CS 350 | Spring 2021

Assignment 4| GJK Algorithm

Files (submit folder) due

- Week 14th
- By midnight

Remember: The major point of this assignment is for you to learn how to submit your homework correctly. You should strive to follow all of the directions exactly as specified in the handouts. The syllabus that was handed out during the first day of class also contains important information on how to submit your homework. If you are still unsure of how to do something, you should ask for help, either from myself or from a tutor in the lab or from another student.

Topics

This assignment focuses on generating implementing the GJK algorithm for testing for the intersection between any two convex objects.

Scene Setup

- Use the scene setup from previous assignments
- Throw a sphere into the scene when the user presses the SPACE_BAR. The sphere originates from the camera position and moves in the direction of the lookAt vector for the camera. We want to calculate the intersection of the sphere with the meshes in the scene.
- You will demonstrate the GJK algorithm as outlined in class on the spatial partitioning implemented in the previous assignments. The hierarchies used in this assignment are:
    - AABB-BVH, and
    - Adaptive Octree
- The collision at terminal nodes will be detected via the GJK algorithm.
- User has option to select debug drawing of GJK.
    - If debug draw is selected, then go through each iteration of GJK based on user input (SPACE_BAR).
- Debug draw functionality: Once a collision is detected, do the following:
    - STOP all animation.
    - Display the polygons for the intersecting geometry and sphere in RED color
    - If Debug_Draw selected by user
        - Display the **initial simplex** for GJK iterations as a wireframe/transparent polygon in red. You can choose to display this in a separate window / viewport on the screen.
        - User presses SPACE_BAR
            - Perform the GJK refinement step(s)
            - Display the intermediate/final simplices as wireframe/transparent polygon in red color.
            - User presses SPACE_BAR → Move to next iteration.
        - User presses SPACE_BAR. Now remove the sphere from the scene and render the meshes normally

Implementation

The collision detection algorithm will be covered in – broad-, mid-, and narrow phase.

We can probably implement everything in one recursive function, **but separating the broad-, mid-, and narrow-phase calculations allows us to customize them individually in the future.**

For the narrow phase calculations, there are multiple approaches to calculate the "convex" bounding volumes required for GJK -

a. Pre-calculate the convex hull of terminal geometry when building the hierarchy. Use the convex hull as one of the convex inputs to GJK (most accurate)
b. Use AABB or OBB, whichever is the tightest, as the convex approximation to the geometry. You will get inexact results, but it will save you from calculating the convex hull. This approach is more useful if you only need a confirmation of intersection but not further information (like contact points, separation distance etc.)
c. Use a lower poly approximation of the actual geometry which is convex in shape. This is useful for humanoid meshes or dynamic meshes.

For this assignment, use approach b (using AABB for GJK algorithm). The goal is to learn how to implement the GJK algorithm (first), before implementing the most optimal algorithm on the market.

```
bool DetectCollision_BroadPhase( Sphere S, Node *RootNode )
{
      // The broad phase part of collision detection
      BBox worldBox = RootNode->getBBox();
      BBox sphereBox = S->getBBox();

      // standard box-box intersection
      if( !BBOX_intersection( worldBox, sphereBox ) )
            return false;

      // boxes collide, move to mid-phase
      // In mid-phase, we check for intersection of S with internal
nodes
      for each childNode of the RootNode
      {
            // For BVH and Octree: box-box collision
            if( DetectCollision_MidPhase( S, childNode ) )
                  return true;
      }

      // no intersection with the scene
      return false;
}
// More pseudo-code on Page 4 of the handout
```

Note

- You should have the ability to explain and derive every single line of code used in your project

Grade Breakdown

- GJK algorithm: 80 points
- Rendering the simplices (initial & iteratively refined): 20 points

```
bool DetectCollision_MidPhase( Sphere S, Node *treeNode )
{
      // if treeNode == LEAF, then perform GJK
      if( treeNode->type == LEAF_NODE )
      {
            // Potential approaches for GJK with S
            // a. (pre-)Calculate the convex hull of treeNode geometry –
most precise calculation
            // b. use BBox of treeNode geometry – approximate solution
            //  c.  Level-of-detail  based  pseudo-geometry  for  dynamic
objects
            // We will use approach b. for our assignment
            if(DetectCollision_GJK(S->getBBox(), treenode )
            {

                  // Render polygons of treeNode & sphere in RED color
            }
      }

      BBox nodeBox = treeNode->getBBox();
      BBox sphereBox = S->getBBox();

      // standard box-box intersection
      if( !BBOX_intersection( nodeBox, sphereBox ) )
            return false;

      // Now recurse through the child nodes
      for each childNode of treeNode
      {
            // For BVH and Octree: box-box collision
            if( DetectCollision_MidPhase( S, childNode ) )
                  return true;
      }

      // no intersection with the scene
      return false;
}
```

Please refer to your course syllabus for submission guidelines.