Collision Detection

- How to determine if a shell hit the target?
- How to determine if the driver is off the road?
- Did the kick hit the target?
- Will the submarine escape the torpedo?

Collisions in the gaming world

- Collisions in the gaming world are a two step operation:
  - Collision Detection
    - The action of determine what or who collided and when?
  - Collision Resolution
    - What to do about the collision?
    - Changes the game state (e.g., losing an airplane)

Addressing Collisions

- When addressing collision one must handle
  - Objects may be complex
  - Objects change location
    - Dynamic objects
    - Static objects
  - Many objects to handle
  - Discrete time steps
  - Limited amount of time
    - Real-time detection

- Dynamic environment
  - Many objects
  - Most of the objects change their position over time
  - Collisions between the objects changes the state of the game
- Static environment
  - Few objects change their position over time
  - Collision between objects may not affect the state of the game

Collision Detection Brute Force Approach

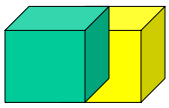
- At each time step of the game check all possible occurrences of a collision.
  - If there are *n* moving objects and m static objects then
  - $O(n^2 + mn)$ collision detection tests
  - Each test can be complex

Collision Detection Approaches

- Detection between objects (“low level”)
  - Overlap testing
  - Intersection detection
- Large scale detection
  - Simplifying objects
  - Space partitioning
  - Time based collisions

Overlap testing

- Every time step of the game test each pair of objects for intersection (overlap)



Pros
- Simple to implement
- Work well for small number of objects
- Easy for simple shapes

Cons
- Time consuming
- Detects whether a collision occurred in the past
- Require backtracking to determine exact collision time

Overlap Tests

- Sphere-Sphere overlap
- Sphere A –
  - c_a – centre of sphere
  - r_a – radius of sphere
- Sphere B –
  - c_b – centre of sphere
  - r_b – radius of sphere
- Overlap if

$$\|c_a - c_b\| \leq r_a + r_b$$

- Cube-Cube overlap
- Each cube is represented by a centre (x_i, y_i, z_i) and width w_i

$$\begin{aligned} |x_1 - x_2| &\leq w \text{ and} \\ |y_1 - y_2| &\leq w \text{ and} \\ |z_1 - z_2| &\leq w \end{aligned}$$

Overlap Tests

- 2 Rectanclges overlap
- Each cube is represented by
  - Centre (x_i, y_i, z_i)
  - Width (w_x, w_y, w_z)

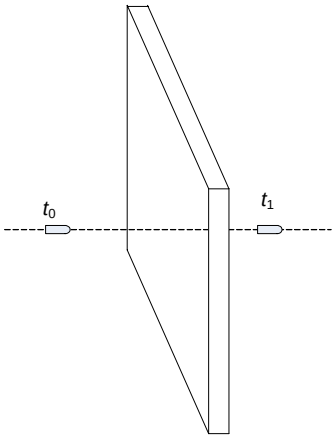
$$\begin{aligned} |x_1 - x_2| &\leq w_x \text{ and} \\ |y_1 - y_2| &\leq w_y \text{ and} \\ |z_1 - z_2| &\leq w_z \end{aligned}$$

Handling overlap collisions

- What happens if more than one collision was found?
- What happens if an object collides with two ore more other objects
- When a collision was detect exact time must be computed
  - Backtrack in time (e.g., using binary search)
  - Use exact geometry to find out when the two borders meet.
- Ensure that the correct collision was used to ensure proper game state

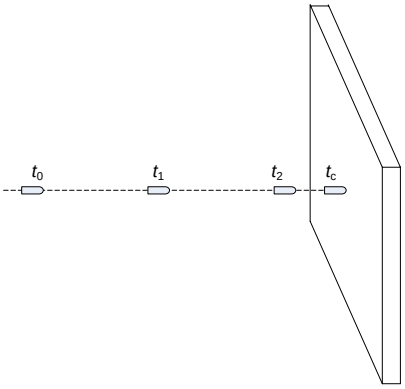
Limitation

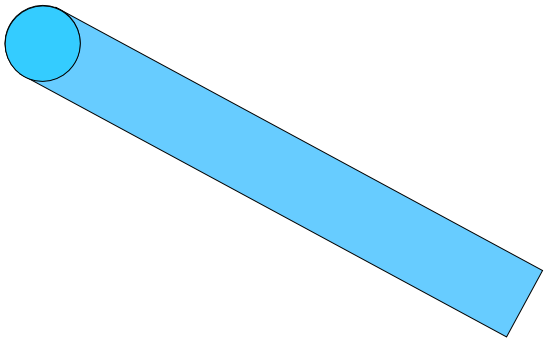
- Approach fails when
  - Objects move two fast
  - Time step too coarse
- Possible “Remedies”
  - Refine time steps
  - Limit speed of objects

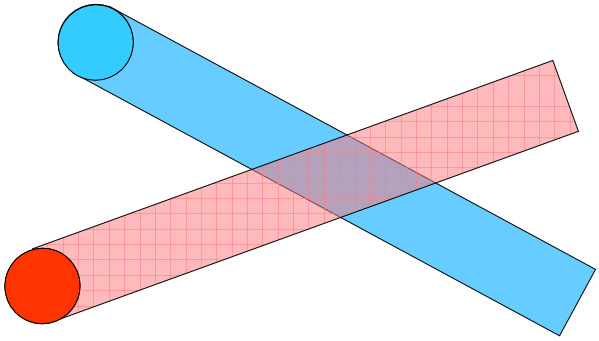


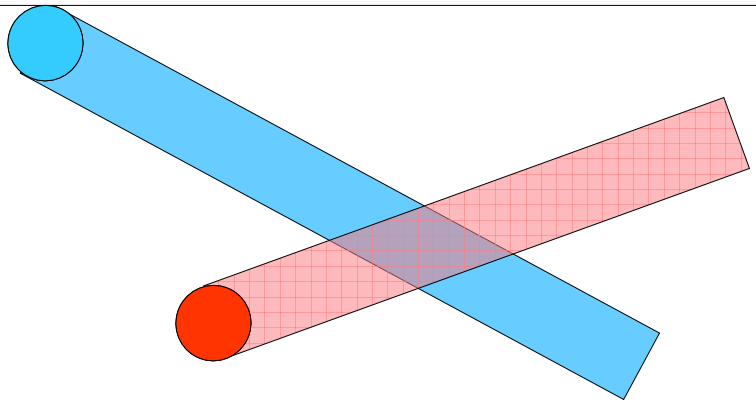
Intersection Testing

- Predicts when the collision will occur
- Operates on simulation time
- When a collision was found
  - Determine collision time
  - Do not recompute collisions until simulation time matches collision time

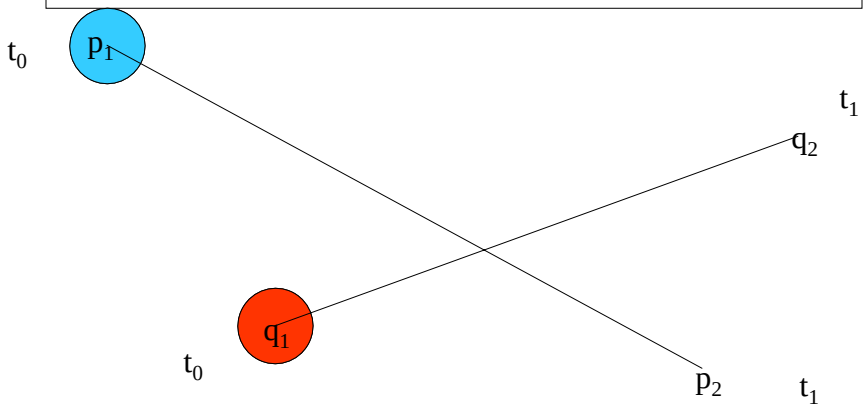




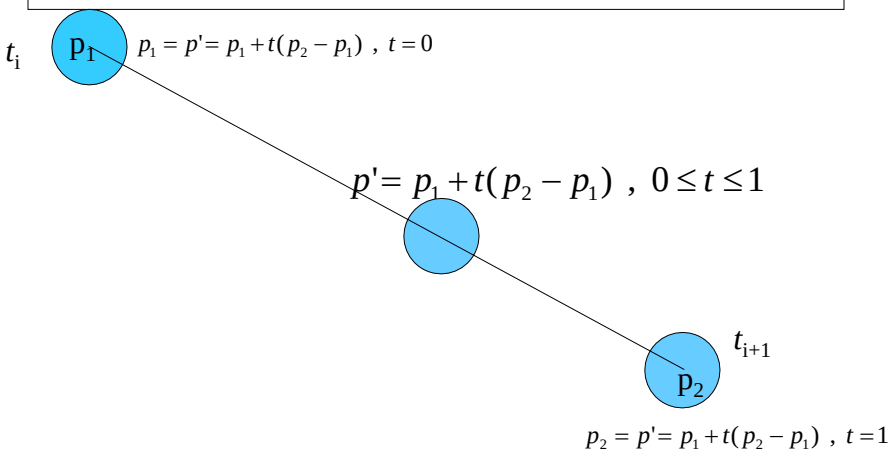




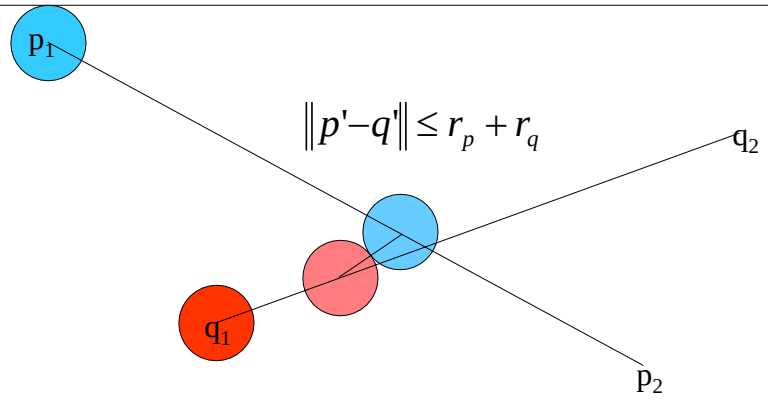
Parametric computation



Parametric computation



Parametric computation



$\|p'-q'\| \le r_p + r_q$

$(p'-q')^2 \le (r_p + r_q)^2$

$[p_1 + t(p_2 - p_1) - (q_1 + t(q_2 - q_1))]^2 \le (r_p + r_q)^2$

$[(p_1 - q_1) + t((p_2 - p_1) - (q_2 - q_1))]^2 \le (r_p + r_q)^2$

Letting $v = (p_1 - q_1)$ and $u = (p_2 - p_1) - (q_2 - q_1)$

we obtain

$[v + tu]^2 \le (r_p + r_q)^2$

$v^2 + 2tuv + t^2u^2 \le (r_p + r_q)^2$

$$t^2u^2 + 2tuv + v^2 - (r_p + r_q)^2 \le 0$$

solving the quadratic equation

$$t_{1,2} = \frac{-2uv \mp \sqrt{(2uv)^2 - 4u^2((v^2 - (r_p + r_q)^2)}}{2u^2}$$

$$t_{1,2} = \frac{-uv \mp \sqrt{(uv)^2 - u^2((v^2 - (r_p + r_q)^2)}}{u^2}$$

a solution exists if (note equality means toucing)

$$(uv)^2 > u^2((v^2 - (r_p + r_q)^2)$$

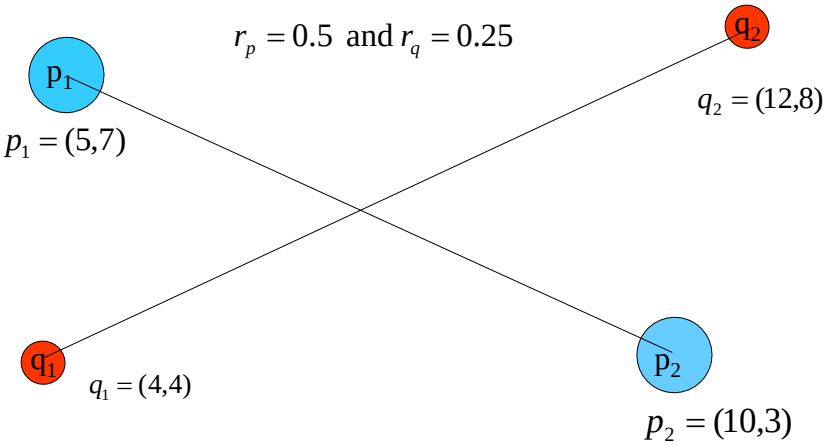
$$(r_p + r_q)^2 > v^2 - \frac{(uv)^2}{u^2}$$

- A collision occurs when

$$(r_p + r_q)^2 > v^2 - \frac{(uv)^2}{u^2}$$

u^2

Example



$v = (p_1 - q_1) = (5,7) - (4,4) = (1,3)$
and $u = (p_2 - p_1) - (q_2 - q_1)$
$u = ((10,3) - (5,7)) - ((12,8) - (4,4))$
$u = (5,-4) - (8,4) = (-3,-8)$
Other quantities
$uv = -27$
$v^2 = 10$
$u^2 = 73$

$$(r_p + r_q)^2 > v^2 - \frac{(uv)^2}{u^2}$$
$$(.5 + .25)^2 > 10 - \frac{729}{73}$$
$$0.5625 > \frac{1}{73}$$

$v = (p_1 - q_1) = (5,7) - (4,4) = (1,3)$
and $u = (p_2 - p_1) - (q_2 - q_1)$
$u = ((10,3) - (5,7)) - ((12,8) - (4,4))$
$u = (5,-4) - (8,4) = (-3,-8)$
Other quantities
$uv = -27$
$v^2 = 10$
$u^2 = 73$

$$t_{1,2} = \frac{-uv \mp \sqrt{(uv)^2 - u^2((v^2 - (r_p + r_q)^2)}}{u^2}$$
$$t_{1,2} = \frac{27 \mp \sqrt{729 - 73((10 - 0.5625)}}{73}$$
$$t_{1,2} = \frac{27 \mp 6.33}{73} \approx 0.28$$
$$t_{1,2} = \frac{27 \mp 6.33}{73} \approx 0.45$$

$p' = p1 + (p2 - p1) * t$
$p' = (5,7) + (5,-4) * 0.28 = (6.4,5.88)$

Similar computation

- Sphere-square
- Box-Box
- Sphere-box

- What is in common?

  Convex Objects

How to take advantage of it? (last)

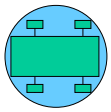
- Reduce number of collision detection operations
  - Compute collisions only as required
  - Organize collision by time
- Preprocessing
  - Examine all collisions ($O(n^2)$)
  - Store the collisions in a heap
- Processing
  - Remove top of heap (corresponding to game time)
  - Determine effect of collision (collision resolution)
  - Compute new collisions (only $O(n)$)
  - Update the heap

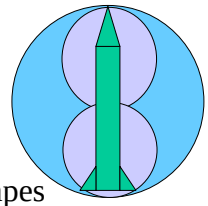
Limitations

- Assumes constant speed
  - Can be modified to use acceleration
- Network games
  - Time may not be synchronized
  - Delay in packet transmission
  - Prediction may be problematic

Addressing complexity

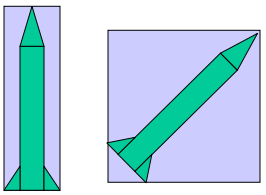
- Simplifying the operations
  - Approximations - sacrifice accuracy
  - Convert object to a simple object
    - Bounding box
    - Sphere
    - E.g., convert a car to a sphere
  - Can be combined in a hierarchy
    - Two levels of spheres
- Convert object to a hierarchy of convex shapes
  - Spheres
  - Boxes

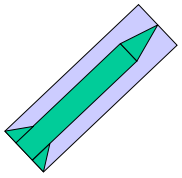




Bounding box

- Usually axis aligned
  - Simple computation
  - Small amount of space
- Limitation
  - Crude approximation
  - High error rate
  - Use hierarchical representation





Addressing a Large Number of Collision Tests (last)

- Brute force / naïve
  - O(n^2) tests per frame
- Parametric computation
  - can be time consuming
  - Require careful bookkeeping
- Partition the space into buckets
  - Sort objects into buckets
  - Test collisions between objects in the buckets

Addressing a Large Number of Collision Tests (last)

- Brute force / naïve
  - $O(n^2)$ tests per frame
- Parametric computation
  - can be time consuming
  - Require careful bookkeeping
- Partition the space into buckets
    - Sort objects into buckets
    - Test collisions between objects in the buckets

Collisions

- When to use an auxiliary data structure
  - Large number of objects
    - What is large? – depending on the computer
  - tradeoffs between collision computation and data structure updating are favourable
  - Space (memory) is available
  - Time restriction
    - Allotted time in a frame – amortized, absolute

Auxiliary DS

Properties
- Handle large number of data elements
- Simple
- Good fundamental operations performance
- Independent of data arrival order
- Little or no data redundancy
- Handle heterogeneous objects

- Possible other properties
  - Handle apriori knowledge (e.g., data distribution)
  - Performance relative to data size

Required Operations

Fundamental Operations
- Insertion
- Deletion
- Update
  - Deletion
  - Insertion
  - Support dynamic

Query Operations
- Range search
  - Proximity
- Ray shooting
  - Ray tracing
- Collisions
  - Proximity

Space partitioning

- A large number of space partitioning schemes
  - Strips
  - 3D Grid
  - Quadtree/Octree
  - BSP tree – binary space partition tree
  - K-D tree
  - R-tree

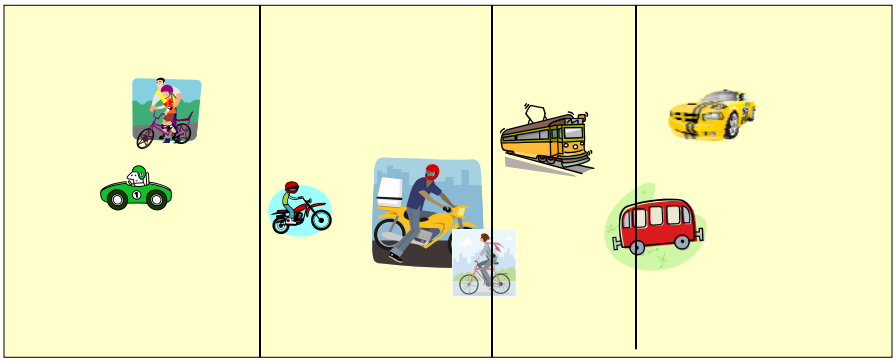
Space partitioning

- Data driven DS
  - Dynamic partitioning
  - DS is modified
  - DS is initially constructed with respect to data
  - Static may lead to poor performance
- Examples
  - BSP tree
  - K-D tree
  - R-tree

- Space driven DS
  - Static partitioning
  - DS is modified
  - DS is initially constructed with respect to data
  - Static may lead to poor performance
  - Dynamic partition may be expensive
- Examples
  - Quadtree/Octree
  - Strips
  - 3D Grid

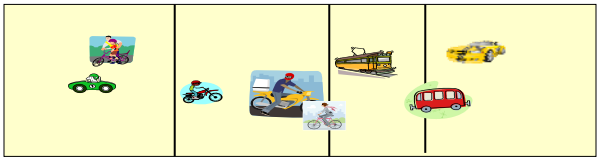
Strips

- Partition the space into strips
  - Key – strip size



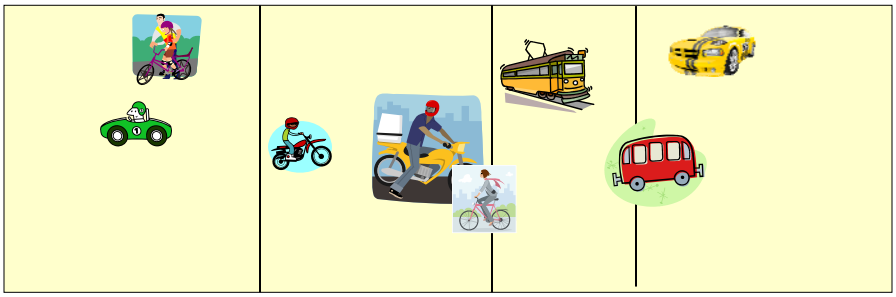
Partitioning

- Two types of objects
  - Guard objects
    - Shared between two or more buckets
  - Contained objects
    - Contained in a single bucket



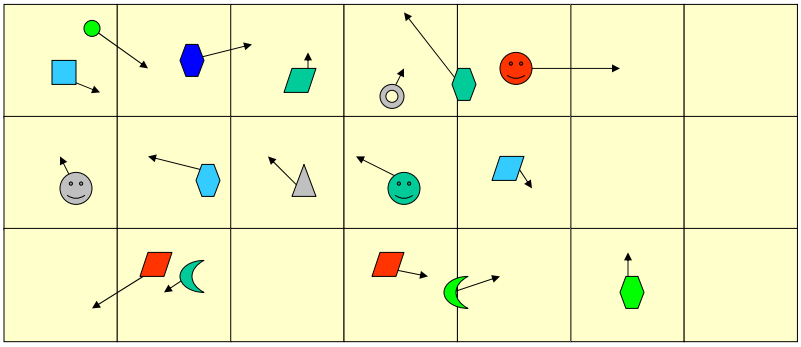
Strips

- Simple
  - Use bounding box to determine bucket
  - As object moves update the buckets
- In determining strip size – min strip size > 2 * max object width
  - Avoid collision detection of objects in non adjacent buckets.



3D Grid

- Similar to strips but two dimensional
  - Key – determining the grid size
  - Guard objects are shared by at most four backets



Collisions - Parametric

- Compute collisions within each buckets
- Organize each bucket in a priority queue, Q_b
- Organize a priority queue of all buckets, Q

Priority Queue Events

- Insertion (repeat for a guard)
  - Insert to bucket
  - Compute collisions with all bucket elements
  - Insert earlier collision to Qb
- Deletion object u (repeat for a guard)
  - Remove object from bucket
  - Search object v in Q that collides with u
  - Compute next collision for v

Priority Queue Events

- Collision between u and v (repeat for a guard)
  - Delete u
  - Delete v
  - Determine collision response
  - Insert u
  - Insert v
- Boundary
  - Make object a guard
  - Special insert into new bucket

Fast Moving Objects – Ray Query

- For each ray
  - Check entry and exit time