Computer Graphics Lecture 09: Collision Detection and Response

DR. ELVIS S. LIU

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

SPRING 2018

Assignment 1: Simple Ray Tracer

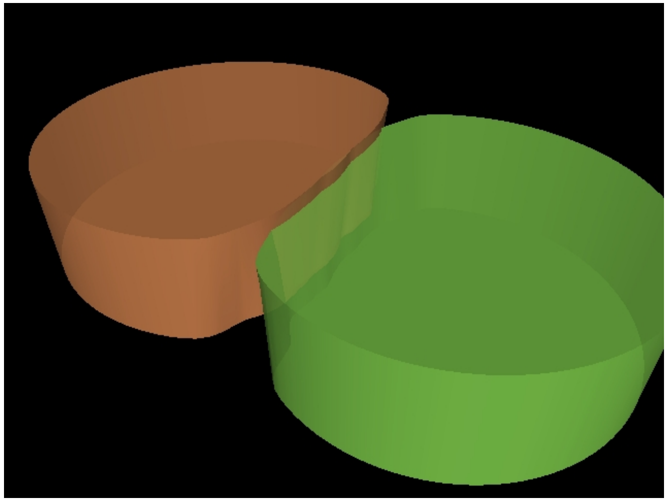
- Deadline: 12 June 2018
- Grading Criteria (total: 10%)
  - Scene construction 2%
  - Colour 2%
  - Reflection 2%
  - Transparency 2%
  - Shadow 2%
- Bonus (total: 4%)
  - Supersampling 2%
  - Spatial data structure 2%

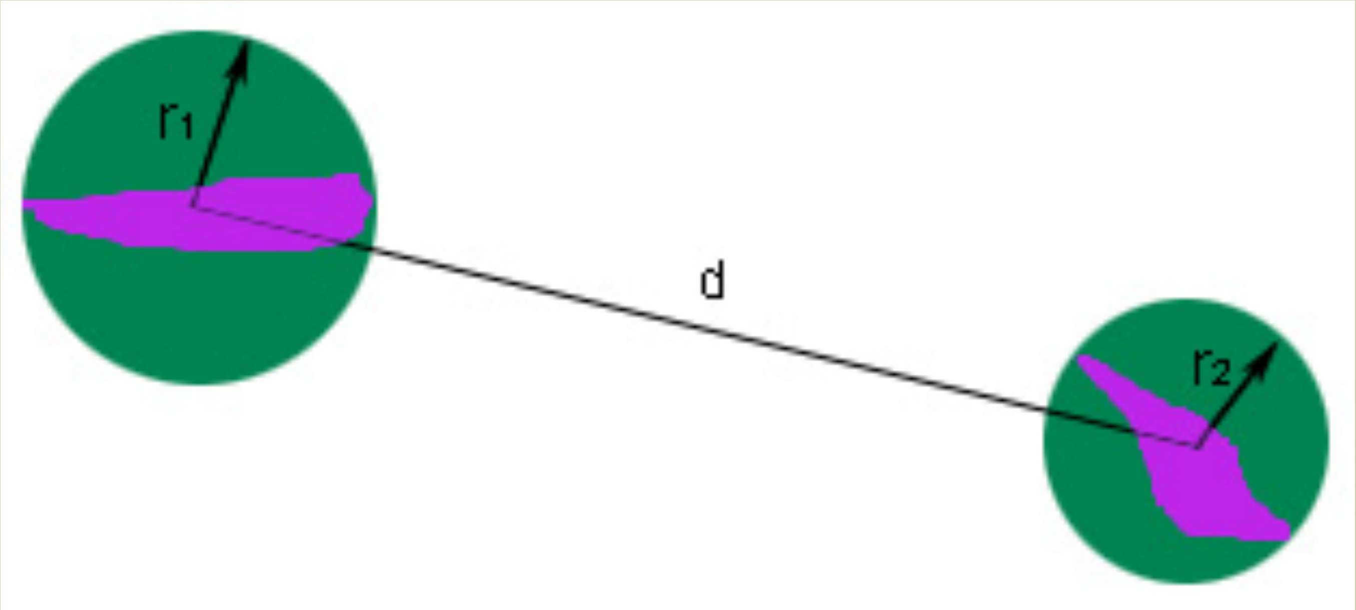
Illusion of Presence

- Cars fly through trees?
- Human walk through building walls?
- Bullets pass through targets without notice?
- Example
  - https://www.youtube.com/watch?v=8-gv7HO4OU4

Collision Detection

- Necessary for realistic gameplay
- Used by most of the games
- Prevent walking through obstacles (e.g. walls)
- Prevent bullets penetrating a soldier's body without wounding him





d < r1 + r2?

Collision Detection in Unity3D

- https://www.youtube.com/watch?t=102&v=QRp4V1JTZnM

Exact Collision Detection

- A 3D model is a collection of primitives, such as triangles
- Exact collision detection is done at the primitive level, which usually involves a large number to triangle to triangle comparisons
- World of Warcraft is famous for its low polygon modelling
- A female Blood Elf model contains 19000 polygons
- Exact collision detection could be very time consuming

Collision Response for Particles

LECTURE 09: COLLISION DETECTION AND RESPONSE

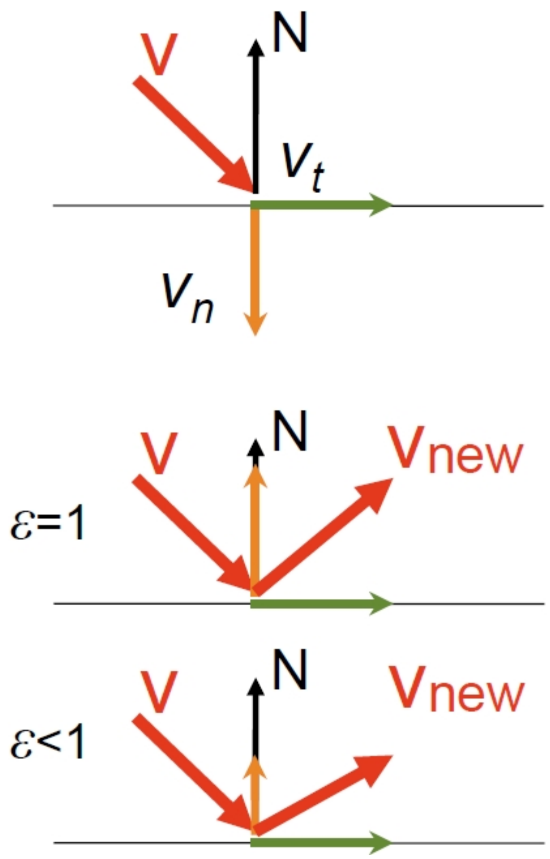
Collision Response for Particles

- Tangential velocity v_t often unchanged
- Normal velocity v_n reflects:

$$v = v_t + v_n$$

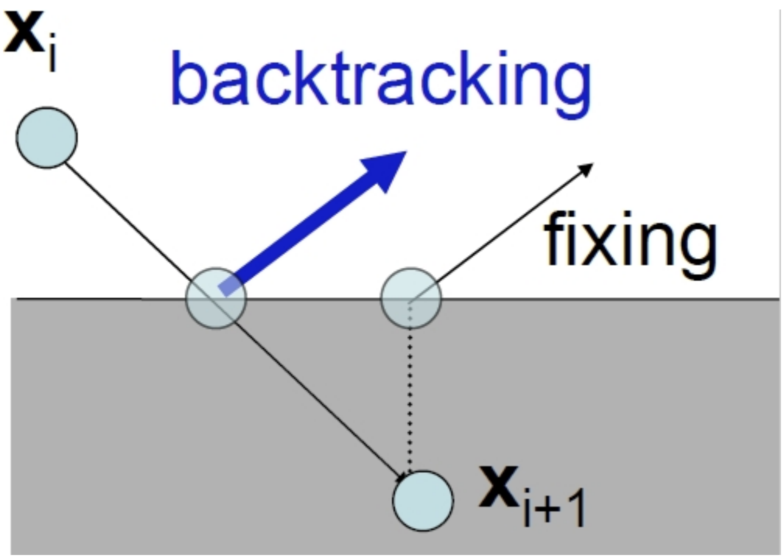
$$v \leftarrow v_t - \varepsilon v_n$$

- Coefficient of restitution ε
- When ε = 1, mirror reflection



Overshooting

- Usually, we detect collision when it is too late: we are already inside
- Solution: Back up
  - Compute intersection point
  - Ray-object intersection!
  - Compute response there
  - Advance for remaining fractional time step
- Other solution: Quick and dirty hack
  - Just project back to object closest point



Separating Axis Theorem

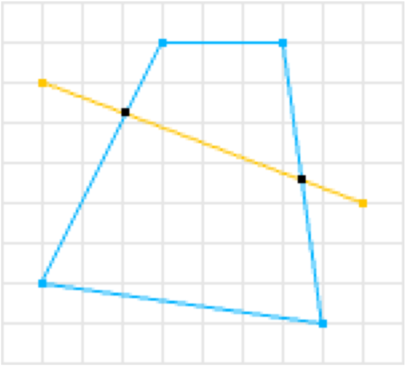
LECTURE 09: COLLISION DETECTION AND RESPONSE

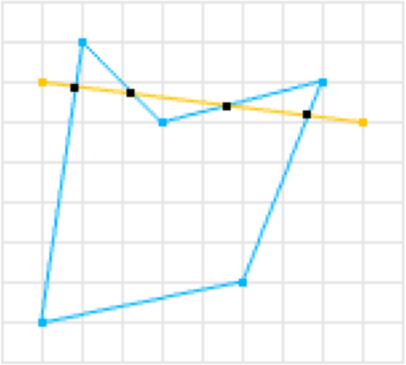
Separating Axis Theorem

- The Separating Axis Theorem, SAT for short, is a method to determine if two convex shapes are intersecting
- The algorithm can also be used to find the minimum penetration vector which is useful for physics simulation and a number of other applications
- SAT is a fast generic algorithm that can remove the need to have collision detection code for each shape type pair thereby reducing code and maintenance
- It is an ideal method for triangle-triangle tests and also OBB-OBB tests

Convexity

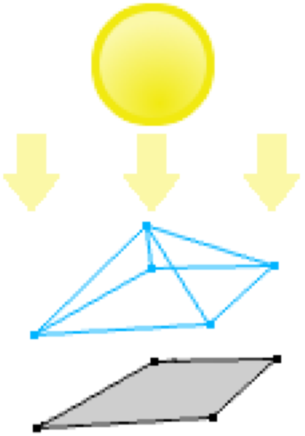
- A shape is considered convex if, for any line drawn through the shape, that line crosses only twice
- If a line can be drawn through the shape and cross more than twice the shape is non-convex





Projection

- The next concept that SAT uses is projection
- Imagine that a light source whose rays are all parallel
- If we shine that light at an object it will create a shadow on a surface
- A shadow is a two dimensional projection of a three dimensional object
- The projection of a two dimensional object is a one dimensional "shadow"

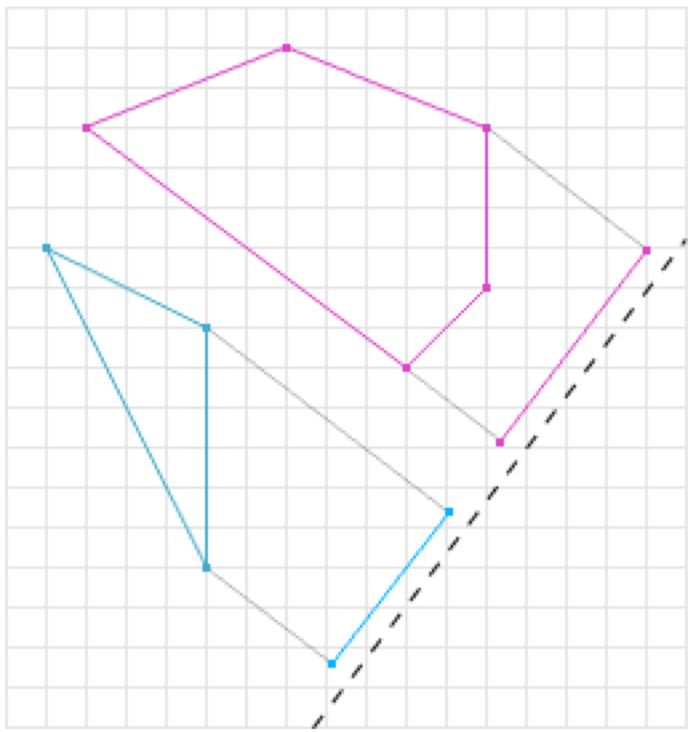


Algorithm

- “If two convex objects are not penetrating, there exists an axis for which the projection of the objects will not overlap.”

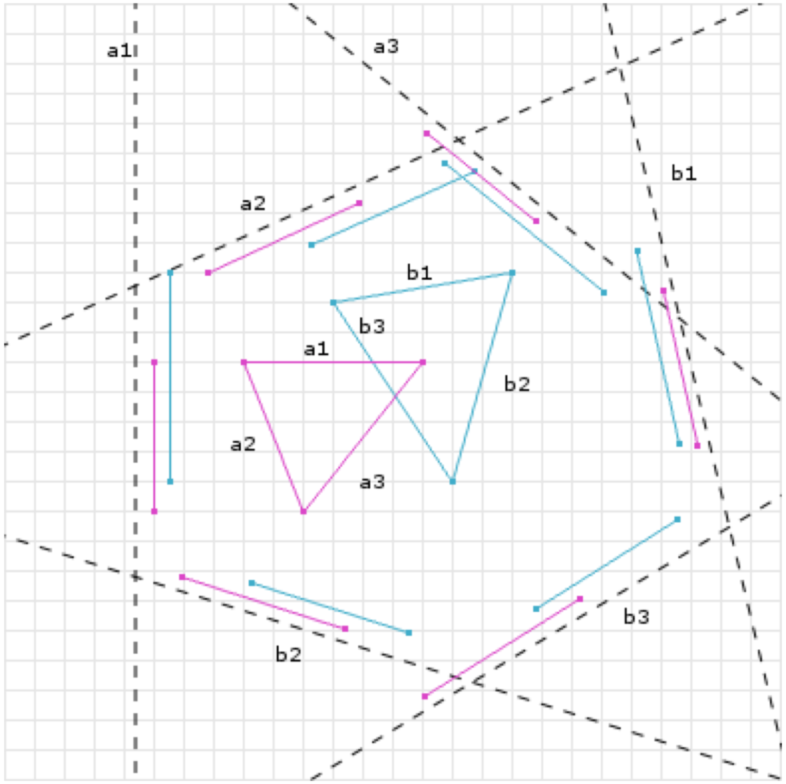
Example – No Intersection

- The dark grey line is a separation axis and the respective coloured lines are the projections of the shapes onto the separation axis
- The projections are not overlapping, therefore according to SAT the shapes are not intersecting
- SAT may test many axes for overlap, however, the first axis where the projections are not overlapping, the algorithm can immediately exit determining that the shapes are not intersecting



Example - Intersection

- If, for all axes, the shape's projections overlap, then we can conclude that the shapes are intersecting
- This figure illustrates two convex shapes being tested on a number of axes
- The projections of the shapes onto those axes all overlap, therefore we can conclude that the shapes are intersecting



Bounding Volumes

LECTURE 09: COLLISION DETECTION AND RESPONSE

Bounding Volume

- Using bounding volume for collision detection is very common in real-time simulations or computer games
- It bounds an odd shape 3D model with a simple volume, such as sphere or box
- Collision detection between two spheres or boxes is much faster than two 3D models that contains thousands of triangles

Sphere as Bounding Volume

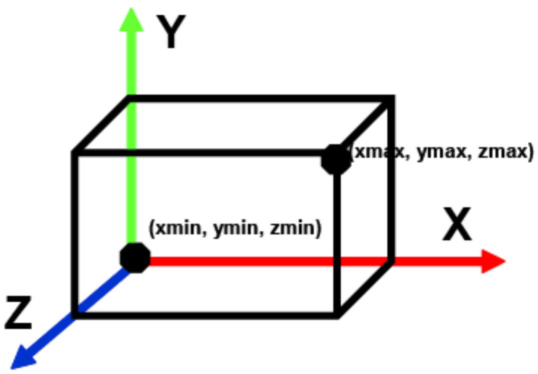
- Simplest 3D bounding volume
- Contain only a centre point and a radius
- Sphere to sphere test
  - Calculate the distance between two centres
  - If the distance is smaller than the sum of two radii, they collide
  - Otherwise, they don't
- It is always worthwhile to do sphere test before any more complicated test

Axis-Aligned Bounding Box (AABB)

- Specified as two points

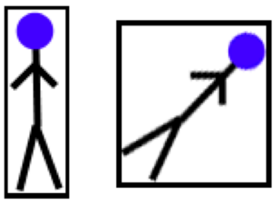
$$B_{\min} = (x_{\min}, y_{\min}, z_{\min}), B_{\max} = (x_{\max}, y_{\max}, z_{\max})$$

- AABB to AABB test
- Two AABB collide iff their orthogonal projections overlap on all dimensions

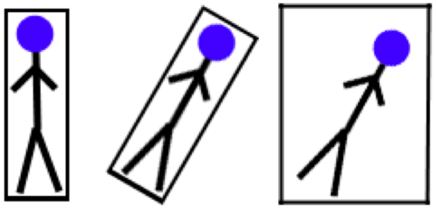


Problems with AABB

- Not very efficient (compact)
  - In some cases

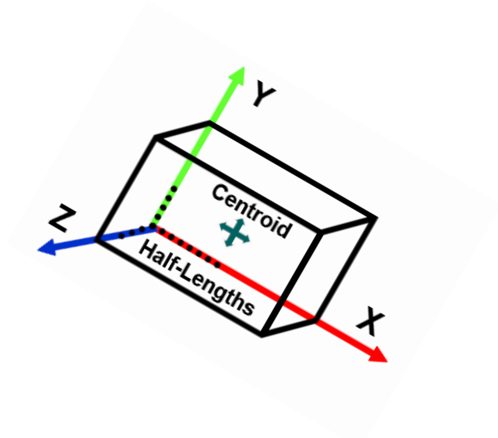


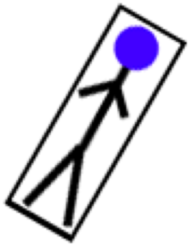
- Rotation can be complicated
  - Rotate model and rebuild AABB



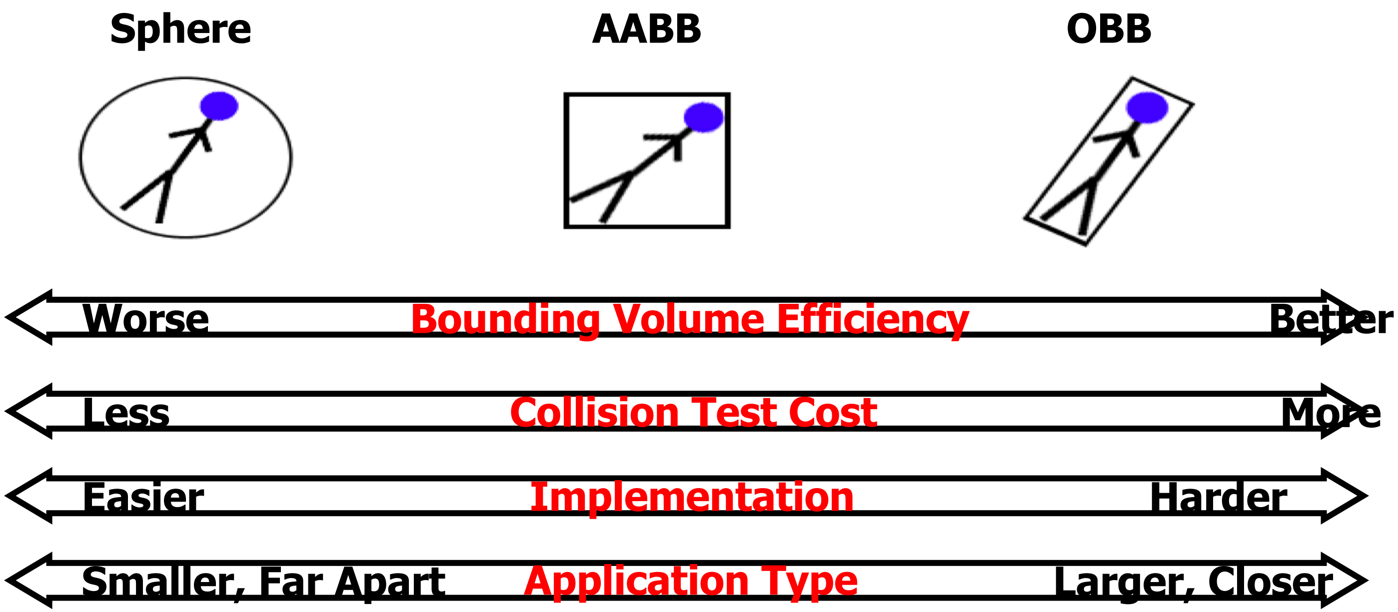
Oriented Bounding Box (OBB)

- Center point, 3 normalized axes, 3 edge half-lengths
- Can store as 8 points, sometimes more efficient
  - But can become not-a-box after transformations due to rounding errors
- Axes are 3 face normals
- Better at bounding objects than sphere and AABB

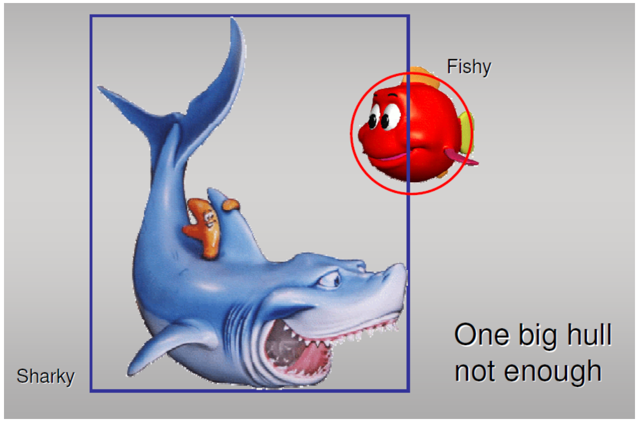


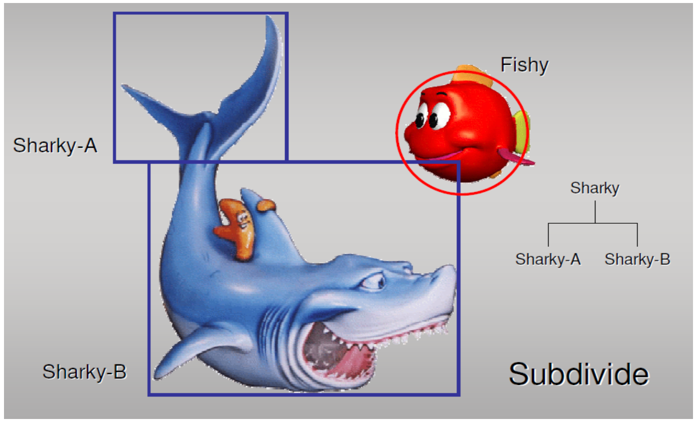


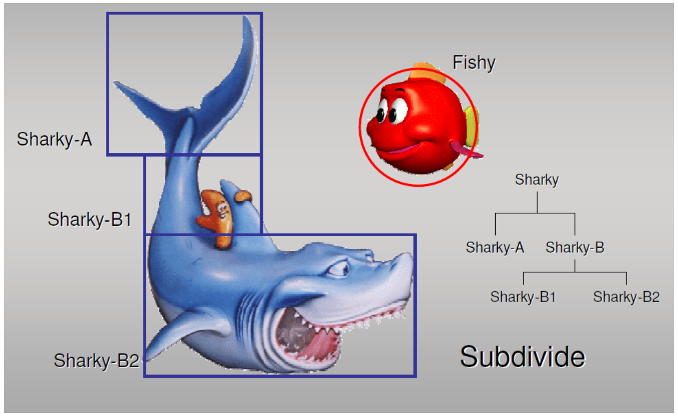
Bounding Volume Comparison



Bounding Volume Hierarchies

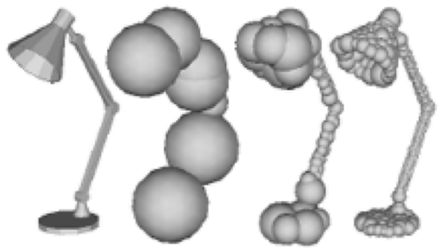


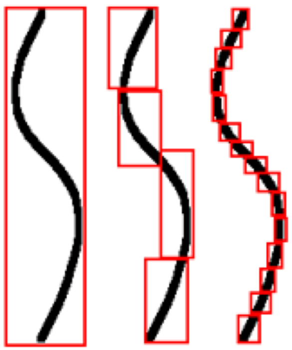




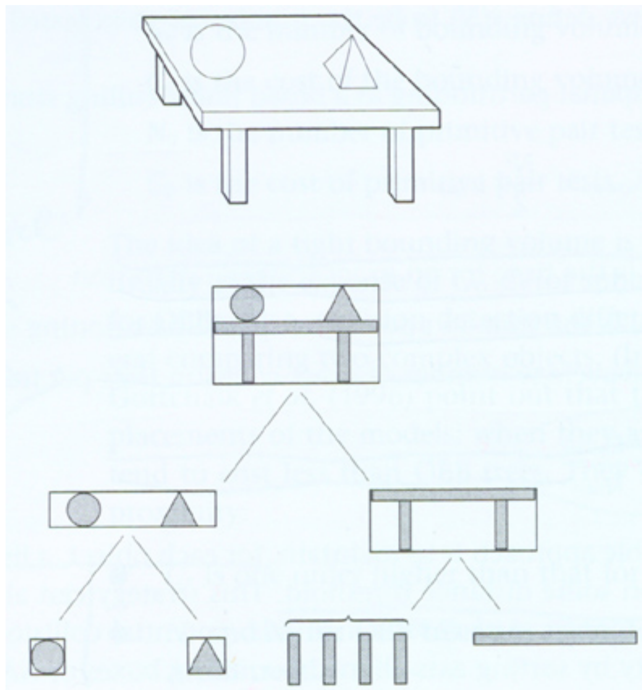
Bounding Volume Hierarchies (cont.)

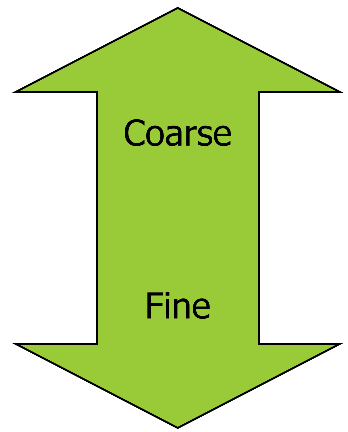
- Represent different levels of detail and accuracy
- Keeps number of comparisons small
  - Makes coarse comparisons first
- Trees are built automatically
  - Usually pre-computed, fitting is expensive





AABB Tree Example





Hierarchies must be updated for moving objects

N-Body Problem

LECTURE 09: COLLISION DETECTION AND RESPONSE

The N-Body Problem

- Pairwise collision detection is fast but what about a scene of N objects?
- Number of overlap tests for ${}_nC_2$ objects pairs = N(N-1)/2
- Brute-force pairwise collision detection becomes an $O(N^2)$ problem

The Bottleneck

- Bounding volume (e.g. sphere) is fast but inaccurate
- More accurate pairwise collision test requires more computational overhead
- An accurate method would be inefficient for $O(N^2)$ pairs

Solution: Two-Phase Collision Detection Approach

- Basic idea: Cull out object pairs that are far apart
- Leave pairwise comparison to a small number of object pairs
- Reduce the $O(N^2)$ complexity
- Examples: Spatial Decomposition, Sweep and Prune Algorithm

Spatial Decomposition

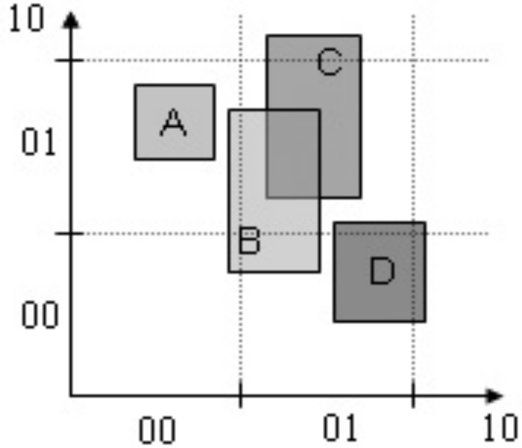
LECTURE 09: COLLISION DETECTION AND RESPONSE

Spatial Decomposition Methods

- Decompose the space into regions (or “cells”)
- Pairwise comparisons for each cell
- First Phase Complexity: dependent on the method, e.g. O(N) for spatial hashing
- Second Phase Complexity: $O(N_r^2)$ for N_r pairs in each cell
- Ideally, most of the cells contain only one object

Example – Spatial Hashing in 2D

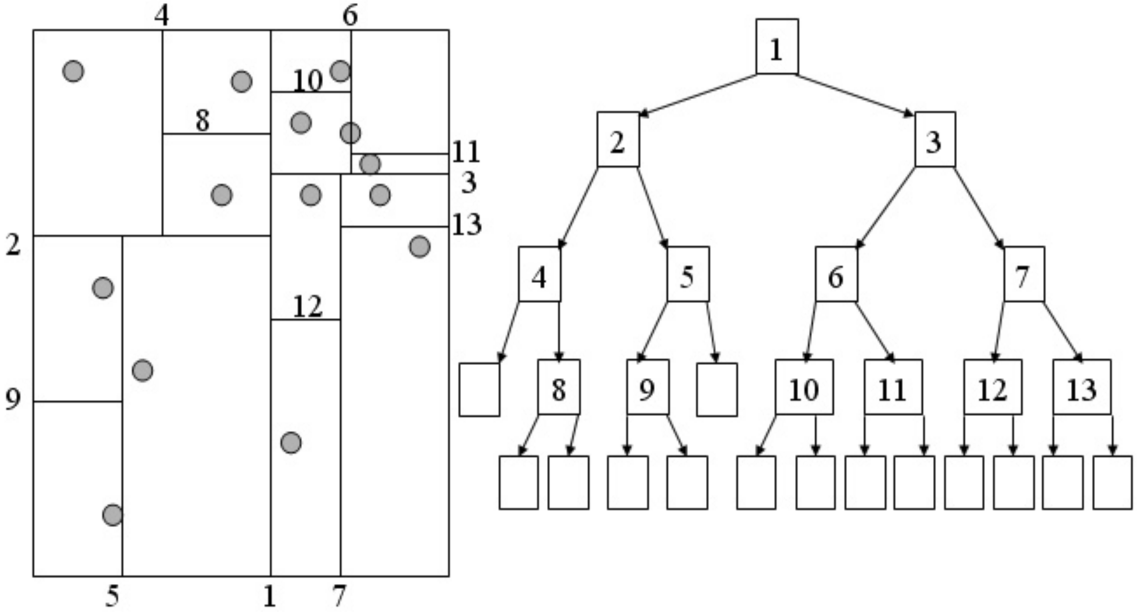
- Object A is hashed into 0100
- Object B is hashed into 0100, 0000, 0101 and 0001
- Object C is hashed into 1001, 0101
- Object D is hashed into 0101, 0110, 0001, 0010
- A-B, B-C, B-D, C-D are all potential collisions



Properties of Spatial Hashing

- Collision query for one object can be processed in O(1)
- Crowded subdivisions => number of potentially collided pairs becomes large
- What is the worst case?
  - All objects reside in a single subdivision
- What is the time complexity for worst case?
  - O (N^2)

Example: Hierarchical Structure (K-D Tree) in 2D



Properties of Hierarchical Structures

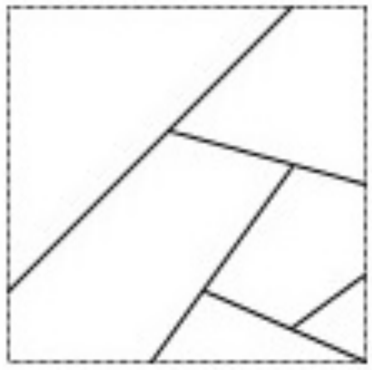
- A collision query is O(log N) in the average case
- What is the average case?
  - A balanced tree
- What is the time complexity of collision queries for N objects in the average case
  - O (N log N)
- In the worst case, collision queries for N objects could become O (N^2)
- What is the worst case?
  - The height of the tree = the number of objects

Reconstruction of the Tree

- Objects move from time to time
- Reconstruction of the Tree is required
- Insert an object into the tree takes O(log N) time in the average case
- Total reconstruction might happen, which takes O(N log N) time in the average case
- Hierarchical structures are efficient for largely static scenes, but inefficient for scenes with multiple fast moving objects

Binary Space Partitioning (BSP) Tree

- BSP tree recursively partitions a space into two subdivisions using a partition plane
- Partition plane can be free chosen
  - Easy to keep the tree balanced
- Partition query is relatively complicated



BSP Tree Construction

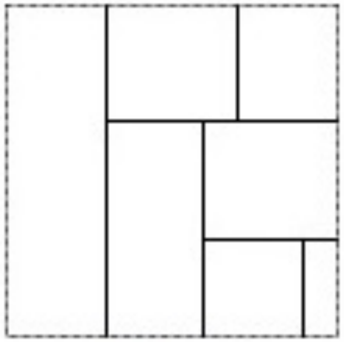
- Step 1: Choose a plane to partition the virtual space into two subdivision, each contains equal number of objects
- Step 2: Recursively partition each of the subdivisions into two, until the subdivision contains only one object
- There are other ways to construct the BSP tree, counting the number of objects is just one of the approaches.

BSP Tree collision query for one object

- Step 1: Traverse the BSP tree from its root node
- Step 2: Check whether the space of left subtree contains this object. If this is the case, traverse recursively the left subtree
- Step 3: Check whether the space of right subtree contains this object. If this is the case, traverse recursively the right subtree
- Step 4: When a leaf node is reached, check whether the objects it contains collide with the object in question

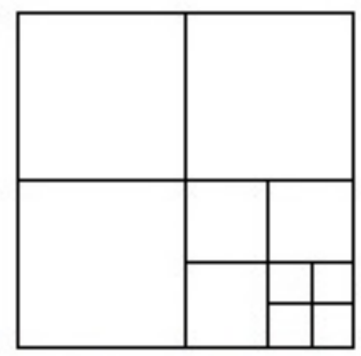
K-Dimensional (K-D) Tree

- 2-D Tree in this case
- K-d tree recursively partitions a space into two subdivisions using a partition plane
- Partition planes must be axis-aligned
    - More difficult to keep the tree balanced than the BSP tree
- Collision queries are simpler than BSP tree



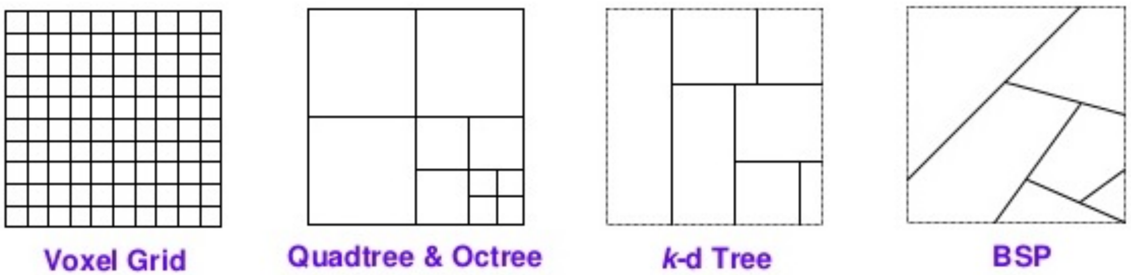
Quadtree (2D) or Octree (3D)

- Quadtree in this case
- A quadtree recursively partitions a space into four uniform subdivisions using two axis-aligned partition planes
- An octree recursively partitions a space into eight uniform subdivisions using three axis-aligned partition planes
- More difficult to keep the tree balanced than BSP tree and k-d tree
- Collision queries are simpler than BSP tree and k-d tree



Hierarchical Structures

- K-D tree is a subset of BSP tree
- Quadtree (or octree) is a subset of K-D tree
- Uniform grid (a special case) is a subset of quadtree (or octree)



Granularity Problem

- Difficult to choose optimal cell size (or tree height)
- Large cell size (or small tree height) =>
  - Many objects reside in a cell
  - More pairwise comparisons
- Small cell size (or large tree height) =>
  - More cells require collision detection
- Essentially, this becomes a trade-off problem

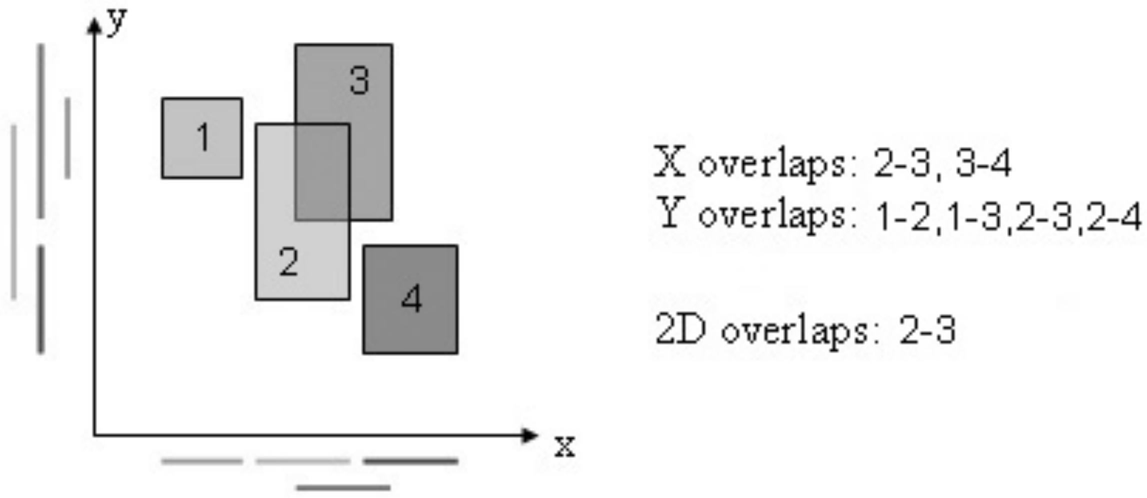
Sweep and Prune Algorithm

LECTURE 09: COLLISION DETECTION AND RESPONSE

Sweep and Prune Algorithm

- Introduced by the iCollide system in the 90's
- Work with Axis-Aligned Bounding Boxes (AABBs) and uses dimension reduction.
- Project extrema of AABBs onto each coordinate axis
- Sort the projections for each axis
- When temporal and geometric coherence is exploited, sorting process would be extremely fast
- First phase complexity: O(N) for N objects
- Second phase complexity: O(S) for S pairs whose AABBs overlap.
- It is perhaps the fastest collision detection method for the N-Body

Dimension Reduction in 2D



Two AABBs overlap if and only if their orthogonal projections overlap in all dimensions

Temporal and Geometric Coherence

- Objects do not move swiftly between time steps
- The order of projections (for AABB extrema) only has little change for each frame
- Insertion sort or bubble sort for the extrema can run in O(N) time

Sweep and Prune Algorithm - Problem

- The sorting process would become time consuming if temporal and geometric coherence is not preserved
- Example: objects are bouncing
- Fail to preserve the coherence may lead to the $O(N^2)$ worst case run time complexity

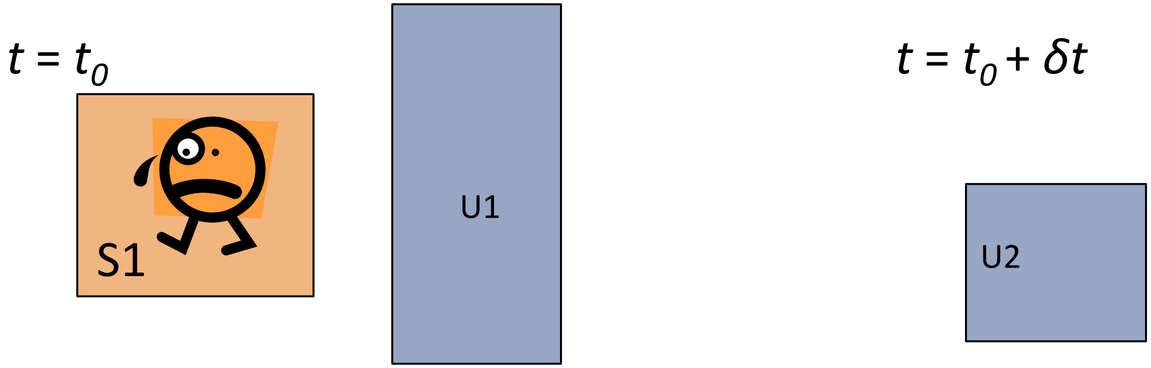
Continuous Collision Detection

LECTURE 09: COLLISION DETECTION AND RESPONSE

The Mission Collision Problem

- So far we have talked about collision detection in discrete time-steps
- An object moves across another object without notice
  - Collision detection fails to report some of the events
  - Players perform incorrect actions
- This occurs when:
  - Objects move at a high speed
  - Bounding volumes are small
  - Time-step is large

The Missing Collision Problem – Example



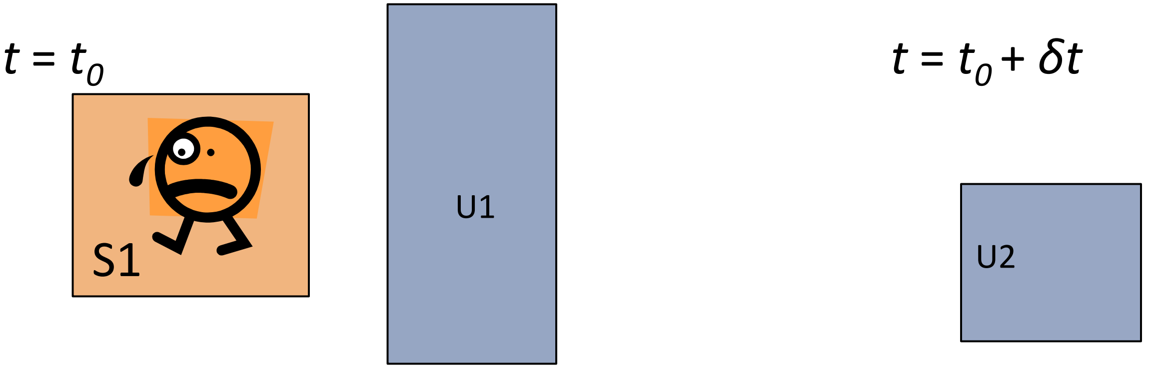
The Missing Collision Problem - Video

- https://www.youtube.com/watch?v=cw7BVTa6Xmw

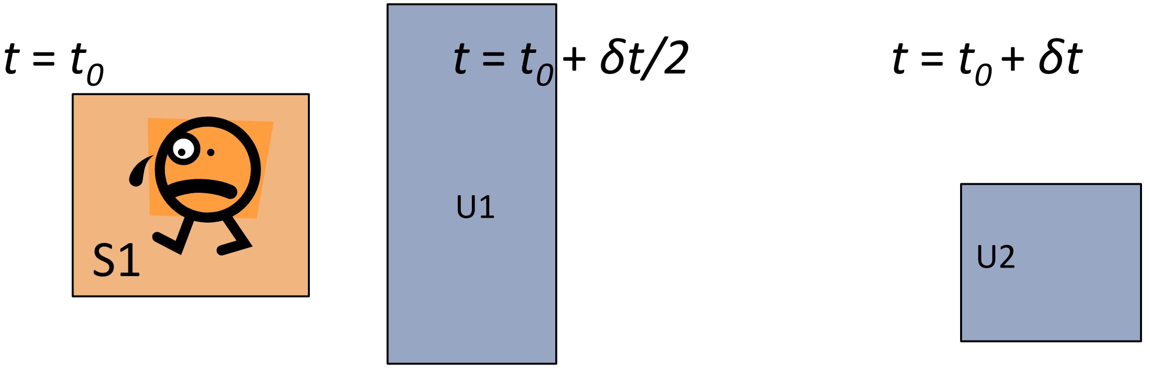
Missing Collision Problem – Simple Solutions

- Enlarge bound volumes
  - Advantage: capture more missing collisions
  - Disadvantage: may generate irrelevant collisions
- Frequent collision detection
  - Reduce time-step of simulation and carry out extra overlap tests
  - Advantage: capture more missing collisions
  - Disadvantage: additional computational overhead

Enlarge Bounding Volumes



Frequent Collision Detection



Continuous Collision Detection

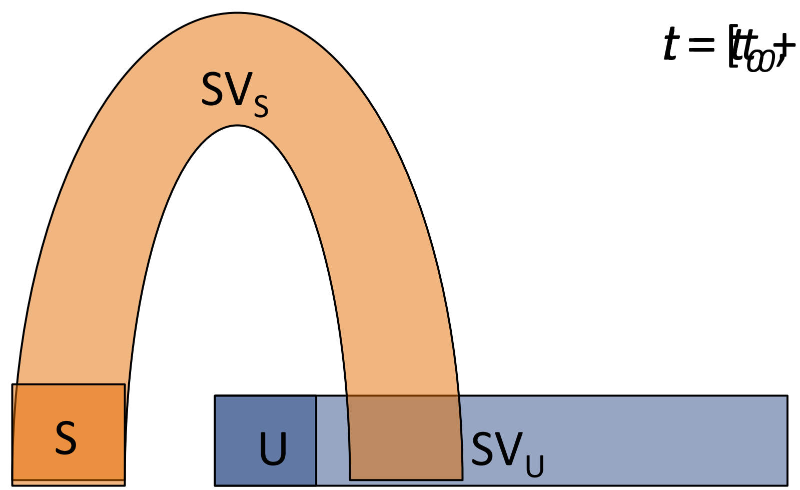
- Advantages
  - Does not increase bounding volume size => Preserve precision
  - Does not perform extra collision detections for all objects => Reduce computational overhead
- Example: use of swept volume

Swept Volume

- A bounding volume that bounds the motion of an object along an arbitrary path over a time interval
- Use of swept volume has been studied extensively in
  - Robotics
  - Computer graphics
  - Interest management (what is this?)

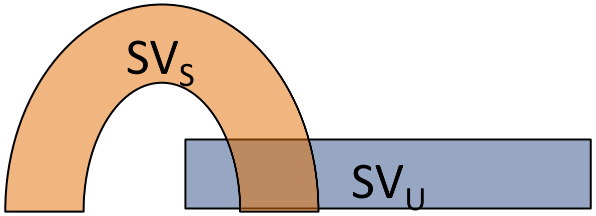
Swept Volume (Example)

$t = [t_0, t_0 + \delta t]$



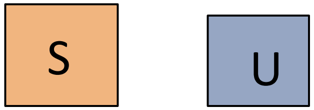
Overlap Determination

- Two swept volumes, $\boldsymbol{SV_U}$ and $\boldsymbol{SV_S}$, overlap iff there exists a common point that lies within both of them
- i.e. $\boldsymbol{SV_U} \cap \boldsymbol{SV_S} \neq \emptyset$
- However, it is not sufficient to say that the two objects actually overlap each other at a certain time



Pairwise Space-Time Overlap Test

- Two objects overlap within [*a*,*b*] if and only if there exists a common point that lies within the areas occupied by the two objects at a certain time $t \in [a,b]$



Divide-and-Conquer Algorithm

- The algorithm first tests $\boldsymbol{SV_U}$ and $\boldsymbol{SV_S}$ for an overlap
- If an overlap occurs, it then splits the time interval $[a, b]$ into two equal subintervals and splits each of the swept volumes into two smaller ones accordingly
- This process continues recursively until no overlap occurs or the tested subinterval is smaller than a user defined threshold

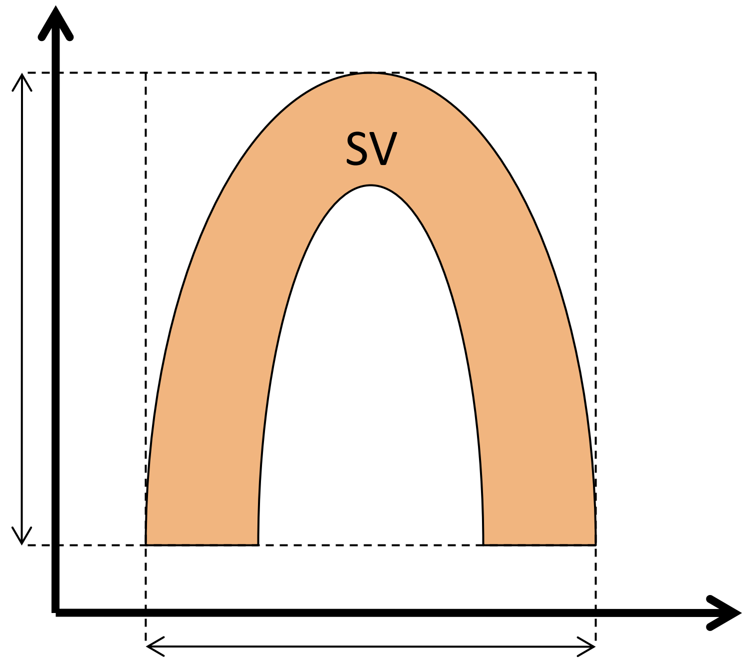
Divide-and-Conquer Algorithm (Example)

$I = [(b-(b-a))/2, b]$

$SV_S(I)$

$SV_U(I)$ $\vee$ $SV_U(I)$

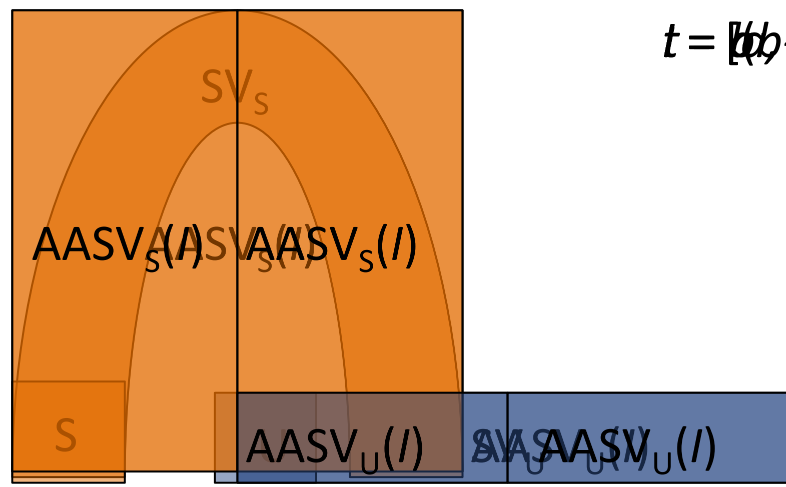
Orthogonal Projection for Swept Volume



Integrate SV with Dimension Reduction

- Use of Axis-Aligned Swept Volume (AASV)
- AASV is formed by the orthogonal projections of SV
- Properties
  - A loose bound of the actual SV
  - Faster Computation
  - Can be integrated with the dimension reduction approach

Pairwise Overlap Test with AASVs (Example)



More on Collision Detection – Deformable Objects

- So far we have talked about rigid bodies
- What about cloths?
- Collision detection for deformable objects and self collision detection are huge topics
- Let’s watch this video
  - https://www.youtube.com/watch?v=Rl6GI0YtwpE
- It is very difficult to be done in real time
- Therefore, most of the computer games don’t use it