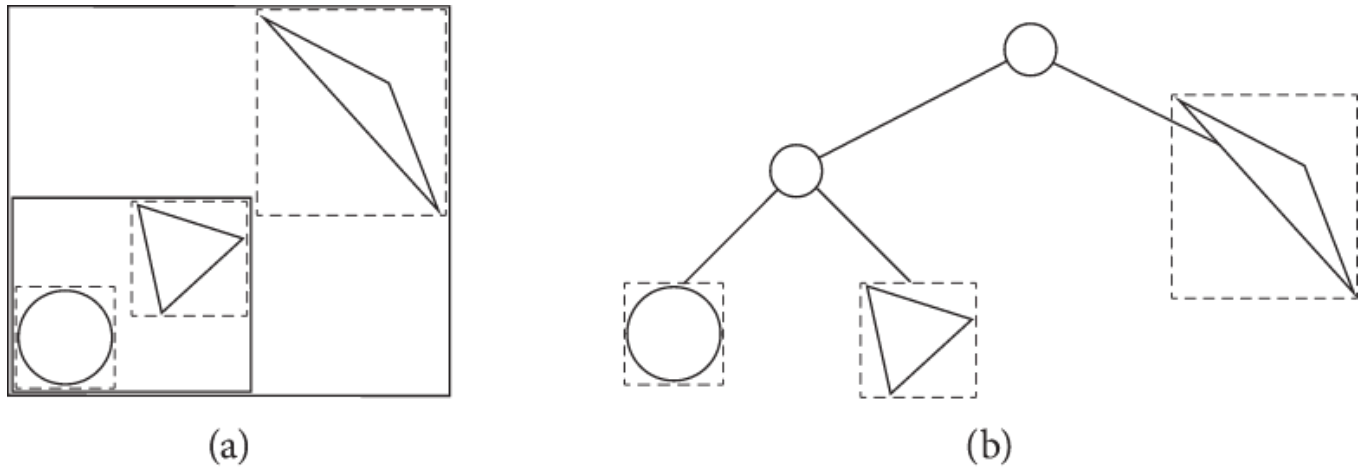
Sequential and Parallel Construction of Bounding Volume Hierarchies

Hoang Duong Amir Sabbag

June 11, 2026

Why BVHs?

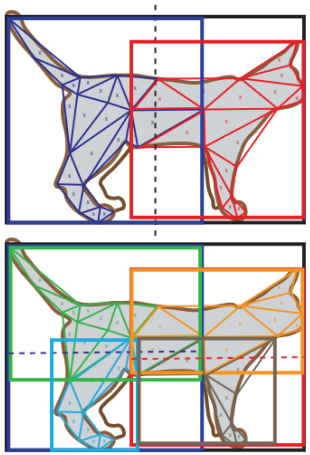


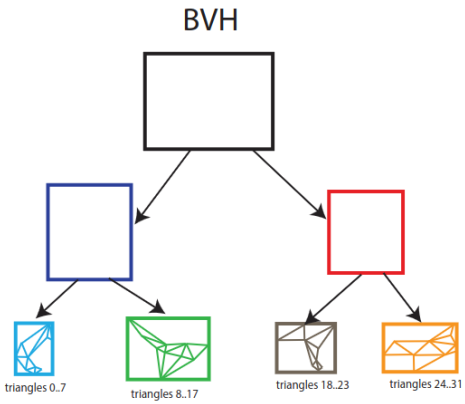
- Ray tracing: millions of rays $\times$ millions of triangles
- Tree of nested bounding boxes — ray misses a box $\Rightarrow$ skip everything inside
- Trade-off: **build time** vs **traversal quality**

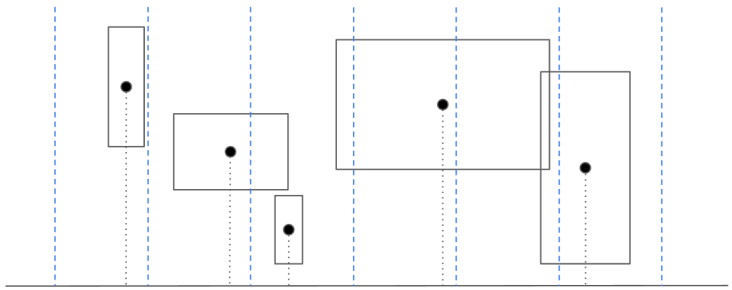
Render goal: `Maria_C` (∼816k triangles)



Sequential build: recursive splitting + SAH

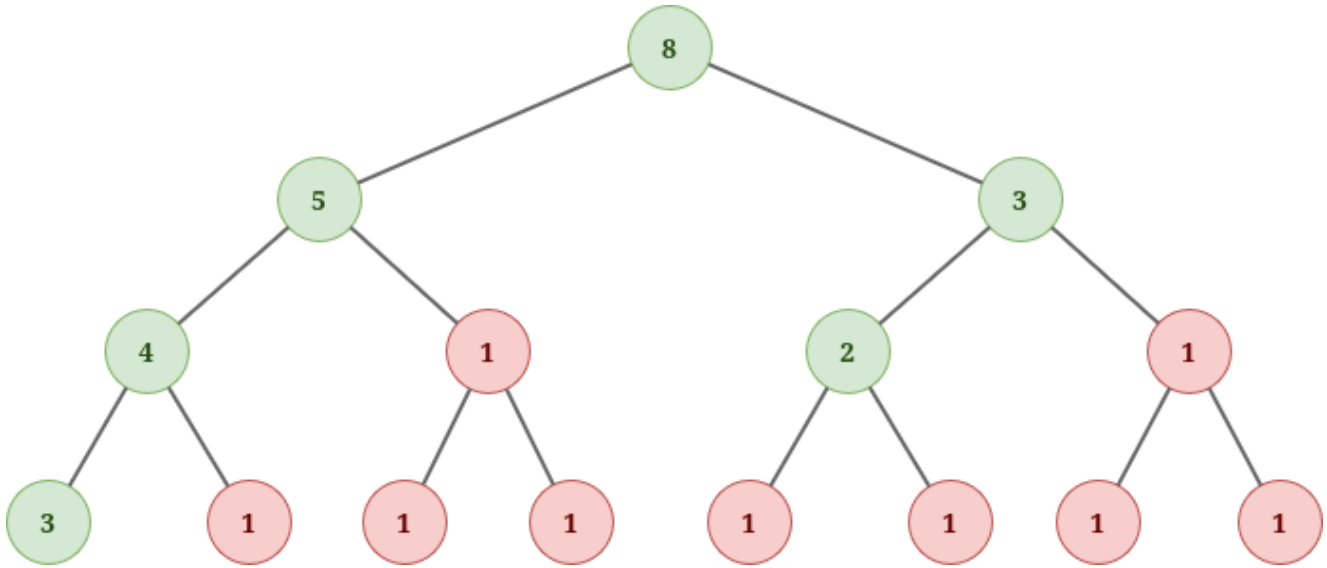






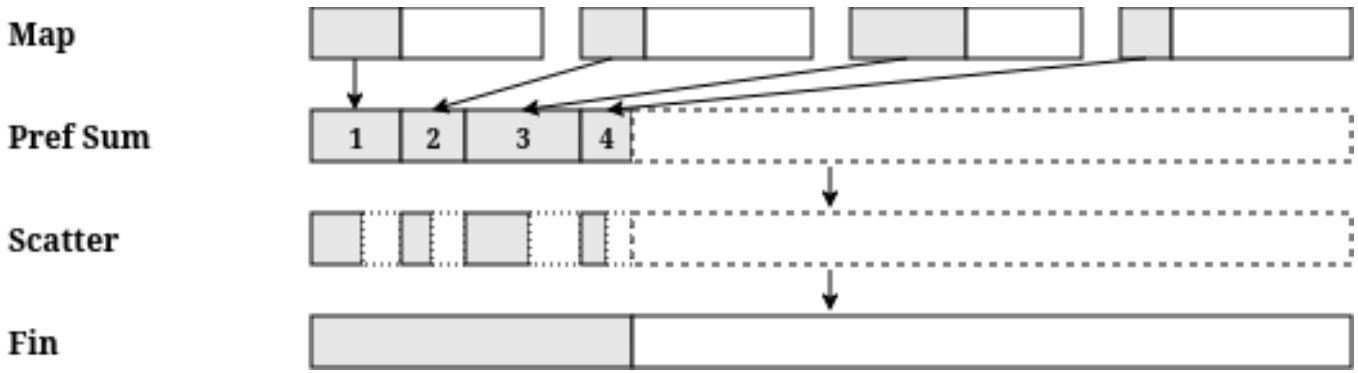
- Top-down: split primitives at median centroid along longest axis
- Surface Area Heuristic (16 bins) picks the cheapest split: $C = \frac{A_L}{A_P} N_L + \frac{A_R}{A_P} N_R$

Top-down parallel BVH (C++): thread budget



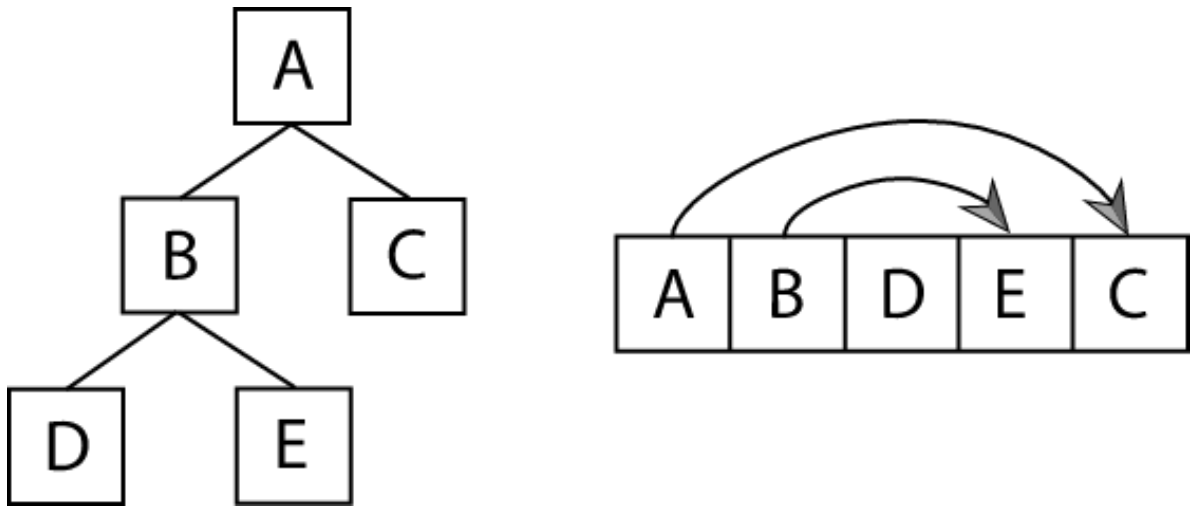
- Threads split between children proportionally to primitive count
- Upper levels: data parallelism (map-reduce SAH binning)
  Lower levels: task parallelism over independent subtrees

Parallel BVH: lock-free SAH binning & partitioning



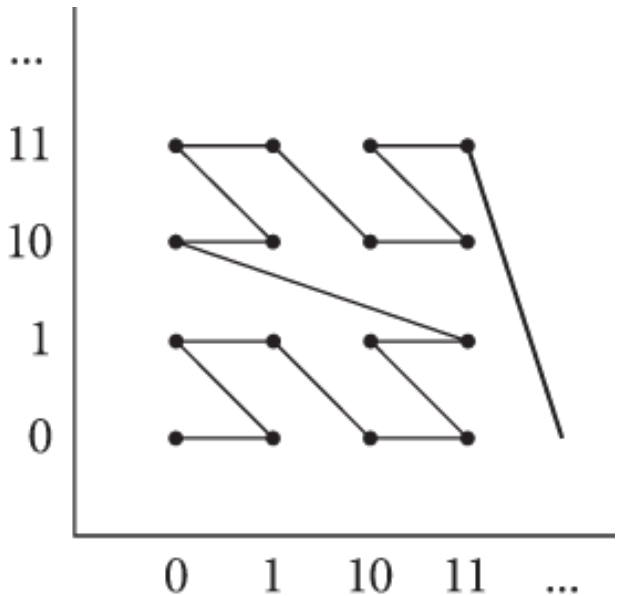
- SAH binning via **map-reduce**: thread-local bins, cheap sequential merge — zero contention
- Partitioning: count → prefix sum → scatter; prefix sum gives each thread disjoint destination ranges, so writes are 100% lock-free
- Nodes allocated from a pre-allocated flat array via one atomic counter

Parallel BVH: tree flattening



- Concurrent node allocation fragments the tree in memory ⇒ cache misses during traversal
- Post-build depth-first flattening pass restores a contiguous, cache-friendly layout

Linear BVH (Rust): Morton codes



-	-	-	-	x_9	-	-	x_8	-	-	x_7	-	-	x_6	-	-	x_5	-	-	x_4	-	-	x_3	-	-	x_2	-	-	x_1	-	-	x_0

-	-	-	y_9	-	-	y_8	-	-	y_7	-	-	y_6	-	-	y_5	-	-	y_4	-	-	y_3	-	-	y_2	-	-	y_1	-	-	y_0	-

-	-	z_9	-	-	z_8	-	-	z_7	-	-	z_6	-	-	z_5	-	-	z_4	-	-	z_3	-	-	z_2	-	-	z_1	-	-	z_0	-	-

-	-	z_9	y_9	x_9	z_8	y_8	x_8	z_7	y_7	x_7	z_6	y_6	x_6	z_5	y_5	x_5	z_4	y_4	x_4	z_3	y_3	x_3	z_2	y_2	x_2	z_1	y_1	x_1	z_0	y_0	x_0

- Interleave centroid bits $\rightarrow$ one 30-bit integer key
- Nearby points $\Rightarrow$ similar codes: a sort *is* the spatial grouping

LBVH: parallel radix sort

Why radix? Morton codes are fixed-width integers — no comparisons needed

LSD radix sort, 6-bit digits $\Rightarrow$ 64 buckets, 5 passes over 30 bits: $O(5n)$, linear.

Each pass, parallelized with Rayon

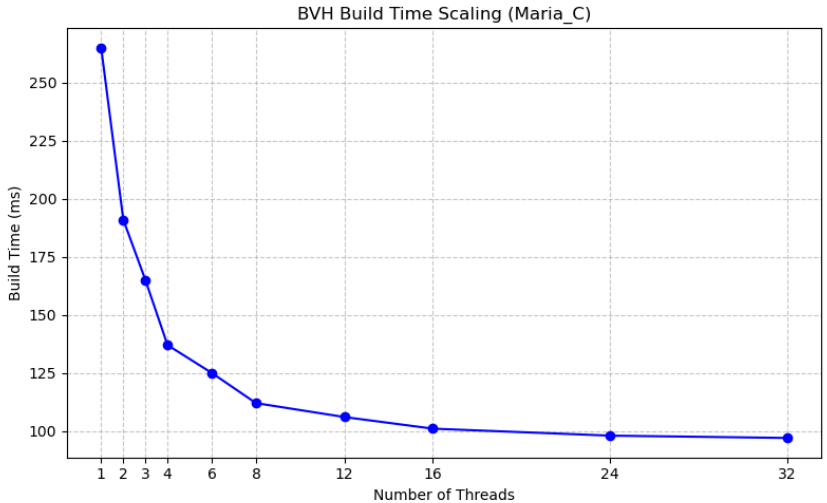
per-thread **histograms** $\rightarrow$ **prefix sum** of bucket offsets $\rightarrow$ parallel **scatter**

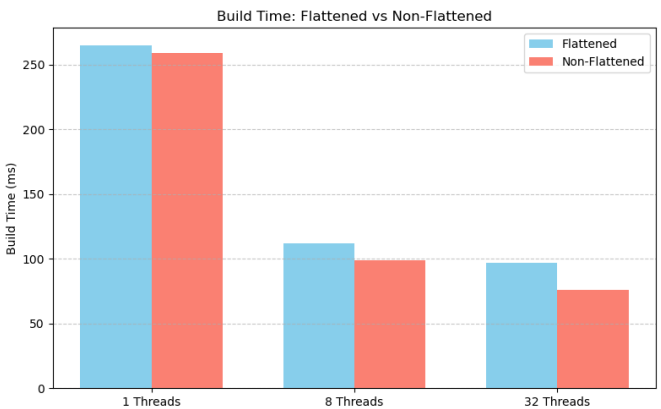
- Predictable linear scans — cache-friendly, unlike comparison sorts
- Memory-bandwidth bound: records slimmed to (code, index), bounding boxes stored separately

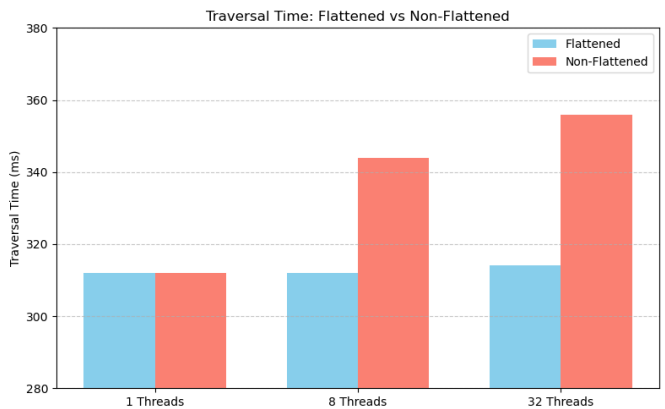
LBVH: tree topology from sorted codes (Karras 2012)

- n leaves + $n-1$ internal nodes $\Rightarrow$ whole array allocated up front, every node's slot known
- Each internal node finds its primitive range and split **independently**, by comparing common prefixes of neighboring Morton codes
- No dependencies between nodes: all $n-1$ internal nodes built **simultaneously** — embarrassingly parallel, ideal for GPUs
- Final pass: propagate bounding boxes up the hierarchy

Benchmark: parallel BVH

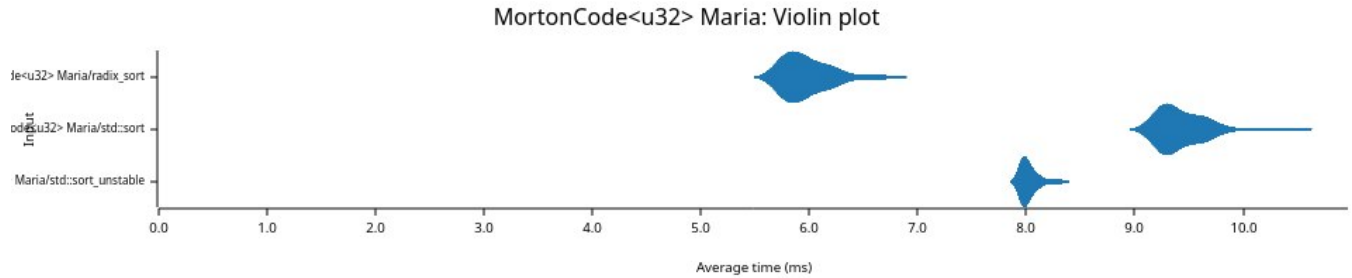


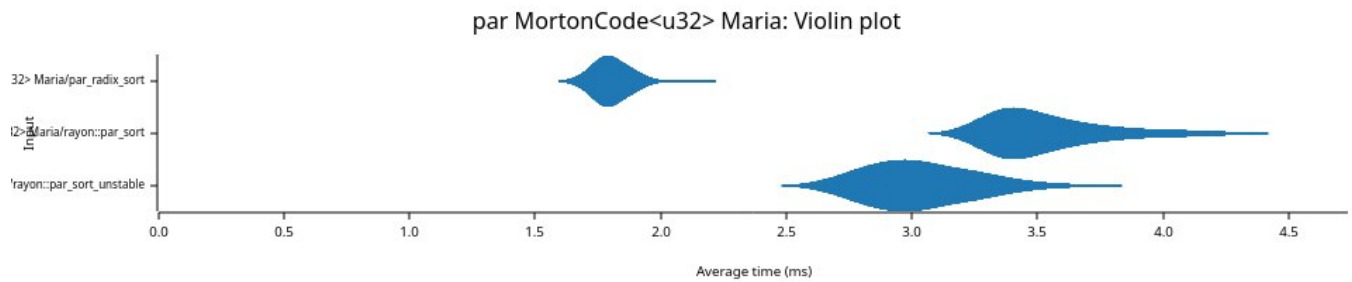


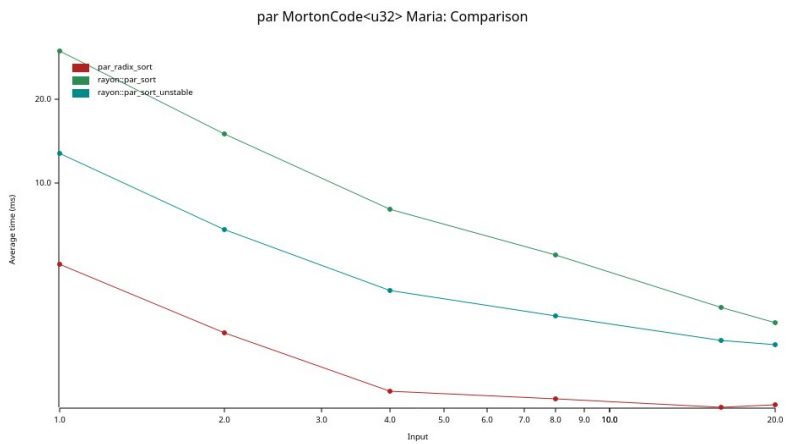


- Build: 265ms $\rightarrow$ 97ms; flattened traversal stays $\sim$312ms at any thread count

Benchmark: radix sort vs comparison sorts

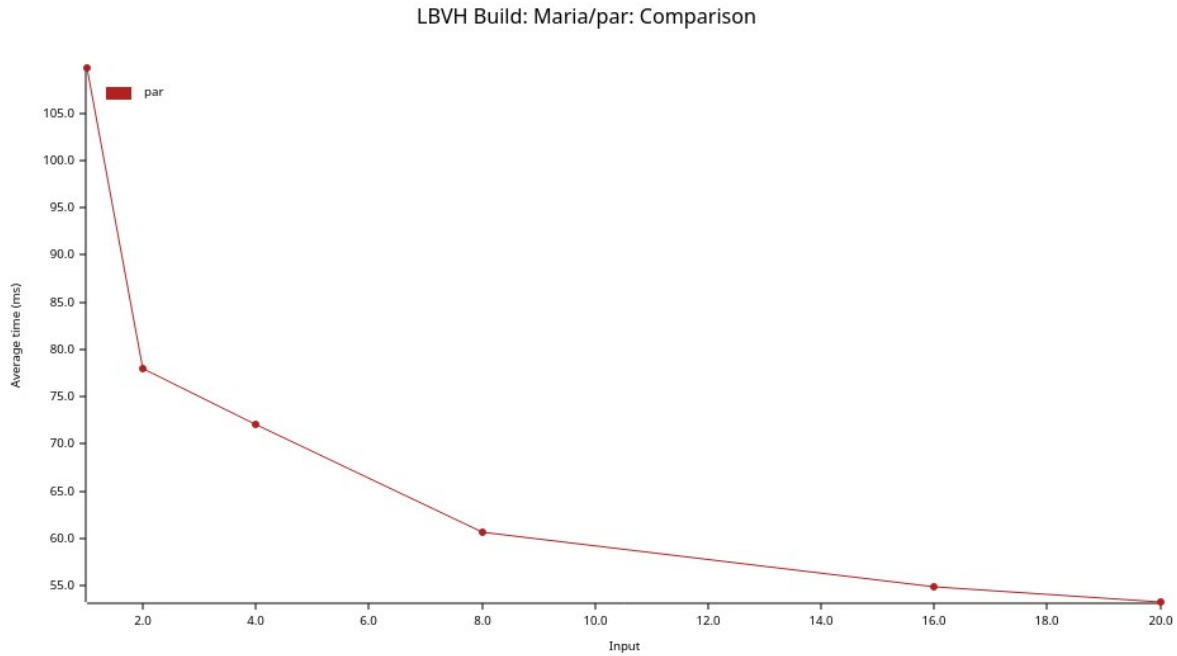






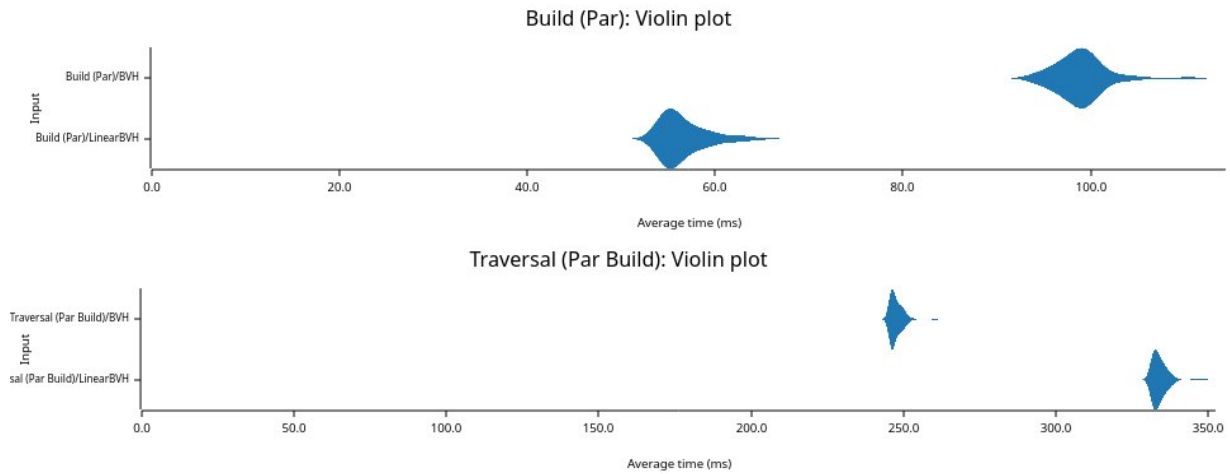
- Radix sort wins sequentially *and* in parallel, at every thread count up to 20

Benchmark: LBVH build scaling



- 110ms sequential $\rightarrow$ 62ms at 8 threads $\rightarrow$ 53ms at 20
- Same shape as the parallel BVH curve, but from a much lower starting point

Benchmark: BVH vs LBVH



- LBVH builds much faster; pays ∼60ms in traversal from a lower-quality tree

Conclusion

- **SAH BVH**: better trees, slower build $\Rightarrow$ static scenes
- **LBVH**: much faster, massively parallel build $\Rightarrow$ dynamic / real-time
- Flattening to contiguous memory is essential after parallel construction
- Cache-aware layout and precomputation alone gave $\sim$50% speedups

References

[1] T. Karras. *Maximizing parallelism in the construction of BVHs, octrees, and k-d trees*. EGGH-HPG'12, 2012.

[2] I. Wald. *On fast construction of SAH-based bounding volume hierarchies*. IEEE Symposium on Interactive Ray Tracing, 2007.

[3] I. Wald, T. Ize, S. G. Parker. *Fast, parallel, and asynchronous construction of BVHs for ray tracing animated scenes*. Computers & Graphics 32(1), 2008.

[4] M. Pharr, W. Jakob, G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. `https://www.pbr-book.org/`, 2024.

[5] G. E. Blelloch. *Prefix Sums and Their Applications*. CMU-CS-90-190, 1990.

[6] M. Harris, S. Sengupta, J. D. Owens. *Parallel Prefix Sum (Scan) with CUDA*. GPU Gems 3, Ch. 39.

[7] C. Zhang. *Rendering Notes: BVH*. `https://blog.zcy.moe/en/blog/bvh/`, 2022.