Collision Detection

Overview

- Collision detection
  - Coarse collision detection
    - Bounding volumes
    - Hierarchies
    - Spatial data structures
  - Fine collision detection
    - Contact generation
  - Collision queries

Collision Detection

- Collisions
  - ➢ Simulating collisions consists of
    - Collision detection (programming problem)
      - To detect whether a collision has occurred
    - Collision response (physics problem)
      - How objects react when they collide
  - ➢ Games without physics still use collision
    - For example
      - Ray casts: select objects, shoot at things, distance to the ground
      - To move interpenetrating objects apart
      - To query the surroundings, e.g., explosions, entities within range
      - Triggers: pick up items, trigger events, activate AI

Collision Detection

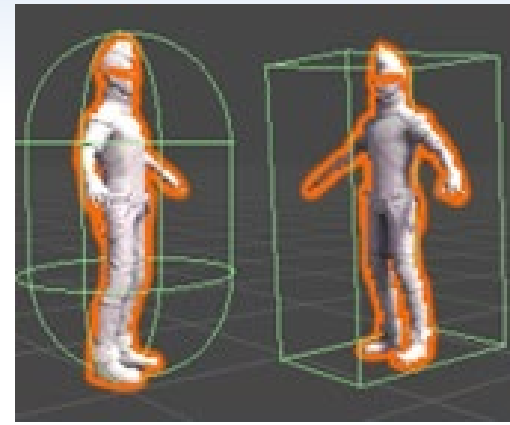
- Collision detection system
  - Determining whether objects come into *contact*
    - More than just true/false
    - Information about the nature of each contact
  - Responsible for finding
    - Pairs of colliding objects
    - Single objects that are colliding with immovable objects
  - Needs to take geometry into account
    - Geometric shapes can be simple or complex
    - Collision representation can differ from visual representation

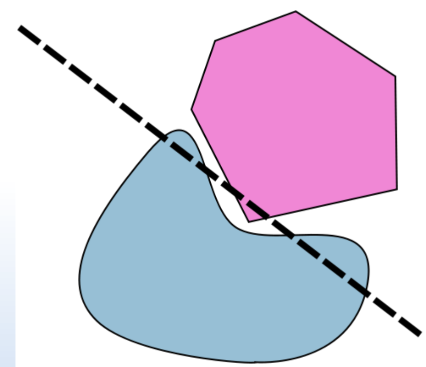
Collision Detection

- Unity
  - `Collider`
    - Base class for all colliders, e.g.,
      - `BoxCollider`, `SphereCollider`, `CapsuleCollider`, `MeshCollider`
    - If an object with a Collider needs to be moved during gameplay, should attach a Rigidbody component
      - The Rigidbody can be set to be kinematic if you don't want the object to have physical interaction with other objects
    - Some properties and methods
      - `isTrigger`: specifies if the collider used as a trigger
      - `OnCollisionEnter`: called when this collider touches another collider

Collision Detection

- Unity
  - ➢ Collision detection often not carried out directly on meshes
  - ➢ Primitive colliders
    - Box, sphere and capsule
      - – Relatively simple collision detection algorithms
  - ➢ Mesh collider
    - Very computationally expensive
    - Only convex mesh colliders supported
    - Collision between two mesh colliders may not always be detected





http://en.wikipedia.org/wiki/Separating_axis_theorem

Collision Detection

- Complex problem
  - Can be very time-consuming
    - Each object in the game may be colliding with any other object
    - Each pair has to be checked
    - Each object may have different geometry
    - Number of possible collisions:

$$N_c = \frac{n!}{2(n-2)!}$$

Number of objects	Number of possible collisions
2	1
4	6
10	45
100	4950
1000	499,500
10000	49,995,000

Collision Detection

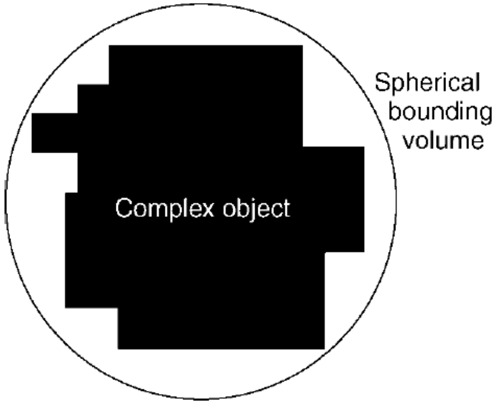
- Computational complexity
  - Key problems
    - Too many possible collisions
    - Expensive checks
  - To reduce the number of checks
    - Phase 1: Coarse collision detection (broad phase)
      - Find sets of objects *likely* to be in contact
    - Phase 2: Fine collision detection (narrow phase)
      - Check whether candidate collisions (from broad phase) are actually in contact
      - If so, obtain contact information (contact generation)

Collision Detection

- Coarse collision detection
  - Key features
    - As fast as possible
    - Should be conservative
      - Allowed to generate checks that end up not being actual collisions (‘false positives’ to be discarded in phase 2)
    - At the same time, should minimise number of false positives to reduce computational overheads in phase 2
  - Common approaches
    - Bounding volume and hierarchies
    - Spatial data structures
      - Same structure can be used elsewhere in the game engine, e.g., in the rendering systems

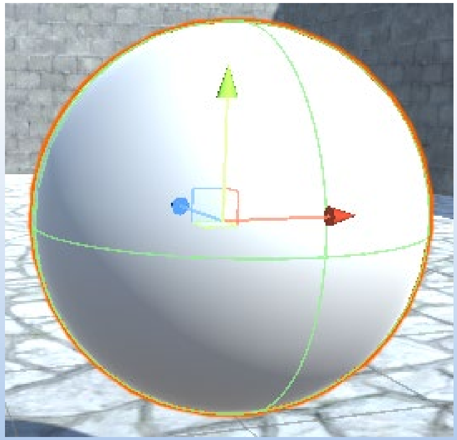
Bounding Volumes

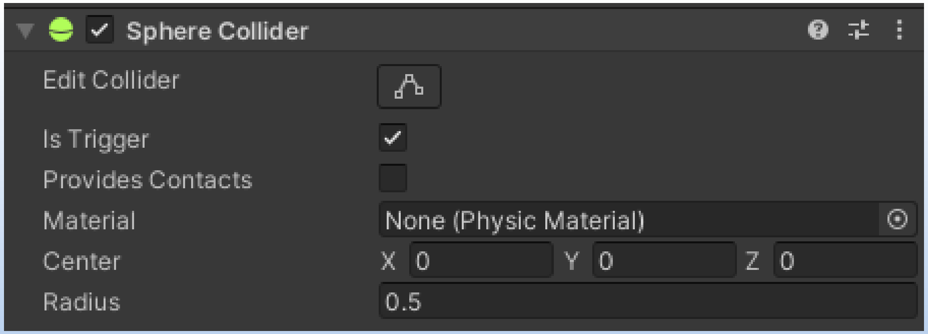
- Bounding volumes
  - ➢ An area of space known to contain the object
  - ➢ Large enough for the object to be inside
    - Ideally, as close fitting as possible
  - ➢ Simple shape used
    - Simplifies collision tests
    - Simplifies repositioning
    - Minimises data storage overheads
  - ➢ Concept
    - If the bounding volumes of two objects do not touch
      - – The objects inside them cannot be in contact



Bounding Volumes

- Bounding sphere
  - ➢ Easy to represent, only needs
    - Centre and radius
  - ➢ Invariant under rotation
    - When moving, only its position needs to be updated
  - ➢ Easy to check whether two spheres overlap
  - ➢ Unity: `SphereCollider`



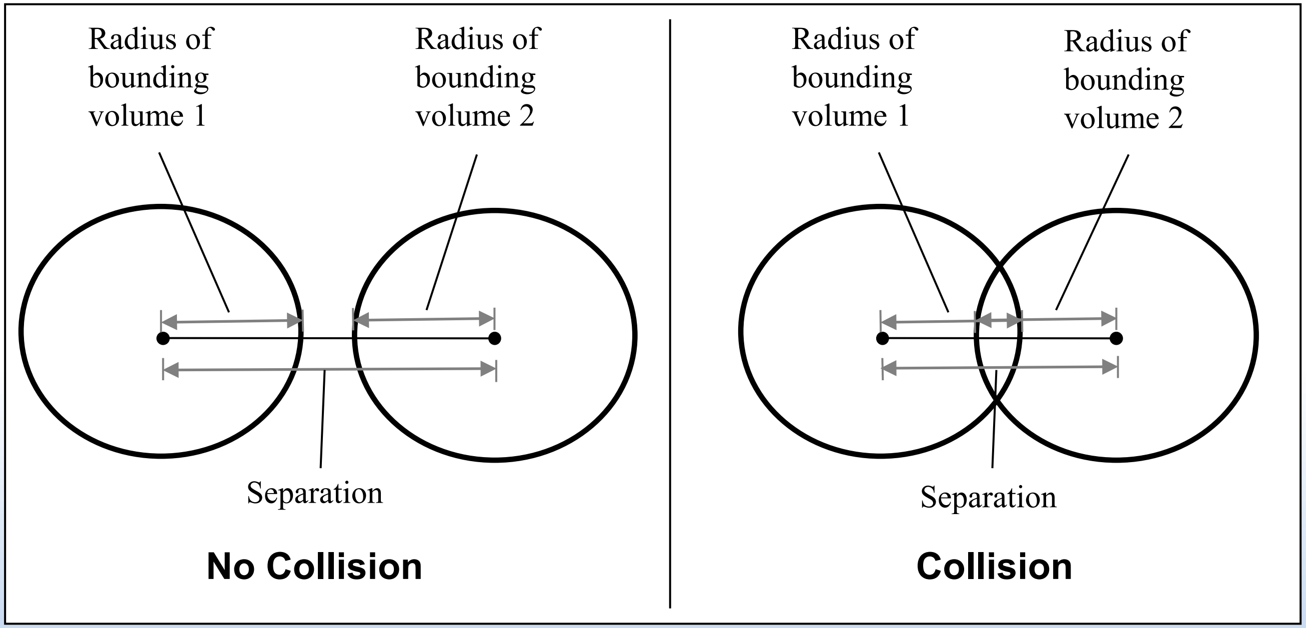


Bounding Volumes

➢ Simple collision check

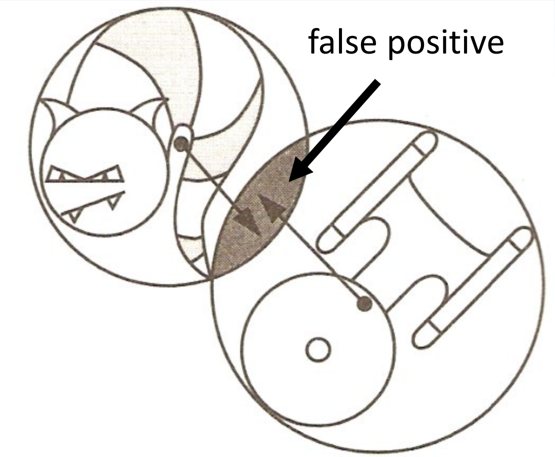
- Overlap if distances between centres less than sum of radii

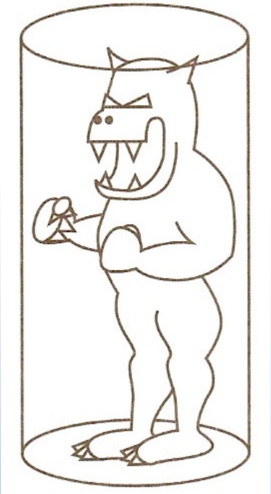
```
if(separationDistance < radius1 + radius2)
```



Bounding Volumes

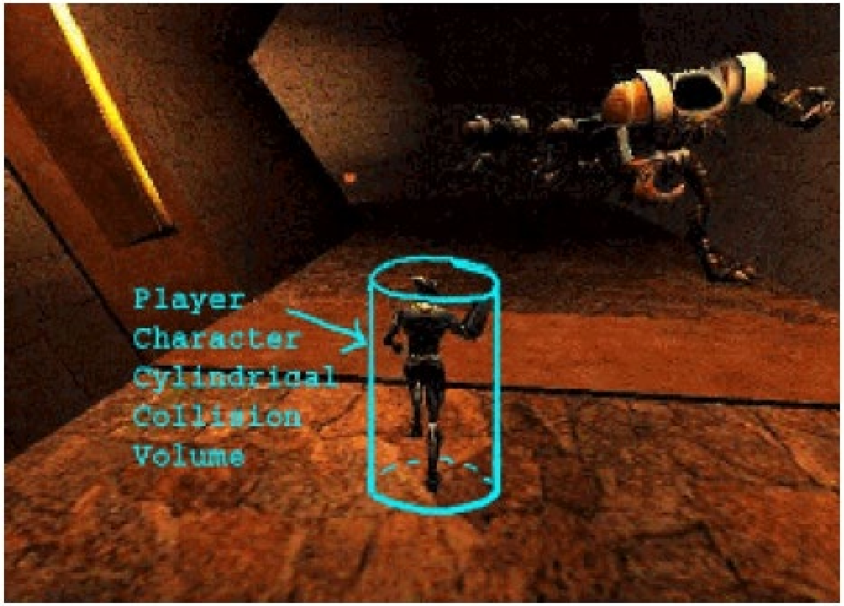
- Bounding cylinders
  - ➢ Problem with spheres
    - Good for objects that spread out in a number of directions
    - But not good for tall and thin objects
  - ➢ Work well for games where most objects orientated about the same way toward some surface





Bounding Volumes

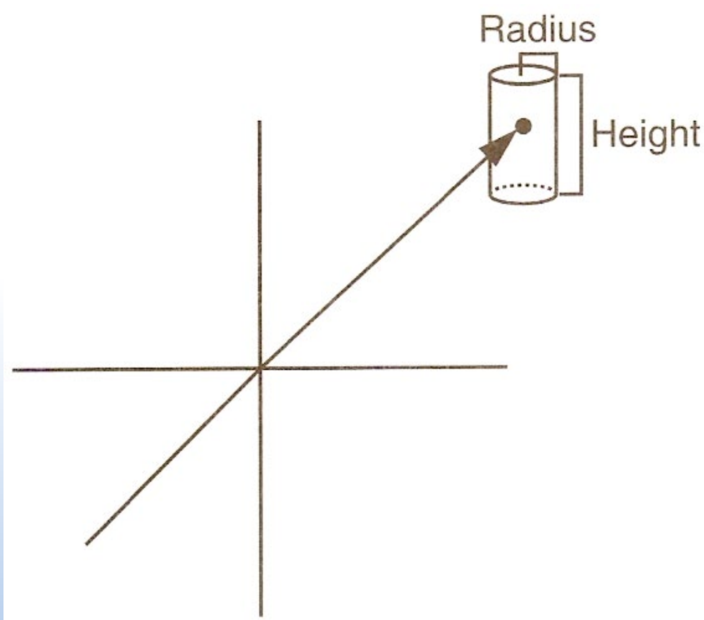
- In many games
  - Characters stay orientated in same direction with respect to the floor
- If always aligned to floor
  - When moving only need to update position
  - No need to rotate

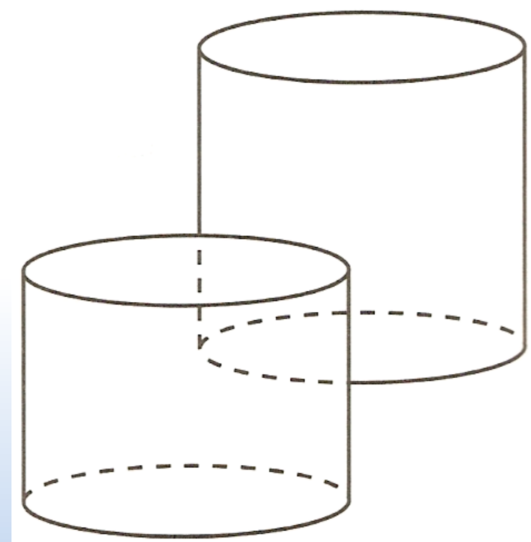


In *MDK2* we used a cylinder model for the character-to-environment collision detection.

Bounding Volumes

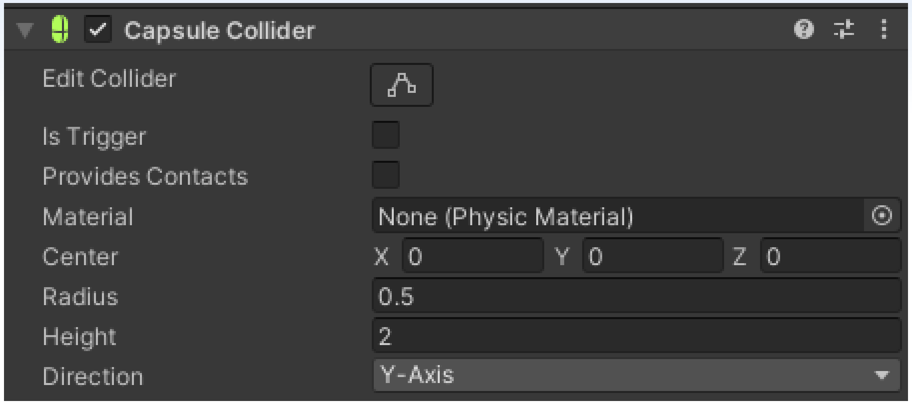
- Collision check
  - Two-steps process
    - Check radius (the same method as spheres)
    - If the radii overlap
      - Check whether heights of the cylinders overlap

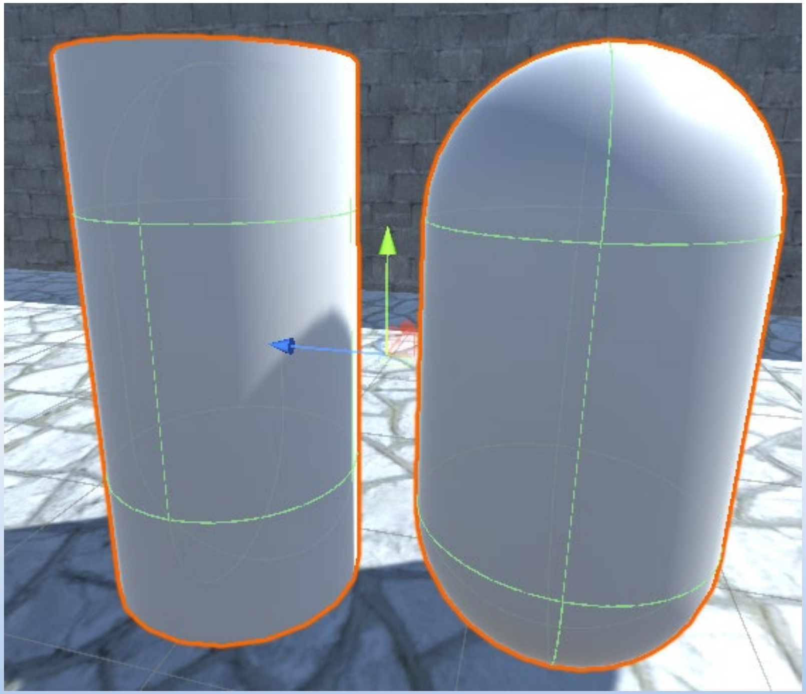


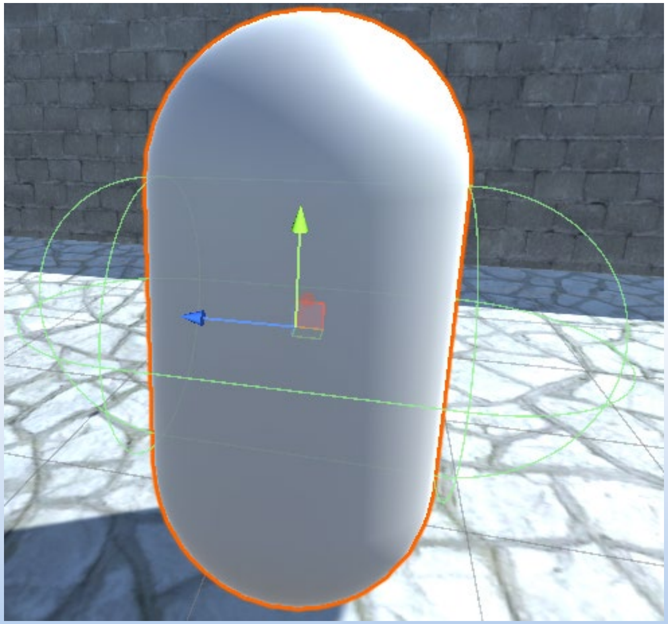


Bounding Volumes

- Unity
  - ➢ `CapsuleCollider`
    - Used for cylinders as well
    - Can set Direction







Bounding Volumes

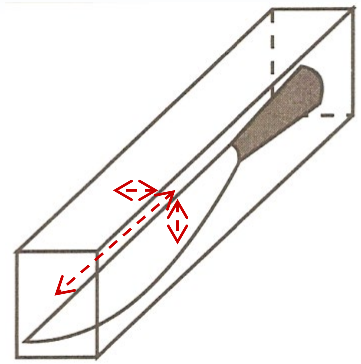
- Bounding boxes
  - ➢ Good for rectangular objects
  - ➢ Can be represented as
    - Central point and a set of dimensions, one for each direction
      - E.g., distance from the central point to its sides in x, y and z directions

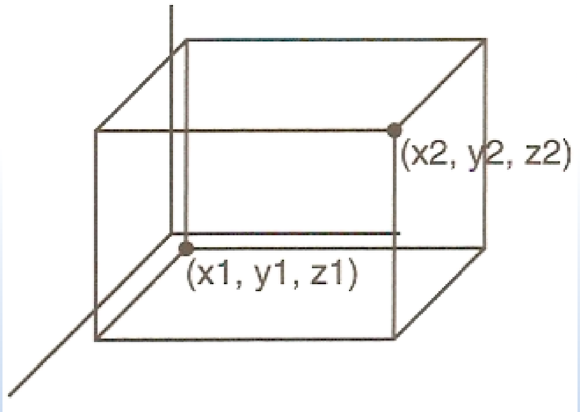
        (half the overall box size, 'half-size')

```
Vector3 centre;
Vector3 halfSize;
```

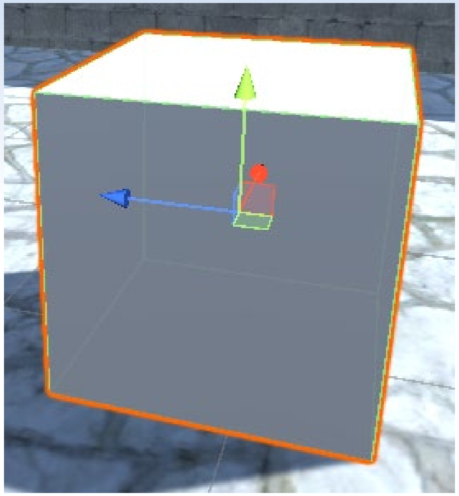
    - Other representations are also possible

```
Vector3 maxVertex;
Vector3 minVertex;
```

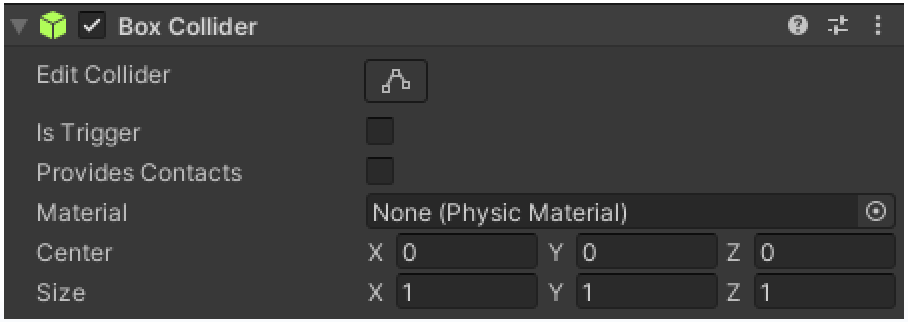


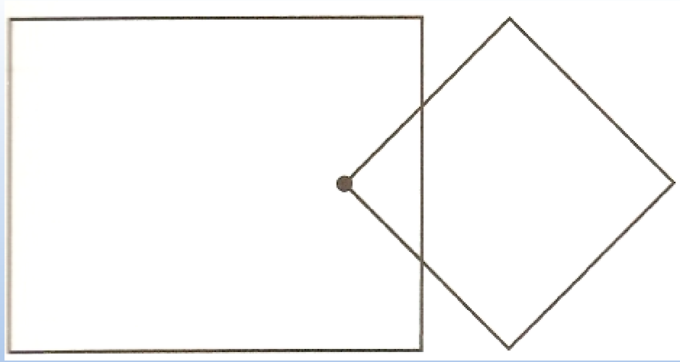


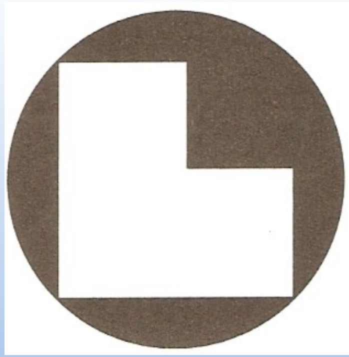
Bounding Volumes

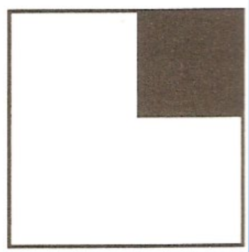


- Collision check
  - More complex than spheres or cylinders
  - Two bounding boxes are colliding if the vertex of one is inside the other
  - Requires more computation compared to spheres or cylinders
- Unity: `BoxCollider`



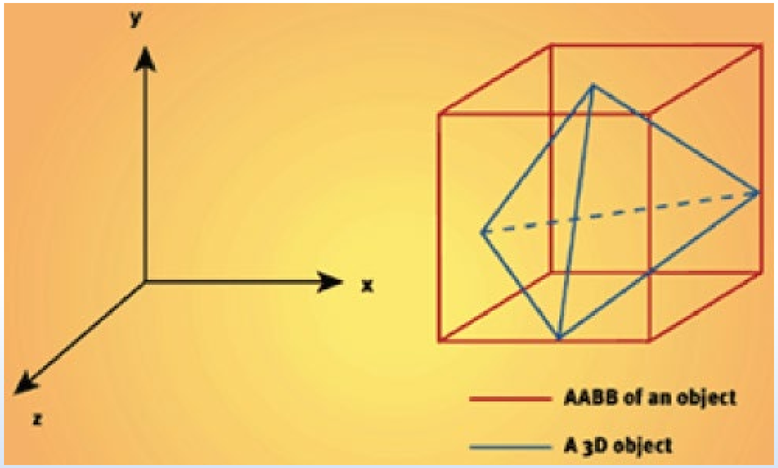






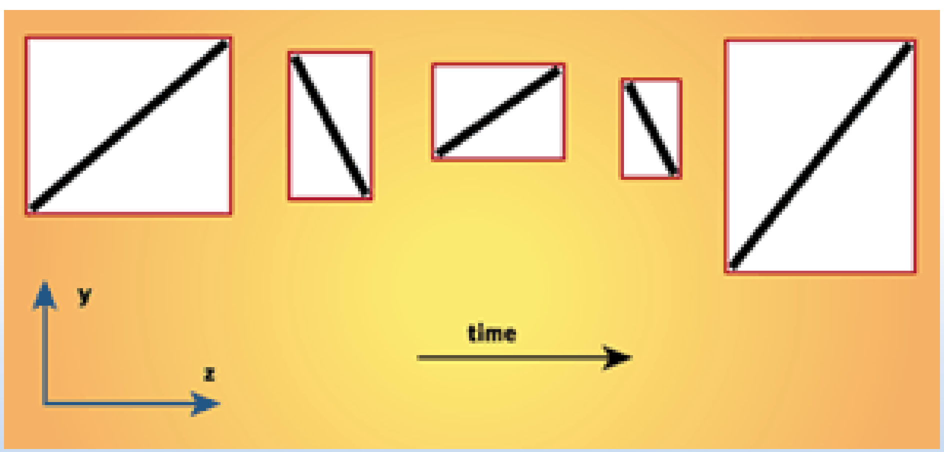
Bounding Volumes

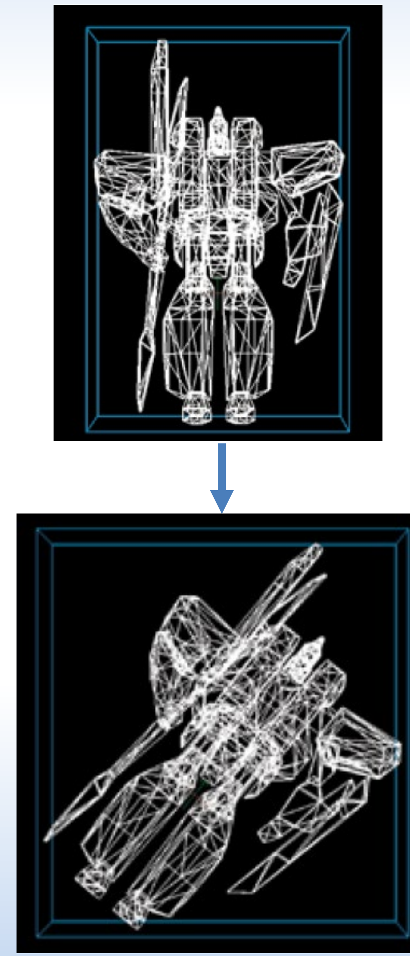
- Two types of bounding boxes in game engines
  - Axis-Aligned Bounding Boxes (AABBs)
  - Orientated Bounding Boxes (OBBs)
- AABBs
  - Aligned to world axes
  - Each box face perpendicular to one coordinate axis
  - AABBs do not rotate
  - Good for static objects that are aligned to axes and do not rotate



Bounding Volumes

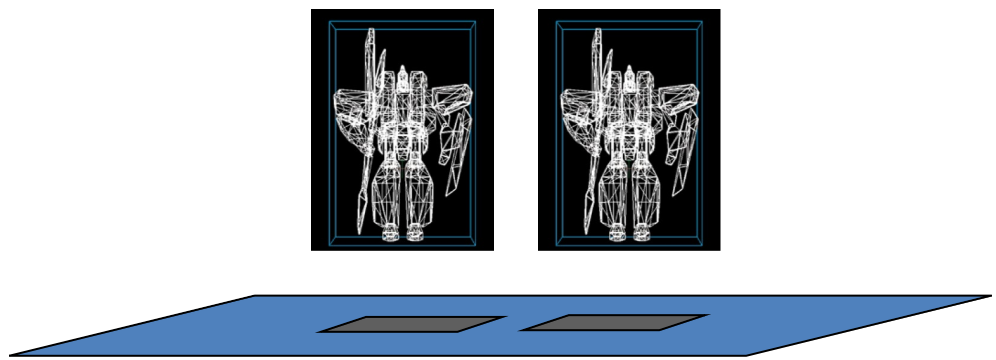
- When object rotates, need to re-compute bounds
- Assumption
  - Faster than rotating the bounding box, but more empty space, i.e. more false positives
  - Trade-off between speed and accuracy



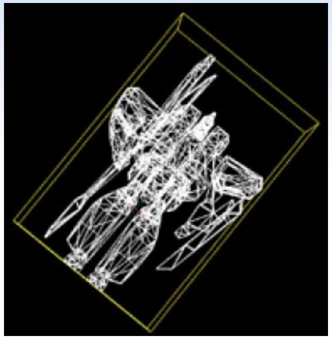


Bounding Volumes

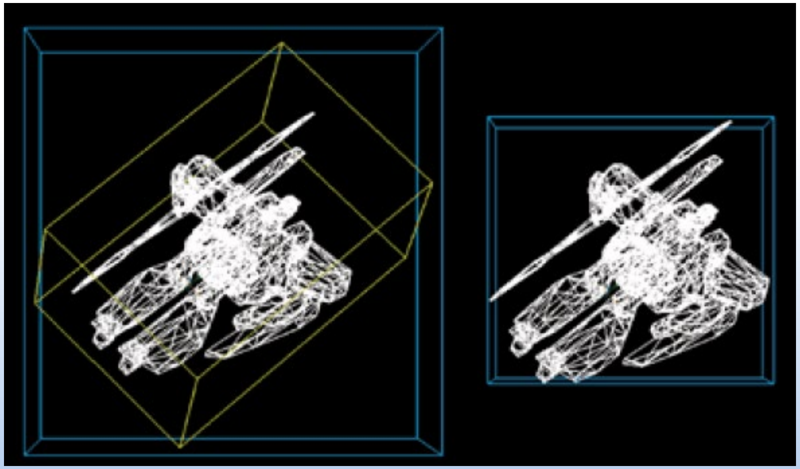
- ➢ Collision check
  - Two AABBs do not collide if their projections on a plane do not overlap
- Unity
  - ➢ `Bounds`
    - Represents an AABB
    - Some properties
      - `center`
      - `extents`: half sizes
      - `max`: center + extents
      - `min`: center - extents

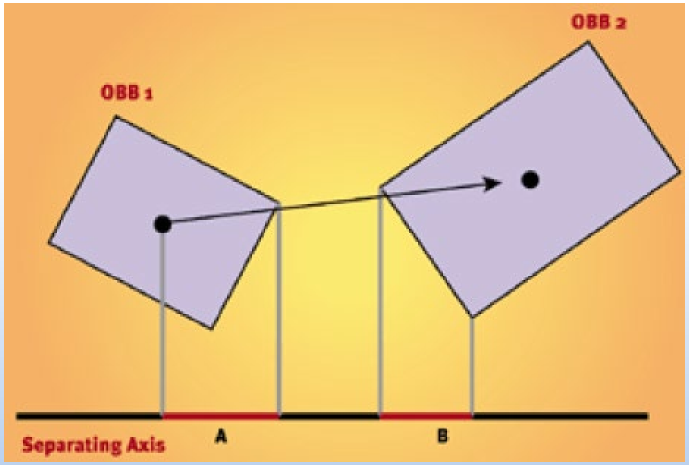


Bounding Volumes



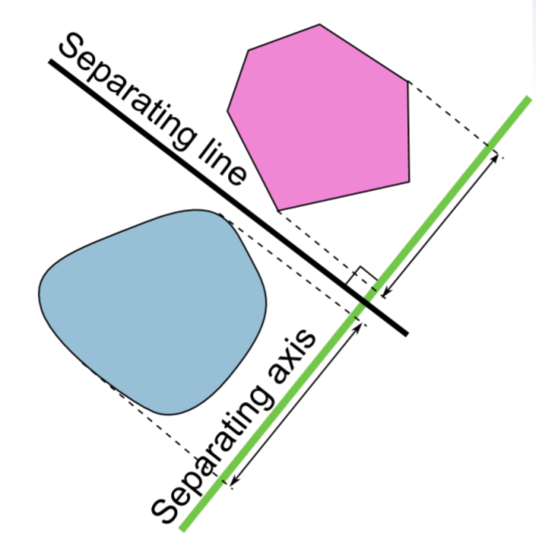
- OBB
  - ➢ Aligned to the object's coordinates
    - Rotates with the object
  - ➢ Less empty space, but more difficult to implement and slower
  - ➢ Can convert OBB to AABB, but not tight fitting

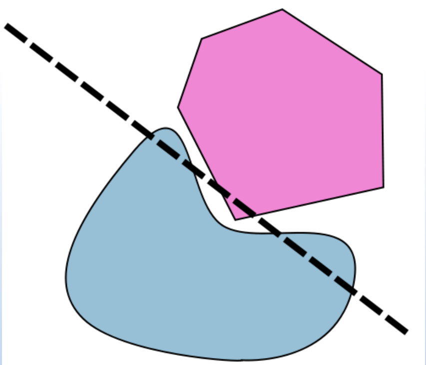




Bounding Volumes

- Separating axis theorem
  - ➢ Two ***convex*** shapes do not intersect
    - If an axis can be found along which the projection of the shapes do not overlap
    - If such an axis does not exist and shapes are convex then they intersect
  - ➢ Used in most collision detection systems
    - Only applies to convex shapes



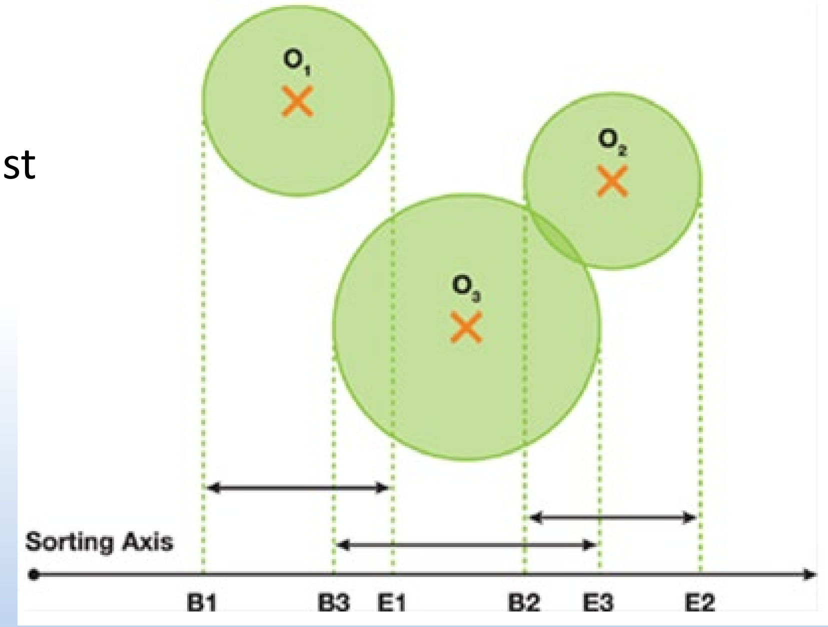


Not convex

http://en.wikipedia.org/wiki/Separating_axis_theorem

Bounding Volumes

- PhysX's sort and sweep algorithm
  - Sphere example
    - Each sphere is projected onto *x*, *y*, or *z* axis
    - Interval [b_i, e_i] for each projection of each object is added to a list
    - The list is sorted
    - If b_{i+1} is greater than e_i
      - Object *i* is removed from the list



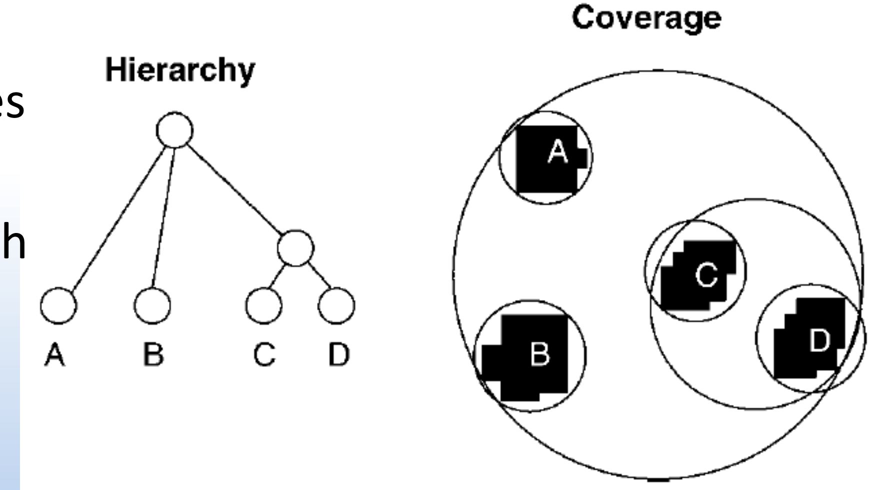
Scott Le Grand, Chapter 32: Broad-Phase Collision Detection, Nvidia Corporation

Bounding Volume Hierarchies

- Bounding volumes alone
  - Still involves checking every pair of objects

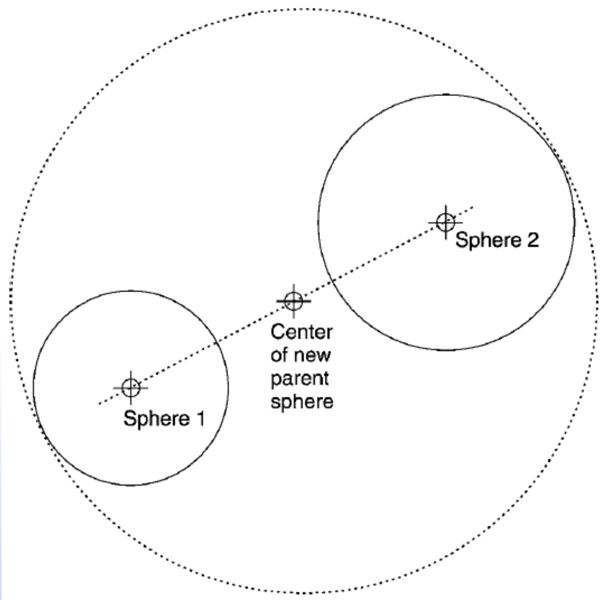
$$N_c = \frac{n!}{2(n-2)!}$$

- Bounding volume hierarchy (BVH)
  - Helps to avoid checking every object pair
  - Each object (in its bounding volume) is a leaf of a tree data structure
    - These connect to parent nodes
    - Each parent has its own bounding volume large enough to enclose all descendents



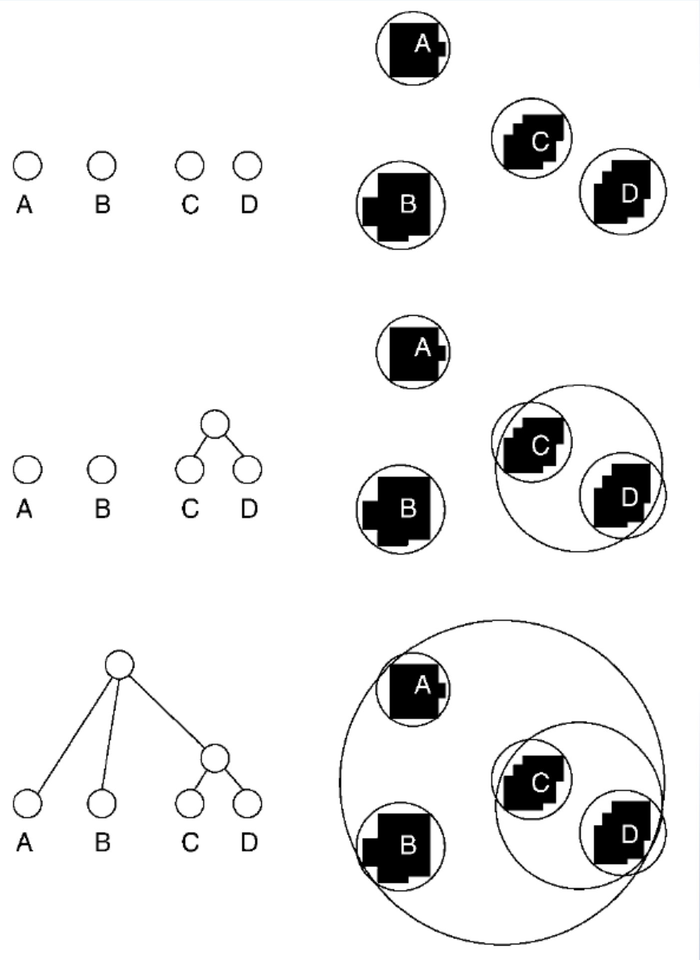
Bounding Volume Hierarchies

- Concept
  - If bounding volumes of two nodes in the tree do not touch
    - None of their descendants can possibly be in contact
- Different bounding volumes for each object and parent
  - Gives best fit
- Simpler implementation
  - Use spheres
  - Large parent bounding volumes
    - But fast calculations



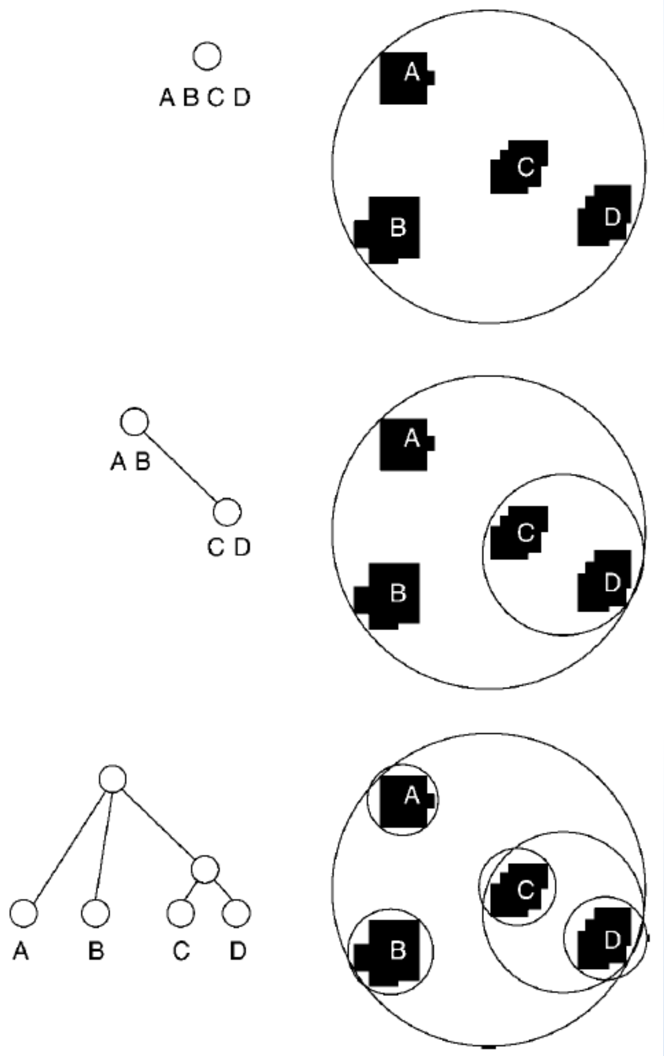
Bounding Volume Hierarchies

- Building the hierarchy
  - Static environments
    - Hierarchies can be constructed offline
  - Dynamic environments
    - Hierarchies need to be recalculated in game
  - Bottom-up
    - Starts with bounding volumes of individual objects
    - Parents added
    - Continue until only one node left in the list



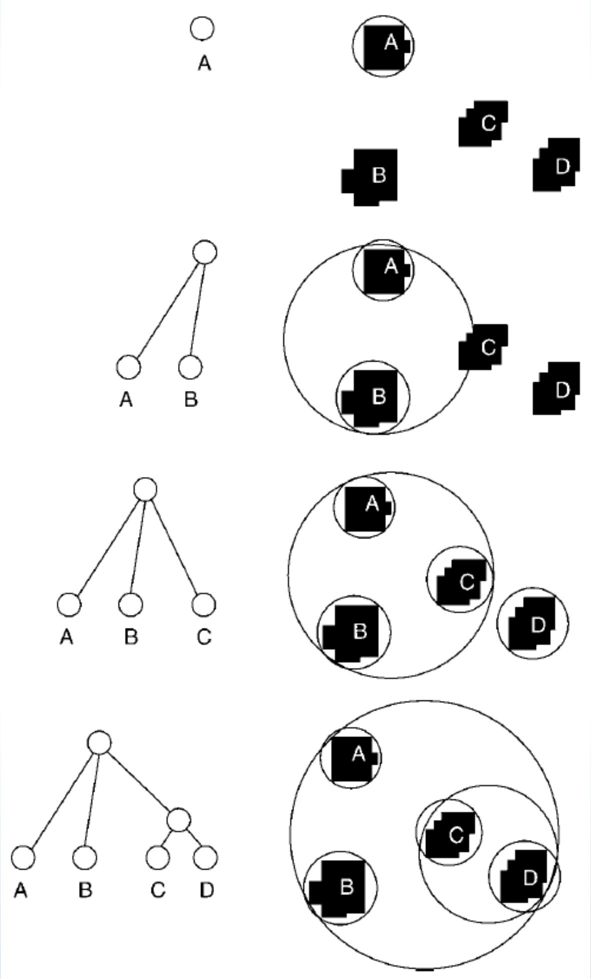
Bounding Volume Hierarchies

- Top-down
  - Also starts with bounding volumes of individual objects
  - Each iteration, objects in each group separated into two groups
  - Continue until only one object in each group



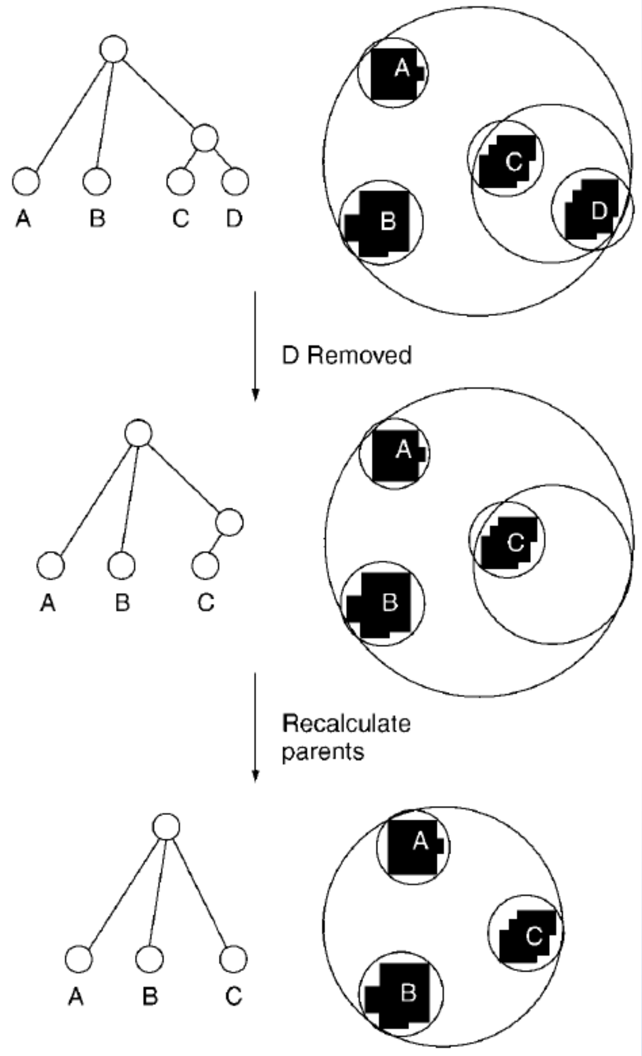
Bounding Volume Hierarchies

- Insertion
  - Only approach suitable for use during the game
  - Can adjust hierarchy without having to rebuild it completely
  - Starts with existing tree (or empty tree)
  - Objects added/inserted into the tree

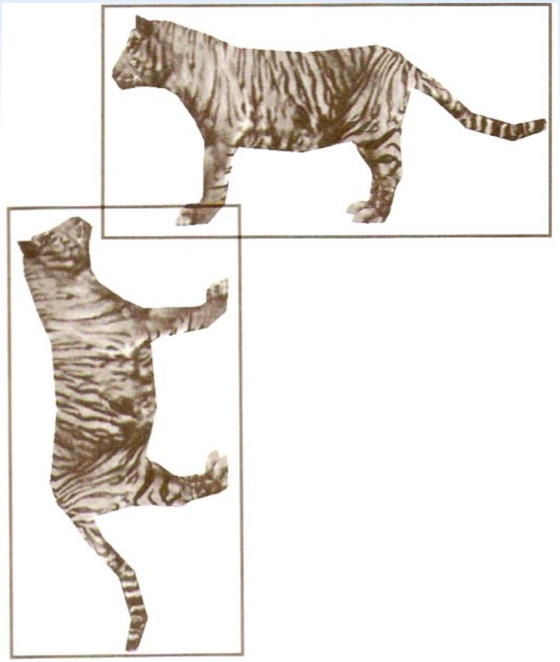


Bounding Volume Hierarchies

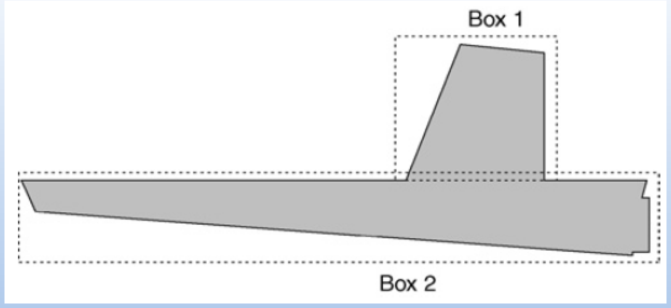
- Removing objects
  - Cannot simply remove an object
  - Potentially
    - Need to replace or remove parent nodes
    - Need to recalculate the bounding volumes further up the hierarchy

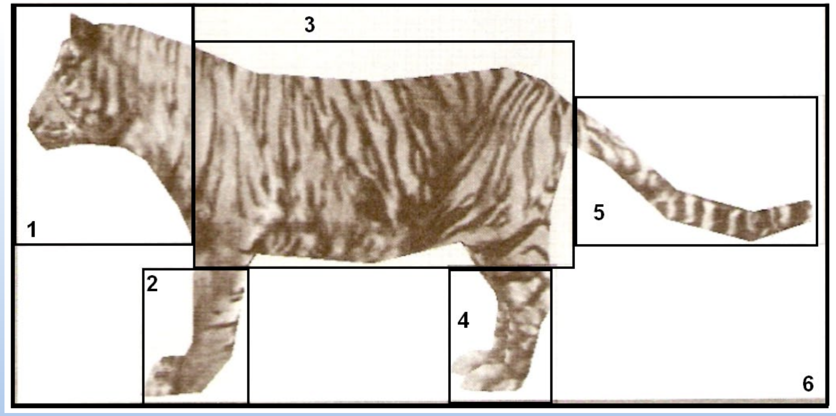


Sub-object Hierarchies



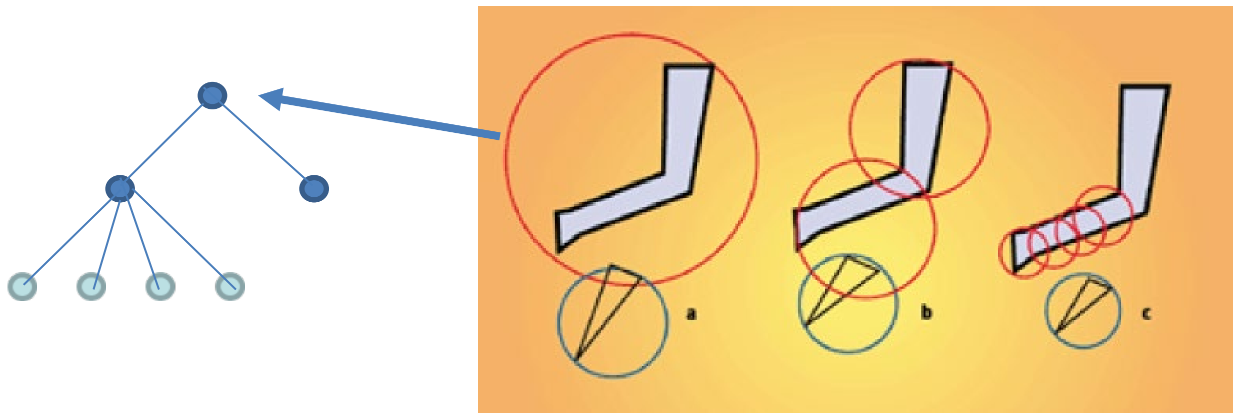
- Sub-object hierarchies
  - ➢ Objects with awkward shapes
    - Difficult to fit within simple bounding volumes
    - Results in inaccurate collision detection
  - ➢ Solution
    - Use multiple bounding volumes arranged in a hierarchy for the object
      - – Gives tighter fit

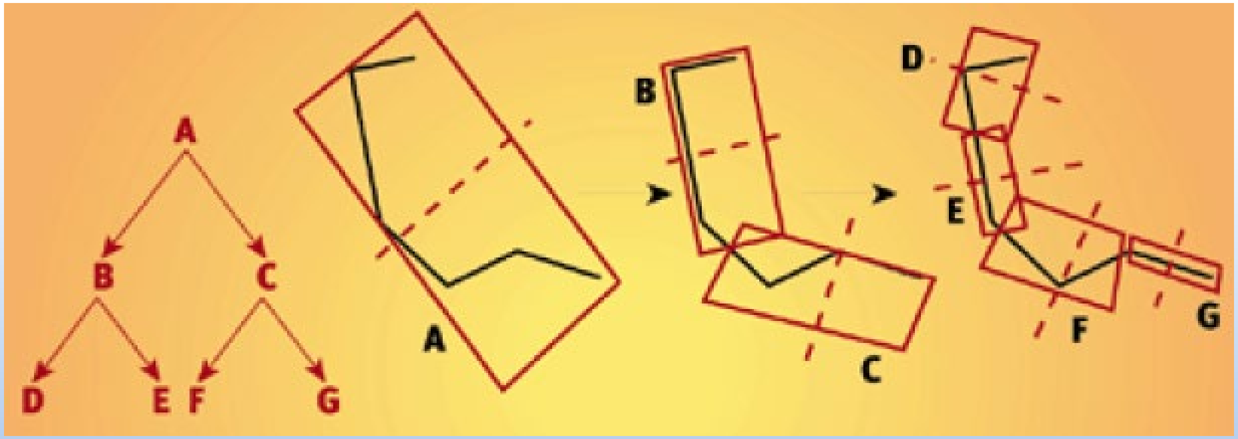




Sub-object Hierarchies

- Sub-object hierarchies





Spatial Data Structures

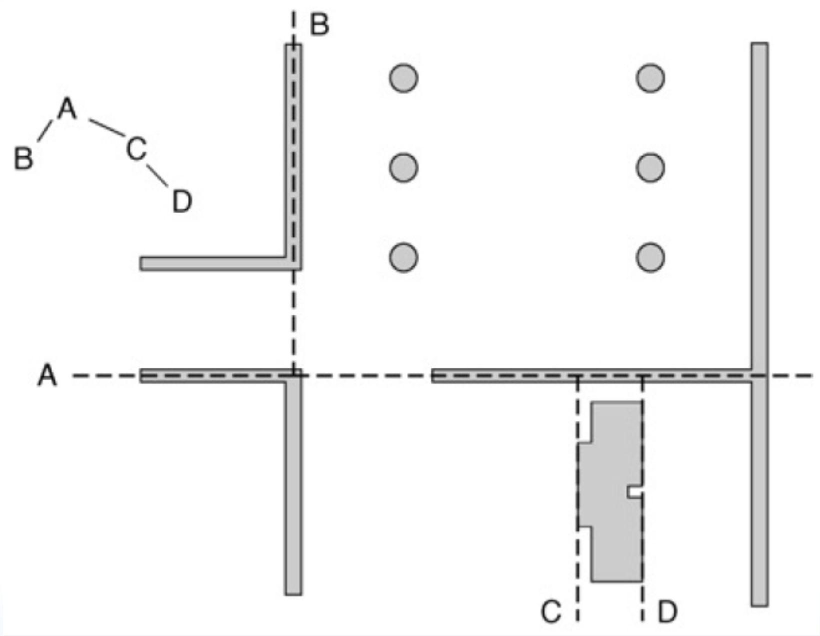
- Bounding volume hierarchy
  - Groups objects based on relative positions and sizes
  - If objects move, the hierarchy needs to be recalculated
- Spatial data structure
  - Locked to the world
  - Structure doesn’t change with regard to objects
  - Common structures found in games
    - Binary Space Partitioning (BSP)
    - Octree or quadtree
    - Grids

Spatial Data Structures

- BSP tree
  - First proposed in 1980!! But only gained popularity when 3D games started using it
    - Original purpose was to solve the visibility determination problem in computer graphics
  - Often seen in First-Person-Shooter (FPS) game engines
    - Doom was the first game that used this as part of the rendering pipeline for back-face culling, partial z-ordering, hidden surface removal
  - Also used to accelerate rendering and other things like shadow generation, collision detection

Spatial Data Structures

- Binary tree
  - Built by recursively subdividing objects or polygons in a scene using planes
  - Unlike BVH, each node uses a plane to divide space
  - Each parent only has a maximum of two children
- Two phases
  - Building the BSP tree
    - Can be done offline
  - Traversing the BSP tree
    - At run-time



Spatial Data Structures

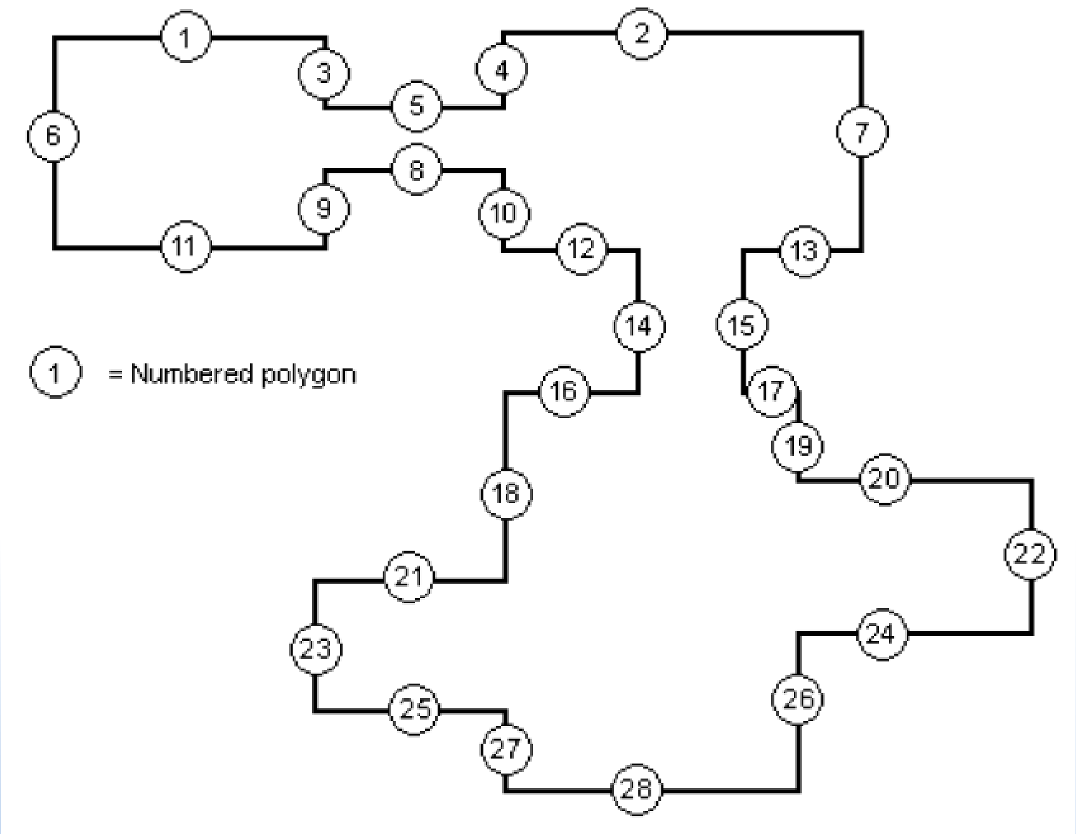
- Concept
  - Properties of partitioning a scene using planes
    - Objects on one side of a plane can never collide with objects on the other side
    - Given a viewpoint in the scene space, objects on the same side as the viewer are nearer than any objects on the other side
- Works well for static objects in scenes
  - Useful for indoor scenes in games
    - Boundaries are often static planes, e.g., walls, floor
- Many different variations to BSP trees
  - Can even put BVHs at the leaves of BSP tree

Spatial Data Structures

- Constructing the tree is no trivial task, among things to consider
  - How to choose partition planes
    - Best to get a balanced tree
  - If an object crosses a partitioning plane
    - Split object? (expensive)
    - Place in parent node
    - Place in closest child node
    - Place in both child nodes
  - When to stop partitioning space

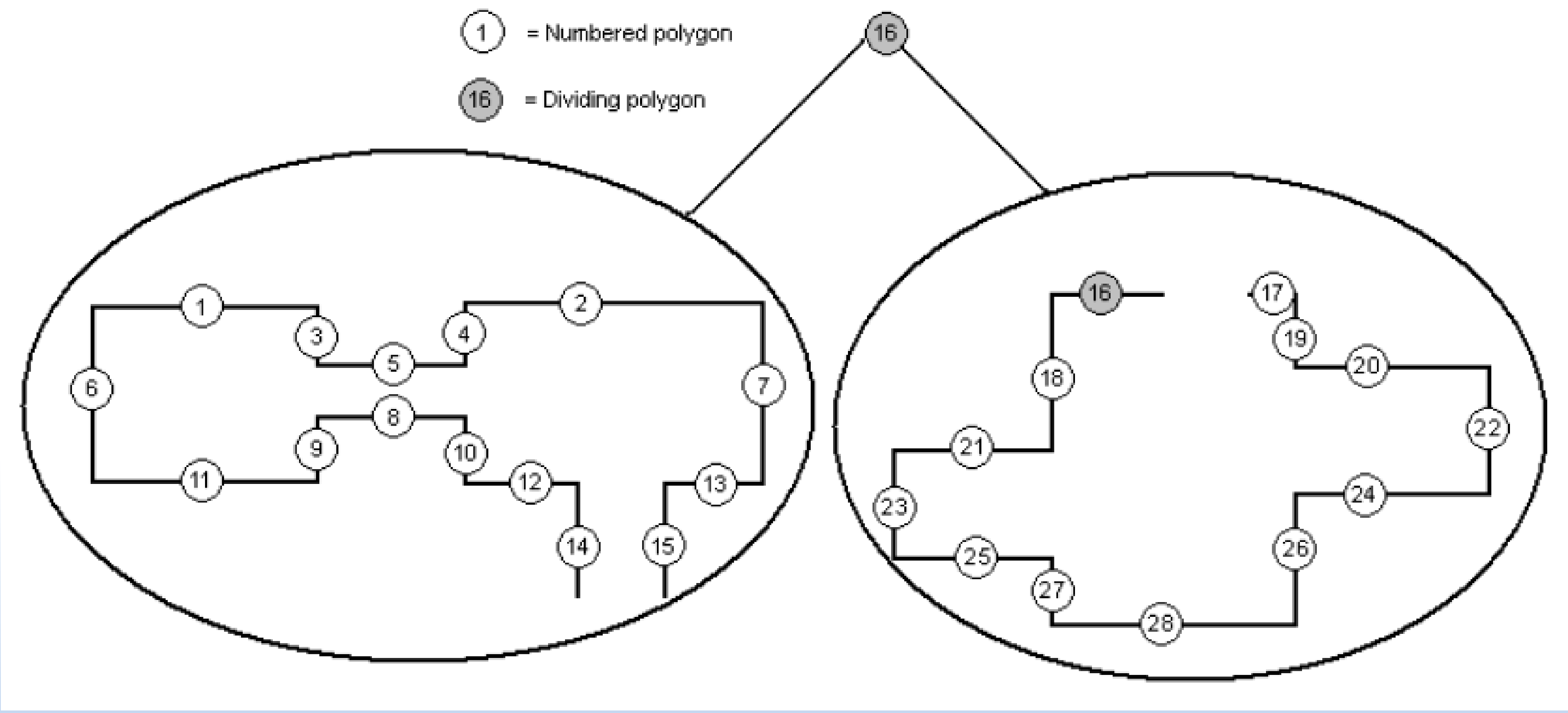
Spatial Data Structures

- BSP tree example



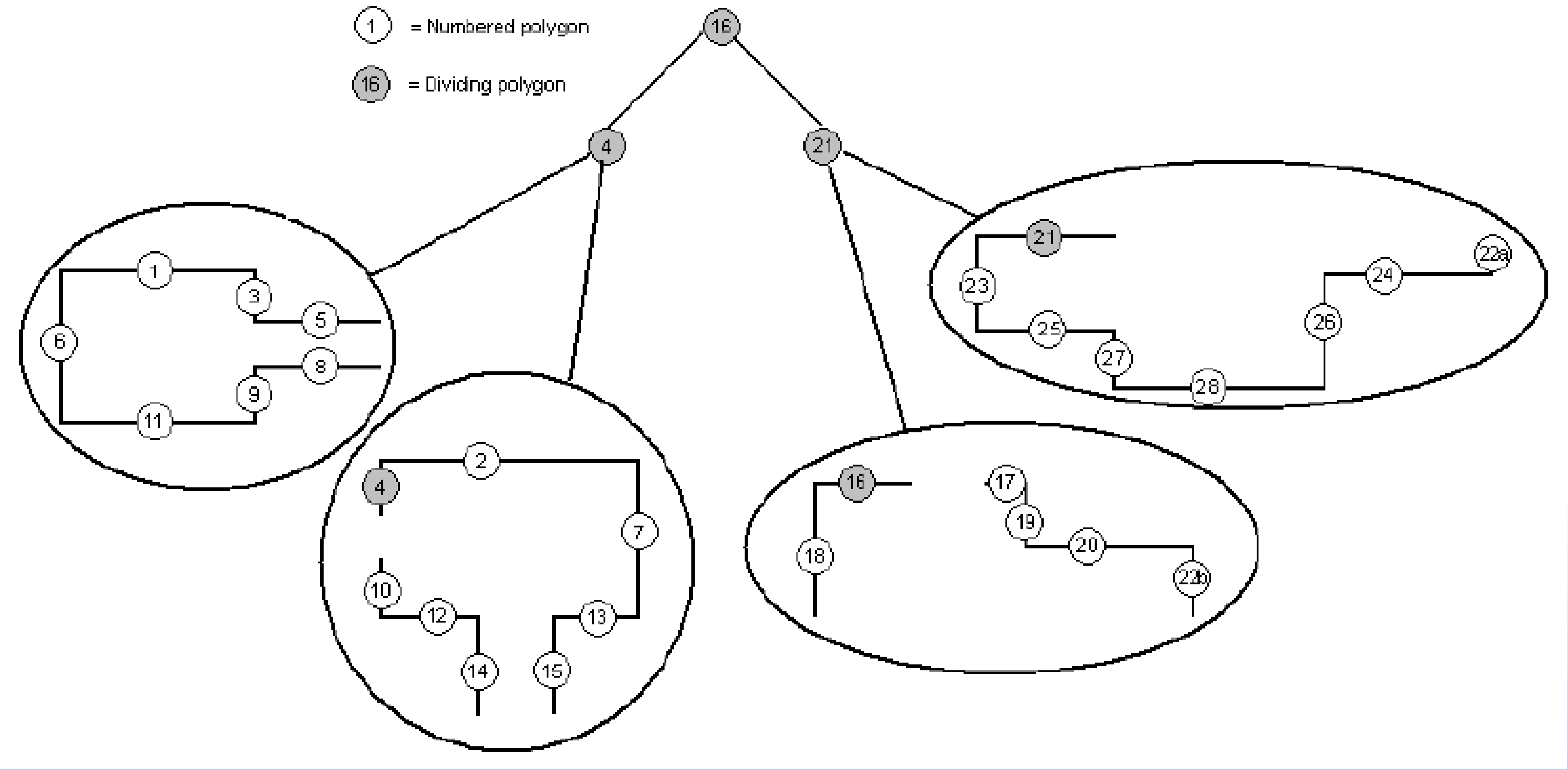
Spatial Data Structures

- BSP tree example



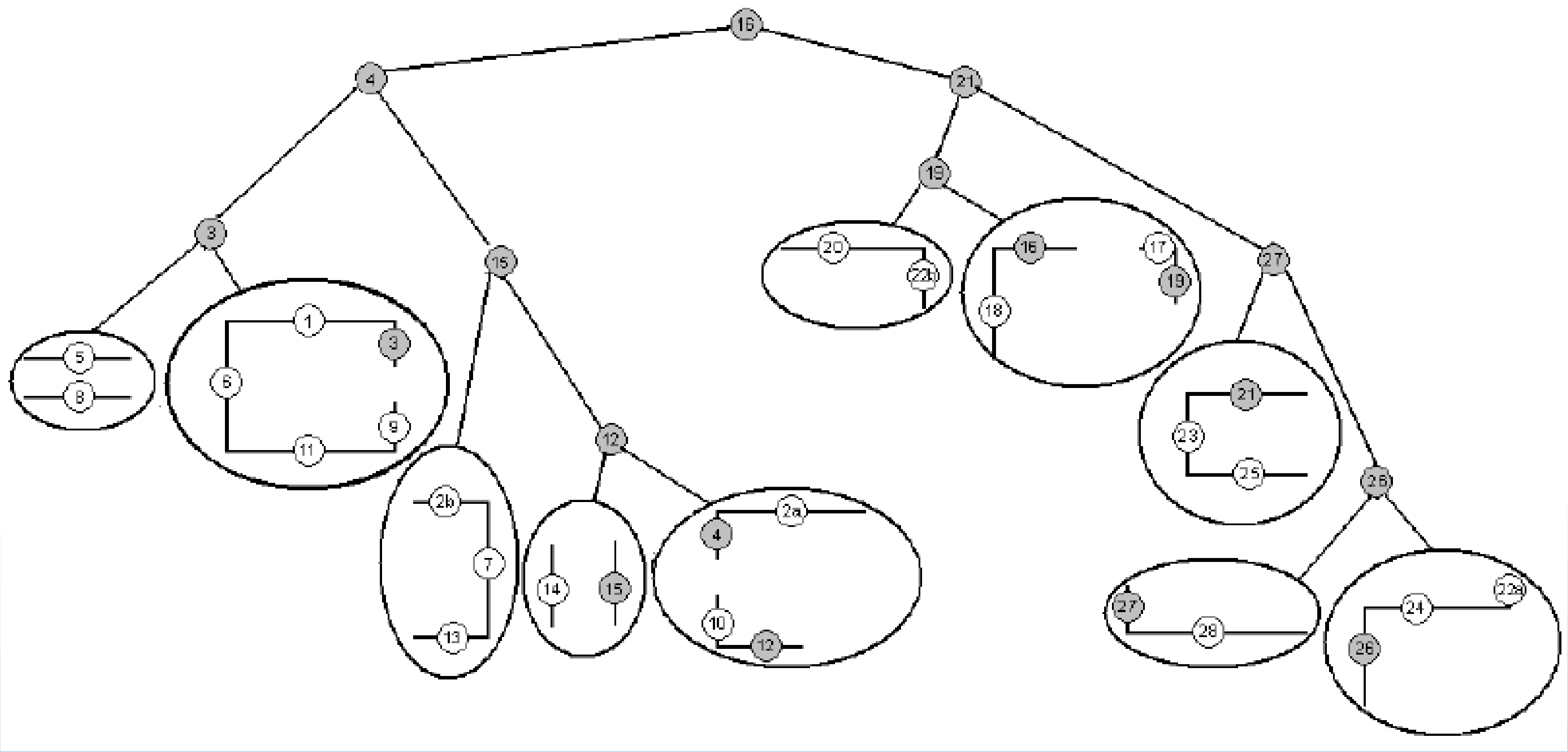
Spatial Data Structures

- BSP tree example



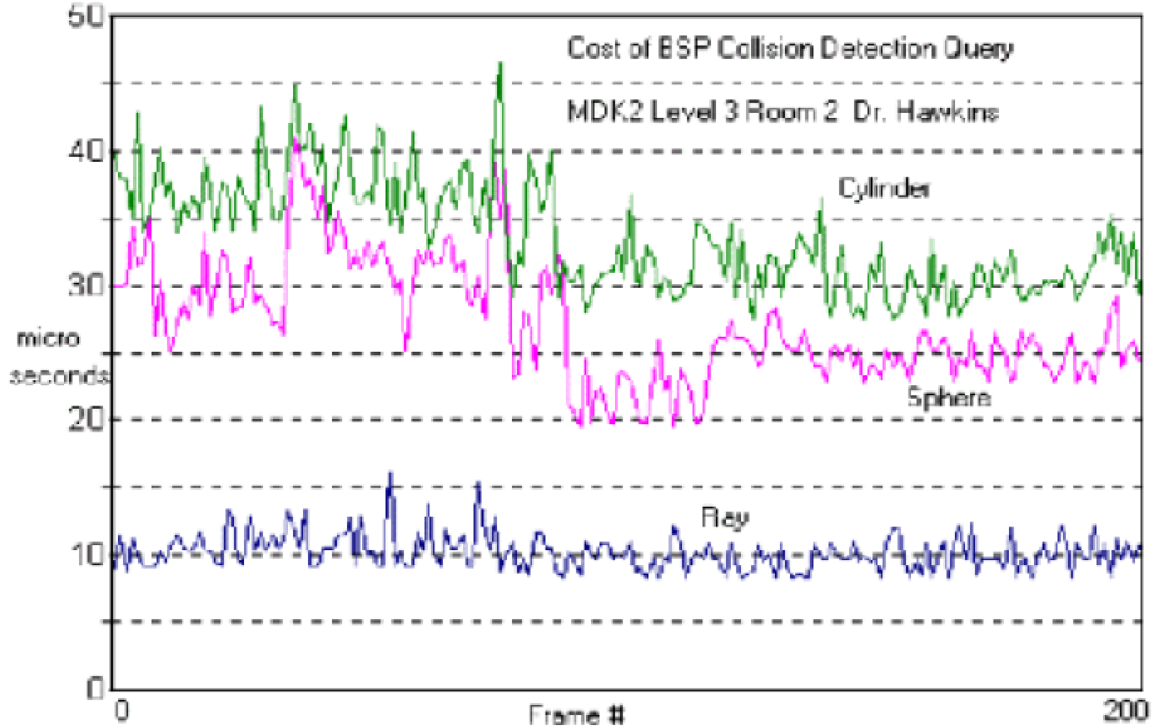
Spatial Data Structures

- BSP tree example



Spatial Data Structures

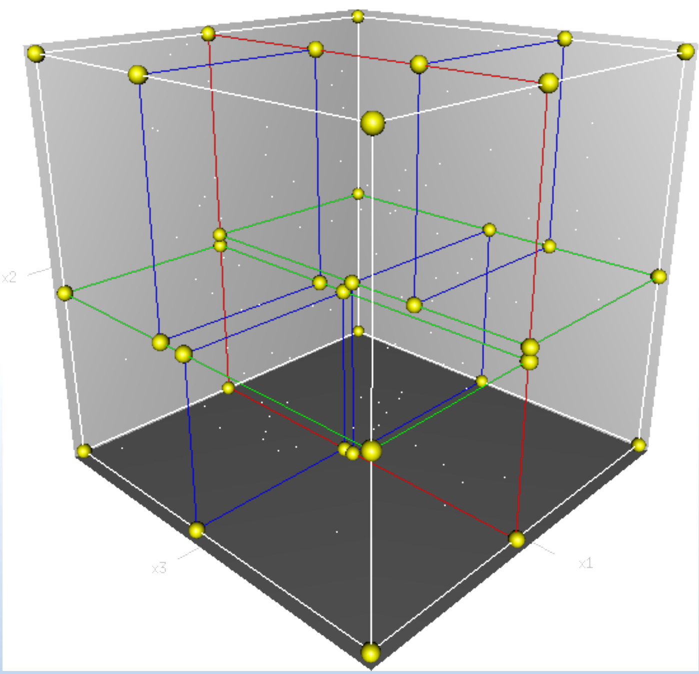
- BSP tree example

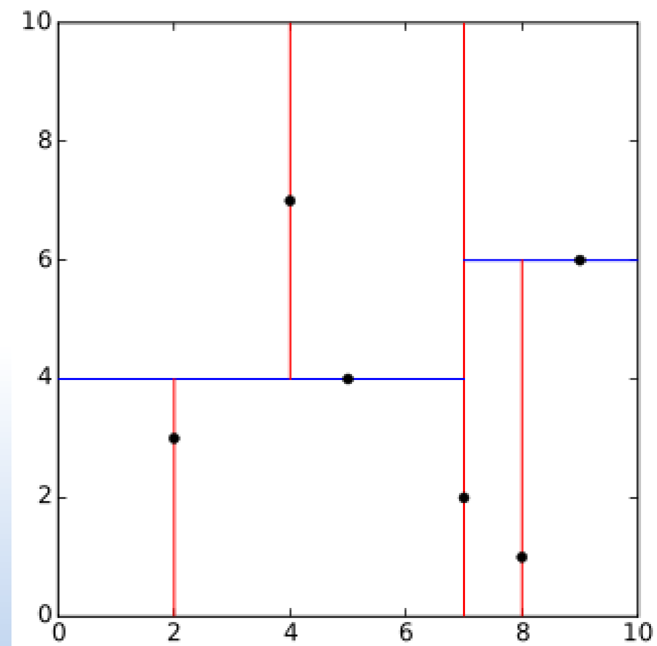


Comparison of three methods of collision detection: regular BSP collision (labeled Ray), spherical offset, and cylindrical offset.

Spatial Data Structures

- *k*-dimensional tree (*k*d-tree)
  - ➢ A special case of BSP tree
    - Partition planes perpendicular to one of the coordinate axes

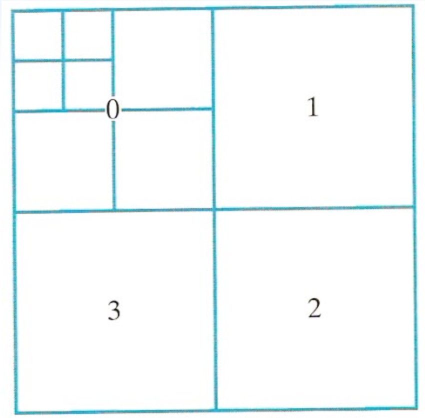


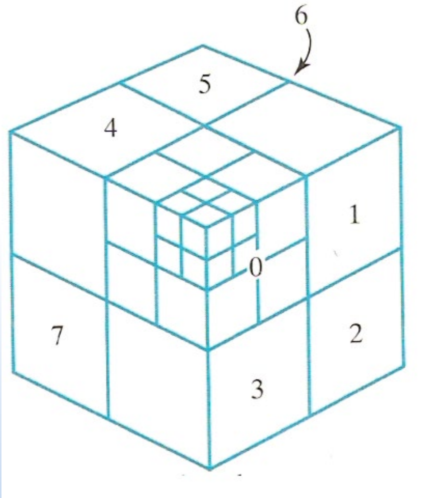


http://en.wikipedia.org/wiki/Kd-tree

Spatial Data Structures

- Octrees and quadtrees
  - ➢ Quadtree
    - Splits space into four areas
      - – Each node can have up to four children
    - Can also be used in 3D games
      - – Where most objects stuck to the ground
      - – Takes up less memory than an octree
  - ➢ Octree
    - Splits space into eight areas
      - – Each node can have up to eight children
  - ➢ Do not have to store empty nodes



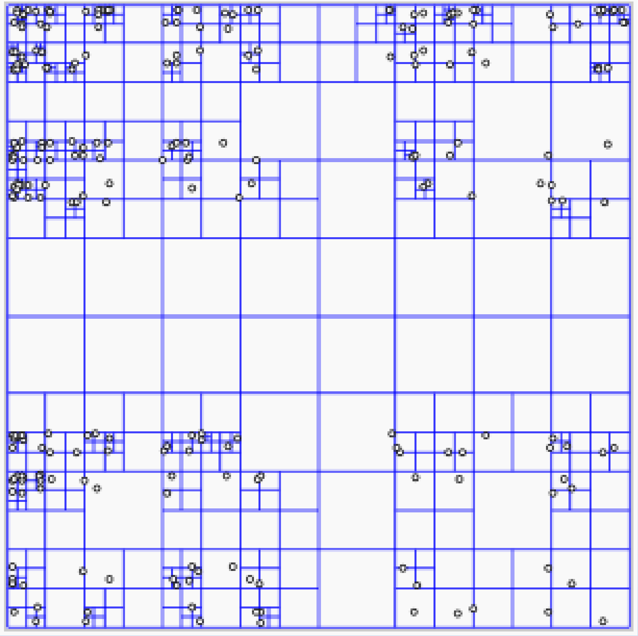


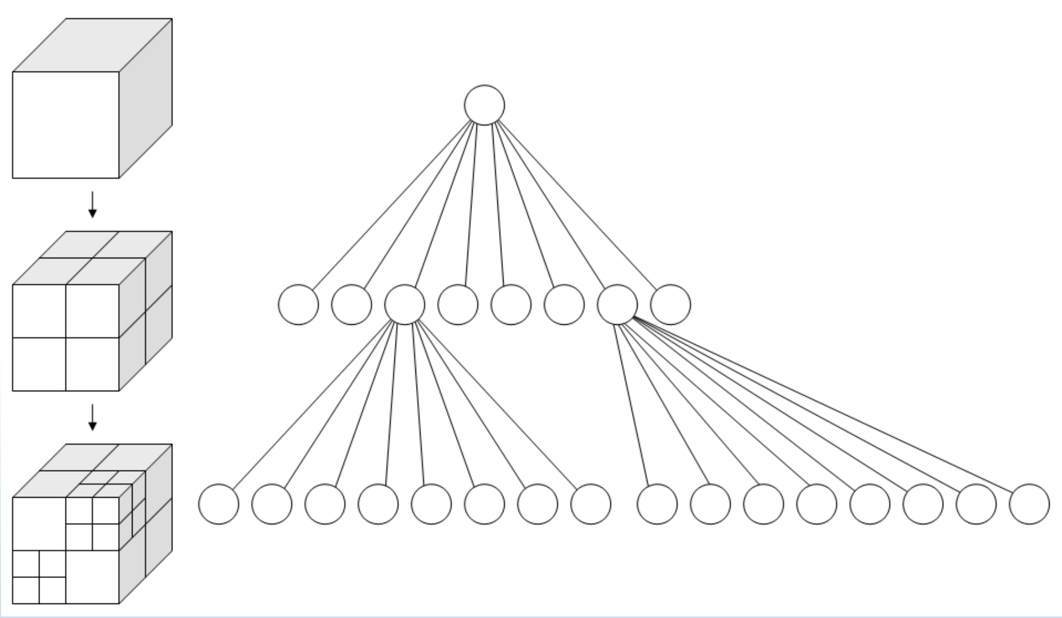
Spatial Data Structures

- Subdivide space
  - Until a termination criterion is reached
    - E.g., contains less than a predefined number of objects
- Each level does not require space to be divided into boxes/cubes of the same sizes
  - However, two advantages
    - Do not have to store the position of the nodes (can infer from the hierarchy level and node number)
      - » Saves memory
    - Do not need to perform calculations to find the best location to divide into nodes
      - » Faster to build

Spatial Data Structures

- Particularly useful for
  - Outdoor scenes in games where most objects are on the ground
  - Less for indoor games
    - Cannot easily use to detect collisions with walls

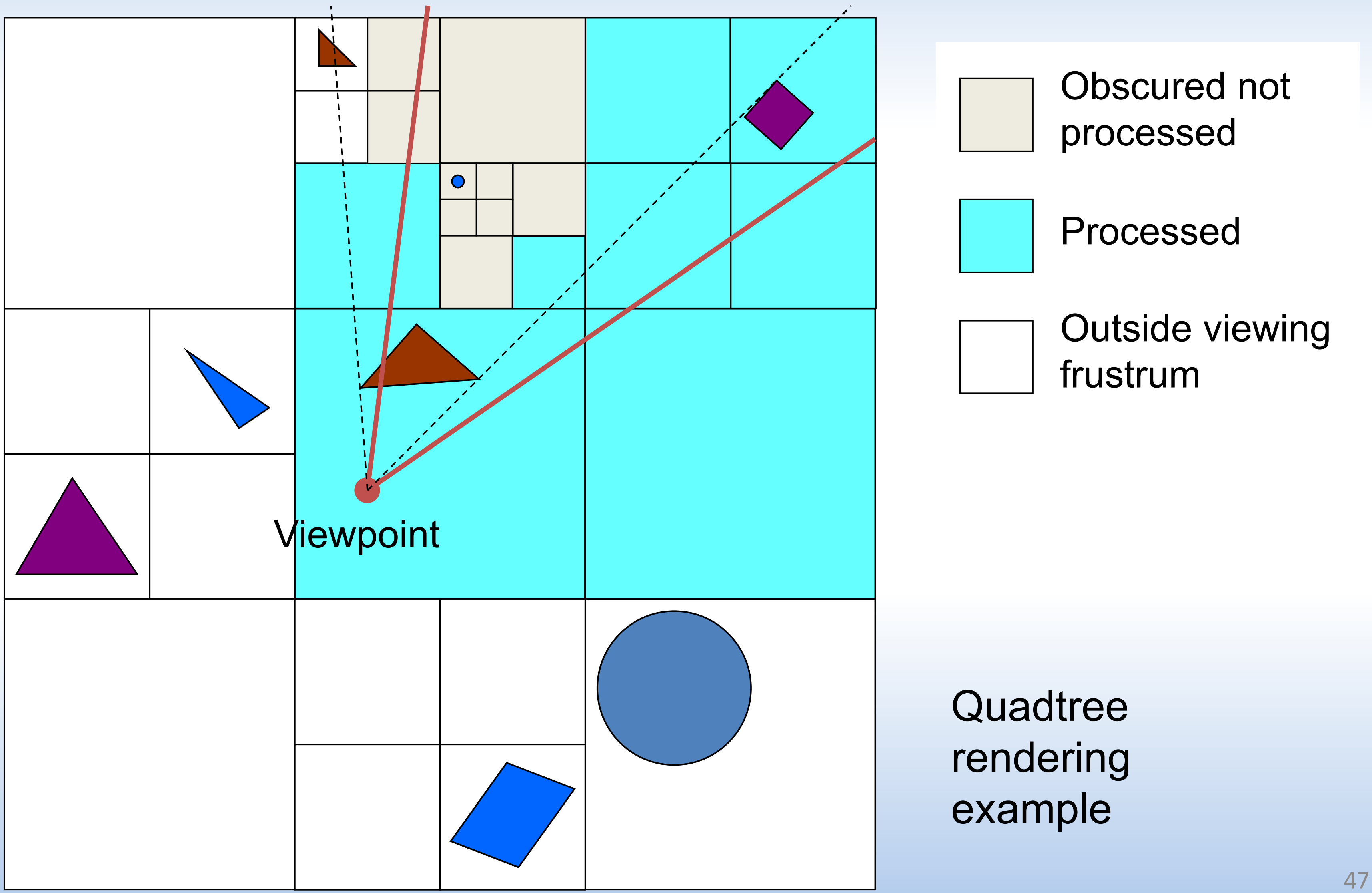




http://en.wikipedia.org/wiki/Quadtree

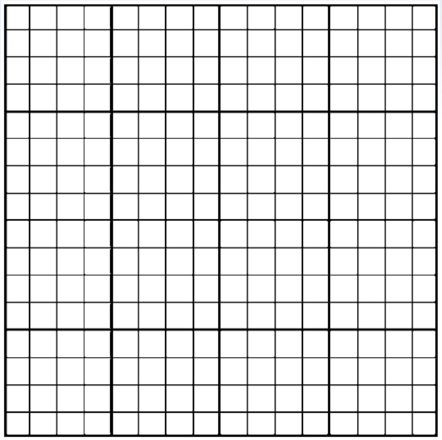
http://en.wikipedia.org/wiki/Octree

Spatial Data Structures



Quadtree rendering example

Spatial Data Structures



- Grids
  - ➢ Unlike quadtree/octree
    - Does not use a tree data structure
      - – Just a regular grid array
  - ➢ Advantages over tree-based approaches
    - An object's location in the grid
      - – Can be determined directly from the object's position
      - – Once found just check for collision with other objects in that cell
      - – Faster as do not have to traverse a tree structure
    - Can be defined and built without requiring any information about objects in the environment

Spatial Data Structures

- Tradeoff in selecting cell size
  - Large cells, may need to check lots of objects
  - Small cells, may have to keep multiple records of objects

➢ Disadvantages

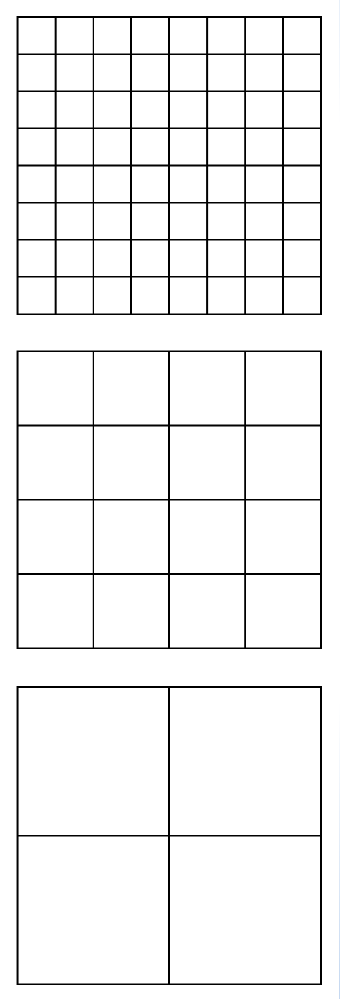
- Each cell can contain any number of objects
  - Quadtrees/Octrees typically have a maximum number of objects per node
- Cells exist even if space is empty
  - Sparse environments can have lots of empty space
  - Wastes memory
- Still have to check empty cells
  - Check whether they contain objects
  - Wasted effort

Spatial Data Structures

- Large objects span multiple cells
  - Tree structure: large objects can be placed higher in the tree (i.e. in parent nodes)
  - Grids: each cell must store the object, wastes memory
  - To remove a large object, must remove from all cells

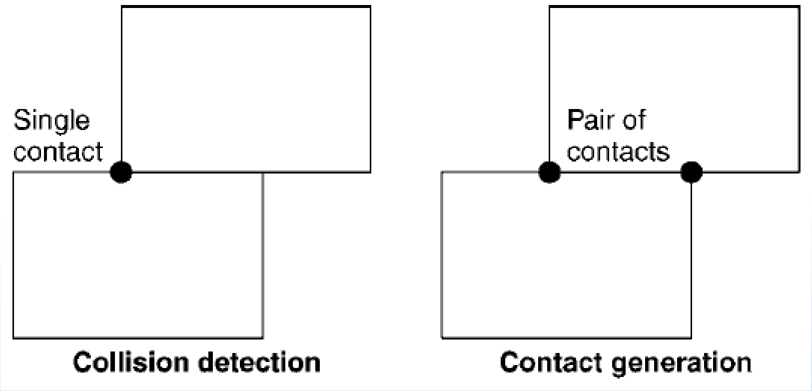
➢ Multi-resolution grids

- A set of grids with increasing cell sizes
  - Usually four times the size of previous level, so grids align
- Objects added into one of the grids only
  - Based on the size of the object
    - » i.e. smallest cells that can contain object



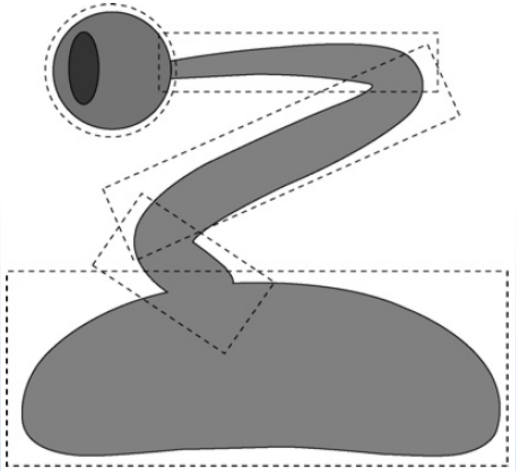
Fine Collision Detection

- Fine collision detection
  - ➢ Coarse collision detection produces a list of object pairs that may have collided
    - Need to check whether there are contacts to generate
  - ➢ More complex than single-intersection collision detection
    - Collision may have multiple contact points
    - Single point may not be enough to generate realistic response
    - Takes more processor time to complete



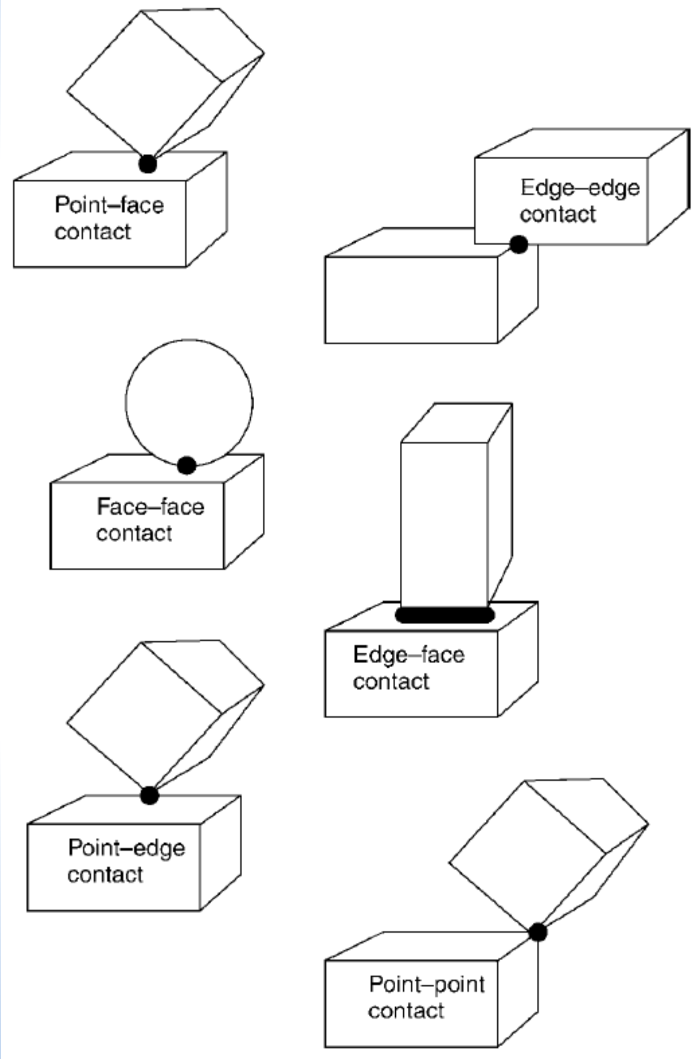
Fine Collision Detection

- Contact generation
  - ➢ Work out the set of contact points
    - Not just whether objects are touching or overlapping
    - Need to find contact data before can determine collision response
    - Needs to be fast
      - – Improve speed by performing against simplified geometry rather than rendering geometry
  - ➢ Compound bodies
    - Assemblies of primitive objects
    - Each primitive typically has a transformation matrix to offset it from the object's origin
    - To collide two assemblies, find collisions between all pairs of primitives for each object



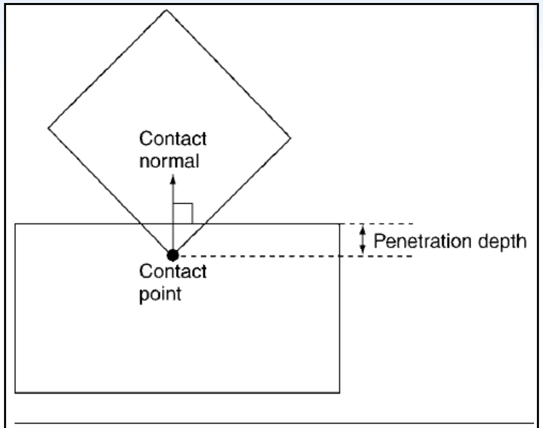
Fine Collision Detection

- Contact generation
  - ➢ A limited set of contact situations used to approximate contacts
    - Accuracy is sufficient for video games
    - Face-face collision is only used when one object has a curved surface
      - – Otherwise use edge-face
    - Edge-face can be approximated by a pair of contact points
    - Point-point contacts are so rare that are ignored by some physics engines

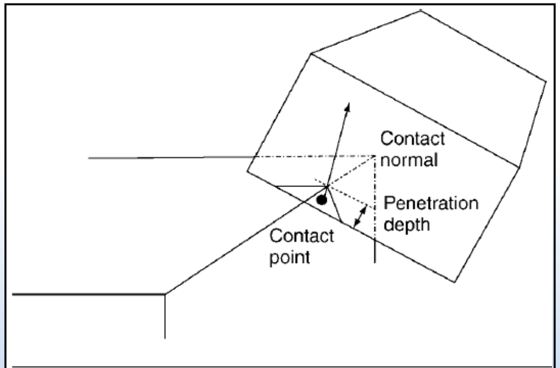


Fine Collision Detection

- Contact data
  - ➢ Collision point
    - Point of contact between two objects
    - Objects will be interpenetrating somewhat, may be a number of possible points
  - ➢ Collision normal
    - Direction in which impact impulse will be felt between two objects
  - ➢ Penetration depth
    - The amount that two objects are interpenetrating
    - Measured along the direction of the collision normal passing through the collision point



The point–face contact data.



The edge–edge contact data.

Fine Collision Detection

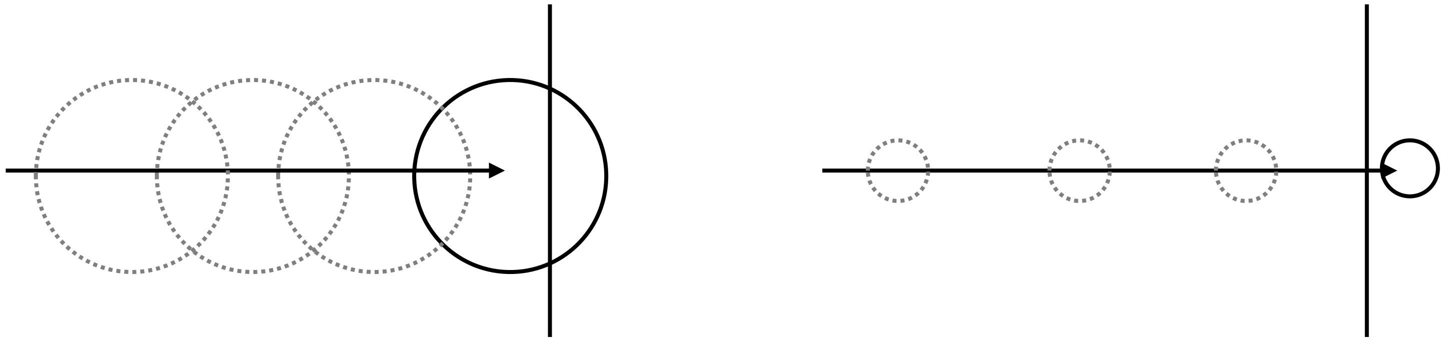
- Unity
  - `Collision`
    - Describes a collision and passed to the following event methods:
      - `OnCollisionEnter()`
      - `OnCollisionStay()`
      - `OnCollisionExit()`
    - Some properties
      - `contactCount`: number of contacts for this collision
      - `relativeVelocity`: the relative linear velocity of the two colliding objects
      - `gameObject`: the GameObject whose collider you are colliding with
    - Public method
      - `GetContacts()`: retrieves all contact points for this collision

Fine Collision Detection

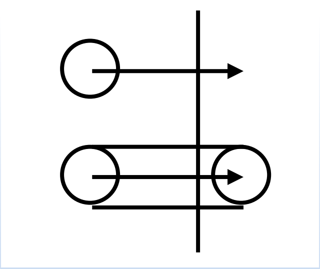
- Unity
  - `ContactPoint`
    - Describes a contact point where the collision occurs
    - Properties
      - `impulse`: impulse applied for collision resolution
      - `normal`: normal of the contact point
      - `point`: point of contact
      - `separation`: distance between colliders at the point of contact
      - `thisCollider` : the first collider in contact at the point
      - `otherCollider` : the other collider

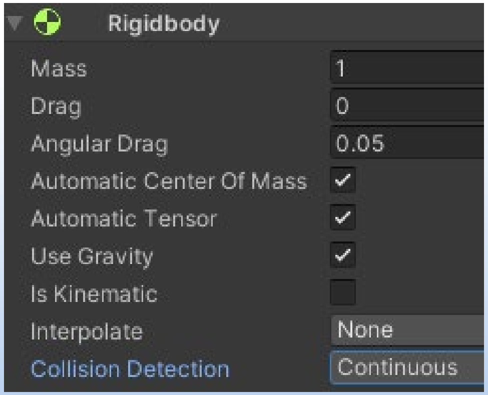
Collision Detection

- Discrete collision detection
  - ‘Tunneling’ problem
    - Potential problem for small objects moving at high speeds



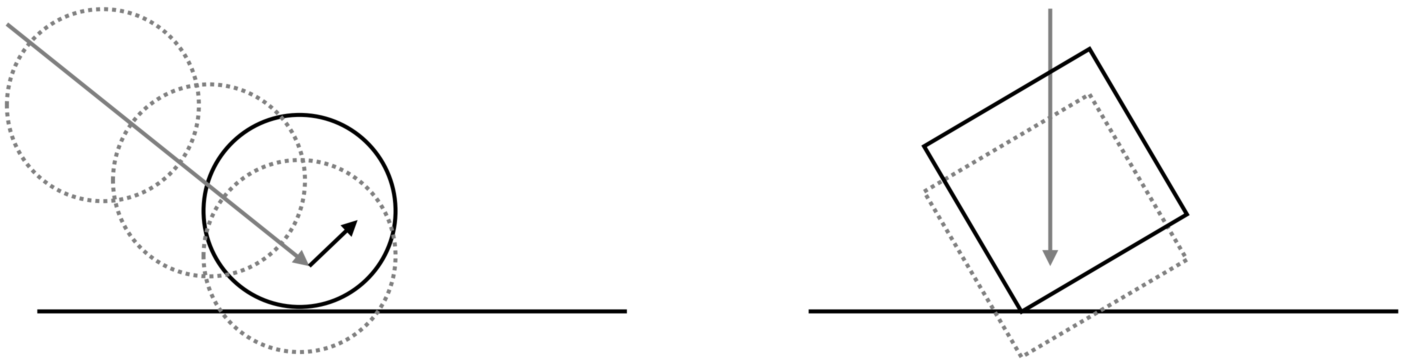
  - Solution
    - Ray casting
    - Sweep shapes
    - Continuous collision detection
      - Predict time of impact along current path



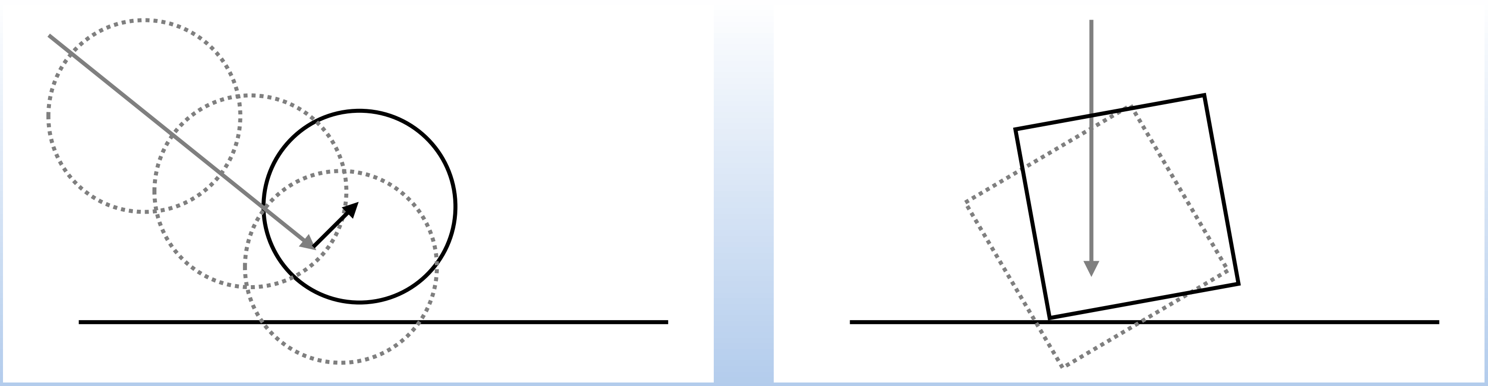


Collision Detection

- Resolving penetration
  - ➢ Simple but inaccurate approximation



  - ➢ More accurate method



Collision Queries

- Ray cast
  - ➢ Cast a directed line segment
    - From a starting point to an endpoint
      - – Some game engines do not support infinite rays
    - Line segment tested against collidable objects
      - – Returns contact point(s)
      - – Typically want the closest intersection point
      - – Spatial data structures speed up collision tests
  - ➢ Example applications
    - Weapons, direct line of sight, movement queries, distance to ground

Collision Queries

- Unity
  - `Physics.Raycast`
    - For example

```
bool Raycast(Vector3 origin, Vector3 direction,
   out RaycastHit hitInfo,
   float maxDistance, // these have default values
   int layerMask,
   QueryTriggerInteraction queryTriggerInteraction);
```

  - Has a number of variants
  - Returns true if the ray intersects with a Collider
  - Casts a ray against all colliders in the Scene and returns detailed information on what was hit

Collision Queries

- Unity
  - `RaycastHit`
    - Structure used to get information back from a raycast
    - Some properties
      - `point`: impact point in world space where the ray hit a Collider
      - `distance`: Distance from ray's origin to impact point
      - `normal`: normal of the surface that was hit
      - `collider`: Collider that was hit
      - `rigidbody`: Rigidbody of the Collider that was hit
      - `transform`: Transform of the Collider/Rigidbody that was hit

Collision Queries

- Shape cast
  - Casting a shape along a directed line segment
    - Shape usually a sphere
  - Example applications
    - Sliding a character forward on uneven terrain
    - Determining whether an object can move between obstacles
    - Virtual camera collision
- Volume queries
  - Determine which collidable objects lie within some specific (invisible) volume
    - Like a zero distance shape cast, but unlike casts can be persistent and takes advantage of temporal coherence

Collision Queries

- Unity
  - `Physics.SphereCast`
    - For example

```
bool SphereCast(Vector3 origin, float radius,
   Vector3 direction, out RaycastHit hitInfo,
   float maxDistance, // these have default values
   int layerMask,
   QueryTriggerInteraction queryTriggerInteraction);
```

- Returns true if the sphere sweep intersects any Collider
- Casts a sphere along a ray and returns detailed information on what was hit
- Useful when a Raycast does not give enough precision

Collision Queries

- Unity
  - `Physics.OverlapSphere`
    - Returns an array with all Colliders touching or inside the sphere

```
Collider[] OverlapSphere(Vector3 position, float radius,
   int layerMask, // these have default values
   QueryTriggerInteraction queryTriggerInteraction);
```

Collision Detection

- Unity
  - Interaction between Colliders
    - Static Collider
      - A GameObject that has a Collider but not a Rigidbody component
      - Does not move when a Rigidbody collides with it
    - Rigidbody Collider
      - Fully simulated by the physics engine and can react to collisions and forces
    - Kinematic Rigidbody Collider
      - Rigidbody with isKinematic property enabled
      - Reacts to Rigidbody Colliders but not affected by forces and collisions
      - Does not react to other kinematic Colliders and static Colliders
  - Bounciness and surface friction set using physics material

Collision Detection

Screen captures from “Making of Halo 3” – Bungie Studios

References

- Among others, material sourced from
  - https://docs.unity3d.com
  - Jason Gregory, Game Engine Architecture, A.K. Peters
  - Ian Millington, Game Physics Engine Development, Morgan Kaufmann
  - David Conger, Physics Modeling for Game Programmers, Thomson Learning
  - Stan Melax, "BSP Collision Detection as used in MDK2 and Never Winter Nights"

    http://www.gamasutra.com/view/feature/3099/bsp_collision_detection_as_used_in_.php
  - Nick Bobic, "Advanced Collision Detection Techniques"

    http://www.gamasutra.com/view/feature/3190/advanced_collision_detection_.php
  - Jeff Lander, "When Two Hearts Collide: Axis-Aligned Bounding Boxes"

    http://www.gamasutra.com/view/feature/3426/when_two_hearts_collide_.php
  - Samuel Ranta-Eskola, "Binary Space Partitioning Trees and Polygon Removal in Real Time 3D Rendering", Masters Thesis, Uppsala University