FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Diplomarbeitspräsentation

Real-time Ray Tracing on the GPU

Ray Tracing using CUDA and kD-Trees

Masterstudium:
Visual Computing

Günther Voglsam

Technische Universität Wien
Institut für Computergraphik und Algorithmen
Arbeitsbereich Computergraphik
Betreuung: DI DI Dr. Michael Wimmer
Mitwirkung: DI Dr. Robert Tobler

Motivation

This thesis is part of the **HILITE**-project at VRVis.

Overall goal of the project:
Dynamic, interactive, realistic real-time lighting simulation for complex architectural environments.

One of the Requirements:
Correct rendering of curved reflections using a GPU-accelerated ray tracer
=> Topic of this thesis: CUDA-Ray Tracer

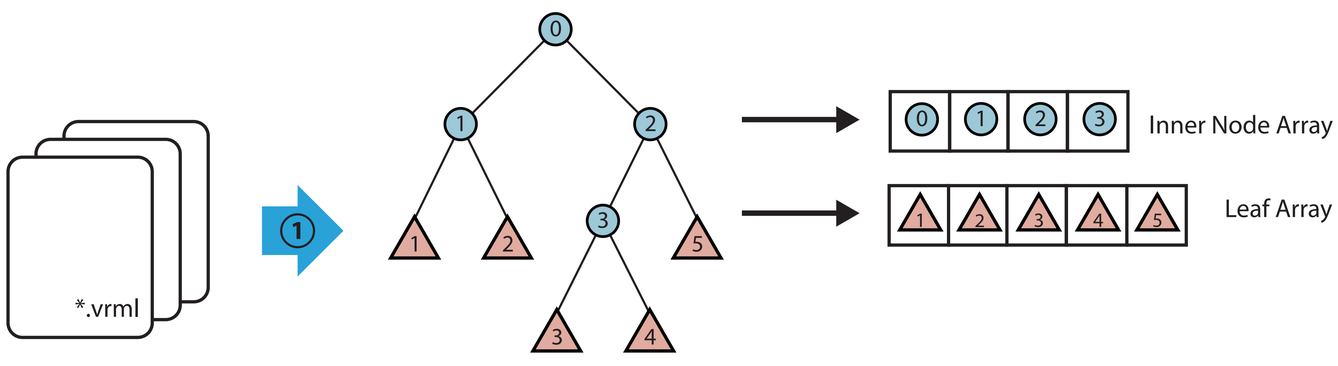
Prerequisite:
CUDA-access from VRVis-internal rendering engine Aardvark (C#) used for HILITE
=> Develop CUDA-Library first

CUDA-Library for Aardvark

Provides access to **CUDA from C# for the Aardvark rendering engine** to gain GPU-processing power for research projects at VRVis.

- High-level object orientation
- Easy to use
- Full Aardvark integration
- Encapsulates **CUDA Driver API**

Features:
- CUDA-**Context** and -**Device management**
- GPU **memory management**
- Easy **kernel calls** from within C#
- Provides ready-to-use **data types**
- **Graphics resource sharing**
- Support for CUDA-Streams, -Timer/-Events, ...



CUDA Ray Tracer

- Ray tracing **completely on the GPU**
- **Real-time to interactive performance**, depending on scene
- **Stack-based iterative** kD-Tree traversal
- Uses two variants of **kD-Trees** as acceleration data structures: *Object-kD-Trees* and *Über-kD-Tree* (see below)
- **Uses CUDA-library** for managing CUDA-resources

Algorithm:

1. Load scene, create kD-Trees on CPU and convert it to a format suitable for GPU: Inner Node Array + LeafArray.

2. In parallel, start a single thread for each pixel on the GPU.

   Each thread traces one primary ray and its secondary rays.

3. The generated image is directly rendered into a DirectX-texture.

4. DirectX renders the texture as full-screen quad.

Object-kD-Tree (OKD):
- One kD-Tree per geometric object
- Rendering processes lists of OKDs

Über-kD-Tree (ÜKD):
- KD-Tree of OKDs
- Rendering traverses ÜKD first, then OKD
- Useful for interactively editing scene and scenes with lots of objects

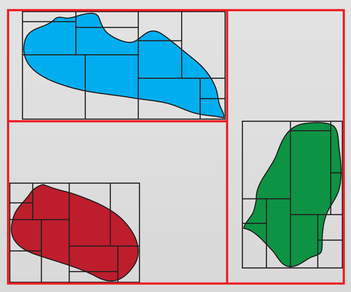


Figure: OKD = black lines
ÜKD = red lines

Results

Rendering performances:
Standford Bunny (image on the left, 69.5k triangles) rendered on an Nvidia GTX480 with:

	Primary rays only:	With reflection (two bounces):
640x480	81 FPS / 24.9 Mrays/sec *)	21 FPS / 1.6 Mrays/sec *)
1024x768	39 FPS / 30.2 Mrays/sec	15 FPS / 2.0 Mrays/sec
1600x1200	17 FPS / 32.0 Mrays/sec	5 FPS / 2.5 Mrays/sec

	With shadow (one point light):	With shadow and reflection:
640x480	48 FPS / 25.6 Mrays/sec *)	15 FPS / 13.4 Mrays/sec (avg)
1024x768	25 FPS / 42.5 Mrays/sec	7 FPS / 16.6 Mrays/sec (avg)
1600x1200	11 FPS / 44.3 Mrays/sec	3 FPS / 19.4 Mrays/sec (avg)

*) Mrays/sec with respect to primary/shadow/reflection rays only

Kontakt: gvoglsam@vrvis.at, tobler@vrvis.at, wimmer@cg.tuwien.ac.at