Fast Parallel Construction of High-Quality Bounding Volume Hierarchies

Tero Karras

Timo Aila

Ray tracing comes in many flavors


NVIDIA


Courtesy of Delta Tracing


Lucasfilm Ltd.™, Digital work by ILM


© Activision 2009, Game trailer by Blur Studio


Courtesy of Dassault Systemes


Courtesy of Columbia Pictures

Interactive apps	Architecture & design	Movie production
1M–100M rays/frame	100M–10G rays/frame	10G–1T rays/frame

Effective performance

$$effective\ ray\ tracing\ performance = \frac{number\ of\ rays}{rendering\ time}$$

Effective performance

$$effective\ ray\ tracing\ performance = \frac{number\ of\ rays}{rendering\ time}$$

$$rendering\ time = time\ to\ build\ BVH + \frac{number\ of\ rays}{ray\ throughput}$$

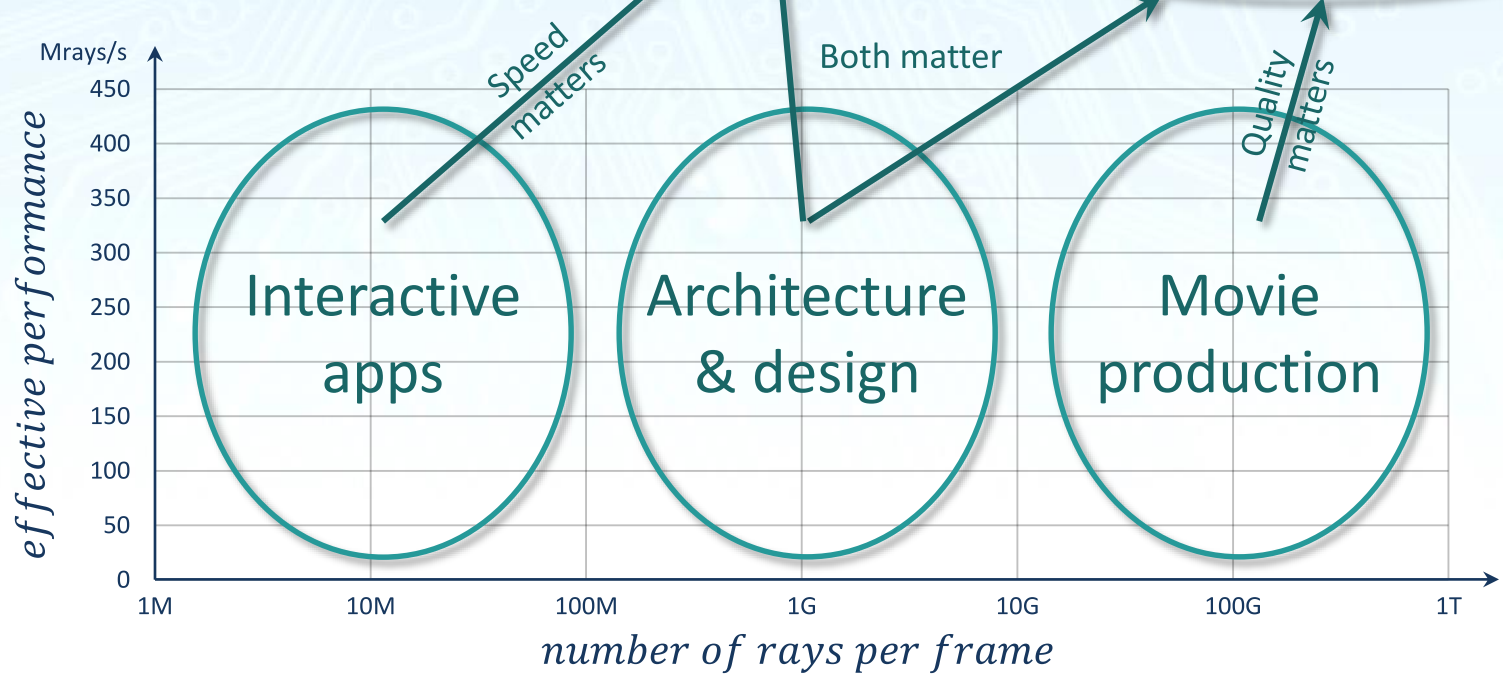
“speed”

“quality”

Effective performance

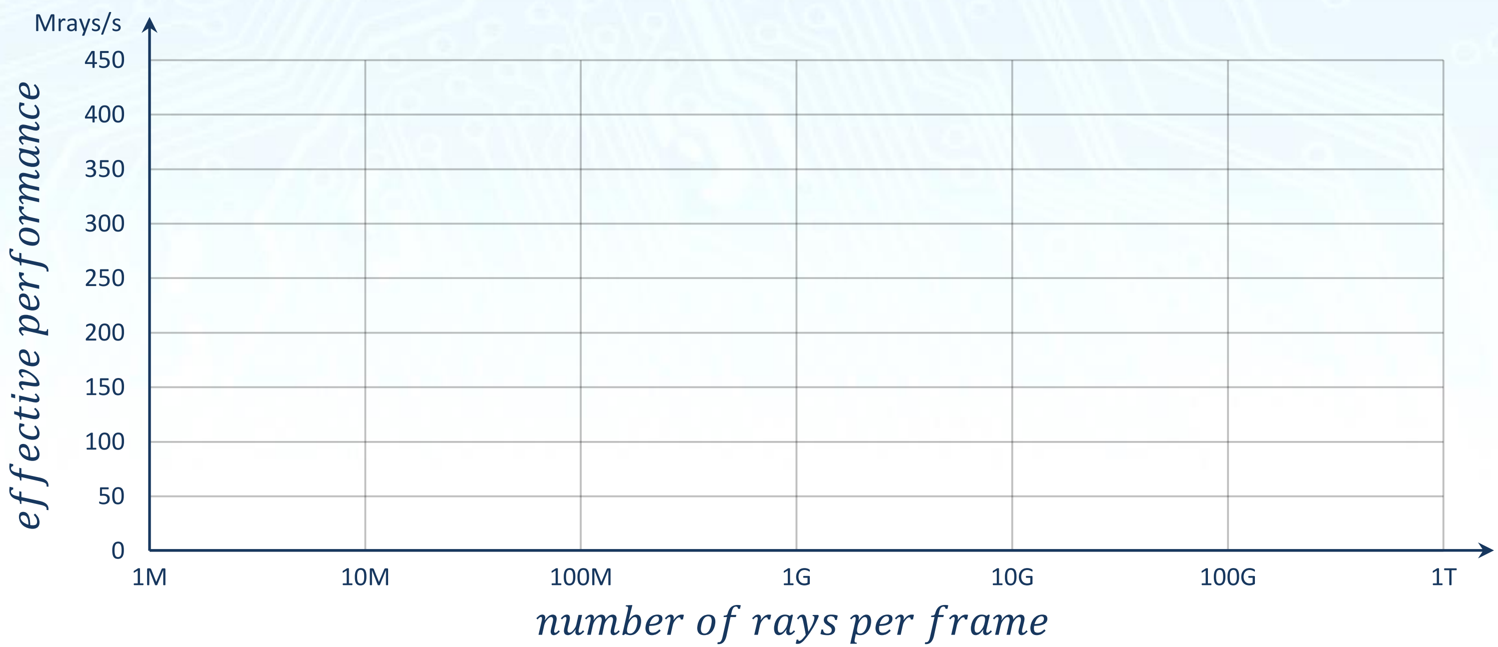
$$effective\ ray\ tracing\ performance = \frac{number\ of\ rays}{rendering\ time}$$

$$rendering\ time = time\ to\ build\ BVH + \frac{number\ of\ rays}{ray\ throughput}$$

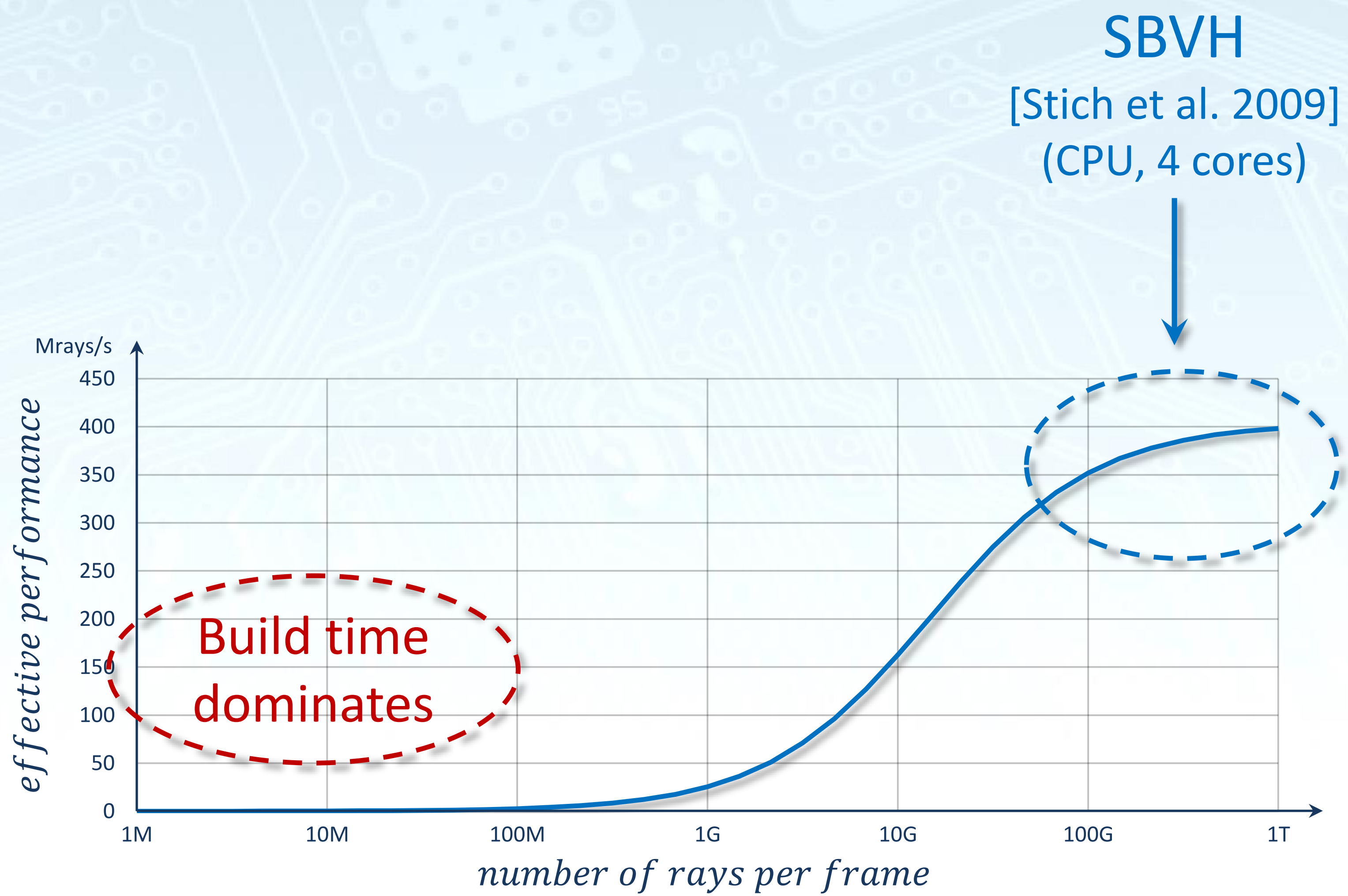


Effective performance

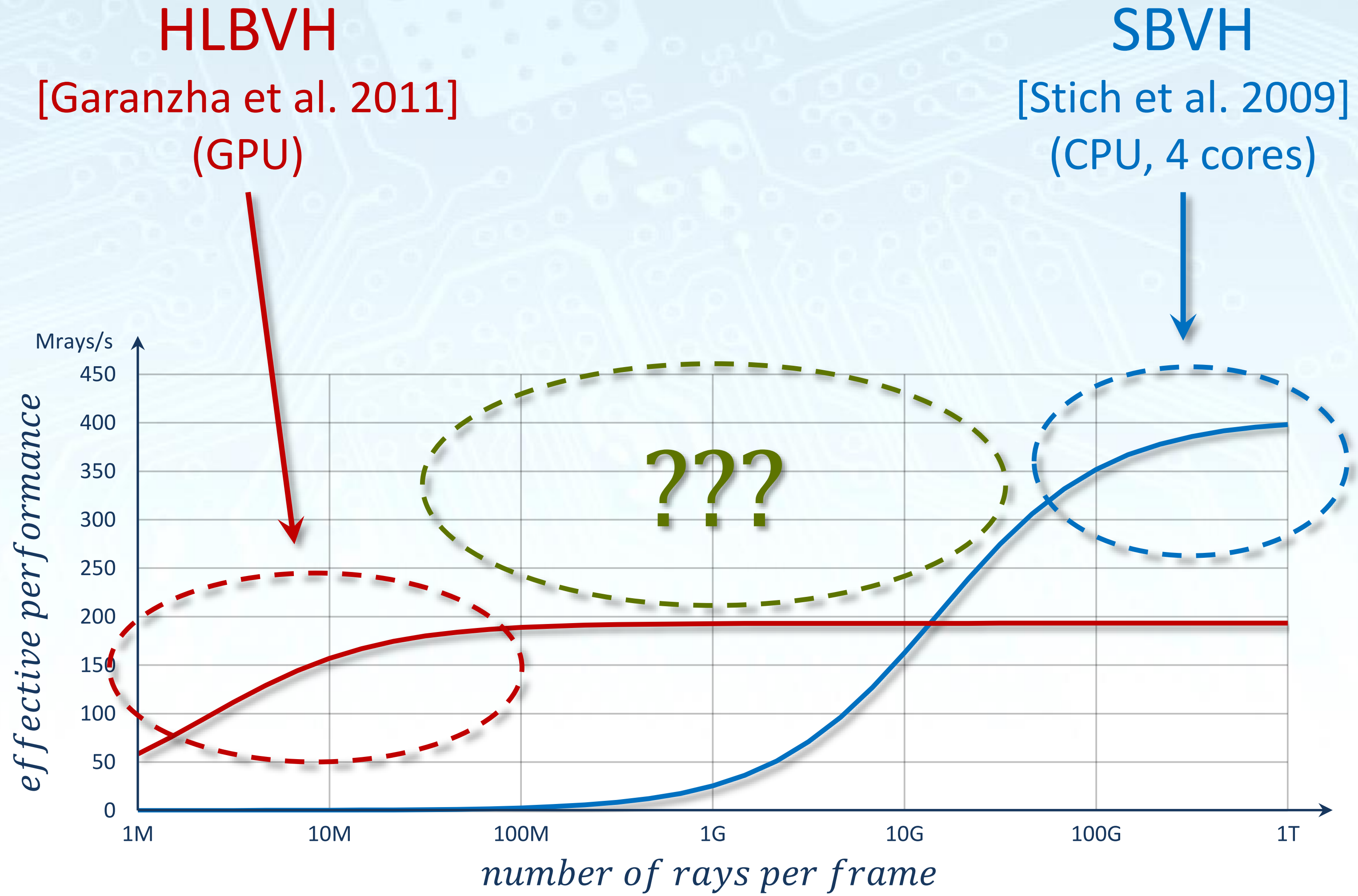
- Soda (2.2M tris)
- NVIDIA GTX Titan
- Diffuse rays



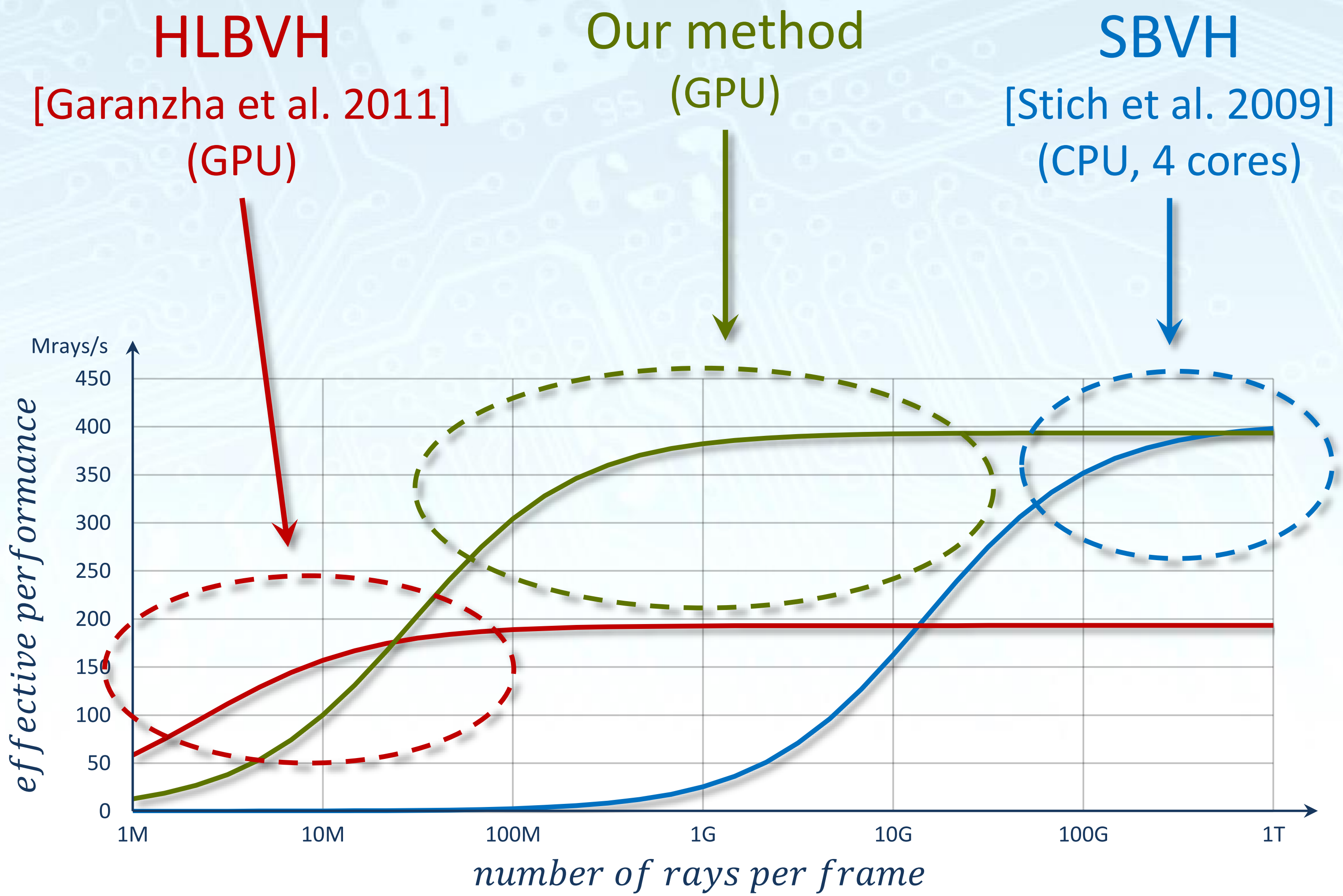
Effective performance



Effective performance

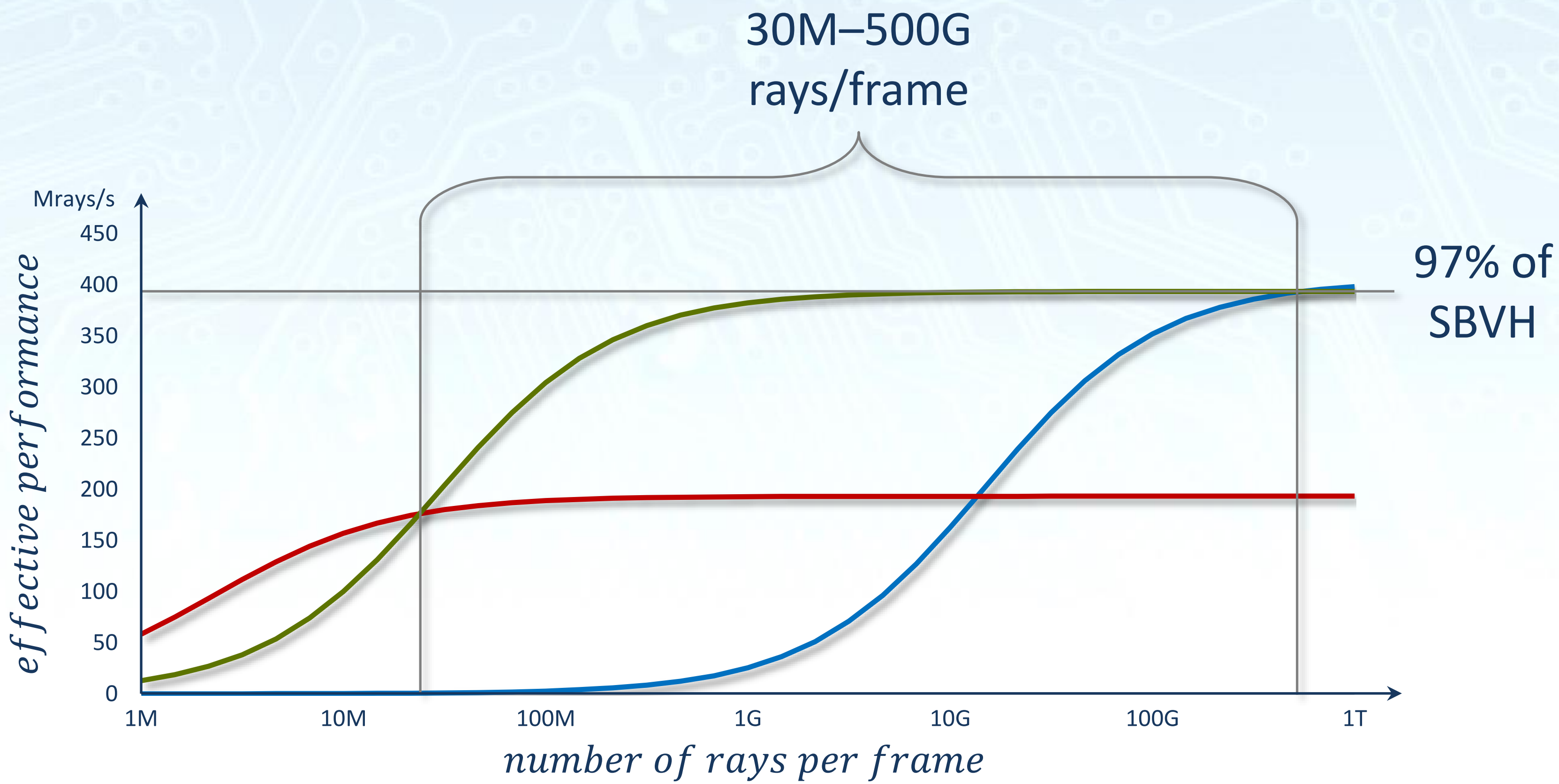


Effective performance



Effective performance

- Best quality–speed tradeoff for wide range of applications

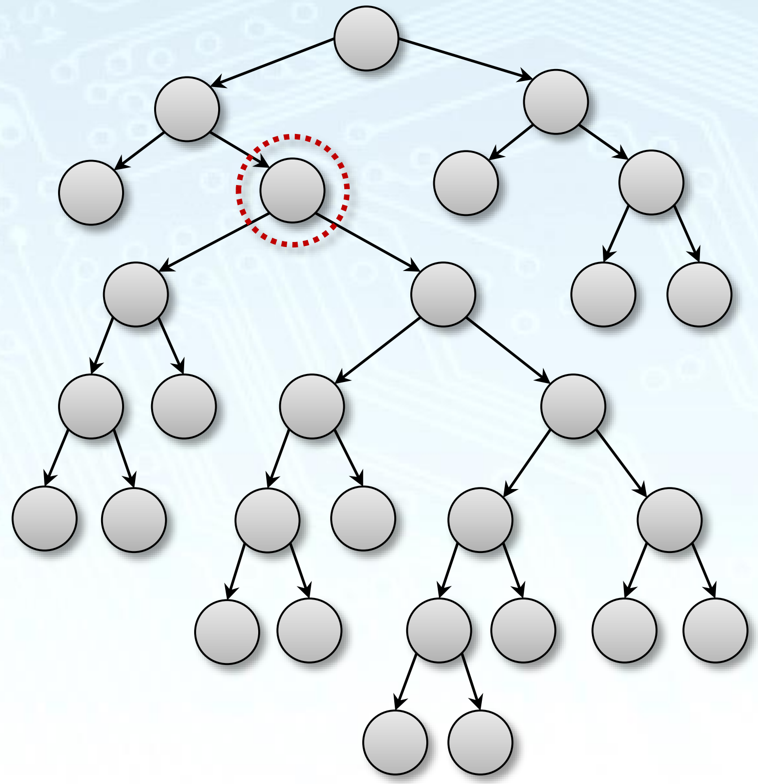


Treelet restructuring

- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once

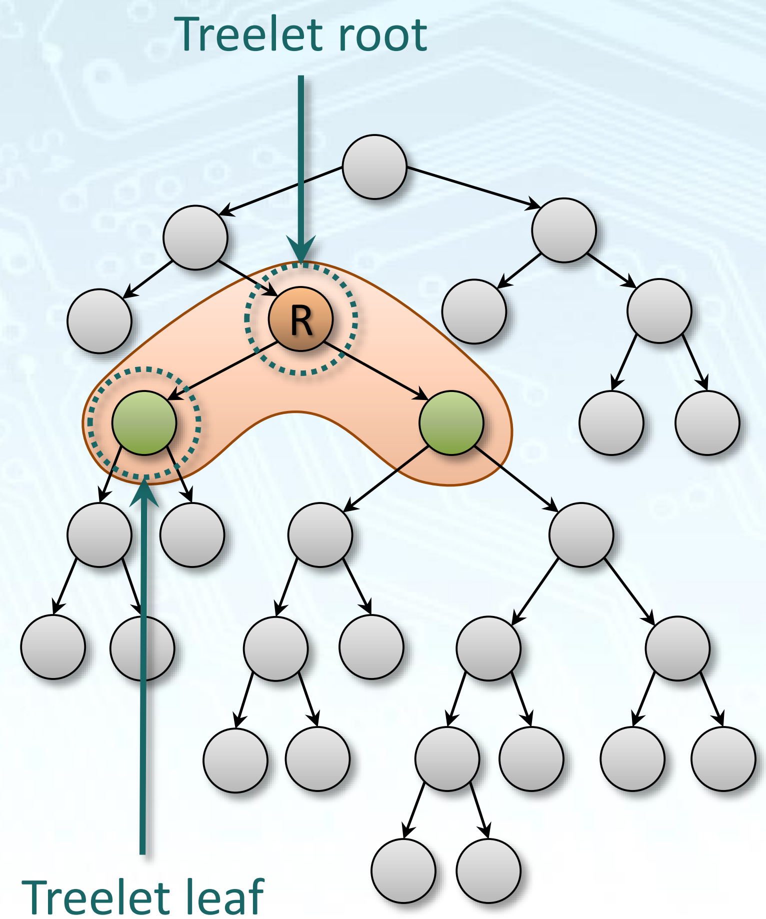
Treelet restructuring

- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once
- Treelet
  - Subset of a node’s descendants



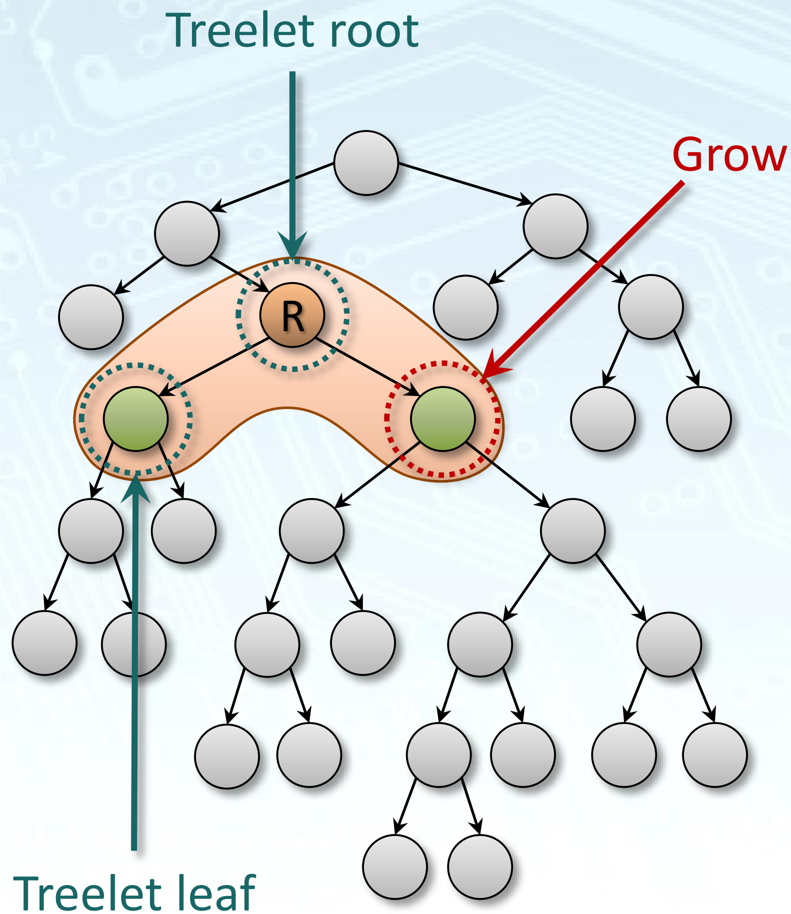
Treelet restructuring

- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once
- Treelet
  - Subset of a node's descendants



Treelet restructuring

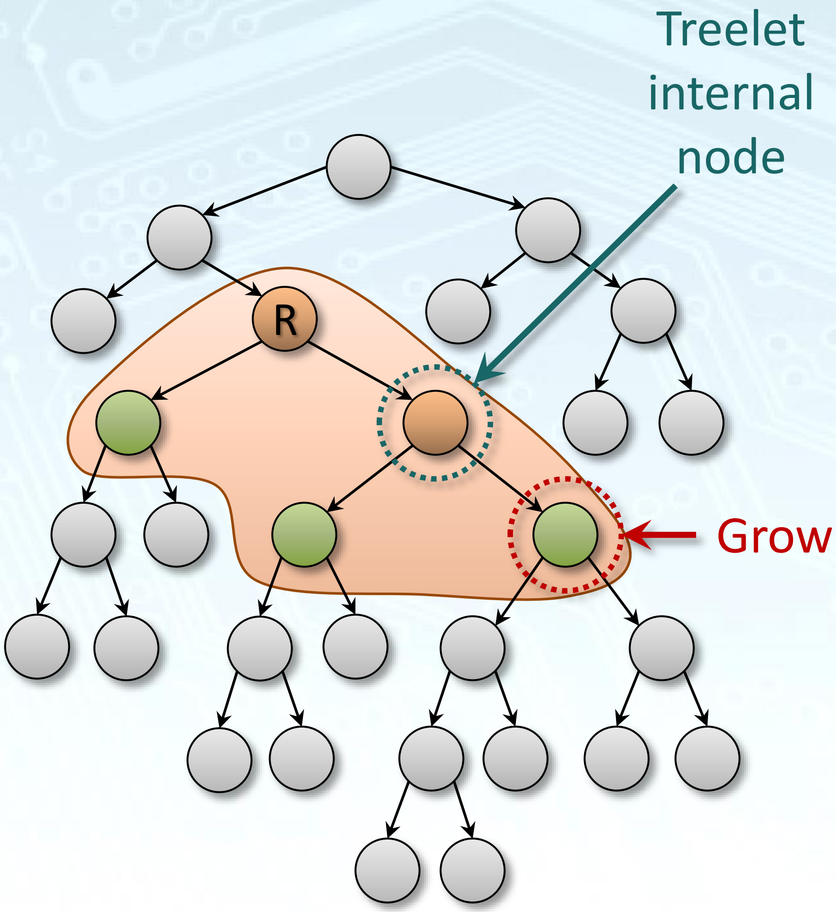
- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once
- Treelet
  - Subset of a node's descendants
  - Grow by turning leaves into internal nodes



Treelet restructuring

- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once

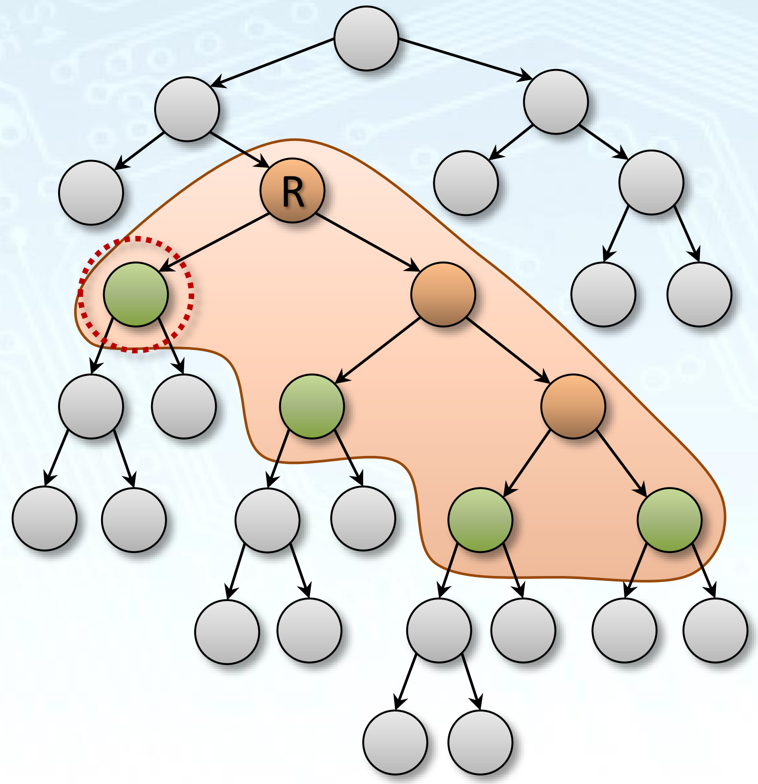
- Treelet
  - Subset of a node's descendants
  - Grow by turning leaves into internal nodes



Treelet restructuring

- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once

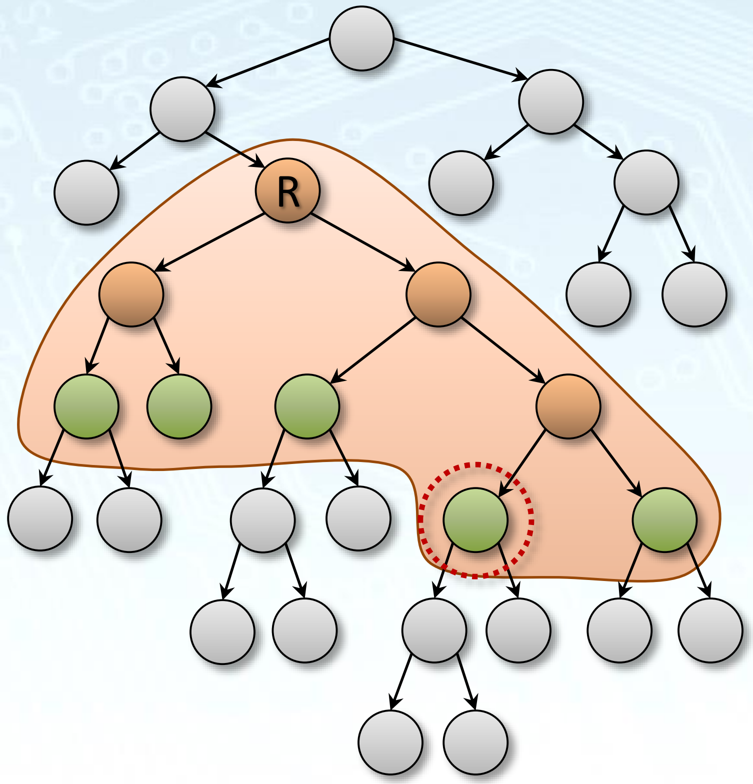
- Treelet
  - Subset of a node's descendants
  - Grow by turning leaves into internal nodes



Treelet restructuring

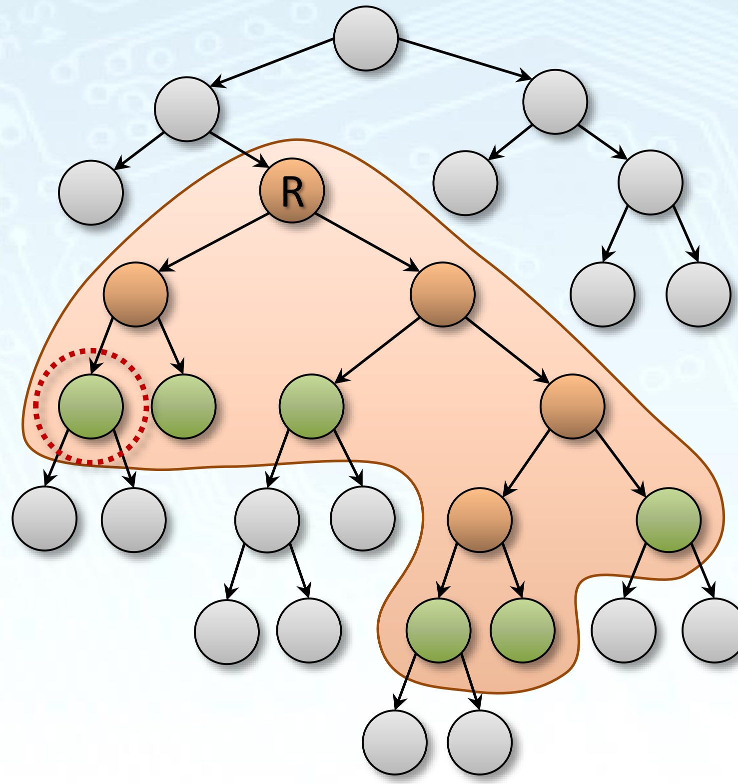
- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once

- Treelet
  - Subset of a node's descendants
  - Grow by turning leaves into internal nodes



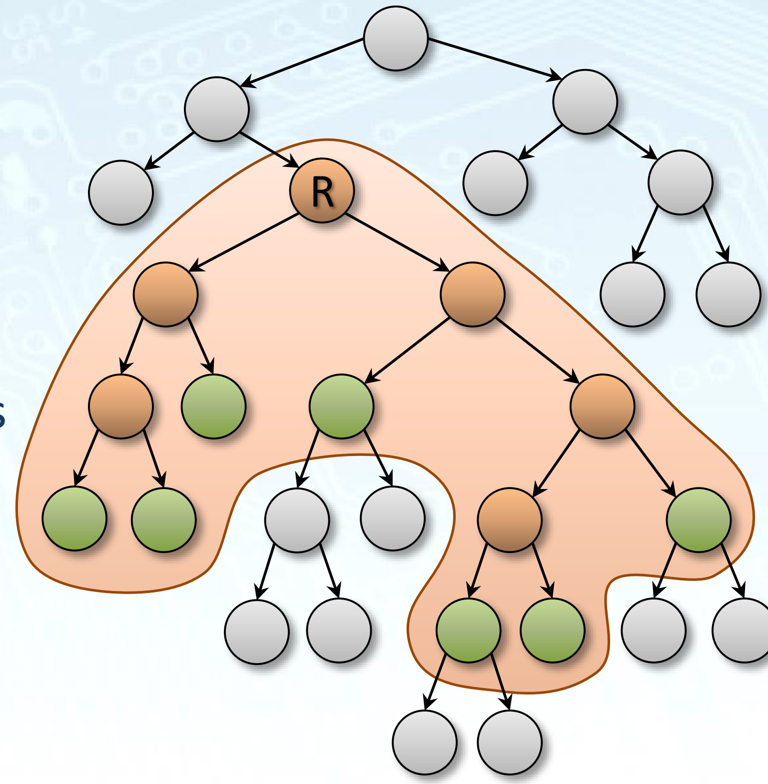
Treelet restructuring

- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once
- Treelet
  - Subset of a node's descendants
  - Grow by turning leaves into internal nodes



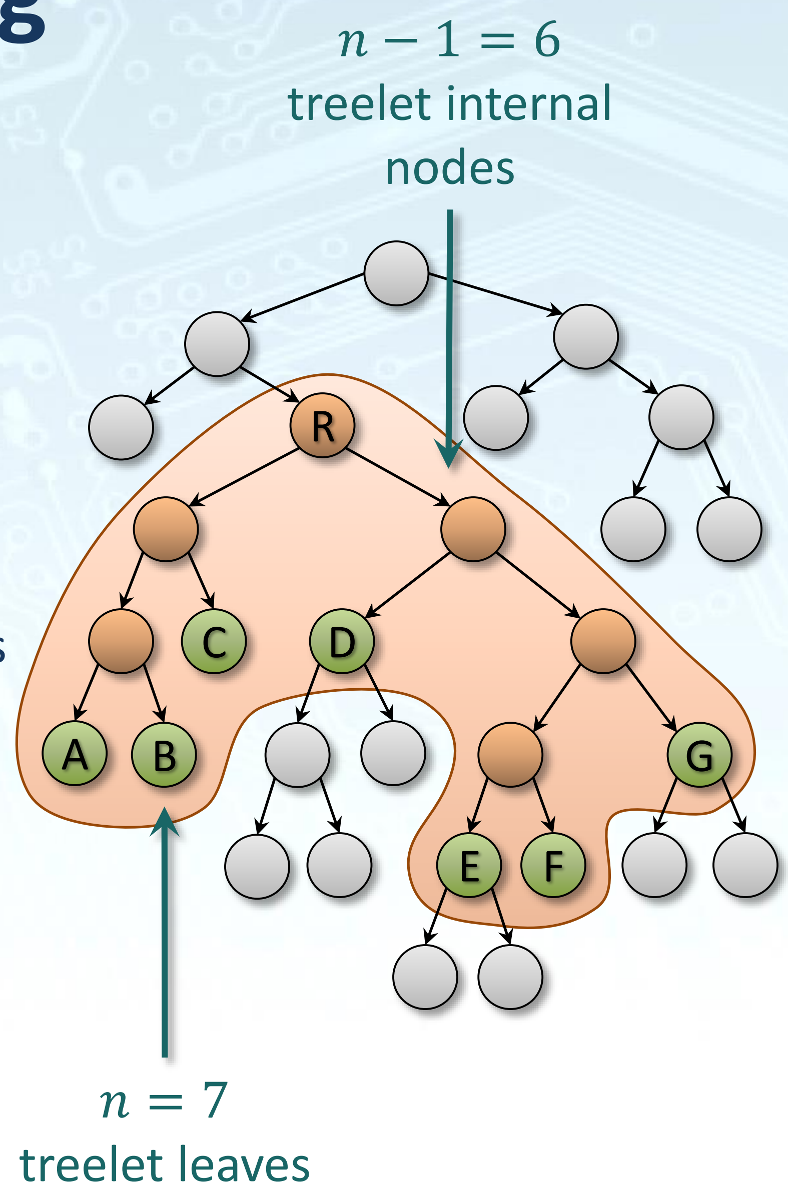
Treelet restructuring

- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once
- Treelet
  - Subset of a node’s descendants
  - Grow by turning leaves into internal nodes
  - Largest leaves → best results



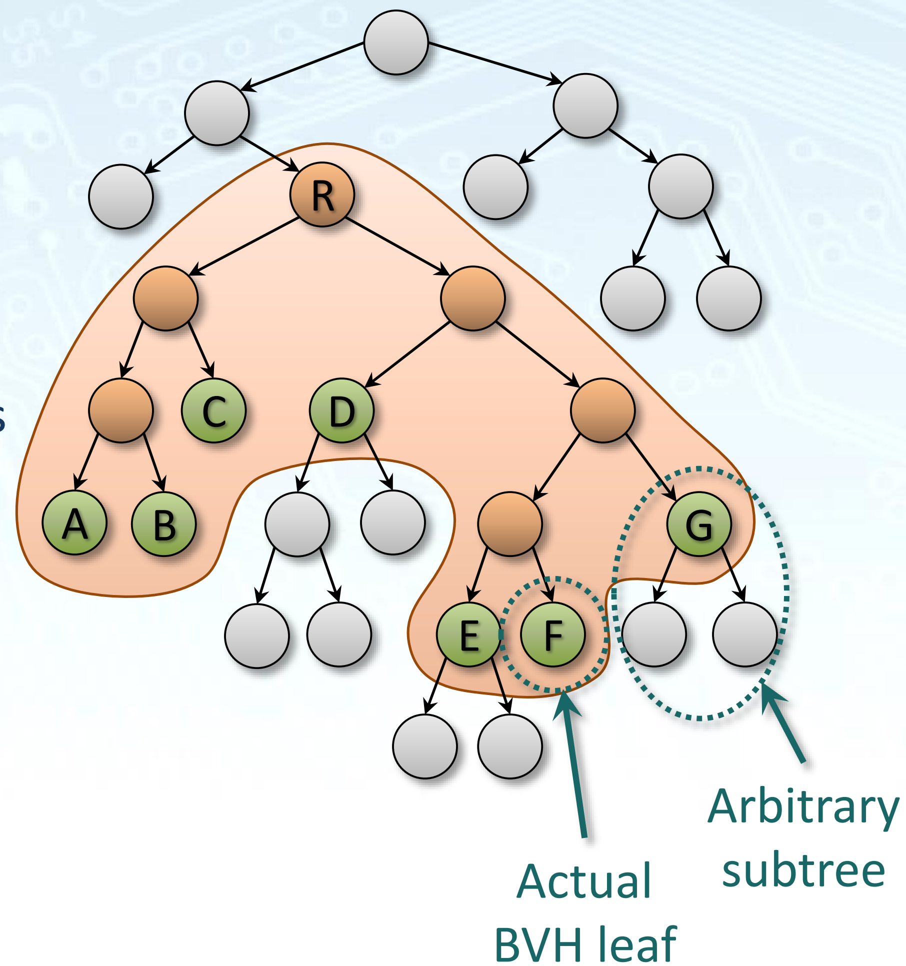
Treelet restructuring

- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once
- Treelet
  - Subset of a node's descendants
  - Grow by turning leaves into internal nodes
  - Largest leaves → best results
- Valid binary tree in itself



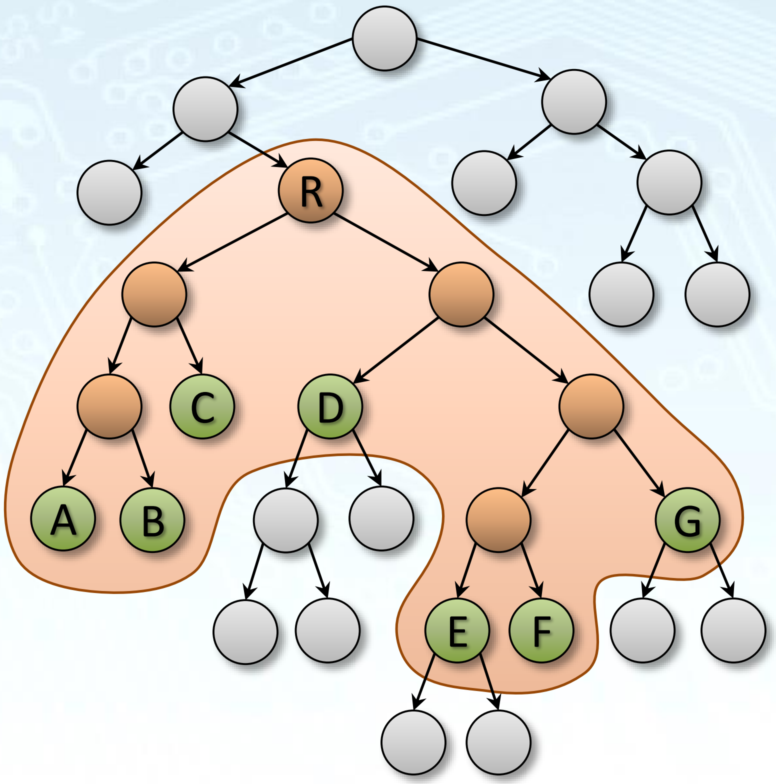
Treelet restructuring

- Idea
  - Build a low-quality BVH
  - Optimize its node topology
  - Look at multiple nodes at once
- Treelet
  - Subset of a node's descendants
  - Grow by turning leaves into internal nodes
  - Largest leaves → best results
- Valid binary tree in itself
  - Leaves can represent arbitrary subtrees of the BVH



Treelet restructuring

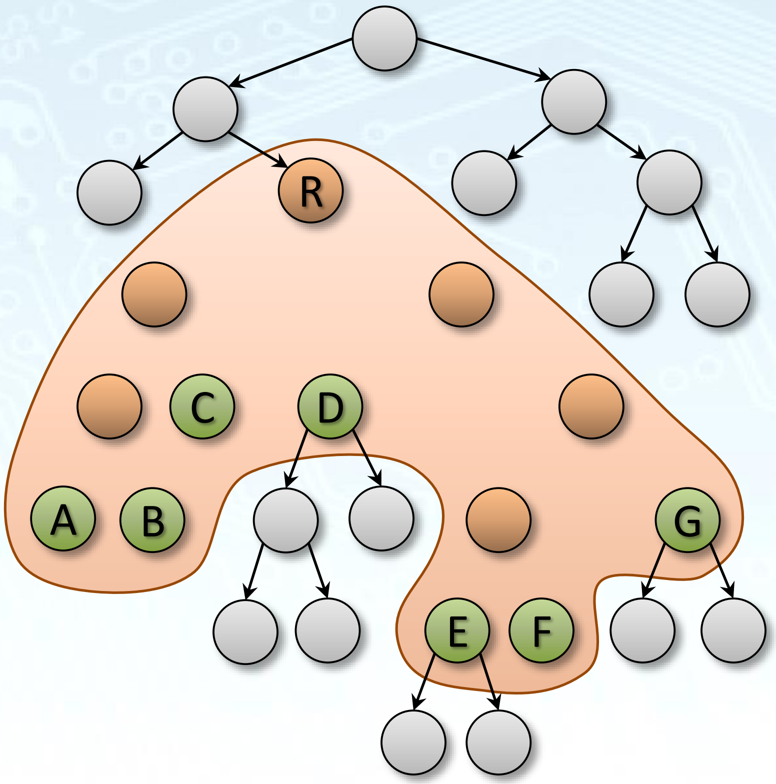
- Restructuring
  - Construct optimal binary tree for the same set of leaves
  - Replace old treelet



Treelet restructuring

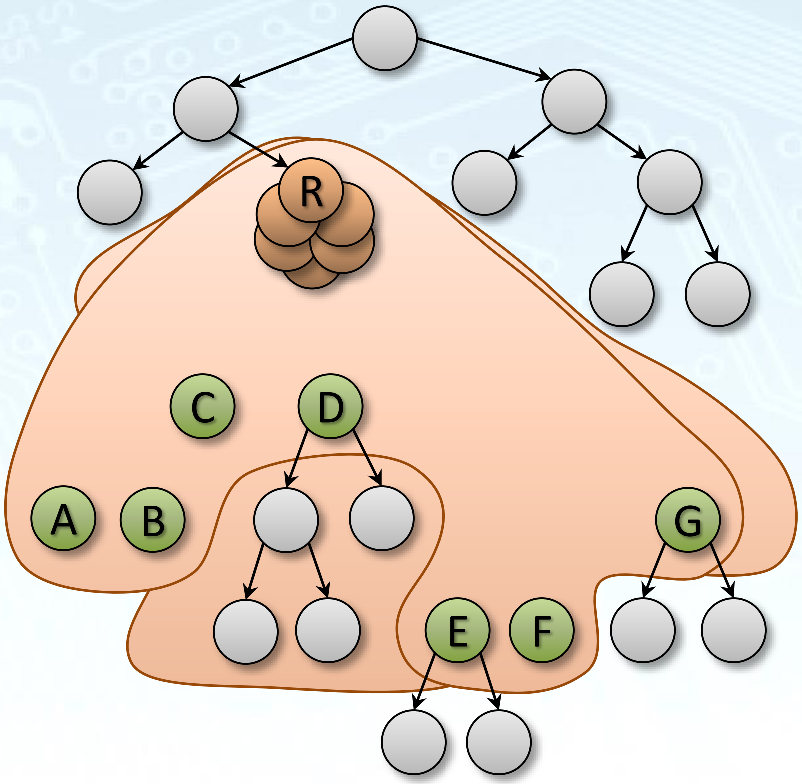
- Restructuring
  - Construct optimal binary tree for the same set of leaves
  - Replace old treelet

- Reuse the same nodes
  - Update connectivity and AABBs
  - New AABBs should be smaller



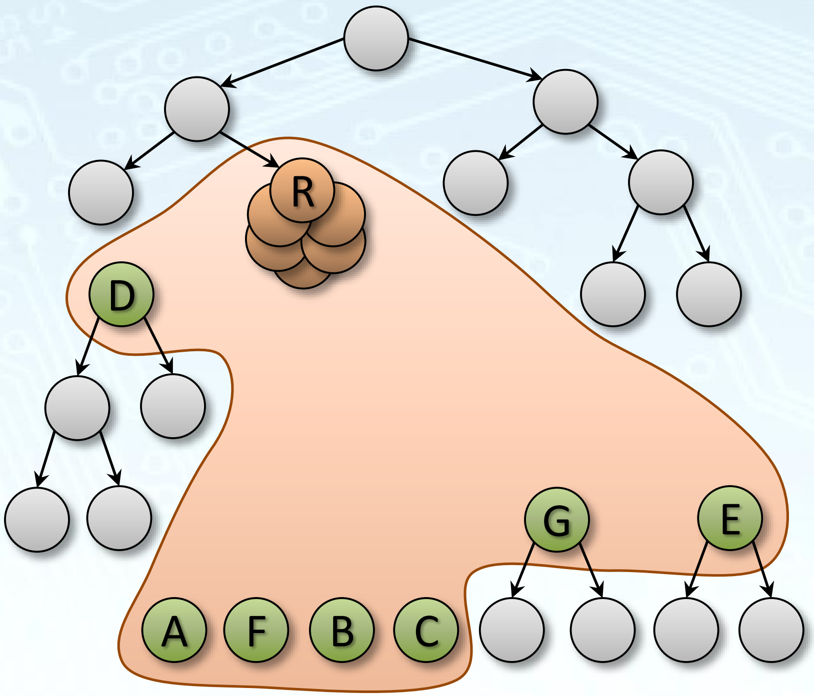
Treelet restructuring

- Restructuring
  - Construct optimal binary tree for the same set of leaves
  - Replace old treelet
- Reuse the same nodes
  - Update connectivity and AABBs
  - New AABBs should be smaller



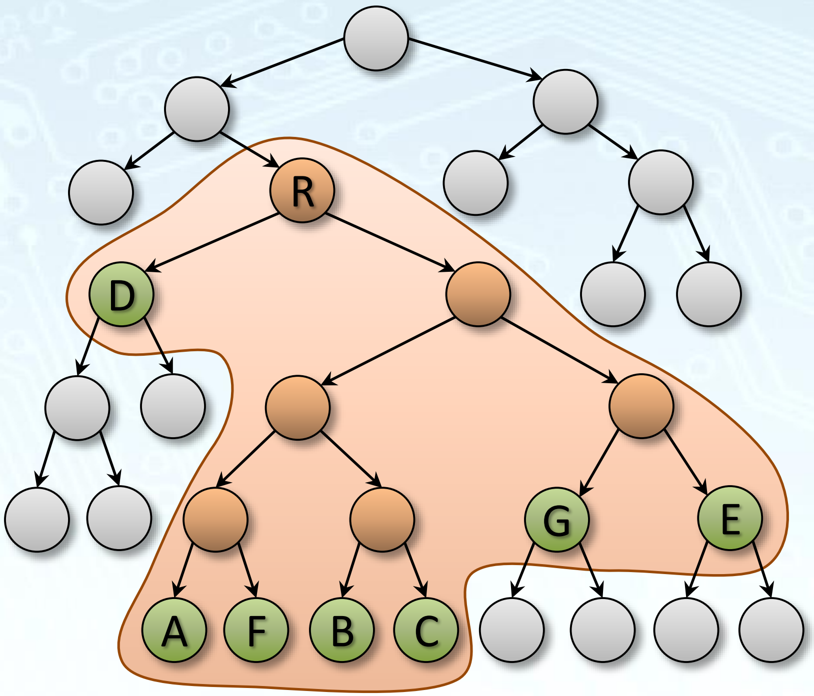
Treelet restructuring

- Restructuring
  - Construct optimal binary tree for the same set of leaves
  - Replace old treelet
- Reuse the same nodes
  - Update connectivity and AABBs
  - New AABBs should be smaller

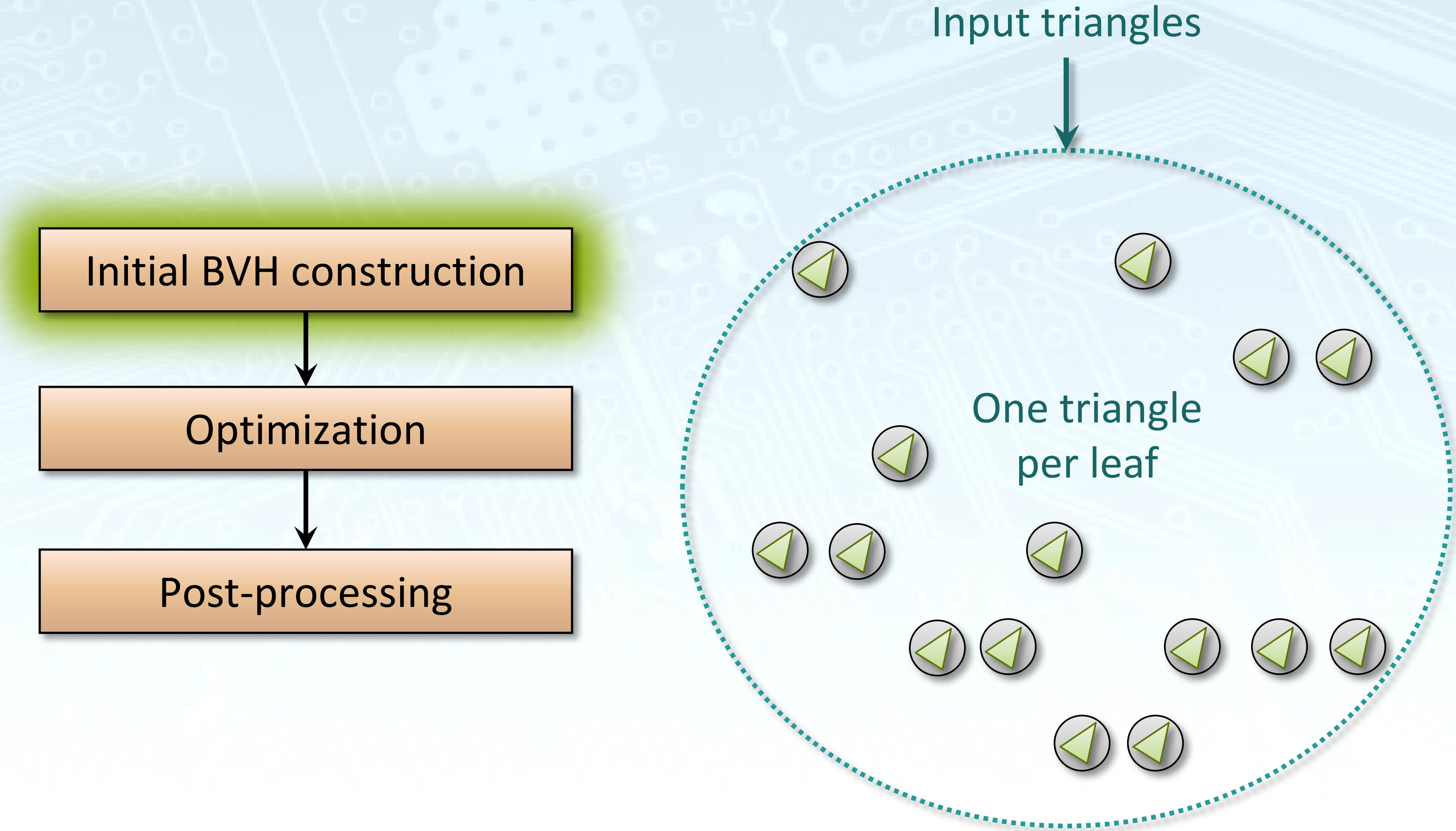


Treelet restructuring

- Restructuring
  - Construct optimal binary tree for the same set of leaves
  - Replace old treelet
- Reuse the same nodes
  - Update connectivity and AABBs
  - New AABBs should be smaller
- Perfectly localized operation
  - Leaves and their subtrees are kept intact
  - No need to look at subtree contents



Processing stages



Processing stages

Initial BVH construction

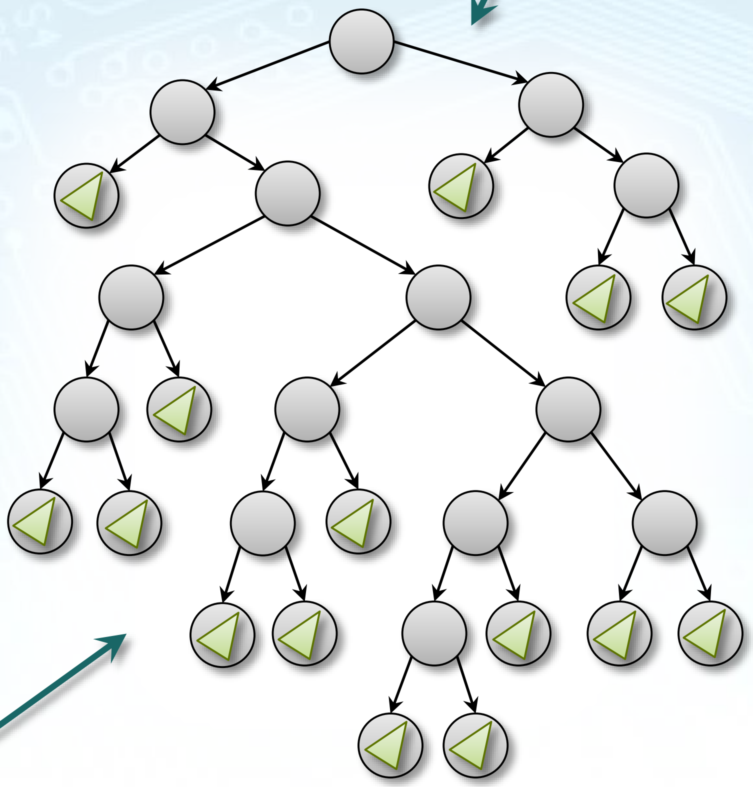
↓

Optimization

↓

Post-processing

Parallel LBVH
[Karras 2012]



60-bit Morton codes
for accurate spatial
partitioning

Processing stages

Initial BVH construction

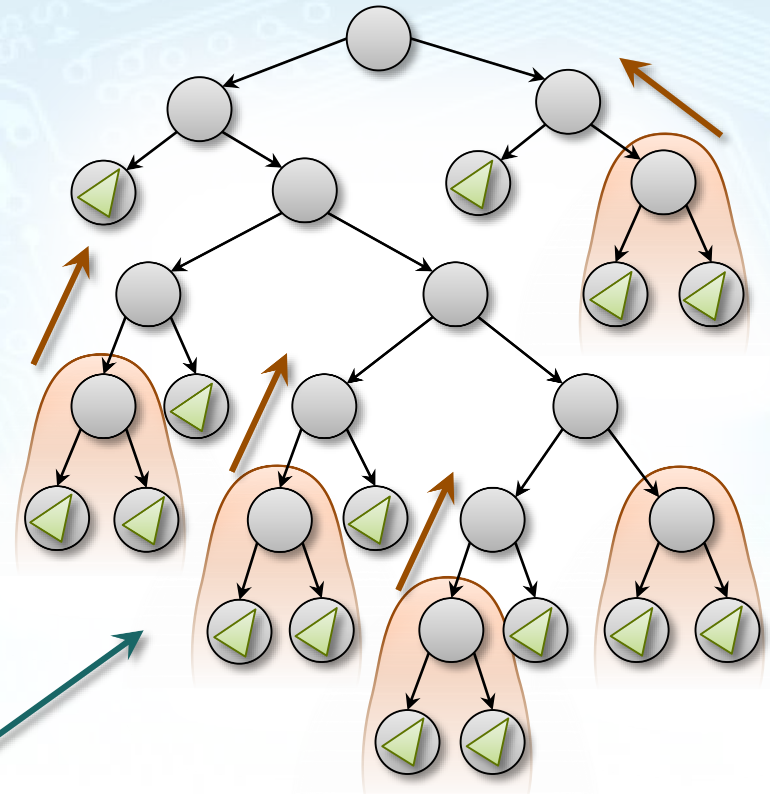
↓

Optimization

↓

Post-processing

Parallel bottom-up traversal
[Karras 2012]



Restructure multiple
treelets in parallel

Processing stages

Initial BVH construction

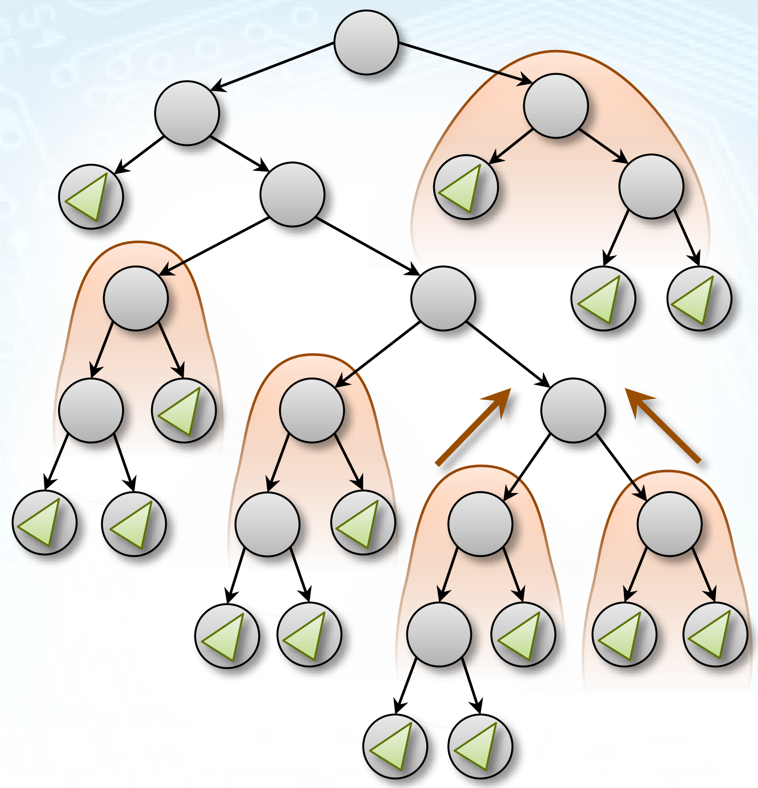
↓

Optimization

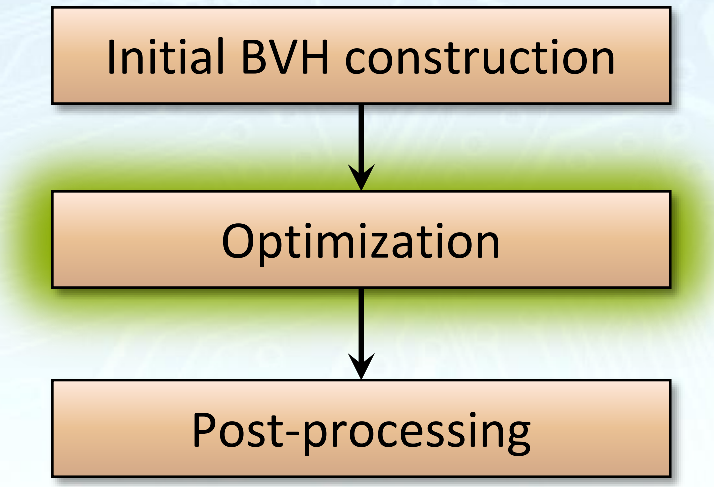
↓

Post-processing

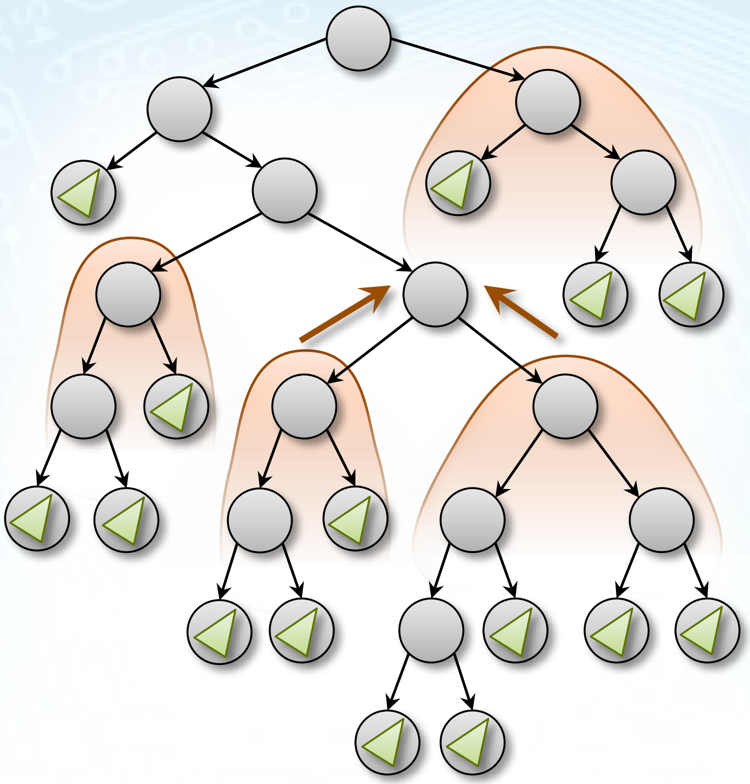
Parallel bottom-up traversal
[Karras 2012]



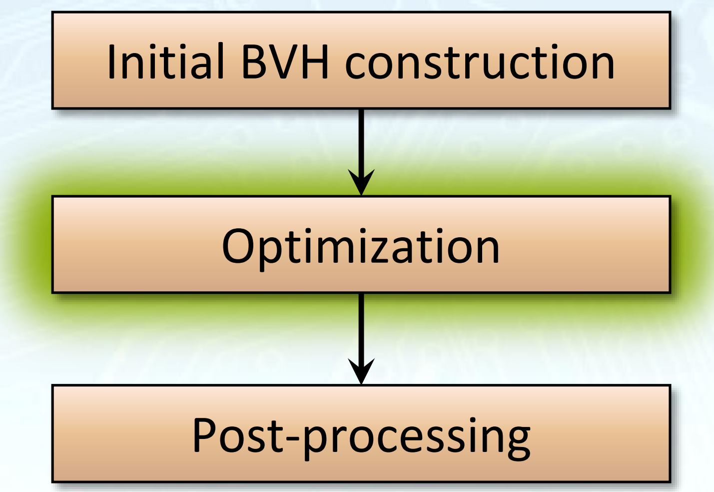
Processing stages



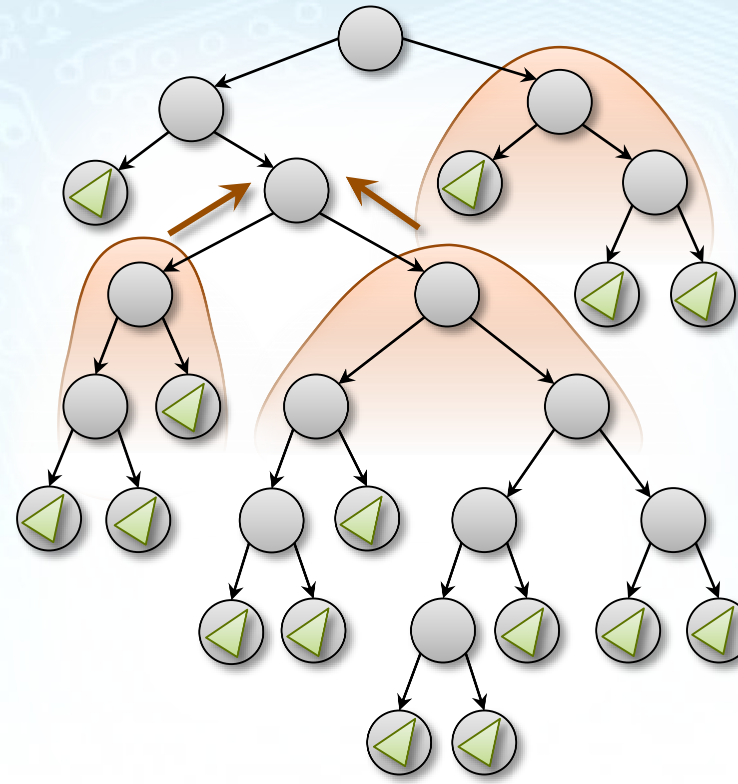
Parallel bottom-up traversal
[Karras 2012]



Processing stages



Parallel bottom-up traversal
[Karras 2012]



Processing stages

Initial BVH construction

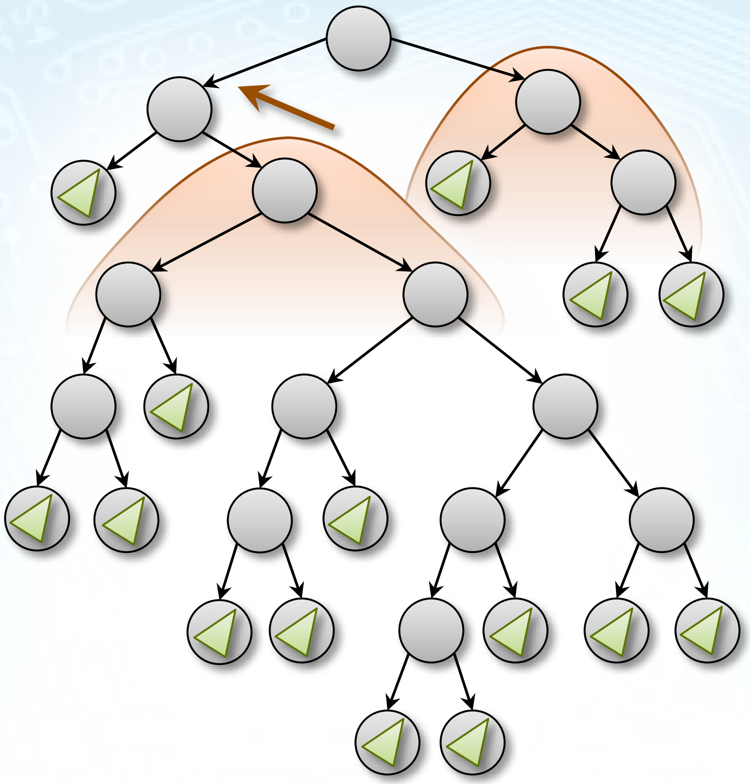
↓

Optimization

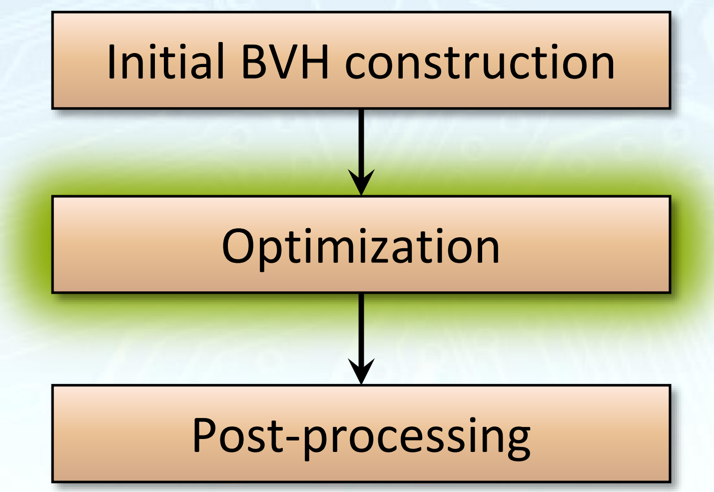
↓

Post-processing

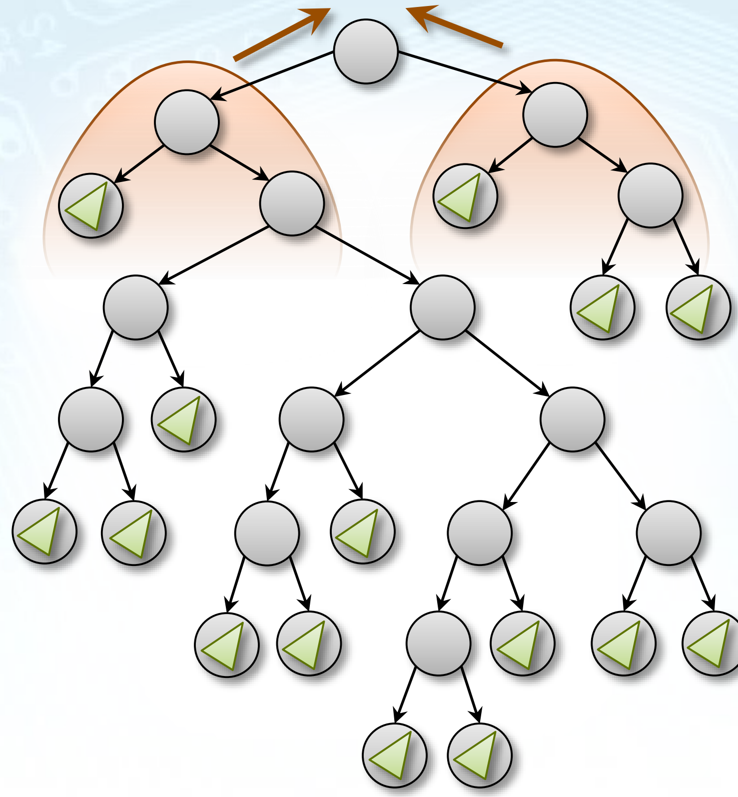
Parallel bottom-up traversal
[Karras 2012]



Processing stages



Parallel bottom-up traversal
[Karras 2012]



Processing stages

Initial BVH construction

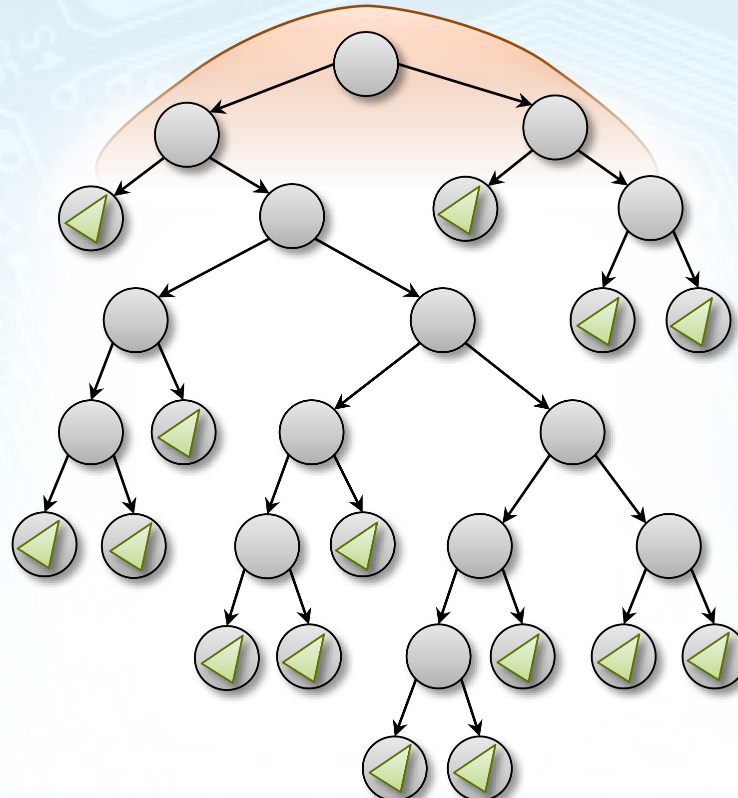
↓

Optimization

↓

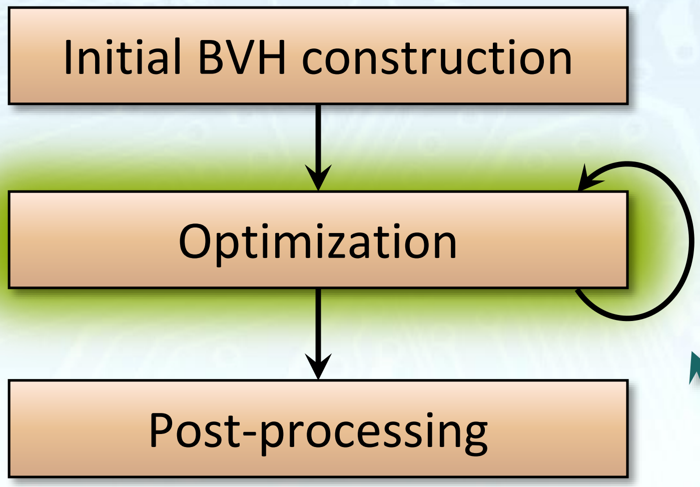
Post-processing

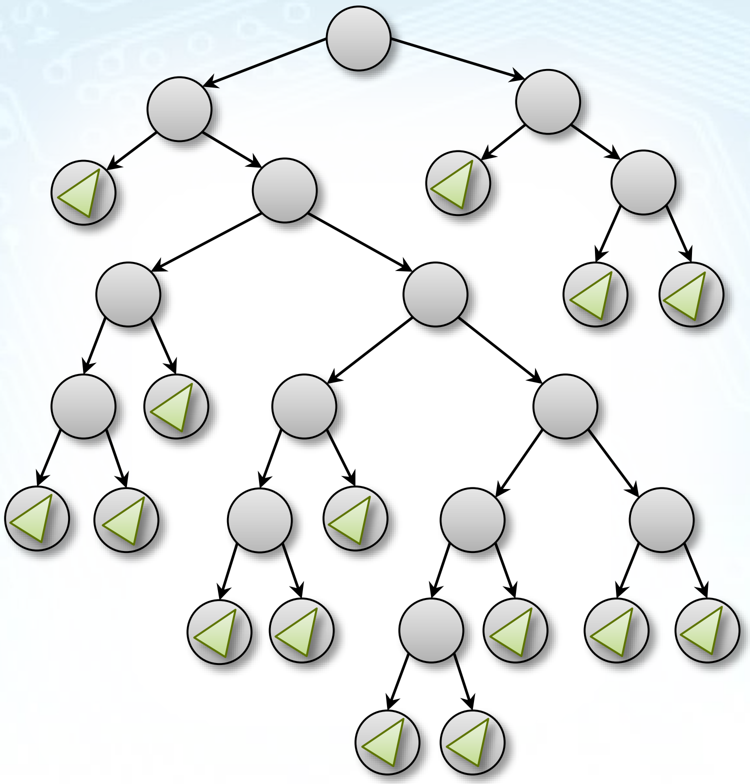
Parallel bottom-up traversal
[Karras 2012]



Strict bottom-up order
→ no overlap between treelets

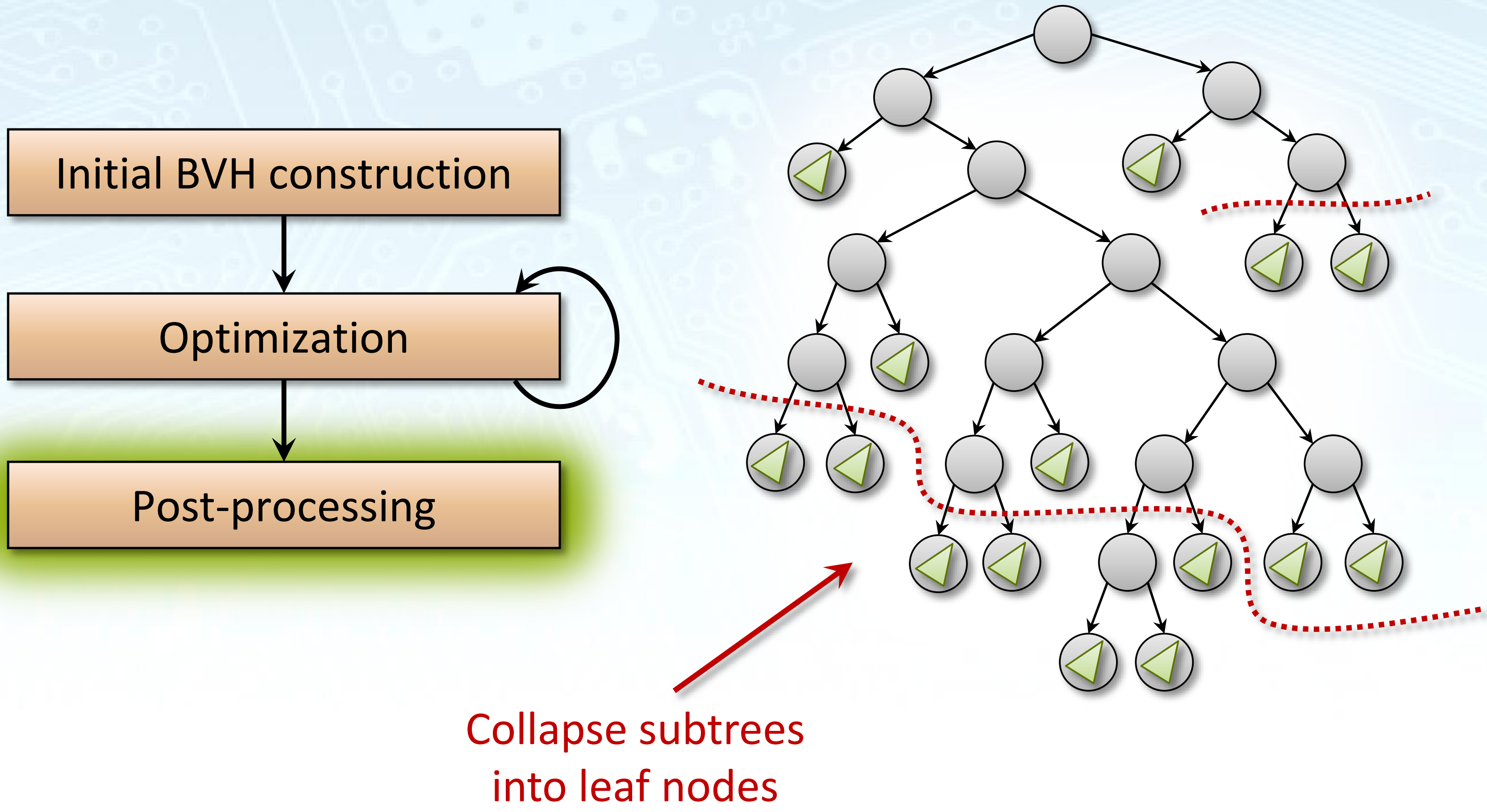
Processing stages



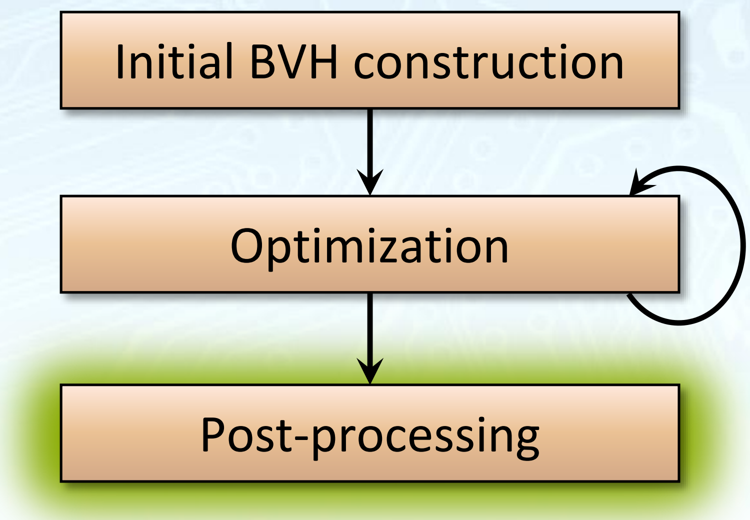


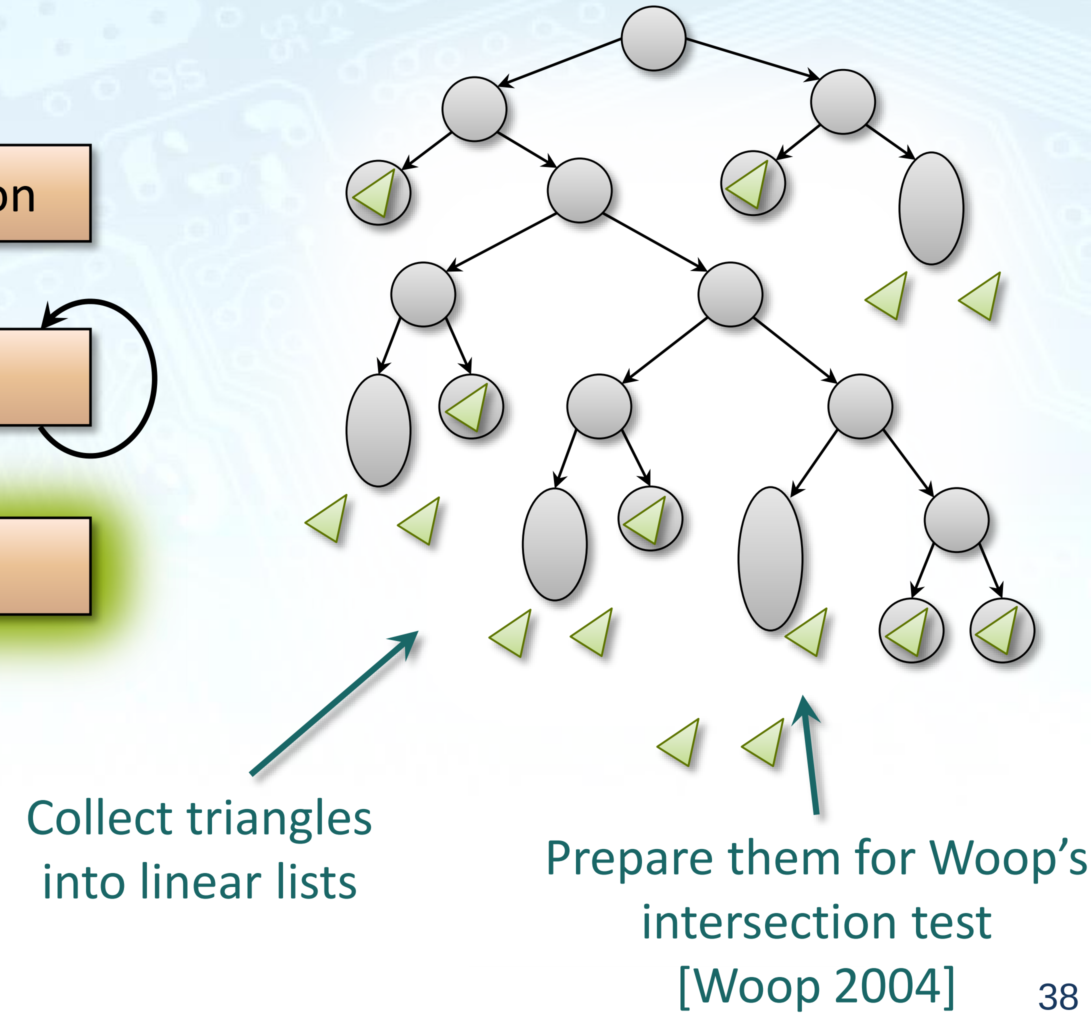
Rinse and repeat
(3 times is plenty)

Processing stages



Processing stages





Processing stages

Initial BVH construction

↓

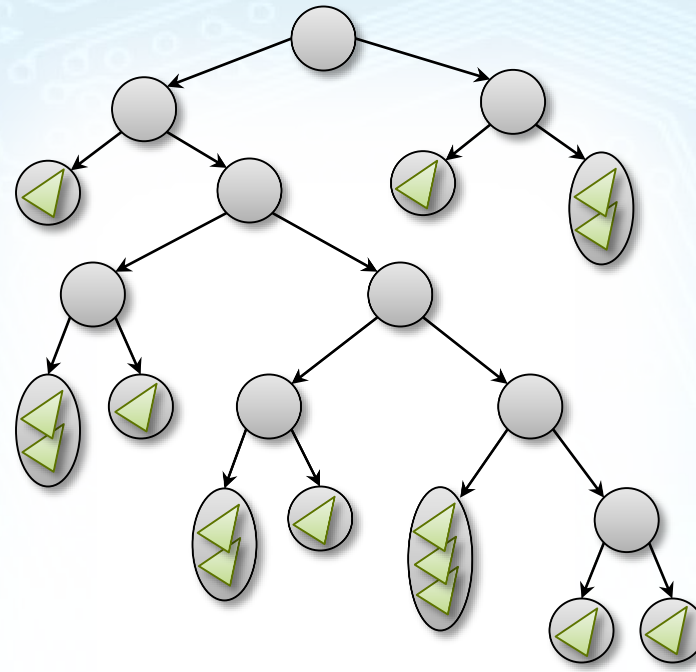
Optimization

↓

Post-processing

↓

Fast GPU ray traversal
[Aila et al. 2012]



Processing stages

Triangle splitting

↓

Initial BVH construction

↓

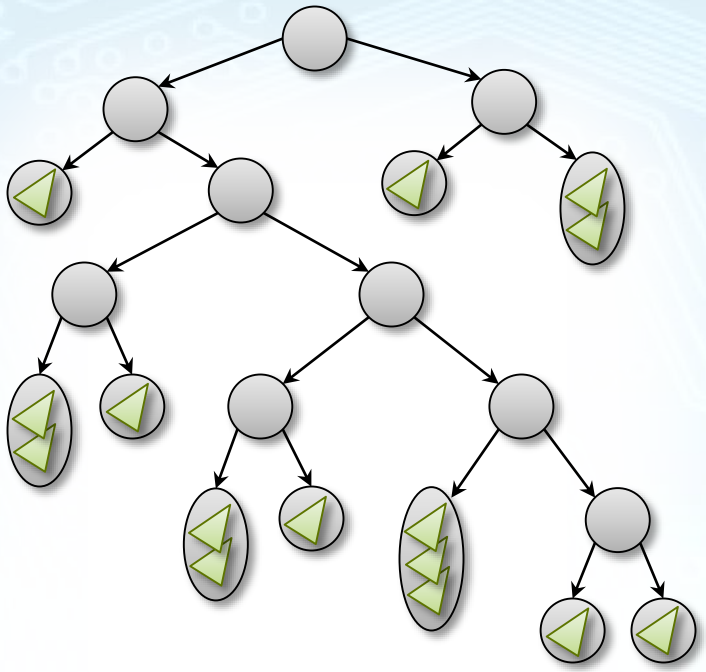
Optimization

↓

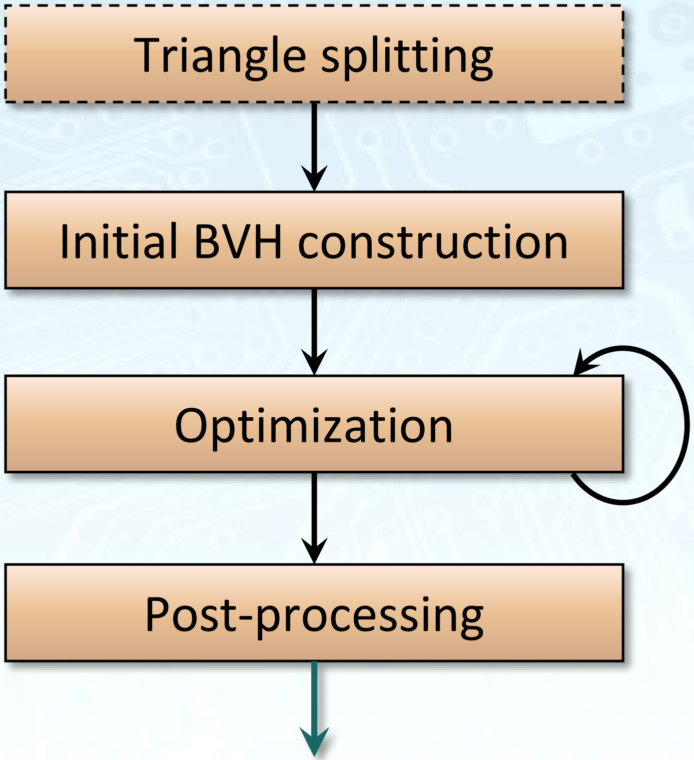
Post-processing

↓

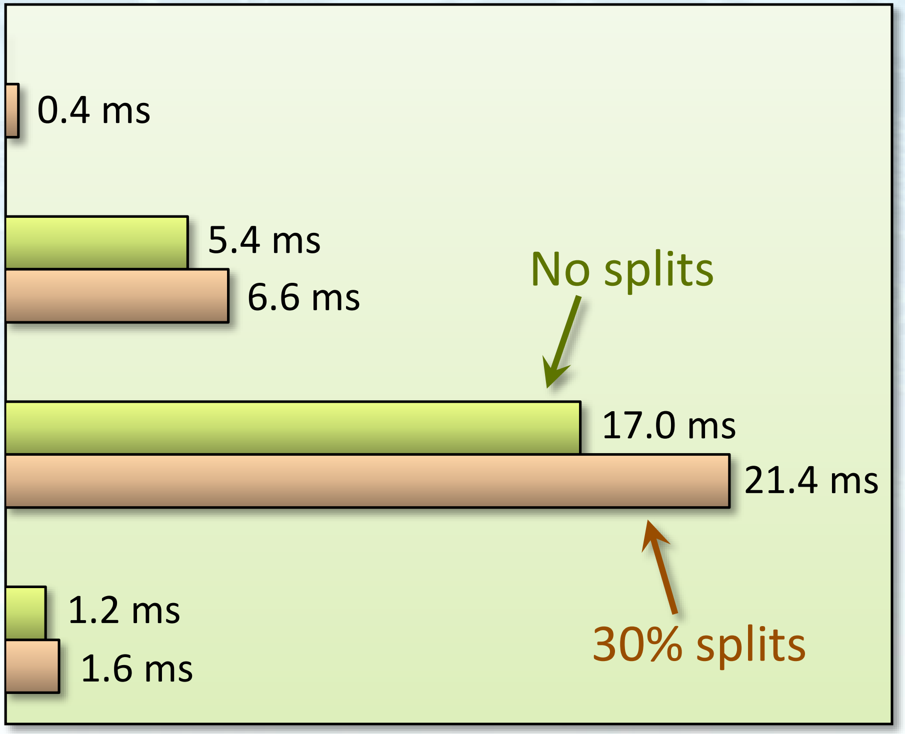
Fast GPU ray traversal
[Aila et al. 2012]



Processing stages



Fast GPU ray traversal
[Aila et al. 2012]



DRAGON (870K tris)
NVIDIA GTX Titan
23.6 ms / 30.0 ms

Cost model

- Surface area cost model

  [Goldsmith and Salmon 1987], [MacDonald and Booth 1990]

$$SAH := C_i \sum_{n \in I} \frac{A(n)}{A(\mathrm{root})} + C_t \sum_{l \in L} \frac{A(l)}{A(\mathrm{root})} N(l)$$

- Track cost and triangle count of each subtree

- Minimize SAH cost of the *final* BVH
  - Make collapsing decisions already during optimization

    → Unified processing of leaves and internal nodes

Optimal restructuring

- Finding the optimal node topology is NP-hard
  - Naive algorithm → $\mathcal{O}(n!)$
  - Our approach → $\mathcal{O}(3^n)$
- But it becomes very powerful as n grows
  - $n = 7$ treelet leaves is enough for high-quality results
- Use fixed-size treelets
  - Constant cost per treelet
    → Linear with respect to scene size

Optimal restructuring

* SODA (2.2M tris)

Treelet size	Layouts	Quality vs. SBVH *
4	15	78%
5	105	85%
6	945	88%
7	10,395	97%
8	135,135	98%

Number of unique ways for restructuring a given treelet

Ray tracing performance after 3 rounds of optimization

Optimal restructuring

* SODA (2.2M tris)

Almost the same thing as tree rotations [Kensler 2008]

Treelet size	Layouts	Quality vs. SBVH *
4	15	78%
5	105	85%
6	945	88%
7	10,395	97%
8	135,135	98%

Varies a lot between scenes

Limited options during optimization
→ easy to get stuck in a local optimum

Optimal restructuring

* SODA (2.2M tris)

Treelet size	Layouts	Quality vs. SBVH *
4	15	78%
5	105	85%
6	945	88%
7	10,395	97%
8	135,135	98%

Can still be implemented efficiently

Consistent across scenes

Surely one of these will take us forward ☺

Further improvement is marginal

Algorithm

- Dynamic programming
  - Solve small subproblems first
  - Tabulate their solutions
  - Build on them to solve larger subproblems
- Subproblem:
  - What's the best node topology for a *subset* of the leaves?

Algorithm

```
input: set of n treelet leaves
for k = 2 to n do
    for each subset of size k do
        for each way of partitioning the leaves do
            look up subtree costs
            calculate SAH cost
        end for
        record the best solution
    end for
end for
reconstruct optimal topology
```

Process subsets from smallest to largest

Record the optimal SAH cost for each

Algorithm

```
input: set of n treelet leaves
for k = 2 to n do
    for each subset of size k do
        for each way of partitioning the leaves do
            look up subtree costs
            calculate SAH cost
        end for
        record the best solution
    end for
end for
reconstruct optimal topology
```

Exhaustive search: assign each leaf to left/right subtree

We already know how much the subtrees will cost

Backtrack the partitioning choices

Scalar vs. SIMD

Scalar processing

- Each thread processes one treelet
- Need many treelets in flight

✗ Spills to off-chip memory
✗ Doesn't scale to small scenes
✓ Trivial to implement

SIMD processing

- 32 threads collaborate on the same treelet
- Need few treelets in flight

✓ Data fits in on-chip memory
✓ Easy to fill the entire GPU
✗ Need to keep all threads busy

Parallelize over subproblems using a pre-optimized processing schedule (details in the paper)

Scalar vs. SIMD

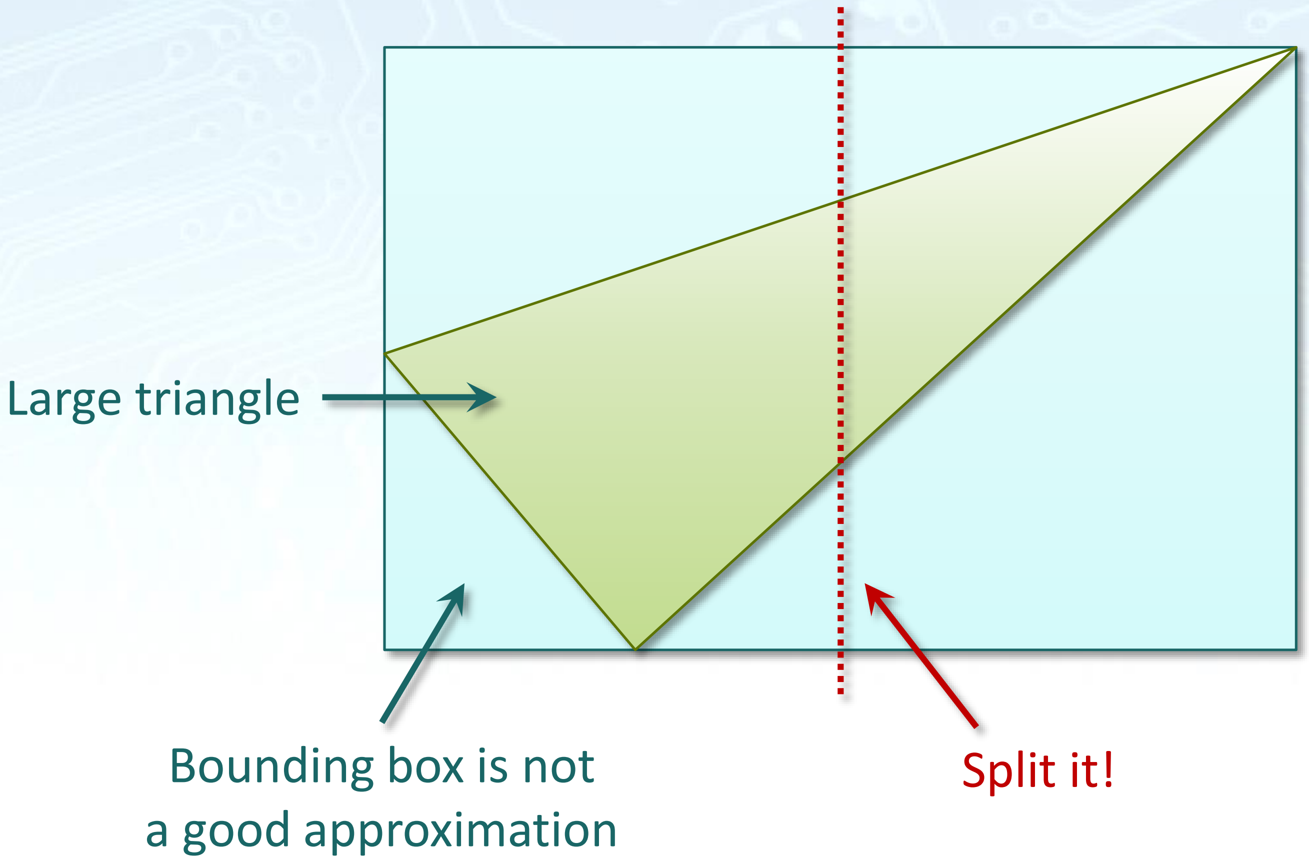
Scalar processing	SIMD processing
• Each thread processes one treelet • Need many treelets in flight	• 32 threads collaborate on the same treelet • Need few treelets in flight
✗ Spills to off-chip memory ✗ Doesn’t scale to small scenes ✓ Trivial to implement	✓ Data fits in on-chip memory ✓ Easy to fill the entire GPU ✓ Possible to keep threads busy

Quality vs. speed

- Spend less effort on bottom-most nodes
  - Low contribution to SAH cost
  - Quick convergence
- Additional parameter γ
  - Only process subtrees that are large enough
  - Trade quality for speed
- Double γ after each round
  - Significant speedup
  - Negligible effect on quality

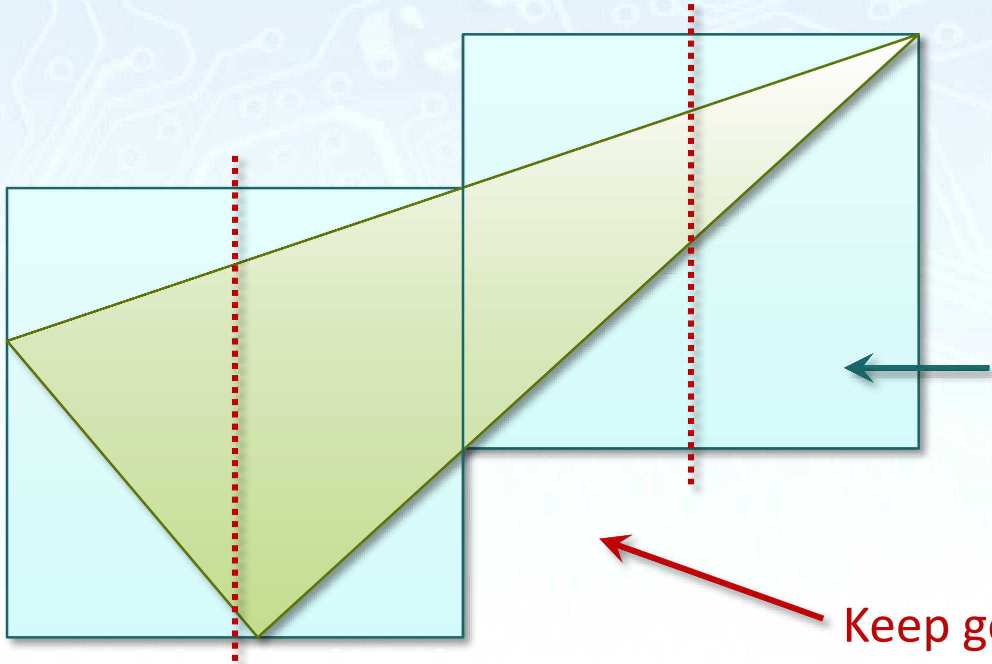
Triangle splitting

- Early Split Clipping [Ernst and Greiner 2007]
  - Split triangle bounding boxes as a pre-process



Triangle splitting

- Early Split Clipping [Ernst and Greiner 2007]
  - Split triangle bounding boxes as a pre-process

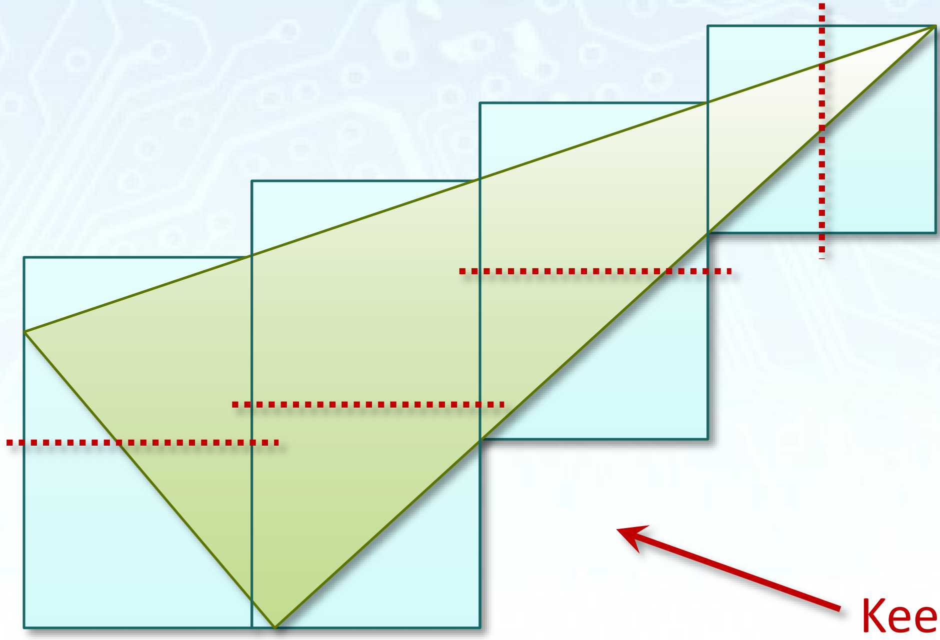


Resulting boxes provide a tighter bound

Keep going until they are small enough

Triangle splitting

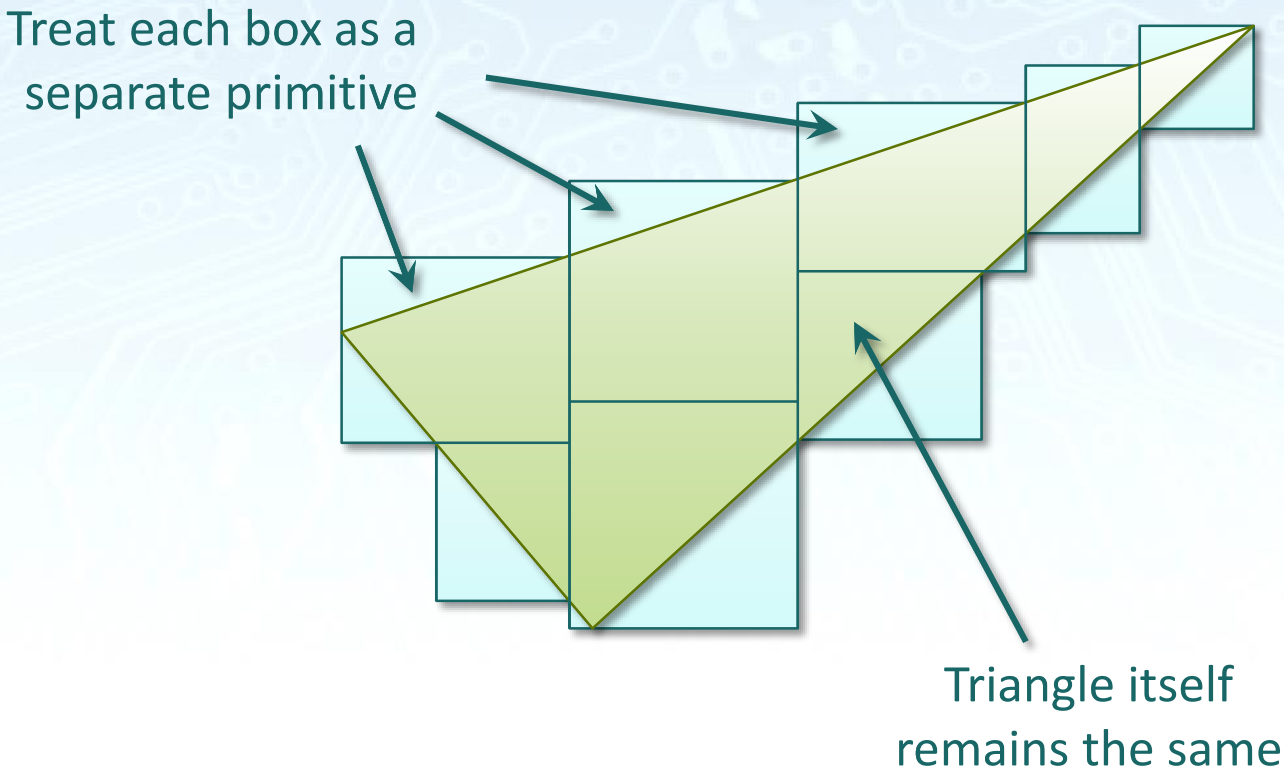
- Early Split Clipping [Ernst and Greiner 2007]
  - Split triangle bounding boxes as a pre-process



Keep going until they are small enough

Triangle splitting

- Early Split Clipping [Ernst and Greiner 2007]
  - Split triangle bounding boxes as a pre-process

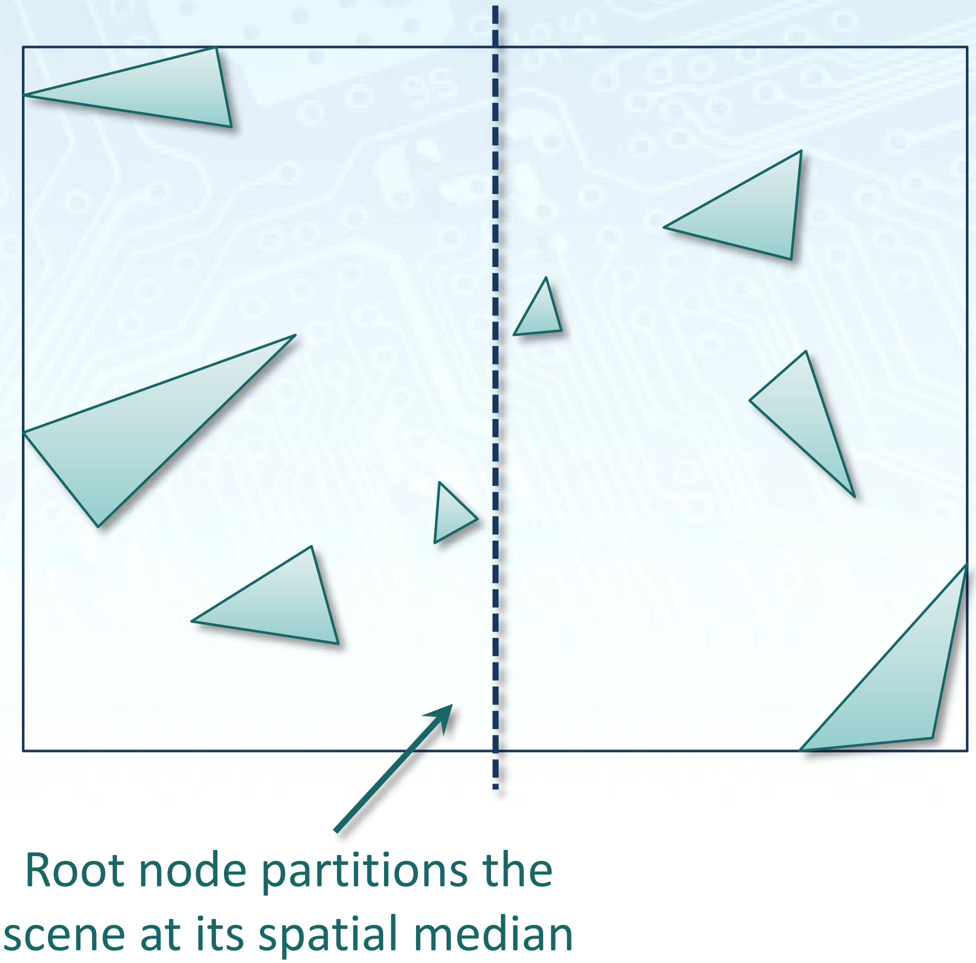


Triangle splitting

- Shortcomings of pre-process splitting
  - Can hurt ray tracing performance
  - Unpredictable memory usage
  - Requires manual tuning
- Improve with better heuristics
  - Select good split planes
  - Concentrate splits where they matter
  - Use a fixed split budget

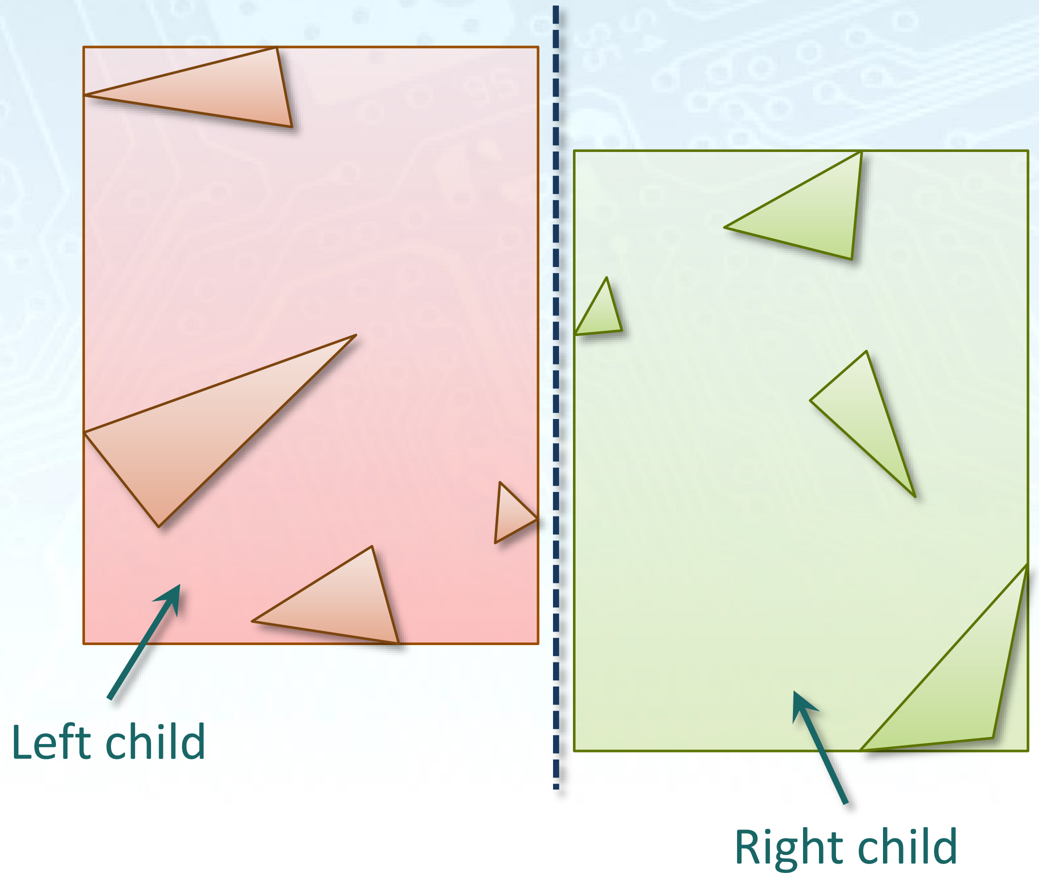
Split plane selection

- Reduce node overlap in the initial BVH



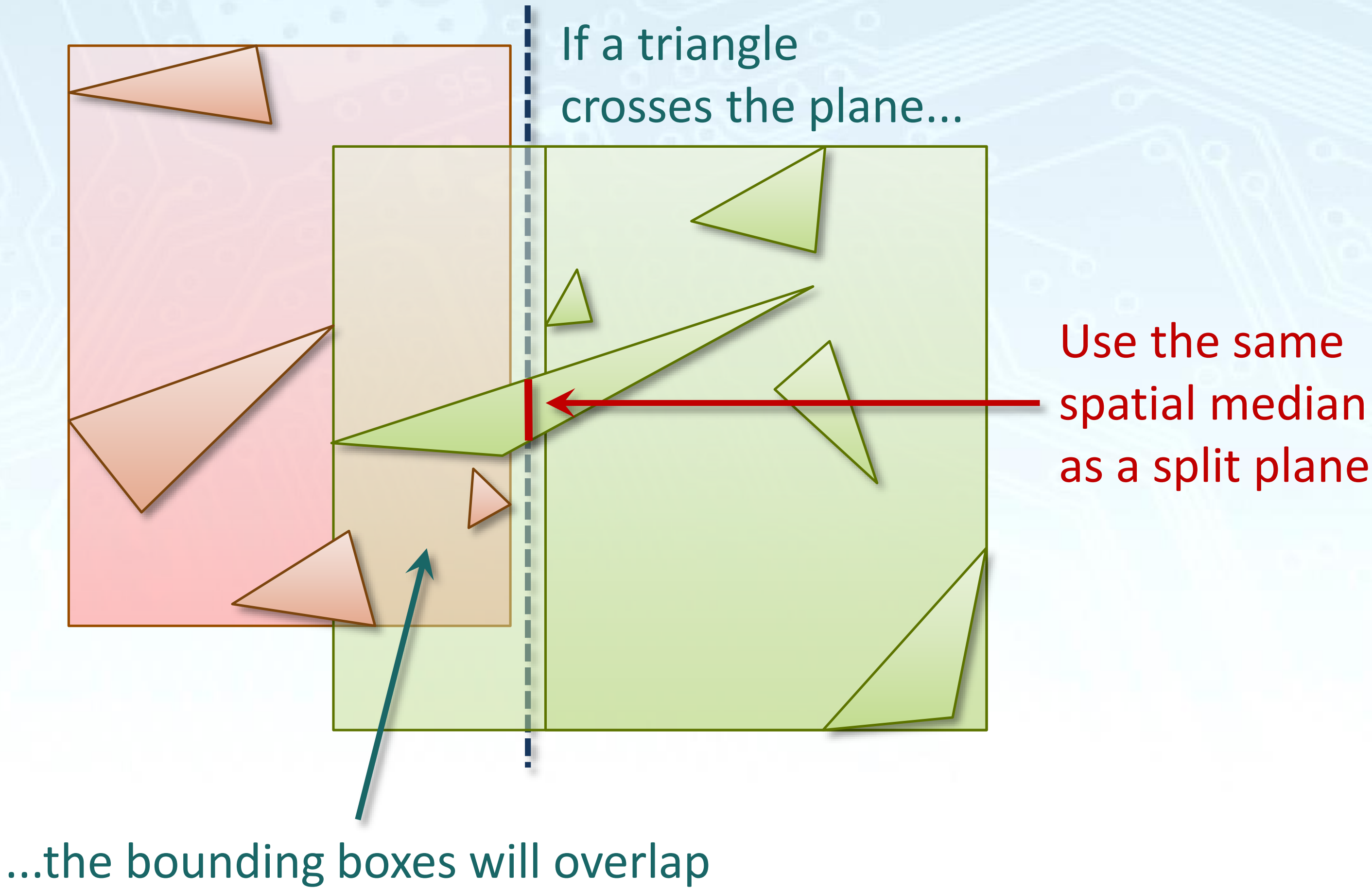
Split plane selection

- Reduce node overlap in the initial BVH



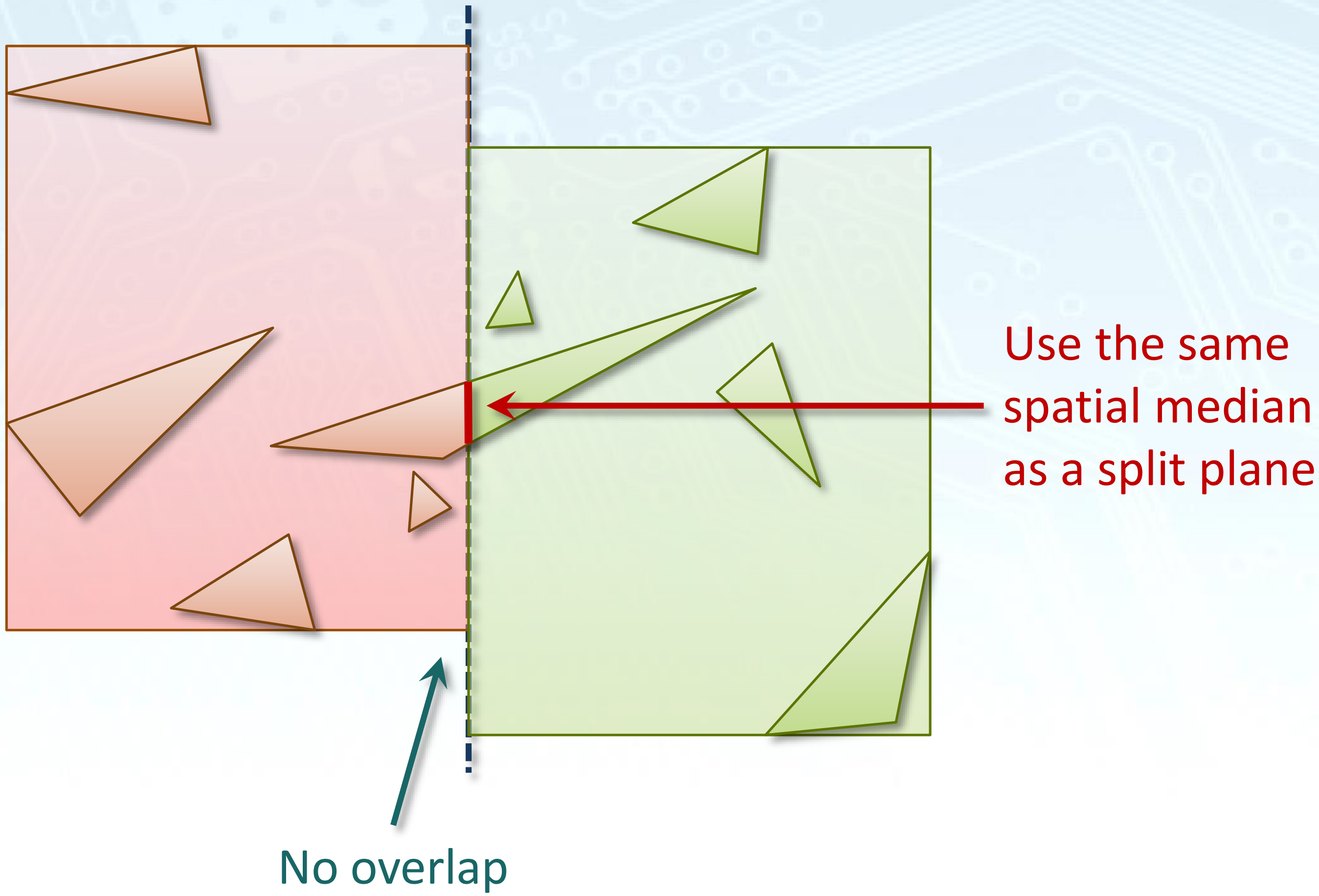
Split plane selection

- Reduce node overlap in the initial BVH



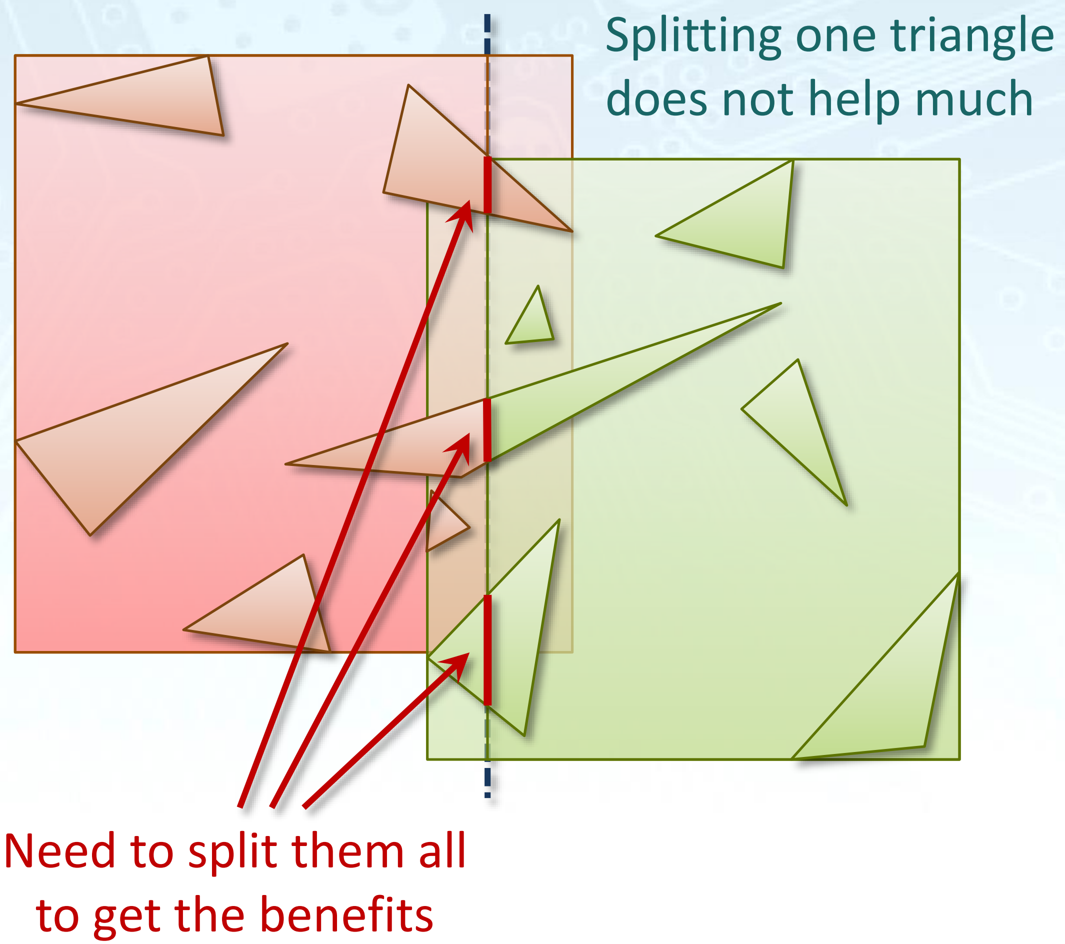
Split plane selection

- Reduce node overlap in the initial BVH



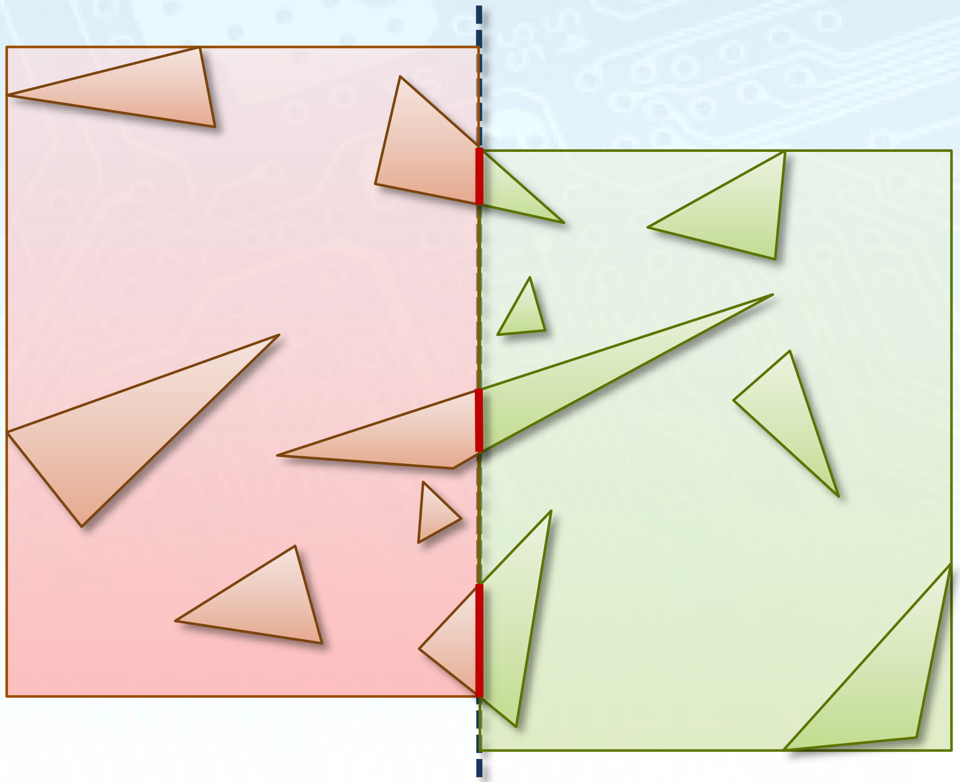
Split plane selection

- Reduce node overlap in the initial BVH



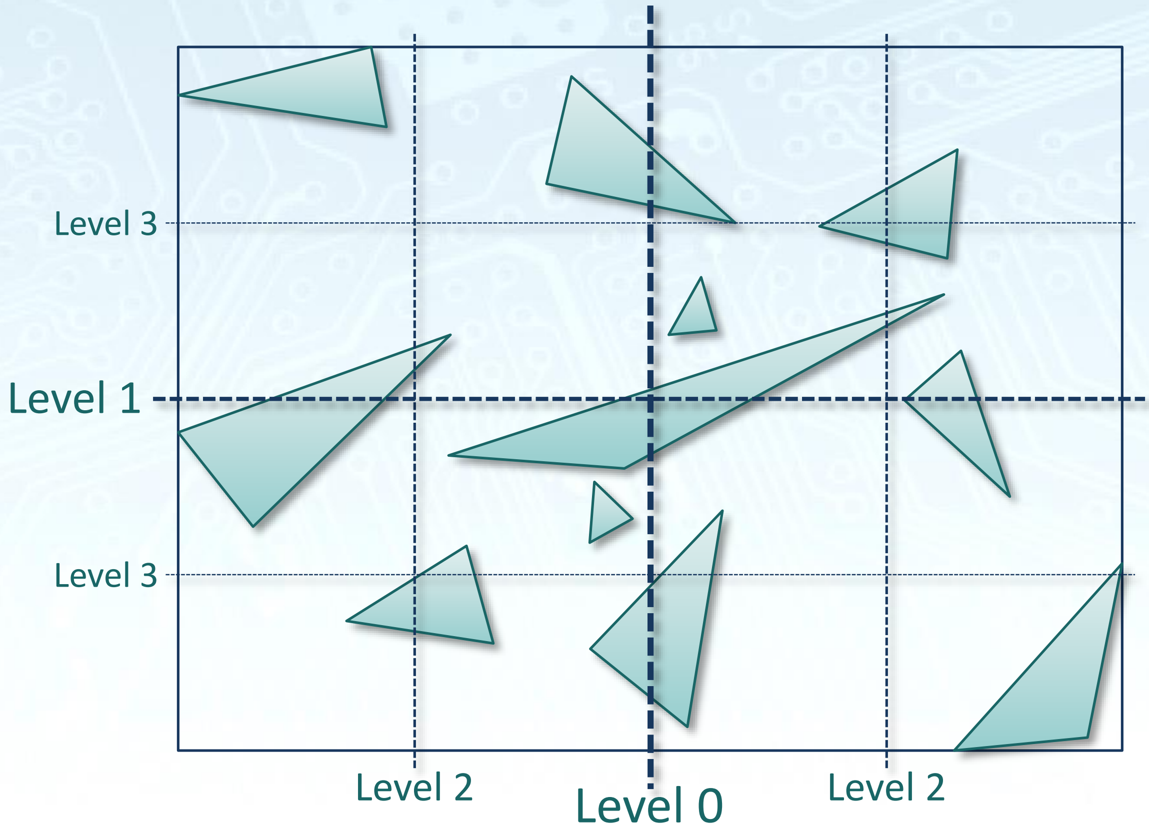
Split plane selection

- Reduce node overlap in the initial BVH



Split plane selection

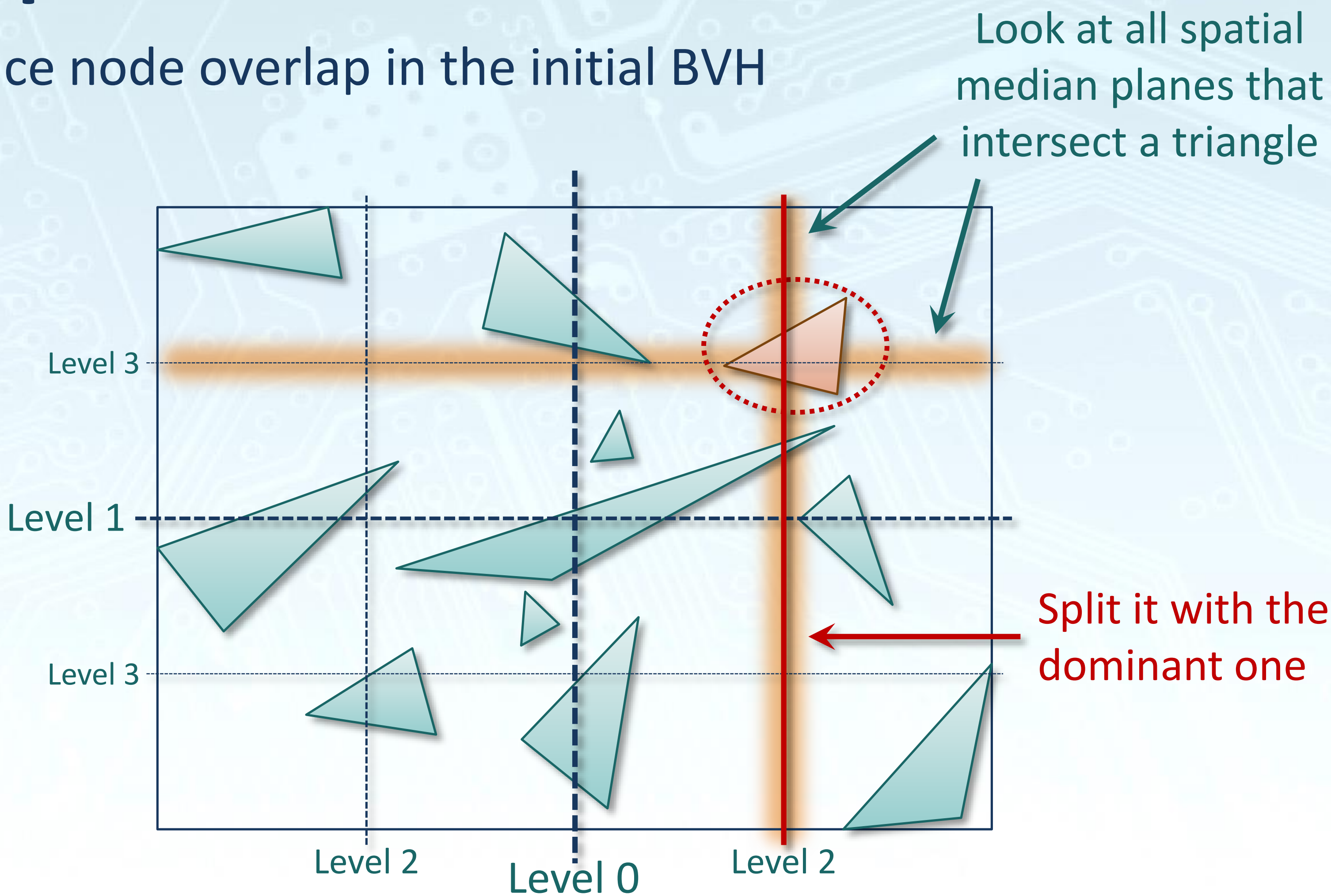
- Reduce node overlap in the initial BVH



Same reasoning holds on multiple levels

Split plane selection

- Reduce node overlap in the initial BVH



Algorithm

1. Allocate memory for a fixed split budget

Algorithm

1. Allocate memory for a fixed split budget

2. Calculate a *priority value* for each triangle

Algorithm

1. Allocate memory for a fixed split budget
2. Calculate a *priority value* for each triangle
3. Distribute the split budget among triangles
   - Proportional to their priority values

Algorithm

1. Allocate memory for a fixed split budget

2. Calculate a *priority value* for each triangle

3. Distribute the split budget among triangles
   - Proportional to their priority values

4. Split each triangle recursively
   - Distribute remaining splits according to the size of the resulting AABBs

Split priority

$$priority = \left(\boxed{2^{(-level)}} \cdot \boxed{(A_{aabb} - A_{ideal})}\right)^{\boxed{1/3}}$$

Crosses an important spatial median plane?

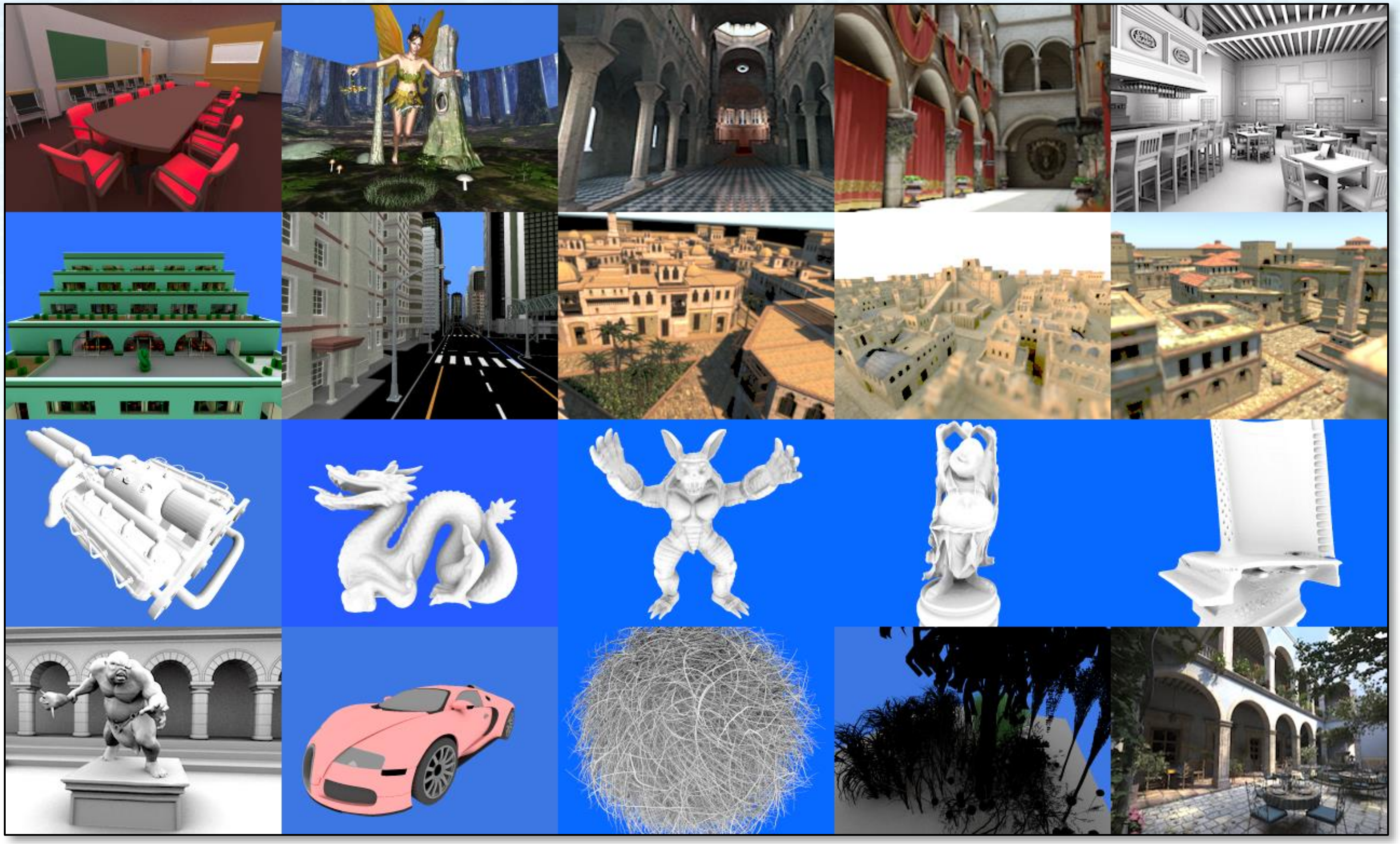
Has large potential for reducing surface area?

Concentrate on triangles where both apply

…but leave something for the rest, too

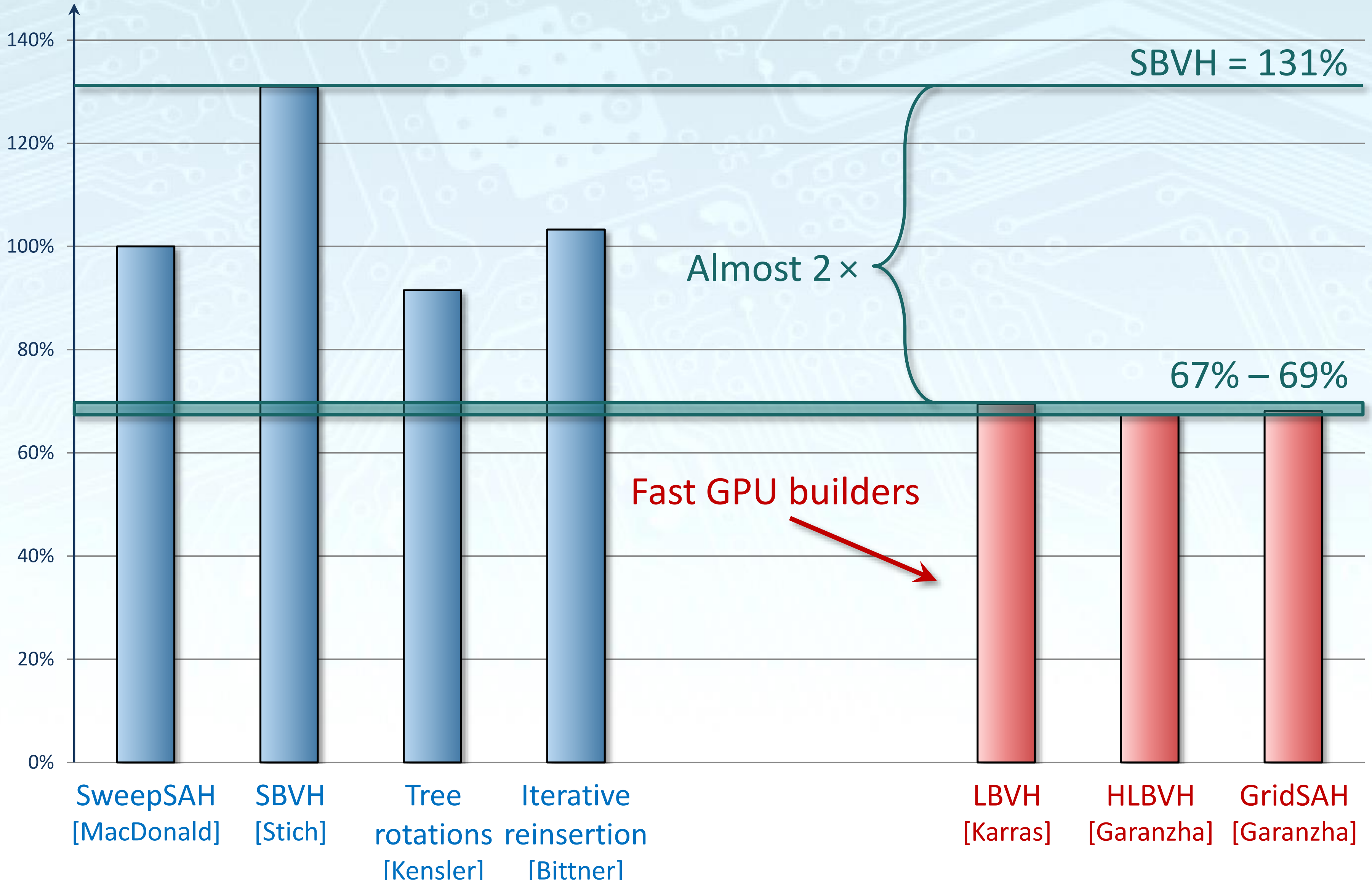
Results

- Compare against 4 CPU and 3 GPU builders
  - 4-core i7 930, NVIDIA GTX Titan
  - Average of 20 test scenes, multiple viewpoints

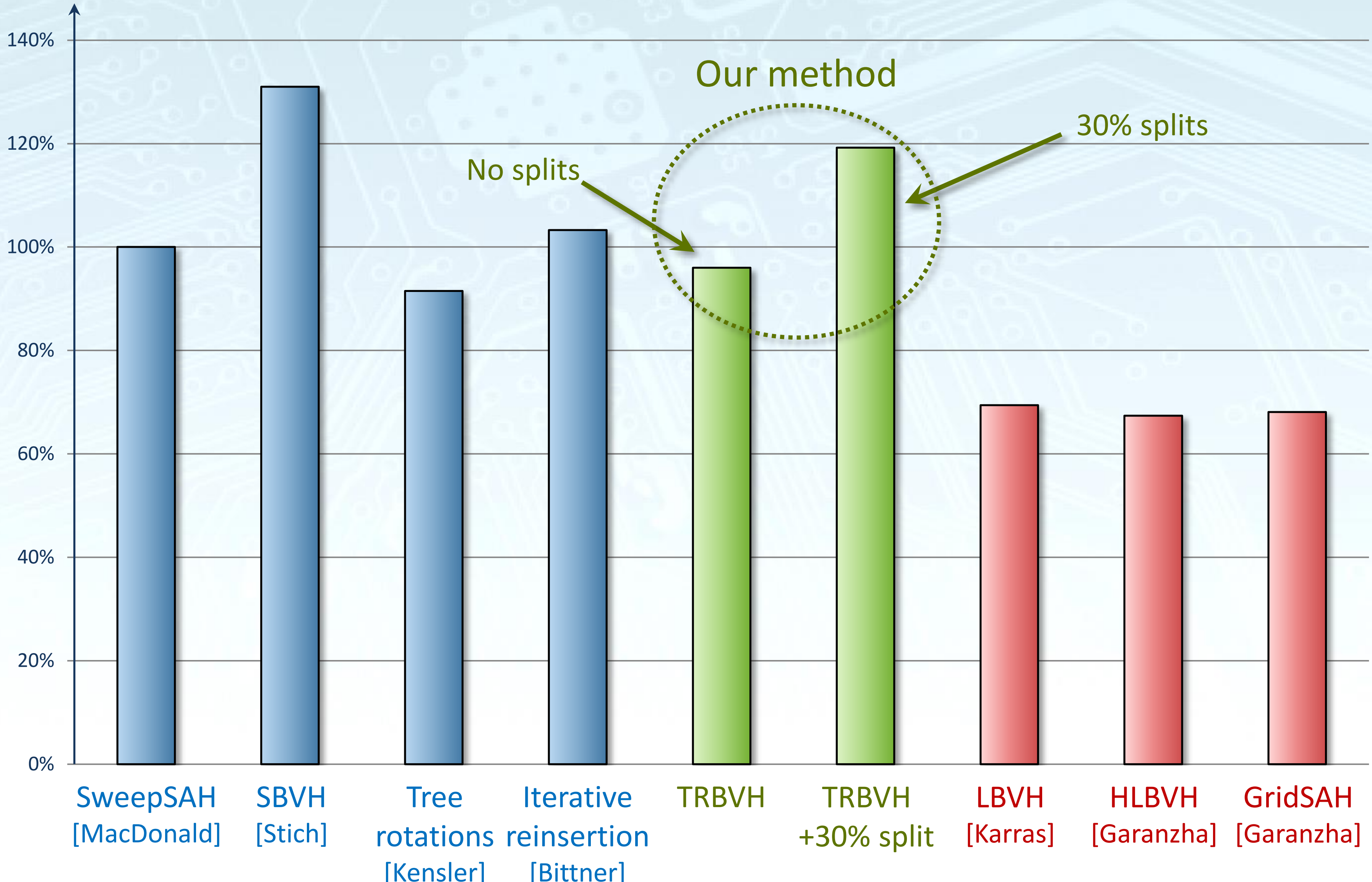


Ray tracing performance

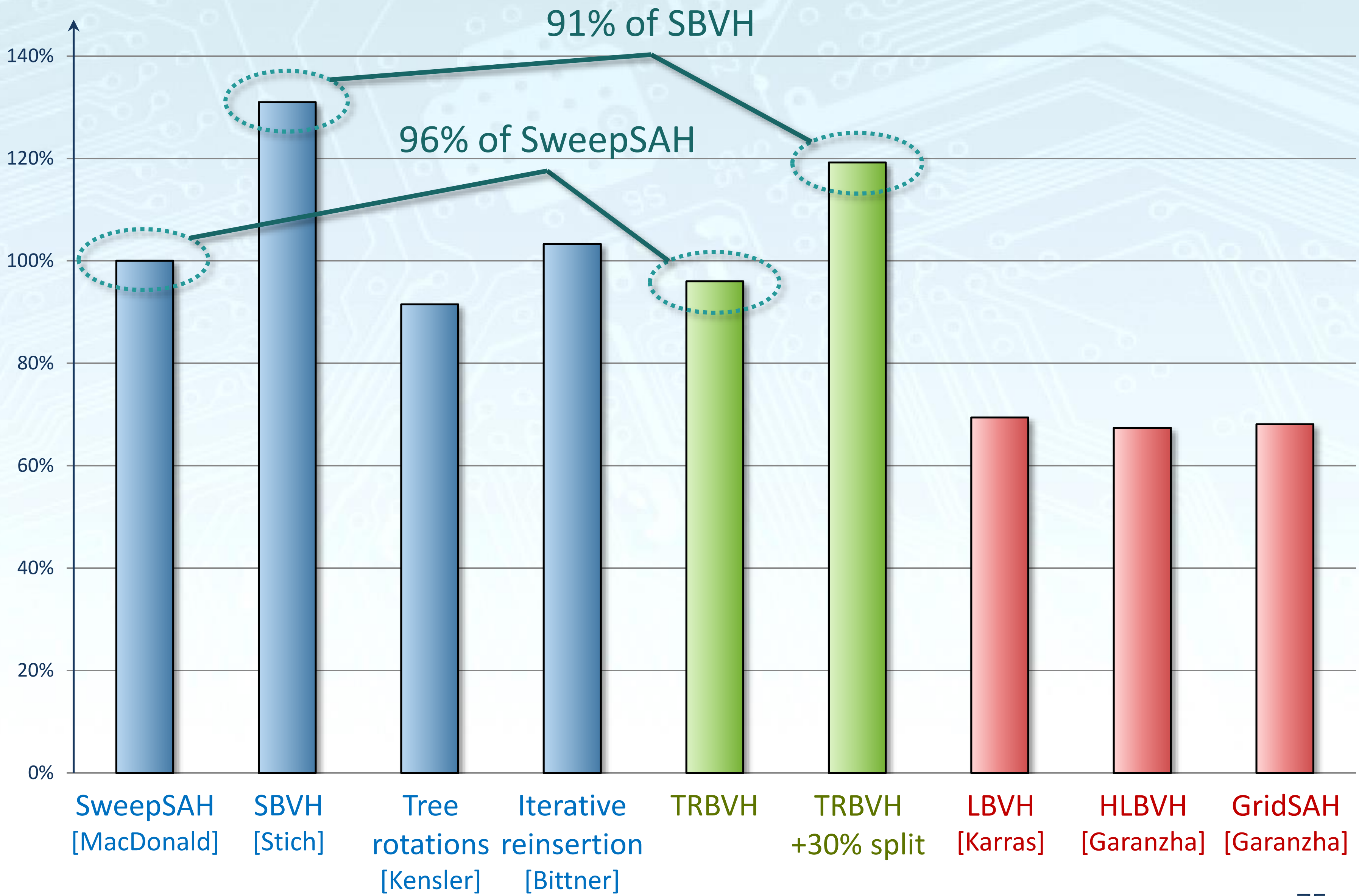
Ray tracing performance



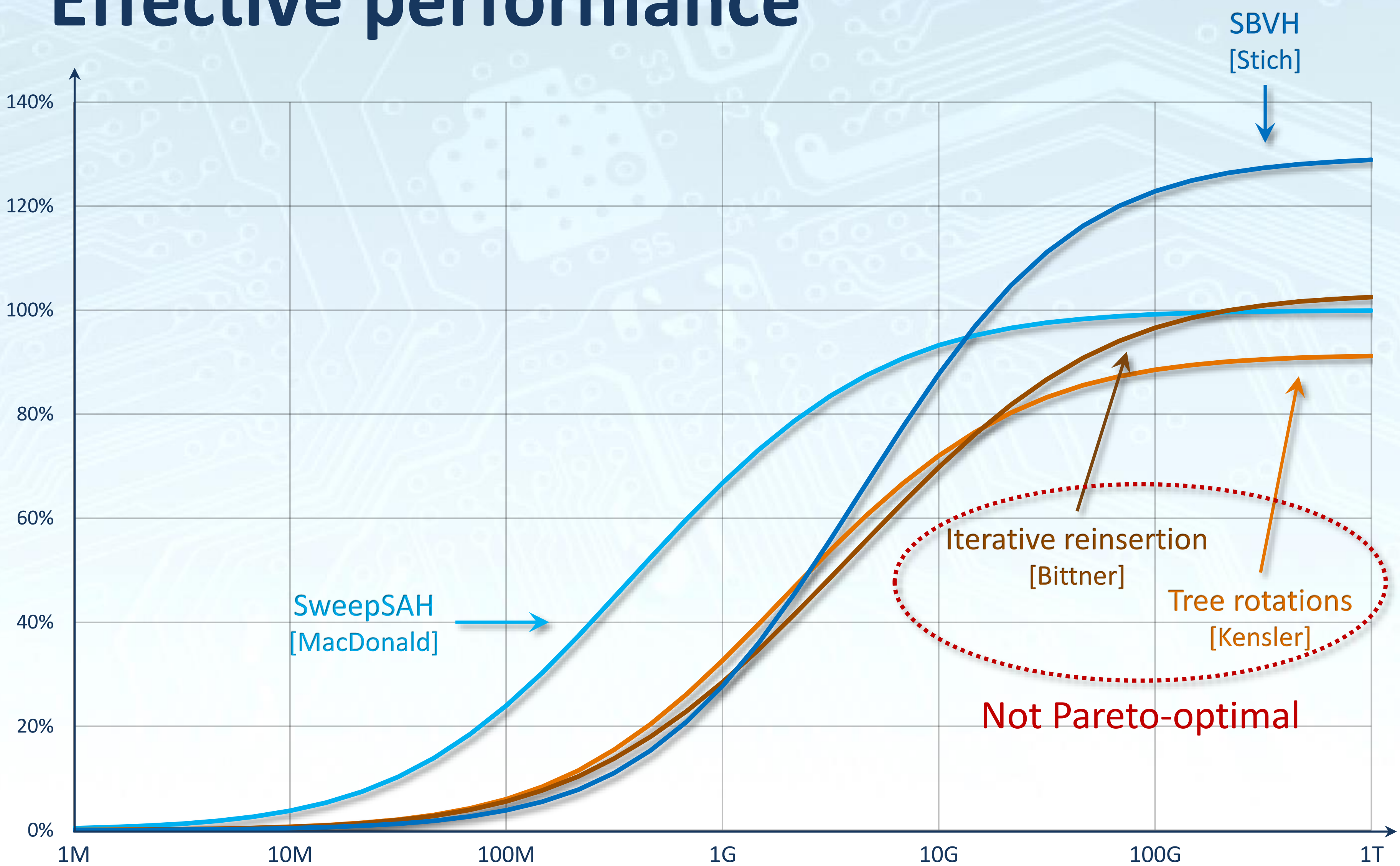
Ray tracing performance



Ray tracing performance

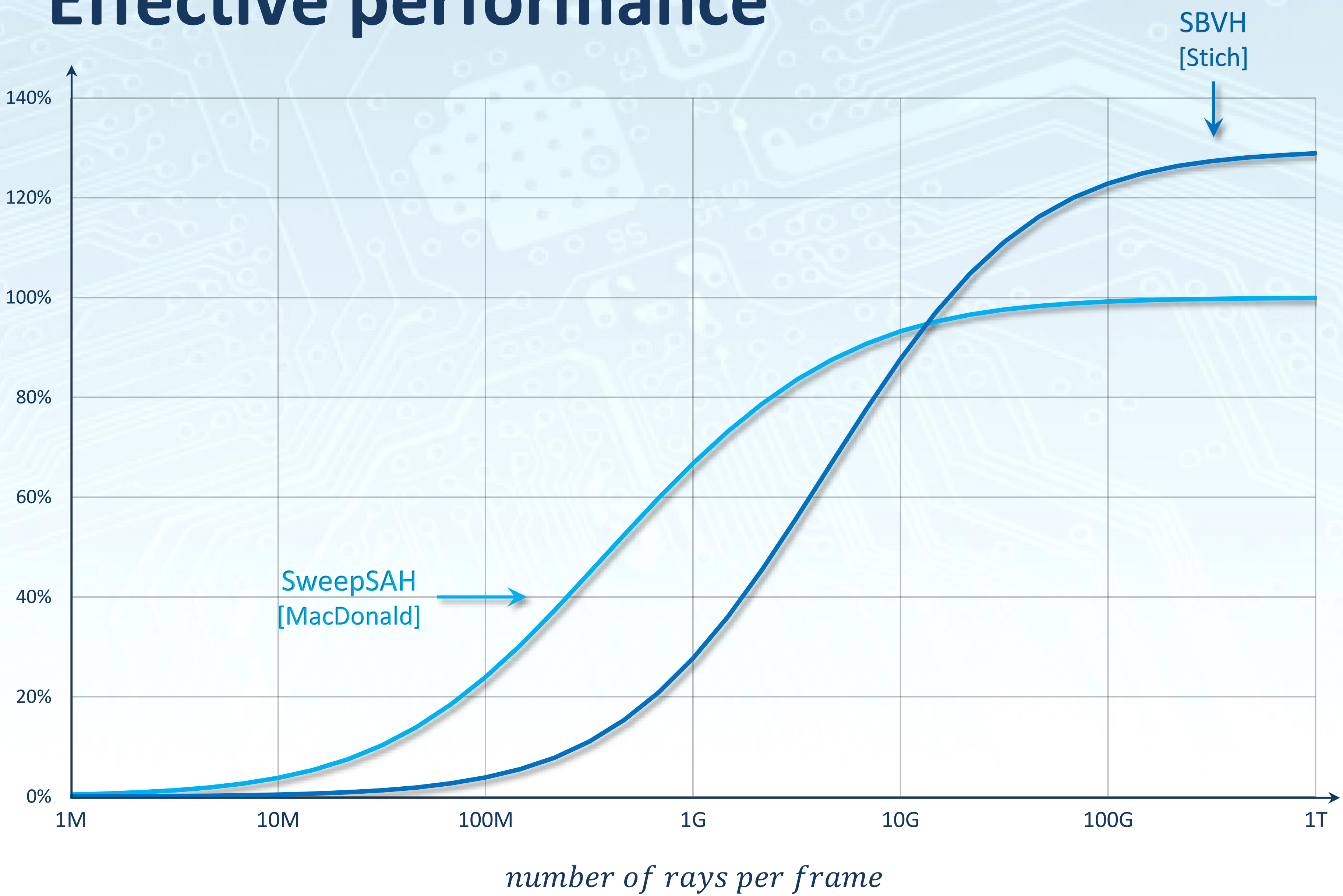


Effective performance

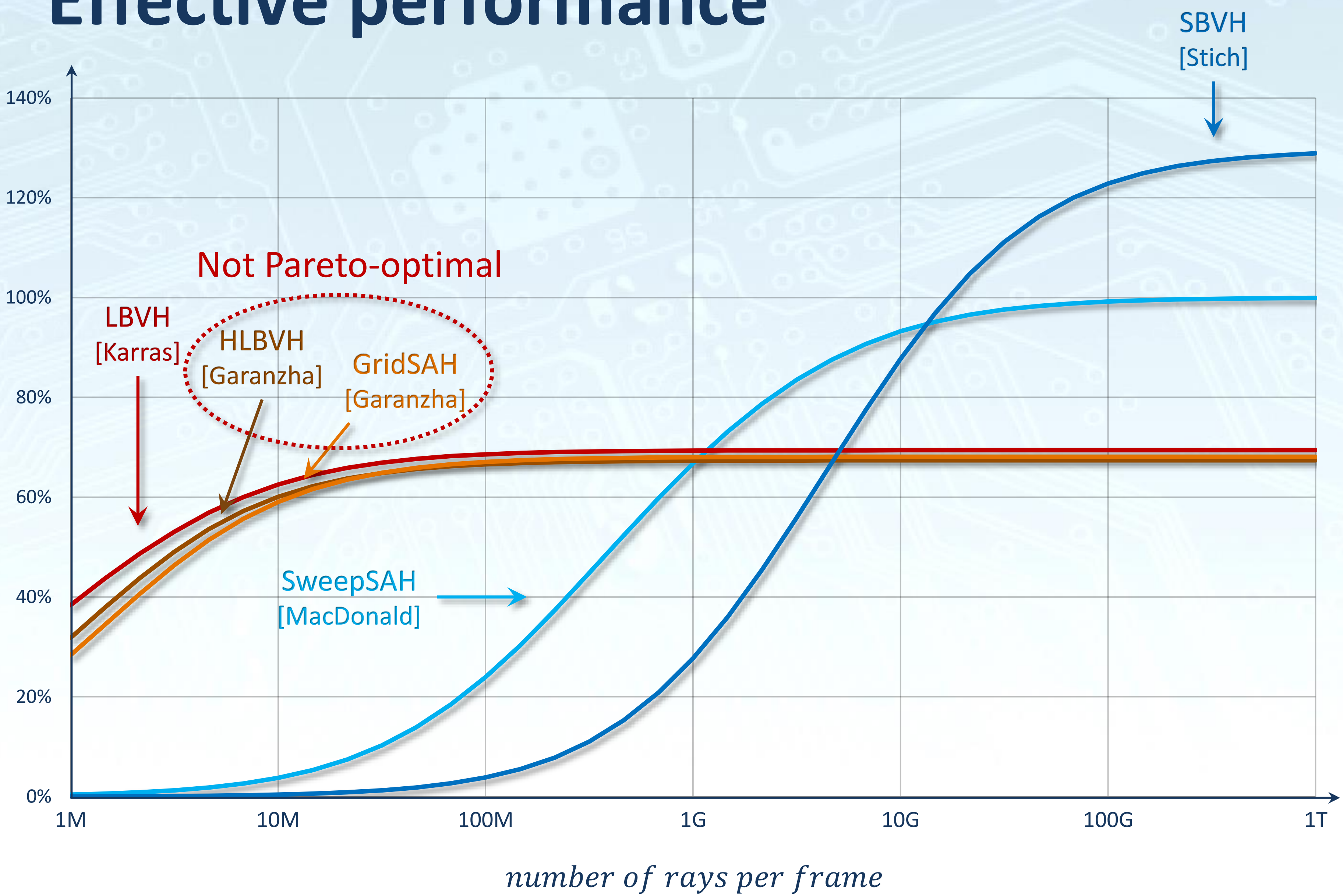


number of rays per frame

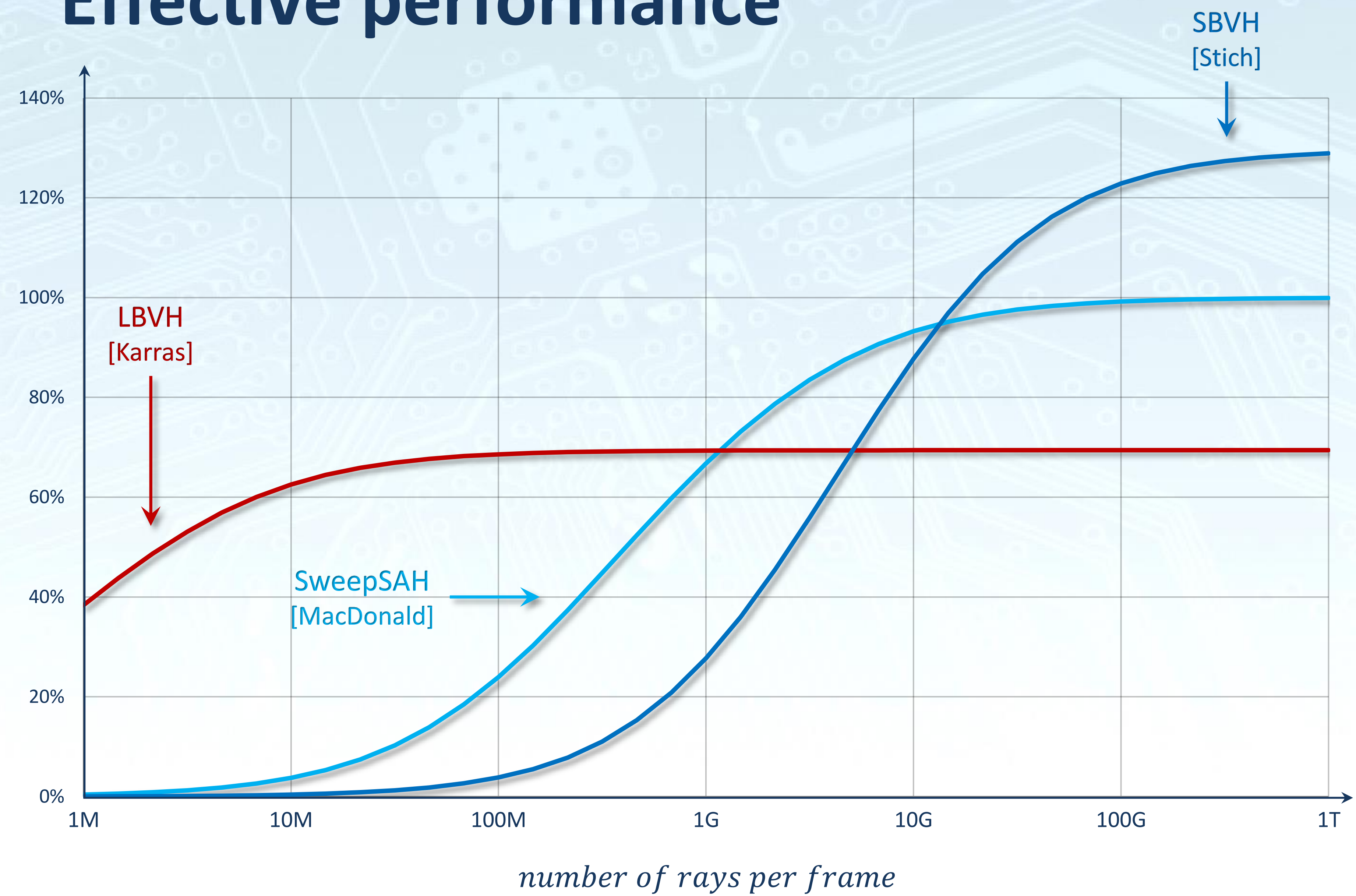
Effective performance



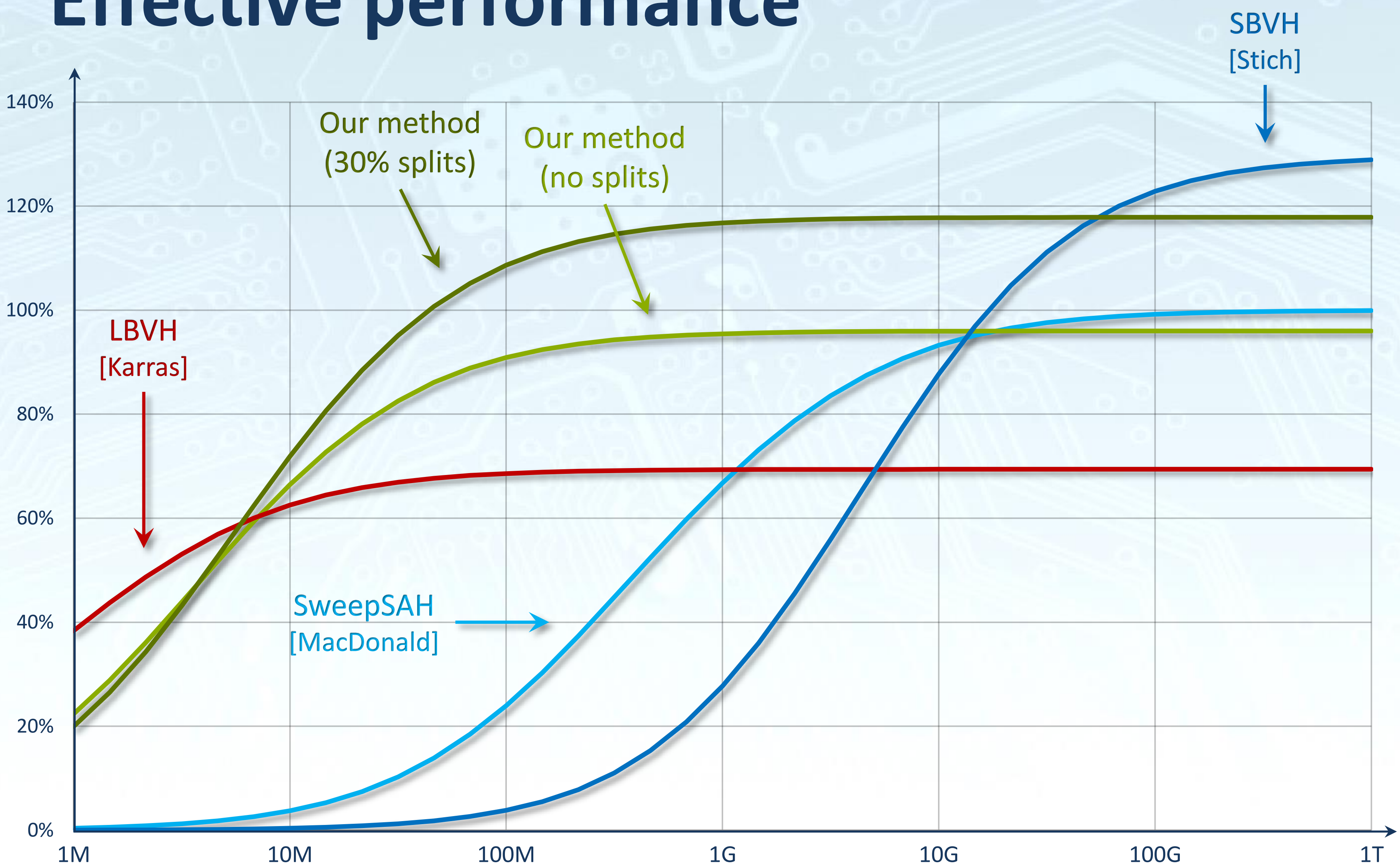
Effective performance



Effective performance

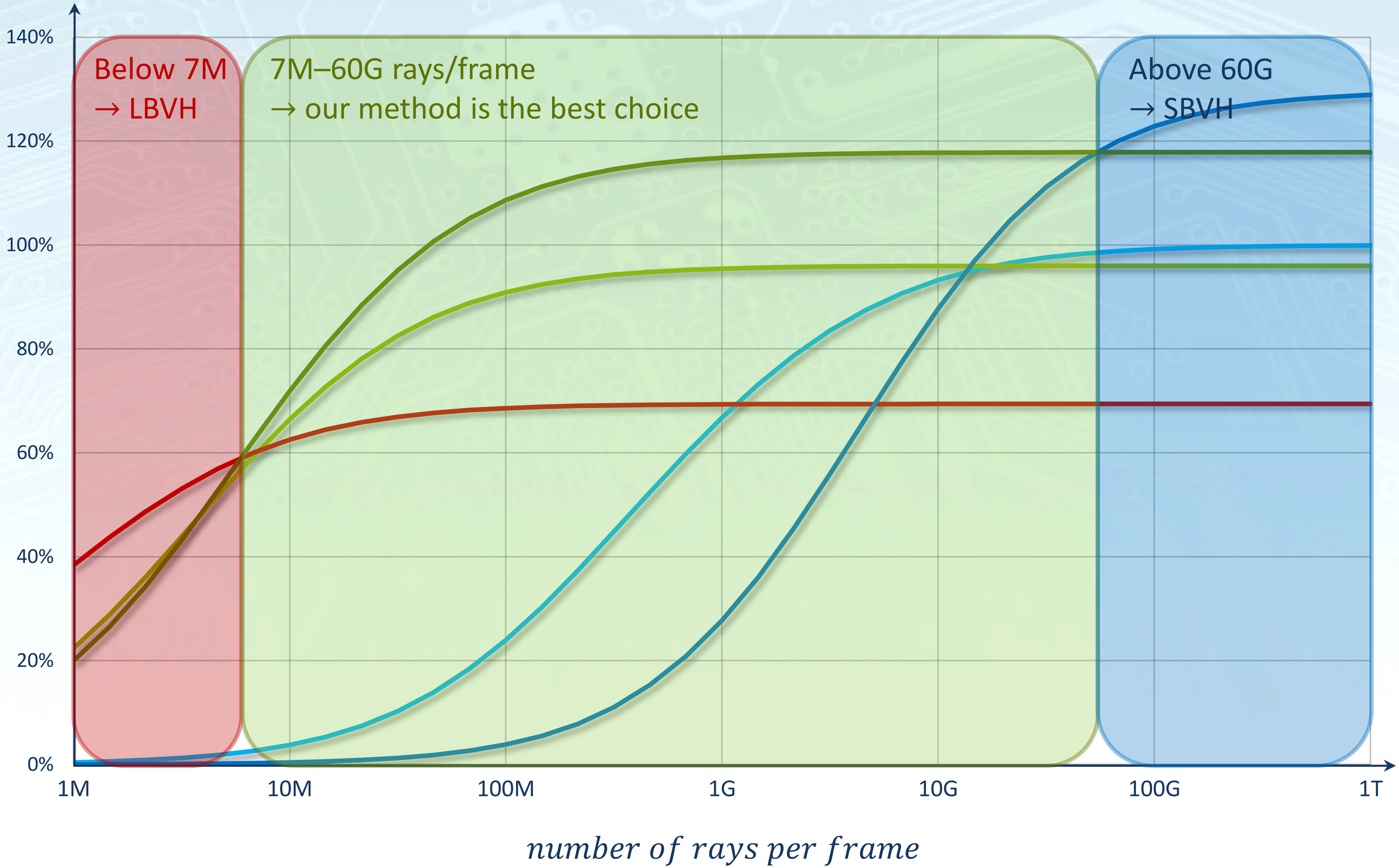


Effective performance



number of rays per frame

Effective performance



Conclusion

- General framework for optimizing trees
  - Inherently parallel
  - Approximate restructuring → larger treelets?
- Practical GPU-based BVH builder
  - Best choice in a large class of applications
  - Adjustable quality–speed tradeoff
- Will be integrated into NVIDIA OptiX

Thank you

Acknowledgements

- Samuli Laine
- Jaakko Lehtinen
- Sami Liedes
- David McAllister
- Anonymous reviewers
- Anat Grynberg and Greg Ward for Conference
- University of Utah for Fairy
- Marko Dabrovic for Sibenik
- Ryan Vance for Bubs
- Samuli Laine for Hairball and Vegetation
- Guillermo Leal Laguno for SanMiguel
- Jonathan Good for Arabic, Babylonian and Italian
- Stanford Computer Graphics Laboratory for Armadillo, Buddha and Dragon
- Cornell University for Bar
- Georgia Institute of Technology for Blade