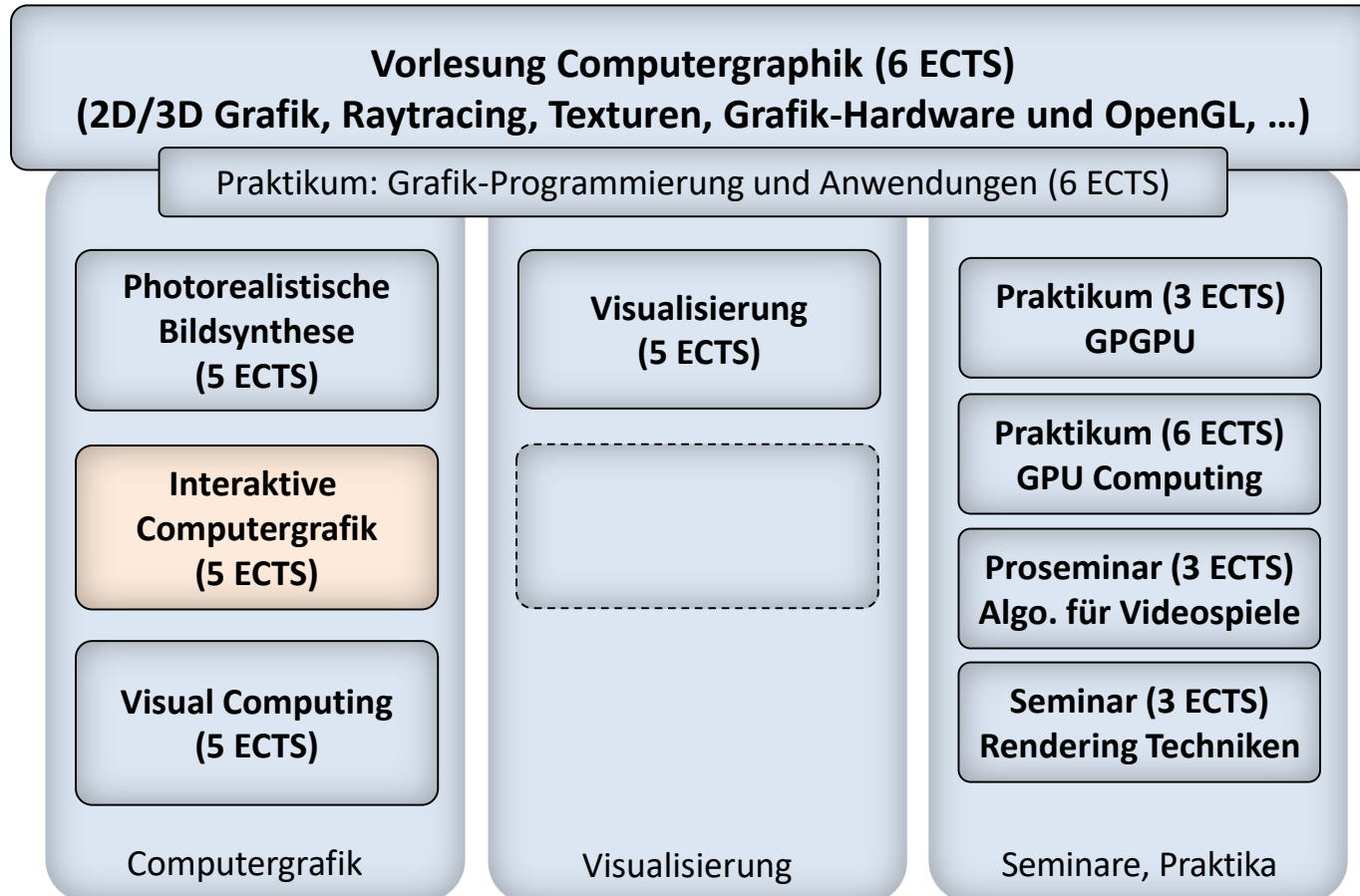


Interaktive Computergrafik

Vorlesung im Sommersemester 2017
Einführung

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

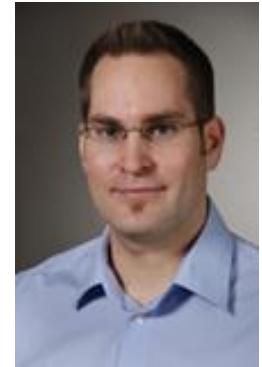




Organisatorisches



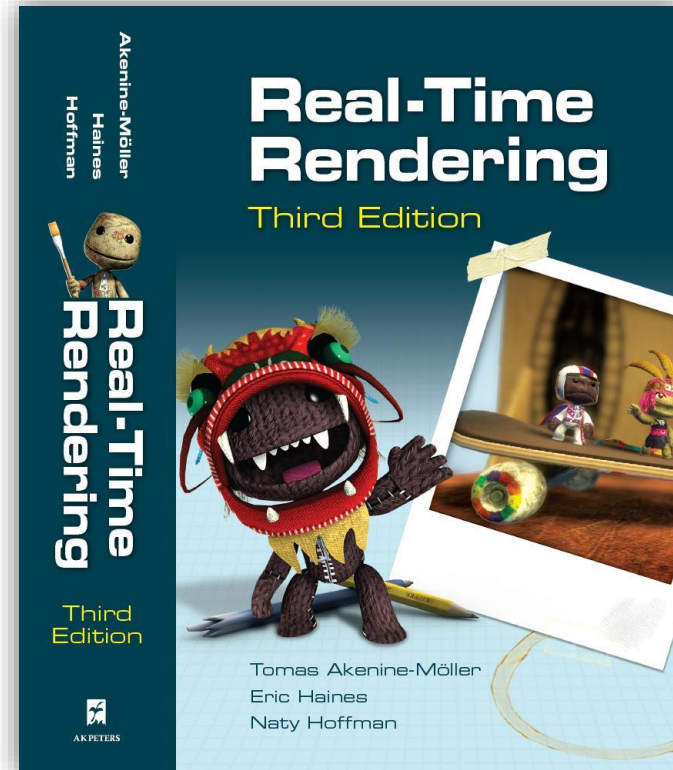
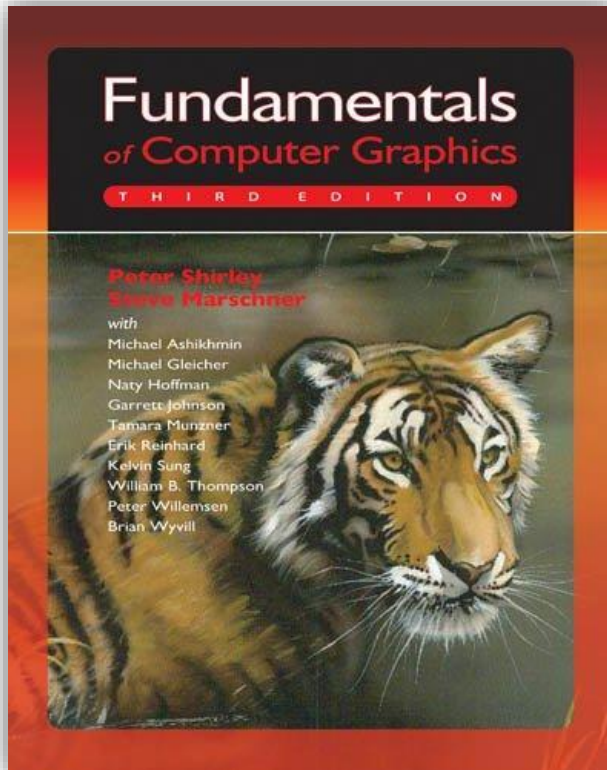
- ▶ Vorlesung (2 SWS): Carsten Dachsbacher
 - ▶ Mittwoch 11:30 – 13:00 HS -101
 - ▶ Ankündigungen auf der Web-Seite beachten



- ▶ Übungen (2 SWS): Christoph Schied
 - ▶ Mittwoch 9:45 – 11:15 HS -101
 - ▶ mehrere Übungsblätter / Programmieraufgaben
 - ▶ Besprechung in der Tafelübung: **10:30 – 11:15**
 - ▶ **erste Tafelübung: Mittwoch 3. Mai**
 - ▶ Unterstützung beim Praxisteil: **9.45 – 10.30**
(die Grafikrechner im ATIS Pool sind reserviert)



- ▶ 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)
- ▶ diverse aktuelle Publikationen und Kurse



- ▶ Vorlesungs- und Übungsmaterial: <https://ilias.studium.kit.edu>
- ▶ Webseite des Lehrstuhls <http://cg.ivd.kit.edu/> → Lehrveranstaltungen
- ▶ aktuelle Infos, Termine



The screenshot shows the website of the Chair of Computer Graphics at the Karlsruhe Institute of Technology (KIT). The header includes the KIT logo and navigation links: HOME | ENGLISH | IMPRESSUM | SITEMAP | KIT. The main content area is titled 'Lehrstuhl für Computergrafik' and features a table of lectures and exercises. A sidebar on the left contains a search bar and a list of navigation links.

Lehrstuhl für Computergrafik

Vorlesungen

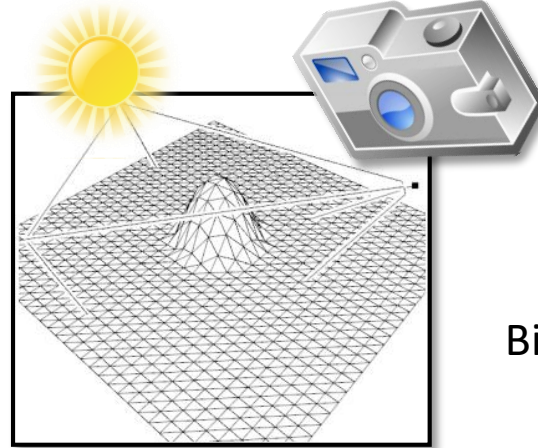
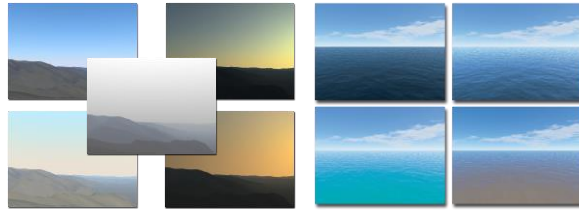
Titel	Zeit und Ort	Dozent
▶ Interaktive Computergrafik	26.04.2017 11:30 - 13:00 wöchentlich ▶ 50.34 ☑ Raum -101 50.34 INFORMATIK, Kollegiengebäude am Fasanengarten ▶ weitere...	▶ Prof.Dr.-Ing. Carsten Dachsbacher
▶ Übung Interaktive Computergrafik	03.05.2017 09:45 - 11:15 wöchentlich ▶ 50.34 ☑ Raum -101 50.34 INFORMATIK, Kollegiengebäude am Fasanengarten ▶ weitere...	▶ Christoph Schied
▶ Fotorealistische Bildsynthese	27.04.2017 15:45 - 17:15 wöchentlich ▶ 50.34 ☑ Raum -101 50.34 INFORMATIK, Kollegiengebäude am Fasanengarten ▶ weitere...	▶ Dr. Johannes Schudeiske
▶ Übung Fotorealistische Bildsynthese	04.05.2017 11:30 - 13:00 wöchentlich ▶ 50.34 ☑ Raum 131 50.34 INFORMATIK, Kollegiengebäude am Fasanengarten ▶ weitere...	▶ Tobias Zirr

Unser Profil

- ▶ Publikationen
- Mitarbeiter
- Forschung
- ▶ **Lehrveranstaltungen**
 - SS 2017**
 - ▶ WS 2016/17
 - ▶ SS 2016
 - ▶ WS 2015/16
 - ▶ SS 2015
 - ▶ WS 2014/15
 - ▶ SS 2014
 - ▶ WS 2013/14
 - ▶ SS 2013
 - ▶ WS 2012/13
 - ▶ SS 2012
 - ▶ WS 2011/12
 - ▶ SS 2011
 - ▶ WS 2010/11
 - Dokumentenvorlagen
 - PSF: Punzel

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

Computergrafik



➔
Bildsynthese

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

Realistisches Bild?
Beleuchtung,
Material, ...?



Betrachter der
Szene auf dem Monitor

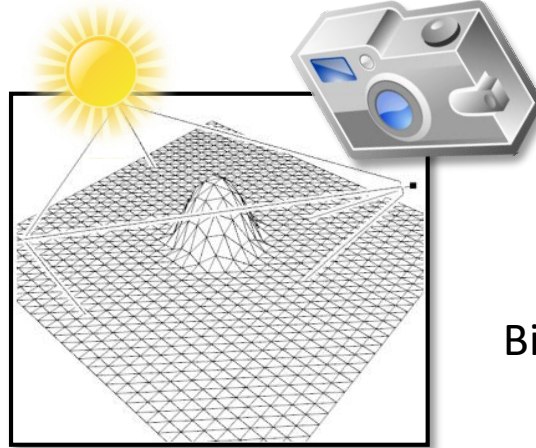
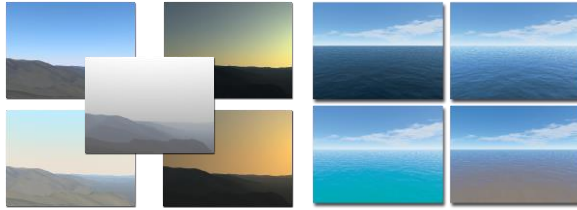


Monitor

←
Ansteuerung des
Monitors



Speicherung,
Verarbeitung,
Bildmanipulation,
Ausgabe



➔
Bildsynthese

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

... ergänzt durch reale Daten
(Texturen, Environment Maps, 3D Scanner, ...)

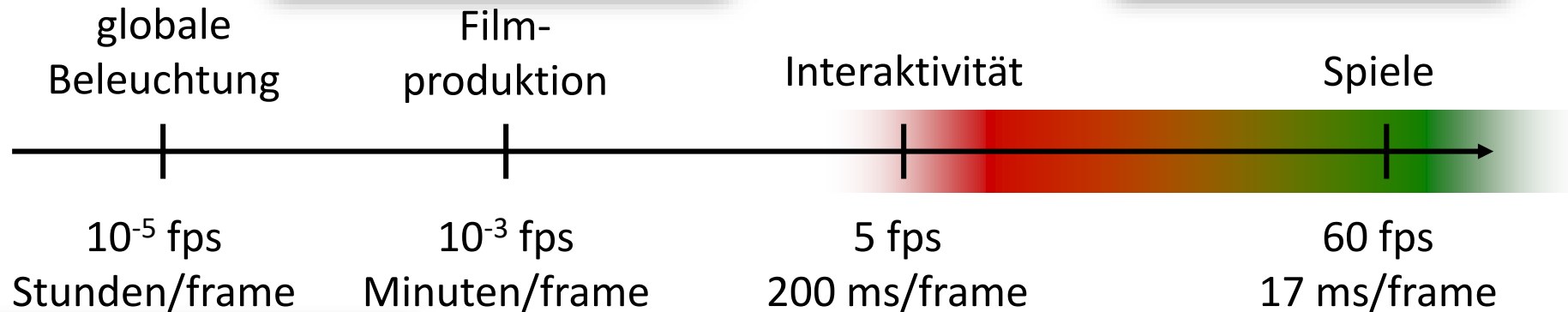


Speicherung,
Verarbeitung,
Bildmanipulation,
Ausgabe

Interaktive Computergrafik, „Randbedingung“

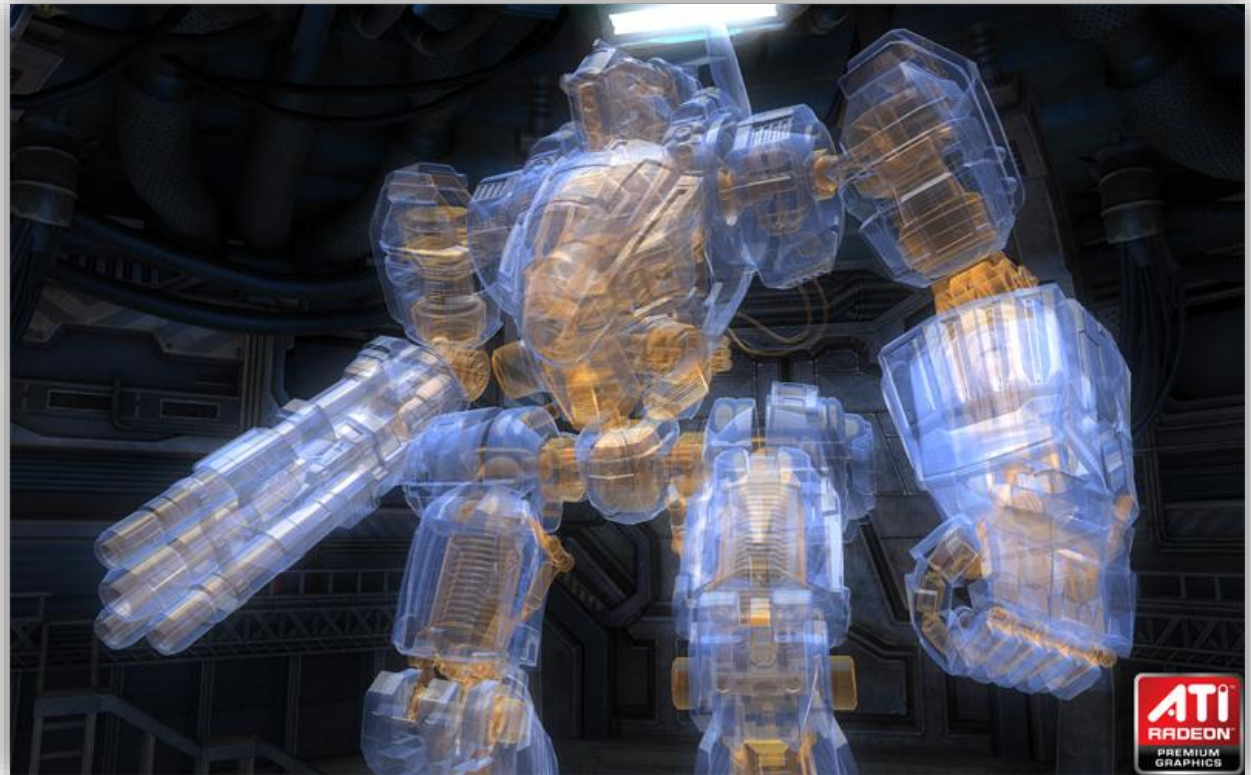
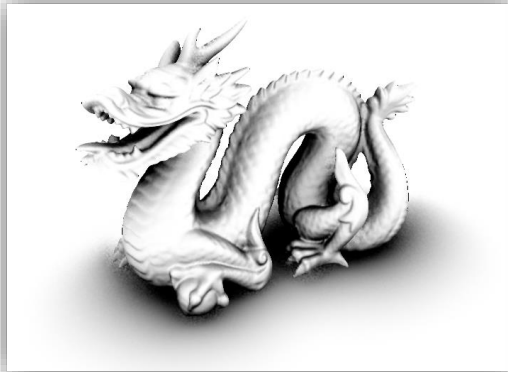


- ▶ wir verfolgen ähnliche Ziele wie bisher, aber mit begrenzten Ressourcen (Zeit, Hardware, etc.) und höheren Ansprüchen!



„Randbedingung“

- ▶ beschränkten (Zeit-)Ressourcen →
 - ▶ verwende schnelle/dedizierte Grafik-Hardware (GPUs)
 - ▶ Rendering-Paradigma: meist Rasterisierung statt Raytracing
 - ▶ die Folge sind spezielle Rendering-Techniken, z.B. für Schatten, viele Lichtquellen, ...

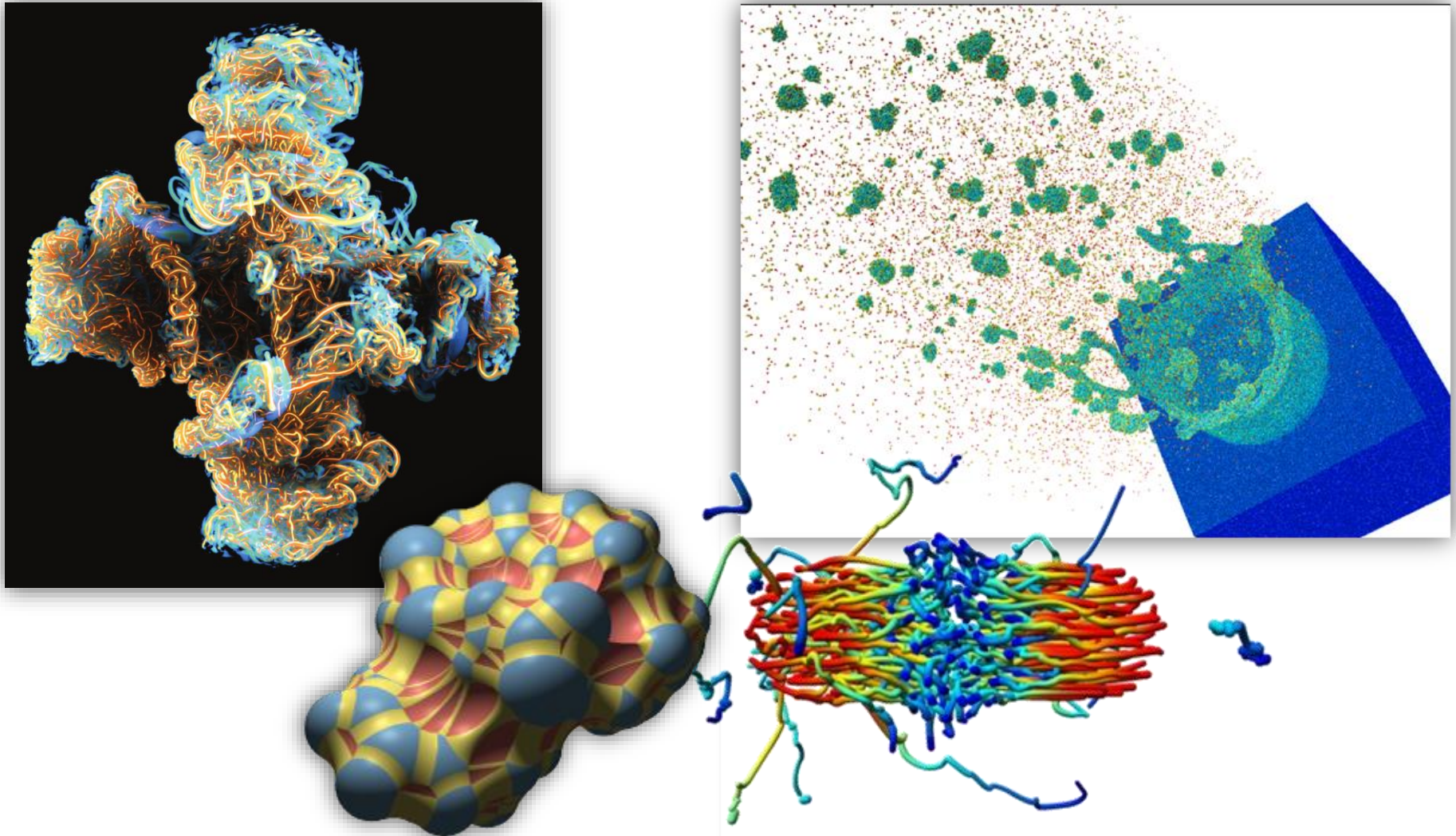


- ▶ interaktiv: $<200\text{ms}$ pro Bild (= 5fps), $33\text{ms} = 30\text{fps}$
- ▶ Anwendungen: Videospiele (Bild: Eve Valkyrie)...



Interaktive Computergrafik

- ▶ interaktiv: $<200\text{ms}$ pro Bild (= 5fps), $33\text{ms} = 30\text{fps}$
- ▶ Anwendungen: Visualisierung...



Interaktive Computergrafik

- ▶ interaktiv: $<200\text{ms}$ pro Bild (= 5fps), $33\text{ms} = 30\text{fps}$
- ▶ Anwendungen: Produktpräsentation/-design (hier mit VR) ...



► Anwendungen: Flug- und Fahrsimulatoren



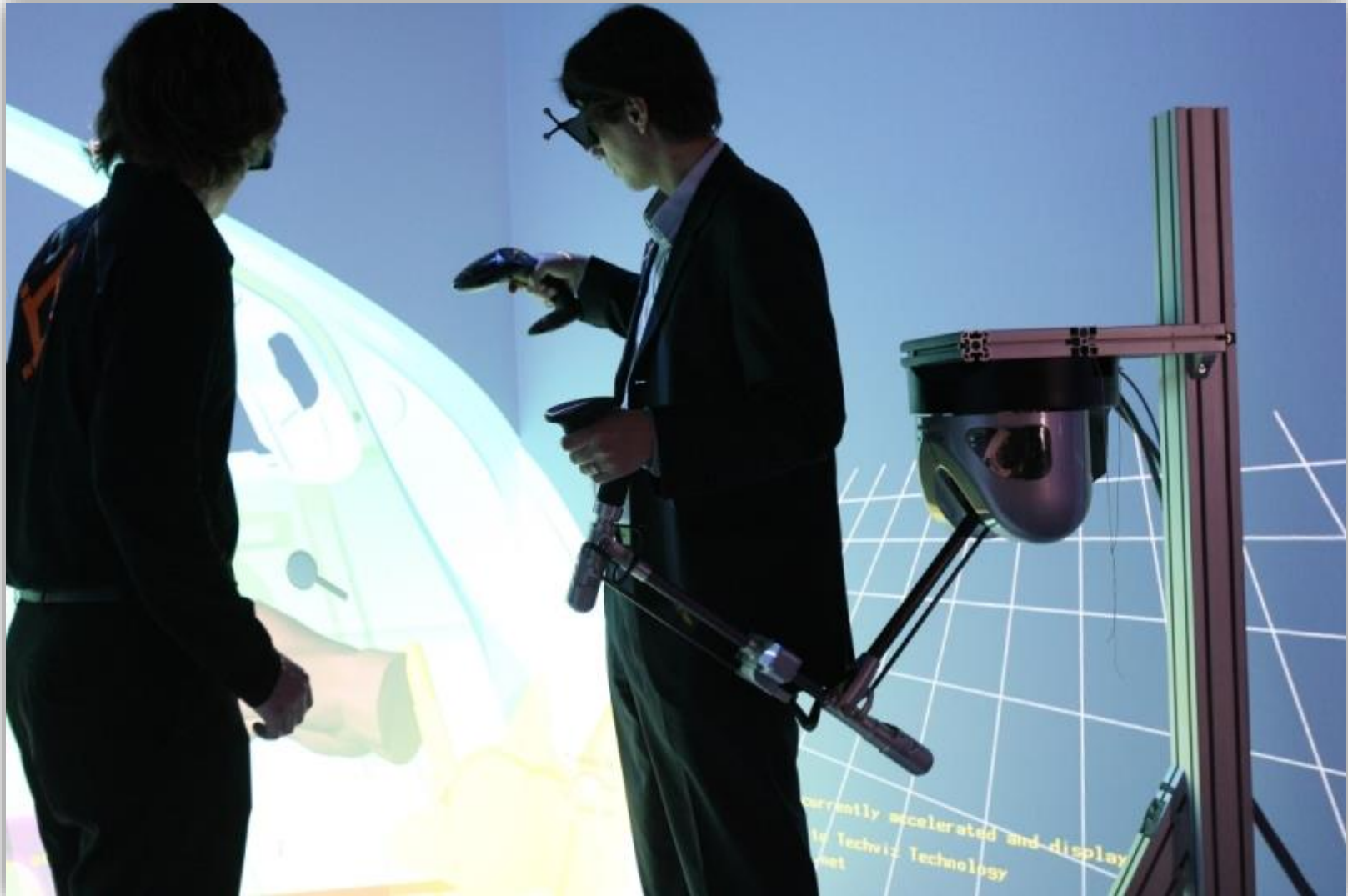
- ▶ Anwendungen: Hardware in the Loop



- Anwendungen: Rendering und Virtual Reality in Ingenieurbereich



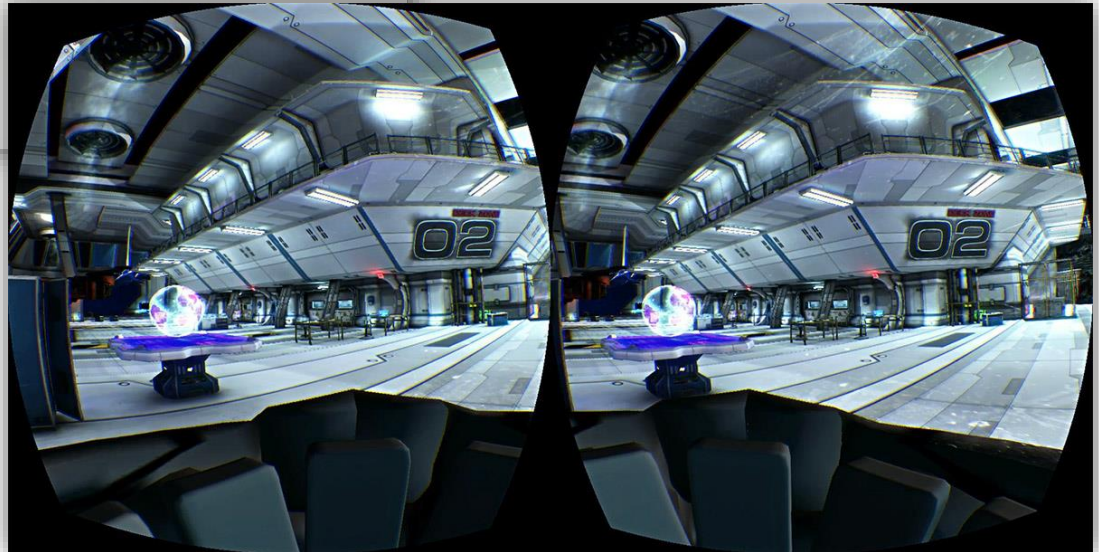
- Anwendungen: Virtual Reality in Ingenieursbereich (Cave)



► Anwendungen: Virtual Reality für Training



- ▶ Oculus Rift, HTC Vive, Microsoft HoloLens, Sony Playstation VR, ...
- ▶ ... kommt jetzt der Durchbruch für virtuelle und erweiterte Realität?



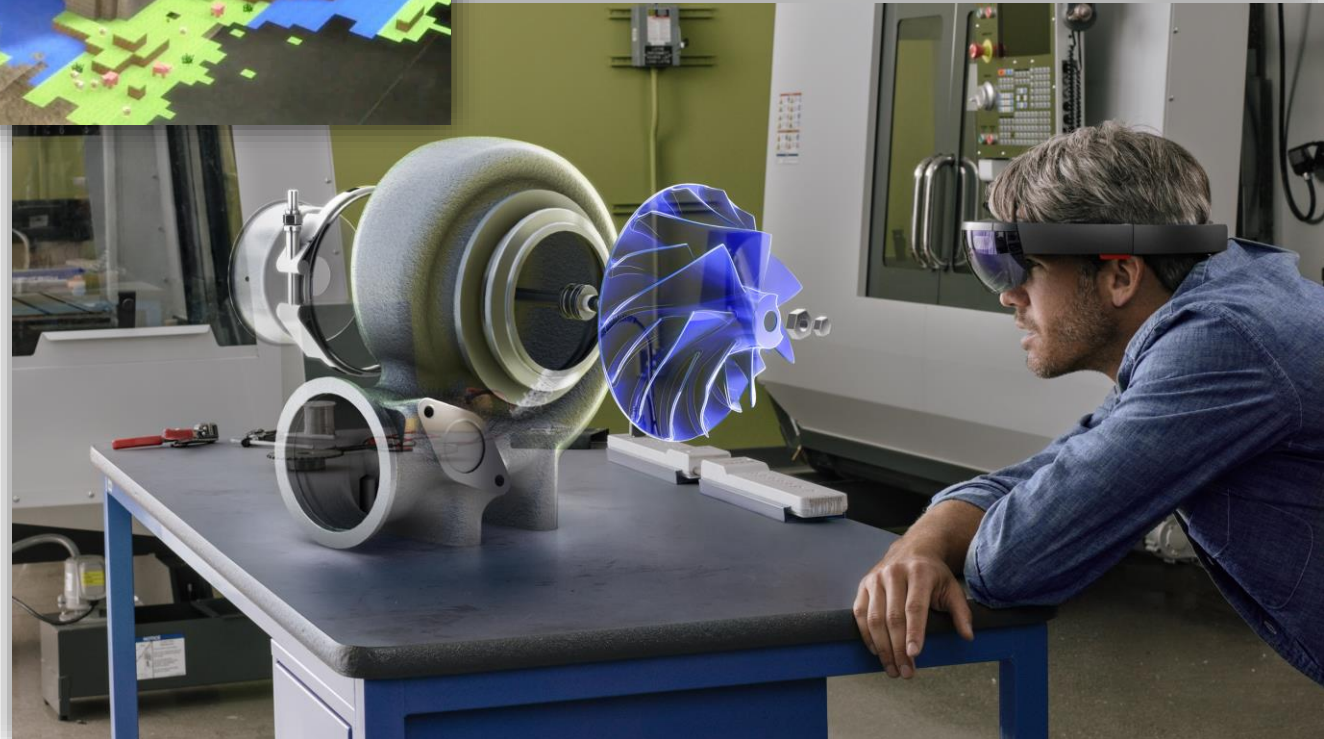


Bilder: Crytek Climb

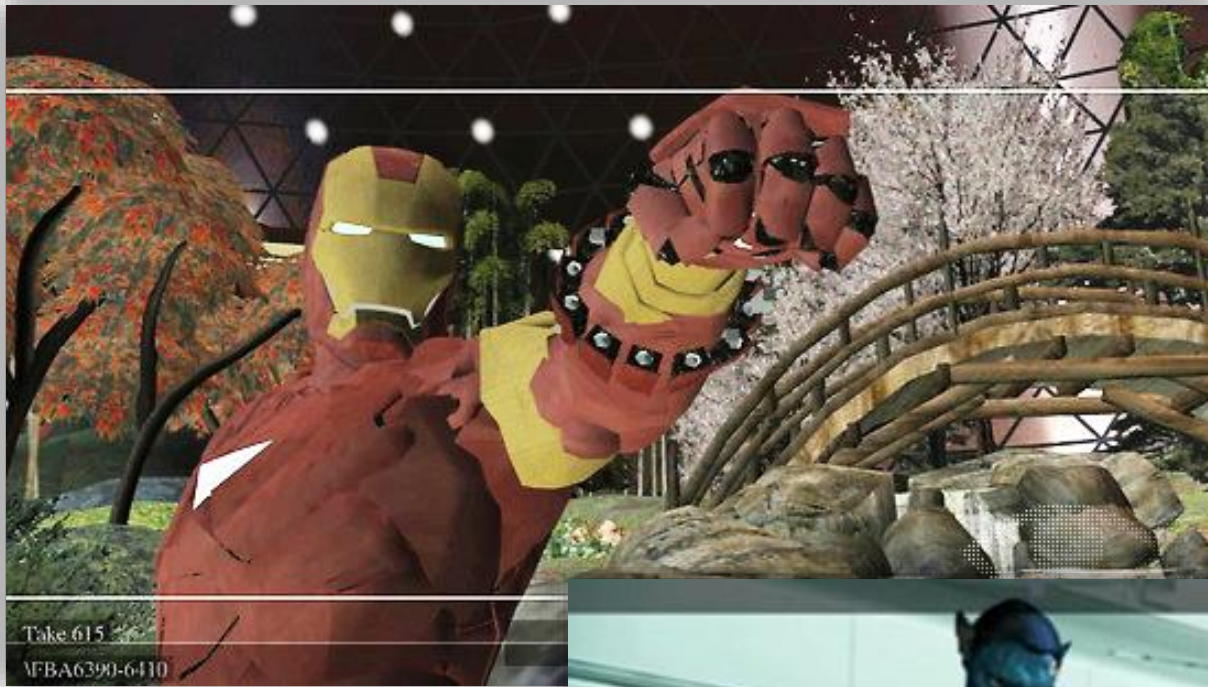


Interaktive Computergrafik

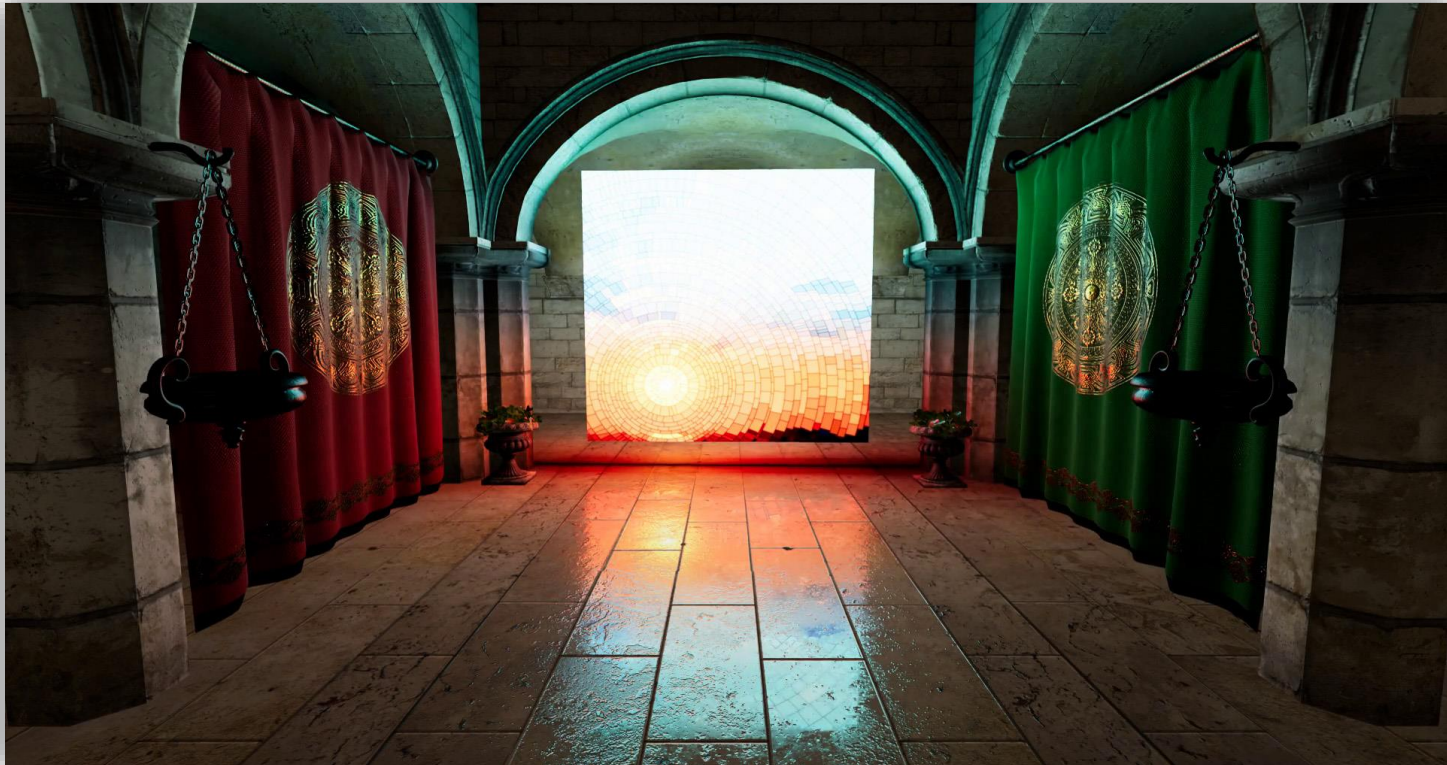
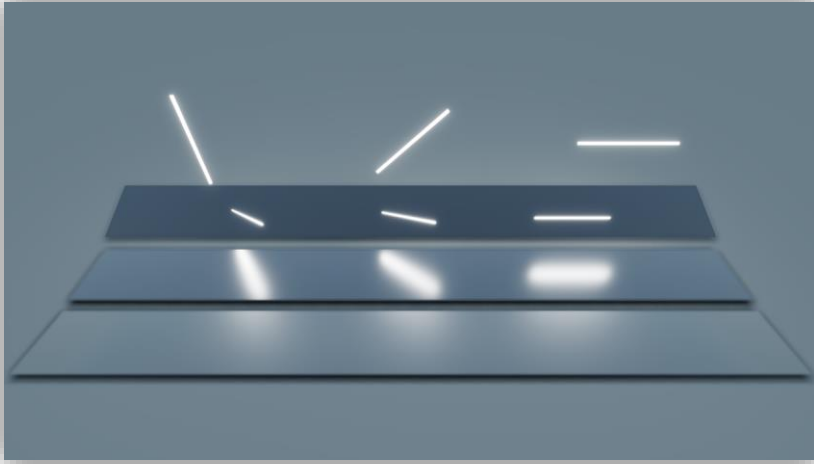
► Anwendungen: Augmented Reality



Previsualization/Lighting Design: Filmproduktionen



Aktuelle Forschungsthemen



Aktuelle Forschungsthemen



- ▶ Deferred Shading, Wasser, Temporal Antialiasing, Morphological AA
<http://advances.realtimerendering.com/s2016/index.html>



► Rendering von Textilien (allg. Details, Details, Details, ...)

<http://www.cs.utah.edu/~kwu/rtfr.html>

Modified Procedural

Yarn Model



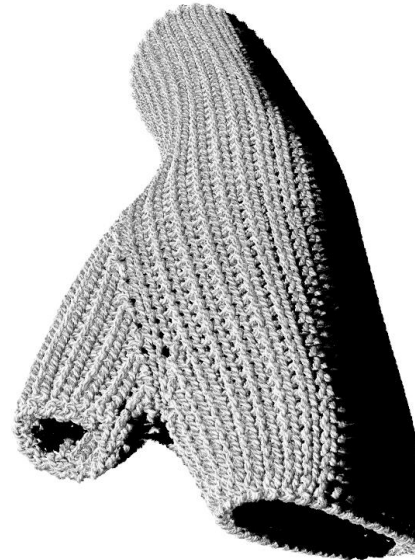
Core Fibers



LoD



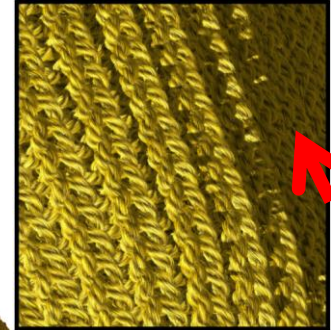
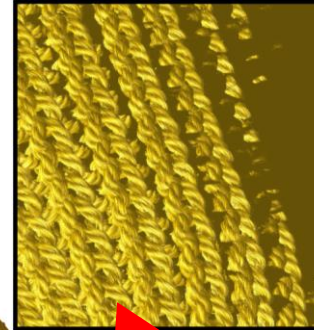
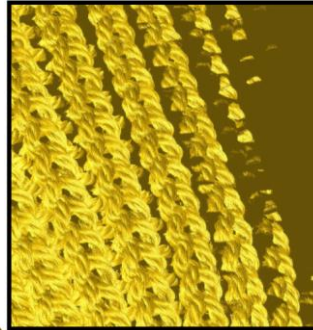
Self-shadows



Ambient Occlusion



- ▶ Rendering von Textilien (allg. Details, Details, Details, ...)
<http://www.cs.utah.edu/~kwu/rtfr.html>



No Shadow

+ Shadow Map

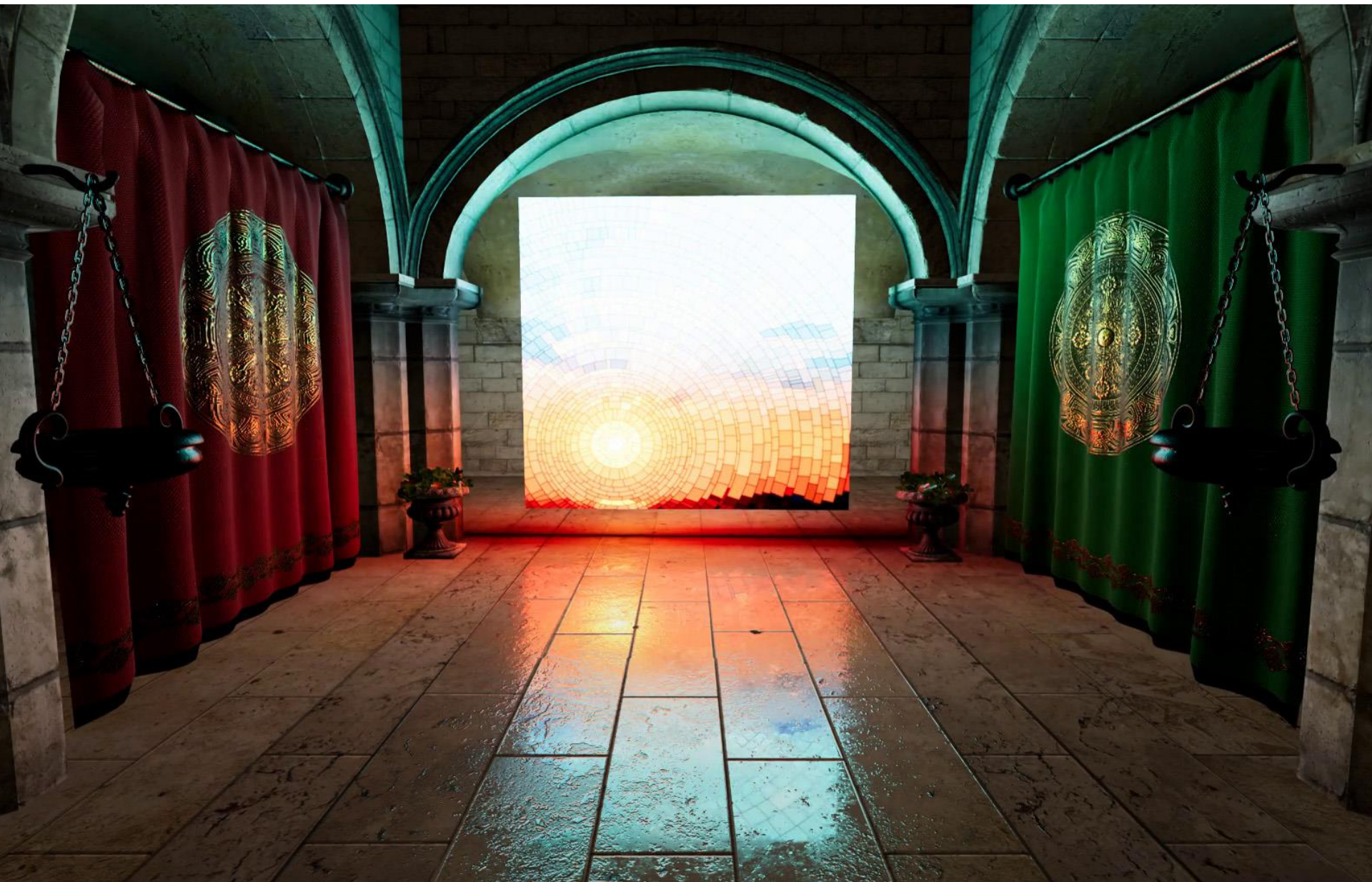
+ Self-shadows

+ Ambient Occlusion

Ziele und nächste Schritte...

- ▶ Ziel: performantes, (foto)realistisches Rendering in hoher Qualität
- ▶ wir lernen wichtige Konzepte für interaktive Grafik kennen, die wir an konkreten Verfahren festmachen, z.B. Vorfilterung, Level of Detail u.v.m.
- ▶ Verfahren für...
 - ▶ mehr Oberflächendetail → realistisches Aussehen
 - ▶ Schatten, Voxelisierung, ... → realistische Beleuchtung
 - ▶ effizientes Shading in komplexen Szenen → Effizienz
 - ▶ vorberechneter Lichttransport → Realismus
 - ▶ Culling und Level of Detail → Effizienz
 - ▶ ... Animation, Tone Mapping, aktuelle Forschungsthemen

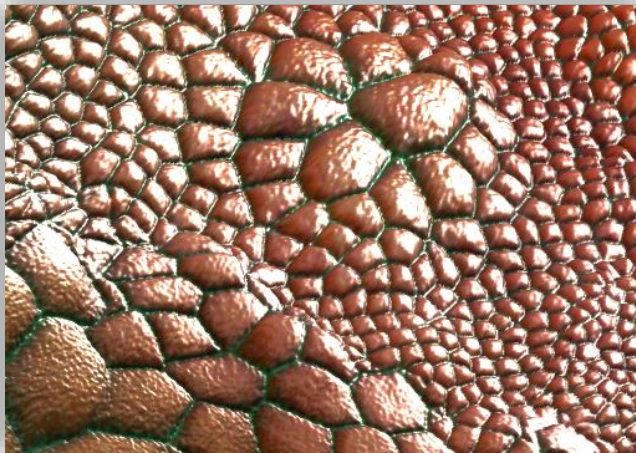
Oberflächendetails (Normal Mapping, Shader Recap)

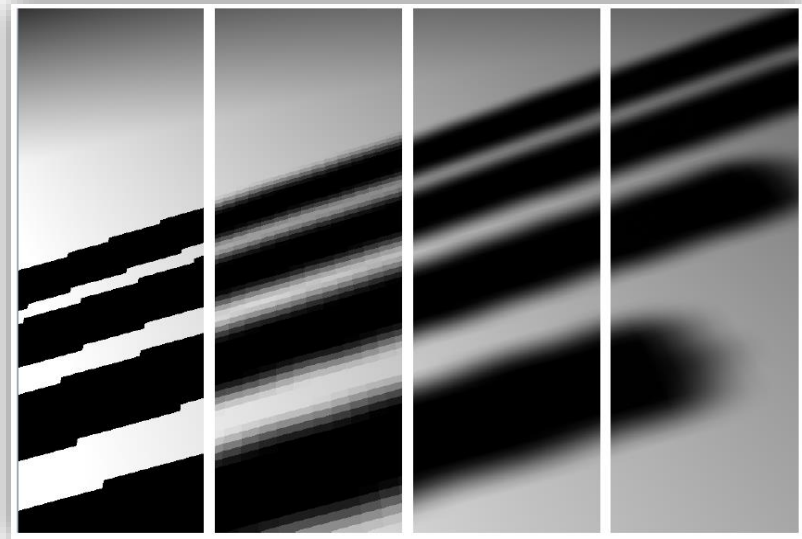
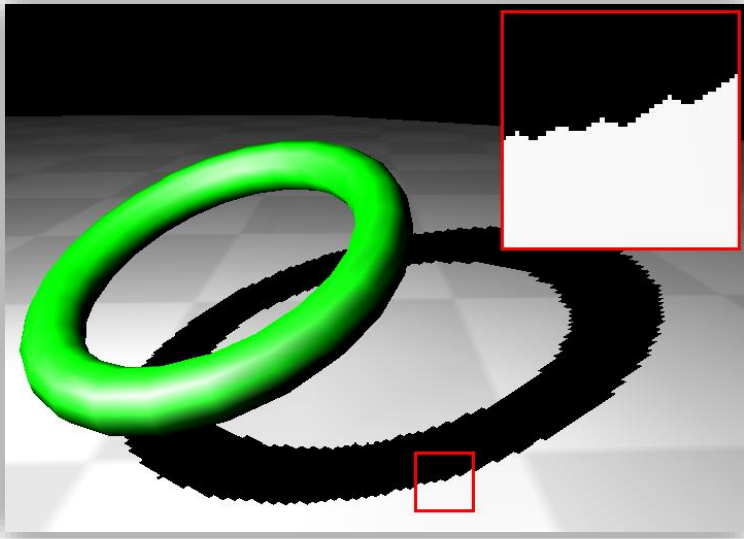
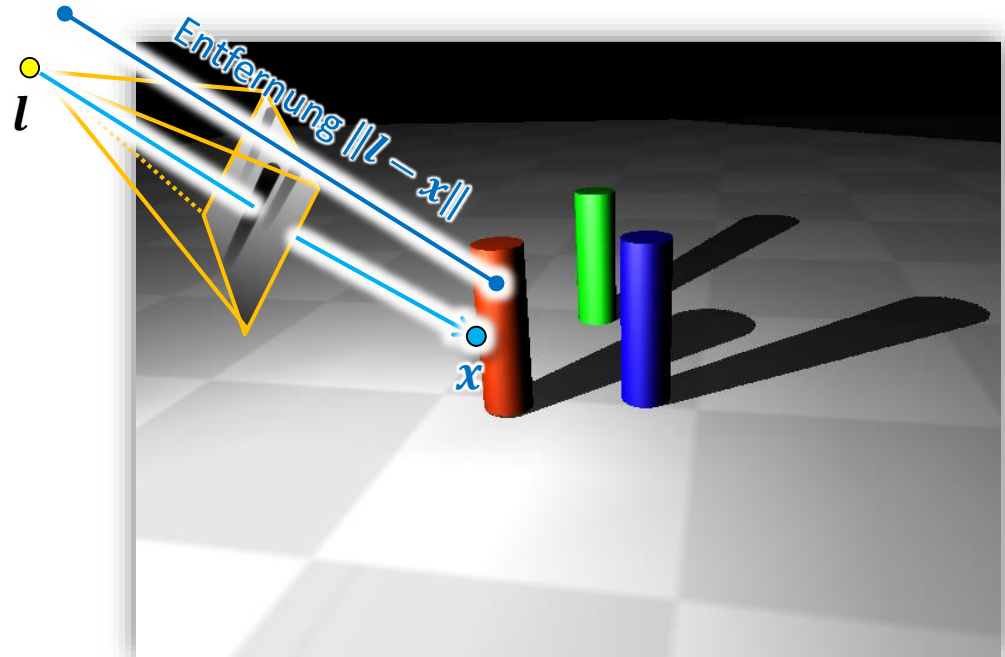
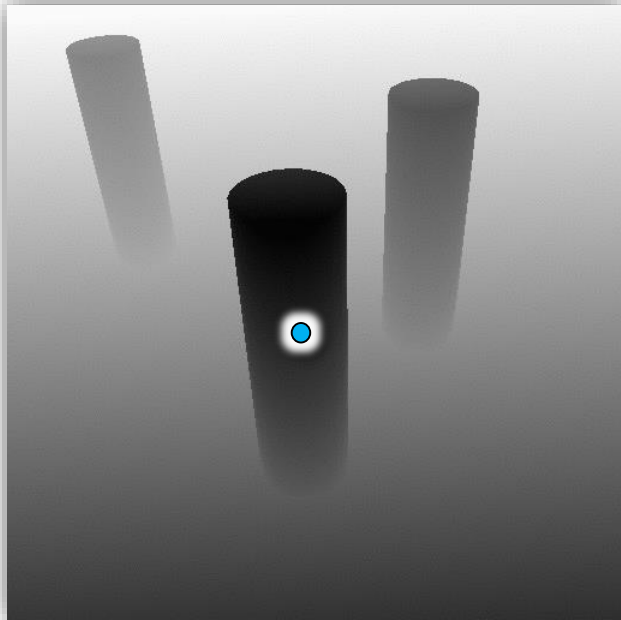




(Vor-)Filterung

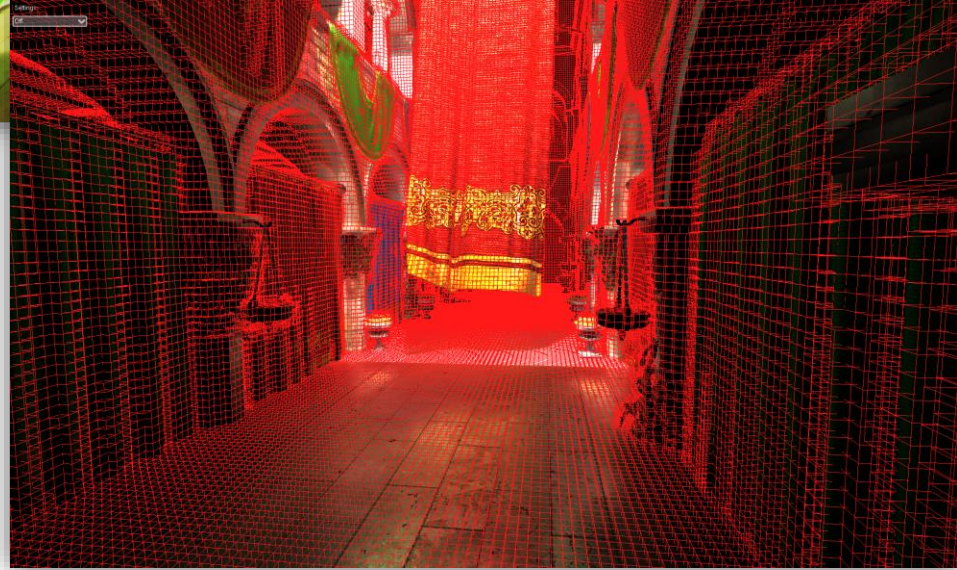
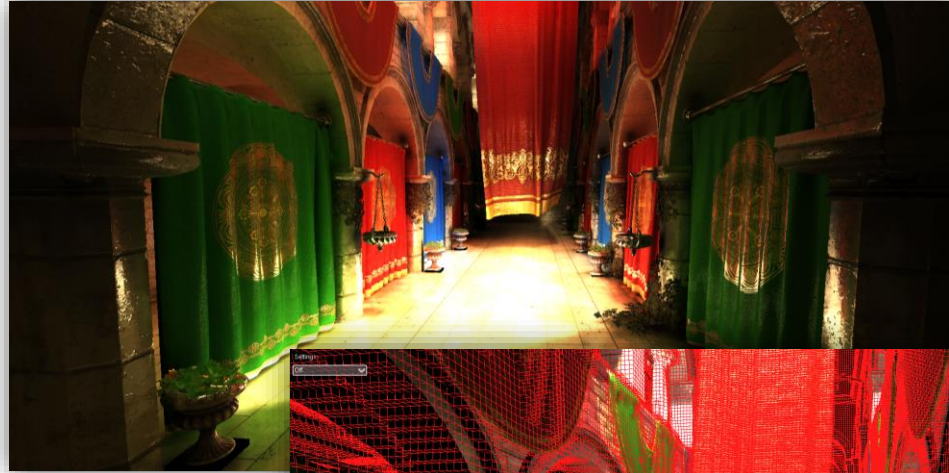
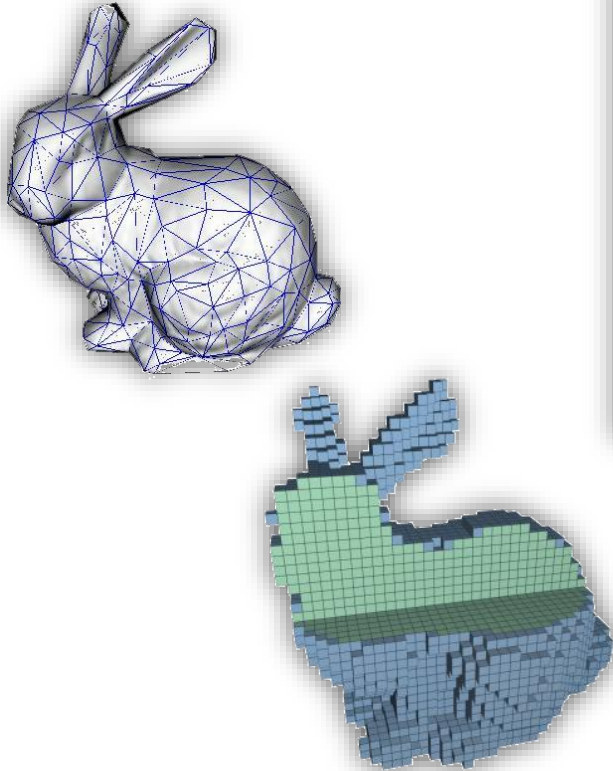
- ▶ Vorfilterung kennen wir von Texturen: Mip-Mapping
- ▶ aber: Vorfilterung Farben $\neq$ Vorfilterung Normalen, Voxel, Tiefe ...





Voxelisierung

- ▶ beschränkte Ressourcen → approximative Lösungen
- ▶ alternative Repräsentationen der Geometrie/Szene



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

Crassin et. al, Pacific Graphics 2011

Voxelisierung

- ▶ beschränkte Ressourcen → approximative Lösungen
- ▶ alternative Repräsentationen der Geometrie/Szene
 - ▶ z.B. für approximative indirekte Beleuchtung

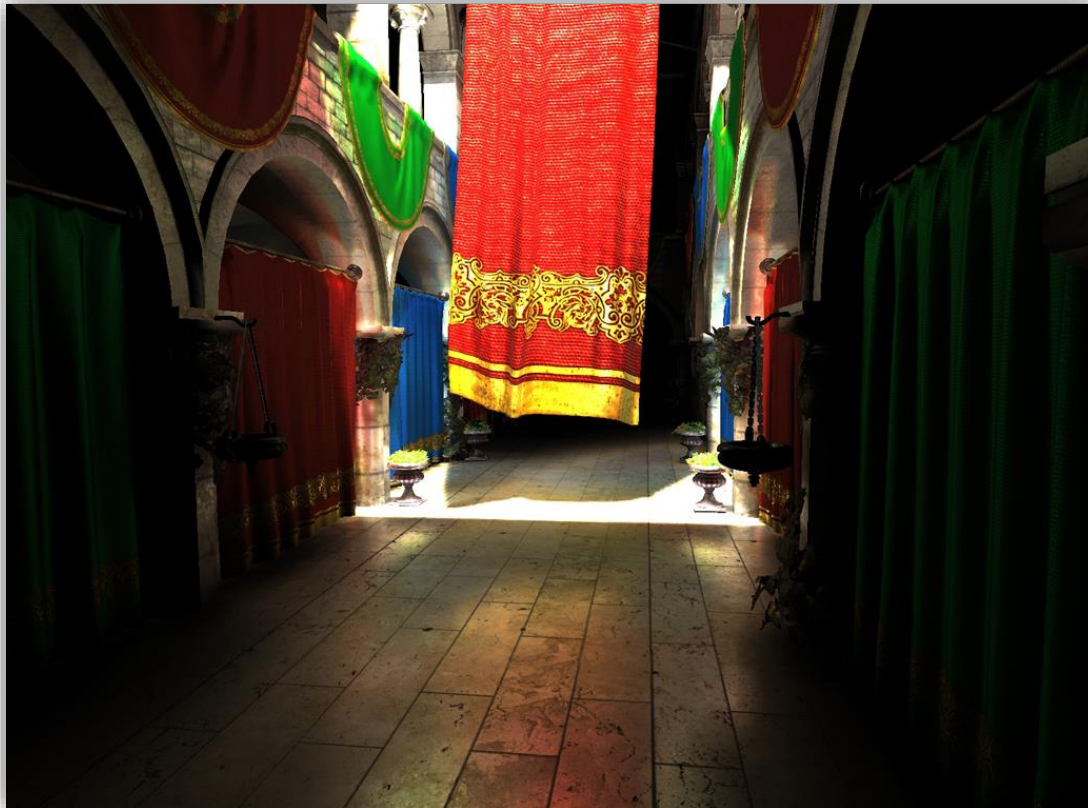
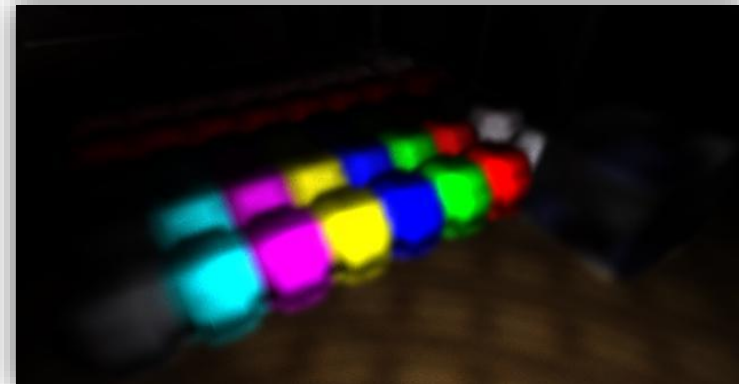


Bild: Cyril Crassin

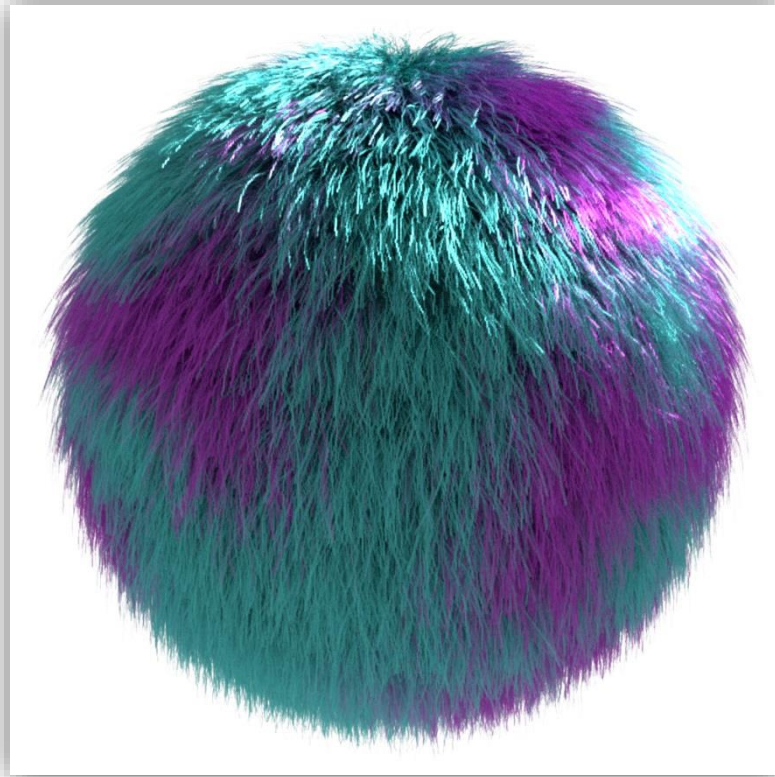


Bilder: The Technology Behind the
"Unreal Engine 4 Elemental demo", SIGGRAPH'12

(Vor-)Filterung

- ▶ Vorfilterung kennen wir von Texturen: Mip-Mapping
- ▶ aber: Vorfilterung Farben $\neq$ Vorfilterung Normalen, Voxel, Tiefe ...

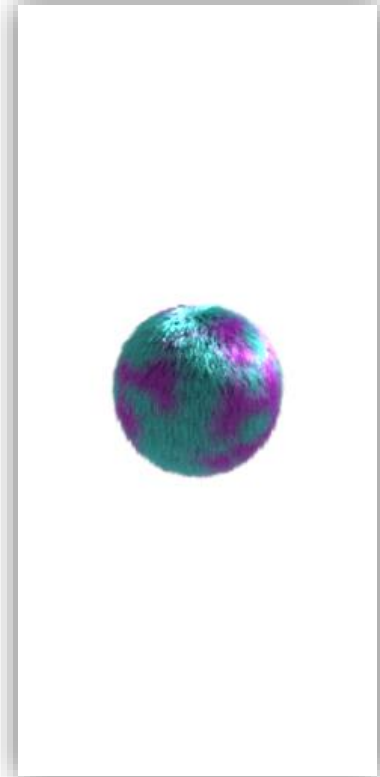
Nahansicht – 1911 MB (1024³ Voxel)



fern – 1911 MB



fern – 14 MB



Distanzfunktionen (Signed Distance Functions)



- ▶ beschränkte Ressourcen → approximative Lösungen
- ▶ alternative Repräsentationen der Geometrie/Szene
 - ▶ Raymarching/Sphere Tracing von Distanzfunktionen

SDF-Schatten →

Cascaded
Shadow Map →



Distanzfunktionen (Signed Distance Functions)

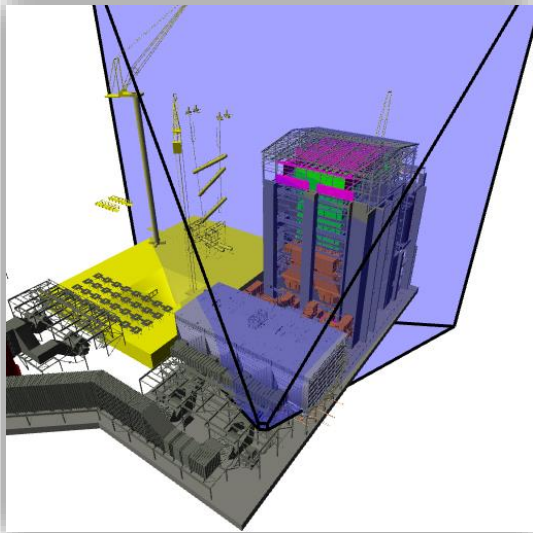


- ▶ beschränkte Ressourcen → approximative Lösungen
- ▶ alternative Repräsentationen der Geometrie/Szene
 - ▶ Raymarching/Sphere Tracing von Distanzfunktionen

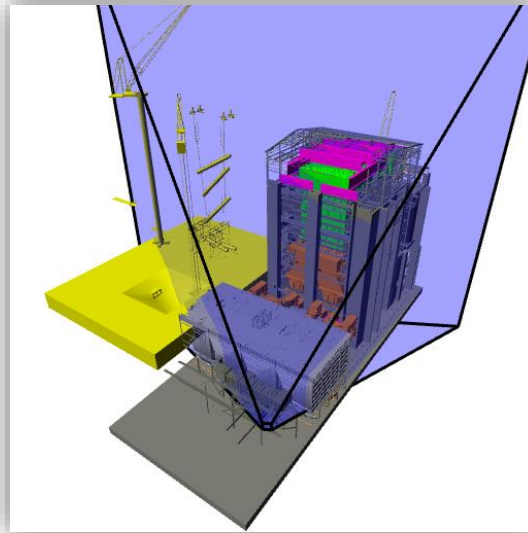


Occlusion Culling

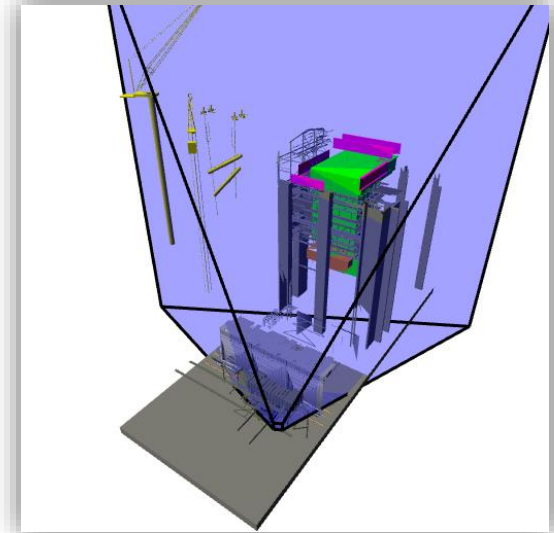
- ▶ beschränkte Ressourcen → keine unnötige Arbeit
- ▶ keine Geometrie, die nicht sichtbar ist
(Vorsicht: was heißt nicht sichtbar? nicht sichtbar in Spiegelungen, bei indirekter Beleuchtung, ...?)



gesamtes Modell, 1.7 Mio Δ



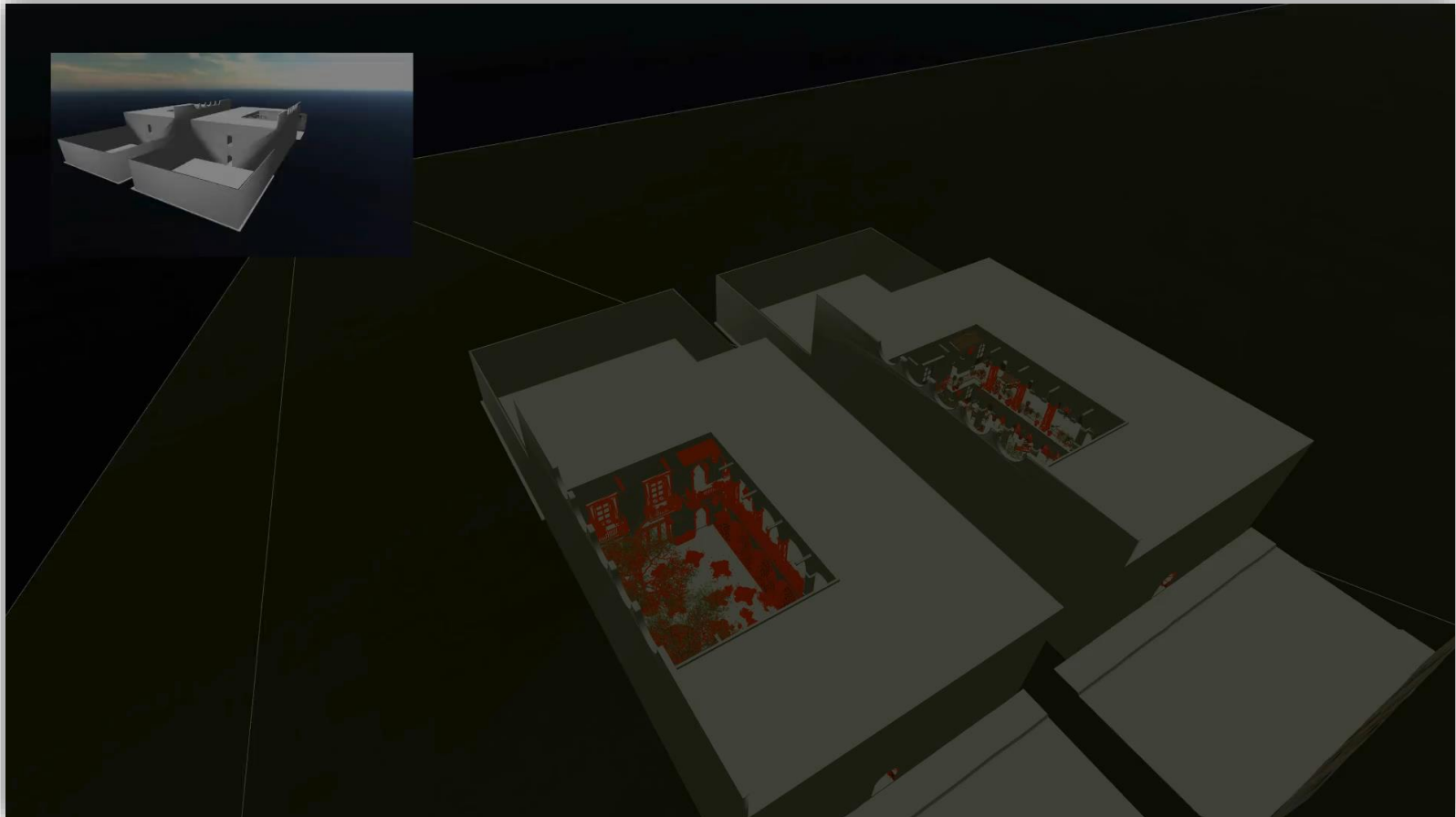
View Frustum Culling,
1.4 Mio Δ



Occlusion Culling,
89k Δ

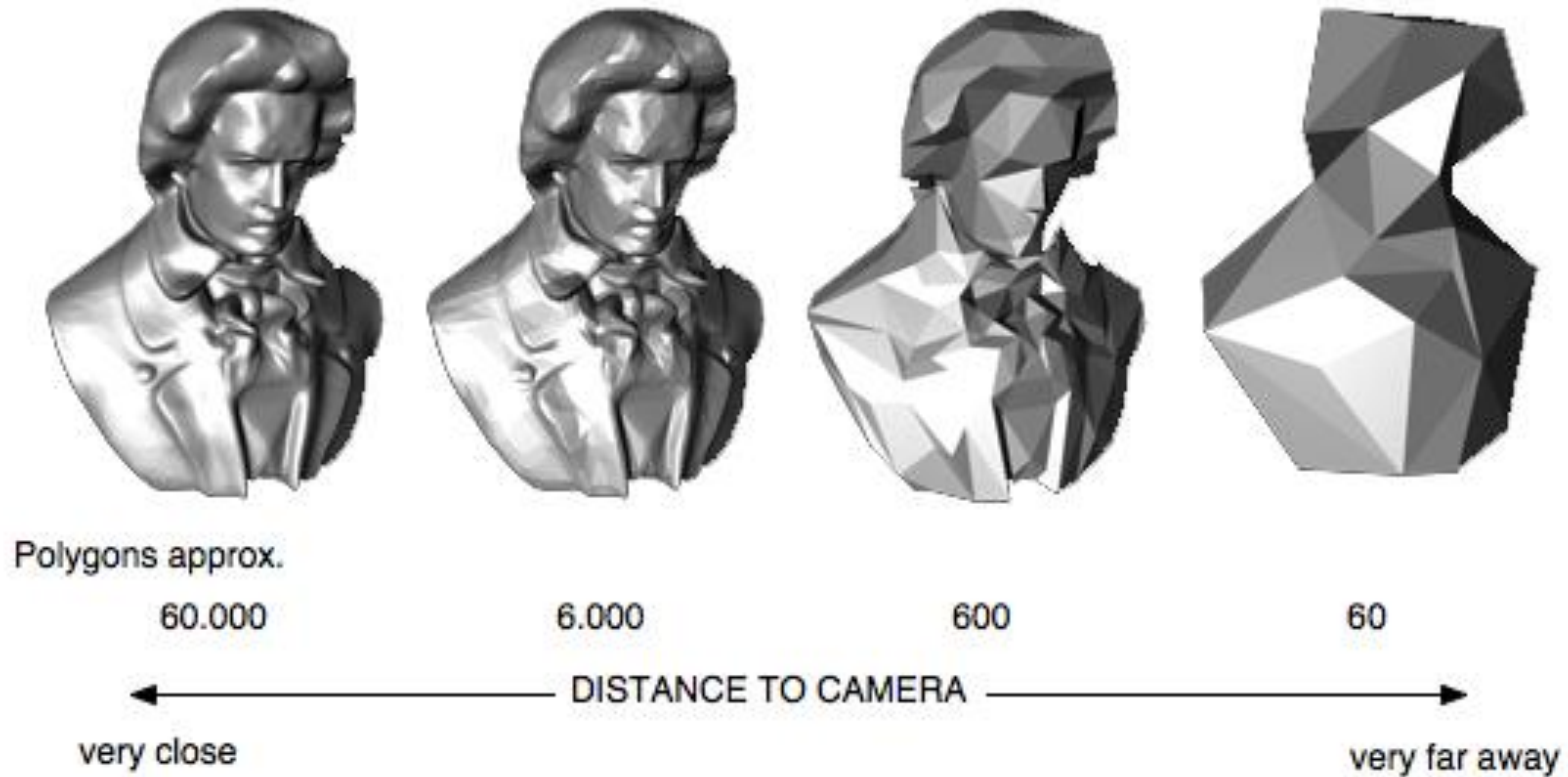
Frustum & Occlusion Culling

- ▶ beschränkte Ressourcen → keine unnötige Arbeit
- ▶ keine Geometrie, die nicht sichtbar ist
(Vorsicht: was heißt nicht sichtbar? nicht sichtbar in Spiegelungen, bei indirekter Beleuchtung, ...?)



Level of Detail-Verfahren

- ▶ beschränkte Ressourcen → keine unnötige Arbeit
- ▶ nicht mehr Detail als nötig oder erkennbar
 - ▶ z.B. reduzierte Dreiecksnetze oder weniger Oberflächendetails



Beispiel Tessellation – Unigine Benchmark

Eingabegeometrie vor der Unterteilung durch die Tessellation-Einheit



Beispiel Tessellation – Unigine Benchmark



Displacement Mapping: Unterteilung und Verschiebung der neuen Vertizes

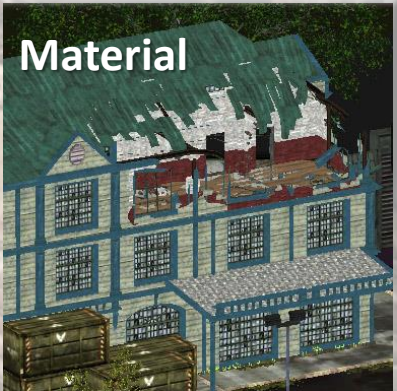
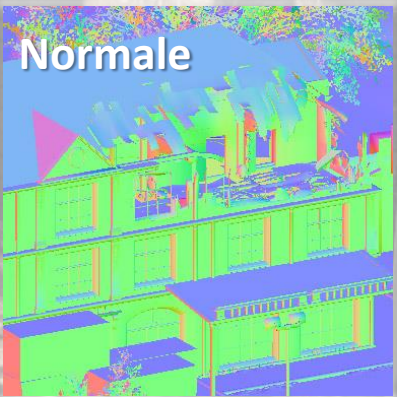


Effizientes Shading

- ▶ beschränkte Ressourcen → keine unnötige Arbeit
- ▶ Shading, Texturierung etc. ist teuer
 - ▶ keine unnötigen Berechnungen (für verdeckte Flächen)
 - ▶ Wiederverwenden von Resultaten (Bild unten, grüne Teile)



Deferred Shading



Deferred Shading

- ▶ bildbasierte Post Processing-Effekte: Spiegelungen aus



Deferred Shading

- ▶ bildbasierte Post Processing-Effekte: Spiegelungen an

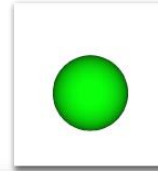


Tiefenunschärfe

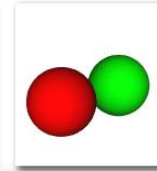


Image-Based Lighting

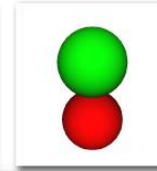
► Precomputed Radiance Transfer, Spherical Harmonics



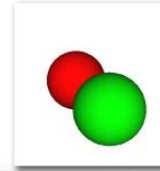
y_0^0



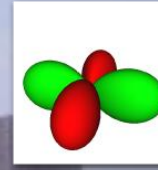
y_1^{-1}



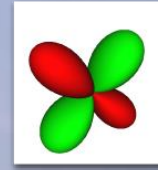
y_1^0



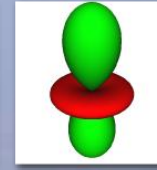
y_1^1



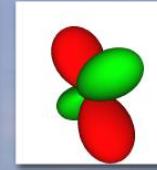
y_2^{-2}



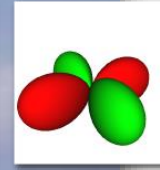
y_2^{-1}



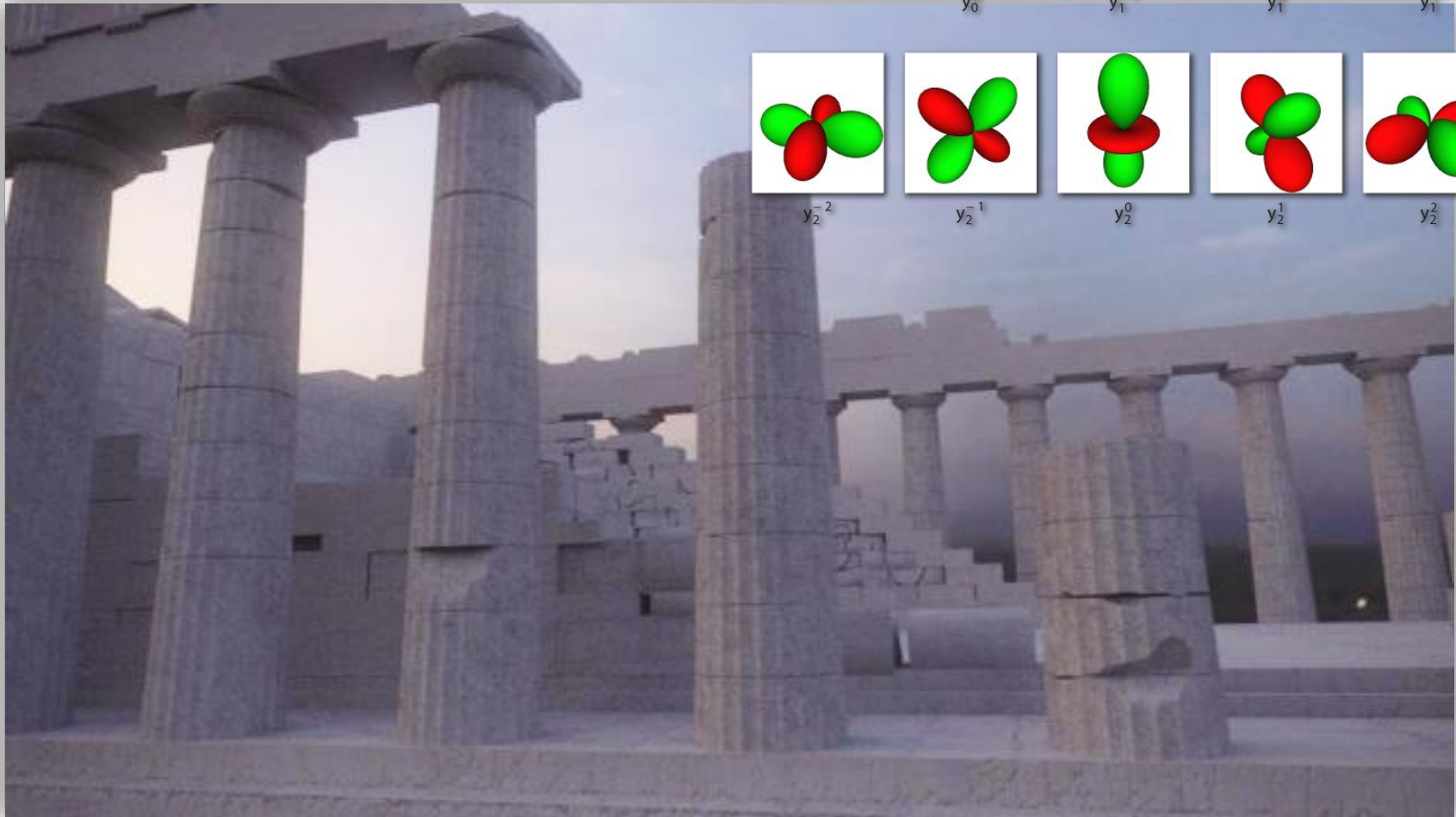
y_2^0



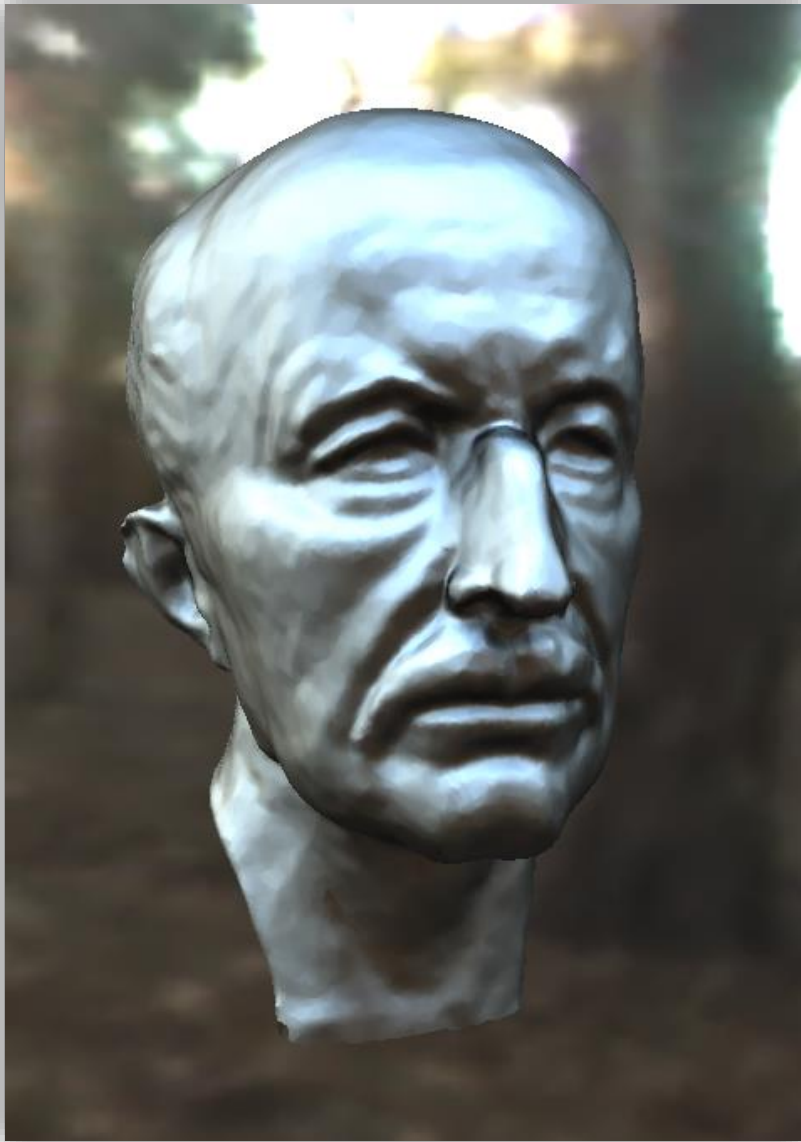
y_2^1



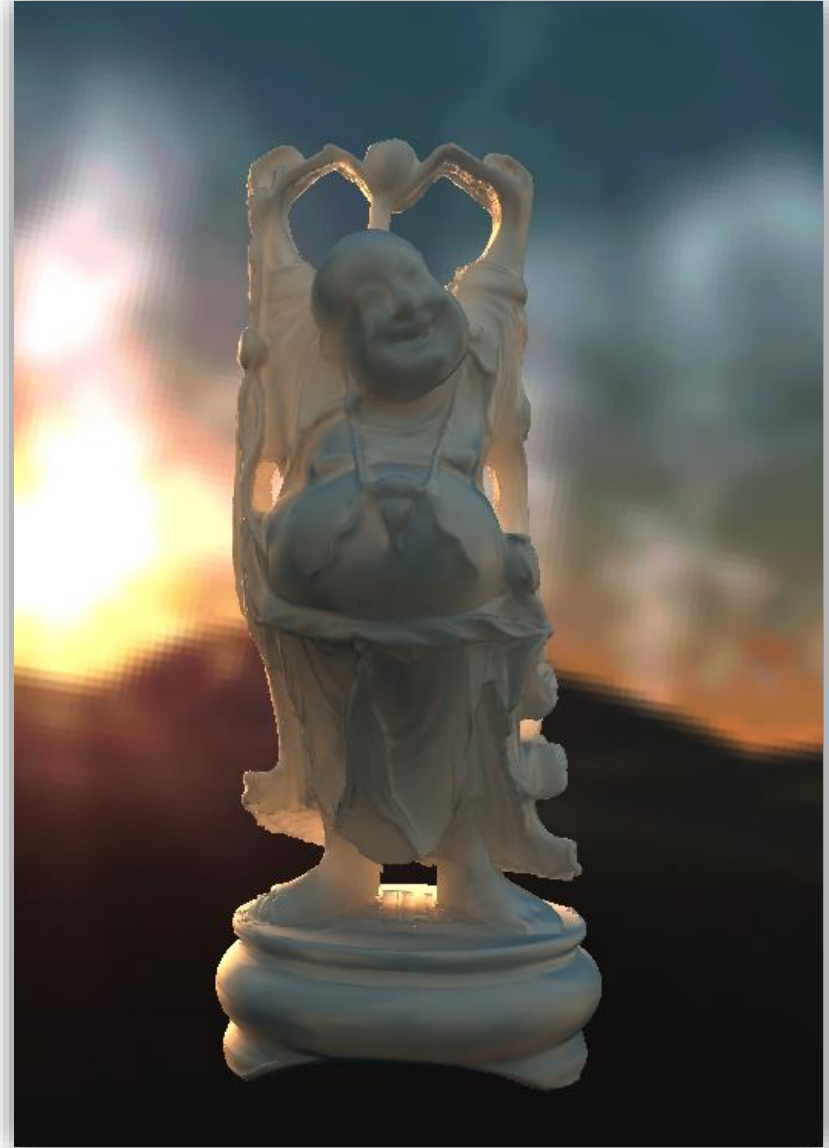
y_2^2



Precomputed Radiance Transfer



Precomputed Radiance Transfer



Tone Mapping

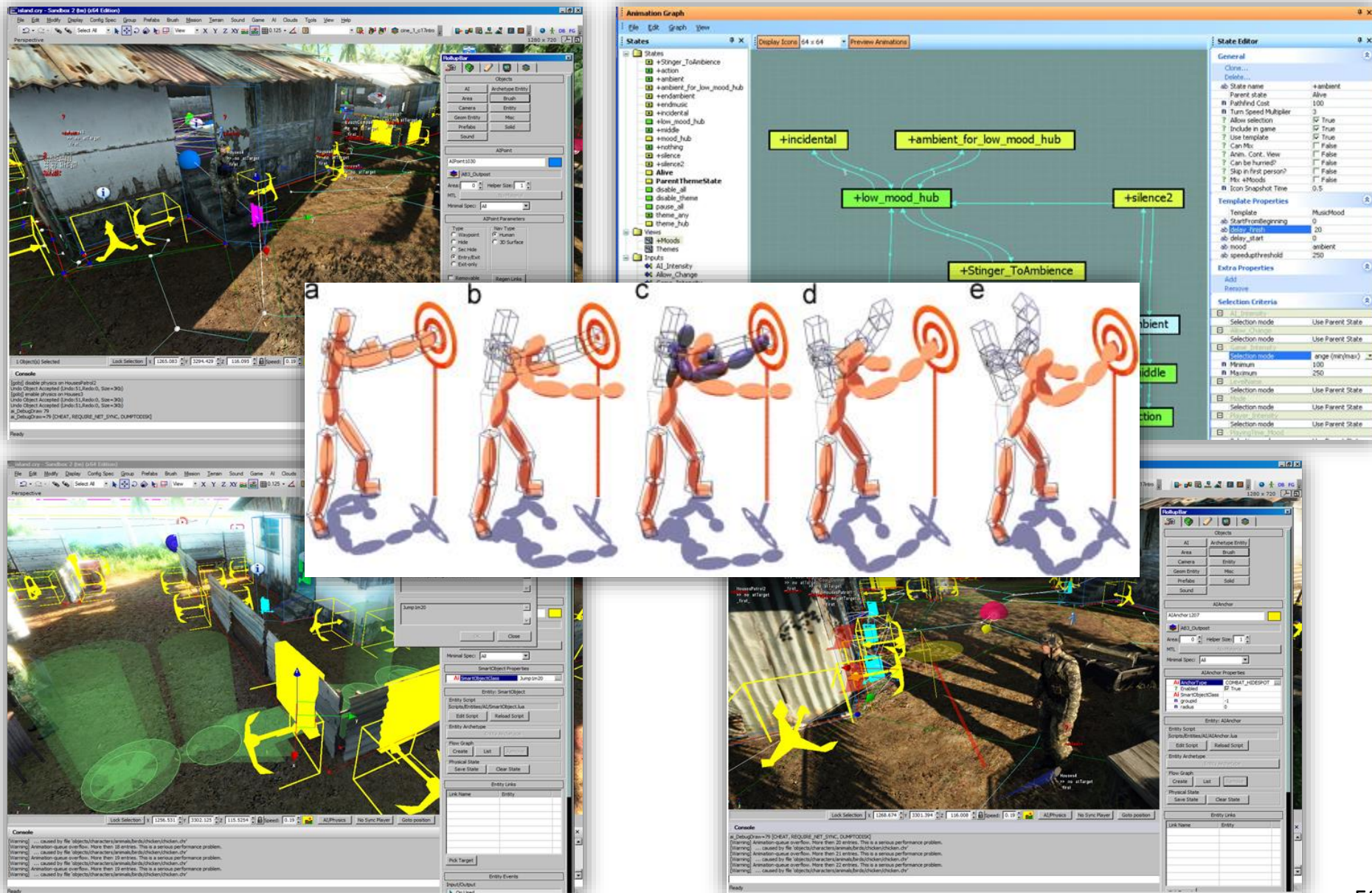
- ▶ Abbilden von berechneten Radiance-Werten auf Pixelfarben



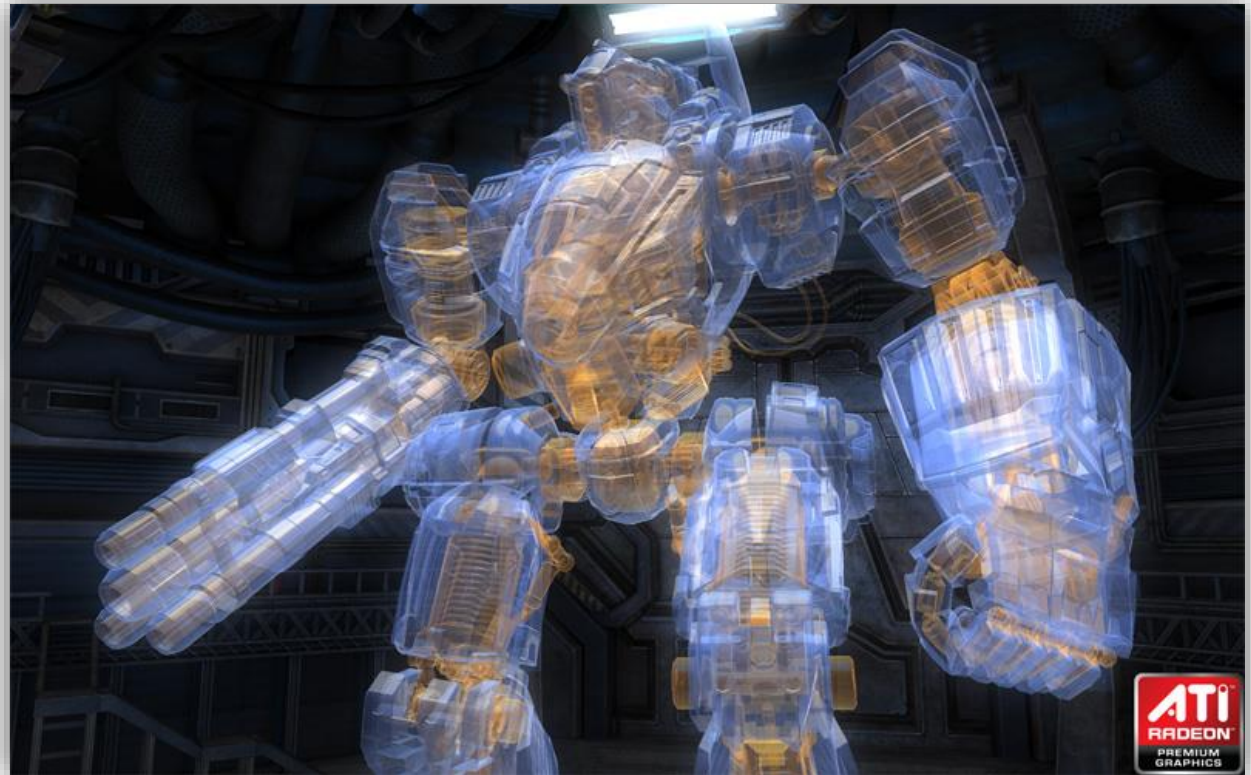
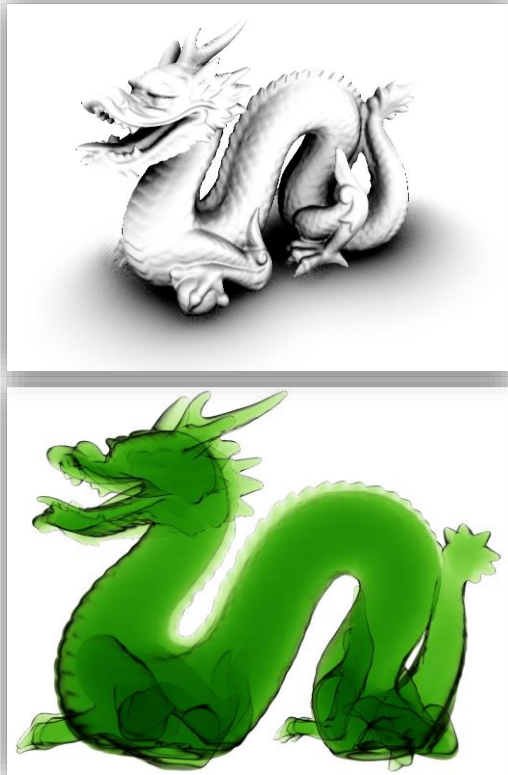
BUNGIE

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

Animation und Szenengraphen



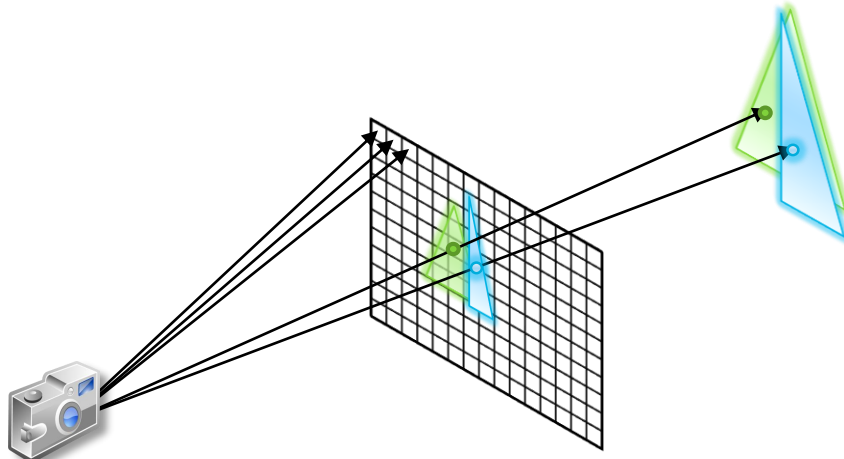
- ▶ beschränkte (Zeit-)Ressourcen →
 - ▶ verwende schnelle/dedizierte Grafik-Hardware (GPUs)
 - ▶ Rendering-Paradigma: meist Rasterisierung statt Raytracing
- ▶ lassen Sie uns kurz über GPUs sprechen...



- ▶ 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 flexibel genug für viele Anwendungen (nicht nur 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

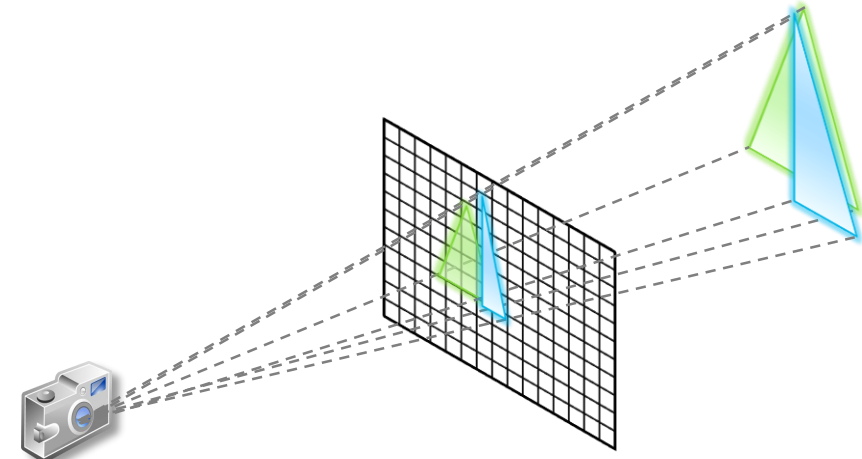
Raytracing vs. Rasterisierung

- ▶ 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



Betrachter

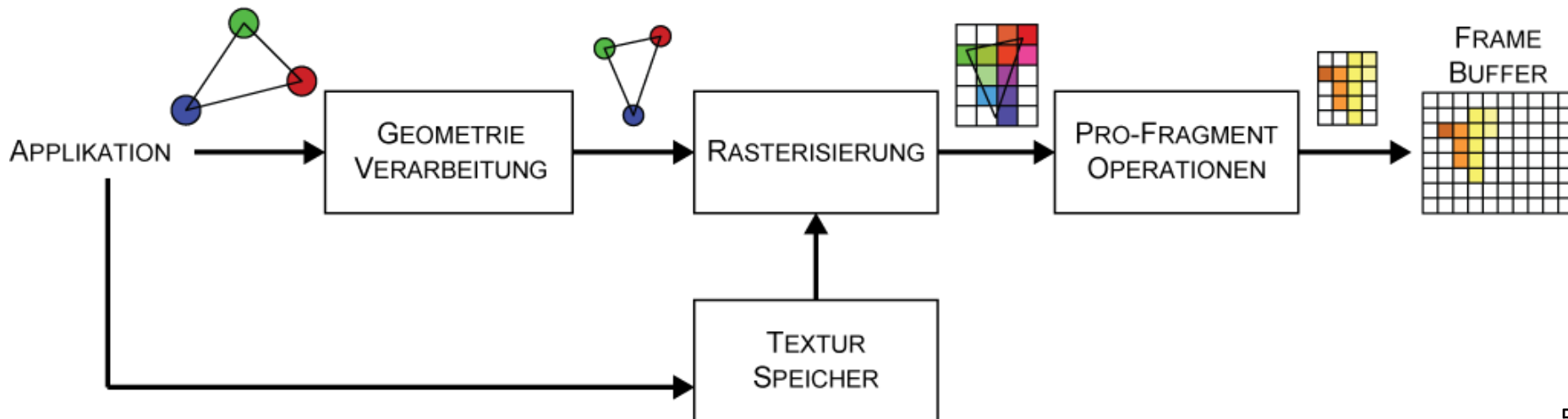
Bildebene



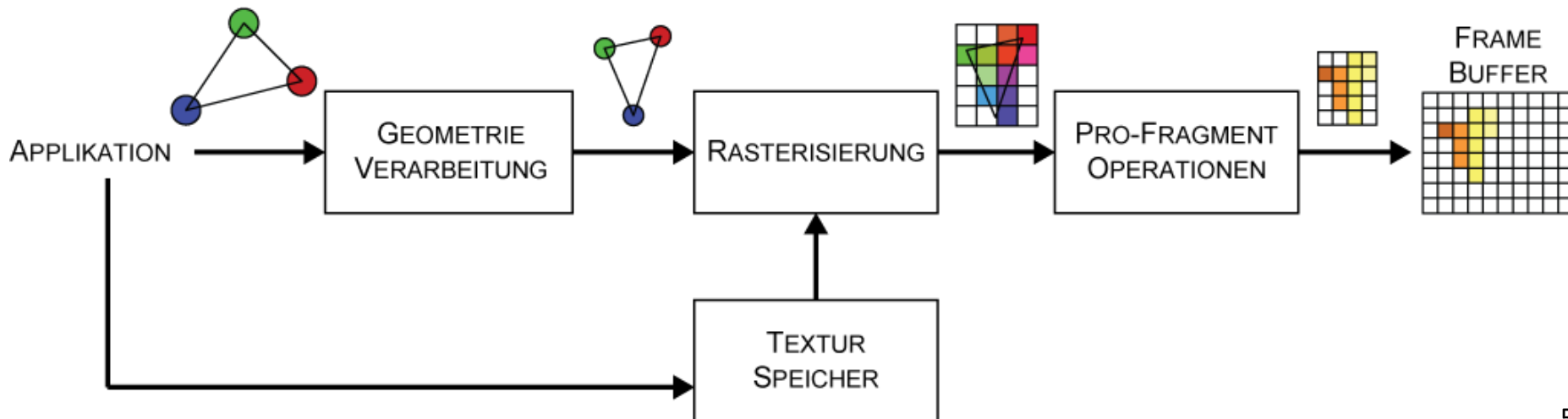
Betrachter

Bildebene

- ▶ **Dreiecke/Primitive** werden in einer **Pipeline** verarbeitet
 - ▶ „eins nach dem anderen“ (faktisch massiv-parallel), immer unabhängig voneinander
- ▶ **Geometrieverarbeitung** (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)

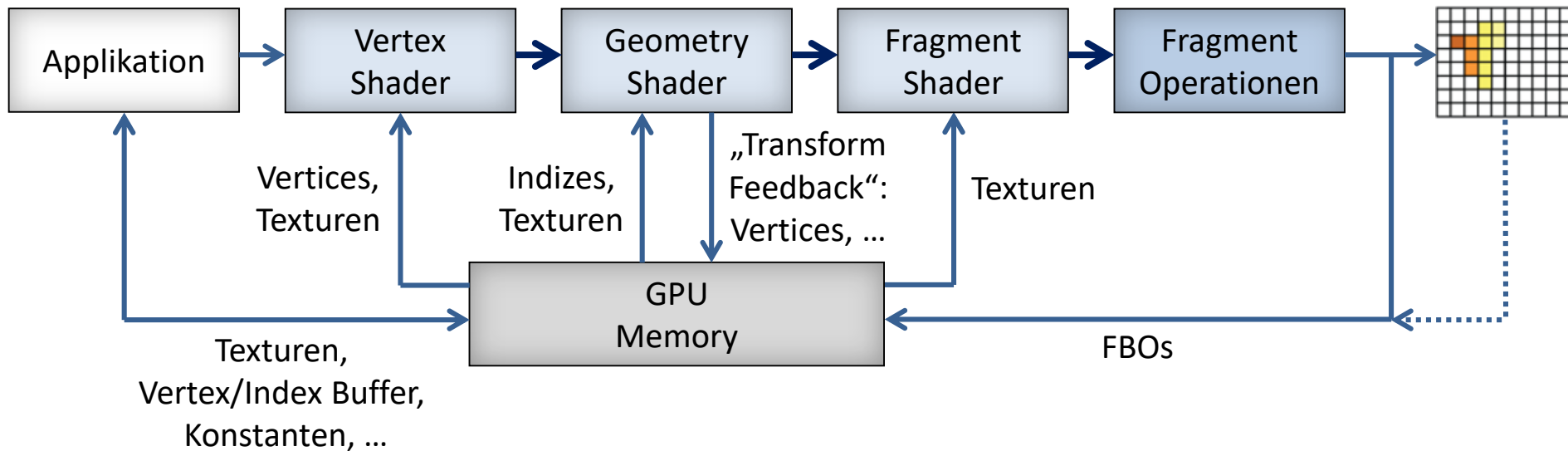


- ▶ Rasterisierung (in Verbindung mit Tiefenpuffer) ist effizient: Dreiecke durchlaufen – nacheinander, aber unabhängig – alle dieselben Verarbeitungsschritte
- ▶ Konsequenzen der Pipeline-Architektur
 - ▶ hoher Durchsatz, einfache Umsetzung (v.a. auch in Hardware)
 - ▶ **lokale Beleuchtungsmodelle**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



Moderne Grafik-Pipeline

- ▶ eine „funktionale“ Sicht auf die zunächst wichtigen programmierbaren Teile der Pipeline



- ▶ u.a. hier nicht gezeigt:
 - ▶ jeder Shader kann lesend/schreibend auf Puffer zugreifen (Ausnahme: der Framebuffer kann nur geschrieben werden)
 - ▶ Tessellierungsstufe zwischen Vertex und Geometry Shader

Ein paar Zahlen

- ▶ Infamous Second Son (PS4 Spiel):
11 Mio. Dreiecke + 120 Tsd. pro Charakter

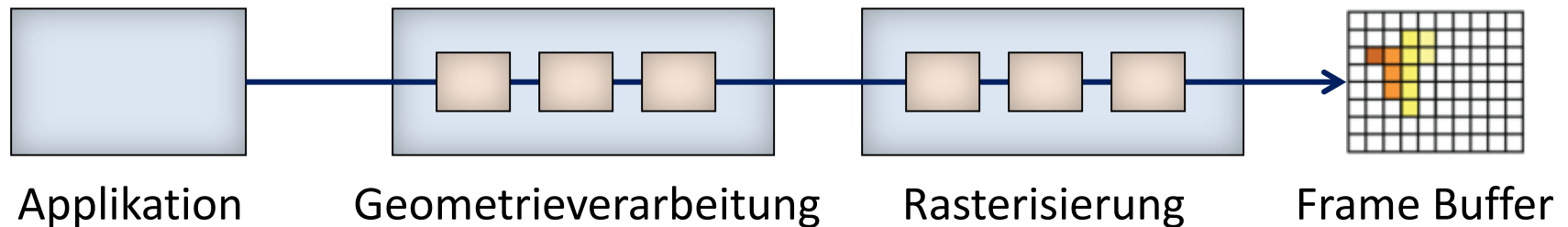


Grafik-Pipeline, etwas abstrakter



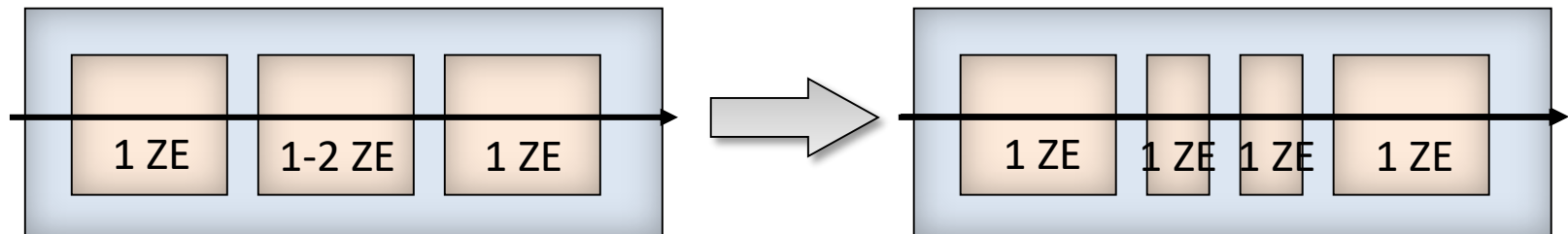
Laufzeit vs. Durchsatz

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

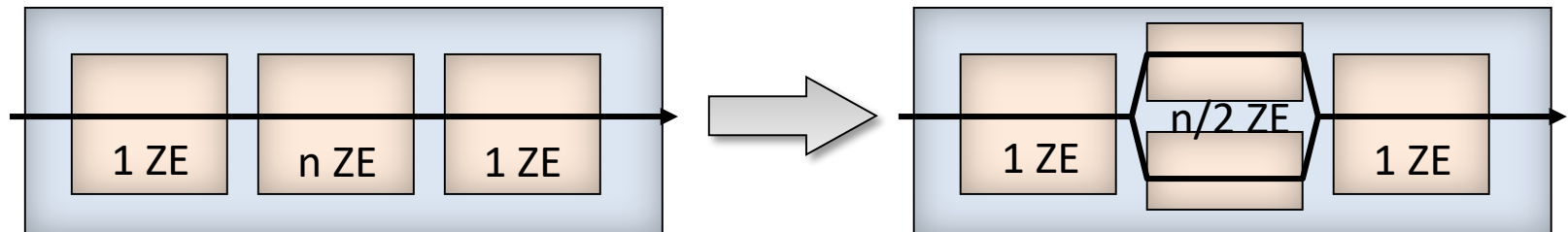


Verbesserung der Performance

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



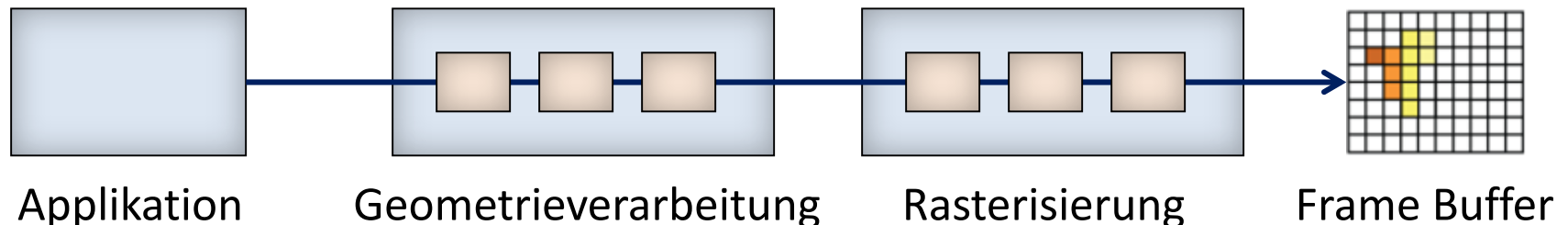
- ▶ parallele Verarbeitungseinheiten (höherer Durchsatz, u.U. kürzere Laufzeit)
 - ▶ Vertex / Geometry / Fragment Shader → unabhängige Verarbeitung



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...
 - ▶ ... Resultat eines Render-Durchgangs weiterverarbeitet werden soll
 - ▶ ... OpenGL Zustände, z.B. Texturen, Shader etc. geändert werden (deshalb sortiert man Objekte nach Material, Shader, ...)
 - ▶ heute: mehr Flexibilität dank moderner HW-Features (z.B. Texture-Arrays)
 - ▶ Flaschenhals oft die API-/Draw-Calls (→ Mantle, DirectX12, Vulkan)



Ziele und nächste Schritte...

- ▶ Ziel: performantes, (foto)realistisches Rendering in hoher Qualität
- ▶ wir lernen wichtige Konzepte für interaktive Grafik kennen, die wir an konkreten Verfahren festmachen, z.B. Vorfilterung, Level of Detail u.v.m.

- ▶ Verfahren für...
 - ▶ mehr Oberflächendetail → realistisches Aussehen
 - ▶ Schatten, Voxelisierung, ... → realistische Beleuchtung
 - ▶ effizientes Shading in komplexen Szenen → Effizienz
 - ▶ vorberechneter Lichttransport → Realismus
 - ▶ Culling und Level of Detail → Effizienz
 - ▶ ... Animation, Tone Mapping, aktuelle Forschungsthemen