Indexing of 3D Terrain Space for Predicting Collisions with Moving Objects

Wanchun Wu° Youngduk Seo Bonghee Hong
{mcoh, ydseo, bhhong}@pusan.ac.kr
Dept. of Computer Engineering, Pusan National University

Abstract

In this paper, to find probable collision positions between moving object and terrain in 3D space efficiently, we use a model, similar to Ray Tracing, which finds the triangles intersected by a directed line segment from a large amount of triangles. We try to reduce dead space as much as possible to find candidate triangles intersected by a directed line segment than previous work's. A new modified octree, LBV-Octree(Least Bounding Voxel Octree), is proposed, and we have a ray tracing with it. In the experiment, ray tracing with LBV-Octree provides 5%~11% better performance than with classical octree.

Keywords: Triangular Irregular Network, Ray Tracing, octree, Least Bounding Voxel

1. Introduction

In 3D GIS (Geographic Information System), there are many moving objects, as cars, airplanes etc, and motionless objects, as buildings, mountains etc. There are maybe many collisions to happen between them; these collisions can be classified into two groups. One is collisions between moving objects, and the other is between moving objects and motionless objects. A good navigation system should find the coming collision fast to avoid these kinds of collisions happening whenever. In this paper, we only study collisions between moving objects and motionless objects, especially collisions between airplane and terrain around it in 3D space.

We can have a query "find the coming collision position of a flying airplane and mountains around it during next 1 minute". In the figure 1, t_0 is current time when airplane starts to predict, t_2 is predicted time, and t_1 is collision time. When $t_0 < t_1 \leq t_2$ the airplane will collide with mountain, then we need change the flying path, or do other operations before collision will happen. Otherwise it will be a disaster, if we can not predict the coming collision to happen at time t_1.

In general, there are two methods that predict the coming collision, hardware method and software method. First is hardware method which mainly means radars and their relevant equipments which

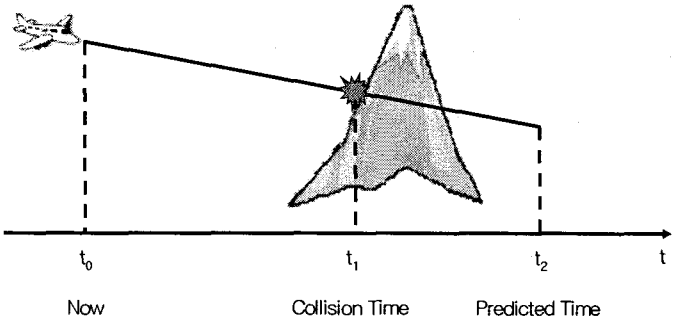

Figure 1. Relation between flying airplane and mountain

are usually set in the airplanes and control centers, but there are some defects with it, 1) it depends on the strength of the objects' signal, 2) it can hardly find the objects which are far and lay low, 3) it can also hardly find the objects if there are electromagnetic interferences. Second is software method, it can overcome these problems, but it has a prerequisite that there are databases which store the terrain information around the air plane. For the defects of radar, we concentrate on proposing a new software approach that can find the probable coming collision positions of the air plane and mountains around it efficiently, but not doing further refinement.

As usual, we assume moving object as point. We assume the object will not change its direction and velocity when it is moving from now to predicted

time, then we can easily get any future position to pass with simple computation $x = x_0 + v(t - t_0)$ (x_0: a vector of now position; v: a vector of velocity; x: a vector of predicted position; t_0: now time). And for collision only happens on the surface of terrain, only the surface information is enough. The surface of terrain is usually sampled as distributed random points (under an assumption that the terrain has no perpendicular slopes and overhangs), with these sampled points, the surface can be presented with TIN (Triangular Irregular Network). Then surface of 3D terrain is presented with composition of a large amount of triangles. So collision between a flying airplane and terrain becomes intersection between a directed line segment and a large amount of triangles. It is mostly similar to Ray Tracing, only a little difference from ours is that we use a directed line segment, but not a ray.

When the surface of the terrain is represented with a large amount of triangles, to reduce fault hit, reduction of dead space is needed. We propose a new modified octree, LBV-Octree (Least Bounding Voxel Octree). In a word, the goal is to reduce dead space in the voxel by giving a bounding box (LBV) to every voxel. And to find the candidate triangles more efficiently we distinguish the empty nodes which are marked with "interior" or "exterior". When the given directed line segment meets the "interior", it means some triangle has already intersected by the given line segment, so candidates selecting can be stopped.

The remainder of the paper is organized as follows. Section 2 presents a short overview on related work. Section 3 describes our modified octree, LBV-Octree, which can find intersection between a directed line segment and a large amount of triangles in 3D efficiently, and a ray tracing algorithm with LBV-Octree is given. Section 4 shows an experiment result of comparing LBV-Octree with octree and discusses it. Last section gives a conclusion and future work.

2. Related Work

Ray tracing (or ray shooting) is mostly widely used in Computer Graphics. Ray Tracing is one of the most realistic methods of generating computer images. It can generate shadows, reflections and refraction.

To perform a ray tracing, one way is spatial subdivision, it can be broadly categorized into bounding volume hierarchies and voxel based structures. Bounding volume hierarchies create a tree that the bounding volume of node encloses the bounding volumes of its children and the bounding of a leaf encloses primitives. It needs less disk space to save objects, but there are maybe many overlaps between MBBs. Voxel based structures are either grids or a hierarchy structure octree. With this method, there is no overlap between voxels, but more disk space is need to save objects duplicate. Recently there are many octree based ray tracing researches.

Classical octree is a tree structure to index three dimensions, it is based on quadtree which extends to three dimensions, and each voxel has either eight children or no children.

Another way to improve ray tracing is efficient algorithms. For octree traversing, it is also mainly categorized into two methods, top-down methods and bottom-up methods. With top-down methods, we start from the root voxel, and obtain its descendants intersected by the ray repeatedly. With bottom-up methods, the first voxel, which is intersected by the ray, is selected first, and then process neighbor finding.

In previous work [1], the objects are divided into many voxels with octree that divide space with spatial median, and an algorithm is given which uses a nine parameter set to compute which voxel is pierced first, then perform a neighbor finding repeatedly.

[2] introduces an octree-variant which divided space with the plane minimizing the number of ray-object intersection tests with cost model. Because it considered the objects' distribution, it gains better performance when the data are more skewed.

3. Ray Tracing with LBV-Octree

There are some important aspects to consider for reduction of number of disk I/O, 1) to reduce overlap between MBB of objects, 2) to reduce dead space.

When surface of the terrain is represent with a large amount of triangles. One important character with it is that all triangles composed of surface of terrain are adjacent each other, the other is that all mountains are cone-shaped under our assumption. When the space is divided, a serious problem is that there are many overlaps if we use bounding volume hierarchies, or there is much dead space if we use grid or octree. Both of results are that more disk I/O is needed when we perform a ray tracing.

To solve this problem, we use an index based on classic octree. Our main idea is that we give every voxel(non-terminal or terminal voxel) a bounding box(i.e. LBV) to reduce dead space, and divide the LBV into 8 children with its spatial median if necessary until all criterion are satisfied, and mark

empty voxels with "interior" or "exterior".

LBV (Least Bounding Voxel) is a minimal bounding box that encloses all internal parts of all objects which are in the voxel partly or wholly. In the figure 2, gray rectangle is an LBV, and it must not be larger than original voxel.

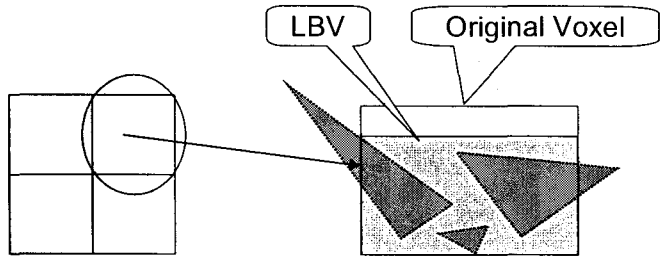

figure 2. Comparison with LBV and Original Voxel

3.1 LBV-Octree

LBV-Octree stands for Least Bounding Voxel Octree. First, we give a comparison between octree and LBV-Octree, see figure 3 and 4. The surface of terrain is represented with a polygon, and the cell's capacity is 4, then to obtain the candidate cells intersected by the line segment, we will have disk I/O 4 times and 2 times respectively.

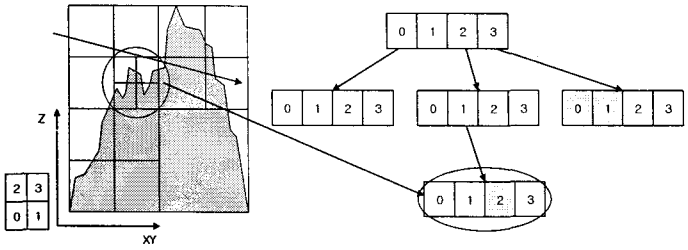

Figure 3. polygon divided with octree and its index

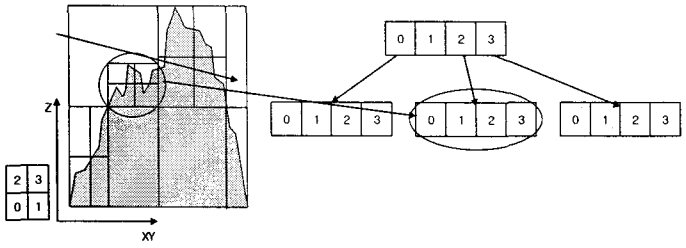

Figure 4. polygon divided with LBV-Octree and its index

A significant difference between classical octree and LBV-Octree is their division. In octree, the voxel will be divided into 8 children regardless of distribution of data and dead space. But in LBV-Octree, we divide LBV into 8 children. As a result, composition of all leaf nodes of LBV-Octree is more similar to object's surface than that of octree in shape.

Construction of an LBV-Octree is a little trivial, because we divide the LBV. We can not determine the LBV of any voxel before all triangles are inserted into the voxel. All triangles have to be inserted into each voxel that intersects triangles. Once all triangles have been inserted, we will determine the LBV of each voxel. If there are some voxels that needs to be divided, then these operations are repeatedly until no voxels need division further.

After the frame of index has been constructed, all triangles will be inserted.

For we only need to obtain the first intersected triangle, we modify the index some little. All empty nodes are marked with different states. One is "interior" which shows inside of the object, and the other is "exterior" which shows outside of the object. Steps of construction of LBV-Octree are:

step 1. insert all triangles to leaf nodes, only save the LBV and count of triangles inserted into each leaf node

step 2. judge whether some leaf nodes need to be divided, if so then divide them

step 3. clear count in the leaf nodes

step 4. if there are leaf nodes that have been divided then go to step 1

step 5. insert all triangles to leaf nodes

step 6. reset the LBVs of the voxel entries, and set empty voxel entries with "interior" or "exterior".

3.2 Ray Tracing

Algorithm of Ray tracing with LBV-Octree is partly same as that of previous ray tracing. We present an algorithm satisfied with our LBV-Octree. The steps of ray tracing are:

step 1. clear queue, and push the root node into stack

step 2. if the stack is empty then go to step 4, popup the uppermost one, if it is "interior" then go to step 4, otherwise calculate its sub-nodes which the LS will intersect and sort them by time ascending

step 3. push nodes that are leaf nodes into queue until meets non-leaf node, and push left nodes into stack on the contrary sequence, go to step 2

step 4. elements in the queue are the final candidates.

4. Experiment

Both classical octree and our proposed LBV-Octree have been implemented in C, and simulated data of

surface of terrain is obtained like that in the figure 5. The range of the terrain is (0, 20000) × (0, 10000) × (0, 1000), and we sampled 5 groups of different points, and obtained triangles created with Delaunay Triangulation with the sampled points, the sampled points are 5000, 10000, 25000, 50000, 100000, and triangles are 9943, 19939, 49902, 99883, 199866 respectively.

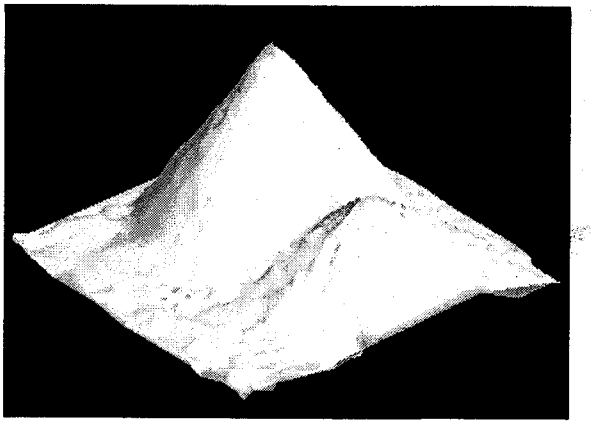

Figure 5. Surface of simulated mountain

For simplicity of implement, an LBV is determine with the MBB of triangles and voxel, but not triangle itself and voxel, so LBV in our experiment is some larger than that is determined with our idea.

We have compared classical octree with LBV-Octree in finding the probable intersected triangles by a given directed line segment. With different data, we obtained similar result of candidate triangles with both index, but got better result of disk I/O with latter than with former. Because we reduce some dead space in LBV-Octree, candidate leaf nodes of LBV-Octree are fewer than that of octree, i.e. the former needs fewer disk I/O than the latter, see figure 6. And for a leaf node in LBV-Octree contains more triangles than that in Octree, total candidate triangles of LBV-Octree are similar equal to that of octree, see figure 7.

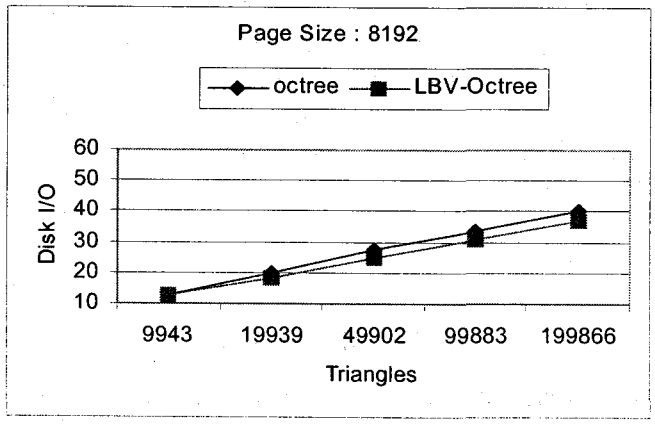

Figure 6. Disk I/O of octree and LBV-Octree

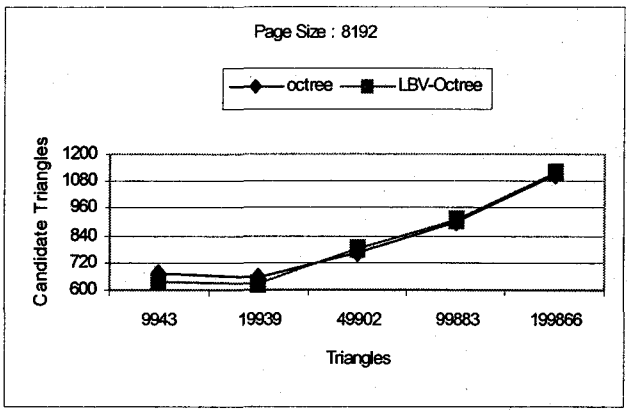

Figure 7. Candidate Triangles of octree and LBV-Octree

From other more experiments with different page size, we also got the same result. A conclusion is that LBV-Octree obtains better performance than octree with the same disk page size in ray tracing when there are a large amount of triangles.

5. Conclusion and Future Work

In this paper we propose a new modified octree, Least Bounding Voxel Octree, which is given each voxel an LBV. We divide LBV if it needs division, but not divide original voxel. We reduce disk I/O by reducing dead space, and we mark empty nodes with "interior" or "exterior", and find candidates faster than previous work's. In experiments we gain 5% ~ 11% better performances than with classical octree in finding first intersection between a directed line segment and a large amount of triangles.

It needs to take a long time to construct an LBV-Octree than octree, but the motionless objects seldom change their position or their shape, so we can construct it only once. It will not be a serious problem in ray tracing.

We will do more experiments, comparing octree with LBV-Octree. And we will improve the construction algorithm of LBV-Octree in the future.

Reference

[1] J. Revelles, C. Urena, M. Lastra. *An Efficient Parametric Algorithm for Octree Traversal.* WSCG'2000 conference, pp. 212-219, 2000

[2] Kyu-Young Whang, Ju-Won Song, Ji-Woong Chang, Ji-Yun Kim, Wan-Sup Cho, Chong-Mok Park, Il-Yeol Song, *Octree-R: An Adaptive Octree for Efficient Ray Tracing,* IEEE Transactions on Visualization and Computer Graphics 1(4): pp. 343-349, 1995