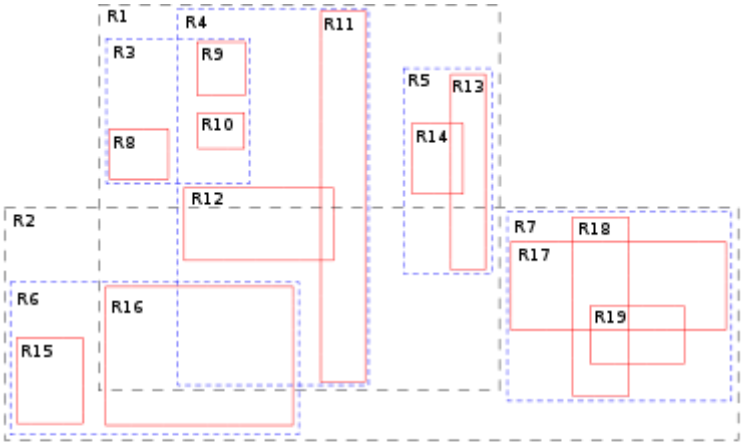
维基百科

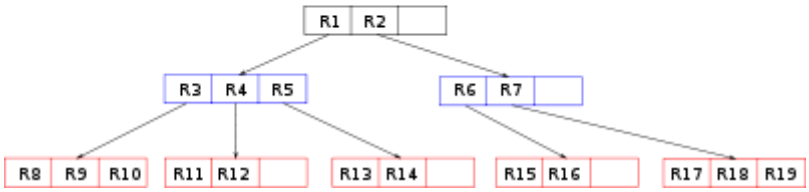
R树

维基百科，自由的百科全书

R树是用来做空间数据存储的树状数据结构。例如给地理位置，矩形和多边形这类多维数据建立索引。R树是由Antonin Guttman于1984年提出的[1]。人们随后发现它在理论和应用方面都非常实用[2]。 在现实生活中，R树可以用来存储地图上的空间信息，例如餐馆地址，或者地图上用来构造街道，建筑，湖泊边缘和海岸线的多边形。然后可以用它来回答“查找距离我2千米以内的博物馆”，“检索距离我2千米以内的所有路段”（然后显示在导航系统中）或者“查找（直线距离）最近的加油站”这类问题。R树还可以用来加速使用包括大圆距离[3]在内的各种距离度量方式的最邻近搜索[4]。

R树	
类型	树
发明时间	1984
发明者	Antonin Guttman
用大O符号表示的时间复杂度	
算法	平均 最差
搜索	$O(\log_M n)$





一个简单的二维矩形上的R树的例子

目录

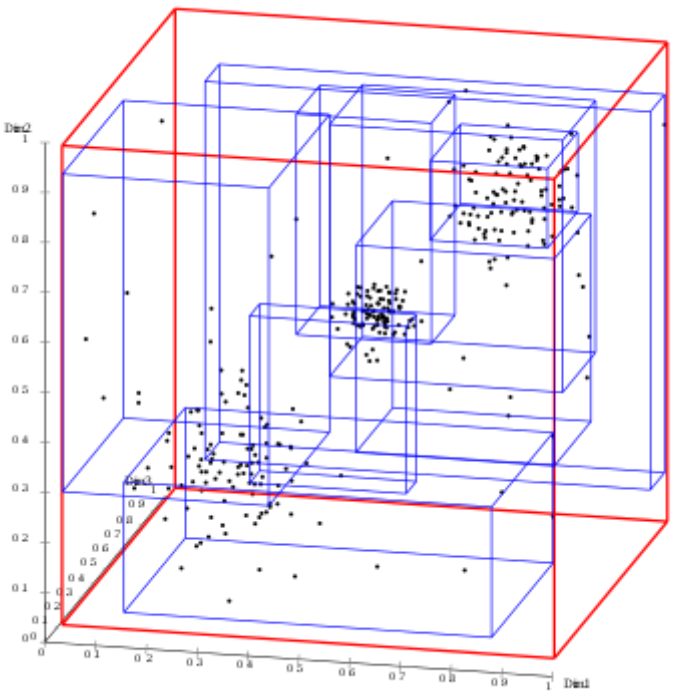
R树的原理

R树的核心思想是聚合距离相近的节点并在树结构的上一层将其表示为这些节点的最小外接矩形，这个最小外接矩形就成为上一层的一个节点。R树的“R”代表“Rectangle（矩形）”。因为所有节点都在它们的最小外接矩形中，所以跟某个矩形不相交的查询就一定跟这个矩形中的所有节点都不相交。叶子节点上的每个矩形都代表一个对象，节点都是对象的聚合，并且越往上层聚合的对象就越多。也可以把每一层看做是对数据集的近似，叶子节点层是最细粒度的近似，与数据集相似度100%，越往上层越粗糙。

跟B树类似，R树也是平衡树（所以所有叶子节点都在同一深度）。它把数据按（内存）页存储，是为磁盘存储设计的（跟数据库的做法相同）。每一页可以存储不超过一个上限的条目，这个上限一

目数在条目上限的30%–40%时性能最佳，而B树会维持条目上限的50%，B*树甚至维持条目上限的66%。这么做的原因是平衡空间数据比平衡B树上的线性数据更复杂。

跟其他树结构一样，R树的搜索算法（例如：交集，子集，最邻近搜索）也非常简单。核心思想是画出查询语句相应的边框，并用它来决定要不要搜索某个子树。这样在搜索时可以跳过树上的大部分节点。跟B树类似，这个特性让R树可以把整棵树放在磁盘里，在需要的时候再把节点读进内存页，从而可以应用在大数据集和数据库上。



用ELKI做出的三维R*树视觉效果图（每个立方体是一个目录页）

R树的主要难点在于构建一棵既能保持平衡（所有叶子节点在同一层），又能让树上的矩形既不包括太多空白区域也不过多相交（这样在搜索的时候可以处理尽量少的子树）的高效的树。例如，最初的通过插入节点来构建一棵高效的R树的构想是选择一棵子树插入，使得对其外接矩形的扩张最小。填满一页后，把数据分成两份，使它们分别包括尽量小的区域。大部分关于R树的研究和改进都是关于如何改进建树的过程。它们可以分为两类，一类是如何从头开始构建一棵高效的树（被称为批量加载），另一类是如何在一棵已经存在的树上插入和删除。

R树不保证最坏情况下的性能，但是在现实数据[5]上一般表现不错。理论上来说，批量加载的优先级R树是最坏情况下的最优解[6]，但由于复杂度太高，目前还没有在实际应用中获得关注。

当数据被构建成R树时，任意Lp空间中的数据的最近k个邻居都可以很高效地用空间交集计算[7]。这对很多基于最近邻居法的算法（例如本地异常因子算法）都很有帮助。 DeLi-Clu[8]提出的Density-Link-Clustering是一种使用R树来进行空间交集，从而高效地计算OPTICS聚类的聚类分析算法.

变体

- R*树
- R+树
- Hilbert R树
- X树

算法

数据格式

R树的数据按（内存）页存储，每一页存储多条数据，数据条数不超过一个事先定义的最大值，一般会多于最小值。非叶子节点上的每一条数据由指向子节点的标识符和该子节点的外接矩形组成。

搜索

输入是一个搜索矩形（搜索框）。搜索算法和B+树类似，从根节点开始，每个非叶子节点包含一系列外接矩形和指向对应子节点的指针，而叶子节点则包含空间对象的外接矩形以及这些空间对象（或者指向它们的指针）。对于搜索路径上的每个节点，遍历其中的外接矩形，如果与搜索框相交就深入对应的子节点继续搜索。这样递归地搜索下去，直到所有相交的矩形都被访问过为止。如果搜索到一个叶子节点，则尝试比较搜索框与它的外接矩形和数据（如果有的话），如果该叶子节点在搜索框中，则把它加入搜索结果。

对于最邻近搜索这类优先级搜索，可以查询一个矩形或者一个点。 先把根节点加入优先队列，然后查询优先队列中离查询的矩形或者点最近的项，直到优先队列被清空或者已经得到要求的邻居数。从优先队列顶端取出每个节点时先将其展开，然后把它的子节点插回优先队列。如果取出的是子节点就直接加入搜索结果[9]。这个方法可以应用于包括（地理信息的）大圆距离在内的多种距离度量方式。[3]

插入

插入节点时，算法从树的根节点开始递归地向下遍历。检查当前节点中所有外接矩形，并启发式地选择在哪个子节点中插入（例如选择插入后外接矩形扩张最小的那个子节点），然后进入选择的那个子节点继续检查，直到到达叶子节点。满的叶子节点应该在插入之前分割，所以插入时到达的叶子节点一定有空位来写数据。由于把整棵树遍历一遍代价太大，在分割叶子节点时应该使用启发式算法。把新分割的节点添加进上一层节点，如果这个操作填满了上一层，就继续分割，直到到达根节点。如果跟节点也被填满，就分割根节点并创建一个新的根节点，这样树就多了一层。

选择插入的子树

插入算法在树的每一层都要决定把新的数据插入到哪个子树中。如果只有一个外接矩形包含新数据，那么很容易选择。但如果有多个外接矩形包含新数据，或者所有外界矩形都不包含新数据，那么选择算法会极大地影响R树的性能。

经典R树的实现会把数据插入到外接矩形需要增长面积最小的那个子树。而更进一步的R*树则使用了一种混合的启发式算法。如果需要选择的子树是叶子节点，它会选择造成最小重叠的子树，如果有多个选择不分上下，则选择外接矩形增长最小的子树，再不分上下则选择面积最小的子树。如果需要选择的子树不是叶子节点，算法和R树类似，但在不分上下时选择面积较小的子树。比R树更少的重叠是R*树对R树的主要优势之一（这也有R*树上应用的其它启发式算法的原因，并不仅仅因为选择子树算法的不同）。

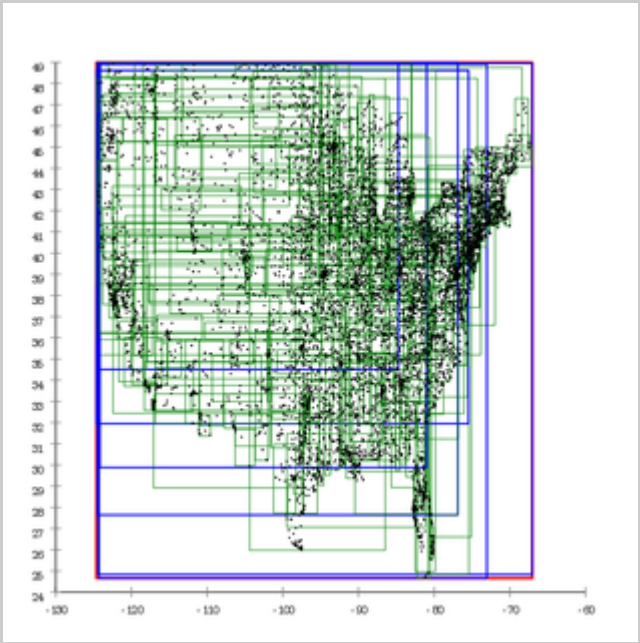
分割溢出的节点

因为有指数多种方式来把一个节点中所有对象分割到两个节点，我们需要使用启发式算法来寻找最优分割。在经典R树中，Guttman提出了二次分割和线性分割两种启发式算法。二次分割算法在节点中查找放在同一个节点中最糟糕的两个对象，把它们作为两个新节点的初始数据。然后把数据集中对其中一个新节点有最强偏向的数据（外接矩形最小面积增长）分给这个节点，直到所有数据都被分配（节点中数据多于最小条目数）。

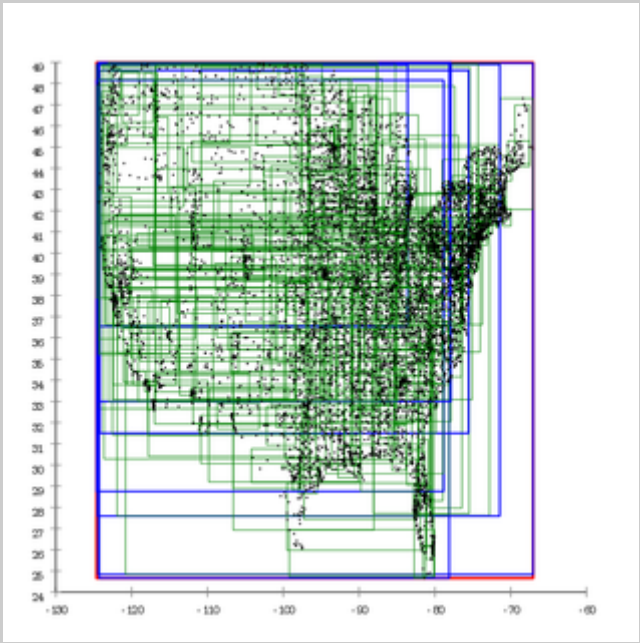
还有其它的分割策略，例如Greene's Split[10]，R*树启发式分割策略[11]（它也试图尽量减小重叠，但同时也偏向正方形的节点）以及Chuan-Heng Ang和T. C. Tan提出的线性分割算法[12]（这种算法会产生形状很不规则的矩形，在很多现实数据分布以及使用搜索框时效率较低）。R*树不仅使用了更高级的启发式分割，而且还通过重新插入一些数值来试图避免分割节点（这跟B树平衡溢出节点的策略类似）。这个方法被证明还可以减少重叠，从而进一步提高R树性能。

最后，X树[13]（可以被视为R*树的变体）也决定避免分割节点。当X树找不到一个好的分割方式（尤其在高维数据的情况下）时会构建一个所谓的“超节点”来放置溢出的数据。

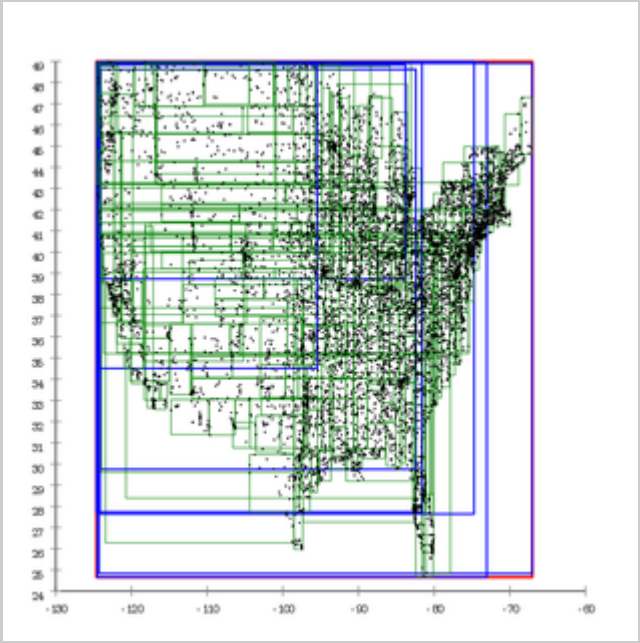
不同启发式分割算法在美国邮政区划数据库上的效果图



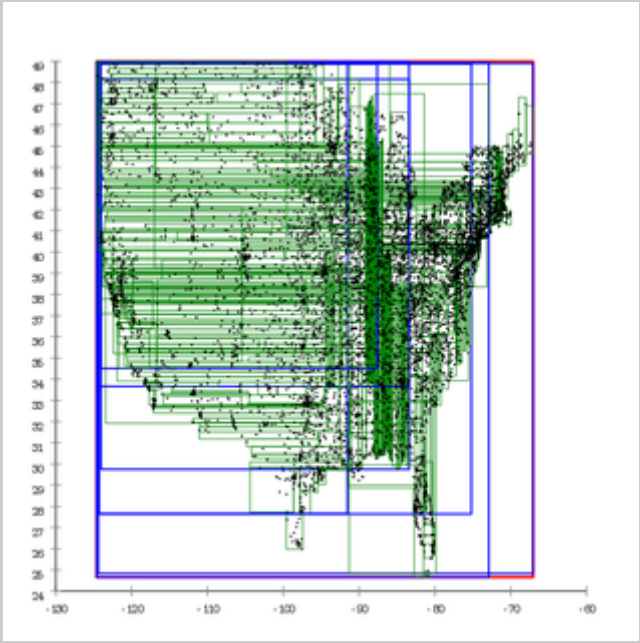
Guttman的二次分割。[1]
这张图上有很多重叠的子树。



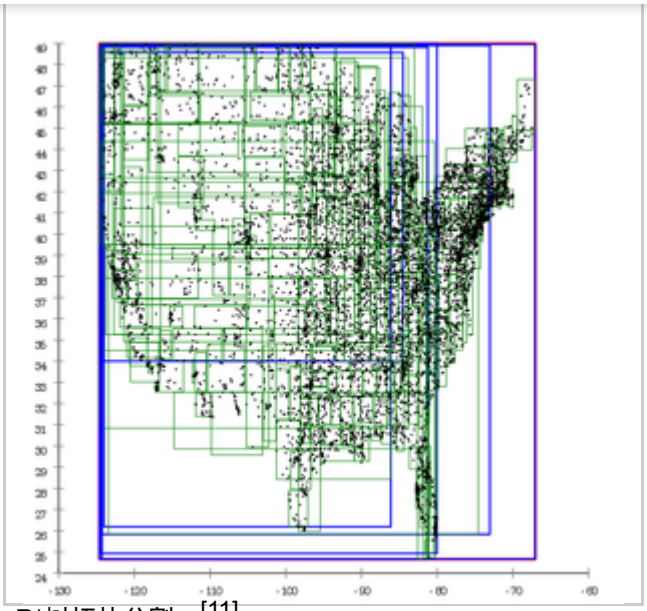
Guttman的线性分割。[1]
结构更加糟糕，但是构建更快。



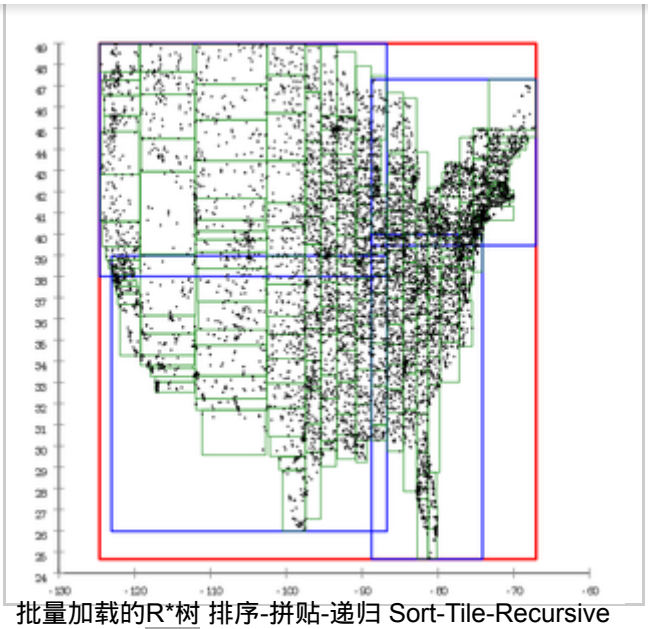
Greene's split.[10] 图上的重叠比Guttman的策略少很多。



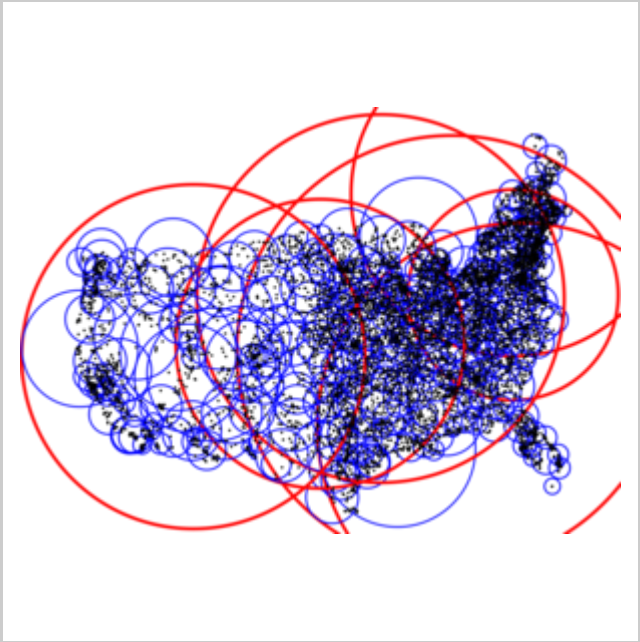
Chuan-Heng Ang和T. C. Tan的线性分割算法。[12]
这个策略导致子树变成一条一条的，在查询时通常会很糟糕。



R*树拓扑分割。[11]
图中的重叠显著减少，因为R*树试图最小化节点重叠，并且重插入策略进一步优化了树结构。分割算法更倾向于分割出近似正方形的节点，这在一般的地图查询中会有更好的性能。



批量加载的R*树 排序-拼贴-递归 Sort-Tile-Recursive (STR).
叶子节点完全不重叠，非叶子节点（目录节点）只有少量重叠。这是一种非常高效的树，但是需要在构建树之前完全了解所有数据。



M树和R树类似，但它使用嵌套的圆型节点。
分割这种节点异常复杂，并且节点间的重叠通常也比其它算法多得多。

删除

从一个节点中删除一条数据可能会导致算法需要更新它的父节点的外接矩形。但是当一个节点中的数据不足时，它不会和兄弟节点重新平衡。这个节点会被直接删除，它的子节点（可能是子树，不

批量加载

- X轴最近原则 – 把数据按照它们第一坐标值（X轴坐标）排序，然后按照想要的大小分割。
- 批量Hilbert R树 – X轴最近原则的变体。把数据按照它们的外接矩形中心点的Hilbert值排序，然后分割。不能保证节点之间没有重叠。
- 排序-拼贴-递归：Sort-Tile-Recursive (STR):[14] X轴最近原则的另一种变体，先估算需要的叶子节点数：$l = \lceil$对象数量/节点容量$\rceil$和每个维度上需要分割的份数：$s = \lceil l^{1/d} \rceil$。然后依次把每个维度分为$s$等分。分割之后如果有哪一份的数据超过了条目上限就使用相同的方法继续等分。如果是点数据，叶子节点间不会重叠。算法会拼接一些数据块，使得所有叶子节点的数据量大致相当。
- 优先级R树

参见

- 线段树
- 区间树 – 一维数据集（通常是时间数据）上的退化的R树。
- Bounding volume hierarchy
- Spatial index
- GiST

参考资料

1. Guttman, A. (1984).
2. Y. Manolopoulos; A. Nanopoulos; Y. Theodoridis (2006).
3. Schubert, E.; Zimek, A.; Kriegel, H. P. (2013).
4. Roussopoulos, N.; Kelley, S.; Vincent, F. D. R. (1995).
5. Hwang, S.; Kwon, K.; Cha, S. K.; Lee, B. S. (2003).
6. Arge, L.; De Berg, M.; Haverkort, H. J.; Yi, K. (2004).
7. Brinkhoff, T.; Kriegel, H. P.; Seeger, B. (1993).
8. Achtert, E.; Böhm, C.; Kröger, P. (2006).
9. Kuan, J.; Lewis, P. (1997).
10. Greene, D. (1989).
11. Beckmann, N.; Kriegel, H. P.; Schneider, R.; Seeger, B. (1990).
12. Ang, C. H.; Tan, T. C. (1997).
13. Berchtold, Stefan; Keim, Daniel A.; Kriegel, Hans-Peter (1996).
14. Leutenegger, Scott T.; Edgington, Jeffrey M.; Lopez, Mario A. (February 1997).

外部链接

- R树的代码： C & C++ (http://superliminal.com/sources/sources.htm#C%20&%20C++%20Code)（页面存档备份 (https://web.archive.org/web/20160912212609/http://superliminal.com/sources/sources.htm#C%20&%20C++%20Code)，存于互联网档案馆）, Java (https://github.com/davidmoten/rtree)（页面存档备份 (https://web.archive.org/web/20170413235921/https://github.com/davidmoten/rtree)，存于互联网档案馆）, Java applet (http://gis.umb.no/gis/applets/rtree2/jdk1.1/)（页面存档备份 (https://web.archive.org/web/20070501040448/http://gis.umb.no/gis/applets/rtree2/jdk1.1/)，存于互联网档案馆）, Common Lisp (http://www.cliki.net/spatial-trees)（页面存档备份 (https://web.archive.org/web/20161106185847/http://www.cliki.net/spatial-trees)

Scala (https://github.com/meetup/archery) (页面存档备份 (https://web.archive.org/web/20180611020509/https://github.com/meetup/archery)，存于互联网档案馆) , Javascript (https://github.com/imbcmdth/RTree) (页面存档备份 (https://web.archive.org/web/20160221162056/https://github.com/imbcmdth/RTree)，存于互联网档案馆) , Javascript (including interactive demo) (http://rtree2d.hepburnave.com) (页面存档备份 (https://web.archive.org/web/20190118085828/http://rtree2d.hepburnave.com/)，存于互联网档案馆)

- Boost.Geometry library containing R-tree implementation (various splitting algorithms) (http://www.boost.org/doc/libs/release/libs/geometry/doc/html/index.html)
