# RectMesh构建

BR顶点

TL顶点

BR顶点

TR顶点

BL顶点

TL顶点

注：

1. 暗色face是虚拟的，只是为了保证HalfEdge逻辑的完整性，边界情况无须判空逻辑
2. 为了可视化，这里将虚face做了翻转，其实就2个虚face，同一颜色的是同一个虚face
3. 边界4条edge的oppositeEdge都是虚edge
4. 红色矩形为ConstraintShape，该shape有4条ConstraintSegment

BR顶点

TL顶点

4个face（2实+2虚），所有face都是逆时针序，具体信息如下：

|  |  |  |
| --- | --- | --- |
| **face** | **edge** | **isReal** |
| fTL\_BL\_TR | eBL\_TR | true |
| fTR\_BL\_BR | eTR\_BL | true |
| fTL\_BR\_BL | eBR\_BL | false |
| fTL\_TR\_BR | eTL\_TR | false |

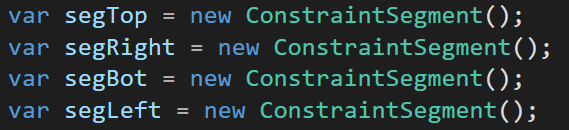
总共12条edge（10实+2虚，只有对角线2条edge是虚的，8条约束边+4条非约束边），具体信息如下：

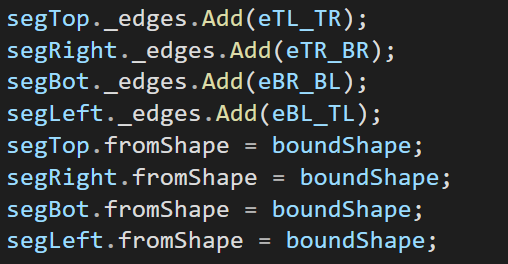
（注：nextLeftEdge是edge逆时针方向的下一个相连的edge）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **edge** | **originVertex** | **oppositeEdge** | **nextLeftEdge** | **leftFace** | **isReal** | **isConstrained** |
| eTL\_TR | vTL | eTR\_TL | eTR\_BR | fTL\_TR\_BR | true | false |
| eTR\_TL | vTR | eTL\_TR | eTL\_BL | fTL\_BL\_TR | true | false |
| eTR\_BR | vTR | eBR\_TR | eBR\_TL | fTL\_TR\_BR | true | true |
| eBR\_TR | vBR | eTR\_BR | eTR\_BL | fTR\_BL\_BR | true | true |
| eBR\_BL | vBR | eBL\_BR | eBL\_TL | fTL\_BR\_BL | true | true |
| eBL\_BR | vBL | eBR\_BL | eBR\_TR | fTR\_BL\_BR | true | true |
| eBL\_TL | vBL | eTL\_BL | eTL\_BR | fTL\_BR\_BL | true | true |
| eTL\_BL | vTL | eBL\_TL | eBL\_TR | fTL\_BL\_TR | true | true |
| eTR\_BL | vTR | eBL\_TR | eBL\_BR | fTR\_BL\_BR | true | false |
| eBL\_TR | vBL | eTR\_BL | eTR\_TL | fTL\_BL\_TR | true | false |
| eTL\_BR | vTL | eBR\_TL | eBR\_BL | fTL\_BR\_BL | false | false |
| eBR\_TL | vBR | eTL\_BR | eTL\_TR | fTL\_TR\_BR | false | false |

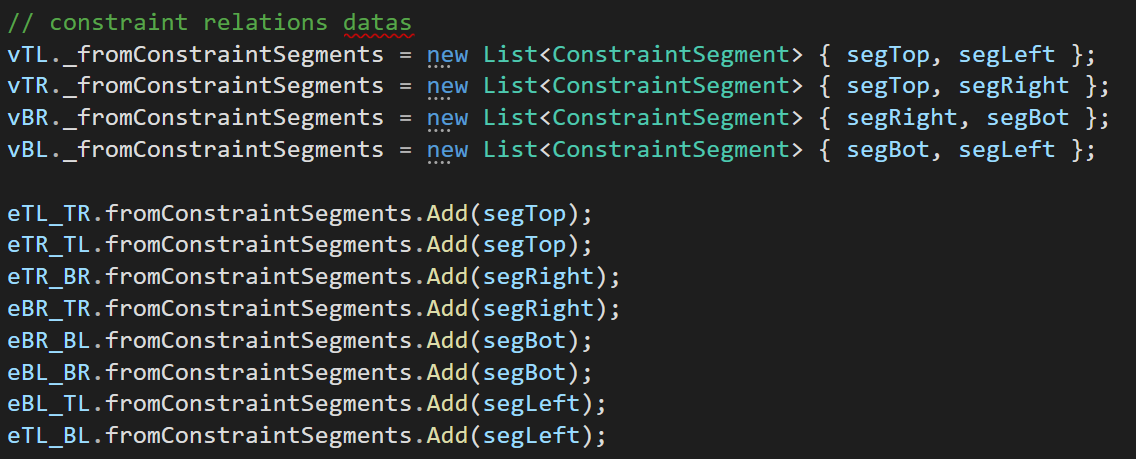
红色矩形为ConstraintShape，该shape有4条ConstraintSegment（顺时针序？）

（1）构建Segement和Edge关系（1对N关系）

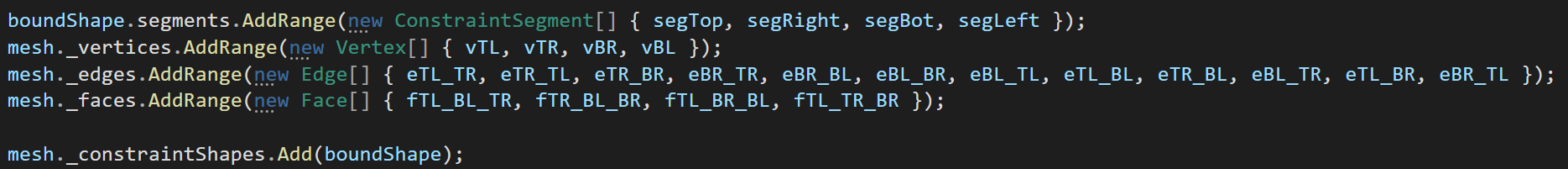




（2）关联Vertex和Edge和Segment，方便删除Segment使用，1个顶点对应2个segment，1个edge对应1个segment



构建ConstraintShape和Mesh数据



# Mesh. insertConstraintSegment

## 基于Bounds裁剪Segment

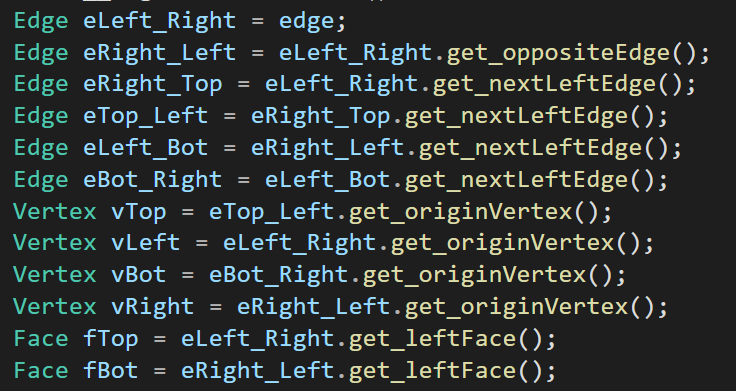
## Mesh. insertVertex

### Geom2D.locatePosition

基于传入的坐标点（x，y）找到和当前Mesh的相交信息（可能是Vertex，Edge，Face这3种情况）

1. 如果找到的是Vertex，直接返回该顶点即可，insertVertex完成
2. 如果找到的是Edge，则执行Mesh.splitEdge函数，将edge分割
3. 如果找到的是Face，则执行Mesh.splitFace函数，将face分割

## Mesh.splitEdge



vLeft

vRight

vTop

vBot

新的顶点命名为vCenter

fTop

fBot

edge

**vCenter**

如图edge被1分为2，两个相邻face被分成4个face。新增的Vertex，Edge，Face以及原来的4个邻接（如图灰色）需要重新构建HalfEdge关系

1. 新增1个Vertex：vCenter
2. 新增8个Edge：eTop\_Center，eCenter\_Top，eBot\_Center，eCenter\_Bot，eLeft\_Center，eCenter\_Left，eRight\_Center，eCenter\_Right
3. 新增4个Face：fTopLeft，fLeftBot，fBotRight，fRightTop

构建HalfEdge关系如下：

vCenter.setDatas((fTop.get\_isReal()) ? eCenter\_Top : eCenter\_Bot);

vCenter.\_pos.X = x;

vCenter.\_pos.Y = y;

Geom2D.projectOrthogonaly(ref vCenter.\_pos, eLeft\_Right);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **edge** | **originVertex** | **oppositeEdge** | **nextLeftEdge** | **leftFace** | **isReal** | **isConstrained** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## Mesh.splitFace