Eurographics 2012
Cagliari, Italy
May 13-18
33rd ANNUAL CONFERENCE OF THE EUROPEAN ASSOCIATION FOR COMPUTER GRAPHICS

Incoherent Ray Tracing without Acceleration Structures

Attila Áfra

Budapest University of Technology, Hungary
Babeș-Bolyai University, Romania

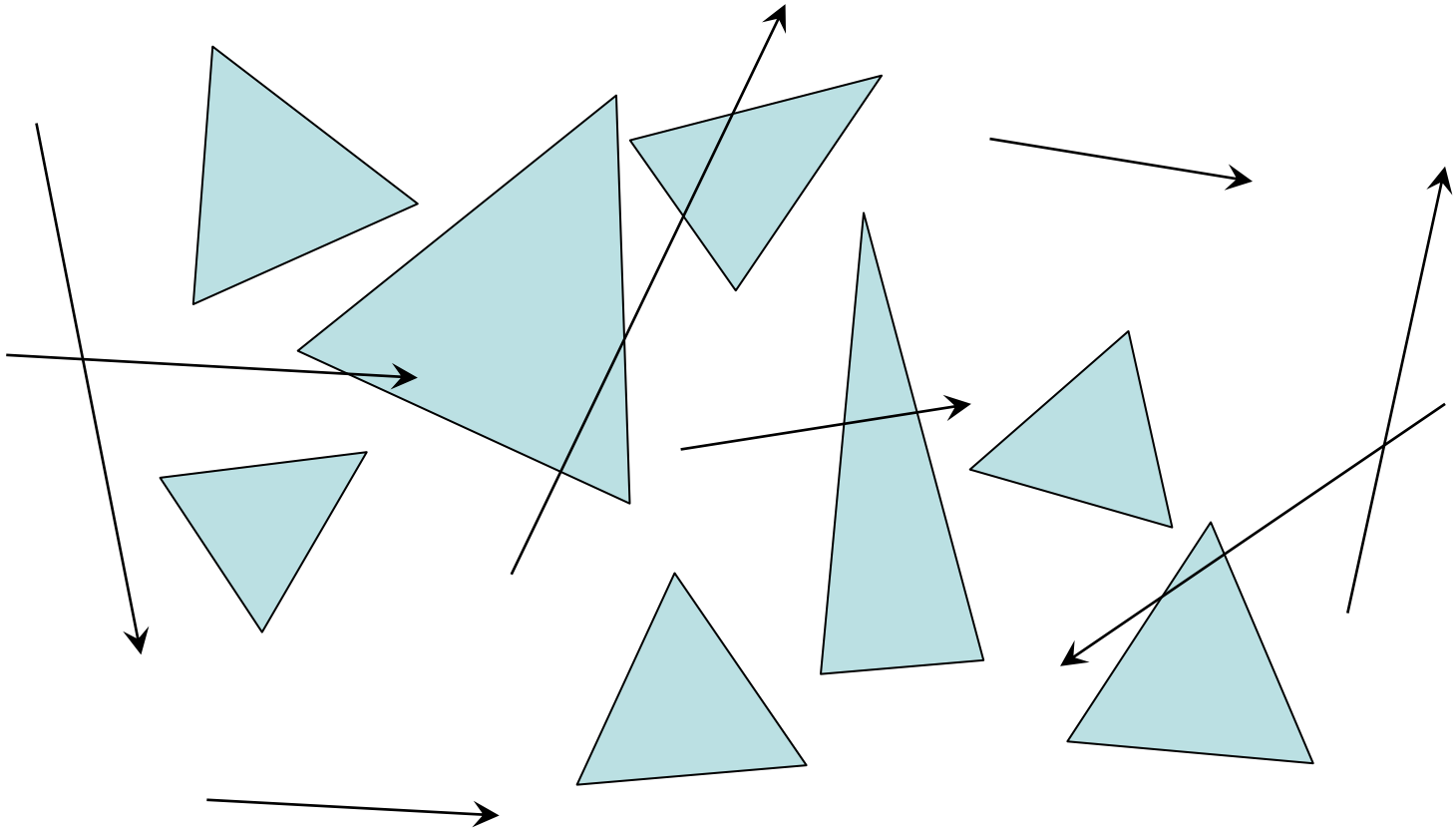
Divide-and-conquer ray tracing

- **Recently introduced**
  - [Keller and Wächter 2011]
  - [Mora 2011]

- **Does not use any acceleration structures!**
- **Competitive performance**

- **Input: set of rays, set of objects**
- **Almost 0 temporary memory usage**

Algorithm



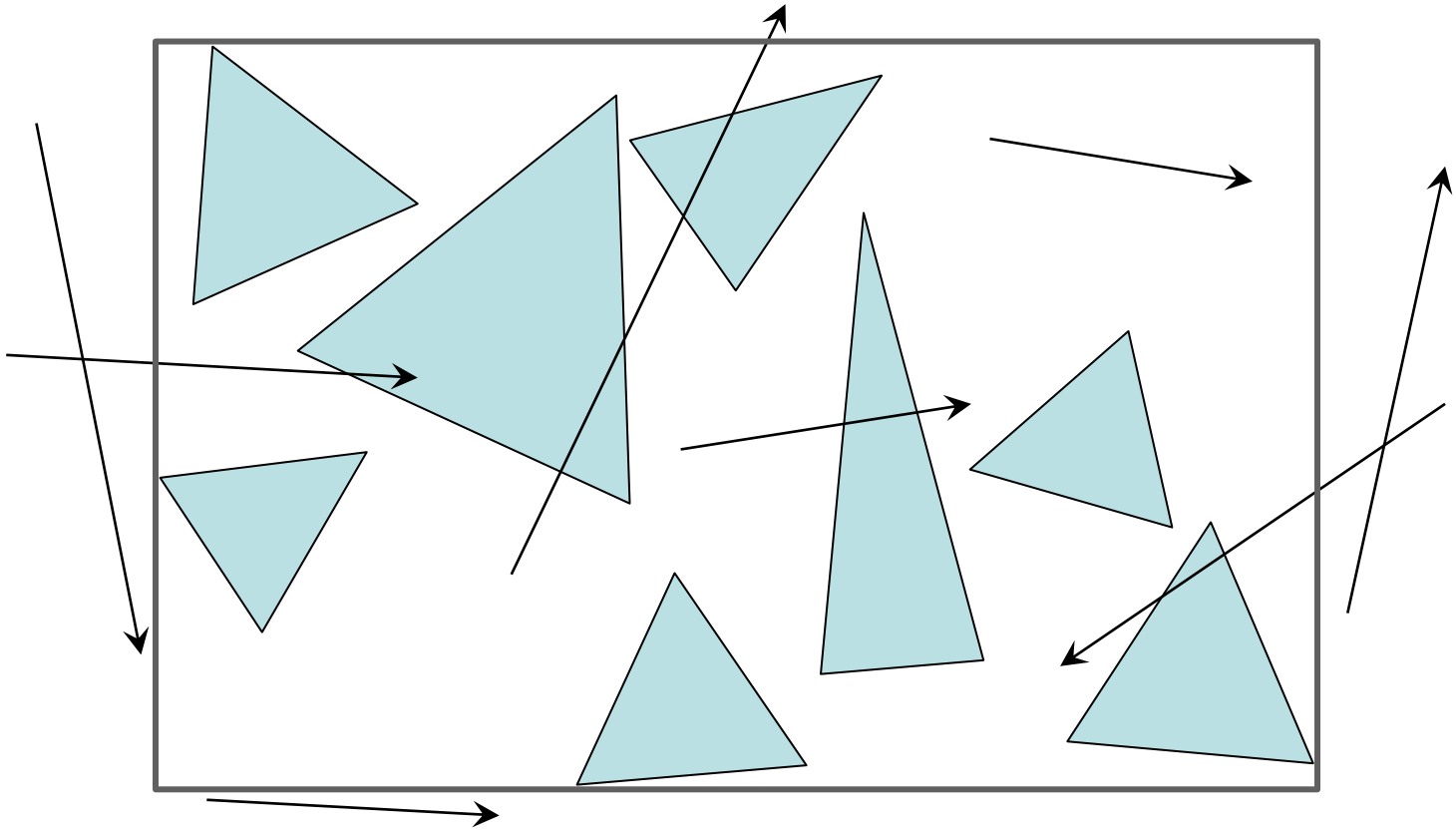
Rays

0	1	2	3	4	5	6	7

Triangles

A	B	C	D	E	F	G	H

Algorithm – Bounding volume



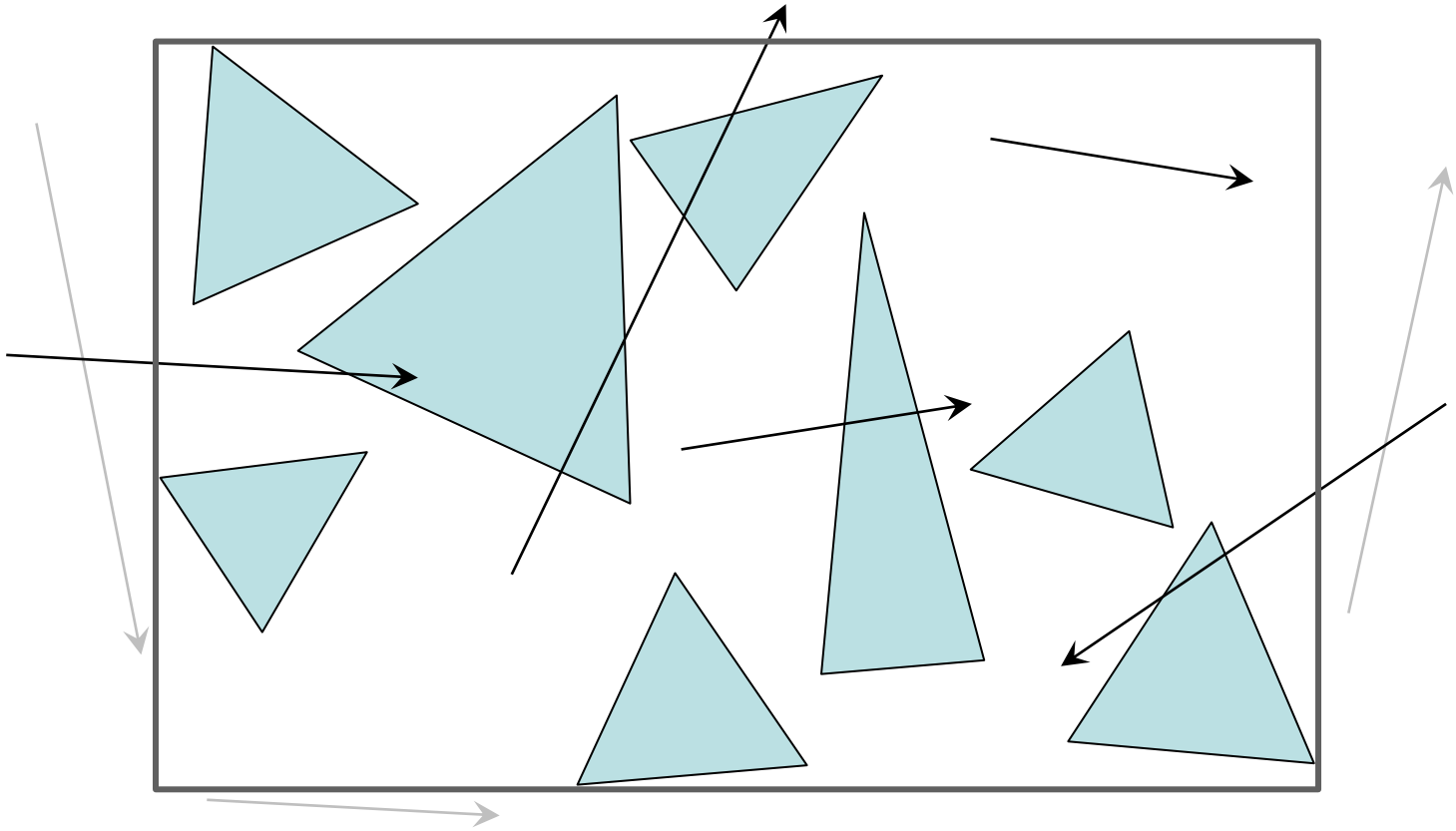
Rays

0	1	2	3	4	5	6	7

Triangles

A	B	C	D	E	F	G	H

Algorithm – Ray filtering



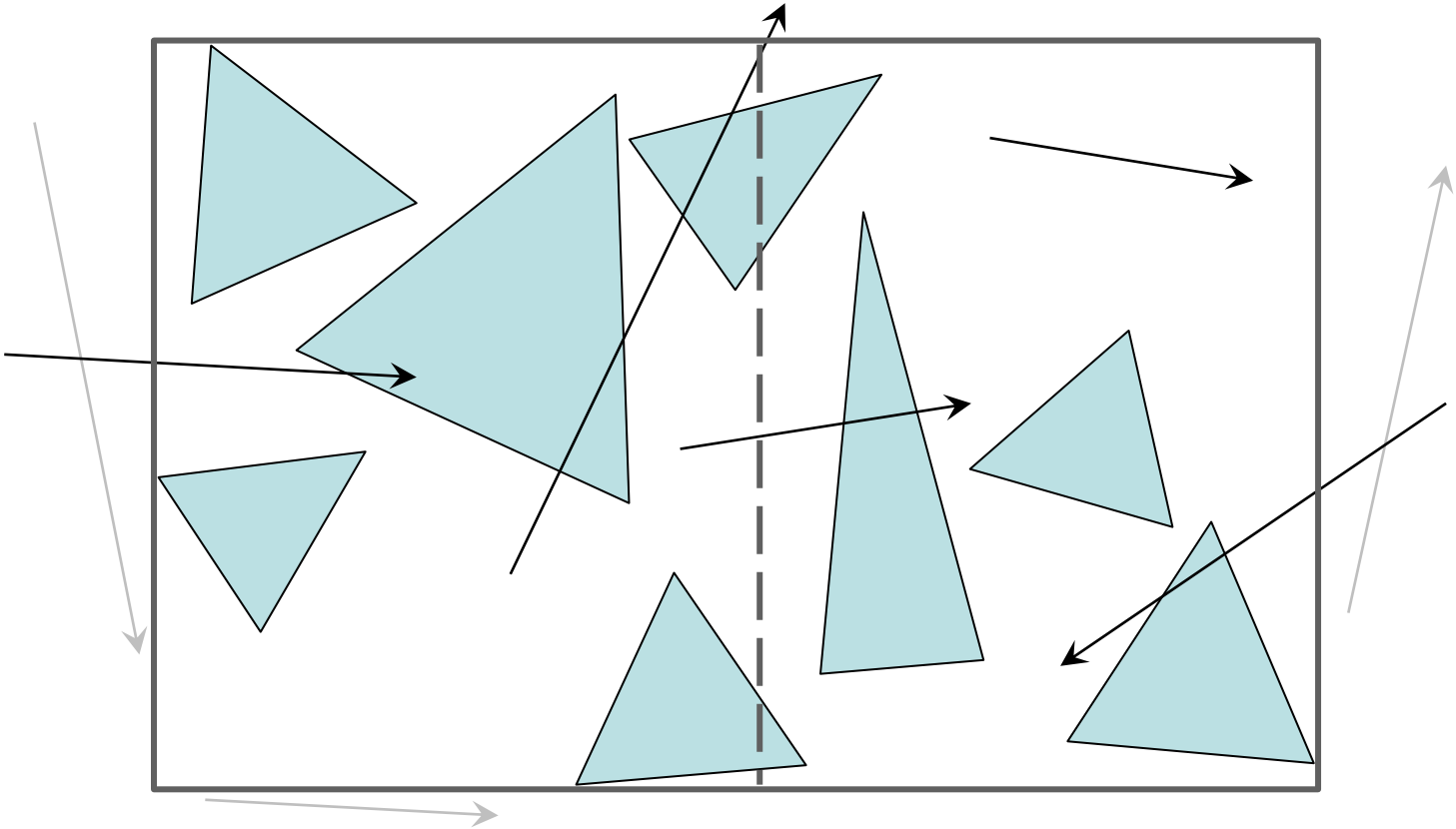
Rays

1	2	4	5	7	3	6	0

Triangles

A	B	C	D	E	F	G	H

Algorithm – Object partitioning



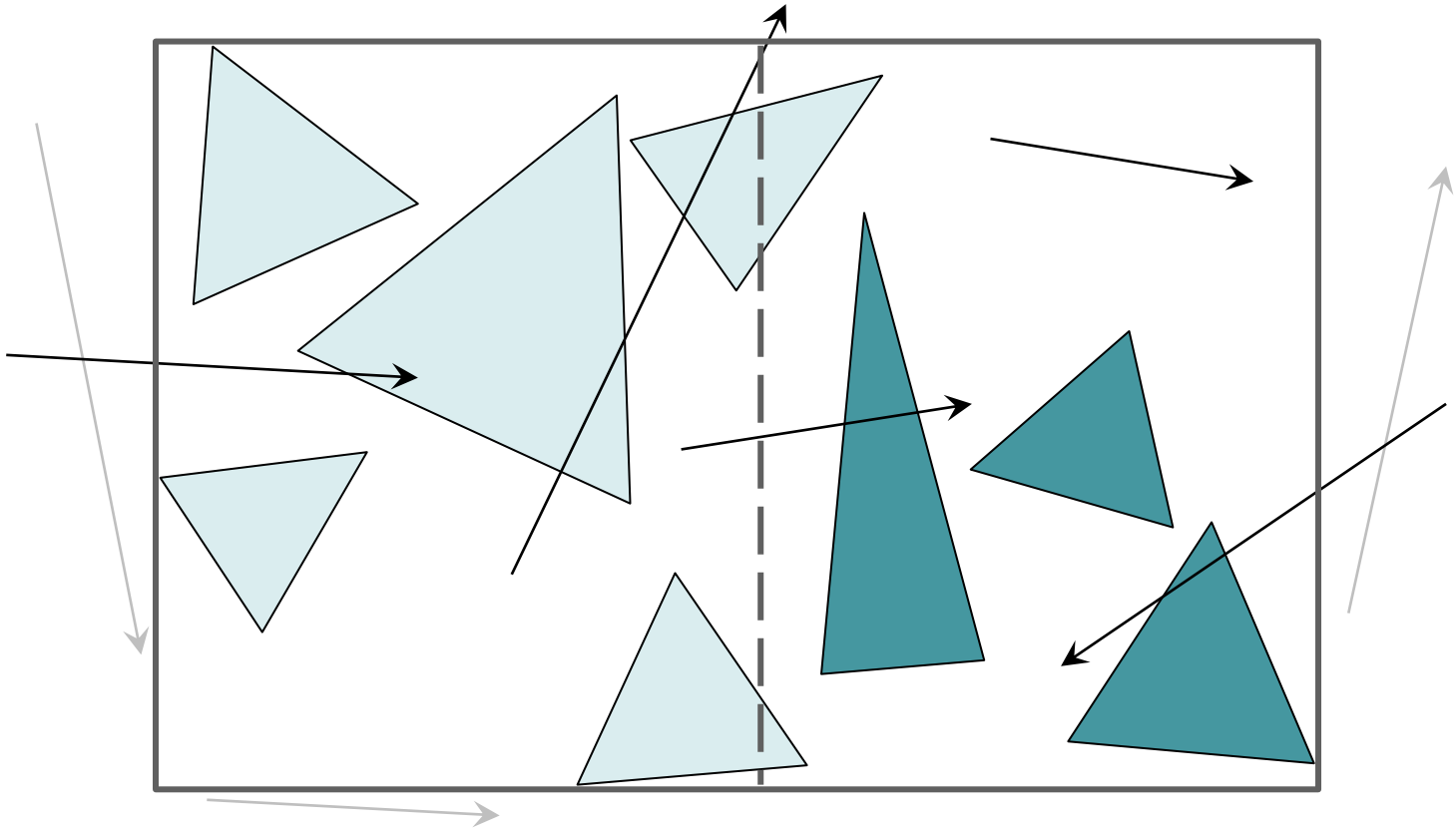
Rays

1	2	4	5	7	3	6	0

Triangles

A	B	C	D	E	F	G	H

Algorithm – Object partitioning



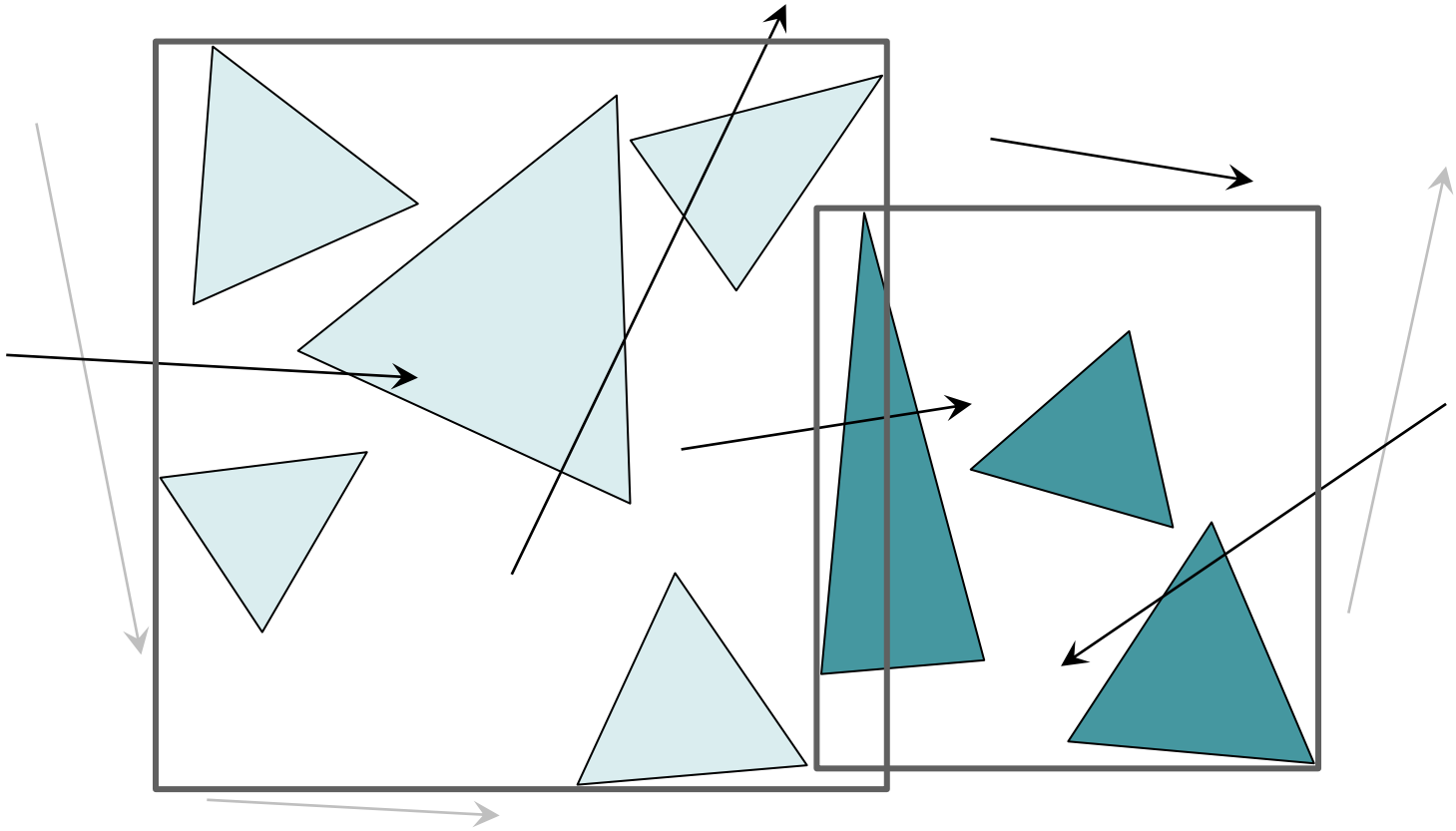
Rays

1	2	4	5	7	3	6	0

Triangles

B	C	E	G	H	F	D	A

Algorithm – Recursion



Rays

1	2	4	5	7	3	6	0

Triangles

B	C	E	G	H	F	D	A

New method

- **Divide-and-conquer ray tracer**
- **Optimized for *incoherent rays***
  - Goal: efficient cache and SIMD utilization
- **Optimized for *8-wide SIMD***
  - **AVX** (Advanced Vector Extensions)
  - Successor of SSE
  - Intel Sandy Bridge (2011)

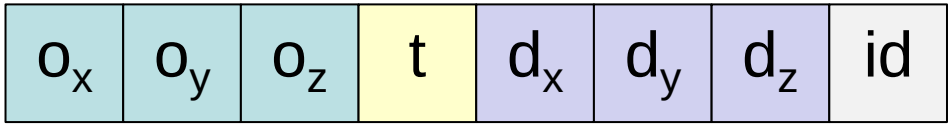
Ray filtering

- **Large ray array (millions of rays)**
- **Filtering partitions it into: active, inactive rays**
  - Active rays intersect the current AABB

- **Reorders *ray data***
- **Reordering *indices* is slower for incoherent rays**
- **Efficient cache usage**
  - Linear memory accesses
  - Cache space not wasted

Ray data layout

- **Should be compact to reduce bandwidth usage**
- **32 bytes**
  - Fits exactly into an AVX register → **SIMD-friendly**
  - Fast moves (one instruction)



- **Origin** (o)
- **Direction** (d)
  - Reciprocal direction not stored, too expensive
- **Hit distance** (t)
  - Frequently accessed, should not be kept separately
- **ID**
  - Required to identify reordered rays

Ray-box intersection

- **SIMD: 8 rays with 1 AABB**

- **Requires rays in SoA layout**
- **Convert from AoS with SIMD loads and shuffles**

- **AABB computed during triangle partitioning**
  - Avoids an extra sweep over the triangles

Triangle partitioning

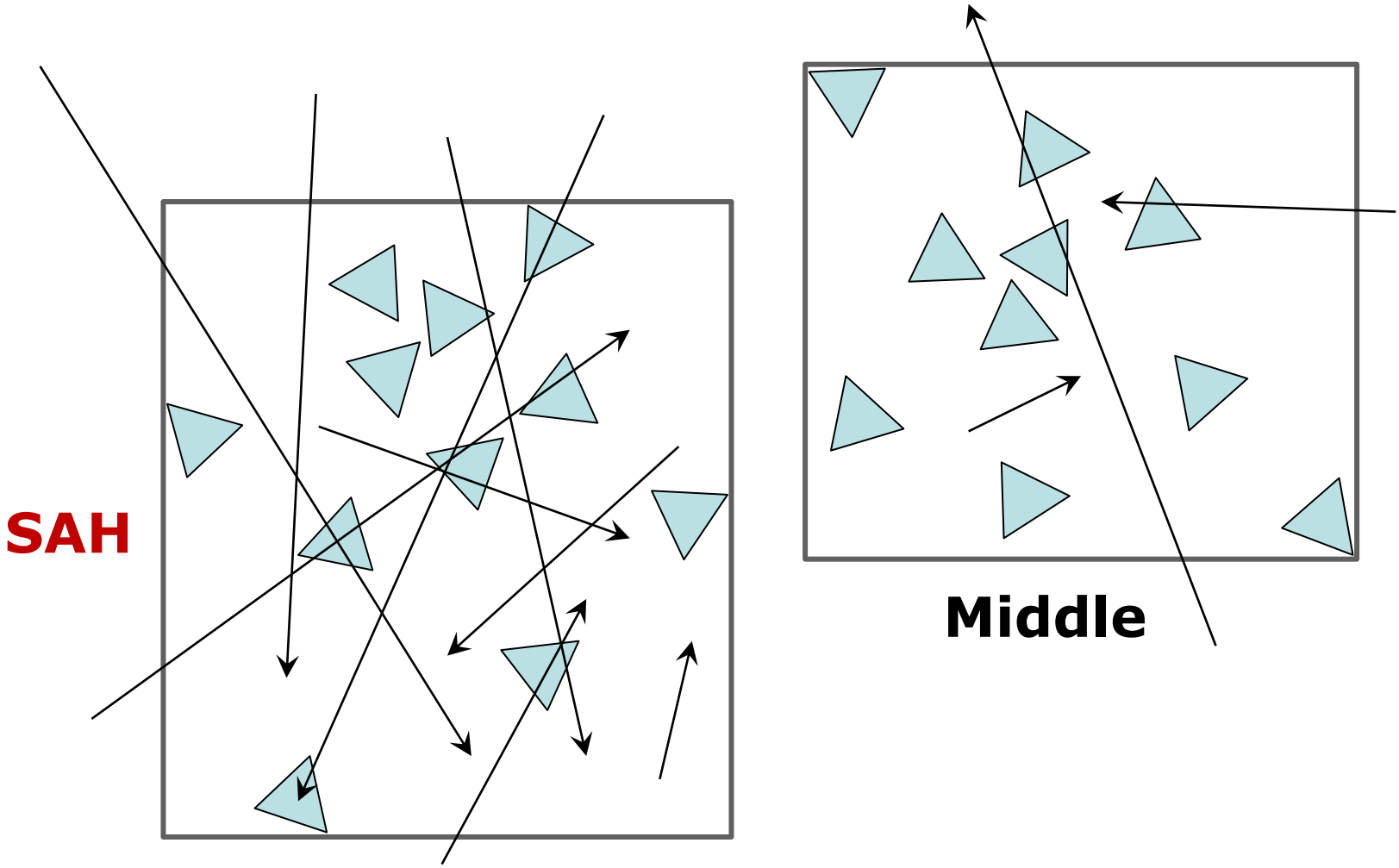
- **BVH-like *object list partitioning***
  - Disjoint partitions (like quicksort)
- **Reorders a triangle ID array**
- **Precomputed triangle AABB array**
  - SIMD-optimized layout
- **Two partitioning methods**
  - *Middle partitioning (spatial median)* → fast
  - *SAH partitioning (binned)* → slower, but higher-quality

Adaptive partitioning

- **Takes into account the ray distribution**
  - High-quality partitioning only where it makes sense
  - Not possible with prebuilt acceleration structures!

- **Method selected according to *ray/triangle ratio***
  - SAH partitioning only if ratio > 1-2
  - Very simple and works quite well

Adaptive partitioning



Ray-triangle intersection

- **Stop partitioning if too few rays or triangles**
  - Threshold: 8
- **Intersect all rays with all triangles**
- **SIMD: 8 rays with 1 triangle**
- **Compact triangle array**
  - AABB array already contains 6 vertex coords per triangle
  - Store only the remaining 3 coords and shuffle info

Multithreading

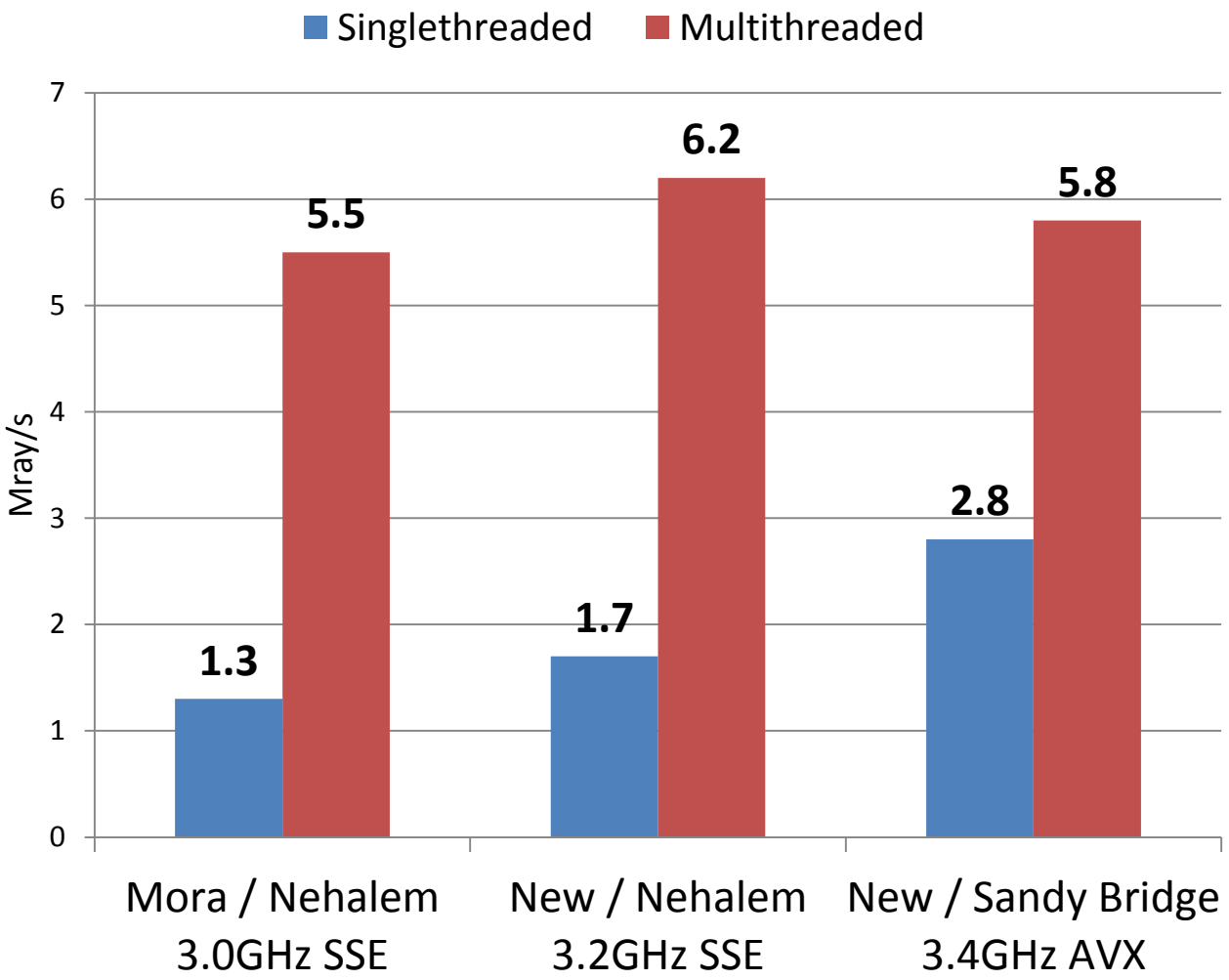
- **Trace N ray batches in parallel**
- **Simple, no synchronization, *but…***
- **Suboptimal**
  - Many triangle partitioning steps are performed multiple times
  - **Not a big problem in practice!**
  - Partitioning usually only 10-20% of total runtime

Results

Conference Room
282K triangles

- **New method**
- **[Mora 2011]**

- 8-bounce diffuse path tracing



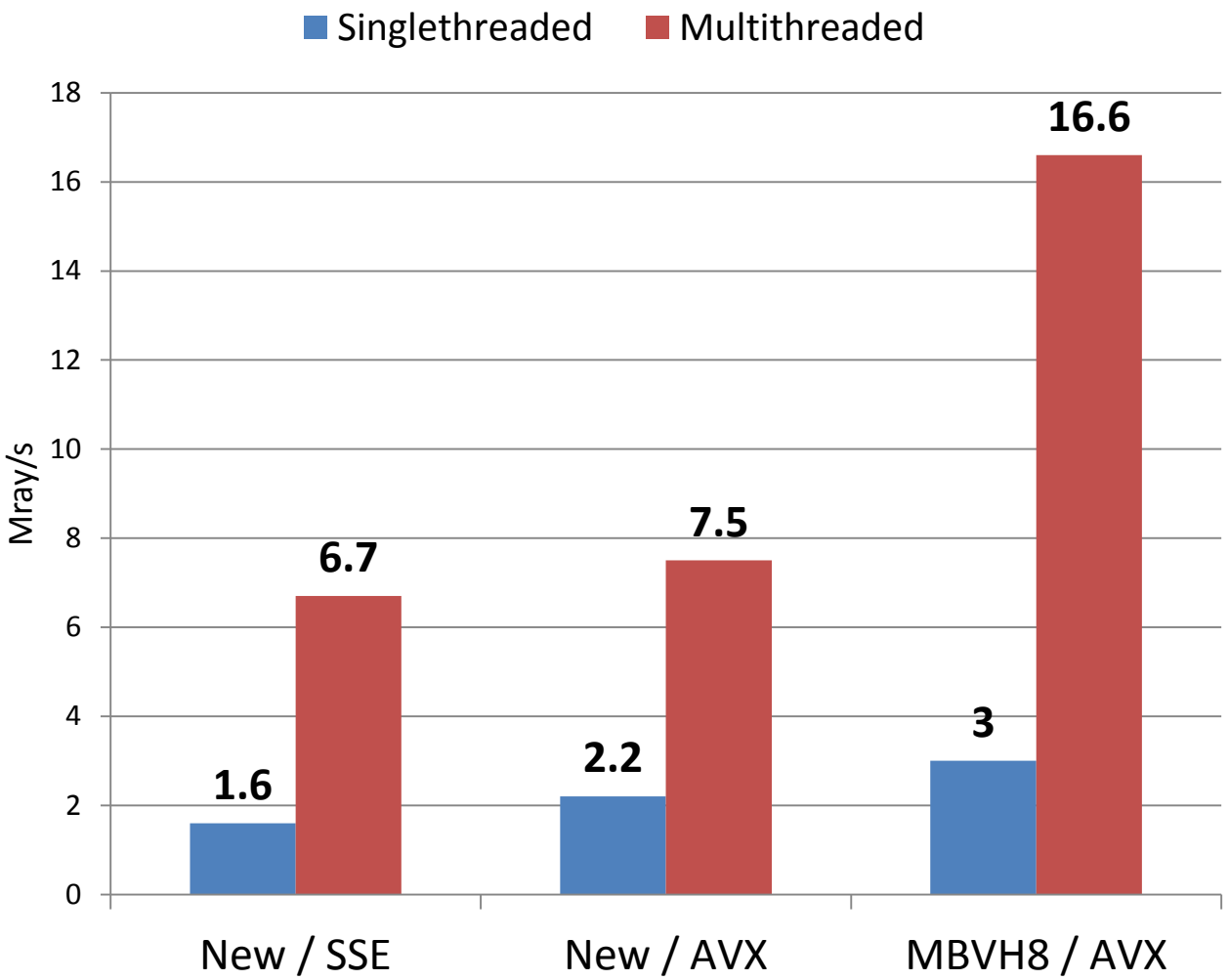
Results

Fairy Forest
174K triangles

- **New method**
- **MBVH8**

- Timings do **not** include MBVH build!

- Sandy Bridge 3.4GHz



Demo: Fairy Forest

Core i7-2600, 2-bounce diffuse, 640x400, 8 spp → 8 Mray/s

Conclusions

- **The method is still quite experimental**
- **Elegant, easy to use!**
- **Not much slower than traditional ray tracing**
  - With prebuilt acceleration structures!
- **Interesting applications**
  - Adaptive tessellation, augment rasterization, etc.

- **Future work**
  - Reduce memory traffic: ray compression, pre-sorting?
  - More efficient multithreading
  - GPU

Thank you!

- **Questions?**