

# OpenGL-basiertes Software Occlusion Culling zur Beschleunigung des 3D-Renderings großer Datenmengen und komplexer Szenen

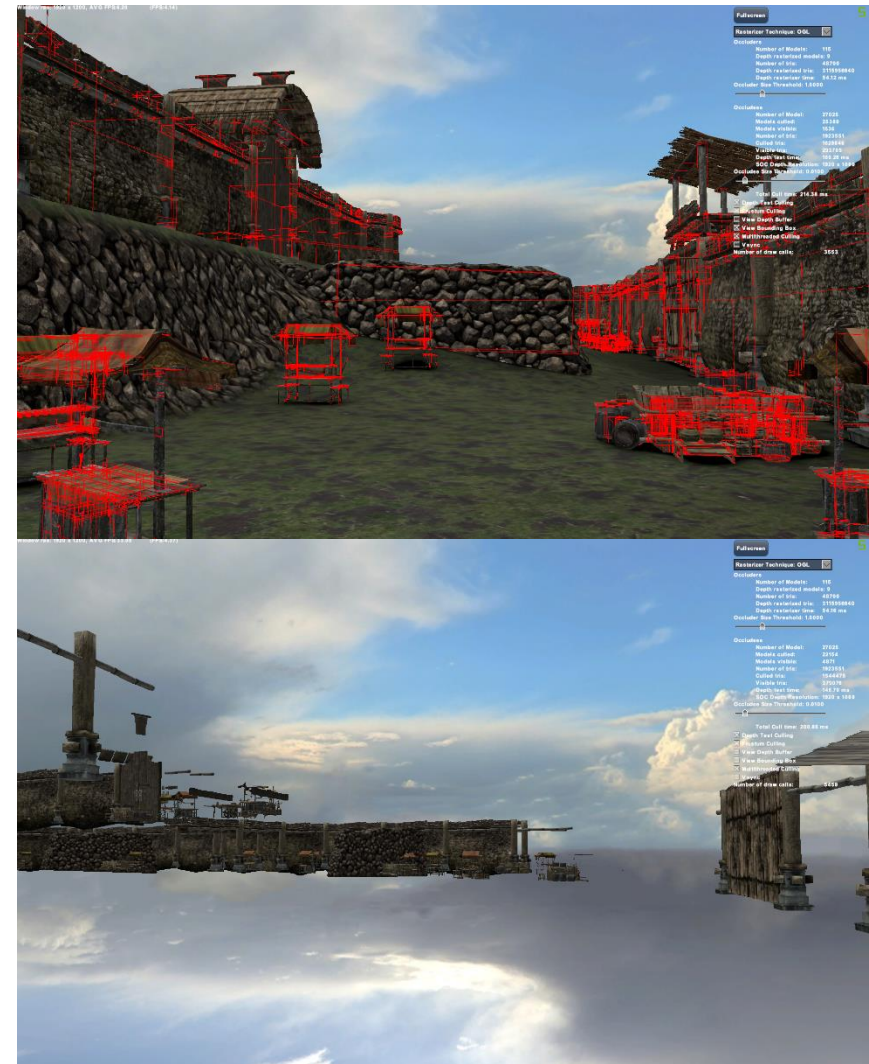
Projekt-INF

Dominik Sellenthin   Christian Stegmaier   Gariharan Kanthasamy

Betreuer: Dr. Guido Reina  
Michael Becher, M. Sc.

# Motivation

- Mehr dynamische Szenen
- ➔ Potentielle Sichtbarkeitsmengen schwierig zu verwenden
- Finde Objekte, die nicht gerendert werden sollen
- ➔ Occlusion Culling
- Ungenutzte CPU



## Related Work

- Intel's Software Occlusion Culling Framework [2][5]
  - Scalar
  - SSE - Streaming SIMD Extension
  - AVX2 – Advanced Vector Extensions
  - Masked AVX2
- Hierarchische Occlusion Map mit konservativen Occludern [6]
- Re-scheduling von Query und Rendering [1]
  - Abwechseln zwischen Query und Objekt-Rendering
  - Keine CPU-Stalls, GPU-Starvation
- Compute based triangle culling [7]
- Deferred+ [7]

# Occlusion Culling

Occludees



Occluder, große Objekte



Occludees (ohne Occluder), kleine Objekte





# Occlusion Culling

- Voraussetzung: Unterteilung Occluder/Occludee
  - $|\text{Occludee}| \gg |\text{Occluder}|$  (ca. 27000 vs. 100)
- Ablauf:
  1. Occluder:
    - i. Frustum Culling
    - ii. Tiefenpuffer-Rasterisierung (Mesa3D Gallium LLVMpipe)
  2. Occludee:
    - i. Frustum Culling
    - ii. Occlusion Queries
  3. Ergebnisweiterleitung an Renderroutine

# Occlusion Culling

- Voraussetzung: Unterteilung Occluder/Occludee
- Ablauf:
  1. **Occluder:**
    - i. **Frustum Culling**
    - ii. Tiefenpuffer-Rasterisierung
  2. Occludee:
    - i. Frustum Culling
    - ii. Occlusion Queries
  3. Ergebnisweiterleitung an Renderroutine



# Occlusion Culling

- Voraussetzung: Unterteilung Occluder/Occludee
- Ablauf:
  1. **Occluder:**
    - i. Frustum Culling
    - ii. **Tiefenpuffer-Rasterisierung**
  2. Occludee:
    - i. Frustum Culling
    - ii. Occlusion Queries
  3. Ergebnisweiterleitung an Renderroutine



# Occlusion Culling

- Voraussetzung: Unterteilung Occluder/Occludee
- Ablauf:
  1. Occluder:
    - i. Frustum Culling
    - ii. Tiefenpuffer-Rasterisierung
  2. Occludee:
    - i. **Frustum Culling**
    - ii. Occlusion Queries
  3. Ergebnisweiterleitung an Renderroutine



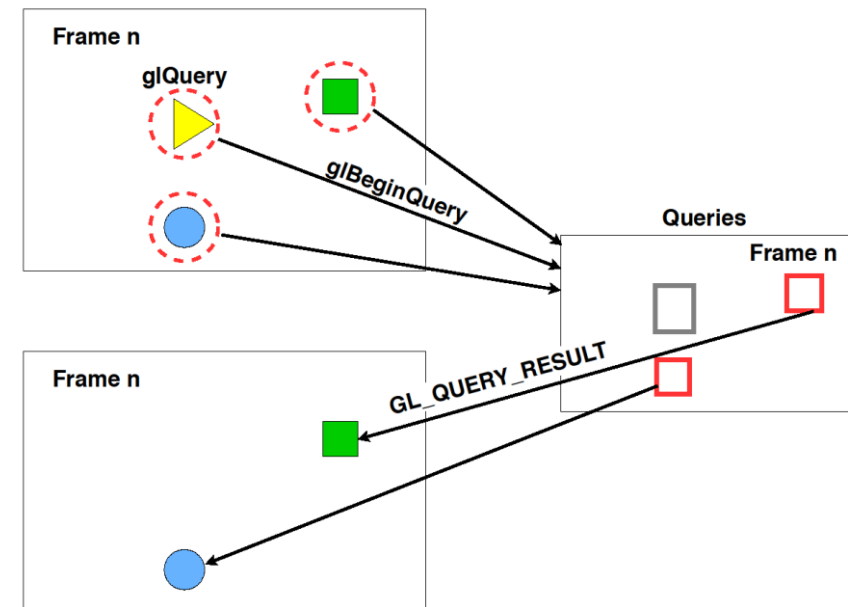


# Occlusion Culling

- Voraussetzung: Unterteilung Occluder/Occludee
- Ablauf:
  1. Occluder:
    - i. Frustum Culling
    - ii. Tiefenpuffer-Rasterisierung
  2. Occludee (inkl. Occluder):
    - i. Frustum Culling
    - ii. **Occlusion Queries**
  3. Ergebnisweiterleitung  
an Renderroutine

# Occlusion Queries

- Pro Occludee eine Occlusion Query (glQuery)
- Ablauf:
  - Starte alle Queries (glBeginQuery, glDrawArrays, glEndQuery)
  - Warte auf Ergebnis (GL\_QUERY\_RESULT\_AVAILABLE)
    - Warte auf letzte gestartete Query
    - Wartezeiten!
  - Hole Ergebnisse aller Queries



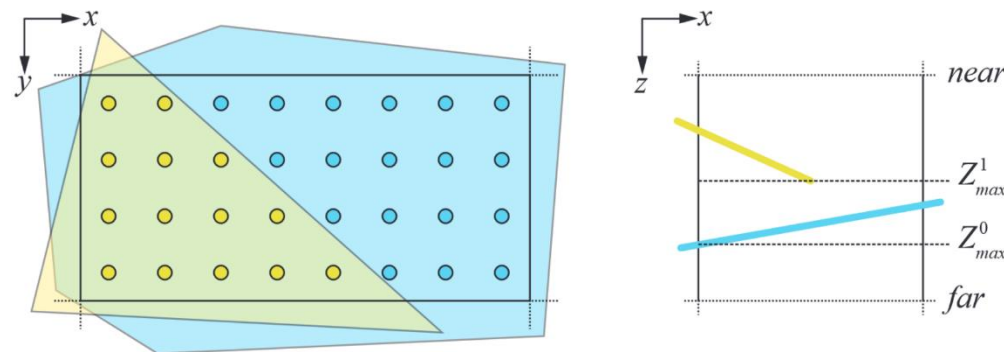
# Occlusion Culling

- Voraussetzung: Unterteilung Occluder/Occludee
- Ablauf:
  1. Occluder:
    - i. Frustum Culling
    - ii. Tiefenpuffer-Rasterisierung
  2. Occludee:
    - i. Frustum Culling
    - ii. Occlusion Queries
  3. Ergebnisweiterleitung an Renderroutine




# Masked AVX2 Occlusion Culling

- Arbeitet mit abgewandeltem hierarchischem Z-Buffer (HiZ)
- Arbeitsebene und Referenzebene
- Explizit für CPU optimiert
  - 8 SIMD-lanes formen 32x8 Kachel
  - SIMD-lane als 8x4 Kachel
- Versus generischer Software-Treiber (Mesa3D)



Masked Software Occlusion Culling [1]

## Ergebnisse: SOC-Framework

- Zur Messung aller Ergebnisse dient das Intel SOC-Framework
  - Testszene:
    - 115 Occluder                   => ~48.700 Dreiecke
    - 27.025 Occludees           => ~1.923.000 Dreiecke
  - Tiefenpuffer Auflösung: 1920x1080
  - Kamerafahrt über 100 Frames
- 

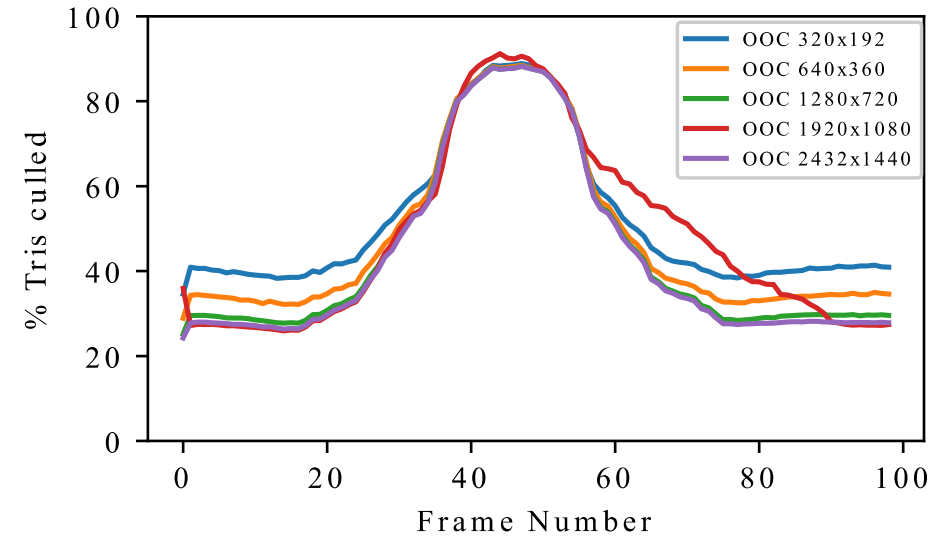
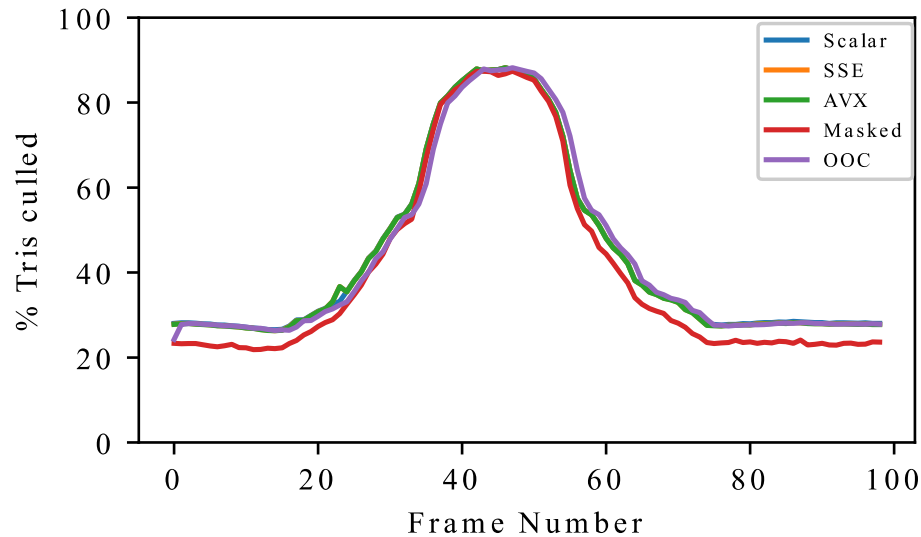




# Ergebnisse: Kamerafahrt

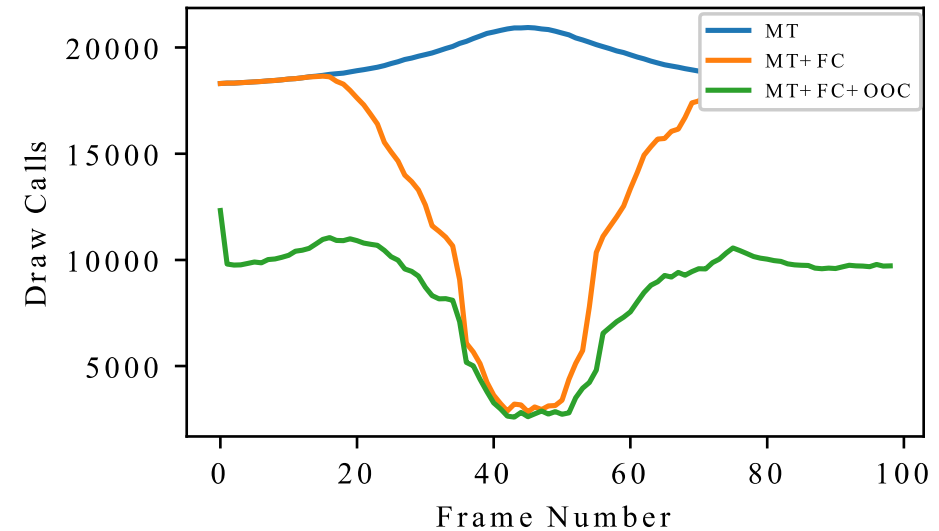
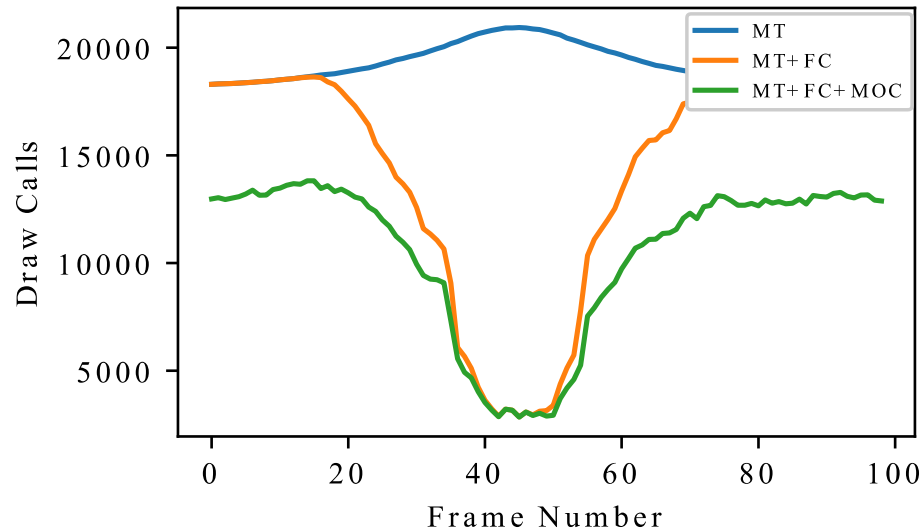


# Ergebnisse: Qualität + unterschiedliche TB Auflösung



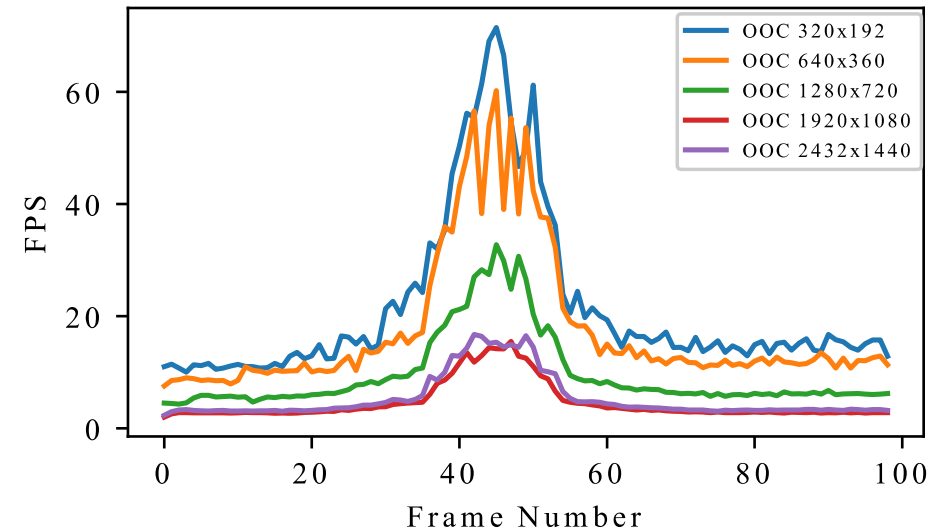
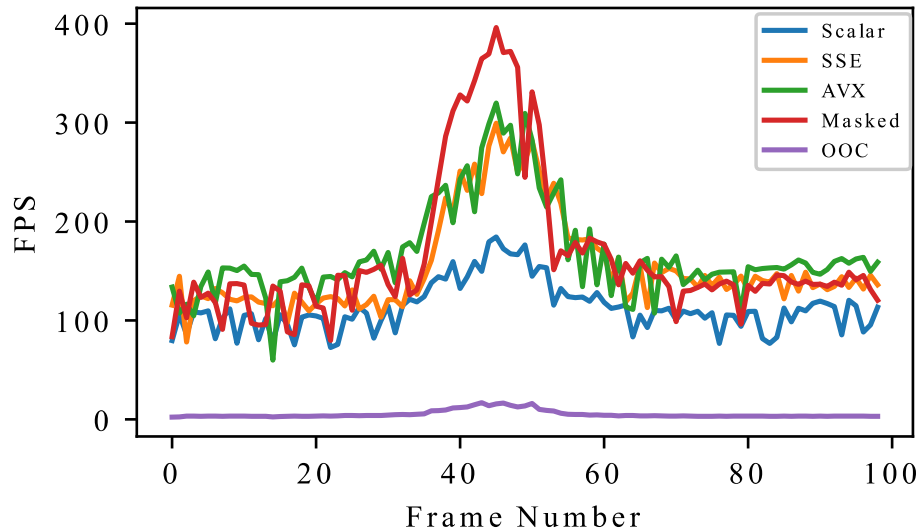
	OpenGL Occlusion Culling				
	320x192	640x360	1280x720	1920x1080	2432x1440
Tris culled (%)	51.75	47.59	44.65	48.18	43.77
Draw Calls	3551	5696	7644	7896	8423
Models Culled	23819.61	21970.67	20320.78	20135.84	19660.16
Models Visible	3205.39	5054.33	6704.22	6889.16	7364.84

# Ergebnisse: Qualität MOC vs. OOC



			MOC	OOC
	MT	MT+Frustum	MT + Frustum + OC	
Tris culled (%)	11.13	11.13	40.41	43.77
Draw Calls	19219	14318	10528	8423
Models Culled	10180.34	14532.39	17903.78	19660.16
Models Visible	16844.66	12492.61	9121.22	7364.84

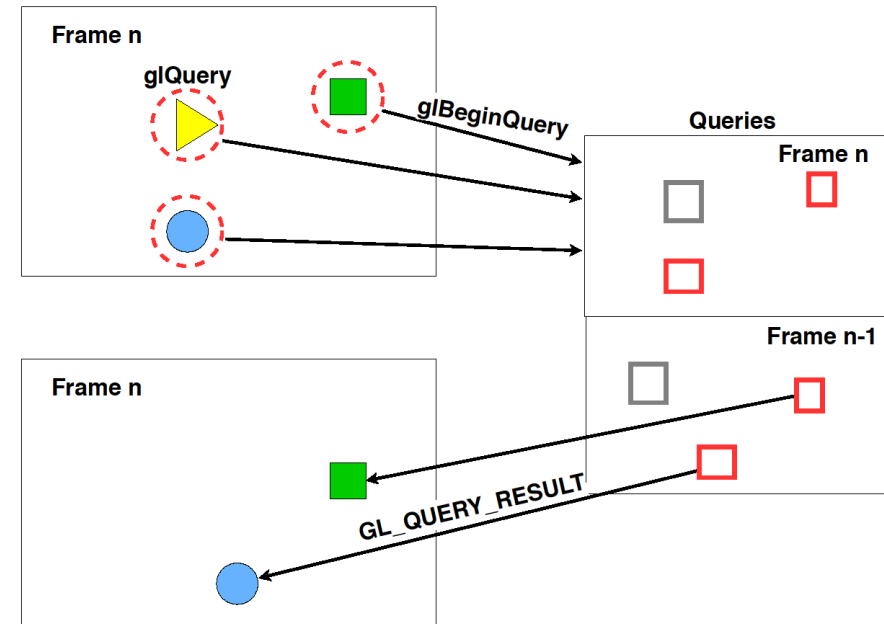
# Ergebnisse: Performance + unterschiedliche TB Auflösung



	OpenGL Occlusion Culling				
	320x192	640x360	1280x720	1920x1080	2432x1440
FPS	21.48	17.53	9.55	4.58	5.31
DepthTest time (ms)	50.4	63.22	122.21	265.67	228.85
Rasterize time (ms)	6.64	6.85	6.87	7.07	8.02

# Probleme mit Occlusion Queries

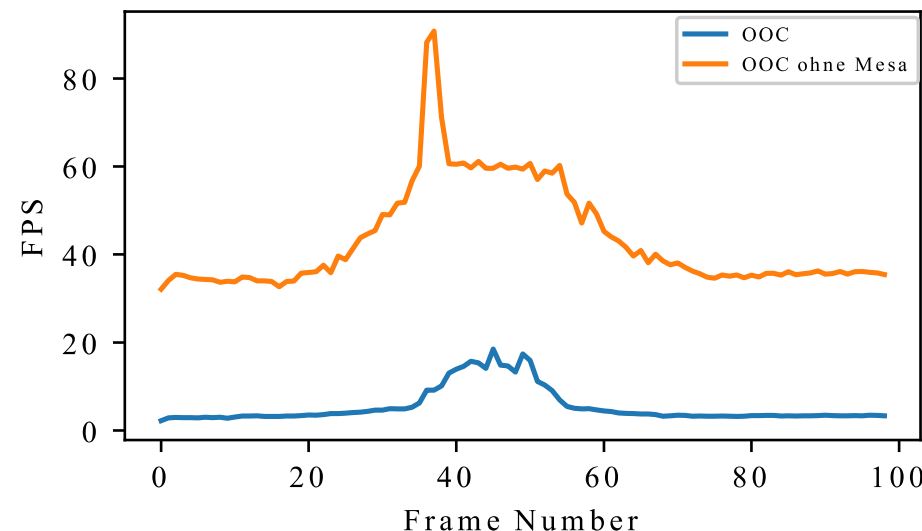
- Queries brauchen lange
- Ansatz: Doppelte Anzahl Queries
  - Einmal für aktuellen Frame
  - Einmal für vorherigen Frame
- Hole Ergebnisse vorheriger Frames ohne zu warten
  - keine Warteroutine
  - keine Wartezeiten
- „Hoffen“, dass Ergebnisse da sind





## Fazit

- Mesa 3D ermöglicht eine zur GPU parallele OC Methode auf der CPU
- Qualität von OOC ist gut
- Performance ist sehr schlecht
  - Hauptgrund: Occlusion Queries
    - Keine Performanceverbesserung durch doppeltes Set
    - Mesa-Implementierung nebenläufig?
  - Geringe CPU-Auslastung

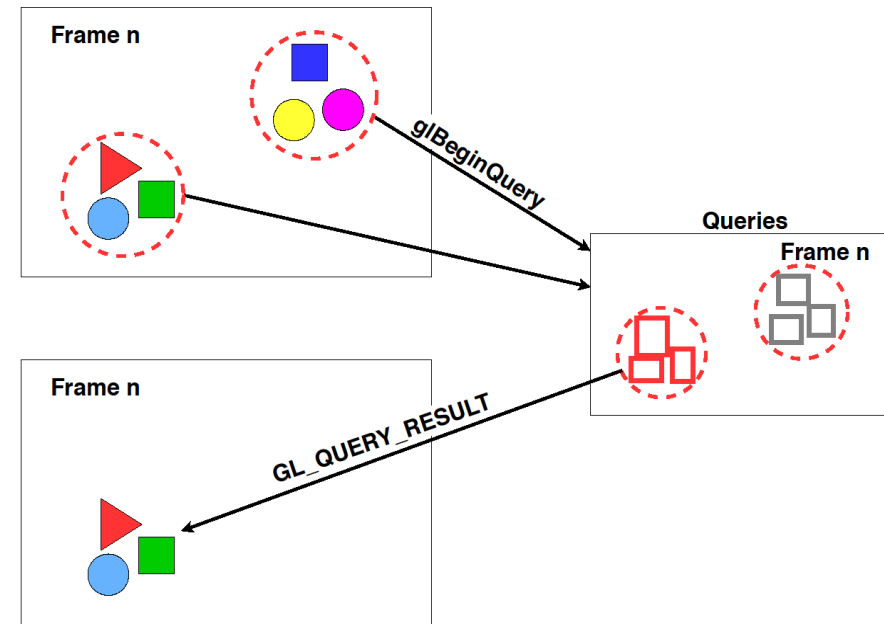


## Future Work

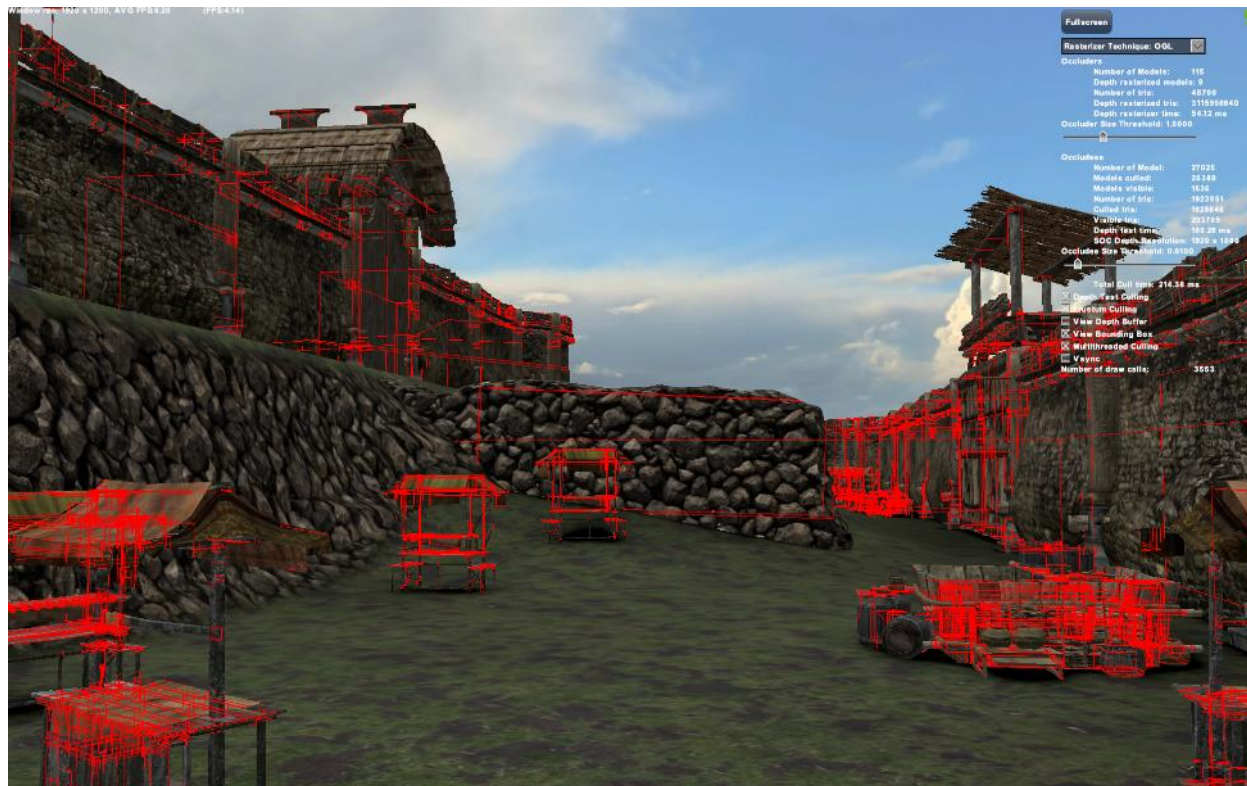
- Anderer Mesa3D Treiber (OpenSWR)
- Hierarchischer Tiefenpuffer
  - Pre-processing mit geringer Auflösung
  - Zweiter Durchlauf mit voller Auflösung
- Occludee-Pakete

## Future Work: Occludee Pakete

- Bisher: Je Objekt eine Occlusion Query
  - schlechte Performance
- Ansatz: Eine Occlusion Query für mehr
  - Anzahl der Queries wird verringert
- Ist ein Occludee sichtbar, wird ganzes Paket gerendert
- Sinnvolle Pakete wichtig
  - Möglichst geringe bounding sphere



# Fragen?



# Literatur

- [1] Bittner, J., Wimmer, M., Piringer, H., & Purgathofer, W. (2004, September). Coherent hierarchical culling: Hardware occlusion queries made useful. In *Computer Graphics Forum* (Vol. 23, No. 3, pp. 615-624). Oxford, UK and Boston, USA: Blackwell Publishing, Inc.
- [2] C. Chandrasekaran, D. McNabb, K. Kiefer, M. Fauconneau, and F. Giesen. Software occlusion culling. <https://software.intel.com/en-us/articles/software-occlusion-culling>, 2013-2016.
- [3] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, 2003.
- [4] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 231–238. ACM, 1993.
- [5] J. Hasselgren, M. Andersson, and T. Akenine-Möller. Masked Software Occlusion Culling. In U. Assarsson and W. Hunt, editors, *Eurographics/ACM SIGGRAPH Symposium on High Performance Graphics*. The Eurographics Association, 2016.
- [6] Leone, M. N., Barbagallo, L. R., Banquero, M. M., Agromayor, D., & Bursztyn, A. P. (2012). Implementing software occlusion culling for real-time applications. In *XVIII Congreso Argentino de Ciencias de la Computación*.
- [7] Engel, W. (2017). *GPU Zen: Advanced Rendering Techniques*. Bowker Identifier Services.