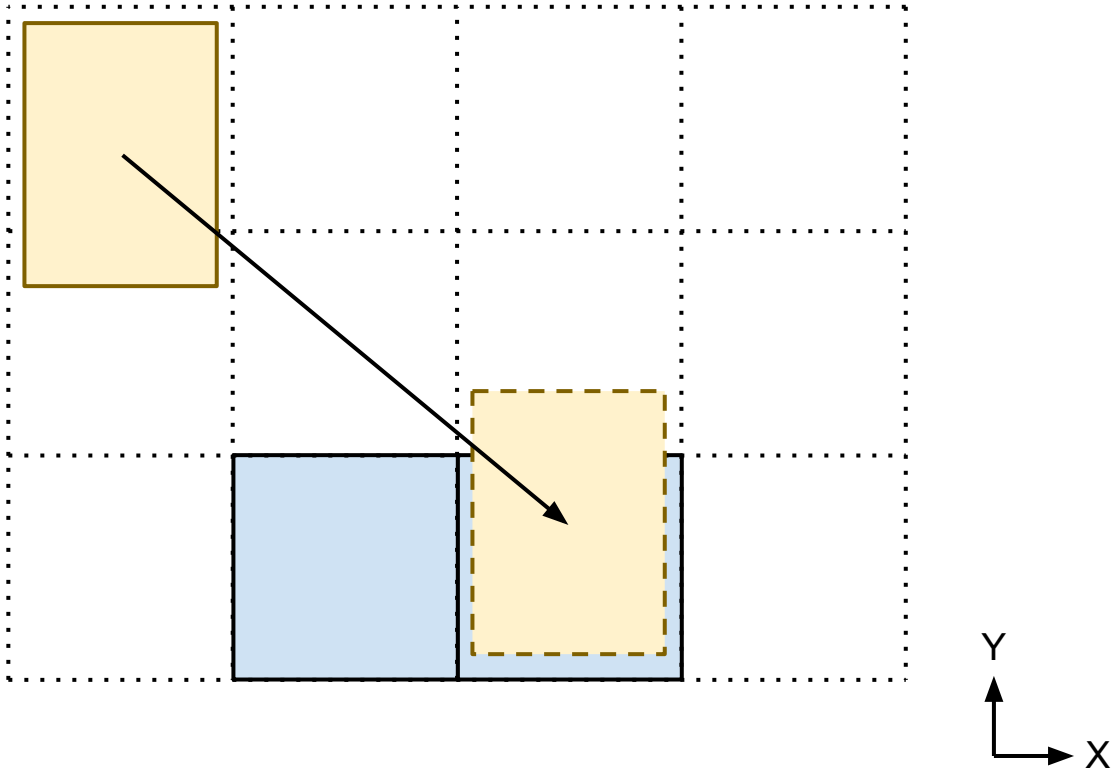

Collision Detection for
Player Movement

Minecraft Week 2 Comments

- Some of you implemented iterative collision detection
  - Collision response will be problematic when moving at high speeds
  - More details later this lecture
- Others implemented the algorithm from lecture incorrectly
  - Need to update the player's position after moving along as axis
  - Examples on next few slides

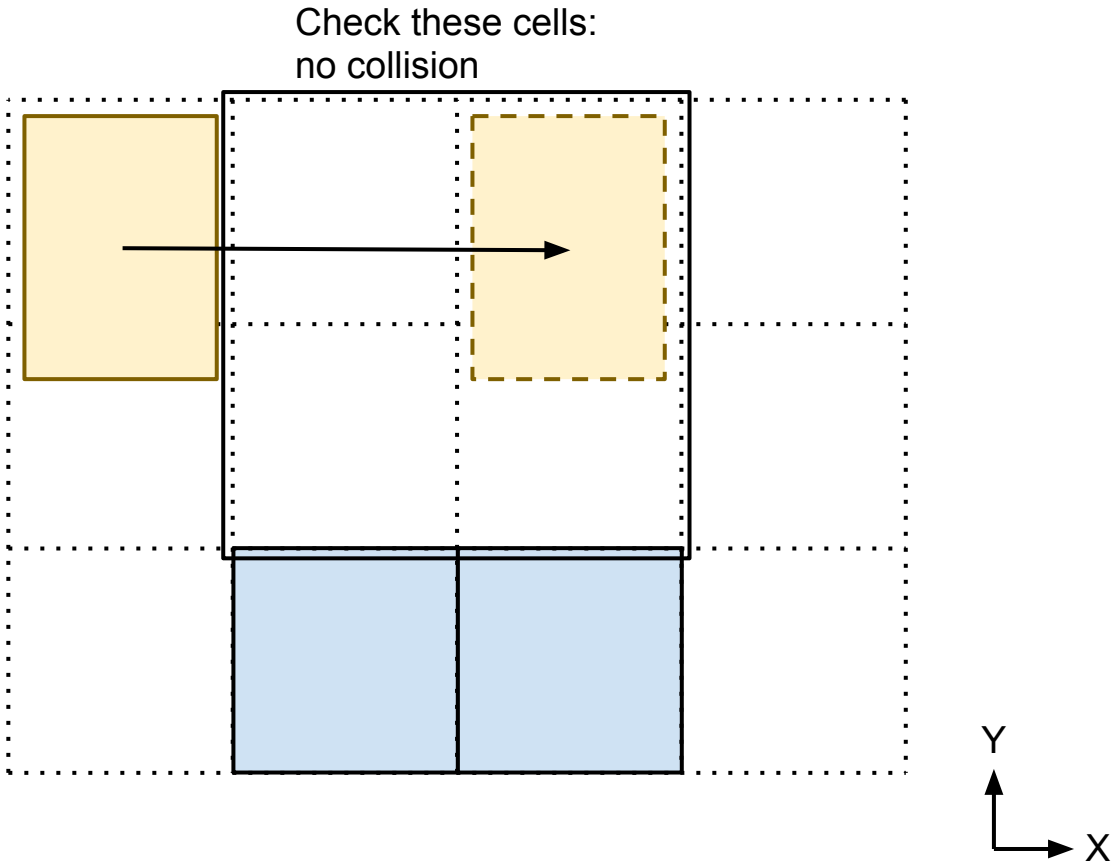
Incorrect Collision Detection

- Player moving diagonally down and to the right



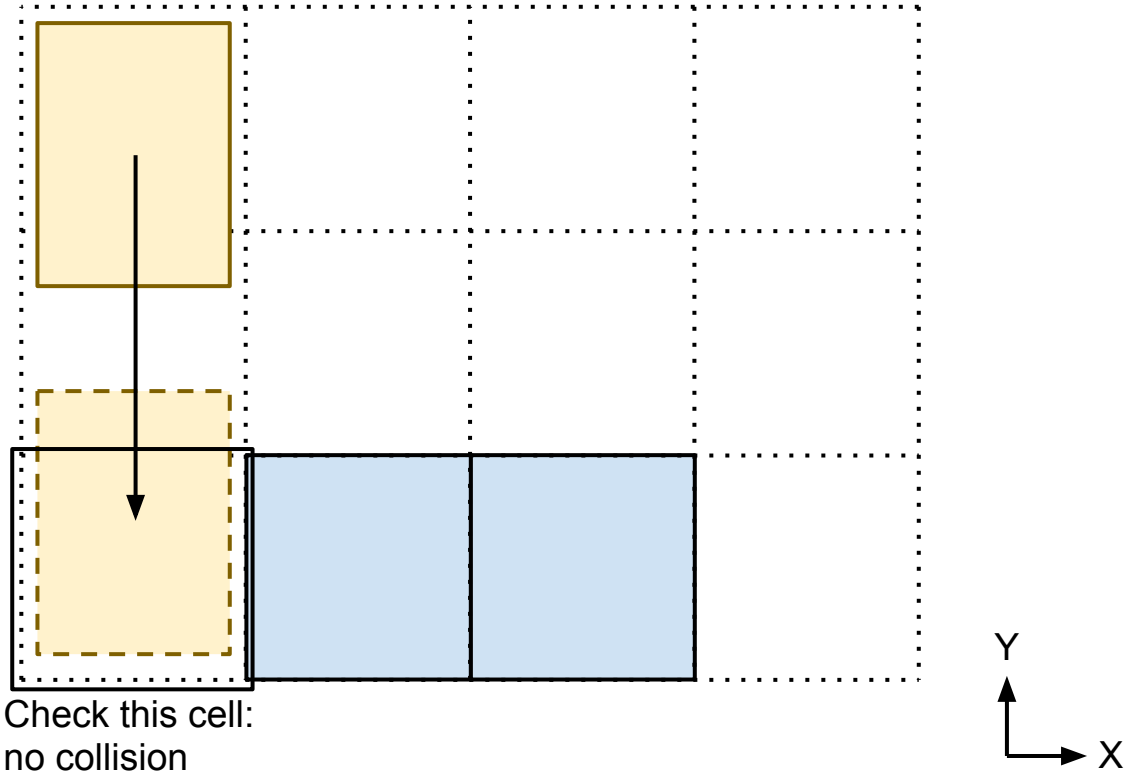
Incorrect Collision Detection: X-axis

- Player moving diagonally down and to the right



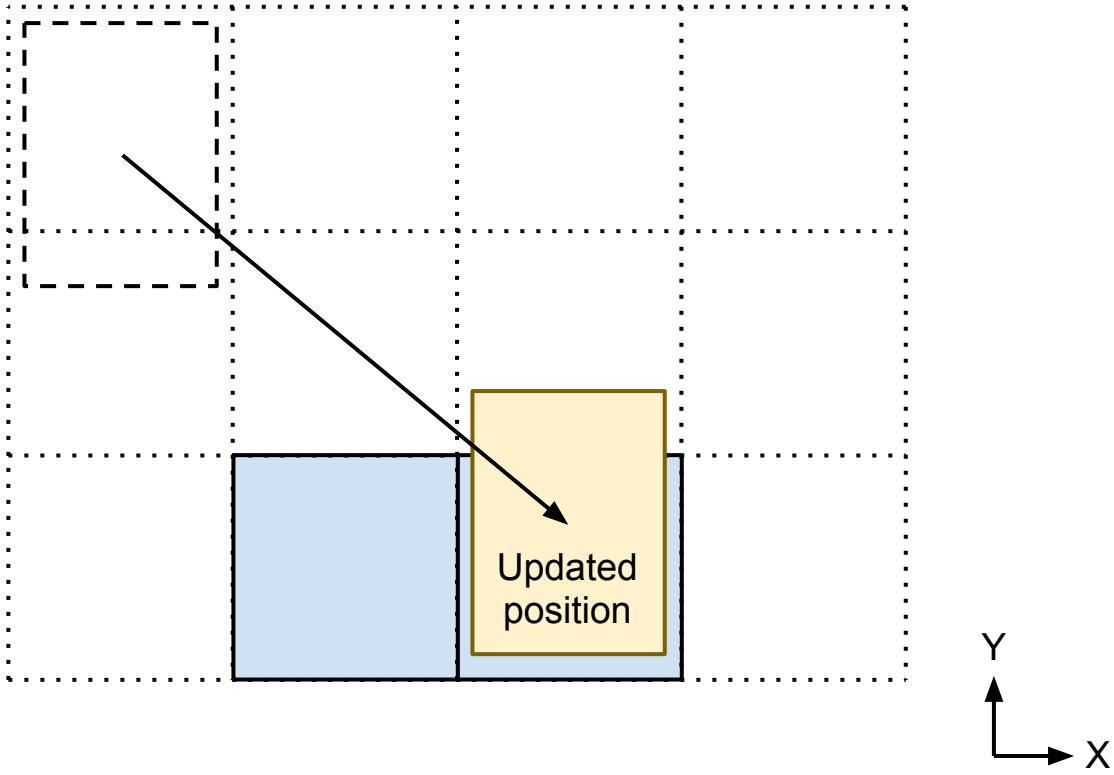
Incorrect Collision Detection: Y-axis

- Player moving diagonally down and to the right



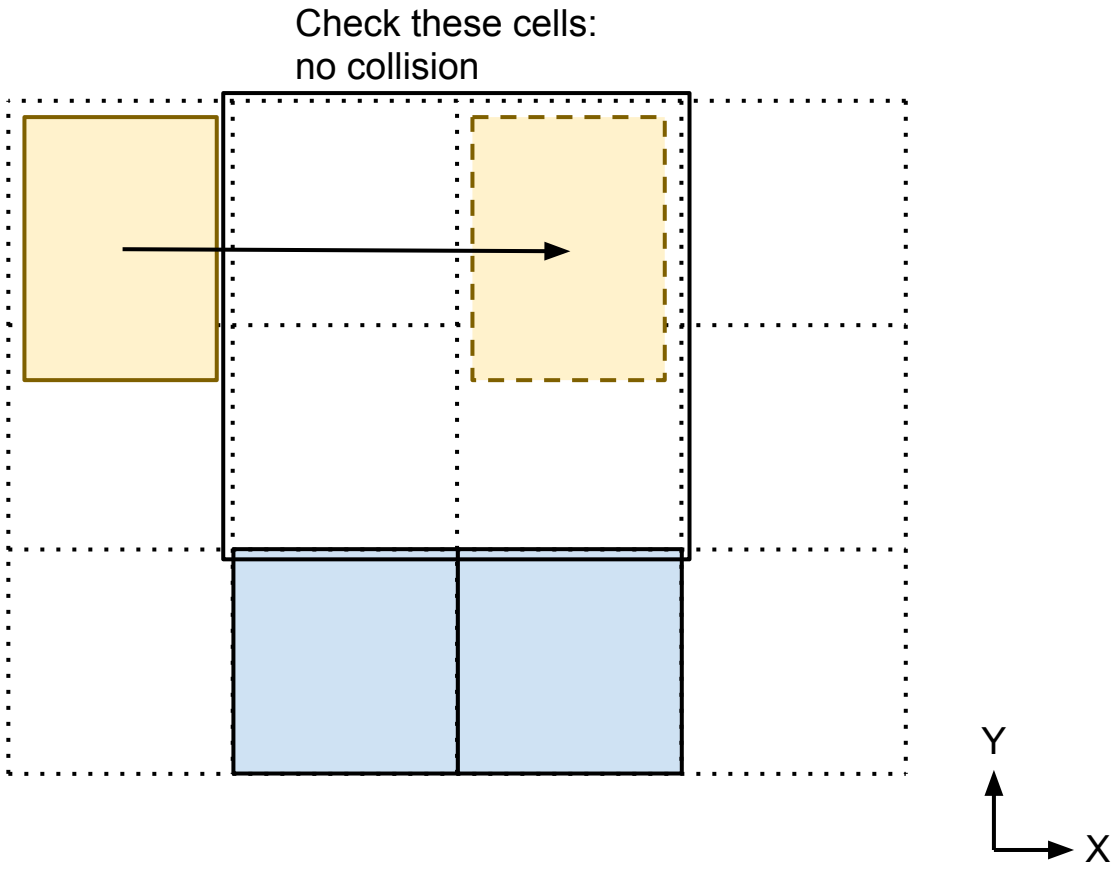
Incorrect Collision Detection: Result

- Player moving diagonally down and to the right



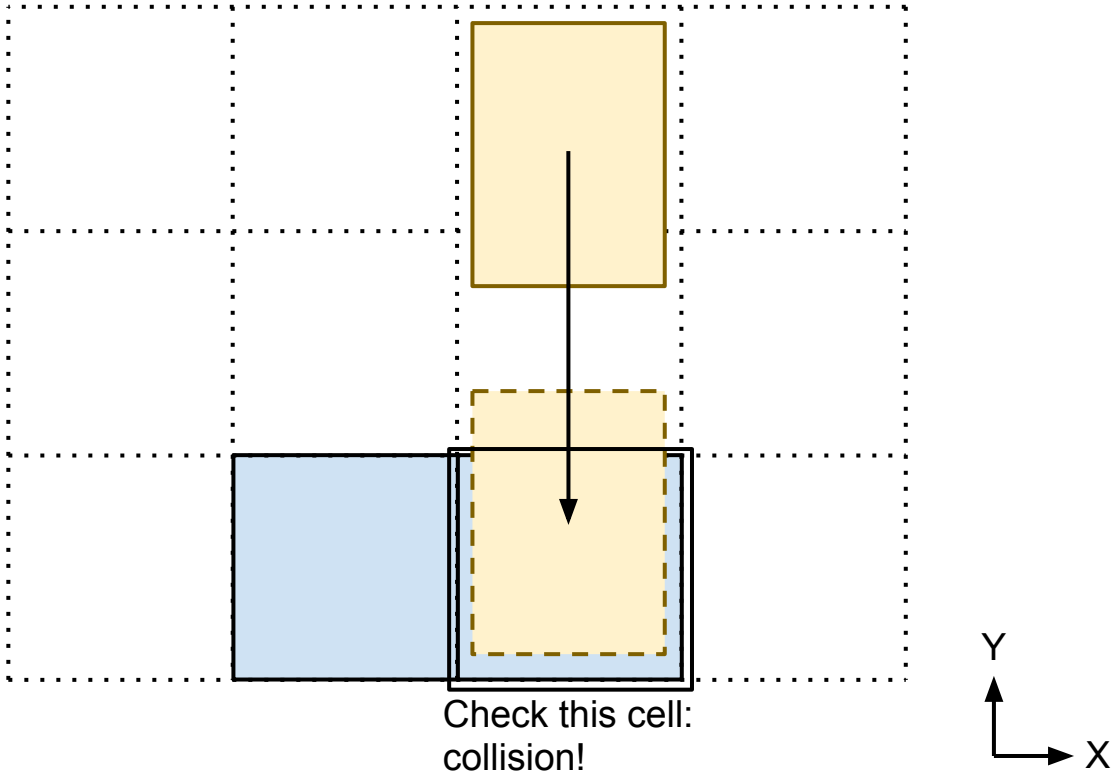
Correct Collision Detection: X-axis

- Player moving diagonally down and to the right



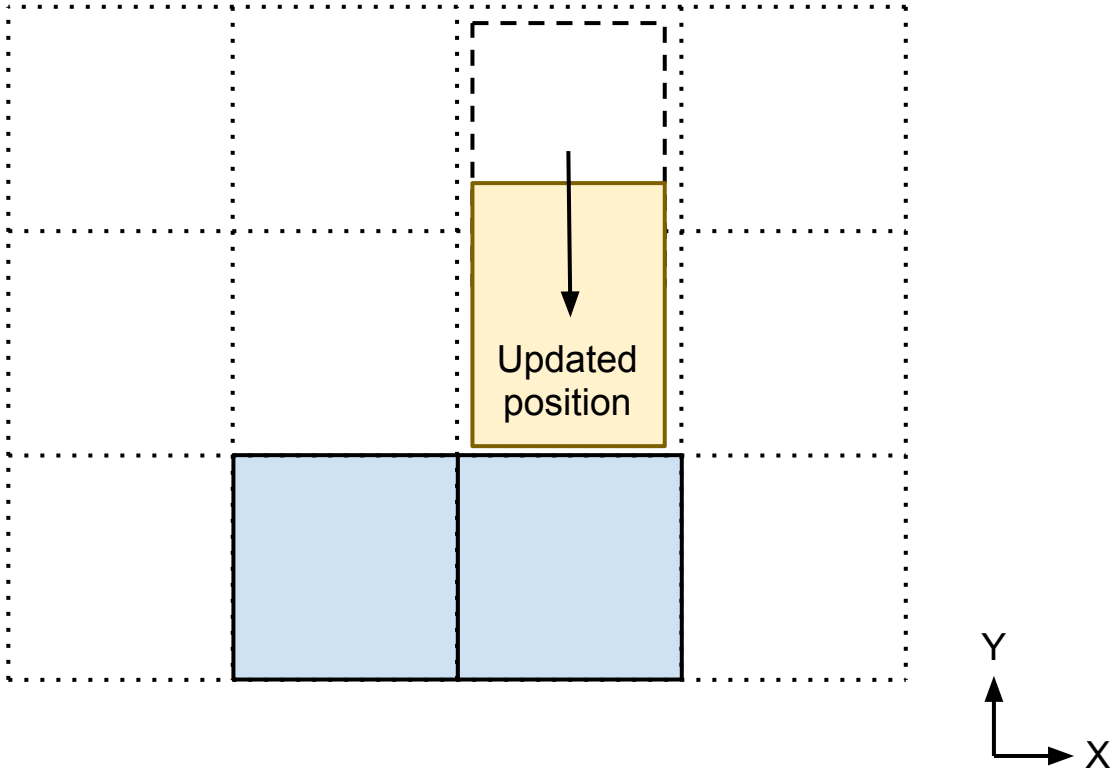
Correct Collision Detection: Y-axis

- Player moving diagonally down and to the right



Correct Collision Detection: Result

- Player moving diagonally down and to the right



Collision Detection for Player Movement

How to represent player movement?

- Real world: left foot, right foot, ...
  - Not natural with mouse and keyboard
  - QWOP in 3D
- Simplify the problem
  - 3D primitive representing player volume
  - Animation state a side effect of collision state
  - Which 3D shape to choose?
- Many ways to do this and no right answer!

Player Movement Differences

- Player movement collision model is not used for entity-to-entity collisions
  - For example, bullets in a FPS would use a more accurate collision model with a bounding volume per body part
- Player movement collision model is not used for physics
  - Player movement is different than physics
  - Player's bounding volume will be locked to the upright position, represents player *intent* while moving

Shape: Axis-Aligned Box

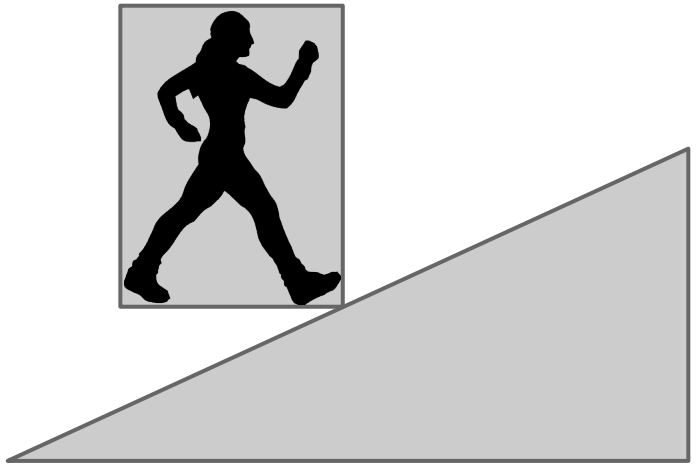
Pros:

- Simple collision tests for axis-aligned worlds

Cons:

- Player doesn't have the same diameter in all directions
- Complicated collision tests for arbitrary worlds
- Stairs will need special handling
- Player will hover on slopes

Shape: Axis-Aligned Box



AABB "hovering" on a slope

Shape: Cylinder

Pros:

- Player has the same diameter in all directions

Cons:

- Collision tests complicated by caps
- Stairs will need special handling
- Player will hover on slopes

Shape: Upside-Down Cone

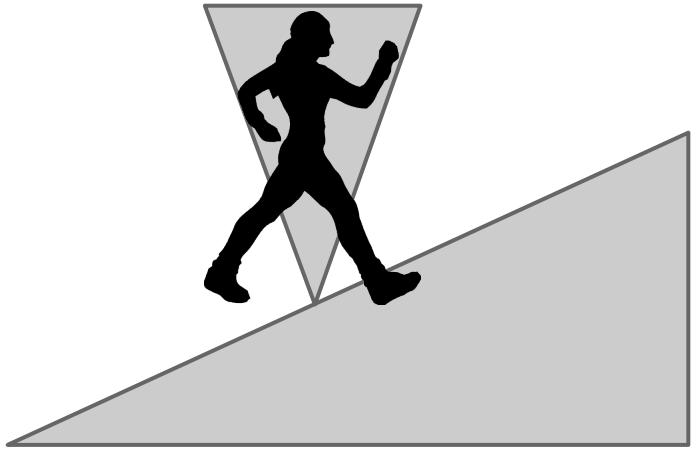
Pros:

- Naturally climb up stairs (and objects less than the player's height)
- Player doesn't hover on slopes

Cons:

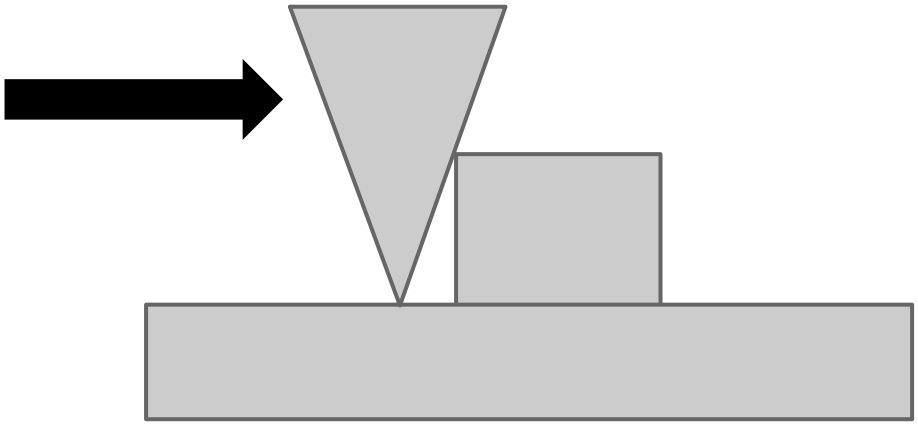
- Complicated collision tests
- Sliding doesn't make sense for some games

Shape: Upside-Down Cone



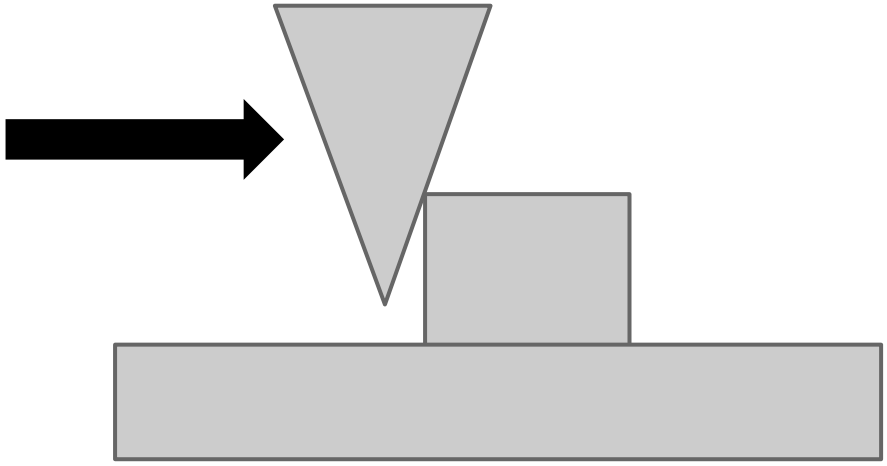
Cone doesn't hover on a slope

Shape: Upside-Down Cone



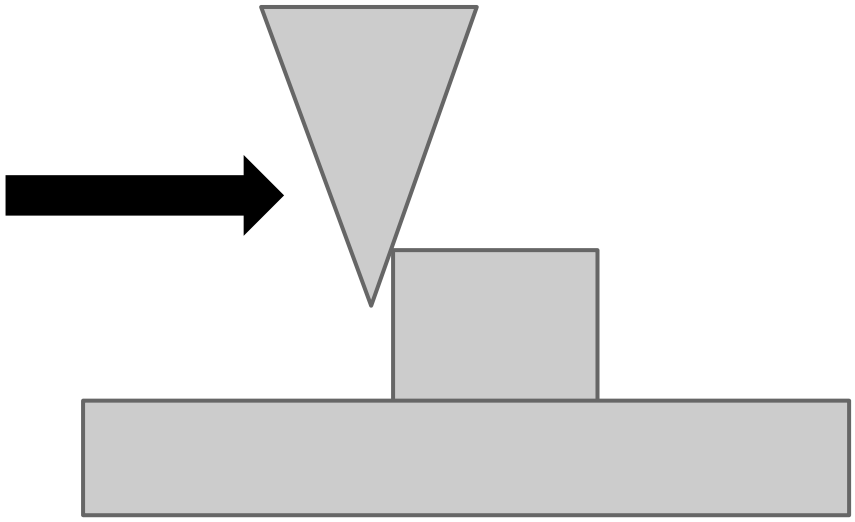
Cone naturally climbs objects
when pushed against them

Shape: Upside-Down Cone



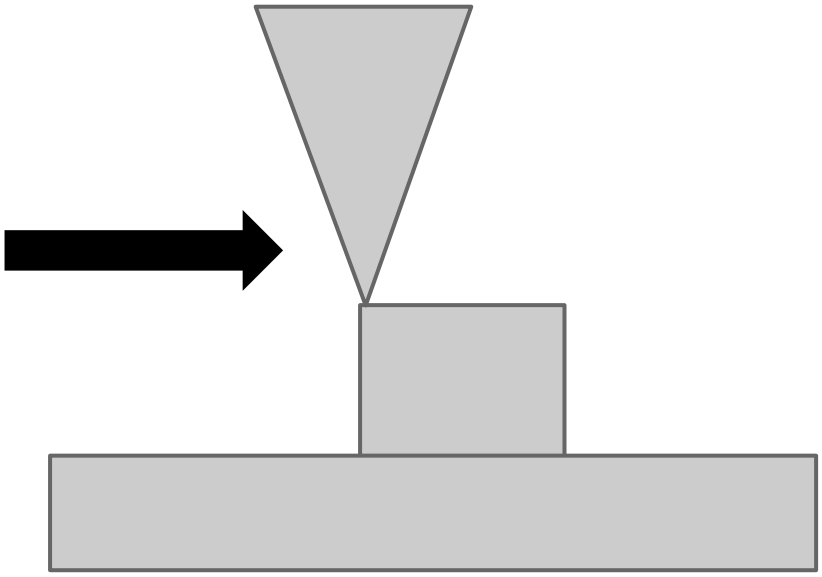
Cone naturally climbs objects
when pushed against them

Shape: Upside-Down Cone



Cone naturally climbs objects
when pushed against them

Shape: Upside-Down Cone



Cone naturally climbs objects
when pushed against them

Shape: Ellipsoid

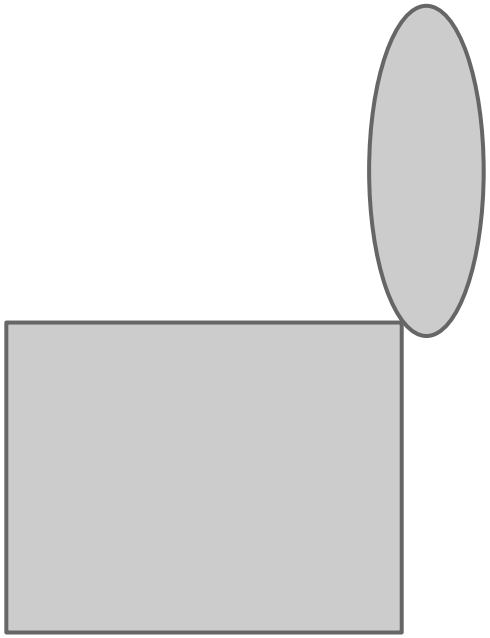
Pros:

- Collision tests are simpler than cylinder
- Player will be closer to the ground on slopes
- Player will naturally slide up stairs

Cons:

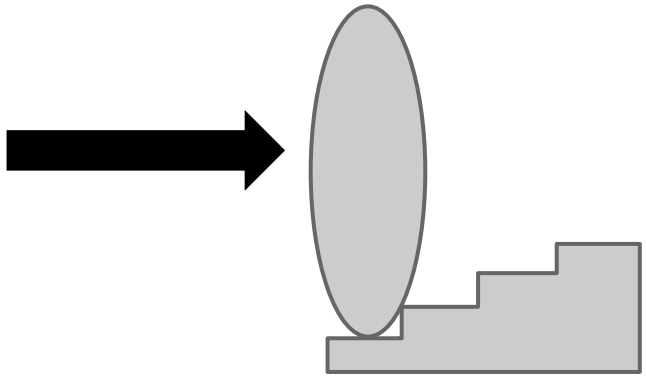
- Player will "dip" down a bit before going off an edge

Shape: Ellipsoid



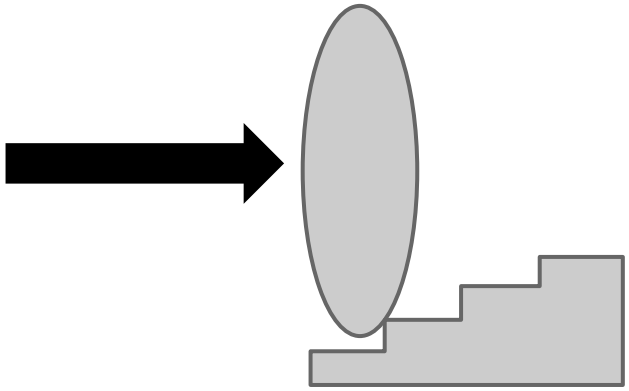
Ellipsoid will "dip" down over a ledge

Shape: Ellipsoid



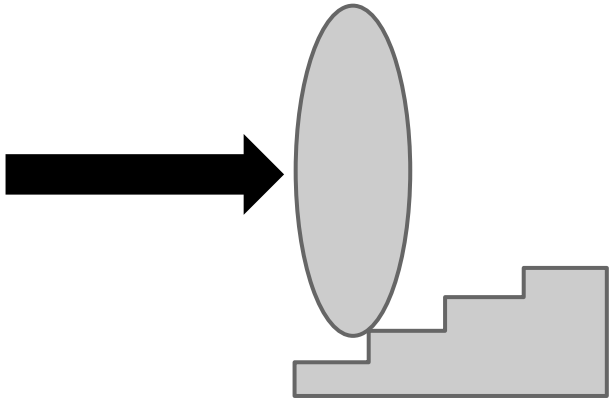
Ellipsoid will automatically climb up
stairs when pushed against them

Shape: Ellipsoid



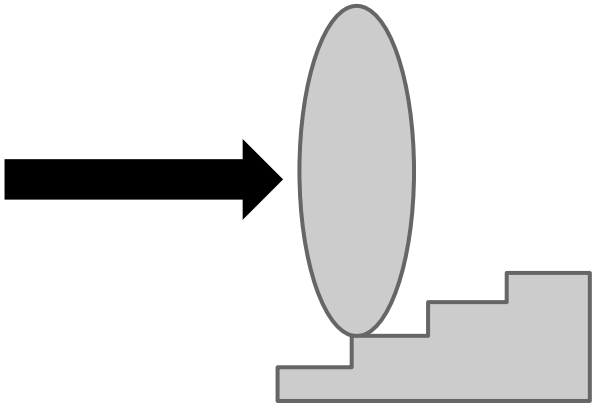
Ellipsoid will automatically climb up
stairs when pushed against them

Shape: Ellipsoid



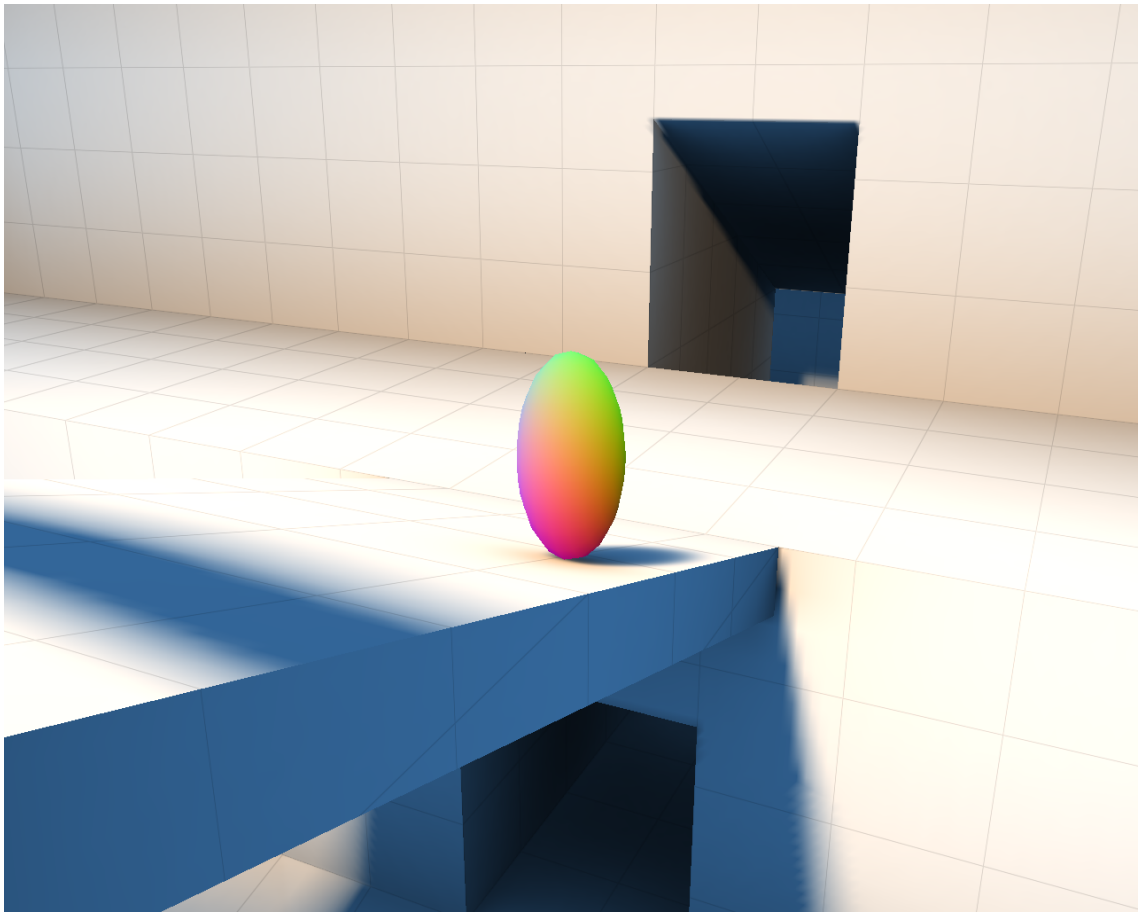
Ellipsoid will automatically climb up
stairs when pushed against them

Shape: Ellipsoid



Ellipsoid will automatically climb up
stairs when pushed against them

Ellipsoid Collision Detection Demo

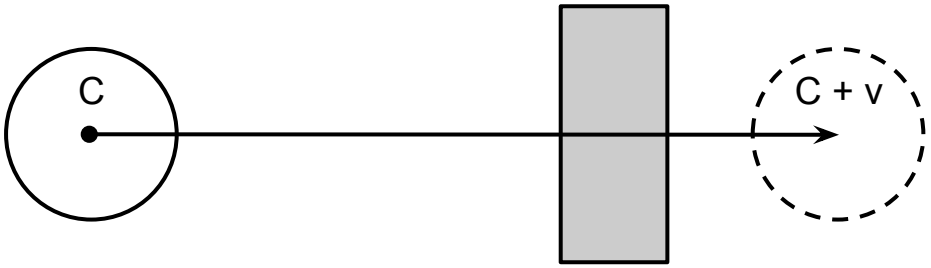


Collision Detection Strategies

- *Static* collision detection is given two (or more) objects and determines if they are overlapping
- *Dynamic* collision detection is given two (or more) objects and determines at what time t they will collide
- Two categories of dynamic collision detection
  - *Iterative*: Sample at points along a path and perform static queries
  - *Analytic*: Compute the exact point of collision using parametric equations

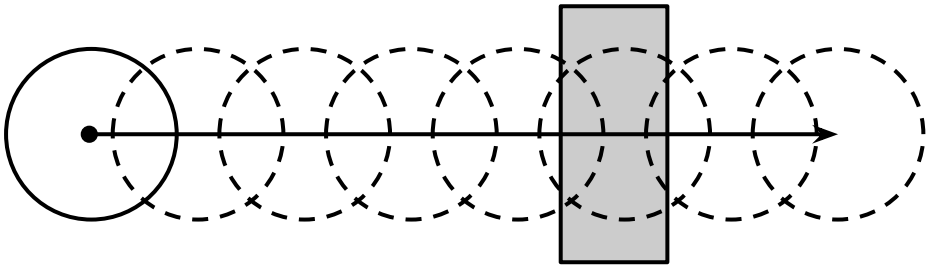
Static Collision Detection

- Appropriate for whenever objects aren't moving
- Also appropriate for large, slowly moving objects
  - Especially when no collision response is needed
- Tunneling is a problem: moving entirely through an object in the span of one frame
  - Collision detection starts failing when frame rate drops!
- May be difficult to generate appropriate collision response



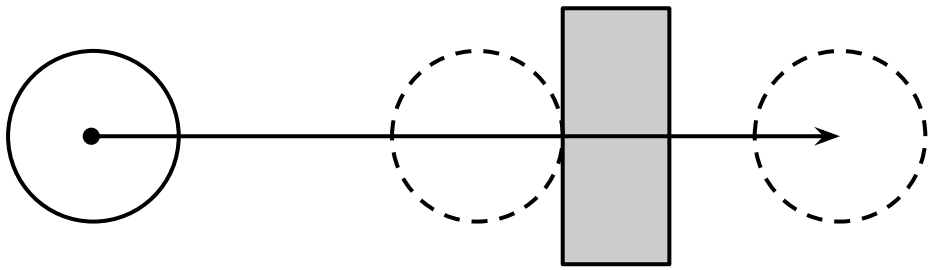
Iterative Collision Detection

- Move in small increments and repeatedly query
- Easy to deal with complex curving paths
- Tunneling is possible if increment size is too large
  - Run collision detection at a fixed frame rate to avoid this
- Expensive to repeatedly query



Analytic Collision Detection

- Formulate path as a parametric equation, solve for intersection
- Limited to simple formula (straight lines)
- Faster than iterative collision detection
- Eliminates tunneling



Player-World Collision Detection

Problem:

- Given a world specified with triangles and player modeled as an ellipsoid
- Do I hit something moving from A to B? Where?

Solution:

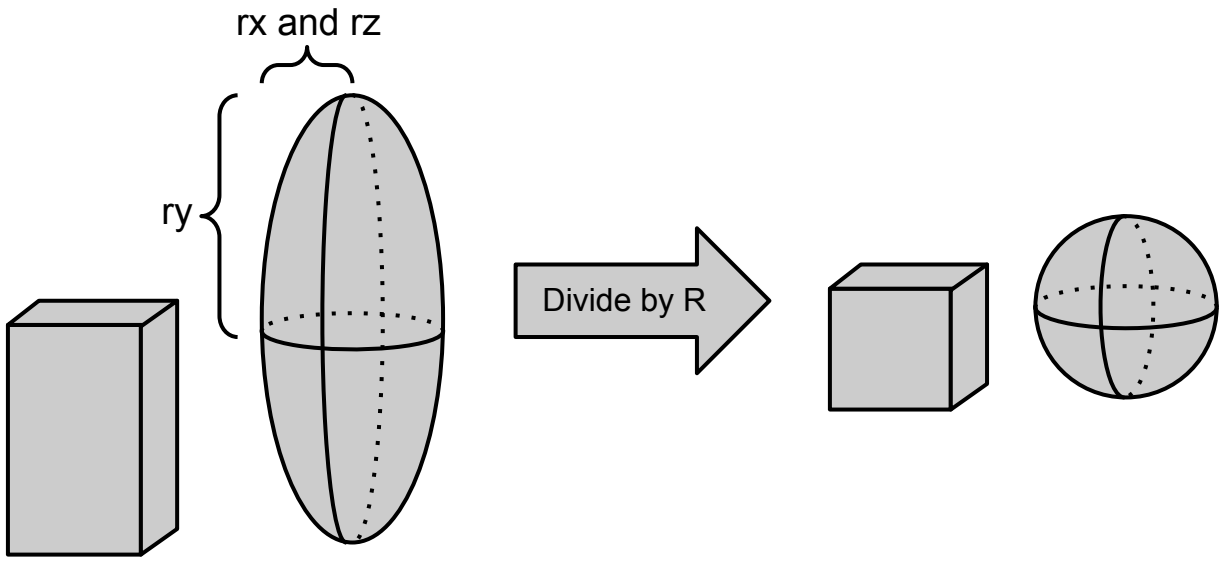
- Since tests are limited to ellipsoids and triangles, we can will derive an analytic solution
- The rest of the lecture goes through the math
- You will be implementing this!
- Iterative solution could also work for this scenario, though we won't cover it in depth

Player-World Collision Detection

- Assume player moves in a straight line in one update step
  - Math for a line is much simpler
  - Raycast from starting position to new position
- Do triangle-ellipsoid sweep test for all triangles in the level and take the closest one
  - Can optimize using spatial acceleration data structure to only test relevant triangles
- We will now talk about colliding an ellipsoid with a single triangle

Analytic Ellipsoid-Triangle Collision

- Sphere intersection tests are easier
  - Squash world so ellipsoid is a unit sphere
  - Do collision detection in that space, then convert back
- Change of vector spaces
  - Ellipsoid has radius R = (r_x,r_y,r_z)
  - Use basis (r_x,0,0), (0,r_y,0), (0,0,r_z)
  - Ellipsoid space to sphere space: divide by R

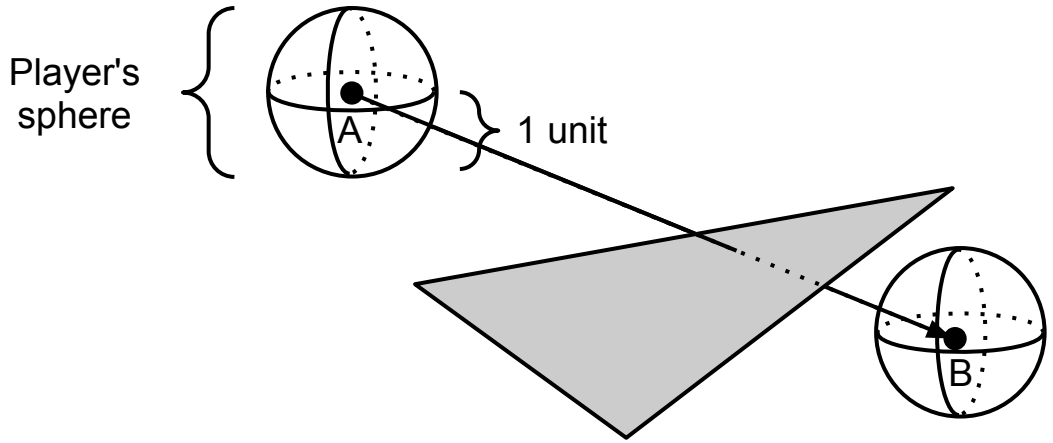


Analytic Ellipsoid-Triangle Collision

- Analytic equation for a moving sphere
  - Unit sphere moving from A to B
  - Center: $A + (B - A)t$
  - Point P on surface at t if $||A + (B - A)t - P||^2 = 1$
- Will be solving for t in sphere space
  - Value of t will also be correct for ellipsoid space
- Split collision detection into three cases:
  - Sphere and interior
  - Sphere and edge
  - Sphere and vertex

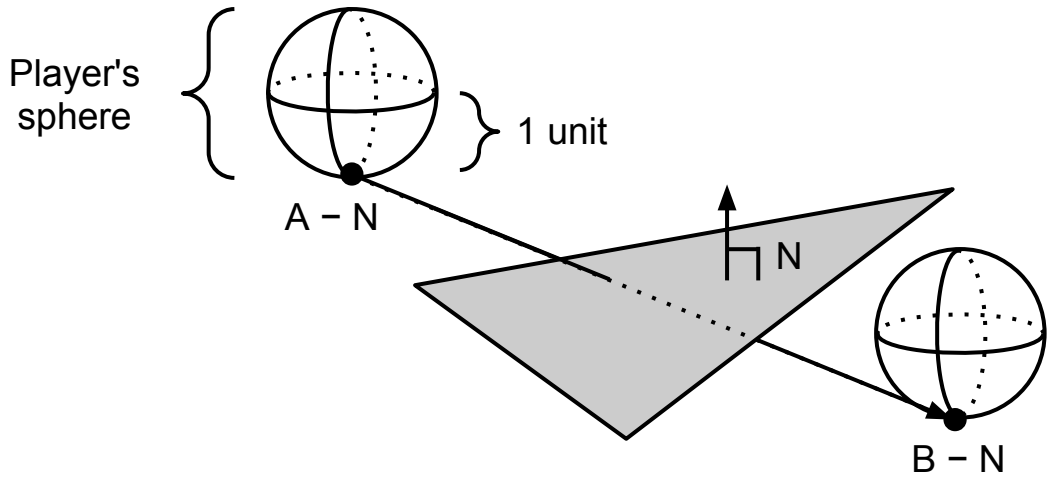
Analytic Sphere-Interior Collision

- Intersect moving sphere with plane
- If intersection is inside triangle
  - Stop the collision test, an interior collision will always be closer than a vertex or edge collision
- If intersection is outside triangle
  - Continue on to testing against edges and vertices



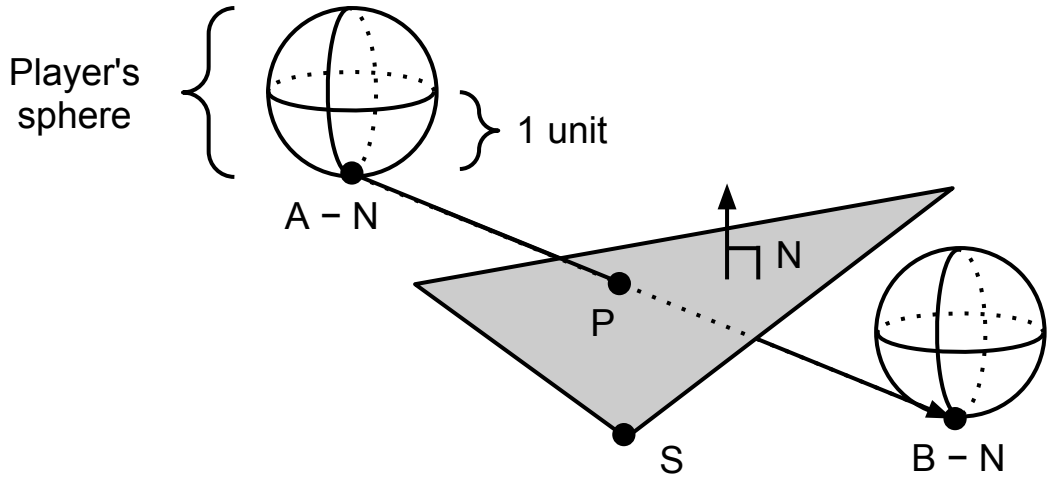
Analytic Sphere-Interior Collision

- Sphere-plane intersection
  - Equivalent to ray-plane intersection using the point on the sphere that is closest to the plane
  - Given plane with normal N, closest point is A - N



Analytic Sphere-Interior Collision

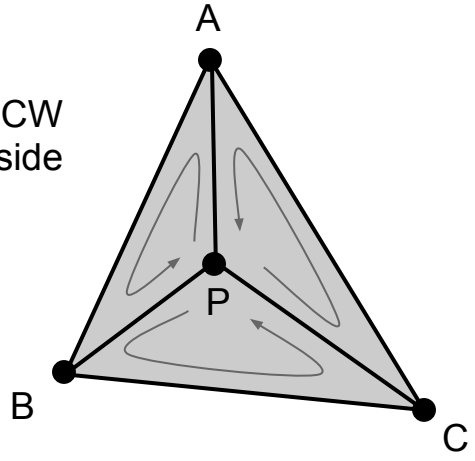
- Point P on plane if N • (P − S) = 0
  - Where S is any point on the plane
- Set P = (A − N) + (B − A)t
- Solve N • [(A − N) + (B − A)t − S] = 0 for t
  - That means t = −[N • (A − N − S)] / [N • (B − A)]

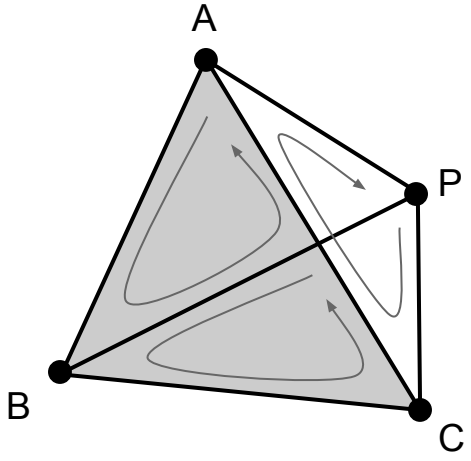


Point-in-Triangle Test

- Point P (on plane) is inside triangle ABC if the sub-triangles PAB, PBC, and PCA are all clockwise or all counterclockwise

PAB, PBC, PCA are CCW
P is inside





PAB, PBC are CCW
PCA is CW
P is outside

Point-in-Triangle Test

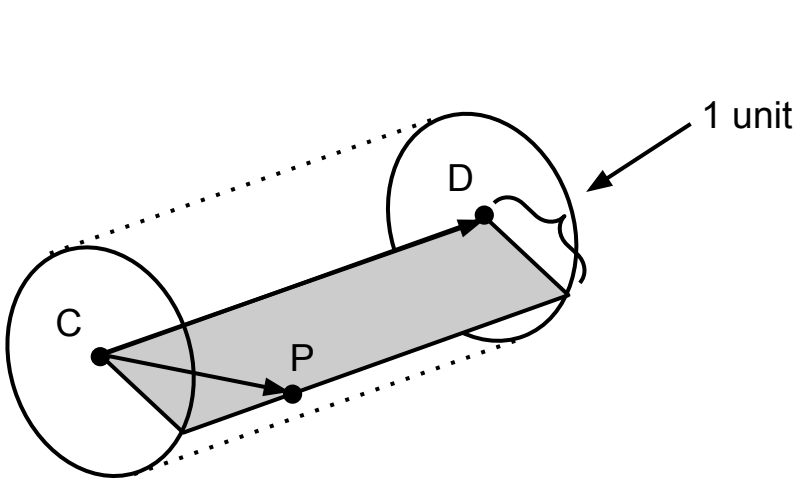
- Sub-triangles have same winding order (CW or CCW) if normals are in same direction
  - Normal is cross product of first edge with second edge
  - Can compare two normals using their dot product
    - Positive dot product: same direction
    - Negative (or zero) dot product: opposite direction

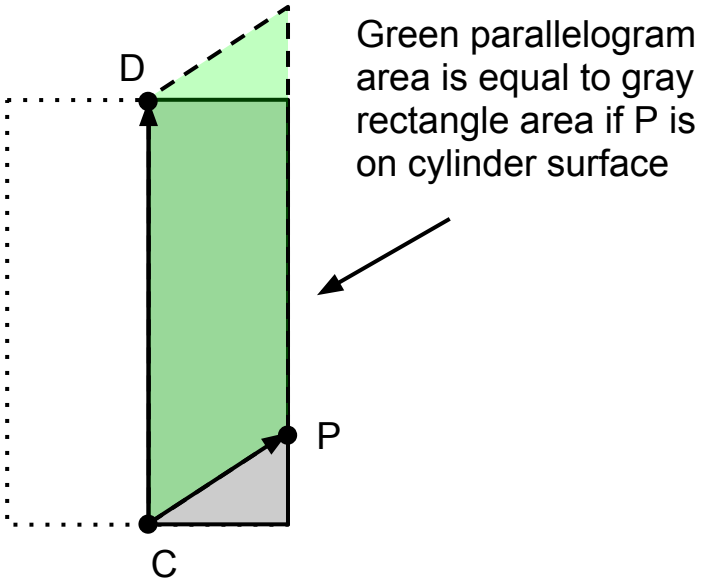
Analytic Sphere-Edge Collision

- Edge of triangle = line segment
- Intersect moving sphere with the infinite line containing the edge
- Reject intersection if it occurs outside the line segment
- How to collide moving sphere with line?
  - Really just a line and a ray that "collide" at a certain distance apart
  - If we treat line as infinite cylinder, we can use ray-cylinder intersection

Analytic Sphere-Edge Collision

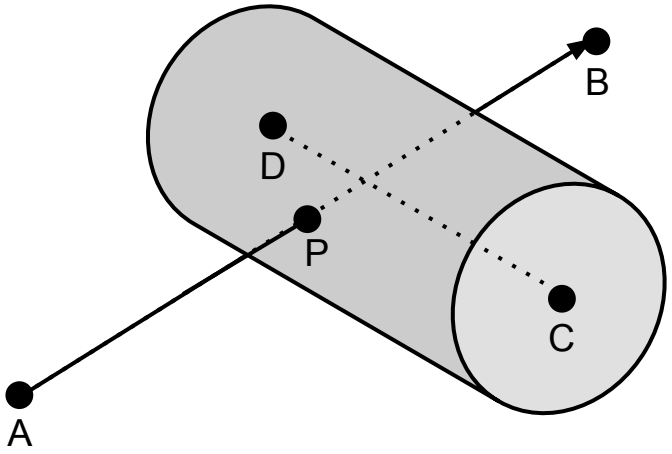
- Finding point P on surface of infinite cylinder
  - Given two points C and D along cylinder axis
  - Point P on surface if $||(P - C) \times (D - C)||^2 = ||D - C||^2$
  - Length of cross product = area of parallelogram formed by the crossed vectors





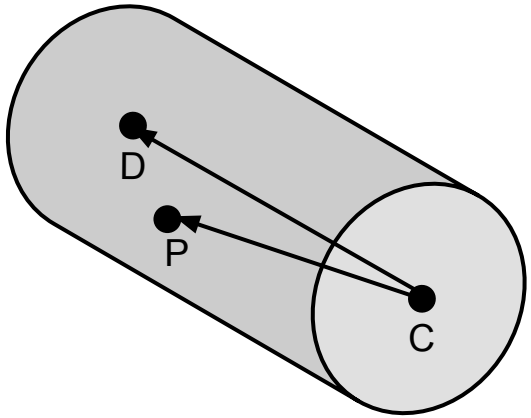
Analytic Sphere-Edge Collision

- Set $P = A + (B - A)t$
- Solve $||(A + (B - A)t - C) \times (D - C)||^2 = ||D - C||^2$ for t
- Looks like $at^2 + bt + c = 0$ where
  - $a = ||(B - A) \times (D - C)||^2$
  - $b = 2[(B - A) \times (D - C)] \bullet [(A - C) \times (D - C)]$
  - $c = ||(A - C) \times (D - C)||^2 - ||D - C||^2$
- Solve using quadratic equation, use lesser t value



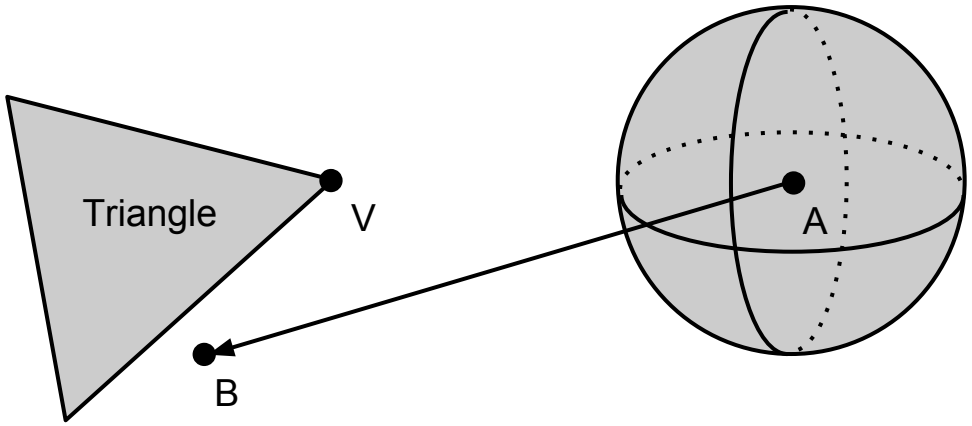
Analytic Sphere-Edge Collision

- Discard intersection if not between C and D
  - Will be handled by vertex collision test
- To check if intersection is between C and D:
  - Get vector from C to intersection point P
  - Project this vector onto cylinder axis
  - Keep intersection if $0 < (P - C) \bullet (D - C) < ||D - C||^2$



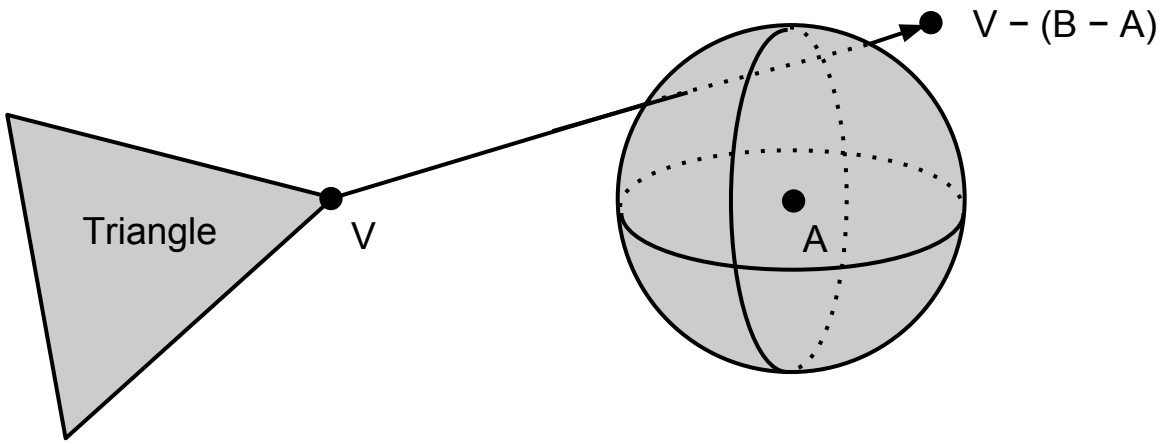
Analytic Sphere-Vertex Collision

- Collision test against a triangle vertex V
- How to collide moving sphere against point?
  - We know how to do a ray-sphere intersection test
  - Moving sphere vs point equivalent to sphere vs moving point (in the opposite direction)
  - Really just two points that "collide" at a certain distance apart



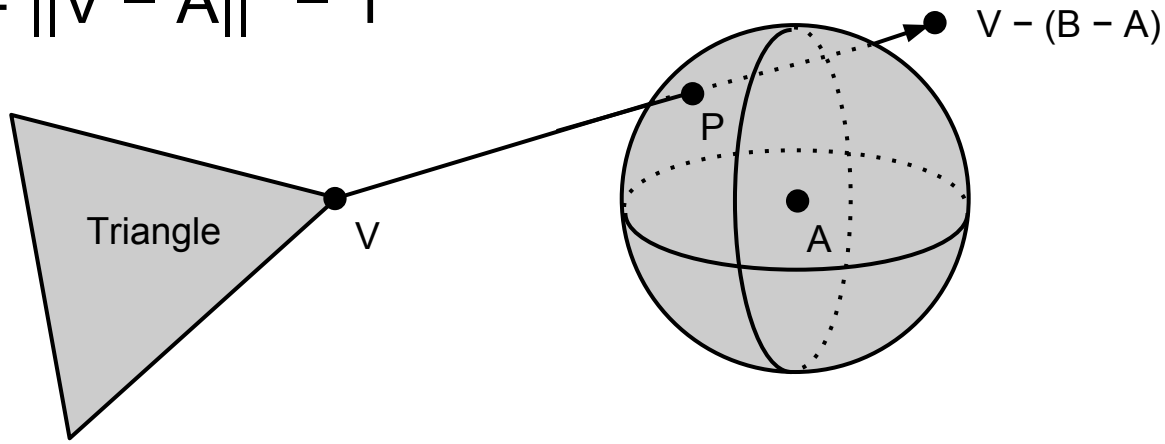
Analytic Sphere-Vertex Collision

- Collision test against a triangle vertex V
- How to collide moving sphere against point?
  - We know how to do a ray-sphere intersection test
  - Moving sphere vs point equivalent to sphere vs moving point (in the opposite direction)
  - Really just two points that "collide" at a certain distance apart



Analytic Sphere-Vertex Collision

- Point P on sphere if $||P - A||^2 = 1$
- Set $P = V - (B - A)t$
- Solve $||V - (B - A)t - A||^2 = 1$ for t
- Looks like $at^2 + bt + c = 0$ where
  - $a = ||B - A||^2$
  - $b = -2(B - A) \bullet (V - A)$
  - $c = ||V - A||^2 - 1$



Hybrid Approaches

- Ellipsoid not best for every situation
  - Possibly undesirable behavior of sliding off ledges
  - Tough to handle climbing up ledges
  - May want to special-case certain movement scenarios
- Possible solution: multiple collision representations
  - Collision model on ledge might be different than on ramp
  - Sometimes use multiple tests at once
    - e.g. climbing up a ledge

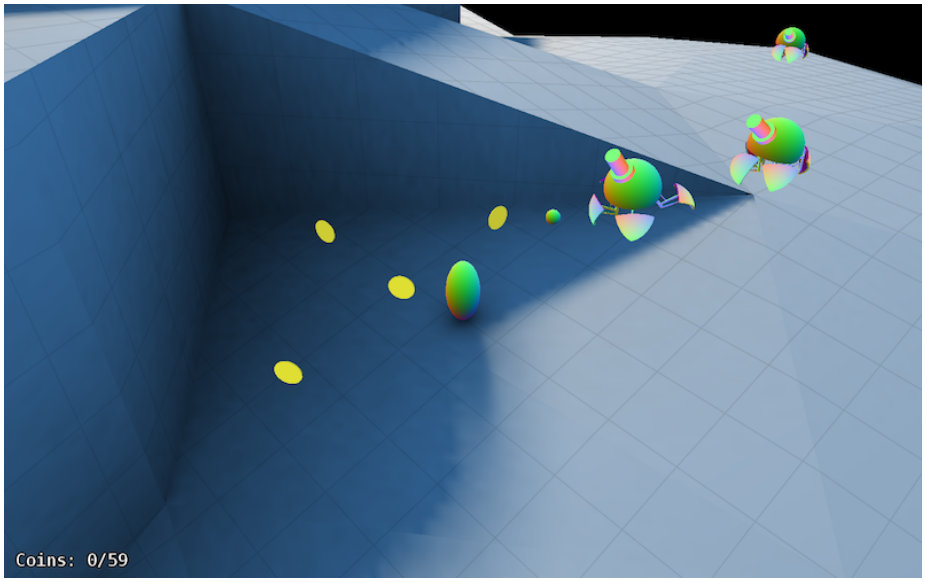
Case Study: *Overgrowth*

- How to handle ledge climbing?
  - Sphere test against wall
  - Cylinder test above ledge

http://www.youtube.com/watch?v=GFu44oeLYPI

Assignment 2: Platformer

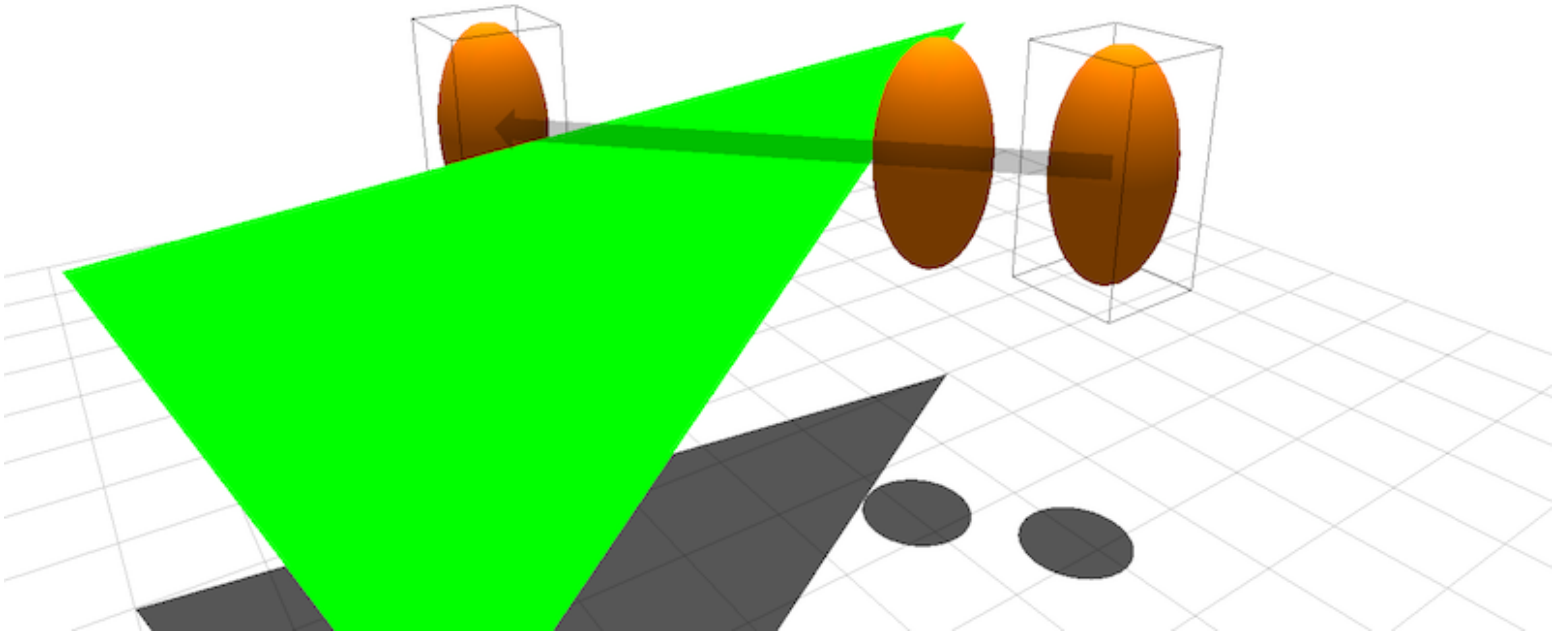
- Four week long assignment
  - Week 1: Collision detection
  - Week 2: Collision response
  - Week 3: Path finding
  - Week 4: Gameplay and in-game UI



Platformer: Week 1

- Collision detection
  - Analytic ellipsoid-triangle collisions
  - Develop collision code separate from game engine
- Use provided debugger project
  - Camera controls and ellipsoid manipulation built in
  - World consists of a single triangle

Collision Debugger Demo



C++ Tip of the Week

- Placement new
  - C++ constructors confuse two concepts: allocating and initializing memory
  - These can be separated with placement new:

```
// Usual way mixes allocation and initialization
Foo *foo = new Foo("text", 2);
delete foo;

// Placement new just does initialization, must call destructor
Foo *foo = (Foo *)malloc(sizeof(Foo));
new (foo) Foo("text", 2);
foo->~Foo();
free(foo);
```

C++ Tip of the Week

- Slab allocators
  - Pack many objects of the same type tightly together
  - Allocate a single slab of memory and call placement new for each element
- Fast file loading
  - Memory map the file and call placement new directly on the file buffer, no need for parsing or extra allocations
  - File buffer needs empty space of correct size for vtable (so it's a platform-specific hack)
  - Especially useful on mobile devices

Playtesting!

(in the Sunlab)