COMPUTER GRAPHIK
UNIVERSITÄT
DES SAARLANDES

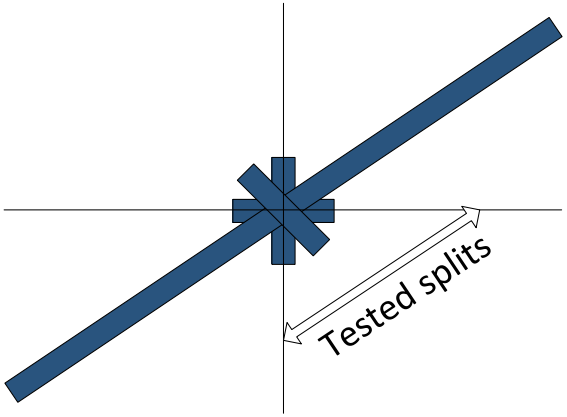
Object Partitioning Considered Harmful: Space Subdivision for BVHs

Stefan Popov, Iliyan Georgiev,
Rossen Dimov, and Philipp Slusallek

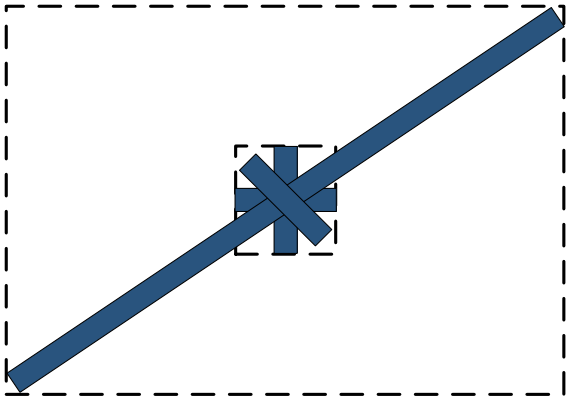
Motivation

- Classical BVH construction is not perfect
  - Looks only at primitive's centroids

Classical : Create leaf



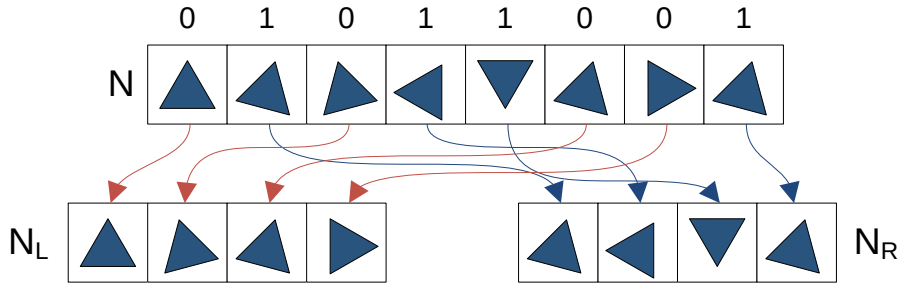
Correct solution



- How much more performance is there?

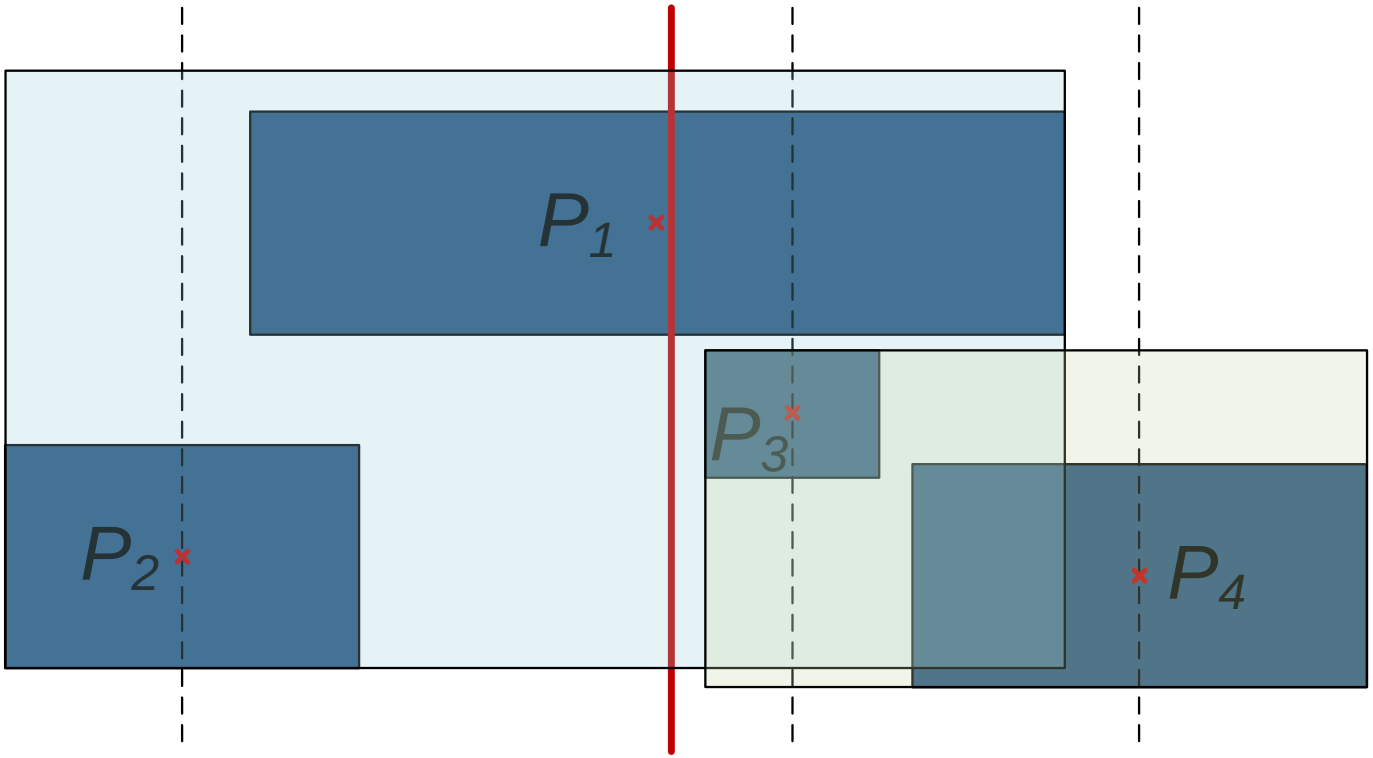
Background

- SAH: $cost(N) = C_T + \frac{SA(N_L)|N_L| + SA(N_R)|N_R|}{SA(N)}$
- Cost based BVH construction: Top-down
  - Partition set of *N*'s primitives into N_L and N_R



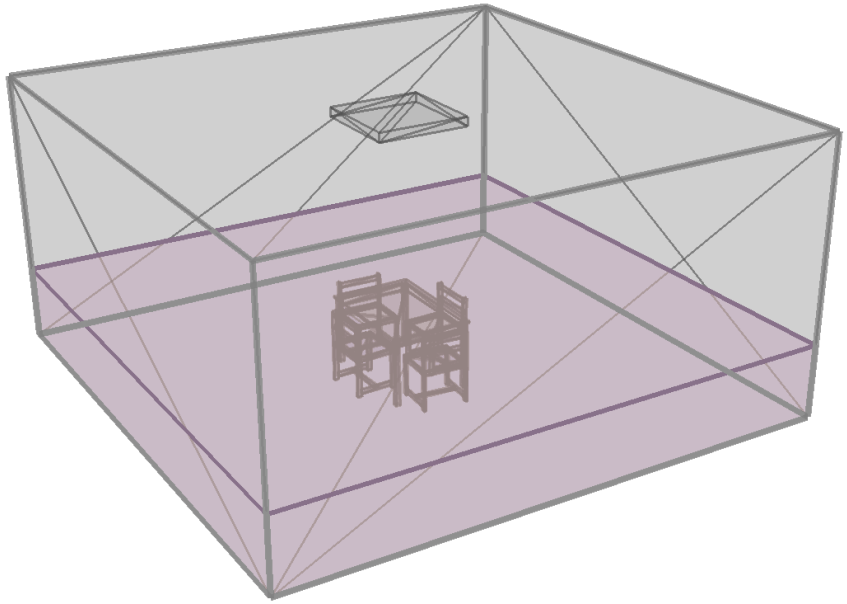
  - Take partition with minimal cost
- Exhaustive search: $O(2^N)$

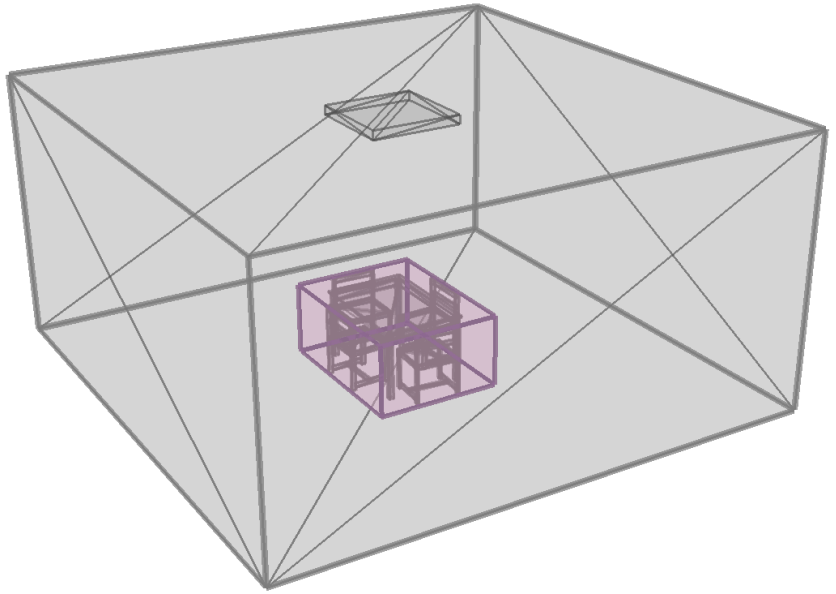
Classical BVH Construction



- Assumes finely tessellated geometry
  - Primitive → point

Can We Do Better?





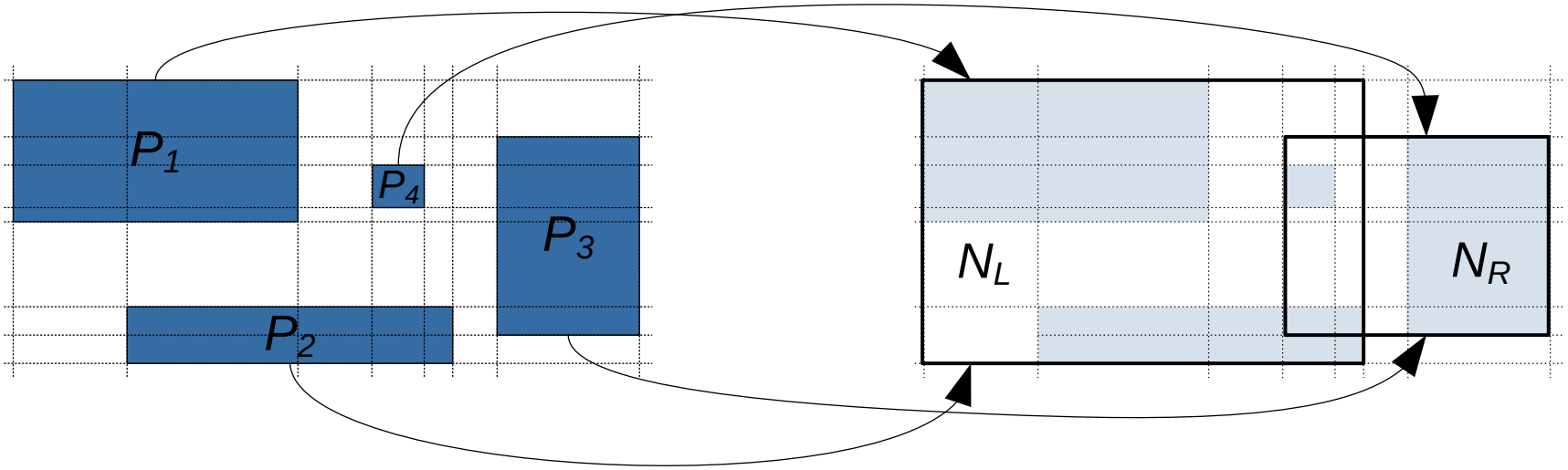
- CBVH split
  - Cost ≈ 700
- Optimal partition
  - Cost ≈ 100

Geometric Partitioning

- Regular approach: Partition *N*'s primitives
  - Evaluate *AABB*s, and use to compute cost
  - $O(2^N)$ partitions to test

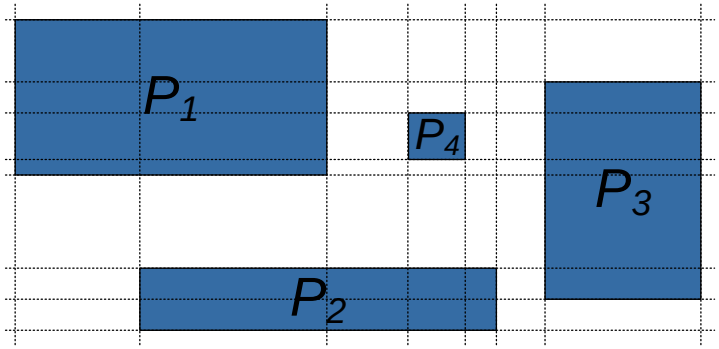
- Geometric partitioning:
  - Fix child *AABB*s and put primitives according to SAH
  - Some configurations are infeasible
  - Child *AABB* boundaries ≡ boundaries of primitives
  - $O(N^{12})$ configurations to test

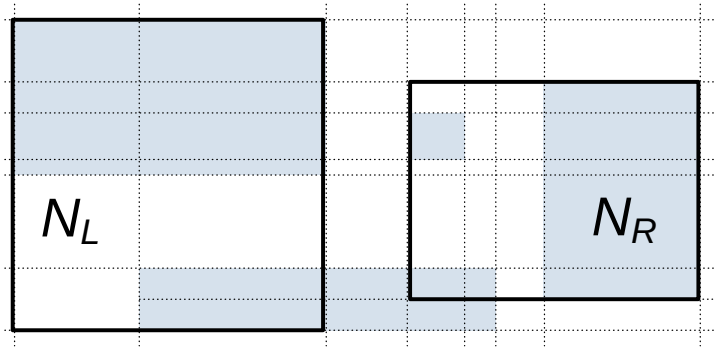
Geometric Partitioning Example



- Boundaries of $N_{L\ or\ R}$ incident with dotted lines
- P_4 shared → put into node with smaller SA

Geometric Partitioning Example

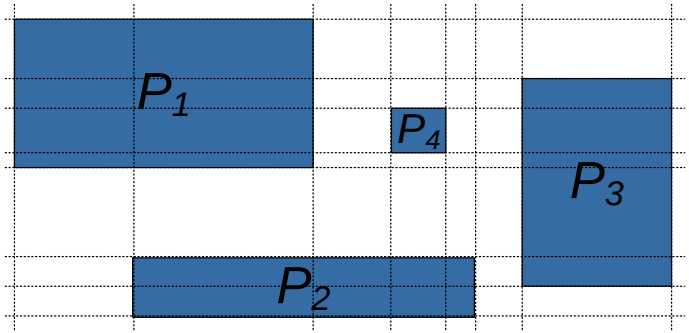


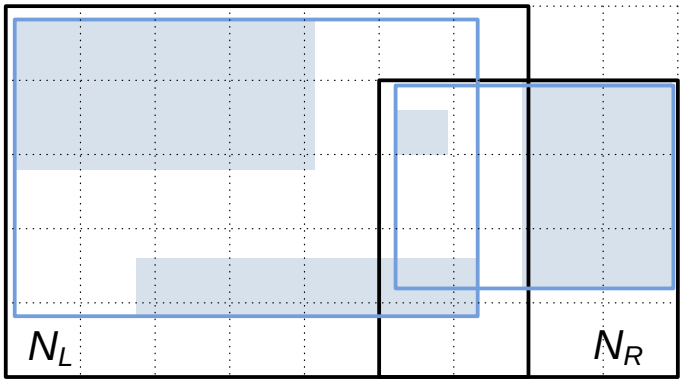


- Configuration infeasible
  - P_2 is not covered

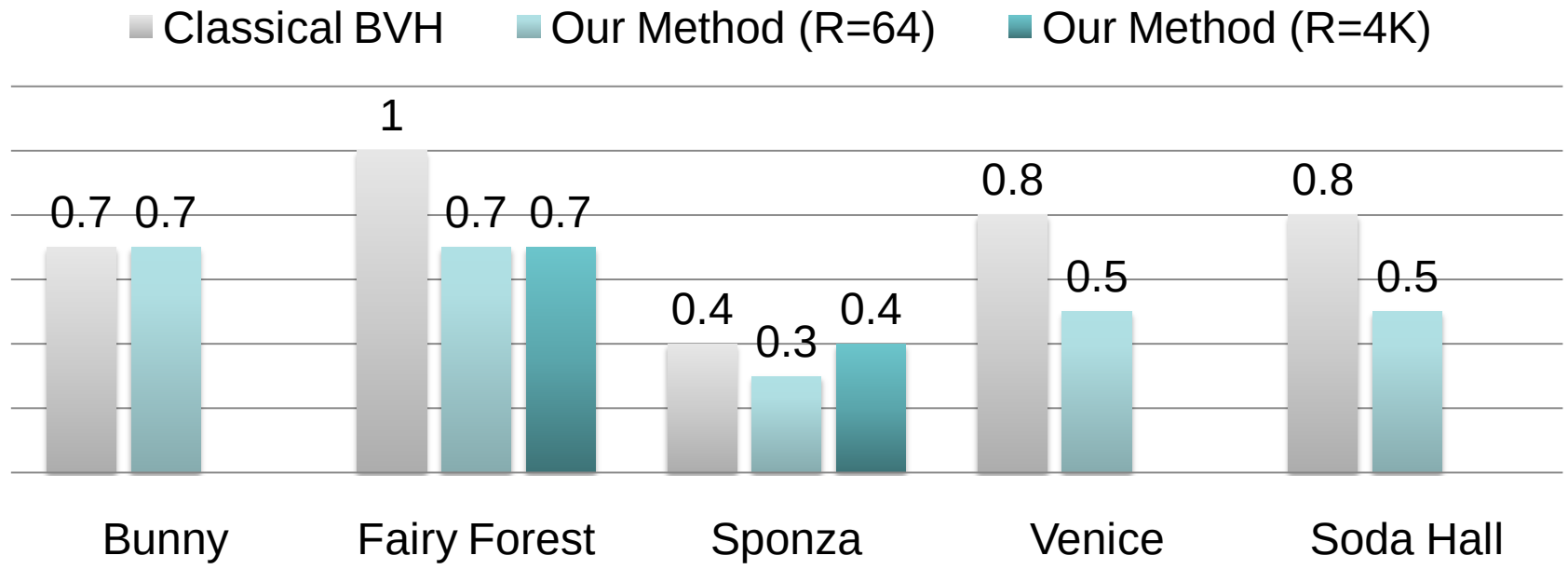
Practical Considerations

- $O(N^{12})$ is actually $O(N^6)$
  - Each side of the parent *AABB* is inherited by a child
- Select child *AABB*s on a regular grid
  - Run-time: $O(G^6 N^{0.5})$ including cost calculation
    - Choosing $G=RN^{1/6}$ yields $O(N^{1.5})$
  - Look at CBVH configurations as well

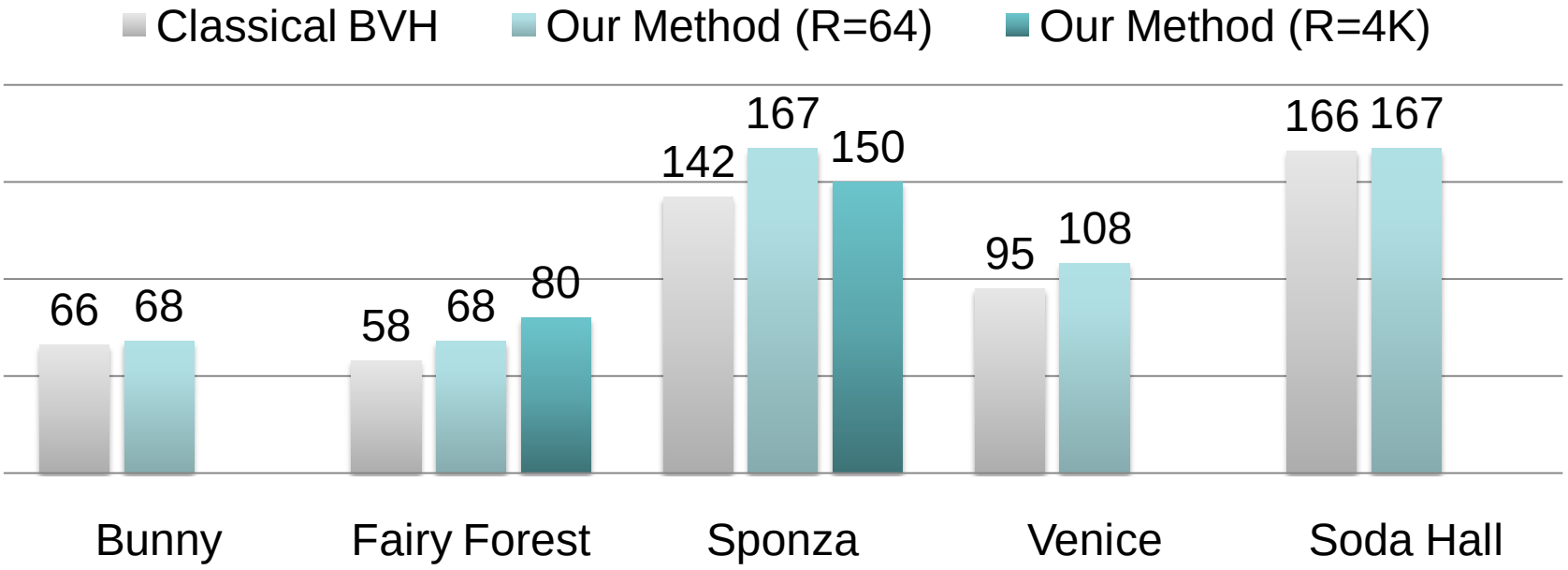




Results: FPS Random Rays



Results: Surface Area Cost

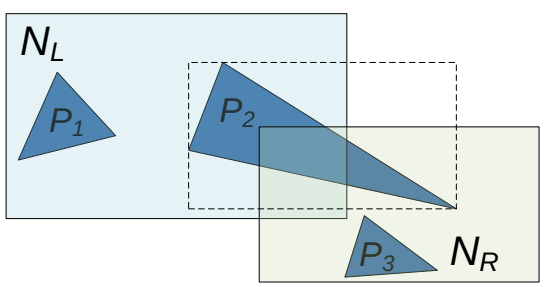


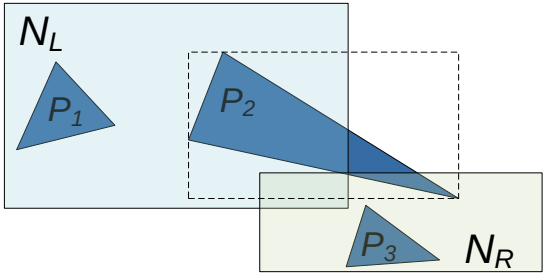
Result Analysis

- Suspect: SAH
  - Overlap + locally minimizing SAH has adverse effect
- Experiment: Use recursive cost evaluation
  - Tree cost better than CBVH but slower FPS!
- Hypothesis: SA model needs space partitioning
  - Intuition: Early ray termination
- New algorithm
  - Penalize overlap in cost function
  - Refine search space by allowing primitive splitting

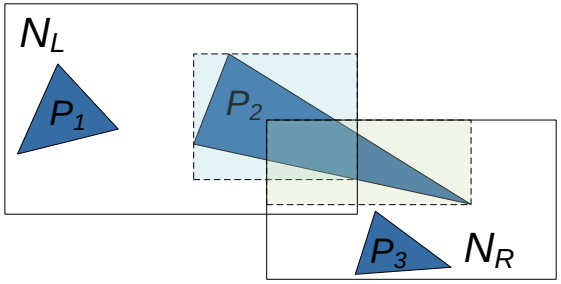
Splitting Primitives

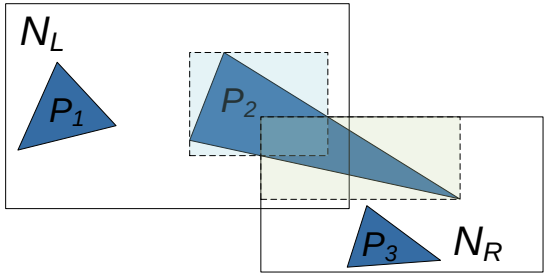
- Feasible and infeasible configurations





- Two possible ways to split a primitive

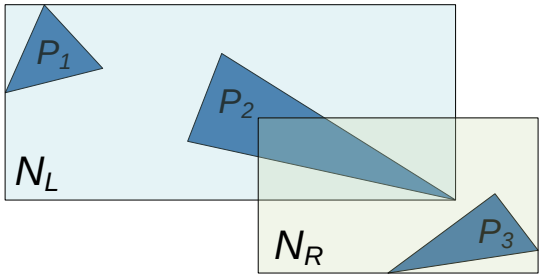


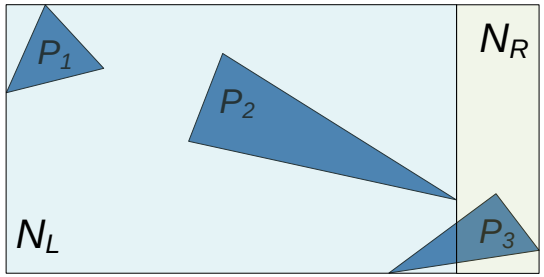


  - SAH cost is the same

Search Spaces

- Child *AABB*s $\in$ continuum inside parent's *AABB*
  - Not limited to boundary of primitives anymore
- Limit search to a grid for practical purposes
- Augment with search space of other algorithms
  - CBVH & KD-tree construction search spaces





Penalizing Overlap

- Bias SAH to account for overlap

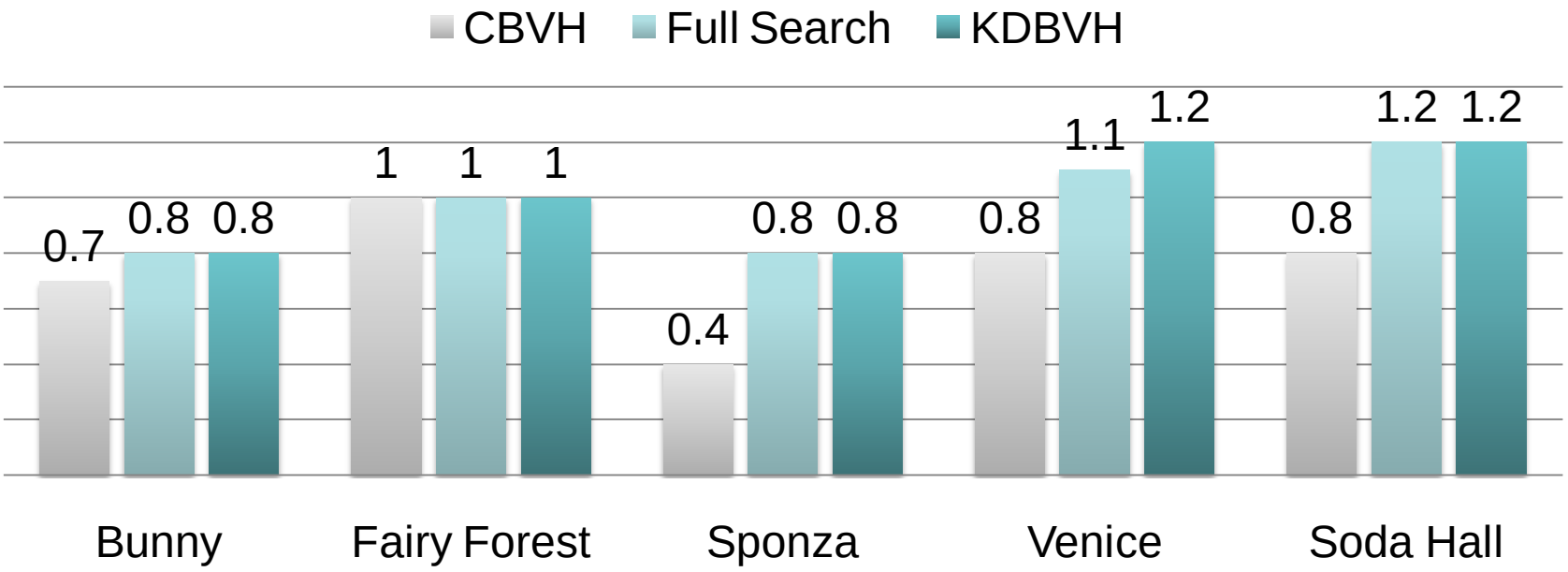
$$cost(N) = C_T + \left(1 + C_O \frac{V(N_L \cap N_R)}{V(N)}\right) \frac{SA(N_L)|N_L| + SA(N_R)|N_R|}{SA(N)}$$

- C_O – the overlap penalty
  - Standard SAH: $C_O = 0$
  - Standard SAH with space partitioning: $C_O \to \infty$

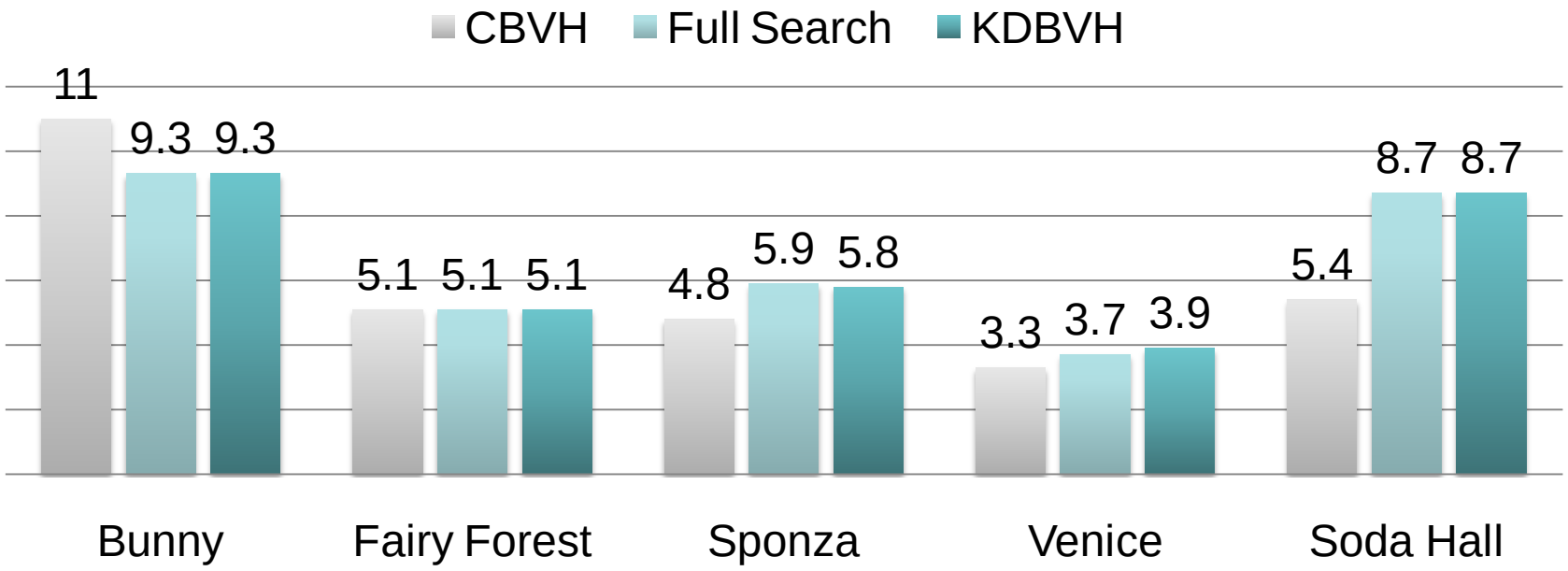
The Generic Algorithm

- Parameters:
  - Search space
  - Overlap penalty
- Algorithm
  - Take configuration $\in$ search space with lowest cost
- Interesting parameters
  - CBVH: BVH, $C_O = 0$
  - Full: Grid + KD tree + BVH, $C_O \rightarrow \infty$
  - KDBVH: KD tree, C_O irrelevant

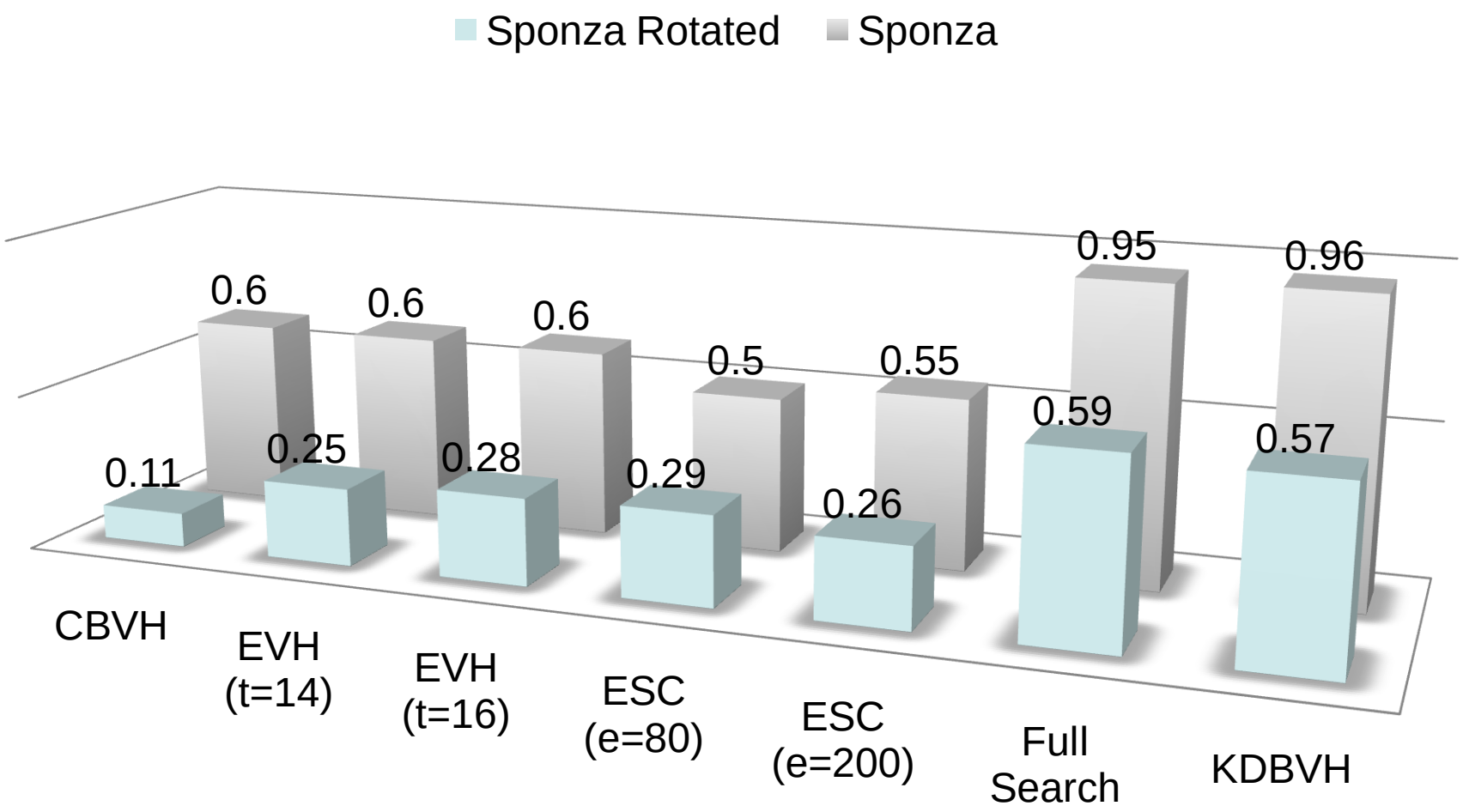
Results: FPS Random Rays



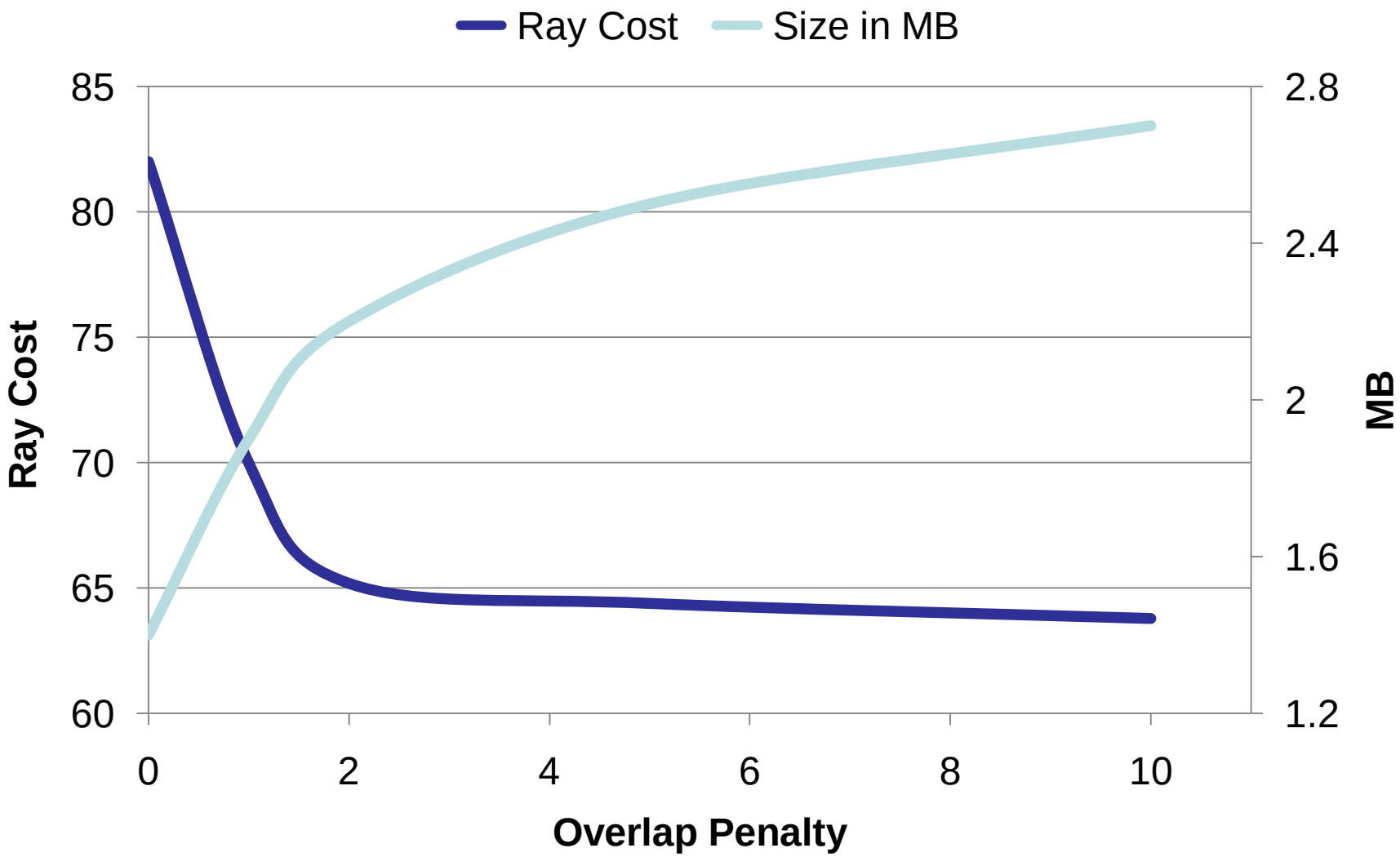
Results: FPS Frustum Traversal



Comparison to Pre-Splitting

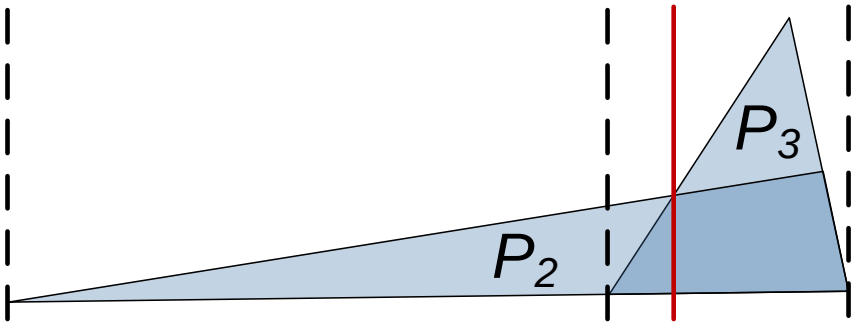


Role of Overlap Penalty



Spatial Build Algorithm

- Implement KDBVH using sweep plane
- Extensions:
  - Combine with CBVH to control size using C_O
  - Sampling of cost function
- Issues: Might miss cost minimum
  - Cost is quadratic between split plane positions



Conclusion & Future Work

- SAH inadequate without space partitioning!
- Generic framework to study BVH construction
  - Can explore full 2^N search space
- Spatial build algorithm
  - Fast with near optimal results
- Research early termination aware cost function

Thank you!