CS350
Assignment 4: BSP-Trees with CSG operations

Submission:

Zip up and submit the entire project folder (the .sln and the CS350Framework folder). Make sure to not include any build artifacts! See the syllabus for submission specifications. Due April 8th (Sunday) by 11:55 pm

Topics:

The purpose of this assignment is to construct a binary space partitioning tree for optimized queries and geometry manipulation. Basic tree construction will be performed using heuristics described in class. The tree will be used for optimized ray-cast queries and for CSG operations. CSG (Constructive Solid Geometry) operations allow complex shapes to be built up from Boolean operations on simpler primitives. In particular, Union, Intersection, and Subtraction of two shapes will be performed using the BSP-tree.

Testing:

You will be given a txt file containing sample results for the assignment. To test your project for this assignment run with the command line arguments of *"4 0"*. The argument "0" runs all of the tests at once.
Do note that the graded tests will not be limited to the ones given to you!

Grade Breakdown:

	Points	Percentage
BspTree.cpp		
SplitTriangle	15	12%
CalculateScore	10	8%
PickSplitPlane	5	4%
Tree construction	7	5%
Ray Casting	30	23%
Debug Drawing	5	4%
Invert	10	8%
ClipTo	30	23%
Union	7	5%
Intersection	5	4%
Subtraction	5	4%
Total	129	100%

Implementation Details:

SplitTriangle:
Triangle splitting must be performed with the robust method described in class (the table with 9 states). That is, you must properly handle going between all permutations of Inside, Outside, and Coplanar points. Note that I do not want you to try and handle the edge clipping order (A to B vs. B to A) for this assignment. You should only attempt to use the table if the triangle is overlapping the split plane, if it is classified as either being in front or behind the plane then just insert it into the corresponding array. If the triangle is coplanar then use the normal of the triangle to determine if it belongs to coplanar back or front (front if the normal points the same direction as the split plane). Also, when the result from a split is a quad, turn the quad into two triangles of the indices [0, 1, 2] and [0, 2, 3].

CalculateScore:
Use the method for calculating score as described in class:
$score = K * N_s + (1 - K) * Abs(N_f - N_b)$, where N_s is the number of triangles straddling the plane, N_f is the number in front, and N_b is the number behind. Build the split plane from the triangle at the testIndex. Note: Do not include coplanar triangles in the score! Also, be careful of degenerate triangles causing degenerate plane normals. You should return Math::PositiveMax() if the triangle is degenerate. Finally, in the case of a tie in score choose the first triangle of that score (from 0 to size).

PickSplitPlane:
Simply choose the triangle that produces the lowest score.

Tree Construction:
Recursively build the tree by splitting the data set with the best scoring triangle's plane. Recursion should stop when a there is only 1 triangle remaining. Also note that this is a Node-Storing tree, that is all coplanar triangles should be stored in the node.
Important! Make sure to not pick a split-plane that is almost the zero vector. If you fail to do this you can end up with every triangle being coplanar. For this assignment, if the normal's length is below the construction epsilon then you should not choose that plane.

GetRoot:
This function should return the root node of your tree. Additionally you should fill out the provided helper functions on the node class. This is used for me to walk and print the structure of your tree.

Ray-Casting:
Optimized ray-casting should be implemented as described in class (using tMin and tMax). Do not implement the "basic traversal" but rather the 4 main cases with the 3 edge cases. Make sure that you only check the geometry in the plane when the ray hits the thick plane otherwise you will end up with more PlaneTriangle tests. Use planeThicknessEpsilon to classify the ray's start. Use triExpansionEpsilon for RayTriangle, if you don't use this then some tests will slip between two triangles. Be careful as there are a lot of edge cases with raycasting (all should have a unit test).

Debug Drawing:
Draw the nodes in the tree at the given level (level of -1 means draw everything). You should use the color and bitMask variable passed in to set the corresponding values on the DebugShape. For color just call DebugShape.Color(color) on the shape returned from a draw function. For the bitMask just call DebugShape.SetMaskBit(bitMask). The bitMask is meant to help you by toggling certain debug shapes at runtime. Hitting any key of 0-9 will toggle drawing of the any shape with those masked bits set (used in CSG operations).

Union, Intersection, Subtraction:
The CSG operations of union, Intersection, and subtraction should be performed using the 3 functions described in class: AllTriangles, Invert, and ClipTo.
AllTriangles:
Collect all triangles of in the tree into one list in a pre-order traversal (node then front then back).
Invert:
When inverting the split plane make sure to properly invert the entire plane (not just the normal).
ClipTo:
The given epsilon should be used when clipping triangles. For ClipTo make sure that you cull any triangles that end up in a solid leaf node, that is any triangles that end up on the back side of a leaf node’s split plane. Coplanar front triangles should be included with front triangles and coplanar back with back.

Performance Penalties:
To run all tests for this assignment should take less than 5 seconds in debug mode. If you take longer there then you are likely doing something really bad (I did no major optimizations). The most likely candidate is allocations in your PlaneTriangle function. You must fix any major performance issues or you will be penalized.

Code Quality:
Code quality is a percentage modifier applied to your grade up to a -20%. Common code quality penalties are: redundant code, re-computed values, unnecessary allocations, and very hard to read code.

Extra Notes:
Avoid creating new files in the framework. I will be copying the relevant files for the assignment into a clean project for testing. For this assignment that is the files:

1. Geometry.hpp/cpp
2. Shapes.hpp/cpp
3. SimpleNSquared.hpp/cpp
4. DynamcAabbTree.hpp/cpp
5. BspTree.hpp/cpp

The full framework should still be submitted though.
To make it easier to find missing functions they contain the comment “/******Student:Assignment4******/” that you can search.