CSci 5608: Programming Assignment #2*

due: 9pm, Monday, March 2

The purpose of this assignment is to extend your understanding of bounding volume hierarchies (BVH). Specifically, you are to modify **pbrt** and incorporate "split clipping" into its BVH implementation. Split clipping is a technique that subdivides the triangles in a scene to reduce the amount of overlap in the bounding volumes that are part of the hierarchy. In this way it is possible to gain some of the advantages of the Kd-tree space subdivision approach to building a hierarchy without paying the storage space penalty that usually accompanies the Kd-tree technique.

You should begin this assignment by reading two references that describe the split clipping algorithm. "Early Split Clipping for Bounding Volume Hierarchies" by Ernst and Greiner was the paper that first introduced the method and showed how it could be competitive with the Kd-tree approach. This article contains a very nice overview of both the BVH and Kd-tree methods. It also has excellent illustrations that show how the split clipping procedure works. Once you have digested the Ernst and Greiner paper take a look at "The Edge Volume Heuristic - Robust Triangle Subdivision for Improved BVH Performance" by Dammertz and Keller. This paper proposes a better way to subdivide triangles during the preprocessing step that is part of the early split clipping algorithm.

Your task in this assignment is to incorporate split clipping into **pbrt**'s BVH implementation. Your approach should follow the basic outline provided by Algorithm 4 in the article by Ernst and Greiner, and your method should subdivide the triangles in the scene before the BVH construction takes place. However, in implementing this algorithm you should modify it to incorporate the triangle subdivision approach described in the Dammertz and Keller paper. Here are some hints on how to apply this approach to **pbrt**, but feel free to edit other files and functions:

- You are only required to support triangle mesh split clipping.
- Relevant code files you will likely need to modify:
  - → `src/core/api.cpp`
  - → `src/core/shape.h`,
  - → `src/shapes/triangle.h`
  - → `src/shapes/triangle.cpp`
  - → `src/main/pbrt.cpp`
- Modify the `Shape` interface to add a function that reports whether or not split clipping is supported for the type, and a function that creates a new shape instance to store the result of the split clipping.
  - → Because `Shape` is an interface, make sure the functions are virtual.
  - → The default implementation should be that split clipping is not supported so you only need to override the functions for triangle meshes.

* This assignment is based on Problem 4.4 in Pharr, Jakob, and Humphreys.

- You will want to modify the `MakeShapes` function in `api.cpp` to use these new split clipping functions of the `Shape` interface. Make sure to delete the original shape if necessary (e.g. a new split-clipped shape was made and the original is unnecessary).
- The `TriangleMesh` class uses an index array to store triangles. Every 3 elements in the `vertexIndices` member form a triangle. The 3 indices can then be used to look up the position, normal, *uv*, and other vertex attributes in the other members of the instance.
- The constructor of `TriangleMesh` expects points to be specified in object space. However, it pre-transforms those points to world space. This means that the positions you operate on during split clipping are in world space (the correct space to be evaluating the split criteria), but you will need to transform them back to object space when you create the newly optimized triangle mesh.
- If normals are provided in the mesh, they are always specified in object space.
- Since the `TriangleMesh` API needs everything allocated up front, a two pass approach could be more efficient. One to calculate the number of splits (i.e. new triangles and vertices needed), and a second to calculate the split positions.
    - → Keep in mind that since this approach only splits triangles, all of the original vertices are still going to be referenced by the new triangle. This can be taken into account when you build the new vertex arrays.
- If the `TriangleMesh` has normals, *uv*'s, or any other attributes you should properly split those as well by averaging their values for the edge being split.

Once you have completed your implementation of the split clipping algorithm you should test it using the approaches described in the Dammertz and Keller paper. To accomplish these tests you will be provided with datasets for the Stanford Bunny, a Spaceship, and the Sponza Atrium. Please use these datasets to perform the following experiments:

1) Demonstrate that objects, such as the Stanford Bunny, that have good ray tracing performance are not subdivided much when the Edge Volume Threshold described in Section 2.2 of Dammertz and Keller assumes its default value.
2) Compare the performance of your algorithm for the Spaceship and unrotated Sponza Atrium datasets. Show how changing the Edge Volume Threshold affects the rendering time and number of subdivided triangles for each of these two scenes. To accomplish this you will need to recreate the graphs shown in Figure 4 of the Dammertz and Keller paper.
3) Show what happens when your algorithm is applied to rotated versions of the Sponza Atrium dataset. For each of the rotated datasets, compare what happens to the frame time and the number of triangle references for different Edge Volume Threshold settings. This test involves recreating information contained in Figure 5 of Dammertz and Keller. (It will be hard to get the exact rotated Sponza scenes found in the paper. Try to get as close as you can, but it's OK if it isn't perfect.)

Following the procedure outlined in Program 1 (submit a document containing a link to a zip file in a Google drive folder), turn in your code and the results of the tests that you performed. Please include a brief one or two page writeup that summarizes what you discovered during your testing. The points for this assignment will be equally divided between your implementation (was it done correctly and does it work at all) and the results of your tests (does it work as expected).