**Develop technical documentation**

**Sunsiy10**

**2014.12.25**

1. **The framework structure of the program**

According to the implementation process of the function, this program is divided into Boolean operation module and legitimacy check module.

* 1. **Boolean operation module**

The Boolean operation module can be divided into three parts according to the process, namely, converting general polygons into BspTree, merging BspTree, and calculating the edges of the resulting polygons. At implementation time, BspTree's nodes are represented by CP\_BSPNode data structures in code.

**1.1.1 Convert general polygons to BspTree**

The functions involved in this part are mainly gb\_buildPolygonBSPTree, gb\_buildRegionBSPTree and gb\_buildLoopBSPTree.

gb\_buildPolygonBSPTree convert CP\_Polygon to BspTree:

At implementation time, the CP\_Region contained in the CP\_Polygon are converted to BspTree in turn, and then these BspTrees are merged with a Boolean operation "merge", and finally the resulting BspTree is returned. The merge operation is described in Section 1.2.

gb\_buildRegionBSPTree convert CP\_Region to BspTree:

When implemented, all CP\_Loop in the CP\_Region are converted to BspTree, and then the BspTree of the outer ring is merged with the Boolean operation "difference" of all the BspTrees of the inner ring in turn, and finally the obtained is returned BspTree。

gb\_buildLoopBSPTree convert CP\_Loop to BspTree. The specific process is described in section 5.1 of the principle document.

* + 1. **Merger of BspTree**

The functions involved in this part are mainly gb\_mergeBSPTree, gb\_partitionBspt and gb\_mergeTreeWithCell.

gb\_mergeBSPTree implement a merge of the two BspTrees A and B of the input:

When one of A and B is a leaf node, call gb\_mergeTreeWithCell merge A and B under the condition of specified Boolean operation and return the merged result;

Otherwise, B is divided into two tree B\_left and B\_right with the partition of the A root node, and this process is implemented by calling the gb\_partitionBspt. Then call gb\_mergeBSPTree merge A's left child and B\_left, A's right child and B\_right, respectively. Returns the result of the merge.

gb\_mergeTreeWithCell realize the merging of the leaf nodes of BspTree and BspTree, the corresponding merge result can be obtained by specifying Boolean operations.

gb\_partitionBspt implementation splits a BspTree T into two T\_left and T\_right with a partition (equivalent to a straight line), one on either side of the partition. At the time of implementation, determine the position relationship between the partition and the partition of the T root node in 8 cases to determine whether it is necessary to continue to use the partition to divide the left and right children of T. For a detailed introduction, see Section 3 of the Principles document.

* + 1. **Calculates the edges of the resulting polygon**

The functions involved in this part are mainly gb\_generateCellPolygons and gb\_generateBSPTreeFaces.

gb\_generateCellPolygons the boundaries of the regions corresponding to each leaf node are obtained, and the contribution of intermediate nodes from the root node to the leaf nodes to the regional boundaries is recorded. For specific introductions, see Section 4 of the Principles document.

gb\_generateBSPTreeFaces get the boundary of the resulting polygon. For specific introductions, see Section 4 of the Principles document.

* 1. **Legality check module**

The legitimacy check module is used to determine the legitimacy of general polygons, and this part mainly has the following functions:

gb\_checkPolygon judge whether the general polygon is legal, and judge the legitimacy of each region by calling gb\_checkRegion inside it; By calling the gb\_checkRegionCrossRegion, it is judged whether there is an illegal situation where the two regions intersect; Then convert the region to BspTree, and determine whether one or two regions overlap by calling gb\_tree1OverlapWithTree2.

gb\_checkRegion judge the legitimacy of each region, and judge whether the self-intersection of the ring and the direction of the inner ring of the outer ring are correct by calling the gb\_checkLoopSelfIntersection inside it; By calling the gb\_checkLoopIntersection, determine whether there is an illegal situation of intersection between the rings; Then convert the ring into BspTree, and determine whether the inner rings cover each other by calling the gb\_tree1OverlapWithTree2; By calling the gb\_tree1InTree2, determine whether the inner ring is inside the outer ring.

1. **Programming skills**

**2.1 Debug assistance for BspTree mergers**

When debugging the merge operation of BspTree, because the CP\_BSPNode nodes of the tree structure are not easy to view their information in the debugging window, it is relatively easy to debug errors by outputting this structure to a document at implementation time and indicating the partition of each node.

**2.2 Legality Check**

When checking the legitimacy of general polygons, the structural properties of BspTree are utilized.

By converting two different regions into BspTree and performing Boolean operations "intersect" operations, the result obtained is that if BspTree contains leaf nodes with attribute "in", there are overlapping parts with an area greater than 0 between the two regions. This feature can be used to assist with legality checks. Similarly, this method can also help determine whether there is an overlap between the inner rings with an area greater than 0.

By converting the two rings L1 and L2 to BspTree, the BspTree of L1 and L2 is Boolean "difference" (L1). – L2), the resulting BspTree does not contain leaf nodes with attribute "in", then L1 is inside L2 。 This strategy can be used to help determine whether the inner ring is inside the outer ring.

**2.3 Memory Management**

Use the global vector container to record dynamically requested memory, and there is also a vector container within the class to record dynamically requested memory. In this way, after the end of a Boolean operation, all containers are used to release the memory applied outside the class, and the memory dynamically requested by the class internal function can be released when the class is destroyed, that is, the memory applied in the container is released.

1. **The method by which the program is compiled, run, and configured**

The project was written in the VS2012 environment and is based on MFC's Ribbon graphical interface. When compiling, you need to use VS2012 to open the project, press Ctrl + F5 to compile and run the program.