Trapezoidal Algorithm Documentation

By Karl Boghossian

N.B: in the *Trapezoidal_Algorithm.h* there is a defined variable called **#define REALTIME_BUILDING 1** that will demonstrate almost all the trapezoidal map phases in real time, so to remove this feature just comment the defined variable (//#define REALTIME_BUILDING 1) so it will build all the map directly.

Trapezoidal_Algorithm.h & Trapezoidal_Algorithm.cpp

The variables in the trapezoidal_algorithm class are:

- An integer that saves the number of segments lying in space, this variable is called `int numberOfSegments;`
- A pointer to the array of segments from which the algorithm will get segment by segment to check the cases, the variable is `Collision_Data *Segments;`
- A list of Trapezoid * to hold the created trapezoids, it's named `list<Trapezoid *> trapezoidalList;`
- A list of Trapezoid * to hold the create solid trapezoids, it's named `list<Trapezoid *> solidtrapezoidalList;`
- A variable to hold the number of trapezoids that a segment intersects, named `int tcount;`
- A vector (array) of trapezoid * to hold the intersected trapezoids by a segment, named `vector<Trapezoid *> trapezoidIntersections;`
- A vector of Vertices, to store all the used points from each segment, to know if the current vertex of the current segment has been used previously or not, name: `vector<Vertex> UsedPoints;`
- A pointer to the search tree class, the tree will serve as a tracker for the creation of the trapezoidal list, name: `SearchTree *searchTree;`
- A pointer to the graph that will be generated after building the trapezoidalList, name: `Graph *graph;`

In the trapezoidal_algorithm constructor I am:

- Initializing the Graph Pointer
- Adjusting the segments by calling `Swap_Segments_StartEndPoints();`
- Create the trapezoidal map by calling `Create_TrapezoidalMap();`
- After building the trapezoidalList, I adjust each trapezoid's segments by calling `Adjust_Trapezoids_Segments();`
- So up until now, I adjust all the trapezoids' segments including the trapezoids lying in the solid area.
- I remove the unused trapezoids, the trapezoids lying in solid areas by calling `Remove_UnusedTrapezoids();`
- Here I have created 2 lists, a list holding the trapezoids lying in empty space and a list holding the trapezoids lying in solid areas.
- I create for each trapezoid a model to represent it, the model is made of 4 lines that describe the top segment, bottom segment, left segment along with the right segment. This is done by calling `Create_MapAsCharacter();`
- After I have each trapezoid made of segments each having a start and end point, I generate the graph by calling `PathBuilding();`

I will describe what each function does:

- `Swap_Segments_StartEndPoints` this function will loop through all the segments in the array of segments to check if the current segment vertices need to be swapped; the checking is done according to the vertices x values, the rule is that the starting vertex's x-value must be less than the ending vertex's x-value.
- `Adjust_Trapezoids_Segments` this function will loop through all the trapezoids in the trapezoidalList to adjust each trapezoid's segment by adjusting it's top & bottom start & end points by getting the point of intersection between 2 lines; the 2 lines are the top segment and the left vertical segment; same with bottom.
- `Remove_UnusedTrapezoids` this function will loop through all the trapezoids in the trapezoidalList to check which one lies in a solid area; this is done by checking the current trapezoid's top segment's normal's y-value with the current trapezoid's bottom segment's normal's y-value (if the top has a positive normal & the bottom has a negative normal → solid trapezoid), so it removes it from the trapezoidalList and adds it to the solidtrapezoidalList.
- `GetBoundingRectangle` this function loops through the segments in the Segments array to get the min & max x-values along with the min & max y-values in order to create the first trapezoid that encloses all the segments in the world. The bounding trapezoid has its top segment's normal downward and its bottom segment's normal upward (not solid).
- `Get_IntersectedTrapezoids` this function takes a segment to check all the intersected trapezoid with this segment. This is done by calling `searchTree->TrapezoidSearch(LS_V0,LS_V1)` that in turn will traverse the tree to check which trapezoid is intersected with the segment. After getting the first trapezoid intersected by the segment, we get all the remaining trapezoids intersected with the segment by checking the current trapezoid's right point x-value with the line segment endpoint x-value (from left trapezoid to the right one).
- `Check_Segment_StartEnd_Used` a function that takes 2 references to Booleans and a segment. It will check if the current segment's vertices have been used previously; if not, it adds the segment's points to the array of used points. These Booleans are used later for the trapezoidalList building process.
- `Create_MapAsCharacter` a function that loops through the created trapezoidalList and solidtrapezoidalList to create their model (to be used for drawing).
- `Draw` this function draws each trapezoid from the trapezoidalList along with each trapezoid in the solidtrapezoidalList, plus it calls the `graph->Draw` to draw the generated graph.
- `Create_TrapezoidalMap` this function creates the trapezoidalList with the tree. First it gets the boudingtrapezoid, then creates the SearchTree and initialize its root treeNode to the boundingtrapezoid. 2 booleans are used to know if the current segment start or end point has been used. Then it loop through all the segments, gets the intersected trapezoids with the current segment, then enters all the cases.
- `GetLeastDistantTrapezoid` this function gets the most left trapezoid from the trapezoidalList; this is done by checking the trapezoid having the least left point's x-value.
- `StatesBuild` this function is a recursive function that takes the current trapezoid to be check with the prev state index of the previous node in the graph (to be used to link the current trapezoid with the prev common edge). If computes the center of the current trapezoid, then checks if the trapezoid has a negative state index (default value is -1), if yes so it creates a node having the trapezoid's center as position, then sets an index to the current trapezoid; if the prev state index is not -1 (default value) so we link the prev node having the prev state index with the create trapezoid; then we loop

through the neighbors of the current trapezoid and check if the current neighbour has not been used yet then create a node having a position as the center of the common edge (the common edge is the segment having a smaller length between the current trapezoid segment with its neighbour; next is creates an edge between the current trapezoid and the common edge.
- `PathBuilding` is a function that first gets the least distant trapezoid, then recursively builds the graph then build the shortest path between nodes by calling `graph->Init_ShortestPath();`
- `Get_TrapezoidState` a function that loops through the trapezoids in the trapezoidalList and classifies each trapezoid with the user's choice position; the first trapezoid holding the vertex returns its stateIndex to be used as source or destination accordingly.

Trapezoid.h

The variables in the trapezoid class are used to define the trapezoid dimension, like top segment, bottom segment, left and right points.
It has an array of 4 pointers to trapezoid to specify its neighbors
It has a state index used for the graph
- `CreateTrapezoidModel` is a function to create the model of the current trapezoid to be drawn later.
- `Compute_Center` is a function to compute the center of a trapezoid according to its segments.
- `Classify_PtToTrapezoid` is a function to classify a point to a trapezoid (used to know if the user's desired source lies in a trapezoid).

SearchTree.h

Contains the searchTree and the searchTreeNode.

The searchTree class is used to build the tree while generating the trapezoids, it is used to track the trapezoids and to search through them in an efficient way.
It contains the root of the tree.
- `TrapezoidSearch` is a function that searches for a trapezoid inside the tree according to a segment, that's why it takes a line segment as parameter.
- `TreeNodeSearch` is a function that searches for a TreeNode containing a Leaf to be returned, to be used when we must modify the tree to remove a Leaf and replace it by another type of node.

The TreeNode class is used to store the type of a node, to store information of that node.
So if the node's type is leaf → it will store a pointer to a trapezoid.
If the node's type is X-Node → it stores a segment end point
If the node's type is Y-Node → it stores a segment
Each of its constructors is used for a specific use.

Graph.h

Contains the graph class along with the graphNode class

The graph class is used to generate the nodes after adjusting the trapezoids and removing the solid trapezoids.
It has the array of nodes in the graph.
The number of nodes in the graph.
A pointer to the AI algorithm that will build the shortest path between 2 nodes.
It has an array of the nodes' indices that the shortest path will generate.
Also it contains many variables used to interpolate the motion of the character from a node to another along with the interpolation factor (the movement speed).
- `Add_Node` a function to add a node to the array of nodes, it will automatically assign a state to this node.
- `Link_Nodes` a function that links two nodes according to their indices.
- `Init_ShortestPath` a function that inits the shortest path algorithm and computes the distance array according to the nodes in the graph.
- `Get_Path` a function that gets a path from 2 nodes, the first parameter is the source node's index and second is the destination's node index. Plus it returns the path array as string to be set in the message box.
- `Get_NearestNode` a function that gets the nearest node to the user's position set by the mouse cursor. It calculates the distance from the picked position to each node in the graph.
- `Traverse_Path` a function that moves the character along the shortest path in an interpolated way. So it first moves the character from the picked point to the source node then from the source to the destination then from the destination to the destination picked point.
- `Draw` a function that draws the graph's nodes and the edges from each node to its neighbors along with the node's indices.

The graphNode class is used to store the position of the node in the world coordinate system.
It stores its model (shape)
It stores its neighbors.
It stores an index of the node.
- `Draw` a function to draw the model of the node.
- `DisplayIndex` a function that displays the index of the current node.

Floyd.h

The AI shortest Path class contains the number of nodes.
Contains an array to store the distances from each two neighbors.
Contains the dieze value that is by default (10000).
Contains the d value that is the direct neighbour.
- Constructor allocated memory for the arrays then initializes their content.
- `Compute_PathArray` the function that computes the shortest path between all the nodes.

- `Get_Path` a function that gets the shortest path from any two nodes.

Main.cpp

The main creates the model that represents the solid in the world
Creates the main character model.
Creates the Trapezoidal_Algorithm variable that builds the map and generate the graph along with the shortest path.
Checks for the used input and checks if he chose a solid node.