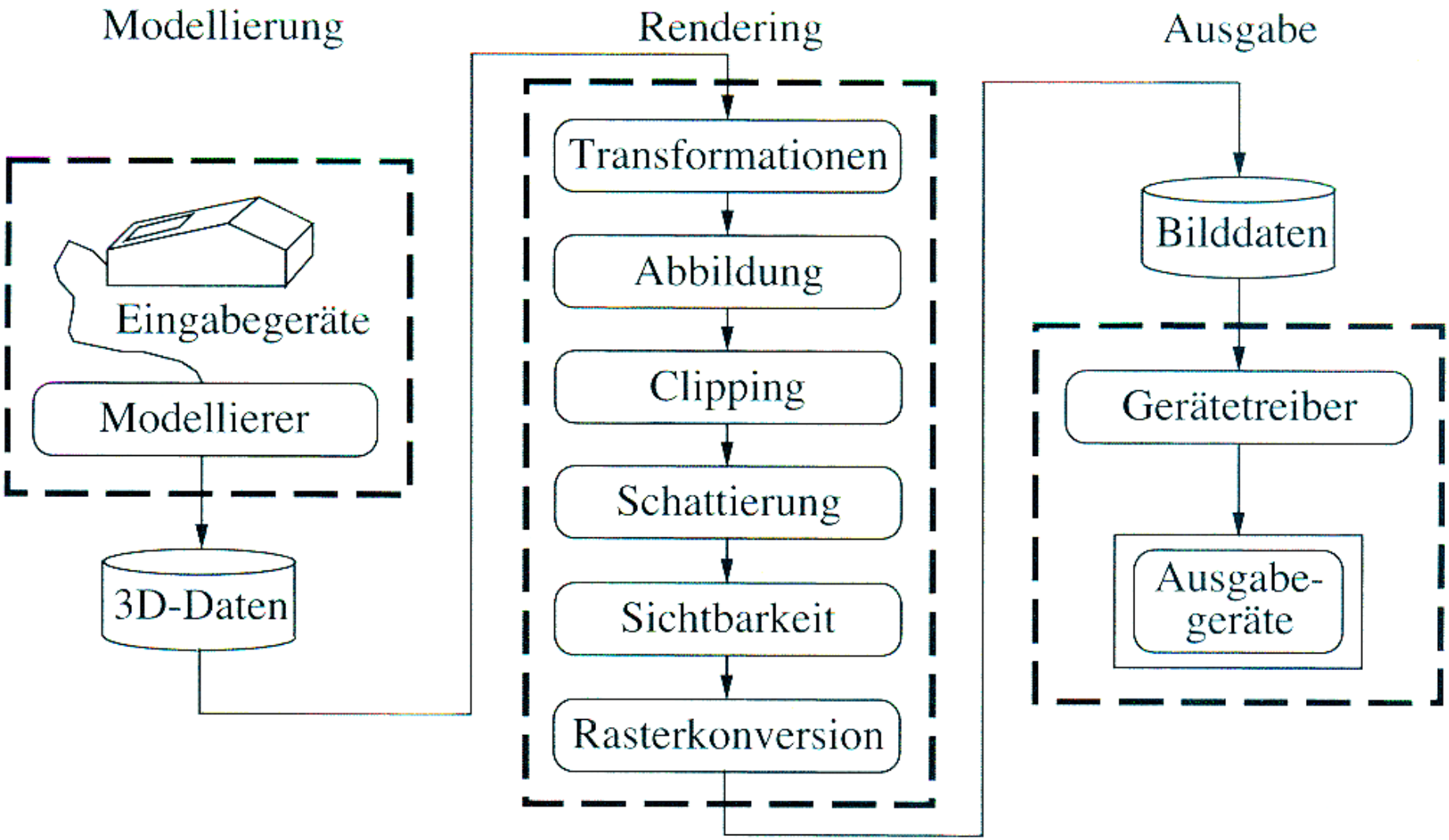
Grafische Datenverarbeitung

Allgemeines

- Grafische Datenverarbeitung umfasst alle Bereiche, in denen am Rechner Bilder oder Bildinformationen erstellt oder verarbeitet werden
- Unterschieden werden kann zwischen der
  - generativen grafischen Datenverarbeitung zur Erstellung und Manipulation von Bildern
  - Bildverarbeitung zur Verbesserung von Bildern und der
  - Mustererkennung zur Extraktion von Informationen aus einem Bild
- Die Trennung zwischen den Bereichen der grafischen Datenverarbeitung ist häufig nicht ganz scharf, da für manche Anwendungsfälle Funktionalität eines anderen Bereichs erforderlich ist
- Im ingenieurwissenschaftlichen Kontext sind die generativen grafischen Verfahren und die Mustererkennung von besonderer Bedeutung, erstere im Bereich CAD, letztere im Bereich Prozessüberwachung und Qualitätssicherung
- Generell ist die grafische Datenverarbeitung ein großer Wirtschaftszweig in der IT von medizinischen Anwendungen über geographische Informationssystems bis hin zu Computerspielen

Ablauf bei der grafischen Datenverarbeitung

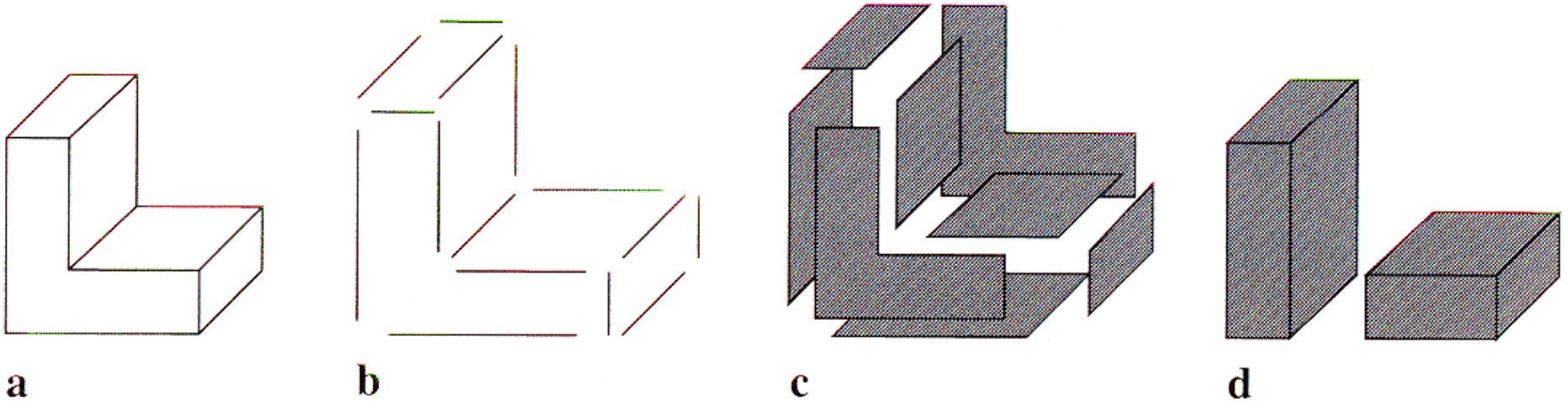


Quelle: Rechenberg/Pomberger

Modellarten

- Geometrische Modellierung beschreibt den Vorgang zur Erstellung rechnerinterner geometrischer Objekte
- Das rechnerinterne Modell wird mit Hilfe expliziter Datenstrukturen beschrieben, die entsprechende Modellierungselemente bereit stellen
- Für 2D-Modelle ist das lineale Modell das wohl gebräuchlichste. Hierbei wird der geometrische Körper durch Punkte und Linien beschrieben. Typische Elemente dabei sind Punkt, Linie, Kreis, Kreisbogen, Buchstabe, Ellipse, Spline usw.
- Sollen darüber hinaus auch Flächen im 2D-Modell beschrieben werden, kommen dafür die arealen Modelle zum Einsatz. Hierbei sind Linien (als Berandungskurven), Punkte (als Eckpunkte einer Fläche), Pixelbilder oder selbstdefinierte Muster als linealen Modellen zur Füllung verwendet
- Im 3D-Bereich kann im einfachen Fall das 2D-lineale Modell in den Raum als Drahtmodell übertragen werden. Darüber hinaus kann zwischen Volumen- und Flächemodellen unterschieden werden

Repräsentation dreidimensionaler Objekte

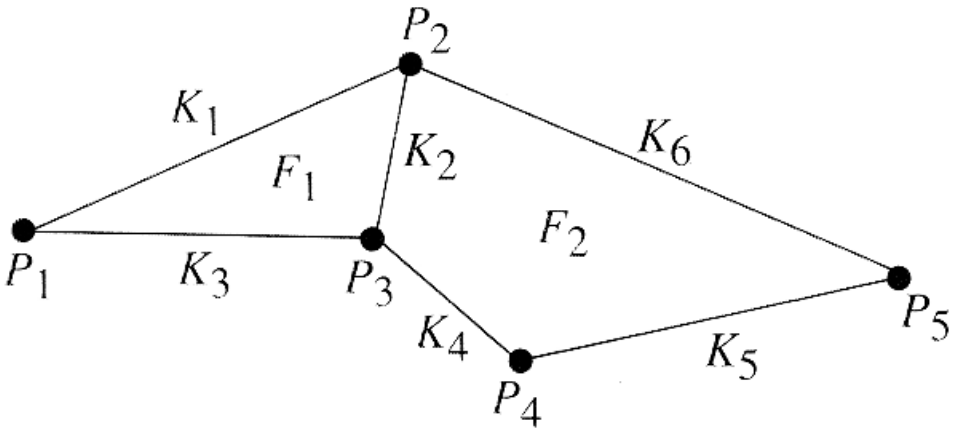


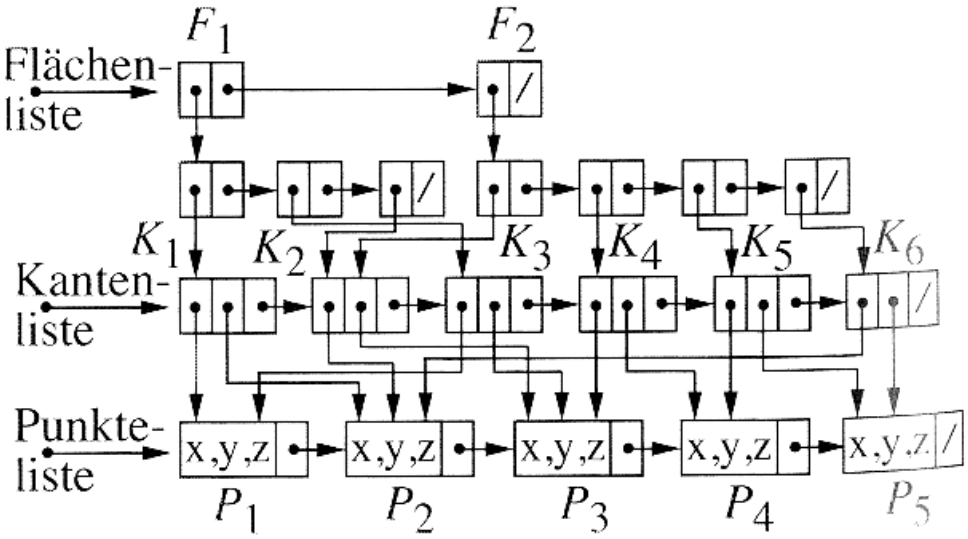
a) das reale Objekt
b) Drahtmodell
c) Flächenmodell
d) Volumenmodell

Quelle: Rechenberg/Pomberger

Abbildung einer Polygonfläche in einer B-Rep-Liste

- Die Fläche wird über die Kanten abgebildet, dabei enthält eine Fläche die Verweise auf die Kantenobjekte aus der Kantenliste
- In der Kantenliste werden Verweise auf die Punktobjekte der Punkteliste gehalten.
- Ähnliche Strukturen finden sich bspw. im vrml-Format

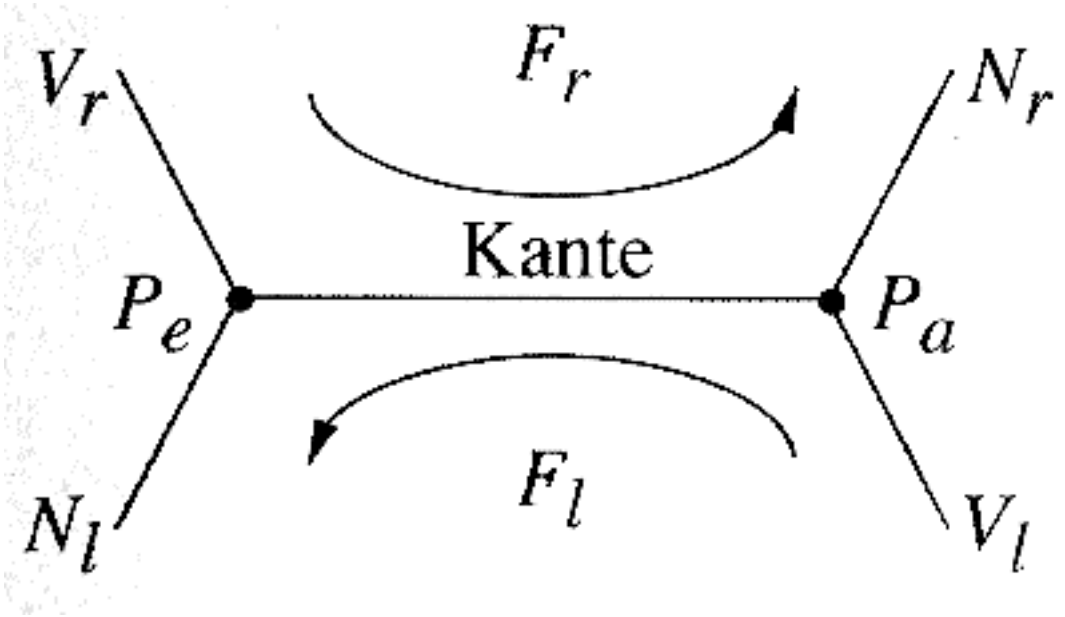




Quelle: Rechenberg/Pomberger

Die Winged-Edge Datenstruktur

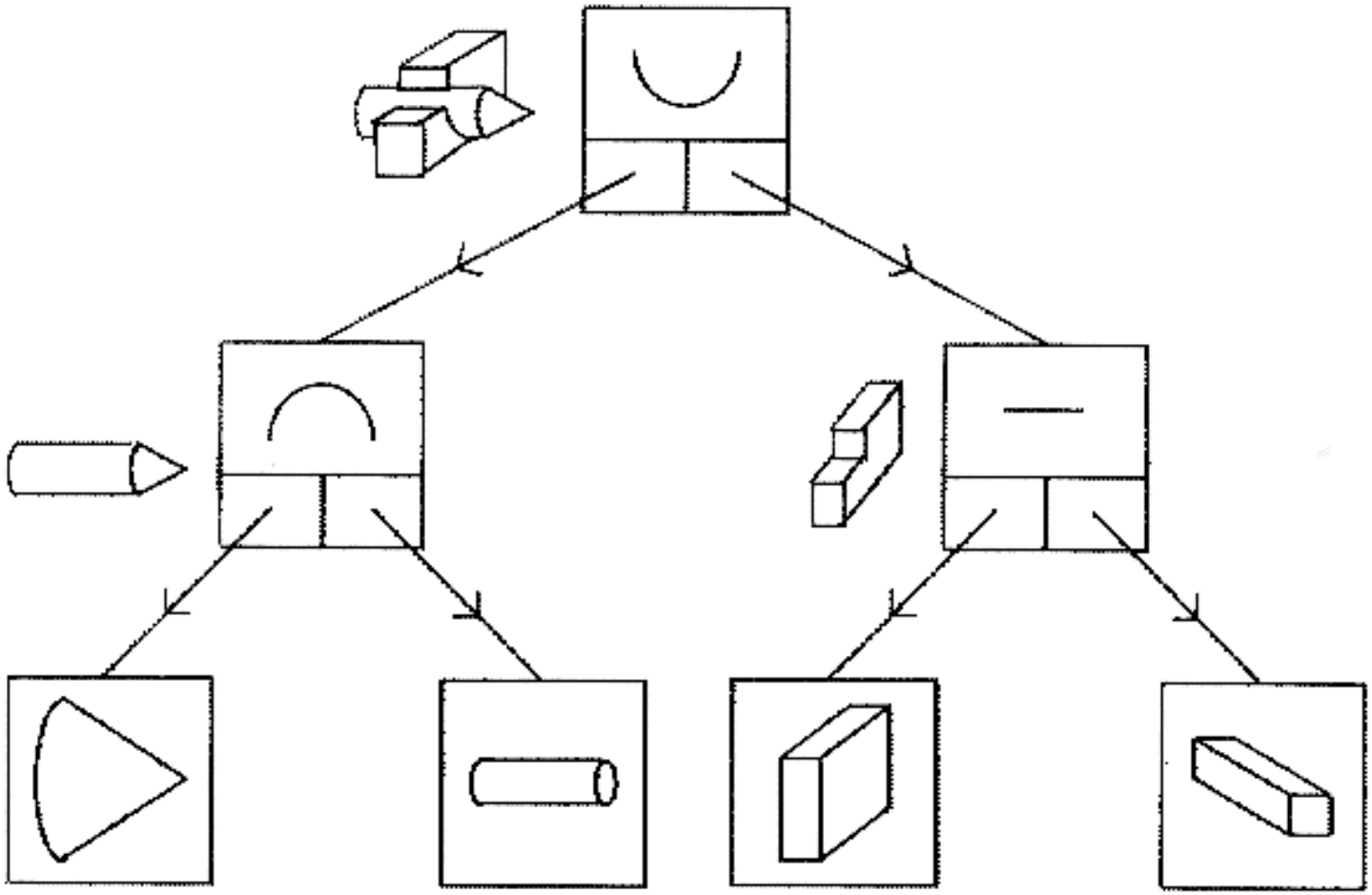
- Zentrales Element ist die Kante, jede Kante verweist auf die Punkte, die sie beschreiben
- Darüber hinaus hat eine Flächen-Kante einen Verweis auf die Vorgänger- und Nachfolgerkante. Die Kante P_e-P_a also auf den Vorgänger V_r und den Nachfolger N_r, die Kante P_a-P_e auf den Vorgänger V_l und den Nachfolger N_l



Quelle: Rechenberg/Pomberger

Ein CSG-Baum

- Der Volumenkörper wird durch boole'sche Operationen primitiver Volumina erzeugt



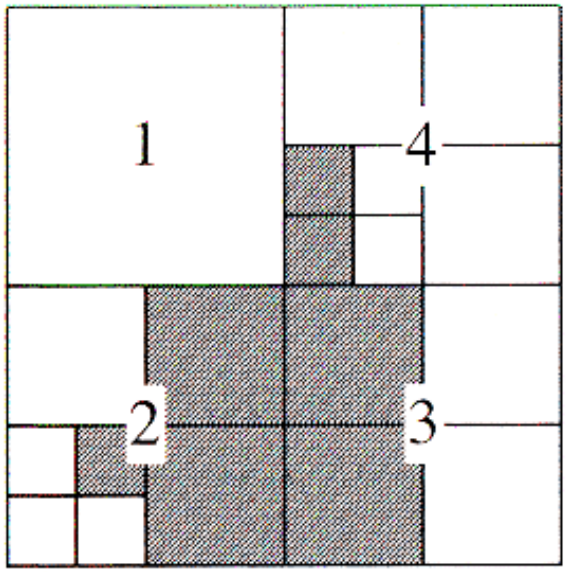
Quelle: Rechenberg/Pomberger

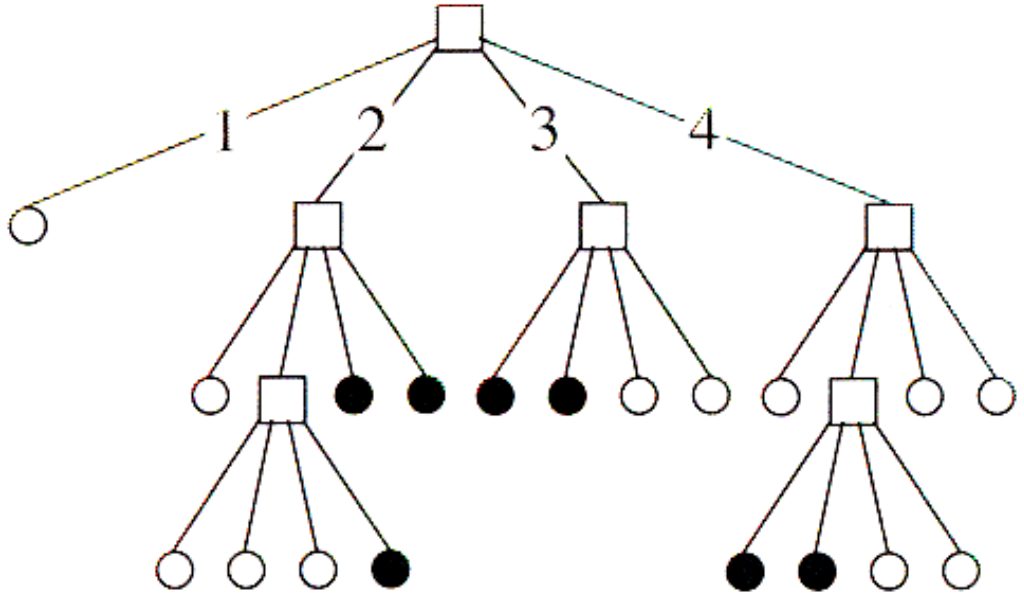
Voxel

- Analog zum Pixel (picture element), beschreibt ein Voxel einen Würfel in einem diskreten Raumstrukturgitter mit zugeordneten Daten (z.B. Dichtedaten)
- Mit einem derartigen Voxelmodell lassen sich beispielsweise medizinische Daten aus Ultraschall oder Computertomographie abbilden, bei denen die zu untersuchende Geometrie nach einem festen Raster abgetastet wird und die Daten dem entsprechenden Voxel zugeordnet werden können
- Eine generierende, manuelle Modellierung von Voxelmodellen ist wie die manuelle Erstellung von Pixelbildern unmöglich oder zumindest sehr aufwendig
- Voxelmodelle könnten allerdings aus Modellierern exportiert und anderen Prozessen zur Verfügung gestellt werden
- Ähnlich wie bei Pixeln sind verlustfreie und verlustbehaftete (weil vereinfachende und zusammenfassende) Formate für die Speicherung denkbar

Abbildung mit Quadtrees und Octrees

- Beim Quadtree (2D) oder Octree (3D) werden Objekte durch Quadrate beim Quadtree und Würfel beim Octree appoximativ dargestellt. Ausgehend von einem das gesamte Objekt umschließenden Quadrat oder Würfel wird solange weiter unterteilt, bis die im einzelnen Segment enthaltene Information eindeutig ist (innerhalb vordefinierter Grenzen)
- Ergänzende Verfahren mit anderen Unterteilungen sind ebenfalls verfügbar (z.B. Bintree mit jeweils einer Halbierung in einer Dimension, nicht parallel wie beim Octree oder Quadtree)

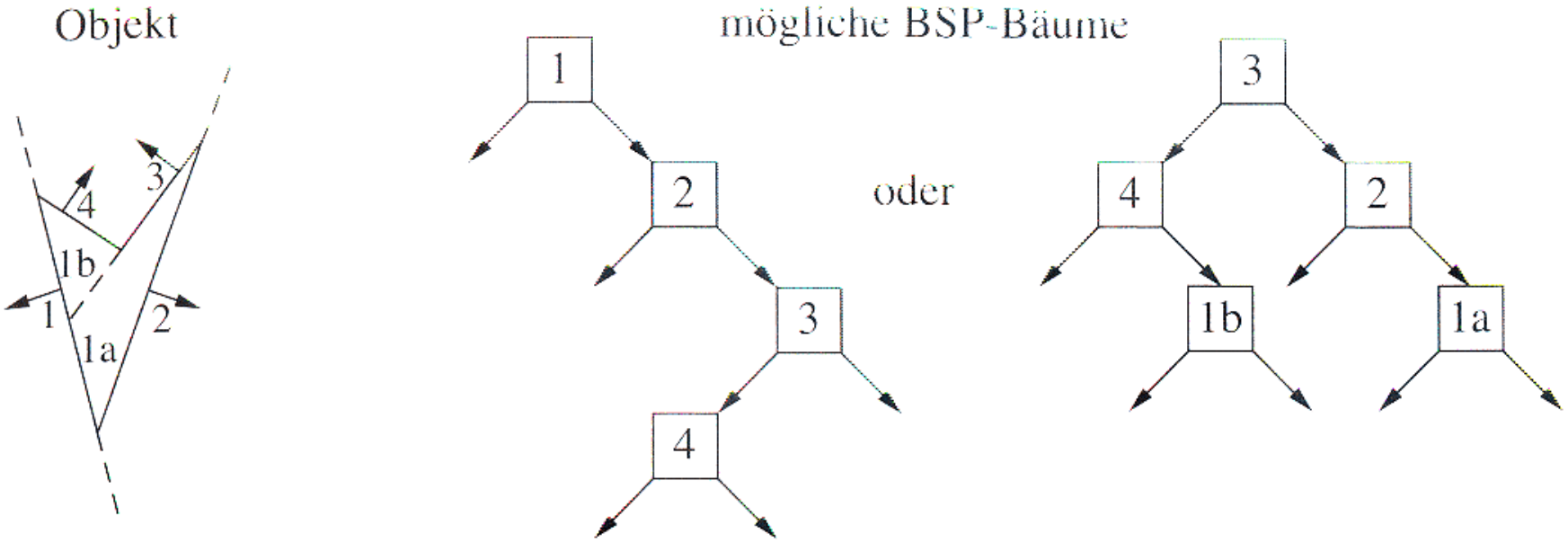




Quelle: Rechenberg/Pomberger

BSP-Baum im 2D-Raum

- Ein k-d-Tree erweitert den BinTree um die Möglichkeit der achsparallelen Verschiebung der Teilungsebene
- BSP (binary space partitioning) sind eine Verallgemeinerung der k-d-Trees auf nicht achparallele Unterteilungsflächen. Dabei wird das Objekt wie gehabt nach seiner Lage bzgl. der Teilungsfläche eingeordnet. Damit lassen sich bspw. Polygone nach ihrer Lage ordnen, was für den Rendering-Prozess vorteilhaft sein kann. Dabei ist das Element des BSP-Baumes kein Voxel mehr

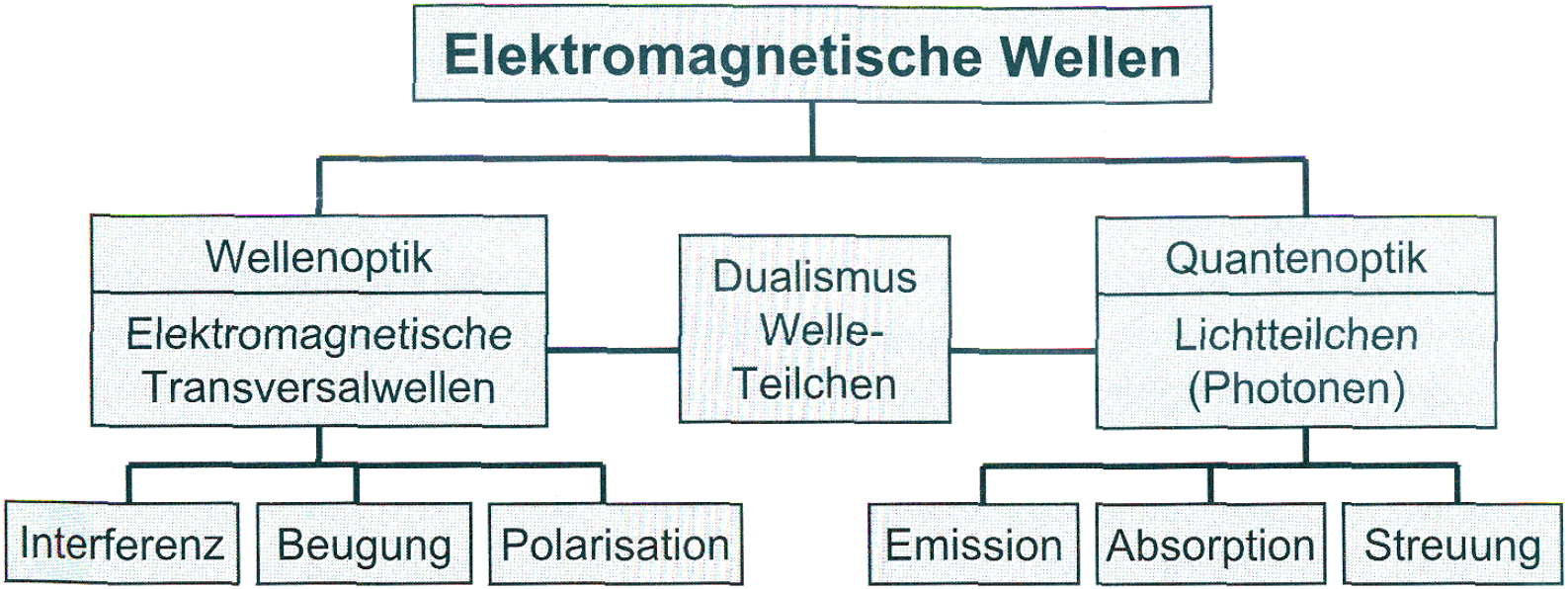


Quelle: Rechenberg/Pomberger

Clipping

- Unter Clipping wird die Beschränkung des sichtbaren Bereichs verstanden. Im 2D ist dies in der Regel ein Rechteck, manche Applikationen nutzen, z.B. für das Clipping bestimmter Bereiche auch Polygonzüge
- Im 3D ist der sichtbare Bereich bei einer Parallelprojektion ein Quader, bei einer perspektivischen Projektion eine Pyramide
- Das Clipping erfolgt häufig auf der Grafikkarte, durch zusätzliche Unterstützung aus der Software heraus können aber die Hardware-Verfahren beschleunigt werden, da nicht sichtbare Bereiche von vornherein aus der Betrachtung herausgelassen werden. Die dabei eingesetzten Verfahren werden als Culling bezeichnet und spielen insbesondere im Bereich der Virtuellen Realität eine Rolle.

Die physikalische Optik

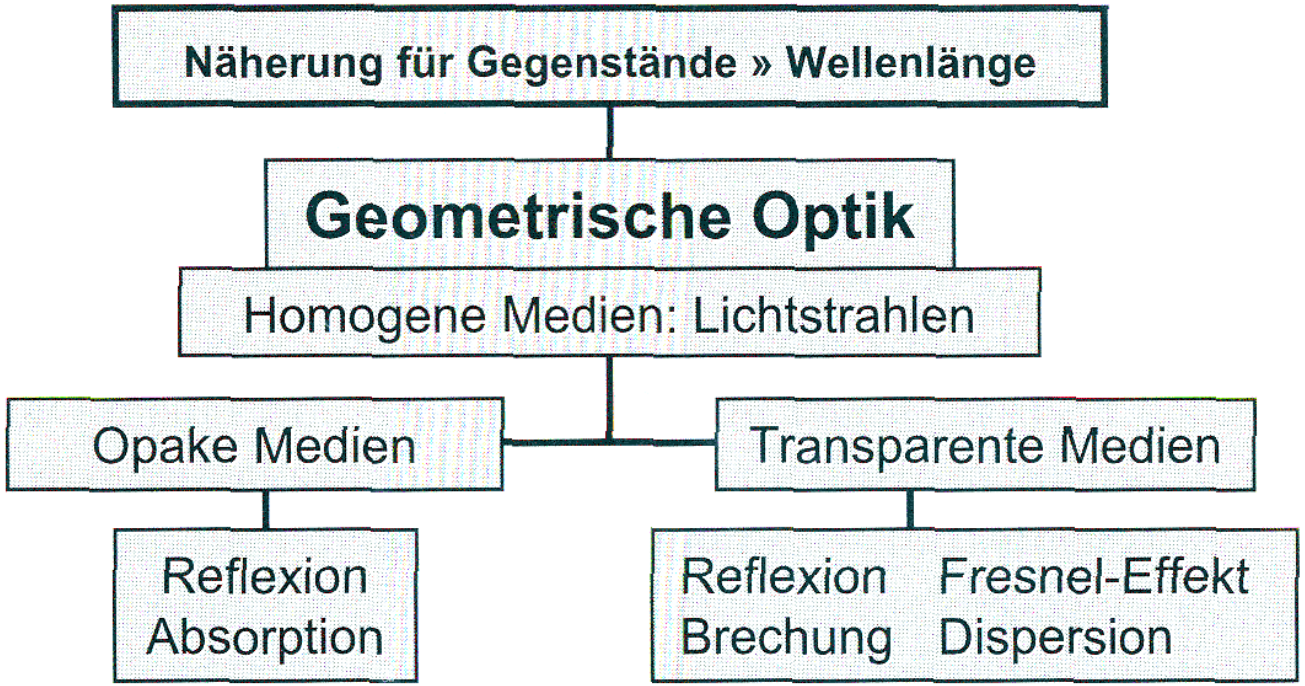


- Interferenz: durch Überlagerung kann die Intensität des Lichts ausgelöscht oder verstärkt werden
- Beugung: Ablenkung des Lichts, zum Beispiel an Blenden
- Polarisation: Wellen nicht frei im Raum, sondern ausgerichtet (Polfilter)

- Emission: Durch Quantensprung wird Licht ausgesendet
- Absorption: Lichtenergie wird in Quantensprung umgewandelt
- Streuung: Ablenkung von Lichtteilchen an anderen Teilchen

Quelle: Nischwwitz/Fischer/Haberächer

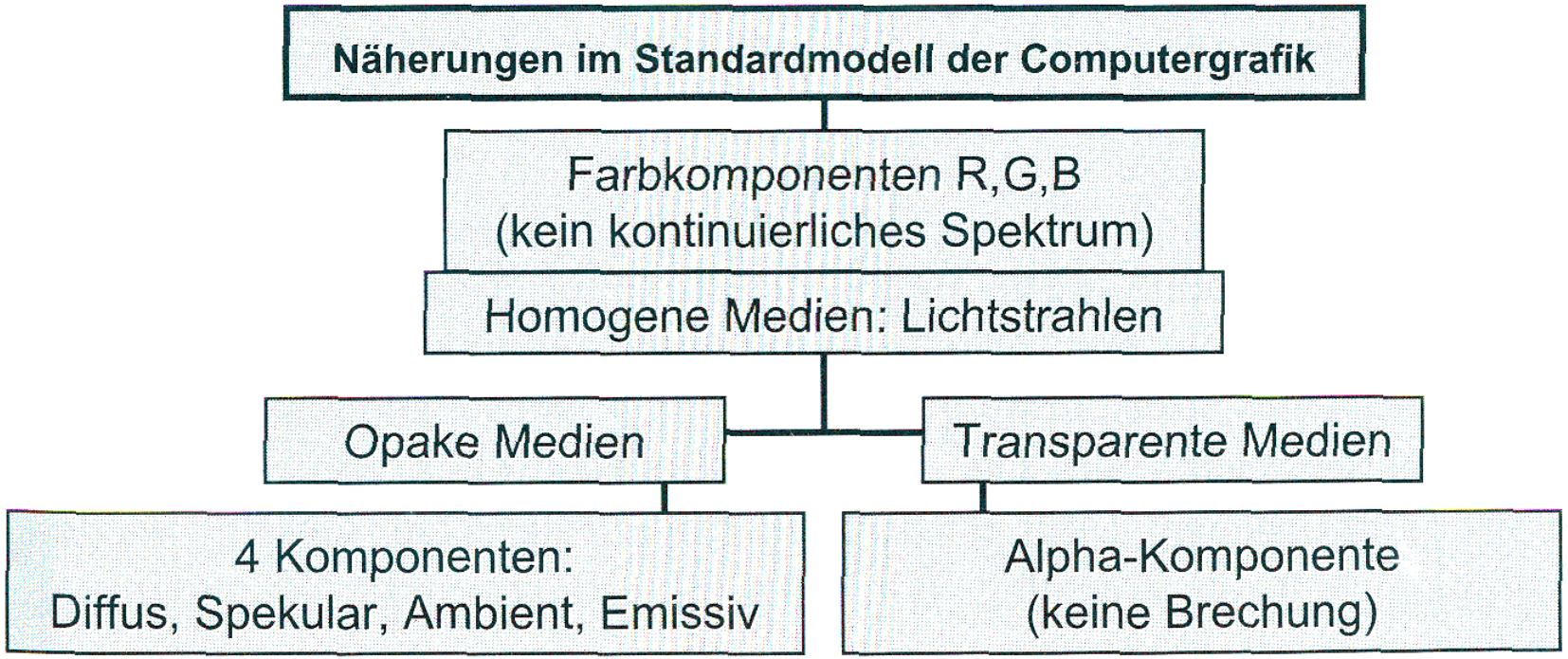
Die geometrische Optik: makroskopische Näherung



- Reflexion: die Spiegelung des Lichtes an einer ideal glatten Oberfläche, der Anteil des reflektierten Lichts steigt mit dem Einfallswinkel (Winkel zwischen Normale und Lichteinfall), dieses Phänomen wird als Fresnel-Effekt bezeichnet
- Dispersion: Abhängigkeit des Brechungsindex von der Wellenlänge (Farbskala beim Zerlegen von Licht am Prisma)

Quelle: Nischwwitz/Fischer/Haberächer

Licht und Beleuchtung in der Computergrafik



- siehe nächste Folie

- Auswahl von drei Farben anstelle eines kontinuierlichen Spektrums (Funktionsweise von Braun'schen Farbmonitoren, menschliches Auge)
- Alpha-Komponente: Maß für die Durchlässigkeit, von nicht durchlässig (opak) bis zu vollständig transparent

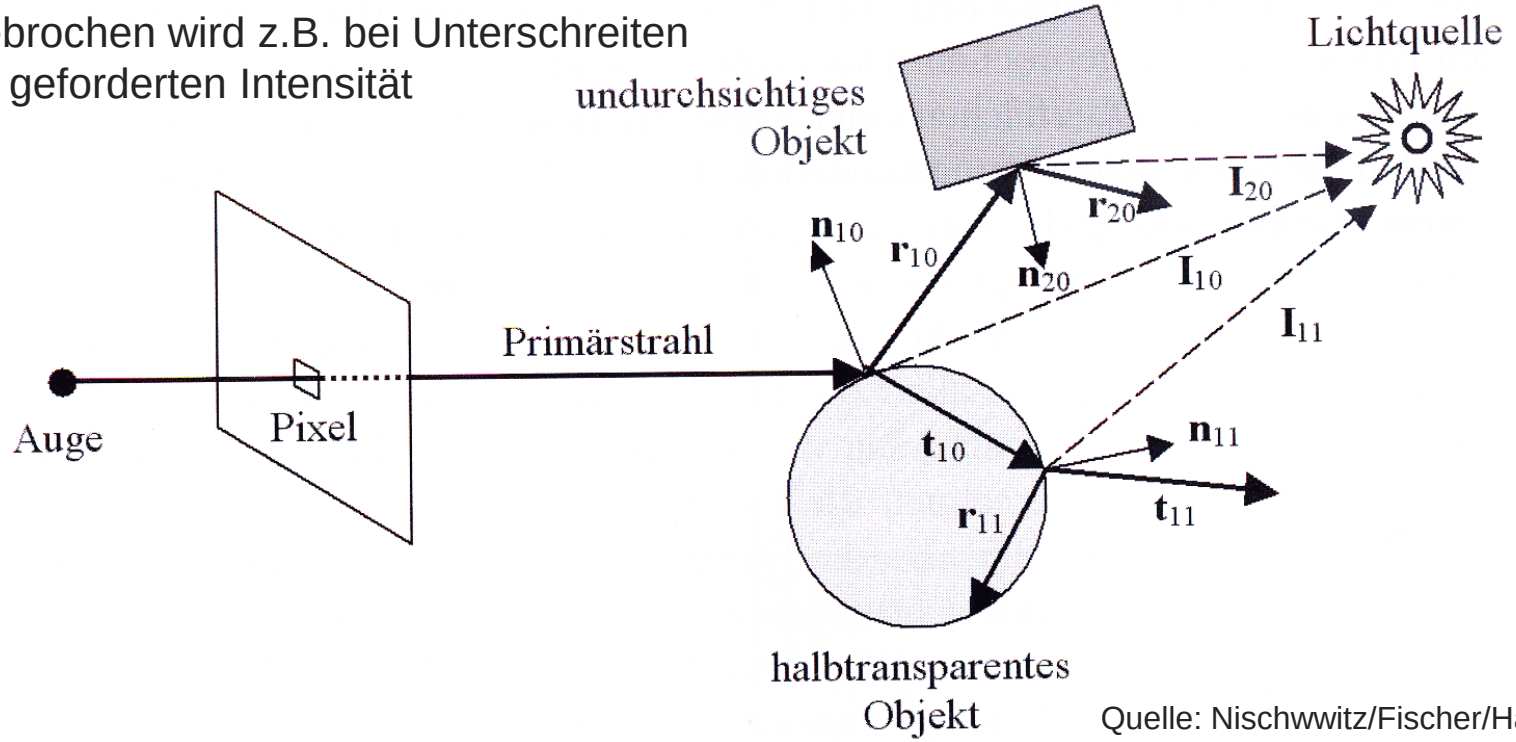
Quelle: Nischwwitz/Fischer/Haberächer

Beleuchtungsmodelle

- Die fotorealistische Darstellung von Objekten beruht auf Beleuchtungssimulation. Dabei werden im wesentlichen folgende Größen berücksichtigt:
  - Materialeigenschaften
  - Normalenvektor
  - Lichtquellen
- Grundsätzlich wird für jeden Punkt des Objektes eine Farbe ausgerechnet, die anhand der Grundfarbe des Materials, dessen Reflexionsverhalten (spiegelnde, diffuse, spekulare und ambiente Anteile), der Position, Intensität und Farbe der Lichtquellen und einem Umgebungslicht berechnet wird
  - Diffus: ein Anteil des reflektierten Lichts weicht von der Reflexionsrichtung ab (verursacht durch Oberflächen-Rauigkeit), nicht diffus: der Hüllkörper entspricht einer langgezogenen Zigarre, ideal diffus: Hüllkörper entspricht einer Halbkugel
  - Spekular: die spekulare Reflexion beschreibt die Spiegelung einer Lichtquelle und deren Reflexion auf dem Körper als spekulare Objektfarbe (nicht die eigentliche Farbe des Objektes)
  - Ambient: Verhalten eines Körpers bei ambienter Beleuchtung, wie viel Prozent eines ambienten Lichtes wird als diffuse Objektfarbe reflektiert
  - Emissiv: charakteristische Emissionsspektren der Materialien (Resonanzfrequenzen aufgrund unterschiedlicher Quantensprungenergien)

Beleuchtungsmodelle / Rendering

- Ray Tracing Verfahren:
  - Für jeden zu berechnenden Pixel wird ein Lichtstrahl rückwärts vom Blickpunkt in die Szene zurück verfolgt, bis ein Objekt getroffen wird (Primärstrahl)
  - Berücksichtigung der Punkt-Lichtquellen inklusive der spekularen und diffusen Anteile der Objektoberfläche (I10)
  - Schließlich wird der ambiente Anteil untersucht, bei dem die reflektierten (r10) und gebrochenen (t10) Strahlen betrachtet werden, um zu prüfen, ob sich durch sie ein ambienter Anteil ergibt (Sekundärstrahlen)
  - Abgebrochen wird z.B. bei Unterschreiten einer geforderten Intensität



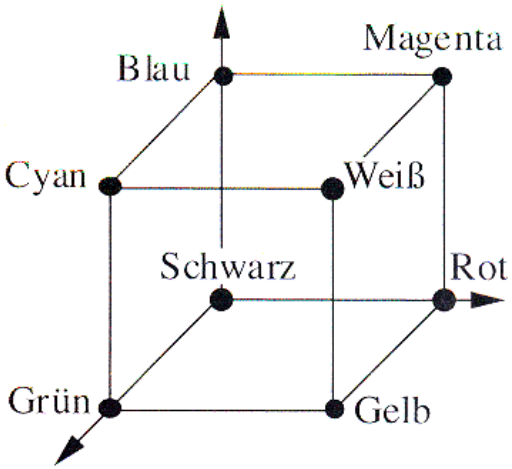
Quelle: Nischwwitz/Fischer/Haberächer

Rendering - Methoden

- Neben dem Ray Tracing Verfahren sind unterschiedliche andere Verfahren in der Literatur beschrieben, die je nach erwünschter Qualität und Geschwindigkeit entsprechend Anwendung finden.
  - Im Inventor gibt es seit einiger Zeit ein nicht-fotorealistisches Rendering, mit dem 3D-Geometrien in Form einer Handskizze dargestellt werden können
  - Beschleunigungsverfahren wie eine Level of Detail- oder eine Sichtbarkeits-Vorberechnung
  - Bildbasiertes Rendering: bei Änderung des Blickpunktes werden die Bitmaps der zuvor berechneten Ansichten von Flächen nur transformiert, was Zeit spart
  - Vereinfachung des Ray Tracing Verfahrens durch Weglassen der ambienten Lichtquellen und damit der ausschließlichen Verfolgung primärer Strahlen
- Auch hinsichtlich der Beleuchtungsmodelle werden unterschiedliche Ansätze und Kombinationen verwendet, in OpenGL beispielsweise
  - Emissive Komponente
  - Ambiente Komponente
  - Phong'sches Beleuchtungsmodell (spekulare Komponente)
  - Lambert'sches Beleuchtungsmodell (diffuse Komponeten)

Farbmodelle

- RGB (additive Primärfarben)
  - weiß ist die Addition aller drei Farben
  - Repräsentation in der Regel über 1byte pro Farbe, d.h. 3byte Farben = 16777216 Farben
  - in css-Notation bspw. als color: #ffffff; für weiß, #000000 für schwarz
  - Basis der Farbdarstellung an Monitoren
- CMY (Komplementärfarben von RGB)
  - Cyan absorbiert rotes Licht
  - schwarz als Addition aller drei absorbierenden Grundfarben
  - Drucker nutzen häufig das CMY-Farbmodell, schwarz wird hier aufgrund problematischer Mischung mit einer eigenen Kartusche umgesetzt (CMYK, K als "Key")
- eine Vielzahl anderer unterschiedlichster Modelle, herstellerabhängig oder genormt, anwenderbezogen oder an Farbtafeln angelehnt, …



Quelle: Rechenberg/Pomberger

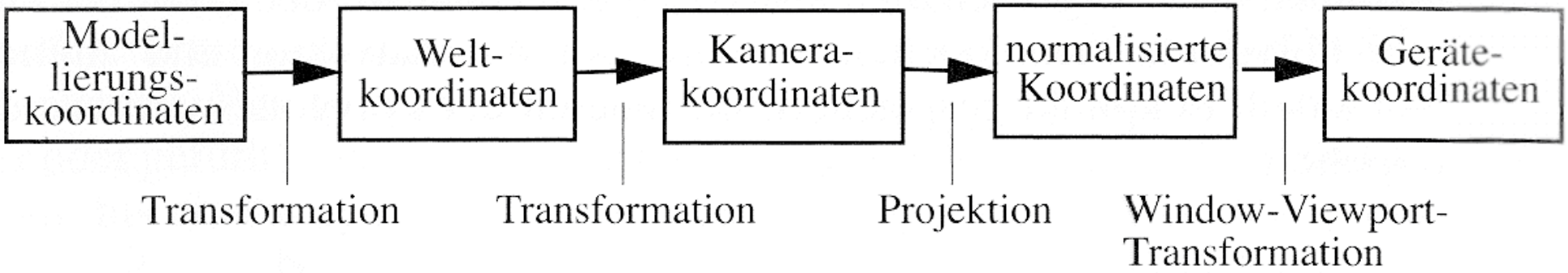
Verfügbare Grafikbibliotheken

- Low-Level und High-Level APIs
- Die wohl am meisten verwendete Bibliothek ist OpenGL
  - OpenGL Utility Library (GLU) Erweiterung von OpenGL
  - verfügbar auch mit C#-Wrapper als openTK
  - da frei verfügbar und auf allen Plattformen verfügbar, sehr breit einsetzbar
- DirectX
  - Anwendung unter Windows
  - Funktionalität von Direct3D als Teil von DirectX sehr stark an OpenGL angelehnt
- OpenInventor, Java3D
  - Möglichkeit der Interaktion durch sogenannte Manipulatoren
  - Java3D nutzt hinsichtlich der Beschreibung einer Szene eine stark an VRML angelehnte Form
  - Java3D unterstützt OpenGL und DirectX, damit kann die für die Hardware günstigste Variante ausgewählt werden

Objektdarstellung
(Rendering Pipeline)

Transformation und Abbildung (Viewing Pipeline)

- Viele Modellierer erlauben das Verschieben des Koordinatensystems zur vereinfachten Koordinateneingabe im Modellierungskontext
- Die Modellierungskoordinaten werden in ein Weltkoordinatensystem transformiert
- Die Blickrichtung des Betrachters wird durch die Transformation in Kamerakoordinaten erreicht, um das Objekt in entsprechende Ausrichtung zu bekommen
- Nun erfolgt die Position auf normalisierte Koordinaten durch Projektion auf die Betrachtungsebene
- Zum Schluß erfolgt die Transformation auf den Ausschnitt des Viewports in Gerätekoordinaten, hier sind der Maßstab und die Größe des Ansichtsfensters von Bedeutung



Quelle: Rechenberg/Pomberger

Erstellen von Modellgeometrie im AutoCAD-Framework objarx

- Im nebenstehenden Beispiel wird in einer vom AutoCAD-Framework abgeleiteten Klasse das zugehörige Modell in Weltkoordinaten gezeichnet und der Viewing Pipeline verfügbar gemacht
- Ebenso wäre es auch möglich, direkt in den Viewport hinein zu zeichnen
- Für die Transformationen zwischen Welt, aktuellem Viewport usw. stehen entsprechende Hilfsfunktionen zur Verfügung (s. auch nächste Folie)

```
Adesk::Boolean bnp_ew::worldDraw(AcGiWorldDraw* mode)
{
  assertReadEnabled();
  AcGePoint3d pt[3];
  AcGePoint3d zpkt;
  const AcGeVector3d dir(1,0,0);
  const AcGeVector3d zdir(0,0,1);
  AcGeVector3d rot;

  Adesk::UInt16 col = mode->subEntityTraits().color();

  double cw = 1;
  double h = LARGETEXTHEIGHT;

  mode->subEntityTraits().setColor(col);
  rot = m_Dir;
  rot.rotateBy(PI*0.5, zdir);

  pt[0] = m_Insert;
  zpkt = m_Insert + EW_BREITE * m_Dir;
  pt[1] = zpkt;
  mode->geometry().polyline(2, pt, NULL);

  pt[0] = zpkt - EW_BREITE * rot;
  pt[1] = zpkt + EW_BREITE * rot;
  mode->geometry().polyline(2, pt, NULL);

  return (AcDbEntity::worldDraw(mode));
}
```

Mathematische Grundlagen

- Die Transformationsmatrix im 2D-Raum ist an sich eine 2x2 Matrix, die aber keine Verschiebung erlaubt, unter Erweiterung eines 2D-Vektors um eine zusätzliche Koordinate, die stets 1 ist (x, y, 1), kann in der dritten Spalte der Matrix eine Verschiebung angegeben werden:

$$\begin{pmatrix} f_1 & 0 & 0 \\ 0 & f_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & f & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix}$$

Skalierung | Drehung | Scherung | Translation

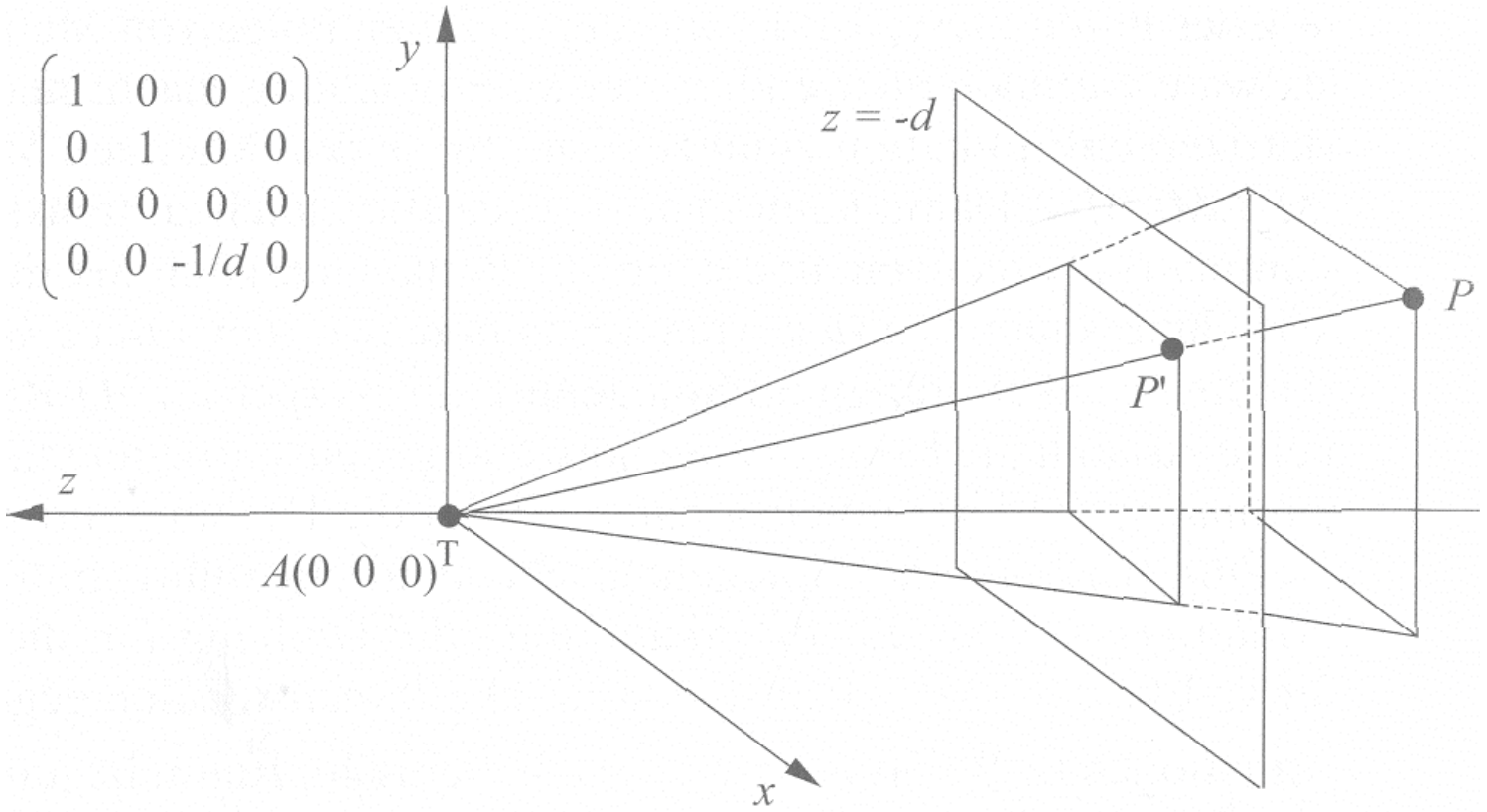
- Dieses Prinzip homogener Koordinaten kann auch im 3D-Raum verwendet werden

$$\begin{pmatrix} f_1 & 0 & 0 & 0 \\ 0 & f_2 & 0 & 0 \\ 0 & 0 & f_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Skalierung | Drehung um x-Achse | Spiegelung (yz-Ebene) | Translation

Mathematische Grundlagen für die Projektion

- Für eine perspektivische Projektion wird die Z-Koordinate auf 0 gesetzt, indem die 3.Zeile der Matrix Nullen erhält, Projektionszentrum sei (0,0,0)
- Die Skalierung erfolgt durch die 4.Komponente, in der der Z-Wert mit dem negativen Kehrwert des Abstandes multipliziert wird und damit die 4. Koordinate ungleich 1 werden kann. In diesem Fall wird der entstehende Vektor mit diesem Wert multipliziert



Quelle: Rechenberg/Pomberger

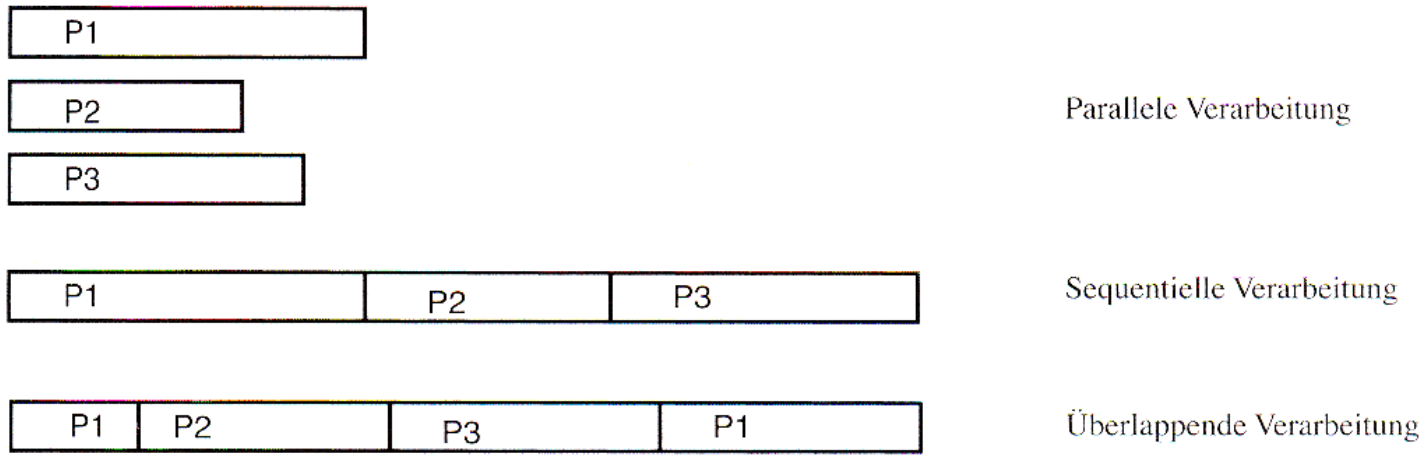
Weitere (hier nicht behandelte) Themen der Computergrafik

- Bildbearbeitung
  - Dithering-Verfahren zur Verbesserung von Farbverläufen in Rasterbildern
  - Anti-Aliasing zur Verbesserung von Übergängen
  - Morphing, Verwischen und andere Effekte der Bildbearbeitung
- Mustererkennung
  - OCR
  - Vektorisierung: Erkennung von Linien in Rastergrafiken
  - Gesichtserkennung
  - …
- Virtuelle Realität, Animation
  - Darstellung großer Modelle
  - augmented Reality
  - Animation von Modellen
  - geometrische, funktionale und physikalische Simulation
  - …

Prozesse und Threading

Allgemeines

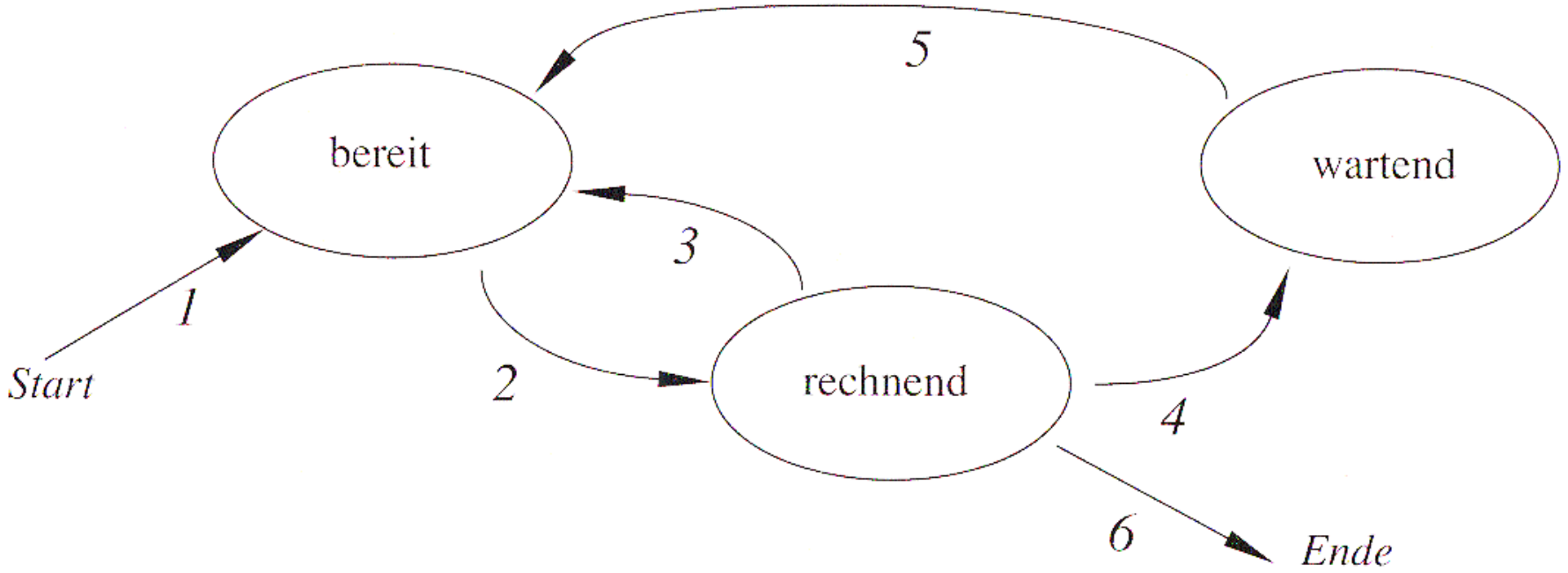
- Ein Prozess bezeichnet den sequentiellen Ablauf eines Programms auf einem Rechner
  - Ein Programm beschreibt den ablaufenden Prozess auf abstrakte Weise
  - Der Prozess ist eine konkret vorhandene Instanz des Programms, das Betriebssystem teilt dem Prozess Prozessorkapazität zu verwaltet ihn
- Betriebssysteme können mehrere Prozesse gleichzeitig ausführen, sie sind "multitasking"-fähig
  - Die Fähigkeit zum Multitasking hängt nicht vom Prozessortyp (Ein- oder Mehrkern) ab
  - Multitasking ist ebenfalls nicht abhängig von der Bearbeitung der Prozesse. Mögliche Varianten dieser Nebenläufigkeit sind im Bild dargestellt



Quelle: Rechenberg/Pomberger

Prozesszustände

- Auf einem Single-Prozessor kann zu einem Zeitpunkt nur ein Prozess in Bearbeitung sein, also running oder rechnend. Andere Prozesse sind zu diesem Zeitpunkt blockiert, wobei sie rechenbereit sind oder nicht. Sofern der Prozess nicht rechenbereit ist, kann die Ursache dafür im noch nicht vorliegenden Ereignis (z.B. Hardwarezugriff oder warten auf Ergebnis eines anderen Prozesses) liegen



Quelle: Rechenberg/Pomberger

Threads

- Sobald ein Prozess selber mehrere Aufgaben parallel erledigen muss (wie z.B. das Drucken und gleichzeitige Weiterarbeiten am Text in Word), stößt ein sequentiell ablaufendes Programm an seine Grenzen.
- Möglichkeiten, die sich hier bieten:
  - Starten eines eigenen, neuen Programms mit eigenem Adressraum, das z.B. das Drucken übernimmt. Sobald die parallelen Prozesse Daten austauschen und kommunizieren müssen, sind derartige "schwergewichte Prozesse" ungeeignet, da Kooperation von Prozessen in getrennten Adressräumen nicht effizient ist (siehe z.B. das Marshalling von Com in .NET-Umgebungen)
  - Starten eines Threads (thread of control – Handlungsfaden) als "leichtgewichtiger" Prozess. Dieser "Prozess im Prozess" beinhaltet nur eine minimale Menge an Kontextinformationen und kann nebenläufige Operationen innerhalb eines Programms abbilden, auch der Zugriff auf den gemeinsam genutzten Speicher ist einfacher
- Ob Threads möglich sind, hängt von den Möglichkeiten des Betriebssystems ab. Windows oder Linux ermöglichen derartige leichtgewichtige Prozesse im Systemkern als kernel threads

Quelle: Rechenberg/Pomberger

Probleme mit Threads

- Wenn sich verschiedene Threads eine gemeinsame Variable teilen und diese auch schreibend bedienen, könnten konkurrierende Zugriffe zu fehlerhaften Ergebnissen führen. Die Ursache liegt darin begründet, dass beim Schreiben der Variablen selbst mehrere Schritte durchlaufen werden, die dann evtl. abwechselnd ablaufen und der zuletzt Schreibende "gewinnt"
- Dieses Problem wird mit der Forderung nach gegenseitigem Ausschluss angegangen:
  - der zweite Prozess führt seine Operation auf dem Ergebnis des ersten aus
  - Implementierungen dieses Prinzips verhängen entsprechende Sperren auf Hardware-Interrupts oder andere Prozesse. Dabei werden Sperren über Variablen (Semaphores) oder algorithmische Lösungen verwendet
- Grundidee des gegenseitigen Ausschlusses ist es, eine Variable bei Eintritt in einen kritischen Bereich zu setzen und so einen anderen Thread so lange zu blockieren, bis der Bereich wieder "frei" wird
- Teilweise werden derartige Aufgaben von höheren Programmiersprachen unterstützt, in Java dient das Schlüsselwort "synchronized" vor einer Methode dazu, eine derartige Sperre einzurichten
- Durch den versuchten gegenseitigen Zugriff können Threads in eine Sackgasse laufen (deadlock). Hier ist ein gutes Design zwingend erforderlich
- Weiteres Handicap: nebenläufige Prozesse lassen sich schlecht debuggen

Thread-Möglichkeiten in C#

- Wie in Java mit dem Schlüsselwort synchronized kann in C# das Attribut

```
[MethodImpl(MethodImplOptions.Synchronized)]
public void threadSichereMethode() …
```

  verwendet werden
- Das Attribut ist auch auf Eigenschaften anwendbar:

```
private long _eigenschaft;
public long Eigenschaft
{
    get { return _eigenschaft; } //beim Lesen hier nicht benötigt
    [MethodImpl(MethodImplOptions.Synchronized)]
    set { _eigenschaft = value; }
}
```

- Eine Implementierung mit Semaphores und explizitem Lock kann wie folgt aussehen:

```
private readonly object syncLock = new object();
public void threadSichereMethode() {
    lock(syncLock) { /* sicherer Code */ }
}
```

Wichtige Funktionen der Thread-Klasse

- Start
  - Bewirkt, dass die Ausführung eines Threads geplant wird
- Abort
  - Löst eine ThreadAbortException im Thread aus, für den der Aufruf erfolgte, um das Beenden des Threads zu beginnen. Durch den Aufruf dieser Methode wird der Thread i. d. R. beendet.
- Begin- und EndCriticalRegion
  - Benachrichtigt einen Host, dass die Ausführung im Begriff ist, zu einem Codebereich überzugehen (oder diesen zu beenden), in dem die Auswirkungen eines Threadabbruchs oder einer nicht behandelten Ausnahme andere Aufgaben in der Anwendungsdomäne gefährden könnten
- Sleep
  - Blockiert den aktuellen Thread für die angegebene Anzahl von Millisekunden
- Suspend
  - Hält den Thread an. Hat keine Auswirkungen, wenn der Thread bereits angehalten ist
- Resume
  - Nimmt die Ausführung eines angehaltenen Threads wieder auf.
- Join
  - Blockiert den rufenden Thread bis der Thread des Thread-Objektes beendet wird (Synchronisation)

Apartment – Modelle (insbesondere relevant für COM)

- Es gibt drei Arten von Apartments (Zugriff auf Objekte in Threads):
  - Single Threaded Apartments (STA) besitzen genau einen Thread und beliebig viele Objekte. Es können beliebig viele STAs in einem Prozess existieren. Es erfolgt nur ein Aufruf gleichzeitig an das aufzurufende Objekt. Die restlichen Aufrufe warten in einer Warteschlange auf die Freigabe des Apartment-Threads. Dies impliziert, dass zwei Objekte in demselben STA auch von zwei verschiedenen Clients nicht parallel aufgerufen werden können. Als Besonderheit wird das erste in einem Prozess initialisierte STA automatisch zum Main-STA. Pro Prozess gibt es nur genau ein Main-STA, alle Objekte die keine explizite Anforderung an das Apartment stellen, werden in diesem erzeugt.
  - Multi Threaded Apartments (MTA) besitzen beliebig viele Threads. Es gibt in einem Prozess allerdings maximal ein MTA. Dadurch können von mehreren Clients gleichzeitig Aufrufe an das gleiche oder auch verschiedene Objekte erfolgen. Die Anforderungen an die Implementierung der Komponenten sind wegen der notwendigen Threadsicherheit sehr hoch.
  - Neutral Threaded Apartments (NTA) haben keine Threadaffinität. Es gibt in einem Prozess allerdings maximal ein NTA. Jedes Objekt in einem NTA kann von einem STA/MTA Apartment ohne Threadübergang aufgerufen werden. Der Thread wird also kurzzeitig in das NTA ausgeliehen, um damit aus Performancegründen das Marshalling zu überspringen. Neutral Threaded Apartments wurden mit Windows 2000 eingeführt, um die Vorzüge von MTA (meist kein Marshalling notwendig) mit den Vorzügen von STA (keine threadsichere Implementierung notwendig) zu vereinen.

Quelle: Wikipedia

Kommunikation zwischen Prozessen

- Threads erlauben die gemeinsame Verwendung von Objekten über Threadgrenzen hinaus, wobei dabei die Zugriffsrechte über Apartment-Modelle oder andere Techniken geregelt werden müssen, lösen aber nicht das Problem der Kommunikation zwischen "schwergewichtigen" Prozessen
- Für die Kommunikation zweier Prozesse ist der direkte Zugriff auf Speicherbereiche des anderen Prozesses geschützt. Für die Kommunikation müssen daher andere Wege gefunden werden. Dazu zählen:
  - Kommunikation über Dateien, das Dateisystem muss dabei den Zugriff auf eine Datei von mehreren Prozessen aus zulassen.
  - Kommunikation über gemeinsame Speicherbereiche, das Betriebssystem muss dafür erlauben, Speicher einzurichten, den mehrere Prozesse gleichzeitig nutzen können, hierfür sind ebenfalls Synchronisationsmechanismen erforderlich
  - Kommunikation über Pipes, wobei eine pipe wie eine Datei behandelt wird, die flüchtig ist, also nicht auf einer Festplatte existieren muss
  - Kommunikation über Nachrichten, hierbei bietet das Betriebssystem einen Mechanismus an, mit dem sich Prozesse Nachrichten schicken können. Je nach Betriebssystem und/oder Programmiersprache werden die Endpunkte als socket, port oder mailbox bezeichnet
  - Kommunikation über Prozedurfernaufrufe (RPC), in der Regel über Nachrichten-Kommunikation, häufig die Basis von Client-Server-Anwendungen

Beispiel: Socket-Kommunikation (Server-Hauptroutine)

```
public void Start()
{
    Socket client;
    //Auf port 8001 auf Anfragen beliebiger IP-Adressen warten
    //Für IPV6 müßte ein weiterer Listener erstellt werden
    listener = new TcpListener(IPAddress.Any, 8001);
    listener.Start();
    Console.WriteLine("Listening...");
    //den Server in Endlos-Schleife laufen lassen, stoppen ist
    //durch Setzen der Variable serverRunning möglich, allerdings
    //wartet die Schleife immer an der Stelle AccceptSocket,
    //hier muss auch der Listener gestoppt werden, damit es weitergeht
    while (serverRunning)
    {
        try
        {
            //Auf synchrone Anfrage warten und Socket(client) erstellen
            client = listener.AcceptSocket();
        }
        catch
        {
            break;
        }
        Console.WriteLine("Client connected: Connection Nr. ({0}) from {1}", ++tc,
            client.LocalEndPoint.ToString());
        //.. und in einem eigenen Thread abarbeiten
        // weitere Clients können abgearbeitet werden, auch wenn andere Abfragen noch laufen
        Thread a = new Thread(new ParameterizedThreadStart(HandleClientRequest));
        //Socket-Objekt wird in die Thread-Funktion kopiert, daher hat jeder Thread seinen eigenen Socket
```

Beispiel: Socket-Kommunikation (Server-Thread)

```
public void HandleClientRequest(object objcurrentClient)
{
    //auf Socket casten
    Socket currentClient = (Socket)objcurrentClient;
    try
    {
        bool ende = false;
        while (false == ende)
        {
            //Ausgabe-Stringliste erzeugen
            List<string> outputData = new List<string>();
            //Aktuelles Datum aufnehmen
            outputData.Add("send at: " + DateTime.Now.ToString());
            //Daten vom Client empfangen (synchron, warten bis Client gesendet hat)
            string cmd = Receive(currentClient);
            //den Anfragestring auswerten und Rückgabedaten aufbereiten...
            //hier einfach nur das Command anhängen..
            outputData.Add(cmd);
            // Antwort an den Client schreiben
            write(currentClient, outputData);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("thread terminated by error!\n" + e.Message + "\n" + e.StackTrace);
```

Beispiel: Socket-Kommunikation (Server-Receive)

```
private string Receive(Socket currentClient)
{
    string stringData = "";
    try
    {
        //Byte-Array für Empfangsdaten anlegen
        byte[] data = new byte[1024];
        //Warten, bis gesendet wurde...
        int receivedDataLength = currentClient.Receive(data);
        //Bytefolge in string wandeln
        stringData = Encoding.ASCII.GetString(data, 0, receivedDataLength);
    }
    catch
    {
        Console.WriteLine("Receive failed");
    }
    return stringData;
}
```

Beispiel: Socket-Kommunikation (Server-Write/Send)

```
private void write(Socket currentClient, List<string> what)
{
    try
    {
        string send = "";
        //Dem verbundenen Client alle Listen-Elemente senden
        foreach (string itm in what)
        {
            send += send != "" ? "\n" : "";
            send += itm + itm + itm + itm;
        }
        currentClient.Send(Encoding.ASCII.GetBytes(send));
    }
    catch
    {
    }
}
```

Beispiel: Socket-Kommunikation (Client-Hauptroutine)

```
public static void Main()
{
    //Mit Server "localhost" auf Port 8001 verbinden
    server = ConnectSocket("localhost", 8001);
    bool ClientRunning = true;

    //Endlosschleife
    while (ClientRunning)
    {
        Console.WriteLine("'exit' zum Verlassen des Programms, 'stop' zum ");
        Console.WriteLine("Beenden der Serververbindung eingeben.");
        Console.Write("Eingabe: ");
        string input = Console.ReadLine();
```

- an dieser Stelle wird der eingegebene String analog der Methode Server-Write an den Server verschickt und danach mit Receive auf eine Antwort des Servers gewartet
- durch die Schleife ClientRunning könnte der Socket für weitere Anfragen verwendet werden, der Server muß dann eine ähnliche Schleife implementieren

Beispiel: Socket-Kommunikation (Client-Connect)

```
private static Socket ConnectSocket(string server, int port)
{
    Socket sock = null;
    try
    {
        //Neues Socket-Objekt erstellen (IPV4 vom Typ Stream, Protokoll Tcp)
        sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        //... und mit dem Server verbinden
        sock.Connect(server, port);
    }
    catch (Exception e)
    {
        Console.WriteLine("Error..... " + e.Message + "\n" + e.StackTrace);
    }
    return sock;
}
```

Beispiel

- Wird der soeben vorgestellte Server auf Port 80 eingestellt und der localhost bzw. 127.0.0.1 im Browser angefordert, wird die Anfrage des Browsers an den Server wiedergegeben