Accelerated 5D Ray Tree construction on the GPU

Ravi P. Kammaje[†1,2] and Benjamin Mora[2]

Abstract

Ray tracing random rays has been a challenge. Due to their reduced coherence, the normal methods of acceleration like packet tracing that make use of coherence of the rays do not work well. These random rays are encountered in global illumination methods. 5D ray classification, first introduced by Arvo and Kirk [AK87], can classify these rays into coherent groups. We introduce a method that builds a hierarchical structure identifying coherence in random rays very quickly using the increased processing power of the GPU.

Categories and Subject Descriptors (according to ACM CCS): Comp. Graph. [I.3.6]: Methodology & Techniques—

1. Introduction

Coherence is used frequently to accelerate Ray Tracing. Several recent methods involve utilizing coherence, both in the object itself and in the rays to make ray tracing faster. Unfortunately, coherence occurs only in primary rays and maybe partially in shadow, reflection and refraction rays. For even basic global illumination and for methods like ambient occlusion, real incoherent rays are necessary to be traced through the model and recent methods like packet tracing [WS01] and MLRTA [RSH05] are not as effective. Thus, the aim of our work is – to quickly identify a hierarchical structure of coherent rays amongst random rays in order to make the acceleration methods more effective.

2. Ray Tree

5D classification – whereby the rays are classified into groups of most coherent rays – was introduced by Arvo and Kirk [AK87]. It is believed that using this method over random rays can produce enough coherence to accelerate their ray tracing. Additionally, this method affords itself to acceleration on the GPU. Recently some researchers have addressed the problem of incoherent rays using similar methods. Roger et al. [RAH07] use 5D classification and show impressive results for shadow and reflection rays. Garanzha and Loop [GL10] use a sorting method to create a flat structure of rays, but address only secondary rays and not the truly incoherent rays.

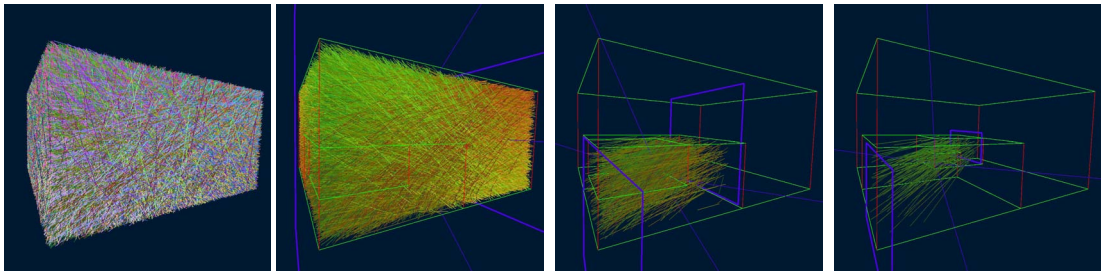
Each ray is considered as a 5D point – the origin contributing (x,y,z) coordinates and the direction contributing two further coordinates – (u,v). Initially, the rays are classified into six root nodes based on their direction's dominant axes – $-X$, $+X$, $-Y$, $+Y$, $-Z$ and $+Z$. The dominant axis is the maximum component in the normalized direction vector of the ray. The ray direction vector can be considered as a line starting at the origin hitting one of the faces of a unit cube centered at the origin. The 2D coordinates on the cube's face where the unit direction vector intersects gives the (u,v) coordinates of the ray. Using these 5 coordinates each ray is classified into one of 32 sub groups or nodes of the tree. This classification is repeated until each group contains less than a small number of rays (determined by the user) or when a node is at a pre-determined maximum depth. The tree thus built is analogous to an octree, but in 5D instead of 3D and exposes the coherence in random rays – i.e. each node contains coherent rays.

3. GPU Ray Tree building

The modern GPU with its multitude of cores can process several operations in parallel. However, GPUs do not handle recursion very well. Thus, tree building, which is essentially a recusive process has to be performed using a breadth first method. In addition, for efficiency reasons, the tree will be represented as an array of nodes. The ray tree building using a breadth first algorithm would be as follows.

- Identify the root nodes of the ray tree
  - Find dominant axis of rays, i.e. longest component of normalized ray direction. The dominant axis indicated by $0,1,2,..5$ for $-X$, $+X$, $-Y..+Z$ is the node to which the ray belongs to
  - Sort the rays to ensure that rays of a node are together in the array
  - Count the number of rays in each node
  - The range of the five dimensions is now - $X_{min}-X_{max}$, $Y_{min}-Y_{max}$, $Z_{min}-Z_{max}$, $U(-1,1)$, $V(-1,1)$
- Build the lower levels of the ray tree

[†] Chairman Eurographics Publications Board



(a) All the random rays (b) Top Level of tree (c) Node at Level 1 of tree (d) Node at Level 2 of tree

Figure 1: *The different levels of the ray tree.*

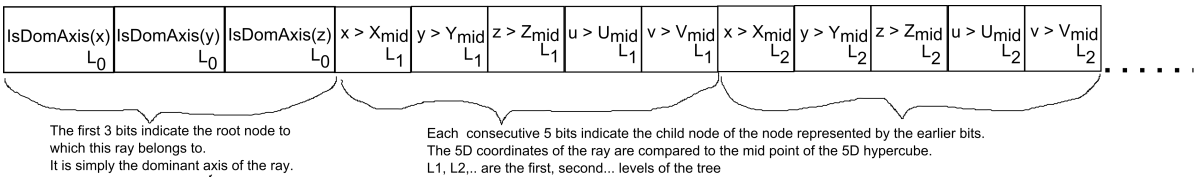


Figure 2: *Determining the bits for presorting the rays*

- Find mid point of each active node (five dimensional hypercube) – (X_{mid}, Y_{mid}, Z_{mid}, U_{mid}, V_{mid})
- Classify each ray based on its 5D representation, using 5 bits to represent this classification.
- Sort the rays to ensure that rays of a node are together in the array
- Count the number of rays in each node
- If any node contains fewer than $leafNodeRays$, make it a leaf node. i.e. do not divide it further
- Continue until all nodes are leaf nodes

It can be observed from the above steps that there is an expensive sort operation involved at each iteration of the tree building process. The rays can be pre-classified to as many levels as needed before the tree building process begins. 3 bits are necesary to indicate the ray's root node and 5 bits to indicate a ray's classification. A 32 bit integer, can thus indicate 5 levels of the tree nodes as shown in Figure 2. Similarly another 32 bit integer can indicate a further 6 lower levels. These integers are stored in a two arrays – $L1$ and $L2$. These arrays and rays are sorted to get these rays in their node order. Just two fast radix sorts [SHG09] are necessary to build the entire tree.

4. Results

To benchmark the results, we use a PC with a Core2 Quad 2.4GHz processor with 4GB of RAM and an Nvidia Quadro FX5800 with 4GB of video memory. CUDA and C++ are used to implement the tree building on the GPU and CPU respectively. It is to be noted here that the GPU algorithm is still a work in progress and needs to be further optimized. 1024×1024 random rays are generated and a tree is built out of these rays using both the CPU and the GPU.

To build a ray tree – with a maximum of 64 rays in the leaf node – using the CPU takes around 450 ms whereas to build the same tree using the GPU takes around 21 ms. Figure 1 shows the coherence produced.

5. Conclusion

As can be inferred from the results, the GPU can be used to effectively identify coherence in random rays. An effective algorithm that utilizes this coherence and traces these rays to produce realistic global illumination effects at interactive frame rates is being pursued.

References

[AK87] ARVO J., KIRK D.: Fast ray tracing by ray classification. *SIGGRAPH Comput. Graph. 21* (August 1987), 55–64. 1

[GL10] GARANZHA K., LOOP C.: Fast ray sorting and breadth-first packet traversal for gpu ray tracing. *Computer Graphics Forum 29*, 2 (2010), 289–298. 1

[RAH07] ROGER D., ASSARSSON U., HOLZSCHUCH N.: Whitted ray-tracing for dynamic scenes using a ray-space hierarchy on the gpu, jun 2007. 1

[RSH05] RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-Level Ray Tracing Algorithm. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005) 24*, 3 (2005), 1176–1185. 1

[SHG09] SATISH N., HARRIS M., GARLAND M.: Designing efficient sorting algorithms for manycore gpus. *Parallel and Distributed Processing Symposium, International 0* (2009), 1–10. 2

[WS01] WALD I., SLUSALLEK P.: State of the Art in Interactive Ray Tracing. In *State of the Art Reports, EUROGRAPHICS 2001*. EUROGRAPHICS, Manchester, United Kingdom, 2001, pp. 21–42. 1