

Incremental 3D Collision Detection with Hierarchical Data Structures

Tsai-Yen Li and Jin-Shin Chen

Computer Science Department
National Chengchi University
Taipei, Taiwan, R.O.C.
Email: {li, s8213}@cs.nccu.edu.tw

VRST'98

November 4, 1998

Outline of the Talk

- **The collision detection problem**
- **Previous work**
- **Algorithms with hierarchical data structures**
 - Cost analysis and common characteristics
- **The proposed approach**
 - Maintaining a *Separation List*
- **Performance evaluation**
- **Conclusion and future work**

The Considered Problem: 3D Collision Detection

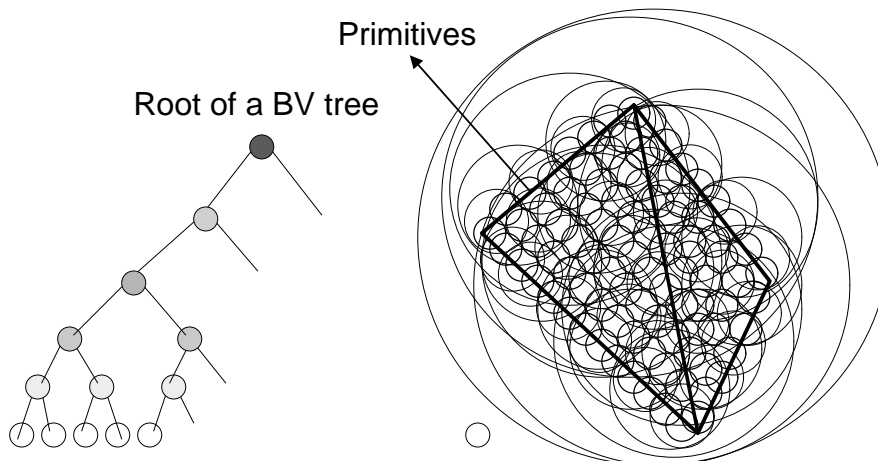
- **Determine interference between objects**
 - Special case of distance determination
- **Applications:**
 - VR and 3D Graphics
 - Robotics
 - CAD/CAM (e.g. Maintainability Study)
- **Key components of:**
 - 3D Interactive Graphics (VR)
 - Dynamic Simulation
 - Motion Planning

Previous Work

- **Tracking Closest Features between Polyhedra**
 - **Gilbert (1988) and Cameron (1997):** $O(n)$ and $O(1)$
 - **Lin (1993):** $O(1)$
 - **common problem:** convexity requirement
- **Bounding Volumes (BV) for Polygonal Facets**
 - **Sphere Trees:** Hubbard (1993), Quinlan (1994)
 - **OBB Trees:** Gottschalk, et al. (1996)
 - **DOP Trees:** Klosowski(1996), Zachmann (1998)
 - **Spherical-Shell Trees:** Krishnan, et al. (1998)
 - etc.

Hierarchical Bounding Volumes : An Example

- Sphere-tree built from bottom up



Basic Recursive Algorithm

```
Boolean CheckBVCollision(node  $N1$ , node  $N2$ )  
if both  $N1$  and  $N2$  are leaves then  
    return CheckPrimitiveCollision( $N1$ ,  $N2$ )  
elseif  $N2$  is larger than  $N1$  then  
    for each child  $N2[i]$  of  $N2$   
        if CheckBVCollision( $N1$ ,  $N2[i]$ ) then  
            return TRUE  
else  
    for each child  $N1[i]$  of  $N1$   
        if CheckBVCollision( $N1[i]$ ,  $N2$ ) then  
            return TRUE  
return FALSE
```

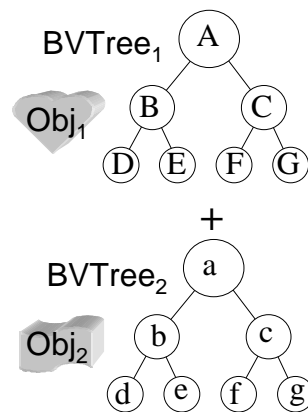
Cost Analysis of Algorithms with Hierarchical Data Structures

$$T = N_u \times C_u + N_v \times C_v + N_p \times C_p$$

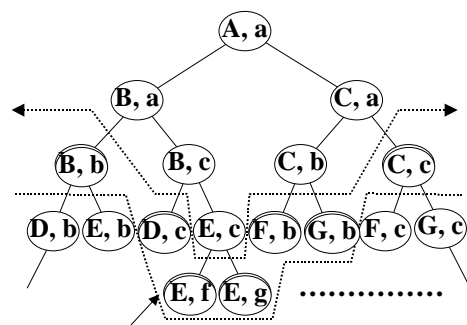
- T : total cost for an interference check
- N_u : no. of BV's updated
- C_u : cost of updating a BV
- N_v : no. of BV overlap tests
- C_v : cost of testing two BV's for overlaps
- N_p : no. of primitive pairs tested for interference
- C_p : cost of testing two primitives for interference

Typical Recursion Tree and Separation List

Bounding Volume Tree

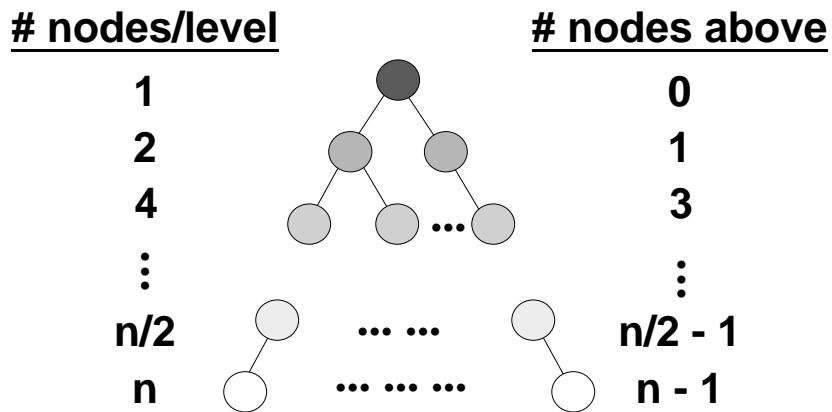


Recursion Tree



Separation list: (B,b), (D,c), (E,f), (E, g), (F, b), (G, b), (C, c)

Number of Nodes in a Recursion Tree



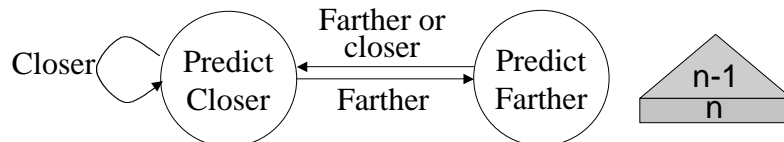
Case 1 (still):
 $2n - 1^{1/2}n = 1/2n$

Case 2 (down):
 $2n - n = n$

Case 3 (up):
 $n - 1^{3/4}n = -3/4n$

Predicting Motions of a Separation List

- **Update Strategy:**
 - Do not shrink the list. Rebuild the separation list when objects are predicted to be moving apart.
- **Prediction Strategy:**
 - **Getting Closer:** when size of the list grows.
 - **Getting Farther:** when the list stops growing.



Case 1 (still):
 $2n - n = n$

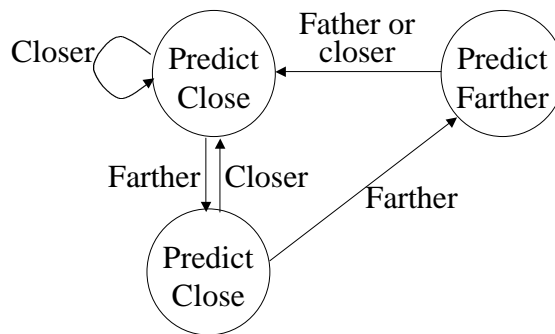
Case 2 (down):
 $2n - n = n$

Case 3 (up):
 $n - n = 0$

Deferring Rebuilding a Separation List

- **Modified Prediction Strategy:**
 - Deferring rebuilding the separation list by one run.
 - Possible situations of no growth after two runs:

(up, up),
(up, still),
(up, down),
(still, up),
(still, still)



Implementation and Experiment

- **Implemented a sphere-tree algorithm (Quinlan, 1994), part of a path planner.**
- **Written in C++.**
- **Runs on most UNIX machines.**
- **Data taken on a PC (K5, 133MHz) running Linux 2.0.7.**
- **Experiments:**
 - Nine bunnies with 500 polygons each moving randomly in a closed workspace.
 - **Increments:** 5 degrees for rotation and radius of the smallest sphere for translation.

Performance Evaluation

Run #	Original	<u>Using SL</u>		<u>Deferred SL</u>	
	time (sec.)	time (sec.)	speedup	time (sec.)	speedup
1	1507.9	1075.7	40%	759.2	99%
2	1744	1252.2	39%	1021.7	71%
3	1552.8	1077.7	44%	781.5	99%
4	1713	1177.3	46%	980.1	75%
5	1548.4	1080.5	43%	808.2	92%
6	1653.4	1168.3	42%	969.5	71%
7	1403	1021.2	37%	687.5	104%
8	1570.5	1120.2	40%	831.4	89%
9	1564.9	1163.1	35%	900.4	74%
10	1580.5	1139.2	39%	865.7	83%
Ave.	1583.8	1127.5	40%	860.5	84%

Conclusion

- Efficient collision detection algorithms are crucial for 3D/VR applications.
- Improved efficiency of a class of algorithms with hierarchical data structures.
- Captured spatial and temporal coherence by using a separation list.
- Proposed strategies for maintaining a separation list.

Future Work

- Needing more experiments on different geometry complexity and settings.
- Experimenting on more hierarchical data structures such as OBB trees (preliminary results).
- Obtaining quantitative relation between spatial coherence and performance improvement.
- Incorporating the collision detection module into a VRML browser.

Q & A