Collision Response (2D, mostly)

What effects should a (game) object suffer during or after a collision?

- No effect, like a boundary wall of simulated world

- Bounce away, instantaneously, like billiard balls or ping-pong balls
- Bounce away, after deformation, like water-filled balloons
- Penetrate something, like arrow or bullet
- Be penetrated, like clay or flesh
- Be set spinning, like tripping up over an obstacle and falling

- Disappear, like raindrop or bomb that explodes
- Explode into fragments, Splash into droplets
- Have shape changed, like dented car or wounded character (hero, monster, NPC)
- Be fragmented, like glass in Angry Birds

With or without nearly-correct physics?

Effects on motion of objects may or may not be determined using physics simulation

- **Kinetics**, analytical dynamics
  - Object masses, forces on objects (as vectors), resultant acceleration
  - Conservation of momentum (and angular momentum, even in 2D)
  - Newton's Laws of Motion
  - It may be computational overkill, giving more realism but perhaps less fun.

- **Kinematics** (regarding the properties of motion without regard to its causes)
  - How acceleration affects velocity, how velocity affects position
  - Direct manipulation (in code) of object position and/or velocity

Non-physics, wholly artificial and unrealistic responses may be scripted

With physics

Both the kinetics and the kinematics approaches require determining

- The moment of impact of two objects
- The parts of two objects making contact
- Normal vector (in 2D, usually one vertex and one line collide, sometimes two lines)

The normal vector can be used

- In the kinematics approach, to calculate
  - Displacement direction (place bouncy objects just touching)
  - Changes to velocity owing to bounce (with coefficient of restitution), friction
- In the kinetics approach, to calculate
  - The forces causing acceleration

Determining the moment of impact

When two game objects collide, it usually happens *between* frames not *at* a frame

Collision detection using swept AABBs (see "Fast-moving Objects?" earlier)

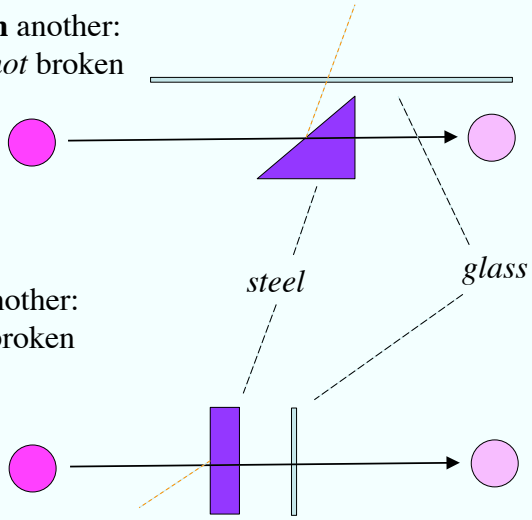
- Needs to decide not just *could objects collide* but also *do objects collide*
- Use repeated binary subdivision of time interval and swept areas
  - 5 iterations generally sufficient (just how fast can objects be?)
    - (gives 1/32nd of frame resolution in time and space, ≈ 1/2ms at 60fps)
  - Narrow-phase collision detector invoked up to 5 times
    - Do sprites change meanwhile?
    - Do polygons or assemblies rotate? (cf. wagon wheel effect)

      ***Cheat**: Will player of game actually notice if such details are wrong?*

- Issue: Can the first collision result in, or avert, other later-in-interval collisions?
  - If so, handle 1st collision first, repeat detection process for remaining interval

Sequencing of collision tests: may need to reconsider an outcome
Necessarily, pairs of objects need to be tested, in some order

One collision may **result in** another:
it will look odd if glass *is not* broken



One collision may **avert** another:
it will look odd if glass *is* broken

Where is the separating normal vector?

It is really unusual to have vertex-to-vertex contacts
- If it happens, probably a grid-based coordinate system is involved
  - ***If*** vertex-to-vertex contacts are required, take vector of relative velocity

For line-to-line contacts, the normal of both lines is appropriate

For vertex-to-line (or curve-to-line) the normal of the line should be used

For vertex-to-curve or curve-to-curve, the normal of the curve at point of contact

In 2.5D, sprites may intentionally resemble non-AA planes;
you may then want collisions to appear to result in bounces in the near/far axis.

In true 2D, use normals to lines

Motion Constraints

Objects should not intersect obstacles in the environment (never outer walls e.g.)

Objects should not usually intersect each other (or themselves)

- or if they do, like arrow in body, interesting things should happen
- so it is highly likely that graphical representations of the objects will change

Animations may involve coordinated and constrained movements of multiple objects:

Link Constraints (Pairwise, commonly in animations) should be observed

- e.g. wrist-to-elbow distance should be constant

Angle Constraints (Three-way) should be observed

- e.g. wrist-elbow-shoulder angle should be between 20° & 180°

Rag-Doll Physics

Make (dying) body object out of bones (links) and joints (as "particles")

- forces on one part of body are propagated to other objects through joints
- "penalty spring" forces can be brought to bear if constraints violated;
  - note however this can lead to numerical instability
- "zero tolerance" involves calculating the moment at which a constraint is about to be violated, processing it to prevent it happening – which gets expensive
- "relaxation" involves changing the state so violation is reduced or zeroed
  - a kinematics approach rather than kinetics

These approaches to resolving constraint violations apply beyond rag-doll realm

Multiple constraints

Dealing with one constraint violation may introduce another

You can deal with this iteratively, with a good chance of success but no guarantee

- collect set of constraint violations and the few objects they involve
- calculate a minimal compensating movement for each one in turn
- repeat until
  - all violations are fixed, or
  - iteration limit is reached, or
  - configuration is stable, the method has failed