




# Virtuelle Realität Real-time Rendering



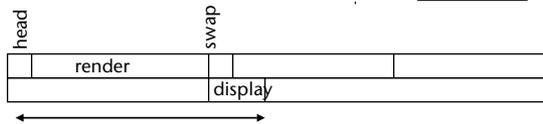
G. Zachmann  
 Clausthal University, Germany  
[cg.in.tu-clausthal.de](http://cg.in.tu-clausthal.de)




## Latenz beim Rendering

- **Klassische Pipeline:**

```

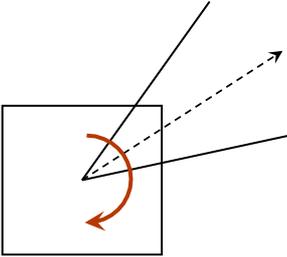
graph TD
    A[Head pos & ori] --> B[Transform]
    B --> C[Scene graph traversal]
    B --> D[Culling]
    D --> E[Clipping]
    E --> F[Viewport mapping]
    F --> G[Front buffer]
    F --> H[Back buffer]
    G --> I[Pixel scan]
    H --> I
    I --> J[DAC]
    J --> K[RGB]
            
```
- **Latenz:**

- **Idee: rendere mehr als den Viewport**

G. Zachmann Virtuelle Realität und Simulation – WS 11/12

Real-time Rendering 2

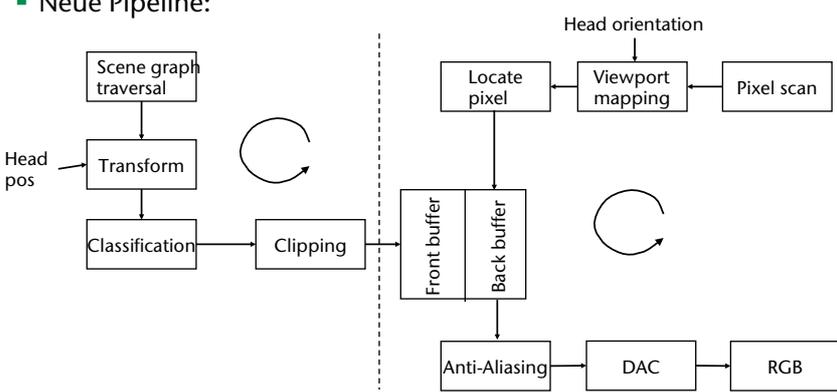
## Viewport independent rendering

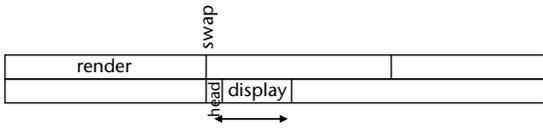
- Konzeptionelle Idee:
  - Rendere Szene auf Kugel um Betrachter
  - Bei Rotation des Viewpoints: Ausschnitt des Viewports neu wählen
  - Wähle statt Kugel einen Würfel (s.a. Cave)



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 3

## Neue Pipeline:

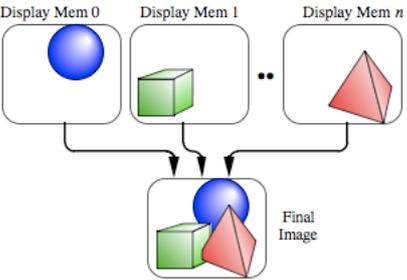


- Latenz:
 

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 4

## Image composition

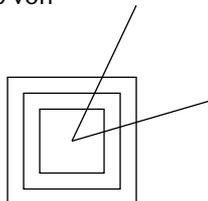
- Konzeptionelle Idee:
  - Jeder Thread rendert nur "sein" Objekt in einen eigenen Frame-Buffer
  - *Video-Hardware* liest *Frame-Buffer* inkl. *Z-Buffer* aus
  - Image Compositor kombiniert die Einzelbilder durch Z-Vergleich pro Pixel
  - In praxi: teile Menge aller Objekte in der Szene in eine kleine Anzahl von Teilmengen auf; rendere jede Teilmenge durch einen PC mit eigener Grafikkarte



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 5

## Prioritized Rendering:

- Weit entfernte / langsame Objekte bleiben länger auf dem Bildschirm "gleich"
- Idee: rendere auf mehrere (gedachte) Schalen
- Wieviele Schalen neu gerendert werden, hängt ab von
  - Geforderter Framerate
  - Szenenkomplexität
  - Viewpoint-Bewegung
  - Objekt-Bewegung
- Human factors beeinflussen Priorität:
  - Keine Kopfdrehung um 180° → Objekte "hinten" nur selten updaten
  - Manipulierte Objekte haben hohe Priorität
  - Objekte in der Peripherie seltener updaten



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 6

## Konstante Framerate durch "Weglassen"

- Gründe für eine konstante Framerate:
  - Vorhersage bei *predictive filtering* des Trackings von Kopf / Hände funktioniert nur, wenn alle nachfolgenden Stufen der Pipeline (d.h., alle Stufen) immer gleich schnell arbeiten
  - Wahrnehmung von Framerate-Sprüngen (z.B. zwischen 60 und 30 Hz)
- Rendering als "*time-critical computing*":
  - Rendering bekommt bestimmtes Zeitbudget (z.B. 17 msec)
  - Rendering-Algorithmus muß Bild produzieren "so gut wie möglich"
- Techniken des "*Weglassens*":
  - *Levels-of-Details (LOD)*
  - Unsichtbare Geometrie weglassen (*Culling*)
  - *Image-based rendering*
  - *Lighting-Model* reduzieren, Texturen reduzieren,
  - ... ?

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 7

## Die Level-of-Detail-Technik

- Definition:
 

Ein *Level-of-Detail (LOD)* eines Objektes ist eine *reduzierte Version* (d.h., weniger Polygone) einer (hoch-aufgelösten) Geometrie.
- Beispiel:



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 8

- Idee: verwende den LOD "passend" zur Entfernung vom Viewpoint

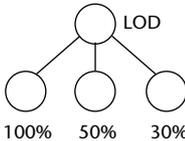


- Zwei Aufgaben:
  1. Preprocessing: Generierung von LODs
  2. Runtime: Auswahl des "richtigen" LODs, und ggf. "Blending"

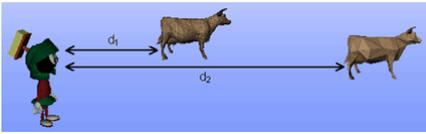
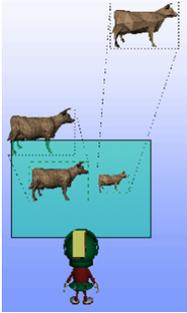
G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 9

## LOD-Selektion

- Visuelle Güte gegen zeitliche Güte
- Wie selektiert man den "richtigen" Level?

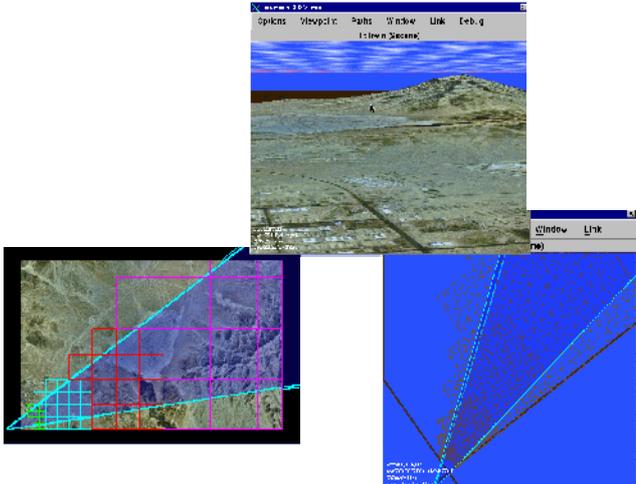


- Statisch:
  - Level  $i$  für Entfernungsbereich  $(d_i, d_{i+1})$
  - Abhängig von FOV
  - Größe des Obj unberücksichtigt
- Dynamisch:
  - Schätze Größe des Objektes auf dem Bildschirm
  - Unabhängig von Bildschirmauflösung, FOV, Größe des Obj
  - Ist automatisch entfernungsabhängig

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 10

■ Für Desktop-Applikationen i.A. schon ausreichend:



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 11

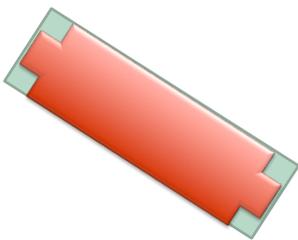
■ Größenabschätzung des Objektes auf dem Screen

■ Naive Methode:

- Berechne Bounding-Box (bbox) des Objektes in 3D
- Projiziere Bbox nach 2D → 8 2D-Punkte
- Bestimme 2D-Bbox (achsenparallel)

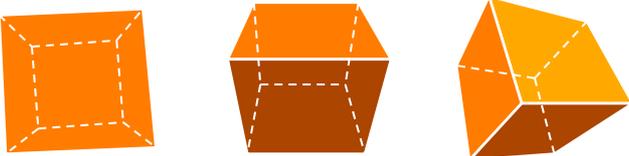
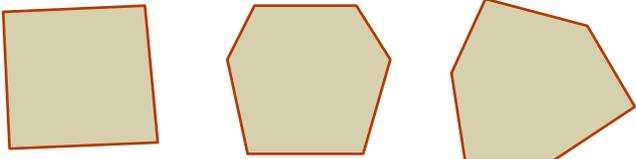
■ Genauere Methode:

- Bestimme wahre Fläche der projizierten 3D-Bbox auf dem Screen



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 12

### Idee des Algorithmus

- Bestimme die Anzahl der Seiten der Box, die in 2D sichtbar sind:
 
- Projiziere die Punkte auf der Silhouette (4 oder 6) nach 2D:
 
- Berechne den Betrag dieser (konvexen!) Fläche

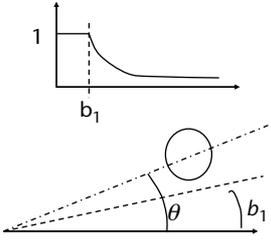
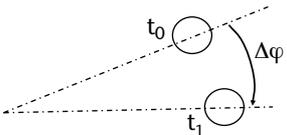
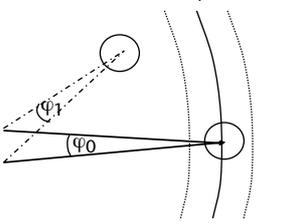
G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 13

### Implementierung

- Für jedes Paar von (parallelen) Seitenflächen (jeden **Slab**): klassifiziere Viewpoint bzgl. dieses Paares als "darunter", "darüber", oder "dazwischen"
- Ergibt  $3 \times 3 \times 3 = 27$  Möglichkeiten
  - M.a.W.: die Ebenen der Seitenflächen eines Würfels unterteilen den Raum in 27 Teilräume
- Verwende Bit-Codes (à la Out-Codes) und eine Lookup-Table
  - Ergibt  $2^6$  theoretische Einträge
  - 27-1 Einträge der LUT listen die 4 oder 6 Vertices der Silhouette
  - Dann nur noch projizieren, triangulieren, Flächen aufsummieren

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 14

## Psychophysiologische LOD-Selektion

- Faktoren bzgl. der Sehschärfe:
  - Zentrum/Peripherie:
 
$$k_1 = \begin{cases} e^{-(\theta-b_1)/c_1} & , \theta > b_1 \\ 1 & , \text{sonst} \end{cases}$$

  - Bewegung:
 
$$k_2 = e^{-\frac{\Delta\varphi-b_2}{c_2}}$$

  - Unschärfe/Tiefe:
 
$$k_3 = e^{-\frac{|\varphi_0-\varphi|-b_3}{c_3}}$$


G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 15

- Bestimmung des LODs:
  - $k = \min\{k_i\} \cdot k_0$  , oder  $k = \prod k_i \cdot k_0$
  - $r_{\min} = 1/k$
  - Selektiere Level  $l$  so, daß
$$\forall p \in P_l : r(p) \geq r_{\min}$$
- Braucht man Eye-Tracking?
  - Teuer, ungenau, "intrusive"
  - Psychophysiologie: Augen immer  $< 15^\circ$  ausgelenkt
  - Treffe die Annahme Kopfrichtung=Augenrichtung, und wähle  $b_1=15^\circ$

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 16

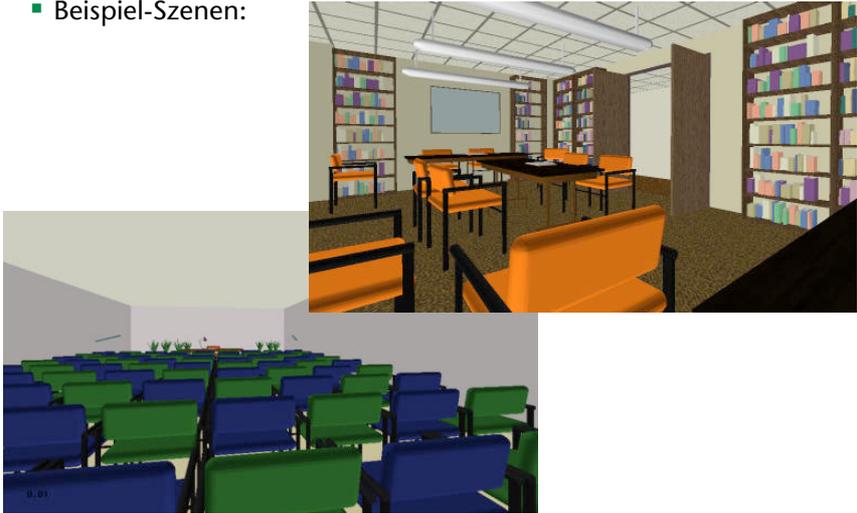
## Reaktive vs. Prädiktive LOD-Selektion

- Reaktive LOD-Selektion:
  - Verwalte Historie von Rendering-Zeiten
  - Bestimme daraus die geschätzte Zeit  $T_r$  für das kommende Frame
  - Sei  $T_b$  = das zur Verfügung stehende Zeit-Budget
  - Falls  $T_r > T_b$  : erniedrige LODs (gröbere Levels)
  - Falls  $T_r < T_b$ : wähle feinere Levels
  - Dann rendern und tatsächlich benötigte Zeit in die Historie aufnehmen

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 17

## Reaktive LOD-Selektion hat Ausreißer

- Beispiel-Szenen:
 



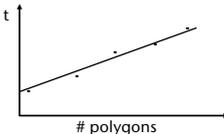
G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 18

## Prädiktive LOD-Selektion [Funkhouser und Sequin]

- Definition **Objekt-Tupel (O,L,R)**:  
 O = Objekt, L = Level,  
 R = Rendering Algo (#Texturen, anti-aliasing, #Lichtquellen)
- Bewertungsfunktionen für Objekt-Tupel:  
 Cost(O,L,R) = Rendering-Zeit  
 Benefit(O,L,R) = "Beitrag zum Bild"
- Optimierungsproblem:  
 suche 
$$\max_{S' \subset S} \sum_{(O,L,R) \in S'} \text{benefit}(O, L, R)$$
  
 unter der Bedingung 
$$T_r = \sum_{(O,L,R) \in S'} \text{cost}(O, L, R) \leq T_b$$
  
 wobei  $S = \{ \text{möglichen Objekt-Tupel in der Szene} \}$

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 19

- Cost-Funktion hängt ab von:
  - Anzahl Eckpunkte (~ # Koord. Trafos + Lighting-Calcs + Clipping)
  - Setup pro Polygon
  - Anzahl Pixel (Scanline-Konvertierung, Alpha-Blending, Textur, Anti-Aliasing, Phong)
  - Theor. Kostenmodell:
$$\text{Cost}(O, L, R) = \max \left\{ \begin{array}{l} C_1 \cdot \text{Poly} + C_2 \cdot \text{Vert} \\ C_3 \cdot \text{Pixels} \end{array} \right\}$$
- Besser experimentell bestimmen:  
 Rendere versch. Objekte mit allen  
 möglichen verschiedenen Parametern



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 20

- Benefit-Funktion: in den "Beitrag" zum Bild gehen ein
  - Größe
  - Shading-Methode:  $\text{Rendering}(O, L, R) = \begin{cases} 1 - \frac{c}{\text{pgons}} & , \text{flat} \\ 1 - \frac{c}{\text{vert}} & , \text{Gouraud} \\ 1 - \frac{c}{\text{vert}} & , \text{Phong} \end{cases}$
  - Distanz vom Zentrum (Peripherie, Tiefe)
  - Geschwindigkeit
  - Semantische "Wichtigkeit"
  - Hysterese:
 
$$\text{Hysterese}(O, L, R) = \frac{c_1}{1 + |L - L'|} + \frac{c_2}{1 + |R - R'|}$$
  - Zusammen:
 
$$\text{Benefit}(O, L, R) = \text{Size}(O) \cdot \text{Rendering}(O, L, R) \cdot \text{Importance}(O) \cdot \text{OffCenter}(O) \cdot \text{Vel}(O) \cdot \text{Hysteresis}(O, L, R)$$

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 21

- Optimierungsproblem: "*multiple-choice knapsack problem*"  
→ NP-vollständig
- Idee: berechne sub-optimale Lösung :
  - Rückführung auf kontinuierliches Rucksackproblem
  - Dann greedy lösen mit zusätzlichem Constraint
  - Definiere
 
$$\text{value}(O, L, R) = \frac{\text{benefit}(O, L, R)}{\text{cost}(O, L, R)}$$
  - Sortiere alle Objekt-Tupel nach  $\text{Value}(O, L, R)$
  - Wähle die ersten Objekt-Tupel bis Rucksack voll
  - Constraint: keine 2 Objekt-Tupel dürfen dasselbe Objekt darstellen

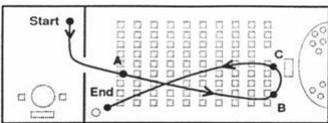
G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 22

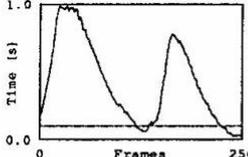
■ Inkrementelle Lösung:

- Starte mit der Lösung  $(O'_1, L'_1, R'_1), \dots, (O'_n, L'_n, R'_n)$  vom letzten Frame
- Falls
 
$$\sum_i \text{cost}(O_i, L_i, R_i) \leq \text{max. Frame-Zeit}$$
 dann suche Objekt-Tupel  $(O_k, L_k, R_k)$ ,  
 so daß
 
$$\text{value}(O_k, L_k + a, R_k + b) - \text{value}(O_k, L_k, R_k) = \text{max}$$
 und
 
$$\sum_{i \neq k} \text{cost}(O_i, L_i, R_i) + \text{cost}(O_k, L_k + a, R_k + b) \leq \text{max. Frame-Zeit}$$
- Analog, falls  $\sum_i \text{cost}(O_i, L_i, R_i) > \text{max. Frame-Zeit}$

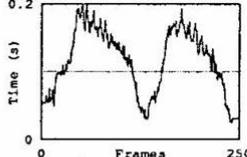
G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 23

■ Performance in der Beispielszene:

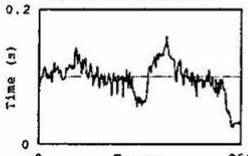




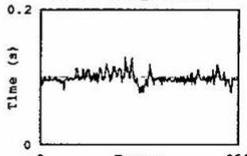
a) No detail elision.



b) Static algorithm.



c) Feedback algorithm.



d) Optimization algorithm.

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 24

### Screenshots aus der Beispielszene



No detail elision, 19,821 polygons

Optimization, 1,389 polys,  
0.1 sec/frame target frame time

Level of detail: darker gray means more detail



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 25

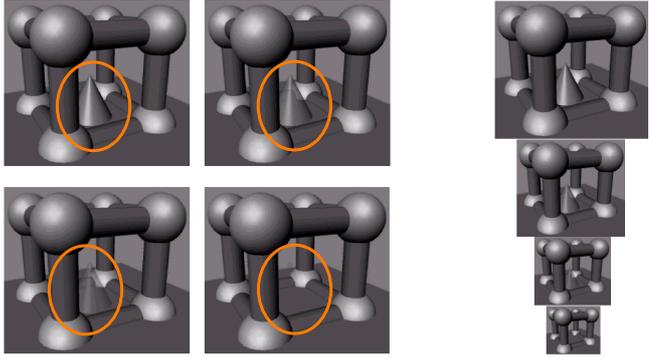
### Probleme von diskreten LODs

- "Popping" beim Umschalten zwischen Levels
- Maßnahmen gegen "Popping":
  - Hysterese
  - Alpha-Blending der beiden benachbarten LOD-Stufen
    - Man kommt vom Regen in die Traufe ;-)
  - Kontinuierliche, view-dependent LODs

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 30

## Alpha-LODs

- Einfache Idee, um Popping zu vermeiden:  
when beyond a certain range, fade out object until gone



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 31

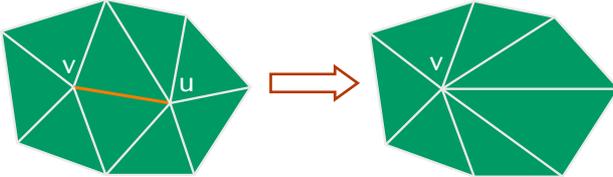
## Progressive Meshes

- A.k.a. **Geomorph-LODs**
- Idee:
  - Gegeben zwei Meshes (LODs desselben Objektes)  $M_i$  und  $M_{i+1}$
  - Erzeuge ein Mesh  $M'$  "zwischen" diesen beiden
- Definition: **Progressive Mesh** = Repräsentation eines Objektes, ausgehend von einem hoch-aufgelösten Mesh  $M_0$ , mit Hilfe derer man (fast) stufenlos zwischen 1 Polygon und  $M_0$  "Zwischen-Meshes" generieren kann (möglichst schnell).

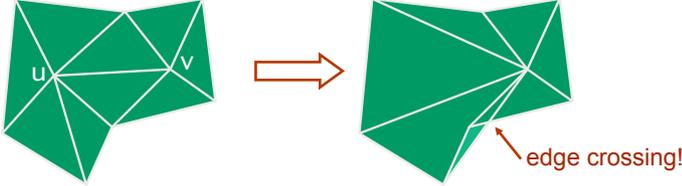
G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 32

## Erzeugung von Progressive Meshes

- Simplifizierung (*simplification*)
- Grundlegende Operation: *edge collapse*

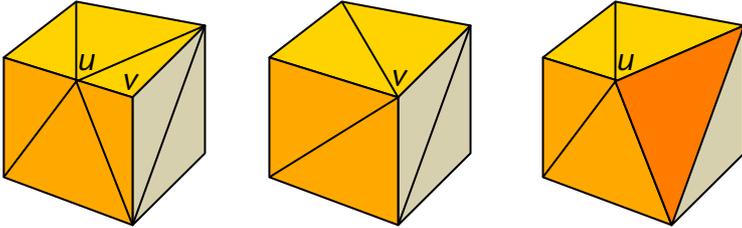


- Bad Edge Collapses:



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 33

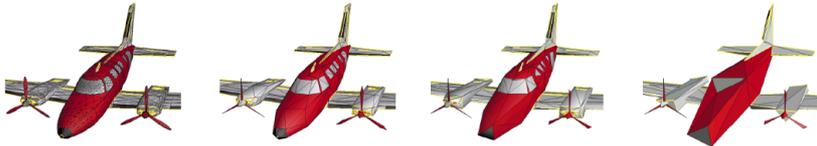
- Reihenfolge der Edge Collapses:
  - Führe Bewertung eines Edge-Collapses ein, die visuellen Effekt "misst"



- Führe zunächst Edge Collapses mit kleinster Auswirkung aus
- Nach jedem Schritt müssen im Prinzip alle Kanten neu bewertet werden

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 34

- Bewertungsfunktion für Edge-Collapses ist nicht trivial und, vor allem, wahrnehmungsbasiert!
- Beeinflussende Faktoren:
  - Krümmung der Kanten / Fläche
  - Beleuchtung, Texturierung, Viewpoint (Highlights!)
  - Bedeutung der Geometrie (Augen & Mund sind besonders wichtig)
- Beispiel eines Progressive Mesh:

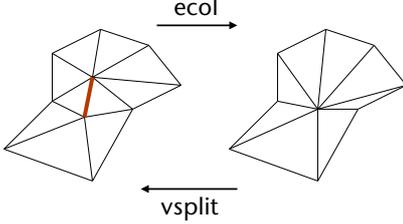


G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 35

- Repräsentation eines Progressive Meshes:

$$M = M^n \begin{array}{c} \xleftarrow{\text{ecol}_{n-1}} \\ \xrightarrow{\text{vsplit}_{n-1}} \end{array} \dots \begin{array}{c} \xleftarrow{\text{ecol}_1} \\ \xrightarrow{\text{vsplit}_1} \end{array} M^1 \begin{array}{c} \xleftarrow{\text{ecol}_0} \\ \xrightarrow{\text{vsplit}_0} \end{array} M^0$$

- $M^{i+1} = i$ -te Verfeinerung (refinement) = 1 Vertex mehr als  $M^i$



- Repräsentation eines Edge Collapse / Vertex Split:
  - Betroffenes Paar von Vertices (Kante)
  - Position des "neuen" Vertex
  - Dreiecke, die gelöscht / eingesetzt werden

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 36

### Eine einfache Bewertungsfunktion

- Heuristik:
  - Kleine Kanten zuerst entfernen
  - Einen Vertex  $U$  auf einen Vertex  $V$  ziehen, falls die Fläche um  $U$  eine geringe (diskrete) Krümmung hat
- Ein einfaches Kostenmaß für einen Edge-Collapse von  $U$  auf  $V$ :
 
$$\text{cost}(U, V) = \|U - V\| \cdot \text{curv}(U)$$

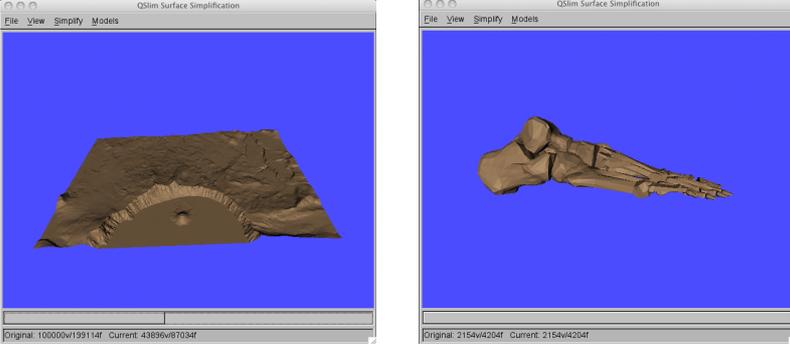
$$\text{curv}(U) = \frac{1}{2} \left( 1 - \min_{f \in T(U) \setminus T(V)} \max_{i=1,2} \mathbf{n}_f \mathbf{n}_i \right)$$

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 37

- Bemerkung:
 
$$\text{cost}(U, V) \neq \text{cost}(V, U)$$
- Beispiel:

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 38

## Demo



[Michael Garland: Qslim]

Wie funktioniert der Funkhouser-Sequin-Algo mit progressiven Meshes?

Diplomarbeit ...

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 39

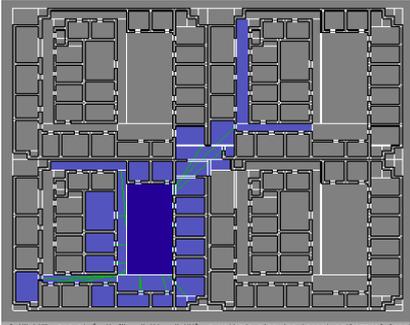
## Exkurs: andere LODs

- Idee: LOD-Technik auf nicht-geometrische Inhalte anwenden
- Z.B. "*Behavioral LOD*":
  - Simuliere Verhalten eines Objektes exakt, wenn im Fokus, sonst nur "grob"

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 40

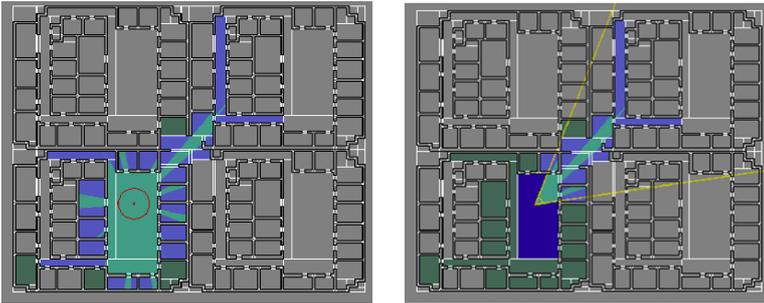
## Culling in Gebäuden (portal culling)

- Beobachtung: viele Räume innerhalb des Viewing-Frustums sind nicht sichtbar
- Idee:
  - Unterteile Raum in "Zellen"
  - Berechne *Cell-to-Cell-Visibility* im voraus



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 41

- Von bestimmtem Viewpoint aus ist innerhalb der Zelle noch weniger sichtbar:



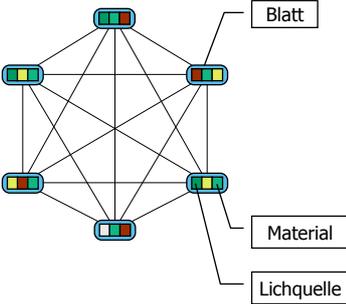
G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 42

## State Sorting

- State = Zustand =
  - Zusammenfassung aller Attribute
  - Beispiele für Attribute: Farbe, Material, Lighting-Param., Textur, Blend-Fkt., Shader-Programm, etc.
  - Jedes Attribut hat zu jedem Zeitpunkt genau 1 Wert aus einer endlichen Menge
- State-Wechsel sind einer der schlimmen Performance-Killer
- Kosten:
 
  - Matrix-Stack-Modifikation
  - Lighting-Modifikation
  - Textur-Modifikation
  - Shader-Programm-Modifikation
- Ziel: kompletten Szenengraphen mit minimaler Anzahl State-Wechsel rendern
- "Lösung": Pre-Sorting

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 43

- Problem: die optimale Lösung ist NP-vollständig
- Denn:
  - Jedes Blatt ist ein Knoten in einem vollständigen Graphen,
  - Kosten einer Kante = Kosten der State-Wechsel (verschiedene State-Wechsel kosten verschieden viel, z.B. sind neue Trafos relativ billig),
  - Gesucht: kürzester Weg
    - *Traveling Salesman Problem*
- Weiteres Problem: klappt nicht bei dynamischen Szenen und Occlusion Culling



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 44

## Der Sorting-Buffer

- Idee & Abstraktion:
  - Betrachte nur ein Attribut ("Farbe")
  - Schalte Buffer zwischen App. und Graphikkarte
  - Buffer enthält Elemente mit verschiedenen Farben
  - Pro Schritt eine von drei Operationen:
    - Element direkt an Graphikkarte weiterreichen
    - Element lesen und in Buffer schreiben
    - Teilmenge aus Buffer löschen und an Graphikkarte schicken

Objektsequenz      Sortierpuffer      Graphik-Hardware

G. Zachmann    Virtuelle Realität und Simulation – WS 11/12
Real-time Rendering    45

- Zwei Algorithmen-Klassen:
  - "Online"-Algorithmen: Algo kennt *nicht* zukünftige Elemente!
  - "Offline"-Algorithmen: Algo kennt *alle* Elemente, muß aber trotzdem Buffer verwenden
- Betrachte nur "lazy" online-Strategie:
  - Elemente werden nur bei Buffer-Overflow aus Buffer entfernt
  - Jede nicht-lazy Online-Strategie läßt sich in eine lazy Strategie mit gleichen Kosten umwandeln
- Frage: welche Elemente muß man bei Buffer-Overflow auswählen, damit minimale Anzahl Farbwechsel auftritt?

G. Zachmann    Virtuelle Realität und Simulation – WS 11/12
Real-time Rendering    46

## Competitive Analysis

- Definition *c-competitive* :  
 Sei  $C_{off}(k)$  die Kosten (Anzahl Farbwechsel) der optimalen Offline-Strategie,  $k$  = Buffer-Größe.  
 Sei  $C_{on}(k)$  die Kosten der Online-Strategie.  
 Dann heißt diese Strategie "*c-competitive*" gdw.
 
$$C_{on}(k) = c \cdot C_{off}(k) + a$$
 wobei  $a$  von  $k$  unabhängig ist.
- Gesucht: Online-Strategie mit möglichst kleinem  $c$   
 (im worst-case, und – wichtiger noch – im average case)

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 47

## Beispiel: LRU (least-recently used)

- Strategie:
  - Pro Farbe ein Timestamp (**nicht pro Element!**)
  - Element wird in Buffer geschrieben → Timestamp seiner Farbe wird auf aktuelle Zeit gesetzt  
 - Achtung: dabei können die Timestamps anderer Elemente im Buffer auch aktualisiert werden
  - Buffer-Overflow: entferne Elemente, deren Farbe ältesten Timestamp hat
- Untere Schranke für die Competitive-Ratio:  $\Omega(\sqrt{k})$
- Beweis durch Beispiel:
  - Setze  $m = \sqrt{k - 1}$ , oBdA  $m$  gerade
  - Wähle die Eingabe  $(c_1 \dots c_m x^k c_1 \dots c_m y^k)^{\frac{m}{2}}$
  - Kosten der **Online**-LRU-Strategie:  $(m + 1) \cdot 2 \cdot m/2 > k$  Farbwechsel
  - Kosten der **Offline**-Strategie: Ausgabe  $(x^k y^k)^{\frac{m}{2}} c_1^m \dots c_m^m$   
 →  $2m$  Farbwechsel

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 48



## Bounded-Waste- & Random-Choice-Strategie

- Idee: zähle Platzbedarf jeder Farbe im Buffer über die gesamte bisherige Zeit aufsummiert
- Führe Waste-Zähler  $W(c)$  ein:
  - Bei Farbwechsel: erhöhe  $W(c)$  um Anzahl Elemente im Buffer mit Farbe  $c$
- Bounded-Waste-Strategie:
  - Bei Buffer-Overflow entferne alle Elemente mit Farbe  $c'$ , mit  $W(c')$  maximal
- Competitive Ratio (o.Bew.):  $O \log^2 k$
- Random-Choice-Strategie:
  - Randomisierte Version von Bounded-Waste
  - Wähle uniform zufälliges Element aus Buffer, entferne alle Elemente mit derselben Farbe
  - Damit: häufige Farbe wird häufiger ausgewählt, über die Zeit ergibt sich gerade  $W(c)$

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 49



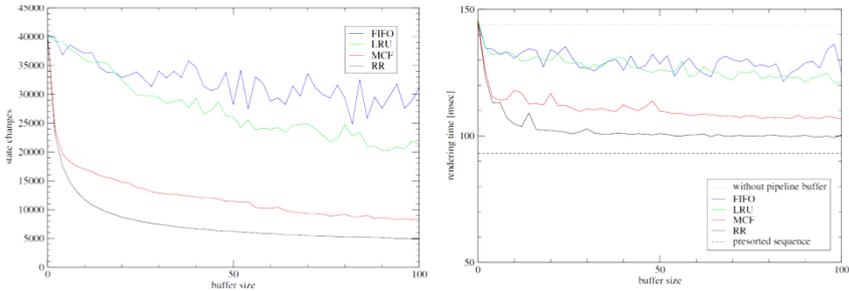
## Round-Robin-Strategie

- Problem: Generierung guter Zufallszahlen dauert rel. lange
- RR-Strategie:
  - Variante von Random-Choice
  - Wähle nicht zufälligen Slot im Buffer, sondern den gemäß eines Pointers
  - Pointer wird in Round-Robin-Manier weitergeschaltet

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 50

## Vergleich

- Fazit:
  - Round-Robin ist sehr gut (obwohl sehr einfach)
  - Worst-Case sagt sehr wenig über prakt. Perf.

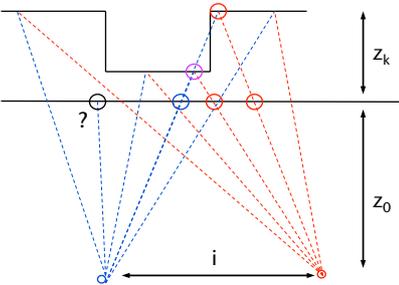
G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 51

## Stereo *ohne* 2x rendern (einfaches Image-Warping)

- Beobachtung: linkes und rechtes Bild unterscheiden sich wenig
- Idee: 1x für rechts rendern, dann Pixel verschieben
- Algo: betrachte alle Pixel auf jeder Scanline *von rechts nach links*, zeichne Pixel  $k$  an neuer x-Pos.

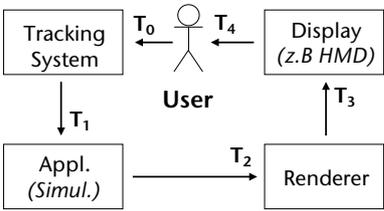
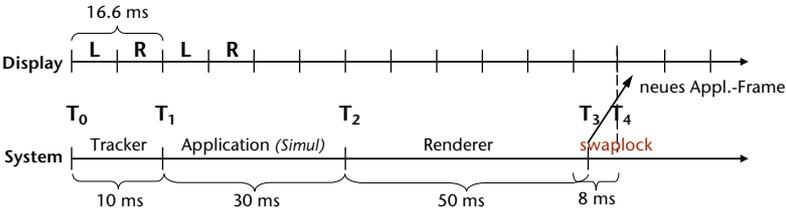
$$x'_k = x_k + \frac{i}{\Delta} \frac{z_k}{z_k + z_0}, \quad \Delta = \text{Pixelbreite}$$

- Probleme:
  - Löcher!
  - Up-Vektor muß senkrecht sein
  - Reflexionen und specular highlights sind an falscher Pos
  - Aliasing



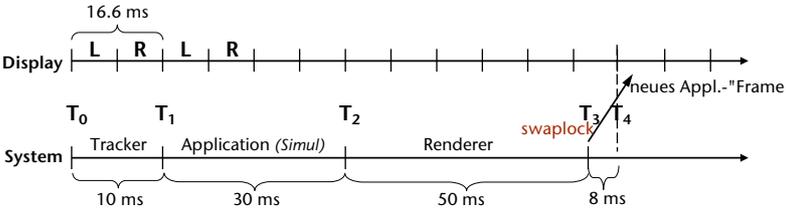
G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 52

## Image Warping

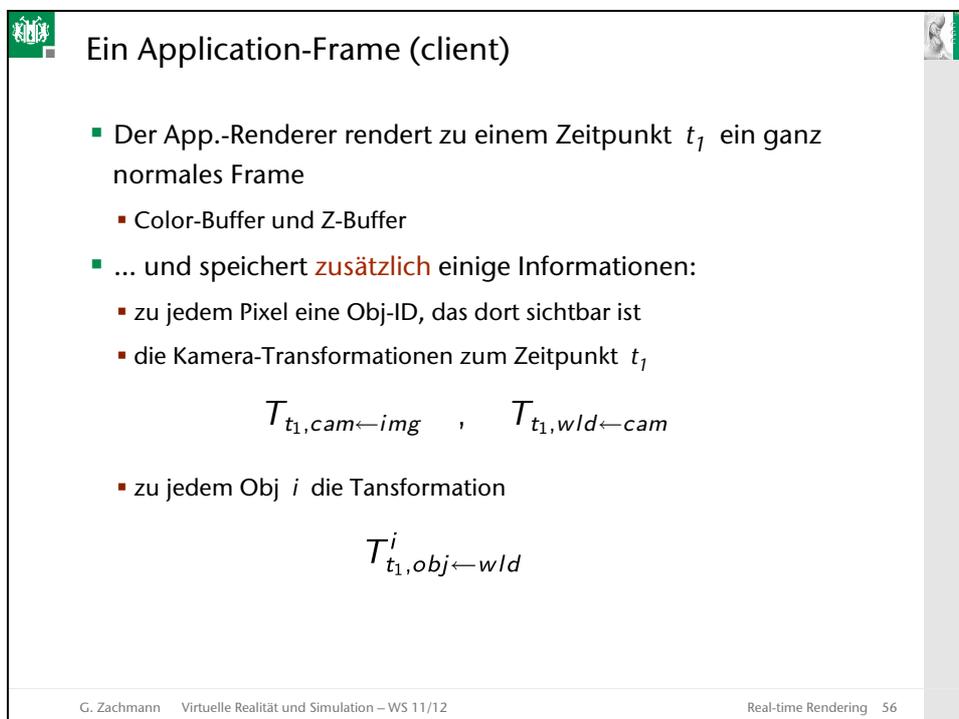
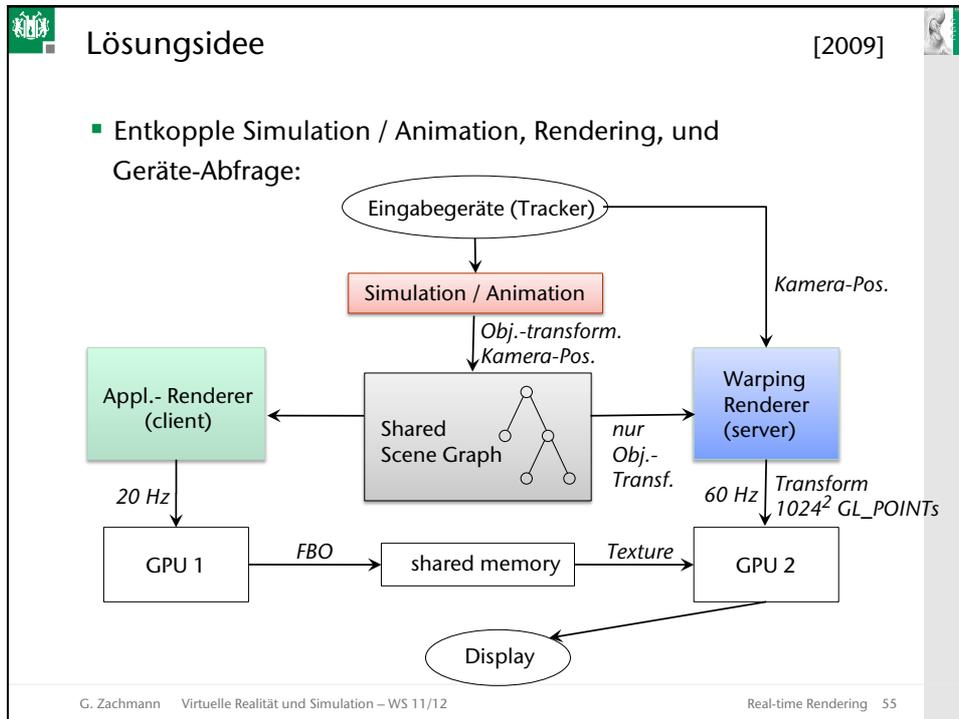
- Ein naives VR-System:
 
- Latenz in diesem System (Stereo mit 60 Hz → Display-Refresh = 120 Hz):
 

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 53

- Probleme / Beobachtungen:
  - Die Appl.-Framerate (inkl. Rendering) ist typischerweise viel langsamer als die Display-Refresh-Rate
  - Die Tracking-Werte, die zu einem bestimmten Bild geführt haben, liegen sehr weit in der Vergangenheit
  - Der Tracker könnte wesentlich öfter aktuelle Werte liefern
  - Die Frames unterscheiden sich (normalerweise) rel. wenig voneinander (**temporal coherence**)



G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 54



### Warping eines Frames (server)

- Zu einem späteren Zeitpunkt  $t_2$  generiert der Server ein Bild aus dem Application-Frame mittels **Warping**
- Transformationen zu diesem Zeitpunkt:
 
$$T_{t_2, wld \leftarrow obj}^i \quad T_{t_2, img \leftarrow cam} \quad T_{t_2, cam \leftarrow wld}$$
- Ein Pixel  $P_A = (x, y, z)$  aus dem App.-Frame wird damit an die richtige Stelle im Server-Frame "gewarped":
 
$$P_S = T_{t_2, img \leftarrow cam} \cdot T_{t_2, cam \leftarrow wld} \cdot T_{t_2, wld \leftarrow obj}^i \cdot T_{t_1, obj \leftarrow wld}^i \cdot T_{t_1, wld \leftarrow cam} \cdot T_{t_1, cam \leftarrow img} \cdot P_A$$
- Diese Transf.-Matrix kann man zu Beginn eines Server-Frames für jedes Objekt vorberechnen

App.-Frame  $\rightarrow$  Server-Frame

$t_1$   $t_2$

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 57

Appl-Frame ( $t_1$ )  $\cdot P_A$

Camera ( $t_1$ )  $\xrightarrow{T_{t_1, cam \leftarrow img}}$

World ( $t_1$ )  $\xrightarrow{T_{t_1, wld \leftarrow cam}}$

Object  $\xrightarrow{T_{t_1, obj \leftarrow wld}^i}$

World ( $t_2$ )  $\xrightarrow{T_{t_2, wld \leftarrow obj}^i}$

Camera ( $t_2$ )  $\xrightarrow{T_{t_2, cam \leftarrow wld}}$

Warped Server-Frame ( $t_2$ )  $\xrightarrow{T_{t_2, img \leftarrow cam}}$

Warped Server-Frame ( $t_2$ )  $\cdot P_S$

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 58

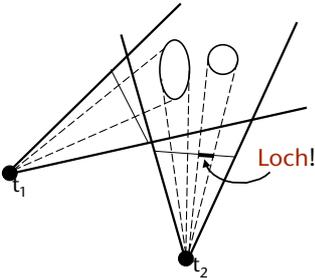
## Bewertung

- Implementierung des Warpings:
  - Im Vertex-Shader
    - Geht nicht im Fragment-Shader, da dort Output-Pos. festgelegt ist !
  - Waring Renderer lädt FBO des App.-Frames in Textur, dito alle Ti's
  - Rendere 1024x1024 viele GL\_POINTS (sog. Point-Splats)
- Vorteile:
  - Die Frames sind wesentlich aktueller (Kamera und Obj-Pos.!)
  - Server-Framerate ist unabhängig von Anzahl Polygone

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 59

## Probleme:

- Löcher im Server-Frame
- Server-Frames sehen unscharf aus (wg. Point-Splats)
- Wie groß sollen die Point-Splats sein? (kann man zwar abschätzen, aber ...)
- Wann die App.-Renderer-Framerate zu langsam, dann werden die Server-Frames doch zu schlecht
- Leere Stellen an den Rändern des Servers-Frames (evtl. View-Frustum im Client vergrößern)

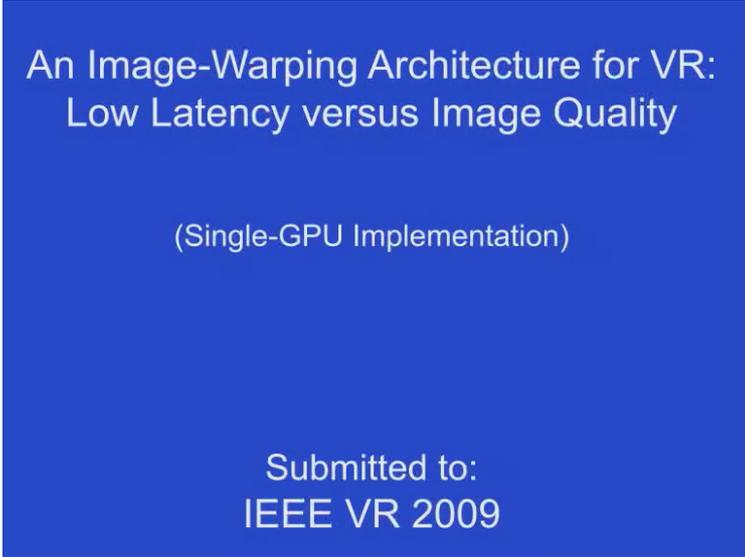


## Performance-Gewinn:

- 12 Mio Polygone, 800 x 600
- ca.(!) Faktor 20 schneller

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 60

Videos

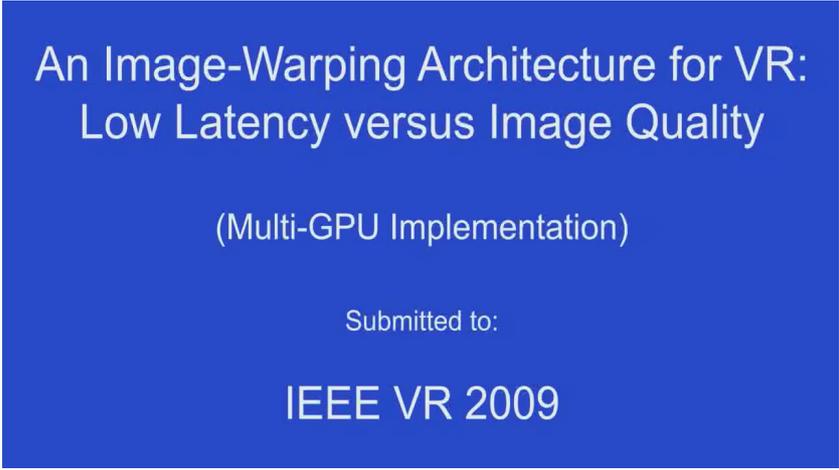


An Image-Warping Architecture for VR:  
Low Latency versus Image Quality

(Single-GPU Implementation)

Submitted to:  
IEEE VR 2009

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 61



An Image-Warping Architecture for VR:  
Low Latency versus Image Quality

(Multi-GPU Implementation)

Submitted to:  
IEEE VR 2009

G. Zachmann Virtuelle Realität und Simulation – WS 11/12 Real-time Rendering 62

