Collision Detection

- Fundamental to graphics, VR applications
- Applications include animation, games, virtual manufacturing, CAD/CAM flight/vehicle simulators, robotics, path/motion planning
- Termed more generally collision handling, it consists of the following:
  - collision detection: Do objects collide? (boolean response)
  - collision determination: Where exactly is the intersection?
  - collision response: What do you do as a result?

Efficient Collision Detection

- Realtime applications demand fast collision detection.
- Testing every pair of triangles impossible; with \( n \) static and \( m \) dynamic objects, cost is

\[
C = nm + \binom{n}{2}
\]

- Hierarchical bounding volume techniques can be used to efficiently solve collision detection,
  - In particular the OBB tree and k-DOP tree algorithms can deal with large numbers of polygons.
  - Can deal with rigid body motion (rotation+translation)
  - Efficient BV fitting, improving performance.

Collision Detection with Rays: Example

- Imagine a car 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
Collision Detection with Rays: Example

- Put a ray at each wheel
- Compute the closest intersection distance, \( t \), between ray and road geometry
  \[
  t = \begin{cases} 
  0 & \text{car on the road} \\
  > 0 & \text{car flying above road} \\
  < 0 & \text{car ploughing into road}
  \end{cases}
  \]
- Use values of \( t \) to compute a simple collision response

Collision Detection with Rays (contd)

- Have simplified car, but not the road
- Can use spatial data structures for the road
- Use BVH or BSP tree, for example
- The distance along ray can be negative
- Therefore, either search ray in both positive and negative direction
- Response: move back ray, until it is outside the BV of road geometry

Collision Detection with BSP Trees

- Efficient algorithms exist for performing set operations (union, intersection, difference, etc.) with BSP trees (Thibault/Naylor, SIGGRAPH ’87)
- Being a spatial data structure, BSP trees can be used to efficiently test for collisions.
- Collisions are an example of intersection with the geometry represented by BSP trees.
- Such operations are also possible with octrees and other spatial data structures.

Collision Detection with BSP Trees

- Assume object moving from \( p_0 \) at frame \( n \) to \( p_1 \) at frame \( n + 1 \).
- Easy to intersect ray segments against BSP tree; can have multiple intersections, but the first intersection is needed; front to back ordering is exploited.
- Easily extended to spheres: shift the plane by radius and test.
  \[
  \pi : \mathbf{n} \cdot \mathbf{x} + d = 0
  \]
  is shifted to test
  \[
  \pi' : \mathbf{n} \cdot \mathbf{x} + d \pm r = 0
  \]
  where the sign of \( r \) depends on the space containing the character.
Collision Detection with BSP Trees (contd.)

- Spheres are not approximate characters very well - convex hull of character or cylinder is better.
- Add

\[-\max_{v_i \in S} (n \cdot (v_i - p_0))\]

- Computationally more expensive, as the vertices are projected along \(n\) to determine the max shift needed.
- Using a cylinder is compromise (see text for details).

General Hierarchical Collision Detection

- Collision detection is performed between models represented using Bounding Volume Hierarchies (BVH)
- High level algorithm is the same, regardless of BVH implementation
- Cost functions are used to evaluate and compare performance

Hierarchy Construction

- Trees can be built bottom-up, incremental insertion, top-down
- To create efficient, tight structures, the bounding volumes must be of minimal size.
- Methods: Bottom-up
  - Closely located objects clustered and their BV computed
  - Clusters are joined into larger clusters, until we are left with a single cluster (entire scene).
  - BOXTREE (Barquet et al., 1996)
- Methods: Incremental Insertion
  - Objects inserted one at a time into an initially empty tree
  - Heuristics used to ensure that inserted object minimizes increase in tree volume
  - Automatic BV Hierarchy (Goldsmith ‘87) for ray tracing.
Hierarchy Construction (contd)

- Methods: Top-Down
  - Most common hierarchy construction algorithm
  - Starts with BV of scene
  - Divide and conquer to recursively split the objects
  - Need to determine a splitting axis, and a split point in the axis.
  - Lazy evaluation possible, but bad idea for real-time applications.
  - Balanced trees vs. more optimal trees

Collision Testing Between Hierarchies

- Given A and B are two nodes in the model hierarchies, $A_{BV}$, $B_{BV}$ are the respective BVs, the following is the general CD testing algorithm:

```c
FindFirstHitCD(A, B)
returns {TRUE, FALSE});
1: if(isLeaf(A) and isLeaf(B))
2: for each triangle pair $T_A \in A$, $T_B \in B_p$
3: if(overlap($T_A, T_B$)) return TRUE;
4: else if(isNotLeaf(A) and isNotLeaf(B))
5: if(overlap($A_{BV}, B_{BV}$))
6: if(Volume(A) > Volume(B))
7: for each child $C_A \in A$
8: FindFirstHitCD($C_A, B$)
9: else
10: for each child $C_B \in B$
11: FindFirstHitCD($A, C_B$)
12: else if(isLeaf(A) and isNotLeaf(B))
13: if(overlap($T_A, B_{BV}$))
14: for each child $C_B \in B$
15: FindFirstHitCD($C_B, A$)
16: else
17: if(overlap($T_B, A_{BV}$))
18: for each child $C_A \in A$
19: FindFirstHitCD($C_A, B$)
20: return FALSE.
```

Cost Function

- Cost of collision detection testing using BVHs can be quantified as follows:

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

where

- $n_v$ = number of BV/BV overlap tests
- $c_v$ = cost of a BV/BV overlap test
- $n_p$ = number of primitive pair overlap tests
- $c_p$ = cost of a primitive pair overlap test
- $n_u$ = number of BV updated due to model's motion
- $c_u$ = cost for updating a BV

Collision Testing Between Hierarchies: Notes

- Node with larger volume is descended; larger boxes generally give better performance
- Algorithm performs BV/BV and triangle/triangle tests; can also modify algorithm to test triangle vs. BV (reduced memory for BVs)
- Algorithm can be modified to return all the overlapping pairs of triangles (collision determination)
**Axis-Aligned Bounding Box**

- A box whose faces have normals that are all pairwise orthogonal, and aligned with the principal axes.
- Defined by extreme points, $a_{\text{min}}$ and $a_{\text{max}}$ where $a_{\text{min}}^i \leq a_{\text{max}}^i$, \( \forall i \in \{x, y, z\} \)

**Oriented Bounding Box (OBB)** (Gottschalk 1999)

- Oriented Bounding Box: A box whose faces have normals that are all pairwise orthogonal, or its an AABB that is arbitrarily rotated.
- OBB defined by a center point, $b^c$ and three normalized normals, $b^u$, $b^v$, $b^w$, with positive half lengths $h^B_u$, $h^B_v$, $h^B_w$.
- OBBS, because of their better fit, perform considerably better than spheres or AABB

**Oriented Bounding Box: Creation**

- First the convex hull of the object needs to be computed, results in a set of $n$ triangle vertices, $p^k$, $q^k$, $r^k$, $0 < k < n$.
- Area of triangle $k$ is $a^k$, total area of convex hull is $a^H = \sum_{k=0}^{n-1} a^k$, centroid of triangle $k$ is $m^i = (p^i + q^i + r^i)/3$. Centroid of convex hull is
  
  $$m^H = \frac{1}{a^H} \sum_{k=0}^{n-1} a^k m^k$$

- Compute the covariance matrix $C$ whose eigen vectors are the direction vectors (principal directions) of a good-fit box:
  
  $$C = [c_{ij}] = \left[ \frac{1}{a^H} \sum_{k=0}^{n-1} a^k \right] _{ij} (m_i^k m_j^k + p_i^k p_j^k + q_i^k q_j^k + r_i^k r_j^k) - m_i^H m_j^H$$

  with $0 < i, j < 3$
### OBB/OBB Overlap Test

- Once the direction vectors of OBB have been found, then a sequence of separating axes is determined, 3 each from the normals to the faces of both OBBs, 9 more from the combinations of edges between the two OBBs.
- Project vertices from OBB onto separating axes and test for overlap.
- Details in Section 13.12.5

### OBB Tree Algorithm (Gottschalk 1996)

- An order of magnitude faster with a new overlap test.
- Reason: One of the OBBs is transformed into an AABB around origin.
- Cost: Transform takes 63 ops, and overlap test may exit after one of the 15 axis tests (axis test range from 17 - 180 ops).
- Might skip the last 9 axis tests, as they tend to cull relatively small portion of overlaps.

### OBB Tree Algorithm: Tree Construction

- Top down approach.
- Top level OBB is split along long axis into two groups, primitives split based on their centroids.
- Can also use the median to split primitives, resulting in balanced trees.

### k-Discrete Orientation Polytope (k-DOP) (Kloslowski 1998)

- Convex Polytope defined by $k/2$ ($k$ is even and usually a small number) normalized normals (orientations), $n_i$, $1 \leq i \leq k/2$, with each $n_i$ associated with two scalar values $d_{i_{\min}}$, $d_{i_{\max}}$, with $d_{i_{\min}} < d_{i_{\max}}$.
- Each triple $(n_i, d_{i_{\min}}, d_{i_{\max}})$ describes a slab, $S_i$. 
**k-Discrete Orientation Polytope (k-DOP)**

Each slab is the volume between the two planes

\[
\pi_{i,\text{min}} : n_i \cdot x + d_{i,\text{min}} \\
\pi_{i,\text{max}} : n_i \cdot x + d_{i,\text{max}}
\]

- k-DOP is the intersection of all slabs, \(S_i\)

\[
k - \text{DOP} = \bigcap_{1 \leq i \leq k/2} S_i
\]

---

**k-DOP/k-DOP Intersection Test**

- k-DOP determined by a small, fixed set of outward \(k\) pointing normals.
- Used as an efficient BV in raytracing (Kay/Kajiya 1986)
- Intersection test costs \(k/2\) interval overlap tests.
- Can be shown for moderate values of \(k\) \((k = 18)\) overlap test of two k-DOPs is an order of magnitude faster than two OBBs.
- AABB: A special case of a 6-DOP
- As \(k\) increases, k-DOP resembles the convex hull

---

**k-DOP in Collision Detection**

- Why k-DOP in CD? Faster BV/BV test compared to using OBB and a tighter fit.
- Why faster? Orientations of the k-DOP are fixed.
- When Objects move, k-DOP has the added cost of updating the BV, while this is integrated into the OBB/OBB overlap test.
- \(k = 18\) has been demonstrated to give good performance in practice (tradeoff between tighter BV and overlap test cost)
- \(k = 18\), In addition to the 3 normals of AABB, also use sum of their normals \((\pm 1, \pm 1, 0)\)
- \(k = 26\), use AABB normals, plus \((\pm 1, \pm 1, \pm 1)\)
**$k$-DOP Tree Construction**

- Uses a binary tree, generally contains multiple primitives (40 used by Klosowski) at leaf nodes, as $k$-DOPs are expensive to transform
- **Top Down Approach:**
  - Min Sum: Minimizes sum of sub-volumes
  - Min Max: Minimizes the larger of the 2 sub-volumes
  - Splatter: Project variances of centroids of triangles onto axis; select axis with largest variance
  - Longest Side: BV with longest side to split
- Splatter performs best in collision detection; using the mean of the centroids is better than the median.
- Classify triangles based on their centroids.

**Updating the $k$-DOP Tree due to Motion**

- When an object undergoes rigid motion, its $k$-DOP is no longer valid and must be updated.

**Approximation Method**

- Transform the original $k$-DOP vertices and then recompute a new $k$-DOP
- Advantageous, if the number of vertices on the $k$-DOP is smaller than the contained geometry, but fit can become worse

**Updating the $k$-DOP Tree: Hill Climbing**

- More expensive, requires convex hull of object
- Local updates are made in each frame, maximal vertices are tested to see if they are valid, else a new vertex (neighbor) is chosen to update the k-DOP.
- More expensive, but tighter BV (use hill climbing at root, and approximation method at lower level nodes)