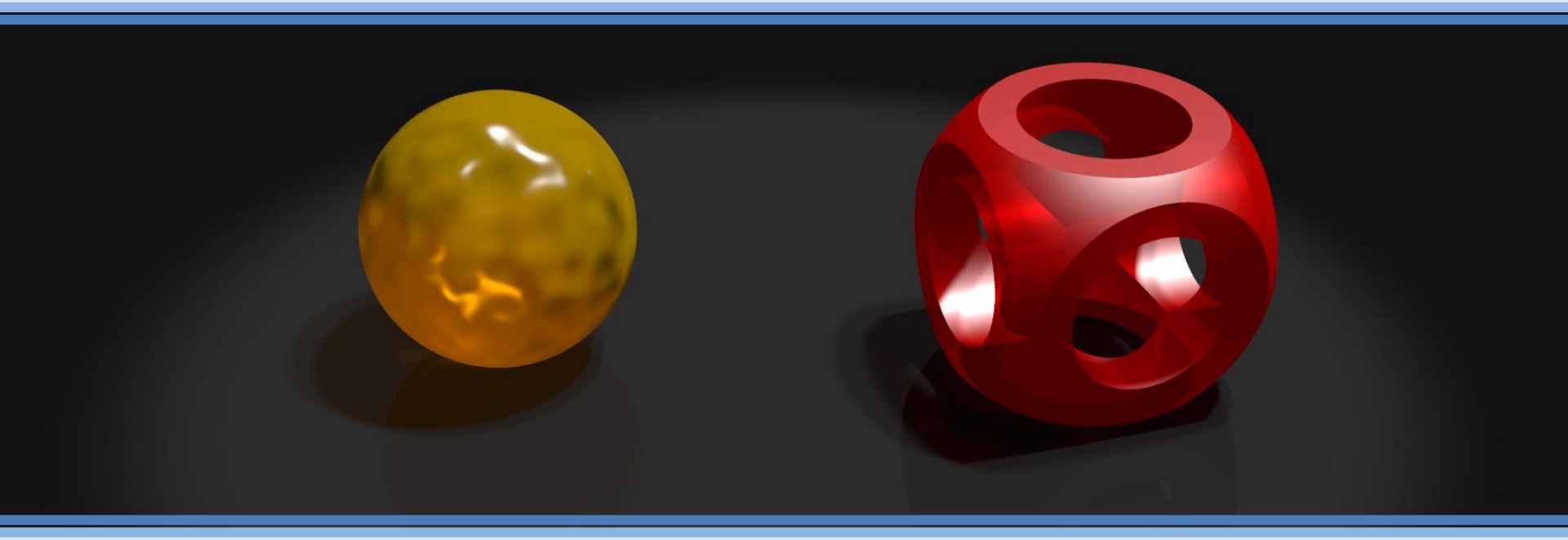
Computergrafik

T. Hopp



Organisatorisches

- 33 Vorlesungsstunden
- 9 Vorlesungstermine: jeweils Mittwochnachmittags
  - Ab 10.10.2018, jeweils 12:30 – 15:45 Uhr

Organisatorisches

- Kontakt: torsten.hopp@kit.edu

- Vorlesungsunterlagen:
  - Folien werden auf der eLearning-Plattform (https://moodle.dhbw.de/course/view.php?id=2000) jeweils nach der Vorlesung bereitgestellt.

  Raum: „**TINF16 Grafik WiSe**“

  - Selbsteinschreibung mit Einschreibeschlüssel: **blinnPhong**

- Wie kann ich Sie kontaktieren?

Organisatorisches

- Organisation der Vorlesung
  - Vorlesung zu ca. 9-10 Themenbereichen
  - Themen allgemein in der Computergrafik angesiedelt, Beispielanwendungen teilweise etwas medizinlastig
  - Begleitende Übungen
    - Beispielaufgaben (z.B. Algorithmen, Berechnungen)
    - Begleitend zur Vorlesung schauen wir uns Programmierübungen an.
      - Wir nutzen OpenGL
      - Grundkenntnisse in C-Programmierung?
      - Wie sieht es mit der Computerinfrastruktur aus?
      - Bitte richten Sie sich eine Programmierumgebung ein um die Übungen nachvollziehen zu können.
  - Klausur: 60 min. Vorläufiger Termin in der Klausurwoche
    - Wissensfragen zur Vorlesung (siehe z.B. Übungsfragen am Ende des Kapitels)
    - Aufgaben wie in den Übungen

Literaturhinweise

- A. Nischwitz, M. Fischer, P. Haberäcker, G. Socher: „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik, Vieweg und Teubner Verlag, 3. Auflage 2011
- J.F. Hughes, A. van Dam, M. McGuire, D.F. Sklar, J.D. Foley, S.K. Feiner, K. Akeley: „Computer Graphics. Principles and Practice“, 3rd edition, Addison-Wesley, 2014
- J. Vince: „Mathematics for Computer Graphics“, 4th edition, Springer, 2014
- [K. Zeppenfeld: „Lehrbuch der Grafikprogrammierung“, 1. Auflage, Spektrum Akademischer Verlag, 2004]
- [F. Klawonn: „Grundkurs Computergrafik mit Java“, 3. Auflage, Vieweg und Teubner, 2010]

1.1. EINLEITUNG

Definition – worum geht es?

- Wikipedia: „Die Computergrafik ist ein Teilgebiet der Informatik, das sich mit der computergestützten Erzeugung, im weiten Sinne auch mit der Bearbeitung, von Bildern befasst.“
- Foley: „Computer graphics is the science and art of communicating visually via a computer‘s display and its interaction devices“.

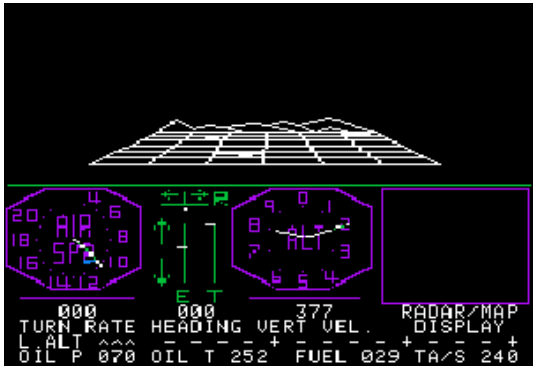
- Grafische Datenverarbeitung
  - **Generative Computergrafik:** Visualisierung komplexer Zusammenhänge, Animationen, Virtual Reality etc.
  - **Bildverarbeitung:** Bildfilterung, Segmentierung, Bildrestauration etc.
  - **Bildanalyse:** Automatische Extraktion von Informationen aus Bildern, z.B. Mustererkennung

Anwendungsbeispiele

- Generative Computergrafik: breites Anwendungsspektrum in zahlreichen Disziplinen
  - Simulation
  - Computer Aided Design
  - Visualisierung
  - Unterhaltung
  - …
- Nachfolgend eine Auswahl typischer Anwendungsgebiete und -beispiele

Ausbildungssimulation

- Triebfeder der modernen Computergrafik, z.B. Flugsimulation



Flight Simulator 1 für Apple II (1979)

Flugtraining-Simulator, Rockwell Collins

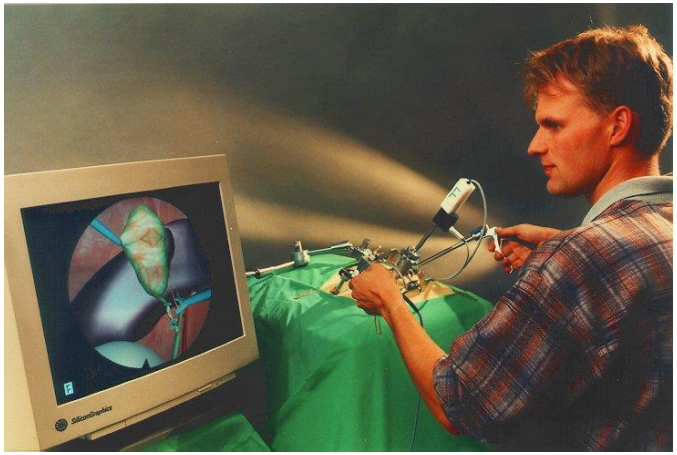
Full Motion Flight Simulator,
Lufthansa Flight Training

http://en.wikipedia.org/wiki/History_of_Microsoft_Flight_Simulator
https://www.rockwellcollins.com/Data/Products/Simulation/Operator_Training_Systems/Full_Flight_Simulator.aspx
http://de.wikipedia.org/wiki/Flugsimulation

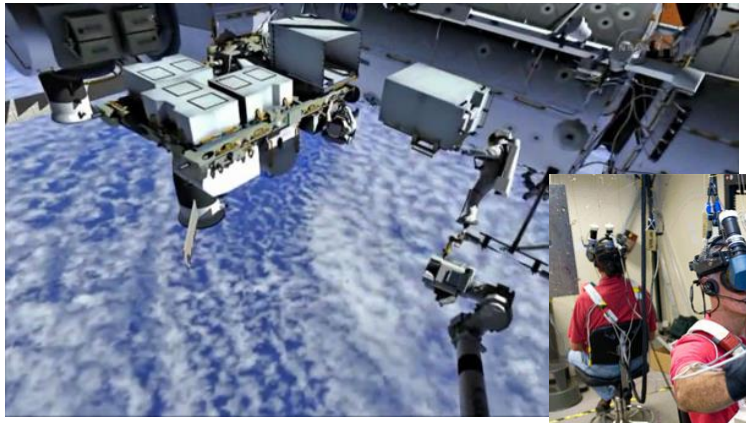
Ausbildungssimulation

Straßenbahnsimulator, Berliner Verkehrsbetriebe

Fluglotsenausbildung, Wien



Karlsruhe Endoskopie Trainingssystem, KIT (1997)

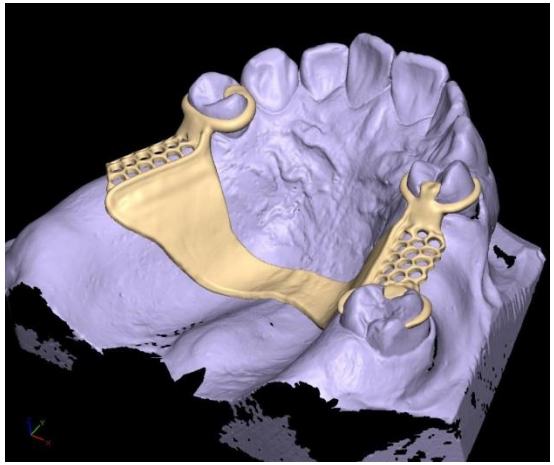


Space Walk Simulator, NASA

https://www.iai.kit.edu/www-extern/index.php?id=2033
http://www.rheinmetall-defence.com/de/rheinmetall_defence/public_relations/news/archive_2013/aktuellesdetailansicht_2753.php
http://www.wien.gv.at/verkehr-stadtentwicklung/fluglotse.html
http://www.nasa.gov/centers/johnson/engineering/robotics_simulation/virtual_reality/index.html
http://www.cbsnews.com/network/news/space/home/spacenews/files/e93930fcf48b6cbdf795f3f81efae7b7-177.html

CAD/CAM

- CAD/CAM = Computer Aided Design/Manufacturing
- Rapid bzw. Virtual Prototyping
- Z.B. Design von Autos, virtuelle Crashtests, Cockpit-Design, …
- Z.B. Patientenspezifisches Design von Prothesen etc.



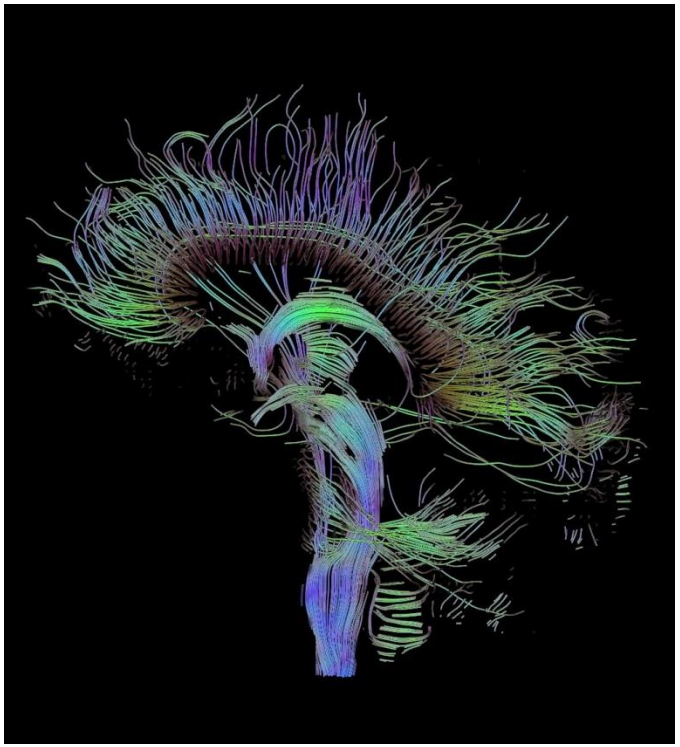
http://www.directindustry.com/prod/esi-group/crash-test-simulation-software-8972-131175.html
http://www.dentsable.com/industries-medical-dental.htm

CAD/CAM

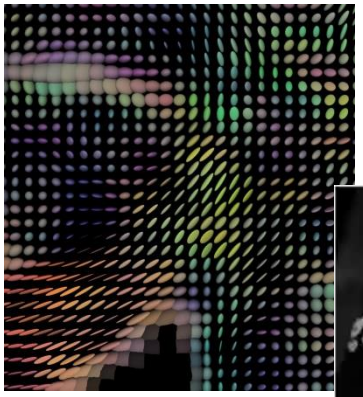
- Z.B. beim Entwurf in der Architektur/Stadtplanung

http://www.pro-eleven.de/

Visualisierung in der Medizin



Visualisierung von Nervenbahnen im Gehirn mit Diffusion-Tensor-MRT

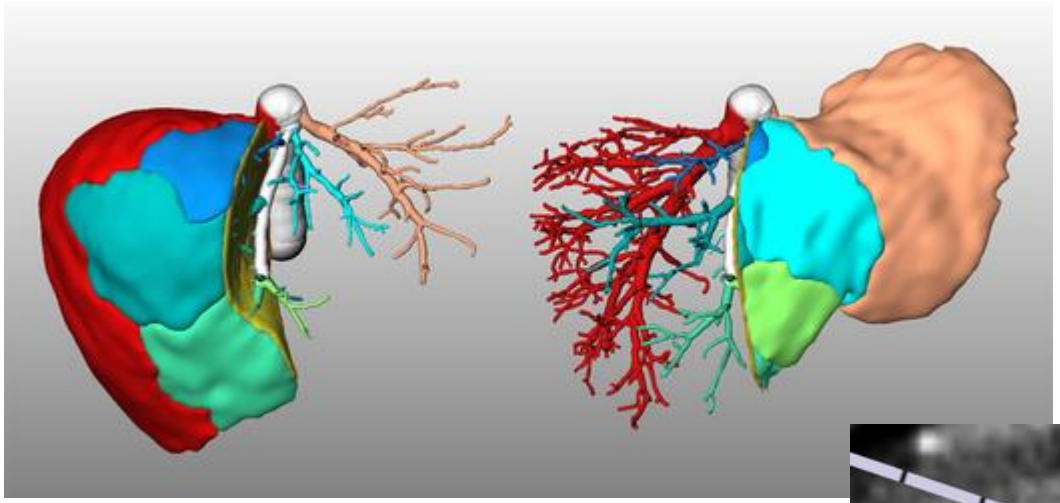


Abstrahierung mit Diffusions-Ellipsen

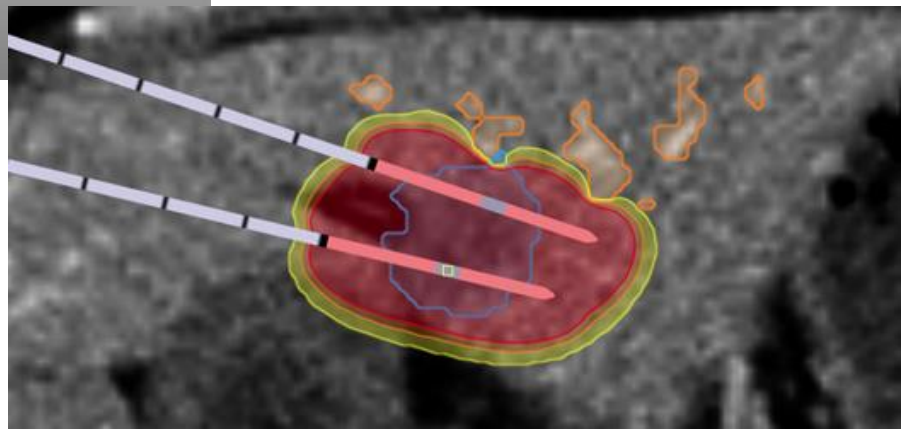
Blutfluss in der Aorta

http://de.wikipedia.org/wiki/Diffusions-Tensor-Bildgebung
http://www.uniklinik-freiburg.de/neurologie/forschung/neurologische-arbeitsgruppen/neurovaskulaere-bildgebung.html

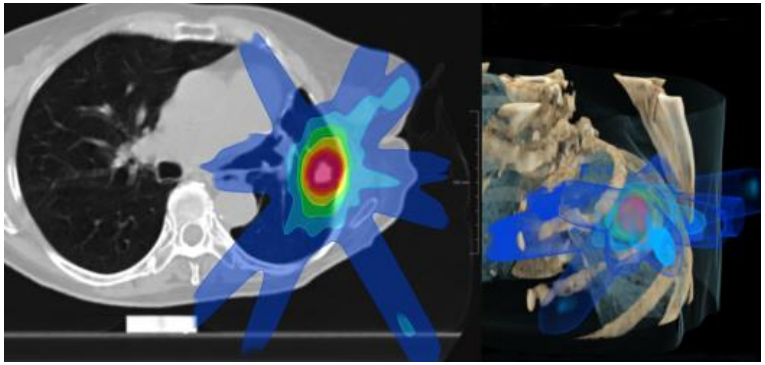
Visualisierung in der Medizin



Optimale Leberpartitionen bei Leberlebendspende, Fraunhofer MEVIS



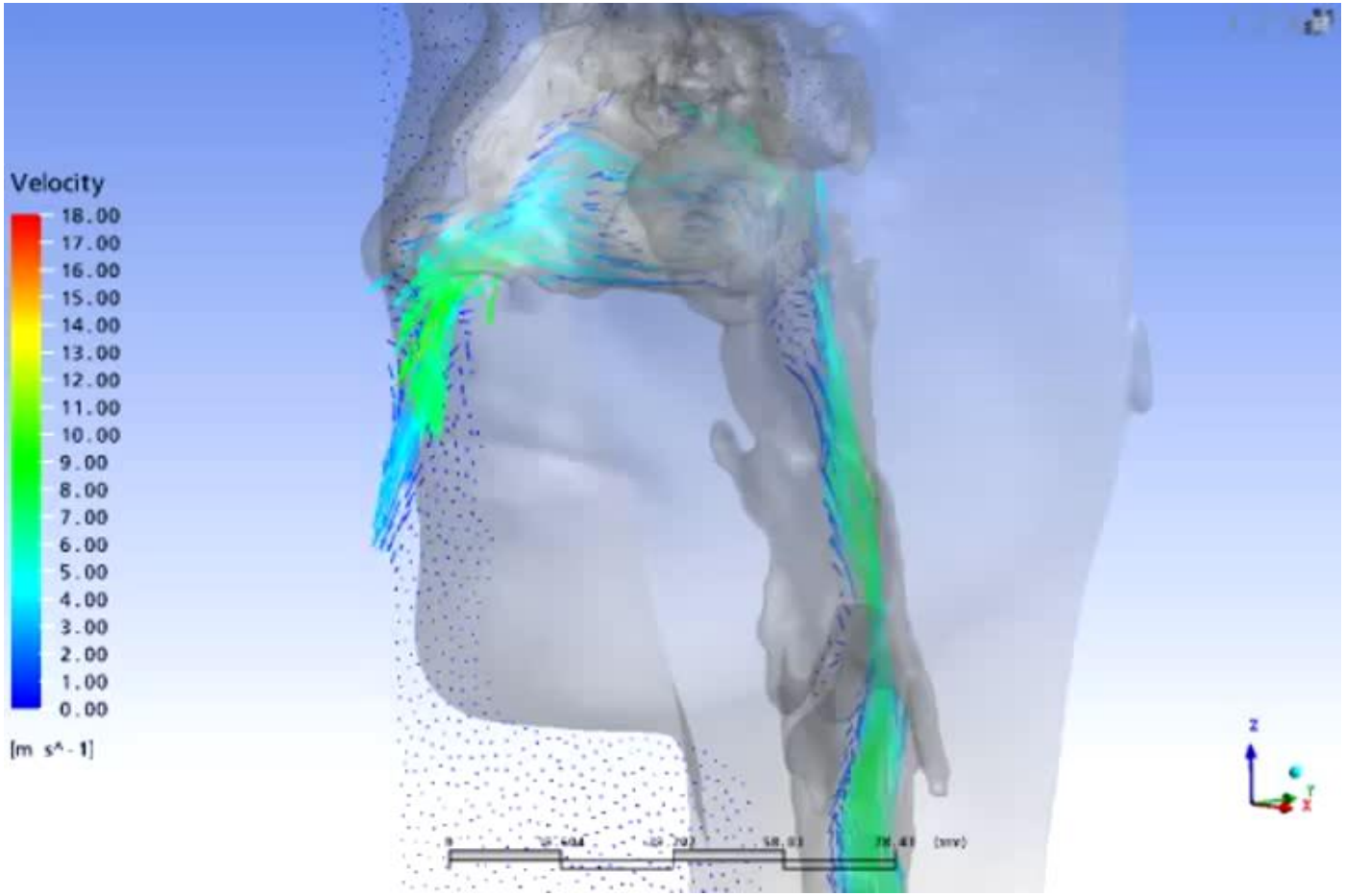
Tumorablations-Planung- und Visualisierung, Fraunhofer MEVIS



Planung von Tumorbestrahlung, Fraunhofer MEVIS

http://www.mevis.fraunhofer.de/klinische-kompetenzen/leber.htm

Visualisierung in der Medizin

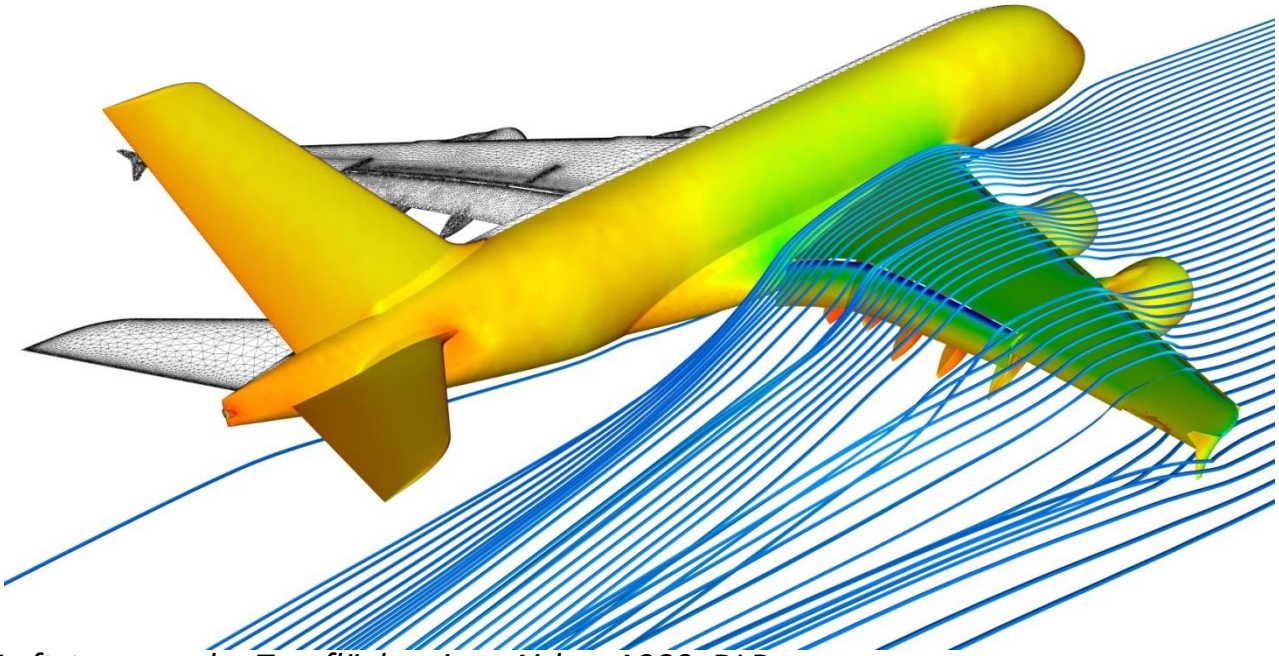


Strömungssimulation der Atmung

https://www.youtube.com/watch?v=1wHTm9JrkFI

Visualisierung

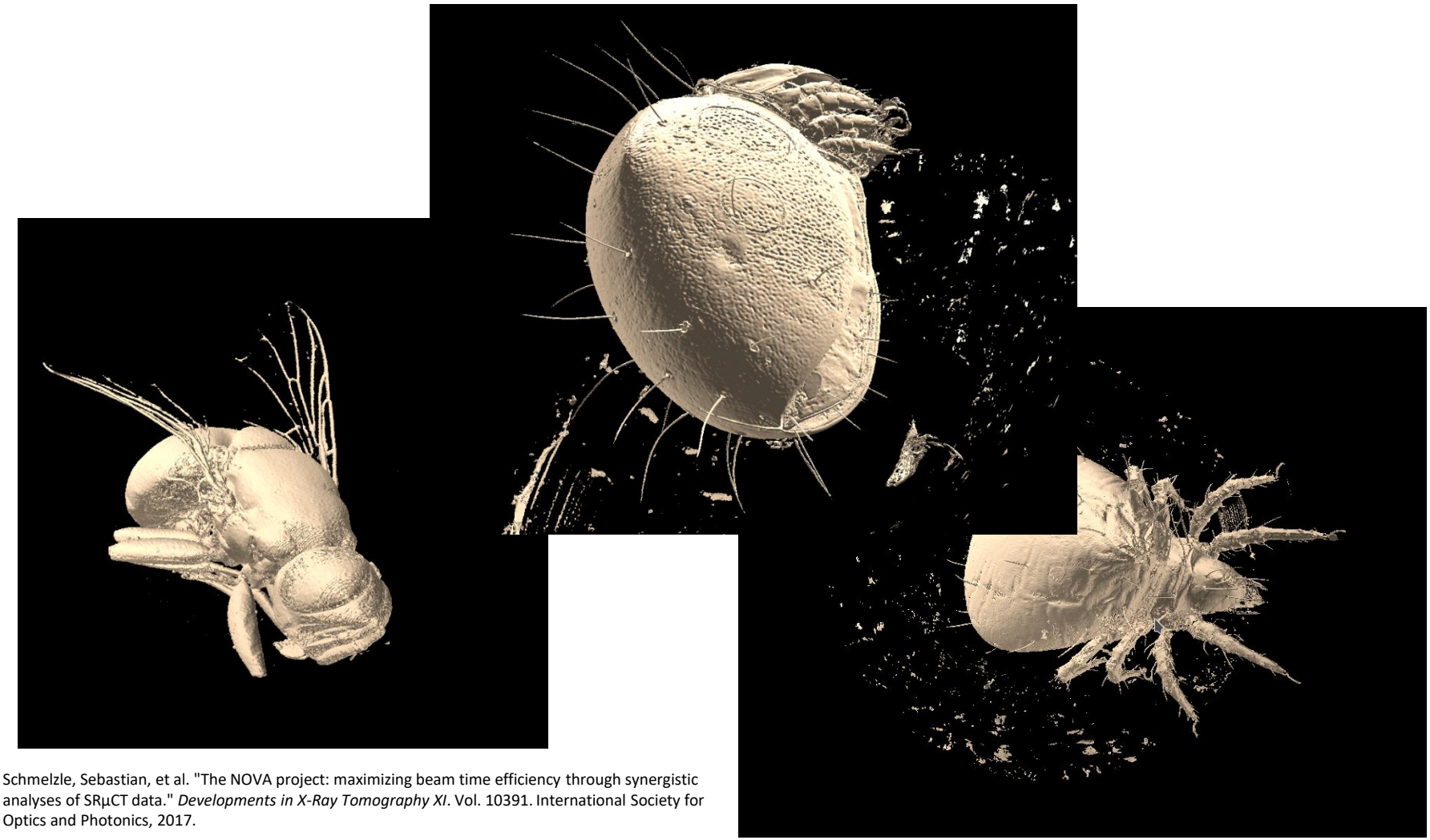
- Darstellung von gemessenen (räumlich bzw. zeitlich aufgelösten) Daten oder Simulationsergebnissen
- Z.B. Strömungssimulation



Luftstrom an der Tragfläche eines Airbus A380, DLR

http://www.dlr.de/media/desktopdefault.aspx/tabid-4985/8422_page-6//8422_read-13200

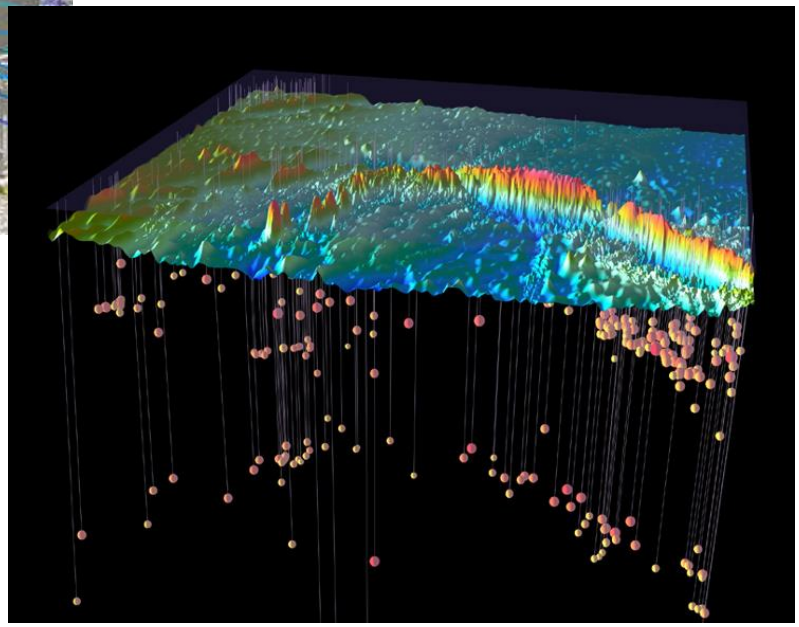
Visualisierung



- Schmelzle, Sebastian, et al. "The NOVA project: maximizing beam time efficiency through synergistic analyses of SRμCT data." *Developments in X-Ray Tomography XI*. Vol. 10391. International Society for Optics and Photonics, 2017.

Visualisierung

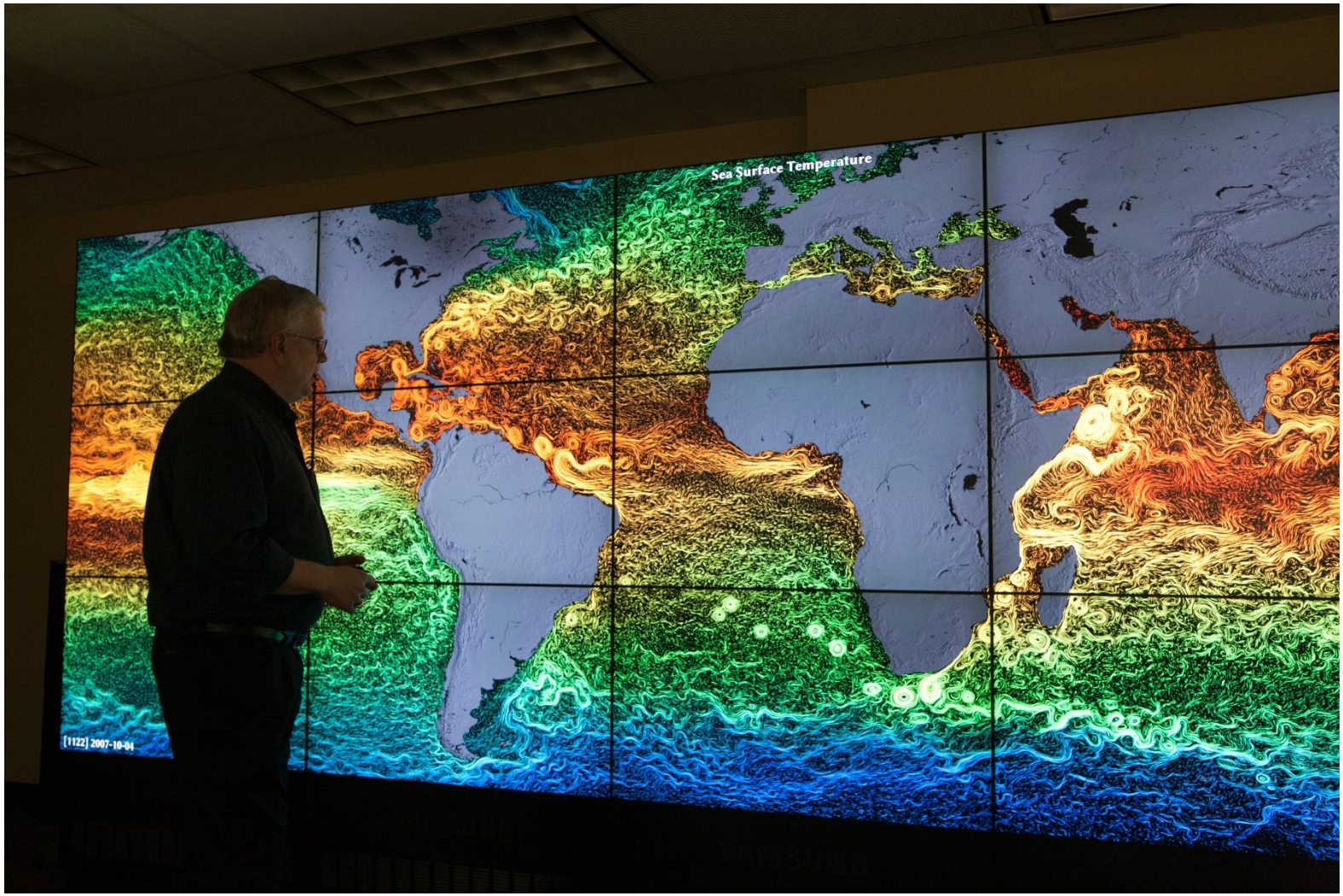
Visualisierung einer Wind-Simulation im Karlsruher 3D Stadtmodell, KIT



Macquarie Fault Zone, Meeresboden-Höhenprofil mit Epizentren, ACES Visualiztion Laboratory

http://www.hitechmex.org/Viz.html
http://www.math.kit.edu/ianm4/seite/visualization/de

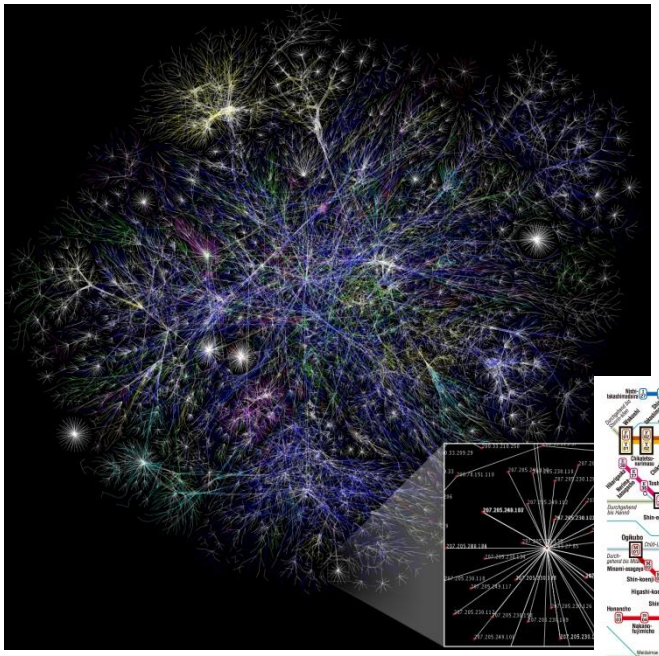
Visualisierung



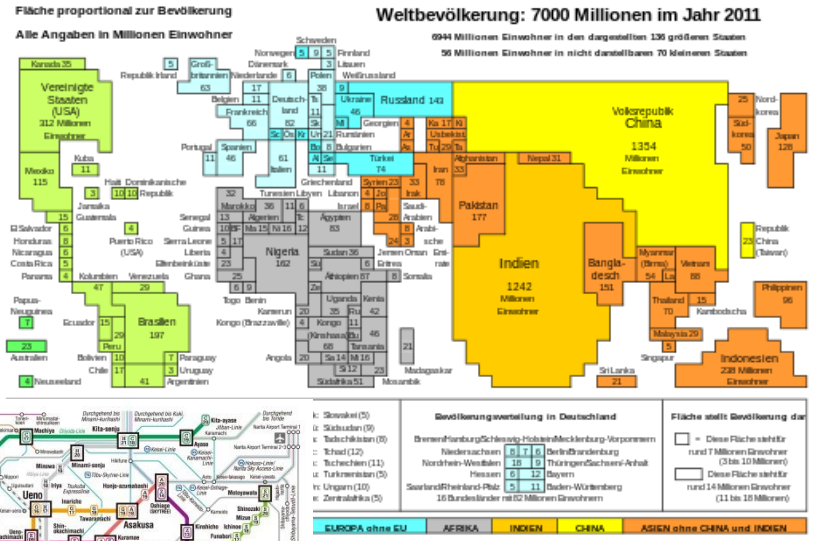
http://blogs.agu.org/wildwildscience/2013/03/18/nasas-science-visualization-wall-cool-is-an-understatement/

Informationsvisualisierung

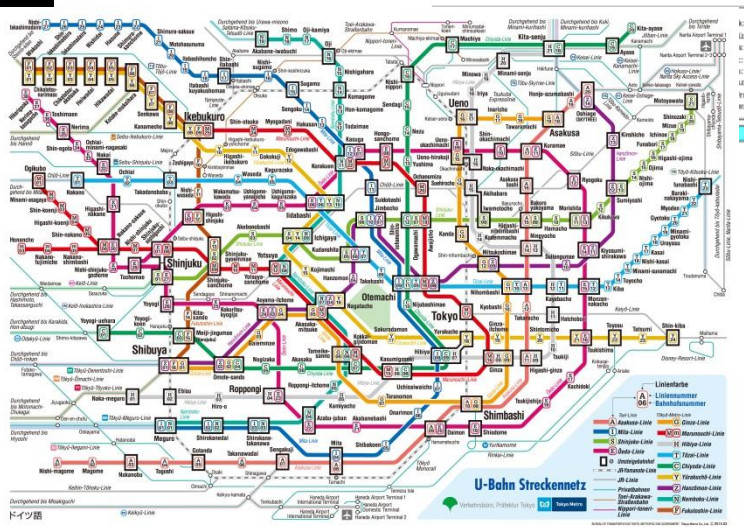
- Intuitive Darstellung von Informationen



Visualisierung des Internets



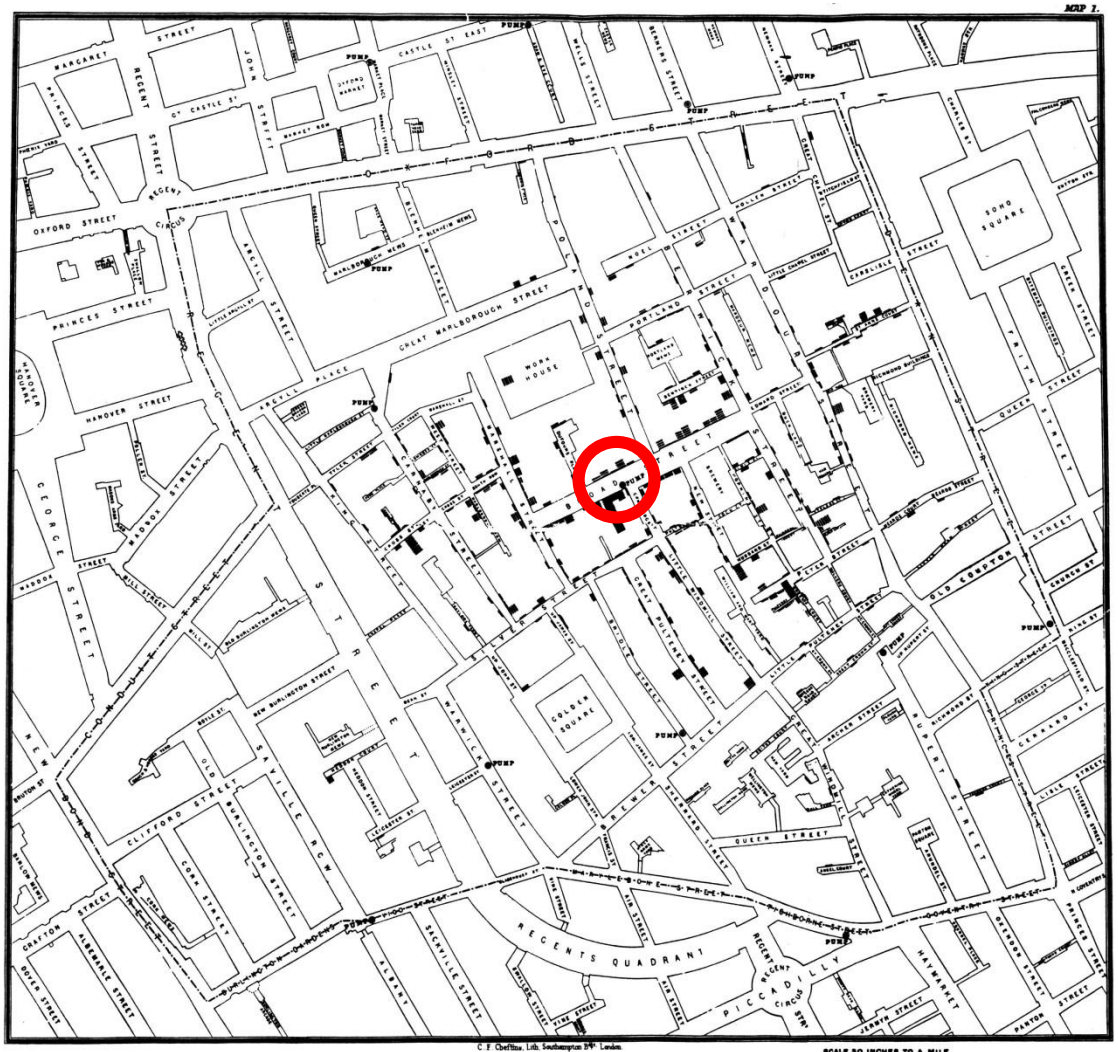
Darstellung der Weltbevölkerung 2011



U-Bahn Plan Tokio

http://www.tokyometro.jp/en/subwaymap/pdf/routemap_de.pdf
http://de.wikipedia.org/wiki/Weltbevölkerung
http://de.wikipedia.org/wiki/Internet

Informationsvisualisierung in der Medizin

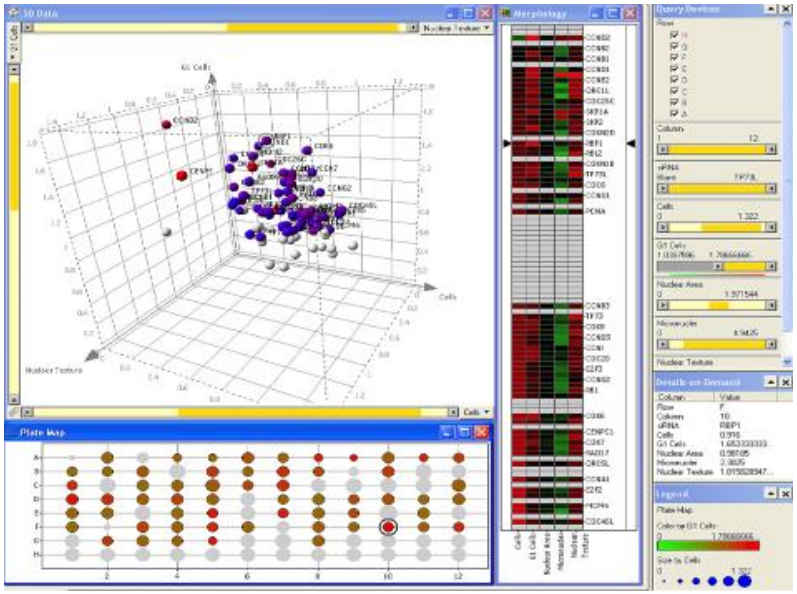


1854: Cholera map (John Snow)

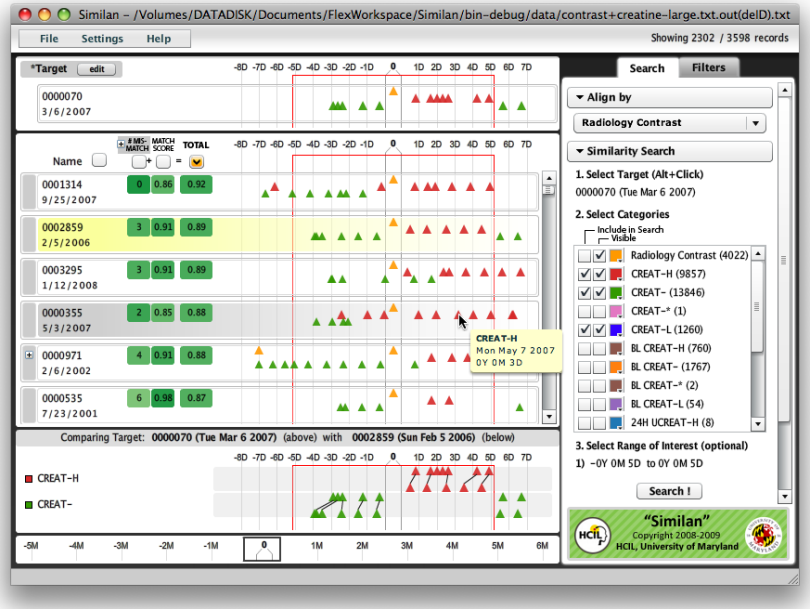
https://en.wikipedia.org/wiki/1854_Broad_Street_cholera_outbreak

Informationsvisualisierung in der Medizin

- Z.B. Visualisierung von Zeitreihen, Visualisierung in der Epidemologie



Visualisierung mehrdimensionaler Daten

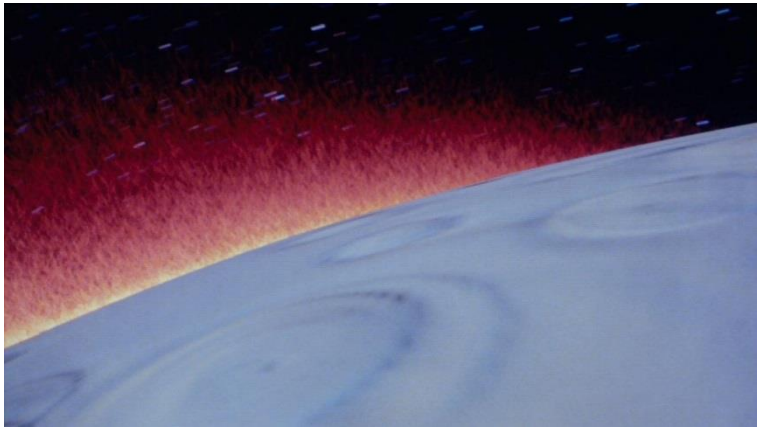


Visualisierung von Zeitreihendaten

http://uxmag.com/articles/beyond-the-medical-chart
http://www.cs.umd.edu/hcil/temporalviz/

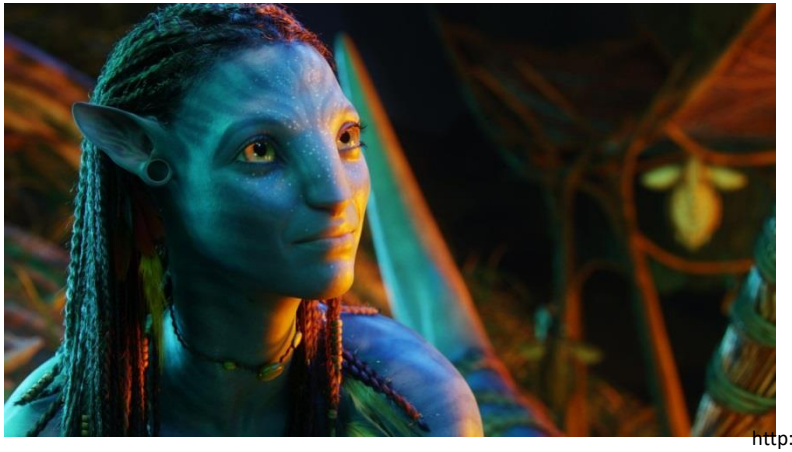
Unterhaltung: Filme

- Computer Generated Imagery (CGI) in Filmen



Star Trek II, erste CGI-Szene in einem Spielfilm, 1982

Planet der Affen: Revolution, 2014



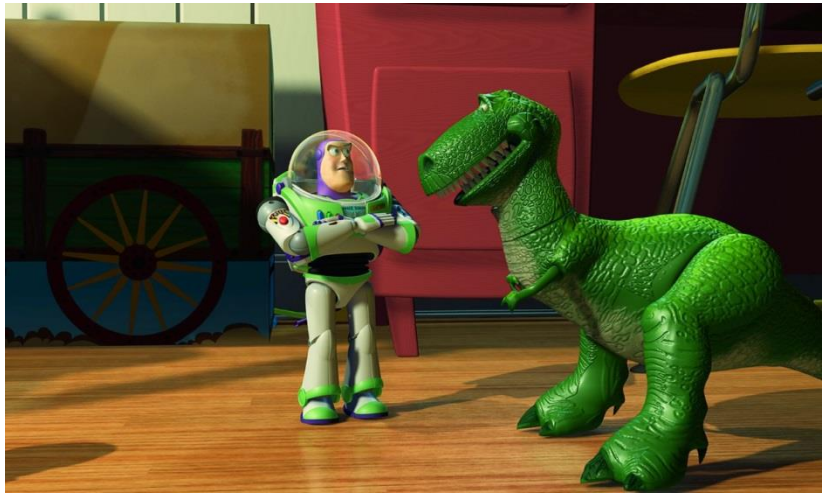
Avatar, 2009

Jungle Book, 2016

http://de.memory-alpha.org/wiki/CGI
http://www.imdb.com/title/tt0499549/
http://www.welt.de/wirtschaft/webwelt/article133262129/Geschichte-der-Hollywood-Bilder-aus-dem-Computer.html
https://www.inverse.com/article/14351-how-the-jungle-book-made-its-animals-look-so-real-with-groundbreaking-vfx

Unterhaltung: Filme

- Animationsfilme



Toy Story 1, Pixar, 1995. Erster vollständig am Computer animierter Spielfilm

Ice Age, 2002

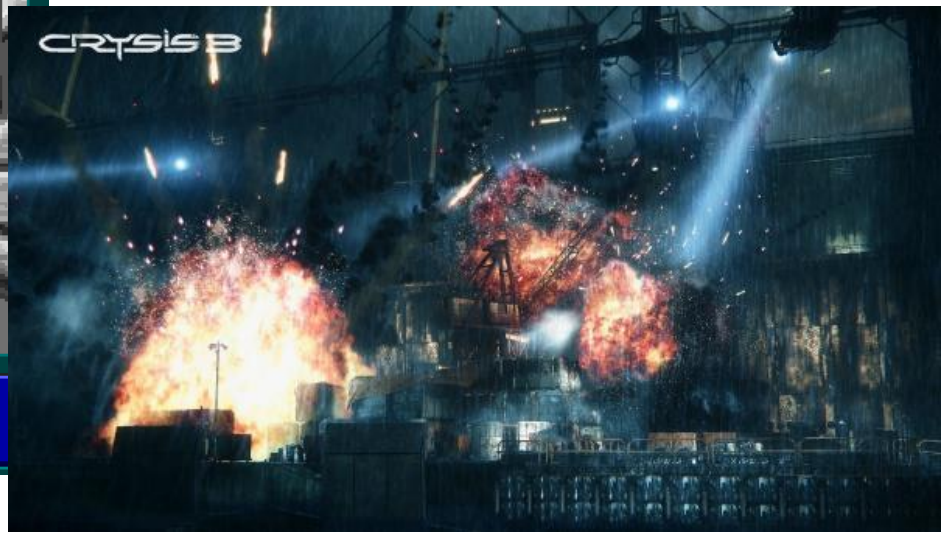
- Vorreiter Pixar unter Steve Jobbs als Geschäftsführer
  - Pixars RenderMan Industriestandard für Rendern von CGI/3D-Animation

- http://www.imdb.com/title/tt0114709/
- http://www.imdb.com/title/tt0268380/

Unterhaltung: Computerspiele

- Heute meist basierend auf einer 3D-Grafik-Engine zum Rendern der Szenen (z.B. CryEngine, Frostbite)
  - geometrische Objektbeschreibung, Oberflächentexturen, Licht und Schatten (Shading), Transparenz, Spiegelungen, physikalische Effekte usw.

Wolfenstein 3D, 1992



Crysis 3, 2013

- http://www.mobygames.com/game/wolfenstein-3d/screenshots
- http://www.gamestar.de/spiele/crysis-3/bilder/screenshots/crysis_3,46254,94722.html

Unterhaltung: Computerspiele

- Simulationsspiele sind professionellen Simulatoren in der 3D-Grafik mittlerweile fast gleichwertig

X-Plane 10

Gran Turismo 6

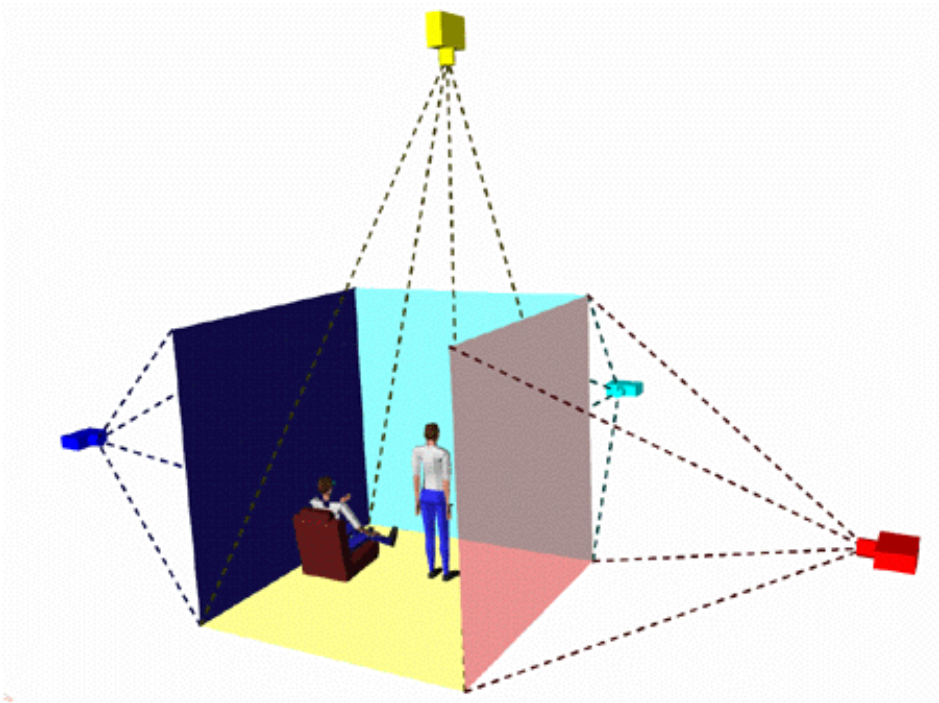
- http://www.x-plane.com/desktop/multimedia/
- http://www.computerbild.de/fotos/Gran-Turismo-6-Das-wuenscht-sich-die-Redaktion-8293248.html

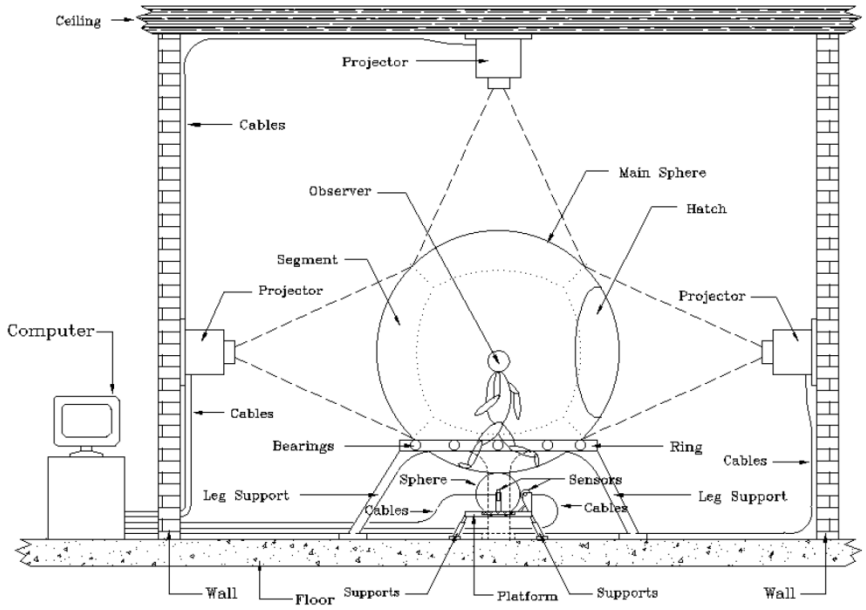
Virtual Reality

- **Darstellung** + Wahrnehmung vollständig computergenerierter virtueller Umgebungen
- **Interaktion** über Eingabegeräte
  - Z.B. Datenhandschuh/Controller, 3D Maus, optisches Tracking
- **Rückkopplung** durch Force Feedback
- Herausforderungen für ein „realistisches Erlebnis“:
  - Darstellung eines großen Sichtfeldes
  - Hohe Bildauflösung
  - Schnelle Reaktionszeit des Displays < 3ms
  - Hohe Bildwiederholfrequenz
  - Hardware zur Darstellung, bei Head Mounted Display z.B. optische Herausforderungen
  - Genaues Tracking der Bewegungen der Person in der Szene
  - Geringe Latenzzeit zwischen Eingabe und Darstellung des Bildes (<25 ms)

CAVE

- = Cave Automatic Virtual Environment
- Stereoprojektion auf umgebende Wände eines Raumes

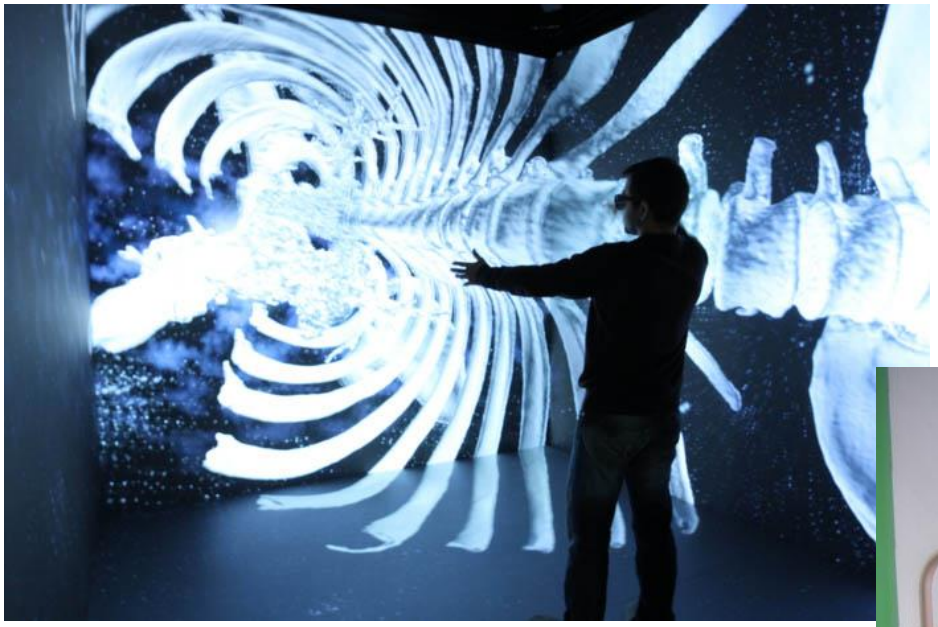




Cybersphere

- http://escience.anu.edu.au/lecture/cg/Display/cave.en.html
- K.J. Fernandes, V. Raja, J. Eyre: „Cybersphere: The Fully Immersive Spherical Projection System“, Communication of the ACM 46, 2003, 141-147

CAVE: Beispiele



Wanderung durch den Körper, Steuerung per Gesten

Planung des Einbaus von Kabinenelementen in ein Flugzeug

- http://phys.org/news/2013-04-high-performance-cave-automatic-virtual-environment.html
- http://www.lufthansa-technik.com/de/virtual-cabin-visit

Head mounted display

- Darstellung der Szene…
  - … auf augennahem Bildschirm
  - … oder auf die Netzhaut projiziert
- Enthält Sensoren zur Bewegungserfassung
- Heute gängige Spezifikation
  - Sichtfeld: ~90°, teilweise bis 180°
  - Auflösung: 1280x1024 und höher
  - Gewicht: <1kg

Virtuelles Fallschirmspringen als Training

Anwendung in der Chirurgie

http://de.wikipedia.org/wiki/Head-Mounted_Display
http://www.innovationtoronto.com/2013/08/sonys-head-mounted-3d-video-display-gives-surgeons-an-inside-view/

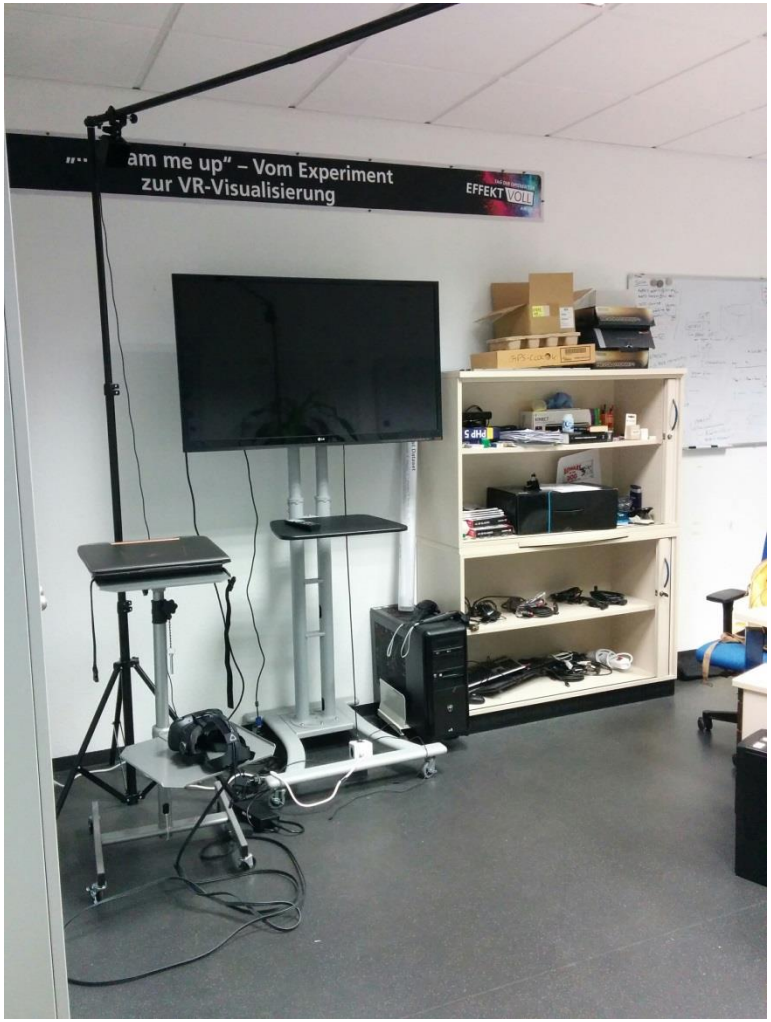
Head mounted display

- „Revival“ durch Kombination mit Smartphones

https://www.youtube.com/watch?v=-4vhtpa8JRA

VR Lab am KIT/IPE

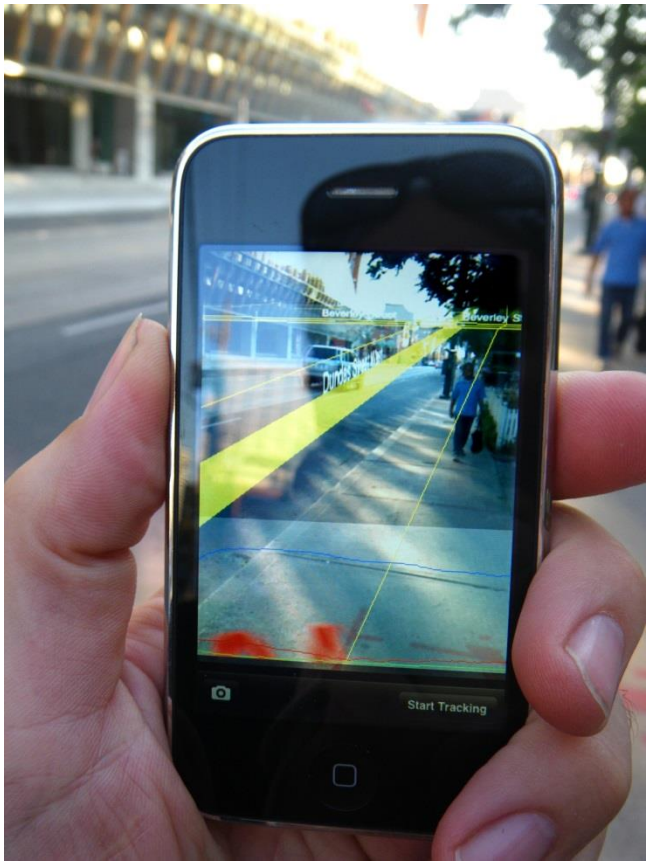
- HTC Vive
- ROG Laptop (GTX 1070)



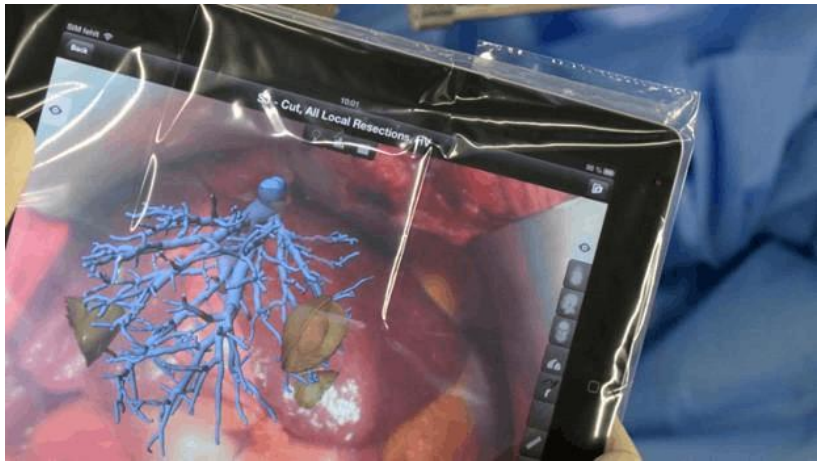
Augmented Reality

- Computergestützte **Erweiterung** der Realitätswahrnehmung
  - ≠ Virtual Reality!
- Kombination von realen Bildern und Computergrafik
  - Oft in Echtzeit
- Herausforderungen:
  - Echtzeitanforderung
  - Nachführung bei Bewegung
  - Geometrische Information über reale Szene notwendig

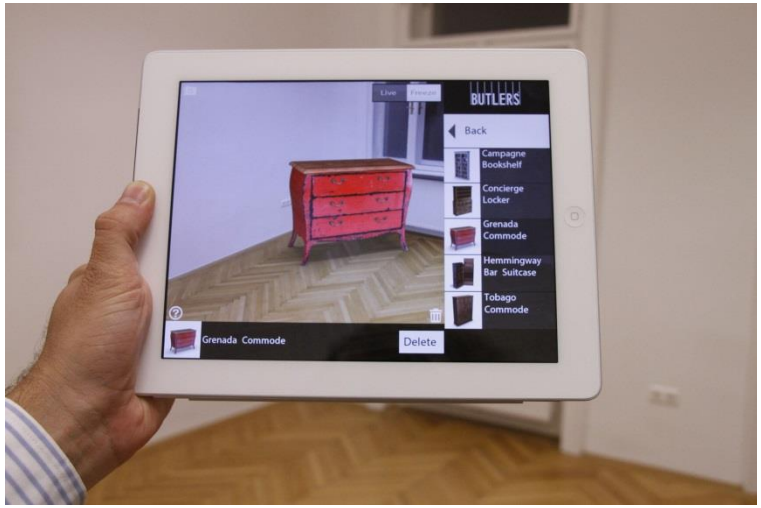
Augmented Reality: Beispiele



Navigation am Smartphone



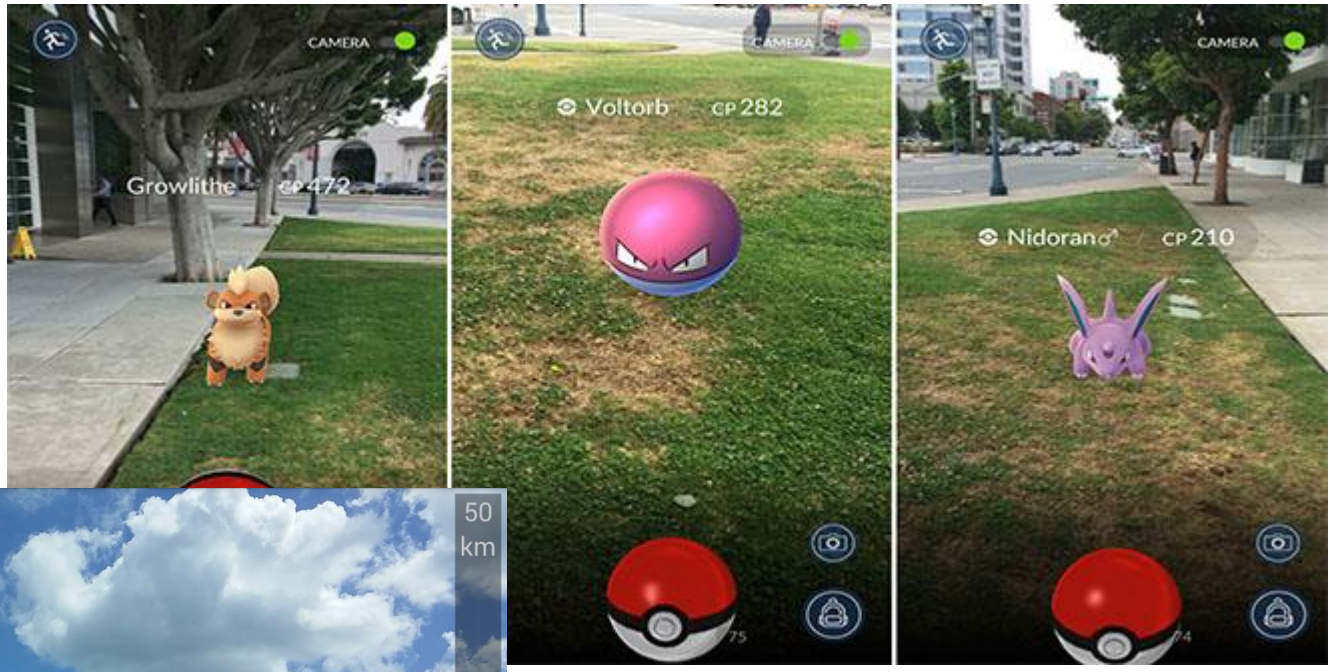
Überlagertes Lebervenen-Modell bei OP



Kamerabild und virtuelles Möbelstück

- http://de.wikipedia.org/wiki/Erweiterte_Realität
- http://www.computerwoche.de/a/augmented-reality-app-hilft-leberchirurgen-bei-der-op,2544964

Augmented Reality: Beispiele



Pokémon Go

Flightradar 24

http://www.vrguru.com/2016/07/07/pokemon-go-now-available-android-ios/
http://www.flightradar.org/flightradar-24-pro-for-iphone-and-ipad-devices-ipad-devices/

Augmented Reality: Beispiele

https://www.youtube.com/watch?v=29OSzQbxpcQ

Augmented Reality: Beispiele

https://www.youtube.com/watch?v=vDNzTasuYEw

Echtzeit?

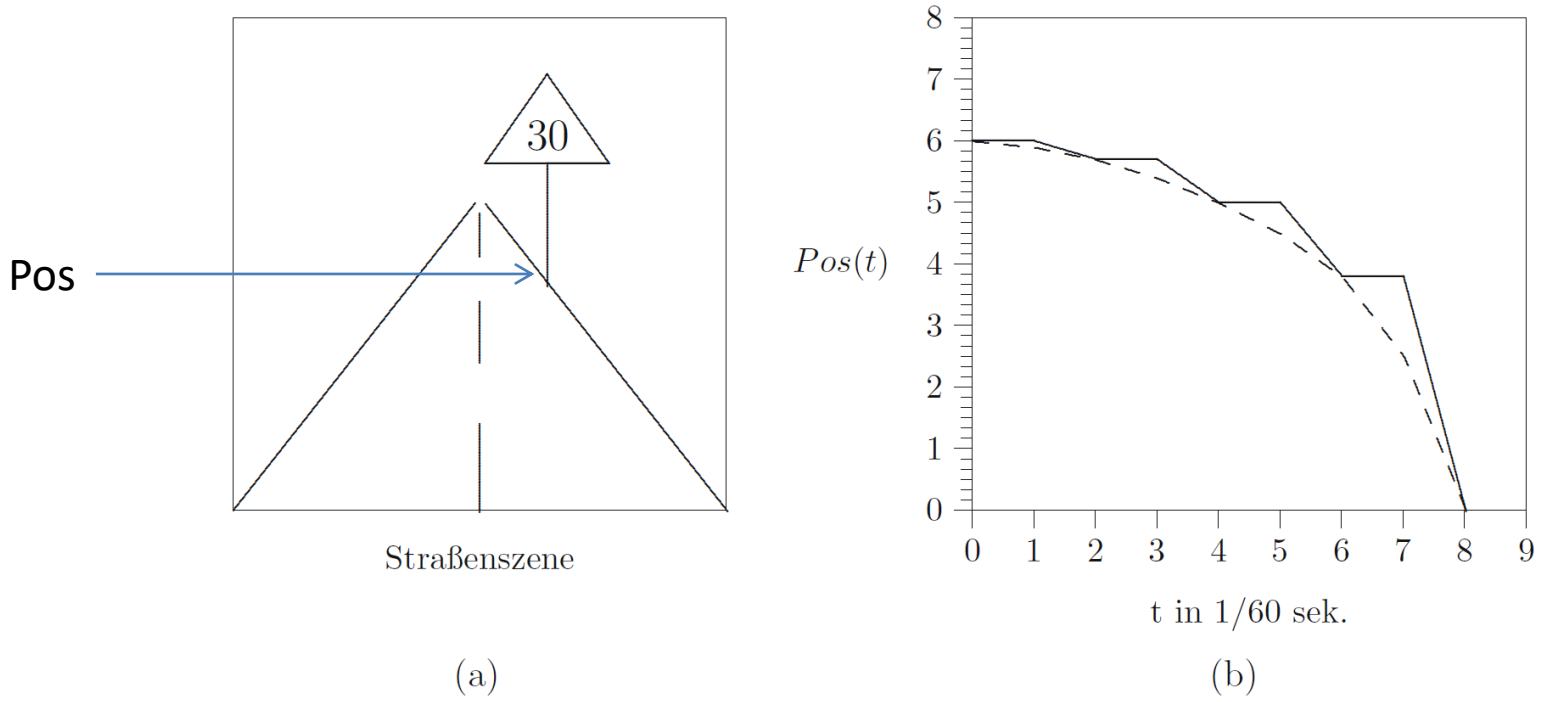
- Interaktive/Echtzeit-CG ⇔ Nicht-Echtzeit-CG
  - Anwendungsabhängig – z.B. Interaktion mit Benutzer?
  - Anforderung an Reaktionszeit bestimmt u.U. Komplexität eines Modells

  - Grenzen verwischen durch gesteigerte Rechenleistung zunehmend

- Bildwiederholfrequenz
  - Anzahl der Bilder pro Sekunde in ***Hz*** oder ***fps***
  - Flüssiger Bildeindruck ab ~30Hz

Bildgenerierrate, Bildanzeigerate

- Oft feste Bildanzeigerate (z.B. Monitor 60Hz)
- Zwischenspeicher im Anzeigegerät
- Doppelbilder wenn Bildgenerierrate < Bildanzeigerate



Nischwitz et al.: Computergrafik und Bildverarbeitung Band 1, Computergrafik, 3. Auflage, Vieweg+Teubner

Zeilensprungverfahren (Interlace)

- Erhöhung der Bildwiederholfrequenz: zeilenweiser Bildaufbau

Halbbild A Halbbild B

- Beispiel: analoges Fernsehen (PAL Fernsehnorm)
  - Übertragung von 50 Halbbildern/Sekunde → 25 Vollbilder/Sekunde
- Vorteile:
  - Erhöhung der zeitlichen Auflösung bei gleicher Datenmenge
  - Reduktion des subjektiven Flimmereindrucks/Doppelbildern
- Nachteile:
  - Artefakte (z.B. an horizontalen Kanten)

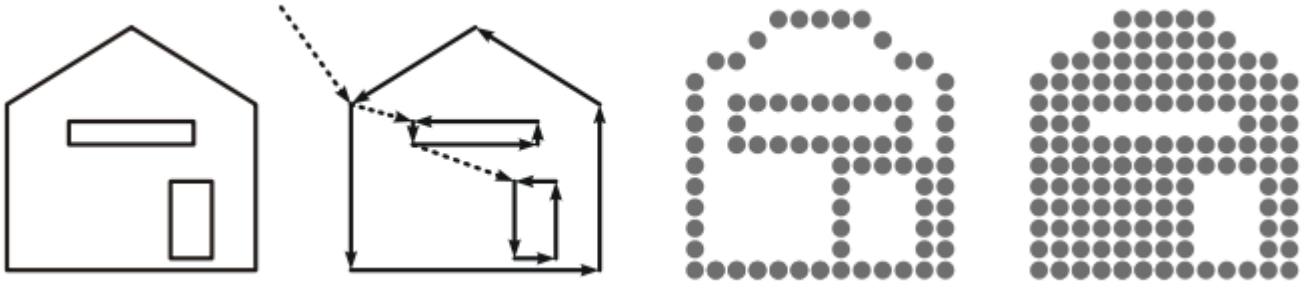
Vollbildverfahren (Progressive Scan)

- Übertragung von vollständigen Bildern
  - Bsp: zeilenweiser Bildaufbau bei Röhrenmonitoren

- Vorteile:
  - Schärferes, ruhigeres Bild, reduzierte Artefakte
- Nachteile:
  - Mehr Daten oder geringere Bildwiederholfrequenz

Dimensionalität, Raster- / Vektorgrafik

- Dreidimensionale Grafik
  - Beschreibung einer 3D-Szene
  - Rendern eines 2D Bildes zur Anzeige auf dem Bildschirm (Bildsynthese)
- Zweidimensionale Grafik
  - Raster- ⇔ Vektorgrafik

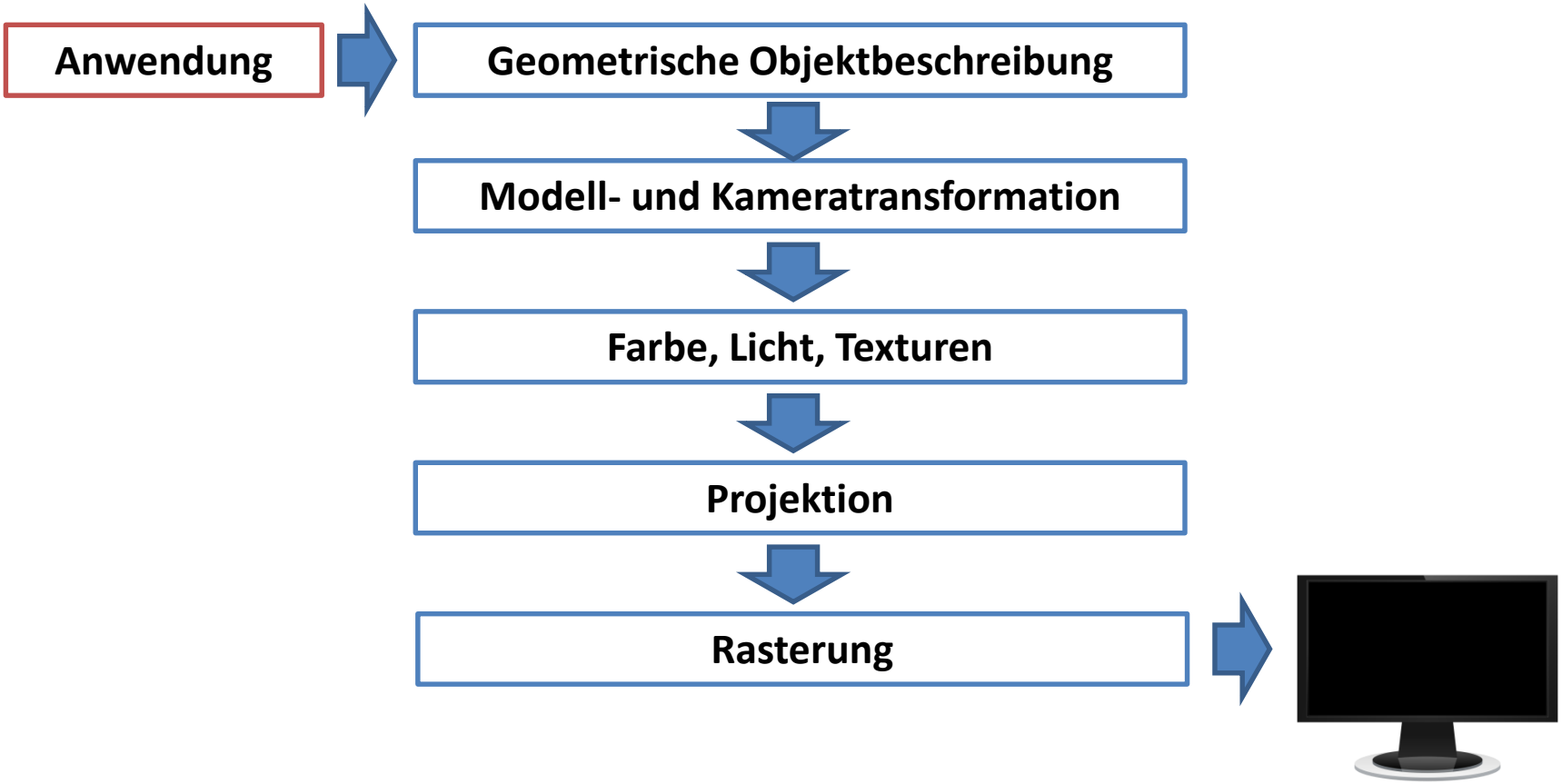


  - Heute nur noch Rasterbildschirme: Rastern von Szenen

    → Rasteralgorithmen

http://de.wikipedia.org/wiki/Computergrafik

Vereinfachte Grafikpipeline



Themenübersicht

1. **Einführung**
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. Raytracing
11. Volumenvisualisierung

1.2 EXKURS: KURZER GESCHICHTLICHER ABRISS

Geschichtlicher Abriss

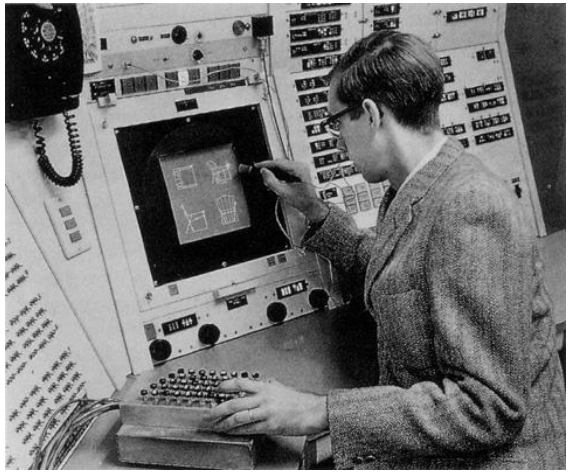
- Bis 1950: Grafikausgabe per Plotterzeichnung
- 1950: Whirlwind MIT. Erster Einsatz einer Kathodenstrahlröhre als Bildschirm
- 1952: Erstes Videospiel „Tennis for two“

- 1959: Erstes CAD-System (IBM DAC-1)

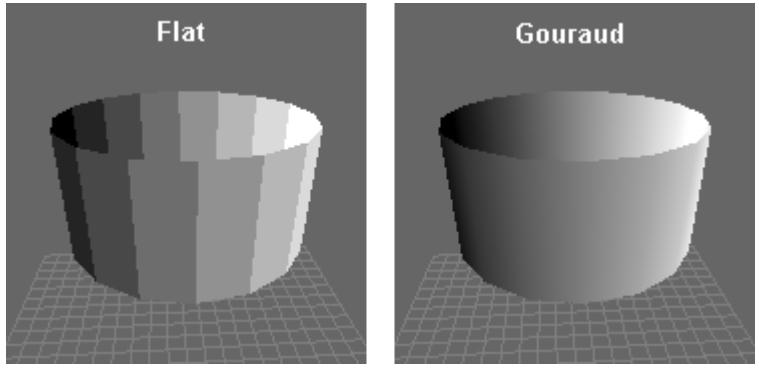
http://design.osu.edu/carlson/history/tree/images/dac.JPG
http://de.wikipedia.org/wiki/Tennis_for_Two

Geschichtlicher Abriss

- 1963: Beginn der modernen Computergrafik: Sketchpad
  Ivan Sutherland „Sketches and Systems“, MIT.
- 1962: Bresenham-Algorithmus zum Rastern von Linien
- 1965: CAD in der Flugzeugindustrie (Lockheed)
- 1968: Erste speicherröhren-basierte grafische Computerterminals (Computer Displays, Tektronix)
- 1968: Gründung von Evans & Sutherland



- 1971: Gouraud Shading (Henri Gouraud)



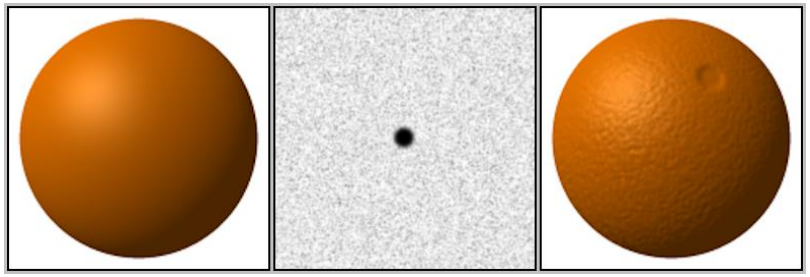
- 1974: Depth-Buffer (Edwin Catmull)

http://www.mprove.de/diplom/text/3.1.2_sketchpad.html
http://en.wikipedia.org/wiki/Gouraud_shading

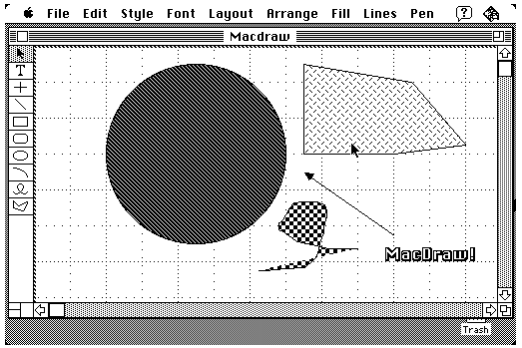
Geschichtlicher Abriss

- 1974: Anti-Aliasing (Herbert Freeman)
- 1975: Phong-Shading, Phong-Beleuchtungsmodell (Bui-Toung Phong)
- 1976: Reflection Mapping (Jim Blinn)

- 1978: Bump Mapping (Jim Blinn)

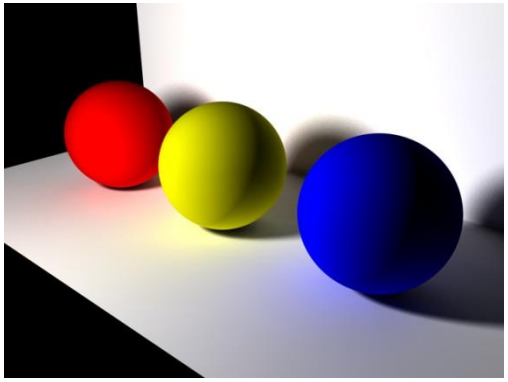


- Ca. 1979/1980: Raytracing für Reflexionsberechnungen
- Ende 70er/Anfang 80er: erste Spielfilme mit CGI-Anteil
- 1981: Gründung von Silicon Graphics
- 1980er: Verbreitung von grafikfähigen PCs
- 1984: MacDraw, MacPaint: Consumer-Markt!



http://de.wikipedia.org/wiki/Computergrafik
http://de.wikipedia.org/wiki/MacDraw

Geschichtlicher Abriss



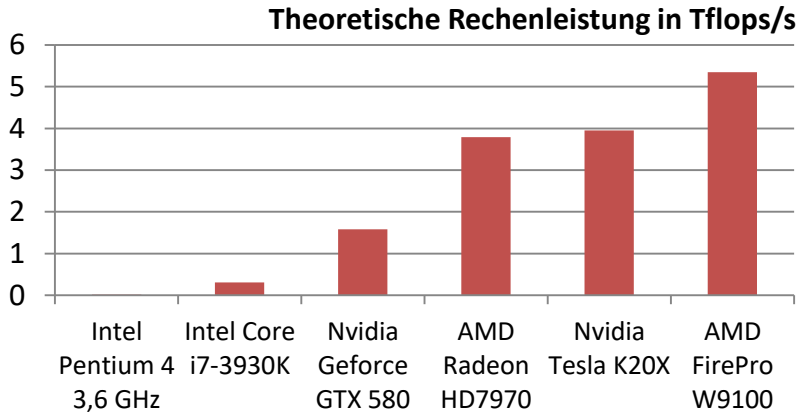
- 1984: Radiosity (Cornell University)
- 1980er: erste Standards für 2D und 3D Computergrafik
  - PHIGS: Programmer's Hierarchical Interactive Graphics System, API für 3D Szene
- 1986: Gründung von Pixar, 1989 erste Version von RenderMan
- 1986: Veröffentlichung der Rendergleichung und von Path Tracing
  - Mathematisches Fundament für globale Beleuchtung

- 1992: OpenGL 1.0 Standard wird von Silicon Graphics veröffentlicht
- 1995: Erster vollständig als Computeranimation erstellter Kinofilm *Toy Story*

http://de.wikipedia.org/wiki/OpenGL
http://de.wikipedia.org/wiki/Radiosity_%28Computergrafik%29

Geschichtlicher Abriss

- Mitte der 90er Jahre
  - 3D Game Engines, z.B. Unreal Engine
  - Computergrafik im Internet
- 1996: 3dfx Vodoo Graphics. Erster 3D-Grafikchip im nicht-professionellen Bereich
  - 3D Grafik erobert den Consumer-Markt
- 2001: erster komplett computeranimierter Spielfilm mit realistischen Charakteren *Final Fantasy.*
- Ab Mitte der 2000er Jahre: extreme Leistungssteigerung der Grafikkarten
  - Shader-Programmierung
  - General Purpose Computation
- 2004: OpenGL Shading Language



http://de.wikipedia.org/wiki/Non-photorealistic_Rendering

ZUSAMMENFASSUNG

Zusammenfassung

- Generative Computergrafik: Erstellung künstlicher Bildwelten
- Typische Anwendungsgebiete:
  - Ausbildungssimulation
  - CAD/CAM
  - Visualisierung
  - Informationsvisualisierung
  - Unterhaltung
- Virtual Reality ⇔ Augmented Reality
- Echtzeitanforderungen ⇔ Realitätstreue
- Grafikpipeline: typische Verarbeitungsschritte von der Anwendung/Modell zur Darstellung auf dem Bildschirm

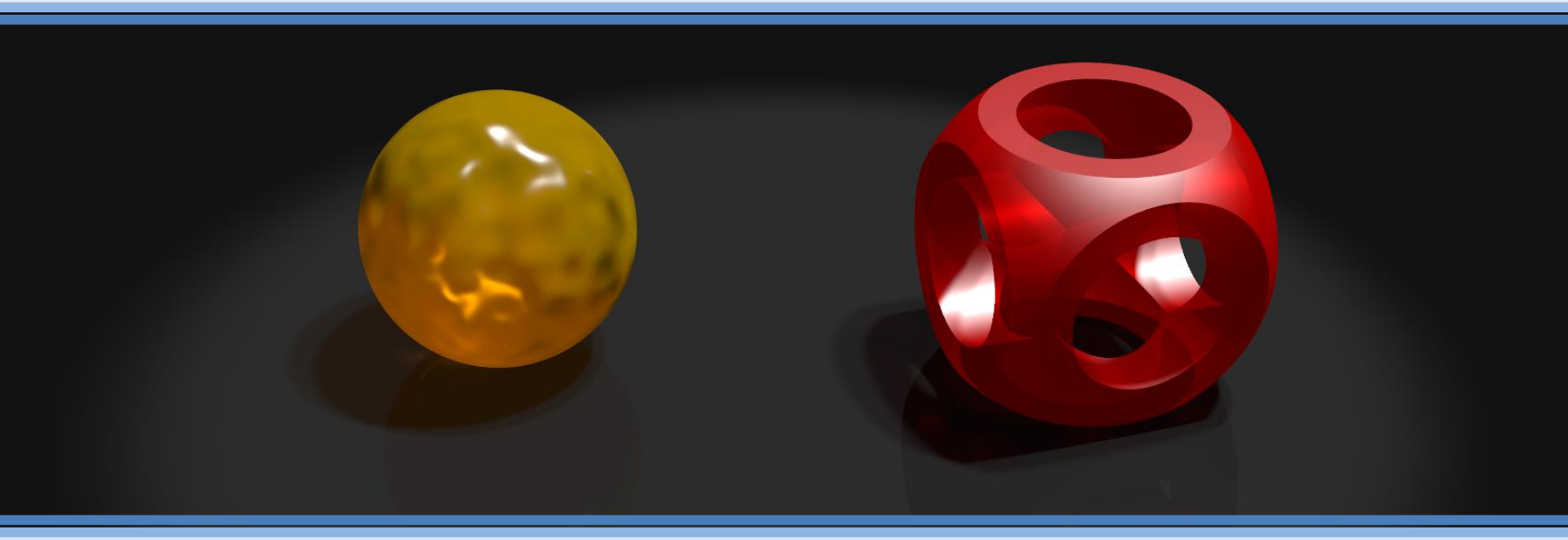
ÜBUNGSAUFGABEN

Übungsfragen Kapitel 1

- Beschreiben Sie die Begriffe „Virtual Reality“ und „Augmented Reality“ – was ist der Unterschied?
- Was ist der Unterschied zwischen Bildgenerierrate und Bildanzeigerate?

Computergrafik

T. Hopp



Themenübersicht

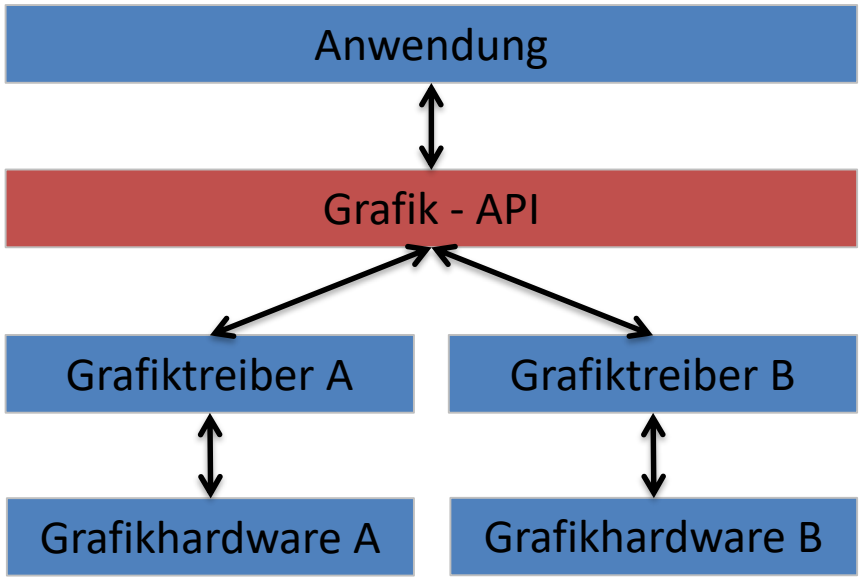
1. Einführung
2. **Programmierbibliotheken / OpenGL**
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. Raytracing
11. Volumenvisualisierung

2.1 PROGRAMMIERBIBLIOTHEKEN

Mittel zur Abstraktion

- Entwicklung von GPUs zur Beschleunigung
- Spezifische Instruktionen, Schnittstellen für verschiedene Hersteller, Generationen etc.

  ➔ API zur **Abstraktion des Hardwarezugriffs**



Grafik-API Standards

- Beispiele für international normierte Standards (ISO):
  - GKS / GKS-3D: Graphical Kernel System
  - PHIGS / PHIGS+: Programmer‘s Hierarchical Interactive Graphics System

- Sehr allgemein und schwerfällig
- Nutzen Möglichkeiten der Beschleunigung durch Grafikkarten nicht aus
- Haben sich in der Praxis nicht durchgesetzt

De-facto-Standards der Industrie:

- **Direct3D** (Microsoft, ***proprietär***)
- **OpenGL** (ursprünglich SGI, ***open source, plattformunabhägig***)
  - Heute: OpenGL Architecture Review Board Working Group

Evolution der Programmierbibliotheken

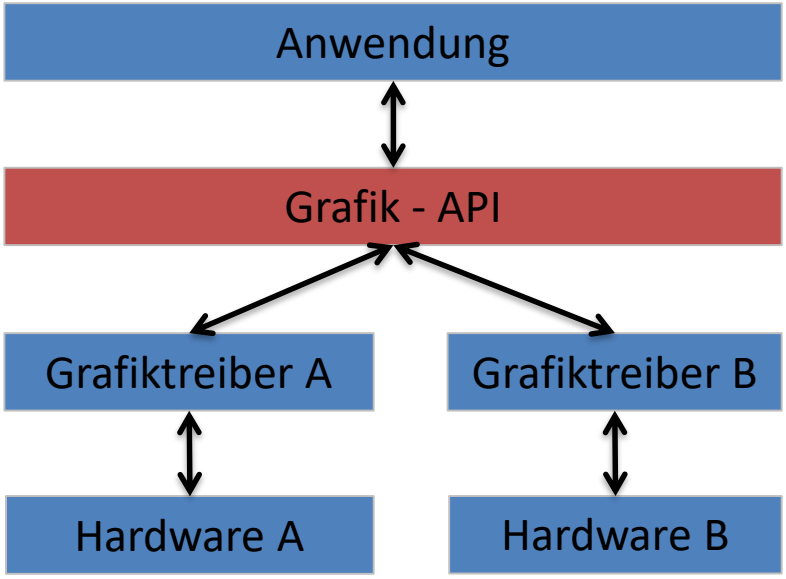
- Fixed Function
  - Konfigurierbare Module für z.B. Beleuchtung, Puffer etc.
  - Zusammensetzbar zur Grafik-Pipeline
  - „Klassisches“ OpenGL, Direct3D
- Shader
  - Teile der Grafik-Pipeline können frei definiert werden
  - Z.B. Implementierung eines eigenen Beleuchtungsmodells
  - OpenGL Shading Language (GLSL), C for graphics (Cg), High Level Shading Language (HLSL)
- Programmierbare Pipeline
  - Volle Freiheit in der Programmierung von GPUs mit Nutzung der vollen Beschleunigungsmöglichkeiten
  - OpenCL, CUDA

Historische Entwicklung

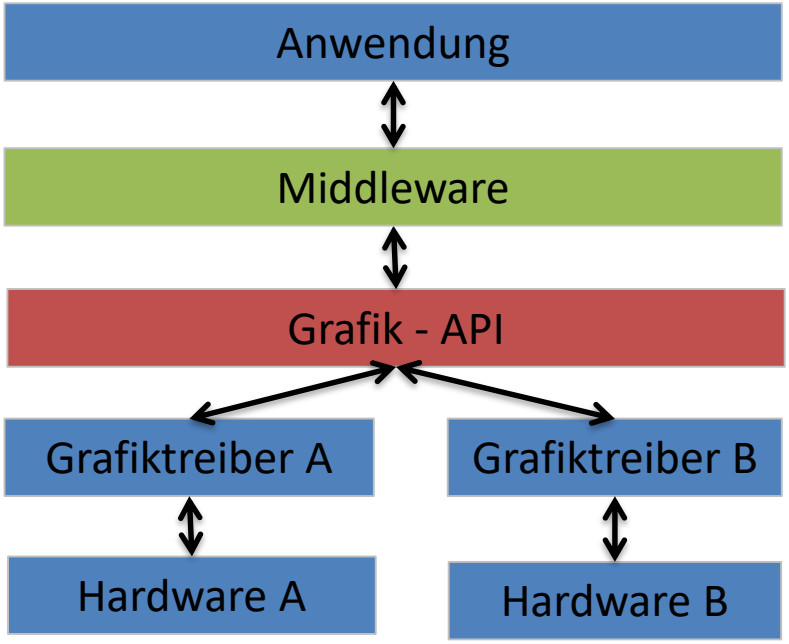
Jahr	Normen	Industrie-Standards		Shading-Sprachen		GPGPU	
1982		IrisGL (SGI)					
1985	GKS						
1988	GKS-3D						
1989	PHIGS						
1992		OpenGL 1.0					
1995			Direct3D 1.0				
1997		OpenGL 1.1	Direct3D 5.0				
1999		OpenGL 1.2	Direct3D 7.0				
2001		OpenGL 1.3	Direct3D 8.0				
2002		OpenGL 1.4	Direct3D 9.0		Cg 1.0		
2003		OpenGL 1.5	Direct3D 9.0a	GLSL 1.0	Cg 1.1		
2004		OpenGL 2.0	Direct3D 9.0b	GLSL 1.1	Cg 1.3		
2006		OpenGL 2.1	Direct3D 9.0c	GLSL 1.2	Cg 1.5		
2007			Direct3D 10.0		Cg 2.0		CUDA 1.0
2008		OpenGL 3.0	Direct3D 10.1	GLSL 1.3	Cg 2.1		CUDA 2.0
2009		OpenGL 3.2	Direct3D 11.0	GLSL 1.5	Cg 2.2	OpenCL 1.0	CUDA 3.0
2010		OpenGL 4.1		GLSL 4.1	Cg 3.0	OpenCL 1.1	CUDA 3.2
2012		OpenGL 4.3		GLSL 4.3	Cg 3.1		CUDA 5.0
2013			Direct3D 11.2	GLSL 4.4		OpenCL 2.0	CUDA 6.0
2014		OpenGL 4.5		GLSL 4.5			
2015			Direct3D 12.0			OpenCL 2.1	CUDA 7.0
2016							CUDA 8.0
2017		OpenGL 4.6		GLSL 4.6		OpenCL 2.2	CUDA 9.1

Architekturmodelle

Immediate-mode (IM)

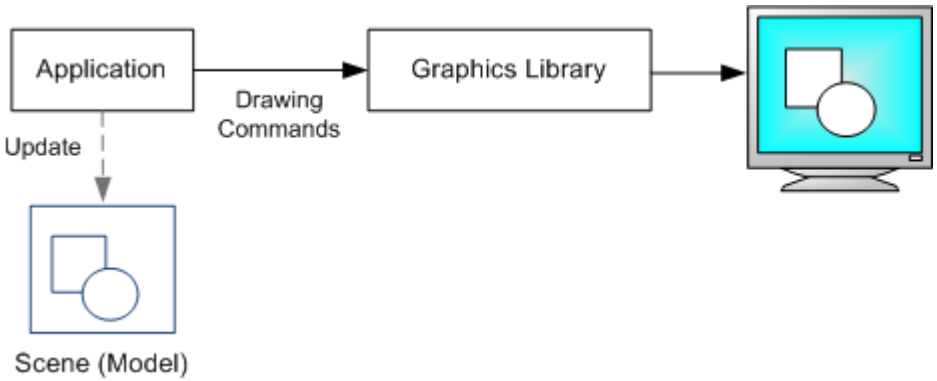


Retained-mode (RM)



Architekturmodelle

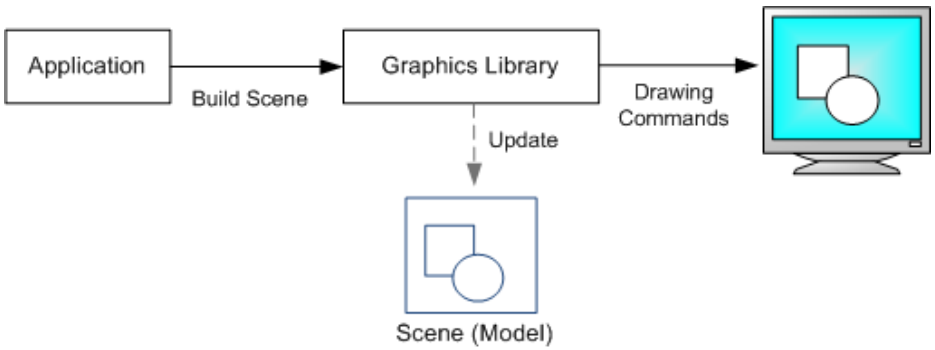
- Immediate Mode



- Minimaler Overhead
- Anwendung verantwortlich für Szene

z.B. OpenGL, Direct3D

- Retained Mode



- Vereinfachtes Szenen-Management in der Anwendung
- Geringere Performanz

z.B. Java3D (als Middleware zwischen Applikation und OpenGL / Direct3D)

https://msdn.microsoft.com/en-us/library/windows/desktop/ff684178%28v=vs.85%29.aspx

2.2 OPEN GL

Programmierinterface OpenGL

- Definition Architecture Review Board:

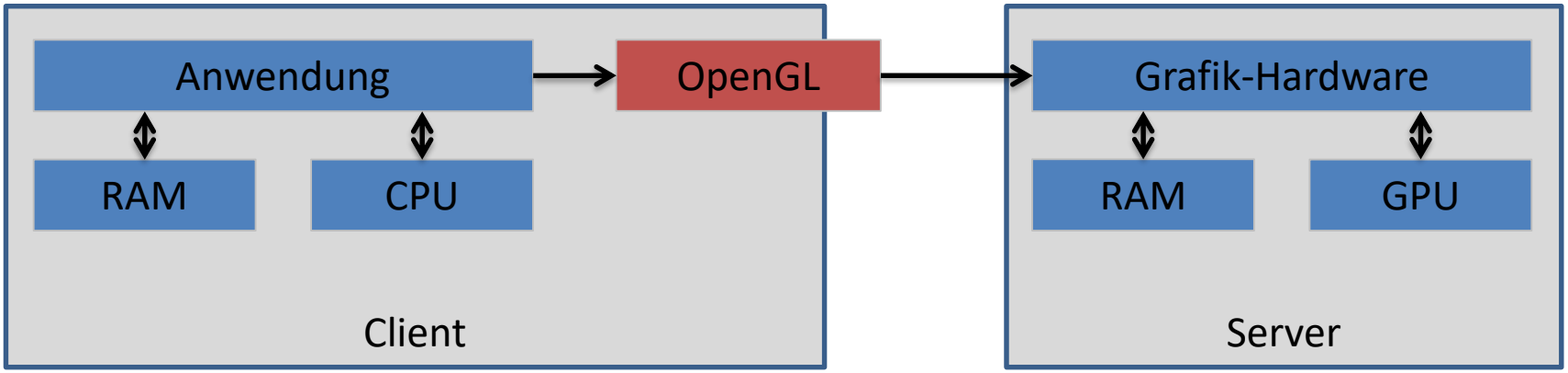
  „ein Software-Interface zur Hardware. Zweck ist die Darstellung von zwei- und dreidimensionalen Objekten mittels eines Bildspeichers.“

- Sprachunabhängig: C/C++, FORTRAN, …
- Betriebssystem- und Windowsystem-unabhängig
- Weite Unterstützung durch Hardwarehersteller

- In Anwendung verantwortlich für das Rendern einer Szene
  - Implementierung des OpenGL-API erfolgt in der Regel durch Systembibliotheken oder Grafikkarten-Treiber
- Immediate Mode: Anwendung definiert und verwaltet Szene

Programmiermodell

- Client-Server-Modell



- Serverseite: Treiber für Hardware setzt OpenGL in Bilder um
- Emulation auch in Software möglich
- „Extensions“: Hardware-spezifische Erweiterung möglich

Versionen

- OpenGL 1.x: Fixed Function Pipeline
- OpenGL 2.x: Shader
  - OpenGL Shading Language (GLSL): Vertex + Fragment Shader
- OpenGL 3.x: weitere Shader
  - OpenGL Shading Language (GLSL): Geometry Shader + Deprecation
  - Ab 3.2: „Compatibility Profile“ und „Core Profile“
- OpenGL 4.x: Programmierbare Rendering Pipeline
  - OpenCL
  - Shader: Tesselation

- Varianten:
  - „OpenGL ES“ (Embedded Systems): z.B. Smartphones
  - „WebGL“: 3D-Inhalte in Browsern

Bibliotheken

- **GL**: Low-Level-API (Basisfunktionalität)
  - Plattformunabhängig: keine Darstellung von Fenstern, kein Behandeln von Benutzereingaben

```
#include <GL/gl.h>
```

- **GLU**: OpenGL Utility Library
  - Erweiterung der Low-Level-API um
    - Vereinfachte geometrische Funktionen
    - High-Level-Primitive (z.B. Kugel, Zylinder, Torus)
    - Non-uniform rational B-Splines (NURBS)
    - Polygon Tesselation
    - Skalieren von Bildern
    - Generieren von Texturen aus Bildern
    - Fehlermeldungen

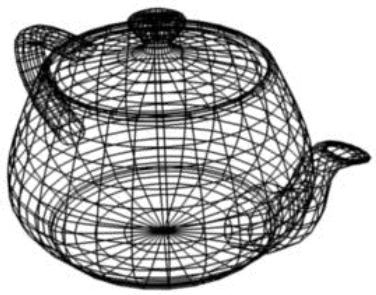
```
#include <GL/glu.h>
```
→ beinhaltet bereits `gl.h`!

Bibliotheken

- **GLUT**: OpenGL Utility Toolkit
  - Erweiterung der Grafik-API von GL und GLU um
    - Fensterverwaltung
    - Callback-Mechanismen (Keyboard, Mouse)
    - Timer-Prozeduren
    - Einfache Menüsteuerungen
    - Highest-Level Primitive (z.B. Teapot)

```
#include <GL/glut.h>
```

→ beinhaltet bereits `glu.h`!



http://battis.net/staging/tag/opengl/

Namenskonventionen

- Bibliotheksaufrufe beginnen mit folgendem Präfix:

`gl`	Alle OpenGL Funktionen, CamelCase-Schreibweise
`glu`	Alle Aufrufe der Utility Bibliothek
`glut`	Alle Aufrufe des Utility Toolkits
`GL_`	Vordefinierte Konstanten der Headerdateien Flags zum Steuern der Modi

Datentypen

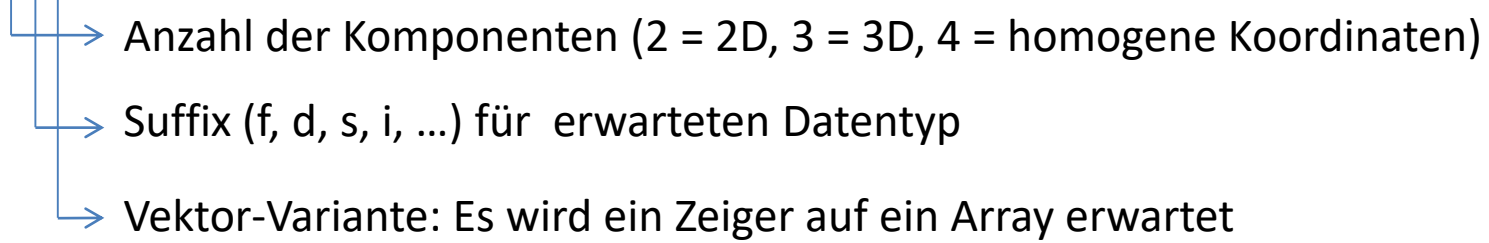
- OpenGL definiert eigene Datentypen → plattformunabhängig!
- Bibliotheksaufrufe enthalten Suffixe die auf erwarteten Datentyp hinweisen

Datentyp	Suffix	OpenGL-Typ	Bits
signed char	b	GLbyte	8
short	s	GLshort	16
int	i	GLint	32
float	f	GLfloat	32
double	d	GLdouble	64
unsigned char	ub	GLubyte	8
unsigned short	us	GLushort	16
unsigned int	ui	Gluint, GLenum	32
signed int 64	i64	GLint64	64
unsigned int 64	ui64	Gluint 64, GLbitfield	64

Beispiel: Datentypen und Namenskonventionen

- OpenGL-Aufrufe zur Definition eines Punktes mit Koordinate $x = 1, y = 2$

```
glVertex2i(1, 2);
glVertex2f(1.0, 2.0);

GLfloat v[] = {1.0, 2.0};
glVertex2fv(v);
```



OpenGL ist eine State Machine

- Periodisches Neuzeichnen der Szene
- API-Zugriff setzt eine Zustandsvariable, z.B. aktuelle Farbe
- Zustand bleibt so lange erhalten bis er explizit geändert wird
- Default-Werte für alle Variablen, z.B.
  - Ansicht- und Projektionstransformationen
  - Polygon Drawing Modes
  - Positionen und Eigenschaften des Lichtes, Materialeigenschaften
- Zustandsänderungen und -abfrage

```
glColor3f(1.0,0.0,0.0);             // current color set to red

glEnable(GL_DEPTH_TEST);            // Enable depth buffer testing
glDisable(GL_DEPTH_TEST);           // Disable depth buffer testing

GLboolean state;
state = glIsEnabled(GL_DEPTH_TEST);         // false
state = glIsDisabled(GL_DEPTH_TEST);        // true
glGetBooleanv(GL_DEPTH_TEST, &state);       // false
```

Setup

- Voraussetzungen:
  - Compiler + Entwicklungsumgebung für C/C++
  - Treiber vom Grafikkartenhersteller / Alternative: Software Version
    - Siehe https://www.opengl.org/wiki/Getting_Started#Downloading_OpenGL
    - Enthält i.d.R. GLUT Umgebung
    - Z.B. freeglut: http://www.transmissionzero.co.uk/software/freeglut-devel/
- Einrichtung:
  - Linken der OpenGL / GLU / GLUT Bibliothek
    - „OpenGL32.lib“ / „freeglut.lib“ (Windows)
    - „libGL“ (Linux): -lGL
  - Einbinden der Include-Verzeichnisse für Headerdateien

Hello World Beispiel

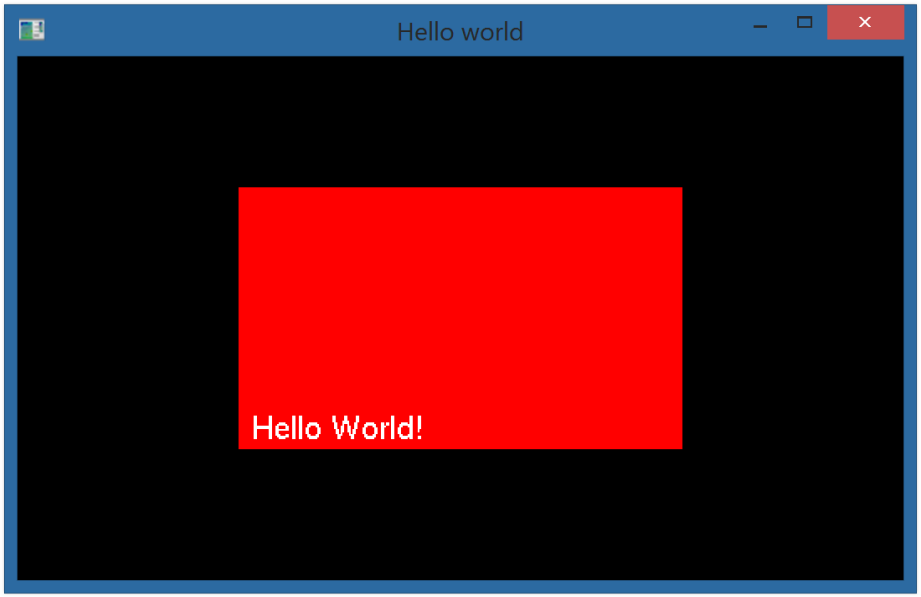
```
#include <glut.h>

int width = 640;
int height = 480;

void init(int argc, char** argv) {
   glutInit(&argc, argv);                    // init GLUT library
   glutInitDisplayMode(GLUT_SINGLE);         // Single buffered
   glutInitWindowSize(width, height);        // Window Size + Pos.
   glutInitWindowPosition(100, 100);
   glViewport(0,0,width,height);             // Viewport Size + Pos.
   glutCreateWindow("Hello world");
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0, width, 0, height);          // orthographic projection
   glMatrixMode(GL_MODELVIEW);
}
```

Hello World Beispiel

```
void display(void) {

   char *myText = "Hello World!";
   int j;

   glColor3f(1.0,0.0,0.0);

   glBegin(GL_POLYGON);
      glVertex3f((width/2)-(width/4), (height/2)-(height/4), 0.0);
      glVertex3f((width/2)+(width/4), (height/2)-(height/4), 0.0);
      glVertex3f((width/2)+(width/4), (height/2)+(height/4), 0.0);
      glVertex3f((width/2)-(width/4), (height/2)+(height/4), 0.0);
   glEnd();

   glColor3f(1.0,1.0,1.0);
   glRasterPos2i((width/2)-(width/4)+10,(height/2)-(height/4)+10);
   for (j=0; j<strlen(myText); j++) {
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,myText[j]);
   }

   glFlush();
}
```

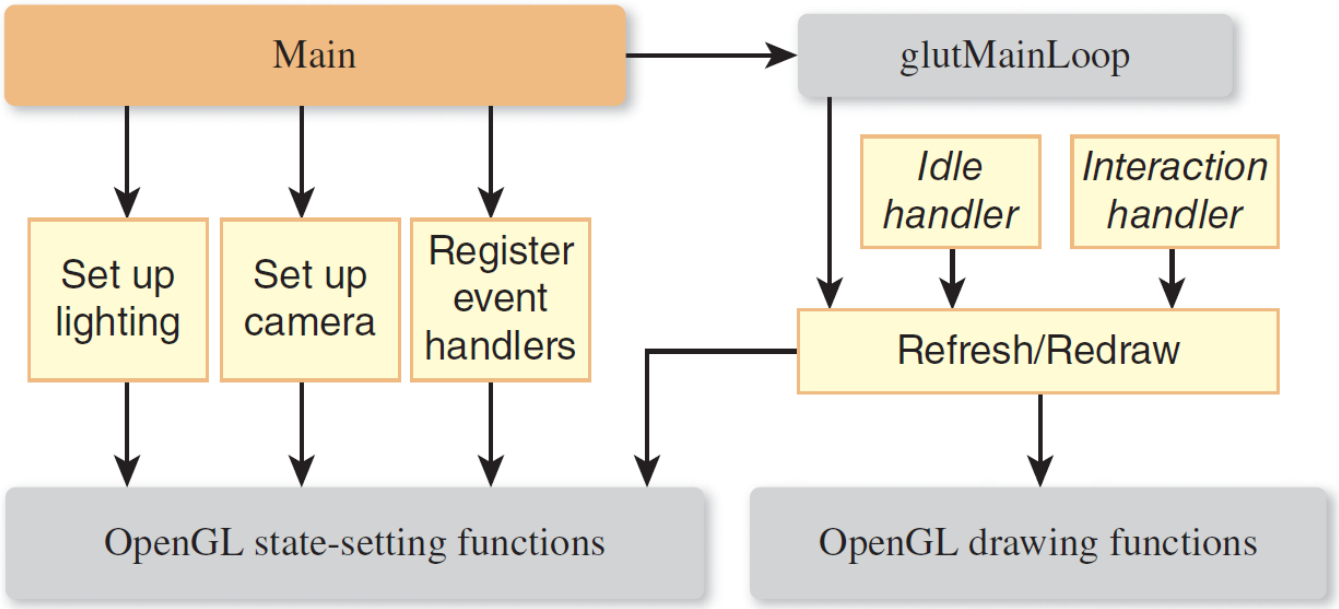
Hello World Beispiel

```
int main(int argc, char** argv) {
   init(argc, argv);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```



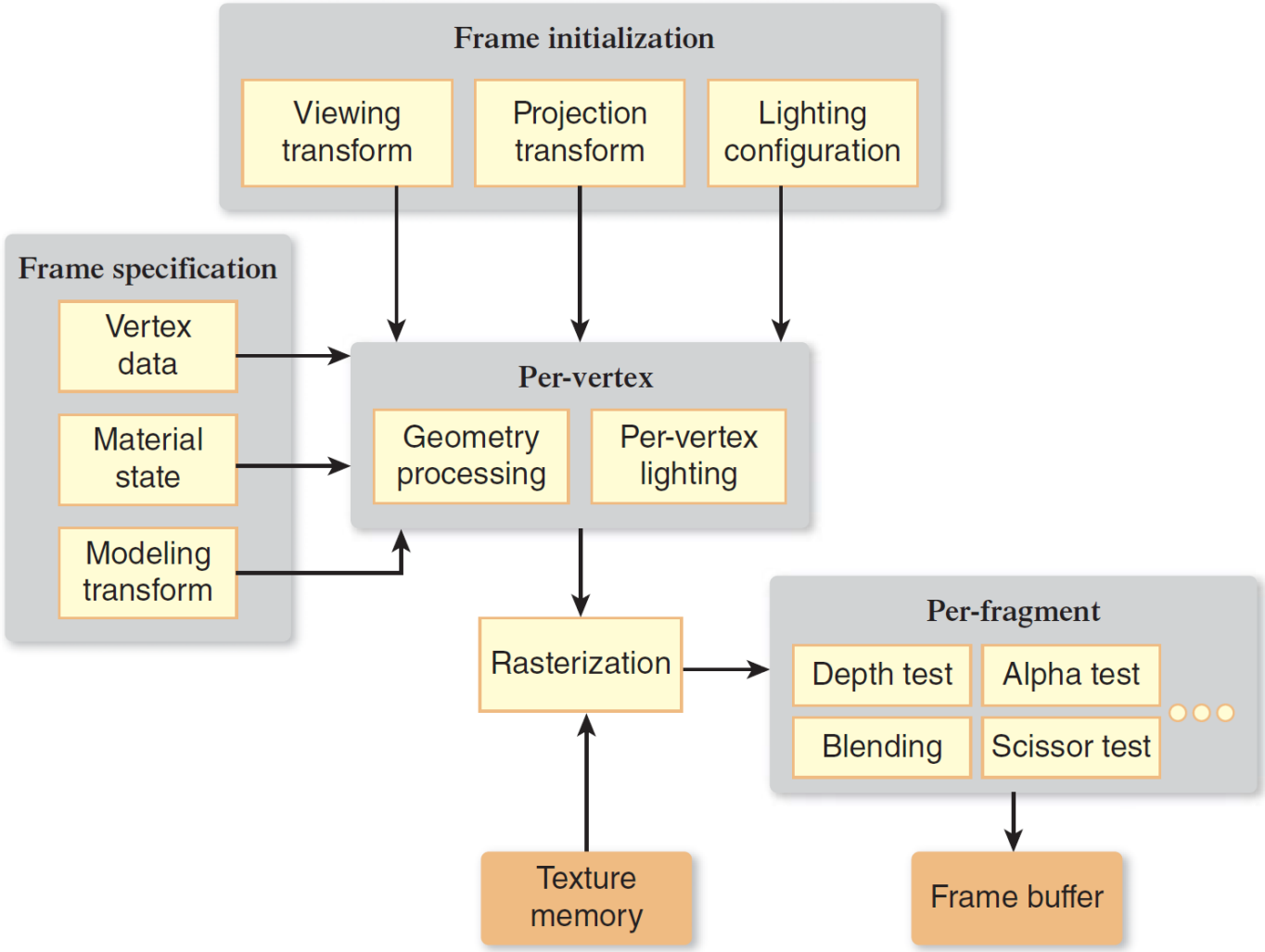
Typischer Programmablauf mit GLUT

1) Initialisierung der Grafikpipeline (glutInit)
2) Definition von Event Handlern
3) Spezifikation von Fenster, Bildausschnitt, Kamera, Licht, Laden von Daten (Meshes, Texturen)
4) Abgabe der Kontrolle an das Event-Pooling (glutMainLoop)



J.F. Hughes, A. Van Dam, M. McGuire, D. F. Sklar, J.D. Foley, S. K. Feiner, K. Akeley: „Computer Graphics Principles and Practice“ Third Edition, Addison-Wesley

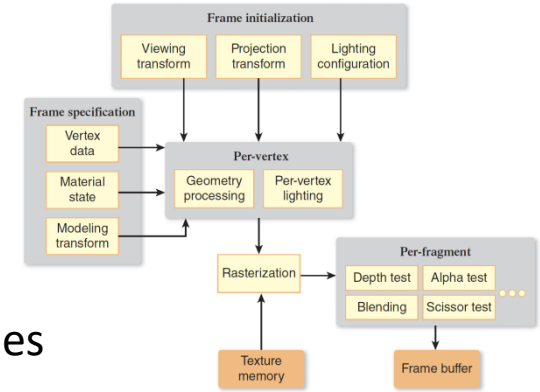
OpenGL Rendering Pipeline



J.F. Hughes, A. Van Dam, M. McGuire, D. F. Sklar, J.D. Foley, S. K. Feiner, K. Akeley: „Computer Graphics Principles and Practice“ Third Edition, Addison-Wesley

OpenGL Rendering Pipeline

1) Frame Initialization
   - Z.B. Definition der Kameraposition, Blickrichtung, Projektionsart
   - Z.B: Setzen von Statusvariablen für Licht
2) Frame Specification: Definition der Szene
   - Geometrische Daten, z.B. geometrische Körper, Meshes definiert durch Vertices (Punktkoordinaten)
   - Festlegung von Materialeigenschaften (Farben, Verhalten bei Licht)
   - Transformationen des Modells (Verschiebung, Rotation, …)
3) Per-Vertex-Operationen
   - Konvertierung der geometrischen Daten aus Modellkoordinaten zu Fensterkoordinaten auf dem Bildschirm
   - Berechnung der beleuchtungs- und materialabhängigen Farbe eines geometrischen Objektes



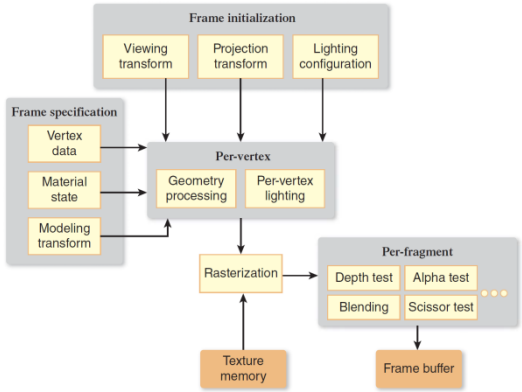
J.F. Hughes, A. Van Dam, M. McGuire, D. F. Sklar, J.D. Foley, S. K. Feiner, K. Akeley: „Computer Graphics Principles and Practice“ Third Edition, Addison-Wesley

OpenGL Rendering Pipeline

4) Rasterisierung
   - Umwandlung der „kontinuierlichen“ Geometrie in diskrete Pixel
   - Festlegung von Pixel-Farben basierend auf Licht, Texturen, etc.

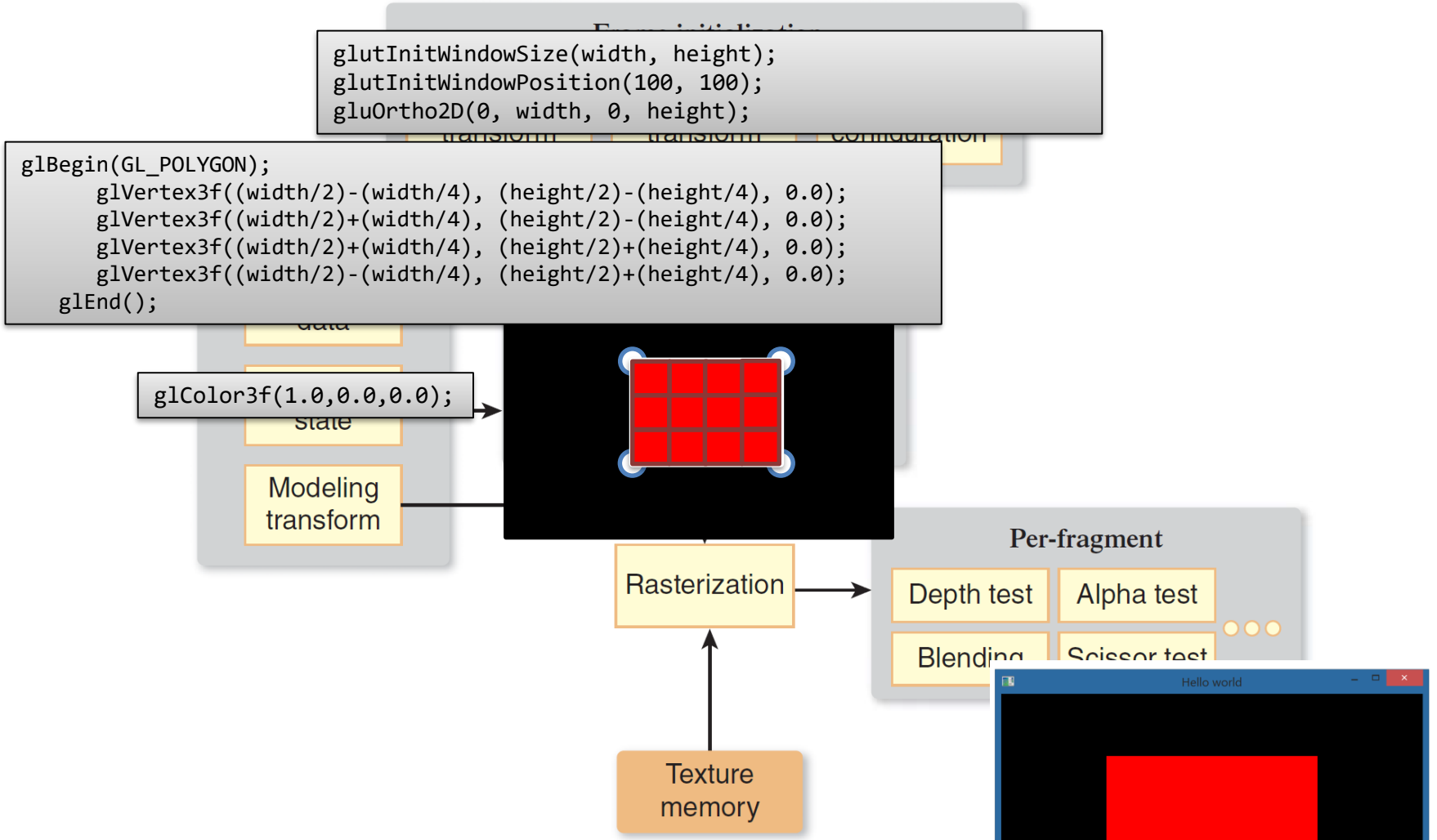
5) Per-Fragment Operationen
   - Festlegung welche/wie die Pixel auf dem Bildschirm dargestellt werden, z.B. Verdeckung von Objekten, Transparenz

6) Frame-Buffer
   - Zwischenpuffer für die Anzeige der gerenderten Szene



J.F. Hughes, A. Van Dam, M. McGuire, D. F. Sklar, J.D. Foley, S. K. Feiner, K. Akeley: „Computer Graphics Principles and Practice“ Third Edition, Addison-Wesley

OpenGL Rendering Pipeline für unser Beispiel



J.F. Hughes, A. Van Dam, M. McGuire, D. F. Sklar, J.D. Foley, S. K. Feiner, K. Akeley ldison-Wesley

Für Interessierte…

- http://glslsandbox.com/ - Beispiele für Implementierungen mit der OpenGL Shading Language, interaktiv veränderbar!
- Google: „openGL examples“
- http://wiki.delphigl.com/index.php/Hauptseite - deutsches Wiki zu OpenGL-Themen mit Funktionsübersicht und Erläuterungen
- D. Shreiner, G. Sellers, J. Kessenich, B. Licea-Kane: „OpenGL Programming Guide“, Eighth Edition, The official guide to learning OpenGL, Version 4.3: http://www.ics.uci.edu/~gopi/CS211B/opengl_programming_guide_8th_edition.pdf

ZUSAMMENFASSUNG

Zusammenfassung

- Programmierbibliotheken abstrahieren Zugriff auf Hardware
- Immediate mode ⇔ retained mode
- Fixed functions → Shaders → Programmierbare Pipeline

- OpenGL
  - Client-Server-Modell
  - Wichtige Bibliotheken: GL, GLU, GLUT mit entsprechenden Namenskonventionen für die Funktionsaufrufe
  - OpenGL definiert eigene Datentypen
  - OpenGL ist eine State Machine
  - OpenGL Rendering Pipeline: notwendige Verarbeitungsschritte bis man eine Szene auf dem Bildschirm dargestellt bekommt.

ÜBUNGSAUFGABEN

Programmierübung

1. Richten Sie sich eine Entwicklungsumgebung für die OpenGL Programmierung ein
2. Implementieren sie das Hello-World-Beispiel aus den Folien.
3. Ändern Sie die Farbe des Fensterhintergrundes. Verwenden Sie hierzu den Befehl **`glClearColor`** und **`glClear(GL_COLOR_BUFFER_BIT)`**
4. Ändern Sie Größe und Position des Fensters auf dem Bildschirm.
5. Ändern Sie die Farbe und Position des Rechtecks. Verwenden Sie bei der Farbeinstellung die Vektor-Variante
6. Zeichnen Sie statt des Rechtecks ein Dreieck in der Mitte des Fensters.
7. Fügen Sie der Szene ein weiteres Polygon in einer anderen Farbe hinzu.

Lösung

3. Änderung der Farbe des Fenster-Hintergrundes: Einfügen zu Beginn der Display-Funktion

```
glClearColor(0.0,0.0,1.0,1.0);
glClear(GL_COLOR_BUFFER_BIT);
```

4. Ändern der Größe und Position des Fensters: Ändern der Einträge in der init-Funktion, z.B.

```
glutInitWindowSize(width*2, height);
glutInitWindowPosition(200, 100);
```

5. Ändern von Farbe und Position des Rechtecks: Ändern in der Display Funktion: z.B. gelbes Rechteck nach rechts oben um je 50 Pixel verschoben

```
GLfloat myColor[3] = {1.0,1.0,0.0};
glColor3fv(myColor);

glBegin(GL_POLYGON);
glVertex3f((width/2)-(width/4)+50, (height/2)-(height/4)+50, 0.0);
glVertex3f((width/2)+(width/4)+50, (height/2)-(height/4)+50, 0.0);
glVertex3f((width/2)+(width/4)+50, (height/2)+(height/4)+50, 0.0);
glVertex3f((width/2)-(width/4)+50, (height/2)+(height/4)+50, 0.0);
glEnd();
```

Lösung (2)

6. Dreieck statt Rechteck zeichnen. Änderung in der Display-Funktion, z.B.

```
glBegin(GL_POLYGON);
glVertex3f((width/2)-(width/4), (height/2)-(height/4), 0.0);
glVertex3f((width/2)+(width/4), (height/2)-(height/4), 0.0);
glVertex3f((width/2), (height/2)+(height/4), 0.0);
glEnd();
```

7. Hinzufügen eines weiteren Polygons in einer anderen Farbe: Änderung in der Display-Funktion, z.B.

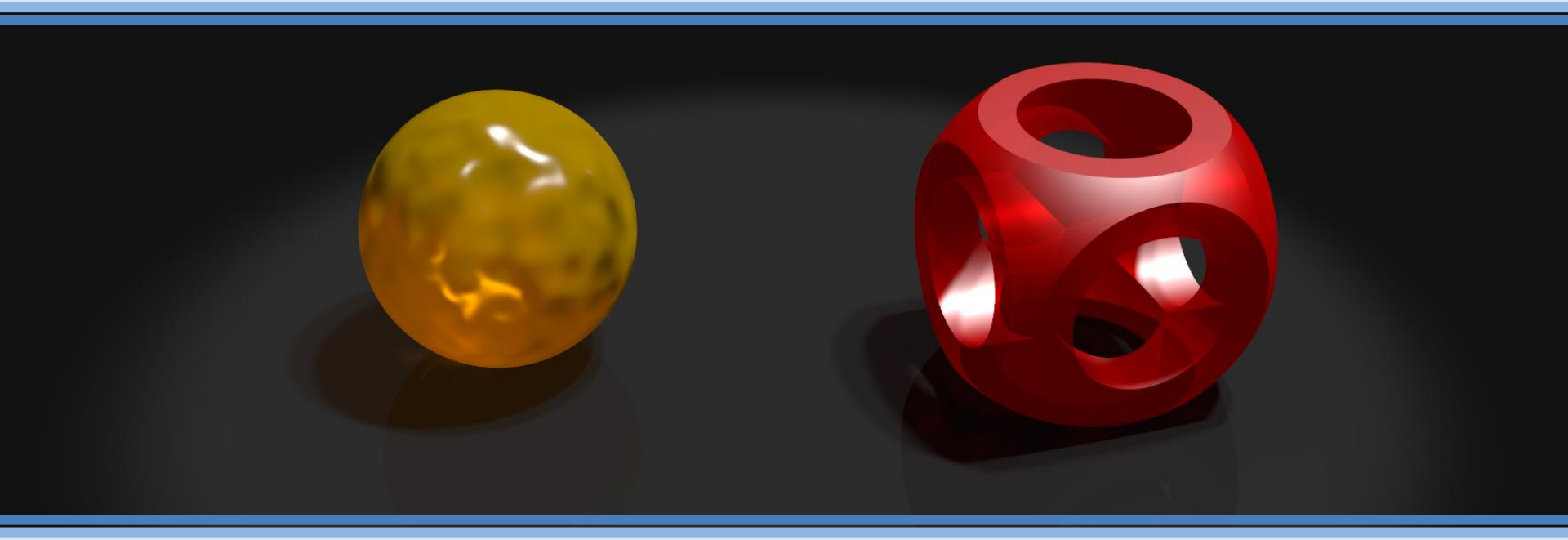
```
GLfloat myColor[3] = {1.0,1.0,0.0};
GLfloat myColor2[3] = {1.0,0.0,0.0};

glColor3fv(myColor);

glBegin(GL_POLYGON); // Rechteck
glVertex3f((width/2)-(width/4)-50, (height/2)-(height/4)-50, 0.0);
glVertex3f((width/2)+(width/4)-50, (height/2)-(height/4)-50, 0.0);
glVertex3f((width/2)+(width/4)-50, (height/2)+(height/4)-50, 0.0);
glVertex3f((width/2)-(width/4)-50, (height/2)+(height/4)-50, 0.0);
glEnd();

glColor3fv(myColor2);

glBegin(GL_POLYGON); // Dreieck
glVertex3f((width/2)-(width/4)+100, (height/2)-(height/4)+100, 0.0);
glVertex3f((width/2)+(width/4)+100, (height/2)-(height/4)+100, 0.0);
glVertex3f((width/2)+100, (height/2)+(height/4)+100, 0.0);
glEnd();
```

Übungsfragen Kapitel 2

- Welche Aufgabe übernehmen Programmierbibliotheken wie OpenGL bei der Grafikprogrammierung?
- Worin liegen die Unterschiede zwischen Immediate Mode und Retained Mode Programmierbibliotheken?
- Was sind Shader und welchen Vorteil bringen sie gegenüber Fixed Functon Bibliotheken?

Computergrafik

T. Hopp



Themenübersicht

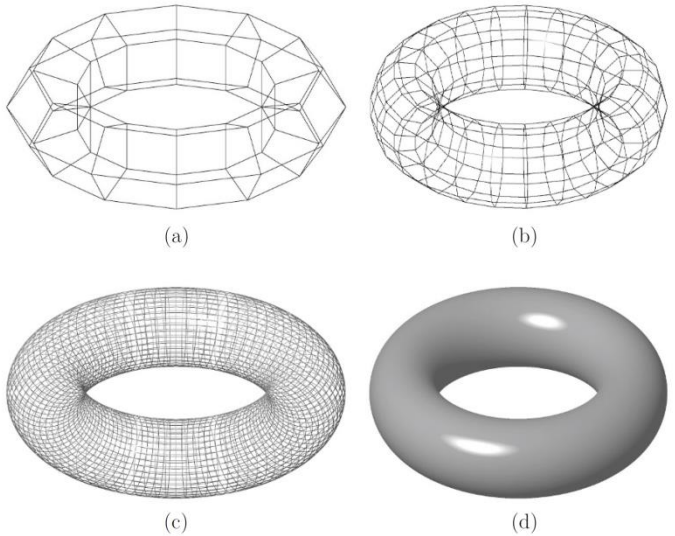
1. Einführung
2. Programmierbibliotheken / OpenGL
3. **Geometrische Repräsentation von Objekten**
4. Koordinatensysteme und Transformationen
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. Raytracing
11. Volumenvisualisierung

Einordnung

- Nachbildung von (realen) Objekten durch ***abstrakte*** Objekte
- Meist nur Abbildung der opaken Oberflächen eines Objektes
- Typische Fragestellungen:
  - Direkte Repräsentation von einfachen geometrischen Objekten ⇔ Annäherung an komplexe geometrische Objekte aus mehreren einfachen geometrischen Objekten
  - Exakte Beschreibung ⇔ ausreichende Approximation zur Darstellung

Planare Polygone

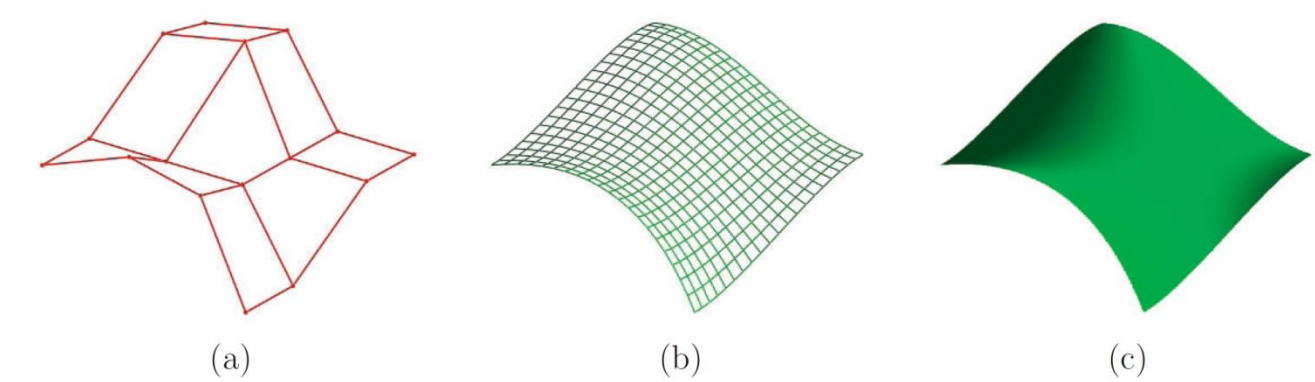
- Oberflächennetze aus planaren Polygonen am häufigsten zur Approximation von Objekten eingesetzt (→ Kapitel 3.2)
- Grundobjekte meist (→ Kapitel 3.1)
  - Dreiecke (*triangles*)
  - Viereck (*quads*)
- Vorteil: schnell berechenbar
- Nachteil: Genauigkeit der Approximation abhängig von Polygonauflösung



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Gekrümmte Flächen

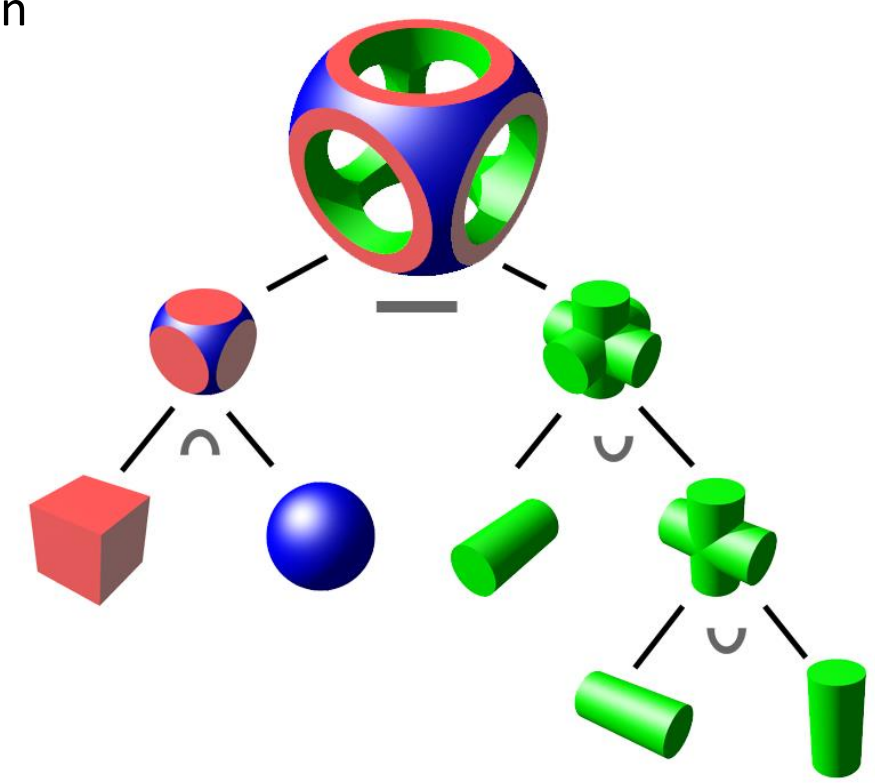
- Räumlich gekrümmte Flächen zur Approximation einer Oberfläche (→ Kapitel 3.3)
  - Bézier-Flächen, B-Splines, NURBS
- Kontrollpunkte = Parametrisierung der Fläche
- Vorteil: wenig Speicherplatz, exaktere Approximation möglich
- Nachteil: rechenaufwändigere Operationen



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Konstruktive Körpergeometrie

- Zusammensetzen komplexer Objekte aus elementaren Körpern (CSG = Constructive Solid Geometry)
  - Bool‘sche Operationen oder lineare Transformationen zur Vereinigung von elementaren Objekten
- Baumstruktur
  - Blätter = Grundobjekte
  - Knoten = Kombinationen

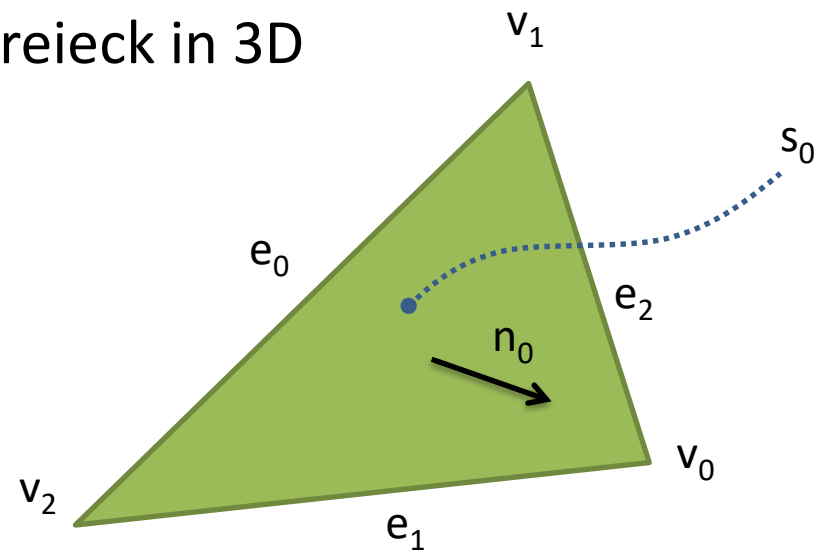


https://de.wikipedia.org/wiki/Geometrische_Modellierung

3.1. GRUNDKÖRPER UND PLANARE POLYGONE

Beschreibung eines Körpers

- Die Geometrie eines Körpers wird in der CG beschrieben durch
  - Punkte (*Vertices*)
  - Kanten (*Edges*)
  - Flächen (*Surfaces*)
  - Normalen
- Beispiel: Dreieck in 3D



$$v_0 = (v_{0,x}, v_{0,y}, v_{0,z})$$

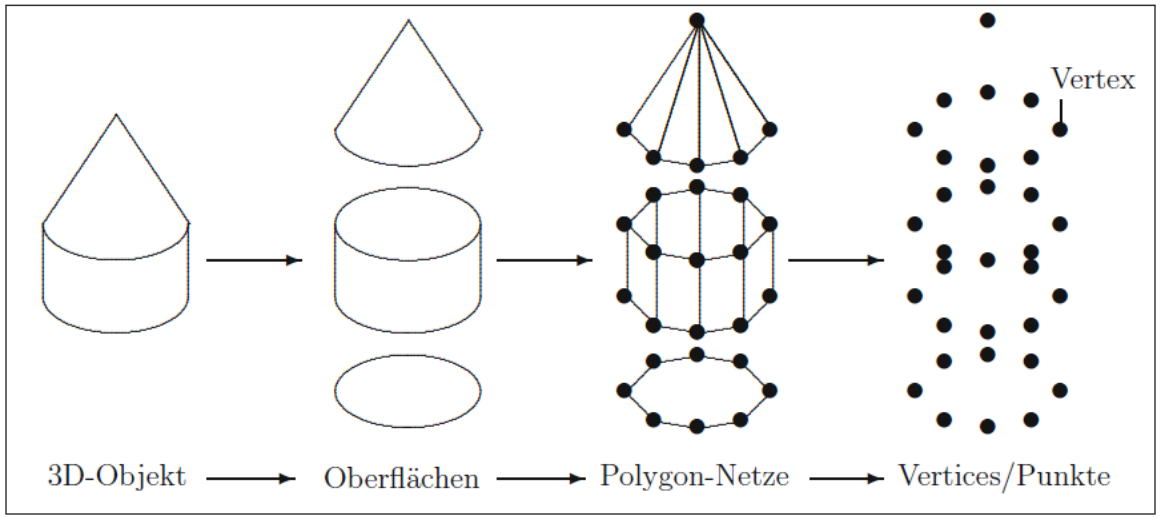
Geometrische Grundobjekte in OpenGL

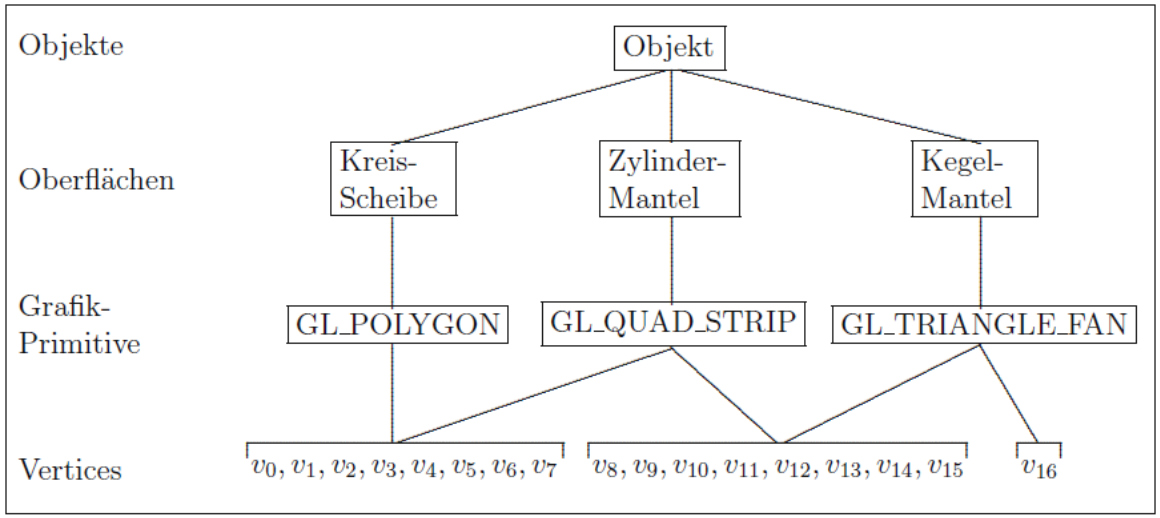
- In OpenGL: Beschreibung aller Objekte durch Vertices, Kanten und Flächen (planare Polygone)
  - Geordneter Satz von Vertices
  - Verbindung der Vertices durch Kanten
  - Eine bzw. Verbindung mehrerer Kanten ergeben Objekte bzw. Flächen

  ➔ **Polygonnetze**

- Definition komplexer Oberflächen durch Grundobjekten (OpenGL Grafik-Primitive)

Geometrische Objekte in OpenGL





A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Vertex-Definition in OpenGL

- „Vertex Array“-Methoden

Skalarform	Vektor-Form
`glVertex2f(x,y)`	`glVertex2fv(vec)`
`glVertex2d(x,y)`	`glVertex2dv(vec)`
`glVertex2s(x,y)`	`glVertex2sv(vec)`
`glVertex2i(x,y)`	`glVertex2iv(vec)`
`glVertex3f(x,y,z)`	`glVertex3fv(vec)`
`glVertex3d(x,y,z)`	`glVertex3dv(vec)`
`glVertex3s(x,y,z)`	`glVertex3sv(vec)`
`glVertex3i(x,y,z)`	`glVertex3iv(vec)`
`glVertex4f(x,y,z,w)`	`glVertex4fv(vec)`
`glVertex4d(x,y,z,w)`	`glVertex4dv(vec)`
`glVertex4s(x,y,z,w)`	`glVertex4sv(vec)`
`glVertex4i(x,y,z,w)`	`glVertex4iv(vec)`

← 2D: Kartesische Koordinate eines Punktes $v_0 = (x, y)$

← 3D: Kartesische Koordinate eines Punktes $v_0 = (x, y, z)$

← w = inverser Streckungsfaktor
→ Homogene Koordinaten

Kart. Koordinate eines Punktes $v_0 = (x/w, y/w, z/w)$

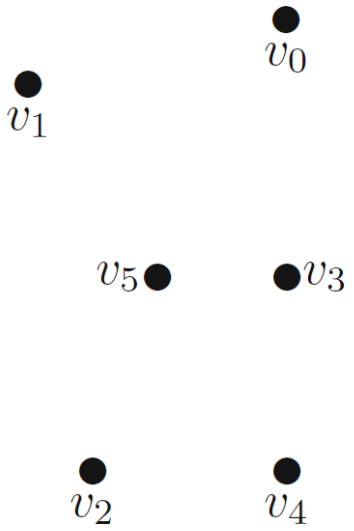
- Vektorform: Zeiger auf Array. Flexibler, schneller!

Definition von Grafik-Primitiven in OpenGL

- Verbindung von Vertices zu Flächen, Linien, Punkten

  glBegin / glEnd

1) GL_POINTS
   - Für jeden Vertex wird ein Punkt gerendert




```
glBegin(GL_POINTS);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
glEnd();
```

Punkteigenschaften in OpenGL

- Größe eines Punktes
  - *Default*: 1 Pixel
  - Änderung über Zustandsvariable: **`glPointSize(Glfloat size)`**
  - Zulässige Punktgröße ist hardwareabhängig. Abfrage mit:

```
Glfloat sizes[2];
Glfloat incr;

glGetFloatv(GL_POINT_SIZE_RANGE, sizes);
glGetFloatv(GL_POINT_SIZE_GRANULARITY, &incr);
```

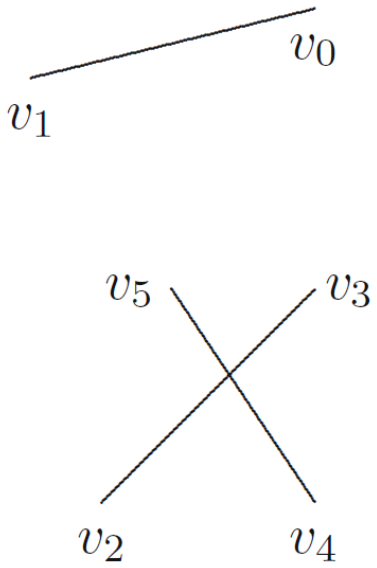
- Form eines Punktes: quadratisch
  - Runde Punkte nur durch Vortäuschung per Anti-Aliasing und Transparenzberechnung

```
glEnable(GL_POINT_SMOOTH);   glEnable(GL_BLEND);
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Definition von Grafik-Primitiven in OpenGL

2) GL_LINES

- Nicht-verbundene Liniensegmente zwischen jeweils zwei aufeinanderfolgenden Vertices



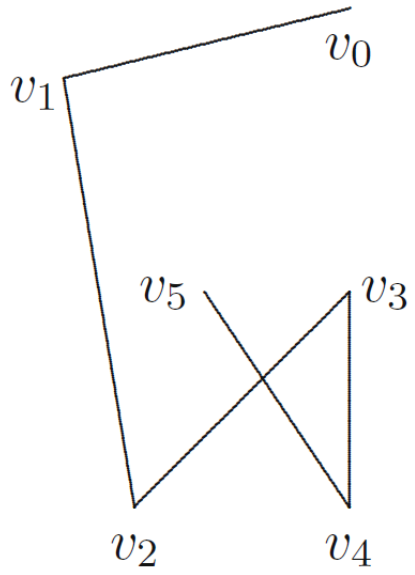

```
glBegin(GL_LINES);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
glEnd();
```

- Linienbreite analog zu Punktgröße: **`glLineWidth(Glfloat width)`**

Definition von Grafik-Primitiven in OpenGL

3) GL_LINE_STRIP

- Verbundene Linien zwischen jeweils zwei aufeinanderfolgenden Vertices



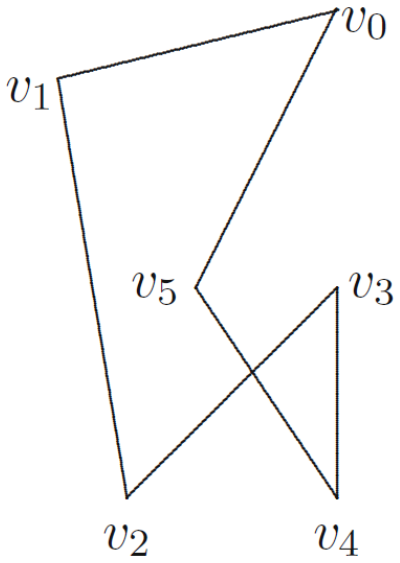

```
glBegin(GL_LINE_STRIP);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
glEnd();
```

- Keine Verbindung zwischen erstem und letztem Vertex

Definition von Grafik-Primitiven in OpenGL

4) GL_LINE_LOOP

- Verbundene Linien zwischen jeweils zwei aufeinanderfolgenden Vertices
- Verbindung zwischen erstem und letztem Vertex (geschlossener Linienzug)

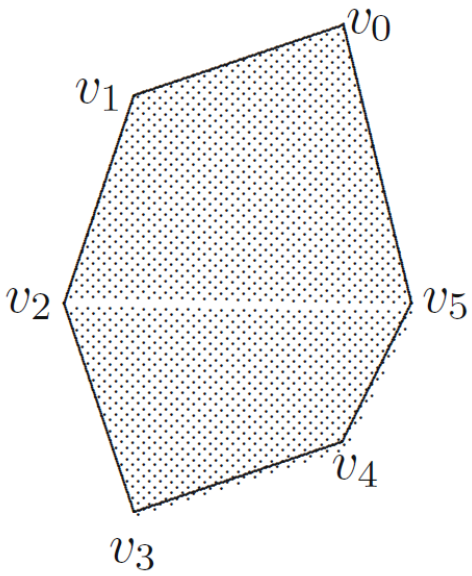



```
glBegin(GL_LINE_LOOP);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
glEnd();
```

Definition von Grafik-Primitiven in OpenGL

5) GL_POLYGON

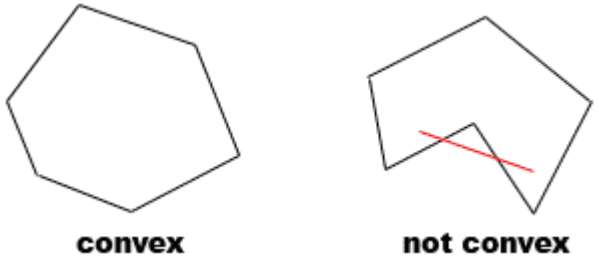
- Gefülltes <u>konvexes</u> Polygon
- Vieleck mit Anzahl Ecken = Anzahl Vertices
- *deprecated* – kann durch GL_TRIANGLE_FAN ersetzt werden




```
glBegin(GL_POLYGON);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
glEnd();
```

Polygone

- Anforderung in OpenGL: konvex und planar
  - Ein Polygon ist konvex, wenn alle Linien, die zwei beliebige Punkte des Polygons verbinden, vollständig innerhalb des Polygons liegen



  - Test auf Konvexität durch Ablaufen des Umrisses. Wenn man an Vertices immer in die gleiche Richtung abbiegt, ist das Polygon konvex

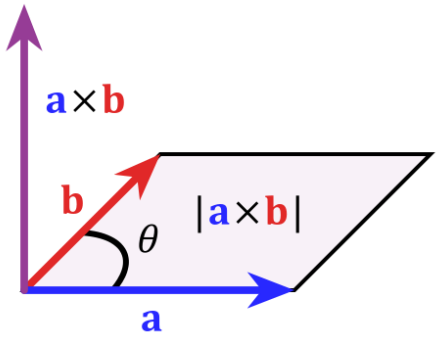
  - Planar: Alle Punkte liegen in einer Ebene

http://www.sunshine2k.de/coding/java/Polygon/Convex/polygon.htm

Mathematische Grundlagen

Kreuzprodukt zweier Vektoren

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$



Geometrisch: entspricht einem senkrechten Vektor zu beiden Vektoren (**=Normale**)

Skalarprodukt zweier Vektoren

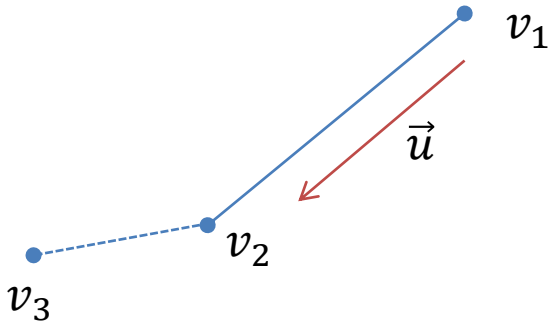
$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

Geometrisch: Kosinus des Winkels zwischen den beiden Vektoren.

- Skalarprodukt zweier Vektoren gegebener Länge = 0 wenn sie senkrecht zueinander stehen, und maximal wenn sie die gleiche Richtung haben.

Polygone: Test auf Konvexität

- Bestimmung der „Abbiegerichtung“ (Algorithmus nach P. Bourke)



Gerade g durch v_1 und v_2:

$g: \vec{w} = \vec{v}_1 + r\vec{u}$ mit $\vec{u} = \overrightarrow{v_2} - \overrightarrow{v_1}$, $r \epsilon \mathbb{R}$

Gleichungssystem:

$$v_{3_x} = v_{1_x} + ru_x$$
$$v_{3_y} = v_{1_y} + ru_y$$

Jeweils auflösen nach r und gleichsetzen ergibt:

$$(v_{3_x} - v_{1_x})u_y = (v_{3_y} - v_{1_y})u_x$$

Gleichung erfüllt wenn v_3 auf der Geraden liegt, d.h.

$$f(v_3) = (v_{3_x}u_y - v_{3_y}u_x) - (v_{1_x}u_y - v_{1_y}u_x)$$

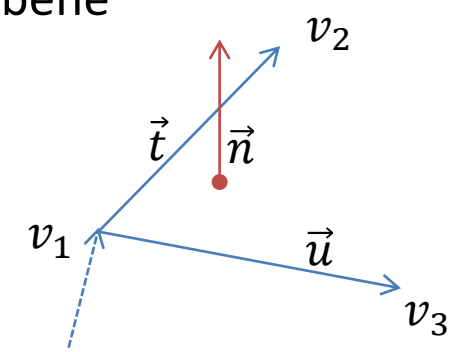
$f(v_3) = 0$	v_3 liegt auf der Geraden g
$f(v_3) > 0$	v_3 liegt rechts von der Geraden g
$f(v_3) < 0$	v_3 liegt links von der Geraden g

Implizite Form der Geradengleichung.
Auch 3D-Equivalent!

Polygone: Test auf Planarität

- Drei Punkte eines Polygons definieren eine Ebene

$$E: x = \overrightarrow{v_1} + r\underbrace{(\overrightarrow{v_2} - \overrightarrow{v_1})}_{\vec{t}} + s\underbrace{(\overrightarrow{v_3} - \overrightarrow{v_1})}_{\vec{u}}$$



- Test ob alle weiteren Punkte in dieser Ebene liegen
  - Z.B.: Einsetzen des Punktes in Ebenengleichung. Bei Lösung des LGS liegt der Punkt in der Ebene
  - Z.B.: Bestimmung des Normalenvektors der Ebene durch Kreuzprodukt

$$\begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \times \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} t_2 u_3 - t_3 u_2 \\ t_3 u_1 - t_1 u_3 \\ t_1 u_2 - t_2 u_1 \end{pmatrix}$$

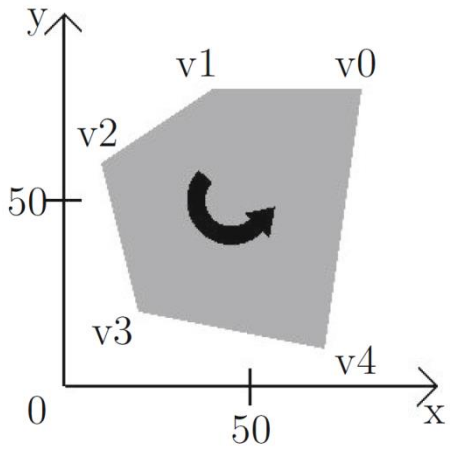
Damit v_4 in der Ebene liegt muss die Normale orthogonal zu $\vec{w} = (v_4 - v_1)$ sein:

$$f(v_4) = \vec{n} \cdot \overrightarrow{w} = n_1 w_1 + n_2 w_2 + n_3 w_3, \quad \text{orthogonal wenn } f(v_4) = 0$$

Polygone: Orientierung

- Vorder- und Rückseite bestimmt durch Vertex-Reihenfolge:
  - Definition gegen den Uhrzeigersinn: Vorderseite

```
glBegin(GL_POLYGON);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
glEnd();
```



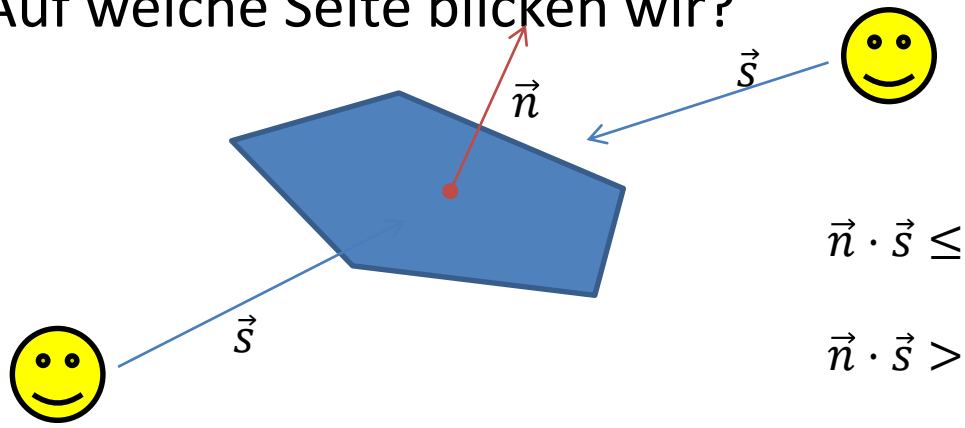
- Umdrehen der Konvention in OpenGL: **`glFrontFace(Glenum mode)`**

`GL_CW` **`GL_CCW`** (*default*)

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Polygone: Front/Back Face Culling

- OpenGL rendert standardmäßig Vorder- und Rückseite
- Deaktivieren des Renderns von Vorder- oder Rückseite durch Culling: **`glCullFace(Glenum mode)`**
  - **`GL_FRONT`**
  - **`GL_BACK`**
  - **`GL_FRONT_AND_BACK`**
- Erhöhung der Darstellungsgeschwindigkeit
- Auf welche Seite blicken wir?



$\vec{n} \cdot \vec{s} \leq 0$ Blick auf Vorderseite

$\vec{n} \cdot \vec{s} > 0$ Blick auf Rückseite

Polygone: Füllmodi

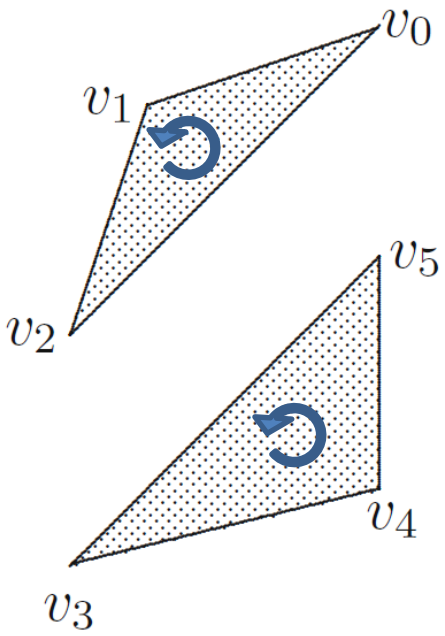
- Festlegung des Füllmodus durch

  `glPolygonMode(GLenum face, Glenum mode)`
- Modi:
  - Füllung der Polygonflächen: **`GL_FILL`**
  - Nur Zeichnen der Linien: **`GL_LINE`** (Drahtgittermodell)
  - Nur Zeichnen der Vertices: **`GL_POINT`**
- Für welche Polygonseite der Füllmodus gilt wird durch den **`face`**-Parameter festgelegt.

Definition von Grafik-Primitiven in OpenGL

6) GL_TRIANGLES

- Nicht verbundene Dreiecke aus jeweils drei aufeinanderfolgenden Vertices
- Reihenfolge wichtig, damit Dreiecke die gleiche Orientierung bekommen




```
glBegin(GL_TRIANGLES);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
glEnd();
```

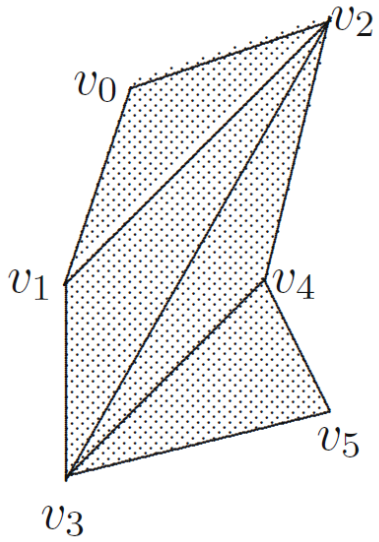
Definition von Grafik-Primitiven in OpenGL

7) GL_TRIANGLE_STRIP

- Verbundene Dreiecke
- Implizite Änderung der Vertexreihenfolge beim Rendern:
  - Dreieck 1: v_0, v_1, v_2
  - Dreieck 2: v_2, v_1, v_3
  - Dreieck 3: v_2, v_3, v_4

Ungerade n: $[v_{n-1}, v_n, v_{n+1}]$

Gerade n: $[v_n, v_{n-1}, v_{n+1}]$




```
glBegin(GL_TRIANGLE_STRIP);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
glEnd();
```

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Verbundene Dreiecke

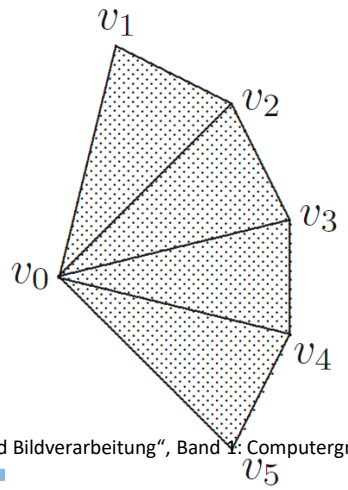
- Am häufigsten verwendete Grafik-Primitive
  - Approximation von komplexen Oberflächen:
    - Beliebig genau
    - Speichersparend
    - Am schnellsten zu zeichnen

- Konsistente Dreieckorientierung innerhalb eines Triangle-Strips durch Reihenfolge der Vertex-Verwendung.

Definition von Grafik-Primitiven in OpenGL

8) GL_TRIANGLE_FAN

- Fächer aus Dreiecken. Wie GL_TRIANGLE_STRIP, nur andere Vertex-Reihenfolge:
  - Dreieck 1: v_0, v_1, v_2 $[v_0, v_n, v_{n+1}]$
  - Dreieck 2: v_0, v_2, v_3
- Fläche entspricht dem eines konvexen Polygons aller Vertices
- Oft Verwendung für runde oder kegelförmige Flächen
- Orientierung konsistent



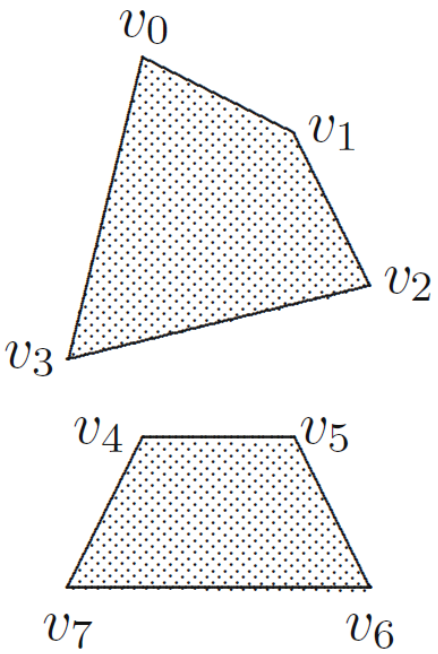
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

```
glBegin(GL_TRIANGLE_FAN);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
glEnd();
```

Definition von Grafik-Primitiven in OpenGL

9) GL_QUADS

- Einzelvierecke aus jeweils vier aufeinander folgenden Vertices
- (*deprecated* – kann durch GL_TRIANGLE_STRIP ersetzt werden*)




```
glBegin(GL_QUADS);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
   glVertex3fv(v6);
   glVertex3fv(v7);
glEnd();
```

**Reihenfolge der Vertices beachten!*

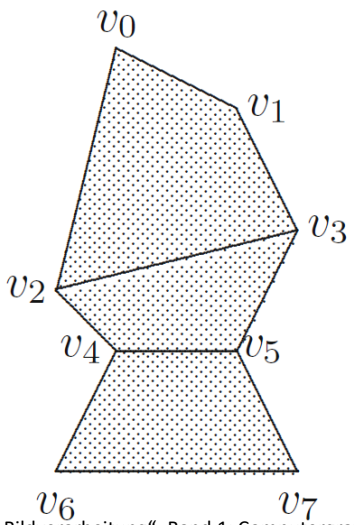
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Definition von Grafik-Primitiven in OpenGL

10) GL_QUAD_STRIP

- Verbundene Vierecke aus jeweils vier aufeinander folgenden Vertices
- (*deprecated* – kann durch GL_TRIANGLE_STRIP ersetzt werden)
- Reihenfolge wichtig für die Orientierung der Quads
  - Viereck 1: $[v_0, v_1, v_3, v_2]$
  - Viereck 2: $[v_2, v_3, v_5, v_4]$
  - Viereck 3: $[v_4, v_5, v_7, v_6]$

Vorschrift allgemein:
$[v_{2(n-1)}, v_{2(n-1)+1}, v_{2n+1}, v_{2n}]$



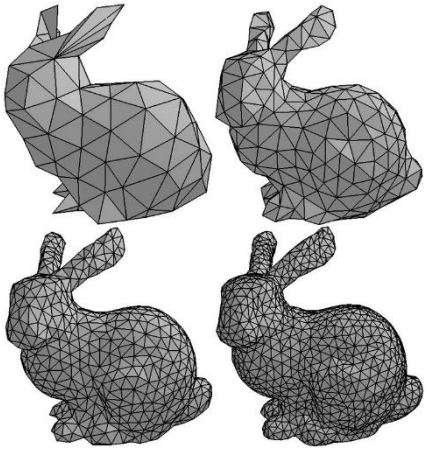
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

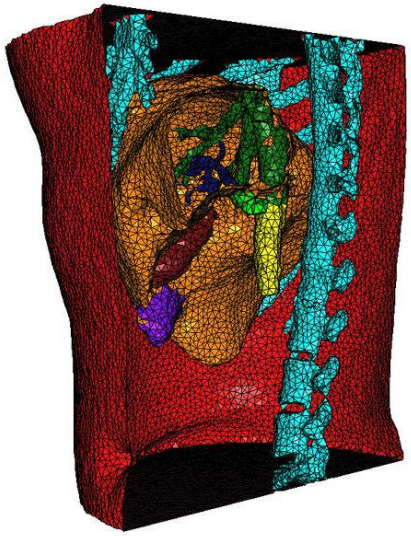
```
glBegin(GL_QUAD_STRIP);
   glVertex3fv(v0);
   glVertex3fv(v1);
   glVertex3fv(v2);
   glVertex3fv(v3);
   glVertex3fv(v4);
   glVertex3fv(v5);
   glVertex3fv(v6);
   glVertex3fv(v7);
glEnd();
```

3.2. POLYGONNETZE

Polygonnetze

- Zusammenfügen von Grundobjekten (i.d.R. Dreiecke, Vierecke) zur Approximation komplexer Geometrien

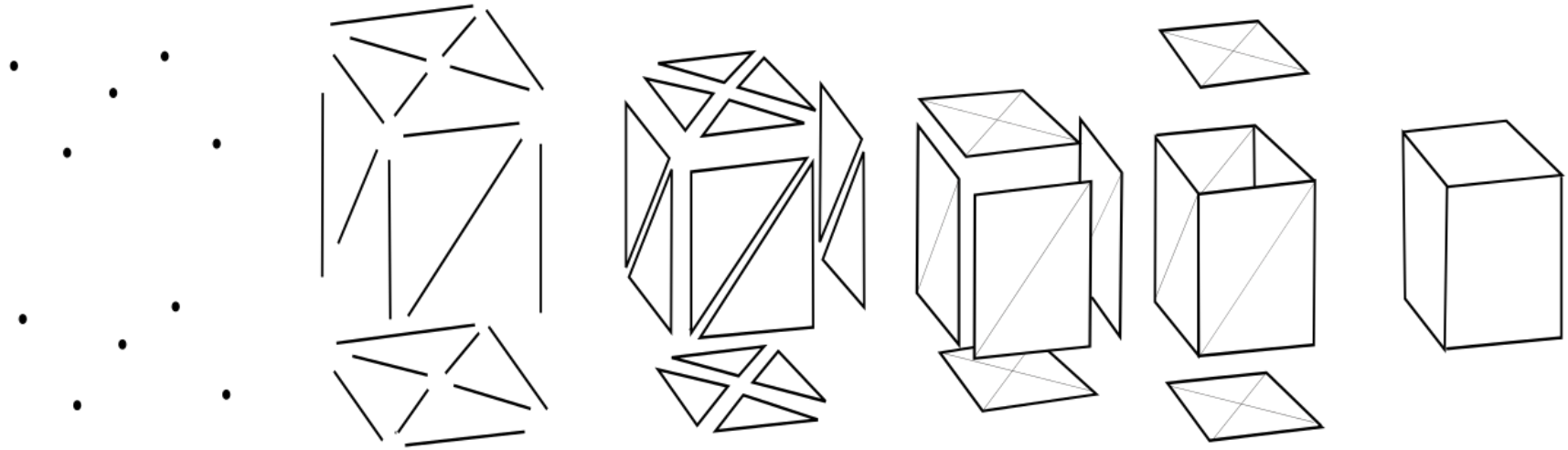




- Bestehend aus
  - Knoten (= Vertices)
  - Kanten (= Edges) → Flächen (= Faces)/Polygone
- Jeder Knoten muss mindestens eine Verbindung zum Restnetz haben
- In der Computergrafik üblicherweise Oberflächennetze

http://www.cmap.polytechnique.fr/~peyre/geodesic_computations/
https://de.wikipedia.org/wiki/Polygonnetz
http://doc.cgal.org/latest/Mesh_3/

Terminologie bei Polygonnetzen (Meshes)



vertices — Knoten

edge — Kanten

faces — Facetten

polygons — Polygone

surfaces — Oberflächen

https://en.wikipedia.org/wiki/Polygon_mesh

Datenstrukturen bei Polygonnetzen

Knotenliste

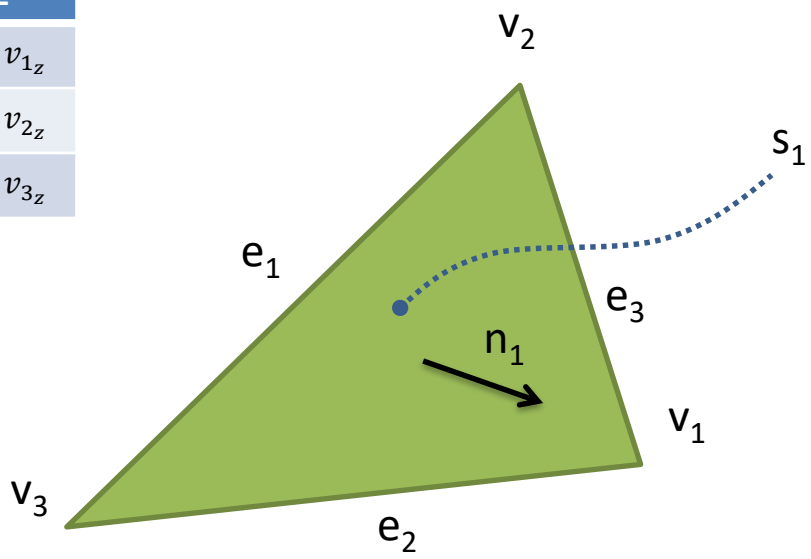
Beispiel: Dreiecksnetz

Polygone

ID	Knoten 1	Knoten 2	Knoten 3
1	1	2	3

Knoten

ID	X	Y	Z
1	v_{1_x}	v_{1_y}	v_{1_z}
2	v_{2_x}	v_{2_y}	v_{2_z}
3	v_{3_x}	v_{3_y}	v_{3_z}



- Polygon definiert als Liste von Zeigern auf Vertices

Datenstrukturen bei Polygonnetzen

Kantenliste

Beispiel: Dreiecksnetz

Polygone

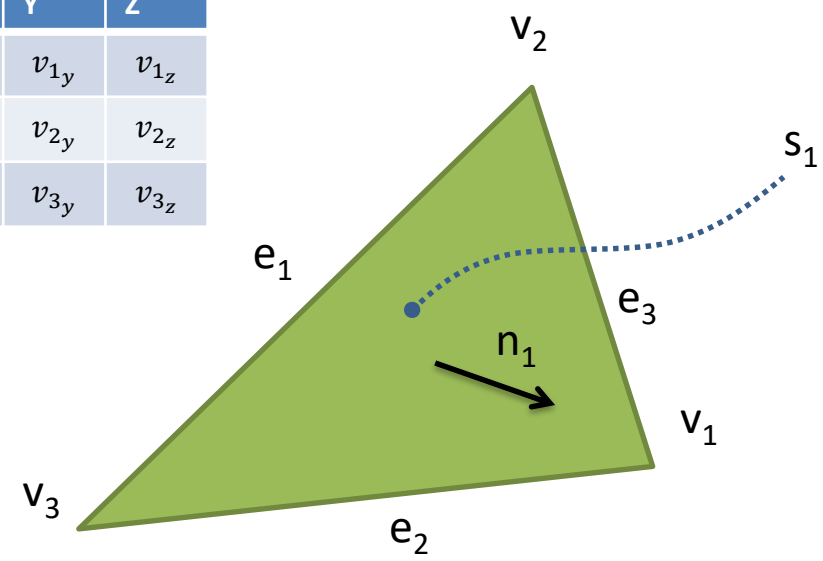
ID	Kante 1	Kante 2	Kante 3
1	1	2	3

Kanten

ID	Knoten 1	Knoten 2
1	2	3
2	3	1
3	1	2

Knoten

ID	X	Y	Z
1	v_{1_x}	v_{1_y}	v_{1_z}
2	v_{2_x}	v_{2_y}	v_{2_z}
3	v_{3_x}	v_{3_y}	v_{3_z}



Vergleich der Datenstrukturen

	Vorteile	Nachteile
Knotenliste	• Trennung von Geometrie und Netztopologie • Geringer Speicherbedarf	• Kanten werden mehrmals definiert • Suche nach Polygonen, die eine Kante enthalten ineffizient
Kantenliste	• Trennung von Geometrie und Netztopologie • Schnelle Bestimmung von Randkanten (Kanten mit nur einem Verweis auf Polygon	• Suche nach Polygonen, die einen Vertex enthalten ineffizient

Winged Edge Datenstruktur

- Zusätzlich zu Kantenliste: Zeiger auf ankommende/abgehende Kanten
- Ermöglicht effizientere Abfragen, z.B. welche Polygone zu einer Kante gehören

Polygone

ID	Knoten 1	Knoten 2	Knoten 3
1	1	2	3

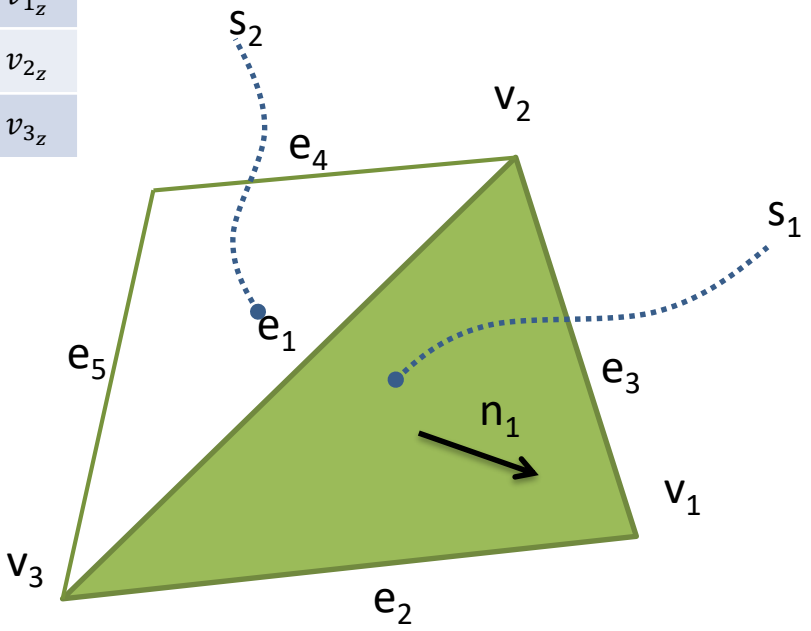
Knoten

ID	X	Y	Z
1	v_{1_x}	v_{1_y}	v_{1_z}
2	v_{2_x}	v_{2_y}	v_{2_z}
3	v_{3_x}	v_{3_y}	v_{3_z}

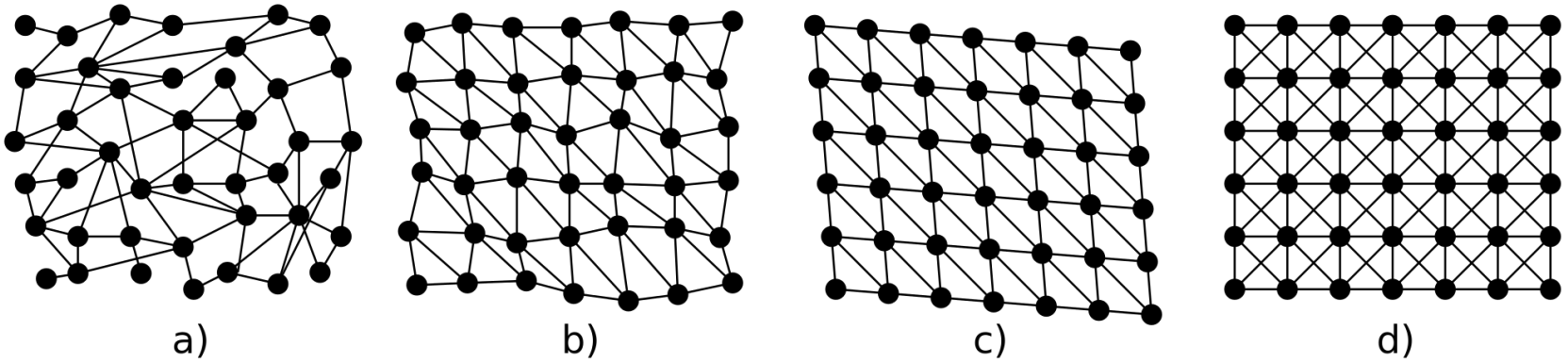
Kanten

ID	Vertex Start	Vertex Ende	Polygon links	Polygon rechts	Linke Traverse, vorher	Linke Traverse, nach	Rechte Traverse, vorh.	Rechte Traverse, nach
1	2	3	1	2	3	2	4	5
2	3	1	1	…	1	3	…	…
3	1	2	1	…	2	1	…	…

Beispiel: Dreiecksnetz



Eigenschaften von Polygonnetzen

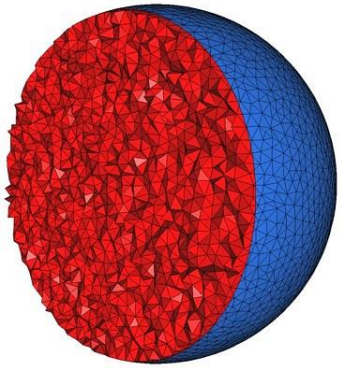


a) Keine besonderen Eigenschaften
b) Strukturiertes Polygonnetz
c) Strukturiertes, reguläres Polygonnetz
d) Strukturiertes, reguläres, orthogonales Polygonnetz

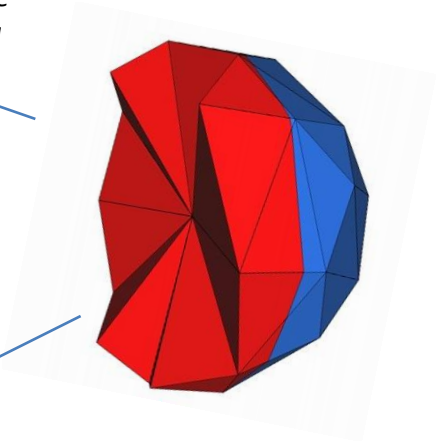
https://de.wikipedia.org/wiki/Polygonnetz

Eigenschaften von Polygonnetzen

- Nicht-adaptive Polygonnetze
  - Global gleiche Auflösung

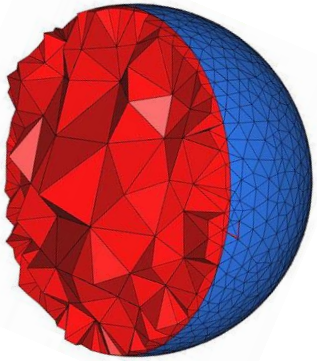


Nicht-adaptive Verfeinerung



- Adaptive Polygonnetze
  - Lokale Verfeinerung der Auflösung

adaptive Verfeinerung



http://doc.cgal.org/latest/Mesh_3/

Polygonisierung

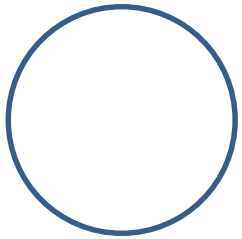
- Polygonisierung (Meshing) = Prozess der Berechnung einer Repräsentation einer gegebenen Oberfläche durch einfache Oberflächenpolygone (meist Dreiecke).

- Oberfläche in ***impliziter*** Darstellung: Lösung einer Gleichung, z.B. Kugelgleichung

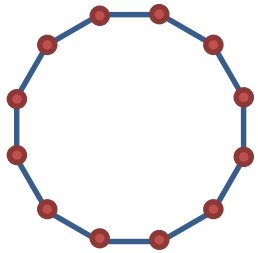
  $$f\colon r^2 = x^2 + y^2 + z^2$$

- Extraktion aus Daten, z.B. Oberflächenscan, Iso-Fläche aus Volumendaten

- Für die Computergrafik meist ***explizite*** Darstellung einer Oberfläche notwendig → Polygonisierung („Meshing“)
- Annahme für Algorithmen: glatte Oberfläche, keine Singularitäten



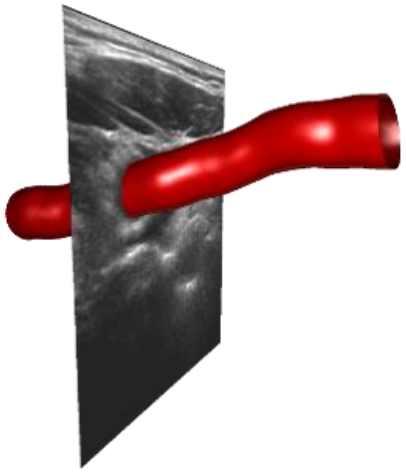
Implizite Darstellung
(analytische Gleichung)

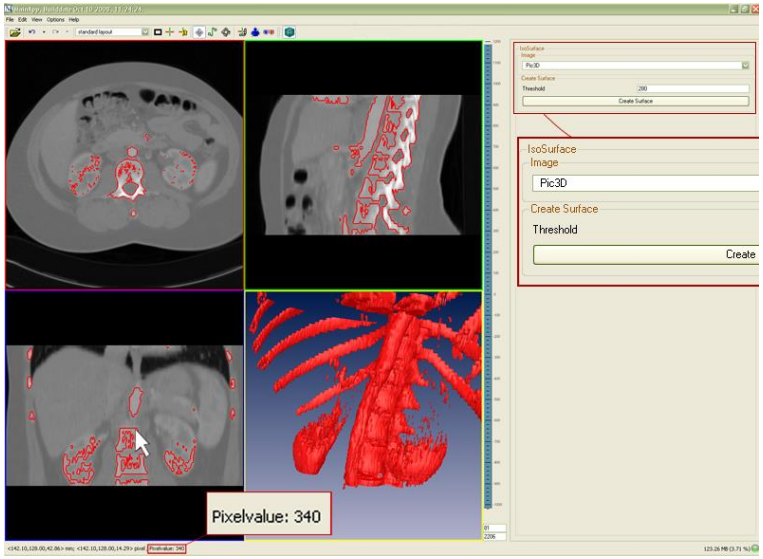


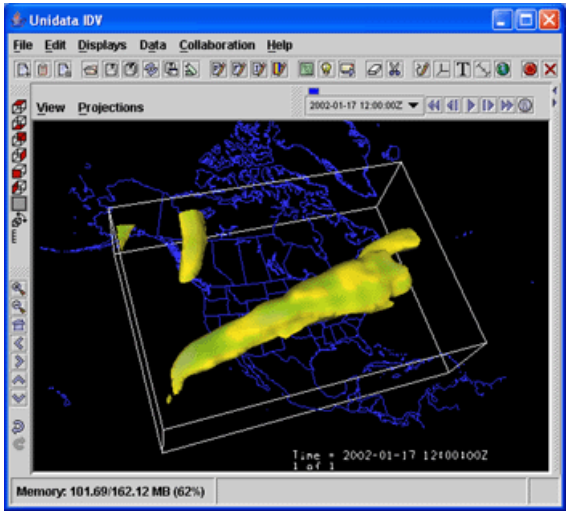
Explizite Darstellung
(Knoten, Kanten, …)

Isoflächen

- Flächen, die im Raum benachbarte Punkte gleicher Merkmale oder Werte miteinander verbinden.
- Oft verwendet um aus Bildern/Volumendaten Oberflächen zu extrahieren,
  - z.B. Organe in der Medizin
  - z.B. Meteorologie um Gebiete gleicher Eigenschaften räumlich darzustellen



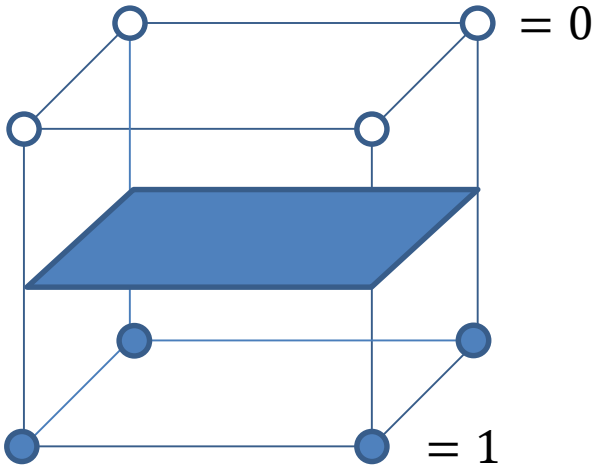




http://www.imfusion.de/products/imfusion-suite
http://docs.mitk.org/2014.10/IsoSurfaceGUI.png
http://www.unidata.ucar.edu/software/idv/docs/userguide/examples/3DSurface.html

Marching Cubes Algorithmus

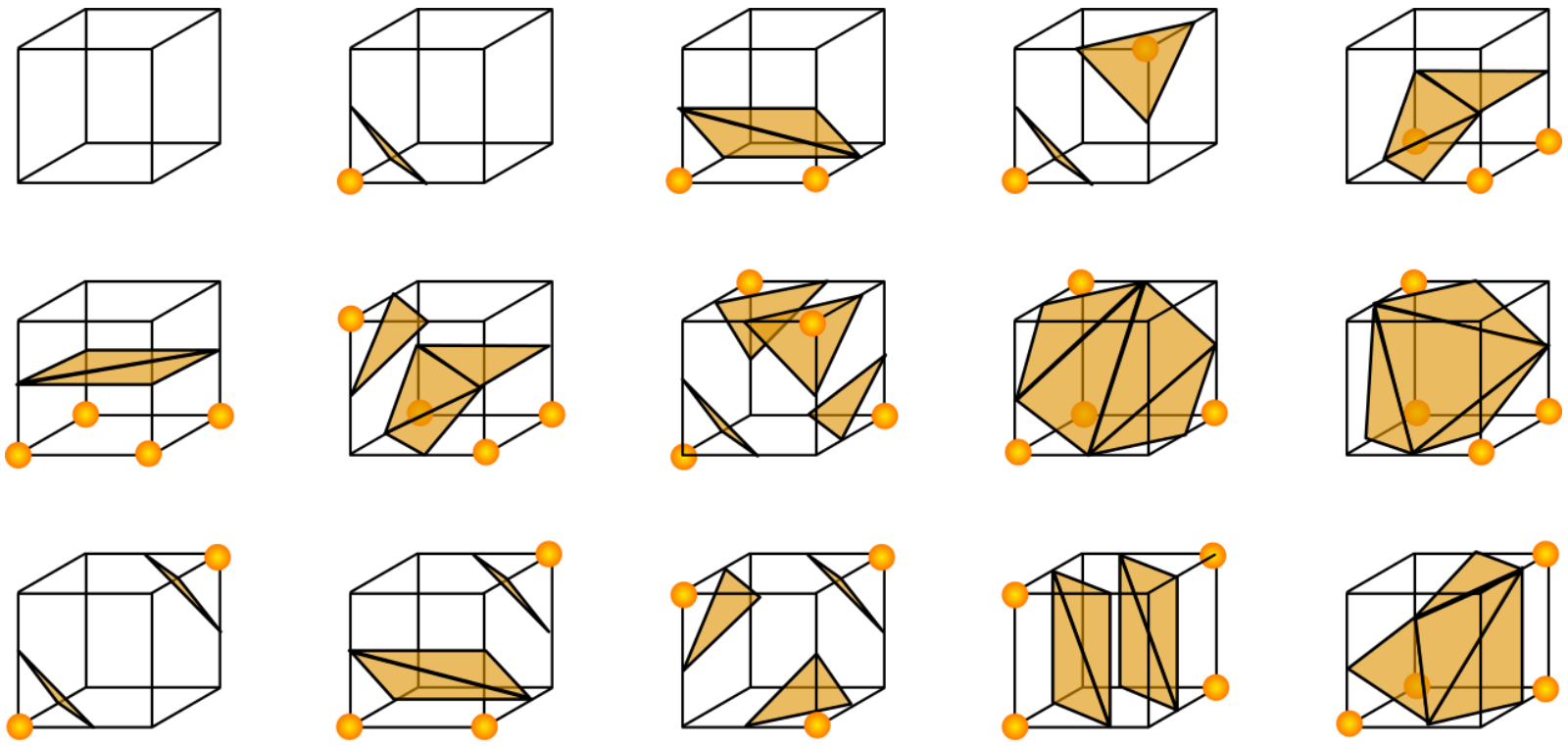
- Oft verwendet bei Erzeugung von Polygonnetzen aus Volumendaten: Annäherung einer Isofläche mit Polygonen (meist Dreiecke).
- Grundidee:
  - Unterteilung des Raums in kleine Würfel (*Cubes*)
  - Für jeden Würfel: Schnitt mit Objektoberfläche (Isofläche) bestimmen (*lokales Meshing*)
- Umsetzung für Meshgenerierung aus Voxeldaten
  - Jeder Knoten v des Würfels liegt auf einem Voxel des Volumendatensatzes
  - Voxel-Grauwert I + Schwellwert T bestimmt ob Knoten innerhalb oder außerhalb des Objektes liegt. Zuordnung eines Wertes an jedem Knoten:



$$0 \quad \text{falls } I(v) < T$$
$$1 \quad \text{falls } I(v) > T$$

Marching Cubes Algorithmus

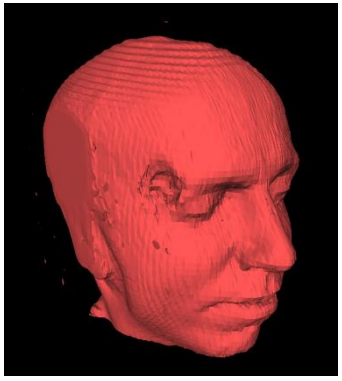
- 15 mögliche Würfelklassen entsprechend der Verteilung der Werte an den Knoten



https://de.wikipedia.org/wiki/Marching_Cubes

Marching Cubes Algorithmus

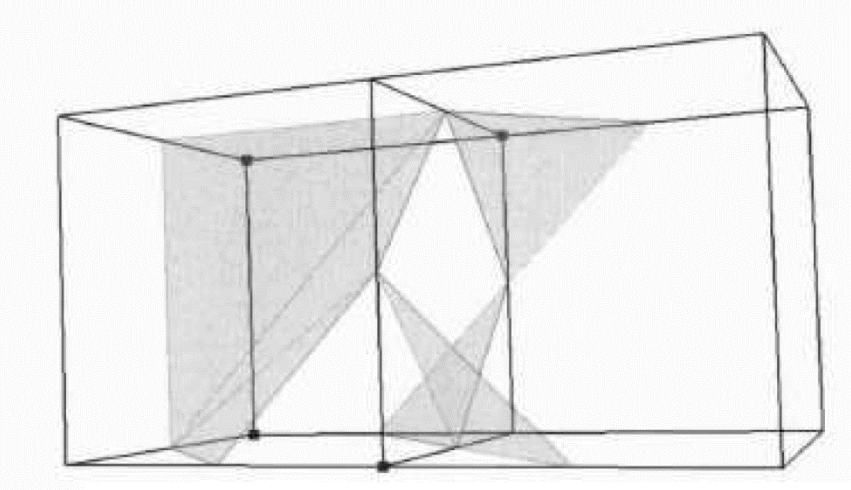
- Knoten des Polygonnetzes = Schnittpunkte der Oberfläche mit den Würfelkanten
  - Bestimmt durch lineare Interpolation der Grauwerte der zwei Vertices einer Würfelkante
- Bestimmung der Normalen der Oberfläche:
  - Schätzen des Grauwertgradienten an den Knoten des Würfels
  - Interpolation des Gradienten am berechneten Knoten des Polygonnetzes
- Zusammensetzen der Oberflächen aller Würfel ergibt gesamtes Polygonnetz des Objektes



https://de.wikipedia.org/wiki/Marching_Cubes

Marching Cubes Algorithmus

- Vorteile:
  - Schnelle Implementierung über Lookup-Table
- Nachteile:
  - Löcher in der Oberfläche, Mehrdeutigkeitsprobleme
  - Hohe Anzahl Polygone
  - Probleme bei spitzen Details an Oberflächen



G.M. Treece, R.W. Prager, A.H. Gee, “Regularised marching tetrahedra: improved iso-surface extraction”; Computers & Graphics Volume 23, Issue 4, Pages 583–598, August 1999

Triangulation

- = Teilung einer Oberfläche in Dreiecke

- **Polygon-Triangulation**
  - Unterteilung gegebener Polygone in Dreiecke (= *Tesselation*)
  - Algorithmen i.d.R. abhängig von Eigenschaften des Polygons

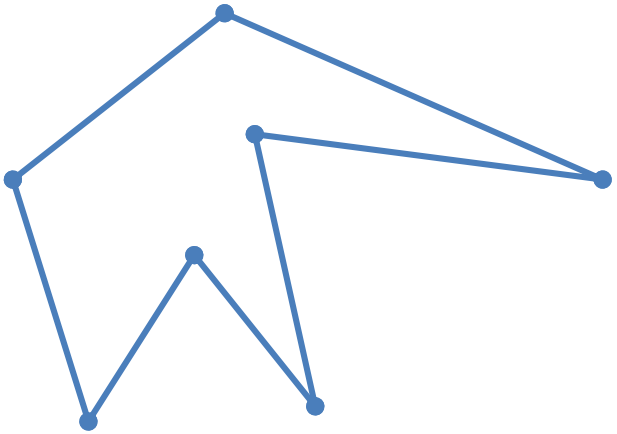
- **Punktwolken-Triangulation**
  - Triangulation komplexer Geometrien (z.B. aus Oberflächenscan)
  - Verbinden der Vertices zu Dreiecken

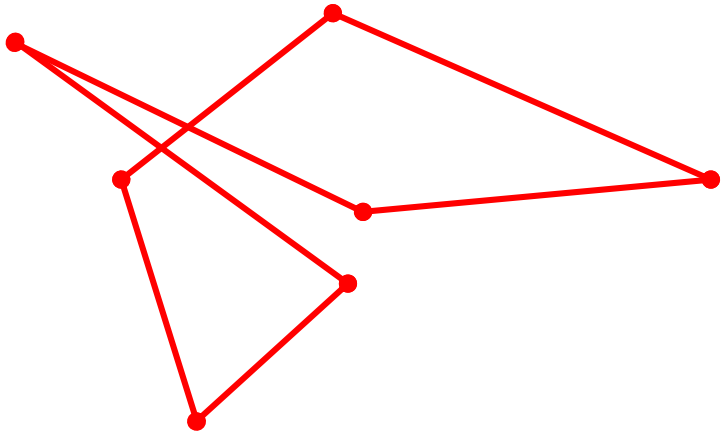
- Mehr oder weniger Einschränkung der Eigenschaften und Meshqualität
  - z.B. Maximierung des Innenwinkels des Dreiecks

Polygontriangulation

- **Triangulations-Theorem:**
  - Jedes *einfache* Polygon hat eine Triangulation
  - Jede Triangulation eines n-gons besteht aus genau $n - 2$ Dreiecken

- Ein *einfaches* Polygon ist ein geschlossener Kantenzug der sich nicht selbst schneidet.

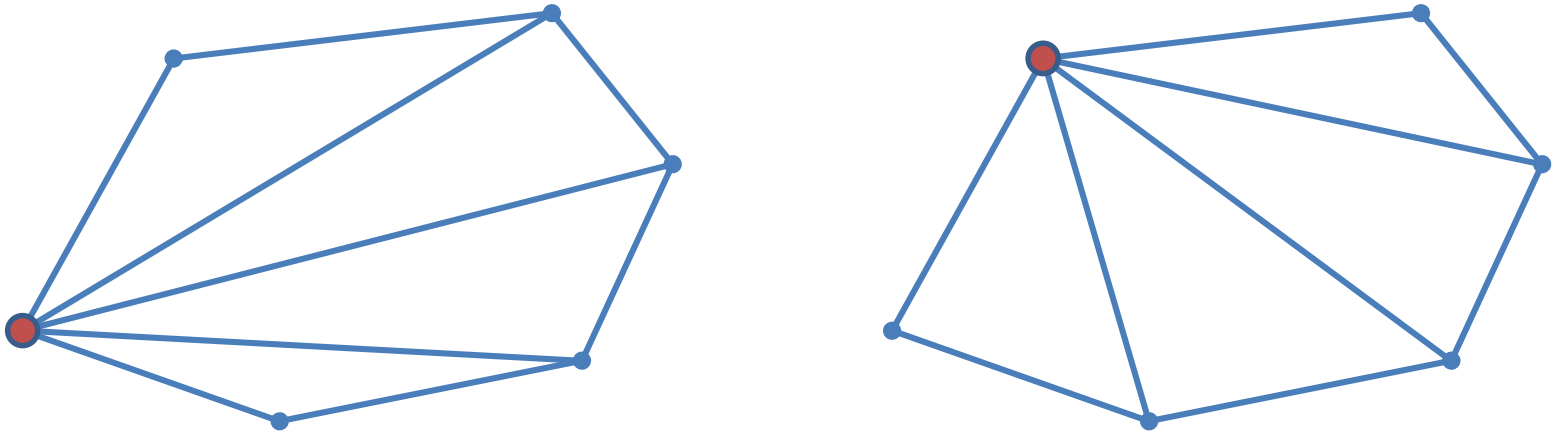




- Jedes konvexe Polygon ist ein einfaches Polygon.

Polygontriangulation: konvexe Polygone

- trivial, siehe **`GL_TRIANGLE_FAN`**
- Verbinden eines Vertex mit allen anderen

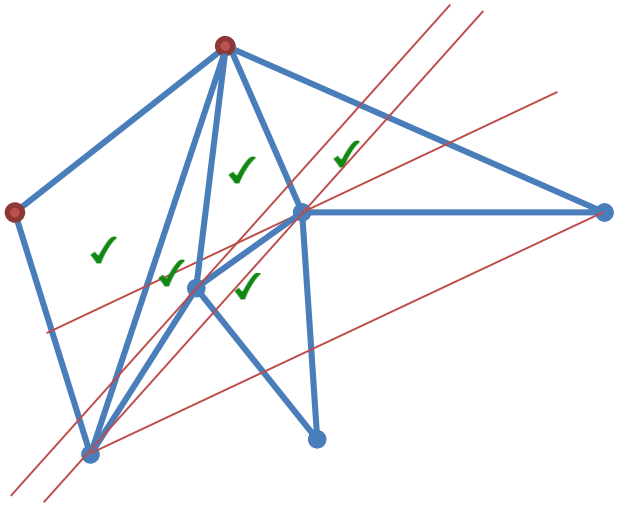


- Startvertex bestimmt u.U. Ergebnis

Polygontriangulation: einfache Polygone

- Naiver Algorithmus für einfache Polygone: Bruteforce Diagonalen-Suche:
  - Durchlaufen aller Vertices des Polygons entlang der Polygonkante
  - Vorhergehenden und nachfolgenden Vertex verbinden
  - Rekursives Vorgehen für entstehende Teil-Polygone

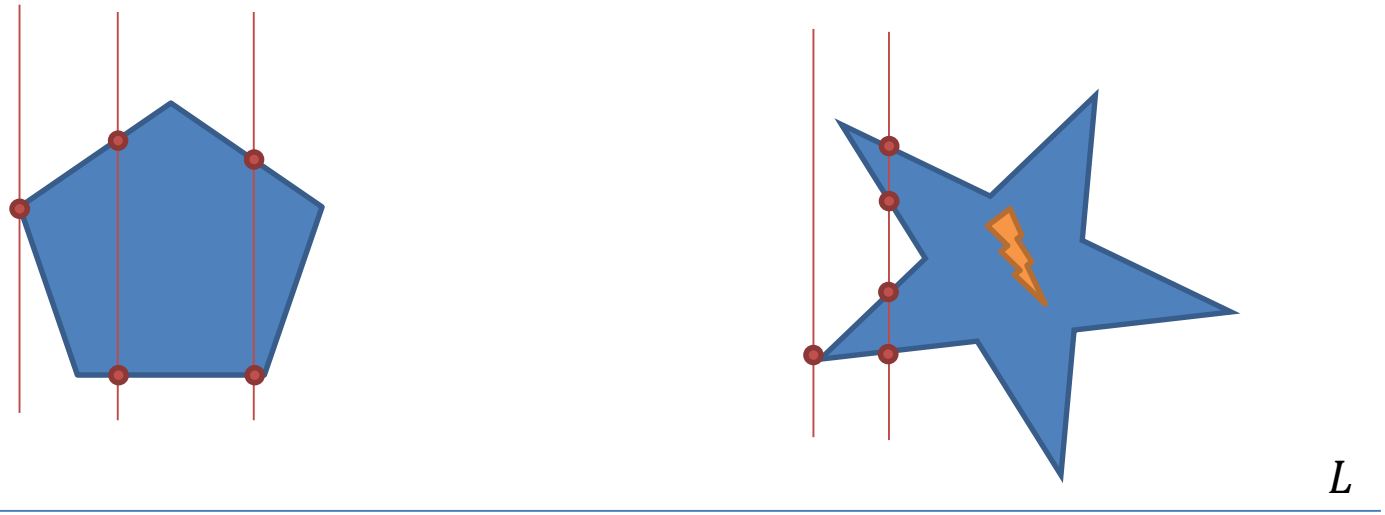
Beginn an linker Ecke



  - ***Achtung:*** Befindet sich ein anderer Vertex im entstandenen Dreieck: Finde nächste parallele Gerade zu dieser Kante durch einen der Vertices im Dreieck
  - Aktueller Startvertex: Position halten bis Vertex nicht mehr Teil eines weiter teilbaren Polygons ist.

Monotonie von Polygonen

- Ein planares Polygon P ist monoton bzgl. einer Geraden L wenn jede Senkrechte zu L das Polygon maximal zweimal schneidet.



- Jedes konvexe Polygon ist auch monoton
- Jedes Polygon, das monoton zu jeder beliebigen Geraden L ist, ist auch konvex

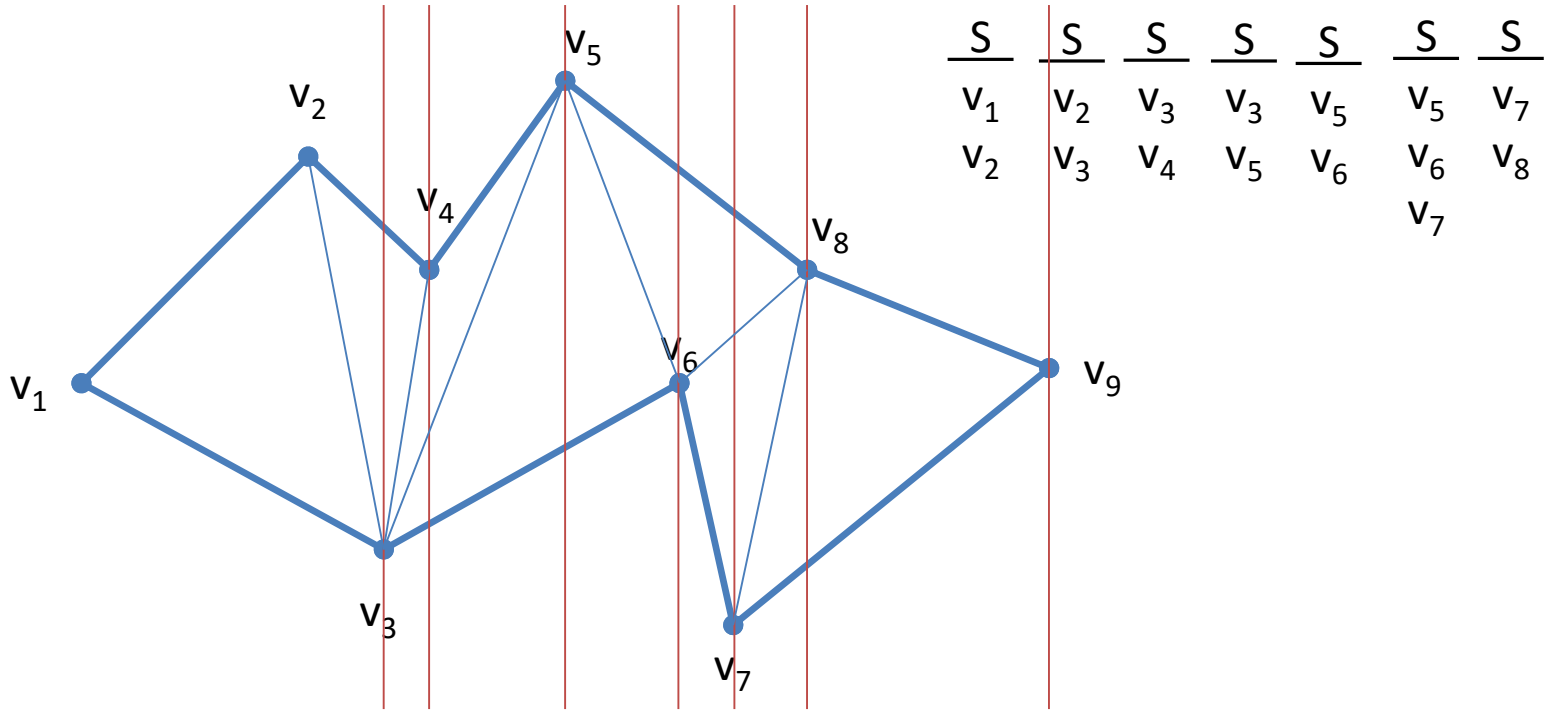
Polygontriangulation: monotone Polygone

- Sweep-Line-Algorithmus: am Beispiel x-monotones Polygon
- Initialisierung:
  - Sortiere Vertices nach x-Koordinate (1. Kriterium) → $v_1 \dots v_n$
  - Erzeuge Stack S mit nicht-bearbeiteten Punkten: v_1 (S.top-1) und v_2 (S.top)
- Pseudo-Code:

```
FOR i=3 bis n
    IF v_i auf anderer Seite als S.top
        Kante von v_i zu Punkten in S bis auf Untersten
        Entferne alle Punkte aus S
        Lege v_i-1 und v_i auf S
    ELSE
        WHILE S.top-1 nicht für v_i von S.top verdeckt wird
            Erstelle Kante von v_i nach S.top-1 und entferne S.top
        Lege v_i auf S

Füge Kante von v_n zu Punkten in S bis auf Obersten und Untersten ein
```

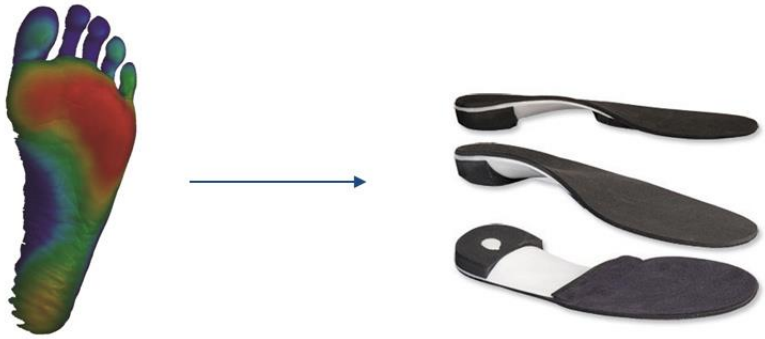
Polygontriangulation: monotone Polygone

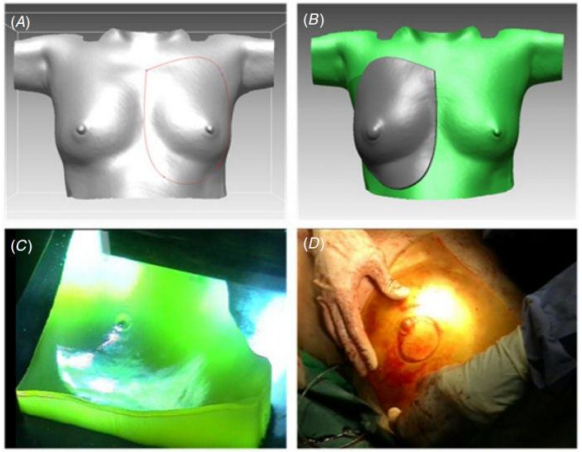
Beispiel:



Punktwolkengenerierung

- Z.B. Automatisiert durch 3D Scanner: Abtastung der Oberfläche an diskreten Punkten
- Z.B. Manuelle oder automatische Extraktion aus Bilddaten



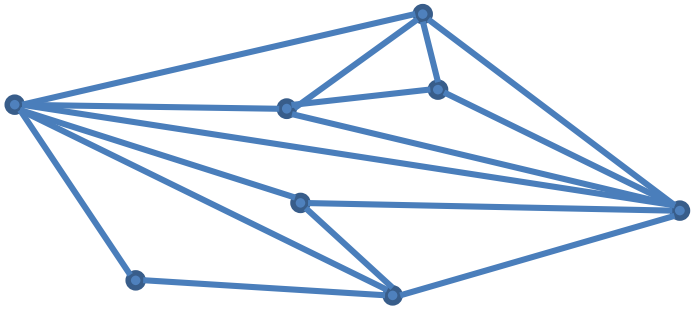


http://www.lmi3d.com/blog/medical-applications-3d-scanning
http://medicalphysicsweb.org/cws/article/research/47264
http://gispoint.de/news-einzelansicht/1395-neue-wege-der-datenerfassung-im-baubereich.html

Punktwolkentriangulation

Triangle Splitting

- Bilden der konvexen Hülle aus der Punktmenge
- Triviale Triangulation des konvexen Polygons
- Für jeden inneren Punkt: Einfügen von Kanten zu den Vertices des umgebenden Dreiecks (in 3D Tetraeder!)

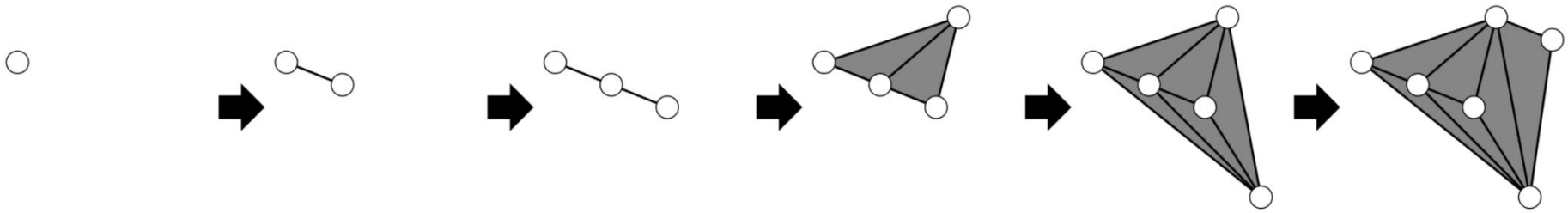


- 3D Punktwolken: Evtl. Identifizierung der Oberfläche aus dem resultierenden 3D Mesh!
- Vorsicht bei nicht-konvexen Punktwolken: schlecht-gestelltes Problem

Punktwolkentriangulation

Incremental construction

- Sortiere Vertices nach Ihrer x-Koordinate
- Die ersten drei Punkte bilden ein Dreieck
- Verbinde jeden weiteren Punkt mit den für ihn „sichtbaren“ bisherigen Punkten
- *Sichtbarkeit*: die neue Kante darf kein bisheriges Dreieck schneiden

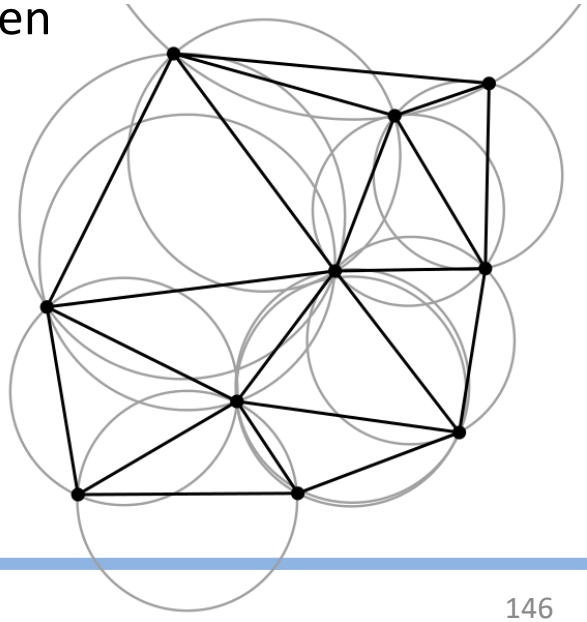


http://www.cse.ust.hk/~scheng/book/Delmesh/chapter2.pdf

Delaunay Triangulation

- Für Interpolationen - z.B. bei der Beleuchtung - sind Dreiecksnetze mit möglichst großen Innenwinkeln besser geeignet.
- Delaunay Triangulation maximiert den minimalen Winkel in einem Dreieck
- Benannt nach russischem Mathematiker Boris Nikolajewitsch Delone (1890–1980, franz. Form des Nachnamens: Delaunay)

- **Grundprinzip**: Der Umkreis jedes Dreiecks des Netzes darf keine weiteren Vertices der vorgegebenen Verticemenge enthalten
  - In 3D: Umkugel-Bedingung für Tetraeder-Generierung



Delaunay Triangulation

- Test ob ein Punkt v im Umkreis eines Dreiecks abc liegt (2D):

  Berechnung der Determinante $\det$ der Matrix D

$$\det(D) = \begin{vmatrix} a_x & a_y & {a_x}^2 + {a_y}^2 & 1 \\ b_x & b_y & {b_x}^2 + {b_y}^2 & 1 \\ c_x & c_y & {c_x}^2 + {c_y}^2 & 1 \\ v_x & v_y & {v_x}^2 + {v_y}^2 & 1 \end{vmatrix}$$

- $\det(D) < 0$: v liegt außerhalb des Umkreises des Dreiecks abc
- $\det(D) = 0$: v liegt auf dem Umkreises des Dreiecks abc
- $\det(D) > 0$: v liegt innerhalb des Umkreises des Dreiecks abc

Delaunay Triangulation

- Konstruktion – Dualität mit **Voronoi-Diagrammen**:
  - *Gegeben*: eine Menge M von n Vertices.
  - Das Voronoi-Diagramm von M zerlegt die Ebene in n disjunkte Gebiete (sogenannte Voronoi-Zellen)
  - Die Voronoi-Zelle V eines Vertex v enthält genau einen Vertex aus M sowie alle geometrischen Punkte w, die näher an v als an jedem anderen Vertex v' liegen:

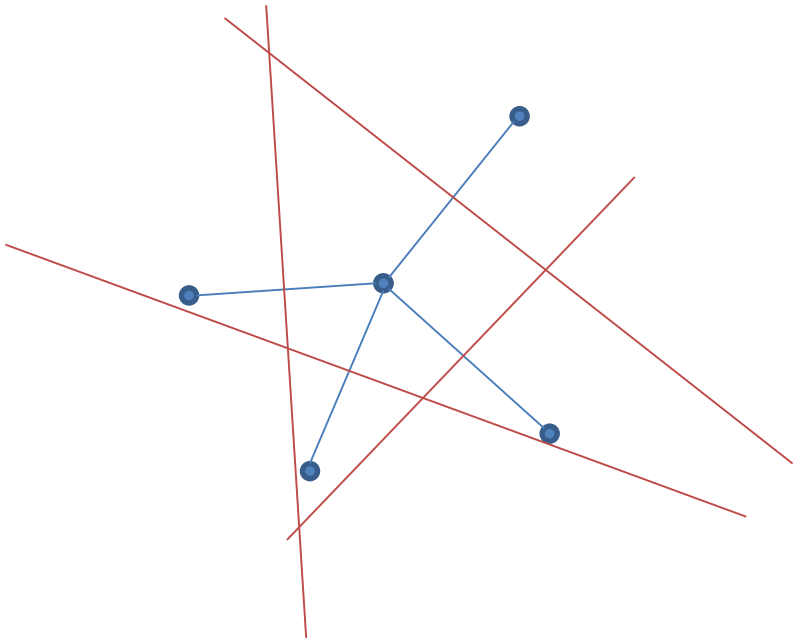
$$V(v) = \{w \in \mathbb{R}^2 : \forall v' \in M \backslash \{v\} : dist(w, v) < dist(w, v')\}$$

  - Naive Konstruktion: Bilden von Halbebenen h zwischen den Vertices v und v':

$$h(v, v') = \{w \in \mathbb{R}^2 : dist(w, v) < dist(w, v')\}$$

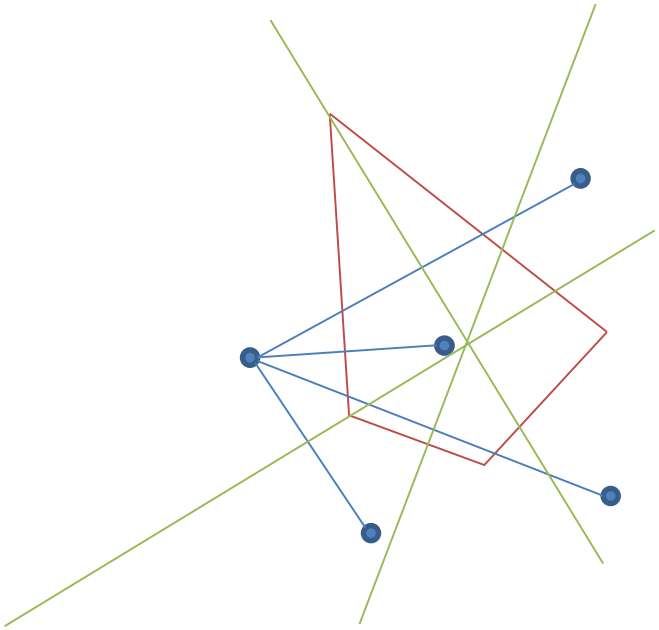
Delaunay Triangulation

- Konstruktion eines Voronoi-Diagramms



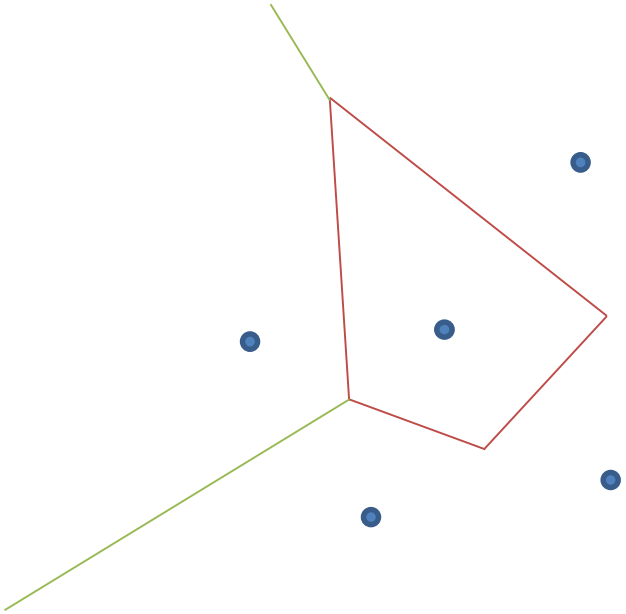
Delaunay Triangulation

- Konstruktion eines Voronoi-Diagramms



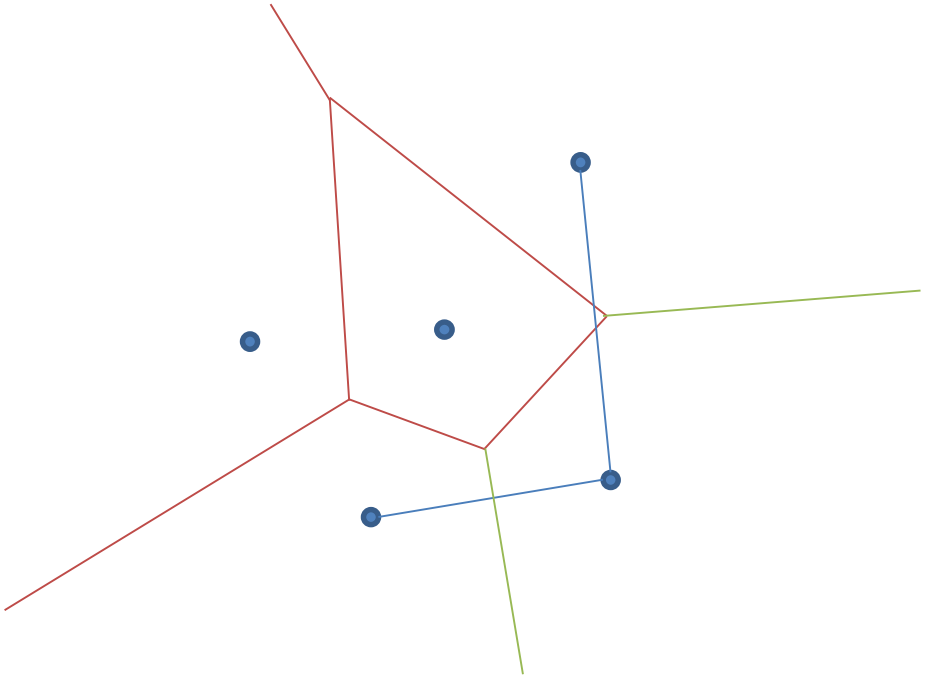
Delaunay Triangulation

- Konstruktion eines Voronoi-Diagramms



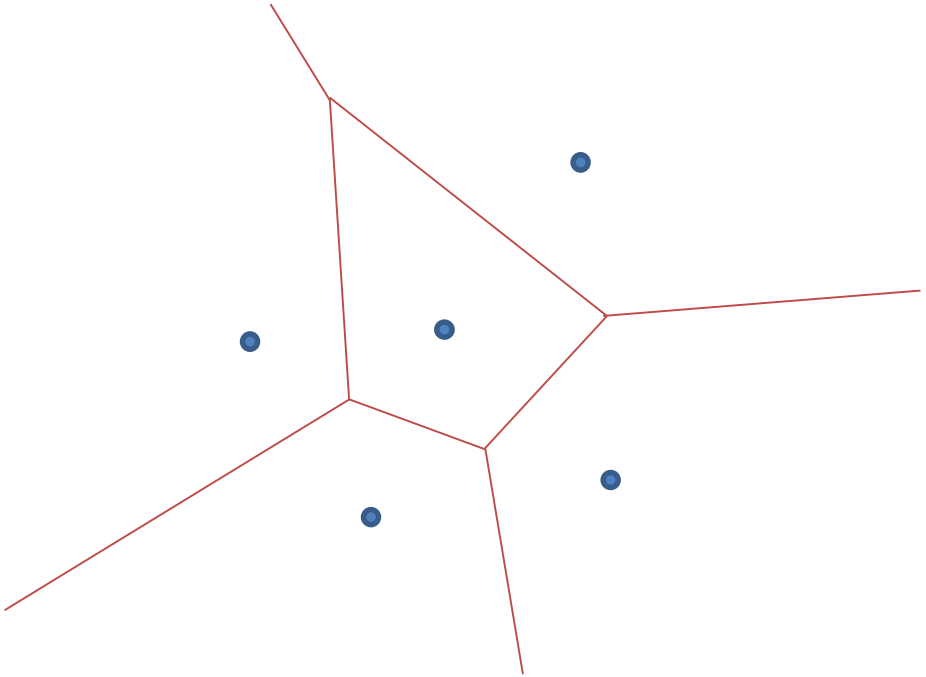
Delaunay Triangulation

- Konstruktion eines Voronoi-Diagramms



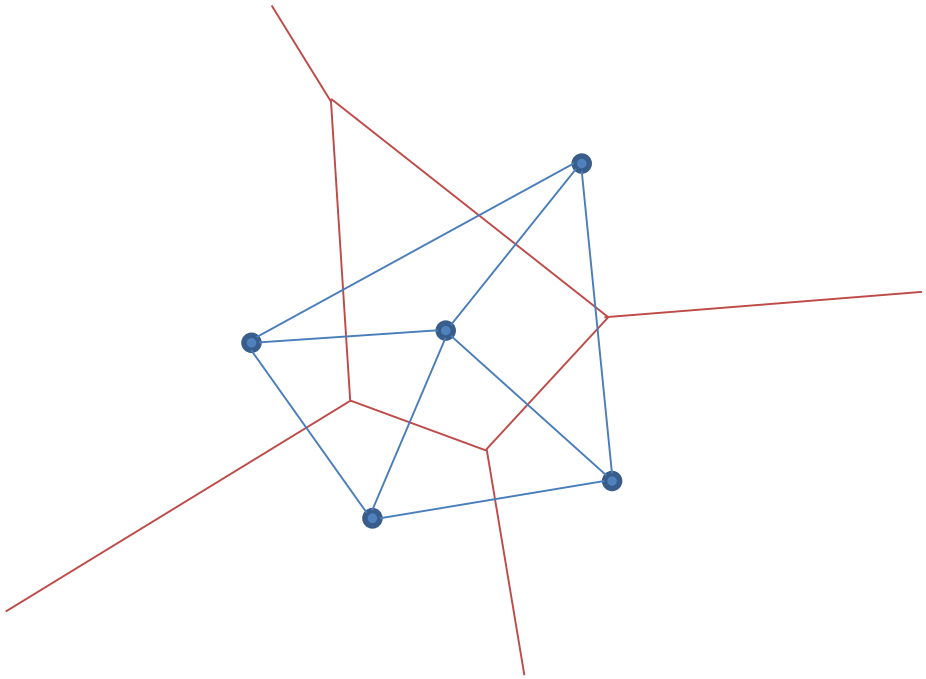
Delaunay Triangulation

- Konstruktion eines Voronoi-Diagramms



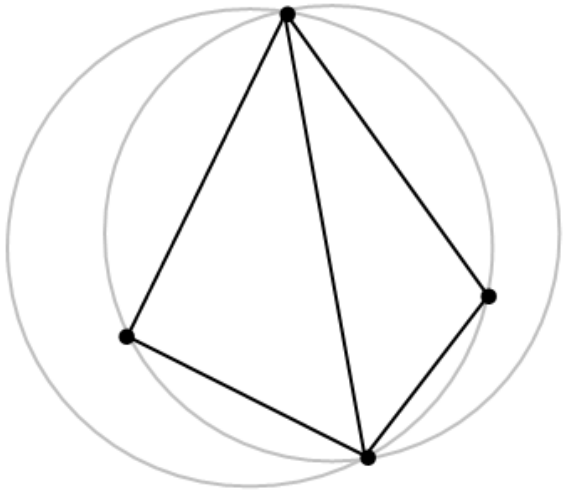
Delaunay Triangulation

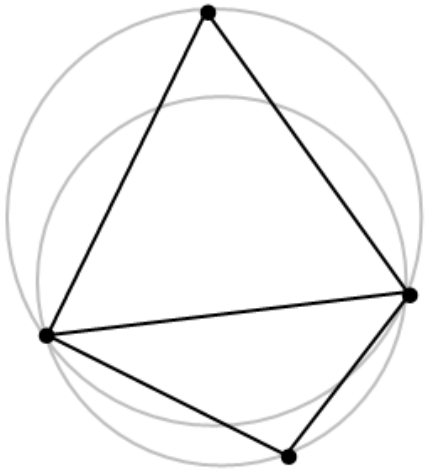
- Die Delaunay Triangulation einer Punktmenge ist der duale Graph eines Voronoi-Diagramms
- Konstruktion durch orthogonale Linie zu jeder Voronoi-Kante



Delaunay Triangulation

- Zahlreiche Algorithmen zur Berechnung der Delaunay Triangulation aus einer Punktmenge vorhanden
- **Edge Flipping**
  - Erzeugen eines beliebigen Dreiecksnetzes
  - Für jedes Dreieck: prüfen ob der Umkreis einen weiteren Punkt einschließt, der Teil eines angrenzenden Dreiecks ist.
  - Ist dies der Fall, wird ein Flip der gemeinsamen Kante durchgeführt.





https://de.wikipedia.org/wiki/Delaunay-Triangulation

Delaunay Triangulation

- **Inkrementelle Methode:**
  - Start: initiales Dreiecksnetz, das alle zu erwartenden Vertices einschließt
  - Einfügen eines beliebigen neuen Vertex: Suche des Dreiecks, das den Vertex enthält
  - Neuer Punkt wird mit den drei Vertices des gefundenen Dreiecks verbunden

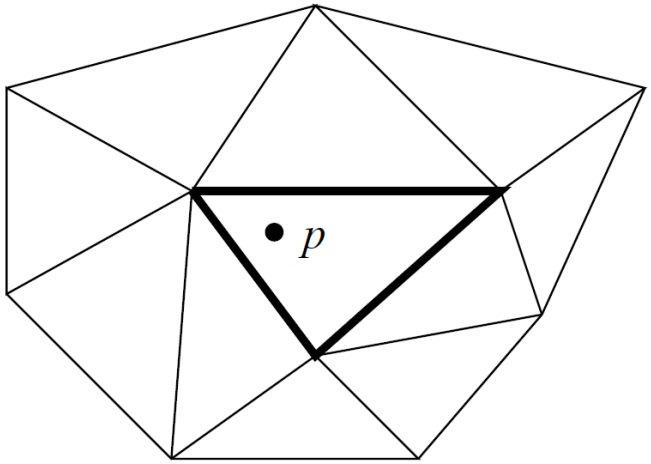
    → es entstehen drei neue Dreiecke, die nicht mehr unbedingt die Umkreisbedingung erfüllen
  - Test jedes neuen Dreiecks auf Umkreisbedingung
  - Korrekturen der Umkreisbedingung mit Flip-Algorithmus
  - Nach jeder Korrektur gibt es möglicherweise Dreiecke die die Umkreisbedingung nicht mehr erfüllen: iteratives Vorgehen.

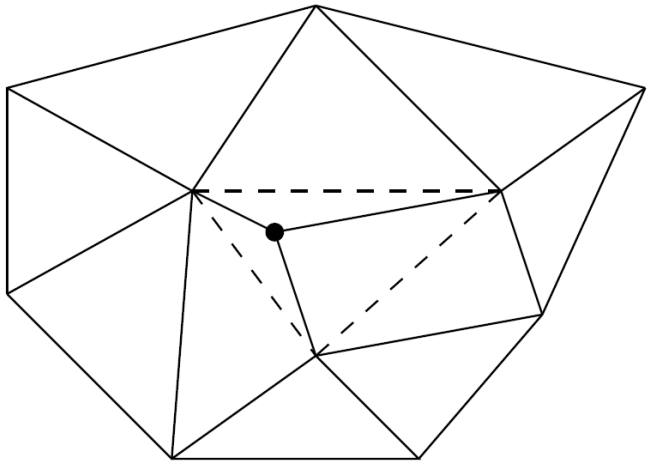
- **Modifikation der inkrementellen Methode**
  - Pro Schritt: Anfügen eines benachbarten Dreiecks (statt eines beliebigen Dreiecks bei der inkrementellen Methode)

- **Ableitung aus Voronoi-Graph**

Delaunay Triangulation

Beispiel: Inkrementelle Methode

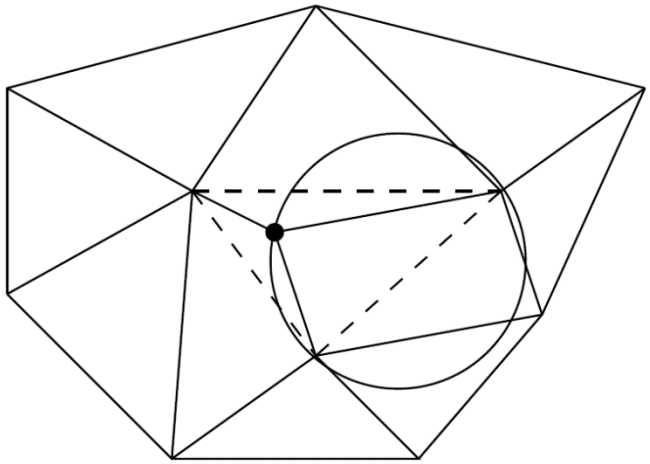


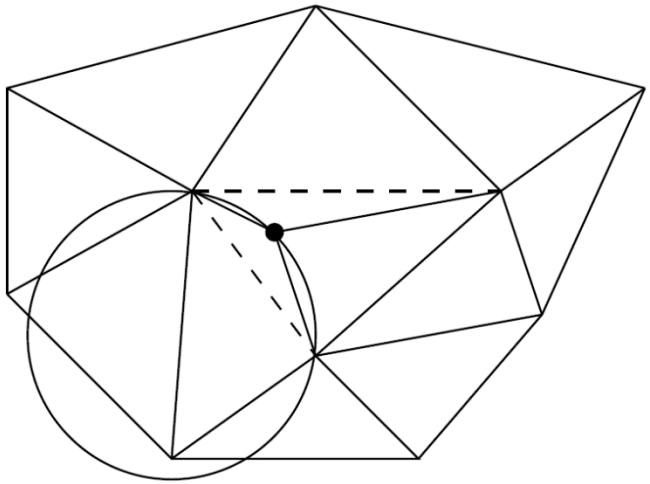


D. Lischinski: Incremental Delaunay Triangulation, in: P.S. Heckbert: „Graphics Gems“, Academic Press, 1994

Delaunay Triangulation

Beispiel: Inkrementelle Methode

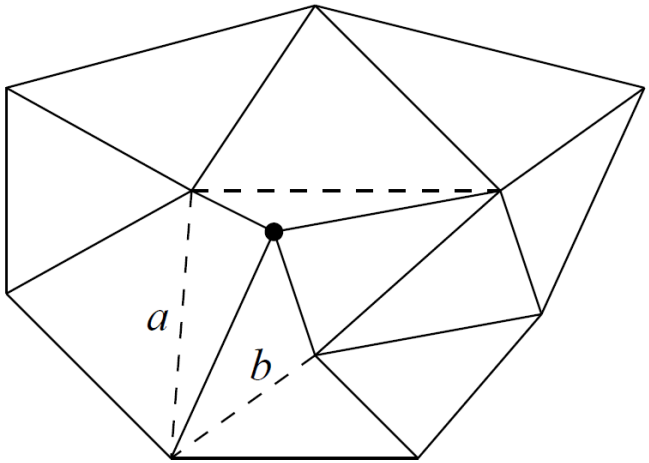


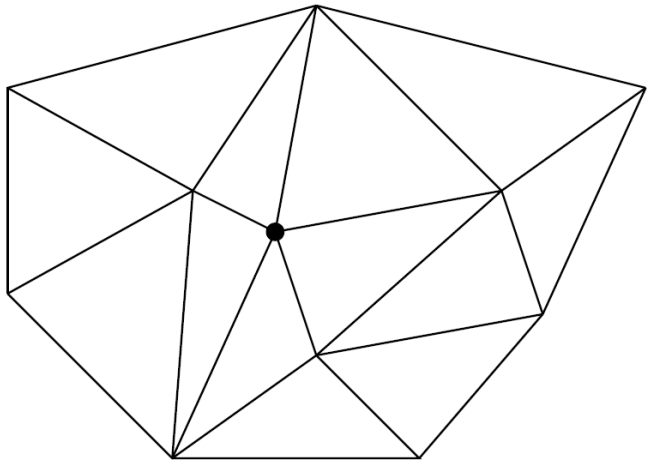


D. Lischinski: Incremental Delaunay Triangulation, in: P.S. Heckbert: „Graphics Gems“, Academic Press, 1994

Delaunay Triangulation

Beispiel: Inkrementelle Methode

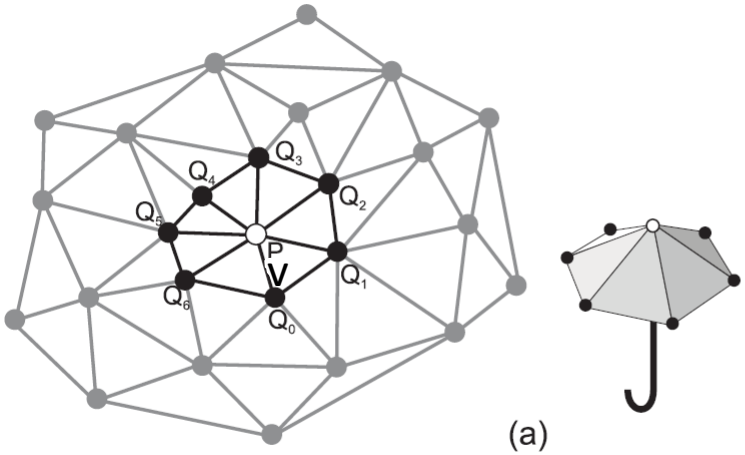




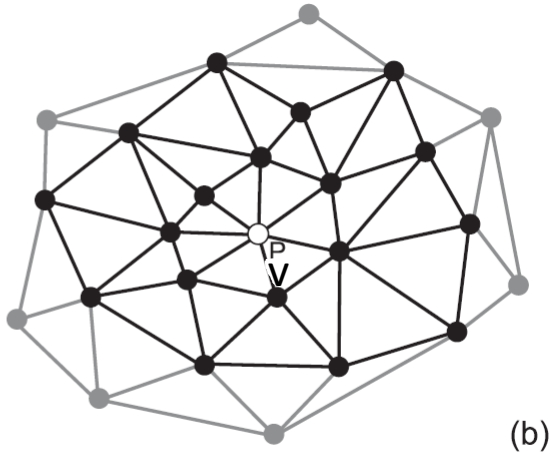
D. Lischinski: Incremental Delaunay Triangulation, in: P.S. Heckbert: „Graphics Gems“, Academic Press, 1994

Mesh-Glättung

- Bei Generierung aus Volumendaten oft Ausreißer, Spikes, etc.
- Glättung im Allgemeinen durch Filter realisiert (siehe Bildverarbeitung)
- Zu filternde Region über *Umbrella*-Operator definiert (= Nachbarschafts-Operator)
  - Umbrella-Region 1. Grades: Alle Knoten q_i die mit v eine verbindende Kante haben
- Auswirkung der Glättung meist nur auf Knotenposition, Topologie bleibt erhalten



Region 1. Grades



Region 2. Grades

J. Haase: „Glättung von Polygonnetzen in medizinischen Visualisierungen“, Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, 2005

Mesh-Glättung

- **Laplace-Filter:** Verschiebung des zentralen Knotens v in das Zentrum seiner Nachbarn:

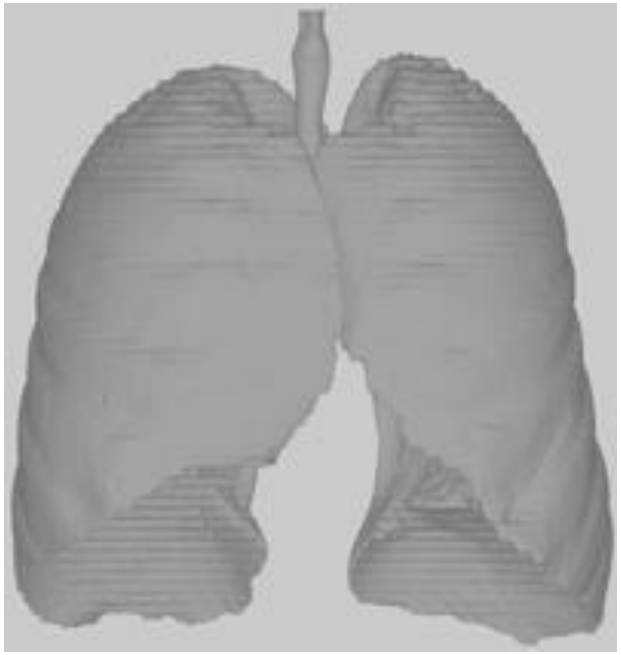
$$v' = \frac{1}{n}\sum_{i=0}^{n-1} q_i \qquad \forall v \in M$$

- Iteratives Vorgehen: Glättungsgrad kann durch Anzahl der Iterationen eingestellt werden
- Nachteil:
  - Polygonnetze schrumpfen
  - Invertierte Elemente möglich
- Abwandlung: Einführung eines Relaxationswertes λ

$$v' = v + \frac{\lambda}{n}\sum_{i=0}^{n-1} (q_i - v) \qquad \forall v \in M, 0 \leq \lambda \leq 1$$

Mesh-Glättung

- Laplace-Glättung



- Weitere Varianten der Laplace-Glättung:
  - Z.B. HC-Algorithmus: Iterative Vor- und Zurückverschiebung

https://www.cg.tuwien.ac.at/courses/Visualisierung2/HallOfFame/2012/4-Cornel/Homepage/index.html

Mesh-Glättung

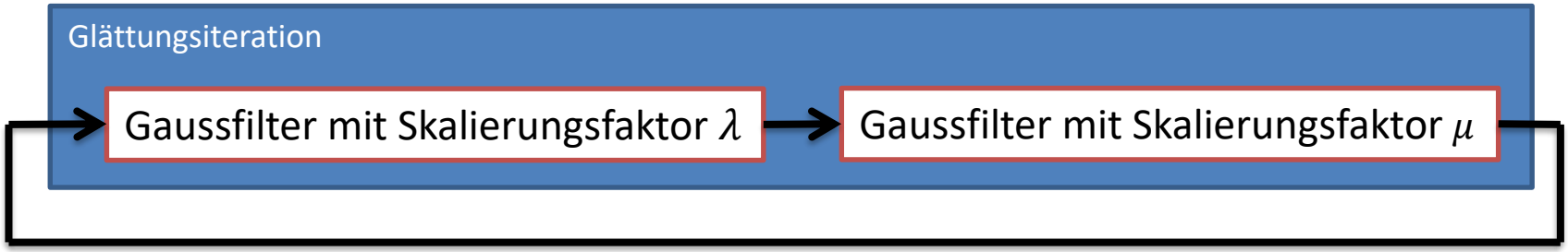
- **Gauß-Filter:** Knotenpositionen werden aus gewichtetem Mittel der Umbrella-Region 1. Grades berechnet:

$$v' = v + \lambda \sum_{i=0}^{n-1} w_i (q_i - v) \qquad , \forall v \in M, 0 \leq \lambda \leq 1$$

- Iteratives Vorgehen
- Wichtung w kann pro Iteration verschieden eingestellt werden.
  - Summe der Wichtungen w_i ergibt immer 1.
  - Gebräuchlich: $w_i = \frac{1}{n}$ oder Nachbarschaftsstrukturen mit einbeziehen (Distanzfunktion)
- Skalierungsfaktor λ analog zu Relaxationswert bei Laplace-Glättung
- Im Allgemeinen sehr ähnliche Ergebnisse wie Laplace-Filterung
  - Gut zur Unterdrückung von Rauschen (kleinen Artefakten)
  - Neigt ebenfalls zum Schrumpfen

Mesh-Glättung

- **Lowpass-Filter:** zweifache Ausführung der Gauss-Filterung mit unterschiedlichen Skalierungsfaktoren
  - Einführung eines zweiten Skalierungsfaktors μ mit negativem Wert
  - μ erfüllt die Bedingung $0 < \lambda < -\mu$
  - μ sollte einen „geringfügig“ größeren Wert als λ haben.

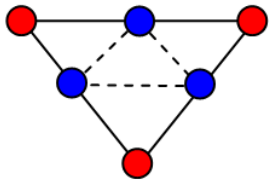


„Schrumpfen“ „Aufblähen“

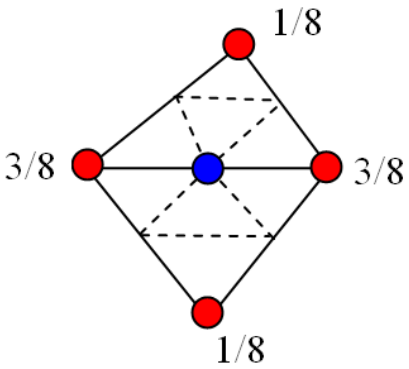
- Verringert Schrumpfen des Meshs
- Erhält Details besser als Laplace- und Gauss-Filterung
- Benötigt mehr Iterationen um optisch glatte Netze zu erzeugen

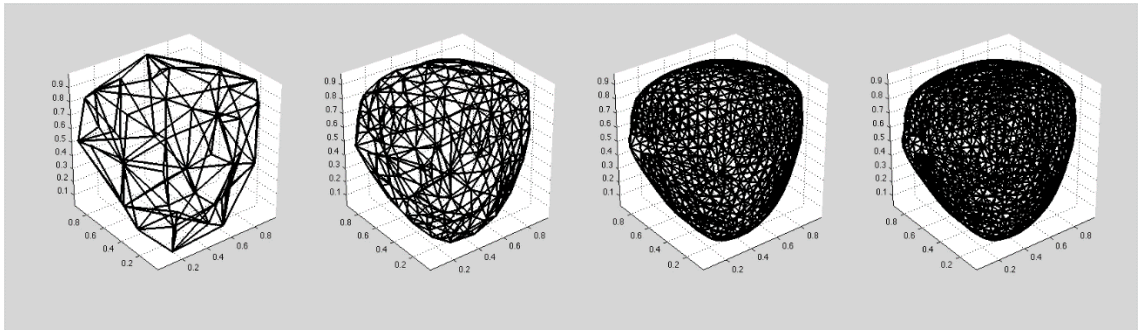
Mesh-Glättung

- **Mesh Subdivision:** Verfeinerung des Mesh durch neue Knoten, gleichzeitige Verschiebung der Knoten (Topologie-Änderung!)
- Zwei Verarbeitungsschritte:
  1) Topologische Unterteilung
  2) Geometrische Positionierung



- Beispiel: Loop Subdivision
  - Topologische Unterteilung: Dreieck in 4 kleinere Dreiecke zerlegen durch initiales Einfügen von Knoten in der Mitte der Kanten (=Edge Vertices)
  - Geometrische Positionierung:
    - Edge Vertices: Lineare Kombination der benachbarten Knoten
    - Ursprüngliche Vertices: Verschiebung anhand Umbrella-Region.





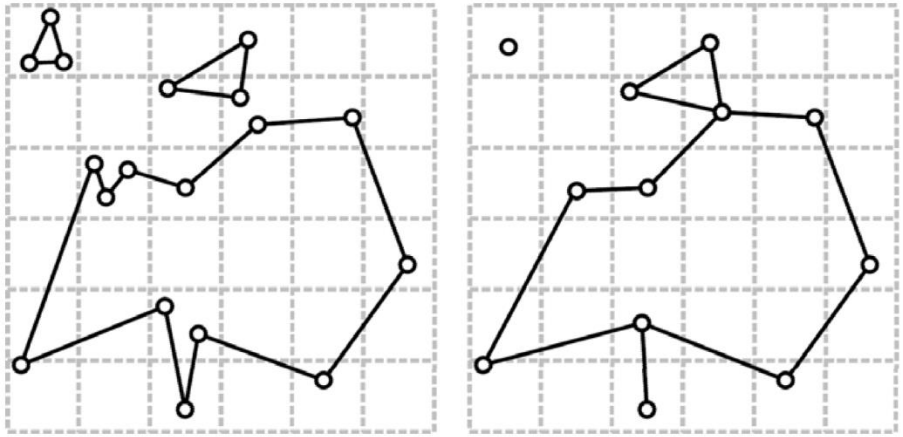
https://graphics.stanford.edu/~mdfisher/subdivision.html
http://www.mathworks.com/matlabcentral/fileexchange/32727-fast-loop-mesh-subdivision

Mesh-Dezimierung

- Oft resultiert eine große Polygon-Anzahl aus dem Meshing
- Mesh-Dezimierung versucht die Polygon-Anzahl zu verringern, die Geometrie jedoch so weit wie möglich zu erhalten

Vertex Clustering

- Abbildung des Meshs in ein gleichmäßiges Gitter
- Für Knoten die innerhalb einer Gitterzelle liegen: repräsentativen Knoten bestimmen (z.B. Mittelwert)
- Verbindung der repräsentativen Knoten über Gitterzellengrenzen hinweg zu neuen Kanten (sofern es dort zuvor eine Kante gab)

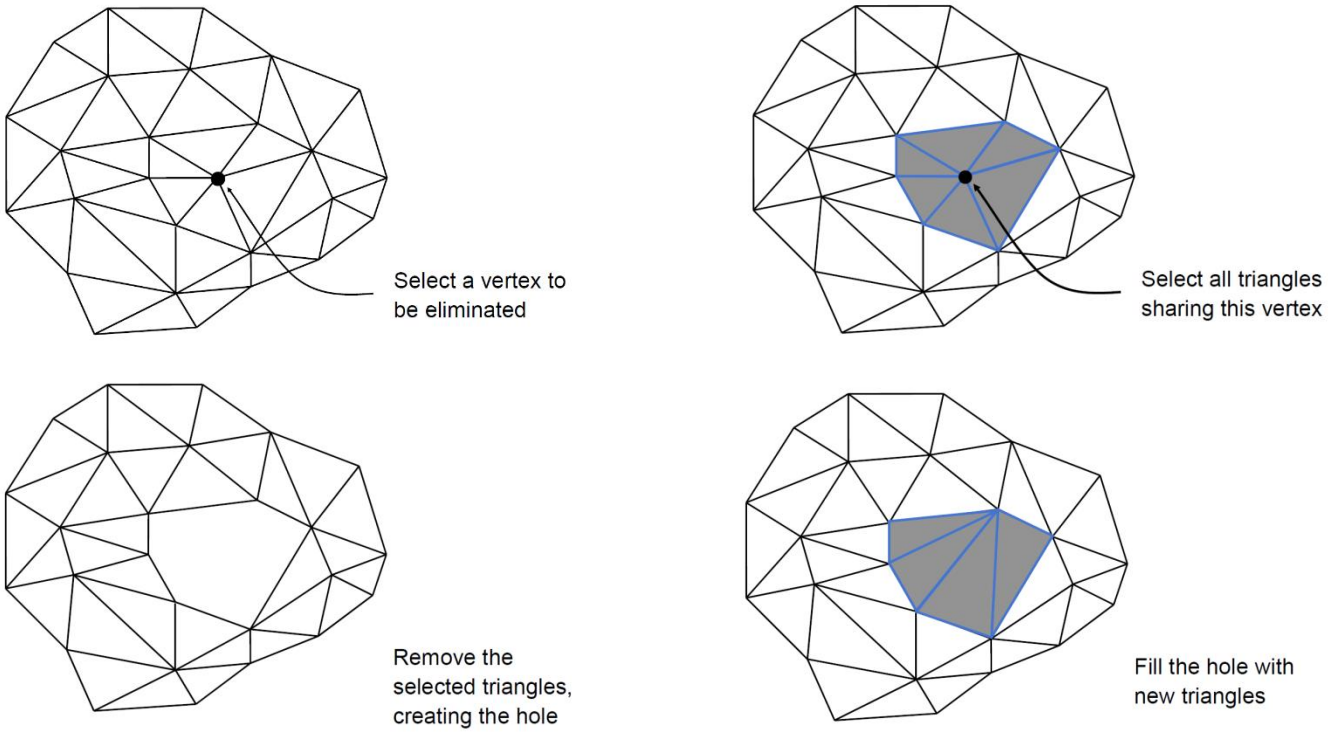


http://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/08_Simplification.pdf

Mesh-Dezimierung

Incremental Decimation:

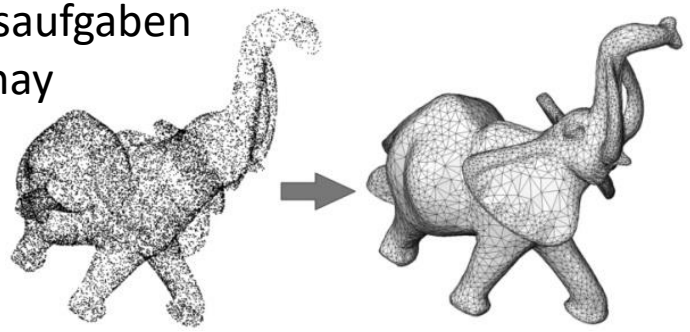
- Grundidee: Auswahl einer Mesh-Region, z.B. Umbrella-Region. Darin Anwendung eines Dezimierungs-Operators
- Beispiel-Operator „Vertex removal“:



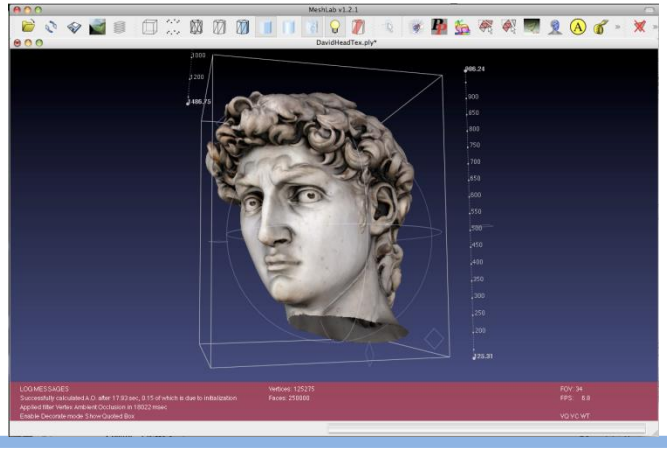
http://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/08_Simplification.pdf

Softwarepakete für Meshing (Beispiele)

- CGAL: The Computational Geometry Algorithms Library
  - Open Source Bibliothek (C++) für geometrische Berechnungen
  - Enthält diverse Algorithmen für Polygonisierungsaufgaben (u.a. 2D / 3D Surface Mesh basierend auf Delaunay Triangulation)
  - Sehr gutes Manual: http://doc.cgal.org/latest/Manual/index.html

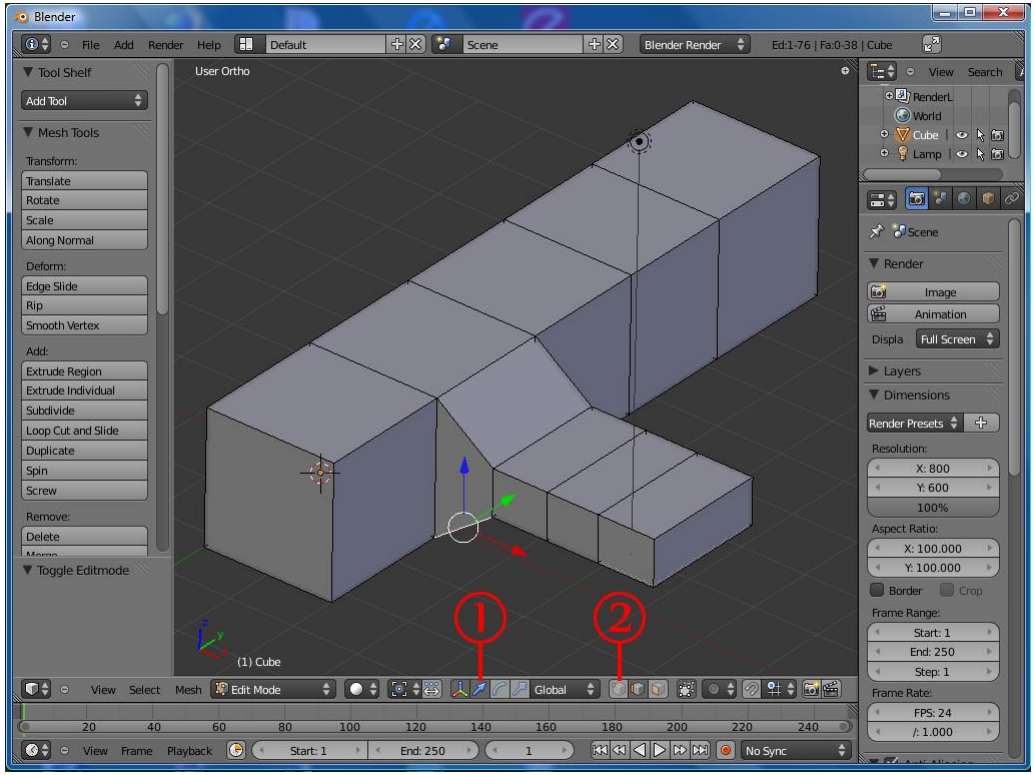


- MeshLab
  - Open Source zur Erstellung und Bearbeitung von 3D-Dreieck-Polygonnetzen
  - http://meshlab.sourceforge.net/
  - Grafische Benutzeroberfläche



Meshgenerierung durch CAD

- Erzeugung komplexer Körper in OpenGL ohne Datengrundlage schwierig
- Meist Modellierung in entsprechender CAD-Anwendung oder 3D Grafikwerkzeugen, z.B. Blender

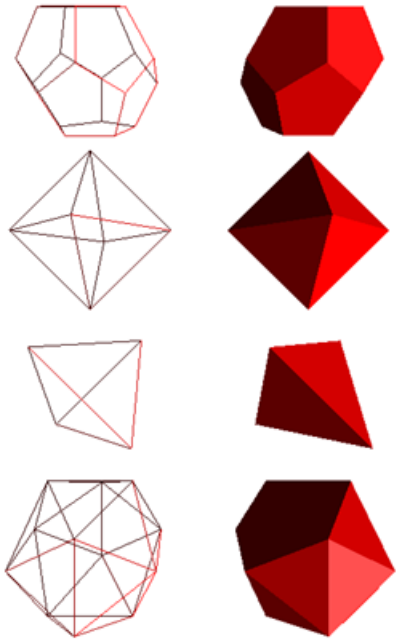


Geometrische Körper in GLUT

- Würfel:
  - **`glutWireCube(size):`** Würfel-Drahtmodell mit Seitenlänge *size*
  - **`glutSolidCube(size):`** Gefüllter Würfel, Seitenlänge size
- Kugel:
  - **`glutWireSphere(radius, slices, stacks):`** Kugel mit Radius *radius*, Mittelpunkt im Ursprung, Anzahl der Längengrade (*slices*) und Breitengrade (*stacks*).
  - **`glutSolidSphere(radius, slices, stacks):`** analog
- Kegel:
  - **`glutWireCone(base, height, slices, stacks):`** Kegel mit Grundradius *radius*, Grundfläche liegt in der x/y-Ebene mit Mittelpunkt im Ursprung, Höhe des Kegels in z-Richtung (*height*). Anzahl der Längengrade (*slices*) und Breitengrade (*stacks*).
  - **`glutSolidCone(base, height, slices, stacks):`** analog
- Torus:
  - **`glutWireTorus(innerRadius, outerRadius, nsides, rings):`** Torus mit Mittelpunkt im Ursprung und z-Achse als Achse. Zahl der Seiten in jedem Radialschnitt *nsides* und Zahl der Radialschnitte (*rings*).
  - **`glutSolidTorus(innerRadius, outerRadius, nsides, rings):`** analog

Geometrische Körper in GLUT

- Platonische Polyeder, (jeweils Mittelpunkt im Ursprung)
  - Dodekaeder:
    - **`glutWireDodecahedron(void)`:** Radius $\sqrt{3}$
    - **`glutSolidDodecahedron(void)`**
  - Oktaeder:
    - **`glutWireOctahedron(void)`:** Radius 1
    - **`glutSolidOctahedron(void)`**
  - Tetraeder:
    - **`glutWireTetrahedron(void)`:** Radius $\sqrt{3}$
    - **`glutSolidTetrahedron(void)`**
  - Ikosaeder:
    - **`glutWireIcosahedron(void)`:** Radius 1
    - **`glutSolidIcosahedron(void)`**



- Teekanne
  - **`glutWireTeapot(scale)`**: klassische Teekanne, um Faktor *scale* gestreckt
  - **`glutSolidTeapot(scale)`**: analog

http://www.programming-techniques.com/2012/05/rendering-dodecahedron-octahedron.html

3.3. FREIFORMFLÄCHEN

Freiformkurven- und Flächen

- Polygonnetze aus explizit definierten planaren Polygonen approximieren gekrümmte Flächen i.d.R. durch eine Vielzahl von Polygonen
- Andere Möglichkeit für eine speichersparende Repräsentation: parametrische Beschreibung gekrümmter Flächen ➔ „Freiformflächen“
- Vorteile:
  - Ermöglicht Arbeiten mit unterschiedlichen Auflösungen → Tessellierung je nach Bedarf.
  - Normalenvektoren für Lichtberechnung können direkt aus gekrümmter Fläche abgeleitet werden.

- Gewünschte Eigenschaften:
  - **Kontrollierbarkeit**: Einfluss der Parameter für Benutzer intuitiv verständlich
  - **Lokalitätsprinzip**: Möglichkeit die Kurve/Fläche lokal zu verändern.
  - **Glattheit**: Kurve/Fläche soll gewisse Glattheitseigenschaften besitzen
    - Stetig (=keine Lücken/Sprünge) (C0-Kontinuität)
    - mindestens einmal stetig differenzierbar (= keine Knicke) (C1-Kontinuität)
    - besser zweimal stetig differenzierbar (= keine abrupte Änderung der Krümmung) (C2-Kontinuität)

Freiformkurven- und Flächen

- Beschreibung erfolgt mittels **parametrischer Funktionen**
  - Punkte (x, y, z) auf der Kurve werden als Funktion einer oder mehrerer Variablen durchlaufen:

$$\begin{aligned} x &= x(t) \\ y &= y(t) \\ z &= z(t) \end{aligned} \qquad t_1 < t < t_2$$

  - Für jeden Wert t aus $[t_1, t_n]$ erhält man eine kartesische Koordinate (x, y, z) eines Punktes
  - Menge der Punkte legt die Funktionskurve der durch den Parameter t dargestellten Funktion fest.
  - Beschreibung einer Kurve im Raum: 1 Parameter *(t)*
  - Beschreibung einer Fläche im Raum: 2 Parameter *(t,s)*

- Beispiel: parametrische Beschreibung eines Kreises

$$r(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \cos t \\ \sin t \end{pmatrix} \qquad \text{für } 0 \leq t \leq 2\pi$$

Freiformkurven- und Flächen

- Stützpunkte definieren Kurven- bzw. Flächenverlauf
  - *Interpolation*: Kurve läuft exakt durch Stützpunkte

    für $(n+1)$ Stützpunkte existiert immer eine Zerlegung in ein Polynom n-ten Grades

    Nachteile: - oft hoher Polynomgrad → hoher Rechenaufwand
    - Lokalitätsprinzip verletzt
    - Schwingen

  - *Approximation*: Kurve nähert Stützpunkte gut an

    → gewünschte Eigenschaften können erhalten werden

- Bekannte Ansätze für Freiformkurven/-flächen u.a.:
  - Bézierkurven/-flächen
  - B-Splines
  - Non-uniform rational B-Splines (NURBS)

Bézierkurven

- Grundbaustein: Bernstein-Polynome
- Besondere Familie reeller Polynome mit ganzzahligen Koeffizienten

Das i-te Bernstein-Polynom n-ten Grades (mit $i \in \{0, \dots, n\}$) ist durch die Gleichung

$$B_i^{(n)}(t) = \binom{n}{i} \cdot (1-t)^{n-i} \cdot t^i$$

gegeben, wobei $t \in [0,1]$.

Erinnerung aus der Kombinatorik (Binomialkoeffizient):

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!} = \frac{\prod_{k=1}^{n} k}{\prod_{k=1}^{i} k \cdot \prod_{k=1}^{(n-i)} k}$$

- Eigenschaften von Bernstein-Polynomen:
  - Im Definitionsbereich $[0, 1]$ nehmen Bernstein-Polynome nur Werte zwischen 0 und 1 an.
  - An jeder Stelle des Definitionsbereichs $[0, 1]$ addieren sich die Bernstein-Polynome zu 1 auf.

Bézierkurven

- **Beispiel für $n = 3$:** einsetzen von $i = 0 \ldots 3$ und $n = 3$ liefert die 4 Bernsteinpolynome

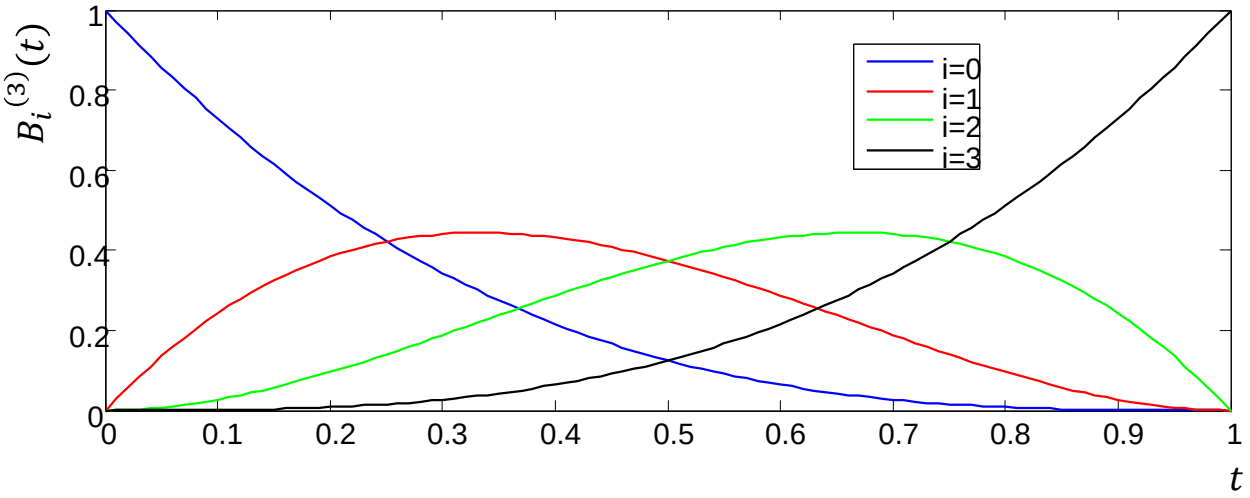
$$B_0^{(3)}(t) = (1-t)^3$$

$$B_1^{(3)}(t) = 3(1-t)^2 t$$

$$B_2^{(3)}(t) = 3(1-t)t^2$$

$$B_3^{(3)}(t) = t^3$$

$$B_i^{(n)}(t) = \binom{n}{i} \cdot (1-t)^{n-i} \cdot t^i$$



Bézierkurven

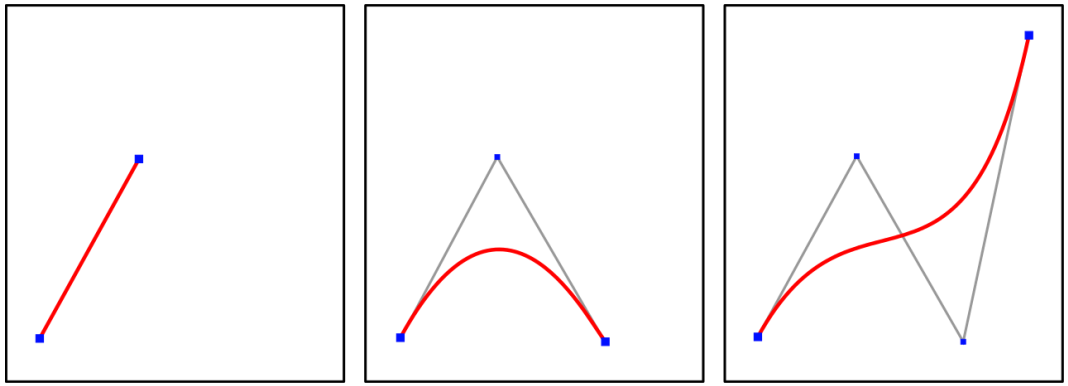
- Bézierkurven verwenden Bernstein-Polynome n-ten Grades zur Approximation von $(n+1)$ Stützpunkten $b_0, \dots, b_n \in \mathbb{R}^p$.
- Die durch diese Stützpunkte definierte Kurve

$$p(t) = \sum_{i=0}^{n} b_i \cdot B_i^{(n)}(t)$$

An jeder Stelle $t \in [0,1]$ der Bézierkurve: Bernsteinpolynom i an der Stelle t gibt ~Gewichtungsfaktor für den Stützpunkt i an

heißt Bézierkurve vom Grad n.

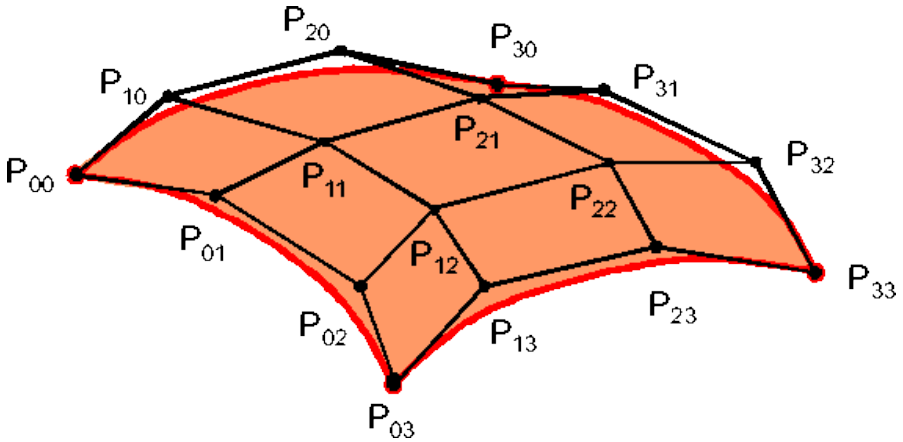
- Bézierkurven interpolieren den Anfangs- und Endpunkt, die anderen Stützpunkte liegen i.d.R. nicht auf der Kurve.



https://de.wikipedia.org/wiki/B%C3%A9zierkurve

Bézierflächen

- Hinzufügen einer Parameterdimension ermöglicht Bézierflächen im Raum
- Statt der Bedingung $t \in [0,1]$ gilt nun $s, t \in [0,1]$.

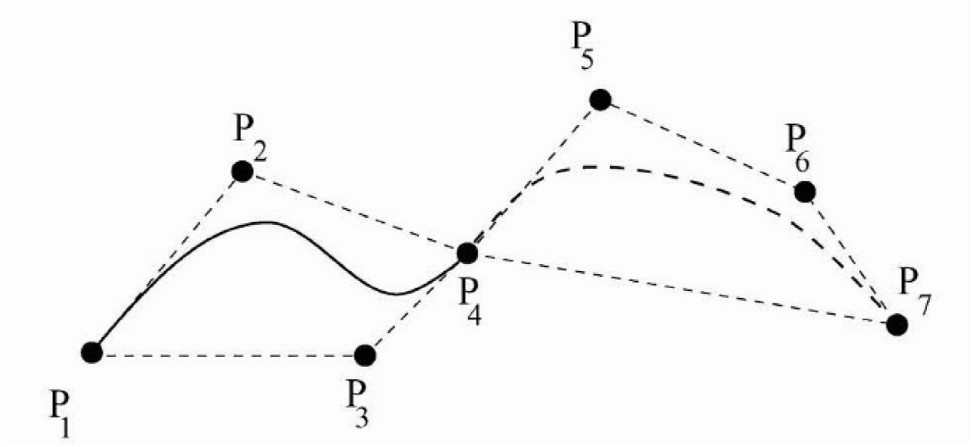


- Eigenschaften:
  - Invariant gegenüber affinen Transformationen (= Rotation, Verschiebung, Skalierung)
  - Symmetrisch gegenüber Stützpunkten, d.h. $b_0, \dots, b_n$ ergibt die selbe Kurve wie $b_n, \dots, b_0$.
- Nachteil Bézierkurven: Hoher Polynomgrad bei vielen Stützpunkten

http://www-lehre.inf.uos.de/~cg/2008/skript/node132.html

Stückweise Bézierkurven

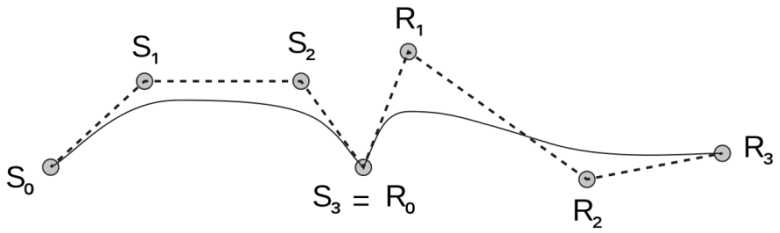
- Setzen sich aus mehreren Bézierkurven niedrigen Grades n zusammen
  - Üblicherweise dritter oder vierter Grad
- Berechnung einer Bézierkurve für jeweils $n + 1$ aufeinander folgende Stützpunkte
- Letzter Stützpunkt der ersten Bézierkurve wird erster Stützpunkt der nächsten usw.
  - Stützpunkte an Nahtstellen werden daher interpoliert



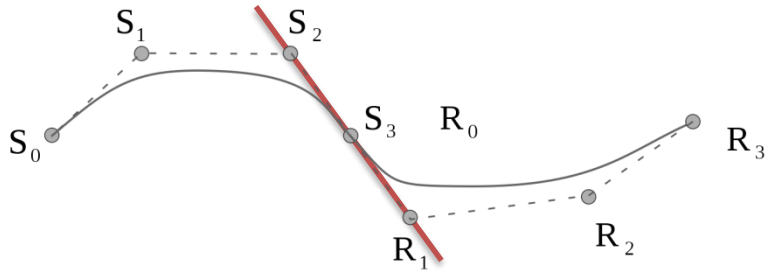
F. Klawonn: „Grundkurs Computergrafik mit Java“, Vieweg und Teubner, 3. Auflage, 2010

Stückweise Bézierkurven

- Vermeidung von Knicken an Nahtstellen durch Kollinearität des vorhergehenden und nachfolgenden Punktes (= C1-Kontinuität)

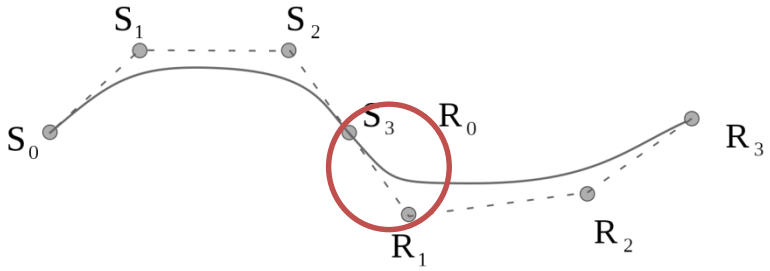


C0-Kontinuität



C1-Kontinuität

- C2-Kontinuität bei stückw. Bézierkurven nicht zwingend gegeben



- Gewünschte Eigenschaften *Interpolation*, *lokale Kontrolle* und *C2-Kontinuität* lassen sich für kubische Kurven nicht garantiert vereinbaren .

https://en.wikipedia.org/wiki/Smoothness

B-Spline-Kurven

- Durch Verzicht auf die Interpolation lässt sich lokale Kontrolle und C2-Kontinuität erreichen.
- Möglich durch Austausch der Basisfunktion zur B-Spline-Basisfunktion:

$$p(t) = \sum_{i=0}^{n-1} b_i \cdot N_{i,k}(t)$$

B-Spline Basisfunktion:

$$N_{i,0}(t) = \begin{cases} 1 & \text{für} \quad x_i \leq t < x_{i+1} \\ 0 & \text{sonst} \end{cases}$$

$$N_{i,k}(t) = \frac{t - x_i}{x_{i+k} - x_i} N_{i,k-1}(t) + \frac{x_{i+k+1} - t}{x_{i+k+1} - x_{i+1}} N_{i+1,k-1}(t)$$

Cox- de Boor-Rekursion

- Ordnung der Basisfunktion $k + 1$
- Polynom hat den Grad k.
- Knotenvektor aus aufsteigend sortierten Werten (bestimmt durch Anzahl der Stützpunkte n und Polynomgrad k).

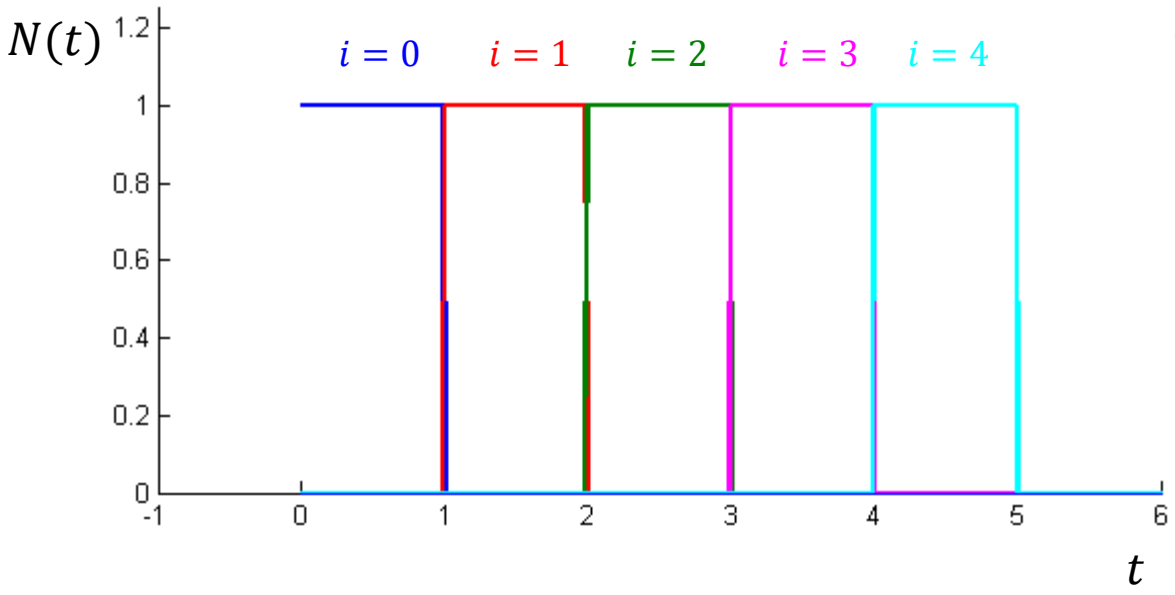
$$x_i : x_1 < \cdots < x_n < x_{n+1} < x_{n+k+1}$$

- Definitionsbereich von t frei wählbar
- Knotenvektor im Definitionsbereich frei wählbar (Kontrolle über Kurvenverlauf)

B-Spline-Kurven

- Beispiel: $k = 0, n = 5$ Stützpunkte → $n + k + 1 = 6$ Knoten, $i = [0 \dots 4]$

Plot des Kurvenabschnitts für Knotenvektor $[0\ 1\ 2\ 3\ 4\ 5]$ durch Einsetzen in die Cox-de-Boor-Rekursion, für $t = [0 \dots 5]$



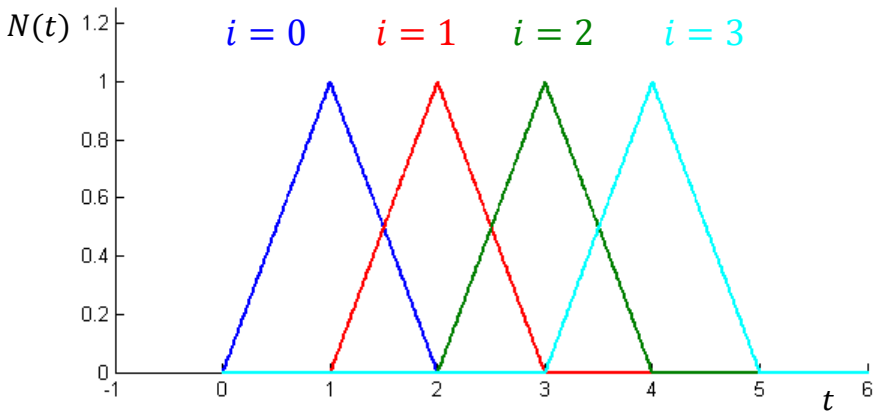
$\boldsymbol{k = 0}$

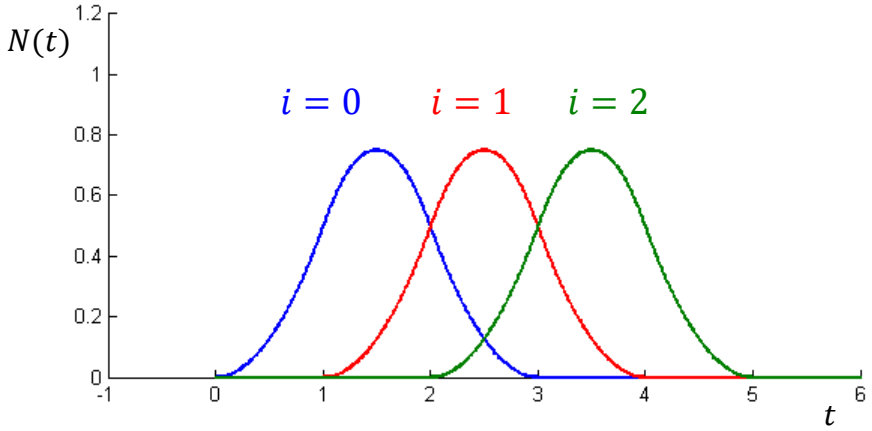
$$N_{i,0}(t) = \begin{cases} 1 & \text{für} \quad x_i \le t < x_{i+1} \\ 0 & \text{sonst} \end{cases}$$

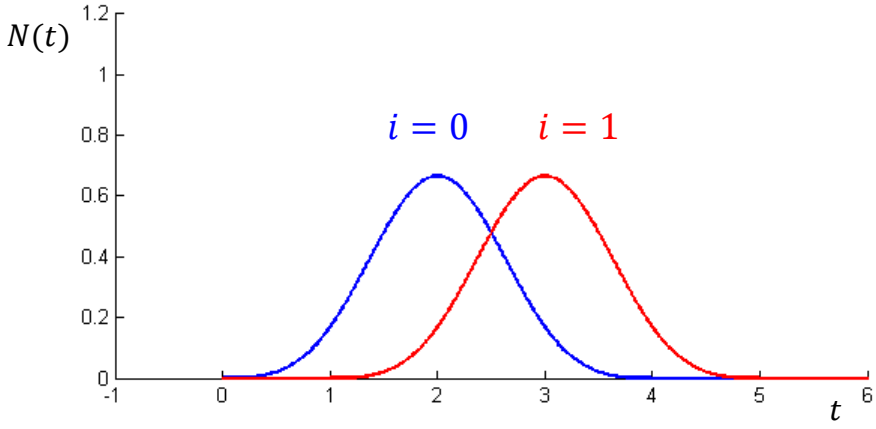
$i = 0$: $N_{0,0}(t) = 1$ für $0 \le t \le 1$

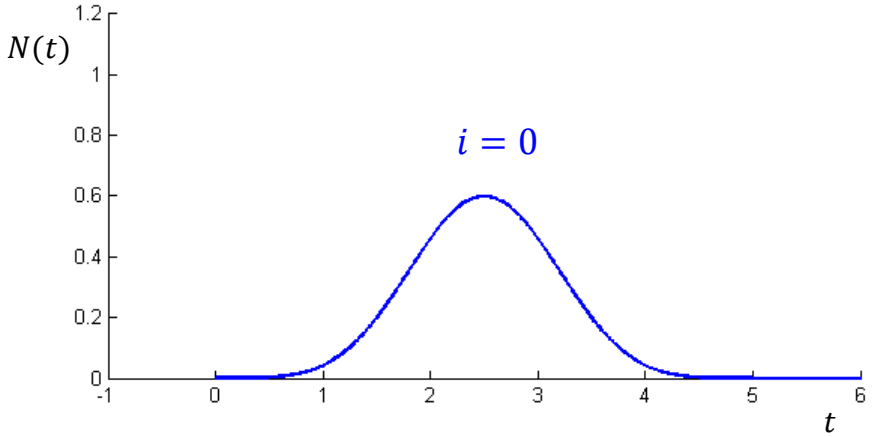
B-Spline-Kurven

- Beispiele für $k = 1, 2, 3, 4$ mit Knotenvektor [0 1 2 3 4 5]





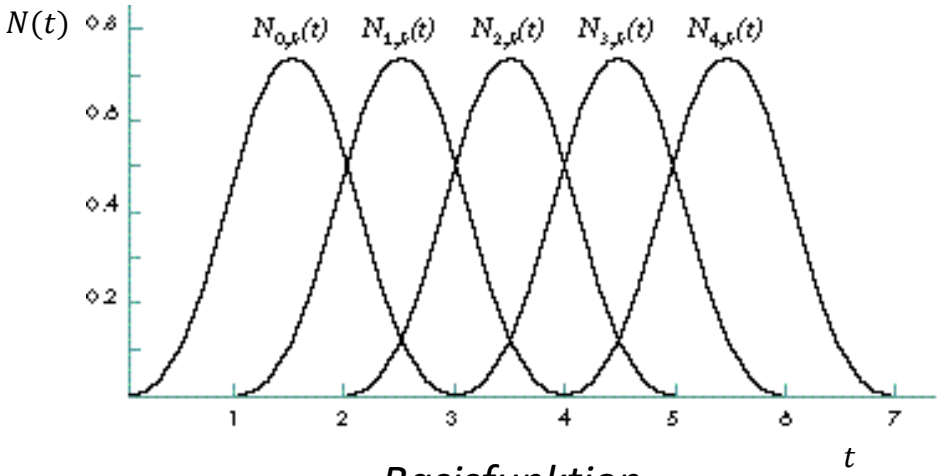




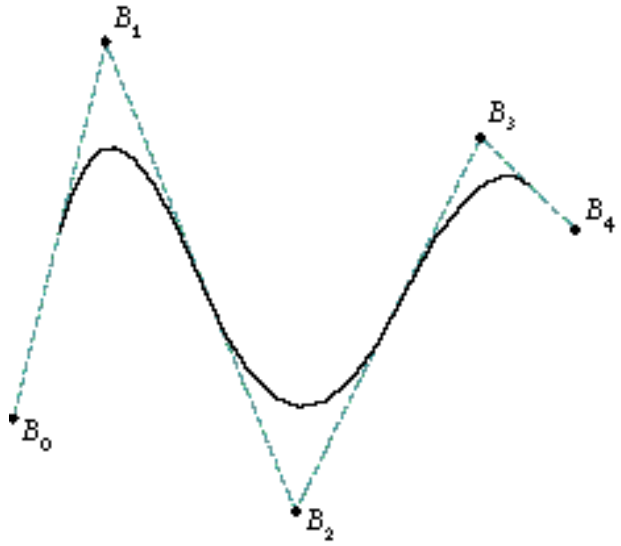
B-Spline-Kurven

- Beispiel:
  - 5 Stützpunkte b_0 … b_4, Grad $k = 2$ → Länge Knotenvektor $= 8$
  - mit $t = 0 \ldots 7$: <u>uniformer</u> Knotenvektor $[0\ 1\ 2\ 3\ 4\ 5\ 6\ 7]$

$$p(t) = \sum_{i=0}^{n-1} b_i \cdot N_{i,k}(t)$$



Basisfunktion



2D B-Spline

http://www.mactech.com/articles/develop/issue_25/schneider.html

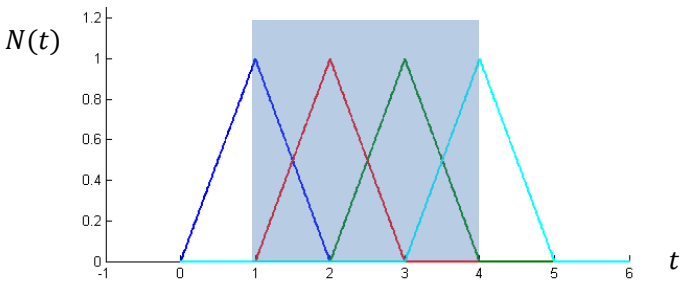
B-Spline-Kurven

$$N_{i,0}(t) = \begin{cases} 1 & \text{für} \quad x_i \leq t < x_{i+1} \\ 0 & \text{sonst} \end{cases}$$

$$N_{i,k}(t) = \frac{t - x_i}{x_{i+k} - x_i} N_{i,k-1}(t) + \frac{x_{i+k+1} - t}{x_{i+k+1} - x_{i+1}} N_{i+1,k-1}(t)$$

- Eigenschaften der Basisfunktion:
  - Eine Kurve der Ordnung $k + 1$ ist nur definiert wenn $k + 1$ Basisfunktionen ungleich 0 sind.

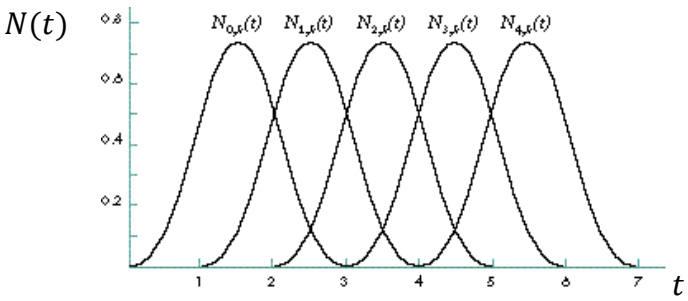
    Beispiel für $k = 1$:



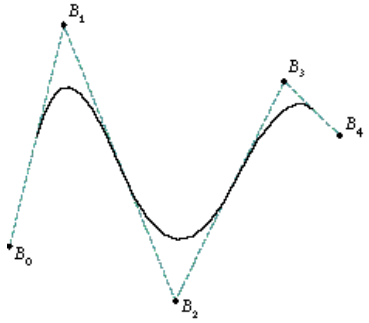
  - Für jedes t im Definitionsbereich addieren sich die Werte der Basisfunktionen zu 1.
  - Für jedes t im Definitionsbereich haben nie mehr als $k + 1$ Basisfunktionen einen Einfluss auf den Kurvenverlauf

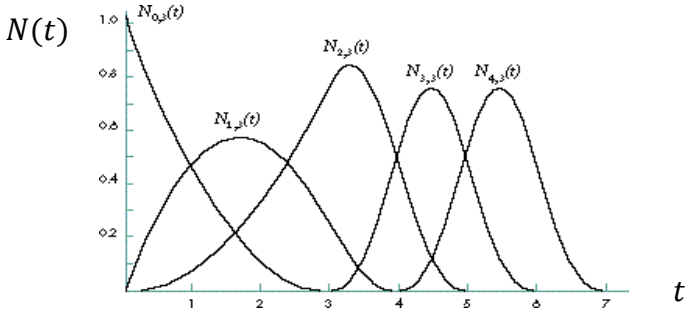
B-Spline-Kurven

- Bisher: periodische, uniforme Knotenvektoren
- Auswirkung offener uniformer Knotenvektoren:
  - k Wiederholungen des ersten Elements, Beispiel: $k = 2$

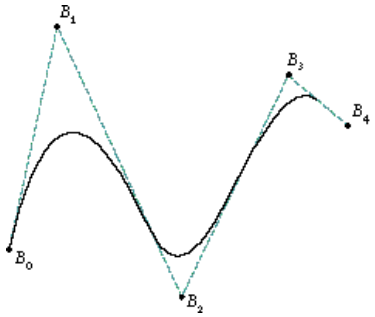


Knotenvektor = [0 1 2 3 4 5 6 7]





Knotenvektor = [0 0 0 3 4 5 6 7]

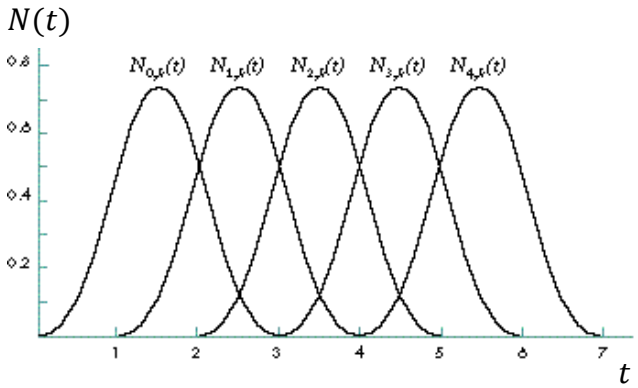


Geometrisch: Erster Knotenpunkt fällt mit erstem Stützpunkt zusammen (= Interpolation)

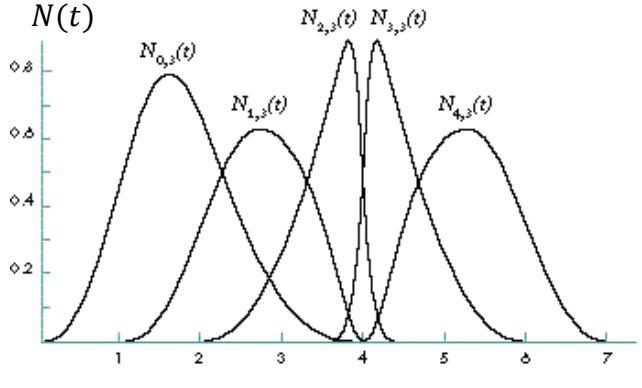
http://www.mactech.com/articles/develop/issue_25/schneider.html

B-Spline-Kurven

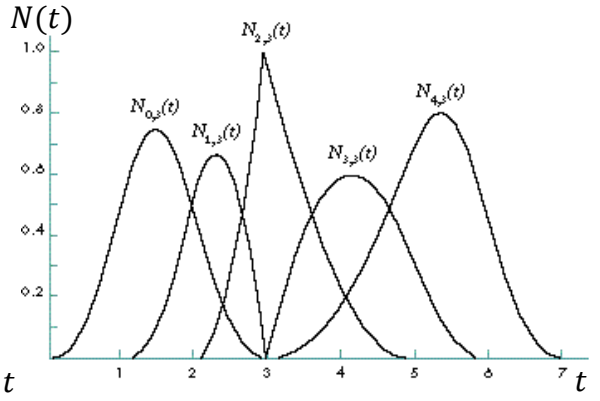
- Auswirkung nicht-uniformer Knotenvektoren:
  - Beispiele für $k = 2$



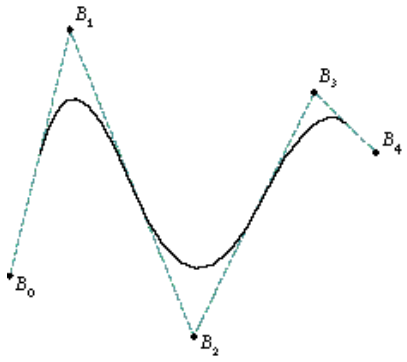
Knotenvektor = [0 1 2 3 4 5 6 7]

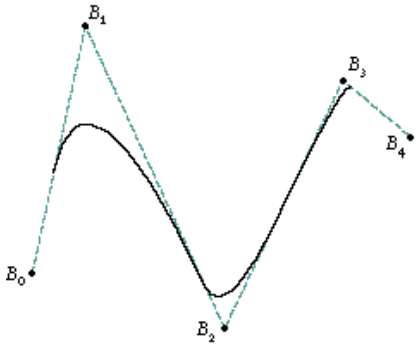


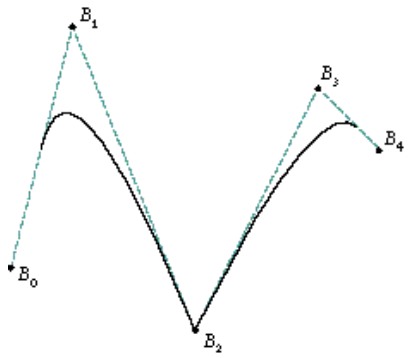
Knotenvektor = [0, 1, 2, 3.75, 4, 4.25, 6, 7]



Knotenvektor = [0, 1, 2, 3, 3, 5, 6, 7]





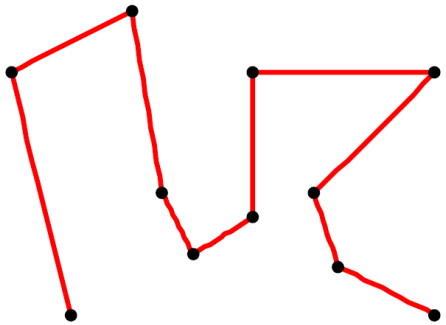


http://www.mactech.com/articles/develop/issue_25/schneider.html

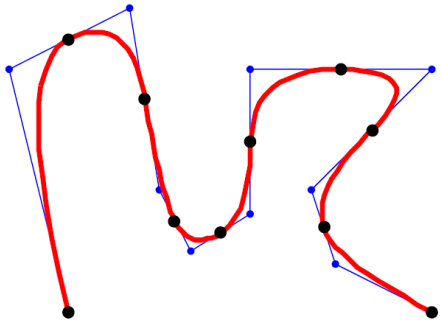
B-Spline-Kurven

- Auswirkung des B-Spline-Grades: Beispiele für Approximationen einer Kurve bei $k = 1, 2, 3, 4$

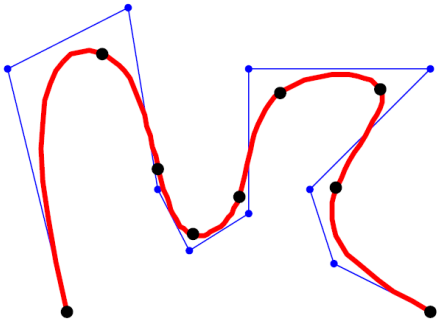
$k = 1$



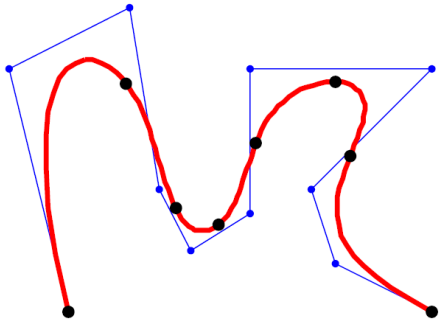
$k = 2$



$k = 3$



$k = 4$



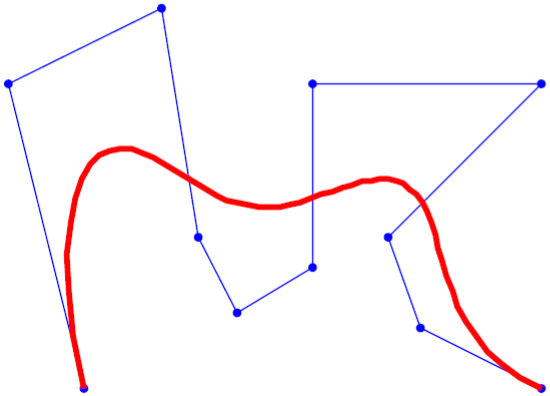
S. Terzi: „Splines in der Datenverarbeitung“, Seminararbeit Fachhochschule Aachen, 2012

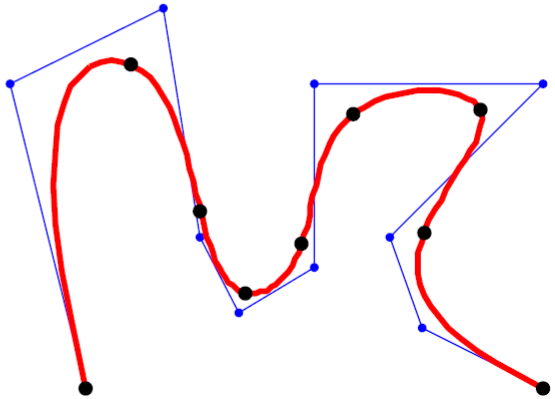
B-Spline-Kurven

- Zusammenfassend: Definition eines B-Splines durch
  - Kontrollpolygon (= Stützpunkte)
  - Knotenvektor
  - Grad bzw. Ordnung der Kurvensegmente
- Eigenschaften:
  - Grad der Kurve wird nicht durch Anzahl der Stützpunkte angegeben
  - Maximale Ordnung der Kurve entspricht Anzahl der Stützpunkte
  - Kurve liegt innerhalb der konvexen Hülle der Stützpunkte
  - Invarianz gegenüber affinen Transformationen

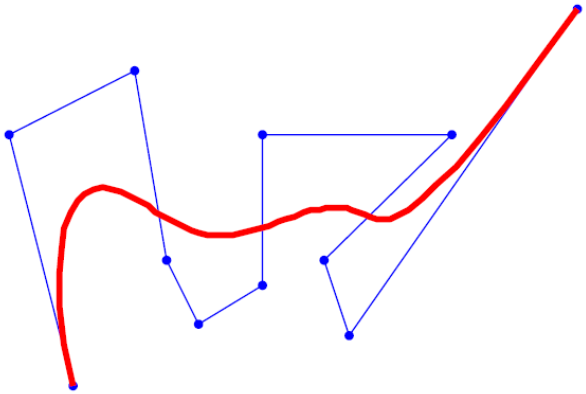
Vergleich: Bézier vs. B-Splines

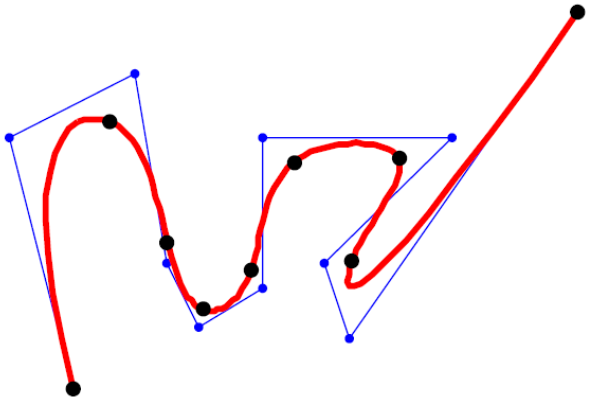
Bessere Modellierungseigenschaften der B−Splines:





Lokalitätseigenschaft der B−Splines:





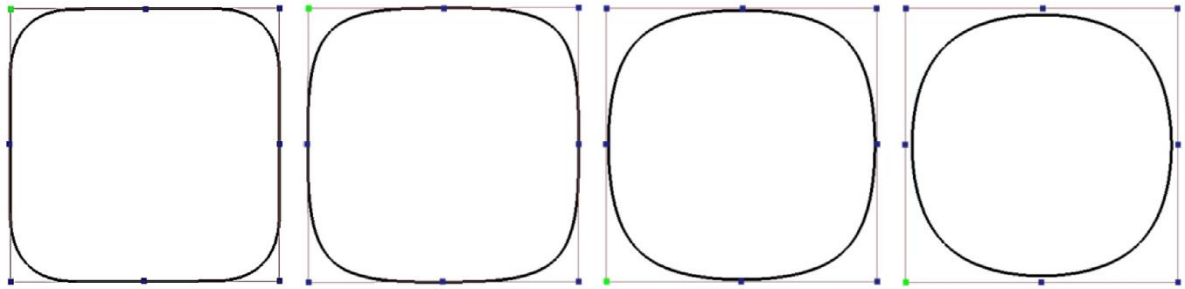
S. Terzi: „Splines in der Datenverarbeitung“, Seminararbeit Fachhochschule Aachen, 2012

B-Spline-Kurven

- Kontrolle über den Kurvenverlauf:
  - Änderung des Knotenvektors (periodisch uniform, offen uniform, nicht-uniform)
  - Änderung des Grades k
  - Änderung der Zahl/Position der Stützpunkte des Kontrollpolygons

  ➔ Deutlich mehr Flexibilität als Bézierkurven

- Limitationen
  - Können z.B. einen Kreis nicht exakt darstellen, nur approximieren



NURBS

- NURBS = Non-uniform rational B-Splines
  - Erweiterung der Basisfunktion auf rationale Funktion
  - Einführung eines Gewichtungsfaktors w für jeden Stützpunkt
- Kurve ist definiert durch:

$$p(t) = \frac{1}{\sum_{i=0}^{n-1} N_{i,k}(t) \cdot w_i} \sum_{i=0}^{n-1} b_i \cdot N_{i,k}(t) \cdot w_i$$

$$p(t) = \sum_{i=0}^{n-1} b_i \cdot R_{i,k}(t)$$

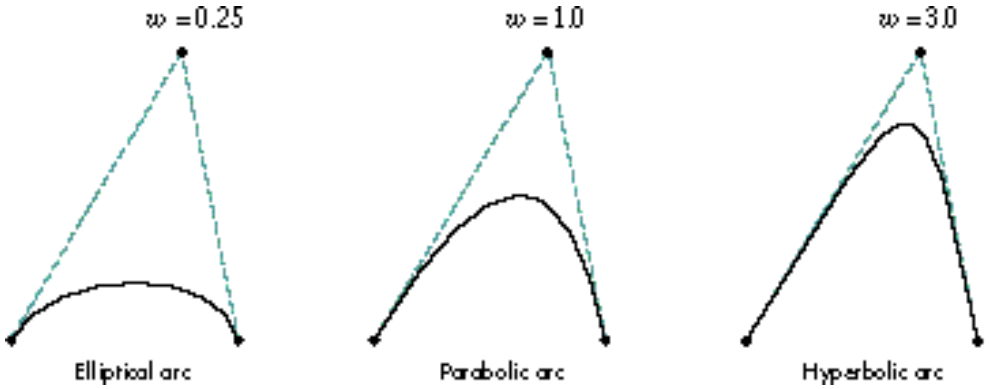
mit

$$R_{i,k}(t) = \frac{N_{i,k}(t) \cdot w_i}{\sum_{j=0}^{n-1} N_{j,k}(t) \cdot w_j}$$

NURBS Basisfunktion

NURBS

- Einfluss der Gewichte
  - Beispiel: 3 Stützpunkte, $w_0 = 1$, $w_2 = 1$, Knotenvektor $[0\ 0\ 0\ 1\ 1\ 1]$



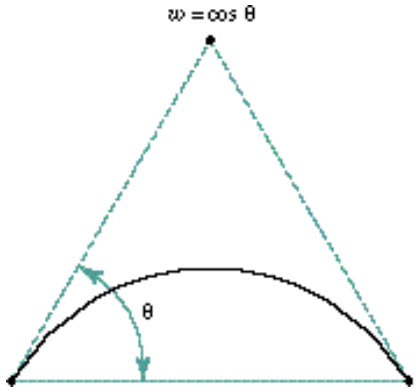
  - Erhöhung des Gewichts eines Stützpunktes erhöht dessen Einfluss

    → Kurve wird zum Stützpunkt hingezogen

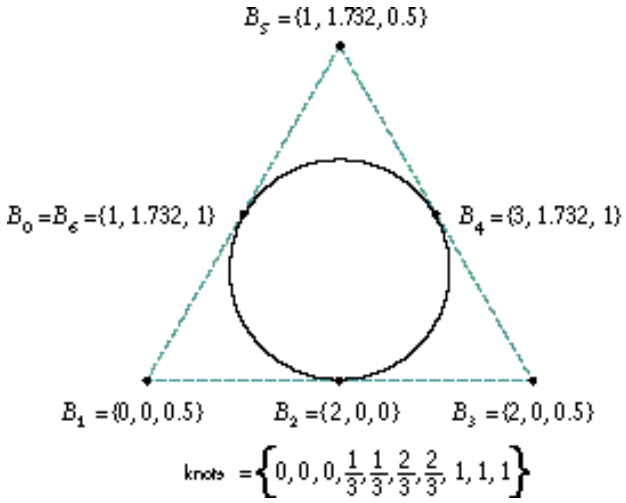
http://www.mactech.com/articles/develop/issue_25/schneider.html

NURBS

- Beispiel: Kreis aus NURBS durch Konstruktion von Kreisbögen
  - Kontrollpolygon gleichschenklig
  - Winkel θ so groß wie halbe Größe des Kreisbogens.
    z.B. Kreisbogen $120° \rightarrow \theta = 60°$
  - Gewicht des inneren Knotens $w_1 = \cos\theta$.
    z.B. $\theta = 60° \rightarrow w_1 = 0{,}5$
  - Knotenvektor = $[0\ 0\ 0\ 1\ 1\ 1]$



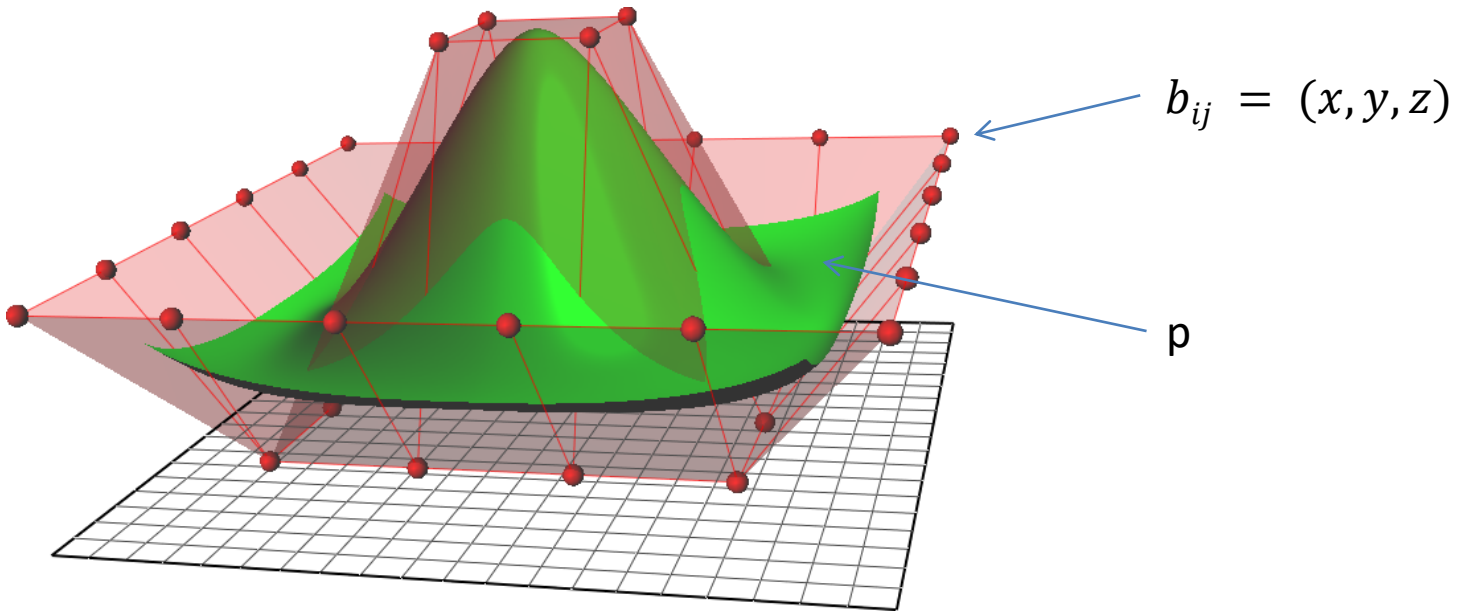
- Zusammengesetzt:



http://www.mactech.com/articles/develop/issue_25/schneider.html

B-Spline-/NURBS-Flächen

- Erweiterung der Definitionen auf NURBS-Flächen analog zu Bézierflächen.
  - Statt t nun zwei parametrische Richtungen: s, t
  - Stützpunkte in einem Gitter b_{ij} angeordnet.



https://de.wikipedia.org/wiki/Non-Uniform_Rational_B-Spline

Freiformkurven/-flächen in OpenGL

- „Evaluator“-Konzept für Bézierflächen
  - Definition eines eindimensionalen Evaluators: **`glMap1*()`**
  - Auswerten der Funktion an diskreten Stellen: **`glEvalCoord1()`**

- Eindimensionaler Evaluator:

Parameter target	Bedeutung
GL_MAP1_VERTEX_3	x, y, z Vertex Koordinaten
GL_MAP1_VERTEX_4	x, y, z, w Vertex Koordinaten
:	:

Definiert Intervall von *t: [t1 t2]*

Vektorlänge pro Stützpunkt

```
void glMap1fd(GLenum target, TYPE t1, TYPE t2, GLint stride,
              GLint order, const TYPE *points);
```

Ordnung des Polynoms
(= Anzahl der Stützpunkte)

Datenstruktur der Stützpunkte

Freiformkurven/-flächen in OpenGL

- Auswertung an diskreten Stellen (Vertexposition bestimmen):

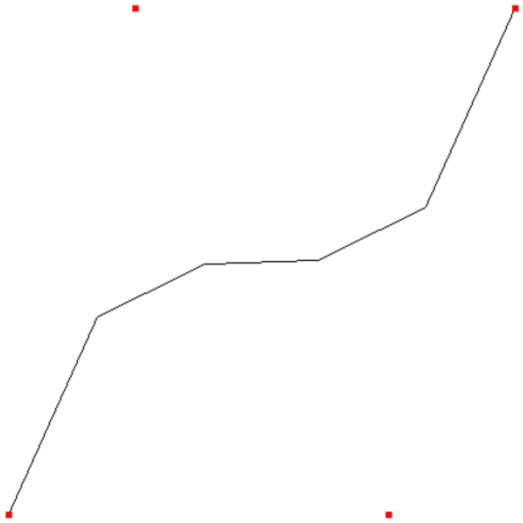
  `glEvalCoord1{fd} (TYPE t);`

- Beispiel:

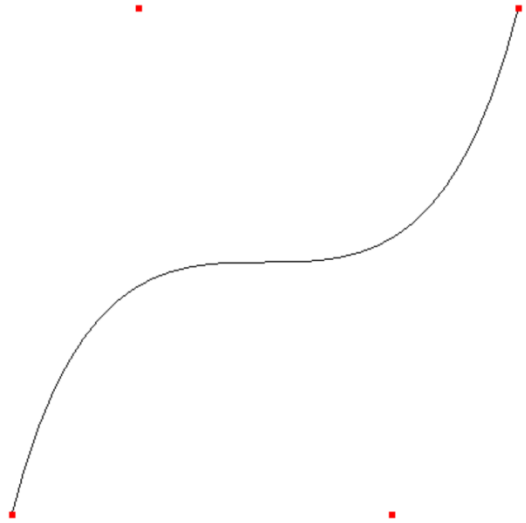
```
#define STEPS 5

GLfloat ctrlpts[4][3] = {{-4.0,-4.0,0.0}, {-2.0,4.0,0.0}, {2.0,-4.0,0.0}, {4.0,4.0,0.0}};

void init(void){
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &ctrlpts[0][0]);
    glEnable(GL_MAP1_VERTEX_3);
}

void display(void) {
    int i;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP);
        for (i = 0; i <= STEPS; i++)
        glEvalCoord1f((GLfloat) i / (GLfloat) STEPS);
    glEnd();
}
```

Freiformkurven/-flächen in OpenGL

- Beispiel



STEPS = 5



STEPS = 30

- Analog: zwei-dimensionale Evaluatoren für Flächen

Freiformkurven/-flächen in OpenGL

- GLU NURBS interface: high-level Interface zum Evaluator-Konzept
  - Erzeugung eines NURBS Objektes

    ```
    theNurb = gluNewNurbsRenderer();
    ```

  - Einstellung der NURBS Rendering Eigenschaften

    ```
    gluNurbsProperty();
    ```

  - Kurve zeichnen

    ```
    gluBeginCurve(theNurb);
        gluNurbsCurve(theNurb, knotCount, knots, stride,
                      &ctrlpts, order, type);
    gluEndSurface(theNurb);
    ```

    Anzahl der Knoten

    Knotenvektor

  - Flächen äquivalent:
    - `gluBeginSurface(theNurb);`
    - `gluNurbsSurface(…);`

ZUSAMMENFASSUNG

Zusammenfassung

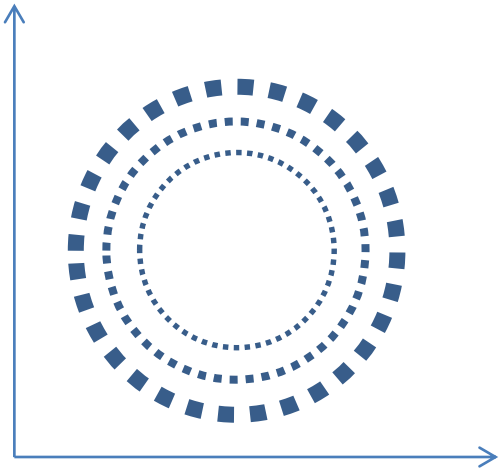
- Geometrische Beschreibung von Objekten durch Zusammensetzen von Grundobjekten, meist planare Polygone
  - Datenstrukturen: Knoten-/Kantenliste, …
- Polygonisierung/ Triangulation zum Erzeugen von Polygonnetzen
  - Daten aus Isoflächen, Punktwolken etc.
  - Z.B. per Marching Cubes Algorithmus, Triangulation von Polygonen, Triangulation von Punktwolken
  - Gewisse Eigenschaften sollten eingehalten werden, z.B. keine degenerierten oder sehr flache Dreiecke → Delaunay Triangulation
  - Nachbearbeitungsschritte: Meshverfeinerung, Meshdezimierung, Meshglättung
- Darstellung gekrümmter Kurven/Flächen über parametrische Funktionen
  - Bézierkurven/-flächen, stückweise Bézierkurven/-flächen
  - B-Splines
  - NURBS

ÜBUNGSAUFGABEN

Programmierübung

Zeichnen von Linien

1. Zeichnen Sie mit OpenGL eine 2D-Szene, die ein Koordinatensystem mit zwei Achsen x/y darstellt, in dem drei Kreise aus runden Punkten unterschiedlicher Farbe dargestellt werden. Die Punktgröße soll vom innersten Kreis zum äußeren jeweils verdoppelt werden. Nutzen Sie das bereitgestellte Template zur Implementierung der mit ////// markierten Bereiche.



Programmierübung

Culling

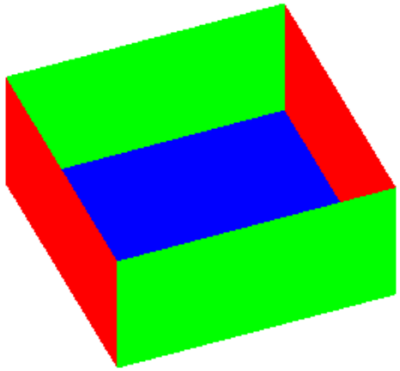
1. Das bereitgestellte Template *template_culling.c* zeigt eine 3D Szene mit einer Box bestehend aus 4 Seitenflächen und einer Bodenfläche. Schalten Sie nacheinander Backface- und Frontface-Culling ein und beobachten Sie den Effekt.

2. Wie müsste man die Orientierung des Bodens ändern, damit man aus dem aktuellen Blickwinkel auf die Vorderseite des Polygons schaut?

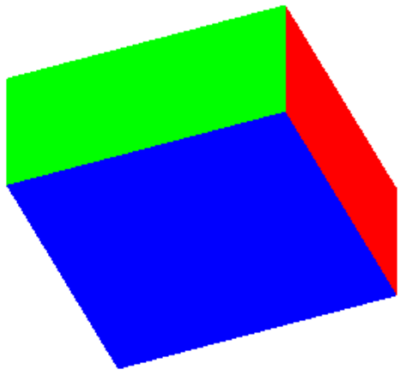
Lösung

1. Das bereitgestellte Template *template_culling.c* zeigt eine 3D Szene mit einer Box bestehend aus 4 Seitenflächen und einer Bodenfläche. Schalten Sie nacheinander Backface- und Frontface-Culling ein und beobachten Sie den Effekt.

Initial:



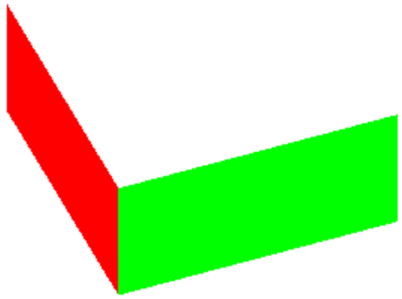
Front-Face Culling



```
glEnable(GL_CULL_FACE);
glCullFace(GL_FRONT);
```

Es werden nur die Flächen gerendert, für die der Beobachter aus dem aktuellen Blickwinkel auf die Rückseite schaut.

Back-Face Culling



```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```

Es werden nur die Flächen gerendert, für die der Beobachter aus dem aktuellen Blickwinkel auf die Vorderseite schaut.

Lösung

2. Wie müsste man die Orientierung des Bodens ändern, damit man aus dem aktuellen Blickwinkel auf die Vorderseite des Polygons schaut?

Einfaches ändern der Reihenfolge der Vertices von hinten nach vorne:

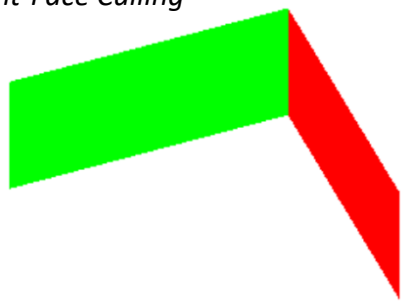
(Ursprünglicher Quellcode:)

```
// BOTTOM
glVertex3f(-0.5f, -0.5f,  0.5f);
glVertex3f(-0.5f, -0.5f, -0.5f);
glVertex3f( 0.5f, -0.5f, -0.5f);
glVertex3f( 0.5f, -0.5f,  0.5f);
```

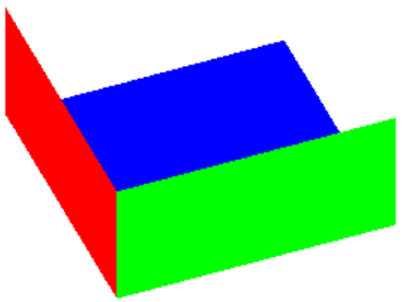
(Neuer Quellcode:)

```
// BOTTOM
glVertex3f( 0.5f, -0.5f,  0.5f);
glVertex3f( 0.5f, -0.5f, -0.5f);
glVertex3f(-0.5f, -0.5f, -0.5f);
glVertex3f(-0.5f, -0.5f,  0.5f);
```

Front-Face Culling



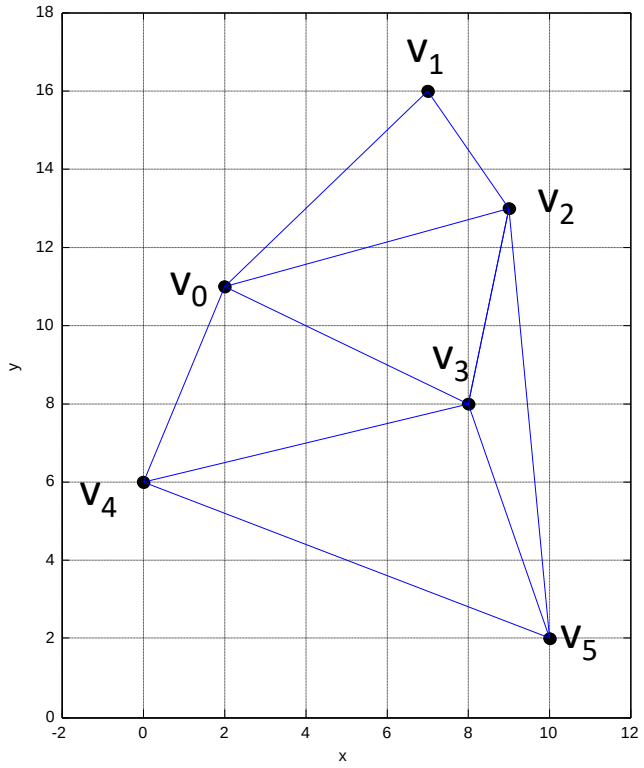
Back-Face Culling



Übungsaufgaben

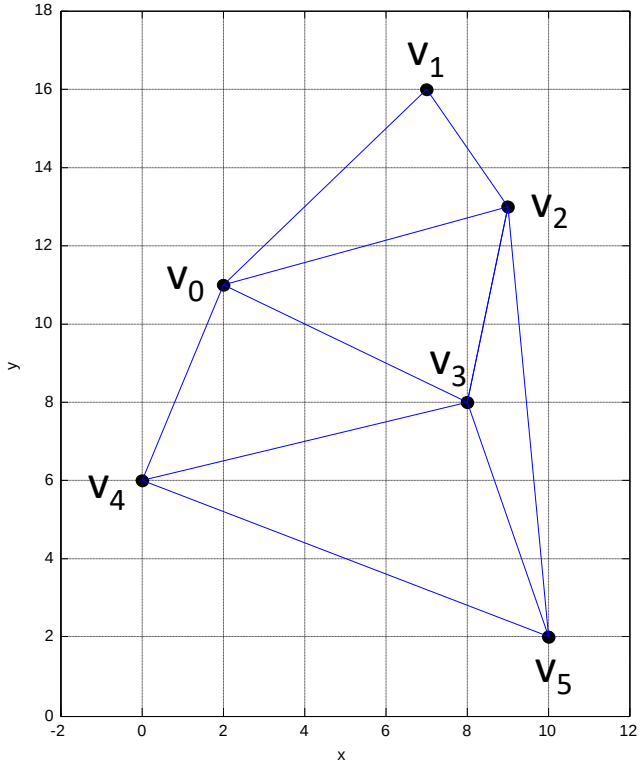
Datenstrukturen für Polygonnetze

1. Geben Sie das folgenden Polygonnetze als Knoten- und Kantenliste an. Achten Sie auf konsistente Orientierung der Polygone (im Uhrzeigersinn)



Lösung

Geben Sie das folgenden Polygonnetz als Knoten- und Kantenliste an. Achten Sie auf konsistente Orientierung der Polygone (im Uhrzeigersinn).



Knoten

Knoten ID	X	Y
0	2	11
1	7	16
2	9	13
3	8	8
4	0	6
5	10	2

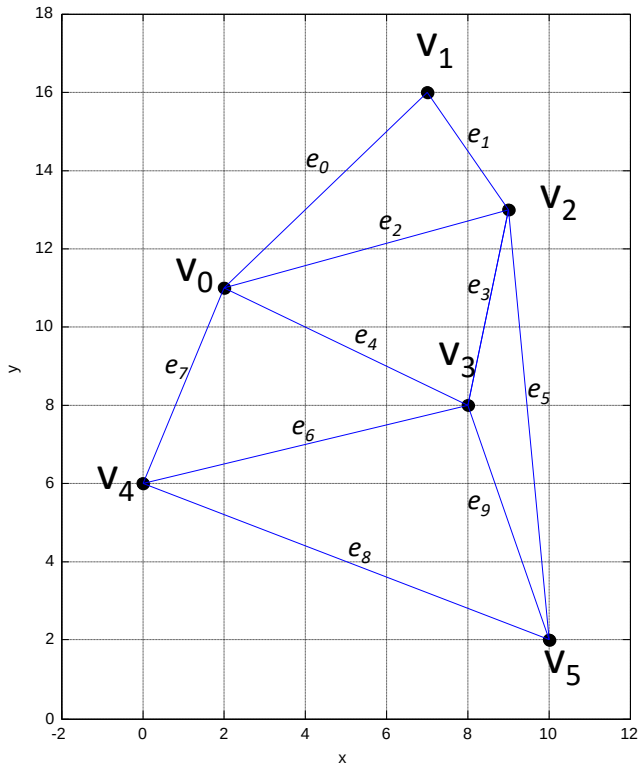
Als Knotenliste

Polygone

Polygon ID	Knoten ID 1	Knoten ID 2	Knoten ID 3
0	0	1	2
1	0	2	3
2	0	3	4
3	2	5	3
4	4	3	5

Lösung (2)

Geben Sie das folgenden Polygonnetz als Knoten- und Kantenliste an. Achten Sie auf konsistente Orientierung der Polygone (im Uhrzeigersinn).



Als Kantenliste

Knoten

Knoten ID	X	Y
0	2	11
1	7	16
2	9	13
3	8	8
4	0	6
5	10	2

Kanten

Kanten ID	Knoten ID 1	Knoten ID 2
0	0	1
1	1	2
2	2	0
3	2	3
4	0	3
5	2	5
6	3	4
7	0	4
8	4	5
9	3	5

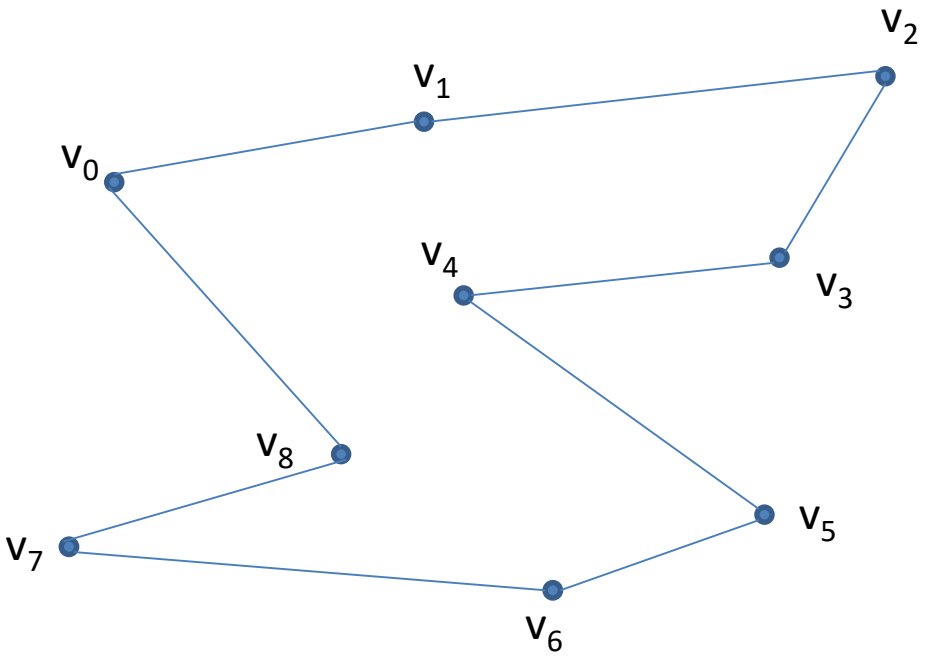
Polygone

Polygon ID	Kanten ID 1	Kanten ID 2	Kanten ID 3
0	0	1	2
1	2	3	4
2	4	6	7
3	6	9	8
4	3	5	9

Übungsaufgaben

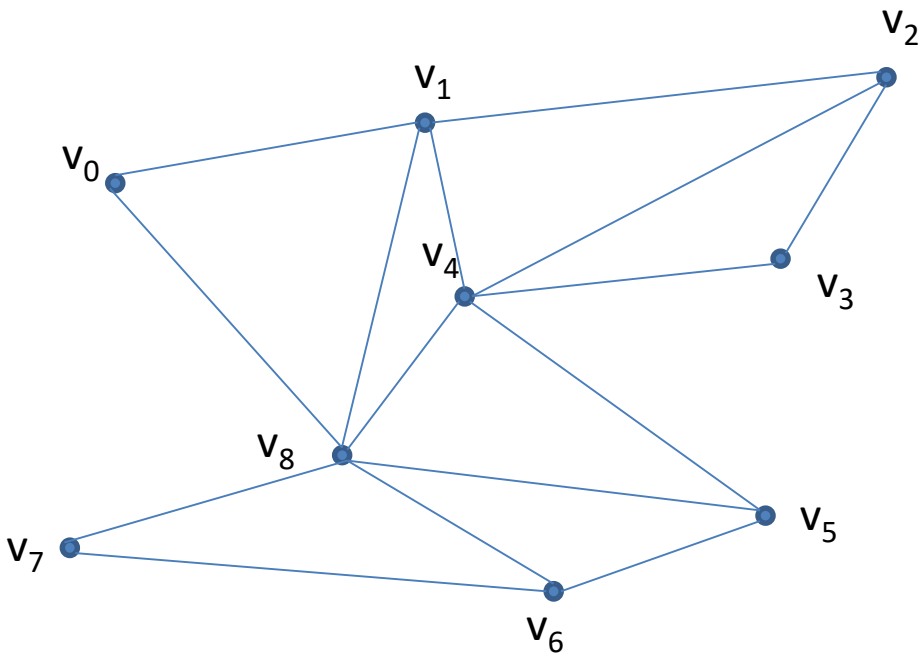
Triangulation

Triangulieren Sie das folgende Polygon nach der Brute-Force-Diagonalensuche. Starten Sie bei Vertex v_0



Lösung

Triangulieren Sie das folgende Polygon nach der Brute-Force-Diagonalensuche. Starten Sie bei Vertex v_0

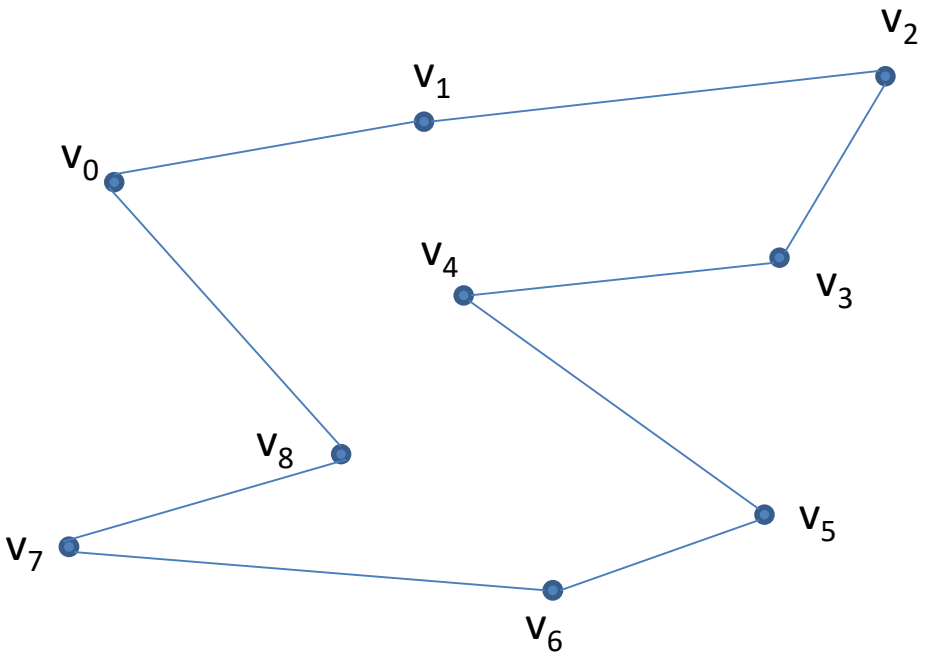


1	Initialer Vertex: v0 v0: L = v1, R = v8 Kein weiterer Vertex im Dreieck v0-v1-v8 → Kante v1-v8 einfügen
2	Nächster Knoten in verbliebenem Polygon: v1 L = v2, R= v8 V4 würde im Dreieck v1-v2-v8 liegen → nächste parallele Gerade zur Kante v2-v8 durch Vertices im Dreieck v1-v2-v8 → Kante v1-v4 einfügen
3	Nächster Knoten im ersten verbliebenem Polygon: v1 L = v4, R = v8 Kein weiterer Vertex im Dreieck v1-v4-v8 → Kante v4-v8 einfügen
4	Nächster Knoten im ersten verbliebenem Polygon: v4 L = v5, R = v8 Kein weiterer Vertex im Dreieck v4-v5-v8 → Kante v5-v8 einfügen
5	Nächster Knoten im ersten verbliebenem Polygon: v5 L = v6, R = v8 Kein weiterer Vertex im Dreieck v5-v6-v8 → Kante v6-v8 einfügen
6	(Sprung zu zweitem verbliebenen Polygon (v1-v2-v3-v4) Nächster Knoten im zweiten verbliebenen Polygon: v1 L = v2, R = v4 Kein weiterer Vertex im Dreieck v1-v2-v4 → Kanten v2-v4 einfügen

Übungsaufgaben

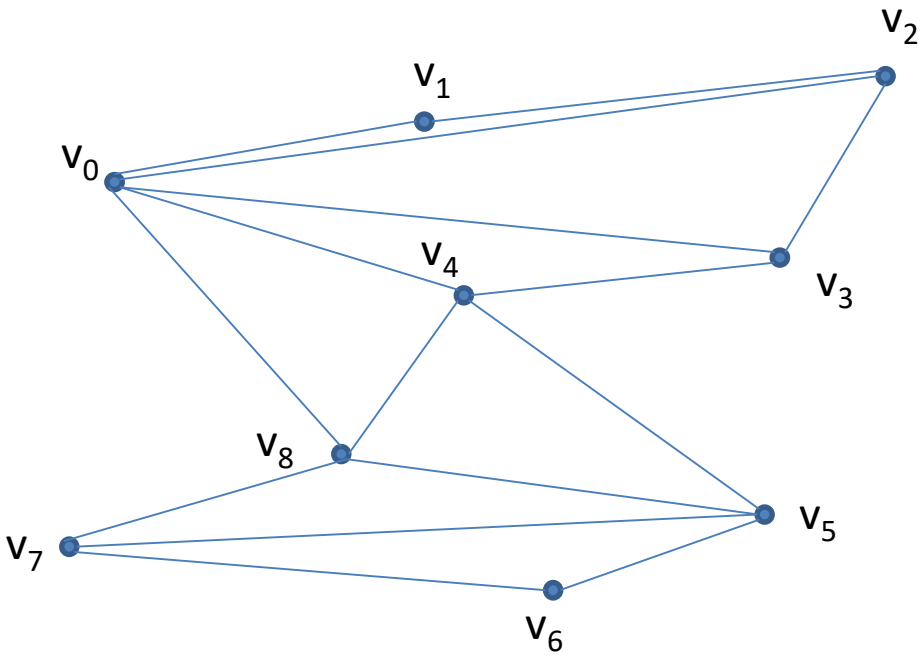
Triangulation II

Triangulieren Sie das folgende Polygon nach der Sweep-Line Methode (Monotonie bzgl. der y-Achse!)



Lösung

1. Triangulieren Sie das folgende Polygon nach der Sweep-Line Methode (Monotonie bzgl. der y-Achse!)



Bearbeitungsschritte:

		Stack
INIT	Sortiere v2, v1, v0,v3,v4,v8,v5,v7,v6 Leeren Stack erzeugen v2, v1 auf Stack	v2 v1
v0	v0 ⇔ v1? Nein! Kante v0 – v2 da von v1 nicht verdeckt.	v2 v0
v3	v3 ⇔ v0? Ja! Kante v3 – v0	v0 v3
v4	v4 ⇔ v3? Nein! Kante v4 – v0 da von v3 nicht verdeckt.	v0 v4
v8	v8 ⇔ v4? Ja! Kante v8 – v4	v4 v8
v5	v5 ⇔ v8? Ja! Kante v5 – v8	v8 v5
v7	v7 ⇔ v5? Ja! Kante v7 – v5	v5 v7
v6	Letzter Knoten: Kante zu allen außer erstem/letzten auf Stack → nichts zu tun	

Übungsaufgaben

Bezierkurven, B-Splines, NURBS

1. Es soll ein B-Spline vom Polynomgrad 3 für 7 Stützpunkte gezeichnet werden. Der erste und letzte Stützpunkt sollen interpoliert werden. Geben Sie einen uniformen Knotenvektor an, wobei $t \in [0\ 1]$.
2. Eine Bézierkurve soll die 3 Stützpunkte $v_0 = (1{,}5)$, $v_1 = (4{,}6)$ und $v_2 = (2{,}10)$ verbinden. Welchen Grad muss das Bernsteinpolynom haben? Skizzieren sie den Kurvenverlauf!
3. Wie würde ein B-Spline zweiten Grades ($k = 2$) mit Knotenvektor $[0, 0, 0, {}^1/_3, {}^2/_3, 1]$ die Punkte approximieren? Skizzieren sie den Kurvenverlauf!
4. Gegeben sind erneut die 3 Stützpunkte $v_0 = (1{,}5)$, $v_1 = (4{,}6)$ und $v_2 = (2{,}10)$, die mit einem B-Spline approximiert werden sollen. Der Grad des B-Splines soll 2 sein, der Knotenvektor uniform $[0\ 1\ 2\ 3\ 4\ 5]$. Geben Sie den Startpunkt P des B-Splines an.

Lösung

1. Es soll ein B-Spline vom Polynomgrad 3 für 7 Stützpunkte gezeichnet werden. Der erste und letzte Stützpunkt sollen interpoliert werden. Geben Sie einen uniformen Knotenvektor an, wobei $t \in [0\ 1]$.

Polynomgrad = 3 → $k = 3$

Anzahl der Stützpunkte = 7 → $n = 7$

Anzahl der Knoten im Knotenvektor = $n + k + 1$ → $n + k + 1 = 11$

Erster und letzter Punkt sollen interpoliert werden → Knotenvektor ist offen zu wählen, d.h. k Wiederholungen des ersten und letzten Elementes

Uniformer Knotenvektor: d.h. äquidistante Verteilung der inneren Knoten.

Resultierender Knotenvektor:

[0, 0, 0, 0, ¼, ½, ¾, 1 1 1 1]

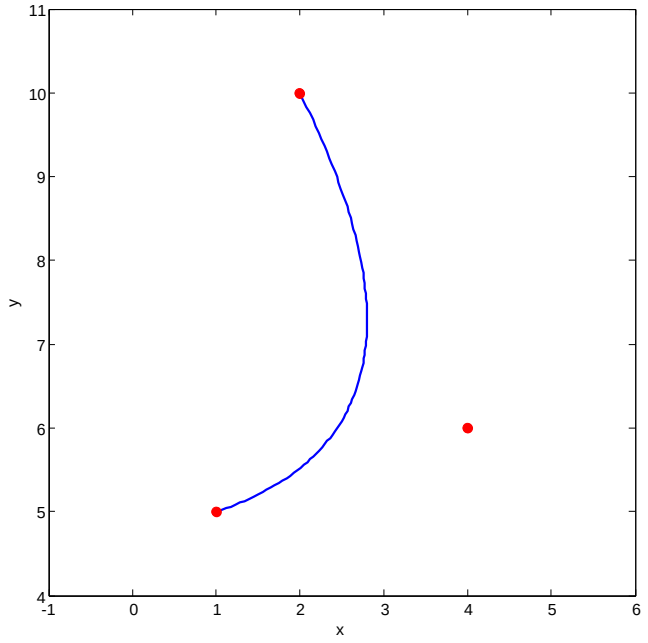
Lösung (2)

2. Eine Bézierkurve soll die 3 Stützpunkte $v_0 = (1, 5)$, $v_1 = (4, 6)$ und $v_2 = (2, 10)$ verbinden. Welchen Grad muss das Bernsteinpolynom haben? Skizzieren sie den Kurvenverlauf!

Das Bernsteinpolynom hat Grad $n = 2$, da $n + 1 = 3$ Stützpunkte verbunden werden sollen.

Skizze des Kurvenverlaufs:

- Interpolation des ersten und letzten Punktes
- Kurve gekrümmt zum zweiten Punkt hingezogen.

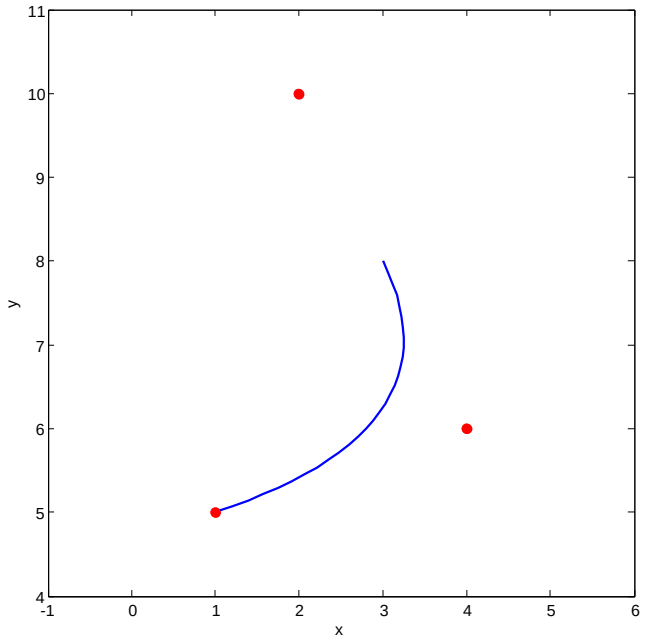


Lösung (3)

3. Wie würde ein B-Spline zweiten Grades ($k = 2$) mit Knotenvektor $[0, 0, 0, 1/3, 2/3, 1]$ die Punkte approximieren? Skizzieren sie den Kurvenverlauf!

Skizzieren des Kurvenverlaufs

- Erster Punkt wird interpoliert, da das erste Elements im Knotenvektor 3 mal Auftritt (k Wiederholungen, offener Knotenvektor).
- Letzter Punkt wird nur approximiert, Kurve endet bereits zwischen dem zweiten und dritten Stützpunkt auf halber Strecke, da die Knoten uniform verteilt sind.



Lösung (4)

4. Gegeben sind erneut die 3 Stützpunkte $v_0 = (1, 5)$, $v_1 = (4, 6)$ und $v_2 = (2, 10)$, die mit einem B-Spline approximiert werden sollen. Der Grad des B-Splines soll 2 sein, der Knotenvektor uniform $[0\ 1\ 2\ 3\ 4\ 5]$. Geben Sie den Startpunkt P des B-Splines an.

Der B-Spline ist gültig für den Bereich k bis n (d.h. dort wo mindestens $k + 1$ Kurven ungleich 0 sind, vergleiche Plot in den Folien). Bei gegebenem $k = 2$ und Knotenvektor entspricht dies dem Bereich zwischen $x_k = x_2 = 2$ und $x_n = x_3 = 3$ des Knotenvektors. Der Startpunkt befindet sich also bei $t = 2$, der Endpunkt bei $t = 3$.

Nun wird die Cox-De-Boor Rekursion durchgeführt. Entsprechend der Formel für eine B-Spline-Kurve

$$p(t) = \sum_{i=0}^{n-1} b_i \cdot N_{i,k}(t)$$

müssen wir für alle $i = 0 \ldots 2$ die Werte der B-Spline Basisfunktion $N_{0,2}$, $N_{1,2}$, $N_{1,2}$ berechnen. Diese ergeben sich aus der Rekursion. Geschickterweise beginnt man daher in der untersten Hierarchieebene der Rekursion für $k = 0$. Man setzt in die Gleichung die entsprechenden Werte für t, i und k ein. Wir beginnen bei $i = 0$ und $k = 0$:

$$N_{i,0}(t) = \begin{cases} 1 & \text{für} \quad x_i \leq t < x_{i+1} \\ 0 & \text{sonst} \end{cases}$$

$$N_{0,0}(t) = \begin{cases} 1 & \text{für} \quad 0 \leq t < 1 \\ 0 & \text{sonst} \end{cases} \qquad \Rightarrow N_{0,0}(2) = 0$$

Lösung (5)

Da im Rekursionsaufruf einmal auf $N_{i+1,k-1}$ zugegriffen wird und von $k = 2$ bis $k = 0$ zwei Rekursionsaufrufe erfolgen, muss bis $N_{4,0}(2)$ berechnet werden.

Für die unterste Hierarchieebene der Rekursion ergibt sich:

$$N_{0,0}(2) = 0 \quad N_{1,0}(2) = 0 \quad N_{2,0}(2) = 1 \quad N_{3,0}(2) = 0 \quad N_{4,0}(2) = 0$$

Wir fahren auf der nächsten Rekursionsstufe fort, Beispiel: für $i = 1, k = 1$

$$N_{i,k}(t) = \frac{t - x_i}{x_{i+k} - x_i} N_{i,k-1}(t) + \frac{x_{i+k+1} - t}{x_{i+k+1} - x_{i+1}} N_{i+1,k-1}(t)$$

$$N_{1,1}(t) = \frac{t - x_1}{x_2 - x_1} N_{1,0}(t) + \frac{x_3 - t}{x_3 - x_2} N_{2,0}(t)$$

$$N_{1,1}(2) = \frac{2 - 1}{2 - 1} \cdot 0 + \frac{3 - 2}{3 - 2} \cdot 1 = 1$$

Die zweite Hierarchieebene baut sich somit wie folgt auf:

$$N_{0,0}(2) = 0 \quad N_{1,0}(2) = 0 \quad N_{2,0}(2) = 1 \quad N_{3,0}(2) = 0 \quad N_{4,0}(2) = 0$$

$$N_{0,1}(2) = 0 \quad N_{1,1}(2) = 1 \quad N_{2,1}(2) = 0 \quad N_{3,1}(2) = 0$$

Lösung (6)

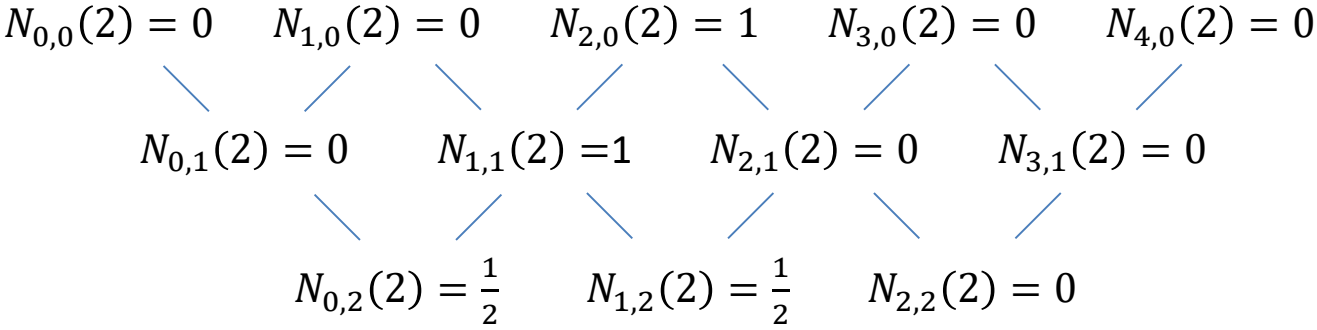
Entsprechend wird die dritte Hierarchieebene berechnet (damit $k = 2$ – der Grad der B-Spline-Basis-Funktion ist erreicht. Ende der Rekursion), Beispiel für $i = 0$:

$$N_{i,k}(t) = \frac{t - x_i}{x_{i+k} - x_i} N_{i,k-1}(t) + \frac{x_{i+k+1} - t}{x_{i+k+1} - x_{i+1}} N_{i+1,k-1}(t)$$

$$N_{0,2}(t) = \frac{t - x_0}{x_2 - x_0} N_{0,1}(t) + \frac{x_3 - t}{x_3 - x_1} N_{1,1}(t)$$

$$N_{0,2}(2) = \frac{2 - 0}{2 - 1} \cdot 0 + \frac{3 - 2}{3 - 1} \cdot 1 = \frac{1}{2}$$

Es ergibt sich die vollständige dritte Hierarchiestufe:



Lösung (7)

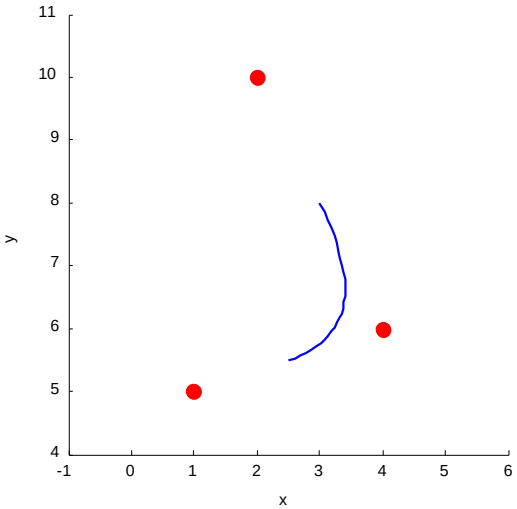
Um den Startpunkt P der Kurve zu ermitteln, muss nun noch in die Gleichung einer B-Spline-Kurve eingesetzt werden:

$$p(t) = \sum_{i=0}^{n-1} b_i \cdot N_{i,k}(t)$$

$$p(2) = \sum_{i=0}^{2} v_i \cdot N_{i,2}(2)$$

$$p(2) = v_0 \cdot N_{0,2}(2) + v_1 \cdot N_{1,2}(2) + v_2 \cdot N_{2,2}(2)$$

$$p(2) = \begin{pmatrix}1\\5\end{pmatrix} \cdot {}^1/_2 + \begin{pmatrix}4\\6\end{pmatrix} \cdot {}^1/_2 + \begin{pmatrix}2\\10\end{pmatrix} \cdot 0 = \begin{pmatrix}2{,}5\\5{,}5\end{pmatrix}$$



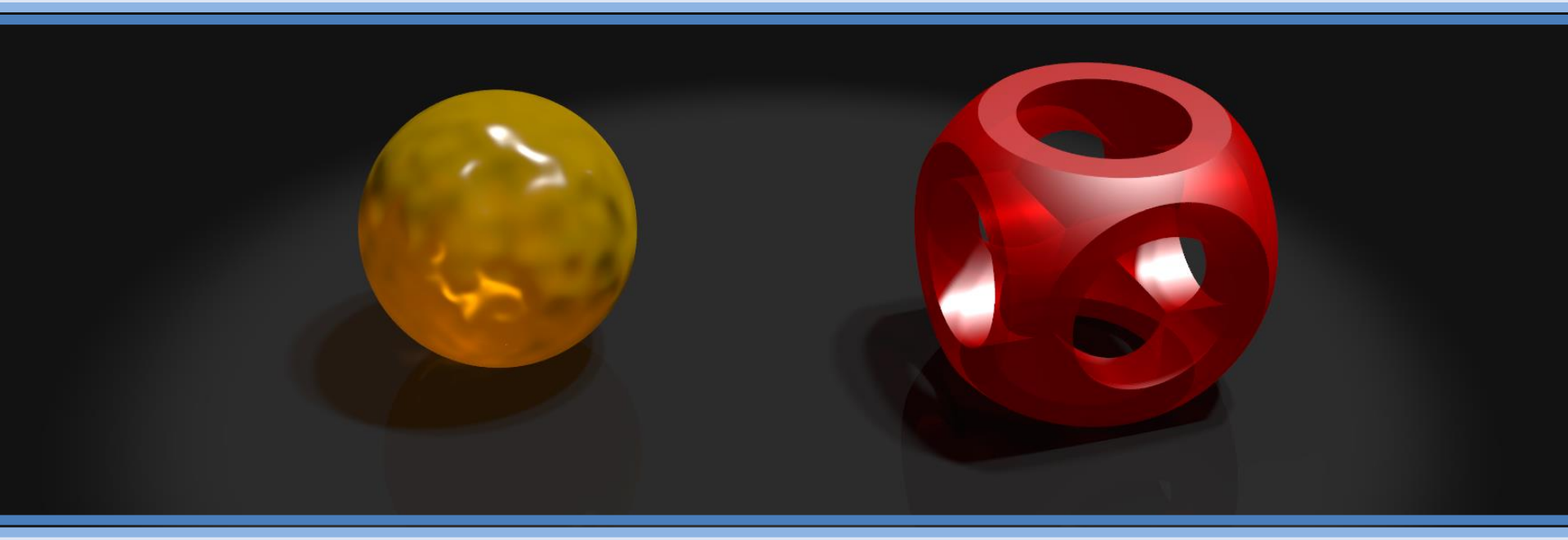
*D.h. für jedes t im Definitionsbereich ergibt sich ein Punkt auf der B-Spline-Kurve als eine Linearkombination **aller** Stützpunkte mit Gewichtsfaktoren die sich aus der B-Spline-Basisfunktion an der entsprechenden Stelle t ergeben.*

Übungsfragen Kapitel 3

- Was ist Front- und Backface-Culling? Wie wird bestimmt ob ein Betrachter auf Vorder- oder Rückseite eines Polygons blickt?
- Was ist adaptives Meshing? Welchen Vorteil bietet es?
- Wann ist ein Polygon a) einfach? b) monoton bzgl. einer Achse? c) planar, d) konvex
- Beschreiben Sie eine Möglichkeit zur Generierung eines Polygonnetzes aus Punktwolken
- Wann erfüllt ein Polygonmesh die Delaunay-Eigenschaft?
- Erläutern Sie ein Verfahren zur Meshglättung
- Was ist ein offener B-Spline?
- Wie kann eine B-Spline-Kurve einen Stützpunkt interpolieren?

Computergrafik

T. Hopp



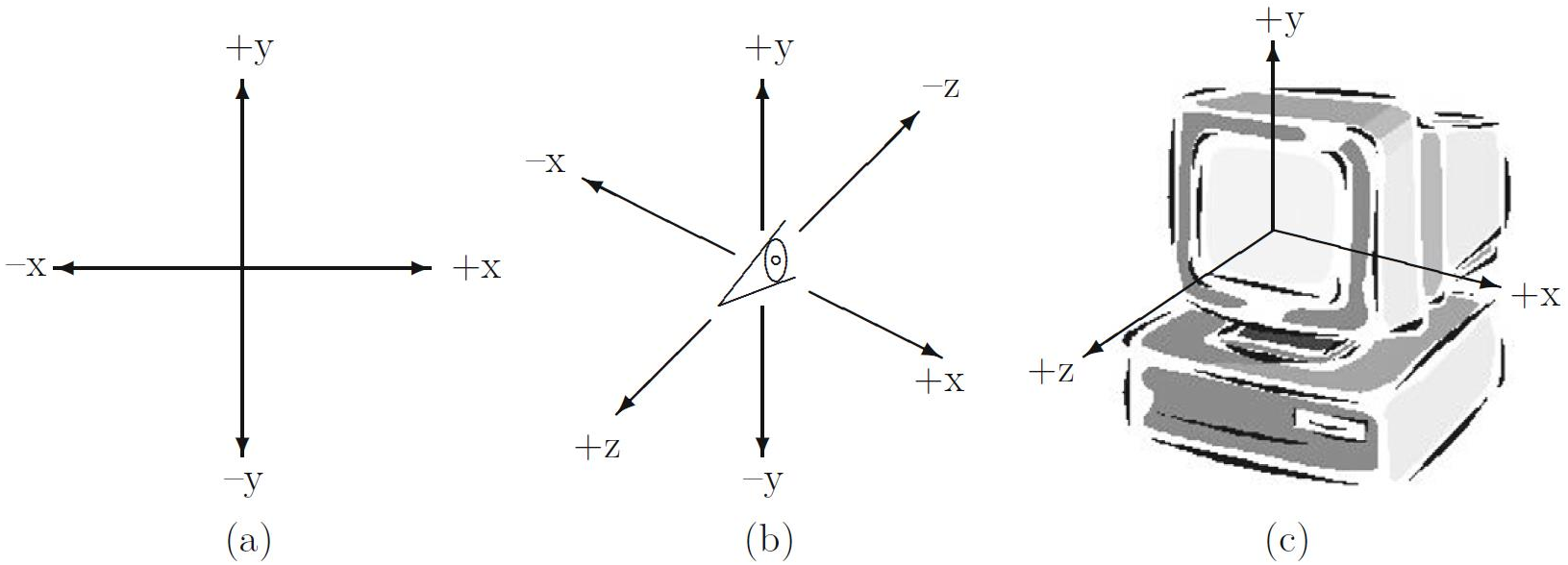
Themenübersicht

1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. **Koordinatensysteme und Transformationen**
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. Raytracing
11. Volumenvisualisierung

4.1. KOORDINATENSYSTEME

(OpenGL-) Koordinatensystem

- In der Regel 3D Euklidisches Koordinatensystem mit x-, y-, z-Achse
- Betrachter (=Augpunkt) standardmäßig im Ursprung ($x = 0, y = 0, z = 0$)



OpenGL Koordinatensystem aus
(a) Sicht des Augpunktes, (b) Sicht eines externen Beobachters
(c) mit Blick auf den Bildschirm

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Koordinatensysteme in der CG: Unterscheidung

- In der Computergrafik gibt es nicht nur ein Koordinatensystem. Es werden i.d.R. die folgenden unterschieden:

- **Objektkoordinaten:**

  Lokale Koordinatensysteme für 3D Objekte
- **Weltkoordinatensystem:**

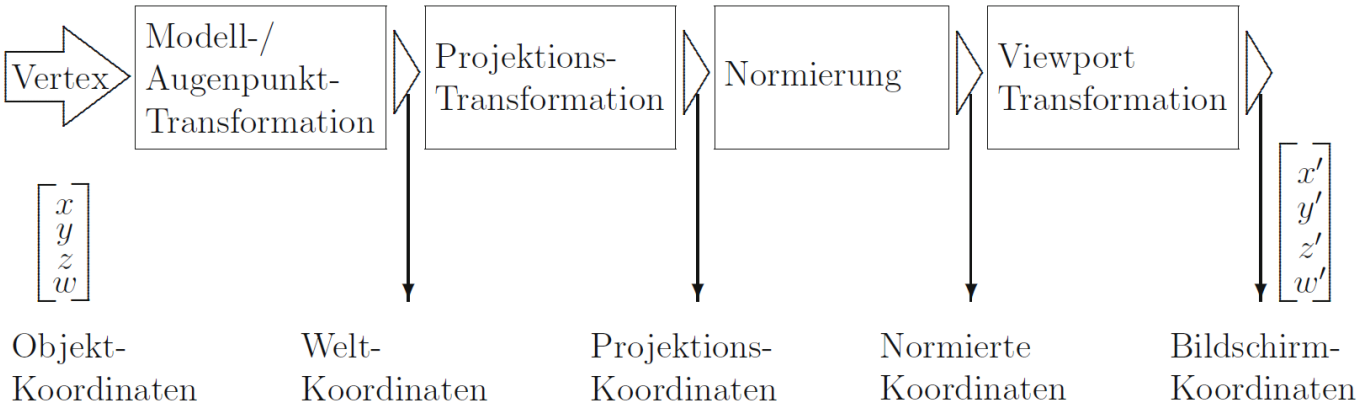
  Koordinaten die für die gesamte Szene gelten
- **Projektionskoordinaten:**

  Koordinaten nach perspektivischer bzw. orthogonaler Projektion
- **Normierte Koordinaten:**

  Auf vorgegebenen Wertebereich beschränkte Koordinaten, die nach Division der Projektionskoordinaten mit dem inversen Streckungsfaktor *w* entstehen.
- **Bildschirmkoordinaten:**

  Koordinaten, die Szene in der gewählten Fenstergröße darstellen

Transformationskette



- **Modell-/Augenpunkttransformation**: Positionierung von Objekten in der Szene
  (= ModelView Matrix)
- **Projektionstransformation:** Definition des sichtbaren Volumens, z.B. Blickwinkel
  (= Frustum)
- **Normierung:** Transformation der Koordinaten auf Intervall [-*w*, +*w*] und Division durch inversen Streckungsfaktor *w*
  (= Normalized Device Coordinates)
- **Viewport-Transformation:** Positionierung der Vertices im Fenster

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

4.2. HOMOGENE KOORDINATEN

Transformationen in 3D

- Ein Punkt im 3-dimensionalen Euklidischen Raum kann durch drei Koordinaten x, y, z beschrieben werden
  - Darstellung in Vektorform: $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ bzw. (x, y, z) in transponierter Schreibweise
- Eine Transformation T des Ortsvektors des Punktes $v = (x, y, z)$ kann durch Multiplikation mit einer 3 x 3 Transformationsmatrix M durchgeführt werden: $v' = Mv$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\begin{aligned} x' &= m_{11} \cdot x + m_{12} \cdot y + m_{13} \cdot z \\ y' &= m_{21} \cdot x + m_{22} \cdot y + m_{23} \cdot z \\ z' &= m_{31} \cdot x + m_{32} \cdot y + m_{33} \cdot z \end{aligned}$$

- Translation kann durch die Transformationsmatrix nicht abgebildet werden!

Homogene Koordinaten

- Beschreibung eines Punktes im 3D Raum durch vier Komponenten:

$$v = (x_h, y_h, z_h, w)$$

„inverser Streckungsfaktor“

- Euklidische Koordinate (x, y, z) berechnet sich als

$$x = \frac{x_h}{w}, \qquad y = \frac{y_h}{w}, \qquad z = \frac{z_h}{w}$$

- Werte für *w:*
  - $w = 1$: homogene Koordinaten = Euklidische Koordinaten
  - $w < 1,\ w > 1$: Streckung/Stauchung um Faktor $1/w$
  - $w = 0$ Division durch 0 → Richtungsvektoren
- Transformation mit homogenen Koordinaten

Translation

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Transformation mit homogenen Koordinaten

- Allgemein:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

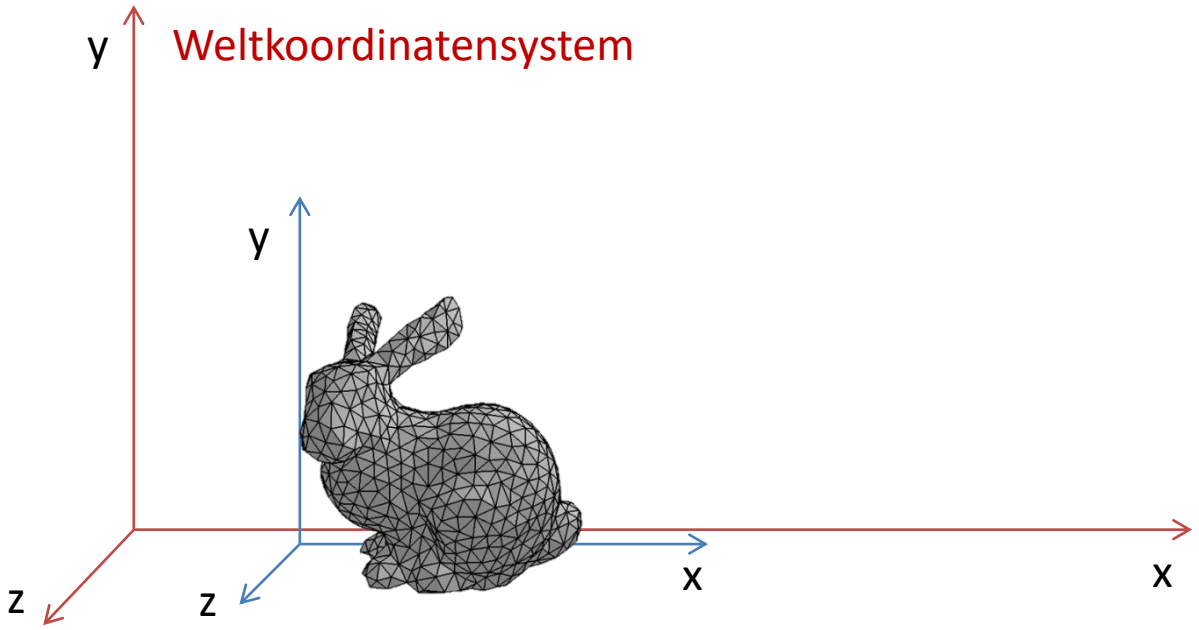
$$v' \quad = \quad \boldsymbol{M} \quad v$$

- Alle Transformationen der Transformationskette arbeiten mit homogenen Koordinaten → Effiziente Abbildung der Operationen in der Hardware
- Ausführung mehrerer Transformationen hintereinander
  - Möglichkeit 1: $v' = \left(\ldots \cdot \mathbf{M}_2 \cdot (\mathbf{M}_1 \cdot v)\right)$
  - Möglichkeit 2: $v' = (\ldots \cdot \mathbf{M}_2 \cdot \mathbf{M}_1) \cdot v$
- OpenGL nutzt Möglichkeit 2, da effizienter.
- Matrizen-Operationen in OpenGL:

```
glLoadMatrixf(const Glfloat *M);    // laden einer Matrix
glLoadIdentity(Glvoid);             // Identitätsmatrix
glMultMatrixf(const Glfloat *M);    // Multipl. mit Matrix im Speicher
```

4.3. MODELL- UND AUGPUNKTTRANSFORMATIONEN

Modell-Transformationen



- Definition von Objekten i.d.R. in lokalem Koordinatensystem
  - Feste Kopplung 3D-Objekt an lokales Koordinatensystem
- Positionierung in der 3D Szene: lokales Koordinatensystem wird im Weltkoordinatensystem „bewegt“:
  - Translation
  - Drehung
  - Skalierung

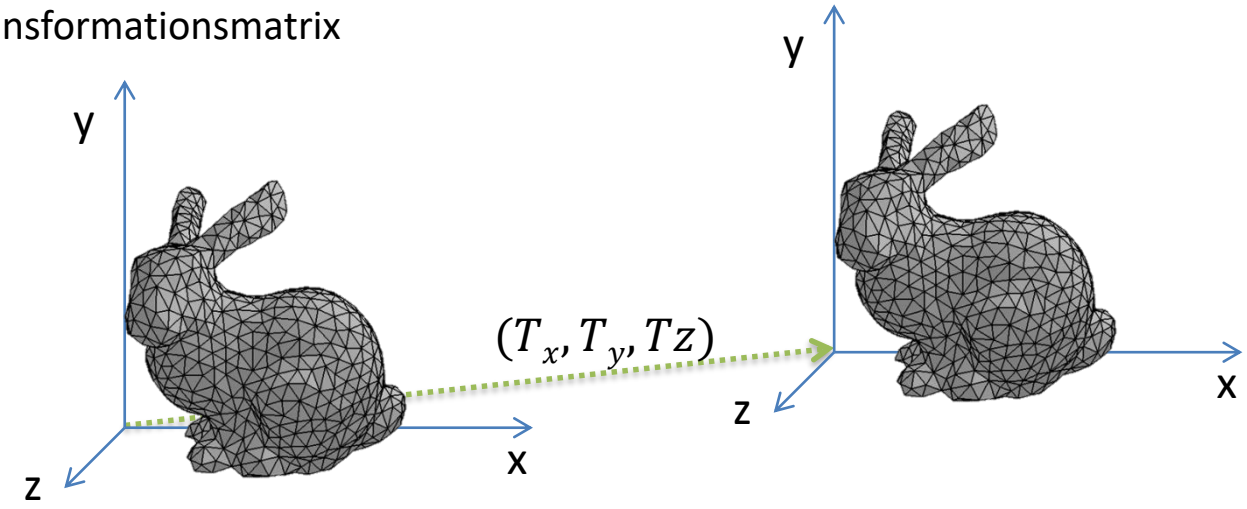
  = Affine Transformationen

Translation

- Translation eines Punktes um (T_x, T_y, T_z)

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

- In OpenGL: **`glTranslatef(T_x,T_y,T_z)`**
  - Definition einer 4 x 4 Translationsmatrix anhand T_x, T_y, T_z
  - State Machine (!): Multiplikation mit der aktuell im Speicher vorliegenden Transformationsmatrix



Rotation

- Rotation um die x-Achse:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

```
glRotatef(α,1,0,0);
```

- Rotation um die y-Achse:

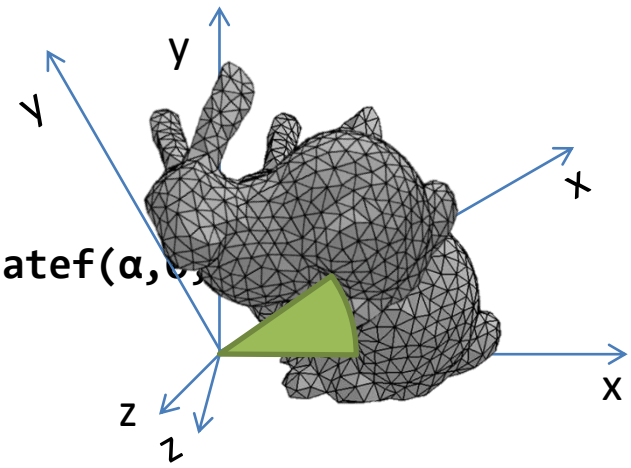
$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

```
glRotatef(α,0,1,0);
```

- Rotation um die z-Achse:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

```
glRotatef(α,0,
```



Rotation

- Rotation eines Punktes v um einen Richtungsvektor $r = (R_x, R_y, R_z)$:

$$v' = \boldsymbol{M}\, v$$

$$M = \begin{pmatrix} {R_x}^2(1-\cos\alpha)\cos\alpha & R_xR_y(1-\cos\alpha) - R_z\sin\alpha & R_xR_z(1-\cos\alpha) + R_y\sin\alpha & 0 \\ R_yR_x(1-\cos\alpha) + R_z\sin\alpha & {R_y}^2(1-\cos\alpha)\cos\alpha & R_yR_z(1-\cos\alpha) - R_x\sin\alpha & 0 \\ R_zR_x(1-\cos\alpha) - R_y\sin\alpha & R_zR_y(1-\cos\alpha) + R_z\sin\alpha & {R_z}^2(1-\cos\alpha)\cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- In OpenGL:

```
glRotatef(α,R_x,R_y,R_z);
```

Länge des Richtungsvektors = 1

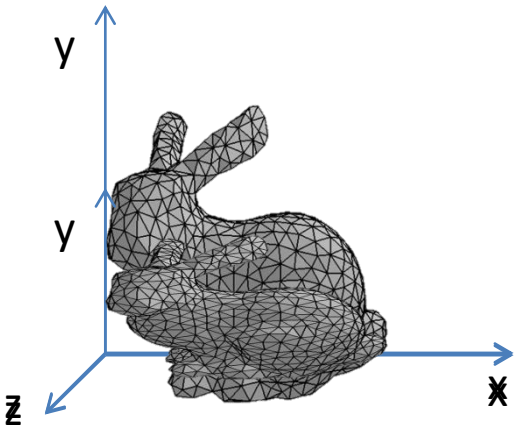
Skalierung

- Skalierungsmatrix: Skalierung um Faktoren (S_x, S_y, S_z)

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

- In OpenGL:

```
glScalef(S_x,S_y,S_z);
```



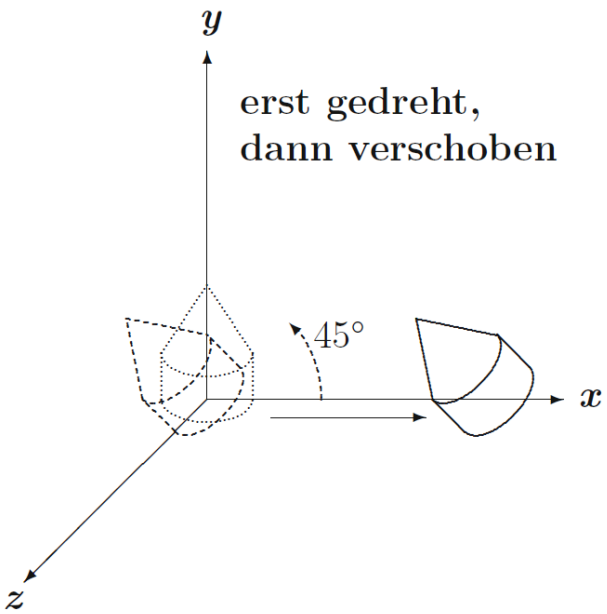
Beispiel für $S_x = 1, S_y = 0.5, S_z = 1$

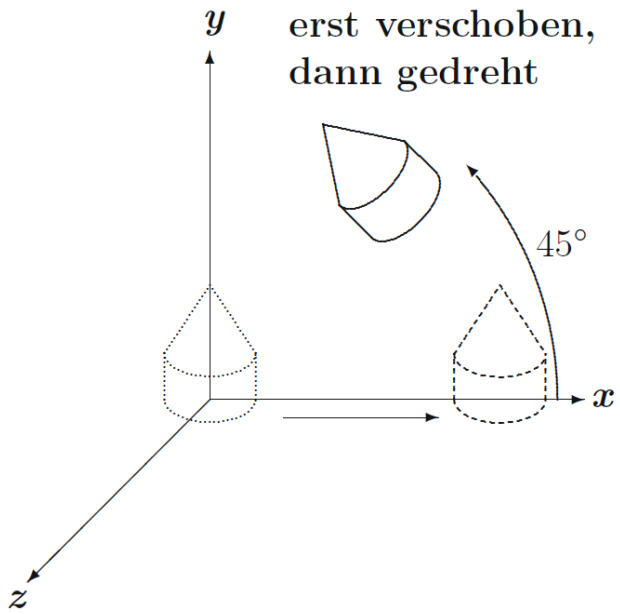
Skalierung

- $S_x = S_y = S_z$ uniforme Skalierung
- $S < 0$ Spiegelung an der jeweiligen Achse
- $S = 0$ unzulässiger Wert
- $|S| = 1$ Dimension unverändert (aber evtl. Spiegelung wenn $S = -1$)
- $0 < |S| < 1$ Stauchung, Dimension wird verkleinert
- $|S| > 1$ Streckung, Dimension wird vergrößert

Reihenfolge der Transformationen

- Reihenfolge der Transformationen ist wichtig, da sie nicht unabhängig von einander sind
- Unterscheidung zweier Denkweisen:
  1. Transformation eines Objektes im Weltkoordinatensystem

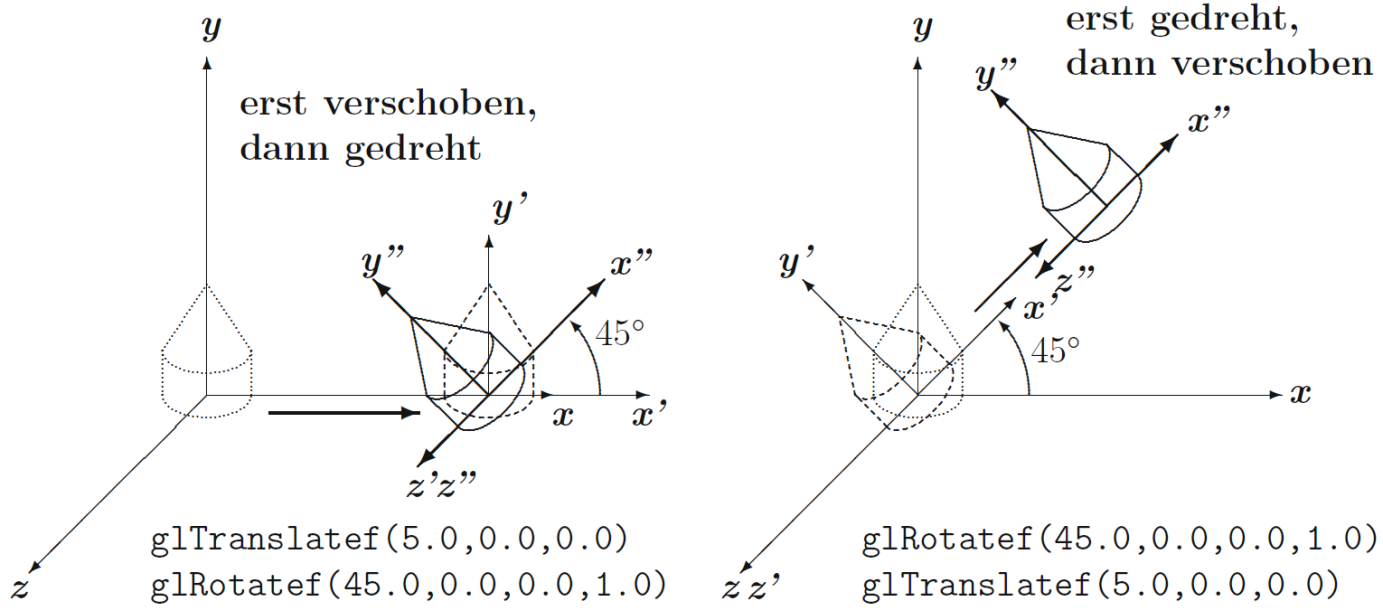




A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Reihenfolge der Transformationen

2. Transformation des lokalen Koordinatensystem im Weltkoordinatensystem



- Verwendung 2. Variante im Programmcode: Verkettung durch Multiplikation auf vorhandene Matrix (=Rechtsmultiplikation) → folgt der Denkweise „Transformation des lokales Koordinatensystem“

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Reihenfolge der Transformationen

- Implementierungsbeispiel:

```
glLoadIdentity();
glTranslatef(…);
glRotatef(…);
glBegin(GL_TRIANGLES);
glVertexfv(v);
:
glEnd();
```

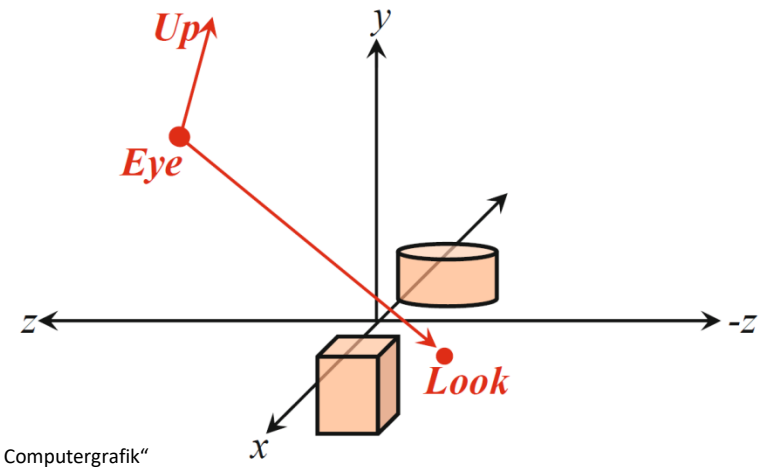
1. Laden der Identitätsmatrix $\boldsymbol{I}$ zur Initialisierung
2. Multiplikation der Translationsmatrix $\boldsymbol{T}$ von rechts auf $\boldsymbol{I}$: $\boldsymbol{I} \cdot \boldsymbol{T} = \boldsymbol{T}$
3. Multiplikation der Rotationsmatrix $\boldsymbol{R}$ von rechts auf $\boldsymbol{T}$: $\boldsymbol{T} \cdot \boldsymbol{R}$
4. Multiplikation der Vertices v von rechts auf $\boldsymbol{TR}$: $\boldsymbol{T} \cdot \boldsymbol{R} \cdot v$

   Dies entspricht: $\boldsymbol{T}(\boldsymbol{R}v) \Rightarrow v' = \boldsymbol{R}v, v'' = \boldsymbol{T}v'$

Augpunkttransformationen

- Ändern der Position und der Blickrichtung des Augpunktes
  - = Positionierung der Kamera
  - In OpenGL standardm. in Punkt $(0,0,0)$, Blick entlang der negativen z-Achse
- Gleichbedeutend mit Verschiebung der Szene (Modelltransformation)
- Zusammenfassung in einer Modelview-Matrix
- Augpunkttransformationen immer vor allen Transformationen ausführen!
- In OpenGL:

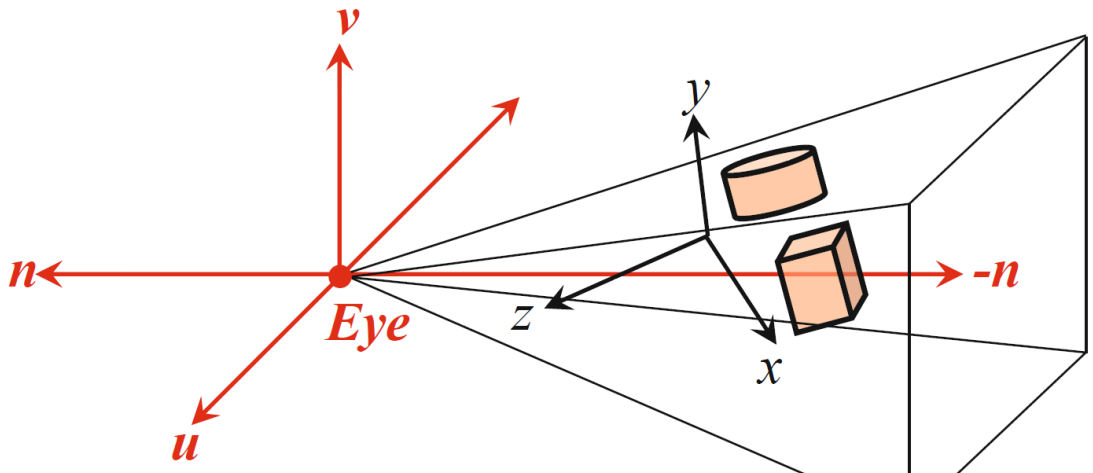
```
gluLookAt(Eye.x, Eye.y, Eye.z, Look.x, Look.y, Look.z, Up.x, Up.y, Up.z);
```



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Augpunkttransformationen

- **`gluLookAt`** Transformation ist zweiteilig:
  1. Drehung im Raum so dass Sichtachse n der z-Achse entspricht und gleichzeitig der Vektor nach oben (v, $\boldsymbol{Up}$) parallel zur y-Achse ausgerichtet ist
  2. Verschiebung des Ortsvektors $\boldsymbol{Eye}$ in den Ursprung $(0,0,0)$

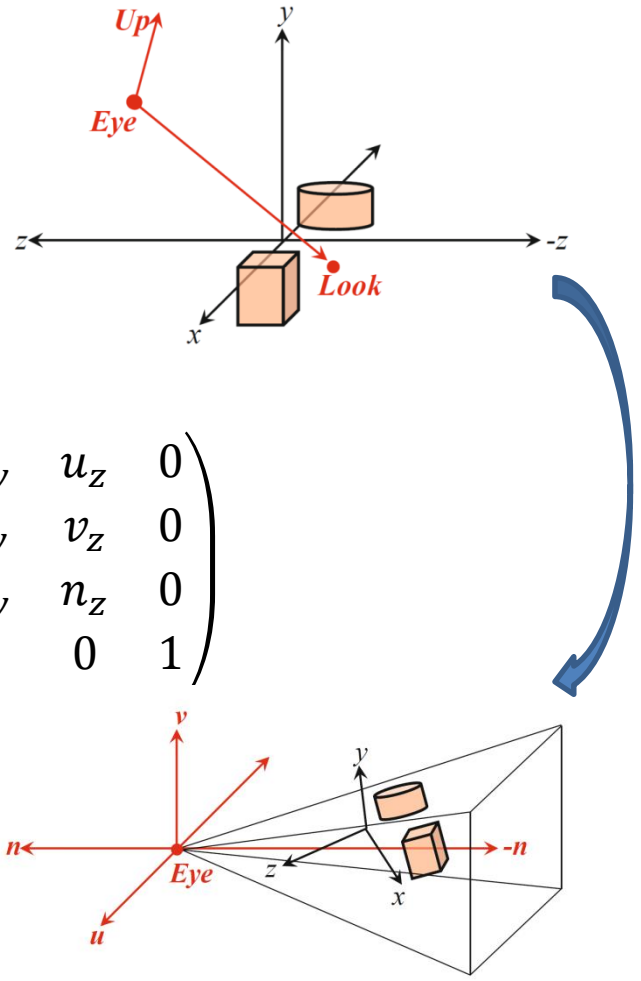


- Randbedingungen:
  - $\boldsymbol{Eye}$ und $\boldsymbol{Look}$ dürfen nicht identisch sein.
  - $\boldsymbol{Up}$ darf nicht parallel zu n gewählt werden.

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Augpunkttransformationen

1. Berechnung der Rotationsmatrix:
   - $-n = \boldsymbol{Eye} - \boldsymbol{Look}$
   - $u, v \perp n$, $u \perp v$, v muss nach oben zeigen
     a.) $u = \boldsymbol{Up} \times n$
     b.) $v = n \times u$
   - Normierung der Länge von n, u, v auf 1.
   - Rotationsmatrix ergibt sich als: $\mathbf{M}_R = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$



Augpunkttransformationen

2. Berechnung der Verschiebung:

- Einfache Translation um $-\boldsymbol{Eye}$ reicht nicht aus, da i.d.R. das Koordinatensystem gedreht wurde.
- Multiplikation der gesamten Augpunkt-Transformationsmatrix $\boldsymbol{M_{RT}}$ mit $\boldsymbol{Eye}$, gleichsetzen mit gewünschtem Ursprung in homogenen Koordinaten $(0,0,0,1)$

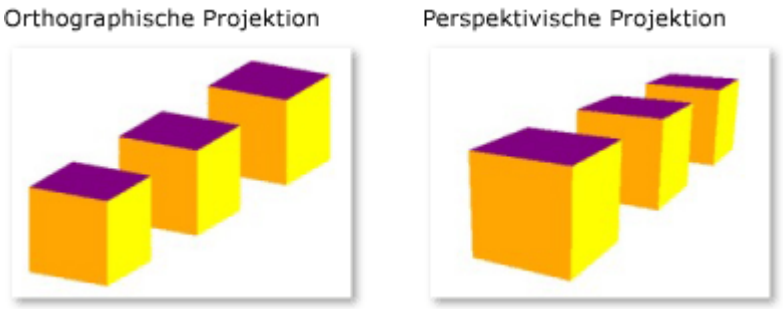
$$\begin{pmatrix} u_x & u_y & u_z & t_x \\ v_x & v_y & v_z & t_y \\ n_x & n_y & n_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} Eye.x \\ Eye.y \\ Eye.z \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} t_x = -u \cdot Eye \\ t_y = -v \cdot Eye \\ t_z = -n \cdot Eye \end{pmatrix}$$

4.4. PROJEKTIONSTRANSFORMATIONEN

Projektions-Transformationen

- Abbildung der 3D-Szene auf den zweidimensionalen Raum
- Projektionsarten:
  - Orthografische Projektion (auch: Parallelprojektion)
  - Perspektivische Projektion (auch: Zentralprojektion)



- Wegfall einer Dimension, z.B. $(x, y, z) \rightarrow (x, y)$
- z-Werte werden aber weiterhin normiert gespeichert für spätere Verarbeitungsschritte (z.B. Verdeckungsberechnung)

https://msdn.microsoft.com/de-de/library/system.windows.media.media3d.orthographiccamera%28v=vs.110%29.aspx

Orthografische Projektion

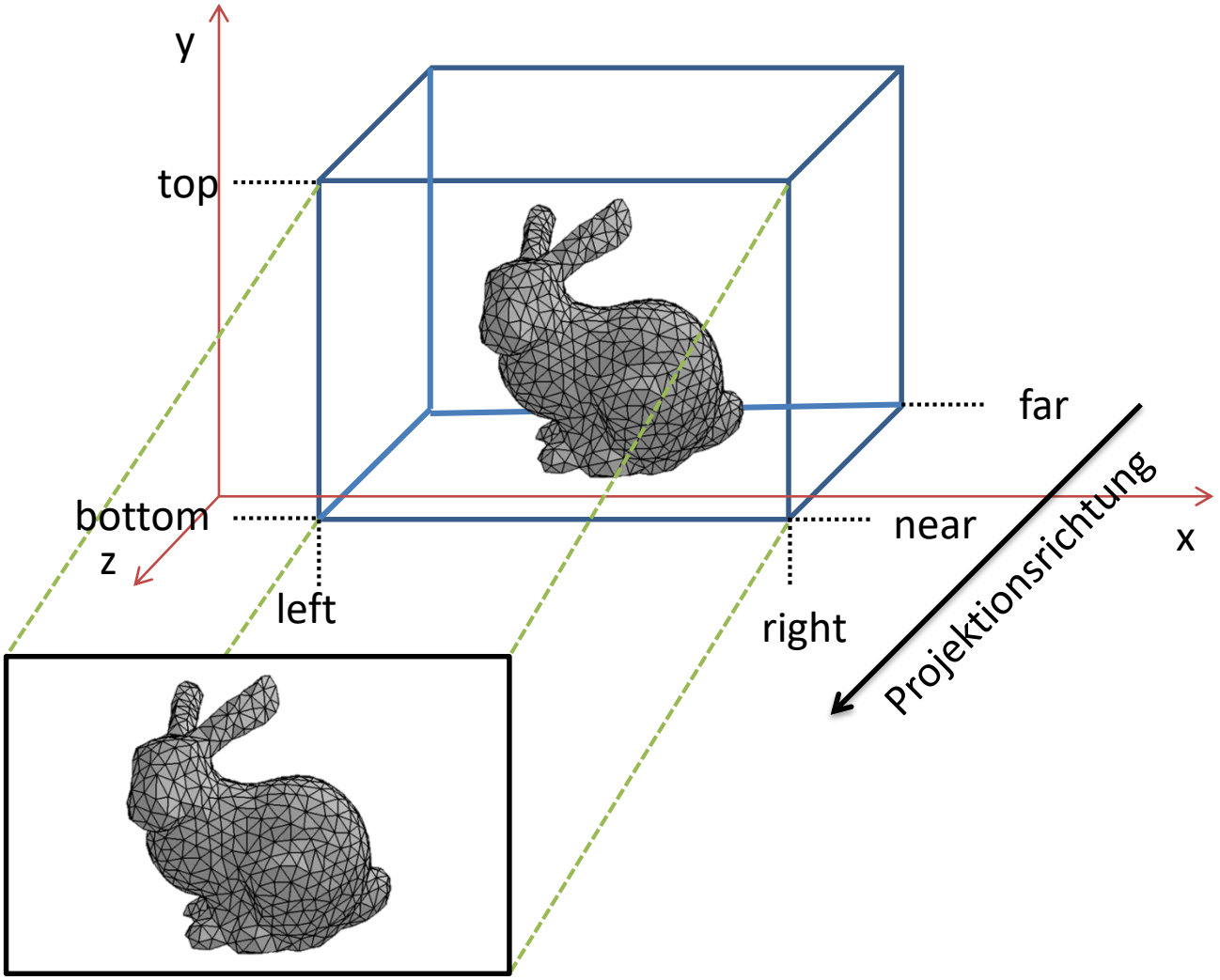
- Parallele Strahlen von Objekten zur Bildfläche
  - Größen und Winkel aller Objekte bleiben erhalten
- Ausschnitt aus der Szene durch *Clipping Planes.*
- In OpenGL:

```
glOrtho( GLdouble left, GLdouble right, Gldouble bottom,
         GLdouble top,  Gldouble near,  Gldouble far);
```

- Transformationsmatrix der orthografischen Projektion: Identitätsmatrix

$$\mathbf{P}_{ortho} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Orthografische Projektion



Perspektivische Projektion

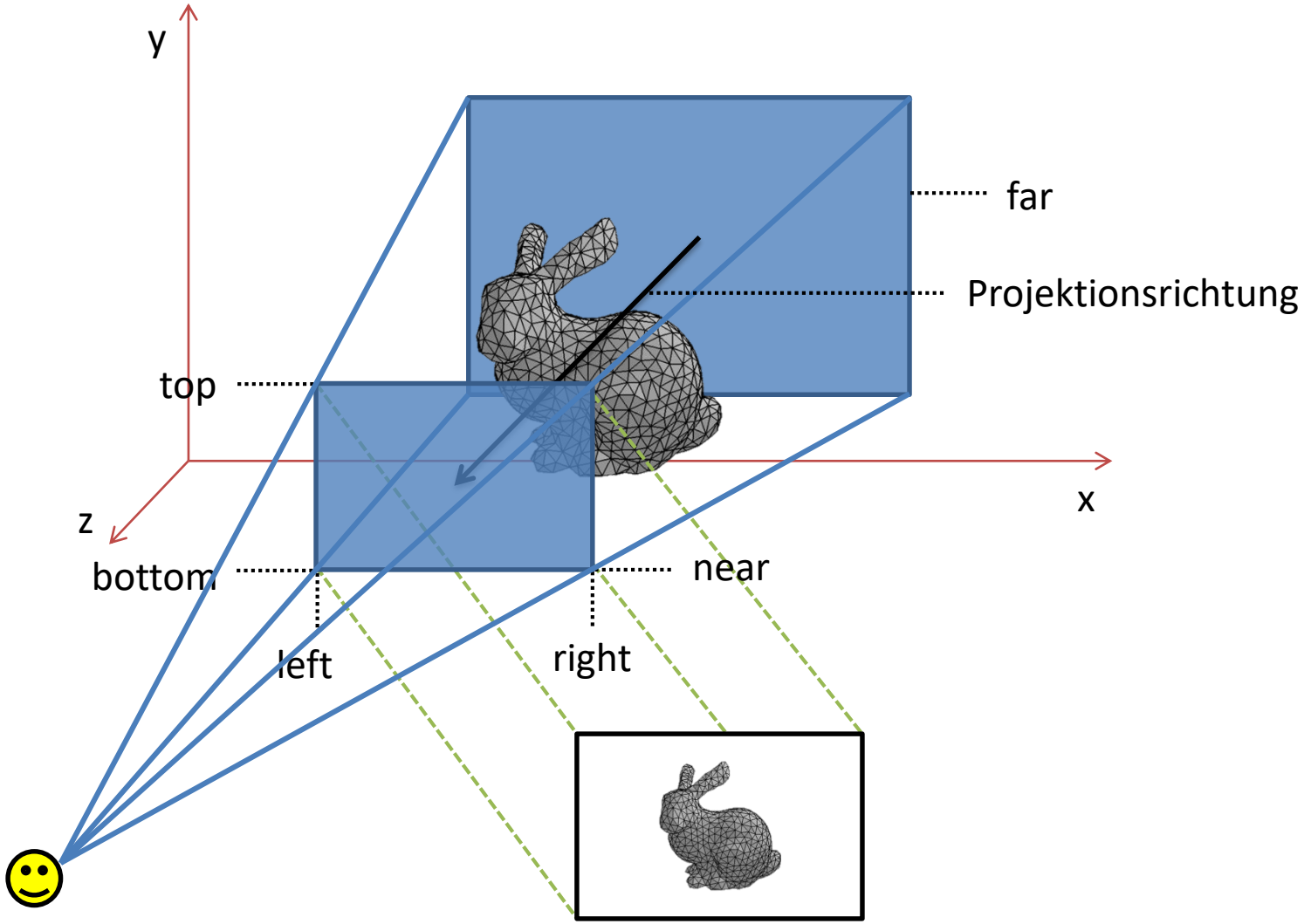
- Konvergierende Strahlen von allen sichtbaren Objekten zum Augpunkt
- Objekte nah am Augpunkt erscheinen größer als entfernte Objekte
- Ausschnitt aus der Szene durch einen Kegelstumpf, definiert durch *Clipping planes*
- In OpenGL:

```
glFrustum( GLdouble left, GLdouble right, Gldouble bottom,
           GLdouble top,  Gldouble near,  Gldouble far);
```

- Transformationsmatrix der perspektivischen Projektion:

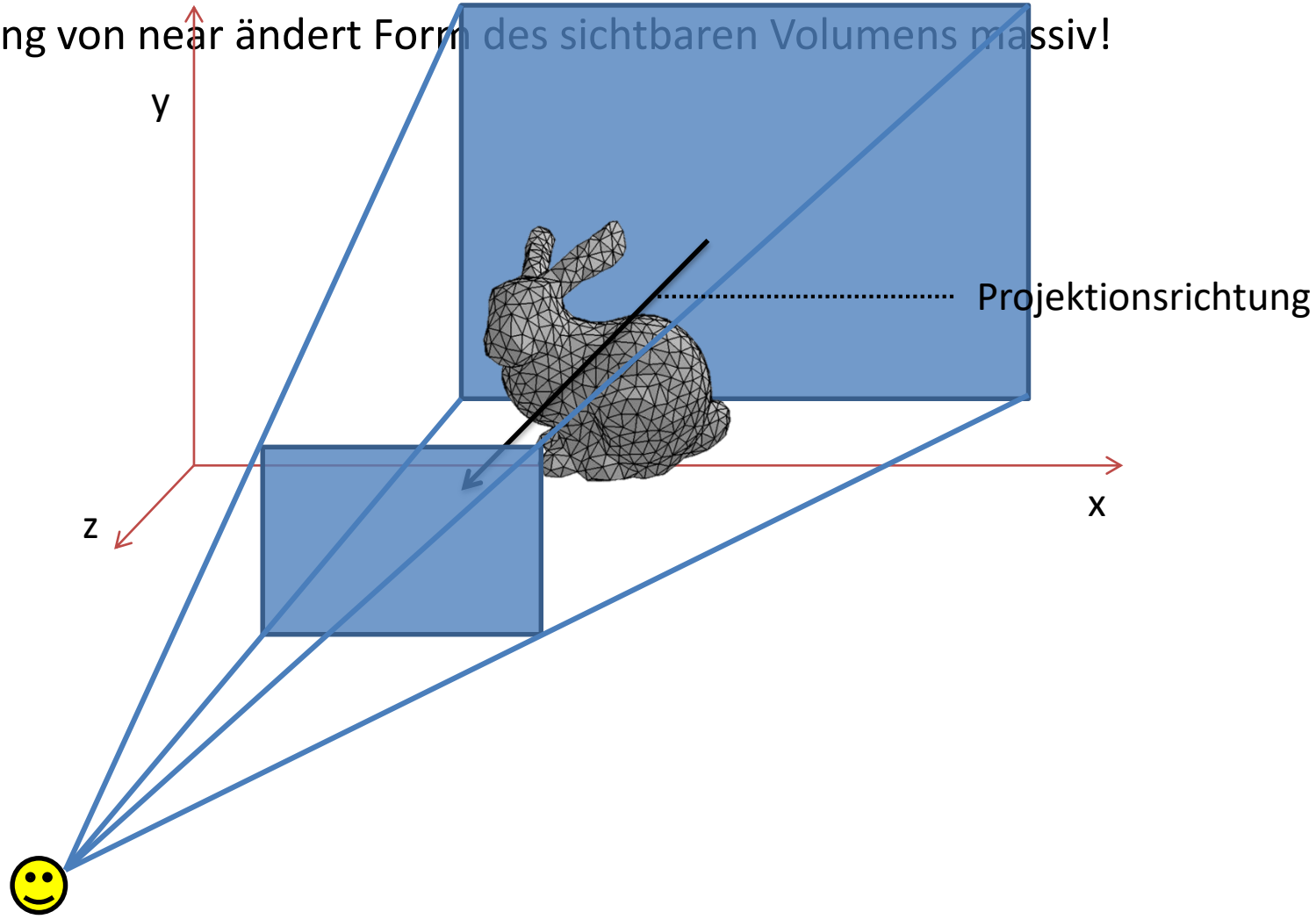
$$\mathbf{P}_{persp.} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 + \frac{far}{near} & far \\ 0 & 0 & -\frac{1}{near} & 0 \end{pmatrix}$$

Perspektivische Projektion



Perspektivische Projektion

- Änderung von near ändert Form des sichtbaren Volumens massiv!



Perspektivische Projektion

- **`glFrustum`** oft etwas umständlich: left, right, bottom, top müssen z.B. erst aus Blickwinkeln berechnet werden.
- OpenGL Utility Library definiert bequemeren Befehl für solche Fälle:

```
gluPerspective(GLdouble alpha, GLdouble aspect,
               GLdouble near, GLdouble far);
```

  - α: Vertikaler Blickwinkel im Wertebereich $[0, 180]°$
  - aspect: Verhältnis zwischen vertikalem (α) und horizontalen (β) Blickwinkel: β/α.

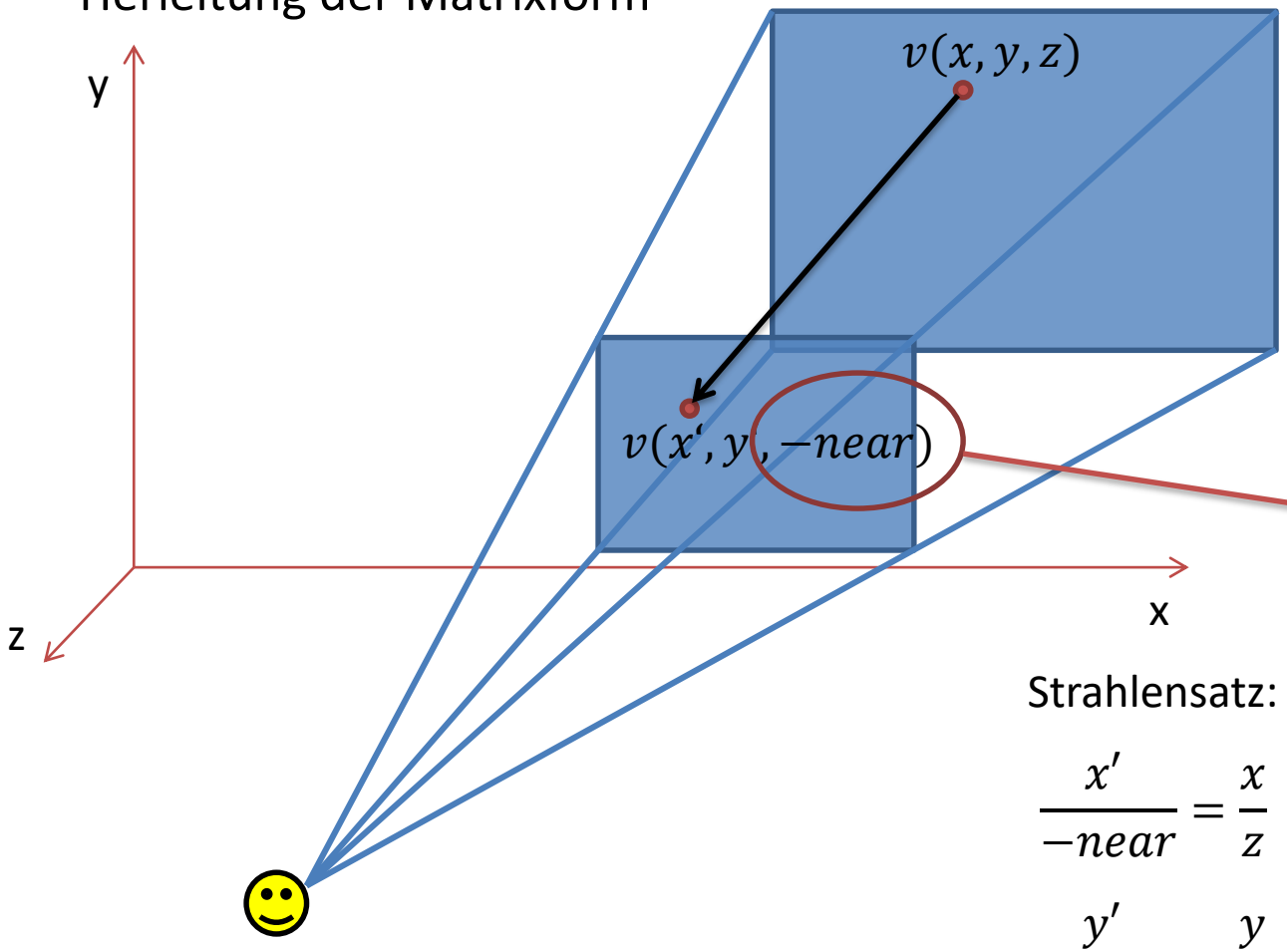
- Berechnung von left, right, top, bottom:

$$left = -near \cdot \tan\frac{\beta}{2}$$
$$right = near \cdot \tan\frac{\beta}{2}$$
$$bottom = -near \cdot \tan\frac{\alpha}{2}$$
$$top = near \cdot \tan\frac{\alpha}{2}$$

- Einschränkung: Blickwinkel müssen nach links/rechts, oben/unten symmetrisch sein!

Perspektivische Projektion

- Herleitung der Matrixform



Beachte hier Vorzeichen!
$near$ entspricht Abstand vom Augpunkt (entlang der negativen z-Richtung)

Strahlensatz:

$$\frac{x'}{-near} = \frac{x}{z} \Leftrightarrow x' = -\frac{near}{z}x$$

$$\frac{y'}{-near} = \frac{y}{z} \Leftrightarrow y' = -\frac{near}{z}y$$

Perspektivische Projektion

- Perspektivische Projektionsmatrix erfüllt die Strahlensatzgleichungen:

$$v' = \mathbf{P}v \Leftrightarrow \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 + \frac{far}{near} & far \\ 0 & 0 & -\frac{1}{near} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x \\ y \\ \left(1 + \frac{far}{near}\right) z + far \cdot w \\ -\frac{z}{near} \end{pmatrix}$$

- Dividieren durch den inversen Streckungsfaktor ergibt die euklidischen Koordinaten:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} -\frac{near}{z} x \\ -\frac{near}{z} y \\ -\frac{near}{z} far \cdot w - (far + near) \end{pmatrix}$$

Siehe Strahlensatz auf der letzten Folie!

Normierung

- Abbildung der *near clipping plane* auf Bildschirmfenster: Zwischenschritt Normierung
  - Division durch halbe Ausdehnung der *near clipping plane*
  - Verschieben des Zentrums des Wertebereichs in den Ursprung

$$x' = \frac{2}{right - left}\, x - \frac{right + left}{right - left}$$

$$y' = \frac{2}{top - bottom}\, y - \frac{top + bottom}{top - bottom}$$

$$z' = \frac{-2}{far - near}\, z - \frac{far + near}{far - near}$$

$$w' = w$$

  - Wertebereich nach der Transformation: $[-w\ w]$
  - Division durch w bildet x, y, z-Werte auf Intervall $[-1\ 1]$ ab.

Normierung

- Erzielt Unabhängigkeit von Größe des sichtbaren Volumens
- In Matrix-Schreibweise:

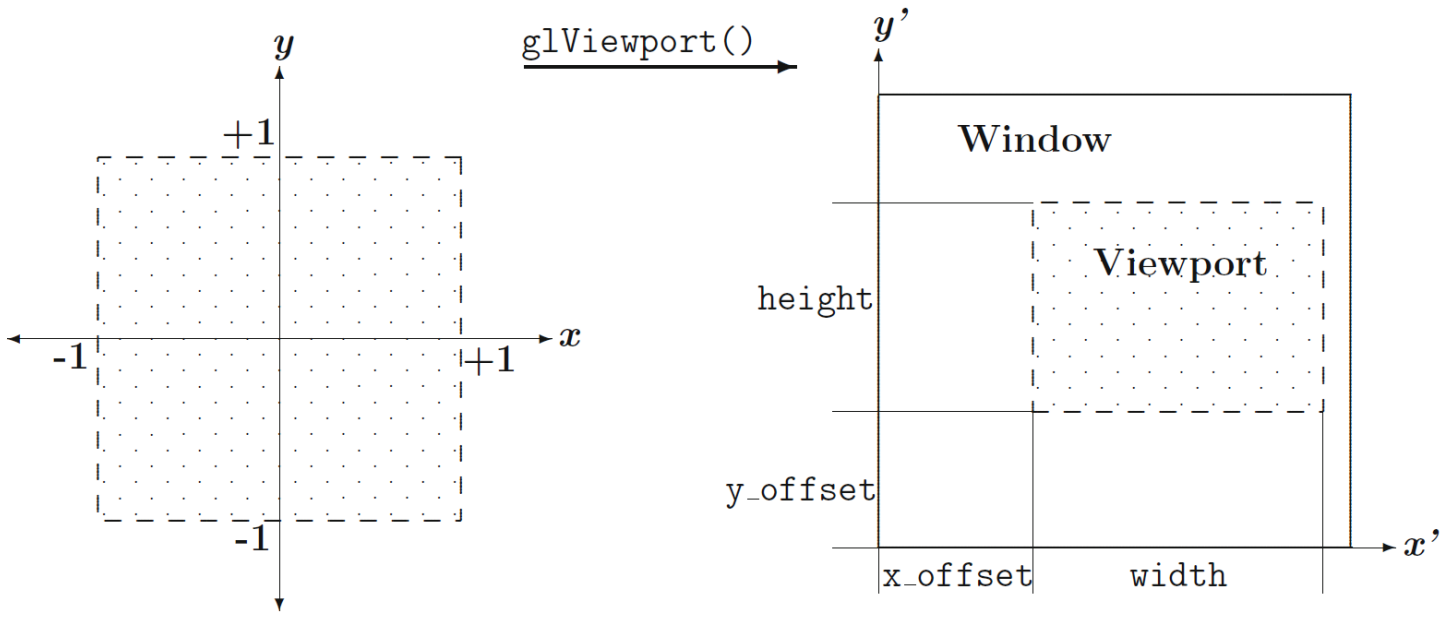
$$\mathbf{N} = \begin{pmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- **`glOrtho()`**, **`glFrustum()`** und **`gluPerspective()`** führen Projektion und Normierung gemeinsam durch!

4.5. VIEWPORT-TRANSFORMATIONEN

Viewport-Transformation

- Abbildung der Szene auf Ausschnitt des Bildschirms (*Viewport*)
- Viewport definiert in Pixeln
  - Startpunkt (x_{Offset}, y_{Offset})
  - Ausdehnung (width, height)



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Viewport-Transformation

- Umrechnung aus den normierten projizierten Daten (für $w = 1$):

$$x' = \frac{width}{2}x + \left(x_{\text{offset}} + \frac{width}{2}\right)$$
$$y' = \frac{height}{2}y + \left(y_{\text{offset}} + \frac{height}{2}\right)$$

- In Matrix-Schreibweise:

$$\mathbf{V} = \begin{pmatrix} \frac{width}{2} & 0 & 0 & x_{\text{offset}} + \frac{width}{2} \\ 0 & \frac{height}{2} & 0 & y_{\text{offset}} + \frac{height}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

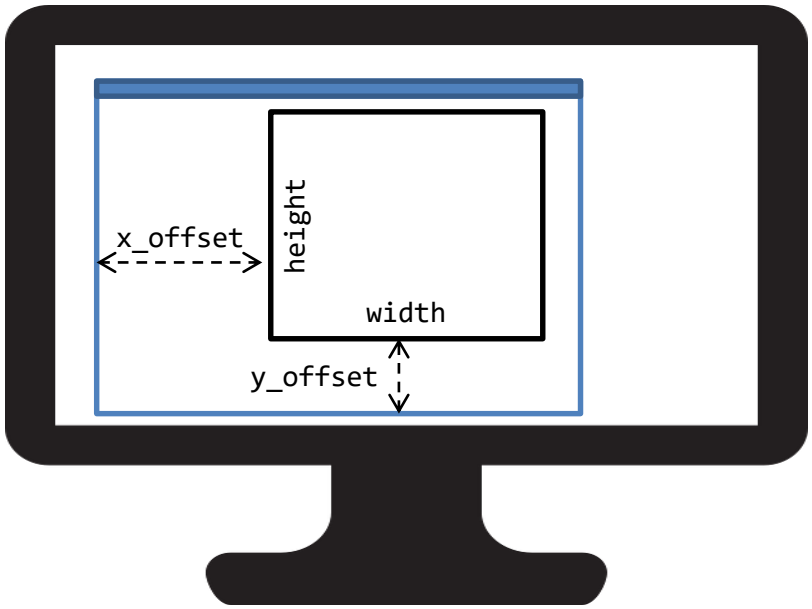
- Die z-Koordinate normiert auf den Wertebereich $[-1\ 1]$ wird erhalten um spätere Auswertungen durchführen zu können (z.B. Verdeckungsberechnung)

Viewport-Transformation

- Umsetzung in OpenGL:

```
glViewport(x_offset, y_offset, width, height);
```

- Größen in Pixel
- Offset bezieht sich auf Fenster
- Wenn nicht explizit angeben: Viewport = Window.
- Window kann mehrere Viewports enthalten
- Aspektverhältnis des Viewports muss dem des sichtbaren Volumens entsprechen, sonst werden Objekte gestaucht/gedehnt dargestellt.



4.6. MATRIZENSTAPEL

Matrizenstapel

- Abarbeitungsreihenfolge: Jeder Vertex durchläuft alle Transformationsstufen

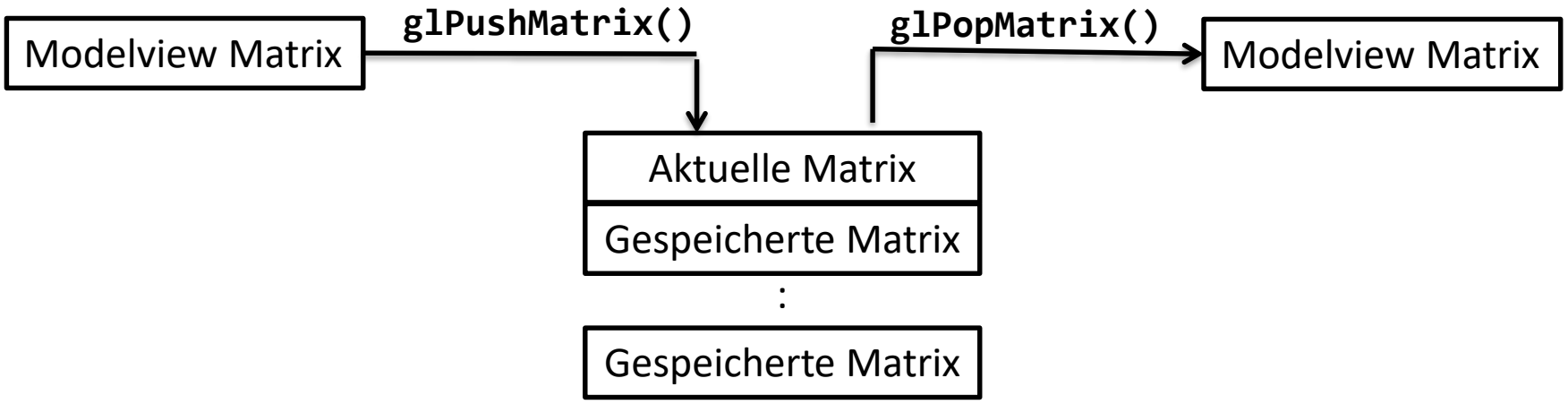
$$v' = \mathbf{V}(\mathbf{N}(\mathbf{P}(\mathbf{S}(\mathbf{R}(\mathbf{T} \cdot v)))))$$

  - Hoher Aufwand, da große Zwischenergebnisse gespeichert werden müssen.
  - Effizienterer Weg:

$$v' = (\mathbf{V} \cdot \mathbf{N} \cdot \mathbf{P} \cdot \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{T}) \cdot v$$

- Meist mehrere Matrizen vorhanden, z.B. zur Wiederverwendung von Objekten → Matrizenstapel zur Zwischenspeicherung von Matrizen
- In OpenGL:
  - „Modelview-Matrizen“: bis 32 Matrizen (**`glMatrixMode(GL_MODELVIEW)`**)
  - „Projektions-Matrizen“: bis 2 Matrizen (**`glMatrixMode(GL_PROJECTION)`**)

Matrizenstapel



- **`glPushMatrix:`**
  - Anfertigen einer Kopie der aktuellen Matrix + auf dem Stapel speichern.
  - Weitere Transformationen können hinzugefügt werden (Multiplikation auf bisher aktuelle Matrix).
- **`glPopMatrix:`**
  - Entfernt aktuelle Matrix und kehrt zur letzten auf dem Stapel zurück.

Matrizenstapel

- Beispielhafte Benutzung der Matrizenstapel:
  - Projektions-Matrizen-Stapel: Umschalten zwischen orthografischer und perspektivischer Projektion
  - Modelview-Matrizen-Stapel: Wiederverwendung von Objekten

```
glLoadIdentity();
glTranslatef(…);

for(int i=0; i<5; i++) {
    glPushMatrix()  // - ab hier arbeiten im „lokalen“ Koordinatensystem
    glRotatef(…);
    glTranslatef(…);
    glBegin(GL_TRIANGLES);  // Objektdefinition
    glVertexfv(v);
                :
    glEnd();
    glPopMatrix();
}
```

ZUSAMMENFASSUNG

Zusammenfassung

- Transformationskette von lokalen Koordinaten bis Bildschirmkoordinaten
- Verwendung von homogenen Koordinaten in allen Transformationen
  - Transformationsmatrizen immer 4x4.
- Modelltransformationen: Positionierung von Objekten in Weltkoordinaten
  - Translation, Rotation, Skalierung
  - Reihenfolge der Transformationen wichtig
- Augpunkttransformationen zur Positionierung des Beobachters
  - Lassen sich in Modelltransformationen überführen
- Projektionstransformation zur Abbildung der 3D-Szene in 2D
  - Orthografische oder perspektivische Projektion
  - Normierung der Koordinaten
- Viewport-Transformationen: Positionierung auf dem Bildschirm bzw. im Fenster
- Wiederverwendung von Matrizen/Transformationen durch Matrizenstapel

ÜBUNGSAUFGABEN

Übungsaufgaben

Modelltransformationen

1. Drücken Sie die folgenden affinen 3D Transformationen durch jeweils eine entsprechende homogene 4×4 Matrix aus:
   - Rotation um $90°$um die X-Achse
   - Translation um den Vektor $(0,5,0)$
2. Bestimmen Sie die zusammengesetzte Transformationsmatrix wie sie entstehen würde, wenn im Programmcode zunächst die Rotation, dann die Translation ausgeführt wird.
3. Transformieren Sie den Punkt $v = (1,0,0)$ mit der zusammengesetzten Matrix.
4. Welche Matrix und welcher Punkt ergibt sich, wenn die beiden Transformationen umgekehrt ausgeführt werden?

Lösung

1. **Drücken Sie die folgenden affinen 3D Transformationen durch jeweils eine entsprechende homogene 4×4 Matrix aus:**
   - Rotation um $90°$ um die X-Achse
   - Translation um den Vektor $(0,5,0)$

Rotation $90°$ um x-Achse: $\alpha = 90°$

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(90°) & -\sin(90°) & 0 \\ 0 & \sin(90°) & \cos(90°) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Translation um den Vektor $(0,5,0)$

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Lösung (2)

2. Bestimmen Sie die zusammengesetzte Transformationsmatrix wie sie entstehen würde, wenn im Programmcode zunächst die Rotation, dann die Translation ausgeführt wird.

Die Gesamtmatrix setzt sich durch Rechtsmultiplikation von $\boldsymbol{T}$ an $\boldsymbol{R}$ zusammen:

$$\mathbf{M} = \mathbf{R} \cdot \mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Transformieren Sie den Punkt $\boldsymbol{v} = (\mathbf{1}, \mathbf{0}, \mathbf{0})$ mit der zusammengesetzten Matrix.

Erweiterung auf homogene Koordinaten und Rechtsmultiplikation des Vektors an die Matrix M

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 5 \\ 1 \end{pmatrix} \Rightarrow v' = \begin{pmatrix} 1 \\ 0 \\ 5 \end{pmatrix}$$

Lösung (3)

4. Welche Matrix und welcher Punkt ergibt sich, wenn die beiden Transformationen umgekehrt ausgeführt werden?

Die Gesamtmatrix setzt sich durch Rechtsmultiplikation von $\boldsymbol{R}$ an $\boldsymbol{T}$ zusammen:

$$\mathbf{M} = \mathbf{T} \cdot \mathbf{R} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Erweiterung auf homogene Koordinaten und Rechtsmultiplikation des Vektors an die Matrix $\boldsymbol{M}$

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 5 \\ 0 \\ 1 \end{pmatrix} \Rightarrow v' = \begin{pmatrix} 1 \\ 5 \\ 0 \end{pmatrix}$$

Übungsaufgaben

Projektionstransformation

Nach Modelltransformation und Augpunkttransformation liegt ein Polygon wie folgt vor: $v_0 = (1 \,/\, -1 \,/\, -2{,}5)$, $v_1 = (-2 \,/\ 2{,}5 \,/\ -3)$, $v_2 = (-1 \,/\, 4 \,/\, -3)$. Mit **`gluPerspective`** ist eine perspektivische Projektion mit folgenden Parametern definiert: $\alpha = 90°$, $\text{aspect} = 1$. Der Abstand vom Augpunkt zur near clipping plane beträgt 2, der Abstand zur far clipping plane beträgt 4.

1. Berechnen Sie left, right, top, bottom.
2. Geben Sie die Projektionsmatrix für eine perspektivische Projektion mit den gegebenen Werten an.
3. Welche kartesischen Koordinaten haben die Vertices nach einer Projektion mit den gegebenen Werten?
4. Welche Vertices liegen innerhalb des Darstellungsbereichs?
5. Fertigen Sie eine Skizze des Darstellungsbereichs an und zeichnen Sie das projizierte Dreieck ein.
6. Führen Sie gegebenenfalls ein Clipping des projizierten Dreiecks am Darstellungsbereich durch. Welche Schnittpunkte mit dem Darstellungsbereich entstehen? *(Kapitel 5)*
7. Führen Sie eine Normalisierung der Koordinaten auf den Wertebereich $[-1\ 1]$ durch. Wie sieht die Normierungsmatrix aus?

Lösung

1. Berechnen Sie left, right, top, bottom.

β lässt sich aus α und *aspect* berechnen: $aspect = \frac{\beta}{\alpha} \Rightarrow \beta = aspect \cdot \alpha = 1 \cdot \alpha = 90°$

$$left = -near \cdot \tan\left(\frac{\beta}{2}\right) = -2 \cdot \tan(45°) = -2$$

$$right = near \cdot \tan\left(\frac{\beta}{2}\right) = 2 \cdot \tan(45°) = 2$$

$$bottom = -near \cdot \tan\left(\frac{\alpha}{2}\right) = -2 \cdot \tan(45°) = -2$$

$$top = near \cdot \tan\left(\frac{\alpha}{2}\right) = 2 \cdot \tan(45°) = 2$$

2. Geben Sie die Projektionsmatrix für eine perspektivische Projektion mit den gegebenen Werten an.

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 + \frac{far}{near} & far \\ 0 & 0 & -\frac{1}{near} & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & -0{,}5 & 0 \end{pmatrix}$$

Lösung (2)

3. Welche kartesischen Koordinaten haben die Vertices nach einer Projektion mit den gegebenen Werten?

$$v_0{}' = \mathbf{P} \cdot v_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & -0{,}5 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \\ -2{,}5 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ -3{,}5 \\ 1{,}25 \end{pmatrix}$$

Errechnen der kartesischen Koordinaten durch Divisionen durch den inversen Streckungsfaktor *w*:

$$v_0{}'' = \begin{pmatrix} \frac{1}{1{,}25} \\ \frac{-1}{1{,}25} \\ \frac{-3{,}5}{1{,}25} \\ 1 \end{pmatrix} = \begin{pmatrix} 0{,}8 \\ -0{,}8 \\ -2{,}8 \\ 1 \end{pmatrix} \Rightarrow v_0{}'' = \begin{pmatrix} 0{,}8 \\ -0{,}8 \\ -2{,}8 \end{pmatrix}$$

Analog ergeben sich:

$$v_1{}'' = \begin{pmatrix} -1{,}33 \\ 1{,}67 \\ -3{,}33 \end{pmatrix} \qquad v_2{}'' = \begin{pmatrix} -0{,}67 \\ 2{,}67 \\ -3{,}33 \end{pmatrix}$$

Lösung (3)

4. Welche Vertices liegen innerhalb des Darstellungsbereichs?

Der Darstellungsbereich ist die near clipping plane, auf die die Szene projiziert wird. Nach der Projektion liegen alle Punkte in dieser Ebene, die z-Koordinate wird ignoriert bzw. nur für spätere Berechnungen weiterhin mitgeführt. Es muss also getestet werden ob die x- und y-Koordinate der projizierten Vertices zwischen [left right] bzw [bottom top] liegen.

v_0'': $left \leq v_{0,x}'' \leq right$ und $bottom \leq v_{0,y}'' \leq top$

→ v_0 liegt im Darstellungsbereich

v_1'': $left \leq v_{1,x}'' \leq right$ und $bottom \leq v_{1,y}'' \leq top$

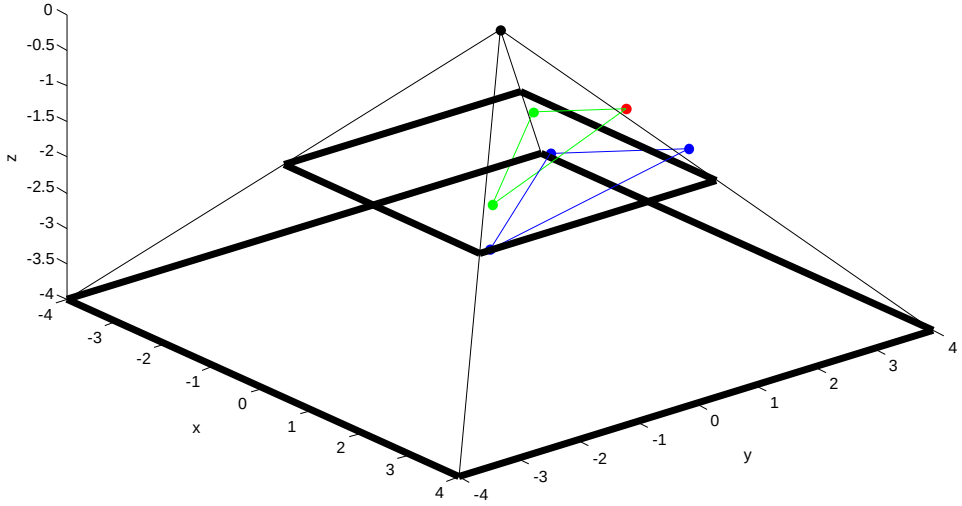
→ v_1 liegt im Darstellungsbereich

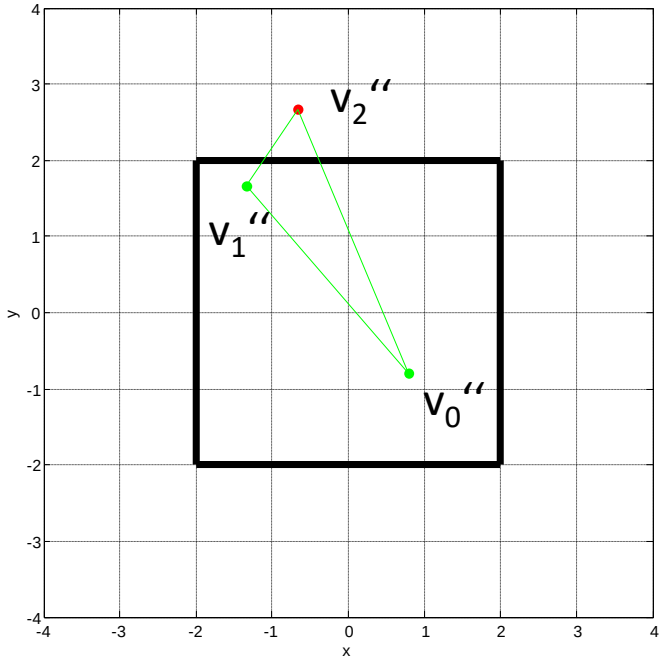
v_2'': $left \leq v_{2,x}'' \leq right$ und $v_{2,y}'' \geq top$

→ v_2 liegt nicht im Darstellungsbereich

Lösung (4)

5. Fertigen Sie eine Skizze des Darstellungsbereichs an und zeichnen Sie das projizierte Dreieck ein.





Lösung (5)

6. Führen Sie gegebenenfalls ein Clipping des projizierten Dreiecks am Darstellungsbereich durch. Welche Schnittpunkte mit dem Darstellungsbereich entstehen?

Anhand der Skizze kann abgelesen werden, dass die Kanten v_1‘ $\rightarrow$ v_2‘ und v_0‘ $\rightarrow$ v_2‘ den Darstellungsbereich an der Kante top schneiden. Dadurch ist die y-Koordinate der Schnittpunkte bereits bekannt: $s_{1,y} = 2$, $s_{2,y} = 2$. Die x-Koordinate lässt sich berechnen durch

$$s_x = v_{1,x} + \frac{s_y - v_{1,y}}{v_{2,y} - v_{1,y}} (v_{2,x} - v_{1,x})$$

Somit ergeben sich die folgenden Schnittpunkte (der einfacheren Schreibweise halber sei $v_1 = v_1$‘ usw.):

$$s_{1,x} = v_{1,x} + \frac{s_{1,y} - v_{1,y}}{v_{2,y} - v_{1,y}} \left(v_{2,x} - v_{1,x}\right) = -1{,}33 + \frac{2 - 1{,}67}{2{,}67 - 1{,}67} (-0{,}67 + 1{,}33) = -1{,}11$$

$$s_{2,x} = v_{0,x} + \frac{s_{2,y} - v_{0,y}}{v_{2,y} - v_{0,y}} \left(v_{2,x} - v_{0,x}\right) = 0{,}8 + \frac{2 + 0{,}8}{2{,}67 + 0{,}8} (-0{,}67 - 0{,}8) = -0.39$$

Die Schnittpunkte sind damit $s_1 = (-1{,}11 \,/\, 2)$ und $s_2 = (-0{,}39 \,/\, 2)$

Lösung (6)

7. Führen Sie eine Normalisierung der Koordinaten auf den Wertebereich $[-1\ 1]$ durch. Wie sieht die Normierungsmatrix aus?

$$\mathbf{N} = \begin{pmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0{,}5 & 0 & 0 & 0 \\ 0 & 0{,}5 & 0 & 0 \\ 0 & 0 & -1 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$v''_{0,norm} = \mathbf{N} \cdot v''_0 = \begin{pmatrix} 0{,}5 & 0 & 0 & 0 \\ 0 & 0{,}5 & 0 & 0 \\ 0 & 0 & -1 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0{,}8 \\ -0{,}8 \\ -2{,}8 \\ 1 \end{pmatrix} = \begin{pmatrix} 0{,}4 \\ -0{,}4 \\ -0{,}2 \\ 1 \end{pmatrix}$$

Analog ergeben sich

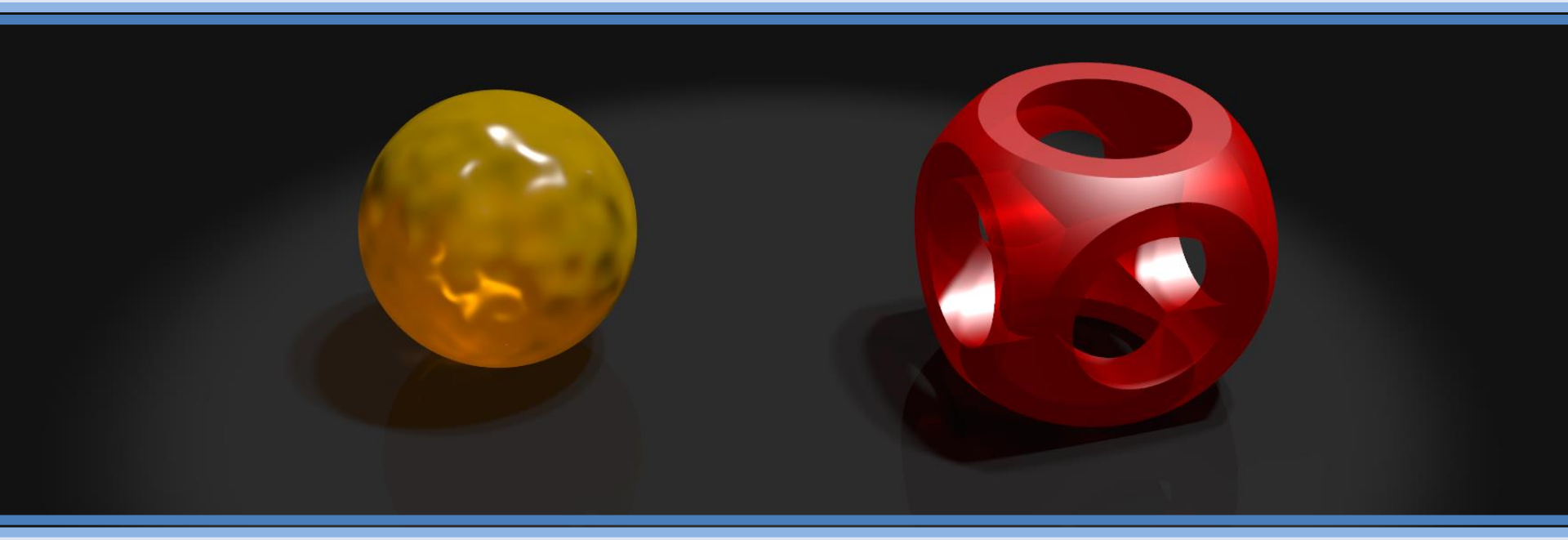
$$v''_{1,norm} = \begin{pmatrix} -0{,}67 \\ 0{,}83 \\ 0{,}33 \\ 1 \end{pmatrix} \qquad v''_{2,norm} = \begin{pmatrix} -0{,}33 \\ 1{,}33 \\ 0{,}33 \\ 1 \end{pmatrix}$$

Übungsfragen Kapitel 4

- Welche Koordinatensysteme und Transformationen kommen in der Transformationskette vor? Geben Sie die Reihenfolge der Abarbeitung an.
- Warum werden Modelltransformation und Augpunkt-Transformation meist in einer gemeinsamen Modelview-Matrix zusammengefasst?
- Was ist der Matrizenstapel? Wofür wird er benutzt?

Computergrafik

T. Hopp

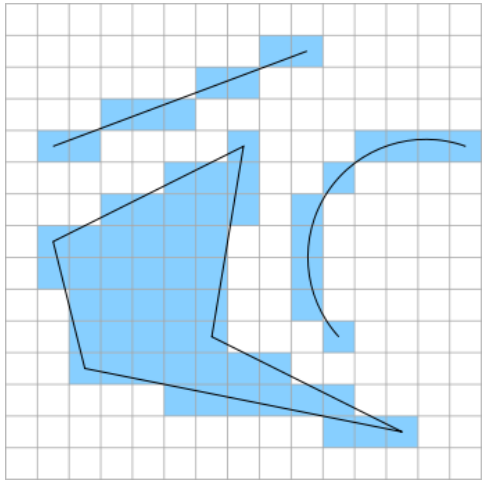


Themenübersicht

1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. **Zeichenalgorithmen**
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. Raytracing
11. Volumenvisualisierung

Zeichenalgorithmen

- Durch Objekte und Transformationen definierte Szene muss auf den Bildschirm gezeichnet werden.
- Zeichenalgorithmen übernehmen diesen Vorgang des sogenannten Rasterns
  - D.h. Umsetzung eines kontinuierlichen Objektes in diskrete Pixel

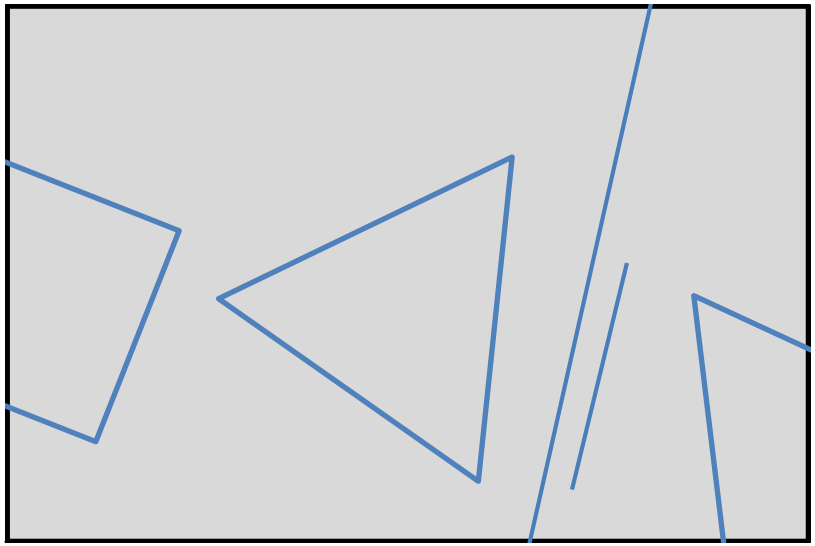


http://iloveshaders.blogspot.de/2011/05/how-rasterization-process-works.html

5.1. CLIPPING

Clipping

- Nach Durchlaufen der Transformationskette ist nur noch ein Ausschnitt der 3D Szene zu sehen.
- Alle Objekte die außerhalb dieses sichtbaren Bereichs (=Viewport) liegen müssen nicht gezeichnet werden.
- Clipping = Zuschneiden von Objekten an einem vorgegebenen Bereich.

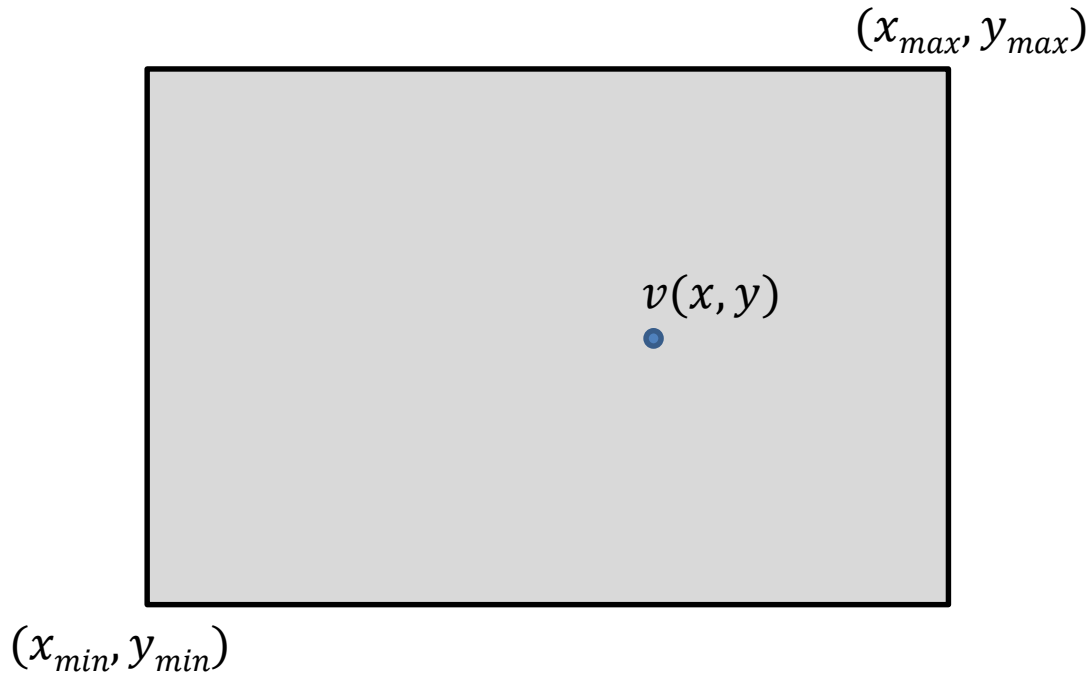


Clipping von Vertices

- Einfachste Form des Clippings
  - Test ob der Vertex innerhalb des Viewports liegt:

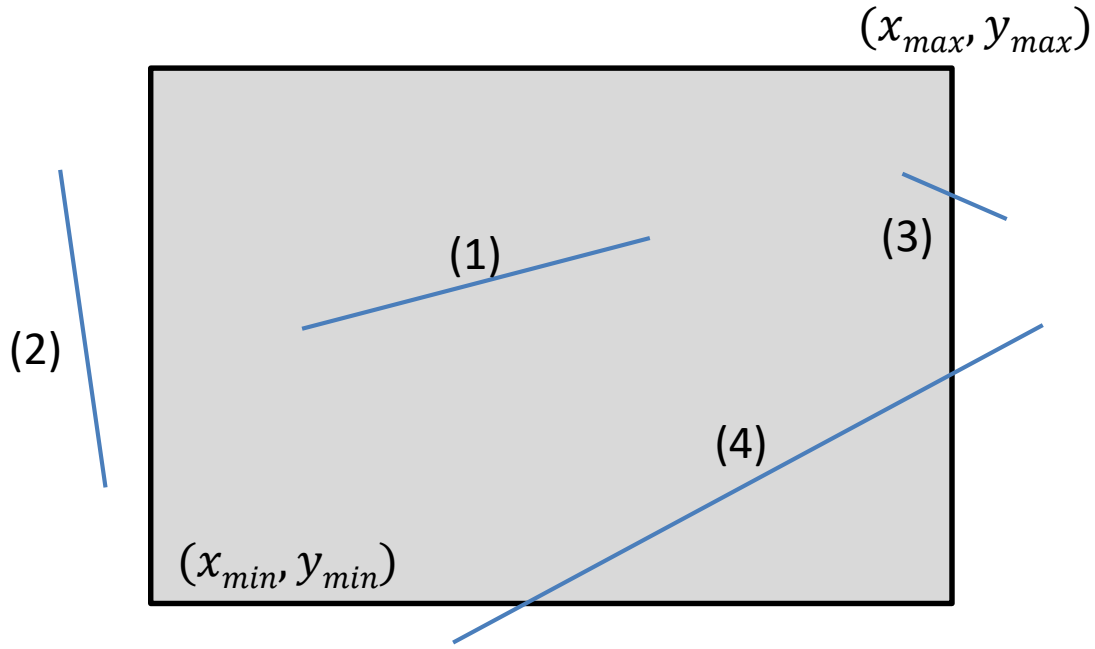
Es müssen folgende Bedingungen erfüllt sein:

$$x_{min} \leq x \leq x_{max}$$
$$y_{min} \leq y \leq y_{max}$$



Clipping von Linien

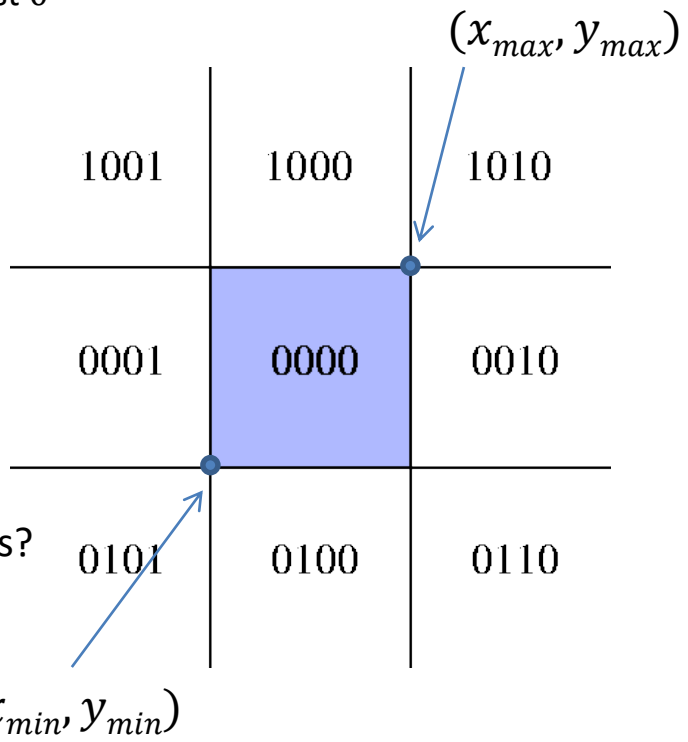
- 4 mögliche Lagebeziehungen von Linien zum Darstellungsbereich:
  (1) Linie befindet sich vollständig im Darstellungsbereich
  (2) Linie befindet sich vollständig außerhalb des Darstellungsbereichs
  (3) Ein Endpunkt innerhalb, ein Endpunkt außerhalb
  (4) Beide Endpunkte außerhalb, aber Linie schneidet den Darstellungsbereich



Clipping von Linien

- Algorithmus von Cohen und Sutherland:
  - Prinzip der Bereichsprüfung: Einteilung des Darstellungsbereichs in 9 Teile
  - Zuordnung eines Bereichscodes zu jedem Anfangs- und Endvertex
  - Bereichscodes aus binärer Darstellung der Lagebeziehung der Bereiche
    - Erstes Bit (hinten): 1 = links von Darstellungsbereich, sonst 0
    - usw.

$y_{max} - y < 0$: viertes Bit = 1
$y - y_{min} < 0$: drittes Bit = 1
$x_{max} - x < 0$: zweites Bit = 1
$x - x_{min} < 0$: erstes Bit = 1

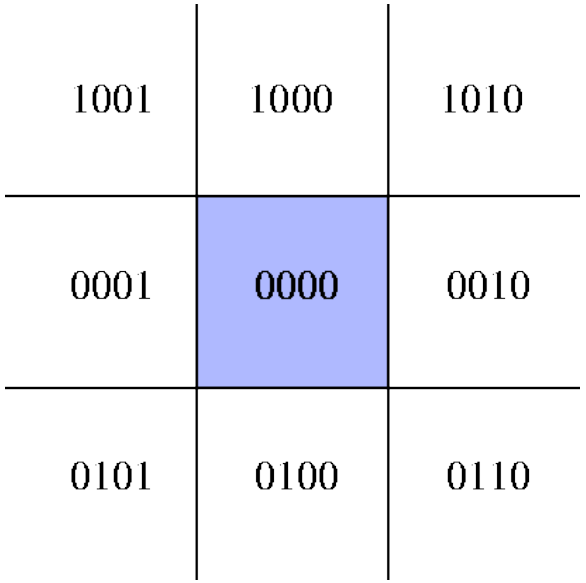


  - Beide Punkte innerhalb des Fensters?
    - Ja, wenn bitweise ODER-Verknüpfung von v_1 und $v_2 = 0$
  - Beide Punkte und gesamte Linie außerhalb des Fensters?
    - Ja, wenn bitweise UND-Verknüpfung von v_1 und v_2 an einer Stelle ungleich 0

http://www-lehre.inf.uos.de/~rkunze/flashweather/Diplomarbeit/node37.html

Clipping von Linien

- Ist keiner der beiden Tests erfolgreich, ist nicht auszuschließen dass die Linie das Fenster schneidet.
- In diesem Fall wird ein Schnitttest durchgeführt:
  - Berechnung des Schnittpunktes der Linie mit einer Seite des Fensters
  - Seite(n) kann (können) anhand von Bereichscode gewählt werden.
  - Für beide Teilsegmente der Linie wird erneut getestet ob Anfangs- oder Endpunkt außerhalb des Fensters liegen
  - Gegebenenfalls erneute Schnittpunktberechnung für Teilsegmente



Clipping von Linien

- Bestimmung des Schnittpunktes einer Linie mit dem Darstellungsbereich:
  - Parameterform (Punkt-Richtungsform) einer Geraden: $x = v_0 + r(v_1 - v_0)$
  - Eine Koordinate des Schnittpunktes ist durch den Algorithmus von Cohen und Sutherland i.d.R. bekannt, z.B. $s_y = y_{max}$
  - Gesucht wird in diesem Beispiel nun s_x.
  - Daraus ergibt sich folgendes Gleichungssystem:

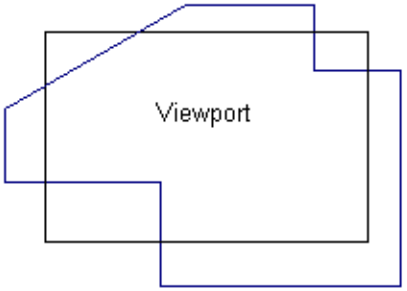
$$(1)\ s_x = v_{0,x} + r\left(v_{1,x} - v_{0,x}\right)$$
$$(2)\ s_y = v_{0,y} + r\left(v_{1,y} - v_{0,y}\right)$$

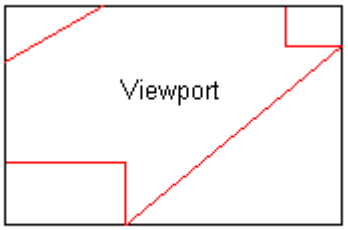
  - Aus (2) lässt sich nun r berechnen, da z.B. s_y bekannt ist.
  - Das Ergebnis wird in (1) eingesetzt und nach s_x aufgelöst:

$$s_x = v_{0,x} + \frac{s_y - v_{0,y}}{v_{1,y} - v_{0,y}}\left(v_{1,x} - v_{0,x}\right)$$

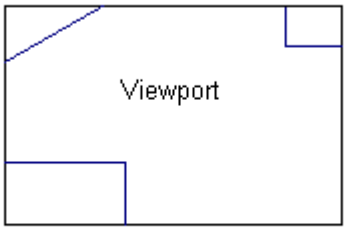
Clipping von Polygonen

- Linien-Clipping-Verfahren kann bei Anwendung auf Polygone zu falschen Ergebnissen führen: Topologie eines Polygons wird u.U. zerstört.





falsch



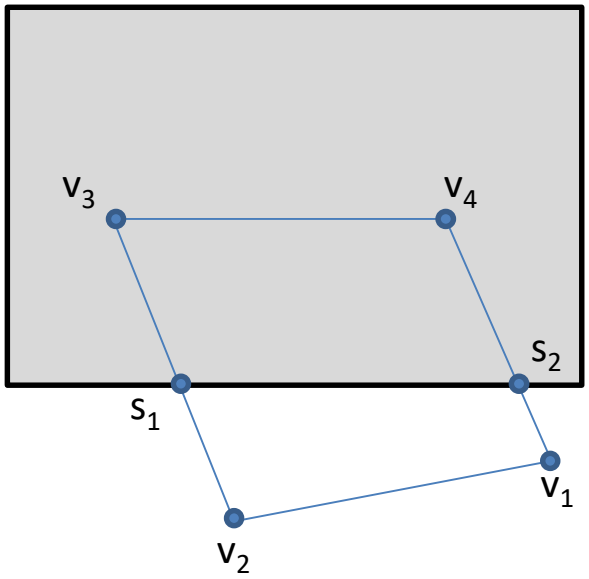
richtig

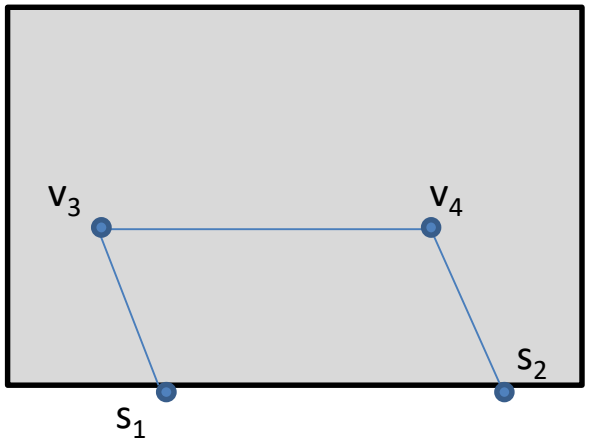
- Algorithmus von Sutherland und Hodgman:
  - Interpretation der Kanten des Darstellungsbereichs als Geraden ohne Begrenzung
  - Schrittweise Schneiden aller Kanten des Polygons an jeweils einer Geraden des Darstellungsbereichs

  - Entscheidung welche Vertices in Ausgabepolygon übernommen werden:
    (1) Beide Vertices der Kante außerhalb: keinen Vertex in Ausgabepolygon übernehmen
    (2) Gerichtete Kante von v_1 zu v_2 von außen nach innen: Schnittpunkt und v_2 übernehmen
    (3) Beide Vertices der Kante innerhalb: beide Vertices in das Ausgabepolygon übernehmen
    (4) Gerichtete Kante von v_1 zu v_2 von innen nach außen: Schnittpunkt und v_1 übernehmen

http://www.schulportal.bremerhaven.de/wypior/GraphDV/GrafDVTeil2.html

Clipping von Polygonen





Kante $v_1 \rightarrow v_2$: Beide außerhalb. Nicht in Ausgabepolygon übernehmen

Kante $v_2 \rightarrow v_3$: Gerichtete Kante von außen nach innen: s_1 und v_3 einfügen

Kante $v_3 \rightarrow v_4$: Beide innerhalb. v_4 einfügen (v_3 wurde schon)

Kante $v_4 \rightarrow v_1$: Gerichtete Kante von innen nach außen: s_2 einfügen (v_4 wurde schon)

5.2. ZEICHNEN VON LINIEN

Brute Force

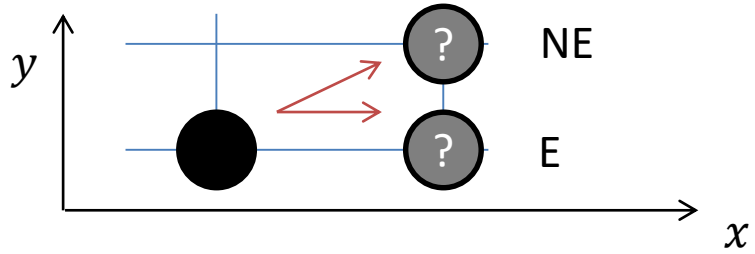
- Naiver Algorithmus basierend auf Steigungsform:
  - Berechnung der Steigung: $m = \frac{\Delta y}{\Delta x} = \frac{v_{2,y} - v_{1,y}}{v_{2,x} - v_{1,x}}$
  - Starte von links: $y_i = mx_i + B$
  - Zeichne den Pixel $(x_i, round(y_i))$
  - Inkrementelles Ermitteln des nächsten Pixels:

$$\begin{aligned} y_{i+1} &= mx_{i+1} + B \\ &= m(x_i + \Delta x) + B \\ &= m\Delta x + mx_i + B \\ &= m\Delta x + y_i \end{aligned}$$

- Nachteile: Speicherung als Gleitkommazahl + aufwendige Rundung

Midpoint Line Algorithmus

- = Bresenham Algorithmus, nach Jack Bresenham, 1965
- Grundidee: Nutzung der Scanline. Aufbau des Bildes von links nach rechts
- Vorteil: Beschränkung auf ausschließlich Ganzzahl-Arithmetik möglich
- Reduktion des Problems auf Linien, deren Steigungswinkel zwischen $0°$ und $45°$ liegt.
  - Alle anderen Linien lassen sich aus Symmetrieüberlegungen auf diesen Fall zurückführen.
- Ablauf:
  - X-Koordinate wird schrittweise um 1 erhöht
  - Für zugehörige y-Koordinate wird festgestellt, ob sie gleich bleibt (E) oder um 1 erhöht wird (NE)



Midpoint Line Algorithmus

Herleitung der y-Wert-Entscheidung: E oder NE?

- Konventionen
  - Zeichnen einer Linie von (x_0, y_0) zu (x_1, y_1)
  - (x_p, y_p) sei ein bereits ausgewählter Pixel auf dieser Linie
- Steigungsform einer Linie:

$$y = \frac{\Delta y}{\Delta x} x + B \quad \text{mit} \quad \Delta y = y_1 - y_0, \Delta x = x_1 - x_0$$

- Umgeschrieben in implizite Formulierung:

$$F(x, y) = \Delta y \cdot x - \Delta x \cdot y + B \cdot \Delta x$$

- $F(x, y)$ ist
  - $= 0$ für Punkte auf der Linie
  - > 0 für Punkte unterhalb der Linie
  - < 0 für Punkte oberhalb der Linie

Midpoint Line Algorithmus

- Berechnet wird nun der Funktionswert von Punkt M (Midpoint) als

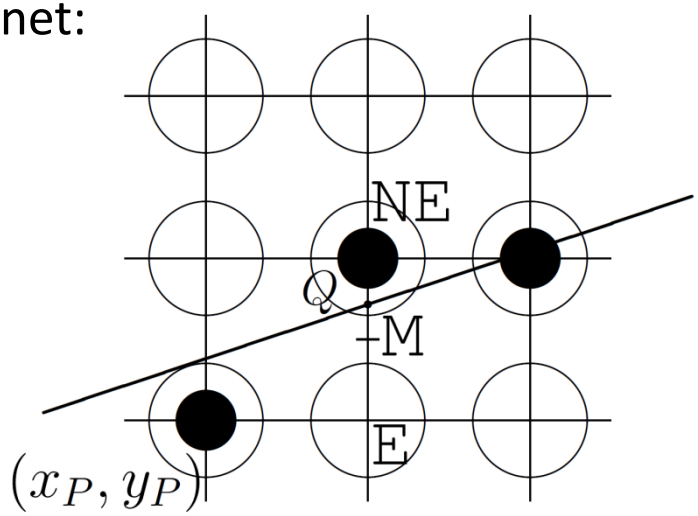
$$F(M) = F\left(x_p + 1, y_p + \tfrac{1}{2}\right) = d$$

- Das Vorzeichen der Entscheidungsvariablen d entscheidet nun darüber ob der y-Wert inkrementiert wird oder gleich bleibt:
  - $d > 0$: wähle NE *(entspr.: M liegt unterhalb der Geraden)*
  - $d \leq 0$: wähle E *(entspr.: M liegt oberhalb der Geraden)*
- Die neue Entscheidungsvariable d_{new} wird in Abhängigkeit von der Wahl NE oder E inkrementell aus der alten berechnet:
  - Wenn E gewählt wurde:

$$d_{new} = F\left(x_p + 2, y_p + \tfrac{1}{2}\right)$$
$$\Rightarrow\ d_{new} = d_{old} + \Delta y$$

  - Wenn NE gewählt wurde:

$$d_{new} = F\left(x_p + 2, y_p + \tfrac{3}{2}\right)$$
$$\Rightarrow\ d_{new} = d_{old} + (\Delta y - \Delta x)$$



S. Krömker: Computergrafik I, Universität Heidelberg

Midpoint Line Algorithmus

- Bei der Initialisierung wird der anfängliche Wert der Entscheidungsvariablen festgelegt:
  - Funktionswert bei (x_0, y_0)

$$d_{ini} = F\left(x_0 + 1, y_0 + \tfrac{1}{2}\right) = \underbrace{F(x_0, y_0)}_{= 0} + \Delta y - \tfrac{\Delta x}{2} = \Delta y - \tfrac{\Delta x}{2}$$

- Somit ergibt sich folgender Pseudocode:

```
// init
dx = x1-x0;
dy = y1-y0;
incrE = dy*2;
incrNE = (dy-dx)*2;
d = dy*2 - dx;
x = x0;
y = y0;
```

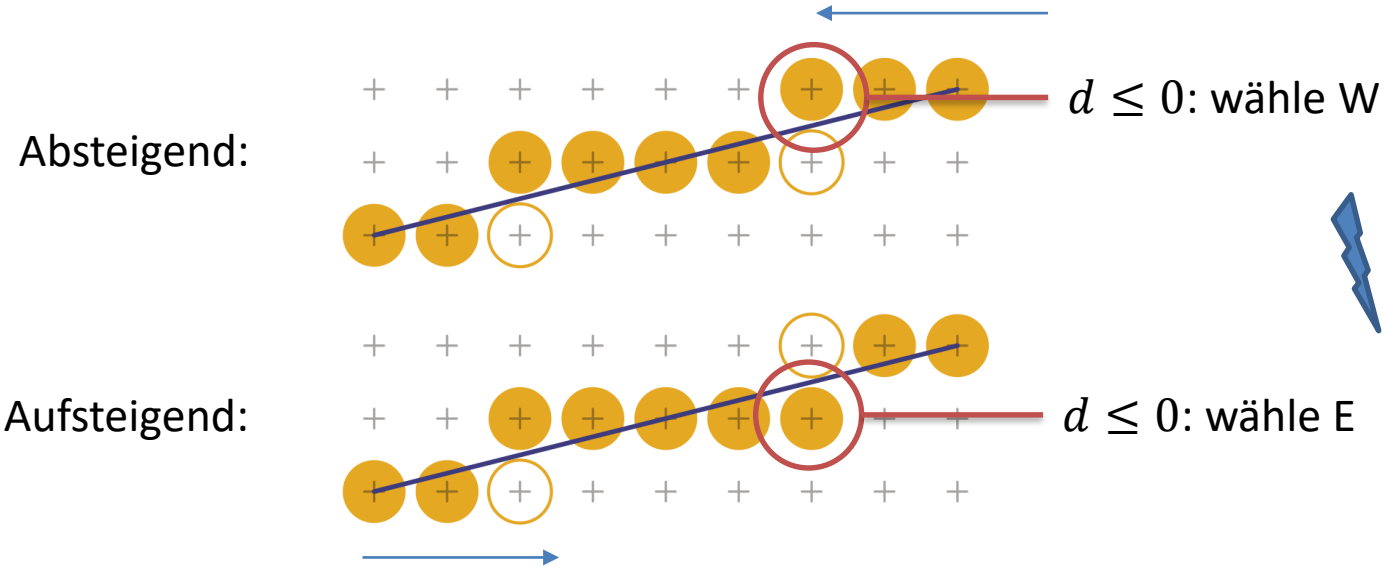
Inkremente entsprechend der Wahl E oder NE

→ Hier multipliziert mit 2 um den Bruch $(y + ½)$ zu sparen

```
writePixel(x,y);
while(x < x1) {
   if(d <= 0) {
      d += IncrE;     // choose E
      x++;
   } else {
      d += IncrNE;    // choose NE
      x++;
      y++;
   }
   writePixel(x,y)
}
```

Midpoint Line Algorithmus

- Unabhängigkeit der Laufrichtung: Was passiert wenn $d = 0$?

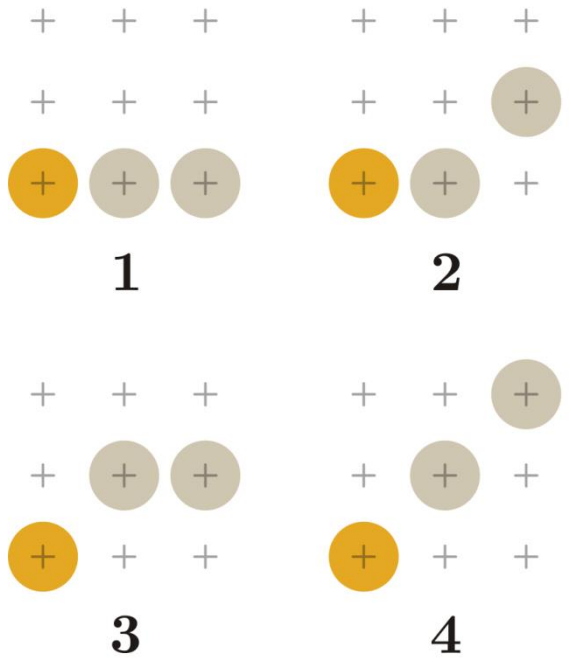


- Anwenden der Konvention für $d = 0$:
  - Wähle **E** wenn die Linie aufsteigend von links nach rechts gezeichnet wird
  - Wähle **SW** wenn die Linie absteigend von rechts nach links gezeichnet wird
- Zur Beschleunigung kann der Algorithmus auch von beiden Endpunkten gleichzeitig gestartet werden

https://de.wikipedia.org/wiki/Rasterung_von_Linien

N-Schritt-Verfahren

- Grundidee: Schritte von mehreren Pixeln in x-Richtung vollziehen, alle dazwischen liegende Pixel werden auf einmal eingefärbt
- Wahl von verschiedenen Pixelmustern, z.B. **Doppelschrittverfahren**:
  - Wenn Steigung $m \leq ½$: Muster 4 kann nicht auftreten
  - Wenn Steigung $m \geq ½$: Muster 1 kann nicht auftreten



- Entscheidung für Pixelmuster:
  Betrachtung der letzten Pixelspalte
  - Pixel oben → (4)
  - Pixel unten → (1)
  - Pixel in der Mitte → weiterer Test
- Analoge Vorgehensweise wie bei Midpoint Line Algorithmus: Betrachtung der Midpoints M

https://de.wikipedia.org/wiki/Rasterung_von_Linien

N-Schritt-Verfahren

- Pseudocode für den 1. Fall ($m < ½$) :

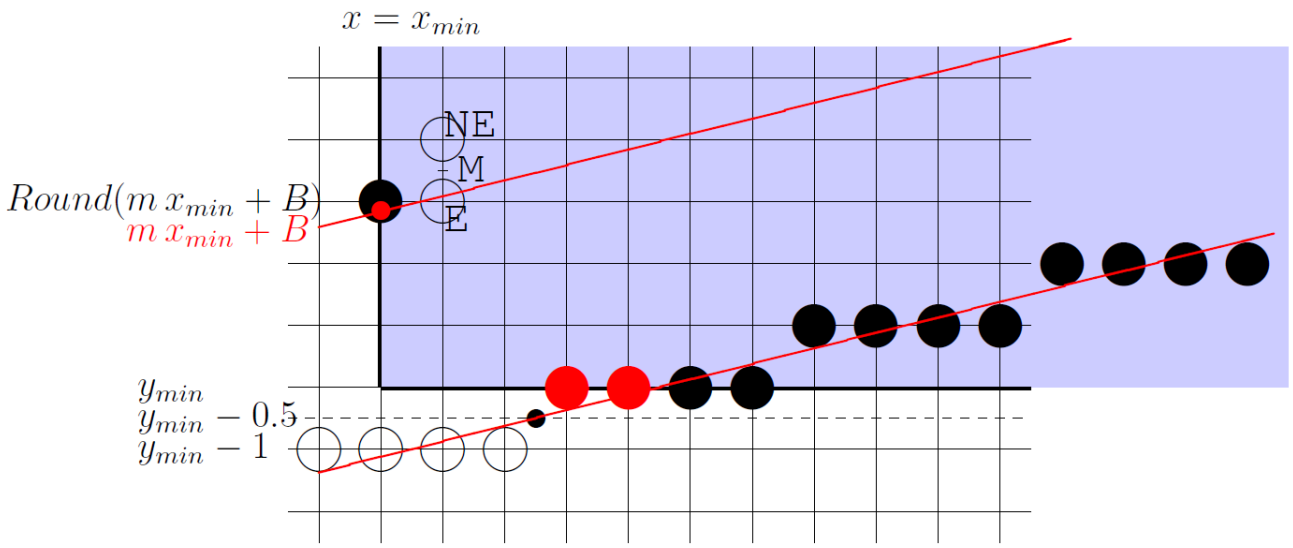
```
// init
dx = x1-x0;
dy = y1-y0;
incr1= 4*dy;
incr2= 4*dy-2*dx;
cond = 2*dy
d = 4*dy - dx;
x = x0;
y = y0;
```

```
while(x < x1) {
   if(d <= 0) {        // Pattern 1
      drawPixels(pattern1,x);
      d += incr1;
   } else {
      if(d < cond) {  // Pattern 2
         drawPixels(pattern2,x);
      } else          // Pattern 3
         drawPixels(pattern3,x);
      }
      d += incr2;
   }
   x += 2;
}
```

- Algorithmen für Dreifach- und Vierfachschritt nach gleichem Prinzip vorhanden
- Midpoint-Line-Algorithmus ist Spezialfall des N-Schritt-Verfahrens mit einer Schrittweite von 1 und zwei Mustern aus denen gewählt wird.

Schnittpunkt mit Viewport

- Beim Erzeugen von Linien muss auf Schnittpunkte mit dem Viewport geachtet werden

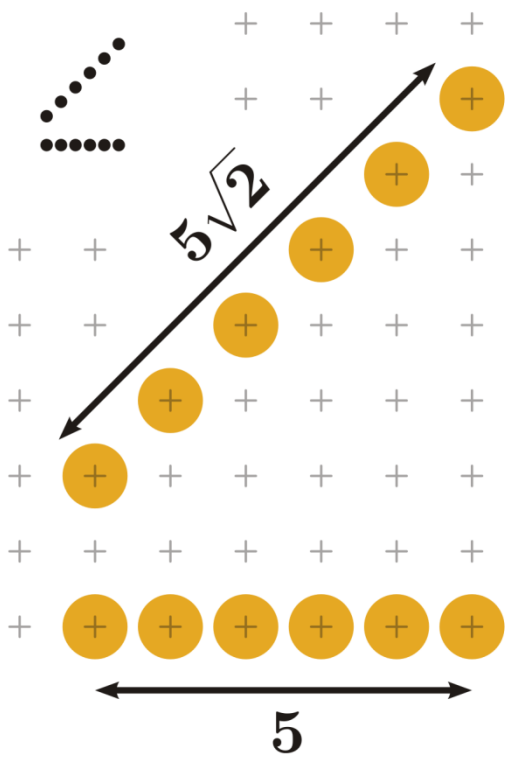


- Initialisierung kann angepasst werden (keine Rundung!):
- Schnitt mit unterer/oberer Kante: Schnitt mit $y_{min} - 0.5$ bzw. mit $y_{max} + 0.5$ verwenden.

S. Krömker: Computergrafik I, Universität Heidelberg

Intensitätsschwankungen

- Konstante Anzahl von Pixeln für unterschiedlich lange Linien (bis Faktor $\sqrt{2}$).
- Intensität der Linie schwankt
- Bei Bilevel-Displays keine Abhilfe möglich
- Sonst: Variieren der Intensität einer Linie als Funktion der Steigung

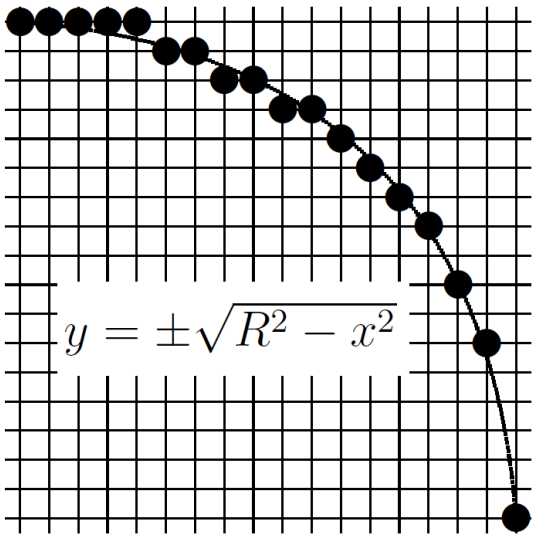


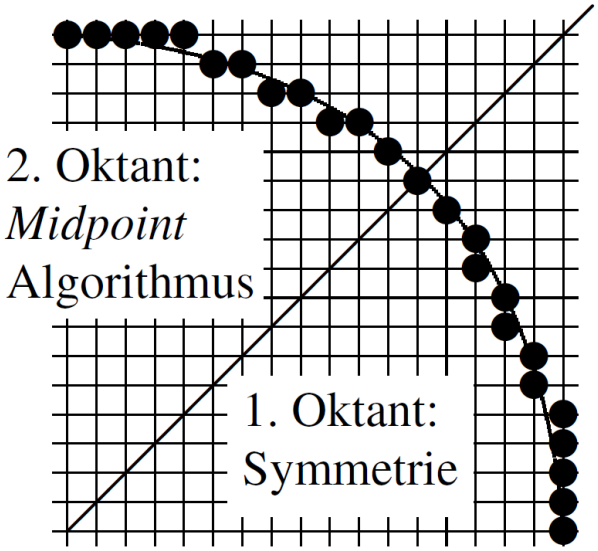
https://de.wikipedia.org/wiki/Rasterung_von_Linien

5.2. ZEICHNEN VON KREISEN

Midpoint Circle Algorithmus

- Zeichnen von Kreisen analog zum Midpoint Line Algoritmus.
- Kreis wird im zweiten Oktanten gezeichnet und die Symmetrieeigenschaft für eine Spiegelung genutzt.

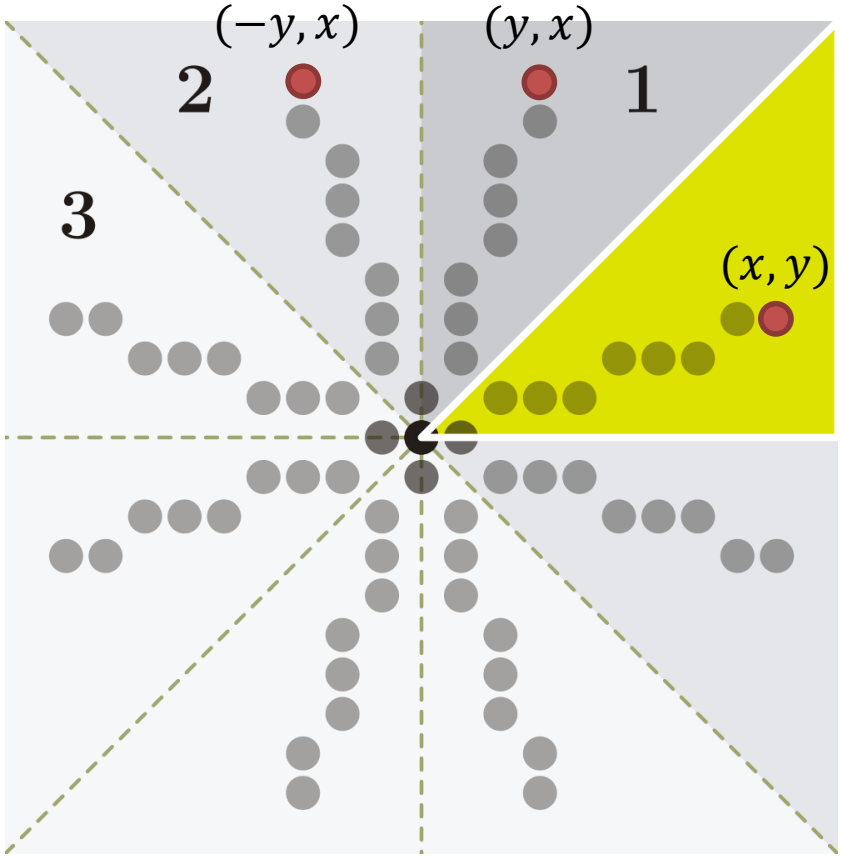




S. Krömker: Computergrafik I, Universität Heidelberg

Midpoint Circle Algorithmus

- Erinnerung: Ausnutzung der Symmetrie




```
void circlePoint (int x, int y) {
   writePixel( x, y);
   writePixel( y, x);
   writePixel( y,-x);
   writePixel( x,-y);
   writePixel(-x,-y);
   writePixel(-y,-x);
   writePixel(-y, x);
   writePixel(-x, y);
}
```

https://de.wikipedia.org/wiki/Rasterung_von_Linien

Midpoint Circle Algorithmus

- Gleichung eines Kreises: Implizite Formulierung

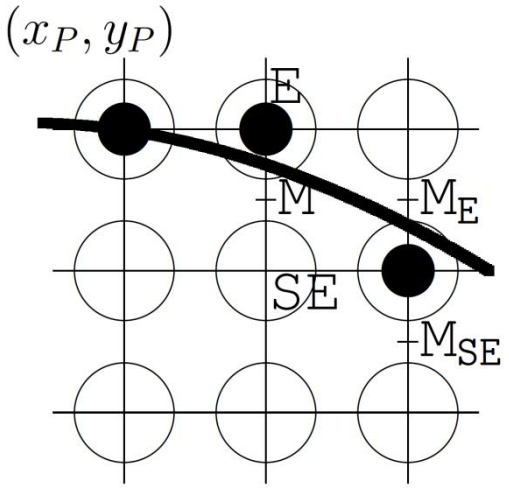
$$F(x,y) = x^2 + y^2 - R^2 = 0$$

- Konvention:
  - Start bei 12 Uhr
  - Zeichnen bis 1.30 Uhr
- Entsprechend wird die Entscheidungsvariable berechnet

$$F(M) = F\left(x_p + 1, y_p - \tfrac{1}{2}\right) = d$$

$$d = (x_p + 1)^2 + (y_p - \tfrac{1}{2})^2 - R^2$$

- Vorzeichen von d entscheidet über nächsten Punkt auf der Linie:
  - $d \geq 0$: wähle SE
  - $d < 0$: wähle E



S. Krömker: Computergrafik I, Universität Heidelberg

Midpoint Circle Algorithmus

- Die neue Entscheidungsvariable d_{new} wird in Abhängigkeit von der Wahl SE oder E aus der alten Entscheidungsvariablen d_{old} berechnet:
  - Wenn E gewählt wurde:

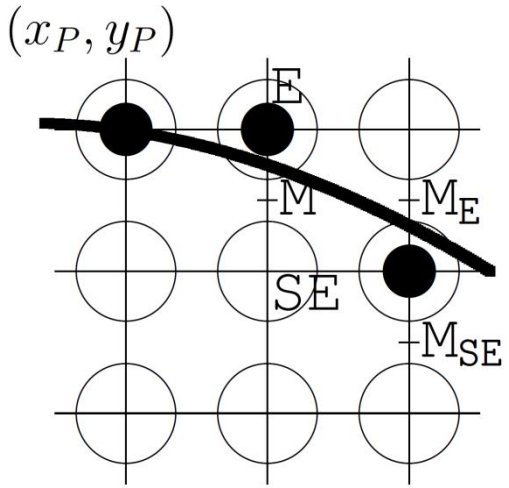
$$d_{new} = F\left(x_p + 2, y_p - \tfrac{1}{2}\right) = (x_p + 2)^2 + (y_p - \tfrac{1}{2})^2 - R^2$$

$$\Rightarrow\ d_{new} = d_{old} + 2x_p + 3$$

  - Wenn SE gewählt wurde:

$$d_{new} = F\left(x_p + 2, y_p - \tfrac{3}{2}\right) = (x_p + 2)^2 + (y_p - \tfrac{3}{2})^2 - R^2$$

$$\Rightarrow\ d_{new} = d_{old} + 2x_p - 2y_p + 5$$



- Initialisierung von *d*:

$$d = F\left(x_0 + 1, y_0 - \tfrac{1}{2}\right) = F\left(1, R - \tfrac{1}{2}\right) = 1 + \left(R^2 - R + \tfrac{1}{4}\right) - R^2$$
$$= \tfrac{5}{4} - R$$

S. Krömker: Computergrafik I, Universität Heidelberg

Midpoint Circle Algorithmus

- Pseudocode:

```
// init
d = 5/4-radius;       // float!
x = 0;
y = radius;
circlePoint(x,y);     // first pixel

// iterate
while (y > x) {
   if(d < 0) {        // choose E
      d += x*2.0 + 3
      x++;
   } else {           // choose SE
      d += (x-y)*2.0 + 5;
      x++;
      y--;
   }
   circlePoint(x,y);
}
```

Midpoint Circle Algorithmus

- Vergleich Midpoint Line / Midpoint Circle Algorithmus

	Midpoint Line	Midpoint Circle
Offset	E: Δy NE: $(\Delta y - \Delta x)$	E: $2x_p + 3$ SE: $2x_p - 2y_p + 5$
Inkremente	Konstant	Lineare Funktion
Operationen	Integer	Floating Point

- Weitere Bresenham-Algorithmen:
  - Ellipsen
  - Bézierkurven
  - …

http://members.chello.at/~easyfilter/bresenham.html

5.3. ANTI-ALIASING BEI LINIEN

Anti-Aliasing bei Linien

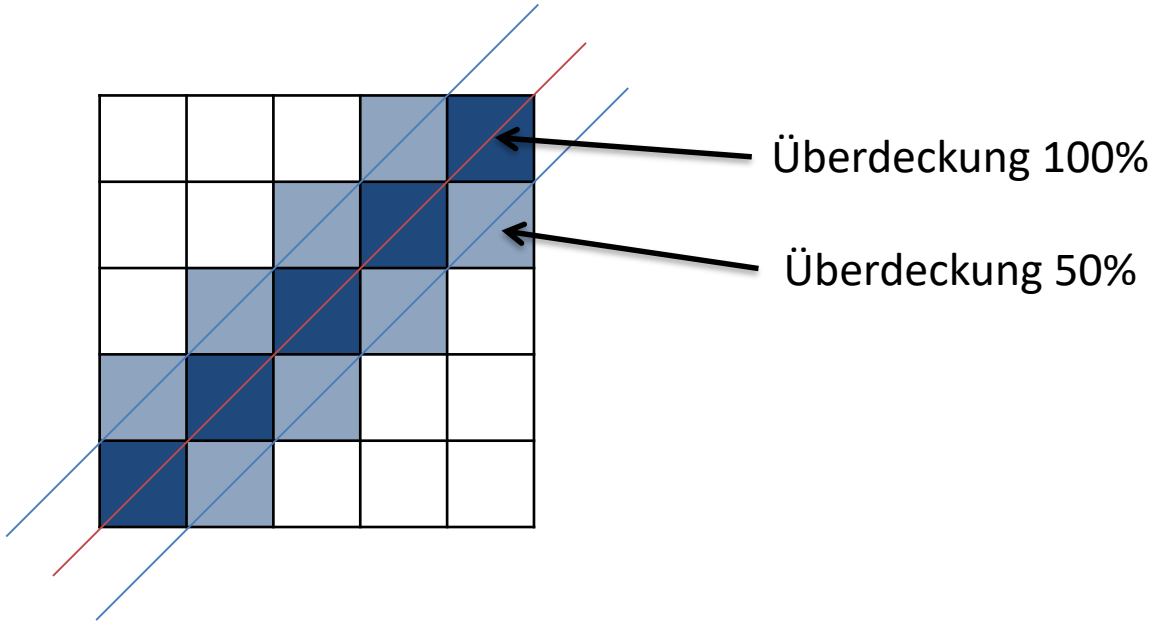
- Binäre Entscheidung führt zu Treppen-Effekten von Linien/Kreisen
  - Mögliche Abhilfe: Erhöhung der Bildschirmauflösung
- Anti-Aliasing: Variation der Intensität der Pixel entsprechend des Anteils am jeweiligen Objekt.
  - Meist beschränkt auf Linien/Kanten

- Alternative: Szenen-Antialiasing durch z.B. geschickte Nutzung des Accumulationbuffers *-- folgt im nächsten Kapitel*

https://commons.wikimedia.org/wiki/Category:Anti-aliasing

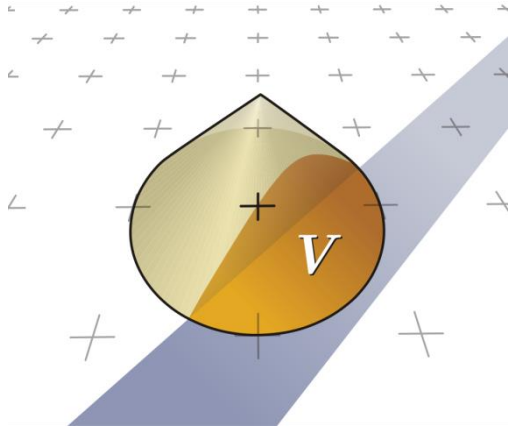
Anti-Aliasing: ungewichtete Abtastung

- Ideale Linie hat keine Breite
- Beim Rastern geht man von einer Breite von mind. 1 Pixel aus
- Berechnung der Flächenüberdeckung für alle Pixel.
  - Intensität des Pixels = lineare Funktion der am Objekt beteiligten Fläche
  - Abstand der angeschnittenen Fläche zum Pixelmittelpunkt geht nicht mit ein.



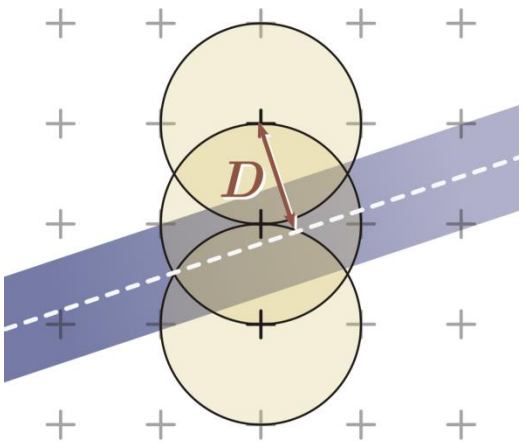
Anti-Aliasing: Gupta-Sproull Methode

- 3D-Kegel als Glättungskern
- Intensität proportional zum überdeckten Volumen



- Volumen V abhängig von Abstand des Pixelmittelpunktes zur Mittellinie (D)

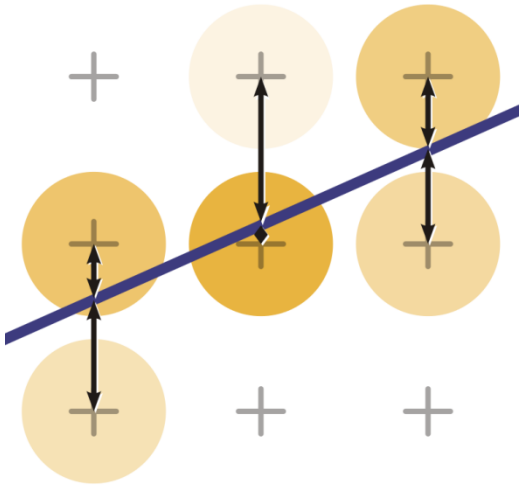
- Erweiterung des Bresenham-Algorithmus
  - Berechnet zusätzlich Distanz D, für jedes Pixel, dessen Glättungskern mit der Linie überlappt.
  - Effiziente Implementierung: Reduktion auf 24 mögliche Werte für D. Intensität wird aus Lookuptabelle entnommen.



https://de.wikipedia.org/wiki/Rasterung_von_Linien

Anti-Aliasing: Wu-Methode

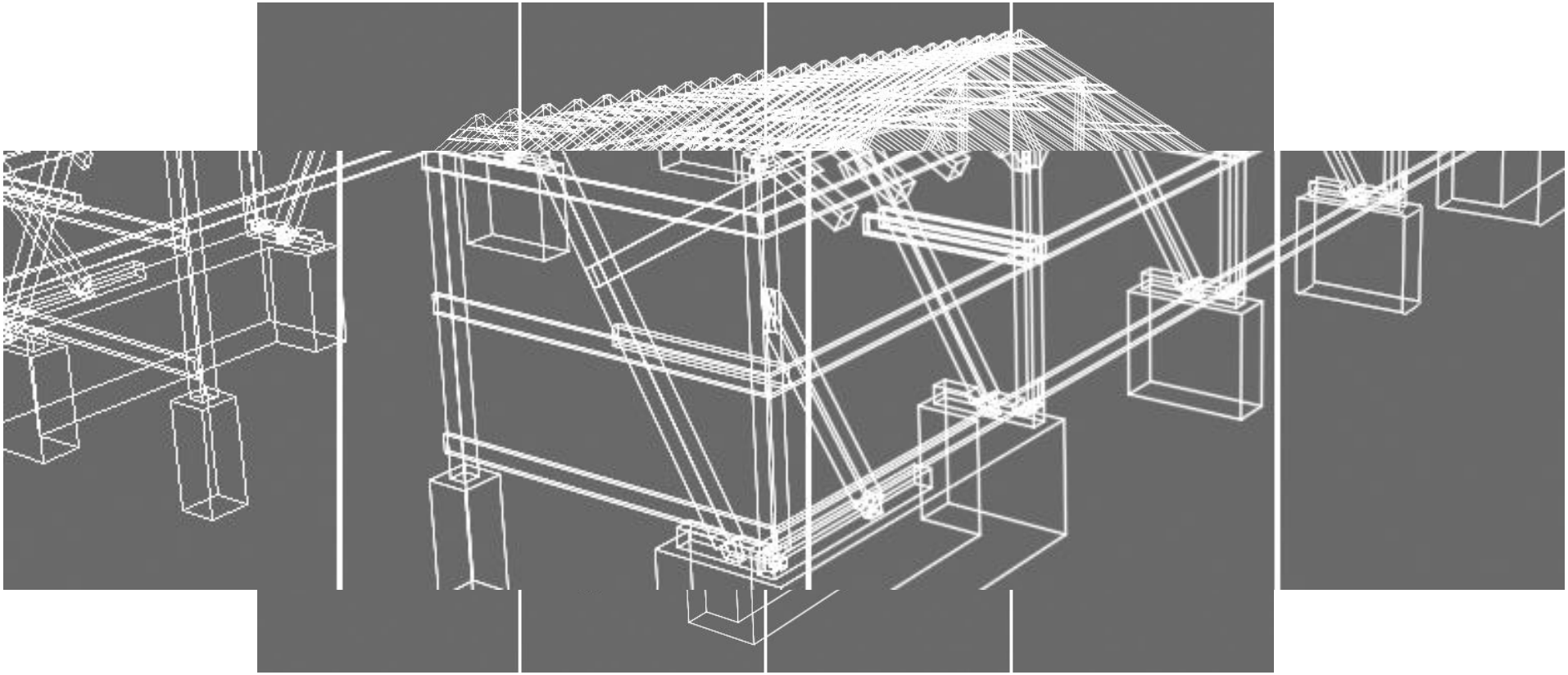
- Basiert auf Fehlermaß des Bresenham-Algorithmus
  - Statt binärer Linie werden beliebige Farbwerte zugelassen
  - Pixel erhalten Farbwerte proportional zur vertikalen Distanz zur idealen Linie



- Sehr performante Implementierung als Erweiterung des Bresenham-Algorithmus mit Kontrollvariable

Anti-Aliasing bei Linien

- Vergleich der Methoden



Bresenham | Gupta-Sproull | ungewichtet | Wu

https://de.wikipedia.org/wiki/Rasterung_von_Linien

Anti-Aliasing bei Linien

- Punkt/Linien/Polygon-Antialiasing in OpenGL:

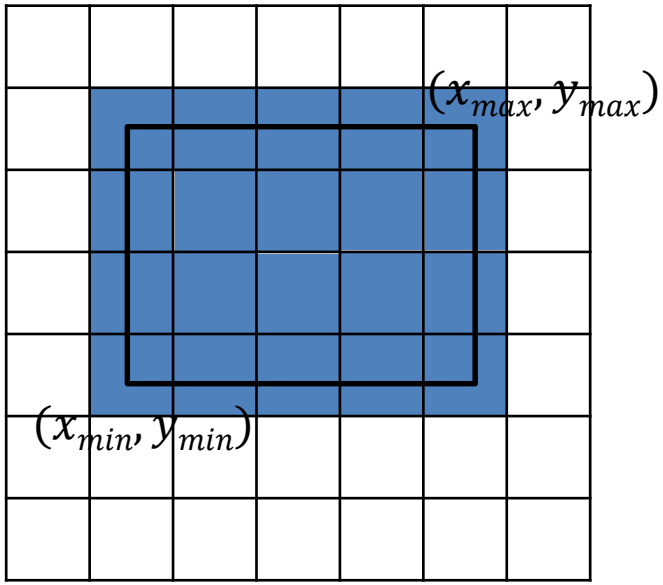
```
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glHint (GL_LINE_SMOOTH_HINT, GL_NICEST);
glHint (GL_POINT_SMOOTH_HINT, GL_NICEST);
glHint (GL_POLYGON_SMOOTH_HINT, GL_NICEST);


glEnable (GL_BLEND);
glEnable (GL_POINT_SMOOTH);
glEnable (GL_LINE_SMOOTH);
glEnable (GL_POLYGON_SMOOTH);
```

- Nutzt Alpha Blending – *sehen wir später in der Vorlesung*
- *Hints*: Hinweise an den Treiber schnellere Berechnung oder qualitativ hochwertigere Berechnung durchzuführen (Umsetzung obliegt dem Treiber)

5.4. FÜLLEN VON POLYGONEN

Füllen von Rechtecken

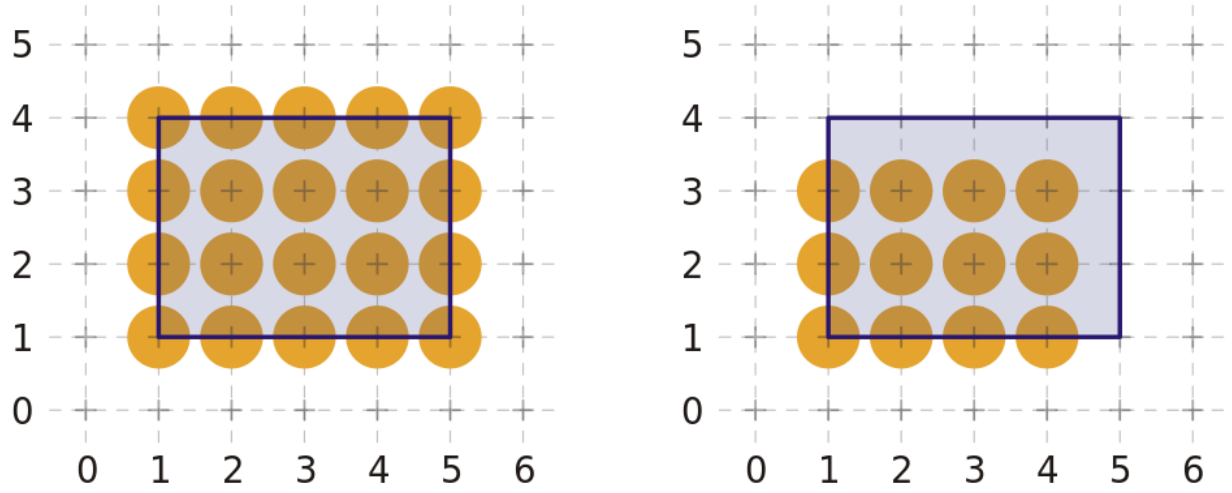
- Füllen eines achsenparallelen Rechtecks trivial:
  - Pixel zwischen (x_{min}, y_{min}) und (x_{max}, y_{max}) setzen.
  - i.d.R durch Scanline




```
int x, y;
int color;

for (y=ymin; y <= ymax; y++) {
   for (x=xmin; x <= xmax; x++) {
      setPixel(x,y,color);
   }
}
```

Füllen von Rechtecken

- Interpretation der Koordinaten:



Rechteck größer als definiert

- Abhilfe:
  - Rasterung um einen halben Pixelabstand nach links und nach unten verschieben.
  - Rastern von $x \geq x_{min} - 0{,}5$ bis $x \leq x_{max} - 0{,}5$ bzw. $y \geq y_{min} - 0{,}5$ bis $y \leq y_{max} - 0{,}5$
  - Rechteck rechts, mit den Koordinaten $(0{,}5, 0{,}5)$ und $(4{,}5, 3{,}5)$, wird korrekt gezeichnet.

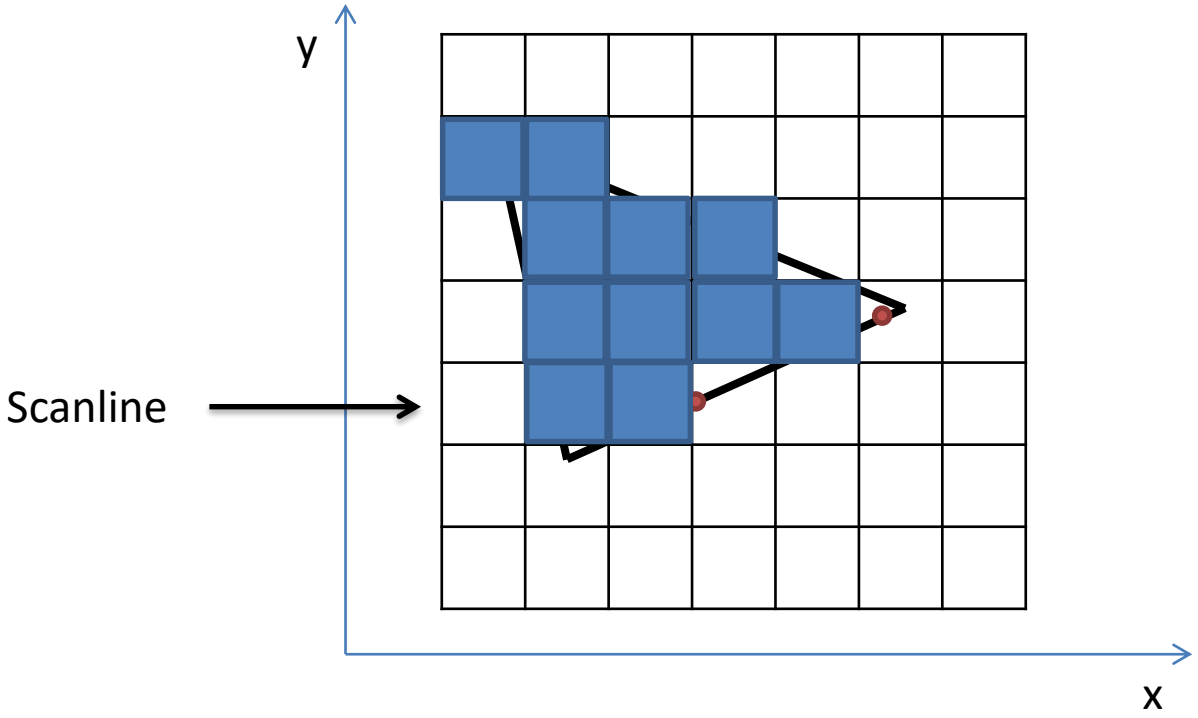
https://de.wikipedia.org/wiki/Rasterung_von_Polygonen

Füllen von (beliebigen) Polygonen

- Für jede Kante des Polygons Bestimmung des Schnittpunktes mit der Bildzeile (Scanline)
- Horizontale Kanten werden ignoriert
- Sortierung der Schnittpunkte:
  - Nach *y*-Koordinaten sortiert (Scanline)
  - Bei gleichen *y*-Koordinaten wird aufsteigend nach *x*-Koordinaten sortiert
- Liste der Schnittpunkte enthält immer eine gerade Anzahl von Werten mit gleicher *y*-Koordinate
- Zeichnen:
  - Punktpaare aus der Liste von der Form $(x_1;\ y)\ (x_2;\ y)$
  - Zeichnen aller Pixel der entsprechenden Bildzeile deren *x*-Koordinate sich im Intervall $[x_1 - 0{,}5\ x_2 - 0{,}5]$ befindet.
- Hoher Sortieraufwand!

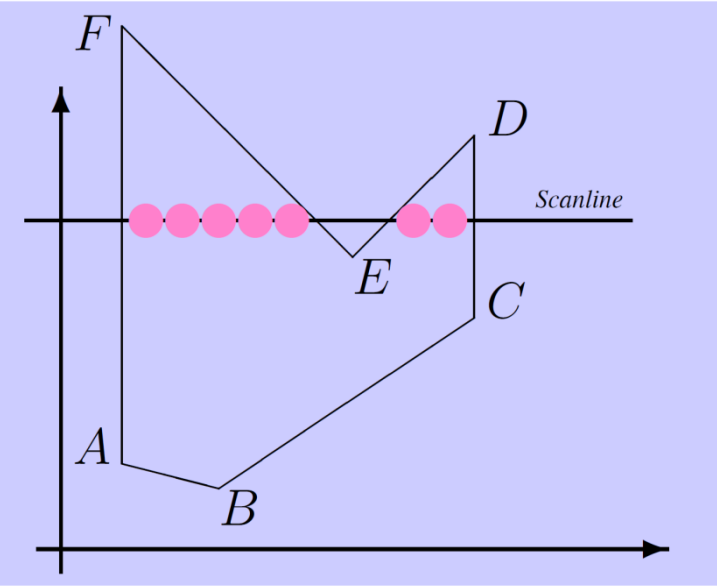
Füllen von Polygonen

- Füllen des Innern des Polygons per Scanline anhand sortierter Schnittpunkte



Füllen von Polygonen

- Modifikation:
  1. Für jeden Schnittpunkt einer Polygonkante mit einer Bildzeile: Einfärben des ersten Pixels mit $x > s_x + 0{,}5$.
  2. Füllen des Polygons durch Negation von innerhalb/außerhalb



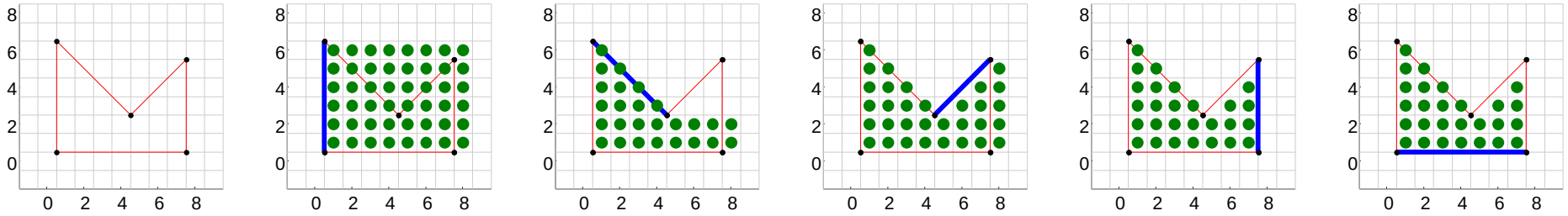
Bei bereits gezeichnetem Umriss:

```
Für jede Bildzeile y, die das Polygon schneidet
  Innerhalb = Falsch
  Für jedes x von links bis rechts
    Wenn Pixel (x, y) eingefärbt ist   // Umriss
      Innerhalb negieren
    Wenn Innerhalb
      Pixel (x, y) einfärben
    ansonsten
      Pixel (x, y) auf Hintergrundfarbe zurücksetzen
```

S. Krömker: Computergrafik I, Universität Heidelberg

Füllen von Polygonen

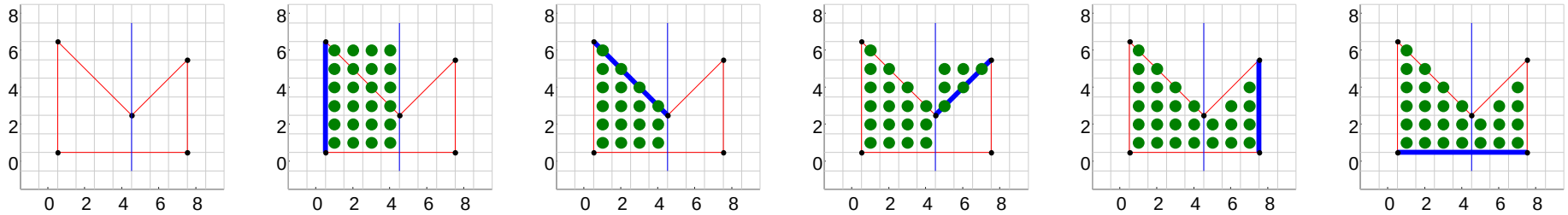
Edge-Fill-Algorithmus



- Verschiebung der Vertices um $\Delta x = -0{,}5$ und $\Delta y = -0{,}5$
- Für jede Bildzeile y, die bei Schnittpunkt s_x eine Polygonkante schneidet: alle Pixel mit $x > s_x$ invertieren (binäre Wertzuordnung)
- Reihenfolge der Polygonkanten-Abarbeitung beliebig.
- Nachteil: viele Pixel müssen mehrmals geändert werden.

Füllen von Polygonen

Fence-Fill-Algorithmus



- Erweiterung des Edge-Fill-Algorithmus:
  - Verschiebung der Vertices um $\Delta x = -0{,}5$ und $\Delta y = -0{,}5$
  - Vertikale Gerade durch Schnittpunkt zweier Polygonkanten $(= fence)$, Beispiel hier: bei $4{,}5$ (nach Verschiebung der Vertices)
  - Für alle Schnittpunkte auf der linken Seite des Zauns werden alle Pixel $x > s_x$ bis $x < fence$ invertiert.
  - Für alle Schnittpunkte auf der rechten Seite des Zauns werden alle Pixel $x > fence$ bis $x < s_x$ invertiert.
- Reduziert Invertierungsvorgänge

ZUSAMMENFASSUNG

Zusammenfassung

- Rastern: Umsetzung der kontinuierlichen Projektion der Szene in diskrete Pixel auf dem Bildschirm
- Clipping = Zuschneiden von Objekten an einem vorgegebenen Bereich.
- Zeichnen von Linien:
  - Midpoint Line Algorithmus, N-Schritt-Verfahren.
- Zeichnen von Kreisen:
  - Midpoint Circle Algorithmus
- Anti-Aliasing verringert optisch Treppeneffekte beim Zeichnen
  - Intensität von geschnittenen Pixel anhand Flächenüberdeckung
  - Ungewichtete Abtastung, Gupta-Sproull Methode, Wu-Methode
- Füllen von Rechtecken und Polygonen
  - Schnitt von Kanten mit Scanline
  - Negierung entlang der Scanline
  - Edge-Fill-Algorithmus, Fence-Fill-Algorithmus

ÜBUNGSAUFGABEN

Übungsaufgaben

1. Berechnen Sie mit dem Bresenham (Midpoint-Line)-Algorithmus eine Linie vom Punkt $v_0 = (2, 1)$ zum Punkt $v_1 = (10, 4)$. Wie sehen die d-Werte der Entscheidungsvariablen aus? Skizzieren Sie die eingefärbten Pixel in einem Koordinatensystem.

2. Erstellen Sie die gleiche Linie noch einmal mit dem Doppelschritt-Verfahren.

a) Auf welche Muster kann das Abprüfen eingeschränkt werden?

b) Welche Werte ergeben sich für die Entscheidungsvariable d?

3. Berechnen Sie mit dem Bresenham (Midpoint Circle)-Algorithmus einen Kreis mit Mittelpunkt im Ursprung und Radius $R = 10$.

a) Welche Werte für die Entscheidungsvariable ergeben sich?

b) Skizzieren Sie den Kreis in einem Koordinatensystem und geben Sie die eingefärbten Pixel an

Lösung

1. Berechnen Sie mit dem Bresenham (Midpoint-Line)-Algorithmus eine Linie vom Punkt $v_0 = (2, 1)$ zum Punkt $v_1 = (10, 4)$. Wie sehen die d-Werte der Entscheidungsvariablen aus? Skizzieren Sie die eingefärbten Pixel in einem Koordinatensystem

Vorberechnungen (Ganzzahl-Bresenham)

- $\Delta x = v_{1,x} - v_{0,x} = 10 - 2 = \mathbf{8}$
- $\Delta y = v_{1,y} - v_{0,y} = 4 - 1 = \mathbf{3}$
- Inkremente der Entscheidungsvariablen:
  - Bei Wahl E: $inc_E = 2\Delta y = 6$
  - Bei Wahl NE: $inc_{NE} = 2(\Delta y - \Delta x) = 2(3 - 8) = -10$

Initialisierungsschritt ($x = v_{0,x} = 2, y = v_{0,y} = 1$)

- $d_0 = 2\Delta y - \Delta x = 3 \cdot 2 - 8 = \mathbf{-2}$ da $d_0 \leq 0 \Rightarrow E$

Lösung (2)

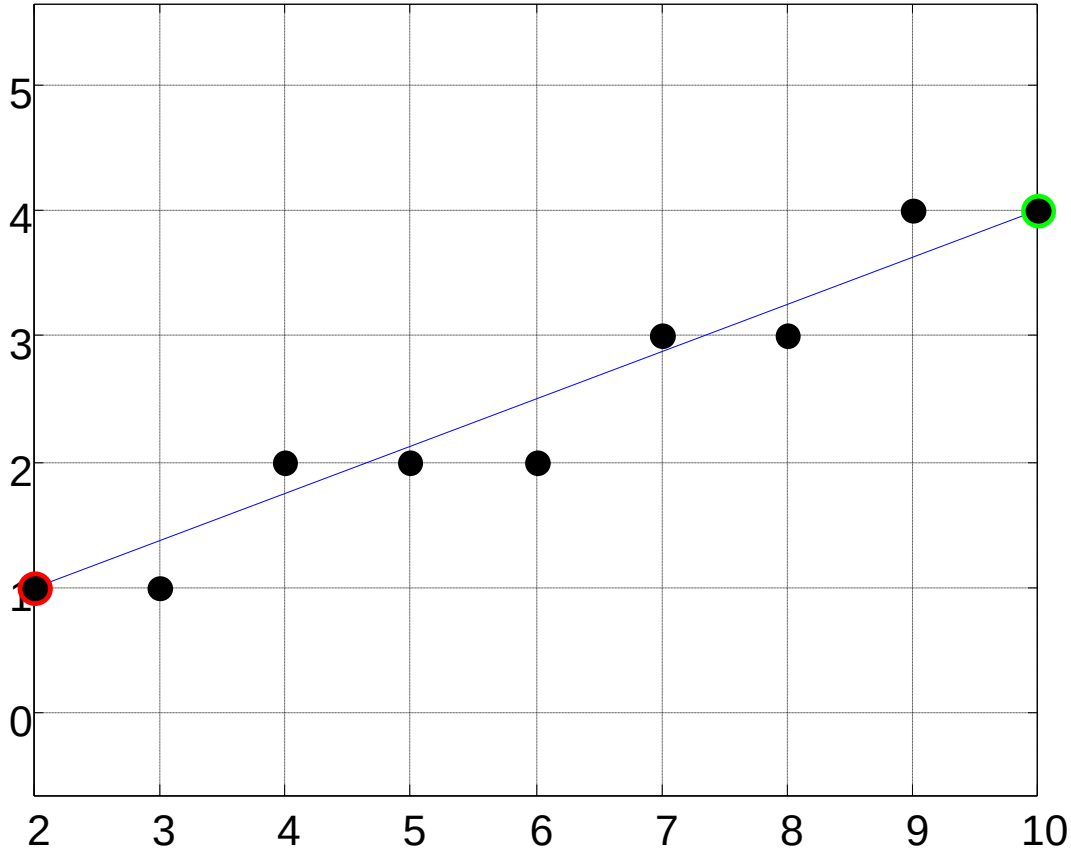
Iterationen

Neues x	**Neues y**	d	*Entscheidung für nächsten Schritt*
3	1	$d_1 = d_0 + inc_E = -2 + 6 = \mathbf{4}$	da $d_1 > 0 \Rightarrow$ NE
4	2	$d_2 = d_1 + inc_{NE} = 4 - 10 = \mathbf{-6}$	da $d_2 \leq 0 \Rightarrow$ E
5	2	$d_3 = d_2 + inc_E = -6 + 6 = \mathbf{0}$	da $d_3 \leq 0 \Rightarrow$ E
6	2	$d_4 = d_3 + inc_E = 0 + 6 = \mathbf{6}$	da $d_4 > 0 \Rightarrow$ NE
7	3	$d_5 = d_4 + inc_{NE} = 6 - 10 = \mathbf{-4}$	da $d_5 \leq 0 \Rightarrow$ E
8	3	$d_6 = d_5 + inc_E = -4 + 6 = \mathbf{2}$	da $d_6 > 0 \Rightarrow$ NE
9	4	$d_7 = d_6 + inc_{NE} = 2 - 10 = \mathbf{-8}$	da $d_7 \leq 0 \Rightarrow$ E
10	4	While-Bedingung $x < v_{1,x}$ nicht mehr erfüllt → Ende.	

Lösung (3)

Gefärbte Punkte

$$
\begin{aligned}
d_0 &= -2.0 \\
d_1 &= \ 4.0 \\
d_2 &= -6.0 \\
d_3 &= \ 0.0 \\
d_4 &= \ 6.0 \\
d_5 &= -4.0 \\
d_6 &= \ 2.0 \\
d_7 &= -8.0
\end{aligned}
$$



Lösung (4)

2. Erstellen Sie die gleiche Linie noch einmal mit dem Doppelschritt-Verfahren.

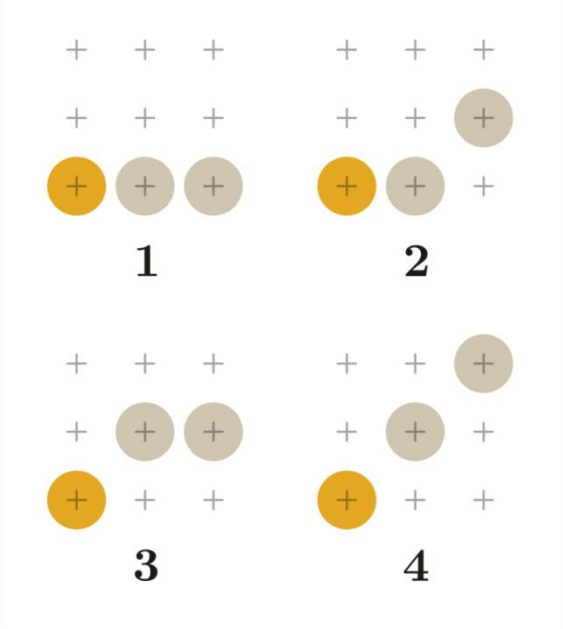
a) Auf welche Muster kann das Abprüfen eingeschränkt werden?

Da die Steigung $< ½$ ist, reicht es die Muster 1, 2, 3 abzuprüfen. ($m = \frac{\Delta y}{\Delta x} = \frac{3}{8} = 0{,}375$)

b) Welche Werte ergeben sich für die Entscheidungsvariable d?

Vorberechnungen:

- $\Delta x = v_{1,x} - v_{0,x} = 10 - 2 = \mathbf{8}$
- $\Delta y = v_{1,y} - v_{0,y} = 4 - 1 = \mathbf{3}$
- Inkremente der Entscheidungsvariablen:
  - Bei Wahl Pattern 1: $inc_{P1} = 4\Delta y = \mathbf{12}$
  - Bei Wahl Pattern 2 oder 3: $inc_{P23} = 4\Delta y - 2\Delta x = \mathbf{-4}$
- Prüf-Kondition für Pattern 2 und 3:
  - $cond = 2\Delta y = \mathbf{6}$



Lösung (5)

Initialisierungsschritt ($x = v_{0,x} = 2, y = v_{0,y} = 1$)

- $d_0 = 4\Delta y - \Delta x = 4 \cdot 3 - 8 = \mathbf{4}$ da $d_0 > 0$ ⇒ Pattern 2 oder 3, Kondition prüfen
- $d_0 < cond$ $(4 < 6)$ ⇒ Zeichnen von Pattern 2

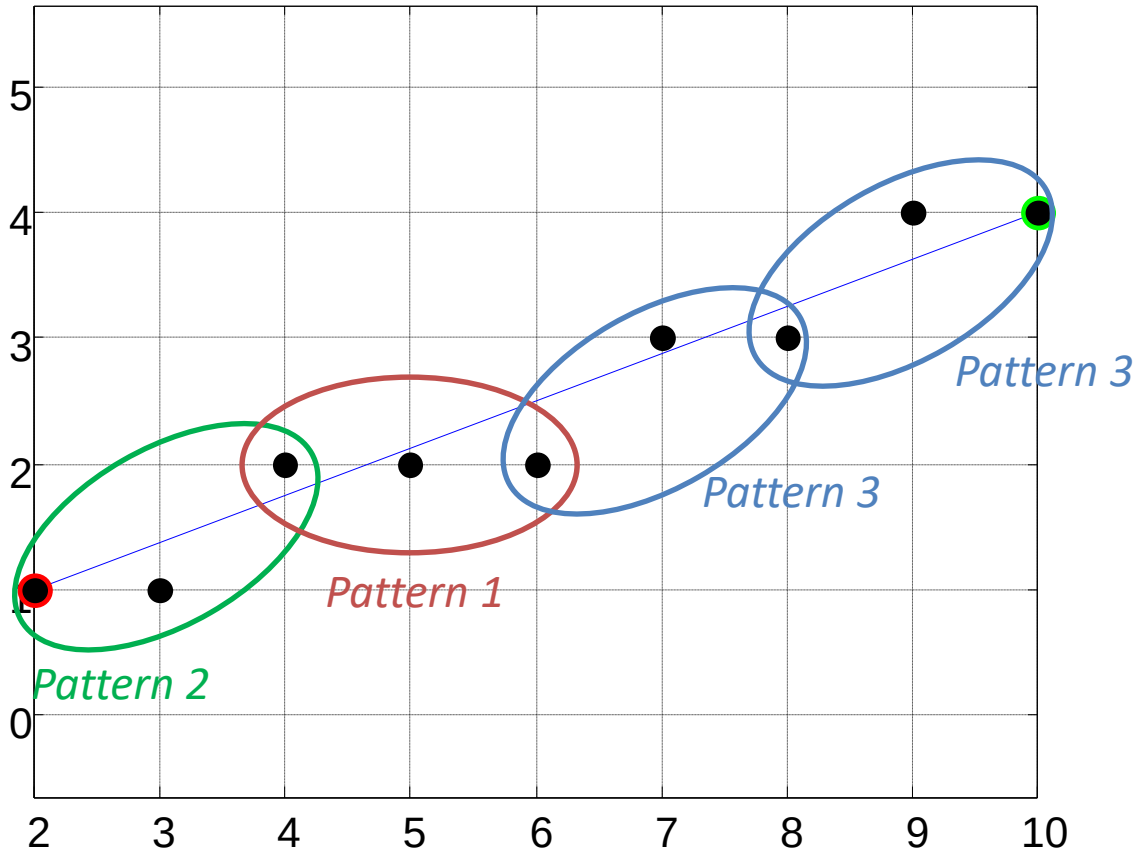
Iterationen

x	y	d	*Entscheidung für nächsten Schritt*
4	2	$d_1 = d_0 + inc_{P23} = 4 - 4 = \mathbf{0}$	da $d_1 \leq 0$ ⇒ Pattern 1 zeichnen
6	2	$d_2 = d_1 + inc_{P1} = 0 + 12 = \mathbf{12}$	da $d_2 > 0$ ⇒ $cond$ prüfen. da $d_2 > cond$ ⇒ Pattern 3 zeichnen
8	3	$d_3 = d_2 + inc_{P23} = 12 - 4 = \mathbf{8}$	da $d_3 > 0$ ⇒ $cond$ prüfen. da $d_3 > cond$ ⇒ Pattern 3 zeichnen
10	4	While-Bedingung $x < v_{1,x}$ nicht mehr erfüllt → Ende.	

Lösung (6)

Skizze:

$d_0 = 4.0$
$d_1 = 0.0$
$d_2 = 12.0$
$d_3 = 8.0$



Lösung (7)

3. Berechnen Sie mit dem Bresenham (Midpoint Circle)-Algorithmus einen Kreis mit Mittelpunkt im Ursprung und Radius $R = 10$.

a) Welche Werte für die Entscheidungsvariable ergeben sich?

b) Skizzieren Sie den Kreis in einem Koordinatensystem und geben Sie die eingefärbten Pixel an

Das Zeichnen beginnt bei 12 Uhr, d.h. $(x = 0, y = R = 10)$.

Im zweiten Oktant wird daher Pixel $(0{,}10)$ eingefärbt. Nutzt man die Symmetrieüberlegungen aus, so werden zusätzlich die folgenden Pixel gefärbt:

$(y, x) \quad = (10{,}0)$
$(y, -x) \quad = (10{,}0)$
$(x, -y) \quad = (0, -10)$
$(-x, -y) \quad = (0, -10)$
$(-y, -x) \quad = (-10{,}0)$
$(-y, x) \quad = (-10, 0)$
$(-x, y) \quad = (0{,}10)$

Lösung (8)

Initialisierungsschritt:

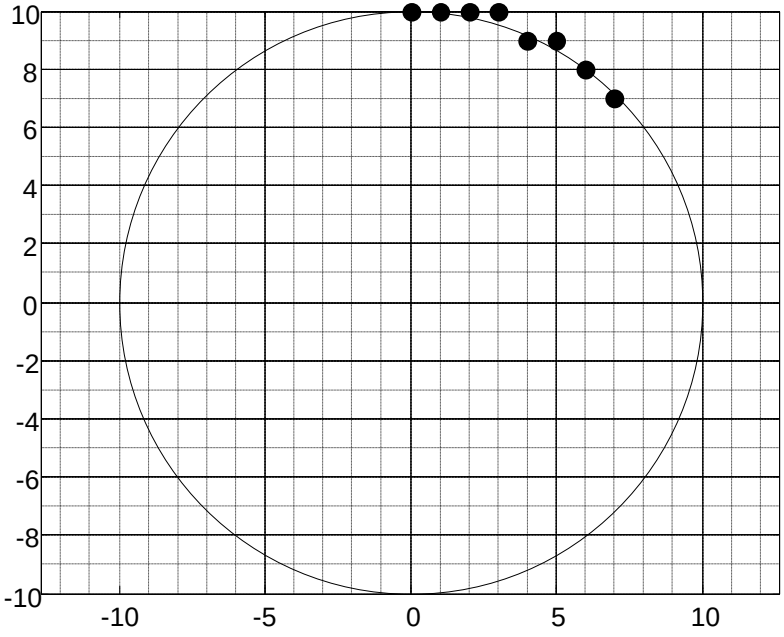
- $d_0 = \frac{5}{4} - R = -\mathbf{8,75}.$ da $d_0 < 0 \Rightarrow E$

Iterationen

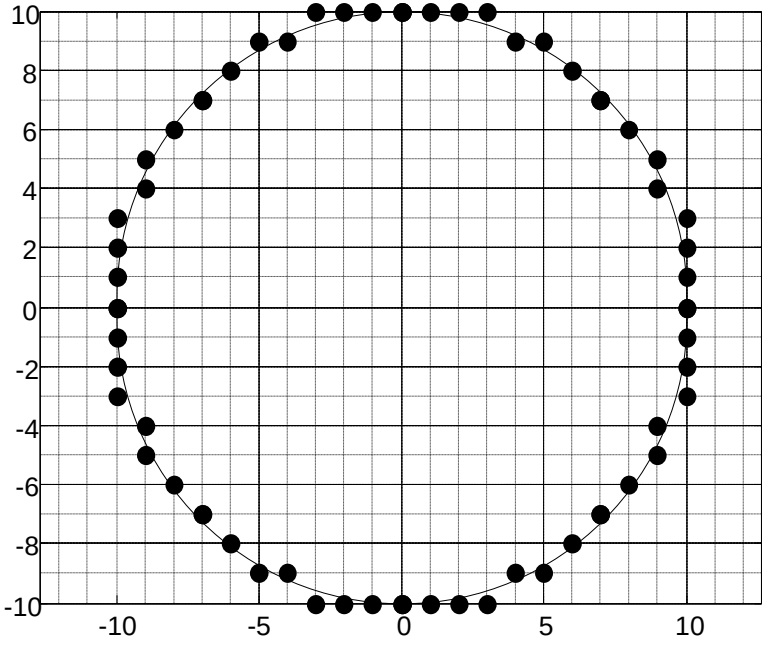
x	y	d	***neues*** x	***neues*** y	*Entscheidung für nächsten Schritt*
0	10	$d_1 = d_0 + 2x + 3 = -8{,}75 + 2 \cdot 0 + 3 = -5{,}75$	1	10	da $d_1 < 0 \Rightarrow$ E
1	10	$d_2 = d_1 + 2x + 3 = -5{,}75 + 2 \cdot 1 + 3 = -0{,}75$	2	10	da $d_2 < 0 \Rightarrow$ E
2	10	$d_3 = d_2 + 2x + 3 = -0{,}75 + 2 \cdot 2 + 3 = 6{,}25$	3	10	da $d_3 \geq 0 \Rightarrow$ SE
3	10	$d_4 = d_3 + 2(x - y) + 5 = 6{,}25 + 2(3 - 10) + 5 = -2{,}75$	4	9	da $d_4 < 0 \Rightarrow$ E
4	9	$d_5 = d_4 + 2x + 3 = -2{,}75 + 2 \cdot 4 + 3 = 8{,}25$	5	9	da $d_5 \geq 0 \Rightarrow$ SE
5	9	$d_6 = d_5 + 2(x - y) + 5 = 8{,}25 + 2(5 - 9) + 5 = 5{,}25$	6	8	da $d_6 \geq 0 \Rightarrow$ SE
6	8	$d_7 = d_6 + 2(x - y) + 5 = 5{,}25 + 2(6 - 8) + 5 = 6{,}25$	7	7	da $d_7 \geq 0 \Rightarrow$ SE
7	7	While-Bedingung $y > x$ nicht mehr erfüllt → Ende.			

Lösung (9)

Skizzen



2. Oktant



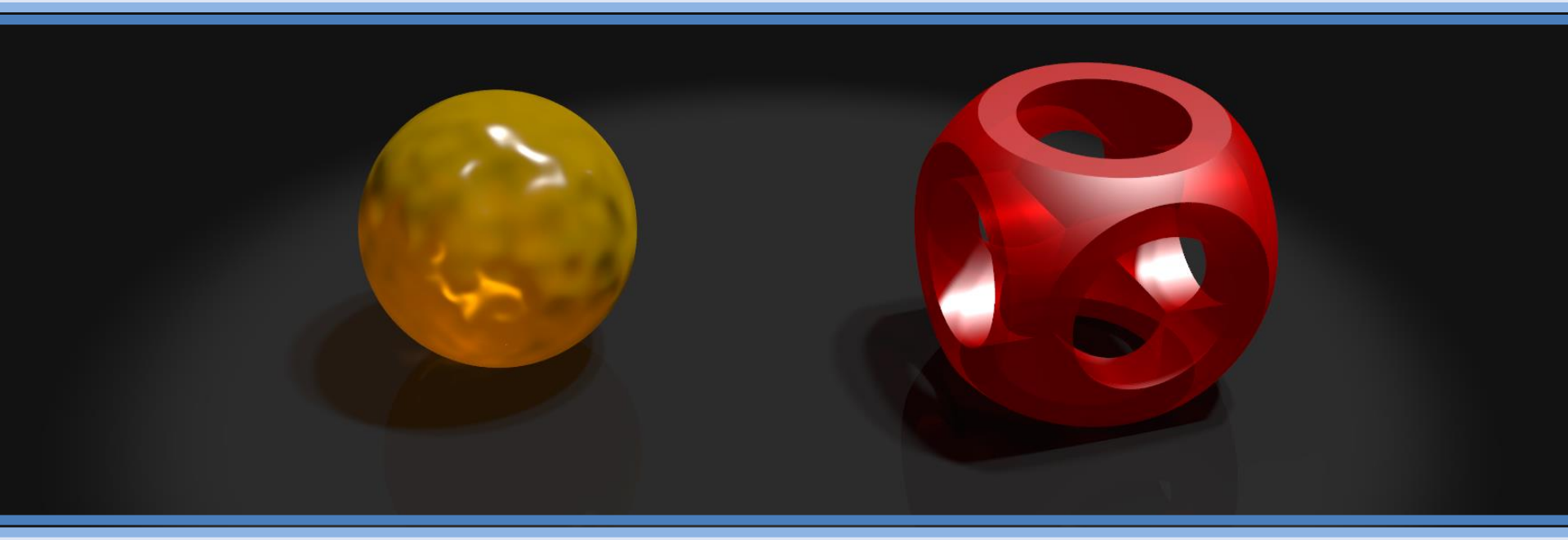
Gesamter Kreis aus Symmetrieüberlegung

Übungsfragen Kapitel 5

- Was ist Clipping?
- Warum kommt bei Linien unterschiedlicher Steigung zu Intensitätsschwankungen? Wie kann man Abhilfe schaffen?
- Beschreiben Sie ein Verfahren zum Antialiasing von Linien

Computergrafik

T. Hopp



Themenübersicht

1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. Zeichenalgorithmen
6. **Buffer-Konzepte**
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. Raytracing
11. Volumenvisualisierung

Buffer in der Computergrafik

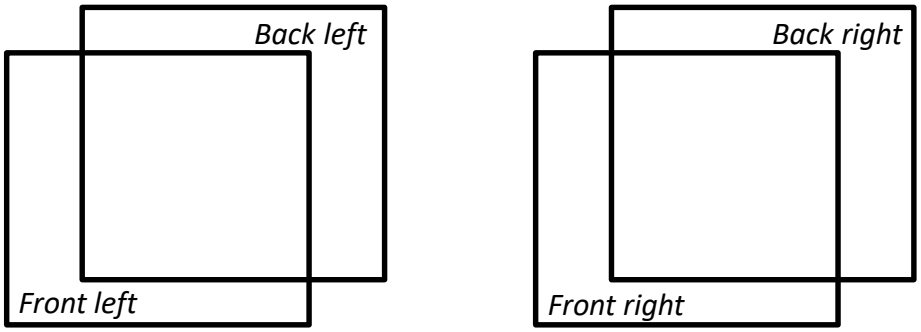
- Gleichmäßig für alle Pixel gespeicherte Daten nennt man Buffer (Pufferspeicher)
- Der Frame-Buffer (auch Color Buffer) umfasst alle anderen Buffer
  - = Bildspeicher
  - Entspricht einer digitalen Kopie des Monitorbildes
  - Größe abhängig von verwendeter Auflösung und Farbtiefe

- Wir betrachten einige Buffer-Konzepte, die zusammen den Frame-Buffer füllen:
  - Double und Stereo Buffer
  - Depth Buffer
  - Stencil Buffer
  - Accumulation Buffer

6.1. DOUBLE U. STEREO BUFFERING

Double Buffering

- Single Buffering: Löschen & Bildaufbau dauert oft zu lange um den Neuaufbau des Bildes für das Auge unsichtbar zu machen
  - Typisches „Flackern“ / Verwischen: vor allem bei Animationen/Bewegung störend
- Double Buffering schafft Abhilfe:
  - Unterteilung des Frame-Buffers in Front-Buffer und Back-Buffer
  - Front-Buffer wird ausgelesen und auf dem Bildschirm dargestellt
  - Back-Buffer wird zeitgleich gelöscht und mit neu berechnetem Bild gefüllt
  - Anschließend Austausch von Front- und Back-Buffer (*Swap*)
- Bei stereofähigem Grafiksystem: zwei zusätzliche Buffer für links und rechts:



Double Buffering

- In OpenGL:
  - Setzen des Buffermodus beim Initialisieren eines Fensters:

    `glutInitDisplayMode(…|GLUT_SINGLE|…)`

  - GLUT_SINGLE: Single Buffering
  - GLUT_DOUBLE: Double Buffering
  - GLUT_STEREO: Stereo Buffering

- Unterstützte Modi des Systems können mit **`glGetBooleanv(GL_DOUBLEBUFFER)`** bzw. **`glGetBooleanv(GL_STEREO)`** abgefragt werden
- Jedes System hat mindestens einen Colorbuffer (Frame Buffer): *Front left*
- Der Tausch zwischen Front- und Back-Buffer erfolgt am Ende der Zeichenroutine durch den Aufruf **`glutSwapBuffers();`**
- Neuzeichnen der Szene beginnt mit Löschen des jeweiligen Buffers

Tausch-Operation

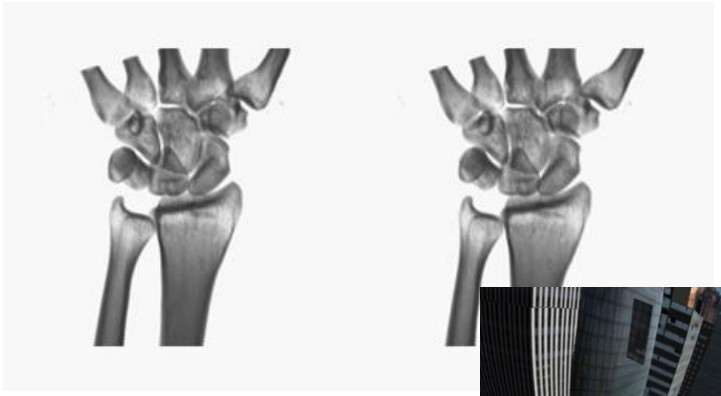
- Wichtig: Synchronisation zwischen Anzeige-Framerate und Generierungs-Framerate
- Ohne gesonderte Behandlung (bei deaktiviertem VSYNC):
  - Sofortiger Tausch zwischen Front- und Back-Buffer sobald der Back-Buffer gefüllt ist
  - Ist das Bild auf dem Bildschirm noch nicht vollständig aufgebaut, wird ab der aktuellen Stelle auf dem Bildschirm bereits das neue Bild dargestellt: Tearing-Artefakte
- Vertikale Synchronisation (VSYNC): verhindert Aktualisierung der Bilddaten, während der Bildschirm das Bild aufbaut.
- Bei aktiviertem VSYNC:
  - Tausch findet erst statt wenn der Bildschirm das Bild vollständig aufgebaut hat
  - Künstliche Reduktion der Leistung der GPU
  - Kompensation durch Dreifach-Pufferung
- Typische Bildwiederholrate:
  - 60Hz , 75 Hz bei LCD Bildschirmen

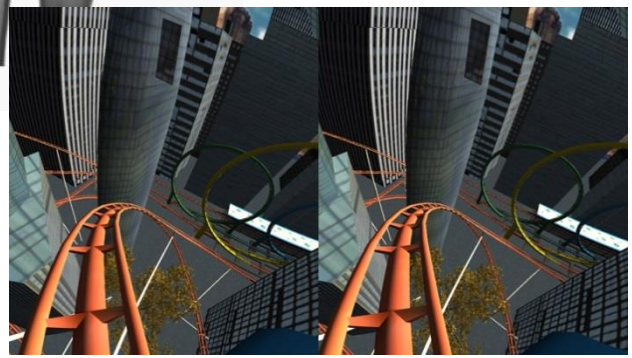
http://www.geforce.com/hardware/technology/adaptive-vsync/technology

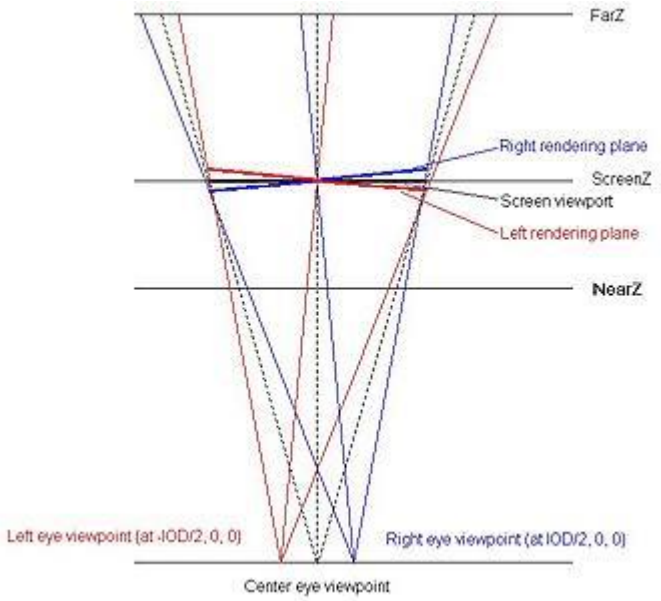
Stereo Buffering

- Bei Stereo-Darstellungen werden Bilder für das linke und rechte Auge aus unterschiedlicher Kameraposition erzeugt. → Stereosehen
  - Vergl. **`gluLookAt`**-Befehl
- Der Buffer in den gezeichnet werden soll wird ausgewählt durch:

```
glDrawBuffer(GL_BACK_RIGHT);
```







http://www.orthostereo.com/geometryopengl.html
http://www.pcadvisor.co.uk/test-centre/gadget/12-best-google-cardboard-apps-2016-uk-3585299/

Umsetzung von Stereo-Darstellungen

Passiv-Stereo: Trennung der Information für linkes und rechtes Auge durch Filter

- Farb-Filter: meist rot-grün Filter
  - Farbinformation geht u.U. verloren

- Polfilter: Vertikal- und horizontal polarisiertes Licht
  - Farbinformation bleibt erhalten
  - Betrachter darf Kopf nicht neigen. Abhilfe: zirkular polarisiertes Licht

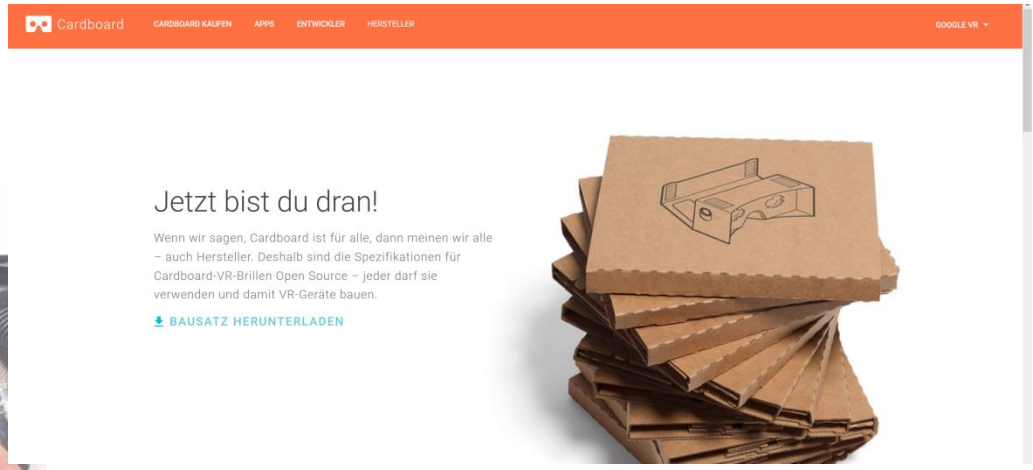
Aktiv-Stereo: Abwechselnde Darstellung von Vollbildern für das linke und rechte Auge

- Trennung der Bilder für das Auge durch synchronisierte Shutterbrille

https://www.3d-brillen.de/3d-brillen/3d_brille_anaglyph_rot_gruen.html
https://www.conrad.de/de/passive-3d-brille-hama-3d-polfilterbrille-943701.html
http://www.hifi-forum.de

Umsetzung von Stereo-Darstellungen

- … oder ganz einfach durch zwei getrennte Displays für die Augen



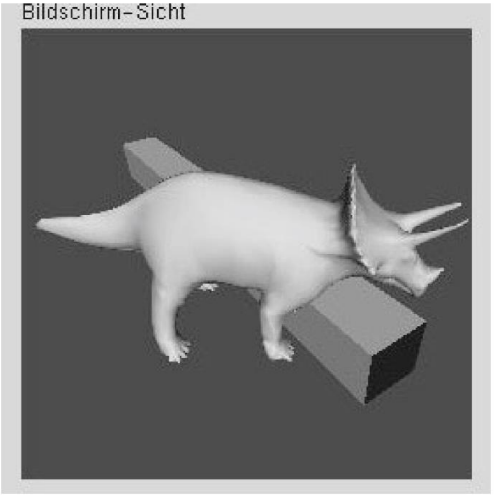
https://vr.google.com/intl/de_de/cardboard/manufacturers/

http://www.pcwelt.de/ratgeber/Die-besten-VR-Apps-fuer-Handy-und-Cardboard-9976062.html
https://de.wikipedia.org/wiki/Oculus_Rift

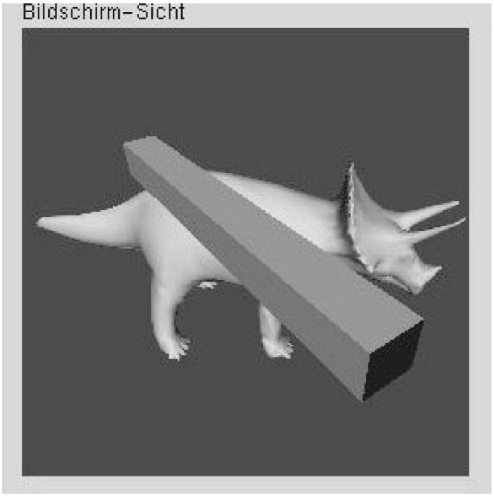
6.2. DEPTH- BZW. Z-BUFFER

Verdeckung

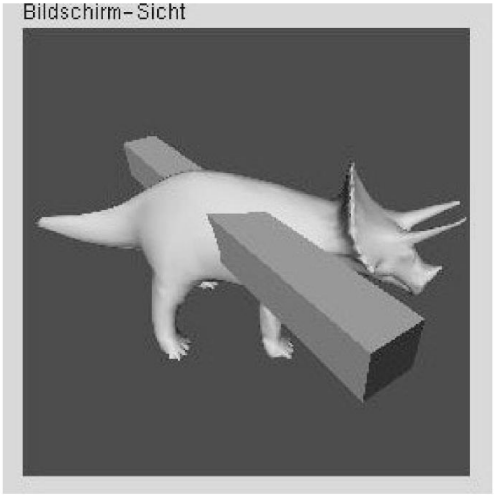
- Gegenseitige Verdeckung von Objekten gibt räumlichen Eindruck und Hinweis auf Entfernung der Objekte vom Augpunkt.
- I.a. werden Objekte in der Reihenfolge ihrer Definition gezeichnet
  - D.h. Pixel erhält ohne weitere Behandlung die Farbe des zuletzt gezeichneten Objektes



Dinosaurier zuletzt gezeichnet



Quader zuletzt gezeichnet



Korrekte Darstellung (z-Buffer)

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

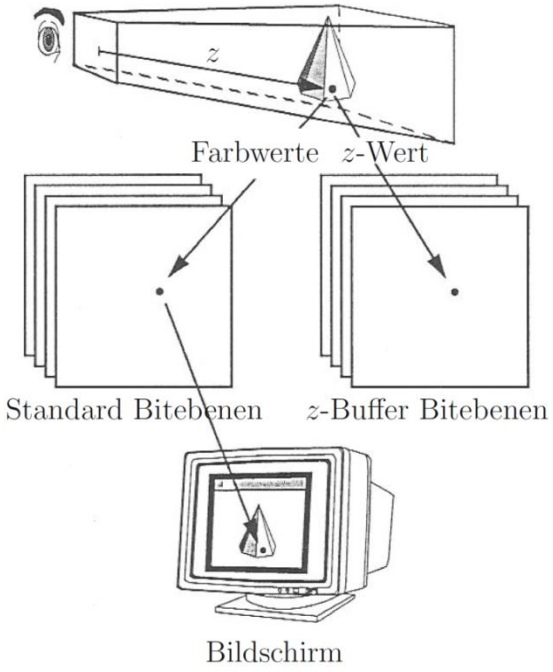
Maler-Algorithmus

- Ermöglicht Unabhängigkeit von Reihenfolge der Definition
- Einfacher zweistufiger Ablauf:
  1. Sortieren aller Objekte/Polygone in Bezug auf ihren Abstand zum Augpunkt
  2. Zeichnen aller Objekte/Polygone in der neuen Reihenfolge, beginnend mit entferntestem

- Nachteile:
  - Hoher Sortieraufwand, speziell bei großer Objekt-/Polygonanzahl
  - Bei Objekt- oder Augpunktbewegung Neu-Sortierung erforderlich
  - Gegenseitige Durchdringung von Objekten/Polygonen kann nicht abgebildet werden

Z-Buffer

- Grundidee: nutzen zusätzlichen Speichers für Tiefeninformation (z-Wert) für jedes Pixel
  - Für jedes Pixel eines Objektes: Test ob es näher am Augpunkt liegt als das vorher gezeichnete. → Wenn ja: in den Color-Buffer schreiben.



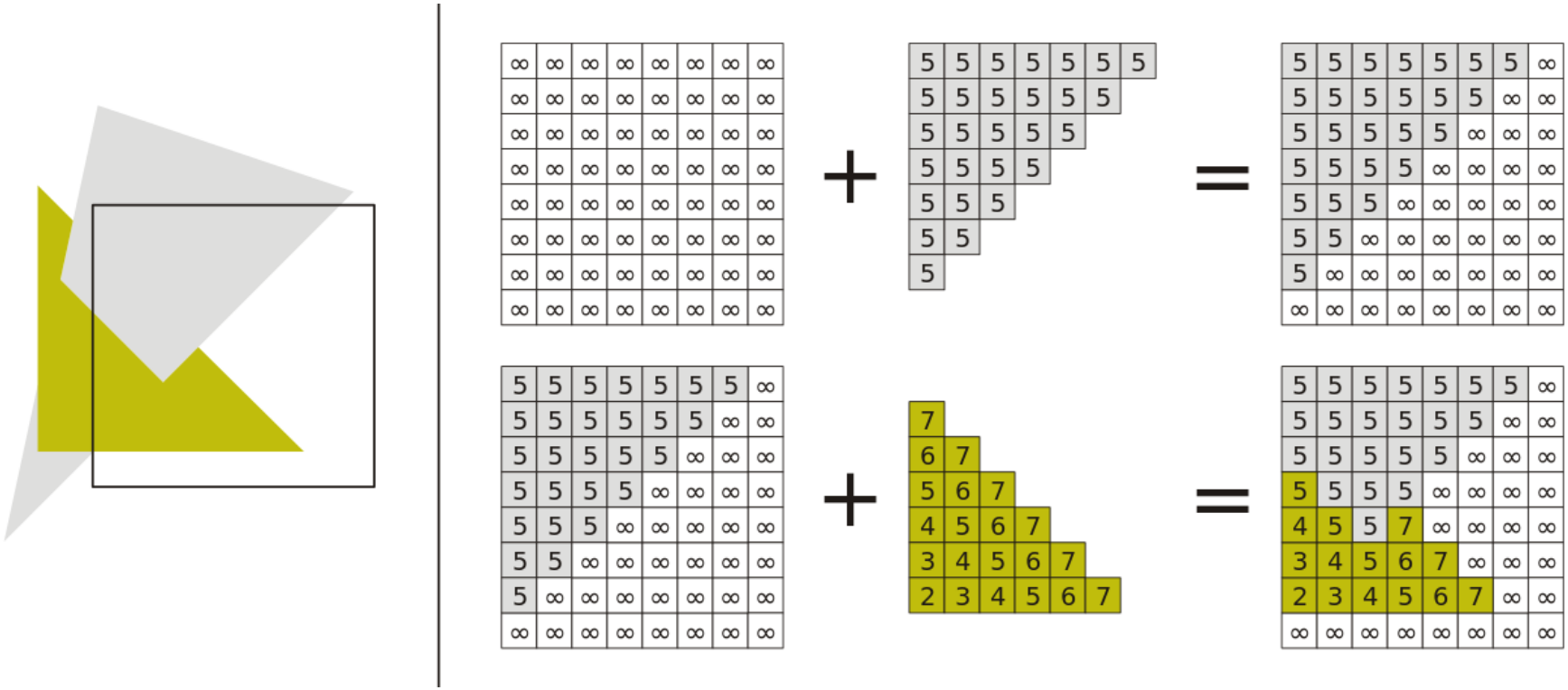
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Z-Buffer-Algorithmus

- Pseudocode:

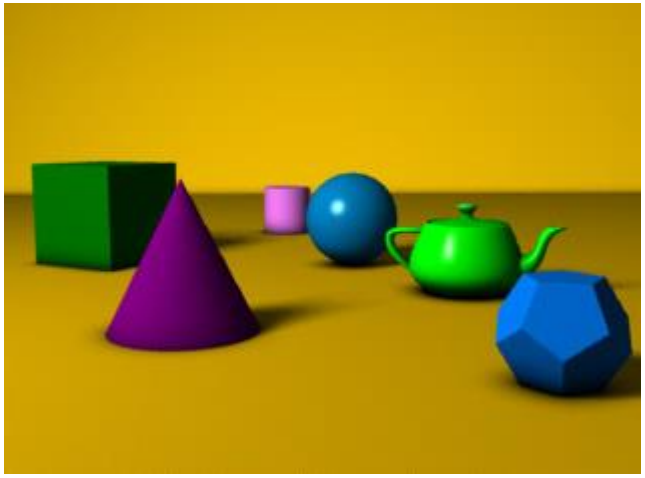
```
void zBuffer() {
   int x, y;
   for (y = 0; y < YMAX; y++){
      for (x = 0; x < XMAX; x++){
         writePixel(x,y, BACKGROUND_COLOR);  /* Clear color */
         writeZ(x,y, DEPTH);                 /* Clear depth */
      }
   }

   for(eachPolygon){
      for(eachPixel in polygon's projection){
         pz = polygon's z-value at pixel's coordinates(x,y);
         if (pz <= readZ(x,y)){              /* New point is nearer */
            writeZ(x,y,pz);
            writePixel(x,y, polygon's color at (x,y));
         }
      }
   }
}
```

Z-Buffer-Algorithmus



https://de.wikipedia.org/wiki/Z-Buffer

Z-Buffer-Algorithmus



Generiertes Bild

Zugehöriger z-Buffer

https://de.wikipedia.org/wiki/Z-Buffer

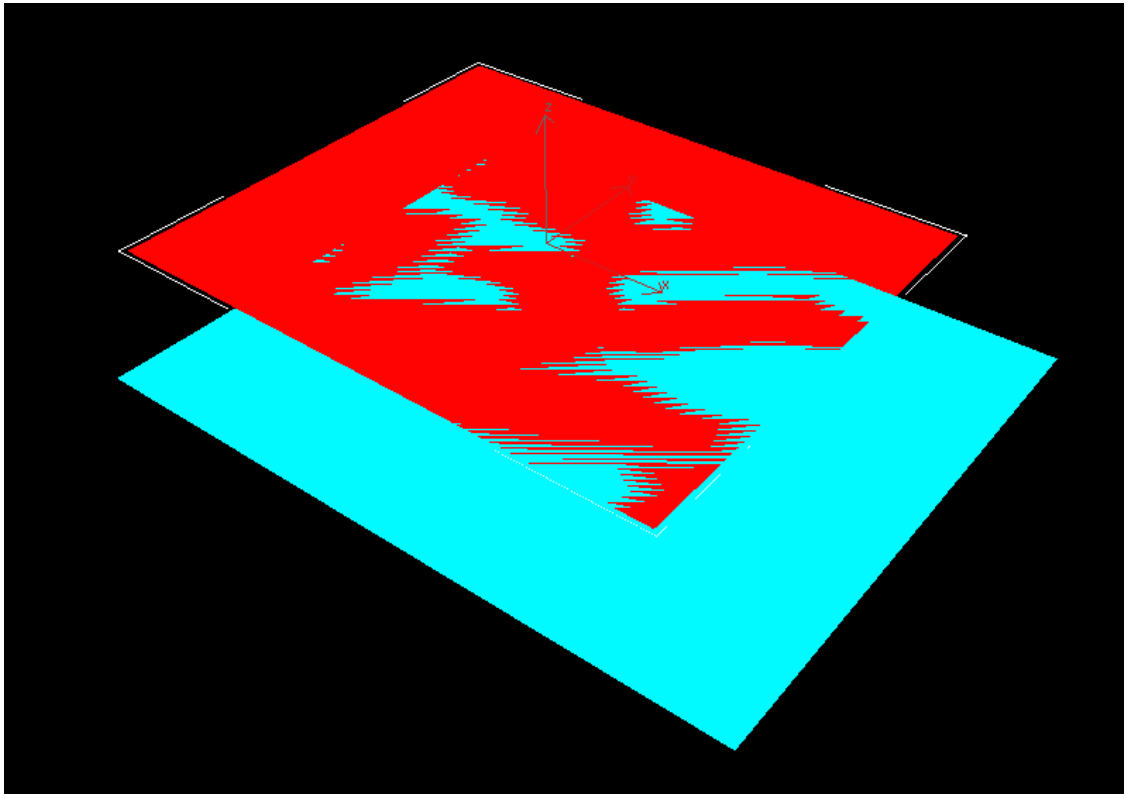
Z-Buffer-Algorithmus

- Vorteile:
  - Kein aufwändiges Sortieren von Objekten nötig
  - Pixelgenaue Verdeckung von sich durchdringenden Objekten
  - Z-Wert-Berechnung einfach und schnell
    - Eckpunkte von Polygonen schon vorhanden durch Transformationskette
    - Lineare Interpolation für jedes Pixel eines Polygons
  - einfache Parallelisierung möglich

- Probleme:
  - Begrenzte Genauigkeit des z-Wertes: Z-Buffer-Flimmern („Z-Fighting“)
  - Transparente Oberflächen werden nicht korrekt berücksichtigt

Z-Buffer-Algorithmus

- Z-Fighting



https://de.wikipedia.org/wiki/Z-Buffer

Z-Buffer-Algorithmus

- Anwendung in OpenGL:
  - Aktivierung des z-Buffers: **`glEnable(GL_DEPTH_TEST);`**
  - Anfordern des zusätzlichen Bildspeichers: **`glutInitDisplayMode(…|GLUT_DEPTH|…);`**

- Initialisierung des z-Buffers:
  - Default: z-Werte zwischen 0.0 (*near clipping plane*) und 1.0 (*far clipping plane*).
  - Zuweisen eines Initialisierungswertes: **`glClearDepth(Gldouble depth);`**
  - In der Regel der Maximalwert
  - Initialisierungswert der Hintergrundfarbe:

    `glClearColor(Gldouble r, Gldouble g, Gldouble b);`
  - Löschen ist eine der teuersten Aktionen im Darstellungsprozess: Jedes Pixel muss angesteuert und zurückgesetzt werden!
  - Gleichzeitiges Zurücksetzen von Farbe und z-Buffer:

    `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
  - Einschränkung des Wertebereichs, falls nötig:

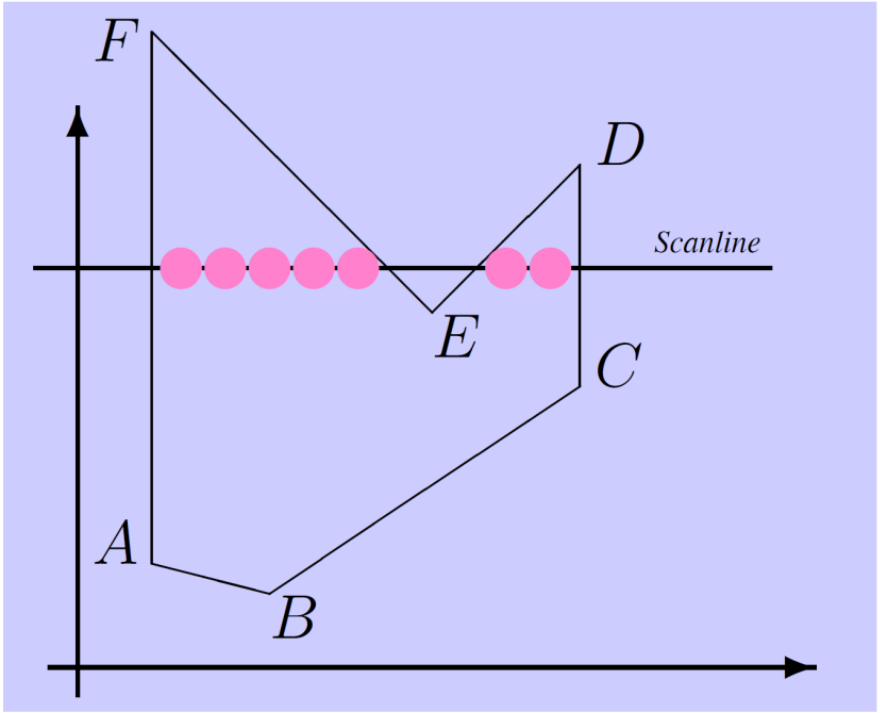
    `glDepthRange(Gldouble near, Gldouble far);`

Z-Buffer-Algorithmus

- Auswahl der Vergleichsoperation: **`glDepthFunc(Glenum operator)`**
- Folgende Vergleichsoperationen stehen zur Verfügung:
  - **`GL_LESS:`** <, kleiner (Default Operation)
  - **`GL_NEVER:`** 0, liefert immer den Wahrheitswert **`false.`**
  - **`GL_EQUAL:`** =, gleich
  - **`GL_LEQUAL:`** <=, kleiner oder gleich
  - **`GL_GREATER:`** >, größer
  - **`GL_GEQUAL:`** >=, größer oder gleich
  - **`GL_LESS:`** <, kleiner
  - **`GL_NOTEQUAL:`** ≠, ungleich
  - **`GL_ALWAYS:`** 1, liefert immer den Wahrheitswert **`true`**

Z-Wert-Bestimmung

- Zur Erinnerung: Zeichnen per Scanline:



- z-Werte nach Anwendung der Transformationskette bekannt für jeden Vertex

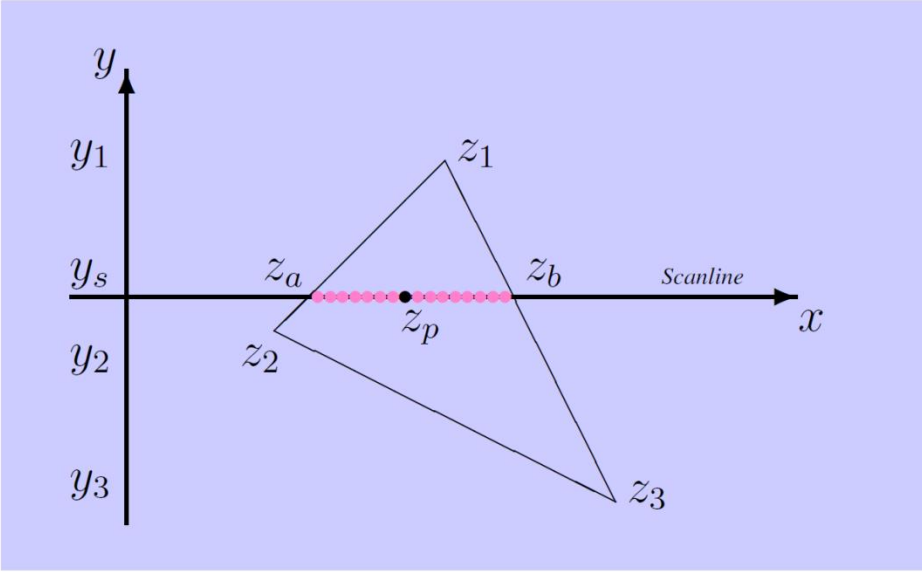
S. Krömker: Computergrafik I, Universität Heidelberg

Z-Wert-Bestimmung

- Lineare Interpolation der z-Werte entlang der Scanline für alle Pixel:

$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$
$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$



S. Krömker: Computergrafik I, Universität Heidelberg

Z-Wert-Bestimmung: Beschleunigung

- Ebenengleichung zur Darstellung des Polygons:

$$Ax + By + Cz + D = 0 \quad \Rightarrow \quad z(x,y) = \frac{-Ax - By - D}{C}$$

  - (A, B, C) stellt den Normalenvektor der Ebene dar, D ist Abstand vom Ursprung
- Beschleunigung durch Berechnung aus dem vorherigen Wert:

$$z(x + \Delta x, y) = z(x,y) - \frac{A}{C}(\Delta x)$$

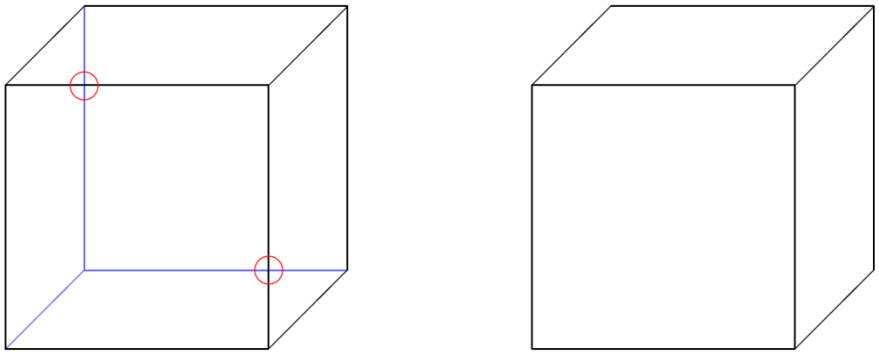
  - Typischerweise ist $\Delta x = 1$ und $^{A}/_{C}$ konstant.
- So ergibt sich eine Subtraktion einer Konstanten entlang einer Scanline:

$$z(x + \Delta x, y) = z(x,y) - const$$

- Gleiches gilt für die Berechnung des ersten z-Wertes der darauffolgenden Scanline!
- Bei vielen Polygonen: Beschleunigung durch Vorsortierung der Objekte nach z-Werten

Z-Buffer: Drahtgittermodelle

- Drahtgittermodelle: Nur Schnittpunkte der Linien werden durch z-Buffer berücksichtigt.



- Hidden Line-Darstellung (gefüllte Objekte mit Umriss):
  - Objekt als gefülltes Polygon zeichnen: **`glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);`**
  - Objekt erneut als Linienzug zeichnen: **`glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);`**
- Problem: benachbarte Pixel an Polygonkante haben (nahezu) gleiche z-Werte („Stitching“ durch „z-Fighting“)

S. Krömker: Computergrafik I, Universität Heidelberg

Hidden-Line

- Abhilfe für Stitching: Polygonoffset
  - `glEnable(GL_POLYGON_OFFSET_FILL);`
  - `glEnable(GL_POLYGON_OFFSET_LINE);`
  - `glEnable(GL_POLYGON_OFFSET_POINT);`
- Addiert zum jeweiligen z-Wert einen Offset, der wie folgt berechnet wird:

$$o = m \cdot factor + r \cdot units$$

Aktuell auflösbarer z-Wert

Maximaler z-Unterschied des Objektes

- *factor* und *units* werden vom Anwendungsprogramm vorgegeben:

```
glPolygonOffset(Glfloat factor, Glfloat units);
```

Hidden-Line

- Wahl von *factor* und *units*:
  - Standardmäßig *factor=1.0* und *units=1.0*
  - Bei größerer Linienstärke sollte *factor* erhöht werden
  - Bei perspektivischer Projektion: *factor* an z-Wert koppeln (Offset größer für weiter entfernte Objekte)
  - Besser zu viel Offset als zu wenig

- Für Hidden-Line-Darstellung zwei Möglichkeiten:
  - Offset für gefülltes Polygon >0 → Polygon erscheint weiter hinten
  - Offset für Linie <0 → Linie erscheint weiter vorne

6.3 STENCIL BUFFER

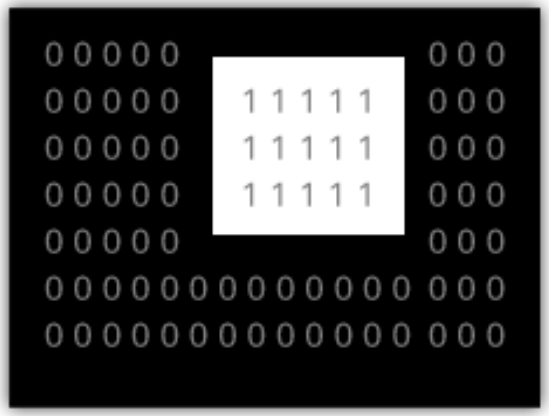
Stencil Buffer

- = Schablonenspeicher: Erlaubt pixelweise Zuweisung von Eigenschaften
- 1-Bit-Stencil: 2 Zustände
  - Dient dem Maskieren von Bereichen, die weiter verarbeitet werden sollen oder die nicht gerendert werden müssen.

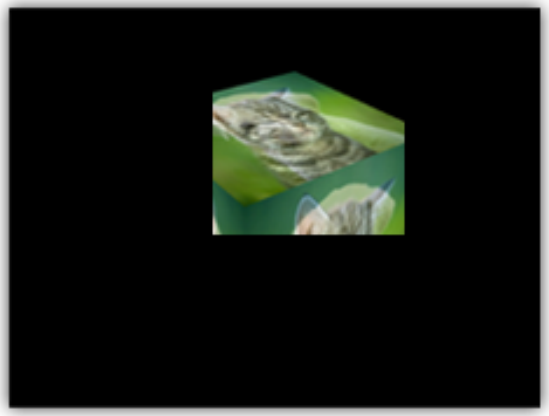
- In OpenGL:
  - Aktivierung mit **`glEnable(GL_STENCIL_TEST);`**
  - Festlegung des Stencil-Tests: **`glStencilFunc(op, ref, mask);`**
    - op: **`GL_NEVER, GL_ALWAYS, GL_EQUAL, GL_NOTEQUAL, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GREATER`**
    - ref: Integer-Wert mit dem verglichen werden soll
  - Festlegung der Aktion: **`glStencilOp(sfail, dpfail, dppass);`**
    - sfail: Test der glStencilFunc nicht erfolgreich
    - dpfail: Test der glStencilFunc erfolgreich, Depth-Test aber nicht
    - dppass: Test der glStencilFunc und Depth-Test erfolgreich.
    - Kann jeweils den Wert **`GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_INCR_WRAP, GL_DECR, GL_DECR_WRAP, GL_INVERT`** annehmen.

Stencil Buffer

Color buffer without stencil test



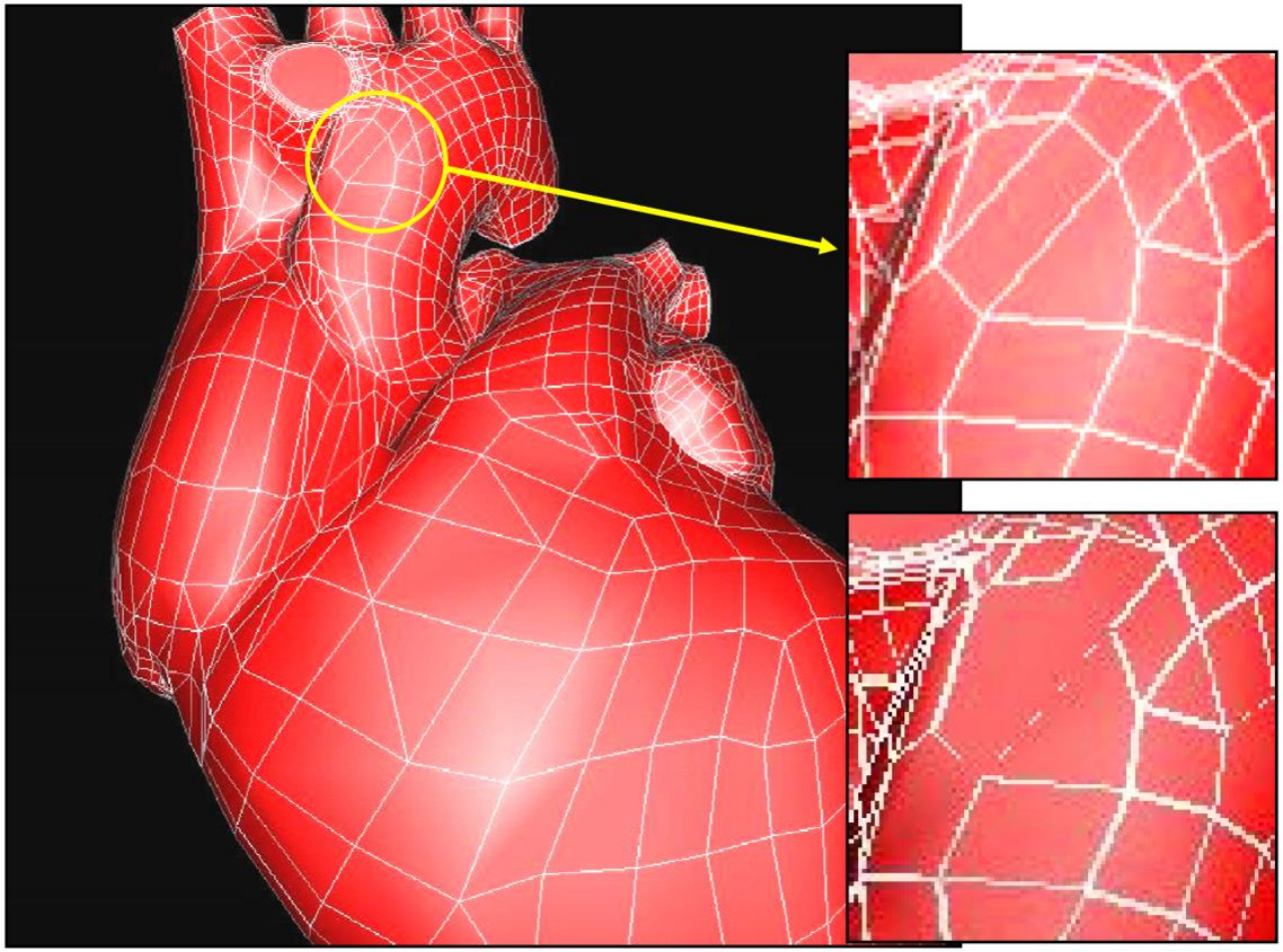
Stencil buffer



Color buffer with stencil test

https://open.gl/depthstencils

Hidden Line mit Stencil Buffer



Mit Stencil Buffer

Ohne Korrektur: Z-Fighting

M. Bailey: Stencil Buffer Examples, Oregon State University

6.4 ACCUMULATION BUFFER

Accumulation Buffer

- Vervielfältigung des Color Buffers:
  - Sammeln von Frames
  - Definierte Aktion zum Zusammenfassen der Frames und Übergabe an Color Buffer

- Nutzung indem Durchschnitt mehrerer Bilder mit unterschiedlichen Einstellungen berechnet wird, z.B. für:
  - Tiefenunschärfe: Änderung des Fokus der virtuellen Kamera
  - Antialiasing komplexer Szenen: leichtes Verschieben der Kamera
  - Bewegungsunschärfe (Motion Blur): unterschiedliche Zeitpunkte
  - Weiche Schatten: durch unterschiedliche Interpretation der Position der Lichtquelle.

Accumulation Buffer

- OpenGL-Befehle:
  - Initialisierung: **`glutInitDisplayMode(…|GLUT_ACCUM|…);`**
  - Operationen auf dem Accumulation Buffer: **`glAccum(operation, f);`**

operation	Formel	Bedeutung
GL_ACCUM	$C'_{acc} = f \cdot C_{col} + C_{acc}$	Addition der mit f skalierten Color-Buffer-Werte und der Accumulation Buffer-Werte
GL_LOAD	$C'_{acc} = f \cdot C_{col}$	Überschreiben des Accumulation Buffer durch mit f skalierten Color-Buffer
GL_ADD	$C'_{acc} = f + C_{acc}$	Addition von f auf die RGBA-Werte des Accumulation Buffer
GL_MULT	$C'_{acc} = f \cdot C_{acc}$	Multiplikation von f mit den RGBA-Werten des Accumulation Buffer
GL_RETURN	$C_{col} = f \cdot C_{acc}$	Kopieren der mit f skalierten Accumulation Buffer-Werte in den aktuellen Color-Buffer

Accumulation Buffer

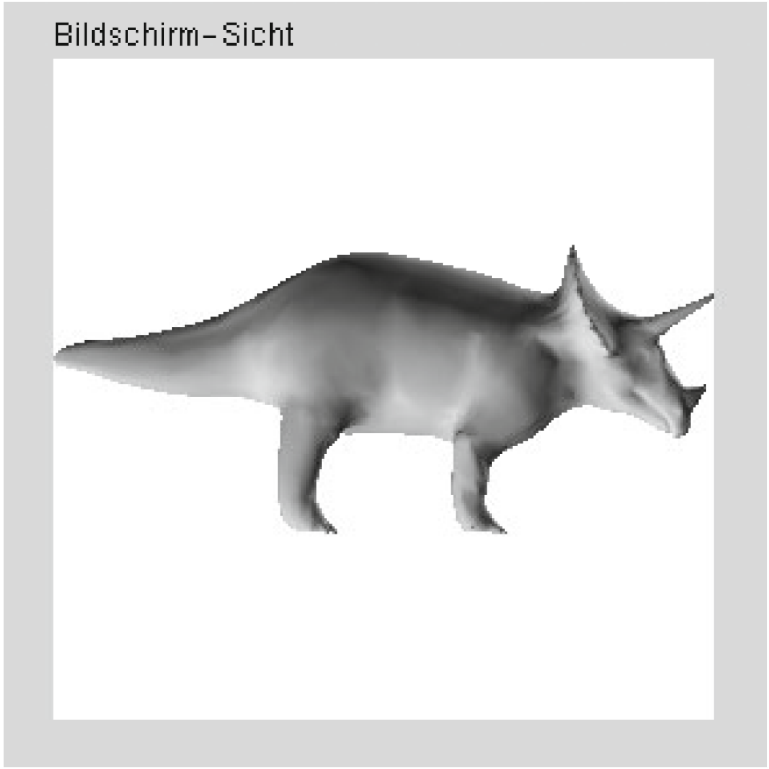
- Beispiel: Szenen-Anti-Aliasing per Accumulation Buffer
  - Idee: Reduzierung des Treppeneffektes durch höhere Abtastrate

- Verarbeitungsschritte:
  1. Zurücksetzen des Accumulation Buffers: **`glClearAccum(background);`**
  2. Simulation einer n-fach höheren Auflösung durch 2^n Bilder die sich um einen Bruchteil eines Pixels unterscheiden.

     Erzeugung der Bilder durch leichte Verschiebung der Kamera

     Aufaddieren der Bilder im Accumulation Buffer: **`glAccum(GL_ACCUM,1.0/2^n);`**
  3. Kopieren des Accumulation Buffers in den Frame Buffer: **`glAccum(GL_RETURN,1.0);`**

Accumulation Buffer



Ohne Anti-Aliasing

Anti-Aliasing mittels Accumulation Buffer

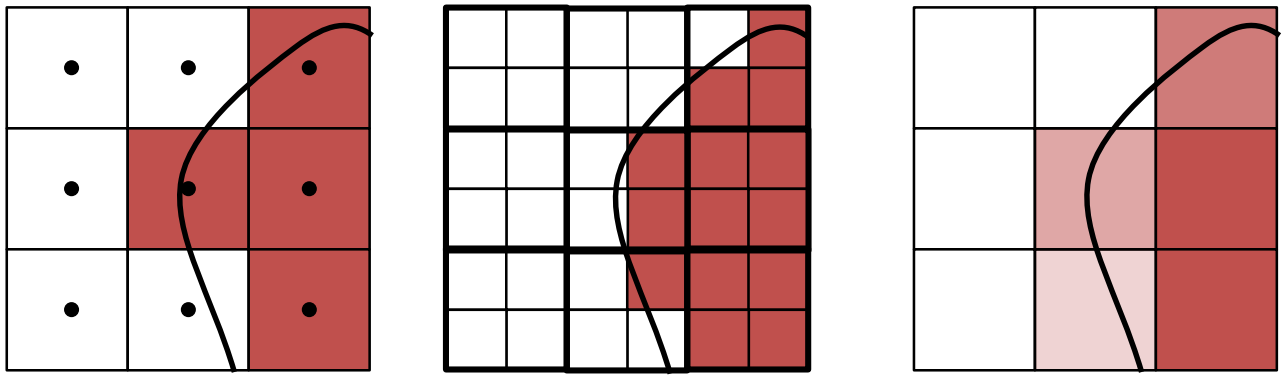
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Szenen-Anti-Aliasing

Supersampling Anti-Aliasing (SSAA)

- Rastern in 2-fach (4-fach, 8-fach) höherer Auflösung als für Viewport erforderlich
  - Pro Pixel: 2x2 / 4x4 / 8x8 / … Subpixel, die voll gerendert werden
  - End-Intensität eines Pixels im Viewport ergibt sich aus Mittelwert der Subpixel

- Erfordert 4-fachen (16-fache, 64-fachen) Speicher und 4-fache (16-fache, 64-fache) Rechenleistung.



Szenen-Anti-Aliasing

Multisample Anti-Aliasing (MSAA)

- Nachteile von SSAA:
  - SSAA rechnet komplette Rendering-Pipeline für alle Subpixel.
  - Aliasing typischerweise nur an Kanten.
- MSAA optimiert SSAA:
  - Pixel-basierte Berechnungen (z.B. Verdeckung) wird nur einmal für das dargestellte Pixel berechnet, nicht für Subpixel.
  - Zusätzlich kann Kantendetektion eingesetzt werden um Verfeinerung zu Subpixeln auf Kantenpixel zu reduzieren.

- GLUT bietet MSAA an:
  - Initialsierung: **`glutInitDisplayMode(GLUT_MULTISAMPLE);`**
  - Aktivierung: **`glEnable(GLUT_MULTISAMPLE);`**

Szenen-Anti-Aliasing

- Bisherige Verfahren betreiben Anti-Aliasing zum Renderzeitpunkt: Vervielfachung der zu rendernden Pixel.
- GPU-Hersteller bieten heute zusätzlich **Post-Rendering-Anti-Aliasing**:
  - Arbeiten auf bereits fertig gerendertem Bild

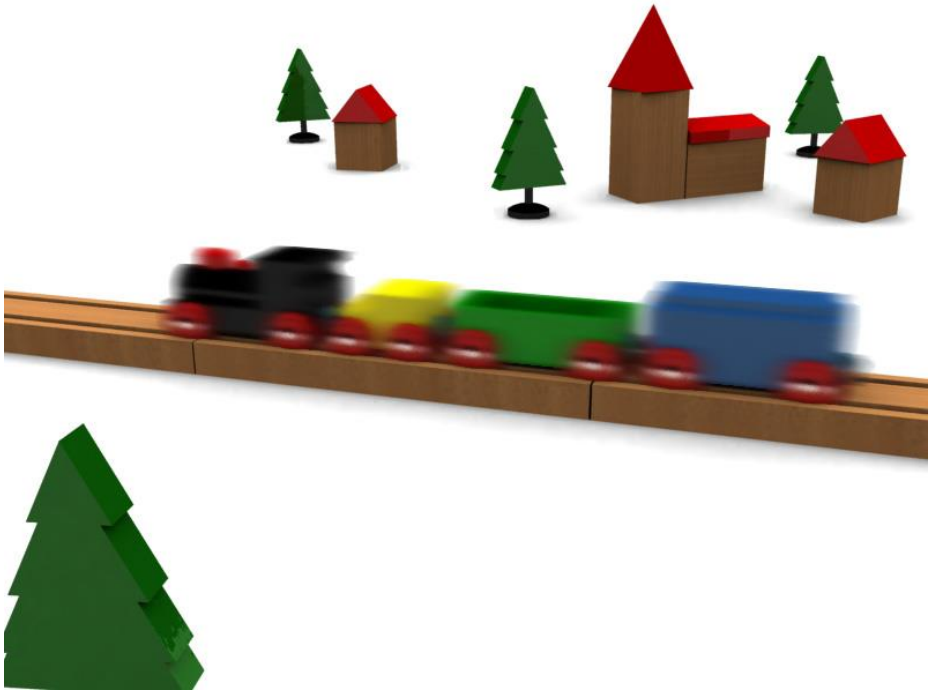
z.B. NVidia FXAA / AMD MLAA:

- Kantendetektion durch Analyse des lokalen Kontrasts
- Glättungsfilter auf detektierte Kantenbereiche

Zeitliches Anti-Aliasing

- Verwischender Effekt durch Akkumulieren mehrerer nacheinander gerenderter Bilder einer Animation („Motion Blurring“)
- Z.B. bei rotierenden Objekten, Eindruck von Bewegung

Motion Blur durch Accumulation Buffer



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“
http://www.dma.ufg.ac.at/app/link/Grundlagen:3D-Grafik/module/13136?step=all

Abfolge der Testoperationen

1. Scissor Test:

   entscheidet über neu zu zeichnende Bildschirmausschnitte

2. Alpha Test:

   gibt an ab welchem Schwellenwert der Durchsichtigkeit nicht mehr gezeichnet werden muss.

3. Stencil Test:

   entscheidet über irregulär geformte Bereiche und die Art, wie darin zu zeichnen ist.

4. Depth Test:

   entscheidet nach z-Wert welches Pixel gezeichnet wird

5. Blending:

   Verrechnet mit dem Hintergrund-Farbwert

6. Dithering:

   mischt die Farben

ZUSAMMENFASSUNG

Zusammenfassung

- Buffer: für alle Pixel gespeicherte Daten
- Double/Stereo-Buffer:
  - Verhindern von Flackern durch Hintergrund-Buffer in den gezeichnet wird
  - Stereo-Darstellung
- Depth- bzw. Z-Buffer:
  - Pixelgenaue Verdeckungsberechnung
  - Z-Buffer enthält für jedes Pixel die Tiefeninformation
  - Interpolation der z-Werte je Pixel anhand der Vertices
- Stencil Buffer
  - Pixelgenaue Maskierung
- Accumulation Buffer
  - Sammeln von Frames
  - Definierte Aktion zum Zusammenfassen der Frames und Übergabe an Color Buffer
  - Nutzung der Darstellung von Effekten wie z.B. Tiefenschärfe, Bewegungsunschärfe, Szenen-Antialiasing

ÜBUNGSAUFGABEN

Übungsaufgaben

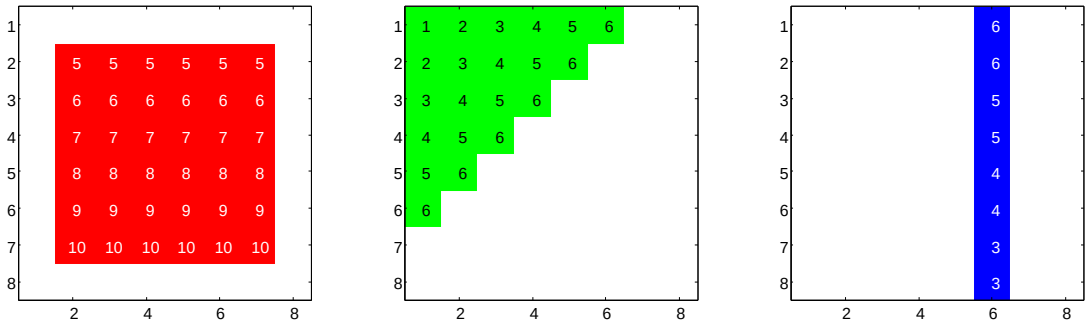
3. Gegeben sind die folgenden Polygone mit ihren zugehörigen z-Koordinaten pro Pixel.

a) Zeichnen Sie diese in der definierten Reihenfolge unter Verwendung des z-Buffers mit der Operation „<=“ in das vorgegebene 8x8 Pixelfeld und geben Sie den z-Buffer an.

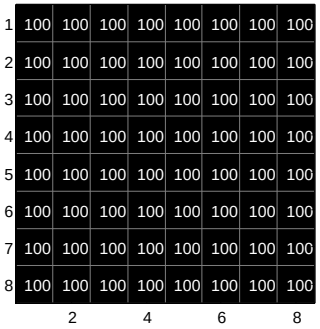
b) Wie ändert sich die Darstellung bei der Operation „<“

c) Wie ändert sich die Darstellung wenn die Objekte in umgekehrter Reihenfolge mit der Operation „<=“ gezeichnet werden?

Einzufügende Muster:



Initiales Bild und z-Buffer:

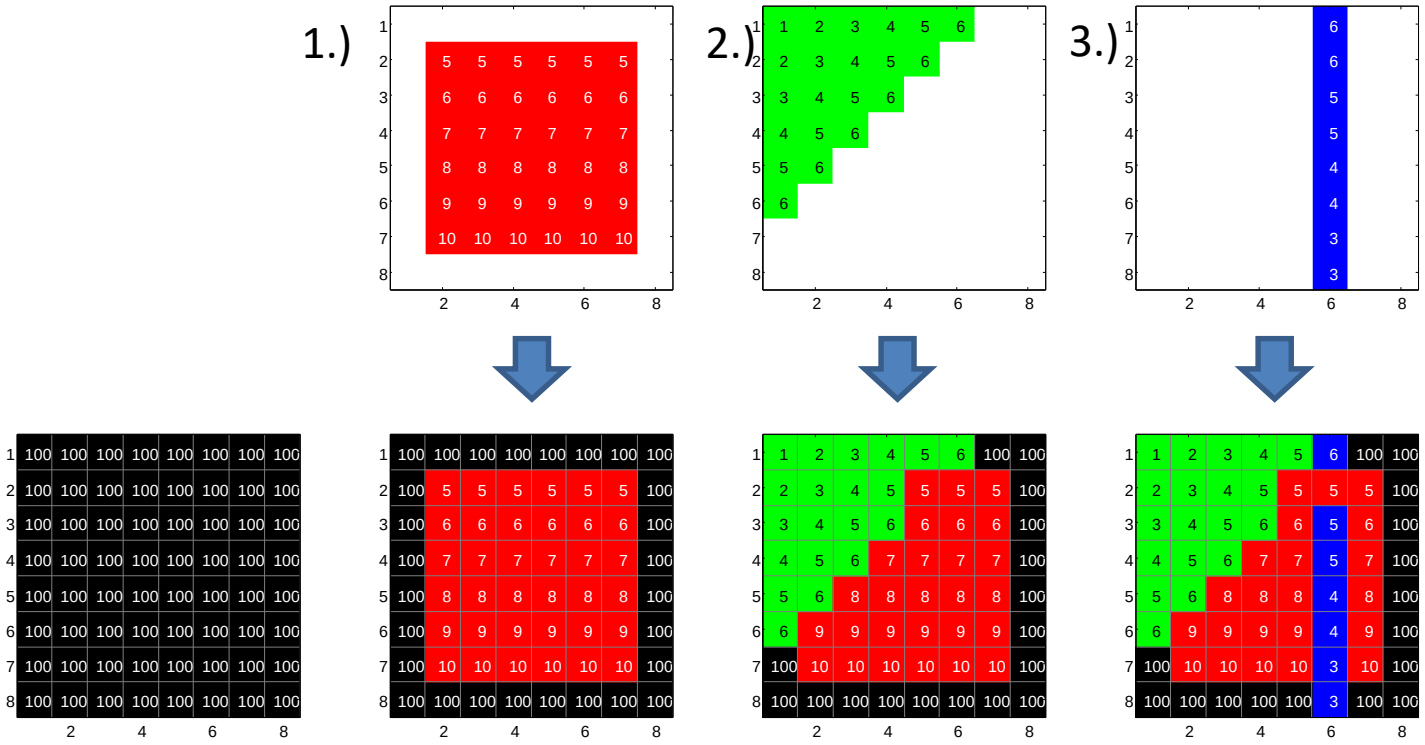


Lösung

3. Gegeben sind die folgenden Polygone mit ihren zugehörigen z-Koordinaten pro Pixel.

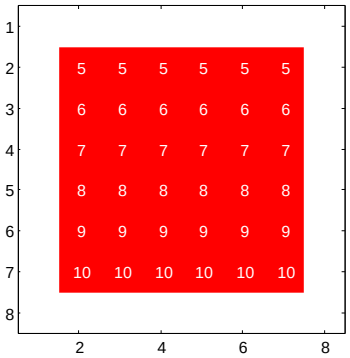
a) Zeichnen Sie diese in der definierten Reihenfolge unter Verwendung des z-Buffers mit der Operation „<=“ in das vorgegebene 8x8 Pixelfeld und geben Sie den z-Buffer an.

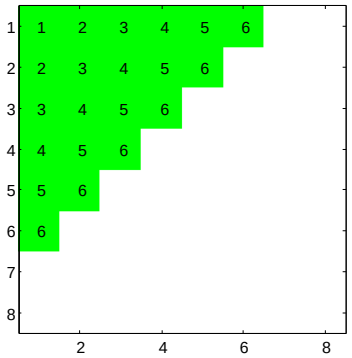
Pro Pixel wird der Reihenfolge nach der z-Wert mit dem Operator „<=“ verglichen. Ist die Bedingung erfüllt wird der Wert im z-Buffer und im Bild aktualisiert:

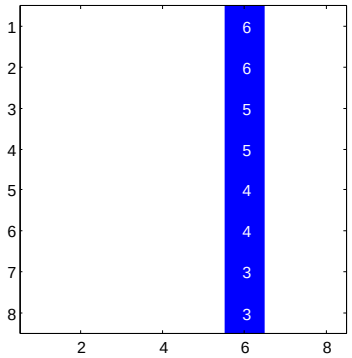


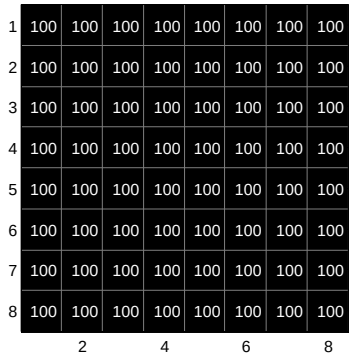
Lösung (2)

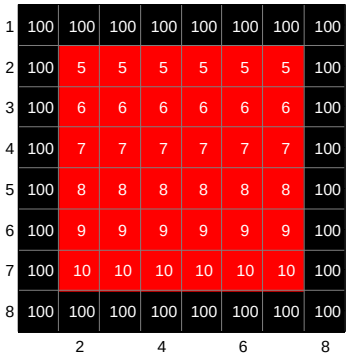
b) Wie ändert sich die Darstellung bei der Operation „<“

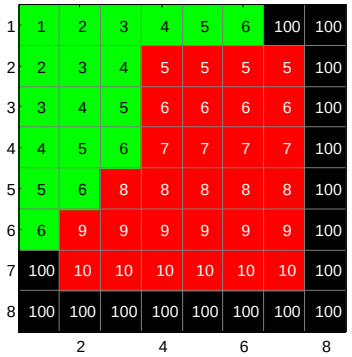


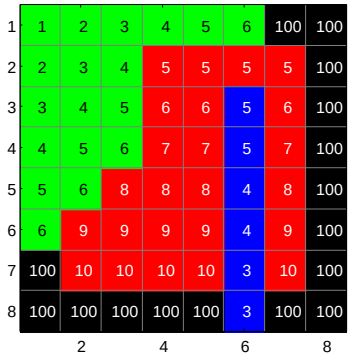






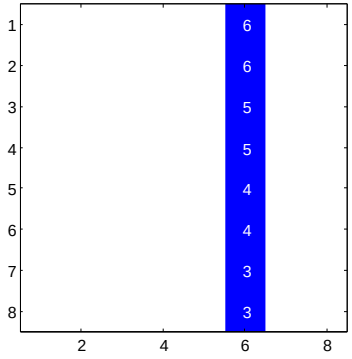


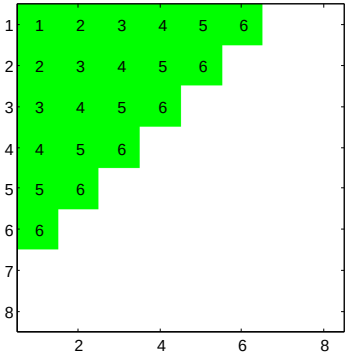


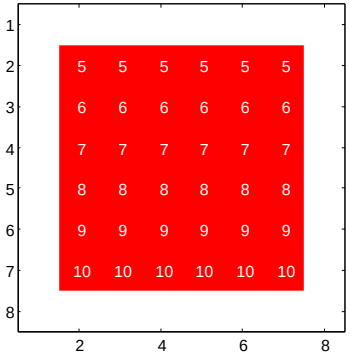


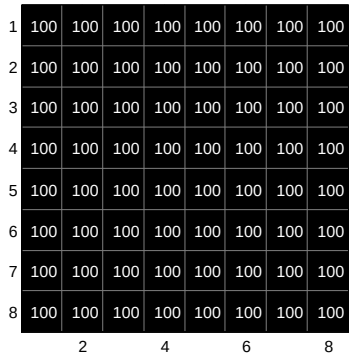
Lösung (3)

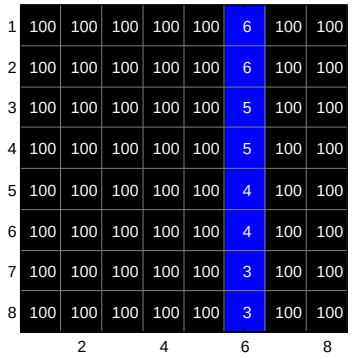
c) Wie ändert sich die Darstellung wenn die Objekte in umgekehrter Reihenfolge mit der Operation „<=“ gezeichnet werden?

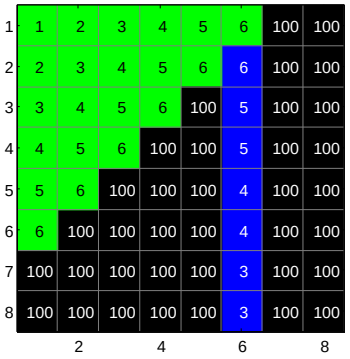


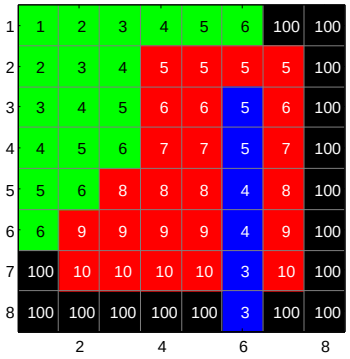






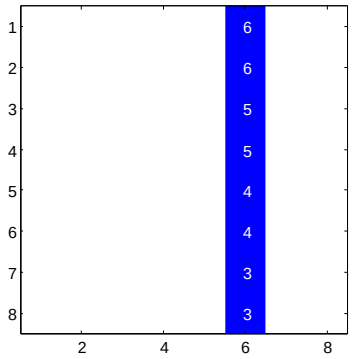


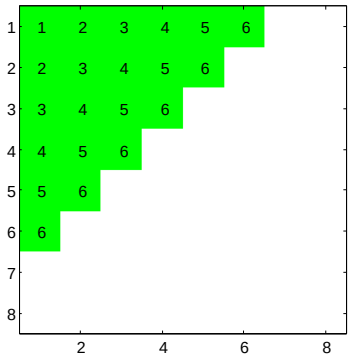


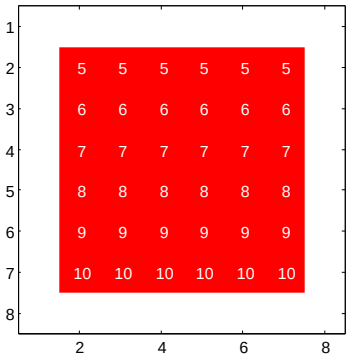


Übungsaufgaben

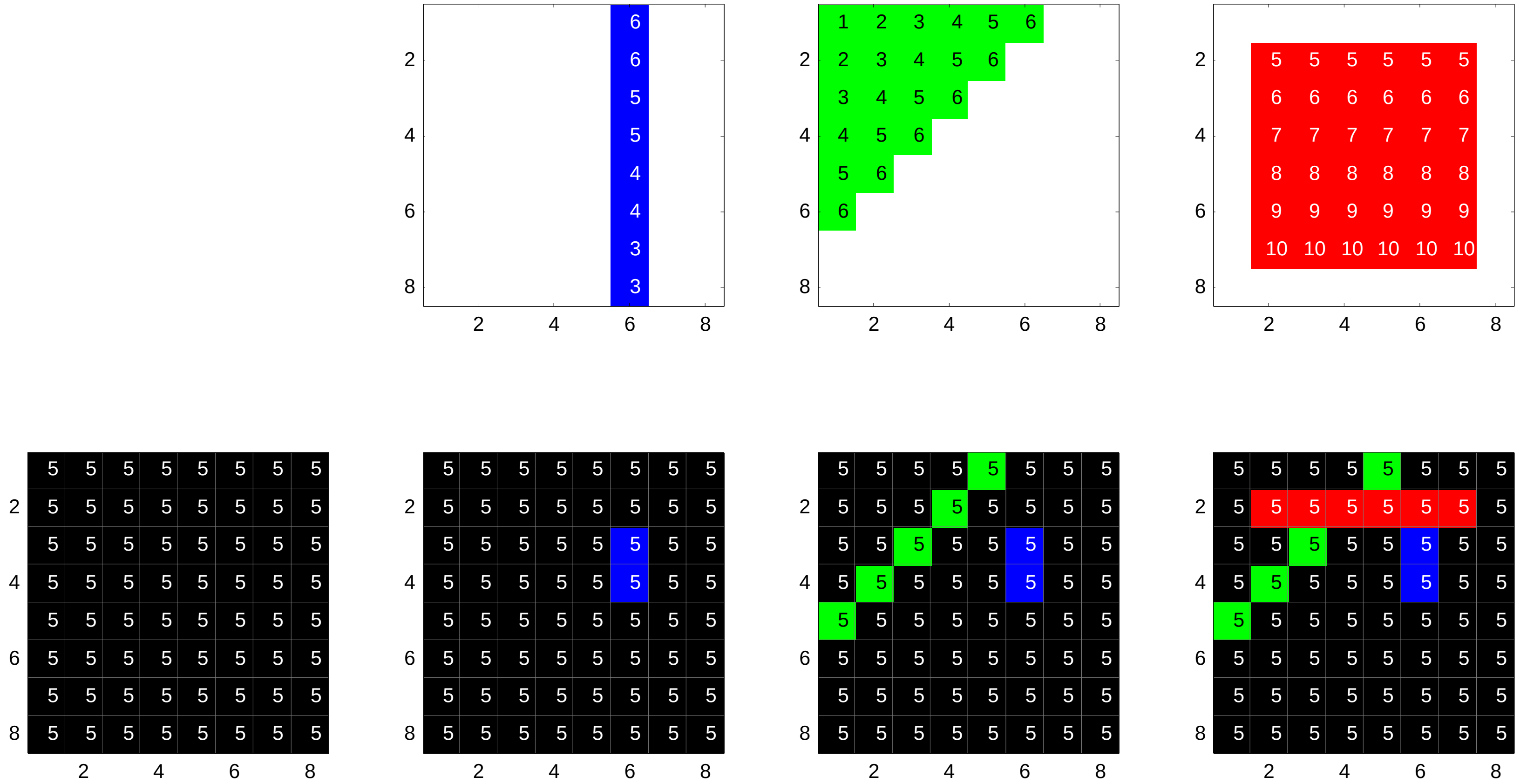
4. Gehen Sie nun von einer Initialisierung des z-Buffers mit dem z-Wert 5 aus. Als Vergleichsoperation ist ‚=‘ definiert. Welches Bild und welche z-Werte ergeben sich durch Einfügen der Pixelmuster in der gegebenen Reihenfolge?







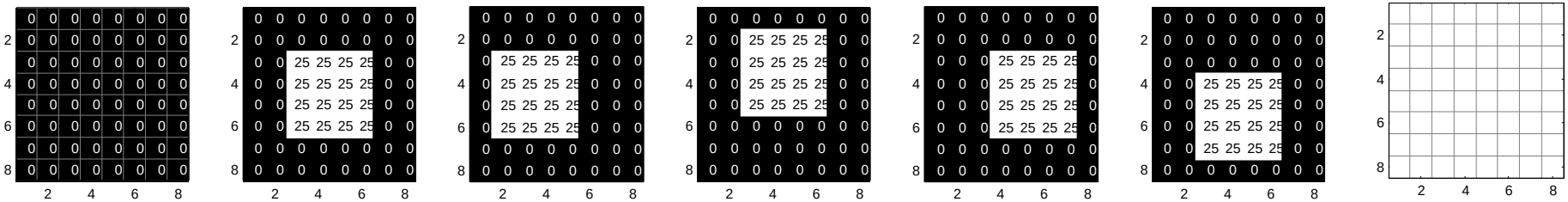
4. Gehen Sie nun von einer Initialisierung des z-Buffers mit dem z-Wert 5 aus. Als Vergleichsoperation ist ‚=' definiert. Welches Bild und welche z-Werte ergeben sich durch Einfügen der Pixelmuster in der gegebenen Reihenfolge?



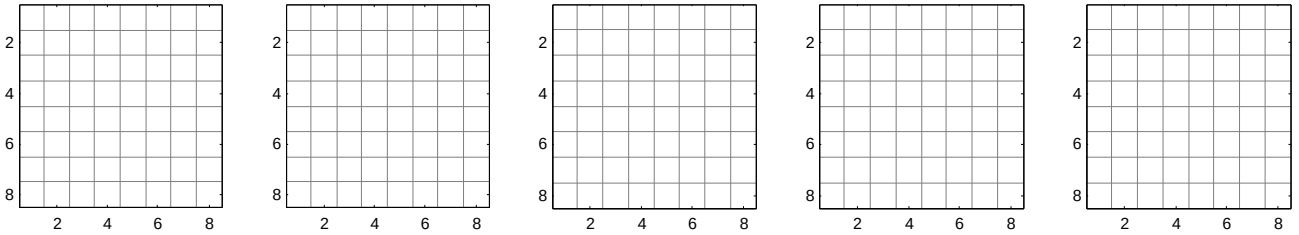
Übungsaufgaben

5. Für ein Szenen-Anti-Aliasing mit dem Accumulation-Buffer werden folgende 5 Bilder nacheinander durch Verschiebung der Kamera im Colorbuffer erzeugt. Es ist die die Accumulation-Funktion **`glAccum(GL_ACCUM, 1.0/5.0);`** definiert. Geben Sie in der unteren Zeile den Inhalt des Accumulation Buffer nach jedem Akkumulationsschritt an. Wie sieht der Colorbuffer aus wenn mit **`glAccum(GL_RETURN, 1.0);`** der Inhalt des Accumulationbuffers zurückgeschrieben wird?

Colorbuffer:



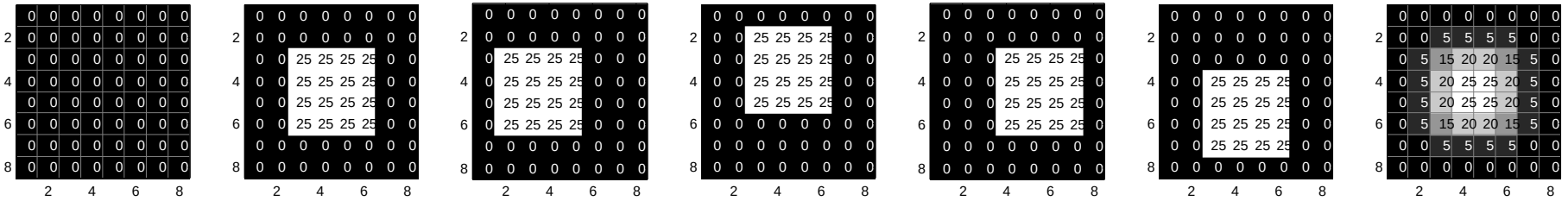
Accumulation-Buffer:



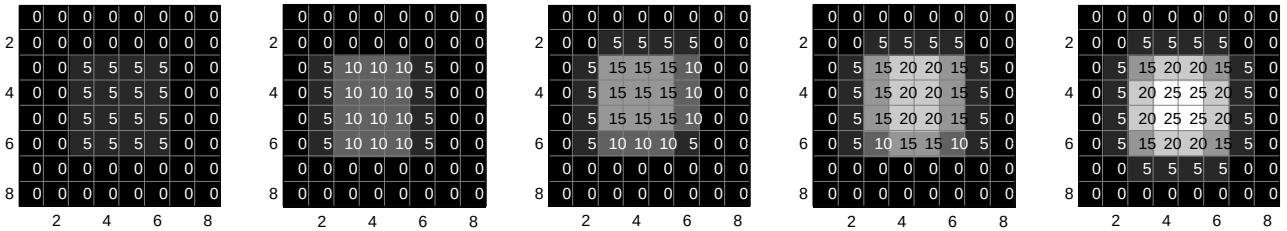
Lösung

5. Für ein Szenen-Anti-Aliasing mit dem Accumulation-Buffer werden folgende 5 Bilder nacheinander durch Verschiebung der Kamera im Colorbuffer erzeugt. Es ist die die Accumulation-Funktion `glAccum(GL_ACCUM, 1.0/5.0);` definiert. Geben Sie in der unteren Zeile den Inhalt des Accumulation Buffer nach jedem Akkumulationsschritt an. Wie sieht der Colorbuffer aus wenn mit `glAccum(GL_RETURN, 1.0);` der Inhalt des Accumulationbuffers zurückgeschrieben wird?

Colorbuffer:



Accumulation-Buffer:

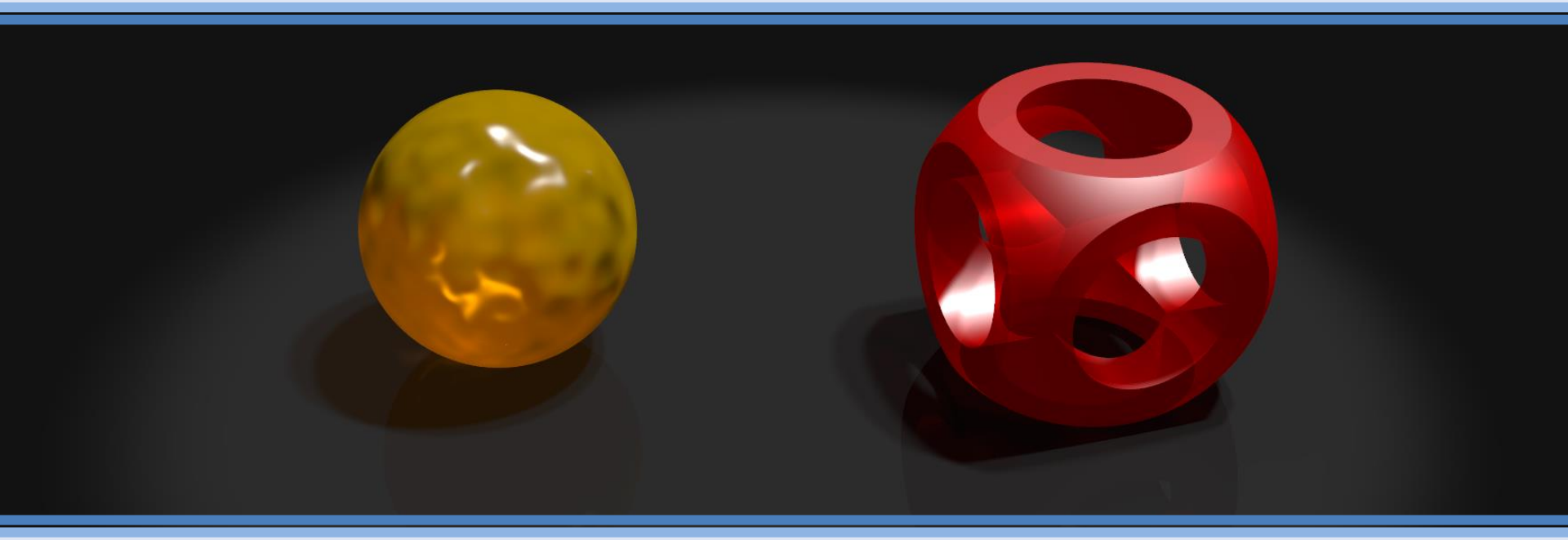


Übungsfragen Kapitel 6

- Was ist VSYNC? Wozu wird es verwendet?
- Nennen und Beschreiben Sie ein Verfahren für eine Hidden-Line-Darstellung eines Würfels. Welche Zeichen- und Verarbeitungsschritte sind nötig?
- Was ist z-Fighting? Wann kann es auftreten? Was ist der Grund dafür?
- Beschreiben Sie die Verarbeitungsschritte zur Erzeugung einer Bewegungsunschärfe mit dem Accumulation Buffer.

Computergrafik

T. Hopp



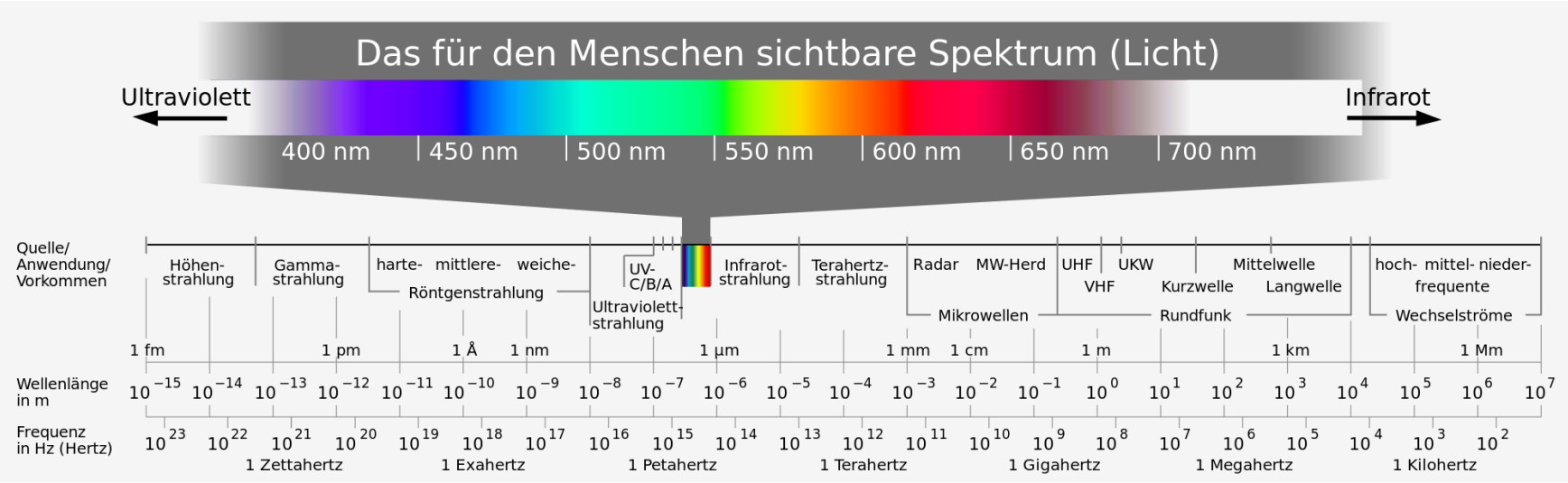
Themenübersicht

1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. **Farbe, Beleuchtung und Schattierung**
8. Texturen
9. Animationen
10. Raytracing
11. Volumenvisualisierung

7.1. LICHT UND FARBEN

Licht

- Ursache der Sehwahrnehmung des menschlichen Auges
- Lichtquellen: künstlich oder natürlich
- Menschliches Auge kann Gegenstände nur wahrnehmen wenn von Ihnen Licht ausgeht oder wenn von Ihnen Licht reflektiert wird
- Wellenförmige Ausbreitung elektromagnetischer Strahlen



https://de.wikipedia.org/wiki/Licht

Achromatisches Licht

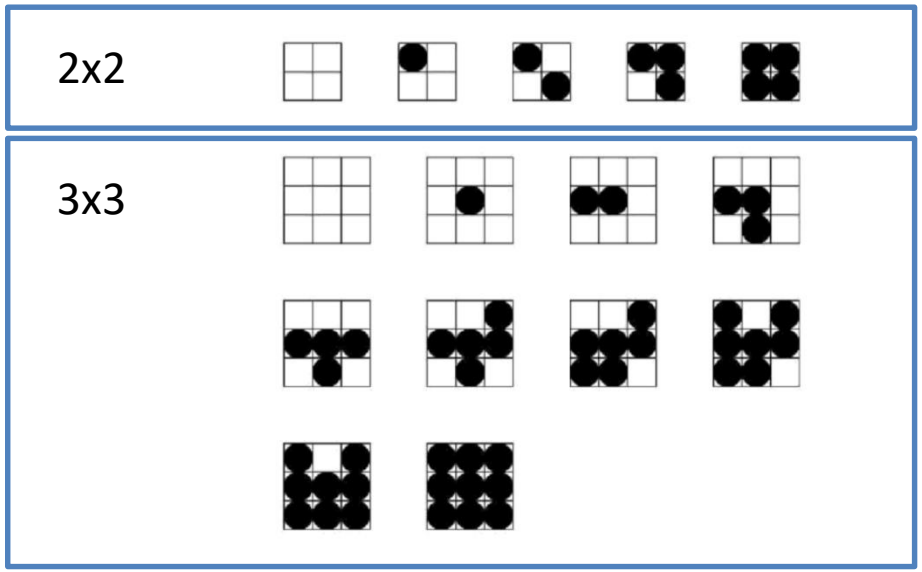
- Achromatisch = Abwesenheit von Farbe
- Betrachtung der Intensität (Luminanz)
  - Meist Angabe als Gleitkommawert zwischen 0.0 und 1.0
  - Alternativ als Integerwert, z.B. zwischen 0 und 255 (8 Bit)

- Konvention:
  - Intensität = 0.0 bzw. 0: schwarz
  - Intensität = 1.0 bzw. 255: weiß

- Darstellbare Intensitätslevel sollten logarithmisch (dem Auge angepasst) werden

Halbtonverfahren

- Darstellung von Intensitätsstufen auf Bi-level-Displays
- Erzeugung durch räumliche Integration:

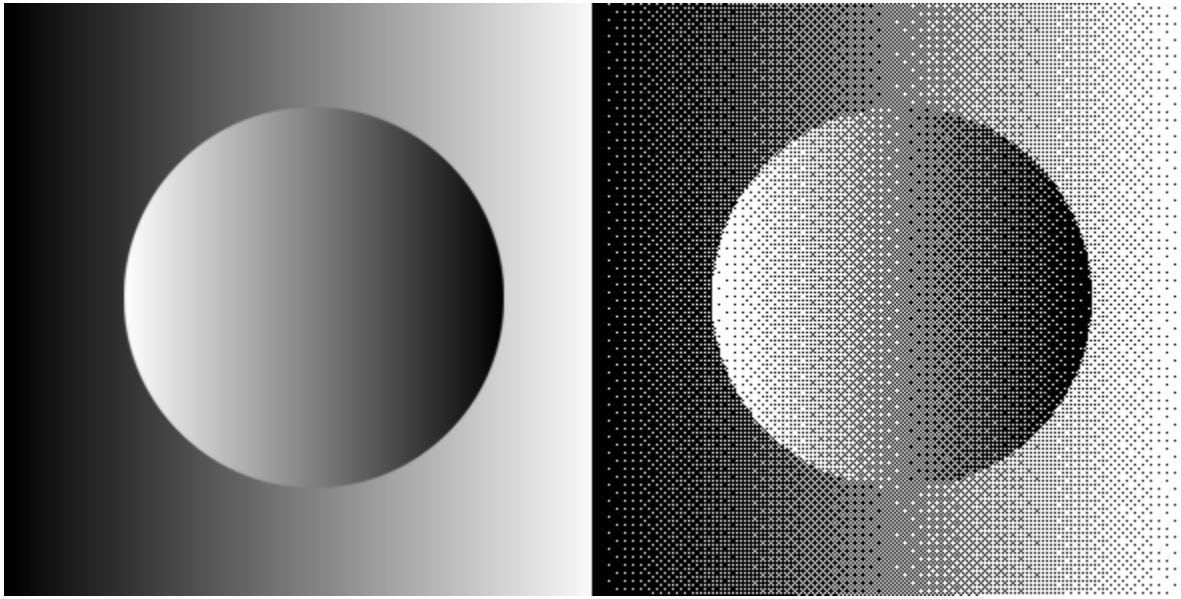


- Durch $n \times n$ Pixel $n^2 + 1$ Intensitätsstufen darstellbar.
- „Dither-Matrizen“

F. Klawonn: Grundkurs Computergrafik mit Java, 3. Auflage, Vieweg+Teubner, 2010

Halbtonverfahren

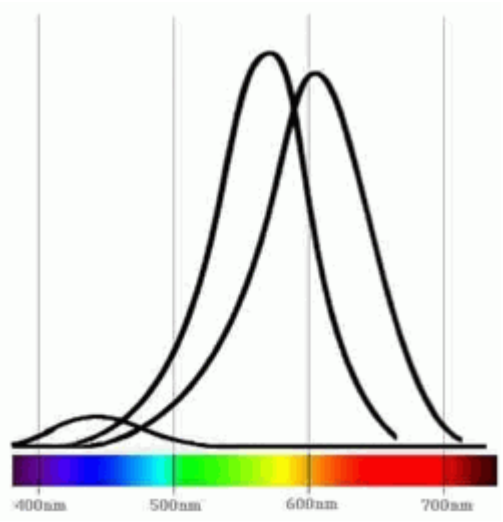
- Beispiel: Farbverlauf mit 8x8 Dither-Matrizen

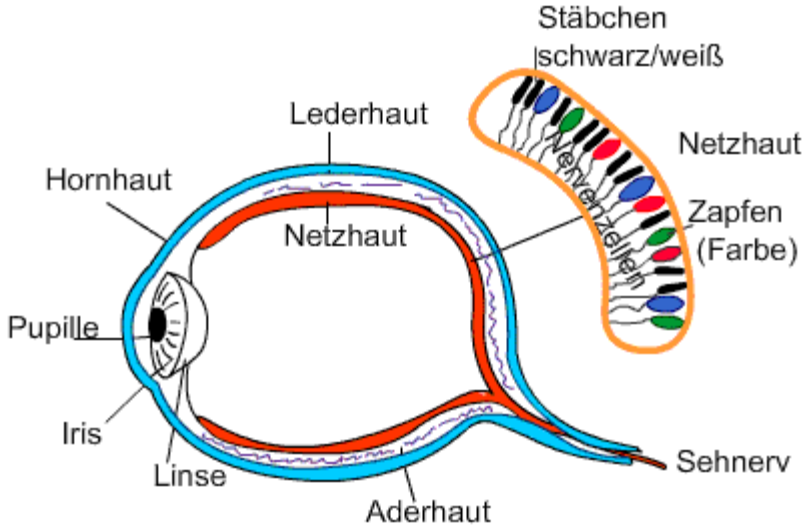


https://de.wikipedia.org/wiki/Dithering_(Bildbearbeitung)

Chromatisches Licht und Farbmodelle

- Chromatisches Licht = „farbiges Licht“
- Das menschliche Auge besitzt 3 Typen von Zapfen, die bei unterschiedlichen Wellenlängen ihre maximale Sensitivität haben



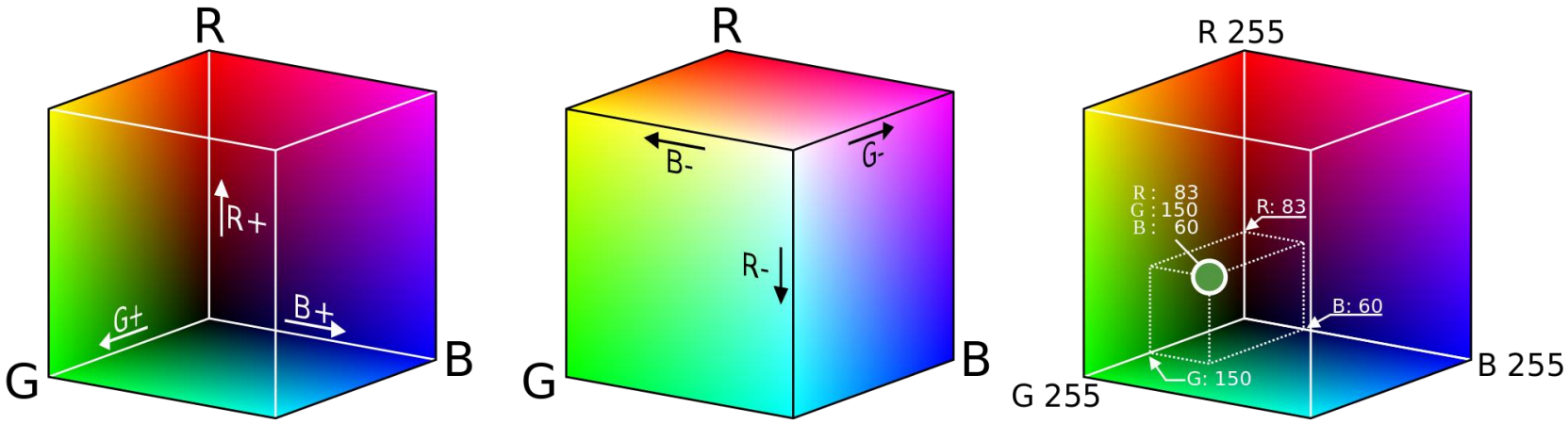


- In der CG Verwendung von Farbmodellen: Eindeutige Zuordnung von Zahlenwerten zu jeder darstellbaren Farbe
  - Unterschiedliche Modelle für unterschiedliche Anwendungsgebiete

http://www.joggysite.de/Seiten/Fotoschule/4_2Farbe_sehen.html
http://www.kidsweb.de/gesundheits_spezial/das_auge.html

Farbmodelle: RGB

- Nachbildung der Farbwahrnehmungen durch additives Mischen der drei Grundfarben Rot (R), Grün (G) und Blau (B).
- Meist in der Computergrafik verwendet

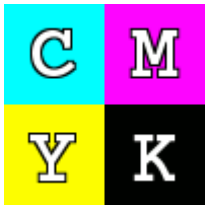


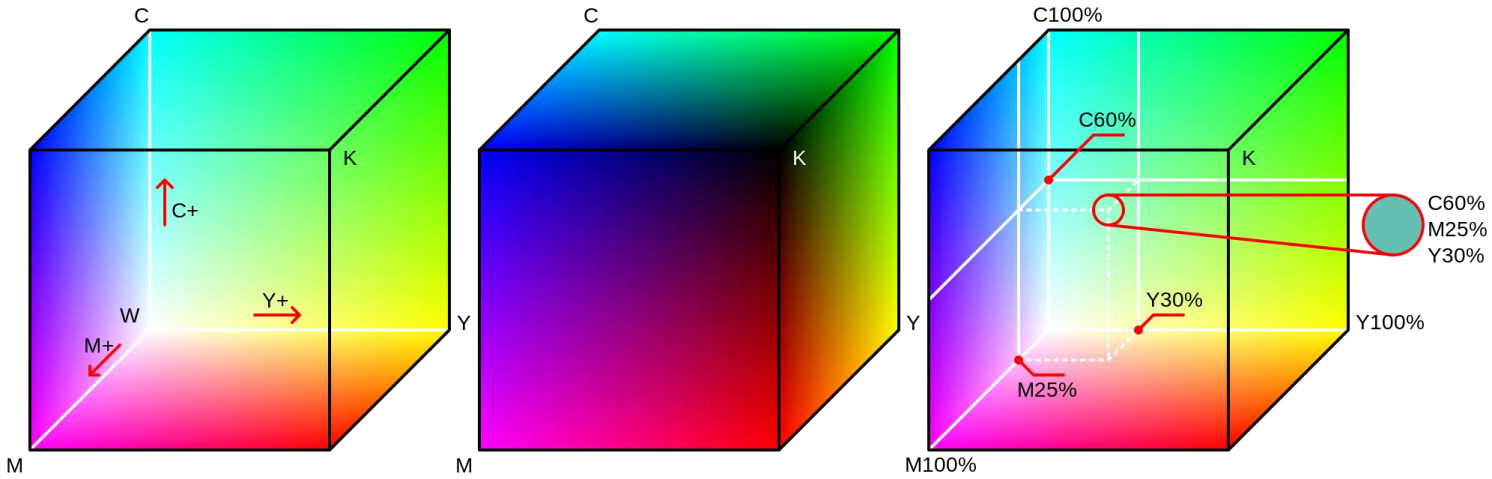
- Wertebereich: typischerweise Gleitkommazahlen zwischen 0.0 und 1.0 oder Integerzahlen zwischen 0 und 255 (= 3 x 8 Bit = 24 Bit Farbtiefe)
- Erweiterung zu RGBA-Modell: Hinzufügen eines Alphakanals für Transparenz

https://de.wikipedia.org/wiki/RGB-Farbraum

Farbmodelle: CMY(K)

- Ursprung in der Drucktechnik: subtraktives Farbmodell.
- Stellt Farben durch Mischen von Cyan (C), Magenta (M) und Gelb (Y) dar.
- Der Schwarzanteil wird im K-Kanal codiert.





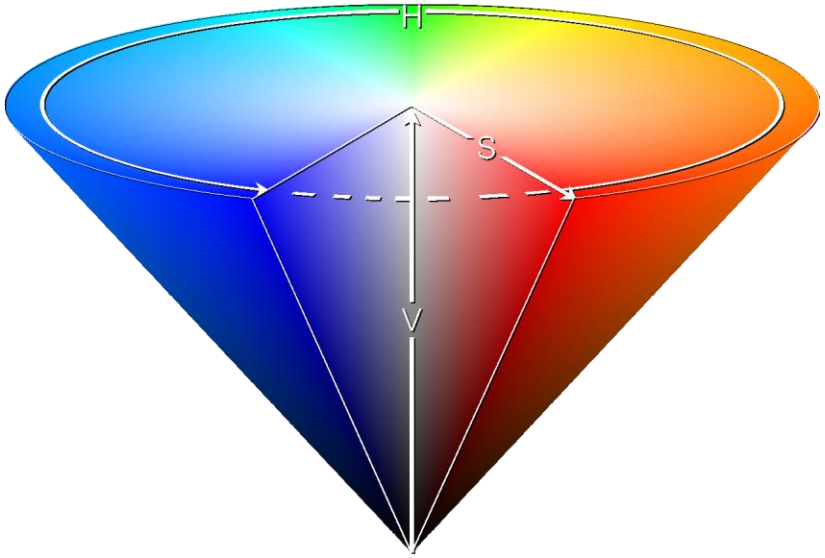
- Wertebereich: typischerweise Angabe in % zwischen 0 und 100
- Dualität von RGB und CMYK: Umrechnung erfolgt durch

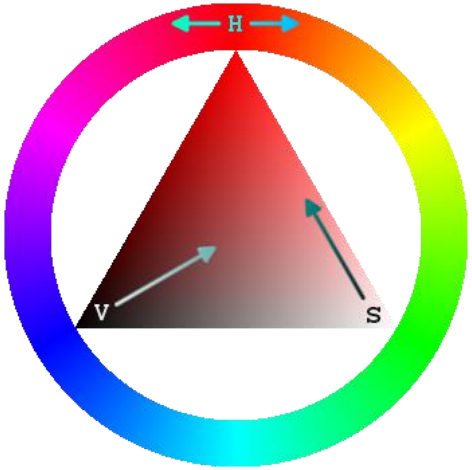
$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \text{bzw.} \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

https://de.wikipedia.org/wiki/CMYK-Farbmodell

Farbmodelle: HSV

- Ähnelt der menschlichen Farbwahrnehmung: Definition einer Farbe durch
  - Farbton H: Farbwinkel auf einem Farbkreis: $0° =$ rot, $120° =$ grün, $240° =$ blau
  - Sättigung S: Angabe in %: $0\% =$ grau bis $100\% =$ gesättigte reine Farbe
  - Hellwert V: Angabe in %: $0\% =$ keine Helligkeit, $100\% =$ volle Helligkeit





https://de.wikipedia.org/wiki/HSV-Farbraum

Farbmodelle: HSV

- Umrechnung von HSV zu RGB:
  - Vorbedingung: $H \in [0°, 360°]$, S und $V \in [0,1]$.
  - $C = V \cdot S$
  - $X = C \cdot \left(1 - \left|\frac{H}{60°} \; mod \; 2 \; - \; 1\right|\right)$
  - $m = V - C$

$$(R', G', B') = \begin{cases} (C, X, 0) & , 0° \leq H < 60° \\ (X, C, 0) & , 60° \leq H < 120° \\ (0, C, X) & , 120° \leq H < 180° \\ (0, X, C) & , 180° \leq H < 240° \\ (X, 0, C) & , 240° \leq H < 300° \\ (C, 0, X) & , 300° \leq H < 360° \end{cases}$$

  - $(R, G, B) = ((R' + m) \cdot 255, \; (G' + m) \cdot 255, \; (B' + m) \cdot 255)$

Farbmodelle: HSV

- Umrechnung von HSV zu RGB: Beispiel
  - $H = 60°,\ S = 1,\ V = 0{,}5$
  - $C = V \cdot S\ =\ 0{,}5$
  - $X = C \cdot \left(1\ -\ \left|\frac{H}{60°}\ mod\ 2\ -\ 1\right|\right) = 0{,}5 \cdot \left(1\ -\ \left|\frac{60°}{60°}\ mod\ 2\ -\ 1\right|\right) = 0{,}5$
  - $m = V - C = 0{,}5 - 0{,}5 = 0$

$$(R', G', B') = \begin{cases} (C, X, 0) & , 0° \leq H < 60° \\ (X, C, 0) & , 60° \leq H < 120° \\ (0, C, X) & , 120° \leq H < 180° \\ (0, X, C) & , 180° \leq H < 240° \\ (X, 0, C) & , 240° \leq H < 300° \\ (C, 0, X) & , 300° \leq H < 360° \end{cases}$$

$R' = 0{,}5$
$G' = 0{,}5$
$B' = 0$

  - $(R, G, B) = ((R' + m) \cdot 255,\ (G' + m) \cdot 255,\ (B' + m) \cdot 255)$
    $= ((0{,}5 + 0) \cdot 255, (0{,}5 + 0) \cdot 255, (0 + 0) \cdot 255)$
    $= (128, 128, 0)$

Wechselwirkung von Licht und Materie

- Physikalische Grundlagen beruhen auf Welle-Teilchen-Dualismus
  - Modellierung der Ausbreitung von Licht als elektromagn. Welle: Wellentheorie
  - Modellierung der Wechselwirkung von Licht und Materie: Teilchentheorie

- Viele Effekt nur im mikroskopischen Bereich relevant.

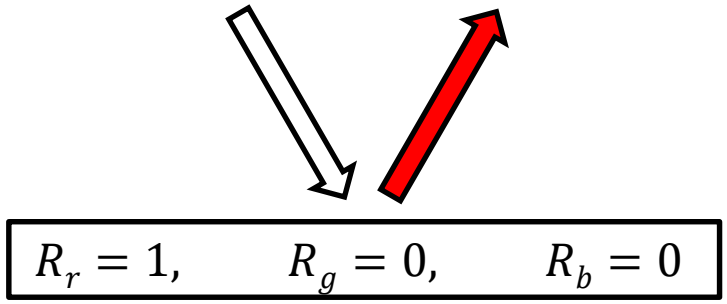
- Vereinfachungen in der Computergrafik auf makroskopische Effekte
  - Lichtausbreitung in homogenen Medien wird als gerader Lichtstrahl modelliert
  - Kontinuierliches Spektrum wird an drei Stellen abgetastet: rot, grün, blau
  - Interferenz, Beugung, Polarisation werden vernachlässigt
  - Wechselwirkung von Licht und Materie wird stark vereinfacht.

➔ Geometrische Optik

Wechselwirkung von Licht und Materie

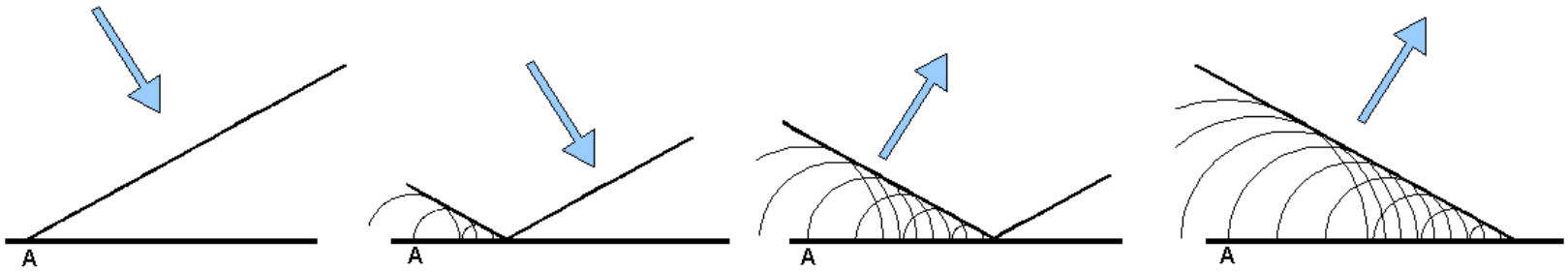
- Photonen regen Elektronen und Protonen zur Schwingung an
  - Wenn Anregungsfrequenz = Resonanzfrequenz des Materials → Absorption
  - Sonst: Streuung/Reflexion

- Resonanzfrequenz (=Materialkonstante) bestimmt Absorptions- ($A(\lambda)$) und Reflexionskoeffizient ($R(\lambda)$).
- Energieerhaltung: $A(\lambda) + R(\lambda) = 1$
- Computergrafik: Abtastung an drei Wellenlängen reduziert Koeffizienten auf drei Komponenten: z.B. (R_r, R_g, R_b)
- Oberfläche erscheint bei Bestrahlung mit weißem Licht in der Farbe entsprechend der Größe der drei Reflexionskoeffizienten.



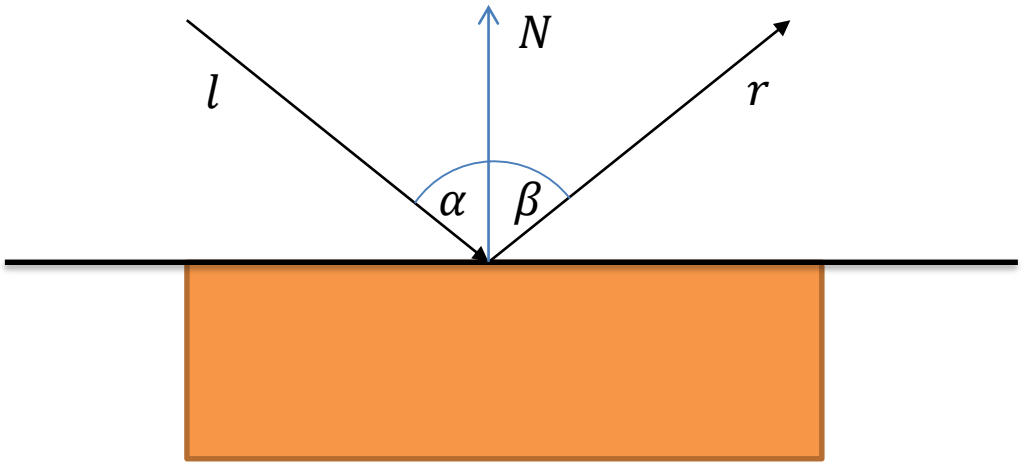
Streuung / Reflexion

- Ausbreitung des Lichts als Wellenfront
- Streuung folgt dem Huygen‘schen Prinzip
  - In jedem Punkt einer Wellenfront sitzt ein Streuzentrum, von dem aus elementare Kugelwellen ausgehen.
  - Überlagerung aller Kugelwellen ergibt eine neue Wellenfront



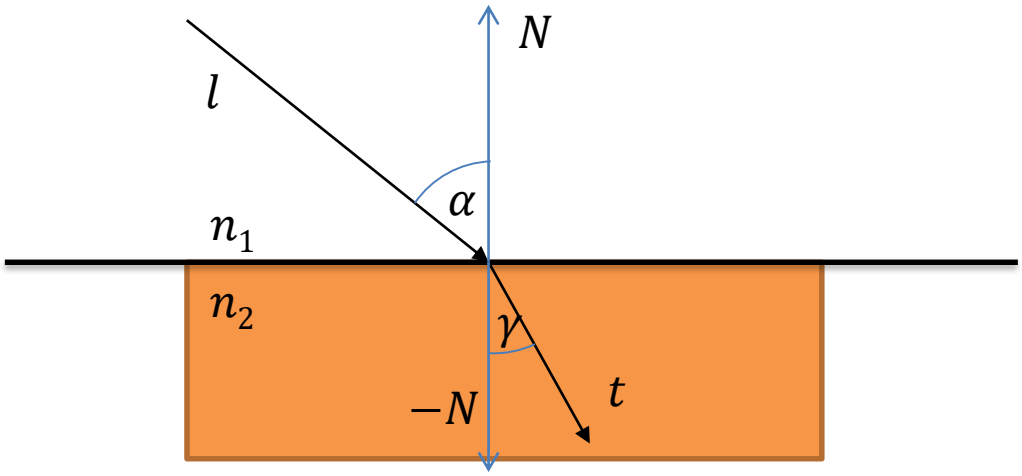
https://de.wikipedia.org/wiki/Huygenssches_Prinzip

Reflexionsgesetz



- Reflexion erfolgt an Oberflächen bzw. Mediengrenzen
- Einfallswinkel α = Reflexionswinkel β
- Hier: Ideale Reflexion → komplettes einfallendes Licht wird reflektiert

Brechungsgesetz

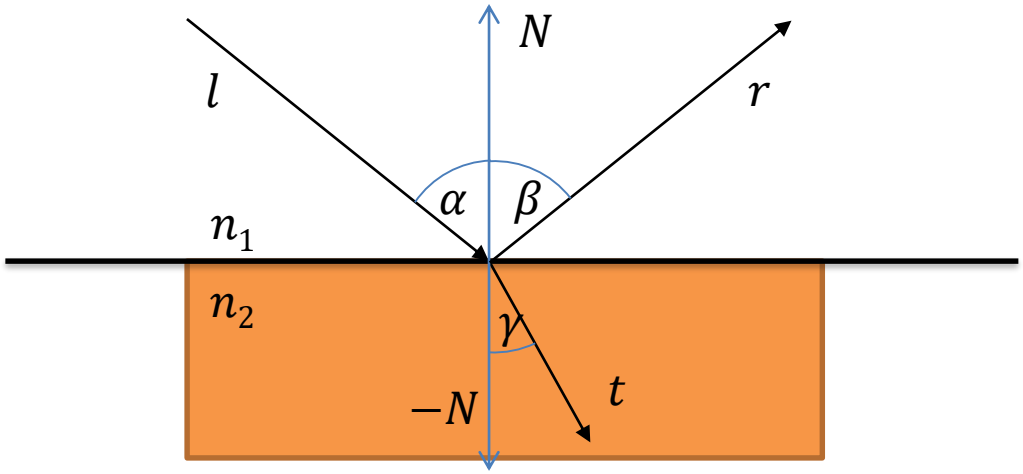


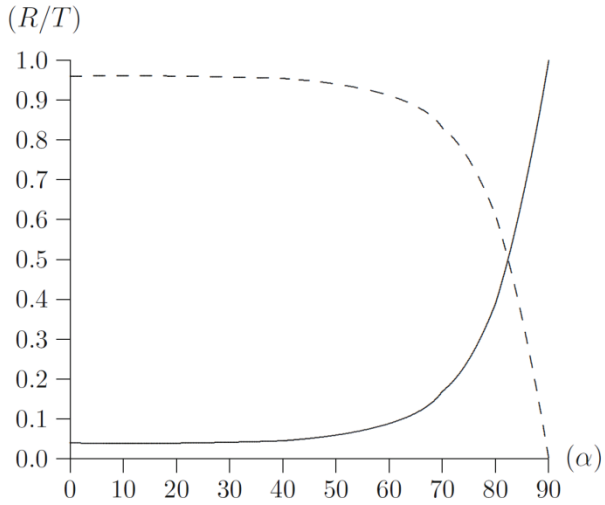
- Verhältnis zwischen dem Sinus des Einfallswinkels α und dem Sinus des Brechungswinkels γ entspricht dem Verhältnis der Brechungsindizes n_1 und n_2 der beiden Medien:

$$\frac{\sin \alpha}{\sin \gamma} = \frac{n_2}{n_1}$$

- Dispersion: Brechungsindex ist abhängig von Wellenlänge des Lichtes
  - D.h. dreifacher Berechnungsaufwand für die Brechungsindizes bei Rot, Grün, Blau

Fresnel-Effekt



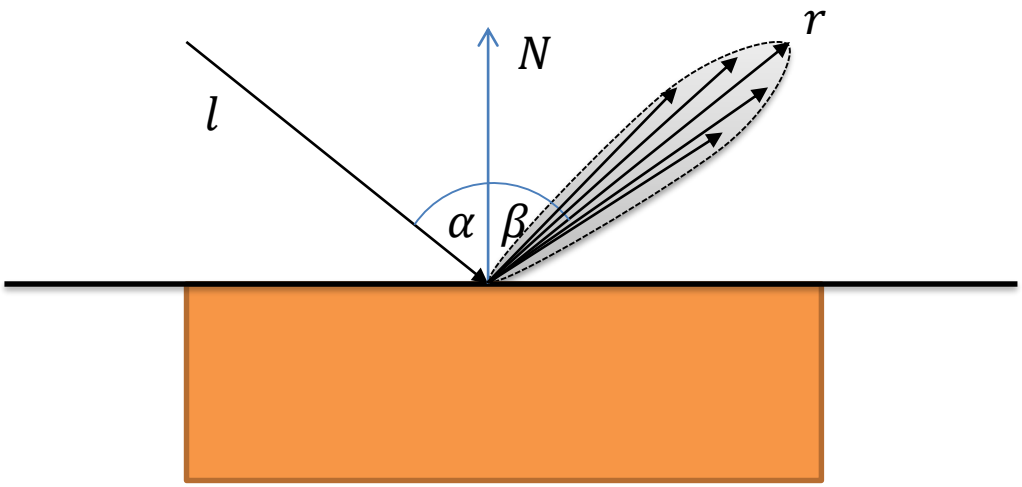


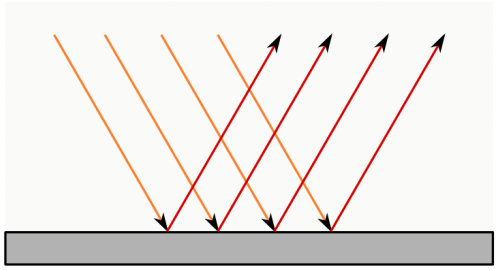
- Bei der Brechung wird auch an ideal glatten Grenzflächen ein Anteil des Lichts reflektiert
- Je größer (flacher) der Einfallswinkel
  - desto größer wird der Anteil des reflektierten Lichts
  - desto geringer wird der Anteil des gebrochenen Lichts
- Vereinfachte Gleichung: $R = \frac{1}{2}\left(\frac{\sin^2(\gamma-\alpha)}{\sin^2(\gamma+\alpha)} + \frac{\tan^2(\gamma-\alpha)}{\tan^2(\gamma+\alpha)}\right)$

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

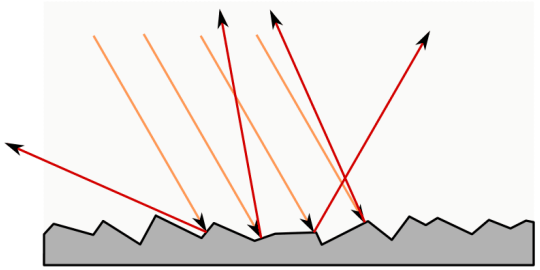
Diffuse Reflexion

- Bisher: ideale Reflexion
- Realität: Rauigkeit einer Oberfläche spaltet Lichtstrahl auf (→ Leuchtkörper)





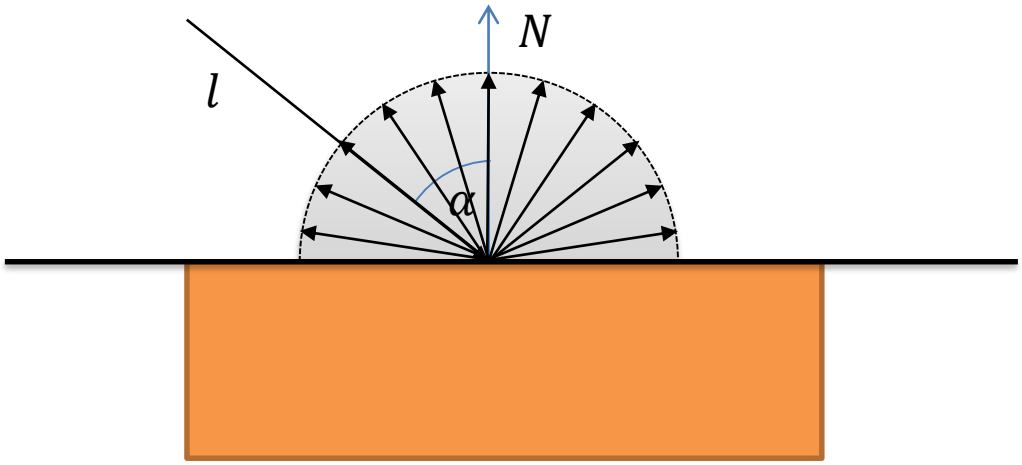
Glatter Spiegel
Direkte Reflektion



Rauher Spiegel
Diffuse Reflektion

http://www.grund-wissen.de/physik/optik/lichtreflexion.html

Ideal diffuse Reflexion



- Lambert‘sches Streumodell: gleichmäßige Streuung des einfallenden Lichtes in alle Richtungen (ideal diffus streuend)
- Allgemein: beliebig geformte Leuchtkörper, Anisotropie etc.

Bidirectional Reflectance Distribution Function

- Allgemein: Anteil der Lichtintensität der in eine Raumrichtung reflektiert wird abhängig von 5 Variablen:
  - Elevations- und Azimuthwinkel des einfallenden Strahls α und θ
  - Elevations- und Azimuthwinkel des reflektierten Strahls β und φ
  - Wellenlänge des Lichtes λ.

- Bei bekannten Materialeigenschaften kann Intensität der in alle Raumrichtung reflektierten Strahlen I_r berechnet werden:

$$I_r(\alpha, \theta, \lambda) = R(\alpha, \beta, \theta, \varphi, \lambda) \cdot I_e\ (\alpha, \theta, \lambda) \cdot \cos(\alpha)$$

- Geht von der Idealvorstellung punktförmiger Quellen aus.
- Bei Lichtquellen mit endlicher Ausdehnung treffen Lichtstrahlen aus verschiedenen Raumrichtungen auf die Oberfläche: es muss über die Fläche der Lichtquelle integriert werden → aufwendige Integralformel!
- Analog lässt sich die Bidirectional Transmssion Distribution Function für Brechung berechnen.

7.2. BELEUCHTUNGSMODELLE

Lokale und globale Beleuchtung

- Trotz aller Vereinfachungen bleibt Beleuchtungsrechnung komplex:
  - Z.B. reflektiertes Licht von Oberflächen entspr. erneutem Aussenden von Licht, ausgehend von allen Punkten auf dieser Oberfläche

- Lokale Beleuchtung:
  - Nur punktförmige Lichtquellen
  - Berechnung der (einfachen) Reflexion anhand z.B. BRDF
  - Interaktive Computergrafik

heute

- Globale Beleuchtung:
  - Berechnet auch Licht das vom Punkt einer Oberfläche ausgestrahlt wird (indirekte Beleuchtung)
  - Integralformel! → sehr aufwändig.
  - Raytracing, Radiosity.

Kapitel 10

Phong Reflexionsmodell

- Einfaches Reflexions- bzw. Beleuchtungsmodell für die interaktive Computergrafik
  - Lokales Beleuchtungsmodell
  - Streulicht von Oberflächen: Approximation durch „*ambient*“
  - Kein Schattenwurf
  - Abtastung des kontinuierlichen Spektrums an drei Stellen: Rot, Grün, Blau
  - Atmosphärische Effekte (z.B. Verblassung) werden nicht berücksichtigt
  - Transparenz wird vernachlässigt
  - Brechungseffekte, Dispersion, Fresnel-Effekte werden vernachlässigt

- Farbe eines (elementaren) Objektes wird definiert über
  - Zugehörigkeit zu einem Objekt mit definierten Materialeigenschaften
  - Farbe und Position der Lichtquelle

Phong Reflexionsmodell

- Realisierung über drei Beleuchtungskomponenten
  - *Ambiente*: ungerichtetes Streulicht, Ersatz für globale Beleuchtung
  - *Diffuse*: gerichtetes Licht - Lambert‘sches Beleuchtungsmodell – ideal diffuse Oberfläche
  - *Spekulare*: gerichtetes Licht - spiegelnde Oberfläche

- Summe der 3 Beleuchtungskomponenten ergibt gesamte Intensität des reflektierten Lichtes an einem Punkt der Oberfläche:

$$I = I_{ambiente} + I_{diffus} + I_{spekular}$$

- Teilweise vierte Komponente: *emissive*
  - Selbstleuchtende Oberfläche, unabhängig von Umgebungslicht
  - Phong: Emissive Oberflächen beleuchten andere Oberflächen nicht.

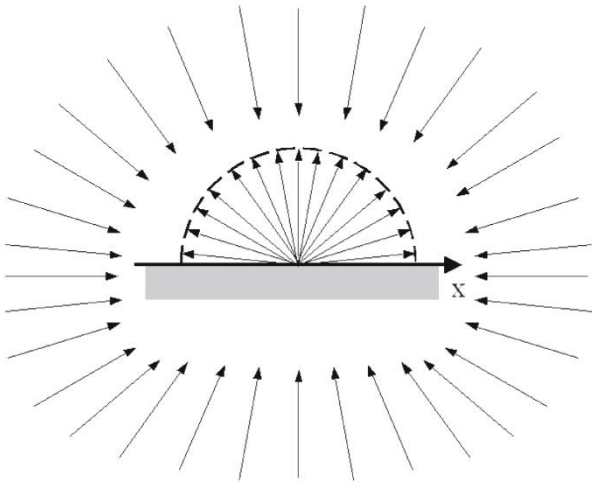
Umgebungslicht (*ambiente*)

- Approximation von Licht das nicht direkt auf die Oberfläche fällt, sondern zuerst von anderen Oberflächen gestreut wird.
  - Unabhängig vom Einfallswinkel des Lichtes

$$I_{ambiente} = I_a \cdot k_a$$

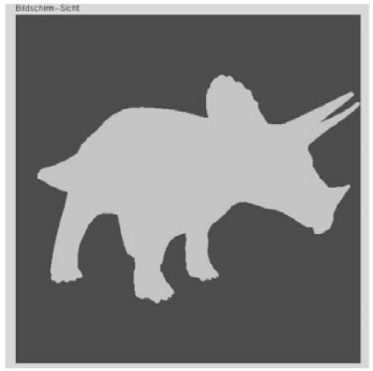
k_a: Materialkonstante

I_a: Intensität des Umgebungslichtes



Ungerichtetes Licht aus allen Richtungen

(a)



(b)

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Diffuses Streulicht (*diffus*)

- Ideal diffuse Reflexion an der Oberfläche:
  - Reflektierte Lichtstärke unabhängig vom Standpunkt des Betrachters
  - Reflektierte Lichtstärke abhängig vom Einfallswinkel α.

$$I_{diffus} = I_{in} \cdot k_d \cdot \cos(\alpha)$$
$$I_{diffus} = I_{in} \cdot k_d \cdot (l \cdot N)$$

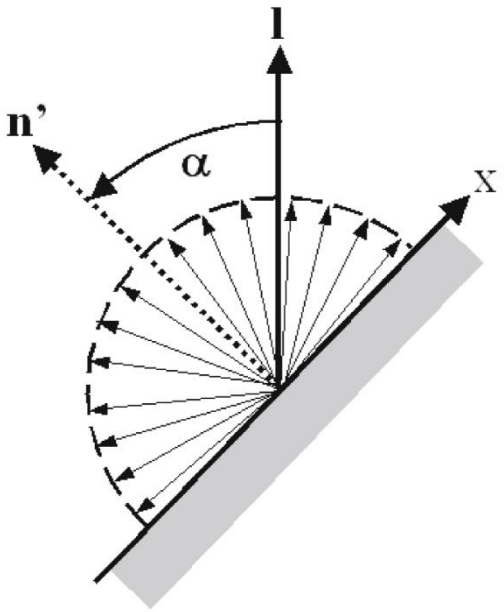
Skalarprodukt: $\cos(\alpha) = \frac{l \cdot N}{|l| \cdot |N|}$

Empirisch bestimmter Reflexionsfaktor (Materialkonstante)

Intensität des einfallenden Lichtstrahls der Punktlichtquelle

- Nur Oberflächen die der Lichtquelle zugewandt sind können beleuchtet werden: $\cos(\alpha)$ wird über Maximumfunktion ausgedrückt: $\max(l \cdot N, 0)$
- Bei n Lichtquellen ist die Summe über alle einzelnen Intensitätsquellen und die Richtungen zu bilden: $I_{diffus} = k_d \sum_n I_{in,n}(L_n \cdot N)$

Diffuses Streulicht (*diffus*)

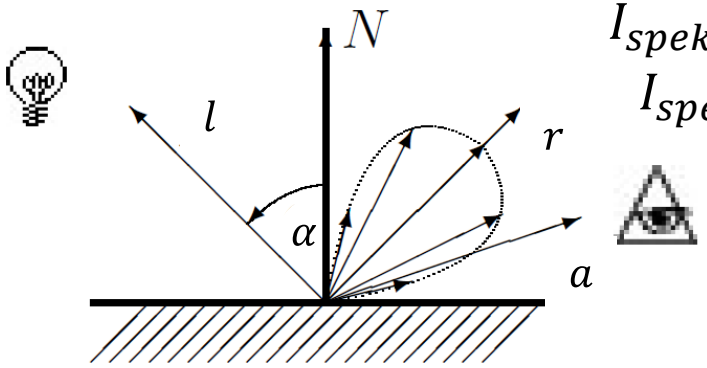


A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Spiegelnde Reflexion (*spekular*)

- Größte Lichtintensität in der idealen Reflexionsrichtung r.

- Je größer der Winkel ϕ zwischen r und dem Augpunktvektor a wird, desto kleiner wird die gespiegelte Lichtintensität in Richtung Augpunkt



$$I_{spekular} = I_{in} \cdot k_s \cdot (\cos(\phi))^s$$

$$I_{spekular} = I_{in} \cdot k_s \cdot (r \cdot a)^s$$

Spiegelungsexponent

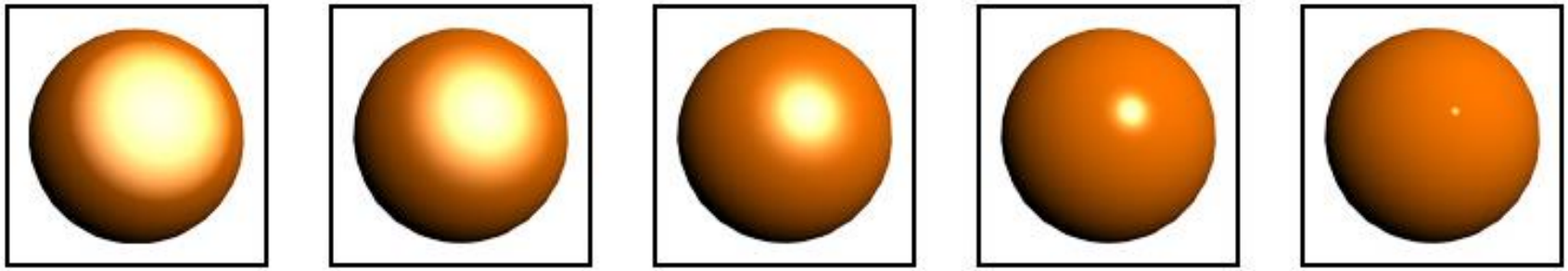
Empirisch bestimmter Reflexionsfaktor (Materialkonstante)

Intensität des einfallenden Lichtstrahls der Punktlichtquelle

- Große S: sehr glatte Oberfläche (punktförmiges Glanzlicht)
- Kleine S: matte Oberfläche (ausgedehntes Glanzlicht)

Spiegelnde Reflexion (*spekular*)

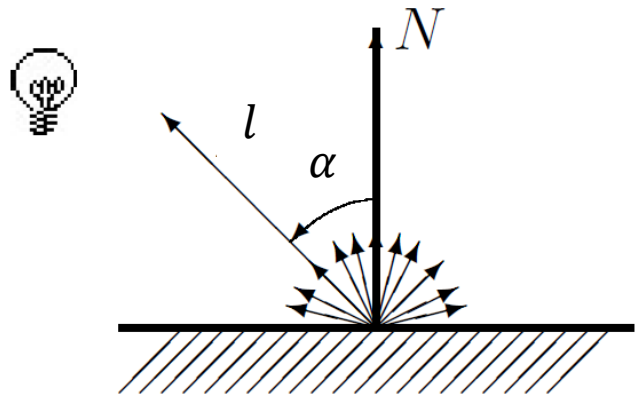
- Auswirkung des Spiegelungsexponenten

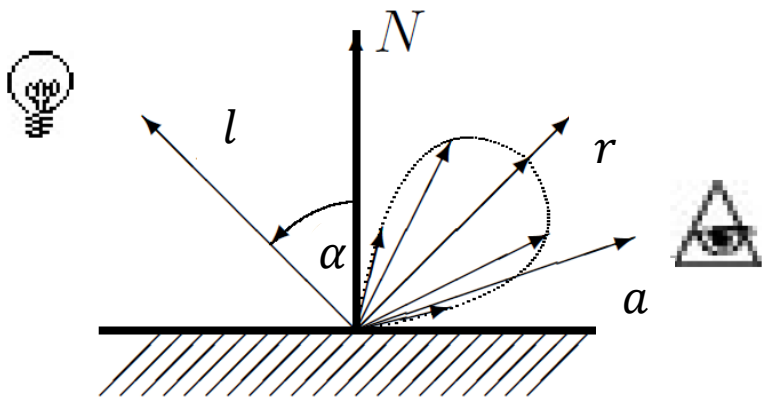


http://www.dma.ufg.ac.at/app/link/Grundlagen%3A3D-Grafik/module/9746?step=2

Vergleich diffuse/spiegelnde Reflexion

Diffuse Reflexion	Spiegelnde Reflexion
Unabhängig von Betrachterposition	**Abhängig** von Betrachterposition
Farbe des reflektierten Lichtes = **Farbe des Objektes + Farbe der Lichtquelle**	Farbe des reflektierten Lichtes oft nur **Farbe der Lichtquelle**





S. Krömker: Computergrafik I, Universität Heidelberg

Phong Reflexionsmodell

- Aus der Summe der Beleuchtungskomponenten ergibt sich:

$$I = I_a k_a + I_{in} k_d \, (l \cdot N) + I_{in} k_s (ra)^S$$
$$= I_a k_a + I_{in} (k_d \, (l \cdot N) + k_s (ra)^S)$$

- Bei mehreren Lichtquellen wird über alle Intensitäten und Positionen der verschiedenen Quellen aufsummiert:

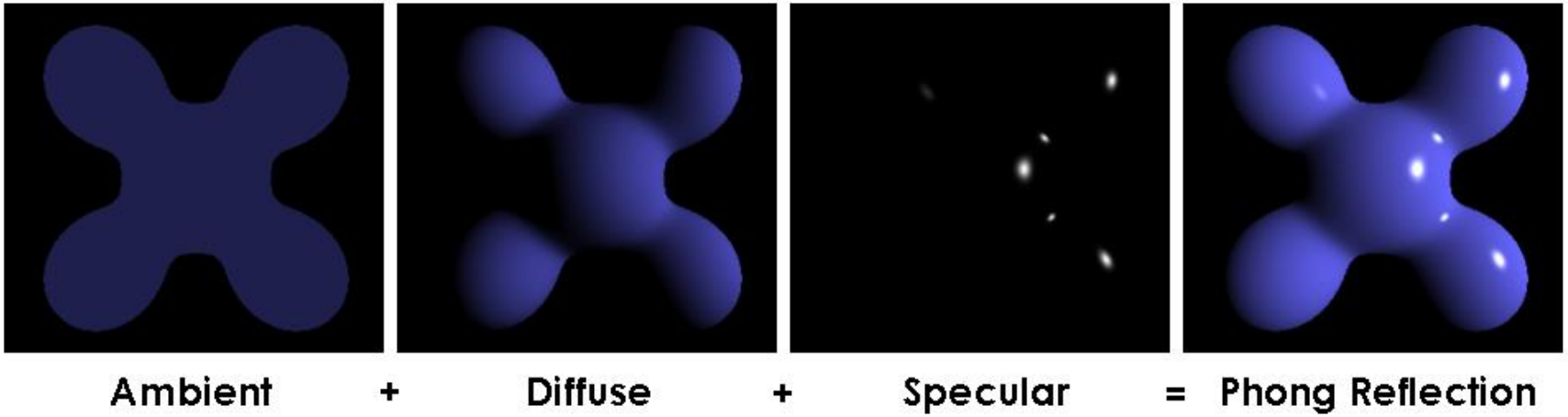
$$I = I_a k_a + k_d \sum_j I_{in,j} \, \left(l_j \cdot N\right) + k_s \sum_j I_{in,j} \, (ra)^S$$

- In der Regel wird die Berechnung auf drei Farbanteile aufgeteilt:

$$I_{\mathbf{r}} = I_{a,\mathbf{r}} k_{a,\mathbf{r}} + I_{in,\mathbf{r}} \left(k_{d,\mathbf{r}} \, (l \cdot N) + k_s (ra)^S\right)$$
$$I_{\mathbf{g}} = I_{a,\mathbf{g}} k_{a,\mathbf{g}} + I_{in,\mathbf{g}} (k_{d,\mathbf{g}} \, (l \cdot N) + k_s (ra)^S)$$
$$I_{\mathbf{b}} = I_{a,\mathbf{b}} k_{a,\mathbf{b}} + I_{in,\mathbf{b}} (k_{d,\mathbf{b}} \, (l \cdot N) + k_s (ra)^S)$$

Phong Reflexionsmodell

- Komponentendarstellung:



https://en.wikipedia.org/wiki/Phong_shading

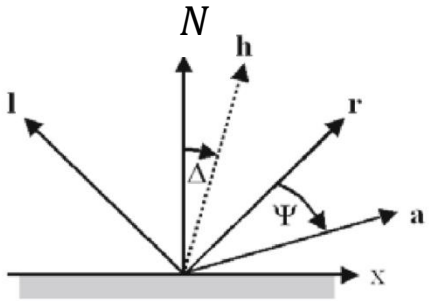
Blinn-Reflexionsmodell

- Motivation: Aufwand der Berechnung von r bei Phong-Reflexionsmodells recht hoch:

$$r = 2N\cos\theta - l = 2(l \cdot N) \cdot N - l$$

- Blinn: Statt r zu berechnen ermittelt man den Vektor h:
  $h = \frac{l+a}{|l+a|}$ verhält sich proportional zum r-Vektor.
  - Der *Halfwayvektor* h entspricht der Normalen zu einer hypothetischen Oberfläche eines perfekt spiegelnden Körpers
  - Der spiegelnde Anteil wird also wie folgt berechnet:

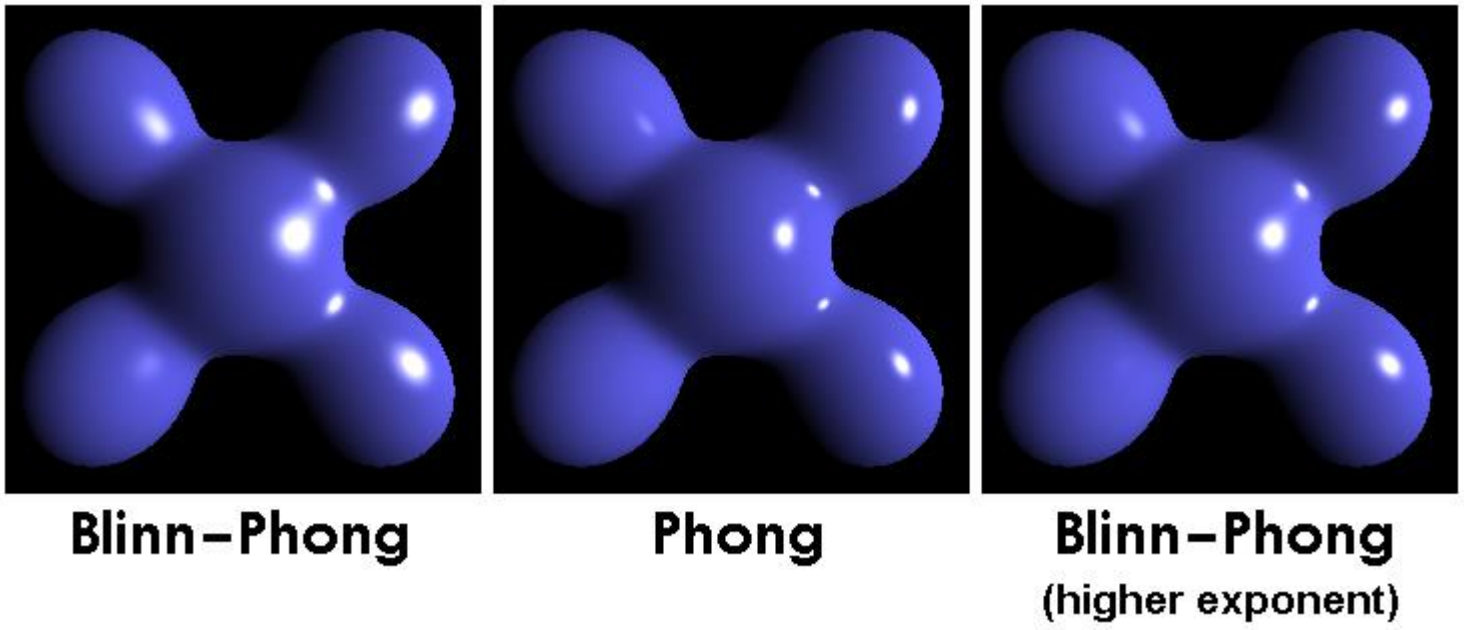
$$I_s = I_{in} \cdot k_s \cdot (h \cdot N)^S$$



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Blinn-Reflexionsmodell

- Betrag des Winkels zwischen N und h ist nur halb so groß wie zwischen r und a
  - Lösung: um gleiches Ergebnis zu erzielen Exponent S erhöhen.



https://de.wikipedia.org/wiki/Blinn-Beleuchtungsmodell

Abschwächung des Lichts durch Abstände

- Einfache Verbesserung: Modellierung der Streuung des Lichts in Abhängigkeit von der Entfernung c des Objektes zum Betrachter:

$$I = I_a k_a + \frac{I_{in}(k_d\ (l \cdot N) + k_s(ra)^S)}{c + \varepsilon}$$

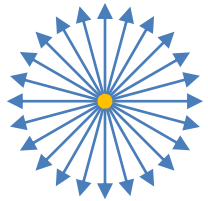
- Zum Abstand wird eine konstante ε addiert um eine Division durch 0 zu vermeiden.
- In diesem Beispiel linearer Abfall, Alternative: Modellierung eines exponentiellen Abfalls

7.3. LICHTQUELLEN

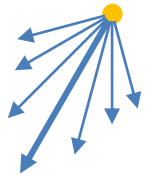
Arten von Lichtquellen

- Headlight
  - Lichtquelle, die direkt aus der Betrachterposition eine Szene beleuchtet, d.h. $l = a$

- Punktquellen (Standard bei lokalen Beleuchtungsmod.)
  - Punktförmige Lichtquelle an Position P
  - Radialsymmetrisch gleichmäßige Abstrahlung
  - Parallele Strahlen bei unendlich entfernter Lichtquelle

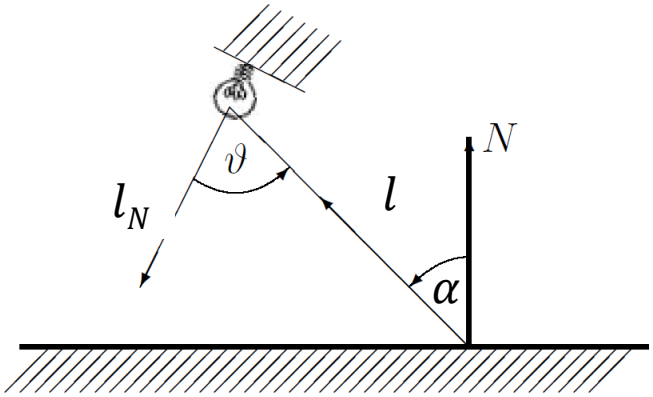


- Gerichtete Lichtquellen (Spotlights)
  - Strahler an Position P
  - Abstrahlintensität abhängig vom Abstrahlwinkel



Gerichtete Lichtquellen im Beleuchtungsmodell

- Erweiterung des Phong-Reflexions-Modells:
  - Quellintensität I_{in} hängt von Winkel v zwischen den Vektoren $-l$ und l_N ab.



  - Gewichtung der Intensität durch $\cos(v) = l_N \cdot (-l)$
  - Fokussierung mit Exponent t

$$I = I_a k_a + I_{in}(l_N \cdot (-l))^t (k_d\,(l \cdot N) + k_s(ra)^S)$$

S. Krömker: Computergrafik I, Universität Heidelberg

Beleuchtung in OpenGL

- Aktivierung eines Lichtmodells: **`glEnable(GL_LIGHTING)`**
  - Setzt Befehl **`glColor`** außer Kraft!
  - Voreingestellt ist ein graues Ambiente-Licht!

- Einstellungen am Reflexionsmodells: **`glLightModelf(name, param)`**
  - **`Z.B. GL_LIGHT_MODEL_TWO_SIDE:`** Einseitig oder zweiseitige Beleuchtungsberechnung

- Spätestens beim Einschalten des Lichts müssen normierte Normalen der Oberflächen bekannt sein: **`glNormal3f(…)`**
  - Das muss der Programmierer selbst übernehmen!

Beleuchtung in OpenGL

- Festlegung von Lichtquelleneigenschaften: **`glLightf(light, name, param)`**
  - **`GLenum light`** ist eine Durchnummerierung der Lichtquellen von **`GL_LIGHT0`** bis **`GL_LIGHT7`**
  - **`Name`** und **`parameter`** legen Eigenschaften und deren Parameter fest:
    - **`GL_AMBIENT:`** RGBA-Wert für ambienten Anteil der Lichtquelle
    - **`GL_DIFFUSE:`** RGBA-Wert für diffusen Anteil der Lichtquelle
    - **`GL_SPECULAR:`** RGBA-Wert für den spekularen Anteil der Lichtquelle
    - **`GL_POSITION:`** (x,y,z,w)-Position der Lichtquelle. w=0: Lichtquelle im Unendlichen
- Einschalten einer Lichtquelle: **`glEnable(GL_LIGHT0)`**.
- Spotlights werden ebenfalls über **`glLightf(…)`** spezifiziert.

Beleuchtung in OpenGL

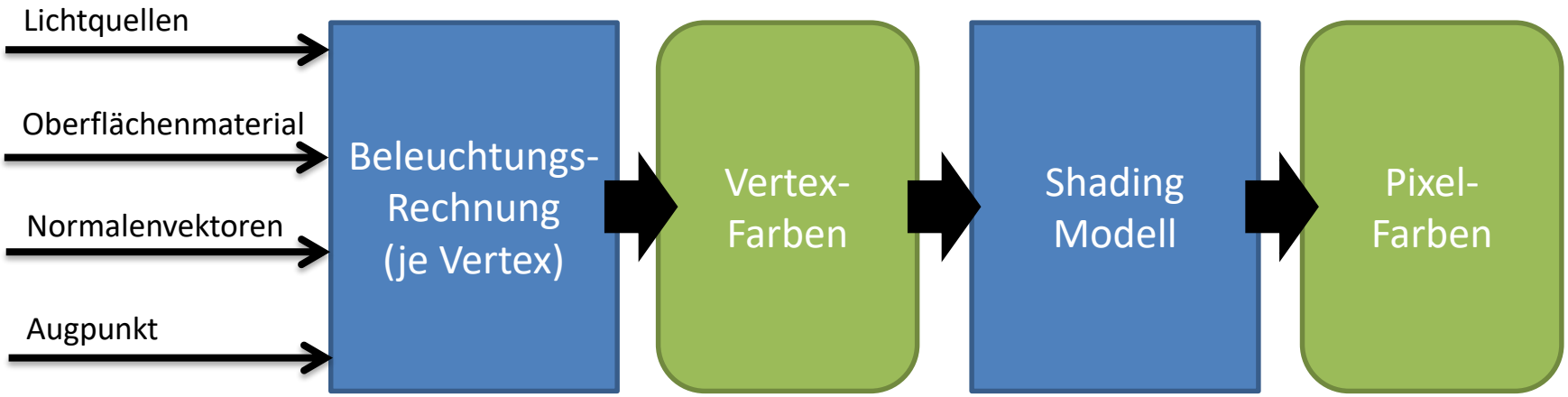
- Festlegung von Oberflächeneigenschaften: **`glMaterialf(face,name,param)`**
  - **`Face`** legt fest welche Seite die Eigenschaften zugewiesen werden (front, back)
  - **`Name`** und **`parameter`** legen Eigenschaften und deren Parameter fest:
    - **`GL_AMBIENT:`** RGBA-Wert für ambiente Reflexion
    - **`GL_DIFFUSE:`** RGBA-Wert für diffusen Reflexion
    - **`GL_AMBIENT_AND_DIFFUSE :`** gleiche RGBA-Werte für *ambiente* und *diffus*
    - **`GL_SPECULAR:`** RGBA-Wert für den spekularen Anteil der Reflexion
    - **`GL_SHININESS:`** Spiegelungsexponent S
    - **`GL_EMISSION:`** emmisive Farbe des Materials

- Zweite Möglichkeit: Color Material Mode **`glEnable(GL_COLOR_MATERIAL)`**
  - **`glColorMaterial(face, mode)`** legt fest welche Seite der Oberfläche bezüglich welcher Materialeigenschaft durch **`glColor`**-Aufrufe geändert wird.

7.4. SCHATTIERUNG

Schattierungsverfahren

- Reflexions- bzw. Lichtmodelle beschreiben die Intensität an jedem beliebigen Punkt im Raum
- Bei Berechnung für jedes Pixel (Millionen!) sehr aufwendig
- Abhilfe:
  - ➔ Auswertung nur an Vertices, Färben der Pixel anhand schnellerer Schattierungsverfahren („Shading“)
  - ➔ Ausnahme: Phong-Shading, *dazu später mehr*



Flat Shading

- Auswertung des Beleuchtungsmodells an einem Vertex eines Polygons
- Alle Pixel des Polygons erhalten diesen gleichen Farbwert
- Als Normalenvektor wird die Flächennormale des Polygons verwendet:

$$\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} v_{1,x} - v_{2,x} \\ v_{1,y} - v_{2,y} \\ v_{1,z} - v_{2,z} \end{pmatrix} \times \begin{pmatrix} v_{2,x} - v_{3,x} \\ v_{2,y} - v_{3,y} \\ v_{2,z} - v_{3,z} \end{pmatrix} = \begin{pmatrix} (v_{1,y} - v_{2,y})(v_{2,z} - v_{3,z}) - (v_{1,z} - v_{2,z})(v_{2,y} - v_{3,y}) \\ (v_{1,z} - v_{2,z})(v_{2,x} - v_{3,x}) - (v_{1,x} - v_{2,x})(v_{2,z} - v_{3,z}) \\ (v_{1,x} - v_{2,x})(v_{2,y} - v_{3,y}) - (v_{1,y} - v_{2,y})(v_{2,x} - v_{3,x}) \end{pmatrix}$$

  - In OpenGL Definition beim Anlegen des Polygons:

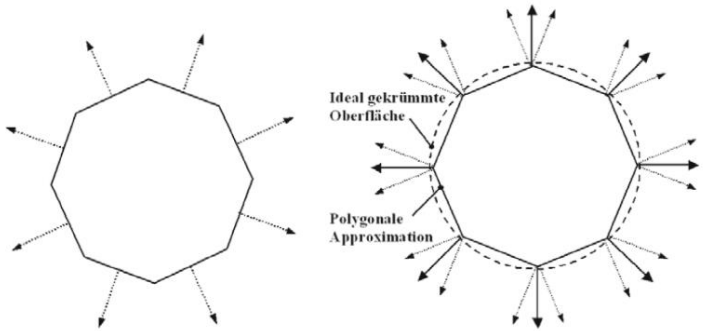
```
glBegin(GL_POLYGON)
   glNormal3f(0.0, 0.0, 1.0);
   glVertex3f(0.0, 0.0, 1.0);
   glVertex3f(0.0, 1.0, 1.0);
   glVertex3f(1.0, 0.0, 1.0);
glEnd();
```

- Aktivierung des Flat-Shadings in OpenGL: **`glShadeModel(GL_FLAT);`**
- Facettenartiges aussehen: Sprünge zwischen Polygonen
  - Abhilfe: kleinere Polygone bis Pixelgröße
- Sehr schnell, facettenartige Ansicht manchmal gewünscht (z.B. CAD)

Gouraud Shading

- Ziel: stetiger Farbverlauf durch lineare Interpolation der Pixelfarbe

- Ablauf der Berechnung:
  1. Berechnung der Flächennormalen des Polygons.
  2. An jedem Vertex Mittelung der Normalen angrenzender Flächen → **Vertex-Normalenvektor**



  3. Beleuchtungsrechnung für jeden Vertex → Vertexfarbe
  4. Farbwerte der Pixel einer Fläche durch lineare Interpolation
     - Zunächst entlang der Kanten
     - Dann zwischen den Kanten entlang der Scanline

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Gouraud Shading

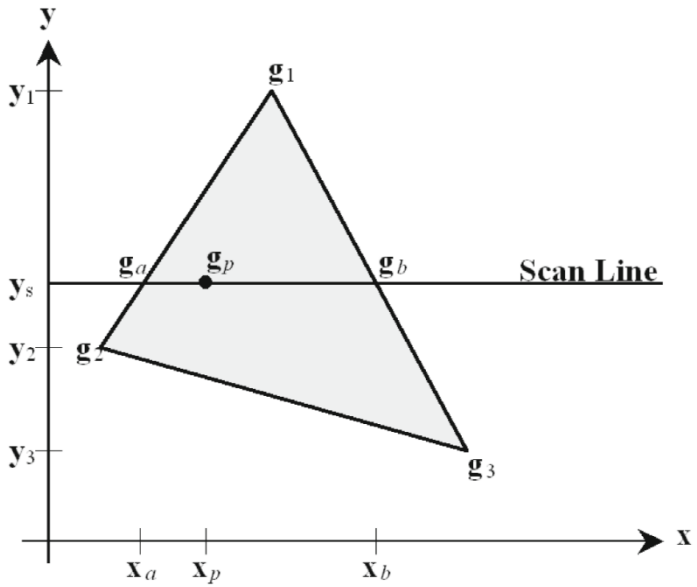
- Interpolation der Intensität: Scan-Line-Algorithmus

$$g_a = g_1 - (g_1 - g_2) \cdot \frac{y_1 - y_s}{y_1 - y_2}$$

$$g_b = g_1 - (g_1 - g_3) \cdot \frac{y_1 - y_s}{y_1 - y_3}$$

$$g_p = g_b - (g_b - g_a) \cdot \frac{x_b - x_p}{x_b - x_a}$$

Rückführung auf konstante Inkremente möglich



- Aktivierung in OpenGL: **`glShadeModel(GL_SMOOTH);`**
- Wichtig: Jedem Vertex muss eine Normale zugeordnet werden:

```
glBegin(GL_POLYGON)
   glNormal3f(0.5, 0.5, 0.707);
   glVertex3f(1.0, 1.0, 0.0);
   glNormal3f(-0.5, 0.5, 0.707);
   glVertex3f(0.2, 0.5, 0.0);
   :
glEnd();
```

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Gouraud Shading

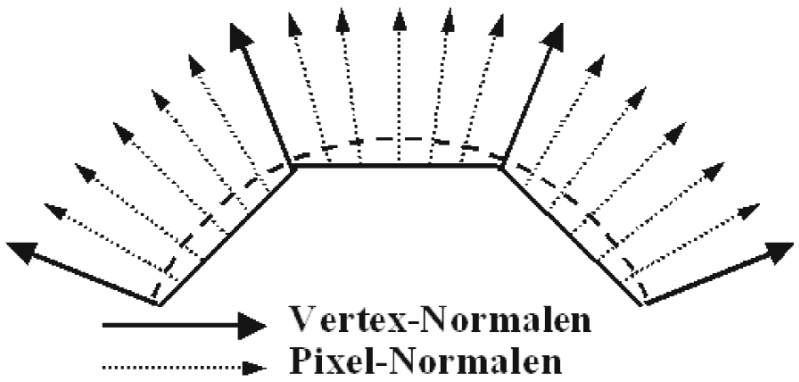
- Bewertung:
  - Farb- und Helligkeitssprünge werden eliminiert
  - Mittlere Polygonanzahl reicht bereits aus für akzeptable Bildqualität
  - Glanzlichter werden nicht korrekt dargestellt
  - Bei Bewegung: Aufblitzen von Glanzlichtern
  - Silhouette des Objektes bleibt so eckig wie das polygonale Modell
  - Probleme bei Spotlights und lokalen Lichtquellen: Ausfransen der Ränder

- Vertex-Normale nur bis zu einem gewissen Unterschied der Flächennormalen sinnvoll → Limitierung durch Grenzwinkel

- Guter Trade-off zwischen Qualität und Geschwindigkeit

Phong Shading

- Interpolation der Normalen (Phong) statt der Intensitäten (Gouraud) pro Pixel



- Ablauf der Berechnung:
  1. Berechnung der Flächennormalenvektoren und Vertexnormalenvektoren wie beim Gouraud Shading
  2. Interpolation der Pixel-Normalenvektoren: Scanline-Algorithmus.
  3. Beleuchtungsberechnung für jedes Pixel → Farbwert pro Pixel

Phong Shading

- Interpolation der Normalen: Scan-Line-Algorithmus

1.) $N_a = N_1 - (N_1 - N_2) \cdot \frac{y_1 - y_s}{y_1 - y_2}$

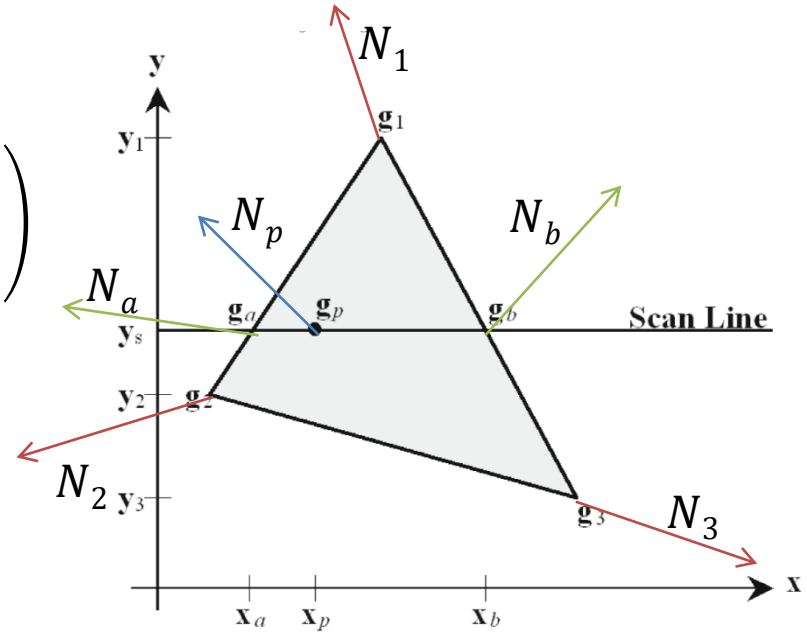
2.) Normalisierung:

$$N_{a'} = \frac{N_a}{|N_a|} = \frac{1}{\sqrt{{N_{a,x}}^2 + {N_{a,y}}^2 + {N_{a,z}}^2}} \cdot \begin{pmatrix} N_{a,x} \\ N_{a,y} \\ N_{a,z} \end{pmatrix}$$

3.) $N_b = N_1 - (N_1 - N_3) \cdot \frac{y_1 - y_s}{y_1 - y_3}$

4.) Normalisierung → $N_{b'}$

5.) $N_p = N_{b'} - (N_{b'} - N_{a'}) \cdot \frac{x_b - x_p}{x_b - x_a}$



Rückführung auf konstante Inkremente möglich!

Phong Shading

- Bewertung:
  - Beseitigt gravierende Probleme des Gouraud-Shadings (Glanzlichter, Ränder)
  - Silhouette des Objektes bleibt weiterhin so eckig wie das polygonale Modell
  - Grundlage für weitere Verfahren wie z.B. Bump Mapping

- Im Compatibility-Profile von OpenGL nicht wählbar

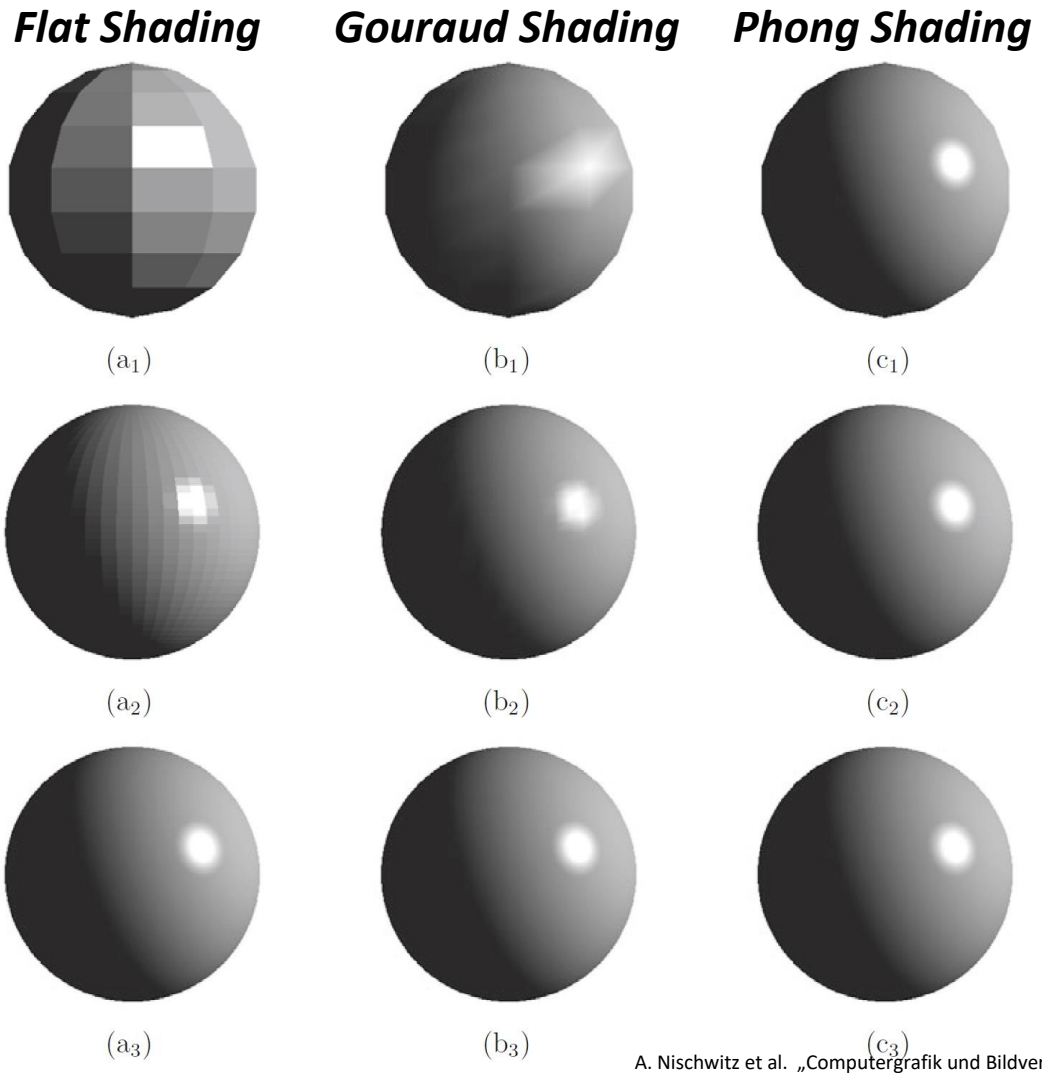
  → eigene Implementierung als Shader notwendig

Vergleich Flat/Gouraud/Phong Shading

- Vergleich der Schlüsselkomponenten

	Flat-Shading	Gouraud-Shading	Phong-Shading
Verwendete Normale	Flächennormale	Vertexnormale	Interpolierte Normale
Beleuchtungsrechnung	An einem Vertex des Polygons	An allen Vertices des Polygons	An jedem gerasterten Pixel
Farbe des Pixels	Alle Pixel im Polygon gleiche Farbe	Lineare Interpolation anhand Farbe der Vertices des Polygons	Farbe durch Beleuchtungsrechnung an jedem Pixel

Vergleich Flat/Gouraud/Phong Shading

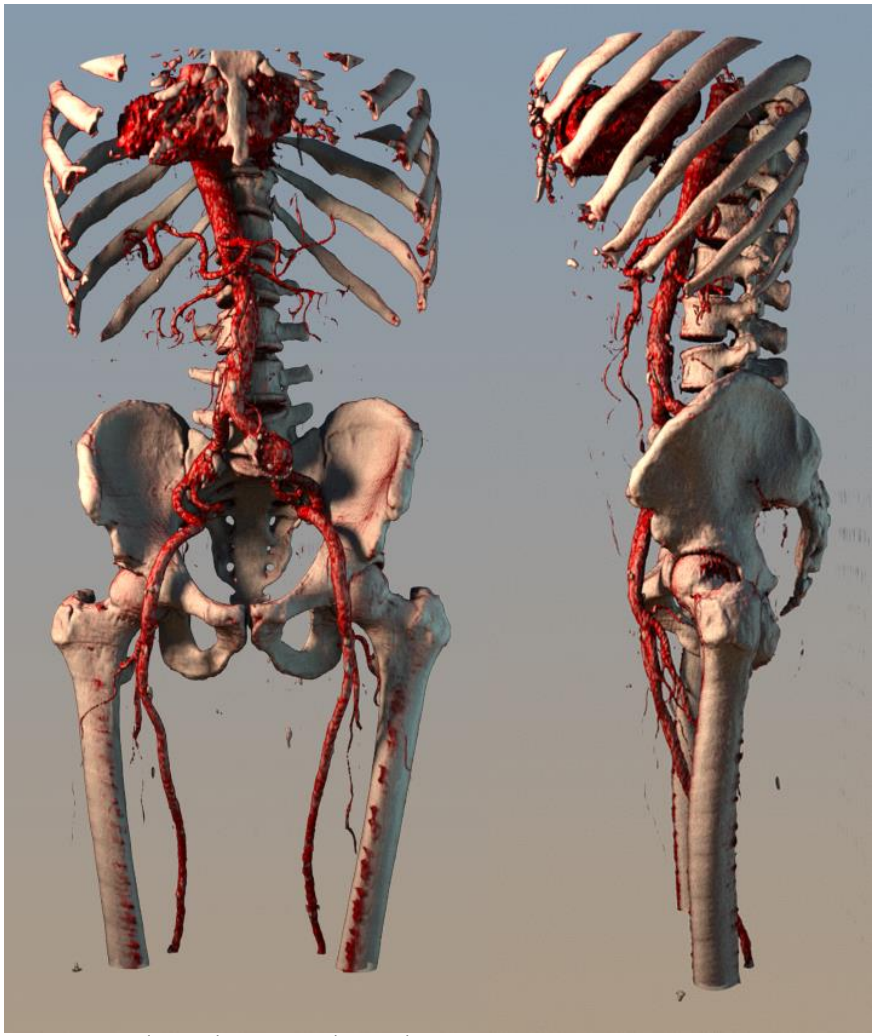


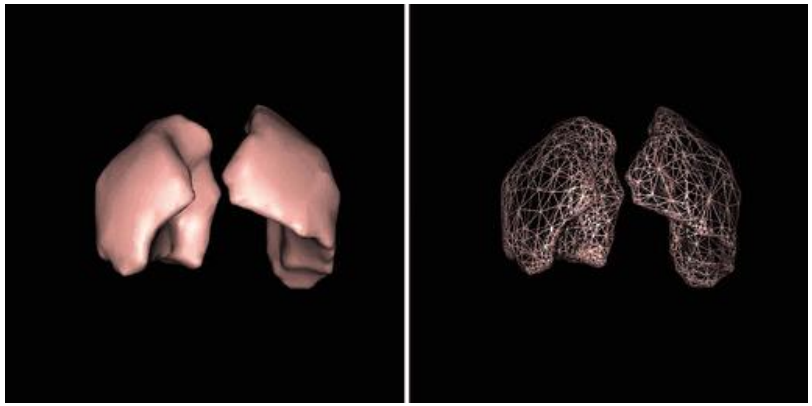
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

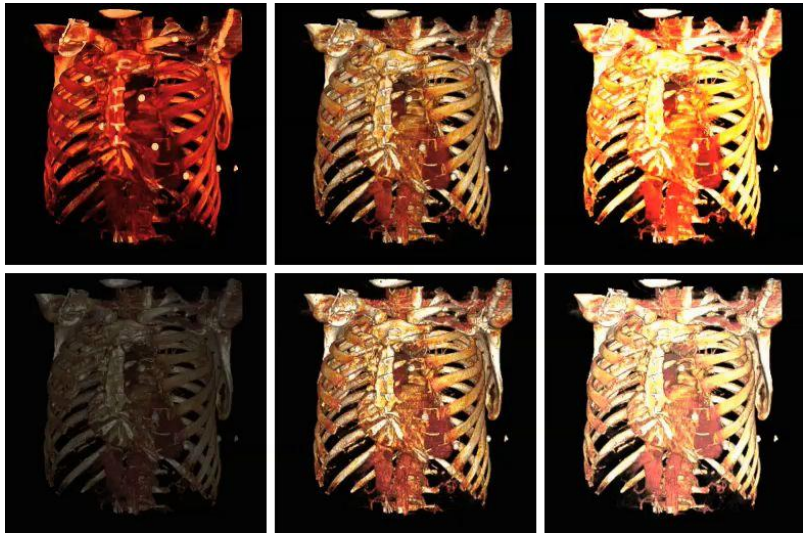
Vergleich Flat/Gouraud/Phong Shading

- Schattierungsverfahren und Polygonauflösung (Tesselierung) sind austauschbar im Hinblick auf Bildqualität
  - Je einfacher das Schattierungsverfahren desto feiner muss das Polygonnetz sein.
- Bei gleicher Bildqualität: was ist der effizienteste Weg?
  - Flat Shading vs. Phong Shading:
    - Flat Shading mit vielen Polygonen langsamer als Phong Shading mit wenigen Polygonen
  - Flat Shading vs. Gouraud Shading
    - Gouraud-Shading benötigt 10-100 mal weniger Polygone als Flat Shading
    - Gouraud-Shading wertet an 3 Vertices die Beleuchtungsfunktion aus, Flat Shading nur 1 mal
    - In der Summe: Gouraud Shading ca. 3-30 mal effizienter als Flat Shading
  - Gouraud Shading vs. Phong Shading
    - Phong benötigt ca. Faktor 10 weniger Polygone, Limit durch sichtbare Silhouette
    - Auswertung pro Pixel deutliche aufwändiger
- Gouraud Shading und Blinn-Reflexionsmodell oft der Standard in der interaktiven Computergrafik

Beispiele aus der medizinischen Visualisierung

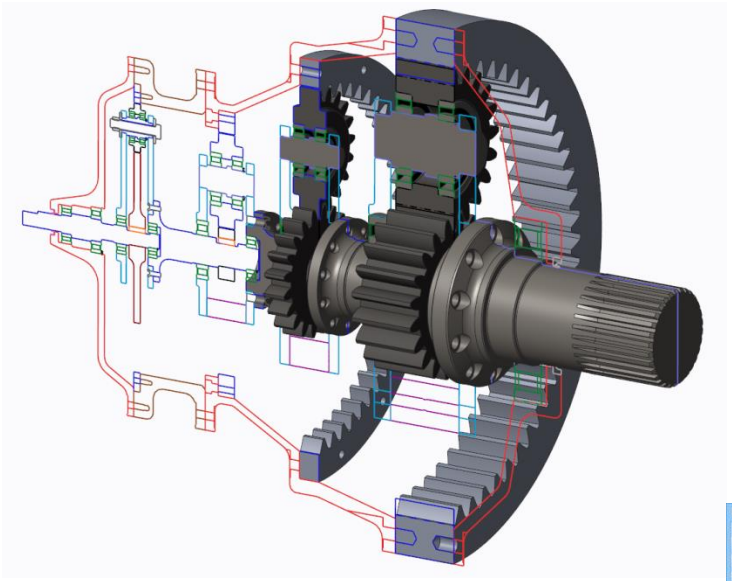


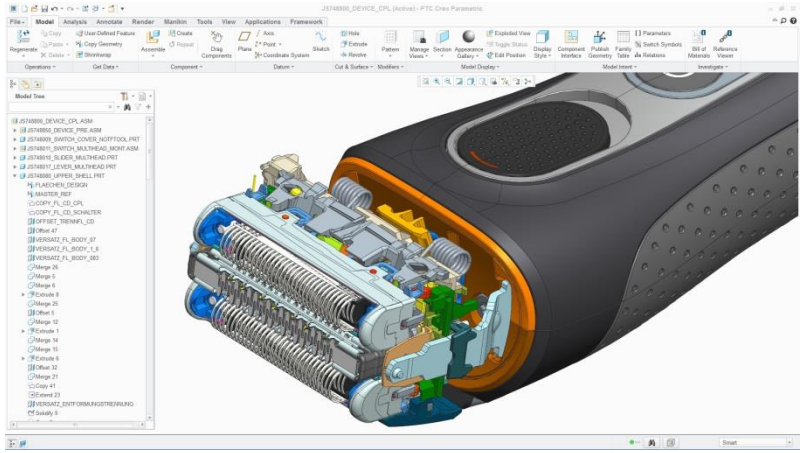


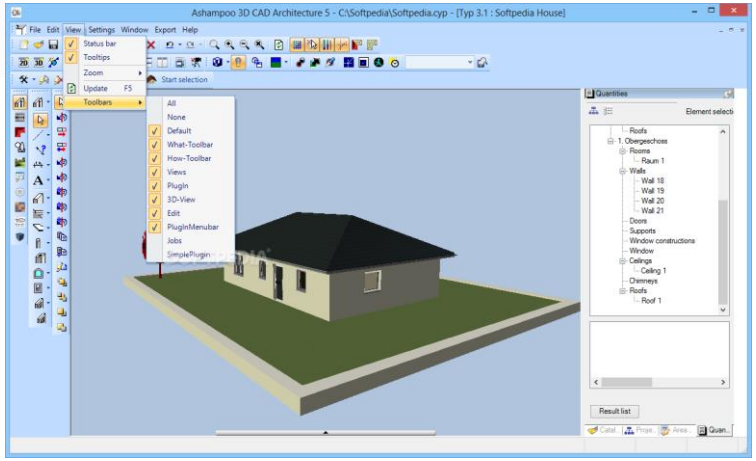


https://plus.google.com/photos/+ChadHaney/albums/5936877961446459345
https://en.wikibooks.org/wiki/Basic_Physics_of_Nuclear_Medicine/Three-Dimensional_Visualization_Techniques

Beispiele aus der CAD-Visualisierung







http://web-designbristol.com/wp-content/uploads/2016/03/computer-aided-desig-and-drafting.png
http://www.softpedia.com/get/Science-CAD/Ashampoo-3D-CAD-Architecture.shtml
http://www.heise.de/ct/ausgabe/2014-18-aktuell-Technische-Anwendungen-2284326.html

ZUSAMMENFASSUNG

Zusammenfassung

- Licht ist Ursache der Farbwahrnehmung
  - Farbmodelle in der Computergrafik zur Beschreibung von Farben: RGB, CMY(K), HSV, LAB
- Komplexe Interaktion von Licht mit Materie:
  - Spiegelnde und diffuse Reflexion, Absorption, Brechung, Dispersion, Fresnel Effekt
  - Vereinfachungen für Computergrafik notwendig
- Lokale vs. Globale Beleuchtung
- Reflexionsmodelle in der interaktiven Computergrafik
  - Phong Reflexionsmodell: ambiente, diffuse, spekulare Intensitätskomponenten
  - Blinn-Phong Reflexionsmodell: Vereinfachte Berechnung gegenüber Phong
- Lichtquellen
  - Gerichtet, Punktförmig,
- Schattierungsverfahrung: setzen Vertex-bezogene Beleuchtungsberechnung für alle Pixel eines Objektes um
  - Flat Shading, Gouraud Shading, Phong Shading

ÜBUNGSAUFGABEN

Übungsaufgaben

2. Ein Lichtstrahl trifft aus der Luft kommend im 45° Winkel auf eine ideal glatte Wasseroberfläche. Der Brechungsindex von Luft beträgt $n_L = 1{,}00$, der Brechungsindex von Wasser beträgt $n_W = 1{,}33$.

a) Wie groß ist der Reflexionswinkel?

b) Wie groß ist der Brechungswinkel?

c) Welcher Intensitätsanteil wird reflektiert und absorbiert/transmittiert?

Lösung

Ein Lichtstrahl trifft aus der Luft kommend im 45° Winkel auf eine ideal glatte Wasseroberfläche. Der Brechungsindex von Luft beträgt $n_L = 1,00$, der Brechungsindex von Wasser beträgt $n_W = 1,33$.

a) Wie groß ist der Reflexionswinkel?

Da es sich um eine ideal glatte (damit ideal reflektierende) Oberfläche handelt, entspricht der gesuchte Reflexionswinkel β dem Einfallswinkel α: $\beta = \alpha = 45°$.

b) Wie groß ist der Brechungswinkel?

Der Brechungswinkel berechnet sich aus dem Verhältnis der beiden Brechungsindizes und dem Einfallswinkel:

$$\frac{\sin\alpha}{\sin\gamma} = \frac{n_W}{n_L} \quad \Rightarrow \sin\gamma = \frac{\sin\alpha}{\frac{n_W}{n_L}} = \frac{\sin\alpha \ \cdot n_L}{n_W}$$

$$\sin\gamma = \frac{0{,}7071 \cdot 1}{1{,}33} = 0{,}5317 \Rightarrow \gamma = \arcsin 0{,}5317 = \mathbf{32,12°}$$

Lösung (2)

c) Welcher Intensitätsanteil wird reflektiert und absorbiert/transmittiert?

Der reflektierte Intensitätsanteil wird über die Formel zum Fresnel-Effekt bestimmt:

$$R = \frac{1}{2}\left(\frac{\sin^2(\gamma-\alpha)}{\sin^2(\gamma+\alpha)} + \frac{\tan^2(\gamma-\alpha)}{\tan^2(\gamma+\alpha)}\right)$$

$$R = \frac{1}{2}\left(\frac{\sin^2(32{,}12° - 45°)}{\sin^2(32{,}12° + 45°)} + \frac{\tan^2(32{,}12° - 45°)}{\tan^2(32{,}12° + 45°)}\right)$$

$$R = \frac{1}{2}\left(\frac{\sin^2(-12{,}88°)}{\sin^2(77{,}12°)} + \frac{\tan^2(-12{,}88°)}{\tan^2(77{,}12°)}\right)$$

$$R = \frac{1}{2}(0{,}0523 + 0{,}0027) = \mathbf{0,0275}$$

D.h. etwa $2{,}75\%$ der einfallenden Intensität wird reflektiert. Aufgrund der Energieerhaltung wird der restliche Anteil absorbiert bzw. transmittiert: $A = 100\% - 2{,}75\% = 97{,}25\%$.

Übungsaufgaben

3. Bei achromatischem Licht (Nur Betrachtung der Intensitäten) ist eine Lichtquelle an der Stelle $L = (10|\ 5|10)$ gegeben. Diese hat die Quellintensität $I_{in} = 10$. Ein Betrachter befindet sich im Punkt $A = (0|5|10)$. Das Licht fällt auf ein Dreieck mit den Vertices $v_0 = (5|0|5)$, $v_1 = (10|1|5)$, $v_2 = (8|2|0)$. Dieses hat die Materialeigenschaften $k_a = 0{,}3$, $k_d = 0{,}3$ und $k_s = 1$, sowie $s = 2$. Berechnen Sie die Lichtintensität die beim Betrachter A für den Vertex v_0 ankommt nach dem Phong Reflexionsmodell. Gehen Sie von einem Ambiente-Licht $I_a = 3$ aus. Nutzen Sie die Flächennormale.

Lösung

3. Bei achromatischem Licht (Nur Betrachtung der Intensitäten) ist eine Lichtquelle an der Stelle $L = (10|\ 5|10)$ gegeben. Diese hat die Quellintensität $I_{in} = 10$. Ein Betrachter befindet sich im Punkt $A = (0|5|10)$. Das Licht fällt auf ein Dreieck mit den Vertices $v_0 = (5|0|5)$, $v_1 = (10|1|5)$, $v_2 = (8|2|0)$. Dieses hat die Materialeigenschaften $k_a = 0,3$, $k_d = 0,3$ und $k_s = 1$, sowie $s = 2$. Berechnen Sie die Lichtintensität die beim Betrachter A für den Vertex v_0 ankommt nach dem Phong Reflexionssmodell. Gehen Sie von einem Ambiente-Licht $I_a = 3$ aus. Nutzen Sie die Flächennormale.

Betrachte die Beleuchtungsformel von Phong

$$I = I_a k_a + I_{in}(k_d\ (l \cdot n) + k_s(ra)^S)$$

I_a, k_a, I_{in}, k_d, k_s und s sind gegeben. Wir benötigen die Vektoren l, n, r und a.

1. Berechnen des Vektors l und Normalisierung des Vektors:

$$l = L - v_0 = \begin{pmatrix}10\\5\\10\end{pmatrix} - \begin{pmatrix}5\\0\\5\end{pmatrix} = \begin{pmatrix}5\\5\\5\end{pmatrix}$$

$$l' = \frac{l}{\sqrt{{l_x}^2 + {l_y}^2 + {l_z}^2}} = \frac{l}{\sqrt{75}} = \begin{pmatrix}5/8{,}66\\5/8{,}66\\5/8{,}66\end{pmatrix} = \begin{pmatrix}0{,}58\\0{,}58\\0{,}58\end{pmatrix}$$

Lösung (2)

2. Berechnen des Normalenvektors des Dreiecks:

$$\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} v_{1,x} - v_{0,x} \\ v_{1,y} - v_{0,y} \\ v_{1,z} - v_{0,z} \end{pmatrix} \times \begin{pmatrix} v_{2,x} - v_{0,x} \\ v_{2,y} - v_{0,y} \\ v_{2,z} - v_{0,z} \end{pmatrix} = \begin{pmatrix} 10 - 5 \\ 1 - 0 \\ 5 - 5 \end{pmatrix} \times \begin{pmatrix} 8 - 5 \\ 2 - 0 \\ 0 - 5 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} \times \begin{pmatrix} 3 \\ 2 \\ -5 \end{pmatrix}$$

$$= \begin{pmatrix} 1 \cdot (-5) - 0 \cdot 2 \\ 0 \cdot 3 - 5 \cdot (-5) \\ 5 \cdot 2 - 1 \cdot 3 \end{pmatrix} = \begin{pmatrix} -5 \\ 25 \\ 7 \end{pmatrix}$$

Anmerkung zur Reihenfolge der Vertices im Kreuzprodukt:

- Der Winkel zwischen n und l muss $\leq 90°$ sein, siehe Folien: max Bedingung.
- Testen kann man dies durch Berechnung von $\arccos(l \cdot n)$, hier 54°.
- Ist der Winkel $> 90°$ zeigt die Normale des Dreiecks in entgegengesetzter Richtung, das Licht würde auf die Rückseite des Dreiecks fallen

Erinnerung: Kreuzprodukt

$$\vec{a} \times \vec{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

Normalisierung der Normalen:

$$n' = \frac{n}{\sqrt{{n_x}^2 + {n_y}^2 + {n_z}^2}} = \frac{n}{\sqrt{25 + 625 + 49}} = \begin{pmatrix} -5/26{,}44 \\ 25/26{,}44 \\ 7/26{,}44 \end{pmatrix} = \begin{pmatrix} -0{,}19 \\ 0{,}95 \\ 0{,}26 \end{pmatrix}$$

Lösung (3)

3. Berechnen des Vektors a und Normalisierung des Vektors:

$$a = A - v_0 = \begin{pmatrix} 0 \\ 5 \\ 10 \end{pmatrix} - \begin{pmatrix} 5 \\ 0 \\ 5 \end{pmatrix} = \begin{pmatrix} -5 \\ 5 \\ 5 \end{pmatrix}$$

$$a' = \frac{a}{\sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2}} = \frac{a}{\sqrt{75}} = \begin{pmatrix} -5/8{,}66 \\ 5/8{,}66 \\ 5/8{,}66 \end{pmatrix} = \begin{pmatrix} -0{,}58 \\ 0{,}58 \\ 0{,}58 \end{pmatrix}$$

Erinnerung: Skalarprodukt

$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3.$

4. Berechnung des Vektors r:

$$r = 2(l' \cdot n') \cdot n' - l'$$

$$r = 2 \cdot \left(\begin{pmatrix} 0{,}58 \\ 0{,}58 \\ 0{,}58 \end{pmatrix} \cdot \begin{pmatrix} -0{,}19 \\ 0{,}95 \\ 0{,}26 \end{pmatrix} \right) \cdot \begin{pmatrix} -0{,}19 \\ 0{,}95 \\ 0{,}26 \end{pmatrix} - \begin{pmatrix} 0{,}58 \\ 0{,}58 \\ 0{,}58 \end{pmatrix}$$

$$r = 2 \cdot (0{,}58 \cdot (-0{,}19) + 0{,}58 \cdot 0{,}95 + 0{,}58 \cdot 0{,}26) \cdot \begin{pmatrix} -0{,}19 \\ 0{,}95 \\ 0{,}26 \end{pmatrix} - \begin{pmatrix} 0{,}58 \\ 0{,}58 \\ 0{,}58 \end{pmatrix}$$

$$= 2 \cdot 0{,}59 \cdot \begin{pmatrix} -0{,}19 \\ 0{,}95 \\ 0{,}26 \end{pmatrix} - \begin{pmatrix} 0{,}58 \\ 0{,}58 \\ 0{,}58 \end{pmatrix} = \begin{pmatrix} -0{,}22 \\ 1{,}12 \\ 0{,}31 \end{pmatrix} - \begin{pmatrix} 0{,}58 \\ 0{,}58 \\ 0{,}58 \end{pmatrix} = \begin{pmatrix} -0{,}80 \\ 0{,}54 \\ -0{,}27 \end{pmatrix}$$

r ist bereits normalisiert, da wir ihn aus normalisierten Vektoren berechnet haben .

Lösung (4)

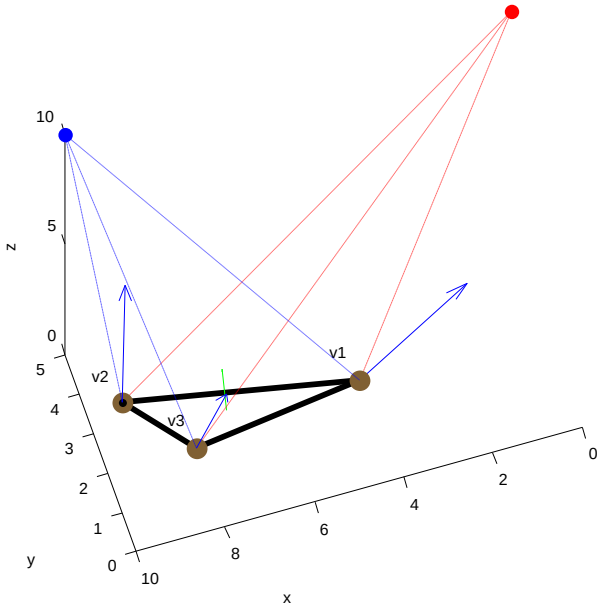
5. Beleuchtungsberechnung

$$I = I_a k_a + I_{in}(k_d\ (l \cdot N) + k_s (ra)^S)$$

$$I = 3 \cdot 0{,}3 + 10 \cdot \left(0{,}3 \cdot \left(\begin{pmatrix} 0{,}58 \\ 0{,}58 \\ 0{,}58 \end{pmatrix} \cdot \begin{pmatrix} -0{,}19 \\ 0{,}95 \\ 0{,}26 \end{pmatrix} \right) + 1 \cdot \left(\begin{pmatrix} -0{,}80 \\ 0{,}54 \\ -0{,}27 \end{pmatrix} \cdot \begin{pmatrix} -0{,}58 \\ 0{,}58 \\ 0{,}58 \end{pmatrix} \right)^2 \right)$$

$$I = 3 \cdot 0{,}3 + 10 \cdot (0{,}3 \cdot 0{,}59 + 1 \cdot 0{,}62^2)$$

$$I = 6{,}51$$



Anmerkung zur Reihenfolge der Vertices im Kreuzprodukt bei Normalenberechnung:

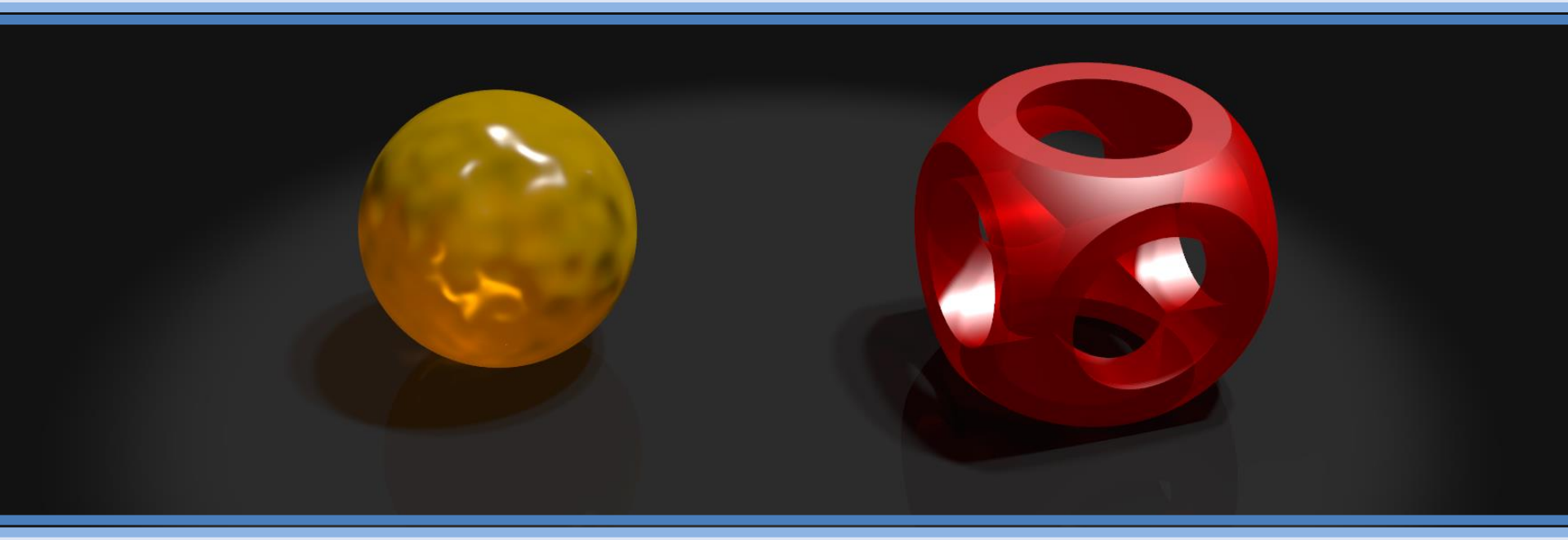
- Eine Normale in die Falsche Richtung würde hier einen negativen Wert für die diffuse Beleuchtungskomponente zur Folge haben, die nicht erlaubt ist!

Übungsfragen Kapitel 7

- Was ist der Fresnel-Effekt?
- Wie entsteht die Farbwahrnehmung eines Objekts bei Bestrahlung mit weißem Licht?
- Aus welchen drei Komponenten setzt sich das Phong-Reflexionsmodell zusammen und von welchen Parametern ist es abhängig?
- Worin liegt der Unterschied zwischen Phong- und Blinn-Reflexionsmodell?
- Beschreiben Sie die Unterschiede der Schattierungsverfahren Flat, Gouraud und Phong Shading. Welches Verfahren ist bei gleich bleibender Bildqualität das effizienteste Verfahren?

Computergrafik

T. Hopp

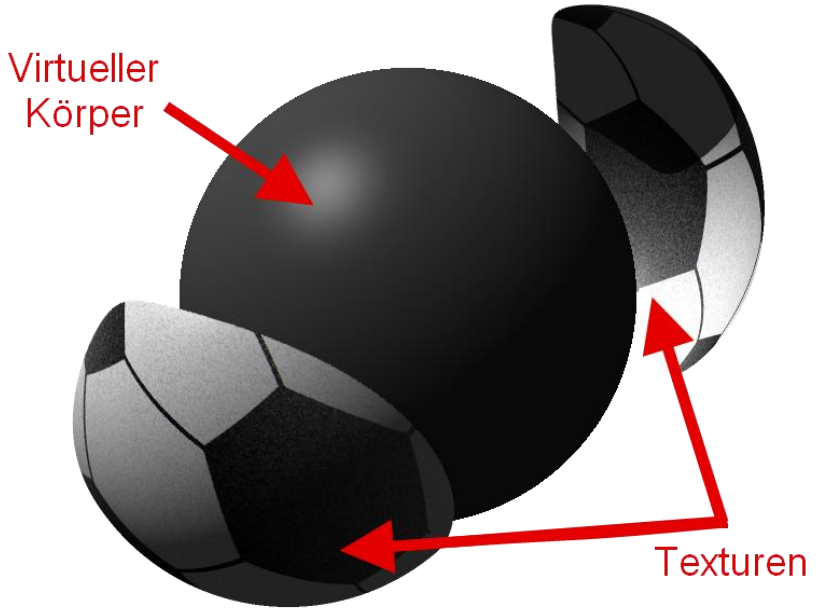


Themenübersicht

1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. **Texturen**
9. Animationen
10. Raytracing
11. Volumenvisualisierung

Texturen

- Bisher: Farbzuordnung eines Pixels basierend auf Shadingverfahren, Farbdefinition etc.
  - i.d.R. künstliches Aussehen
  - Extrem hoher Aufwand um Fotorealismus zu erreichen
- Textur: „Überzug“ eines Objektes mit einer Struktur
  - Einfachste Variante: Texture Mapping



https://de.wikipedia.org/wiki/Textur_%28Computergrafik%29

Anwendungen von Texturen

- **Texture Mapping**: „Fototapete“
  - Ändert diffuse Reflexion eines Objektes
- Gloss Mapping: Glanztexturen
  - Ändert spekulare Reflexion eines Objektes
- **Bump Mapping**: Reliefartige Darstellung
  - Ändert Normalenfeld eines Objektes
- **Environment**-/Shadow-**Mapping**: Approximation globaler Beleuchtung
- Light-Mapping: Imitation von Radiosity-Verfahren

8.1. TEXTURE MAPPING

Texture Mapping

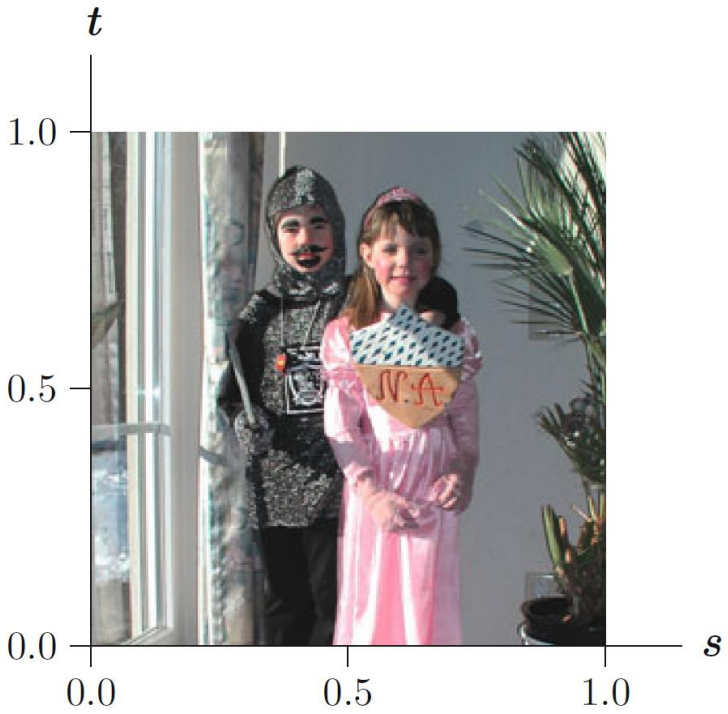
- Verwendung von Bildern als Texturen
  - 1D, 2D, 3D
  - Grauwertbilder, Farbbilder, Farbbilder + Alpha-Kanal
- „Überziehen“ eines Objektes mit einem Bild
  - Zuordnung: Texturkoordinate auf Vertices der Oberfläche
  - Lineare Interpolation in der Rasterisierungsstufe der Renderingpipeline
- Ablauf beim Texture Mapping:
  - Spezifikation der Textur
  - Festlegung wie die Textur auf jedes Pixel aufgetragen wird
    - Texturfilter
    - Mischung von Textur und Beleuchtungsfarbe
  - Zuordnung von Texturkoordinaten zu Vertices
  - Einschalten des Texture Mappings: **`glEnable(GL_TEXTURE_2D)`**

Texture Mapping

- Typische Transformationen

Spezifikation der Textur

- Bildmatrix: $\mathbf{T}$
- Texturkoordinatensystem: (s, t), normiert auf Parameterbereich $[0, 1]$
- Bildpunkt = „Texel“:
  - 1 Wert (Graustufen)
  - 2 Werte (Grausstufen + Transparenz)
  - 3 Werte (Farbe)
  - 4 Werte (Farbe + Transparenz)
- Quantisierung: 1 Bit, 4 Bit, … 32 Bit
- OpenGL: **`glTexImage2D(…)`**
  - Daten aus dem Hauptspeicher



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Spezifikation der Textur

- Beispiel in OpenGL: 512 x 256 Texel, RGBA

```
GLint width = 512;
GLint height = 256;
static GLubyte image[width][height][4];
:
:
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
          width, height, 0,
          GL_RGBA, GL_UNSIGNED_BYTE, image);
```

Level (siehe Mipmapping)

Border (Breite des Randes der Textur)

Datentyp

- Laden/Einlesen des Bildes (hier: *image*) obliegt dem Programmierer
- Oft Skalierung/Zuschneiden des Bildes erforderlich, z.B. auf Zweierpotenz.
  - Extern, oder per GLU-Bibliothek: **`gluScaleImage(…)`**
- Textur ist Teil des OpenGL Zustandsautomaten
  - Zustandswechsel möglichst minimieren!

Spezifikation der Textur

- Texturspeicher: extrem schneller Zugriff innerhalb der Grafikkarte
- Überprüfung ob Speicher für Textur ausreicht:

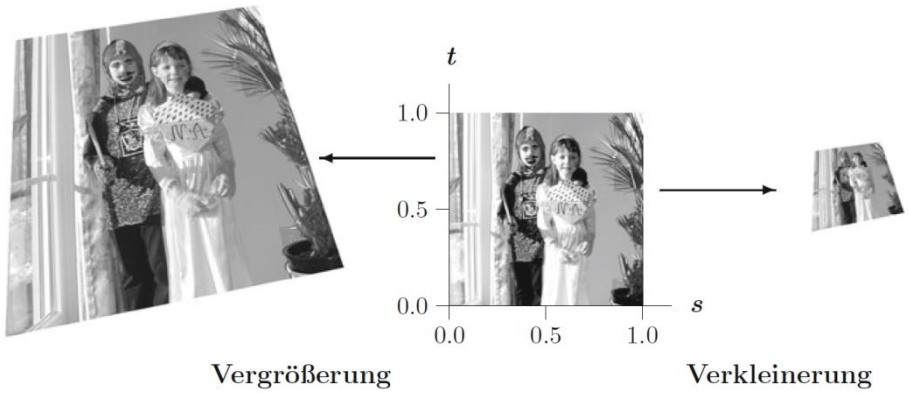
```
glTexImage2D(GL_PROXY_TEXTURE_2D, …);
glGetTexLevelParameteriv(GL_PROXY_TEXTURE_2D,
                         level, GL_TEXTURE_WIDTH,
                         &width);
```

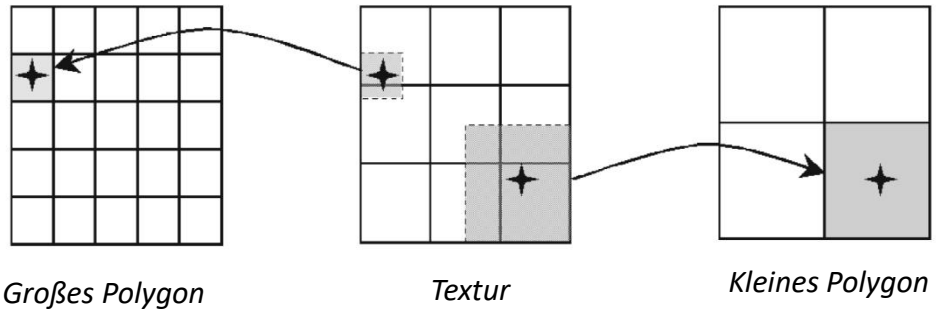
  - Width = 0 → Texturspeicher reicht nicht aus

- Statt Neudefinition auch Modifikation einer Textur möglich
- Sub-Texturen zur Vermeidung von Neudefinitionen
- Multipass-Rendering: rekursive Verwendung von Texturen aus dem Bildspeicher
  - Erheblicher Geschwindigkeitsverlust bei steigender Rekursionstiefe

Textur-Filter

- Beim Abbilden der Texturkoordinaten in die 3D-Szene können erhebliche Verzerrungen der Textur entstehen





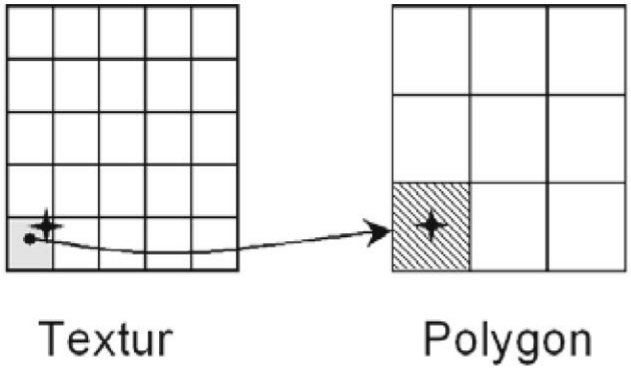
- Keine eindeutige Zuordnung zwischen Texel und Pixel
- Interpolation erforderlich

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

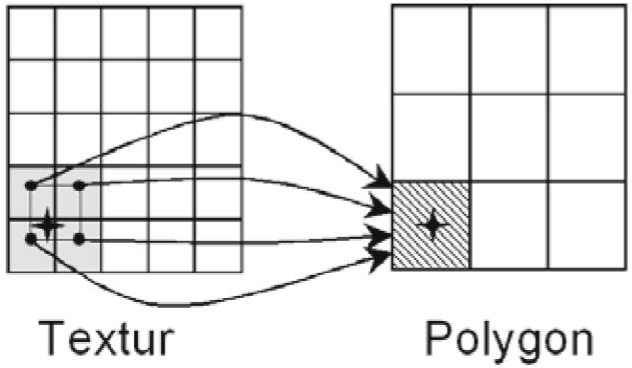
Textur-Filter

Standard-Filter-Methoden

Nearest neighbour



Linear



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Mipmapping

- Vergrößerungsfaktor sollte nicht > 2 oder $< \frac{1}{2}$ werden.
- Mipmapping: Vorfilterung der Texturen und optimale Skalierung
  - Pro Level: Halbierung der Kantenlänge bis zur Größe 1×1

Gauß-Pyramide

MipMap-Level: (0) (1) (2) (3) (4)(5)

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Mipmapping

- Berechnung „offline“ (z.B. in der Initialisierung)
- Generierung der Gauß-Pyramide:
  - In externem Programm: Aufruf von **`glTexImage2D(…)`** zum Einlesen jedes Levels
  - Per GLU-Bibliothek: **`gluBuild2DMipmaps()`**
- Speicherbedarf: $1 + {}^1/_4 + {}^1/_{16} + \ \ldots = 1 + \sum_{n\to\infty} \frac{1}{n^2} = 1\frac{1}{3}$

- Auswahl des Mipmap-Levels:
  - Verkleinerungsfaktoren in x-/y-Richtung: ρ_x, ρ_y
  - Berechnung des Skalierungsfaktors $\lambda = -\log_2\left(\max\left(\frac{1}{\rho_x}, \frac{1}{\rho_y}\right)\right)$
  - Skalierungsfaktor entspricht Mipmap-Level

  Beispiel: maximaler Verkleinerungsfaktor $\rho = \frac{1}{4}$ ➔ $\lambda = 2$

- Verkleinerungsfilter: zwei Interpolationen (Skalierungsfaktor, Farbwerte)

Mipmapping

http://www.tomshardware.com/reviews/ati,819-2.html

Mipmapping

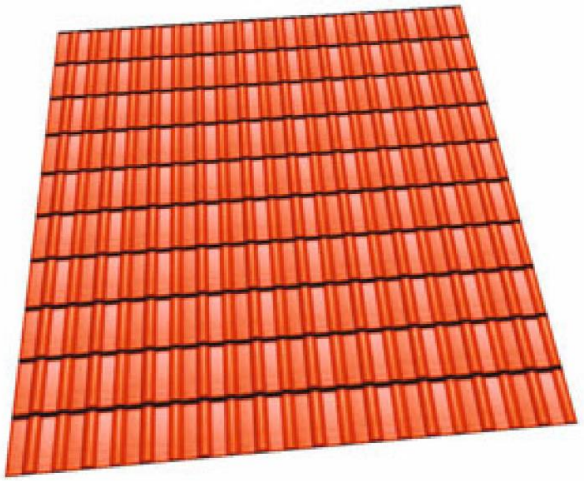
- Probleme bei sehr flachem Blickwinkel

- Mögliche Abhilfen:
  - Vorberechnung zusätzlicher Level:
    - z.B. 8×8-Textur: 8×4, 8×2, 8×1, 4×8, 2×8, 1×8, 4×4,...
    - Hoher Speicherbedarf, heute meist nicht mehr verwendet

  - Abhilfe durch anisotrope Filterung:
    - Auswahl der Mipmap für geringeren Verkleinerungsfaktor
    - Vermeidung von Aliasing-Artefakten: in der anderen Richtung durch Mittelwertbildung über größere Anzahl an Texeln (stärkere Filterung → „anisotrop“)

Texture Wraps

- Texturen in vielen Fällen regelmäßige Struktur
- Textur-Fortsetzungsmodus (Texture Wrap):
  - Definition kleiner Ausschnitte
  - Fortsetzung der Textur über gesamtes Polygon

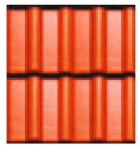


- OpenGL: Zulassen von Texturkoordinaten außerhalb des Bereichs $[0,1]$
  - Z.B. $[0,5]$: fünfmalige Wiederholung in jeweils x- und y-Richtung = 25 Kacheln
  - In OpenGL: hier Texturfortsetzungsmodus **GL_REPEAT**

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

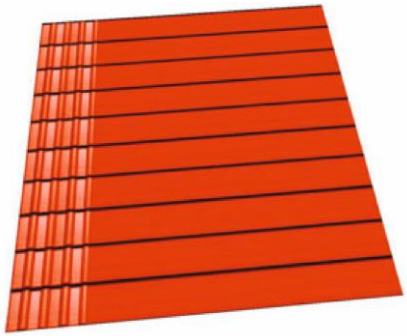
Texture Wraps

- Textur muss geeignet sein:
  - Unterer Rand ähnlich oberem Rand
  - Linker Rand ähnlich rechtem Rand



- Möglichkeit zur Spiegelung
  - In OpenGL: Texturfortsetzungsmodus auf **`GL_MIRRORED_REPEAT`**
- Kappen von Werten außerhalb der Textur: Wiederholung der ersten/letzten Zeile/Spalte
  - In OpenGL: **`GL_CLAMP`**

- Getrennte Einstellung für x-/y-Richtung



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Mischung von Textur und Beleuchtung

- Textur-Fragmentfarbe g_t muss mit Beleuchtungsfarbe g_f aus dem Shading kombiniert werden.

```
glTexEnvf(target, GL_TEXTURE_ENV_MODE, param)
```

- Verschiedene Kombinationsmöglichkeiten:
  - Ersetzen (param = **`GL_REPLACE`**): resultierende Fragmentfarbe $g_r = g_t$
  - Modulieren (param = **`GL_MODULATE`**): komponentenweise Multiplikation:

$$g_r = \begin{pmatrix} R_t \cdot R_f \\ G_t \cdot G_f \\ B_t \cdot B_f \\ A_t \cdot A_f \end{pmatrix}$$

  - Gewichtung mit der Alpha-Komponente der Textur (param = **`GL_DECAL`**)

$$g_r = \begin{pmatrix} (1 - A_t)R_f + A_t R_t \\ (1 - A_t)G_f + A_t G_t \\ (1 - A_t)B_f + A_t B_t \\ A_f \end{pmatrix}$$

Mischung von Textur und Beleuchtung

- Gewichtung mit dem Farbwert der Textur (param = **`GL_BLEND`**)

$$g_r = \begin{pmatrix} (1-R_t)R_f + R_t R_h \\ (1-G_t)G_f + G_t G_h \\ (1-B_t)B_f + B_t B_h \\ A_t \cdot A_f \end{pmatrix}$$

- Addition (param = **`GL_ADD`**): komponentenweise Addition:

$$g_r = \begin{pmatrix} R_t + R_f \\ G_t + G_f \\ B_t + B_f \\ A_t \cdot A_f \end{pmatrix}$$

- Kombination von Beleuchtung und Textur meist per Modulation
  - Definition weißer Materialeigenschaften für Shading → Textur bestimmt Farbe, Beleuchtung bestimmt Helligkeit (3D-Eindruck!)

Mischung von Textur und Beleuchtung

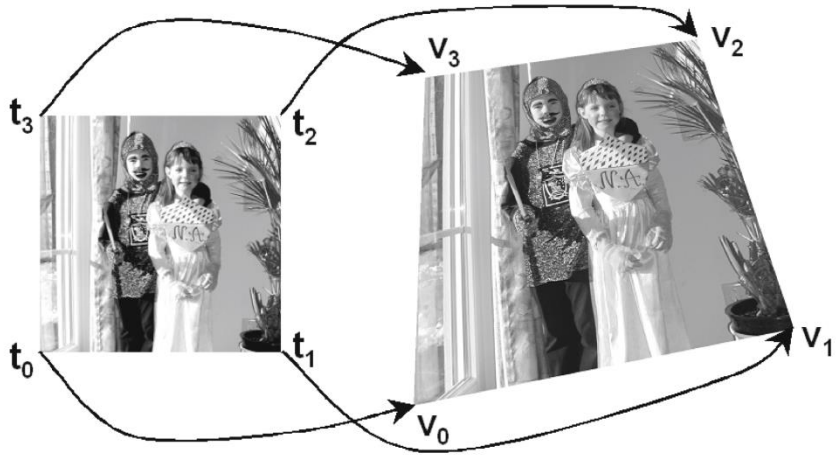
- Glanzlichter (spekulare Beleuchtung) gehen meist verloren
- Abhilfe: Berechnung von zwei Farben pro Vertex/Fragment:
  - Primäre Farbe: diffuser, ambienter und submissiver Beleuchtungsanteil
  - Sekundäre Farbe: spekularer Anteil
- Nachträgliche Addition der Sekundärfarbe
- Beschränkung auf den Wertebereich [0, 1]

http://www.3dgep.com/texturing-and-lighting-with-opengl-and-glsl/

Zuordnung von Texturkoordinaten zu Vertices

- Texturabbildung: $(s, t) \rightarrow (x_w, y_w, z_w)$
  - Jedem Vertex wird eine Texel-Koordinate in der Textur zugewiesen
- Explizite Zuordnung:
  - Definition einer Texel-Koordinate in OpenGL: **`glTexCoord2f`**
  - Zuordnung bei der Definition eines Polygons
  - Beispiel 2D-Textur auf 3D-Polygon aufbringen

```
glBegin(GL_QUADS);
   glTexCoord2fv(t0); glVertex3fv(v0);
   glTexCoord2fv(t1); glVertex3fv(v1);
   glTexCoord2fv(t2); glVertex3fv(v2);
   glTexCoord2fv(t3); glVertex3fv(v3);
glEnd();
```



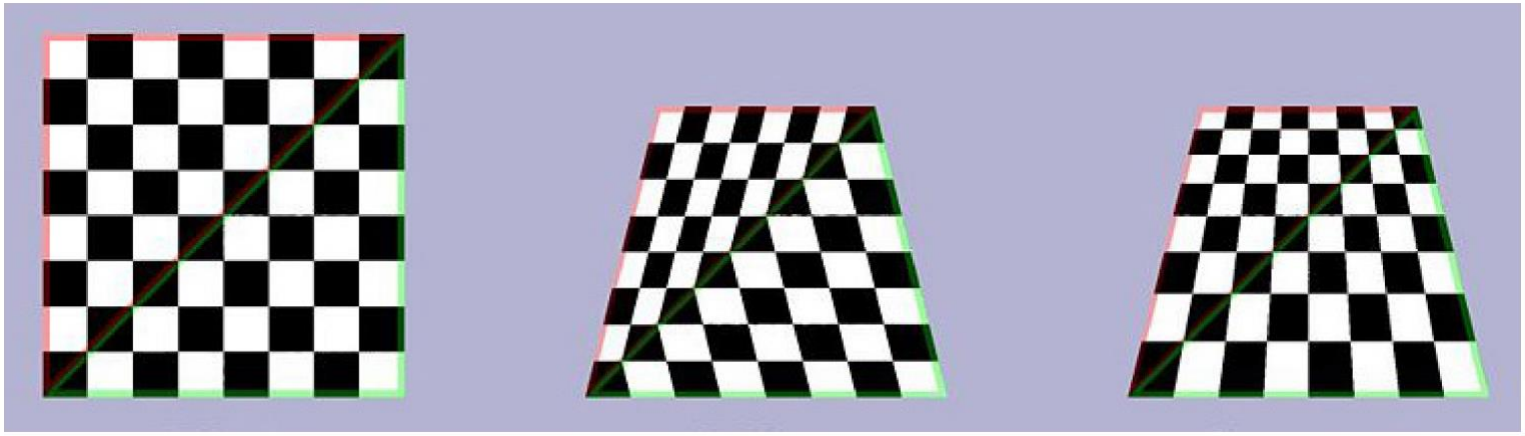
- Rasterisierungsstufe führt Interpolation der Texturkoordinaten durch.

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Zuordnung von Texturkoordinaten zu Vertices

- Perspektivische Projektion: unbefriedigenden Ergebnisse, da Texturkoordinaten nach der Projektion interpoliert werden.
- Lösung: Perspektivkorrektur
  - Statt s und t: $\frac{s}{z}, \frac{t}{z}$ sowie $\frac{1}{z}$ linear interpolieren (z = Koordinate in Sichtrichtung)
- Um für ein Pixel die Texturkoordinate zu berechnen gilt nun:

$$s' = {}^{\frac{s}{z}}\Big/_{\frac{1}{z}} \quad \text{bzw.} \quad t' = {}^{\frac{t}{z}}\Big/_{\frac{1}{z}}$$



https://en.wikipedia.org/wiki/Texture_mapping

Zuordnung von Texturkoordinaten zu Vertices

- Ungleiche Seitenverhältnisse von Textur und Polygon:
  - Verzerrung der Textur (b)
  - Ausschnitt aus der Textur ohne Verzerrung (c)
  - Wiederholung der Textur ohne Verzerrung (d)

(a)

(b)

(c)

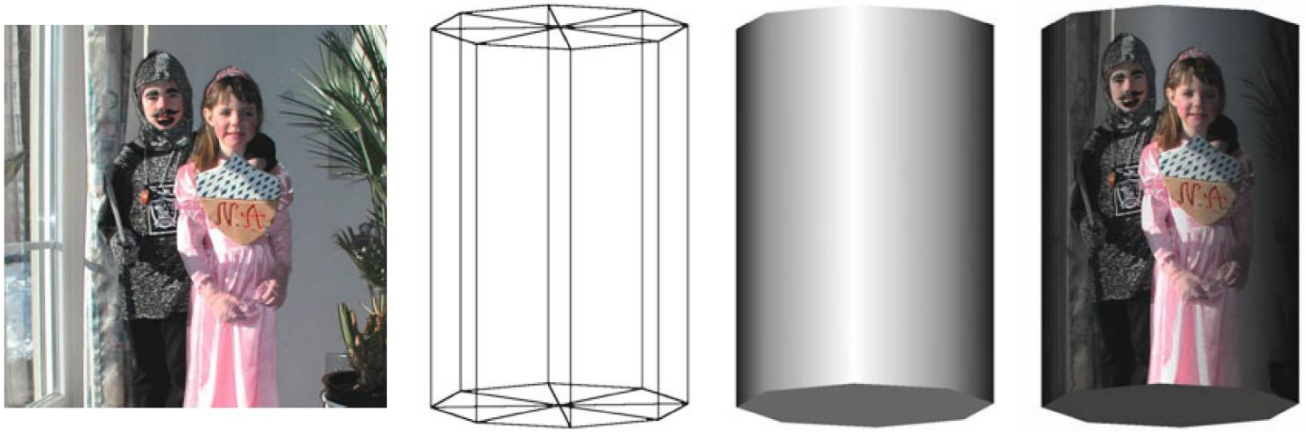
(d)

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Zuordnung von Texturkoordinaten zu Vertices

- Beispiel: Auftragen einer Textur auf Manteloberfläche eines Zylinders
  - Grundidee: Abwicklung der Mantelfläche ergibt Rechteck

```
glBegin(GL_QUAD_STRIP);
   for(phi=0; phi<2*3.1415; phi+=3.1415/4) {
      x = cos(phi);
      y = sin(phi);
      s = phi/(2*3.1415);
      glNormal3f(x,y,0); glTexCoord2f(s,0); glVertex3f(r*x,r*y,0);
      glNormal3f(x,y,0); glTexCoord2f(s,1); glVertex3f(r*x,r*y,h);
   }
glEnd();
```

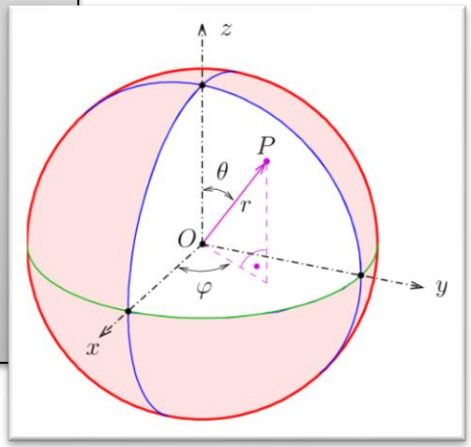


A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Zuordnung von Texturkoordinaten zu Vertices

- Beispiel: Auftragen einer Textur auf Kugeloberfläche

```
for(theta=3.1416/2; theta>-3.1416/2; theta-=3.1416/32) {
   thetaN = theta - 3.1416/32;
   glBegin(GL_QUAD_STRIP);
      for(phi=0; phi<2*3.1416; phi+=3.1416/32) {
         x = cos(phi) * cos(theta);
         y = sin(phi) * cos(theta);
         z = sin(theta);
         s = phi/(2*3.1416);
         t = theta/3.1416 + 0.5;
         glNormal3f(x,y,z); glTexCoord2f(s,t); glVertex3f(r*x,r*y,r*z);

         x = cos(phi) * cos(thetaN);
         y = sin(phi) * cos(thetaN);
         z = sin(thetaN);
         t = thetaN/3.1416 + 0.5;
         glNormal3f(x,y,z); glTexCoord2f(s,t); glVertex3f(r*x,r*y,r*z);
      }
   glEnd();
}
```



https://de.wikipedia.org/wiki/Kugel

Zuordnung von Texturkoordinaten zu Vertices

- Starke Verzerrung, insb. am Pol
- Verzerrungen abhängig von der Krümmung der Oberfläche

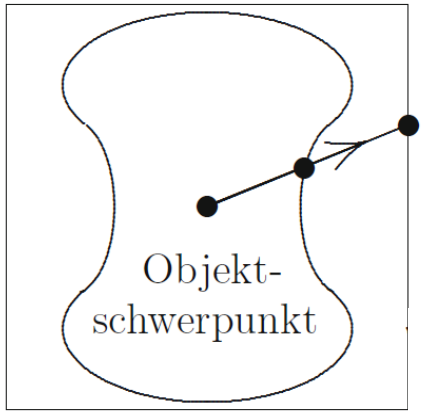
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Zuordnung von Texturkoordinaten zu Vertices

- Allgemeiner Fall: Aufbringen von Texturen auf Polygonnetze
- Zweiteiliges Abbilden:
  - S-Mapping: Textur auf analytisch beschreibbare Oberfläche, z.B. Ebene, aufbringen
  - O-Mapping: Texturkoordinaten der Zwischeneben auf eigentliches Objekt abbilden

- O-Mapping-Varianten:

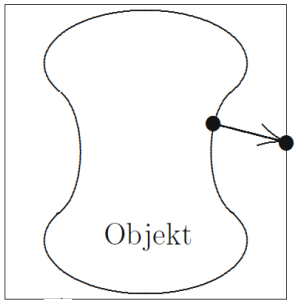
  (1) Strahl von Objektschwerpunkt durch Vertex der Oberfläche auf Zwischenebene



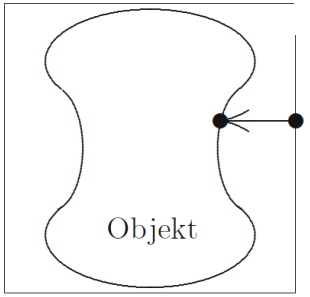
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Zuordnung von Texturkoordinaten zu Vertices

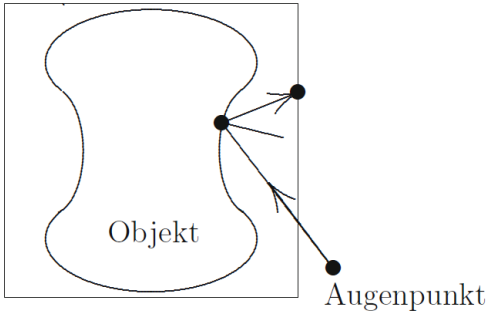
(2) Schnittpunkt von Vertex-Normalenvektor mit Zwischenebene



(3) Schnittpunkt des Zwischenebenen-Normalenvektors am Oberflächen-Vertex mit Zwischenebene



(4) Schnittpunkt der Zwischenebene mit der idealen Reflexion eines Strahls an der Oberfläche, der vom Augpunkt ausgeht.



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Zuordnung von Texturkoordinaten zu Vertices

- OpenGL erlaubt eine automatische Texturkoordinatengenerierung
  - Beschreibung einer Zwischenebene in Abhängigkeit der Vertices in Objekt- oder Weltkoordinaten (S-Mapping)

    `glTexGen*(coord, name, param)`

    mit `name` = GL_TEXTURE_GEN_MODE
- `param` = **`GL_OBJECT_LINEAR`**: feste Verbindung von Textur und Objekt

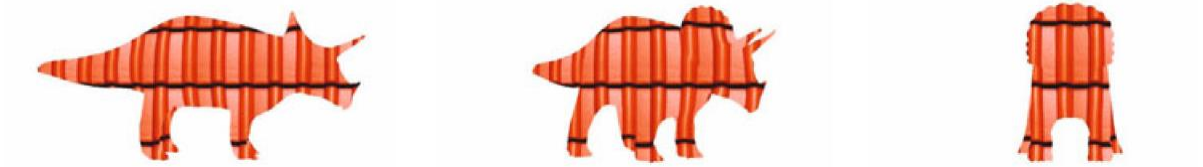
$$s(x_o, y_o, z_o, w_o) = ax_o + by_o + cz_o + dw_o$$

Objektkoordinaten

- `param` = **`GL_EYE_LINEAR`**: feste Verbindung von Textur und Augpunkt

$$s(x_e, y_e, z_e, w_e) = ax_e + by_e + cz_e + dw_e$$

Weltkoordinaten

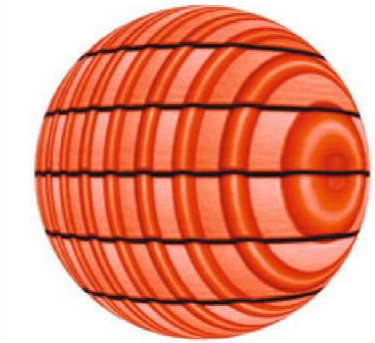


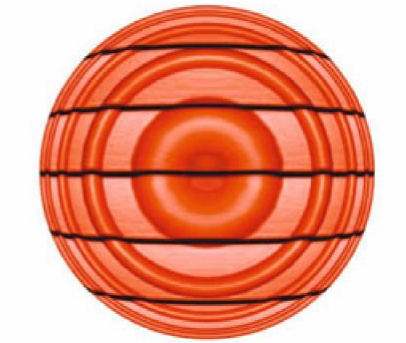
- Parameter a,b,c,d definieren Skalierung und Lage der Ebene (Koordinatenform)

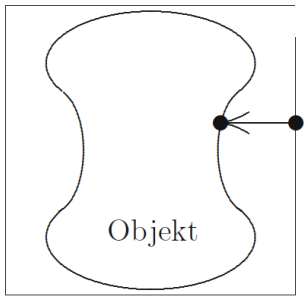
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Zuordnung von Texturkoordinaten zu Vertices

- Ebenengleichung definiert Mapping der Texturkoordinaten durch orthographische Projektion auf Vertices eines Objektes (O-Mapping)





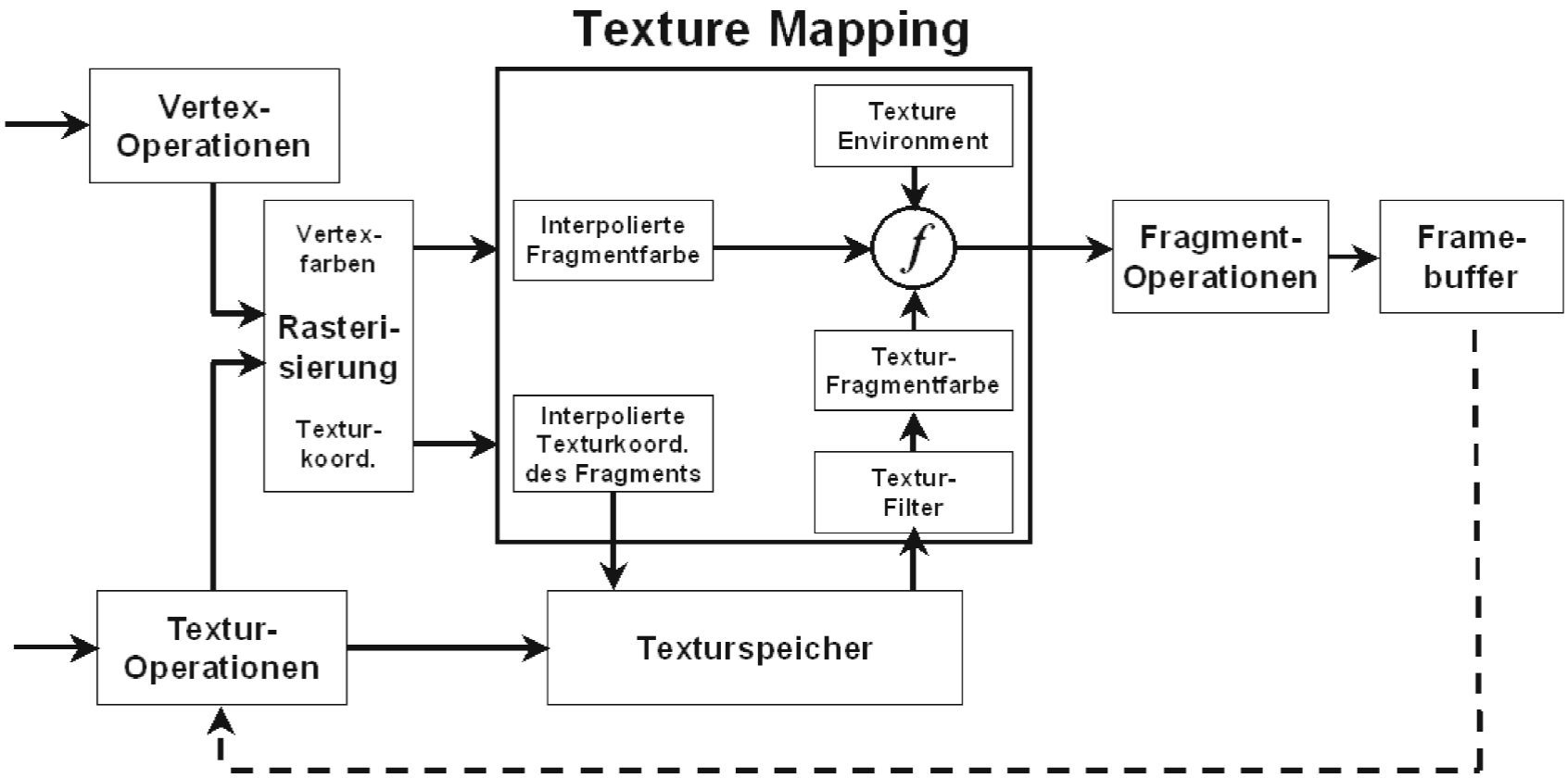


- Beispiel: Angabe der Parameter der Ebenengleichung

```
GLfloat s_plane[] = (a,b,c,d);
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_TEXTURE_OBJECT_PLANE, s_plane);
glEnable(GL_TEXTURE_GEN_S);
```

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

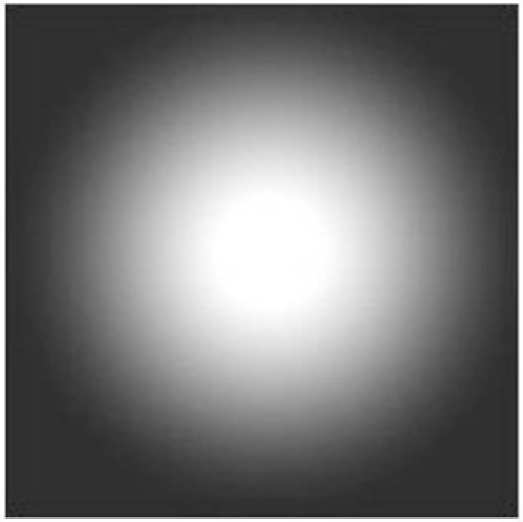
Texture Mapping: Ablauf



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Multitexturing

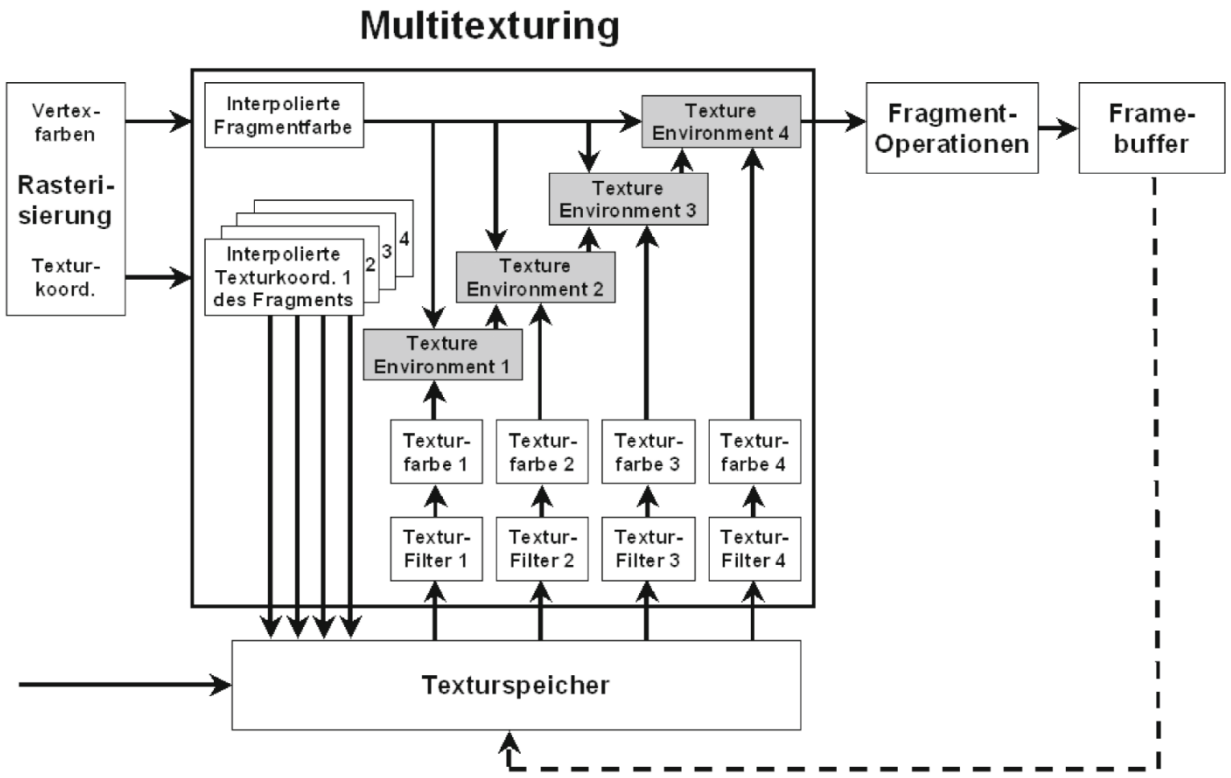
- Bisher: Aufbringen einer Textur auf ein Polygon
- Noch mehr Möglichkeiten durch Aufbringen von mehreren Texturen in einem Rendering-Durchlauf
  - Z.B. Lightmaps



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Multitexturing

- Mathematisch: Kombination von zwei oder mehr Texturen durch Interpolation + Multiplikation/Addition
- In OpenGL: üblicherweise 8 oder 16 Texturen möglich

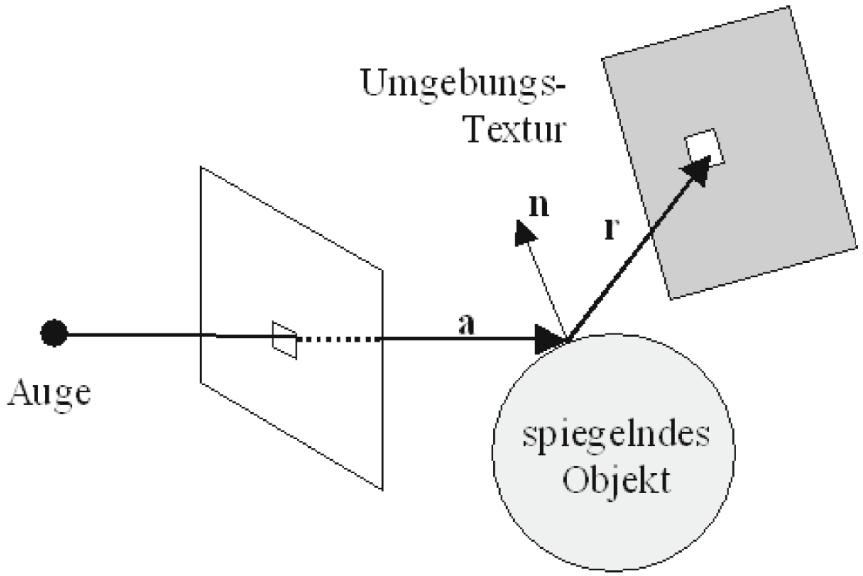


A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

8.2. ENVIRONMENT MAPPING

Environment Mapping

- Approximation von Spiegelungen einer umgebenden Textur auf Objekt



- Strahlverfolgung vom Augpunkt zu spiegelndem Objekt
- Bestimmung des Reflexionsvektors
- Texturkoordinaten in Umgebungstextur bestimmen

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Environment Mapping

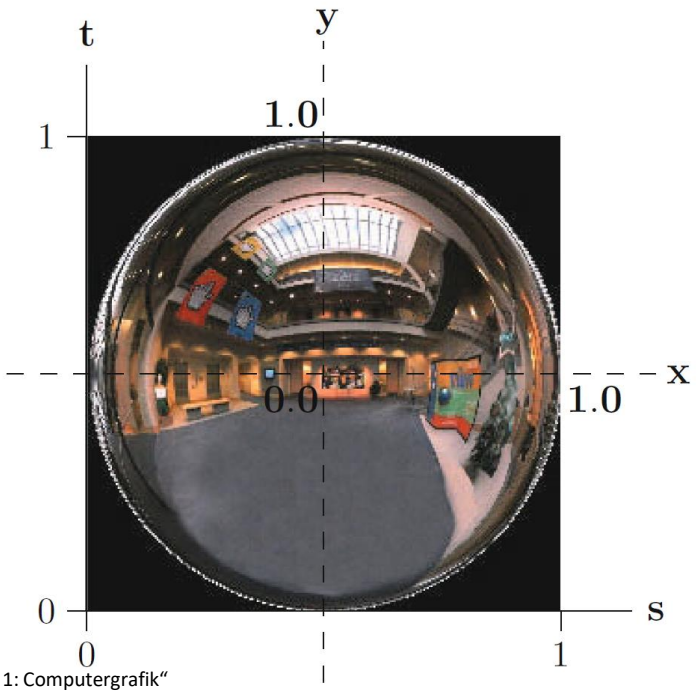
- Berechnung der Texturkoordinate:
  - Pro Vertex:
    - Berechnung des Reflexionsvektors und Texturkoordinaten nur an Vertices
    - Pixelbezogene Interpolation der Texturkoordinaten
  - Pro Pixel:
    - Berechnung des Reflexionsvektors und Texturkoordinaten für jedes Pixel

- In der interaktiven CG zwei übliche Verfahren:
  - Sphärische Texturierung (Sphere Mapping)
  - Kubische Texturierung (Cube Mapping)

Environment Mapping: Sphere Mapping

- Grundidee: Textur entspricht…
  - … orthografischer Projektion einer ideal verspiegelten Kugel
  - … Spiegelung der Kugel-Umgebung in der Kugel

- Kugel deckt kreisförmigen Bereich mit Mittelpunkt $(0{,}5 \,/\, 0{,}5)$ und Radius $0{,}5$ einer quadratischen Textur ab



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Environment Mapping: Sphere Mapping

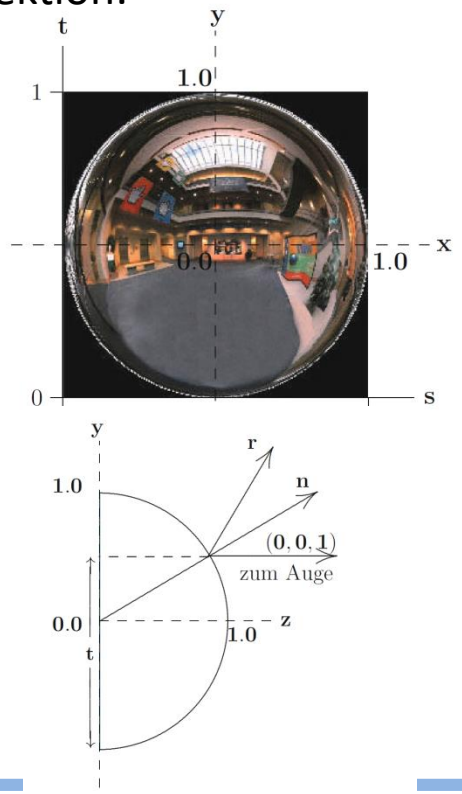
- Gesucht: Abbildung der Richtung des Reflexionsvektors auf einen Punkt in der sphärischen Textur

- Zusammenhang zwischen (planaren) Texturkoordinaten (s, t) und Punkt (x, y, z) auf der (sphärischen) Einheitskugel durch orthographische Projektion:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2s - 1 \\ 2t - 1 \\ \sqrt{1 - x^2 - y^2} \end{pmatrix}$$

- Einheitsnormalenvektor: für Einheitskugel durch die Koordinaten (x, y, z) gegeben:

$$n = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2s - 1 \\ 2t - 1 \\ \sqrt{1 - x^2 - y^2} \end{pmatrix}$$



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Environment Mapping: Sphere Mapping

Zusammenhang zwischen Reflexionsvektor und Normale der Objekt-Oberfläche:

- Reflexionsvektor r berechnet sich mit Hilfe des Reflexionsgesetzes:

$$r = a - 2(a \cdot n) \cdot n$$

  mit a = Vektor von Augpunkt zu Oberflächenpunkt

- a verläuft bei orthographischer Projektion entlang der negativen z-Achse: $a = (0,0,-1)$. Eingesetzt ergibt sich:

$$\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} - 2\left(\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}\right) \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

$$= \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} + 2n_z \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

- Aufgelöst nach dem Normalvektor und normiert ergibt sich:

$$n = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \frac{1}{\sqrt{{r_x}^2 + {r_y}^2 + (r_z+1)^2}} \begin{pmatrix} r_x \\ r_y \\ r_z + 1 \end{pmatrix}$$

Environment Mapping: Sphere Mapping

- Setzt man nun beide Normalen gleich und löst nach s bzw. t auf, ergeben sich die gesuchten Texturkoordinaten in Abhängigkeit vom Reflexionsvektor r (Koordinatentransformation):

$$\begin{pmatrix} 2s-1 \\ 2t-1 \\ \sqrt{1-x^2-y^2} \end{pmatrix} = \frac{1}{\sqrt{{r_x}^2+{r_y}^2+(r_z+1)^2}} \begin{pmatrix} r_x \\ r_y \\ r_z+1 \end{pmatrix}$$

$$s = \frac{r_x}{2\sqrt{{r_x}^2+{r_y}^2+(r_z+1)^2}} + \frac{1}{2}$$

$$t = \frac{r_y}{2\sqrt{{r_x}^2+{r_y}^2+(r_z+1)^2}} + \frac{1}{2}$$

- OpenGL übernimmt diese Berechnung bei Verwendung der automatischen Texturkoordinatengenerierung:

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
```

Environment Mapping: Sphere Mapping

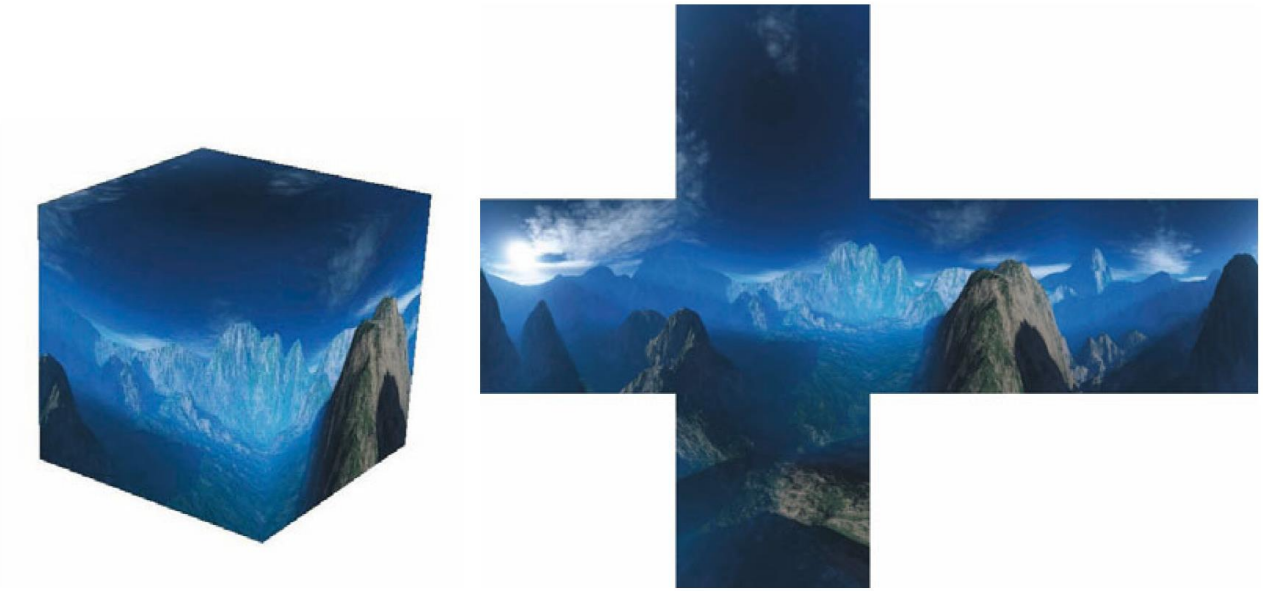
- Beeindruckende Ergebnisse bei vergleichsweise geringem Rechenaufwand

- Einschränkungen:
  - Probleme bei konkaven Objekten: keine Eigenspiegelung möglich!
  - Gilt nur für einen Blickwinkel

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Environment Mapping: Cube Mapping

- Umgebungstextur auf 6 Flächen eines Kubus
  - Gewinnung durch Fotografie/Rendern mit Öffnungswinkel 90° in alle Richtungen
  - Nahtloser Übergang an den Kanten des Kubus
- Zu texturierendes Objekt befindet sich in der Mitte des Kubus

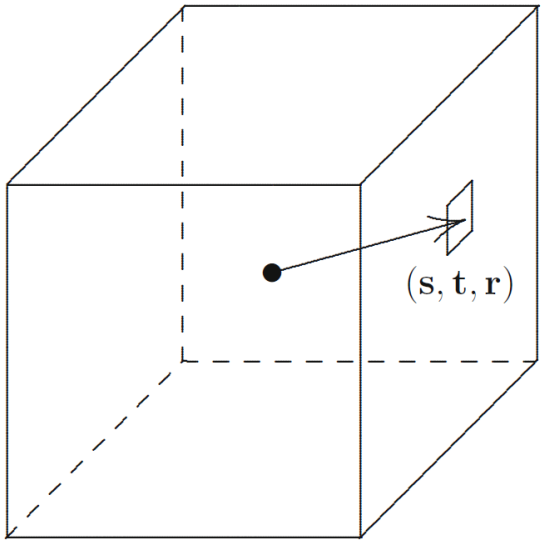


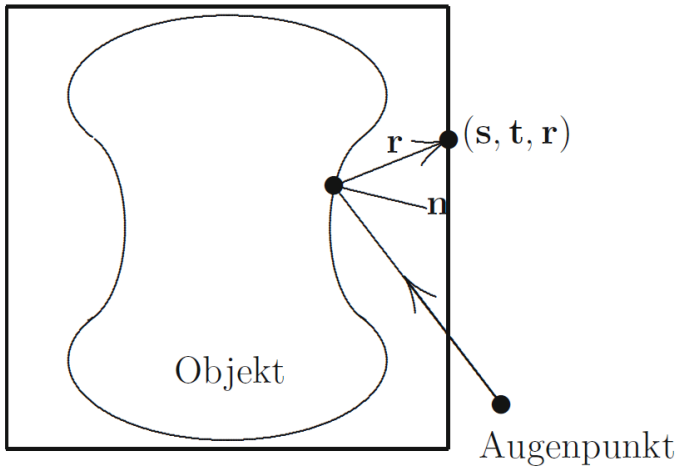
A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Environment Mapping: Cube Mapping

- Bestimmung der Texturkoordinaten:
  - Reflexionsvektor dient als Richtungsvektor im Mittelpunkt des Kubus
  - Koordinate des Reflexionsvektors mit größtem Absolutwert wählt Fläche des Kubus
  - Verbleibende Koordinaten werden bestimmt durch (Beispiel für Fläche y):

$$x = \frac{1}{2}\left(\frac{r_x}{r_y}\right) + 0{,}5 = s \qquad z = \frac{1}{2}\left(\frac{r_z}{r_y}\right) + 0{,}5 = t$$





A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Environment Mapping: Cube Mapping

- Bei Drehung des Augpunktes um das Objekt: inversen rotatorischen Anteil der Augpunkttransformation auf automatisch generierte Texturkoordinaten anwenden (=Drehung des Texturkubus)
- Skybox: Texturierter Kubus der gesamte Szene umschließt

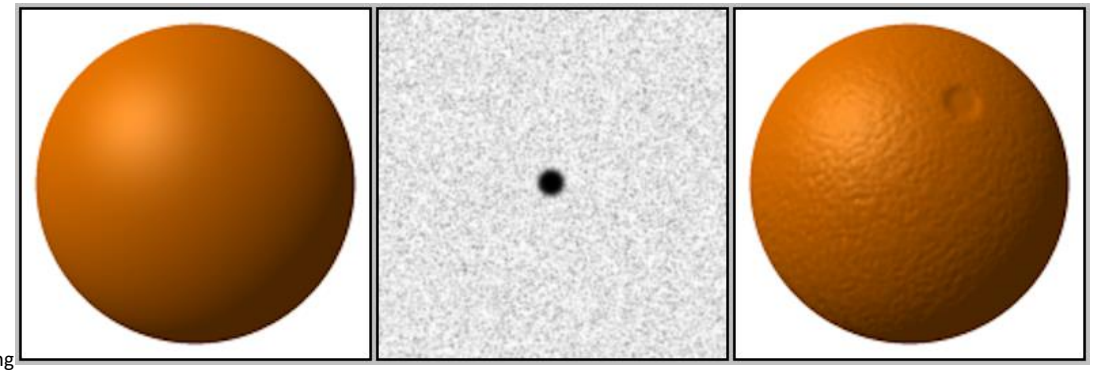
- Vorteile Cube Mapping:
  - Blickwinkelunabhängig
  - Texturen einfacher zu generieren
  - Geringere Verzerrungen

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

8.3. BUMP MAPPING

Bump Mapping

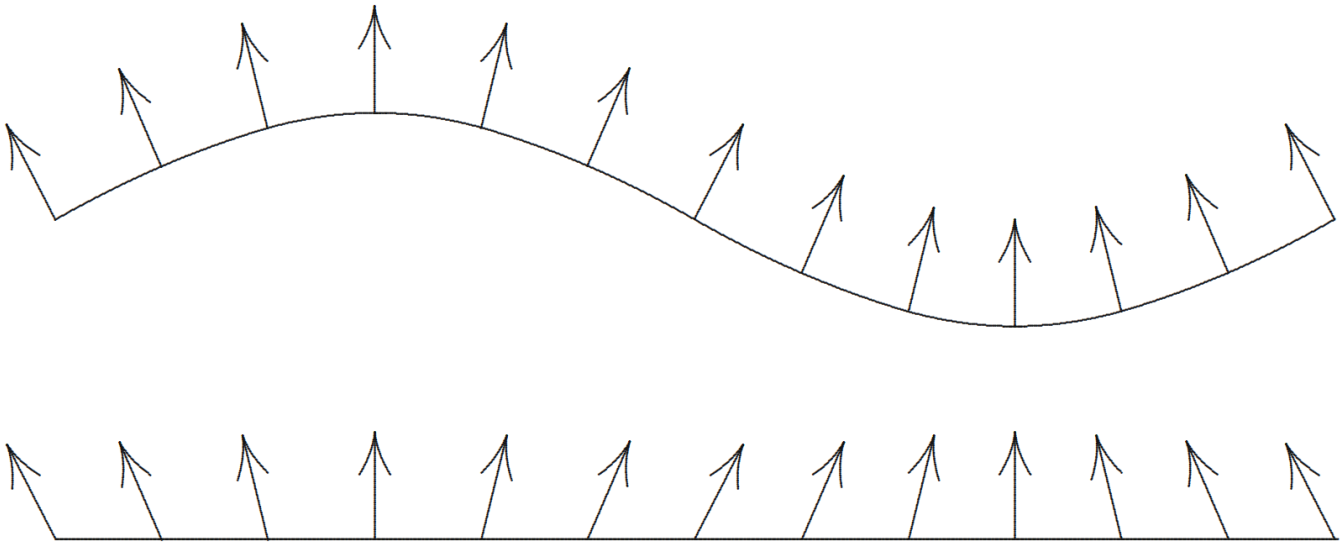
- Nachbildung einer rauen/reliefartigen Oberfläche ohne Änderung der Objektgeometrie
- Mit bisherigen Verfahren: Auftrag eine Fototextur
  - Nur für statische Szenen verwendbar, da Fototextur nur aus einem Blickwinkel aufgenommen wurde
- Abhilfe schafft das Bump Mapping (J.F. Blinn, 1978)
- Lokale Veränderung der Oberflächennormalen zur Nachahmung eines Reliefs + Pixelbezogene Beleuchtungsrechnung (Phong-Shading)
  - Bump Map: Werte geben Modifikation der Normalen an
  - Normal Map: Gibt modifizierte Normalen für jedes Pixel direkt an



https://de.wikipedia.org/wiki/Bumpmapping

Bump Mapping

- Grundprinzip ähnlich dem (Phong-)Shading:
  - Shading: Interpolation der Normalen an Vertices/Pixeln zur virtuellen Krümmung einer Oberfläche (Ziel: glatte Übergänge)



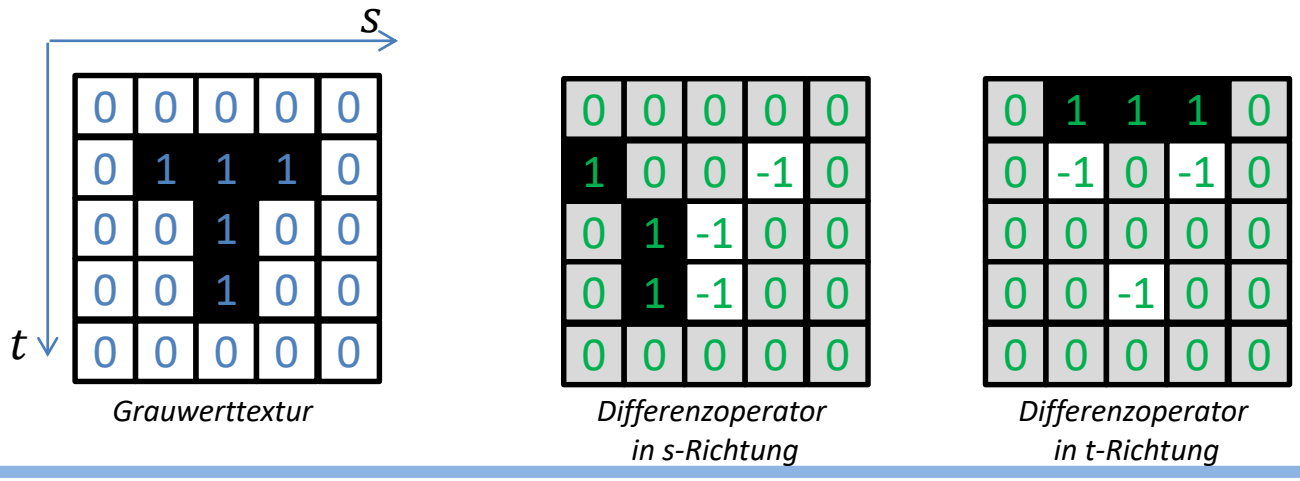
  - Bump Mapping: Modifikation der Normalen pro Pixel zur virtuellen Oberflächenkrümmung mit hoher Frequenz

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Bump Mapping für ebene Rechtecke

- Voraussetzung:
  - Vorliegen einer Grauwert-Textur (oder Generierung einer Grauwerttextur aus RGB-Textur durch geeignete Operation, z.B. Mittelung)
- Vorgehen zur Erstellung einer Normal Map
  1. Berechnung der Gradienten (Ableitung) der Grauwert-Textur in Richtung der s- und der t-Texturkoordinaten für jedes Texel → z-Komponente eines Gradientenvektors

z.B. durch Differenzoperator: $g_s(s,t) = \begin{pmatrix} 1 \\ 0 \\ g(s+1,t) - g(s,t) \end{pmatrix}$, $g_t(s,t) = \begin{pmatrix} 0 \\ 1 \\ g(s,t+1) - g(s,t) \end{pmatrix}$



Bump Mapping für ebene Rechtecke

2. Berechnen des Kreuzproduktes der beiden Gradientenvektoren g_s und g_t um einen Normalenvektor zu erhalten

$$n = \begin{pmatrix} 1 \\ 0 \\ g(s+1,t) - g(s,t) \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ g(s,t+1) - g(s,t) \end{pmatrix} = \begin{pmatrix} g(s,t) - g(s+1,t) \\ g(s,t) - g(s,t+1) \\ 1 \end{pmatrix}$$

3. Normieren des Normalenvektors → Einheitsnormalenvektor (Wertebereich [-1 1]), Länge 1

$$n = \frac{1}{\sqrt{\big(g(s,t) - g(s+1,t)\big)^2 + \big(g(s,t) - g(s,t+1)\big)^2 + 1}} \begin{pmatrix} g(s,t) - g(s+1,t) \\ g(s,t) - g(s,t+1) \\ 1 \end{pmatrix}$$

4. Transformieren des Einheitsnormalenvektors in den Wertebereich $[0,1]$ zur Kodierung der x-,y-,z-Koordinate in einer RGB-Textur

$$\boldsymbol{n} = \frac{1}{2}(n+1)$$

Bump Mapping für ebene Rechtecke

Originaltextur

Rot-Kanal
$\boldsymbol{n}_x$-Komponente

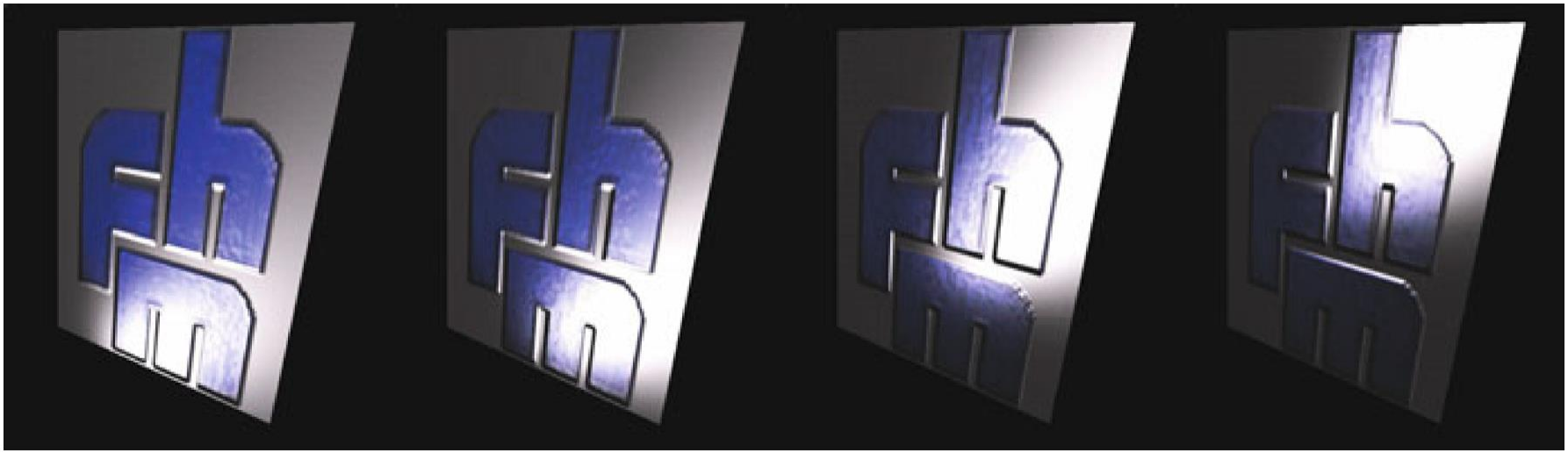
Grün-Kanal
$\boldsymbol{n}_y$-Komponente

Blau-Kanal
$\boldsymbol{n}_z$-Komponente

A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Bump Mapping für ebene Rechtecke

- Vorgehen zum Mapping:
  - Erstellte RGB-Textur + Phong-Shading nutzen
  - Statt linearer Interpolation der Normalen an jedem Pixel (=Phong Shading) können die Normalen aus der RGB-Textur direkt für die Beleuchtungsberechnung verwendet werden.



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

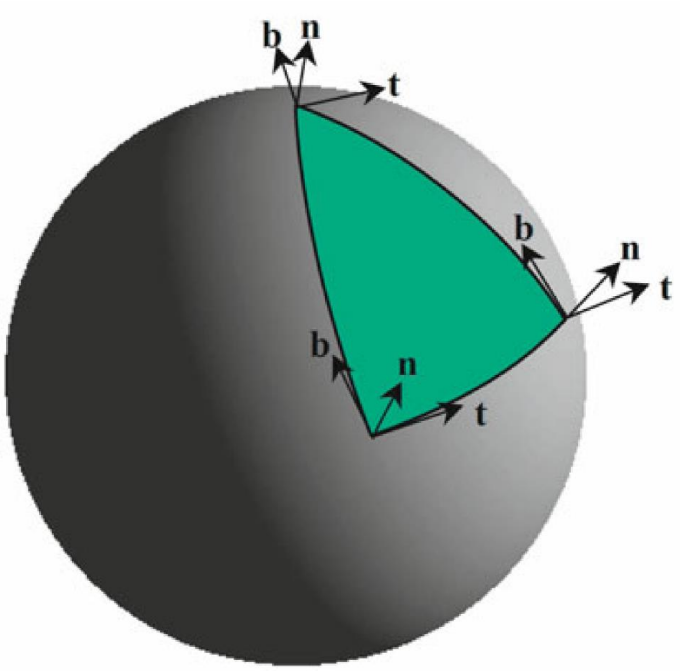
Bump Mapping

- Normalenvektoren des Rechtecks, auf die die Textur aufgebracht wird, sind einheitlich, z.B. $(0,0,1)$ für alle Pixel auf dem Rechteck.
- Bei Drehung, Verschiebung werden die Bump Mapping Normalen entsprechend mit der Transformationsmatrix gedreht bzw. verschoben

- Gekrümmte Objekte:
  - Bump Mapping Normalen können nicht verwendet werden, da sie für eine ebene Fläche (= ursprüngliche Textur) bestimmt wurden.
  - Gekrümmte Struktur aufwendig, nicht allgemein wiederverwendbar

Bump Mapping für gekrümmte Oberflächen

- Abhilfe schafft eine Koordinatensystemtransformation:
  - Für jede (interpolierte) Oberflächennormale n ist eine Tangentialebene definiert: aufgespannt durch Tangentialvektor t und Binormalenvektor b.
  - t, b und n definieren ein oberflächenlokales Koordinatensystem (=orthonormale Basis)



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Bump Mapping für gekrümmte Oberflächen

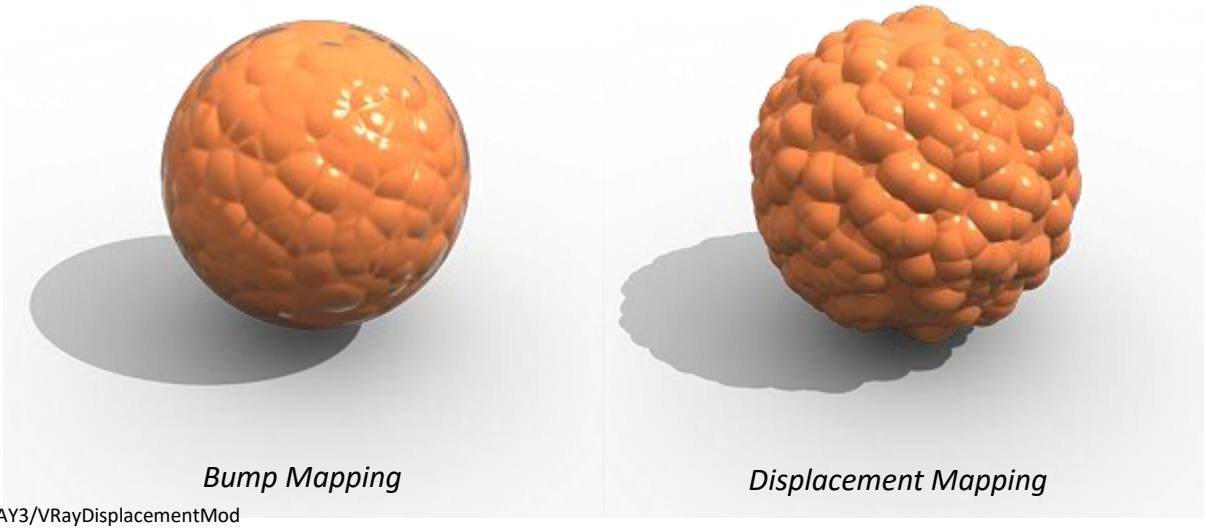
- Multiplikation mit der TBN-Matrix basierend auf der orthonormalen Basis transformiert den Normalenvektor aus der Normal-Map in das Weltkoordinatensystem:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Tangentenvektor t sollte konsistent gewählt werden um Ecken in der Textur zu vermeiden.
- t wird üblicherweise so gewählt dass er der s-Achse der Textur folgt.
- Binormalenvektor b steht senkrecht auf t und n → Kreuzprodukt

Displacement Mapping

- Verschiebung der Oberflächenpunkte eines Objektes auf Basis der Höhenkarte entlang der Oberflächennormalen
  - Senkrecht zur Oberfläche
  - Verschieben der Punkte verändert die Normalen der neuen Oberfläche
  - In Abhängigkeit von der Feinheit des Polygonnetzes kann eine Verfeinerung notwendig werden (= Tesselation)
- Vorteil gegenüber Bump Mapping:
  - Tatsächliche Änderung der Geometrie



Bump Mapping *Displacement Mapping*

http://docs.chaosgroup.com/display/VRAY3/VRayDisplacementMod

ZUSAMMENFASSUNG

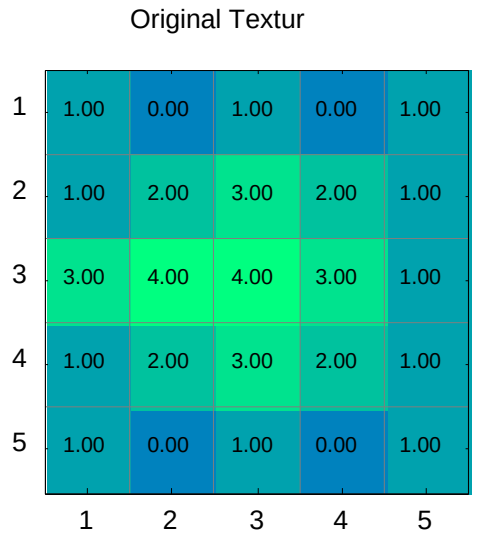
Zusammenfassung

- Texturen sind ein einfaches Mittel um Fotorealismus zu erlangen
- Texture Mapping: „Fototapete“
  - Spezifikation der Textur
  - Festlegung wie die Textur auf jedes Pixel aufgetragen wird
    - Texturfilter, Mipmapping, Texture Wraps
    - Mischung von Textur und Beleuchtungsfarbe
  - Zuordnung von Texturkoordinaten zu Vertices
  - Multitexturing
- Environment-/Shadow-Mapping: Approximation globaler Beleuchtung
  - Approximation von Spiegelungen einer umgebenden Textur auf Objekt
  - Sphere Mapping oder Cube Mapping
- Bump Mapping: Reliefartige Darstellung
  - Lokale Veränderung der Oberflächennormalen zur Nachahmung eines Reliefs + Pixelbezogene Beleuchtungsrechnung (Phong)

ÜBUNGSAUFGABEN

Übungsaufgaben

1. Gegeben ist folgende Textur. Bestimmen Sie die Ableitung der Textur in s und t-Richtung. Verwenden Sie an den Rändern ein Padding mit 0
2. Generieren Sie aus der Textur eine Normal Map, die für das Bump Mapping eingesetzt werden kann.



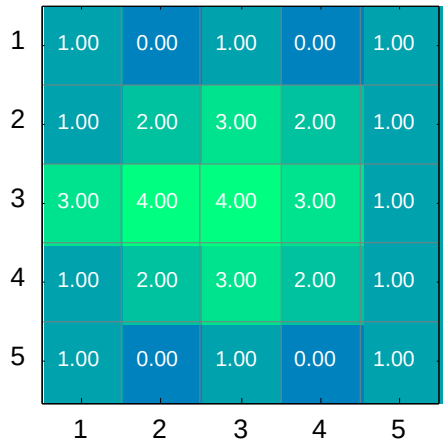
Lösung

1) Gegeben ist folgende Textur. Bestimmen Sie die Ableitung der Textur in s und t-Richtung. Verwenden Sie an den Rändern ein Padding mit 0.

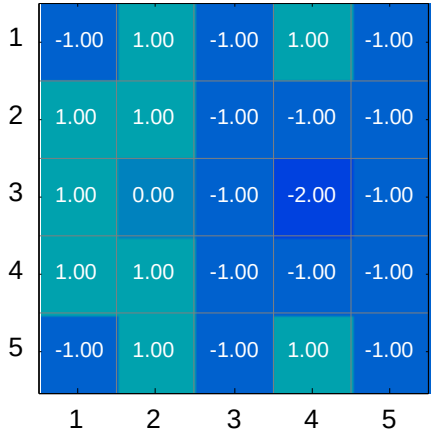
Vorgehen: Pixelweise entlang der s- (horizontal) bzw. t-Achse (vertikal) Subtraktion der Einträge bei $(s+1)$ und s. Beispielhaft für Ableitung in s-Richtung bei $(1,1)$:

$$g(s+1,t) - g(s,t) = 0 - 1 = -1$$

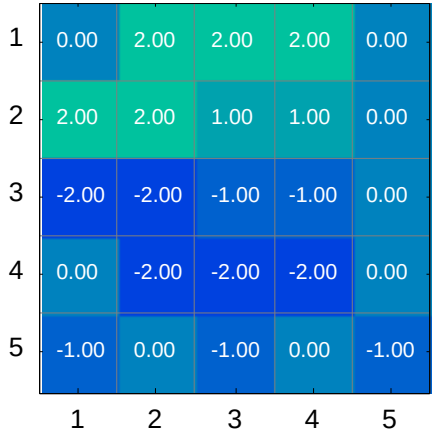
Original Textur



Ableitung s



Ableitung t



Lösung (2)

2) Generieren Sie aus der Textur eine Normal Map, die für das Bump Mapping eingesetzt werden kann.

Berechnung des Kreuzproduktes direkt aus der Textur:

$$n = \begin{pmatrix} 1 \\ 0 \\ g(s+1,t) - g(s,t) \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \\ g(s,t+1) - g(s,t) \end{pmatrix} = \begin{pmatrix} g(s,t) - g(s+1,t) \\ g(s,t) - g(s,t+1) \\ 1 \end{pmatrix}$$

→ Pixelweise Berechnung, z.B. Eintrag in Normal Map x bei 1,1: $g(s,t) - g(s+1,t) = g(1,1) - g(2,1) = 1 - 0 = 1$ wobei g der Originaltextur entspricht. Alternativ: Invertierung der bereits berechneten Ableitung bzw. Kreuzprodukt aus den berechneten Ableitungen

Normal Map x

	1	2	3	4	5
1	1.00	-1.00	1.00	-1.00	1.00
2	-1.00	-1.00	1.00	1.00	1.00
3	-1.00	0.00	1.00	2.00	1.00
4	-1.00	-1.00	1.00	1.00	1.00
5	1.00	-1.00	1.00	-1.00	1.00

Normal Map y

	1	2	3	4	5
1	0.00	-2.00	-2.00	-2.00	0.00
2	-2.00	-2.00	-1.00	-1.00	0.00
3	2.00	2.00	1.00	1.00	0.00
4	0.00	2.00	2.00	2.00	0.00
5	1.00	0.00	1.00	0.00	1.00

Normal Map z

	1	2	3	4	5
1	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	1.00

Lösung (3)

Normalisierung:

$$n = \frac{1}{\sqrt{\big(g(s,t) - g(s+1,t)\big)^2 + \big(g(s,t) - g(s,t+1)\big)^2 + 1}} \begin{pmatrix} g(s,t) - g(s+1,t) \\ g(s,t) - g(s,t+1) \\ 1 \end{pmatrix}$$

Beispiel für Normal Map x bei (1,1): $n_x = \frac{1}{\sqrt{1^2+0^2+1}} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = 0{,}71$

Normal Map x (normalisiert)

	1	2	3	4	5
1	0.71	-0.41	0.41	-0.41	0.71
2	-0.41	-0.41	0.58	0.58	0.71
3	-0.41	0.00	0.58	0.82	0.71
4	-0.71	-0.41	0.41	0.41	0.71
5	0.58	-0.71	0.58	-0.71	0.58

Normal Map y (normalisiert)

	1	2	3	4	5
1	0.00	-0.82	-0.82	-0.82	0.00
2	-0.82	-0.82	-0.58	-0.58	0.00
3	0.82	0.89	0.58	0.41	0.00
4	0.00	0.82	0.82	0.82	0.00
5	0.58	0.00	0.58	0.00	0.58

Normal Map z (normalisiert)

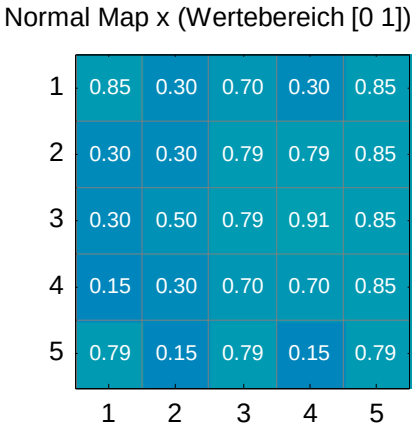
	1	2	3	4	5
1	0.71	0.41	0.41	0.41	0.71
2	0.41	0.41	0.58	0.58	0.71
3	0.41	0.45	0.58	0.41	0.71
4	0.71	0.41	0.41	0.41	0.71
5	0.58	0.71	0.58	0.71	0.58

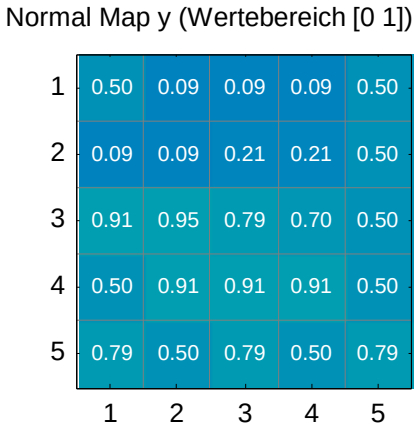
Lösung (4)

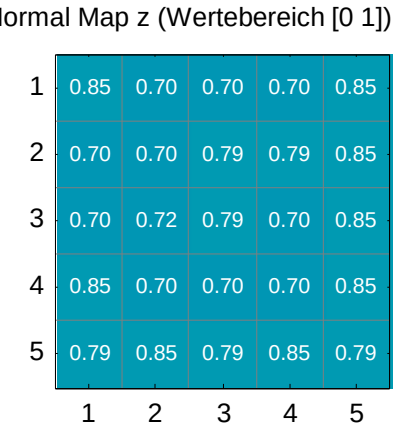
Transformieren des Einheitsnormalenvektors in den Wertebereich $[0,1]$: $t = \frac{1}{2}(n+1)$

Beispielhaft für den Eintrag in der Normal Map x bei (1,1)

$$t = \frac{1}{2}(0{,}71 + 1) = 0{,}85$$







Übungsaufgabe

3. Gegeben sind die folgenden Werte für die Farben eines Pixels der Texturpixel g_t, des Texturhintergrundes g_h und aus der Beleuchtungsberechnung g_f. Geben Sie die resultierende Pixelfarbe g_r für die Modulationsfunktionen **GL_REPLACE**, **GL_MODULATE**, **GL_DECAL**, **GL_BLEND** und **GL_ADD** an:

$$g_t = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0{,}5 \end{pmatrix}, \quad g_f = \begin{pmatrix} 0{,}5 \\ 0{,}5 \\ 0{,}5 \\ 1 \end{pmatrix}, \quad g_h = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Gehen Sie davon aus, dass Farbwerte im Intervall $[0{,}1]$ liegen und gegebenenfalls abgeschnitten werden.

Lösung

3. Gegeben sind die folgenden Werte für die Farben eines Pixels der Texturpixel g_t, des Texturhintergrundes g_h und aus der Beleuchtungsberechnung g_f. Geben Sie die resultierenden Pixelfarbe g_r für die Modulationsfunktionen GL_REPLACE, GL_MODULATE, GL_DECAL, GL_BLEND und GL_ADD an:

$$g_t = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0,5 \end{pmatrix}, \quad g_f = \begin{pmatrix} 0,5 \\ 0,5 \\ 0,5 \\ 1 \end{pmatrix}, \quad g_h = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Gehen Sie davon aus, das Farbwerte im Intervall $[0, 1]$ liegen und gegebenenfalls abgeschnitten werden.

GL_REPLACE: Die Pixelfarbe entspricht der Texturfarbe.

$$g_r = g_t = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0,5 \end{pmatrix}$$

GL_MODULATE:

$$g_r = \begin{pmatrix} R_t \cdot R_f \\ G_t \cdot G_f \\ B_t \cdot B_f \\ A_t \cdot A_f \end{pmatrix} = \begin{pmatrix} 1 \cdot 0,5 \\ 0 \cdot 0,5 \\ 0 \cdot 0,5 \\ 0,5 \cdot 1 \end{pmatrix} = \begin{pmatrix} 0,5 \\ 0 \\ 0 \\ 0,5 \end{pmatrix}$$

Lösung (2)

GL_DECAL:

$$g_r = g_r = \begin{pmatrix} (1-A_t)R_f + A_t R_t \\ (1-A_t)G_f + A_t G_t \\ (1-A_t)B_f + A_t B_t \\ A_f \end{pmatrix} = \begin{pmatrix} (1-0{,}5)\cdot 0{,}5 & +0{,}5\cdot 1 \\ (1-0{,}5)\cdot 0{,}5 & +0{,}5\cdot 0 \\ (1-0{,}5)\cdot 0{,}5 & +0{,}5\cdot 0 \\ 1 & \end{pmatrix} = \begin{pmatrix} 0{,}75 \\ 0{,}25 \\ 0{,}25 \\ 1 \end{pmatrix}$$

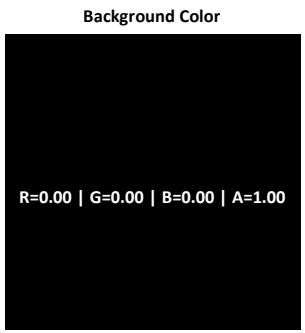
GL_BLEND:

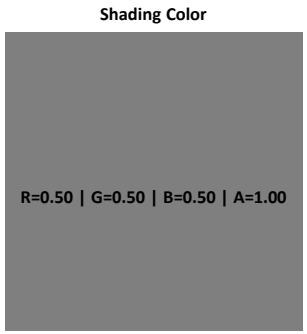
$$g_r = \begin{pmatrix} (1-R_t)R_f + R_t R_h \\ (1-G_t)G_f + G_t G_h \\ (1-B_t)B_f + B_t B_h \\ A_t \cdot A_f \end{pmatrix} = \begin{pmatrix} (1-1)0{,}5 & +1\cdot 1 \\ (1-0)0{,}5 & +0\cdot 1 \\ (1-0)0{,}5 & +0\cdot 1 \\ 0{,}5\cdot 1 & \end{pmatrix} = \begin{pmatrix} 1 \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \end{pmatrix}$$

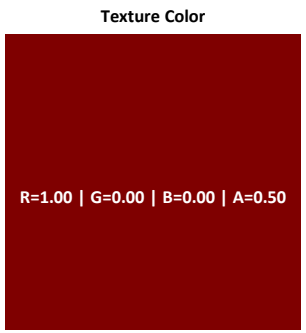
GL_ADD: hier wird der Wertebereich begrenzt durch Abschneiden der Werte über 1,0. Entspricht auch der OpenGL-Konvention.

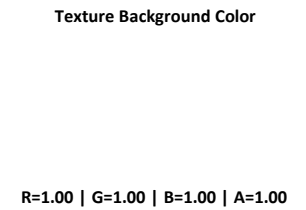
$$g_r = \begin{pmatrix} R_t + R_f \\ G_t + G_f \\ B_t + B_f \\ A_t \cdot A_f \end{pmatrix} = \begin{pmatrix} 1+0{,}5 \\ 0+0{,}5 \\ 0+0{,}5 \\ 0{,}5\cdot 1 \end{pmatrix} = \begin{pmatrix} \mathbf{1{,}5} \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 \\ 0{,}5 \\ 0{,}5 \\ 0{,}5 \end{pmatrix}$$

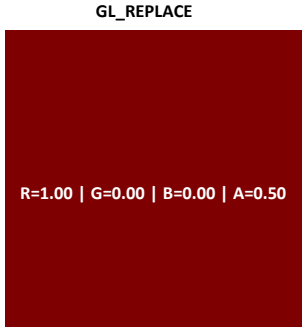
Lösung (3)

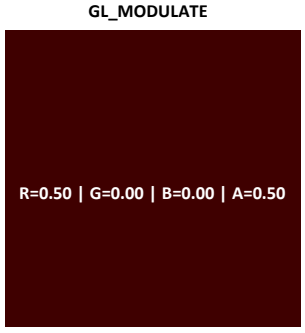

Background Color
R=0.00 | G=0.00 | B=0.00 | A=1.00

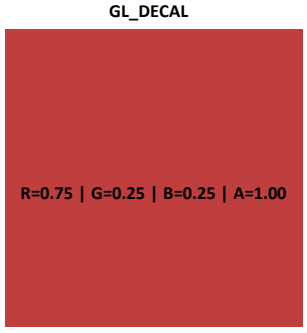

Shading Color
R=0.50 | G=0.50 | B=0.50 | A=1.00

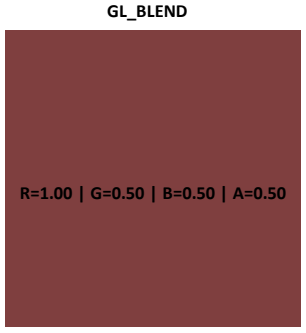

Texture Color
R=1.00 | G=0.00 | B=0.00 | A=0.50

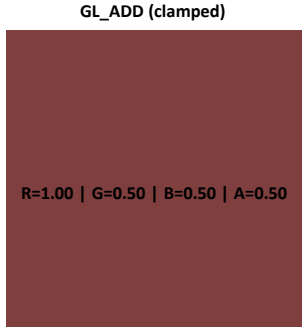

Texture Background Color
R=1.00 | G=1.00 | B=1.00 | A=1.00


GL_REPLACE
R=1.00 | G=0.00 | B=0.00 | A=0.50


GL_MODULATE
R=0.50 | G=0.00 | B=0.00 | A=0.50


GL_DECAL
R=0.75 | G=0.25 | B=0.25 | A=1.00


GL_BLEND
R=1.00 | G=0.50 | B=0.50 | A=0.50

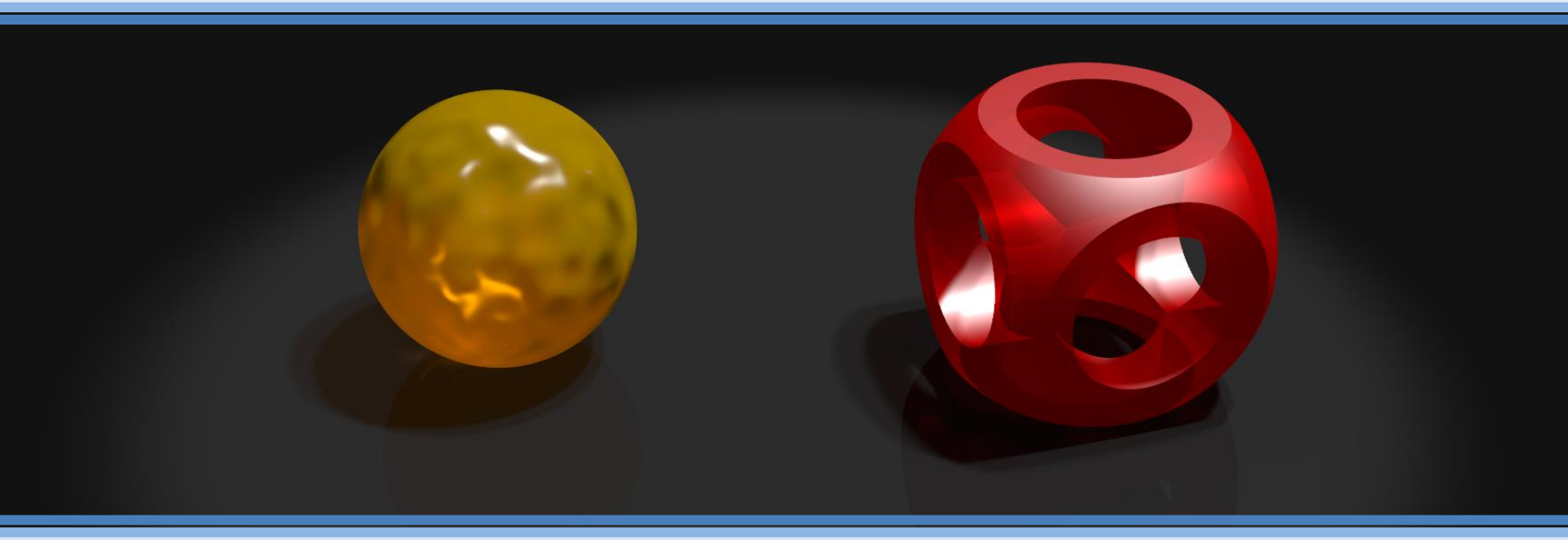

GL_ADD (clamped)
R=1.00 | G=0.50 | B=0.50 | A=0.50

Übungsfragen Kapitel 8

- Was ist Mipmapping?
- Was sind Texture Wraps?
- Warum ist bei der perspektivischen Projektion eine Perspektivkorrektur für das Texture Mapping notwendig?
- Erläutern Sie wie aus einer Grauwert-Textur eine Normal Map erstellt werden kann.
- Nennen und beschreiben Sie kurz drei O-Mapping-Varianten.
- Erläutern Sie kurz den Unterschied zwischen Bump Mapping und Displacement Mapping.

Computergrafik

T. Hopp



Themenübersicht

1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. **Animationen**
10. Raytracing
11. Volumenvisualisierung

Animation

- Jegliche Veränderung einer Szene mit der Zeit
- Häufigste Animationen
  - Bewegung des Augpunktes
  - ***Bewegung von Objekten***
  - Bewegung von Lichtquellen
  - ***Veränderung der Gestalt eines Objektes*** (Form, Farbe, Textur, …)

- Unterscheidung von Hierarchieebenen für Animationen:
  - **Pfadanimation**: Bewegung starrer Objekte entlang einer räumlichen Bahn
  - **Artikulation**: Bewegung innerer Freiheitsgrade eines Objektes (z.B. Gelenk)
  - **Morphing**: elastische oder plastische Verformung eines Objektes
  - **Partikelsysteme**: gleichartige Bewegung einer Objektgruppe

9.1. PFADANIMATION

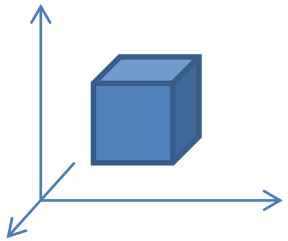
Pfadanimation

- Bewegung eines starren Objektes folgt einer Bahnkurve $\mathbf{K}$ im dreidimensionalen euklidischen Raum
- Darstellung über parametrische Funktion:

$$\mathbf{K}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}$$

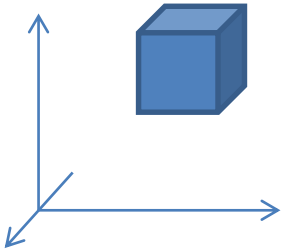
- Beispiel: geradlinige Bewegung entlang der x-Achse mit konstanter Geschwindigkeit v:

$$\mathbf{K}(t) = \begin{pmatrix} x(t) = v \cdot t \\ y(t) = 0 \\ z(t) = 0 \end{pmatrix}$$



- Beispiel: Kreisbewegung in der x-y-Ebene mit Radius R und konstanter Winkelgeschwindigkeit w

$$\mathbf{K}(t) = \begin{pmatrix} x(t) = R \cdot \cos(w \cdot t) \\ y(t) = R \cdot \sin(w \cdot t) \\ z(t) = 0 \end{pmatrix}$$



Pfadanimation

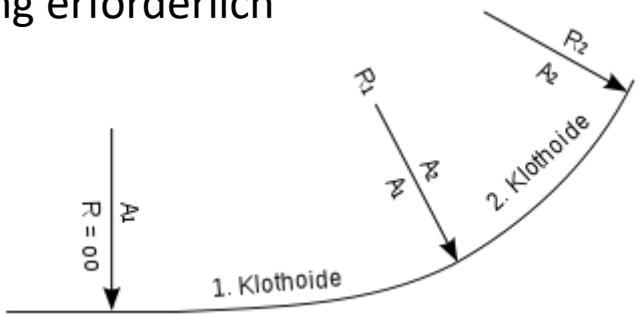
- Kinematische Betrachtung, keine Abbildung der physikalischen Realität
  - Z.B. Übergang von gerader Bewegung in Kurve durch langsam ansteigende Krümmung

- Mathematische Formulierung mit langsam ansteigender Krümmung: *Klothoide*

$$\mathbf{K}(t) = \begin{pmatrix} x(t) = a\sqrt{\pi}\int_0^t \cos\left(\frac{\pi u^2}{2}\right) du \\ y(t) = a\sqrt{\pi}\int_0^t \sin\left(\frac{\pi u^2}{2}\right) du \\ z(t) = 0 \end{pmatrix}$$

Laufvariable t in der Obergrenze des Integrals
→ Numerische Berechnung erforderlich

- Aufwändige Berechnung des Integrals!



Pfadanimation

- Bei komplexen Raumkurven meist Vorberechnung an diskreten Stellen
  - Interpolation (Lineare Interpolation, Splines) zwischen berechneten Stützstellen.

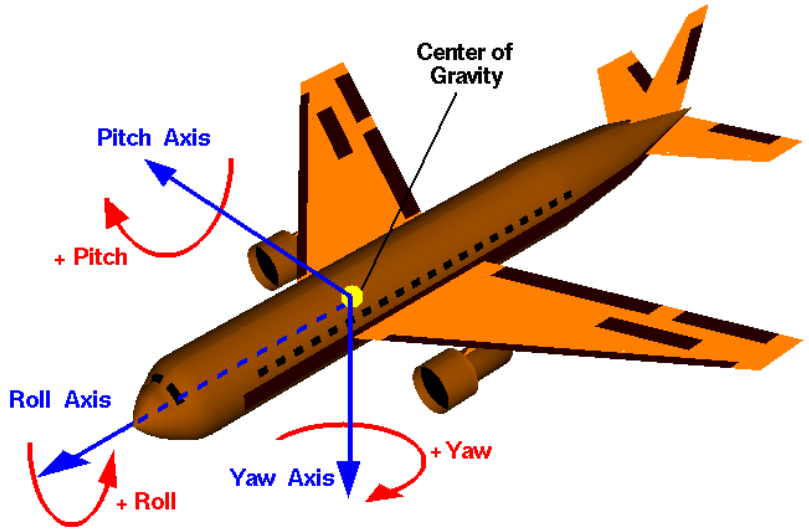
- Allgemein lässt sich dies in die *Key Frame* Technik überführen: Definition einer Bewegung durch n diskrete Abtastpunkte:

$$\mathbf{K} = \begin{pmatrix} x = (x_0, x_1, \dots, x_n) \\ y = (y_0, y_1, \dots, y_n) \\ z = (z_0, z_1, \dots, z_n) \end{pmatrix}$$

- Abspeicherung der Position der animierten Objekte zu diskreten Zeitpunkten.
- Zwischenwerte werden durch Interpolation gewonnen.

Pfadanimation

- Bisher nur Betrachtung von Translation
- Darüber hinaus üblicherweise Drehwinkel des Objekts um die Koordinatenachsen

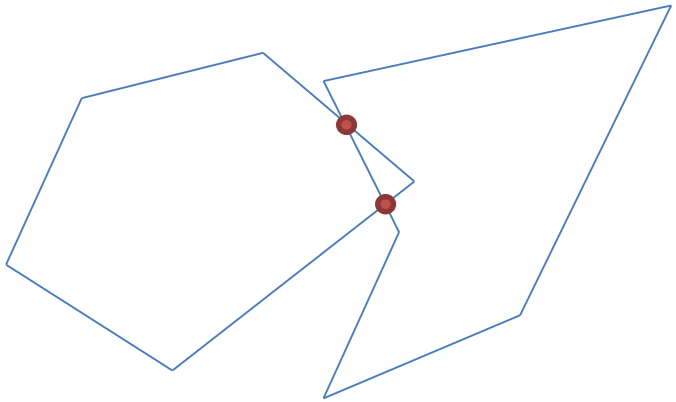


$$\mathbf{K} = \begin{pmatrix} x = (x_0, x_1, \ldots, x_n) \\ y = (y_0, y_1, \ldots, y_n) \\ z = (z_0, z_1, \ldots, z_n) \\ h = (h_0, h_1, \ldots, h_n) \\ p = (p_0, p_1, \ldots, p_n) \\ r = (r_0, r_1, \ldots, r_n) \end{pmatrix}$$

https://de.wikipedia.org/wiki/Roll-Nick-Gier-Winkel

Kollisionserkennung

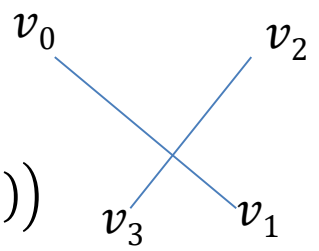
- Komplexität der Berechnung abhängig von Objektart und Komplexität der Szene bzw. Menge der Objekte
  - Analytische Lösungen für einfache Objekte, z.B. Schnittpunkte zwischen Geraden, zwischen Gerade und Kreis etc.
- Naiver Algorithmus zur Detektion einer Kollision zwischen zwei Polygonen:



  - Test auf Schnittpunkte aller Kanten miteinander

Kollisionserkennung

- Berechnung des Schnittpunkts zweier Strecken
  - Parametrische Darstellung der Strecken:



$$g_1 = v_0 + s(v_1 - v_0)$$
$$\Rightarrow (x(s), y(s)) = \left(v_{0,x} + s(v_{1,x} - v_{0,x}),\ v_{0,y} + s(v_{1,y} - v_{0,y})\right)$$

$$g_2 = v_2 + t(v_3 - v_2)$$
$$\Rightarrow (x(t), y(t)) = \left(v_{2,x} + t(v_{3,x} - v_{2,x}),\ v_{2,y} + t(v_{3,y} - v_{2,y})\right)$$

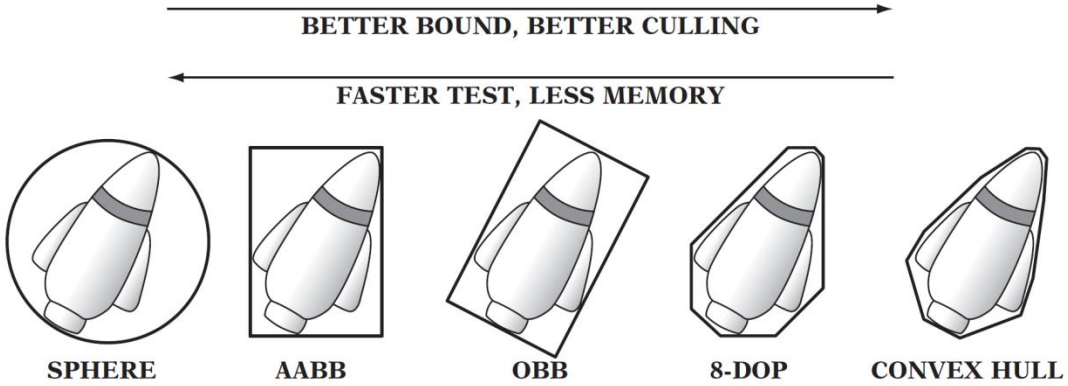
  - Schneiden sich die Strecken, so muss der Schnittpunkt (x_0, y_0) der zugehörigen Geraden Parameter s_0 und t_0 haben mit der Eigenschaft $0 \leq s_0, t_0 \leq 1$
  - Die Schnittparameter s_0 und t_0 sind Lösung des LGS:

$$s(v_{1,x} - v_{0,x}) - t(v_{3,x} - v_{2,x}) = v_{2,x} - v_{0,x}$$
$$s(v_{1,y} - v_{0,y}) - t(v_{3,y} - v_{2,y}) = v_{2,y} - v_{0,y}$$

  - Die Lösung erfolgt z.B. über die Cramer‘sche Regel. Man erhält s_0, t_0.
  - Anschließend wird getestet ob die Bedingung $0 \leq s_0, t_0 \leq 1$ erfüllt ist.
  - Gegenebenfalls wird der Schnittpunkt durch Einsetzen in eine Geradengleichung berechnet.

Kollisionserkennung

- Kollisionserkennung bei komplexeren Objekten durch Hüllkörper: Vereinfachung des komplexen Objekts

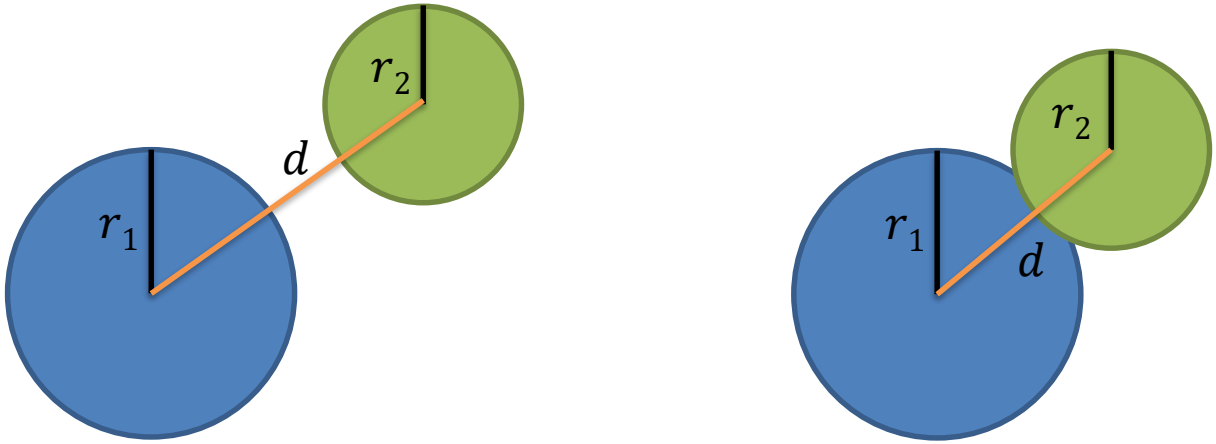


- Approximation durch
  - Kugel
  - Achsen-Parallele Bounding Box (AABB)
  - Am Objekt ausgerichtete Bounding Box (OBB)
  - Polytope, auch k-DOP (Discrete Oriented Polytopes)
  - Konvexe Hülle

C. Ericson: „Real Time Collision Detection“, Elsevier, 2005

Kollisionserkennung

- Beispiel: Kugel-Hüllkörper



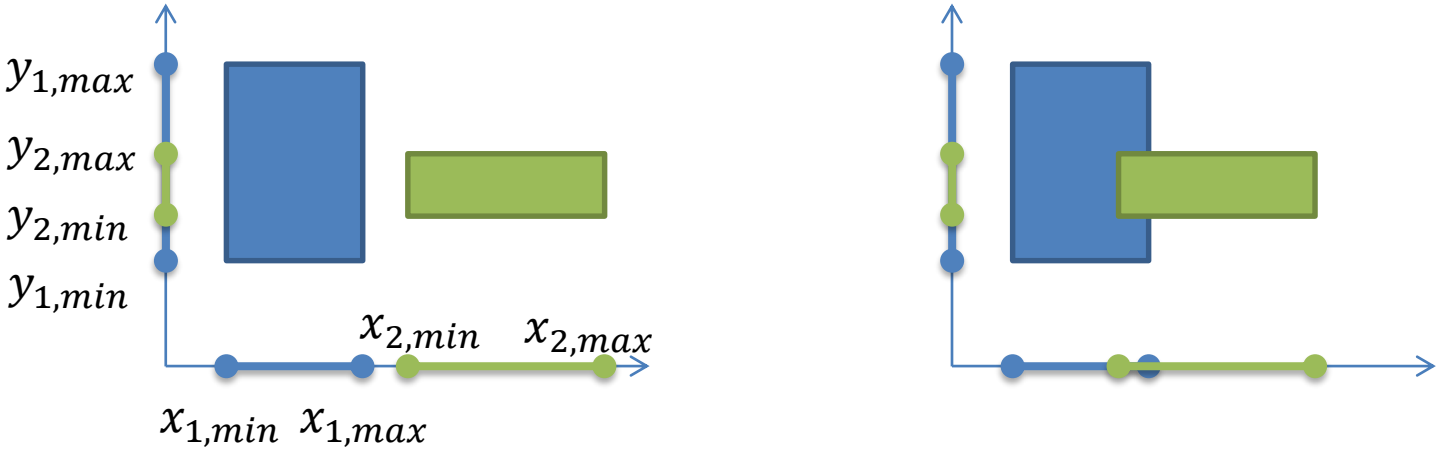
$d = r_1 + r_2$: Kugeln berühren sich in einem Punkt (= Kollision)

$d < r_1 + r_2$: Kugeln schneiden sich (= Kollision)

$d > r_1 + r_2$: Kugeln berühren und schneiden sich nicht (= keine Kollision)

Kollisionserkennung

- Beispiel: AABB Hüllkörper



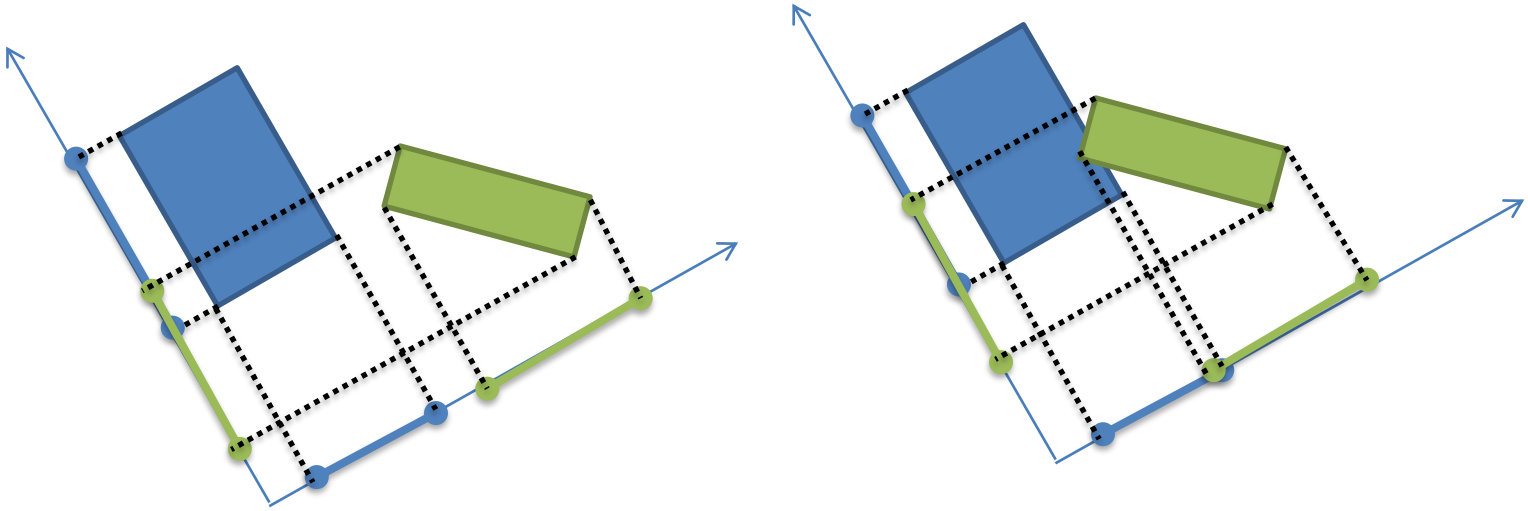
$x_{1,max} < x_{2,min} \vee y_{1,max} < y_{2,min}$: keine Kollision

$x_{1,max} \geq x_{2,min} \wedge y_{1,max} \geq y_{2,min}$: Kollision

Entsprechender Test für $x_{2,max}$und $x_{1,min}$ bzw. $y_{1,min}$ und $y_{2,max}$

Kollisionserkennung

- Beispiel: OBB Hüllkörper



  - Projektion der Körper auf Achsen, die Parallel zu einer Seite des OBB Hüllkörpers verlaufen
  - Testbedingungen analog zum AABB Hüllkörper

Kollisionserkennung

- Bei großer Anzahl von Objekten sind sehr viele Tests nötig: Bei n Objekten $\frac{n \cdot (n-1)}{2}$ Tests
  - 10 Objekte: 45 gegenseitige Tests
  - 100 Objekte: 4950 gegenseitige Tests!

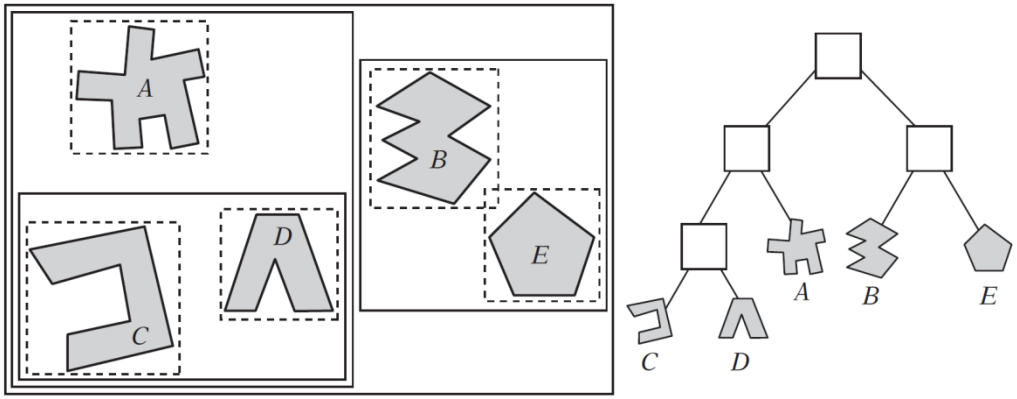
- Abhilfe: Hierarchiebäume und Raumunterteilung

- Hüllkörper meist nur in erster Näherung genutzt.
  - Wenn eine Kollision entdeckt wird, wird der Test verfeinert.

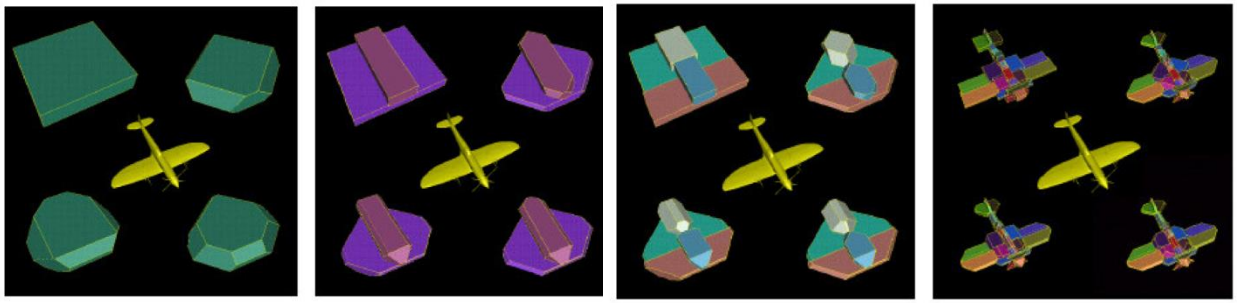
  → Hierarchiebäume

Kollisionserkennung

- Hierarchiebäume
  - Zusammenfassen von mehreren Objekten



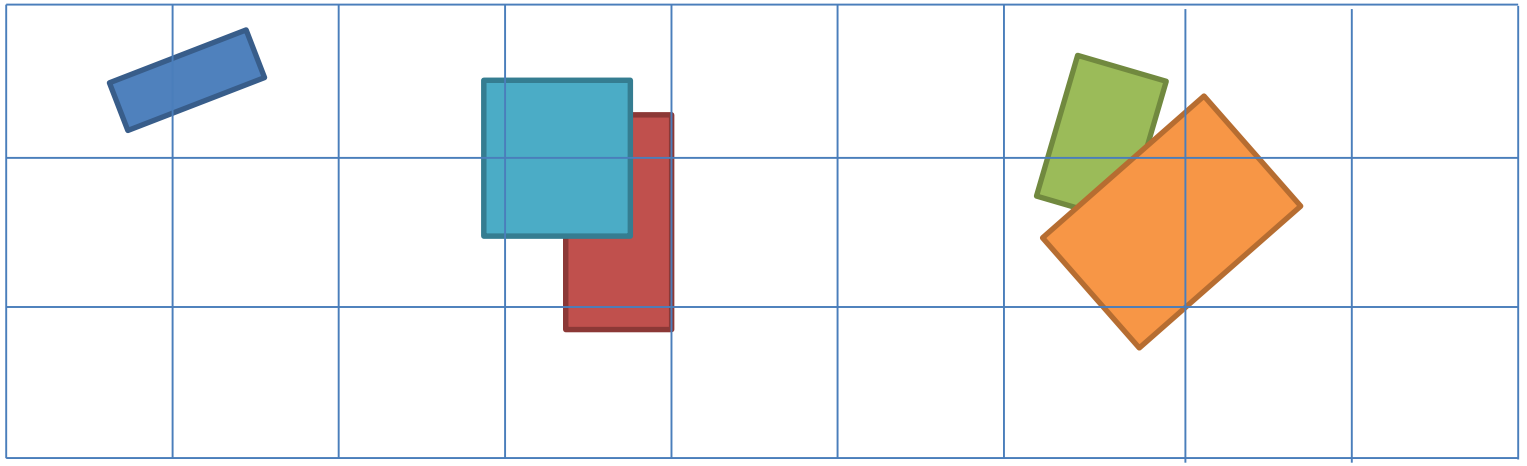
  - Weitere Unterteilung von Hüllkörpern wenn Kollision entdeckt wurde



J.T. Klosowski et al.: „Efficient collision detection using bounding volume hierarchies of k-DOPs", IEEE Trans. on Visualization and Computer Graphics 4(1), 1998

Kollisionserkennung

- Raumunterteilung:
  - Z.B. Uniforme Raumaufteilung



  - Zwei Objekte können sich nur überlappen, wenn sie in einer gemeinsamen Region liegen
  - Die Zahl paarweiser Tests lässt sich damit reduzieren
  - Wahl der Raumaufteilung wichtiger Parameter
  - Alternative Unterteilungsmöglichkeiten: Quad-/Octrees

Anwendungen

- Z.B. Kollissionserkennung in Computerspielen und Simulationen

http://www.flightsimworld.com/gallery/displayimage.php?pid=757
http://www.tagesspiegel.de/mobil/lkw-simulator-von-sifat-road-safety-alarmfahrt-auf-probe/10114080.html

Medizinische Anwendung

- z.B. Bewegung von starren Instrumenten bei OP-Simulation
- Kollisionserkennung wichtig: Force-Feedback!

https://www.oh-tech.org/blog/surgery_simulation_research_garners_london_praise#.VjJF8msmkUF

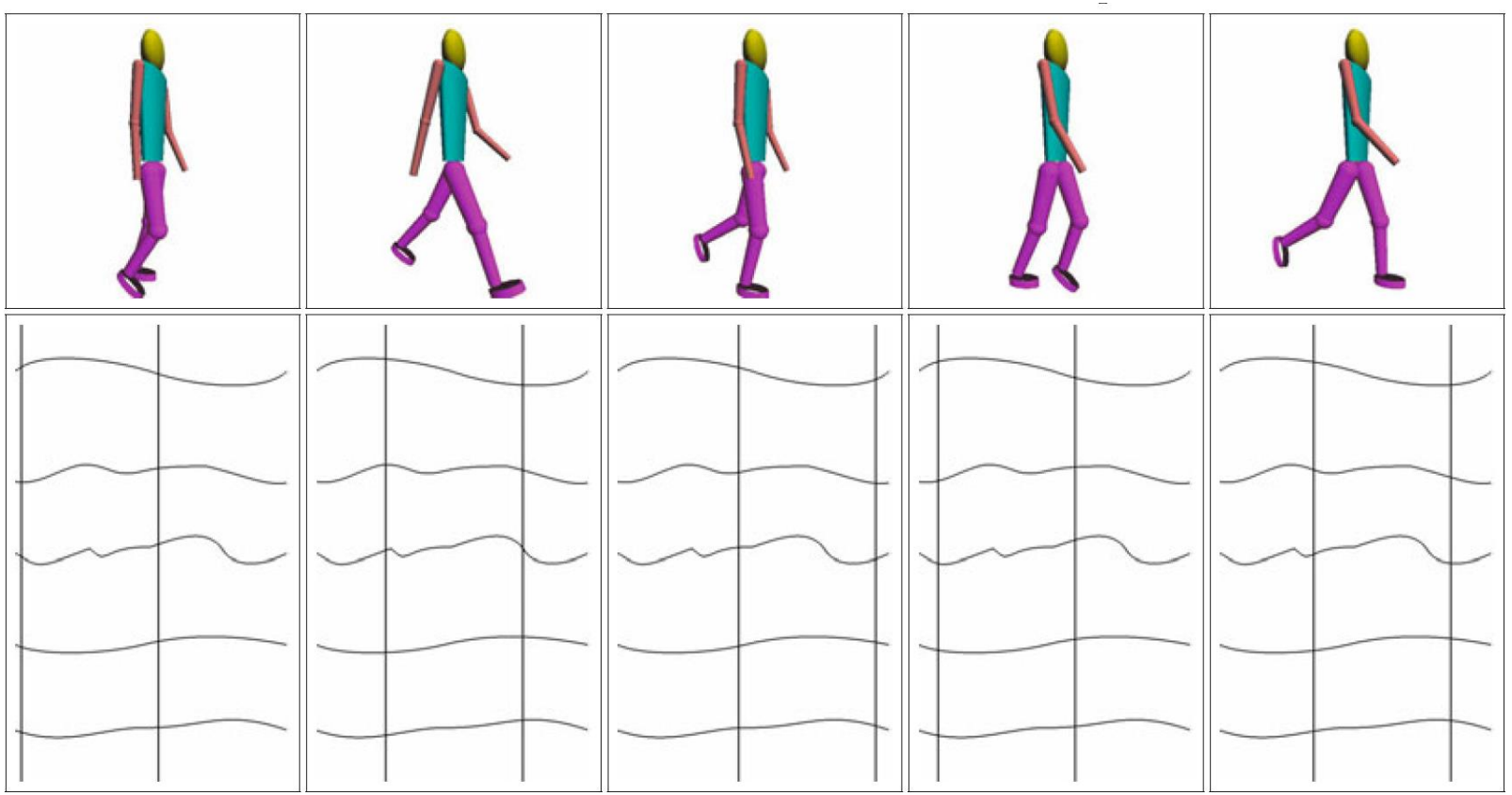
9.2 ARTIKULATION

Artikulation

- Bewegung von Gelenken eines Objekts, Teilobjekte sind starr
- Beschreibung der Gelenkwinkel als Kurve über der Zeit
  (= „Bewegungssatz“)

Artikulation

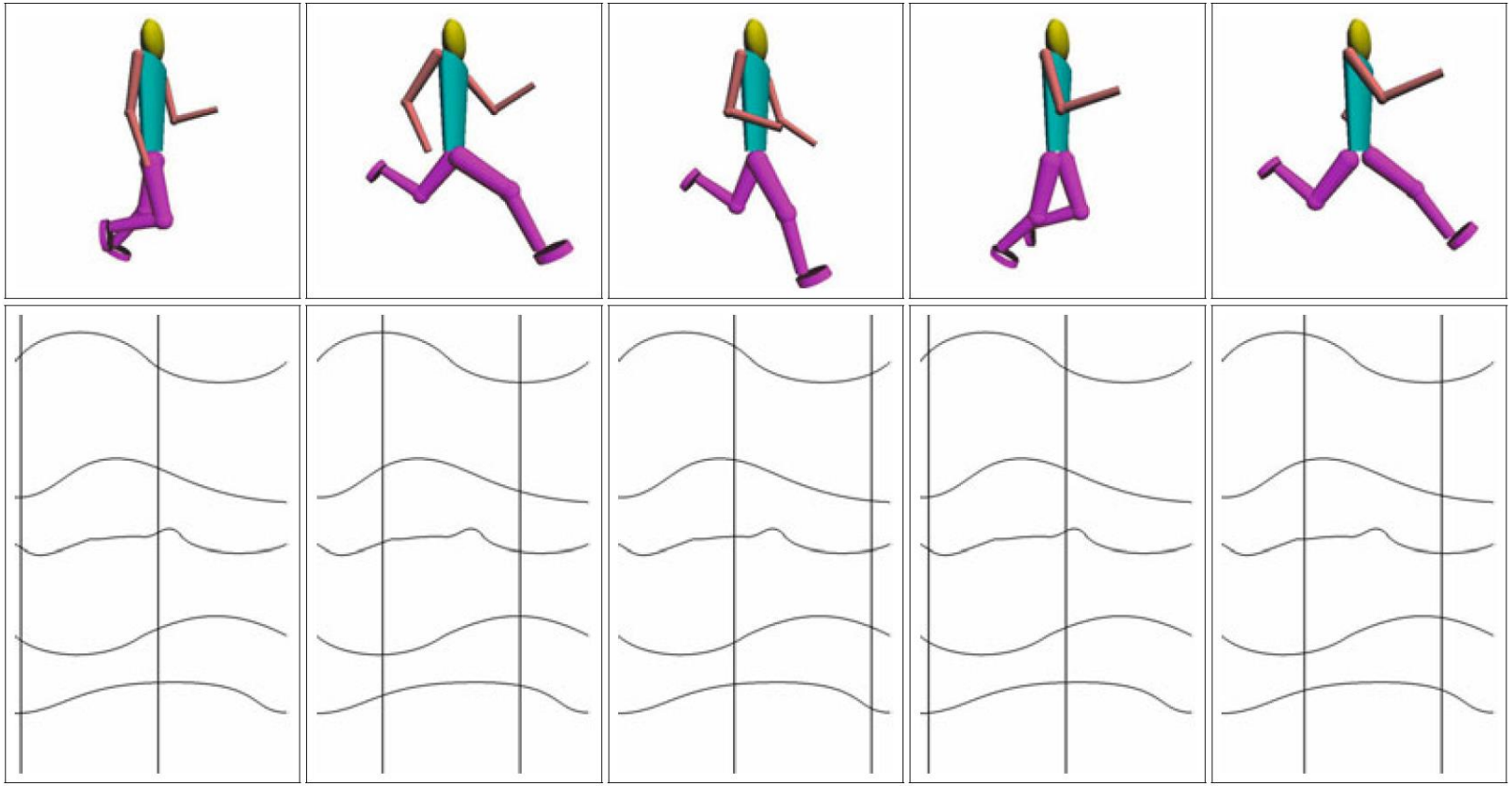
- Beispiel: Laufbewegung (10 Gelenke)



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Artikulation

- Änderung der Bewegung durch Austausch des Bewegungssatzes



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Artikulation

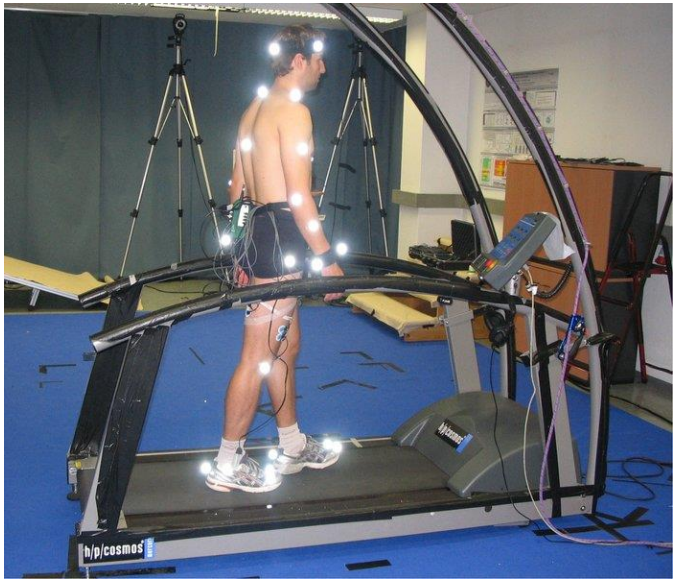
- Statt kontinuierlicher Kurven kann auch Key-Frame-Technik eingesetzt werden:
  - Diskreter Winkel zu einem bestimmten Zeitpunkt
  - Interpolation des Winkels
  - Glattheit der Bewegung bestimmt durch Anzahl der diskreten Winkel und Interpolationsmethode.

- Übergänge zwischen zwei Bewegungsmustern durch Interpolation der Bewegungssätze

- Oft umgekehrtes Problem: Optimale Bewegung für ein Szenario herausfinden
  - Z.B. Berechnen des Bewegungsablaufs zum Ergreifen eines Gegenstandes
  - Hochdimensionales Optimierungsproblem: „Inverse Kinematik“

Artikulation

- Realistischere Personenmodelle für vorgegebene Bewegungen durch Motion Capture
  - Z.B. Bewegungsanalyse → Schätzung der Gelenkwinkel



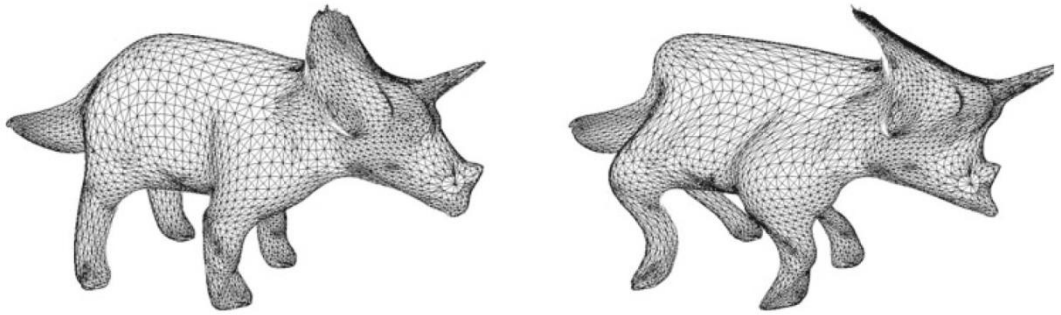
http://www.cin.uni-tuebingen.de/mission-methods/methods/medical-technical-applications.html
https://www.yahoo.com/movies/a-brief-history-of-motion-capture-in-the-movies-91372735717.html

9.3 MORPHING

Morphing

- Morphing = Verformung von Oberflächen
  - Objekt wird nicht mehr als starrer Körper betrachtet
  - Objekt nicht nur an Gelenken beweglich

- Direkte Verschiebung der Vertex-Positionen
  - Rechenintensive Operationen, da meist Glattheitseigenschaften berücksichtigt werden
  - Normalen der Oberflächen müssen in jedem Verformungsschritt neu berechnet werden



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Morphing

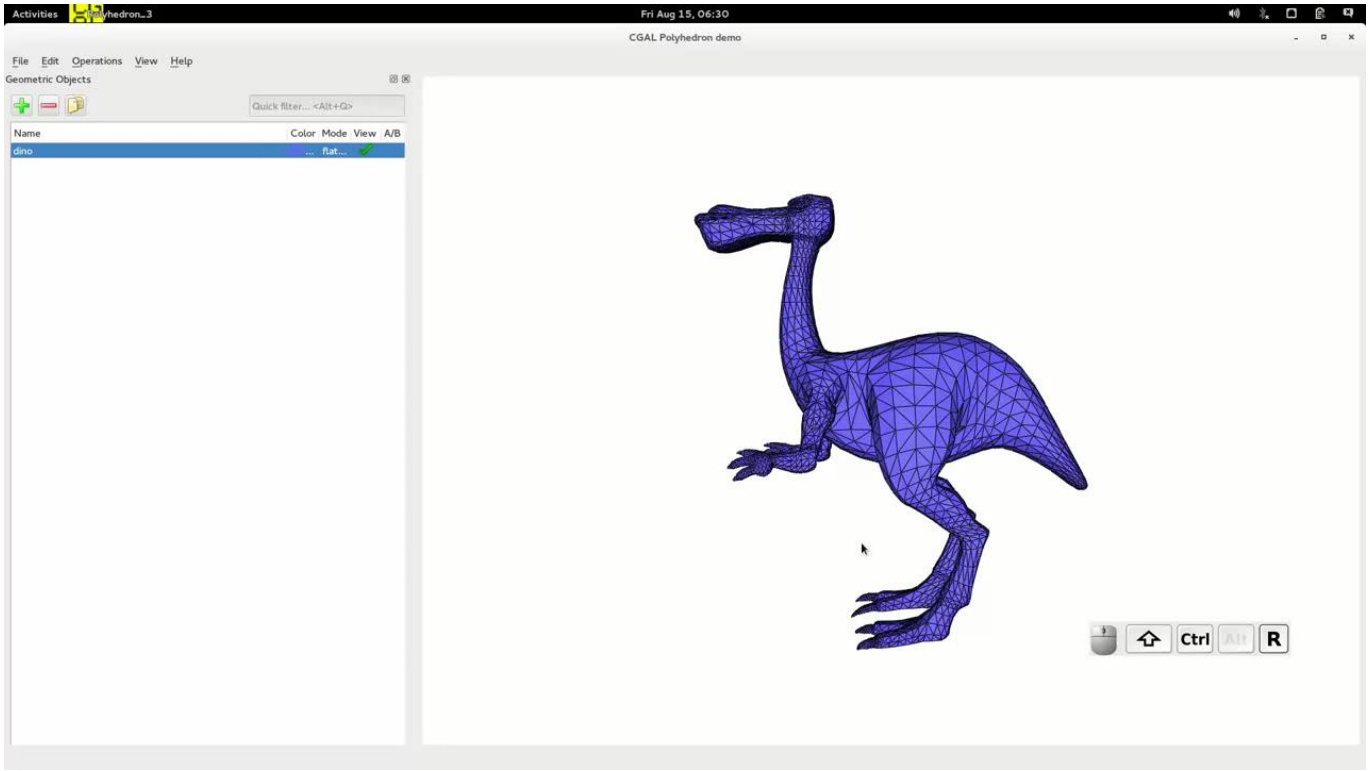
- Ähnlich der Starrkörper-Bewegung, hier aber Definition einer Bewegungsbahn pro Vertex
  - Key-Frame Technik kann eingesetzt werden
  - Topologie des Polygonnetzes bleibt i.d.R. gleich, nur die Vertex-Positionen werden verschoben.

- Bewegungsbahn durch explizite Definition, physikalische Berechnung etc.

- Kollisionserkennung
  - Kollisionen zwischen mehreren Objekten: in jedem Morphing-Schritt bestimmen
  - Eigenkollision möglich!

Morphing

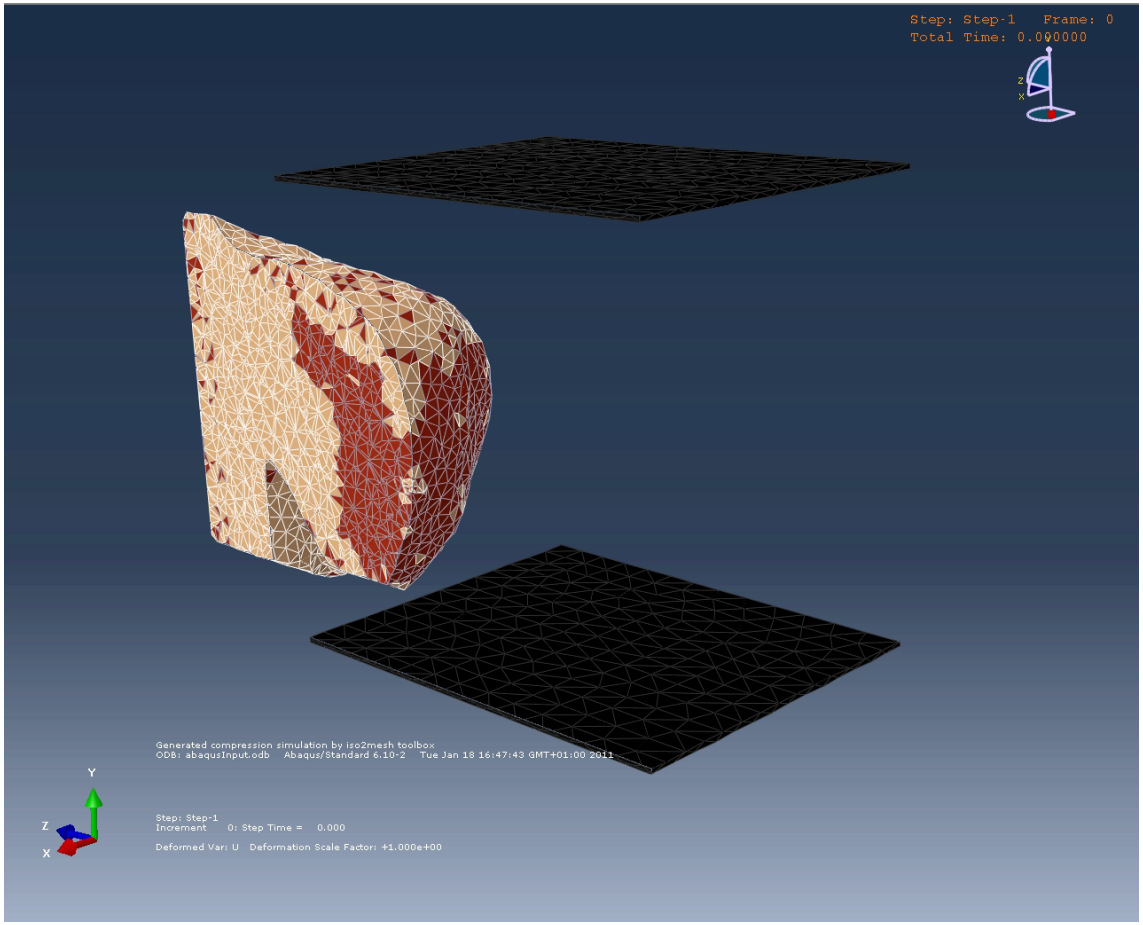
- Beispiel: Methoden von CGAL



http://doc.cgal.org/latest/Surface_modeling/index.html

Morphing

- Beispiel aus der Medizin: physikalische Simulation



Simulation und Visualisierung mit ABAQUS

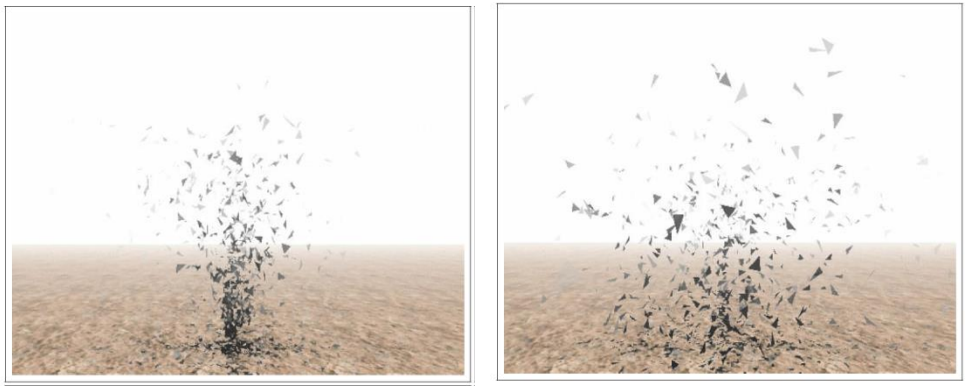
T.Hopp: „Multimodal Registration of X-Ray Mammograms with 3D Volume Datasets", PhD Thesis, University of Mannheim, 2012

9.4 SCHWÄRME UND PARTIKELSYSTEME

Partikelsysteme

- Ansammlung von einfachen Grafikobjekten (Partikel) bildet zusammen ein großes Objekt
- Verwendet zur Darstellung von natürlichen Phänomenen
  - … deren Form nicht eindeutig definierbar ist
  - … die keine eindeutige glatte Oberfläche besitzen

- Beispiele: Feuer, Wasser, Wolken, Bäume, Gräser, Dampf/Rauch, Textilien

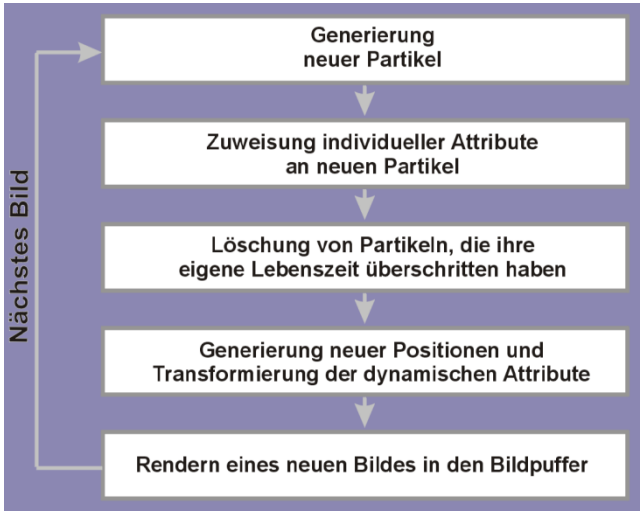


A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Partikelsysteme

- Objekt wird durch Häufung von Partikeln beschrieben
- Partikelsystem beschrieben durch zwei Komponenten:
  - Partikel = Elemente des Systems
  - Emitter = Kontrolle über Partikel

- Attribute der Partikel ändern sich über die Zeit, z.B.
  - ***Position***: Punkt im Raum, an dem sich der Partikel zu einem Zeitpunkt befindet
  - ***Geschwindigkeit*** und ***Richtung***
  - ***Größe***
  - ***Farbe***
  - ***Transparenz***
  - ***Form/Aussehen***
  - ***Lebensdauer***



I. Sejdic, D. Erdmann, D. Wilhelm, C. Kurz: „Partikelsysteme & Schwärme“, Seminararbeit, Universität Kassel

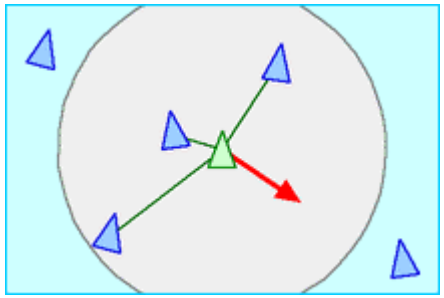
Partikelsysteme

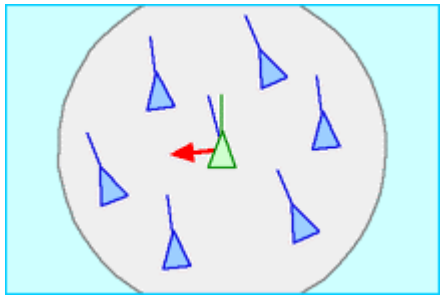
- Stochastische Partikelsysteme
  - Verhalten der Partikel unabhängig. (Beispiel: Feuer)
- Strukturierte Partikelsysteme
  - Verhalten der Partikel in Abhängigkeit von z.B. Nachbarn. (Beispiel: Grasanimation)

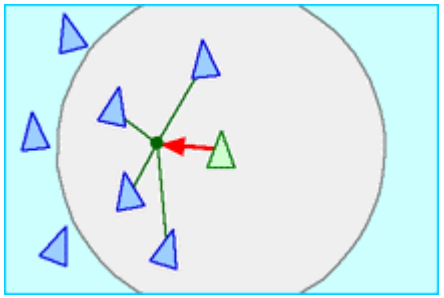
https://www.youtube.com/watch?v=pd0jSmExgB0

Schwärme

- Partikelsystemen sehr ähnlich
- Dennoch kleine Unterschiede:
  - Schwarmelemente bestehen so lange wie der Schwarm selbst
  - Dynamischer Eindruck (z.B. Vogelschwarm-Animation)
  - Schwarmelemente komplexer als Partikel
  - Schwarmelemente greifen aktiv in Geschehen ein. Komplexe Abhängigkeiten können entstehen:
    - Berechnung der Kollisionsvermeidung (z.B. durch Simulation von Kraftfeldern), Variation der Geschwindigkeit
    - Regeln für Schwarmverhalten (z.B. gemeinsame Ausrichtung, Zentrierung)







http://www.red3d.com/cwr/boids/

ZUSAMMENFASSUNG

Zusammenfassung

- Animation = jegliche Veränderung einer Szene über die Zeit
  - **Pfadanimation**: Bewegung starrer Objekte entlang einer räumlichen Bahn
    - Meist Vorberechnung der Bahn an diskreten Stellen + Interpolation (Key Frame Technik)
    - Kollisionserkennung wichtig: einfache Objekte, Hüllkörper, Raumunterteilung, Hierarchieunterteilung
  - **Artikulation**: Bewegung innerer Freiheitsgrade eines Objektes (z.B. Gelenk)
    - Bewegungskurve pro Gelenk: Winkel eines Gelenks
  - **Morphing**: elastische oder plastische Verformung eines Objektes
  - **Partikelsysteme**: gleichartige Bewegung einer Objektgruppe

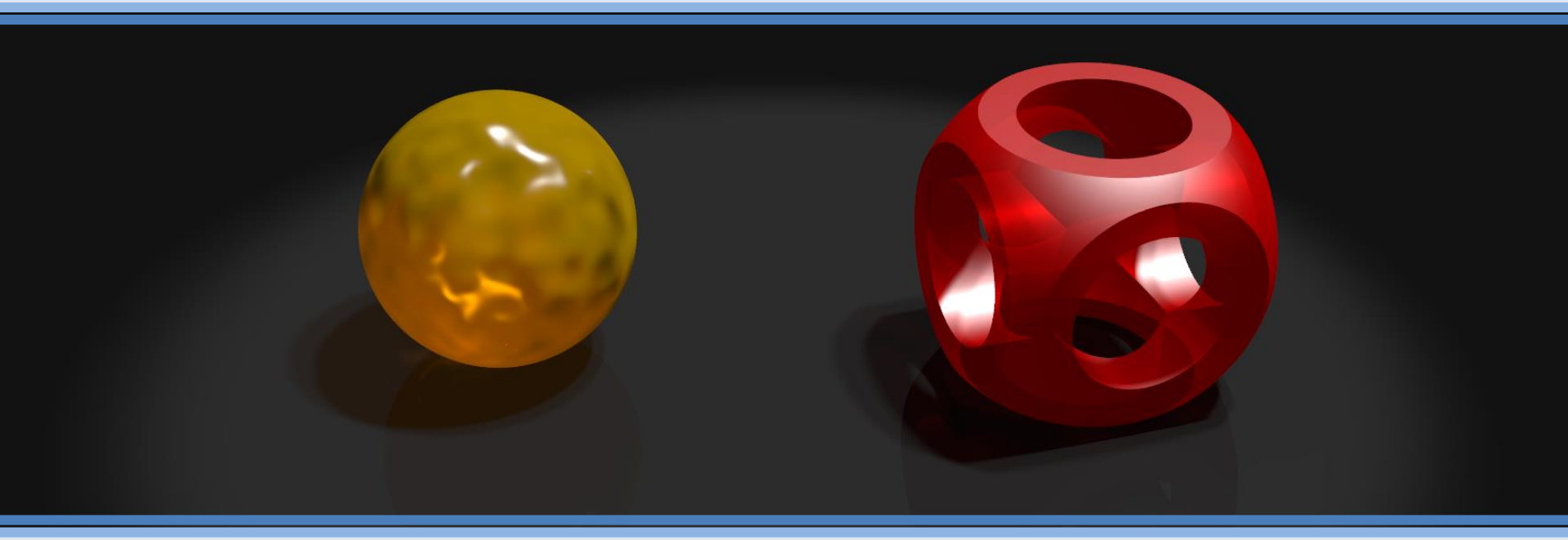
ÜBUNGSAUFGABEN

Übungsfragen Kapitel 9

- Was ist die Key-Frame-Technik?
- Beschreiben Sie die Funktion von Hüllkörpern für die Kollisionserkennung
- Was ist der Unterschied zwischen Partikelsystem und Schwarm?

Computergrafik

T. Hopp



Themenübersicht

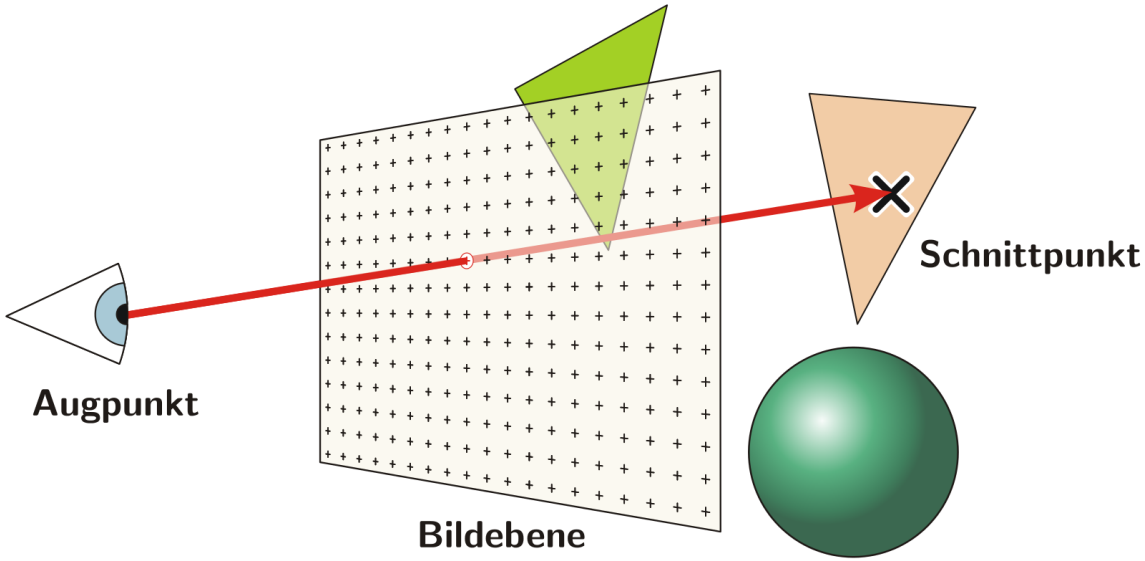
1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. **Raytracing**
11. Volumenvisualisierung

Raytracing

- Berechnung von (Mehrfach-)Reflexionen und Transmission von Licht → globale Illumination!
- Raytracing arbeitet im Objektraum: Prinzip der Strahlverfolgung in die Szene für jedes Pixel.
- Vorteile: das Raytracing-Modell berechnet durch Strahlverfolgung
  - Verdeckung
  - Transparenz
  - Direkte Beleuchtung
  - Indirekte Beleuchtung
  - Schatten
- Nachteile:
  - Extremer Aufwand!
  - Bei Streulicht an diffusen Oberflächen versagt „klassisches“ Raytracing → z.B. Radiosity
  - Bündeln von Lichtstrahlen (=Kaustik) benötigt ebenfalls Erweiterungen (Photon Mapping, Path Tracing)

Verdeckungsberechnung

- Grundsätzlicher Raytracing-Algorithmus führt eine Verdeckungsberechnung durch:
  - Sichtstrahl vom Augpunkt durch jedes Pixel der *near clipping plane*
  - Berechnung des ersten Schnittpunktes mit einem Objekt. (Schnittpunkt mit geringstem Abstand zum Augpunkt)
  - Farbberechnung durch Beleuchtungsrechnung, Schattierungsalgorithmus



https://de.wikipedia.org/wiki/Raytracing

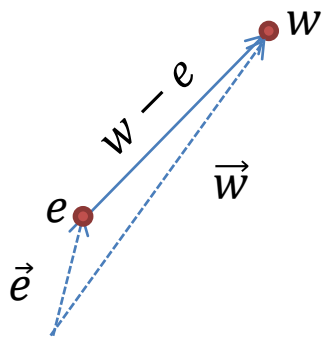
Berechnung von Sichtstrahlen

- Parameterdarstellung einer Geradengleichung vom Augpunkt e zu einem Pixel w in der Bildschirmebene:

$$p(t) = e + t(w - e)$$

- Durch jedes t ist ein Punkt auf der Geraden beschrieben:
  - $t = 0$: Punkt e
  - $t = 1$: Punkt w
  - $t_1 < t_2$: t_1 ist näher am Augpunkt als t_2.

- Bestimmung von w:
  - Augpunkt-Transformation der Szene
  - Diskretisierung der near clipping plane.



Verdeckungsberechnung

- Pseudocode für ein einfaches Ray Tracing, das Ergebnisse ähnlich dem z-Buffer liefert

```
Funktion Bild_Rendern
    Strahl.Ursprung := Augpunkt
    Für jedes (x,y)-Pixel der Rastergrafik
        Strahl.Richtung := [3D-Koordinaten des Pixels der Bildebene] – Augpunkt
        Farbe des (x,y)-Pixels := Farbe_aus_Richtung(Strahl)

Funktion Farbe_aus_Richtung(Strahl)
    Schnittpunkt := Nächster_Schnittpunkt(Strahl)
    Wenn Schnittpunkt.Gewinner = (kein) dann
        Farbe_aus_Richtung := Hintergrundfarbe
    sonst
        Farbe_aus_Richtung := Farbe_am_Schnittpunkt(Strahl, Schnittpunkt)

Funktion Nächster_Schnittpunkt(Strahl)
    MaxDistanz := ∞
    Schnittpunkt.Gewinner := (kein)
    Für jedes Primitiv der Szene
        Schnittpunkt := Teste_Primitiv(Primitiv, Strahl)
        Wenn Schnittpunkt.Distanz < MaxDistanz dann
            MaxDistanz := Schnittpunkt.Distanz
            Schnittpunkt.Gewinner := Primitiv
    Nächster_Schnittpunkt := Schnittpunkt
```

Beschleunigung der Schnittpunktberechnung

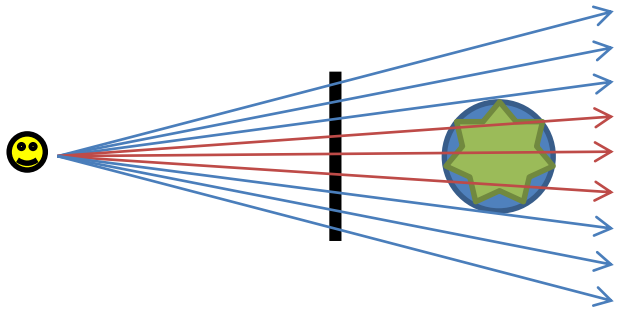
- Implementierung schneidet jeden Sichtstrahl mit jedem Objekt: extrem aufwändig!
  - Auflösung 1024 x 768 bei 100 Objekten: ca. 78 Mio. Schnittpunkberechnungen!
  - Ca. 75-95% der Rechenzeit wird für Schnittpunktberechnung benötigt

- Beschleunigungs-Strategien: sehr ähnlich zu Kollisionsdetektion

  - Z-Sort:
    1. Transformation der Strahlen, so dass sie entlang der z-Achse verlaufen
    2. Transformation der Objekte und Sortierung nach z-Werten → Schnittpunkt mit vorderstem Objekt
    3. Zurücktransformation der Schnittpunkte für Schattierungsberechnung

Beschleunigung der Schnittpunktberechnung

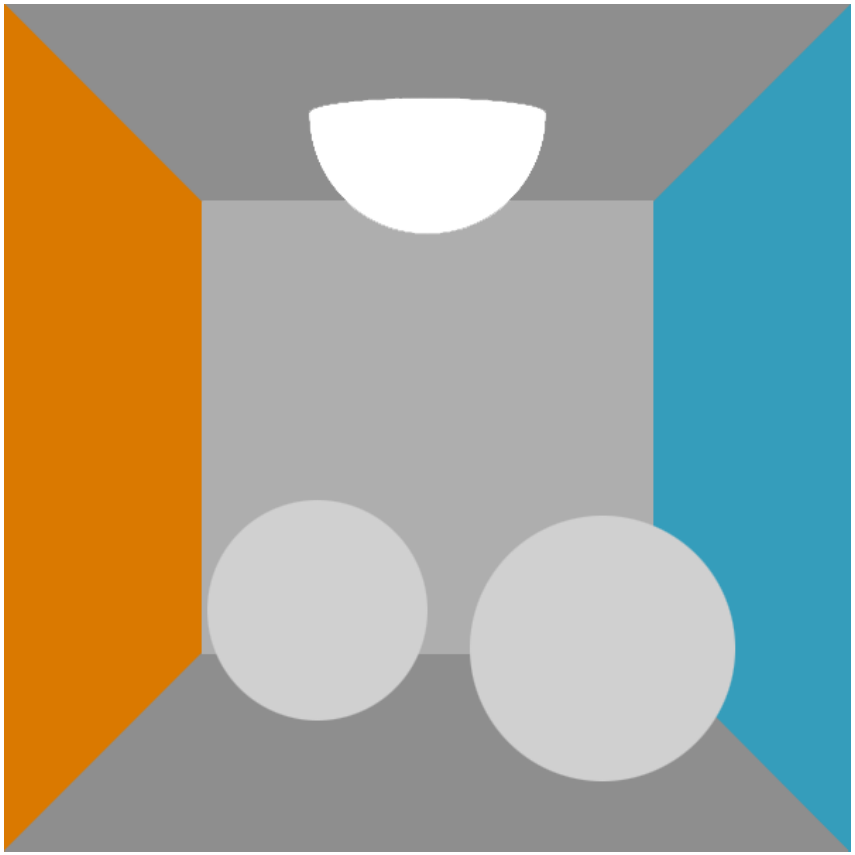
- Begrenzende Volumen:
  - Komplexe Objekte mit aufwändiger Schnittpunktberechnung durch umhüllenden Körper, z.B. Kugel oder Quader umgeben
  - Testen des komplexen Objektes nur wenn der umhüllende Körper vom Sichtstrahl geschnitten wird



- Hierarchien:
  - Organisation von begrenzenden Volumen in verschachtelten Hierarchien
  - Baumartiger Aufbau mit komplexen Objekten als Blätter
  - Test der Objekte eines Baumzweiges kann entfallen wenn Vaterknoten schon nicht geschnitten wird.
- Bereichsunterteilung
  - Unterteilung der gesamten Szene in ein regelmäßiges Gitter
  - Zuweisung der Objekte zu Bereichen
  - Schnittberechnung nur mit Objekten wenn ein Strahl den zugehörigen Bereich durchläuft.

Verdeckungsberechnung

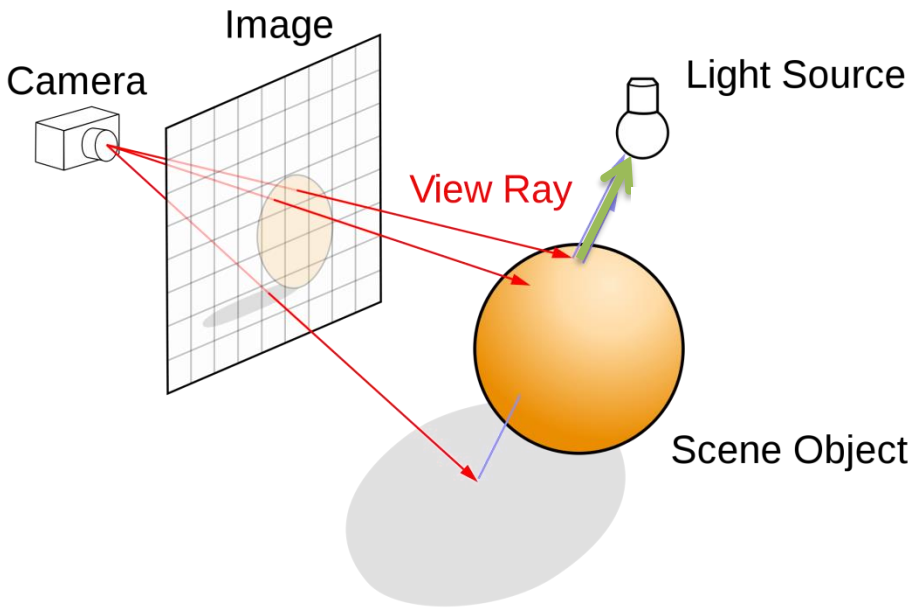
- Bisher:



https://de.wikipedia.org/wiki/Raytracing

Beleuchtung von Objekten

- Die Beleuchtung von Objekten erfolgt analog zu den bisher besprochenen Beleuchtungsmodellen:
  - Winkelberechnung zwischen eintreffendem Sichtstrahl zur Oberflächennormale
  - Winkelberechnung zwischen Oberflächennormale und Gerade von Schnittpunkt zu Lichtquelle
  - Immer noch lokale Beleuchtung!

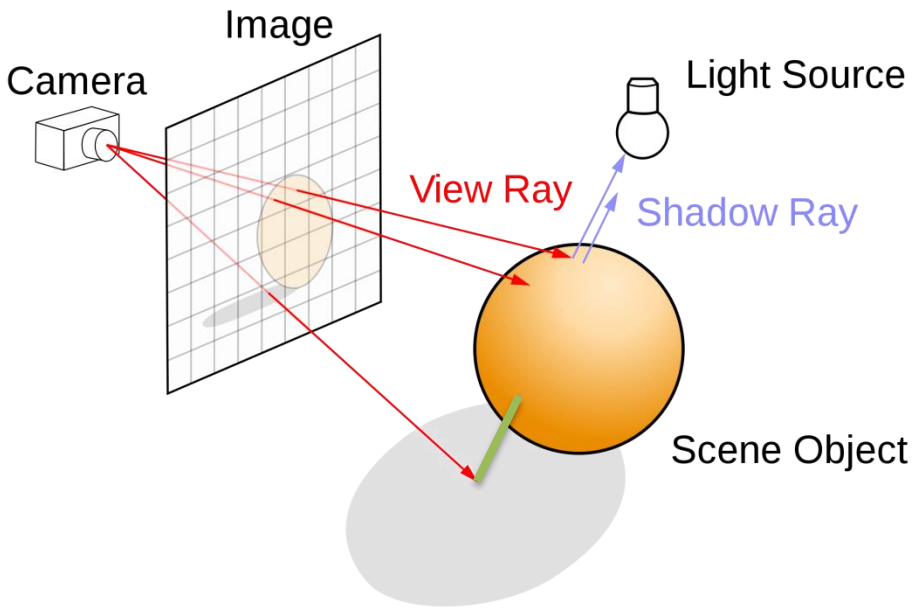


  - Bei mehreren Lichtquellen: Beleuchtungsberechnung für alle Lichtquellen!

https://en.wikipedia.org/wiki/Ray_tracing_%28graphics%29

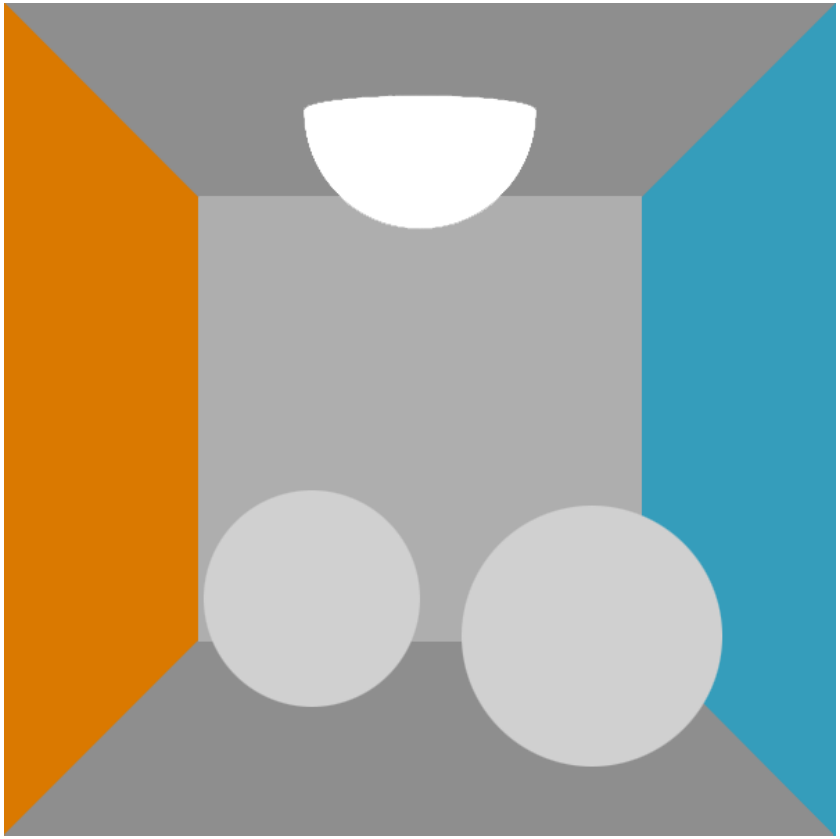
Schattenberechnung

- Analog können auch Schatten berechnet werden:
  - Schnittpunkte auf Oberflächen, die durch Verdeckung durch andere Objekte die Lichtquelle nicht sehen können.
  - Schnittpunkt-Test für den Strahl vom Schnittpunkt zur Lichtquelle.
  - Wenn ein Schnittpunkt existiert: Farbwahl = Schattenfarbe (meist schwarz)

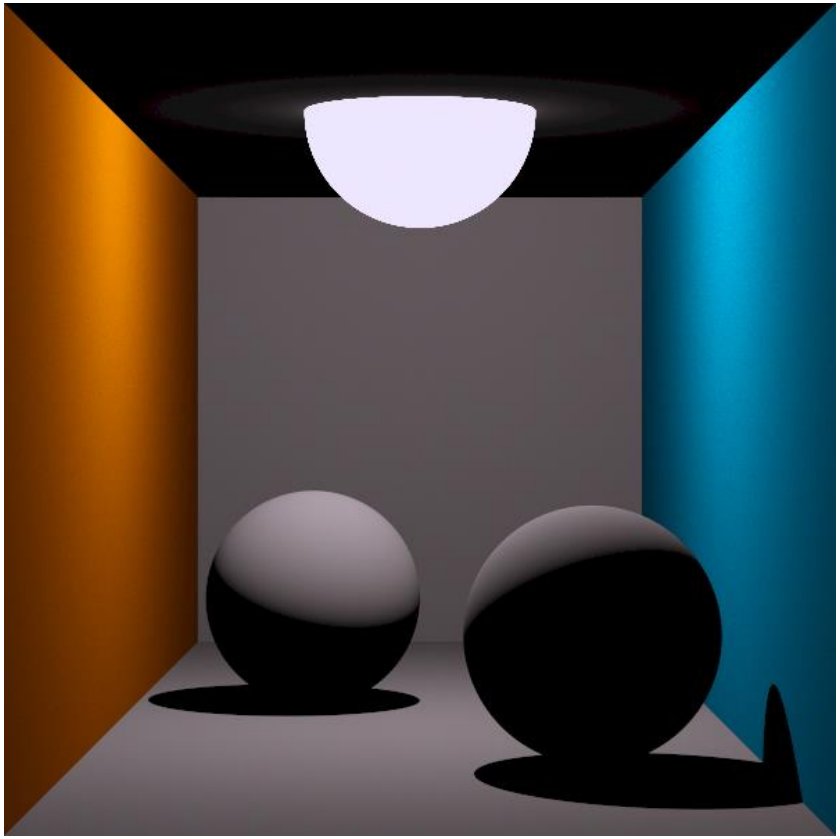


https://en.wikipedia.org/wiki/Ray_tracing_%28graphics%29

Beleuchtung & Schatten



Verdeckungsberechnung



Raytracing mit Beleuchtungsberechnung und Schatten

https://de.wikipedia.org/wiki/Raytracing

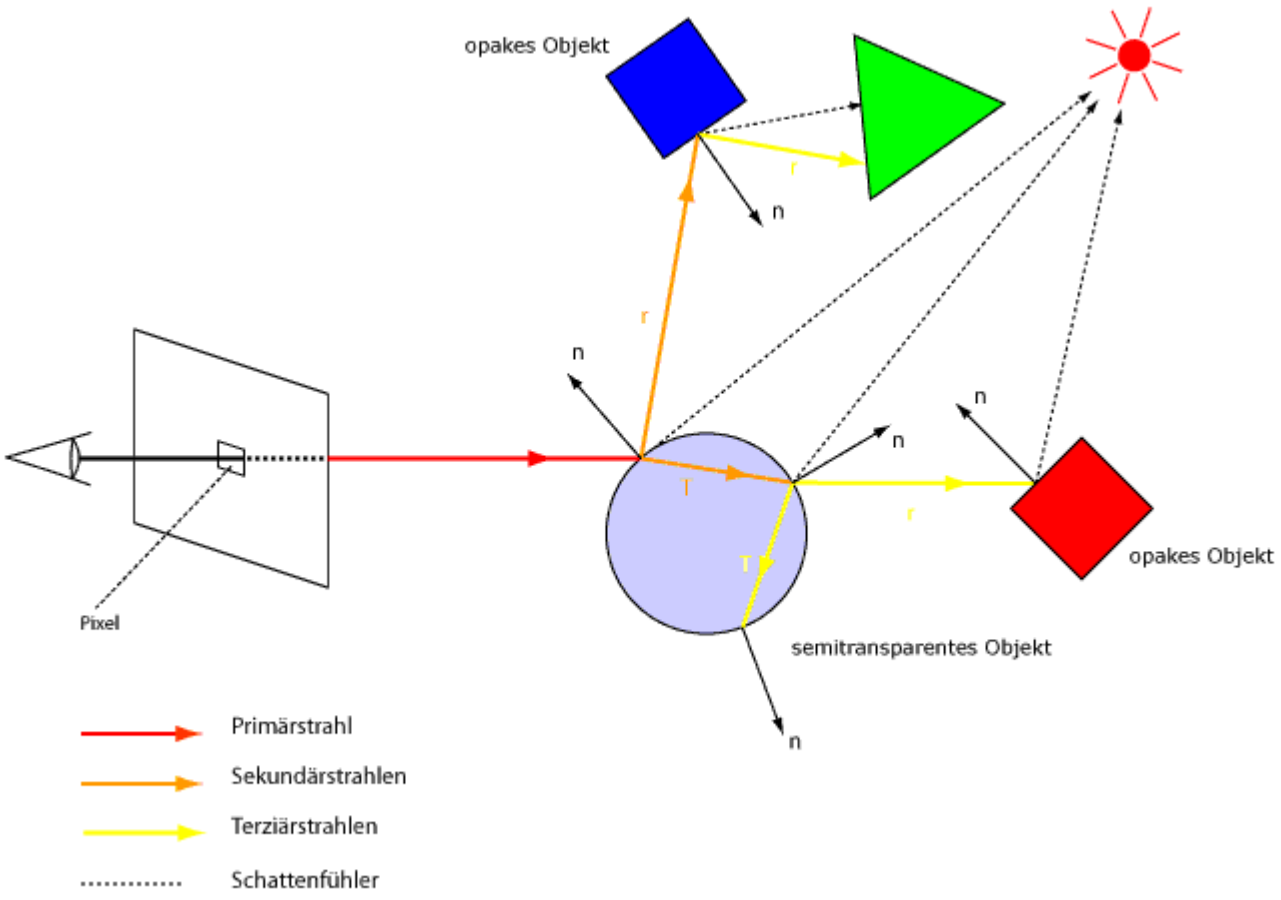
Rekursives Raytracing

- Verallgemeinerung: Verfolgung von Sekundärstrahlen durch Spiegelung und Brechung des Sichtstrahls.
  - Berechnung des Reflexionsvektor r durch Reflexionsgesetz: $r = l + 2(l \cdot N) \cdot N$
  - Für transparente Objekte Berechnung des Transmissionsvektors t durch Brechungsgesetz: $t = \frac{n_1(l - N(l \cdot N))}{n_2} - N\sqrt{\frac{n_1(1-(l \cdot N)^2)}{{n_2}^2}}$
  - Beachtung des Fresnel-Effektes in Abhängigkeit des Auftreffwinkels eines Sichtstrahls auf die Oberfläche: Gewichtung der Farbintensitäten der Sekundärstrahlen

- Entscheidungskriterium für Weiterverfolgung: Abbruch bei erreichter Rekursionstiefe oder wenn Strahl kein Objekt trifft.

- Pixelfarbe ergibt sich durch Integration über die Primär- und Sekundärstrahlen (Beleuchtungsberechnung!)

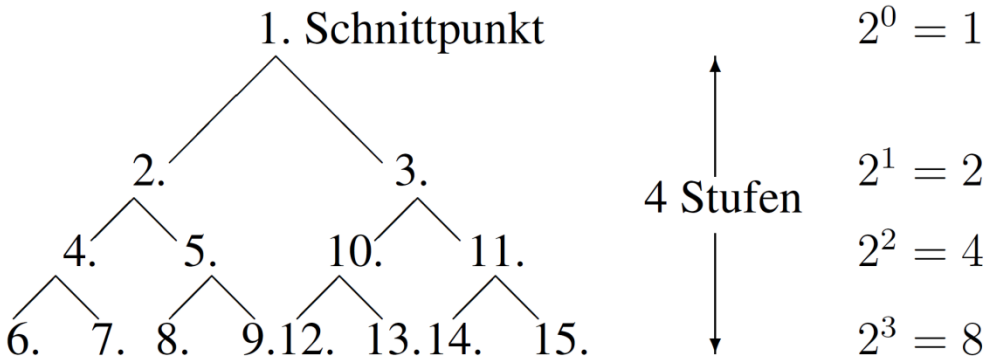
Rekursives Raytracing



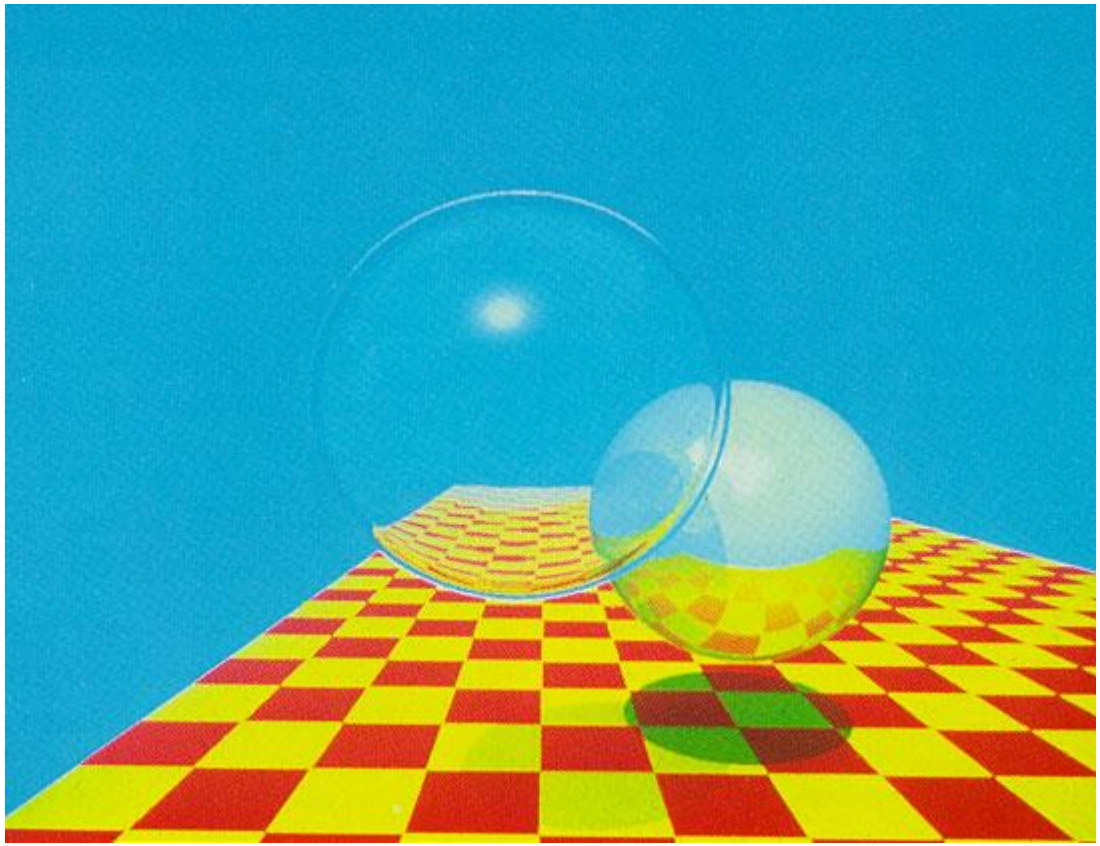
http://www.dma.ufg.ac.at/app/link/Grundlagen%3A3D-Grafik/module/8653?step=all

Rekursives Raytracing

- Implementierung: Binärer Baum
  - Bei jedem Auftreffen auf eine Oberfläche, entsteht ein reflektierter und gebrochener Strahl
  - Jeder dieser Strahlen schneidet erneut ein Objekt usw.
  - Es entsteht ein binärer Baum: bei Schachtelungstiefe n weist er eine Anzahl von $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ Schnittpunkten auf.

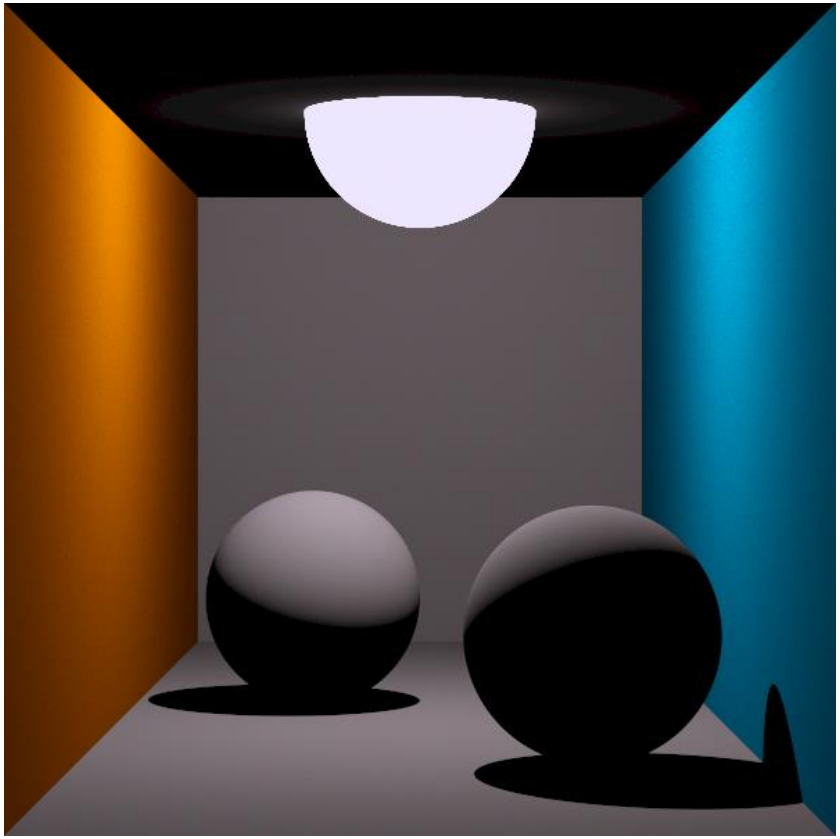


Rekursives Raytracing

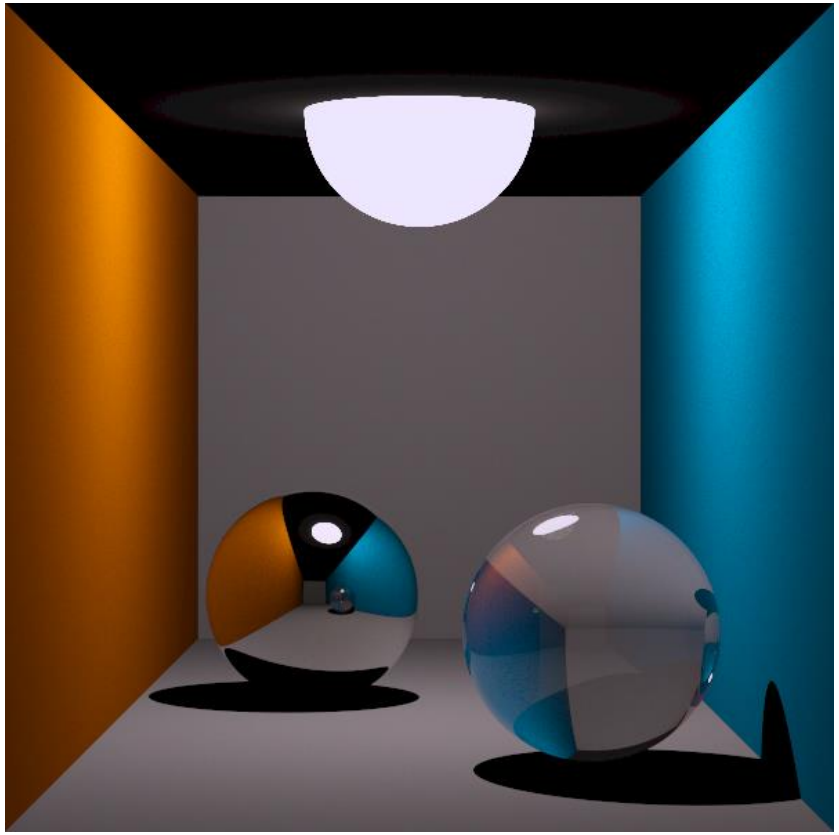


https://de.wikipedia.org/wiki/Raytracing

Rekursives Raytracing



Raytracing mit Beleuchtungsberechnung und Schatten

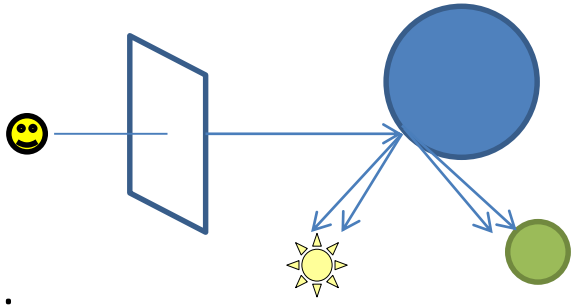


Rekursives Raytracing

https://de.wikipedia.org/wiki/Raytracing

Diffuses Raytracing

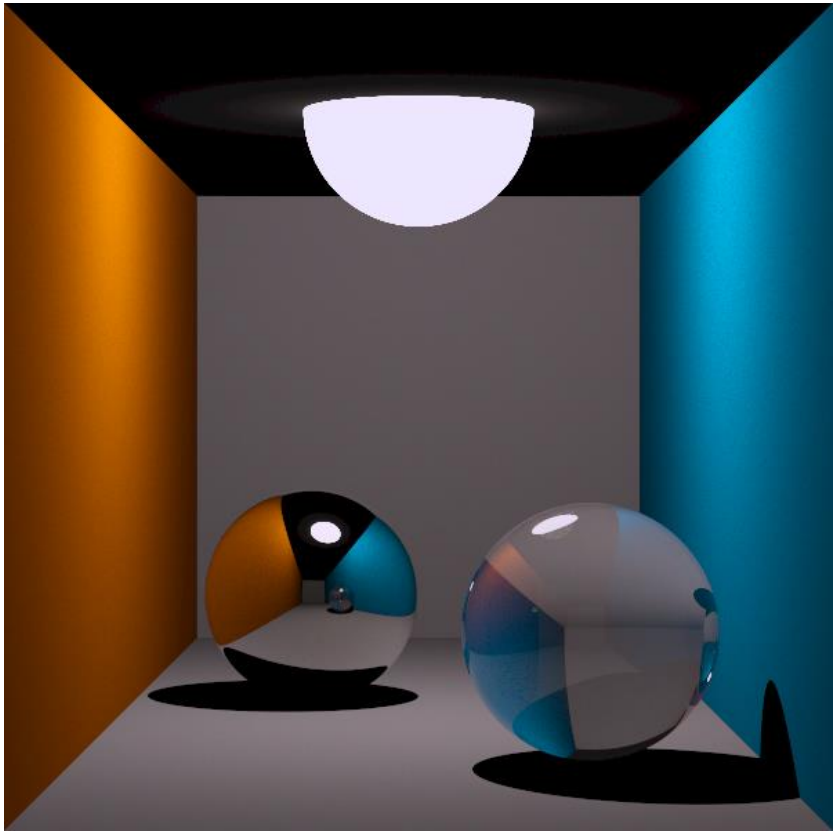
- Grundidee: stochastische Verteilung mehrerer Strahlen statt nur einem gebrochenem/reflektierten Strahl
  - Simulation von weichen bzw. verschwommenen Eigenschaften



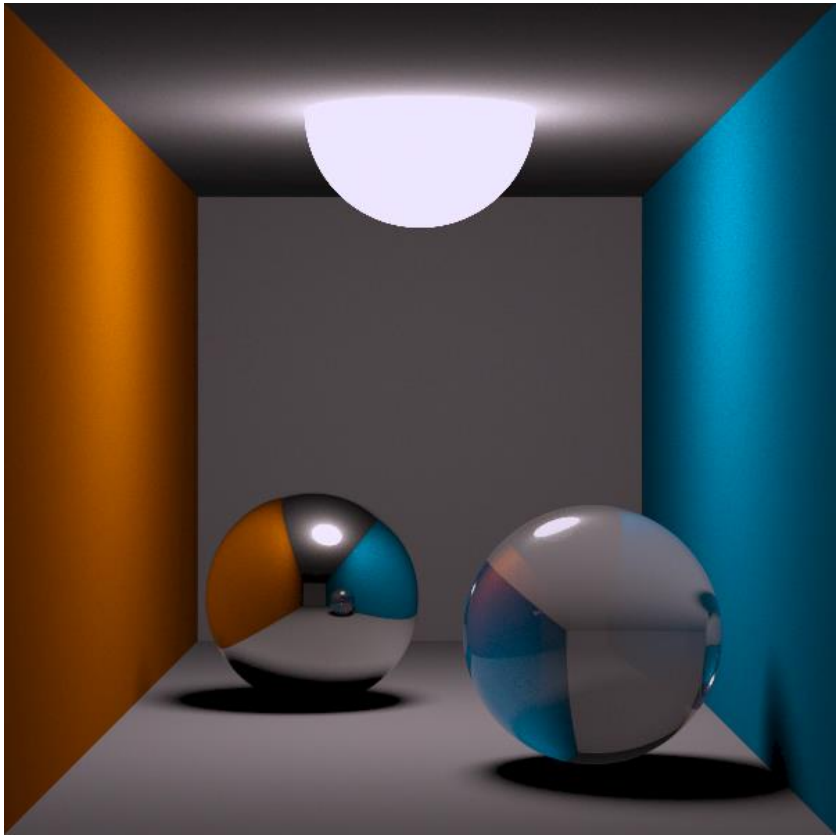
- Erzielbare Effekte:
  - Echte, weiche Lichtschatten (Abtastung der sichtbaren Oberfläche der Lichtquellen)
  - Verschwommene Lichtreflexionen auf glänzenden Oberflächen
  - Tiefenunschärfe (Abtastung der Linsenoberfläche)
  - Bewegungsunschärfe (Aufintegration der Strahlen über die Zeit)
  - Antialiasing (Mittelwert mehrerer Strahlen pro Pixel)

- Deutlich höherer Aufwand!
- Erlaubt noch keine Berechnung globaler Beleuchtung
  - Keine Sekundärstrahlen bei diffus streuenden Objekten

Diffuses Raytracing



Rekursives Ray Tracing



Diffuses Ray Tracing

https://de.wikipedia.org/wiki/Raytracing

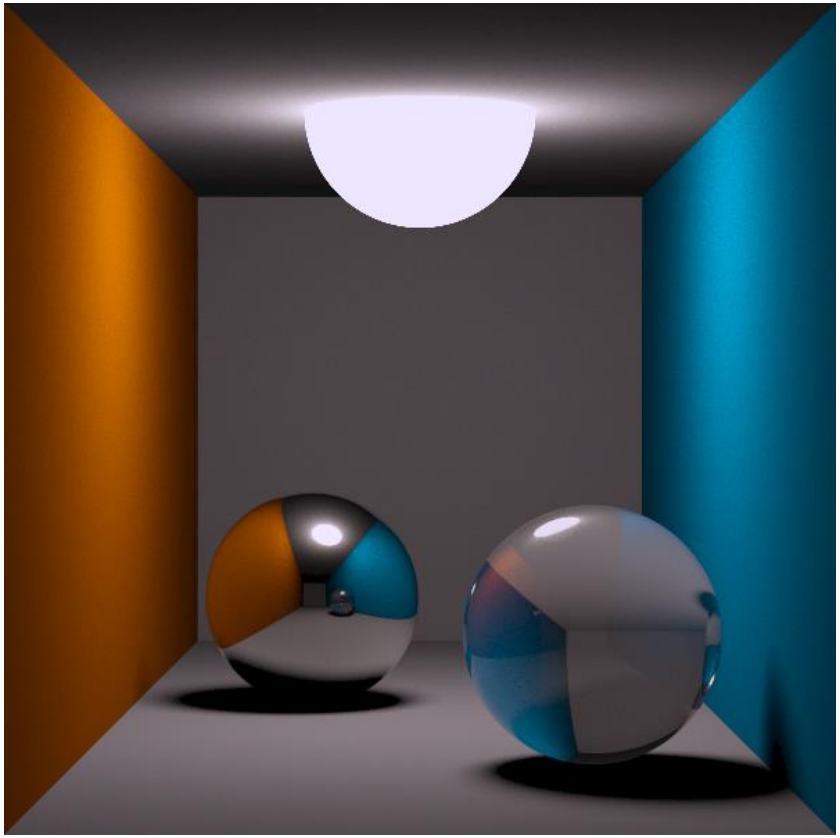
Path Tracing

- Diffuses Raytracing löst die Rendergleichung näherungsweise für spiegelnde Objekte (Strahlverfolgung nur in Reflexions- und Transmissionrichtung, Beleuchtungsrechnung).
- Grundidee von Path Tracing: „echte“ Verfolgung von zufälligen Strahlen.
- Integration über viele zufällige Strahlen pro Pixel in verschiedene Richtungen.
- Erlaubt Berechnung von Kaustiken
  - =Bündelung von Lichtstrahlen durch Objekte

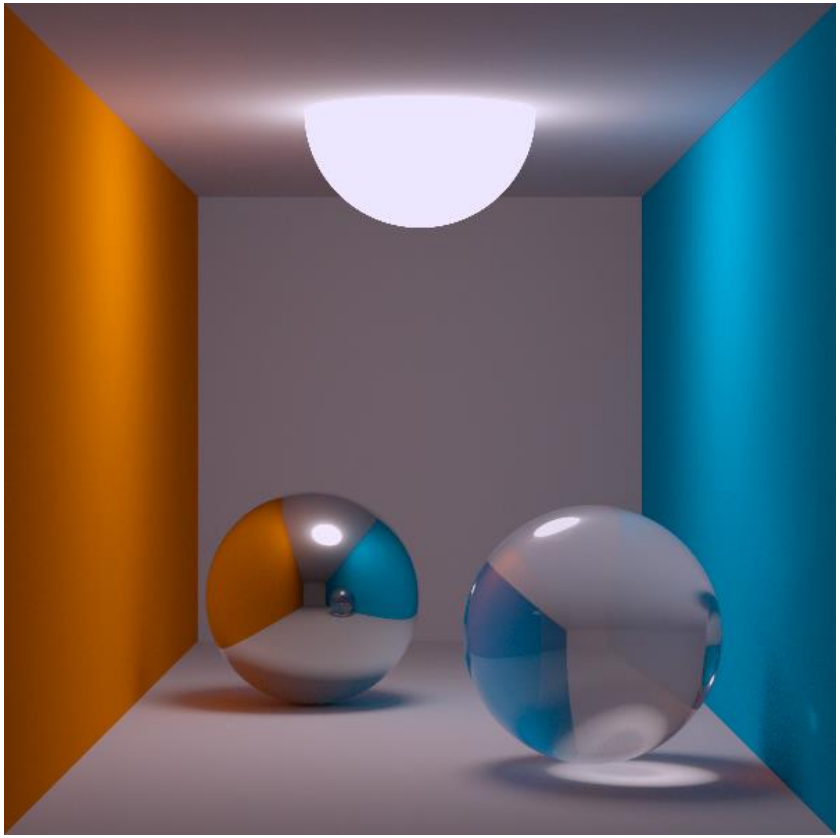
- Hoher Aufwand
- Rauschen bei zu niedriger Strahlanzahl

- Seltene Erweiterung: Light Raytracing:
  - Umkehrung der Tracing-Richtung: Strahlen ausgehend von der Lichtquelle

Path Tracing



Diffuses Raytracing



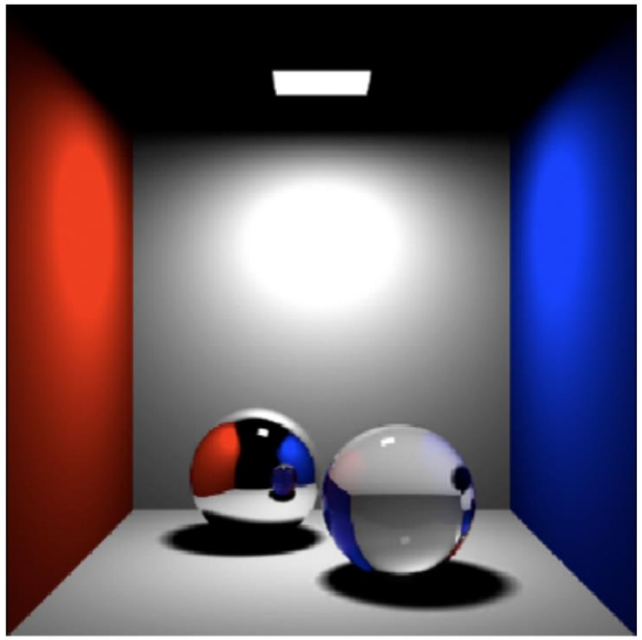
Path Tracing

https://de.wikipedia.org/wiki/Raytracing

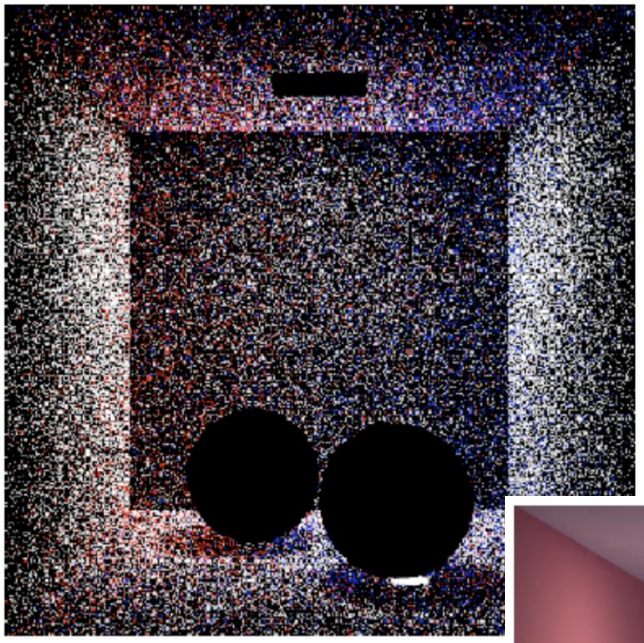
Photon Mapping

- Grundidee: Ermitteln der indirekten Beleuchtung durch umgekehrte Tracing-Richtung:
  - Strahlen werden ausgehend von Lichtquelle in die Szene „geschossen“
  - Reflexion, Brechung, Streuung, Absorption
  - Speicherung in Photon Map wenn Strahl auf diffus reflektierende Oberfläche trifft
    - Spezielle Datenstruktur, meist dreidimensionaler Baum
- Kombination mit „Vorwärtsrichtung“ des Raytracings
  - Wenn ein Strahl auf eine diffuse Oberfläche trifft, wird die indirekte Beleuchtung durch die Photon Map bestimmt.
  - Addition von indirekter Beleuchtung (Photon Map) und direkter Beleuchtung (Diffuses Raytracing, Path Tracing) ergibt globale Beleuchtung
- Erweiterung um Kaustik Map: Beschleunigung von Path Tracing

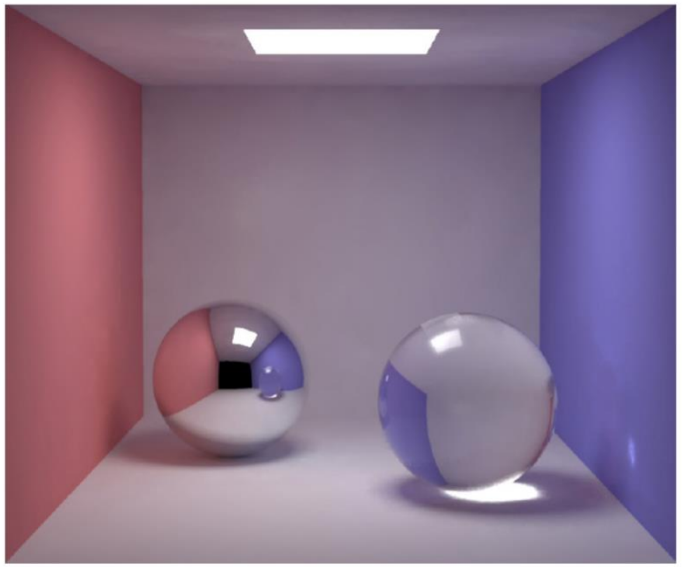
Photon Mapping



Diffuses Ray Tracing



Photon Map



Kombination: Photon Mapping

S. Krömker: Computergrafik I, Universität Heidelberg

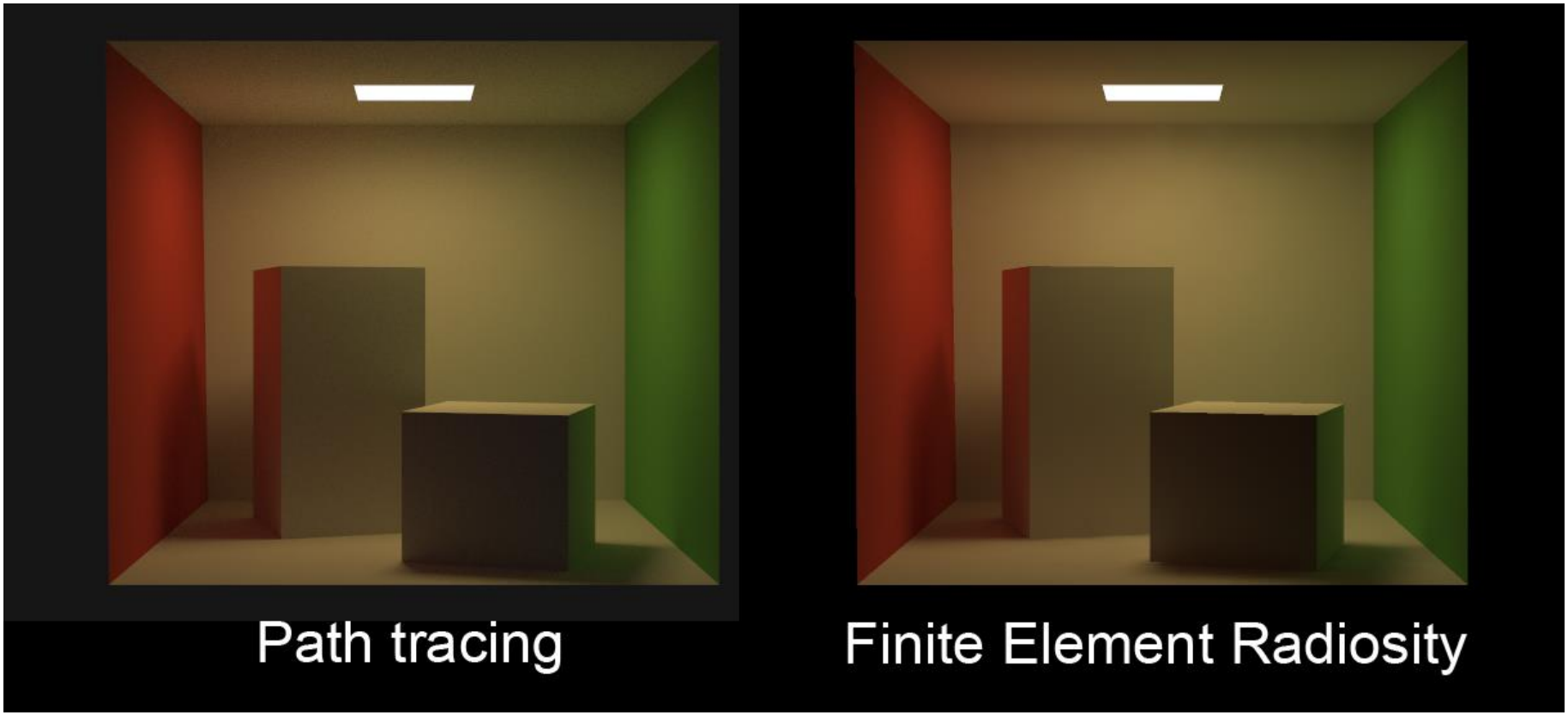
Photon Mapping

https://rosgar.wordpress.com/tag/yafaray/

Radiosity

- Alternative zu Raytracing-Verfahren für globale Illumination
- Beruht auf Energieerhaltungssatz: Alles Licht, das auf eine Fläche fällt und von dieser nicht absorbiert wird, wird von ihr zurückgeworfen.
- Nicht vom Blickpunkt abhängig: Beleuchtung für die gesamte Szene unabhängig von der Position des Betrachters berechnet.
  - Blickpunktabhängige Verdeckungsberechnung in unabhängigen Schritt.
- Alle Oberflächen ideal diffuse Reflektoren
- Löst die aus der Rendergleichung abgeleitete Radiosity-Gleichung
- Sehr aufwendige Berechnung, heute Tendenz eher zu Path Tracing, Photon Mapping um etwa gleiche Ergebnisse zu erreichen

Radiosity



http://www.deluxerender.com/2017/01/comparing-two-global-illumination-methods/

Beispiel: Brigade Rendering Engine

https://www.youtube.com/watch?v=FbGm66DCWok

ZUSAMMENFASSUNG

Zusammenfassung

- Berechnung von Reflexionen und Transmission von Licht → globale Illumination!
- Standard-Raytracing:
  - Prinzip der Strahlverfolgung vom Betrachter in die Szene
  - Verdeckungs-, Beleuchtungs-, Schattenberechnung
- Rekursives Raytracing:
  - Verallgemeinerung: Mehrfachreflexionen/-brechung
- Diffuses Raytracing:
  - stochastische Verteilung mehrerer Strahlen → weiche Darstellung
- Path Tracing:
  - Lösung der Rendergleichung für diffus streuende Objekte → indirekte Beleuchtung
- Photon Mapping:
  - Ermitteln der indirekten Beleuchtung durch umgekehrte Tracing-Richtung → Alternative und/oder Beschleunigung
- Radiosity:
  - Alternatives Verfahren zur globalen Beleuchtung
  - Lösung der diskreten Radiosity-Gleichung

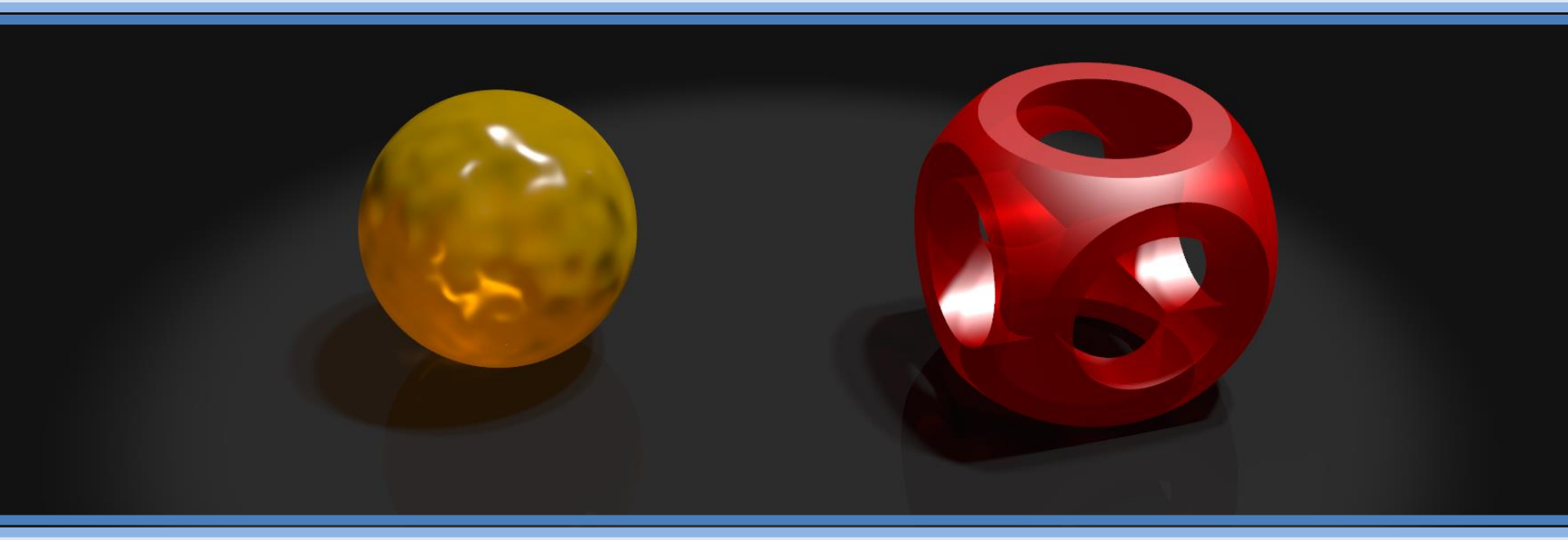
ÜBUNGSAUFGABEN

Übungsfragen Kapitel 10

- Erläutern Sie kurz das Prinzip des rekursiven Raytracings
- Erläutern Sie den Unterschied zwischen Standard- (rekursivem) Raytracing und diffusem Raytracing? Welche Effekte lassen sich dadurch erzielen?

Computergrafik

T. Hopp

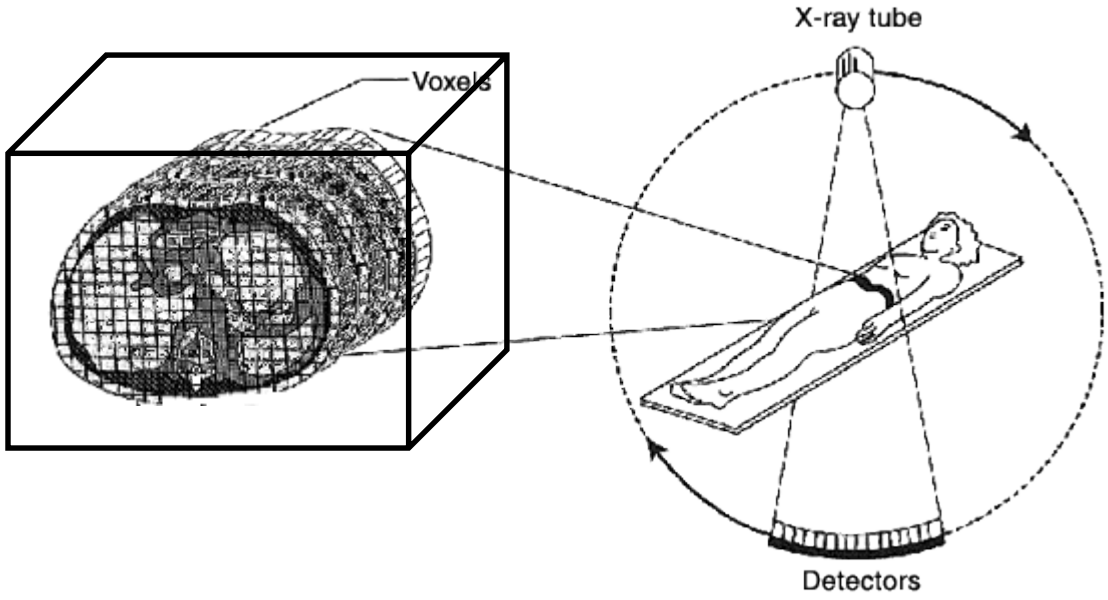


Themenübersicht

1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. Zeichenalgorithmen
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. Raytracing
11. **Volumenvisualisierung**

Volumendaten

- Viele bildgebende Verfahren erzeugen 3D Volumendaten
- Oft in diskretem kartesischem Gitter angeordnete Voxel
- Zur Darstellung auf dem Bildschirm muss die Information auf 2D reduziert werden

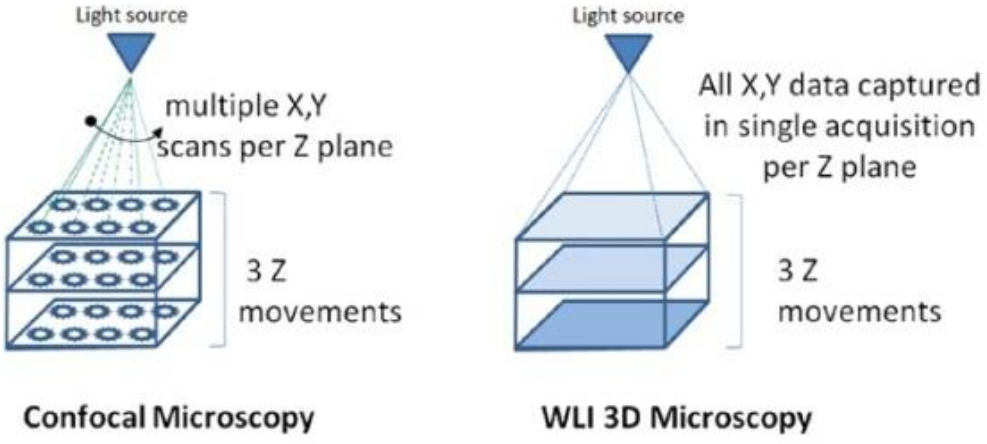


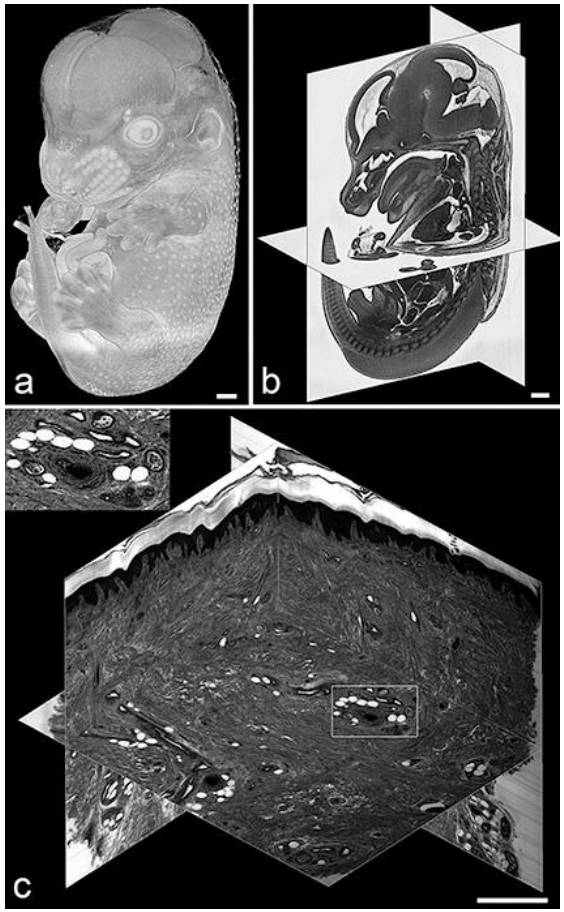
Beispiel: Computertomographie

http://www.physicscentral.com/explore/action/images/scans-img2.gif

Volumendaten

- Z.B. 3D Mikrospkopie



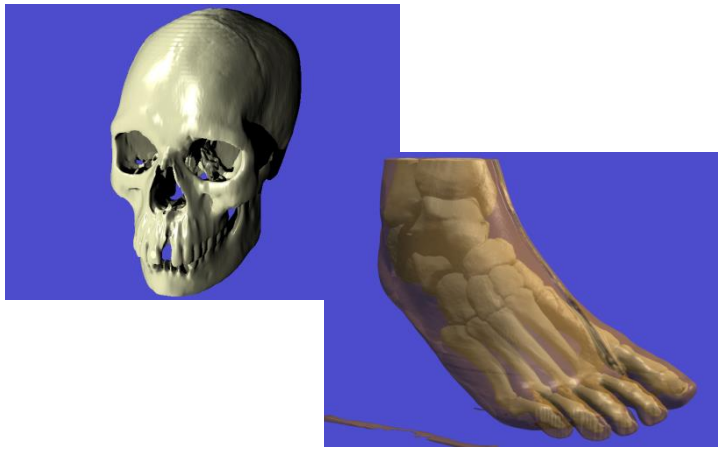


- Extrem hohe Auflösungen, bis Teravoxel!
- Oft Zeitreihen (4D)

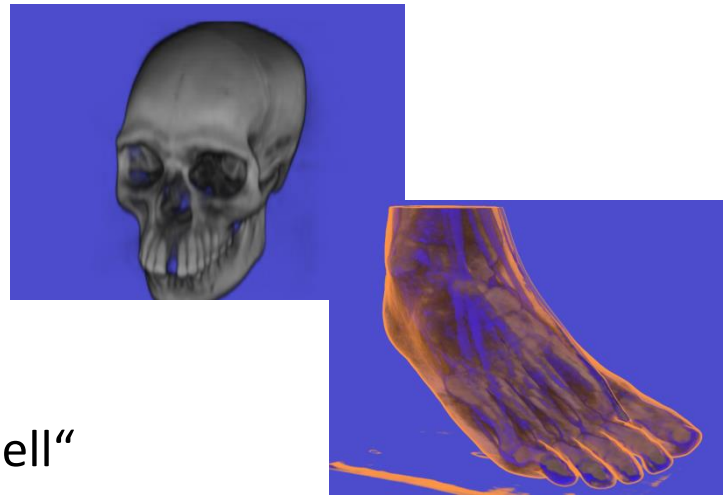
http://www.azom.com/article.aspx?ArticleID=10612
http://www.imaging-git.com/science/light-microscopy/hrem-high-resolution-episcopic-microscopy

Indirekte ⇔ Direkte Volumenvisualisierung

- Indirekte Volumenvisualisierung:
  - Extraktion einer Oberfläche aus Volumendaten, z.B. Isofläche
  - Siehe Polygonisierung!

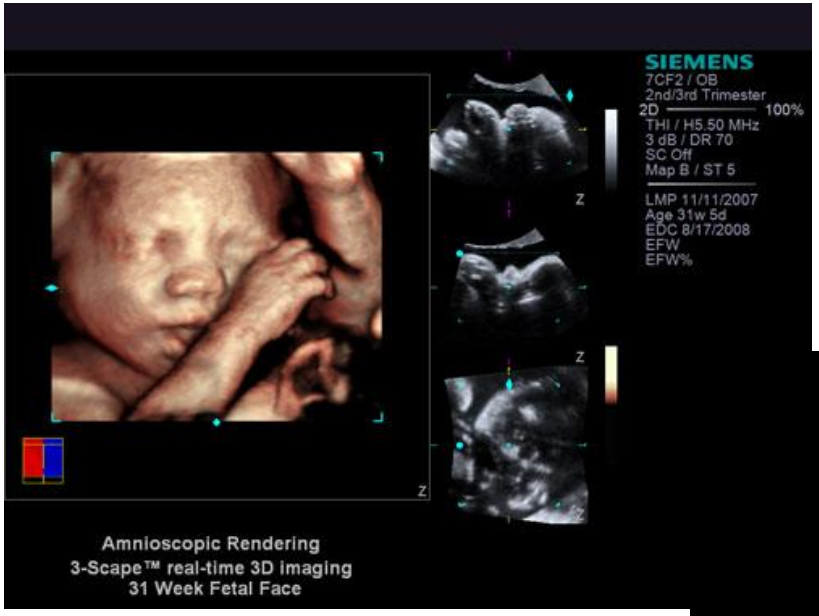


- Direkte Volumenvisualisierung:
  - Betrachtung des Intensitätswerts jedes einzelnen Voxels
  - Verwendung eines optischen Modells: Übersetzung des Intensitätswertes in physikalische Eigenschaften
  - Daraus Beschreibung der Interaktion des Lichts an jedem Voxel → „Optisches Modell“

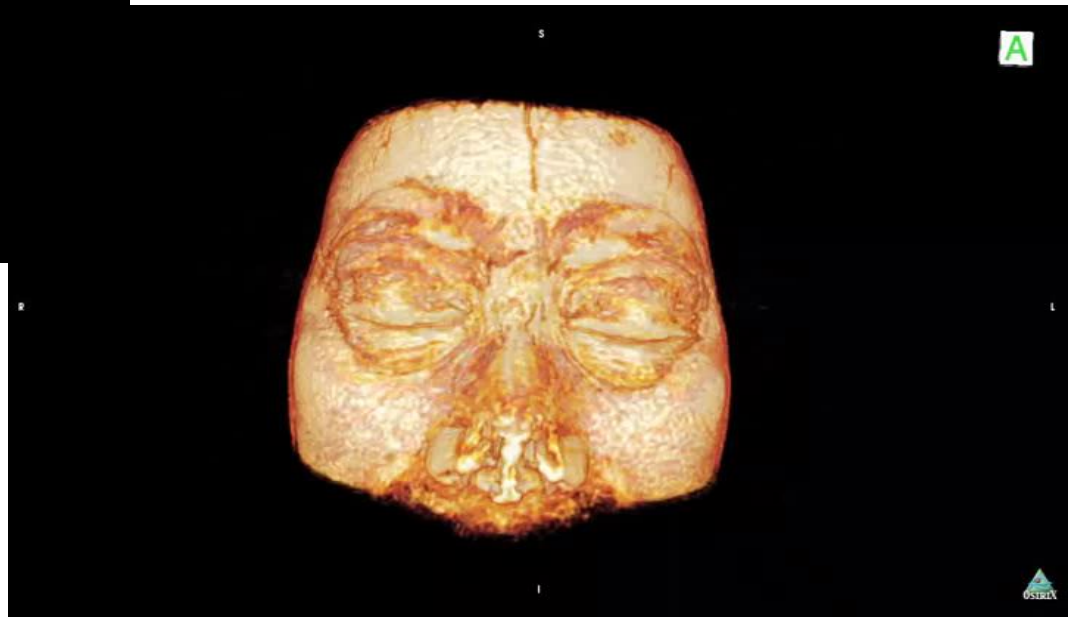


http://idav.ucdavis.edu/~okreylos/PhDStudies/Spring2000/ECS277/DataSets.html

Direkte Volumenvisualisierung in der Medizin



3D-Ultraschall in der Schwangerschaft



Visualisierung eines Aneurysmas aus MRT-Daten

http://www.nvidia.de/object/io_1259596736392.html
https://www.youtube.com/watch?v=v1fdoabjCWk

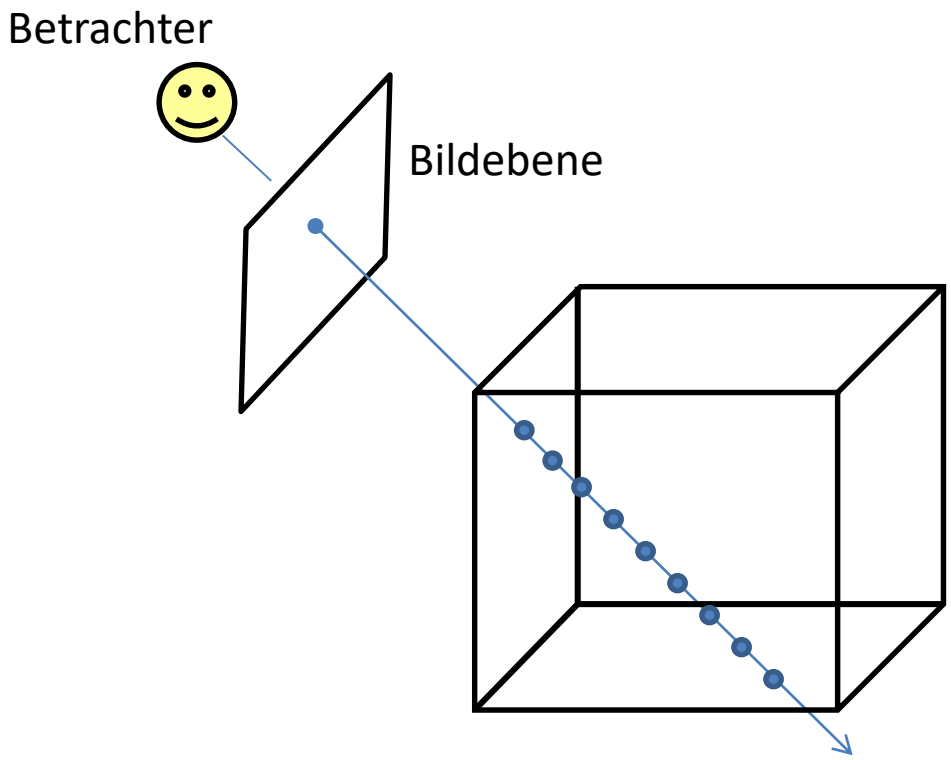
11.1 VOLUME RENDERING INTEGRAL

Optische Modelle

- Optisches Modell basiert auf Konzept des Strahlungstransportes: Modellierung des Verhaltens von Licht, das durch das Volumen läuft.
- Vereinfachung für die Volumenvisualisierung: Volumen = Ansammlung licht-emittierender und -absorbierenden Kugeln (=Voxel)
- Die wichtigsten Modelle sind:
  - **Nur Absorption:** Kugeln absorbieren nur Licht
  - **Nur Emission:** Kugeln emittieren nur Licht
  - **Absorption + Emission:** Kugeln emittieren Licht und absorbieren Licht, das auf sie trifft. Keine Streuung und Reflexion
  - **Streuung + Schattenwurf:** Teilweise Streuung des Lichtes an Kugeln, Schattenwurf auf verdeckte Kugeln

Grundidee der direkten Volumenvisualisierung

- Sichtstrahlen von jedem Pixel der Bildebene durch das Volumen
- Voxel des Volumens liefern definierte optische Dichtewerte (Emission, Absorption)
- Summation entlang der Sichtstrahlen



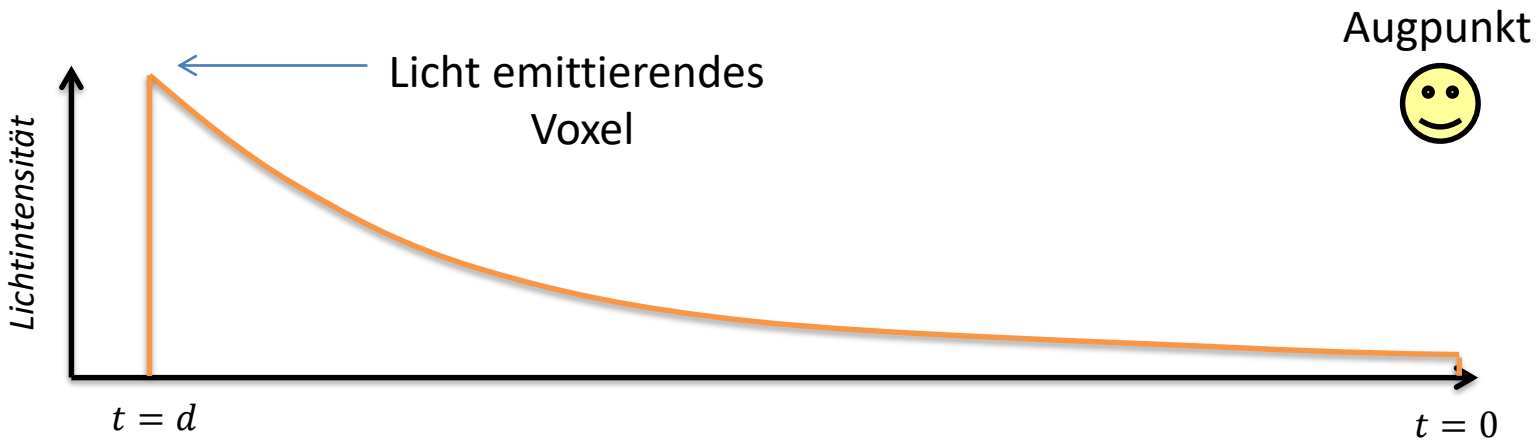
Volume Rendering Integral

- Volumenvisualisierungsalgorithmen lösen i.d.R. das Volume Rendering Integral (… mit mehr oder weniger Approximationen)
- Annahme für Herleitung: Volumen und Abbildung der optischen Dichte sind kontinuierlich.

- Definitionen

$\vec{x}(t)$	Strahl vom Augpunkt in das Volumen, t ist die Distanz zum Auge
$s\ (\vec{x}(t))$	Intensitätswert an der Position $\vec{x}(t)$
$c(t) \coloneqq c(s\ (\vec{x}(t)))$	Emission
$\kappa(t) \coloneqq \kappa(s\ (\vec{x}(t)))$	Absorption

Volume Rendering Integral



- Licht wird auf dem Weg von der Quelle zum Auge kontinuierlich absorbiert
- Einfachster Fall: κ konstant ⇒ von einem Voxel ausgestrahlte Intensität die am Augpunkt ankommt:

$$c^* = c \cdot e^{-\kappa d}$$

- I.d.R wird κ nicht als konstant angenommen, d.h. Absorption ist vom Raumpunkt abhängig:

$$c^* = c \cdot e^{-\underbrace{\int_0^d \kappa(t)\, dt}_{\text{= optische Tiefe}}}$$

Volume Rendering Integral

- Im Auge kommt nicht nur emittiertes Licht eines Voxels an, sondern von allen Voxeln!
- Dies wird abgebildet durch Integration über alle Positionen auf dem Sichtstrahl. Es ergibt sich das gesamte eintreffende emittierte Licht C

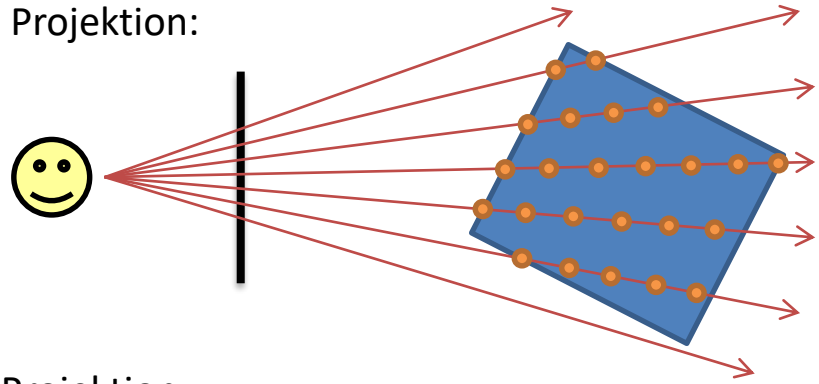
$$C = \int_0^\infty c(t) \cdot e^{-\int_0^d \kappa(\hat{t})\, d\hat{t}}\, dt$$

Volume Rendering Integral

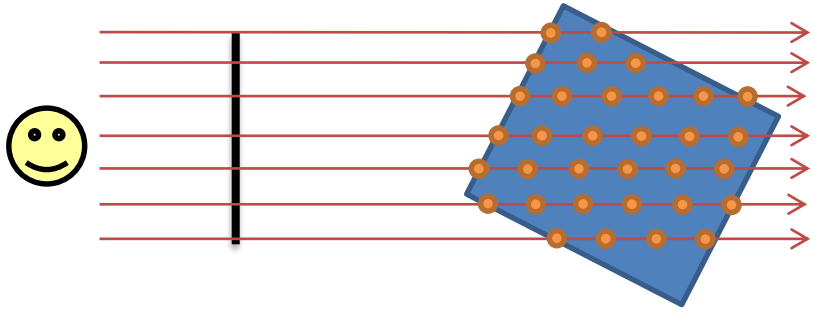
11.2 RAY CASTING

Ray Casting

- Numerische Lösung des Volume Rendering Integrals
- Von jedem Bildpunkt in der *near clipping plane* ausgehend Verfolgung eines Sichtstrahls durch das Volumen.
- Startpunkt der Sichtstrahlen wird durch Kameraeinstellungen bestimmt:
  - Perspektivische Projektion:



  - Orthografische Projektion:



Ray Casting

- Approximation des Volume Rendering Integrals durch Riemann-Summe zur Diskretisierung.
- Exponent des Volume Rendering Integrals:

$$-\int_0^t \kappa(t) \approx -\sum_{i=0}^{t/\Delta t} \kappa(i \cdot \Delta t) \cdot \Delta t$$

- Damit ändert sich das Volume Rendering Integral zu:

$$\tilde{C} = \int_0^\infty c(t) \cdot e^{-\sum_{i=0}^{t/\Delta t} \kappa(i \cdot \Delta t) \cdot \Delta t} \, dt$$

$$\tilde{C} = \int_0^\infty c(t) \cdot \prod_{i=0}^{t/\Delta t} e^{-\kappa(i \cdot \Delta t) \cdot \Delta t} \, dt$$

$$\tilde{C} = \sum_{i=0}^{n} c_i \cdot \prod_{j=0}^{t/\Delta t} e^{-\kappa(j \cdot \Delta t) \cdot \Delta t}$$

Ray Casting: Alpha Blending

- Die Berechnung des diskreten Volume Rendering Integrals kann nun als iterative ***Blending Operation*** zwischen den Abtastpunkten entlang eines Sichtstrahls beschrieben werden.
- Opazität A_j eines Abtastpunktes approximiert dessen ***Absorption***
- Intensität c_i eines Abtastpunktes approximiert dessen ***Emission***
- Mit $A = 1 - e^{-\int_0^d \kappa(t)dt}$ (= Opazität, Alphawert) ergibt sich die Näherungsformel

$$\tilde{C} = \sum_{i=0}^{n} c_i \cdot \prod_{j=0}^{\frac{t}{\Delta t}} 1 - A_j$$

Ray Casting: Alpha Blending

Zwei Strategien für die Berechnung:

- **Back-to-Front**:
  - Iterative Aufsummation von Volumenende zu Bildebene

$$c'_i = c_i + (1 - A_i)c'_{i+1}$$
$$c'_n = 0$$

- **Front-to-back:**
  - Iterative Aufsummation von Bildebene zu Volumenende

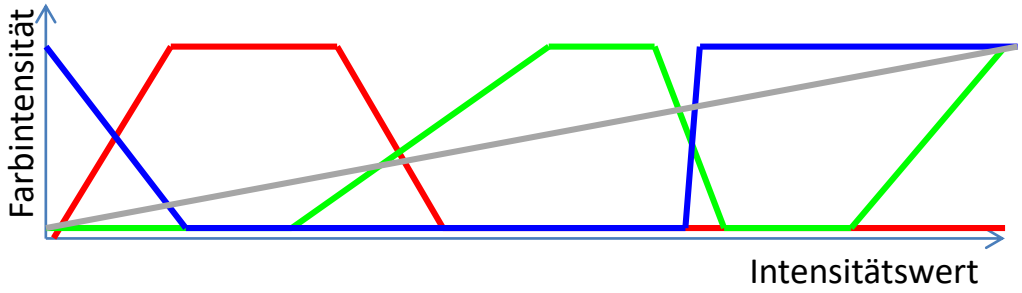
$$c'_i = c'_{i-1} + (1 - A'_{i-1})c_i \qquad c'_0 = 0$$
$$A'_i = A'_{i-1} + (1 - A'_{i-1})A_i \qquad A'_0 = 0$$

  - Erfordert neben Berechnung des Farbwertes c auch die Verfolgung des Opazitätswertes A: zusätzlicher Rechenaufwand
  - Vorteil wenn entlang eines Sichtstrahls vorzeitig abgebrochen werden kann, da z.B. bereits volle Deckkraft erreicht wurde

Ray Casting: Transferfunktion

- Abtastung des Volumens an diskreten Punkten entlang des Sichtstrahls
  - Trilineare Interpolation in $2 \times 2 \times 2$ Zelle
- Aufsummation der ermittelten Werte (vgl. Volume Rendering Integral)
  - Back-to-Front oder Front-to-Back
  - → Intensitätswert pro Pixel in der *near clipping plane*

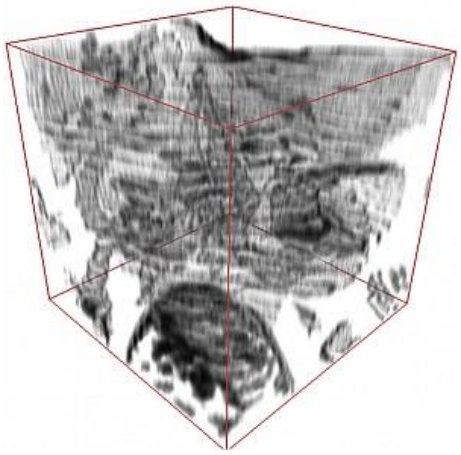
- Mapping der Intensitätswerte mit Transferfunktion: RGB und Alpha-Werte



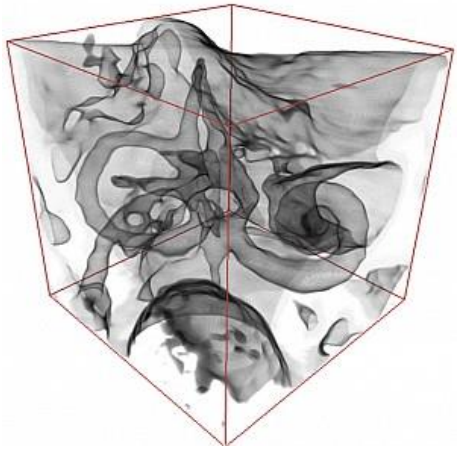
R, G, B-Kanal
+ Alpha-Kanal

Transferfunktion: Prä-/Postklassifizierung

- Prä-Klassifizierung
  - Vor Interpolation Umsetzung der Intensitätswerte des Volumens in Farbwerte und ggf. eine Opazität
- Post-Klassifizierung
  - Umsetzung des skalaren Summenwertes nach Interpolation in einen Farbwert und ggf. eine Opazität



Prä-Klassifizierung



Post-Klassifizierung

http://schorsch.efi.fh-nuernberg.de/roettger/index.php/VolumeRendering/Pre-AndPost-Classification

Ray Casting: Beleuchtung

- Zusätzlich Beleuchtung der Szene möglich (vgl. Beleuchtungsmodelle)
- Berechnung der Normalen für Volumendaten durch Approximation mittels Gradient:
  - Im kontinuierlichen Fall:

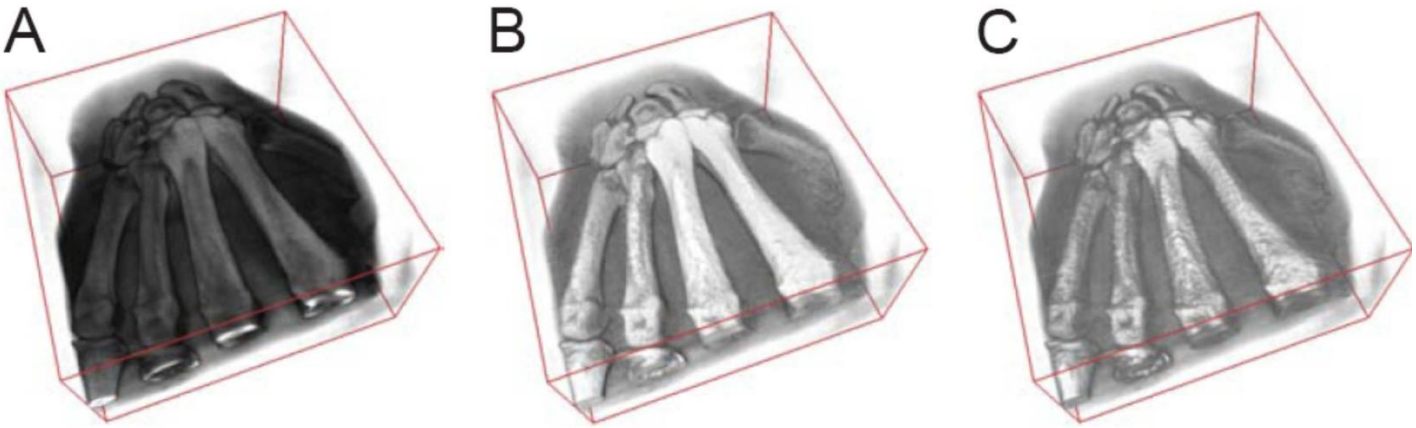
$$\nabla f(x,y,z) = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix}$$

  - Bei Voxeldaten Diskretisierung erforderlich:

$$\frac{\partial f}{\partial x} \approx \frac{f(x+1,y,z) - f(x,y,z)}{1}$$ (Vorwärtsdifferenz)

$$\frac{\partial f}{\partial x} \approx \frac{f(x+1,y,z) - f(x-1,y,z)}{2}$$ (Zentrale Differenz)

Ray Casting: Beleuchtung



(A) Ohne Beleuchtung (B) Diffuses Licht (C) Spekulares Licht

R. Erfrurt: „Vorintegriertes Volume Rendering: Slicing vs. Raycasting“, Seminar Medizinische Datenverarbeitung SS 2007- Lehr- und Forschungsgebiet Informatik 8 ”Computergrafik und Multimedia”, RWTH Aachen

Ray Casting

Pro

- Einfach zu implementieren
- Liefert sehr gute Ergebnisse

Contra

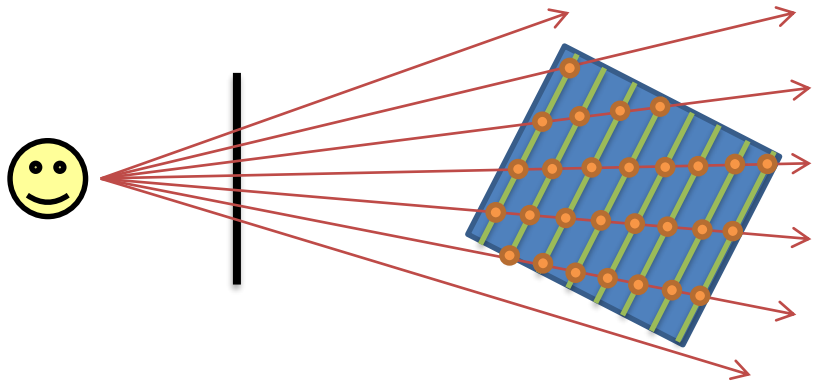
- Sehr rechenintensiv

https://de.wikipedia.org/wiki/Raycasting

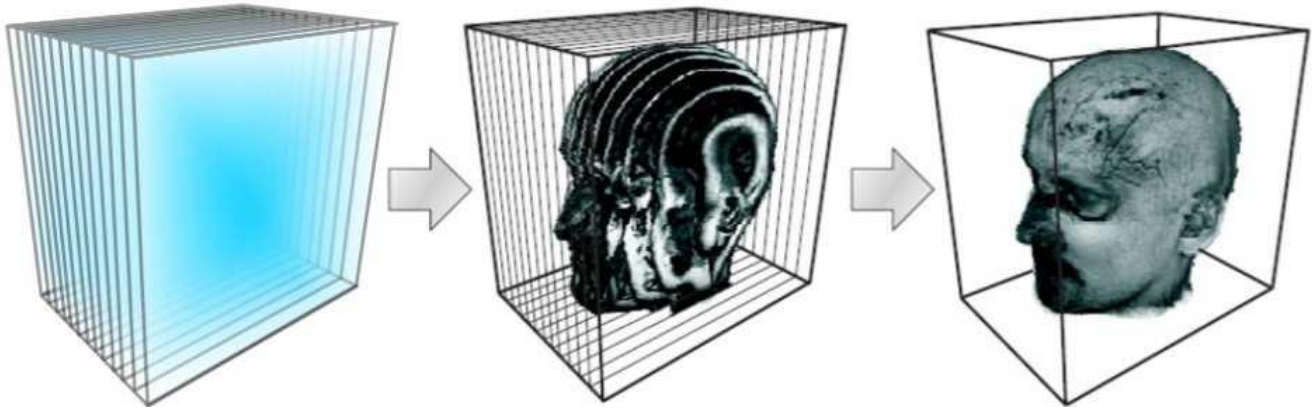
11.3 TEXTURE SLICING

2D Texture Slicing

- Ziel: Optimale Nutzung der GPU (Textur-Speicher) zur Beschleunigung der Berechnung
- Vorgehensweise
  - Zerteilung des Volumens entlang der Hauptachsen
  - Stapel von 2D Schichten → Speichern als 2D Texturen in der GPU
  - Automatische bilineare Interpolation



2D Texture Slicing

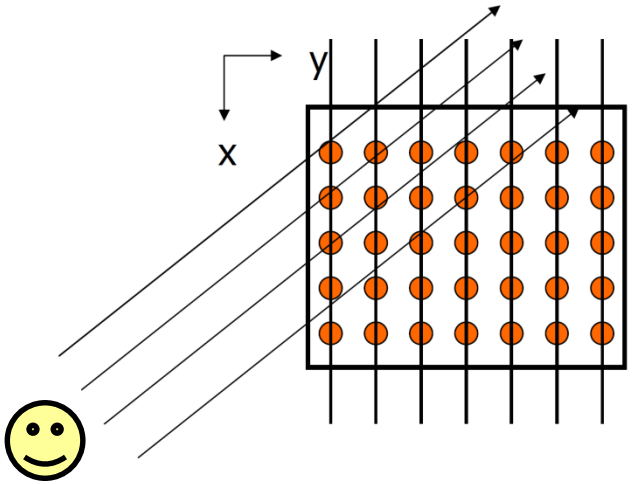


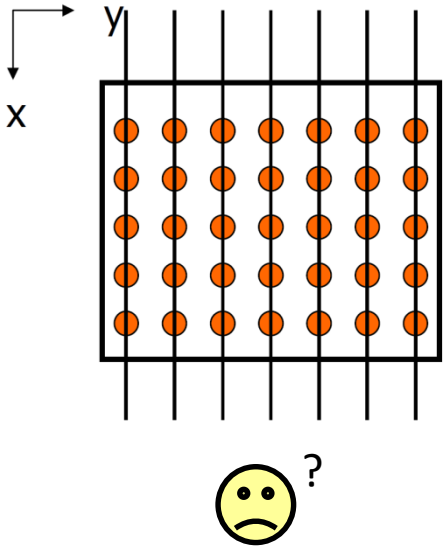
1. Unterteilung der Volumendaten in Schichten
2. Augpunkt und Blickrichtung festlegen
3. Definition von texturierten Polygonen parallel zu den Schichten
4. Rendern jeder Schicht als texturiertes Polygon
5. Blending-Funktion bestimmt Aggregation der Schichten

R. Erfrurt: „Vorintegriertes Volume Rendering: Slicing vs. Raycasting“, Seminar Medizinische Datenverarbeitung SS 2007- Lehr- und Forschungsgebiet Informatik 8 ”Computergrafik und Multimedia”, RWTH Aachen

2D Texture Slicing

- Was wenn sich der Blickwinkel ändert?
  - Ändern der Kameraposition oder Drehen der Polygone





➔ Ändern der Orientierung der Schichten!

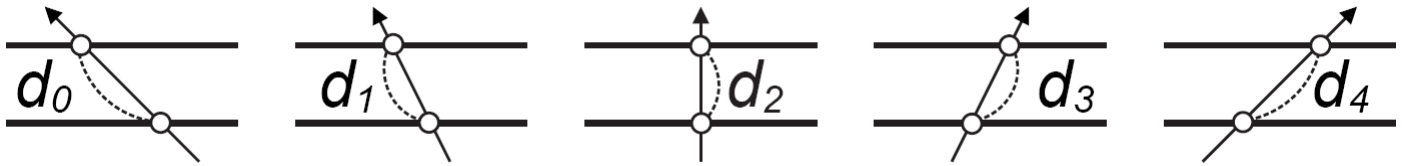
anhand maximaler Komponente des Sichtvektors, z.B. x maximal → yz-Ebene

- Oft werden für alle drei Schichtungsrichtungen texturierte Polygone im Speicher gehalten.

R. Gravis, Ohio State University, 2003

2D Texture Slicing

- Fester Abstand der 2D Schichten: Abtastrate ist bei perspektivischer Projektion nicht konstant über das Volumen
- Abtastrate verringert sich mit Abstand von der Hauptsichtachse



➔ Volumen wird in diesen Bereichen zu hell dargestellt

ACM Siggraph. Real-Time Volume Graphics, Los Angeles, California USA, 2004.

2D Texture Slicing

- Im Gegensatz zu Raycasting: Object-order-Verfahren
  - Ausgehend vom Objekt wird errechnet was dargestellt wird

Pro:

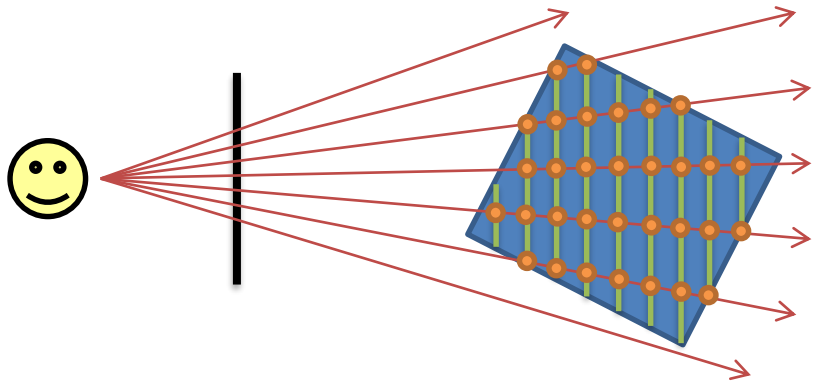
- Schnell
- Auf allen GPU hardware-beschleunigt

Contra:

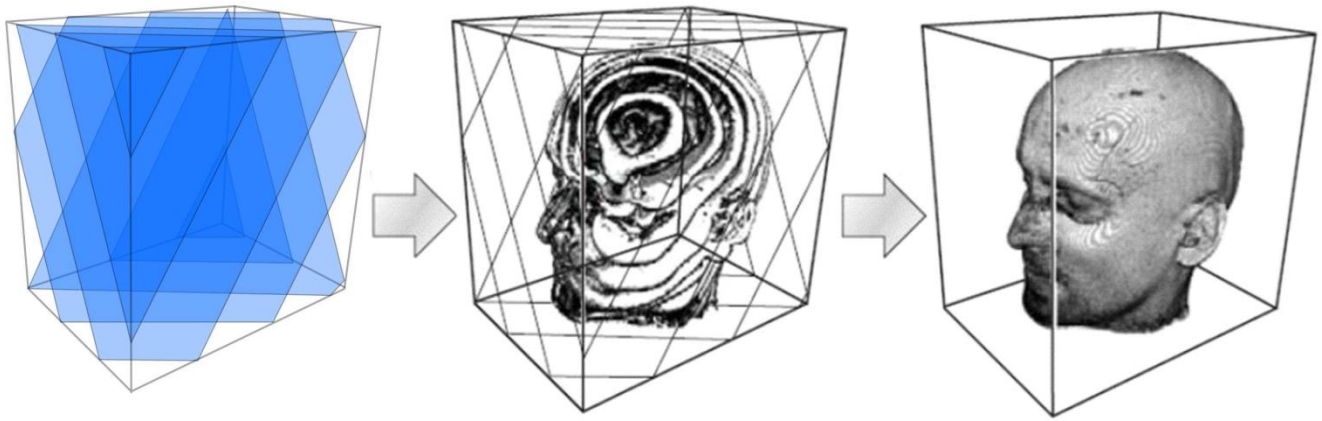
- Mäßige Ausgabequalität
- Artefaktbildung bei hohen Blickwinkeln
- Dreifache Datenmenge für alle Schichtungsrichtungen im Speicher

3D Texture Slicing

- Vorgehensweise:
  - Speichern einer dreidimensionalen Textur in der GPU
  - Berechnung von Texture Slices parallel zur Bildebene
  - Automatische **trilineare** Interpolation



3D Texture Slicing



1. Laden des Volumens in 3D Textur
2. Anzahl der Schichten festlegen, gängigerweise Voxelauflösung
3. Augpunkt und Blickrichtung festlegen
4. Polygone aus dem Volumen „schneiden“, Textur generieren, Orientierung festlegen
5. Rendern jeder Schicht als Polygon von der letzten zur ersten, Blending-Funktion bestimmt Aggregation der Schichten

R. Erfrurt: „Vorintegriertes Volume Rendering: Slicing vs. Raycasting“, Seminar Medizinische Datenverarbeitung SS 2007- Lehr- und Forschungsgebiet Informatik 8 ”Computergrafik und Multimedia”, RWTH Aachen

3D Texture Slicing

Pro:

- Gute Ergebnisse
- Keine Artefakte, unabhängig vom Blickwinkel

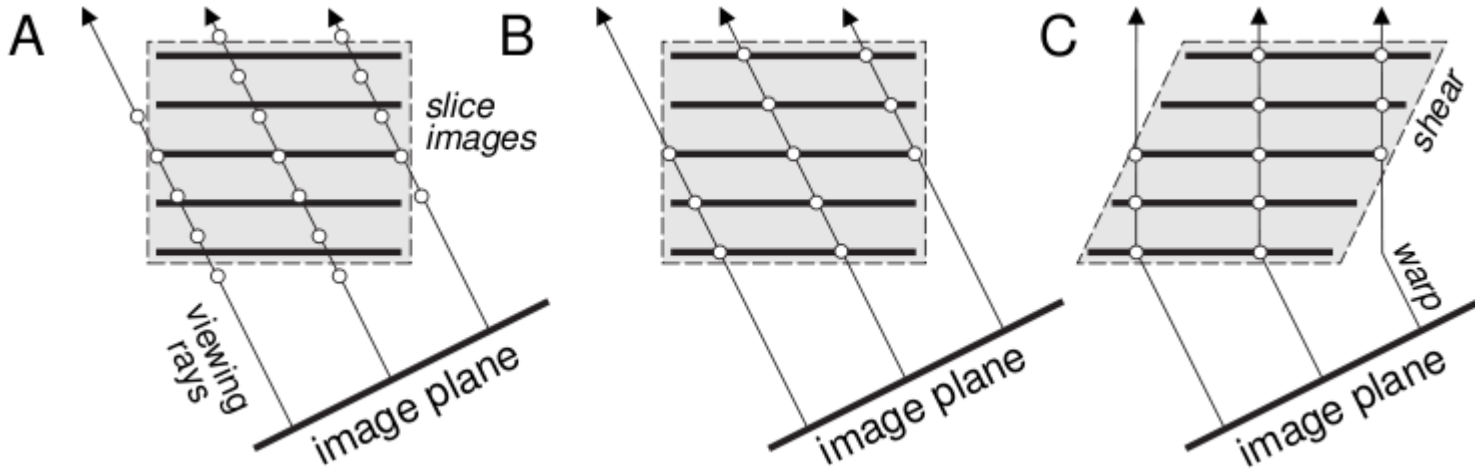
Contra:

- Langsamer, aufwändiger als 2D Texture Mapping
- Bei Blickwinkeländerung Neuberechnung der Schichten erforderlich
- Nicht auf allen GPU hardware-beschleunigt

11.4 SHEAR-WARP RENDERING

Shear-Warp-Rendering

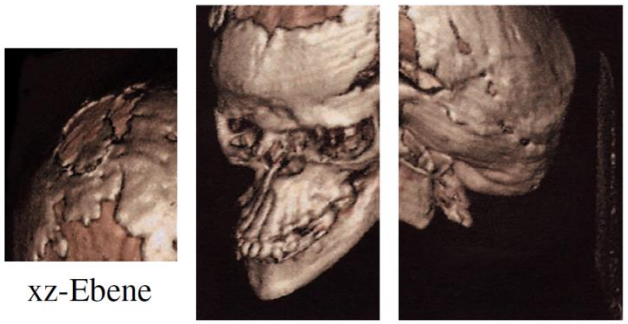
- Schneller Ansatz zum Berechnen des Volume Rendering Integrals, Alternative zu Ray Casting
- Projektion des gesamten Volumens auf die Bildebene
  - bilineare Interpolation ausreichend

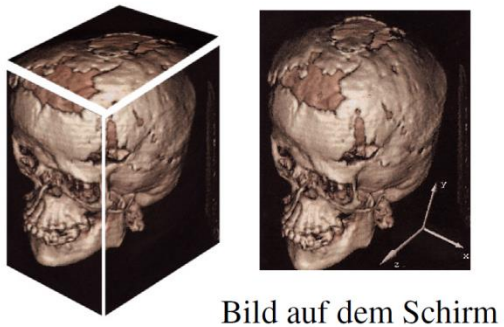


H. Jänicke: Visualisierung II, 2. Volumenvisualisierung, Universität Heidelberg

Shear-Warp-Rendering (orthografisch)

1. Scherung anhand des Winkels der Strahlen zum Volumen (shear)
   - 2D Resampling der Schichten
2. Projektion auf Basisebene (Parallele zum Volumen)
   - Durch das Scheren sind die Projektionsstrahlen orthogonal zur Basisebene!
   - Projiziertes Bild auf die Basisebene ist verzerrt
3. Verzerrung der Basisebenen auf die Bildebene (warp)
   - Orthographische Projektion: Zwischenbilder auf die drei dem Betrachter zugewandten Seiten eines zum Volumen achsenparallelen Quaders in den Proportionen der Bilddaten

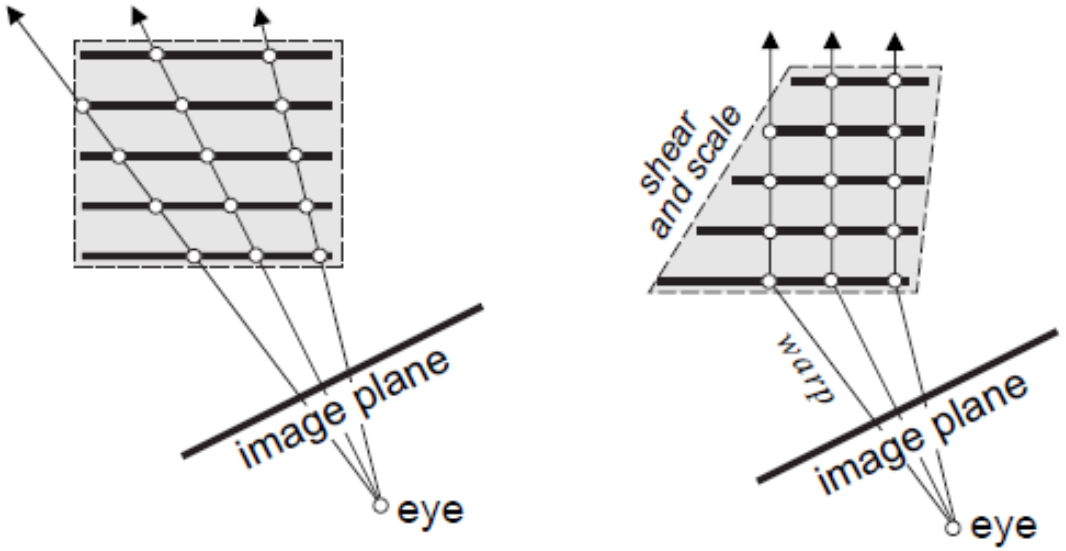




S. Krömker: Computergrafik I, Universität Heidelberg

Shear-Warp-Rendering (perspektivisch)

- Perspektivische Projektion: Verkürzung weiter entfernter Schichtbilder muss beim Scheren berücksichtigt werden
- Bei Projektion der Basisebenen auf die Bildebene: Verwendung eines perspektivisch erscheinenden Quaders

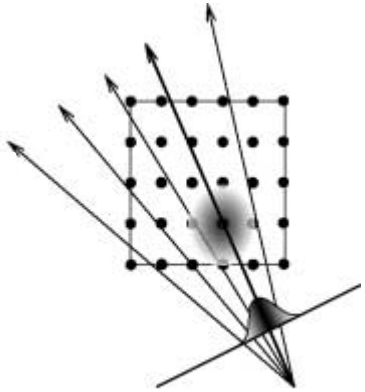


H. Jänicke: Visualisierung II, 2. Volumenvisualisierung, Universität Heidelberg

11.5. SPLATTING

Splatting

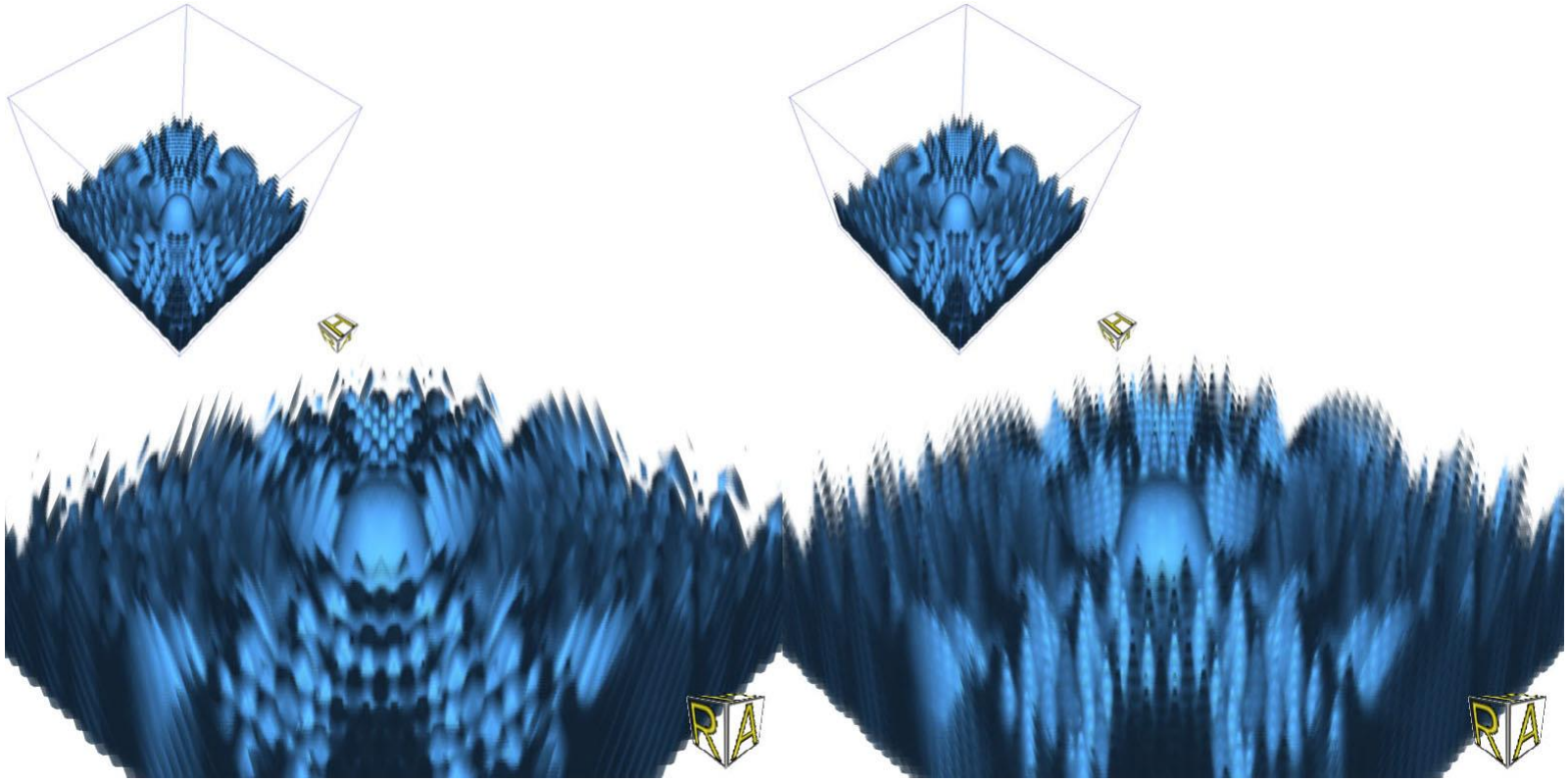
- Object-order-Methode: Jedes Voxel des Volumens wird auf Bildebene projiziert
  - Voxel = 3D Rekonstruktionsfilter
  - Integration der Rekonstruktionsfilter entlang eines Sichtstrahls



- Praktische Umsetzung:
  - Vor-Integration des 3D Rekonstruktionsfilters in Bildebene
    → 2D Rekonstruktionsfilter
  - Speichern des 2D Rekonstruktionsfilters im Textur-Speicher
  - Gewichtung der Textur mit Farbe und Opazität (Transferfunktion)
  - Aufbringen der Textur auf ein Proxypolygon (2D) an der Position des Voxels, parallel zur Bildebene
  - Aufsummieren der 2D Rekonstruktionsfilter durch Alpha-Blending

H. Pfister: „Moderne Volumenvisualisierung“, it – Information Technology 46 (2004) 3

Splatting



3D Texture Slicing

Volume Splatting

F. Vega-Higuera, P. Hastreiter, R. Naraghi, Fahlbusch R., and G. Greiner. High Performance Volume Splatting for Visualization of Neurovascular Data. 2006.

Splatting

Pro

- Geringer Speicheraufwand
- Verschiedene Rekonstruktionsfilter möglich
- Gute Wiedergabe hoher Frequenzen

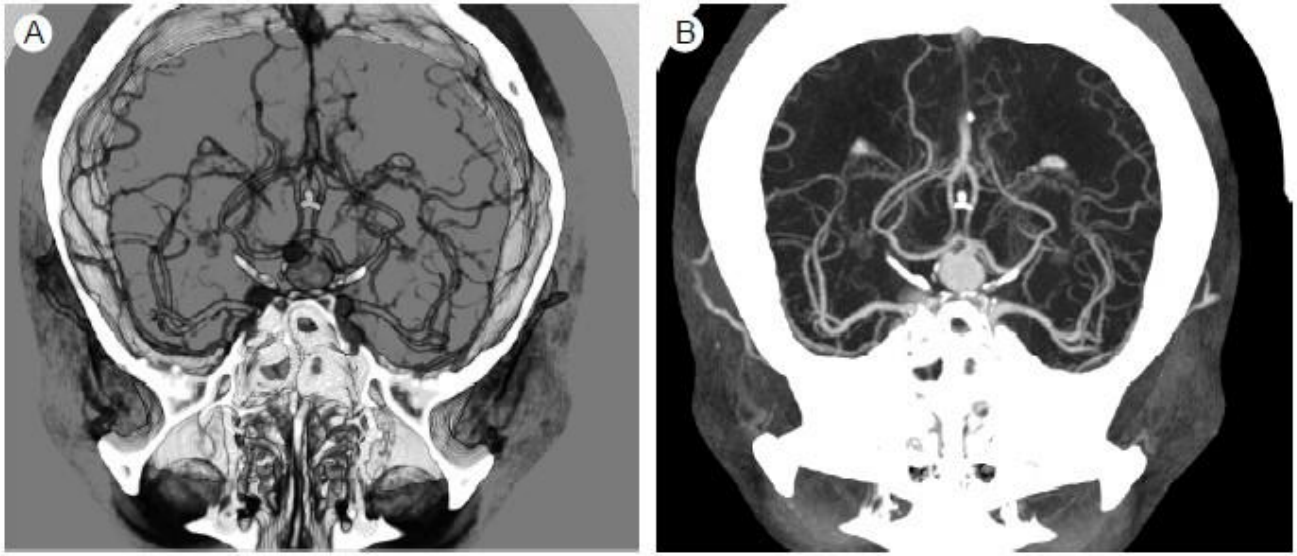
Contra

- Sehr viele Punkte müssen bearbeitet werden da jedes Pixel betrachtet wird → starke Belastung der Geometrieeinheit

11.6 MAXIMUM INTENSITY PROJECTION

Maximum Intensity Projection (MIP)

- Spezielle Art der direkten Volumenvisualiserung
- Statt lösen des Volume Rendering Integrals: höchster Intensitätswert entlang des Strahls



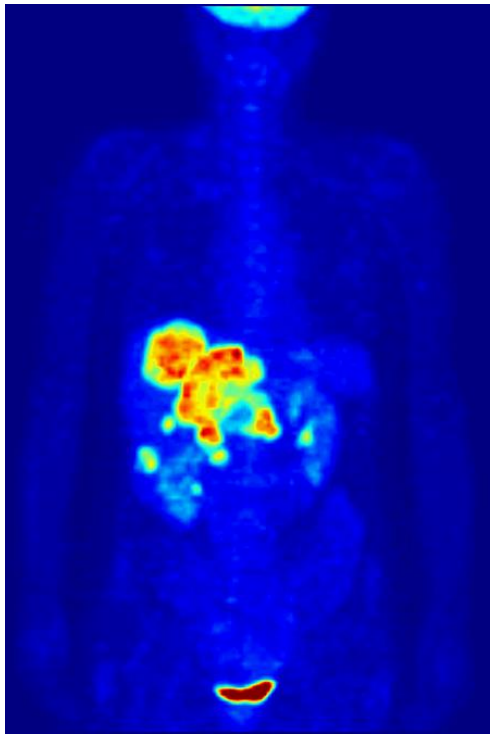
(a) Volume Rendering (b) Maximum Intensity Projection

H. Jänicke: Visualisierung II, 2. Volumenvisualisierung, Universität Heidelberg

Maximum Intensity Projection (MIP)

- Vor allem in der Medizin und Biologie verwendet, da Daten sehr verrauscht sind
- Gute Darstellung z.B. von Gefäßen in der CT-Angiografie, z.B. von Kontrastmittelsubtraktionen, PET-Aufnahmen
- Fehlender Tiefeneindruck
- Drei-dimensionaler Eindruck entsteht oft erst durch Rotation/Änderung der Projektionsrichtung

- Analog gibt es eine Minimum Intensity Projection



https://de.wikipedia.org/wiki/Maximumintensit%C3%A4tsprojektion

ZUSAMMENFASSUNG

Zusammenfassung

- Direkte Volumenvisualisierung:
  - Sichtstrahlen von jedem Pixel der Bildebene durch das Volumen
- Basis: Volume Rendering Integral $C = \int_0^\infty c(t) \cdot e^{-\int_0^d \kappa(\hat{t})\, d\hat{t}}\, dt$
- Ray Casting:
  - Approximation des Volume Rendering Integrals durch Diskretisierung
  - Berechnung durch Alpha Blending (Front-to-Back, Back-to-Front)
  - Transferfunktion: Intensitätswert → Farbwert (Prä- vs. Postklassifikation)
  - Beleuchtung: Normalenberechnung aus Gradient des Volumens
- 2D Texture Slicing / 3D Texture Slicing: Nutzung des Texturspeichers der GPU zur Beschleunigung der Interpolation
- Shear-Warp-Rendering: Scherung der Texturschichten Projektion auf Bildebene
- Splatting: 2D Rekonstruktionsfilter pro Voxel: Aufsummieren durch Alpha-Blending
- Maximum Intensity Projection: höchster Intensitätswert entlang des Strahls

ÜBUNGSAUFGABEN

Übungsfragen Kapitel 11

- Was ist der Unterschied zwischen direkter und indirekter Volumenvisualisierung?
- Was ist Prä- bzw. Post-Klassifizierung bei der Transferfunktion?