Fast Ray Tracing of Scenes with Unstructured Motion

Pankaj Khanna[*] Jesper Mortensen Insu Yu Mel Slater
Department of Computer Science, University College London, London WC1E 6BT, UK.

1 Introduction

Ray tracing dynamically changing scenes with unstructured motion for animated objects has long been a problem for ray-traversal acceleration schemes. When polygons are transformed independently of one another, the cost of updating traditional spatial data-structures can be quite high [2001]. We propose a ray traversal scheme that is well suited to scenes with dynamically changing objects during ray tracing. A similar data structure for propagation and walkthrough only rendering of globally illuminated scenes was introduced for global illumination in [2004]. Here we concentrate on an application and modification of that data structure for the task of ray tracing scenes composed of static and dynamic objects. The major computation for handling arbitrary transformations of dynamic objects reduces to low resolution 2D polygon rasterisation.

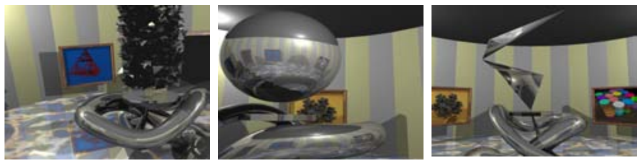
Figure 1. Frames from the animated museum walkthrough.

2 Data Structure and Algorithm

Find the smallest bounding cuboid that encloses the scene, and divide the bottom face (parallel to the *xy* plane) into n×n 2D square tiles. Each tile can be considered as one end of a 3D beam parallel to the z-axis that intersects a number of polygons. The tiling can be represented as a 2D array, where each array element stores a list of identifiers of polygons intersected by the beam. This tiling is very easy and fast to compute – since the scene can be orthographically projected onto the cuboid base, and a 2D scan-line algorithm for each polygon will identify the tiles in which it falls. The whole set of beams is called a 'parallel subfield' (PSF) and ultimately represents the set of all possible rays in the beam direction, parallel to the z-axis, and is called the 'canonical' PSF.

We can consider several PSFs passing through the scene in *l* different directions. The tiling can be computed for any direction by rotating it so that it is coincident with the canonical direction, and then carrying out the polygon fill scan-line algorithm as described above. A set of directions is chosen by a recursive subdivision of a regular tetrahedron. To summarize the data structure: we have *l* distributed directions; associated with each direction is a 2D array of tiles with each tile containing a list of polygon identifiers. This list corresponds to the set of polygons that are intersected by the beam in the given direction. These polygon identifiers in the tile are further sorted into a low resolution one-dimensional BSP tree.

The fundamental operation in ray tracing is to find the object with the nearest intersection along a ray. Given any arbitrary ray through the scene we look up the closest PSF direction and project the ray end-points onto the base of the corresponding PSF to find which tiles it intersects – typically both end-points are in one tile. In the case that a ray spans multiple tiles, these are traversed in a near to far order. These look-up operations immediately identify a very small proportion of the original scene polygons as candidates for intersection. These can then be searched in logarithmic time.

3 Dynamic Object Changes

Polygon identifiers corresponding to the static objects in the scene are written into the tiles as described above. For movable objects, polygon identifiers are also written into the tiles (flagged as dynamic). Following each update to the dynamic objects, a 2D rasterisation at tile-resolution in all directions is performed and polygon identifiers are stored in lists on the overlapping tiles. In each tile there is a variable *V* which determines if the dynamic tiles are valid for the current frame. There is also a global identifier *GV* which is incremented each time the dynamic scene is transformed, automatically invalidating all currently inserted tiles (without having to visit them). Each time the dynamic scene is transformed, it is rendered into the tiles, with the current identifier *V*=*GV*. During this rendering memory corresponding to *V_GV* can be reclaimed. During ray tracing, ray-intersection requests are created. Each ray-intersection is first carried out with the static polygons in logarithmic search time, and then with the dynamic polygons with a linear search, and the closest intersection is found. The polygons to be tested for intersection are limited to those in the relevant tiles as described earlier.

Figure 1 shows frames from the museum7 scene walkthrough [2001] which has 10,284 static and 16,384 dynamic triangles. The accompanying video shows the dynamic part undergoing animation with dynamic triangles moving independently of each other. Our method rendered the 300 frame animation in comparable time on a dual Xeon 2.8Ghz workstation. Currently the main bottleneck in performance is the cost of intersecting the linear list of dynamic polygons in a tile. If these polygons were inserted into a coarse 1D-BSP tree performance would be significantly improved.

The work presented herein is at a very preliminary stage and demonstrates how accelerated ray tracing of dynamic scenes can be broken down to low resolution rasterisation and matrix operations. Implemented on a GPU we expect to increase performance by at least an order of magnitude.

Acknowledgements

This research is funded by the UK EPSRC, grant number GR/R13685/01. Mel Slater is supported by an EPSRC Senior Research Fellowship.

References

[2001] LEXT, J., ASSARSSON, U., AKENINE-MOLLER, T. (2001) A Benchmark for Animated Ray Tracing, *IEEE Computer Graphics and Applications*, pp. 22-31, March/April 2001.

[2004] SLATER, M., MORTENSEN, J., KHANNA, P., YU, I. (2004) A Virtual Light Field Approach to Global Illumination, *Proceedings of Computer Graphics International*, Greece, June 12-19, 2004.

[*] e-mail: [p.khanna | j.mortensen | i.yu | m.slater]@cs.ucl.ac.uk