

Interaktive Computergrafik

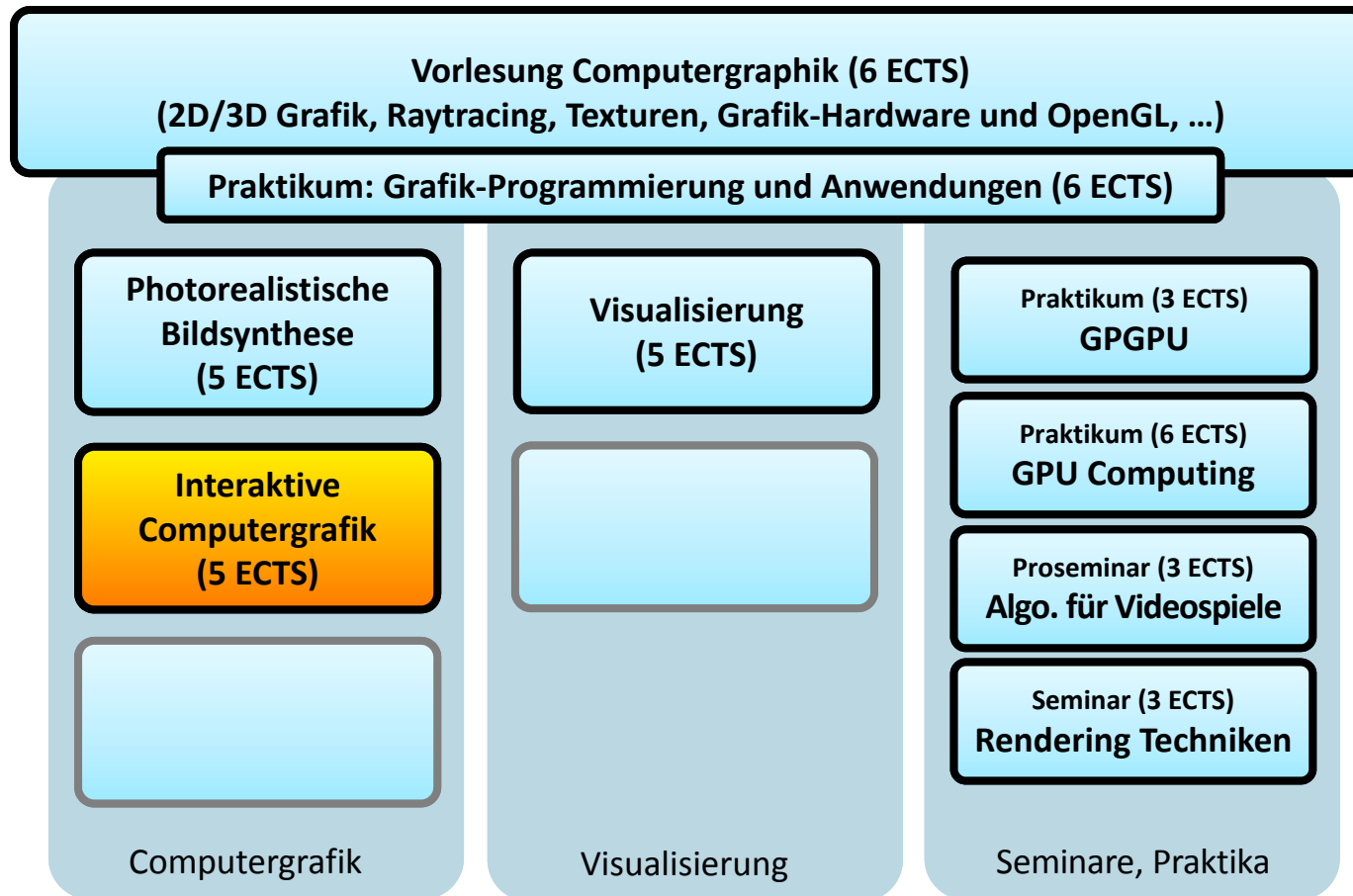
Vorlesung im Sommersemester 2014

**Kapitel 1: Einführung, Normal und
Displacement-Mapping**

Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie



Lehrangebot Computergrafik



HiWi

HiWi

HiWi

HiWi

Studien-/Diplomarbeiten bzw. BSc/MSc Arbeiten

Organisatorisches



- ▶ Vorlesung (2 SWS): Carsten Dachsbacher
 - ▶ Mittwoch 11:30 – 13:00 HS-101
 - ▶ Ankündigungen auf der Web-Seite beachten

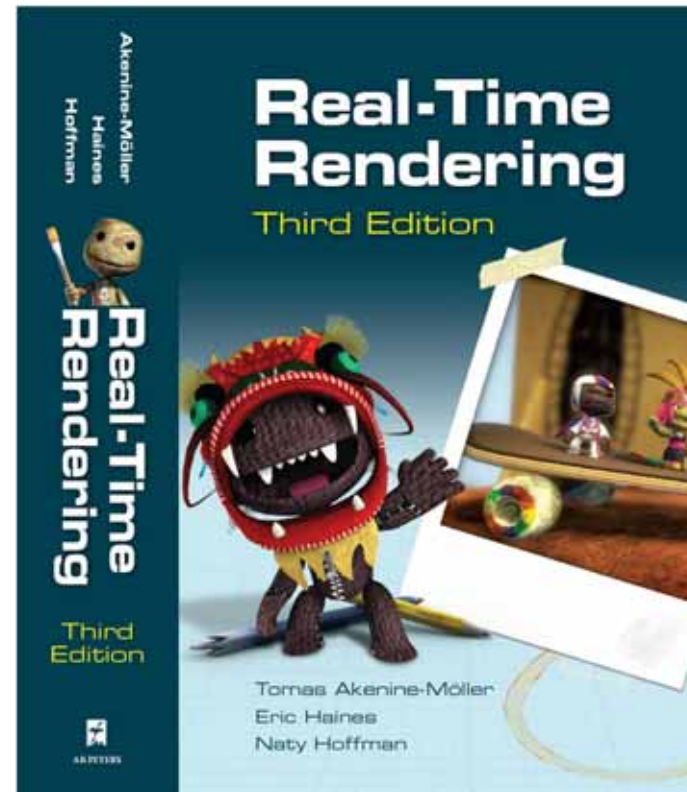
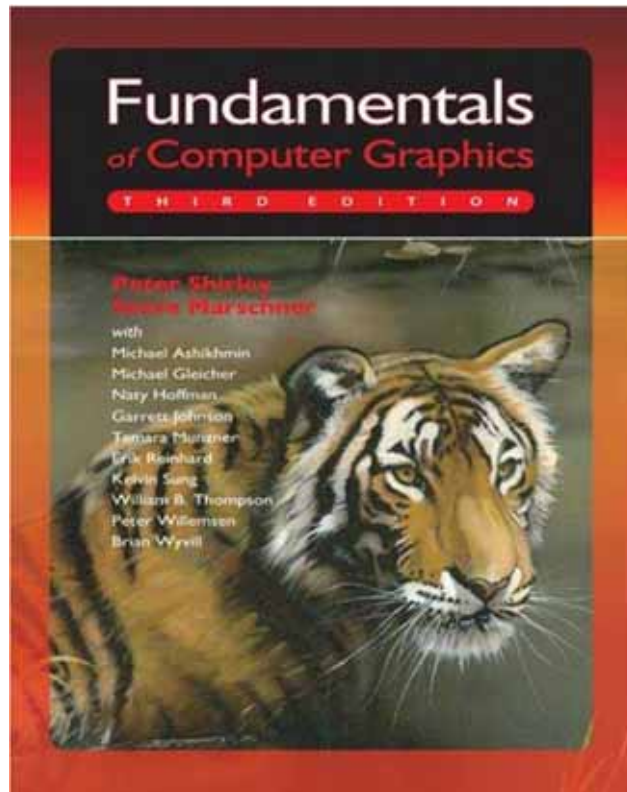
- ▶ Übungen (2 SWS): Florian Simon
 - ▶ Mittwoch 9:45 – 11:15 HS-101
 - ▶ mehrere Übungsblätter / Programmieraufgaben
 - ▶ Besprechung in der Tafelübung: **10:30 – 11:15**
 - ▶ **erste Übung: Mittwoch 23. April**
 - ▶ Unterstützung beim Praxisteil: **9.45 – 10.30**
 - ▶ die Grafikrechner im ATIS Pool werden reserviert



Literatur



- ▶ P. Shirley, S. Marschner: **Fundamentals of Computer Graphics**, 3rd Edition, AK Peters (Bibliothek, Google Books)
- ▶ T. Akenine-Möller, E. Haines, N. Hoffman: **Real-Time Rendering**, 3rd Edition, AK Peters (www.realtimerendering.com)



Organisatorisches



- ▶ Webseite des Lehrstuhls <http://cg.ivd.kit.edu/> → Lehrveranstaltungen
 - ▶ zentrale Informationsquelle
 - ▶ Aktuelle Infos, Folien und Übungsblätter, Termine
- ▶ Mailingliste: cg.info@ira.uka.de
 - ▶ Anmeldung → Mail an cg.info-join@ira.uka.de



KIT
Karlsruher Institut für Technologie

Lehrstuhl für Computergrafik

Lehrveranstaltungen im Sommersemester 2013

Vorlesungen		
Name	Termine	Dozenten
* Interaktive Computergrafik	Mittwoch, 09:45-11:15 wöchentlich HS -102 (UG) * 50.34 Informatik, Kollegiengebäude am Fasanengarten	Dachsbacher
* Übung zu Interaktive Computergrafik	Mittwoch, 08:45-09:30 wöchentlich HS -102 (UG) * 50.34 Informatik, Kollegiengebäude am Fasanengarten	Dachsbacher, Schmidt
* Fotorealistische Bildsynthese	Donnerstag, 15:45-17:15 wöchentlich HS -101 (UG) * 50.34 Informatik, Kollegiengebäude am Fasanengarten	Dachsbacher
* Übung zu Fotorealistische Bildsynthese	Donnerstag, 11:30-13:00 wöchentlich Raum 131 (1. Stock) * 50.34 Informatik, Kollegiengebäude am Fasanengarten	Dachsbacher, Meng

Unser Profil
Publikationen
Mitarbeiter
Forschung
Lehrveranstaltungen
SS 2013
Interaktive Computergrafik
Übung zu Interaktive Computergrafik
Fotorealistische Bildsynthese
Übung zu Fotorealistische Bildsynthese
Proseminar: Computergrafik
Appearance Acquisition and Fabrication
Praxis der Softwareentwicklung
GPU Computing
General-Purpose Computation on Graphics Processing Units

Organisatorisches

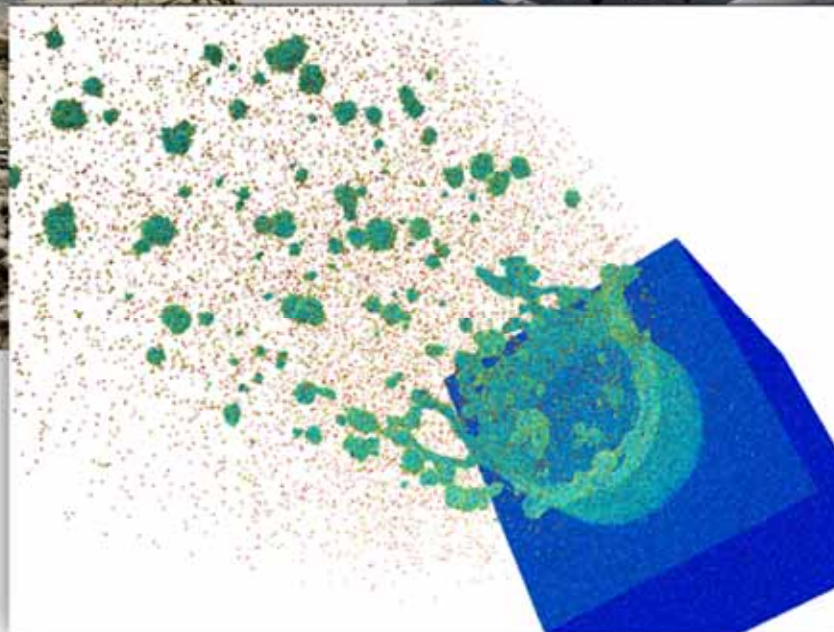
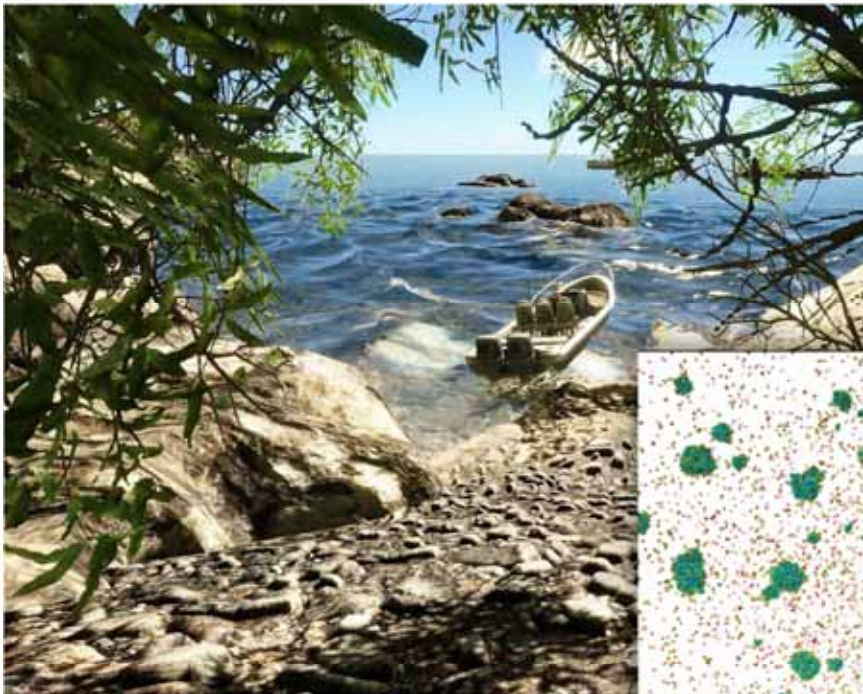


- ▶ Prüfung (mündlich)
 - ▶ Informatik Diplom: im Rahmen einer Vertiefungsprüfung
 - ▶ Informatik Bachelor: Wahlbereich
 - ▶ Informatik Master: Vertiefungsfach Computergrafik
- ▶ anderer (Diplom-)Studiengang?
- ▶ Bewertung der Übungsaufgaben hat keinen Einfluss auf die Note
 - ▶ ... aber Sie lernen eine Menge!
 - ▶ $5 \text{ LP} = 2\text{V} + 2\text{Ü}$

Interaktive Computergrafik



- ▶ interaktiv: <200ms pro Bild
- ▶ Anwendungen: Videospiele, Produktpräsentation/-design, Visualisierung...



Interaktive Computergrafik



► Anwendungen: Flug- und Fahrsimulatoren



Interaktive Computergrafik



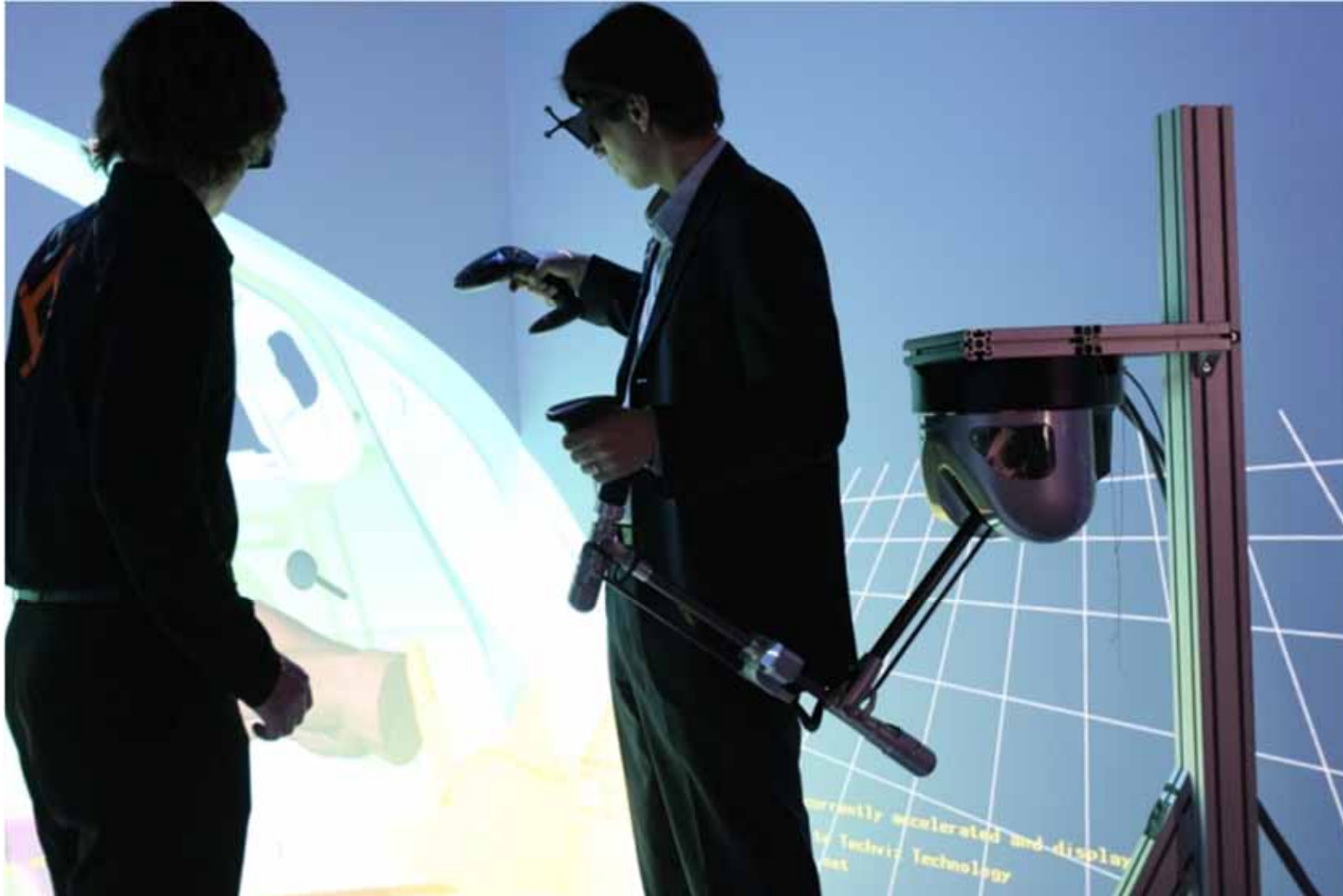
► Anwendungen: Virtual Reality



Interaktive Computergrafik



► Anwendungen: Virtual Reality



Interaktive Computergrafik

► Anwendungen: Virtual Reality



Game Developers Conference



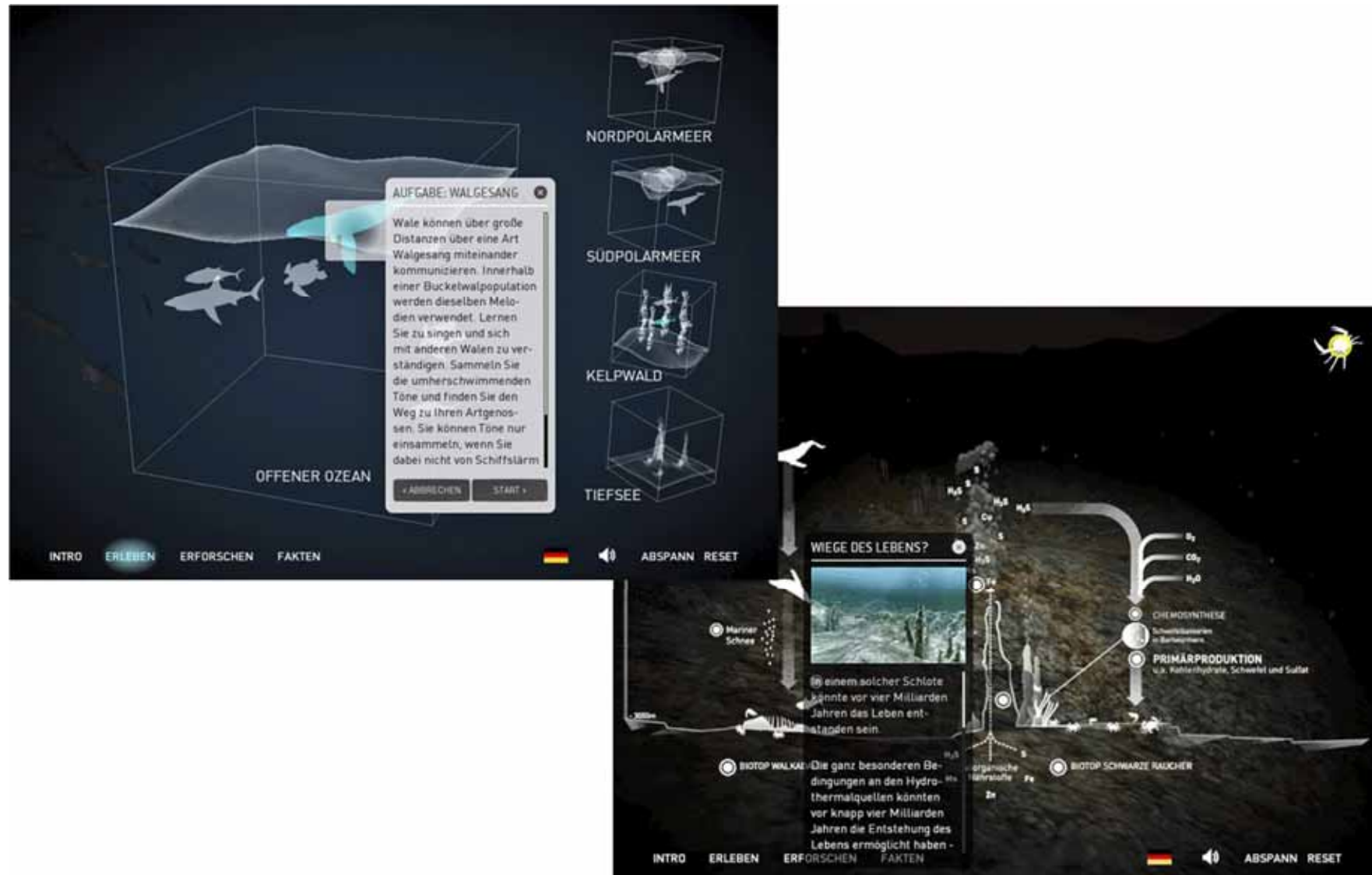
- ▶ Sony Project Morpheus / Oculus Virtual Reality Headset



Interaktive Computergrafik



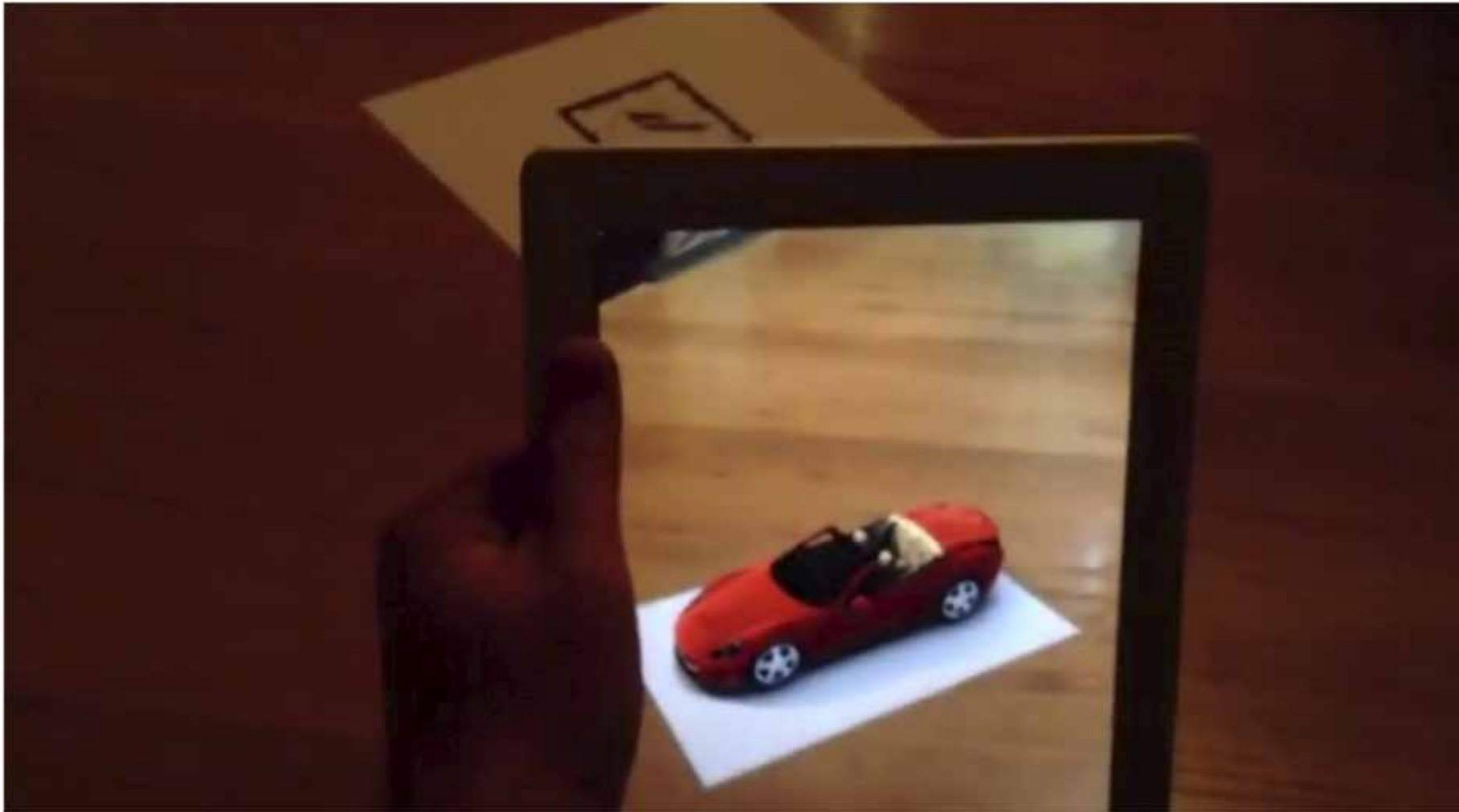
- Anwendungen: Serious Games („Universum der Ozeane“)



Interaktive Computergrafik



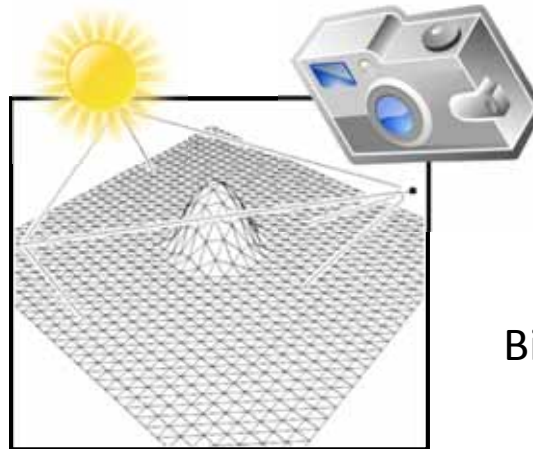
► Anwendungen: Augmented Reality



Previsualization/Lighting Design: Filmproduktionen



Computergrafik



→
Bildsynthese

Virtuelle Szene

(Geometrie, Material, Kamera, Lichtquellen, ...)



Betrachter der
Szene auf dem Monitor



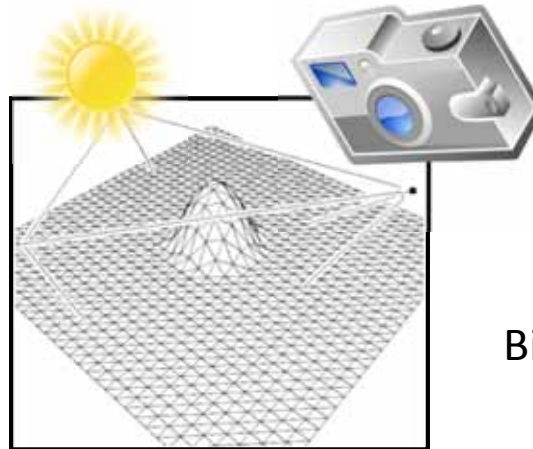
Monitor

←
Ansteuerung des
Monitors



Speicherung,
Verarbeitung,
Bildmanipulation,
Ausgabe

Computergrafik



➔
Bildsynthese

Virtuelle Szene

(Geometrie, Material, Kamera, Lichtquellen, ...)

... ergänzt durch reale Daten

(Texturen, Environment Maps, 3D Scanner, ...)

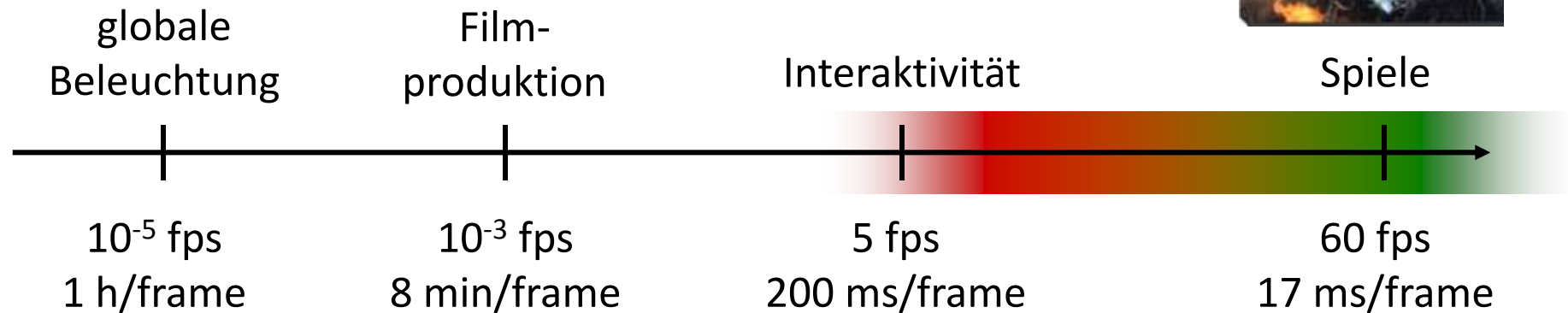


Speicherung,
Verarbeitung,
Bildmanipulation,
Ausgabe

„Randbedingung“



- ▶ wir verfolgen dieselben Ziele wie bisher: realistische Bilder
- ▶ aber: andere Anforderungen (als bspw. mit Raytracing)



OpenGL Recap, Shader, ...



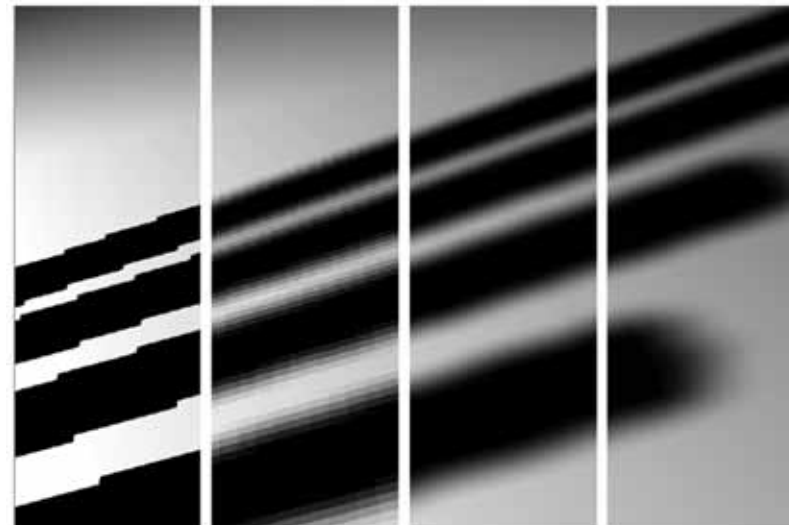
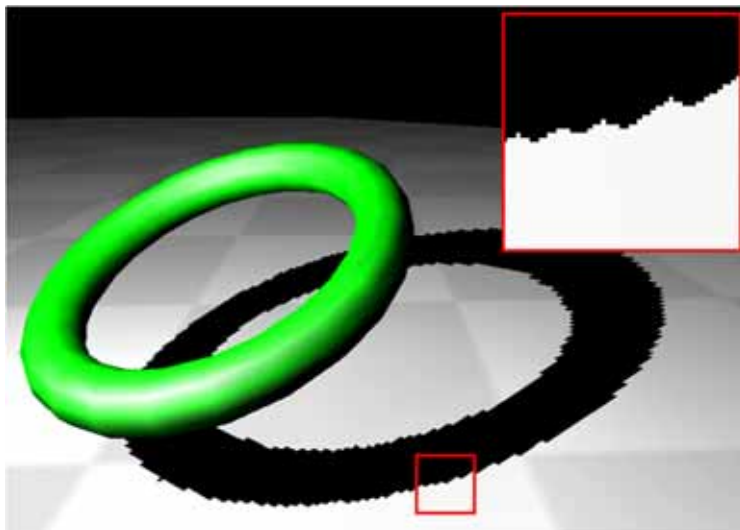
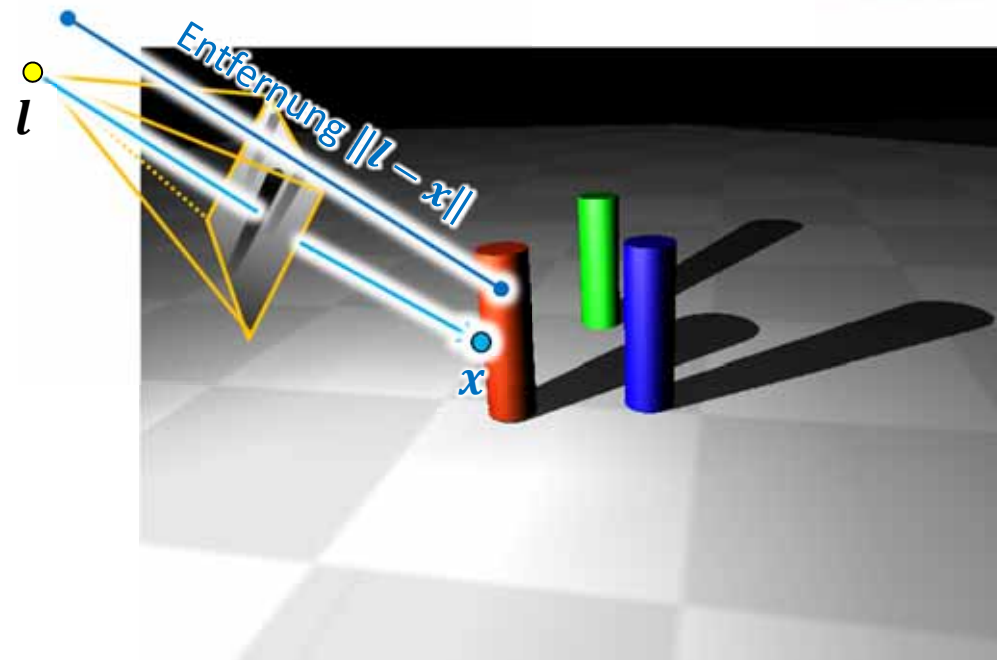
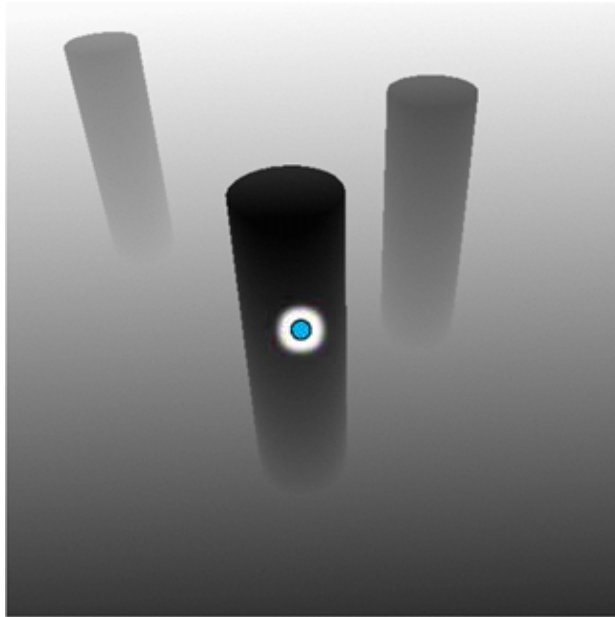
Oberflächendetails



Oberflächendetails



Echtzeit-Schatten-Verfahren



Interaktive Computergrafik



- ▶ wir möchten nach wie vor (meistens) fotorealistische Bilder berechnen
 - ▶ jetzt allerdings mit beschränkten Ressourcen
- ▶ was können wir also tun?
 - ▶ approximative Lösungen, z.B. bei indirekter Beleuchtung

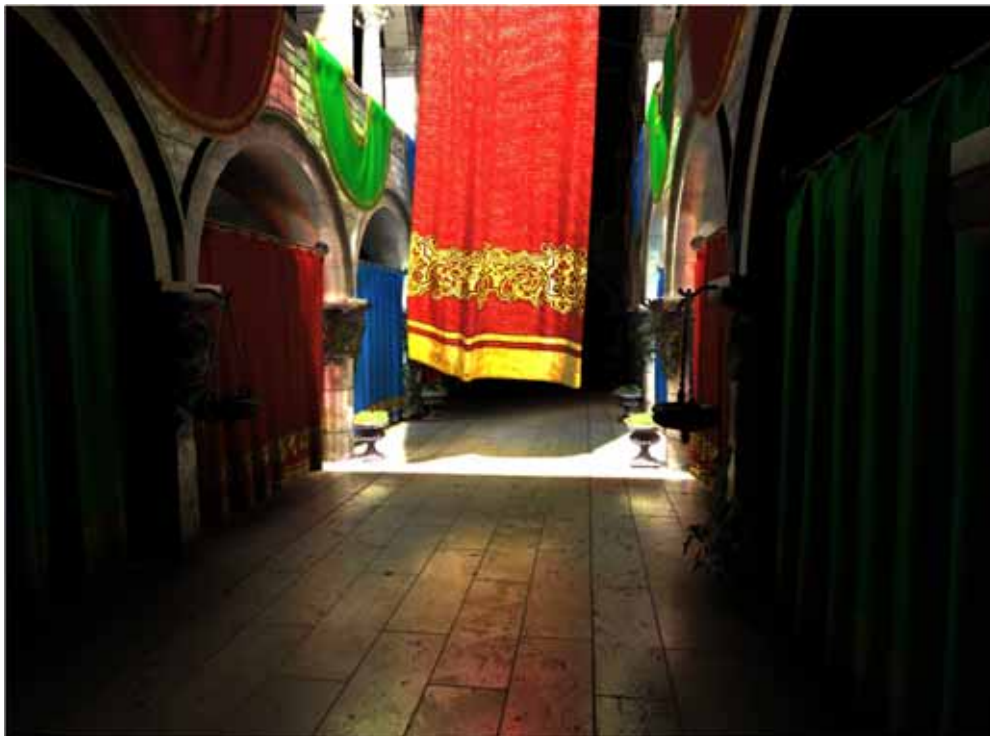
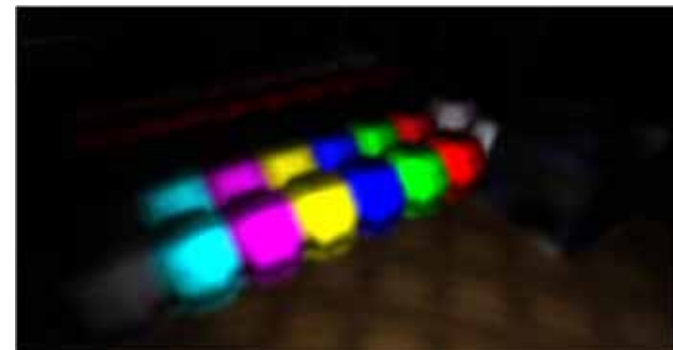
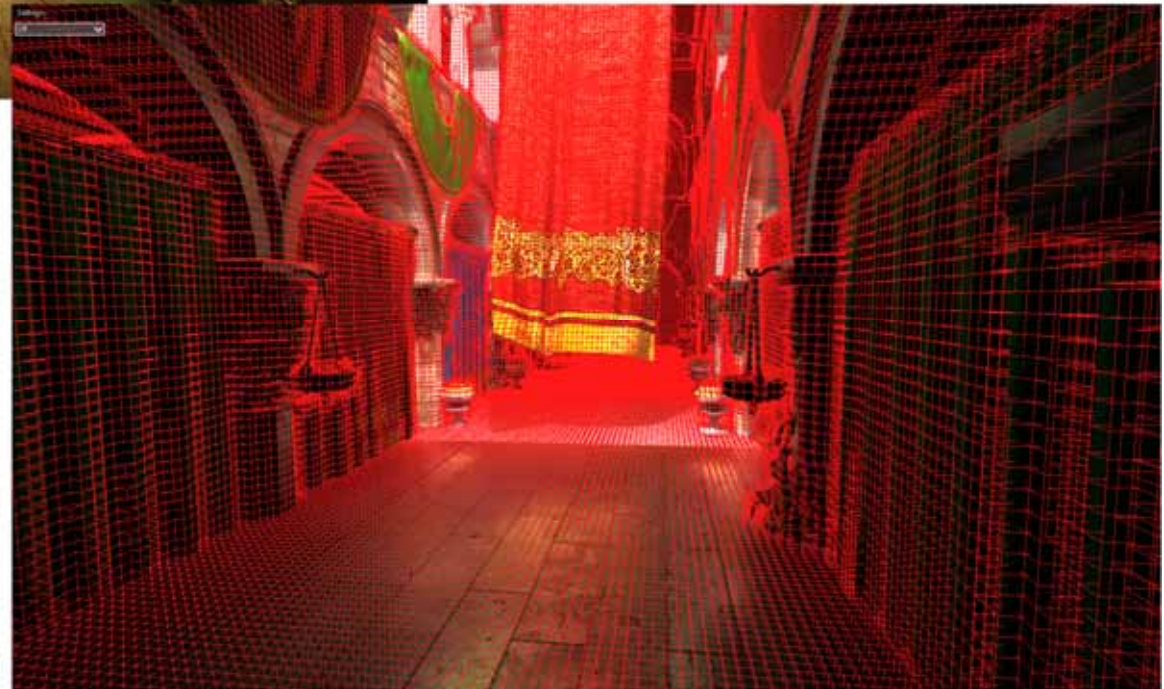
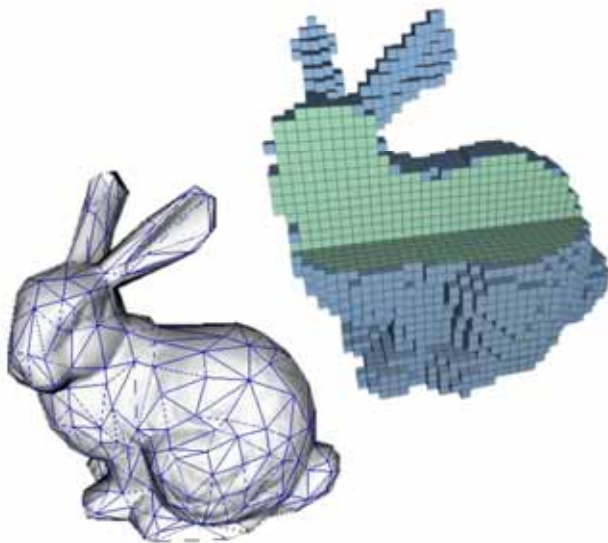


Bild: Cyril Crassin



Bilder: The Technology Behind the
“Unreal Engine 4 Elemental demo”, SIGGRAPH’12

Voxelisierung



Beispiel: **Real-time global illumination with voxel-based cone tracing**

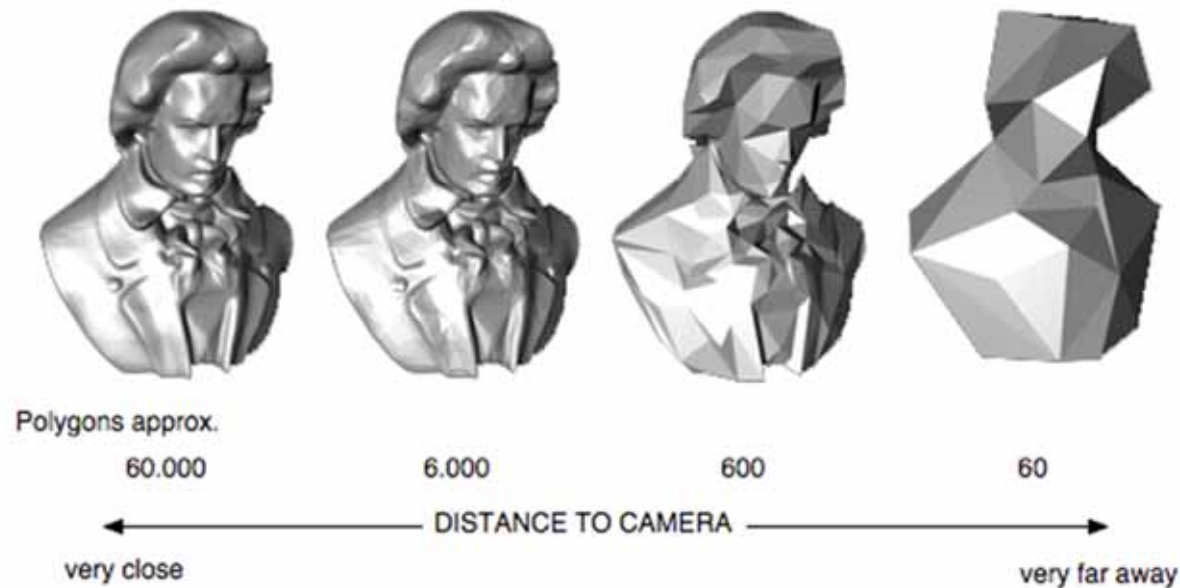
Crassin et. al, Pacific Graphics 2011

Voxelisierung: Anwendung



Beispiel: **Real-time voxel cone tracing** in Unreal Engine 4

- ▶ wir möchten nach wie vor (meistens) fotorealistische Bilder berechnen
 - ▶ jetzt allerdings mit beschränkten Ressourcen
- ▶ was können wir also tun?
 - ▶ nicht mehr Detail als nötig oder erkennbar
 - ▶ z.B. reduzierte Dreiecksnetze oder weniger Oberflächendetails



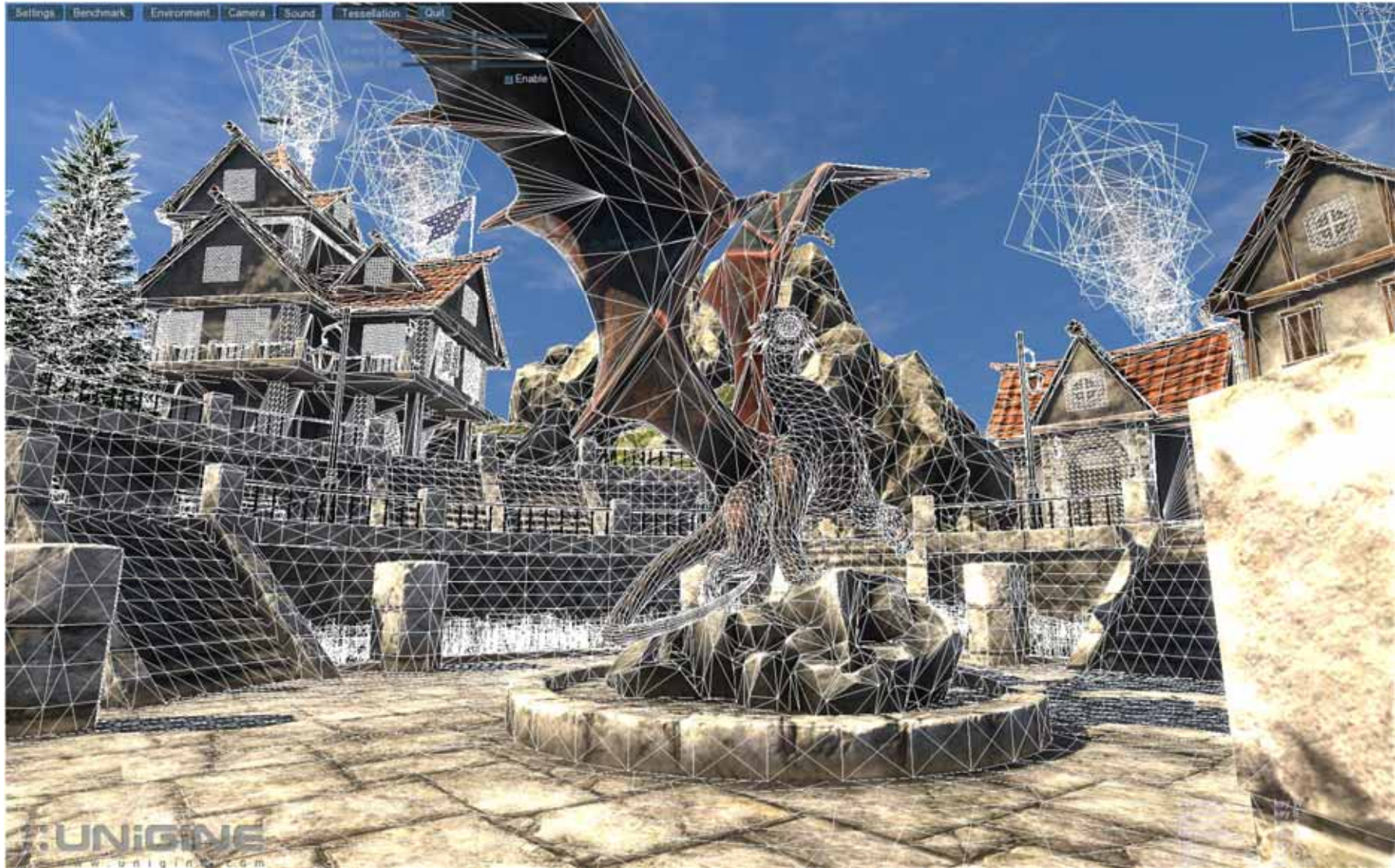
Level of Detail



Beispiel Tessellation – Unigine Benchmark



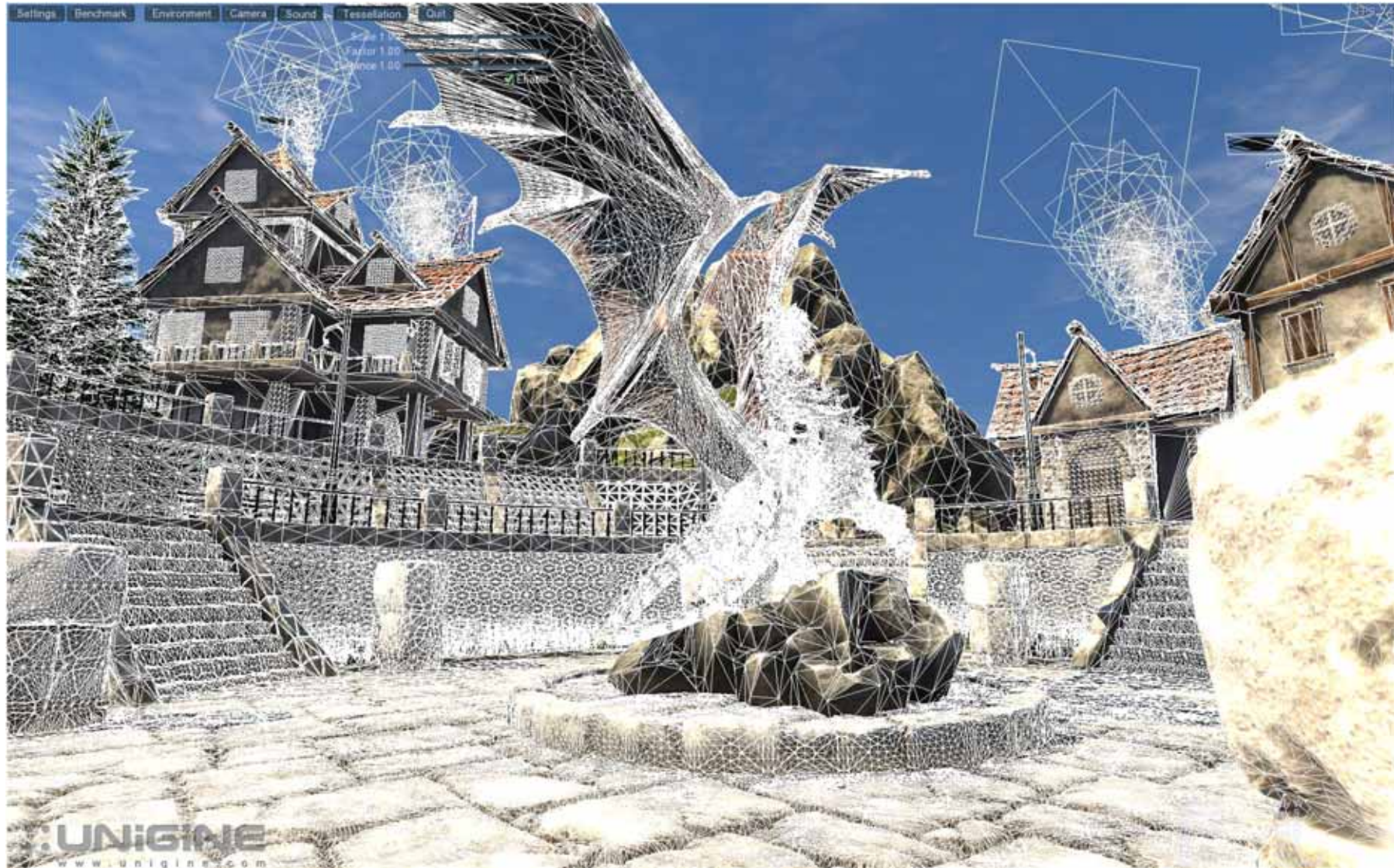
Eingabegeometrie vor der Unterteilung durch die Tessellation-Einheit



Beispiel Tessellation – Unigine Benchmark



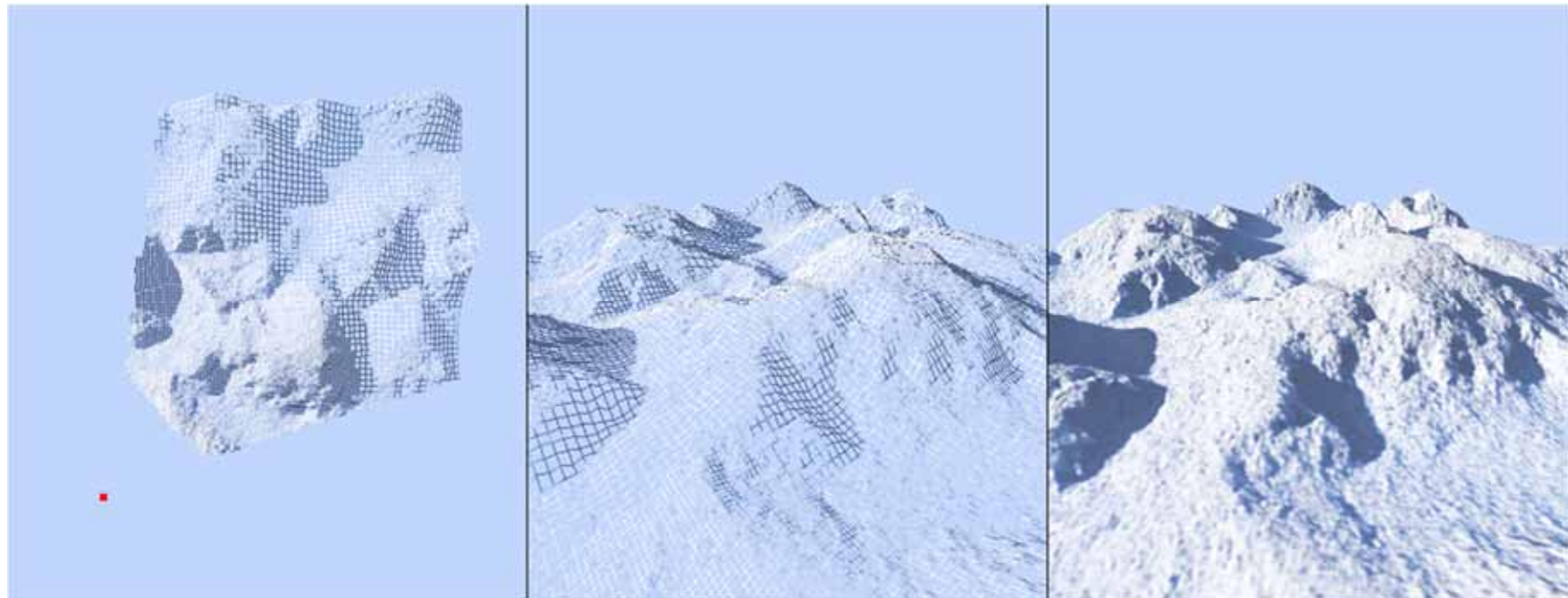
Displacement Mapping: Unterteilung und Verschiebung der neuen Vertizes



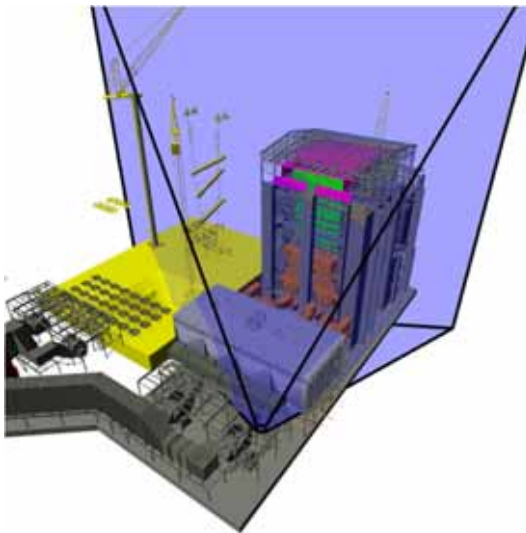
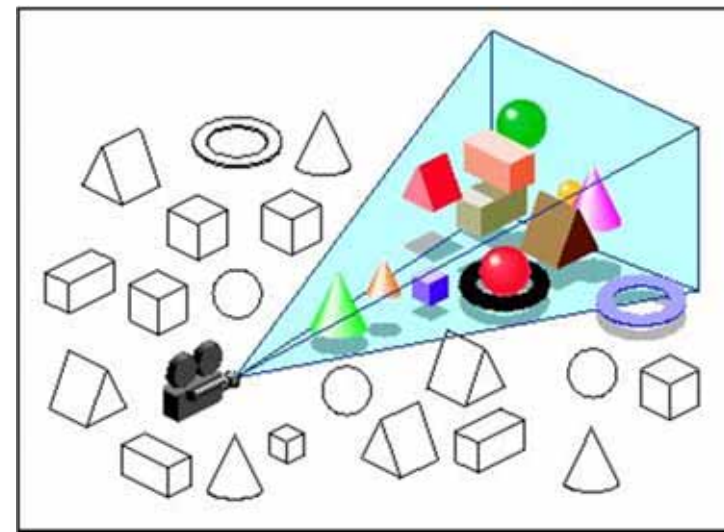
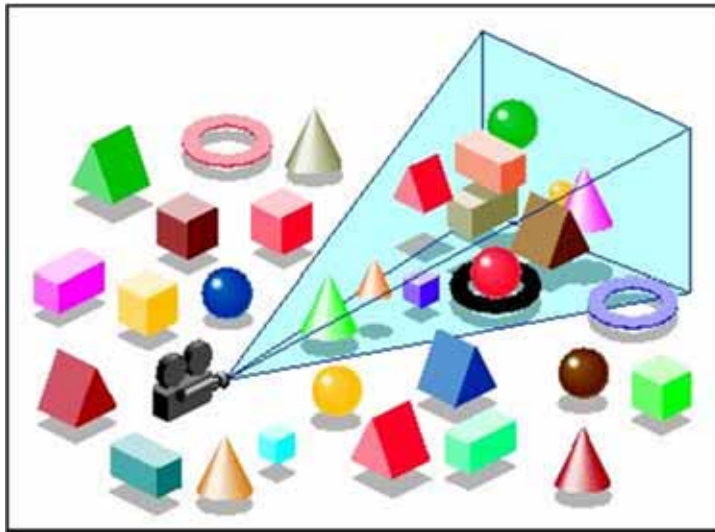
Interaktive Computergrafik



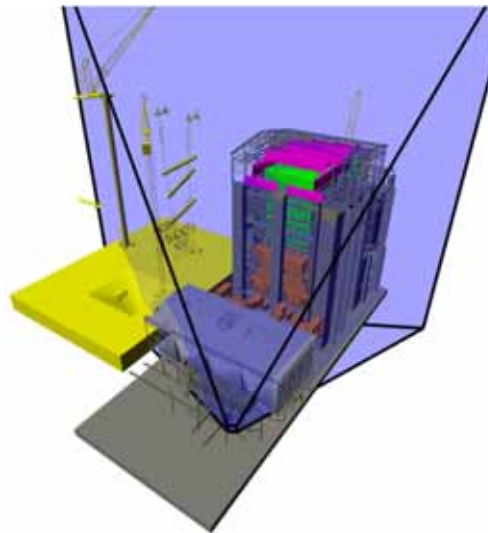
- ▶ wir möchten nach wie vor (meistens) fotorealistische Bilder berechnen
 - ▶ jetzt allerdings mit beschränkten Ressourcen
- ▶ was können wir also tun?
 - ▶ keine „unnötige“ Arbeit
 - ▶ z.B. Objekte außerhalb der Sichtpyramide nicht zeichnen



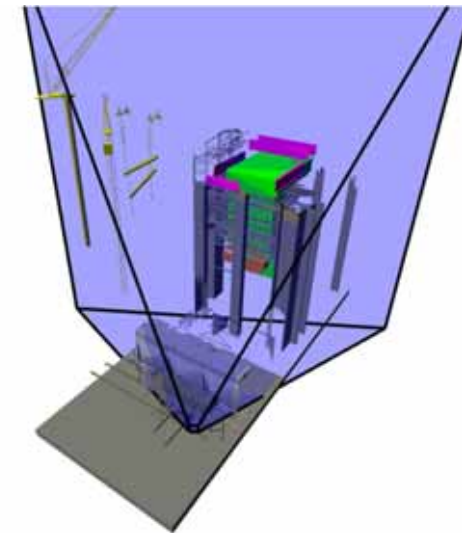
View Frustum und Occlusion Culling



gesamtes Modell, 1.7 Mio Δ



View Frustum Culling,
1.4 Mio Δ



Occlusion Culling,
89k Δ

Interaktive Computergrafik



- ▶ wir möchten nach wie vor (meistens) fotorealistische Bilder berechnen
 - ▶ jetzt allerdings mit beschränkten Ressourcen
- ▶ was können wir also tun?
 - ▶ keine „unnötige“ Arbeit
 - ▶ z.B. bei Shading, Texturierung, Anti-Aliasing etc.



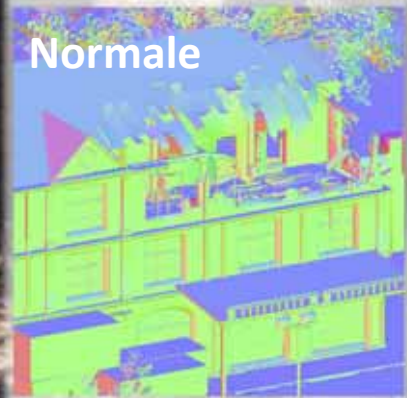
Deferred Shading



Tiefe



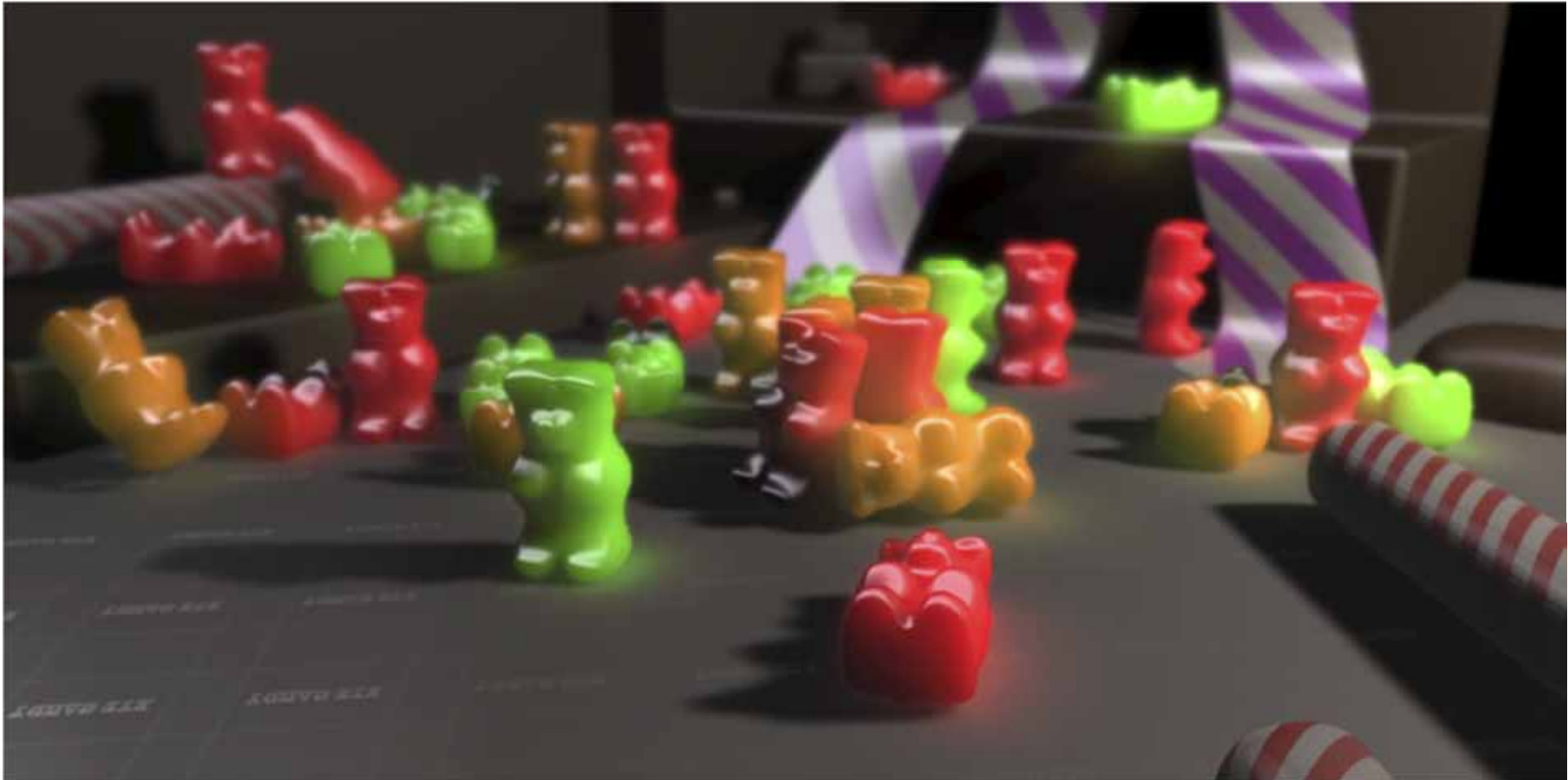
Normale



Farbe



Deep Screen Space



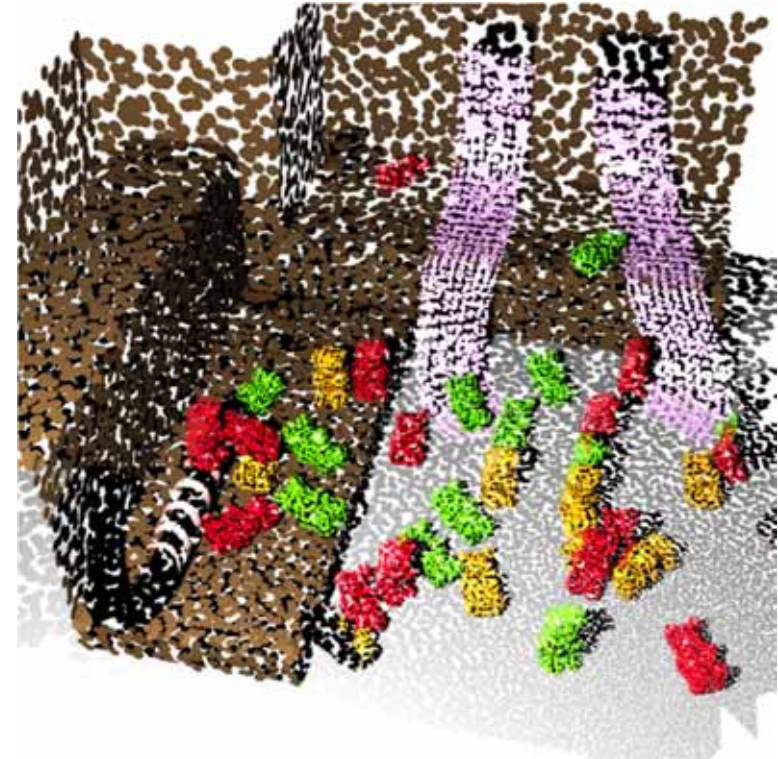
Deep Screen Space



Framebuffer



Screen space information



Deep screen space information

Tiefenunschärfe



Interaktive Globale Beleuchtung

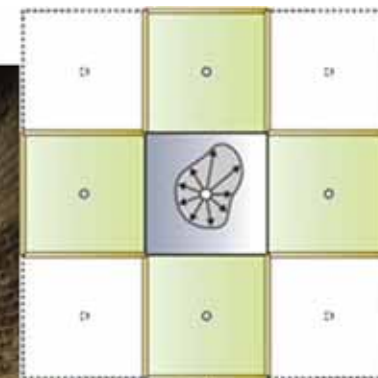
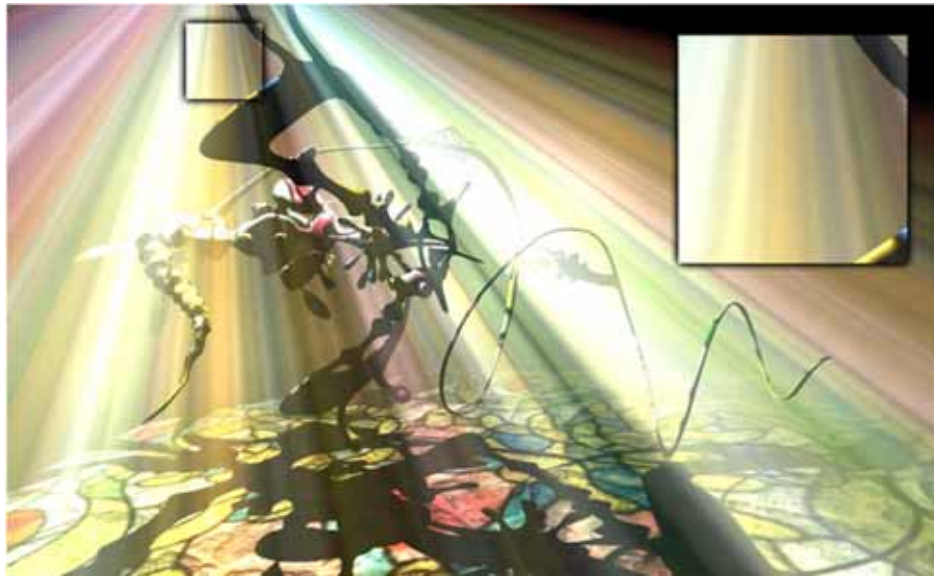
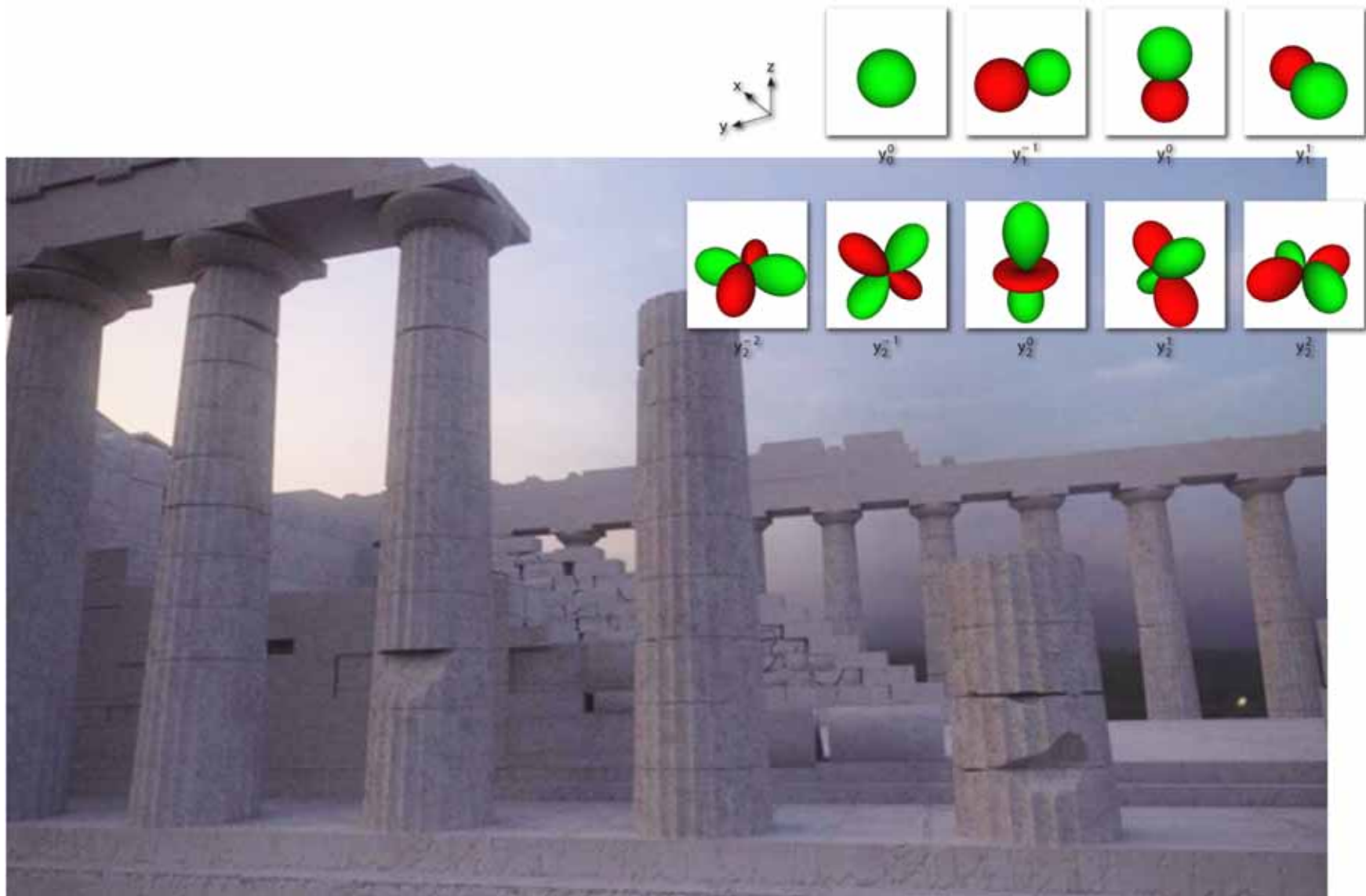


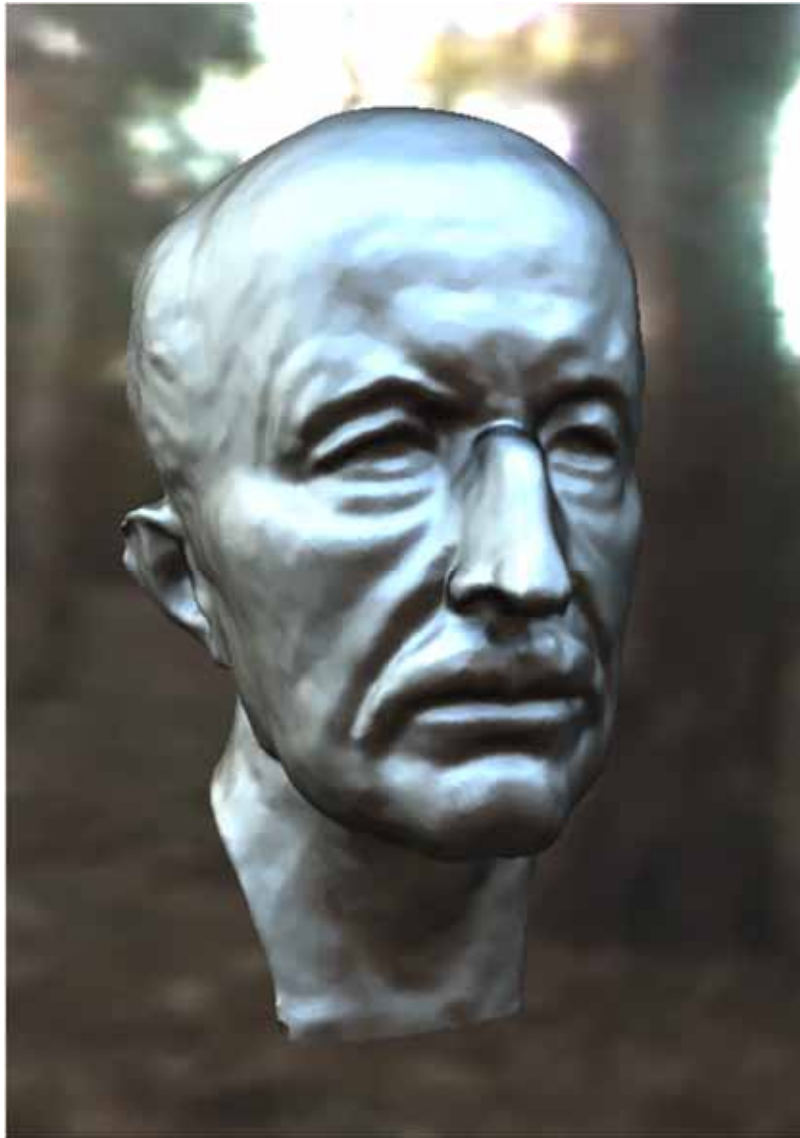
Image-Based Lighting



- Precomputed Radiance Transfer, Spherical Harmonics



Precomputed Radiance Transfer



Precomputed Radiance Transfer



Tone Mapping und Post-Processing

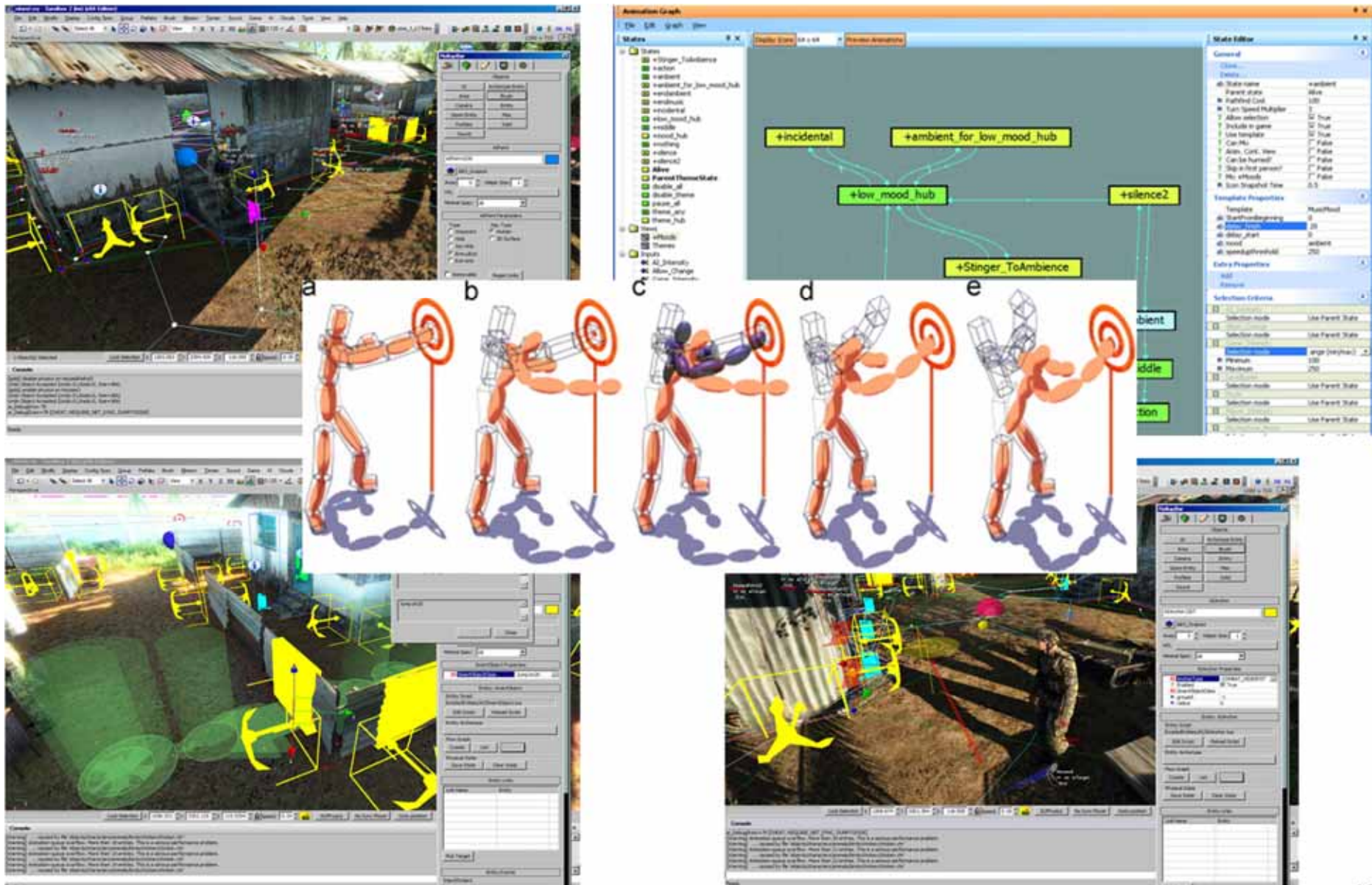


- ▶ Abbilden von berechneten Radiance-Werten auf Pixelfarben



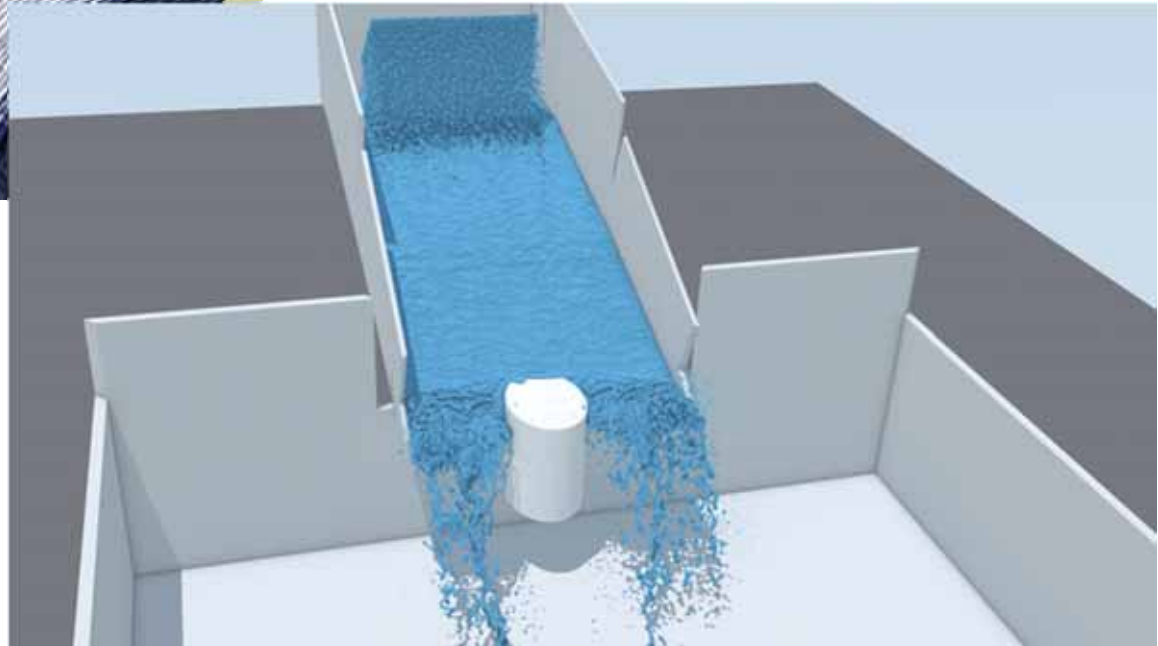
Bild: Lighting and Material of HALO 3, <http://www.bungie.net/inside/publications.aspx>

Animation und Szenengraphen



Nicht Thema dieser Vorlesung...

Non-photorealistic Rendering



Smoothed Particle Hydrodynamics

Interaktive Computergrafik



- ▶ wir möchten nach wie vor (meistens) fotorealistische Bilder berechnen
 - ▶ jetzt allerdings mit beschränkten Ressourcen
- ▶ was können wir also tun?
 - ▶ verwende schnelle/dedizierte Grafik-Hardware (GPUs)
 - ▶ Rendering-Paradigma: Rasterisierung statt Raytracing
 - ▶ die Folge sind spezielle Rendering-Techniken, z.B. für Schatten, viele Lichtquellen, ... und:



Vergleich GPU ↔ CPU

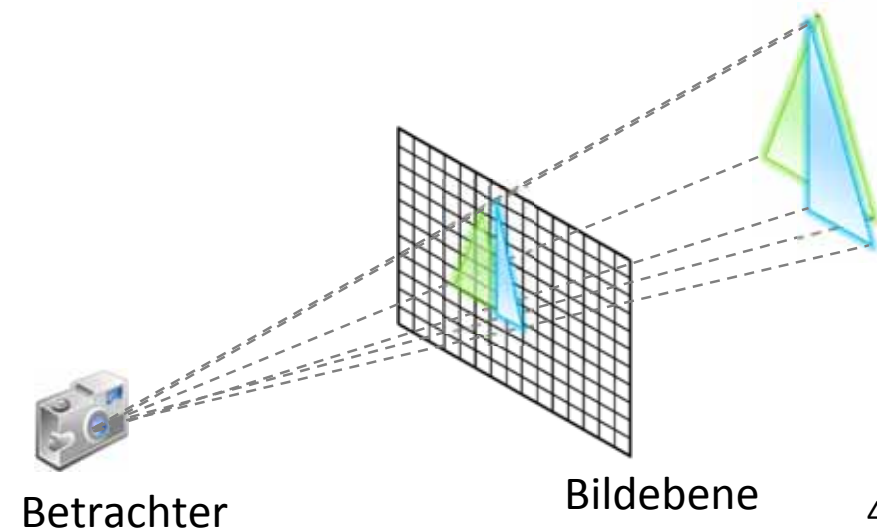
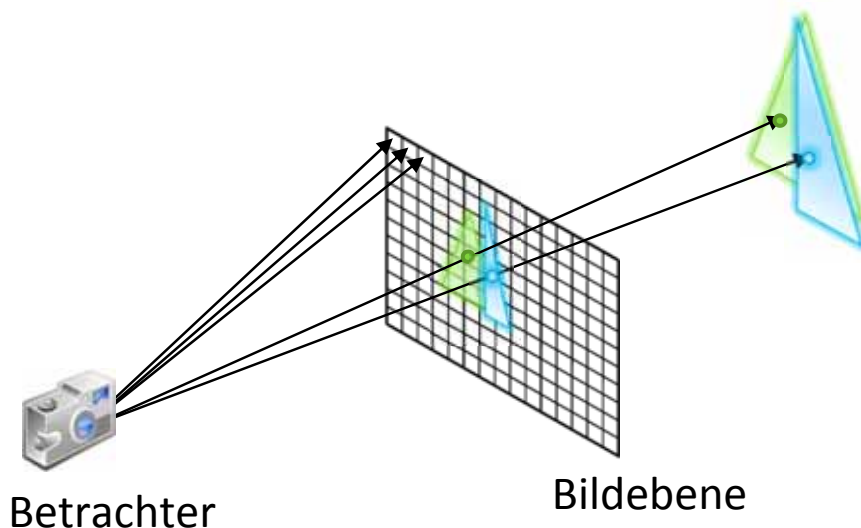


- ▶ Vergleich der Kennzahlen von CPUs und GPUs würde natürlich hinken
 - ▶ es handelt sich um völlig unterschiedliche Architekturen
 - ▶ dementsprechend unterscheiden sich die Programmiermodelle
 - ▶ es gibt aber vereinheitlichte APIs, z.B. OpenCL (siehe GPU Computing Praktikum)
- ▶ die Grenzen werden aber etwas „fließender“: GPUs sind inzwischen flexibel genug für viele Anwendungen in der Computergrafik, z.B.
 - ▶ Raytracing, Path Tracing, ..., inklusive dem Aufbau von BVH/kD-Bäume
 - ▶ Physiksimulationen (Flüssigkeit, Rigid/Soft-Body, ...)
 - ▶ Finite-Elemente-Methoden (Radiosity u.a.)
 - ▶ Molekulardynamik, Finanzmarktsimulationen, ...
 - ▶ spezielle APIs und Bibliotheken: OpenCL, CUDA, Thrust, ...
 - ▶ siehe www.gpgpu.org

Ray Tracing vs. Rasterisierung



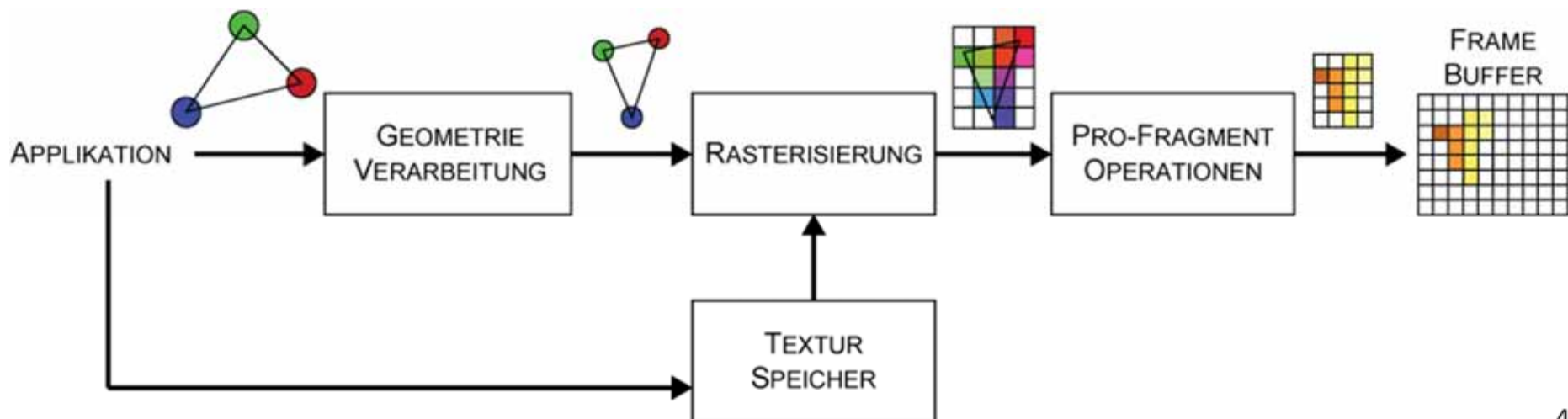
- ▶ interaktive CG basiert meist auf Rasterisierung
 - ▶ einfacher (und partiell nach und nach) in Hardware umzusetzen
 - ▶ für „Primärstrahlen“ ist die Performance bei Rasterisierung i.A. höher
 - ▶ **Raytracing:** Aufwand $O(\log n)$, benötigt aber zwingend eine räumliche Datenstruktur zur Beschleunigung
 - ▶ **Rasterisierung:** Aufwand $O(n)$, aber nur für sehr große n tatsächlich langsamer; außerdem ebenfalls Beschleunigungsstrukturen möglich
 - ▶ zunächst wichtig: geometrisches Detail und (lokale) Beleuchtung
 - ▶ Beleuchtungseffekte (Schatten, Spiegelungen, einfach indirektes Licht) können gut mit Rasterisierungsverfahren approximiert werden



Grafik-/Rasterisierungspipeline



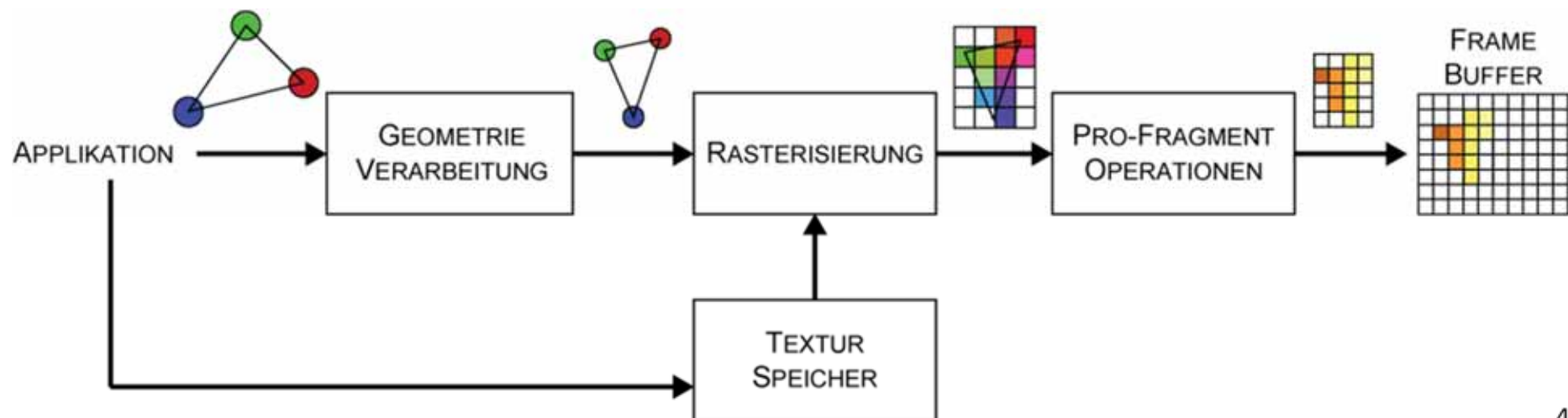
- ▶ **Dreiecke/Primitive** werden in einer **Pipeline** verarbeitet
 - ▶ „eins nach dem anderen“ (faktisch massiv-parallel), immer unabhängig voneinander
- ▶ **Geometrieverarbeitung** (u.a. mit Vertex/Geometry Shader)
 - ▶ Transformation der Eckpunkte (von Objekt- bis in Kamerakoordinaten)
 - ▶ evtl. Beleuchtung, Projektion auf die Bildebene
- ▶ **Rasterisierung**
 - ▶ Bestimmen der Pixel, die ein Dreieck bedeckt
 - ▶ Interpolation von Attributen, Texturkoordinaten, ...
 - ▶ Ausführen eines Fragment Shader
- ▶ **Fragment Operationen** (u.a. Verdeckungsrechnung)



Rasterisierungspipeline



- ▶ Rasterisierung (in Verbindung mit Tiefenpuffer) ist effizient:
Dreiecke durchlaufen – nacheinander, aber unabhängig – alle dieselben Verarbeitungsschritte
- ▶ Konsequenzen der Pipeline-Architektur
 - ▶ hoher Durchsatz, einfache Umsetzung (v.a. auch in Hardware)
 - ▶ **lokale Beleuchtungsmodelle**
 - ▶ globale Beleuchtungseffekte (z.B. Schatten) nur über mehrere Rendering-Durchgänge
 - ▶ Grafik-Hardware über APIs wie OpenGL, Direct3D, ...

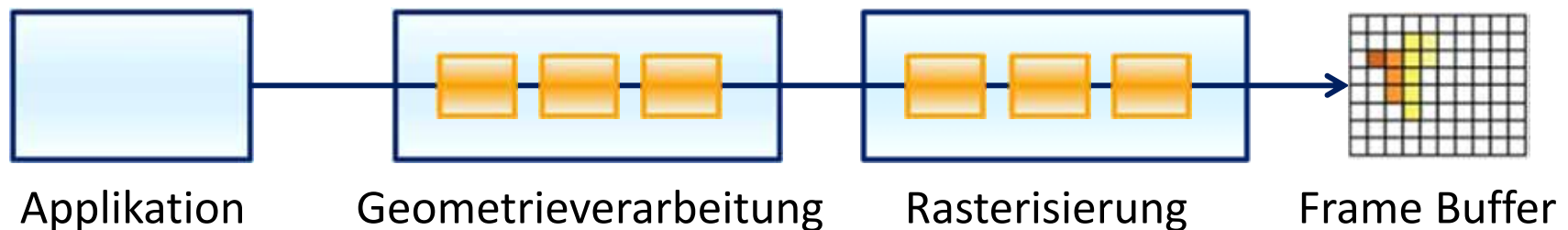


Grafik Pipeline, etwas abstrakter...



Laufzeit vs. Durchsatz

- ▶ Laufzeit/Verarbeitungszeit eines Primitives ist die Summe aller Verarbeitungsschritte/-stufen
- ▶ der Durchsatz („wie viele Primitive pro Zeit“) ist begrenzt durch die langsamste Stufe

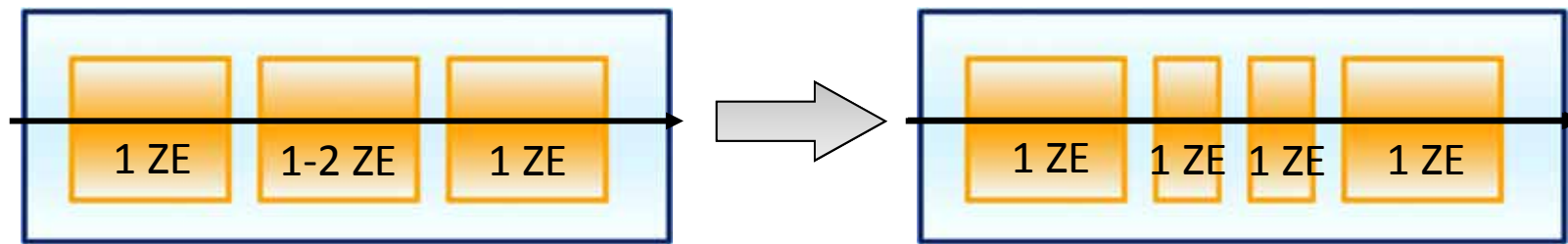


Grafik Pipeline, Optimierung

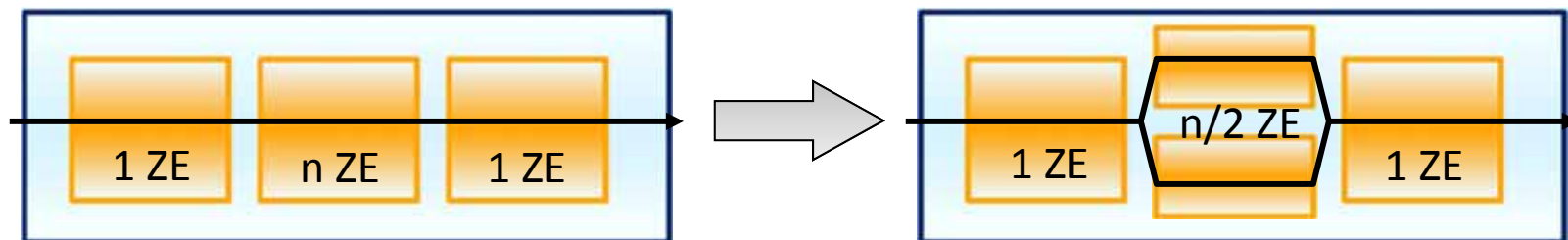


Verbesserung der Performance

- ▶ ist eine Verarbeitungsstufe der Flaschenhals, dann kann diese in zwei (oder mehr) aufeinander folgende Verarbeitungsschritte zerlegt werden
 - ▶ höherer Durchsatz (dafür u.U. längere Laufzeit)
 - ▶ klassisches Beispiel: Clipping



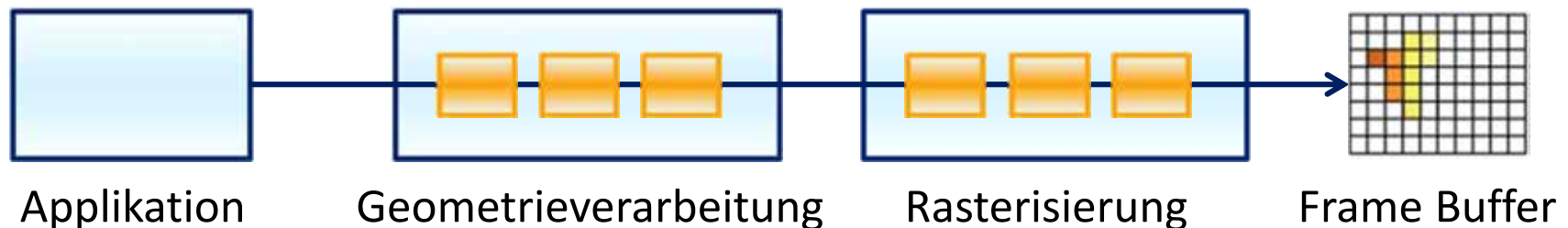
- ▶ parallele Verarbeitungseinheiten (höherer Durchsatz)
 - ▶ Vertex / Geometry / Fragment Shader → unabhängige Verarbeitung



Grafik Pipeline, etwas abstrakter...



- ▶ Anhalten/Entleeren der Pipeline (sog. „stalling“) ist teuer
 - ▶ auf Durchsatz optimierte Pipelines – z.B. moderne Grafik-HW – haben i.d.R. lange Laufzeiten
 - ▶ Entleeren der Pipeline ist aber häufig notwendig, z.B. wenn
 - ▶ Rendering-Resultat eines Durchgangs weiterverarbeitet werden soll
 - ▶ OpenGL Zustände, z.B. Texturen/Material, Lichtquellen, etc. geändert werden (deshalb sortiert man Objekte nach Material, Shader, ...)
 - ▶ heute: Flexibilität durch moderne HW-Features (z.B. Texture-Arrays)
 - ▶ Flaschenhals oft die API-/Draw-Calls (siehe Mantle, DirectX12)



Die nächsten Schritte



- ▶ InCG verwendet fast immer Grafik-Hardware und meist Rasterisierung
 - ▶ wir orientieren uns an modernem OpenGL
 - ▶ die Techniken sind aber API-invariant, auch oft hybrid CPU-GPU, ...
- ▶ Überblick über dieses Kapitels
 - ▶ eine kurze Wiederholung zu OpenGL und Shader
 - ▶ Rendering-Techniken für detaillierte Oberflächen:
Normal und Displacement Mapping

