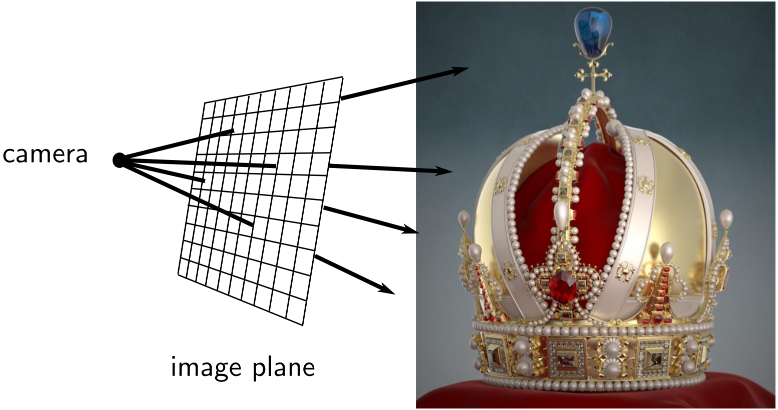
Parallel Reinsertion for Bounding Volume Hierarchy Optimization

Daniel Meister and Jiří Bittner

Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
Czech Technical University in Prague

Motivation: High-Performance Ray Tracing

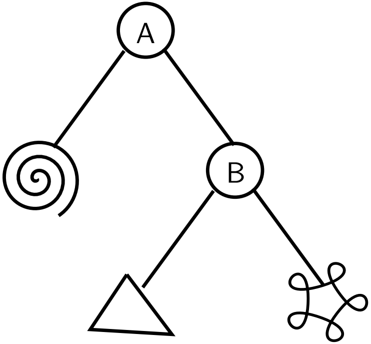
- Movie industry - saving hours of computational time
- Computer games - precomputed BVH for static geometry

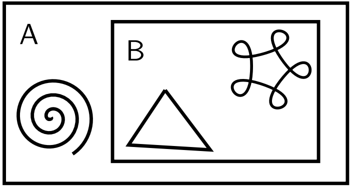


[courtesy of Martin Lubich]

Bounding Volume Hierarchy (BVH)

- Ray tracing, collision detection, visibility culling
- Rooted tree of arbitrary branching factor
  - References to geometric primitives in leaves
  - Bounding volumes in interior nodes

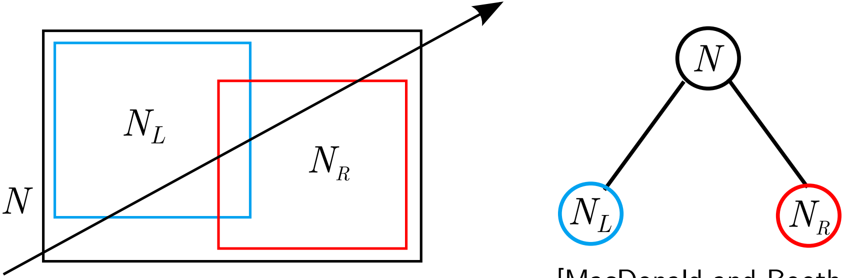




[Clark 1976]

Surface Area Heuristic (SAH)

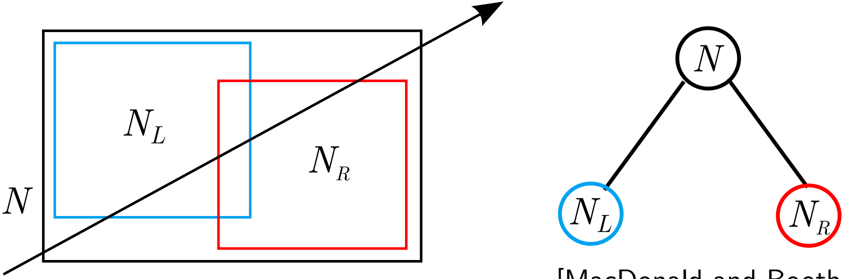
$$c(N) = \begin{cases} c_T + P(N_L|N)c(N_L) + P(N_R|N)c(N_R) & \text{if } N \text{ is interior node} \\ c_I|N| & \text{otherwise} \end{cases}$$



[MacDonald and Booth 1990]

Surface Area Heuristic (SAH)

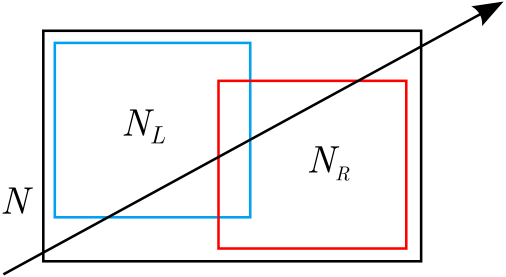
$$c(N) = \begin{cases} c_T + P(N_L|N)c(N_L) + P(N_R|N)c(N_R) & \text{if } N \text{ is interior node} \\ c_I|N| & \text{otherwise} \end{cases}$$

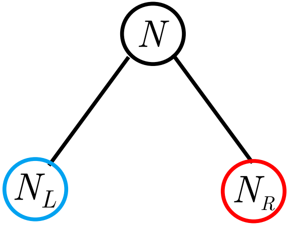


[MacDonald and Booth 1990]

Surface Area Heuristic (SAH)

$$c(N) = \begin{cases} c_T + P(N_L|N)c(N_L) + P(N_R|N)c(N_R) & \text{if } N \text{ is interior node} \\ c_I|N| & \text{otherwise} \end{cases}$$

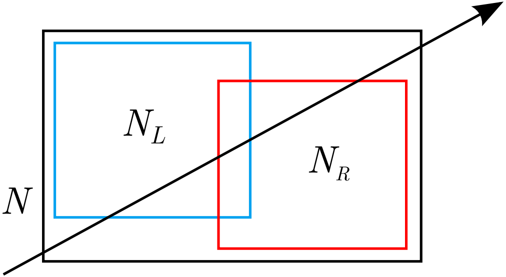


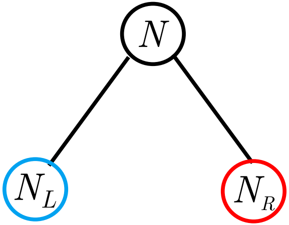


[MacDonald and Booth 1990]

Surface Area Heuristic (SAH)

$$c(N) = \begin{cases} c_T + \dfrac{SA(N_L)}{SA(N)} c(N_L) + P(N_R|N) c(N_R) & \text{if } N \text{ is interior node} \\ c_I |N| & \text{otherwise} \end{cases}$$

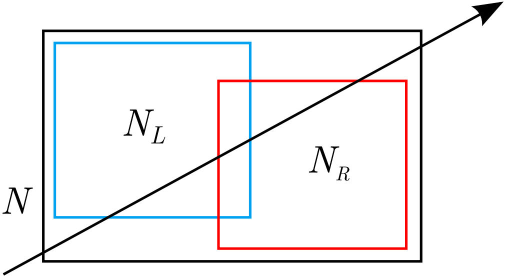


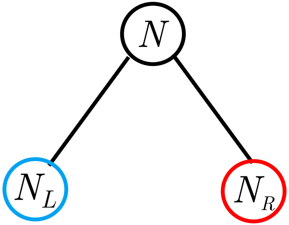


[MacDonald and Booth 1990]

Surface Area Heuristic (SAH)

$$c(N) = \begin{cases} c_T + \frac{SA(N_L)}{SA(N)}c(N_L) + \frac{SA(N_R)}{SA(N)}c(N_R) & \text{if } N \text{ is interior node} \\ c_I|N| & \text{otherwise} \end{cases}$$



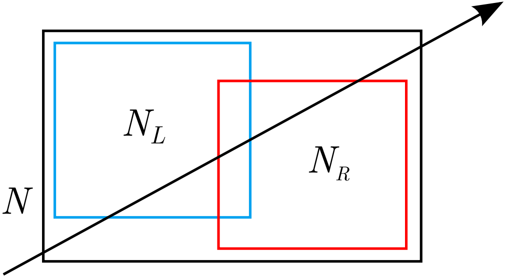


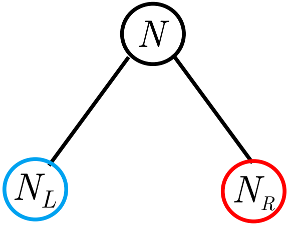
[MacDonald and Booth 1990]

Surface Area Heuristic (SAH)

$$c(N) = \begin{cases} c_T \boxed{+ \frac{SA(N_L)}{SA(N)} c(N_L)} + \boxed{\frac{SA(N_R)}{SA(N)} c(N_R)} & \text{if } N \text{ is interior node} \\ c_I |N| & \text{otherwise} \end{cases}$$

$$c(N_{root}) = \frac{1}{SA(N_{root})} \left[c_T \sum_{N_i} SA(N_i) + c_I \sum_{N_l} SA(N_l)|N_l| \right]$$



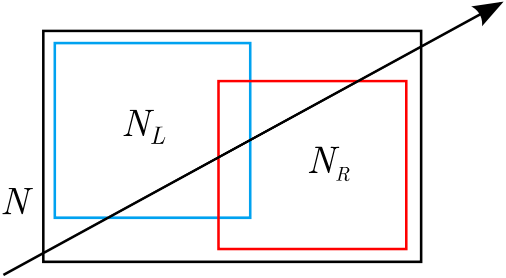


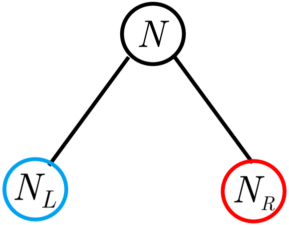
[MacDonald and Booth 1990]

Surface Area Heuristic (SAH)

$$c(N) = \begin{cases} c_T + \frac{SA(N_L)}{SA(N)} c(N_L) + \frac{SA(N_R)}{SA(N)} c(N_R) & \text{if } N \text{ is interior node} \\ c_I |N| & \text{otherwise} \end{cases}$$

$$c(N_{root}) = \frac{1}{SA(N_{root})} \left[c_T \sum_{N_i} SA(N_i) + c_I \sum_{N_l} SA(N_l) |N_l| \right] \propto \sum_{N_i} SA(N_i) \quad \text{if } |N_l| = 1$$



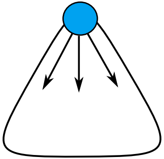


[MacDonald and Booth 1990]

BVH Construction Methods

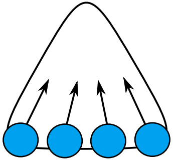
Top-down

- Surface Area Heuristic [Hunt et al. 2007]
- Binning [Ize et al. 2007, Wald 2007]
- k-means clustering [Meister and Bittner 2016]



Bottom-up

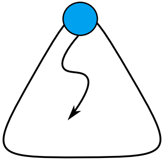
- Agglomerative clustering [Walter et al. 2008, Gu et al. 2013]
- Approx. aggl. clustering [Gu et al. 2013, Meister and Bittner 2017]



BVH Construction Methods

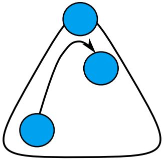
Insertion

- Heuristic greedy search [Goldsmith and Salmon 1987]
- Online construction [Bittner et al. 2015]



Optimization

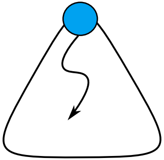
- Rotations [Kensler 2008, Kopta et al. 2012]
- Insertion-based optimization [Bittner 2013 et al.]
- Treelet restructuring [Karras and Aila 2013, Domingues and Pedrini 2015]



BVH Construction Methods

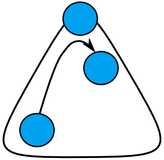
Insertion

- Heuristic greedy search [Goldsmith and Salmon 1987]
- Online construction [Bittner et al. 2015]

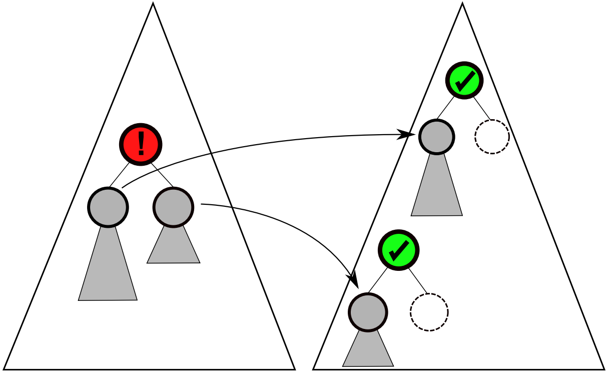


Optimization

- Rotations [Kensler 2008, Kopta et al. 2012]
- **Insertion-based optimization [Bittner 2013 et al.]**
- Treelet restructuring [Karras and Aila 2013, Domingues and Pedrini 2015]



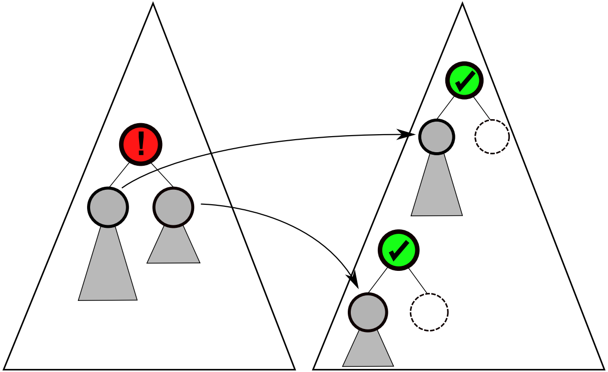
Sequential Insertion-Based Optimization



[Bittner et al. 2013]

Sequential Insertion-Based Optimization

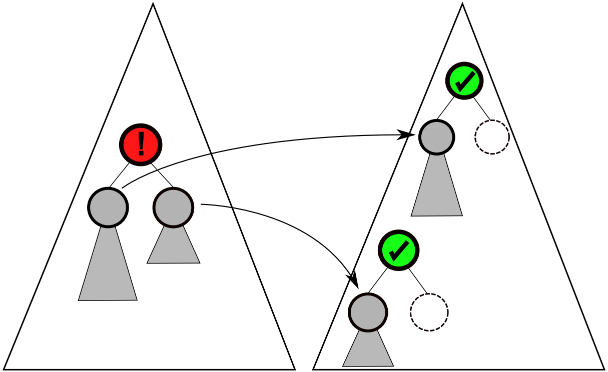
- **Remove** a node causing the cost overhead and **update** bounding boxes



[Bittner et al. 2013]

Sequential Insertion-Based Optimization

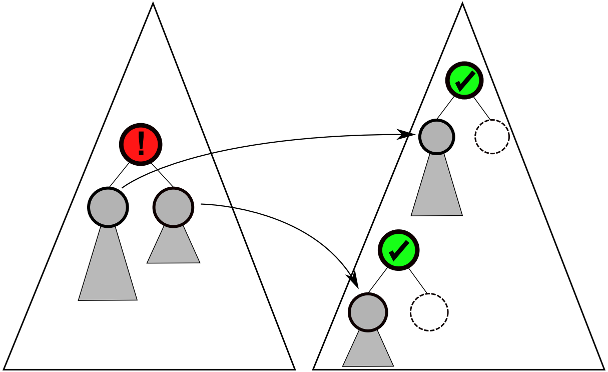
- **Remove** a node causing the cost overhead and **update** bounding boxes
- **Search** for a new position using branch-and-bound search with priority queue



[Bittner et al. 2013]

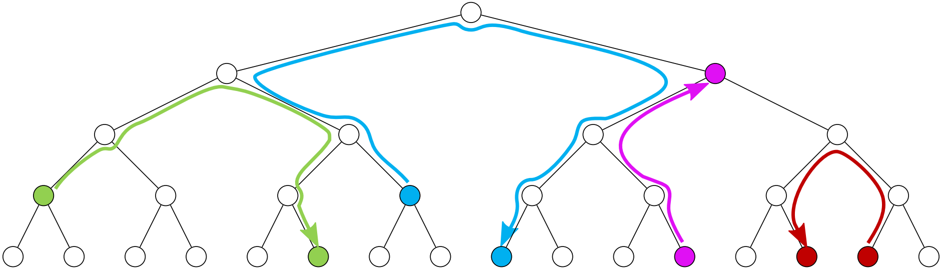
Sequential Insertion-Based Optimization

- **Remove** a node causing the cost overhead and **update** bounding boxes
- **Search** for a new position using branch-and-bound search with priority queue
- **Insert** the child nodes into the found position decreasing the global cost



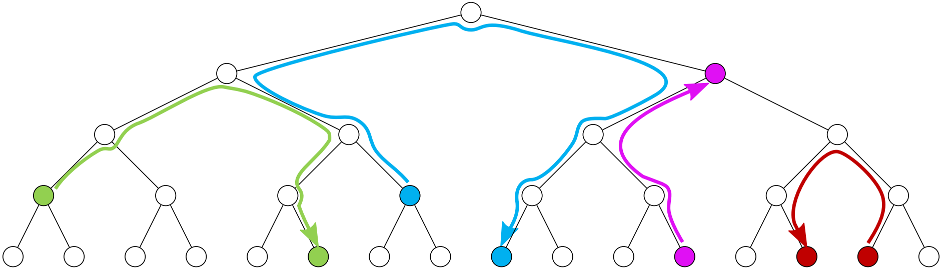
[Bittner et al. 2013]

Parallel Insertion-Based Optimization



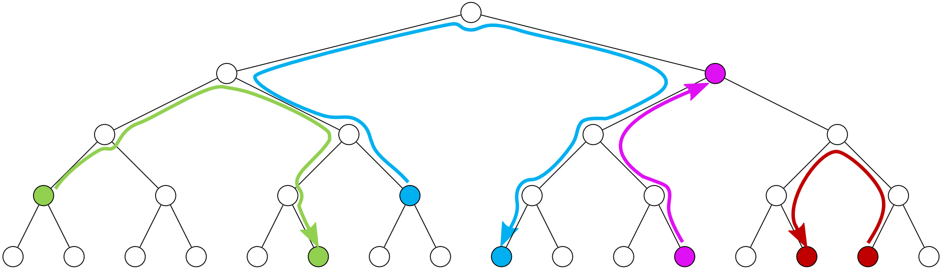
Parallel Insertion-Based Optimization

- Search for new positions for all nodes **in parallel**



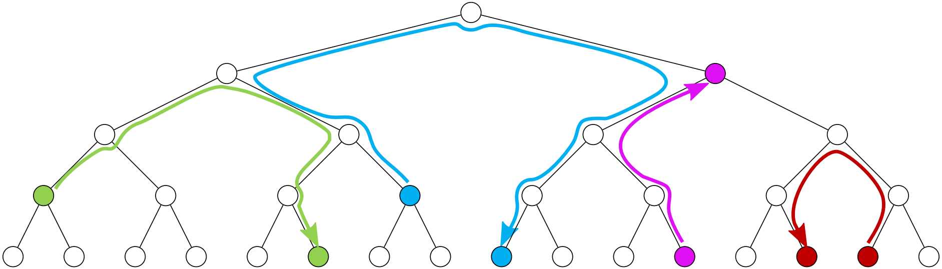
Parallel Insertion-Based Optimization

- Search for new positions for all nodes **in parallel**
- Resolve conflicts prioritizing nodes with the higher cost reduction **in parallel**

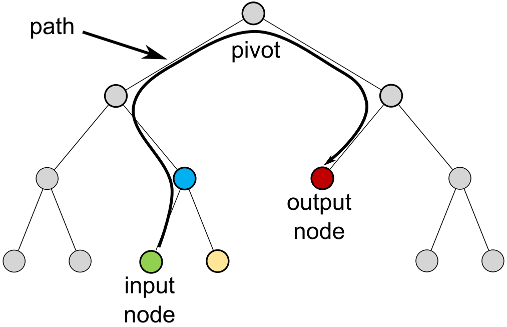


Parallel Insertion-Based Optimization

- Search for new positions for all nodes **in parallel**
- Resolve conflicts prioritizing nodes with the higher cost reduction **in parallel**
- Reinsert not conflicting nodes **in parallel**

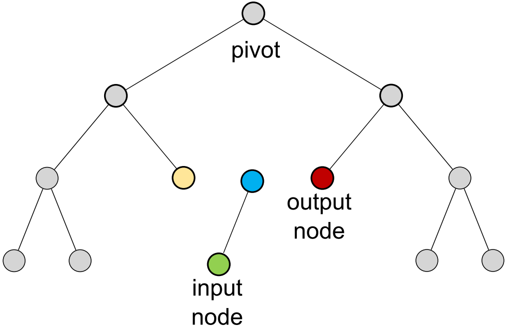


Reinsertion = Removal + Insertion



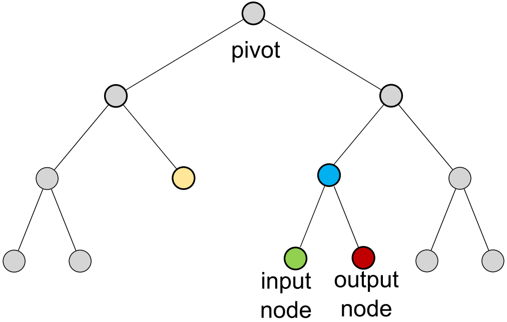
Reinsertion = Removal + Insertion

- Removal - remove input node and its parent



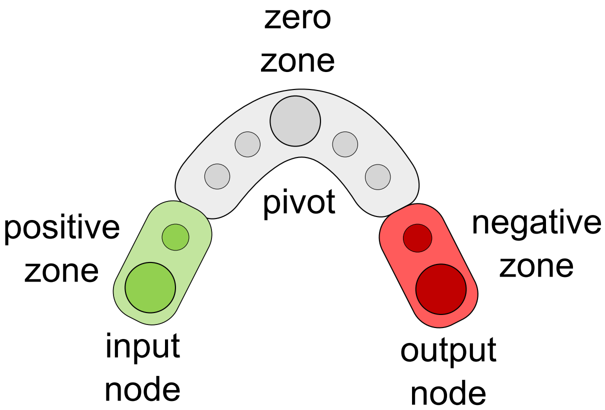
Reinsertion = Removal + Insertion

- Removal - remove input node and its parent
- Insertion - use parent as a common parent for input and output nodes



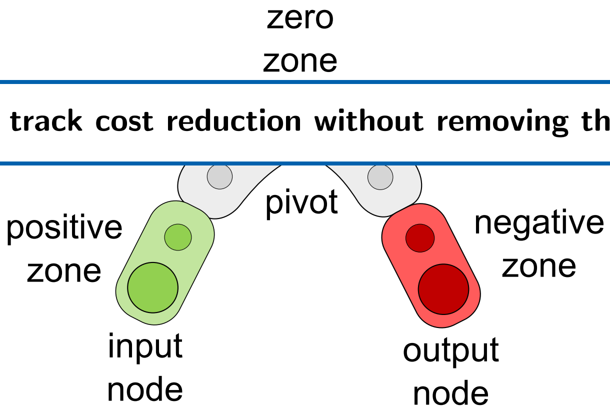
Bounding Boxes on Path

- Positive zone - removals shrinking bounding boxes
- Zero zone - removals and insertions not changing bounding boxes
- Negative zone - insertions enlarging bounding boxes



Bounding Boxes on Path

- Positive zone - removals shrinking bounding boxes
- Zero zone - removals and insertions not changing bounding boxes
- Negative zone - insertions enlarging bounding boxes

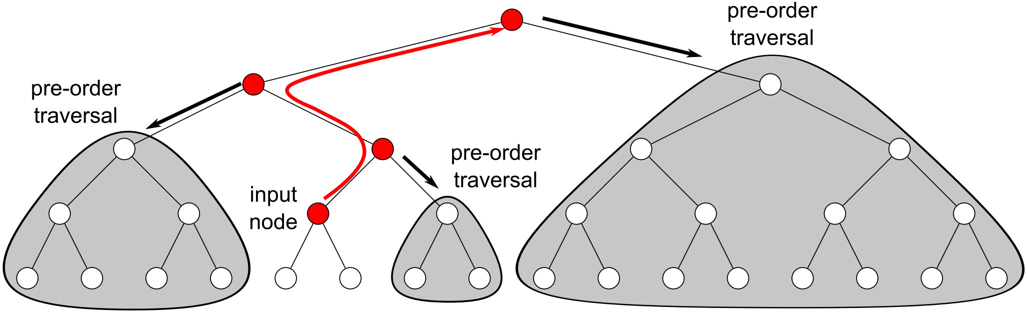


We can track cost reduction without removing the node!

Search Overview

Proceeding up to the root visiting sibling subtrees

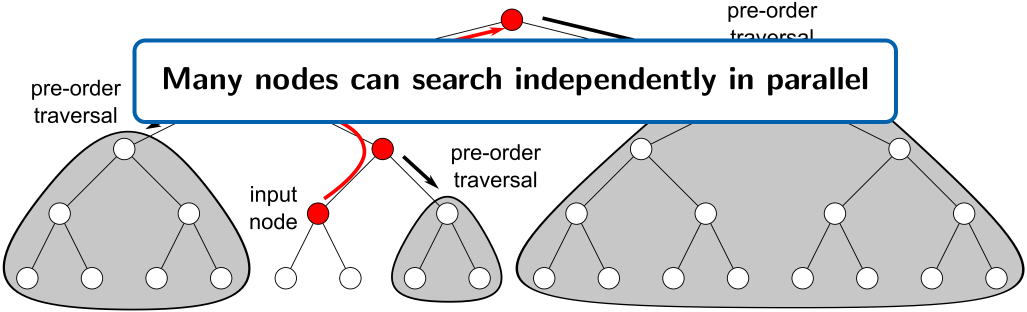
- Pre-order traversal using parent links (no priority queue!)
- Incrementally tracking the cost reduction
- Pruning the search using the best output node found so far



Search Overview

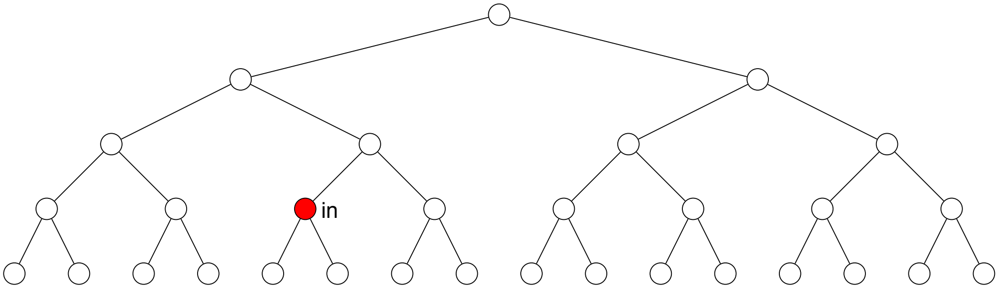
Proceeding up to the root visiting sibling subtrees

- Pre-order traversal using parent links (no priority queue!)
- Incrementally tracking the cost reduction
- Pruning the search using the best output node found so far

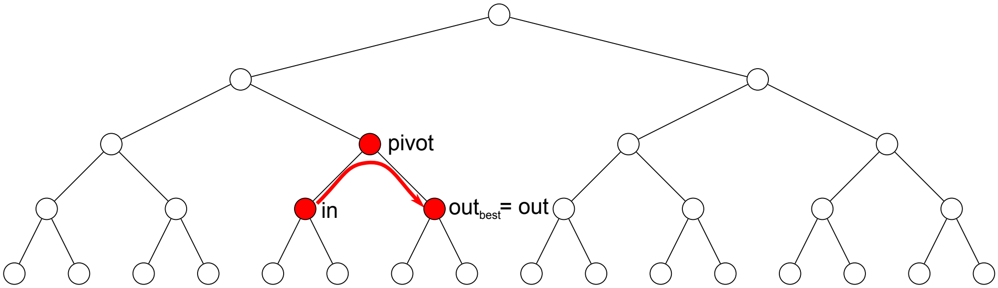


Many nodes can search independently in parallel

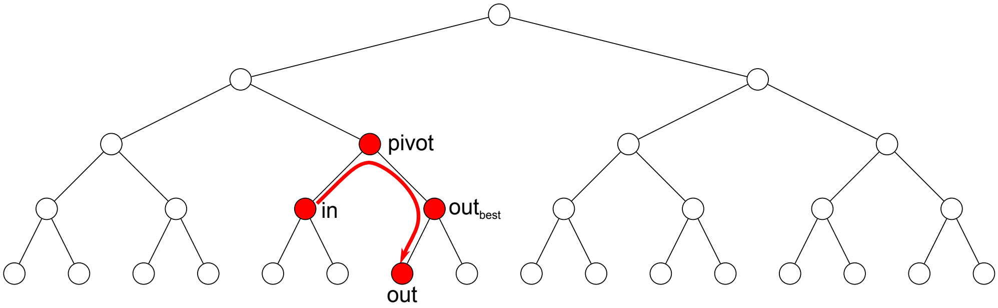
Search - Example



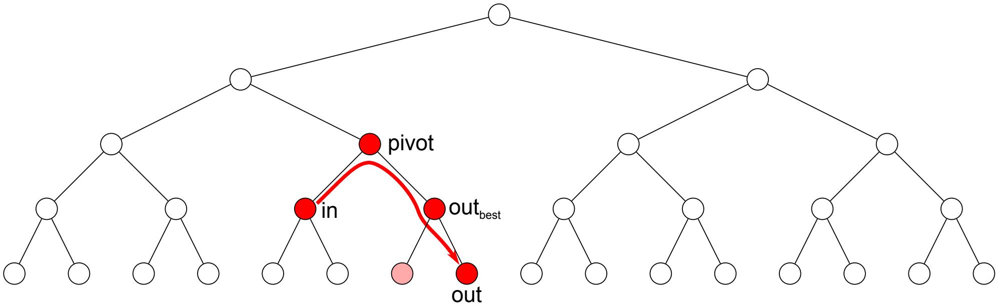
Search - Example



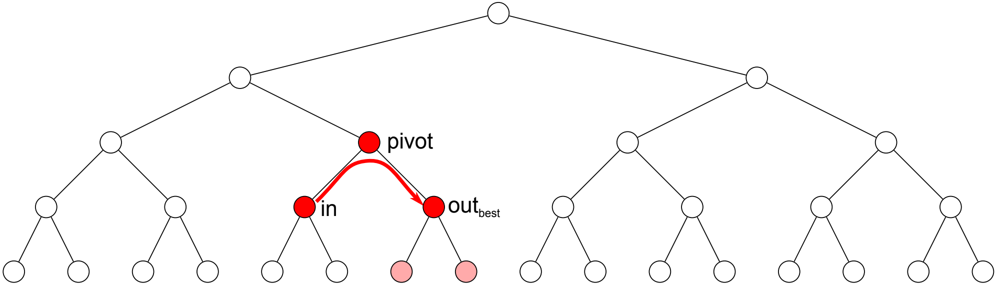
Search - Example



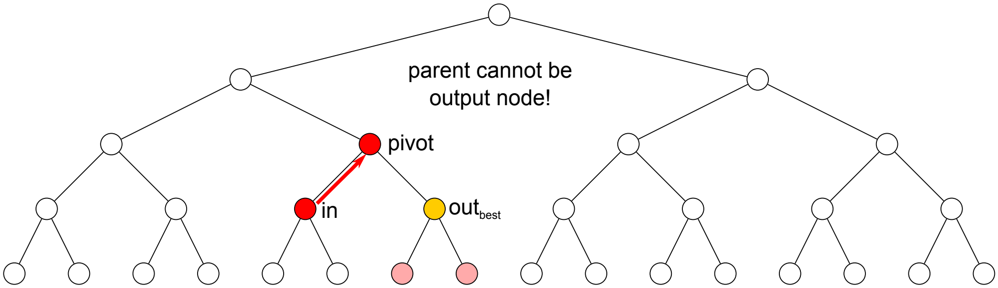
Search - Example



Search - Example

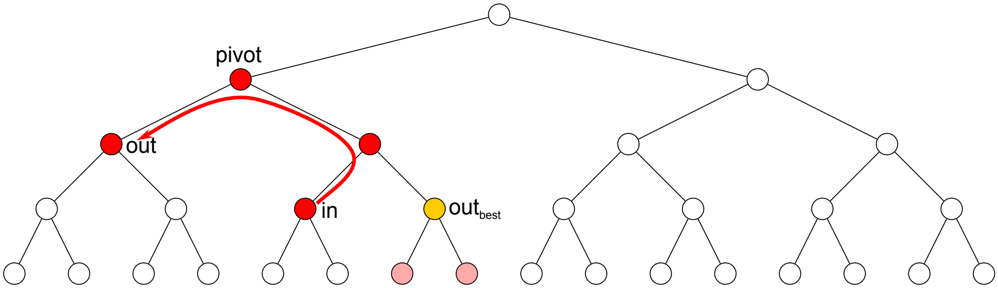


Search - Example

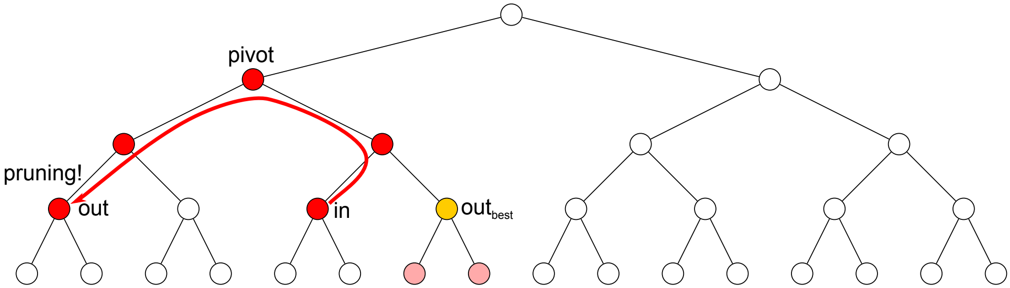


parent cannot be output node!

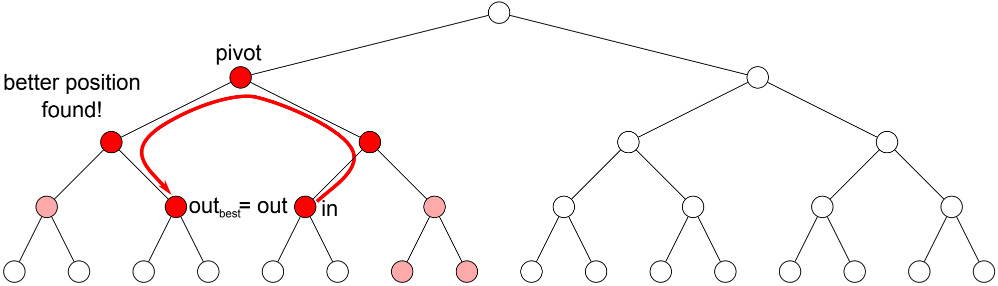
Search - Example



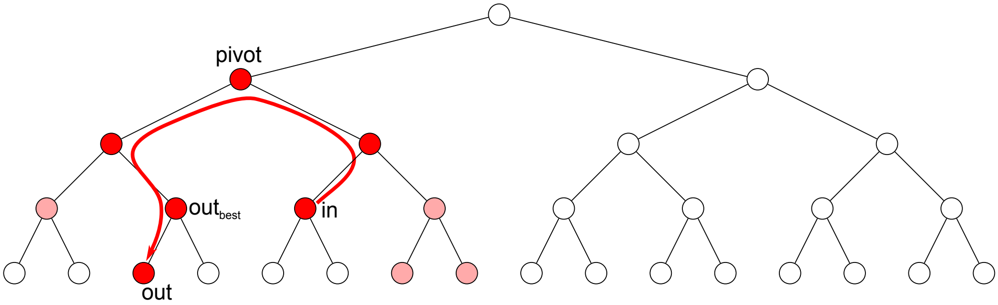
Search - Example



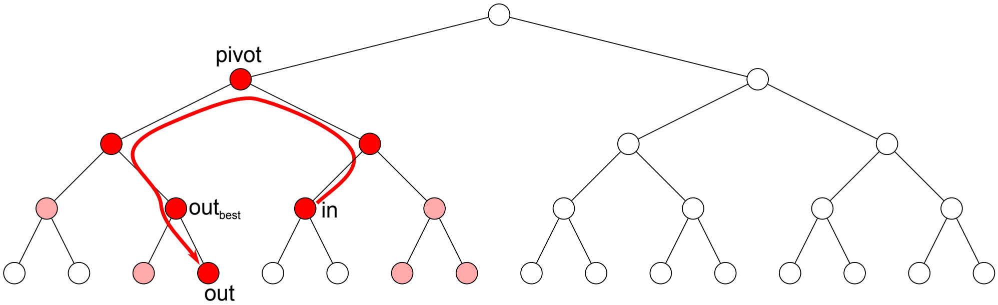
Search - Example



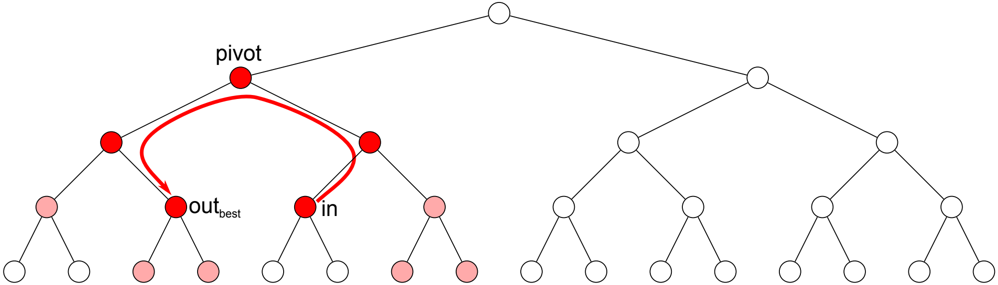
Search - Example



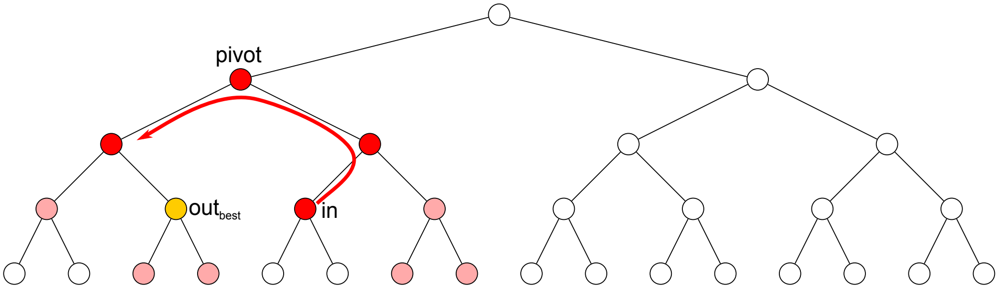
Search - Example



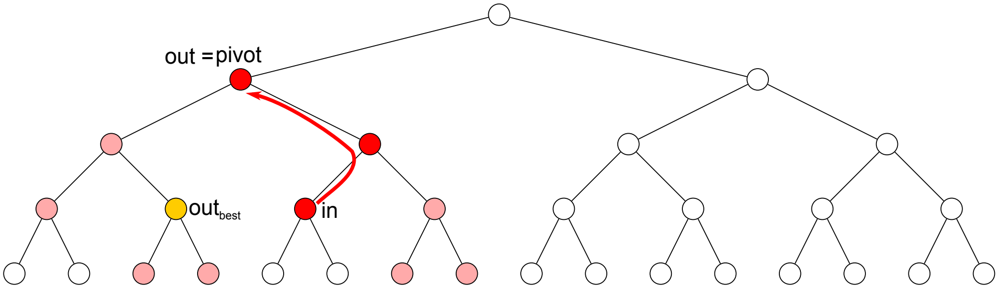
Search - Example



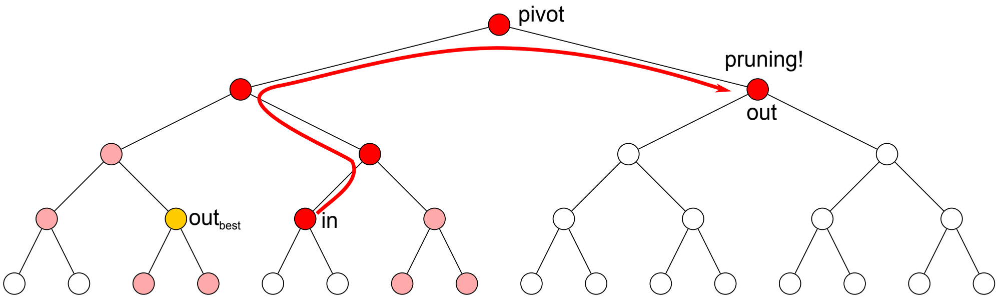
Search - Example



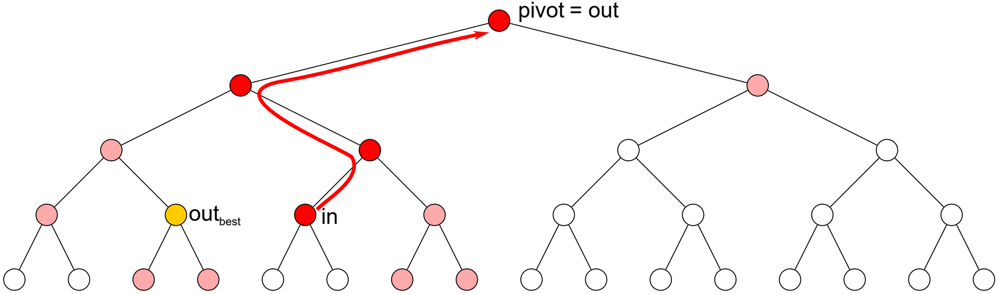
Search - Example



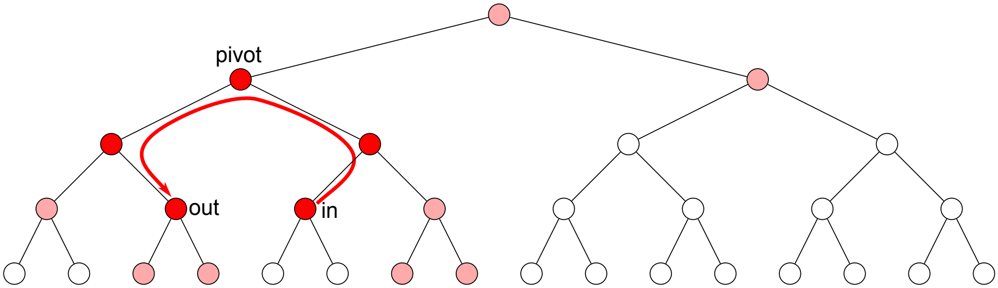
Search - Example



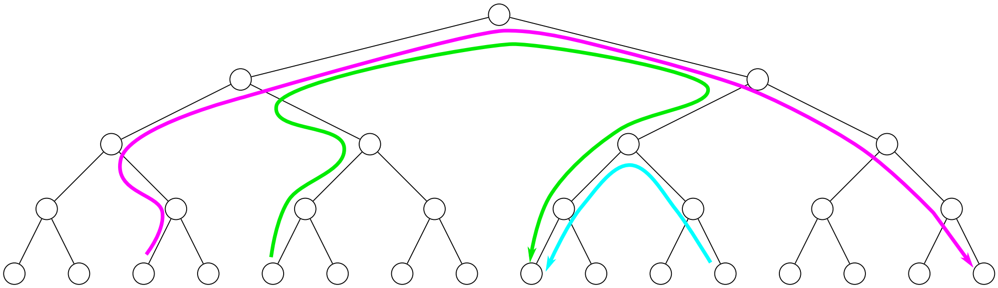
Search - Example



Search - Example

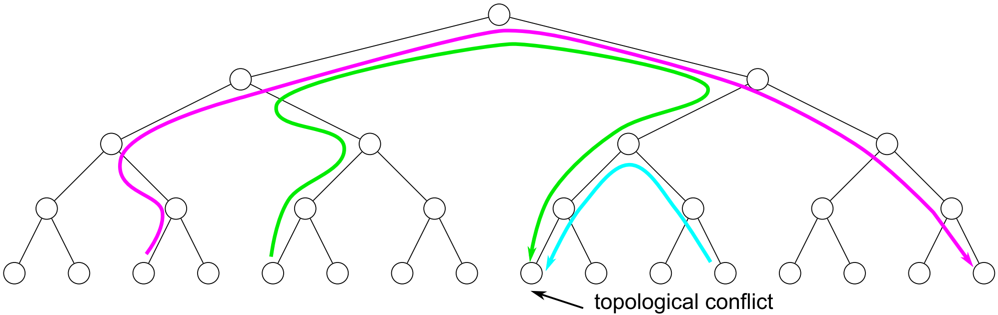


Parallel Reinsertion



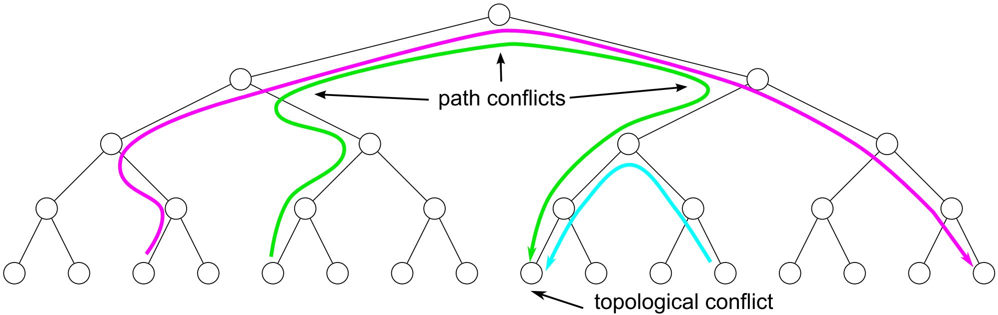
Parallel Reinsertion

- Topological conflicts - concurrent modification of topology



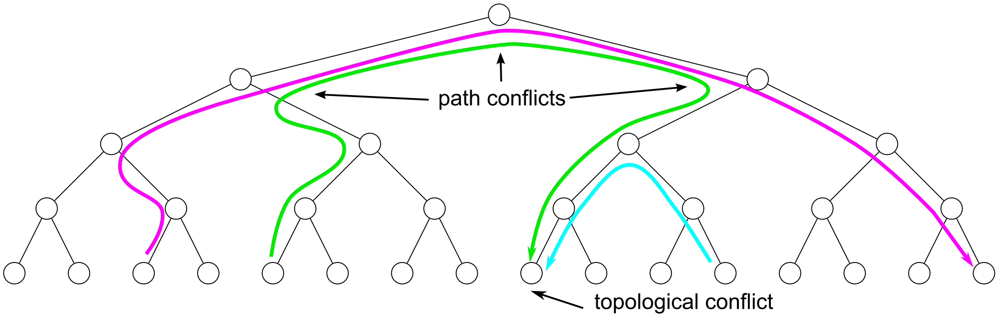
Parallel Reinsertion

- Topological conflicts - concurrent modification of topology
- Path conflicts - sharing nodes on the paths



Parallel Reinsertion

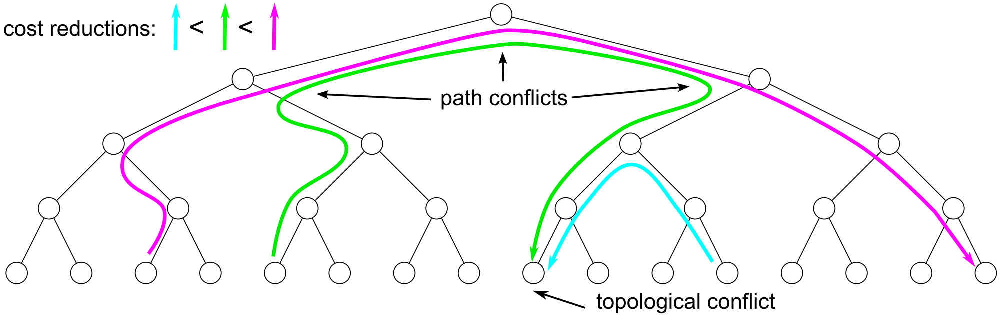
- Topological conflicts - concurrent modification of topology
- Path conflicts - sharing nodes on the paths



Conflict resolution by atomic locks prioritizing paths with higher cost reduction

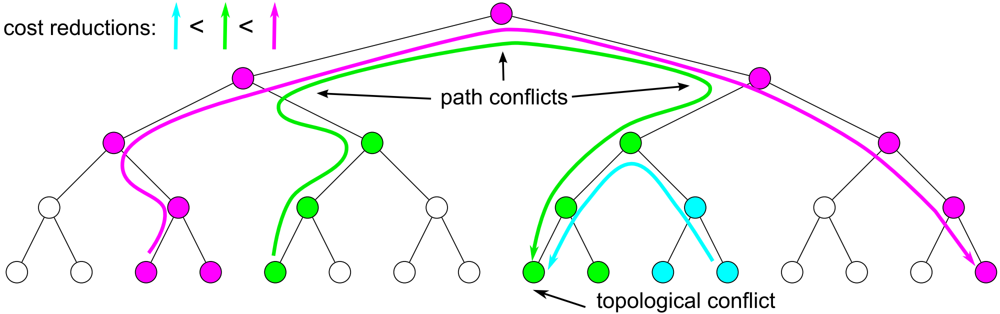
Conservative Strategy

Resolve both topological and path conflicts



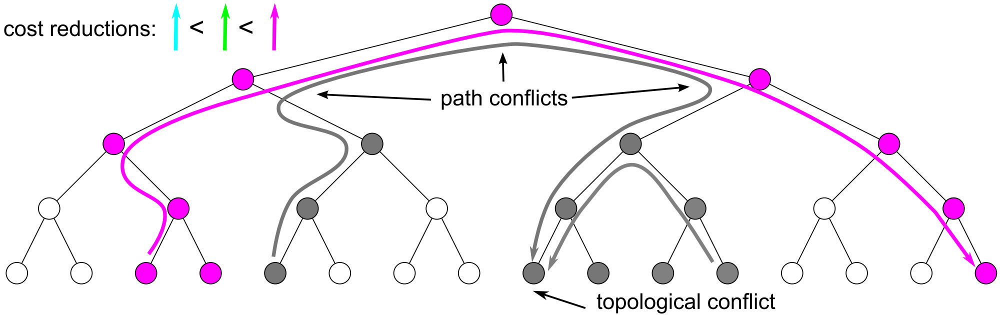
Conservative Strategy

Resolve both topological and path conflicts



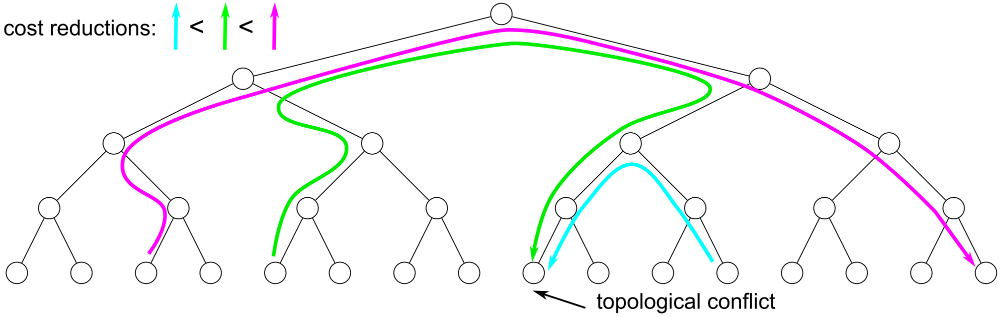
Conservative Strategy

Resolve both topological and path conflicts



Aggressive Strategy

Resolve only topological conflicts

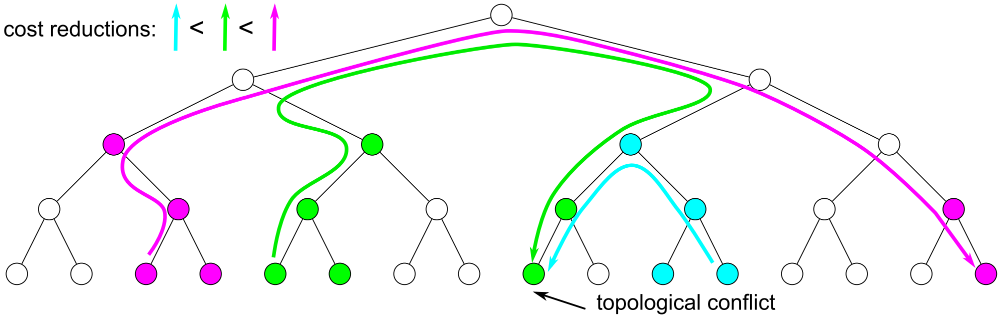


cost reductions: ↑ < ↑ < ↑

topological conflict

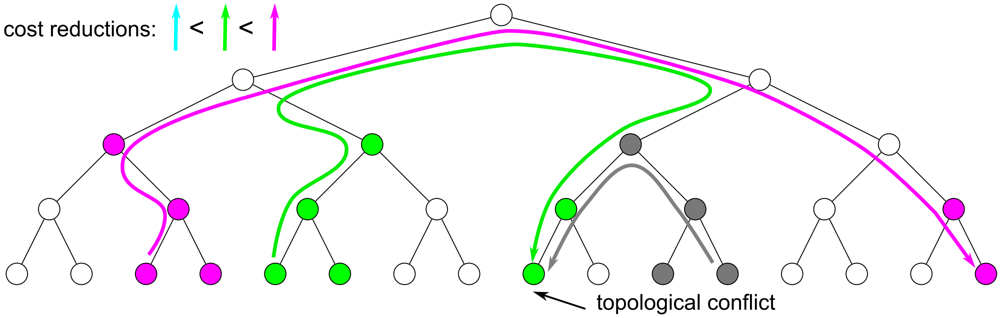
Aggressive Strategy

Resolve only topological conflicts



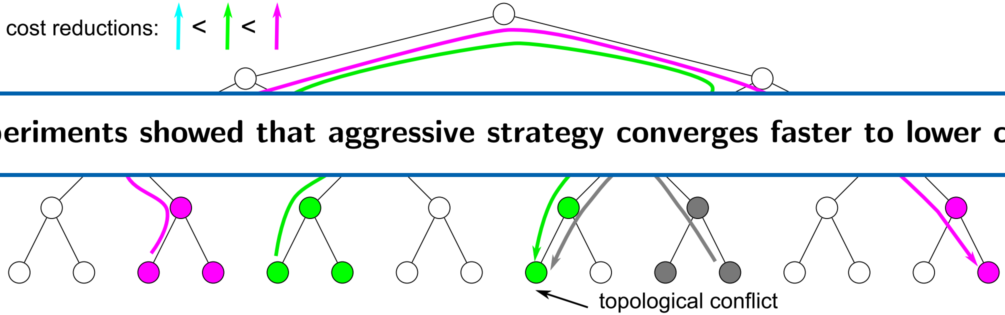
Aggressive Strategy

Resolve only topological conflicts



Aggressive Strategy

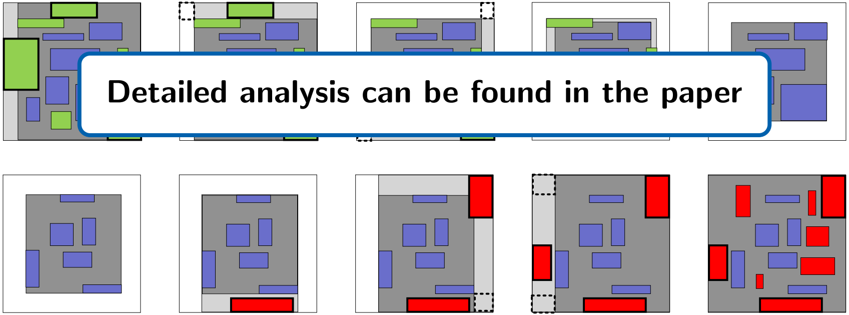
Resolve only topological conflicts



Experiments showed that aggressive strategy converges faster to lower costs

Superiority of Aggressive Strategy

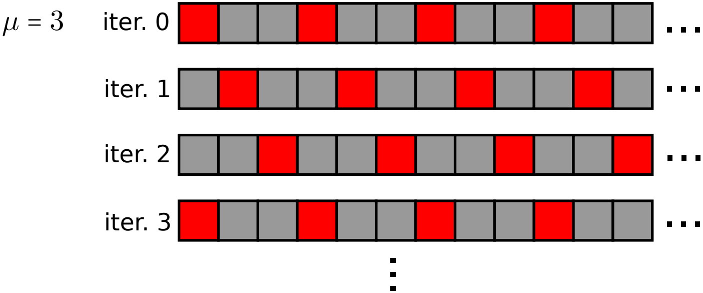
- Significantly more reinsertions performed in parallel
- Total cost reduction is not sum of costs reductions of individual reinsertions



Detailed analysis can be found in the paper

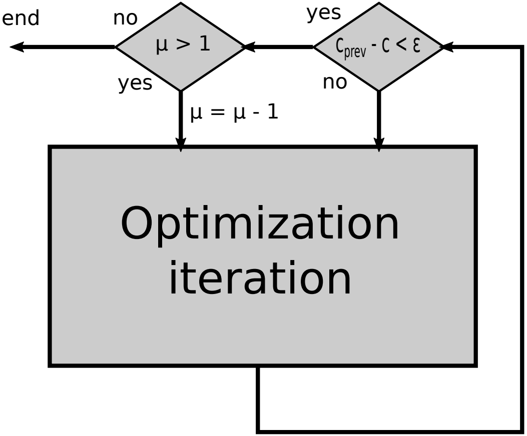
Performance Optimization - Sparse Search

- Search phase is the bottleneck
- Chance of conflicts between neighboring nodes
- Process every μ-th node shifted by index of iteration (parameter μ)



Termination Criteria

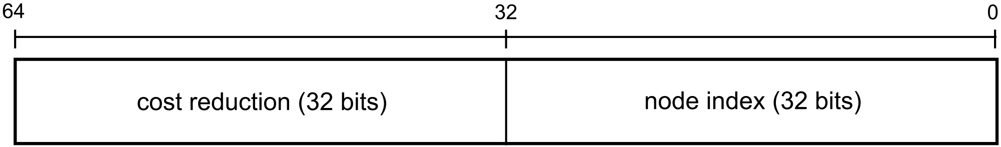
parameter ε (reasonable choice $\varepsilon = 0.1$)



Implementation in CUDA

Atomic lock

- 64-bit integers with atomic max
- Comparison of positive floats in integer representation
- Using node index to prevent deadlocks



64 32 0

cost reduction (32 bits)

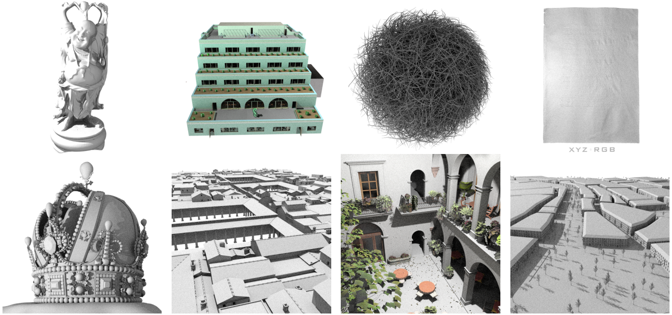
node index (32 bits)

Path encoding

- Only necessary for the conservative strategy
- Path in binary tree encoded in bitset
- 128 bits enough for all paths

Results

- 8 scenes (1-8.6M tris)
- Path tracing (GPU ray tracing kernel [Aila and Laine 2009])
- Intel Core I7-3770 3.4 GHz CPU (4 cores), 16 GB RAM
- CUDA 9.1, NVIDIA GeForce GTX TITAN X (Maxwell), 12 GB RAM



Tested Methods

LBVH [Karras 2012]

- Spatial medians

ATRBVH [Domingues and Pedrini 2015]

- Treelet restructuring by agglomertive clustering

PLOC [Meister and Bittner 2017]

- Parallel locally-ordered clustering

RBVH [Bittner et. al 2013]

- Sequential insertion-based optimization

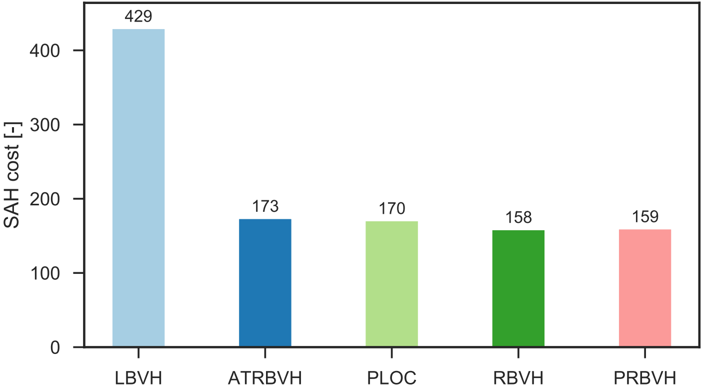
PRBVH

- Parallel insertion-based optimization (our algorithm)

Adaptive leaf sizes, SAH cost constants $c_T = 3$, $c_I = 2$

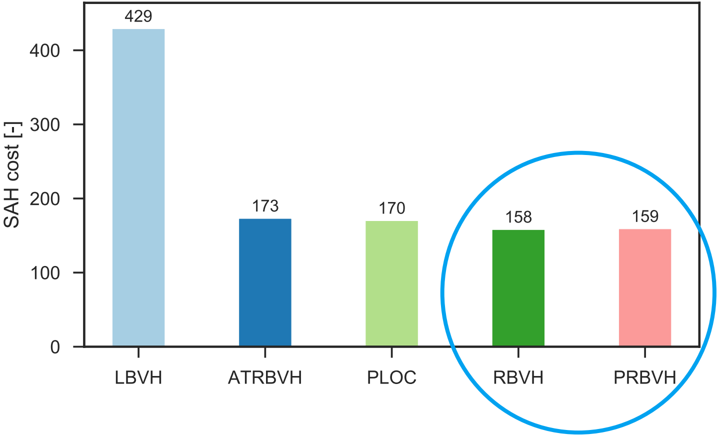
Pompeii

5632k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



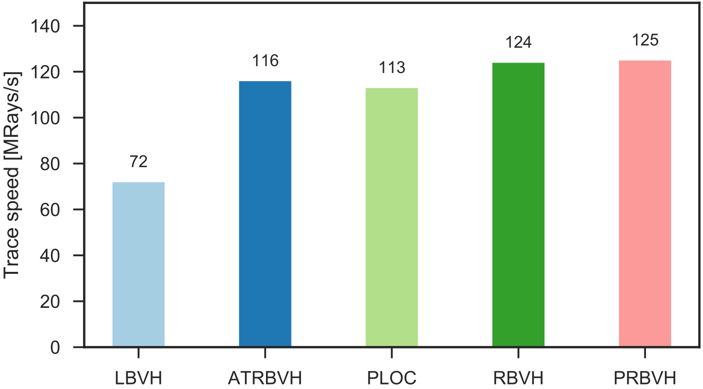
Pompeii

5632k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



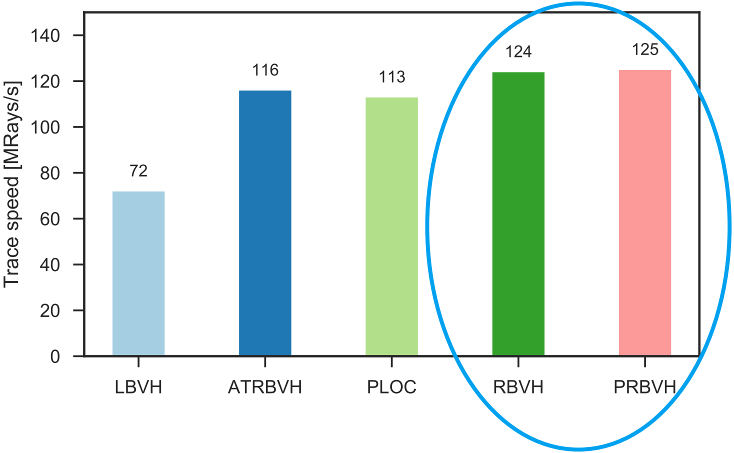
Pompeii

5632k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



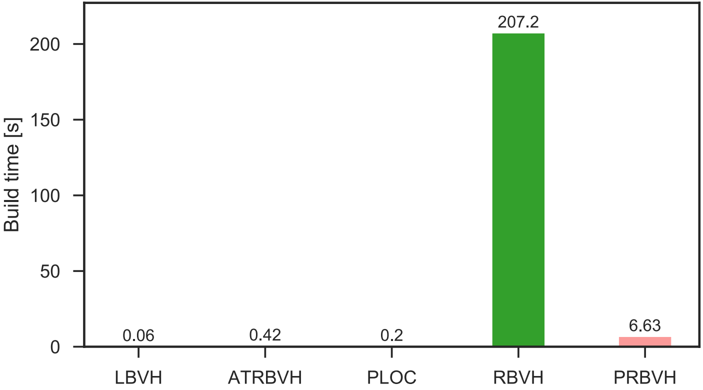
Pompeii

5632k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



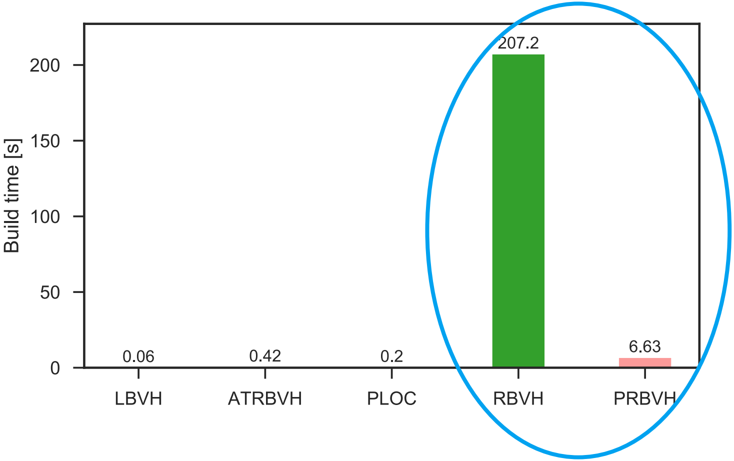
Pompeii

5632k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



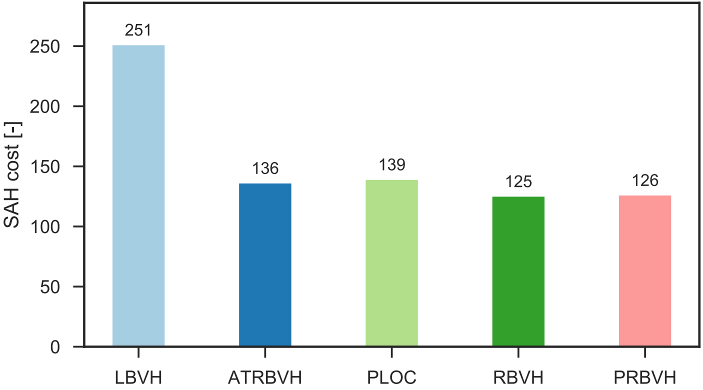
Pompeii

5632k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



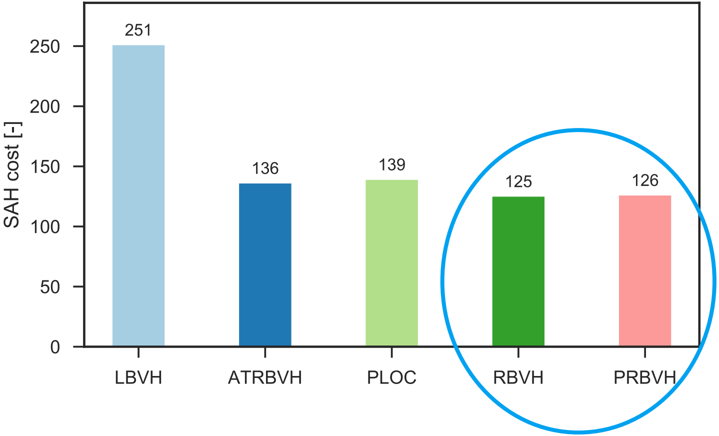
San Miguel

7880k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



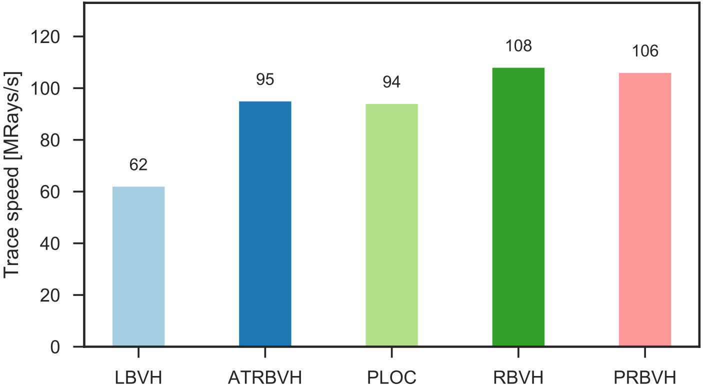
San Miguel

7880k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



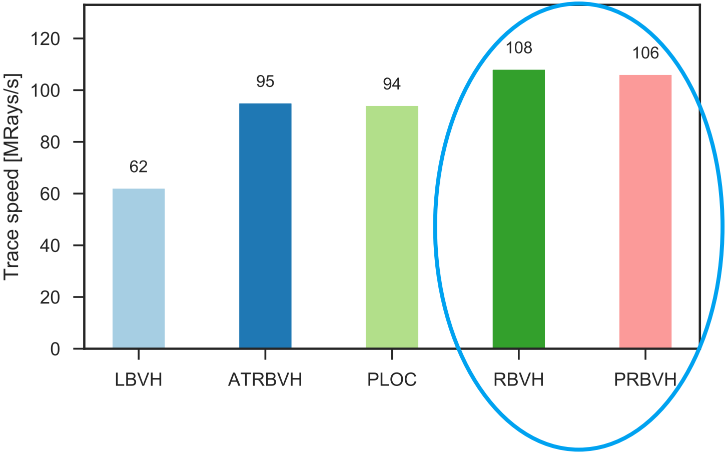
San Miguel

7880k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



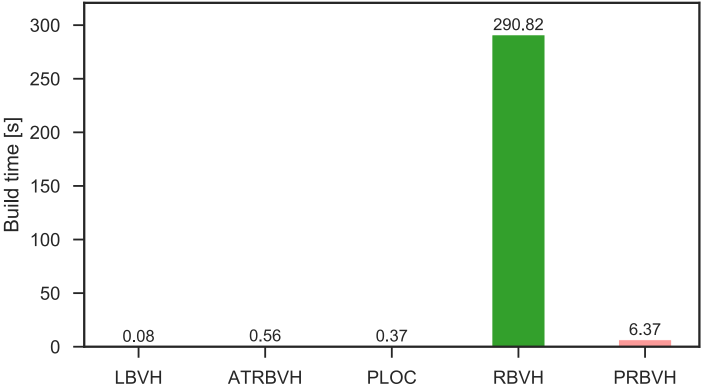
San Miguel

7880k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



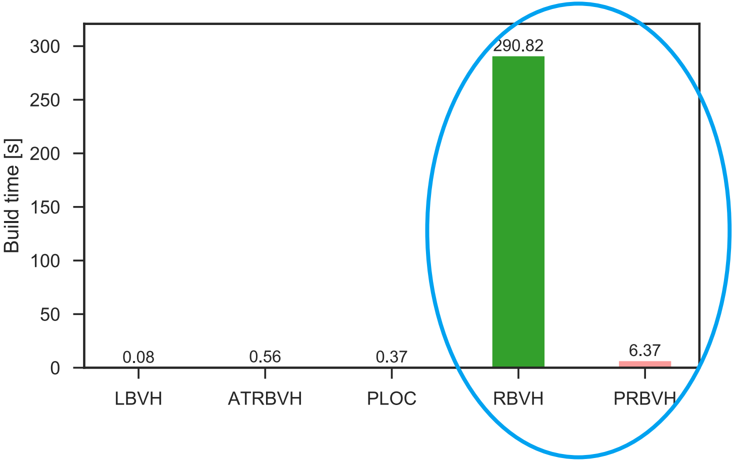
San Miguel

7880k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



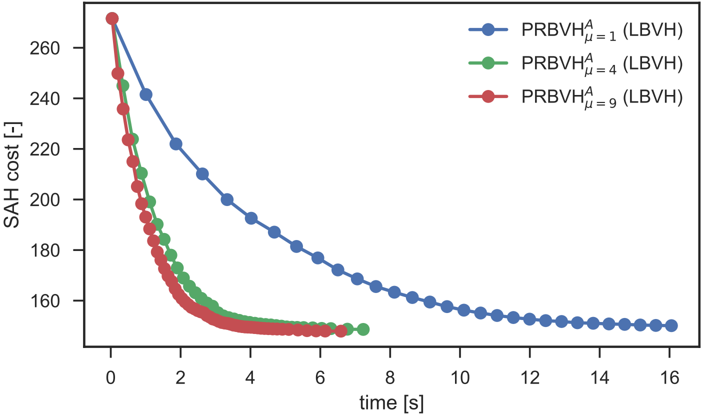
San Miguel

7880k tris, aggressive strategy, $\mu = 9$, $\varepsilon = 0.1$



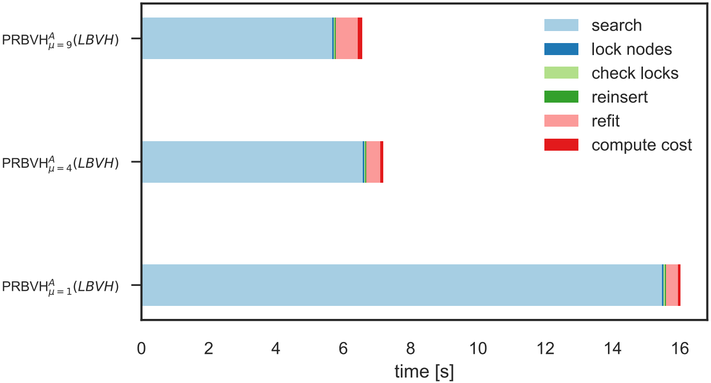
San Miguel

Influence of sparse search (the μ parameter)



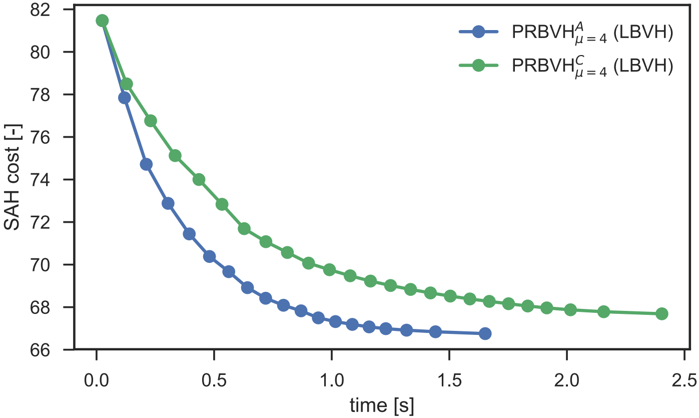
San Miguel

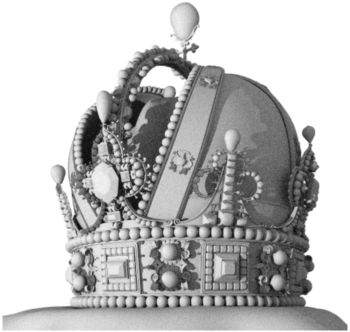
Influence of sparse search (the μ parameter)



Crown

Aggressive and conservative strategies





Conclusion and Future Work

Parallel BVH optimization

- Parallel search and locking scheme
- Two orders of magnitude faster than sequential method
- Trace performance w.r.t. state-of-the-art GPU builders
  - speedup 8% - 31% w.r.t. PLOC
  - speedup 4% - 12% w.r.t. ATRBVH
- Implementation in CUDA with released source codes

Future work

- Wide BVHs
- Spatial splits

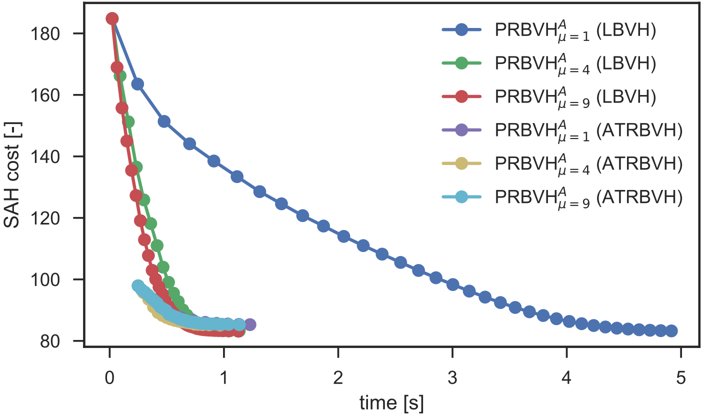
Thank you for your attention!

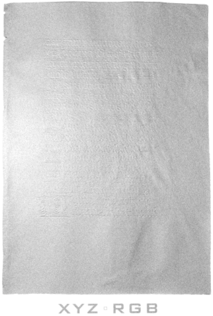
The project website with source codes

`http://dcgi.felk.cvut.cz/projects/prbvh/`

Manuscript

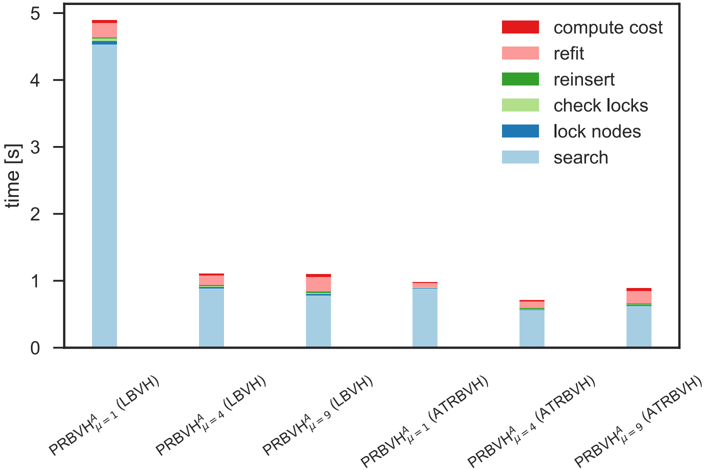
Initial BVH built by LBVH and ATRBVH





Manuscript

Initial BVH built by LBVH and ATRBVH


time [s]
🟥 compute cost
🟥 refit
🟩 reinsert
🟩 check locks
🟦 lock nodes
🟦 search
PRBVH$^A_{\mu=1}$ (LBVH)
PRBVH$^A_{\mu=4}$ (LBVH)
PRBVH$^A_{\mu=9}$ (LBVH)
PRBVH$^A_{\mu=1}$ (ATRBVH)
PRBVH$^A_{\mu=4}$ (ATRBVH)
PRBVH$^A_{\mu=9}$ (ATRBVH)