DIRECTIONALITY IN RAY TRACING

Tomáš Kopal, Vlastimil Havran, Jiří Bittner, Jiří Žára
Dept. of Computer Science & Engineering, CTU, Fac. of Electrical Eng.,
Karlovo nám. 13, 121 35 Praha 2, phone: +420-2-24357470, fax: +420-2-298098
e-mail: {kopal, havran, bittner, zara}@sgi.felk.cvut.cz

Keywords: view dependent, directional, data structures, ray tracing, BSP

Ray tracing is one of the most popular and most often used method for computer generated realistic images. But from it's early days, it had one major drawback – it's speed. From then, many has changed, but the speed of ray tracing is still not very encouraging for wide use. There are several ways to make ray tracing more effective. One of the methods for reducing computation time is reduction of ray-object intersection tests performed by means of so called bounding boxes. Another class of algorithms employs 3D space subdivision to implement culling functions.

These algorithms use knowledge of spatial distribution of objects within the scene, but they do not utilize a viewer position and spatial ray distribution, so they are *view independent*. Other methods exploiting this aditional information exist, and they are called *view dependent* or *directional*. Some of the *directional* methods can be regarded as specialized cacheing methods.

The first attempt to use ray coherence was done by Speer et al. [1]. They kept the ray tree generated for the rendering of one image sample and attempted to re-use the information for the next sample, but results of this approach were poor. Hanrahan [2] improved this method by combining it with a beam tracing and obtained better results. The first really directional algorithm was **light buffer** proposed by Haines and Greenberg [3]. Their method accelerates the tracing of shadow rays for point light sources using *direction cube*. Similar approach was chosen by Ohta and Maekawa [4]. Arvo and Kirk [5] came up with another method, ray classification. Collections of rays originating from a common 3D rectangular volume and directed through a 2D solid angle are represented as hypercubes in 5-space. These hypercubes are further subdivided and each hypercube gets associated a list of possible candidates for intersection with ray from within the object set. Rays are classified into unique hypercubes and checked for intersection with the associated candidate object set.

Here we propose a new approach of exploiting view dependency in ray tracing. We use BSP trees as a data structure for culling objects. Current methods build BSP trees by recursively subdividing the bounding volume of the scene by splitting plane and placing resulting volumes in the leaves of the binary tree. Orientation and position of the splitting plane is chosen using cost estimation heuristic (MacDonald and Booth [6]). The tree is built in the preprocessing step and it is not changed during the rendering.

Our proposed method uses very similar approach. We are also using a cost function to choose the best position and orientation of the splitting plane in the BSP tree construction, but the cost function is different. Our estimate of the probability of ray hitting the object isn't based on the area of object surface, but we use additional preprocesing step, in which the sampling is performed. In this step, several pixels of the image are computed (evenly distributed), and the real number of ray-object intersections found is used for estimation. As a special case, we can render the whole image and we can get an exact number of intersections of each object in the scene. This approach can be used e.g. in animation, where the first frame is rendered using arbitrary data structure, and all the intersections are counted. Subsequent frames are rendered using newly built BSP tree.

Our experiments shows that cost function based only on intersection counters associated with each object is not satisfactory. The results of a BSP tree built by this approach are very poor. It is probably due to ignoring the path of the ray and making the estimation based only on the ray target. To overcome this limitation, the BSP tree built using the original cost function is used for the sampling step and intersection counters are associated not only with objects, but also with the bounding boxes of internal nodes of the tree. This approach seems to give better results, but it still needs more measurements.

Our group has developed ray tracer called **GOLEM**, which is a highly modular, object oriented program. This program is used to implement, test and compare new methods. It is based on the OORT (Object Oriented Ray Tracer) by Nicholas Wilt, and its development took 10 months and still continues. We have used it successfully to implement and test the BSP tree construction with the use of the cost estimation function as well as with the new directional approach.

[1] Speer,L,R,Derose,T,D,Barsky,B,A,: A Theoretical and Empirical Anlysis of Coherent Ray Tracing, *Proc. of Graphics Interface '85*, pp. 1-8, May 1985

[2] Hanrahan,P,: Using Caching and Breadth-First Search to Speed Up Ray Tracing, *Proc. Graphics Interface '86*, pp. 56-61, May 1986, extended abstract

[3] Haines,E,A,Greenberg,D,P: The Light Buffer: A Ray Tracer Shadow Testing Accelerator, *IEEE Comp. Graphics and Appl.*, 9(6), pp. 6-16, Sept. 1986

[4] Ohta,M,Maekawa,M: Ray Coherence Theorem and Constant Time Ray Tracing Algorithm, *Computer Graphics 1987 (Proc. CGI '87)*, pp. 303-314, Tokyo, 1987

[5] Arvo,J,Kirk,D: Fast Ray Tracing by Ray Classification, *Computer Graphics (Proc. SIGGRAPH '87)*, 4(21), pp.55-64, July 1987

[6] MacDonald,J,D,Booth,K,S: Heuristics for Ray Tracing Using Space Subdivision, *The Visual Computer*, 6(3), pp. 153-166, June 1990