Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

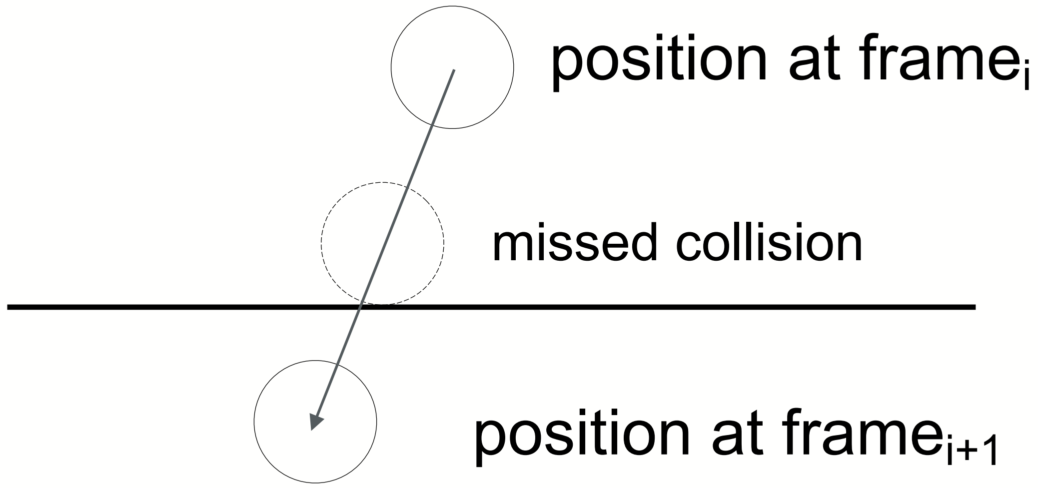
Brian Russin

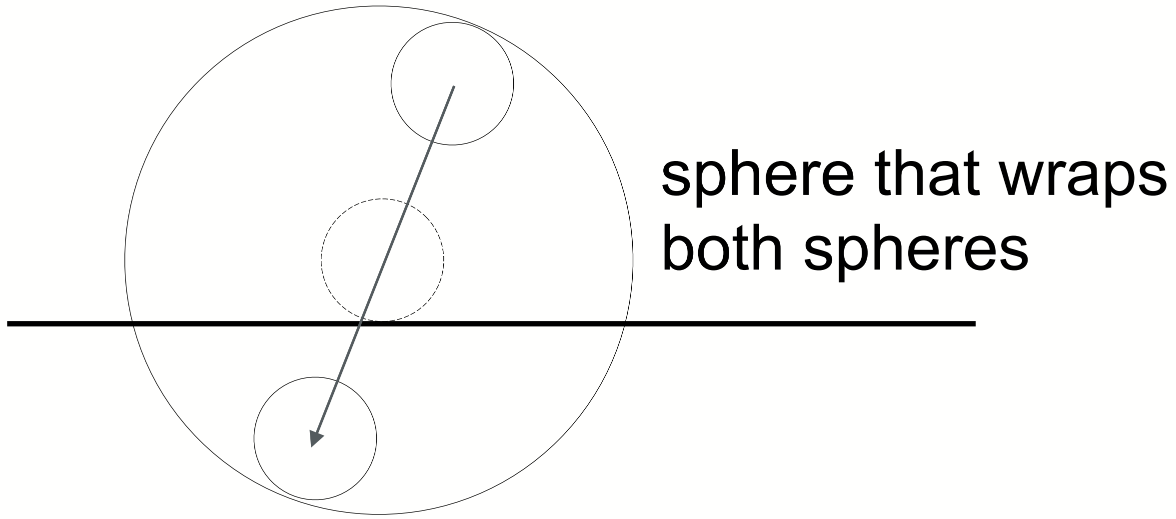
JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Module 10D
Dynamic Intersections

Dynamic Intersection Testing

- With moving objects we render frames at discrete times
- Discrete collision detection not effective
  - Ball on one side of closed door at time t and other side at t = t + dt
    - May not detect a collision with static intersect tests
    - Sometimes called **quantum tunneling**
  - One solution
    - Make several tests at uniform intervals between t and t + dt
      - Increases computational load – may still miss collision
  - Better solution
    - Create BV that encloses geometry at $frame_i$ and $frame_{i+1}$



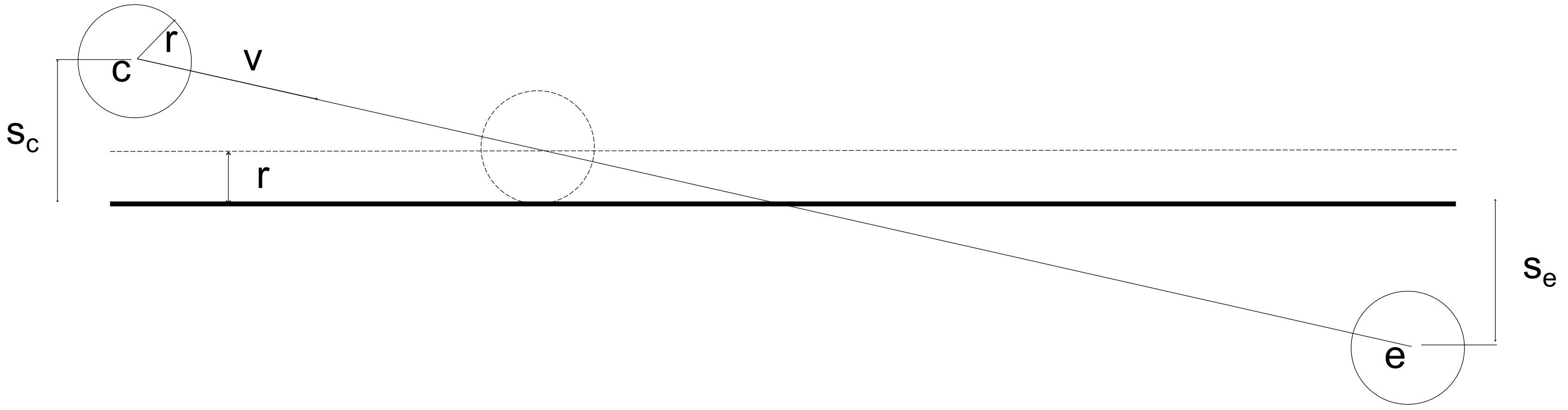


Dynamic Intersection Testing (cont.)

- Dynamic intersection tests can be constructed
  - Simpler and more efficient methods if moving object is enclosed within bounding sphere
    - Can also use a set of spheres to represent the moving object
- Simplify calculations by considering relative motion
  - If 2 objects are moving: v_A and v_B are velocities of object A and B
  - Simplify calculations by considering A is moving and B is still
    - A's velocity is then represented as $v = v_A - v_B$

Sphere / Plane

- Assume sphere has velocity v for entire frame time dt
  - At next frame sphere will be located at e = c + (dt)v
    - For simplicity assume dt = 1
- Find signed distance (d) from plane
  - Plugging sphere center into plane equation
    - Do the same for position e
  - If sphere centers are on same side of the plane and distances both greater than r then no intersect can occur
  - Otherwise intersect occurs at time where sphere first touches plane:
    - Sphere center is located at c+tv
- Simple collision response: reflect v around the plane normal
  - Move sphere along this vector from collision point: (1-t)r



$$t = \frac{s_c - r}{s_c - s_e}$$

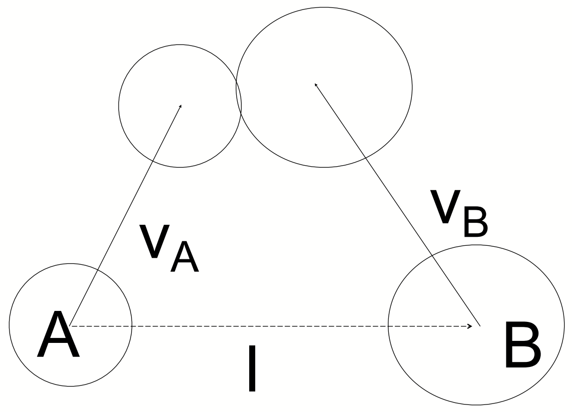
Sphere / Sphere

- Testing intersection of 2 moving spheres reduces to testing a ray intersect with a sphere!
  - Can then use ray-sphere intersection test developed earlier
    - To find whether an intersect occurs and the nearest t value of intersect
  - Haines and Moller present alternate solution in Section 16.18.2 of 3rd Edition
    - Omitted in the 4th Edition
    - Does not require v_{AB} to be normalized
    - Values in quadratic equation are: $a = v_{AB} \cdot v_{AB}$ $b = 2(l \cdot v_{AB})$ $c = l \cdot l - (r_A + r_B)^2$

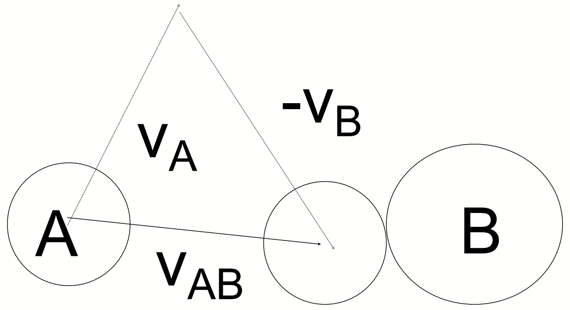
$$q = -\frac{1}{2}\left(b + \text{sign}(b)\sqrt{b^2 - 4ac}\right)$$

sign(b) = 1 when b >=0, else b = -1

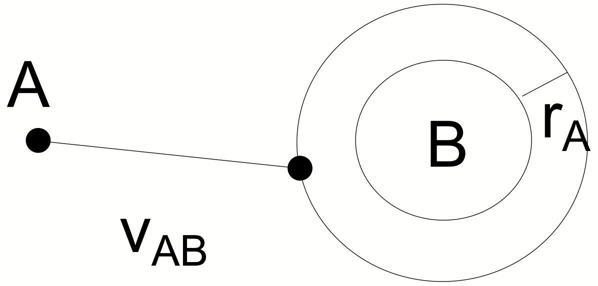
Roots are: $t_0 = \frac{q}{a}$ $t_1 = \frac{c}{q}$



Two moving spheres colliding



Make sphere B stationary – use relative velocity



Radius of A added to sphere B and subtracted from sphere A. Turns moving sphere A into a ray

Sphere / Polygon

- Intersecting a moving sphere with a polygon
  - More involved than sphere/plane intersection
  - Haines and Moller detail an approach by Schroeder
    - Only in 3rd Edition, omitted in 4th Edition
- Sphere/plane test finds where the sphere first hits the plane
  - Quick reject test: if sphere/plane intersect reveals sphere does not intersect the plane
  - Intersection point can be used in a point in polygon test
    - If intersect point is in polygon the sphere intersects at this point
  - However, intersect point can be outside the polygon but polygon could still intersect a sphere edge/vertex further along its path
    - Sphere/edge test is computationally complex
  - Sphere/polygon test is same as testing sphere against a “puffy” polygon
    - Sphere-swept polygon

General Hierarchical Collision Detection

- Haines and Moller describe general methods for hierarchical collision detection
  - Build a representation of each model hierarchically using BVs
  - Similar high-level code for collision query is used – regardless of BV
    - BV/BV overlap tests and primitive/primitive overlap tests differ
    - Offer pseudo-code for testing between hierarchies (Section 17.3.2 of 3rd Edition)
  - Simple cost function can be used to compare performance tradeoffs
    - $t = n_v c_v + n_p c_p + n_u c_u$
      - n_v – number of BV/BV overlap tests
      - c_v – cost of BV/BV overlap test
      - n_p – number of primitive pairs tested for overlap
      - c_p – cost of testing primitive overlap
      - n_u – number of BVs updated due to motion of object
      - c_u – cost of updating a BV
    - Note: in general tighter BVs have more costly BV/BV overlap test (c_v)
      - Looser BV results in more primitive pairs requiring testing (n_p)