G54GAM Games

Building Games

Physics (2)

Newtonian Linear (Angular) Dynamics

- Newton's laws of physics
  - A body will remain at rest or continue to move in a straight line at a constant speed unless acted upon by a force.
  - The acceleration of a body is proportional to the resultant force acting on the body, and is in the same direction as the resultant force
  - For every action there is an equal and opposite reaction
- Forces affect movement
  - Based on the mass m of an object
    - F=ma
    - Gravity, impulses, repulsion, inertia
  - *Constrained* by springs, joints, connections, other objects
  - Calculate changing velocity and acceleration from the forces applied over the entire frame
- Need a general solution
  - Generally cannot find closed-form solutions for movement under force for all values of time t
  - Force, acceleration are rarely constant
    - Function of position, velocity

Numerical Integration

- Given position, velocity and force

  $s(t_1)$, $v(t_1)$, $F(t, p, v)$

- Find $s(t_2)$, $v(t_2)$

  $v(t) = s'(t)$

Approximate by

  $s(t_2) = s(t_1) + v(t_1)\, \Delta t$ ←first order differential

  $a(t) = F_{net}(t)/m = v'(t)$ ← what's the new velocity?

Approximate by

  $v(t_2) = v(t_1) + F_{net}(t_1)/m\, \Delta t$

Euler Integration*

- Starting with a stationary object at the origin weighing one kilogram, we apply a constant force of 10 Newtons and step forward with time steps of one second

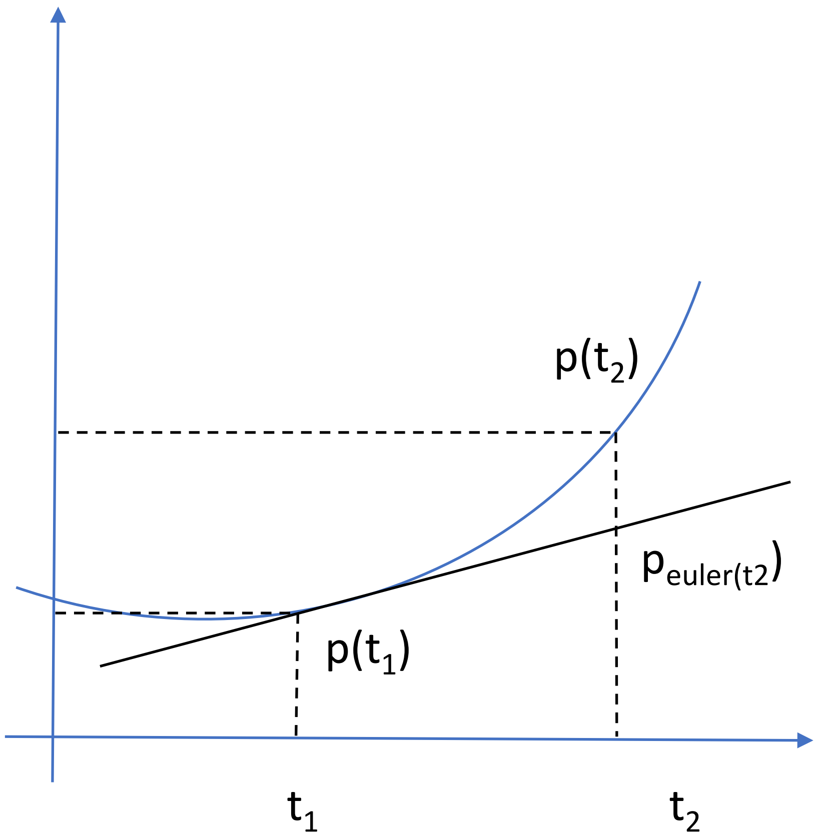
```
float t = 0;
float dt = 1;
float velocity = 0;
float position = 0;
float force = 10;
float mass = 1;

while ( t <= 10 )
{
   position = position + velocity * dt;
   velocity = velocity + ( force / mass ) * dt;
   t = t + dt;
}
```

time	position	velocity
0	0	0
1	0	10
2	10	20
3	30	30
4	60	40
5	100	50
6	150	60
7	210	70
8	280	80
9	360	90
10	**450**	100

Explicit Euler Integration*, Implicit Euler Integration, Semi-implicit Euler Integration, Verlet Integration, RK2, RK4

Euler Integration Errors



$s = ut + 0.5at^2$

$s = (0*10) + 0.5(10)(10)^2$

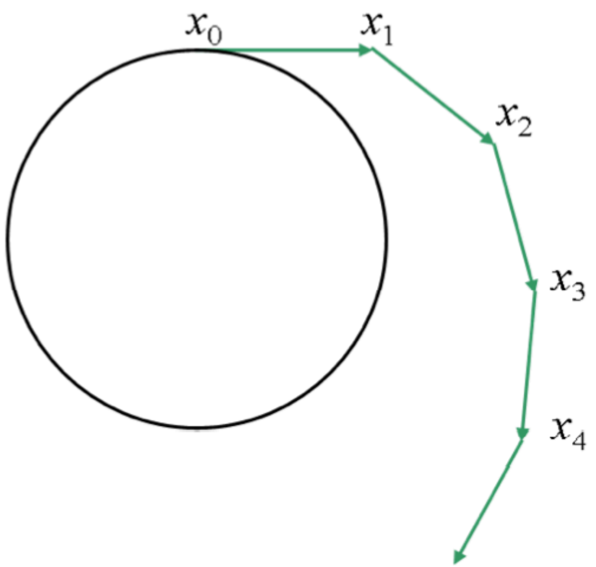
$s = 0.5(10)(100)$

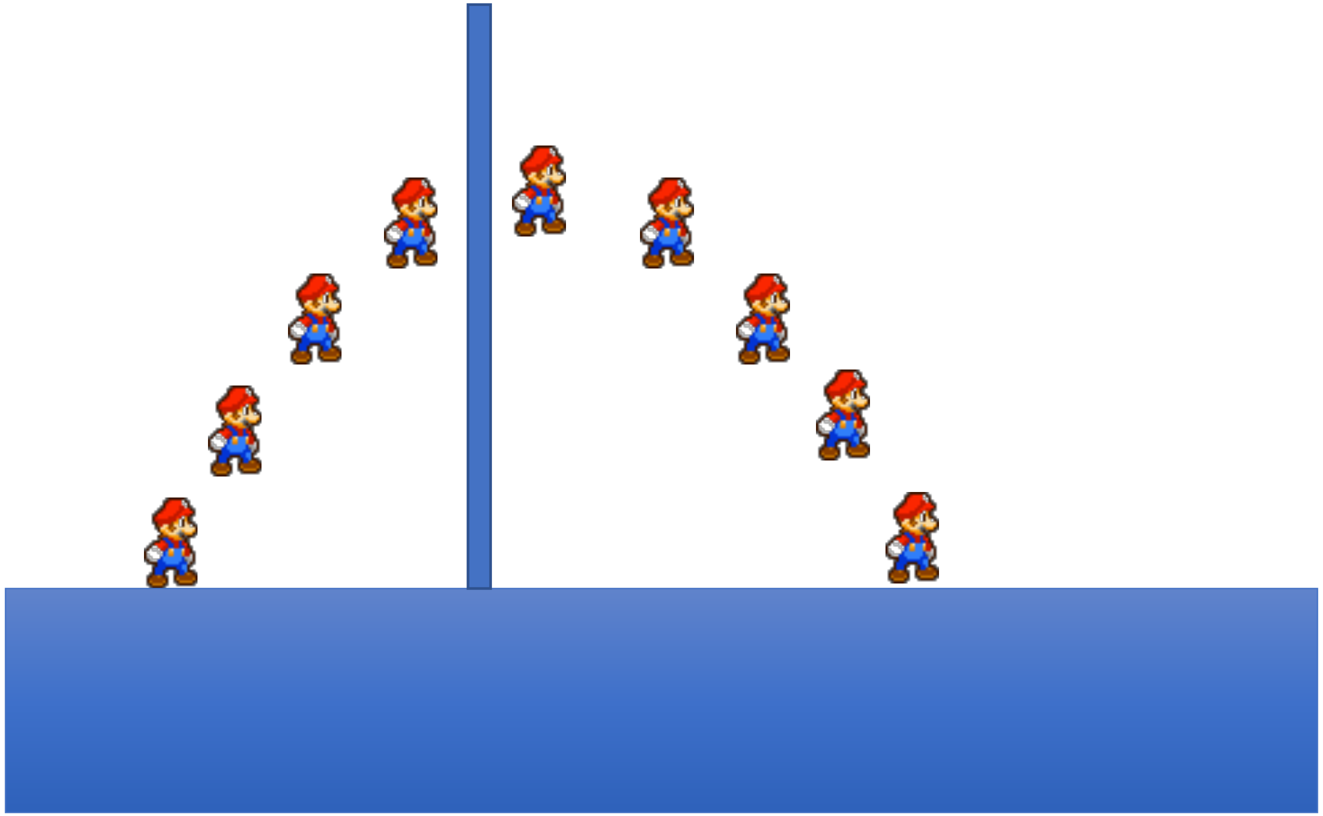
s = **500**

- Velocity is calculated per frame
  - But *changing* constantly
- As Δt approaches 0
  - Accuracy becomes perfect
- (Actual solution is an Infinite Taylor Series expansion
  - Euler only 1st order derivatives)

Issues with Physics

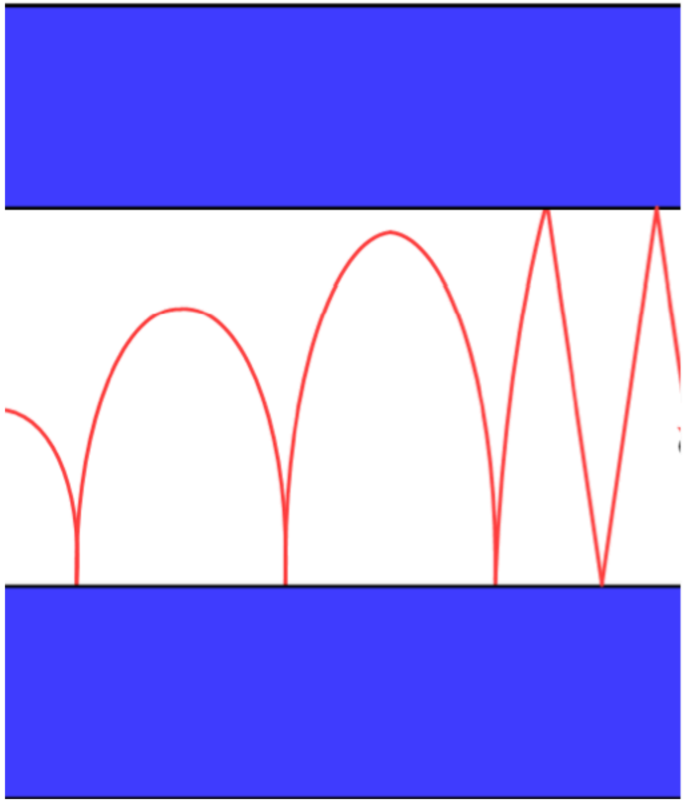
- "Flipbook" syndrome
- *Events* typically happen in-between snapshots
  - We never actually see the ball hitting the ground
- Curved trajectories are actually piecewise linear
- Terms assumed constant throughout the frame
- **Errors** accumulate





Properties of Numerical Integration

- Convergence
  - Does the solution approach the real solution as Δt approaches 0?
- Order
  - How bad is the error?
    - Cannot remove it
  - Error is proportional to some power of Δt
    - Square of the step size
- Stability
  - Does the solution add energy to the system?
    - Energy gain causes extreme glitches
    - Simulations explode
  - Requires ad-hoc solutions
    - Clamping to max-values
    - Manual damping
  - Remove energy from the system
    - Damping



Fixed Time Step

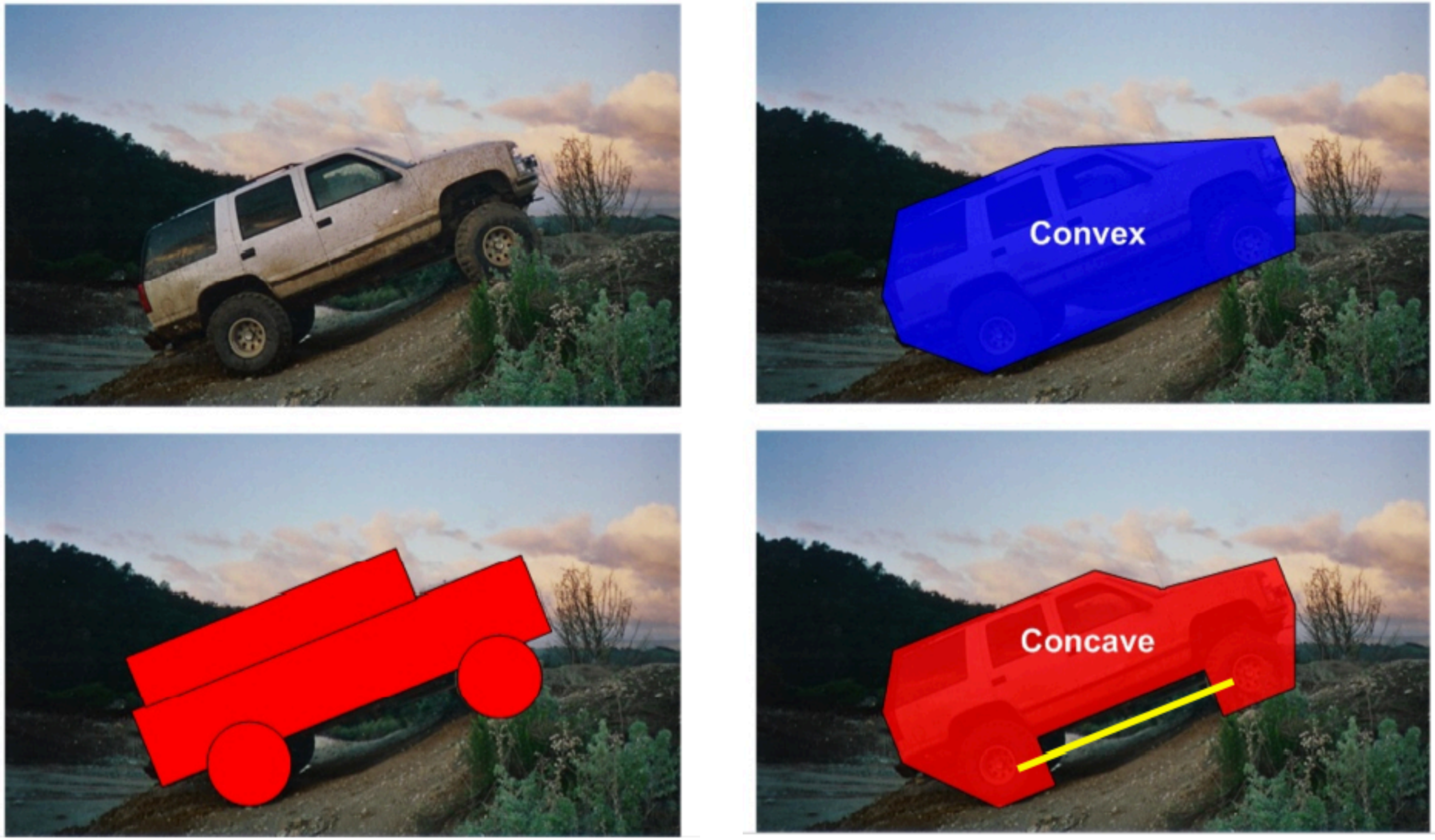
- Determinism
  - Consistent error regardless of CPU
    - Modifies how the simulation behaves
    - Informs damping parameters
- If Δt too large
  - Errors cause unexpected behaviours
- Typically a multiple of expected framerate
  - 30Hz, 60Hz, 120Hz
  - Can't simulate ½ Δt
    - Changes simulation behaviour
    - Simulate(½ Δt) Simulate(½ Δt) != Simulate(Δt)
  - "The renderer produces time and the simulation consumes it in discrete Δt sized chunks"
    - Renderer running at variable rate
- "It doesn't matter"
  - As long as the system is stable, it's doing its job
  - If the physics system performs 120 updates per second, and the level takes 20 minutes to complete, the physics system only needs to be stable for 144,000 updates

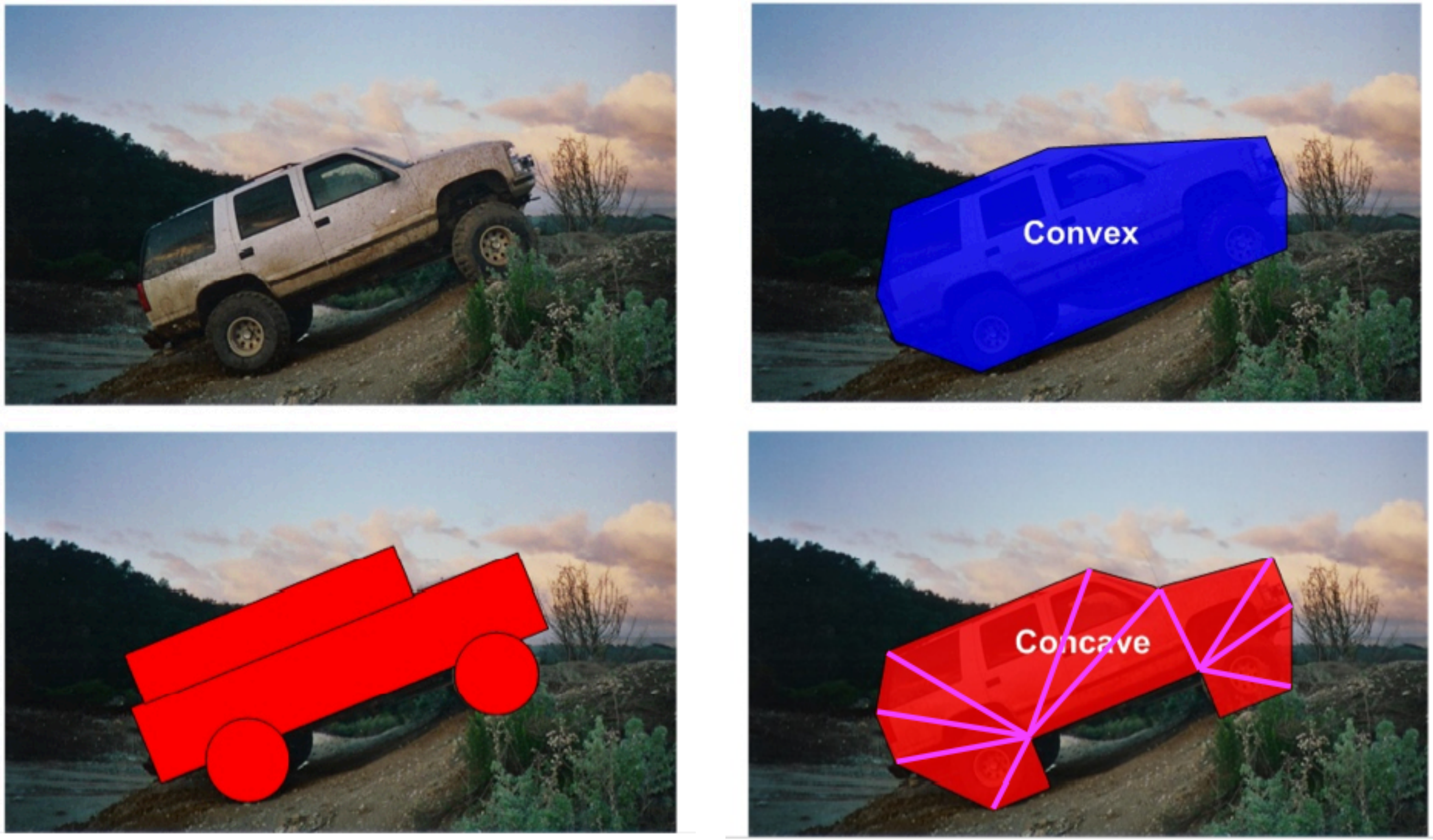
Physics in Games

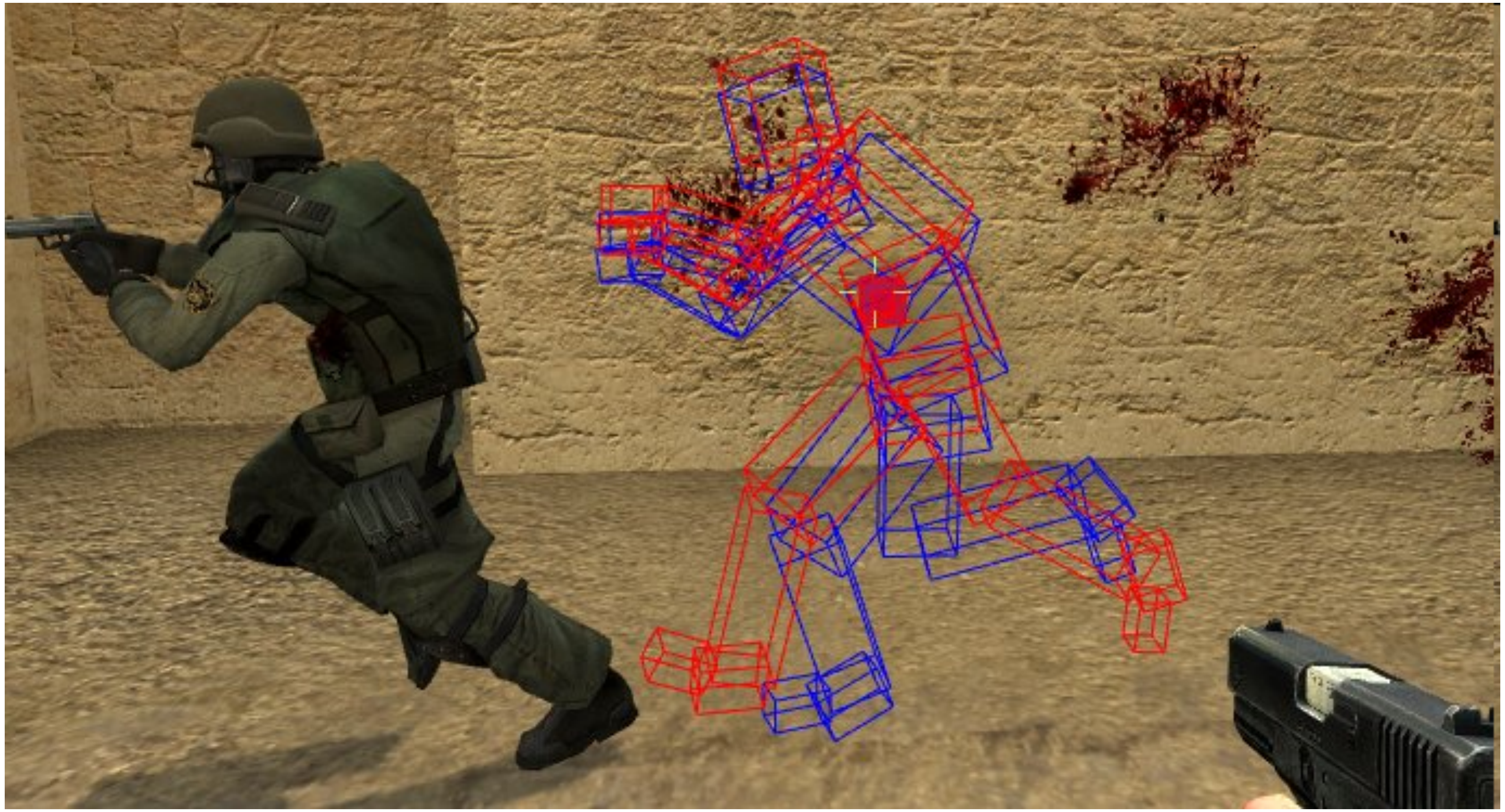
- "Simulate the game world"
  - Iterate the fundamental model
  - Moving objects based on **elapsed time**
    - Kinematics
      - Motion ignoring external forces
      - Only considering position, velocity
    - Dynamics
      - The effect of forces on the objects
- "Resolve Collisions"
  - Collisions between moving objects
    - Collision detection
      - Did a collision occur?
    - Collision resolution
      - Are we now in a restricted state due to constraints?
      - How do we fix it?

Collisions and Geometry

- Collisions require geometry
  - Points are no longer enough to model the object
  - To know *if*, *when* and *how* objects meet when they collide
    - Need to reason about interactions between shapes
- Sprites, 3d
  - Shape defined by pixels, mesh
- Separate collision mesh from renderable mesh
  - Simple convex mesh
    - Easiest shapes to compute collisions with
    - Break complex concave shapes into multiple convex components
    - Triangles are always convex
  - Collection of primitive objects


Convex
Concave


Convex
Concave



Scalability

- Key constraint
  - 16ms per clock tick (60Hz physics update)
    - Limited time to perform collision detection
- Naïve approach
  - Check two objects pixel-by-pixel for collisions
    - Shapes can be complicated
  - Check every object pair for collisions
    - Many objects can potentially collide
  - $O(n^2)$ checks
    - 200 objects = 19900 checks per tick
- Scaling approaches
  - Reduce number of collision pairs to check
  - Reduce cost of each collision check

Broad vs Narrow Phase

- If there are many small objects in large level
  - Each object only has the potential to collide with nearby objects
- Broad Phase collision
  - Perform a quick, inaccurate pass over n objects to determine pairs with potential to intersect, p
- Narrow Phase collision
  - Perform p x p check of object pairs to determine actual collision
    - Triangle Intersection calculation
    - Gilbert-Johnson-Keerthi (GJK) algorithm

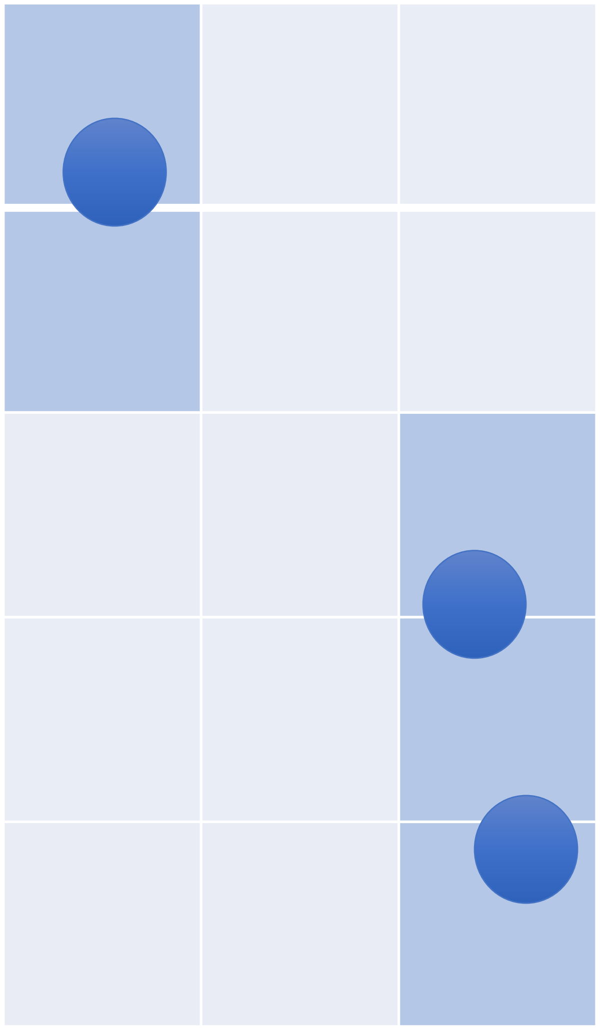
-> Dramatically reduce the number of checks

Collisions

- Collision Detection
  - **If** two objects intersect
  - Did the bullet hit the opponent
- Collision Resolution
  - Determining **when** two objects came into contact
    - At what point during their motion did they intersect
      - With what velocity
  - Determining **where** two objects intersect
    - Which points on the objects touch during a collision
      - Determine how to act next
- Complexity increases
  - If < when < where

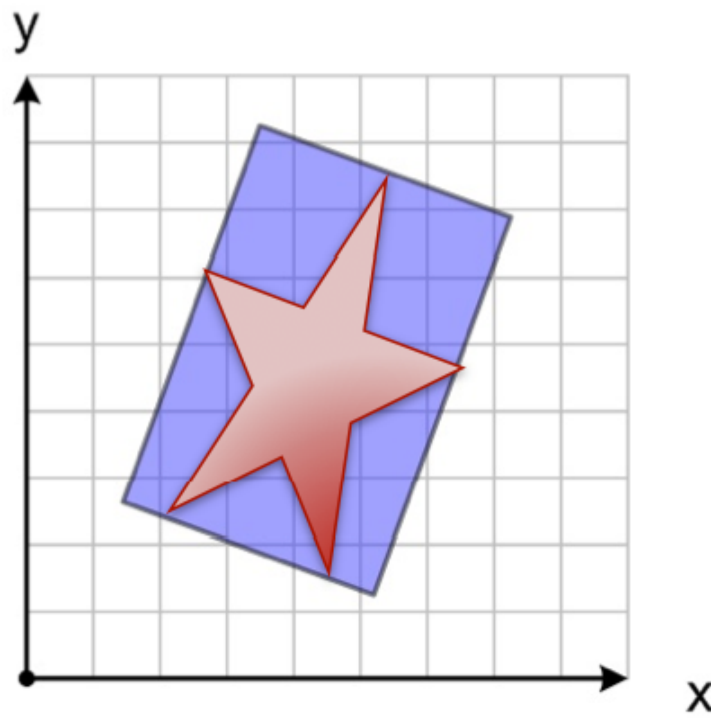
Broad Phase Optimisation Approaches

- Grid
  - Divide level into a grid
  - Place each object in a square
  - Only check the contents of the square against neighbouring squares
- Application-specific
  - Identify classes of object that should collide
    - E.g. Aliens need not collide with one another



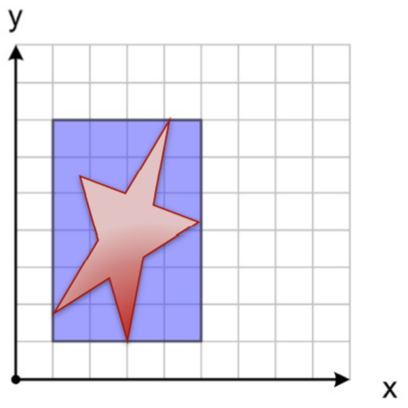
Broad Phase Optimisation Approaches

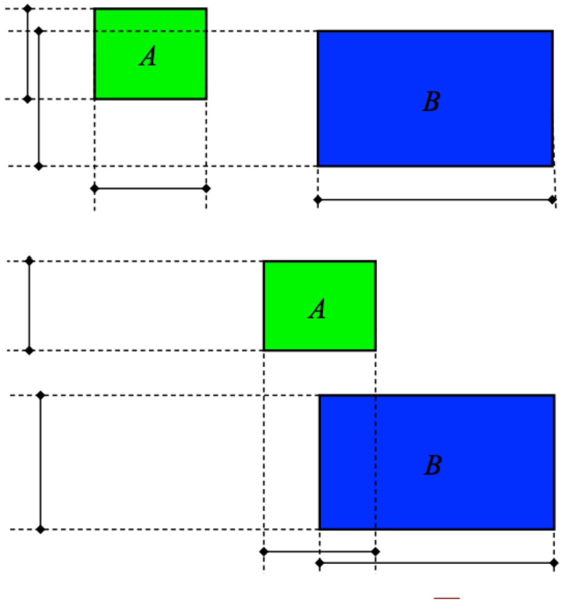
- Oriented Bounding Box
  - OOB
- Rectangular bounds
  - Minimal rectangle fitting
  - Angled to best fit
- Often less tight a fit
  - Creates false positives
- Just as slow as triangles
  - Boxes may have many different orientations
  - Corner cases



Broad Phase Optimisation Approaches

- Axis Aligned Bounding Box
  - AABB
- Similar to OBB
  - Rectangular fit
  - Align with x-y axes
- Often a very poor fit
  - False positives likely
- Checking is very cheap
  - Project box onto axes
  - Check whether intervals overlap





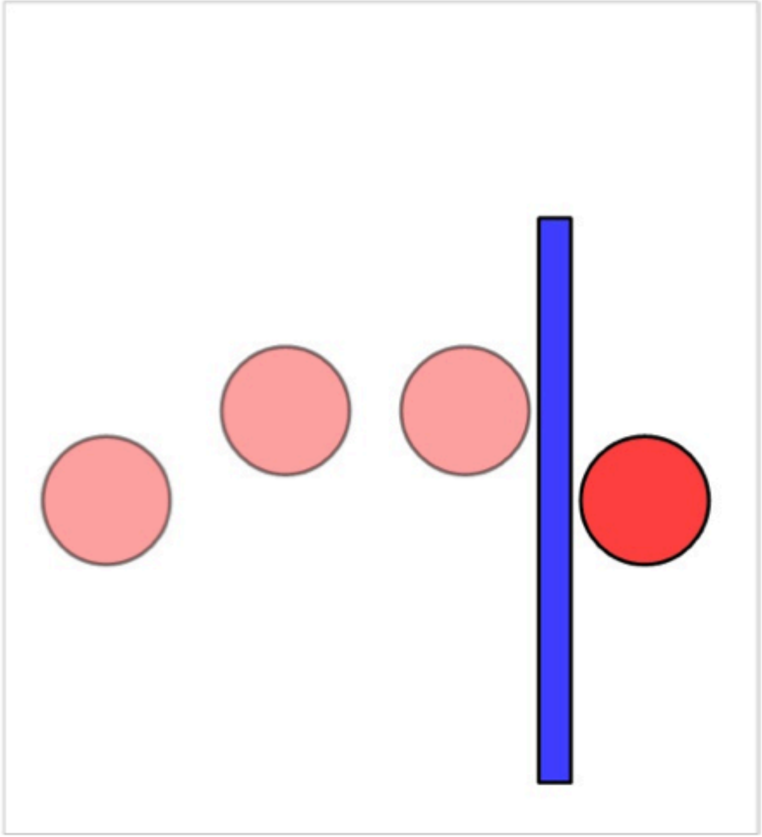
Physics in Games

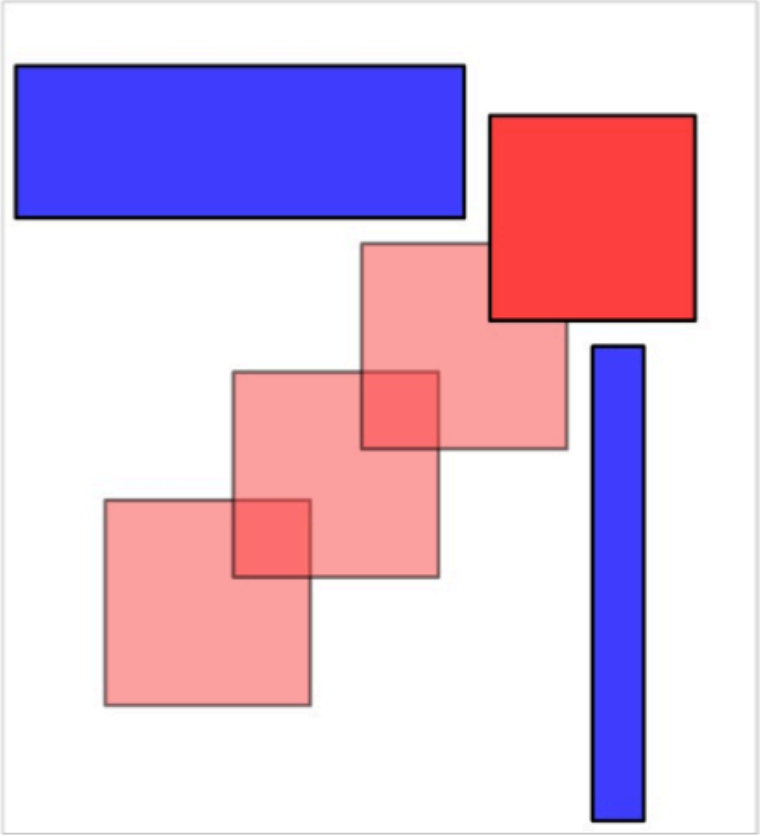
- “Simulate the game world”
  - Iterate the fundamental *model*
- Moving objects based on elapsed time
  - Kinematics
    - Motion ignoring external forces
    - Only consider position, velocity
  - Dynamics
    - The effect of forces on the objects
- “Resolve Collisions”
- Collisions between moving objects
  - Collision detection
    - Did a collision occur?
  - Collision resolution
    - What is the result of a collision?
    - How are objects allowed to move wrt one another?

Collision Detection - Tunneling

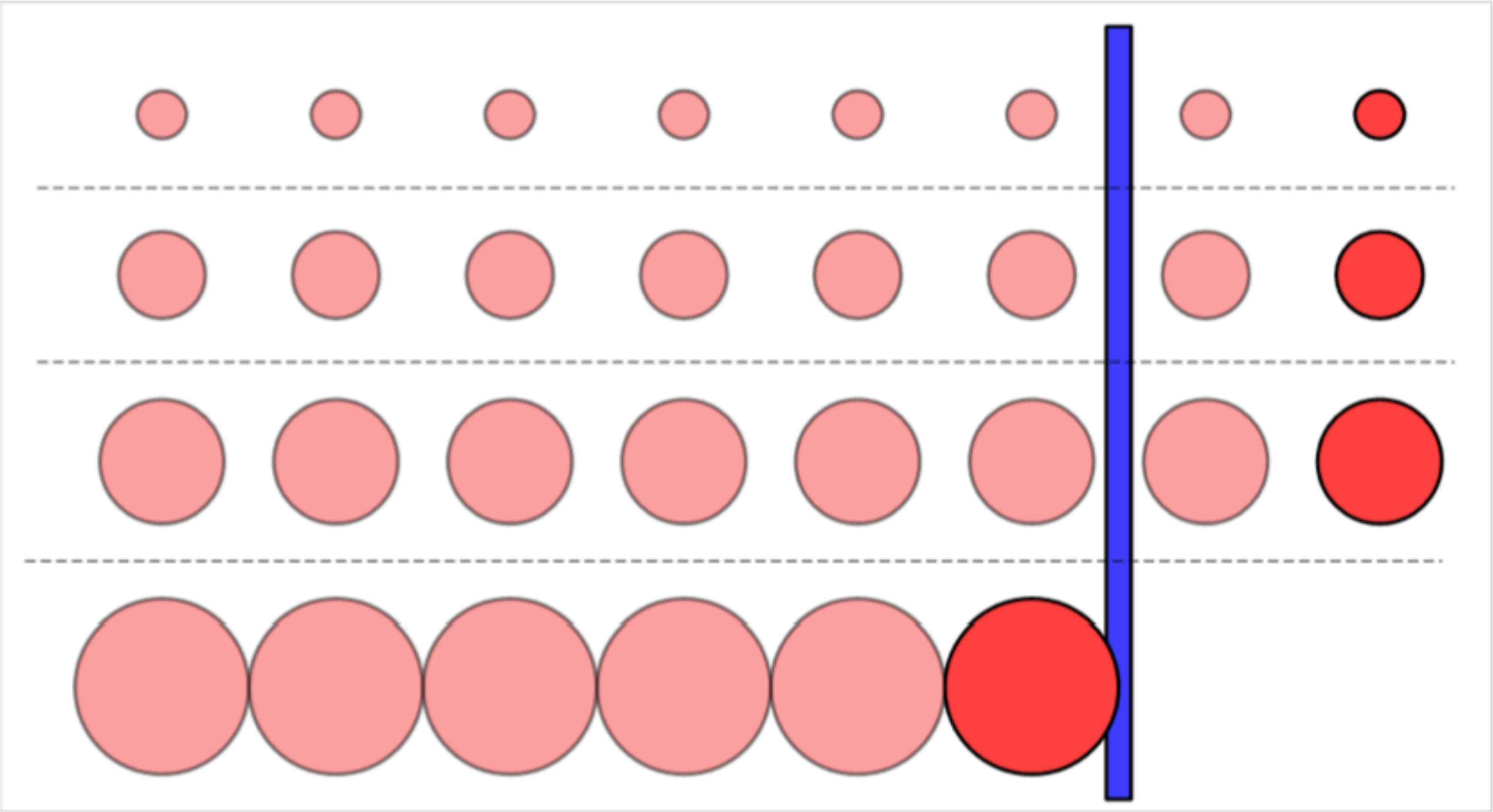
- Collisions in mid-step can lead to tunneling
  - Objects that pass through one another
    - Not colliding at start or end of the simulation
    - They collided somewhere in between
    - A false negative
  - *When* did the collision occur
- A serious issue for gameplay
  - Players getting into places that they shouldn’t
  - Players missing an event trigger boundary

Tunneling

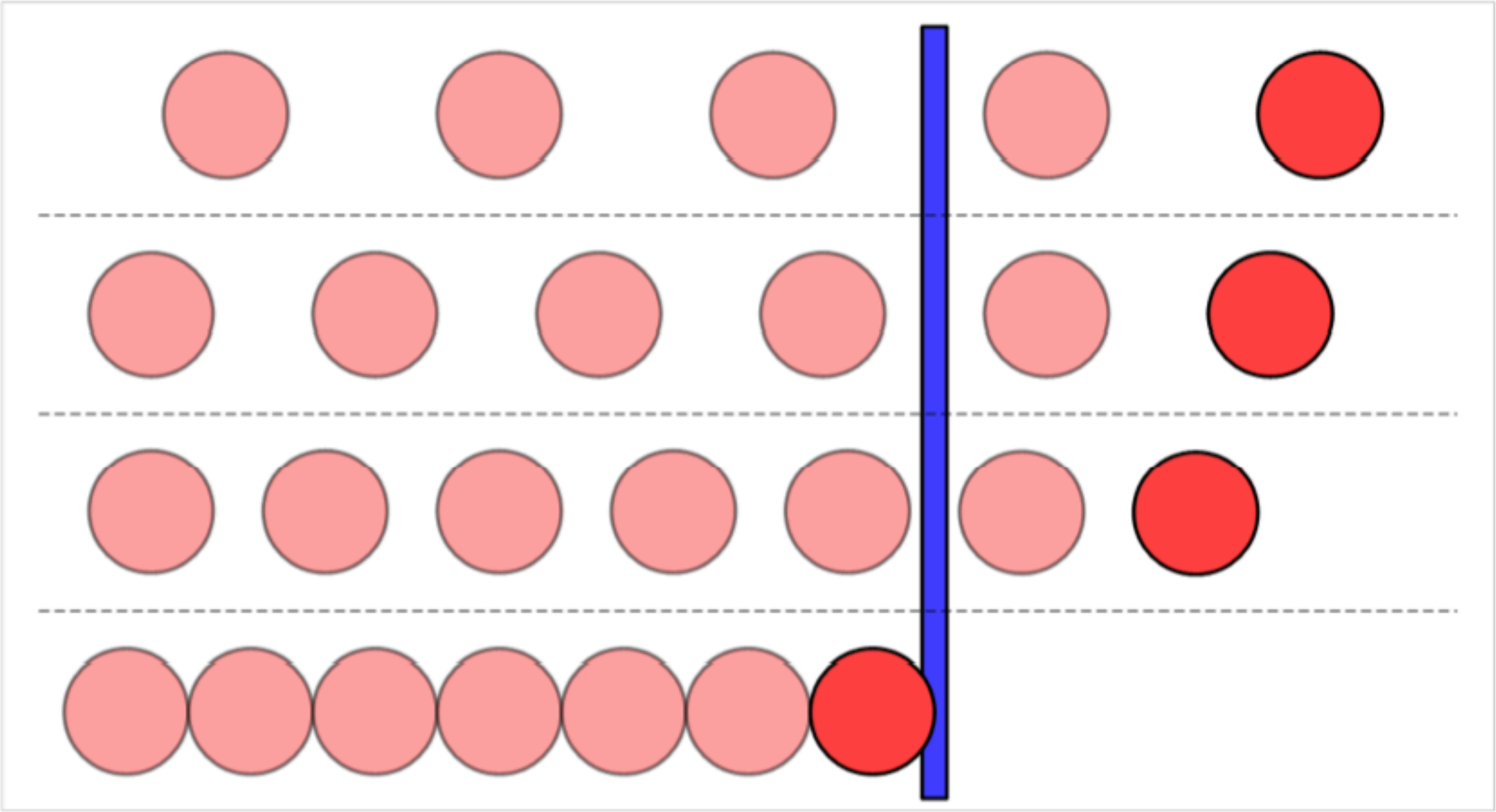




Tunneling – Small Objects



Tunneling – Fast Objects

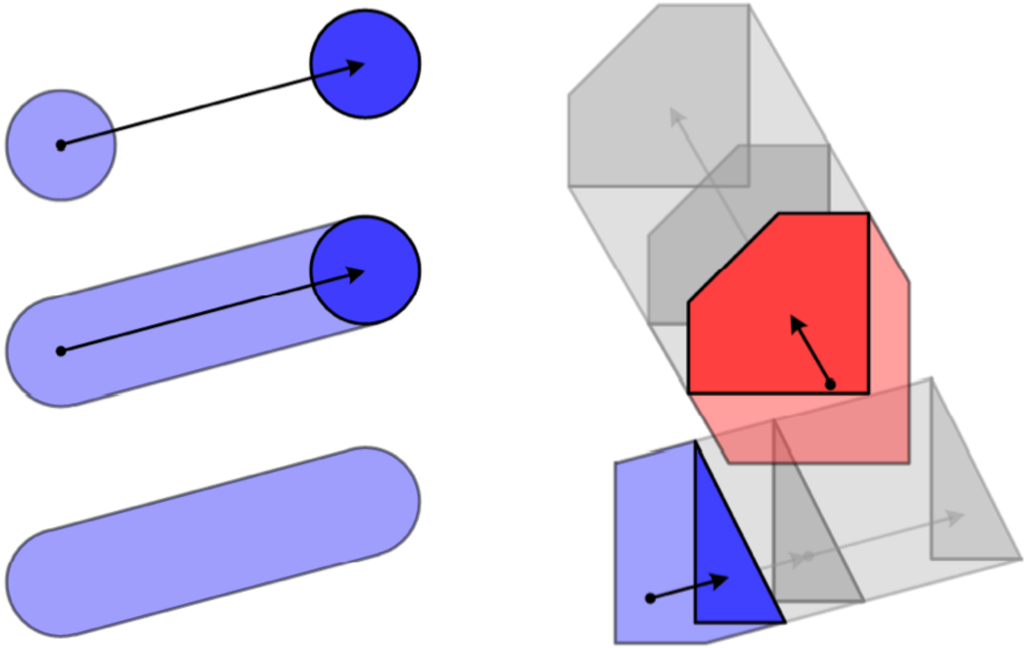


Solutions to Tunneling

- Minimum size requirement
    - Fast objects still tunnel
- Maximum speed limit
    - Speed limit is a function of object size
    - Small and fast objects not allowed
- Smaller time step
    - Essentially the same as a speed limit
- Solutions generally inadequate
    - However physics engine should be expected to operate best within a certain *range*

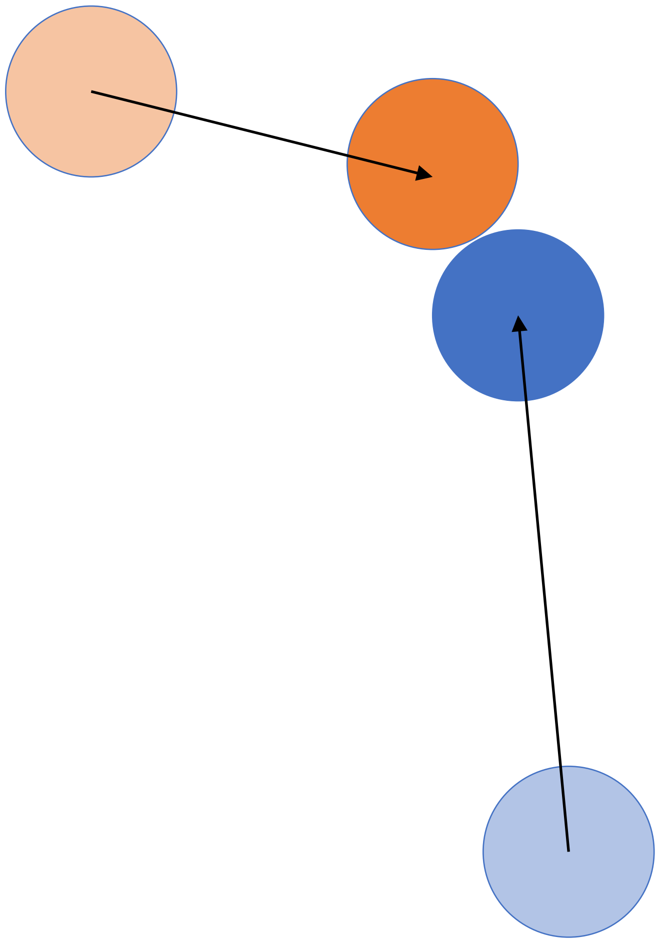
Swept Collision Shapes

- Movement volume defined by start and end position of the object
  - Swept disk = capsule
  - Swept triangle convex poly
  - Swept convex = convex poly
  - Swept AABB = convex poly
- Still has false positives
  - Removes many problems



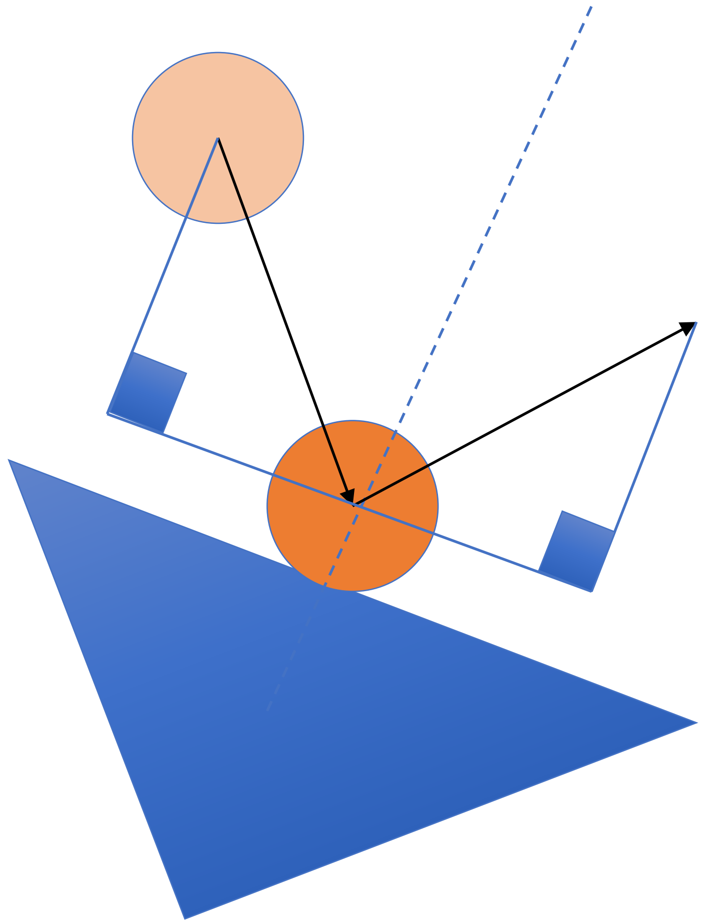
Collision Types

- Inelastic collisions
  - No energy preserved
  - Objects stop in place (v=0)
  - Easy to implement
- Elastic collisions
  - 100% of energy preserved
  - Snooker balls
- Partially elastic
  - x% of energy preserved
  - What is the object made of?
    - Rubber, steel
    - Parameter for the engine



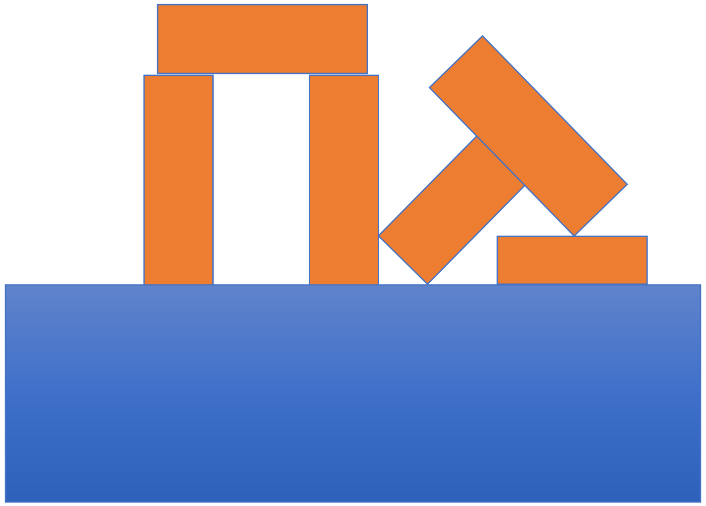
Simplest Case – Disks / Spheres

- Single point of contact on collision
  - Energy transferred at contact point
- Use relative coordinates
  - Point of contact is origin
  - Perpendicular component
    - Line through origin, center
  - Parallel component
    - Axis of collision surface
- Reverse object motion / exchange energy along the perpendicular component



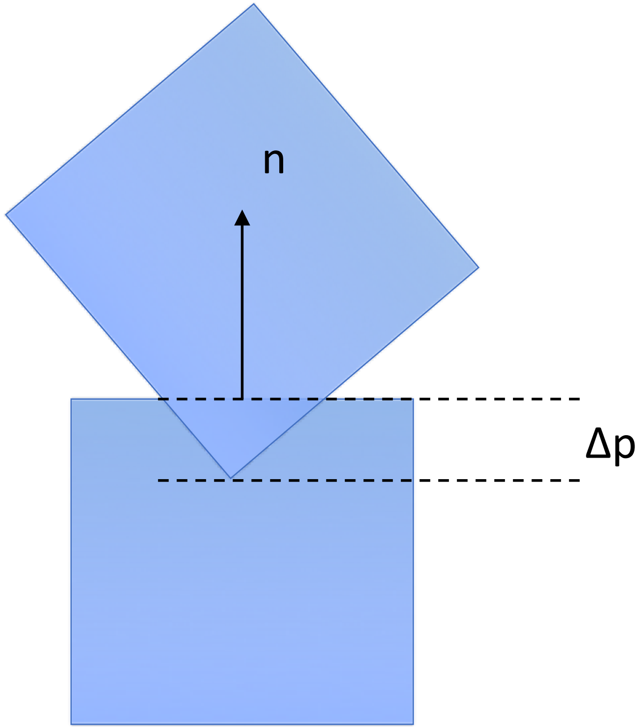
More Complex Shapes

- Point of contact more difficult to define
  - Could just be a *point*
  - Could be an *edge*
- Model with rigid bodies
  - Break object into points
  - Connect with *constraints*
  - Calculate forces at point of contact
- Requires a *constraint solver*
  - Solve a complex system of linear equations

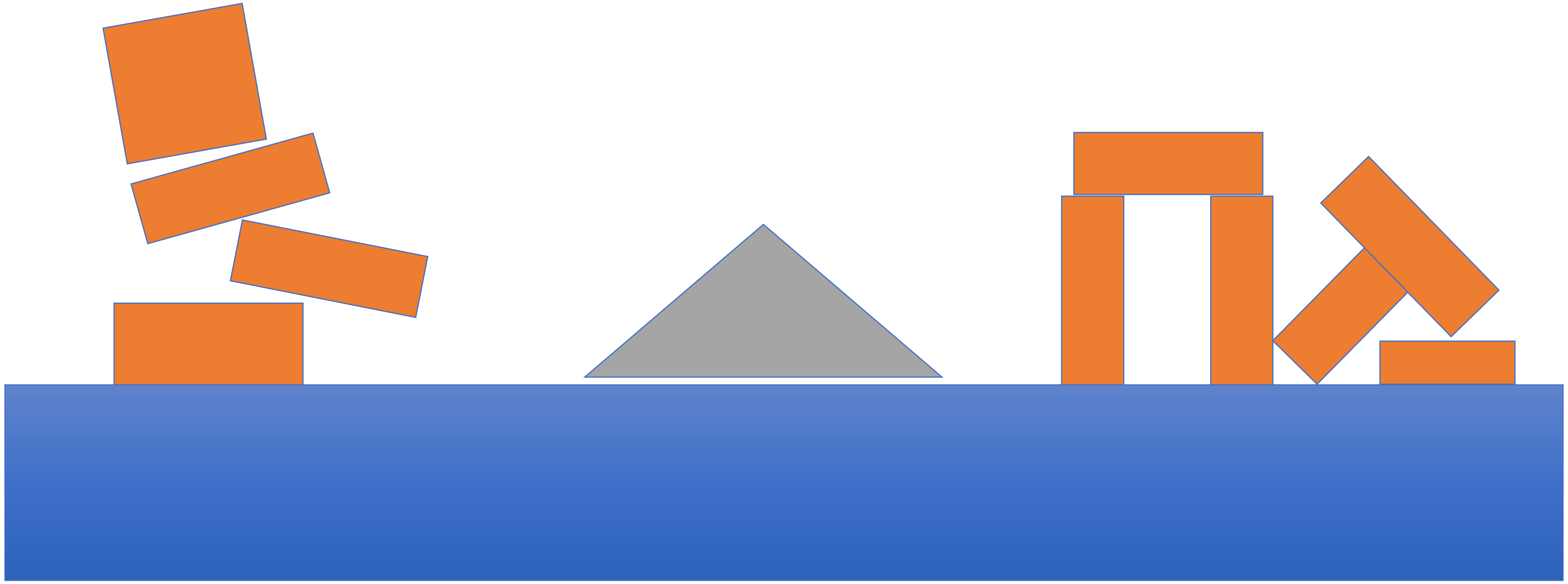


Non-Penetration Constraint

- What velocity is required to resolve the constraint?
- Baumgarte Stabilization
  - Apply impulse force proportional to Δp
  - Momentum
  - Spring / dampener
- Allow a bit of *slop*
  - Acceptable interpenetration
  - Reduces jitter

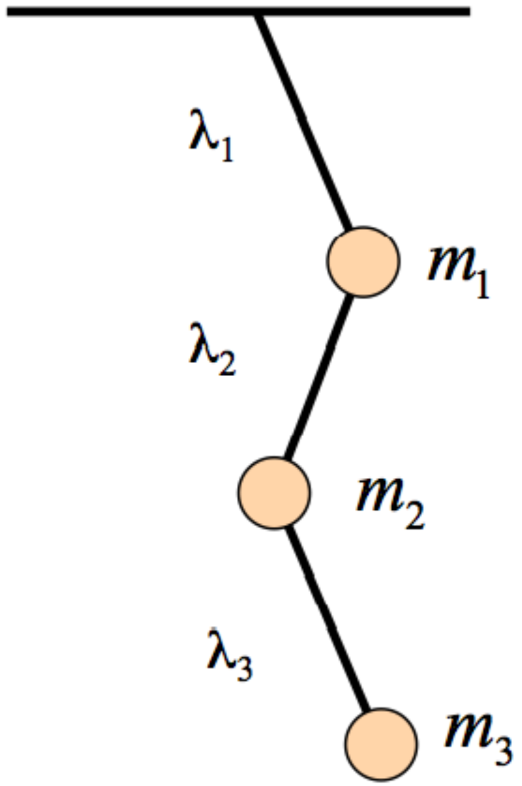


Sleeping



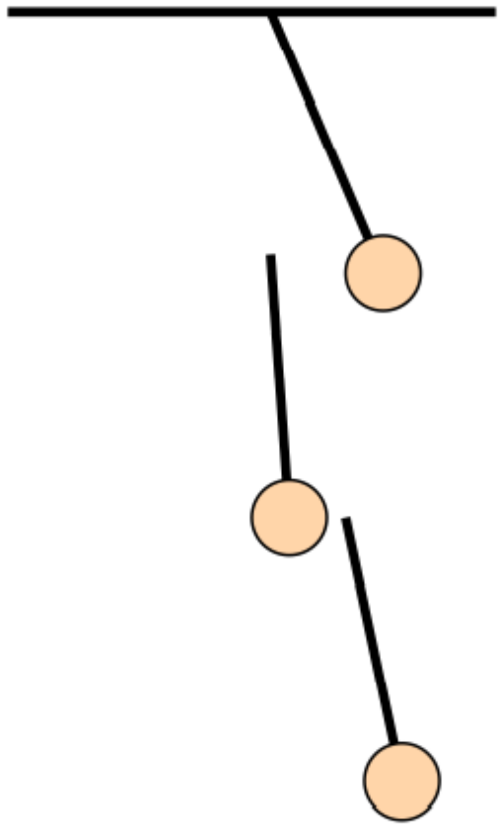
Constraint Solvers

- Solve a collection of constraints
  - Position, velocity
- Global constraint solver
  - Slow
  - Solve for λ1, λ2, λ3 simultaneously
- Iterative constraint solvers
  - Fast
  - Solve for λ1, λ2, λ3 in sequence
  - Apply impulse forces at each constraint
  - Converge to a global solution
- Applications
  - Ropes, chains, cloth
  - Box stacking



Constraint Solvers

- Difficult to implement
  - Errors
    - Causes joints to fall apart
  - Position drift
- Use the physics engine as a black box
  - Engine implements constraint solvers for various *kinds* of joint / constraint



Physics in Practice

- A collection of constraint models
  - Joints
    - Rotational, linear
  - Springs and masses
  - Cloth
    - Connected systems of rigid bodies
  - Ragdolls
    - Avatars constructed from jointed collections of capsules
  - Soft-body physics
- Reduce problem to a collection of simple rigid-body primitives
  - Capsules, cubes, spheres
  - Abstraction, limited range of reasonable values, shapes
- How to integrate with animation?
  - Kinematics meets linear dynamics
    - Inverse kinematics
  - Simple capsule constructs as player controllers
    - Change to ragdoll on “death”


fps: 66 ping: 56 ms 30/s
in : 61 1.53 k/s 17.3/s
out: 66 1.44 k/s 18.2/s
30/s
77
0
$ 0
30 | 53
5.56