Department of Computer Science,
University College London,
Gower Street,
London WC1E 6BT.

A Visibility Field for Dynamic Ray Tracing

Pankaj Khanna Jesper Mortensen Insu Yu Mel Slater
(p.khanna | j.mortensen | i.yu | m.slater)@cs.ucl.ac.uk

Department of Computer Science, University College London, London WC1E 6BT, UK.

1 Introduction

We have invented a new ray traversal scheme that is especially suitable for dynamically changing objects during image generation with ray tracing. This algorithm has recently been introduced for walkthrough [Mortensen, et al. 2004], and it compared favorably with the leading algorithm in the field, Coherent Ray Tracing [Wald et al., 2001], at least in the case of non-parallel single ray processing. Here we only concentrate on dynamic changes to objects. Our approach supports dynamic object changes in a very simple way since the major operation in the algorithm is only 2D polygon rasterisation at very low resolutions. At the moment our implementation is single processor, and does not make use of the graphics hardware, nevertheless the results are promising.

2 The VLF-RT Data Structure and Algorithm

Find the smallest bounding cuboid that encloses the scene, and divide the bottom face (parallel to the *xy* plane) into $n \times n$ 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 sets of parallel beams 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. The total set of directions is chosen by a recursive subdivision of a regular tetrahedron. This scheme is often used to find a 'uniform' distribution of points on a hemisphere. It also has the critical property that the closest stored direction to any arbitrary direction can be *looked up* in constant time [Slater, 2002]. To summarize the data structure: we have l directions on a hemisphere. 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.

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 direction and project the ray end-points onto the base of the corresponding cuboid to find which tiles it intersects – typically both end-points are in one tile. 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 using a type of BSP tree. Results using this method for walkthrough can be found in [Mortensen et al, 2004].

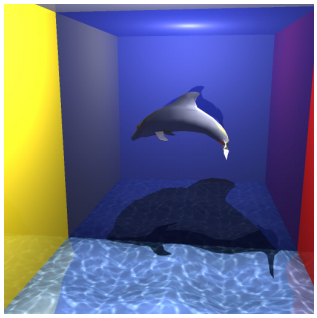

Figure 1. Ray traced scene (dolphin is dynamic- 564polys)

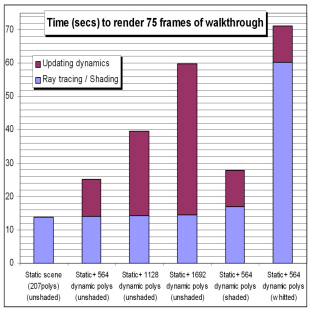

Figure 2. Timing Results on a single 2.8Ghz Xeon processor

3 Dynamic Object Changes

The polygon identifiers corresponding to the static objects in the scene are written into the tiles as described above. Polygon identifiers for moveable objects are also written into the tiles (flagged as dynamic). During rendering, 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. 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 $V\neq GV$ can be reclaimed.

Figure 1 shows a test scene which has a static and dynamic part. The accompanying video shows the dynamic part changing. Figure 2 shows the timing performance for adding an increasing number of dynamic objects. The results show that the overhead of adding dynamics to the visibility field framework is significant as is the cost of shading. Performing tile rasterisation for dynamics update on the GPU would improve performance by at least an order of magnitude thereby allowing larger and more complex dynamic objects to be animated in interactive time.

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

MORTENSEN, J., SLATER, M., KHANNA, P., YU, I. (2004) A Visibility Field for Ray Tracing, submitted. www.cs.ucl.ac.uk/research/vr/Projects/VLF/Media/vlfRTpaper

SLATER, M. (2002) Constant Time Queries on Uniformly Distributed Points on a Hemisphere, *Journal of Graphics Tools*, 7(1):33-44.

WALD, I., SLUSALLEK, P., BENTHIN, C., WAGNER, M.: Interactive Rendering with Coherent Ray Tracing, *Eurographics* 2001 Proceedings, *Computer Graphics Forum*, 20(3), A. Chalmers and T.-M. Rhyne (eds.), 2001, 153-164.