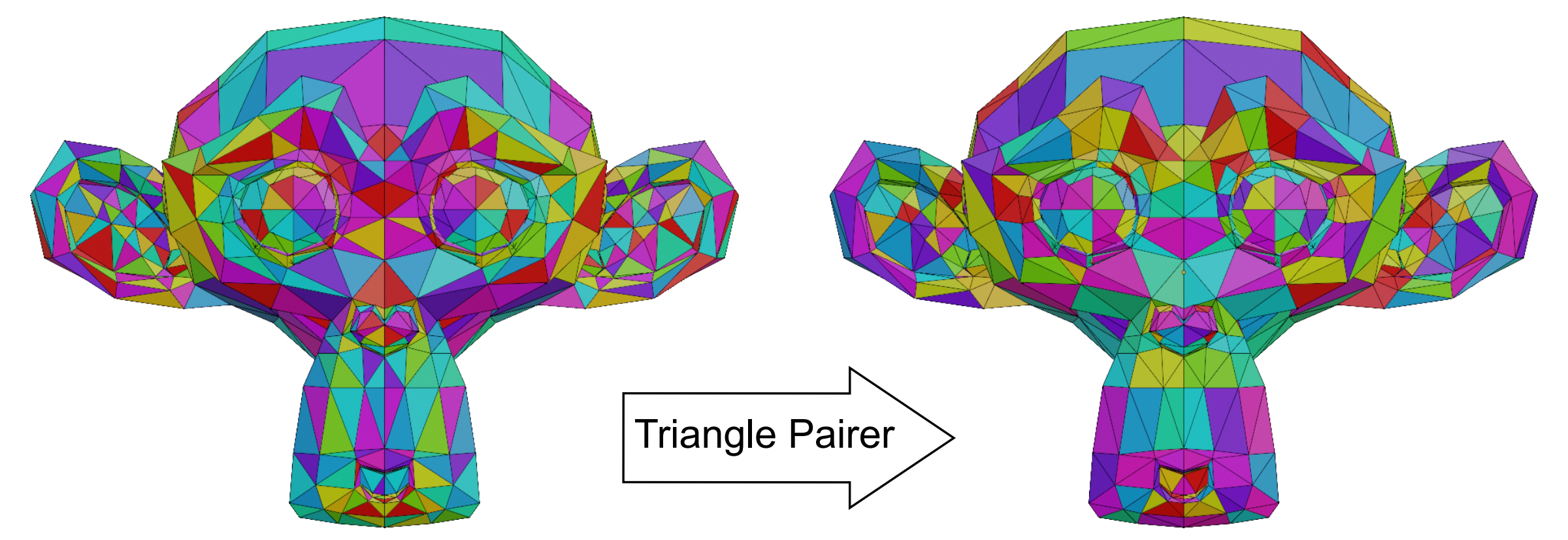
FAST TRIANGLE PAIRING FOR RAY TRACING

Aytek Aman & Simon Fenney Imagination Technologies

INTRODUCTION

We propose a fast and efficient triangle pairing algorithm to increase the utilisation of ray tracers that test triangles in pairs. This method globally reorders the triangles in a way that the number of pairs is maximized while keeping the AABBs of these pairs as small as possible.

- **This increases the utilization of a dual triangle testing unit**
- **Reduces the memory footprint of the acceleration structure**
- **Reduces the number of box and triangle tests**



BACKGROUND

At least two hardware ray tracing systems, Intel's[2] and Imagination's, store and/or test rays against triangle pairs. Testing triangles in pairs has considerable advantages[1]:

1. BVHs tend to use AABBs. The AABB for a contiguous pair of triangles is rarely much larger than either of the two AABBs of the constituent triangles. Treating them as a pair will thus usually reduce BVH footprint, traversal depth, and AABB testing effort.
2. It reduces the number of *false* hits, where ray hits the bounding box but not the triangles.
3. When performing the ray triangle pair tests, the maths relating to the common edge can be shared in the evaluation.
4. A triangle pair is more compact than 2 individual triangles, which can save memory bandwidth and increase cache utilisation.

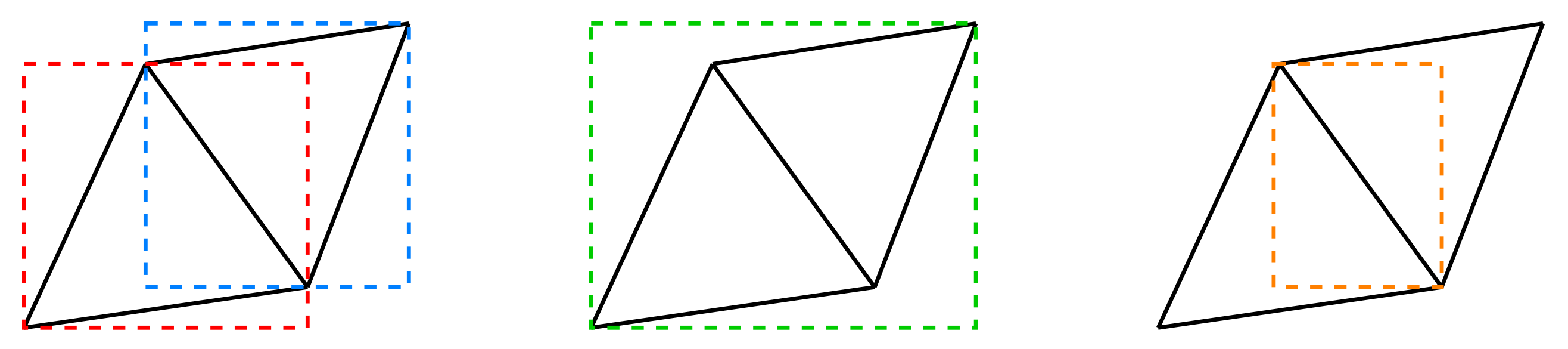
Some ray tracing systems rely on well ordered triangles to form pairs. Usually, a window of size *N* is considered during the scan of the triangles, and best partner for a given triangle needs to be selected from that window. However, this approach does not always give good results:

1. A pair may not always be located in the search window.
2. For small to moderate *N*, the located pair candidate may not be the best candidate for pairing. It can increase a leaf bounding box size unnecesarily and may prevent a better pairing to be made.
3. There are also cases where meshes are 100% paired (e.g. a quad based mesh). However, if such mesh has certain characteristics (a 45 degree rotated model where the rotation is pre-baked into the vertices), original pairings can lead to suboptimal BVH performance.

To solve these issues, we propose a more comprehensive approach, where more pairings are considered by using a surface area heuristic based on shared edge bounding boxes.

EDGE BBOX SURFACE AREA HEURISTIC

The Triangle pairing algorithm aims to pair triangles on shared edges where the surface area of the bounding box of the shared edges are maximized in a greedy manner. This algorithm does not consider the bounding boxes of the triangles and only uses the shared edge data to improve the runtime. Experiments show that this has negligible effect on the quality of the pairings.



Left: The bounding box of each triangle. Middle: Combined bounding box of the triangle pair. Right: The bounding box of the shared edge.

This heuristic simultaneously prioritizes the pairing of triangles with these features:

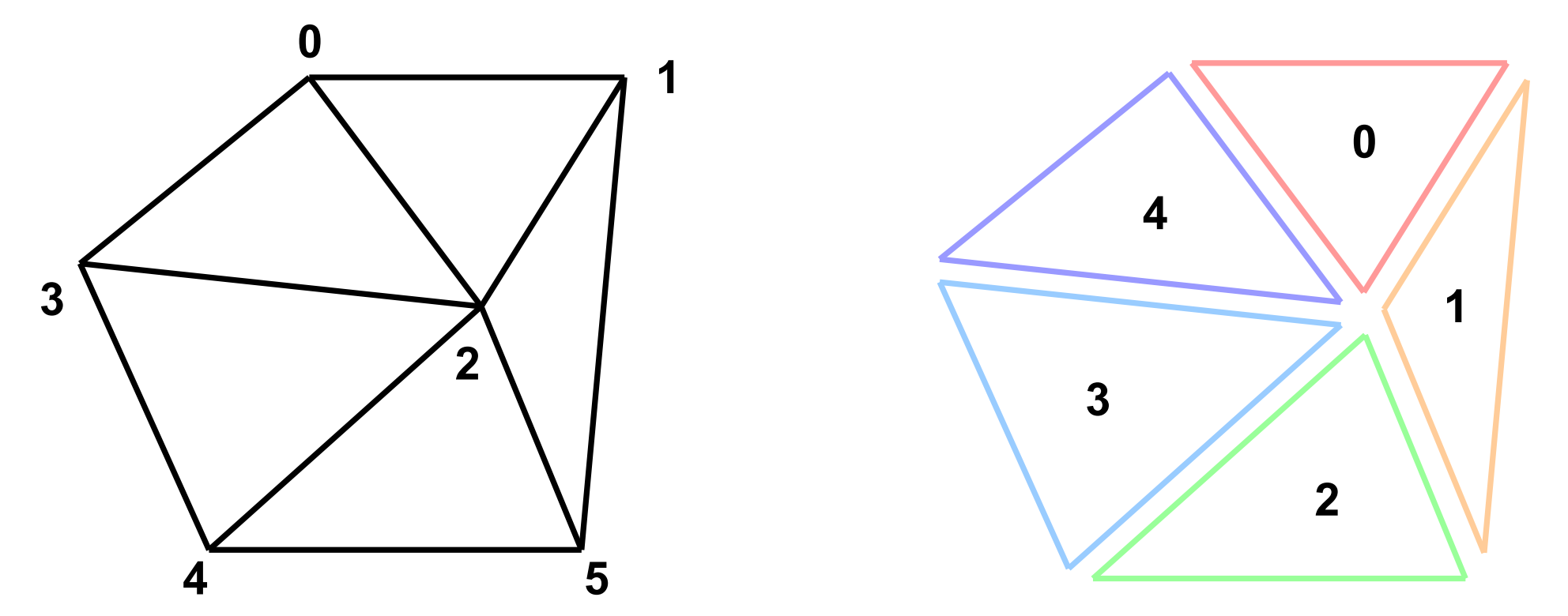
1. **Triangles where shared edge is "diagonal":** Having a diagonal shared edge (relative to the primary axes) increases the chance that the combined bounding box of the triangles will be compact.
2. **Triangles with larger area:** Larger triangles will be paired first to minimize their negative impact on ray tracing performance.

Furthermore, this heuristic can be computed in a trivial manner using the coordinates of an edge. It does not rely on locating the neighboring triangle on the given shared edge.

Our method has four main stages.

1. Halfedge table construction
2. Edge key compression [Optional]
3. Halfedge table sorting
4. Greedy pairing

Given a triangle mesh with 6 vertices and 5 triangles, triangle pairing method will work as follows.



HALFEDGE TABLE CONSTRUCTION

The half-edge table is constructed, containing each half-edge in the mesh. Each half-edge record contains

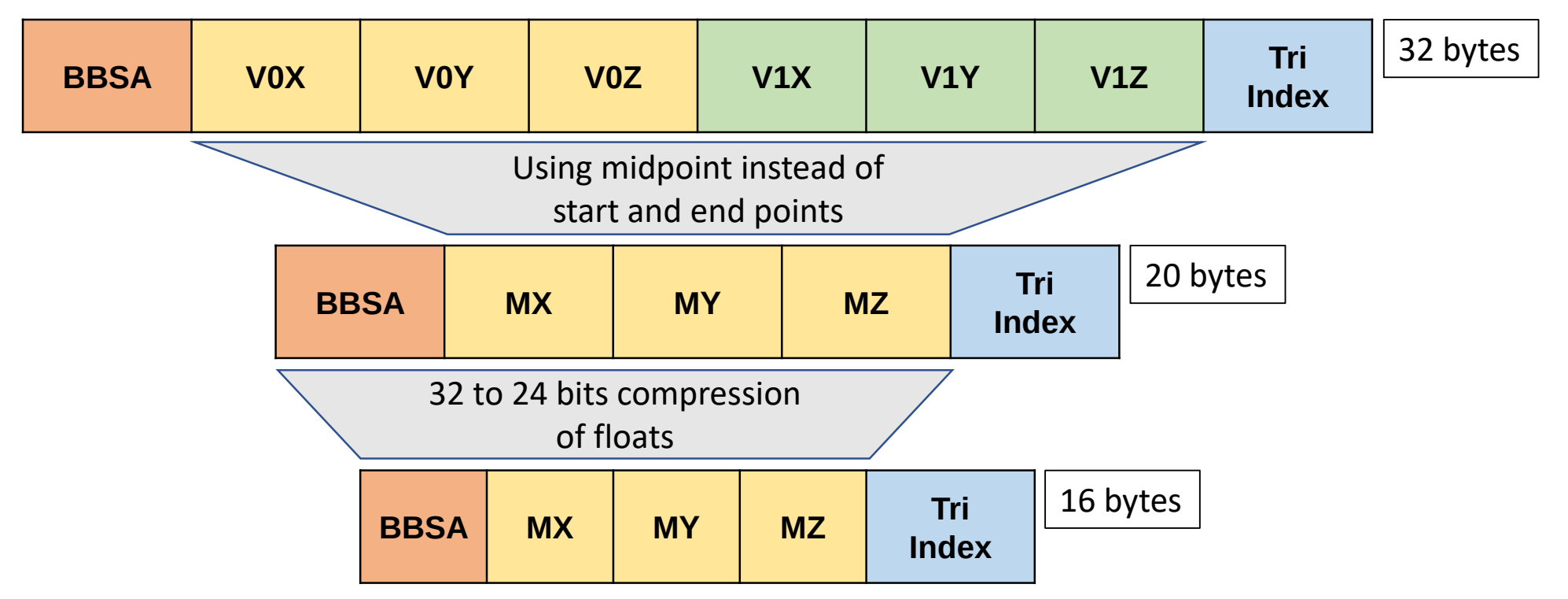
- **BBox Surface Area (BBSA):** Surface area of the axis-aligned bounding box of the half-edge
- **First Vertex Index:** Smaller vertex index of the edge
- **Second Vertex Index:** Larger vertex index of the edge
- **Triangle Index:** Index of the triangle that the edge belongs to

Resulting halfedge table where each column is an halfedge. (Color indicates the parent triangle)

BBox Surface Area	0	15	20	10	10	15	0	30	10	18	9	30	20	20	9
First Vertex Index	0	1	0	1	2	1	4	2	2	3	2	2	0	0	2
Second Vertex Index	1	2	2	5	5	2	5	4	5	4	3	4	3	2	3
Triangle Index	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4

EDGE KEY REPRESENTATION & COMPRESSION

In a ray tracing pipeline, vertex indices may not be available in all stages. Our method supports the use of vertex coordinates directly in such cases. This naturally leads to a larger half-edge table. However, it is possible to compress the half-edge data. For compact representation, we use the midpoint of the edge. Also, both BBSA and edge midpoint coordinates could be converted to a 24 bit representation. As a result, a halfedge entry becomes a 16 byte structure in total.



HALFEDGE TABLE SORTING

The half-edge table is sorted lexicographically using keys **[BBox Surface Area, First Vertex Index, Second Vertex Index]**. After sorting:

- Half-edges will be ordered based on their BBox Surface Area.
- Twin half-edges will be next to each other since their vertex indices are in a canonical form, i.e., First Vertex Index < Second Vertex Index for all half-edges.
- Axis aligned half-edges will be at the end of the table as their BBox Surface Area is zero.

BBox Surface Area	30	30	20	20	20	18	15	15	10	10	10	9	9	0	0
First Vertex Index	2	2	0	0	0	3	1	1	2	2	1	2	2	4	0
Second Vertex Index	4	4	3	2	2	4	2	2	5	5	5	3	3	5	1
Triangle Index	2	3	4	0	4	3	0	1	1	2	1	3	4	2	0

GREEDY PAIRING

A linear scan is performed to identify triangle pairs. If keys of two halfedges are identical (which means that they are halfedge-twins), we check to see whether incident triangles are paired already. If not, we mark these two triangles as a pair.

BBox Surface Area	30	30	20	20	20	18	15	15	10	10	10	9	9	0	0
First Vertex Index	2	2	0	0	0	3	1	1	2	2	1	2	2	4	0
Second Vertex Index	4	4	3	2	2	4	2	2	5	5	5	3	3	5	1
Triangle Index	2	3	4	0	4	3	0	1	1	2	1	3	4	2	0
Pairing Status	Accept			Accept			Reject		Reject			Reject			

RESULTS

We evaluated the performance of our CPU based implementation on different scenes. We report time, paired triangle ratio before (where search window size $N = 2$) and after optimisation, box and triangle test counts and finally million rays per second (MRPS) that is measured on a hardware ray tracing simulator.

Scene	Time (sec)	Old #pairs (%)	New #pairs (%)	∆Box Tests (%)	∆Tri Tests (%)	∆MRPS (%)
bistro	0.077	92.76	94.09	-0.41	-0.76	-0.54
bmw	0.003	99.61	99.78	0.24	-0.44	-4.99
buddha	0.147	55.37	89.70	-3.85	-21.26	20.88
buddha_quad	-	100.00	100.00	0	0	0
bunny	0.009	65.69	97.30	-12.4	-34.36	11.94
cloudgate	0.024	90.66	94.42	-0.51	-1.03	2.29
cloudgate_ship_only	0.122	92.58	93.72	-1.72	-5.00	-0.03
conference	0.032	98.69	95.34	-1.73	-9.56	11.06
dmdeck	0.093	92.35	94.30	-8.27	-9.82	11.02
dragon	0.134	56.52	89.64	-3.49	-19.76	17.62
dungeon	0.009	80.92	96.36	-5.93	-18.72	7.32
epiccitadel	0.017	93.66	93.27	-10.8	-17.10	11.06
examplemap	0.015	92.34	94.49	-2.32	-10.00	3.98
expedition	0.045	56.89	91.07	-3.42	-14.05	2.88
gallery	0.050	89.73	96.06	4.02	-6.77	4.46
hairball	-	100.00	100.00	0	0	0
hangar	0.034	85.83	93.83	-2.51	-13.83	9.26
pagoda	0.011	85.98	92.80	3.67	-8.50	4.43
pipes	0.022	85.49	94.92	-1.90	-5.04	5.57
prefabs	0.041	84.18	89.06	-4.09	-23.28	15.96
relicoflife	0.139	90.89	94.95	-0.94	-4.72	-0.15
rungholt	-	100.00	100.00	0	0	0
sanmiguel	0.178	85.15	92.21	-7.48	-16.72	12.28
scifi	0.015	84.16	92.17	-14.4	-30.88	2.90
sibenik	0.010	95.93	95.78	-2.72	-3.96	4.81
sponza	0.006	97.72	97.93	-0.19	-3.92	-0.03
stylised	0.005	89.40	94.83	-8.47	-21.01	17.87
toys	0.009	89.43	96.11	-7.86	-21.64	21.55
village	0.005	83.81	88.93	-7.41	-19.93	22.65

Note that our pairer performs a pre-scan to decide if pairing optimisation is needed. In scenes *buddha_quad*, *hairball*, *rungholt* the input data is already well-paired, thus pairer has not been run on these scenes. On average, **we observed 7-8 % increase in ray tracing performance**.

CONCLUSION

High quality triangle pairs can increase ray tracing performance. For models with low quality & few pairs, our method can significantly increase the number of good pairings. This, in turn, increases the ray tracing performance while reducing the memory used by the acceleration structure as well as time spent to build that structure. This method is based on a simple scheme that relies on sorting, which can be further accelerated on the GPU by using radix or bitonic sort.

REFERENCES

[1] Simon Fenney and Alper Ozkan. Compressed opacity maps for ray tracing. *High Performance Graphics*, 2022.

[2] Gruen Holger and Joshua Barzack. A quick guide to intel's ray tracing architecture. *GDC*, 2022.