When it makes sense to use uniform grids for ray tracing

M. Hapala, O. Karlík, V. Havran

Czech Technical University in Prague
Faculty of Electrical Engineering
Intro

- Ray tracing/casting
  - Basic visibility operation
  - Finding closest intersections between rays and objects in a scene

- Intersection search complexity
  - Naïve in $O(N)$
  - Acceleration data structure as fast as $O(\log N)$

- Applications almost always use one data structure
Intro 2

- Uniform grid
  - Build in $O(N)$
  - Traversal in $O(\sqrt[3]{N})$

- Hierarchical data structures (HDS)
  - Build in $O(N \cdot \log N)$
  - Traversal in $O(\log N)$

- Hidden constants for HDS traversal
  - “Quality” of the structure, how it can adapt to the scene
  - Implementation and hardware performance
Idea

- Take the best from both worlds
- Which is more efficient for a particular scene?
- Change from grid to HDS when advantageous
- Need rough number of rays to be computed

![Graph showing comparison between HDS and Uniform Grid]

Break-even point
Calibration

- Executed once
- Set of representative scenes
- Build a HDS and measure
  - Time to build the data structure
  - Time to compute a single ray
- What do we need these for?
  - Build time constant
  - Shooting a single ray constant
Calibration

- Executed once
- Set of representative scenes
- Build a HDS and measure
  - Time to build the data structure
  - Time to compute a single ray
- What do we need these for?
  - Build time constant
  - Shooting a single ray constant

\[
\frac{1}{s} \sum_{i=1}^{s} \frac{T_B^H(i)}{N(i) \cdot \log_2 N(i)}
\]

Average over all scenes

\[
\frac{1}{s} \sum_{i=1}^{s} \frac{T_R^H(i)}{M(i) \cdot \log_2 N(i)}
\]

Number of rays

Shooting complexity

Shooting all rays time
Application

- Build a uniform grid
- Compute a small set of representative rays
- Estimate HDS performance
  - Build time
    \[ N \cdot \log_2 N \cdot \frac{1}{s} \sum_{i=1}^{s} \frac{T_B^H(i)}{N(i) \cdot \log_2 N(i)} \]
  - Shooting a single ray time
    \[ \log_2 N \cdot \frac{1}{s} \sum_{i=1}^{s} \frac{T_R^H(i)}{M(i) \cdot \log_2 N(i)} \]
- Compute break-even point
- Decide if we need to build and use HDS
- Shoot all the rays

Implementation/Hardware constants
Results

- Tested on 28 scenes
  - Primitive count 500 – 1.6M
  - Various levels of uniformity
  - X scenes for calibration
  - 28-X scenes estimated

- 2M rays
  - Randomly generated
  - Uniform distribution

- Estimate accuracy

- Speedup
Break-even point estimate accuracy

- Relative estimate error [%] = $100 \cdot \frac{R_{est} - R_C}{R_C}$
- Red - sum of relative errors
- Blue - sum of absolute values of relative errors
Speedup

- Median time per ray

![Graph showing speedup of Kd-tree and uniform grid with respect to the number of rays. The graph indicates a decrease in time with an increase in the number of rays.](image-url)
Speedup

- Median time per ray

![Graph showing speedup over number of rays]
Speedup

- Median hybrid algorithm speedup versus using only one data structure
Conclusion

- Choose a data structure based on the number of rays
- Minimal overhead
- High speedup
- Uniform grid efficient even for a significant number of rays
  - In the range of millions