Tyler Paulsen
CSCI-711-01 Global Illumination
Weekly report for Week #3

Summary:
The real time KD-Tree construction on graphics hardwares optimises the advantages of a GPU. The implementation was in CUDA, and was 4 ~ 7 times faster than a single core algorithm, and it was comparable to the multicore algorithms. The k-d tree is used in ray tracers by walking through the tree until it reaches a leaf node. A k-d tree takes a scene and splits it up based on what the algorithm deems as an important plane, and then keeps subdividing until a threshold is met.

Speed
The multi core algorithms, although comparable in speed, did not match the accuracy of the GPU version. The GPU is so powerful at this because of its high number of cores. Each core can work its way up or down the tree to find the solution.

Algo
The algorithm uses SAH to optimise the accuracy of the tree, but this in turn slows down the construction of the tree. SAH - surface area heuristic - is a prediction cost-estimate function that helps divide up the k-d tree. The algorithm uses triangles as inputs, and builds the tree in BFS order. The nodes are split up into two categories: large and small. The large node go through a series of steps to eventually be broken up into smaller nodes. Small node use two stages: preprocess and process. The process stage is where it splits the nodes, and stores the triangles.

The algorithm works well with high triangle densities and is scalable with the number of GPU cores, but it starts to be slower with a triangle count less than 5,000. If a small amount of triangles is present, it is better to go back to the CPU only version. A high memory usage and is also associated with this algorithm when compared to its CPU predecessor.

Octrees
(A) What are they?
Structure that for each internal node has eight children and is normally used to subdivide 3d space.
(B) How can they help us with ray tracing?
Speeds up the ray intersection process. Allows the objects of a scene to be partitioned in this structure, when a ray is fired, the structure is then traversed to find an intersection.
(C) How do they differ from the other data structure that you chose?
Octrees subdivide the scene into nodes that represent a cube of space within the scene while BVH partition off parts of the screen with different objects and creates a tree out of them. The tree does not have to be equal across all nodes.

references:
http://wscg.zcu.cz/wscg2000/Papers_2000/X31.pdf
https://en.wikipedia.org/wiki/Octree

Tyler Paulsen
CSCI-711-01 Global Illumination
Weekly report for Week #3

Bounding Volume Hierarchy
(A) What are they?
Tree like structure that contains a set of objects. The objects in the scene can piece together one large object, or they can be separate parts of the scene
(B) How can they help us with ray tracing?
They are used to represent the objects in the scene, so the purpose is much the same as the Octree, but it seems like it would be slower. The octree could partition the scene as many depths as it needed while the BVH only breaks up the scenes objects and does not narrow the location of the ray intersection too well.
(C) How do they differ from the other data structure that you chose?
The BVH implementation that i looked at did not break the scene down nearly as much as the octree did. The BVH can create a single object from multiple pieces by piecing them together.

references:
https://en.wikipedia.org/wiki/Bounding_volume_hierarchy
http://www-ljk.imag.fr/Publications/Basilic/com.lmc.publi.PUBLI_Inproceedings@117681e94b6_1860ffd/bounding_volume_hierarchies.pdf