Collision Detection

Originally created by
Tomas Akenine-Möller
Updated by Ulf Assarsson
Department of Computer Engineering
Chalmers University of Technology

Introduction

- Without collision detection (CD), it is practically impossible to e.g., games, movie production tools (e.g., Toy Story)
- Because, without CD, we'll get "quantum effects" all the time
  - Objects will pass/slide through other objects
- So, CD is a way of increasing the level of realism
- Not a pure CG algorithm, but extremely important
  - And we have many building blocks in place already (spatial data structures, intersection testing)

What we'll treat today

- Three techniques:
- 1) Using ray tracing
  - Very simple
  - Not accurate
  - Very fast
  - Sometimes sufficient
- 2) Using bounding volume hierarchies
  - More complicated
  - More accurate
  - Slower
  - Can compute exact results
- 3) Efficient CD for several hundreds of objects

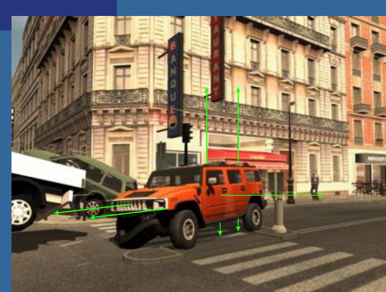
In general

- Three major parts
  - Collision detection
  - Collision determination
  - Collision response
- We'll deal with the two first
  - The third involves physically-based animation
- Use rays for simple applications
- Use BVHs to test two complex objects against each other
- But what if several hundreds of objects?

For many, many objects…

- Test BV of each object against BV of other object
- Works for small sets, but not very clever
- Reason…
- Assume moving n objects
- Gives: $\binom{n}{2}$ tests
- If m static objects, then: $nm+\binom{n}{2}$
- There are smarter ways: third topic of CD lecture

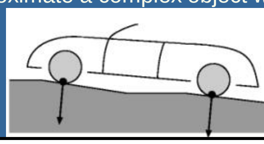
Example



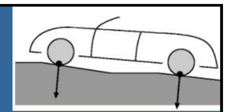
Midtown Madness 3, DICE

Collision detection with rays

- Imagine a car is driving on a road sloping upwards
- Could test all triangles of all wheels against road geometry
- For certain applications, we can approximate, and still get a good result
- Idea: approximate a complex object with a set of rays



CD with rays, cont'd



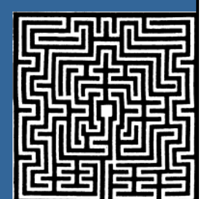
- Put a ray at each wheel
- Compute the closest intersection distance, t, between ray and road geometry
- If t=0, then car is on the road
- If t>0, then car is flying above road
- If t<0, then car is ploughing deep in the road
- Use values of t to compute a simple collision response

CD with rays, cont'd

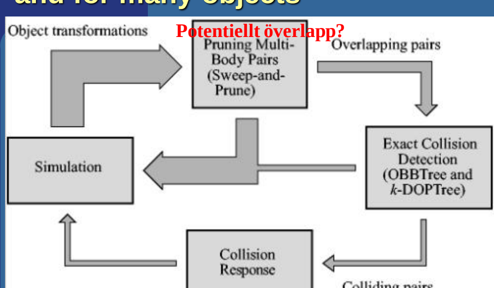
- We have simplified car, but not the road
- Turn to spatial data structures for the road
- Use BVH or BSP tree or height field, for example
- The distance along ray can be negative
- Therefore, either search ray in both positive and negative direction
- Or move back ray, until it is outside the BV of the road geometry

Another simplification

- Sometimes 3D can be turned into 2D operations
- Example: maze
- A human walking in maze, can be approximated by a circle
- Test circle against lines of maze
- Or even better, move walls outwards with circle radius
  - test center of circle against moved walls



A CD system for accurate detection and for many objects



- We'll deal with "pruning" and "exact CD"
- "Simulation" is how objects move

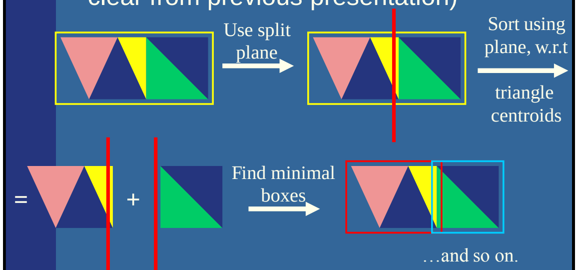
Complex object against complex object

- For object against object CD, see http://www.realtimerendering.com/int/
- If accurate result is needed, turn to BVHs
- Use a separate BVH for the two objects
- Test BVH against other BVH for overlap
- When triangles overlap, compute exact intersection, if needed

- But, first, a clarification on BVH building

BVH building example

- Can split on triangle level as well (not clear from previous presentation)



Pseudo code for BVH against BVH

If (not overlap(A,B)) return false

```
FindFirstHitCD(A, B)
returns ({TRUE, FALSE});
if(isLeaf(A) and isLeaf(B))
    for each triangle pair T_A ∈ A_c and T_B ∈ B_c
        if(overlap(T_A, T_B)) return TRUE;
else if(isNotLeaf(A) and isNotLeaf(B))
    if(Volume(A) > Volume(B))
        for each child C_A ∈ A_c
            FindFirstHitCD(C_A, B)
    else
        for each child C_B ∈ B_c
            FindFirstHitCD(A, C_B)
else if(isLeaf(A) and isNotLeaf(B))
    for each child C_B ∈ B_c
        FindFirstHitCD(C_B, A)
else
    for each child C_A ∈ A_c
        FindFirstHitCD(C_A, B)
return FALSE;
```

Pseudocode deals with 4 cases:

1) Leaf against leaf node
2) Internal node against internal node
3) Internal against leaf
4) Leaf against internal

Pseudo coden har ett litet fel:
Byt FindFirstHitCD()
mot if(FindFirstHitCD())
return true;

Comments on pseudocode

- The code terminated when it found the first triangle pair that collided
- Simple to modify code to continue traversal and put each pair in a list
- Reasonably simple to include rotations for objects as well
- Note that if we use AABB for both BVHs, then the AABB-AABB test becomes a AABB-OBB test

Tradeoffs

n_v : number of BV/BV overlap tests
c_v : cost for a BV/BV overlap test
n_p : number of primitive pairs tested for overlap
c_p : cost for testing whether two primitives overlap
n_u : number of BVs updated due to the model's motion
c_u : cost for updating a BV

- The choice of BV
  - AABB, OBB, k-DOP, sphere
- In general, the tighter BV, the slower test

- Less tight BV, gives more triangle-triangle tests in the end
- Cost function:

$$t = n_v c_v + n_p c_p + n_u c_u$$

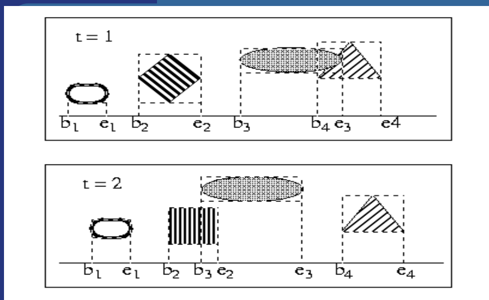
CD between many objects

- Why needed?
- Consider several hundreds of rocks tumbling down a slope…
- This system is often called "First-Level CD"
- We execute this system because we want to execute the 2nd system less frequently
- Assume high frame-to-frame coherency
  - Means that object is close to where it was previous frame
  - Reasonable

Sweep-and-prune algorithm
[by Ming Lin]

- Assume objects may translate and rotate
- Then we can find a minimal AABB, which is guaranteed to contain object for all rotations
- Do collision overlap three times
  - One for x,y, and z-axes
- Let's concentrate on one axis at a time
- Each AABB on this axis is an interval, from s_i to e_i, where i is AABB number

1-D Sweep and Prune



Original by Michael Zyda

Sweep-and-prune algorithm

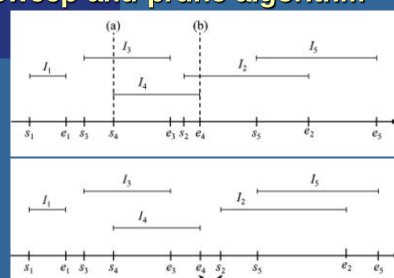
- Sort all s_i and e_i into a list
- Traverse list from start to end
- When an s is encounted, mark corresponding interval as active in an **`active_interval_list`**
- When an e is encountered, delete the interval in **`active_interval_list`**
- All intervals in **`active_interval_list`** are overlapping!

Sweep-and-prune algorithm

- Now sorting is expensive: O(n*log n)
- But, exploit frame-to-frame coherency!
- The list is not expected to change much
- Therefore, "resort" with bubble-sort, or insertion-sort
- Expected: O(n)

```
BUBBLE SORT
for (i=0; i<n-1; i++) {
   for (j=0; j<n-1-i; j++)
        if (a[j+1] < a[j]) { //compare the two neighbors
           tmp = a[j]; /* swap a[j] and a[j+1] */
           a[j] = a[j+1];
           a[j+1] = tmp;
        }
}
```

Sweep-and-prune algorithm



- Keep a boolean for each pair of intervals
- Possible invert when sort order changes
- If all boolean for all three axes are true, → overlap

Bonus

Toggling the flags (the gritty details)

These flags are only modified when insertion sort performs a swap. We decide whether or not to toggle a flag based on whether the coordinate values both refer to bounding box minima, both refer to bounding box maxima, or one refers to a bounding box minimum and the other a maximum.

When a flag is toggled, the overlap status indicates one of three situations:

1. All three dimensions of this bounding box pair now overlap. In this case, we add the corresponding polytope pair to a list of active pairs.
2. This bounding box pair overlapped at the previous time step. In this case, we remove the corresponding polytope pair from the active list.
3. This bounding box pair did not overlap at the previous time step and does not overlap at the current time step. In this case, we do nothing.

CD Conclusion

- Very important part of games!
- Many different algorithms to choose from
- Decide what's best for your case,
- and implement…