Additional Material for submission Geometry Presorting for Implicit Object Space Partitioning

Martin Eisemann, Pablo Bauszat, Stefan Guthe, Marcus Magnor

May 25, 2012

1 Memory Requirements for the Two-Level Approach

In Table 1 and Figure 1 we list the memory requirements of our two-level approach. The first n levels of the hierarchy are saved as uncompressed Bounding Volume Hierarchy (BVH) nodes created with the surface area heuristic (SAH). As the SAH can separate nodes, e.g. for large triangles, early in the hierarchy, the number of nodes and therefore the memory requirements can vary strongly depending on the scene. However, the upper limit is $(2^{\text{lvl}} - 1) \cdot 32$ Bytes for lvl uncompressed levels.

Scene	5	6	7	8	9	10	11	12	13	14	15	16
Breaking Lion	1	2	4	8	16	33	65	131	260	515	1,001	1,904
Crytek Sponza	1	2	4	7	14	26	50	94	171	303	535	919
Fairy	1	2	4	8	16	32	61	112	202	354	582	902
Robot Girl	1	2	3	6	11	20	37	69	134	264	513	966
Thai Statue	1	2	3	5	9	17	34	67	132	262	520	1,035
Venice	1	2	4	8	16	33	65	130	255	494	939	1,757

Table 1: Memory requirements for the two-level approach. Numbers are given in kilobytes.

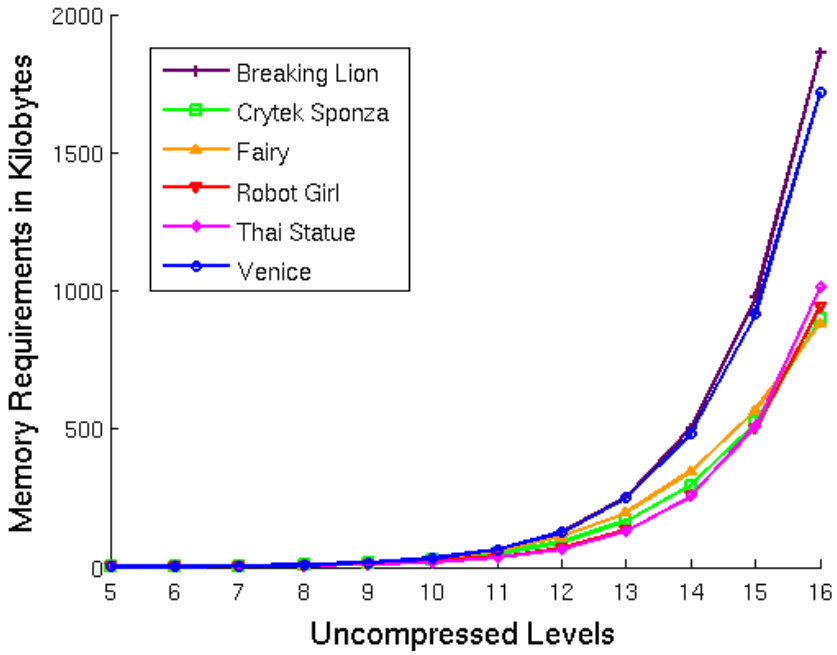

Figure 1: Memory requirements for the two-level approach. Numbers are given in kilobytes.

2 Optimization Strategy

In Section 3.1 we mentioned an optimization strategy to optimize the bounding triangle intersection test and reduce the bandwidth requirements of our data structure. For further clarity we want to provide an example. Let us assume that the bounding axis of our current node is the x-axis. In the first step, we load the x-values of the six vertices belonging to the two bounding triangles of this node. We then reconstruct the bounding planes along the x-axis and test our ray first against the minimum bound plane. This can already result in a rejection of the ray when comparing the intersection parameter with the active ray interval. If the hit is valid, we test against the maximum bounding plane. If again a valid intersection is found, we know that the node was hit and need to estimate whether an intersection with the bounding triangles exists. As the probability of a hit is very low in the first levels of the hierarchy we do not test the triangles directly. Instead, we first reconstruct the remaining bounding planes along the bounding axis for each triangle from the already loaded x-coordinates. If no valid intersection is found the triangle is rejected without further processing. Otherwise, we proceed to the next bounding axis, in this case the y-axis. We load the y-coordinates of the bounding triangles and again reconstruct the bounding planes and test against it. If a valid intersection is found, we proceed to the z-axis, reconstruct the bounding planes and test against it. Only if for all six bounding planes a valid intersection is confirmed, we test the actual triangle.

Please note, that we do not use the optimization for the triangle intersection as it provides no benefit on current graphics hardware, even though the theoretical bandwidth is strongly reduced. Instead, we test the triangles directly for intersection if the node is hit, as mentioned in the text.