JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 7, NO. 12, DECEMBER 2004(pp. 1729-1736)

An Efficient Collision Detection in the Dynamic Spatial Subdivisions for an MMORPG Engine

Sung-ug Lee[†], Kyung-hwan Park[††]

ABSTRACT

This paper proposes an efficient collision detection method in the dynamic spatial subdivisions for the MMORPG engine which requires realtime interactions. An octree is a suitable structure for static scenes or terrain processing. An octree spatial subdivision enhances rendering speed of scenes. Current spatial subdivisions tend to be highly optimized for efficient traversal, but are difficult to update quickly for a changing geometry. When an object moves to the outside extent for the spatial subdivisions, the acceleration structure would normally have to be rebuilt. The OSP based on a tree is used to divide dynamically wide outside which is the subject of 3D MMORPG. TBV does not reconstruct all tree nodes of OSP and has reduced rebuilding times by TBV information of a target node. A collision detection is restricted to those objects contained in the visibility range of sight by using the information established in TBV. We applied the HBV and ray tracing for an efficient collision detection.

Keywords: Octree, collision detection, spatial subdivisions, ray tracing, mmorpg

1. INTRODUCTION

A game is composed of a game engine and various game contents. The game engine includes game logic, level format, audio processing, event handler and input processing. Fig. 1 shows a general structure of the game design.

A speed is important in a game which should manage wide outside regions like 3D MMORPG (Massively Multi-player Online Role Playing Game). It is scene of subdivision about the large terrain that uses OSP. An octree is a suitable structure static scene or terrain processing. Current spatial subdivisions tend to be highly optimized for efficient traversal, but are difficult to update quickly for changing geometry. When an object moves outside the exit of the spatial subdivision, the acceleration structure would normally have to be rebuilt. Therefore, dynamic environment requires rapid updates to acceleration structure.

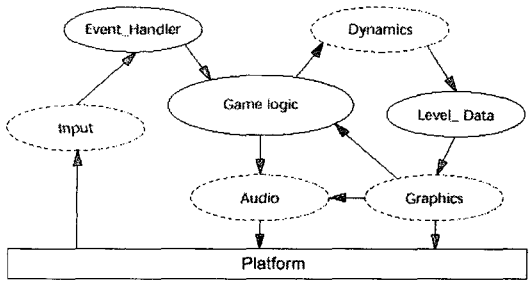


Fig. 1. Structure of game design.

In this paper we seek to an collision detection method in the spatial subdivisions for an MMORPG engine.

TBV does not reconstruct all tree nodes of OSP for dynamic process and has reduced rebuilding times by changing the TBV information of a working target node. A collision detection is restricted to those objects contained in the visibility

※ Corresponding Author : Sung-ug Lee, Address : (604-714) 840, Hadan2-dong, Saha-gu, Busan, Korea, TEL : +82-51-200-7776, FAX : +82-51-200-7783, E-mail : sunlee@smail.donga.ac.kr
Receipt date : July 20, 2004, Approval date : Oct. 5, 2004
[†] Dept. of Computer Engr. Dong-A Univ.
[††] Dept. of Computer Engr. Dong-A Univ.
(E-mail : khpark@daunet.donga.ac.kr)
※ This paper was supported by the Dong-A University matching fund for IT facilities supporting program of Ministry of Information and Communication, in 2002.

range of sight by using the information established in TBV. Also, we apply the HBV and ray tracing for a collision detection.

2. DYNAMIC SCENE PROCESSING

2.1 Spatial Subdivision

The game design such as 3D MMORPG requires various theories of several technologies. The speed is an important point in a game engine design. Dynamic scene means that objects in the model are not static but can move just as the camera moves. This means that the model must be preprocessed to obtain a hierarchical representation. If an object in the model moves, the whole structure should be rebuilt again. An octree is a data structure to represent objects in the 3D environment. An octree is a suitable structure for static scenes or terrain processing. It is used in a data structure to store transparency benefits for interactive object and attach catalog of each individual appending various environment such as game.

However, when an object moves to the outside extent of the spatial subdivision, the acceleration structure would normally have to be rebuilt. As this is too expensive to perform repeatedly, we seek to logically replicate the grid over space. If an object exceeds the bounds of the grid, the object wraps around before reinsertion. Then, ray traversal wraps around the grid when a boundary is reached. In order to provide a stopping criterion for ray traversal, a logical bounding box is maintained by containing all objects and including the ones that have crossed the original perimeter.

As this scheme does not require grid recomputation whenever an object moves far away, the cost maintaining the spatial subdivision will be substantially lower. On the other hand, because rays now may have to wrap around, more voxels may have to be traversed per ray which will slightly increase ray traversal time. Specifically, the length of the object diagonal divides the length of the grid diagonal. The result determines the grid level. Hierarchical grid traversal is effective grid traversal following modifications[3,6,13].

2.2 HBV Method

A bounding volume simply represents the volume of real object. The most suitable geometric objects for bounding volumes are spheres and boxes. Fig. 2 shows Hierarchical bounding volume. It uses the following basic function to evaluate the performance of a general algorithm working on hierarchical data structures:

T = Nv*Cv+Np*Cp where
T : total time spent on processing
Nv : the number of bounding volume test
Cv : the cost of single bounding volume test
Np : the number of objects processed face by face
Cp : the cost of processing a single object

The choice of bounding volume type is affected by the following mutually contradicting factors:

- Bounding volume tests should be as accurate as possible to lower Np(and hence the second term Np*Cp) and Nv. Usually, the better bounding volume fits the scene object's geometry, the more accurate the test is.
- A single bounding volume test must be as fast as possible.[1,7]

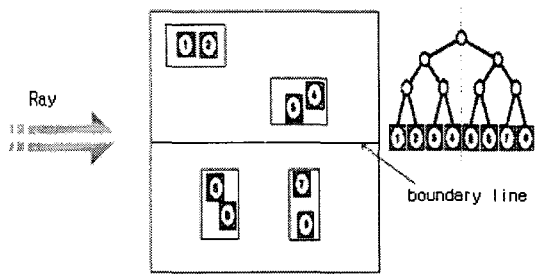


Fig. 2. Hierarchical bounding volume.

2.3 OSP's Spatial Subdivision Method

An octree is a suitable structure for static scenes or terrain processing. The first octree node is the root cell which is an array of eight contiguous

elements. Each of these elements can point to another block of eight contiguous elements, where each one can point to another block of eight contiguous elements and so forth until a certain maximum number of levels are reached. The last level is the leaf level where the leaf elements or vowels are stored[8]. One problem with octree is that as objects are inserted and deleted, the tree structure could become arbitrarily inefficient unless some sort of prebalancing step is performed. Voxel based structures are either grids or can be hierarchical in nature, such as octree[2].

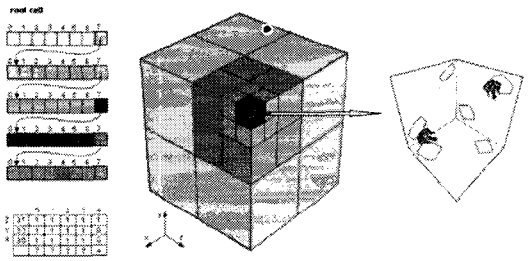


Fig. 3. OSP node structure.

The cost of building a spatial subdivision tends to be O(n) in the number of objects.

Fig. 3 shows OSP node structure. An OSP is a hierarchical method for spatial subdivisions. The processing concept of octree needs adapt to interactive environment. An OSP is used by spatial subdivisions method. It minimizes the processing of spatial subdivisions in 3D environment. Fig. 4 shows OSP class code.

```
class octree {
        int iType ;
        int Depth ;
        FACE3D * flist ;
        VECTOR3D min ;
        VECTOR3D max ;
        VECTOR3D vCenter ;
} ;
 class octree_node : public Limit_octree {
        limit_octree * child[8] ;
        } ;
 class octree_leaf : public limit_octree {
        polygon_list polygons ;
        object_list objects ;
        ...
        } ;
```

Fig. 4. OSP class code.

2.4 TBV

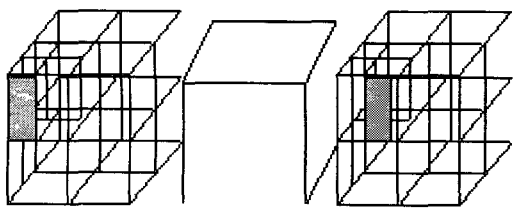
Bounding volume hierarchy is suitable to dynamic object scenes. If the scenes are dynamic, an object movement should be contained, and then additional time is needed to update the model to reflect the motions of this object. The data structure used in an occlusion method is updated to reflect the possible positions of the object. To avoid updating the structure for every dynamic object at each frame, a TBV is created for each occluded dynamic object, using some known constraints on the object's motion. The TBV is inserted into the structure instead of the object.

The data structure needed is as follows.

① Priority queue for TBVs
② List of moving visible objects
③ Time stamps for all objects
④ Data structure for the model

Following steps are executed for each frame: As Fig. 5 shows, OSP has reconstructed all tree node of OSP for dynamic process. But as Fig. 6 shows, TBV has not reconstructed all tree nodes of OSP for dynamic process, but it is able to reduce rebuilding times by changing only the TBV information for the working target node.

1) If there are any expiring TBVs in the TBV priority queue, they are removed from the queue and their objects are added to the list of moving visible objects.



(a) (b) (c)

Fig. 5. OSP nodes might be needlessly deleted and recreated during update (a) A vertical view of an initial model (two primitives) (b) after the deletion of dynamic primitive (c) after the insertion of dynamic primitive at new configuration.

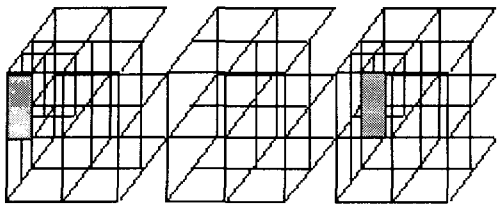


Fig. 6. Using OSP eliminates unnecessary tree updates. Step (a)(b)(c) OSF for spatial subdivisions.

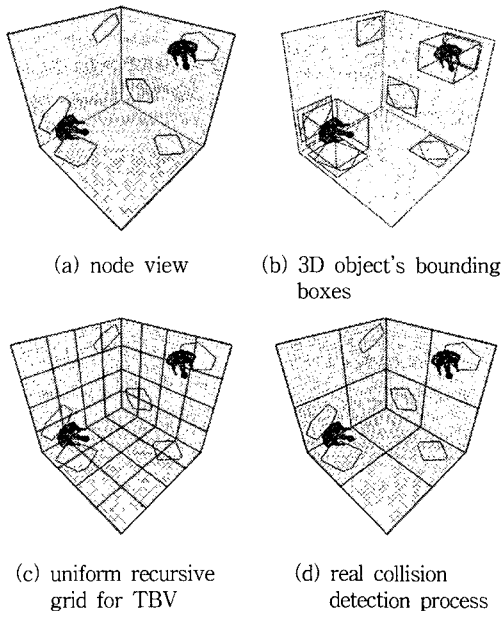
2) All objects in the list of moving visible objects are calculated for their new positions, and these new positions are updated in the data structure that represents the model.

3) The visibility algorithm is executed on the data structure as usual. If one encounters a node that contains a TBV, this TBV is removed from the TBV priority queue and its objects are added to the list of visible objects. Also its new position is calculated. For each visible object in the node, stamp information should be updated in current time.

4) When the algorithm ends, the list of visible objects is updated so that each object that has an old time stamp is removed from the list. Since these objects are now becoming invisible, they are associated with a new TBV that is inserted into the TBV priority queue[5,13].

3. DYNAMIC ACCELERATION STRUCTURE FOR COLLISION DETECTION

Interactive ray tracing has become a reality, allowing exploration of scenes rendered with higher quality shading than with traditional interactive rendering algorithms. One of the problems with interactive ray tracing is that previous implementations only dealt with static scenes or scenes with a small number of specially handled moving objects. Fig. 7 shows, An each step a processing overview. Dynamic environments required that rapid update to insert is usually accomplished by mapping the axis aligned bounding box of an object



(a) node view (b) 3D object's bounding boxes

(c) uniform recursive grid for TBV (d) real collision detection process

Fig. 7. An each step a processing overview.

to the voxels of the grid. This effectively limits the usefulness of interactive ray tracing to applications that allow changes in camera position[1,8].

TBVs are used in conjunction with the hierarchical structures that represent the moving objects. A moving object is ignored until either of the events happens: ① its TBV expires, which means that the TBV no longer is guaranteed to contain the object or ② visibility algorithm finds out that the TBV that represents the object is visible again, thus implying that the object might be visible too.

```
pocedure updatee(Octree, primitive, new_config);
begin
        v←primitive.node;
        dissociate(v,primitive);
        primitive,config←new_config;
while v.parent can be collapsed and not
   (v contains primitive) do begin
   v←v.parent;
   collapse(v);
end;
while not (v contains primitive) do
   v←v.parent;
   insert(v,primitive);
end
```

Fig. 8. The OSP update algorithm.

The large models can take a long time to render, even when hardware support such as Z-buffering is available. If the scenes are dynamic, containing moving objects, additional time is needed to update the model to reflect these objects' motions. This makes it hard to attain the goal of interactivity. Fig.9 shows, Occlusion culling is one of the key techniques for output-sensitive rendering.

The data structure used by an occlusion culling method is updated to reflect the object's possible positions. To avoid updating the structure for every dynamic object at each frame, a TBV is created for each occluded dynamic object, using some known constraints on the object's motion.

```
void CullOctreee(Octree **pptrNodes , ViewFrustum&
viewFrustum,) {
//Check if the current node is in our frustum
   if(!viewFrustum.isCubeelnFrustum(m_BBoxVetrices)){
      m_Lift=0;
      return;
    }
    if(m_bSubDivided) {
 //Recurse to the bottom of these nodes and draw and
 the end node's vertices
      CullOctree(pptrNodes,viewFrustum);
       ... }
     else
       { pptrNodes[q_NumVisibleNodes]=this;
         g_NumVisibleNodes++;
     }
 }
```

Fig. 9. The data structure used by an culling method.

In either case, the object must be considered again. Spatial subdivisions allow efficient ray traversal as well as rapid insertion and deletion for scenes where the extent of the scene grows over time. In theory, the tree structure allows O(n*log*n) insertion and deletion time which may be fast enough.

The ray will detect all parents node until the root node is reached. This is all levels in the hierarchy that may occupy the same space as the currently traversed leaf node. If an intersection is found within the space of the leaf node, then traversal is finished. If not, the next leaf node is selected and the process is repeated. This traversal scheme is same parent nodes that may be repeatedly traversed for the same ray.

Here is how sphere plane collision works: If a distance that the center is from the plane is less than the radius of the sphere, it must be intersecting the plane. We take the absolute value of the distance when the center of the sphere goes behind the polygon. It used the distance formula to find the distance to the center point of the sphere which is from the polygon's plane.

```
Ax + By + Cz + d = 0 with ABC = Normal, XYZ = Point
distance = (vNormal.x * vCenter.x + vNormal.y *
            vCenter.y + vNormal.z * vCenter.z + d);
```

Now we query the information just gathered. The distance turns into negative numbers (with 0 being that the center is exactly on the plane). If the absolute value of the distance just found is less than the radius, the sphere intersects the plane.

```
if(fabs(distance) < radius)
        return INTERSECTS;
```

else, if the distance is greater than or equal to the radius, the sphere is completely in FRONT of the plane.

```
else if(distance >= radius)
        return FRONT;
```

If the sphere isn't intersecting or in FRONT of the plane, it must be BEHIND

```
        return BEHIND;
 }
```

Real collision detection is performed by three steps: In the first step, we set up comparative simple formed bounding box enclosing itself and analyze collision between ray and bounding box, then check on a collision possibility through an impact or not with ray.

Second, next operation inspects a clash between ray and bounding box. Lastly, third step searches

a clear collision or not using collision detection method in case only where an impact occurs between ray and bounding box. The advantage of this processing is that the method can detect a collision at early time without doing all object collision examination.

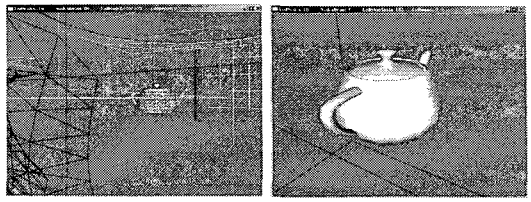
However, to make the traversal efficient, the tree is augmented with extra data, and is occasionally flattened into an array representation which enables to fast traversal but either insertion or deletion incurs a nontrivial cost.

```
void hbv_ftb (const Point & point) {
int near = (point dot node.plane_normal >= 0.0) ;
 if (child[near]) child[near] → hbv_ftb(point) ;
 //for each polygon facing the near node in this plane
 {
        collision_detection() ;
          }
 if (child[near^1]) child[near^1] → planar_ftb (point) ;
 }
```

Fig. 10. collision detection code

The larger problem with spatial subdivisions is that the grid structure is built within volume bounds that are fixed before construction. It can use scene of graph information getting the location of the 3D object. It suggests the difference of the method by hierarchical bounding box tree and BSP(Binary Space Partitioning Tree). Also BSP does not divide equally spatial subdivisions.

Spatial subdivisions become unit that forms bounding pair. The most important requirements for ray tracing are fast ray traversal and adaptation in unevenly distributed data. We construct a hierarchical uniform of recursive grid. Using a value of divided space the processing builded a TBV then it constructs the space as tile map. A title structure in 2D game is applied equally in 3D game. This is broadly categorized into HBV and voxel based structures. Time complexity is as follow. If 3D objects are n items and a number of triangle formating a side of each 3D object is average m, the time complexity necessary for the



(a) The search of collision possible node (b) OSP collision detection node

Fig. 11. Collision detection performed

collision detection is $O(n^2n^2)$ in case of not using bounding box but it is $O(n^2+a)$ in time of applying bounding box. Moreover, if HBV is used for collision, all bound box is not need to detect. In conclusion, the result may be required fast enough when time complexity is $O(nlogn)$.

Fig. 11 shows each step processing. The work is the functionality of interactive ray tracing to include applications where objects need to interactively manipulate.

Table 1. Comparision of the method

Algorithm	Method	Data Structure	Time Complexity
incremental tree-insertion	tree-insertion	tree	O(n*log*n)
k-DOP	top-down,	approximation, hill climbing	*log*kn
sweep-and-prune	bubble or insertion sorting	sweep-and-prune	O(n*log*n+k)
our method	tree-insertion, TBV	octree, queue, binary tree	O(n*log*n)

4. CONCLUSIONS

An acceleration structures used for ray tracing have been designed and optimized for an efficient traversal of static scenes. As it becomes feasible to do interactive ray tracing for moving objects, new requirement are posed upon the acceleration structures. Dynamic environment requires rapid updates to the acceleration structures.

We proposed an efficient collision detection in the spatial subdivisions for a game engine which

requires realtime interactions. The OSP based on tree to dynamically fast process for wide outside become a main target of 3D MMORPG. An octree is a suitable structure for static scenes or terrain processing. An octree spatial subdivisions enhance rendering speed for scenes. Current spatial subdivisions tend to be highly optimized for efficient traversal, but are difficult to update quickly for changing geometry. When an object moves to the outside extent of the spatial subdivisions the acceleration structure would normally have to be rebuilt. However a realistic processing is required in the environment of realtime dynamic game.

The results obtained in this study have shown that:

In view of processing forms

1) We have reduced a processing sphere divided a space with OSP to fast dynamically settle wide outside.

2) We have constructed the position information of each object using TBV.

3) We makes use ray tracing for collision detection by HBV.

In view of time complexity

1) A case of not using bounding box for collision detection is $O(n^2n^2)$

2) A case of using bounding box for collision detection is $O(n^2+\alpha)$

3) A case of using HBV for collision detection is $O(n\log n)$

In conclusion, the result has required fast enough when time complexity is $O(n\log n)$.

5. REFERENCES

[1] Sung ug Lee, "A Collision Detection from Division Space for Performance Improvement of MMORPG Game Engine," Korea Information Processing Society, Vol 10-B, No 5, pp. 567-574, August 2003.

[2] Sung-ug Lee, Kyung-hwan Park, "A Collision Detection Octree Partitioning Method using CLOD," Korea Information Processing Society, Proc. of the 16th KIPS FALL Conference, pp. 615-618, 2001.

[3] Batagelo Harlen Costa and Wu Shin-Ting, "Dynamic Scene Occlusion Culling using a Regular Grid," Proc. of the XV Brazilian Symposium on Computer Graphics and Image Processing, pp. 43-50, 2002.

[4] D. Cohen-Or, Y. Chrysanthou, C. T. Silva, and F. Durand. "A Survey of Visibility for Walk through Applications," SIGGRAPH 2001 Course Notes, August 2001.

[5] Eberly, David H., 3D Game Engine Design, Academic Press, 2001.

[6] H. C. Batagelo and S. T. Wu, "Dynamic Scene Occlusion Culling using Regular Grids", Technical report, State University of Campinas, Campinas, Brazil, December 2001.

[7] H. J. Haverkort, M. de Berg and J. Gudmundsson, "Box-Trees for Collision Checking in Industrial Installations," Proc. of the 18th ACM Symp. on Computational Geometry, pp. 53-62, 2002.

[8] J. Cohen, M. Lin, D. Manocha, K. Ponamgi, "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments," Proc. of the 1995 ACM International 3D Graphics Conference, pp. 189-196, 1995.

[8] M. Kelleghan, "Octree Partitioning Techniques," Game Developer Magazine, pp.30-33, 1997.

[9] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, "Computational Geometry: Algorithms and Applications," Springer-Verlag, Heidelberg, 1997.

[10] M. de Berg, M. de Groot and M. Overmars, "New Results on Binary Space Partitions in the Plane, Computational Geometry: Theory and Applications," pp. 317-333, 1997.

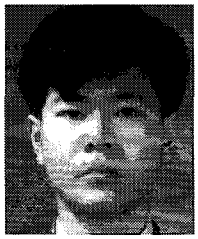
[11] M. de Berg. Ray Shooting, "Depth Orders and

Hidden Surface Removal," Lecture Notes in Computer Science 703, Springer-Verlag, Berlin, 1993.

[12] N. Greene, "Occlusion Culling with Optimized Hierarchical Z-Buffering," SIGGRAPH 2001 Course Notes, August 2001.

[13] O. Sudarsky and C. Gotsman, "Dynamic Scene Occlusion Culling," IEEE Trans. on Visualization and Computer Graphics, 5(1), pp. 217-223, 1999.

[14] S. Gottschalk, M. C. Lin, D. Manocha, "A Hierarchical Structure for Rapid Interference Detection," Proc of Siggraph 96, ACM Press, New York, pp., 171-180, 1996.



Sung-ug Lee

He received the BS degree in department of economics from Dong-A University, in 1989 and M.S degree in computer engineering from Dong-A University, in 1991. Since 2001, he has been a professor of computer engineering at Dong-A University. His research interests include Multimedia Systems, 3D Graphics and Game Distance Education.

Kyung-hwan Park

He received the BS degree in computer engineering from Kyungpook National University, in 1981 and M.S and Ph.D degree in computer engineering from Seoul National University, in 1983 and 1990. In 1998, he was a visiting scholar at University of California, Irvine. Since 1987, he has been a professor of computer engineering at Dong-A University. His research interests include Multimedia Systems, Electronic Commerce and Distance Education.