Lecture 16: Data Structures – Review Questions

- How do we do intersection testing between a ray and a bounding box? Note that we do not need to find the point of intersection, but we only need to determine whether or not the ray intersects the bounding box.
- How do we do intersection testing between a ray and a bounding sphere? Again, we only need to know whether the ray intersects the sphere. We do not need to find the specific intersection point.
- Describe a technique for constructing a bounding box hierarchy. What data structure will you use to store this hierarchy?
- Explain in detail how to use this bounding box hierarchy to identify the intersection between a ray and the closest object in the environment. Your algorithm should, of course, be more efficient (in the general case) than the brute force process of checking the ray for intersection with all objects.
- How would you update the bounding box hierarchy if there are moving objects in the environment? Is there a way to do this efficiently, i.e., without rebuilding the entire tree?
- One alternative to a bounding box hierarchy is to use a BSP tree with axis-aligned splitting planes. Describe how to construct such a tree. Describe how to perform ray-object intersection using this data structure. Your ray-object intersection algorithm should check regions from front to back order, so that the search for an intersection may be halted when an intersection point is found.
- In the BSP scenario, what problem is introduced when the splitting planes intersect some of the objects? Assume that you do not want to use the plane to split the objects (e.g., splitting a sphere into two parts may not be desirable for ray tracing). How can you solve or work around this problem?
- Give a data structure for storing vertices, edges, and faces that makes it easy to perform queries such as ‘find all faces adjacent to a vertex’ and ‘find all edges attached to a face.’ The winged-edge data structure is one common choice.