Project Report:Collision Detection using Quadtree

Meng Zhou
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
MengZhou@cmail.carleton.ca

April 22, 2014

1 Introduction

A quadtree is a tree data structure in which each internal node has exactly four children. Quadtrees are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions[3]. In games and data science, quadtrees are used to finding nearby objects, organize data for fast visit and depth sorting.

This project will study quadtree related algorithms such as how to build, travel and find neighbour of quadtree. Additionally, this project will implement quadtree algorithms and use it to do collision detection. However, if apply classic quadtree to collision detection, the efficiency is not satisfying due to many cross-sides objects. Inspired by Hanan Samet and their demo on the website[2], this project will implement a mutated quadtree called ”loose quadtree” which will be more efficiency for collision detection. Also a compare will be made between traditional quadtree and loose quadtree in collision detection of 2D objects.

2 Implementation Details

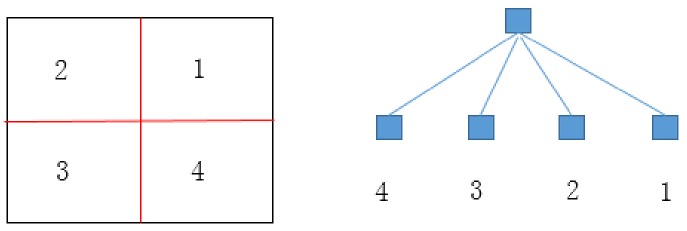

Figure 1: Space partition using quad tree

The idea of using quad tree for collision detection is popular in video games. In a scene with hundred or thousand of objects, calculating whether each pair of them is collided is brute-force. By recursively split the space into four parts like figure 1, we can get the correct result by only compare objects in the same branch. The initial algorithm for construct quad

Algorithm 1 Quad tree construction

```
function BUILDQTREE(in: objectlist:list out: scenetree:qtree)
    Calculate bounding box for object in objectlist
    for each object do
        Travel the tree and find the insert node for object
        Insert object into node
        if Count of object in node exceed THRESHOLD then
            SPLITNODE(node)
        end if
    end for
end function
function SPLITNODE(in: node)
    Create four children
    for each object in node do
        if Bounding box of object does not intersect with split axis then
            Insert object into corresponding child
            Delete object from node
        end if
    end for
end function
```

tree is inserting object into the tree one by one and maintain good property of quad tree. The construction algorithm is listed in algorithm 1.

There are two kinds of nodes in the quad tree, leaf node and mid-node. As the splitting function described, if the bounding box of the object intersect splitting axis, that means the object cannot fit into any of the children of the current node. So this object belongs to the mid-node. We intentionally point this out because this is important for the collision detection algorithm.

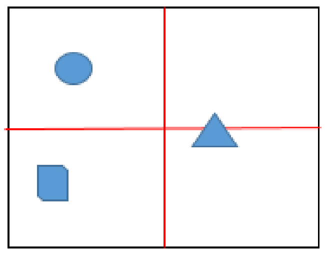

Figure 2: Different kinds of objects in quad tree

As we described before, with quad tree, we can do collision detection within leaf node in quad tree. However, there are objects that does not fit into any of the children. In order to do collision detection precisely, we store all the object references of all the ancestors of a leaf node when travelling quad tree for collision detection. As illustrated in algorithm 2, the travelling is done in a DFS manner, and the additional information for collision detection is stored in *midnodeobjlist*.

Commonly objects in a scene are moving, so the quad tree for the scene should be

Algorithm 2 Collision detection using quad tree

```
function COLLISIONDETECTION(in: scenetree:qtree out: collisionlist:list)
    midnodeobjlist:list ← new list
    Get root node of scenetree
    CDNODE(node)
end function
function CDNODE(in: node:qtreenode, midnodeobjlist:list out: collisionlist:list)
    if node is a leaf node then
        Compare each pair for each objects in node and add those collided into collisionlist
    else
        Add object in node into midnodeobjlist
        for each childnode of object do
            CDNODE(childnode)
        end for
        Remove object in node into midnodeobjlist
    end if
end function
```

updated for each frame. If the objects in a scene are not moving very fast, the tree structure does not change so much. We don't delete the tree and start over. Instead, we update the tree. We travel the tree and if an object does not fit into the node it belongs any more, we delete this object from it and insert it again.

Due to the addition expense brought by the objects that belongs to the mid-nodes, this project tries to eliminate this expense by using loose quad tree. The only difference between regular quad tree and normal quad tree is that when calculating if a bounding box belongs to a node, loose quad tree extend its border by p percent so that all the object can fit into leaf node. This project implement this algorithm and its faster than regular quad tree. However, some of the collisions are missing. In loose quad tree the lemma that we use for regular quad tree does not hold. We have to do collision detection for a leaf node and all its neighbours to make it precise. But in other applications such as finding the nearest object, loose quad tree is better than regular quadtree.

Algorithm 3 Update quad tree

```
function UPDATEQTREE(in: scenetree:qtree)
    Travel scenetree:
    for each object in node
    if Bounding box of object across border of node then
        Delete object from node
        Insert object into scenetree using algorithm 1
    end if
    end for
end function
```

3 Experiment Results

This porject visualize quad tree and collision using OpenGL. The running interface is shown by figure 3.

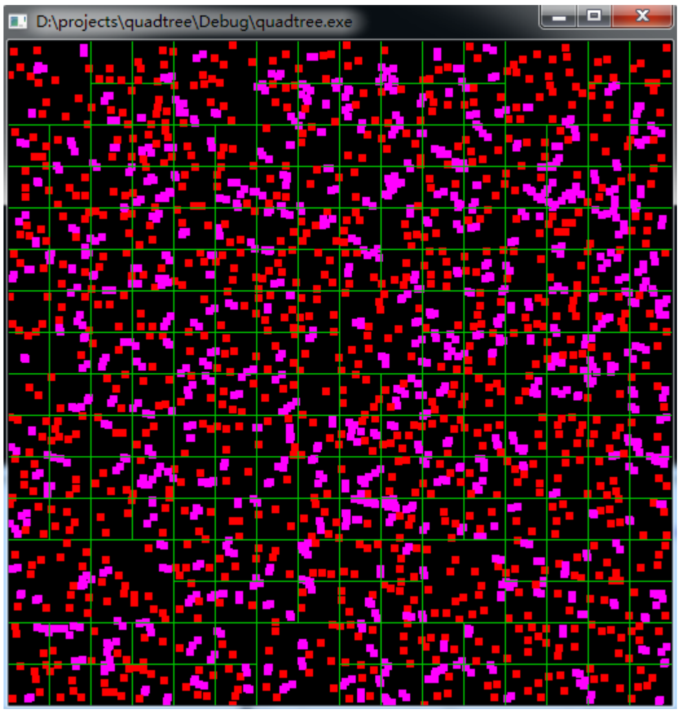

Figure 3: GUI of our project

Source code package and report can be found on http://www.dreamchou.com/course/comp5008

References

[1] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[2] Hanan Samet, Jagan Sankaranarayanan, and Michael Auerbach. Indexing methods for moving object databases: Games and other applications. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 169–180, New York, NY, USA, 2013. ACM.

[3] Wikipedia. Plagiarism — Wikipedia, the free encyclopedia, 2014. [Online; accessed 20-Feb-2014].