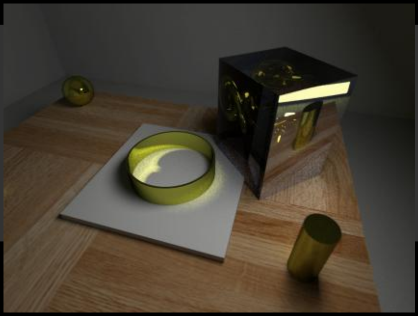
BRIGADE
real-time path tracing



15/6/2010

Jacco Bikker, NHTV University Breda
Dietger van Antwerpen, Technical University Delft

Brigade is an interactive path tracer for games: its purpose is to facilitate the development of ‘proof of concept’ path traced interactive applications on consumer hardware. A hybrid architecture employs both the CPU and the GPU in a single system. The workload is dynamically scaled between the CPU cores and the GPU, keeping each fully occupied.

Dynamic scenes, multiple acceleration structures

The top-level scene BVH is constructed from smaller BVHs for each scene graph node. These nodes in turn are updated per frame using refitting or rebuilding, if needed. Also per frame, the top-level BVH is converted to an MBVH for rapid divergent ray traversal, and to a GPU BVH for GPU rendering. During rendering, the BVH is traversed using packet traversal. Divergent rays are traced using MBVH/RS.

Heterogenous rendering

The CUDA tracer implements a full path tracer: Its input data are the vertices of the view frustum, its output are rendered tiles of pixels. The CPU tracer implements the same interface. The CPU and GPU systems can be extended with others that implement the same interface, such as a network renderer, or an OpenCL module.

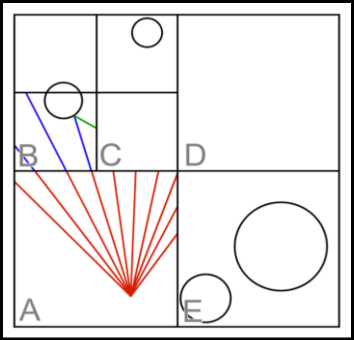
Work is divided over the rendering modules using a balancing system, that adjust the workload assigned to each module based on the amount of time spent in the previous frame. This way, no communication is necessary during frame rendering.

For games

As the successor to the Arauna real-time ray tracer, Brigade will be used by IGAD students to develop games in our GameLab. This introduces students (both programmers and visual artists) to future graphics technology, and thus helps introduce this technology into the game industry. As of today, Brigade is open source.

Fast traversal of incoherent rays

Tracing rays through an octree with neighbour links in the leaf nodes is a competitive alternative to single-ray traversal. We batch rays in the nodes, and advance rays in saturated nodes.

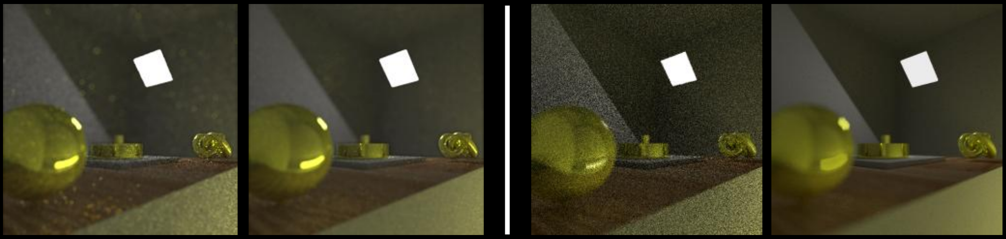


Octree traversal with neighbour links: rays are batched until a node is saturated. Rays of various depths are traced together.

Using a shallow octree, rays batched in leafs can be traversed through an MBVH. Traversal schemes for this data structure benefit significantly from the improved coherency of the rays in the leafs.

Ray depth	**0**	**1**	**2**	**3**	**4**
Mono traversal	2009	1308	1167	1105	1083
Partition traversal	7622	1903	1194	1022	950
Range traversal	11376	1332	721	593	541
Octree traversal	2126	1477	1293	1226	1232
MBVH/RS	4134	2131	1639	1453	1386
Hybrid	3002	2352	2005	1889	1882

Tracing various ray depths using various traversal algorithms



Visualizing light transport with discs. Shown here are 1 and 8 spp using discs, versus 1 and 64 spp using direct plotting.

Post processing

The output of the ray tracer can be post-processed by plotting the transported light as discs (with floating point coordinates) rather than pixels. The radius of each disc depends on the distance to the focal plane. Using this approach, out-of-focus image regions converge quickly. When path probability is also allowed to influence disc radius, caustic paths too effect a greater area. This approach works well in the context of a path tracer, as it is not screen space based.

Models by IGAD students Nils Ruisch & Gabrian van Houdt. Additional coding by Jeroen van Schijndel. Maya exporter by Eric den Boer. Maya exporter, source files and poster art assets can be obtained from http://igad.nhtv.nl/~bikker.