UNIVERSITY OF WATERLOO
Faculty of Mathematics

Exploring the application of Space Partitioning Methods on river segments

S.S. Papadopulos & Associates
Bethesda, MD, US

Max Ren
20413992
3A Computer Science/BBA Double Degree
May 2014

Executive Summary

Nearing the end of my co-op term, I was tasked with aiding in the development of code for a grid approximation of a river. By setting a grid structure to a river, it allows for a better estimation of how the surrounding land will be affected given a water well location in that grid. The input files of the river, along with the graphical nature of the task, lead me to think about how to best partition the given space. Two methods of space partitioning are explored within this report, quadtrees and k-d trees. These two methods lie under what is referred to as binary space partitioning, quadtrees being an exception as it is not binary. Through the development of C++ pseudo code, drawing out the partitioned cells, and building the trees, it is evident the differences that these two methods produce. While the quad tree method produces even grids, the k-d tree does not. This leads to different implications and applications to what can be accomplished with the two methods. However for this specific application of dividing up the area closest to the river, the quadtree method proves to provide a better partitioning system. This is due to the fact that the cells are subdivided evenly; cutting the space into quarters every time, whereas the k-d tree method depends on splitting the space based on the location of the middle point. However for a different usage case, where the criteria may be to provide collision detection or maximizing the percentage of the river to be contained within each cell, it may be better to use a k-d tree to partition the space.

1.0 Introduction

Binary space partitioning is a method for recursively subdividing a space into convex sets by hyperplanes (Golodetz, 2008). This recursive subdivision gives rise to a representation of the space by ways of a tree structure, known as a binary space (BSP) tree. Two methods of partitioning that this report will focus on are quadtrees and k-d trees. These two methods differ in the way a given space is partitioned and as such can serve different purposes. As there can be many applications of space partitioning, one use that was explored in during my work term is the partitioning of a river. A river in this context will be described as a series of points connected by line segments. As it is how the graphical file for the river is set up, the partitions will be created based on dividing the points of the river, the very basic method of space partitioning. Through the description of algorithms in pseudo C++ code, this report will serve to explore and analyze, for a given river, the appropriateness of the method in subdividing the approximate area which surrounds a given river.

1.1 What is a Quad Tree

A quadtree is a type of tree data structure in which each node has exactly four children. Typically divided into northeast (NE), southeast (SE), southwest (SW), and northwest (NW). The method of subdivision on a given space with points is to continuously cut the space into even quarters until there is only one point per section.

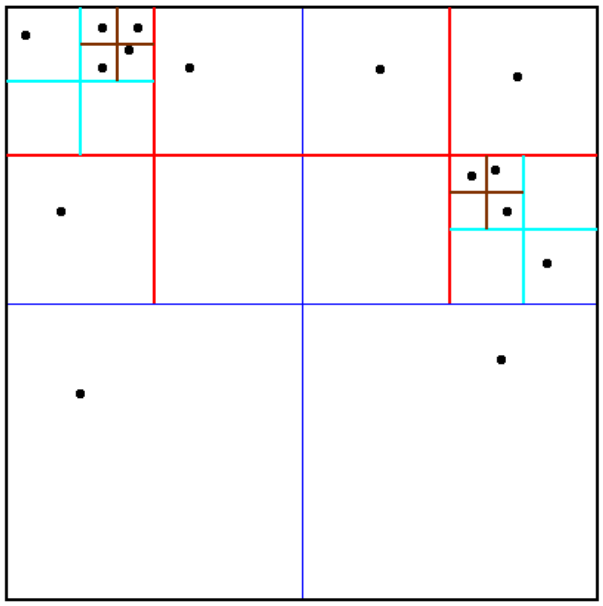

Figure 1 Quad tree partition example

As depicted in the figure above, the original space is divided into 4 quadrants. Subsequent quadrants that still contain more than one point are then further subdivided. Some subdivisions will contain less than 4 child nodes, with 4 being the max. The tree representation for the figure above is therefore:

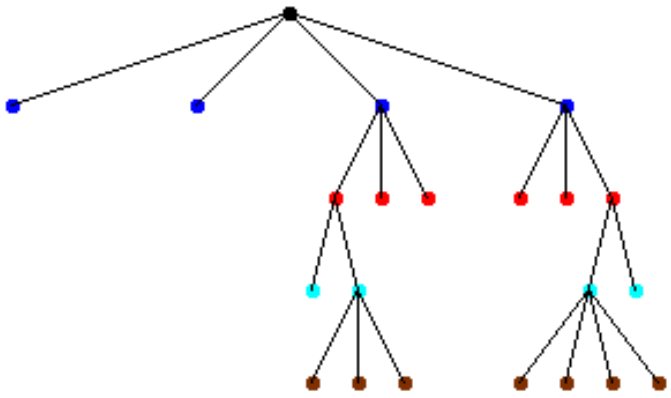

Figure 2 Quad tree example

Furthermore the C++ code structure can look something as follows:

```
struct Quadtree{
  double x, y, dx, dy;
  //x and y are the coordinates of the top right corner
  //dx and dy are the lengths of the cell
  Node node;
  Quadtree  *nw, *ne, *se, *sw;
};
```

Figure 3 Quad tree C++ struct

1.2 What is a KD Tree

An alternate to the quadtree is the k-d tree. As with the quadtree, the starting point is a space with points. Instead of quadrants, this method divides the space on the x and y axes alternately. Furthermore, instead of encapsulating each point inside of a quadrant, the k-d tree creates its grid by making a cut along the point. Given a set of points in the space, the points are first divided on the x axis based on the point with the middle x value.

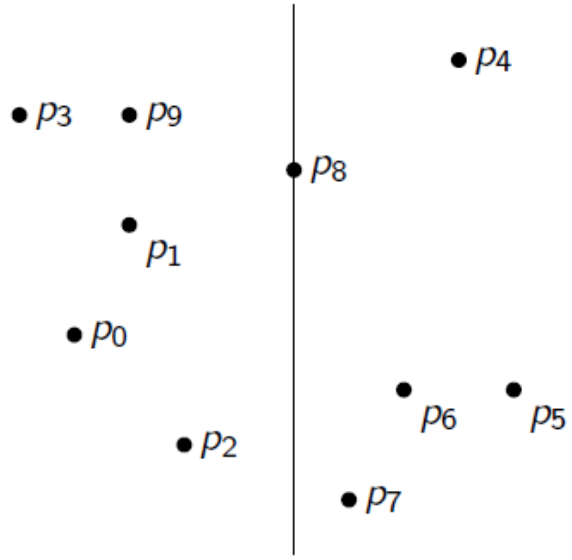

Figure 4 K-D tree first partition: partition on the x axis

From here, each side is then divided on the y axis based on the point with the middle y value.

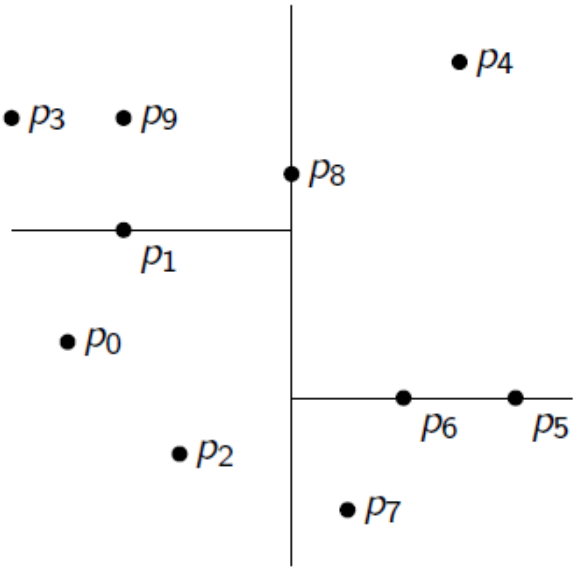

Figure 5 K-D tree partition on the y axis

This continues until no more cuts can be made.

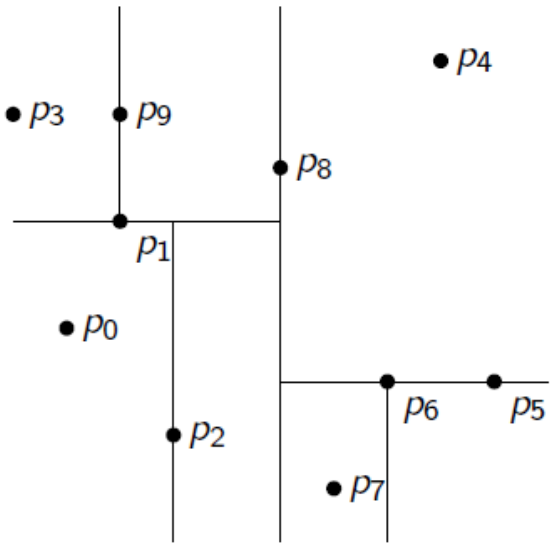

Figure 6 Complete k-d partitioning

This means that no single node has more than 2 possible children. The resulting tree from the above example is as follows:

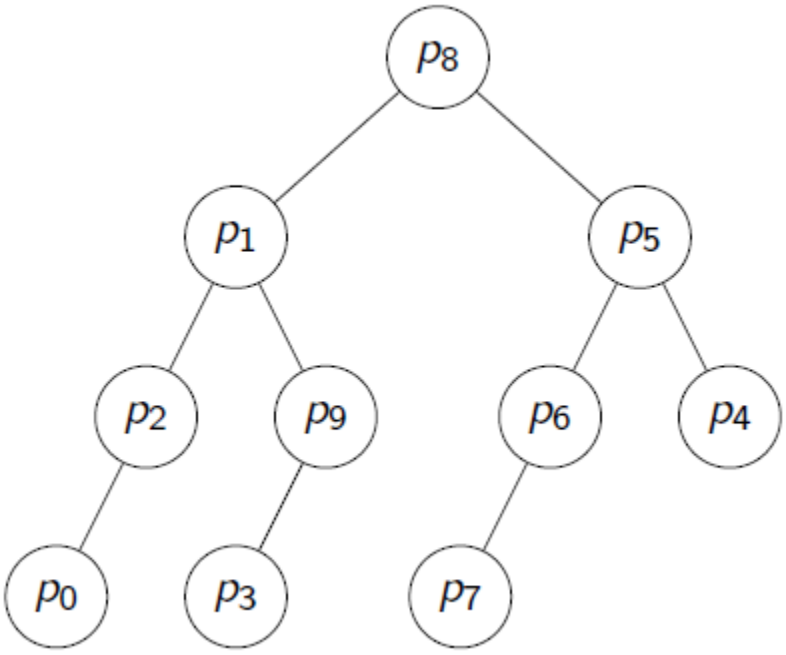

Figure 7 K-D tree

As before, the sample C++ structure:

```
struct KDtree{
  double x, y, dx, dy;
  //x and y are the coordinates of the top right corner
  //dx and dy are the lengths of the cell
  Node node;
  KDtree *left, *right;
};
```

Figure 8 K-D tree C++ struct

2.0 Analysis

With the above samples of how quadtrees and k-d trees are built, we then focus on how this may apply to a given river. The river in this case is only a collection of points expressed as coordinates on a plane and as a line segment between points. With knowledge of the location of the points on the plane, we can then apply the two methods of BSP previously mentioned. Through the example of a river, we can determine the usefulness of the two methods. Sample section of the river that will be used for analysis:

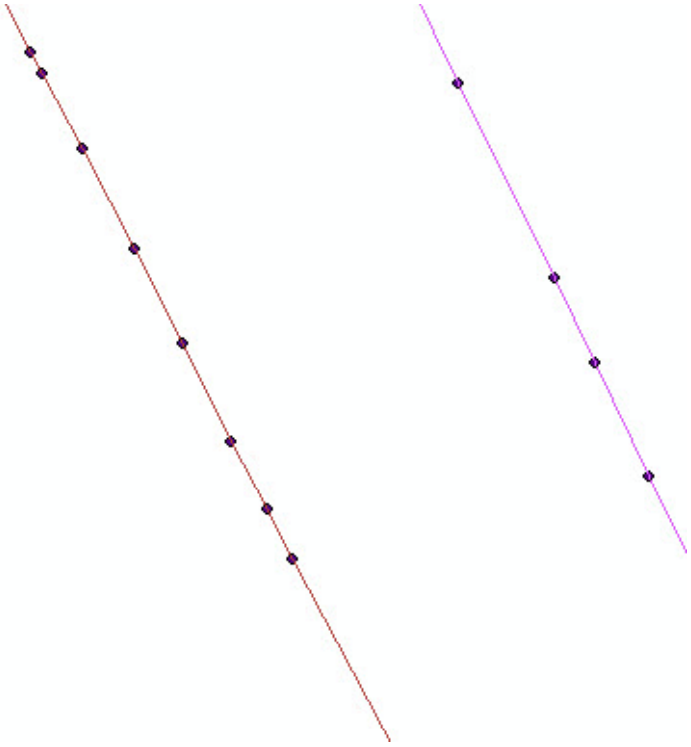

Figure 9 Sample river points

2.1 Partitioning a quad tree

The recursive C++ code for this tree is as follows:

```
void qtBuilder(Quadtree head, vector<Node> nodes, double x, double y, double dx, double dy){
  if(contains(head, node)){
    //if the given qt head has more than one point within its bounds, further cutting is required
    qtBuilder(head->ne, nodes, x, y, dx/2, dy/2);
    qtBuilder(head->nw, nodes, x - dx/2, y, dx/2, dy/2);
    qtBuilder(head->se, nodes, x, y - dy/2, dx/2, dy/2);
    qtBuilder(head->sw, nodes, x - dx/2, y - dy/2, dx/2, dy/2);
  }else{
    //otherwise there is either one node or no nodes in that cell
    putNode(head, nodes);
  }
}
```

Figure 10 Quad tree builder

We can apply the quadtree way of partitioning:

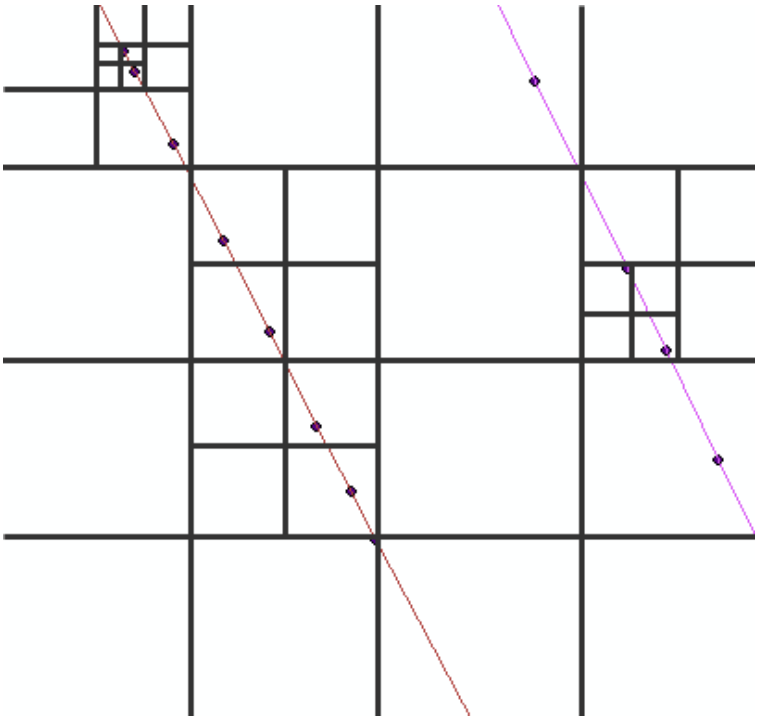

Figure 11 Quad tree partition of the river

This will result in a tree that looks like this:

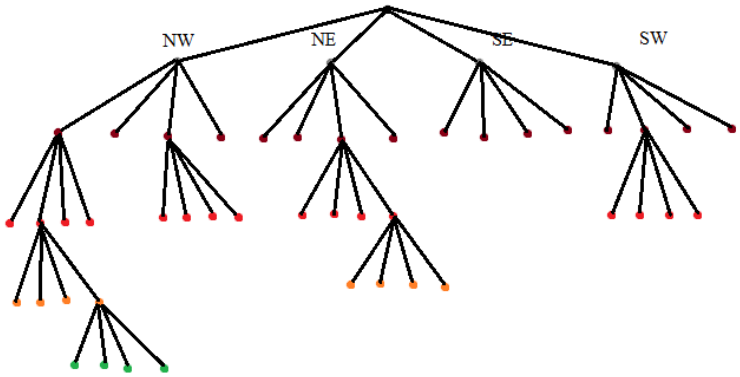

Figure 12 Quad tree structure representation

2.2 Partitioning a K-D tree

The recursive C++ code that builds the tree:

```
void kdbuilder(KDtree head, vector<Node> nodes, int split){

  if(!nodes.empty()){
    //since we need to alternate between splitting on the x and y axis,
    //there needs to be a sort trigger (pseudocode)
    if split == 0
      sortOnX(nodes);
      split = 1;
    else
      sortOnY(nodes);
      split = 0;

    //then take the middle node
    Node mid = nodes.at(nodes.size()/2);

    //and make that the head
    head = mid;

    //split the vector into 2
    vector<Node> left (nodes.begin(), nodes.at(nodes.size()/2));
    vector<Node> right (nodes.at(nodes.size()/2), nodes.end());

    kdbuilder(head->left, left, split);
    kdbuilder(head->right, right, split);
  }
}
```

Figure 13 K-D tree C++ builder

Using the same segment, we visualize the final partition based on the k-d tree:

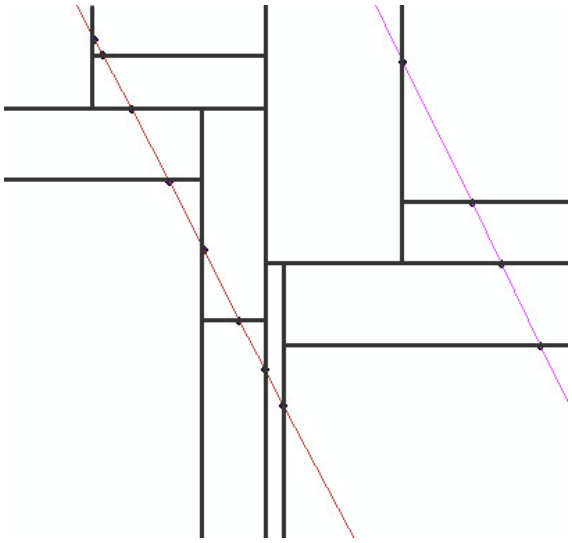

Figure 14 K-D partition of the river

The k-d tree then looks like this:

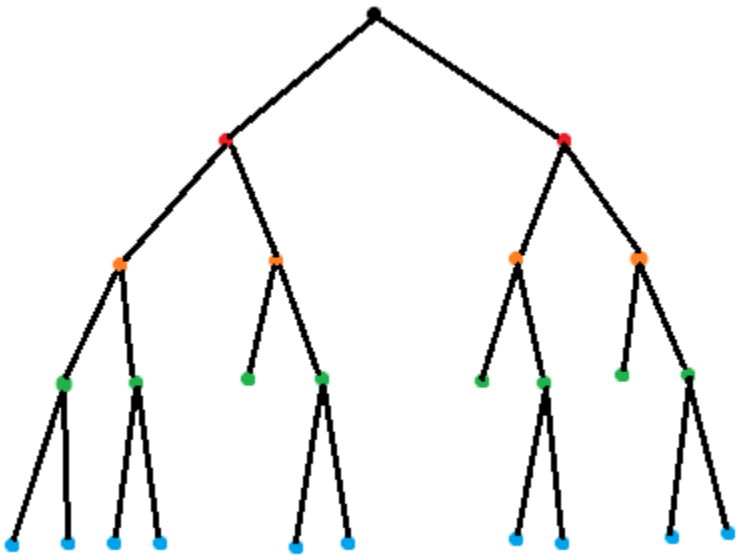

Figure 15 K-D tree structure

2.3 Quadtree or K-D tree

Looking at the two partitions that are created, it is evident visually what the differences are. For the k-d tree method, the partitions are not as uniform as the quadtree partitioning method. This difference creates different applications for the uses of the two data structures.

The uniformity of partitions in the quadtree method can be made analogous to first learning about solving the area under a function. Rectangles are created under the curve to approximate the total area. Increasing the number of rectangles gives a better approximation. Similarly, the more points there are along the river, the more partitions there will be. For the case of investigating wells and water pumping on a nearby river, if parts of a well are contained within some cells, information can be collected together from those adjacent cells. Further information and effects can be extrapolated from this collected data.

For the partitions in the k-d tree method, they are not uniform. The partitions vary in size and thus for the example above concerning water wells, the data extrapolated may not be very accurate. Depending on the size of the well, there could be more than one well in a given cell. Or perhaps one well protrudes very little into a cell that is larger in proportion.

With the disproportionate sizes of each cell in a k-d tree, it is difficult to determine what types of information to store within each node. The number of partitions that can be made is restricted upon the number of points. Compared with a quadtree partition, without points more partitions can be made to even further divide up the area closer to the river. One application of this is called mesh generation (Berkley, 1996). Starting with a big plot of space with a river, there are white spaced areas that do not require detailing. However as the area contains more river portions, there can be increased partitioning to create more detailed mesh.

However this is not to say that the k-d tree method of partitioning is generally bad. For certain other purposes it can be rather useful. An example can be found within collision detection (Berkley, 1996). One application of using k-d trees is the ability to find the nearest neighbour. Through utilising this nearest neighbour algorithm, potential collisions can be detected. This type of analysis works better when the partition space contains objects instead of points. Thus it would be possible to divide the objects into pieces. Nodes that contain pieces of a certain object can then detect the neighbours. If for example that the river, instead of being a series of points and line segments, was represented as a polygon, the use of k-d partitioning can help, for example, detect hazardous waste spills that may eventually come into contact with the river.

3.0 Conclusion

As evident from the river partitions, in this given case it is better to use a quadtree approach to partition the space. Also given that the partitioning is more versatile to subdivide each cell may be particularly useful for creating a finer mesh. Especially in this case where the given river is created by points and line segments, the even grids are easier to use and analyze. Another use for quadtrees in a similar situation is if the river was a polygon shape (footnote). Cells can be partitioned based on a percentage of the river contained within it.

This is not to say that the k-d tree method of partitioning is bad. However within this context it would serve a different purpose and thus would not be a very useful way of subdividing the space. But under a different circumstance, such as collision detection, it may very well serve as a better method.

References

"CS267: Lecture 24, Apr 11 1996." *CS267: Notes for Lecture 24, Apr 11 1996*. N.p., n.d. Web. 30 Apr. 2014. <http://www.cs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html>.

Golodetz, Stuart. "Divide and Conquer: Partition Trees and Their Uses." *ACCU :: Divide and Conquer: Partition Trees and Their Uses*. N.p., n.d. Web. 30 Apr. 2014. <http://accu.org/index.php/journals/1506>.