CS350
Assignment 5: GJK

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 22th (Sunday) by 11:55 pm.

Topics:

The purpose of this assignment is to implement the GJK algorithm to detect collision between arbitrary convex shapes as well as to find the closest features between them when they are separating. Additionally, some optimized support functions should be implemented.

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 *"5 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
Gjk.cpp		
SupportShape:	*5*	*6%*
Support	2	2%
GetCenter	2	2%
DebugDraw	1	1%
SphereSupportShape	*6*	*7%*
Support	6	7%
ObbSupportShape	*6*	*7%*
Support	6	7%
Gjk	*65*	*79%*
Point Region Tests	2	2%
Edge Region Tests	6	7%
Triangle Region Tests	12	15%
Tetrahedron Region Tests	20	24%
Intersect	25	30%
Total	82	100%

Implementation Details:
class SupportShape:
You must implement the non-virtual functions of the SupportShape class, namely: GetCenter, Support, and DebugDraw. For GetCenter just compute the centroid of the point set. The DebugDraw function should draw the point set with the given transform applied. The Support function should just find the point furthest in the given direction with the applied transformation. When there is a tie you should pick the first point, that is only choose a new point if the distance is strictly greater than the old furthest distance. **You should not transform every point into world space to test as this is less efficient!** You should instead transform the search direction back into local space, do the search there, and then transform the resultant point into world space. Doing the correct local-space test will account for some points for this test!

SphereSupportShape::Support:
Implement the optimized support function for a sphere. That is, find the point on the sphere furthest in the direction passed in.

ObbSupportShape::Suport:
Implement the optimized support function for an Obb. Do not just build up all 8 points and test them. Instead use the optimized function covered in class (in SAT slides). This basically amounts to transforming the search direction to local space, computing the support of an Aabb, then transforming back to world space.

GJK:
Fully implement the GJK algorithm to both determine if two convex shapes collide (the Boolean result) and to find the closest pair of points if they are separating (the closest features). To compute the closest points you should project the origin onto the CSO's simplex and then use barycentric coordinates to compute each shape's respective points. The helper function ComputeSupport has been stubbed out for you to compute a CsoPoint (as well as each shape's point) from a given search direction. It is highly recommended to implement this function.

For each simplex size you need to implement the IdentifyVoronoiRegion helper function. IdentifyVoronoiRegion takes the input points (labeled p_i) and the query point (q) and return which voronoi region q is in. Additionally, this function needs to fill out several parameters depending on which voronoi region q is in. First, the points of the new simplex (it may reduce) need to be filled out. This is handled by filling out the new size and the indices of the input points (where 0 corresponds to p_0). Additionally, the closest point on the new simplex to q as well as the final new search direction need to be filled out.

Note: Do not brute force these functions (testing all features), you must use an efficient method such as barycentric coordinates.

The meat of the GJK algorithm should be implemented in the Intersect function. This function should determine if the two support shapes overlap. As GJK is an iterative algorithm, you should try up to "maxIterations" before giving up (although no test should run out of iterations). If the two shapes do not intersect, then you should fill out closestPoint with the closest features of the two shapes.

Two values are passed in for debugging: debugDraw and debuggingIndex. The bool is simply whether or not you should debug draw. The index passed in is meant as an aid to help you debug sub-steps. When viewing a level in the UI you can set the debuggingIndex. I recommend using this to allow stepping through the algorithm to view how your simplex is updated over time.

When it comes to the termination conditions. Check exactly for the zero vector when determining if the projected point is the origin. Also when checking if no more progress can be made, check the distance of the vector from the projection point to the new search point and see if that distance (along the search direction) is less than or equal to the passed in epsilon.

While it shouldn't matter when it comes to the final result of GJK, there are some varying ways you can start GJK. If you run into any output differences consider the following ways that I setup the algorithm. When computing the initial search direction I used the difference of shape A's center and shape B's center. If this produced the zero vector I used Vector3(-1, 0, 0) as the starting search direction.

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
6. Gjk.hpp/cpp

The full framework should still be submitted though.

To make it easier to find missing functions they contain the comment "/******Student:Assignment5******/" that you can search.