$$

Die ***Rotation (Drehung)*** T_{Rx} eines Punktes $\vec{P} = [x_P, y_P, z_P, 1]^t$ um den Winkel φ um die x-Achse liefert den Punkt $\vec{P}' = [x_P', y_P', z_P',1]^t$ mit:

(6-19)

$$\begin{bmatrix} x_P' \\ y_P' \\ z_P' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) & 0 \\ 0 & \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix} \quad \text{bzw. } \vec{P}' = T_{Rx} * \vec{P}.$$

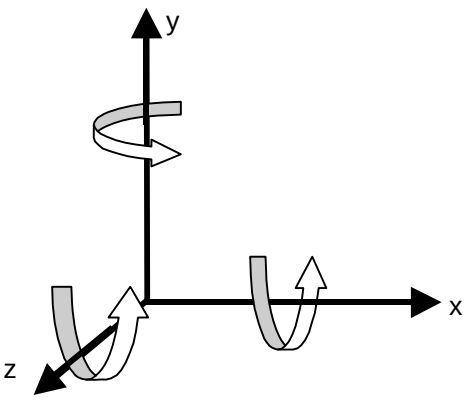


Abb. 6.7: Demonstration des mathematisch positiven Drehsinns.

Die ***Rotation (Drehung)*** T_{Ry} eines Punktes $\vec{P} = [x_P, y_P, z_P, 1]^t$ um den Winkel φ um die y-Achse liefert den Punkt $\vec{P}' = [x_P', y_P', z_P', 1]^t$ mit:

(6-20)

$$\begin{bmatrix} x_P' \\ y_P' \\ z_P' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & 0 & \sin(\varphi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix} \qquad \text{bzw. } \vec{P}' = T_{Ry} * \vec{P}\,.$$

Bemerkungen zu Rotationen:

1.) Wenn der Daumen der rechten Hand in die Richtung der betreffenden Achse zeigt, geben die angewinkelten Finger den mathematisch positiven Drehsinn an; siehe Abbildung 6.7.

2.) Die vordersten drei Spalten einer Rotationsmatrix sind die Transformierten der Einheits-Richtungs-Vektoren der x-Achse, y-Achse und z-Achse. Die Spalten geben also an, in welche Richtung die transformierten Koordinaten-Achsen verlaufen; siehe Abbildung 6.8. (Bei Richtungs-Vektoren in homogener Schreibweise hat die homogene Komponente den Wert 0).

$$T_{Ry}(90°) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

transformierte z-Achse z' zeigt in Richtung der positiven x-Achse

y-Achse bleibt unverändert

transformierte x-Achse x' zeigt in Richtung der negativen z-Achse

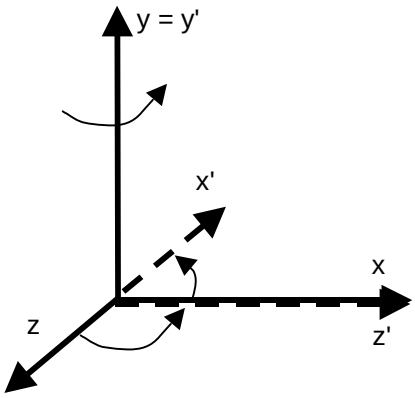


Abb. 6.8: Beispiel für eine Rotation um 90 Grad um die y-Achse; die Spalten der Rotations-Matrix (links) geben die Richtung der transformierten Achsen (rechts) an.

7 Digitale Bilder

In diesem Kapitel werden einige grundlegende Begriffe eingeführt, die im Zusammenhang mit der Erzeugung, Repräsentation, Manipulation und Ausgabe digitaler Bilder eine Rolle spielen.

7.1 Geometrische Betrachtungen

In den folgenden Teilkapiteln wird kurz auf die geometrischen Aspekte des Bildraums und der Darstellung von Objekten in digitalen Bildern eingegangen.

7.1.1 Bildpunkte (Pixel) und digitale Bilder

Im Gegensatz zu einem mathematischen Punkt, der im kontinuierlichen (analogen) Objektraum keine Ausdehnung besitzt, ist die Darstellung dieses Punktes im diskreten Bildraum nur möglich, wenn der Punkt eine bestimmte Ausdehnung zugewiesen bekommt. Darüber hinaus kann der Punkt im Bildraum nicht beliebig platziert werden, da er in das Raster des Bildraums eingepasst werden muss und so nur ganzzahlige Positionen auf der Darstellungsfläche einnehmen kann. Eine weitere Folge dieser Tatsache ist, dass auf einer endlich großen Darstellungsfläche nur endlich viele Punkte platziert werden können.

Ein endlicher Ausschnitt des gerasterten Bildraums wird als ***digitales Bild*** bezeichnet und die Rasterelemente heißen ***Bildpunkte*** oder ***Pixel*** (von engl. **pic**ture **el**ement). Ein Pixel lässt sich durch folgende Eigenschaften beschreiben (Modellvorstellung eines Pixels):

- Position
- Helligkeit (Intensität, Grauwert, Grauton) bzw. Farbe
- Ausdehnung
- Form

Im Folgenden wird stets von quadratischen Pixeln ausgegangen.

Die Position eines Pixels wird mit Hilfe von Koordinatenangaben (x- und y- Koordinate) in einem ***Koordinatensystem*** beschrieben. Aus historischen Gründen werden in der graphischen Datenverarbeitung und in der digitalen Bildverarbeitung unterschiedliche Koordinatensysteme benutzt: während die graphische Datenverarbeitung ein "normales" Koordinatensystem verwendet (siehe Abbildung 7.1(a)), sitzt der Ursprung bei der digitalen Bildverarbeitung (wie beim Fernsehbild) links oben (siehe Abbildung 7.1(b)).

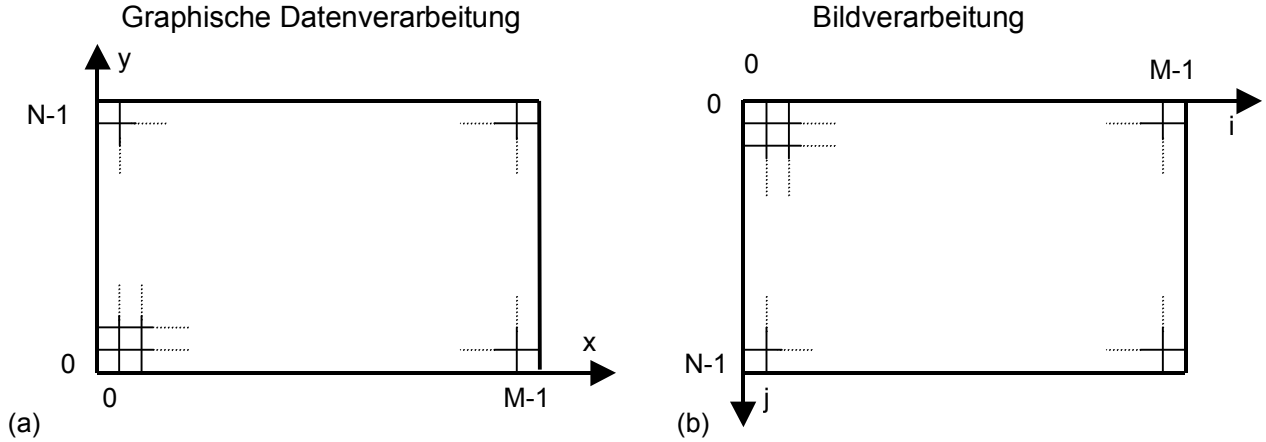


Abb. 7.1: Koordinatensysteme in der Graphischen Datenverarbeitung (a) und der digitalen Bildverarbeitung (b).

Wenn die Ausgabefläche aus M Spalten und N Zeilen besteht, so besitzt sie eine ***absolute Auflösung*** von M*N Bildpunkten. Das Seitenverhältnis M:N wird als ***Aspect Ratio (Darstellungsverhältnis)*** bezeichnet. In Tabelle 7.1 sind einige gängige Beispiele für absolute Auflösungen und die zugehörigen Darstellungsverhältnisse aufgelistet.

7.1.2 Vektor- und Rastertechnik

Beim Zeichnen und Repräsentieren graphischer Objekte werden zwei unterschiedliche Techniken eingesetzt, die im Folgenden am Beispiel einer Strecke kurz erläutert werden.

Spaltenzahl M	Zeilenzahl N	Absolute Auflösung	Aspect Ratio
640	480	307 200	4 : 3
800	600	480 000	4 : 3
1.024	768	786 432	4 : 3
1.280	1.024	1 310 720	5 : 4
1.600	1.200	1 920 000	4 : 3
1.920	1.200	2 304 000	16 : 10
2.048	1.536	3.145.728	4 : 3
3.840	2.400	9.216.000	16 : 10
7.680	4.800	36.864.000	16 : 10

Tabelle 7.1: Beispiele für absolute Auflösungen und Darstellungsverhältnisse.

In der ***Vektortechnik*** wird eine Strecke gezeichnet, indem das „Zeichengerät" (z.B. Zeichenstift oder Elektronenstrahl) am Anfangspunkt der Strecke aufgesetzt und dann bis zum Endpunkt der Strecke geführt wird; siehe Abbildung 7.2 (a). (Diese Technik wird heute fast nur noch bei Stift-Plottern eingesetzt). Zur Repräsentation der Strecke in vektorieller Form genügt es, die Koordinaten des Anfangs- und des Endpunktes anzugeben und festzulegen, dass diese Punkte miteinander verbunden werden sollen. (Zur Abspeicherung graphischer Informationen in Vektortechnik gibt es spezielle Meta- und Vektor-Bildformate; siehe Kapitel 9.3).

Beim Zeichnen in ***Rastertechnik*** wird unabhängig vom tatsächlichen Bildinhalt jedes der M*N Pixel der Ausgabefläche „gezeichnet". Dabei erhalten die Pixel, die auf der Strecke liegen, eine andere Intensität bzw. Farbe, als die des Hintergrunds; siehe Abbildung 7.2 (b). Bei schräg verlaufenden Linien entstehen dabei „Treppeneffekte", die als ***Aliasing*** bezeichnet werden. Diese Effekte sind in Abhängigkeit von der gewählten Auflösung mehr oder weniger gut sichtbar.

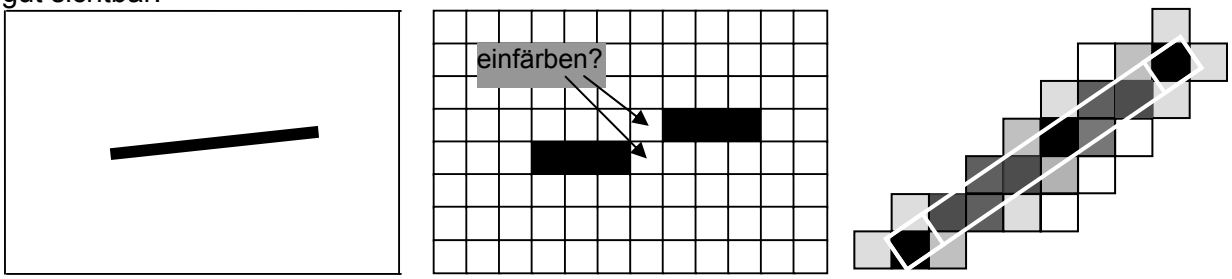


Abb. 7.2: Beispiel für das Zeichnen einer Strecke in Vektortechnik (a) bzw. und in Rastertechnik (b). In Teil (b) sind die beiden durch den Pfeil gekennzeichneten Pixel beide gleich ungünstig platziert: es entsteht in beiden Fällen Aliasing (Treppeneffekt). Eine mögliche Abhilfe für schräg verlaufende Linien zeigt (c); (***Anti-Aliasing***).

Zur Repräsentation graphischer Information in Rasterform kann man beispielsweise eine Matrix als virtuelle Entsprechung einer gerasterten Ausgabefläche definieren und alle betroffenen Matrixelemente auf einen bestimmten (Farb-)Wert setzen. Zur Abspeicherung graphischer Information in Rastertechnik gibt es spezielle Raster-Bildformate; siehe Kapitel 9.

Die Konvertierung von Daten aus der vektoriellen Form in die Rasterform bzw. umgekehrt ist ein wesentlicher Bestandteil der Computer Graphik bzw. der Digitalen Bildverarbeitung:

- in der Graphischen Datenverarbeitung wird üblicherweise in Vektortechnik modelliert, so dass die Daten zur Ausgabe auf einem Rasterbildschirm Vektor-Raster-konvertiert werden müssen; (***Vektor-Raster-Konvertierung***).
- in der Bildverarbeitung ist die Eingabe ein Rasterbild, aber die Ergebnisse werden häufig in vektorieller Form gewünscht; (***Raster-Vektor-Konvertierung***, z.B. Repräsentation der Kontur eines im Bild enthaltenen Objektes durch einen Polygonzug; siehe z.B. Kapitel 8.2).

7.2 Photometrische Betrachtungen

Während mit geometrischen Aussagen der Ort beschrieben wird, den ein Objekt einnimmt, beziehen sich photometrische Betrachtungen auf das Aussehen, d.h. die Darstellung eines Objektes mit Hilfe von Grautönen bzw. Farben.

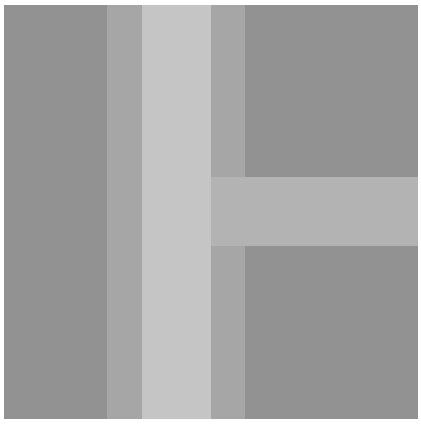
7.2.1 Schwarz-Weiß- und Grauwertbilder

Unter Schwarz-Weiß- und Grauwertbildern versteht man „unbunte“ Bilder, wie sie beispielsweise bei einfachen industriellen Bildverarbeitungs-Aufgaben und bei schwarz-weißen Printmedien (z.B. im nichtfarbigen Zeitungsdruck) verwendet werden.

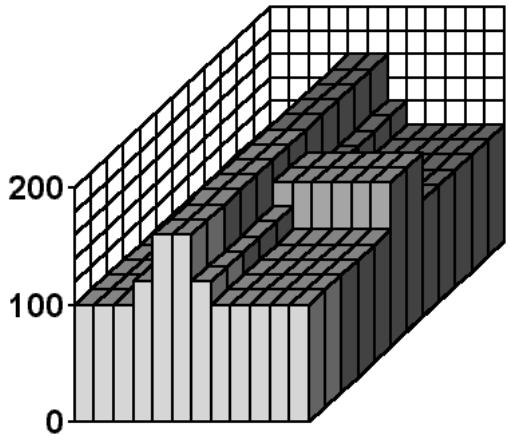
Die ***photometrische Auflösung*** (hier ***Graustufenauflösung***) gibt an, wie viele Graustufen g zwischen Schwarz und Weiß unterschieden werden. Die Graustufenauflösung ist üblicherweise eine Zweierpotenz 2^n mit n=1, 2, ... Bei n=1 spricht man von ***Schwarz-Weiß-Bildern***, während man Bilder mit n>1 als ***Halbton-***, ***Grauton-*** oder ***Grauwertbilder*** bezeichnet. Der minimale Grauwert g_{MIN}=0 steht dabei für die Farbe Schwarz und der maximale Grau-

	0	1	2	3	4	5	6	7	8	9	10	11
0	100	100	100	120	160	160	120	100	100	100	100	100
1	100	100	100	120	160	160	120	100	100	100	100	100
2	100	100	100	120	160	160	120	100	100	100	100	100
2	100	100	100	120	160	160	120	100	100	100	100	100
3	100	100	100	120	160	160	120	100	100	100	100	100
4	100	100	100	120	160	160	120	100	100	100	100	100
5	100	100	100	120	160	160	140	140	140	140	140	140
6	100	100	100	120	160	160	140	140	140	140	140	140
7	100	100	100	120	160	160	120	100	100	100	100	100
8	100	100	100	120	160	160	120	100	100	100	100	100
9	100	100	100	120	160	160	120	100	100	100	100	100
10	100	100	100	120	160	160	120	100	100	100	100	100
11	100	100	100	120	160	160	120	100	100	100	100	100

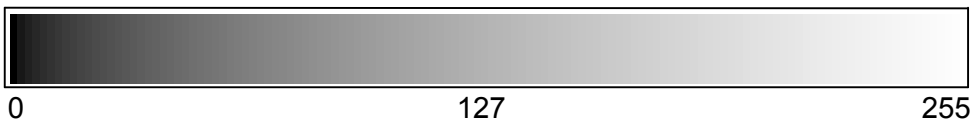
(a) Grauwertmatrix mit Grauwerten $g \in [0,255]$



(b) Grauwertbild



(c) Grauwertgebirge



(d) Graukeil

Abb. 7.5: Repräsentation von Grauwertbildern:
(a) Grauwertmatrix (b) Grauwertbild
(c) Grauwertgebirge (d) Graukeil

wert $g_{MAX} = 2^n-1$ steht für Weiß. Häufig haben Grauwertbilder eine Graustufenauflösung von $2^8 = 256$ Graustufen, da hier genau ein Byte zur Abspeicherung des Grauwertes eines Bildpunktes erforderlich ist.

Abbildung 7.5 zeigt ein Beispiel für die Repräsentation von Grauwertbildern. In Abbildung 7.5 (a) ist die ***Grauwertmatrix*** mit den Grauwerten in Form von Zahlen dargestellt. Abbildung 7.5 (b) zeigt diese Matrix als Grauwertbild, und in 7.5 (c) sind die Grauwerte in die dritte Dimension abgetragen, wodurch man ein ***Grauwertgebirge*** erhält. Um das Grauwertspektrum von Bildern visuell abschätzen zu können, wird häufig noch ein sogenannter ***Graukeil*** dargestellt; siehe Abbildung 7.5 (d).

7.2.2 Farbbilder

Die Farbwahrnehmung ist ein wesentlicher Aspekt der visuellen Informationsaufnahme und so werden Bilder heute i.d.R. farbig aufgenommen bzw. synthetisch erzeugt. Um einerseits Farbbilder im Rechner repräsentieren und auf Ausgabegeräten wie Bildschirm oder Drucker ausgeben zu können und andererseits Farben intuitiv definieren zu können, wurden unterschiedliche Farbmodelle entwickelt.

7.2.2.1 RGB- und CMY-Farbmodell

Das ***RGB-Farbmodell*** (**R**ed, **G**reen, **B**lue), bei dem jede Farbe in Ihren Rot-, Grün und Blau-Anteil zerlegt wird, wird z.B. bei Farbmonitoren verwendet. Jeder Bildpunkt eines Bildschirms setzt sich aus drei Einzelpünktchen zusammen, die rotes, grünes und blaues Licht abstrahlen können. Wenn zur Übertragung der Bildinformation (wie z.B. beim Fernsehen) ein anders Farbmodell verwendet wird, so wird vor der Bildausgabe eine Konvertierung in das RGB-Farbmodell vorgenommen.

Der RGB-Farbraum lässt sich als Einheitswürfel darstellen; siehe Abbildung 7.6 (a). Dabei bilden die Primärfarben Rot, Grün und Blau die drei positiven Hauptachsen: r-Achse, g-Achse und b-Achse. Jede Farbe wird durch einen Punkt im Einheitswürfel, d.h. ein Tripel $[r, g, r]^t_{RGB}$ mit $r, g, b \in [0, 1]$ dargestellt.

Die Farbe Schwarz liegt im Ursprung bei r=g=b=0; (wenn alle drei Farben an einer Bildschirmposition den Wert Null haben, so bleibt der Bildschirm an dieser Stelle schwarz). Die Farben Rot, Grün und Blau werden durch die Tripel $[1,0,0]^t_{RGB}$, $[0,1,0]^t_{RGB}$ und $[0,0,1]^t_{RGB}$ repräsentiert. Aus diesen drei Primärfarben lässt sich jede gewünschte Farbe F durch ***additive Farbmischung*** erzielen:

$$F = \alpha*\text{Rot} + \beta*\text{Grün} + \gamma*\text{Blau} = \alpha * \begin{bmatrix}1\\0\\0\end{bmatrix}_{RGB} + \beta * \begin{bmatrix}0\\1\\0\end{bmatrix}_{RGB} + \gamma * \begin{bmatrix}0\\0\\1\end{bmatrix}_{RGB} \quad \text{mit } \alpha,\beta,\gamma \in [0,1].$$

Wenn alle drei Primärfarben mit voller Intensität strahlen, so erhält man Weiß.

Die Raumdiagonale des Würfels, die die Punkte Schwarz und Weiß verbindet, heißt ***Unbunt-Gerade***. Diese Gerade enthält alle Grauwerte zwischen Schwarz und Weiß, für die also gilt: r = g = b.

Im Gegensatz zu selbstleuchtenden Medien, bei denen das RGB-Modell mit seiner additiven Farbmischung verwendet wird, setzt man bei reflektierenden Medien (z.B. Farbdruck) das komplementäre ***CMY-Farbmodell*** ein (**C**yan, **M**agenta und Gelb bzw. **Y**ellow); siehe Abbildung 7.6 (b). Der Einheitswürfel wird hier genau entgegengesetzt zum RGB-Modell definiert und jede Farbe wird durch ein Tripel $[c, m, y]^t_{CMY}$ mit $c, m, y \in [0, 1]$ repräsentiert. Der Ursprung wird durch die Farbe Weiß gebildet: wenn man auf ein weißes Blatt Papier keine Farbe aufträgt, so reflektiert das Papier nach wie vor das volle Spektrum des Lichtes und erscheint Weiß.

Durch Auftragen von Farbe werden bestimmte Frequenzbereiche des Lichtes herausgefiltert, weshalb man hier von einer ***subtraktiven Farbmischung*** spricht. So filtert ein Farbauftrag mit Cyan die Farbe Rot aus weißem Licht heraus, Magenta filtert Grün und Gelb reflektiert kein Blau mehr.

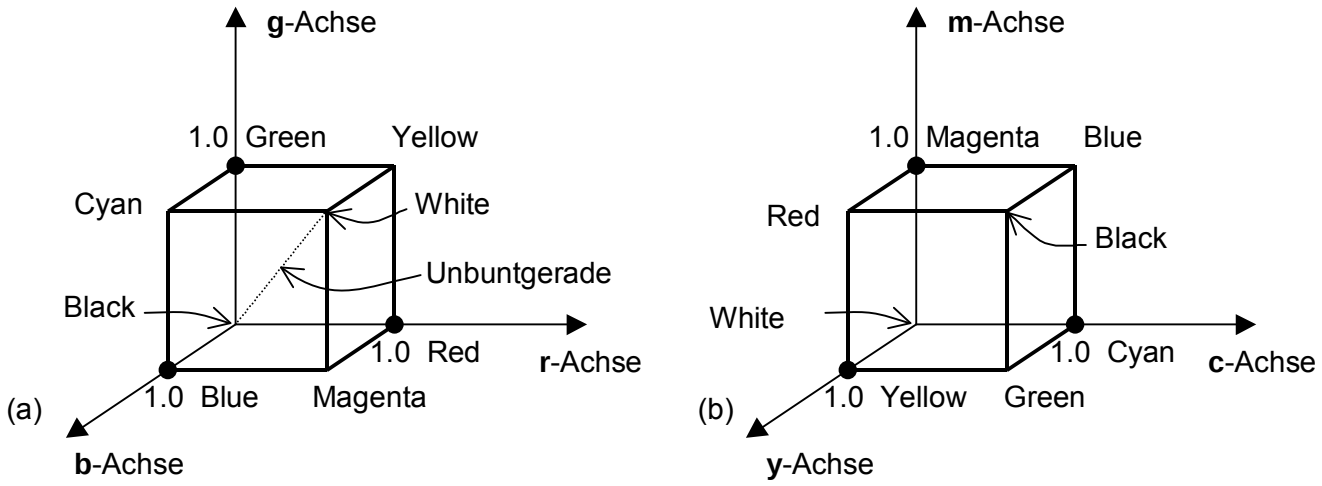


Abb. 7.6: (a) RGB-Farbmodell; (b) CMY-Farbmodell.

Auf einer weißen Fläche werden dicht benachbarte Farbpünktchen in Cyan, Gelb und Magenta vom Betrachter als Mischfarbe wahrgenommen. Nach diesem Prinzip arbeiten z.B. Farbdrucker und andere Hardcopy-Geräte.

Die Umrechnung zwischen dem RGB- und dem CMY-Farbmodell und umgekehrt ergibt sich nach folgender Formel:

$$\begin{bmatrix} c \\ m \\ y \end{bmatrix}_{CMY} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}_{RGB} - \begin{bmatrix} r \\ g \\ b \end{bmatrix}_{RGB} \quad \text{bzw.} \quad \begin{bmatrix} r \\ g \\ b \end{bmatrix}_{RGB} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}_{RGB} - \begin{bmatrix} c \\ m \\ y \end{bmatrix}_{CMY} \quad \text{mit r, g, b, c, m, y} \in [0,1].$$

RGB(A)-Farbmodell

Beim ***RGB(A)-Farbmodell*** wird das RGB-Modell um einen sogenannten ***Alpha-Kanal*** erweitert. Dabei wird zu jedem Pixel zusätzlich zu seinem rgb-Farbtripel noch ein Alpha-Wert (zwischen 0 und 1) angegeben, der die Transparenz der Pixelfarbe festlegt.

Bei der Überlagerung zweier Bilder kann so für jedes Pixel festgelegt werden, ob und in welchem Maße das „untere" Bild oder das „obere" Bild sichtbar sein soll. Mit Hilfe des Alpha-Kanals kann man also z.B. einen fließenden Übergang zwischen zwei überlagerten Bildern schaffen. Eine weitere Möglichkeit besteht darin, eine Graphik mit einer transparenten Hintergrundfarbe zu versehen; anschaulich entspricht dies dem Zeichnen der Graphik auf einer transparenten Folie. Wenn man ein derartiges Bild dann einer Textseite oder einem Bild überlagert, so ist nur der beliebig geformte Vordergrund sichtbar und nicht das üblicherweise rechteckige Gesamtbild. Diese Technik wird beispielsweise bei der Gestaltung von Internet-Seiten häufig eingesetzt, um Graphiken und andere Gestaltungselemente in einen beliebig eingefärbten Hintergrund einzubetten; (z.B. GIF-Graphiken, siehe Kapitel 9).

CMYK-Farbmodell

Im CMY-Farbmodell wird die Farbe Schwarz dadurch erzielt, dass durch einen Farbauftrag in Cyan, Magenta und Gelb alle Farben aus weißem Licht herausgefiltert werden, d.h. keine Farbe mehr reflektiert wird. In der Praxis ist es jedoch sehr unbefriedigend, die Farbe Schwarz durch einen dreifachen Farbauftrag zu erreichen, zumal man hierbei in der Regel statt Schwarz eher ein dunkles Braun erhält. Aus diesem Grund wird im ***CMYK-Modell*** noch die Farbe Schwarz (Blac**K** bzw. **K**ey) als vierte Komponente hinzugefügt. Die zusätzliche Verwendung von Schwarz ermöglicht beim Druck außerdem kontrastreichere Farbabstufungen.

7.2.2.2 Weitere Farbmodelle (YUV, HSV)

Während das RBG- und das CMY-Farbmodell von ihrer Konzeption her eher auf die Technologie von Ausgabegeräten zugeschnitten sind, erlauben andere Farbmodelle u.a. eine

intuitivere Farbdefinition durch den Benutzer. Bei diesen Modellen wird die Farbe als Kombination einer ***Intensität (Helligkeit, Luminanz)*** und zweier ***Chrominanzen*** definiert. Beispielsweise gibt beim HSV-Farbmodell (siehe unten) die erste Chrominanz den ***Farbton***, also die eigentliche Farbabstufung (Rot, Violett, Blau usw.) an, und die zweite Chrominanz bestimmt die ***Farbsättigung***, d.h. die Reinheit der Farbe. Während gesättigte Farben klar und leuchtend erscheinen, erhält man z.B. bei reflektierenden Medien durch Beimischung von Weiß ungesättigte, pastelltonartige Farben.

Es gibt zahlreiche Farbmodelle und diese können hier nicht eingehender behandelt werden. Im Folgenden soll nur ein grober und verallgemeinernder Zusammenhang zwischen dem RGB-Farbmodell, dem YUV-Modell und dem HSV-Modell vorgestellt werden.

Der Übergang vom RGB- zum ***YUV-Farbmodell*** (Y nicht verwechseln mit Yellow im CMY-Modell), ist dadurch gekennzeichnet, dass man jede Farbe durch eine (farblose) Intensität y und zwei (intensitätslose) Farben u und v kennzeichnet. Die einfachste Möglichkeit, um die Intensität eines rgb-Farbtripels $[r, g, b]^t$ zu bestimmen wäre, den Mittelwert aus dem r-, g- und b-Wert zu bestimmen:

$y = 1/3 * (r + g + b)$

Diese Gleichgewichtung der drei Farbkanäle entspricht aber nicht der menschlichen Farbwahrnehmung, die im grünen Spektralbereich wesentlich empfindlicher als im roten und insbesondere im blauen Spektralbereich ausgeprägt ist. Farbphysiologische Untersuchungen haben deshalb zu folgendem europäischen Standard geführt:

$y = 0.3*r + 0.59*g + 0.11*b$

(Der amerikanische Intensitäts-Standard im YIQ-Farbmodell unterscheidet sich geringfügig in den Gewichtungen von Rot, Grün und Blau).

Um wieder auf einen dreidimensionalen Farbraum zu kommen, müssen nun zusätzlich zur farblosen Helligkeitsachse y noch zwei weitere Achsen (Dimensionen) definiert werden. Um zwei intensitätslose Farben u und v zu erhalten, subtrahiert man von den Farbkomponenten b bzw. r des rgb-Farbtripels die Intensität y und erhält:

$u = b - y = 1.0*b - (0.3*r + 0.59*g + 0.11*b) = -0.3*r - 0.59*g + 0.89*b$ und

$v = r - y = 1.0*r - (0.3*r + 0.59*g + 0.11*b) = 0.7*r - 0.59*g - 0.11*b$

Der Zusammenhang zwischen dem RGB- und dem YUV-Farbmodell ist gegeben durch:

$$\begin{bmatrix} y \\ u \\ v \end{bmatrix}_{YUV} = \begin{bmatrix} 0.3 & 0.59 & 0.11 \\ -0.3 & -0.59 & 0.89 \\ 0.7 & -0.59 & -0.11 \end{bmatrix} * \begin{bmatrix} r \\ g \\ b \end{bmatrix}_{RGB} \quad \text{mit } r, g, b, y, u, v \in [0,1].$$

Durch Einsetzen der Farben Schwarz bzw. Weiß des RGB-Farbmodells kann man leicht nachprüfen, dass die Unbuntgerade des RGB-Farbmodelles der y-Achse des YUV-Farbmodells entspricht.

Auch beim ***HSV-Farbmodell*** (**H**ue = Farbton, **S**aturation = Sättigung, **V**alue = Intensität, Helligkeit), das beispielsweise in den meisten Bildbearbeitungsprogrammen (z.B. Gimp) Verwendung findet, wird die Unbuntgerade des RGB-Farbmodells als Intensitätsachse v verwendet. Weiterhin kann man am RGB-Würfel eine Ebene definieren, die den Weiß-Punkt enthält und senkrecht auf dieser Intensitätsachse v steht. Wenn man nun den RGB-Würfel entlang der Intensitätsachse auf diese Ebene projiziert, so erhält man die in Abbildung 7.7 gezeigte Darstellung. Die sechs farbigen Ecken des Würfels bilden in der Projektion auf die Ebene ein Sechseck, dessen Mittelpunkt der Unbuntgeraden, also allen Grautönen zwischen Schwarz und Weiß entspricht.

Zusammen mit der Intensitätsachse v erhält man hieraus den sogenannten HSV-Hexakegel aus Abbildung 7.8. Die Intensität v einer Farbe wird in der Vertikalen definiert und ihre beiden Chrominanzen h und s lassen sich in Polarkoordinaten bezüglich des Schwarz-Punktes beschreiben: der Winkel zwischen 0° und 360° gibt den Farbton h an und der Abstand

vom Mittelpunkt bestimmt die Farbsättigung s. Punkte mit großem Abstand vom Mittelpunkt entsprechen also gesättigten Farben.

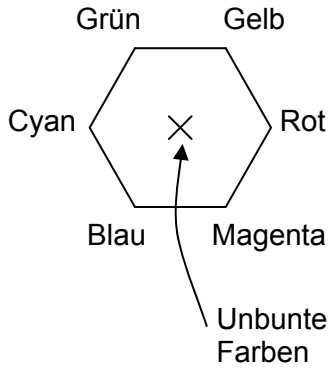


Abb. 7.7: Projektion des RGB-Würfels auf eine Ebene, die senkrecht auf der Unbuntgeraden steht.

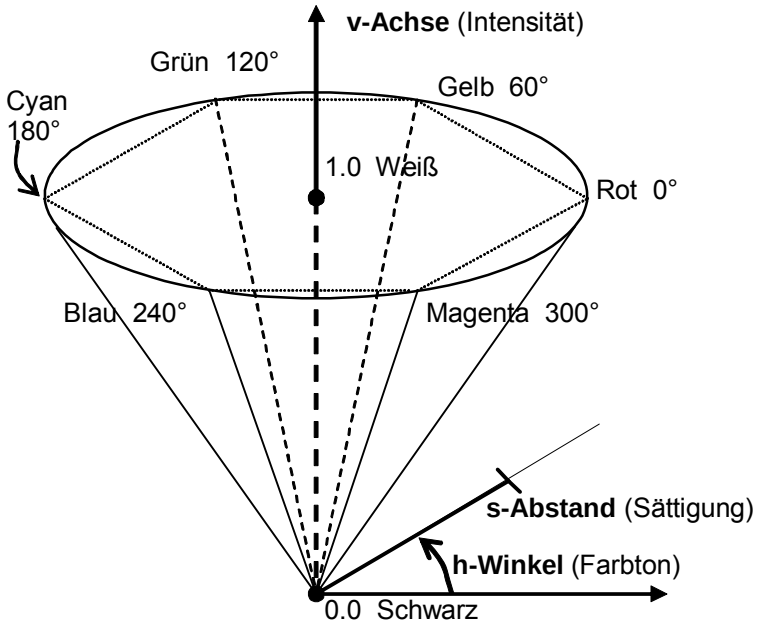


Abb. 7.8: Der HSV-Hexakegel

Im folgenden ist die Umrechnung eines Farbtripels (r,g,b) des RGB-Farbmodells in ein Tripel (h,s,v) des HSV-Farbmodells angegeben. Mit den Vereinbarungen

max = max(r,g,b), min = min(r,g,b) und delta = max - min

ergeben sich die Helligkeit v und die Sättigung s zu:

v = max bzw. s = delta/max.

(Falls max=0 ist, so wird s=0 gesetzt und der Farbton h wird als "nicht definiert" erklärt. Die Division durch „max“ ist erforderlich, damit Farben unabhängig von ihrer Helligkeit voll gesättigt sein können).

Der Farbton h ergibt sich zu:

- Falls r= max ist: $h = ((g-b)/delta)*60°$; falls $h<0°$: $h = h + 360°$;
- Falls g= max ist: $h = (2 + (b-r)/delta)*60°$;
- Falls b= max ist: $h = (4 + (r-g)/delta)*60°$.

Farbmodelle, die auf einer Intensitäts- und Chrominanz-Zerlegung basieren, bringen eine Reihe von Vorteilen mit sich:

1. Sie entsprechen in der intuitiven Farbdefinition besser der menschlichen Farbwahrnehmung und ermöglichen eine anschaulichere Beschreibung einer Farbe.
2. Sie erlauben in Übereinstimmung mit dem menschlichen Sehsystem eine Bandbreitenbegrenzung der Chrominanz-Komponenten. Da die räumliche Auflösung der Chrominanzen in unserem Sehsystem wesentlich schlechter erfolgt, als die der Intensitäten, können die Chrominanzen z.B. bei der JPEG-Bildkompression (s. Kap. 9.2) in der halben räumlichen Auflösung erfasst werden; (also: beträchtliche Platzersparnis ohne Qualitätsverluste).
3. Bei der Ausgabe auf Schwarz-Weiß-Geräten (z.B. Schwarz-Weiß-Drucker) erfordern sie keine weiteren Berechnungen, da man direkt den Intensitätskanal ausgeben kann.
4. Bei Algorithmen zur Bildverarbeitung oder Bildbearbeitung erfordern sie weniger Berechnungsaufwand, da z.B. zur Aufhellung oder Kontrastverbesserung eines Bildes nur der Intensitätskanal verändert werden muss, während beim RGB-Modell alle drei Farbkanäle betrachtet werden müssen. Außerdem ergeben sich beim RGB-Modell in solchen Fällen

häufig Farbverfälschungen, wenn z.B. eine der Farbkomponenten „in die Sättigung geht“, d.h. die zulässigen Intervallgrenzen überschreiten würde.

7.2.2.3 Rechnerinterne Repräsentation von Farbbildern

Zur Verarbeitung und Repräsentation von Farbbildern in einem Digitalrechner müssen die einzelnen Farbkanäle ***quantisiert***, d.h. auf diskrete Werte abgebildet werden. Beispielsweise ist beim RGB-Modell eine Unterscheidung von $2^8 = 256$ Werten je Farbkanal üblich, so dass man insgesamt eine ***Farbauflösung*** von $2^8 * 2^8 * 2^8 \approx 16.7$ Millionen verschiedenen Farben erhält. Man spricht in diesem Fall auch von ***True-Color***, da (fast) die Farbauflösung des menschlichen Sehsystems erreicht wird. Da zur Abspeicherung eines Pixels hier je Kanal 8 Bit, d.h. insgesamt 24 Bit erforderlich sind, benötigen True-Color-Bilder in Abhängigkeit von ihren Abmessungen sehr viel Speicherplatz.

Falls zusätzlich ein Alpha-Kanal verwendet wird, so wird dieser üblicherweise ebenfalls mit 8 Bit pro Pixel quantisiert.

Eine andere Möglichkeit zur rechnerinternen Repräsentation von Farbbildern ist durch die Verwendung einer ***Farbtabelle*** (***LUT***, für **L**ook **U**p **T**able) gegeben. Statt das volle True-Color-Spektrum abzuspeichern, werden z.B. nur die 256 „optimalen“ Farben ausgewählt und in der LUT erfasst; (siehe z.B. GIF-Bilder in Kapitel 9). Die Pixelwerte stellen jetzt keine direkten Farben mehr dar, sondern sie werden als Indizes in die LUT aufgefasst und dort stehen in der entsprechenden Zeile die RGB-Farbtripel. Im vorliegenden Fall müssen pro Pixel im Bild nur noch 8 Bits zur Abspeicherung des Index zur Verfügung gestellt werden; allerdings muss auch die LUT noch abgespeichert werden.

Auch Grauwertbilder werden mit Hilfe einer LUT dargestellt. Die Grauwerte der einzelnen Pixel stellen Indizes in eine LUT dar, die die eigentlichen Grautöne in Form von rgb-Tripeln enthält, bei denen die r-, g- und b- Einträge dieselben Werte haben; siehe Abbildung 7.9.

Wenn man in die zu einem Grauwertbild gehörende Farbtabelle „echte“ Farben einsetzt, so erhält man ein sogenanntes ***Pseudo-Farbbild***. Dies ist eine (z.B. im Bereich der Medizin) häufig eingesetzte Visualisierungstechnik, um auch kleinste Grauwertunterschiede anhand unterschiedlicher Ausgabe(-Pseudo)-Farben gut erkennen zu können.

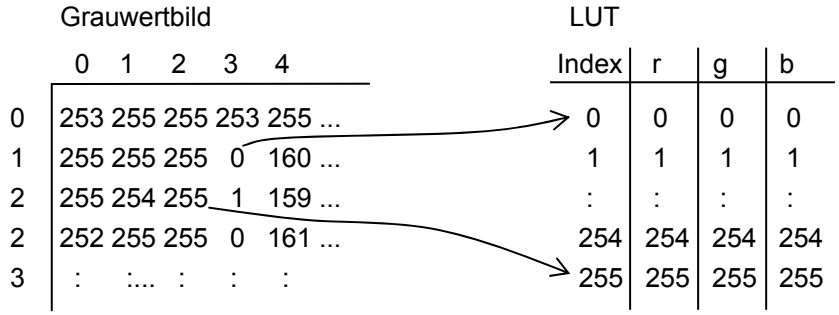


Abb. 7.9: Beispiel für die Verwendung einer LUT bei einem Grauwertbild.

Im Gegensatz zu einem Pseudo-Farbbild spricht man von einem ***Falsch-Farbbild***, wenn die Farbwerte eines Farbbildes verändert oder vertauscht wurden, so dass die im Bild enthaltenen Objekte in einer anderen (als ihrer tatsächlichen) Farbe dargestellt werden.

Da bei dem Wert eines Pixels nicht erkennbar ist, ob damit ein Grau- oder ein Farbwert gemeint ist, wird dieser Wert im Weiteren auch als ***Pixelwert*** bezeichnet.

Schluss-Bemerkung: In Bildbearbeitungs-Programmen werden Look-Up-Tabellen häufig eingesetzt, um auch für sehr große Bilder bei bestimmten Bildoperationen eine Echtzeit-Vorschau zu ermöglichen; siehe Kapitel 8.1. (Statt bei jeder Parameter-Änderung alle Pixel des Ausgabebildes bereits zu berechnen, werden (nur für die Vorschau) die Eingabe-Grauwerte zunächst mit Hilfe einer LUT auf die gewünschten Ausgabe-Grauwerte abgebildet; erst wenn die gewünschte Parameter-Einstellung erreicht und akzeptiert ist, wird das Ausgabebild Pixel für Pixel berechnet).

8 Bildbearbeitung für Rasterbilder

Die Bildbearbeitung ist häufig der erste Verarbeitungsschritt bei der ***Digitalen Bildverarbeitung***. Diese ist ein Sammelbegriff, der alle Verarbeitungsschritte vom Bildeinzug (z.B. mit Hilfe einer Kamera), über die Bildbearbeitung, die Bildsegmentierung und die Klassifizierung bis hin zur Bildinterpretation (maschinelles Sehen, Computer Vision) umfasst.

Bei der Bildbearbeitung werden die zu analysierenden Bilder photometrisch und geometrisch manipuliert (s. Kap. 8.1 bzw. 8.3). Dabei werden z.B. Helligkeiten oder Farben und Kontraste beeinflusst, Störungen eliminiert und bestimmte Sachverhalte (z.B. ***Kanten***, d.h. Grenzen zwischen unterschiedlich eingefärbten Bildbereichen) hervorgehoben, bzw. die Bilder werden z.B. in ihrer Größe geändert. Das Ergebnis einer Bildbearbeitung ist stets wieder ein Bild.

Darüber hinaus wird die Bildbearbeitung häufig auch ohne nachfolgende Auswerteschritte eingesetzt, um Bilder zu Präsentationszwecken (z.B. Bildersammlungen, Plakate, Web-Graphiken usw.) zuzuschneiden, zu überlagern bzw. zu verschmelzen, zu verbessern, zu retuschieren oder zu verfremden. Diese Arbeitsschritte, die in den Handbüchern der gängigen Bildbearbeitungsprogramme (Paint Shop Pro, Photoshop, Gimp usw.) meist ausführlich beschrieben sind, werden hier nicht betrachtet.

Im Folgenden werden nur solche grundlegenden Bildbearbeitungs-Methoden vorgestellt, die sich auch als Vorstufe zu einer Bildinterpretation einsetzen lassen. Dabei werden die Methoden nur am Beispiel von Grauwert- oder Schwarz-Weiß-Bildern behandelt. (Die Übertragung auf Farbbilder ist problemlos möglich, indem man je nach Anwendung beispielsweise alle Farbkanäle des RGB-Farbmodells oder aber z.B. nur den Helligkeits-Kanal eines anderen Farbmodells (siehe Kapitel 3.2.2.2) bearbeitet).

8.1 Photometrische Bildmanipulationen

Die Grauwerte der Bilder sollen im Weiteren zwischen $g_{MIN} = 0$ (schwarz) und g_{MAX} (weiß; meist $g_{MAX} = 255$) liegen.

Unter ***photometrischen Bildmanipulationen*** versteht man Verarbeitungsschritte, die die Helligkeit oder den Kontrast von Bildern ändern. Unter ***Kontrast*** versteht man dabei die Differenz zwischen dem minimalen Grauwert g_{min} und dem maximalen Grauwert g_{max}, die in einem Bildausschnitt bzw. dem gesamten Bild tatsächlich auftreten. Um ein objektives Kontrastmaß zu erhalten, kann diese Differenz noch zu dem Intervall der insgesamt zulässigen Grauwerte $[g_{MIN}=0, g_{MAX}]$ in Beziehung gesetzt werden:

(8-1)

$$\text{Kontrast} = \frac{g_{max} - g_{min}}{g_{MAX} - g_{MIN}} = \frac{g_{max} - g_{min}}{g_{MAX}} \qquad (\text{es gilt: Kontrast} \in [0,1]).$$

Eine gute Möglichkeit zur Beurteilung von Kontrast und Helligkeit eines Bildes bietet das sogenannte ***Histogramm***. Das Histogramm der absoluten Häufigkeiten ist eine Tabelle, die zu jedem Grauwert g angibt, mit welcher absoluten Häufigkeit h(g) er im Bild auftritt. Wenn man die jeweiligen Pixelhäufigkeiten durch die absolute Auflösung, d.h. die Gesamtzahl der Pixel des Bildes (= Spaltenzahl * Zeilenzahl) dividiert, so erhält man die relativen Häufigkeiten $h^{\#}(g)$; für diese gilt: $0 \le h^{\#}(g) \le 1$ für alle Grauwerte g und $\Sigma h^{\#}(g)=1$, wobei die Summe über alle Grauwerte läuft.

Ein Histogramm lässt sich auch als Bild darstellen; siehe z.B. mittlere Spalte von Abbildung 8.1. Aus dem Histogramm in Abbildung 8.1 (b) ist zu ersehen, dass Bild 8.1 (a) von mittlerer Helligkeit und mäßigem Kontrast ist; (keine sehr hellen bzw. sehr dunklen Grauwerte).

Ein weiteres wichtiges Maß zur Beurteilung von Bildern ist die ***Entropie*** H, die den mittleren Informationsgehalt eines Bildes angibt:

(8-2)

$$H = -\sum_{g \in [g_{MIN}, g_{MAX}]} h^{\#}(g) * \log_2(h^{\#}(g)).$$

Dabei kennzeichnet $\log_2(h^{\#}(g))$ den mittleren Informationsgehalt des Grauwertes g.

Die Entropie eines Bildes gibt im Prinzip an, wie viele Bits zur Abspeicherung eines Bildpunktes erforderlich sind, d.h. in welchem Umfang sich der Speicherplatzbedarf eines Bildes mit geeigneten Bildkompressionstechniken (s. Kapitel 9) mindestens reduzieren lässt.

Im Folgenden werden einige Möglichkeiten zur Änderung von Bildhelligkeit und -kontrast, sowie zur „Glättung" der Bildfunktion (Weichzeichnen und Störungselimination) und zur Hervorhebung von Grauwertübergängen (z.B. Kanten-Detektion) angegeben.

8.1.1 Punkt-Operatoren

Bei den ***Punkt-Operatoren*** wird eine Berechnungsvorschrift auf alle Punkte eines Bildes angewendet. Im Gegensatz zu den „lokalen Bildoperatoren" (siehe Kapitel 8.1.2) hängt das Ergebnis des Operators hier nur von der gewählten Transformation und dem Grauwert des jeweils zu transformierenden Bildpunktes ab. Mit Hilfe von Punkt-Operatoren können die Helligkeit und der Kontrast eines Bildes auf besonders einfache Weise verändert werden.

Punkt-Operatoren werden in gängigen Bildbearbeitungs-Programmen häufig mit Hilfe einer LUT (Look Up Table; siehe Kapitel 7.2.2.3) realisiert, um eine Echtzeit-Vorschau auch für sehr große Bilder zu ermöglichen.

8.1.1.1 Lineare Grauwerttransformation

Bei einer (stückweise) ***linearen Grauwerttransformation*** errechnet sich der neue Grauwert g'(i,j) aus dem alten Grauwert g(i,j) nach folgender Vorschrift:

(8-3)

$g'(i,j) = \texttt{mult} * g(i,j) + \texttt{add}$ (mult, add reelle Zahlen).

Um sicherzustellen, dass die Grauwerte das zulässige Intervall $[0, g_{MAX}]$ nicht verlassen, wird zusätzlich vereinbart:

(8-3a)

$$g'(i,j) = \begin{cases} g_{MIN} = 0 & \text{falls } \texttt{mult} * g(i,j) + \texttt{add} < g_{MIN} = 0 \\ g_{MAX} & \text{falls } \texttt{mult} * g(i,j) + \texttt{add} > g_{MAX} \end{cases}$$

(Durch dieses „Abschneiden" kann Bildinformation verloren gehen: vorher unterscheidbare helle bzw. dunkle Bildbereiche werden u.U. auf ein einheitliches Weiß bzw. Schwarz abgebildet. Korrekterweise muss die Transformation wegen (6-3a) als "stückweise" lineare Grauwerttransformation bezeichnet werden).

Der Zusammenhang zwischen den alten Grauwerten g und den neuen Grauwerten g' wird häufig in einem g-g'-Koordinatensystem dargestellt. Dabei wird nur das zweidimensionale Intervall $[0, g_{MAX}]*[0, g_{MAX}]$ betrachtet.

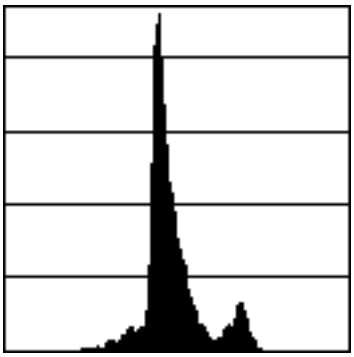
Die Abbildungen 8.1 (a) und (b) zeigen ein Grauwertbild mit g_{MAX}=255 und sein zugehöriges Histogramm. In Abbildung 8.1 (c) ist der Verlauf der identischen Grauwerttransformation (mult =1 und add =0) dargestellt, bei der das Eingangsbild nicht verändert wird.

Durch eine Erhöhung bzw. Verringerung des Summanden add in (8-3) erreicht man eine Parallelverschiebung der Diagonalen aus Abbildung 8.1 (c). Dies entspricht einer Aufhellung bzw. Abdunkelung des Bildes, ohne dass der Bildkontrast beeinflusst wird; siehe (8-1). Die Abbildungen 8.1 (d) bis (f) zeigen ein Beispiel für eine lineare Grauwerttransformation mit add = 75 (und mult = 1). Sehr helle Bildpartien des Eingangsbildes sind nun einheitlich weiß; siehe Balken am rechten Rand des Histogrammes.

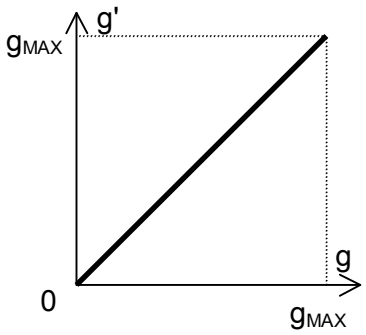
Mit Hilfe des Faktors mult kann man den Bildkontrast - siehe (8-1) - beeinflussen: für mult>1 nimmt der Bildkontrast zu, und für 0<mult<1 wird das Bild kontrastärmer. Ein Beispiel für eine Kontraständerung mit mult=1.5 (und add =0) ist in Abbildung 8.1 (g) bis (i) dargestellt. Auch hier sind sehr helle Bildstellen nach der Transformation wieder einheitlich weiß.

Durch geeignete Wahl von mult und add kann man erreichen, dass der zur Verfügung stehende Grauwertbereich optimal genutzt wird. Wenn g_{min} und g_{max} den kleinsten und den größten im Eingabebild tatsächlich auftretenden Grauwert bezeichnen, so erhält man mit $\texttt{mult} = g_{MAX}/(g_{max} - g_{min})$ und $\texttt{add} = -g_{min} * \texttt{mult}$ die Grauwerttransformation:

(a)

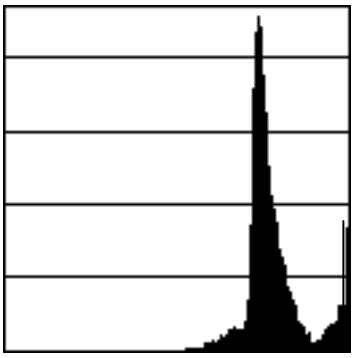


(b)

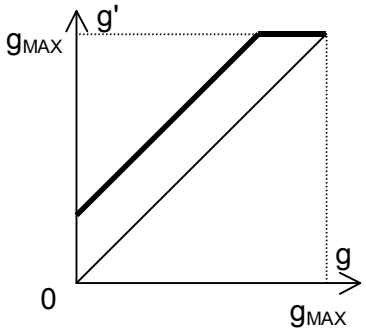


(c)

(d)

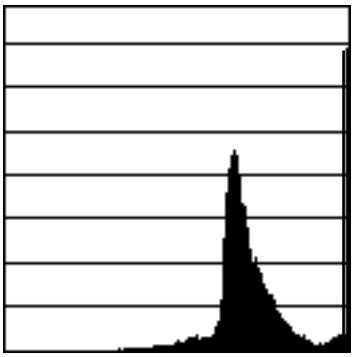


(e)

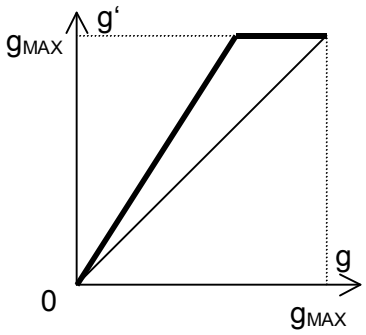


(f)

(g)

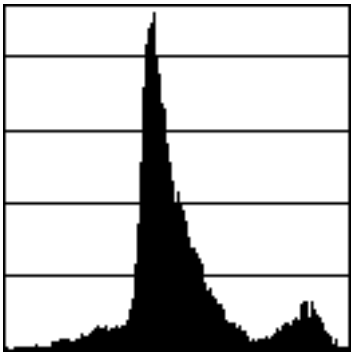


(h)

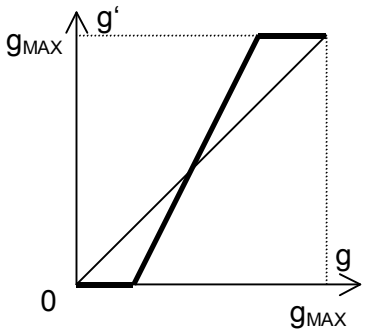


(i)

(j)



(k)



(l)

Abb. 8.1: Beispiele für die lineare Grauwerttransformation:
(a)-(c) Eingangsbild, Histogramm und identische Grauwerttransformation;
(d)-(f) aufgehelltes Bild, Histogramm und Grauwerttransformation;
(g)-(i) Bild mit gespreiztem Kontrast, Histogramm und Grauwerttransformation;
(j)-(l) optimal transformiertes Bild, Histogramm und Grauwerttransformation.

(8-4)

$$g' = \frac{g_{MAX} * (g - g_{min})}{g_{max} - g_{min}}.$$

Nach dieser Transformation wird das zur Verfügung stehende Grauwertintervall [0, g_{MAX}] vollständig ausgenutzt; siehe Abbildung 8.1 (j) bis (l).

Häufig wird eine derartige lineare Transformation auch eingesetzt, wenn das Ergebnis einer Bildbearbeitungsoperation eine „Bild"-Matrix mit nicht darstellbaren Werten e<0 oder e>g_{MAX} liefert; (z.B. Ergebnis einer Faltung aus Kapitel 8.1.2.1). Eine Transformation entsprechend (8-4) transformiert die Ergebniswerte dann auf das zulässige Grauwert-Intervall [g_{MIN}=0, g_{MAX}], so dass das Ergebnisbild anschließend dargestellt werden kann; siehe (8-9).

Ein weiterer Spezialfall der stückweise linearen Grauwert-Transformation, die Binarisierung, wird in Kapitel 8.2.1 vorgestellt.

8.1.1.2 Gamma-Korrektur

Eine weitere Punktoperation zur Beeinflussung der Bildhelligkeit und des Bildkontrastes stellt die ***Gammakorrektur*** dar, die eine nicht-lineare Veränderung der Grauwerte bewirkt.

Die Gammakorrektur wird vor allem verwendet, um die nicht-linearen Grauwertverfälschungen auszugleichen, die z.B. bei der Aufnahme eines Bildes mit Hilfe einer CCD-Kamera (Charge Coupled Device) oder bei der Ausgabe eines Bildes auf einem Monitor zwangsläufig entstehen. Die Ursache für diese Verfälschungen ist in der Nichtlinearität der jeweiligen Bauelemente zu sehen: ein CCD-Sensorelement liefert bei einer Verdoppelung der empfangenen Lichtmenge nicht die doppelte Ausgangsspannung (d.h. keinen doppelt so hohen Grauwert), und umgekehrt bewirkt die Verdoppelung eines Bildpunkt-Grauwertes nicht die doppelte Helligkeit an der entsprechenden Monitorposition.

Beispielsweise gilt für einen Monitor, dessen Bildschirmpunkte die Grauwerte $g \in [0, g_{MAX}]$ mit einer Intensität $I \in [0, I_{MAX}]$ abbilden, die folgende Beziehung:

(8-5)

$$\frac{I}{I_{MAX}} = \left(\frac{g}{g_{MAX}}\right)^{\gamma}$$

wobei der Gammawert γ typischerweise zwischen 1.3 und 2.4 liegt. Damit die Ausgabe-Intensitäten den tatsächlichen (Grauwert-)Gegebenheiten entsprechen, müssen die Grauwerte vor der Bildausgabe mit Hilfe des Kehrwertes von γ korrigiert werden:

(8-6)

$$\frac{g'}{g_{MAX}} = \left(\frac{g}{g_{MAX}}\right)^{1/\gamma} \quad \text{d.h.} \quad g' = \left(\frac{g}{g_{MAX}}\right)^{1/\gamma} * g_{MAX}$$

(Dabei liegen die Quotienten g/g_{MAX} und g'/g_{MAX} im Intervall [0,1.)]

Bei der Bildaufnahme besteht zwischen der einfallenden Lichtenergie und dem daraus erzeugten Grauwert eine (8-5) entsprechende Beziehung, wobei hier die Gammawerte üblicherweise zwischen 0.4 und 1.0 liegen.

Abbildung 8.2 zeigt ein Beispiel für eine Gammakorrektur nach (8-6). Auf der linken Seite ist ein Eingangsbild einschließlich eines synthetischen Graukeils mit gleichmäßig ansteigenden Grauwerten zu sehen. In dieser unkorrigierten Darstellung sieht man, dass durch die spezifische Grauwertverfälschung bei der Bildausgabe große Teile des Graukeils (und damit auch des restlichen Bildes) zu dunkel abgebildet werden. In Abbildung 8.2 (b) ist das Ergebnis der Gammakorrektur mit γ = 1.8 dargestellt. Der Funktionsverlauf dieser Gammakorrektur ist in Abbildung 8.2 (c) als durchgezogene Linie eingezeichnet.

Schlussbemerkungen zur Gammakorrektur:

Falls der Gammawert bei der Bildaufnahme der Kehrwert des Gammawertes bei der Bildausgabe ist, so gleichen sich die Verzerrungen aus und man muss keine Gammakorrektur durchführen. Dieser Fall ist aber in der Realität fast nie gegeben.

(a) (b) (c)

Abb. 8.2: Beispiel für die Gammakorrektur: (a) Eingangsbild mit Graukeil; (b) gammakorrigiertes Bild; (c) Funktionsverlauf der Gammakorrektur. (Die im Text angesprochenen Effekte sind hier durch den Druck möglicherweise beeinträchtigt).

In der Praxis sind (z.B. im Internet) viele Bilddateien bereits im Hinblick auf einen durchschnittlichen Monitor gammakorrigiert, so dass keine weitere Bearbeitung erforderlich ist.

Nicht vorkorrigierte Bilddaten müssen mit Hilfe des monitorspezifischen Gammawertes nach der Vorschrift (8-6) korrigiert werden, um eine optimale Darstellung zu erzielen. Den monitorspezifischen Gammawert ermittelt man am besten mit Hilfe eines Messprogramms, wie es z.B. in den meisten Bildbearbeitungsprogrammen enthalten ist.

Auch für Scanner, Kameras und andere Geräte zum Bildeinzug kann der spezifische Gammawert z.B. mit Hilfe genormter Grauwertverläufe (z.B. Graukeil) oder spezieller Messgeräte (Densitometer) ermittelt werden.

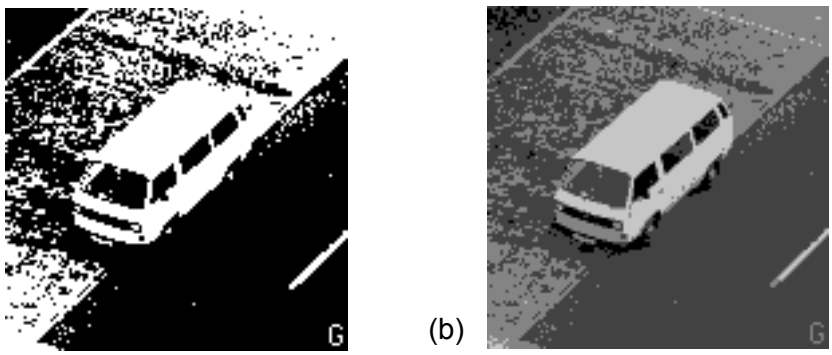
8.1.1.3 Reduktion der Anzahl verschiedener Grauwerte

Ein häufig eingesetztes Mittel, um Bilder künstlerisch zu verfremden oder Speicherplatz einzusparen ist, die Anzahl der zulässigen Graustufen eines Bildes zu reduzieren.

Wenn man alle Grauwerte g, deren höchstwertiges Bit in der binären Darstellung des Grauwertes nicht gesetzt ist, auf den neuen Grauwert $g'_{MIN} = 0$ (schwarz), und alle Grauwerte mit gesetztem höchstwertigen Bit auf $g'_{MAX} = 1$ (weiß) transformiert, so erhält man ein Binärbild; siehe Abbildung 8.3 (a). (Eine flexiblere Alternative zur Erstellung eines ***Binärbildes*** wird in Kapitel 8.2.1 vorgestellt).

Statt nur das höchstwertige Bit zu berücksichtigen, kann man auch die n = 1, 2, 3, ... höchstwertigen Bits auswerten und erhält damit eine äquidistante Aufspaltung des ursprünglichen Grauwertintervalls und ein neues Bild mit 2^n Graustufen. Die Abbildung 8.3 (b) zeigt ein Beispiel für ein derartiges ***Äquidensitenbild*** (auch: „Poster"), wobei hier n= 2 gewählt wurde.

Äquidensitenbilder (oder allgemeiner Binärbilder) werden häufig auch als Vorstufe für eine ***Bildsegmentierung*** erzeugt; siehe Kapitel 8.2.



(a) (b)

Abb. 8.3: Beispiele für die Reduzierung der Grauwert-Anzahl zu Abbildung 8.1 (j):
(a) Äquidensitenbild unter Berücksichtigung des höchstwertigen Bits;
(b) Äquidensitenbild unter Berücksichtigung der beiden höchstwertigen Bits.

8.1.2 Lokale Bildoperatoren

Bei den ***lokalen Bildoperatoren*** wird – wie bei den Punkt-Operatoren – eine Berechnungsvorschrift parallel auf alle Bildpunkte angewendet, um die alten Grauwerte g(i,j) des Eingangsbildes B in die neuen Grauwerte g‘(i,j) des Ergebnisbildes B‘ zu überführen. Im Gegensatz zu den Punkt-Operatoren werden dabei aber die Grauwerte aus der Nachbarschaft der jeweiligen Position (i,j) mitberücksichtigt.

(8-7)

In der Regel wird eine der beiden folgenden Nachbarschaftsbeziehungen verwendet:

- die ***4er-Nachbarschaft (N4)***, die aus den horizontalen und den vertikalen Nachbarn an den Positionen (i,j-1), (i-1,j), (i+1,j) und (i,j+1) besteht (siehe Abbildung 8.4 (links)),
- und die ***8er-Nachbarschaft (N8)***, die zusätzlich die diagonalen Nachbarn (i-1,j-1), (i+1,j-1), (i-1,j+1) und (i+1,j+1) umfasst; (siehe Abbildung 8.4 (rechts)).

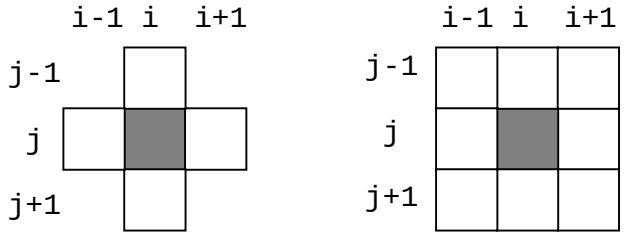


Abb. 8.4: N4-Nachbarschaft (links) und N8-Nachbarschaft (rechts).

Im Weiteren wird der Einfachheit halber meist die N8-Nachbarschaft verwendet; möglich wären aber auch die N4-Nachbarschaft oder "größere" Nachbarschaften z.B. aus der 5*5- oder 7*7-Umgebung eines Pixels.

Als Beispiele für lokale Bildoperatoren werden im Folgenden die Faltung und die Rangfolge-Operatoren betrachtet. Dabei werden diese beiden Operatoren hier hauptsächlich zur Glättung des Bildes, zur Markierung von Kanten und zur Hervorhebung von Grauwertübergängen eingesetzt. Die Bilder sollen M Spalten und N Zeilen besitzen.

8.1.2.1 Faltung

Bei der ***Faltung*** (engl. convolution) zwischen einem Bild B mit Grauwerten g(i,j) und einer 3*3-Faltungsmatrix F mit Werten f(k,l) wird die Faltungsmatrix über alle Positionen (i,j) des Bildes mit 0<i<M-1 und 0<j<N-1 gelegt. Die Elemente der Faltungsmatrix werden auch als Gewichte bezeichnet. An jeder Position (i,j) wird ein Ergebniswert e(i,j) als gewichtete Summe nach folgender Vorschrift berechnet:

(8-8)

$$e(i,j) = \sum_{l=0}^{2}\sum_{k=0}^{2}\{g(i-1+k, j-1+l) * f(k,l)\}.$$

Die vorderste und die hinterste Spalte sowie die oberste und die unterste Zeile der Ergebnismatrix E wird auf 0 gesetzt, um Sonderfallbehandlungen zu vermeiden.

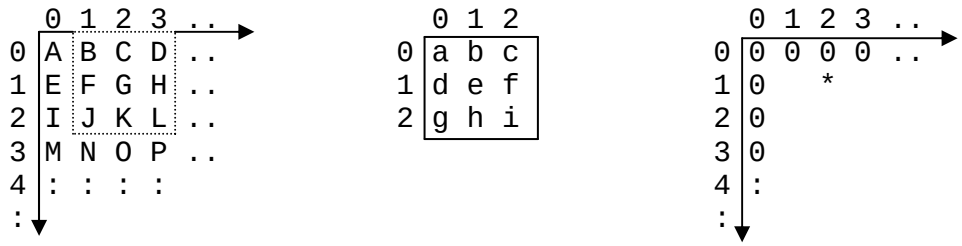
Die Ergebniswerte e(i,j) sind i.d.R. keine zulässigen Grauwerte; (sie können z.B. sehr groß, oder auch negativ sein). Um die Ergebnismatrix als Bild ausgeben zu können, wird deshalb üblicherweise abschließend eine lineare Transformation ähnlich zu (8-4) durchgeführt, die die Ergebniswerte e(i,j) in das Intervall $[g_{MIN}=0, g_{MAX}]$ abbildet:

(8-9)

$$g'(i,j) = \frac{g_{MAX} * (e(i,j) - e_{min})}{e_{max} - e_{min}}.$$

Dabei stehen e_{min} bzw. e_{max} für den minimalen bzw. den maximalen tatsächlich auftretenden Ergebniswert der Faltung. Der Einfachheit halber könnten hier aber auch die theoretisch möglichen Extremwerte e_{MIN} bzw. e_{MAX} verwendet werden; das Ergebnisbild verliert dadurch i.d.R. aber deutlich an Kontrast.

Beispiel: Bild B Faltungsmatrix F Ergebnismatrix E



An der mit einem Sternchen markierten Stelle in der Ergebnismatrix E erhält man z.B.:

e(2,1) = B*a + C*b + D*c + F*d + G*e + H*f + J*g + K*h + L*i.

Die Faltung mit der Faltungsmatrix F_I (mit I für Identität)

(8-10)

$F_I =$

0	0	0
0	1	0
0	0	0

verändert ein Bild nicht, wenn man von dem ein-pixel-breiten schwarzen Rand im Ergebnisbild absieht. Diese Faltung wird hier nur vorgestellt, da sie als Teil einiger späterer Faltungen verwendet wird.

Im Folgenden werden einige anwendungsorientierte Beispiele für Faltungen angegeben.

Glättung der Bildfunktion (Weichzeichnen).

Mit der Faltungsmatrix

(8-11)

$F_M =$

1	1	1
1	1+n	1
1	1	1

(beispielsweise mit n= 0).

wird an jeder Bildposition der gewichtete Mittelwert der Grauwerte aus der 3*3-Umgebung eingesetzt. Durch den ***gewichteten Mittelwert-Operator*** werden Störungen verwischt, so dass das Bild „glatter“ wirkt. Als nachteiliger Effekt werden die Kanten im Bild unscharf; siehe Abbildung 8.4 (b).

Ein weiteres Beispiel zur ***Glättung (Weichzeichnung)*** des Bildes stellen sogenannte Gauss-Filter dar, die einen Tiefpass-Filter (siehe Kapitel 9) realisieren. Das bedeutet, dass die hohen Frequenzanteile, die den scharfen Kanten entsprechen, herausgefiltert werden und nur die niedrigeren Frequenzen „passieren“ können. Ein Beispiel für einen Gauss-Filter erhält man mit der Faltungsmatrix F_G mit:

(8-12)

$F_G =$

1	2	1
2	4	2
1	2	1

Kantendetektion und Hervorhebung von Kanten

Die im Folgenden angegebenen Faltungsmatrizen sind ***Kantendetektoren***, die Kanten senkrechter, waagrechter sowie diagonaler Ausrichtung ermitteln:

(8-13)

$F_S =$

-1	0	1
-1	0	1
-1	0	1

, $F_w =$

-1	-1	-1
0	0	0
1	1	1

, $F_{D1} =$

0	-1	-1
1	0	-1
1	1	0

und $F_{D2} =$

-1	-1	0
-1	0	1
0	1	1

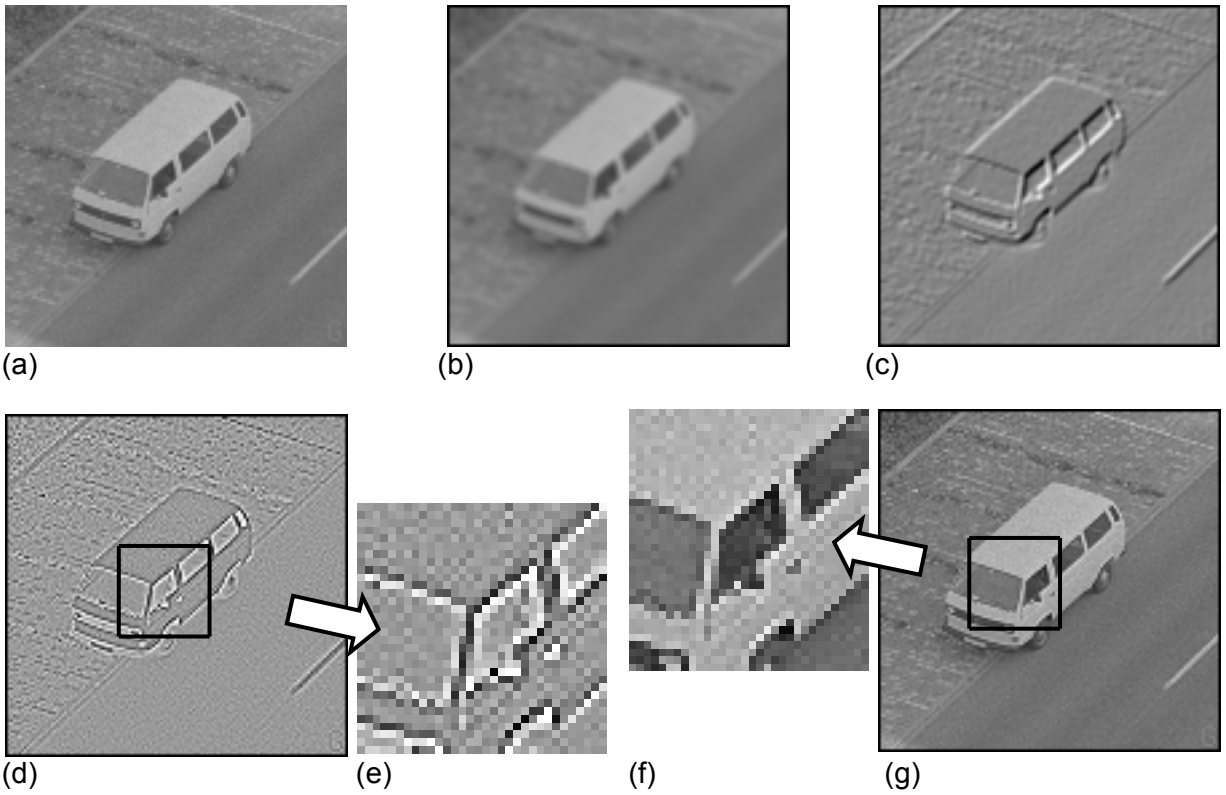


Abb. 8.4: Beispiele für Faltungen:
(a) Eingangsbild;
(b) Mittelwert-Operator zur Glättung der Bildfunktion (Weichzeichnen);
(c) Kantendetektor für diagonal (/) verlaufende Kanten;
(d) Laplace-Operator zur Kantendetektion;
(e) Ausschnittsvergrößerung aus dem Ergebnis des Laplace-Operators;
(f) Ausschnittsvergrößerung zum Konturhervorhebungs-Operator;
(g) Konturhervorhebungs-Operator.

Abbildung 8.4 (c) zeigt ein Beispiel für die Anwendung des Kanten-Filters F_{D2}. Die mittelgrauen Bildbereiche entsprechen Positionen mit einem Faltungswert $e \approx 0$, d.h. es wurden dort keine Kanten detektiert.

Die oben aufgeführten Faltungen basieren auf einer partiellen Ableitung der Bildfunktion in waagrechter, senkrechter bzw. diagonaler Richtung.

Durch eine Addition der vier Kantendetektoren aus (8-13) erhält man den ***Kanten-Kombinationsfilter*** F_K, der Kanten aller Richtungen ermittelt:

(8-14)

$$F_K = F_S + F_W + F_{D1} + F_{D2} = \begin{bmatrix} -3 & -3 & -1 \\ -1 & 0 & 1 \\ 1 & 3 & 3 \end{bmatrix} \quad \text{(gleichwertig zu: } \begin{bmatrix} -1 & -1 & -1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\text{).}$$

Einen ***Relief-Filter*** F_R, der die Kanten einer bestimmten Richtung im Bild hervorhebt, erhält man, wenn die Matrizen der identischen Faltung F_I und eines ausgewählten Kantendetektors F_S, F_W, F_{D1} oder F_{D2} gewichtet addiert; z.B.:

(8-15)

$$F_R = n*F_I + F_{D2} = \begin{bmatrix} -1 & -1 & 0 \\ -1 & n & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \text{(beispielsweise mit n= 5).}$$

Ein weiterer Operator zur Kantendetektion ist der ***Laplace-Operator*** mit der Faltungsmatrix F_L:

(8-16)

$$F_L = \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$

Die Anwendung dieser Faltung entspricht einer zweifachen partiellen Ableitung der Bildfunktion in senkrechter und waagrechter Richtung; siehe Abbildung 8.4 (d). Die Ausschnittsvergrößerung in Abbildung 8.4 (e) zeigt, dass im Bereich von Kanten aller Richtungen je ein dunkler und ein heller Streifen entsteht. Die exakte Position der Kante ist der Übergang zwischen diesen beiden Streifen, der einem Nulldurchgang der zweiten Ableitung entspricht.

Durch eine gewichtete additive Verknüpfung der identischen Faltung mit dem Laplace-Operator erhält man den ***Konturhervorhebungs-Operator*** F_V; siehe Abb. 8.4 (g) und (f):

(8-17)

$$F_V = n*F_I - F_L = \begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & n+4 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array}$$ (beispielsweise mit n= 5).

8.1.2.2 Rangfolge-Operatoren

Bei den ***Rangfolge-Operatoren*** werden – wie bei der Faltung – an jeder Bildposition (i,j) der Grauwert der aktuellen Stelle und die Grauwerte einer Nachbarschaft (z.B. N8-Nachbarschaft) betrachtet. (Im Anhang zu diesem Kapitel finden Sie ein Beispiel für die Anwendung unterschiedlicher Rangfolge-Operatoren im eindimensionalen Fall).

Die Grauwerte des aktuellen Pixels und die der Nachbarschaft werden der Größe nach sortiert, so dass sich z.B. im Falle der N8-Nachbarschaft die folgende Sortierung ergibt:

(8-18)

$$g_0 \le g_1 \le \ ... \ \le g_7 \le g_8.$$

Im Weiteren wird an einigen Beispielen gezeigt, wie man durch die Wahl bestimmter Positionen aus dieser Sortierung z.B. Bildstörungen wirkungsvoll eliminieren oder Kanten im Bild ermitteln kann.

Wenn man aus der Sortierung den Median, d.h. den in der Mitte stehenden Wert $g'(i,j) = g_4$ auswählt, so erhält man den ***Median-Operator***:

(8-19)

$$g'(i,j) = g_4,$$

Dieser Operator unterdrückt Störungen (wie z.B. Bildrauschen), ohne dabei jedoch die Kanten im Bild zu verschleifen, wie dies beim Mittelwert-Operator (8-11) der Fall ist; siehe Abbildung 8.5 (b).

Abb. 8.5: Beispiele für Rangfolge-Operatoren:
(a) Eingangsbild;
(b) Median-Operator zur Störungselimination;
(c) Spannweiten-Operator zur Detektion von Kanten aller Richtungen.

Zwei weitere Rangfolge-Operatoren, mit deren Hilfe z.B. einfache „Störungen“ an Objekten erkannt und eliminiert werden können, sind die ***Dilatation*** mit

$$g'(i,j) = g_8 \qquad (8\text{-}20)$$

und die ***Erosion*** mit

$$g'(i,j) = g_0. \qquad (8\text{-}21)$$

Im Weiteren sollen die Objekte stets heller als der Hintergrund sein; (falls dies nicht der Fall ist, kehrt sich die Wirkung der Dilatation bzw. Erosion um). Bei der Dilatation wachsen helle Objekte, indem auf allen Seiten eine ein-Pixel-breite Schicht von Objektpixeln angelagert wird. (Bei Binärbildern bedeutet dies vereinfacht, dass alle Hintergrundpixel, die in ihrer N8-Nachbarschaft ein Objektpixel besitzen, zu Objektpixeln umgefärbt werden). Bei einer Dilatation schließen sich Risse, Lücken und Löcher, deren Breite kleiner-gleich zwei Pixel ist. Durch mehrfache Anwendung der Dilatation (***n-fache Dilatation***) kann man auch breitere Risse, Lücken und Löcher schließen. Ein Beispiel für eine zweifache Dilatation ist in Abbildung 8.6 (b) dargestellt.

(a)  (b)  (c) 

(d)  (e) 

Abb. 8.6: Beispiel für die Elimination von einfachen „Störungen“ an hellen Objekten:
(a) Beispielbild mit Rissen, Löchern und Ausläufern;
(b) Anwendung einer zweifachen Dilatation;
(c) anschließend zweifache Erosion; Ergebnis: zweifaches Closing;
(d) anschließend weitere zwei Erosionen;
(e) abschließend zweifache Dilatation; Ergebnis: zweifaches Opening bezüglich des in (c) gezeigten Closing-Bildes.

Bei der Erosion schrumpfen helle Objekte um eine Pixelschicht. (Bei Binärbildern entspricht dies der Umfärbung aller Objektpixel, die ein Hintergrundpixel in ihrer N8-Nachbarschaft besitzen, zu Hintergrundpixeln). Bei einer Erosion verschwinden Objekte und Ausläufer, deren Breite kleiner-gleich zwei Pixel ist. Wenn man auch breitere Objekte bzw. Ausläufer verschwinden lassen will, so wendet man die Erosion mehrfach hintereinander an: ***n-fache Erosion***. Abbildung 8.6 (c) zeigt ein Beispiel für eine zweifache Erosion, die in diesem Falle auf Abbildung 8.6 (b) angewendet wird.

Obwohl die Dilatation und die Erosion gegensätzlich definiert sind, sind sie nicht invers zueinander. Bei den Kombinationen Dilatation gefolgt von Erosion bzw. Erosion gefolgt von Dilatation erhält man wieder ein Objekt, das ungefähr die Größe des ursprünglichen Objektes hat, jedoch können sich bestimmte Sachverhalte geändert haben. So schließen sich z.B. bei einer Dilatation/Erosion ein bis zwei Pixel breite Spalten und Löcher, weshalb diese Kombination auch als ***Closing*** bezeichnet wird. Umgekehrt verschwinden bei der als ***Opening*** bezeichneten Kombination Erosion/Dilatation ein bis zwei Pixel breite Objekte bzw. Ausläufer.

Um größere Lücken zu schließen (und die Objektgröße ungefähr beizubehalten) kann man beim ***n-fachen Closing*** auch n Dilatationen und danach n Erosionen durchführen. Entsprechend kann man auch größere Objekte verschwinden lassen, indem man durch ***n-faches Opening*** die Objekte erst n-fach erodiert und danach n-fach dilatiert. Häufig werden Closing und Opening nacheinander ausgeführt.

Abbildung 8.6 (c) zeigt das Ergebnis der Anwendung eines zweifachen Closing auf das Ausgangsbild aus Abbildung 8.6 (a). Um nun auch noch den kleinen Ausläufer an der unteren quadratischen Fläche zu löschen, wird anschließend zuerst eine zweifache Erosion (siehe Abbildung 8.6 (d)) und abschließend eine zweifache Dilatation durchgeführt. Das Ergebnis dieses zweifachen Openings ist in Abbildung 8.6 (e) dargestellt.

Einschub: Ermittlung von Unterschieden zwischen zwei Bildern

Wenn man z.B. im Rahmen einer Qualitätssicherung Risse, Löcher und Ausläufer eines Objektes ermitteln möchte, so führt man für das Binärbild B', das das Objekt enthält, ein n-faches Closing und danach ein n-faches Opening durch und bildet anschließend ein ***Differenzbild*** D zwischen dem Binärbild B' und dem so erhaltenen Ergebnisbild B'', indem man an allen Bildpositionen die Werte g''(i,j) des Ergebnisbildes von den Werten g'(i,j) des Binärbildes abzieht und den Betrag bildet:

(8-22)

$$D = B' - B'' \quad \text{bzw. pixelweise} \quad d(i,j) = \left| g'(i,j) - g''(i,j) \right|.$$

Im Differenzbild sind nun alle Stellen, an denen sich das Binärbild und das Ergebnisbild unterscheiden, von Null verschieden. Dies sind i.d.R. genau die zu ermittelnden Risse, Löcher und Ausläufer.

Abbildung 8.7 zeigt für das Beispielbild aus Abbildung 8.6 die ermittelten Löcher, Risse und Ausläufer. In Abbildung 8.7 (a) ist nochmals das Ausgangsbild dargestellt. Daneben ist das von Störungen bereinigte Endergebnis aus Abbildung 8.6 (e) gezeigt. In Abbildung 8.7 (c) sind nach einer Differenzbildung zwischen 8.7 (a) und 8.7 (b) nur noch die Objektdefekte zu sehen. Um diese Störungen zu den Objekten in Beziehung zu setzen, ist in Abbildung 8.7 (d) eine Überlagerung des Ausgangsbildes mit den ermittelten Defekten dargestellt.

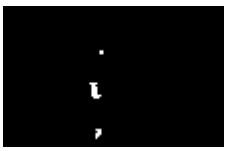
(a)  (b)  (c)  (d) 

Abb. 8.7: Beispiel für die Bilddifferenz zur Anzeige von Rissen, Löchern und Ausläufern:
(a) Ausgangsbild mit Störungen aus Abbildung 8.6 (a);
(b) Ergebnisbild mit eliminierten Störungen aus Abbildung 8.6 (e);
(c) Differenzbildung zwischen dem Ausgangsbild und dem zweifachen Closing-Opening-Bild aus Abbildung 8.6 (e);
(d) Überlagerung des Ausgangsbildes mit den ermittelten Störungen.

Abschließend soll noch der ***Spannweiten-Operator***

(8-23)

$$g'(i,j) = g_8 - g_0$$

vorgestellt werden, der ein „kombinierter" Rangfolge-Operator ist, der als Kantendetektor wirkt und Kanten aller Ausrichtungen ermittelt; siehe Abbildung 8.5 (c).

8.2 Einführung in die kontur- und flächenorientierte Segmentierung

In den vorangegangenen Teilkapiteln wurden photometrische Operatoren der Bildbearbeitung vorgestellt, bei denen aus einem Eingabebild wieder ein Bild erzeugt wird. Das jetzt folgende Teilkapitel soll eine kurze Einführung in die Bildverarbeitung geben, bei der z.B. Objekte in Bildern „erkannt" oder Bilder ganz oder teilweise interpretiert werden, um so beispielsweise die autonome Navigation eines Roboters zu ermöglichen.

Der erste Schritt hierzu besteht i.d.R. aus einer ***Bildsegmentierung***, d.h. der Zerlegung eines Bildes in sogenannte ***Regionen***. Diese Regionen sollen maximal sein, d.h. bestimmte Regionen sollen möglichst den einzelnen Objekten oder zumindest unterschiedlich eingefärbten Objektteilen entsprechen. Andere Regionen kennzeichnen den Hintergrund des Bildes. Das Ergebnis einer Bildsegmentierung kann zwar auch wieder als Bild dargestellt werden, es kann aber z.B. auch aus einer Liste aller beteiligten Regionen bestehen.

Für diese Bildsegmentierung gibt es zwei duale Vorgehensweisen:

- die ***konturorientierte Segmentierung***, die die Regionen über ihre Konturen, d.h. ihre Grenzlinien zu anderen Regionen ermittelt, und
- die ***flächenorientierte Segmentierung***, die die Regionen über ihr „Inneres", d.h. die Menge aller Bildpunkte, die zu der jeweiligen Region gehören, ermittelt.

Im Folgenden wird nur auf die flächenorientierte Segmentierung eingegangen. Bei dieser Methode werden (vereinfacht ausgedrückt) Zusammenhangskomponenten gebildet, die bezüglich irgendeines Merkmals (z.B. Grauwert) homogen, d.h. einheitlich sind. Diese Zusammenhangskomponenten bilden dann die Regionen des Bildes.

Im Weiteren wird vorausgesetzt, dass die zu segmentierenden Objekte sich in den Bildern deutlich vom Hintergrund abheben: z.B. helle Objekte auf dunklem Hintergrund. In diesem Fall kann eine flächenorientierte Segmentierung wie folgt besonders einfach durchgeführt werden:

- zunächst wird mit Hilfe einer Binarisierung zwischen den Objekt- und den Hintergrundpixeln unterschieden (siehe Kapitel 8.2.1),
- anschließend werden die Objektpixel zu Zusammenhangskomponenten zusammengefasst, die dann den einzelnen Objekten entsprechen; (siehe Kapitel 8.2.2).

8.2.1 Binarisierung

Die ***Binarisierung*** stellt einen Spezialfall der stückweise linearen Grauwert-Transformation (siehe Kapitel 8.1.1.1) bzw. eine Verallgemeinerung der Äquidensitenbildung (siehe Kapitel 8.1.1.3) dar. Bei einer Binarisierung wird ein Eingabebild B mit Grauwerten g(i,j) mit Hilfe eines Schwellwertes `schwelle` in ein binäres Ausgabebild (***Binärbild***) B‘ mit Grauwerten g‘(i,j) überführt, wobei g‘(i,j) nur die beiden Werte g_1‘ bzw. g_2‘ annehmen kann:

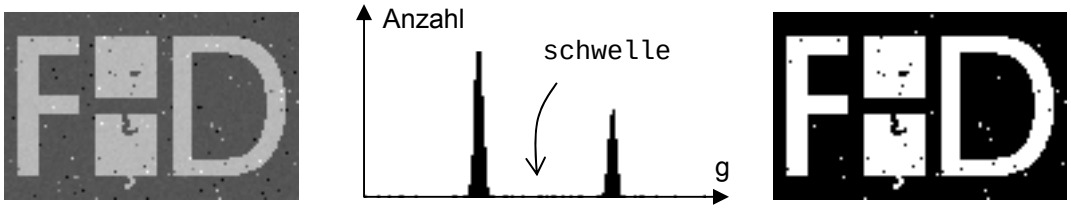
(14-1)

$$g'(i,j) = \begin{cases} g_1' & \text{falls } g(i,j) \le \texttt{schwelle} \\ g_2' & \text{falls } g(i,j) > \texttt{schwelle} \end{cases}$$

(Häufig wird g_1' als Schwarz und g_2' als Weiß gewählt).

Bei günstiger Wahl des Schwellwertes `schwelle` haben nach der Binarisierung alle hellen Objektpixel den Wert g_2‘ und alle dunkeln Hintergrundpixel den Wert g_1‘. Ein geeigneter Schwellwert kann z.B. mit Hilfe des Histogrammes ermittelt werden.

Abbildung 8.8 zeigt links ein Bild mit mehreren hellen Objekten. In der Mitte ist das zugehörige Histogramm dargestellt. Bei derartigen ***bimodalen Histogrammen*** liegt ein günstiger Binarisierungs-Schwellwert zwischen den beiden „Bergen“ des Histogrammes.



(a) (b) (c)

Abb. 8.8: Beispiel für eine Binarisierung:
(a) Original-Grauwertbild B;
(b) Zugehöriges Histogramm mit Schwellwert `schwelle`;
(c) Ergebnisbild B‘ nach der Binarisierung.

Das Ergebnis der Binarisierung mit dem in der Mitte eingezeichneten Schwellwert `schwelle` ist auf der rechten Seite von Abbildung 8.8 gezeigt.

8.2.2 Markierung von Zusammenhangskomponenten (ZHK-Markierung)

Im nächsten Verarbeitungsschritt müssen die als Objektpixel gekennzeichneten Bildpunkte mit dem Grauwert g_2' (heller Vordergrund) zusammengefasst werden.

Eine zusammenhängende Menge von Bildpunkten, bei der sich jeder Punkt mit jedem anderen Punkt der Menge durch einen Pfad verbinden lässt, der nur aus benachbarten Bildpunkten der Menge besteht, wird als ***Zusammenhangskomponente (ZHK)*** bezeichnet. Entsprechend der beiden Nachbarschaftsbeziehungen (siehe Kapitel 8.1.2) unterscheidet man zwischen N4- und N8-Zusammenhangskomponenten.

Zur Ermittlung der Zusammenhangskomponenten kann z.B. ein ***rekursives Markierungsverfahren*** eingesetzt werden, das das Binärbild B‘ in ein Ergebnisbild E mit unterschiedlich markierten Zusammenhangskomponenten überführt.

Die Grundidee dieses Verfahrens ist, dass das Bild zeilenweise abgearbeitet wird, bis das erste Objekt- bzw. Vordergrundpixel erreicht wird. Von diesem ausgehend wird die zugehörige ZHK rekursiv „geflutet“, d.h. jedes Pixel erhält dieselbe Marke. Danach wird die zeilenweise Vorgehensweise „neben“ dem oben gefundenen ersten Vordergrundpixel fortgesetzt, bis wieder ein Vordergrundpixel gefunden wird, das noch nicht markiert ist. Die zugehörige ZHK wird – unter Verwendung einer neuen Marke – wieder geflutet, usw..

Das unten gezeigte Struktogramm demonstriert den genauen Ablauf des rekursiven Markierungsverfahrens. Dabei habe der Hintergrund des Eingabe-Binärbildes den Grauwert 0 und der Vordergrund den Grauwert 255; es können hier also maximal 253 verschiedene ZHKs markiert werden. Ein Beispiel für eine N8-ZHK-Markierung ist in Abbildung 8.9 zu sehen.

Rekursives Fluten ()

```
marke = 0;
Für alle Zeilen j
  Für alle Spalten i
    g(i,j) = 255 ?
    Ja:  marke++;
         g(i,j) = marke;
         pixel_anlagern(i,j,marke);
    Nein: %
```

pixel_anlagern()

```
Für alle Positionen (k,l)
der N4 oder N8 von (i,j)
  g(k,l) = 255 ?
  Ja:  g(k,l) = marke;
       pixel_anlagern(k,l,marke);
  Nein: %
```

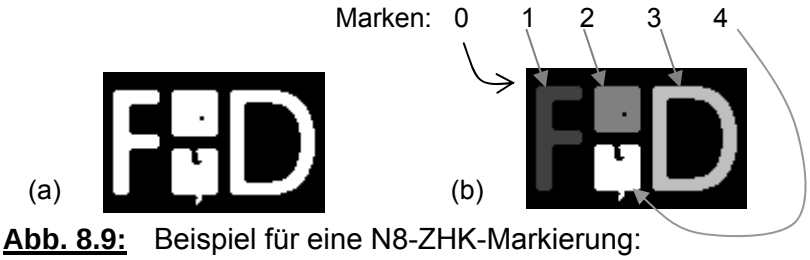


Abb. 8.9: Beispiel für eine N8-ZHK-Markierung:
(a) Binärbild B‘ aus Abbildung 8.8 (c); nach Median-Elimination kleiner Störungen;
(b) Ergebnisbild E der N8-ZHK-Markierung für die Objektpixel mit den endgültigen Marken 0 (Hintergrund) und 1 bis 4 für die vier im Bild enthaltenen Objekte.

8.3 Geometrische Bildmanipulationen

Unter ***geometrischen Bildmanipulationen*** versteht man Verarbeitungsschritte, die die geometrischen Eigenschaften eines Bildes ändern: z.B. Größenänderung, Spiegelung, Drehung, Scherung oder perspektivische Projektion eines Bildes.

Auf diesen globalen Transformationen, die ein komplettes Bild nach einer einheitlichen Vorschrift verändern, bauen weitergehende geometrische Transformationen, wie z.B. Warping und Morphing auf, die unterschiedliche Teile eines Bildes unterschiedlich verändern können. Beim ***Warping*** kann ein Bild beliebig verzerrt werden: so können z.B. in einem Gesichtsbild gezielt die Mundwinkel angehoben werden, um ein Lächeln zu bewirken. Beim ***Morphing*** werden Bilder geometrisch und photometrisch verändert, um durch eine zeitliche Interpolation einen fließenden Übergang zwischen zwei Bildern zu erreichen: z.B. kontinuierliche Verwandlung einer Person in eine andere.

Im Folgenden wird nur auf die Größenänderung (Skalierung) eines Bildes eingegangen, da diese in der Praxis besonders häufig auftritt: z.B. Anpassung der Größe eines Bildes, damit es auf einer Text- oder Bildschirmseite eine bestimmte Fläche belegt.

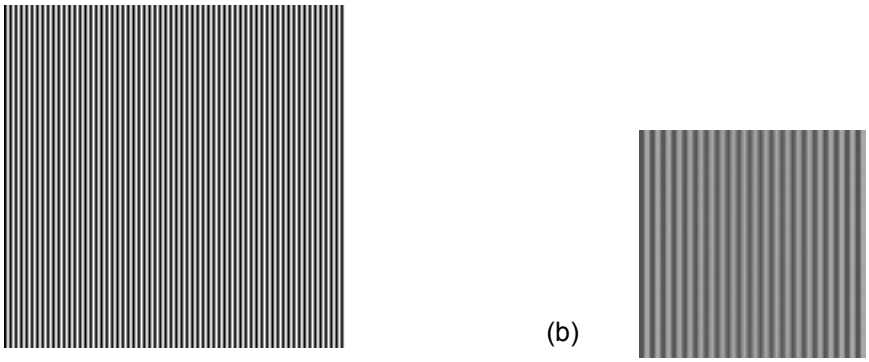
Besonders einfach sind ***ganzzahlige Vergrößerungen*** und ***Verkleinerungen***, da sie durch Vervielfachen bzw. Zusammenfassen von Pixeln erreicht werden können. Die Abbildungen 8.10 (b) und (c) zeigen zwei Beispiele für Verkleinerungen des Originalbildes aus Abbildung 8.10 (a) um den Faktor 4 bzw. 8 mit anschließender Rückvergrößerung auf die Originalgröße. Derartige Verkleinerungen (ohne Rückvergrößerung) stellen eine Bilddatenreduktion dar, die aber mit einem erheblichen Verlust an Detailinformation verbunden ist.

(a) (b) (c)

Abb. 8.10: Beispiele für ganzzahlige Verkleinerungen eines Bildes mit anschließender Rückvergrößerung:
(a) Originalbild;
(b) Verkleinerung in x- und y-Richtung um den Faktor 4;
(c) Verkleinerung in x- und y-Richtung um den Faktor 8.

Bei ganzzahligen Vergrößerungen ohne Interpolation ist zu berücksichtigen, dass störende Pixelblöcke entstehen können; siehe Abbildungen 8.10. (b) und (c). Wenn diese Blöcke durch anschließendes Weichzeichnen verschliffen werden, so ergibt sich ein Verlust an Bildinformation. Derselbe Effekt (der einer Tiefpass-Filterung entspricht) ergibt sich bei der Verkleinerung, da auch hier feine Bilddetails und scharfe Kanten durch die Zusammenfassung von Pixeln verloren gehen können.

Bei Verkleinerungen können Effekte entstehen, wie sie auch bei der Abtastung einer Vorlage mit einer zu grob gewählten Auflösung entstehen (Unterabtastung). Abbildung 8.11 (b) zeigt als Beispiel die Verkleinerung der Vorlage aus Abbildung 8.11 (a) auf zwei Drittel der Länge und Breite. Das Originalbild enthält ein-pixel-breite senkrecht verlaufende schwarze bzw. weiße Linien, die in der Verkleinerung zu erheblich weniger hell- und dunkelgrauen Streifen zusammengefasst werden.



(a) (b)

Abb. 8.11: Beispiel für eine Verkleinerung (b) einer Vorlage (a).

Eine besonders effiziente Möglichkeit zur Bildskalierung geht auf Fant[1] zurück;. Bei diesem Ansatz wird zuerst jede Bildzeile auf die gewünschte Länge skaliert. Dazu wird aus der Eingangsbildbreite und der Zielbildbreite der Skalierungsfaktor berechnet. Nun laufen zwei Indizes an der Eingangsbild- und der Zielbildzeile entlang (siehe waagrechte Pfeile in Abbildung 8.12), wobei der Skalierungsfaktor die Bewegungsgeschwindigkeiten der Indizes

[1] K. Fant; IEEE CG&A 1/86; Patent Honeywell

bestimmt. Die Pixel der längeren Zeile werden in einem Akkumulator gesammelt. Jedes Mal, wenn der Index der kürzeren Zeile ein Pixel vorgerückt ist, wird aus den Pixeln des Akkumulators ein Mittelwert gebildet, und dieser wird als Ausgabe-Grauwertwert in die Ausgabezeile eingesetzt. „Angeschnittene“ Pixel der längeren Zeile werden anteilig berücksichtigt.

Abschließend wird derselbe Vorgang spaltenweise (auf das bereits zeilenweise interpolierte Bild) angewendet.

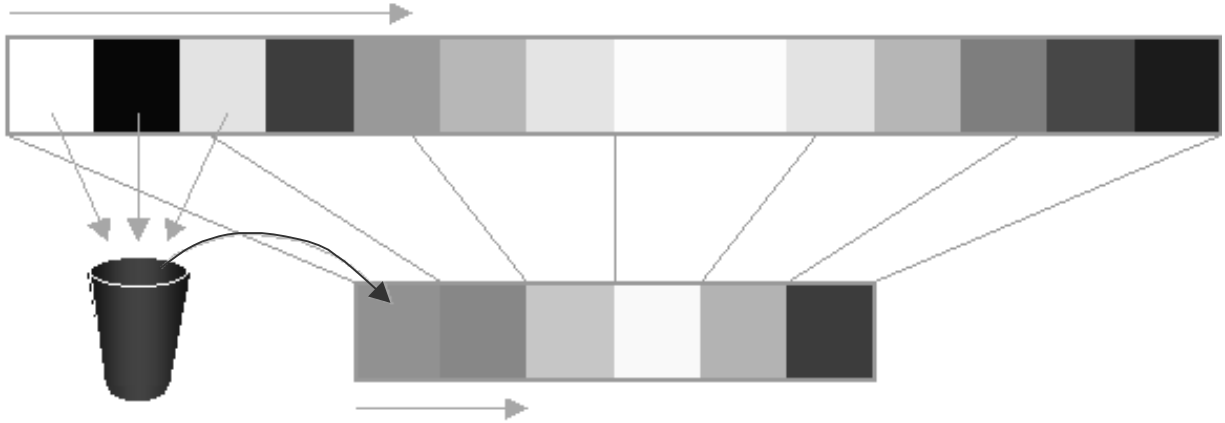


Abb. 8.12: Prinzip der Skalierungsmethode nach Fant; Abbildung aus [2].

Die Methode von Fant wird z.B. bei der Windows-Bitmap-Skalierung und bei vielen anderen Bildbearbeitungs-Tools eingesetzt.

Für beliebige geometrische Bildtransformationen kann man die sogenannten ***indirekten Verfahren*** einsetzen. Bei diesen wird mit Hilfe der inversen Transformationsvorschrift zu jedem Pixel des Ausgabebildes die entsprechende Urbildstelle im Eingabebild berechnet; siehe Abbildung 8.13. Falls diese Stelle nicht exakt im Pixelraster liegt (siehe schwarzer Punkt in Abb. 8.13 (a)), so kann an der Zielposition der Grauwert des nächst liegenden Pixels aus dem Eingabebild eingesetzt werden (Interpolation 0. Ordnung: nearest neighbor resampling; siehe Pfeil von rechts oben). Statt den Grauwert des nächstliegenden Pixels einzusetzen, kann der Zielgrauwert auch aus den Grauwerten der im Eingabebild benachbarten Bildpunkte höherwertig interpoliert werden;(z.B. Interpolation 1. Ordnung: bilineare Interpolation mit Hilfe der vier benachbarten Bildpunkte; siehe alle vier kurzen Pfeile in Abbildung 8.13 (a)).

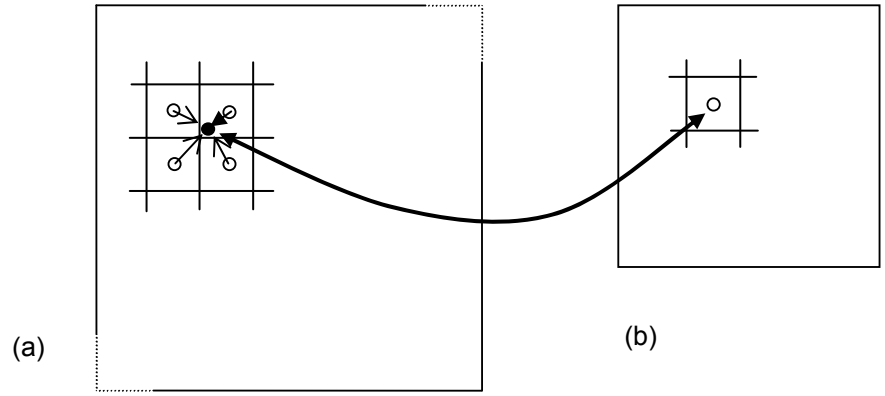


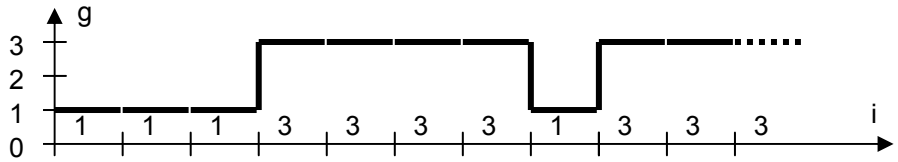
Abb. 8.13: Prinzip der indirekten Methode zur geometrischen Bildtransformation:
(a) Eingabebild, das verkleinert werden soll; (die nicht gefüllten Kreise stehen stellvertretend für die Mittelpunkte von vier benachbarten Pixeln);
(b) verkleinertes Ausgabebild.

Schlussbemerkung: Grundsätzlich gilt, dass bei (fast) jeder geometrischen Bildtransformation Bildinformation verloren geht. Hieraus folgt für den praktischen Einsatz, dass man ein Bild nie mehrmals hintereinander transformieren (z.B. skalieren) soll. Stattdessen sollte man bei jedem neuen Versuch wieder auf das Originalbild zurückgreifen.

[2] B. Kreling; Skript zur Vorlesung „Multimedia“

Anhang: 1D-Beispiele zu Rangfolge-Operatoren

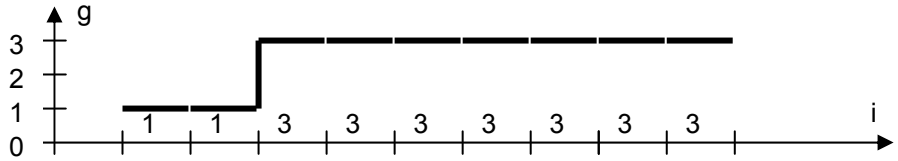
Original-Grauwerte



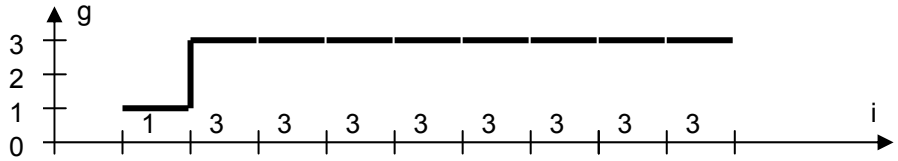
g(i-1), g(i), g(i+1) aufsteigend sortiert:

--- 111 113 133 333 333 133 133 133 333 ---

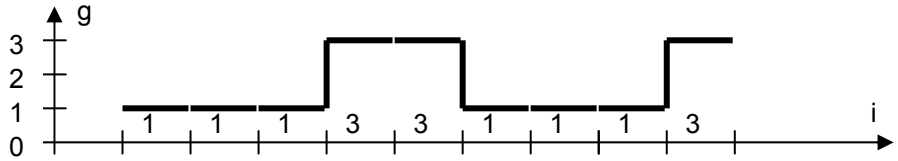
1.) mittleres Element wählen: Median-Operator



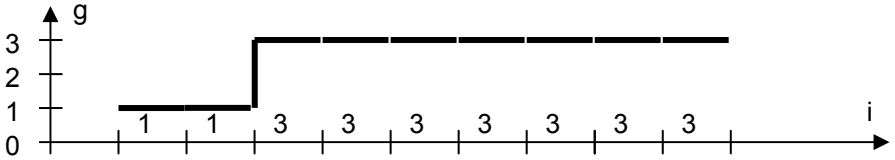
2.) maximales Element wählen: Dilatation



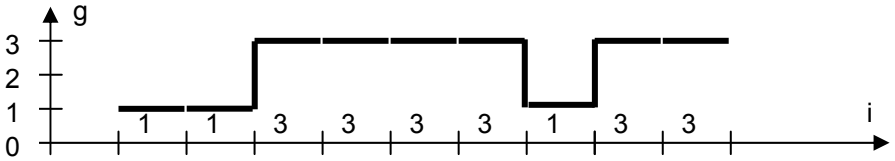
3.) minimales Element wählen: Erosion



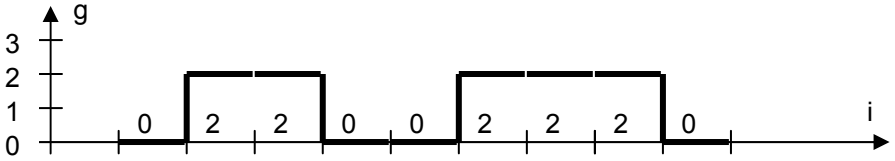
4.) Dilatation gefolgt von Erosion: Closing



5.) Erosion gefolgt von Dilatation: Opening



6.) maximales minus minimales Element: Spannweite



Bemerkungen:

1.) Der Median-Operator bewirkt eine kantenerhaltende Glättung.
2.) Die Dilatation dehnt helle Bereiche in alle Richtungen um ein Pixel aus.
3.) Die Erosion reduziert helle Bereiche in alle Richtungen um ein Pixel.
4.) Closing schließt dunkle Löcher
5.) Opening löscht dunkle Ausläufer und schließt helle Löcher; (hier keine Auswirkungen).
6.) Der Spannweiten-Operator ist ein Kantendetektor

9 Bildkompression und Dateiformate

Digitale Einzelbilder und insbesondere Bildsequenzen (z.B. Filme) benötigen in der Regel sehr viel Speicherplatz und entsprechend lange Übertragungszeiten (z.B. im Internet). In den folgenden Teilkapiteln werden die gebräuchlichsten Ansätze zur Reduzierung des erforderlichen Speicherbedarfs bzw. der Übertragungszeiten vorgestellt.

Bei Rasterbilddaten geschieht dies (häufig unter Inkaufnahme von Qualitätsverlusten) durch spezielle Codierungen des Bildinhaltes, die eine Kompression der Bilddaten bewirken; siehe Kapitel 9.1. Bei Bildern, die ursprünglich nicht gerastert vorliegen und die z.B. mit Hilfe eines Zeichen- oder Konstruktionsprogramms erstellt wurden, speichert man statt des gerasterten Bildes die vektorielle Beschreibung des Bildes ab; siehe Kapitel 9.3. In Kapitel 9.2 werden außerdem die wichtigsten Rasterbild-Dateiformate vorgestellt; die gängigsten bildbeschreibenden Dateiformate werden in Kapitel 9.3 behandelt.

9.1 Kompressionstechniken für Rasterbilddaten

Das wichtigste Unterscheidungsmerkmal von Rasterbild-Kompressionsverfahren ist, ob sie eine verlustfreie oder verlustbehaftete Rekonstruktion der Bilddaten zulassen. Während bei den Verfahren mit ***verlustfreier Rekonstruktion*** das rekonstruierte Bild exakt mit dem Originalbild übereinstimmt, verzichtet man bei den Verfahren mit ***verlustbehafteter Rekonstruktion*** auf "unwichtige" Details. Dabei kann man i.d.R. die gewünschte Rekonstruktions-Qualität wählen und damit die Größe der komprimierten Bilddatei beeinflussen.

Verlustfreie Kompressionsverfahren arbeiten mit ***Redundanz-Reduktion*** was auch als ***Entropie-Codierung*** bezeichnet wird. Unter Redundanz versteht man dabei, dass Informationen aufgrund räumlicher oder zeitlicher Ähnlichkeiten in den Daten mehrfach vorhanden sind; (z.B. homogene Bildbereiche in Einzelbildern bzw. ähnliche aufeinanderfolgende Bilder einer Bildfolge). Verlustbehaftete Kompressionsverfahren verwenden meist auch Redundanz-Reduktion, hauptsächlich setzen sie aber ***Irrelevanz-Reduktion*** ein. Irrelevanz bezieht sich dabei auf Informationsanteile, die bei der Dekompression hinsichtlich der Rekonstruktionsgüte unerheblich sind; (z.B. sehr feine und kaum wahrnehmbare Bilddetails).

Weiterhin unterteilt man die Codierungsverfahren in ***eindimensionale*** und ***zweidimensionale***, meist ***blockorientierte Verfahren***. Eindimensionale Verfahren können nur eindimensionale Daten (z.B. Strings) verarbeiten, so dass Bilddaten zuerst in eine geeignete eindimensionale Anordnung überführt werden müssen. Hierzu kann man z.B. die Zeilen eines Bildes linear hintereinander anordnen. Bei den blockorientierten Verfahren werden die zweidimensionalen Eingangsdaten in eine gitterartige Anordnung rechteckiger Blöcke zerlegt, von denen jeder isoliert betrachtet wird. Daneben gibt es noch die ***globalen Verfahren***, bei denen das Bild als Ganzes betrachtet und komprimiert wird.

Im Vergleich der für die Codierung bzw. Decodierung erforderlichen Rechenzeiten unterscheidet man zwischen ***symmetrischen*** und ***asymmetrischen Verfahren***. Bei den symmetrischen Methoden benötigt die Codierung etwa genau soviel Zeit, wie die Decodierung. Im Gegensatz dazu dauert die Codierung bei den asymmetrische Methoden ein Vielfaches länger als die Decodierung.

Im Hinblick auf die Decodierungsansätze wird zwischen ***sequentieller*** und ***progressiver Decodierung*** unterschieden. Bei der sequentiellen Decodierung wird ein Bild Zeile für Zeile auf dem Bildschirm aufgebaut. Im Gegensatz dazu zeigt die progressive Decodierung gleich zu Beginn eine unscharfe bzw. vergröberte Kopie des ganzen Bildes an, die dann schrittweise verbessert wird. Auf diese Weise kann man z.B. bei Web-Anwendungen die Übertragung eines Bildes frühzeitig abbrechen und so teure Übertragungszeit und Speicherplatz einsparen, wenn es sich nicht um das gewünschte Bild handelt. Wenn ein Bild in mehreren Auflösungsstufen verfügbar ist, so spricht man von ***hierarchischer*** oder ***skalierbarer Codierung***.

Im Folgenden werden die drei für Bilddaten am häufigsten eingesetzten Kompressionstechniken kurz erläutert: Lauflängencodierung, Huffman Codierung und LZW-Codierung.

9.1.1 Lauflängencodierung (Run Length Encoding RLE)

Die ***Lauflängencodierung (Run Length Encoding, RLE)*** stellt eine eindimensionale Kompressionstechnik dar, bei der die in homogenen Bildbereichen enthaltenen Redundanzen ausgenutzt werden; (räumliche Kohärenz).

Bei der Lauflängencodierung wird in der eindimensionalen Anordnung der Farbwerte des Bildes (siehe oben) jedes Intervall identischer Werte dadurch ersetzt, dass man den Wert selbst und die Häufigkeit seines Auftretens in diesem Intervall angibt.

Da bei der Lauflängencodierung keine Bildinformationen unterdrückt werden, ist eine verlustfreie Rekonstruktion der Bilder möglich. Die Lauflängencodierung, die zu den symmetrischen Ansätzen zählt, empfiehlt sich nur, wenn größere homogen eingefärbte Bildbereiche vorhanden sind.

9.1.2 LZW-Codierung

Die ***LZW-Codierung*** nach **L**empel, **Z**iv und **W**elch ist eine eindimensionale Kompressionstechnik, die davon ausgeht, dass die eindimensionale Anordnung der Farbwerte Abschnitte enthält, die an vielen verschiedenen Bildstellen immer wieder unverändert auftreten. An allen Stellen, an denen ein bereits bekannter Abschnitt auftritt, wird statt dieses Abschnitts nur ein Rückverweis auf das erste Auftreten dieses Abschnittes abgespeichert.

Die LZW-Codierung stellt eine musterorientierte Kompressionstechnik dar, die eine verlustfreie Rekonstruktion ermöglicht. Mit der LZW-Codierung, die zu den redundanzreduzierenden, symmetrischen Verfahren zählt, kann man für ein Bild nur dann eine erfolgreiche, d.h. platzsparende Codierung durchführen, wenn das Bild große homogene Bereiche oder sich häufig wiederholende Muster, wie z.B. Texturen, besitzt.

9.1.3 Huffman-Codierung

Der auf Huffman zurückgehenden ***Huffman-Codierung***, die ebenfalls ein eindimensionales Codierungsverfahren ist, liegt dieselbe Idee wie dem Morsealphabet zugrunde. Zunächst wird für alle auftretenden Farbwerte die Häufigkeit ihres Auftretens bestimmt. Anschließend wird ein Code variabler Länge verwendet, um häufig auftretende Farbwerte mit einem sehr kurzen Code, seltener auftretende Farbwerte dagegen mit einem längeren Code zu repräsentieren.

Die Huffman-Codierung gehört zu den verlustfreien Kompressionsverfahren. Sie empfiehlt sich nur, wenn die Häufigkeitsverteilung der zu codierenden Werte deutlich von einer Gleichverteilung abweicht. Bei realen Bilddaten ist der Kompressionsfaktor i.d.R. niedrig. Die Huffman-Codierung wird daher in der Praxis erst dann auf Bilddaten angewandt, wenn diese zuvor einer speziellen Transformation unterzogen wurden, die eine klare Ungleichverteilung der Werte bewirkt; (z.B. Diskrete Cosinus-Transformation bei der JPEG-Kompression; siehe Kapitel 9.1.5).

9.1.4 Fraktale Bildkompression

Die ***fraktale Bildkompression*** wurde im Rahmen der fraktalen Geometrie entwickelt, die ihrerseits ein Teilgebiet der Chaostheorie ist.

Das Grundprinzip der fraktalen Kompression basiert auf der sogenannten ***Selbstähnlichkeit***, die in der Natur genauso wie in realen fotografischen Bildern überall zu beobachten ist. Der Begriff „Selbstähnlichkeit“ sagt hier aus, dass viele Naturphänomene im Großen wie im Kleinen dieselben Charakteristika aufweisen: die Küstenlinie eines ganzen Kontinentes zeigt im Satellitenbild dieselbe Struktur von Ein- und Ausbuchtungen, wie ein kleiner Küstenabschnitt von wenigen Kilometern oder Metern Länge; die Hauptäste eines Laubbaumes sehen wie verkleinerte Kopien des ganzen Baumes aus und dasselbe gilt auch für die kleineren Äste des Baumes.

Diese Selbstähnlichkeit ist die Grundidee des sogenannten ***Collage-Theorems*** [Barn], demzufolge sich eine natürliche Figur aus geeignet geometrisch transformierten (d.h. verkleinerten, gedrehten und verschobenen) Kopien der Figur selbst zusammensetzen läßt.

Mit Hilfe des Collage-Theorems kann man in der Computer Graphik mit geringem Konstruktionsaufwand und minimalem Speicherbedarf realistisch wirkende Objekte und Objektoberflächen wie z.B. Pflanzen, Planeten oder Landschaftsausschnitte erzeugen; beispielsweise lässt sich ein realistisch aussehendes Schwarzweißbild eines Farns, das obendrein beliebig vergrößerbar ist, mit Hilfe von 24 leicht zu erzeugenden reellen Zahlen erstellen.

Das Collage-Theorem lässt sich aber nicht nur zur Erzeugung synthetischer Bilder, sondern auch zur effizienten Bildkompression einsetzen. So wird bei der fraktalen Kompression eines Bildes nach Bereichen gesucht, die an anderen Stellen des Bildes nochmals in geeignet geometrisch transformierter Form auftreten; (z.B. Verkleinerungen, Scherungen und Rotationen). Auch photometrische Änderungen (wie Farb-, Helligkeits- oder Kontraständerungen) sind hierbei zugelassen.

Wenn man das ganze Bild mit derartigen Bereichen abdeckt, so genügt es, sich die Parameter der jeweils verwendeten geometrischen und photometrischen Transformationen zu merken, um das Bild zu rekonstruieren.

Da in der Regel einige tausend Transformationen ausreichen und diese durch wenige Parameter beschrieben werden, erreichen fraktale Kompressionsverfahren oft extreme Kompressionsraten mit zum Teil weniger als fünf Prozent des ursprünglichen Speicherbedarfs.

Die fraktale Kompression, die mit Redundanz- und Irrelevanz-Reduktion arbeitet, gehört zu den verlustbehafteten Verfahren, jedoch ist der Qualitätsverlust meist relativ gering. Während sehr feine und gering kontrastierende Bilddetails je nach Einstellung oft ungenau rekonstruiert werden, werden starke Kontrastsprünge im Gegensatz zu anderen verlustbehafteten Verfahren weder abgeschwächt noch verwischt. Ein weiterer Vorteil ist, dass ein fraktal komprimiertes Bild in beliebiger Auflösung rekonstruiert werden kann, und dass Bildausschnitte beliebig vergrößerbar sind, ohne dass blockartige Pixelstrukturen auftreten.

Die fraktale Kompression, die zu den zweidimensionalen Kompressionsverfahren gehört, ist aufgrund der Vielzahl zu untersuchender Transformationen sehr rechenintensiv. Im Gegensatz dazu konvergiert die Dekompression, bei der alle ermittelten Transformationen iterativ immer wieder angewendet werden, i.d.R. in wenigen Schritten und mit geringem Rechenaufwand, so dass es sich um ein asymmetrisches Verfahren handelt. Die Dekompression erfolgt häufig progressiv.

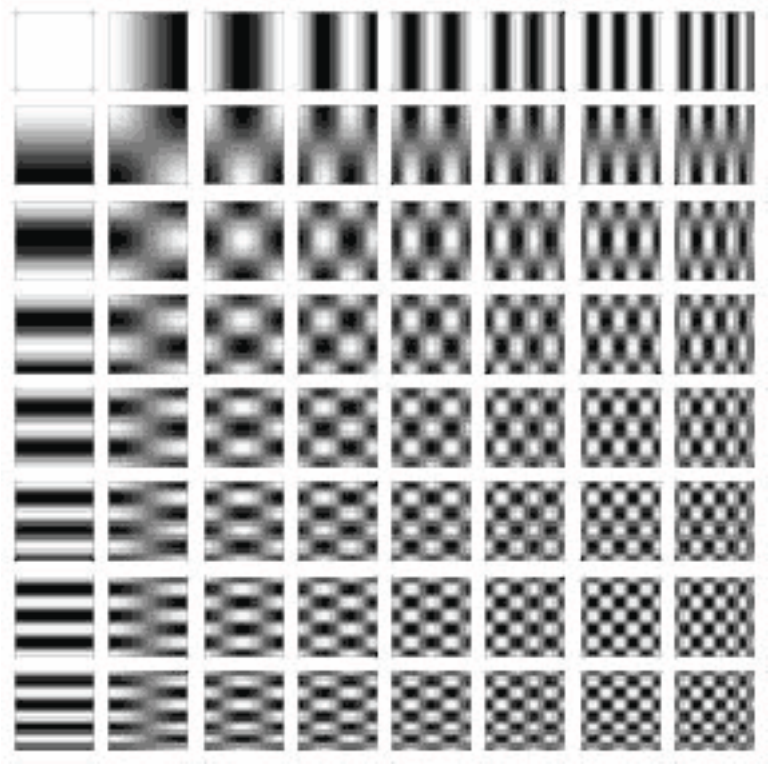
9.1.5 Diskrete Cosinus-Transformation mit Frequenzfilterung

Die ***Diskrete Cosinus-Transformation*** (DCT) ist eine bei der Kompression zwei- und dreidimensionaler Daten (z.B. konventionelle 2D-Bilder und 3D-Volumenbilder) sehr häufig eingesetzte Transformation. Mit Hilfe der DCT, die eng mit der Diskreten Fourier-Transformation verwandt ist, werden die Eingangsdaten vom Orts- in den Ortsfrequenzbereich transformiert. Die eigentliche Kompression erfolgt im Ortsfrequenzbereich. Dabei behält man die besonders informationsträchtigen niedrigen Frequenzanteile bei, reduziert aber die hohen Frequenzen, die nicht oder kaum wahrnehmbaren Bilddetails entsprechen; (da das Auge bei hohen Frequenzen nicht so empfindlich auf Frequenzunterschiede reagiert, können diese gröber quantisiert werden). Durch eine Invertierung der Quantisierung und der inversen Diskreten Cosinus-Transformation können die Ausgangsdaten mit geringen Verlusten wieder rekonstruiert werden.

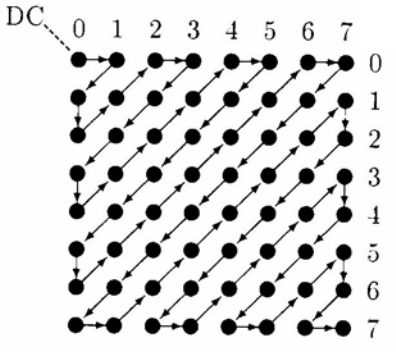
Bei der DCT wird eine Bildmatrix zunächst in Blöcke von je 8*8-Pixel zerlegt, die danach einzeln transformiert werden. Die Grundidee der Transformation besteht darin, dass man jeden Block als gewichtete Summe der in Abbildung 9.1 (a) gezeigten 64 Basisfunktionen darstellt. Dies entspricht einer Überlagerung der passenden Basisfunktionen mit entsprechend gewählten Amplituden. Die Matrix dieser 8*8 reellwertigen Amplituden (inkl. Vorzeichen) wird als Koeffizientenmatrix C(u,v) bezeichnet und stellt das Ergebnis der DCT dar:

$$C(u,v) = \frac{2}{N} * c(u) * c(v) * \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} \left(g(i,j) * \cos\left(\frac{(2i+1)u\pi}{2N} \right) * \cos\left(\frac{(2j+1)v\pi}{2N} \right) \right) \quad \text{Hier : } N = 8$$

Dabei steht g(i,j) für den Grauwert in Spalte i und Zeile j des jeweils zu transformierenden 8*8-Blockes, C(u,v) ist der resultierende Koeffizient (Amplitude) der Basisfunktion in der u. Spalte und v. Zeile aus Abbildung 9.1(a). Die Faktoren c(u) und c(v) haben den Wert $1/\sqrt{2}$, falls u bzw. v Null sind; ansonsten gilt c(u)=1 bzw. c(v)=1.



(a)



(b)

Abb. 9.1: (a) Basisfunktionen der Diskreten Cosinus-Transformation.
(b) Zick-Zack-Kurs zur eindimensionalen Abarbeitung eines Bildblockes.

Das eigentliche Vorgehen der DCT wird mit Hilfe des folgenden Beispiels demonstriert; ([Reim], Seite 62).

Beispiel:

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

(a)

1259,6	– 1.0	– 12.1	– 5.2	2.1	– 1.7	– 2.7	– 1.3
– 22.6	– 17.5	– 6.2	– 3.2	– 2.9	– 0.1	0.4	– 1.2
– 10.9	– 9.3	– 1.6	1.5	0.2	– 0.9	– 0.6	– 0.1
– 7.1	– 1.9	0.2	1.5	0.9	– 0.1	0.0	0.3
– 0.6	– 0.8	1.5	1.6	– 0.1	– 0.7	0.6	1.3
– 1.8	– 0.2	1.6	– 0.3	– 0.8	1.5	1.0	– 1.0
– 1.3	– 0.4	– 0.3	– 1.5	– 0.5	1.7	1.1	– 0.8
– 2.6	1.6	– 3.8	– 1.8	1.9	1.2	– 0.6	– 0.4

(b)

Abb. 9.2: Beispiel zur Diskreten Cosinus-Transformation
(a) 8*8-Block (Bildausschnitt) (b) DCT-Koeffizienten

Das Beispiel zeigt die typische Wirkungsweise der DCT an einem 8*8 Pixel großen Bildausschnitt (Abbildung 9.2 (a)): nach der DCT sind in der Koeffizientenmatrix (Abbildung 9.2 (b)) nur noch wenige betragsmäßig große Werte vorhanden, und diese stehen vorwiegend in der linken, oberen Ecke, die den niedrigen Frequenzanteilen des Bildes entspricht. Der Wert in der linken oberen Ecke, der dem konstant eingefärbten Feld in Abbildung 9.1 (a) entspricht,

wird als Gleichspannungs- bzw. DC-Koeffizient bezeichnet. Alle anderen Koeffizienten sind die Wechselspannungs.- bzw. AC-Koeffizienten.

Im nächsten Schritt erfolgt nun mittels Frequenzfilterung die eigentliche Datenreduktion, indem alle Koeffizienten mit Hilfe der sogenannten Normalisierungs- oder Quantisierungstabellen „gedämpft“ werden; (***Quantisierung***). In der unten dargestellten Fortsetzung des Beispiels ist eine der Quantisierungstabellen gezeigt (Abbildung 9.3 (a)), wie sie vom JPEG-Komitee für Grauwertbilder bzw. für die Luminanzkomponente von Farbbildern empfohlen wird. Jeder DCT-Koeffizient wird durch den an derselben Position stehenden Wert der Quantisierungstabelle dividiert und das Ergebnis wird zum normalisierten DCT-Koeffizienten gerundet.

Fortsetzung des Beispiels:

(a)

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
25	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

(b)

79	0	- 1	0	0	0	0	0
- 2	- 1	0	0	0	0	0	0
- 1	- 1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(c)

1264	0	- 10	0	0	0	0	0
- 24	- 12	0	0	0	0	0	0
- 14	- 13	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(d)

144	146	149	152	154	156	156	156
148	150	152	154	156	156	156	156
155	156	157	158	158	157	156	155
160	161	161	162	161	160	157	155
163	163	164	163	162	160	158	156
163	164	164	164	162	160	158	157
160	161	162	162	162	161	159	158
158	159	161	161	162	161	159	158

Abb. 9.3: Beispiel zur Diskreten Cosinus-Transformation (Fortsetzung)
(a) Quantisierungstabelle (b) normalisierte DCT-Koeffizienten
(c) rekonstruierte DCT-Koeffizienten (d) rekonstruierter Bildausschnitt

Das Beispiel zeigt in Abbildung 9.3 (b), dass in der Matrix der normalisierten DCT-Koeffizienten nur noch wenige Werte im linken oberen Bereich, der den niedrigen Frequenzanteilen entspricht, von Null verschieden sind.

Im nächsten Schritt wird für die AC-Koeffizienten dieser Matrix eine Lauflängencodierung vorgenommen, wobei wegen der speziellen Belegung der Matrix der Zick-Zack-Kurs aus Abbildung 9.1 (b) zur Umwandlung der 2D-Matrix in eine 1D-Zeichenkette verwendet wird. Für den DC-Koeffizienten wird lediglich die Differenz zum DC-Koeffizienten des vorangegangenen 8*8-Blockes abgespeichert. In einem letzten Schritt werden die DC-Differenz und das Ergebnis der Lauflängencodierung durch die Verwendung einer Huffman-Codierung nochmals (verlustfrei) komprimiert.

Auf diese Weise kann der im Beispiel gezeigte 8*8-Bildblock mit nur 31 Bit repräsentiert werden.

Indem alle diese Schritte in umgekehrter Reihenfolge durchlaufen werden, kann das Bild mit wenigen Verlusten rekonstruiert werden; siehe Abbildungen 9.3 (c) und (d). Die verlustfreie inverse DCT ist dabei gegeben durch:

$$g(i,j) = \frac{2}{N} * \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} \left(C(u,v) * c(u) * \cos\left(\frac{(2i+1)u\pi}{2N} \right) * c(v) * \cos\left(\frac{(2j+1)v\pi}{2N} \right) \right) \quad \text{Hier : } N = 8$$

(Die Bedeutung von C(u,v), g(i,j) sowie c(u) und c(v) ist dabei wie oben angegeben).

Bei der Diskreten Cosinus-Transformation (DCT) mit nachfolgender Frequenzfilterung, die ein sehr häufig eingesetztes verlustbehaftetes Kompressionsverfahren darstellt (z.B. JPEG- und MPEG-Codierung), handelt es sich um ein 2D-blockorientiertes symmetrisches Verfahren.

Die DCT mit nachfolgender Frequenzfilterung erreicht sehr hohe Kompressionsraten, jedoch können hochfrequente Bilddetails verloren gehen und starke Kontrastsprünge abgeschwächt oder verwischt werden. Außerdem werden u.U. bei hohen Kompressionsraten die Grenzen der 8*8-Blöcke sichtbar.

9.1.6 Wavelet-Transformation mit Quantisierung

Die ***Wavelet-Transformation*** wird im Bereich der Computer Graphik hauptsächlich zur Bildkompression eingesetzt. Gegenüber der Diskreten Cosinus-Transformation mit Frequenzfilterung erzielt die Wavelet-Transformation mit Quantisierung insbesondere bei hohen Kompressionsraten qualitativ deutlich bessere Ergebnisse.

Die Wavelet-Transformation analysiert ein Bild in unterschiedlichen Auflösungsstufen (Bildpyramide), wobei im Weiteren zur Vereinfachung davon ausgegangen wird, dass das Bild quadratisch und seine Kantenlänge eine Zweierpotenz ist. Das Grundprinzip, das iterativ immer wieder angewendet wird, ist einfach: für eine Auflösungsstufe eines Bildes wird zunächst eine zeilenweise Glättung (Tiefpass-Filterung) sowie eine Reduzierung der Bildbreite auf die Hälfte vorgenommen; siehe Abbildung 9.4 oben links. Die bei der Glättung verloren gegangenen Details werden in einem zweiten Bild erfasst, das ebenfalls auf die Hälfte der ursprünglichen Bildbreite verkleinert wird; siehe Abbildung 9.4 unten links. Da die Fehler bei der Glättung genau an den vertikalen Bildkanten auftreten, entspricht die Erzeugung des Detailbilds der Anwendung eines Kantendetektors für senkrechte Kanten (gerichtete Hochpass-Filterung). In einem weiteren Schritt werden die beiden Teilbilder nun spaltenweise geglättet und auf die halbe Höhe reduziert und die dabei entstehenden Detail-Fehler werden wieder in Detailbildern erfasst; siehe rechte Seite von Abbildung 9.4.

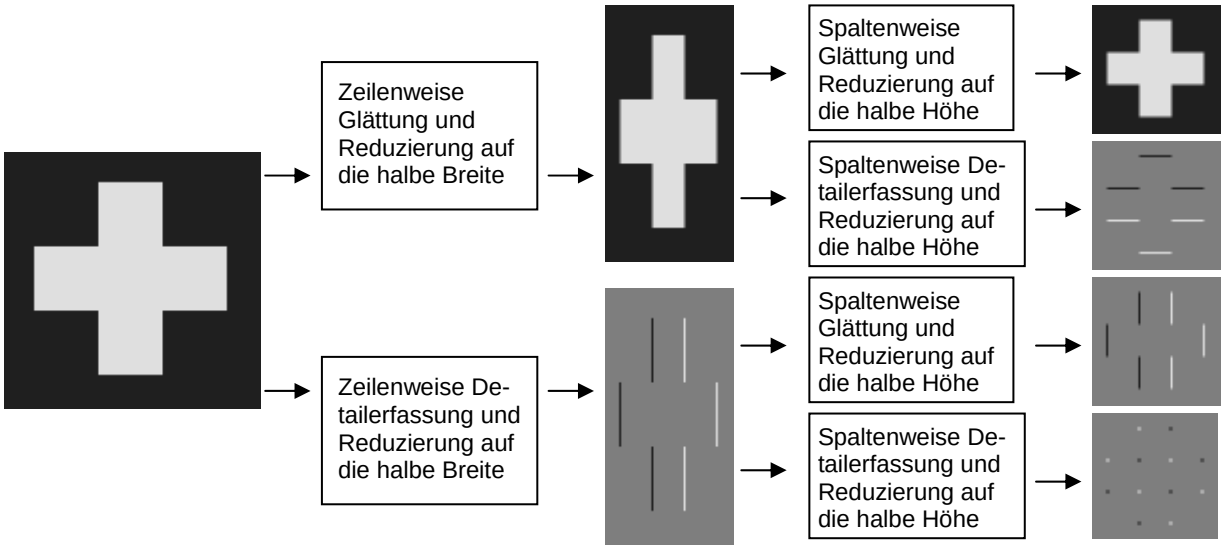
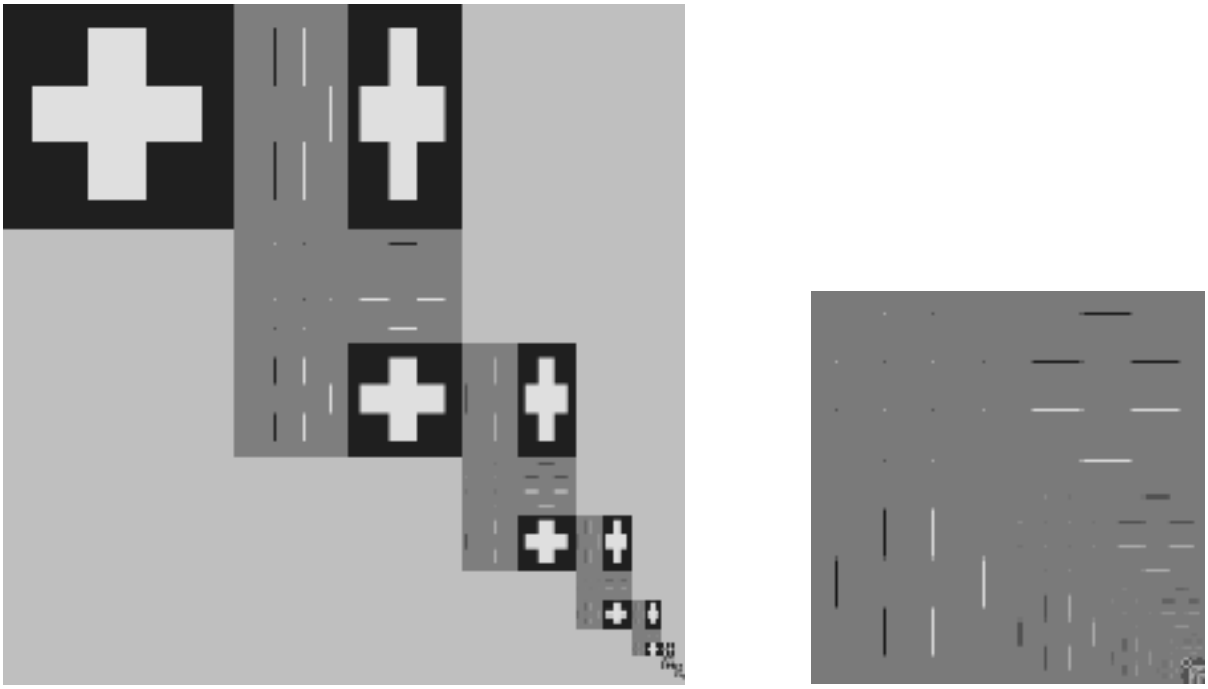


Abb. 9.4: Beispiel für einen Schritt der Wavelet-Transformation; die Farbe Mittelgrau in den Detailbildern entspricht dem Wert Null.

Die drei Detailbilder, die jeweils ein Viertel der Größe des ursprünglichen Bildes besitzen, enthalten die sogenannten ***Wavelet-Koeffizienten*** dieses Iterationsschrittes. Für das ebenfalls in beide Richtungen um den Faktor zwei verkleinerte geglättete Bild wird nun dasselbe Vorgehen aus Tief- und Hochpass-Filterung wiederholt. Bei jedem Iterationsschritt werden zunehmend gröbere Bildstrukturen herausgefiltert und das letzte, ein Pixel große Bild zeigt den mittleren Grauwert des Originalbildes. In Abbildung 9.5 (a) ist das iterative Vorgehen bei der Wavelet-Transformation vom Originalbild (oben links) bis zum letzen Filterungsschritt (unten rechts) dargestellt. Die vier Teilbilder aus Abbildung 9.4 sind oben links im diagonal unter dem Originalbild stehenden Bild zu sehen.

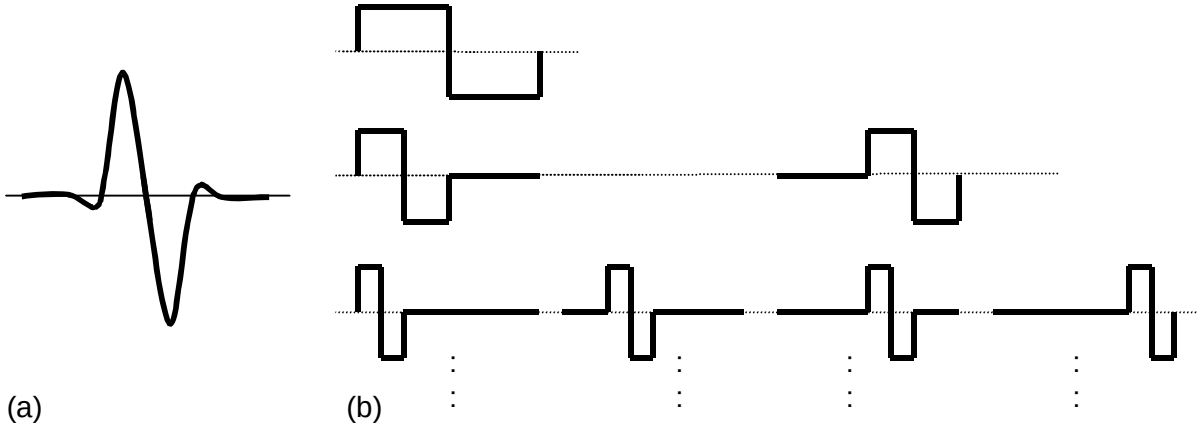


(a) (b)

Abb. 9.5: Beispiel für das iterative Vorgehen bei der Wavelet-Transformation; die Farbe Mittelgrau in den Detailbildern entspricht dem Wert Null.

Die aus allen Iterationsschritten angesammelten Wavelet-Koeffizienten bilden zusammen mit dem auf ein Pixel reduzierten Ausgangsbild das Endergebnis der Wavelet-Transformation; siehe Abbildung 9.5 (b). Diese Koeffizienten-Matrix, die die verlustfreie Rekonstruktion des Originalbildes erlaubt, hat dieselben Abmessungen, wie das Originalbild. Entscheidend ist aber, dass die meisten Koeffizienten Null oder aber betragsmäßig sehr klein sind.

Da betragsmäßig kleine Wavelet-Koeffizienten nur geringe Grauwert-Änderungen bei der Rekonstruktion des Bildes bewirken, kann man diese im Rahmen einer Irrelevanz-Reduktion unterdrücken. Dabei ist allerdings zu bedenken, dass die bei den iterativen Verkleinerungen ermittelten Koeffizienten von Schritt zu Schritt wichtiger werden, da sie einen größeren Bereich des Bildes betreffen. Die Unterdrückung der Wavelet-Koeffizienten wird durch eine spezielle ***Quantisierung*** erreicht, die unterschiedliche Bereiche der Koeffizienten-Matrix unterschiedlich grob quantisiert. Mit Hilfe dieser Quantisierung kann auch die Rekonstruktionsgüte bzw. der Speicherbedarf des komprimierten Bildes eingestellt werden.



(a) (b)

Abb. 9.6: (a) Beispiel für ein Mother-Wavelet;
(b) drei Auflösungsstufen aus der Haar-Wavelet-Basis.

Für die Tief- und Hochpass-Filterung stehen unterschiedliche Mother-Wavelets zur Verfügung, wobei man unter einem ***Wavelet*** (englisch für "kleine Welle") eine besondere Wellen-

form versteht, die (z.B. im Gegensatz zur Cosinus-Funktion) nur in einem eng begrenzten Intervall von Null verschieden ist. Abbildung 9.6 (a) zeigt ein spezielles Mother-Wavelet. Aus dem Mother-Wavelet werden iterativ verkleinerte und gegenseitig verschobene Versionen erzeugt, die in ihrer Gesamtheit als ***Wavelet-Basis*** bezeichnet werden. Abbildung 9.6 (b) zeigt einen Teil der diskreten Haar-Wavelet-Basis. Bei der iterativen Tief- und Hochpass-Filterung des Originalbildes werden von Schritt zu Schritt die nächst "größeren" Versionen der Wavelets verwendet.

Die Wavelet-Transformation mit nachfolgender irrelevanzreduzierender Quantisierung, die zu den zweidimensionalen, globalen Kompressionsverfahren gehört, erzielt Ergebnisse, die hinsichtlich der Kompressionsrate und der Bildqualität mit der fraktalen Kompression vergleichbar sind. Im Gegensatz zur fraktalen Kompression handelt es sich hier aber um ein symmetrisches Verfahren, das mit vergleichsweise geringem Rechenaufwand auskommt.

9.2 Bilddatenformate für Rasterbilder

In diesem Teilkapitel werden die gängigsten Datenformate, die zur Archivierung und zum Austausch von Rasterbilddaten verwendet werden, kurz vorgestellt. Neben den Einzelbildformaten BMP, GIF, PNG, FIF, JPEG und JPEG-2000 wird auch das Bildfolgenformat MPEG erläutert.

Zu den jeweiligen Formaten werden nur die für den Benutzer wesentlichen Aspekte (wie z.B. die Verwendung eines Kompressionsverfahrens und die Vor- und Nachteile des jeweiligen Formates) angesprochen. Auf die Schilderung des internen Aufbaus der jeweiligen Datenformate, der i.d.R. aus einem Vorspann (Header), dem eigentlichen Datenteil mit den Farbwerten und gegebenenfalls der Farbpalette, sowie u.U. weiteren Abschnitten besteht, wird hier nicht näher eingegangen. Einzelheiten können z.B. in [Born] oder [Lipp] nachgelesen werden.

9.2.1 BMP

Das Windows **B**it**m**a**p** Format BMP wurde von Microsoft anlässlich der Vorstellung von Windows 3.0 als geräteunabhängiges Datenformat für Rasterbilder eingeführt. Im BMP-Format können Bilder mit 2, 16, 256 oder ca. 16.7 Millionen Farben (True Color) verlustfrei abgespeichert werden. Dabei können Bilder mit 16 oder 256 Farben, die mit Hilfe einer Farbpalette definiert sind, nach dem RLE-Verfahren (Lauflängencodierung) komprimiert werden. Das BMP-Format ist im Vergleich zu anderen Formaten sehr einfach aufgebaut.

9.2.2 GIF

Das **G**raphics **I**nterchange **F**ormat GIF wurde von CompuServe entwickelt und erlaubt neben der reinen Abspeicherung von Bilddaten z.B. die Anzeige von Texten in Rasterbildern und das Warten auf Benutzerreaktionen. Weiterhin können in einer GIF-Datei mehrere Bilder enthalten sein („animated GIF“), so dass kurze Animationen wie z.B. animierte Icons oder aus wenigen Bildern bestehende Spots effizient abgespeichert werden können. Für umfangreiche Bildfolgen ist GIF aufgrund der trotz Kompression recht großen Datenmenge nicht geeignet.

Bei der Entwicklung des HTML-Standards (**H**yper**t**ext **M**arkup **L**anguage) wurde GIF neben dem JPEG-Format als eine der Möglichkeiten zur Einbindung von Graphik in Internetseiten festgelegt.

GIF-Bilder können, obwohl sie rechteckig definiert sind, bei der Ausgabe beliebige Formen annehmen, was z.B. zur Gestaltung von Firmenschriftzügen und Logos genutzt werden kann. Dieser Effekt wird dadurch erreicht, dass Teile des rechteckigen Gesamtbildes als transparent markiert werden, so dass der Hintergrund, in den das GIF-Bild eingeblendet wird, an diesen Stellen durchscheint.

Ein Nachteil von GIF ist die Beschränkung auf maximal 256 verschiedene Farben, die in einer Farbtabelle (Look Up Table, LUT) erfasst werden. Zwar gibt es Verfahren, wie z.B. das

Median-Cut- oder den Popularitäts-Algorithmus, die zu einem Bild mit beliebig vielen Farben eine optimal angepasste Farbtabelle mit nur noch 256 Farben berechnen, jedoch ist die Qualität des Ausgabebildes nicht so gut, wie bei einem Echtfarbbild. (Außerdem ist der Platzbedarf eines derart farbreduzierten Bildes im Vergleich zu einer ebenfalls verlustbehafteten JPEG-Kompression (siehe Kapitel 9.2.7) in der Regel wesentlich höher).

In einem GIF-Bild stellen die Pixelwerte also keine Grau- oder Farbwerte dar, sondern sie sind Indizes in die LUT und dort stehen dann in der entsprechenden Zeile die eigentlichen RGB-Farbwerte.

Zur Komprimierung wird bei GIF die verlustfreie LZW-Codierung eingesetzt. Diese war bis vor einigen Jahren patentrechtlich geschützt, so dass Software-Entwickler und Betreiber kommerzieller Internetseiten bei Verwendung von GIF-Bildern Lizenzgebühren bezahlen mussten. Mittlerweile darf die LZW-Codierung gebührenfrei genutzt werden.

GIF ist heute insbesondere im Bereich von Web-Anwendungen und dort besonders für synthetische Bilder bzw. Bilder mit einheitlich eingefärbten Objekten und Hintergründen immer noch ein sehr häufig eingesetztes Datenformat.

9.2.3 PNG

Das **P**ortable **N**etwork **G**raphics Format PNG (sprich „ping“) kann als Nachfolger von GIF angesehen werden. Bei PNG werden zuerst zeilenweise von Pixel zu Pixel die Differenzen der Werte berechnet und diese werden dann mit dem verlustfreien Deflate-Algorithmus komprimiert, der auch im zip-Format eingesetzt wird. Dabei handelt es sich um eine Kombination des nach Lempel und Ziv benannten LZ77-Verfahrens (Variante des LZW-Verfahrens, die von Anfang an lizenzkostenfrei war) und der Huffman-Codierung.

PNG erlaubt die Speicherung von Echtfarbbildern und erreicht im Vergleich zu GIF um bis zu 30 Prozent höhere Kompressionsraten. Damit stellt PNG eine echte Alternative zu GIF für die Einbindung von Graphiken in Internetseiten dar.

Als zusätzliche Erweiterung zu GIF stellt PNG einen echten Alpha-Kanal (siehe Kapitel 7.2.2.1) zur Verfügung, mit dem für jedes Pixel der Grad der Transparenz eingestellt werden kann. Auf diese Weise lassen sich z.B. weiche Übergänge zwischen einem Hintergrundbild und dem eingeblendeten PNG-Bild erzielen.

PNG-Bilder können bereits während des Ladens in einer „blockigen“ Grobansicht angezeigt werden, die dann schrittweise verfeinert wird; (progressive Decodierung). Außerdem kann man in PNG-Dateien Stichworte z.B. zum Autor und zur Beschreibung des Bildes unterbringen, so dass Web-Benutzer mit entsprechender Software-Unterstützung beispielsweise gezielt nach Bildern mit bestimmten Bildinhalten suchen können.

Eine Erweiterung zu PNG ist das **MNG**-Format (**M**ultiple Image **N**etwork **G**raphics) geplant, mit dem Animationen abgespeichert werden können.

9.2.4 FIF / FCI

Das **F**ractal **I**mage **F**ormat FIF wurde von der Firma Iterated Systems eingeführt, deren Mitbegründer M. Barnsley, der „Vater“ der fraktalen Bildkompression ist; siehe Kapitel 9.1.4.

FIF ist verlustbehaftet, d.h. das rekonstruierte Bild unterscheidet sich geringfügig vom Original. In der Kompressionsphase, die erheblich länger als die Dekompressionsphase dauert (asymmetrisches Verfahren), kann der Benutzer die Kompressionsrate und damit den Verlust an Detailgenauigkeit einstellen; die Kompression erfordert um so mehr Zeit und Speicherplatz je genauer die spätere Rekonstruktion möglich sein soll.

Erfahrungsgemäß kann man reale Bilder mit FIF bei kaum merklichen Qualitätsverlusten auf 5 bis 15 Prozent des ursprünglichen Speicherbedarfs reduzieren. Im Vergleich zu JPEG (s. Kap. 9.2.7) schneidet FIF bei photographischem Bildmaterial und gleich hoher Kompres-

sionsrate in der Regel besser ab, da es hier keine Blöcke gibt, die bei hoher Kompression sichtbar werden könnten; siehe z.B. MS-Encarta.

Die Dekompression kann progressiv erfolgen, so dass dem Betrachter schon zu Anfang eine grobe Bilddarstellung gezeigt wird, die dann schrittweise verfeinert wird. Ein weiterer positiver Aspekt von FIF zeigt sich beim Vergrößern von Bildausschnitten. Das Zooming dauert zwar je nach Vergrößerungsstufe im Vergleich zu anderen Ansätzen erheblich länger, jedoch wirken die Ergebnisse deutlich besser, denn die fraktale Vergrößerungstechnik führt zu „glatt“ wirkenden Vergrößerungen, die nicht die üblichen „Vergröberungen“ aufweisen.

Nachteilig an FIF ist, dass derzeit keine Transparenz (Alpha-Kanal) und keine Animationen vorgesehen sind. Eine Stichwortsuche, wie sie z.B. für Web-Benutzer interessant ist, wird ebenfalls noch nicht unterstützt.

FCI (**F**ractal **C**ompressed **I**mage) ist ein weiteres, aber nicht so effizientes Bilddatenformat, das fraktale Kompressionstechniken verwendet.

9.2.5 JPEG

Das JPEG-Format wurde von der **J**oint **P**hotographic **E**xpert **G**roup als internationaler Standard zur Kompression von Schwarz-Weiß-, Graustufen und Farbbildern entwickelt. Nicht zuletzt, da JPEG neben GIF bei der Entwicklung des HTML-Standards als eine der Möglichkeiten zur Einbindung von Graphiken in Internetseiten festgelegt wurde, gehört JPEG heute wegen seiner hohen Kompressionsraten (insbesondere für Photographien) im Bereich des Internets zu den sehr häufig benutzten Bilddatenformaten. Mit JPEG können Bilddaten auf vier verschiedene Weisen codiert werden, von denen eine verlustfrei, die anderen aber verlustbehaftet arbeiten.

Die verlustfreie JPEG-Codierung verzichtet im Gegensatz zu den anderen drei Möglichkeiten auf eine Transformation in den Ortsfrequenzbereich und erreicht vergleichsweise bescheidene Kompressionsraten. Diese Alternative wird deshalb im folgenden nicht weiter betrachtet.

Bei den übrigen drei Möglichkeiten wird die Diskrete Cosinus-Transformation (siehe Kapitel 9.1.5) mit nachfolgender Frequenzfilterung eingesetzt. Bei der einfachsten dieser Möglichkeiten, der sequentiellen Codierung, die auch als JPEG-Baseline bezeichnet wird, wird das zu codierende Bild systematisch von oben nach unten abgearbeitet. Entsprechend wird das Bild bei der Decodierung in voller Auflösung von oben nach unten aufgebaut.

Eine Erweiterung hierzu stellt die fortschreitende Codierung (progressive encoding) dar, bei der das Bild mehrfach bearbeitet wird, wobei in jedem Schritt die Bildqualität erhöht wird. Bei der Decodierung sieht der Benutzer also sehr früh ein vollständiges, aber unscharfes Bild und er kann den zeit- und kostenintensiven Bildaufbau bzw. die Bildübertragung gegebenenfalls frühzeitig abbrechen.

Die letzte Codierungsmöglichkeit ist die sogenannte hierarchische Codierung, bei der das Bild mehrfach in unterschiedlichen Auflösungen codiert wird (Bildpyramide). Auf diese Weise kann man z.B. auf eine niedriger aufgelöste Version eines Bildes zurückgreifen, ohne die höchstmögliche Auflösung decodieren zu müssen.

Im Weiteren wird das Vorgehen bei der JPEG-Baseline Codierung erläutert, auf der die beiden anderen Ansätze aufbauen. Der Gesamtablauf mit seiner mehrfachen Komprimierung ist in Abbildung 9.7 schematisch dargestellt.

Die einzelnen Teilschritte aus Abbildung 9.7 wurden in Kapitel 9.1.5 kurz erläutert und werden hier nur noch einmal aufgezählt. Zunächst wird das Eingangsbild in quadratische Blöcke von 8*8 Bildpunkten zerlegt, die dann getrennt weiterverarbeitet werden; (diese Blockzerlegung ist in Abbildung 9.7 nicht separat aufgeführt). Für jeden Block wird eine FDCT (**F**ast **D**iscrete **C**osine **T**ransformation), die eine optimierte, schnelle Version der normalen DCT ist, durchgeführt, um die Bilddaten in den Ortsfrequenzbereich zu transformieren. Danach werden hohe Frequenzanteile, die nicht oder kaum wahrnehmbaren Bilddetails entsprechen, im Rah-

men der Quantisierung unterdrückt; (1. Stufe der Kompression). Die verbleibenden Frequenzen werden anschließend in einem Zick-Zack-Kurs mit Hilfe einer Lauflängencodierung erfasst; (2. Stufe der Kompression). Das Ergebnis (über alle Blöcke) wird abschließend mit der Huffman-Codierung weiter komprimiert; (3. Stufe der Kompression). Die Decodierung dieses symmetrischen Ansatzes durchläuft alle Schritte in umgekehrter Reihenfolge.

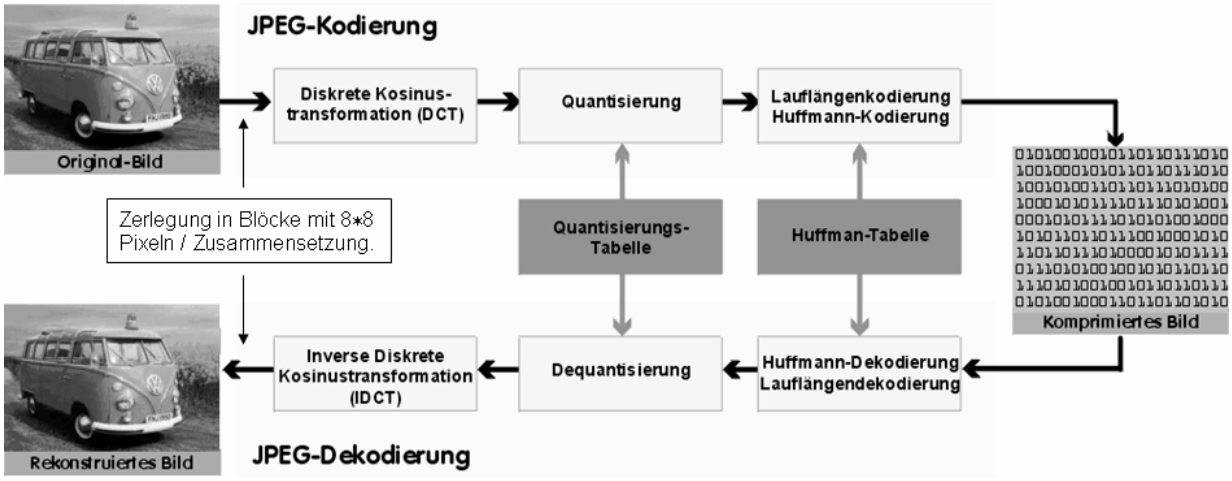


Abb. 9.7: Schematisches Vorgehen bei der JPEG-Baseline-Codierung; alle Schritte außer der Huffman-Kodierung bzw. -Dekodierung beziehen sich auf die 8∗8-Blöcke.

Die JPEG-Codierung, die sich vor allem für photographische Aufnahmen hoher Auflösung und weniger für einfache Graphiken eignet, ist sehr zeitaufwendig, aber sie erlaubt wie FIF extrem hohe Kompressionsraten. Der Kompressionsfaktor, der sich umgekehrt proportional zur Rekonstruktionsgüte verhält, kann vom Benutzer fein abgestuft eingestellt werden. Für reale Bilder sind Kompressionen auf bis zu zehn Prozent des ursprünglichen Umfangs möglich. Zu stark komprimierte Bilddaten neigen allerdings zu Rekonstruktionsfehlern, wie 8*8-Blockstrukturen, sowie zu „Schwingungen“ und Unschärfen an Kanten.

Schwarz-Weiß-Bilder und Grautonbilder können direkt verarbeitet werden, wohingegen Farbbilder, die eine Farbtabelle verwenden, zuerst in echte Farbbilder umgerechnet werden. Für Farbbilder wird i.d.R. der YUV-Farbraum verwendet, da sich hier eine weitere Kompressionsmöglichkeit durch Irrelevanz-Reduktion ergibt: während beim RGB-Farbmodell alle drei Farbkanäle für den Betrachter gleich wichtig sind, kann man beim YUV-Modell die Chrominanzen U und V mit der halben räumlichen Auflösung erfassen, ohne dass dies zu erkennbaren Qualitätsverlusten führt; (vgl. Kapitel 7.2.2.2).

Als Erweiterung zu JPEG wurde Motion-JPEG eingeführt, um auch Bildfolgen komprimieren zu können. Da hierbei aber die Ähnlichkeit der zeitlich aufeinanderfolgenden Einzelbilder nicht genutzt wird, schneidet Motion-JPEG im Vergleich zu MPEG (siehe 9.2.9) wesentlich schlechter ab und hat sich deshalb nicht durchsetzen können.

JPEG könnte jederzeit durch den neuen Standard JPEG-2000 (siehe nächstes Teilkapitel) abgelöst werden, wenn der Markt dies mittragen würde, da JPEG-2000 bei gleichen Kompressionsraten eine deutlich bessere Bildqualität verspricht.

9.2.6 JPEG-2000 / WI

Wegen der Nachteile des auf der Diskreten Cosinus-Transformation aufbauenden JPEG-Formats, wurde das wavelet-basierte JPEG-2000-Format (JP2) bzw. das **W**avelet-**I**mage-Format WI eingeführt.

Obwohl JPEG-2000 gegenüber dem ursprünglichen JPEG bei gleicher Bildqualität deutlich kleinere Dateigrößen bzw. bei gleicher Kompressionsrate eine spürbare bessere Bildqualität gewährleistet, bleibt abzuwarten, ob sich JPEG-2000 auf dem Markt durchsetzt. Derzeit wird es nur von wenigen Browsern unterstützt und auch die Hersteller von Digital-Kameras

setzen bei der Kompression der aufgenommenen Bilder meist noch auf das alte JPEG-Format.

Bei der JPEG-2000-Codierung eines Bildes schließt sich an die in Kapitel 9.1.4 geschilderte Wavelet-Transformation mit nachfolgender Quantisierung der Wavelet-Koeffizienten noch eine aufwendige verlustfreie Codierung der Koeffizienten an. Diese stützt sich insbesondere auf die Tatsache, dass in Abhängigkeit von der "Stärke" der eingestellten Quantisierung ein sehr hoher Anteil der Koeffizienten Null ist, und ansonsten nur noch wenige verschiedene Werte auftreten.

Durch eine geeignete Sortierung der quantisierten Koeffizienten kann außerdem erreicht werden, dass z.B. bei einer web-basierter Nutzung das Bild bereits nach wenigen übertragenen Daten in einer schlechten Auflösung angezeigt werden kann. Dazu müssen die Daten aus der rechten unteren Ecke der Koeffizientenmatrix zuerst übertragen werden. Indem die bei der Wavelet-Transformation vorgenommene iterative Verkleinerung des Bildes schrittweise rückgängig gemacht wird, entsteht das Bild mit der Übertragung der nächsten Detailbild-Koeffizienten Schritt für Schritt in immer besserer Auflösung (skalierbare progressive Übertragung und Decodierung).

JPEG-2000 ist in gleicher Weise für reale Bilder mit weichen Farbverläufen und für synthetische Bilddaten mit harten Farbübergängen geeignet. JPEG-2000 verhält sich robust gegen Übertragungsfehler und unterstützt Transparenz. Außerdem wird die Erfassung von Meta-Daten (z.B. zum Bildinhalt) unterstützt.

JPEG-2000 ist bereits in den DICOM-Standard für medizinische 2D- und 3D-Bilddaten aufgenommen.

9.2.7 MPEG

Das MPEG-Format wurde von der **M**otion **P**icture **E**xpert **G**roup entwickelt und erlaubt, wie der Name aussagt, die Kompression von Bewegtbildern, d.h. Bildfolgen. Mit der MPEG-Kompression kann zusätzlich zu den Bilddaten auch Audio-Information wie z.B. die Tonspur eines Filmes erfasst werden. Der Audio-Aspekt, der auf einer eindimensionalen DCT beruht, wird im Weiteren nicht betrachtet.

Während andere Verfahren wie z.B. GIF und JPEG dahingehend erweitert wurden, dass sie statt einem auch mehrere Bilder einzeln und unabhängig voneinander erfassen können (***Intraframe-Codierung***), wurde MPEG als ***Interframe-Codierungsverfahren*** entwickelt. Das bedeutet, dass zusätzlich zu den anderen Kompressionsansätzen noch die zeitliche Kohärenz, d.h. die Ähnlichkeit aufeinanderfolgender Bilder einer Bildfolge, ausgenutzt wird. Indem also im Prinzip nur die von Bild zu Bild auftretenden Änderungen abgespeichert werden, ergibt sich gegenüber der Abspeicherung der komprimierten Einzelbilder nochmals eine deutliche Datenreduktion.

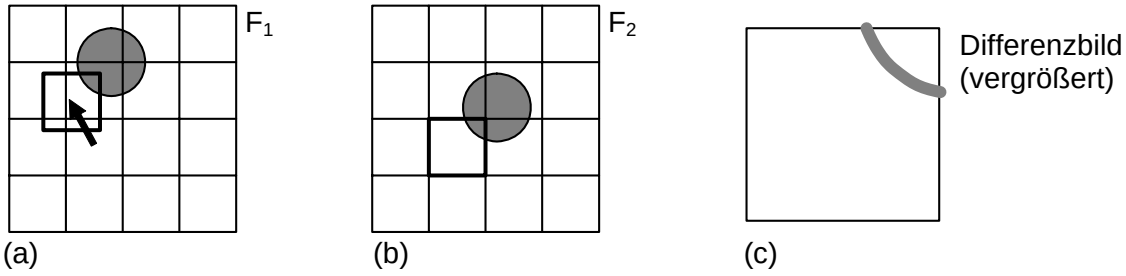


Abb. 9.8: Beispiel für die Ermittlung eines Verschiebevektors und eines Differenzbildes bei der MPEG-Codierung; der Ball ändert von Bild F_1 zu F_2 seine Position und Größe.
(a) In Bild F_1 ist (zusammen mit dem zugehörigen Verschiebevektor) eingezeichnet, wo die Farbwerte des in Bild F_2 dick markierten Makroblocks am besten aus Bild F_1 ausgelesen werden können;
(b) Bild F_2 mit hervorgehobenem Makroblock;
(c) Differenzbild zu diesem Makroblock in vergrößerter Darstellung.

Die Grundidee der bei MPEG eingesetzten Interframe-Codierung basiert darauf, dass für zwei zeitlich nicht zu weit auseinanderliegende Bilder F_1 und F_2 einer Bildfolge gilt, dass große Teile von F_1 in F_2 mehr oder weniger unverändert an derselben oder einer geringfügig verschobenen Bildposition wieder auftreten. Indem man die Verschiebungen für sogenannte Makroblöcke mit Hilfe von ***Verschiebevektoren*** angibt ***(Bewegungsprädiktion)*** und die eventuellen Änderungen mit Hilfe von ***Differenzbildern*** beschreibt, lässt sich Bild F_2 mit wenigen Zusatzinformationen aus Bild F_1 rekonstruieren; siehe Abbildung 9.8.

Bild F_2 wird als ***unidirektional prädiziertes*** (vorhergesagtes) sogenanntes ***P-Bild*** bezeichnet. Ergänzend hierzu gibt es noch die ***bidirektionale Prädiktion***, bei der ein Bild aufgrund seiner Unterschiede zu einem vorhergehenden und einem nachfolgenden Bild vorhergesagt wird. Diese Bilder werden als ***B-Bilder*** bezeichnet.

Im konkreten Fall werden bei MPEG in bestimmten Abständen Bilder der Bildfolge vollständig mit der Intraframe-Codierung von JPEG als sogenannte ***I-Bilder*** abgespeichert. In den Bereichen zwischen diesen vollständig abgespeicherten I-Bildern werden in kleineren Abständen mehrere unidirektional prädizierte P-Bilder erfasst. Die restlichen Zwischenbilder werden bidirektional aus den vorangegangenen bzw. nachfolgenden I- und P-Bildern abgeleitet; siehe Abbildung 9.9.

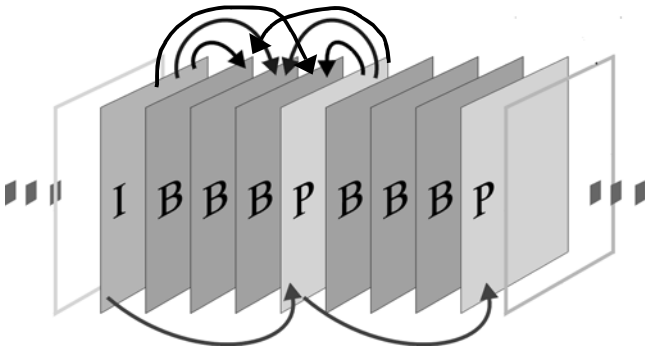


Abb. 9.9: Beispiel für die MPEG-Codierung eines Ausschnittes aus einer Bildfolge.

Der aufwendigste Teil der MPEG-Kompression ist die Bewegungsprädiktion, d.h. die Berechnung der Verschiebevektoren und Differenzbilder. Da dieser Aufwand bei der Decodierung entfällt, handelt es sich bei MPEG um ein asymmetrisches Verfahren. Da die Abspeicherung bzw. Übertragung von Bildfolgen eine konstante Datenrate erfordert, muss die MPEG-Codierung kontinuierlich auf den Bildinhalt reagieren: z.B. würde ein fein strukturierter Bildinhalt mit vielen hohen Frequenzanteilen eine wesentlich gleichmäßiger besetzte Matrix der DCT-Koeffizienten liefern, als dies bei einem „ruhigen" Bild der Fall wäre. Um in beiden Fällen eine Reduzierung auf die gleiche Datenmenge zu erreichen, wird im ersten Fall durch eine Anpassung der Quantisierungstabellen eine wesentlich gröbere Quantisierung der Frequenzen vorgenommen, was zu einer deutlich schlechteren Bildqualität führen kann.

Das Abspielen einer Bildfolge in umgekehrter Reihenfolge (Bildrücklauf), sowie der wahlfreie Zugriff auf Einzelbilder (z.B. zum Editieren und Schneiden) ist beim MPEG-Format aufgrund der Interframe-Codierung nur mit zusätzlichen Berechnungen und Zwischenspeicherungen möglich. Unproblematisch ist dagegen der schnelle Vor- und Rücklauf (Zeitraffer), da hier einfach nur auf die I-Bilder (und dabei u.U. sogar nur auf die DC-Koeffizienten jedes 8*8-Blockes dieser Bilder) zugegriffen wird.

Als Weiterentwicklung zum ursprünglichen MPEG-1-Verfahren gibt es hauptsächlich MPEG-2 und MPEG-4, die sich aber an andere Zielgruppen richten. MPEG-1 wurde für den Videomarkt entwickelt, um Bildsequenzen mit mäßigem Detailgehalt und nicht zu gravierenden Szenenwechseln mit einer Datenrate von bis zu 1.5 MBit/s abdecken zu können; (z.B. Video von CD). Die Qualität der MPEG-1-Codierung einer Bildfolge entspricht in etwa der eines VHS-Recorders. Im Gegensatz hierzu ist MPEG-2 bei gleicher Vorgehensweise mit größeren Bildformaten und besserer Bildqualität bei einer Datenrate von bis zu 40 MBit/s auf Video-Übertragungen in Breitbandnetzen zugeschnitten. Die Forderung nach HDTV-Formaten

(High Definition TV), die ursprünglich mit MPEG-3 abgedeckt werden sollte, wurde mittlerweile in MPEG-2 integriert. MPEG-4 ist ein skalierbares Format, das damit z.B. auch für die mobile Kommunikation in schmalbandigen Netzen geeignet ist. Das Spektrum der Skalierbarkeit reicht vom (künftigen) Handy bis HDTV, wobei die Datenrate und damit die Bildgröße und Güte des decodierten Videos den jeweiligen Gegebenheiten bzw. Erfordernissen angepasst wird. Dabei kann MPEG-4 im Vergleich zu den Vorgängerversionen noch höher komprimieren bzw. es liefert bei gleicher Bitrate eine bessere Bildqualität.

Weitere Aspekte von MPEG-4 sind der objektorientierte Ansatz mit Hilfe sogenannten ***audio-visueller Objekte*** (Avs) und die Möglichkeit, mit solchen Objekten zu interagieren und so z.B. Aktionen auszulösen.

9.2.8 Schlussbemerkungen zu Kompressionstechniken für Rasterbilder

Neben den in diesem Kapitel aufgezählten Bilddatenformaten gibt es noch zahlreiche Alternativen, die aber zum Teil nur noch in geringem Umfange genutzt werden oder sich auf dem Markt (noch) nicht entsprechend durchsetzen konnten. So gibt es z.B. zur Wavelet-Kompression von Bildfolgen das kommerzielle und nicht standardisierte **Indeo**-Videokompressionsverfahren, das von Intel entwickelt wurde.

Im gegenseitigen Vergleich der unterschiedlichen Bilddatenformate kann hinsichtlich der Kompressionsrate generell gesagt werden, dass verlustbehaftete Verfahren deutlich günstiger abschneiden, als verlustfreie Verfahren. Bei realen Daten kann man mit den verlustfreien Verfahren selten um mehr als 20 Prozent komprimieren. In ungünstigen Fällen kann das codierte Bild sogar mehr Speicherplatz belegen, als das Original. Mit den verlustbehafteten Kompressionsverfahren sind (in Abhängigkeit vom Bildinhalt) häufig Kompressionen auf 10 bis 30 Prozent des ursprünglichen Platzbedarfs möglich. Der Benutzer muss bei der Einzelbildcodierung also in erster Linie entscheiden, ob er Qualitätsverluste hinnehmen kann und wie hoch diese sein dürfen.

Bei Bewegtbildern gibt es bei hohen Kompressionsanforderungen kaum eine Alternative zu MPEG oder verwandten Formaten.

9.3 Bildbeschreibende Dateiformate

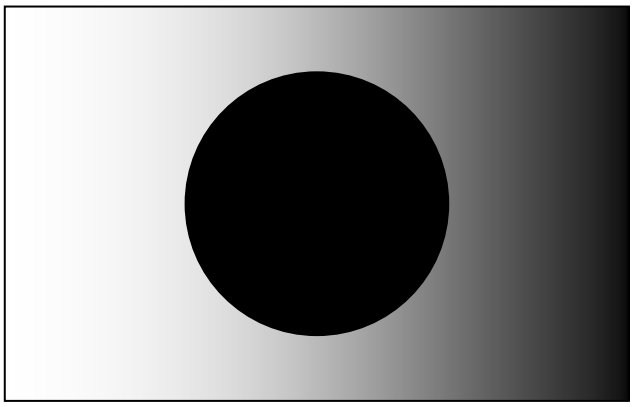
Im Gegensatz zu den Raster-Dateiformaten, die die Bildinformation in Form von Pixeln (i.d.R. komprimiert) enthalten, besteht der Inhalt von bildbeschreibenden Dateiformaten aus ***Meta-Daten***, d.h. Daten, die Informationen über andere Daten liefern. Diese Meta-Daten bestehen im Wesentlichen aus den vektoriellen Daten, die die im Bild dargestellten Objekte definieren (siehe Kapitel 7). Zusätzlich können die Meta-Daten auch 2D-Rasterdaten enthalten, die z.B. zur Texturierung einer Objektoberfläche verwendet werden. Im CAD-Bereich können die verwendeten Dateiformate noch weitere Meta-Daten (z.B. Materialnamen und Materialeigenschaften) umfassen.

Im Folgenden werden mehrere gebräuchliche bildbeschreibende Dateiformate vorgestellt. Neben den z.B. von Vektor-Zeichenprogrammen bekannten Dateiformaten CGM, SVG und VML gehören hierzu auch die Formate PostScript und PDF aus der Druckvorstufe sowie IGES, STEP und DXF aus dem CAD-Bereich. Abbildung 9.10 vergleicht an einem Beispiel den Speicherplatzbedarf einer Graphik bei Verwendung verschiedener Formate.

9.3.1 CGM

Das ***CGM***-Format ***(Computer Graphics Metafile)*** ist international genormt und wird in vielen verschiedenen Industriezweigen als Standard-Format für geräteunabhängige zweidimensionale Graphik-Daten verwendet. Die Dateien werden üblicherweise im ASCII-Format abgelegt.

Neuerdings wächst die Bedeutung von CGM-Dateien, da das auf CGM aufbauende WebCGM-Format z.B. dazu verwendet wird, Graphiken in interaktiven elektronischen Katalogen und Handbüchern darzustellen.



Name	Größe
Kreise.bmp	372 KB
Kreis.svg	2 KB
Kreis.ps	2 KB
Kreis.pdf	4 KB
Kreis.eps	2 KB
Kreis.dxf	2 KB

Abb. 9.10: Beispiel für die Abspeicherung einer Graphik (links: BMP-Bild der Größe 447*283 Pixel) mit verschiedenen Datei-Formaten (rechts).

9.3.2 SVG

SVG (Scalable Vector Graphics) ist ein Standard zur Beschreibung zweidimensionaler Vektor-Graphiken in XML-Syntax (XML: Extensible Markup Language = erweiterbare Auszeichnungssprache). SVG-Dateien können auch Rastergraphiken und Text enthalten; Animationen sind möglich. SVG kann von den meisten Browsern dargestellt und mit Hilfe eines Text-Editors bearbeitet werden.

9.3.3 VML

Vector Markup Language (VML) ist eine Auszeichnungssprache zur Beschreibung zweidimensionaler vektororientierter Graphiken in XML. Obwohl VML als Web-Standard abgelehnt wurde, wird VML in vielen Anwendungen eingesetzt (z.B. Microsoft Office, Google).

9.3.4 PostScript - Encapsulated PostScript

Bei ***PostScript*** von Adobe handelt es sich um eine vektorgraphikfähige Seitenbeschreibungssprache, die eine vollständige Programmiersprache darstellt. PostScript ist ein Standard in der Druckindustrie, der aber mehr und mehr in Konkurrenz zu PDF (siehe 9.3.5) steht.

Encapsulated PostScript (EPS) wurde entwickelt, um die Einbettung von im PostScript-Format vorliegenden Graphiken in andere PostScript-Dateien zu erleichtern. In einer EPS-Datei stehen also die PostScript-Anweisungen für eine Graphik, die höchstens eine Seite umfassen darf. Jede EPS-Datei enthält zusätzlich zur eigentlichen Graphik noch Angaben über die sogenannte Bounding-Box, d.h. das kleinste achsenparallele Rechteck, das die Graphik enthält.

9.3.5 PDF

Das graphikfähige ***PDF (Portable Document Format)*** von Adobe ist aus PostScript hervorgegangen und verwendet dasselbe Graphikmodell, das hier aber z.B. noch um Transparenz erweitert wurde. PDF dient als plattformübergreifendes Dokumenten-Dateiformat, das z.B. in der Druckvorstufe und in der Dokumenten-Archivierung eingesetzt wird. Kostenlose Reader sind für jedes Betriebssystem verfügbar. In naher Zukunft soll PDF ISO-normiert werden.

9.3.6 IGES

IGES (Initial Graphics Exchange Specification) ist ein herstellerunabhängiges Format zum Austausch von 2D- und 3D-CAD-Daten, das u.a. in der Automobil-Industrie eingesetzt wird. Die Zeichnungselemente sind in sogenannte Entities gegliedert und werden i.d.R. mit

ASCII-Zeichen beschrieben. Für bestimmte Anwendungsfälle, die z.B. die Unterscheidung von „innen“ und „außen“ bei Flächen erfordern, empfiehlt sich die Verwendung des neueren Formates STEP (siehe 9.3.7).

9.3.7 STEP

STEP (Standard for Exchange of Product Data Model) ist ein ISO-Standard zur Beschreibung von Produktdaten, der den gesamten Produkt-Lebenszyklus abdecken kann. Aufgrund der Standardisierung eignet sich STEP auch für den Datenaustausch zwischen unterschiedlichen Systemen im gesamten CAx-Bereich. Aufgrund der Standardisierung ist bei STEP gewährleistet, dass beim Datenaustausch keine Informationen verloren gehen, was bei früheren Systemen häufig der Fall ist.

9.3.8 DXF

Das von Autodesk entwickelte ***DXF (Data Exchange Format)*** wurde für das CAD-Programm AutoCAD entwickelt und dient zum CAD-Datenaustausch. DXF-Dateien enthalten ASCII-Texte mit Anweisungen, die die einzelnen Elemente eines CAD-Modells beschreiben.

10 Gewinnung digitaler Bilder

Digitale Bilder können auf unterschiedlichste Art erzeugt werden. Beispielsweise können reale Gegebenheiten mit Hilfe eines Sensors (z.B. Fotoapparat, Video-Kamera) aufgenommen werden, oder aber man verwendet einen Scanner, um eine Vorlage einzuscannen; siehe rechte Seite von Abbildung 10.1. Alternativ hierzu können digitale Bilder auch im Rahmen der Computer Graphik synthetisch generiert werden; siehe linke Seite von Abbildung 10.1.

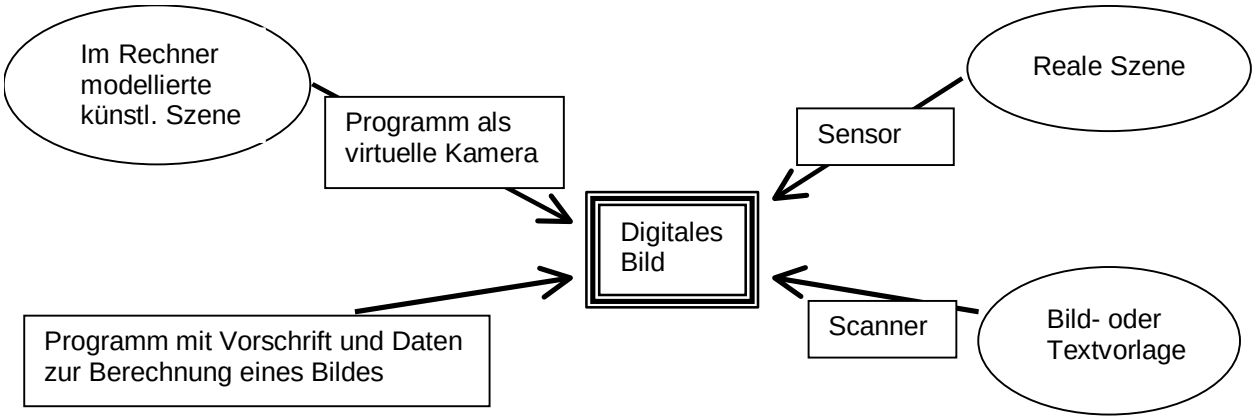


Abb. 10.1: Schema für die Gewinnung digitaler Bilder.

10.1 Sensoren

Es gibt eine Vielzahl unterschiedlichster ***Sensoren***, um Bilder einer realen Szene zu erzeugen. Am meisten verbreitet sind dabei Digitalkameras (inkl. Handys und Handycams) zur Aufzeichnung von Einzelbildern und Bildfolgen. Die Aufnahmetechnik ist häufig nicht auf den sichtbaren Bereich des elektromagnetischen Spektrums beschränkt, sondern kann auch andere Spektralbereiche erfassen, wie z.B. Ultra-Violettes-Licht (UV), Wärmestrahlung, Laser, Ultraschallwellen und Radar- oder Röntgenstrahlung.

Bei den Sensoren unterscheidet man zwischen passiven und aktiven Systemen. Bei den ***passiven Systemen*** wird die im Raum vorhandene Strahlung zur Bilderzeugung genutzt; (z.B. Fotoapparat und Wärmebildkamera). Bei den ***aktiven Systemen*** wird die Strahlung vom Gerät selbst erzeugt und abgestrahlt; (z.B. Laserabtaster und abbildendes Radar). Die von der Szene zurückgestrahlte Energie wird empfangen und aufgezeichnet. (Ausnahme: beim Röntgengerät wird die die Objekte durchdringende Strahlung gemessen).

Einige spezielle Aufnahmetechniken erzeugen analoge Signale, die dann in diskrete Grau- bzw. Farbwerte konvertiert werden müssen. Bei den meisten Aufnahmetechniken entsteht aber direkt ein digitales Bild: digitale Fotografie, Kernspin-Tomographie usw.. Letzere ist ein Beispiel dafür, dass digitale Bilddaten auch in dreidimensionaler Form erfasst werden können.

Die Abbildung der realen Außenwelt auf die Bildebene eines Sensors soll im Folgenden am Beispiel des ***Kameramodells der Lochkamera***, das allen Fotoapparaten und Kameras zugrunde liegt, erläutert werden; siehe Abbildung 10.2. Eine Lochkamera ist im Prinzip ein Kasten, dessen hintere innere Fläche die ***Bildebene*** darstellt. Gegenüber der Bildebene befindet sich im Abstand der ***Bildweite*** ein Loch (die sogenannte ***Blende***), durch das die Lichtstrahlen auf die Bildebene fallen. Um einen Punkt eines Objektes der realen Außenwelt abzubilden, berechnet man den Schnittpunkt des Strahls, der von diesem Punkt ausgeht und durch die Blende verläuft, mit der Bildebene. Ein derartiger Strahl (mit umgekehrter Richtung) wird als ***Sehstrahl*** bezeichnet. Aufgrund der geschilderten Abbildungsgeometrie werden alle Objekte auf der Bildebene kopfstehend und seitenverkehrt dargestellt. Da alle 3D-Punkte des Sehstrahls auf ein und denselben 2D-Punkt der Bildebene abgebildet werden, ist diese Projektion nicht umkehrbar, und die realen Gegebenheiten können anhand eines einzigen Bildes nicht dreidimensional rekonstruiert werden. (Hierzu ist z.B. ein zweites Bild - wie bei der Stereo-Fotografie - erforderlich).

In der Bildebene sitzen die ***Sensorelemente***; z.B. ***CCD-Chip*** (**C**harge **C**oupled **D**evice = ladungsgekoppelter Bildsensor). Jedes Sensorelement empfängt durch die Blende die Lichtenergie, die von einem kleinen Fleck der abgebildeten Szene ausgeht. Vereinfacht ausgedrückt wird also z.B. bei der Erzeugung eines Grauwertbildes die mittlere Helligkeit dieses Flecks gemessen und in eine elektrische Spannung umgewandelt. Diese Spannung wiederum wird dann mittels A/D-Wandlung in einen Grauwert überführt, der an der entsprechenden Stelle des digitalen Bildes eingetragen wird.

Die Anzahl der Sensorelemente bestimmt bei den digitalen Kameras die räumliche Auflösung des Bildes; näheres zur Auflösung in Bezug auf die Erkennbarkeit von Bilddetails ist in Kapitel 10.2 geschildert.

Viele Sensoren liefern heute eine Farbauflösung von weit mehr als 24 Bit (***RAW-Daten*** von englisch „roh“). Diese Farbauflösung kann dann entweder exportiert oder intern optimal auf 24 Bit Farbtiefe umgerechnet werden (je 8 Bit für R, G und B; siehe Kapitel 7), so dass die resultierenden digitalen Farbbilder über etwa 16,7 Millionen Farben verfügen können.

Das oben geschilderte Vorgehen wird als ***Digitalisierungs-Modell*** bezeichnet:
- zuerst ***Rasterung***, d.h. Aufteilung der gesamten „gesehenen“ Fläche in eine Rastermatrix;
- dann ***Quantisierung***, d.h. A/D-Wandlung der analogen Licht-Energie (Spannung) in diskrete Grauwerte.

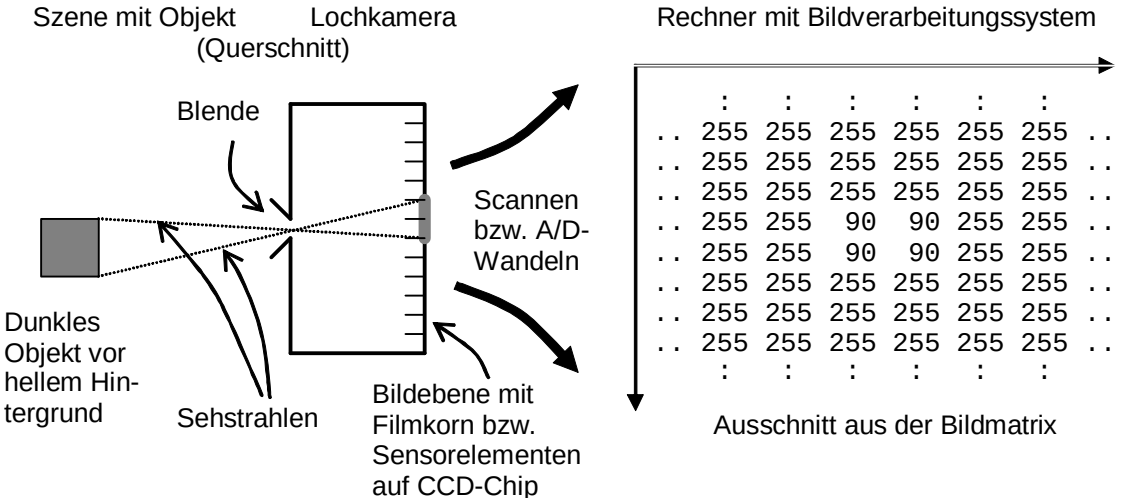


Abb. 10.2: Prinzip der Erzeugung eines Grauwertbildes mit Hilfe einer Lochkamera: jedes Element der Bildmatrix entspricht dem vom entsprechenden Sensorelement „gesehenen“ Fleck der aufgenommenen Szene.

10.2 Scanner

Mit ***Scannern*** sind im Folgenden nur solche 2D-Scanner gemeint, die von einer analogen Hardcopy-Vorlage (z.B. Papierbild, Diapositiv oder Textseite) ein digitales Abbild für die weitere Verarbeitung im Rechner erstellen. (Darüber hinaus gibt es z.B. noch 3D-Scanner, die ein räumliches Objekt dreidimensional erfassen können).

Die üblichen Scanner sind ***Flachbettscanner***, bei denen die Vorlage wie bei einem Fotokopierer auf eine Glasplatte gelegt wird. Beim eigentlichen Scanvorgang von nicht-transparenten Vorlagen – wie z.B. Papierbildern – wird die Vorlage von unten mit einer starken Lichtquelle angestrahlt (Auflicht) und die reflektierte Bildhelligkeit bzw. –farbe wird mit Hilfe der Sensorelemente eines oder mehrerer CCD-Chips gemessen und aufgezeichnet. Dazu sind die Sensorelemente vom Prinzip her dicht benachbart als Sensorzeile angeordnet, und diese Sensorzeile wird senkrecht zu ihrer Längsrichtung unter der Glasplatte hindurchbewegt; siehe Abbildung 10.3.

Bei transparenten Vorlagen – wie z.B. Diapositiven – wird die Vorlage von oben her angestrahlt, und die Sensorelemente der durchlaufenden Sensorzeile messen die die Vorlage durchdringende Strahlung (Durchlicht).

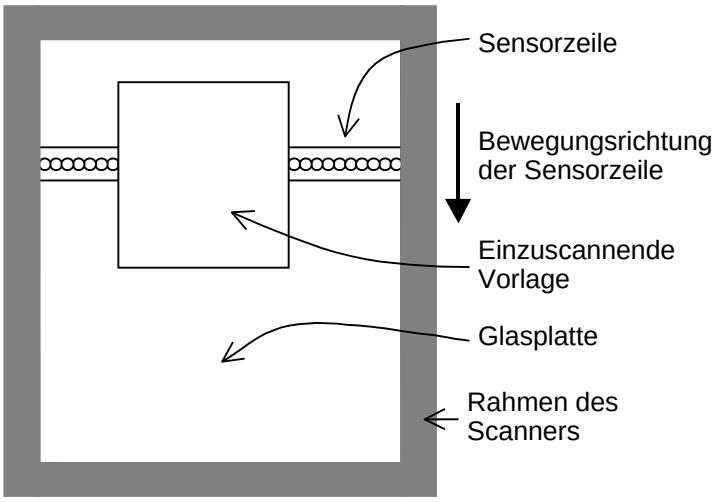


Abb. 10.3: Prinzip des Flachbettscanners

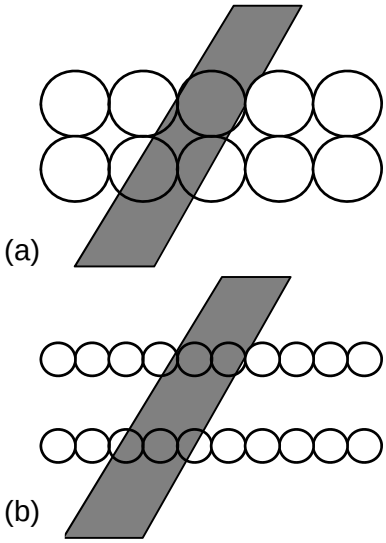


Abb. 10.4: Prinzip des Abtasttheorems

Unter der ***optischen Auflösung*** eines Scanners versteht man die Anzahl der Abtastpunkte, die auf einem Inch (= Zoll ≈ 25,4 mm) der abzutastenden Vorlage platziert werden. Die optische Auflösung wird in ***dpi*** bzw. ***ppi*** (d.h. **d**ots bzw. **p**ixel **p**er **i**nch) angegeben.

Im Folgenden wird kurz auf die Wahl der Scanner-Auflösung im Hinblick auf eine spätere Auswertung der Bilder eingegangen; (die Wahl der Auflösung hinsichtlich der Bildausgabe auf einem Bildschirm oder Drucker wird in Kapitel 10.1.4 bzw. 10.2 behandelt.)

Falls die Bilder (z.B. mit Hilfe der digitalen Bildverarbeitung) ausgewertet werden sollen, so bestimmt laut ***Abtasttheorem*** (siehe unten) die Größe der kleinsten interessierenden Bilddetails die Wahl der Scanneraufl̈osung.

Anschaulich (und vergröbert) besagt das Abtasttheorem im vorliegenden Fall, dass man z.B. eine im Bild vorhandene Linie nur dann korrekt erfassen kann, wenn die Abtastfleckgröße höchstens so groß wie die halbe Linienbreite ist. In Abbildung 10.4 (a) ist der Abtastfleck zu groß gewählt (***Unterabtastung***) und bei ungünstiger Platzierung der Sensorelemente (siehe untere Abtastzeile in Abbildung 10.4 (a)) kann die Linie nicht korrekt erfasst werden; durch die Mittelung der Helligkeit innerhalb eines Abtastflecks kann die Linie verschwinden oder zumindest stark abgeschwächt werden. Abbildung 10.4 (b) zeigt, dass bei Berücksichtigung des Abtasttheorems immer mindestens ein Abtastfleck vollständig innerhalb der Linie liegt.

Beispiel: Beim Scannen mit einer Scanner-Auflösung von 300 dpi werden Details mit einer Seitenlänge von etwa 0.2 mm noch mit mindestens einem vollständigen Abtastpunkt erfasst.

***Abtasttheorem von Shannon / Nyquist*:** Ein kontinuierliches Signal mit begrenzter Bandbreite kann durch eine Menge von Abtastungen in äquidistanten Zeitabschnitten vollständig repräsentiert werden, wenn die Abtastung häufiger, als mit der halben Periode der höchsten im Signal auftretenden Frequenz erfolgt. Die Abtastfrequenz muss also mehr als doppelt so hoch sein, wie die höchste im Frequenzgemisch des Signals vorhandene Frequenz.

Das Abtasttheorem muss nicht nur beim Scannen, sondern bei jeglicher Form der Bilderzeugung berücksichtigt werden. Bei einer Verletzung des Abtasttheorems können Artefakte (Aliasing) entstehen, die das Bild verfälschen. Ein Spezialfall hiervon sind die sogenannten ***Moiré-Muster***, die beim Scannen oder bei Bildaufnahmen auftreten können, wenn z.B. eine periodische Bildstruktur vorliegt, die etwa dieselbe Frequenz wie die Abtastfrequenz hat. Diese störenden Muster erhält man z.B. häufig dann, wenn man aufgerastert gedruckte Bildvorlagen aus Zeitungen oder Zeitschriften einscannt; (Interferenz zwischen dem Druck- und dem Scanraster).

Schlussbemerkung: Bei den sogenannten ***interpolierenden Scan-Auflösungen*** wird die Bildvorlage mit der höchsten möglichen optischen Auflösung abgetastet. Danach werden zwischen diesen „echten" Bildpunkten zusätzliche Bildpunkte eingefügt, deren Grau- bzw. Farbwert aus den Werten der umgebenden Bildpunkte interpoliert wird. Im Sinne der Bildverarbeitung liefert eine derartige Abtastung also keinen Informationsgewinn, sondern lediglich ein rechnerisch vergrößertes Bild.

10.3 Prozedurale Erzeugung synthetischer Bilder

Während die Kapitel 10.1 und 10.2 gezeigt haben, wie digitale Bilder einer Szene oder Vorlage mit Hilfe von Sensoren oder Scannern erzeugt werden können, sollen in diesem und dem nächsten Kapitel kurz zwei Prinzipien vorgestellt werden, wie Bilder im Rechner synthetisch generiert werden können; siehe linke Seite der Abbildung 10.1.

Der einfachste Ansatz zur Erzeugung eines synthetischen Bildes besteht darin, alle Informationen über das Aussehen des Bildes in eine Berechnungsvorschrift zu integrieren. Diese Möglichkeit wird als ***prozeduraler Ansatz*** bezeichnet; siehe links unten in Abb. 10.1.

Ein besonders einfaches Beispiel für den prozeduralen Ansatz stellt der Graukeil aus Kapitel 7 Abbildung 7.4 (d) unten dar. Bei diesem werden alle Grauwerte des Bildes in Abhängigkeit von der Spaltenposition des jeweiligen Bildpunktes berechnet; (Bilder müssen nicht quadratisch sein!) Bei 256 Bildspalten und 2^8=256 möglichen Grauwerten erhält jedes Element jeder Bildzeile seine Spaltenposition als Grauwert zugewiesen. Das Bild wird hier direkt in der Rasterdarstellung erzeugt.

Ein zweites einfaches Beispiel für eine prozedurale Bildgenerierung – hier aber in Vektordarstellung – stellt die Anwendung eines iterativen Ersetzungs-Systems dar. Solche Systeme stellen eine von vielen Möglichkeiten zur Erzeugung sogenannter ***Fraktale*** dar; siehe Abbildung 10.5.

Mit der Ausgangsgegebenheit X_0 und der Vorschrift f iteriert man: $X_{n+1} := f(X_n)$. Konkret bedeutet dies hier, dass jede Strecke bei jedem Iterationsschritt durch eine identische Strecke, sowie weitere, teilweise kürzere Strecken, die in der Mitte der "alten" Strecke abzweigen bzw. als Verzweigung am oberen Ende der "alten" Strecke angesetzt sind, ersetzt wird. Die Ausgangsgegebenheit, die hier einer senkrecht angeordneten Strecke entspricht (siehe Abb. 10.5 (a)), wird durch den in Abbildung 10.5 (b) gezeigten Streckenzug ersetzt. Im weiteren Verlauf der Iteration wird jede Strecke des Streckenzuges wieder durch eine entsprechend verkleinerte und gedrehte Kopie des Streckenzuges aus (b) ersetzt; siehe Abbildung 10.5 (c). Nach acht Iterationsschritten entsteht das in Abbildung 10.5 (d) vergrößert dargestellte Bild.

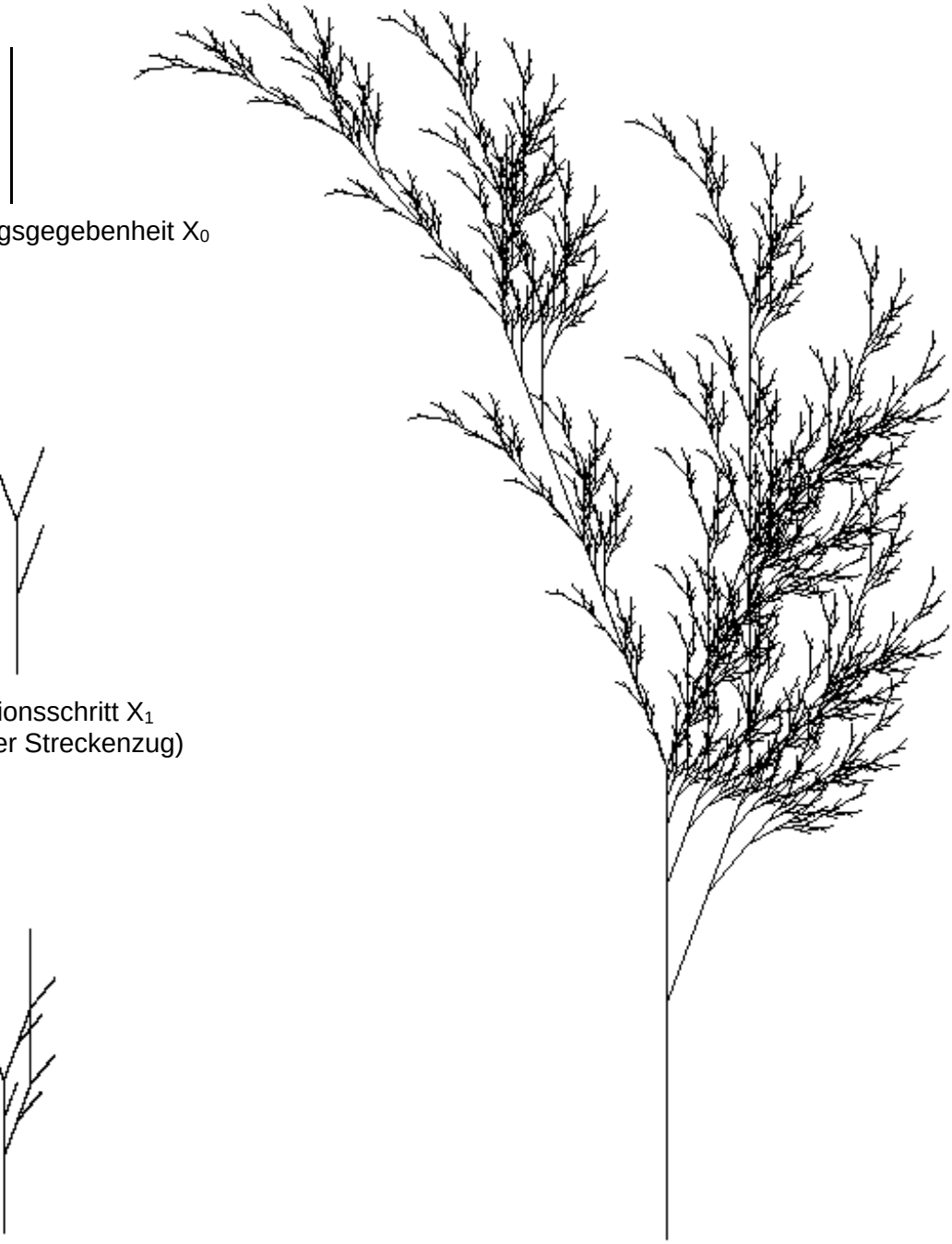
10.4 Deklarative Erzeugung synthetischer Bilder

Im Gegensatz zum prozeduralen Ansatz, der nur die Erzeugung relativ einfach aufgebauter Bilder erlaubt, lassen sich beim ***deklarativen Ansatz*** (siehe links oben in Abbildung 10.1) beliebig komplexe Bilder generieren.

Der Grundgedanke dieses Ansatzes ist, dass die Beschreibung der abzubildenden Szene von der Erzeugung des Bildes getrennt wird. Das Programm dient hier als ***virtuelle Kamera*** (aktiver Teil), mit dem die separat modellierte und abgespeicherte Szene (passiver Teil) aufgenommen wird. Beispielsweise kann eine komplexe Szene unabhängig von ihrer Darstellung erstellt und z.B. als VRML- oder X3D-Datei (**V**irtual **R**eality **M**odelling **L**anguage) abgelegt werden. Mit Hilfe eines Browser-Plugins kann diese Welt dann visualisiert und „begangen" werden.

Abschließend sei darauf hingewiesen, dass es bei anspruchsvolleren Anwendungen (wie z.B. der Erzeugung realistisch wirkender Szenendarstellungen) nicht ausreicht, lediglich die Objekt- bzw. Szenengeometrie abzuspeichern. Zusätzlich muss das ***Szenenmodell*** auch die Oberflächenbeschaffenheit der Objekte erfassen (***Reflektionsmodell***). Da das Aussehen einer Szene weiterhin stark von der verwendeten Beleuchtung abhängt, muss das ***Beleuchtungsmodell*** ebenfalls beschrieben werden: Art der Beleuchtung (z.B. Punktstrahler oder diffuse

Beleuchtung, Intensität und Position der Lichtquellen). Und auch die virtuelle Kamera zur Projektion der modellierten Szene auf die Bildebene kann separat mit Hilfe des ***Kameramodells*** erfasst werden: z.B. Position, Orientierung und Bild- oder Brennweite der Kamera.



(a) Ausgangsgegebenheit X_0

(b) 1. Iterationsschritt X_1
(ersetzender Streckenzug)

(c) 2. Iterationsschritt X_2

(d) Endergebnis nach acht Iterationsschritten
(vergrößerte Darstellung)

Abb. 10.5: Erzeugung einer fraktalen Pflanze mit Hilfe eines Ersetzungs-Systems.

11 Ausgabe digitaler Bilder

In diesem Kapitel sollen kurz die Prinzipien erläutert werden, wie graphische Daten auf den beiden gängigsten Ausgabemedien Bildschirm und Drucker ausgegeben werden.

11.1 Ausgabe digitaler Bilder auf einem Bildschirm

Um ein Bild auf einem Rasterdisplay ausgeben zu können, muss das Bild, falls es nicht schon als Rasterbild vorliegt (z.B. eingescanntes Bild), von der Bilderzeugungs-Komponente zuerst in die Raster-Repräsentation konvertiert und dann in den Bildspeicher (Frame Buffer) geschrieben werden; siehe Abbildung 11.1. Anschließend liest die Bilddarstellungs-Komponente das Bild aus dem Bildspeicher aus und gibt es auf dem Bildschirm aus. Bilderzeugungs- und Bilddarstellungs-Komponente sowie der Bildspeicher sind in der Regel auf der Graphik-Karte untergebracht.

Abb. 11.1: Hauptkomponenten eines Rasterdisplays

11.1.1 Bildspeicher (Frame Buffer)

Der ***Bildspeicher (Frame Buffer)*** ist Teil des in der Regel wesentlich umfangreicheren ***Graphik-Speichers***, der neben den eigentlichen Bildern noch der Zwischenspeicherung für z.B. Texturen und zahlreiche spezielle „Buffer“ dient.

Um Animationen flüssig und störungsfrei erzeugen und ausgeben zu können, muss das Beschreiben und Auslesen des Bildspeichers mit sehr hoher Geschwindigkeit erfolgen. Dabei könnte es zu einem ***Zugriffs-Konflikt*** kommen, wenn eine Speicherposition gleichzeitig beschrieben und ausgelesen werden soll. Um dies zu vermeiden, ist der Bildspeicher meist „mehrfach“ als ***Wechselspeicher*** ausgelegt. Bei diesem ***Double-*** oder sogar- ***Triple-Buffering*** wird das neue Bild in einen reservierten Bereich des Bildspeichers geschrieben, während beim Auslesen auf einen anderen Bildbereich (mit „älteren“, im vorangegangenen Schritt erzeugten Bilddaten) zugegriffen wird; wenn das neue Bild vollständig erzeugt ist, wird zwischen den Speicher-Bereichen umgeschaltet usw..

11.1.2 Bilderzeugungs-Komponente

Die Bilderzeugungskomponente muss die Bilder in gerasterter Form generieren und in den Bildspeicher schreiben.

Während diese Aufgabe früher komplett von der Host-CPU miterledigt wurde, wurden nach und nach immer mehr Berechnungs-Teile auf hochspezialisierte Teile der Graphik-Karte ausgelagert; z.B. Vektor-Raster-Konvertierung, Texture-Mapping, und „Transform- und Lighting“ (T&L: Transformation den Szene in die Sicht des Betrachters und Lichtberechnung für mehrere Lichtquellen).

Mittlerweile werden Graphik-Karten zusätzlich mit einer ständig wachsenden Zahl sogenannter ***Shader*** ausgestattet. Dabei handelt es sich um frei programmierbare Recheneinheiten, die alle parallel auf den Bilddaten arbeiten können; (Streaming; SIMD, d.h. **S**ingle **I**nstruction, **M**ultiple **D**ata). Um den Entwicklern eine komfortable Programmierung zu ermöglichen, wurden c-ähnliche Hochsprachen entwickelt. Diese Möglichkeiten werden i.d.R. am besten über ein unterstütztes API (Application Programming Interface) wie OpenGL oder DirectX genutzt.

Da die extreme und hochflexibel einsetzbare Rechenleistung der Shader nicht nur für Graphik-Anwendungen interessant ist, hat sich ***GPGPU*** (**G**eneral **P**urpose Computation on **G**raphic **P**rocessing **U**nit) als eigenständiges Gebiet entwickelt; (z.B. für beliebige Physik- und Meteorologie-Simulationen, Kryptographie, Molekularbiologie (beispielsweise Protein-Faltungen), usw.).

11.1.3 Bilddarstellungs-Komponente

Die Aufgabe der Bilddarstellungs-Komponente ist das Auslesen der Bildinformation aus dem Bildspeicher und die Aufbereitung und Weitergabe der Farbwerte je Bildposition an den Bildschirm. Falls der Bildspeicher dabei direkt die darzustellenden RGB-Tripel enthält, so müssen diese nur der entsprechenden Position des Bildschirms zugeordnet werden.

Wenn der Inhalt des Bildspeichers jedoch ein LUT-Bild ist (z.B. Grauwertbild), so muss der entsprechenden Bildschirmposition erst noch mit Hilfe der LUT ein RGB-Tripel zugeordnet werden; (der Eintrag im Bildspeicher ist hier ein Index in die LUT und dort stehen dann die gewünschten RGB-Tripel; siehe Abbildung 11.2).

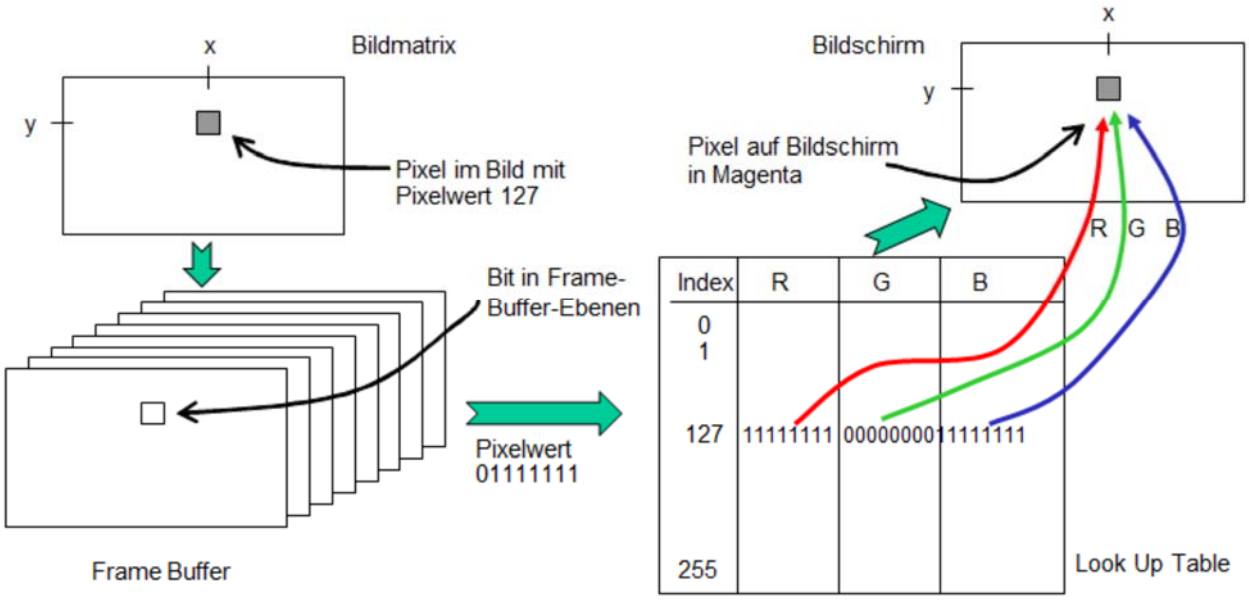


Abb. 11.2: Prinzipbild für die Verwendung einer Look-Up-Table (LUT)

11.1.4 Bildschirm

Die meisten Displays sind heute ***Flüssigkristall-Flachbildschirme*** (***LCD***s; **L**iquid **C**rystal **D**isplays mit TFTs; TFT siehe nächste Seite).

Abbildung 11.3 zeigt im Querschnitt den prinzipiellen Aufbau von TFT-LCDs. Die Hintergrundbeleuchtung wirkt als große, sehr helle und homogen leuchtende Fläche. Aus diesem Licht wird durch einen ersten Polarisationsfilter Licht einer Schwingungsebene herausgefiltert. Wenn dieses polarisierte Licht später auf einen zweiten, um 90 Grad gedrehten Polarisationsfilter trifft, so kann es diesen nur passieren, wenn die Schwingungsebene des Lichts zwischenzeitlich gedreht wurde. Diese Drehung erfolgt mit Hilfe der Flüssigkristalle, die zwischen den Polarisationsfiltern angebracht sind, und die durch Anlegen von Spannung ihre Ausrichtung ändern und damit das Licht „drehen“ (bzw. nicht „drehen“), so dass es den zweiten Polarisationsfilter passieren kann; andernfalls bleiben die zugehörigen Pixel dunkel.

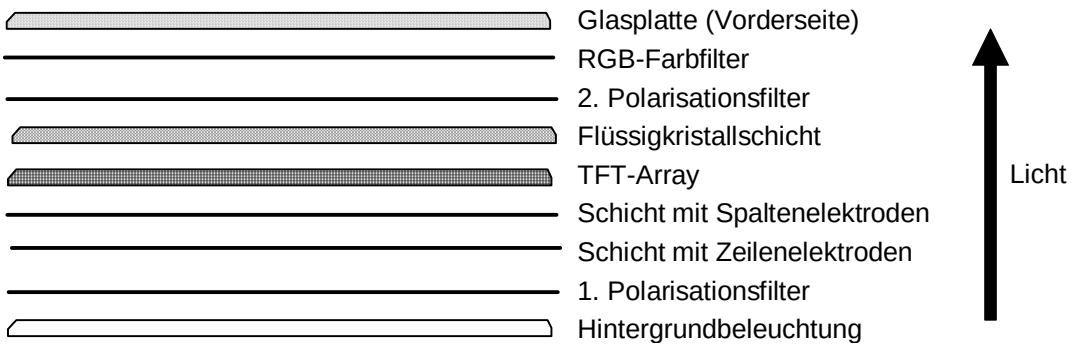


Abb. 11.3: Prinzipieller Aufbau eines TFT-LCDs im Querschnitt. (Der 2. Polarisations-Filter kann auch „vor“ der Glasplatte sitzen.)

Zur Farbwiedergabe besteht jedes Bildschirmpixel aus drei einzelnen "Farbpunkten" (***Triade***), die durch Verwendung von Farbfolien in Rot, Grün oder Blau leuchten. Jedem dieser drei Farbpunkte ist ein Flüssigkristall zur Helligkeitssteuerung zugeordnet.

Zur Ausgabe eines Bildes wird der darzustellende Inhalt des Bildspeichers an den Bildschirm übertragen. Die Ansteuerung der einzelnen Pixel geschieht mit Hilfe zeilenweise bzw. spaltenweise angeordneter Leitungen (Zeilen- bzw. Spalten-Elektroden). Um einen Farbpunkt gezielt ein- oder auszuschalten, wird eine schwache Spannung auf die Leitungen der entsprechenden Zeile und Spalte gelegt. Diese baut an dem Kreuzungspunkt der Leitungen, der ja dem zu beeinflussenden Pixel entspricht, ein stärkeres elektrisches Feld auf, das einen ***Dünnfilm-Transistor*** (***TFT;*** **T**hin **F**ilm **T**ransistor) schaltet. Dieser verstärkt das Feld um ein Vielfaches und richtet so das zugehörige Flüssigkristall aus, wodurch der „Licht-Durchlass“ gesteuert wird.

Da die Transistoren als Speicherzelle wirken, lässt die Strahlungsintensität nicht nach und der bei alten Röhrenmonitoren erforderliche Refresh des Bildinhaltes kann entfallen. Weitere Vorteile von LC-Displays sind der geringe Energieverbrauch, die hohe räumliche Auflösung, der hohe Kontrast, und der große Blickwinkel.

Jedes LC-Display besitzt eine sogenannte physikalische Standard-Auflösung, die durch die Anzahl der Bildschirmpixel festgelegt ist; (z.B. 1600 * 1200 bei UXGA). Wenn die Graphikkarte eine von dieser Standard-Auflösung abweichende Auflösung liefert, so wird diese in der Regel durch Interpolation auf eine Vollbildschirm-Darstellung umgerechnet. Dabei können z.B. Unschärfen entstehen, so dass eine optimale Bildqualität nicht mehr gewährleistet ist.

Da LC-Displays digitale Signale zur Ansteuerung der Pixel benötigen, muss die verwendete Graphikkarte einen digitalen Signalausgang besitzen; (z.B. ***DVI*** für Digital Visual Interface oder ***HDMI*** für High Definition Multimedia Interface).

An speziellen Monitoren können z.B. unter Verwendung von ***Shutterbrillen*** Stereobilder und Stereo-Bildfolgen betrachtet werden. Dazu werden die linken und rechten Bilder in schnellem Wechsel ausgegeben und die damit synchronisierte Shutterbrille ermöglicht abwechselnd jeweils nur dem linken bzw. rechten Auge den Blick auf den Monitor.

Neuere Spezialentwicklungen erlauben eine ***autostereoskopische Darstellung***, bei der zur Stereo-Wahrnehmung auf zusätzliche Hilfsmittel wie z.B. Shutter-Brillen verzichtet werden kann. Durch zusätzlich eingebaute spezielle Filterschichten - sogenannte ***Parallax-Barrieren*** - wird erreicht, dass das linke bzw. rechte Auge nur die Hälfte aller Pixel sieht: beispielsweise sieht das linke Auge nur die Bildspalten mit gerader und das rechte Auge nur die mit ungeradzahliger Spaltennummer. Die entsprechend aufbereiteten Bilder des Stereobildpaares können in diesem Falle natürlich nur mit halber horizontaler Auflösung dargestellt werden. Dies kann vermieden werden, wenn die Parallax-Barriere bei jedem Bildwechsel um ein Pixel nach links bzw. rechts verschoben wird, so dass jedes Auge nun durch den schnellen Wechsel dieser beiden „Halb-Bilder“ ein komplettes Bild sieht. Jetzt ist aber die zeitliche Auflösung halbiert, so dass eine hohe ***Bildwiederholfrequenz*** erforderlich ist.

Weitere Display-Technologien sind neben den ***Gasplasma-Displays***, zunehmend auch ***LED-Bildschirme*** (**L**ight **E**mitting **D**iodes). Die aktive Farbmatrix wird hier je Pixel durch drei farbige LEDs für Rot, Grün und Blau und meist noch ein weißes LED gebildet.

Außerdem gibt es noch sogenannte ***OLED***s (**O**rganic **L**ight **E**mitting **D**iodes), die für extrem dünne und flexible Folienbildschirme (z.B. ausrollbare Bildschirme) verwendet werden können.

Nachtrag: Bilder einscannen für die Bildschirmausgabe

Beim Einscannen von Bildern für die Bildschirmausgabe (z.B. für eine Web-Page) richtet sich die Scan-Auflösung nach der gewünschten absoluten Auflösung des Ausgabebildes. Da ein nachträgliches Skalieren des Bildes wegen der Qualitätsverluste möglichst vermieden werden sollte, empfiehlt es sich, den interessierenden Ausschnitt der Bildvorlage so abzutasten, dass dabei die gewünschte Auflösung möglichst genau erreicht wird. Die optimale Scan-Auflösung ergibt sich zu:

$$\text{Scan\_Auflösung (in dpi)} = \frac{\text{Zielbild\_Seitenlänge (in Pixel)} * 2.54}{\text{Quellbild\_Seitenlänge (in cm)}}$$

Wenn das abgetastete Bild Moiré-Muster aufweist (siehe Kapitel 10.2), so kann man das Bild entweder „glätten“ (siehe Kapitel 8.1), oder aber man tastet das Bild mit der doppelten Scannerauflösung ab und verkleinert es anschließend auf die halbe Größe.

11.2 Ausgabe digitaler Bilder auf einem Drucker

Im Gegensatz zu Plottern, die vektororientiert arbeiten und die hier nicht näher betrachtet werden, sind Drucker rasterorientiert. Die gängigsten Drucker sind heute Tintenstrahl- und Laserdrucker. Bei diesen wird die Vektor-Raster-Konvertierung durch einen integrierten Mikroprozessor durchgeführt. Ein Drucker muss alle an ihn übertragenen Daten speichern, verarbeiten und ausgeben können. Zu diesem Zweck muss der Druckerspeicher ausreichend groß bemessen sein.

Eine wesentliche Größe zur Kennzeichnung eines Druckers ist seine ***relative Auflösung***, die in ***dpi*** (**d**ots **p**er **i**nch) angegeben wird. Eine Angabe von beispielsweise 600 dpi bedeutet dabei z.B. für einen Schwarz-Weiß-Drucker, dass er auf einen Zoll (2.54 cm) gezielt 600 schwarze Rasterpünktchen nebeneinander setzen kann. Diese Auflösung gilt also nur für die Ausgabe binärer Information, wie z.B. schwarzem Text auf weißem Papier.

Zur Ausgabe von Grauwertbildern können unterschiedliche Graustufen beispielsweise durch eine mehr oder weniger dichte Anordnung von schwarzen Rasterpünktchen simuliert werden. Dazu werden z.B. m*n Punkte des Druckrasters zu einem ***Makropixel (Halbtonzelle)*** zusammengefasst. Färbt man dann beispielsweise in einem typischen Makropixel von 5*5 Pixeln keinen oder alle Punkte ein, so erhält man einen weißen bzw. schwarzen Bildpunkt; wenn man etwa 8 bzw. 16 Punkte einfärbt, so ergibt sich ein helles bzw. dunkles Grau.

Alternativ zu Makropixeln gibt es das leistungsfähigere ***Dithering (Fehler-Diffusion)***. Beim Dithering wird bei jedem Pixel einzeln entschieden, ob es gesetzt wird oder nicht. Dabei wird versucht, bzgl. einer Umgebung den Gesamtfehler, der durch das Setzen oder Nicht-Setzen des Pixels entsteht, zu minimieren.

Zur Ausgabe von Farbbildern werden die oben genannten Verfahren dahingehend erweitert, dass die Farben Cyan, Magenta, Gelb und Schwarz des subtraktiven CMY(K)-Farbmodells verwendet werden; siehe Kapitel 7.2.2.

Bei der Ausgabe von Grauwert- oder Farbbildern erhält man in der Regel also immer eine schlechtere Auflösung bzgl. der Farbe und/oder die räumliche in ***lpi*** (**l**ines **p**er **i**nch) angegebene ***effektive Auflösung*** ist schlechter als die in dpi angegebene relative Auflösung des Ausgabegerätes.

Nachtrag: Bilder einscannen für die Druckerausgabe

Beim Einscannen von Bildvorlagen für den späteren Druck richtet sich die einzustellende Scan-Auflösung nach der effektiven Auflösung des Druckers (in lpi) und dem Skalierungsfaktor, der sich aus dem Verhältnis zwischen der Quellbild- und der Zielbildgröße ergibt. Aus drucktechnischen Gründen wird empfohlen, noch einen Sicherheitsfaktor zu berücksichtigen, der meist zwischen 1.4 und 2 gewählt wird. Die optimale Scan-Auflösung ergibt sich zu:

$$\text{Scan\_Auflösung (in dpi)} = \frac{\text{effek.\_Druckauflösung (in lpi)} * \text{Zielbild\_Größe (in cm)} * \text{Sicherheitsfaktor}}{\text{Quellbild\_Größe (in cm)}}$$

Leider ist bei vielen Druckern die effektive Auflösung nicht angegeben und es ist auch nicht möglich, diese durch die Festlegung der Anzahl zu verwendender Grau- bzw. Farbabstufungen für das Zielbild zu beeinflussen. Die erreichbaren effektiven Auflösungen liegen in der Regel zwischen 100 und 200 lpi (Sicherheitsfaktor = 2.0): Letztlich helfen hier jedoch nur Probedrucke.

Literatur

Computer Graphik:

[Hug] Hughes J.F. et al., „Computer Graphics Principles and Practice“, Addison Wesley, 2013

[Nisch1] Nischwitz A. et al., "Computergrafik und Bildverarbeitung: Band I: Computergrafik: 1", Springer , 2013

[Gort] Gortler S. J., „Foundations of 3D Computer Graphics”; MIT Press; 2012

Graphikprogrammierung und Szenengraphen:

[Däßl] Däßler R., „Das Einsteigerseminar VRML“, Redline, 2002

[Kloss] Kloss J., Schickel P., „X3D – Programmieren interaktiver 3D-Anwendungen für das Internet“, Addison-Wesley, 2010

[Shr] Shreiner D. et al., „OpenGL Programming Guide: The Official Guide to Learning OpenGL “, Addison Wesley, 2013

oder als Download: http://it-ebooks.info/book/2138/

[Wrig] Wright R. S., Sweet M., „OpenGL Superbible“, Addison Wesley, 2013

Bildbearbeitung / Bildverarbeitung:

[Burg] Burger W., Burge M.J., „Principles of Digital Image Processing”, Springer, 2011

[Dem] Demant, C et al., „Industrielle Bildverarbeitung", Springer Verlag, 2011

[Gölk] Gölker K., „Fotobearbeitung und Bildgestaltung mit GIMP 2.8 ”, Dpunkt, 2012

[Gon] Gonzales R., Woods R., „Digital image Processing“, Pearson, 2011

[Hamm] Hammel M. J., „The Artists‘ Guide to the Gimp“,No Starch Press, 2012

[Nisch2] Nischwitz A. et al., „Computergrafik und Bildverarbeitung: Band II: Bildverarbeitung: 2", Springer , 2013

[Son] Sonka M. et al., „Image Processing, Analysis and Machine Vision“, Thomson Computer Press, 2008

Rasterbildkompression:

[Barn] Barnsley M. et al., „Bildkompression mit Fraktalen“, Vieweg Verlag, 1996

[Berg] Bergh J et al., „Wavelets: mit Anwendungen in Signal- und Bildverarbeitung", Springer, 2007

[Heyn] Heyna A. et al. „Datenformate im Medienbereich", Fachbuchverlag Leipzig im Carl Hanser Verlag, 2003

[Wiki] Wikipedia, „Grafikformat: Portable Network Graphics, …”, General Books, 2011

[Strutz] Strutz T., „Bilddatenkompression“, Vieweg+Teubner, 2009

[Reim] Reimers, U. (Hrsg.), „DVB - Digitale Fernsehtechnik“, Springer Verlag, 2008

4er-Nachbarschaft (N4) 8-6
8er-Nachbarschaft (N8) 8-6
absolute Auflösung 7-1
Abtasttheorem 10-3
additive Farbmischung 7-4
aktive Bildaufnahmesysteme 10-1
Aliasing 2-2, 7-2
Allgemeines Sweeping 3-11
Alpha-Kanal 7-5
angereicherte Realität (AR) 1-3
Anti-Aliasing 2-2, 7-2
Äquidensitenbild 8-5
AR (Augmented Reality) 1-3
Aspect Ratio 7-1
asymmetrische Kompressionsverf. 9-1
Attribute, graphische 3-2
audio-visuelles Objekt 9-14
Auflösung, absolute 7-1
Auflösung, effektive 11-4
Auflösung, Farbe 7-8
Auflösung, optische 10-3
Auflösung, photometrische 7-3
Auflösung, relative 11-4
Augmented Reality 1-3
Augmented Virtuality 1-3
äußere Kameraparameter 3-4, 3-5, 4-5
autostereoskopisch 11-3
AV (Augmented Virtuality) 1-3
Backward ray tracing 5-5
B-Bild 9-13
Beleuchtungsmodell 10-4
Biddatenformate 9-8
bidirektionale Prädiktion 9-13
Bildbearbeitung 1-3
Bildebene 10-1
Bildinterpretation 1-3,8-1
Bildkompression, fraktale 9-2
Bildkoordinatensystem 3-5
Bildmanipulation, geometrische 8-13
Bildmanipulation, photometrische 8-1
Bildoperator, lokaler 8-6
Bildpunkt 7-1
Bildsegmentierung 1-3,8-5, 8-11
Bildspeicher 11-1
Bildverarbeitung, digitale 1-3, 2-3, 8-1
Bildweite 10-1
Bildwiederholfrequenz 11-3
Bilineare Interpolation 5-2
bimodales Histogramm 8-12
Binärbild 8-5,8-12
Binarisierung 8-12
Blatt, Endknoten, terminaler Knoten 3-6
Blende 10-1
blockorientierte Kompressionsverf. 9-1
BMP 9-8
boundary representation, BREP 3-11
bounding boxes 5-6
Brennpunkt 3-5
BREP, boundary representation 3-11
CAD (Computer Aided Design) 1-4, 2-1
Cathode Ray Tube 11-2
CCD-Chip 10-2
cell decomposition, Zellzerlegung 3-10
Center of Interest, CoI 4-5
Center of Projection, COP 3-5
CGM (Computer Graphics Metafile) 9-14
Chaostheorie 1-2
Chernoff-Ikone 5-8
Chrominanz 7-6
CIM (Computer Integr. Manufact.) 1-4
Clipping 3-5
Clipping 3-5
Clipping-Ebene 3-5
Closing 8-10
CMY-Farbmodell 7-4
CMYK-Farbmodell 7-5
Codierung, hierarchische 9-1
Codierung, interframe 9-12
Codierung, intraframe 9-12
Codierung, skalierbare 9-1
CoI, Center of Interest 4-5
Collage-Theorem 9-3
Computer Aided Design (CAD) 1-4, 2-1
Computer Geometrie 1-4
Computer Graphics 1-1, 2-3
Computer Graphics Metafile (CGM) 9-14
Computer Graphik 1-1, 2-3
Computer Integr. Manufact. (CIM) 1-4
Computer Vision 1-4,8-1
Constant Shading 5-4
Constructive Solid Geometry, CSG 3-11
COP, Center of Projection 3-5
CRT (Cathode Ray Tube) 11-2
CSG, Constructive Solid Geometry 3-11
Darstellungsverhältnis 7-1
Data Exchange Format (DXF) 9-16
Datenstrukturen, hierarchische 2-3
Datenverarbeitung, graphische 1-1
DCT (Diskrete Cosinus Transform.) 9-3
Decodierung, progressive 9-1
Decodierung, sequentielle 9-1
deklarativer Ansatz zur Bilderzeugung 10-4
Differenzbild 9-13,8-11
digitale Bildverarbeitung 1-3 ,2-3, 8-1
digitales Bild 7-1,10-1
Digitalisierungs-Modell 10-2
Dilatation 8-10
Diskrete Cosinus Transformation 9-3,9-10
Dithering 11-4
dots per inch, dpi 11-4
Double-Buffering 11-1
dpi, dots per inch 11-4
Drahtmodell, Kantenmodell 3-7
Drehung, Rotation 3-3, 6-4,6-5,
Drehung, Rotation 6-7, 6-8
Dünnfilm-Transistoren 11-3
DVI 11-3
DXF (Data Exchange Format) 9-16
Ecke, Eckpunkt 3-7, 4-2
Eckennormale 5-4
Eckpunkt, Ecke 4-2
effektive Auflösung 11-4
Eigenschaften, geometrische 3-6, 3-7
Eigenschaften, topologische 3-6, 3-7
eindimensionale Kompressionsverf. 9-1
Elementar-Transformationen 6-3, 6-6

Encapsulated PostScript (EPS) 9-15
Endknoten, terminaler Knoten, Blatt 3-6
Entkopplung von Transformationen 4-3
Entropie 8-1
Entropie-Kodierung 9-1
Enumerationsverfahren 3-9
EPS (Encapsulated PostScript) 9-15
Erosion 8-10
erweiterte Virtualität (AV) 1-3
Facette 3-8, 5-3
Falsch-Farbbild 7-8
Faltung 8-6
Farbauflösung 7-8
Farbmischung, additive 7-4
Farbmischung, subtraktive 7-4
Farbmodell, CMY 7-4
Farbmodell, CMYK 7-5
Farbmodell, HSV 7-6
Farbmodell, RGB 7-4
Farbmodell, RGB(A) 7-5
Farbmodell, YUV 7-6
Farbsättigung 7-6
Farbtabelle (LUT) 7-8,11-2
Farbton 7-6
Fehler-Diffusion 11-4
Festkörperdarstellung 3-11
FIF (Fractal Image Format) 9-9
Fixpunkt 6-4
Fixpunkt-Bedingung 6-6
Flachbettscanner 10-2
Flachbildschirme 11-2
Flächenbegrenzungsdarstellung 3-11
Flächenmodell, surface model 3-8
Flächennormale 3-9, 5-4
flächenorientierte Segmentierung 8-11
Flat shading 5-4
Flüssigkristall-Flachbildschirme 11-2
Forward ray tracing 5-5
Fractal Compressed Image (FCI) 9-10
Fractal Image Format (FIF) 9-9
Fraktal 10-4
fraktale Bildkompression 9-2,9-9
fraktale Geometrie 1-2
frame buffer 11-1
Frustum 3-5, 4-5
Gammakorrektur 8-4
Gasplasma-Display 11-3
General Purpose Comp. on GPU 11-1
Geometrie, fraktale 1-2
geometrische Bildmanipulation 8-13
geometrische Eigenschaften 3-6, 3-7
geometrisches Modell 3-6
Gerade 6-2
gerichteter Graph 3-6
Gesamttransformation 6-5
gewichteter Mittelwert-Operator 8-7
GIF 9-8
Glättung 8-7
globale Kompressionsverfahren 9-1
globales Verfahren (Rendering) 5-3
Glyphen 5-7
Gouraud shading 5-4
GPGPU 11-1
Graph, gerichteter 3-6
Graph, zyklenfreier, schleifenfreier 3-6
graphical data processing 1-1
Graphical User Interface (GUI) 1-1
Graphik-Speicher 11-1
graphische Attribute 3-2
Graphische Datenverarbeitung 1-1
graphische Primitive 4-1
Graukeil 7-4
Graustufenauflösung 7-3
Grautonbild 7-3
Grauwertbild 7-3
Grauwertgebirge 7-4
Grauwertmatrix 7-4
Grauwerttransformation, lineare 8-2
Größenänderung, Skalierung 6-3,6-4, 6-7
GUI (Graphical User Interface) 1-1
Halbtonbild 7-3
Halbtonzelle 11-4
HDMI 11-3
Helligkeit 7-6
hierarchische Codierung 9-1
hierarchische Datenstrukturen 2-3
Hintereinanderausführung (Trafos) 6-5
Histogramm 8-1,8-12
homogene Komponente 6-4
homogene Koordinaten 6-4, 6-7
HSV-Farbmodell 7-6
Huffman-Codierung 9-2,9-5
I-Bild 9-13
IGES (Initial Graph. Exchange Spec.) 9-15
Ikonen 5-7
immediate mode 4-1
indirektes Verfahren 8-15
Initial Graph. Exchange Spec. (IGES) 9-15
innere Kameraparameter 3-4, 3-5, 4-5
Intensität 7-6
Interframe-Codierung 9-12
interpolierende Scan-Auflösung 10-4
Intraframe-Codierung 9-12
irrelevanzreduzierende Verfahren 9-1
JPEG 9-2,9-4,9-10
JPEG 2000 9-11
Kamera, virtuelle 10-4
Kamerakoordinatensystem 3-4
Kameramodell 10-5
Kameramodell der Lochkamera 10-1
Kameraparameter, innere, äußere 3-4, 3-5, 4-5
Kante 3-7, 8-1
Kantendetektor 8-7
Kanten-Kombinationsfilter 8-8
Kantenmodell, Drahtmodell 3-7
kartesisches Koordinatensystem 3-1
Kathodenstrahl-Bildröhre 11-2
Klassifizierung 1-3,8-1,
Kompressionsverfahren, asymmetr. 9-1
Kompressionsverfahren, blockorient. 9-1
Kompressionsverfahren, eindimens. 9-1
Kompressionsverfahren, globale 9-1
Kompressionsverfahren, symmetr. 9-1
Kompressionsverfahren, zweidimens. 9-1
Konstruieren, Rechnergestütztes 1-4
Kontrast 8-1

Konturhervorhebungs-Operator 8-9
konturorientierte Segmentierung 8-11
Koordinaten, homogene 6-4, 6-6
Koordinatensystem 3-1, 7-1
Koordinatensystem, kartesisches 3-1
Koordinatensystem, körpereigenes 3-3
Koordinatensysteme, Bildverarbeitung 7-1
Koordinatensysteme, graphische DV 7-1
Koordinatentransformation 6-3
körpereigenes Koordinatensystem 3-3
Kugelkoordinaten 3-3
Laplace-Operator 8-8
Lauflängencodierung 9-2,9-5,9-8
LCD 11-2
LED-Bildschirm 11-3
Lichtquelle, primäre bzw. sekundäre 5-3
Light Emitting Diodes 11-3
lineare Interpolation 5-1
Liquid Cristal Display 11-2
Lochkamera (Kameramodell) 10-1
lokaler Bildoperator 8-6
lokales Verfahren (Rendering) 5-3
Luminanz 7-6
LUT (Look Up Table) 7-8,11-2
LZW-Codierung 9-2
Makropixel 11-4
Markierungsverfahren, rekursives 8-13
Matrix-Schreibweise 6-2
Matrix-Stack 4-4
Median-Operator 8-9
mehrdimensionale Daten 5-8
Mengenoperationen 3-11
Meta-Daten 9-12,9-14
Mittelwert-Operator, gewichteter 8-7
Mixed Reality (MR) 1-2
MNG 9-9
Modell, geometrisches 3-6
Modellierungskoordinatensystem 3-2
Moiré-Muster 10-3
Morphing 8-13
MPEG 9-12
MR (Mixed Reality) 1-2
Multimedia 1-1
multivariate Daten 5-8
Nachbarschaft, 4er (N4) 8-6
Nachbarschaft, 8er (N8) 8-6
Objekt, audio-visuelles 9-14
Objekthierarchie 3-4
OLED 11-3
OpenGL 4-1
Opening 8-10
Operator, lokaler 8-6
Operator, Punkt- 8-2
optische Auflösung 10-3
Organic Light Emitting Diodes 11-3
orthographische, Projektion 4-5
Parallax-Barriere 11-3
Parameterform, Matrix-Schreibweise 6-2
Parameterform, vektoriell 6-1
Parametrisierung, primitive instancing 3-10
passive Bildaufnahmesysteme 10-1
P-Bild 9-13
PDF (Portable Document Format) 9-15
perspektivische, Projektion 4-5
Phong shading 5-5
photometrische Auflösung 7-3
photometrische Bildmanipulation 8-1
Photo-Sensorelemente 10-2
Pixel 2-2, 7-1
Pixelwert 7-8
PNG 9-9
Polarkoordinaten 3-2
Portable Document Format (PDF) 9-15
PostScript 9-15
Prädiktion 9-13
Prädiktion, bidirektional 9-13
Prädiktion, unidirektional 9-13
präzidiertes Bild 9-13
primäre Lichtquelle 5-3
Primärstrahlen 5-5
primitive instancing, Parametrisierung 3-10
Primitive, graphische 4-1
Profilkörper, Translationskörper 3-10
progressive Decodierung 9-1
Projektion 6-5
Projektion 5-5, 4-5
Projektion, orthographische 4-5
Projektion, perspektivische 4-5
prozeduraler Ansatz zur Bilderzeug. 10-4
Pseudo-Farbbild 7-8
Punkt 6-1
Punkt-Operator 8-2
Quantisierung 7-8,9-5,9-7,
Quantisierung 10-2
Radiosity 5-6
Rangfolge-Operator 8-9
Repräsentation 2-2
Rastertechnik 7-2
Rasterung 10-2
Raster-Vektor-Konvertierung 2-2, 7-2
Raumzellen, Voxel 3-10
RAW-Daten 10-2
Ray Casting 5-6
ray tracing (Strahlverfolgung) 5-5
Ray Tracing, rekursives 5-6
Rechnergestütztes Konstruieren 1-4
rechnerinterne Darstellung (RID) 3-7
redundanzreduzierende Verfahren 9-1
Reflektionsmodell 10-4
Region 8-11
Rekonstruktion, verlustbehaftete 9-1
Rekonstruktion, verlustfreie 9-1
rekursives Markierungsverfahren 8-13
rekursives Ray Tracing 5-6
relative Auflösung 11-4
Relief-Filter 8-8
Renderer 5-2
Rendering Pipeline 5-2
Rendern; Rendering 3-1, 3-4, 5-2
retained mode 4-1
RGB(A)-Farbmodell 7-5
RGB-Farbmodell 7-4
RID (rechnerinterne Darstellung) 3-7
RLE (Run Length Encoding) 9-2,9-5,9-8
root, Wurzel 3-6
Rotation, Drehung 3-3, 6-4,6-5,

Rotation, Drehung 6-7, 6-8
Rotationskörper 3-10
Run Length Encoding (RLE) 9-2,9-5,9-8
Scalable Vector Graphics (SVG) 9-15
Scan-Auflösung, interpolierende 10-4
Scanner 10-2
Schattierungs-Verfahren (shading Verf.) 5-3
schleifenfreier, zyklenfreier Graph 3-6
Schwarz-Weiß-Bilder 7-3
Scientific Visualization 1-2
Segmentation 8-11
Segmentierung, flächenorientiert 8-11
Segmentierung, konturorientiert 8-11
Sehstrahl 10-1
sekundäre Lichtquelle 5-3
Sekundärstrahlen 5-5
Selbstähnlichkeit 9-2
Sensoren 10-1
sequentielle Decodierung 9-1
Shader 11-3
Shader 11-1
shading Verfahren (Schattierungs-Verf.) 5-3
Shannon/Nyquist-Theorem 10-3
skalierbare Codierung 9-1
Skalierung, Größenänderung 6-3, 6-4, 6-7
solid model, Volumenmodell 3-9
Spannweiten-Operator 8-11
spatial occupancy matrix 3-9
Spiegelung 6-5
Standard for Exchange of … (STEP) 9-16
STEP (Standard for Exchange of …) 9-16
Strahl 6-1
Strahlverfolgung (ray tracing) 5-5
Strecke 6-1
subtraktive Farbmischung 7-4
surface model, Flächenmodell 3-8
SVG (Scalable Vector Graphics) 9-15
Sweeping 3-10
Sweeping, allgemeines 3-11
symmetrische Kompressionsverfahren 9-1
Szene 3-4
Szenengraph 2-3, 3-6, 4-6
Szenenmodell 10-4
technische Zeichnung 3-6
Teilefamilien, Varianten 3-10
Teilmodell 3-4
terminaler Knoten, Endknoten, Blatt 3-6
TFT-Bildschirme 11-3
Thin Film Transistor 11-3
topologische Eigenschaften 3-6, 3-7
Transformation 3-4, 4-2, 6-3
Transformationen, Entkopplung von 4-3
Transformationshierarchie 4-2
Transformationsmatrix 4-4
Translation, Verschiebung 3-3, 6-3,
Translation, Verschiebung 6-4, 6-7
Translationskörper, Profilkörper 3-10
Treppeneffekt 2-2
Triade 11-3
Triple-Buffering 11-1
True-Color 7-8
Umlaufsinn 3-9
Unbunt-Gerade 7-4
unidirektionale Prädiktion 9-13
Unterabtastung 10-3
Varianten, Teilefamilien 3-10
Vector Markup Language (VML) 9-15
vektorielle Schreibweise 6-1
Vektor-Raster-Konvertierung 2-2, 3-5, 7-2
Vektor-Raster-Konvertierung 11-1, 11-4
Vektortechnik 7-2
Verfahren, indirektes 8-15
Verfahren, irrelevanzreduzierende 9-1
Verfahren, redundanzreduzierende 9-1
Vergrößerung (Bild), ganzzahlige 8-14
Verkettung von Transformationen 6-5
Verkleinerung (Bild), ganzzahlige 8-14
verlustbehaftete Rekonstruktion 9-1
verlustfreie Rekonstruktion 9-1
vermischte Realität (MR) 1-2
Verschiebevektoren 9-13
Verschiebung, Translation 3-3, 6-3, 6-4,
Verschiebung, Translation 6-7
Vertex 4-2
View 3-5
Virtual Reality (VR) 1-2, 2-1
Virtual Reality Modeling Language, VRML 4-6
virtuelle Kamera 3-4, 10-4
Virtuelle Realität (VR) 1-2, 2-1
Visualisierung 5-1,5-7
Visualisierung, wissenschaftliche 1-2
Visualization, Scientific 1-2
VML (Vector Markup Language) 9-15
Volumenmodell, solid model 3-9
Voxel, Raumzellen, 3-10
VR (Virtuelle Realität) 1-2, 2-1
VRML, Virtual Reality Modeling Language 4-6
Warping 8-13
Wavelet 9-7,9-11
Wavelet Image Format (WI) 9-11
Wavelet-Basis 9-8
Wavelet-Koeffizienten 9-6
Wavelet-Transformation 9-6,9-11
Wechselspeicher 11-1
Weichzeichnung 8-7
Weltkoordinatensystem 3-4
WI (Wavelet Image Format) 9-11
wire frame model, Drahtmodell 3-7
wissenschaftliche Visualisierung 1-2
Wurzel, root 3-6
X3D 4-6
YUV-Farbmodell 7-6,9-11
Zeichnung, technische 3-6
Zellzerlegung, cell decomposition 3-10
ZHK (Zusammenhangskomponente) 8-12
Zugriffs-Konflikt 11-1
Zusammenhangskomponente (ZHK) 8-12
zweidimensionale Kompressionsverf. 9-1
zyklenfreier, schleifenfreier Graph 3-6
Zylinderkoordinaten 3-3