

$$\begin{pmatrix} r_z & 0 & -r_x & 0 \\ 0 & r_z & -r_y & 0 \\ 0 & 0 & -1 & r_z \end{pmatrix} \cdot \begin{pmatrix} \tau x \\ \tau y \\ \tau z \\ \tau \end{pmatrix} =: \begin{pmatrix} \tau' \cdot x_p \\ \tau' \cdot y_p \\ \tau' \end{pmatrix}$$

Bildgenerierung

Holger Arndt

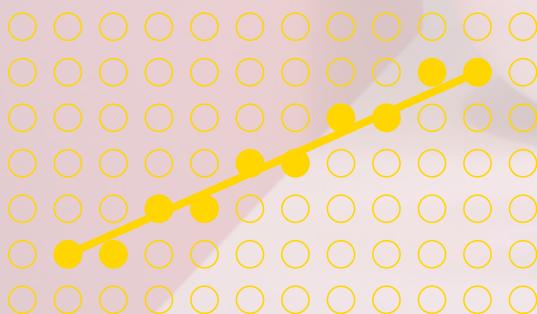
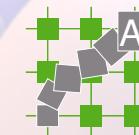
Wintersemester 2021/2022

basierend auf der gleichnamigen Vorlesung

von Prof. Dr. Bruno Lang vom Wintersemester 2013/2014



BERGISCHE
UNIVERSITÄT
WUPPERTAL



Stand: Donnerstag, 20. Januar 2022, 17:22



Inhaltsverzeichnis

1 Einführung	1-1
1.1 Organisatorisches	1-2
1.1.1 Hörerkreis	1-2
1.1.2 Ablauf von Vorlesung und Übung	1-5
1.1.3 Leistungsnachweis/Credits	1-6
1.1.4 Ziele der Vorlesung	1-6
1.2 Fragestellung	1-7
Literatur	Ref-1
2 Ein-/Ausgabe-Hardware und Interaktion	2-1
2.1 Linien-orientierte Ausgabegeräte	2-3
2.1.1 Vector-Displays	2-3
2.1.2 Plotter	2-6
2.2 Raster-orientierte Ausgabegeräte	2-8
2.2.1 Kathodenstrahlröhre mit Lochmaske	2-8
2.2.2 Bildschirme ohne CRT	2-16
2.2.3 Drucker	2-21
2.3 Vergleich	2-23

2.4	Eingabegeräte	2-24
2.4.1	Eingabegeräte für Positionen	2-26
2.5	Eingabemodi	2-30
2.5.1	Sampling	2-30
2.5.2	Ereignisgesteuert	2-31
2.5.3	Vergleich der Verfahren	2-34
2.6	Regeln für den Entwurf von Interaktion	2-35

3 Scan Conversion

3-1

3.1	Scan Conversion für Strecken	3-8
3.1.1	Das naive Verfahren	3-10
3.1.2	Inkrementell mit reeller Rechnung	3-11
3.1.3	Inkrementell mit ganzzahliger Rechnung	3-15
3.2	Scan Conversion für Kreislinien	3-18
3.2.1	Zwei naive Verfahren	3-19
3.2.2	Ein inkrementeller Ansatz	3-21
3.3	Scan Conversion für Polygone	3-26
3.3.1	Polygone	3-27
3.3.2	Der Grundalgorithmus	3-29
3.3.3	Ein inkrementeller Ansatz	3-33
3.3.4	Der Flood Fill Algorithmus	3-40

3.4	Nochmals: Strecken und Kreise	3-43
3.4.1	Linien vorgegebener „Strichbreite“	3-43
3.4.2	Unterbrochene Linien	3-46
3.5	Räumliche und farbliche Auflösung	3-48
3.5.1	Räumliche Auflösung statt farblicher Auflösung	3-48
3.5.2	Farbliche Auflösung statt räumlicher Auflösung	3-52
3.6	Text	3-54
4	Koordinatentransformationen	4-1
4.1	Von der Projektionsebene zu Gerätekordinaten	4-4
4.2	Projektion	4-8
4.2.1	„Standard-Parallelprojektion“	4-10
4.2.2	„Standard-Perspektivische Projektion“	4-12
4.2.3	Allgemeine Parallel- bzw. perspektivische Projektion	4-14
4.2.4	Projektive („homogene“) Koordinaten	4-21
4.3	Transformationen in der Modellierung	4-26
5	Clipping für Strecken und Polygone	5-1
5.1	Der Pixel-orientierte Ansatz	5-2
5.2	Der 2D-analytische Ansatz	5-5
5.2.1	Strecken-Clipping naiv	5-7
5.2.2	Strecken-Clipping nach Cohen und Sutherland	5-11

5.2.3	Strecken-Clipping nach Cyrus, Beck, Liang und Barsky	5-17
5.2.4	Polygon-Clipping nach Sutherland/Hodgman	5-23
5.2.5	Hardware-Realisierung	5-28
5.3	Der 3D-analytische Ansatz	5-32
6	Sichtbarkeit	6-1
6.1	Der Pixel-orientierte Ansatz	6-3
6.1.1	Naive Implementierung	6-4
6.1.2	Optimierung I: Zell-Raster-Technik	6-13
6.1.3	Optimierung II: „z-Puffer-Algorithmus“	6-20
6.2	Der analytische Ansatz	6-25
6.2.1	Hidden Line Removal für Polygonszenen	6-25
6.2.2	Optimierungen für Polyederszenen	6-32
6.3	Vereinfachte Algorithmen	6-37
6.3.1	Painter's Algorithm	6-37
6.3.2	Der Silhouetten-Algorithmus	6-42
7	Färbung	7-1
7.1	Lichtmodelle	7-3
7.1.1	Das physikalische Modell	7-3
7.1.2	Das Sampling-Modell	7-5
7.1.3	Das Faltungs-Modell	7-7

7.1.4	Das RGB-Modell	7-10
7.1.5	Das HSV/HLS-Modell	7-12
7.2	Modellierung optischer Effekte (I)	7-15
7.2.1	Ambientes Licht	7-16
7.2.2	Diffuse Reflexion	7-17
7.2.3	Winkelabhängige Reflexion	7-19
7.2.4	Entfernungsabhängige Dämpfung	7-21
7.2.5	Farbverschiebung (Depth Cueing)	7-23
7.2.6	Gerichtete Lichtquellen nach Warn	7-25
7.3	Lokale Beleuchtungsmodelle	7-29
7.3.1	Das „triviale“ Modell	7-30
7.3.2	Das Modell von Bouknight	7-31
7.3.3	Das Modell von Phong	7-32
7.4	Färbungsstrategien für Polygone	7-33
7.4.1	Konstante Färbung	7-35
7.4.2	Farbwertinterpolation (Gouraud-Shading)	7-40
7.4.3	Normaleninterpolation (Phong-Shading)	7-44
7.5	Modellierung optischer Effekte (II)	7-46
7.5.1	Schatten	7-47
7.5.2	Oberflächenstruktur (Textur)	7-53
7.5.3	Transparenz	7-63

8 Modellierung der Objekte	8-1
8.1 Flächenbasierte Modelle	8-2
8.2 Volumenbasierte Modelle	8-6
8.2.1 Operationen mit bestehenden Volumina	8-8
8.2.2 Definition von Volumina	8-9
8.2.3 Speicherung von Volumina	8-11
8.3 Prozedurale Modelle	8-15
8.3.1 Fraktale Modelle	8-16
8.3.2 Objekterzeugung mit Grammatiken	8-19
8.3.3 Partikelsysteme	8-24
9 Kurven und Flächen	9-1
9.1 Parametrisierte kubische Kurven	9-2
9.1.1 Hermite-Kurven	9-6
9.1.2 Bézier-Kurven	9-9
9.1.3 Kubische Splines	9-14
9.1.4 Unterteilung von Kurven	9-23
9.1.5 Zeichnen von Kurven	9-26
9.2 Parametrisierte bikubische Flächen	9-26
9.2.1 Bézier-Flächen	9-28
9.3 Rotationskörper	9-30

10 Färbung II**10-1**

10.1 Rekursives Raytracing	10-3
10.1.1 Das Brechungsgesetz	10-4
10.1.2 Der Raytracing-Ansatz nach Whitted	10-5
10.1.3 Beschleunigung der Strahlverfolgung (I)	10-13
10.1.4 Beschleunigung der Strahlverfolgung (II)	10-25
10.1.5 Selektives Raytracing	10-32
10.1.6 Bemerkungen	10-33
10.2 Radiosity-Verfahren	10-34
10.2.1 Raumwinkel	10-37
10.2.2 Radiosity	10-39
10.2.3 Von den Objekten zum Gleichungssystem	10-41
10.2.4 Vom Gleichungssystem zum Bild	10-45
10.2.5 Die Formfaktoren	10-47
10.2.6 Numerische Berechnung der Formfaktoren	10-50
10.2.7 Schrittweiser Lösungsaufbau	10-63
10.2.8 Unterteilung der Patches	10-67
10.2.9 Zusammenfassung	10-71
10.3 Rendergleichung und Monte-Carlo-Integration	10-73
10.3.1 Radiance	10-74
10.3.2 Rendergleichung	10-75
10.3.3 Zusammenhang zwischen Rendergleichung und Radiosity	10-79

10.3.4 Monte-Carlo-Integration	10-81
10.3.5 Erweiterungen des Raytracing-Modells	10-82
10.3.6 Path Tracing	10-84
10.3.7 BRDF	10-85
10.3.8 Kombination von BRDFs	10-89
10.3.9 Importance Sampling	10-90
10.3.10 Direkte Beleuchtung	10-91
10.3.11 Multiple Importance Sampling	10-92
10.3.12 Weiterentwicklungen	10-94

1 Einführung

Inhalt

1.1 Organisatorisches	1-2
1.1.1 Hörerkreis	1-2
1.1.2 Ablauf von Vorlesung und Übung	1-5
1.1.3 Leistungsnachweis/Credits	1-6
1.1.4 Ziele der Vorlesung	1-6
1.2 Fragestellung	1-7

1.1 Organisatorisches

1.1.1 Hörerkreis

- **Bachelor Informatik**

Wahlpflichtveranstaltung im Wahlpflichtmodul INF12: Bild- und Audioverarbeitung

- **Bachelor Mathematik, PO 2020** mit Nebenfach Informatik

Wahlpflichtveranstaltung im Wahlpflichtmodul INF12: Bild- und Audioverarbeitung

- **Bachelor Mathematik, PO 2011 und 2013** mit Nebenfach Informatik

Wahlpflichtveranstaltung im Wahlpflichtmodul NIInf.BAV: Bild- und Audioverarbeitung

- **Bachelor Wirtschaftsmathematik, PO 2020**

Wahlpflichtveranstaltung im Wahlpflichtmodul INF12: Bild- und Audioverarbeitung

- **Bachelor Angewandte Naturwissenschaften, PO 2012** mit Schwerpunkt fach Informatik

Wahlpflichtveranstaltung im Wahlpflichtmodul Bild- und Audioverarbeitung

- **Bachelor Angewandte Naturwissenschaften, PO 2007** mit Schwerpunkt fach Informatik

Wahlpflichtveranstaltung im Wahlpflichtmodul I9c: Bild- und Audioverarbeitung

- **Bachelor Informationstechnologie und Medientechnologie**

Wahlpflichtveranstaltung im Wahlpflichtmodul INF12: Bild- und Audioverarbeitung

- **Bachelor Informationstechnologie**

Wahlpflichtveranstaltung im Wahlpflichtmodul BAV: Bild- und Audioverarbeitung

- **Kombinatorischer Bachelor, Teilstudiengang Informatik**

Wahlpflichtveranstaltung im Wahlpflichtmodul INF12: Bild- und Audioverarbeitung

- **Kombinatorischer Bachelor, Optionalbereich, PO 2021**

Wahlpflichtkomponente im Wahlpflichtmodul INF301: Informatik 1

und Wahlpflichtkomponente im Wahlpflichtmodul INF302: Informatik 2

- **Master Chemie, PO 2018**

Wahlpflichtveranstaltung im Wahlpflichtmodul NInf.BildV: Einführung in die Bildverarbeitung

- **Master of Education Gym/Ge und BK, Teilstudiengang Informatik, PO 2021 (bzw. PO 2019)**
Wahlpflichtveranstaltung im Wahlpflichtmodul INF12: Bild- und Audioverarbeitung
(falls *Softwaretechnologie*, *Einführung in die Kryptographie*, *Einführung in Datenbanken* und *Einführung in die Didaktik der Informatik* bereits im Bachelor belegt wurde, *Bild- und Audioverarbeitung* aber nicht)
- **Master of Education Gym/Ge und BK, Teilstudiengang Informatik, PO 2014, PO 2011**
Wahlpflichtveranstaltung im Wahlpflichtmodul BAV: Bild- und Audioverarbeitung
(falls *Einführung in die Didaktik der Informatik* bereits im Bachelor belegt wurde, *Bild- und Audioverarbeitung* aber nicht)
- **Master of Education HRSGe, Teilstudiengang Informatik**
Wahlpflichtveranstaltung im Wahlpflichtmodul INF12: Bild- und Audioverarbeitung
(falls *Internettechnologien*, *Einführung in die Kryptographie* und *Einführung in Datenbanken* bereits im Bachelor belegt wurden, *Bild- und Audioverarbeitung* aber nicht)

1.1.2 Ablauf von Vorlesung und Übung

Termine

Vorlesung: Do., 16:15-17:45 Uhr Hörsaal 3

Beginn: Do., 14.10.

Übung: Mi., 12:15-13:45 Uhr G.14.11

Beginn: Mi., 20.10.

Übungsblätter

Ausgabe: wöchentlich immer mittwochs auf Moodle-Seite

Abgabe: in Vorlesung/Übung bzw. per E-Mail

Weitere Informationen und Material

im Moodle-Kurs zur Vorlesung:

<https://moodle.uni-wuppertal.de/course/view.php?id=26228>

1.1.3 Leistungsnachweis/Credits

Schriftliche oder mündliche Prüfung

- nach dem Wintersemester,
Wiederholung nach dem Sommersemester
- Dauer: 120 Minuten (schriftlich) bzw. 30 Minuten (mündlich)

1.1.4 Ziele der Vorlesung

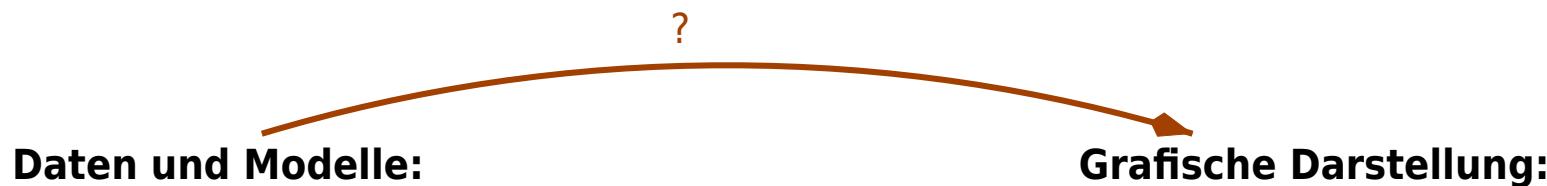
Kenntnisse

- wesentliche Schritte bei der Erzeugung von Bildern unterschiedlicher Komplexität
- dabei eingesetzte Techniken

Fähigkeiten

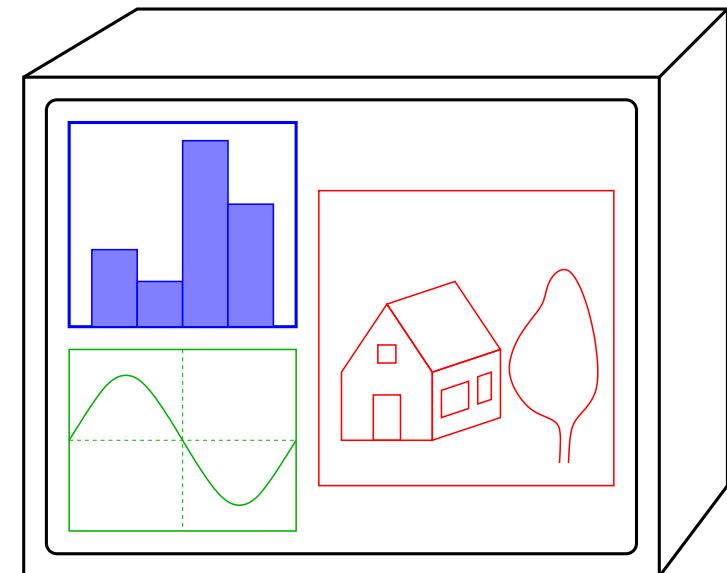
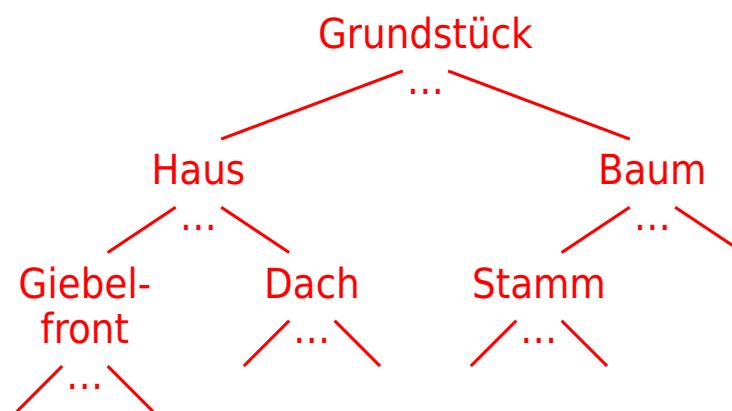
- Erstellung von Bildern mit Bibliotheken unterschiedlicher Mächtigkeit
- bei Bedarf Entwicklung eigener Algorithmen und Datenstrukturen
- Umsetzung von Algorithmen und Datenstrukturen in einer Programmiersprache

1.2 Fragestellung



$$y = \sin(x), x \in [0; 2\pi]$$

Umsatz: Jan 11,3 Mio
Feb 6,7 Mio
Mrz 27,4 Mio
Apr 18,2 Mio



Mögliche Klassifikationen der Grafikanwendungen

Komplexität der Objekte und des Bildes:

- 2D Linienzeichnung
- 2D Graustufen- oder Farbbild
- 3D Linienzeichnung (**wire frame**)
- 3D Linienzeichnung ohne verdeckte Kanten/Flächen
- 3D Graustufen- oder Farbbild mit diversen Effekten

Interaktionsmöglichkeiten:

- **offline** (Objekte festlegen, dann erst zeichnen)
- interaktives Design (immer „aktuellen Zustand“ anzeigen)

Rolle des Bildes:

- Endprodukt
(Visualisierung von Daten, Kartografie, Computerkunst, ...)
- Mittel zum Zweck
(z. B. CAD: Konstruktionsplan als Ausgangspunkt für Teileliste und Maschinensteuerung, ...)

Schritte bei der Darstellung

Auswahl:

- Welcher Teil der Daten/des Modells soll dargestellt werden?

Geometrie festlegen:

- Wie sollen die Daten dargestellt werden?
(z. B. parallele/perspektivische Projektion)
- Wo auf der Zeichenfläche und wie groß soll die Darstellung erfolgen?

Zeichnen:

- Umsetzen der (abstrakten) 2D-Beschreibung des Bildes in eine gerätespezifische Darstellung

(Interaktion:)

- Änderung der Modellparameter – und der Darstellung – gemäß Eingaben des Benutzers

Literatur

- [DBB06] Philip Dutré, Kavita Bala und Philippe Bekaert. **Advanced Global Illumination**. 2. Aufl. Wellesley: A K Peters, Aug. 2006. ISBN: 978-1-56881-307-3.
- [ESK96] José Encarnaçāo, Wolfgang Straßer und Reinhard Klein. **Grafische Datenverarbeitung 1: Gerätetechnik, Programmierung und Anwendung grafischer Systeme**. 4. Aufl. München: R. Oldenbourg Verlag, 1996. ISBN: 3-486-23223-1. TZA 122.
- [ESK97] José Encarnaçāo, Wolfgang Straßer und Reinhard Klein. **Grafische Datenverarbeitung 2: Modellierung komplexer Objekte und photorealistische Bilderzeugung**. 4. Aufl. München: R. Oldenbourg Verlag, 1997. ISBN: 3-486-23469-2. TZA 122.
- [Gla95] Andrew S. Glassner. **Principles of Digital Image Synthesis**. 1.0.1. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. San Francisco: Morgan Kaufmann Publishers, 1995. ISBN: 1-55860-276-3. beide Bände frei verfügbar über Google Books, kombinierte pdf-Datei unter der angegebenen URL. TVV 287-1 und TVV 287-2. URL: http://realtimerendering.com/Principles_of_Digital_Image_Synthesis_v1.0.1.pdf.
- [Hug+09] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner und Kurt Akeley. **Computer Graphics: Principles and Practice**. 3. Aufl. Upper Saddle River: Addison-Wesley, Feb. 2009. ISBN: 978-0-321-39952-6. (2nd ed.: TZA 126).

- [Nis+19] Alfred Nischwitz, Max Fischer, Peter Haberäcker und Gudrun Socher. **Computergrafik**. 4. Aufl. Wiesbaden: Springer Vieweg, Apr. 2019. ISBN: 978-3-658-25383-7. URL: <https://link.springer.com/book/10.1007%2F978-3-658-25384-4>.
- [RA90] David F. Rogers und James Alan Adams. **Mathematical Elements for Computer Graphics**. 2. Aufl. New York: McGraw-Hill, 1990. ISBN: 0-07-053529-9. TZA 233.
- [Sal06a] David Salomon. **Curves and Surfaces for Computer Graphics**. New York: Springer Science+Business Media, 2006. ISBN: 978-0-387-24196-8. URL: <https://link.springer.com/book/10.1007%2F0-387-28452-4>.
- [Sal06b] David Salomon. **Transformations and Projections in Computer Graphics**. London: Springer-Verlag, 2006. ISBN: 978-1-84628-392-5. URL: <https://link.springer.com/book/10.1007%2F978-1-84628-620-9>.
- [WW92] Alan Watt und Mark Watt. **Advanced Animation and Rendering Techniques: Theory and Practice**. New York: ACM Press, Dez. 1992. ISBN: 978-0-201-54412-1. TZA 289.

2 Ein-/Ausgabe-Hardware und Interaktion

Inhalt

2.1 Linien-orientierte Ausgabegeräte	2-3
2.1.1 Vector-Displays	2-3
2.1.2 Plotter	2-6
2.2 Raster-orientierte Ausgabegeräte	2-8
2.2.1 Kathodenstrahlröhre mit Lochmaske	2-8
2.2.2 Bildschirme ohne CRT	2-16
2.2.3 Drucker	2-21
2.3 Vergleich	2-23
2.4 Eingabegeräte	2-24
2.4.1 Eingabegeräte für Positionen	2-26
2.5 Eingabemodi	2-30
2.5.1 Sampling	2-30
2.5.2 Ereignisgesteuert	2-31
2.5.3 Vergleich der Verfahren	2-34
2.6 Regeln für den Entwurf von Interaktion	2-35

Motivation

- Die zur Darstellung eingesetzten Techniken hängen i. Allg. vom Ausgabegerät (Linien- oder Raster-orientiert) ab.
 - **Interaktion** kann es dem Benutzer ermöglichen, durch Eingabe von
 - Befehlen
 - Parametern (z. B. Größenangaben, Platzierung)die Darstellung des Modells (und evtl. den Ablauf des Programms) zu beeinflussen.
-

Bemerkung 2.1: Die Beschreibungen sind i. Allg. gegenüber den (aktuellen) technischen Realisierungen stark vereinfacht und erfassen nur eine geringe Anzahl möglicher Varianten.

2.1 Linien-orientierte Ausgabegeräte

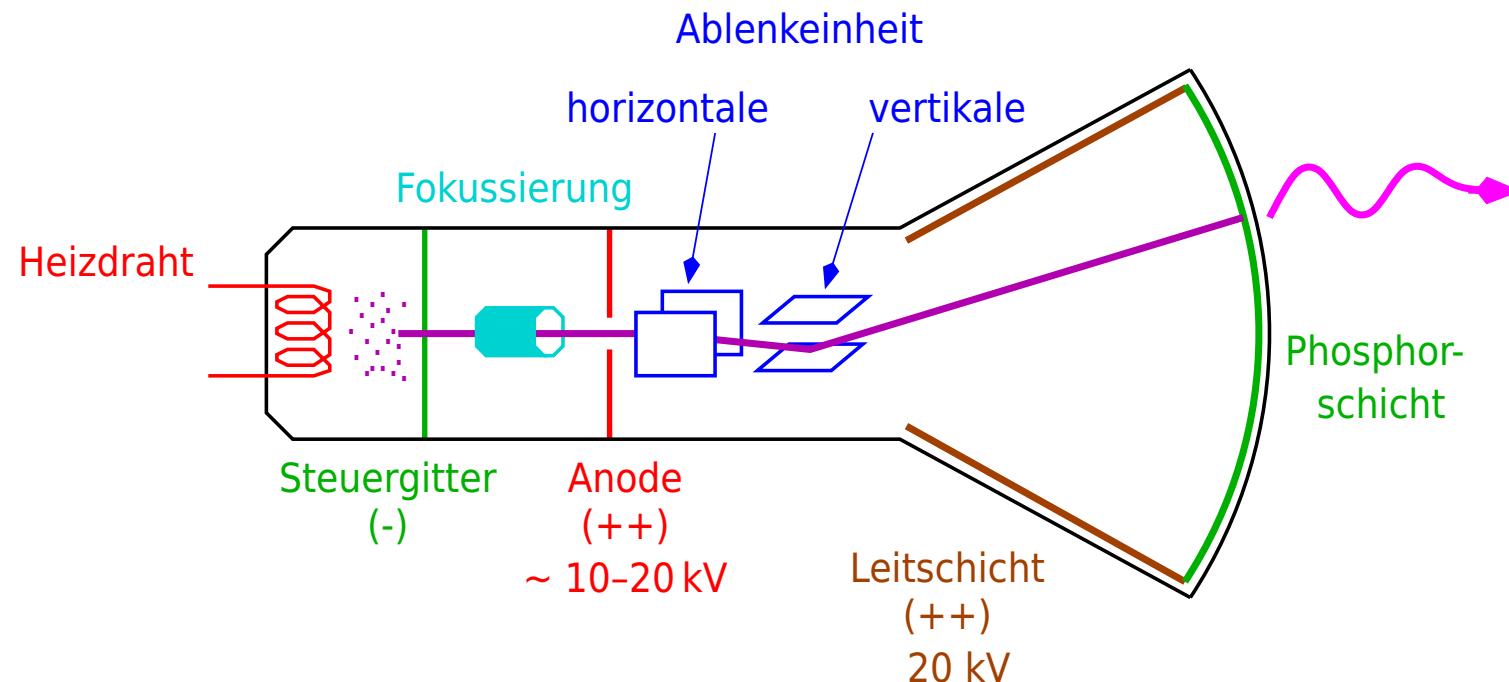
2.1.1 Vector-Displays

Kathodenstrahl-Röhre (Braunsche Röhre, CRT)

Karl Ferdinand Braun
* 1850, Fulda
† 1918, New York
Physiker



Quelle: https://commons.wikimedia.org/wiki/File:Ferdinand_Braun.jpg



Heizdraht: erzeugt eine Elektronenwolke

Anode: beschleunigt die Elektronen in Richtung Bildschirm

Fokussierung: bündelt den Elektronenstrom zu einem feinen Strahl

Steuergitter: regelt die Anzahl der Elektronen im Strahl (Intensität)

Ablenkeinheiten: verändern die Richtung des Strahls (elektrostatisch/magnetisch)

Phosphorschicht: sendet beim Auftreffen des Elektronenstrahls Licht aus

Leitschicht: ermöglicht Abfließen der Elektronen und verhindert dadurch zu starke negative Aufladung der Phosphorschicht

Einige Bemerkungen zur Lichterzeugung

Fluoreszenz: entsteht durch das „Zurückfallen“ angeregter Elektronen aus sehr instabilen Zuständen

⇒ sehr kurzzeitiger Effekt

Phosphoreszenz: entsteht durch das „Zurückfallen“ angeregter Elektronen aus etwas stabileren Zuständen

⇒ von längerer Dauer (10-60 µs bis einige s)

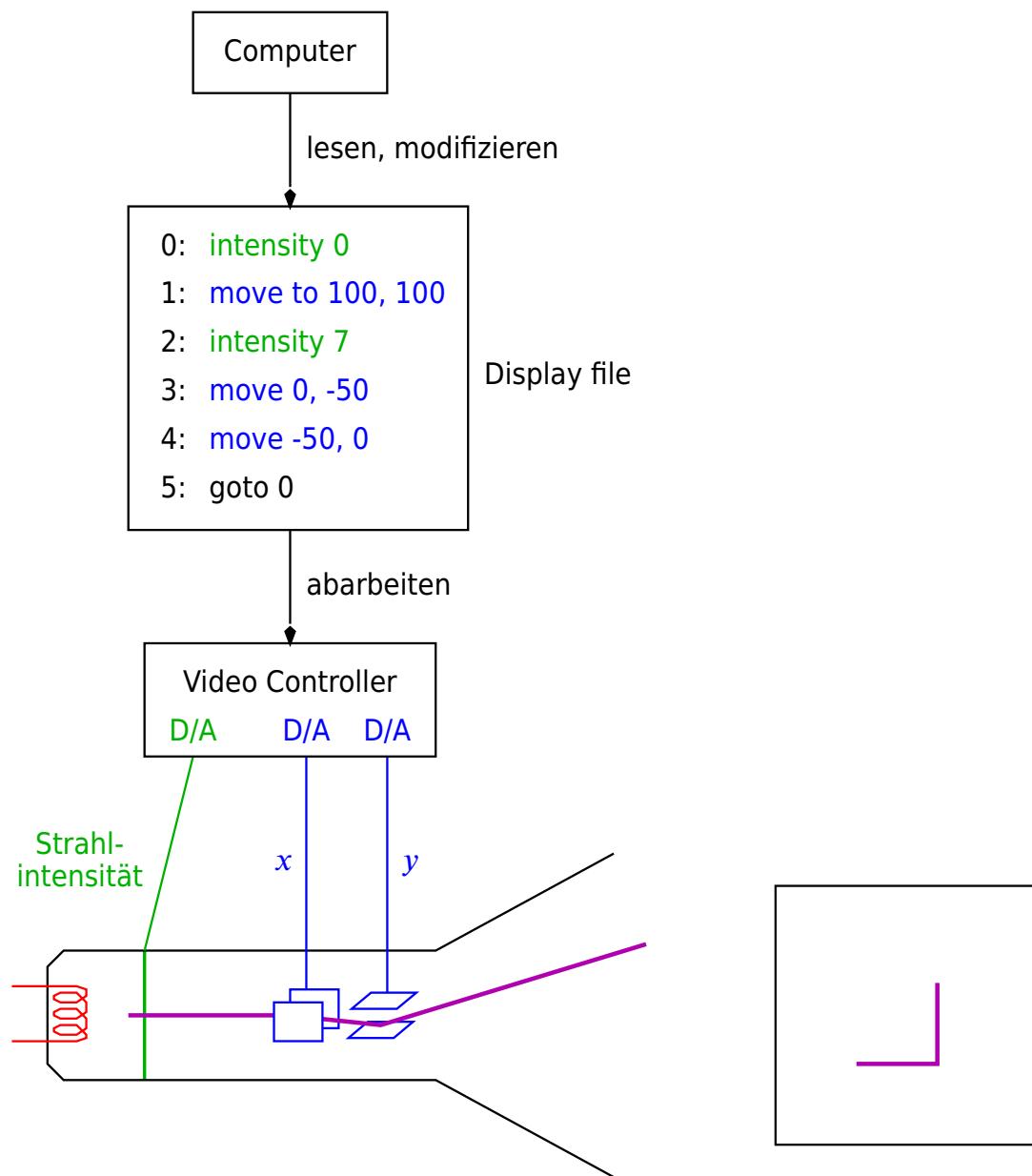
Bei zu niedriger **Bildwiederholfrequenz** nimmt das Auge noch das **Flackern** des Bildes wahr.

⇒ Stress, Kopfschmerzen

Bei zu hoher Bildwiederholfrequenz kann der Phosphor beschädigt werden.

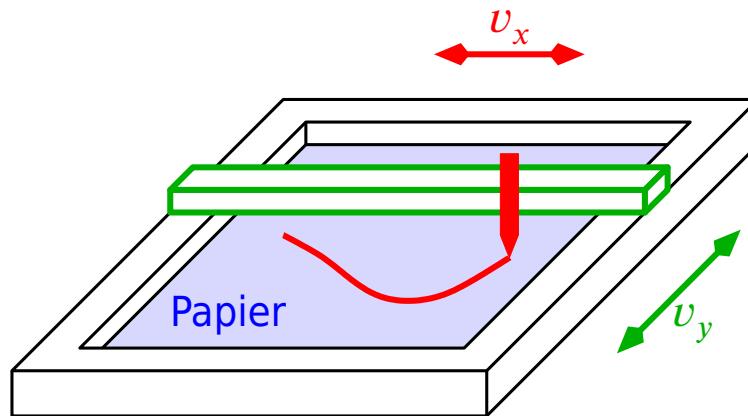
empfohlene Werte: ≥ 60 Bilder je Sekunde

Vector Display

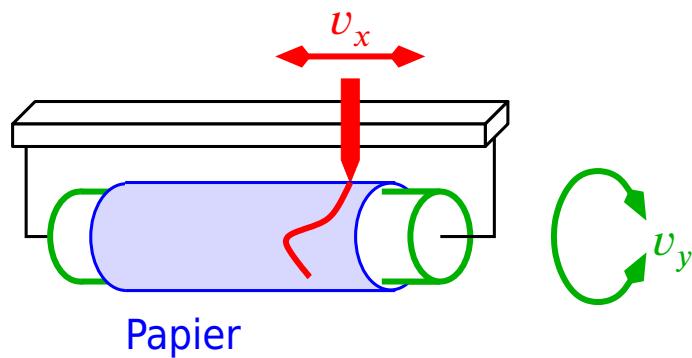


2.1.2 Plotter

Flachbett-Plotter



Trommelplotter

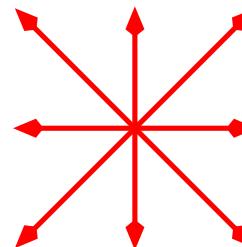


„**idealer Plotter**“: kann beliebige Kurven $\varphi(t) = (x(t), y(t))$ zeichnen durch geeignete Wahl von v_x und v_y :

$$v_x = \dot{x}(t), \quad v_y = \dot{y}(t)$$

„**realer Plotter**“: Aus Genauigkeitsgründen werden i. Allg. **Schrittmotoren** verwendet.

- ⇒ x und y können pro Zeiteinheit nur jeweils um $-\Delta$, 0 oder $+\Delta$ geändert werden.
- ⇒ nur 8 „Hauptrichtungen“:



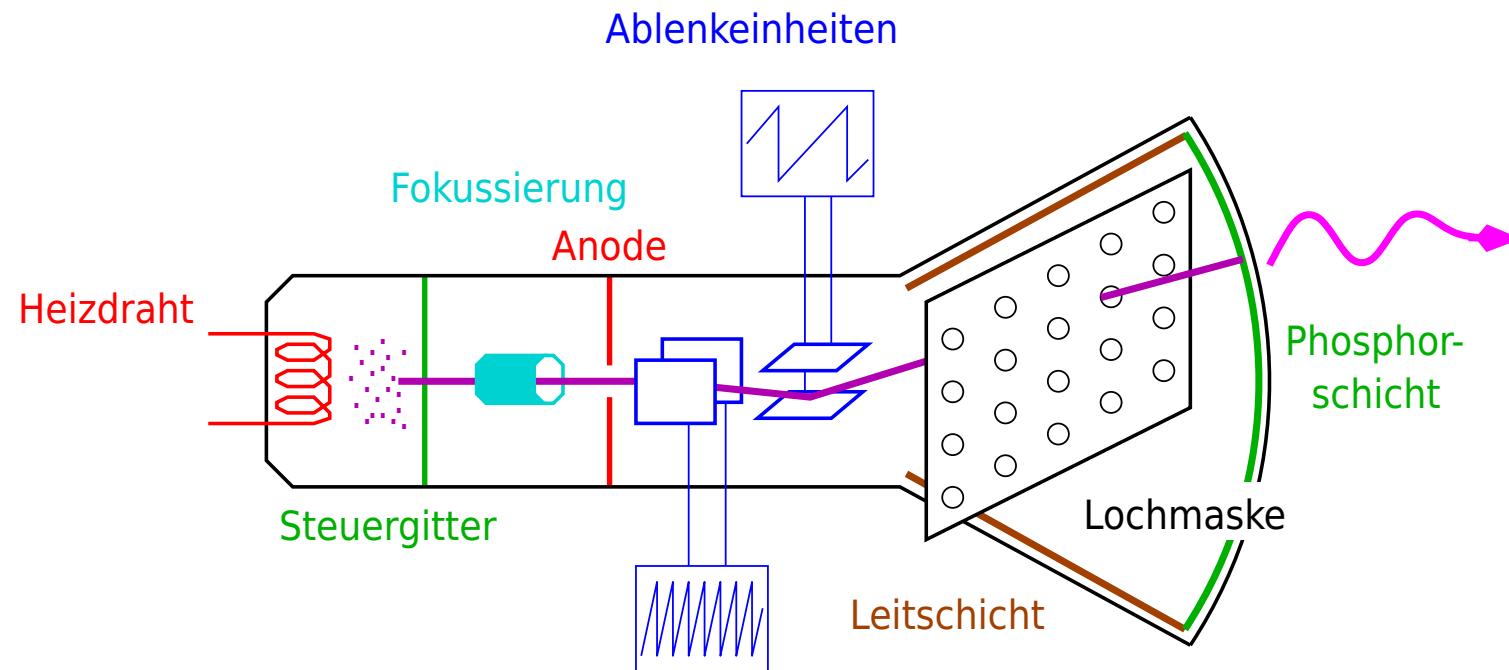
- ⇒ Kurven und Geraden mit anderen Richtungen werden approximiert:



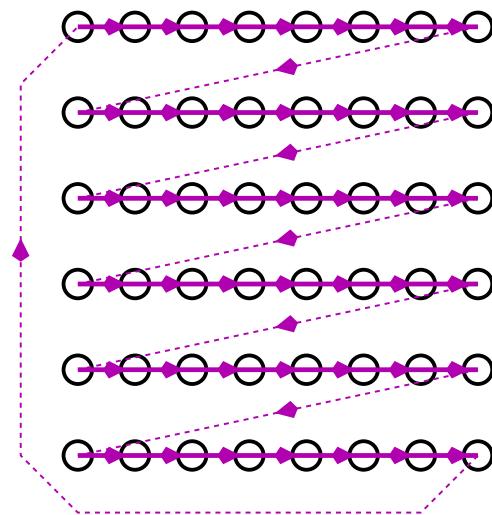
2.2 Raster-orientierte Ausgabegeräte

2.2.1 Kathodenstrahlröhre mit Lochmaske

Raster-Bildschirm

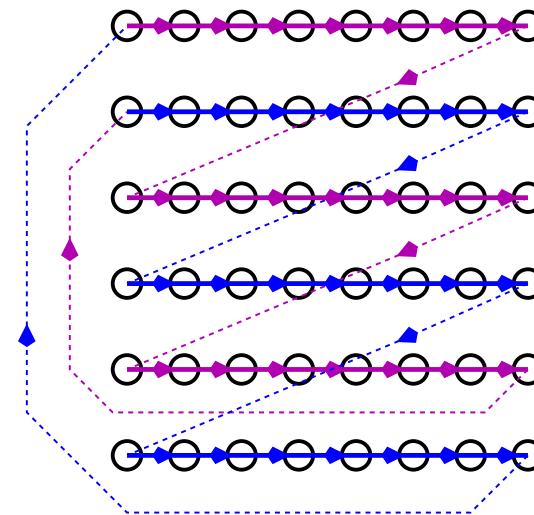


„non-interlaced“:

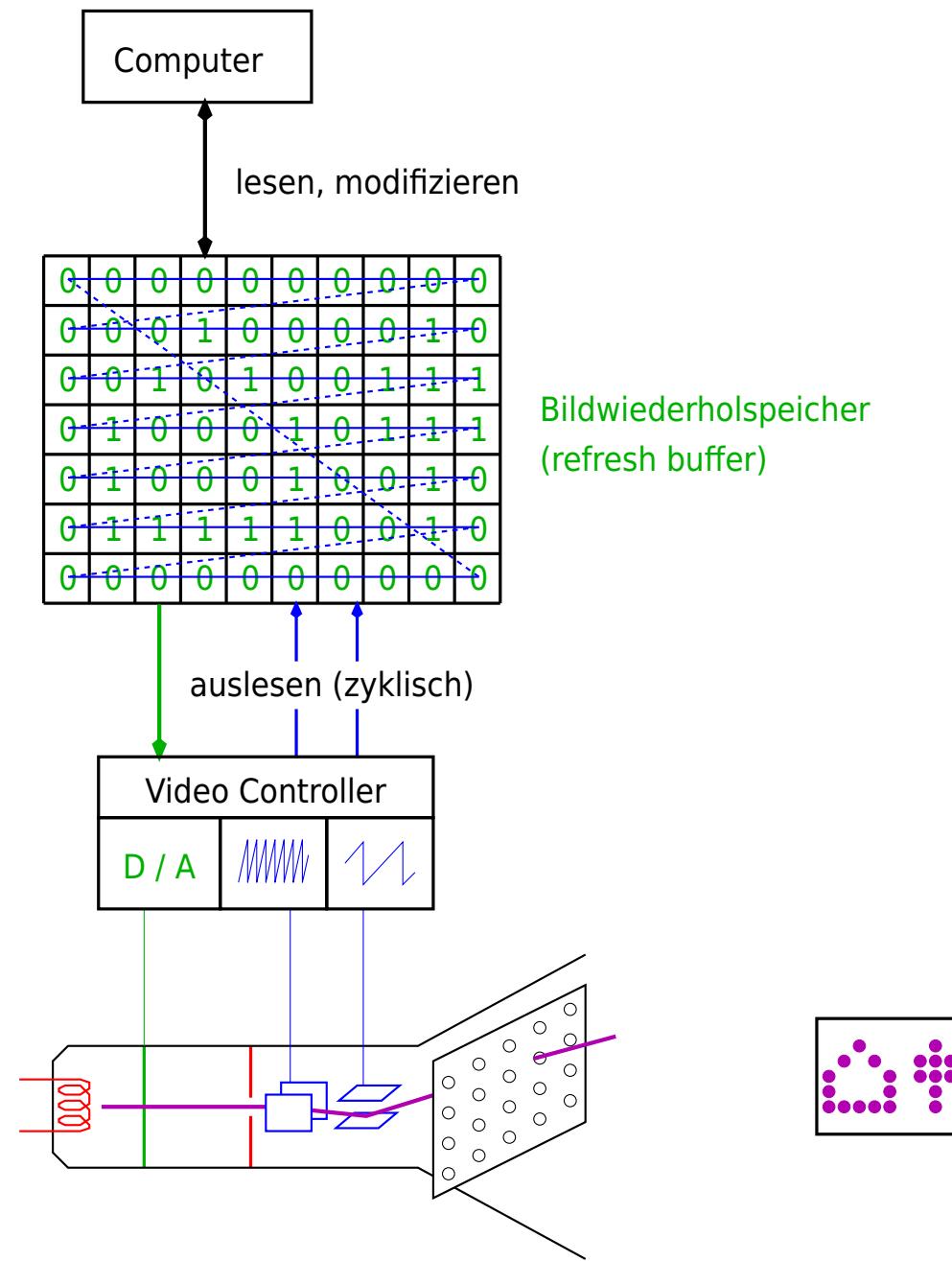


„interlaced“:

erstes Halbbild
zweites Halbbild



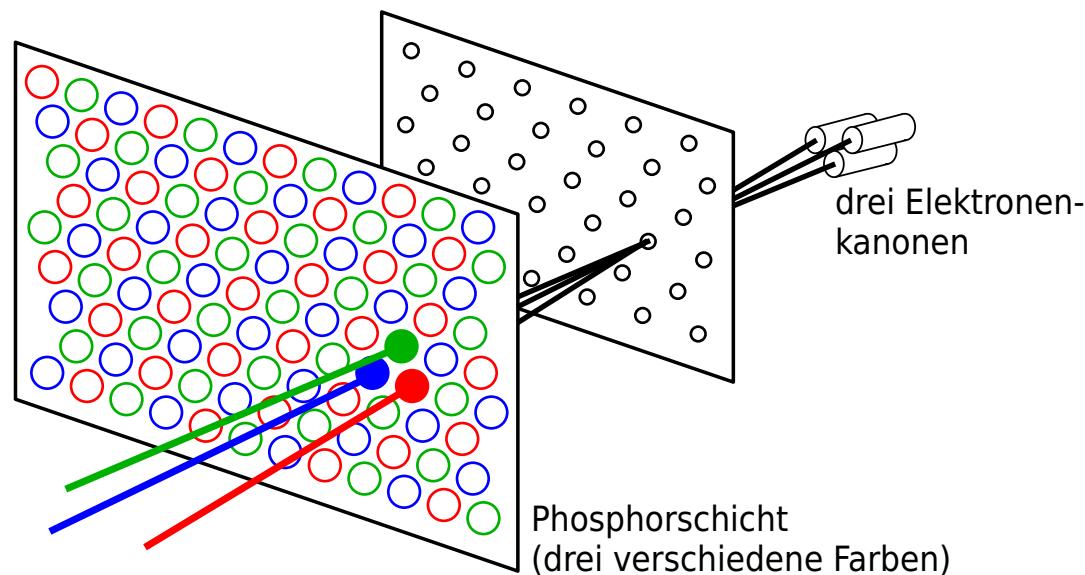
Raster-Display

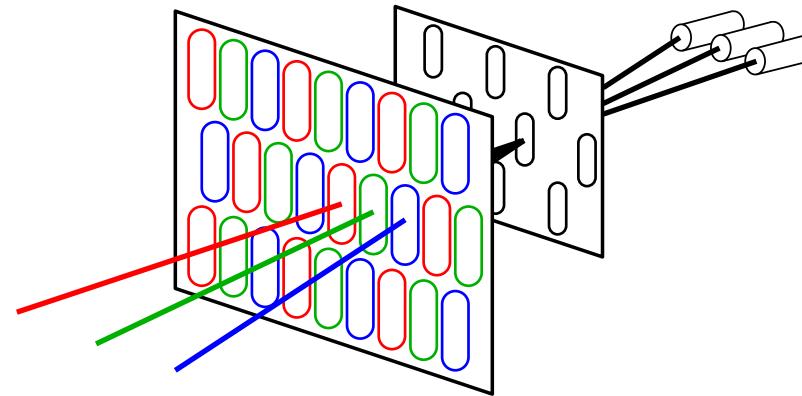


Farb-Raster-Bildschirm

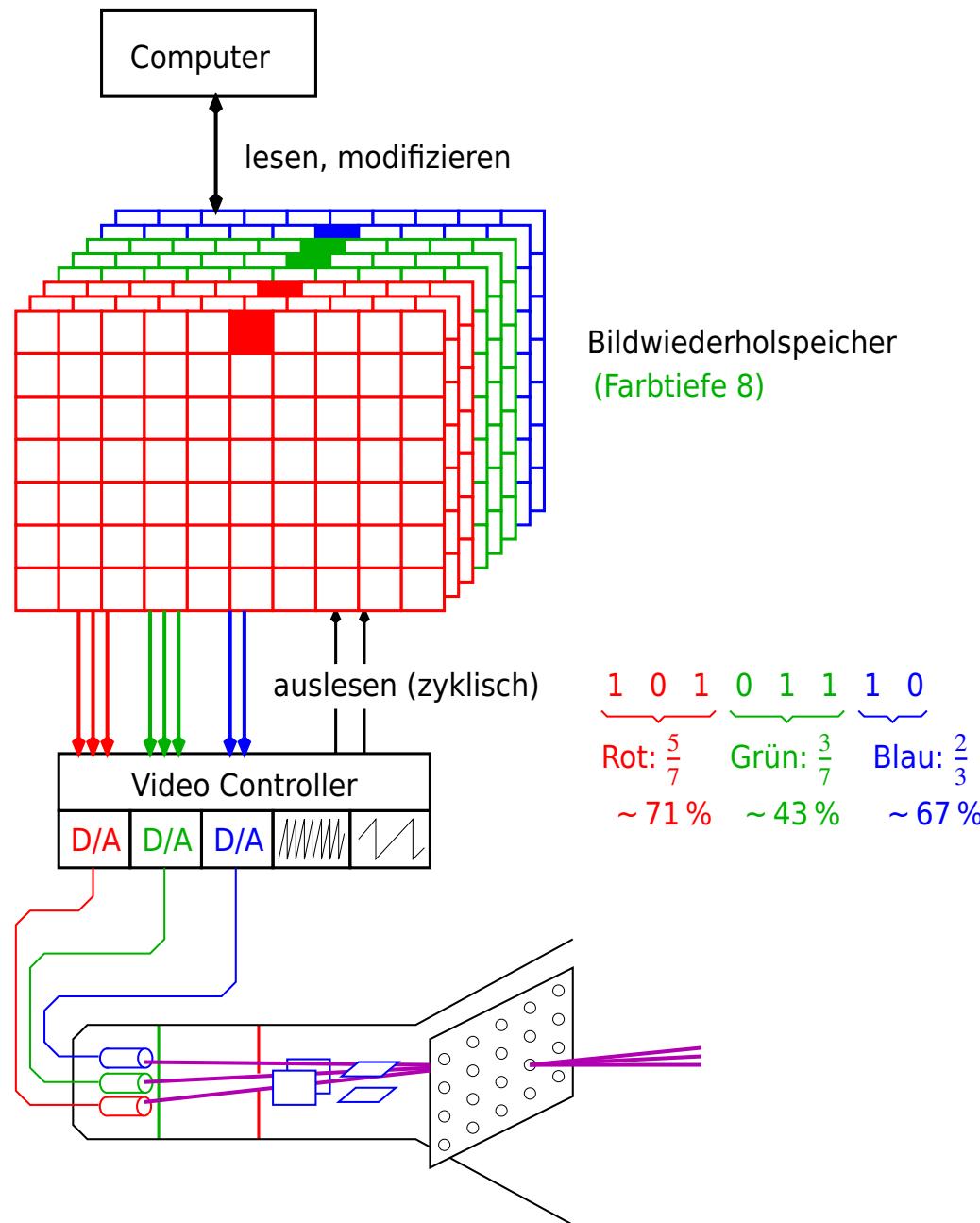
häufig **RGB**: „additive Farbmischung“ der „Grundfarben“ **Rot**, **Grün** und **Blau** mit geeigneten Intensitäten
(mehr zu Farbmodellen in Abschnitt 7.1)

Delta-Delta-Röhre (Phosphormuster Δ , Kanonenanordnung ∇):



Alternative Anordnung: Inline-Röhre:

Farb-Raster-Display



Farb-Raster-Display mit Farbtabelle

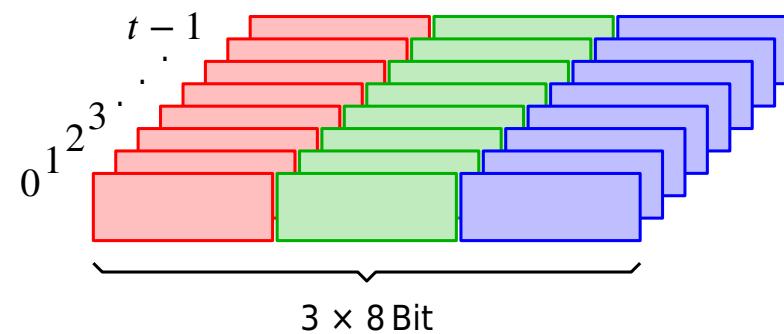
Stufenlose Farbübergänge erfordern mindestens **8 Bit je Grundfarbe**.

⇒ 24 Bit Puffertiefe $\triangleq 2^{24} \approx 16,7$ Mio. mögliche Farben

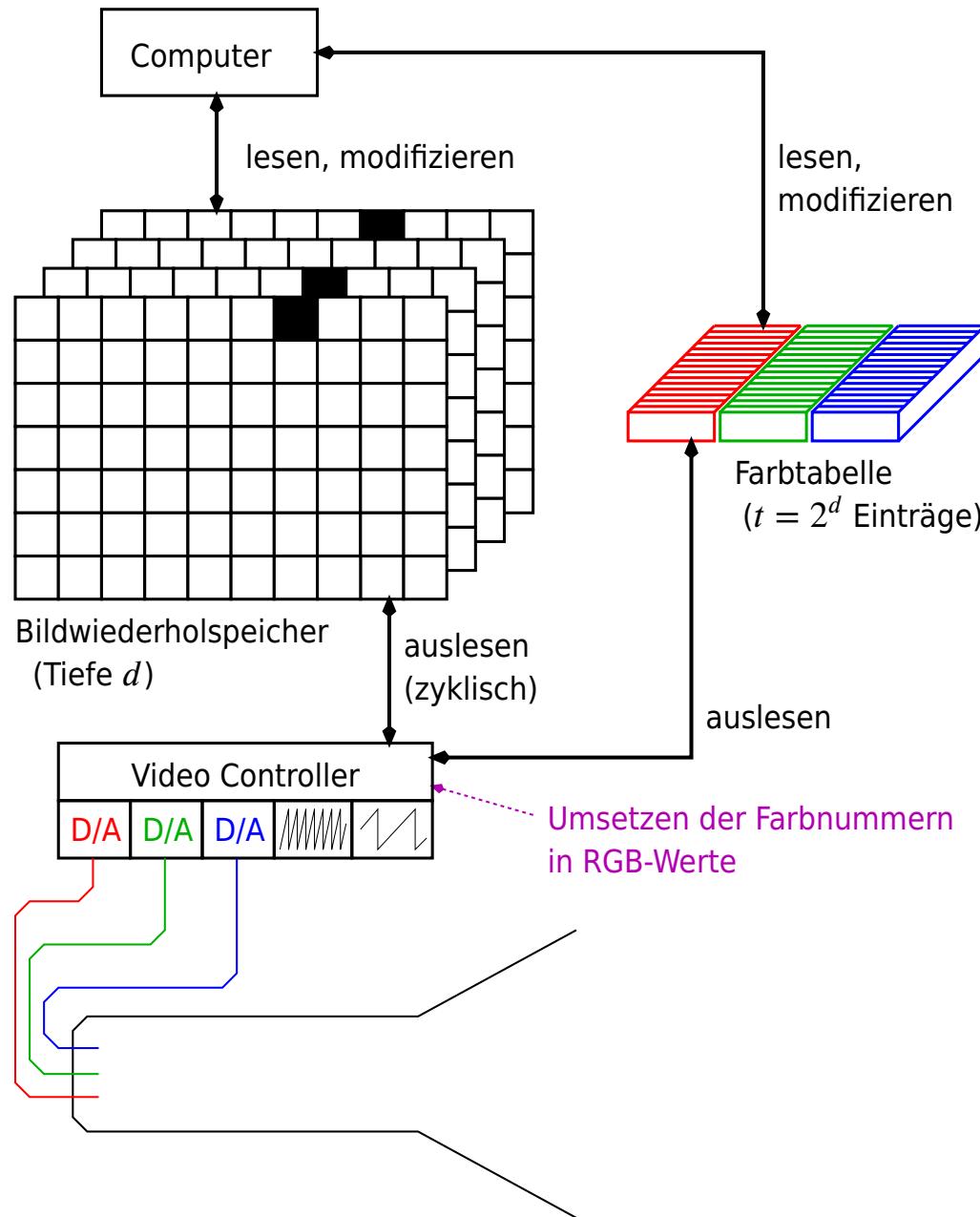
aber: In einem Bild kommen meist wesentlich weniger Farben **gleichzeitig** vor.

(typisch: einige Hundert oder Tausend)

⇒ **Farbtabelle (colour lookup table)**

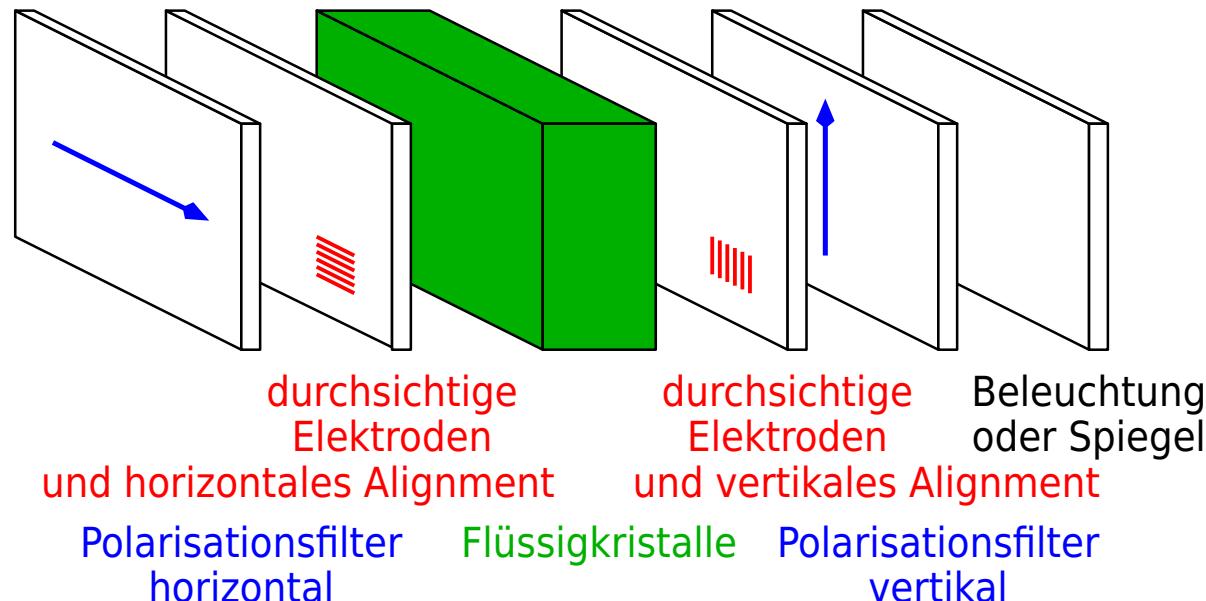


- Eintrag i der Farbtabelle, $0 \leq i < t$, enthält die RGB-Anteile der i -ten Farbe,
- Zelle (x, y) des Bildwiederholspeichers enthält die **Nummer** der an dieser Stelle darzustellenden Farbe.

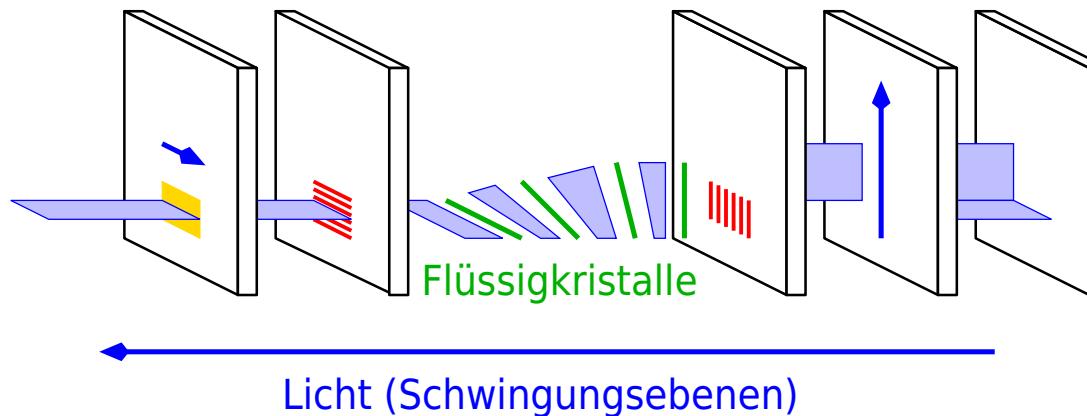


2.2.2 Bildschirme ohne CRT

Flüssigkristall-Bildschirm (liquid crystal display, LCD)



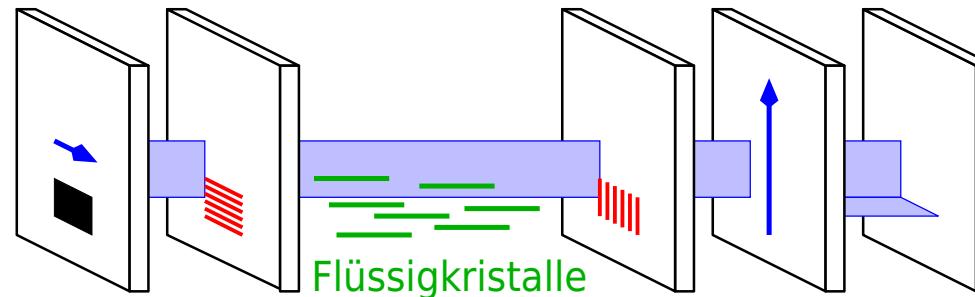
ohne Spannung zwischen den Elektroden:



- In der Nähe der Elektrodenplatten werden die Flüssigkristalle durch die Oberflächenstruktur (**alignment layers**) vertikal bzw. horizontal ausgerichtet.
- Dazwischen ergibt sich eine „Verdrehung“ der Ausrichtung (Helix-ähnlich).
- Dadurch wird auch die (zunächst vertikale) Polarisationsebene des durchgehenden Lichts um 90° gedreht.
- Damit kann das Licht auch den horizontalen Polarisationsfilter passieren.

⇒ Bereich der Elektroden erscheint hell

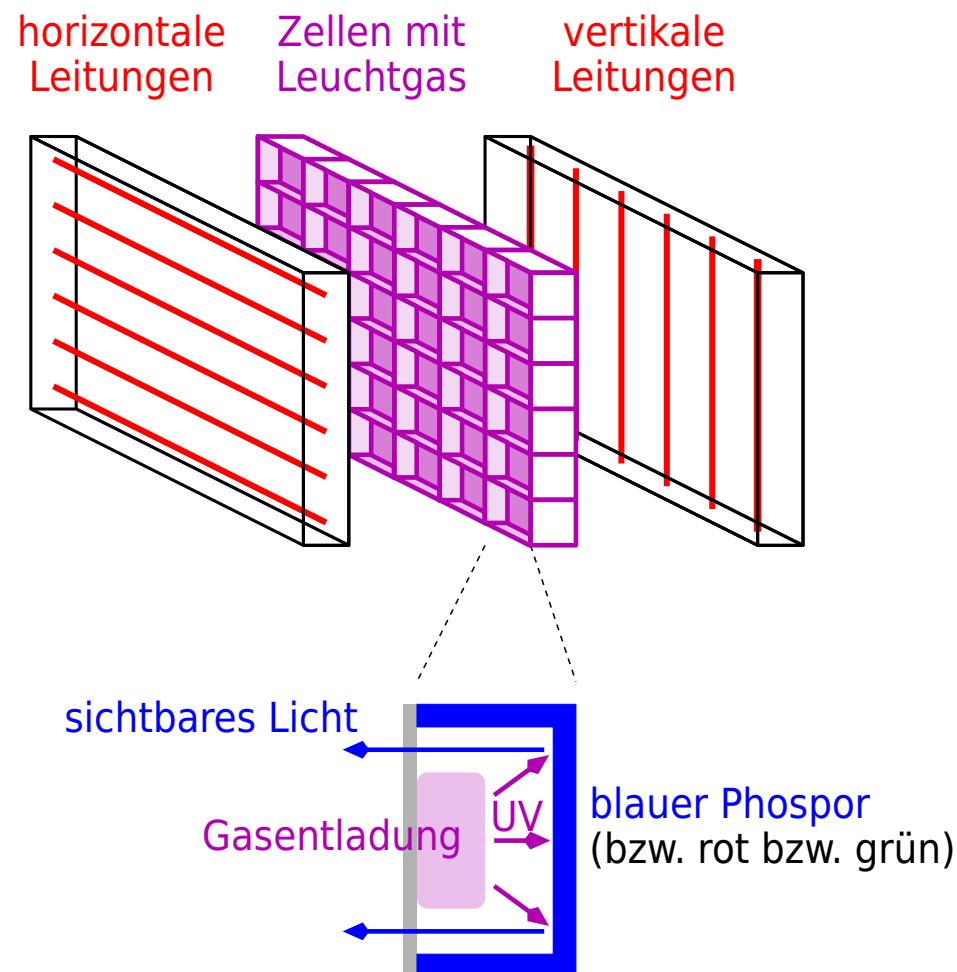
mit Spannung zwischen den Elektroden:



- Flüssigkristalle richten sich gemäß der Spannung aus
- Polarisationsebene des durchgehenden Lichts wird nicht gedreht
- Licht kann den horizontalen Filter nicht passieren

⇒ Bereich der Elektroden erscheint dunkel

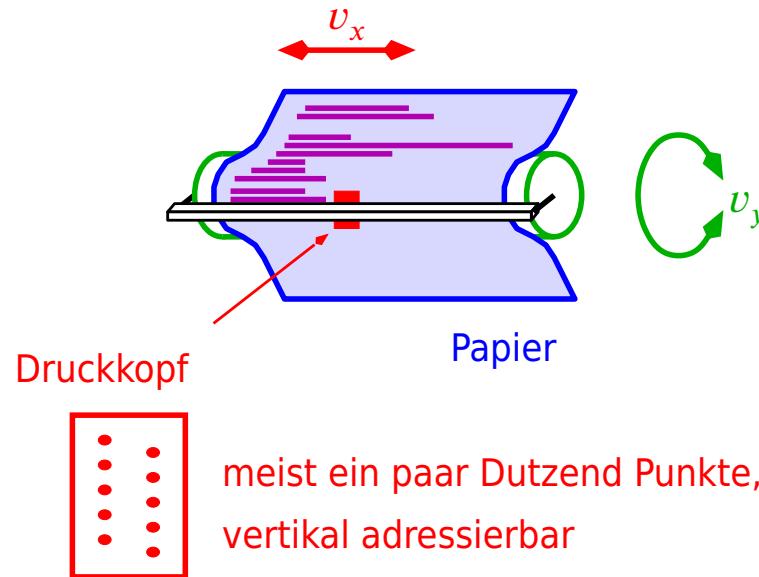
Plasma-Bildschirm



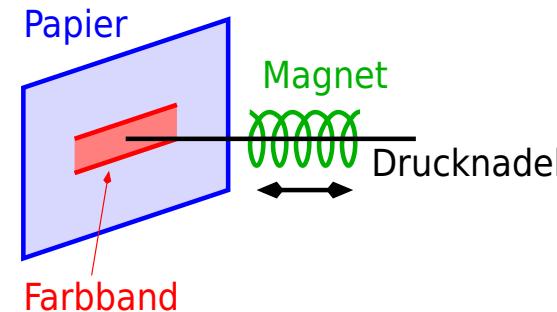
Bemerkungen 2.2:

1. Bei LCDs und Plasma-Bildschirmen sind prinzipiell nahezu beliebige Elektroden- bzw. Kammer-Formen möglich (z. B. für Skalen in Messgeräten). In den meisten Anwendungen wird allerdings eine Raster-Geometrie verwendet.
 2. LCDs und Plasma-Bildschirme haben CRT-basierte Bildschirme weitgehend verdrängt.
-

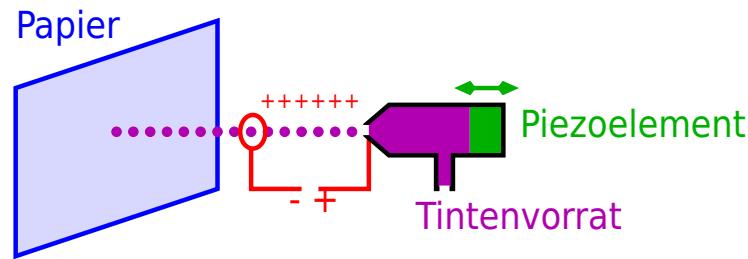
2.2.3 Drucker



Nadeldrucker (im Privatbereich kaum noch eingesetzt):

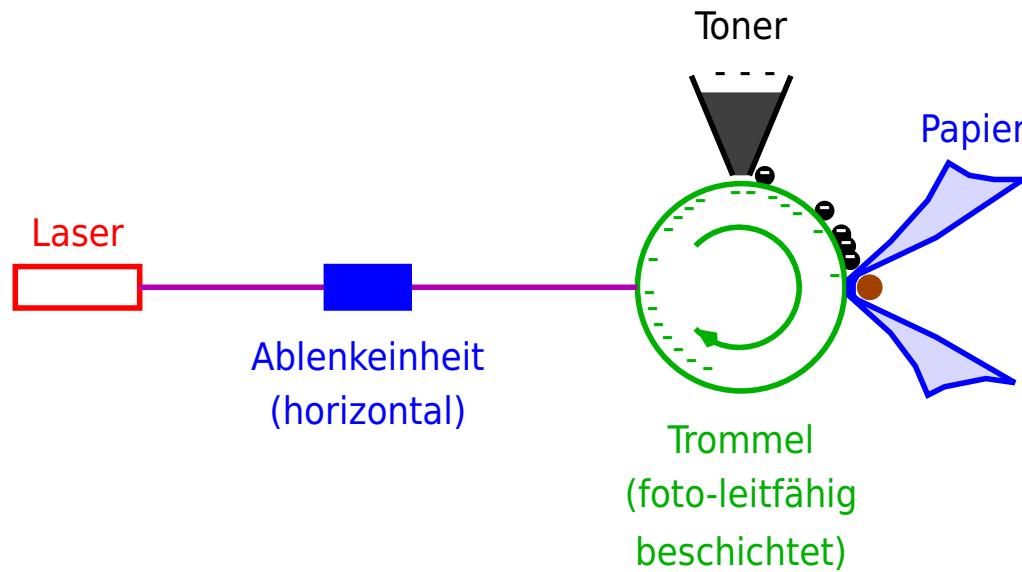


Tintenstrahldrucker:



(Druckimpuls alternativ durch Heizelement: **Bubble Jet**)

Laserdrucker:



- Wo der Laserstrahl auf die Trommel trifft, wird diese leitfähig, und die negative Ladung wird neutralisiert.
 - Der negativ geladene Toner bleibt nur an den ungeladenen Stellen der Trommel hängen und wird auf das Papier übertragen (und danach mit Wärme **fixiert**).

2.3 Vergleich

Linien-orientierte Ausgabegeräte	Raster-orientierte Ausgabegeräte
<ul style="list-style-type: none"> • Bild wird aus Linien aufgebaut + bei Vektor-Display Dynamik relativ einfach realisierbar (bei Papier-Ausgabe keine Dynamik möglich) - Ablenkeinheit/Motor muss sehr präzise sein - Farben nur sehr eingeschränkt möglich - keine Schattierung von Flächen - verdeckte Linien/Flächen müssen vor der Darstellung bestimmt werden 	<ul style="list-style-type: none"> - Linien müssen in Punkte zerlegt („gerastert“) werden • geringe Änderungen erfordern i. Allg. vollständigen Neuaufbau des Bildes + relativ einfache (billige) Ausführung genügt + nahezu uneingeschränkte Farbpalette + diverse Lichteffekte darstellbar + verdeckte Linien/Flächen können entweder berechnet oder durch einfache Hardware entfernt werden

heute eindeutig dominierend

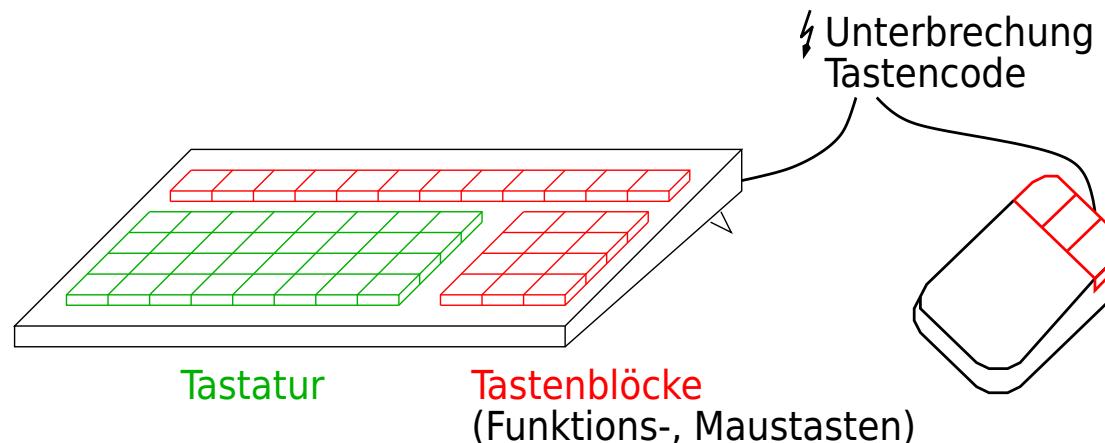
2.4 Eingabegeräte

Logische Eingabegeräte

abhängig vom **Typ** der Eingabe:

string: Zeichenketten, Befehle, ...

⇒ Tastatur

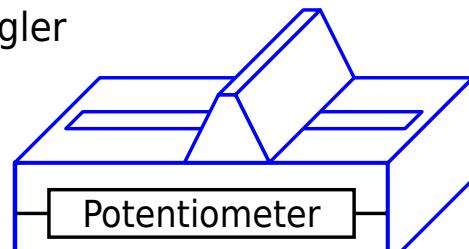


choice: Auswahl unter einer Anzahl von Möglichkeiten

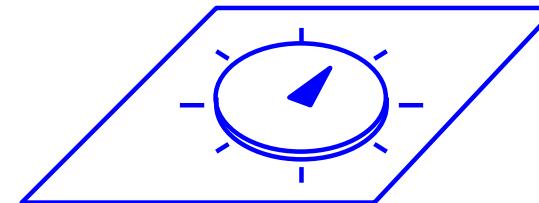
⇒ Tastenblock, ...

valuator: reelle Zahlen

⇒ Schiebe-, Drehregler



Schieberegler



Drehregler

- müssen i. Allg. abgefragt werden
- Einsatz u. a. bei Spielen und in der Konstruktion

locator: Position

⇒ Maus, Touchscreen, ...

⇒ Abschnitt 2.4.1

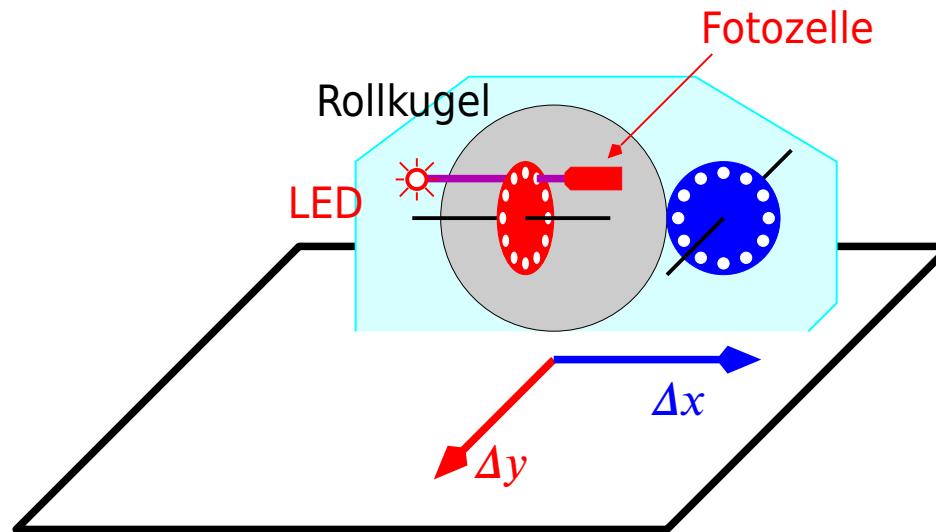
image: Farb-/Graustufenbilder

⇒ Scanner

Bemerkung 2.3: Im Prinzip könnte jedes dieser Eingabegeräte alle Eingabetypen übernehmen (string, choice, ...). Es gibt aber meist eine „natürliche“ Funktion.

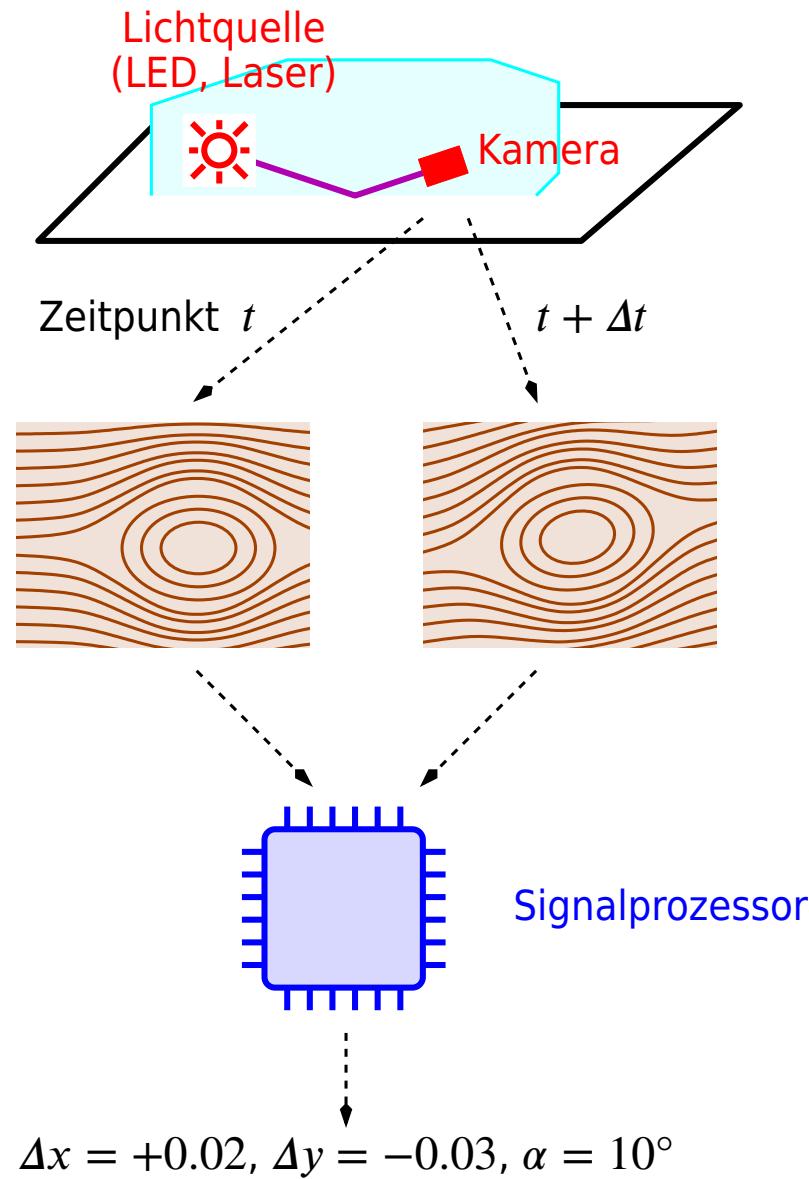
2.4.1 Eingabegeräte für Positionen

Mechanische Maus



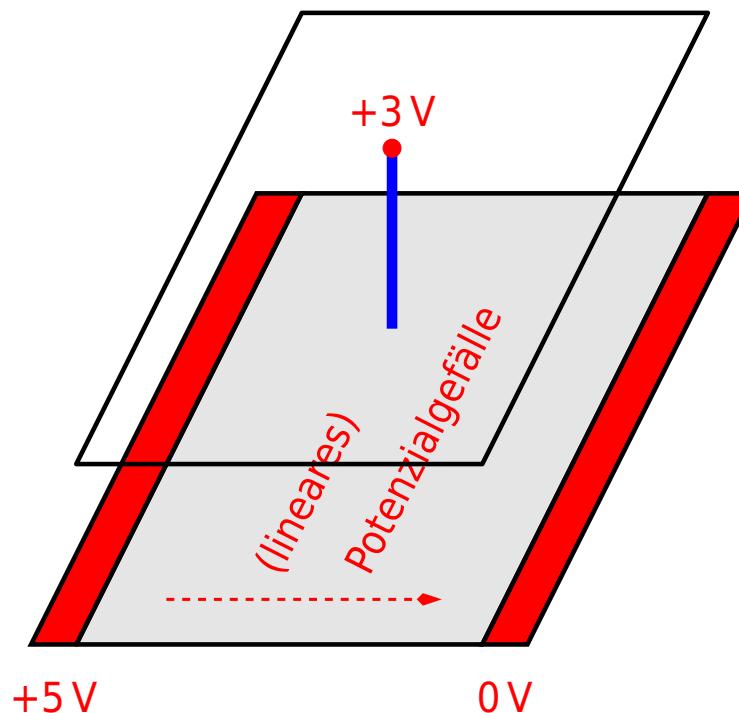
- Bewegung der Rollkugel wird über zwei Räder (x -/ y -Richtung) in optische (und dann in elektrische) Impulse übertragen
- **nur Positionsänderung bestimmbar, nicht die momentane Position!**
 - ⇒ Der Computer (oder Software auf der Maus) muss die Position der Maus verwalten und bei jeder Positionsänderung aktualisieren.
- weitgehend von optischen Mäusen verdrängt

Optische Maus



- wieder nur Positionsänderungen messbar

Touchscreen (resistiv)



- Flexible Frontschicht und starre Hinterschicht sind leitfähig und durch Abstandshalter getrennt.
- In der hinteren Schicht wird ein Potenzialgefälle in x -Richtung erzeugt.
- Durch Druck (z. B. mit dem Finger oder einem Stift) kommen die beiden Schichten in Kontakt, und das Potenzial auf der vorderen Schicht ergibt die x -Position des Berührpunktes.
(hier: 40 % vom linken Rand)
- Für die y -Position wird ein vertikales Potenzialgefälle in der vorderen Schicht erzeugt und am „hinteren“ Berührpunkt gemessen.

Bemerkungen 2.4:

1. „natürliche“ Positionierung bzw. Auswahl möglich, wenn beide Schichten vor dem Bildschirm liegen
 2. Bei **kapazitiven** Touchscreens wird die Veränderung eines elektrischen Feldes durch den Finger gemessen.
⇒ nicht mit Handschuhen verwendbar
- Die Position des Fingers/Stiftes kann auch akustisch oder optisch ermittelt werden.
-

2.5 Eingabemodi

2.5.1 Sampling

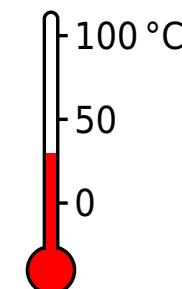
Beim **Sampling** wird der Wert des Eingabegeräts (z. B. die Position der Maus) laufend abgefragt.

Modell



Beispiel 2.5: Einstellen eines reellen Parameters mit der Maus (mit Anzeige des gerade aktuellen Parameterwertes):

wiederhole
 lies Position der Maus
 rechne Mausposition in Temperatur um
 stelle diese Temperatur dar
bis Maustaste gedrückt // *Temperatur endgültig akzeptiert*



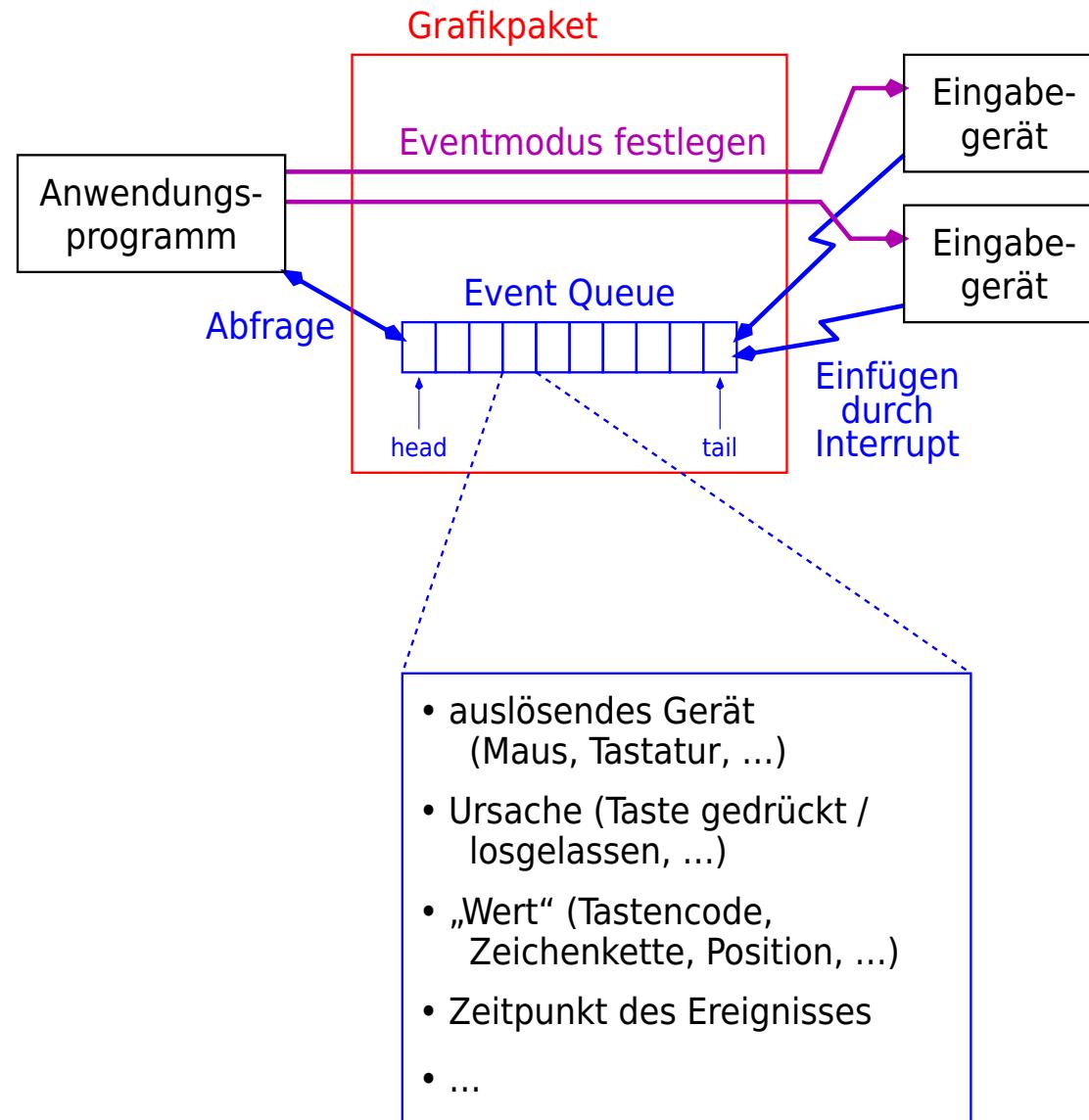
2.5.2 Ereignisgesteuert

Bei der ereignisgesteuerten (**event driven**) Eingabe wird das Eingabegerät nur dann abgefragt, wenn das Gerät das Eintreffen eines bestimmten Ereignisses (**Event**) gemeldet hat.

Beispiele für Ereignisse

Gerät	Ereignis	gelieferter Wert
Tastatur im „Direktmodus“	irgendeine Taste gedrückt	Code der Taste
Tastatur im „Stringmodus“	RETURN-Taste gedrückt	die zuvor getippte Zeichenkette
Maus	Maustaste gedrückt oder losgelassen	„Zustand“ der Maus: <ul style="list-style-type: none"> • Position • welche Taste wurde betätigt? • Zustand aller Tasten
(Maus)	Position der Maus hat sich seit dem letzten Event um mehr als Δ geändert	„Zustand“ der Maus: <ul style="list-style-type: none"> • Position • Zustand aller Tasten
Tastenblock (Choice)	wie Tastatur im Direktmodus	
Regler (Valuator)	wie (Maus)	

Modell



Beispiel 2.6: Polygon mit Hilfe der Maus definieren:

- neue Ecke wird mit der linken Maustaste gesetzt,
- die letzte Ecke mit der rechten Maustaste,
- die Taste „?“ der Tastatur bringt ein Menü mit Erklärungen

wiederhole

 warte auf einen Event
 auswählen (auslösendes Gerät)

Tastatur:

 wenn Tastencode = „?“
 zeige Hilfemenü an

Maus:

 wenn eine Maustaste gedrückt wurde
 lies zugehörige Position der Maus
 rechne Mausposition in Punkt um
 füge Punkt in Eckenliste ein
 verbinde mit dem vorigen Punkt, falls vorhanden

bis Event entstand durch Drücken der rechten Maustaste
verbinde den letzten mit dem ersten Punkt

2.5.3 Vergleich der Verfahren

- Beim Sampling wird ein großer Teil der Gesamtzeit in der Abfrageschleife verbracht. Die meisten der eingelesenen Werte sind allerdings nutzlos, da sie durch nachfolgende Werte überschrieben werden.
⇒ Prozessorleistung wird blockiert (z. B. bei Mehrprozessbetrieb).
- Bei ereignisgesteuerter Eingabe wird nur dann Prozessorleistung benötigt, wenn „interessante“ Eingaben vorliegen. Bis zum Eintreffen des Ereignisses kann der Prozessor für andere Aufgaben genutzt werden.
- Das **Anwendungsprogramm** kann nur dann ereignisgesteuerte Eingaben verwenden, wenn die Verwaltung der Event-Queue und das Bildschirmecho
 - selbstständig vom Grafikpaket durchgeführt werden oder
 - durch Nebenläufigkeit (z. B. in der Programmiersprache) realisiert werden können.
- Die Tastatur wird meist ereignisgesteuert, die Maus je nach Anwendung ereignisgesteuert oder im Sampling-Modus betrieben.

2.6 Regeln für den Entwurf von Interaktion

1. Die Interaktionsfolgen sollen **einfach** sein.

nicht: `CTRL-c h` für Hilfestellung

2. Die Interaktionsfolgen sollen **konsistent** sein, d. h.:

- a) Gleiche Interaktionsfolgen haben auch die gleiche Wirkung.

nicht: Im Programmteil „Entwurf“ bewirken die Tasten

`CTRL-q` Programmende mit Abspeichern der neuesten Daten,

`CTRL-x` Programmende ohne Sicherung,

im Programmteil „Auswertung“ ist es umgekehrt.

- b) Die gleiche Wirkung kann mit identischen Interaktionsfolgen erzielt werden.

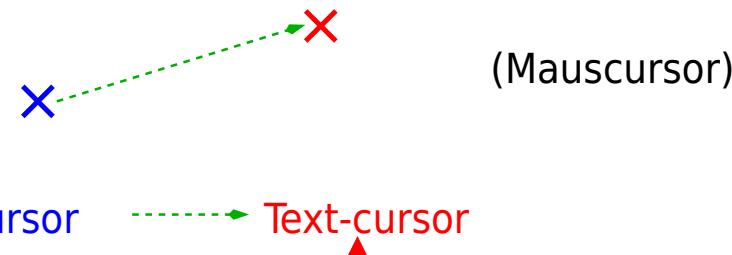
nicht: Im Programmteil „Entwurf“ wird beim Verlassen mit `CTRL-x` nochmals nachgefragt, ob die neuesten Daten wirklich weggeworfen werden sollen, im Programmteil „Auswertung“ nicht.

3. Der Benutzer sollte stets eine **Rückmeldung (Feedback)** für seine Eingaben, deren Wirkung und den aktuellen „Zustand“ des Systems erhalten.

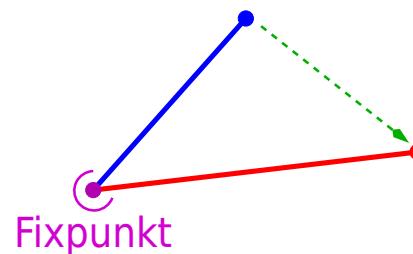
a) „**Echo**“: Eingaben von der Tastatur sollten i. Allg. auch auf dem Bildschirm erscheinen, die aktuelle Position der Maus und ähnlicher Eingabegeräte sollte auf dem Bildschirm sichtbar sein.

Beispiele für Echoes:

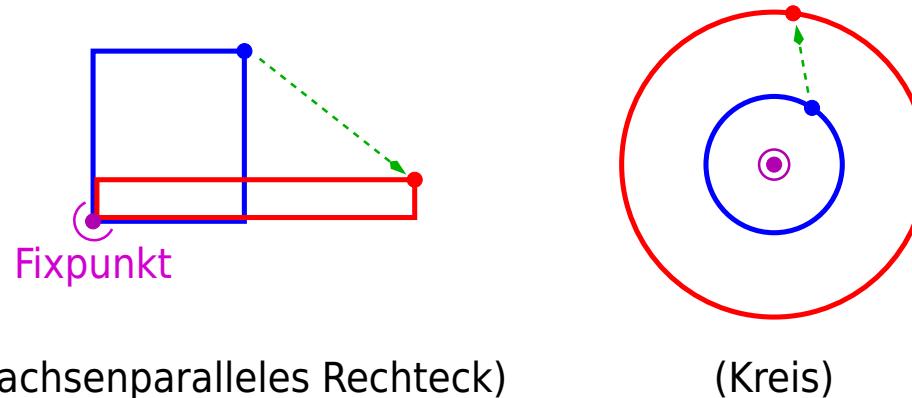
- Cursor (für Text- und Positioneingaben)



- Gummiband (Eingabe von Strecken und Polygonzügen)

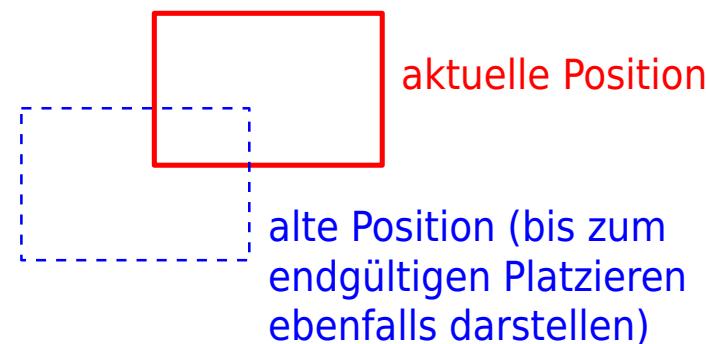


- Gummiumriss (Eingaben von Flächen bestimmter Form)



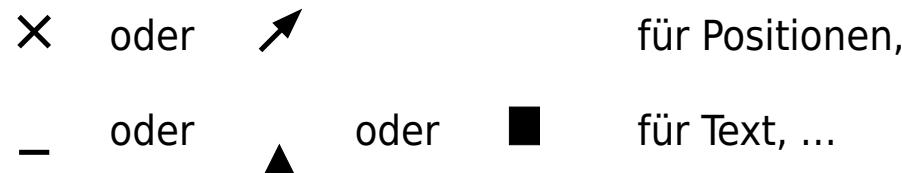
b) nach Möglichkeit immer die neuesten Daten darstellen

Beispiel: Verschieben eines Objekts:



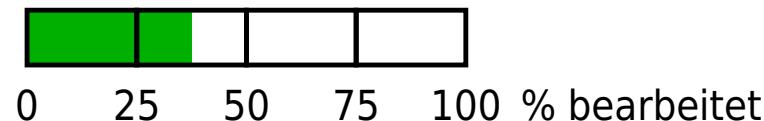
- c) Aus dem Echo sollte schon ersichtlich sein, welcher Typ von Eingabe erwartet wird.

Beispiel: Umschalten der Cursor-Form:



- d) Bei längeren Wartezeiten sollte angezeigt werden, wie lange es noch dauert.

Beispiel:



4. Zu jedem Zeitpunkt der Interaktionsfolge sollten die **möglichen Befehle und Optionen angezeigt** werden.

nicht: Programm, bei dem der Benutzer immer selbst wissen muss, in welchem „Zustand“ das System gerade ist und welche Eingabe erwartet wird (z. B. **vi**)

5. Der Benutzer sollte die Möglichkeit haben, **fehlerhafte Eingaben zurückzunehmen**.

- möglichst aussagekräftige Fehlermeldungen, evtl. mit Vorschlägen für die Beseitigung der Fehler
- Möglichkeit zum Zurücknehmen des letzten (oder der letzten k) Kommandos: **undo**
 - ⇒ Das Programm muss über Kommandos und Zwischenzustände Buch führen.
 - ⇒ sehr aufwändig auf „Pixelbene“ (nach der Rasterung); besser auf Objektebene realisierbar

6. Der Benutzer sollte **nicht zu viele Optionen einstellen müssen**.

nicht: Texteditor, bei dem man jedes Mal vor dem eigentlichen Arbeiten

- Zeichensatz, -höhe, -breite, ...
- Farbe für Hintergrund und Text, ...
- Tabulatorpositionen, ...
- ...

einstellen muss

(In der Regel sind einige der angebotenen Optionen für die meisten Anwendungen völlig überflüssig. Für nahezu alle anderen Optionen gibt es **Voreinstellungen**, die erst bei Bedarf – oder über ein **Profil** – geändert werden müssen.)

Beispiel 2.7: Benutzerführung mittels Menüs

- Anzeige der gerade verfügbaren Befehle und Optionen durch
 - Menü mit Stichworten und/oder
 - Palette mit Symbolen (**Icons**)
- Auswahl einer Aktion oder Option durch „Anklicken“ mit der Maus (o. ä.)
 - ⇒ Interaktionsfolge einfach
- Feedback durch Hervorheben des Menüpunkts, auf den die Maus gerade zeigt
- Werden durch eine Aktion weitere Optionen verfügbar, so erscheinen diese in einem zusätzlichen (**Pop-up-**)Menü.
- Sind für eine Aktion bestimmte Optionen nicht verfügbar, so können die entsprechenden Menüpunkte/Icons
 - entfernt werden oder
 - ihr Aussehen verändern (z. B. farbig ⇒ grau).

3 Scan Conversion

Inhalt

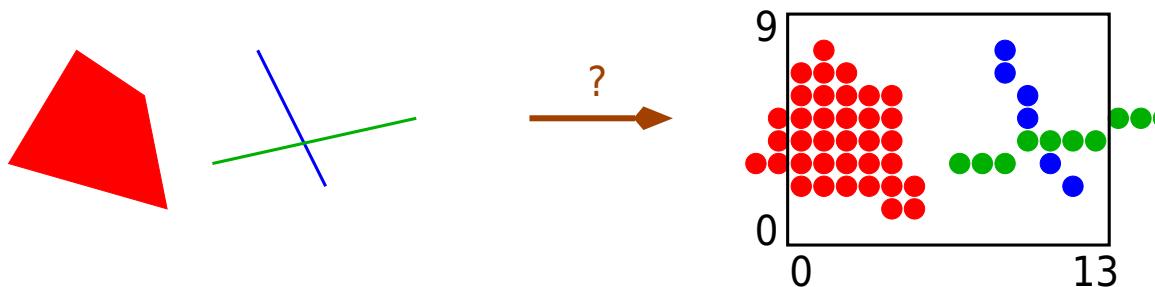
3.1 Scan Conversion für Strecken	3-8
3.1.1 Das naive Verfahren	3-10
3.1.2 Inkrementell mit reeller Rechnung	3-11
3.1.3 Inkrementell mit ganzzahliger Rechnung	3-15
3.2 Scan Conversion für Kreislinien	3-18
3.2.1 Zwei naive Verfahren	3-19
3.2.2 Ein inkrementeller Ansatz	3-21
3.3 Scan Conversion für Polygone	3-26
3.3.1 Polygone	3-27
3.3.2 Der Grundalgorithmus	3-29
3.3.3 Ein inkrementeller Ansatz	3-33
3.3.4 Der Flood Fill Algorithmus	3-40
3.4 Nochmals: Strecken und Kreise	3-43
3.4.1 Linien vorgegebener „Strichbreite“	3-43
3.4.2 Unterbrochene Linien	3-46
3.5 Räumliche und farbliche Auflösung	3-48
3.5.1 Räumliche Auflösung statt farblicher Auflösung	3-48
3.5.2 Farbliche Auflösung statt räumlicher Auflösung	3-52

3.6 Text 3-54

Problem: Gegeben ist eine Menge von „Objekten“.

- Linie von (7, 3) nach (16, 5)
- Linie von (9, 8) nach (12, 2)
- ausgefülltes Polygon mit den Eckpunkten (5, 1), (4, 6), (1, 8), (-2, 3)
- ...

Gesucht sind die zu jedem Objekt gehörenden Rasterpunkte (picture elements, „**Pixels**“).



„logischer Bildwiederholspeicher“,
Pixel Map,
Pixmap

Bemerkung 3.1: Die Pixmap muss nicht mit dem aktuellen (dem Video Controller zugänglichen) Bildwiederholspeicher übereinstimmen!

Anwendung 1: „Doppelpufferung“

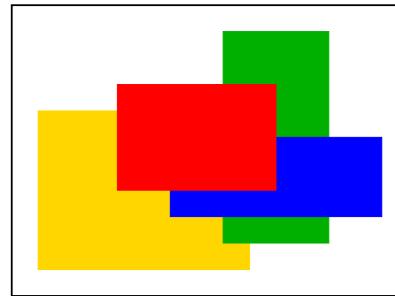
Problem: Gleichzeitiges Verändern und Auslesen des Bildwiederholspeichers führt zu unerwünschten Effekten.

(meist: Verzerrungen, Flackern)

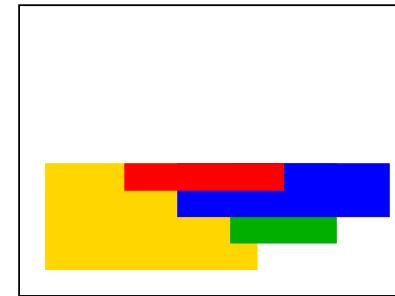
Beispiel: Aus einem (relativ komplexen) Bild soll ein Objekt gelöscht werden.

möglicher Ablauf ohne Doppelpufferung:

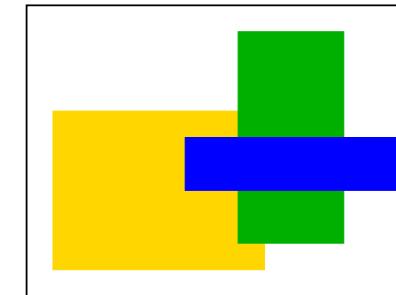
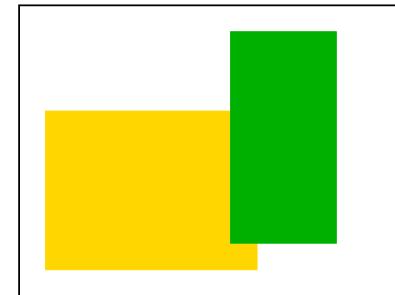
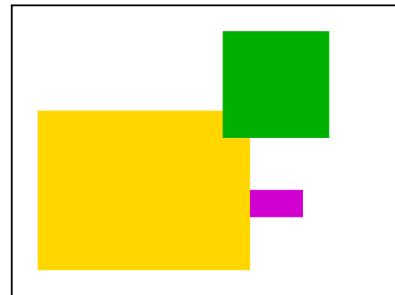
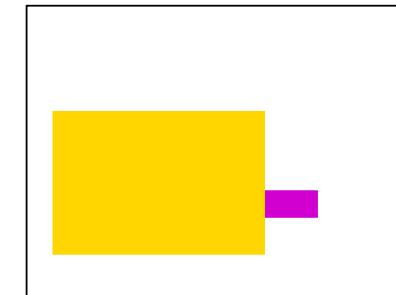
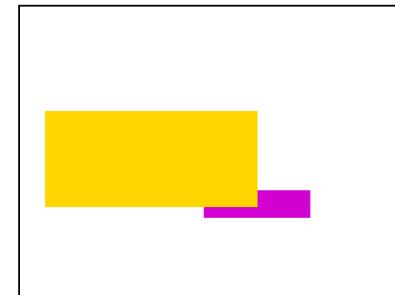
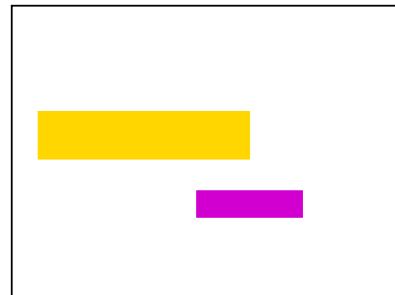
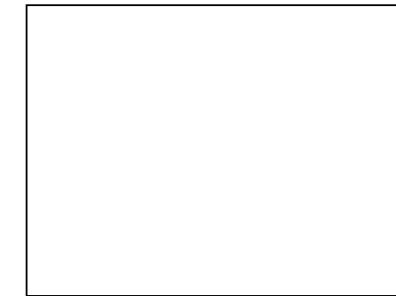
(Painter's Algorithm \Rightarrow Abschnitt 6.3.1)



$t = 0 \text{ s}$

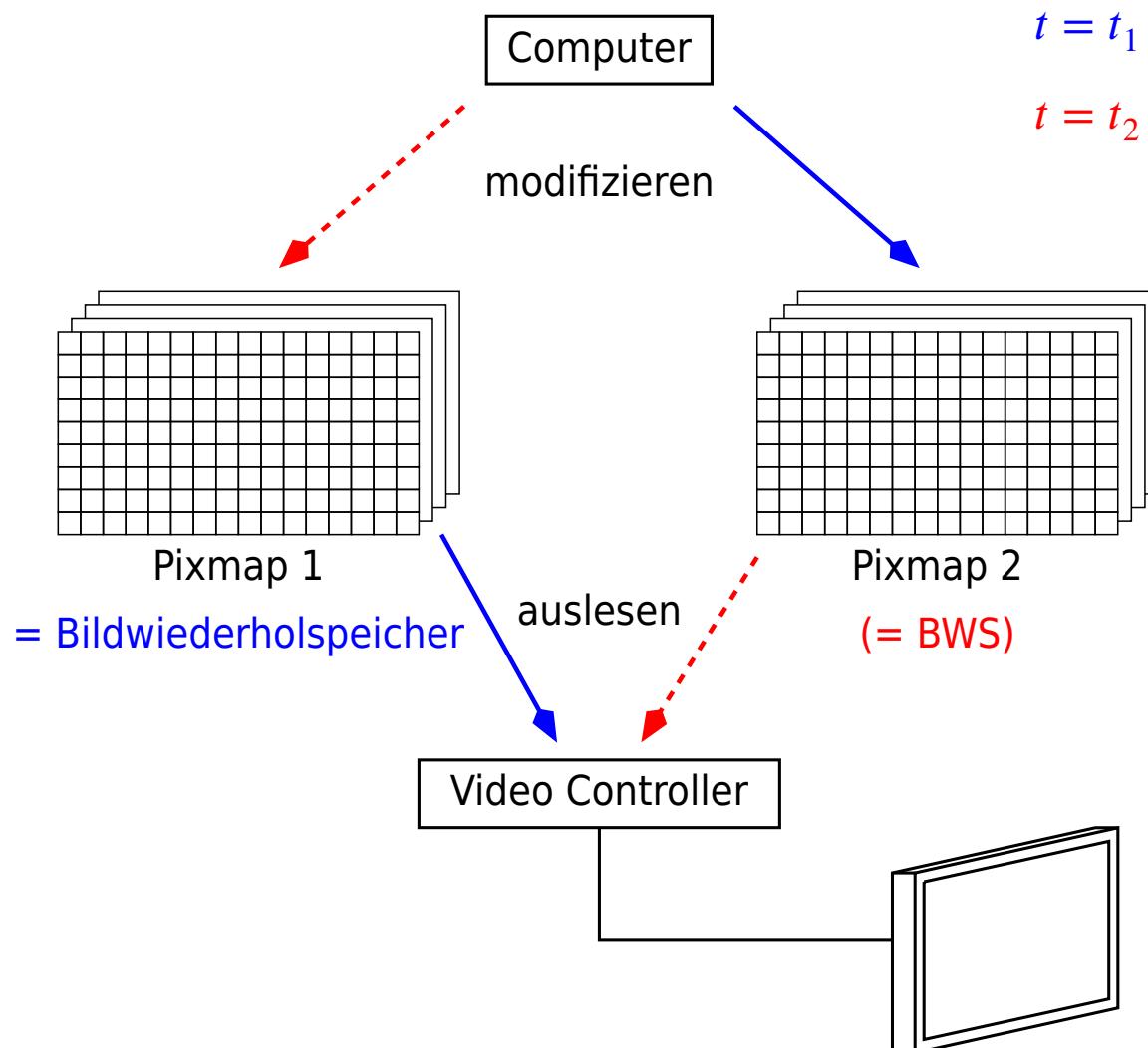


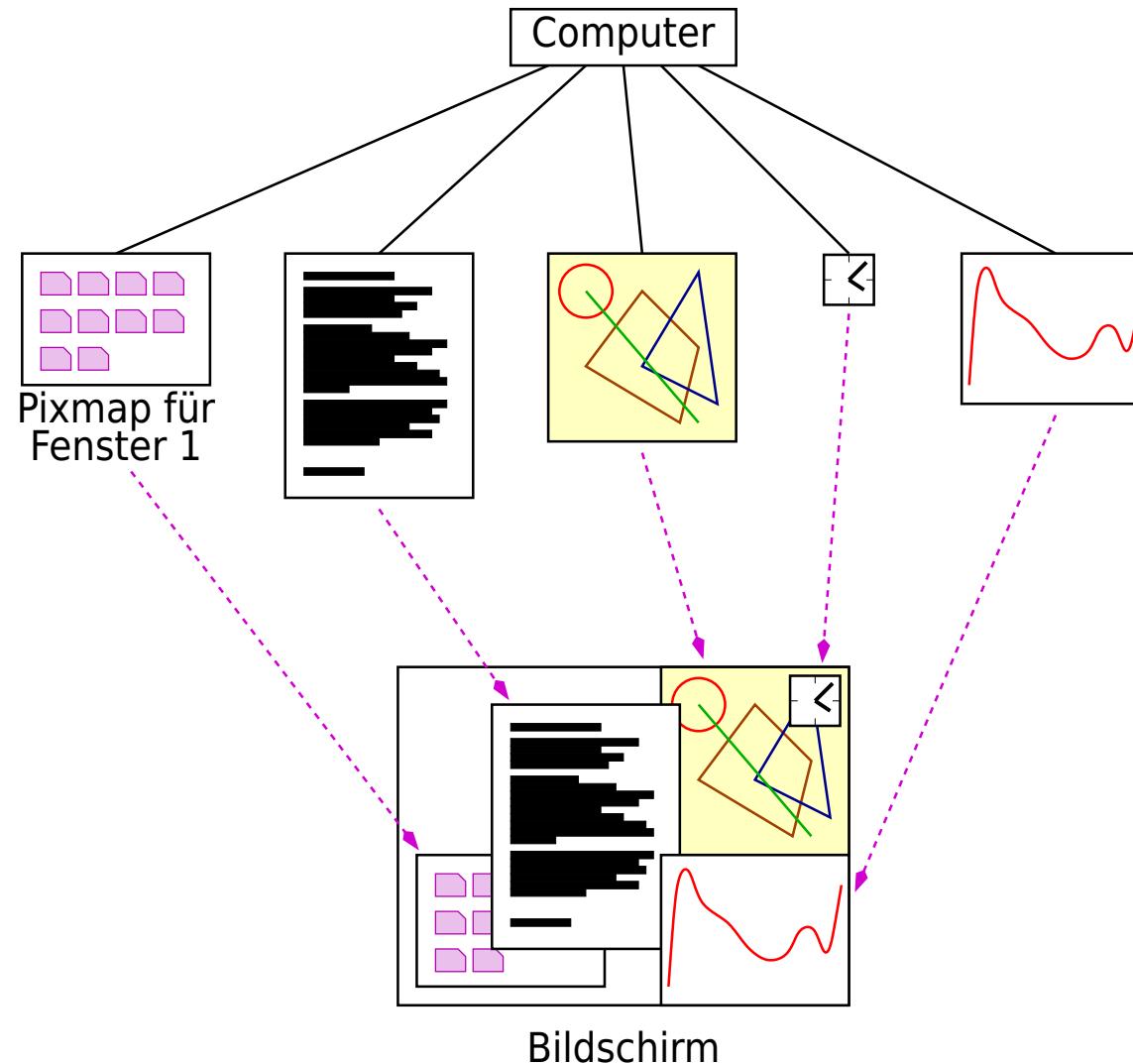
$t = \frac{1}{60} \text{ s}$



Abhilfe: Verwende zwei Pixmaps.

Es wird immer nur diePixmap modifiziert, die gerade nicht vom Video Controller ausgelesen wird.



Anwendung 2: Verwaltung einer Fensteroberfläche

Anwendung 3: Rasterung für ein anderes Gerät

(z. B. Drucker)

Bemerkung 3.2: Für eine Pixmap der Größe $n_x \times n_y$ wird ein Speicherbereich passender Größe reserviert.

- ⇒ Pixel außerhalb der Pixmap dürfen nicht modifiziert werden.
 - ⇒ Die Objekte werden an den Grenzen der Pixmap „abgeschnitten“.
 - ⇒ „Clipping“ (Kapitel 5)
-

3.1 Scan Conversion für Strecken

Annahmen:

- Die Endpunkte der Strecken haben ganzzahlige Koordinaten:

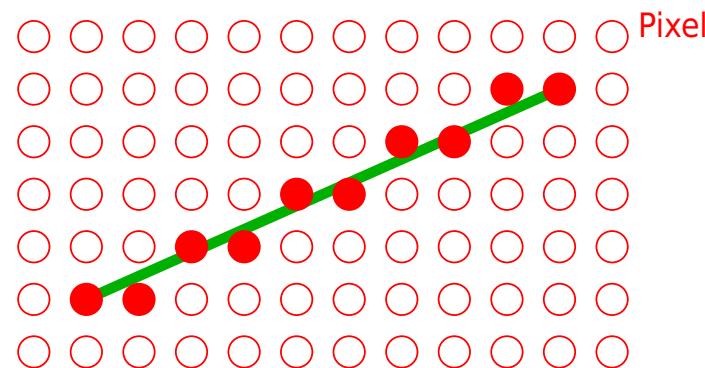
$$(x_1, y_1) \in \mathbb{Z}^2, \quad (x_2, y_2) \in \mathbb{Z}^2$$

- $x_1 < x_2$
- Für die Steigung m der Strecke gilt:

$$|m| \leq 1$$

(Sonst vertauscht man die Rollen von x und y .)

Ziel: Für jeden x -Wert soll genau ein Pixel gesetzt werden, das möglichst nahe an der Strecke liegt.

Beispiel 3.3:

3.1.1 Das naive Verfahren

Algorithmus 3.4:

Algorithmus Scan Conversion für Strecke, naiv

```
m :=  $\frac{y_2 - y_1}{x_2 - x_1}$ 
b :=  $y_1 - m \cdot x_1$ 
//  $y = mx + b$  ist die Steigungsform der Geraden durch  $(x_1, y_1)$  und  $(x_2, y_2)$ 
für  $x = x_1, x_1 + 1, \dots, x_2$ 
  y :=  $mx + b$ 
  modifiziere Pixel  $(x, \text{round}(y))$ 
```

Problem: viele Operationen mit reellen Zahlen sowie

Rundung

- ⇒ relativ langsam,
- nicht für Hardware-Realisierung geeignet

Frage: Gibt es auch einen Algorithmus ohne reelle Operationen?

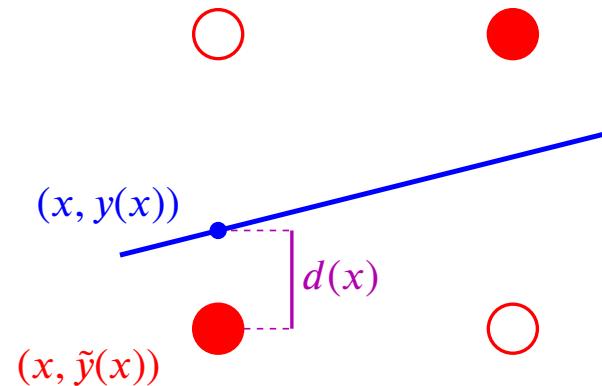
3.1.2 Inkrementell mit reeller Rechnung

Für jedes $x \in \{x_1, x_1 + 1, \dots, x_2\}$ sei:

$y(x) := mx + b$ der zu x gehörige y -Wert auf der (exakten) Geraden

$\tilde{y}(x) := \text{round}(y(x))$ der y -Wert des in Spalte x gewählten Pixels

$d(x) := y(x) - \tilde{y}(x)$ „um wie viel liegt die Gerade an der Stelle x oberhalb des Pixels?“



Idee: Berechne $y(x)$ und $\tilde{y}(x)$ nicht für jeden x -Wert gemäß der obigen Formeln, sondern bestimme, um wie viel sie sich beim Übergang von x zu $x + 1$ ändern („**Inkrement**“):

$$y(x+1) = y(x) + \Delta y(x)$$

$$\tilde{y}(x+1) = \tilde{y}(x) + \Delta \tilde{y}(x)$$

Annahme: Im Folgenden sei $m \geq 0$ (also $m \in [0; 1]$).

Es ist

$$\begin{aligned}\Delta y(x) &= y(x+1) - y(x) \\ &= m \cdot (x+1) + b - (m \cdot x + b) \\ &= m \quad (\text{unabhängig von } x)\end{aligned}$$

und

$$\begin{aligned}y(x+1) - \tilde{y}(x) &= y(x) + m - \tilde{y}(x) \\ &= d(x) + m \\ &\in \left[-\frac{1}{2}; \frac{3}{2}\right] \quad (\text{da } d(x) \in \left[-\frac{1}{2}; \frac{1}{2}\right] \text{ und } m \in [0; 1]).\end{aligned}$$

$$\Rightarrow y(x+1) \in \left[\tilde{y}(x) - \frac{1}{2}; \tilde{y}(x) + \frac{3}{2}\right]$$

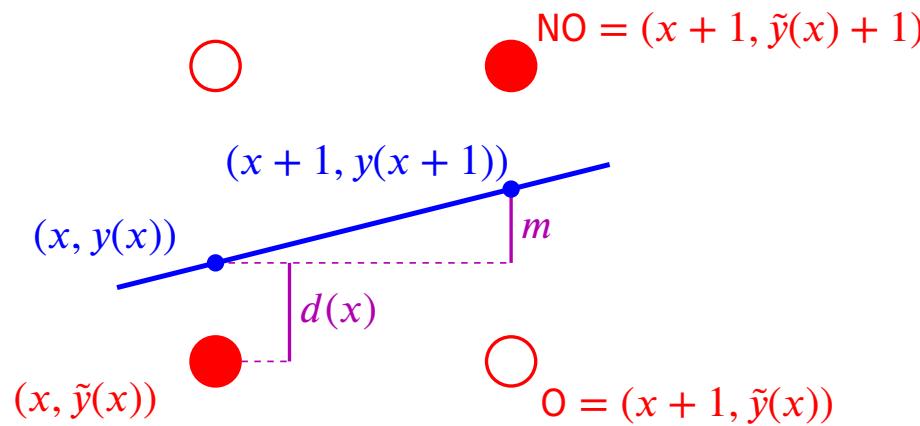
$$\Rightarrow \tilde{y}(x+1) = \text{round}(y(x+1)) \in \{\tilde{y}(x), \tilde{y}(x) + 1\}$$

\Rightarrow In Spalte $x+1$ wird eines der beiden Pixel

$$(x+1, \tilde{y}(x)) =: \mathbf{O} \quad (\text{„östlicher Nachbar“})$$

$$(x+1, \tilde{y}(x) + 1) =: \mathbf{NO} \quad (\text{„nordöstlicher Nachbar“})$$

ausgewählt, d. h. $\Delta \tilde{y}(x) \in \{0, 1\}$.



Entscheidung für „O“

\Leftrightarrow O liegt näher am Geradenpunkt $(x + 1, y(x + 1))$ als NO

$$\Leftrightarrow \underbrace{d(x) + m}_{=: \tilde{d}(x+1)} \leq \frac{1}{2}$$

Algorithmus 3.5:

Algorithmus Scan Conversion für Strecke, inkrementell, reelle Rechnung

$$m := \frac{y_2 - y_1}{x_2 - x_1}$$

$$(x, \tilde{y}(x)) := (x_1, y_1)$$

$$d(x) := 0 \quad // Startpixel liegt auf der Geraden$$

modifizierte Pixel $(x, \tilde{y}(x))$

für $x = x_1 + 1, \dots, x_2$

$$\tilde{d}(x) := d(x - 1) + m$$

wenn $\tilde{d}(x) \leq \frac{1}{2}$

$$\tilde{y}(x) := \tilde{y}(x - 1) \quad // Entscheidung für „0“$$

$$d(x) := \tilde{d}(x)$$

sonst

$$\tilde{y}(x) := \tilde{y}(x - 1) + 1 \quad // „NO“$$

$$d(x) := \tilde{d}(x) - 1$$

modifizierte Pixel $(x, \tilde{y}(x))$

Bemerkungen 3.6:

1. Im Fall $m < 0$ kann auch $\tilde{d}(x) < -\frac{1}{2}$ vorkommen. Dann ist $\Delta \tilde{y}(x) = -1$, also

$$(x + 1, \tilde{y}(x) - 1) =: \text{SO} \quad (\text{„südöstlicher Nachbar“})$$

zu wählen.

2. $y(x)$ tritt nicht mehr explizit auf.
 3. keine reelle Multiplikation und keine Rundung reell \Rightarrow ganzzahlig mehr, aber noch reelle Addition und reeller Vergleich
-

3.1.3 Inkrementell mit ganzzahliger Rechnung

Idee: Die Größen m , \tilde{d} , $\frac{1}{2}$ und d in Algorithmus 3.5 sind **rational**; Multiplikation mit dem **Hauptnenner**

$$\textcolor{orange}{H} := 2 \cdot (x_2 - x_1)$$

macht sie ganzzahlig.

Algorithmus 3.7:

Algorithmus Scan Conversion für Strecke, inkrementell, ganzzahlige Rechnung

// Erinnerung: $m \in [0; 1]$

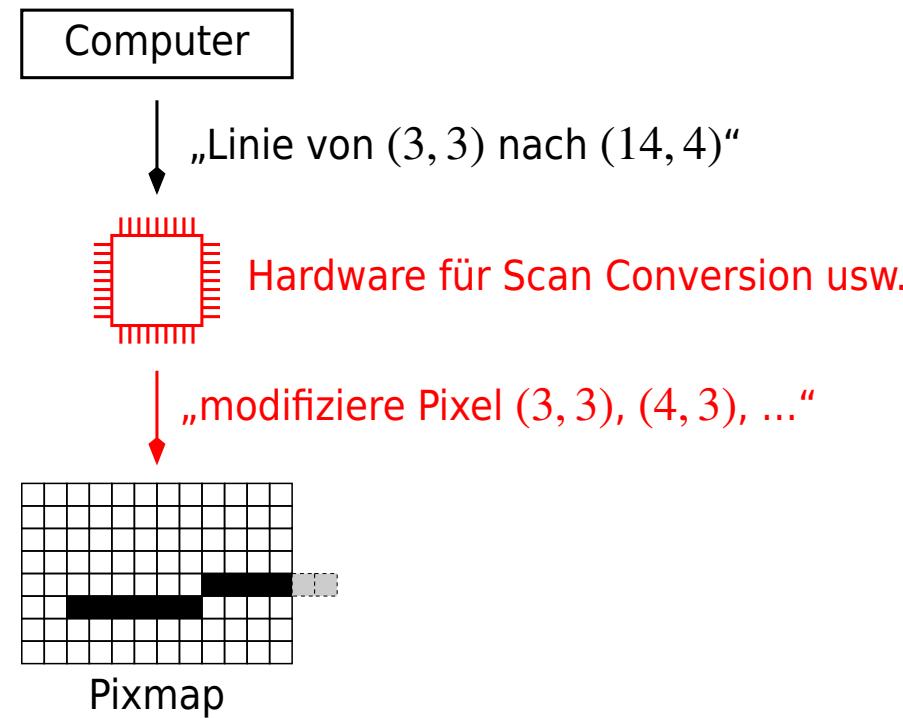
```

 $M := 2 \cdot (y_2 - y_1)$       // =  $H \cdot m$ 
 $Halb := x_2 - x_1$           // =  $H \cdot \frac{1}{2}$ 
 $Eins := 2 \cdot Halb$         // =  $H \cdot 1$ 
 $x := x_1$                   // Startpixel =  $(x_1, y_1)$ 
 $\tilde{y} := y_1$ 
 $D := 0$                     // =  $H \cdot d(x)$ 
modifizierte Pixel ( $x, \tilde{y}$ )
für  $x = x_1 + 1, \dots, x_2$ 
 $D := D + M$               //  $D$  enthält jetzt  $H \cdot \tilde{d}(x)$ 
wenn  $D > Halb$ 
     $\tilde{y} := \tilde{y} + 1$     // Entscheidung für „NO“
     $D := D - Eins$ 
modifizierte Pixel ( $x, \tilde{y}$ )

```

Bemerkung 3.8: keine reellen Operationen mehr, nur noch ganzzahlige Additionen und Vergleiche

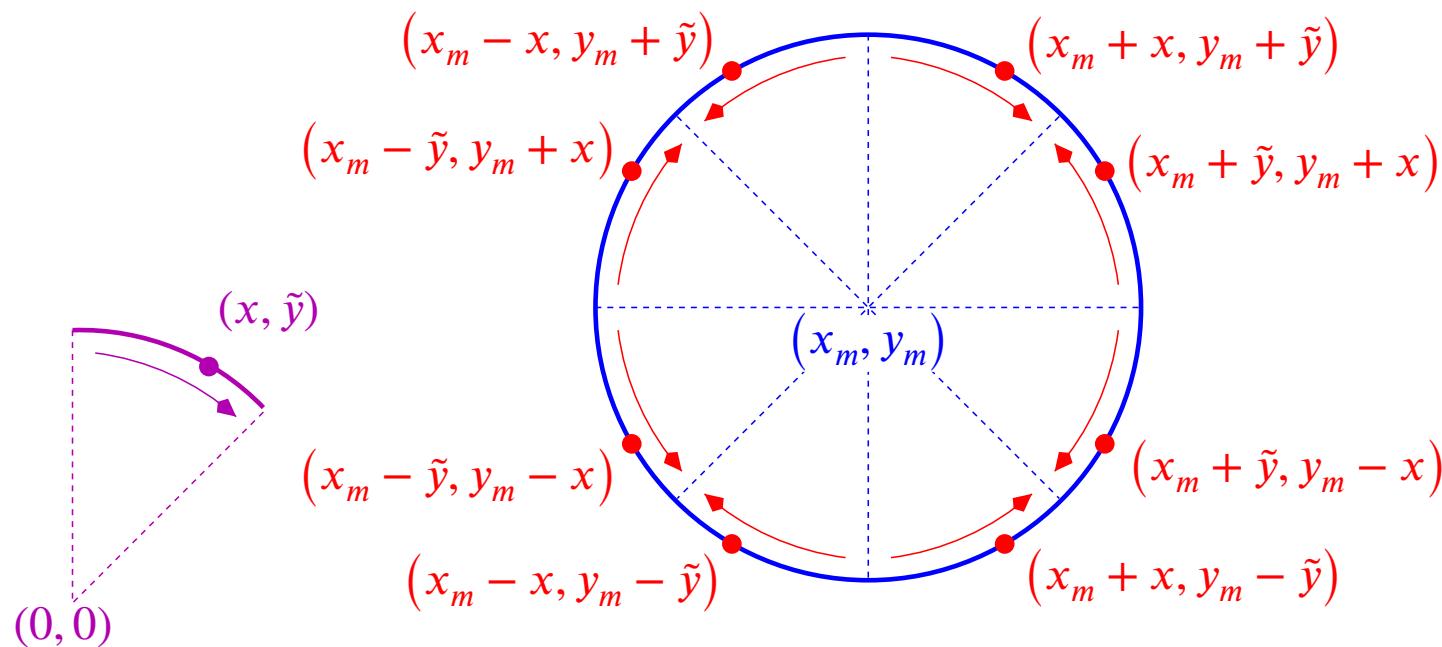
- ⇒ sehr schnell,
- gut für Hardware-Realisierung geeignet



3.2 Scan Conversion für Kreislinien

Ziel: Modifikation der Pixel in derPixmap, die dem Kreis um (x_m, y_m) mit Radius r am nächsten liegen

Bemerkung 3.9: Es genügt, die Pixel (x, \tilde{y}) im „**NNO-Achtel**“ eines Kreises mit Radius r um den **Ursprung** zu erzeugen, denn



durchläuft (x, \tilde{y}) die Pixel des NNO-Achtels, so liefern die Pixel $(x_m \pm x, y_m \pm \tilde{y})$ und $(x_m \pm \tilde{y}, y_m \pm x)$ die gesuchte Kreislinie.

Frage: Wie erzeugt man die zum NNO-Achtel gehörigen Pixel?

3.2.1 Zwei naive Verfahren

Algorithmus 3.10:

Algorithmus Scan Conversion für Kreislinie, naiv via Parametrisierung

// basiert auf der Parametrisierung $(x(t), y(t)) = (r \cdot \cos t, r \cdot \sin t)$, $t \in \left[\frac{\pi}{4}; \frac{\pi}{2}\right]$, des NNO-Achtes

wähle Δt so klein, dass keine „Lücken“ entstehen

für $t = \frac{\pi}{2}$ mit Schrittweite $-\Delta t$ bis $\frac{\pi}{4}$

$(\tilde{x}(t), \tilde{y}(t)) := (\text{round}(r \cdot \cos t), \text{round}(r \cdot \sin t))$

modifiziere die acht zu $(\tilde{x}(t), \tilde{y}(t))$ gehörigen Pixel

Bemerkung 3.11: reelle Rechnung, zwei Rundungen reell \Rightarrow ganzzahlig, **sin, cos**

\Rightarrow langsam,

nicht für Hardware-Realisierung geeignet

Algorithmus 3.12:

Algorithmus Scan Conversion für Kreislinie, naiv via Funktionsdarstellung

```
// basiert auf der Darstellung  $y = \sqrt{r^2 - x^2}$ ,  $x \in [-r; r]$ , des oberen Halbkreises
x := 0           // Startpunkt (0, r)
y := r
solange y ≥ x ist      // noch im NNO-Achtel
    modifiziere die acht zu (x, y) gehörigen Pixel
    x := x + 1
    y := round(√(r² - x²))
```

Bemerkung 3.13: reelle Rechnung, Rundung reell \Rightarrow ganzzahlig, **Wurzel**

\Rightarrow relativ langsam (aber schneller als Algorithmus 3.10),

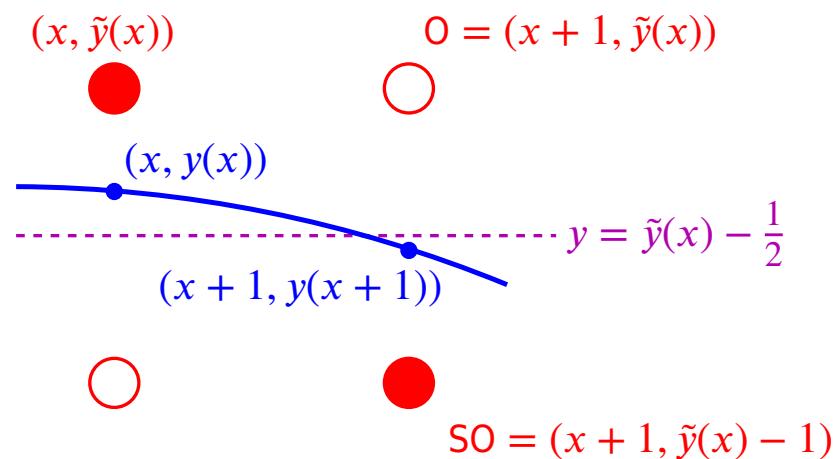
nicht für Hardware-Realisierung geeignet

Frage: Gibt es auch für Kreislinien einen inkrementellen Ansatz (wenn möglich ganzzahlig)?

3.2.2 Ein inkrementeller Ansatz

Für alle x -Werte $x = 0, 1, \dots$ sei wieder

- ($x, y(x)$) der zugehörige Punkt auf der Kreislinie und
- ($x, \tilde{y}(x)$) das entsprechende Pixel.



Ausgehend von Pixel $(x, \tilde{y}(x))$ wird von den beiden Pixeln

$$(x + 1, \tilde{y}(x)) = \mathbf{O} \quad (\text{östlicher Nachbar})$$

$$(x + 1, \tilde{y}(x) - 1) = \mathbf{SO} \quad (\text{südöstlicher Nachbar})$$

das gewählt, welches näher an dem Punkt $(x + 1, y(x + 1))$ auf der Kreislinie liegt.

$(x + 1, y(x + 1))$ liegt näher bei SO als bei O

$$\Leftrightarrow y(x + 1) < \tilde{y}(x) - \frac{1}{2}$$

$$\Leftrightarrow y^2(x + 1) < \left(\tilde{y}(x) - \frac{1}{2}\right)^2 \quad (\text{NNO-Achtel!})$$

$$\Leftrightarrow r^2 - (x + 1)^2 < \left(\tilde{y}(x) - \frac{1}{2}\right)^2$$

$$\Leftrightarrow \underbrace{r^2 - x^2 - 2x - 1}_{\text{y}^2(x)} < \tilde{y}^2(x) - \tilde{y}(x) + \frac{1}{4}$$

$$\Leftrightarrow \text{y}^2(x) - \tilde{y}^2(x) + \tilde{y}(x) - 2x - \frac{5}{4} < 0$$

Also: Setze

$$d(x) := y^2(x) - \tilde{y}^2(x) + \tilde{y}(x) - 2x - \frac{5}{4}$$

und wähle als nächstes Pixel

O, falls $d(x) \geq 0$

SO, falls $d(x) < 0$.

Frage: Kann auch d **inkrementell** berechnet werden, also $d(x + 1)$ aus $d(x)$?

Fall 1: Entscheidung für O, d. h. $\tilde{y}(x + 1) = \tilde{y}(x)$

Dann ist

$$\begin{aligned}
 d(x + 1) &= \underbrace{y^2(x + 1)}_{r^2} - \underbrace{\tilde{y}^2(x + 1)}_{(x+1)^2} + \underbrace{\tilde{y}(x + 1)}_{\tilde{y}(x)} - 2(x + 1) - \frac{5}{4} \\
 &= r^2 - (x + 1)^2 - \tilde{y}^2(x) + \tilde{y}(x) - 2(x + 1) - \frac{5}{4} \\
 &= \underbrace{r^2 - x^2}_{y^2(x)} - 2x - 1 - \tilde{y}^2(x) + \tilde{y}(x) - 2x - 2 - \frac{5}{4} \\
 &= \underbrace{y^2(x)}_{d(x)} - \tilde{y}^2(x) + \tilde{y}(x) - 2x - \frac{5}{4} - 2x - 3 \\
 &= d(x) - (2(x + 1) + 1).
 \end{aligned}$$

Fall 2: Entscheidung für SO, d. h. $\tilde{y}(x+1) = \tilde{y}(x) - 1$

Dann ist

$$\begin{aligned}
 d(x+1) &= \underbrace{y^2(x+1)}_{r^2} - \underbrace{\tilde{y}^2(x+1)}_{(x+1)^2} + \underbrace{\tilde{y}(x+1)}_{(\tilde{y}(x)-1)^2} - 2(x+1) - \frac{5}{4} \\
 &= r^2 - (x+1)^2 - (\tilde{y}(x)-1)^2 + \tilde{y}(x) - 1 - 2(x+1) - \frac{5}{4} \\
 &= \underbrace{r^2 - x^2}_{y^2(x)} - 2x - 1 - \tilde{y}^2(x) + 2\tilde{y}(x) - 1 + \tilde{y}(x) - 1 - 2x - 2 - \frac{5}{4} \\
 &= \underbrace{y^2(x) - \tilde{y}^2(x) + \tilde{y}(x) - 2x - \frac{5}{4}}_{d(x)} + 2\tilde{y}(x) - 2x - 5 \\
 &= d(x) - \left(2 \left((x+1) - \underbrace{(\tilde{y}(x)-1)}_{\tilde{y}(x+1)} \right) + 1 \right) \\
 &= d(x) - (2((x+1) - \tilde{y}(x+1)) + 1) .
 \end{aligned}$$

Algorithmus 3.14:

Algorithmus Scan Conversion für Kreislinie, inkrementell, reelle Rechnung

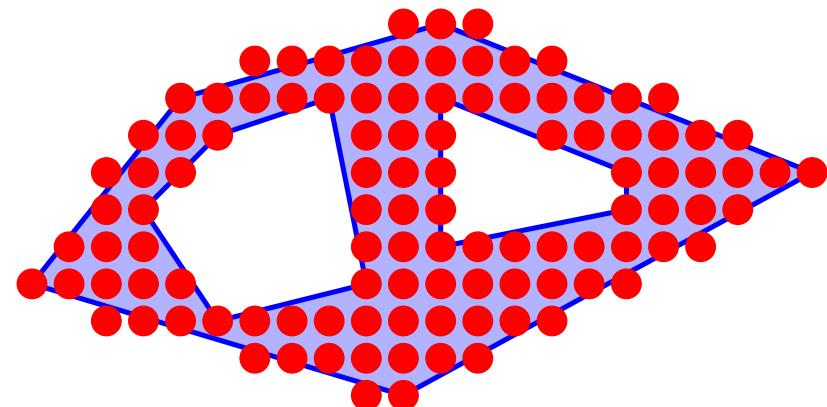
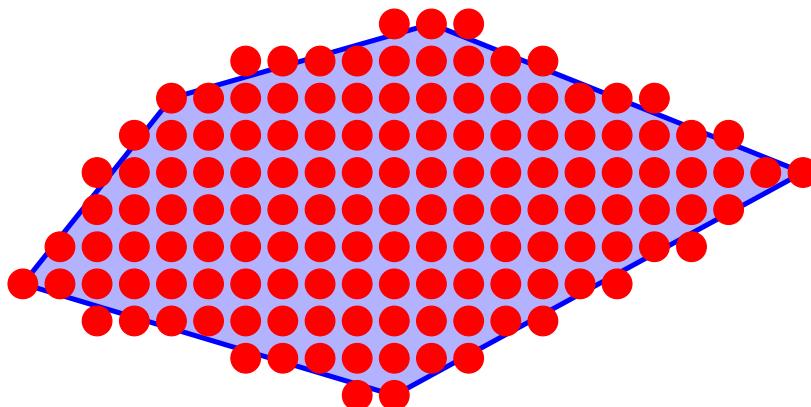
```
// basiert auf der Kreisgleichung  $y^2 = r^2 - x^2$ 
x := 0                                // Startpunkt (0, r)
y := r
d := r - 5/4                           // = d(0)
solange y ≥ x ist                      // noch im NN0-Achtel
    modifiziere die acht zu (x, y) gehörigen Pixel
    x := x + 1
    wenn d ≥ 0
        d := d - (2x + 1)                // Entscheidung für 0
    sonst
        y := y - 1                      // Entscheidung für S0
        d := d - (2(x - y) + 1)
```

Bemerkung 3.15: Algorithmus 3.14 kann (für ganzzahlige r , x_m und y_m) in einen **inkrementellen ganzzahligen** Algorithmus überführt werden.

3.3 Scan Conversion für Polygone

gegeben: ein Polygon P in der Ebene

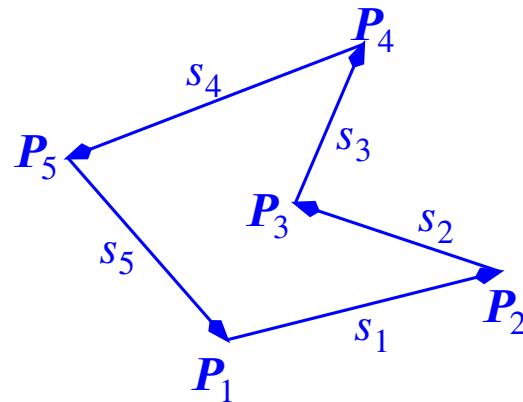
gesucht: die dem Polygon entsprechenden Pixel



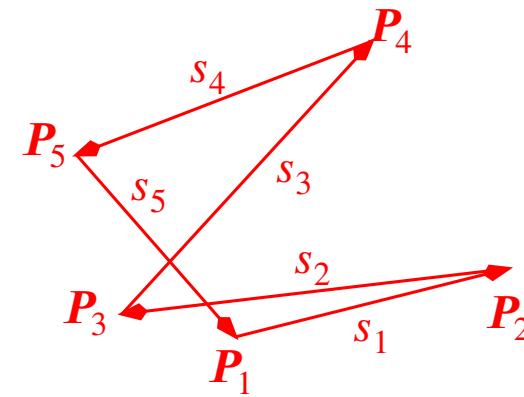
3.3.1 Polygone

Ein **geschlossener einfacher Polygonzug** $Z = (s_1, s_2, \dots, s_l)$ ist eine Folge von **Kanten** (gerichteten Strecken) $s_i = \overrightarrow{P_{i-1} P_i}$, so dass

- $P_0 = P_l$
- s_i hat mit s_{i+1} den Punkt P_i gemeinsam (sowie s_l mit s_1 den Punkt P_l);
weitere Schnittpunkte der Strecken gibt es nicht.



geschlossener einfacher
Polygonzug



kein solcher

Ein **Polygon** ist definiert durch $m \geq 1$ geschlossene einfache Polygonzüge Z_1, \dots, Z_m , die untereinander (als Punktmengen in der Ebene) disjunkt sind. Wird der Kantenzug $Z_j = (s_1^{(j)}, \dots, s_{l_j}^{(j)})$ in dieser Reihenfolge durchlaufen, so liegt das **Innere** des Polygons **links** jeder der gerichteten Strecken $s_i^{(j)}$.

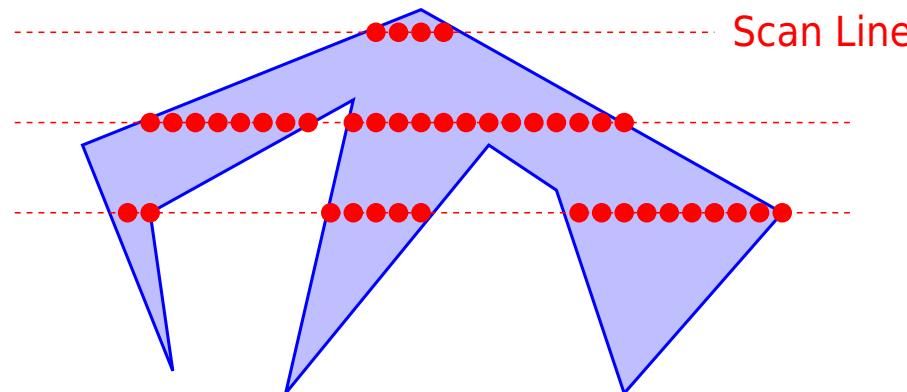
Die Punkte $P_i^{(j)}$ heißen **Ecken** des Polygons.

Ein Polygon heißt **einfach**, wenn es durch einen einzigen Polygonzug definiert wird.

Beispiel 3.16:

3.3.2 Der Grundalgorithmus

Beobachtung: Die Pixel auf jeder horizontalen (oder vertikalen) Linie (**„Scan Line“**) bilden eine oder mehrere Folgen aufeinander folgender Pixel (**„Spans“**).

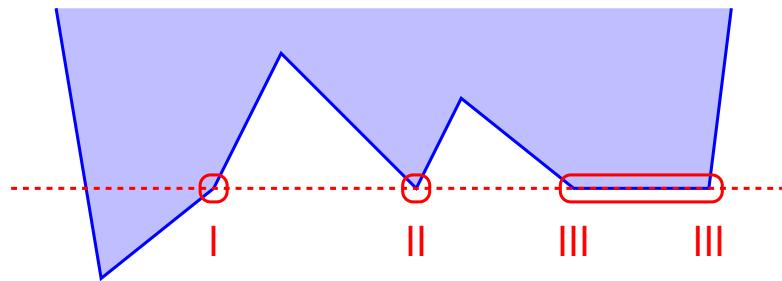


Jede solche Span wird durch zwei Schnittpunkte der Scan Line mit dem **Rand** des Polygons begrenzt.

Läuft man auf der Scan Line von links ($x = -\infty$) nach rechts ($x = +\infty$), so entspricht jeder Schnittpunkt mit einer Polygonkante einem Wechsel vom Äußeren ins Innere des Polygons und umgekehrt.

- ⇒ Ordnet man die Schnittpunkte S_j der Polygonkanten mit der Scan Line nach aufsteigenden x -Werten an, so folgt:
- Span 1 liegt zwischen S_1 und S_2 .
 - Span 2 liegt zwischen S_3 und S_4 .
 - ...

Bemerkung 3.17: Die Ecken des Polygons und seine horizontalen Kanten müssen gesondert behandelt werden:



I unterer Endpunkt der einen, oberer Endpunkt der anderen Kante

⇒ nur einmal zählen

II unterer (oder oberer) Endpunkt beider Kanten

⇒ zweimal (oder gar nicht) zählen

III gemeinsame Ecke einer horizontalen mit einer nicht horizontalen Kante

⇒ nur einmal zählen

Algorithmus 3.18:**Algorithmus Scan Conversion für Polygone, Grundalgorithmus**

// bestimme den Laufbereich für die Scan Line

bestimme unter den Ecken von P die minimale und maximale y -Koordinate y_{\min} und y_{\max}

// laufe mit der Scan Line über das Polygon

für $y = y_{\min}, \dots, y_{\max}$

für alle Seiten des Polygons

prüfe, ob die Scan Line die Seite schneidet und berechne ggf. den Schnittpunkt

sortiere die Schnittpunkte mit der Scan Line nach aufsteigenden x -Werten

entferne einige der doppelten Schnittpunkte gemäß der obigen Bemerkung

// nun seien noch S_1, \dots, S_k mit $x_1 \leq \dots \leq x_k$ übrig

für $i = 1, \dots, \frac{k}{2}$

modifiziere die Pixel der Span $(\text{round}(x_{2i-1}), y), \dots, (\text{round}(x_{2i}), y)$

Bemerkung 3.19: Für die Modifikation der Pixel einer Span gibt es i. Allg. effizientere Methoden als Algorithmus 3.7.

Aufwand: Sei

n

die Anzahl der Ecken bzw. Kanten von P und

l

$= y_{\max} - y_{\min} + 1$ die Anzahl der über das Polygon gelegten Scan Lines.

Dann beträgt der „Verwaltungsaufwand“ des Algorithmus (**ohne die eigentliche Pixelmodifikation**):

$\mathcal{O}(n)$ Operationen zur Bestimmung von y_{\min}, y_{\max}

l-mal:

n Schnitttests bzw. Schnittpunktberechnungen

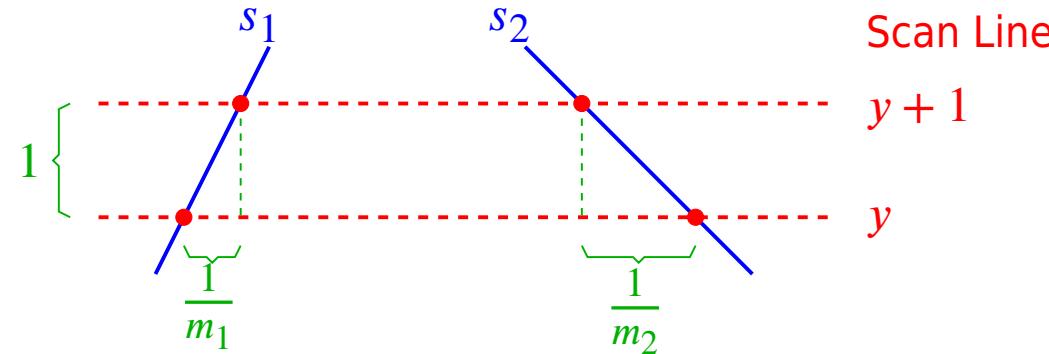
$\mathcal{O}(n \log n)$ Operationen für das Sortieren der $\mathcal{O}(n)$ Schnittpunkte

$\sum: l \cdot n$ Schnitttests/Schnittpunktberechnungen

$\mathcal{O}(l \cdot n \log n)$ sonstige Operationen

3.3.3 Ein inkrementeller Ansatz

Idee: nicht die einzelnen Pixel inkrementell berechnen, sondern die Ausdehnung der **Spans**



Beobachtung: Beim Übergang von einer Scan Line zur nächsten kann der **Schnittpunkt** mit jeder Polygonseite **inkrementell** berechnet werden: Ist

m die Steigung der Strecke und
 $(x(y), y)$ ihr Schnittpunkt mit der Scan Line y ,

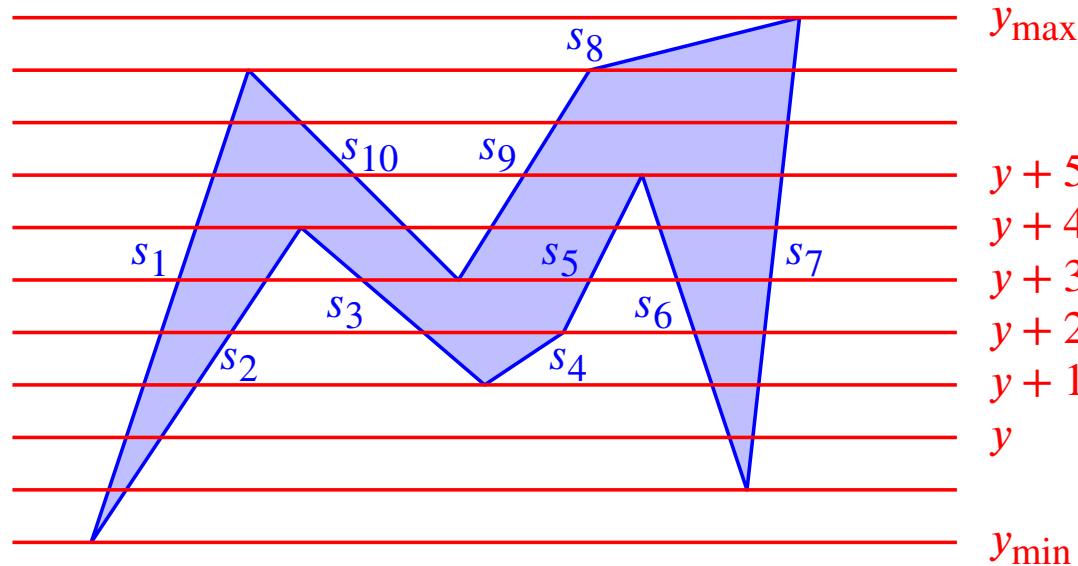
so ist

$$(x(y+1), y+1) = \left(x(y) + \frac{1}{m}, y+1 \right)$$

ihr Schnittpunkt mit der nächsten Scan Line $y+1$.

Bemerkung 3.20: Dies funktioniert natürlich nur, wenn die Strecke bereits die vorige Scan Line geschnitten hat.

Strecke s heißt **aktiv** bei Scan Line y , wenn sie die Scan Line schneidet.



aktive Strecken bei Scan Line

$y + 5:$	s_1		s_{10}	s_9		s_5	s_6	s_7	
$y + 4:$	s_1	s_2	s_3	s_{10}	s_9		s_5	s_6	s_7
$y + 3:$	s_1	s_2	s_3	s_{10}	s_9		s_5	s_6	s_7
$y + 2:$	s_1	s_2	s_3			s_4	s_5	s_6	s_7
$y + 1:$	s_1	s_2	s_3			s_4		s_6	s_7
$y:$		s_1	s_2				s_6	s_7	

Beobachtungen:

- Eine Strecke wird aktiv, sobald die Scan Line ihren **unteren** Endpunkt trifft; sie wird inaktiv, wenn die Scan Line ihren **oberen** Endpunkt passiert hat.
- **Ordnet** man die aktiven Strecken bzgl. ihres Schnittpunkts mit der Scan Line, so bleibt diese Ordnung beim Übergang zur nächsten Scan Line erhalten.

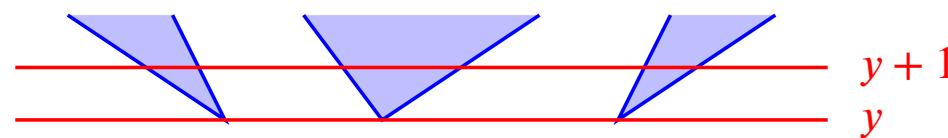
Verwaltung der aktiven Strecken:

$$Aktiv = \left(\underbrace{(i_1, x_1)}, \underbrace{(i_2, x_2)}, \dots, \underbrace{(i_k, x_k)} \right),$$

Nummer der Strecke x-Koordinate des Schnittpunkts der Scan Line mit Strecke i_1

sortiert nach aufsteigenden x-Werten

Bemerkung 3.21: Werden gleichzeitig zwei Strecken mit gleichem unterem Endpunkt aktiv, so müssen sie nach $\Delta x := \frac{1}{m}$ sortiert in $Aktiv$ eingefügt werden, da die Strecke mit kleinerem Δx auf der **nächsten** Scan Line zuerst kommt:



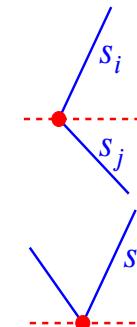
Algorithmus 3.22:**Algorithmus Scan Conversion für Polygone, inkrementell**

```
// Strecke  $s_i$  habe die Endpunkte  $(x_{i,1}, y_{i,1})$  und  $(x_{i,2}, y_{i,2})$ ;  
// dabei sei o.B.d.A.  $y_{i,1} \leq y_{i,2}$   
// _____ Vorverarbeitung _____  
für  $i = 1, \dots, n$   
    berechne  $\Delta x_i := \frac{x_{i,2} - x_{i,1}}{y_{i,2} - y_{i,1}}$  ( $= \frac{1}{m_i}$ )  
// Laufbereich der Scan Line  
bestimme unter den Ecken von  $P$  die minimale und die maximale  $y$ -Koordinate  $y_{\min}$  bzw.  $y_{\max}$   
// bestimme zu jeder Scan Line  $y$  die Mengen  
// wird_aktiv( $y$ ) bzw. wird_inaktiv( $y$ ) der dort aktiv bzw. inaktiv werdenden Strecken  
für  $y = y_{\min}, \dots, y_{\max}$   
    wird_aktiv( $y$ ) :=  $\emptyset$   
    wird_inaktiv( $y$ ) :=  $\emptyset$   
für  $i = 1, \dots, n$   
    füge  $i$  in  $wird\_aktiv(y_{i,1})$  und  $wird\_inaktiv(y_{i,2})$  ein
```

```

// _____ laufe nun mit der Scan Line über P _____
Aktiv := ∅
für y := ymin, ..., ymax
    // zuerst die aktiv gewordenen Strecken einordnen
    für alle i ∈ wird_aktiv(y)
        falls es in wird_inaktiv(y) ein Element j gibt mit xi,1 = xj,2
            ersetze (j, xj,2) in Aktiv durch (i, xi,1)
            entferne j aus wird_inaktiv(y)
        sonst
            füge (i, xi,1) bzgl. x (und ggf. Δx) sortiert in Aktiv ein
    // die Spans zwischen den aktiven Strecken zeichnen;
    // sei Aktiv = ((i1, xi1), (i2, xi2), ..., (ik, xik))
    modifiziere die Pixel der Spans (round(xi2j-1), y), ..., (round(xi2j), y), j = 1, ...,  $\frac{k}{2}$ 
    // entferne die inaktiv gewordenen Strecken
    für alle j ∈ wird_inaktiv(y)
        entferne (j, xj,2) aus Aktiv
    // berechne für die übrigen inkrementell den Schnittpunkt mit der nächsten Scan Line
    für alle (i, xi) ∈ Aktiv
        xi := xi + Δxi

```



Aufwand: Sei wieder

n die Anzahl der Ecken bzw. Kanten von P und
 $l = y_{\max} - y_{\min} + 1$ die Anzahl der Scan Lines.

$\Delta x_i, y_{\min}, y_{\max}$ berechnen:

$\mathcal{O}(n)$

wird_aktiv, wird_inaktiv belegen:

$\mathcal{O}(n + l)$

l -mal:

$\frac{k}{2}$ Spans zeichnen:

—

n -mal:

Eintrag in *wird_inaktiv* suchen, ggf. löschen: je $\mathcal{O}(n)$

Eintrag in *Aktiv* einfügen bzw. einen alten Eintrag ersetzen: je $\mathcal{O}(n)$

Eintrag aus *Aktiv* löschen: je $\mathcal{O}(n)$

$\mathcal{O}(n \cdot l)$ -mal:

x -Wert eines *Aktiv*-Eintrags ändern: je $\mathcal{O}(1)$

gesamter Verwaltungsaufwand: $\mathcal{O}(n \cdot (n + l))$

(keine Schnittpunktberechnungen mehr!)

Bemerkungen 3.23:

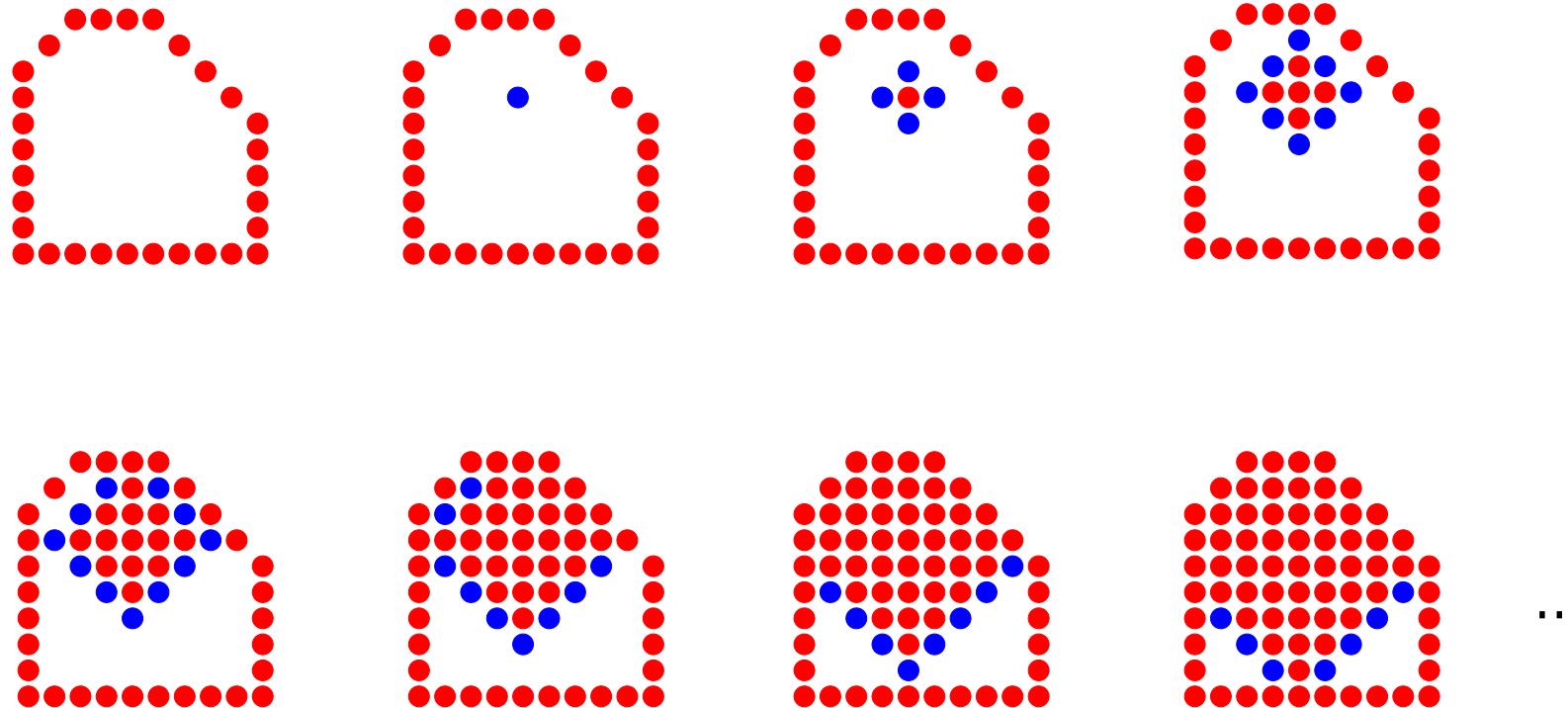
1. Für die Aufwandsanalyse wurde angenommen, dass *Aktiv*, *wird_aktiv* und *wird_inaktiv* mit Feldern implementiert sind.

Bei Verwendung geeigneter Datenstrukturen kann der Verwaltungsaufwand auf $\mathcal{O}(n \cdot (\log n + l))$ reduziert werden.

2. Im Algorithmus fehlt die Behandlung horizontaler Kanten.
 - einfachste Lösung: gar nicht berücksichtigen
 - Folge: „Obere“ horizontale Kanten werden nicht gemalt.
 3. Für **konvexe** Polygone wird der Algorithmus wesentlich einfacher.
-

3.3.4 Der Flood Fill Algorithmus

Idee: Zeichne zuerst den Rand des Objekts und fülle dann das Innere aus:



Algorithmus 3.24:**Algorithmus Flood Fill**

zeichne den Rand des Objekts

wähle ein Pixel (x, y) im Innern des Objekts

$\text{fill}(x, y)$

Algorithmus 3.25:**Algorithmus $\text{fill}(x, y)$**

// rekursive Füll-Funktion

wenn Pixel (x, y) noch nicht den gewünschten Wert hat

modifiziere Pixel (x, y)

$\text{fill}(x + 1, y)$ // Nachbarpunkte

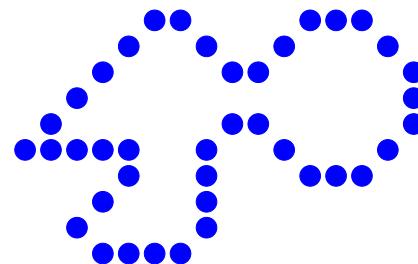
$\text{fill}(x, y + 1)$

$\text{fill}(x - 1, y)$

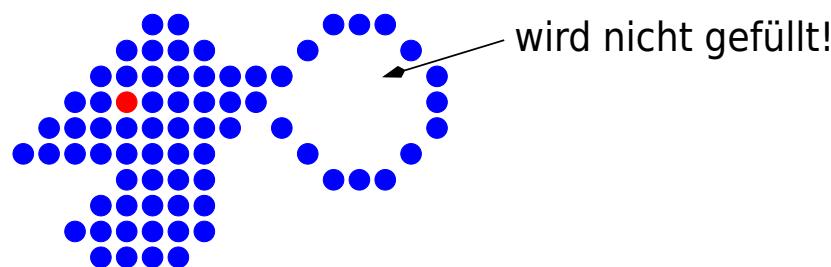
$\text{fill}(x, y - 1)$

Bemerkungen 3.26:

- ⊕ schnell
- ⊕ sehr einfach, auch für Hardware-Realisierung geeignet
- ⊕ erlaubt das Füllen nahezu beliebiger Konturen



- ⊖ Objekte der gleichen Farbe können nicht „übermalt“ werden
- ⊖ Der Rand darf keine Lücken enthalten (z. B. nicht „gestrichelt“ gezeichnet).
- ⊖ arbeitet (zumindest in der angegebenen - vereinfachten - Version) nicht immer korrekt:



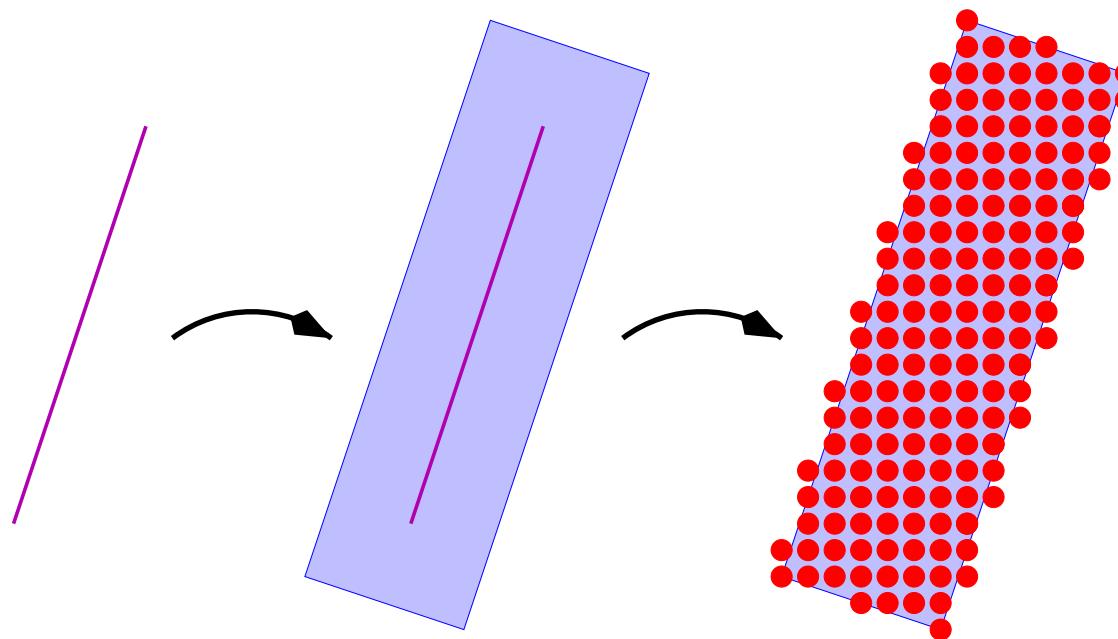
3.4 Nochmals: Strecken und Kreise

3.4.1 Linien vorgegebener „Strichbreite“

Problem: Wie zeichnet man eine Strecke/Kreislinie, die d Pixel breit ist?

Ansatz 1: Eine d Pixel breite Strecke ist ein Rechteck, also ein Polygon.

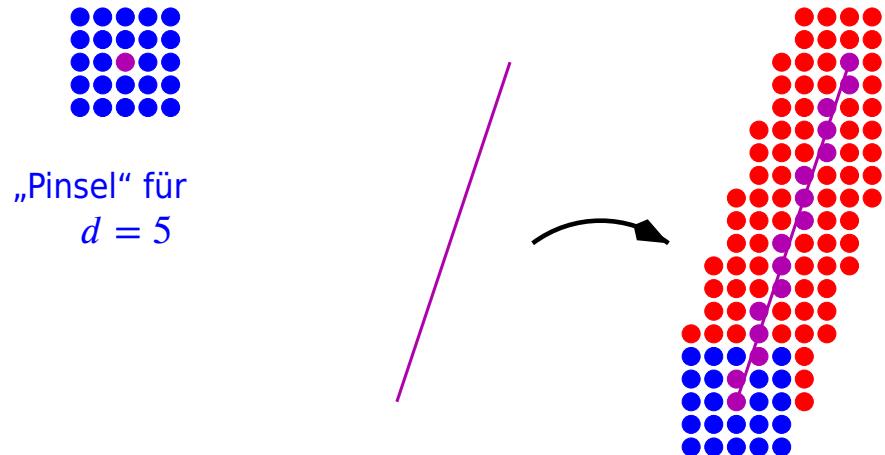
⇒ Berechne die vier Eckpunkte des Polygons und führe dann Scan Conversion für das Polygon durch.



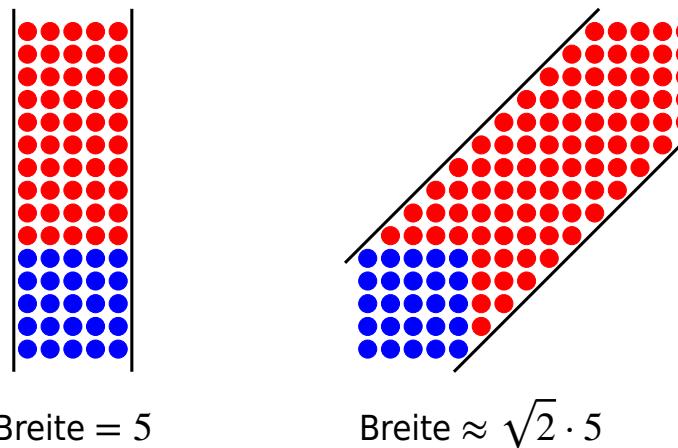
Ansatz 2: Erzeuge inkrementell die der Strecke am nächsten liegenden Pixel.

Modifiziere aber nicht nur diese, sondern einen ganzen Bereich um sie herum (**„Malen mit einem dickeren Pinsel“**).

- quadratische Pinselform:



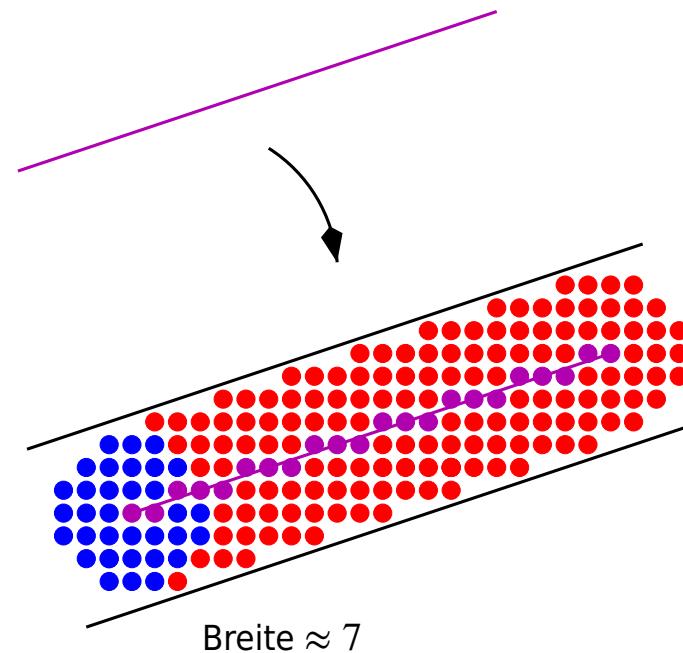
Problem: Die tatsächliche Strichbreite und die Form der Strecken-Enden hängen von der Steigung der Strecke ab:



Gegenmaßnahme: Korrektur der Pinselbreite in Abhängigkeit von der Steigung

oder

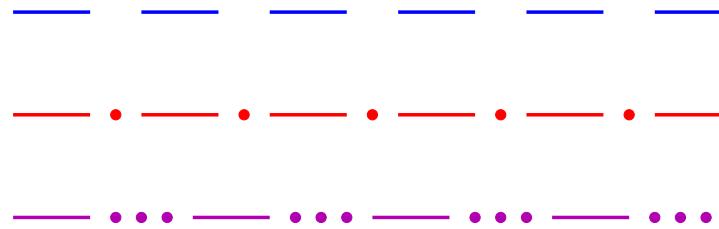
- kreisförmiger Pinsel:



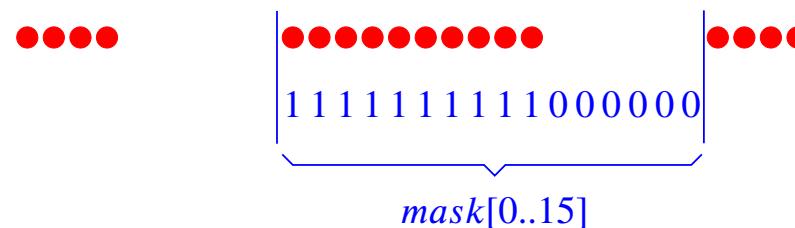
Tatsächliche Strichbreite und Form der Strecken-Enden sind (nahezu) **unabhängig von der Steigung**.

3.4.2 Unterbrochene Linien

Problem: Wie zeichnet man eine „gestrichelte“ oder „gestrichpunktete“ Linie?



Ansatz 1: Erzeuge eine **Maske** für die Periode des Musters:

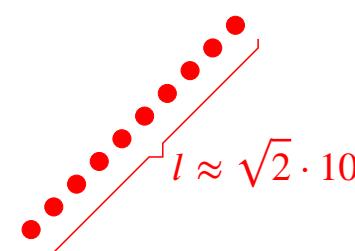
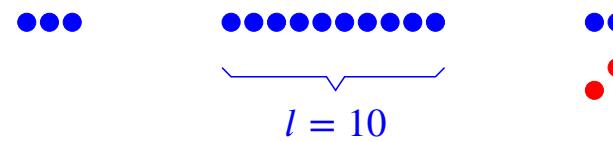


Verwende einen inkrementellen Algorithmus zur Scan Conversion, oder **zähl** die davor generierten Pixel mit (beginnend mit Nummer $i = 0$).

Modifiziere das i -te Pixel nur dann, wenn

$$mask[i \bmod \text{Maskenlänge}] = 1 .$$

Problem: Die Länge der einzelnen Striche hängt von der Steigung der Strecke ab:



Abhilfe: steigungsabhängige Länge der Maske

oder

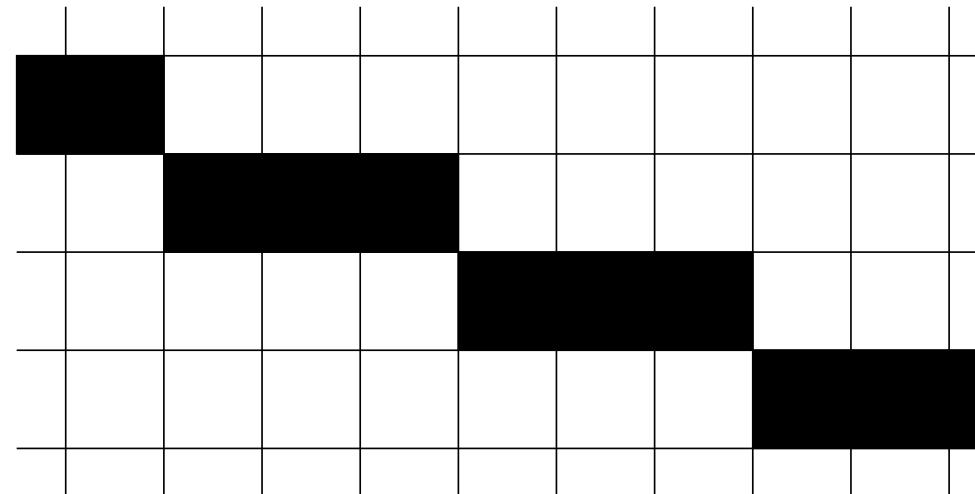
Ansatz 2: Berechne die Länge und die Lage der **einzelnen Striche** und führe für jeden einzelnen Strich die Scan Conversion durch.

Bemerkung 3.27: Beide Verfahren können auch für Strichbreite $d > 1$ verwendet werden.

3.5 Räumliche und farbliche Auflösung

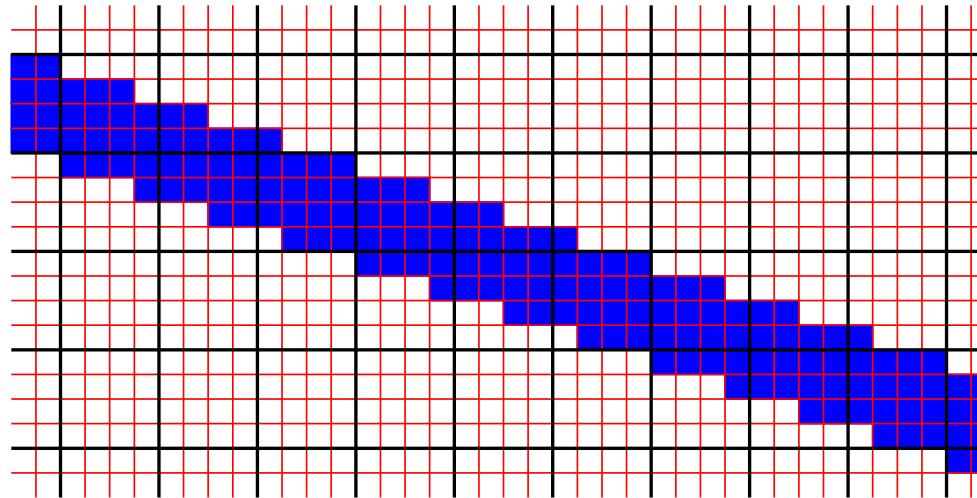
3.5.1 Räumliche Auflösung statt farblicher Auflösung

Problem: Wie reduziert man die „**Stufen**“ bei (gekrümmten oder geraden) Linien?



Idee: Die Stufen wären sicher weniger auffallend, wenn die Pixmap eine höhere räumliche Auflösung (d. h. mehr Pixel pro Flächeninhalt) hätte.

- ⇒ Unterteile jedes vorhandene Pixel in $n_x \times n_y$ „**virtuelle Pixel**“ und führe die Scan Conversion mit dieser **virtuellen Pixmap** durch.



Bestimme, welcher Anteil jedes (großen) Pixels von der Linie überdeckt wird:

	11	6	1	0	0	0	0	0
$\frac{1}{16}$.	5	10	15	11	6	1	0	0
	0	0	0	5	10	15	11	6
	0	0	0	0	0	0	5	10

Belege jedes Pixel mit einer Helligkeitsstufe, die seinem „Überdeckungsanteil“ entspricht.

Bemerkung 3.28: Dieses Verfahren gewichtet alle virtuellen Pixel innerhalb eines Pixels gleich. Meist ist eine **abstandsabhängige Gewichtung** noch besser:

$\frac{1}{16}$	1	1	1	1
	1	1	1	1
	1	1	1	1
	1	1	1	1

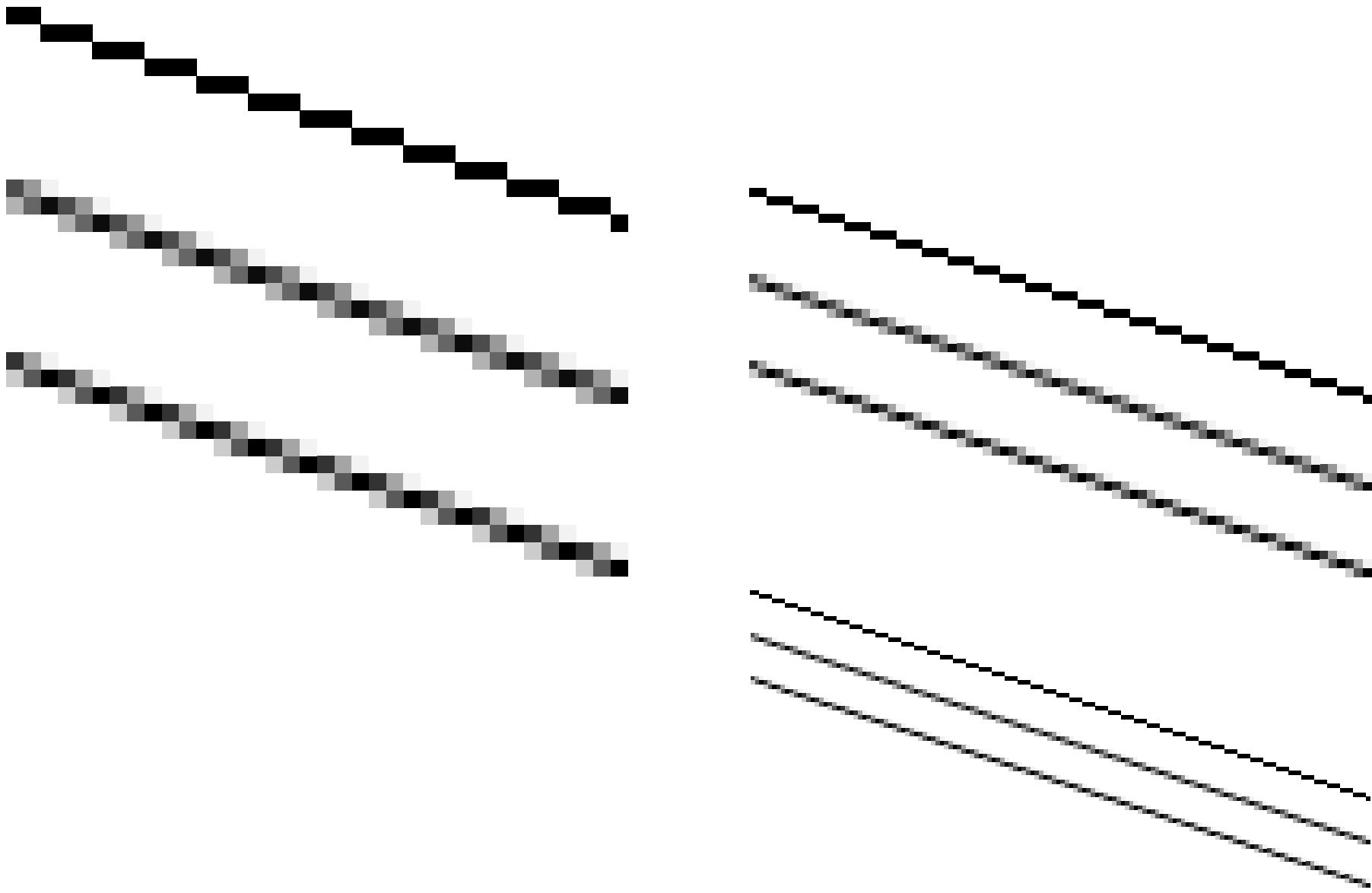
„Maske“ für konstante Gewichtung

$\frac{1}{36}$	1	2	2	1
	2	4	4	2
	2	4	4	2
	1	2	2	1

abstandsabhängige Gewichtung

	28	12	1	0	0	0	0	0
$\frac{1}{36}$	8	24	35	28	12	1	0	0
	0	0	0	8	24	35	28	12
	0	0	0	0	0	0	8	24

Manchmal werden auch überlappende Masken verwendet.



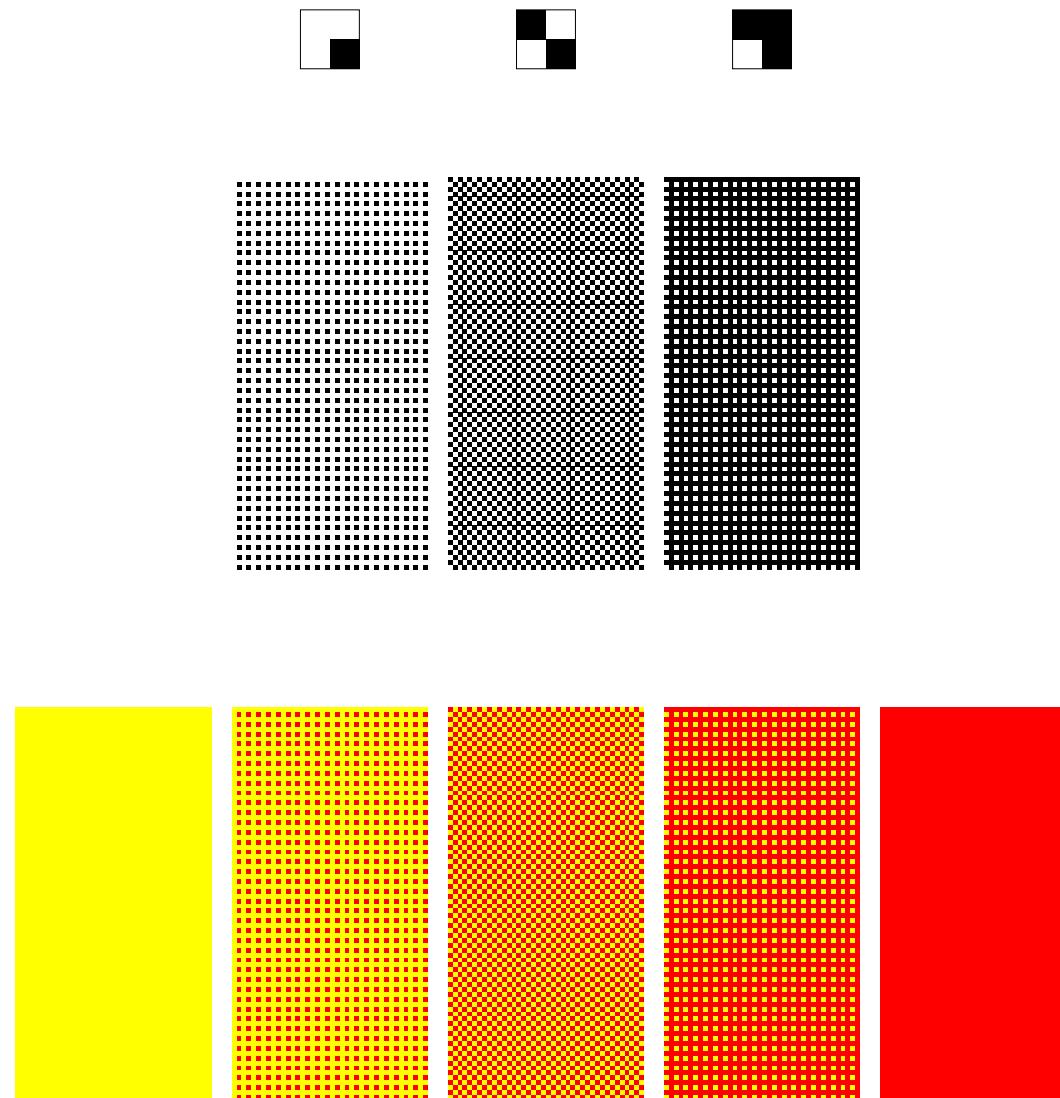
3.5.2 Farbliche Auflösung statt räumlicher Auflösung

Problem: Wie stellt man auf einem Schwarz/Weiß-Display (d. h. jedes Pixel kann nur „an“ oder „aus“ sein) **Graustufen** dar?

Wie kann man auf einem „ k -Farben-Display“ (z. B. $k = 8$ mit den Farben Rot, Grün, Blau, Gelb, Zyan, Magenta, Weiß, Schwarz) wesentlich mehr als k **Farbtöne** gleichzeitig darstellen?

Idee: Graustufen lassen sich durch „**Mischen**“ von Weiß und Schwarz, Farbtöne durch Mischen der vorhandenen „Grundfarben“ erzeugen.

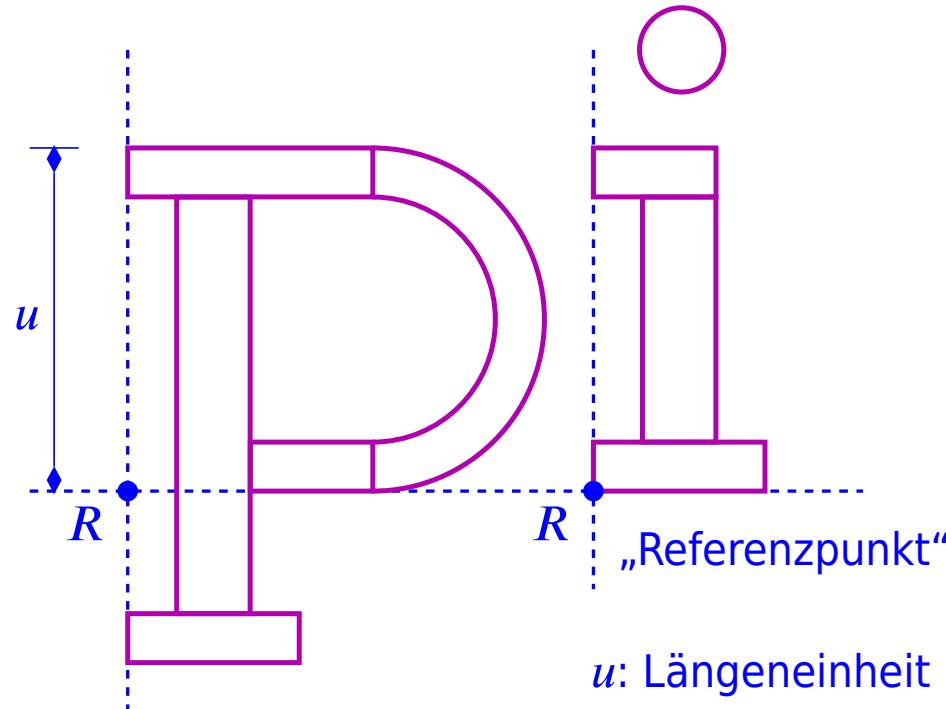
Fasse jeweils $n_x \times n_y$ Pixel zu einem **virtuellen Pixel** zusammen und belege die Pixel jedes virtuellen Pixels - dem Mischverhältnis entsprechend - mit Weiß/Schwarz bzw. mit den vorhandenen Grundfarben.



3.6 Text

Problem: Welche Pixel gehören zu einem bestimmten Zeichen eines **Zeichensatzes**?

„analytischer Ansatz“: Zerlege die Zeichen in einfachere Objekte, die gezeichnet werden können.



„p“: Linie der Breite ... von ... nach ... (relativ zu R)

Halbkreis der Breite ... mit Radius ... um ...

⋮

Übertragung des Zeichens in diePixmap durch **Scan Conversion seiner Teilobjekte**.

⊖ relativ langsam

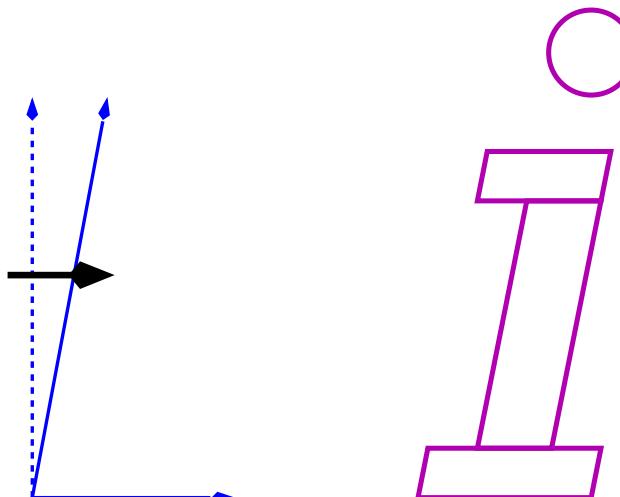
⊕ sehr flexibel bzgl. des Schriftbilds:

Zeichengröße: geeignete Wahl von u

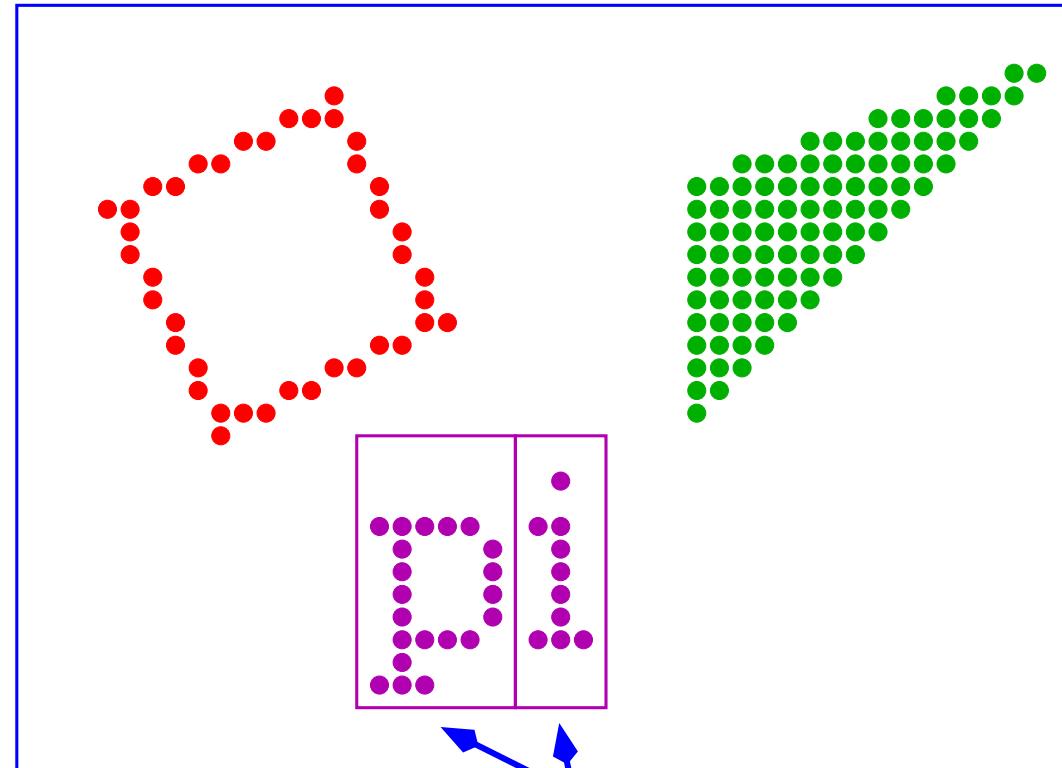
Fettdruck: Linienbreite geeignet wählen

Schrägschrift: Wende eine horizontale Scherung auf die Beschreibung des Zeichens an.

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightsquigarrow \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

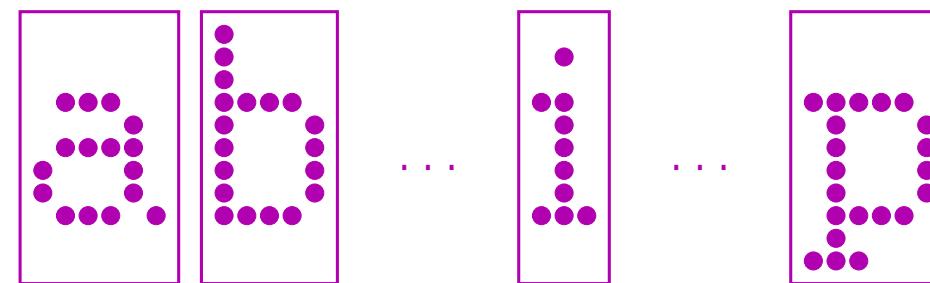


„Pre-converting“: Führe **vorab** für jedes Zeichen die Scan Conversion in eine eigenePixmap durch und **kopiere** später deren Inhalt in die „Ziel-Pixmap“.



Ziel-Pixmap

kopieren



Pixmaps für die einzelnen Zeichen

⊕ sehr schnell

⊖ benötigt viel Speicherplatz: Für jede Erscheinungsform eines Zeichens

- Zeichengröße
- normal/fett
- normal/schräg

muss eine eigene Pixmap gespeichert werden!

In der Praxis wird oft eine **gemischte Strategie** verwendet:

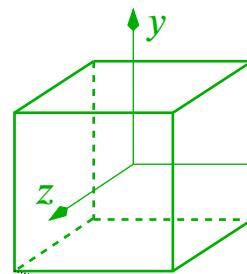
- Alle Zeichensätze liegen in einer analytischen Beschreibung vor;
- die am häufigsten verwendeten Zeichensätze werden zusätzlich pre-converted,
- seltener benutzte Zeichen(sätze) werden erst bei Bedarf in Pixel umgesetzt.

4 Koordinatentransformationen

Inhalt

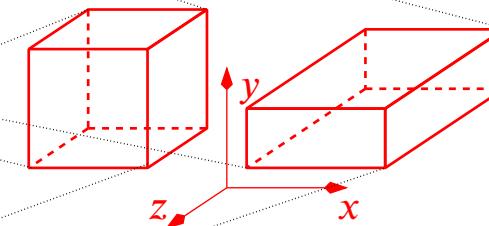
4.1 Von der Projektionsebene zu Gerätekordinaten	4-4
4.2 Projektion	4-8
4.2.1 „Standard-Parallelprojektion“	4-10
4.2.2 „Standard-Perspektivische Projektion“	4-12
4.2.3 Allgemeine Parallel- bzw. perspektivische Projektion	4-14
4.2.4 Projektive („homogene“) Koordinaten	4-21
4.3 Transformationen in der Modellierung	4-26

„Standardobjekte“ (3D, Koordinaten reell)

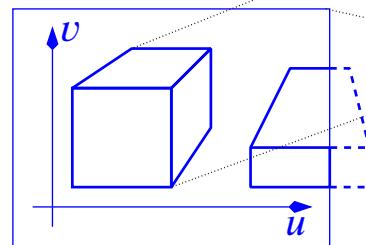


Modellierungstransformation(en)

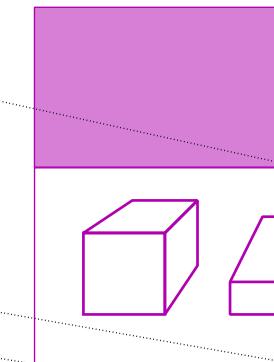
Objekte im Modell (3D, Koordinaten reell)



Projektion



Projektionsebene (2D, Koordinaten reell)



Gerätekoordinaten
(2D, evtl. ganzzahlig)

Bemerkung 4.1: **OpenGL** ist ein weit verbreiteter Grafik-Standard mit verschiedenen Möglichkeiten für

- die Beschreibung von Objekten,
- Handhabung hierarchischer Szenen,
- Festlegung von Projektionen und anderen Transformationen,
- Clipping (⇒ Kapitel 5) und Verdeckungsbehandlung (⇒ Kapitel 6),
- Färbung (⇒ Kapitel 7),
- ...

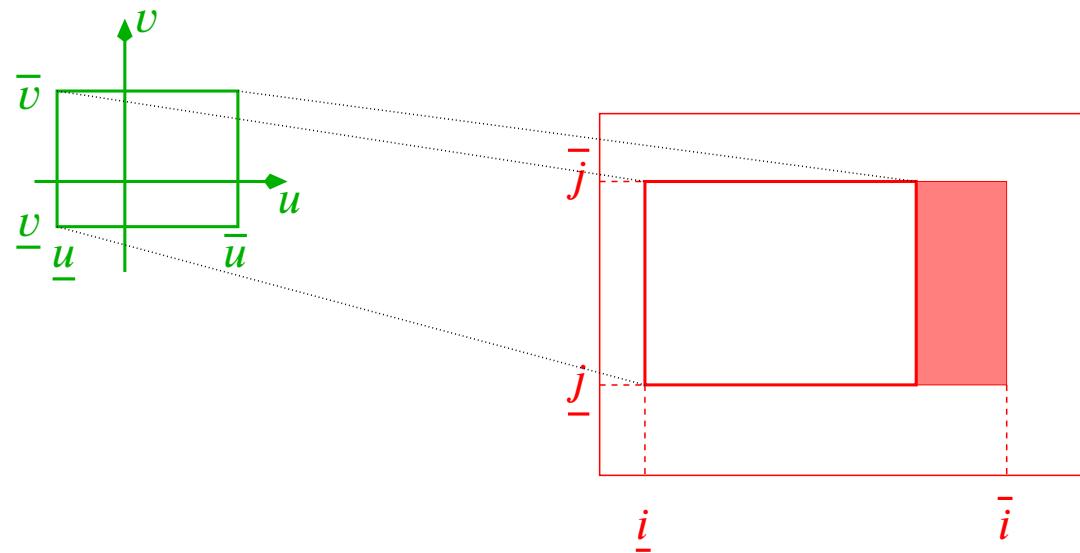
In der Vorlesung werden wir an geeigneten Stellen auf OpenGL verweisen.

Im Februar 2016 wurde Version 1.0 des Nachfolgers **Vulkan** veröffentlicht.

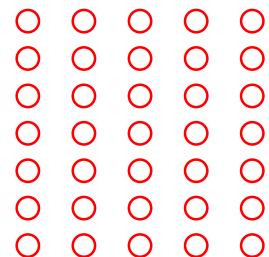


4.1 Von der Projektionsebene zu Gerätekordinaten

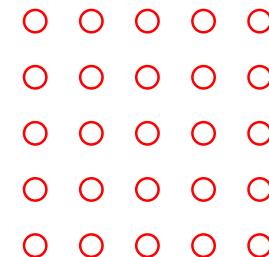
Ziel: Bilde den Bereich $[\underline{u}; \bar{u}] \times [\underline{v}; \bar{v}]$ „mit möglichst geringer Verzerrung“ in den Pixelbereich $[\underline{i}, \dots, \bar{i}] \times [\underline{j}, \dots, \bar{j}]$ ab.



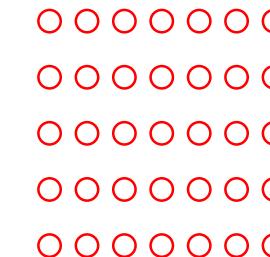
Annahme: Die Pixel liegen in i -Richtung um einen Faktor d dichter als in j -Richtung:



$$d = \frac{2}{3}$$



$$d = 1$$



$$d = \frac{3}{2}$$

⇒ u -Koordinaten müssen d -fach stärker gestreckt werden

$$1. v\text{-Bereich hat die Länge } \Delta v = \bar{v} - \underline{v}$$

$$j\text{-Bereich hat die Länge } \Delta j = \bar{j} - \underline{j}$$

$$\Rightarrow 0 \leq \text{Streckfaktor } s \leq \frac{\Delta j}{\Delta v}$$

$$2. \text{ analog: } 0 \leq d \cdot s \leq \frac{\Delta i}{\Delta u} = \frac{\bar{i} - \underline{i}}{\bar{u} - \underline{u}}$$

$$\Leftrightarrow 0 \leq s \leq \frac{\Delta i}{d \cdot \Delta u}$$

Setze also

$$s := \min \left\{ \frac{\bar{i} - \underline{i}}{d \cdot (\bar{u} - \underline{u})}, \frac{\bar{j} - \underline{j}}{\bar{v} - \underline{v}} \right\}$$

und damit

$$\begin{aligned} i &:= d \cdot s \cdot (\underline{u} - u) + \underline{i} \\ j &:= s \cdot (\underline{v} - v) + \underline{j} \end{aligned}$$

bzw.

$$\begin{aligned} \begin{pmatrix} i \\ j \end{pmatrix} &= \underbrace{\begin{pmatrix} d \cdot s & 0 \\ 0 & s \end{pmatrix} \cdot \begin{pmatrix} \underline{u} \\ \underline{v} \end{pmatrix}}_{=: M_{gp}} + \underbrace{\begin{pmatrix} \underline{i} - d \cdot s \cdot \underline{u} \\ \underline{j} - s \cdot \underline{v} \end{pmatrix}}_{=: t_{gp}} \\ &= \underbrace{M_{gp} \cdot \begin{pmatrix} \underline{u} \\ \underline{v} \end{pmatrix}}_{=: A_{gp}} + t_{gp} \quad (\text{affine Transformation}) . \end{aligned}$$

Bemerkung 4.2: Bei OpenGL wird der verfügbare Pixelbereich ganz genutzt:

$$M_{gp} = \begin{pmatrix} s_i & 0 \\ 0 & s_j \end{pmatrix}, \quad t_{gp} = \begin{pmatrix} \bar{i} - s_i \cdot \underline{u} \\ \bar{j} - s_j \cdot \underline{v} \end{pmatrix}$$

mit

$$s_i = \frac{\bar{i} - \underline{i}}{\bar{u} - \underline{u}}, \quad s_j = \frac{\bar{j} - \underline{j}}{\bar{v} - \underline{v}}$$

Für $(\bar{i} - \underline{i}) : (\bar{j} - \underline{j}) \neq d \cdot (\bar{u} - \underline{u}) : (\bar{v} - \underline{v})$ treten dabei Verzerrungen auf.

Bemerkung 4.3: Werden ganzzahlige Koordinaten benötigt, so setzt man

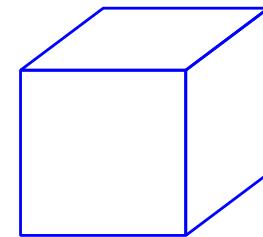
$$\begin{pmatrix} i \\ j \end{pmatrix} = \text{round}\left(A_{gp} \begin{pmatrix} u \\ v \end{pmatrix}\right).$$

4.2 Projektion

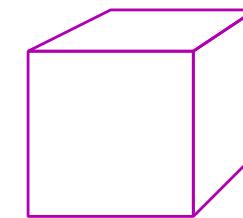
Eine Projektion ist bestimmt durch:

Projektionstyp:

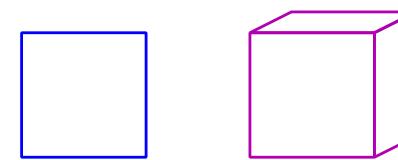
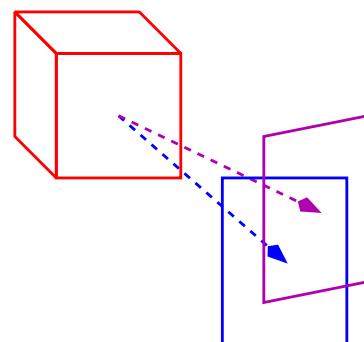
parallel



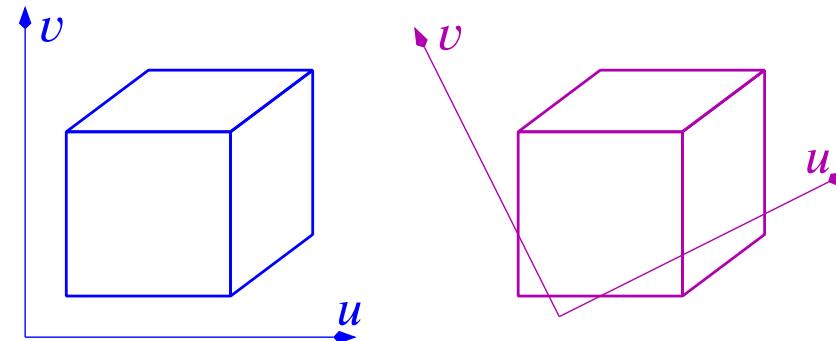
perspektivisch



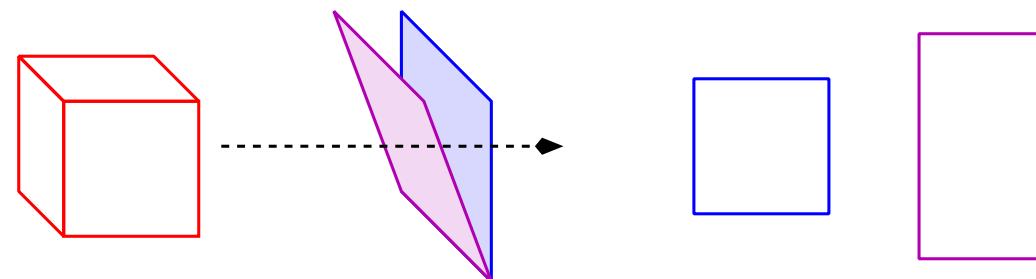
Projektionsrichtung:



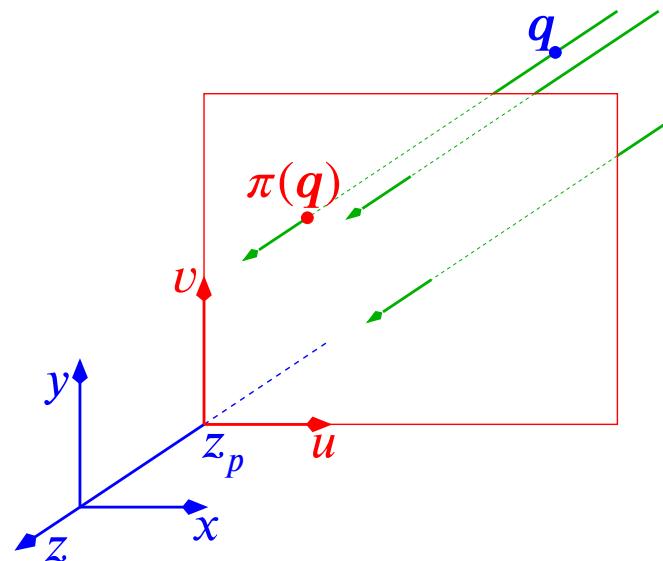
Koordinatensystem in der Projektionsebene:



(Lage der Projektionsebene zur Projektionsrichtung:)



4.2.1 „Standard-Parallelprojektion“



Projktionsebene: Ebene $z \equiv z_p$ (parallel zur xy -Ebene) im (rechtshändigen) „normalen“ dreidimensionalen Koordinatensystem

Projektionsrichtung: parallel zur z -Achse

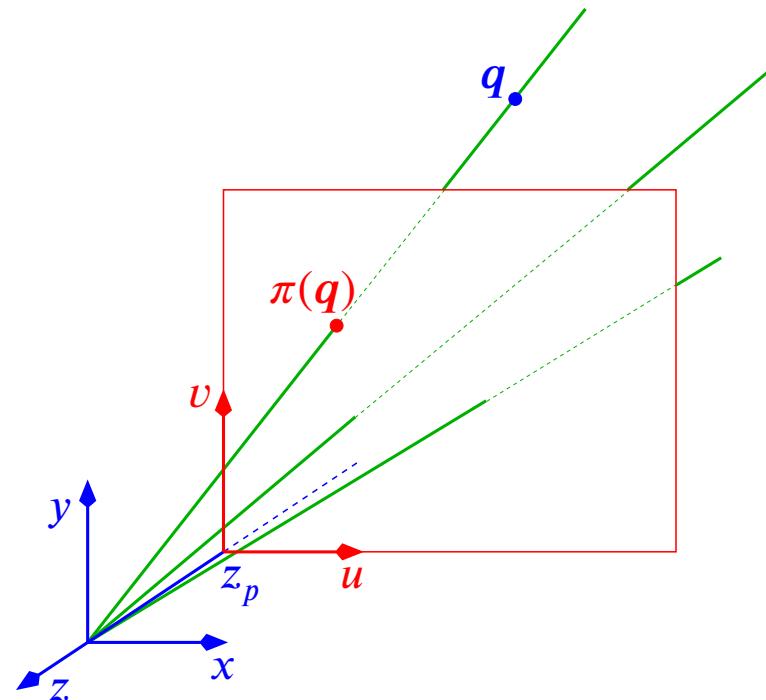
Koordinatensystem in der Projektionsebene: x, y aus dem dreidimensionalen Koordinatensystem

also:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}}_{=: M_{ps}} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\pi(\mathbf{q}) = M_{ps} \cdot \mathbf{q} =: A_{ps}(\mathbf{q}) \quad (\text{affine [sogar lineare] Transformation})$$

4.2.2 „Standard-Perspektivische Projektion“

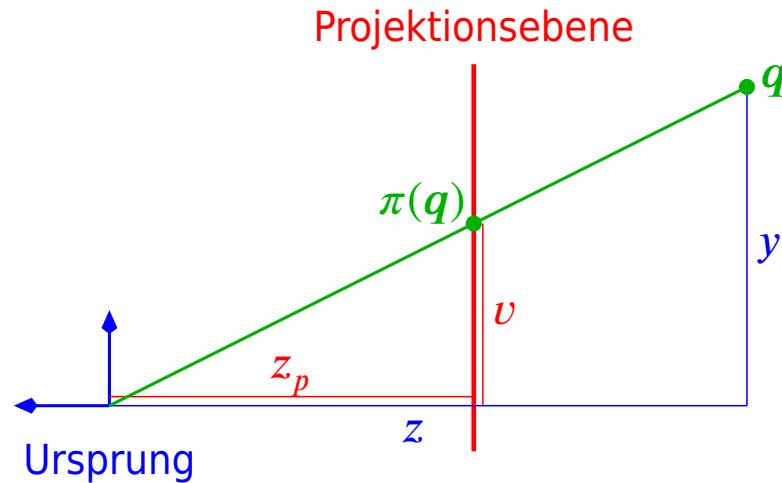


Projektionsebene: Ebene $z \equiv z_p$ im (rechtshändigen) „normalen“ dreidimensionalen Koordinatensystem

Projektionsrichtung: gegeben durch das Projektionszentrum im Ursprung des (x, y, z) -Systems

Koordinatensystem in der Projektionsebene: x, y aus dem dreidimensionalen Koordinatensystem

Wegen $\pi(\mathbf{q}) = (u, v, z_p)^T$ liefert der **Strahlensatz**



$$\frac{v}{y} = \frac{z_p}{z} = \frac{u}{x}$$

und damit

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{z_p}{z} \cdot \begin{pmatrix} x \\ y \end{pmatrix} .$$

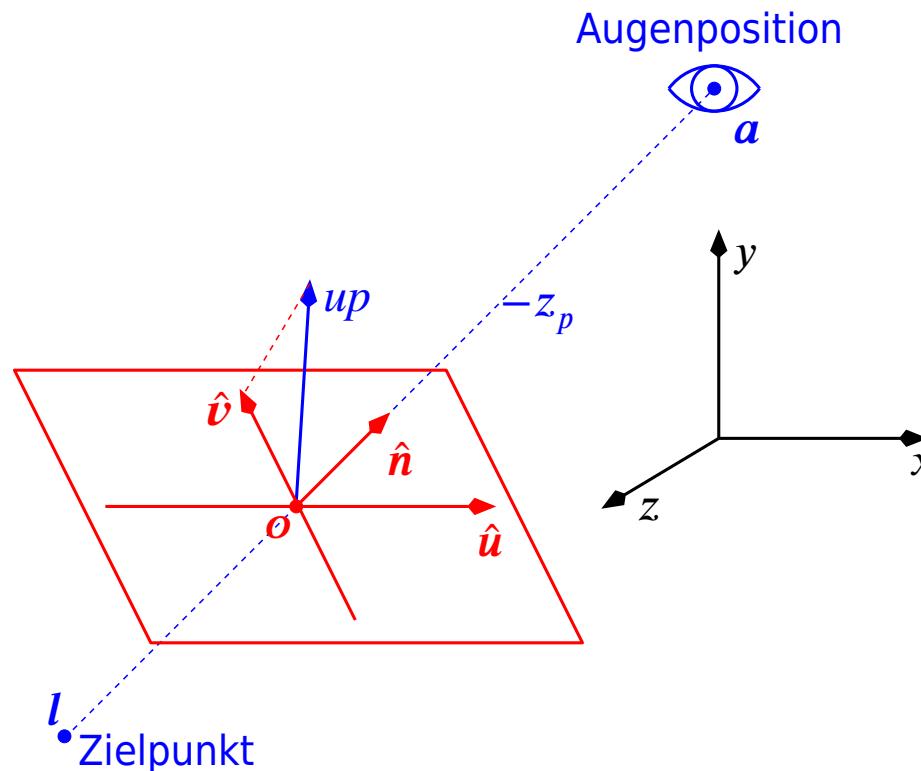
⇒ **keine affine Transformation!**

4.2.3 Allgemeine Parallel- bzw. perspektivische Projektion

Problem: Für die Standard-Projektionen müssen die Objekte so positioniert werden, dass sie bei einer bestimmten Lage der Projektionsebene ($z \equiv z_p$) sichtbar sind.

In Analogie zu einer Aufnahme mit einer Kamera sollte es möglich sein, die Objekte in einem für die Modellierung „natürlichen“ Koordinatensystem zu positionieren und die Projektionsebene geeignet zu wählen.

Dies kann etwa folgendermaßen realisiert werden (z. B. in OpenGL):



- Lege (in dem auch zur Positionierung der Objekte verwendeten (x, y, z) -Koordinatensystem)
 - eine **Augenposition** a und
 - einen **Zielpunkt** l („look at“)

sowie den Abstand $-z_p$ der Projektionsebene fest.

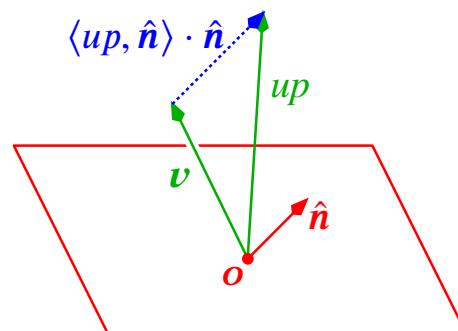
Dann befindet sich die Projektionsebene im Abstand $-z_p$ vom Augenpunkt und ist senkrecht zur Richtung $\mathbf{r} = \overrightarrow{al}$.

$\Rightarrow \hat{\mathbf{n}} := -\frac{\mathbf{r}}{\|\mathbf{r}\|_2}$ ist ein (zum Augenpunkt zeigender, normierter) Normalenvektor zur Projektionsebene.

- Der Ursprung o des (u, v) -Koordinatensystems liegt auf dem Strahl \overrightarrow{al} .

- Die positive v -Achse ergibt sich aus der Orthogonalprojektion einer (in x, y, z gegebenen) „Aufwärtsrichtung“ $up \in \mathbb{R}^3$.
(up bestimmt, ob die [in Richtung \mathbf{l} zeigende] Kamera gedreht ist [z. B. für eine „Hochformat“-Aufnahme].)

$$\begin{aligned}
 \mathbf{v} &:= \text{Orthogonalprojektion von } up \text{ auf Projektionsebene} \\
 &= up - \text{Komponente von } up \text{ in Richtung } \hat{\mathbf{n}} \\
 &= up - \langle up, \hat{\mathbf{n}} \rangle \cdot \hat{\mathbf{n}}
 \end{aligned}$$



$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|_2}$$

- Die u -Achse wird so gewählt, dass $(o; \hat{u}, \hat{v}, \hat{n})$ (normierte Vektoren) ein **rechtshändiges** 3D-Koordinatensystem bilden:

$$\begin{aligned}\hat{u} &= \hat{v} \times \hat{n} \\ &= \begin{pmatrix} \hat{v}_y \hat{n}_z - \hat{v}_z \hat{n}_y \\ \hat{v}_z \hat{n}_x - \hat{v}_x \hat{n}_z \\ \hat{v}_x \hat{n}_y - \hat{v}_y \hat{n}_x \end{pmatrix} = \det \begin{pmatrix} \hat{x} & \hat{y} & \hat{z} \\ \hat{v}_x & \hat{v}_y & \hat{v}_z \\ \hat{n}_x & \hat{n}_y & \hat{n}_z \end{pmatrix}\end{aligned}$$

$(\hat{x}, \hat{y}, \hat{z}$: Einheitsvektoren in x -, y - bzw. z -Richtung)

Bemerkung: Es gilt

$$\begin{aligned}\langle \hat{u}, \hat{n} \rangle &= \langle \hat{u}, \hat{v} \rangle = 0 \\ \|\hat{u}\|_2 &= 1 \\ \det(\hat{u} | \hat{v} | \hat{n}) &= +1 \quad (\text{rechtshändig})\end{aligned}$$

Für

- Parallelprojektion entlang der Richtung $\pm r$ bzw.
- perspektivische Projektion mit Zentrum a

liegt damit bezüglich (u, v, n) -Koordinaten „fast die Standardsituation“ vor. (Der Augenpunkt liegt nicht im Ursprung, sondern an Position $(0, 0, -z_p)$.)

Die Projektion eines Punktes $\mathbf{q} = (x, y, z)^T$ (in „**Weltkoordinaten**“) erfolgt also mit der Standard-Projektion, **nachdem** die folgenden Schritte durchgeführt wurden:

1. Verschiebung in den Ursprung des $(\mathbf{o}; \hat{\mathbf{u}}, \hat{\mathbf{v}}, \hat{\mathbf{n}})$ -Systems:

$$\underbrace{\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix}}_{\tilde{\mathbf{q}}} = \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{\mathbf{q}} - \underbrace{\begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix}}_{\mathbf{o}}$$

mit $\mathbf{o} = \mathbf{a} + z_p \cdot \hat{\mathbf{n}}$

2. alte Koordinatenrichtungen, ausgedrückt in den neuen:

$$u = \text{Länge der Komponente } \tilde{\mathbf{q}} \text{ in Richtung } \hat{\mathbf{u}} = \langle \hat{\mathbf{u}}, \tilde{\mathbf{q}} \rangle = \hat{\mathbf{u}}^T \cdot \tilde{\mathbf{q}}$$

analog:

$$v = \hat{\mathbf{v}}^T \cdot \tilde{\mathbf{q}}, \quad n = \hat{\mathbf{n}}^T \cdot \tilde{\mathbf{q}},$$

also

$$\begin{pmatrix} u \\ v \\ n \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{u}}^T \\ \hat{\mathbf{v}}^T \\ \hat{\mathbf{n}}^T \end{pmatrix} \cdot \tilde{\mathbf{q}} = (\hat{\mathbf{u}} | \hat{\mathbf{v}} | \hat{\mathbf{n}})^T \cdot \tilde{\mathbf{q}}$$

Bemerkung: Die Matrix

$$M_k := (\hat{u} | \hat{v} | \hat{n})$$

drückt „neue“ Koordinaten in „alten“ aus, vermittelt also den Koordinatenwechsel

$$(u, v, n) \rightsquigarrow (\tilde{x}, \tilde{y}, \tilde{z}) .$$

Daher wird der Koordinatenwechsel

$$(\tilde{x}, \tilde{y}, \tilde{z}) \rightsquigarrow (u, v, n)$$

durch die Matrix

$$M_k^{-1} = M_k^T$$

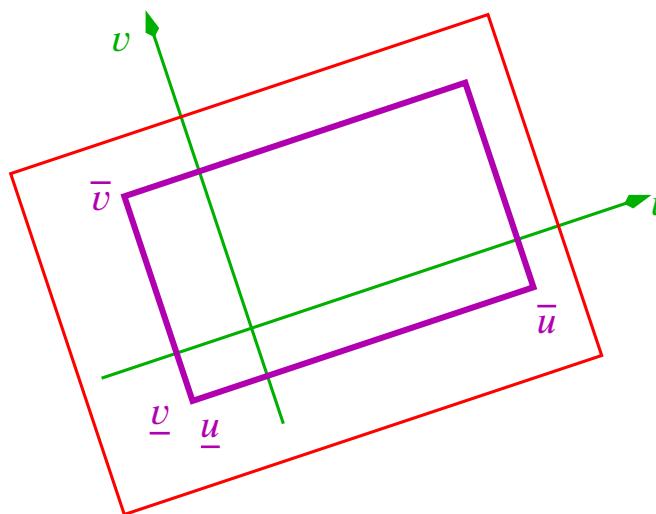
ausgedrückt.

3. Verschiebung des Augenpunktes in den Ursprung:

$$\underbrace{\begin{pmatrix} u' \\ v' \\ n' \end{pmatrix}}_{\mathbf{q}'} = \underbrace{\begin{pmatrix} u \\ v \\ n \end{pmatrix}}_{} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ z_p \end{pmatrix}}_{\mathbf{d}}$$

Insgesamt gilt dann:

$$\begin{aligned}
 q' &= M_k^T \cdot (q - o) + d \\
 &= M_k^T \cdot q + (d - M_k^T \cdot o) \\
 &=: M_{sw} \cdot q + t_{sw} \quad (\text{affine Abbildung})
 \end{aligned}$$



Der später im Bild **sichtbare Bereich** wird festgelegt durch ein **Projektionsfenster** $[\underline{u}, \bar{u}] \times [\underline{v}, \bar{v}]$, das den Ursprung o des Koordinatensystems nicht enthalten muss.

4.2.4 Projektive („homogene“) Koordinaten

Motivation

- Die Schachtelung affiner Abbildungen

$$\mathbf{q} \mapsto M_{gp} \cdot (M_{ps} \cdot (M_{sw} \cdot \mathbf{q} + t_{sw})) + t_{gp}$$

benötigt viele Operationen. Ohne die Translationen könnten die Matrizen zu einer einzigen Matrix

$$M := M_{gp} \cdot M_{ps} \cdot M_{sw}$$

zusammengefasst werden, und die Transformationen

$$\mathbf{q} \mapsto M \cdot \mathbf{q}$$

wären wesentlich effizienter.

Kann eine Translation bei geeigneter Koordinatenwahl als lineare Transformation (Matrix) geschrieben werden?

- Perspektivische Projektion führt nicht einmal auf eine affine Transformation.

Kann sie bei geeigneter Koordinatenwahl „affin (oder sogar linear) gemacht“ werden?

***n*-dimensionaler projektiver Raum:**

$$\mathbb{P}_n := \text{Menge aller eindimensionalen Unterräume des } \mathbb{R}^{n+1}$$

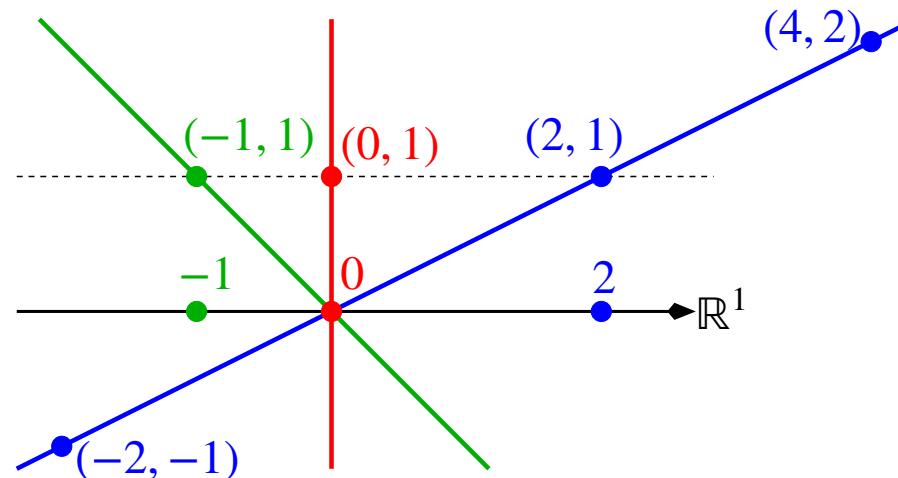
Elemente des \mathbb{P}_n : Ergänze die n -dimensionalen Koordinaten durch eine $(n + 1)$ -te mit Wert 1:

$$(x_1, \dots, x_n) \rightsquigarrow (x_1, \dots, x_n, 1)$$

Identifiziere im \mathbb{R}^{n+1} alle Vielfachen ($\neq 0$) eines Vektors:

$$(x_1, \dots, x_n, x_{n+1}) \equiv (\tau x_1, \dots, \tau x_n, \tau x_{n+1}) \quad \text{für } \tau \neq 0$$

Beispiel:



- Für $x_{n+1} \neq 0$ entspricht $(x_1, \dots, x_n, x_{n+1})$ dem (eindeutigen) Punkt $\left(\frac{x_1}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}}\right)$ im \mathbb{R}^n ,
- Für $x_{n+1} = 0$ entspricht $(x_1, \dots, x_n, 0)$ **keinem** Punkt im \mathbb{R}^n („unendlich ferner Punkt“).

Wie übertragen sich n -dimensionale Transformationen auf projektive Koordinaten?**lineare Transformationen:**

$$\mathbf{x} \mapsto \mathbf{y} := M \cdot \mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^n$$

affine Abbildungen:

$$\mathbf{x} \mapsto \mathbf{y} := M \cdot \mathbf{x} + t, \quad \mathbf{x} \in \mathbb{R}^n$$

Standard-Parallelprojektion:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Standard-Perspektivische Projektion:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{z_p}{z} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Bemerkungen 4.4:

1. In OpenGL können die 4×4 -„view matrix“ (im Wesentlichen M'_{sw}) und die Projektion ($\triangleq M'_{ps}$) auf verschiedene Arten aufgebaut werden:

- automatisch, z. B. durch Angabe von a , l , up bzw. $-z_p$, \underline{u} , \bar{u} , \underline{v} und \bar{v} ,
- durch Zusammensetzen einfacherer Transformationsmatrizen oder
- „von Hand“ durch Angabe der Matrixeinträge

Die beiden letzten Möglichkeiten erlauben auch nicht-orthogonale („oblique“) Parallelprojektionen.

2. In der „modelview matrix“ wird die „view matrix“ mit den Modellierungstransformationen (Abschnitt 4.3) zusammengefasst.
3. Die „viewport transformation“ vermittelt den Übergang von der Projektionsebene zu Gerätekordinaten.
4. Auch die n -Koordinate (Information über räumliche Tiefe) wird durch die eigentliche Projektion geführt (4×4 -Matrix statt 3×4).

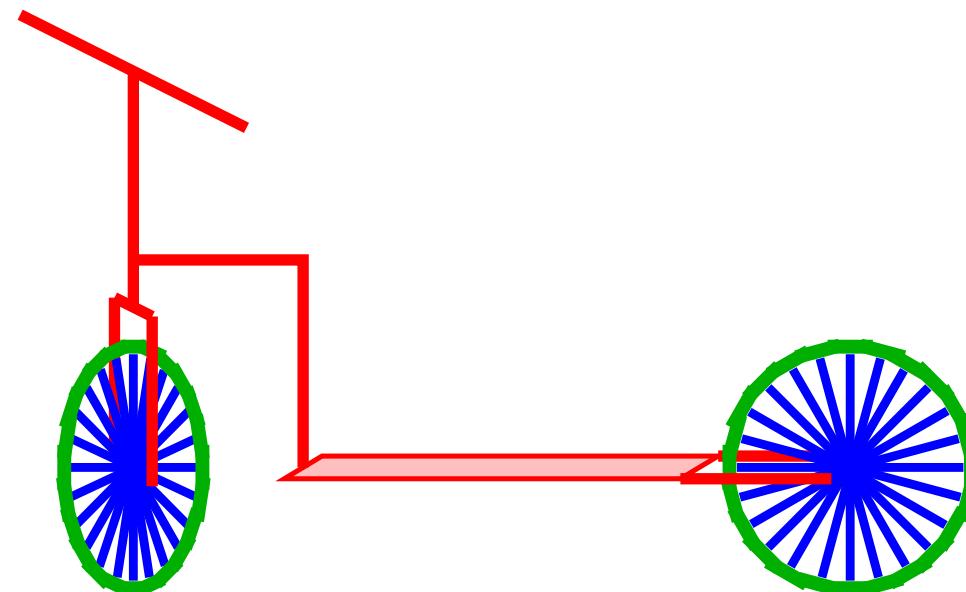
(mehr dazu in Kapitel 5 und 6)

4.3 Transformationen in der Modellierung

hierarchische Modellierung eines Tretrollers (stark vereinfacht):

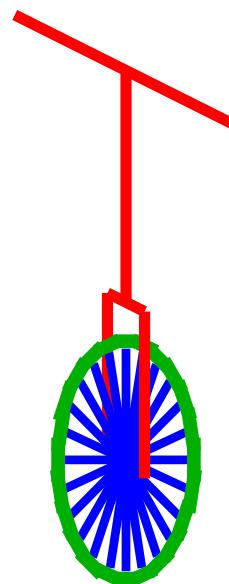
Der **Tretroller** besteht aus

- einer (um eine vertikale Achse drehbaren) „Lenkergruppe“,
- fünf starren (mit Quadern modellierten) Bauteilen und
- dem (um eine horizontale Achse drehbaren) Hinterrad.

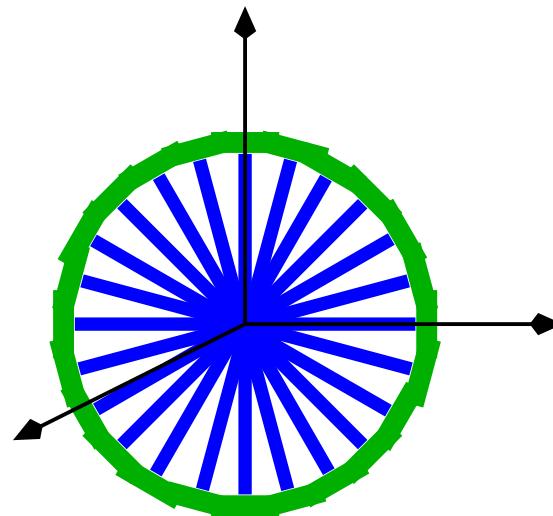


Die **Lenkergruppe** besteht aus

- dem (um eine horizontale Achse drehbaren) Rad und
- fünf starren (mit Quadern modellierten) Bauteilen für Gabel, Lenkerstange, usw.



Das **Rad** besteht aus einer vorgegebenen Anzahl von (jeweils um einen geeigneten Winkel gedrehten) „Speiche/Reifen-Einheiten“.

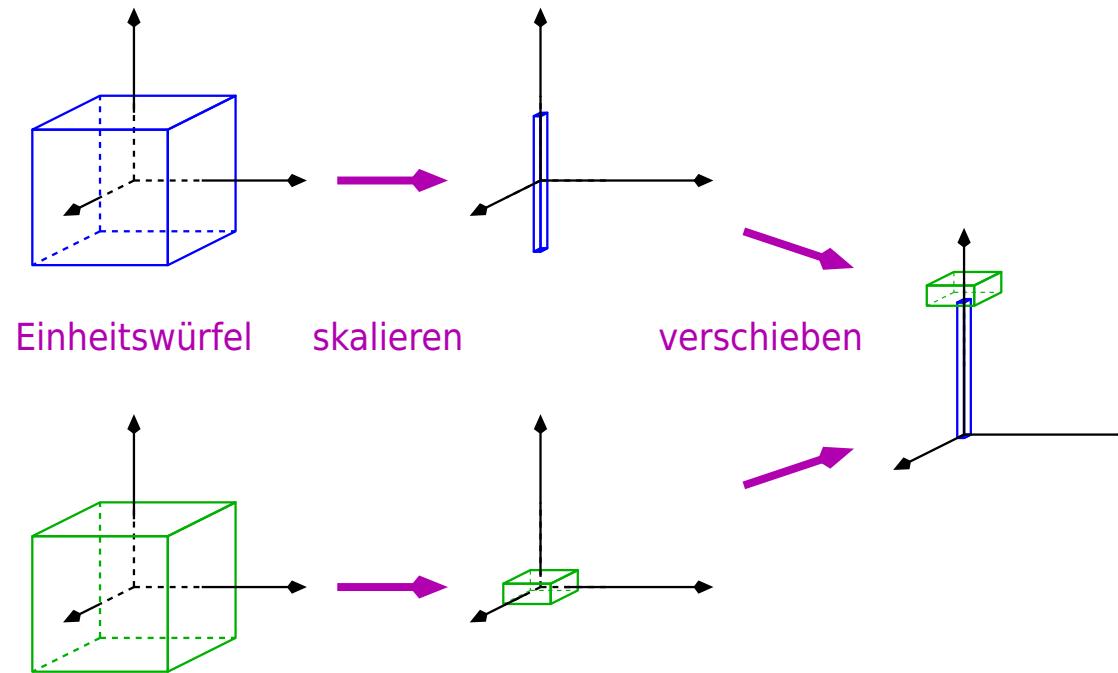


Eine **Speiche/Reifen-Einheit** besteht aus

- einer Speiche und
- einem Stück des Reifens

(beide modelliert durch Quader).

Ein **Quader** kann durch geeignete Streckung aus einem **Einheitswürfel** $[-0,5; 0,5]^3$ erzeugt werden.



Bei der Projektion auf 2D müssen die folgenden Transformationen auf die Ecken $\mathbf{P} \in \mathbb{R}^3$ von Einheitswürfeln angewandt werden

(ausgehend von projektiven Koordinaten $\mathbf{P}' = \begin{pmatrix} \tau \mathbf{P} \\ \tau \end{pmatrix}$ erhält man auch projektive 2D-Koordinaten $\mathbf{u}' = \begin{pmatrix} \tau' u \\ \tau' v \\ \tau' \end{pmatrix}$):

- wenn der Einheitswürfel dargestellt werden soll:

$$\mathbf{u}' = \mathbf{M}'_{\text{proj}} \cdot \underbrace{\mathbf{P}'}_{\substack{\text{Ecke des Würfels}}$$

mit

- $\mathbf{M}'_{\text{proj}}$: Projektionsmatrix $3D \rightarrow 2D$
- wenn der Würfel in ein Element der Speiche/Reifen-Einheit (**Speiche** oder **Stück des Reifens**) transformiert wird:

$$\mathbf{u}' = \mathbf{M}'_{\text{proj}} \cdot \underbrace{\mathbf{M}'_{\text{tr_sr}_s} \cdot \mathbf{M}'_{\text{sc_sr}_s}}_{\substack{\text{Transformation zur Speiche}}} \cdot \underbrace{\mathbf{P}'}_{\substack{\text{Würfel}}} \quad \text{bzw.}$$

$$\mathbf{u}' = \mathbf{M}'_{\text{proj}} \cdot \underbrace{\mathbf{M}'_{\text{tr_sr}_r} \cdot \mathbf{M}'_{\text{sc_sr}_r}}_{\substack{\text{Transformation zum Reifenstück}}} \cdot \underbrace{\mathbf{P}'}_{\substack{\text{Würfel}}}$$

mit

- $\mathbf{M}'_{\text{sc_sr}_s}$ bzw. $\mathbf{M}'_{\text{sc_sr}_r}$: (Diagonal-)Streckungsmatrix („scale“) für die Speiche bzw. das Reifenstück
- $\mathbf{M}'_{\text{tr_sr}_s}$ bzw. $\mathbf{M}'_{\text{tr_sr}_r}$: anschließende Verschiebung („translate“)

- wenn die Speiche/Reifen-Einheit anschließend in ein Rad eingebaut wird:

$$\boldsymbol{u}' = \boldsymbol{M}'_{\text{proj}} \cdot \underbrace{\boldsymbol{M}'_{\text{rot_rad}}}_{\text{Einbau in Rad}} \cdot \underbrace{\boldsymbol{M}'_{\text{tr_sr}} \cdot \boldsymbol{M}'_{\text{sc_sr}} \cdot \boldsymbol{P}'}_{\text{Speiche/Reifen-Einheit}}$$

mit

- $\boldsymbol{M}'_{\text{rot_rad}}$: Rotationsmatrix („rotate“) um die (Naben-) Achse
- wenn das Rad hinten in den Tretroller eingebaut wird:

$$\boldsymbol{u}' = \boldsymbol{M}'_{\text{proj}} \cdot \underbrace{\boldsymbol{M}'_{\text{tr_hr}} \cdot \boldsymbol{M}'_{\text{rot_hr}}}_{\text{Einbau in Roller}} \cdot \underbrace{\boldsymbol{M}'_{\text{rot_rad}} \cdot \boldsymbol{M}'_{\text{tr_sr}} \cdot \boldsymbol{M}'_{\text{sc_sr}} \cdot \boldsymbol{P}'}_{\text{Rad}}$$

mit

- $\boldsymbol{M}'_{\text{rot_hr}}$: Drehung des Hinterrades
- $\boldsymbol{M}'_{\text{tr_hr}}$: anschließende Verschiebung an die Position im Roller

- wenn das „Trittbrett“ in den Tretroller eingebaut wird:

$$\mathbf{u}' = \mathbf{M}'_{\text{proj}} \cdot \underbrace{\mathbf{M}'_{\text{tr_brett}} \cdot \mathbf{M}'_{\text{sc_brett}}}_{\text{Einbau als Trittbrett}} \cdot \underbrace{\mathbf{P}'}_{\text{Würfel}}$$

mit

- $\mathbf{M}'_{\text{sc_brett}}$: Streckungsmatrix für das Trittbrett
- $\mathbf{M}'_{\text{tr_brett}}$: anschließende Verschiebung an die Position im Roller

generelles Vorgehen: ist eine „Struktur“ (z. B. Tretroller) aus „Unterstrukturen“ (z. B. Lenkergruppe, Quader für Trittbrett usw., Hinterrad) aufgebaut, dann

- vor dem „Aufruf“ jeder Unterstruktur die zum Einbau erforderlichen Transformationen setzen,
 - die Unterstruktur aufrufen (und dabei ggf. rekursiv genauso verfahren),
 - anschließend die „Einbau“transformationen wieder zurücksetzen
- ⇒ Verwaltung der Transformationsmatrizen mittels **Stack**

Bemerkung 4.5: Damit am Ende jede Ecke der Szene mit einer einzigen Matrix-Vektor-Multiplikation transformiert werden kann, werden bei **OpenGL** stets alle „bislang bekannten“ Matrizen aufmultipliziert.

Beispielsweise liegt vor dem Einbau einer Speiche/Reifen-Einheit in das Hinterrad die Matrix

$$\mathbf{M}' := \mathbf{M}'_{\text{proj}} \cdot \color{red}{\mathbf{M}'_{\text{tr_hr}}} \cdot \color{red}{\mathbf{M}'_{\text{rot_hr}}} \cdot \color{green}{\mathbf{M}'_{\text{rot_rad}}}$$

vor, innerhalb der Speiche/Reifen-Einheit wird sie durch

$$\mathbf{M}' \cdot \color{blue}{\mathbf{M}'_{\text{tr_sr}_s}} \cdot \color{blue}{\mathbf{M}'_{\text{sc_sr}_s}}$$

ersetzt.

Die auf eine Unterstruktur anzuwendenden Transformationen sind daher **in umgekehrter Reihenfolge** anzugeben.

Tretroller/roller.c

```
22 // eine Speiche und ein dazu gehöriges Stück des Reifens zeichnen
23 void zeichneSpeicheReifen(void)
24 {
25     // für die Speiche die Farbe auf Blau setzen
26     // (Beiträge an Rot, Grün, Blau)
27     glColor3f(0.0, 0.0, 1.0);
28
29     // alte Transformationsmatrix sichern
30     glPushMatrix();
31     // aus einem Einheitswürfel [-0,5; 0,5]3 durch
32     // Skalierung eine Speiche erzeugen und danach(!)
33     // verschieben, so dass sie im Ursprung anliegt
34     glTranslated(0.0, 0.5, 0.0);
35     glScaled(0.02, 1.0, 0.02);
36     glutSolidCube(1.0);
37     // alte Transformation wiederherstellen
38     glPopMatrix();
39
40     // analog für grünes Stück Reifen
41     glColor3f(0.0, 1.0, 0.0);
42     glPushMatrix();
43     glTranslated(0.0, 1.00, 0.00);
44     glScaled(0.3, 0.15, 0.1);
45     glutSolidCube(1.0);
46     glPopMatrix();
47 }
48
49 // ein ausanzahl_speichen bestehendes Rad zeichnen
50 void zeichneRad(int anzahlSpeichen)
51 {
52     glPushMatrix();
53
54     // das Rad ausanzahlSpeichen Speichen zusammensetzen
55     // (jeweils um 360 /anzahlSpeichen Grad WEITERdrehen)
56     for (int k = 0; k < anzahlSpeichen; k++)
57     {
58         glRotated(360.0 / anzahlSpeichen, 0.0, 0.0, 1.0);
59         zeichneSpeicheReifen();
60     }
61
62 } glPopMatrix();
```

```
65 // Die aus Vordergabel, vertikaler Stange und horizontaler
66 // Lenkerstange bestehende Lenkergruppe zeichnen
67 void zeichneLenkergruppe(void)
68 {
69     // Vorderrad zeichnen
70     glPushMatrix();
71     glScaled(0.6, 0.6, 1.8);
72     zeichneRad(16);
73     glPopMatrix();
74
75     // roten Rahmen dazu zeichnen
76     glColor3f(1.0, 0.0, 0.0);
77
78     // Linke Strebe der Radgabel
79     glPushMatrix();
80     glTranslated(0.0, 0.35, -0.2);
81     glScaled(0.1, 0.8, 0.05);
82     glutSolidCube(1.0);
83     glPopMatrix();
84
85     // vier weitere Quader für rechte und horizontale Strebe
86     // der Gabel, vertikale Stange und Lenkerstange
87
88     // ...
89
90     // Den kompletten Tretroller zeichnen; dabei die Lenkergruppe
91     // und das Hinterrad je nach verstrichener Zeit (angegeben in
92     // Minuten) drehen.
93     void zeichneRoller(double Minuten)
94     {
95         // gedrehte Lenkergruppe zeichnen
96         glPushMatrix();
97         glRotated(minuten * -0.2, 0.0, 1.0, 0.0);
98         zeichneLenkergruppe();
99         glPopMatrix();
100
101        // Trittbrett zeichnen
102        glPushMatrix();
103        glTranslated(1.8, 0.0, 0.0);
104        glScaled(2.0, 0.1, 0.4);
105        glutSolidCube(1.0);
106        glPopMatrix();
107
108        // vier weitere Quader für Befestigung an Lenkergruppe
109        // und Hinterradgabel zeichnen
110
111        // ...
```

```
160 // gedrehtes Hinterrad zeichnen
161 glPushMatrix();
162 glTranslated(3.65, 0.0, 0.0);
163 glRotated(minuten * -1.0, 0.0, 0.0, 1.0);
164 glScaled(0.6, 0.6, 1.2);
165 zeichneRad(32);
166 glPopMatrix();
167 }
```

5 Clipping für Strecken und Polygone

Inhalt

5.1 Der Pixel-orientierte Ansatz	5-2
5.2 Der 2D-analytische Ansatz	5-5
5.2.1 Strecken-Clipping naiv	5-7
5.2.2 Strecken-Clipping nach Cohen und Sutherland	5-11
5.2.3 Strecken-Clipping nach Cyrus, Beck, Liang und Barsky	5-17
5.2.4 Polygon-Clipping nach Sutherland/Hodgman	5-23
5.2.5 Hardware-Realisierung	5-28
5.3 Der 3D-analytische Ansatz	5-32

Problem: Alle Strecken- und Polygoneiteile, die nach der Projektion außerhalb des Projektionsfensters liegen, müssen unterdrückt werden.

5.1 Der Pixel-orientierte Ansatz

Idee: Entscheide erst beim Eintragen in die Pixmap, welche Pixel tatsächlich geändert werden.

Algorithmus 5.1:

Algorithmus Pixel-basiertes Clipping

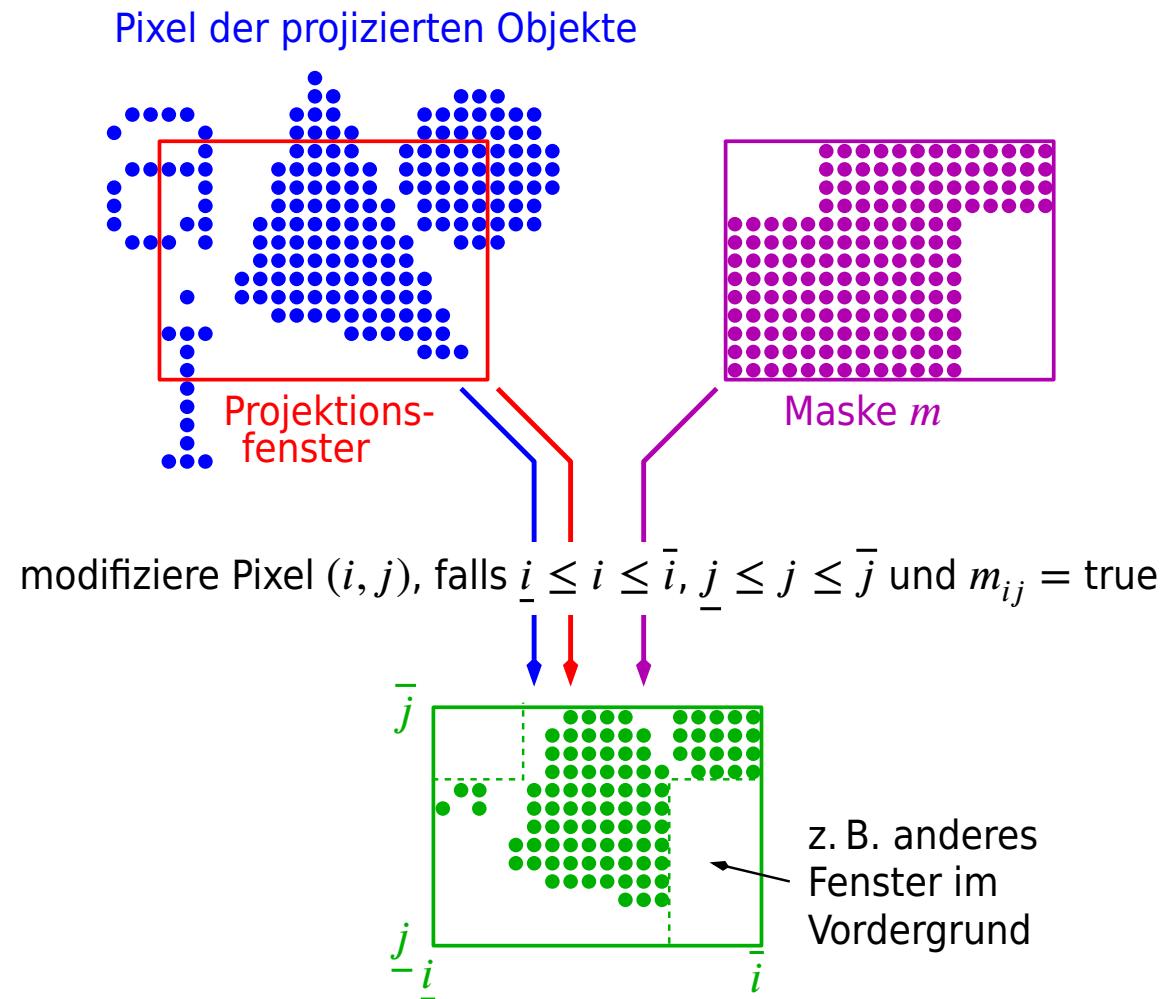
bestimme den Pixelbereich $[i \dots \bar{i}] \times [j \dots \bar{j}]$, auf den das Projektionsfenster abgebildet wird
für jedes Objekt in 3D

bilde die „relevanten Größen“ (z. B. Endpunkte von Strecken) auf Pixelkoordinaten ab
führe Scan Conversion für das Objekt durch

für alle hierbei erzeugten Pixel (i, j)

modifiziere Pixel (i, j) , falls $i \leq i \leq \bar{i}$ und $j \leq j \leq \bar{j}$

- ⊖ Unter Umständen wird viel unnötige Scan Conversion durchgeführt (Objekte, die gar nicht oder nur zu einem geringen Bruchteil sichtbar sind).
- ⊖ erlaubt kein 3D-Clipping
- ⊖ nur für Pixel-orientierte Ausgabegeräte anwendbar
- ⊕ extrem einfach
 - ⇒ auch sehr gut in Hardware realisierbar
- ⊕ erlaubt Clipping beliebiger Objekte; der Aufwand ist unabhängig von der „Komplexität“ des Objekts (er ist proportional zur Anzahl der aus der Scan Conversion resultierenden Pixel)
- ⊕ erlaubt Clipping gegenüber beliebigen Flächen, nicht nur gegenüber dem Projektionsfenster:



5.2 Der 2D-analytische Ansatz

Methode: Bestimme in der Projektionsebene die sichtbaren (Teile der) Objekte.

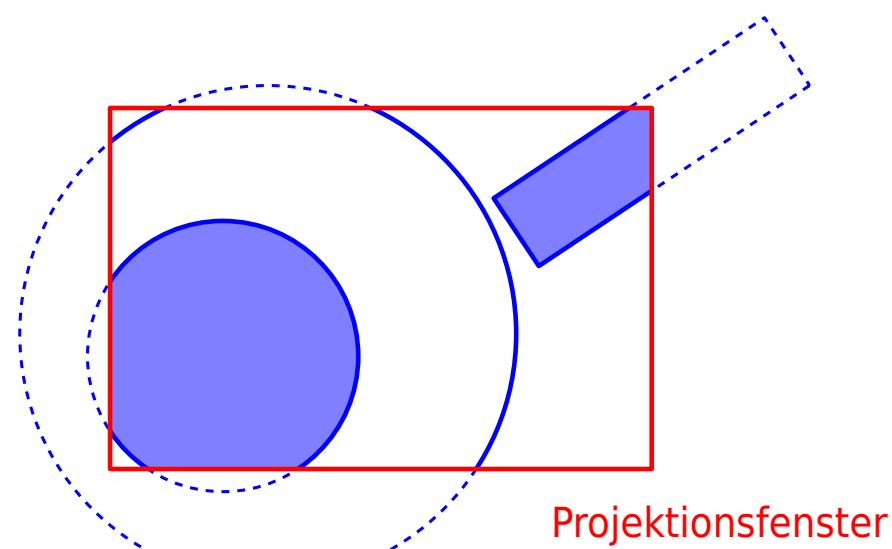
Algorithmus 5.2:

Algorithmus Analytisches Clipping, 2D

für jedes Objekt in 3D
projiziere das Objekt auf die Projektionsebene
schneide das Bild des Objekts mit dem Projektionsfenster
für alle sichtbaren Teilobjekte
bilde das Teilobjekt auf Pixelkoordinaten ab
führe Scan Conversion durch
modifiziere die hierbei erzeugten Pixel in derPixmap

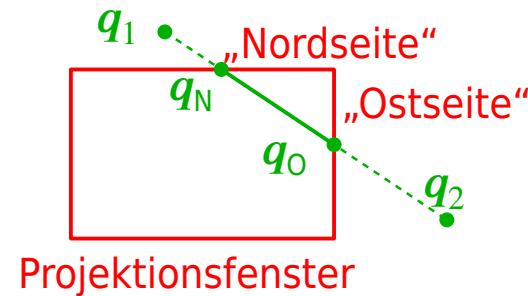
- ⊕ Scan Conversion wird nur für die tatsächlich sichtbaren Teile durchgeführt.
- ⊕ Clipping ist nur einmal durchzuführen, auch wenn die Ausgabe auf mehreren Geräten erfolgt (z. B. zuerst auf dem Bildschirm, dann auf dem Drucker).
- ⊕ auch für Vektor-orientierte Ausgabegeräte geeignet

- ⊖ erlaubt kein 3D-Clipping
- ⊖ Auch die nicht sichtbaren Teile müssen projiziert werden.
- ⊖ deutlich komplexer als der Pixel-orientierte Ansatz:
 - reelle Berechnungen
 - Die in die Scan Conversion eingehenden Objekte werden komplizierter.
⇒ aufwändigere Algorithmen für die Scan Conversion.



5.2.1 Strecken-Clipping naiv

Ansatz: Bestimme die Schnittpunkte der Strecke mit den Seiten des Projektionsfensters; dies liefert den sichtbaren Teil der Strecke.



Algorithmus 5.3:**Algorithmus Strecken-Clipping naiv**

```
// seien  $q_1 = (u_1, v_1)$  und  $q_2 = (u_2, v_2)$  die Endpunkte der Strecke s
E :=  $\emptyset$       // Endpunkte des sichtbaren Teils von s
für  $i = 1, 2$     // liegen die Endpunkte von s im Fenster?
  wenn  $u \leq u_i \leq \bar{u}$  und  $v \leq v_i \leq \bar{v}$ 
    E := E  $\cup \{q_i\}$ 
  für „Ost“-, „Nord“-, „West“- und „Süd“-Seite des Fensters
    wenn s die Seite in einem Punkt q schneidet
      E := E  $\cup \{q\}$ 
  wenn  $E \neq \emptyset$     // dann enthält E i. Allg. genau zwei Punkte  $q'$ ,  $q''$ 
    die Strecke  $s' = \overline{q'q''}$  ist sichtbar
```

Berechnung der Schnittpunkte (etwa mit „West“: $u = \underline{u}$):

- Parameterdarstellung der zu s gehörenden **Geraden**:

$$\begin{aligned} q &= q_1 + \lambda \cdot (q_2 - q_1) , \quad \lambda \in \mathbb{R} \\ \Leftrightarrow \begin{pmatrix} u \\ v \end{pmatrix} &= \begin{pmatrix} u_1 \\ v_1 \end{pmatrix} + \lambda \cdot \begin{pmatrix} u_2 - u_1 \\ v_2 - v_1 \end{pmatrix} \\ \Rightarrow \quad \underline{u} &= u_1 + \lambda \cdot (u_2 - u_1) \\ \Leftrightarrow \quad \lambda &= \frac{\underline{u} - u_1}{u_2 - u_1} \end{aligned}$$

- Der Schnittpunkt liegt auf der **Strecke** s , falls

$$0 \leq \lambda \leq 1 .$$

- Der Schnittpunkt liegt auf der **Westseite** des Fensters, falls

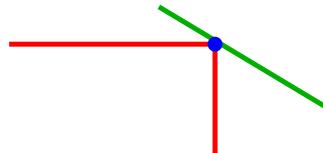
$$\underline{v} \leq v = v_1 + \lambda \cdot (v_2 - v_1) \leq \bar{v} .$$

Sonderfälle:

- Strecke parallel zu einer Seite des Projektionsfensters



- Ein (Schnitt-)Punkt wird mehrfach berechnet:



- E enthält am Ende nur einen Punkt:

numerisches Problem: Ein bei exakter Rechnung mehrfach auftretender Schnittpunkt kann durch **Rundungsfehler** in mehrere Punkte „zerfallen“.

⇒ E kann am Ende mehr als zwei verschiedene Punkte enthalten!

Fazit:

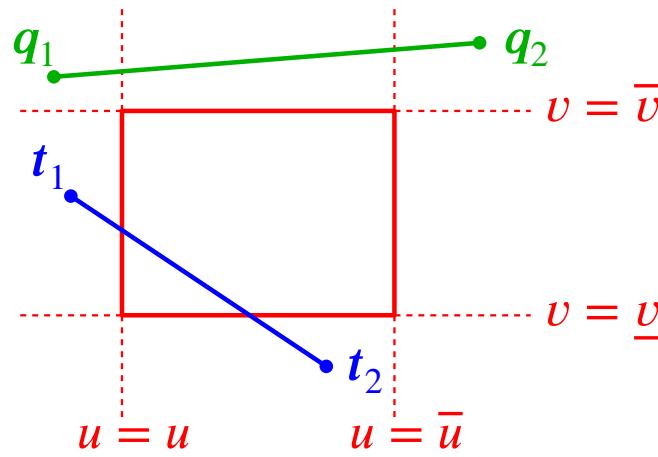
- relativ viel Rechnung: für die vier Seiten jeweils

4 Additionen + 1 Multiplikation + 1 Division + 4 Vergleiche

- Von den vier Schnitttests sind mindestens zwei überflüssig!
- viele Sonderfälle

5.2.2 Strecken-Clipping nach Cohen und Sutherland

Idee: Manchmal ist sehr leicht feststellbar, dass die Strecke eine bestimmte Fensterseite gar nicht schneiden kann oder sogar völlig innerhalb bzw. außerhalb des Fensters liegt.



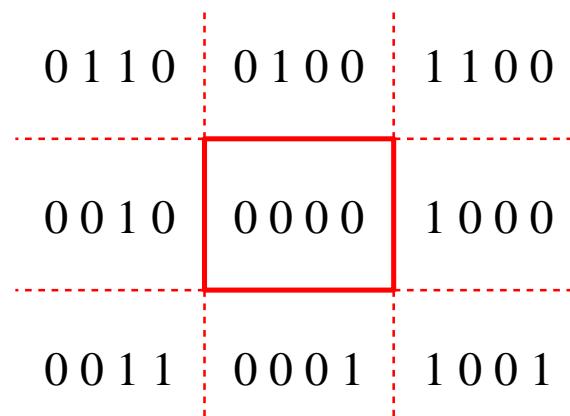
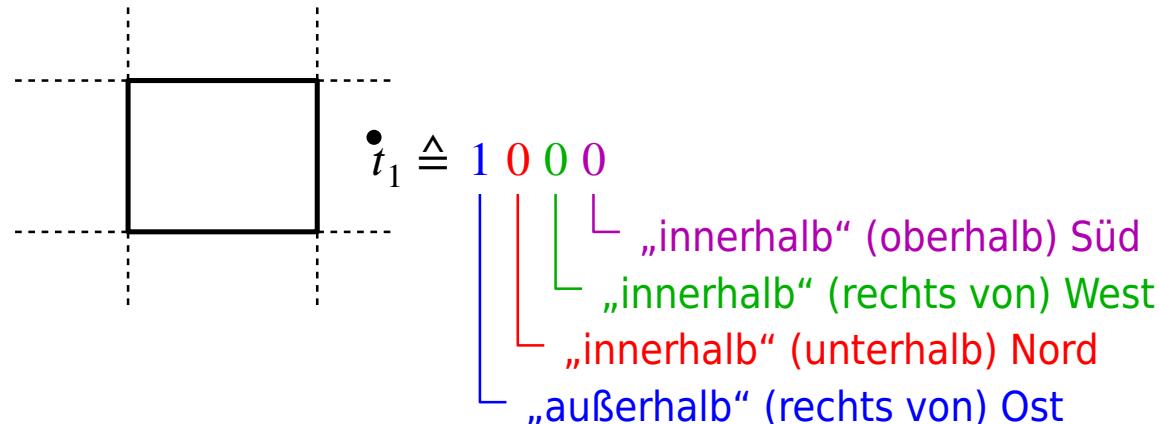
- $\overline{q_1 q_2}$ ist unsichtbar, denn beide Endpunkte liegen oberhalb der **Geraden** $v = \bar{v}$.
- $\overline{t_1 t_2}$ kann die Nordseite des Fensters nicht schneiden, da beide Endpunkte auf derselben Seite der Geraden $v = \bar{v}$ liegen.

(analog: kein Schnitt mit „Ost“)

klar: Eine Strecke s liefert höchstens dann einen Schnittpunkt mit der Nord- (West-, ...) Seite des Fensters, wenn die Endpunkte von s auf verschiedenen Seiten der entsprechenden Geraden liegen.

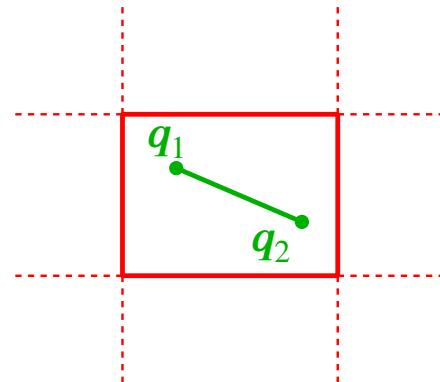
Ziel: diese Tests möglich effizient durchführen

Ansatz: Ordne jedem Endpunkt der Strecke einen **4-stelligen Binärcode** zu, der die Lage des Punktes bzgl. der vier „Fenstergeraden“ angibt:



Beispiele 5.4:

1.

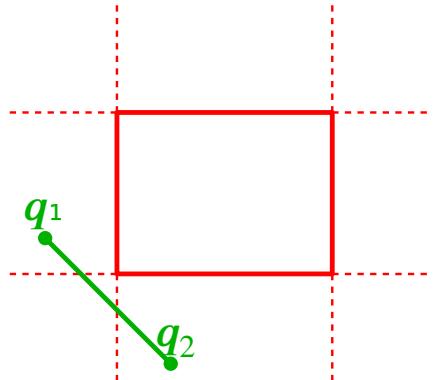


$$\text{code}_1 = 0000$$

$$\text{code}_2 = 0000$$

⇒ vollständig sichtbar

2.

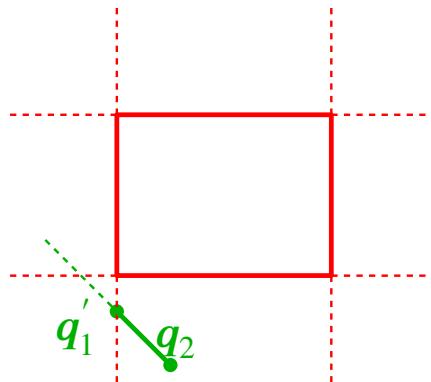


$$\text{code}_1 = 0010 \text{ (ONWS)}$$

$$\text{code}_2 = 0001$$

⇒ Strecke schneidet die West- und die Süd-Gerade

↓ Clipping an der West-Geraden



$$\text{code}'_1 = 0001 \text{ (ONWS)}$$

$$\text{code}_2 = 0001$$

⇒ Strecke liegt ganz unterhalb der Süd-Geraden

⇒ unsichtbar

Algorithmus 5.5:**Algorithmus Cohen/Sutherland**

wiederhole

code₁ := ($u_1 > \bar{u}, v_1 > \bar{v}, u_1 < \underline{u}, v_1 < \underline{v}$)

code₂ := ($u_2 > \bar{u}, v_2 > \bar{v}, u_2 < \underline{u}, v_2 < \underline{v}$)

wenn (**code**₁ and **code**₂) ≠ 0000

fertig: Strecke unsichtbar // beide Endpunkte liegen „außerhalb“ derselben Geraden

sonst

wenn (**code** := (**code**₁ or **code**₂)) = 0000

fertig: Strecke sichtbar // beide Endpunkte liegen im Fenster

sonst

sei i das erste 1-Bit in **code**

bestimme den Schnittpunkt q der Strecke $\overline{q_1 q_2}$ mit der Fenstergeraden zu Bit i

wenn Bit i in **code**₁ gesetzt ist

ersetze q_1 durch q

sonst

ersetze q_2 durch q

bis fertig

Danny Cohen (דני כהן)

* ?, Israel

Mathematiker, Informatiker

Foto: Kvgd, Titel: „2009-DannyCohen“

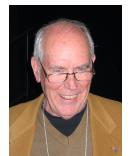
Quelle: <https://commons.wikimedia.org/wiki/File:2009-DannyCohen.jpg>Lizenz: CC BY-SA 3.0, <https://creativecommons.org/licenses/by-sa/3.0/legalcode>

Ivan Edward Sutherland

* 1938, Hastings, Nebraska

Elektroingenieur, Informatiker

Foto: Dick Lyon, Titel: „Ivan Sutherland at CHM“

Quelle: https://commons.wikimedia.org/wiki/File:Ivan_Sutherland_at_CHM.jpgLizenz: CC BY-SA 3.0, <https://creativecommons.org/licenses/by-sa/3.0/legalcode>

Berechnung der Schnittpunkte z. B. mit der Steigungsform

$$v = v_1 + \underbrace{\frac{v_2 - v_1}{u_2 - u_1}}_{=: m_v} \cdot (u - u_1)$$

bzw.

$$u = u_1 + \underbrace{\frac{u_2 - u_1}{v_2 - v_1}}_{=: m_u} \cdot (v - v_1)$$

durch Einsetzen von $u = \underline{u}$, usw.

⇒ 4 Additionen + 1 Multiplikation + 1 Division, wenn die Steigung berechnet werden muss

2 Additionen + 1 Multiplikation, sonst

(m_u und m_v werden höchstens einmal berechnet.)

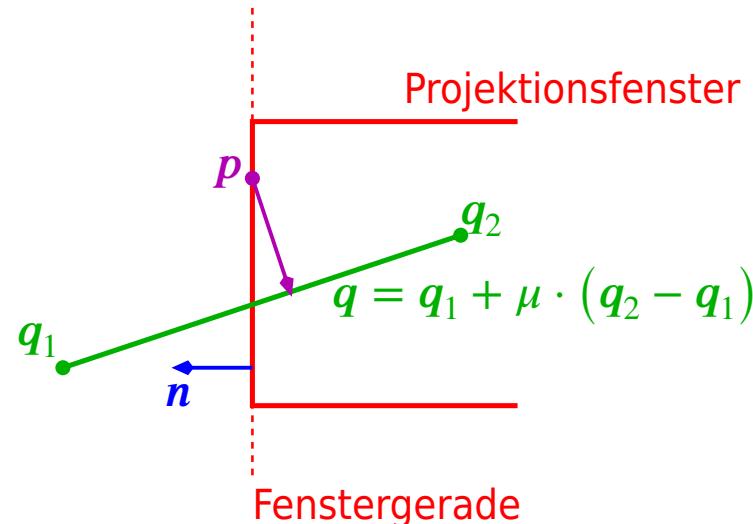
- ⊕ sehr schnell, wenn die meisten Strecken bereits beim ersten Test akzeptiert oder abgelehnt werden können
- ⊖ relativ viele Operationen, wenn mehrere Schnittpunkte bestimmt werden müssen
- ⊖ nicht effizient, wenn gegenüber einem achsenparallelen Rechteck geclipppt wird

5.2.3 Strecken-Clipping nach Cyrus, Beck, Liang und Barsky

Idee: die Strecke sukzessive durch Schneiden mit den Fenstergeraden verkürzen, wie bei Cohen/Sutherland

aber: Schnittpunkte aus der Parameterdarstellung der Strecke gewinnen,

denn: Wenn der Parameterwert des Schnittpunkts bekannt ist, muss der Schnittpunkt selbst oft gar nicht mehr explizit berechnet werden!



Seien

p ein beliebiger (aber fester) Punkt auf der Fenstergeraden,

n ein **nach außen** weisender Normalenvektor zur Fenstergeraden.

Ein Punkt $\mathbf{q} = \mathbf{q}_1 + \mu (\mathbf{q}_2 - \mathbf{q}_1)$ der Geraden $\mathbf{q}_1\mathbf{q}_2$ liegt genau dann auf der Fenstergeraden, wenn

$$\begin{aligned} & \mathbf{q} - \mathbf{p} \perp \mathbf{n} \\ \Leftrightarrow & \mathbf{n}^T \cdot (\mathbf{q}_1 + \mu (\mathbf{q}_2 - \mathbf{q}_1) - \mathbf{p}) = 0 \\ \Leftrightarrow & \mathbf{n}^T \cdot (\mathbf{q}_1 - \mathbf{p}) + \mu \cdot \mathbf{n}^T \cdot (\mathbf{q}_2 - \mathbf{q}_1) = 0 \\ \Leftrightarrow & \mu = \frac{\mathbf{n}^T \cdot (\mathbf{q}_1 - \mathbf{p})}{-\mathbf{n}^T \cdot (\mathbf{q}_2 - \mathbf{q}_1)}. \end{aligned}$$

Der Schnittpunkt wird klassifiziert als:

In, falls $-\mathbf{n}^T \cdot (\mathbf{q}_2 - \mathbf{q}_1) > 0$

(d.h. wenn man von \mathbf{q}_1 nach \mathbf{q}_2 läuft, geht man über die Fenstergerade in die Halbebene, in der auch das Projektionsfenster liegt)

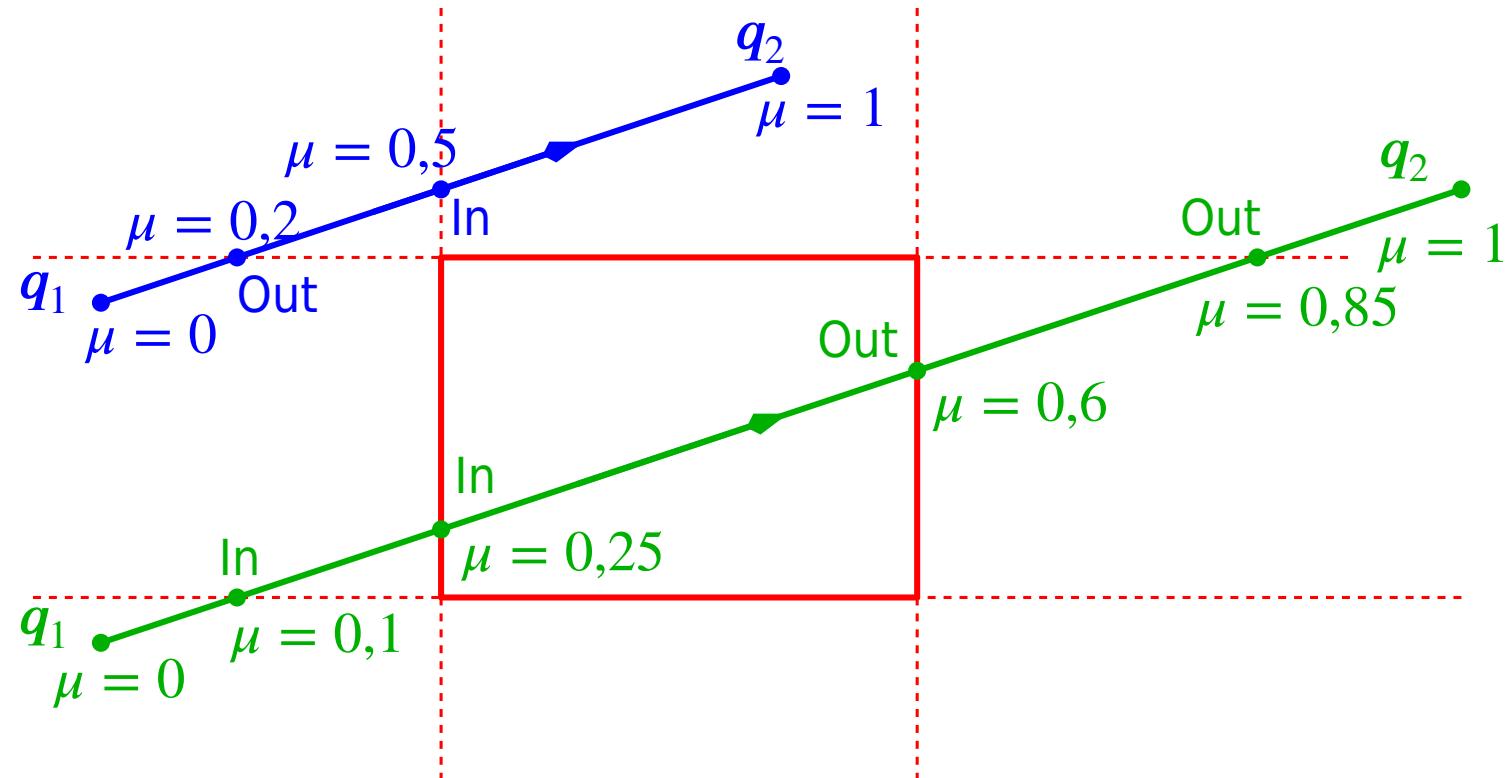
Out, falls $-\mathbf{n}^T \cdot (\mathbf{q}_2 - \mathbf{q}_1) < 0$

⇒ Ist \mathbf{q} ein In-Schnittpunkt, so kann der Abschnitt $\overline{\mathbf{q}_1\mathbf{q}}$ der Strecke weggeworfen werden, bei einem Out-Schnittpunkt der Abschnitt $\overline{\mathbf{q}\mathbf{q}_2}$.

Geeignete Wahlen für p und n

Fenstergerade	Normale \mathbf{n}	Punkt \mathbf{p}	$\mathbf{q}_1 - \mathbf{p}$	$\mu = \frac{\mathbf{n}^T (\mathbf{q}_1 - \mathbf{p})}{-\mathbf{n}^T (\mathbf{q}_2 - \mathbf{q}_1)}$
O: $u = \bar{u}$	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} \bar{u} \\ \tilde{v} \end{pmatrix}$	$\begin{pmatrix} u_1 - \bar{u} \\ v_1 - \tilde{v} \end{pmatrix}$	$\frac{u_1 - \bar{u}}{u_1 - u_2}$
N: $v = \bar{v}$	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} \tilde{u} \\ \bar{v} \end{pmatrix}$	$\begin{pmatrix} u_1 - \tilde{u} \\ v_1 - \bar{v} \end{pmatrix}$	$\frac{v_1 - \bar{v}}{v_1 - v_2}$
W: $u = \underline{u}$	$\begin{pmatrix} -1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} \underline{u} \\ \tilde{v} \end{pmatrix}$	$\begin{pmatrix} u_1 - \underline{u} \\ v_1 - \tilde{v} \end{pmatrix}$	$\frac{\underline{u} - u_1}{u_2 - u_1}$
S: $v = \underline{v}$	$\begin{pmatrix} 0 \\ -1 \end{pmatrix}$	$\begin{pmatrix} \tilde{u} \\ \underline{v} \end{pmatrix}$	$\begin{pmatrix} u_1 - \tilde{u} \\ v_1 - \underline{v} \end{pmatrix}$	$\frac{\underline{v} - v_1}{v_2 - v_1}$

(\tilde{u} und \tilde{v} sind beliebig wählbar.)

Beispiel 5.6:

Algorithmus 5.7:**Algorithmus Cyrus, Beck, Liang, Barsky**

$[\mu_{\text{In}}; \mu_{\text{Out}}] := [0; 1]$ // μ_{In} und μ_{Out} begrenzen den übrig gebliebenen Teil der Strecke

$\Delta u := u_2 - u_1$

$\Delta v := v_2 - v_1$

// der Reihe nach an den „0“-, „N“-, „W“- und „S“-Seiten des Projektionsfensters abschneiden

wenn noch_etwas_übrig($u_1 - \bar{u}$, $-\Delta u$, μ_{In} , μ_{Out})

wenn noch_etwas_übrig($v_1 - \bar{v}$, $-\Delta v$, μ_{In} , μ_{Out})

wenn noch_etwas_übrig($u - u_1$, Δu , μ_{In} , μ_{Out})

wenn noch_etwas_übrig($v - v_1$, Δv , μ_{In} , μ_{Out})

/* die Strecke ist – zumindest teilweise – sichtbar;
berechne die Endpunkte des sichtbaren Teils */

wenn $\mu_{\text{Out}} < 1$

$$q_2 := q_1 + \mu_{\text{Out}} \cdot \underbrace{\begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}}_{=q_2 - q_1}$$

Mike Cyrus

Jay Beck

You-Dong Liang (梁友栋 [Liáng Yǒudòng])
* 1935, Fuzhou (福州) (Fujian [福建], CN)
Mathematiker

wenn $\mu_{\text{In}} > 0$

$$q_1 := q_1 + \mu_{\text{In}} \cdot \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

Brian A. Barsky
Informatiker

Algorithmus 5.8:

Algorithmus noch_etwas_übrig($Z, N, \mu_{\text{In}}, \mu_{\text{Out}}$)

/* führt das durch Zähler Z und Nenner N gegebene Clipping an einer Fenstergeraden durch und modifiziert μ_{In} oder μ_{Out} entsprechend.
Zurückgegeben wird, ob jetzt noch ein Teil der Strecke sichtbar ist. */

wenn $N > 0$ // In-Schnittpunkt

$$\mu_{\text{In}} := \max \left\{ \mu_{\text{In}}, \underbrace{\frac{Z}{N}}_{\mu} \right\}$$

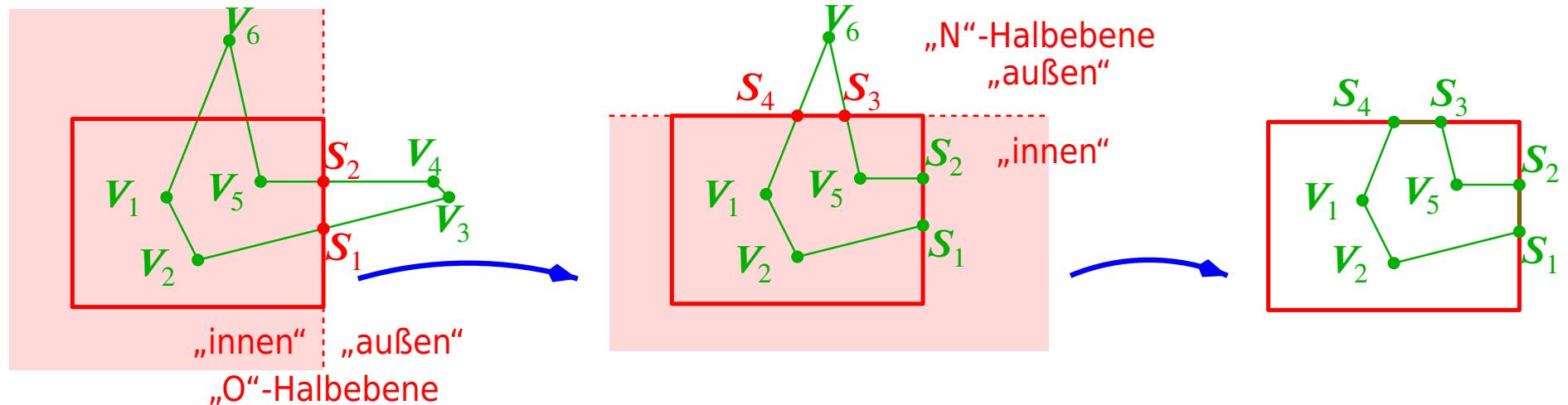
noch_etwas_übrig := ($\mu_{\text{In}} \leq \mu_{\text{Out}}$)
 sonst wenn $N < 0$ // Out-Schnittpunkt
 $\mu_{\text{Out}} := \min \left\{ \mu_{\text{Out}}, \frac{Z}{N} \right\}$
 noch_etwas_übrig := ($\mu_{\text{In}} \leq \mu_{\text{Out}}$)
 sonst // Nenner $N = 0$: Strecke parallel zur Fenstergeraden
 noch_etwas_übrig := ($Z \leq 0$)
 // Strecke liegt innerhalb/außerhalb der Halbebene, in der auch das Fenster liegt

5.2.4 Polygon-Clipping nach Sutherland/Hodgman

Vorbemerkung: Polygon-Clipping ist wesentlich komplizierter als Clipping für einzelne Strecken!

Gary W. Hodgman

Ansatz: Clippe das Polygon der Reihe nach gegenüber den vier Halbebene, die durch die Fensterseiten gegeben sind:



Ziel: Folge von Ecken für das „Eingabepolygon“



Algorithmus für Clipping bzgl. einer Halbebene



Folge von Ecken für das „Ausgabepolygon“

Methode: Durchlaufe die Ecken bzw. Kanten des Eingabepolygons.

Gib dabei die entsprechenden Ecken des Ausgabepolygons aus.

Beispiel 5.9: (obiges Beispiel, Clipping bzgl. „O“-Halbebene)

- starte in Ecke V_1
 V_1 liegt in der Halbebene \Rightarrow Ausgabe V_1
- Kante $\overline{V_1V_2}$ liegt ganz in der Halbebene:
- Kante $\overline{V_2V_3}$ verlässt die Halbebene im Punkt S_1 :

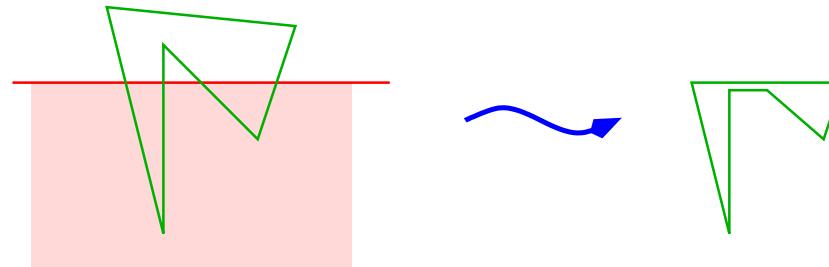
- Kante $\overline{V_3 V_4}$ liegt ganz außerhalb der Halbebene:
- Kante $\overline{V_4 V_5}$ betritt wieder die Halbebene in S_2 :
- Kante $\overline{V_5 V_6}$ liegt wieder ganz in der („O“-)Halbebene:
 - Fall 1 \Rightarrow Ausgabe V_6
- auch Kante $\overline{V_6 V_1}$ liegt ganz in der Halbebene:
 - Fall 1 \Rightarrow Ausgabe V_1

geclippes Polygon: $V_1, V_2, S_1, S_2, V_5, V_6, V_1$

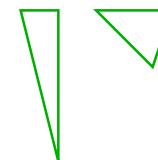
danach Clipping bzgl. der „N“-, „W“- und „S“-Halbebenen

Bemerkungen 5.10:

1. Es treten nur die angegebenen vier Fälle auf.
 2. gut geeignet, wenn viele Strecken tatsächlich geclipppt werden müssen
 3. leicht verallgemeinerbar auf
 - nicht achsenparallele Clip-Fenster
 - konvexe Polygone als Clip-Fenster
 - 3D-Clipping
-

Problemfälle:

Ist diese (zusammenhängende) Lösung „korrekt“, oder müssen in diesem Fall zwei disjunkte Dreiecke geliefert werden?



Antwort: Das ist abhängig von der weiteren Verwendung der Polygone, z. B.

- gelieferte Lösung ist okay, wenn nur das **Innere** der Polygone gezeichnet wird,
- nicht okay, wenn auch (oder nur) der **Rand** gezeichnet wird.

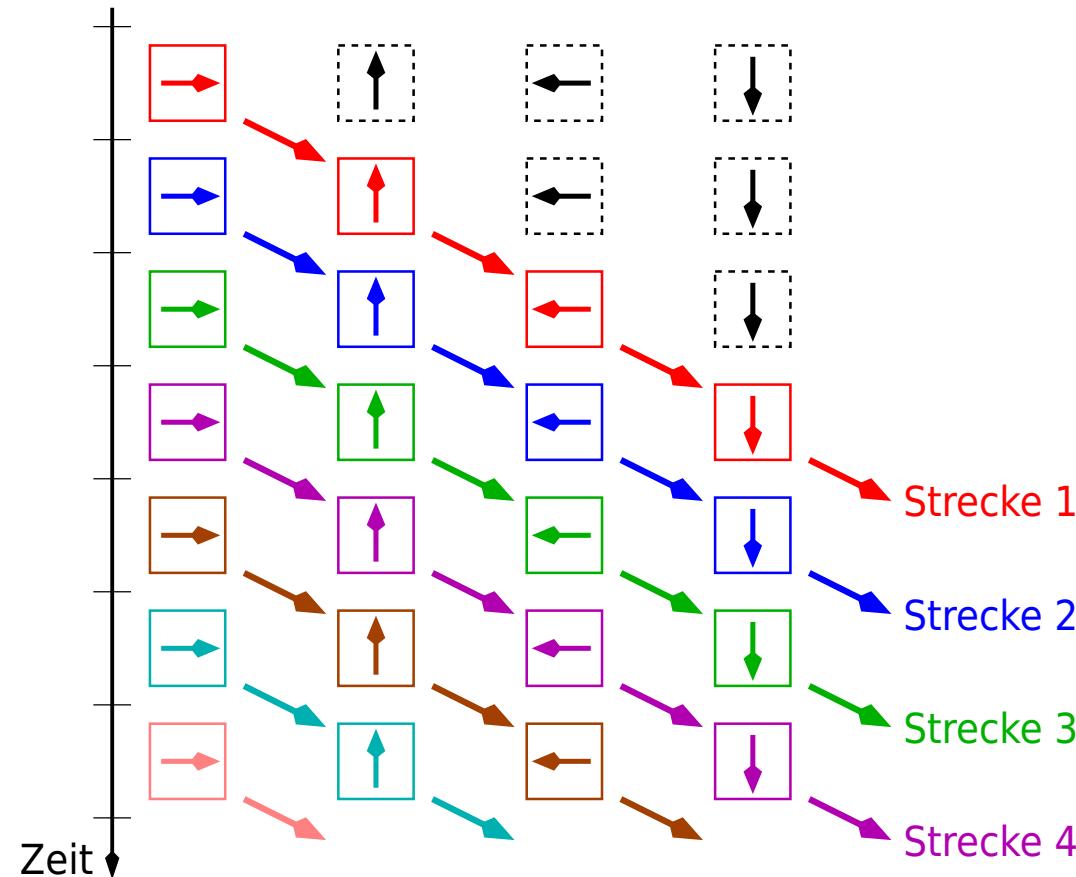
5.2.5 Hardware-Realisierung

Ziel: möglichst hoher „Durchsatz“ (Anzahl geclippster Strecken pro Sekunde)

„konventionelle Lösung“:

Durchsatz: 1 Strecke pro 4 Zeittakte

„Pipeline-Lösung“:



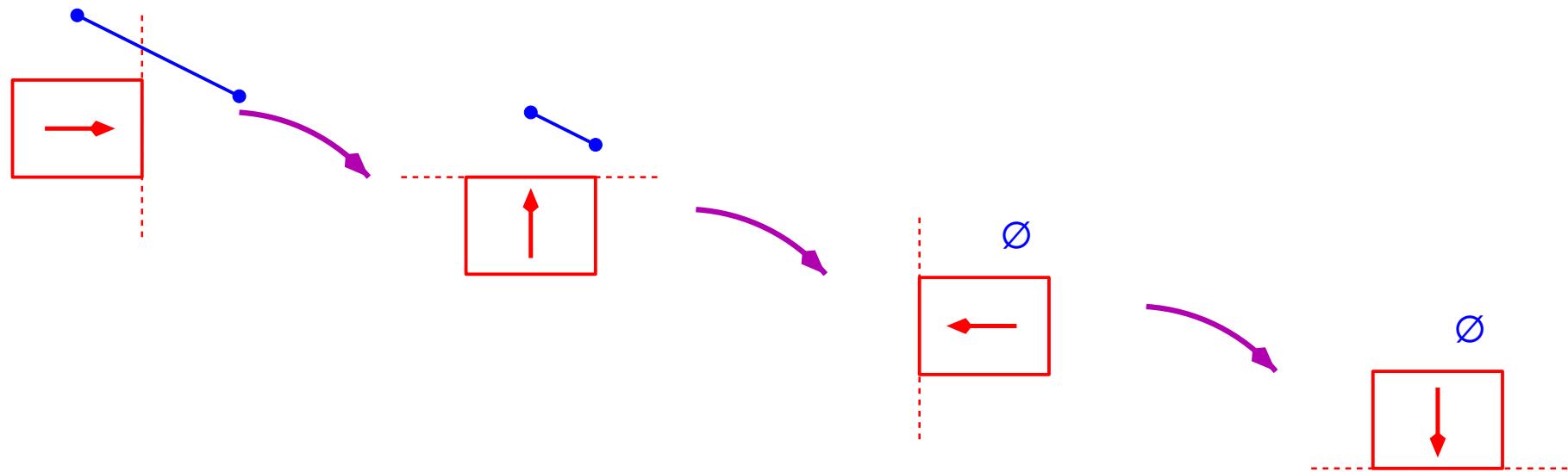
Durchsatz: n Strecken in $n + 3$ Zeittakten

$$\hat{=} 1 \text{ Strecke in } 1 + \frac{3}{n} \text{ Zeittakten}$$

(rund 4-mal schneller für große n !)

Bemerkungen 5.11:

1. Wie bei der konventionellen Lösung liefern die ersten drei Zeittakte noch kein Endergebnis („Startup-Zeit“).
2. Die einzelnen Stufen der Pipeline sind noch einfacher (\Rightarrow billiger, schneller) als der „4-Richtungs“-Baustein in der konventionellen Lösung.
3. Es ist nicht mehr möglich, das Clipping vorzeitig abzubrechen, wenn das Ergebnis schon feststeht:

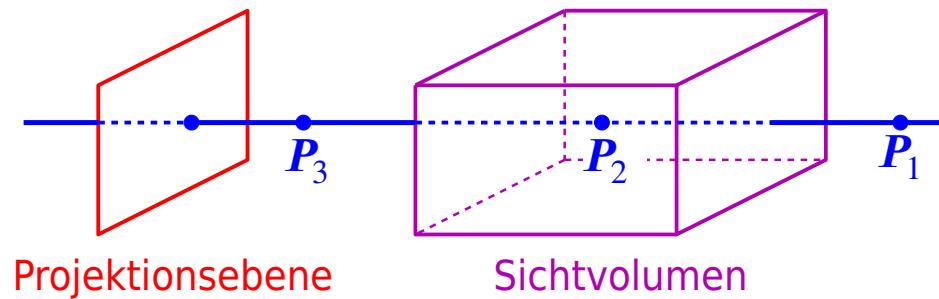


5.3 Der 3D-analytische Ansatz

Ziel: Entferne alle Teile, die nicht in einem vorgegebenen **Sichtvolumen** liegen.

Bemerkung 5.12: Bei der Projektion verliert man die räumliche Tiefeninformation. Clipping gegenüber einem Volumen kann also **nicht** mit den projizierten Punkten erfolgen.

⇒ 3D-Clipping meist analytisch (nicht auf Pixelebene)



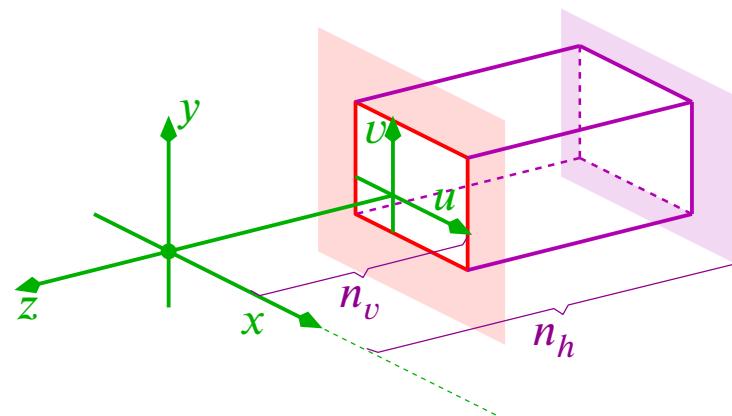
P_1 , P_2 und P_3 werden auf denselben Punkt in der Projektionsebene abgebildet, aber nur P_2 ist sichtbar.

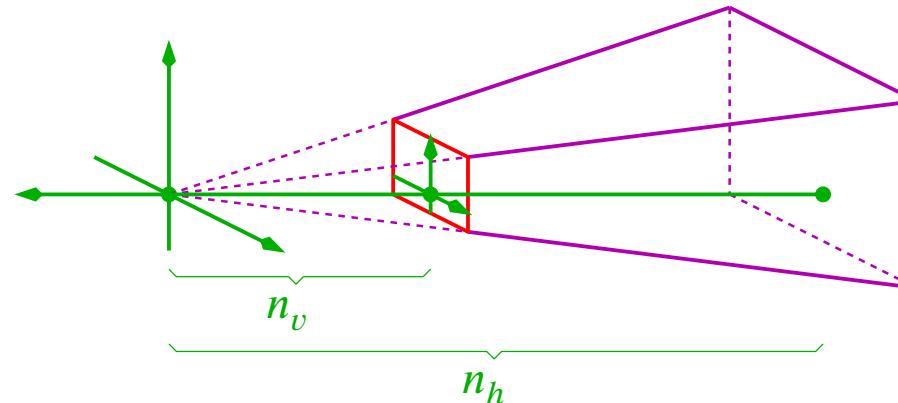
Definition des Sichtvolumens durch

- Angabe des **Projektionsfensters**
(und der Lage von Projektionsebene und Projektionsrichtung bzw. -zentrum)
- Abstand der **vorderen** und **hinteren Begrenzungsebene** vom Ursprung: n_v bzw. n_h

In OpenGL ist die Projektionsebene die vordere Begrenzungsebene, d. h. $n_v = -z_p$.

Parallelprojektion:



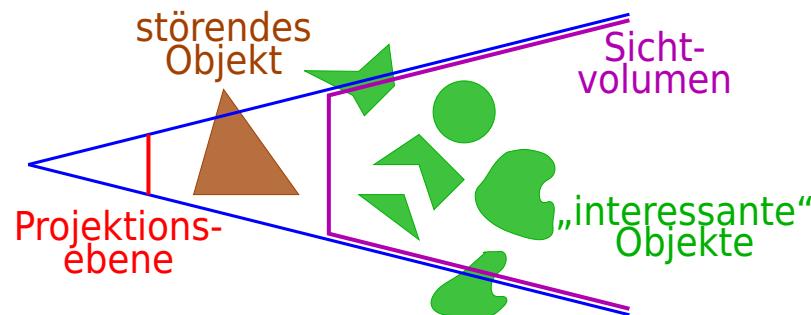
perspektivische Projektion:

Bemerkung 5.13: n_v und n_h werden senkrecht zur Projektionsebene gemessen.

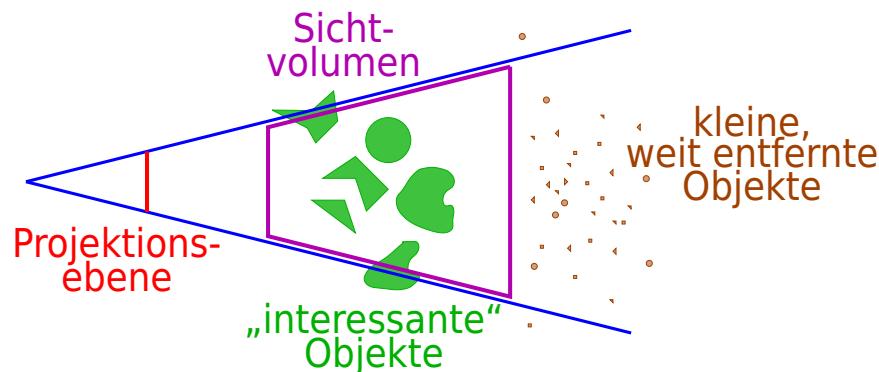
Damit das Sichtvolumen nicht leer ist, muss $n_v < n_h$ sein.

Anwendungen für 3D-Clipping

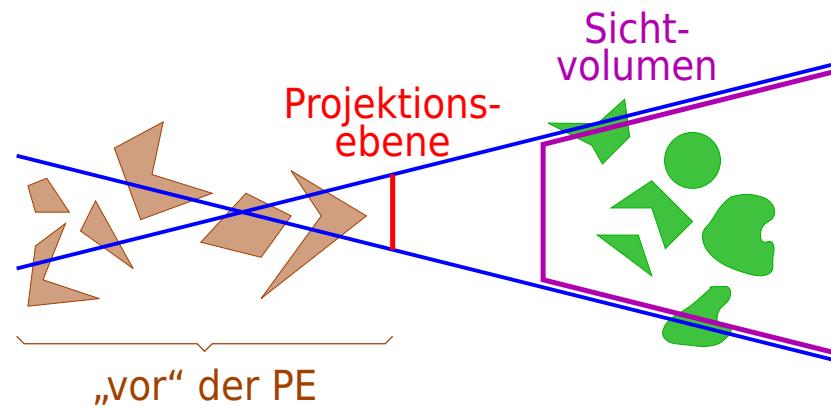
- Ein Objekt nahe der Projektionsebene verdeckt viele andere, die eigentlich interessanter sind.
⇒ störendes Objekt „ausblenden“



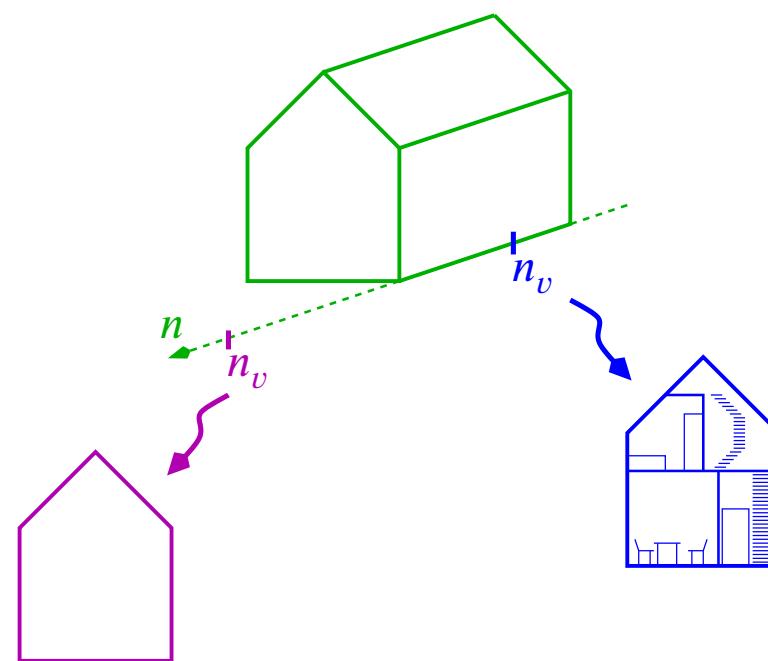
- Sehr weit entfernte (und nicht allzu große) Objekte werden ohnehin zu nicht identifizierbaren kleinen Flecken projiziert.
⇒ ausblenden (Projektion und Scan Conversion entfällt!)



- Gegenstände „vor“ der Projektionsebene sollen nicht gezeigt werden.



- „schichtweise Betrachtung“



Techniken für 3D-Clipping: Übertrage die 2D-analytischen Ansätze (soweit möglich) auf 3D:

- 4 Fenstergeraden \rightsquigarrow 6 Ebenen
- 4-stelliger Code bei Cohen/Sutherland \rightsquigarrow 6-stellig
- Schnittpunkte mit den Ebenen am einfachsten aus der Normalenform

Oft werden die Endpunkte der Strecken auch vorher in **normalisierte Sichtkoordinaten** umgerechnet; bezüglich dieser Koordinaten sind Sichtvolumen und Projektion besonders einfach zu beschreiben:

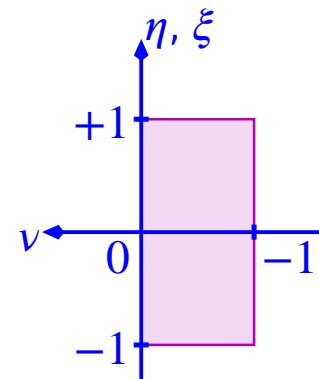
Parallelprojektion:

Clipping-Ebenen (z. B.):

$$\xi = \pm 1, \eta = \pm 1 \quad (\triangleq \text{Fenster})$$

$$\nu = 0 \quad (\text{vordere Begrenzung})$$

$$\nu = -1 \quad (\text{hintere Begrenzung})$$



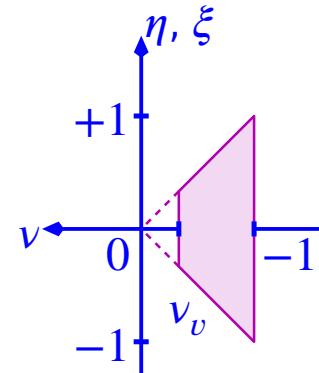
perspektivische Projektion:

Clipping-Ebenen (z. B.):

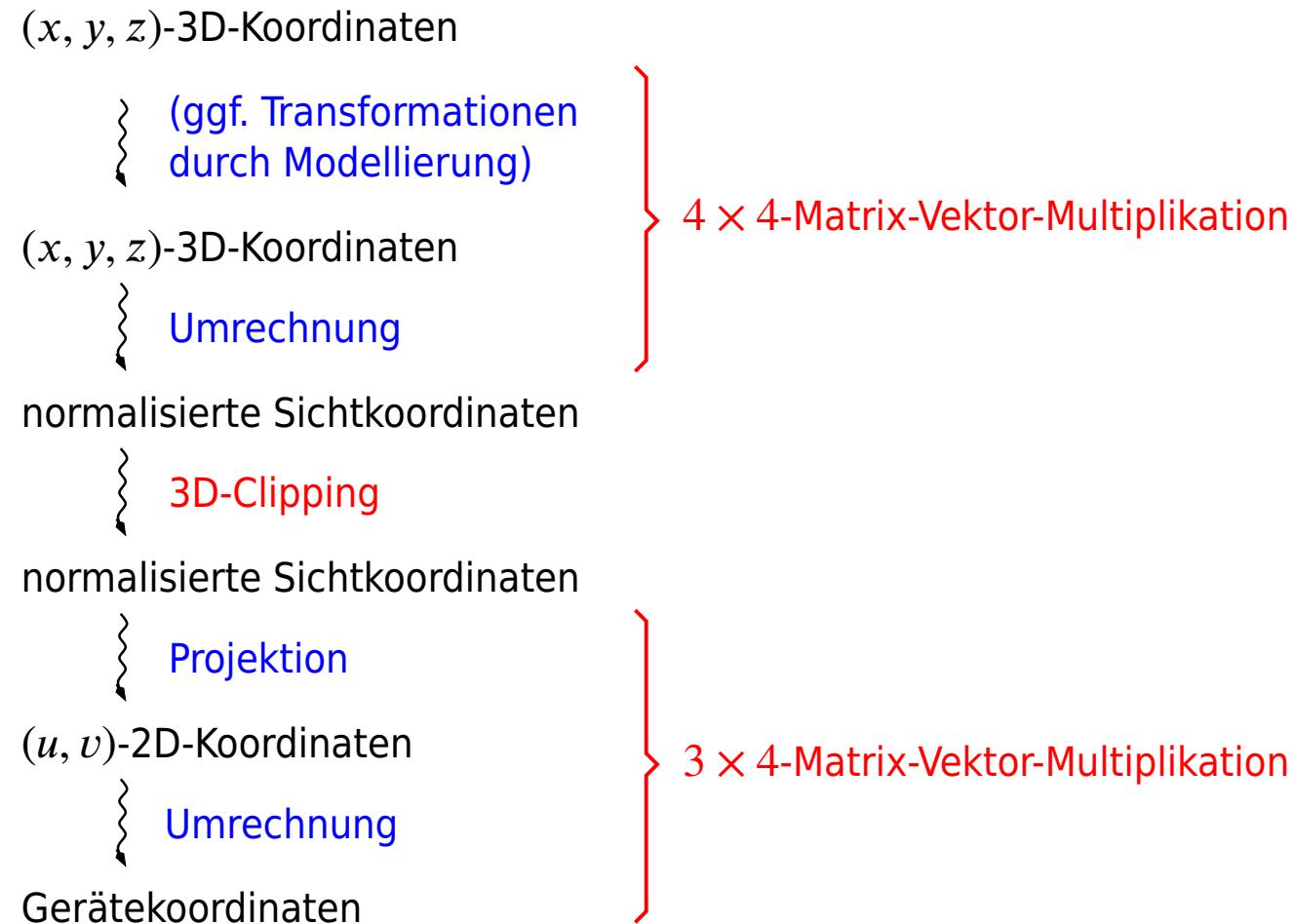
$$\xi = \pm \nu, \eta = \pm \nu \quad (\triangleq \text{Fenster})$$

$$\nu = \nu_v \quad (\text{vordere Begrenzung})$$

$$\nu = -1 \quad (\text{hintere Begrenzung})$$



Mögliche Realisierung im Rahmen der (3D → 2D)-Transformation

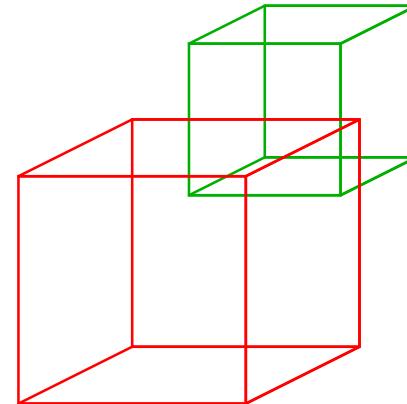


6 Sichtbarkeit

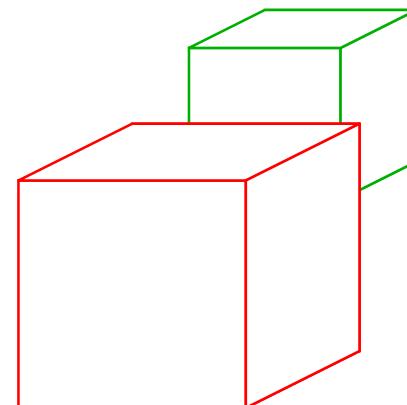
Inhalt

6.1 Der Pixel-orientierte Ansatz	6-3
6.1.1 Naive Implementierung	6-4
6.1.2 Optimierung I: Zell-Raster-Technik	6-13
6.1.3 Optimierung II: „z-Puffer-Algorithmus“	6-20
6.2 Der analytische Ansatz	6-25
6.2.1 Hidden Line Removal für Polygonszenen	6-25
6.2.2 Optimierungen für Polyederszenen	6-32
6.3 Vereinfachte Algorithmen	6-37
6.3.1 Painter’s Algorithm	6-37
6.3.2 Der Silhouetten-Algorithmus	6-42

bisher: „Drahtmodell“-Darstellung (**wire frame**) aller Kanten aller Objekte

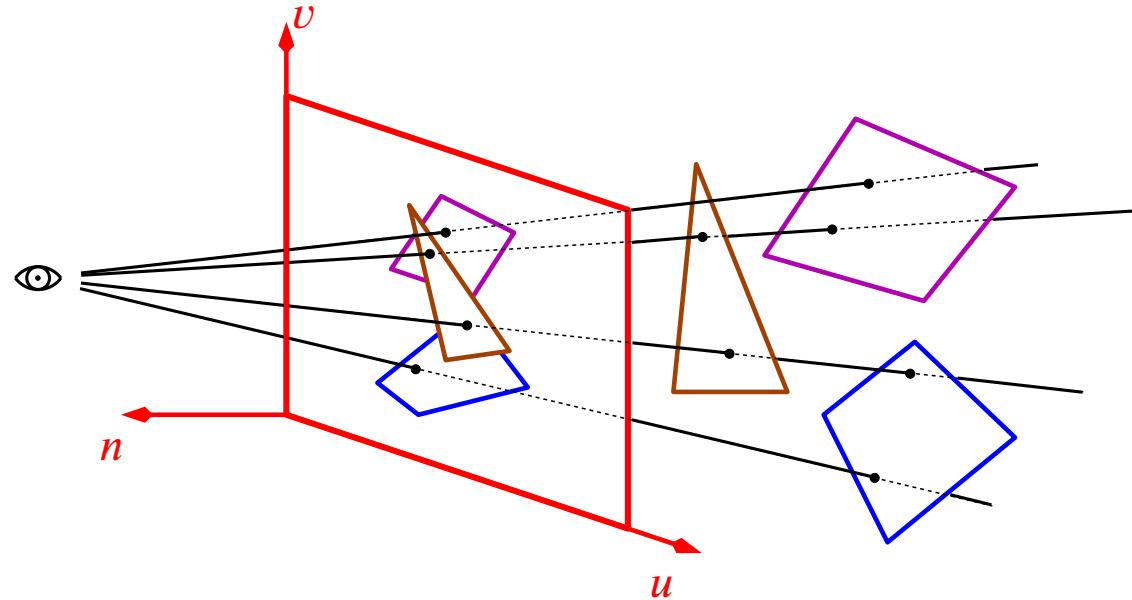


Ziel: Elimination aller Strecken- und Flächenteile, die durch „weiter vorne liegende“ Flächen **verdeckt** werden
(**hidden line / hidden surface removal**)



⇒ wesentlich **plastischere Darstellung**

6.1 Der Pixel-orientierte Ansatz



Beobachtung: Pixel (u, v) erhält die Farbe des Objekts, das

- den durch das Pixel gehenden Projektionsstrahl schneidet

und

- unter allen Objekten den am nächsten bei der Projektionsebene liegenden Schnittpunkt liefert.

Folgerung: Gesucht ist der ***n*-maximale** Schnittpunkt des Projektionsstrahls mit einem Objekt.

6.1.1 Naive Implementierung

Ansatz: Bestimme für jedes Pixel derPixmap, welche Objekte der zugehörige Projektionsstrahl schneidet, und leite aus dem nächstgelegenen Schnittpunkt die Farbe des Pixels ab.

Algorithmus 6.1:

Algorithmus visible surface ray tracing

für alle Pixel P derPixmap

stelle den Projektionsstrahl durch das Pixel in der Form $q = q_1 + \lambda \cdot (q_2 - q_1)$ dar,
so dass Punkte mit größerem λ „weiter vorne“ liegen

$O^* := \text{„Hintergrund“}$ // bislang nächstgelegenes Objekt

$\lambda^* := -\infty$ // Parameterwert des zugehörigen Schnittpunkts

für alle Objekte O_i

Schnitttest: Prüfe, ob der Projektionsstrahl das Objekt O_i trifft, und bestimme ggf.
den Parameterwert λ_i des Schnittpunkts

wenn Schnittpunkt vorhanden und $\lambda_i > \lambda^*$

$O^* := O_i$

$\lambda^* := \lambda_i$

setze Pixel P auf die Farbe von Objekt O^*

Aufwand:

problemrelevante Größen

- p Anzahl der Pixel in derPixmap
- o Anzahl der Objekte
- $\mathcal{O}(s)$ Aufwand für einen Schnitttest
- $\mathcal{O}(z)$ Aufwand zum Zeichnen eines Pixels

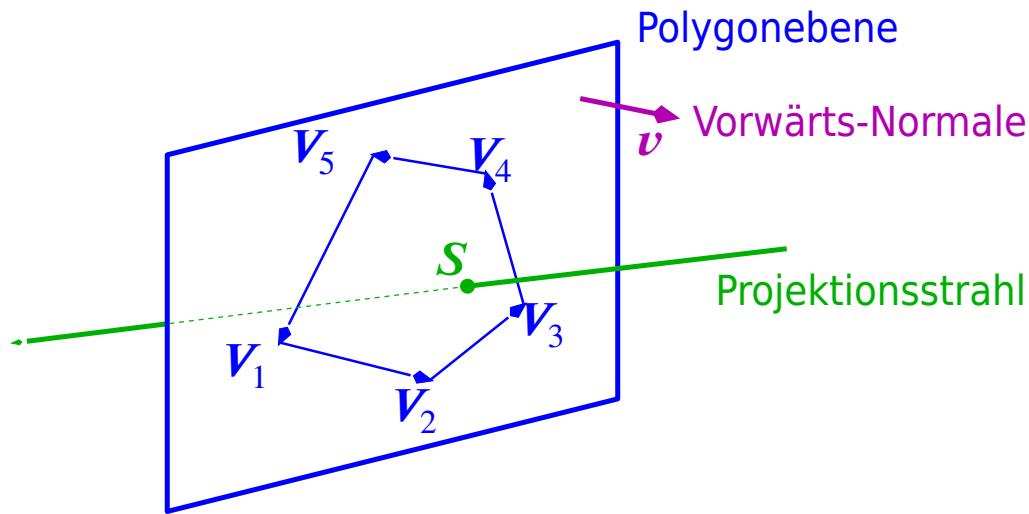
für alle Pixel	p -mal	
Projektionsstrahl darstellen	$\mathcal{O}(1)$	
O^* , λ^* initialisieren	$\mathcal{O}(1)$	
für alle Objekte	o -mal	
Schnitttest	$\mathcal{O}(s)$	
O^* , λ^* aktualisieren	$\mathcal{O}(1)$	
Pixel zeichnen	$\mathcal{O}(z)$	
Σ	$\mathcal{O}(p \cdot o \cdot s + p \cdot z)$	

Bemerkungen 6.2:

1. Zeichnen eines Pixels ist i. Allg. in $\mathcal{O}(z) = \mathcal{O}(1)$ möglich.
2. Der Aufwand $\mathcal{O}(s)$ hängt davon ab, welche Objekte zugelassen werden und ob **Vorverarbeitung der Objekte** erfolgt.

Beispiel 6.3: Die darzustellenden Objekte sind **konvexe Polygone mit höchstens m Ecken**.

Jedes Polygon werde durch seine $m' \leq m$ Ecken und einen „Vorwärts“-Normalenvektor ν zur Polygonebene gegeben.

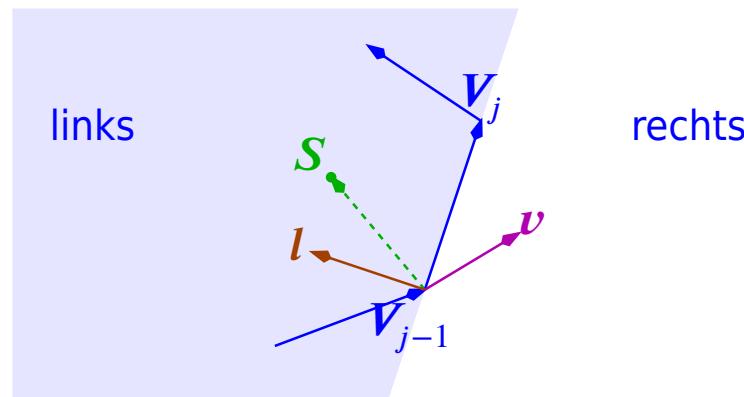


Die Ecken seien **entgegen dem Uhrzeigersinn** nummeriert, wenn die Polygonebene „von vorne“ (d. h. in Richtung $-\nu$) betrachtet wird.

1. Schnitttest ohne Vorverarbeitung:

- Bestimme den Schnittpunkt S des Strahls mit der Polygonebene.
- S liegt genau dann im (konvexen!) Polygon, wenn S **links jeder gerichteten Seite** $\overrightarrow{V_{j-1}V_j}$ liegt ($j = 1, \dots, m'$, mit $V_0 := V_{m'}$).

Wie prüft man, ob S links oder rechts von $\overrightarrow{V_{j-1}V_j}$ liegt?



$l := v \times (V_j - V_{j-1})$ ist ein „nach links“ zeigender Vektor.

⇒ S liegt genau dann links von $\overrightarrow{V_{j-1}V_j}$, wenn

$$l^T \cdot (S - V_{j-1}) > 0 .$$

Algorithmus Schnitttest ohne Vorverarbeitung

```

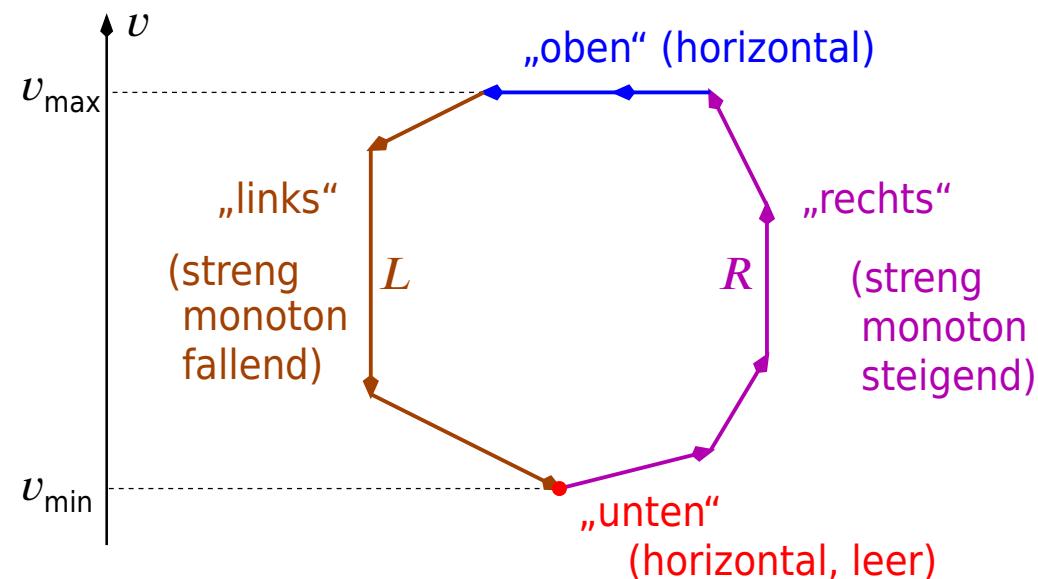
innen := true
für  $j = 1, \dots, m'$ 
    innen := innen and  $(v \times (V_j - V_{j-1}))^T \cdot (S - V_{j-1}) > 0$ 

```

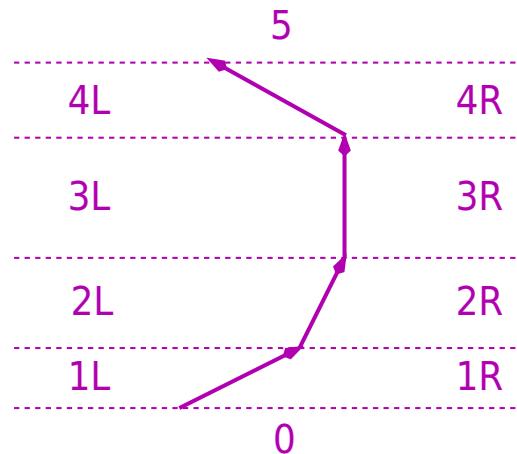
Aufwand: $\mathcal{O}(s) = \mathcal{O}(m') = \mathcal{O}(m)$

2. Schnitttest mit Vorverarbeitung:

- Ein konvexes Polygon zerfällt in zwei (evtl. leere) horizontale und zwei bzgl. der v -Achse **streng monotone Polygonzüge** L bzw. R :



- S liegt genau dann im Polygon, wenn
 - $v_{\min} < S_v < v_{\max}$ und
 - S links der beiden monotonen Polygonzüge liegt.
- Ein monotoner Polygonzug mit k Ecken zerlegt die Ebene in $2k$ Gebiete:



Durch **binäre Suche** kann mit $\mathcal{O}(\log k)$ Vergleichen der horizontale Streifen bestimmt werden, in dem S liegt. Danach genügt **1 „Links/Rechts-Test“** um zu entscheiden, ob S links oder rechts des monotonen Polygonzugs liegt.

Algorithmus Vorverarbeitung

für alle konvexen Polygone O_i

führe ein geeignetes (u, v, w) -Koordinatensystem ein, so dass $w \equiv 0$ die Polygonebene ist
rechne die Ecken V_i in (u, v, w) -Koordinaten um

bestimme minimale und maximale v -Koordinaten v_{\min} bzw. v_{\max}
speichere die beiden monotonen Polygonzüge in Datenstrukturen, die für binäre Suche
geeignet sind (z. B. Felder)

Aufwand:

für alle Polygone	o -mal	
(u, v, w) -System berechnen	$\mathcal{O}(1)$	
V_i umrechnen	$\mathcal{O}(m)$	
v_{\min}, v_{\max} bestimmen	$\mathcal{O}(m)$	
Polygonzüge extrahieren	$\mathcal{O}(m)$	
\sum	$\mathcal{O}(o \cdot m)$	

Algorithmus Schnitttest

berechne den Schnittpunkt $S = (S_x, S_y, S_z)$ des Projektionsstrahls mit der Polygonebene

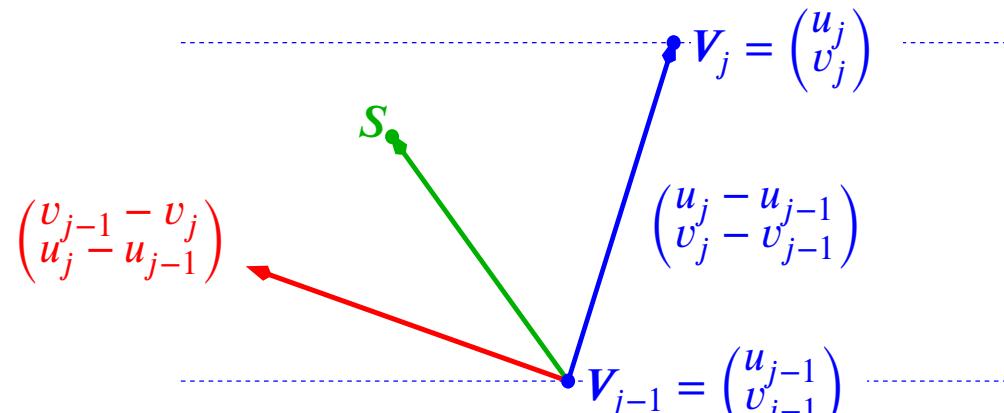
rechne S in (u, v, w) -Koordinaten um

wenn $v_{\min} < S_v < v_{\max}$

// teste „ S links von R ?“

bestimme mit binärer Suche denjenigen durch R definierten Streifen $[v_{j-1}; v_j]$,
der S enthält

$$\text{innen} := \begin{pmatrix} v_{j-1} - v_j \\ u_j - u_{j-1} \end{pmatrix}^T \cdot \begin{pmatrix} S_u - u_{j-1} \\ S_v - v_{j-1} \end{pmatrix} > 0$$



wenn innen
teste „ S links von L ?“ // analog

Aufwand:

S berechnen	$\mathcal{O}(1)$
S in (u, v, w) umrechnen	$\mathcal{O}(1)$
für R und L	2-mal
binäre Suche	$\mathcal{O}(\log m)$
innen testen	$\mathcal{O}(1)$
\sum	$\mathcal{O}(\log m)$

Fazit: Der Gesamtaufwand beträgt

$$\begin{aligned} & \mathcal{O}(p \cdot o \cdot \textcolor{red}{m} + p \cdot z) && \text{ohne Vorverarbeitung} \\ & \text{und} \\ & \mathcal{O}(\textcolor{red}{o} \cdot \textcolor{red}{m} + p \cdot o \cdot \log m + p \cdot z) && \text{mit Vorverarbeitung.} \end{aligned}$$

Bemerkung 6.4: Typische Größenordnungen:

$$p = 1920 \times 1080 \approx 2 \cdot 10^6$$

$$o \gg 1000$$

⇒ >> **10⁹ Schnitttests** notwendig!

Visible surface ray tracing ist

- ⊖ sehr aufwändig
 - ⊕ relativ leicht zu sehr anspruchsvollen Bilderzeugungsverfahren erweiterbar.
⇒ rekursives Raytracing (Abschnitt 10.1)
-

6.1.2 Optimierung I: Zell-Raster-Technik

Idee: In Algorithmus 6.1 wird jeder Projektionsstrahl mit jedem Objekt geschnitten.

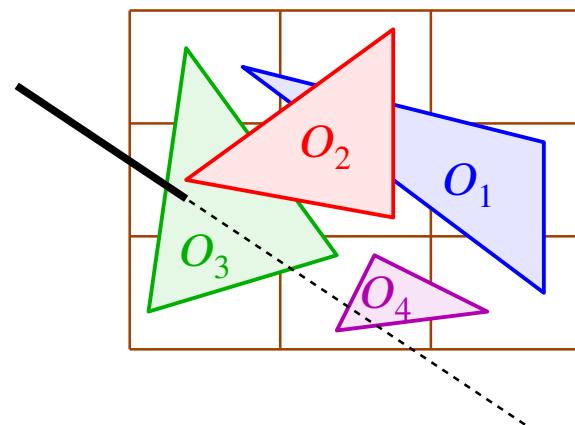
Die Objekte sind aber meist verhältnismäßig klein, überdecken also nur einen Teil des Bildschirms.

- ⇒ Versuche, in einer **Vorbereitungsphase** Information darüber zu sammeln, welche Objekte überhaupt mit welchen Projektionsstrahlen geschnitten werden müssen.

Es ist aufwändig (Rechenzeit, Speicherplatz), für jeden einzelnen Strahl die geschnittenen Objekte zu bestimmen.

⇒ nur **summarische Analyse**:

- Unterteile diePixmap in $r = r_1 \times r_2$ „Zellen“.
- Bestimme für jede Zelle alle Objekte, welche (zumindest teilweise) in die Zelle fallen.



Z_{11} :	O_1, O_2, O_3
Z_{12} :	O_1, O_2
Z_{13} :	O_1
Z_{21} :	O_2, O_3
Z_{22} :	O_1, O_2, O_3
Z_{23} :	O_1
Z_{31} :	O_3
Z_{32} :	O_3, O_4
Z_{33} :	O_1, O_4

- Jeder Projektionsstrahl geht durch eine Zelle. Führe nur Schnitttests mit den Objekten durch, welche die Zelle treffen.

(Im obigen Beispiel geht der Strahl durch Z_{21} , also nur Schnitttests mit O_2 und O_3 .)

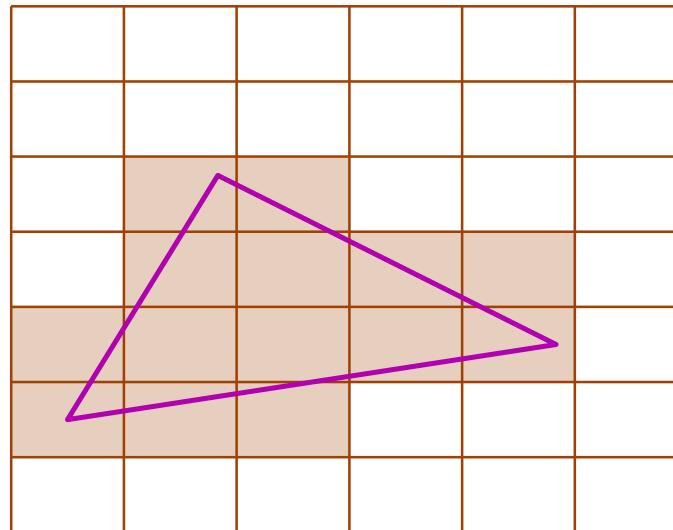
Frage: Wie bestimmt man diese Objekte möglichst effizient?

Antwort: Man geht von den Objekten aus und bestimmt zu jedem Objekt die von ihm berührten Zellen!

Methode I: „Scan Conversion“

Das Zellraster kann als gröberePixmap mit $r_1 \times r_2$ Pixeln interpretiert werden.

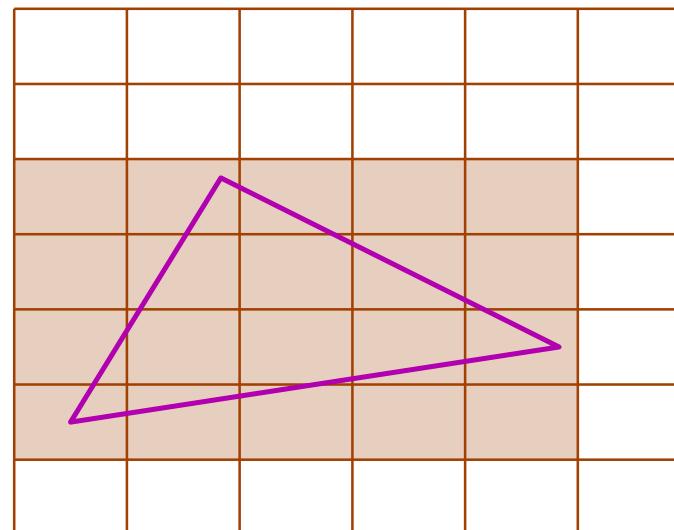
⇒ Die zu einem Objekt gehörenden Zellen können durch Scan Conversion in das Zellraster ermittelt werden.



- ⊕ liefert bestmögliche Aussage, d. h. genau die berührten Zellen
- ⊖ verhältnismäßig aufwändig

Methode II: „Bounding Box“

- Projizierte alle Ecken des Objekts in das Zellraster.
- Bestimme das kleinstmögliche achsenparallele Rechteck im Zellraster, das alle diese Ecken enthält („Bounding Box“).



- Nimm das Objekt zu allen Zellen dieses Rechtecks hinzu.
- ⊖ fügt das Objekt i. Allg. in mehr Zellen ein als unbedingt nötig
- ⊕ sehr einfach zu realisieren, auch für komplexe Objekte

Algorithmus 6.5:

Algorithmus modifizierter Darstellungsalgorithmus

für alle Zellen Z_{ij} des Rasters

für alle Pixel P von Zelle Z_{ij} // insgesamt p -mal:

Projektionsstrahl parametrisch darstellen

$O^* := \text{„Hintergrund“}$

$\lambda^* := -\infty$

für alle Objekte O_k in der Z_{ij} -Liste // ?-mal:

Schnitttest O_k mit dem Projektionsstrahl

O^* und λ^* aktualisieren

Pixel P auf die O^* entsprechende Farbe setzen

// zusammen $\mathcal{O}(1)$

// $\mathcal{O}(1)$

// $\mathcal{O}(s)$

// $\mathcal{O}(1)$

// $\mathcal{O}(z)$

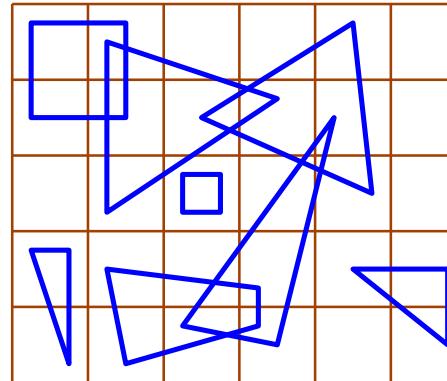
Wie oft die innerste Schleife „**für alle Objekte O_k in der Z_{ij} -Liste**“ **insgesamt** durchlaufen wird, hängt davon ab,

- wie viele Zellen vorhanden sind und
- wie viele Objekte jede Zellenliste (im Mittel) enthält.

Heuristik: Im Mittel überdeckt jedes Objekt $\frac{1}{r}$ der Pixmap-Fläche.

Wähle dann etwa r Zellen.

„gutartige“ Szenen: Jedes Objekt trifft im Mittel $\mathcal{O}(1)$ Zellen.



⇒ Jedes Objekt wird bei $\mathcal{O}\left(\frac{p}{r}\right)$ Pixeln betrachtet.

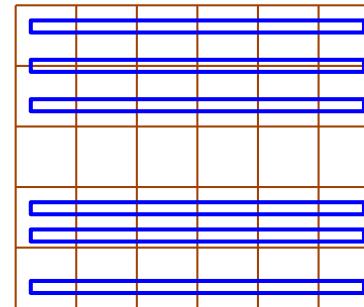
⇒ insgesamt $\mathcal{O}\left(\frac{p \cdot o}{r}\right)$ Schnitttests

Gesamtaufwand:

$$\mathcal{O}\left(\underbrace{o \cdot m}_{\text{Vorverarb.}} + \frac{p \cdot o}{r} \cdot s + p \cdot z\right) \quad \begin{array}{l} \text{Operationen und} \\ \text{zusätzlicher Speicher} \end{array}$$

$$\mathcal{O}(o)$$

„bössartige“ Szenen: Jedes Objekt trifft im Mittel $\mathcal{O}(\sqrt{r})$ Zellen.



⇒ Jedes Objekt wird bei $\mathcal{O}\left(\frac{p}{r} \cdot \sqrt{r}\right) = \mathcal{O}\left(\frac{p}{\sqrt{r}}\right)$ Pixeln betrachtet.

⇒ insgesamt $\mathcal{O}\left(\frac{p \cdot o}{\sqrt{r}}\right)$ Schnitttests

Gesamtaufwand:

$$\mathcal{O}\left(o \cdot \underbrace{(\sqrt{r} + m)}_{\text{Vorverarb.}} + \frac{p \cdot o}{\sqrt{r}} \cdot s + p \cdot z\right)$$

Operationen und
zusätzlicher Speicher

Bemerkung 6.6: Bösartige Szenen mit $o \gg 10^4$ kommen beispielsweise beim Chip-Design vor (lange und schmale Leiterbahnen).

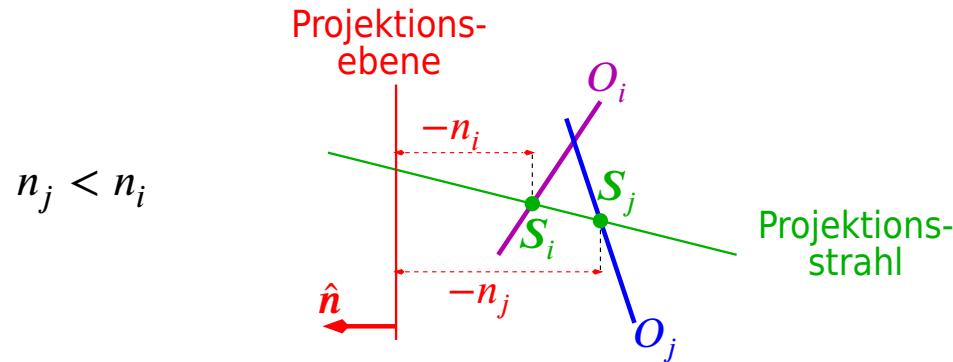
6.1.3 Optimierung II: „z-Puffer-Algorithmus“

Idee: Verzichte darauf, nur die **im endgültigen Bild** sichtbaren Objektteile zu zeichnen.

Zeichne die Objekte der Reihe nach; führe dabei die **Sichtbarkeitsprüfung nur gegenüber den bereits gezeichneten Objekten** durch.

Pixel P wird mit der Objekt O_i entsprechenden Farbe belegt, falls es

- bisher noch zu keinem Objekt O_j ($j < i$) gehörte oder
- bisher die Farbe von O_j ($j < i$) hatte und der zugehörige Schnittpunkt S_j „hinter“ dem Schnittpunkt S_i des Projektionsstrahls mit O_i liegt, d. h. bezüglich des (u, v, n) -„Projektionsebenen-Koordinatensystems“ gilt:



- ⇒ Für jedes Pixel muss die n -Koordinate des dort gerade sichtbaren (Schnit-)Punkts gespeichert werden.
- ⇒ weiteres Feld der gleichen Dimension wie die Pixmap zur Speicherung der n -Koordinaten zu jedem Pixel (oft auch „ z -Koordinate“ ⇒ „**z-Puffer**“)

Algorithmus 6.7:**Algorithmus z -Puffer-Algorithmus**

```
für alle Pixel  $P$  der Pixmap
     $z[P] := -\infty$  // Hintergrund
    setze Pixel  $P$  auf Hintergrundfarbe
für alle Objekte  $O_i$ 
    führe 3D Scan Conversion für Objekt  $O_i$  durch, d. h. bestimme für jedes erzeugte Pixel  $P$ 
        auch die zugehörige  $z$ - bzw.  $n$ -Koordinate  $z_P$  des entsprechenden Punktes auf  $O_i$ 
für alle hierbei erzeugten Pixel  $P$ 
    wenn  $z_P > z[P]$  // ist  $O_i$  bei Pixel  $P$  sichtbar?
         $z[P] := z_P$ 
        setze Pixel  $P$  auf die Farbe von Objekt  $O_i$ 
```

Aufwand:

„Hintergrund“-Initialisierung

$$\mathcal{O}(p \cdot z)$$

für alle Objekte O_i

o -mal

3D Scan Conversion und sichtbare Pixel zeichnen

$$\mathcal{O}(p_i \cdot z) + \mathcal{O}(h)$$

mit

p_i = Anzahl der von Objekt O_i überdeckten Pixel

$\mathcal{O}(h)$ = Verwaltungsaufwand für Scan Conversion

Beispiel 6.8: Die Objekte seien konvexe Polygone mit maximal m Ecken, und im Mittel überdecke jedes Objekt $\frac{1}{r}$ der Pixmap-Fläche.

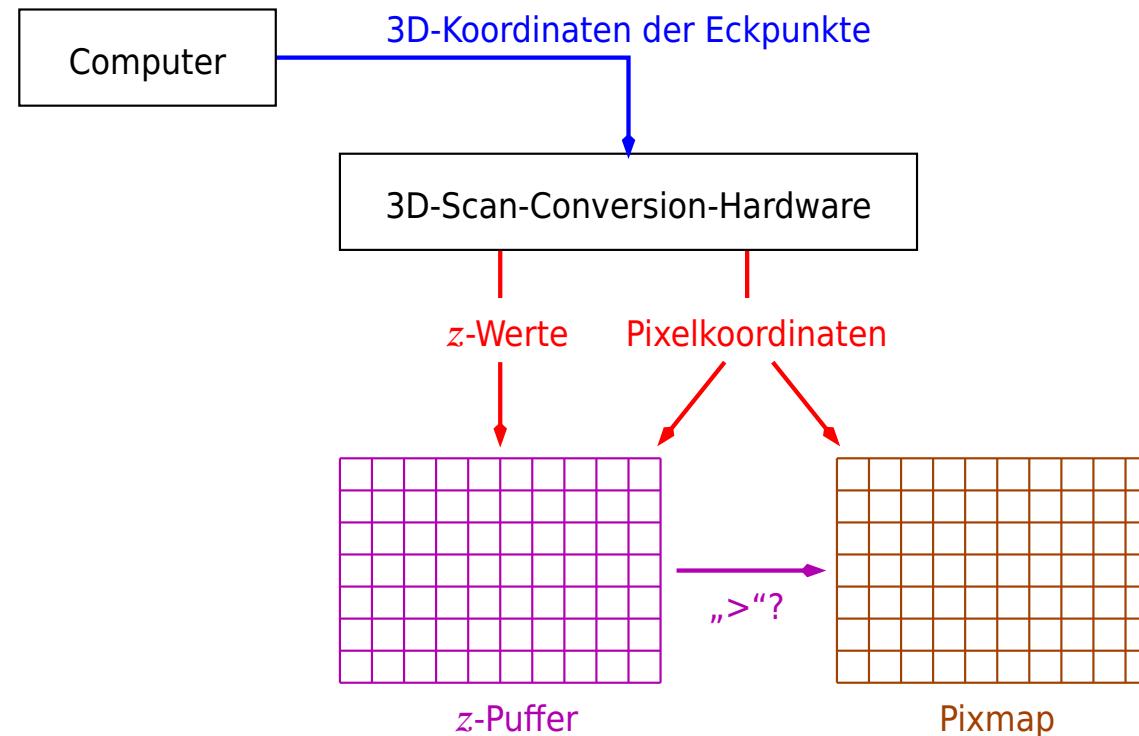
$$\Rightarrow \sum_{O_i} p_i = o \cdot \frac{p}{r},$$

$$\mathcal{O}(h) = \mathcal{O}(m)$$

$$\Rightarrow \text{Gesamtaufwand: } \mathcal{O}\left(p \cdot z + o \cdot \frac{p}{r} \cdot z + o \cdot m\right)$$

- ⊕ ziemlich einfach und effizient, da auch 3D Scan Conversion **inkrementell** möglich ist. Bei **Parallelprojektion** gilt z. B.:
 - Innerhalb einer Scan Line ändert sich z_P von einem Pixel zum nächsten stets um einen festen Betrag $\Delta_1 z$,
 - von einem Pixel zum darüber liegenden der nächsten Scan Line ist die Änderung ebenfalls konstant: $\Delta_2 z$.
- ⊕ praktisch keine reelle Rechnung (keine Schnitttests!)

- ⊕ in der Regel Hardware-unterstützt:



Vorteile:

- sehr bequem (kein Programm erforderlich!)
- sehr schnell, und nicht allzu teuer

6.2 Der analytische Ansatz

Ziel: Bestimme rechnerisch die sichtbaren Teile der Objekte.

6.2.1 Hidden Line Removal für Polygonszenen

Ansatz: Bestimme für jede Strecke s der Szene, welche Stücke von s durch andere Polygone verdeckt werden.

Algorithmus 6.9:

Algorithmus Hidden Line Removal für Polygonszenen

für jede Strecke s der Szene

verdeckt := \emptyset // die durch andere Polygone verdeckten Teile von s

für alle Polygone, die s nicht als Seite enthalten

bestimme den hinter der Polygonebene liegenden Teil s' von s

wenn $s' \neq \emptyset$

verdeckt := verdeckt \cup diejenigen Teile von s' , die durch das Polygon verdeckt werden

sichtbar := $s \setminus$ verdeckt

Teilprobleme

1. Welcher Teil der Strecke s liegt hinter der Polygonebene?

Sei \mathbf{P} ein fester Punkt in der Polygonebene (z. B. eine Ecke des Polygons) und \mathbf{v} eine „Vorwärts“-Normale zur Polygonebene (in Richtung Projektionsrichtung bzw. -zentrum).

Berechne für die beiden Endpunkte \mathbf{E}_i von s die Zahlen

$$\sigma_i := (\mathbf{E}_i - \mathbf{P})^T \cdot \mathbf{v}.$$

Dann ist der hinter der Polygonebene liegende Teil von s bzgl. der Parametrisierung

$$\mathbf{q} = \mathbf{E}_1 + \mu (\mathbf{E}_2 - \mathbf{E}_1), \quad \mu \in [0; 1],$$

von s gegeben durch:

	$\sigma_1 \geq 0$	$\sigma_1 < 0$
$\sigma_2 \geq 0$	$s' = \emptyset$	$s' = \left[0; -\frac{\sigma_1}{\sigma_2 - \sigma_1}\right]$
$\sigma_2 < 0$	$s' = \left[-\frac{\sigma_1}{\sigma_2 - \sigma_1}; 1\right]$	$s' = [0; 1]$

2. Welche Teile von s' werden durch das Polygon O_i verdeckt?

- Projizierte Strecke und Polygon in die Projektionsebene

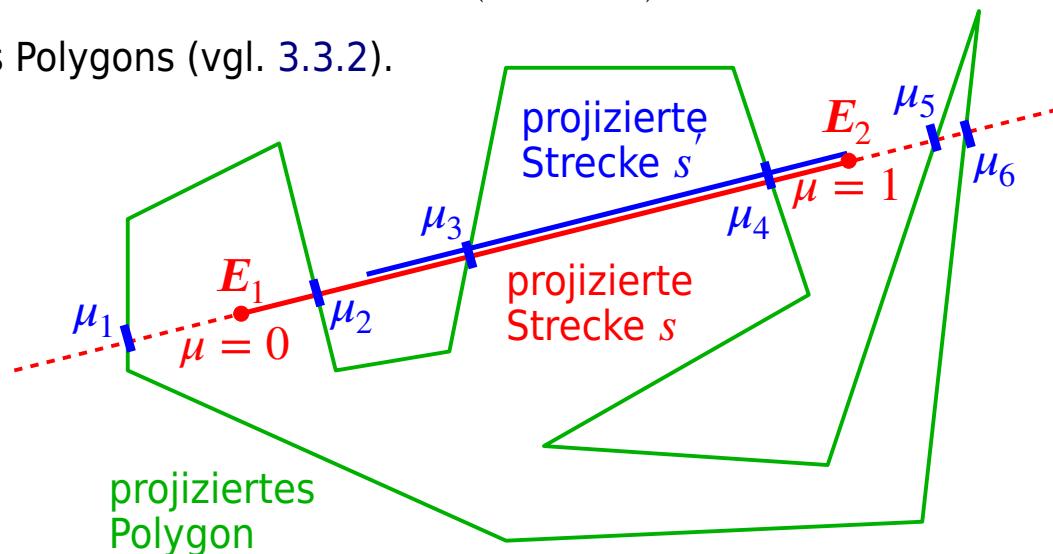
oder

projizierte die Strecke in die Polygonebene und führe dort ein 2D-Koordinatensystem ein.

- Bestimme alle Parameterwerte μ_1, \dots, μ_k von Schnittpunkten der **Geraden**

$$\mathbf{q} = \mathbf{E}_1 + \mu \cdot (\mathbf{E}_2 - \mathbf{E}_1) , \quad \mu \in \mathbb{R} ,$$

mit dem **Rand** des Polygons (vgl. 3.3.2).

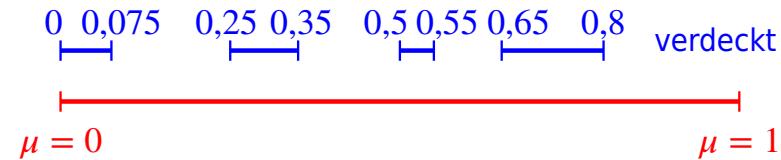


- Ordne diese Parameterwerte aufsteigend: $\mu_1 \leq \dots \leq \mu_k$
- Die verdeckten Teile ergeben sich aus den Intervallen

$$[\mu_{2j-1}; \mu_{2j}] \cap s' , \quad j = 1, \dots, \frac{k}{2} .$$

3. Wie werden die verdeckten Teile von s verwaltet?

- Die verdeckten Teile bilden eine Menge disjunkter Teilintervalle von $[0; 1]$:



- Einige häufig durchgeführte Operation mit dieser Menge: **Einfügen** eines neuen Intervalls und ggf. **Verschmelzen** mit bereits enthaltenen Intervallen

Algorithmus 6.10:**Algorithmus Einfügen und Verschmelzen**

/ fügt $I = [\underline{\mu}; \bar{\mu}]$ in die aufsteigend geordnete Folge $I_1 = [\underline{\mu}_1; \bar{\mu}_1], \dots, I_l = [\underline{\mu}_l; \bar{\mu}_l]$ ein und verschmilzt überlappende Intervalle;
zur Vereinfachung enthalte die Folge die beiden „Abschlussglieder“
 $I_0 := [-1; -1]$ und $I_{l+1} := [2; 2]$ */*

// wo liegen $\underline{\mu}$ und $\bar{\mu}$ bzgl. der Intervalle der Folge?

bestimme $i \in \{0, \dots, l\}$ **mit** $\underline{\mu}_i \leq \underline{\mu} < \underline{\mu}_{i+1}$

wenn $\bar{\mu}_i < \underline{\mu}$

$i := i + 1$ // $\underline{\mu}$ liegt „hinter“ I_i

sonst

$\underline{\mu} := \underline{\mu}_i$ // $\underline{\mu}$ liegt in I_i

bestimme $j \in \{0, \dots, l\}$ **mit** $\underline{\mu}_j \leq \bar{\mu} < \underline{\mu}_{j+1}$

wenn $\bar{\mu}_j \geq \bar{\mu}$

$\bar{\mu} := \bar{\mu}_j$ // $\bar{\mu}$ liegt in I_j

// mit den „alten“ Intervallen verschmelzen

entferne die Intervalle I_i, I_{i+1}, \dots, I_j aus der Folge

füge das Intervall $[\underline{\mu}; \bar{\mu}]$ an der Stelle i der neuen Folge ein

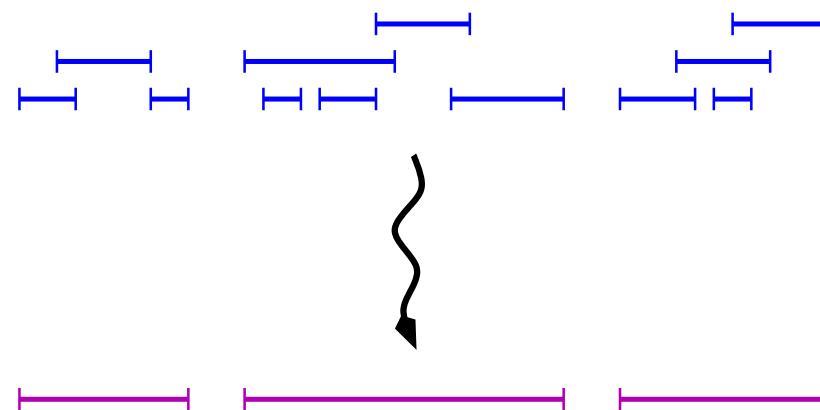
Aufwand:

Annahmen:

- Schneiden der gerade betrachteten Strecke mit allen Polygonen liefere insgesamt k Intervalle I_i .
- Für $j = 1, \dots, k$ enthalte die „bisherige Vereinigung“

$$V_j := \bigcup_{i=1}^j I_i$$

jeweils höchstens g disjunkte Intervalle.

Beispiel:

Die $j = 12$ Intervalle verschmelzen zu $g = 3$ disjunkten Intervallen.

⇒ Es muss k -mal in eine Menge von höchstens g Elementen eingefügt werden.

- Datenstruktur I: **Feld**

für jedes einzufügende Intervall

k -mal

i und j bestimmen (binäre Suche!)

$\mathcal{O}(\log g)$

einfügen und verschmelzen (d. h. höchstens g Folgenelemente

nach vorne oder hinten verschieben)

$\mathcal{O}(g)$

\sum

$\mathcal{O}(k \cdot g)$

- Datenstruktur II: **höhenbalancierter verketteter Baum**

Beispiel: verketteter 2-3-Baum

insgesamt k -mal einfügen

à $\mathcal{O}(\log g)$

insgesamt höchstens k -mal entfernen

à $\mathcal{O}(\log g)$

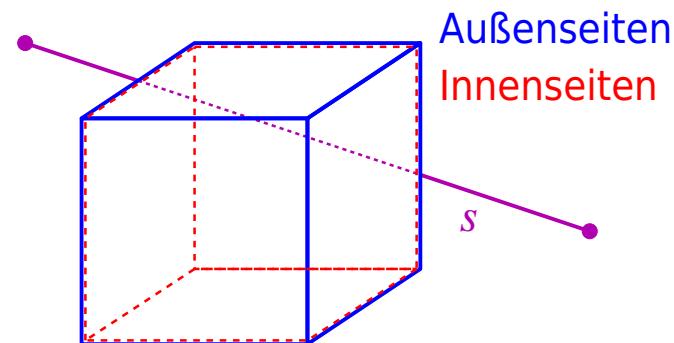
\sum

$\mathcal{O}(k \cdot \log g)$

6.2.2 Optimierungen für Polyederszenen

Annahme: Die darzustellenden Objekte sind **Polyeder**, d. h. durch Polygone begrenzte Körper.

Beobachtung 1:



Wird eine Strecke s von einem Polyeder K teilweise verdeckt, so werden dieselben Streckenteile bereits durch die „**Außenseiten**“ von K verdeckt.

⇒ Zur Bestimmung der verdeckten Teile von s müssen die Innenseiten der Körper gar nicht betrachtet werden.

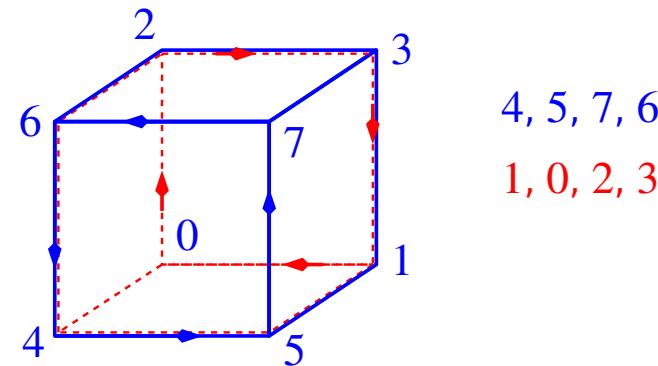
Innenseiten sind außerdem stets durch die Außenseiten desselben Körpers verdeckt.

⇒ Kanten von Innenseiten müssen nicht auf Verdeckung geprüft werden.

⇒ „**Culling**“

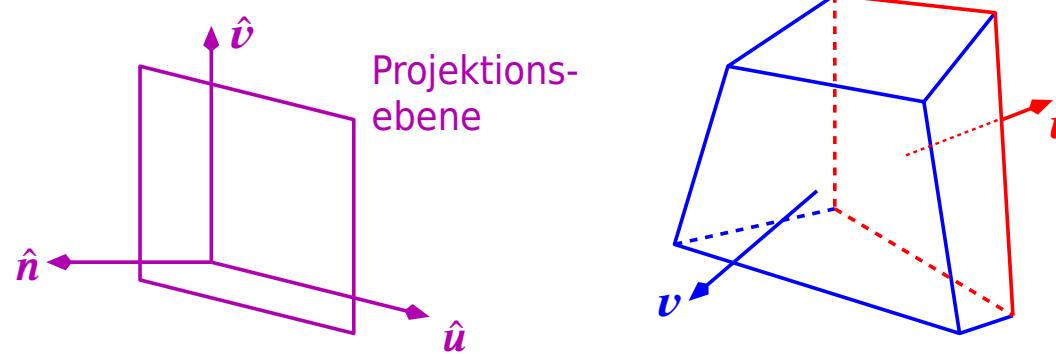
Welche Polygone eines Polyeders sind Außenseiten?

Methode I: Speichere jedes Polygon so ab, dass seine Ecken, **von außerhalb** des Körpers betrachtet, **entgegen dem Uhrzeigersinn nummeriert** sind.



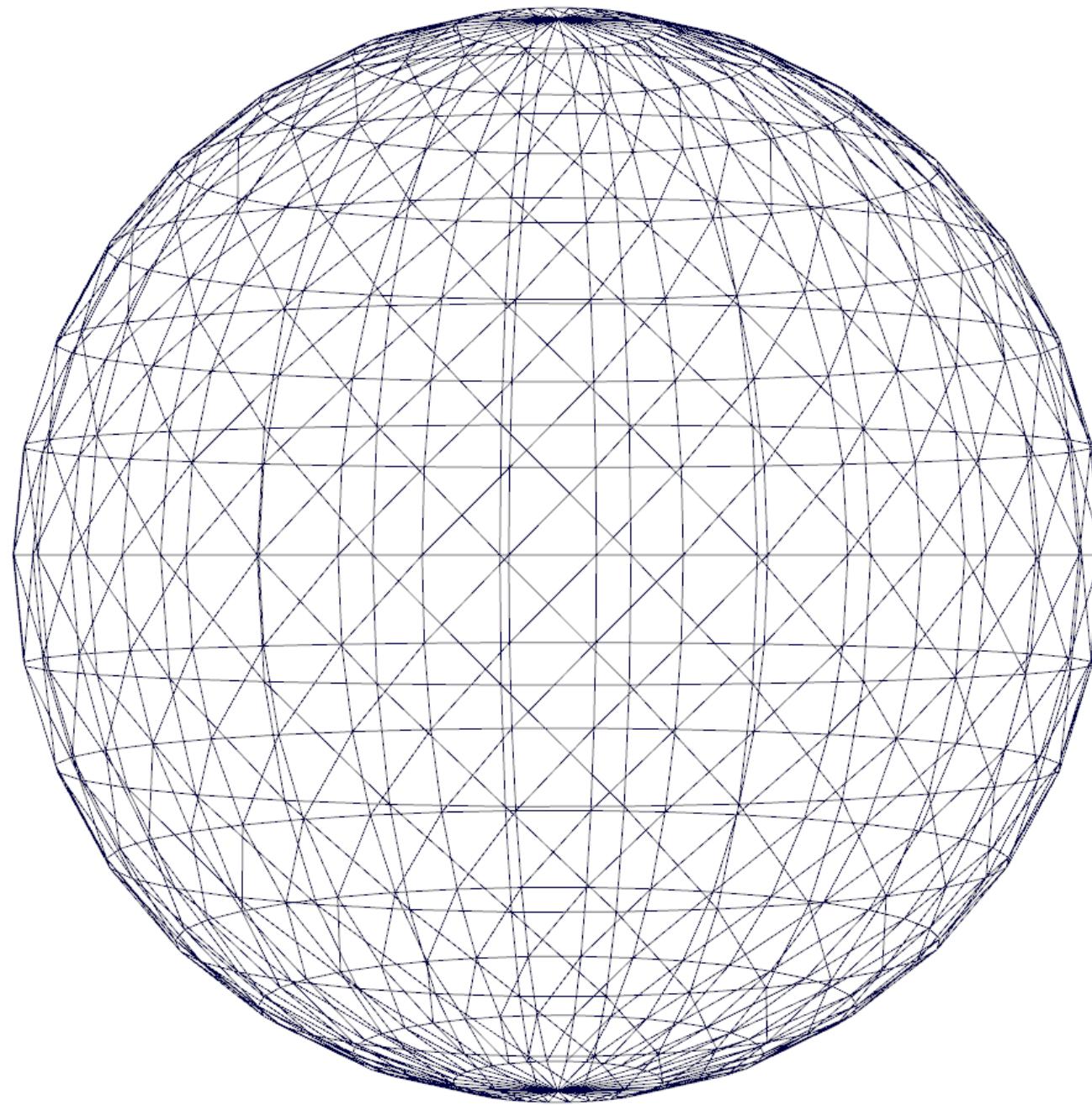
Das Polygon ist genau dann eine Außenseite, wenn auch seine projizierten Ecken entgegen dem Uhrzeigersinn durchlaufen werden.

Methode II: Speichere zu jedem Polygon einen „nach außen“ gerichteten Normalenvektor \mathbf{v} zur Polygonebene ab.



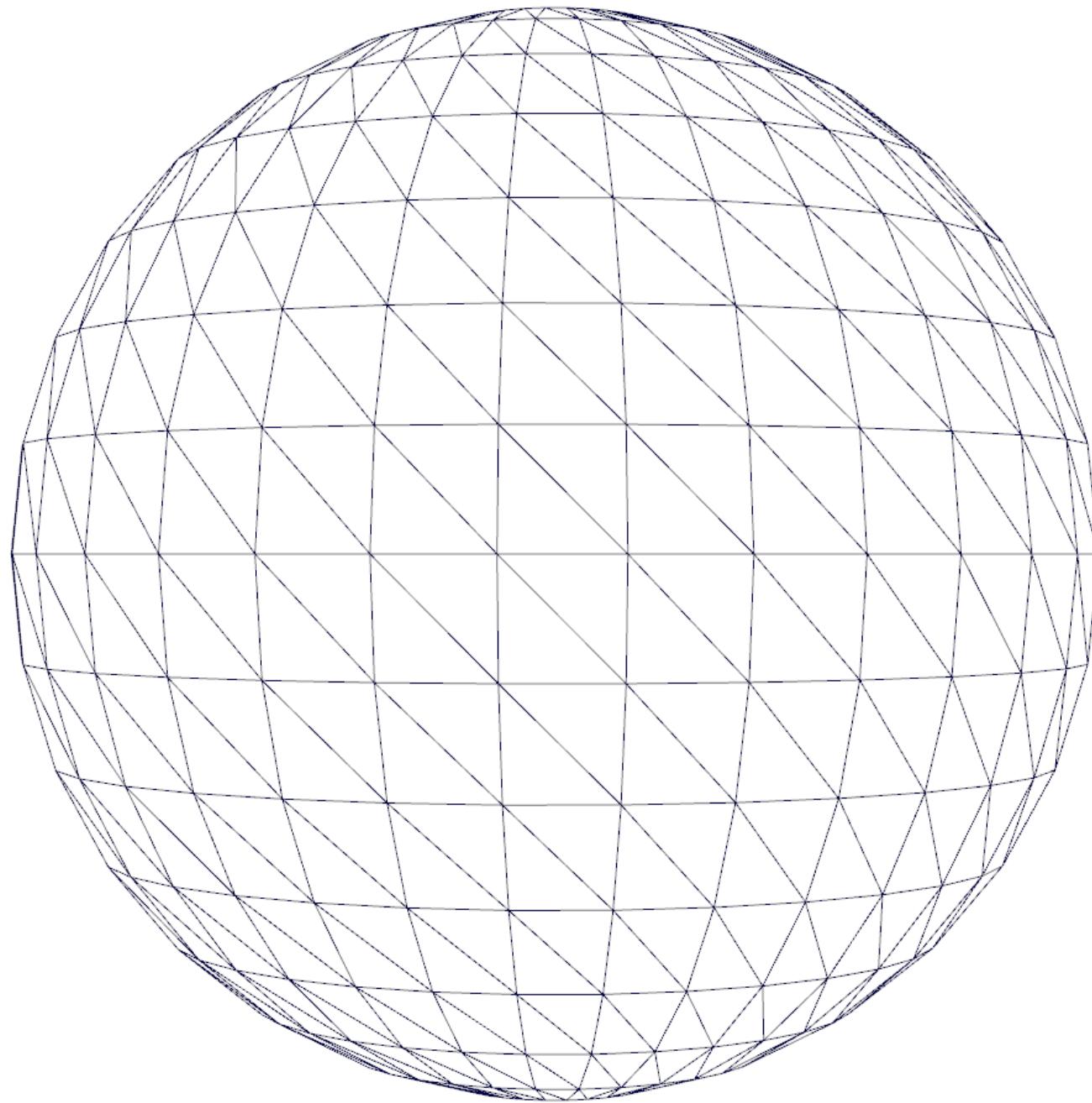
Das Polygon ist genau dann eine Außenseite, wenn

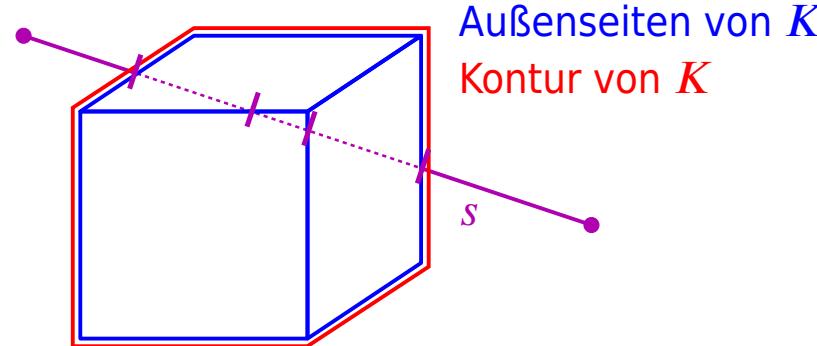
$$\langle \mathbf{v}, \hat{\mathbf{n}} \rangle > 0 .$$



FPS: 59.98

Bilder/glai/flat-sphere-wireframe-culling

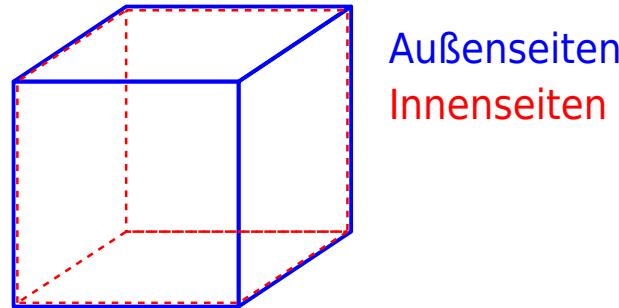


Beobachtung 2:

Die Außenseiten eines Körpers überdecken mehrere Intervalle der Strecke; von diesen Intervallen verschmelzen i. Allg. einige zu größeren Intervallen.

Wenn gesichert ist, dass s den Körper K **nicht durchdringt** (z. B. weil bekannt ist, dass die Körper disjunkt sind) und K konvex ist, dann kann man die größeren Intervalle direkt erhalten, indem man die Strecke s nicht gegenüber den einzelnen Außenseiten von K , sondern gegenüber der **Kontur** von K testet.

Welche Kanten des Körpers K gehören zu seiner Kontur?



- Jede Kante der Kontur gehört sowohl zu einer Außenseite als auch zu einer Innenseite des Körpers.

aber:

- Ist K nicht konvex, gilt die Beobachtung nicht mehr.

(Und Kanten(stücke), die zu Außen- und Innenseiten gehören, gehören nicht unbedingt (vollständig) zur Kontur.)

6.3 Vereinfachte Algorithmen

6.3.1 Painter's Algorithm

Beobachtung: Die weiter hinten liegenden Objekte werden durch weiter vorne liegende Objekte teilweise verdeckt.

(\Rightarrow z-Puffer-Algorithmus, Abschnitt 6.1.3)

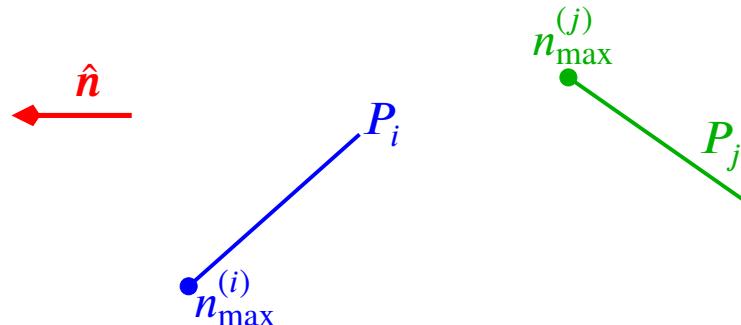
Idee: Statt für jedes einzelne Pixel die z -Koordinaten zu berechnen und dann zu entscheiden, ob das Pixel gezeichnet wird, sorgt man dafür, dass jedes neue Objekt (bzgl. der bereits gezeichneten) **vollständig sichtbar** ist.

- ⇒ Das neue Objekt darf hinter keinem der bereits gezeichneten liegen.
- ⇒ Die Objekte werden in der Reihenfolge „**von hinten nach vorne**“ gezeichnet.

Bemerkung 6.11: Selbst wenn eine solche Reihenfolge möglich ist, ist sie nicht unbedingt einfach herzustellen.

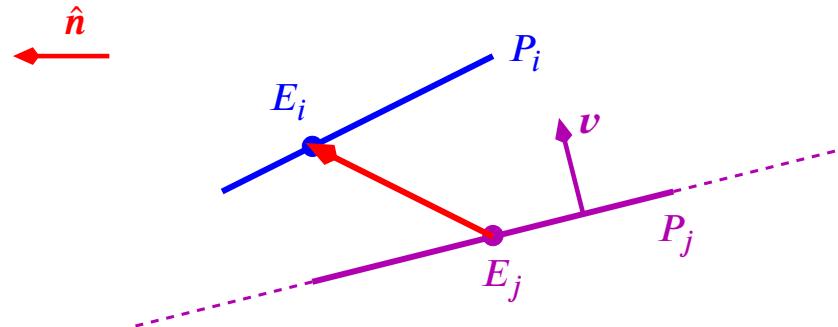
Kriterium I: P_i liegt vor P_j , wenn für die maximalen n -Koordinaten (von Ecken) der Polygone gilt:

$$n_{\max}^{(i)} > n_{\max}^{(j)}$$



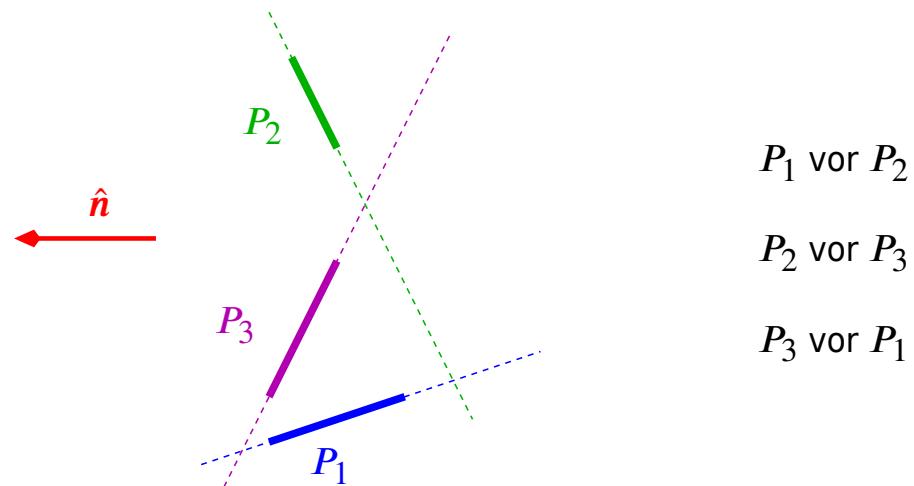
Dieses Kriterium ist i. Allg. **nicht korrekt**:

Kriterium II: P_i liegt vor P_j , wenn P_i vor der **Polygonebene** von P_j liegt.



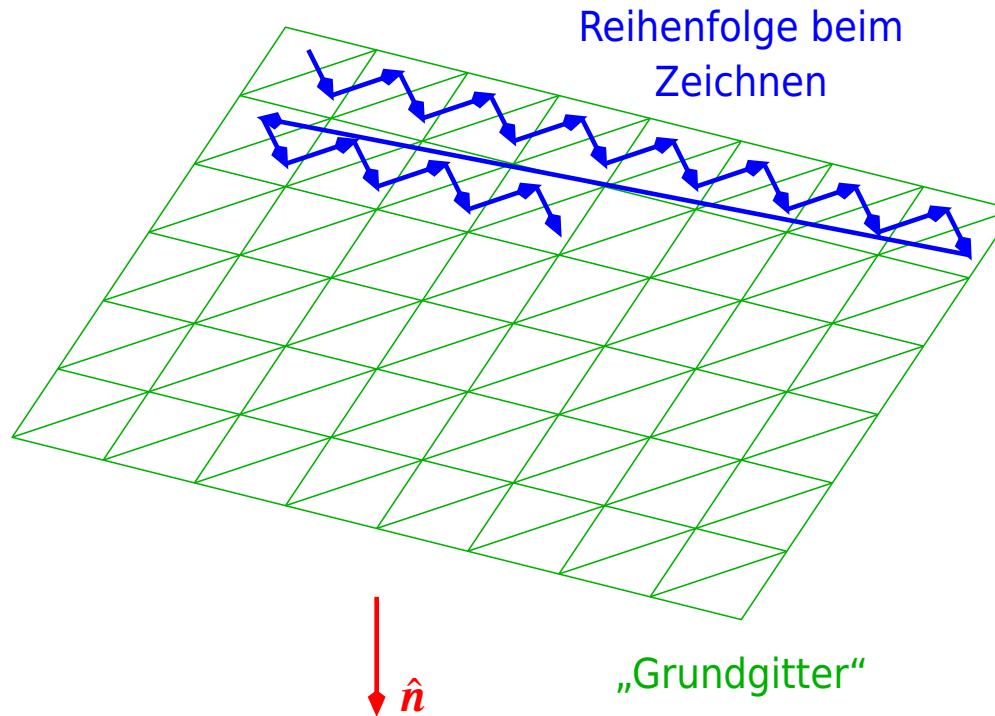
- Wähle eine „Vorwärts“-Normale \boldsymbol{v} zu P_j .
(d. h. $\langle \boldsymbol{v}, \hat{\boldsymbol{n}} \rangle > 0$)
- Wähle eine feste Ecke E_j von P_j .
- P_i liegt vor P_j , wenn
$$\langle E_i - E_j, \boldsymbol{v} \rangle > 0 \quad \text{für alle Ecken } E_i \text{ von } P_i .$$

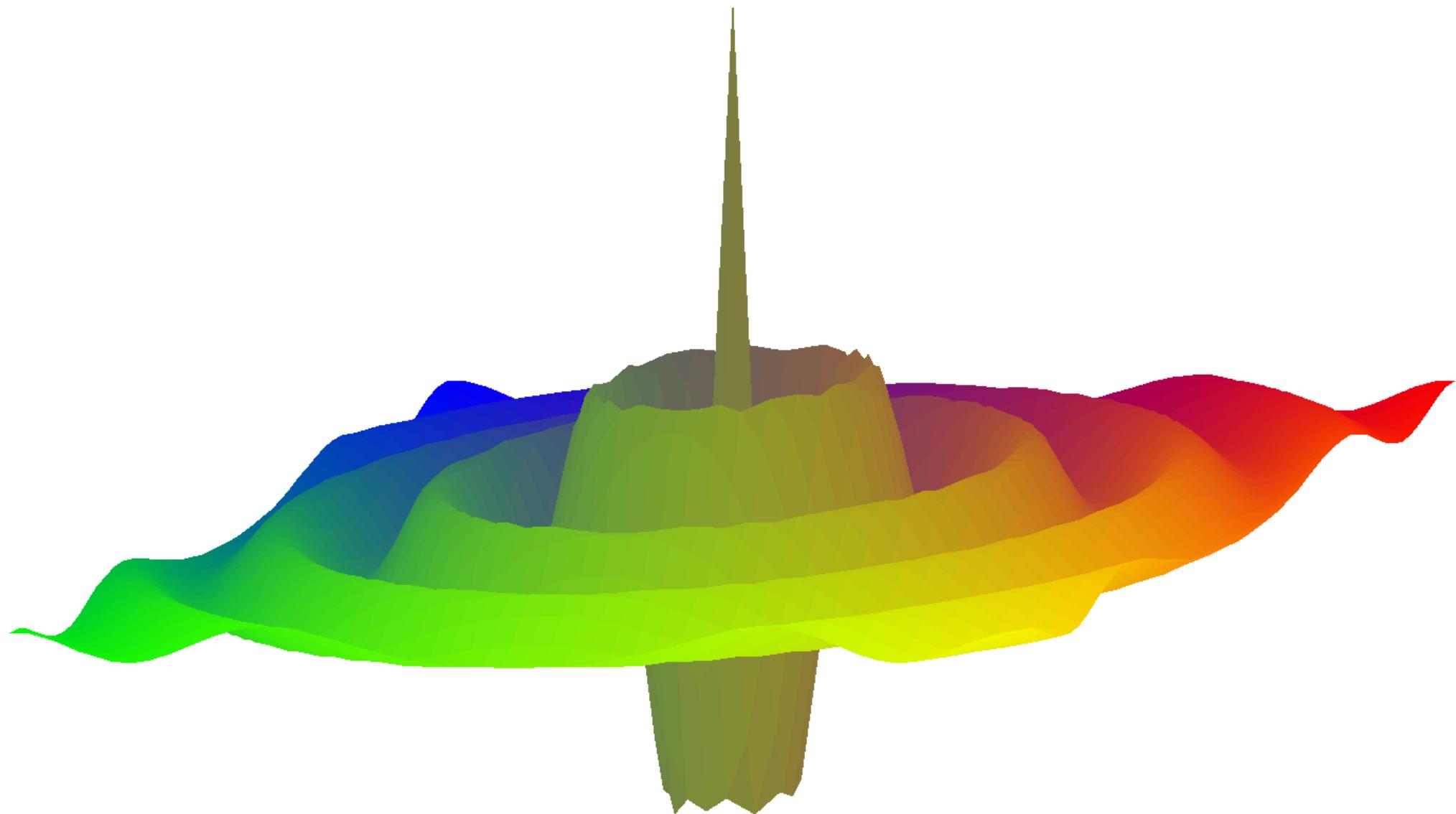
Das Kriterium ist korrekt, wenn die Polygone sich nicht durchdringen; es liefert aber nicht immer eine Ordnung.



aber: nicht immer ist eine Reihenfolge überhaupt möglich:

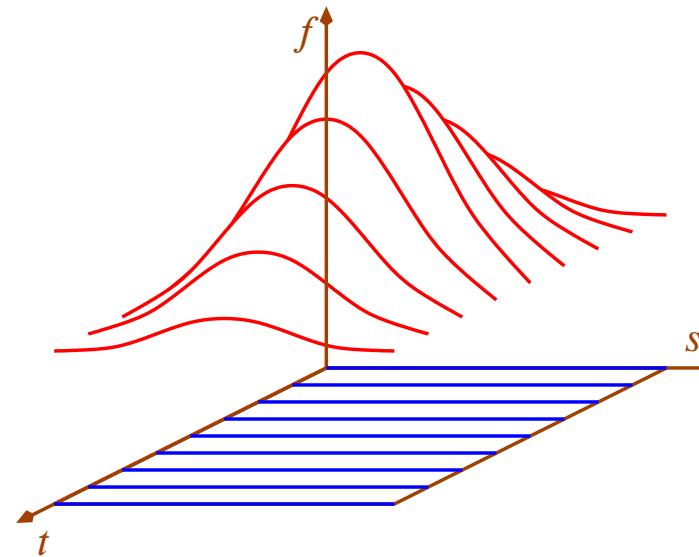
Anwendung von Painter's Algorithm: Wenn die Polygone sich leicht bzgl. der n - bzw. z -Koordinate ordnen lassen, z. B. bei Flächen, die über einem **Gitter** definiert sind:





6.3.2 Der Silhouetten-Algorithmus

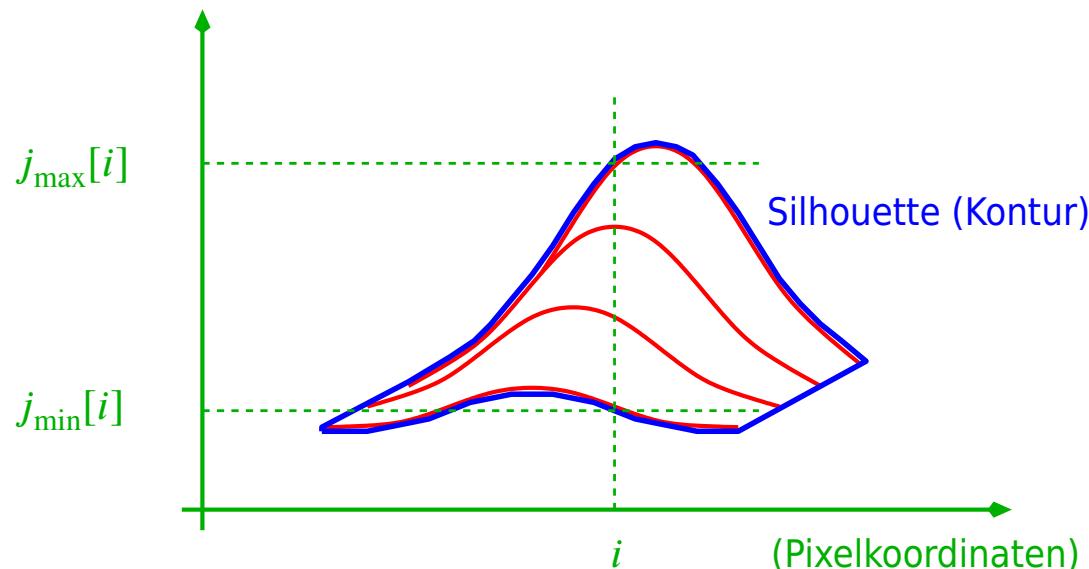
Anwendung: Darstellung von Funktionen $f = f(s, t)$ über einem Parameterrechteck $[\underline{s}; \bar{s}] \times [\underline{t}; \bar{t}]$



Die Funktion wird über parallelen Strecken (z. B. $t \equiv \text{const}$) abgetragen (\Rightarrow Kurven), die verdeckten Teile der Kurven werden nicht gezeichnet.

Bemerkung 6.12: Die Fläche zwischen den Kurven $t \equiv t_1$ und $t \equiv t_2$ ist zusammenhängend und besitzt keine Löcher, also gilt dies auch für ihre Projektion.

\Rightarrow Die Silhouette (Kontur) des Flächenstücks bestimmt, was durch das Flächenstück verdeckt wird.



Wenn man die Kurven $t \equiv \text{const}$ **von vorne nach hinten** zeichnet, dann wird die Kontur der bisher gezeichneten Fläche durch zwei Felder beschrieben:

$j_{\min}[i]$ Nummer des untersten bislang in Pixelspalte i gezeichneten Pixels

$j_{\max}[i]$ Nummer des obersten Pixels

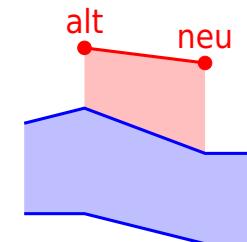
Die nächste zu zeichnende Kurve wird aus einzelnen Strecken

von $\underbrace{(i - 1, j_{i-1})}_{\text{„alt“}}$ nach $\underbrace{(i, j_i)}_{\text{„neu“}}$

zusammengesetzt.

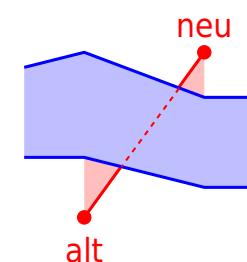
Fall 1: alt und neu liegen auf derselben Seite der Kontur:

- ⇒ Strecke zeichnen,
- Kontur aktualisieren



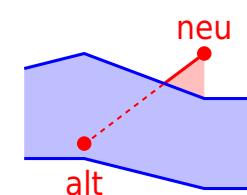
Fall 2: alt und neu liegen auf verschiedenen Seiten der Kontur:

- ⇒ zwei Teilstrecken zeichnen,
- Kontur aktualisieren



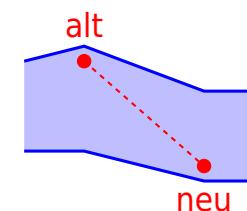
Fall 3: Einer der Punkte alt bzw. neu liegt innerhalb der Kontur, der andere außerhalb:

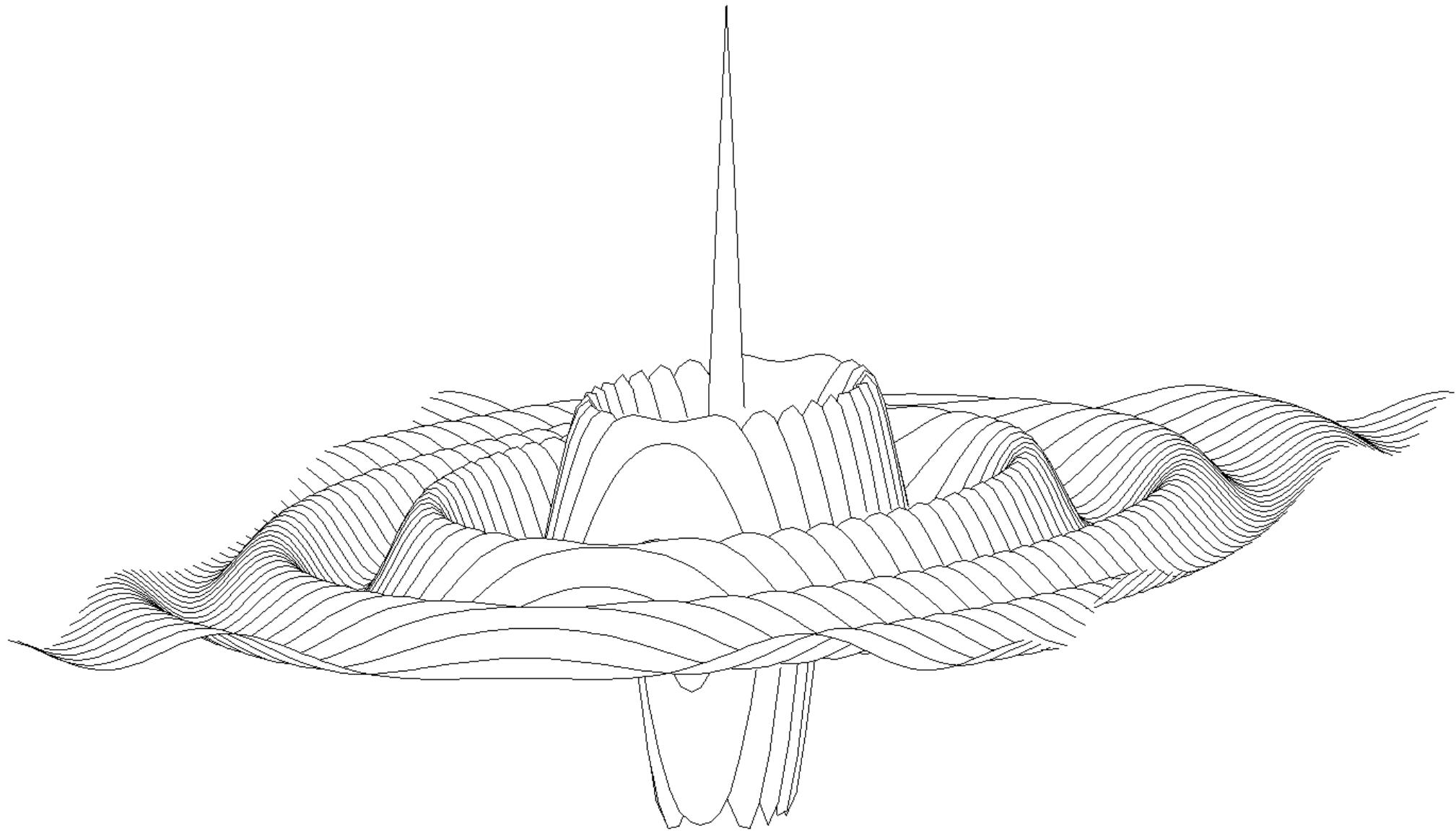
- ⇒ eine Teilstrecke zeichnen,
- Kontur aktualisieren



Fall 4: alt und neu liegen innerhalb der Kontur:

- ⇒ Strecke vollständig verdeckt





7 Färbung

Inhalt

7.1 Lichtmodelle	7-3
7.1.1 Das physikalische Modell	7-3
7.1.2 Das Sampling-Modell	7-5
7.1.3 Das Faltungs-Modell	7-7
7.1.4 Das RGB-Modell	7-10
7.1.5 Das HSV/HLS-Modell	7-12
7.2 Modellierung optischer Effekte (I)	7-15
7.2.1 Ambientes Licht	7-16
7.2.2 Diffuse Reflexion	7-17
7.2.3 Winkelabhängige Reflexion	7-19
7.2.4 Entfernungsabhängige Dämpfung	7-21
7.2.5 Farbverschiebung (Depth Cueing)	7-23
7.2.6 Gerichtete Lichtquellen nach Warn	7-25
7.3 Lokale Beleuchtungsmodelle	7-29
7.3.1 Das „triviale“ Modell	7-30
7.3.2 Das Modell von Bouknight	7-31
7.3.3 Das Modell von Phong	7-32
7.4 Färbungsstrategien für Polygone	7-33

7.4.1 Konstante Färbung	7-35
7.4.2 Farbwertinterpolation (Gouraud-Shading)	7-40
7.4.3 Normaleninterpolation (Phong-Shading)	7-44
7.5 Modellierung optischer Effekte (II)	7-46
7.5.1 Schatten	7-47
7.5.2 Oberflächenstruktur (Textur)	7-53
7.5.3 Transparenz	7-63

Problem: Bei Pixel (i, j) sei Objekt O_k sichtbar.

Wie ist Pixel (i, j) einzufärben?

Die Farbe des Pixels hängt ab von:

- den (Material-)Eigenschaften des Objekts O_k
(Farbe, Rückstrahlkraft, Transparenz, ...)
- den Eigenschaften der Lichtquellen
(Farbe, Leuchtkraft, Ausdehnung, ...)
- der Geometrie der Szene
(Einfallsinkel des Lichts, Verdeckungen, Schatten, ...)

Zur Bestimmung der Pixelfarbe gibt es verschiedene Verfahren.

Diese unterscheiden sich in

- der Beschreibung der obigen Eigenschaften,
- der Umsetzung der Eigenschaften in die Pixelfarbe.

Die Wahl des Verfahrens ist bestimmt durch den zulässigen Aufwand.

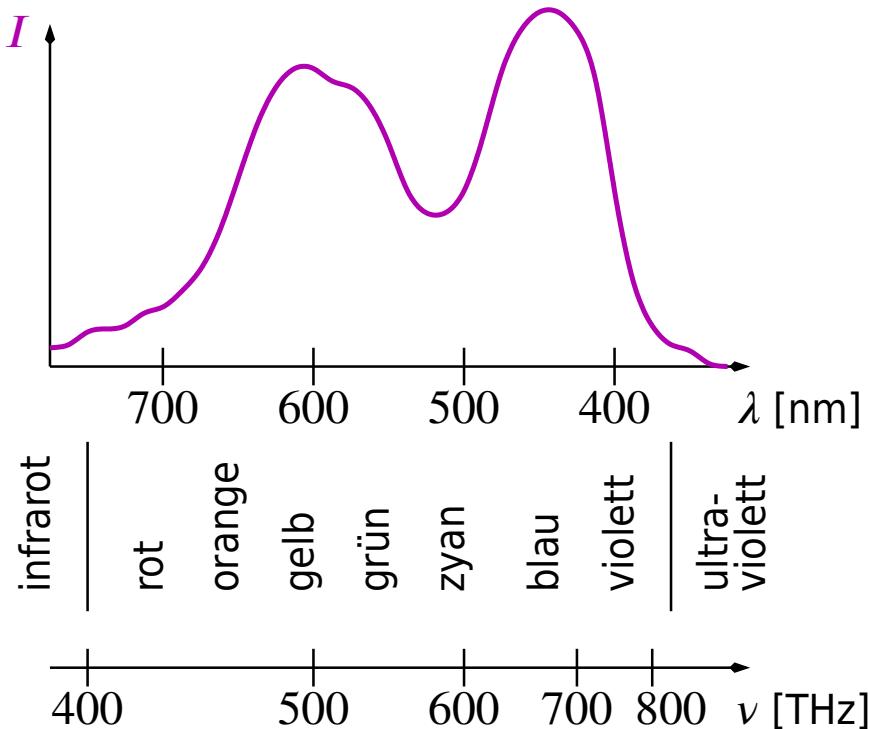
7.1 Lichtmodelle

Zweck: **quantitative** Beschreibung des Lichts, das auf ein Objekt trifft bzw. vom Objekt ausgeht

7.1.1 Das physikalische Modell

Motivation: Licht ist eine Überlagerung von Wellen unterschiedlicher Wellenlängen.

Beschreibung: **kontinuierliche Intensitätsverteilung** $I = I(\lambda)$



⇒ Alle physikalischen Effekte werden ebenfalls durch kontinuierliche Funktionen beschrieben.

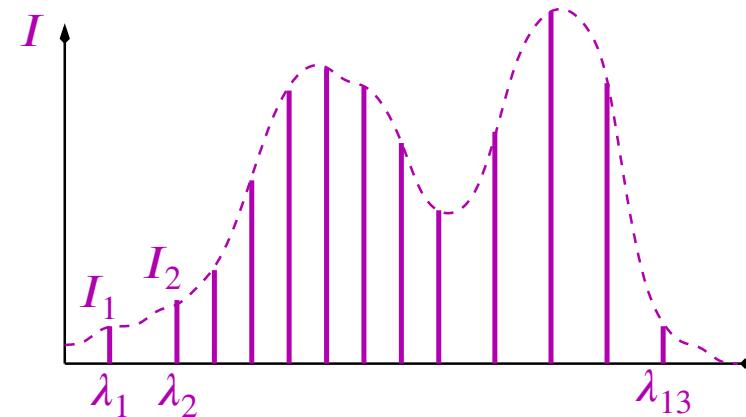
Beispiel: Reflexion

$$\underbrace{I_{\text{aus}}(\lambda)}_{\text{reflektiertes Licht}} = \underbrace{\rho(\lambda)}_{\text{Reflexionskoeffizient}} \cdot \underbrace{I_{\text{ein}}(\lambda)}_{\text{einfallendes Licht}}$$

- ⊕ sehr mächtiges Modell
- ⊖ auf dem Rechner kaum zu realisieren (kontinuierliche Funktionen!)
- ⊖ nicht auf Bildschirm/Papier darstellbar

7.1.2 Das Sampling-Modell

Idee: Der grobe Verlauf der $I(\lambda)$ -Kurve kann oft bereits durch wenige Werte $I_j = I(\lambda_j)$ an **fest vorgegebenen** Stellen $\lambda_j, j = 1, \dots, m$, beschrieben werden.



Beschreibung: **Tupel** $I = (I_1, \dots, I_m)$

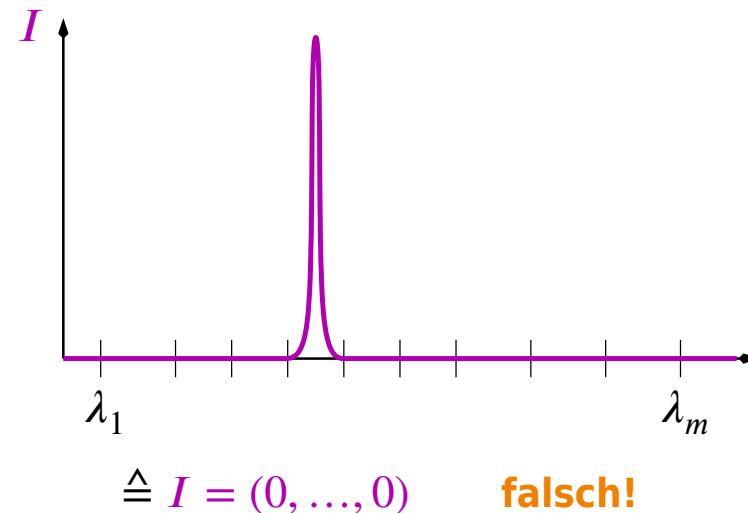
⇒ Physikalische Effekte werden entsprechend modelliert, z. B.

$$I_{\text{aus},j} = \rho_j \cdot I_{\text{ein},j}, \quad j = 1, \dots, m.$$

Bemerkung 7.1: Die λ_j müssen **nicht äquidistant** gewählt werden (i. Allg. weniger dicht im Blau-Bereich).

Bemerkung 7.2: Wenn einfarbiges Licht vorkommt (z. B. Laser), dann muss dieses durch eine entsprechende λ_j -Komponente berücksichtigt werden.

sonst:



- ⊕ bei geeigneter Wahl der λ_j praktisch dieselbe Qualität wie beim physikalischen Modell, falls m „nicht zu klein“
- ⊕ relativ einfach und (für kleine m) effizient auf dem Rechner zu realisieren
- ⊖ Umsetzung in Bildschirm-/Druckerfarben notwendig

7.1.3 Das Faltungs-Modell

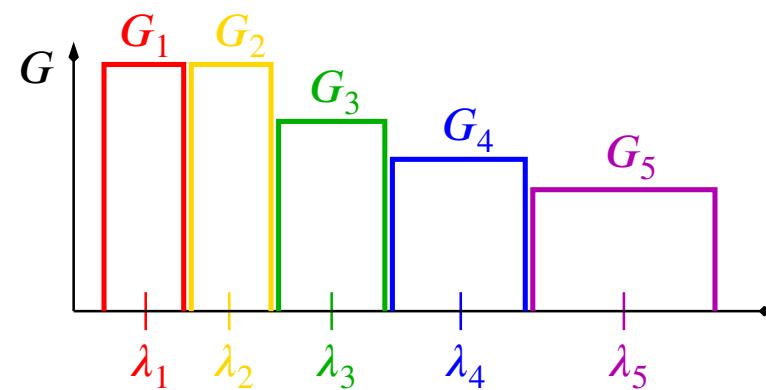
Idee: Berücksichtige bei I_j nicht nur die Wellenlänge λ_j , sondern einen ganzen **Wellenlängenbereich** um λ_j :

$$I_j = \int_0^{\infty} I(\lambda) \cdot G_j(\lambda) d\lambda$$

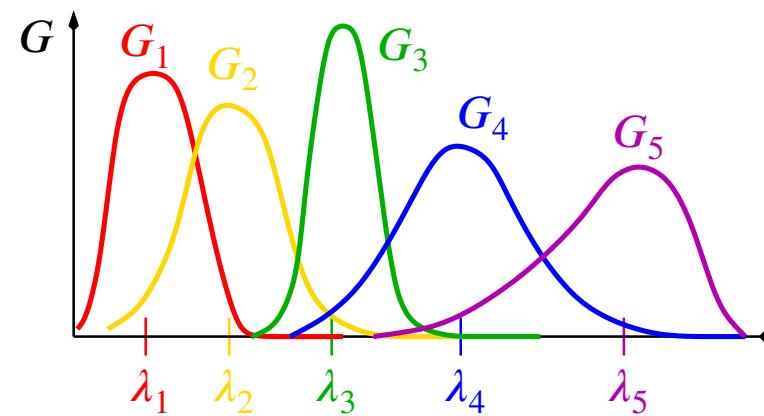
mit einer **Gewichtsfunktion** G_j

Beispiele 7.3:

gleichmäßige Gewichtung:



ungleichmäßige Gewichtung:



Bemerkung 7.4: Die Gewichtsfunktionen sollten so gewählt werden, dass

$$\int_0^\infty G_j(\lambda) d\lambda = 1 \quad \text{für } j = 1, \dots, m$$

sowie

$$\sum_{j=1}^m G_j(\lambda) \geq \text{const} > 0 \quad \text{"im interessanten } \lambda\text{-Bereich"}$$

(d. h. alle Wellenlängen werden „ungefähr gleich stark“ berücksichtigt; dann kann einfarbiges Licht nicht mehr „übersehen“ werden).

⊕ Gerechnet wird wie in 7.1.2, also z. B.

$$I_{\text{aus},j} = \rho_j \cdot I_{\text{ein},j}, \quad j = 1, \dots, m.$$

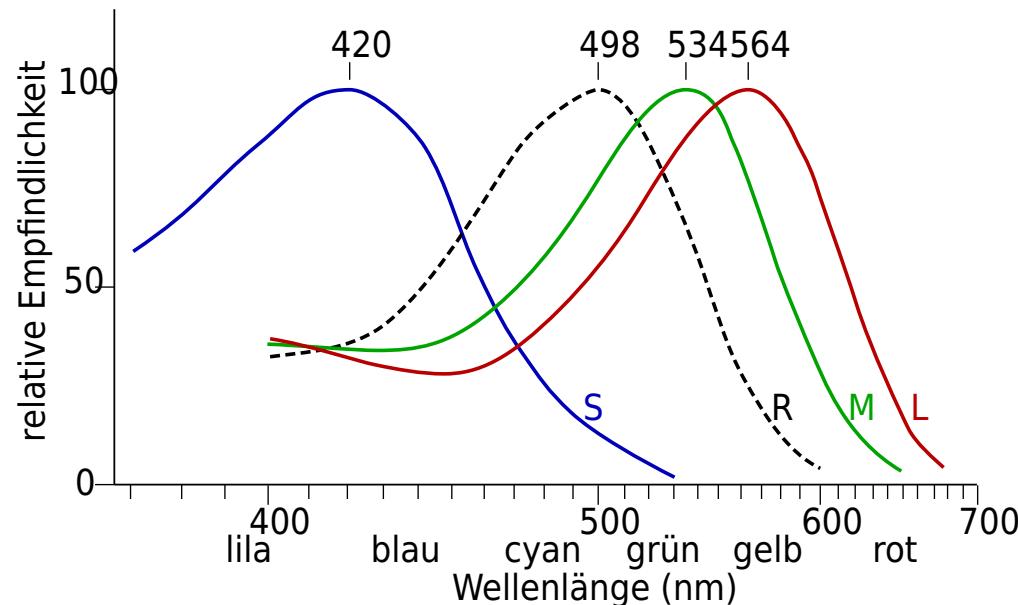
Die Faltung wird nur berücksichtigt

- bei der Ermittlung der Intensitäten I_j jeder Lichtquelle,
- evtl. bei den Reflexionskoeffizienten ρ_j ,

also in einer einmaligen **Vorverarbeitung**.

7.1.4 Das RGB-Modell

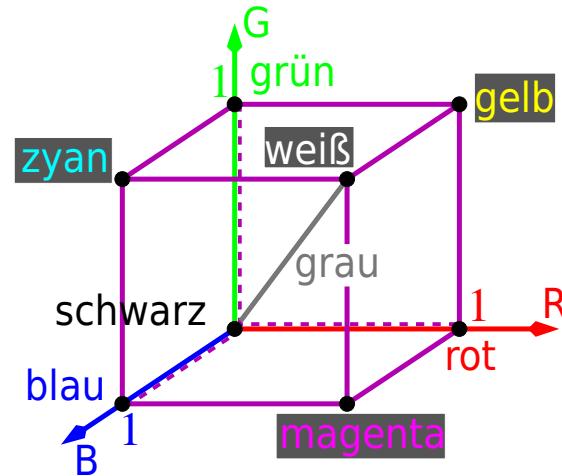
Spezialfall des Faltungsmodells mit $m = 3$



Absorptionskurven der menschlichen Fotorezeptoren: Zapfen (blau, grün, rot) und Stäbchen (scharz gestrichelt)
Grafik: Cone-response.svg: w:User:DrBob and w:User:Zeimusu derivative work: Sgbeer, Quelle: <https://commons.wikimedia.org/wiki/File:Cone-response-de.svg>, Titel: „Cone-response-de“, Lizenz: © ⓘ ⓘ 3.0, <https://creativecommons.org/licenses/by-sa/3.0/legalcode>

Motivation: In der Netzhaut gibt es drei Typen von Rezeptoren, deren Empfindlichkeiten etwa durch die obigen Gewichtungen gegeben sind; sie sprechen vorwiegend auf **Rotes**, **Grünes** bzw. **Blaues** Licht an.

Bemerkung 7.5: Die R-, G-, B-Intensitäten werden oft auf das Intervall $[0; 1]$ normiert:



- ⊖ weniger mächtig als das physikalische Modell
 - nicht alle Farben darstellbar
 - Bei sehr komplexen Beleuchtungsmodellen kann es zu Farbverfremdungen kommen.
- aber meist ausreichend
- ⊕ sehr einfach und effizient
- ⊕ wird von praktisch allen Farb-Raster-Displays direkt unterstützt
- ⇒ eindeutig dominierend

7.1.5 Das HSV/HLS-Modell

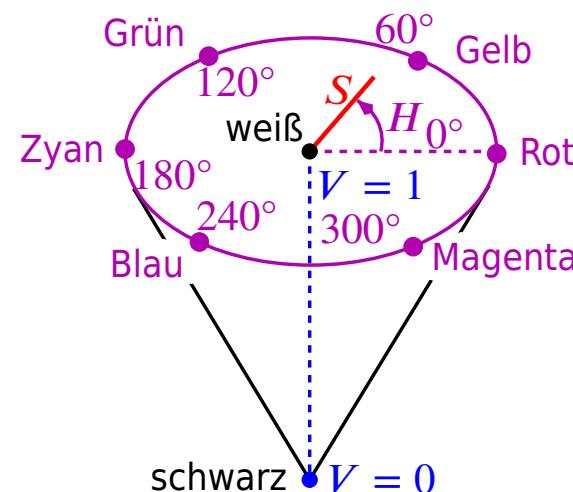
Motivation: Das HSV- bzw. HLS-Modell wird häufig im künstlerischen und Design-Bereich verwendet. Es imitiert das Zusammenstellen der Farben beim Malen:

- Wähle einen **dominanten Farbton** (hier: dominante Wellenlänge, Hue).
 - Mische zu dieser Wellenlänge weißes Licht, d. h. bestimme die **Sättigung** (Saturation) der Farbe.



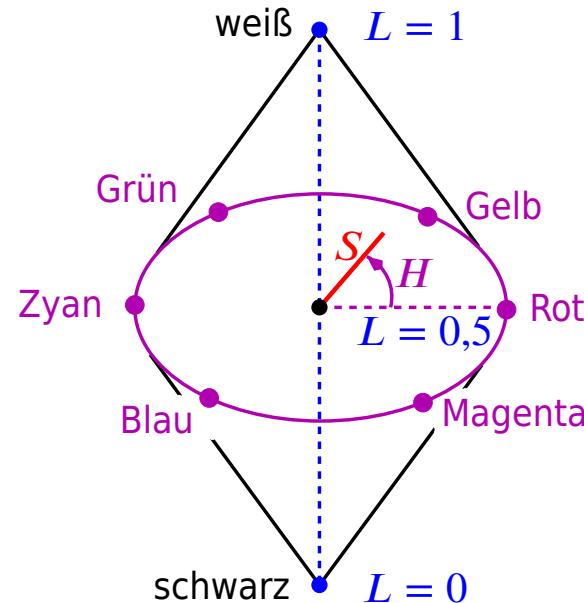
- Lege die **Gesamthelligkeit** (Lightness/Value) der Farbe durch Mischen mit Schwarz fest.

HSV-Kegel: $H \in [0^\circ; 360^\circ]$, $S \in [0; 1]$, $V \in [0; 1]$



$$\text{schwarz} \triangleq \begin{pmatrix} ?, ?, 0 \\ H, S, V \end{pmatrix}, \text{ weiß} \triangleq \begin{pmatrix} ? \\ \text{undefiniert} \end{pmatrix}, 0, 1$$

HLS-Doppelkegel: $H \in [0^\circ; 360^\circ]$, $L \in [0; 1]$, $S \in [0; 1]$



$$\text{schwarz} \triangleq \left(\underbrace{?, 0, ?}_{\begin{matrix} H \\ L \\ S \end{matrix}} \right), \text{weiß} \triangleq \left(\underbrace{?}_{\text{undefiniert}}, 1, ? \right)$$

- HSV/HLS ist äquivalent zum RGB-Modell;
es gibt Algorithmen zur Umrechnung der Modelle.

- + gut zum Festlegen von Farben geeignet
- für Rechnung weniger geeignet

(Welches H , S , V hat Licht, das sich aus der Überlagerung von grünem und magentafarbenem Licht ergibt?)

⇒ In den meisten Grafik-Paketen wird HSV/HLS nur für die Ein-/Ausgabe der Materialeigenschaften bereitgestellt, **gerechnet wird in RGB**.

7.2 Modellierung optischer Effekte (I)

Problem: Bei Pixel (i, j) ist ein Punkt bzw. ein kleines Flächenstück von Objekt O_k sichtbar.

Wie viel Licht fällt auf dieses Flächenstück, und wie viel davon wird in Richtung des Pixels weitergegeben?

Bemerkung 7.6: Die folgenden Gleichungen sind jeweils für alle im Lichtmodell vorkommenden Wellenlängen(bereiche) zu betrachten, also z. B. für R, G, B.

Die vorkommenden Koeffizienten (für Reflexion usw.) sind i. Allg. abhängig von der betrachteten Wellenlänge.

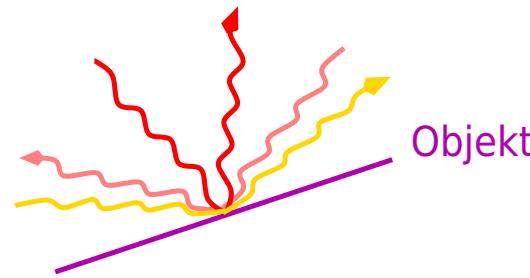
Bemerkung 7.7: Die folgenden Modelle sind alle mehr oder weniger reine **Heuristik**; für viele Anwendungen sind sie jedoch völlig ausreichend.

Ziel ist es, bestimmte Effekte mit möglichst **geringem Rechenaufwand** nachzuahmen.

7.2.1 Ambientes Licht

Motivation: Eine Szene wird meist auch durch Licht erhellt, dem kein Ursprung und keine Richtung mehr zugeordnet werden kann, z. B.

- Tageslicht durch die Atmosphäre (aber nicht direktes Sonnenlicht),
- Licht einer Lampe, das bereits mehrmals an matten Wänden reflektiert wurde.



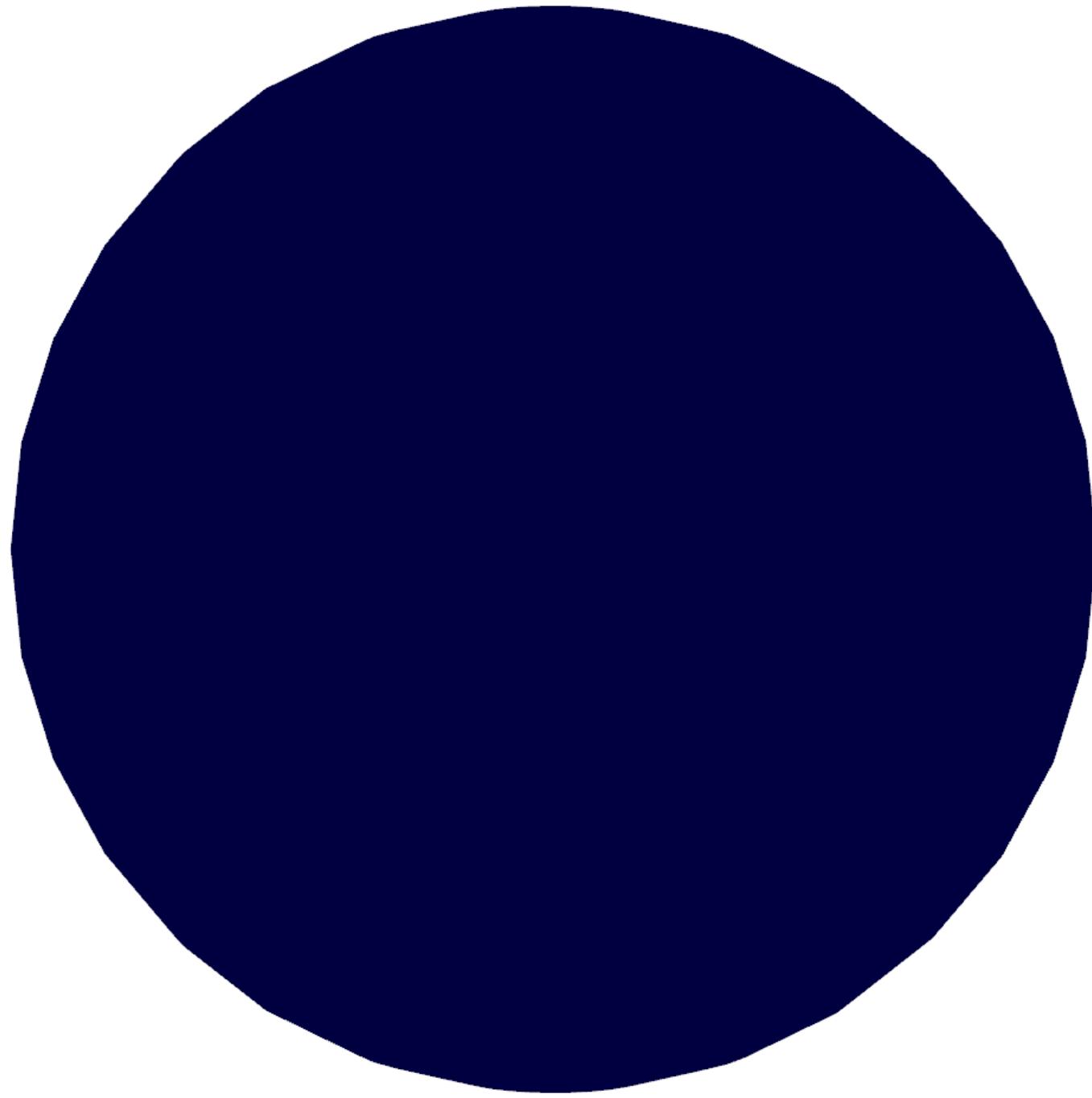
Modell: Die Rückstrahlung hängt nur von der Gesamtintensität des einfallenden ambienten Lichts und von den Materialeigenschaften des Objekts ab, nicht von der Orientierung des Objekts (z. B. bezüglich des „Augenpunktes“).

$$\underbrace{I_a(\mathbf{P})}_{\text{Intensität des bei Punkt } \mathbf{P} \text{ reflektierten ambienten Lichts}} = \underbrace{I_a}_{\text{Intensität des ambienten Lichts in der Szene}} \cdot \underbrace{R_{k,a}}_{\text{Reflexionskoeffizient von Objekt } k \text{ für ambientes Licht}}$$

- ⇒ Die Intensität des reflektierten ambienten Lichts ist für alle zu einem Objekt gehörigen Pixel gleich.
- ⊖ In der Realität ist ambientes Licht i. Allg. abhängig vom Ort \mathbf{P} und besitzt oft (abhängig von \mathbf{P}) bevorzugte Richtungen.

FPS: 59.98

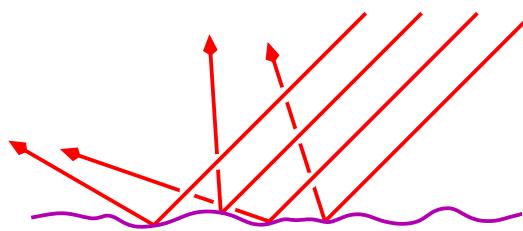
Bilder/glai/flat-sphere-ambient



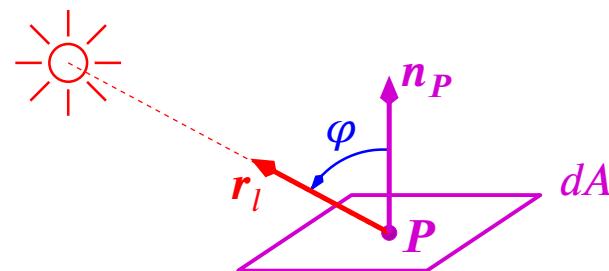
L

7.2.2 Diffuse Reflexion

Motivation: Die Oberfläche der Objekte ist i. Allg. nicht völlig eben. Kleinste Unebenheiten führen zur Streuung des einfallenden Lichts.



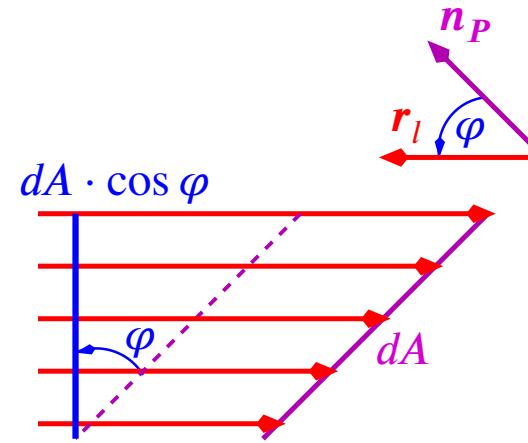
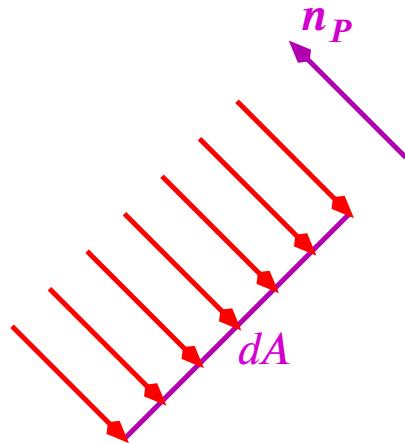
Modell: Ein Teil des (von einer Lichtquelle) beim Punkt \mathbf{P} eintreffenden Lichts wird in alle Richtungen gleichmäßig reflektiert. Die Rückstrahlung hängt nur von der Menge des einfallenden Lichts und den Materialeigenschaften ab, nicht von der Orientierung des Objekts bzgl. des Augenpunktes.



\mathbf{n}_P : Einheitsnormale zu Objekt k im Punkt \mathbf{P}

\mathbf{r}_l : Einheitsvektor in Richtung zu Lichtquelle L_l

dA : kleines Flächenstück von Objekt k

Beobachtung:

Der Lichteinfall auf die Fläche ist proportional zu

$$\cos \varphi = \mathbf{n}_P^T \cdot \mathbf{r}_l .$$

(Lambertsches [Kosinus-]Gesetz)

Insgesamt folgt:

$$\underbrace{I_d(\mathbf{P})}_{\substack{\text{bei Punkt } \mathbf{P} \text{ diffus} \\ \text{reflektiertes Licht}}} = \sum_l \underbrace{(\mathbf{n}_P^T \cdot \mathbf{r}_l)}_{\substack{\text{über alle} \\ \text{Lichtquellen}}} \cdot \underbrace{I_l}_{\substack{\text{Intensität des von} \\ \text{Lichtquelle } l \text{ ausgehenden Lichts}}} \cdot \underbrace{R_{k,d}}_{\substack{\text{Reflexionskoeffizient} \\ \text{von Objekt } k \text{ für} \\ \text{diffuse Reflexion}}}$$

- ⊖ Diffuse Reflexion ist in der Realität meist nicht völlig unabhängig vom „Ausfallswinkel“.

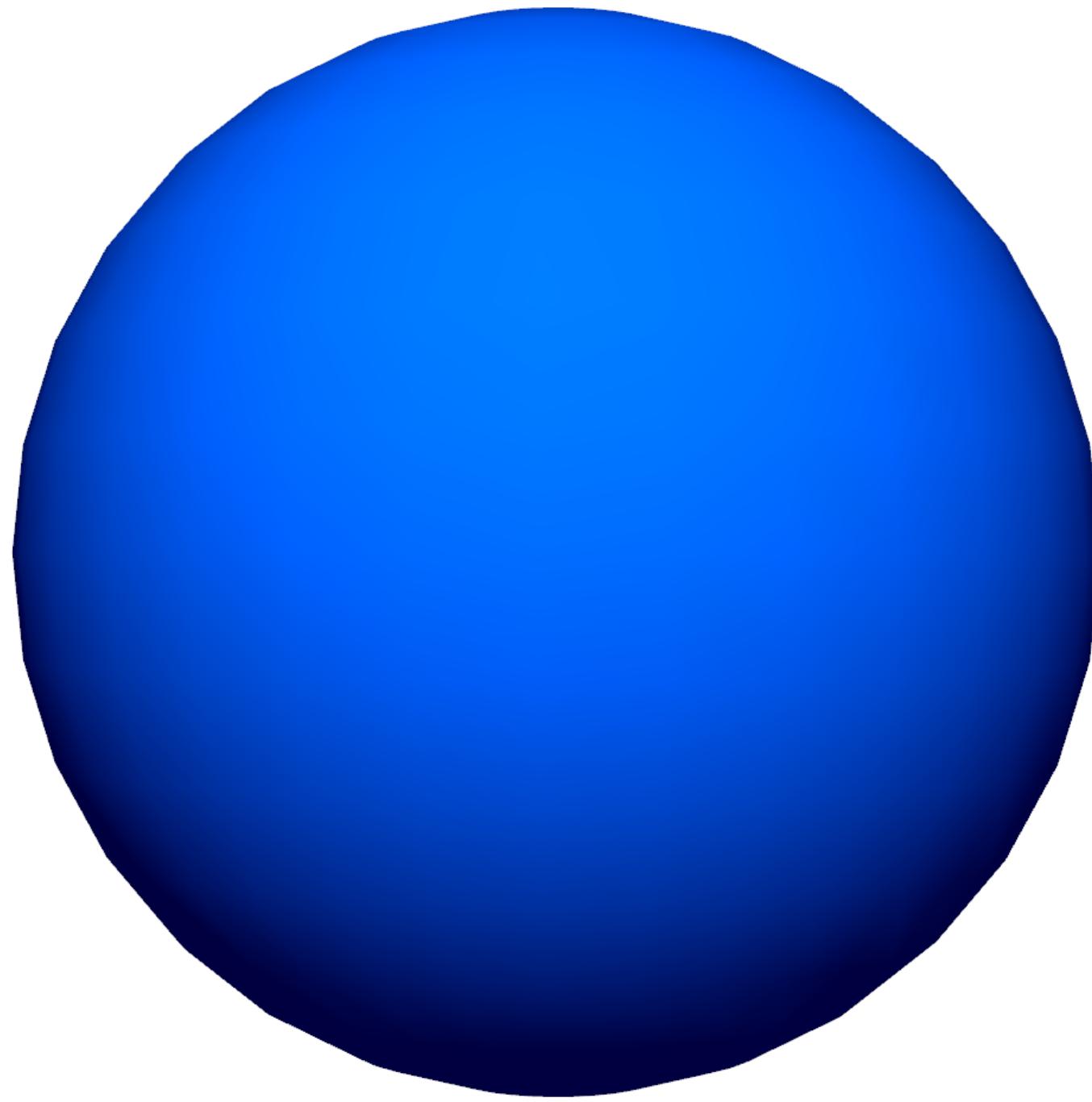
Johann Heinrich Lambert
 * 1728, Mülhausen (damals zur Schweiz, heute Elsass, Frankreich)
 † 1777, Berlin
 Mathematiker, Physiker, Philosoph



Foto: ?, Lithographie Godefroy Engelmann
 Quelle: <https://commons.wikimedia.org/wiki/File:JLambert.jpg>

FPS: 59.94

Bilder/glai/flat-sphere-diffuse

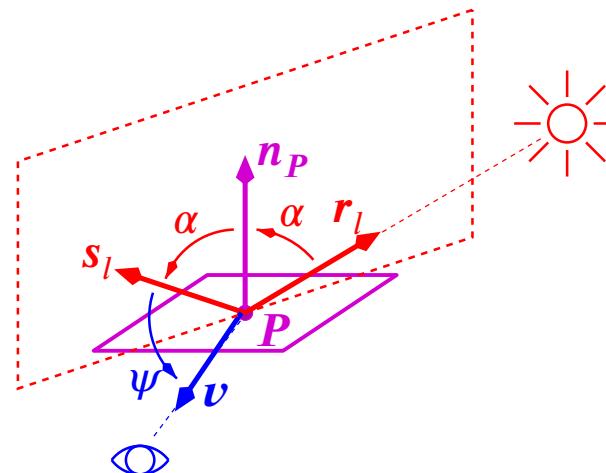


7.2.3 Winkelabhängige Reflexion

Motivation: Bei glatten, stark spiegelnden Oberflächen ist die Reflexion in die durch das „Spiegelgesetz“

$$\text{Einfallswinkel} = \text{Ausfallwinkel}$$

gegebene Richtung s am stärksten, in andere Richtungen nimmt sie rasch ab.



v : Einheitsvektor in Richtung des Augenpunktes

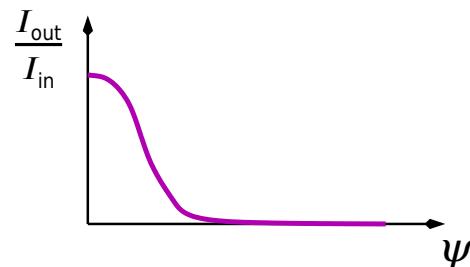
n_P : Einheitsnormale zu Objekt k im Punkt P

r_l : Einheitsvektor in Richtung der Lichtquelle L_l

s_l : Einheitsvektor in der durch das Spiegelgesetz gegebenen Richtung

$$= 2 (\mathbf{n}_P^T \cdot \mathbf{r}_l) \cdot \mathbf{n}_P - \mathbf{r}_l$$

typische Abhängigkeit:



Modell: Der Ansatz

$$I \underset{\substack{\sim \\ \text{proportional}}}{=} \underbrace{(\cos \psi)^{\nu_k}}_{\substack{\text{Exponent vom} \\ \text{Objekt abhängig}}} = (\mathbf{v}^T \cdot \mathbf{s}_l)^{\nu_k}$$

liefert bei verhältnismäßig geringem Aufwand brauchbare Ergebnisse.

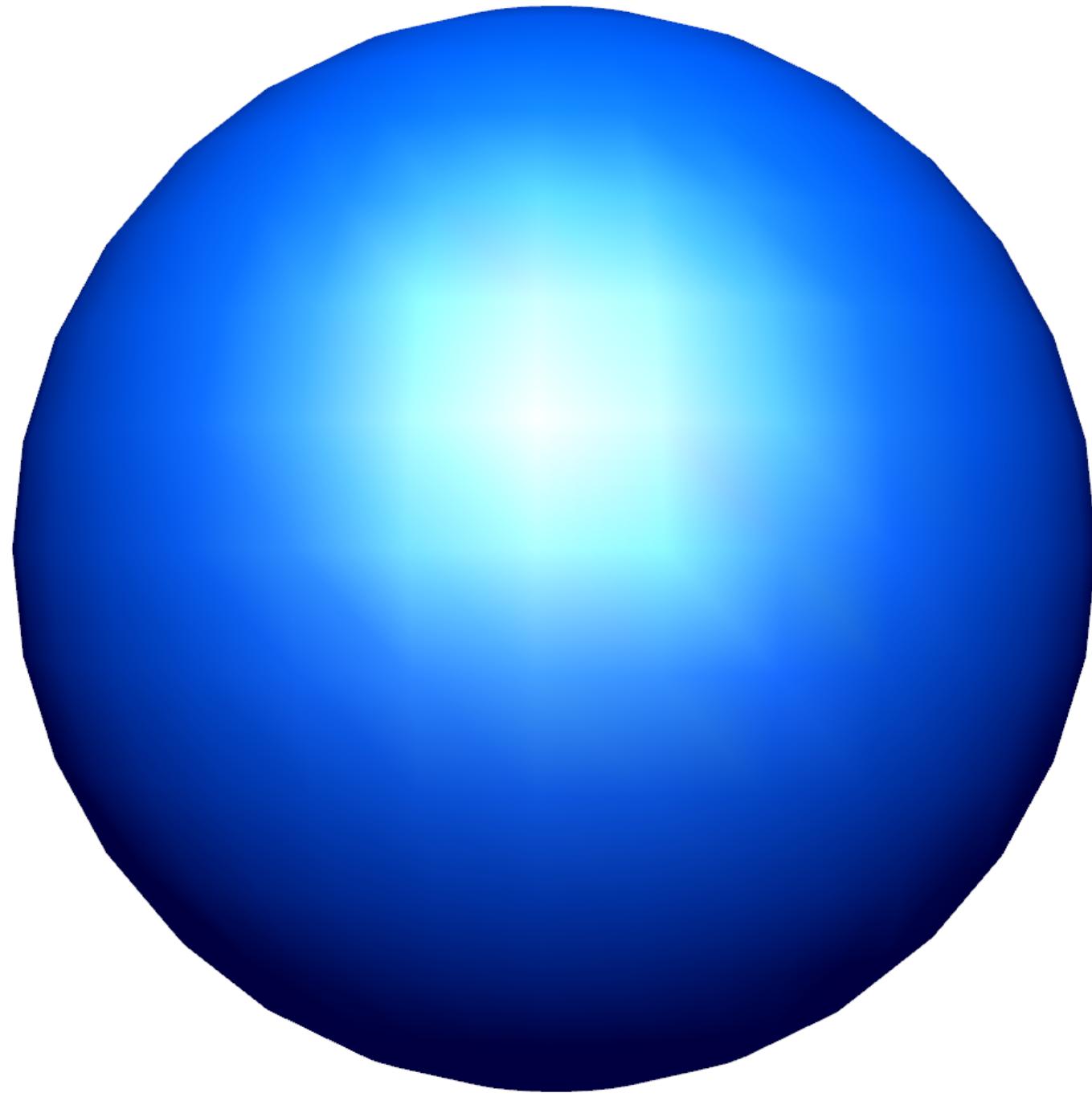
- ν_k „klein“ (z. B. $\nu_k = 5$): Die hellen Flecken (**Glanzlichter**) werden relativ groß und nicht sehr scharf.
- ν_k „groß“ (z. B. $\nu_k = 40$): Glanzlichter werden kleiner und schärfer.

Insgesamt folgt:

$$\underbrace{I_w(\mathbf{P})}_{\substack{\text{bei Punkt } \mathbf{P} \\ \text{winkelabhängig} \\ \text{reflektiertes Licht}}} = \sum_l \underbrace{(\mathbf{v}^T \cdot \mathbf{s}_l)^{\nu_k}}_{\substack{\text{über alle} \\ \text{Lichtquellen}}} \cdot \underbrace{I_l}_{\substack{\text{Intensität des von} \\ \text{Lichtquelle } l \\ \text{ausgehenden Lichts}}} \cdot \underbrace{R_{k,w}}_{\substack{\text{Reflexionskoeffizient} \\ \text{von Objekt } k \text{ für} \\ \text{winkelabhängige} \\ \text{Reflexion}}}$$

FPS: 59.94

Bilder/glai/flat-sphere-specular



7.2.4 Entfernungsabhängige Dämpfung

Motivation: Punkt \mathbf{P} (bzw. Flächenstück dA) auf Objekt O_k erhält umso mehr Licht von Lichtquelle L_l , je geringer der Abstand ist:

$$I_l \sim \frac{1}{d_l^2} \quad \text{mit} \quad d_l = \|\mathbf{P} - \mathbf{L}_l\|_2 \quad (7.1)$$

(Aus der unterschiedlichen Beleuchtung gewinnt das menschliche Sehssystem Information über die Lage der Objekte.)

Bemerkung 7.8: Der Ansatz (7.1) wäre für **Punktlichtquellen** korrekt. Solche kommen aber in der Praxis nicht vor.

⇒ Für Objekte, die sehr nahe bei der Lichtquelle liegen, wird der Helligkeitsunterschied unrealistisch groß.
(Für Objekte, die weit entfernt sind, ist der Helligkeitsunterschied kaum noch wahrnehmbar.)

Heuristik: Verwende einen „**Dämpfungsfaktor**“ der Form

$$f_l(\mathbf{P}) = \min\left(\frac{1}{c_{l,0} + c_{l,1} \cdot d_l + c_{l,2} \cdot d_l^2}, 1\right),$$

wobei die $c_{l,i}$ nur von der Lichtquelle abhängen.

- $c_{l,2}$ liefert die Gesetzmäßigkeit (7.1),
 - $c_{l,0}$ verhindert, dass I in der Nähe der Lichtquelle zu stark variiert,
 - $c_{l,1}$ erhöht die Variation von I in größerer Entfernung.
- ⊕ Bei geschickter Wahl der $c_{l,i}$ kann der räumliche Eindruck deutlich verbessert werden.
- ⊖ Die Wahl der $c_{l,i}$ erfolgt völlig heuristisch.

Bemerkung: OpenGL verzichtet auf das Minimum mit 1; soll Verstärkung verhindert werden, kann $c_{l,0} \geq 1$ gewählt werden.

7.2.5 Farbverschiebung (Depth Cueing)

Motivation: Auf dem Weg vom Objekt zum Augenpunkt werden die verschiedenen Wellenlängen des Lichts durch die **Atmosphäre** unterschiedlich stark gedämpft.

⇒ Weiter entfernte Gegenstände erscheinen blauer.

Modellierung: Mische die „eigentliche“ Farbe des Objekts mit einer „weit weg“-Farbe I_w , wobei das Mischverhältnis von der Entfernung Objekt - Augenpunkt abhängt:

$$I = \sigma \cdot I_{\text{Objekt}} + (1 - \sigma) \cdot I_w$$

mit

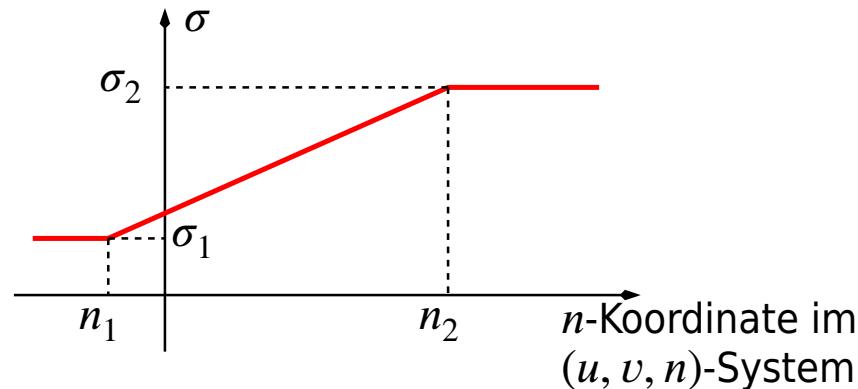
$$\sigma = \sigma \left(\underbrace{\mathbf{P}}_{\text{Punkt auf dem Objekt}}, \underbrace{\mathbf{A}}_{\text{Augenpunkt}} \right)$$

Wahl von I_w und σ :

- I_w = „Blauton“, $\sigma = e^{-\alpha \cdot \|\mathbf{P} - \mathbf{A}\|_2}$ ($\alpha > 0$)

⇒ Rotes Licht wird auf dem Weg ins Auge exponentiell gedämpft (Simulation der Atmosphäre).

- $I_w = \text{"irgendeine Farbe"}$ (z. B. schwarz)

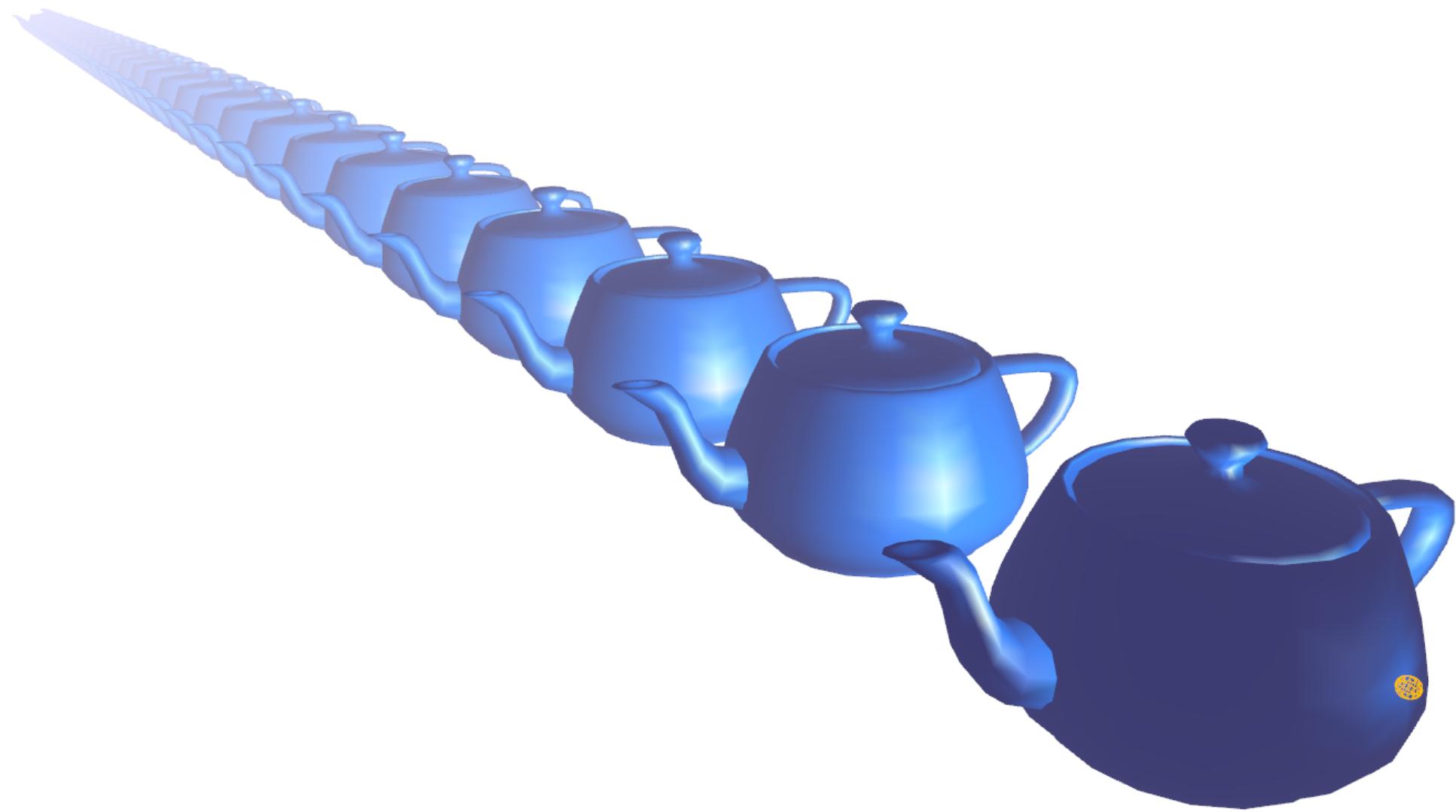


$$\sigma(\mathbf{P}, \mathbf{A}) \equiv \sigma_1 \text{ für } n \leq n_1$$

$$\sigma \text{ linear in } n \quad \text{für } n_1 \leq n \leq n_2$$

$$\sigma(\mathbf{P}, \mathbf{A}) \equiv \sigma_2 \text{ für } n \geq n_2$$

mit $0 \leq \sigma_1 \leq \sigma_2 \leq 1$



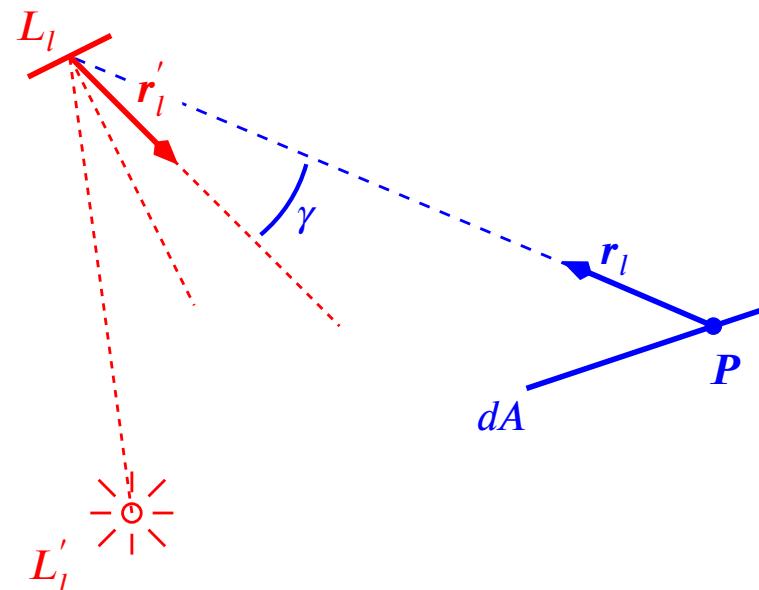
7.2.6 Gerichtete Lichtquellen nach Warn

Motivation: Punktformige Lichtquellen strahlen in alle Richtungen gleichmäßig ab.

Die in der Realität vorkommenden Lichtquellen besitzen aber i. Allg. eine bevorzugte Richtung (z. B. Auto-/Fahrradscheinwerfer: Licht in einem Kegel gebündelt).

Idee: Bei den Objekten ergab die winkelabhängige Reflexion eine bevorzugte Richtung. Verwende diesen Ansatz auch hier.

Modell: Ersetze die Punktlichtquelle durch einen (unendlich kleinen) perfekten Spiegel, der von einer unendlich fernen Lichtquelle L'_l beleuchtet wird.



David R. Warn

⇒ L_l reflektiert am stärksten in die durch das Spiegelgesetz vorgegebene Richtung r'_l .

Liegt L_l von Punkt \mathbf{P} aus in Richtung \mathbf{r}_l , so wird \mathbf{P} von L_l mit der Intensität

$$I_l = \underbrace{\hat{I}_l}_{\text{maximale Intensität in der von } L_l \text{ bevorzugten Richtung}} \cdot \underbrace{(\cos \gamma)^{\mu_l}}_{\substack{\text{Exponent kontrolliert die „Richtcharakteristik“} \\ (\text{Bündelung}) \text{ von } L_l}} = \hat{I}_l \cdot (-\mathbf{r}_l^T \cdot \mathbf{r}'_l)^{\mu_l}$$

beleuchtet.

- $\mu_l = 0$: gleichmäßige Abstrahlung in alle Richtungen
- μ_l „groß“: Mit zunehmendem Winkel γ fällt die Intensität schnell ab (\triangleq starke Bündelung).

Bemerkungen 7.9:

1. L'_l dient nur der Veranschaulichung. Die Lichtquelle wird definiert durch

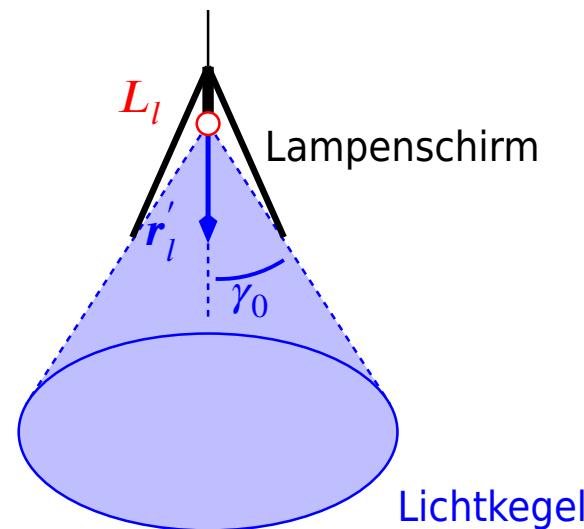
L_l (Ort),
 \hat{I} (maximale Intensität),
 \mathbf{r}'_l (bevorzugte Richtung) und
 μ_l (Bündelung).

2. Für $-\mathbf{r}_l^T \cdot \mathbf{r}'_l < 0$ (d. h. $\gamma > 90^\circ$) setzt man im Fall $\mu_l > 0$ die Intensität $I_l := 0$.

Die Lichtabstrahlung kann leicht noch weiter eingeschränkt werden, z. B. durch:

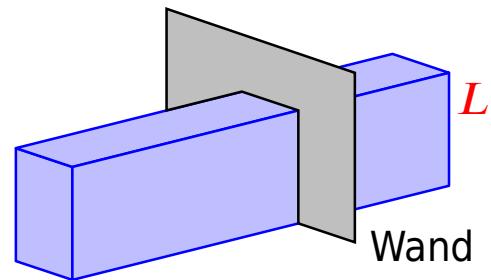
Kegel: Setze $I_l := 0$, falls $\gamma \geq \gamma_0$ (fest).

Damit kann z. B. Licht von Hängelampen simuliert werden.



Klappen: Setze $I_l := 0$, falls $\mathbf{P} \notin [\underline{x}; \bar{x}] \times [\underline{y}; \bar{y}] \times [\underline{z}; \bar{z}]$ mit $-\infty \leq \underline{x} \leq \bar{x} \leq +\infty$ (y, z analog).

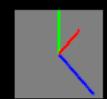
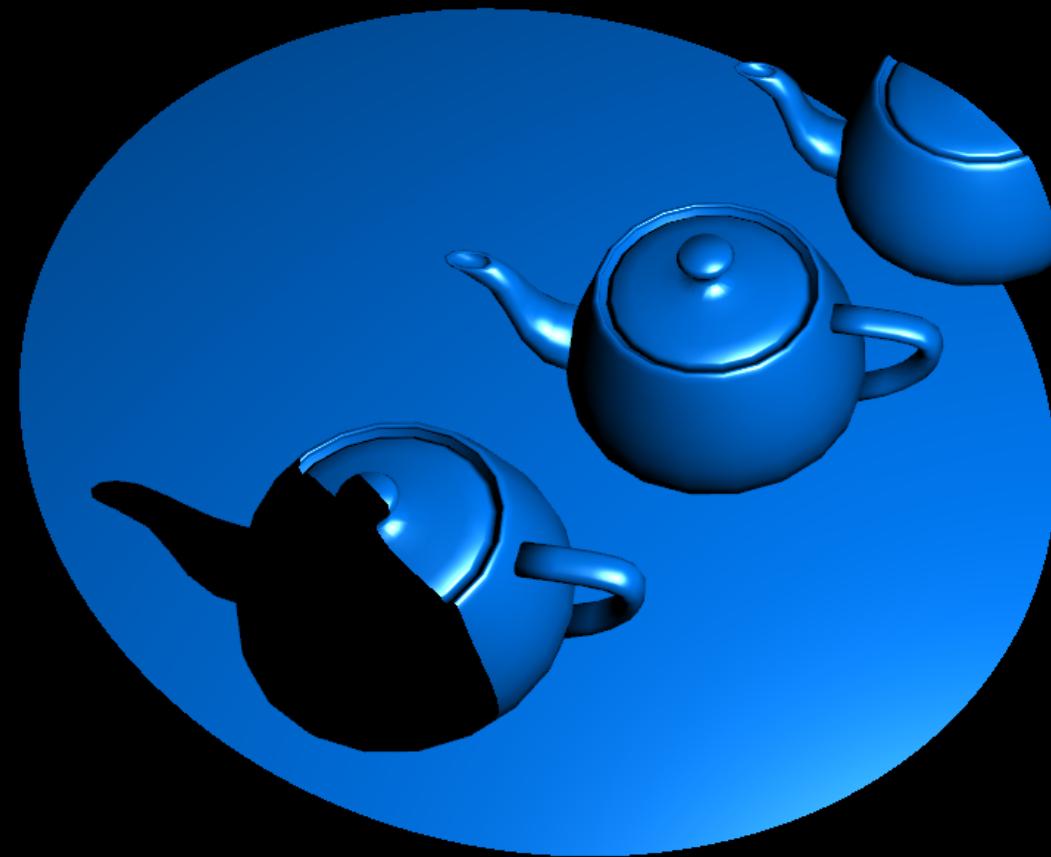
Damit kann z. B. Licht simuliert werden, das durch eine Tür in einen Raum fällt.



- ⊕ bei sehr geringem Aufwand oft relativ gute Ergebnisse
- ⊕ Die Berücksichtigung gerichteter Abstrahlung kann sehr einfach auch noch **nachträglich** in ein Programm eingebaut werden.
- ⊖ stark heuristisch

FPS: 59.94

Bilder/glai/warn-exp0-angle21



FPS: 59.94

Bilder/glai/warn-exp12-angle90



7.3 Lokale Beleuchtungsmodelle

Durch **Beleuchtungsmodelle** werden die Lichtverhältnisse für **einzelne Punkte** der Objekte festgelegt:

- Ursprung des Lichts: Anzahl und Art der Lichtquellen
(ambient, punktförmig ungerichtet/gerichtet, ausgedehnt),
Intensität, Position, ...
- Interaktion des Lichts mit den Objekten: Welche physikalischen Effekte werden modelliert, und wie?
(Reflexion, Brechung, Schatten, ...)

Bei einem **lokalen Beleuchtungsmodell (Modell der Ordnung 0)** werden zur Berechnung der „Farbe“ von Punkt P auf Objekt O_k nur

- die (Material-)Eigenschaften von O_k und
- die Lichtquelle(n) L_l

herangezogen.

- ⊕ geringer Rechenaufwand (für Real-Time-Anwendungen)
- ⊖ **keine** Schatten, keine Lichtbrechung, keine Spiegelung an Objekten, ...

7.3.1 Das „triviale“ Modell

Lichtquellen: Objekte

modellierte Effekte: —

Ansatz:

$$\underbrace{I(\mathbf{P})}_{\begin{array}{l} \text{vom Punkt } \mathbf{P} \text{ aus in Richtung} \\ \text{des Augenpunkts} \\ \text{ausgehende Intensität} \end{array}} = \underbrace{I_k}_{\text{konstanter Wert für Objekt } O_k} \quad (\lambda = R, G, B)$$

Bemerkungen 7.10:

1. extrem einfach (und schnell) zu berechnen
 2. Die Intensität ist für alle zu einem Objekt gehörenden Punkte/Pixel gleich.
-

7.3.2 Das Modell von Bouknight

Lichtquellen: ambientes Licht

1 punktförmige Lichtquelle in beliebiger Position

modellierte Effekte: diffuse Reflexion

Ansatz:

$$I(\mathbf{P}) = \underbrace{I_a \cdot R_{k,a}}_{\text{siehe 7.2.1}} + \underbrace{(\mathbf{n}_P^T \cdot \mathbf{r}_1) \cdot I_1 \cdot R_{k,d}}_{\text{siehe 7.2.2}} \quad (\lambda = R, G, B)$$

W. Jack Bouknight

Bemerkungen 7.11:

1. einfach und schnell zu berechnen
2. Wenn man annimmt, dass das Objekt nicht gekrümmt (Polygon) und die Lichtquelle „weit genug“ entfernt ist, dann variiert die Intensität für die zum Objekt gehörenden Punkte/Pixel nicht wesentlich.

7.3.3 Das Modell von Phong

Lichtquellen: ambientes Licht

mehrere punktförmige Lichtquellen

modellierte Effekte: diffuse Reflexion

winkelabhängige Reflexion

(entfernungsabhängige Dämpfung)

Bùi Tường Phong

* 1942, Hanoi (Hà Nội)

† 1975, USA

Informatiker

Ansatz:

$$I(\mathbf{P}) = \underbrace{I_a \cdot R_{k,a}}_{7.2.1} + \sum_l \underbrace{f_l(\mathbf{P}) \cdot I_l}_{7.2.4} \cdot \left(\underbrace{(\mathbf{n}_P^T \cdot \mathbf{r}_l) \cdot R_{k,d}}_{7.2.2} + \underbrace{(\mathbf{v}^T \cdot \mathbf{s}_l)^{\nu_k} \cdot R_{k,w}}_{7.2.3} \right) \quad (\lambda = \lambda_1, \dots, \lambda_m)$$

Bemerkungen 7.12:

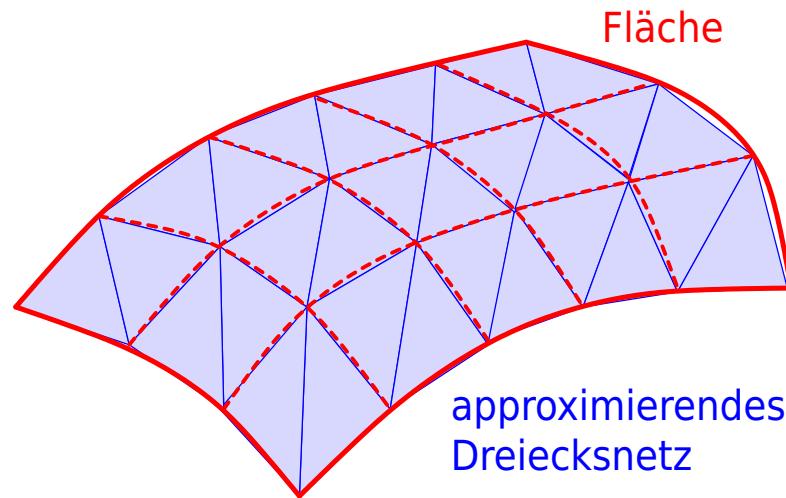
1. Bei Bedarf können Farbverschiebung und gerichtete Lichtquellen leicht hinzugenommen werden.
 2. weitaus mächtiger als das Modell von Bouknight, aber auch wesentlich aufwändiger
-

7.4 Färbungsstrategien für Polygone

Die **Färbungsstrategie** legt fest,

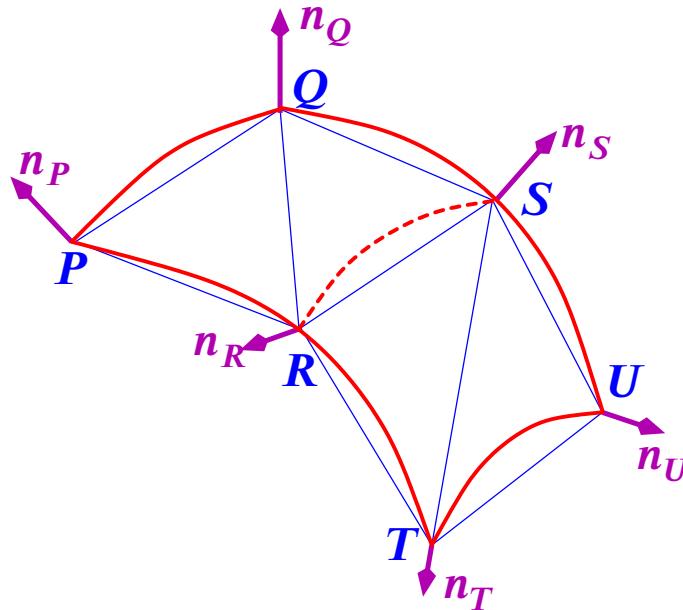
- welches (lokale) Beleuchtungsmodell verwendet wird,
- für welche zu einem Objekt gehörenden Punkte/Pixel das Beleuchtungsmodell angewandt wird und
- wie die Farbe der übrigen Pixel des Objekts bestimmt wird.

Problem: Polygone (insbesondere Drei- und Vierecke) werden oft benutzt, um **gekrümmte Flächen** zu approximieren.



Im Bild sollte wieder der Eindruck einer glatten (gekrümmten) Fläche entstehen.

Ansatz: Ordne jedem Knoten des Polygonnetzes eine Normale zu:



Stimmen diese Normalen mit den Normalen der Fläche in den Knoten überein und verwendet man sie zur Bestimmung der Farbe, so kann u. U. der Eindruck einer glatten Fläche entstehen.

Bemerkung 7.13: Die Farbe innerhalb eines Polygons (\triangleq Stück der Fläche) kann – je nach Krümmung – relativ stark variieren.

7.4.1 Konstante Färbung

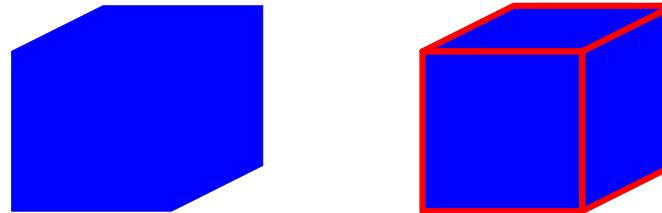
Auswertung: an irgendeinem Punkt P des Polygons

Farbwerte der Pixel: gleich dem einmal für P berechneten Wert

Beleuchtungsmodelle:

1. „trivial“ (7.3.1)

- ⊖ Aus der Farbe kann kein Rückschluss auf die räumliche Orientierung des Polygons gezogen werden.
- ⊖ Werden alle Polygone mit derselben Farbe gezeichnet, so ist nur noch der Umriss ihrer Vereinigung erkennbar.



⇒ Die Ränder der Polygone sollten mit einer anderen Farbe gezeichnet werden.

Bemerkung 7.14: Diese Technik wird oft in Verbindung mit Painter's Algorithm oder dem z-Puffer-Algorithmus verwendet, wenn das Entfernen unsichtbarer Strecken(teile) wichtiger ist als „informative“ Färbung.

Beispiel: Füllfarbe = Hintergrundfarbe

Randfarbe = eigentliche Zeichenfarbe

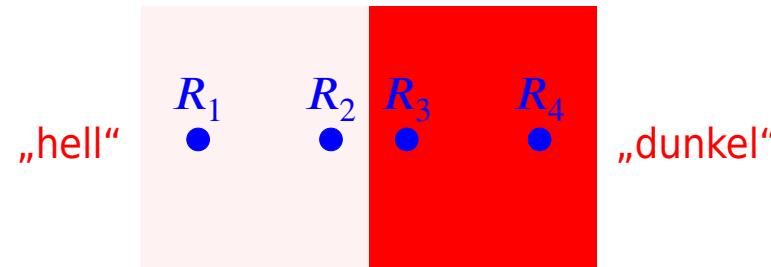
⇒ Drahtmodelldarstellung ohne verdeckte Linien

2. Bouknight-Modell (7.3.2)

- ⊕ Die Helligkeit vermittelt einen groben Eindruck von der räumlichen Orientierung der Polygone.
- ⊖ Wenn die Polygone eine gekrümmte Fläche approximieren, dann wirken die **unstetigen Farbübergänge** an den Polygongrenzen störend, besonders durch den

Mach-Band-Effekt: physiologischer Effekt („optische Täuschung“):

Wird ein (Licht-)Rezeptor im Auge angeregt, so inhibiert er gleichzeitig die übrigen Rezeptoren in seiner Nachbarschaft (umso stärker, je näher sie bei ihm liegen).



R_1 und R_2 empfangen gleiche Lichtintensität, aber R_1 wird stärker inhibiert.

⇒ R_2 meldet mehr Licht als R_1 ,

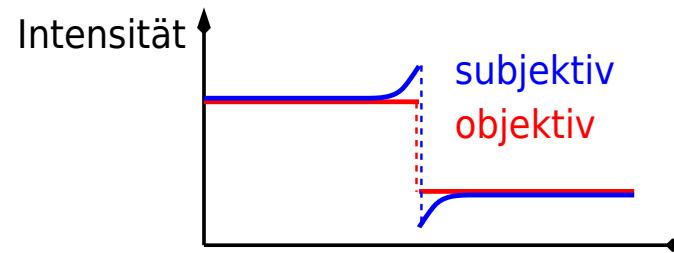
Ernst Waldfried Josef Wenzel Mach
* 1838, Chirlitz (Chrlice, Kaisertum Österreich, heute zu Brünn [Brno], Tschechien)
† 1916, Vaterstetten (nahe München)

R_3 meldet weniger Licht als R_4 (analog)

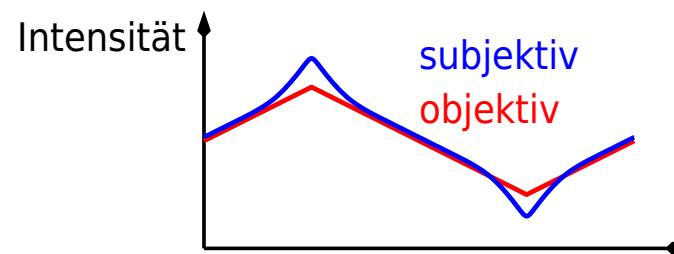


Physiker
Foto: H. F. Jütte

Quelle: https://commons.wikimedia.org/wiki/File:Ernst_Mach_01.jpg



Dieser Effekt tritt auch bei stetigen, nicht differenzierbaren Intensitätsverläufen auf:



Beispiel:



Bemerkung 7.15: Diese Technik liefert eine adäquate Darstellung, wenn

- die Polygone selbst die darzustellenden Objekte sind (und nicht eine Fläche approximieren) und
 - die Lichtquelle „weit genug“ entfernt ist.
-
-

Bemerkungen 7.16:

1. Konstante Färbung ist sehr schnell durchführbar, da
 - das Lichtmodell nur einmal angewandt wird und
 - alle Pixel des Objekts gleich gefärbt werden, d. h. ein effizienter Füllalgorithmus einsetzbar ist.
 2. Evtl. vorhandene „Eckennormalen“ werden i. Allg. ignoriert; im Beleuchtungsmodell wird die Normale zur Polygonebene verwendet.
-

7.4.2 Farbwertinterpolation (Gouraud-Shading)

Henri Gouraud
* 1944, Frankreich
Informatiker

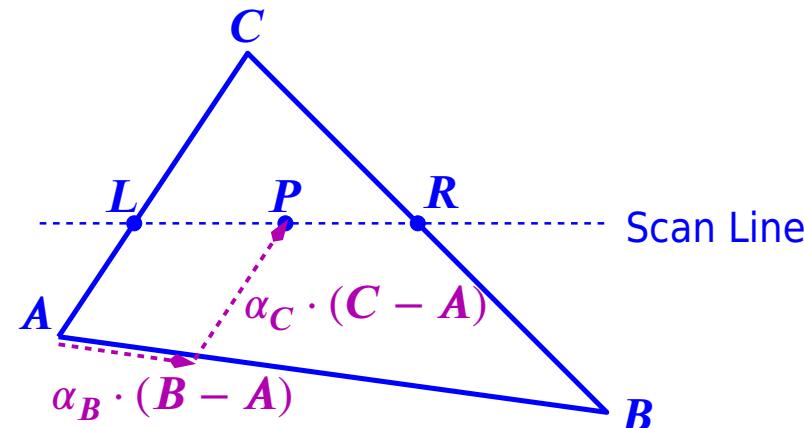
Beleuchtungsmodell: vorwiegend Phong-Modell (7.3.3)

Auswertung: an den Ecken des Polygons mit den vorgegebenen Eckennormalen

Farbwerte der Pixel: Interpolation der Farbwerte der Ecken

Bemerkung 7.17: Gouraud- und Phong-Shading werden in der Praxis meist auf Dreiecke angewandt; Polygone mit mehr Ecken werden vorher **trianguliert** (in Dreiecke zerlegt).

Gouraud- und Phong-Shading werden bei der Scan Conversion **inkrementell** durchgeführt:



- $I(\mathbf{L})$ ergibt sich durch lineare Interpolation von $I(\mathbf{C})$ und $I(\mathbf{A}), I(\mathbf{R})$ aus $I(\mathbf{C})$ und $I(\mathbf{B})$.
 - ⇒ $I(\mathbf{L})$ und $I(\mathbf{R})$ ändern sich beim Übergang zur nächsten Scan Line um feste Beträge $\Delta I(\mathbf{L})$ bzw. $\Delta I(\mathbf{R})$.
 - $I(\mathbf{P})$ ergibt sich durch lineare Interpolation von $I(\mathbf{L})$ und $I(\mathbf{R})$.
 - ⇒ $I(\mathbf{P})$ ändert sich beim Übergang zum Nachbarpixel um einen festen (von der Scan Line abhängigen) Betrag $\Delta I(\mathbf{P})$.
-

Bemerkung 7.18: Jeder Punkt \mathbf{P} im Dreieck besitzt eine eindeutige Darstellung

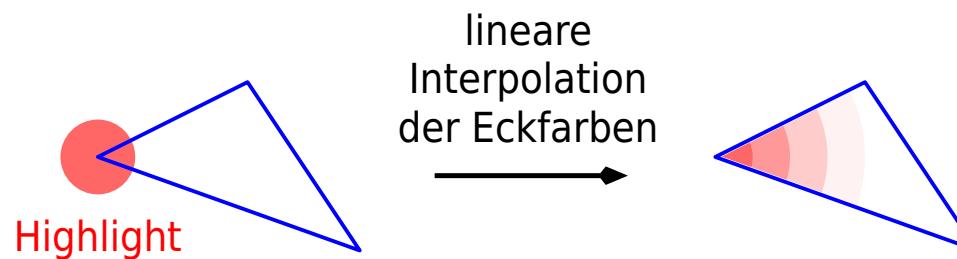
$$\begin{aligned}\mathbf{P} &= \mathbf{A} + \alpha_{\mathbf{B}} \cdot (\mathbf{B} - \mathbf{A}) + \alpha_{\mathbf{C}} \cdot (\mathbf{C} - \mathbf{A}) \\ &= \underbrace{(1 - \alpha_{\mathbf{B}} - \alpha_{\mathbf{C}})}_{=: \alpha_{\mathbf{A}}} \cdot \mathbf{A} + \alpha_{\mathbf{B}} \cdot \mathbf{B} + \alpha_{\mathbf{C}} \cdot \mathbf{C}\end{aligned}$$

mit $\alpha_{\mathbf{A}}, \alpha_{\mathbf{B}}, \alpha_{\mathbf{C}} \in [0; 1]$ und $\alpha_{\mathbf{A}} + \alpha_{\mathbf{B}} + \alpha_{\mathbf{C}} = 1$ (**baryzentrische Koordinaten** von \mathbf{P}).

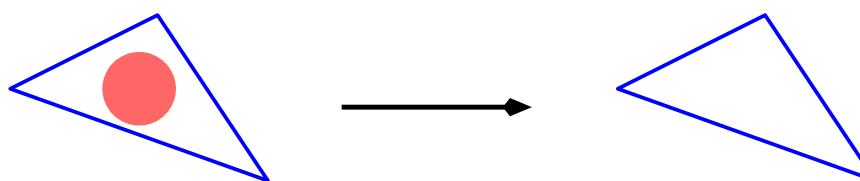
Bei Gouraud-Shading setzt man dann

$$I(\mathbf{P}) = \alpha_{\mathbf{A}} \cdot I(\mathbf{A}) + \alpha_{\mathbf{B}} \cdot I(\mathbf{B}) + \alpha_{\mathbf{C}} \cdot I(\mathbf{C}).$$

- ⊕ relativ geringer Aufwand (drei Auswertungen des Beleuchtungsmodells und inkrementelle Interpolation)
- ⊖ Die Farbübergänge an den Dreiecksgrenzen werden zwar stetig, aber nicht differenzierbar.
 - ⇒ Mach-Band-Effekt
 - (aber schwächer als bei konstanter Färbung)
- ⊖ Glanzlichter (**Highlights**, hervorgerufen durch die winkelabhängige Reflexion) werden nicht korrekt wiedergegeben:
 - Scharfe Highlights (v_k groß, 7.2.3) in den Dreiecksecken werden „verschmiert“.

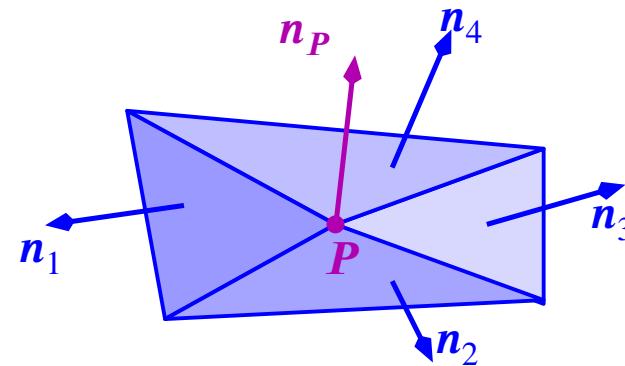


- Highlights, die keine Dreiecksecke treffen, werden gar nicht dargestellt.



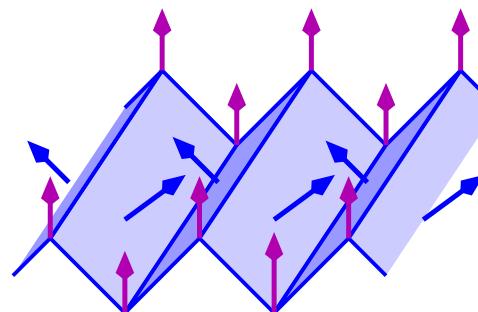
Bemerkungen 7.19:

1. Sind keine **Eckennormalen** vorgegeben, so kann man meist durch Mitteln der **Ebenennormalen** aller an eine Ecke grenzenden Polygone „vernünftige“ Vektoren erhalten:



$$\mathbf{n}_P = \frac{\tilde{\mathbf{n}}_P}{\|\tilde{\mathbf{n}}_P\|_2} \quad \text{mit} \quad \tilde{\mathbf{n}}_P = \frac{1}{4} \cdot (\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4)$$

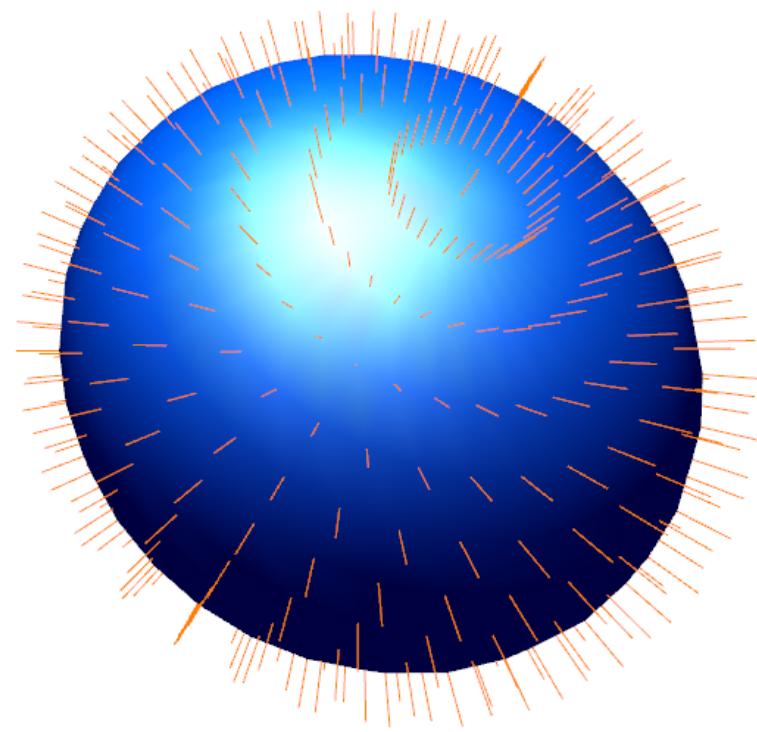
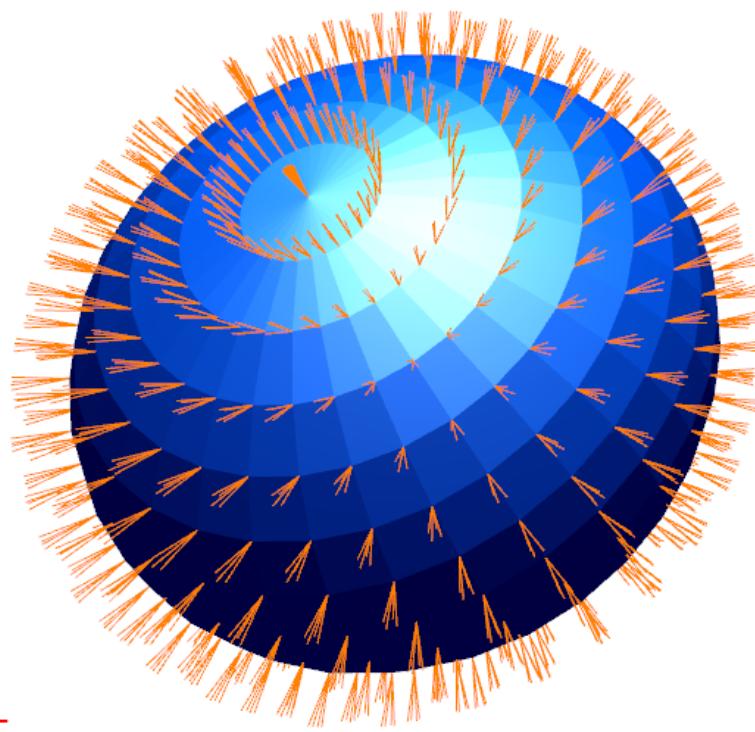
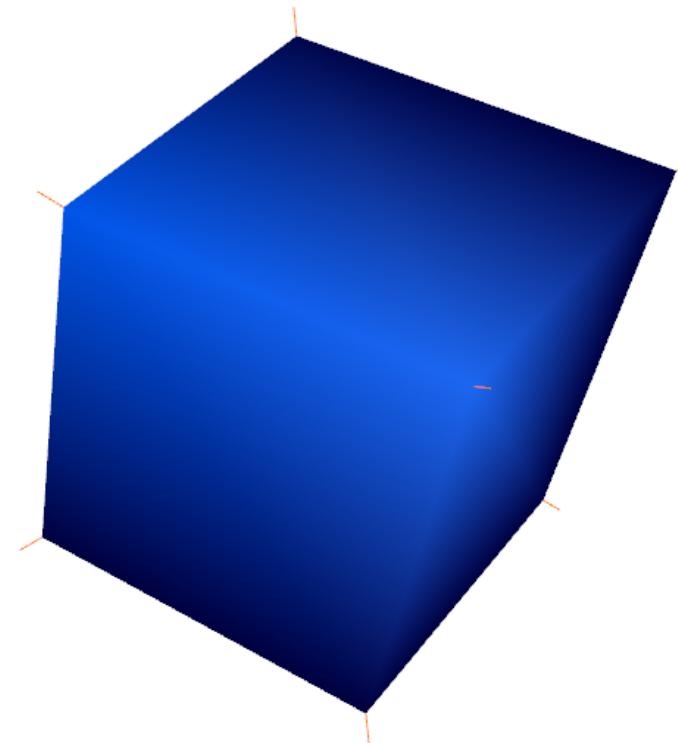
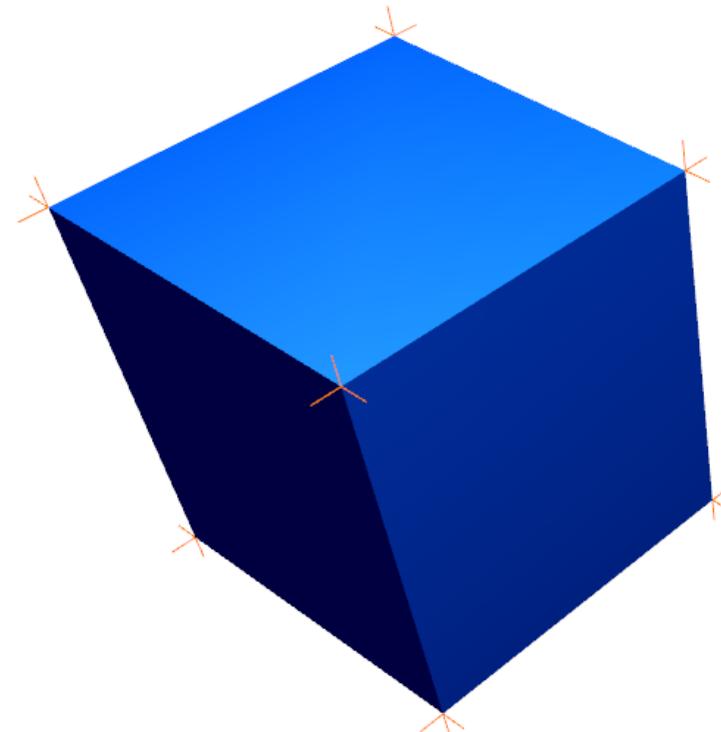
Das klappt aber nicht immer:



Alle gemittelten Eckennormalen sind parallel!

FPS: 59.94

Bilder/glai/flat2



2. Gouraud-Shading wird i. Allg. Hardware-unterstützt.

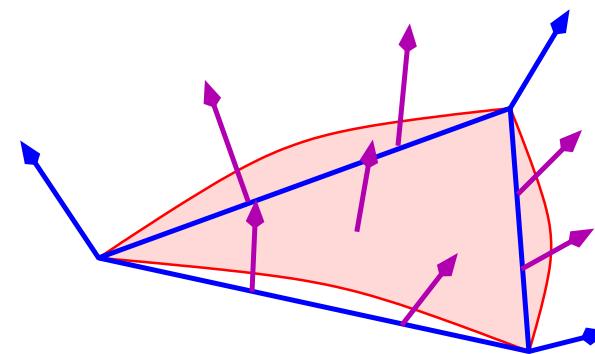
⇒ sehr schnell (viele Millionen Dreiecke/s)

7.4.3 Normaleninterpolation (Phong-Shading)

Beleuchtungsmodell: Phong-Modell

Auswertung: bei jedem zum Polygon (i. Allg. Dreieck) gehörenden Pixel P , wobei die Normale n_P linear aus den Eckennormalen interpoliert wird

Farbwerte der Pixel: gemäß Beleuchtungsmodell



Die interpolierten Normalen sind meist gute Näherungen für die Normalen der approximierten Fläche.

Bemerkung 7.20: Die Normalenrichtung wird (wie die Farbe beim Gouraud-Shading) inkrementell interpoliert; die so bestimmten Vektoren müssen noch **normiert** werden.

- ⊕ Der Farbverlauf ist auch über die Dreiecksgrenzen hinweg glatt.
- ⊕ verhältnismäßig gute Wiedergabe der Krümmung der Fläche, auch wenn die Dreiecke nicht allzu klein sind
- ⊕ Highlights werden (nahezu) korrekt dargestellt.
- ⊖ Der Umriss der Fläche ist ein Polygonzug mit meist deutlich sichtbaren Knicken.
einzige Abhilfe: Polygone kleiner machen
 - ⇒ Anzahl der Polygone steigt
- ⊖ ziemlich aufwändig: Auswertung des Beleuchtungsmodells für jedes Pixel des Polygons
(Dies ist i. Allg. nicht inkrementell möglich.)
dennoch Hardware-Unterstützung

7.5 Modellierung optischer Effekte (II)

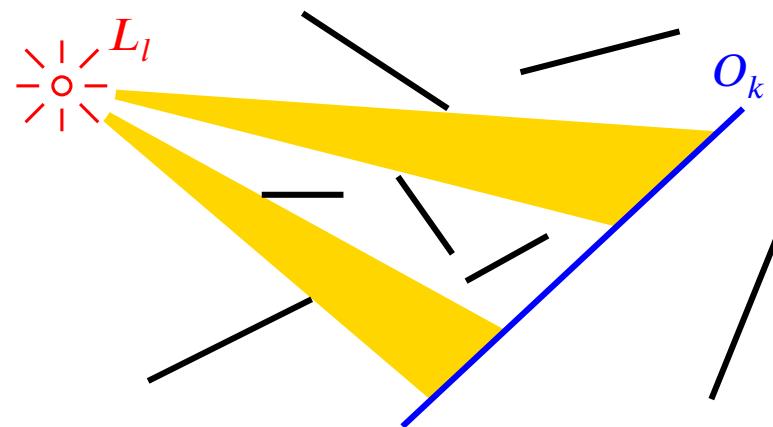
Ziel: Simulation von Effekten, die in den bisherigen **lokalen** Beleuchtungsmodellen noch nicht berücksichtigt sind, **ohne** die Beleuchtungsmodelle „wesentlich“ zu ändern

Beispiele 7.21:

1. **Schattenwurf:** Bevor das Licht der Lichtquelle auf das betrachtete Objekt O_k fällt, hat es bereits mit einem anderen Objekt O_k' , Kontakt.
⇒ Entweder muss ein globales Beleuchtungsmodell verwendet werden, oder der Einfluss der anderen Objekte muss **außerhalb des Beleuchtungsmodells** berücksichtigt werden.
 2. **Oberflächenstruktur:** Die meisten realen Oberflächen von Körpern sind nicht völlig gleichmäßig gefärbt (Marmorierung, Stoffmuster, Holzmaserung, ...).
-

7.5.1 Schatten

Beobachtung:



Genau die Punkte P eines Objekts O_k erhalten von der Lichtquelle L_l Licht, die von L_l aus **sichtbar**, d. h. nicht durch andere Objekte verdeckt, sind.

⇒ Die Bestimmung der durch L_l beleuchteten Teile von O_k entspricht einer **Sichtbarkeitsanalyse von L_l aus**

(vgl. Kapitel 6).

Der analytische Ansatz

Bestimme die (bzgl. der gewählten Projektion) später im Bild sichtbaren Teile $O_i^{(0)}$ der Objekte O_k :

$$\{O_k\} \mapsto \left\{ O_i^{(0)} \right\} \quad \text{mit} \quad \bigcup_i O_i^{(0)} \subseteq \bigcup_k O_k \\ =: \overbrace{O^{(0)}}^{\text{=: } O}$$

für $l = 1, 2, \dots, l_{\max}$ // alle Lichtquellen

bestimme für jedes $O_i^{(l-1)}$ die von L_l aus entweder vollständig sichtbaren oder vollständig unsichtbaren Teile $O_j^{(l)}$:

$$\left\{ O_i^{(l-1)} \right\} \mapsto \left\{ O_j^{(l)} \right\} \text{ mit } \underbrace{\bigcup_j O_j^{(l)}}_{=: O^{(l)}} = \underbrace{\bigcup_i O_i^{(l-1)}}_{=: O^{(l-1)}}$$

färbe die $O_i^{(l_{\max})}$ entsprechend einer geeigneten Strategie

Bemerkungen 7.22:

1. Die Beschreibung der jeweils sichtbaren/unsichtbaren Teile muss in Form von Polygonen geliefert werden, nicht als einzelne Strecken(teile) wie in Abschnitt 6.2.
2. Berücksichtigung der „richtigen“ Lichtquellen bei der Färbung von $O_i^{(l_{\max})}$:
 - Während der Schattenanalyse wird in einem Bitvektor der Länge l_{\max} für jede Lichtquelle L_l festgehalten, ob $O_i^{(\cdot)}$ von L_l Licht erhält

oder

- Färbung und Schattenanalyse „verzahnen“:

**zuerst nur die $O_i^{(0)}$ bestimmen und mit der ambienten Komponente vorbelegen
für $l = 1, 2, \dots, l_{\max}$**

die von L_l beleuchteten Teile von $O_i^{(0)}$ bestimmen und die Intensität der entsprechenden Pixel um den zu L_l gehörenden diffusen und winkelabhängigen Beitrag erhöhen

- ⊕ wesentlich geringerer Verwaltungsaufwand und geringerer Speicherbedarf (weniger Objekte)
 - ⊖ Objekte werden u. U. mehrmals Scan-converted und gefärbt.
-

Der z -Puffer-Ansatz

Idee: Die Sichtbarkeitsinformation bzgl. L_l kann genauso in einem Tiefenpuffer gespeichert werden wie die Sichtbarkeit vom Auge aus.

für $l = 1, 2, \dots, l_{\max}$

führe für alle Objekte O_k Scan Conversion bzgl. L_l durch und speichere die zu jedem „Pixel“ gehörige Entfernung in einem „ z -Puffer“ Z_l

für alle Objekte O_k

führe für O_k Scan Conversion bzgl. der eigentlichen Projektion durch (mit z -Puffer)

für alle sichtbaren Pixel des Objekts

bestimme einen zum Pixel gehörenden Punkt P auf dem Objekt

für $l = 1, 2, \dots, l_{\max}$

bestimme, in welche Zelle $Z_l(i, j)$ von Z_l der Punkt P fällt

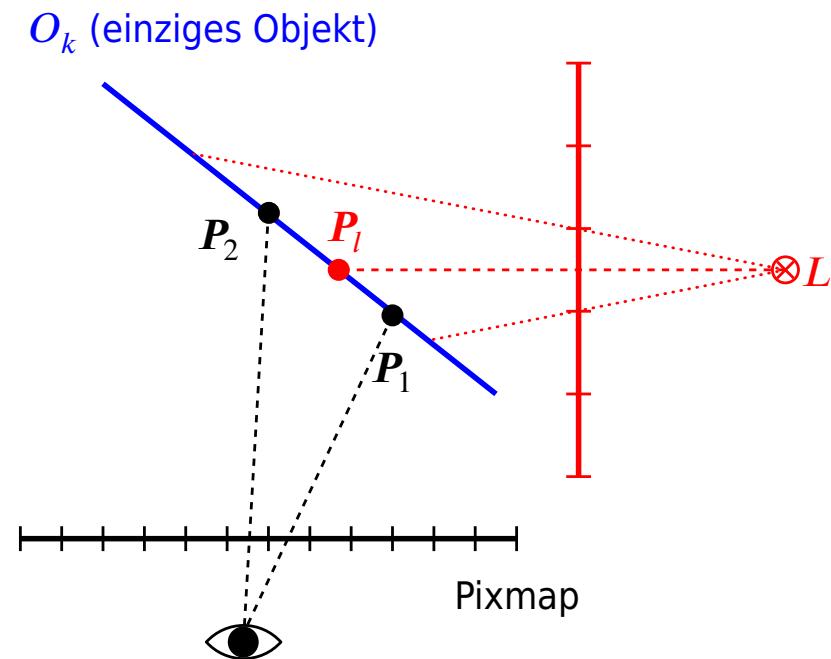
wenn $\|P - L_l\|_2 \leq Z_l(i, j)$ // kein Objekt zwischen P und L_l

berücksichte L_l bei der Färbung des Pixels

Bemerkungen 7.23:

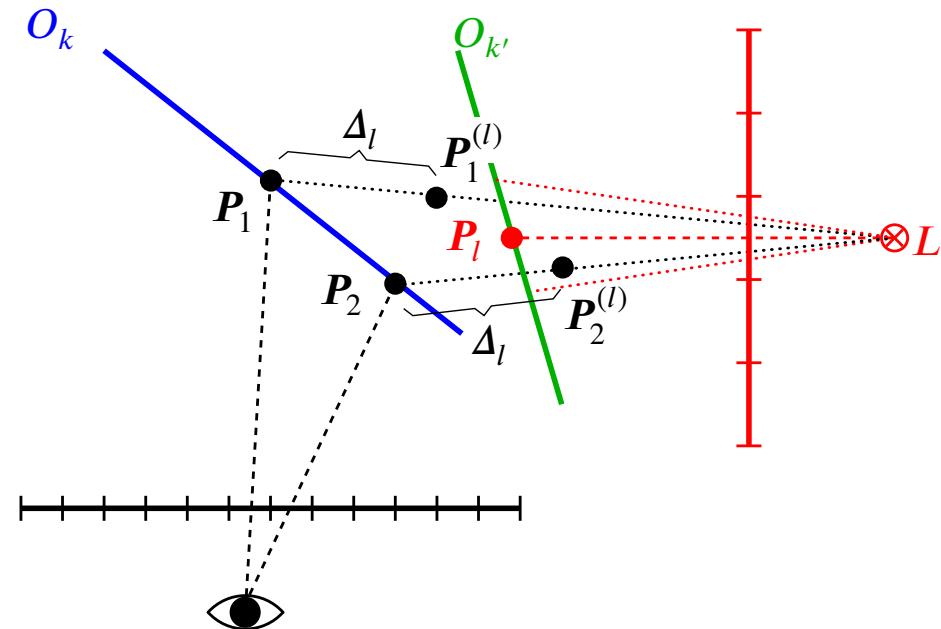
1. Die „Auflösung“ (Anzahl der Zellen, Anzahl der Bits pro Zelle) kann für jeden z -Puffer Z_l frei vorgegeben werden.

2. Der für das Pixel gewählte Punkt P auf dem Objekt stimmt i. Allg. nicht mit dem „Referenzpunkt“ der Zelle $Z_l(i, j)$ überein.



Folge: Teile eines Objekts liegen im Schatten **dieselben** Objekts.

Gegenmaßnahme: Verschiebe den Punkt \mathbf{P} etwas in Richtung der Lichtquelle
(d. h. „korrigiere“ $\|\mathbf{P} - \mathbf{L}_l\|_2$ nach unten): $\mathbf{P} \rightsquigarrow \mathbf{P}^{(l)}$

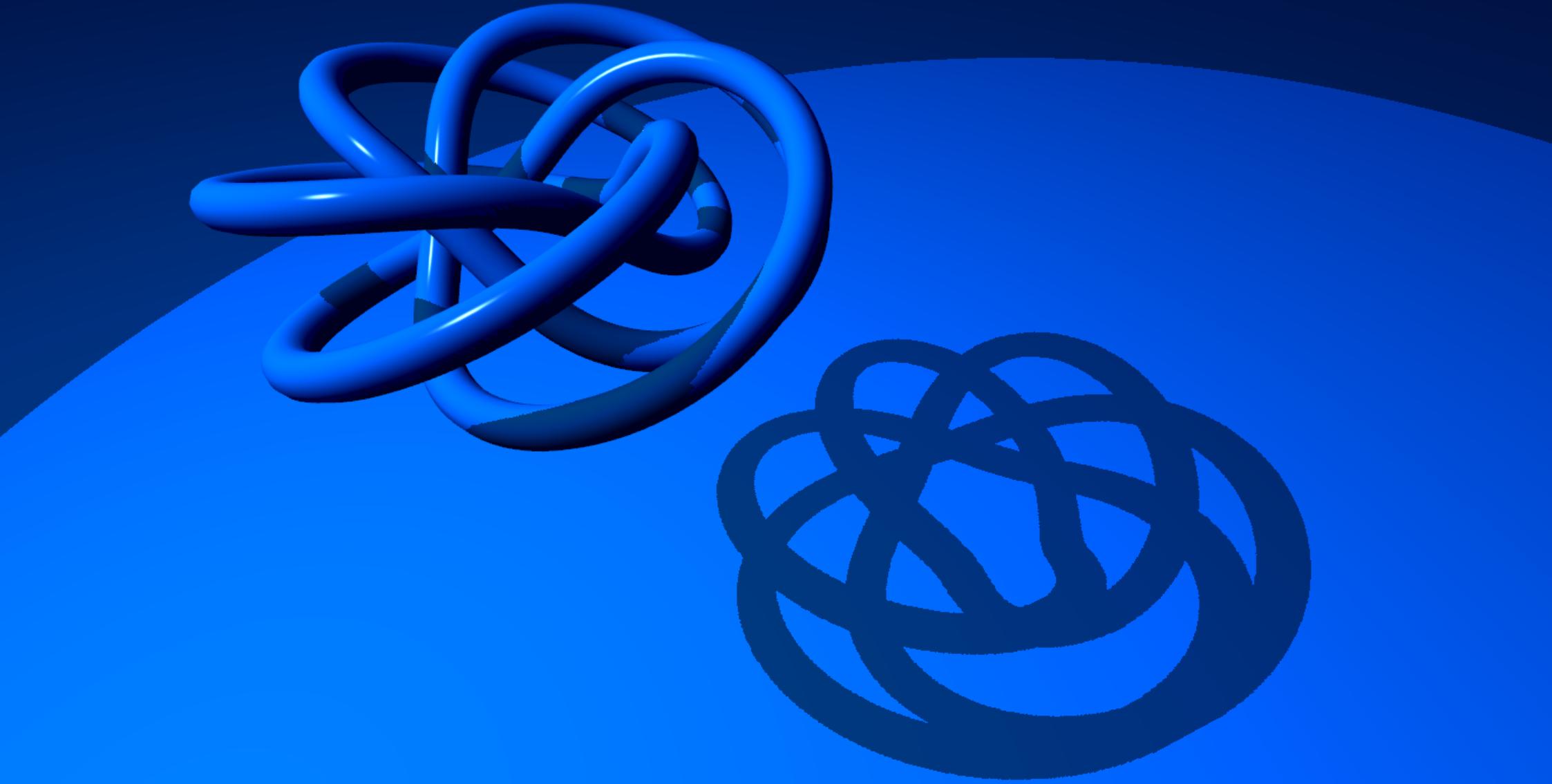


Verschiebt man alle Punkte gleichmäßig um Δ_l zu L_l hin, so können einige davon (hier: \mathbf{P}_2) irrtümlich aus dem Schatten eines anderen Objekts O_k , fallen.

⇒ Diese Technik liefert manchmal falsche Ergebnisse.

FPS: 59.94

Bilder/glai/shadowmap



7.5.2 Oberflächenstruktur (Textur)

Annahme: Das betrachtete Objekt besitzt eine **Parametrisierung**

$$O = \left\{ \mathbf{P} = \mathbf{P}(s, t) = \begin{pmatrix} x(s, t) \\ y(s, t) \\ z(s, t) \end{pmatrix} \mid (s, t) \in D \right\}$$

mit einem geeigneten Parameterbereich D .

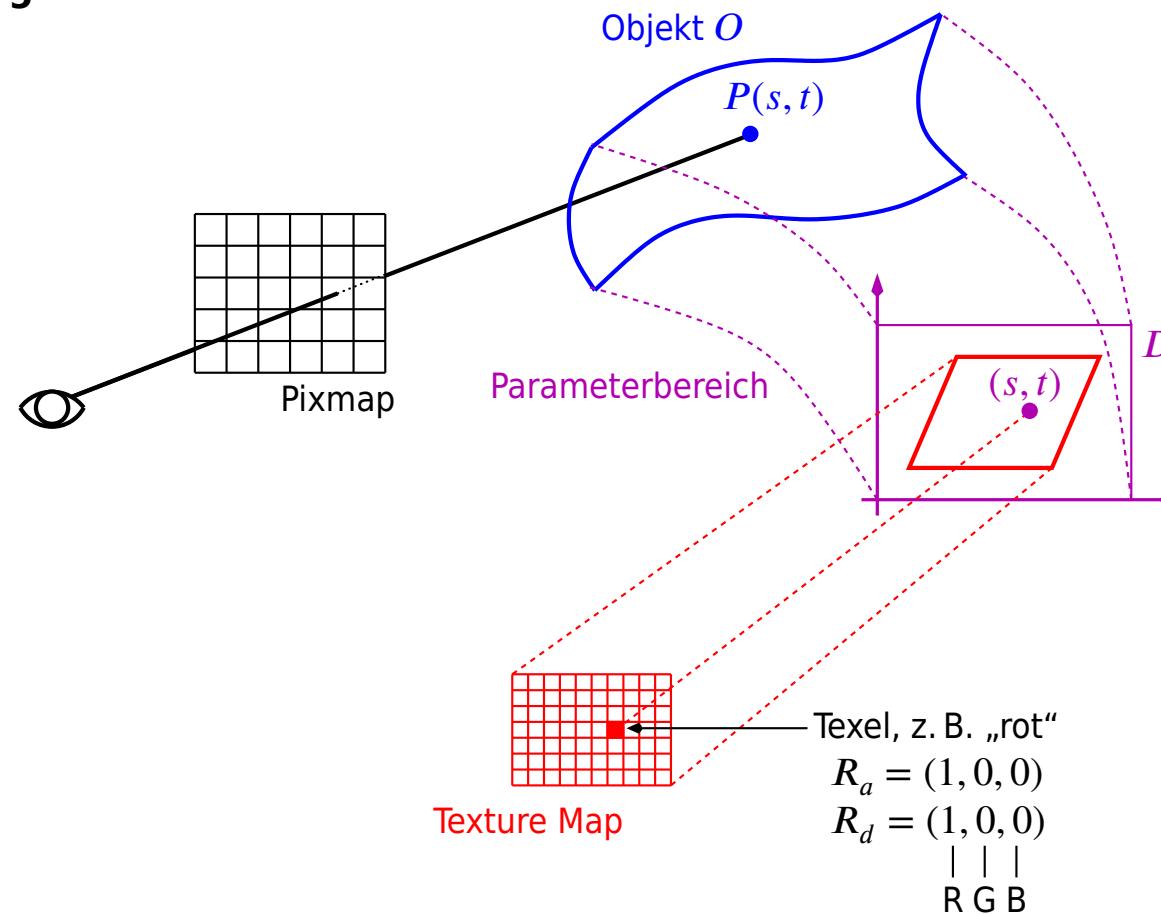
Flat Mapping

Ansatz: Die Textur wird durch ein Muster (**Texture Map**) definiert, das dann (ggf. zyklisch) auf das Objekt abgebildet wird.

Die einzelnen Elemente des Musters (Texture Elements, **Texels**) enthalten Angaben über die Eigenschaften des Objekts an der entsprechenden Stelle, z. B.

- Reflexionskoeffizienten für ambientes und diffuses Licht bei den betrachteten Wellenlängen.

einfachste Realisierung:



- Bestimme den zum Pixel gehörigen Punkt \mathbf{P} auf dem Objekt.
- Bestimme die zugehörigen Parameterwerte s und t .
- Bestimme mit einer (zum Objekt gehörenden) geeigneten Abbildung das zu (s, t) gehörige Texel in einer Texture Map.
- Verwende die dort gespeicherten Koeffizienten zur Färbung des Pixels.

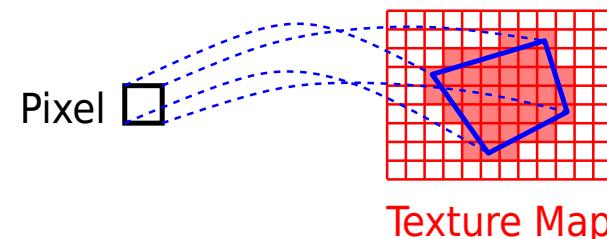
Bemerkungen 7.24:

1. Texture Maps können beispielsweise generiert werden

- „von Hand“,
 - ⊖ **Aufwand, (Speicherplatz)**
- aus vorhandenen (z. B. gescannten) Bildern,
 - ⊖ **Speicherplatz**
- prozedural (Vorschrift zur Berechnung der Einträge der Texture Map), z. B. fraktal.
 - ⊖ **Rechenzeit beim Färben**

2. Bessere Ergebnisse erhält man durch folgendes Vorgehen:

- Bilde die vier **Ecken des Pixels** (wie oben) in die Texture Map ab. Dadurch wird ein Viereck in der Texture Map definiert.
- Verwende zum Färben des Pixels ein gewichtetes Mittel aller zu diesem Viereck gehörigen Texel.



FPS: 59.98

Bilder/glai/tex-marble

Quelle: Holger Arndt, Software: Martin Galgon, Textur: NASA's Earth Observatory, <https://earthobservatory.nasa.gov/Features/BlueMarble/>,



Bump Mapping

Motivation: mit Flat Mapping erzeugte Textur wirkt oft „aufgemalt“, plastische Wirkung kann nicht erzeugt werden

Grund: Die Texture Map kann ein Relief nur bei einer bestimmten (**vorab festgelegten**) Stellung der Lichtquelle zur Fläche wiedergeben.

In der später dargestellten Szene wird das Objekt aber i. Allg. aus einer anderen Richtung beleuchtet.

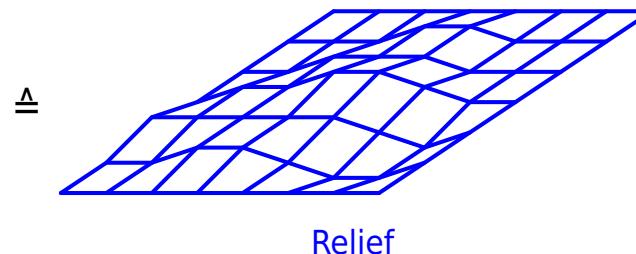
⇒ Relief-Struktur wirkt „unecht“

(besonders deutlich, wenn dieselbe Textur auf unterschiedlich orientierte Polygone abgebildet wird, und bei gekrümmten Flächen)

Folgerung: Um Relief-Wirkung zu erzielen, muss die **Normale** des Objekts beeinflusst werden.

Ansatz: Verwende (an Stelle der oder zusätzlich zur Texture Map) eine **Bump Map**. Jeder Eintrag der Bump Map gibt an, wie weit der entsprechende Punkt des Reliefs von der (glatten) Fläche des Objekts entfernt ist.

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

Bump Map β 

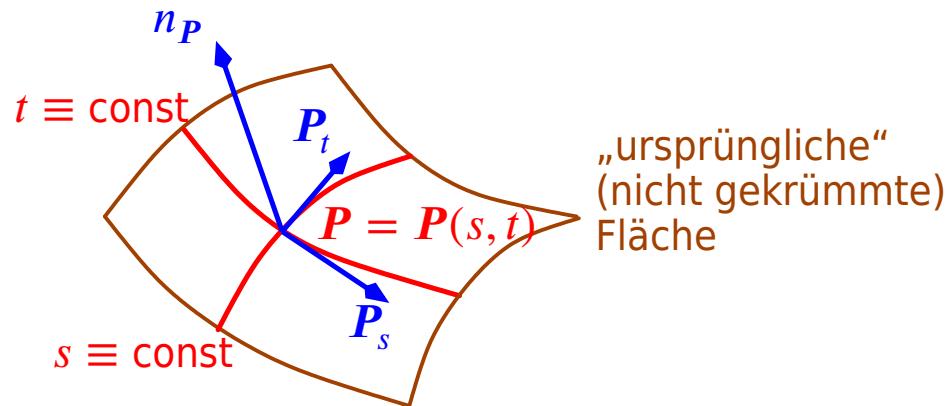
Der Punkt \mathbf{P} der Fläche wird also in den Punkt

$$\tilde{\mathbf{P}} = \mathbf{P} + \underbrace{\beta(\mathbf{P})}_{\text{der zu } \mathbf{P} \text{ gehörige Wert der} \\ (\text{stetig bzw. glatt} \\ \text{fortgesetzten}) \text{ Bump Map}} \cdot \underbrace{\tilde{\mathbf{n}}_{\mathbf{P}}}_{\text{Normalenvektor der Länge 1} \\ \text{an die Fläche im Punkt } \mathbf{P}}$$

verschoben.

Problem: In die Beleuchtungsformel geht die **Normale** der durch die Bump Map „gestörten“ Fläche ein.

Wie kann diese berechnet werden?



Die beiden Vektoren

$$\mathbf{P}_s := \frac{\partial \mathbf{P}}{\partial s}(s, t) \quad \text{und} \quad \mathbf{P}_t := \frac{\partial \mathbf{P}}{\partial t}(s, t)$$

spannen die Tangentialebene im Punkt \mathbf{P} an die Fläche auf. Also ist

$$\mathbf{n}_{\mathbf{P}} := \mathbf{P}_s \times \mathbf{P}_t$$

ein (nicht normierter) Normalenvektor an die ursprüngliche Fläche im Punkt \mathbf{P} .

$\hat{\mathbf{n}}_{\mathbf{P}}$ erhält man durch Normieren von $\mathbf{n}_{\mathbf{P}}$.

Analog ergibt sich eine Normale an die gestörte Fläche durch

$$\begin{aligned}
 \textcolor{orange}{n}_{\tilde{\mathbf{P}}} &= \frac{\partial \tilde{\mathbf{P}}}{\partial s}(s, t) \times \frac{\partial \tilde{\mathbf{P}}}{\partial t}(s, t) \\
 &= \frac{\partial}{\partial s}(\mathbf{P} + \beta(\mathbf{P}) \cdot \hat{\mathbf{n}}_{\mathbf{P}}) \times \frac{\partial}{\partial t}(\mathbf{P} + \beta(\mathbf{P}) \cdot \hat{\mathbf{n}}_{\mathbf{P}}) \\
 &\quad (\text{Vereinfachung: } \hat{\mathbf{n}}_{\mathbf{P}} = \hat{\mathbf{n}}_{\mathbf{P}}(s, t) \text{ wird in der Nähe von } \mathbf{P} \text{ als konstant betrachtet}) \\
 &\approx \left(\mathbf{P}_s + \frac{\partial \beta}{\partial s}(\mathbf{P}) \cdot \hat{\mathbf{n}}_{\mathbf{P}} \right) \times \left(\mathbf{P}_t + \frac{\partial \beta}{\partial t}(\mathbf{P}) \cdot \hat{\mathbf{n}}_{\mathbf{P}} \right) \\
 &= \mathbf{P}_s \times \mathbf{P}_t + \mathbf{P}_s \times (\beta_t \cdot \hat{\mathbf{n}}_{\mathbf{P}}) + (\beta_s \cdot \hat{\mathbf{n}}_{\mathbf{P}}) \times \mathbf{P}_t + (\beta_s \cdot \hat{\mathbf{n}}_{\mathbf{P}}) \times (\beta_t \cdot \hat{\mathbf{n}}_{\mathbf{P}}) \\
 &\quad \text{mit } \textcolor{orange}{\beta}_s := \frac{\partial \beta}{\partial s}(\mathbf{P}) \quad \text{und } \textcolor{orange}{\beta}_t := \frac{\partial \beta}{\partial t}(\mathbf{P}) \\
 &= \mathbf{n}_{\mathbf{P}} + \beta_t \cdot (\mathbf{P}_s \times \hat{\mathbf{n}}_{\mathbf{P}}) + \beta_s \cdot (\hat{\mathbf{n}}_{\mathbf{P}} \times \mathbf{P}_t) + \beta_s \beta_t \cdot \underbrace{(\hat{\mathbf{n}}_{\mathbf{P}} \times \hat{\mathbf{n}}_{\mathbf{P}})}_{= \mathbf{0}} \\
 &= \mathbf{n}_{\mathbf{P}} + \beta_t \cdot (\mathbf{P}_s \times \hat{\mathbf{n}}_{\mathbf{P}}) + \beta_s \cdot (\hat{\mathbf{n}}_{\mathbf{P}} \times \mathbf{P}_t) .
 \end{aligned}$$

(Dieser Vektor muss noch normiert werden.)

Bemerkung 7.25: Die Bump Map ist i. Allg. in einem anderen Parameterbereich $(u, v) \in \tilde{D}$ definiert als die Fläche.

Die „Projektion“ des Reliefs auf die Fläche wird durch eine Zuordnung

$$\begin{aligned}\varphi : D &\rightarrow \tilde{D} \\ (s, t) &\mapsto (u, v)\end{aligned}$$

definiert. Es ist also

$$\beta(\mathbf{P}) = \beta(u, v) = \beta(\varphi(s, t))$$

und damit

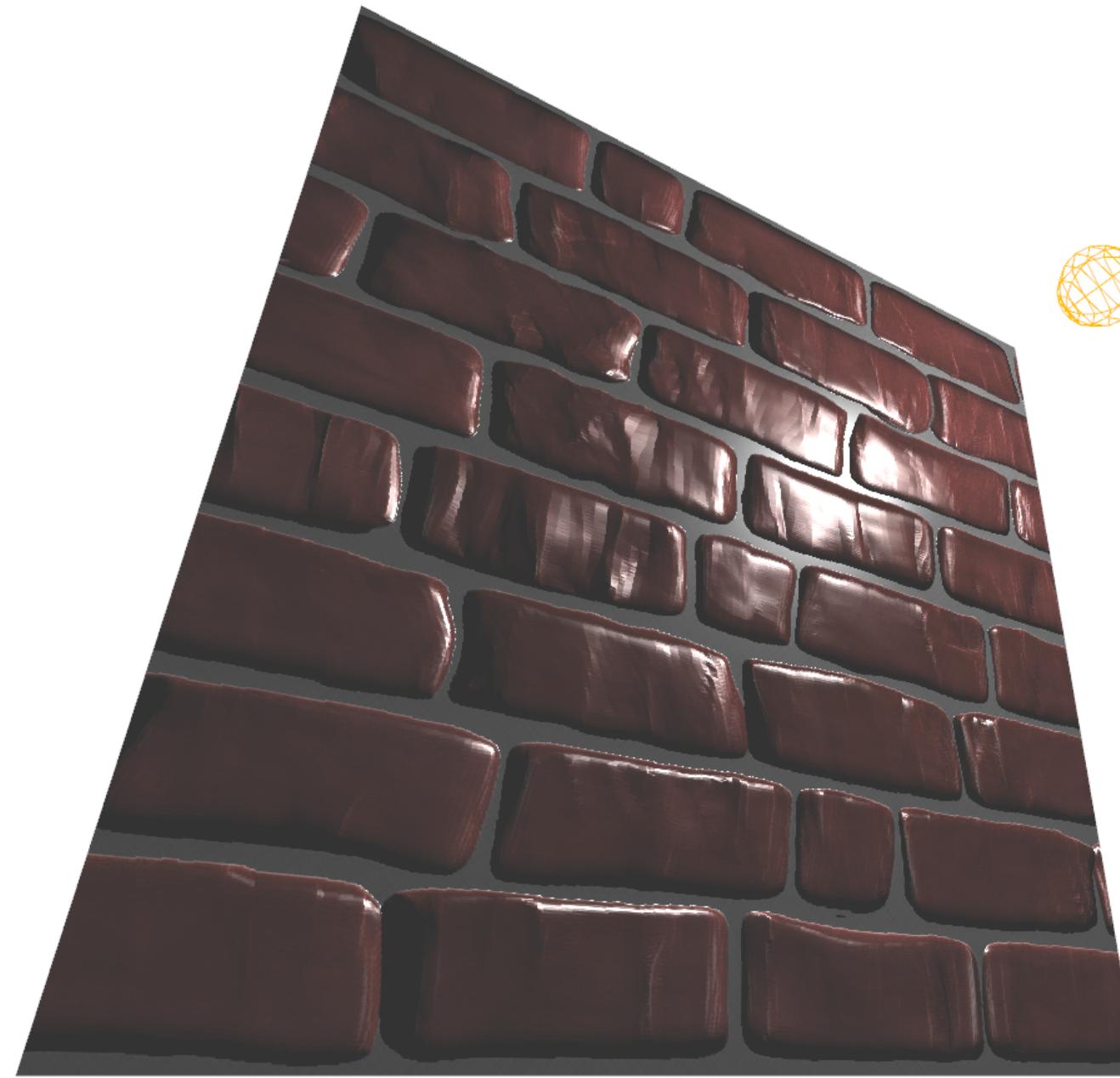
$$\beta_s = \beta'(u, v) \cdot \frac{\partial \varphi}{\partial s}(s, t) \quad (\beta_t \text{ analog}).$$

Bemerkungen 7.26:

1. Zur **Schnittpunktberechnung** wird weiterhin **die ursprüngliche (nicht gestörte) Fläche $P(s, t)$** verwendet; die verschobenen Punkte \tilde{P} gehen nur in die Normalen ein.

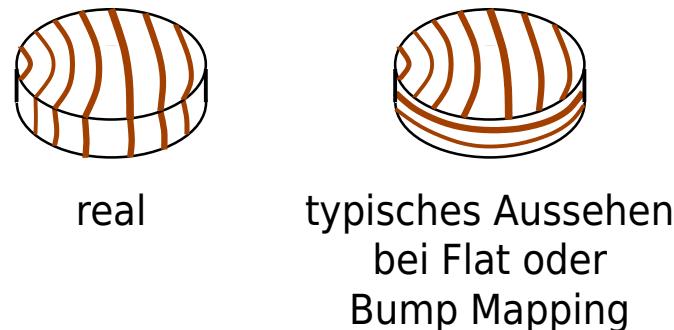
⇒ **Die Silhouette der Fläche bleibt glatt!**
 2. Berechnung der Ableitungen:
 - P_s und P_t können entweder für jeden Punkt P (z. B. mittels Differenzenquotienten) berechnet oder aus einigen gegebenen Werten (z. B. in den Ecken der Polygone) interpoliert werden.
 - β wird i. Allg. nur an diskreten Stellen $(u_i, v_j) \in \tilde{D}$ vorgegeben; Zwischenwerte werden interpoliert, Ableitungen durch Differenzenquotienten.
 - φ ist meist eine affine Abbildung.

⇒ φ' ist eine konstante Matrix.
 3. Bump Mapping kann sehr realistische Bilder liefern.
 4. Durch **Normal Mapping (Dot3 bump mapping)** können Normalenvektoren im Tangentenraum direkt in die Textur integriert werden.
-



Volume Mapping

Motivation: Viele reale Texturen sind dreidimensional, also von der Position und Orientierung der Oberfläche abhängig, z. B. Holzmaserung auf einem „Mühle“-Stein:



Ansatz: Verwende dreidimensionale Texture Map und/oder Bump Map, z. B.

$$\beta = \beta(u, v, w) \quad \text{mit} \quad (u, v, w) \in \tilde{D} \subseteq \mathbb{R}^3$$

und verwende für den Zugriff auf diese Map nicht die Parameter (s, t) des Punktes P , sondern seine dreidimensionalen Koordinaten (relativ zu einem „Referenzpunkt“).

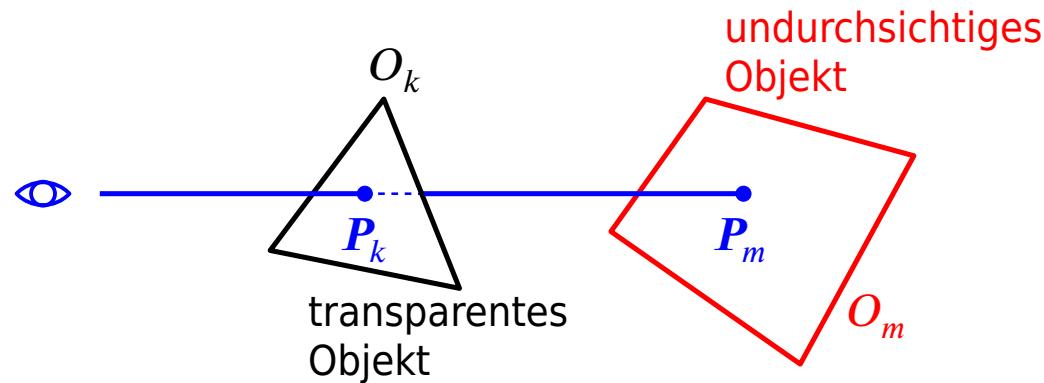
Bemerkung 7.27: Dreidimensionale Maps können i. Allg. nicht mehr explizit gespeichert werden.

⇒ prozedurale Definition

7.5.3 Transparenz

Ziel: Darstellung von Objekten, die (teilweise) lichtdurchlässig sind

Ansatz:



Trifft der Sehstrahl zuerst auf ein transparentes Objekt O_k , so verfolge ihn weiter bis zum ersten undurchsichtigen Objekt O_m . Kombiniere dann die Intensitäten von beiden Objekten gemäß

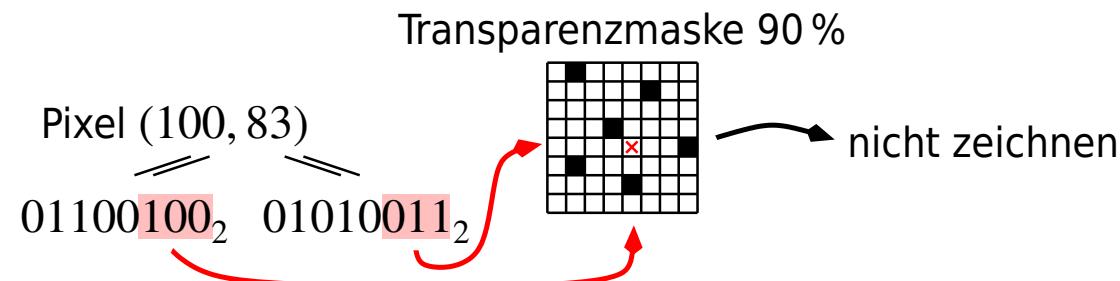
$$I = \left(1 - \underbrace{R_{k,t}}_{\text{Transparenzkoeffizient von Objekt } k \text{ (abhängig von } \lambda)} \right) \cdot I(P_k) + R_{k,t} \cdot I(P_m) \quad (\text{Interpolation})$$

oder

$$I = I(P_k) + R_{k,t} \cdot I(P_m) \quad (\text{Filterung}) .$$

Bemerkung 7.28: Liegen mehrere transparente Objekte vor dem ersten undurchsichtigen Objekt, so muss die Interpolation/Filterung mehrmals durchgeführt werden.

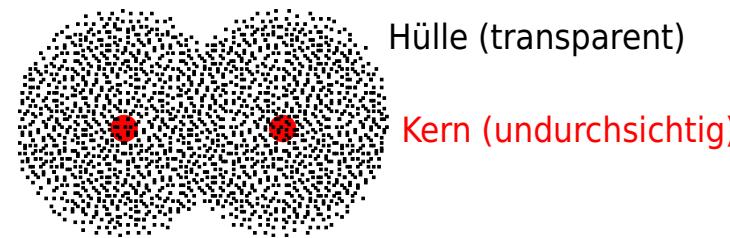
anderer Ansatz: Ist ein Objekt O_k zu 90 % transparent, so zeichne nur 10 % der zu O_k gehörenden Pixel mit der Farbe des Objekts. Die Auswahl der gezeichneten Pixel erfolgt mittels einer Maske:



Problem: Liegen mehrere Objekte mit gleicher Transparenzmaske hintereinander, so ist i. Allg. nur das vorderste davon sichtbar.

⇒ Verwende statt der „absoluten“ Koordinaten der Pixel die relative Position bzgl. eines (vom Objekt abhängigen) „Referenzpixels“.

Anwendung dieser **Masken-Transparenz (Screen-Door Transparency)** z. B. bei Molekülmodellen:



- ⊕ Masken-Transparenz erfordert kein Weiterverfolgen des Sehstrahls (Aufwand!).
 - ⊕ Masken-Transparenz kann sehr leicht in den z -Puffer-Algorithmus eingebaut werden.
-

Bemerkung 7.29: Analytische Berücksichtigung der Transparenz (Interpolation/Filterung) ist nicht problemlos mit dem z -Puffer-Algorithmus kombinierbar; sie wird meist bei analytischer Sichtbarkeitsbestimmung oder Visible Surface Raytracing eingesetzt.

- ⊖ Beide Ansätze vernachlässigen dreidimensionale Effekte:
 - Lichtbrechung
 - Dämpfung des Lichts in Abhängigkeit der im transparenten Medium zurückgelegten Strecke

8 Modellierung der Objekte

Inhalt

8.1 Flächenbasierte Modelle	8-2
8.2 Volumenbasierte Modelle	8-6
8.2.1 Operationen mit bestehenden Volumina	8-8
8.2.2 Definition von Volumina	8-9
8.2.3 Speicherung von Volumina	8-11
8.3 Prozedurale Modelle	8-15
8.3.1 Fraktale Modelle	8-16
8.3.2 Objekterzeugung mit Grammatiken	8-19
8.3.3 Partikelsysteme	8-24

Ziel: der jeweiligen Anwendung angepasste Repräsentation der Objekte

8.1 Flächenbasierte Modelle

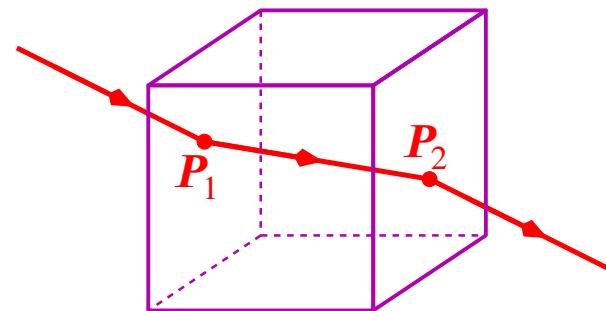
Motivation: Wenn die Objekte nur dargestellt werden sollen, ist die Beschreibung durch ihre Oberfläche ausreichend.

Information über die Zusammengehörigkeit mehrerer Flächenstücke zum selben **Körper** werden nicht benötigt.

Beispiel 8.1: Für rekursives Raytracing wird der Würfel durch sechs Quadrate (jeweils mit Materialangabe für die **Außen- und Innenseite**) beschrieben.

Das vom Würfel belegte **Volumen** ist nur **implizit definiert**:

- Durchdringt der rückverfolgte Strahl bei P_1 eines der sechs Quadrate **von außen nach innen**, so wird angenommen, dass der Strahl bis zum nächsten Schnittpunkt P_2 innerhalb des Würfels verläuft.



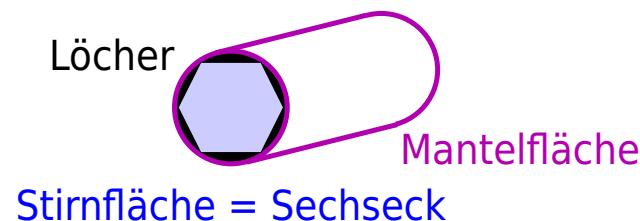
- ⊕ Objekte effizient speicher- und darstellbar
- ⊖ Die Oberfläche der Objekte muss (i. Allg. vom Anwender!) in „einfache“ Teile (**Patches**) zerlegt werden:

- Polygone
- Quadriken (Kugeloberflächen, ...)
- parametrisierte Flächen $\mathbf{P} = \mathbf{P}(s, t)$

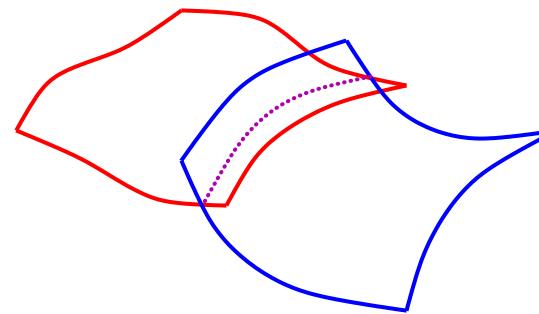
- Splineflächen $\mathbf{P} = \mathbf{P} \left(\underbrace{\mathbf{P}_i}_{\text{Steuerpunkte}}, \underbrace{\omega_i}_{\text{Gewichte}} \right)$

Aufwand
Flexibilität

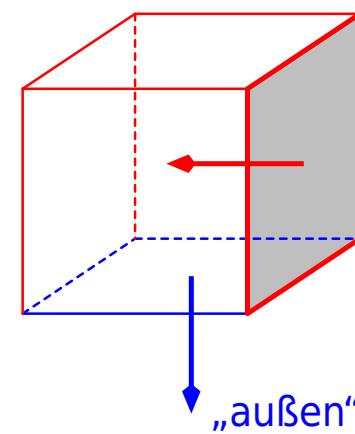
- ⊖ Konsistenz muss explizit gesichert werden.
- Der Rand des Körpers muss vollständig überdeckt sein.



- Die Patches müssen an den „Nahtstellen“ zusammenpassen.

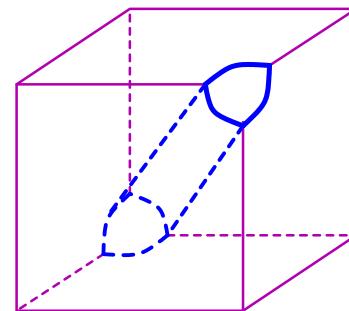


- Die „innen“/„außen“-Information muss konsistent sein.



⊖ Operationen mit Körpern sind nur eingeschränkt möglich:

„Bohre ein zylinderförmiges Loch durch den Würfel.“



⇒ Oberfläche muss (explizit) wieder zerlegt werden

- Volumenabhängige Information (z. B. Gewicht eines Werkstücks) ist nur mit großem zusätzlichem Aufwand zu beschaffen.

8.2 Volumenbasierte Modelle

Motivation: Beim Entwurf von Maschinen usw. sind die Objekte die Körper selbst, nicht Teile ihrer Oberflächen.

⇒ Der vom Objekt ausgefüllte Raum sollte aus der Darstellung des Objekts im Raum hervorgehen.

Forderungen an das Modell:

Mächtigkeit: Die für die jeweilige Anwendung benötigten Objekte sollten (leicht) erzeugbar sein.

Eindeutigkeit: Jedes Objekt sollte eine eindeutige Darstellung besitzen.

(⇒ einfacher Test, ob zwei Objekte gleich sind)

Exaktheit: Die „üblichen“ Objekte sollten ohne Approximation darstellbar sein.

(z. B. Kugel nicht durch Polyeder approximieren)

Intuitivität: Es sollte möglich sein,

- aus bereits konstruierten Objekten andere zusammenzusetzen,
- Stücke aus Objekten „herauszuschneiden“,
- usw.

Konsistenz: Es sollte unmöglich (oder zumindest sehr schwierig) sein, unsinnige Objekte (z. B. nicht geschlossene Körper) zu konstruieren.

Effizienz: Die Darstellung sollte möglichst einfache

- Bilderzeugung,
- Gewinnung relevanter Information (z. B. ein CNC-Programm zum Fräsen des Werkstücks)

erlauben.

Anwendungen:

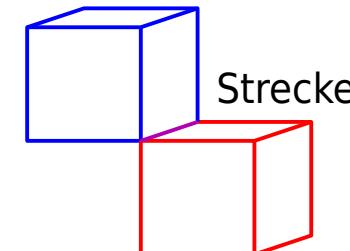
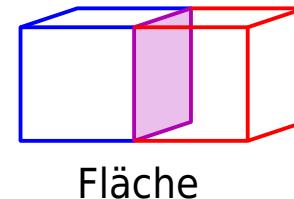
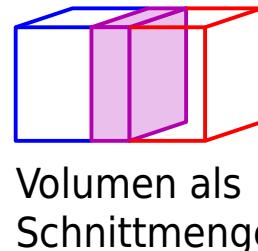
- Entwurf von Werkstücken
- Medizin: Volumenmodell von Organen (gewonnen aus Computer-Tomografie-Daten)
 - ⇒ Volumen eines Tumors,
 - günstiger Zugang für Operation, ...

8.2.1 Operationen mit bestehenden Volumina

gewünschte Operationen:

- Zusammensetzen von Objekten
(\triangleq Vereinigung)
- „Abschneiden“ von Teilen eines Objekts
(\triangleq Durchschnitt, Mengendifferenz)

Problem: Diese Operationen liefern nicht immer ein Volumen:



Abhilfe: regularisierte Mengenoperationen:

$$A \text{ op}^* B := \overline{(A \text{ op } B)^\circ}$$

op: \cup, \cap, \setminus

$(\dots)^\circ$: Inneres

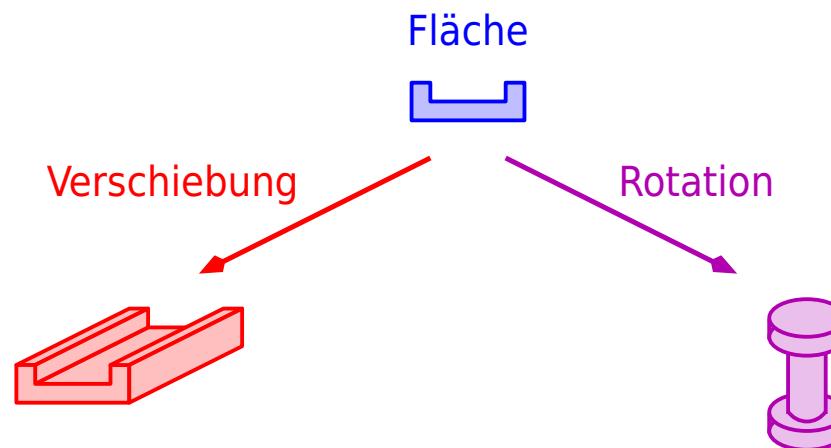
$\overline{(\dots)}$: Abschluss

8.2.2 Definition von Volumina

1. Transformiere gegebene Volumina (z. B. Würfel, Kugel) auf die gewünschte Form.

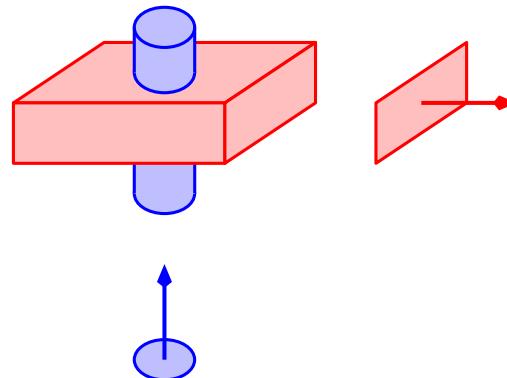
Die vorgegebenen Bausteine sind oft noch parametrisiert, z. B. „Rad mit n Speichen“ (n kann bei jedem Aufruf unterschiedlich festgelegt werden).

2. **Sweep**: Bewege eine Fläche oder ein Volumen im Raum und erzeuge dadurch ein neues Volumen.



Bemerkungen 8.2:

1. Oft werden auch Bewegungen entlang von Raumkurven zugelassen sowie die Möglichkeit, die Fläche während der Bewegung zu ändern.
2. Mengenoperationen mit Sweeps ergeben i. Allg. nicht wieder Sweeps:



⇒ Sweeps können nicht immer in der Form
erzeugendes Element + Bewegung
gehandhabt werden.

3. Mit Sweeps können beispielsweise die von einem Fräskopf aus einem Werkstück geschnittenen „Bahnen“ modelliert werden.
4. Bewegung einer Fläche liefert nicht immer ein Volumen!

8.2.3 Speicherung von Volumina

Randspeicherung

(Boundary representations, **b-reps**)

Das **Modellierungssystem** speichert Volumina in Form ihrer Oberflächen.

Nach Mengenoperationen o. ä. wird die Oberfläche des resultierenden Volumens **automatisch** in einfache Teile zerlegt.

- ⊕ Die Oberflächen können direkt zur Bilderzeugung verwendet werden.
- ⊖ relativ aufwändige Algorithmen zur Verwaltung der Oberflächen notwendig (Bestimmung von Schnittkurven, ...)
 - ⇒ Systeme oft auf Volumina mit polygonalen Oberflächen beschränkt

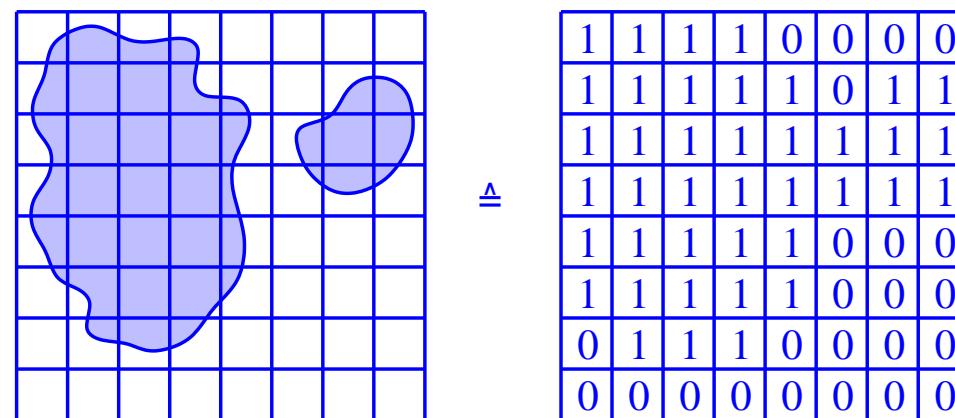
Volumenaufzählung

Zerlege den Raum in kleine Zellen und zähle die vom Objekt (teilweise) überdeckten Zellen auf:

1. Zellraster-Unterteilung:

- Alle Zellen sind gleich groß.
- mögliche Zustände jeder Zelle: „belegt“/„nicht belegt“

$$\begin{array}{c} \overbrace{\quad\quad}^{\triangleq 1} \quad \overbrace{\quad\quad}^{\triangleq 0} \end{array}$$

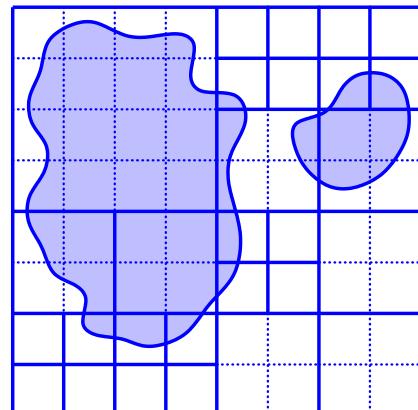


- Speicherung:
 - durch Aufzählen der belegten Zellen oder
 - als Bitfeld
- ⇒ Mengenoperationen werden sehr einfach und effizient.

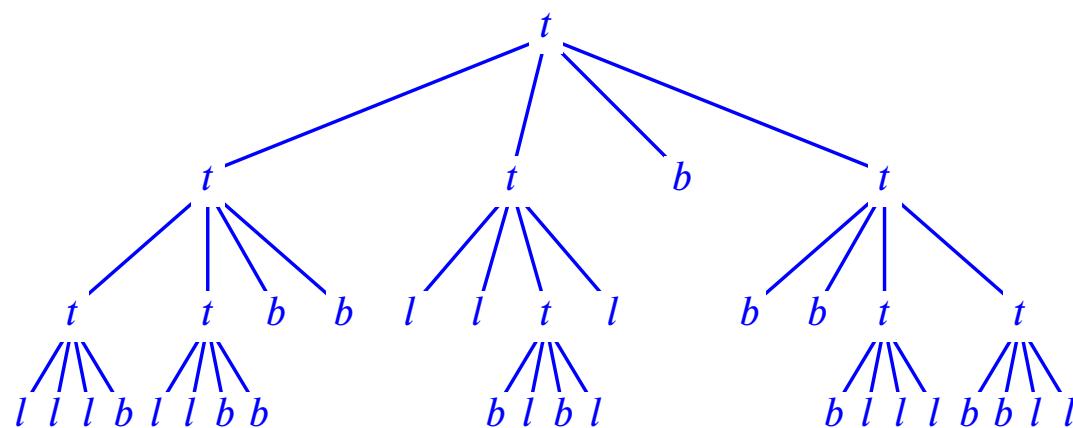
2. Octrees:

- Hierarchie von Zellen.
- mögliche Zustände: „belegt“ / „teilweise belegt“ / „leer“

b t l



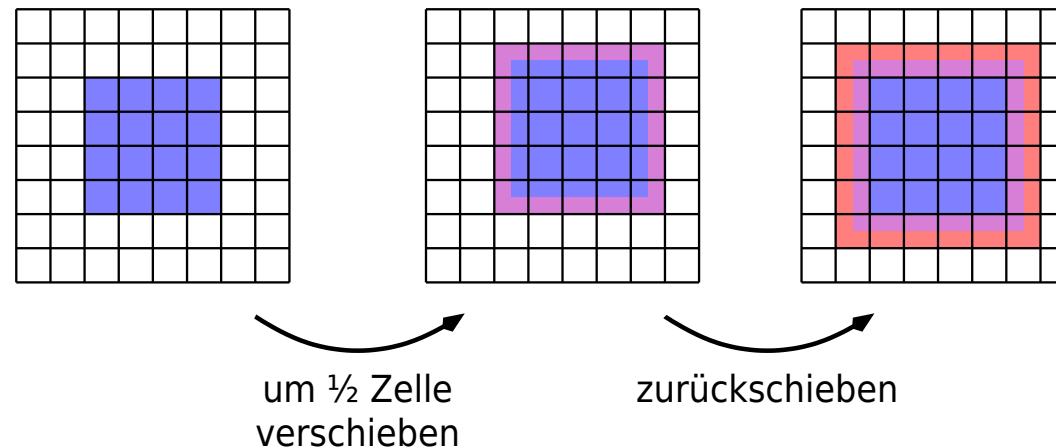
ergibt bei Quadrantennummerierung den Quadtree



- Mengenoperationen einfach durchführbar (Durchlaufen der beiden Bäume)
-

Bemerkungen 8.3:

1. Volumenaufzählung benötigt i. Allg. viel mehr Speicherplatz als die Randspeicherung.
2. Die Objekte werden i. Allg. nur approximiert.
3. Ohne Vorsichtsmaßnahmen sind die Volumina **nicht bewegungs invariant**:



(ähnlich bei Rotationen)

4. Vor der Bilderzeugung muss die **Oberfläche** des Körpers **approximiert** werden (die Wände der „Randzellen“ ergäben ein „Lego“-Bild).
-

8.3 Prozedurale Modelle

Motivation: Enthält eine Szene sehr viele Details, z. B.

- „realistisches“ (nicht-glattes) Terrain,
- Bäume mit Blättern,
- dreidimensionale Textur,
- ...,

so ist es i. Allg. nicht mehr möglich oder sinnvoll, diese Information durch explizite Vorgabe aller entsprechenden Objekte einzubringen.

Ein Terrain kann beispielsweise durch ein „Dreiecksnetz“ approximiert werden.
typische Größenordnung:

$$\gg 10^6 \text{ Dreiecke ,}$$

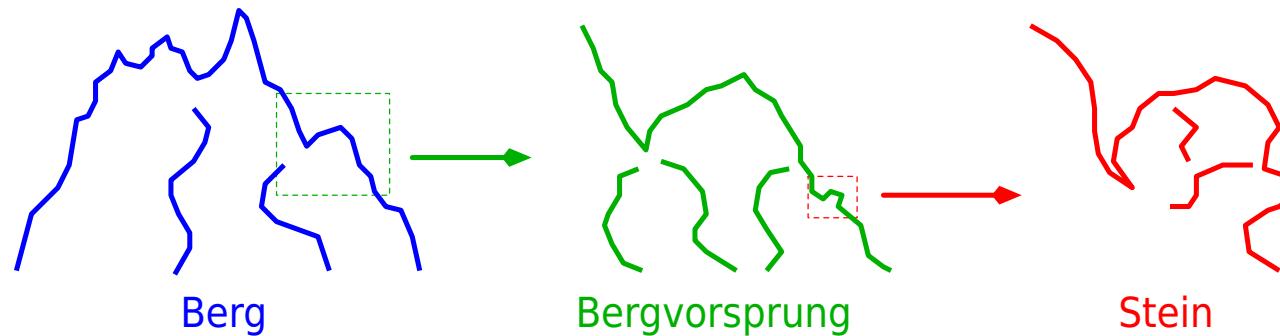
falls auch Bildvergrößerung möglich sein soll

Ansatz: Gib statt der Objekte eine Vorschrift an, mit der die Objekte „bei Bedarf“ erzeugt werden können.

8.3.1 Fraktale Modelle

Beobachtung: Viele in der Natur vorkommenden Formen sind zu einem gewissen Grad **selbstähnlich**, d. h. Teile der Objekte sehen aus wie verkleinerte Kopien der Objekte selbst.

Beispiel: Gebirge

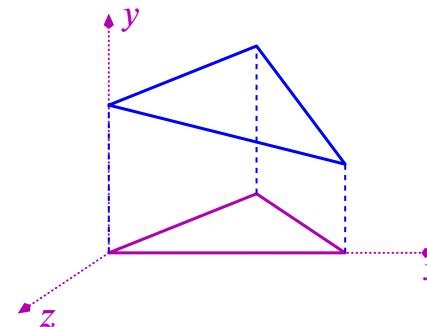


Je näher man kommt, desto kleinere (im Wesentlichen der ursprünglichen Form ähnelnde) Details werden sichtbar.

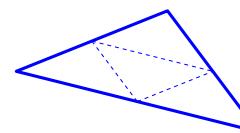
Ansatz: Starte mit einer groben Form und füge nach und nach immer kleinere Details (vom selben „Typ“) hinzu.

Beispiel 8.4: Erzeugung eines Bergs:

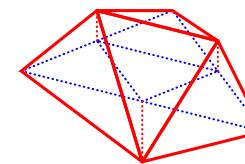
Starte mit einem „beliebigen“ Dreieck.



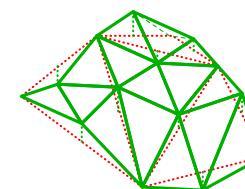
Zerlege das Dreieck an den Seitenmittelpunkten in vier kongruente Teile.



Verschiebe jeden der drei neuen Punkte um einen zufälligen Betrag in y -Richtung.



Wiederhole diese beiden Schritte rekursiv für jedes der entstandenen vier Dreiecke.

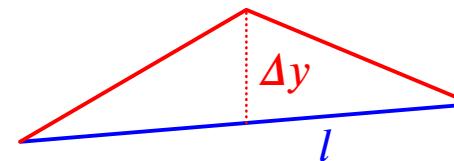


⋮

Bemerkungen 8.5:

1. Die Verschiebung Δy hängt von der Länge der gerade unterteilten Dreiecksseite ab:

$$\Delta y = \underbrace{\text{rnd}}_{\substack{\text{Zufallszahl} \\ \in [-1; 1]}} \cdot \underbrace{f(l)}_{\substack{\text{streng monoton steigend,} \\ \text{z. B. } f(l) = l^\alpha, \alpha \in (0; 1]}}$$



2. Die Rekursionstiefe hängt vom Abstand zum Betrachter ab: je größer die Entfernung, desto weniger Details sind notwendig
(Abbruch z. B., wenn Dreiecke etwa Pixelgröße besitzen)
 3. Die Dreiecke werden i. Allg. nicht explizit gespeichert, sondern bei Bedarf neu erzeugt (dabei wird jeweils nur ein Dreieck weiter unterteilt).
- ⇒ Die Verschiebungen (d. h. die Zufallszahlen) müssen **reproduzierbar** sein!

8.3.2 Objekterzeugung mit Grammatiken

Beobachtung: Ein Zweig **z** mit einer Knospe **K** an der Spitze kann sich auf verschiedene Arten entwickeln:

1. Die Knospe kann absterben.
2. Die Knospe kann eine Blüte **B** erzeugen und dann absterben.
3. Die Knospe kann ein Blatt **I** erzeugen.
4. Die Knospe kann austreiben:
 - Neben der Knospe entstehen evtl. eine oder zwei weitere Knospen.
 - Die ursprüngliche Knospe wächst weiter zu einem Zweig.
 - An der Spitze dieses Zweigs ist wieder eine Knospe.

Abstraktion: Beschreibe diese Zustandsveränderungen durch eine (kontextfreie) „Grammatik“.

Nichtterminale: **K, B**

Terminale: **z, I, (,), [,]**

Produktionen: $K \rightarrow K \mid \epsilon$ ($\epsilon \triangleq$ „nichts“)

$K \rightarrow B$

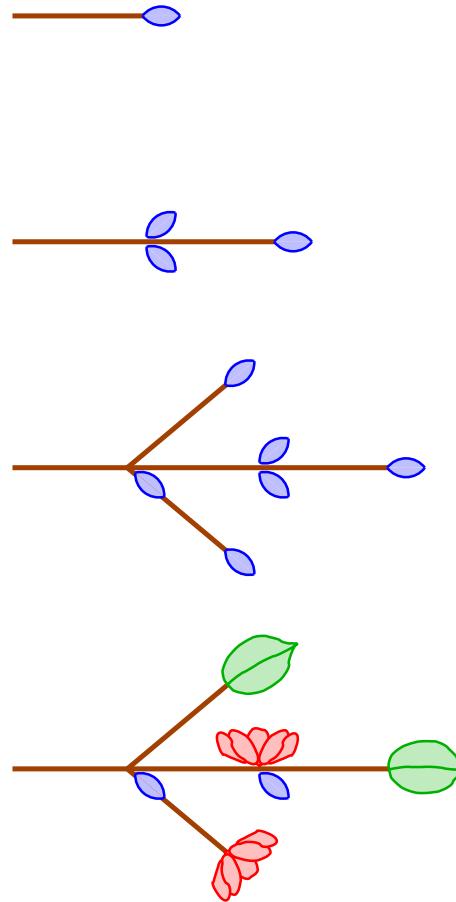
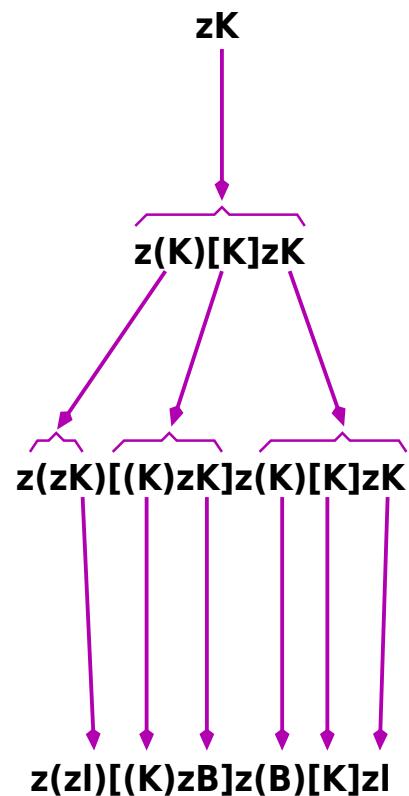
$B \rightarrow B \mid \epsilon$

$K \rightarrow I$

$K \rightarrow zK \mid (K)zK \mid [K]zK \mid (K)[K]zK$

Startwort: **zK**

Beispiel 8.6: Eine mögliche Ableitung:

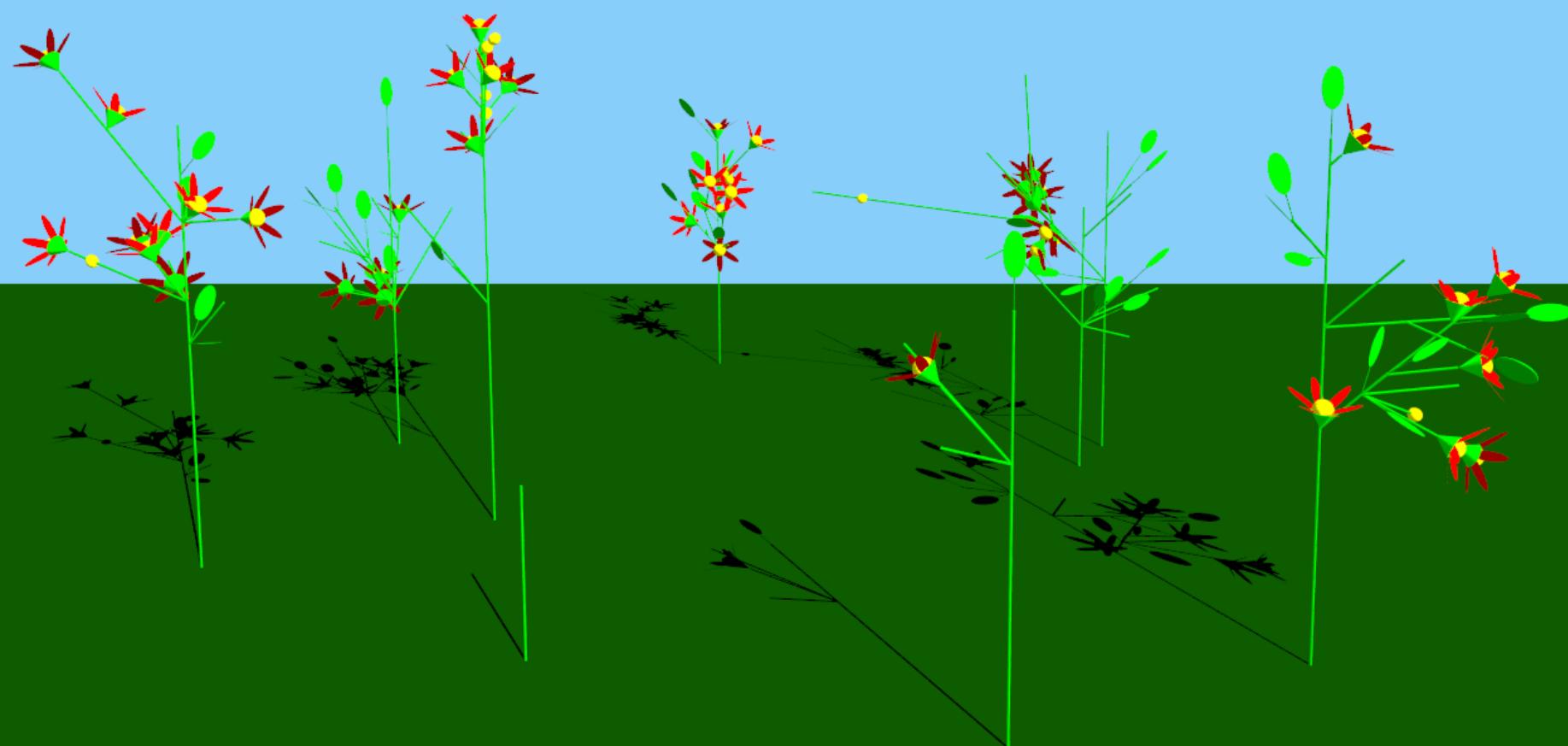


Bemerkungen 8.7:

1. Beim Übergang von einer Generation zur nächsten wird auf jedes Nichtterminal im Wort irgendeine der passenden Produktionen angewandt.
 2. Produktionen der Form $\mathbf{B} \rightarrow \mathbf{B}$ können entfallen, wenn nur das Endergebnis interessiert, seine zeitliche Entwicklung aber ohne Belang ist.
 3. Man kann eine beliebige (i. Allg. kontextfreie) Grammatik und eine **Interpretation** vorgeben, z. B.:
 $(\dots) \triangleq$ Verzweigung nach links mit Inhalt \dots
 4. Die Interpretation eines Symbols muss nicht positionsunabhängig sein, z. B.:
Zweige, die in späteren Generationen entstehen, kürzer machen
 5. Auswahl der Produktionen mit vorgegebenen Wahrscheinlichkeiten (und ggf. nach zusätzlichen Regeln)
 6. Die Objekte können vorab oder bei Bedarf erzeugt werden.
-

Für das folgende Beispielbild wurde einige Erweiterungen vorgenommen:

- Übergang auf 3D
- damit Verzweigungen in verschiedene Raumrichtungen
- Auswahl zwischen mehreren Produktionsregeln durch Zufallszahlen, dabei Anpassung der Zufallssteuerung an Fortschritt des Wachstums
- Erweiterung der Grammatik: Speicherung von Winkeln und Zweiglängen innerhalb der Ableitungsstrings



8.3.3 Partikelsysteme

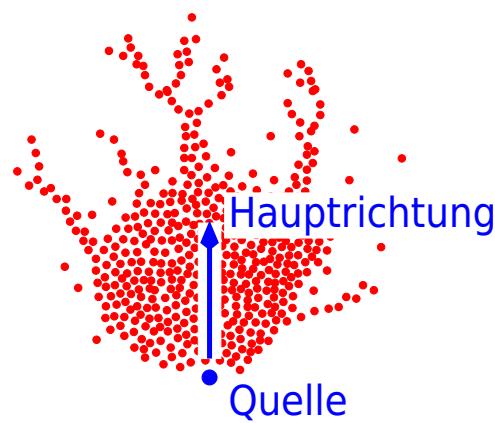
Ziel: Modellierung von „Objekten“, die nicht adäquat durch eine Oberfläche oder ein Volumen beschrieben werden können, etwa weil sich ihre Topologie im Laufe der Zeit stark ändert

Ein **Partikelsystem** ist eine Menge von Teilchen, das sich (durch Anwenden physikalischer Regeln auf die Teilchen) im Laufe der Zeit verändert. Die Teilchen können

- ihre Eigenschaften (z. B. Geschwindigkeit, Farbe) ändern,
- neue Teilchen erzeugen,
- selbst verschwinden.

Anwendung z. B. bei der Modellierung von

- Funkenflug bei Feuer, Explosionen, Feuerwerk,
- Nebel,
- Laub dichter Bäume.

Beispiel 8.8: loderndes Feuer

$\text{Richtung}(\text{Teilchen } i, \text{Zeitpunkt } t + 1) = \text{Richtung } (i, t) + \text{Einfluss der Schwerkraft} + \text{Zufallsstörung}$

Farbe der Teilchen geht mit der Zeit von Weiß(glühend) (entstehen) über Rot nach Schwarz (verschwinden)

Problem: Partikelsysteme umfassen oft mehrere Millionen von Teilchen.

⇒ Partikelsysteme können nicht problemlos in die „Standardverfahren“ zur Bilderzeugung (z. B. rekursives Raytracing) eingebunden werden.

Ansatz: Bestimme direkt den Beitrag jedes Teilchens zum Bild.

hier: Verfolge jedes Teilchen auf einem kurzen Stück seiner Flugbahn und erhöhe die Intensität aller von dieser Strecke betroffenen Pixel.

Auswirkungen:

- Die Teilchen gehen nicht in die Verdeckungsanalyse der übrigen Objekte ein
(d. h. durch andere Objekte verdeckte Teilchen müssen nach anderen Kriterien entfernt werden).
- Der Einfluss der Teilchen auf die übrigen Objekte (hier: Beleuchtung durch das Feuer) bleibt unberücksichtigt.

(Abhilfe hier: Bewegliche Lichtquelle variabler Intensität im Feuer positionieren)

Bemerkungen 8.9:

1. Für jede neue Anwendung der Partikelsysteme (Feuer, Nebel, ...) muss i. Allg. ein neuer Darstellungsalgorithmus (mit vielen heuristischen Ansätzen) entwickelt werden.
 2. Trotz der vielen Heuristik oft überzeugende Effekte.
-

9 Kurven und Flächen

Inhalt

9.1 Parametrisierte kubische Kurven	9-2
9.1.1 Hermite-Kurven	9-6
9.1.2 Bézier-Kurven	9-9
9.1.3 Kubische Splines	9-14
9.1.4 Unterteilung von Kurven	9-23
9.1.5 Zeichnen von Kurven	9-26
9.2 Parametrisierte bikubische Flächen	9-26
9.2.1 Bézier-Flächen	9-28
9.3 Rotationskörper	9-30

Reale Objekte werden often von „**glatten**“ Kurven und Flächen begrenzt.

Wie lassen sich diese mit wenigen „**Kontrollpunkten**“ beschreiben?

9.1 Parametrisierte kubische Kurven

Beschreibung von Kurven (Flächen) auf drei Arten möglich:

1. explizit: $y = f(x)$ und $z = g(x)$ (Kurve), $z = f(x, y)$ (Fläche)
2. implizit: $f(x, y, z) = 0$ (z. B. $x^2 + y^2 + z^2 - 1 = 0$ für Kugel)
3. parametrisiert: $x = x(t)$, $y = y(t)$, $z = z(t)$, $t \in [0; 1]$

Eine **parametrisierte kubische Kurve**

$$Q(t) = (x(t), y(t), z(t))^T , \quad t \in [0; 1]$$

wird durch drei kubische Polynome definiert:

$$\begin{aligned}x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \\z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z\end{aligned}$$

Mit $T = (t^3, t^2, t, 1)^T$ und der Koeffizientenmatrix

$$C = \begin{pmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{pmatrix}$$

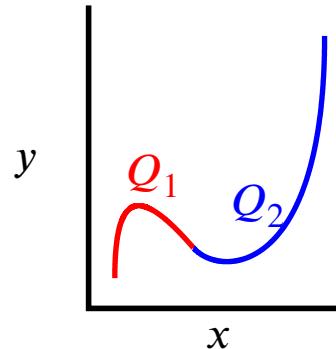
ergibt sich folgende Matrixschreibweise:

$$Q(t) = C \cdot T$$

Tangentialvektor an die Kurve:

$$\frac{d}{dt} Q(t) = Q'(t) = \left(\frac{d}{dt} x(t), \frac{d}{dt} y(t), \frac{d}{dt} z(t) \right)^T = C \cdot \frac{d}{dt} T = C \cdot (3t^2, 2t, 1, 0)^T = \begin{pmatrix} 3a_x t^2 + 2b_x t + c_x \\ 3a_y t^2 + 2b_y t + c_y \\ 3a_z t^2 + 2b_z t + c_z \end{pmatrix}$$

Stetigkeit bei der Verbindung zweier Kurvenstücke Q_1, Q_2 :



- C^0 -/ G^0 -stetig: $Q_1(1) = Q_2(0)$
- G^1 geometrisch stetig: G^0 und gleiche Tangentenrichtung, d. h. $Q'_1(1) = k \cdot Q'_2(0)$ mit $k > 0$
- C^1 stetig: C^0 und gleiche Tangente (Richtung und Betrag), d. h. $Q'_1(1) = Q'_2(0)$
- G^2 geometrisch stetig: G^1 und gleiche Krümmung
- C^2 stetig: C^1 und $Q''_1(1) = Q''_2(0)$

Bemerkung 9.1: G^2 ist z. B. wichtig bei Straßenbau.

Übergang gerade Straße - Kreisbogen wäre G^1 , aber nicht G^2

⇒ Lenkrad muss ruckartig bewegt werden

im folgenden Zerlegung der Matrix C in $C = G \cdot M$ mit 4×4 -**Basismatrix** M und 3×4 -**Geometriematrix** G

$$Q(t) = (G_1, G_2, G_3, G_4) \cdot \begin{pmatrix} m_{11} & \cdots & m_{14} \\ \vdots & & \vdots \\ m_{41} & \cdots & m_{44} \end{pmatrix} \cdot \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix} \quad \text{mit} \quad G_i = \begin{pmatrix} g_{ix} \\ g_{iy} \\ g_{iz} \end{pmatrix}$$

M : feste Matrix für bestimmten Kurventyp

G : Lage (z. B. End-/Kontrollpunkte) einer konkreten Kurve

9.1.1 Hermite-Kurven

Charles Hermite
* 1822, Dieuze, Lothringen, FR
† 1901, Paris
Mathematiker



Quelle: https://commons.wikimedia.org/wiki/File:Charles_Hermite_circa_1901_edit.jpg

Kurve bestimmt durch: Endpunkte P_1 und P_4 und Tangentialvektoren R_1 und R_4 in Endpunkten

Bestimmung der Hermite-Basismatrix M_H in $Q(t) = G_H \cdot M_H \cdot T$ mit $G_H = (P_1, P_4, R_1, R_4)$:

$$x(t) = G_{H_x} \cdot M_H \cdot \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}, \quad x'(t) = G_{H_x} \cdot M_H \cdot \begin{pmatrix} 3t^2 \\ 2t \\ 1 \\ 0 \end{pmatrix}$$

Einsetzen der Punkte/Vektoren für $t = 0$ bzw. $t = 1$ gibt:

$$x(0) = P_{1_x} = G_{H_x} \cdot M_H \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \quad x(1) = P_{4_x} = G_{H_x} \cdot M_H \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$x'(0) = R_{1_x} = G_{H_x} \cdot M_H \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad x'(1) = R_{4_x} = G_{H_x} \cdot M_H \cdot \begin{pmatrix} 3 \\ 2 \\ 1 \\ 0 \end{pmatrix}$$

zusammengesetzt:

$$G_{H_x} = (P_{1_x}, P_{4_x}, R_{1_x}, R_{4_x}) = G_{H_x} \cdot M_H \cdot \begin{pmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

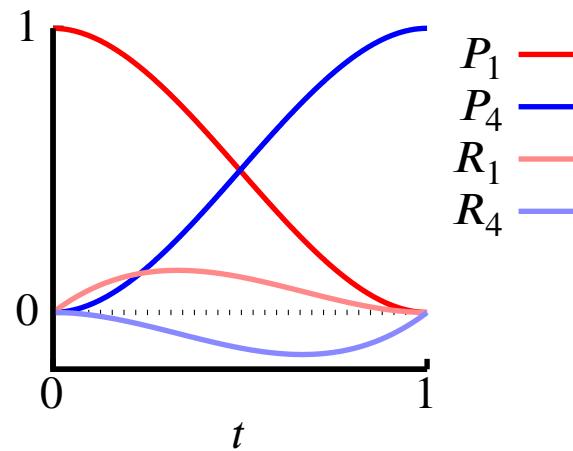
Invertierung obiger Matrix liefert M_H :

$$M_H = \begin{pmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix}$$

Ausmultiplizieren des rechten Produkts in $Q(t) = G_H \cdot M_H \cdot T$ gibt eine Darstellung der Kurve in den **Hermite-Basis-Polynomen**:

$$Q(t) = (2t^3 - 3t^2 + 1) \cdot P_1 + (-2t^3 + 3t^2) \cdot P_4 + (t^3 - 2t^2 + t) \cdot R_1 + (t^3 - t^2) \cdot R_4$$

Darstellung der Basis-Polynome:



Bemerkung 9.2: G^1 -stetiger Übergang zweier Hermite-Kurven, falls die Geometriematrizen die Form (P_1, P_4, R_1, R_4) und (P_4, P_7, kR_4, R_7) mit $k > 0$ haben

9.1.2 Bézier-Kurven

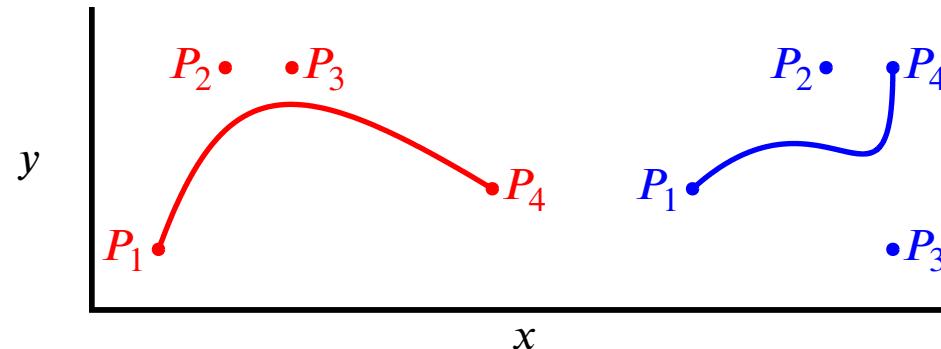
Pierre Étienne Bézier

* 1910, Paris

† 1999, Paris

Bau- und Elektroingenieur, Mathematiker (Renault)

Kurve bestimmt durch: Endpunkte und zwei Kontrollpunkte



Geometriematrix: $G_B = (P_1, P_2, P_3, P_4)$

Zusammenhang Hermite-Bézier: $R_1 = 3(P_2 - P_1)$, $R_4 = 3(P_4 - P_3)$

Übergang zwischen den Geometriematrizen über Matrix M_{HB} :

$$G_H = G_B \cdot M_{HB} \quad \text{mit}$$

$$(P_1, P_4, R_1, R_4) = (P_1, P_2, P_3, P_4) \cdot \begin{pmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{pmatrix}$$

Bézier-Basismatrix aus Hermite-Form:

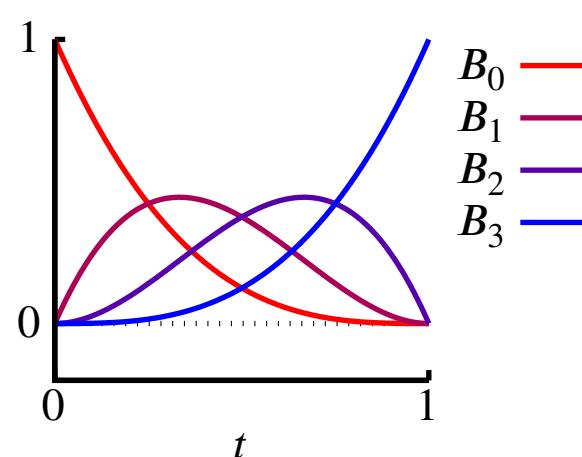
$$Q(t) = G_H \cdot M_H \cdot T = G_B \cdot \underbrace{M_{HB} \cdot M_H}_{=: M_B} \cdot T$$

also:

$$M_B = M_{HB} \cdot M_H = \begin{pmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Basispolynome aus $Q(t) = G_B \cdot M_B \cdot T$ sind die sog. **Bernsteinpolynome**:

$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4$$



Sergej Natanovič Bernštejn [Сергей Натанович Бернштейн]

* 1880, Odessa (Одесса, Russisches Reich, heute Ukraine)

† 1968, Moskau (Москва [Moskva])

Mathematiker



Foto: Konrad Jacobs, Erlangen, Titel: „Sergei Natanowitsch Bernstein“
Quelle: https://commons.wikimedia.org/wiki/File:Sergei_Natanowitsch_Bernstein.jpg
Lizenz: CC BY-SA 2.0, <https://creativecommons.org/licenses/by-sa/2.0/de/legalcode>

Verbindung zweier Kurvenstücke zu Kontrollpunkten (P_1, P_2, P_3, P_4) bzw. (P_4, P_5, P_6, P_7) über gemeinsamen Punkt P_4

G^1 -stetig falls $P_3 - P_4 = k(P_4 - P_5)$ mit $k > 0$

C^1 -stetig falls $P_3 - P_4 = P_4 - P_5$

Interpolation:

gegeben: l Punkte Q_1, \dots, Q_l

gesucht: C^1 -stetige stückweise kubische Interpolation der Q_i

Ansatz: Bézier-Kurven zu Kontrollpunkten $(Q_i = P_{3i-2}, P_{3i-1}, P_{3i}, Q_{i+1} = P_{3i+1}), i = 1, \dots, l-1$

Bestimmung der P_i :

für $i = 1, \dots, l$

$$P_{3i-2} := Q_i$$

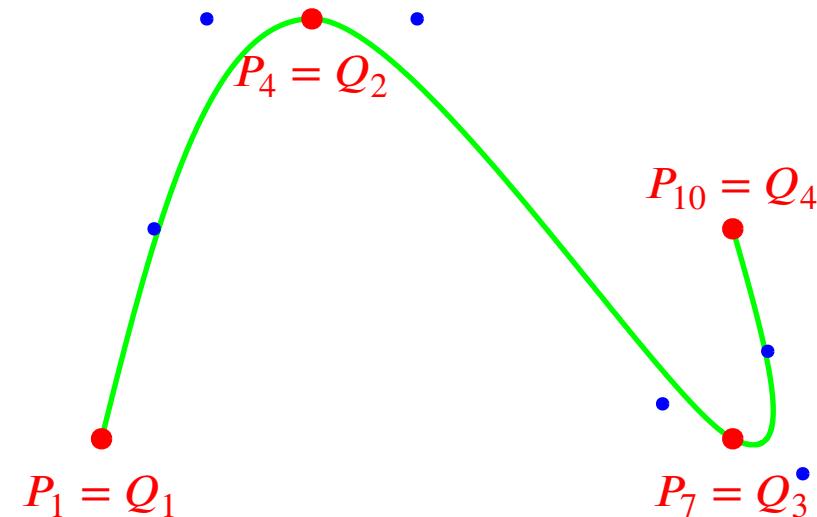
für $i = 2, \dots, l-1$

$$P_{3i-1} := Q_i + \frac{1}{6}(Q_{i+1} - Q_{i-1})$$

$$P_{3i-3} := Q_i - \frac{1}{6}(Q_{i+1} - Q_{i-1})$$

$$P_2 := \frac{1}{2}(Q_1 + P_3)$$

$$P_{3l-3} := \frac{1}{2}(P_{3l-4} + Q_l)$$



Verallgemeinerung: Bézier-Kurven beliebigen Grades

gegeben: Punkte P_1, \dots, P_{n+1}

Bézier-Kurve n -ten Grades:

$$Q(t) = \sum_{i=0}^n P_{i+1} B_{i,n}(t)$$

mit Bernsteinpolynomen

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

Beobachtung: $B_{i,n}(t) \geq 0$ für $t \in [0; 1]$ und $\sum_{i=0}^n B_{i,n}(t) = 1$

⇒ Bézier-Kurve verläuft in konvexer Hülle der Kontrollpunkte

(nützlich für Clipping-Verfahren)

Rekursive Berechnung der Bernsteinpolynome:

$$B_{01}(t) = 1 - t, \quad B_{11}(t) = t$$

$$B_{i,n}(t) = (1 - t)B_{i,n-1}(t) + tB_{i-1,n-1}(t)$$

mit $B_{-1,n-1}(t) = B_{n,n-1}(t) := 0$

9.1.3 Kubische Splines

Ein **kubischer B-Spline** approximiert eine Folge von $m + 1$ Kontrollpunkten P_0, \dots, P_m , $m \geq 3$. („B“ steht für Basis.)

Die Kurve besteht aus $m - 2$ **Kurvensegmenten** Q_3, \dots, Q_m (kubische Polynome).

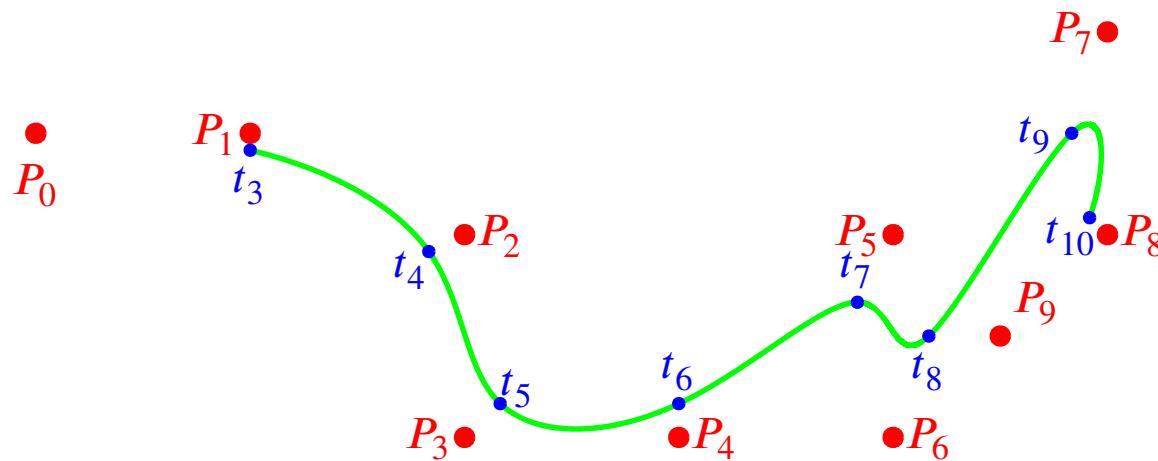
Jedes Segment Q_i ist definiert auf einem Parameterbereich $[t_i; t_{i+1}]$ ($i = 3, \dots, m$).

Bemerkung 9.3: Spezialfall $m = 3$:

4 Kontrollpunkte P_0, \dots, P_3 , ein Polynom Q_3 mit $t_3 \leq t \leq t_4$

Die Verbindungspunkte $Q_{i-1}(t_i) = Q_i(t_i)$ ($i = 4, \dots, m$) sowie Anfangspunkt $Q_3(t_3)$ und Endpunkt $Q_m(t_{m+1})$ heißen **Knotenpunkte**, die t_i **Knotenwerte**.

Der B-Spline heißt **uniform**, falls alle Parameterintervalle gleich lang sind – o. B. d. A. $t_3 = 0$, $t_4 = 1$, ..., $t_m = m - 3$.



Geometrievektor zu Segment Q_i :

$$G_{BS_i} = (P_{i-3}, P_{i-2}, P_{i-1}, P_i)$$

- Jedes Segment wird von vier Kontrollpunkten beeinflusst.
- Umgekehrt beeinflusst jeder Kontrollpunkt (bis auf die ersten und letzten drei) jeweils vier Segmente.

Definition der Kurve aus

$$Q_i(t) = G_{BS_i} \cdot M_{BS} \cdot T_i$$

für $t_i \leq t \leq t_{i+1}$ mit

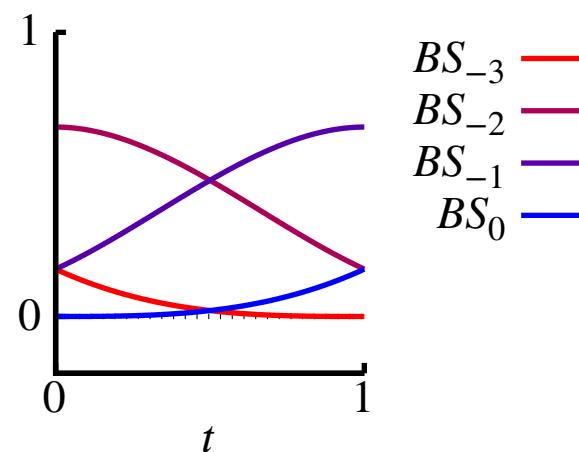
$$T_i = \begin{pmatrix} (t - t_i)^3 \\ (t - t_i)^2 \\ t - t_i \\ 1 \end{pmatrix}$$

und der **Basismatrix**

$$M_{BS} = \frac{1}{6} \cdot \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Basispolynome für Intervall $[0; 1]$:

$$\frac{1}{6} [(1-t)^3, 3t^3 - 6t^2 + 4, -3t^3 + 3t^2 + 3t + 1, t^3]$$



Eigenschaften:

- Summe der vier Basispolynome ist $\equiv 1$
 - Polynome sind ≥ 0
- ⇒ Kurve verläuft in konvexer Hülle der Kontrollpunkte

Satz 9.4: B-Splines sind C^2 -stetig.

Beweis: Für x -Komponente am Übergang Q_i/Q_{i+1} und $t \in [0; 1]$:

Kurve in Basisdarstellung:

$$Q_i(t) = G_{BS_i} \cdot M_{BS} \cdot T = \frac{(1-t)^3}{6} P_{i-3} + \frac{3t^3 - 6t^2 + 4}{6} P_{i-2} + \frac{-3t^3 + 3t^2 + 3t + 1}{6} P_{i-1} + \frac{t^3}{6} P_i$$

C^0 -Stetigkeit:

$$x_i(1) = \frac{1}{6} (P_{i-2_x} + 4P_{i-1_x} + P_{i_x}) = x_{i+1}(0)$$

C^1 -Stetigkeit:

$$\frac{d}{dt} Q(t) = \frac{-(1-t)^2}{2} P_{i-3} + \frac{3t^2 - 4t}{2} P_{i-2} + \frac{-3t^2 + 2t + 1}{2} P_{i-1} + \frac{t^2}{2} P_i$$

$$\frac{d}{dt} x_i(1) = \frac{1}{2} (-P_{i-2_x} + P_{i_x}) = \frac{d}{dt} x_{i+1}(0)$$

C^2 -Stetigkeit:

$$\frac{d^2}{dt^2}Q(t) = (1-t)P_{i-3} + (3t-2)P_{i-2} + (-3t+1)P_{i-1} + tP_i$$

$$\frac{d^2}{dt^2}x_i(1) = P_{i-2_x} - 2P_{i-1_x} + P_{i_x} = \frac{d^2}{dt^2}x_{i+1}(0)$$



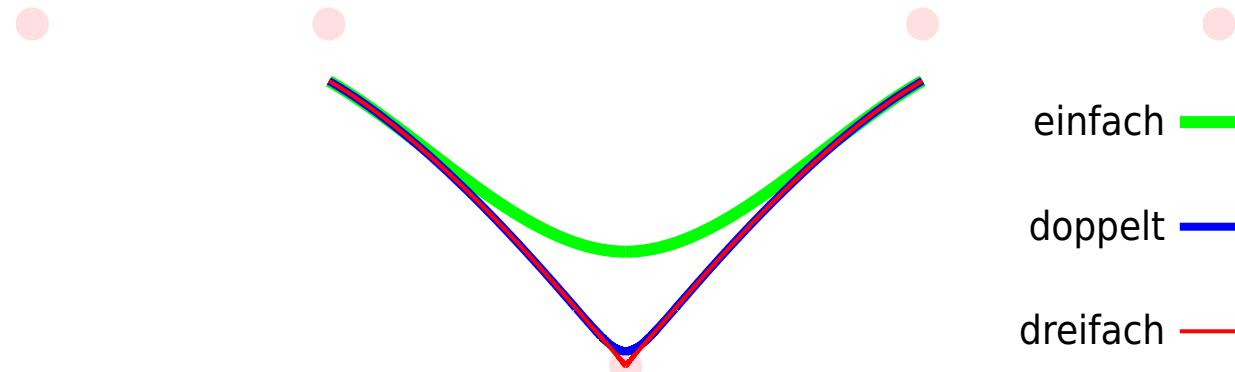
Bemerkung 9.5: C^2 -Stetigkeit wird erkauft durch Approximation statt Interpolation.

Bessere Annäherung ist möglich durch Doppelung von Kontrollpunkten: $P_{i-2} = P_{i-1}$

Interpolation durch Dreifach-Verwendung von Punkten: $P_{i-2} = P_{i-1} = P_i$

$$\Rightarrow Q_i(t) = \frac{(1-t)^3}{6}P_{i-3} + \frac{t^3 - 3t^2 + 3t + 5}{6}P_i \text{ ist eine Gerade.}$$

- C^2 -stetig mit Ableitungen 0 für $t = 1$
- ⊖ nicht G^1 -stetig, Knick!



Nicht-uniforme kubische B-Splines

Definition eines **nicht-uniformen kubischen B-Splines** zur Approximation von Kontrollpunkten P_0, \dots, P_m :

benötigt: Knotenfolge t_0, \dots, t_{m+4} mit $t_i \leq t_{i+1}$, z. B. $(0, 0, 0, 1, 1, 2, 3, 3, 3, 4)$

Definition des Kurvensegments Q_i zu Kontrollpunkten $P_{i-3}, P_{i-2}, P_{i-1}, P_i$:

$$Q_i(t) = P_{i-3} \cdot B_{i-3,4} + P_{i-2} \cdot B_{i-2,4} + P_{i-1} \cdot B_{i-1,4} + P_i \cdot B_{i,4}$$

mit Gewichtsfunktionen

$$B_{i,1}(t) = \begin{cases} 1 & \text{für } t \in [t_i; t_{i+1}] \\ 0 & \text{sonst} \end{cases}$$

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} \cdot B_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \cdot B_{i+1,k-1}(t) \quad \text{für } k > 1$$

(Wird der Nenner bei mehrfachen Knoten 0, so wird der Bruch als 0 definiert.)

Vorteil im Vergleich zu uniformen Splines:

- ⊕ Interpolation durch Dreifach-Knoten möglich, ohne dass Geradenstücke entstehen
- ⊕ gilt auch für Anfangs- und Endpunkt

Rationale kubische Kurven

Eine **rationale kubische Kurve** hat die Form

$$Q(t) = \left(x(t) = \frac{X(t)}{W(t)}, y(t) = \frac{Y(t)}{W(t)}, z(t) = \frac{Z(t)}{W(t)} \right)^T$$

mit kubischen Polynomen $X(t)$, $Y(t)$, $Z(t)$, $W(t)$.

Darstellung in homogenen Koordinaten:

$$Q(t) = (X(t), Y(t), Z(t), W(t))^T$$

Bezeichnung für nicht-uniforme rationale B-Splines: **NURBS**

Vorteile rationaler Kurven:

- ⊕ Invarianz unter perspektivischer Projektion
 - ⇒ Die Projektion kann nur auf die Kontrollpunkte angewandt werden; bei anderen Kurven müssen alle Zwischenpunkte projiziert werden.
- ⊕ exakte Darstellung von Kegelschnitten möglich, z. B. Kreis

9.1.4 Unterteilung von Kurven

Frage: Wie lässt sich eine Kurve (ein Segment) durch Hinzunahme weiterer Kontrollpunkte in zwei Teile aufteilen, ohne dass sich das Bild ändert?

Hintergrund: Dies ist sinnvoll

- bei der Modellierung, wenn eine Kurve nur in einem Bereich verändert werden soll,
- beim Zeichnen von Kurven, vgl. Abschnitt 9.1.5.

Bei Bézier-Kurven ist die Unterteilung besonders einfach. Die neuen Kontrollpunkte sind Zwischenergebnisse bei der Ausführung des **Algorithmus von de Casteljau** zur Bestimmung des Kurvenpunktes für einen Parameterwert t .

Algorithmus von de Casteljau

Eingabe Kontrollpunkte P_1, \dots, P_{n+1} , Parameter $t \in [0; 1]$

für $i = 1, \dots, n + 1$

$$P_i^0 := P_i$$

für $r = 1, \dots, n$

für $i = 1, \dots, n - r + 1$

$$P_i^r := (1 - t)P_i^{r-1} + tP_{i+1}^{r-1}$$

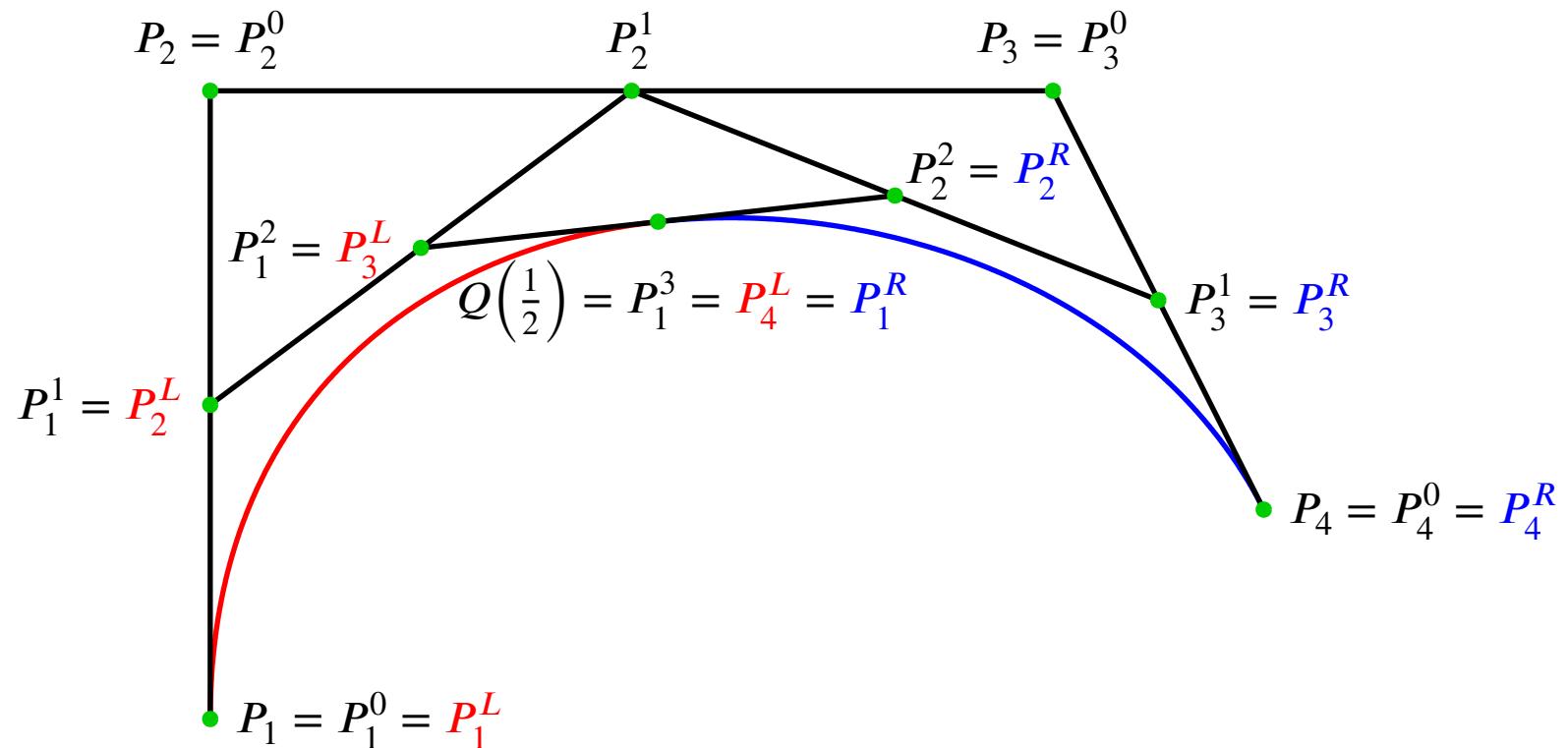
$Q(t) := P_1^n // \text{der gesuchte Punkt auf der Kurve}$

Paul de Faget de Casteljau

* 1930, Besançon, FR

Physiker, Mathematiker (Citroën)

Beispiel 9.6: $n = 3, t = \frac{1}{2}$



Kontrollpunkte für linken (roten) Teil: $P_i^L = P_1^{i-1}$; Kontrollpunkte für rechten (blauen) Teil: $P_i^R = P_i^{n+1-i}$

direkte Berechnung der neuen Kontrollpunkte im Fall einer kubischen Kurve und Unterteilung bei $t = \frac{1}{2}$:

$$G_B^L = (P_1^L, P_2^L, P_3^L, P_4^L) = (P_1, P_2, P_3, P_4) \cdot \frac{1}{8} \cdot \begin{pmatrix} 8 & 4 & 2 & 1 \\ 0 & 4 & 4 & 3 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$G_B^R = (P_1^R, P_2^R, P_3^R, P_4^R) = (P_1, P_2, P_3, P_4) \cdot \frac{1}{8} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 \\ 3 & 4 & 4 & 0 \\ 1 & 2 & 4 & 8 \end{pmatrix}$$

Bemerkung 9.7: Bei B-Splines ergeben sich aus $(P_{i-3}, P_{i-2}, P_{i-1}, P_i) \cdot \frac{1}{8} \cdot \begin{pmatrix} 4 & 1 & 0 & 0 & 0 \\ 4 & 6 & 4 & 1 & 0 \\ 0 & 1 & 4 & 6 & 4 \\ 0 & 0 & 0 & 1 & 4 \end{pmatrix}$ fünf neue Kontrollpunkte, die zwei Segmente definieren, die mit dem ursprünglichen übereinstimmen.

aber: Die Nachbarsegmente haben noch die alten Kontrollpunkte.

⊖ Verschiebung einzelner Punkte bei Modellierung führt zu Unstetigkeiten

9.1.5 Zeichnen von Kurven

Möglichkeit 1: Zerlegung des Parameterintervalls in kleine Teilintervalle, Annäherung der Kurve durch Geradenstücke auf jedem Teilintervall

Polynomauswertung durch

- Horner-Schema oder
- Vorwärtsdifferenzen (dritter Ordnung) (effizienter)

William George Horner
 * 1786, Bristol
 † 1837, Bath
 Mathematiker, Lehrer

Möglichkeit 2: rekursive Unterteilung der Kurve; wenn Kurve flach genug, Approximation durch Linie

9.2 Parametrisierte bikubische Flächen

Erinnerung: Form einer kubischen Kurve (hier mit s statt t):

$$Q(s) = G \cdot M \cdot S \quad \text{mit } G = (G_1, G_2, G_3, G_4) \text{ und } S = \begin{pmatrix} s^3 \\ s^2 \\ s \\ 1 \end{pmatrix}$$

Verändere nun die G_i selbst entlang einer Kurve:

$$G_i(t) = \tilde{G}_i \cdot M \cdot T \quad \text{mit } \tilde{G}_i = (G_{1i}, G_{2i}, G_{3i}, G_{4i})$$

Alle Kurven zusammen ergeben eine Fläche.

Setzt man die transponierte Form $G_i(t)^T = T^T \cdot M^T \cdot \tilde{G}_i^T$ in die Kurvendarstellung ein, ergibt sich die **parametrisierte bikubische Fläche** aus

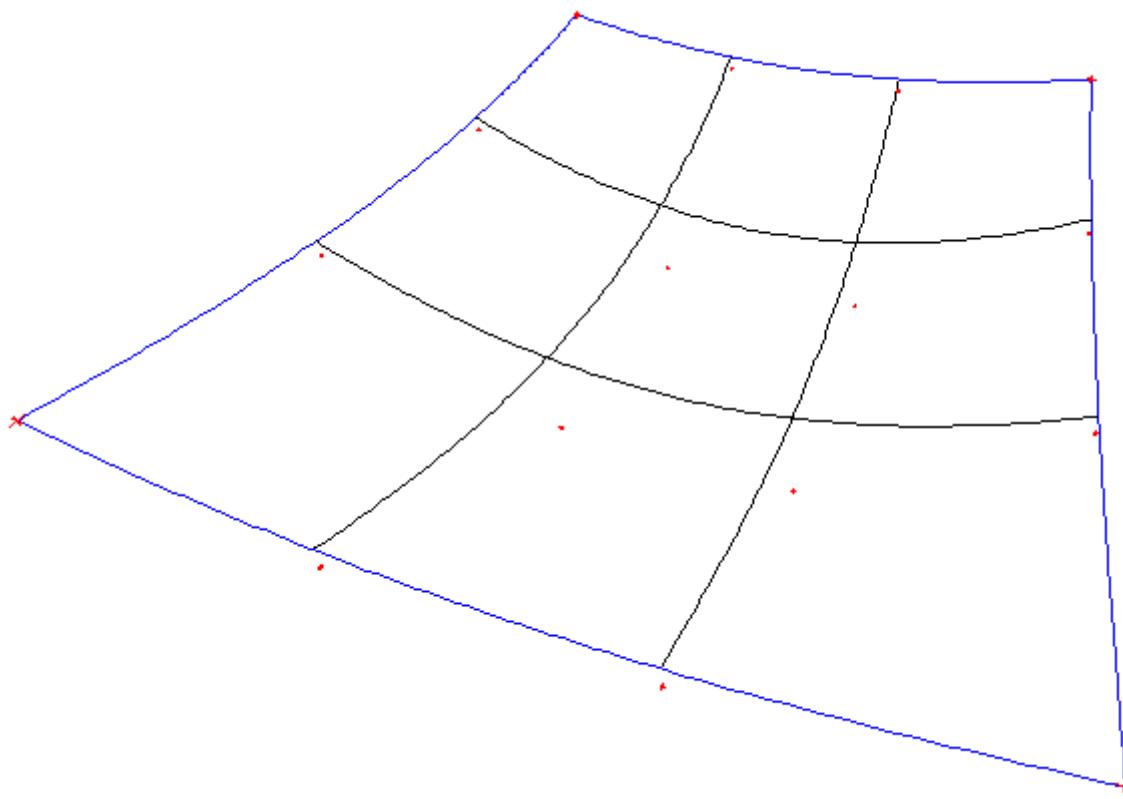
$$Q(s, t) = T^T \cdot M^T \cdot \tilde{G} \cdot M \cdot S \quad \text{mit } \tilde{G} = \begin{pmatrix} G_{11} & G_{12} & G_{13} & G_{14} \\ G_{21} & G_{22} & G_{23} & G_{24} \\ G_{31} & G_{32} & G_{33} & G_{34} \\ G_{41} & G_{42} & G_{43} & G_{44} \end{pmatrix}, \quad s, t \in [0; 1]$$

oder koordinatenweise:

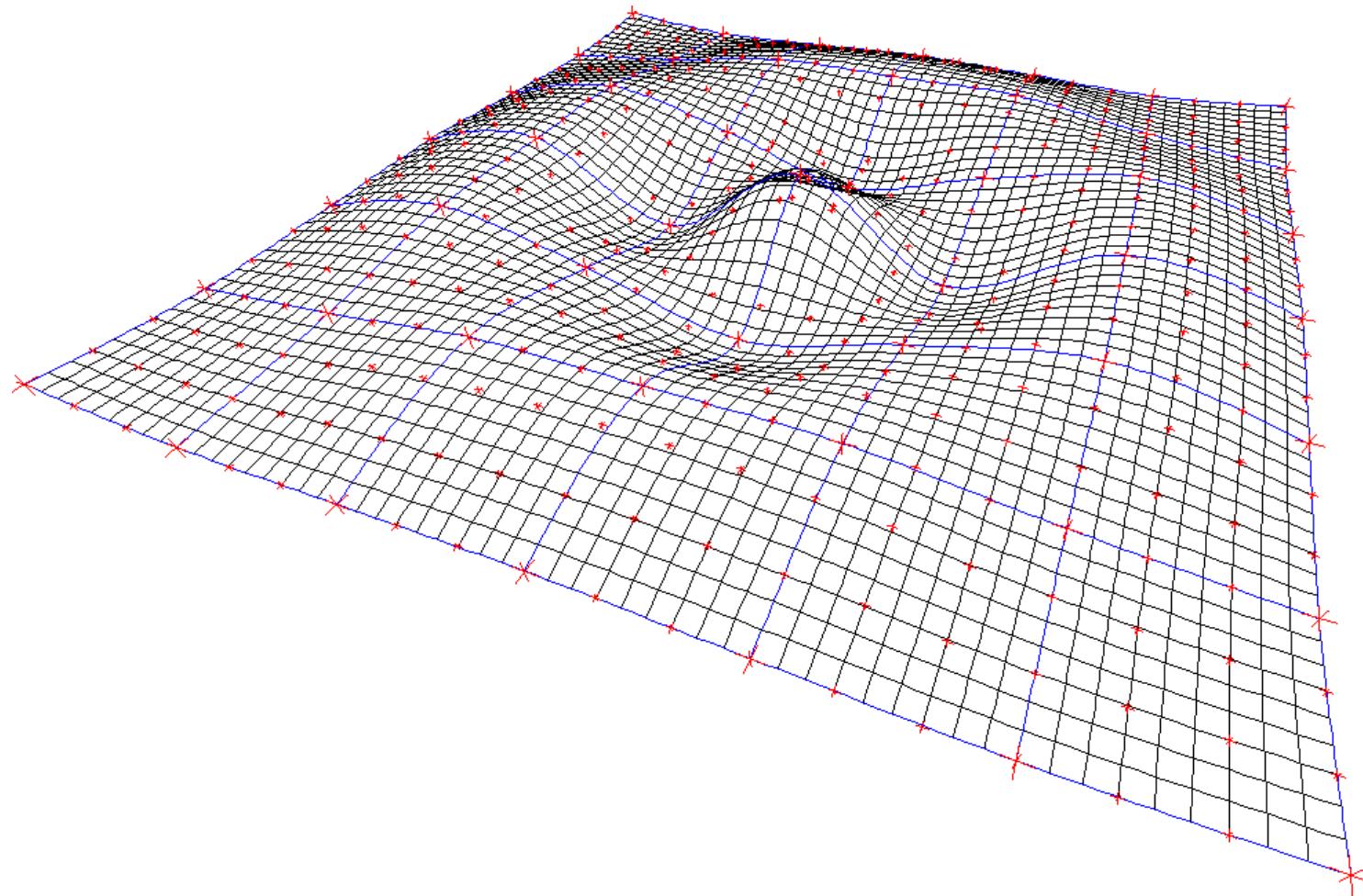
$$x(s, t) = T^T \cdot M^T \cdot \tilde{G}_x \cdot M \cdot S, \quad y(s, t) = T^T \cdot M^T \cdot \tilde{G}_y \cdot M \cdot S, \quad z(s, t) = T^T \cdot M^T \cdot \tilde{G}_z \cdot M \cdot S$$

9.2.1 Bézier-Flächen

G_B enthält $4 \cdot 4 = 16$ Kontrollpunkte; die vier Eckpunkte des Flächenstücks werden interpoliert.

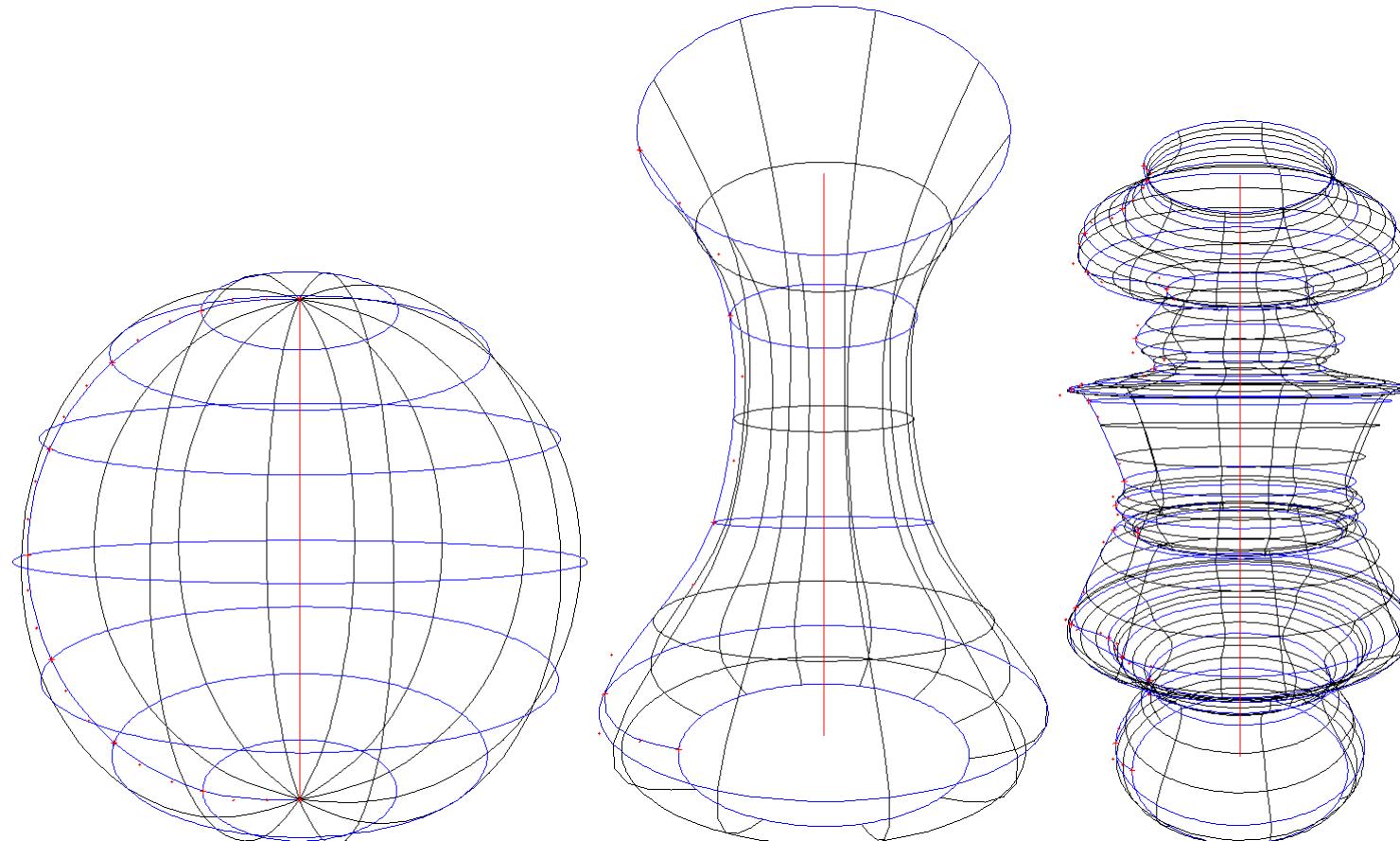


Beispiel 9.8: **Bézier-Fläche**, die 7×7 gegebene Punkte mit 36 Flächenstücken (**Patches**) interpoliert. Berechnung der inneren Kontrollpunkte ähnlich zum Algorithmus auf 9-11:



9.3 Rotationskörper

Lasse eine (z. B. kubische parametrisierte) Kurve um eine Achse rotieren.



10 Färbung II

Inhalt

10.1 Rekursives Raytracing	10-3
10.1.1 Das Brechungsgesetz	10-4
10.1.2 Der Raytracing-Ansatz nach Whitted	10-5
10.1.3 Beschleunigung der Strahlverfolgung (I)	10-13
10.1.4 Beschleunigung der Strahlverfolgung (II)	10-25
10.1.5 Selektives Raytracing	10-32
10.1.6 Bemerkungen	10-33
10.2 Radiosity-Verfahren	10-34
10.2.1 Raumwinkel	10-37
10.2.2 Radiosity	10-39
10.2.3 Von den Objekten zum Gleichungssystem	10-41
10.2.4 Vom Gleichungssystem zum Bild	10-45
10.2.5 Die Formfaktoren	10-47
10.2.6 Numerische Berechnung der Formfaktoren	10-50
10.2.7 Schrittweiser Lösungsaufbau	10-63
10.2.8 Unterteilung der Patches	10-67
10.2.9 Zusammenfassung	10-71
10.3 Rendergleichung und Monte-Carlo-Integration	10-73

10.3.1 Radiance	10-74
10.3.2 Rendergleichung	10-75
10.3.3 Zusammenhang zwischen Rendergleichung und Radiosity	10-79
10.3.4 Monte-Carlo-Integration	10-81
10.3.5 Erweiterungen des Raytracing-Modells	10-82
10.3.6 Path Tracing	10-84
10.3.7 BRDF	10-85
10.3.8 Kombination von BRDFs	10-89
10.3.9 Importance Sampling	10-90
10.3.10 Direkte Beleuchtung	10-91
10.3.11 Multiple Importance Sampling	10-92
10.3.12 Weiterentwicklungen	10-94

10.1 Rekursives Raytracing

Ziel: einheitlicher Ansatz für:

- Sichtbarkeitsanalyse
 - Schatten
 - Reflexion, Spiegelungen
 - Refraktion (Brechung)
-

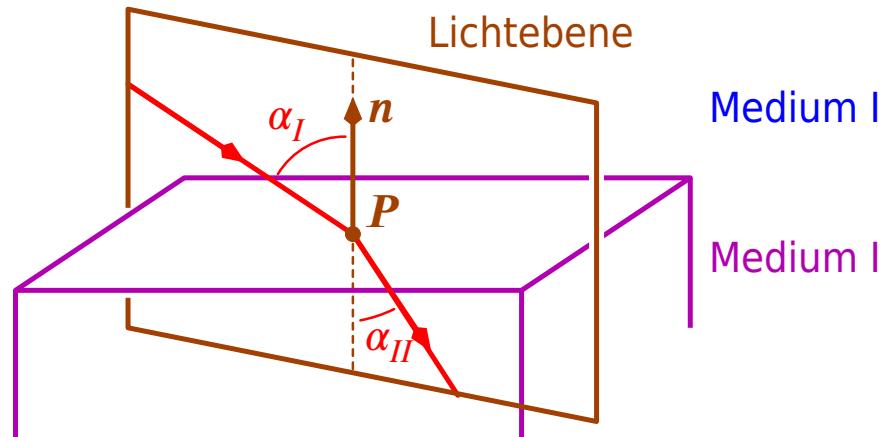
Bemerkung 10.1: Schattenwurf, Spiegelungen und Lichtbrechung entstehen dadurch, dass das von der Lichtquelle kommende Licht das sichtbare Objekt O_k nicht direkt erreicht (Interaktion mit anderen Objekten).

⇒ Diese Effekte können nur mit einem **nicht-lokalen Modell (Modell der Ordnung > 0)** berücksichtigt werden.

10.1.1 Das Brechungsgesetz

Einfallender und gebrochener Strahl liegen in derselben Ebene, und es gilt

$$\frac{\sin \alpha_I}{\sin \alpha_{II}} = \frac{\eta_I}{\eta_{II}} \quad (\text{Brechungsindex von Medium I})$$
$$(\text{Brechungsindex von Medium II}).$$



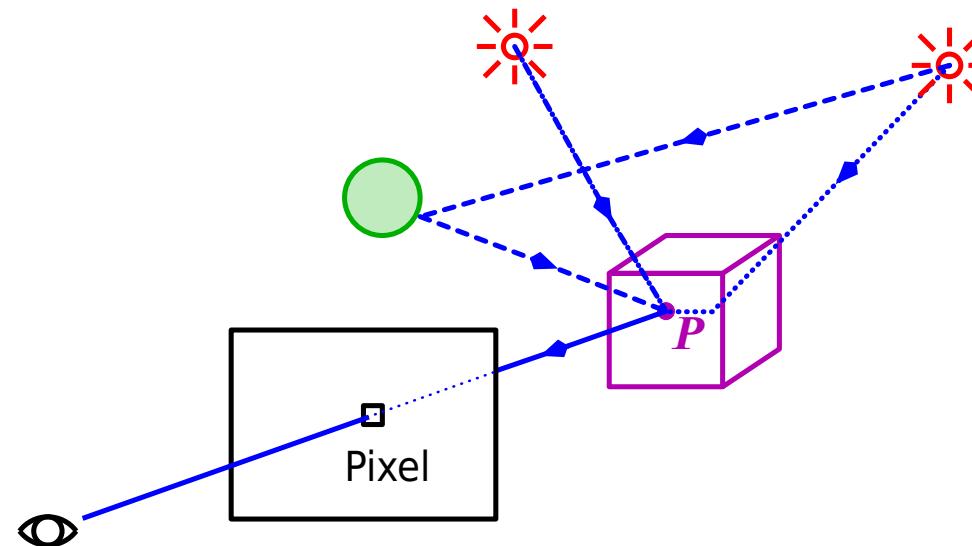
Bemerkung 10.2: η hängt i. Allg. von der Wellenlänge ab.

10.1.2 Der Raytracing-Ansatz nach Whitted

Beobachtung: Bei einem bestimmten Pixel sei der Punkt P eines Objekts sichtbar. Das von P ins Auge fallende Licht setzt sich zusammen aus

- dem bei P einfallenden, ins Auge **reflektierten** Licht und
- dem bei P austretenden, in Augenrichtung **gebrochenen** Licht

(jeweils direkt von den Lichtquellen und/oder bereits vorher – evtl. mehrfach – gespiegelt und/oder gebrochen).



John Turner Whitted
* ?, Durham, North Carolina
Elektroingenieur, Informatiker

Ansatz

$$I(\mathbf{P}', \mathbf{P}) = g(\mathbf{P}', \mathbf{P}) \cdot \left\{ \varepsilon(\mathbf{P}', \mathbf{P}) + \int_{\mathbb{R}^3} \rho(\mathbf{P}'', \mathbf{P}', \mathbf{P}) I(\mathbf{P}'', \mathbf{P}') d\mathbf{P}'' \right\} \quad (10.1)$$

$I(\mathbf{P}', \mathbf{P})$: Intensität des vom Punkt \mathbf{P}' nach \mathbf{P} gestrahlten Lichts

$g(\mathbf{P}', \mathbf{P})$: „Dämpfungsfaktor“
 $= 0$, falls \mathbf{P}' von \mathbf{P} aus nicht sichtbar ist,
 proportional zu $\frac{1}{\|\mathbf{P}' - \mathbf{P}\|_2^2}$ sonst

$\varepsilon(\mathbf{P}', \mathbf{P})$: Eigenemission von \mathbf{P}' nach \mathbf{P}

$I(\mathbf{P}'', \mathbf{P}')$: Intensität des bei \mathbf{P}' von einem Punkt \mathbf{P}'' einfallenden Lichts

$\rho(\mathbf{P}'', \mathbf{P}', \mathbf{P})$: Reflexions-/Brechungskoeffizient für das von \mathbf{P}'' bei \mathbf{P}' eintreffende und nach \mathbf{P} weitergegebene Licht

Bemerkung 10.3: In dieser Formulierung werden Lichtquellen wie alle anderen Objekte behandelt.

Problem: Die kontinuierliche Formulierung (10.1) ist für die praktische Rechnung nicht geeignet.

Erste Umformulierung

$$I(\mathbf{r}, \mathbf{P}) = g(\mathbf{P}', \mathbf{P}) \cdot \left\{ \varepsilon(\mathbf{P}', \mathbf{P}) + \int \rho(\mathbf{r}', \mathbf{P}', -\mathbf{r}) I(\mathbf{r}', \mathbf{P}') d\mathbf{r}' \right\} \quad (10.2)$$

$I(\mathbf{r}, \mathbf{P})$: aus Richtung \mathbf{r} bei \mathbf{P} eintreffendes Licht

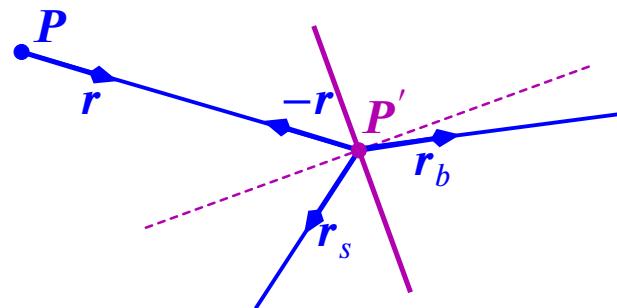
\mathbf{P}' : der erste auf einem Objekt liegende Punkt, wenn man von \mathbf{P} aus in Richtung \mathbf{r} geht

$\rho(\mathbf{r}', \mathbf{P}', -\mathbf{r})$: Reflexions-/Brechungskoeffizient für das aus Richtung \mathbf{r}' bei \mathbf{P}' eintreffende und in Richtung $-\mathbf{r}$ weitergeleitete Licht

$d\mathbf{r}'$: Integration über alle möglichen Richtungen

Vereinfachungen

- Berücksichtige in (10.2) nicht alle Richtungen \mathbf{r}' sondern nur die beiden durch Spiegel- und Brechungsgesetz gegebenen Richtungen:

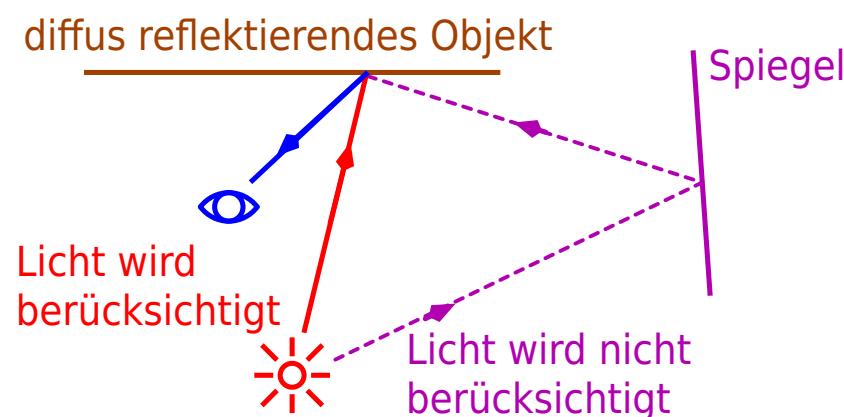


(Aus Richtung \mathbf{r}_s bzw. \mathbf{r}_b einfallendes Licht wird in Richtung $-\mathbf{r}$ gespiegelt bzw. gebrochen.)

- Lichtquellen werden speziell behandelt.
 - $\epsilon \equiv 0$ (Die betrachteten Objekte emittieren nicht selbst.)
 - Für das **direkt** von den Lichtquellen bei P' eintreffende Licht wird eine Kombination von ambientem, diffusem und winkelabhängigem Anteil angesetzt.
-

Bemerkungen 10.4:

1. Lichtbrechung und Spiegelung an Objekten werden „zu perfekt“ nachgebildet. (In der Realität sind die Oberflächen nicht ganz glatt, d. h. nach mehrfacher Spiegelung/Brechung „verschwimmt“ das Bild des Objekts.)
2. Diffuse Reflexion wird nicht korrekt modelliert:



Algorithmus 10.5:

Algorithmus Bilderzeugung mit rekursivem Raytracing

für alle Pixel derPixmap

P := ein zum Pixel gehöriger Punkt in der Projektionsebene
 r := die zu diesem Punkt gehörige „Blickrichtung“
Pixelwert := $I(r, P, 1)$

Algorithmus 10.6:

Algorithmus $I(r, P, t)$

// bestimmt, wie viel Licht aus Richtung r beim Punkt P einfällt; t ist die Rekursionstiefe

verfolge den von P ausgehenden Strahl in Richtung r bis zum nächstgelegenen Schnittpunkt P'

mit einem Objekt O_k

wenn es keinen solchen Schnittpunkt gibt

$I(r, P, t) :=$ „Hintergrund“

sonst

$I := I_a \cdot R_{k,a}$ // **ambienter Anteil**

// I_a : Intensität des ambienten Lichts

$R_{k,a}$: Reflexionskoeffizient für ambientes Licht

// Fortsetzung von „sonst“
 // diffuser und winkelabhängiger „direkter“ Anteil
 für alle Lichtquellen L_l
 wenn P' von L_l aus sichtbar ist

$$I := I + g(L_l, P') \cdot I_l \cdot \left\{ (\mathbf{n}_{P'}^T \cdot \mathbf{r}_l) \cdot R_{k,d} + (-\mathbf{r}^T \cdot \mathbf{s}_l)^{\nu_k} \cdot R_{k,w} \right\}$$

// $g(L_l, P')$: Dämpfungsfaktor (abhängig von Entfernung, Medium und λ)

I_l : Intensität von L_l

$\mathbf{n}_{P'}$: Normale an O_k in P'

\mathbf{r}_l : Richtung von P' nach L_l

$R_{k,d}$: Reflexionskoeffizient diffus

$-\mathbf{r}$: „Blickrichtung“ von P' aus

\mathbf{s}_l : Spiegelrichtung für das von L_l bei P' eintreffende Licht

$R_{k,w}$: Reflexionskoeffizient winkelabhängig

wenn O_k spiegelt und $t < t_{\max}$ // Spiegelung

// t_{\max} : maximale Rekursionstiefe

berechne die zu r gehörige Spiegelrichtung r_s

$$I := I + \mathbf{I}(r_s, P', t + 1) \cdot R_{k,s} \quad // \text{Rekursion!}$$

// $R_{k,s}$: Spiegelungskoeffizient

wenn O_k Licht bricht und $t < t_{\max}$ // Lichtbrechung

berechne die zu r gehörige Brechungsrichtung r_b

$$I := I + \mathbf{I}(r_b, P', t + 1) \cdot R_{k,b} \quad // \text{Rekursion!}$$

// $R_{k,b}$: Brechungskoeffizient

$$\mathbf{I}(r, P, t) := I$$

Bemerkungen 10.7:

1. Die Lichtstrahlen werden vom Betrachter aus bis zur Lichtquelle zurückverfolgt (entgegen der Ausbreitungsrichtung des Lichts).
 2. t_{\max} gibt an, wie viele Interaktionen mit Objekten auf dem Weg von der Lichtquelle bis zum Betrachter berücksichtigt werden.
- Für $t_{\max} = 1$ ergibt sich im Wesentlichen das Beleuchtungsmodell von Phong (mit integrierter Schattenanalyse).
3. Die Operationen

„verfolge den von P ausgehenden Strahl in Richtung $r \dots$ “

und

„wenn P' von L_l aus sichtbar ist“

sind sehr ähnlich. In beiden Fällen muss festgestellt werden, welches Objekt der Strahl zuerst trifft.

Diese beiden Operationen verbrauchen den größten Teil der gesamten Rechenzeit.

„naiver Ansatz“: Prüfe für alle Objekte O_k , ob sie vom Strahl getroffen werden, und bestimme ggf. den nächstgelegenen Schnittpunkt.

typische Größenordnungen:

$$p = \text{Anzahl der Pixel in derPixmap} = 10^6$$

$$n = \text{Anzahl der Objekte} = 10^5$$

$$m = \text{Anzahl der Lichtquellen} = 3$$

$$t_{\max} = \text{maximale Rekursionstiefe} = 5$$

⇒ Es müssen insgesamt bis zu

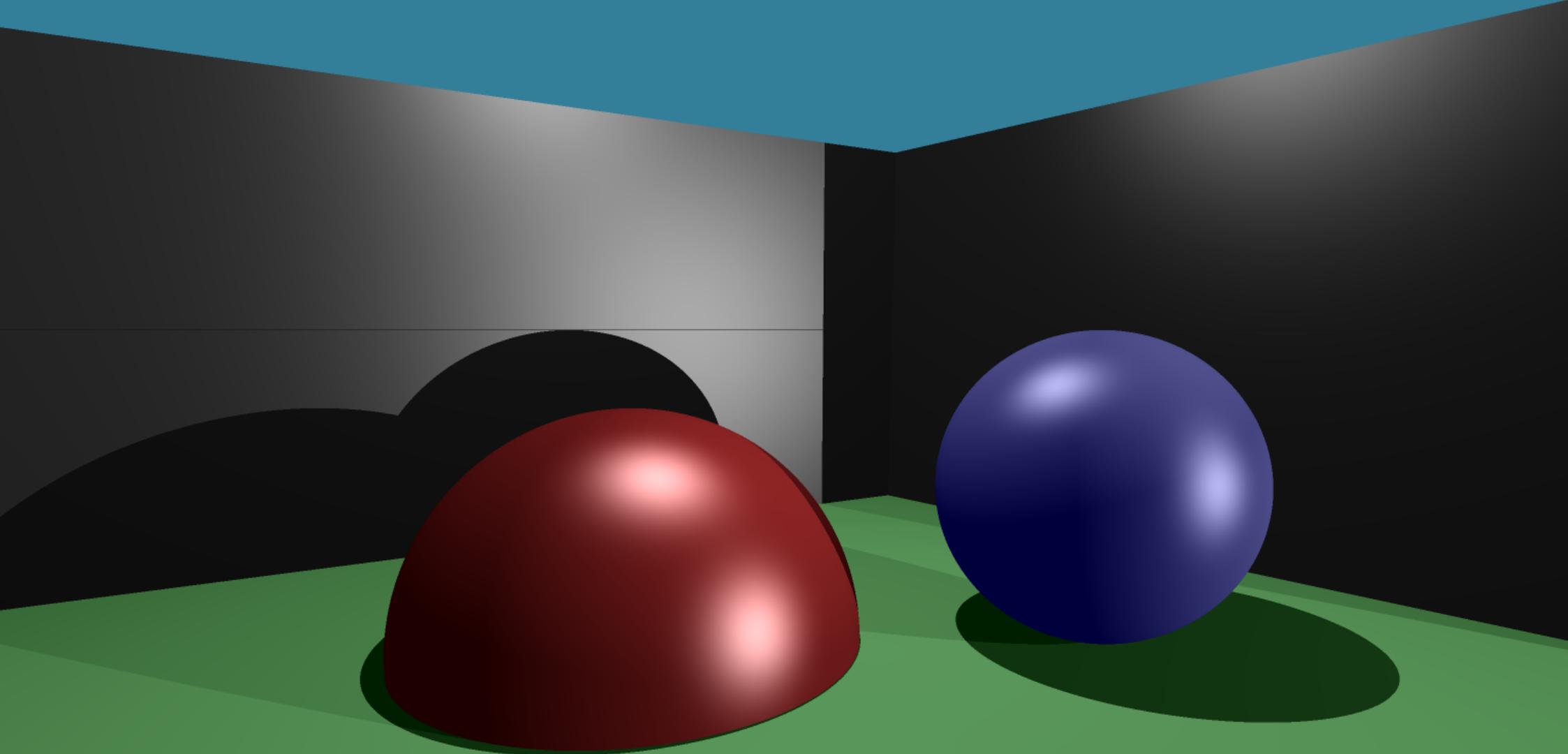
$$p \cdot \underbrace{(2^{t_{\max}} - 1)(m + 1)}_{\text{Aufspaltung des zum Pixel gehörenden Strahls}}$$

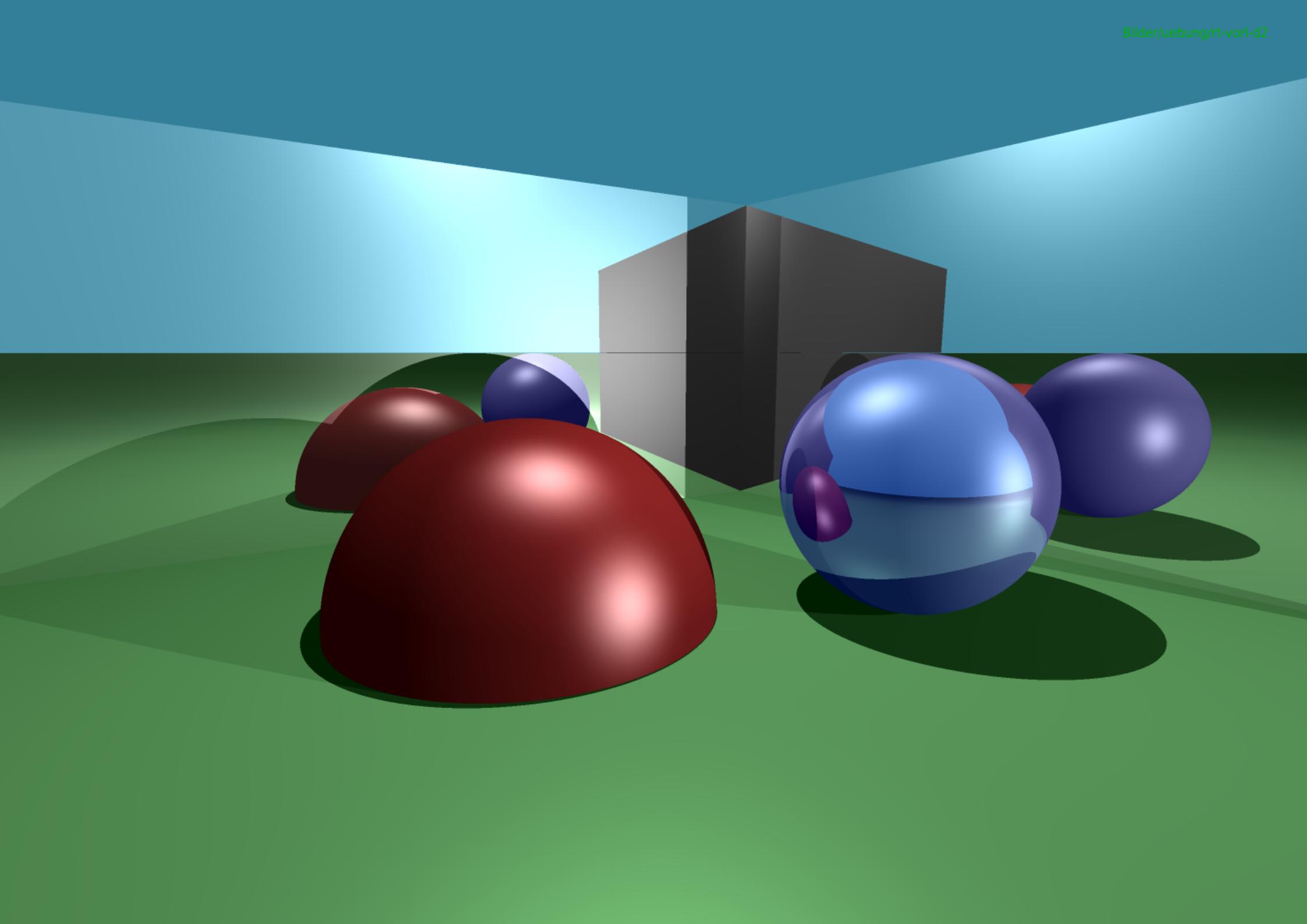
Strahlen verfolgt werden.

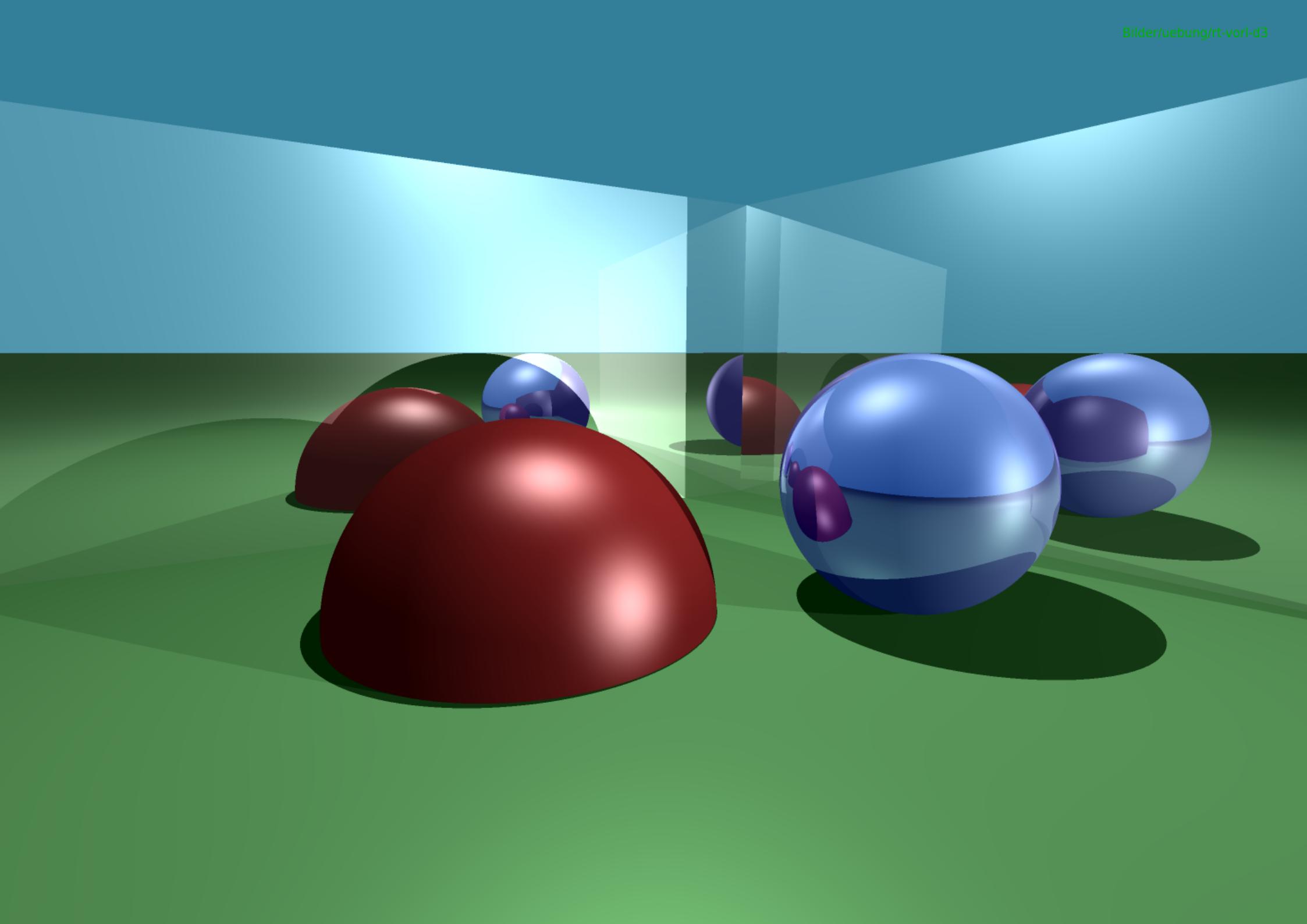
⇒ bis zu

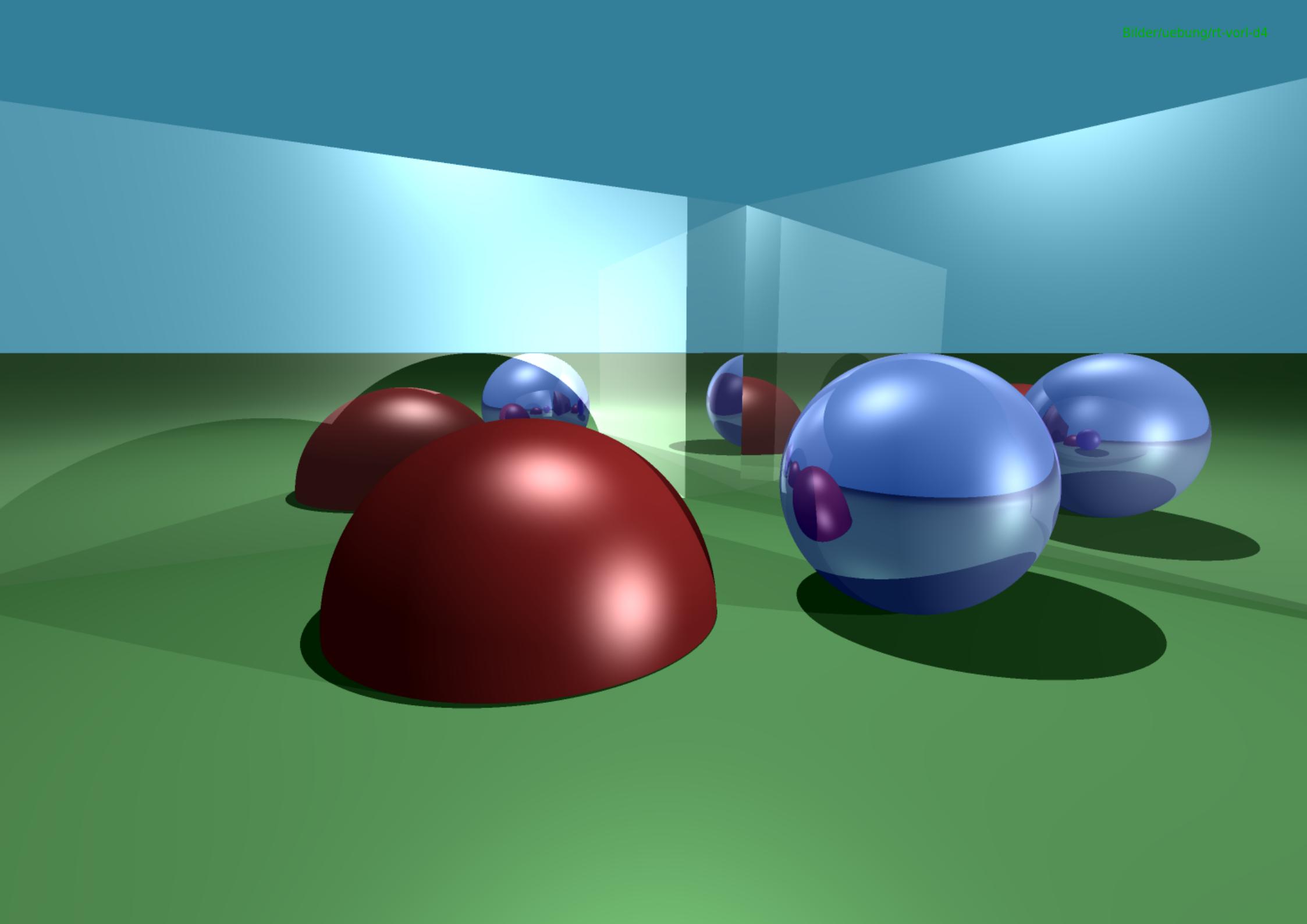
$$\begin{aligned} p \cdot (2^{t_{\max}} - 1)(m + 1) \cdot n \\ = 10^6 \cdot 124 \cdot 10^5 \\ \approx \mathbf{10^{13} \text{ Schnitttests!}} \end{aligned}$$

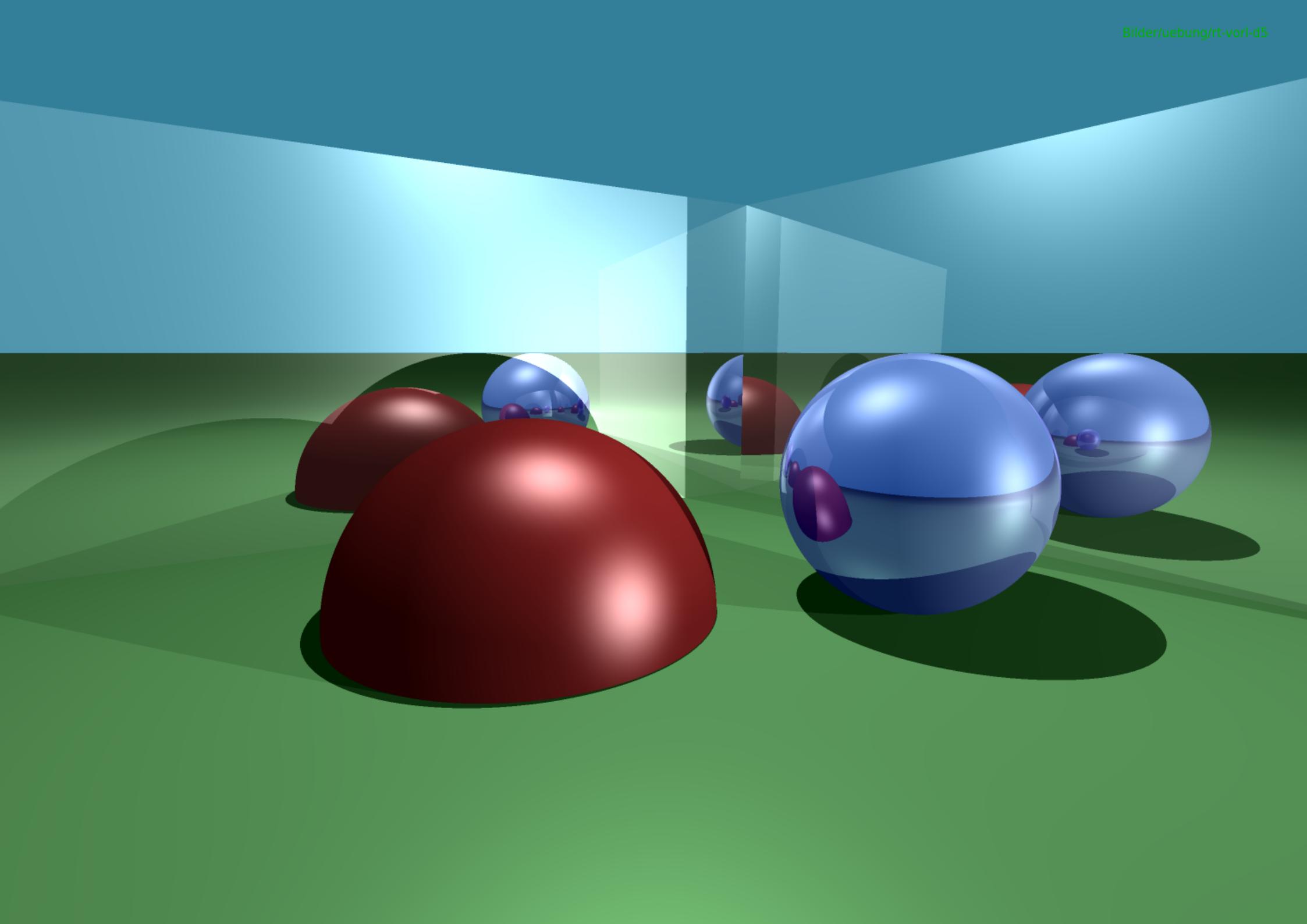
⇒ Raytracing ist - naiv implementiert - nicht praktikabel.

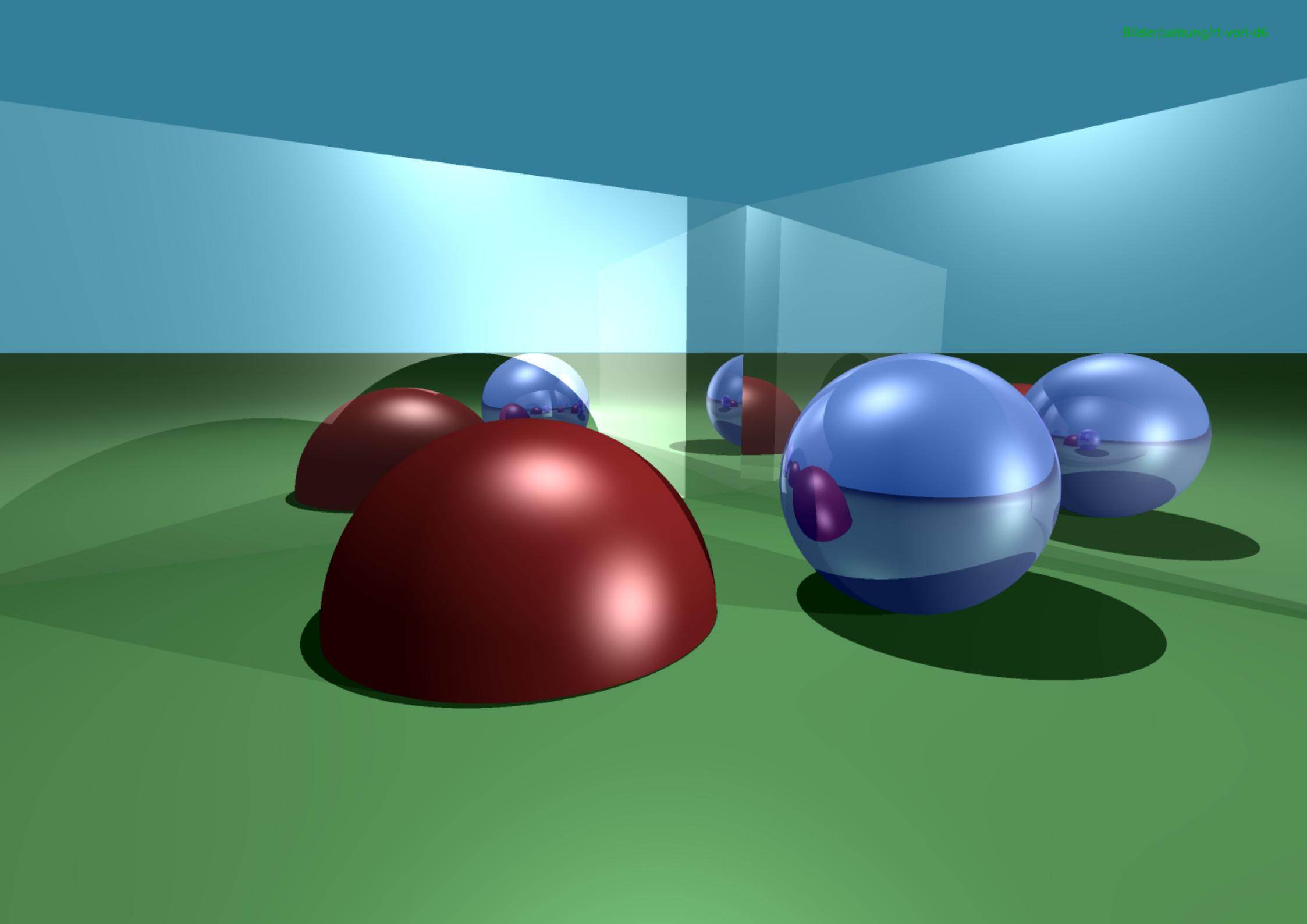












10.1.3 Beschleunigung der Strahlverfolgung (I)

Motivation: Der zeitaufwändigste Teil des Raytracing ist die Bestimmung der Schnittpunkte der Strahlen mit den jeweils nächstgelegenen Objekten.

- ⇒ Versuche, die Anzahl der erforderlichen Schnitttests zu reduzieren (nicht für jeden Strahl alle Objekte testen).
- ⇒ **Vorverarbeitung** notwendig

Ansatz: Unterteile die Szene in Gebiete G_i und bestimme für jedes Gebiet die Menge L_i der (zumindest teilweise) in G_i liegenden Objekte.

- ⇒ Ein Strahl kann Objekt O_k nicht treffen, wenn O_k nicht in den Listen L_i der vom Strahl berührten Gebiete G_i liegt.

Fragen:

- Wie sind die G_i zu wählen?
- Wie bestimmt man die vom Strahl berührten G_i ?

Zellraster-Unterteilung

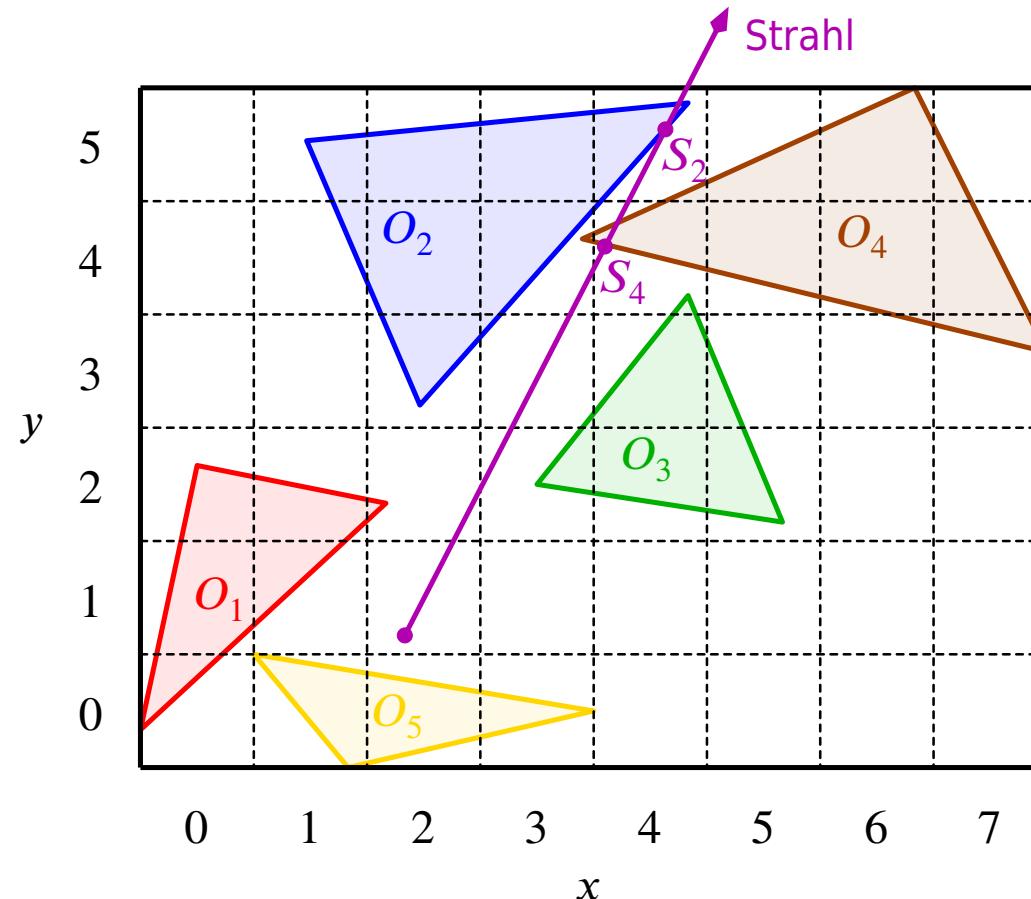
(vgl. Abschnitt 6.1.2)

Vorverarbeitung:

- Bestimme eine **Bounding Box** Q der Szene (achsenparalleler Quader, der alle Objekte O_k – einschließlich Lichtquellen – enthält).
- Unterteile Q in $r_x \times r_y \times r_z$ Zellen Q_{xyz} (Quader **gleicher Größe**).
- Bestimme für jedes Objekt O_k die vom Objekt berührten Zellen (\triangleq **3D Scan Conversion**) und trage O_k in die entsprechenden Listen L_{xyz} ein.

eigentliche Strahlverfolgung:

- Bestimme der Reihe nach die vom Strahl berührten Zellen (\triangleq **inkrementelle 3D Scan Conversion**).
- Führe für jede solche Zelle Q_{xyz} Schnitttests mit den Objekten in L_{xyz} durch.
(evtl. nur für die Objekte, die nicht schon in anderen Zellen geprüft wurden)

Beispiel (2D):

Octree-Unterteilung

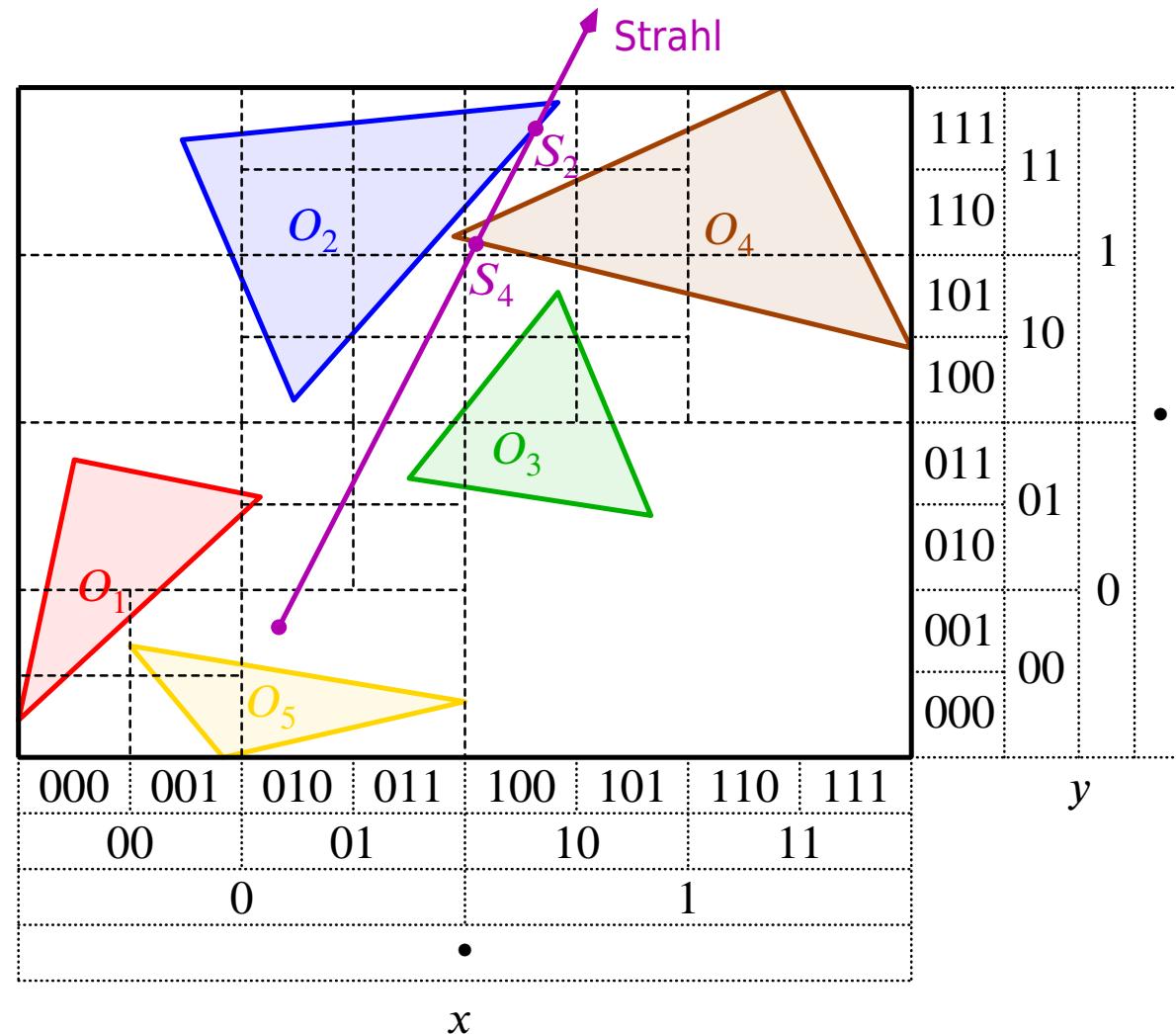
Idee: Führe die Unterteilung **adaptiv** durch, d. h. unterteile dort feiner, wo sich viele (kleine) Objekte befinden.

Vorverarbeitung:

- Bestimme eine Bounding Box Q_{\dots} der Szene und setze $L_{\dots} := \{\text{alle Objekte (einschl. Lichtquellen)}\}$.
- Solange es noch einen Quader Q_{xyz} gibt, der „groß genug“ ist und „zu viele“ Objekte enthält,
zerlege Q_{xyz} in **acht** Quader $Q_{x'y'z'}$,
bestimme für jeden dieser Teilquader die Menge $L_{x'y'z'} := \{O_k \in L_{xyz} \mid O_k \text{ trifft } Q_{x'y'z'}\}$.

eigentliche Strahlverfolgung:

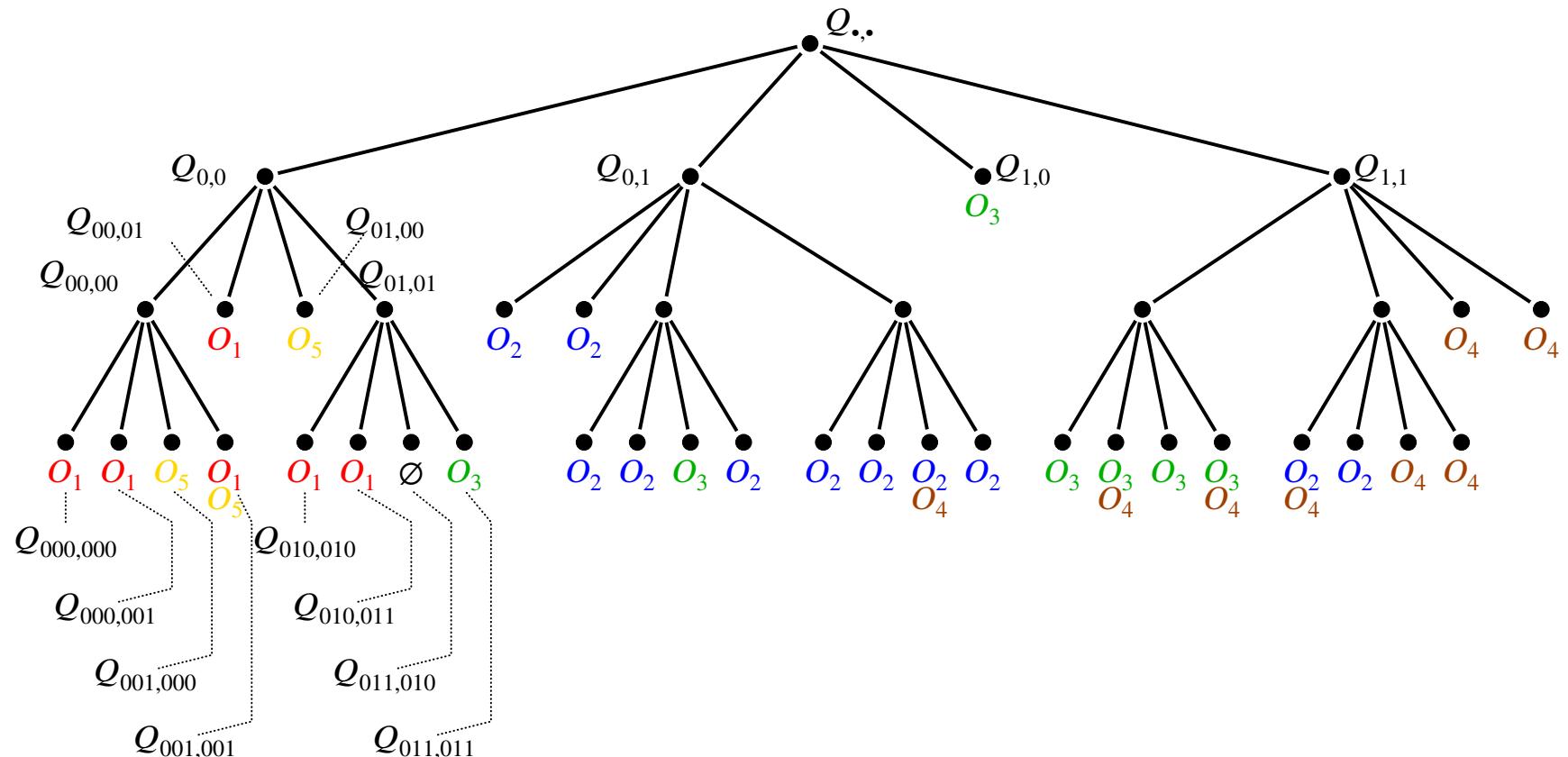
- ähnlich wie bei der Zellraster-Unterteilung (mit modifizierter Scan Conversion)

Beispiel (2D):

Kriterien für die Unterteilung in diesem Beispiel:

- „zu viele“ Objekte: mehr als eines
- „groß genug“: in jeder Koordinatenrichtung in höchstens acht gleiche Teile zerlegen

zugehöriger **Quadtree**:



durchlaufene Zellen Q_{xy}	L_{xy}	Schnitttests mit	Schnittpunkte
$Q_{01,00}$	O_5	O_5	
$Q_{010,010}$	O_1	O_1	
$Q_{010,011}$	O_1		
$Q_{011,011}$	O_3	O_3	
$Q_{011,100}$	O_3		
$Q_{011,101}$	O_2	O_2	S_2^*
$Q_{100,101}$	O_3, O_4	O_4	S_4^*
$Q_{100,110}$	O_2, O_4		

*: bisher nächstgelegener Schnittpunkt

Bemerkung 10.8: Die Strahlverfolgung springt i. Allg. zwischen verschiedenen Niveaus des Octrees hin und her (Strahl läuft durch unterschiedlich große Zellen).

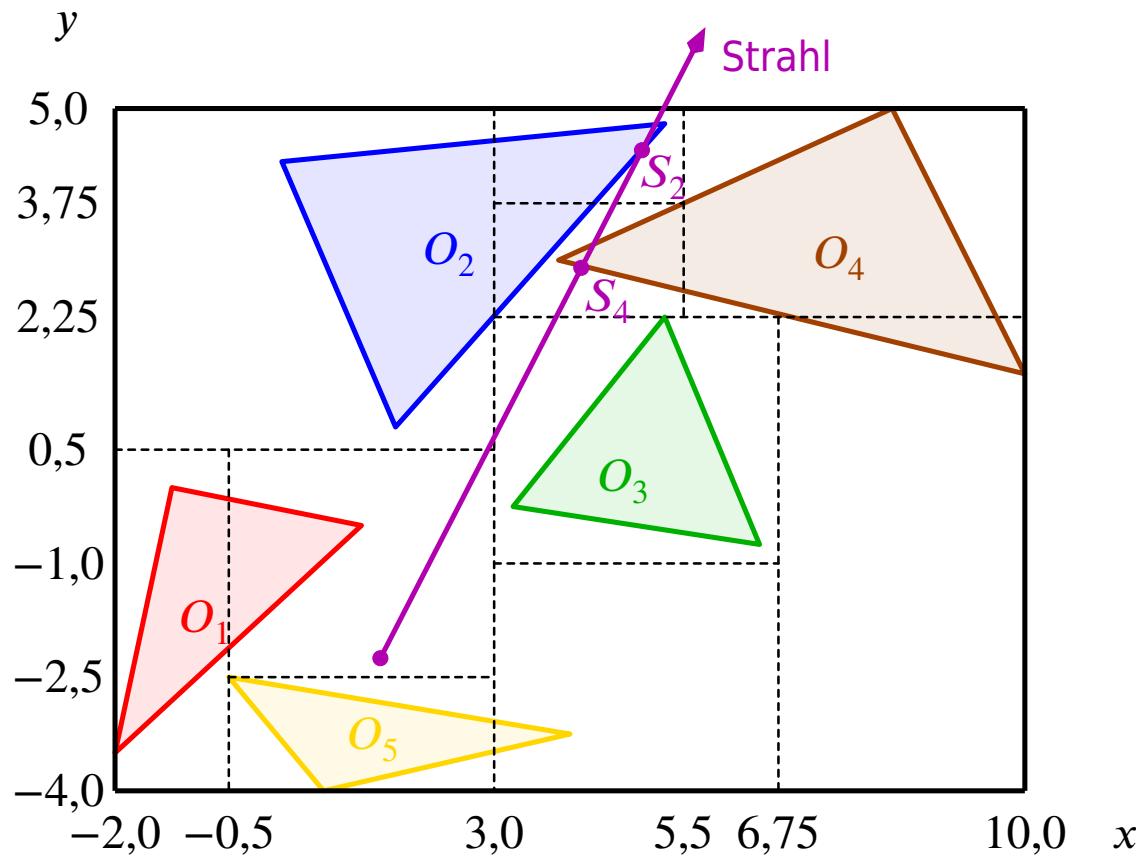
Median-Unterteilung

Idee: Versuche so zu unterteilen, dass die Teilgebiete (etwa) gleich viele Objekte treffen (aber kein „Normmaß“ mehr besitzen).

Unterteile zuerst in x -Richtung, dann jede der beiden „Hälften“ (wenn nötig) in y -Richtung, dann die „Viertel“ in z -Richtung, dann wieder in x -Richtung, usw.

Vorverarbeitung:

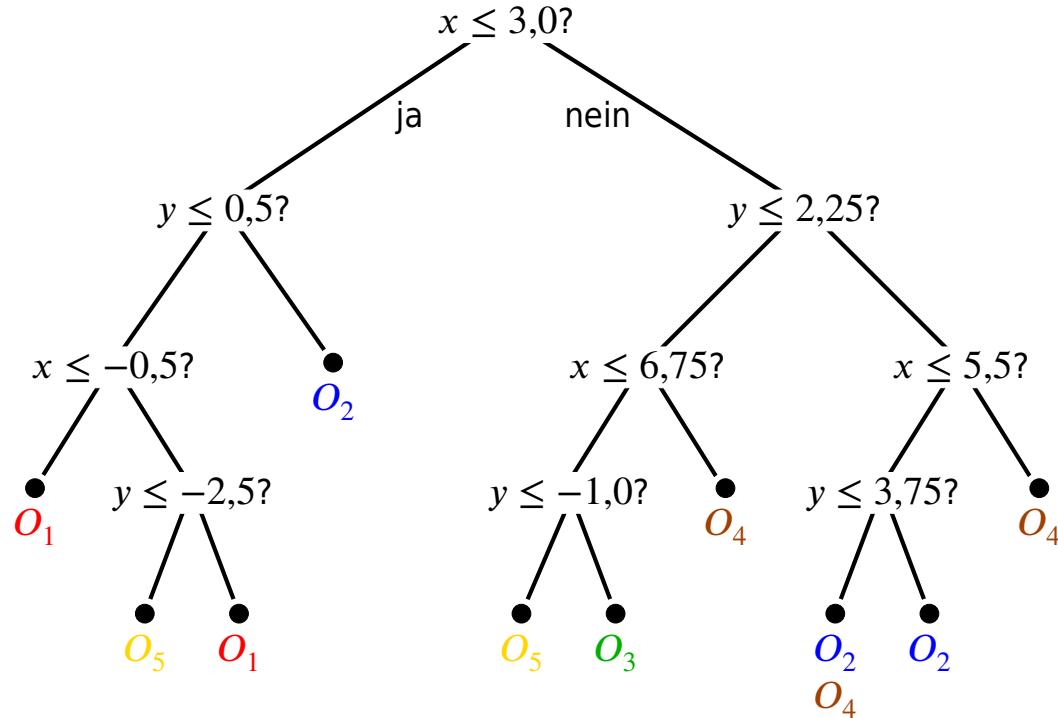
- Bestimme eine Bounding Box $Q_{...}$ der Szene und setze $L_{...} := \{\text{alle Objekte (einschl. Lichtquellen)}\}$.
- Solange es noch einen Quader Q_{xyz} gibt, der „groß genug“ ist und „zu viele“ Objekte enthält, zerlege Q_{xyz} in der „nächsten“ Richtung in **zwei** Quader $Q_{x'y'z'}$, bestimme für jeden dieser Teilquader die Menge $L_{x'y'z'}$.

Beispiel (2D):

durchlaufene Zellen Q_{xy}	L_{xy}	Schnitttests mit	Schnittpunkte
$[-0,5; 3,0] \times [-2,5; 0,5]$	O_1	O_1	
$[-2,0; 3,0] \times [0,5; 5,0]$	O_2	O_2	S_2^*
$[3,0; 6,75] \times [-1,0; 2,25]$	O_3	O_3	
$[3,0; 5,5] \times [2,25; 3,75]$	O_2, O_4	O_4	S_4^*

*: bisher nächstgelegener Schnittpunkt

zugehöriger **BSP-Baum (Binary Space Partitioning)**:



eigentliche Strahlverfolgung:

- Lokalisiere den Ausgangspunkt des Strahls im BSP-Baum.
 - Bestimme, durch welche Fläche der Strahl die aktuelle Zelle verlässt, berechne einen Punkt \tilde{P} „knapp jenseits“ dieser Fläche und lokalisiere \tilde{P} im BSP-Baum.
- ⇒ nächste vom Strahl durchlaufene Zelle

Bemerkungen zu allen drei Unterteilungsstrategien

- Dass ein Objekt mehrfach gegenüber demselben Strahl getestet wird (da es zu mehreren L_i gehört), lässt sich beispielsweise so verhindern:
 - Ordne jedem untersuchten Strahl eine eindeutige (z. B. fortlaufende) Nummer zu.
 - Trage nach dem Schnitttest „Objekt O_k - Strahl j “ die Nummer j bei Objekt O_k ein.
 - Prüfe vor jedem Test „ $O_k - j$ “, ob O_k bereits mit „ j “ markiert ist.
- Die Schnitttests dürfen nicht abgebrochen werden, sobald der erste Schnittpunkt gefunden ist. Man muss bis zu der Zelle laufen, die den (unter den bereits bestimmten) nächstgelegenen Schnittpunkt enthält, und deren Objekte noch prüfen.
- In der Vorverarbeitung kann man O_k auch in alle Zellen aufnehmen, die von der **Bounding Box** von O_k getroffen werden (nicht unbedingt vom Objekt).
 - ⇒ Vorverarbeitung i. Allg. wesentlich einfacher,
 - dafür später mehr überflüssige Schnitttests

Vergleich der Unterteilungsstrategien

Zellraster:

- ⊕ Die Listen L_{xyz} können sehr effizient bestimmt werden.
- ⊕ Die vom Strahl durchlaufenen Zellen können sehr einfach (und schnell) generiert und zugegriffen werden.
- ⊖ benötigt meist wesentlich mehr Zellen als die adaptiven Strategien
 - Speicherplatz
 - Bei der Strahlverfolgung müssen viele Zellen durchlaufen werden.
- ⊖ Die L_{xyz} können noch sehr umfangreich sein.
⇒ geeignet für Szenen mit relativ kleinen, etwa gleichmäßig verteilten Objekten

Octree:

- ⊕ Vorverarbeitung und Strahlverfolgung relativ einfach
- ⊕ moderate Anzahl von Zellen und einigermaßen ausgeglichene Verteilung der Objekte auf die Zellen
- ⊖ Der Octree kann sehr unbalanciert werden.
 - ⇒ Zellengenerierung bei Strahlverfolgung wird langsamer
- ⇒ universell verwendbar, aber nicht immer optimal

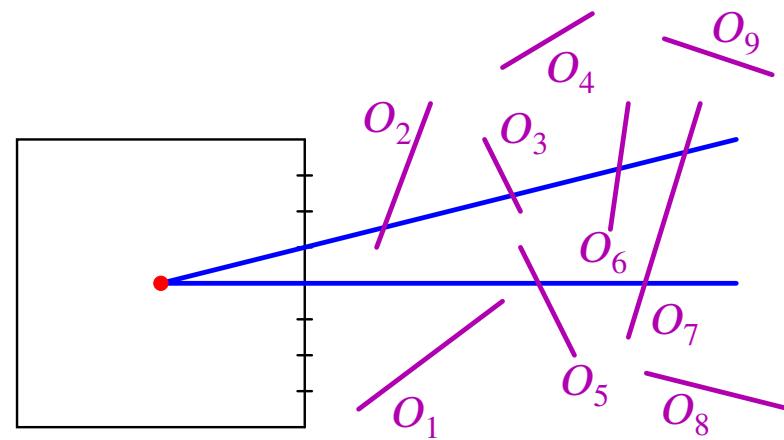
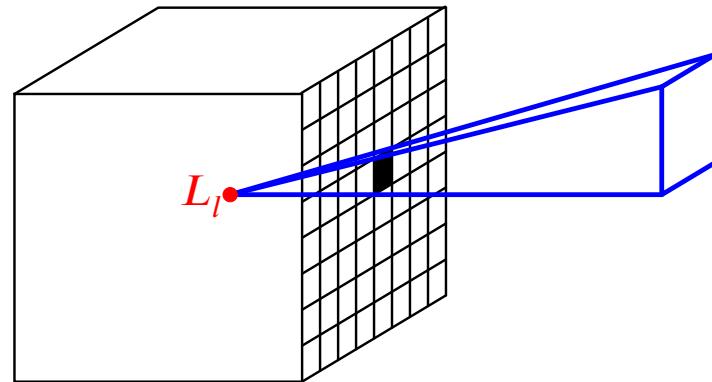
Median-Unterteilung:

- ⊕ liefert oft gut balancierte Bäume mit wenigen Zellen
- ⊖ Vorverarbeitung und Strahlverfolgung werden komplexer
(Wo muss der Quader unterteilt werden?)
- ⇒ geeignet für Szenen mit „Ballungen“ kleiner Objekte

10.1.4 Beschleunigung der Strahlverfolgung (II)

Beobachtung: Die meisten der zu verfolgenden Strahlen gehen von wenigen festen Punkten (den m **Lichtquellen**) aus.

- ⇒ Versuche, die Tests „liegt auf dem Strahl von L_l nach P' ein anderes Objekt?“ möglichst effizient zu machen.
- ⇒ Sammle in einer **Vorverarbeitung** Information über die Lage der Objekte bzgl. der Lichtquellen.

Ansatz: Lichtpuffer

Vorverarbeitung

für jede Lichtquelle L_l

lege einen Würfel um L_l und unterteile dessen Seitenflächen in Zellen $Z_{l,i}$

bestimme für jede Zelle $Z_{l,i}$, welche Objekte (teilweise) in das durch L_l und $Z_{l,i}$ definierte Volumen fallen, und speichere diese in einer Liste $L_{l,i}$
(soweit möglich, bereits nach der Entfernung von L_l geordnet)

/* praktische Realisierung: Führe für jedes Objekt O_k Projektion auf den Würfel und Scan Conversion bzgl. der „Zellen-Pixmaps“ durch */

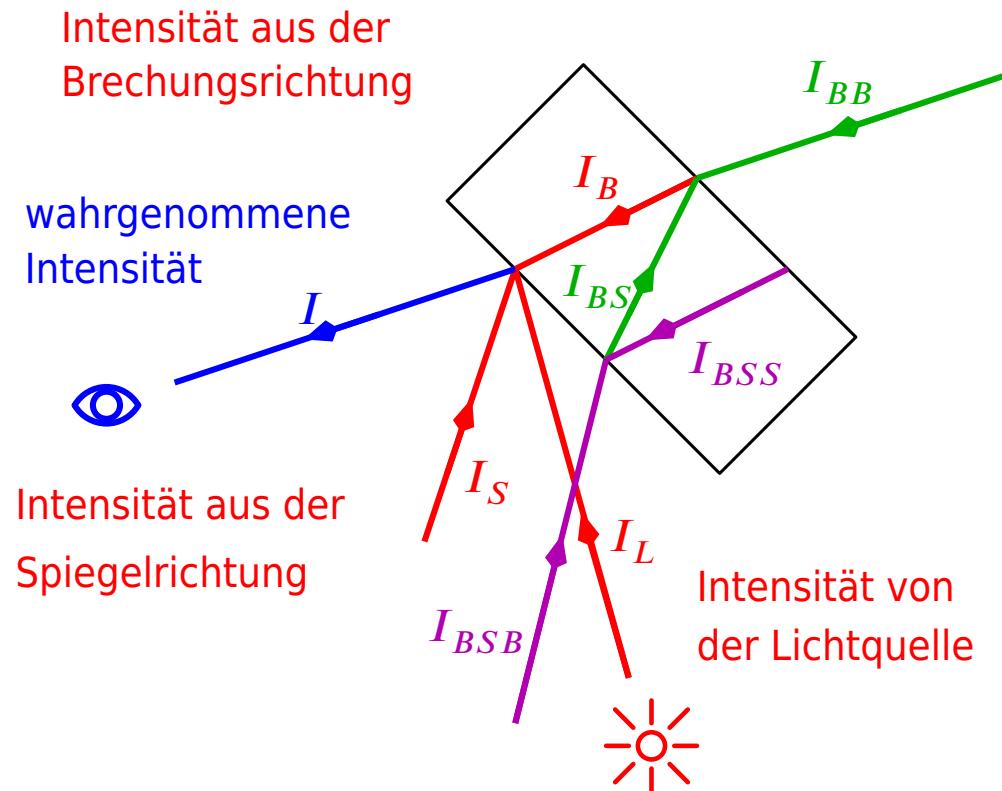
Eigentliche Strahlverfolgung

projiziere P' auf den Würfel und bestimme die Zelle $Z_{l,i}$, in welche der Punkt fällt

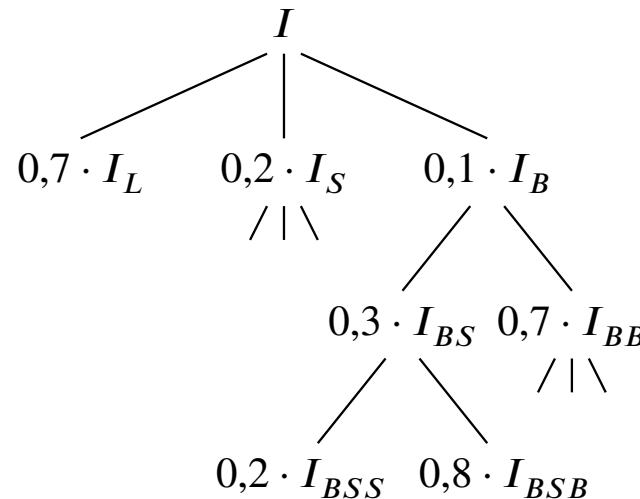
für alle Objekte O_k in $L_{l,i}$

prüfe, ob der Strahl das Objekt zwischen L_l und P' schneidet

/* Wenn dies der Fall und O_k undurchsichtig ist, dann können die Schnitttests abgebrochen werden */

Beobachtung:

Die wahrgenommene Intensität kann sich beispielsweise zusammensetzen aus:



⇒ I_{BSS} geht nur mit dem Faktor $0,1 \cdot 0,3 \cdot 0,2 = 0,006$ in I ein.

⇒ I_{BSS} kann vernachlässigt werden, **wenn** die Gesamtintensität I „nicht zu klein“ ist.

Ansatz: adaptive Tiefenkontrolle

- Führe bei jedem rekursiven Aufruf der Strahlverfolgung zwei zusätzliche Parameter mit:
 - den Faktor f , mit dem die Intensität des gerade verfolgten Strahls in die Gesamtintensität I eingeht
 - die bisher akkumulierte Gesamtintensität I
- Wenn $f \ll I$, dann muss der Strahl nicht verfolgt werden.

Bemerkungen 10.9:

1. Um I möglichst schnell wachsen zu lassen, sollte man die „Beiträge“ in geeigneter Reihenfolge betrachten:
 - zuerst ambienter Anteil
(benötigt keine Strahlverfolgung)
 - dann Beiträge direkt von den Lichtquellen
(Strahlverfolgung besonders effizient und rekursionsfrei)
 - dann gespiegelter oder gebrochener Anteil (je nachdem, welcher der Koeffizienten R_S oder R_B größer ist)
 2. Die „effektiv“ zu verfolgende Tiefe für ein einigermaßen realistisches Bild liegt bei etwa $t_{\text{eff}} \approx 1,7 - 2,0$ (weitgehend unabhängig von t_{\max}).
Enthält die Szene sehr viele stark spiegelnde und/oder brechende Objekte (z. B. Glas), so kann t_{eff} auch deutlich größer sein.
-

Beobachtung: Wenn die „effektive“ Tiefe klein ist, geht ein großer Teil aller zu verfolgenden Strahlen vom Augenpunkt aus.

⇒ Versuche, die zuerst getroffenen Objekte möglichst effizient zu bestimmen.

Idee: Strahlverfolgung vom Auge aus entspricht **Visible Surface Ray Tracing**

verwende statt dessen ***z*-Puffer**-Ansatz zur Bestimmung der sichtbaren (= zuerst erreichten) Objekte

Vorverarbeitung

bestimme mit einem modifizierten ***z*-Puffer**-Algorithmus die Entfernung $z_{i,j}$ des bei jedem Pixel (i,j) sichtbaren Objekts $O_{k_{i,j}}$

// Die $k_{i,j}$ werden im sog. Objektpuffer gespeichert.

Eigentliche Bilderzeugung

für jedes Pixel (i,j)

wenn $k_{i,j}$ = „Hintergrund“

färbe das Pixel mit der Hintergrundfarbe

sonst

bestimme den Schnittpunkt P' des „Sehstrahls“ mit $O_{k_{i,j}}$

färbe Pixel (i,j) gemäß der von P' einfallenden Intensität

// ≈ Raytracing ohne die erste Strahlverfolgung

10.1.5 Selektives Raytracing

Motivation: Bei „nicht allzu künstlichen“ Szenen werden die durch Raytracing simulierten Effekte (Spiegelungen, usw.) nur an wenigen Stellen des Bildes sichtbar sein.

⇒ Versuche, den Aufwand für Raytracing an den „harmlosen“ Stellen des Bildes zu sparen.

Ansatz:

- Erzeuge zunächst mit einem „Standard-Verfahren“ (z. B. z-Puffer-Algorithmus mit Phong-Shading) ein „**rohes Bild**“.

Führe dabei einen **Objektpuffer** mit.

- „**Nachbesserung**“:

für alle Pixel (i, j)

wenn das beim Pixel (i, j) sichtbare Objekt $O_{k_{i,j}}$ spiegelt, durchsichtig ist o. ä.

berechne die Färbung des Pixels mit Raytracing

10.1.6 Bemerkungen

- Trotz aller Optimierungen sind immer sehr viele Schnitttests notwendig – typisch: $\mathcal{O}(10^8)$
 - ⇒ durch geeignete **Vorverarbeitung der Objekte** diese Schnitttests (vor allem die erfolglosen!) möglichst schnell machen:
 - * Bounding Box, ...
- Die lokale Intensitätsberechnung kann leicht erweitert werden:
 - Textur
 - bessere Simulation der diffusen/winkelabhängigen Reflexion (Mikrofacetten-Modell, ...)
 - ...

10.2 Radiosity-Verfahren

(nach Goral/Torrance/Greenberg/Battaile 1984)

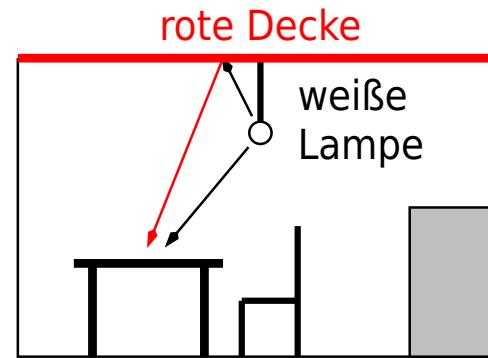
Ziel: möglichst gute Nachbildung der Effekte (mehrmaliger) diffuser Reflexion

Cindy M. Goral

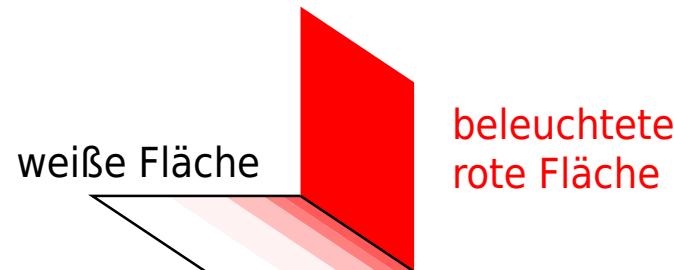
Kenneth E. Torrance
† 2010

Donald Peter Greenberg
* 1934

Bennett Battaile

Beispiele 10.10:**1. „Farbverschiebung“**

⇒ Alle Objekte im Raum erhalten einen „Rotstich“.

2. „Colour Bleeding“

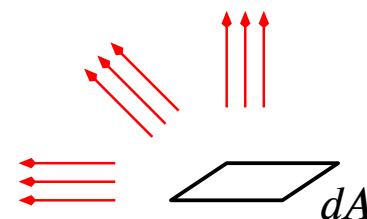
⇒ rote Farbe „läuft“ auch auf die weiße Fläche über

Ansatz: (Diffuse) Reflexion ist ein **Energieaustausch** zwischen den Objekten.

⇒ Übertrage ein Modell für den Wärmeaustausch auf den Austausch von Licht.

vereinfachende Annahmen:

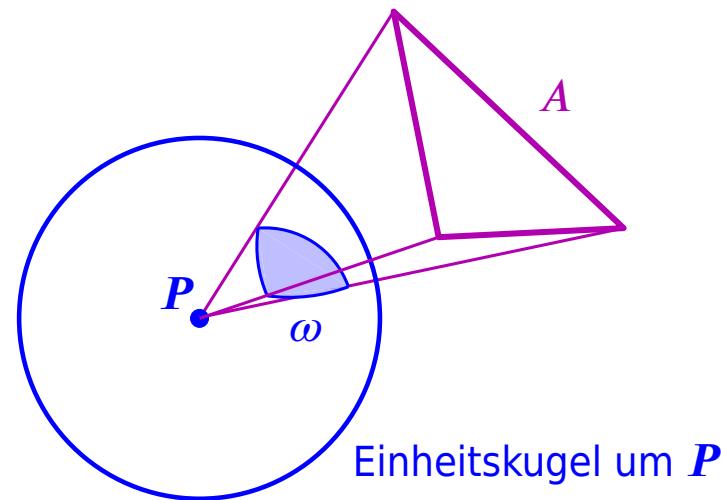
- Die Szene bildet ein **geschlossenes System im Gleichgewicht**, d. h.:
 - Die gesamte von einem Objekt abgestrahlte (emittierte oder reflektierte) Energie wird von den Objekten der Szene absorbiert oder weiter reflektiert (kein Energietransport „nach außen“).
 - Der Energieaustausch variiert nicht mit der Zeit.
- Die Objekte sind **vollkommen diffuse** Strahler bzw. Reflektoren, d. h. sie erscheinen aus jeder Richtung „gleich hell“.



10.2.1 Raumwinkel

Raumwinkel (Solid Angle): der von einem Punkt P aus durch eine Fläche A überdeckte Winkelbereich

Einheit: sr (Steradian): 1 sr ist der zu einer Fläche 1 auf der Einheitskugel gehörende Raumwinkel.

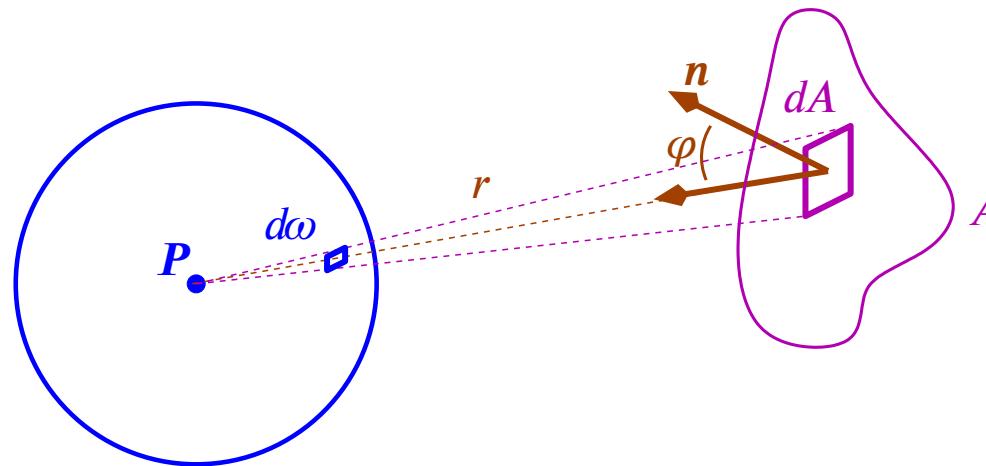


Beispiel 10.11: Die gesamte Einheitskugel besitzt die Oberfläche 4π .

⇒ Sie verdeckt (vom Mittelpunkt aus) den Raumwinkel $\omega = 4\pi$ [sr].

Bemerkung 10.12: Von \mathbf{P} aus ist statt dA nur die „verkürzte“ Fläche $dA \cdot \cos \varphi$ sichtbar.

(φ : Winkel zwischen Normale \mathbf{n} zu dA und Projektionsrichtung $\overrightarrow{dA\mathbf{P}}$)



$$\Rightarrow d\omega = \frac{\cos \varphi}{r^2} dA$$

Damit verdeckt A von \mathbf{P} aus den Raumwinkel

$$\omega = \int_A \frac{\cos \varphi}{r^2} dA \quad (\varphi \text{ und } r \text{ hängen von } dA \text{ ab!})$$

(falls A sich nicht teilweise selbst verdeckt).

10.2.2 Radiosity

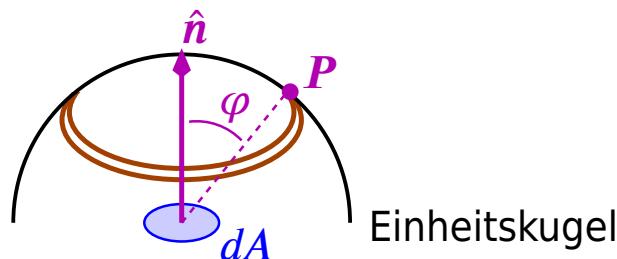
Radiosity: die von einem Objekt pro Zeit- und **sichtbarer** Flächeneinheit abgestrahlte Energie $\left[\frac{W}{m^2} \right]$

Radiance (Strahldichte): die von einem Objekt pro Zeit- und sichtbarer Flächeneinheit in eine bestimmte Richtung abgestrahlte Energie pro Steradian $\left[\frac{W}{m^2 \cdot sr} \right]$

Bemerkungen 10.13:

1. Da das Objekt völlig diffus abstrahlt, ist die Radiosity unabhängig von der Richtung.
 2. Wegen der Gleichgewichtsannahme kann man Leistung [W] und Energie [J] gleichwertig verwenden.
-

Beispiel 10.14: Wie viel Energie strahlt eine Fläche dA der Radiosity B insgesamt (pro Zeiteinheit) ab?



Von einem Punkt \mathbf{P} auf der Einheitskugel aus ist statt dA nur die „verkürzte“ Fläche $dA \cdot \cos \varphi$ sichtbar.

Die gesamte Strahlung von dA in Richtung \mathbf{P} ist also

$$B \cdot dA \cdot \cos \varphi .$$

Die Punkte mit gleichem φ überdecken auf der Einheitskugel die Fläche

$$\frac{2\pi}{\text{Radius des }} \underbrace{\sin \varphi}_{\text{„Streifens“}} \cdot \underbrace{d\varphi}_{\text{Breite des }} = \underbrace{d\omega}_{\substack{\text{überdeckter} \\ \text{Raumwinkel}}} .$$

Sie erhalten also die Leistung

$$B \cdot dA \cdot \cos \varphi \cdot 2\pi \sin \varphi \cdot d\varphi .$$

Also gibt dA insgesamt die Leistung

$$\begin{aligned} E_{dA} &= \int_0^{\frac{\pi}{2}} B \cdot dA \cdot \cos \varphi \cdot 2\pi \sin \varphi \cdot d\varphi \\ &= 2\pi BdA \int_0^{\frac{\pi}{2}} \cos \varphi \sin \varphi d\varphi \\ &= 2\pi BdA \cdot \frac{1}{2} \sin^2 \varphi \Big|_0^{\frac{\pi}{2}} \\ &= \pi BdA \end{aligned} \tag{10.3}$$

ab (auf eine Halbkugel über dA verteilt).

10.2.3 Von den Objekten zum Gleichungssystem

Ansatz:

$$\underbrace{\text{abgestrahlte Energie}^*}_{\text{Radiosity} \cdot \text{Fläche}} = \underbrace{\text{emittierte Energie}^*}_{\text{Emission pro Flächeneinheit} \cdot \text{Fläche}} + \underbrace{\text{reflektierte Energie}^*}_{\text{Reflexionskoeffizient} \cdot \text{von allen Objekten einfallende Energie}^*}$$

* : pro Zeiteinheit [W]

$$B_i \cdot dA_i = E_i \cdot dA_i + \rho_i \cdot \int_{\Omega} B_j \cdot F_{dA_j, dA_i} \cdot dA_j \quad (10.4)$$

dA_i : betrachtete Fläche

B_i : Radiosity der Fläche dA_i $\left[\frac{W}{m^2} \right]$

E_i : Eigenemission der Fläche dA_i $\left[\frac{W}{m^2} \right]$

ρ_i : Reflexionskoeffizient der Fläche dA_i (welcher Anteil des einfallenden Lichts wird reflektiert?)

dA_j : irgendeine Fläche auf irgendeinem Objekt

B_j : Radiosity der Fläche dA_j

Ω : gesamte Fläche aller Objekte

F_{dA_j, dA_i} : **Formfaktor** (der Anteil der von dA_j insgesamt abgestrahlten Energie [W], der bei dA_i ankommt)

Bemerkung 10.15: In den F_{dA_j, dA_i} steckt die „Geometrie“ der Szene:

- Entfernung der Flächen dA_i und dA_j
 - ihre relative Orientierung
 - die Sichtbarkeit (Liegt ein Objekt dazwischen?)
-

Problem: In der kontinuierlichen Formulierung (10.4) kann der Lichtaustausch nicht (mit dem Computer) berechnet werden.

⇒ **Diskretisierung:** Zerlege die Objekte in insgesamt n Flächenstücke (**Patches**) A_i . Für jeden Patch werden

- Radiosity B_i ,
- Eigenemission E_i und
- Reflexionskoeffizient ρ_i

als konstant angesetzt.

Dies führt auf

$$B_i \cdot A_i = E_i \cdot A_i + \rho_i \cdot \sum_{j=1}^n B_j \cdot F_{ji} \cdot A_j \quad (10.5)$$

mit den **Formfaktoren** F_{ji} .

(Welcher Anteil der von A_j insgesamt abgestrahlten Energie erreicht A_i ?)

In Abschnitt 10.2.5 wird die Beziehung

$$F_{ji} \cdot A_j = F_{ij} \cdot A_i \quad (10.6)$$

gezeigt. Einsetzen in (10.5) und Division durch A_i liefert

$$B_i = E_i + \rho_i \cdot \sum_{j=1}^n B_j \cdot F_{ij}$$

bzw.

$$(1 - \rho_i F_{ii}) \cdot B_i - \sum_{j \neq i} \rho_i \cdot F_{ij} \cdot B_j = E_i$$

bzw.

$$\underbrace{\begin{pmatrix} 1 - \rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 - \rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 - \rho_n F_{nn} \end{pmatrix}}_{\text{Radiosity Matrix } M_R} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}. \quad (10.7)$$



Bemerkungen 10.16:

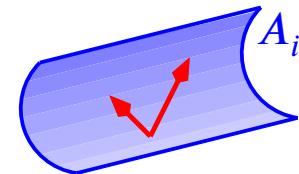
1. ρ_i und E_i sind i. Allg. abhängig von der Wellenlänge λ .

⇒ Für jede betrachtete Wellenlänge muss ein solches System gelöst werden.

(F_{ij} wird meist als λ -unabhängig betrachtet.)

2. $F_{ii} = 0$, falls der zugehörige Patch A_i eben oder konvex ist,

$F_{ii} > 0$, falls A_i konkav ist; dann fällt ein Teil der abgestrahlten Energie wieder auf A_i .



3. $\sum_{j=1}^n F_{ij} = 1$

(wegen der Annahme, dass alles von A_i abgestrahlte Licht wieder auf die Patches A_j fällt)

4. $F_{ij} = 0$, wenn A_j von A_i aus unsichtbar (verdeckt) ist.

⇒ Die Radiosity Matrix ist meist **dünn besetzt**

(typisch: ca. 10 % der F_{ij} sind $\neq 0$).

Die Matrix ist aber **ziemlich groß** (meist $n \sim 10^4 - 10^6$).

⇒ hoher Speicherbedarf ($\gg 10^6$ Koeffizienten $\neq 0$)

-
5. $E_i = 0$, außer wenn A_i selbst leuchtet.

10.2.4 Vom Gleichungssystem zum Bild

Die Lösung des Gleichungssystems gibt die Helligkeit der Patches an; sie ist **unabhängig davon, von wo aus die Szene betrachtet wird**.

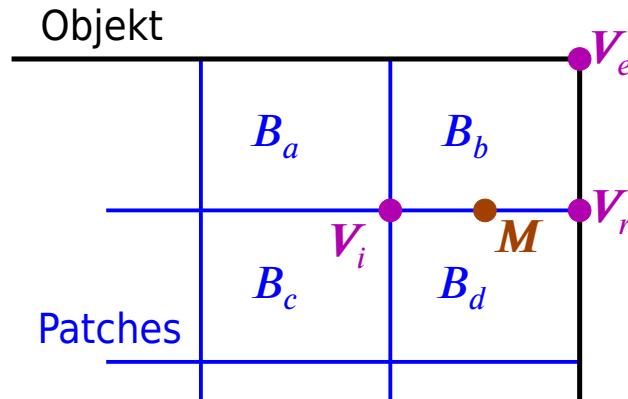
- ⇒ Aus der Lösung können mit geringem Aufwand mehrere Ansichten der Szene erstellt werden (z. B. „Walk Through“-Animation in Echtzeit).

Die eigentliche Bilderzeugung erfordert dann nur noch die Wiedergabe der Patches. Dies geschieht z. B. mit einer Kombination aus

- geeigneter $3D \rightarrow 2D$ -Transformation,
- z -Puffer-Algorithmus,
- Gouraud-Shading.

Problem: Beim Radiosity-Verfahren erhält jeder **Patch** konstante Helligkeit zugeordnet. Hieraus müssen Helligkeiten für die **Knoten** abgeleitet werden, so dass Gouraud-Shading an den Objekt-internen Patchgrenzen stetige Farbübergänge liefert.

Ansatz: Inter- bzw. Extrapolation



Fall 1: V_i ist „innere Ecke“.

$$B(V_i) := \frac{1}{4} \cdot \left(\underbrace{B_a + B_b + B_c + B_d}_{\text{Radiosities der mit } V_i \text{ inzidenten Patches}} \right)$$

Fall 2: V_r ist „Randecke“, aber nicht Ecke des Objekts.

Seien A_b und A_d die mit V_r inzidenten Patches, V_i der andere (im Innern des Objektes gelegene) Endpunkt und M der Mittelpunkt der Grenze $A_b|A_d$.

$$B(V_r) := 2B(M) - B(V_i)$$

mit

$$B(M) := \frac{1}{2} \cdot (B_b + B_d)$$

Fall 3: V_e ist Ecke des Objekts.

Sei A_b der mit V_e inzidente Patch und V_i die (auf A_b) „gegenüberliegende“ innere Ecke.

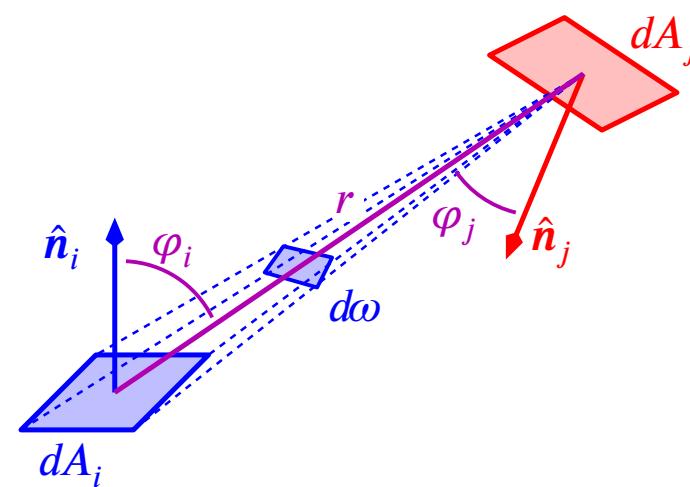
$$B(V_e) := 2B_b - B(V_i)$$

10.2.5 Die Formfaktoren

Von dA_i aus ist statt dA_j nur die verkürzte Fläche

$$dA'_j = dA_j \cdot \cos \varphi_j$$

sichtbar.



Von dA_j aus erscheint die (verkürzte) Fläche dA'_j unter dem Raumwinkel

$$d\omega = \frac{dA_i \cdot \cos \varphi_i}{r^2} .$$

Also erhält dA_i von dA_j die Energie

$$\begin{aligned} E_{dA_j, dA_i} &= B_j \cdot dA'_j \cdot \underbrace{d\omega \cdot H_{ji}}_{\begin{array}{l} = 1 \text{ falls } dA_j \text{ von } dA_i \text{ aus sichtbar ist} \\ = 0 \text{ sonst} \end{array}} \\ &= \frac{B_j \cdot \cos \varphi_i \cdot \cos \varphi_j}{r^2} \cdot H_{ji} \cdot dA_j \cdot dA_i , \end{aligned}$$

und damit erhält Patch A_i von Patch A_j die Energie

$$E_{A_j, A_i} = \int_{A_i} \int_{A_j} \frac{B_j \cdot \cos \varphi_i \cdot \cos \varphi_j}{r^2} \cdot H_{ji} \cdot dA_j \cdot dA_i .$$

Die gesamte Abstrahlung von A_j ist

$$\begin{aligned} E_{A_j} &= \int_{A_j} E_{dA_j} dA_j \\ &= \int_{A_j} \pi B_j dA_j \quad (\text{gemäß (10.3)}) \\ &= \pi B_j A_j . \end{aligned}$$

Damit folgt

$$F_{ji} = \frac{E_{A_j, A_i}}{E_{A_j}} = \frac{1}{A_j} \cdot \int_{A_i} \int_{A_j} \frac{\cos \varphi_i \cdot \cos \varphi_j}{\pi r^2} H_{ji} dA_j dA_i . \quad (10.8)$$

Bemerkungen 10.17:

1. Aus (10.8) folgt wegen $H_{ji} = H_{ij}$ sofort die Symmetriebeziehung (10.6).
 2. Für die Berechnung der F_{ij} mit dem Computer ist (10.8) nicht zu gebrauchen (Vierfach-Integral, usw.).
 3. (10.8) könnte mit dem Satz von Stokes in ein doppeltes Kontur-Integral umgewandelt werden.
-

10.2.6 Numerische Berechnung der Formfaktoren

Vereinfachung 1: φ_i, φ_j, r und H_{ji} seien unabhängig von dem gerade auf A_j betrachteten Punkt.

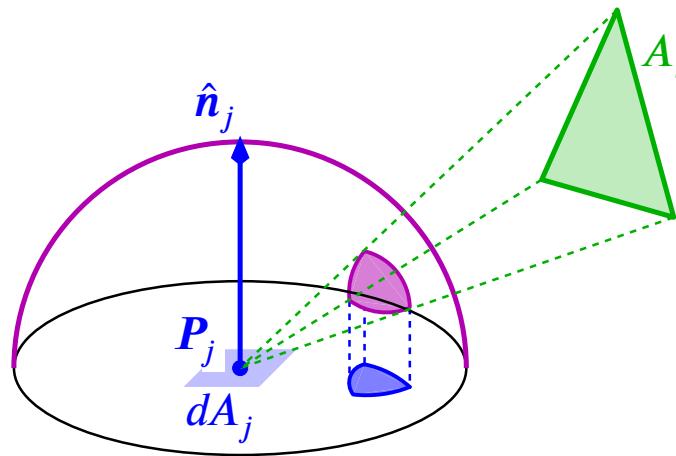
(liefert gute Näherung, falls A_j klein gegenüber r)

Einsetzen in (10.8) ergibt

$$\begin{aligned} F_{ji} &= \frac{1}{A_j} \cdot \int_{A_i} \frac{\cos \varphi_i \cdot \cos \varphi_j}{\pi r^2} H_{ji} dA_i \cdot \int_{A_j} dA_j \\ &= \int_{A_i} \frac{\cos \varphi_i \cdot \cos \varphi_j}{\pi r^2} H_{ji} dA_i. \end{aligned} \quad (10.9)$$

Als „Referenzpunkt“ P_j auf A_j wird dann beispielsweise der Mittelpunkt von A_j gewählt.

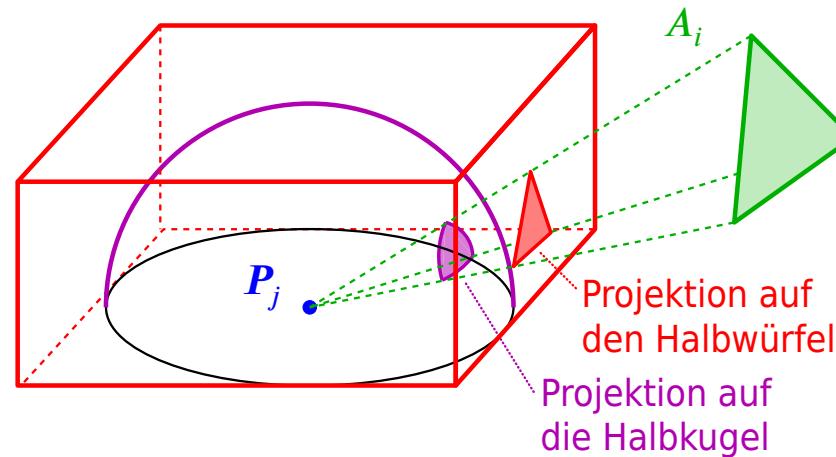
Bemerkung 10.18: Der Wert F_{ji} in (10.9) kann auch geometrisch gewonnen werden (ohne Beweis).



- Projizierte den sichtbaren Teil von A_i auf die Einheitshalbkugel (in Richtung \hat{n}_j) um P_j .
(Dies liefert $\int_{A_i} \frac{\cos \varphi_i}{r^2} H_{ji} dA_i$.)
- Projizierte diese Fläche auf den „Grundkreis“ der Halbkugel (liefert $\cos \varphi_j$).
- Dividiere diese Fläche durch die Kreisfläche (liefert $\frac{1}{\pi}$).

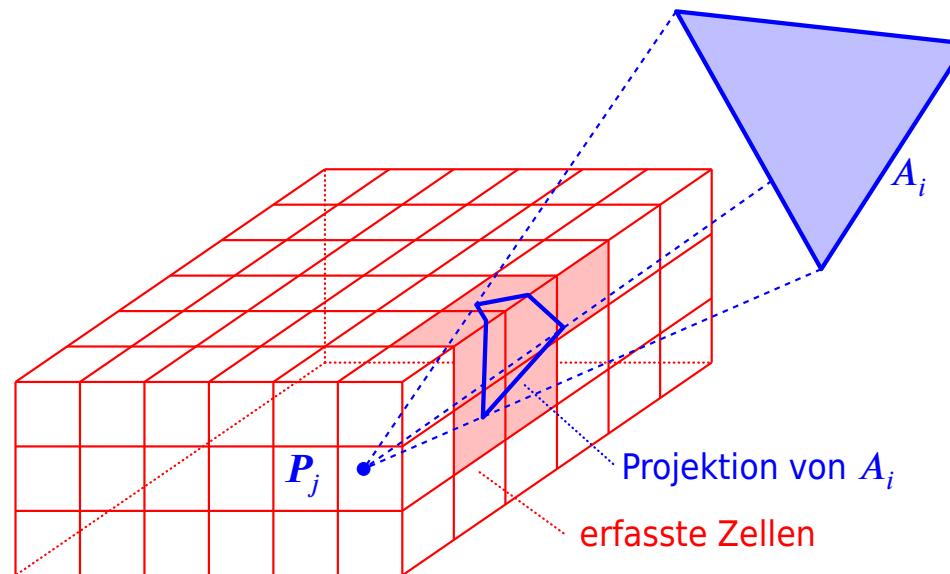
Der Formfaktor ist also aus der Projektion des Patches A_i auf die Einheitskugel (Größe und Lage der projizierten Fläche) ableitbar.

Beobachtung: Statt der Einheitshalbkugel ist auch eine beliebige konvexe „Halbumgebung“ von P_j verwendbar, z. B. ein **Halbwürfel**:



(Vom Halbwürfel kann „weiter“ auf die Halbkugel projiziert werden.)

Vereinfachung 2: Zerlege die Oberfläche des Halbwürfels in kleine Flächenstücke (**Zellen**) C_q und ersetze die Projektion durch die Vereinigung der davon erfassten Zellen (\triangleq **Diskretisierung**).



Beobachtung: Jede Zelle C_q liefert einen Beitrag ΔF_q zu F_{ji} :

$$F_{ji} = \sum_{q: C_q \text{ wird bei der Projektion von } A_i \text{ erfasst}} \Delta F_q$$

mit

$$\Delta F_q = \frac{\text{Projektion von } C_q \text{ auf Einheitskugel und Kreis}}{\text{Kreisfläche}}$$

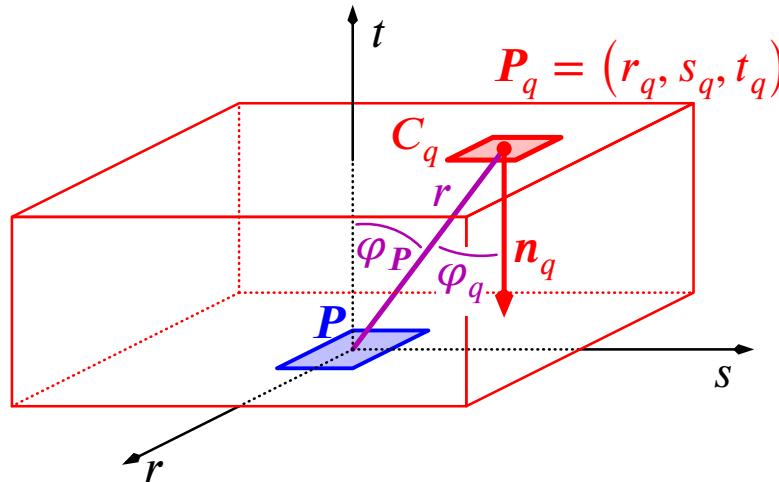
Bemerkung 10.19: Die **Delta-Formfaktoren** ΔF_q hängen nur von der Unterteilung des Würfels in Zellen ab, nicht von projiziertem Patch A_i oder von A_j (letzteres nur, wenn alle Halbwürfel um die P_j gleich unterteilt werden).

⇒ Die ΔF_q können **vorab** berechnet werden!

Oft ist es günstiger, die zu verschiedenen P_j gehörigen Halbwürfel – oder sogar die verschiedenen Flächen eines Halbwürfels – unterschiedlich fein zu unterteilen. Dann sind mehrere (z. B. 4) „Sätze“ der ΔF_q zu speichern.

typische Anzahl der Zellen pro Halbwürfelfläche: 100 – 100 000

Berechnung der Delta-Formfaktoren:



Gemäß (10.9) gilt

$$\Delta F_q = \int_{C_q} \frac{\cos \varphi_q \cdot \cos \varphi_P}{\pi r^2} dC_q .$$

($H \equiv 1$, da die ganze Zelle von P aus sichtbar ist.)

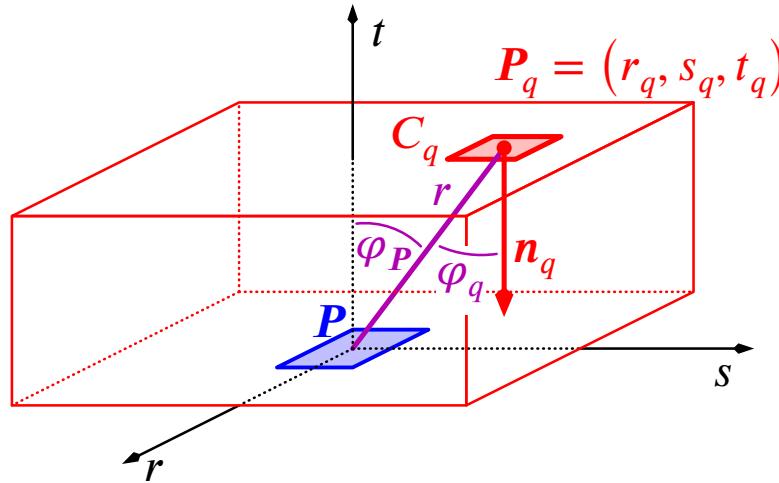
Vereinfachung 3: φ_q , φ_P und r seien unabhängig von dem gerade auf C_q betrachteten Punkt P_q .

Dann ist

$$\Delta F_q = \frac{\cos \varphi_q \cdot \cos \varphi_P}{\pi r^2} \cdot \underbrace{C_q}_{\text{Fläche der Zelle}} .$$

Als „Referenzpunkt“ wählt man etwa den Mittelpunkt $P_q = (r_q, s_q, t_q)$ von C_q .

Fall a: C_q liegt auf der „Deckfläche“ des Würfels.



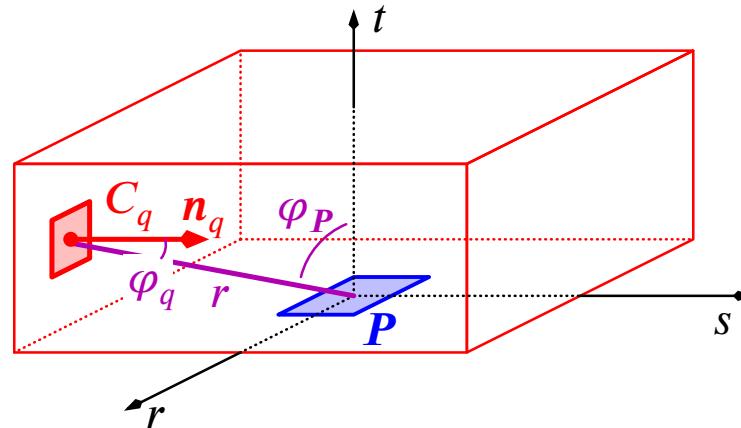
Dann ist

$$\begin{aligned} t_q &= 1, \\ r &= \sqrt{r_q^2 + s_q^2 + t_q^2} = \sqrt{r_q^2 + s_q^2 + 1}, \\ \cos \varphi_q &= \cos \varphi_P = \frac{1}{r} \end{aligned}$$

und damit

$$\Delta F_q = \frac{C_q}{\pi r^4} = \frac{C_q}{\pi (r_q^2 + s_q^2 + 1)^2}.$$

Fall b: C_q liegt auf einer Seitenfläche $s_q = \pm 1$.



Dann ist

$$\begin{aligned} r &= \sqrt{r_q^2 + 1 + t_q^2}, \\ \cos \varphi_q &= \frac{1}{r}, \\ \cos \varphi_P &= \frac{t_q}{r} \end{aligned}$$

und damit

$$\Delta F_q = \frac{t_q \cdot C_q}{\pi (r_q^2 + 1 + t_q^2)^2}.$$

Fall c: C_q liegt auf einer Seitenfläche $r_q = \pm 1$. Dann folgt analog

$$\Delta F_q = \frac{t_q \cdot C_q}{\pi (1 + s_q^2 + t_q^2)^2}.$$

Bemerkung 10.20: Aus Symmetriegründen müssen nur die Delta-Formfaktoren für ein Achtel der Deckfläche und eine halbe Seitenfläche wirklich berechnet werden.

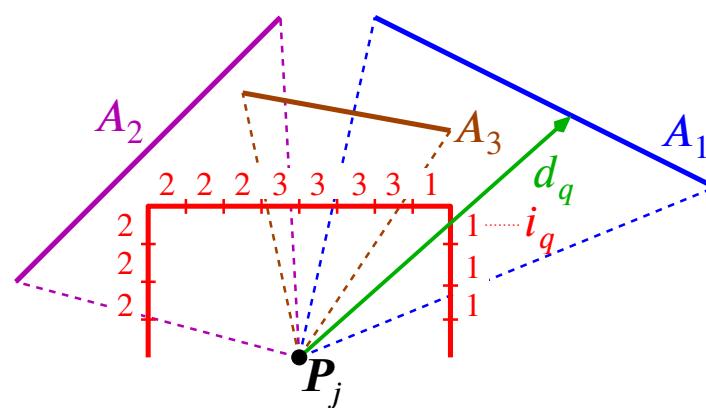
Problem: Wie bestimmt man möglichst effizient die von der Projektion (der nicht verdeckten Teile) von A_i erfassten Zellen?

Idee: Die Zellen auf einer Fläche des Halbwürfels können als **Pixmap** interpretiert werden.

- ⇒ Die erfassten Zellen ergeben sich durch perspektivische Projektion bzgl. P_j und anschließende **Scan Conversion**.

Mit einem **z-Puffer**-ähnlichen Ansatz kann dabei auch gleich die Verdeckungsanalyse durchgeführt werden.

- ⇒ **Objektpuffer:** Speichere für jede Zelle C_q die Nummer i_q und die Entfernung d_q des dort gerade sichtbaren Patches.



Algorithmus 10.21:

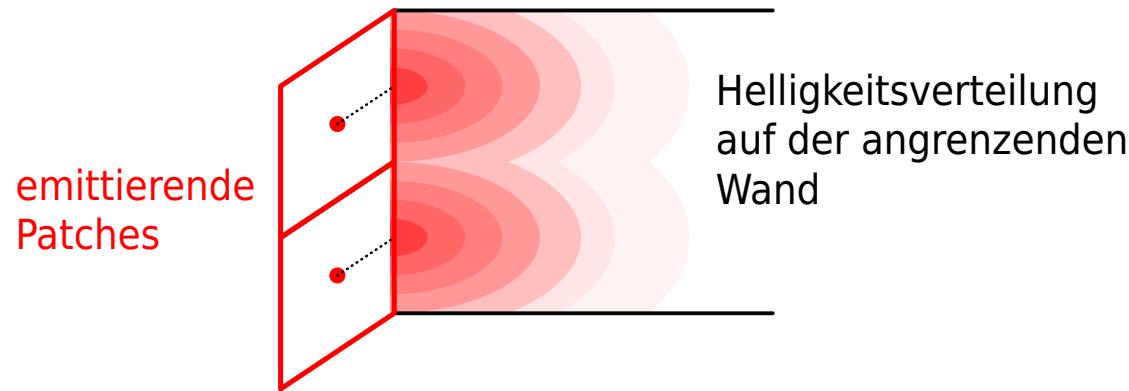
Algorithmus Bestimmung der Formfaktoren F_{ji} , $i = 1, \dots, n$

```
// Initialisiere Objektpuffer
für alle Zellen  $C_q$  des Halbwürfels
   $(i_q, d_q) := (0, +\infty)$  // „Hintergrund“
// Projiziere alle  $A_i$  auf den Halbwürfel um  $P_j$ 
für  $i = 1, \dots, n$ 
  für jede der fünf Flächen des Halbwürfels
    projiziere  $A_i$  auf die Fläche und führe Scan Conversion durch
    für jede hierbei gelieferte Zelle  $C_q$ 
      bestimme den Abstand  $d$  des zugehörigen Punktes auf  $A_i$  von  $P_j$ 
      wenn  $d < d_q$  // Patch  $A_i$  bei  $C_q$  sichtbar?
         $(i_q, d_q) := (i, d)$ 
```

```
// Berechne aus den Projektionen die Formfaktoren
für  $i = 1, \dots, n$ 
   $F_{ji} := \sum_{q: i_q=i} \Delta F_q$  // Summe über die Zellen, bei denen  $A_i$  sichtbar ist
```

Bemerkungen 10.22:

1. Vereinfachung 1 (ersetze A_j durch \mathbf{P}_j) führt dazu, dass sich ausgedehnte Lichtquellen (z. B. Fenster, Leuchtkörper) wie Punktlichtquellen verhalten.



⇒ Ausgedehnte Lichtquellen müssen i. Allg. in relativ kleine Patches zerlegt werden.

2. Alle drei Vereinfachungen können zu wesentlichen Fehlern in den F_{ji} führen, wenn
 - die Patches (relativ zu ihrer Größe) nahe beieinander liegen, z. B. aneinanderstoßen
(Regel: Abstand $\geq 5 \cdot$ Patchdurchmesser)
 - der Halbwürfel in zu wenige Zellen unterteilt wird.

3. trotz der Vereinfachungen **sehr hoher Aufwand**:

$\underbrace{n\text{-mal}}$ komplette „Bilderzeugung“
für $j = 1, \dots, n$

$\underbrace{\text{Projektion}}_{\text{Scan Conversion}}_{\text{Verdeckungsanalyse}}$

allerdings für „Pixmaps“ verhältnismäßig geringer Auflösung

10.2.7 Schrittweiser Lösungsaufbau

Motivation:

- Die Berechnung der B_i ist sehr aufwändig.
 - ⇒ Ist es möglich, schon relativ früh eine „grobe“ Näherung zu erhalten und diese im Laufe der Zeit zu verbessern?
- Die F_{ji} erfordern sehr viel Speicher.
 - ⇒ Kann das Verfahren so umstrukturiert werden, dass immer nur ein kleiner Teil der F_{ji} gleichzeitig benötigt wird?

Ansatz: geeigneter iterativer Algorithmus zur Lösung des LGS

Gauß-Seidel-Verfahren zur Lösung von (10.7):

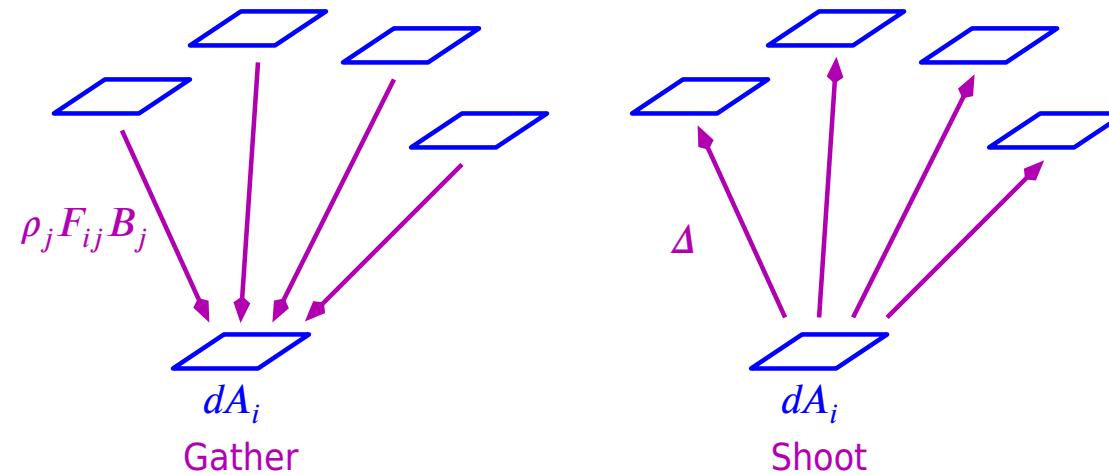
starte mit $B_i = E_i$, $i = 1, \dots, n$
 wiederhole bis gewünschte Genauigkeit erreicht
 für $i = 1, \dots, n$

$$B_i := E_i + \rho_i \cdot \sum_{j=1}^n F_{ij} \cdot B_j$$

Die neue Radiosity B_i von A_i ergibt sich also durch „Einsammeln“ (**Gather**) der Beiträge $\rho_i \cdot F_{ij} \cdot B_j$ der anderen Patches.

Idee: Betrachte die umgekehrte Abhängigkeit: (nach Cohen/Chen/Wallace/Greenberg 1988)

Wie ändern sich die übrigen Radiosities, wenn sich B_i um ΔB_i ändert (**Shoot**)?



Michael F. Cohen

Shenchang Eric Chen

John R. Wallace

Antwort auf die Frage:

$$\begin{aligned}\Delta B_j &= \rho_j F_{ji} \cdot \Delta B_i \\ &= \rho_j F_{ij} \cdot \frac{A_i}{A_j} \cdot \Delta B_i \quad (\text{gemäß (10.6)})\end{aligned}$$

Algorithmus 10.23:

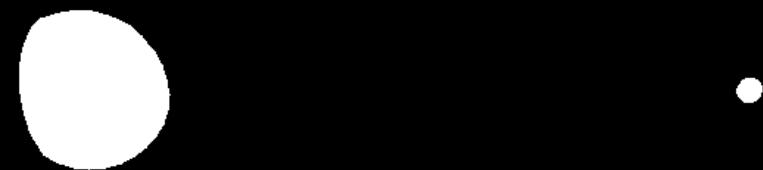
Algorithmus Schrittweiser Lösungsaufbau

```

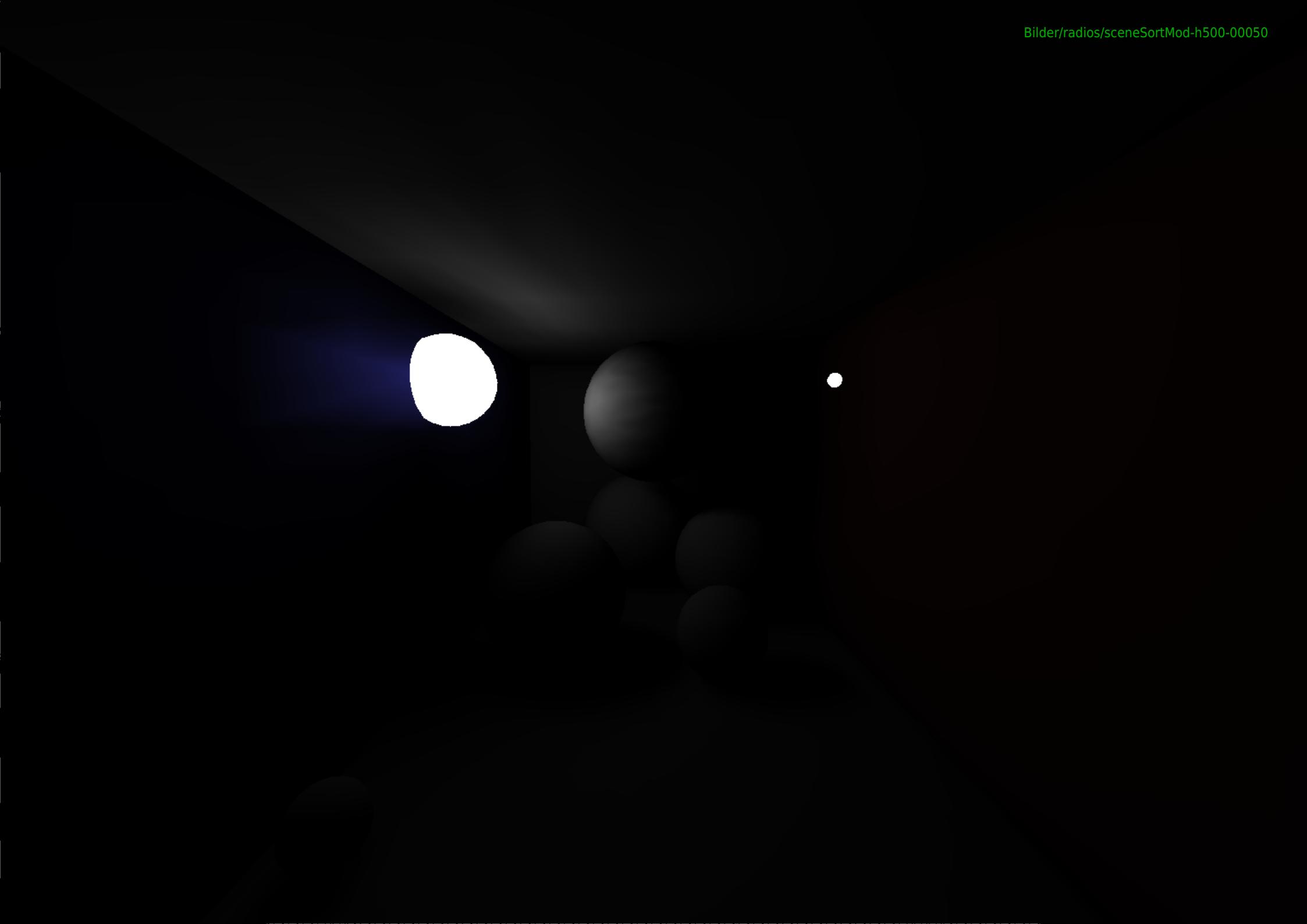
für  $i = 1, \dots, n$ 
   $B_i := E_i$           // berücksichtige nur Eigenemission
   $\tilde{\Delta}B_i := E_i$     // um wie viel hat sich  $B_i$  geändert, seit  $A_i$  letztmalig ausgewählt wurde?
  ggf. Ansicht erstellen
  wiederhole bis gewünschte Genauigkeit erreicht
    wähle einen Patch  $A_i$  mit maximalem  $\tilde{\Delta}B_i \cdot A_i$  // Maß für Änderung der ges. abgestrahlten Energie
    bestimme  $F_{ij}$ ,  $j = 1, \dots, n$ 
    für  $j = 1, \dots, n$ 
       $\Delta B_j := \rho_j F_{ij} \frac{A_i}{A_j} \cdot \Delta B_i$ 
       $B_j := B_j + \Delta B_j$ 
       $\tilde{\Delta}B_j := \tilde{\Delta}B_j + \Delta B_j$ 
    ggf. Ansicht erstellen
     $\tilde{\Delta}B_i := 0$ 
```

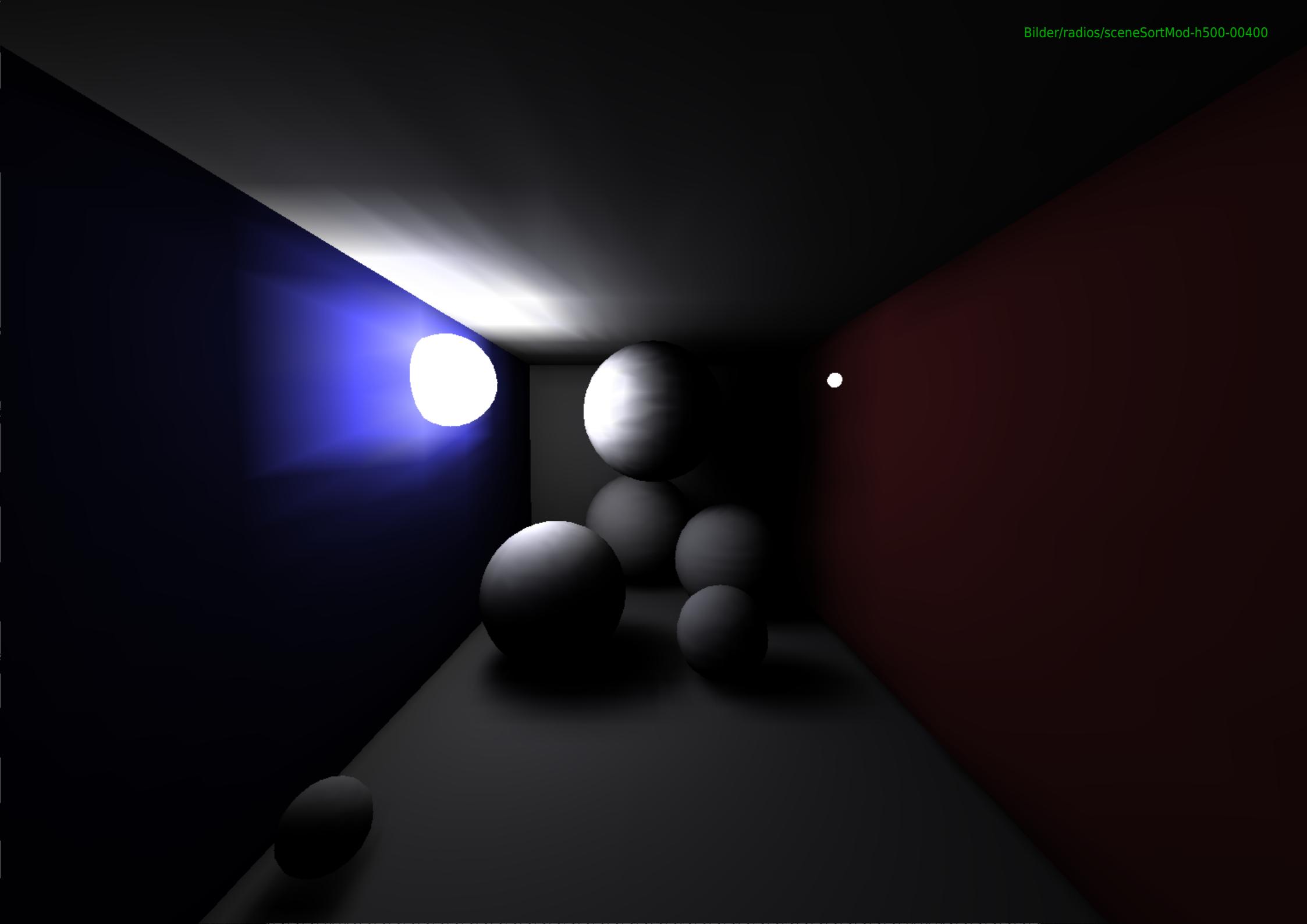
Bemerkungen 10.24:

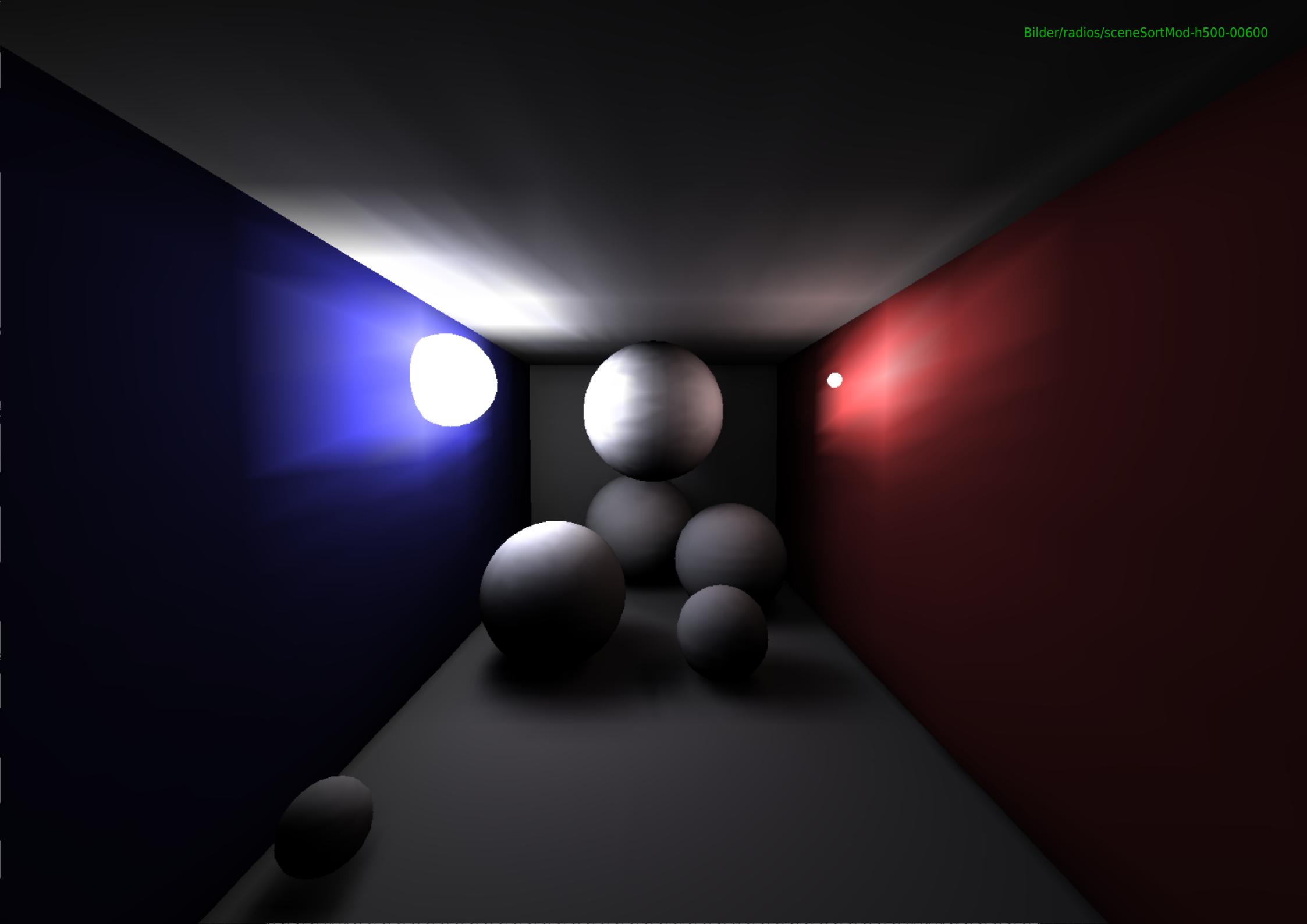
1. Zu Beginn sind nur die selbst emittierenden Objekte sichtbar, dann wird der Beitrag des stärksten Strahlers an alle Objekte berücksichtigt, dann der zweitstärkste usw.
⇒ Die Szene ist anfangs überwiegend dunkel und hellt sich langsam auf.
 2. Wenn die F_{ij} „schnell genug“ erzeugt werden können (z. B. mit einer z-Puffer-Hardware), dann muss nur noch eine **Zeile** der Radiosity-Matrix gespeichert werden; die Formfaktoren werden allerdings mehrmals berechnet.
⇒ Möglichkeit, die Formfaktoren im Laufe der Zeit zu ändern
-

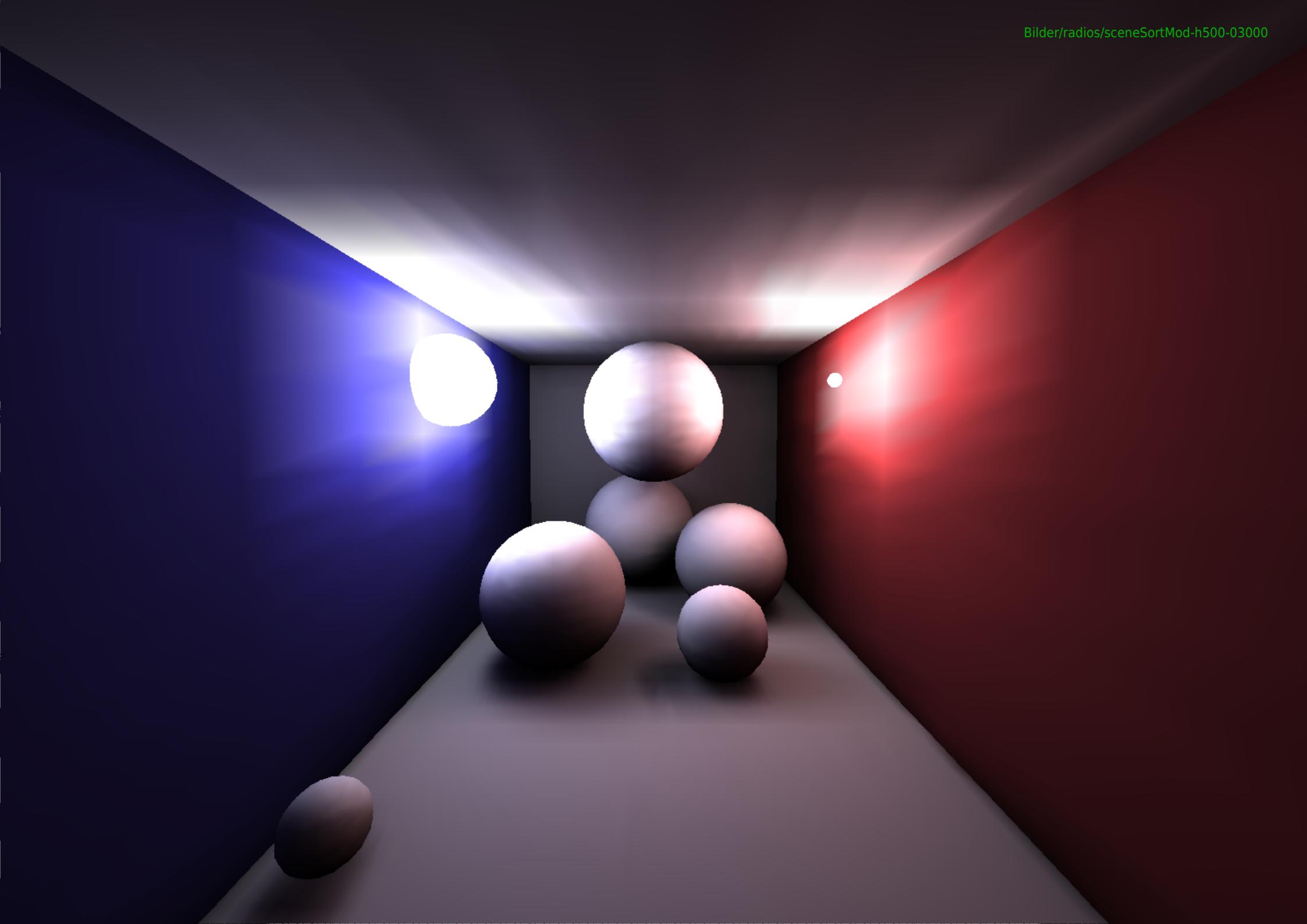


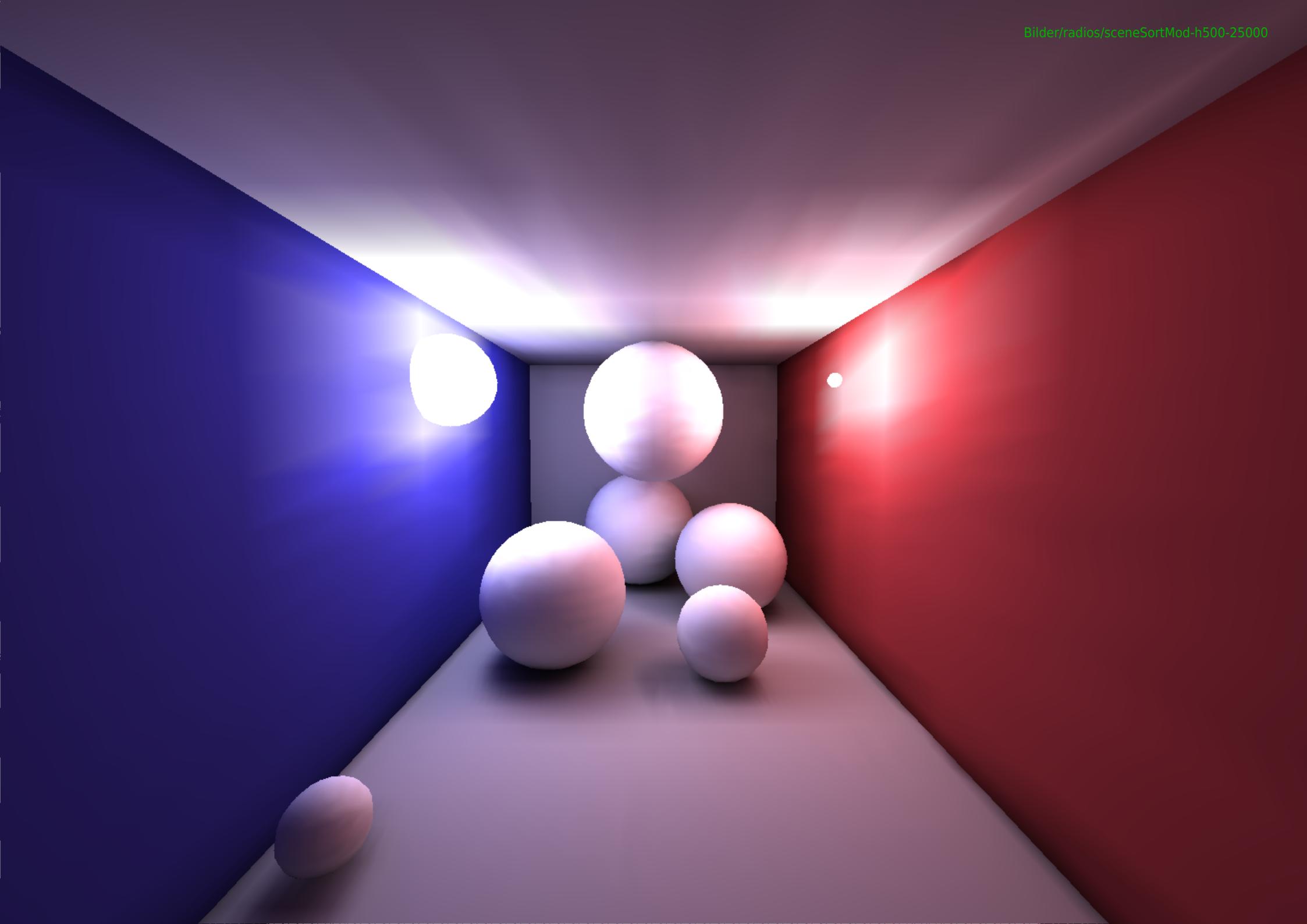


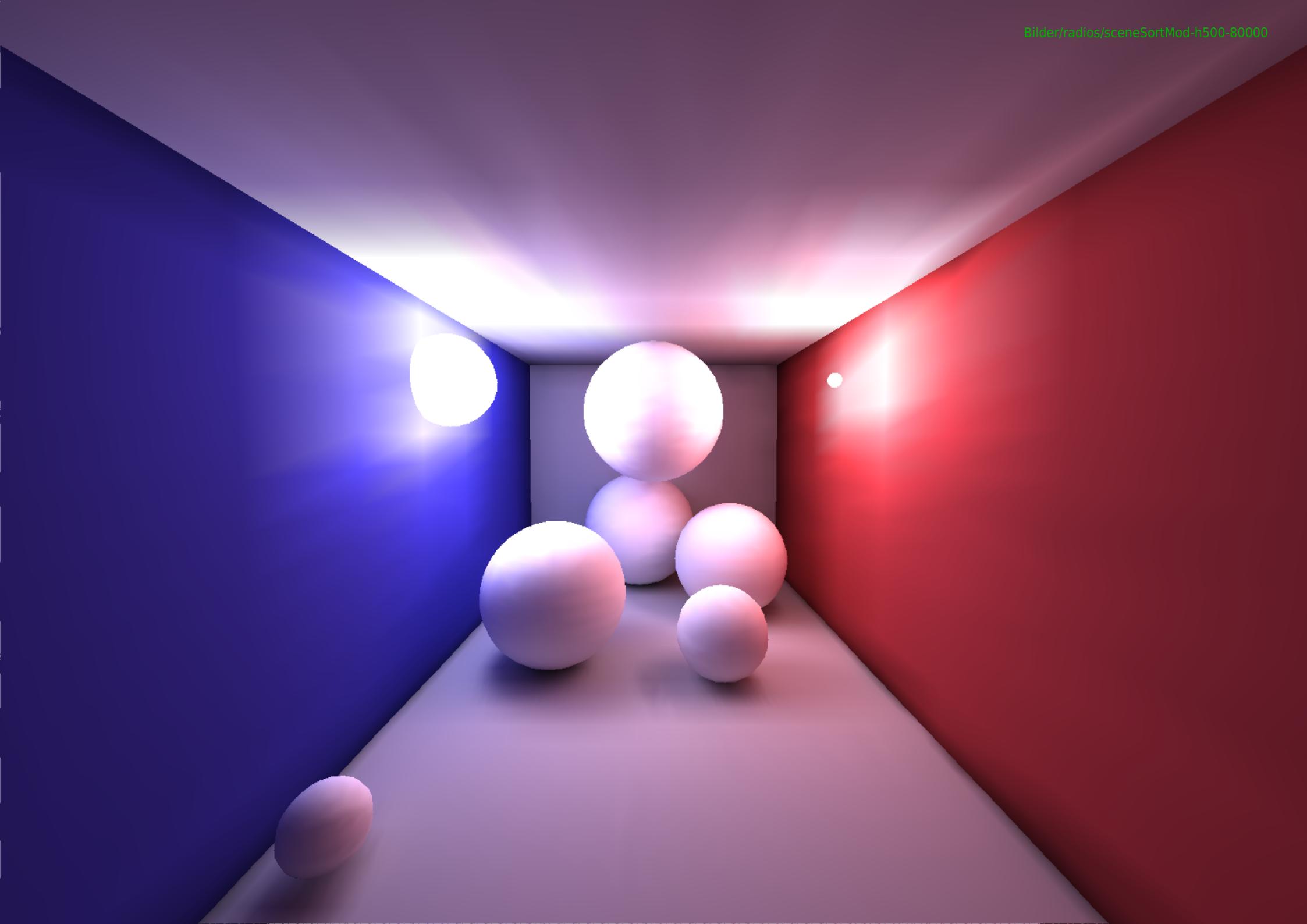










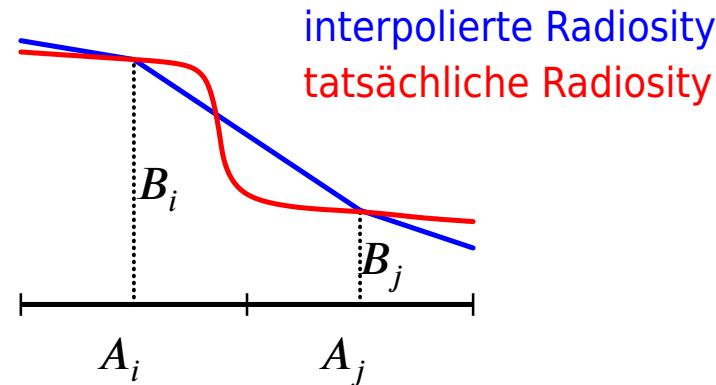


10.2.8 Unterteilung der Patches

Problem: Oft wird nach der Berechnung der Formfaktoren und dem Lösen des Gleichungssystems deutlich, dass die ursprüngliche Unterteilung in Patches zu grob war.

typisches Kriterium: große Radiosity-Unterschiede zwischen benachbarten Patches

- ⇒ Die Patches enthalten einen deutlichen Farbübergang (z.B. **Schattengrenze**).
Dieser wird bei der Wiedergabe der Patches „glattinterpoliert“.



- ⇒ Nach der Interpolation ist der Übergang nicht mehr lokalisierbar.

Ansatz I: Erhöhe die Anzahl der Patches auf N und fange wieder von vorne an:

$$\begin{array}{ll} N^2 & \text{Formfaktoren berechnen} \\ N \times N & \text{Gleichungssysteme lösen} \end{array}$$

- ⇒ für $N \gg n$ starke Zunahme des Aufwands

Ansatz II: Unterteile, wo notwendig, die Patches in **Elemente**:

$$A_i \rightsquigarrow A_{i,1}, \dots, A_{i,m_i}$$

- Der Einfluss der übrigen Patches A_j auf A_i wird für jedes Element $A_{i,q}$ von A_i bestimmt.
⇒ Die Beleuchtungsverhältnisse auf A_i können detaillierter modelliert werden.
 - Der Einfluss von A_i auf die übrigen A_j wird **summarisch** bestimmt, d. h. die Elemente von A_i werden hierzu zusammengefasst.
⇒ Reduktion des Rechenaufwands
- i) Berechne die $N \cdot n$ Formfaktoren F_{j,i_q} (Anteil des von A_j ausgehenden Lichts, das bei Element $A_{i,q}$ eintrifft) bzw. $F_{i_q,j}$.
 - ii) Verwende die Lösung $(B_j : j = 1, \dots, n)$ des ursprünglichen $n \times n$ -Gleichungssystems und berechne hieraus die Element-Radiosities gemäß

$$B_{i,q} = \underbrace{E_i}_{\text{evtl. } E_{i,q}} + \underbrace{\rho_i}_{\text{evtl. } \rho_{i,q}} \cdot \sum_{j=1}^n B_j F_{i_q,j}. \quad (10.10)$$

Ansatz III: ähnlich wie II, aber

- ii) Berechne verbesserte „Patch-zu-Patch“-Formfaktoren

$$\tilde{F}_{ij} = \frac{1}{A_i} \cdot \sum_{q=1}^{m_i} F_{i_q,j} \cdot A_{i,q}$$

(Flächen-gewichtetes Mittel der $F_{i_q,j}$).

- iii) Berechne die Patch-Radiosities B_j über ein $n \times n$ -Gleichungssystem (mit \tilde{F}_{ij} statt F_{ij}).
- iv) Berechne hieraus die Element-Radiosities gemäß (10.10).

Bemerkungen 10.25:

1. Die Elemente werden ggf. nochmals unterteilt:

**solange es noch benachbarte Patches/Elemente mit „zu großem“ Radiosity-Unterschied gibt
unterteile diese in kleinere Elemente
berechne Radiosities (zumindest) für die neuen Elemente**

2. Unterteilung in Elemente ändert nichts an der Tatsache, dass emittierende Patches wie Punktlichtquellen strahlen

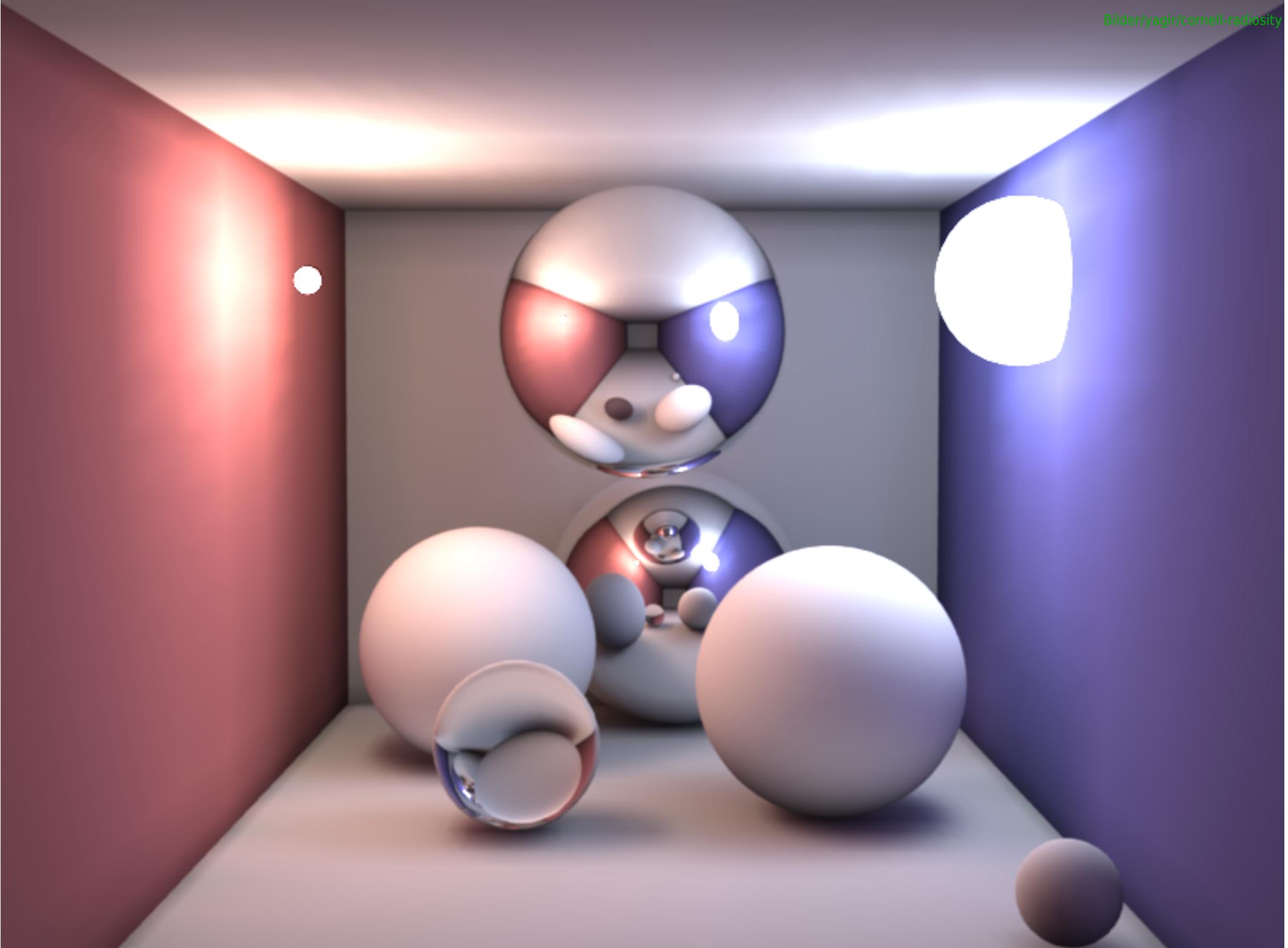
⇒ Ausgedehnte Lichtquellen müssen (i. Allg. bereits zu Beginn) in kleine **Patches** zerlegt werden.
 3. Die Unterteilung wird während des schrittweisen Lösungsaufbaus durchgeführt.
-

10.2.9 Zusammenfassung



- ⊕ trotz der vielen Vereinfachungen gute Simulation von
 - (mehrfacher) diffuser Reflexion
 - ausgedehnten Lichtquellen
- ⊕ unabhängig von Betrachtungsposition
 - ⇒ für statische Szenen Vorabberechnung möglich, dann sehr schnelles Rendering
- ⊖ keine Berücksichtigung winkelabhängiger Phänomene:
 - winkelabhängige Reflexion
 - Refraktion
- ⊖ feine Unterteilung in Patches nötig für gute Ergebnisse
- ⊖ Formfaktoren schwierig zu berechnen
- ⊖ insgesamt sehr aufwändig

Bemerkung 10.26: Das Radiosity-Verfahren wird oft auf **Innenräume** angewandt (Annahme eines geschlossenen Systems!).



10.3 Rendergleichung und Monte-Carlo-Integration

Notation von Funktionsargumenten mit Richtungen (nach [DBB06]):

- $f(x \rightarrow \Theta)$ Funktion $f(x, \Theta)$, vom Punkt x ausgehend in Richtung Θ
- $f(x \leftarrow \Theta)$ Funktion $f(x, \Theta)$, den Punkt x aus Richtung Θ treffend
- $f(x, \Theta \leftrightarrow \Psi)$ Funktion $f(x, \Theta, \Psi)$, aus Richtung Θ auftreffend und in Richtung Ψ verlassend bzw. umgekehrt (impliziert Helmholtz-Reziprozität [Umkehrbarkeit])
- $(\Psi^T \cdot \Theta)$ Kosinus als Skalarprodukt normalisierter Vektoren

10.3.1 Radiance

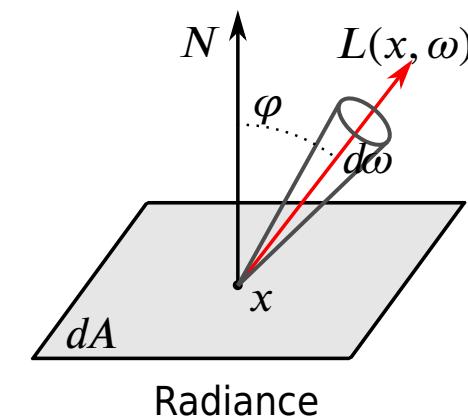
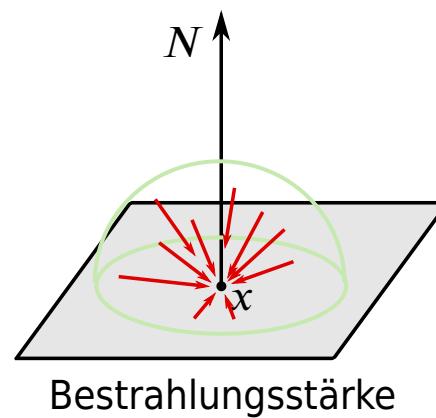
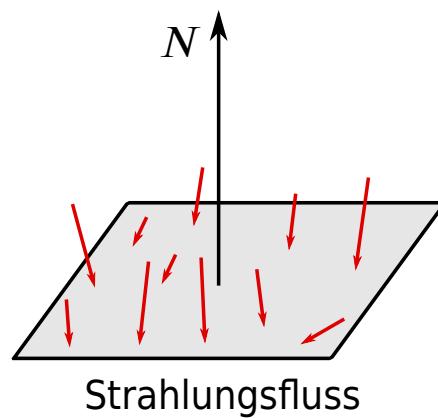
Radiance (Strahldichte) (vgl. 10.2.2): die von einem Objekt pro Zeit- und sichtbarer Flächeneinheit in eine bestimmte Richtung abgestrahlte Energie pro Einheitsraumwinkel

$$L(x, \omega) = \frac{d^2\Phi(x, \omega)}{dA \cos(\varphi) d\omega}$$

Φ Strahlungsleistung (Strahlungsfluss, radiant flux)

$dA \cos(\varphi)$ projiziertes Flächenelement

$d\omega$ Raumwinkelement



10.3.2 Rendergleichung

Rendergleichung (rendering equation)

1986 unabhängig von Immel/Cohen/Greenberg und Kajiya aufgestellt:

Radiance von x in Richtung Θ ausgesendet

= emittierte Radiance + reflektierte Radiance (aus allen Richtungen Ψ der Halbkugel Ω)

James Thomas Kajiya
* 1951
Informatiker

Foto: Derrick Coetzee
Quelle: https://de.wikipedia.org/wiki/Datei:Jim_Kajiya.jpg
Lizenz: CC-BY



- | | |
|-------------------------------------|--|
| $L(x \rightarrow \Theta)$ | Radiance ausgehend von x in Richtung Θ |
| $L_e(x \rightarrow \Theta)$ | von x in Richtung Θ emittierte Radiance |
| $f(x, \Theta \leftrightarrow \Psi)$ | BRDF |
| $L(x \leftarrow \Psi)$ | Radiance aus Richtung Ψ |
| N | Normale |

BRDF: beschreibt Reflexions- und Absorptionseigenschaften des Materials (siehe 10.3.7)

Die Radiance ist entlang gerader Linien (Strahlen) konstant (zumindest im Vakuum).

Ist von einem Punkt x in Richtung Ψ der Punkt x' sichtbar, dann gilt:

Radiance, die x aus Richtung Ψ empfängt = Radiance, die x' in Richtung $-\Psi$ aussendet

also

$$L(x \leftarrow \Psi) = L(x' \rightarrow -\Psi)$$

Formulierung mit Flächenstücken

Erinnerung an Bemerkung 10.12: $d\omega = \frac{\cos \varphi}{r^2} dA = \frac{N'^T \cdot -\Psi}{\|x-x'\|^2} dA$

Integriere über alle von x sichtbaren Flächenstücke A' :

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{A'} f(x, \Theta \leftrightarrow \Psi) \cdot L(x' \rightarrow -\Psi) \cdot G(x, x') \cdot V(x, x') dA'$$

mit

$$G(x, x') = \frac{(N^T \cdot \Psi) \cdot (N'^T \cdot -\Psi)}{\|x - x'\|^2}$$

$$V(x, x') = \begin{cases} 1 & \text{falls } x' \text{ von } x \text{ aus sichtbar ist} \\ 0 & \text{sonst} \end{cases}$$

Bemerkung 10.27: mit obiger Formulierung nicht modellierbar:

- Abhängigkeit von der Wellenlänge: Farbenzerstreuung (Dispersion) durch Prismen, chromatische Aberration o. ä. kann durch Sampling des Spektrums simuliert werden (vgl. 7.1.2).
 - Effekte der Wellenoptik wie Beugung (Diffraction), Interferenz, Polarisation
(Dünnfilm-Interferenz kann innerhalb der BRDF simuliert werden.)
 - Transluzenz (partiell lichtdurchlässige Medien), Approximation für Volumenstreuung (subsurface scattering) möglich oder Ausweitung auf Integration entlang des gesamten Pfades (full path integration) und Volumengrafik (volumetric rendering)
-

10.3.3 Zusammenhang zwischen Rendergleichung und Radiosity

Radiosity: gesamte von einem Punkt x aus in **alle Richtungen** abgestrahlte Leistung

Annahme 1: $L(x \rightarrow \Theta) = L(x)$ ist für alle Richtungen konstant (vollkommen diffuse Abstrahlung)

$$\begin{aligned}\Rightarrow B(x) &= \int_{\Omega} L(x \rightarrow \Theta) \cdot (N^T \cdot \Theta) d\omega \\ &= L(x) \cdot \int_{\Omega} (N^T \cdot \Theta) d\omega \\ &= \pi L(x) \quad (\text{vgl. Gleichung (10.3)})\end{aligned}$$

Formulierung der Rendergleichung mit Flächenstücken - multipliziert mit π - gibt:

$$B(x) = E_e(x) + \int_{A'} f(x, \Theta \leftrightarrow \Psi) \cdot B(x') \cdot G(x, x') \cdot V(x, x') dA'$$

Annahme 2: $f(x, \Theta \leftrightarrow \Psi) = \frac{\rho}{\pi}$ konstant (vollkommen diffuse Abstrahlung)

$$\Rightarrow B(x) = E_e(x) + \frac{\rho}{\pi} \cdot \int_{A'} B(x') \cdot G(x, x') \cdot V(x, x') dA'$$

Aufteilung in Patches ($A' = \bigcup_{j=1}^n A_j$) und Aufsummierung der Radiosities gibt:

$$B(x) = E_e(x) + \frac{\rho}{\pi} \cdot \sum_{j=1}^n \int_{A_j} B(x_j) \cdot G(x, x_j) \cdot V(x, x_j) dA_j$$

Annahme 3: Radiosity innerhalb der Patches konstant; verwende Mittelwert über die Fläche:

$$B_i = \frac{1}{A_i} \cdot \int_{A_i} B(x_i) dx_i$$

Einsetzen ergibt:

$$\begin{aligned} B_i &= E_i + \frac{1}{A_i} \cdot \int_{A_i} \frac{\rho_i}{\pi} \cdot \sum_{j=1}^n B_j \cdot \int_{A_j} G(x_i, x_j) \cdot V(x_i, x_j) dA_j dA_i \\ &= E_i + \rho_i \cdot \frac{1}{A_i} \cdot \sum_{j=1}^n B_j \cdot \int_{A_i} \int_{A_j} \frac{G(x_i, x_j)}{\pi} \cdot V(x_i, x_j) dA_j dA_i \\ &= E_i + \rho_i \cdot \frac{1}{A_i} \cdot \sum_{j=1}^n B_j \cdot \int_{A_i} \int_{A_j} \frac{(N_i^T \cdot \Psi) \cdot (N_j^T \cdot -\Psi)}{\pi \|x_i - x_j\|^2} H_{ji} dA_j dA_i \\ &= E_i + \rho_i \cdot \frac{1}{A_i} \cdot \sum_{j=1}^n B_j \cdot \int_{A_i} \int_{A_j} \frac{\cos \varphi_i \cdot \cos \varphi_j}{\pi r_{ij}^2} H_{ji} dA_j dA_i \end{aligned}$$

Das ist exakt Gleichung (10.8) eingesetzt in Gleichung (10.5), dividiert durch A_i .

also: Die Radiositygleichung ist ein Spezialfall der Rendergleichung.

10.3.4 Monte-Carlo-Integration

Frage: Wie berechnet man

$$I = \int_{\Omega} g(x) dx ?$$

Idee: Wähle n zufällige Richtungen, gleichmäßig verteilt über die Einheitshalbkugel Ω .

$$I \approx \int_{\Omega} dx \cdot \frac{1}{n} \sum_{i=1}^n g(x_i) = \frac{2\pi}{n} \sum_{i=1}^n g(x_i)$$

Aus der Rendergleichung wird dann:

$$L_{out} \approx L_e + \frac{2\pi}{n} \sum_{i=1}^n (f_i \cdot L_i \cdot (N^T \cdot \Psi_i))$$

Konvergenz: Standardabweichung σ hat Konvergenzordnung $\mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$

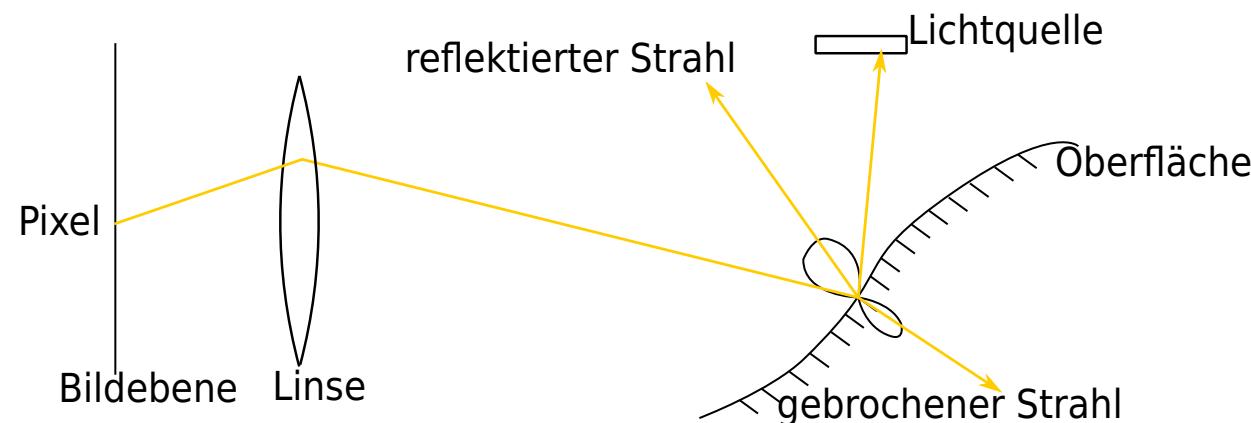
10.3.5 Erweiterungen des Raytracing-Modells

Erweiterung des Whitted-Ansatzes zu **diffusem Raytracing (stochastisches Raytracing, distributed ray tracing)** (Cook/Porter/Carpenter 1984)

Idee: verfolge mehrere zufällig ausgewählte Strahlen

ermöglicht:

- weiche Schatten (durch mehrere Strahlen zu ausgedehnten Lichtquellen)
- Tiefunschärfe (Wahl mehrerer Punkte in Blendenöffnung des virtuellen Kamerasytems)
- verschwommene Reflexion bei glänzenden Oberflächen (mehrere Reflexionsstrahlen)
- Lichtdurchlässigkeit (Transluzenz) (mehrere Brechungsstrahlen)
- Bewegungsunschärfe (zeitliche Verteilung der Strahlen)



Robert L. Cook
* 1952, Knoxville, Tennessee
Computergraphiker, Pixar
Foto: U.S. General Services Administration
Quelle: https://commons.wikimedia.org/wiki/File:Rob_Cook.png



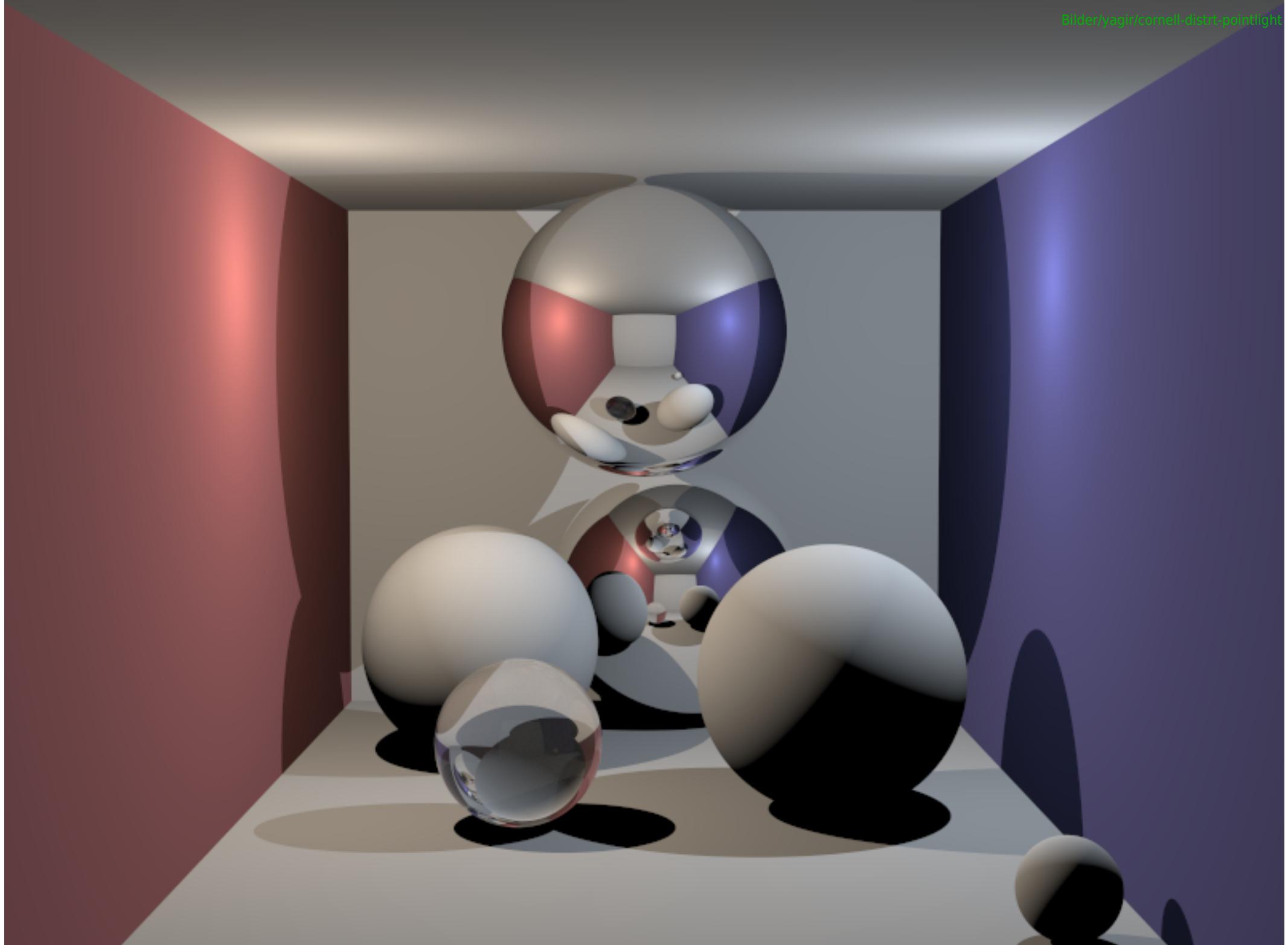
Thomas K. Porter
Computergraphiker, Pixar

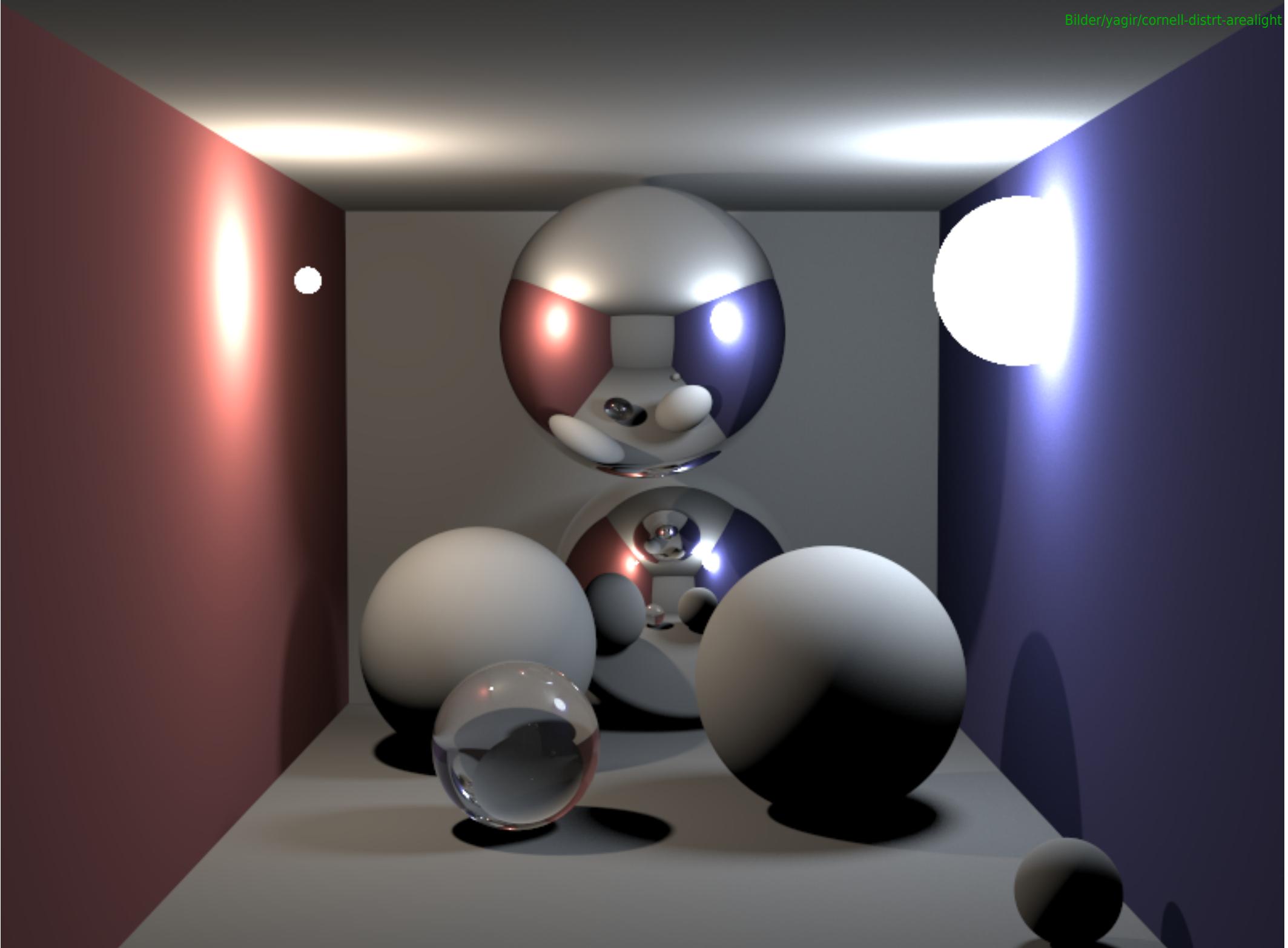
Loren C. Carpenter
* 1947, Brighton, Michigan
Computergraphiker, Pixar

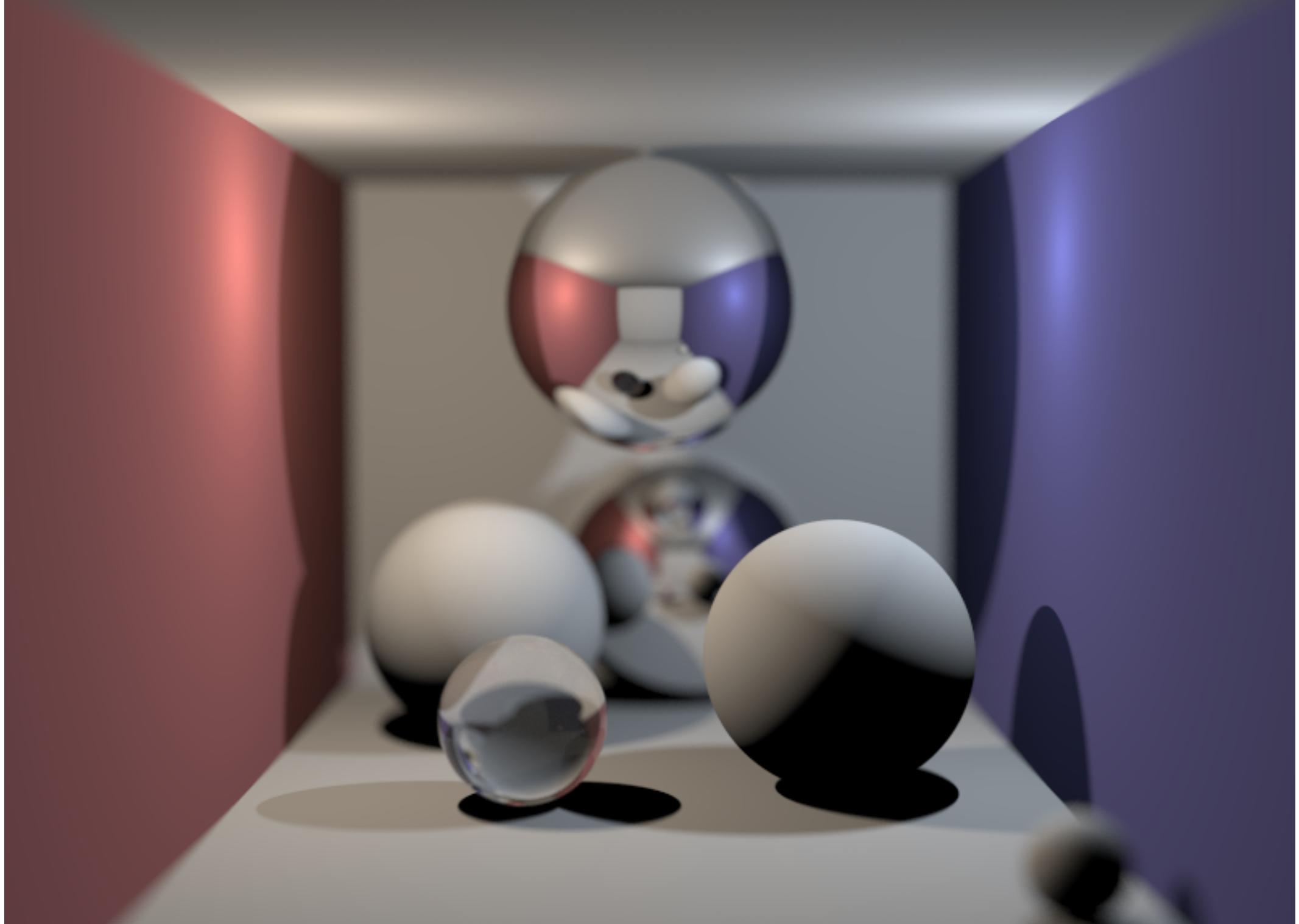
aber:

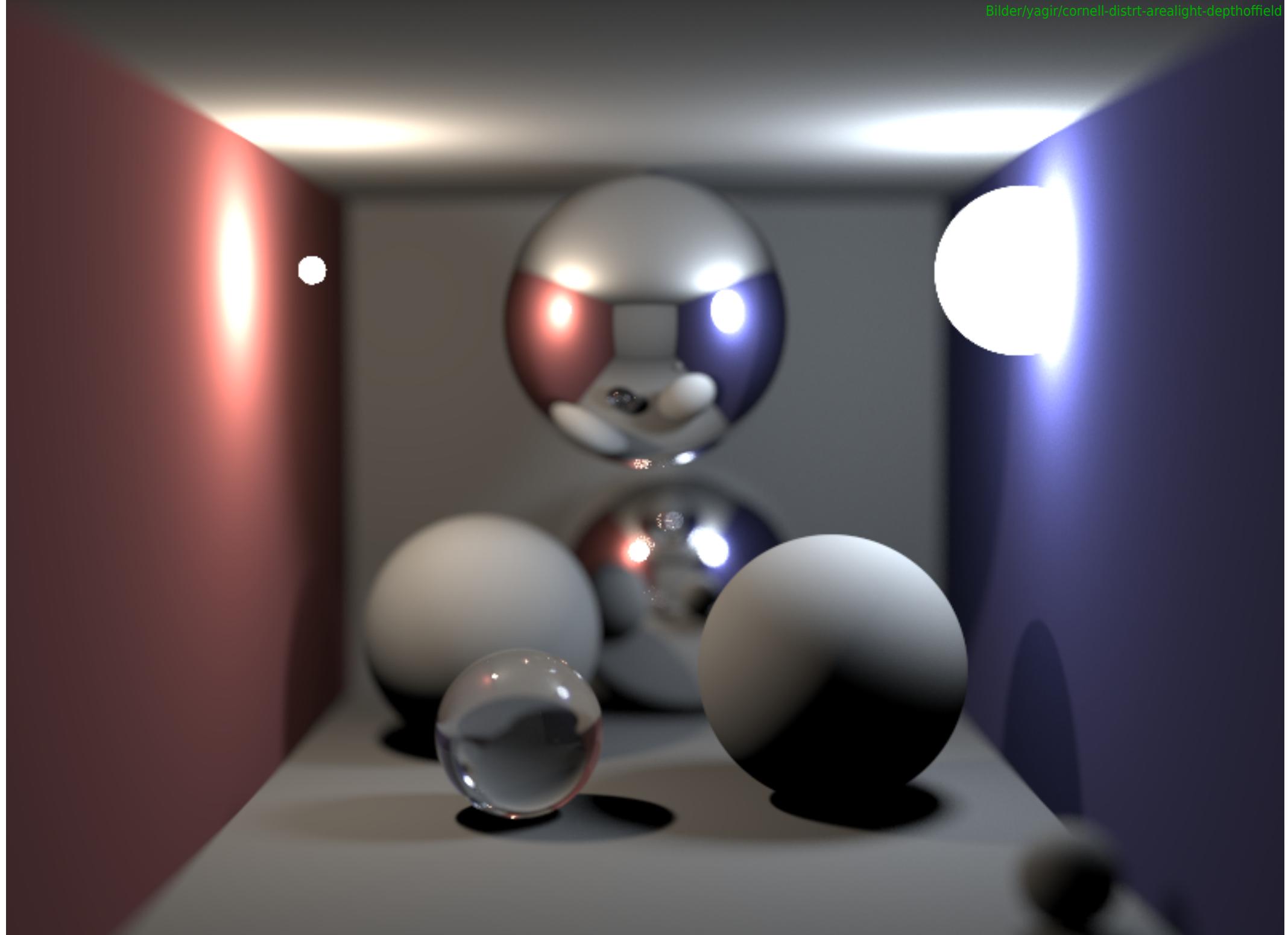
- nach wie vor nur Strahlen zu Lichtquellen, für Reflexion und Brechung
- keine Berücksichtigung indirekter Beleuchtung (diffuse Reflexion) – außer ambientem Anteil
- löst nicht die Rendergleichung
- Aufwand durch starke Verzweigung der Strahlen

Umsetzung: keine Strahlvervielfachung an jedem Objekt, sondern mehrere Strahlen pro Bildpunkt (wie für Antialiasing) – danach Verzweigung wie bei Standard-Raytracing mit zufälligen Richtungen





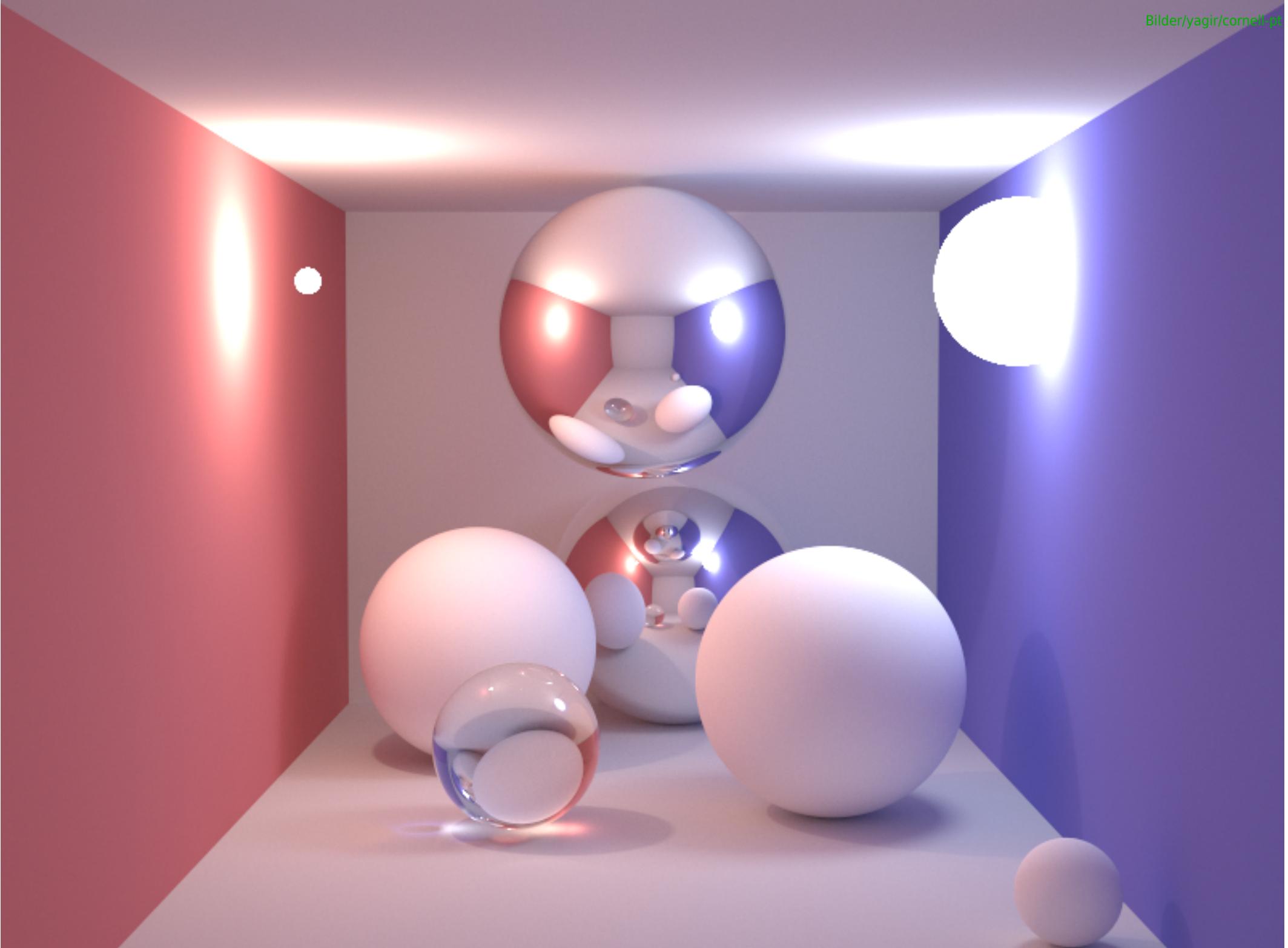




10.3.6 Path Tracing

Idee: (nach Kajiya, 1986)

- für jeden Bildpunkt mehrere Strahlen
 - Verfolgung eines einzelnen Strahls, d. h. im Gegensatz zu Raytracing keine Aufspaltung in reflektierten oder gebrochenen Strahl
 - Reflexion oder Brechung wird zufällig entschieden (einschließlich diffuser Reflexion)
 - zufällige Wahl der Richtung anhand BRDF, dadurch einheitliche Behandlung von indirekter Beleuchtung, Spiegelung, Brechung
 - bei Treffen eines Objekts Lichtbeitrag aus einer zufällig ausgewählten Lichtquelle
(auch möglich ohne Strahlverfolgung zu Lichtquellen, aber dann langsame Konvergenz durch zufälliges Treffen der Lichtquellen)
- ⊕ Lösung der Rendergleichung
- ⊖ benötigt viele Strahlen (langsam), sonst Bildrauschen

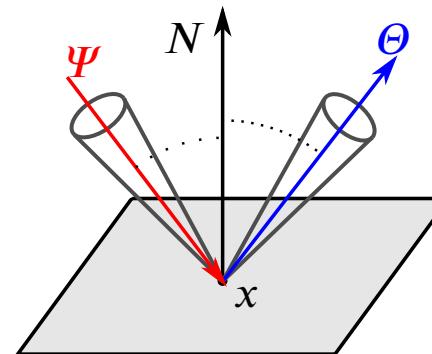


10.3.7 BRDF

BRDF: bidirektionale Reflektanzverteilungsfunktion (bidirectional reflectance distribution function): beschreibt Reflexions- und Absorptionseigenschaften des Materials

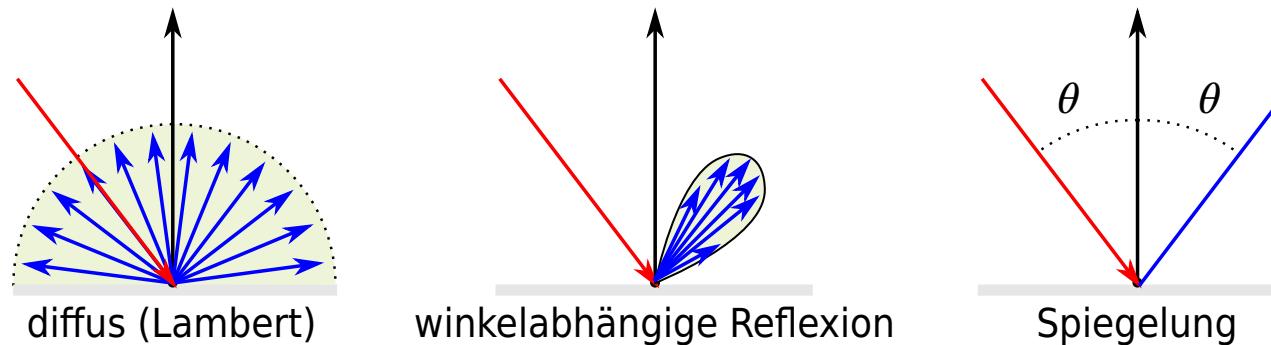
formale Definition:

$$f(x, \Theta \leftrightarrow \Psi) = \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi) (N^T \cdot \Psi) d\omega} \quad \left[\frac{1}{sr} \right]$$



anschauliche Bedeutung: Verhältnis zwischen in Richtung Θ ausgestrahltem und aus Richtung Ψ empfangenen Licht im Punkt x

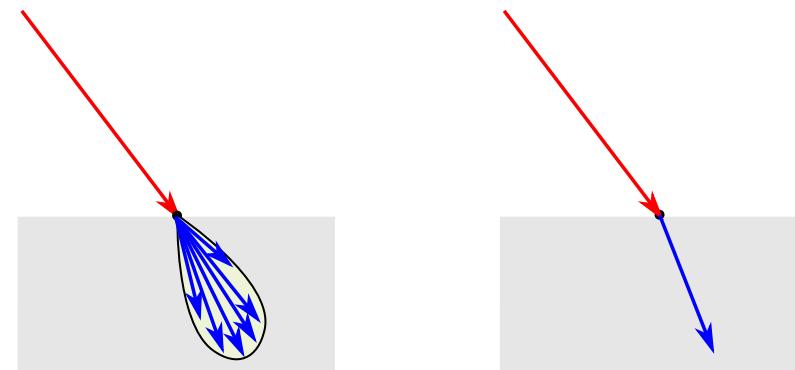
(eigentlich Quotient aus Strahlungsdichte und Bestrahlungsstärke)



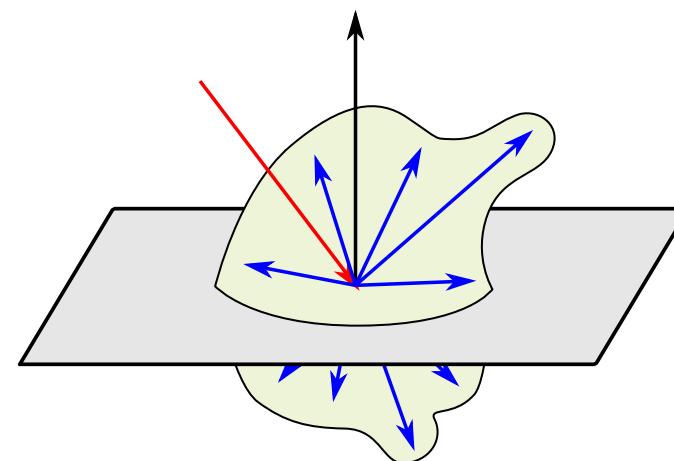
notwendige Eigenschaften für physikalische Korrektheit:

- $f(x, \Theta \leftrightarrow \Psi) \geq 0$ (Positivität)
- $f(x, \Theta \leftrightarrow \Psi) = f(x, \Psi \leftrightarrow \Theta)$ (Helmholtz-Reziprozität)
- $\int_{\Omega} f(x, \Theta \leftrightarrow \Psi) (N^T \cdot \Psi) d\omega \leq 1$ für alle Θ (Energieerhaltung)

BTDF: bidirectional transmittance distribution function, analog für lichtdurchlässige Materialien für Bereich unter Oberfläche



BSDF: bidirectional scattering distribution function (BRDF + BTDF zusammen)



BRDF: Beispiele

diffus (Lambert):

$$f = \frac{R_d}{\pi}$$

Spiegelung:

$$f = R_s \cdot \frac{\delta(\Psi - S(\Theta, N))}{N^T \cdot \Psi}$$

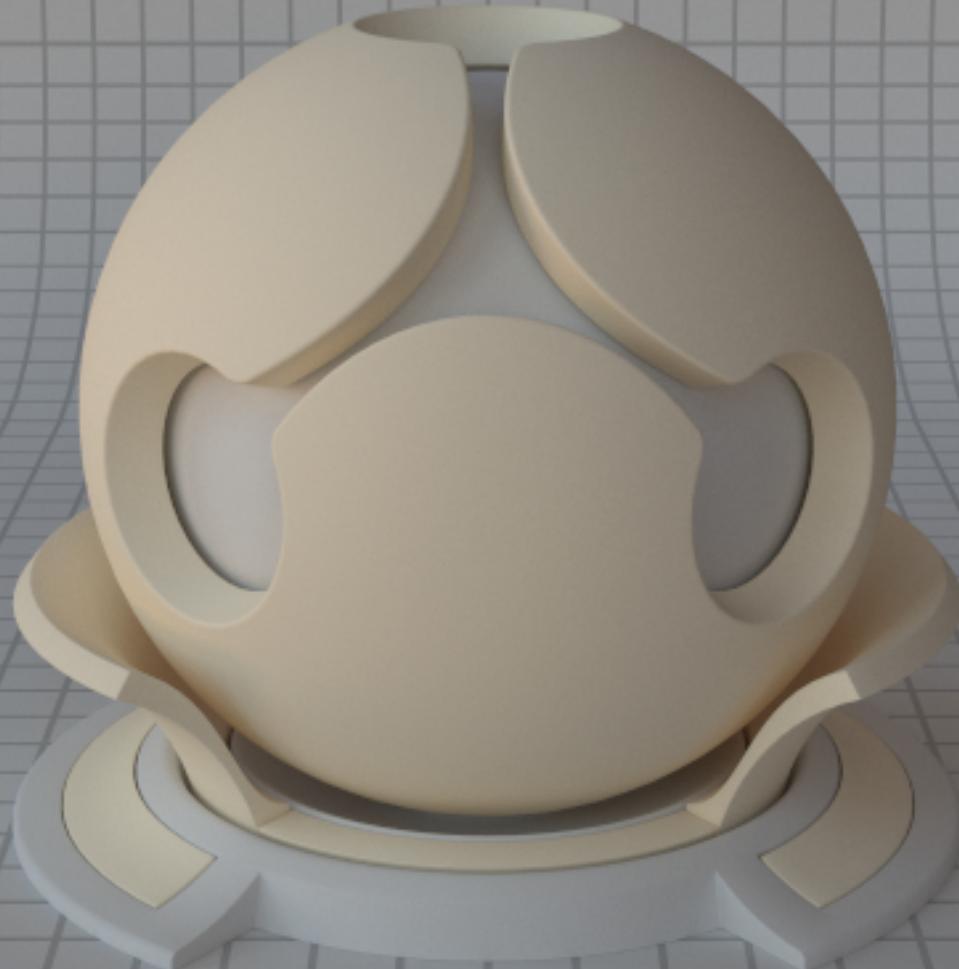
($\delta(0) = 1, \delta(w) = 0$ für $w \neq 0$, $S(\Theta, N)$: Spiegelrichtung)

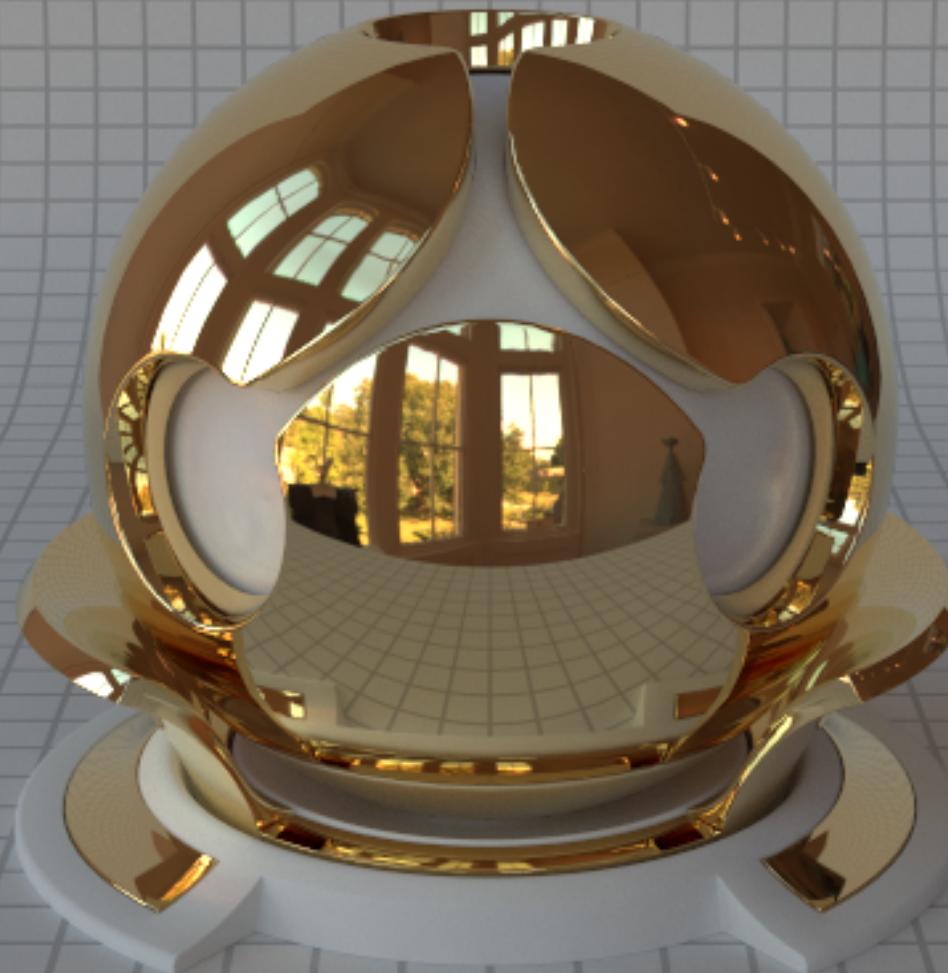
Damit folgt:

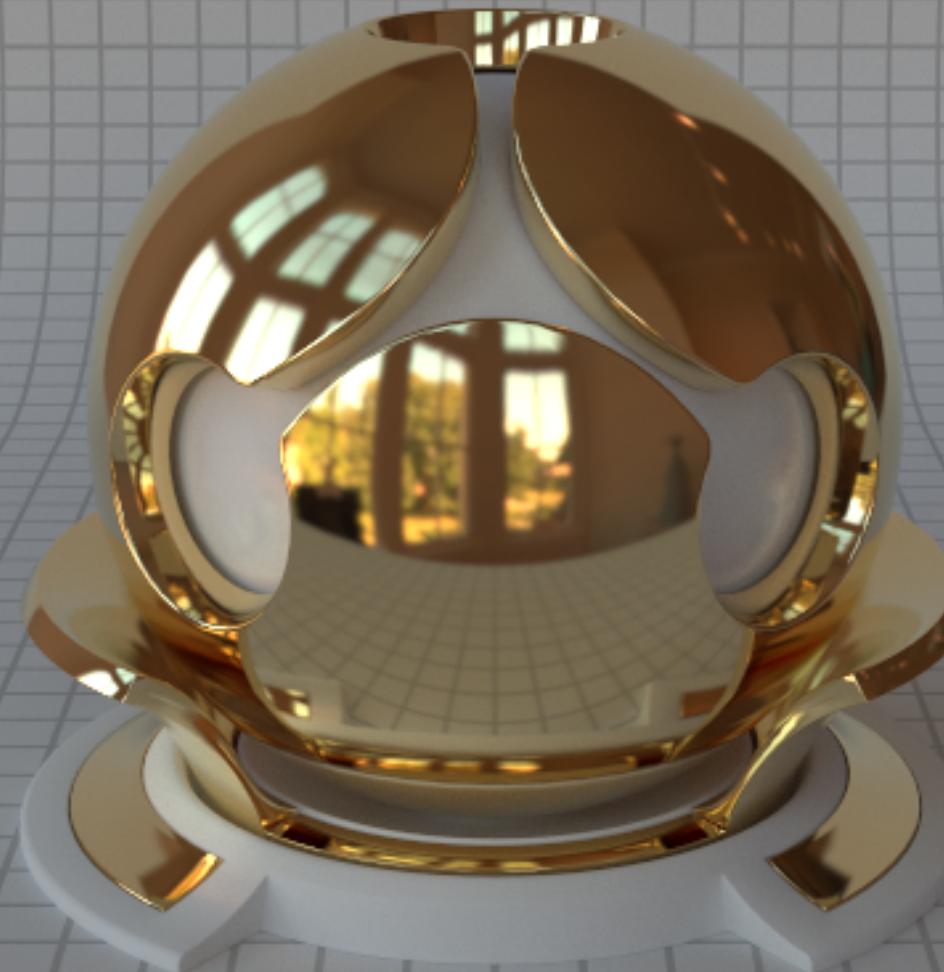
$$\int_{\Omega} R_s \cdot \frac{\delta(\Psi - S(\Theta, N))}{N^T \cdot \Psi} \cdot L(x \leftarrow \Psi) \cdot (N^T \cdot \Psi) d\omega = R_s \cdot L(x \leftarrow S(\Theta, N))$$

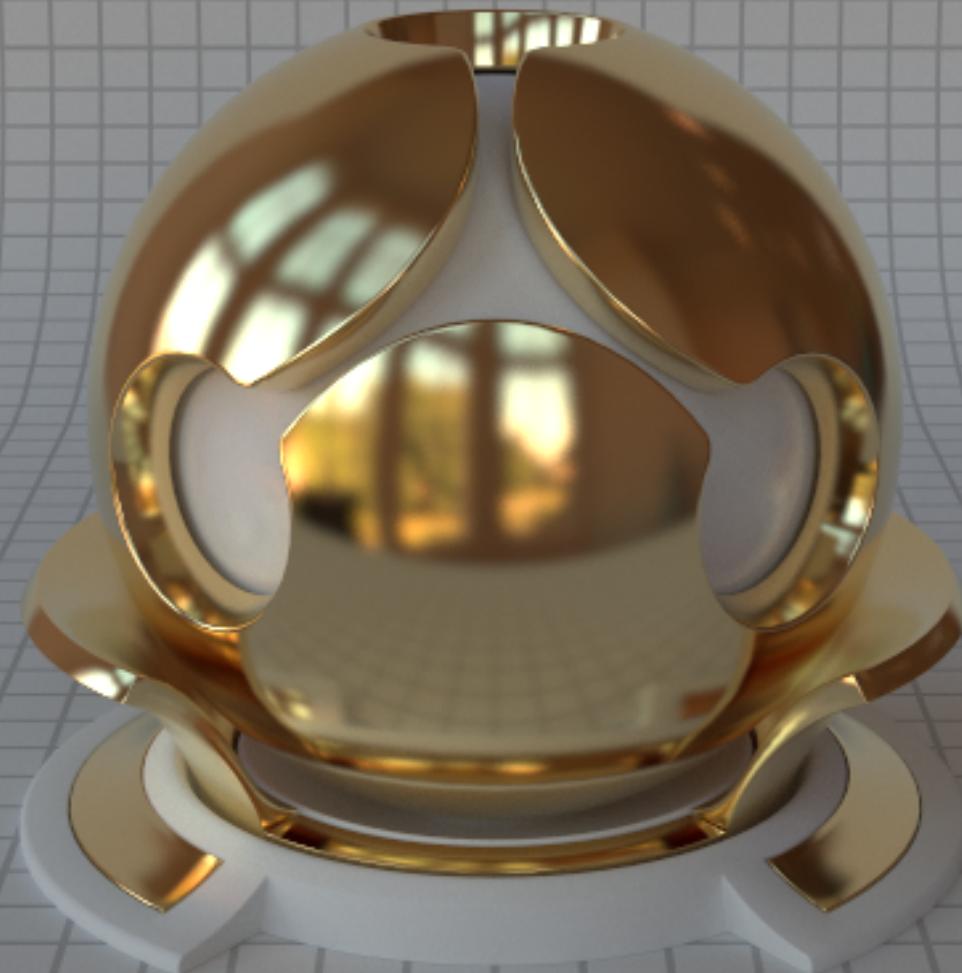
winkelabhängige Reflexion (Phong):

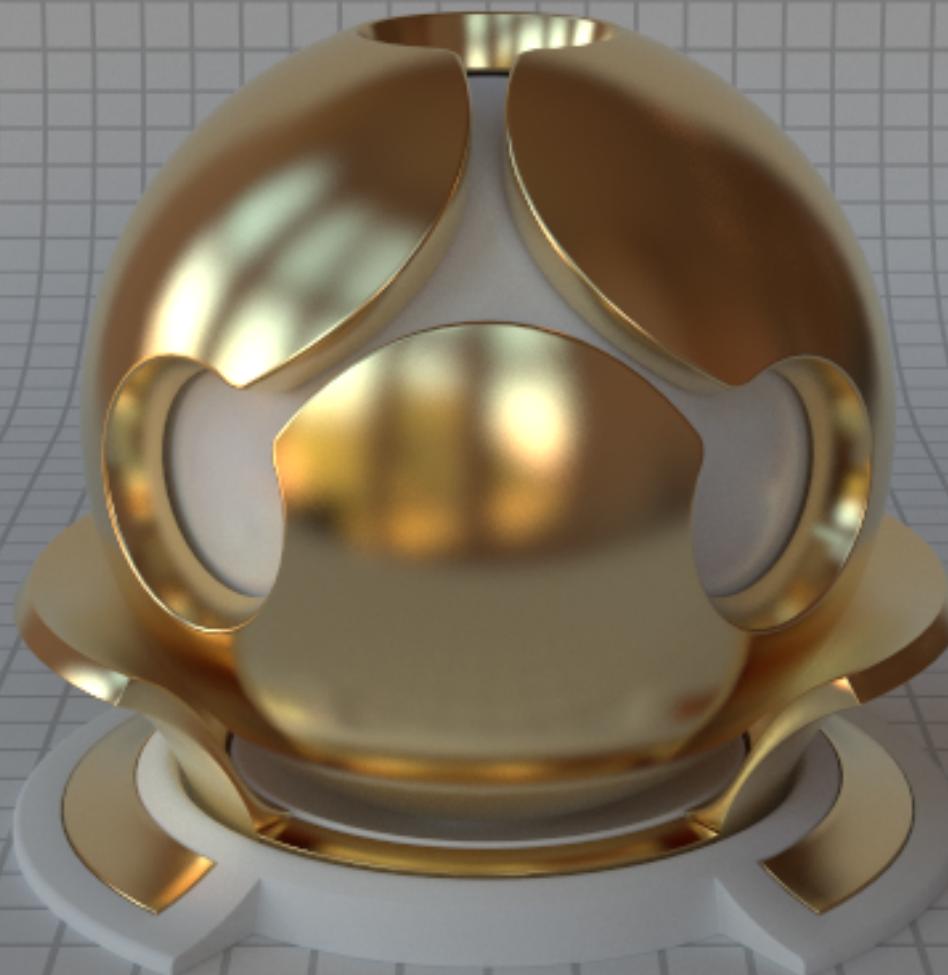
$$f = R_w \cdot \frac{\nu + 2}{2\pi} \cdot \frac{(S(\Psi, N)^T \cdot \Theta)^\nu}{N^T \cdot \Psi}$$

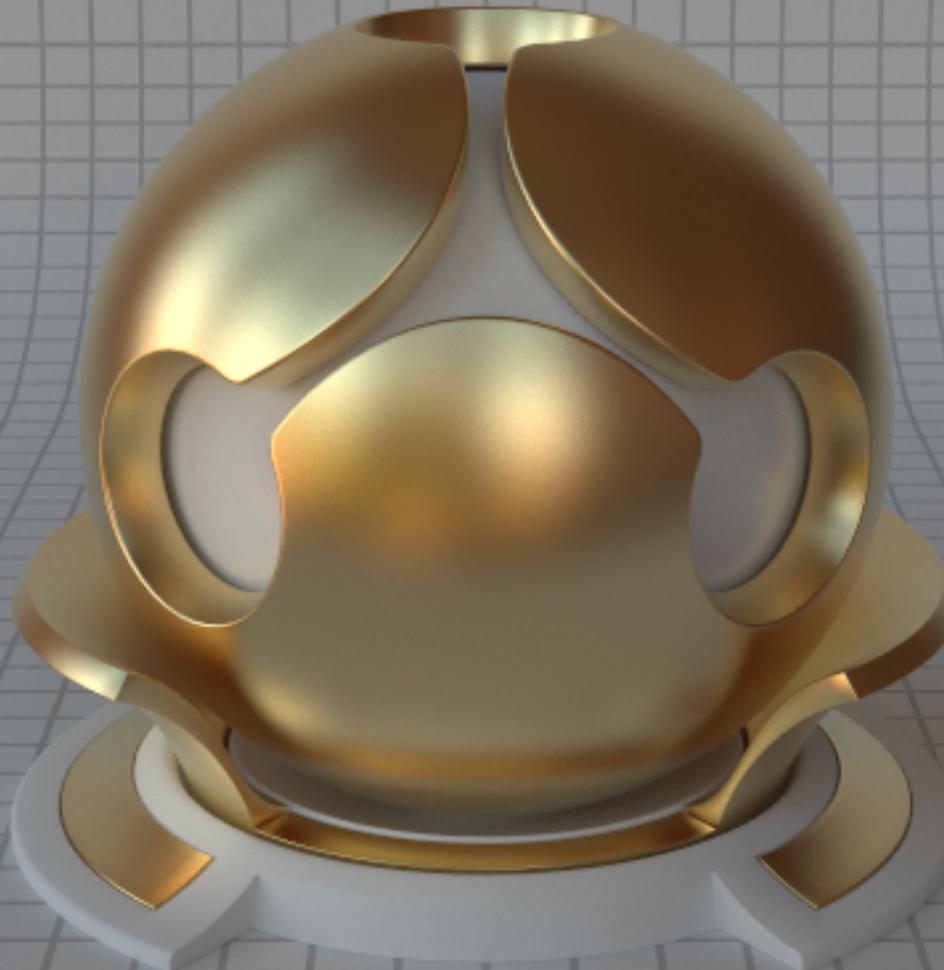


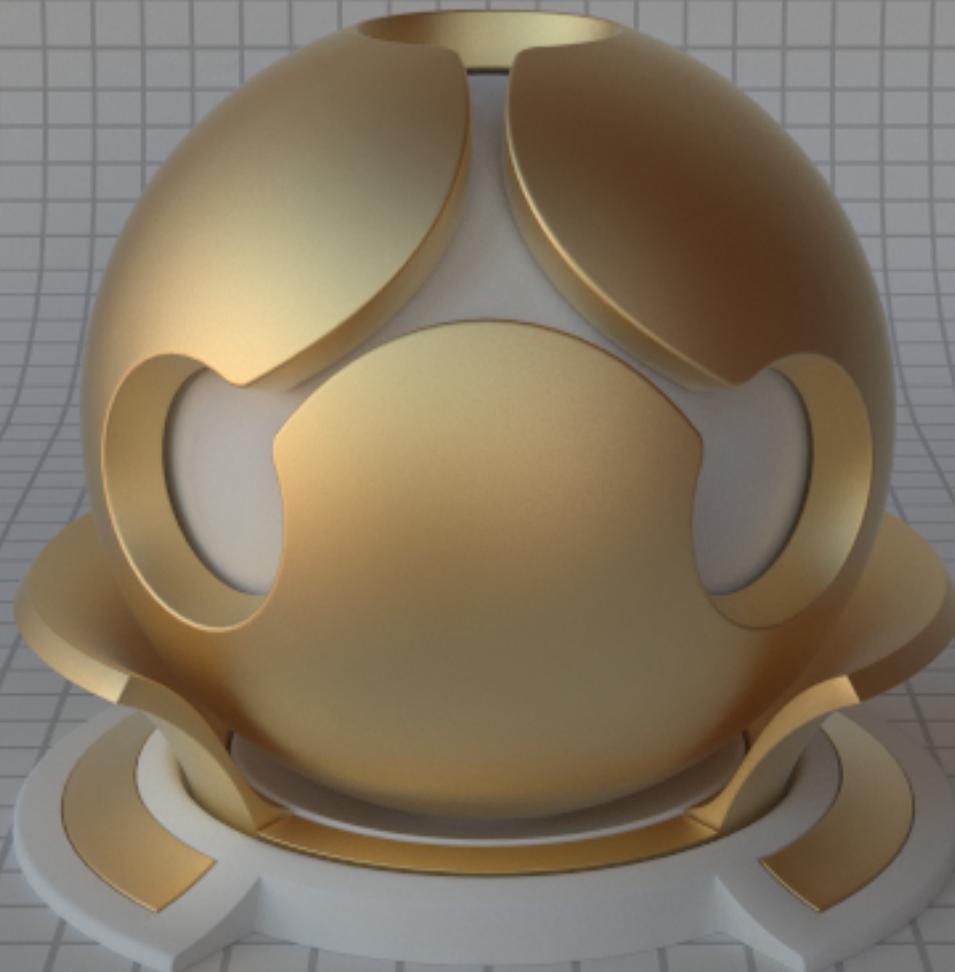












10.3.8 Kombination von BRDFs

z. B. für Glas, glänzende Oberflächen, ...

Beobachtung: Der Anteil gebrochenen und reflektierten Lichts hängt vom Winkel ab.

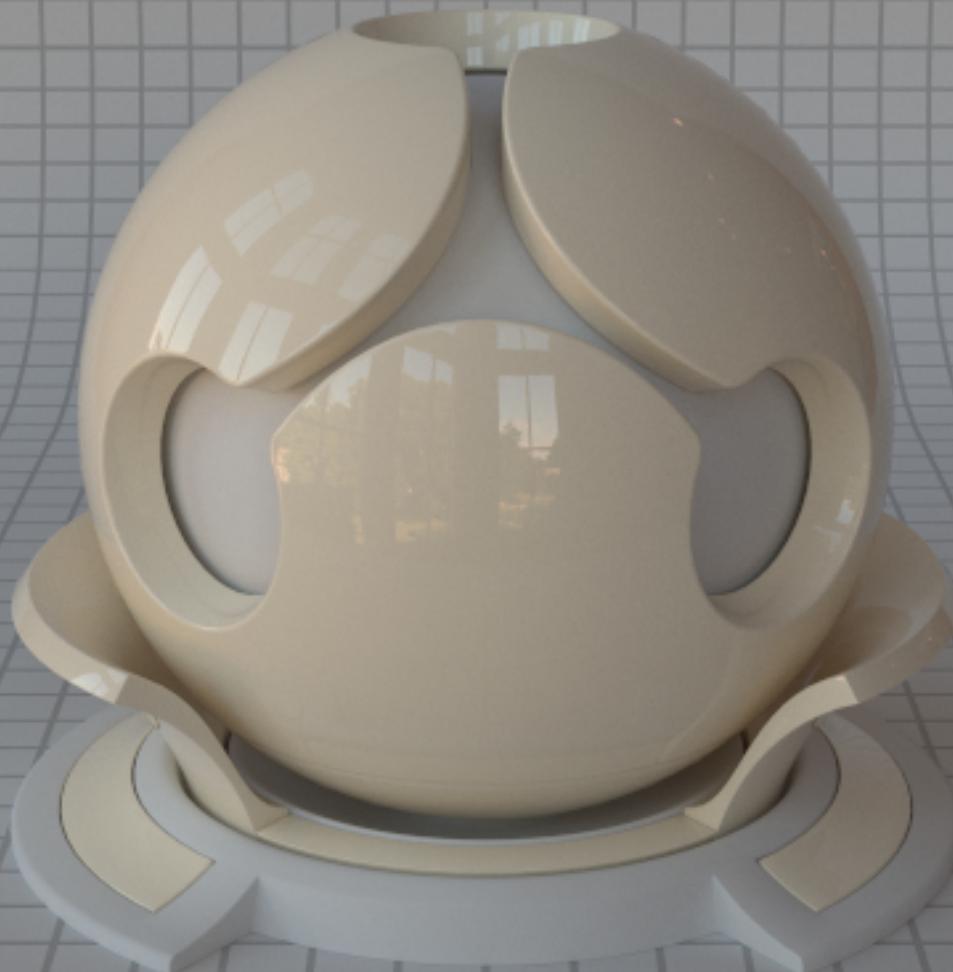
Beispiel: Wasseroberflächen spiegeln stärker, wenn man aus flachem Winkel darauf schaut.

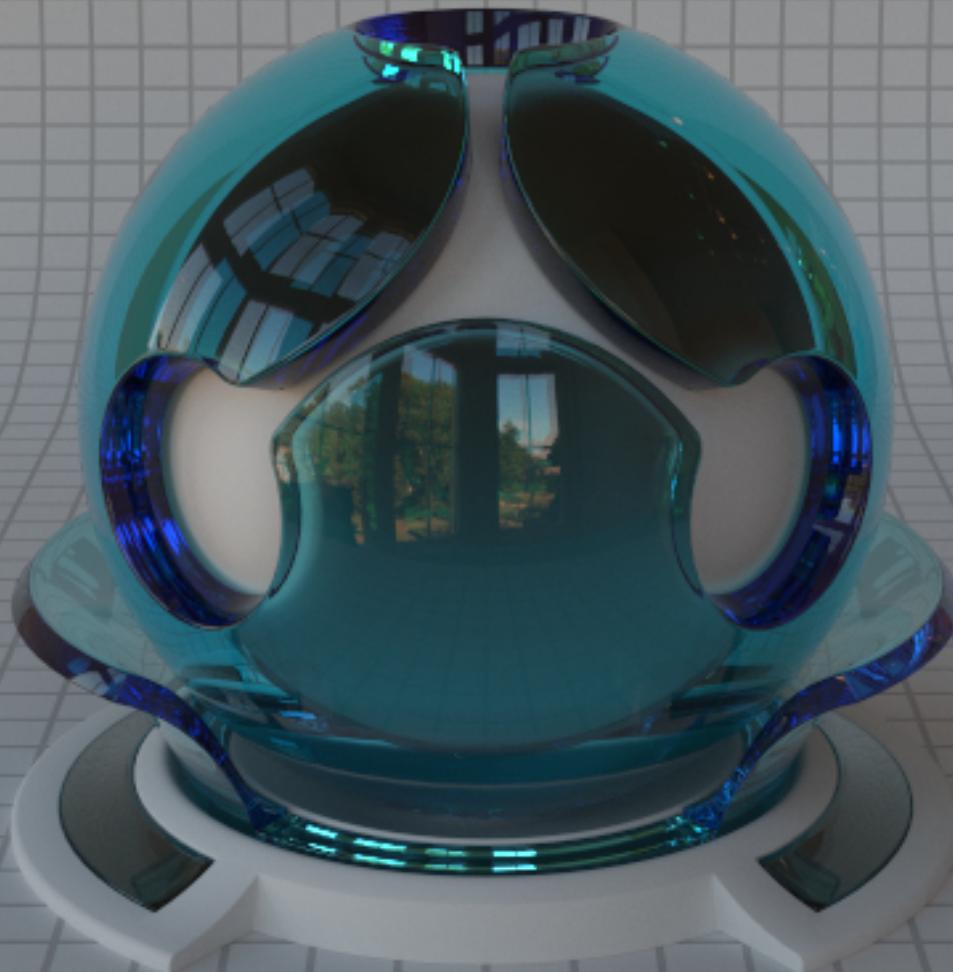
Berechnung durch fresnelsche Formeln aus parallel und senkrecht zur Einfallsebene polarisiertem Licht.

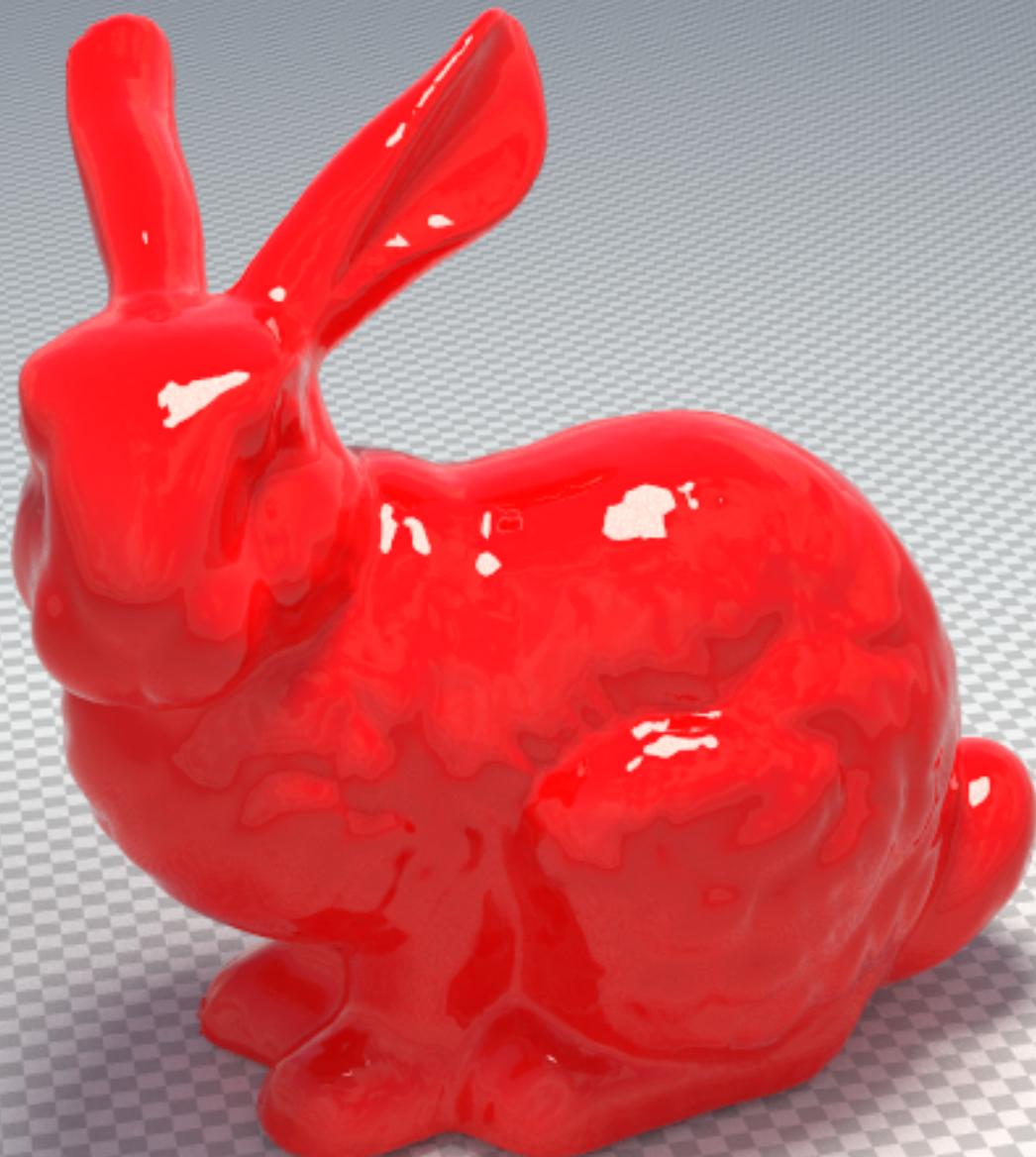
Verwendung in Path Tracing: Reflexionsgrad als Wahrscheinlichkeit bei der Auswahl zwischen mehreren BRDFs.

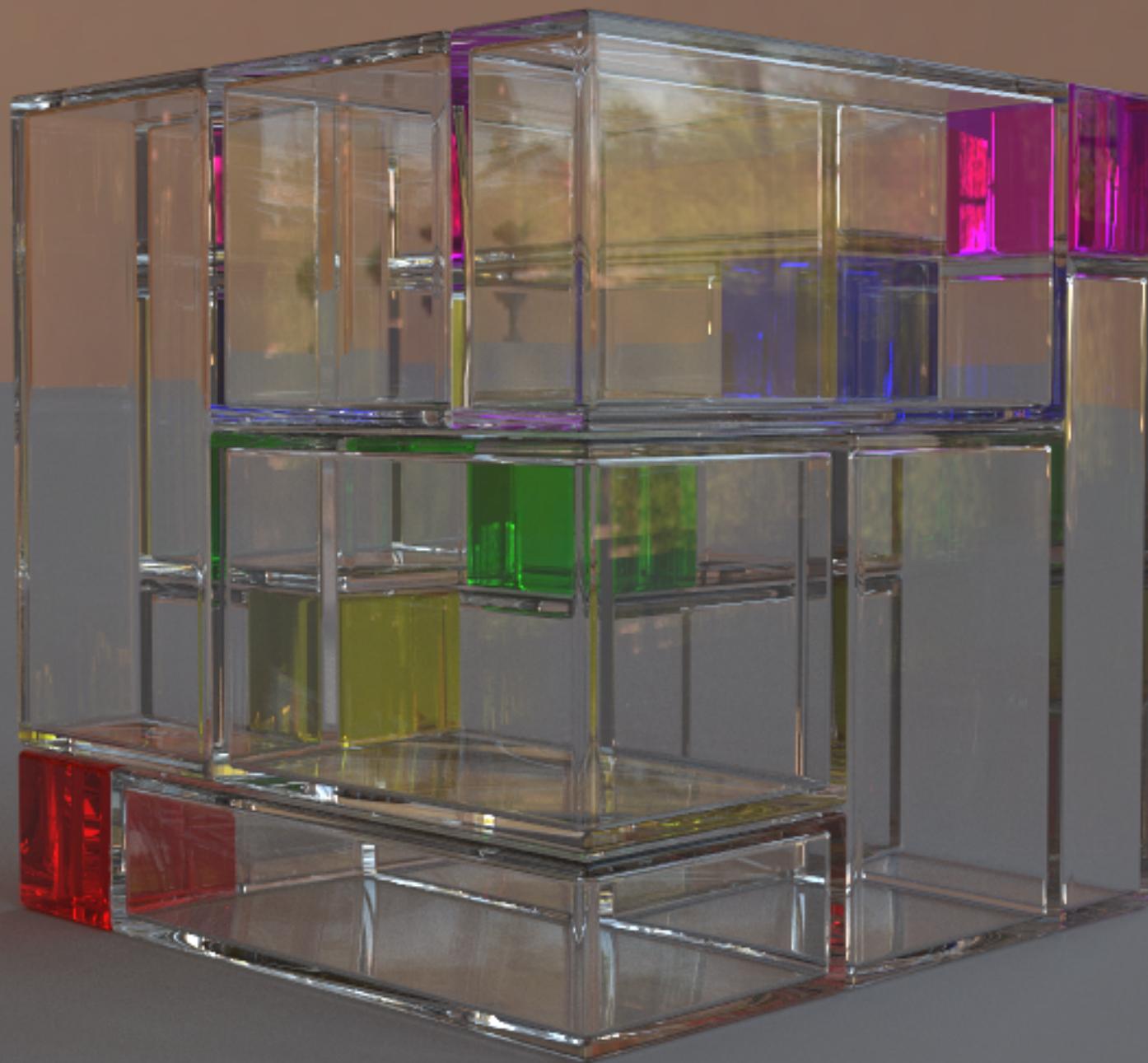
komplexere Materialen: modellierbar durch Schichten von BRDFs und Lichttransport dazwischen











10.3.9 Importance Sampling

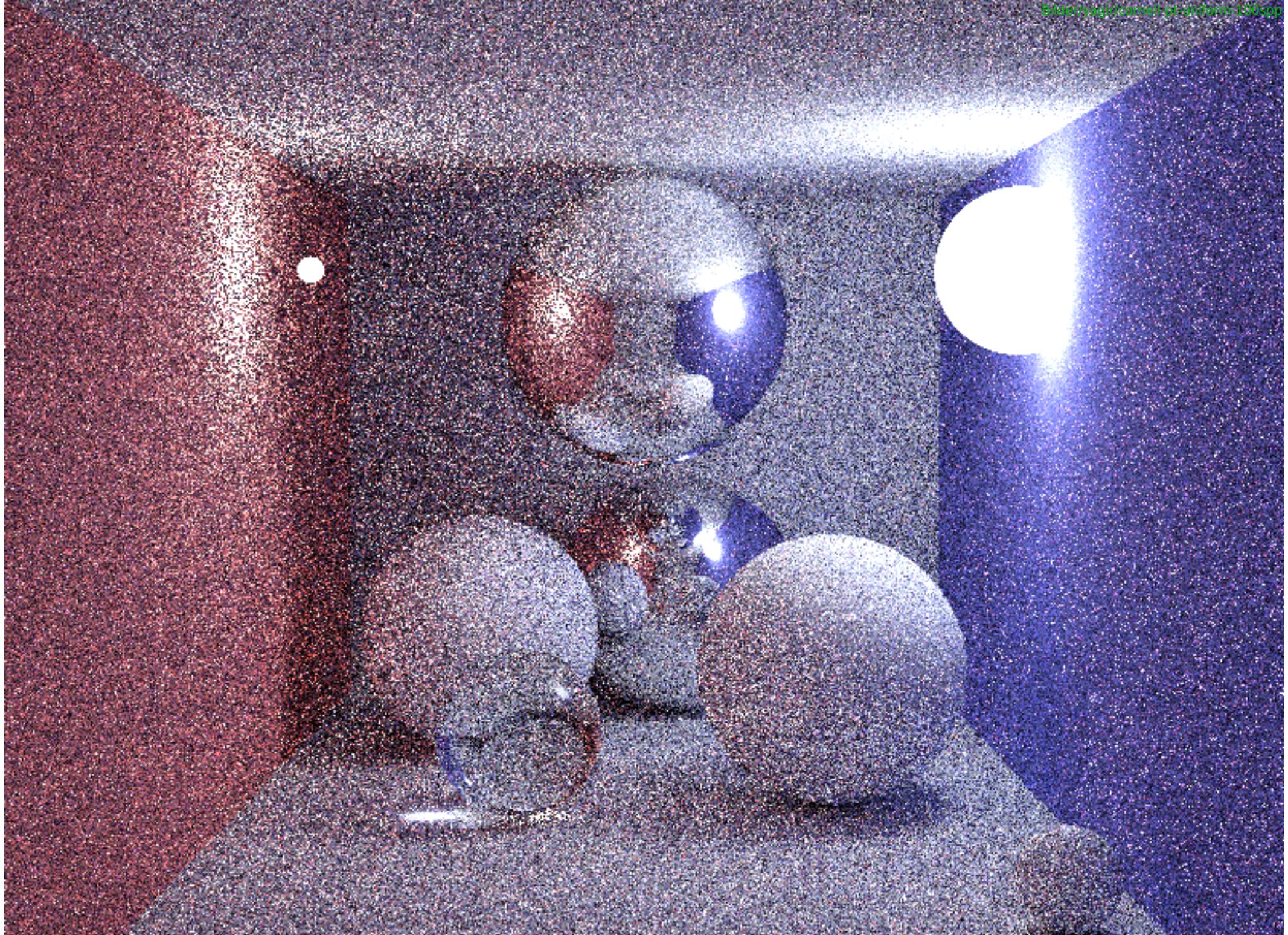
$$I \approx \frac{1}{n} \sum_{i=1}^n \frac{g(x_i)}{p(x_i)}$$

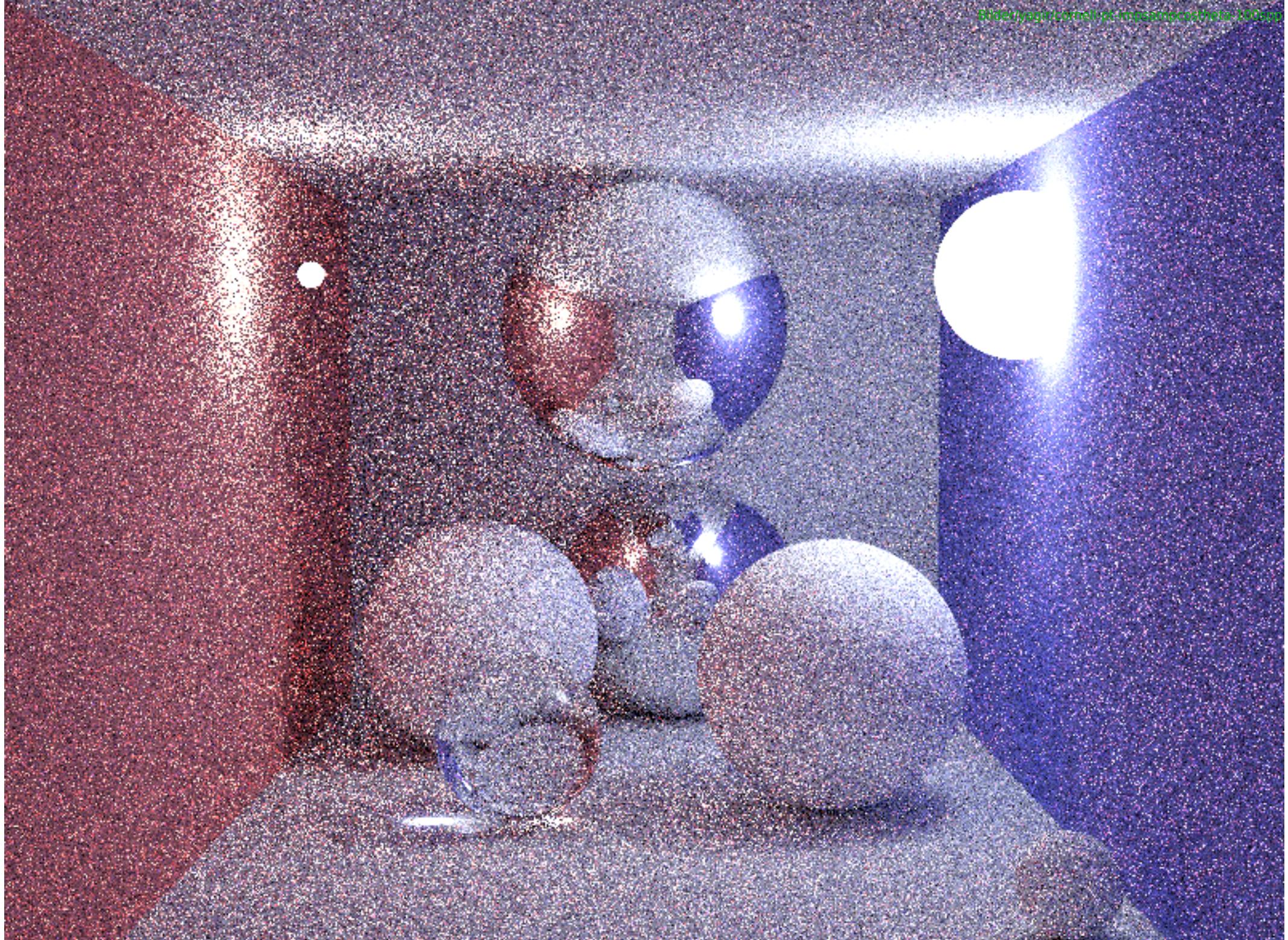
Rendergleichung wird zu:

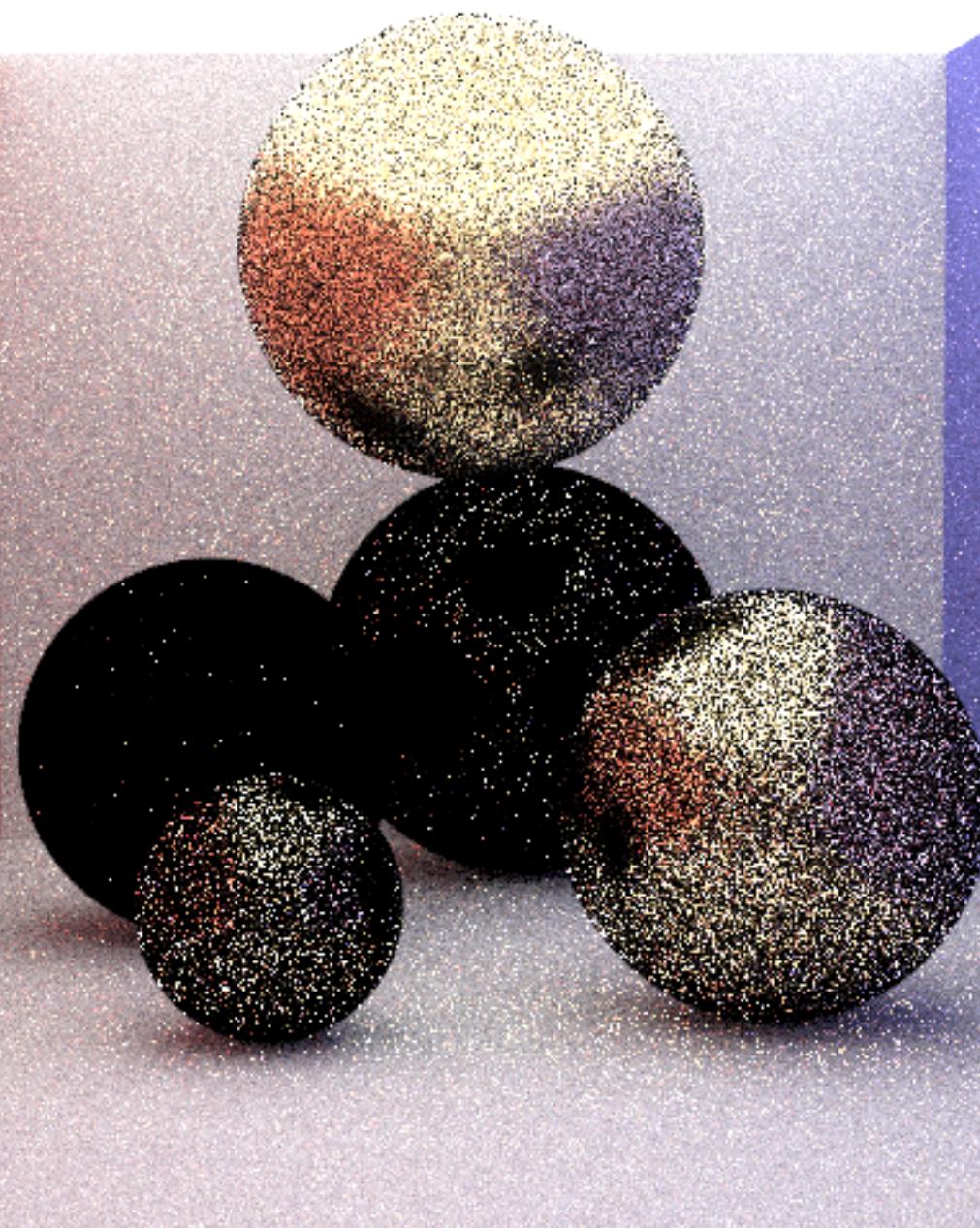
$$L_{out} \approx L_e + \frac{1}{n} \sum_{i=1}^n \left(\frac{f_i}{p_i} \cdot L_i \cdot (N^T \cdot \Psi_i) \right)$$

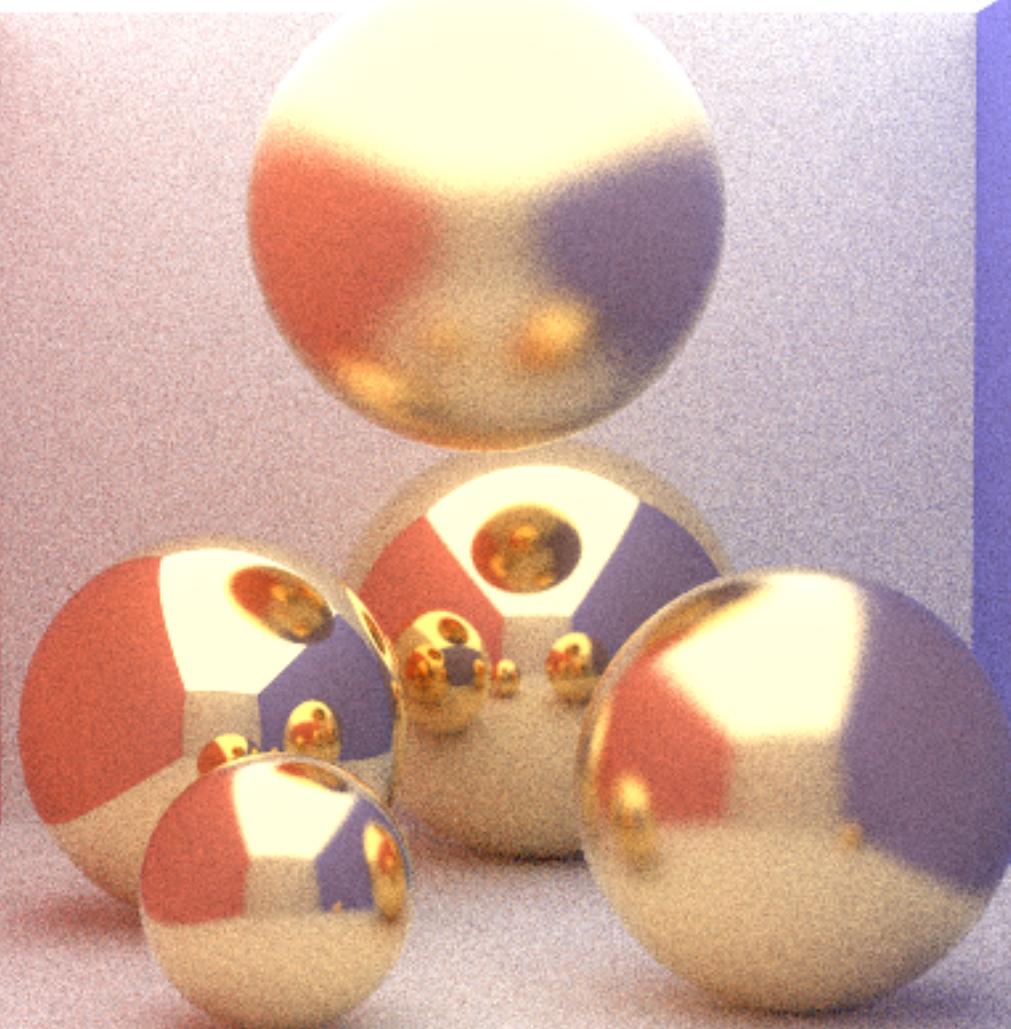
Beispiel:

$$p(\Psi) = \frac{N^T \cdot \Psi}{\pi}$$









10.3.10 Direkte Beleuchtung

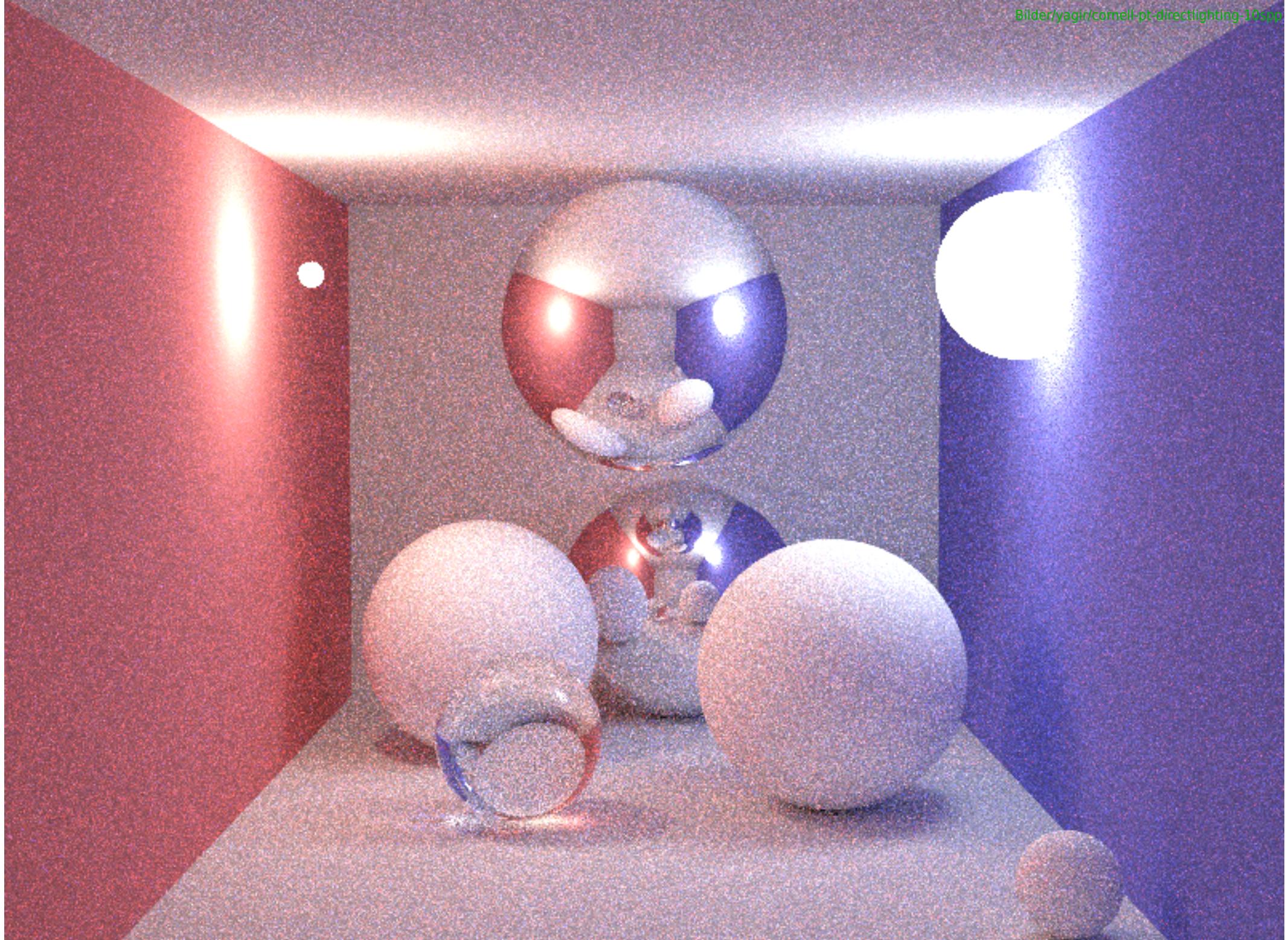
Idee: Bei Punkten in der Szene, bei denen klar ist, dass Sie direkt von einer Lichtquelle getroffen werden, soll diese nicht zufällig gefunden werden müssen.

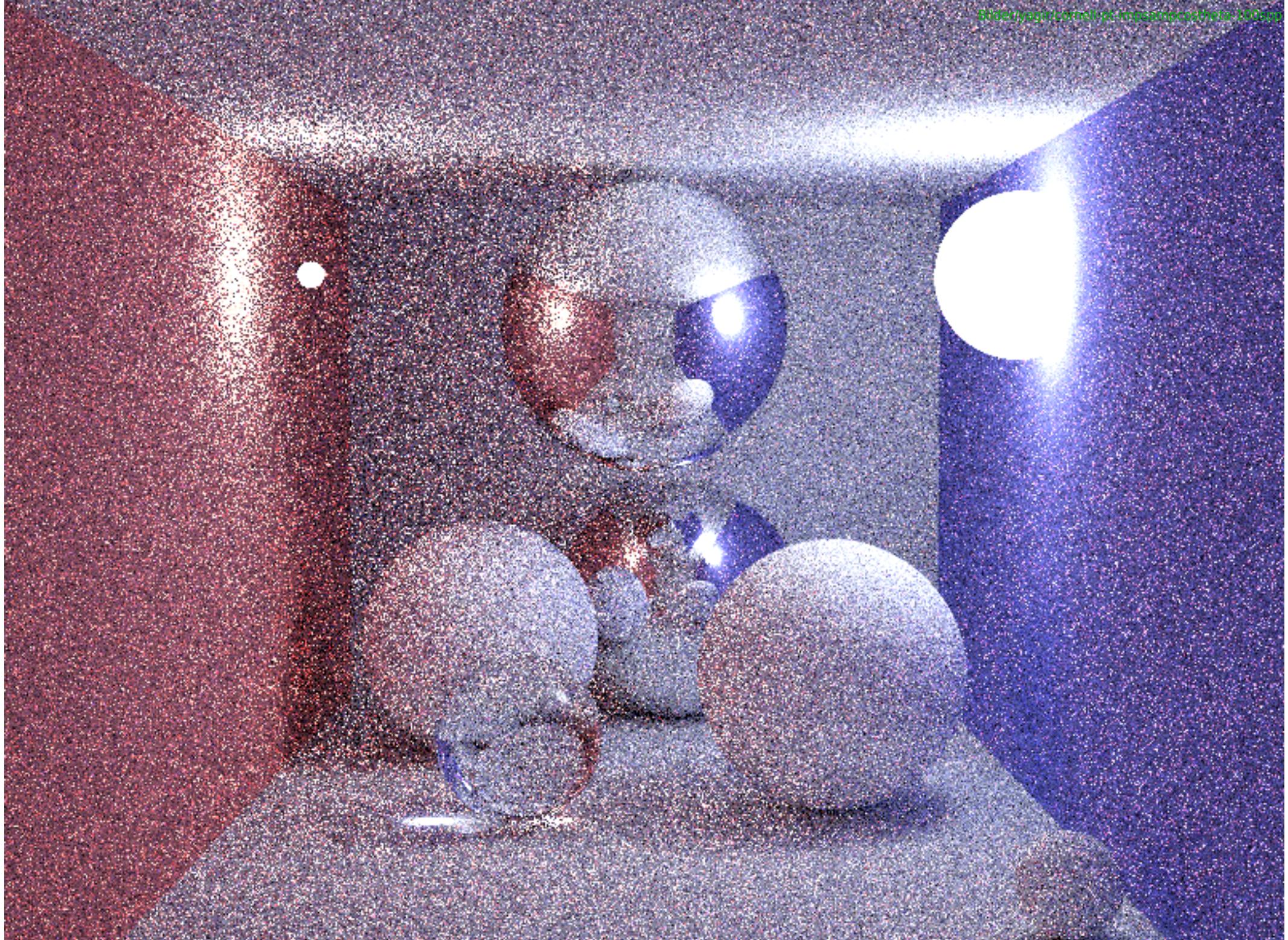
⇒ Aufteilung des Monte-Carlo-Prozesses in zwei Teile (direct lighting, next event estimation)

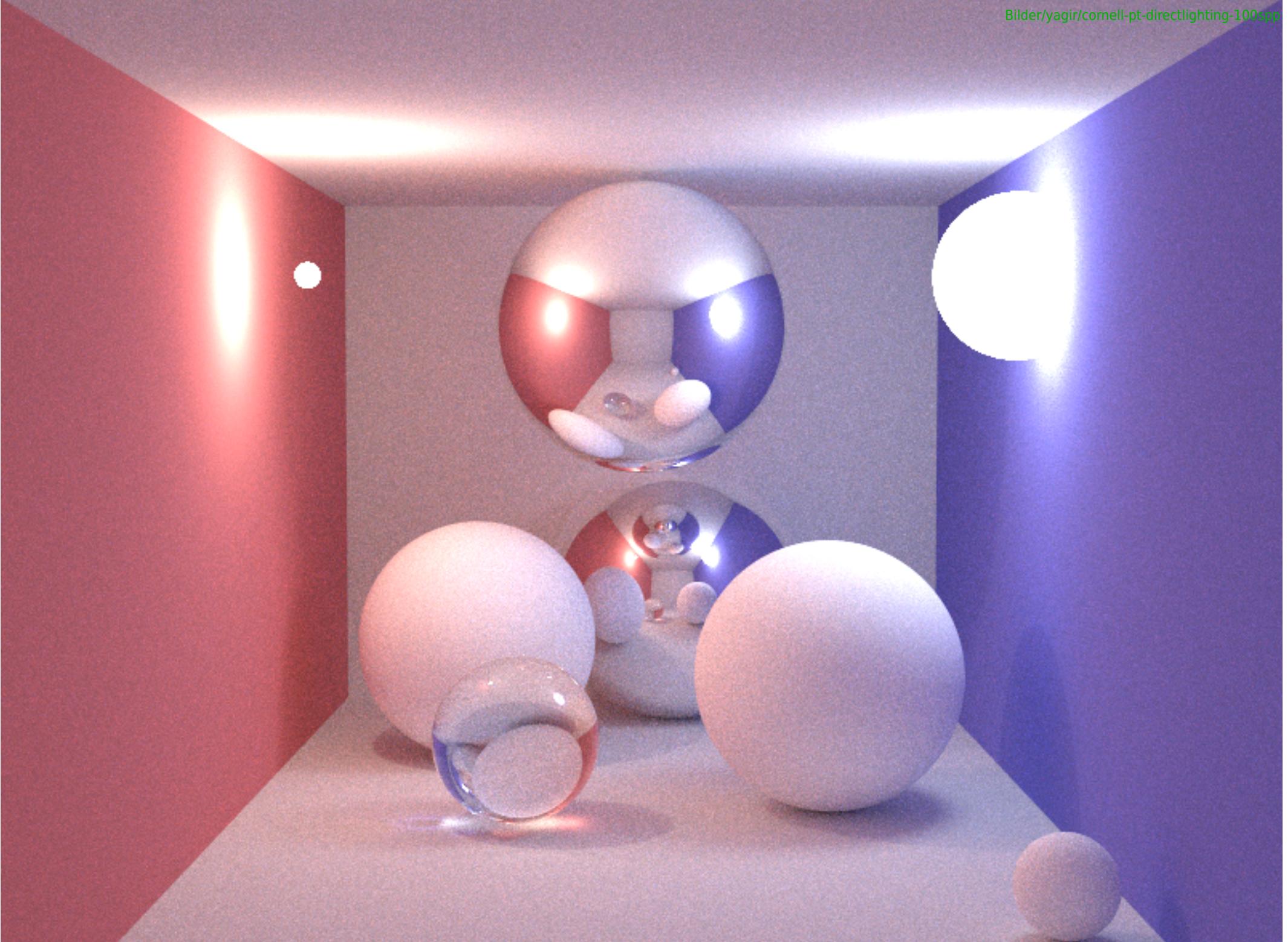
$$\begin{aligned} L(x \rightarrow \Theta) &= L_e + \int_{\Omega} f \cdot L(x \leftarrow \Psi) \cdot (N^T \cdot \Psi) d\omega \\ &= L_e + \int_{\hat{\Omega}} f \cdot L_e(x' \rightarrow -\Psi) \cdot (N^T \cdot \Psi) d\omega + \int_{\Omega \setminus \hat{\Omega}} f \cdot L(x \leftarrow \Psi) \cdot (N^T \cdot \Psi) d\omega \end{aligned}$$

also indirekter Anteil ohne Emission









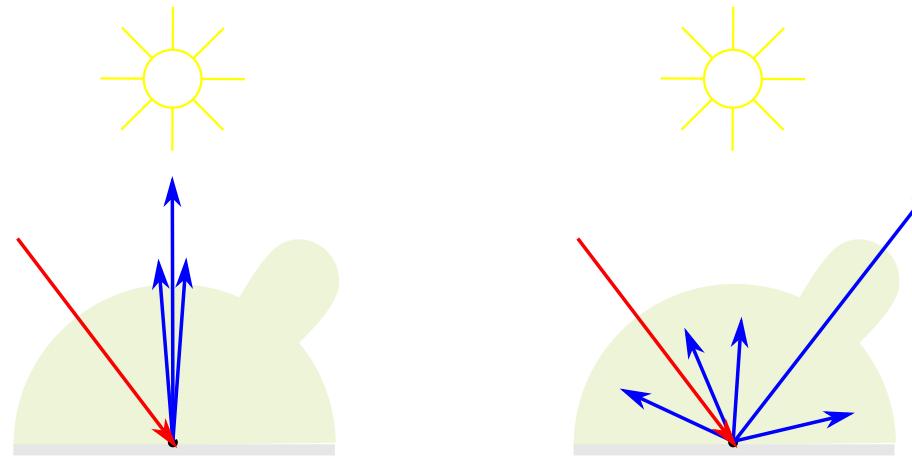
10.3.11 Multiple Importance Sampling

(nach Veach, 1997)

Eric Veach
Informatiker

Beobachtung 1: Wahl der Richtungen für direkte Beleuchtung durch Abtasten der Lichtquellen gut bei diffusem Material und kleinem Raumwinkel zur Lichtquelle

Beobachtung 2: Wahl der Richtungen für direkte Beleuchtung durch BRDF gut bei glänzendem Material und großem Raumwinkel zur Lichtquelle



Idee: kombiniere beides

Frage: Wie lassen sich beide berechneten Ergebnisse kombinieren?

Multi-Sample-Modell:

$$I \approx \sum_{i=1}^n \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(x_{i,j}) \frac{f(x_{i,j})}{p_i(x_{i,j})}$$

n Anzahl verschiedener Sampling-Strategien p_i

n_i Anzahl der Samples für Strategie i

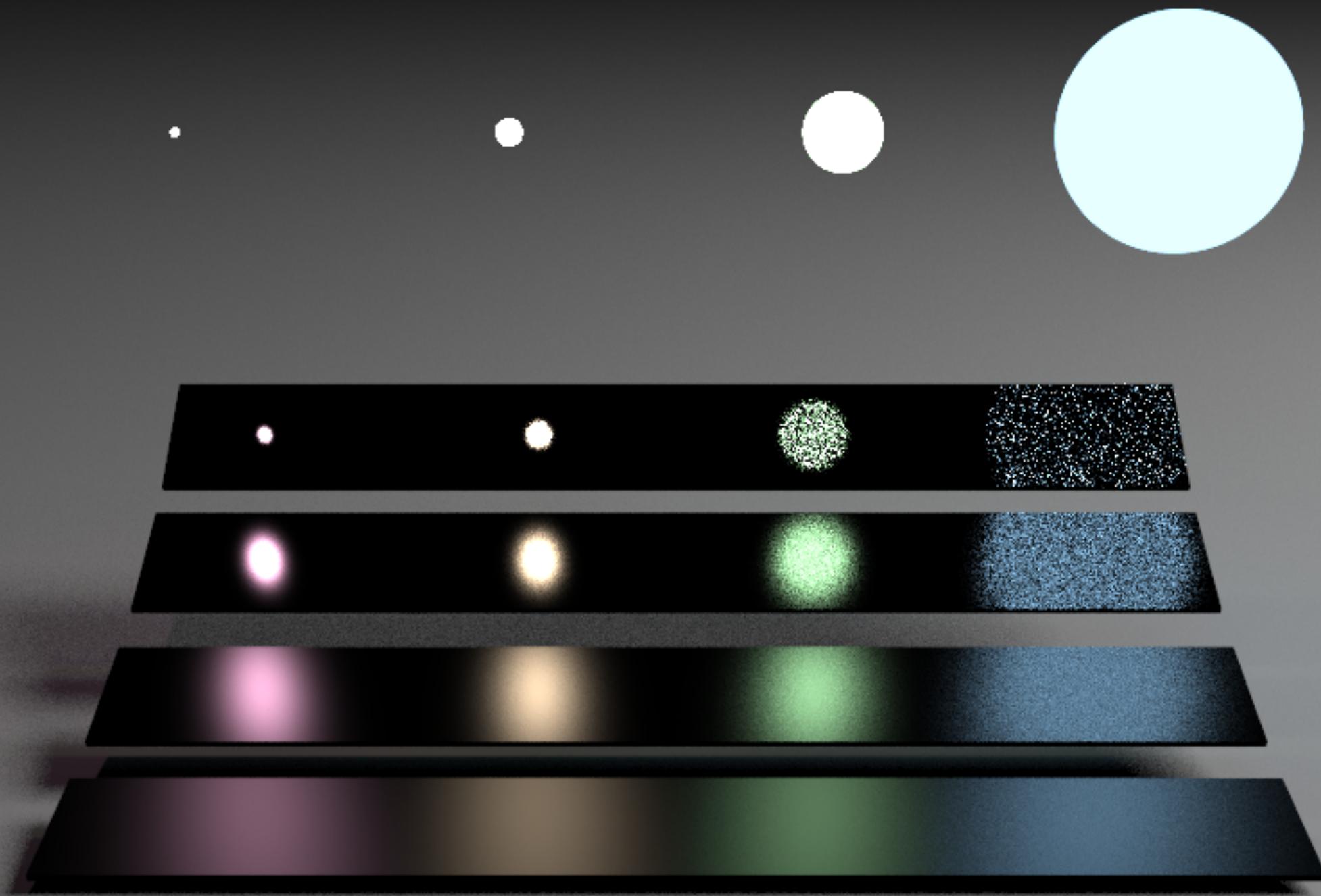
w_i Gewichtsfunktion für Samples der Strategie i

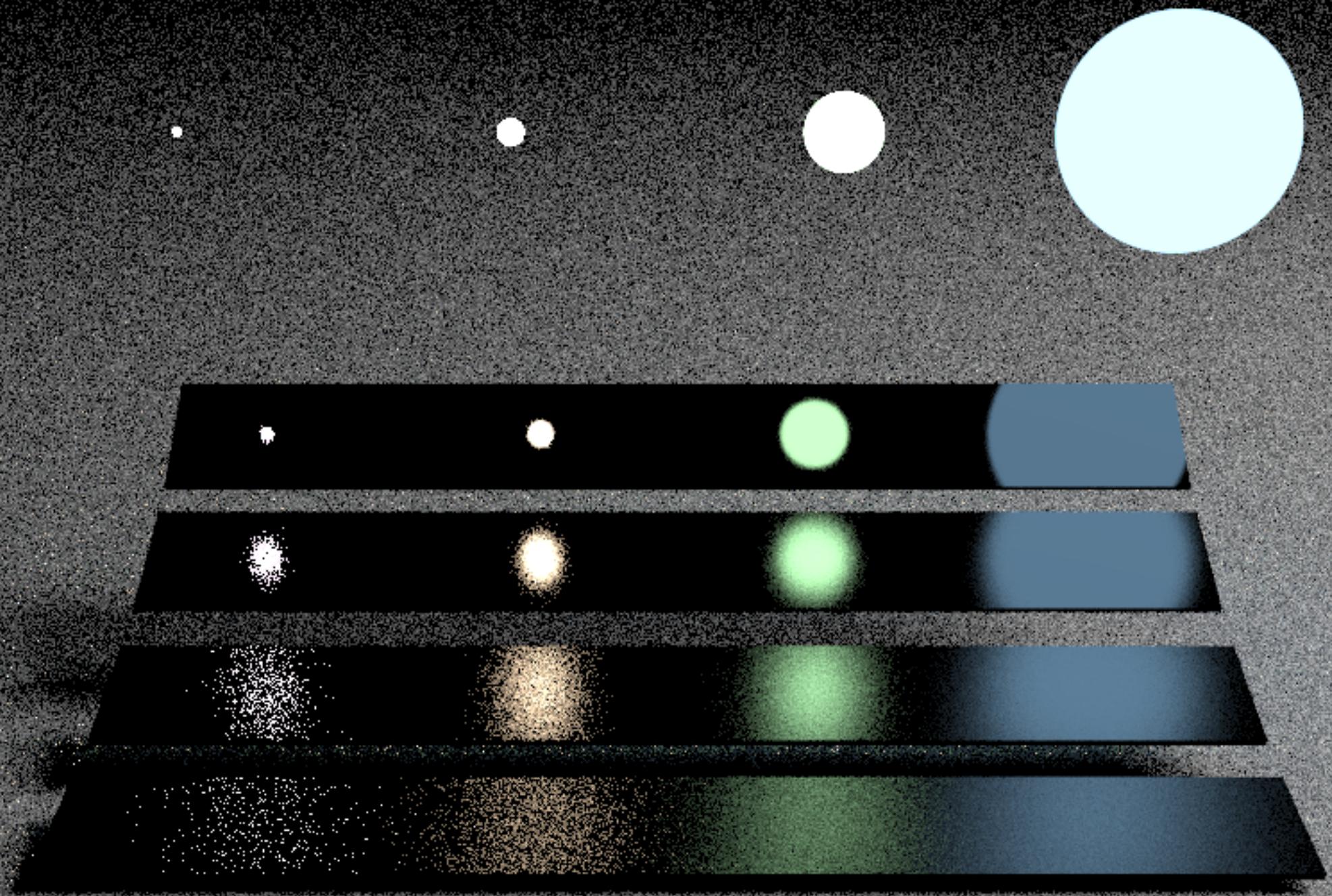
Wahl der w_i z. B. per Balance-Heuristik:

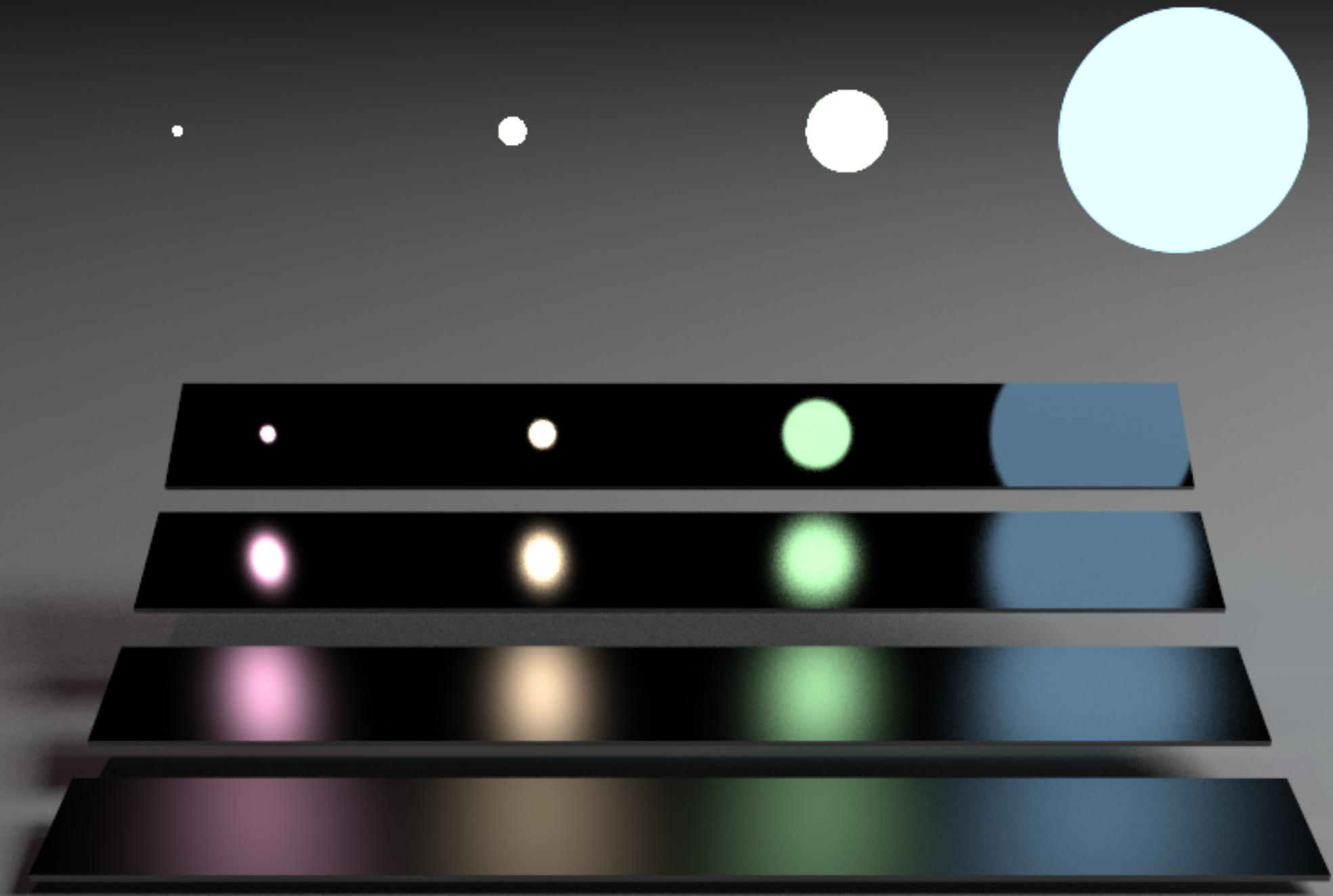
$$w_i(x) = \frac{n_i p_i(x)}{\sum_k n_k p_k(x)}$$

Single-Sample-Modell: Wähle Strategie p_i mit Wahrscheinlichkeit c_i .

$$I \approx \frac{w_i(x_i) f(x_i)}{c_i p_i(x_i)}$$







10.3.12 Weiterentwicklungen

Eric P. F. Lafontaine
Informatiker

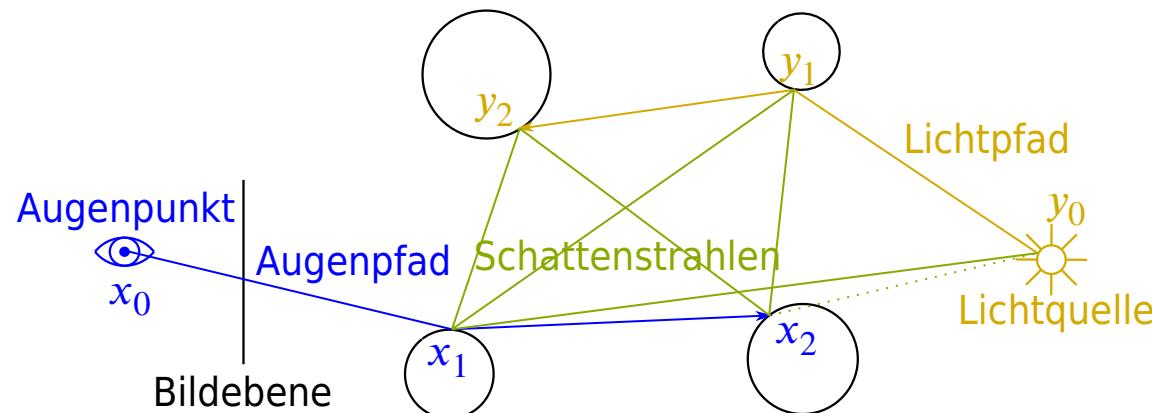
Bidirektionales Path Tracing

(Lafontaine/Willems 1993, Veach/Guibas 1994)

Yves D. Willems
Informatiker

- Konstruiere jeweils zwei Pfade jeweils fester Länge:
 - einen ausgehend vom Kamerapunkt
 - einen ausgehend von einer Lichtquelle
- Verbinde Zwischenpunkte der Pfade zu vollständigen Pfaden Lichtquelle – Kamerapunkt
- ⊕ bessere Darstellung von Kaustiken (durch Lichtbündelung entstehende helle Bereiche)

Leonidas John Guibas (Λεωνίδας Ιωάννης Γκύμπας [Leónidas Iōánnēs Gkímpas])
* 1949
Informatiker
Quelle: https://commons.wikimedia.org/wiki/File:Leonidas_Guibas_2010_06_29.png
Lizenz: CC BY-SA 3.0, <https://creativecommons.org/licenses/by-sa/3.0/legalcode>



hier: gewichteter Mittelwert der Strahlen $(x_0, x_1, y_0), (x_0, x_1, x_2, y_0)$ (Anteil 0 wegen Verdeckung),
 $(x_0, x_1, y_1, y_0), (x_0, x_1, x_2, y_1, y_0), (x_0, x_1, y_2, y_1, y_0), (x_0, x_1, x_2, y_2, y_1, y_0)$



Metropolis Light Transport

(Veach/Guibas 1997)

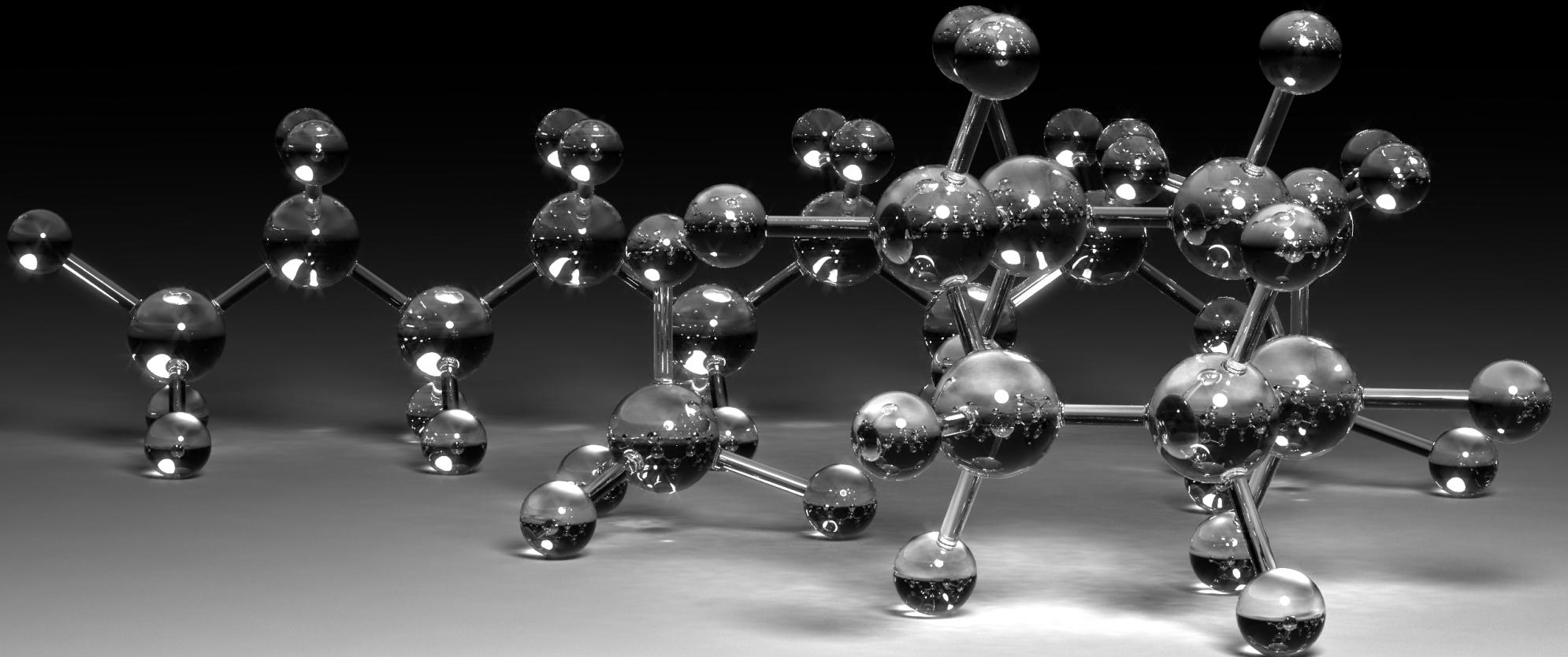
- Erweiterung des bidirektionalen Path Tracings
- Ist ein „guter“ Pfad gefunden worden (hoher Beitrag zum Bild), betrachte zusätzliche Pfade, die sich durch kleine Änderungen ergeben.
 - Hinzufügen eines zusätzlichen Punktes zum Pfad
 - kleine Verschiebungen der Knotenpunkte des Pfads

Photon Mapping

(Jensen 1995)

Henrik Wann Jensen
* 1969, Harlev, DK
Computergraphiker

- zweistufiges Verfahren
- 1. Schritt: Aussenden von Lichtpaketen („Photonen“) von den Lichtquellen, Speicherung des Ergebnisses in einer Photon Map (unabhängig vom Betrachtungspunkt)
für Kaustiken zusätzliche Speicherung einer „Caustic Map“
- 2. Schritt: Rendern z. B. durch Kombination von diffusem Raytracing für direkte Beleuchtung und Information aus Photon Map für indirekte Beleuchtung
auch in Verbindung mit Path Tracing zur Beschleunigung
- Verfahren ist „biased“ (keine Konvergenz gegen Lösung der Rendergleichung durch Mittelwert vieler gerenderter Bilder) aber Konvergenz mit zunehmender Anzahl an Photonen





Index

A

- adaptive Tiefenkontrolle bei Raytracing 10-29
- aktive Strecke 3-34
- Algorithmus von de Casteljau 9-23
- alignment layer 2-17
- ambientes Licht 7-16
- Augenposition 4-15
- Augenpunkt 7-16
- Außenseite 6-32

B

- baryzentrische Koordinaten 7-41
- Basismatrix 9-5
- Basispolynom 9-8
- Beleuchtungsmodell 7-29
 - lokales 7-29
- Bernsteinpolynom 9-10, 9-13
- Bézier-Fläche 9-29
- Bézier-Kurve 9-9, 9-13

- bidirektionales Path Tracing 10-94
- Bildwiederholfrequenz 2-4
- Bildwiederholspeicher 2-10
 - logischer 3-2
- Binary Space Partitioning 10-22
- bösartige Szene
 - Zell-Raster-Technik 6-19
- Bounding Box 6-16
- Braunsche Röhre 2-3
- BRDF 10-85
- b-rep (boundary representation) 8-11
- BSDF 10-87
- BSP-Baum 10-22
- B-Spline 9-14
- BTDF 10-87
- Bubble Jet 2-22
- Bump Map 7-57
- Bump Mapping 7-56

C

- choice 2-24
Clipping
 Pipelining 5-30
Colour Bleeding bei diffuser Reflexion 10-35
colour lookup table 2-14
CRT 2-3
Culling 6-32

D

- Dämpfungs faktor 7-22
de Casteljau, Algorithmus 9-23
Delta-Delta-Röhre 2-11
Delta-Formfaktor 10-54
diffuse Reflexion 7-17
 Colour Bleeding 10-35
 Farbverschiebung 10-35
diffuses Raytracing 10-82
Doppelpufferung 3-3
Drahtmodell-Darstellung 6-2
Druckkopf
 Nadeldrucker 2-21

Tintenstrahldrucker 2-22

E

- Echo 2-36
Ecke eines Polygons 3-28
einfaches Polygon 3-28
Eingabe
 event driven 2-31
 Sampling 2-30

Eingabegerät

- logisches 2-24
Element bei Radiosity 10-68
Event 2-31
event driven 2-31

F

- Farb-Raster-Bildschirm 2-11
Farb-Raster-Display 2-13
 mit Farbtabelle 2-14
Farbtabelle 2-14
Farbtiefe 2-13, 2-15
Färbungsstrategie 7-33
Farbverschiebung durch diffuse Reflexion 10-35

Fixierung	2-22
Flachbett-Plotter	2-6
Flat Mapping	7-53
Fluoreszenz	2-4
Flüssigkristall-Bildschirm	2-16
Formfaktor	10-41, 10-43
Objektpuffer	10-59
Zelle	10-53

G

Gather bei Radiosity	10-63
Geometriematrix	9-5
Gewichtsfunktion	
bei Intensitätskomponente	7-7
Glanzlicht	7-20
gutartige Szene	
Zell-Raster-Technik	6-18

H

Hermite-Basis-Polynom	9-8
Hermite-Kurve	9-6
hidden line removal	6-2
hidden surface removal	6-2

hierarchische Modellierung	4-26
Highlight	7-42
HLS-Doppelkegel	7-13
HLS-Modell	7-12
HSV-Kegel	7-12
HSV-Modell	7-12
hue	7-12

I

Icon	2-40
image	2-25
Importance Sampling	10-90
Inkrement	3-11
Inline-Röhre	2-12
Inneres eines Polygons	3-28
In-Schnittpunkt	5-18
Intensitätsverteilung	
kontinuierliche	7-3
Interaktion	2-2
interlaced	2-9
Interpolation	
mit Bézier-Kurven	9-11

K

- Kante 3-27
kapazitativer Touchscreen 2-29
Kathodenstrahl-Röhre 2-3
Knotenpunkt (B-Spline) 9-14
Knotenwert (B-Spline) 9-14
kontinuierliche Intensitätsverteilung 7-3
Kontur 6-35
Koordinaten
 baryzentrische 7-41
kubischer B-Spline 9-14
Kurve
 parametrisierte kubische 9-2
Kurvensegment (B-Spline) 9-14
- L**
- Laserdrucker 2-22
LCD 2-16
Lichtpuffer bei Raytracing 10-26
lightness 7-12
liquid crystal display 2-16
locator 2-25

- logischer Bildwiederholspeicher 3-2
logisches Eingabegerät 2-24
lokales Beleuchtungsmodell 7-29

M

- Mach-Band-Effekt 7-37
Maske 3-46
Masken-Transparenz 7-65
Maus
 mechanische 2-26
 optische 2-27
mechanische Maus 2-26
Mengenoperation
 regularisierte 8-8
Metropolis Light Transport 10-95
Modell der Ordnung > 0 10-3
Modell der Ordnung 0 7-29
Modellierung
 hierarchische 4-26
Monte-Carlo-Integration 10-81
Multiple Importance Sampling 10-92

N

- Nadeldrucker
 Druckkopf 2-21
nicht-lokales Modell 10-3
nicht-uniformer kubischer B-Splines 9-21
non-interlaced 2-9
normalisierte Sichtkoordinaten 5-38
NURBS 9-22

O

- Objektpuffer bei Raytracing 10-31
Objektpuffer zur Formfaktorberechnung 10-59
Octree 10-16
offline 1-8
OpenGL 4-3, 4-33
optische Maus 2-27
Out-Schnittpunkt 5-18

P

- parametrisierte bikubische Fläche 9-27
parametrisierte kubische Kurve 9-2
Parametrisierung eines Objekts 7-53
Partikelsystem 8-24

- Patch 8-3
Patch bei Radiosity 10-42
Patch (Bézier) 9-29
Path Tracing 10-84
 bidirektionales 10-94
Phosphoreszenz 2-4
Photon Mapping 10-96
Pipelining
 beim Clipping 5-30
Pixel 3-2
 virtuelles 3-48
Pixel Map 3-2
Pixmap 3-2
 virtuelle 3-48
Plasma-Bildschirm 2-19
Plotter
 Flachbett 2-6
 Trommel 2-6
Polyeder 6-32
Polygon 3-28
 Ecke 3-28
 einfaches 3-28

Inneres	3-28
Polygonzug	
geschlossener einfacher	3-27
streng monotoner	6-8
Pop-up-Menü	2-40
Profil	2-39
Projektionsfenster	4-20
projektiver Raum	4-21
Punkt	
unendlich ferner	4-22
Punktlichtquelle	7-21
Q	
Quadtree	10-18
R	
Radiance	10-74
Radiosity	10-39
Element	10-68
Gather	10-63
Patch	10-42
Shoot	10-64
Radiosity Matrix	10-43

Raster-Bildschirm	2-8
Raster-Display	2-10
rationale kubische Kurve	9-22
Raum	
projektiver	4-21
Raumwinkel	10-37
Raytracing	
adaptive Tiefenkontrolle	10-29
diffuses	10-82
Lichtpuffer	10-26
Objektpuffer	10-31
Reflexion	
diffuse	7-17
refresh buffer	2-10
regularisierte Mengenoperation	8-8
Rendergleichung	10-75
resistiver Touchscreen	2-28
RGB	2-11
RGB-Modell	7-10
Röhre	
Braunsche	2-3
Kathodenstrahl-	2-3

Rotationskörper 9-30
S
Sampling 2-30
saturation 7-12
Scan Line 3-29
Schrittmotor 2-7
Screen-Door Transparency 7-65
selbstähnliche Form 8-16
Shoot bei Radiosity 10-64
Sichtkoordinaten
 normalisierte 5-38
Sichtvolumen 5-32
Solid Angle 10-37
Span 3-29
sr, Steradian 10-37
Startup-Zeit 5-31
Steradian 10-37
stochastisches Raytracing 10-82
Strahldichte 10-74
Strecke
 aktive 3-34

streng monotoner Polygonzug 6-8
string 2-24
Sweep 8-9
T
Tangentialvektor (Kurve) 9-3
Texel 7-53
Texture Map 7-53
Tintenstrahldrucker
 Druckkopf 2-22
Touchscreen
 kapazitativer 2-29
 resistiver 2-28
Trommelplotter 2-6
U
undo 2-39
unendlich ferner Punkt 4-22
uniformer B-Spline 9-14
Unterteilung von Bézier-Kurven 9-24
V
valuator 2-25

value	7-12
Vector Display	2-5
virtuelle Pixmap	3-48
virtuelles Pixel	3-48
visible surface ray tracing	6-4
Volume Mapping	7-62
Vulkan	4-3

W

Weltkoordinaten	4-18
wire frame	1-8, 6-2

Z

Zeichensatz	3-54
Zelle	6-14
Zelle bei Formfaktorberechnung	10-53
Zell-Raster-Technik	6-13
bösartige Szene	6-19
gutartige Szene	6-18
Zielpunkt	4-15
z-Puffer	6-20