Collision and Proximity Queries

Dinesh Manocha

(based on slides from Ming Lin)

COMP790-058
Fall 2013

Geometric Proximity Queries

- **Given two object, how would you check:**
  - *If they intersect with each other while moving?*
  - *If they do not interpenetrate each other, how far are they apart?*
  - *If they overlap, how much is the amount of penetration*

Collision Detection

- Update configurations w/ TXF matrices
- Check for edge-edge intersection in 2D
  (Check for edge-face intersection in 3D)
- Check every point of A inside of B &
  every point of B inside of A
- Check for pair-wise edge-edge intersections

Imagine larger input size: N = 1000+ ……

Classes of Objects & Problems

- 2D vs. 3D
- Convex vs. Non-Convex
- Polygonal vs. Non-Polygonal
- Open surfaces vs. Closed volumes
- Geometric vs. Volumetric
- Rigid vs. Non-rigid (deformable/flexible)
- Pairwise vs. Multiple (N-Body)
- CSG vs. B-Rep
- Static vs. Dynamic

And so on… This may include other geometric representation schemata, etc.

Some Possible Approaches

- Geometric methods
- Algebraic Techniques
- Hierarchical Bounding Volumes
- Spatial Partitioning
- Others (e.g. optimization)

Voronoi Diagrams

- Given a set S of n points in R^2, for each point p_i in S, there is the set of points (x, y) in the plane that are closer to p_i than any other point in S, called Voronoi polygons. The collection of n Voronoi polygons given the n points in the set S is the "***Voronoi diagram***", $Vor(S)$, of the point set S.

Intuition: To partition the plane into regions, each of these is the set of points that are closer to a point p_i in S than any other. The partition is based on the set of closest points, e.g. bisectors that have 2 or 3 closest points.

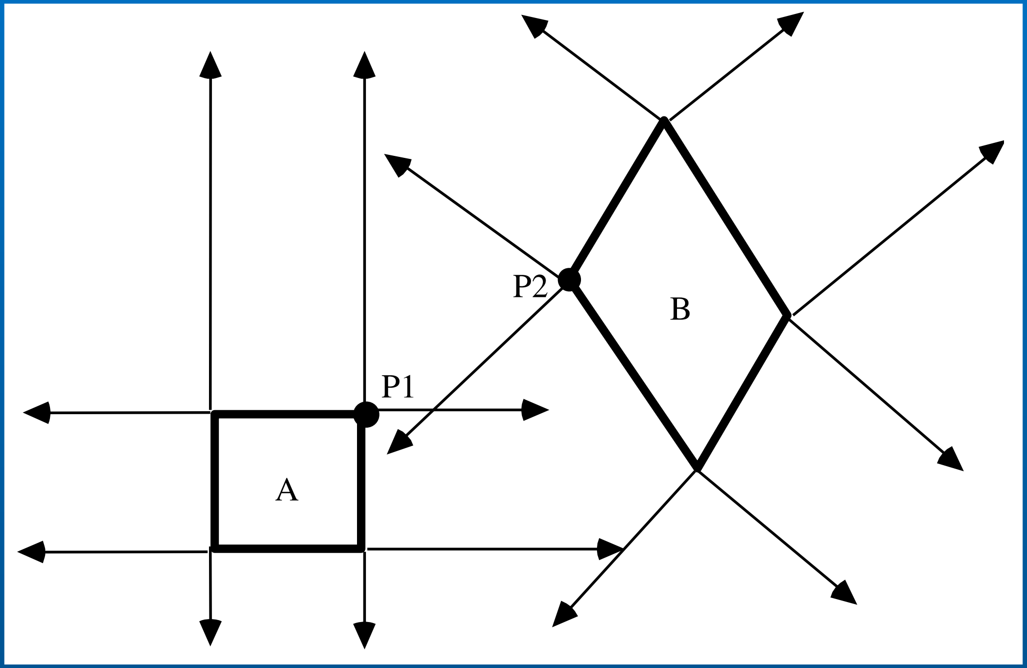
Generalized Voronoi Diagrams

- **The extension of the Voronoi diagram to higher dimensional features (such as edges and facets, instead of points); i.e. the set of points closest to a *feature*, e.g. that of a polyhedron.**
- **<u>FACTS:</u>**
  - In general, the generalized Voronoi diagram has quadratic surface boundaries in it.
  - If the polyhedron is convex, then its generalized Voronoi diagram has planar boundaries.

Voronoi Regions

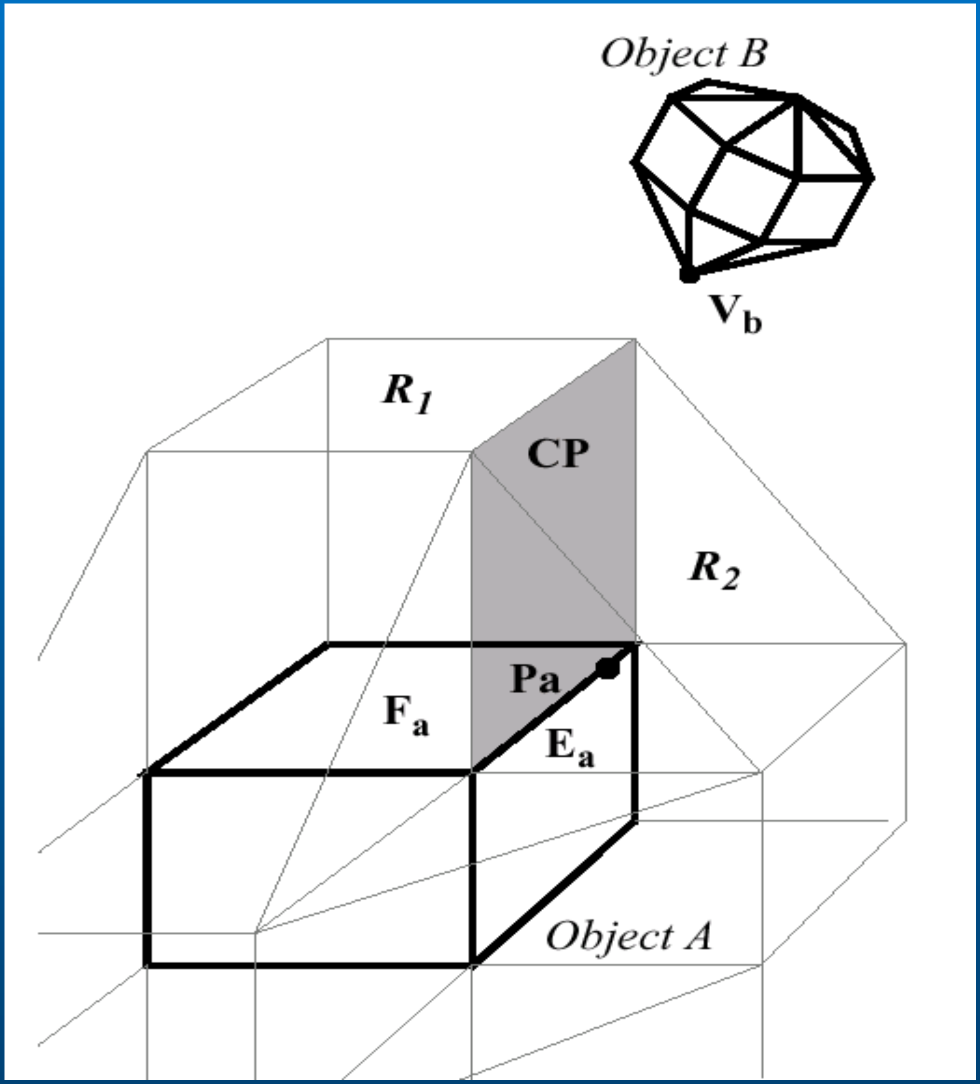
- A ***Voronoi region*** associated with a ***feature*** is a set of points that are closer to that feature than any other.
- **FACTS:**
  - The Voronoi regions form a partition of space outside of the polyhedron according to the closest feature.
  - The collection of Voronoi regions of each polyhedron is the generalized Voronoi diagram of the polyhedron.
  - The generalized Voronoi diagram of a convex polyhedron has linear size and consists of polyhedral regions. And, all Voronoi regions are convex.

Simple 2D Example



Objects A & B and their Voronoi regions: P1 and P2 are the pair of closest points between A and B. Note P1 and P2 lie within the Voronoi regions of each other.

Basic Idea for Voronoi Marching



Linear Programming

In general, a d-dimensional linear program-ming (or linear optimization) problem may be posed as follows:

- **Given a finite set A in R^d**
- **For each a in A, a constant K_a in R, c in R^d**
- **Find x in R^d which minimize $<x, c>$**
- **Subject to $<a, x> \geq K_a$, for all a in A .**

where $<*, *>$ is standard inner product in R^d.

LP for Collision Detection

Given two finite sets A, B in R^d

For each ***a*** in A and ***b*** in B,

Find ***x*** in R^d which minimize *whatever*

Subject to $\langle \boldsymbol{a}, \boldsymbol{x}\rangle > \boldsymbol{0}$, for all ***a*** in A

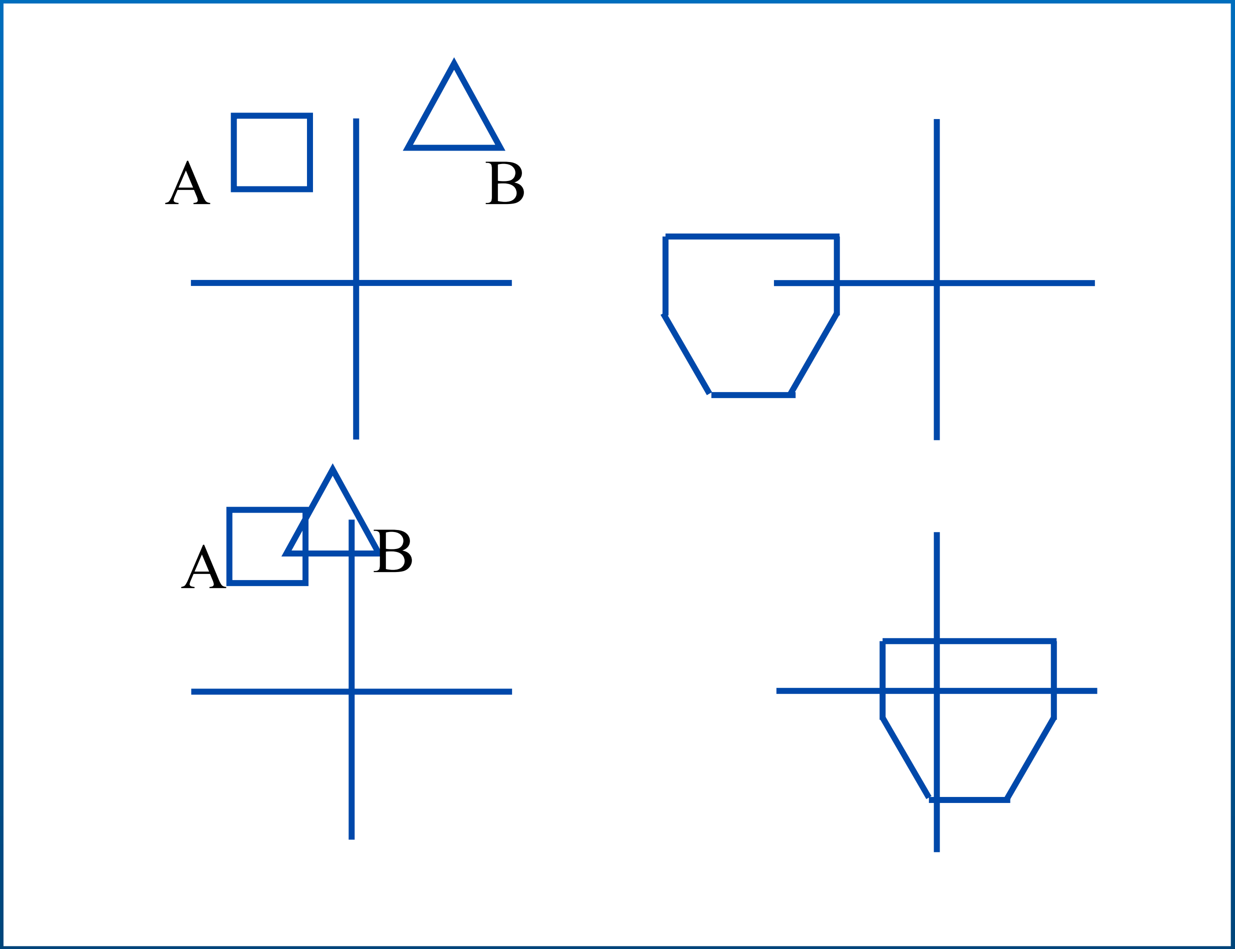
And $\langle \boldsymbol{b}, \boldsymbol{x}\rangle < \boldsymbol{0}$, for all ***b*** in B

where d = 2 (or 3).

Minkowski Sums/Differences

- Minkowski Sum (A, B) = { $a + b \mid a \in A, b \in B$ }
- Minkowski Diff (A, B) = { $a - b \mid a \in A, b \in B$ }
- A and B collide iff Minkowski Difference(A,B) contains the point 0.

Some Minkowski Differences



Minkowski Difference & Translation

- Minkowski-Diff(Trans(A, t_1), Trans(B, t_2)) = Trans(Minkowski-Diff(A,B), $t_1 - t_2$)

$\Rightarrow$ Trans(A, t_1) and Trans(B, t_2) intersect iff Minkowski-Diff(A,B) contains point ($t_2 - t_1$).

Properties

- **Distance**
  - distance(A,B) = $\min_{a \in A,\ b \in B} \| a - b \|_2$
  - distance(A,B) = $\min_{c \in \text{Minkowski-Diff}(A,B)} \| c \|_2$
  - if A and B disjoint, c is a point on boundary of Minkowski difference
- **Penetration Depth**
  - pd(A,B) = $\min\{ \| t \|_2 \mid A \cap \text{Translated}(B,t) = \emptyset \}$
  - pd(A,B) = $\min_{t \notin \text{Minkowski-Diff}(A,B)} \| t \|_2$
  - if A and B intersect, t is a point on boundary of Minkowski difference

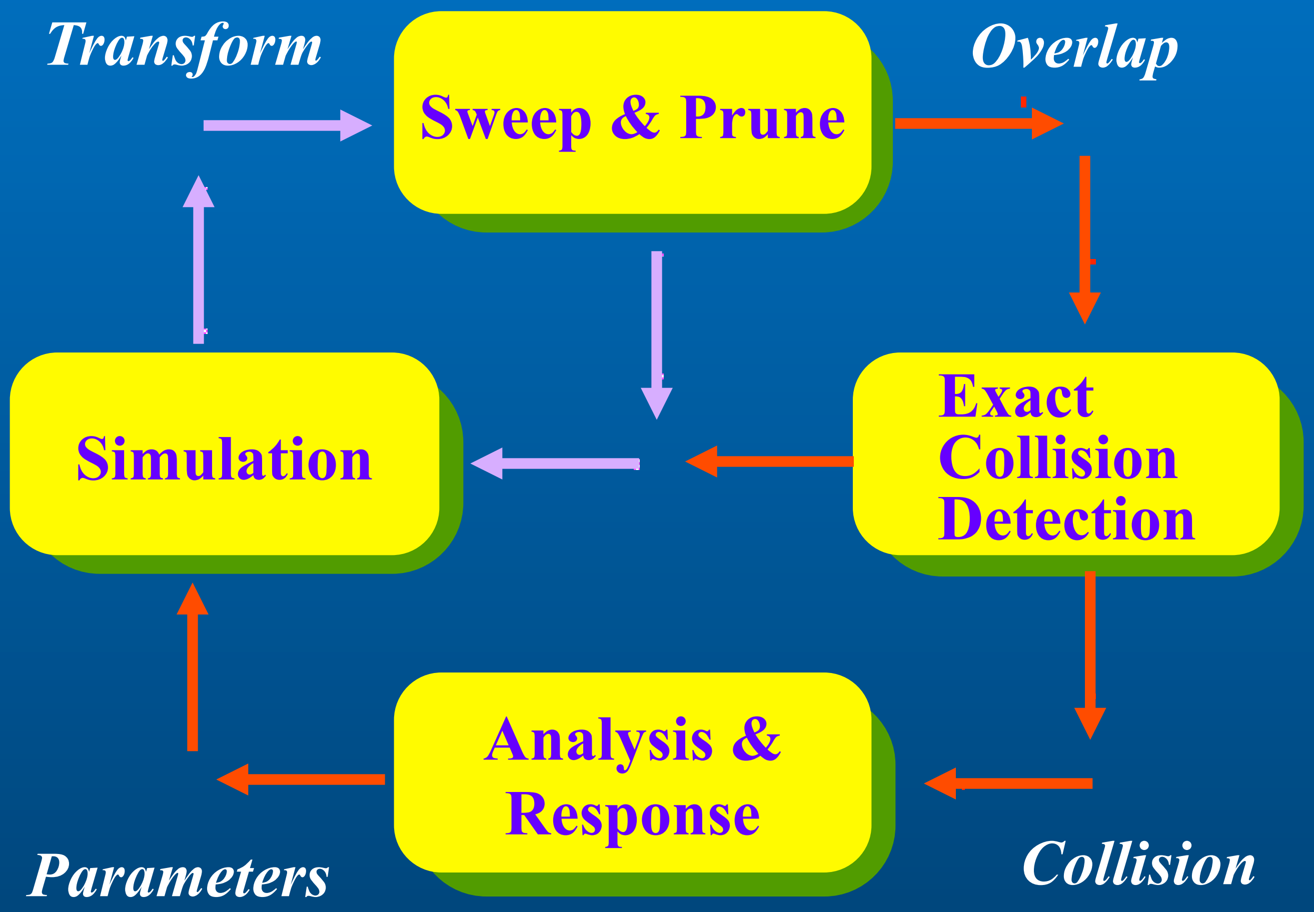
GJK for Computing Distance between Convex Polyhedra

```
GJK-DistanceToOrigin ( P )   // dimension is m
Initialize P_0 with m+1 or fewer points.
k = 0
while (TRUE) {
        if origin is within CH( P_k ), return 0
        else {
            find x ∈ CH(P_k) closest to origin, and S_k ⊂ P_k s.t. x ∈ CH(S_k)
            see if any point p_-x in P more extremal in direction -x
            if no such point is found, return |x|
            else {
                P_k+1 = S_k ∪ {p_-x}
                k = k + 1
            }
        }
}
```

Large, Dynamic Environments

- For dynamic simulation where the velocity and acceleration of all objects are known at each step, use the scheduling scheme (implemented as heap) to prioritize "critical events" to be processed.
- Each object pair is tagged with the estimated time to next collision. Then, each pair of objects is processed accordingly. The heap is updated when a collision occurs.

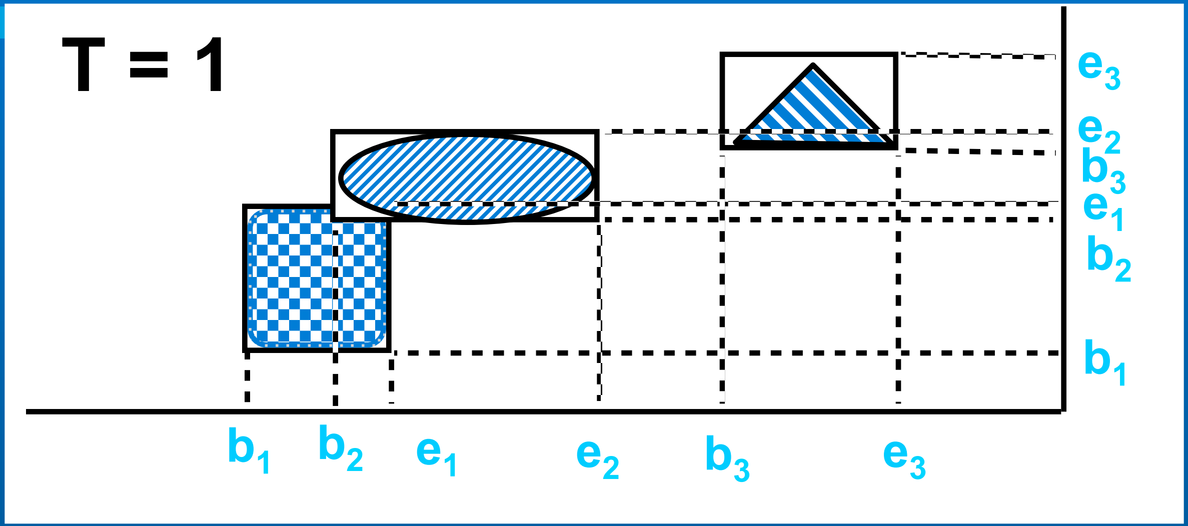
Collide System Architecture

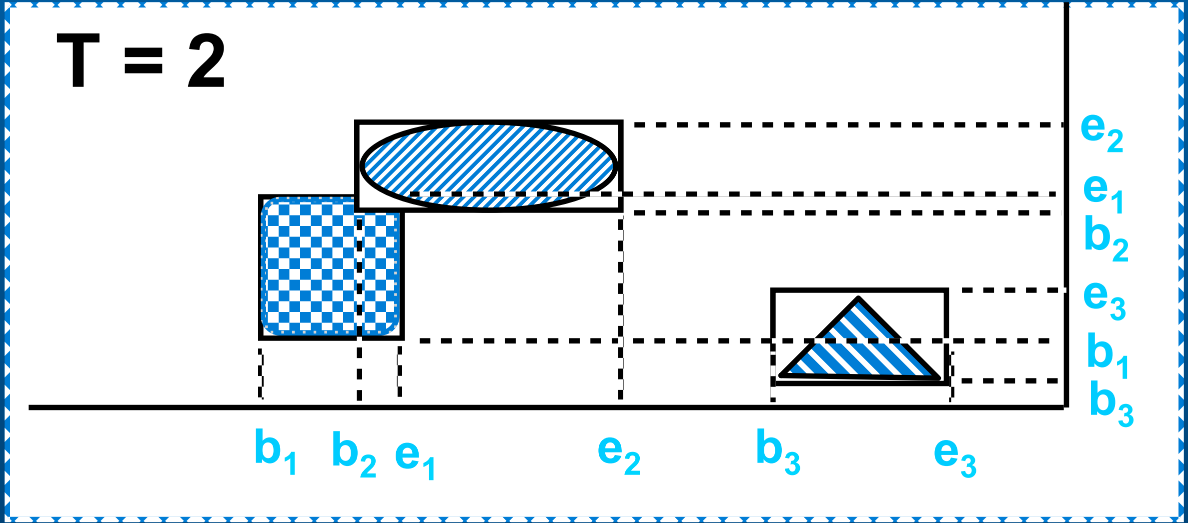


Sweep and Prune

- Compute the axis-aligned bounding box (fixed vs. dynamic) for each object
- Dimension Reduction by projecting boxes onto each *x, y, z*- axis
- Sort the endpoints and find overlapping intervals
- Possible collision -- only if projected intervals overlap in all 3 dimensions

Sweep & Prune


T = 1
e3
e2
b3
e1
b2
b1
b1 b2 e1 e2 b3 e3


T = 2
e2
e1
b2
e3
b1
b3
b1 b2 e1 e2 b3 e3

Updating Bounding Boxes

- **Coherence** (greedy algorithm)
- **Convexity properties** (geometric properties of convex polytopes)
- **Nearly constant time**, if the motion is relatively “small”

Collision and Proximity Queries

Dinesh Manocha

(based on slides from Ming Lin)

Methods for General Models

- Decompose into convex pieces, and take minimum over all pairs of pieces:
  - Optimal (minimal) model decomposition is NP-hard.
  - Approximation algorithms exist for closed solids, but what about a list of triangles?
- Collection of triangles/polygons:
  - n*m pairs of triangles - brute force expensive
  - Hierarchical representations used to accelerate minimum finding

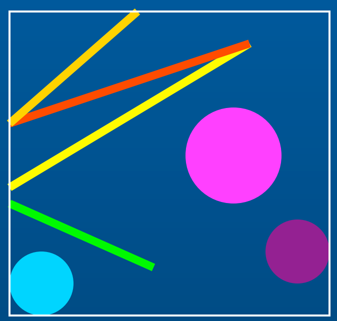
Hierarchical Representations

- Two Common Types:
  - Bounding volume hierarchies – trees of spheres, ellipses, cubes, axis-aligned bounding boxes (AABBs), oriented bounding boxes (OBBs), K-dop, SSV, etc.
  - Spatial decomposition - BSP, K-d trees, octrees, MSP tree, R-trees, grids/cells, space-time bounds, etc.
- Do very well in "rejection tests", when objects are far apart
- Performance may slow down, when the two objects are in close proximity and can have multiple contacts

BVH vs. Spatial Partitioning

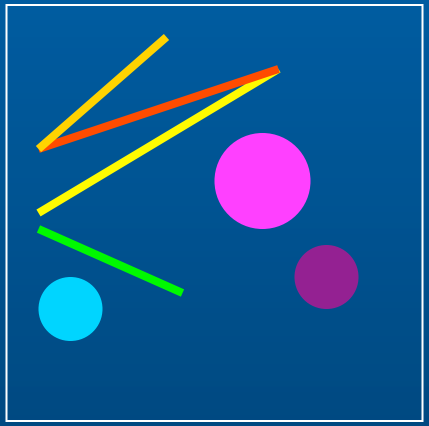
BVH:

- Object centric
- Spatial redundancy



SP:

- Space centric
- Object redundancy



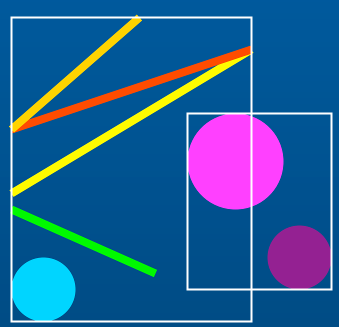
BVH vs. Spatial Partitioning

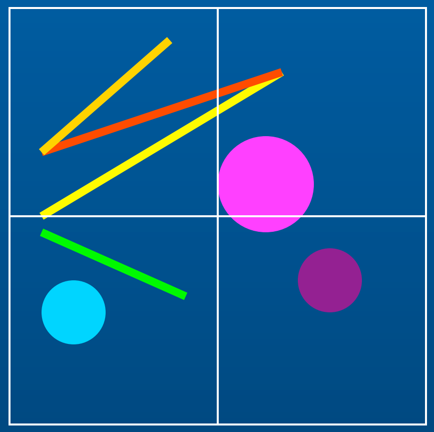
BVH:

- Object centric
- Spatial redundancy

SP:

- Space centric
- Object redundancy

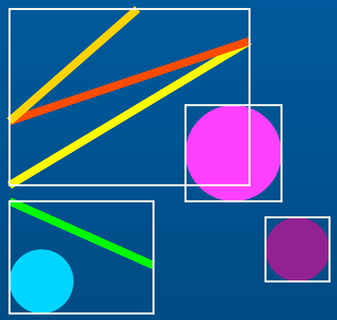




BVH vs. Spatial Partitioning

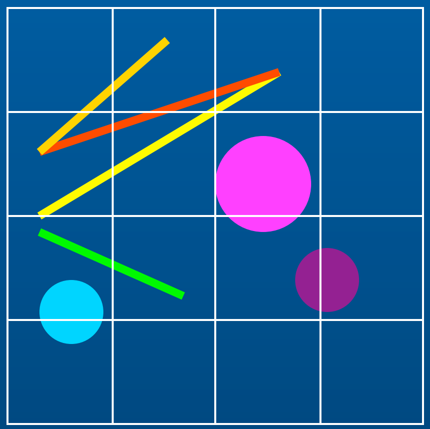
BVH:

- Object centric
- Spatial redundancy



SP:

- Space centric
- Object redundancy



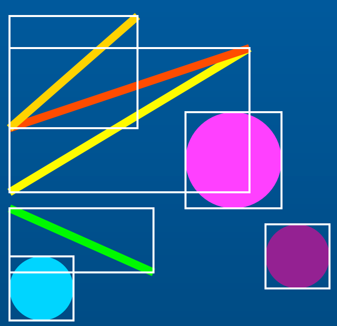
BVH vs. Spatial Partitioning

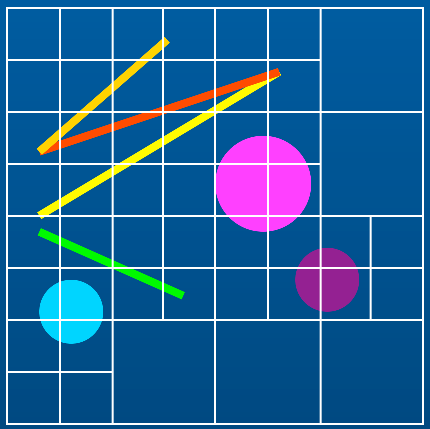
BVH:

- Object centric
- Spatial redundancy

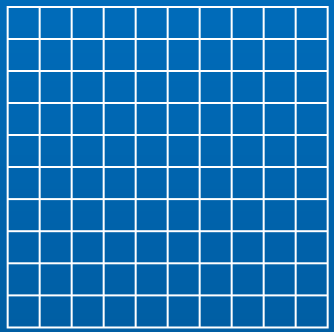
SP:

- Space centric
- Object redundancy

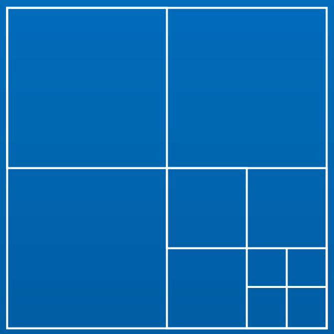




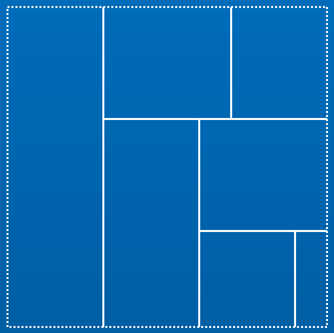
Spatial Data Structures & Subdivision



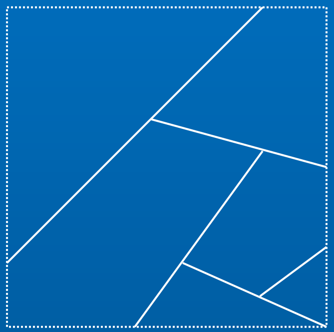
Uniform Spatial Sub



Quadtree/Octree



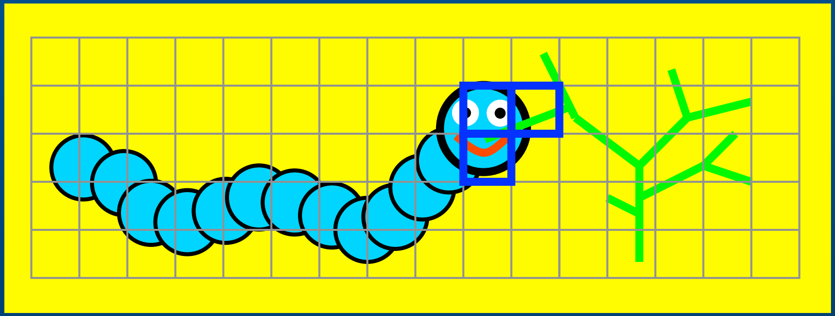
kd-tree



BSP-tree

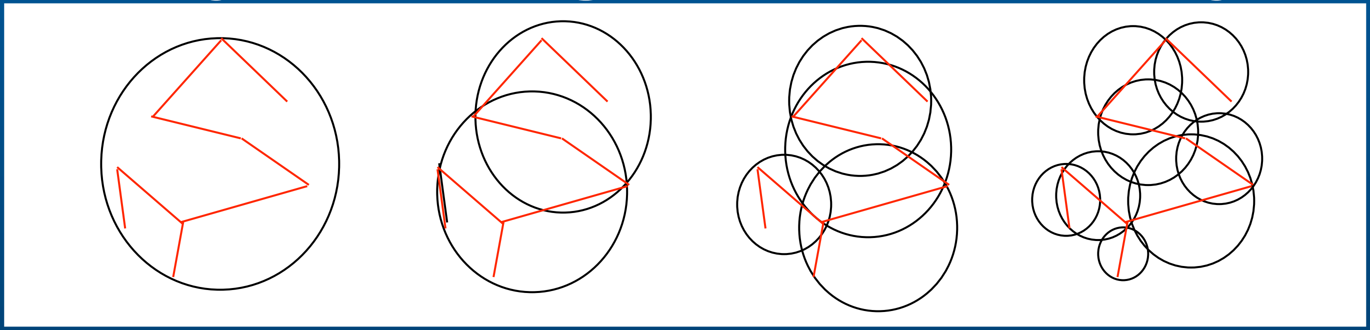
Uniform Spatial Subdivision

- Decompose the objects (the entire simulated environment) into identical cells arranged in a fixed, regular grids (equal size boxes or voxels)
- To represent an object, only need to decide which cells are occupied. To perform collision detection, check if any cell is occupied by two object
- Storage: to represent an object at resolution of *n* voxels per dimension requires upto n^3 cells
- Accuracy: solids can only be "approximated"



Bounding Volume Hierarchies

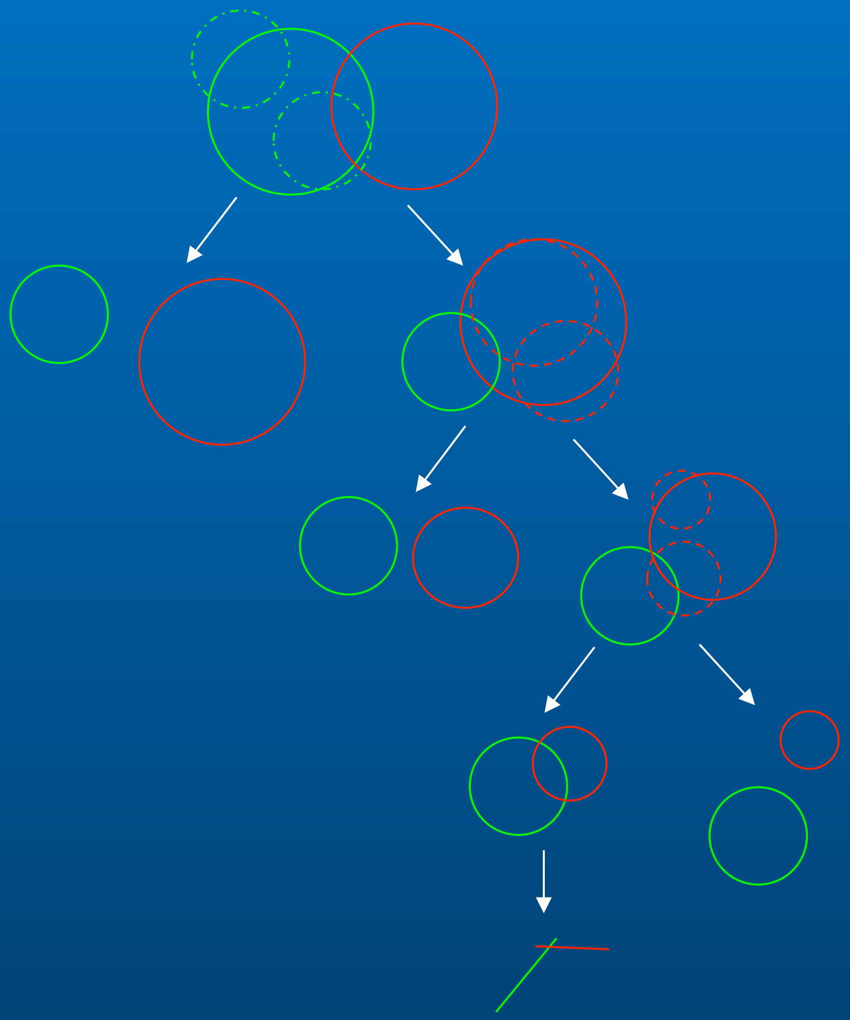
- Model Hierarchy:
  - each node has a simple volume that bounds a set of triangles
  - children contain volumes that each bound a different portion of the parent's triangles
  - The leaves of the hierarchy usually contain individual triangles
- A binary bounding volume hierarchy:



Type of Bounding Volumes

- Spheres
- Ellipsoids
- Axis-Aligned Bounding Boxes (AABB)
- Oriented Bounding Boxes (OBBs)
- Convex Hulls
- k-Discrete Orientation Polytopes (k-dop)
- Spherical Shells
- Swept-Sphere Volumes (SSVs)
  – Point Swetp Spheres (PSS)
  – Line Swept Spheres (LSS)
  – Rectangle Swept Spheres (RSS)
  – Triangle Swept Spheres (TSS)

BVH-Based Collision Detection



Collision Detection using BVH

1. Check for collision between two parent nodes (starting from the roots of two given trees)
2. If there is no interference between two parents,
3. Then stop and report "no collision"
4. Else All children of one parent node are checked against all children of the other node
5. If there is a collision between the children
6. Then If at leave nodes
7. Then report "collision"
8. Else go to Step 4
9. Else stop and report "no collision"

Evaluating Bounding Volume Hierarchies

Cost Function:

$$F = N_u \text{ x } C_u + N_{bv} \text{ x } C_{bv} + N_p \text{ x } C_p$$

F: total cost function for interference detection

N_u: no. of bounding volumes updated

C_u: cost of updating a bounding volume,

N_{bv}: no. of bounding volume pair overlap tests

C_{bv}: cost of overlap test between 2 BVs

N_p: no. of primitive pairs tested for interference

C_p: cost of testing 2 primitives for interference

Designing Bounding Volume Hierarchies

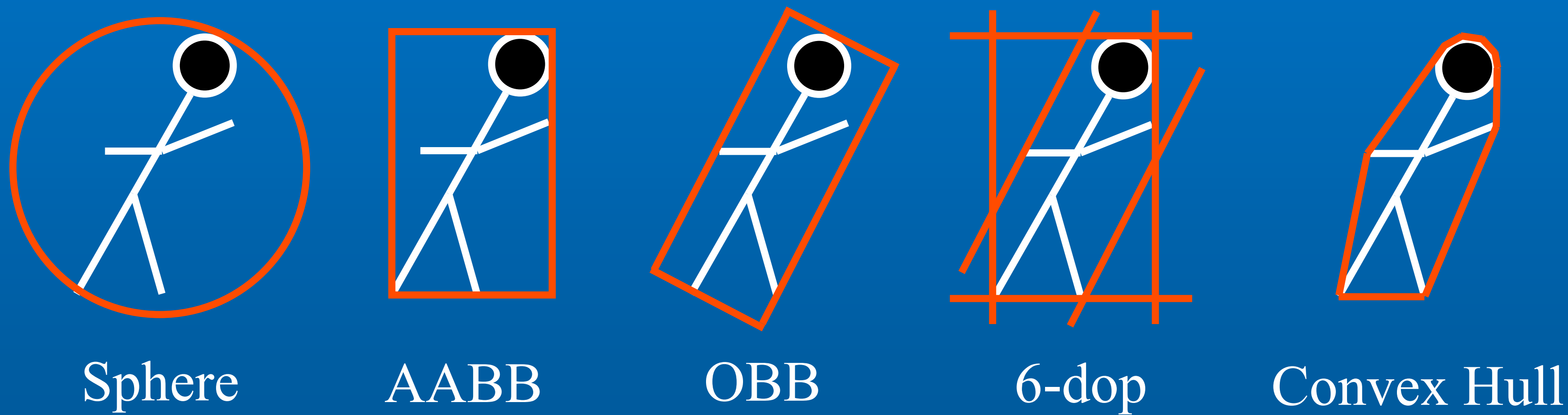
The choice governed by these constraints:

- It should fit the original model as tightly as possible (to lower N_{bv} and N_p)
- Testing two such volumes for overlap should be as fast as possible (to lower C_{bv})
- It should require the BV updates as infrequently as possible (to lower N_u)

Observations

- Simple primitives (spheres, AABBs, etc.) do very well with respect to the second constraint. But they cannot fit some long skinny primitives tightly.
- More complex primitives (minimal ellipsoids, OBBs, etc.) provide tight fits, but checking for overlap between them is relatively expensive.
- Cost of BV updates needs to be considered.

Trade-off in Choosing BV's



increasing complexity & tightness of fit →

← decreasing cost of (overlap tests + BV update)

Building Hierarchies

- Choices of Bounding Volumes
  - cost function & constraints
- Top-Down vs. Bottum-up
  - speed vs. fitting
- Depth vs. breadth
  - branching factors
- Splitting factors
  - where & how