Abstract

To eliminate unnecessary ray packet-triangle intersection tests, we check for separation of two convex objects: a triangle and a frustum containing intersections of rays with a leaf node of an acceleration structure. We show a performance improvement that is proportional to the ratio of lengths of average leaf edge to triangle edge, which opens new possibilities for creation of better acceleration structures.

Historical Background

Acceleration structures (kd-trees, grids, BVHs) exploit scene coherency to greatly reduce the number of potential ray-triangle intersection tests in ray-tracing applications. During a scene rendering, the acceleration structure is traversed and all triangles in visited leaf nodes are tested for intersections. The cost of traversing an acceleration structure can be reduced by using packets of rays, which also allows amortization of per-triangle pre-processing costs among all rays in a packet.

Most cases when a ray is not intersecting the axis-aligned bounding box (AABB) of a triangle are eliminated during traversal of an acceleration structure, but even when a ray does intersect the AABB, the probability of it intersecting the triangle is only about 20%. It is important to quickly reject the remaining 80%. Amongst the approaches analyzed in the literature [Boulos et al, 2006] are: use of interval arithmetic; testing the intersection of a frustum containing rays (typically represented as 4 corner rays) with the AABB, the probability of it intersecting the triangle is only about 8.25 CPU cycles per test to 4.3 cycles (including all extra costs). This allows the creation of more compact yet more effective acceleration structures, based on surface area heuristics.

Vertex Culling

We propose the following 3-stage algorithm to find all intersections of a packet of rays with triangles inside a leaf node:

1. Choose a prevalent direction of rays in a packet (one for which the absolute components of direction are bigger). Suppose that it is direction x. Then we find the intersections of rays with the \( x=x_{00} \) and \( x=x_{10} \) planes of the leaf node and compute axis-aligned rectangles containing these intersections for each plane (Figure 1a). These rectangles form a frustum containing the intersections of the rays with the leaf's AABB.

2. For a given triangle, we check whether all three triangle vertices are separated from the frustum by any one of the four frustum planes.

3. If the separation is not found, we look for the intersection of four corner rays with the triangle. Using the Barycentric coordinates of intersection points we can identify the situations where the frustum is separated from the triangle by any one of the triangle edges.

This covers all possible separation cases of two convex objects and allows for a tight frustum, which is different for each leaf (rays could intersect the frustum outside the current leaf's AABB). The second step of the algorithm is remarkable in its simplicity and requires only one S.S.E. multiplication and two additions per vertex. For each packet, we compute two S.S.E. vectors \( q_0 \) and \( q_1 \) as 

\[
\begin{align*}
q_0 &= [x_{00} \ z_{10} \ z_{00} \ x_{10}] \\
q_1 &= [z_{00} \ -z_{00} \ y_{11} \ -y_{01}] \\
\end{align*}
\]

Then for each vertex \( v = [v_x, v_y, v_z] \), the four components of the S.S.E. variable 

\[
d = [v_x, v_y, -v_z, v_z] + [v_z, v_z, -v_x, v_x] \ q_1 + q_0
\]

will be proportional to the distances to the four frustum planes. If at least one component is negative for all three triangle vertices, the triangle is separated from the frustum by the corresponding plane. This can be easily verified using three MOVEMS operations.

Figure 1 (a) frustum parameters (b) CPU cycles per test vs leaf size

Figure 1: (a) culling triangle vertices \( v_0, v_1, \) and \( v_2 \) against frustum containing intersections of rays with a leaf’s AABB; (b) ray – triangle intersection time in CPU cycles (measured on a Core 2 2.4 GHz CPU) as a function of ratio of average leaf edge length to average triangle edge length. Baseline: Kensler and Shirley implementation; \( uv \): barycentric culling alone (step 3 of the algorithm); frustum: frustum culling alone (step 2); frustum + \( uv \): both.

Using exactly the same settings as in Kensler and Shirley [2006], we improve performance of ray-triangle intersection tests from 8.25 CPU cycles per test to 4.3 cycles (including all extra costs required to compute the frustum). In addition, the performance improves as leaf cells grow (Figure 1b). This allows the creation of more compact yet more effective acceleration structures, based on surface area heuristics.

According to our preliminary experiments, by using two additional frustum planes for near and far distances, performance improves by an extra 30%. In the current implementation, we test all vertices separately. If leaf triangles form a mesh, per vertex costs could be amortized. This would make the proposed algorithm even more effective. Another potential improvement is based on using interval arithmetic to compute AA rectangles in the first step of the algorithm.

References
