# Raytracing und Szenengraphen

Zwischenvortrag zur Diplomarbeit von

Björn Schmidt

30. August 2006

J.W. Goethe Universität Frankfurt am Main

# Inhalt

- Ziel der Arbeit
- Raytracing durch die CPU
- Raytracing durch die GPU
- Die Wahl der Beschleunigungsdatenstruktur
- Integration eines Raytracers in OGRE 3D
- Aktueller Stand
- Ausblick

# Ziel der Arbeit

- Integration eines Echtzeit-Raytracers in eine bestehende Szenengraphen-API
- Raytracing statischer und dynamischer Szenen

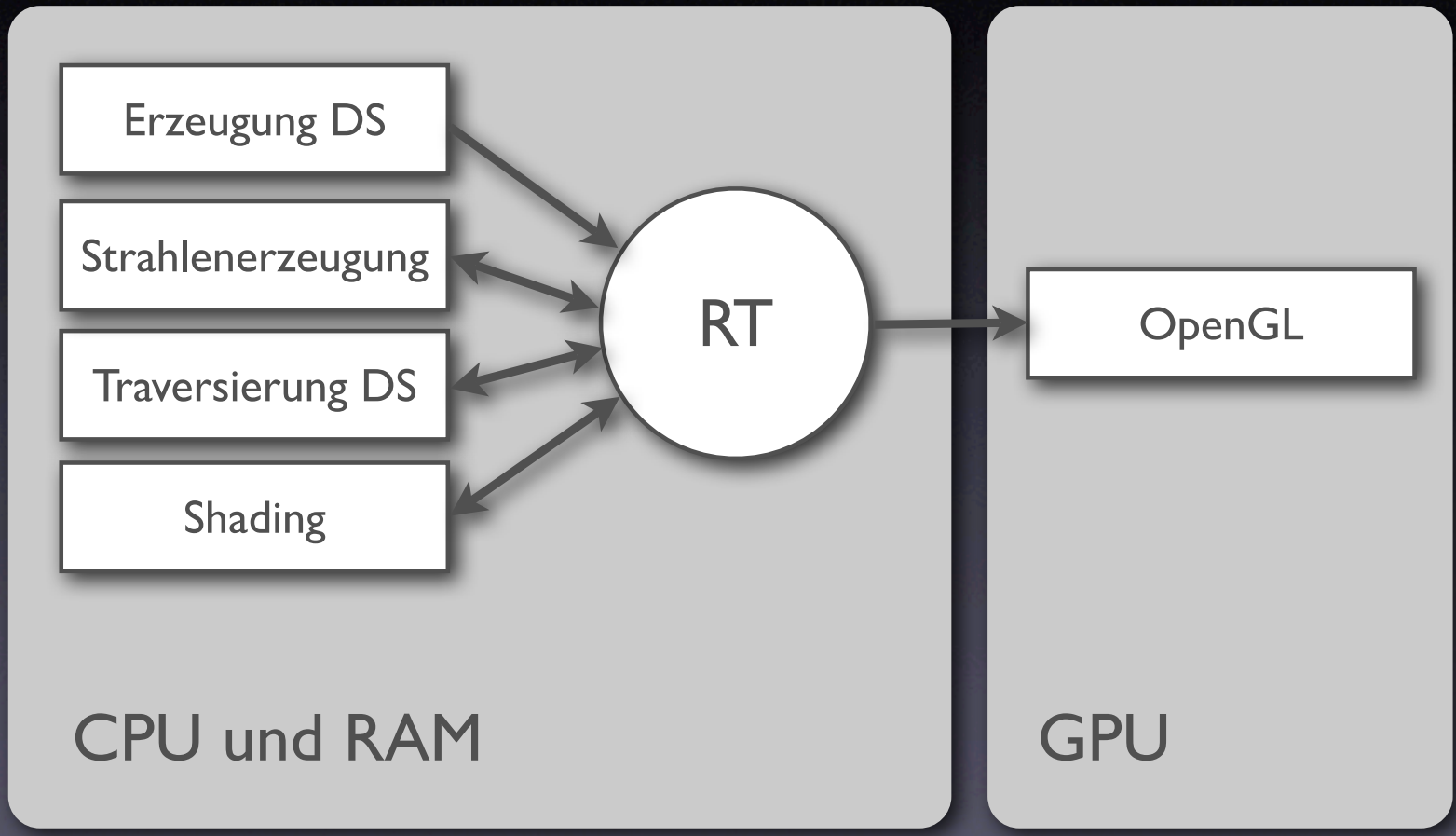
# Welche Probleme müssen gelöst werden?

- Verwendet man einen CPU- oder einen GPU-basierten Raytracer?
- Welche Beschleunigungsdatenstruktur soll verwendet werden?
- Welche Szenengraph-API eignet sich am besten für die Integration eines Raytracers?
- Wie soll der Raytracer in den Szenengraph integriert werden?

# Raytracing durch die CPU

- Raytracing ist ein sehr "rechenfreudiges" Verfahren
- Toni Nemec hat in seiner Diplomarbeit [TN05] einen CPU-basierten Online-Raytracer entwickelt
  - Erreicht 5-10 fps bei statischen Szenen durch:
    - geschickte Wahl der Beschleunigungsdatenstruktur
    - Verwendung von sog. "Pixel Selected Ray Tracing"

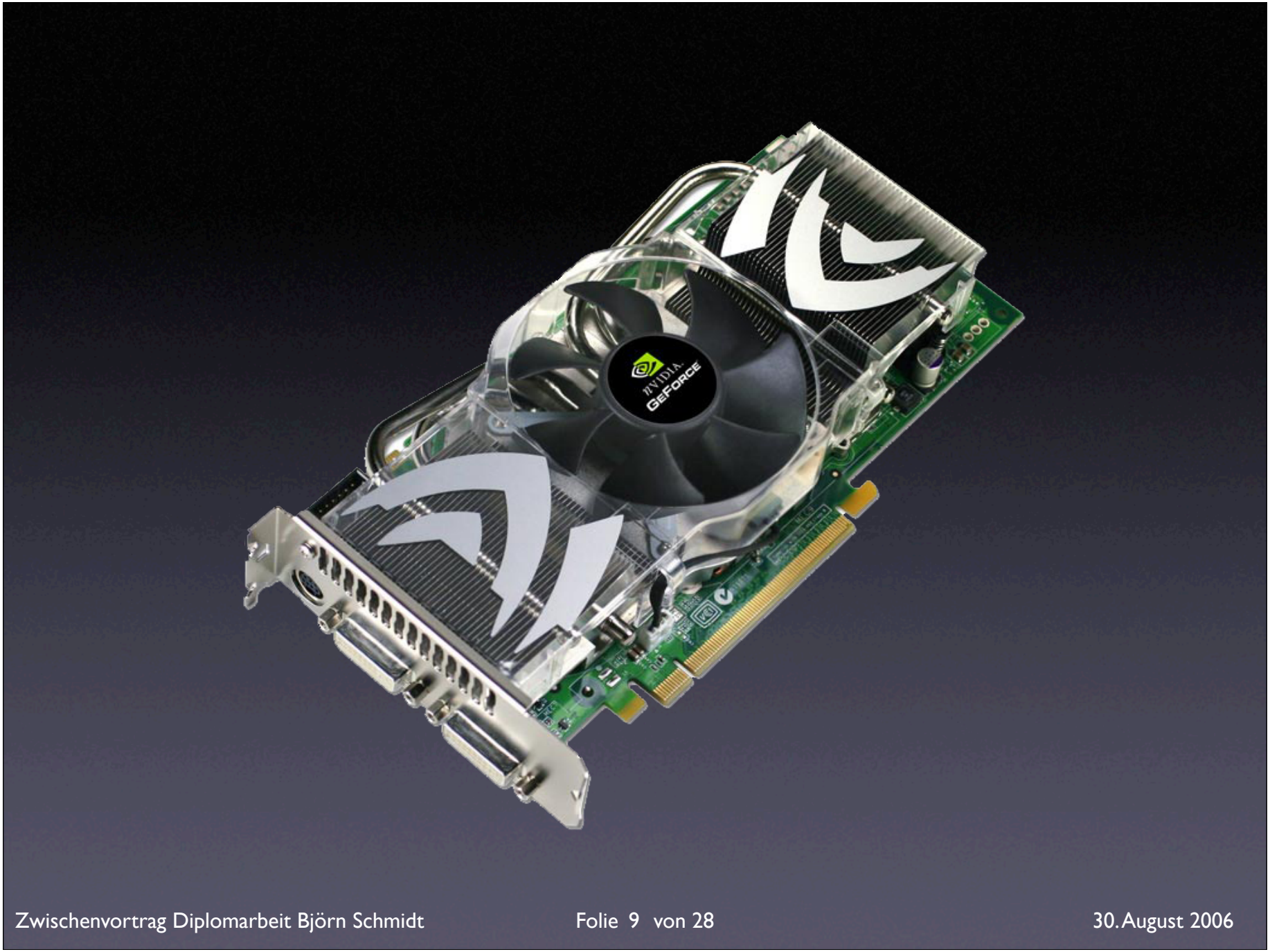
# Raytracing durch die CPU Details

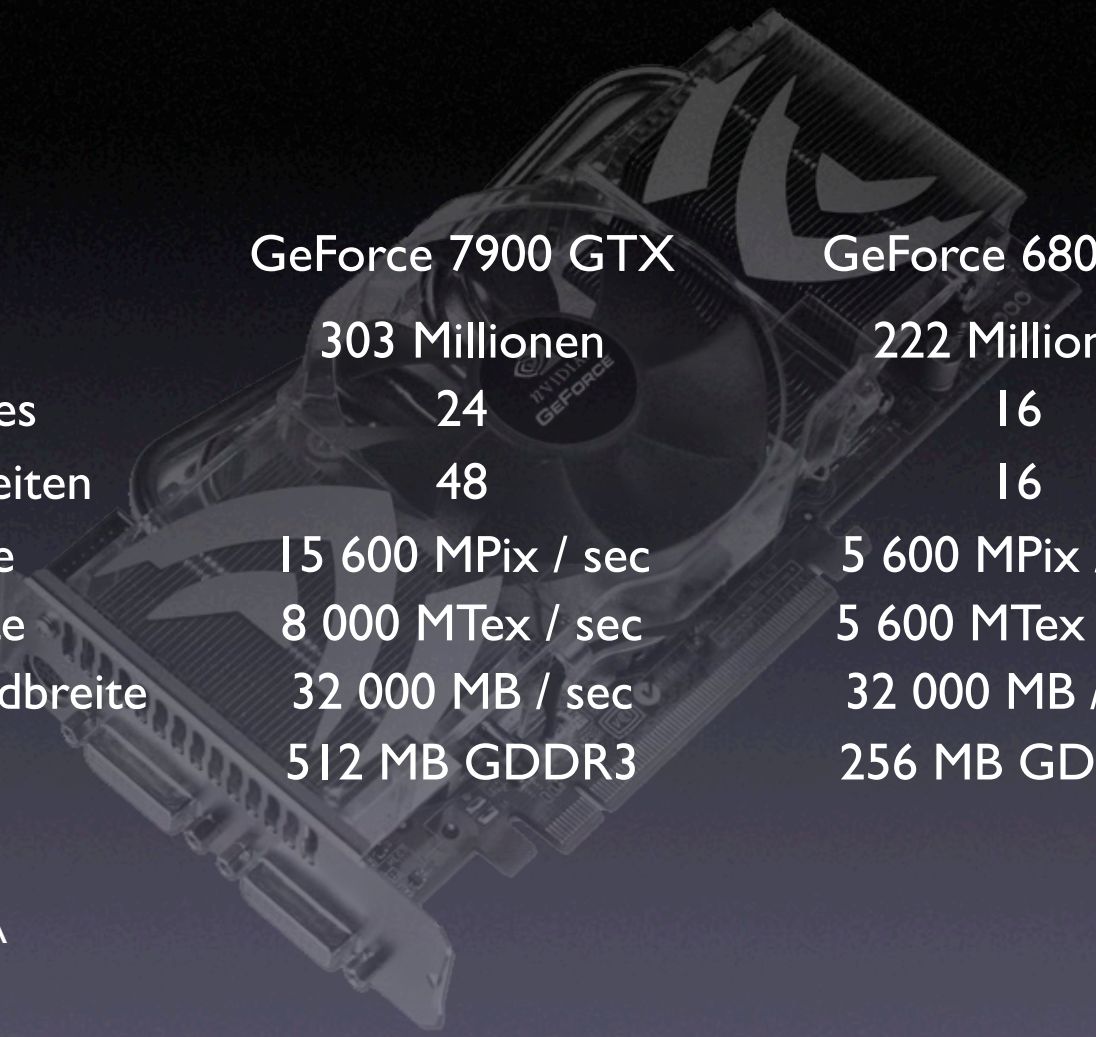


# Raytracing durch die CPU

- Raytracing ist ein massiv paralleles Verfahren.
- Gibt es eine Möglichkeit, diese Parallelität bei der Berechnungen auszunutzen, ohne Rechner-Cluster zu verwenden?

# Die Antwort ist ja!


NVIDIA
GeForce



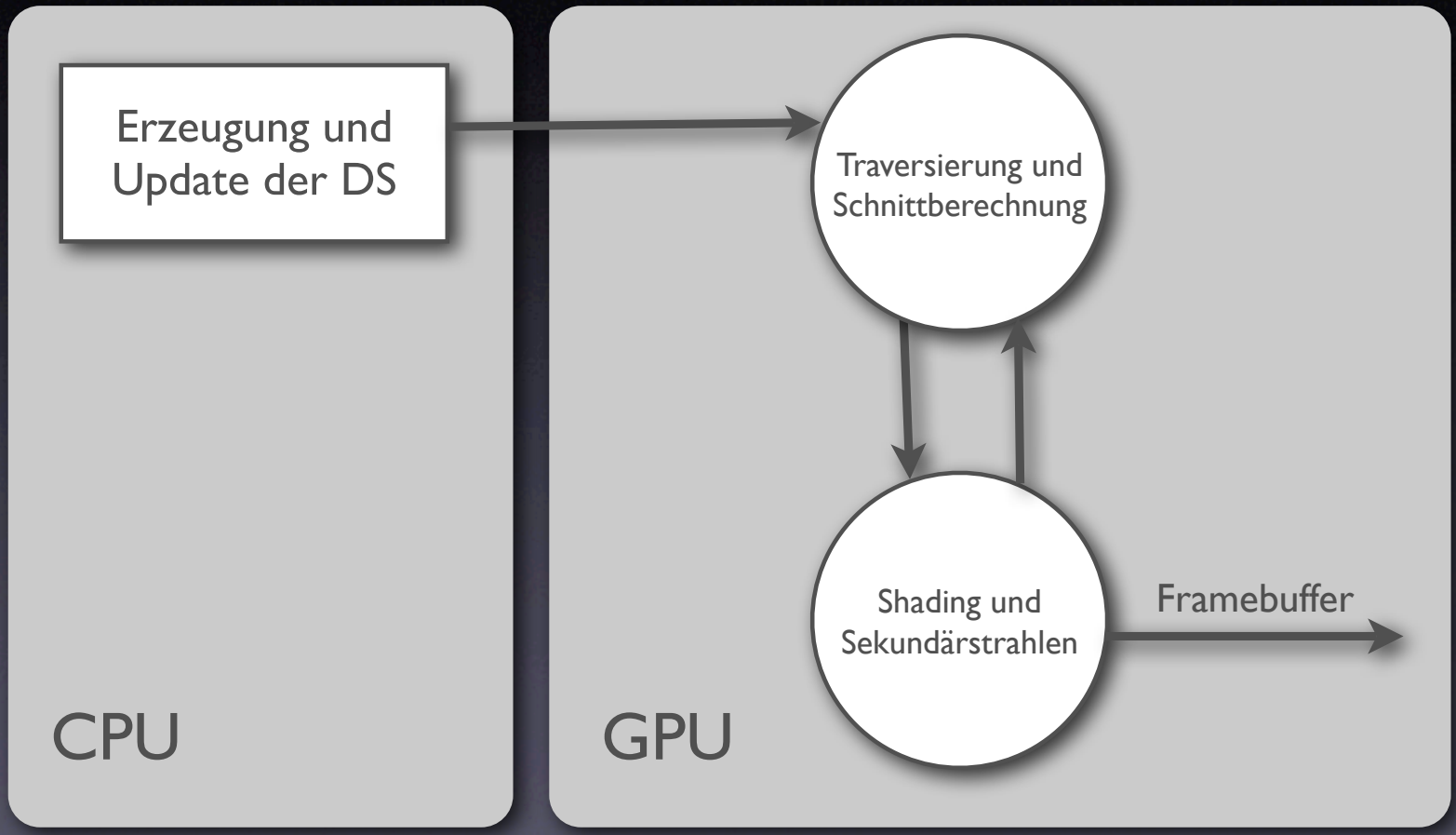
| | GeForce 7900 GTX | GeForce 6800 GT |
|---|---|---|
| Transistoren | 303 Millionen | 222 Millionen |
| Pixel Pipelines | 24 | 16 |
| Shader-Einheiten | 48 | 16 |
| Pixel-Füllrate | 15 600 MPix / sec | 5 600 MPix / sec |
| Texel-Füllrate | 8 000 MTex / sec | 5 600 MTex / sec |
| Speicherbandbreite | 32 000 MB / sec | 32 000 MB / sec |
| Speicher | 512 MB GDDR3 | 256 MB GDDR3 |

Quelle: NVIDIA

# Raytracing durch die GPU

- Verwendung der GPU als Streaming Prozessor
  - Eingaben und Ausgaben der Graphik-Pipeline werden als allgemeine Daten angesehen.
  - Die Graphik-Pipeline kann mittels Vertex- und Fragment-Shadern umkonfiguriert werden.
  - “General-Purpuse Computation Using Graphics-Hardware.“ (siehe http://www.gpgpu.org)

# Raytracing durch die GPU



# Die Entscheidung: GPU-basiertes Raytracing

- Nutzt Parallelität optimal aus
- Weitere Leistungssteigerungen bei GPUs zu erwarten

# Welche Probleme müssen gelöst werden?

- Verwendet man einen CPU- oder einen GPU-basierten Raytracer? ✓
- Welche Beschleunigungsdatenstruktur soll verwendet werden?
- Welche Szenengraph-API eignet sich am besten für die Integration eines Raytracers?
- Wie soll der Raytracer in den Szenengraph integriert werden?

# Die Wahl der Beschleunigungsdatenstruktur

- Die meisten Verfahren zur Traversierung der Datenstruktur sind rekursiv - Rekursivität lässt sich auf der GPU nicht umsetzen.
- Thrane und Simonsen haben in [TS05] die Datenstrukturen Reguläres Gitter, KD-Baum und Hüllkörperhierarchie auf ihre Performance beim GPU-basierten Raytracing evaluiert.
- In [TS05] stellt sich die Hüllkörperhierarchie BVH klar als Sieger heraus.

# Die Wahl der Beschleunigungsdatenstruktur

- Verwendung von BVH als Beschleunigungsdatenstruktur.
- Verfahren zur Erzeugung angelehnt an [RR03] und mit Optimierungen versehen.
- Verfahren zur Traversierung angelehnt an [TS05], funktional erweitert und mit Optimierungen versehen.

# Die Wahl der Beschleunigungsdatenstruktur

- BVH hat neben ihrer Performance noch einen weiteren Vorteil
- Statische Objekte werden später im Szenengraphen in eine BVH umgewandelt und in die BVH der Szene eingefügt
  - Vorteil: Es reicht Teile der Datenstruktur zu aktualisieren, wenn sich an der Transformation eines Objekts etwas geändert hat.

# Welche Probleme müssen gelöst werden?

- Verwendet man einen CPU- oder einen GPU-basierten Raytracer? ✓
- Welche Beschleunigungsdatenstruktur soll verwendet werden? ✓
- Welche Szenengraph-API eignet sich am besten für die Integration eines Raytracers?
- Wie soll der Raytracer in den Szenengraph integriert werden?

# Wahl der Szenengraphen-API Anforderungen

- Open Source
- Plug-In basiert
- Unterstützung der Shader-Sprache Cg (“C for graphics”)
- Unterstützung Shader Model 3.0
- Unterstürzung von DirectX 9c und OpenGL 2.0
- Relevanz für öffentliche Projekte

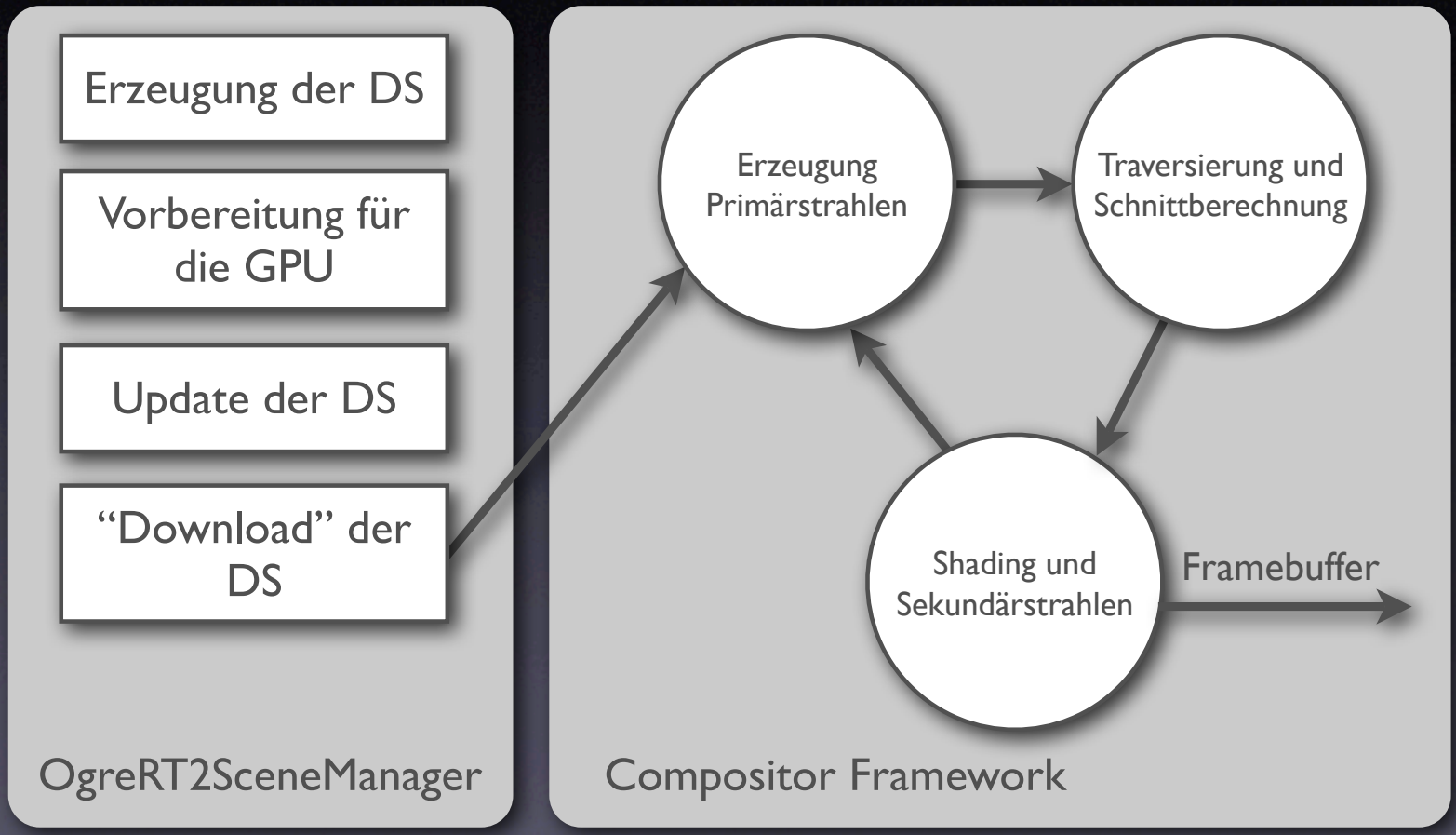
# Wahl der Szenengraphen-API OGRE 3D

- OGRE = Object-Oriented Graphics Rendering Engine
- Erfüllt alle Anforderungen
- Bietet zwei Konzepte, die sich hervorragend für die Integration des Raytracers eignen:
  - SceneManager
  - Compositors

# Welche Probleme müssen gelöst werden?

- Verwendet man einen CPU- oder einen GPU-basierten Raytracer?
- Welche Beschleunigungsdatenstruktur soll verwendet werden?
- Welche Szenengraph-API eignet sich am besten für die Integration eines Raytracers?
- Wie soll der Raytracer in den Szenengraph integriert werden?

# Integration eines Ray Tracers in OGRE 3D



# Welche Probleme müssen gelöst werden?

- Verwendet man einen CPU- oder einen GPU-basierten Raytracer? ✓
- Welche Beschleunigungsdatenstruktur soll verwendet werden? ✓
- Welche Szenengraph-API eignet sich am besten für die Integration eines Raytracers? ✓
- Wie soll der Raytracer in den Szenengraph integriert werden? ✓

# Aktueller Stand

- Erzeugung der Beschleunigungsdatenstruktur komplett.
- Generierung der Primärstrahlen durch die GPU funktioniert.
- Traversierung der Datenstruktur und Shading funktioniert für Basisprimitive in der ersten Iteration.
- Die Umrisse der ersten Objekte werden somit bereits korrekt dargestellt.

# Ausblick

- Erstellung der BVH einer beliebigen Geometrie
- Raytracing dynamischer Szenen
- Reflektion und Refraktion
- Dynamisches Anpassen der Rekursionsstufe

# Quellen


- [TN05] Toni Nemec: *Implementierung eines Echtzeit Raytracers*, Diplomarbeit 2005
- [PU04] Timothy Purcell: *Raytracing on a stream processor*, Phd. -Arbeit 2004
- [TS05] Nils Thrane und Lars Ole Simonsen: *A comparison Of Acceleration Struktures for GPU Assisted Ray Tracing*, Diplomarbeit 2005
- [RR03] P. Shirley und R.K. Morley: *Realistic Ray Tracing*, 2003

# Fragen?

# Vielen Dank für die Aufmerksamkeit!