kdTree Pseudocode

Caveat

Pseudocode is there to help understand the structure of code

- Not sufficient to implement the code!
- Not a substitute for thinking through the problem!
- Make sure you understand the reasoning before implementing!

SplitNode properties:

- axis
- position
- kdTree* left
- kdTree* right

LeafNode properties:

- objList

```
function buildTree(objList, boundingBox, depth, leafSize):
    if (objList.size <= leafSize or ++depth == depthLimit) then
        return LeafNode(objList)
    bestPlane ← findBestSplitPlane(objList, boundingBox)
    for each obj in objList:
        if (obj.boundingBoxMinOnBestAxis < bestPlane.position)
            then addToLeftList
        if (obj.boundingBoxMaxOnBestAxis > bestPlane.position)
            then addToRightList
        if (bestPlane.position equals obj.bBMaxOnAxis and
            bestPosition equals obj.bBMinOnAxis && obj.N < 0) then
            addToLeftList
        else if (bestPlane.position equals obj.bBMaxOnAxis and
            obj.bBMinOnAxis and obj.N >= 0) then addToRightList
    if (rightList.isEmpty or leftList.isEmpty) then return
        LeafNode(objList)
    else return SplitNode(bestPlane.position, bestPlane.axis,
        buildTree(leftList, bestPlane.leftBBox, depth, leafSize),
        buildTree(rightList, bestPlane.rightBBox, depth, leafSize))
```

```
function findBestSplitPlane(objList, boundingBox):
    for each axis:
        for each object:
            SplitPlane p1.position = obj.bBMinOnAxis
            SplitPlane p2.position = obj.bBMaxOnAxis
            candidateList.pushback(p1)
            candidateList.pushback(p2)
    for each plane in candidateList:
        plane.leftCount = countLeftObjects()
        plane.leftBBoxArea = calculateLeftBBox()
        plane.rightCount = countRightObjects()
        plane.rightBBoxArea = calculateRightBBox()
    for each plane in candidateList:
        SAM = (plane.leftCount * plane.leftBBoxArea + plane.rightCount
            * plane.rightBBoxArea)/boundingBox
        if (SAM < minSam) then
            minSam = SAM
            bestPlane = plane
    return bestPlane
```

```
function SplitNode::findIntersection(r, i, tmin, tmax):
    if (ray is nearly parallel to split plane) then
        calculateAsNearParallel()
    else
        if (ray hits only left bounding box) then
            if (left→findIntersection(r, i, tmin, tmax)) return true
        else if (ray hits only right bounding box) then
            if (right→findIntersection(r, i, tmin, tmax)) return true
        else
            if (findNearestIntersection(r, i, tmin, tmax)) return true
            if (findFartherIntersection(r, i, tmin, tmax)) return true
    return false
```

```
function LeafNode::findIntersection(r, i,
   tmin, tmax):
    for each obj in objList:
        isect c_i
        if (obj.intersect(r, c_i) and c_i.t >=
           tmin and c_i.t <= tmax) then
            i = c_i
```

Handling Trimeshes

Note that a trimesh is a **single** object

Naively building a kd-tree to handle this object will **not** accelerate the individual faces

Place the **faces** of the trimesh directly into the kd-tree to gain speed ups

C++ Construction

- Nodes in kdTree can either be SplitNodes or LeafNodes
- Both node types exist within same tree structure
- LeafNodes need to contain a general-purpose vector to store pointers to Geometry objects
  - Will want some templating/virtual functions across all kdTree classes to handle this

C++ Class/Struct Suggestions

- kdTree builds the tree
- SplitNode stores split plane and left/right children
- LeafNode stores geometry
- Both SplitNode and LeafNode need to find intersections (broad vs narrow phase)

C++ Template Example

```
template <typename Objects>
class kdTree;
template <typename Objects>
class SplitNode;
template <typename Objects>
class LeafNode;
```

C++ Template Example

```
template <typename Objects>
class LeafNode : kdTree<Objects> {
    std::vector<Objects*> objectList;
    //constructor/destructor + functions here
}
```