Physics Tutorial 6: Collision Response

New Concepts

- Collision Response as a Concept
- Methods of Collision Response
  - Projection
  - Impulse
  - Penalty

- Impulse-Based Collision Response
- Penalty-Based Collision Response
- Soft Bodies

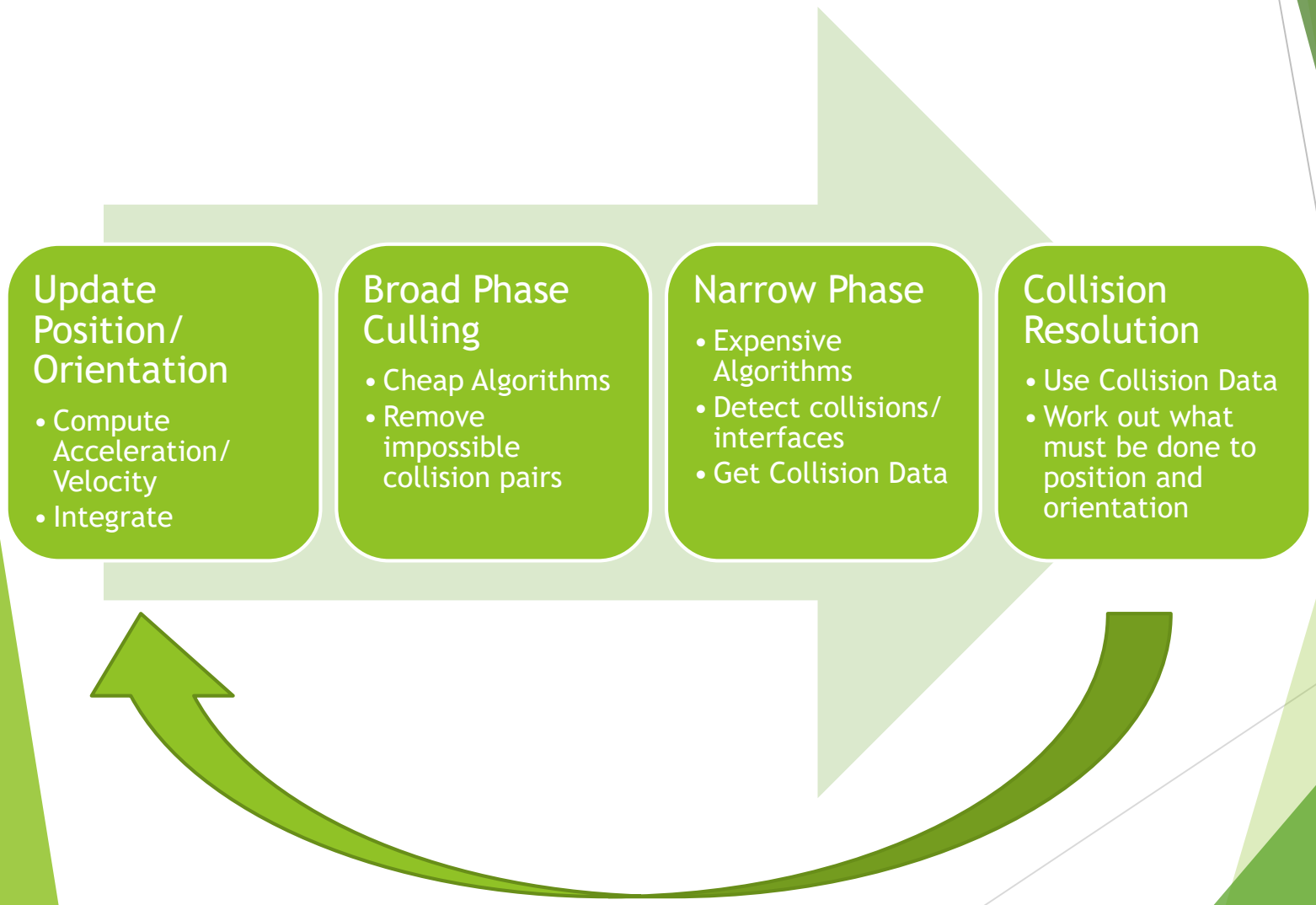
- Constraint-based Collision Response

Collision Response

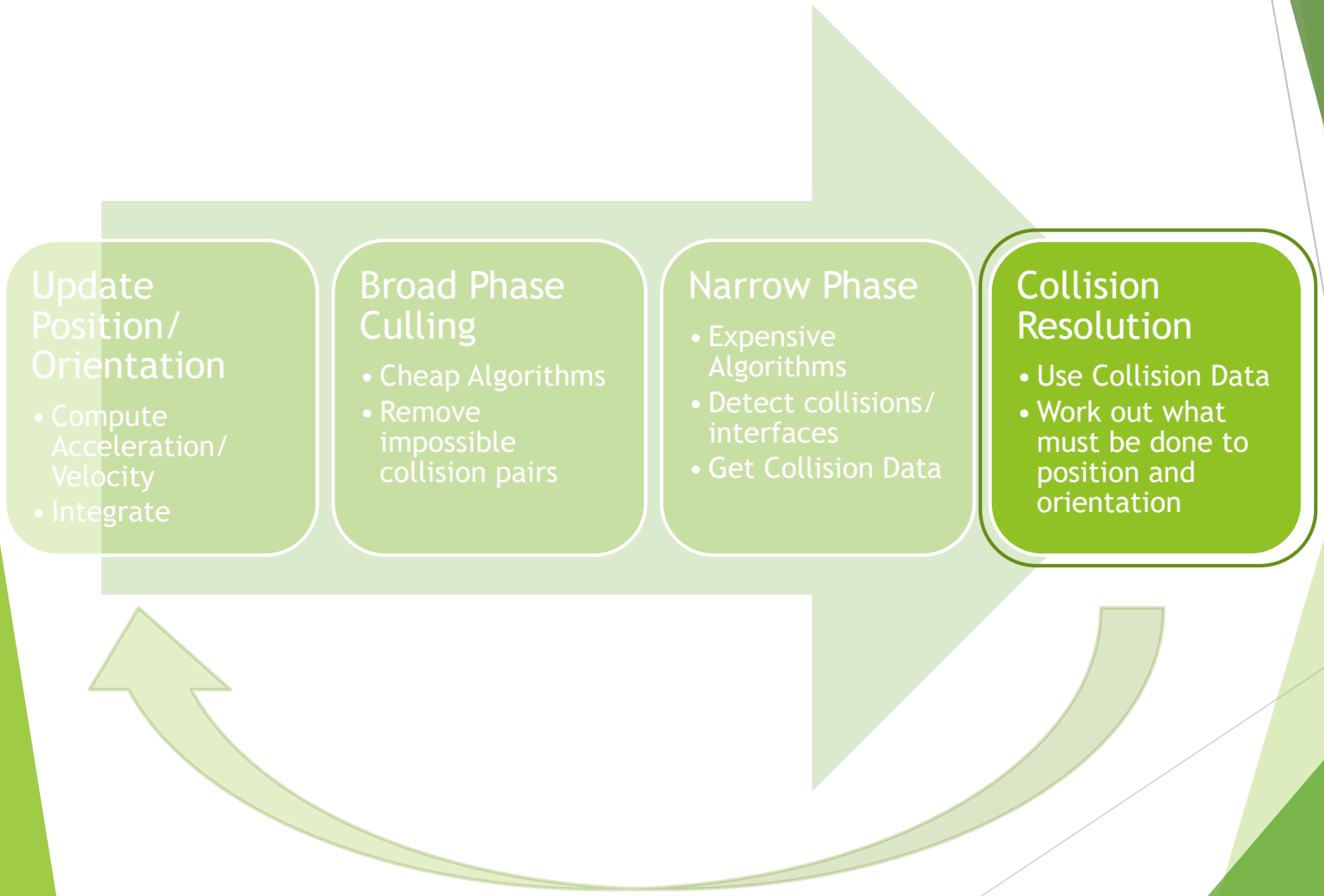
The Physics Engine

- We’re almost there
- We can move objects around our environment
- We can cull impossible collision pairs
- We can detect actual collisions and extract useful data from them

The Physics Engine



The Physics Engine



Collision Response

- Our response to a collision (interface) is generally to resolve it
- In this, there are two goals:
- First: we prevent an invalid configuration (we ensure that when the renderer updates, our objects are not overlapping)
  - Remember, we can ensure the renderer is making objects appear in the last position they occupied until we say otherwise

Collision Response

- Our response to a collision (interface) is generally to resolve it
- In this, there are two goals:
- Second: We make sure that the objects respond to their collision in a manner which is believable to the player
  - Has to be consistent with the scenario
  - In our case, we consider the purely physical accuracy as our metric, but there can be design considerations
  - After all, Mario jumps several times his height from a standing start, and lands without breaking knees

Collision Response

- The first problem is actually pretty easy to solve
- We could simply move both objects along the collision normal, separating them
- Move each of them a portion of the penetration depth and, assuming the objects are convex, there'll be no further collision
- If our concave objects are broken down into multiple convex hulls, the same applies
- So, why don't we just do this and go home?

Collision Response

- Because it's not believable.
- When snooker balls impact one another, there's an actual physical consequence to the collision
- If we simply move our objects such that they weren't intersecting, they'd either
  - Continue in the direction they were travelling, but bonded together
  - Stop moving, and remain bonded together
  - Which of these depends on our implementation, but normally it's the latter – because we're correcting the same collisions over and over again and resolving them the same way each time

Collision Response

- There has to be a better way…
- In fact, there are a few

- In this tutorial, we address the physics underpinning those ‘better ways’

- Next tutorial, we address the practical issues of how we do this computationally, reconnecting everything using the idea of global solvers

Collision Response Data

- We recall from that the collision response data required to resolve a collision is as follows:

- Contact Manifold – the point or points of the objects which were in contact at the instant of collision
- Contact Normal – the direction vector from the object's centre to the collision surface (alternatively, the normal of the surface which was penetrated)
- Penetration Depth – the depth to which the objects have interfaced

Methods of Collision Response

The Three Methods

- Projection Method – this acts on the position of our objects
- Impulse Method – this acts on the velocity of our objects
- Penalty Method – this acts on the acceleration of our objects
- Three attributes of motion, three methods for resolving collisions

Projection Method

- Basically, this is the simple approach we just outlined
- Only looks at the positions of the objects, and updates them to resolve interfaces
- Doesn’t consider what happens afterwards

Projection Method

- Generally not suitable for modern games
- But can be used to good effect, depending on how it's employed
- For example, you could probably implement old Mario using Projection Method and some cunning finite state machines. When Mario hits a brick wall, he stops and begins to fall, he doesn't bounce off it – motion after could be provided by gravity, at the integration step
- Don't use it for your coursework

Impulse Method

- This method acts on velocities of objects, and is based on conservation of momentum
- Commonly employed in video games
- The easiest ‘believable’ collision response system to integrate with your solver-based physics engine

Impulse Method

- When objects impact an immobile wall under this method, the resultant velocity is (almost – often you'll take out a little energy in damping) of the same magnitude as the incident velocity
- This represents rigid objects, such as snooker balls, very well.
- It's expected that you'll have implemented this in your coursework. For more marks, you're encouraged to investigate…

Penalty Method

- This acts directly on the acceleration of our objects, and involves a little understanding of the spring equation (covered in this tutorial)
- This permits deformable objects to act as colliders (soft bodies)
- This approach is also regularly employed in commercial engines
  - Remember our first lecture – a versatile physics solution supports as many options as it can, and permits the programmer to decide which is appropriate to a given scenario

Penalty Method

- Isn’t very complex to implement, if you consider the nature of a constraint-based approach to physical collisions
- Requires you to work out new constraints
- Then apply those between the elements of a soft body
- This is challenging, and is expected if you’re aiming for top marks in the coursework

Physics!

- For the rest of this tutorial, we'll be discussing the physics underpinning Impulse and Penalty methods

- We don't go into the physics underpinning the projection method, as it's both obvious (at the superficial level we need to consider – e.g., our objects can't share the same space in the same time step), and stupidly complex (at the actual-physics level) to derive from first principles

Impulse-Based Collision Response

What is Impulse?

- Impulse is the quantity defining force applied over a given time

- So, an object will gain impulse J if subject to force F for time interval Δt, or

$$J = F\Delta t$$

What is Impulse?

$$J = F\Delta t$$

- From Newton's second law, we can reformulate this

$$J = ma\Delta t$$

- And since $a = \frac{\Delta v}{\Delta t}$, we can simplify this down to

$$J = m\frac{\Delta v}{\Delta t}\Delta t = m\Delta v$$

What is Impulse?

$$J = m\frac{\Delta v}{\Delta t}\Delta t = m\Delta v$$

- Since we know from our first tutorial that *Momentum* is the product of mass and velocity, it stands to reason that *Impulse* is the rate of change of momentum (for an object of constant mass)

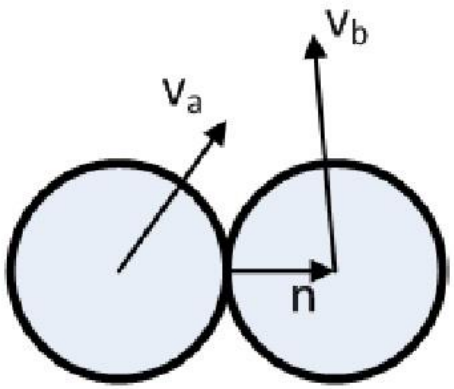
- The rationale, then, is to give colliding objects a nudge, by changing their velocity thus:

$$\Delta v = \frac{J}{m}$$

What is Impulse?

$$\Delta v = \frac{J}{m}$$

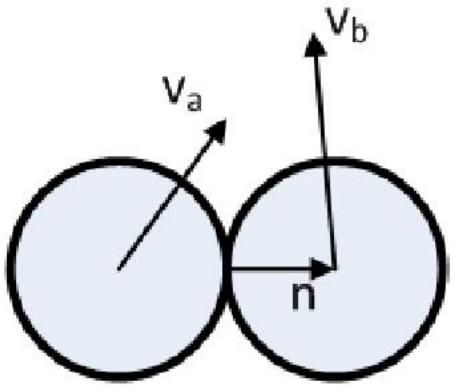
- The question then becomes how do we compute J
- To do this, let's consider the scenario below:



What is Impulse?

- Ignoring the masses of the objects for a moment, and focusing on the velocities involved, what we care about is their **relative** velocity
- The formula to determine this relative velocity, possibly counter-intuitively, is

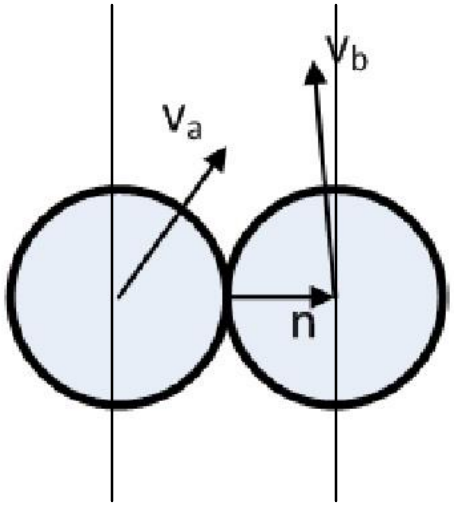
$$v_{ab} = v_a - v_b$$



What is Impulse?

$$v_{ab} = v_a - v_b$$

- Looking at the diagram, we can see how this is the case if we consider the Cartesian axes and how we represent vectors in them

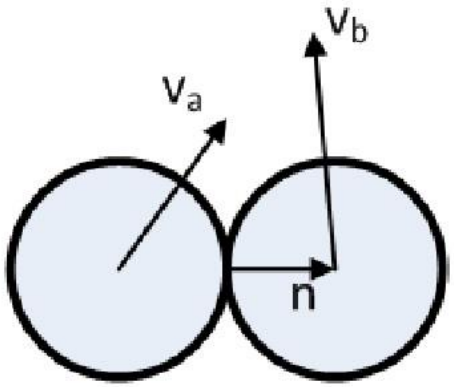


What is Impulse?

$$v_{ab} = v_a - v_b$$

- Regarding the relative velocity, what really matters is the projection of that relative velocity along the contact normal, or

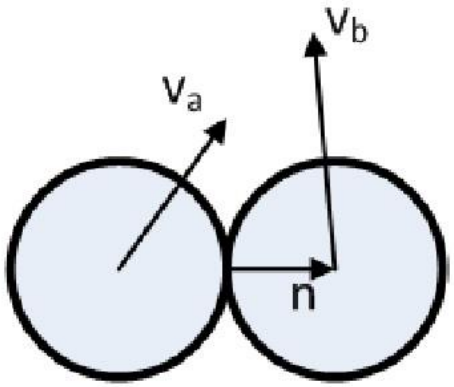
$$v_n = v_{ab} \cdot n$$



What is Impulse?

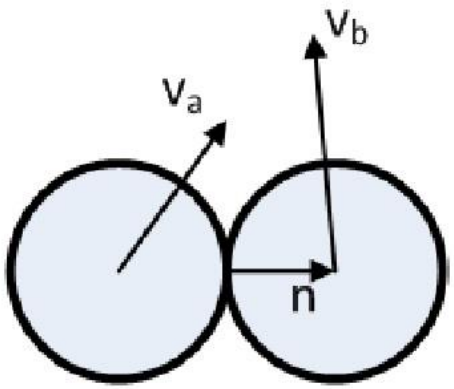
$$v_n = v_{ab} \cdot n$$

- We've talked a little about damping. One means of representing damping is through the *coefficient of elasticity* ε.
- If $\varepsilon = 1$, a collision is totally elastic (no damping)
- If $\varepsilon = 0$, a collision is totally inelastic (objects stick together)
- Most collisions, naturally, fall somewhere between these extremes



What is Impulse?

- Following on from that, we can assert that
$$v_n^+ = -\varepsilon v_n^-$$
- Which, subbing back in for v_n gives
$$(v_a^+ - v_b^+) \cdot n = -\varepsilon(v_a^- - v_b^-) \cdot n$$
- We've added the $^+$ and $^-$ notation to indicate 'after' and 'before' the collision, respectively

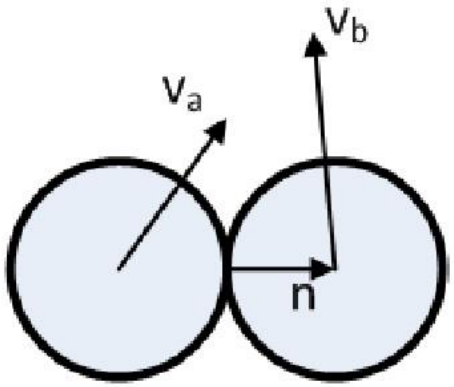


What is Impulse?

- Now we bring mass back into the equation – because we want to inject a little momentum into our system, along vector n, to resolve the collision
- Of course, we want our system to be stable, so we need to balance the injected momentum, leading to:

$$m_a v_a^+ = m_a v_a^- + Jn$$
$$m_b v_b^+ = m_b v_b^- + Jn$$

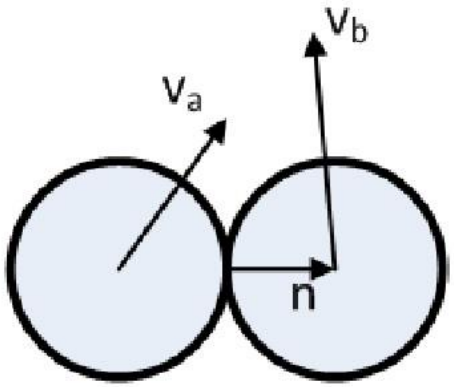
- Note that this notation assumes constant mass.



What is Impulse?

- Rearranging the last few equations allows us to determine the formula for J in terms of n, v_{ab}, and the masses involved

$$J = \frac{-(1+\varepsilon)v_{ab} \cdot n}{n \cdot n\left(\frac{1}{m_a} + \frac{1}{m_b}\right)}$$



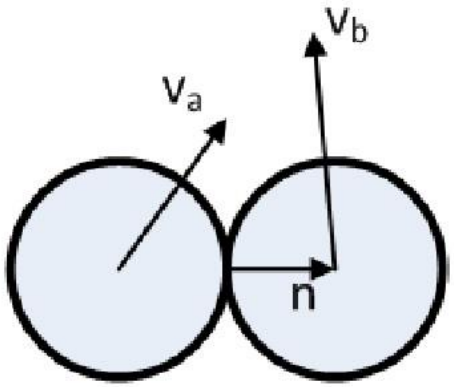
What is Impulse?

- In turn, this allows us to rearrange the linear velocity updates to

$$v_a^+ = v_a^- + \frac{J}{m_a} n$$

$$v_b^+ = v_b^- - \frac{J}{m_a} n$$

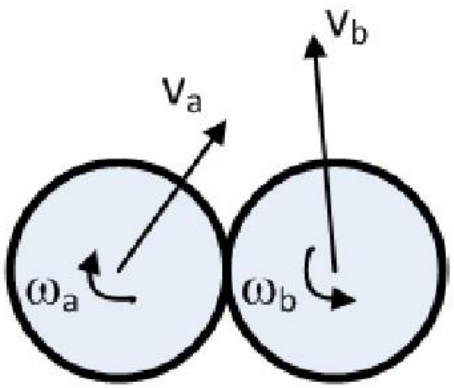
- Thinking back to how we define our constraint-based update, in the context of λ, you might be able to derive some commonality here…



What is Angular Impulse?

- That commonality is more obvious when we consider the angular scenario. Consider the diagram below:
- Obtaining an actual velocity from an angular velocity, assuming we've kept our system in radians, is a simple case of

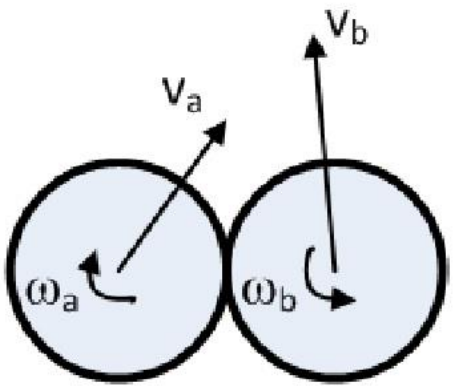
$$v_t = \omega r$$



What is Angular Impulse?

- Extending this to the contact manifolds of our spheres (remembering spheres will only ever have one contact point), leads to

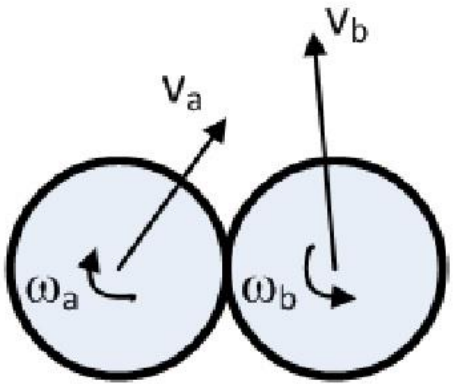
$$v_{C_a} = v_a + \omega_a r_a$$
$$v_{C_b} = v_b + \omega_b r_b$$



What is Angular Impulse?

- Remembering that we need to conserve angular momentum, we introduce the inertia matrix, leading to

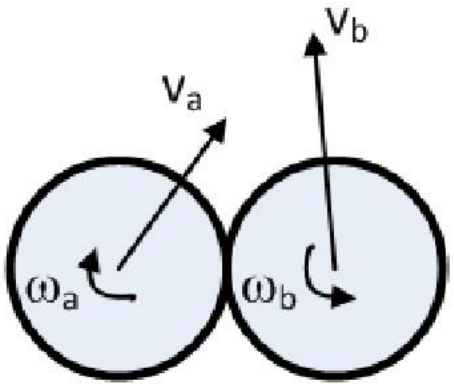
$$I_a \omega_a^+ = I_a \omega_a^- + r_a \times Jn$$
$$I_b \omega_b^+ = I_b \omega_b^- - r_b \times Jn$$



What is Angular Impulse?

- Rearranging again to determine J gives us the rather hideous

$$J = \frac{-(1+\varepsilon)v_{ab} \cdot n}{\left(\frac{1}{m_a} + \frac{1}{m_b}\right) + \left[\left(I_a^{-1}(r_a \times n)\right) \times r_a + \left(I_b^{-1}(r_b \times n)\right) \times r_b\right] \cdot n}$$

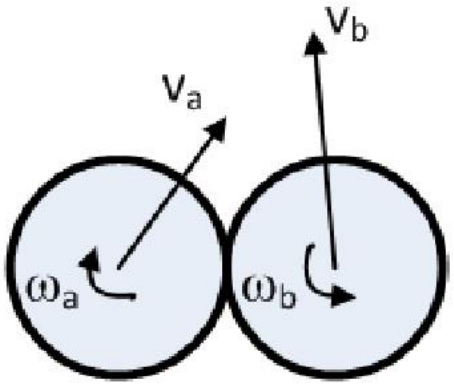


What is Angular Impulse?

$$J = \frac{-(1+\varepsilon)v_{ab} \cdot n}{\left(\frac{1}{m_a} + \frac{1}{m_b}\right) + \left[\left(I_a^{-1}(r_a \times n)\right) \times r_a + \left(I_b^{-1}(r_b \times n)\right) \times r_b\right] \cdot n}$$

- Which we can sub in to:

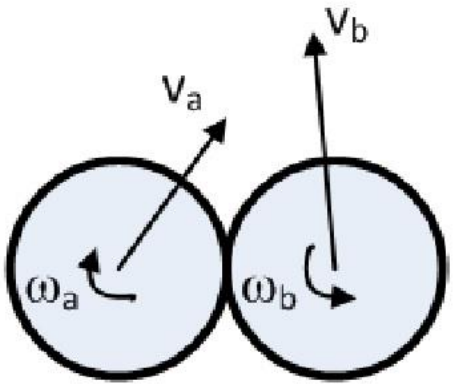
$$\omega_a^+ = \omega_a^- + \frac{r_a \times Jn}{I_a} \qquad \omega_b^+ = \omega_b^- + \frac{r_b \times Jn}{I_b}$$



What is Angular Impulse?

$$\omega_a^+ = \omega_a^- + \frac{r_a \times Jn}{I_a} \quad \omega_b^+ = \omega_b^- + \frac{r_b \times Jn}{I_b}$$

- Thinking back to the form of the elements of our Jacobian, we should definitely be seeing a clear similarity at this point



Penalty-Based Collision Response

Springs

- The physical properties of a spring, assuming it is not stretched beyond the point it will return to equilibrium, are
  - When a spring is compressed by two objects, it forces them apart
  - When a spring is elongated by two objects, it forces them together
- Essentially, a spring is always trying to return to rest/equilibrium

Springs

- The spring equation:

$$F = -kx - cv$$

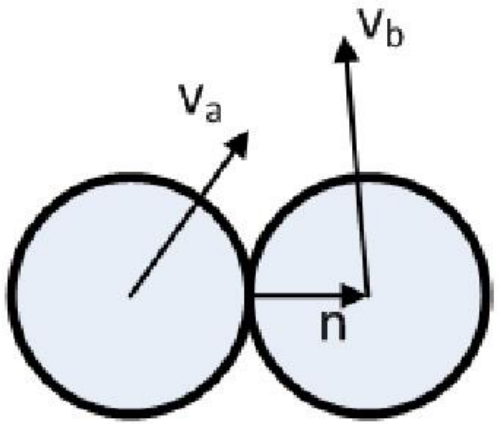
- Where k is the spring constant, x is its displacement from equilibrium, c is its damping factor (eventually slowing it to stop oscillating), v is the velocity at which it's oscillating, and F is the resultant force

- Note that the sign is negative because F is a restorative force – always trying to move back towards equilibrium

Springs in Collision Response

- Consider again the scenario we used in the earlier section.
- In this scenario, $v = v_n = v_{ab} \cdot n$, making the equation:

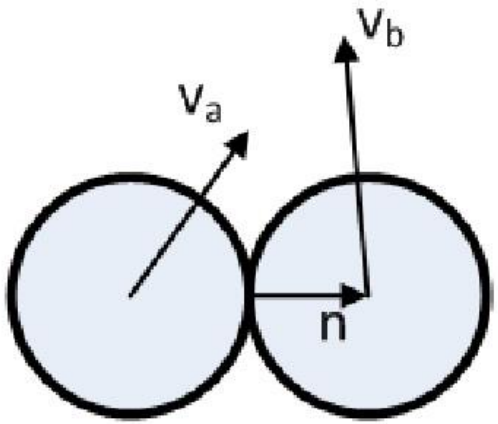
$$F = -kd - c(v_{ab} \cdot n)$$



Springs in Collision Response

$$F = -kd - c(v_{ab} \cdot n)$$

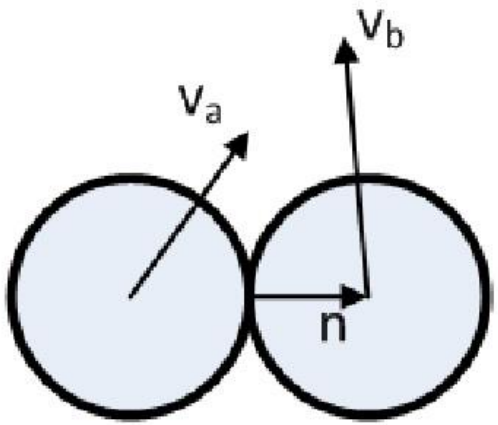
- Equal and opposite (Newton's 3rd law) forces are applied to each object involved in the collision, though obviously if one end of the 'spring' is attached to a stationary element of the environment, that end doesn't have a force applied to it (it is ignored)



Springs in Collision Response

$$F = -kd - c(v_{ab} \cdot n)$$

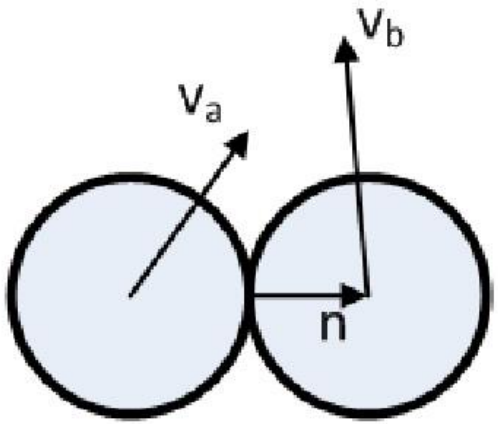
- This method results in a force, which is processed in exactly the same way as any other force (Newton's second law)
- Penalty method acts direction on acceleration



Springs in Collision Response

$$F = -kd - c(v_{ab} \cdot n)$$

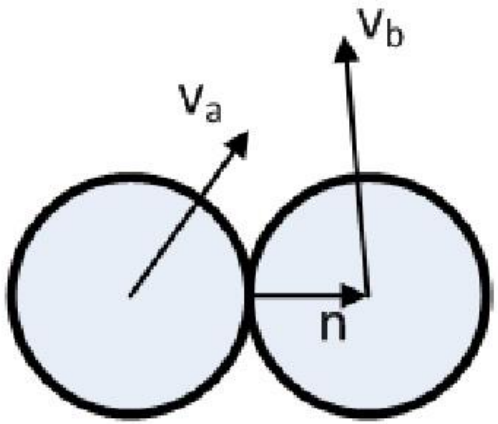
- Selection of k and c is important
  - High k makes objects stiff, low k makes them flimsy
  - High c makes objects bounce fewer times before settling, while low c makes them trampoline-like



Springs in Collision Response

$$F = -kd - c(v_{ab} \cdot n)$$

- You can implement springs as a constraint type, just as you have an implementation of distance constraints
- Have fun with this and experiment, should you decide to do it – can add some very nice effects to your physics demo

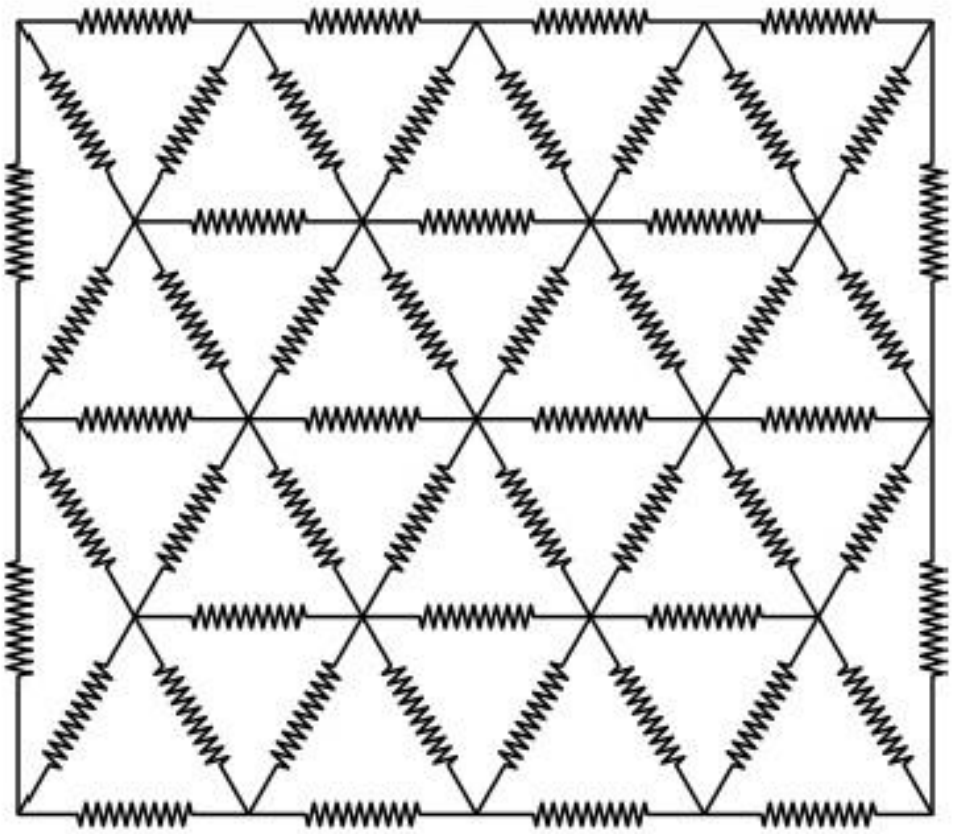


Soft Bodies

Soft Body Representation

- We've discussed Soft Bodies before in this tutorial series. We know they let us represent objects which are deformable, but how do we actually simulate them?
- Well, fundamentally, we simulate them through the constraints we have discussed over the course of this series.
- Consider a web of physical nodes, connected by constraints

Soft Body Representation



Soft Body Representation

- Some of you have already experimented with this approach, making rope bridges and pendulums with multiple distance constraints
- This is the fashion in which we would normally represent soft bodies
- A set of points connected by constraints
- Which constraints we apply depends upon the properties we want the soft body to exhibit
- Common use for spring constraints

Contact Constraint

Constraints

- Our entire physics implementation is constraint-based
- By now, we have gained quite a bit of experience exploring the consequences of a constraint-based approach to physics simulation
- We have hopefully attempted to implement our own constraints

Inequality Constraints

- Sometimes we want a constraint which will only act in a certain area of our environment
- Sometimes we want a constraint which will only act in a certain direction, or set of directions
- We can think of a ratchet example – we only want that constraint to operate within specific parameters
  - Can only rotate clockwise, never rotates anti-clockwise

Inequality Constraints

- To do this, we revisit λ
- We apply an inequality restriction on λ which limits the constraint force – what this means is

$$\lambda_- \leq \lambda \leq \lambda_+$$

- Where λ_- is some lower boundary value, and λ_+ is some upper boundary value, to our constraint force.
- In an unrestricted constraint, $\lambda_- = -\infty$ and $\lambda_+ = +\infty$

The Contact Constraint

- The contact constraint is bounded such that $\lambda_- = 0$ and $\lambda_+ = +\infty$
- As such, the force between the surfaces of our objects can only ever push out-over under this constraint
- Doesn't affect other constraints which apply to the objects

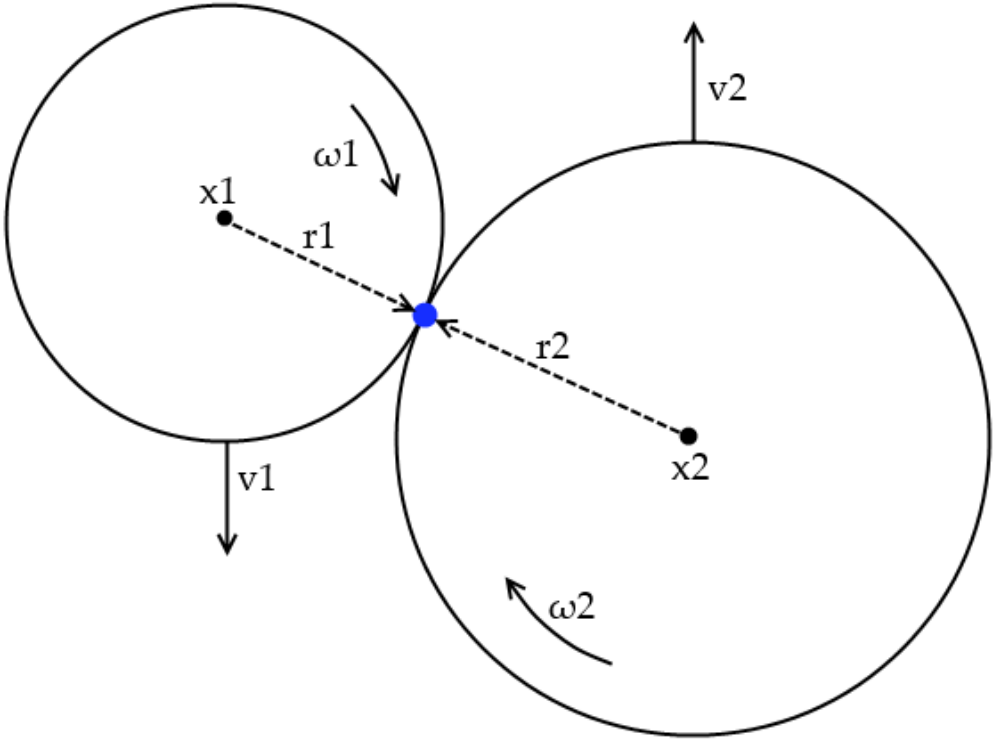
- The constraint we will use to model collision is defined

$$C = (\mathbf{x}_2 + \mathbf{r}_2 - \mathbf{x}_1 - \mathbf{r}_1) \cdot \mathbf{n}$$

Constraint-Based Collision Resolution

$$C = (\mathbf{x}_2 + \mathbf{r}_2 - \mathbf{x}_1 - \mathbf{r}_1) \cdot \mathbf{n}$$

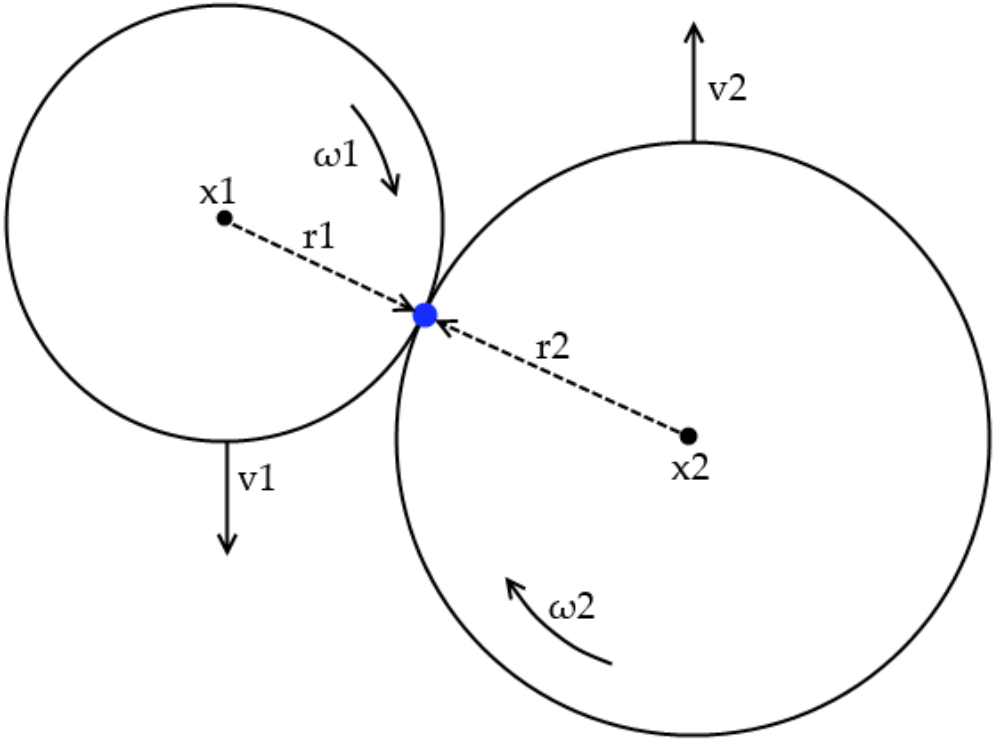
- Consider the diagram below, and how this constraint maps to its physical characteristics:



Constraint-Based Collision Resolution

$$C = (\mathbf{x}_2 + \mathbf{r}_2 - \mathbf{x}_1 - \mathbf{r}_1) \cdot \mathbf{n}$$

- We define the contact point $\mathbf{p} = \mathbf{x} + \mathbf{r}$, indicated by the blue circle on the diagram



Constraint-Based Collision Resolution

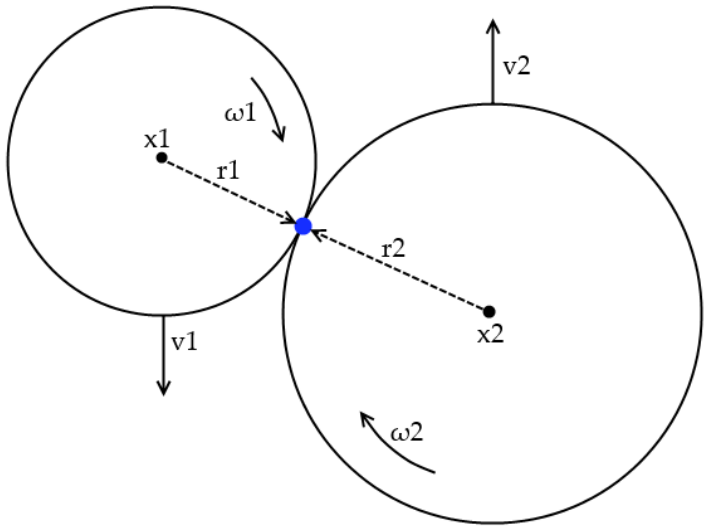
$$\mathbf{p} = \mathbf{x} + \mathbf{r}$$

- Differentiating with respect to time provides

$$\frac{d\mathbf{p}}{dt} = \mathbf{v} + \omega \times \mathbf{r}$$

- Remembering our tutorial on Constraints, differentiating C gives:

$$\dot{C} = (\mathbf{v}_2 + \omega_2 \times \mathbf{r}_2 - \mathbf{v}_1 - \omega_1 \times \mathbf{r}_1) \cdot \mathbf{n}_1 + (\mathbf{x}_2 + \mathbf{r}_2 - \mathbf{x}_1 - \mathbf{r}_1) \cdot \omega_1 \times \mathbf{n}_1$$

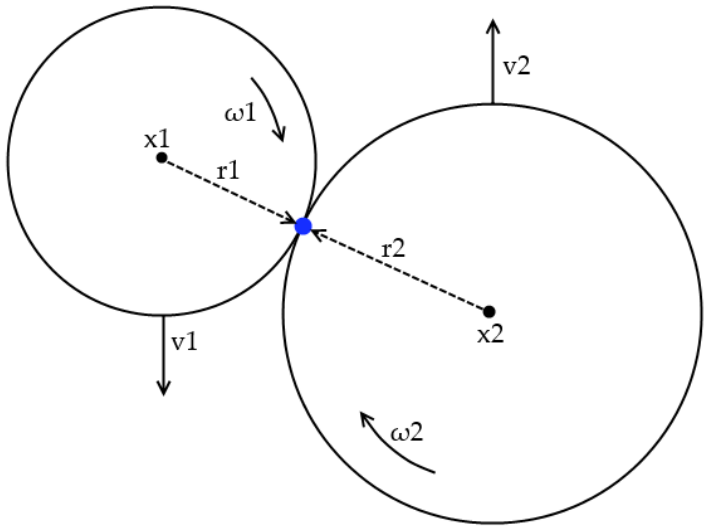


Constraint-Based Collision Resolution

$$\dot{C} = (\mathbf{v}_2 + \omega_2 \times \mathbf{r}_2 - \mathbf{v}_1 - \omega_1 \times \mathbf{r}_1) \cdot \mathbf{n}_1 + (\mathbf{x}_2 + \mathbf{r}_2 - \mathbf{x}_1 - \mathbf{r}_1) \cdot \omega_1 \times \mathbf{n}_1$$

- The second term, if we look closely, is basically our penetration depth. Assuming this value to be small enough to discount, we can rewrite cee-dot:

$$\dot{C} \approx (\mathbf{v}_2 + \omega_2 \times \mathbf{r}_2 - \mathbf{v}_1 - \omega_1 \times \mathbf{r}_1) \cdot \mathbf{n}_1$$

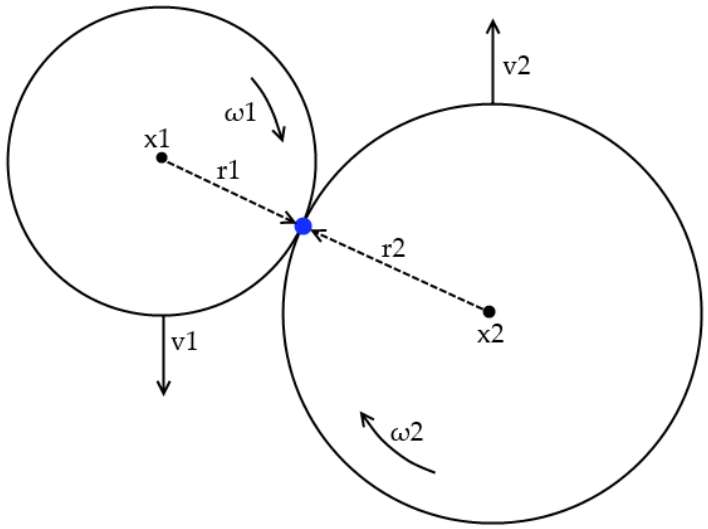


Constraint-Based Collision Resolution

$$\dot{C} \approx (\mathbf{v}_2 + \omega_2 \times \mathbf{r}_2 - \mathbf{v}_1 - \omega_1 \times \mathbf{r}_1) \cdot \mathbf{n}_1$$

- Repeating the steps we went through in tutorial 4, we can rearrange this into terms of $\mathbf{v}_1$, ω_1, $\mathbf{v}_2$, and ω_2:

- $\dot{C} \approx (-\mathbf{n}) \cdot \mathbf{v}_1 + \big(-(\mathbf{r}_1 \times \mathbf{n})\big) \cdot \omega_1 + \mathbf{n} \cdot \mathbf{v}_2 + (\mathbf{r}_2 \times \mathbf{n}) \cdot \omega_2$
- Which provides a Jacobian of the form

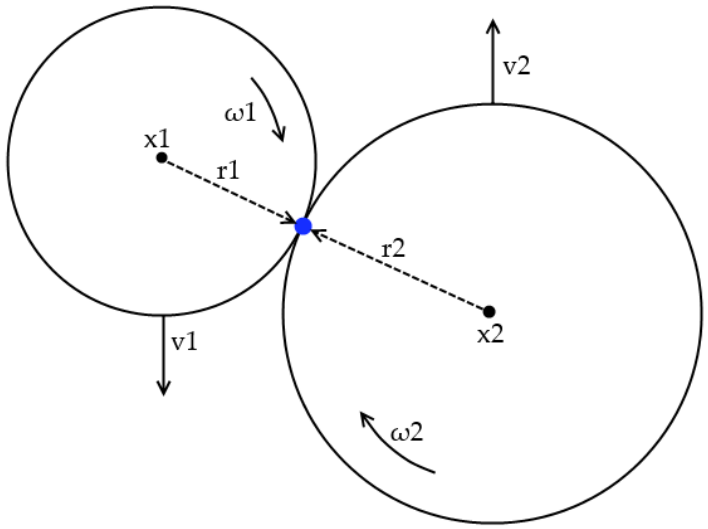


$$\mathbf{J} = \begin{bmatrix} -\mathbf{n}^T \\ -(\mathbf{r}_1 \times \mathbf{n})^T \\ \mathbf{n}^T \\ (\mathbf{r}_2 \times \mathbf{n})^T \end{bmatrix}$$

Constraint-Based Collision Resolution

$$\mathbf{J} = \begin{bmatrix} -\mathbf{n}^T \\ -(\mathbf{r}_1 \times \mathbf{n})^T \\ \mathbf{n}^T \\ (\mathbf{r}_2 \times \mathbf{n})^T \end{bmatrix}$$

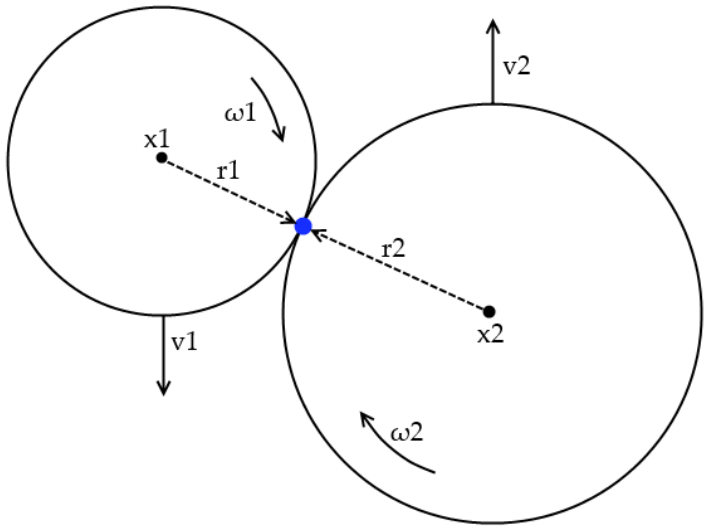
- We remember the form of the Jacobian – each element maps to one attribute of motion.
- Since the purpose of this constraint is to push the two objects apart, we apply the following restriction to our solution: $0 \leq \lambda \leq \infty$



Constraint-Based Collision Resolution

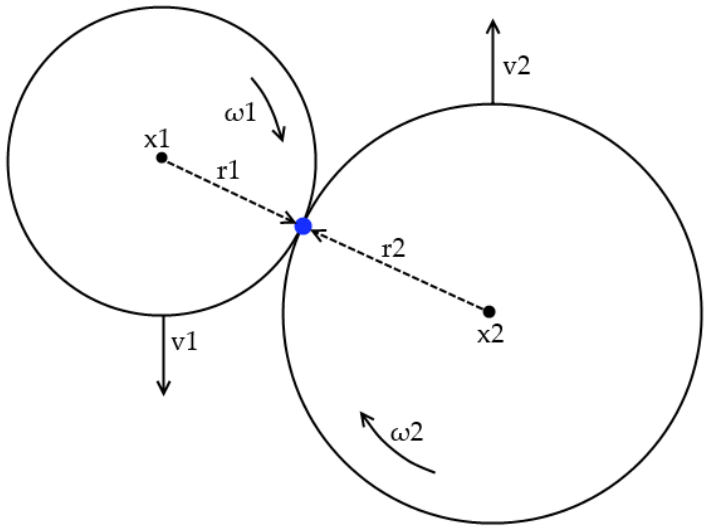
$$0 \leq \lambda \leq \infty$$

- Our collision is resolved by solving this constraint and clamping the value of λ to this region



Constraint-Based Collision Resolution

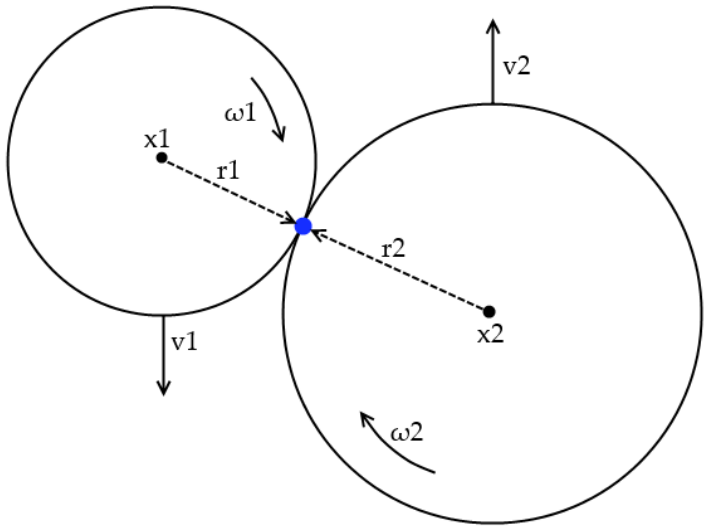
- But what about our Collision Manifold?
- We spent a lot of time computing it, aren't we meant to use it now?



Constraint-Based Collision Resolution

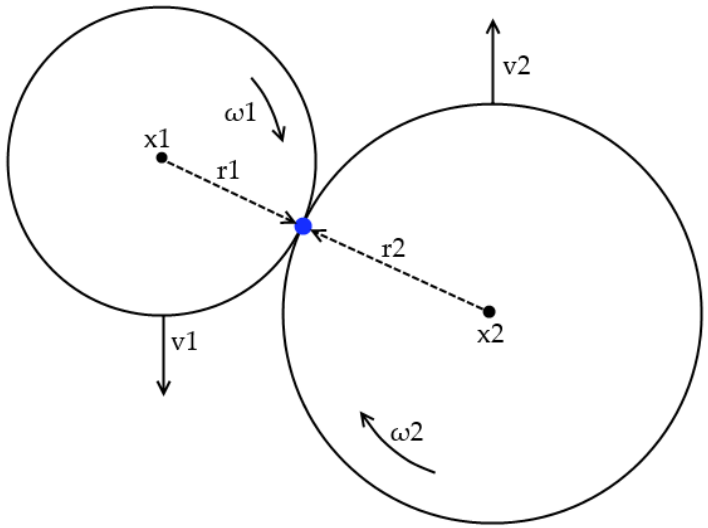
- Yes!

- There are two options as to how you can leverage the collision manifold in these scenarios



Constraint-Based Collision Resolution

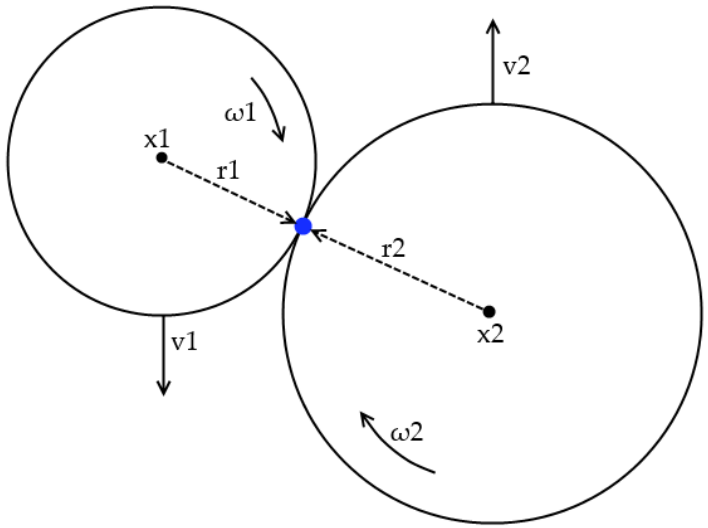
- The first sacrifices a little accuracy.
- We iterate through each contact point
- The bulk of the collision, generally, is resolved by the first contact point
- Then a chunk of the remainder by the second, and so on, and so on.



Constraint-Based Collision Resolution

- A more accurate approach would be to divide the constraint resolution between contact points

- How you would go about this is an extension to the coursework
- Experiment and have fun!



Friction

Constraints as Friction

- We can use this inequality constraint approach to model friction in our system
- We begin by defining two perpendicular unit vectors, the cross product of which is a normal to both

$$\mathbf{u}_1 \times \mathbf{u}_2 = \mathbf{n}$$

- These unit vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ are tangents to the surface of objects which are in contact – so, in the case of a cube being dragged along the ground, they might be unit vectors of the edges in contact with the ground

Constraints as Friction

- We can apply the constraint constructed to resolve collisions to $\mathbf{u}_1$ and $\mathbf{u}_2$, giving two new constraints

$$C_1 = (\mathbf{x}_2 + \mathbf{r}_2 - \mathbf{x}_1 - \mathbf{r}_1) \cdot \mathbf{u}_1$$
$$C_2 = (\mathbf{x}_2 + \mathbf{r}_2 - \mathbf{x}_1 - \mathbf{r}_1) \cdot \mathbf{u}_2$$

- Together, these two constraints allow us to restrict the movement of the contact point(s) along the surface in *any direction* along the surface
- We derive their Jacobians to be:

$$\mathbf{J}_1 = \begin{bmatrix} -\mathbf{u}_1{}^T \\ -(\mathbf{r}_1 \times \mathbf{u}_1)^T \\ \mathbf{u}_1{}^T \\ (\mathbf{r}_2 \times \mathbf{u}_1)^T \end{bmatrix} \quad \mathbf{J}_2 = \begin{bmatrix} -\mathbf{u}_2{}^T \\ -(\mathbf{r}_1 \times \mathbf{u}_2)^T \\ \mathbf{u}_2{}^T \\ (\mathbf{r}_2 \times \mathbf{u}_2)^T \end{bmatrix}$$

Constraints as Friction

$$\mathbf{J}_1 = \begin{bmatrix} -\mathbf{u}_1{}^T \\ -(\mathbf{r}_1 \times \mathbf{u}_1)^T \\ \mathbf{u}_1{}^T \\ (\mathbf{r}_2 \times \mathbf{u}_1)^T \end{bmatrix} \quad \mathbf{J}_2 = \begin{bmatrix} -\mathbf{u}_2{}^T \\ -(\mathbf{r}_1 \times \mathbf{u}_2)^T \\ \mathbf{u}_2{}^T \\ (\mathbf{r}_2 \times \mathbf{u}_2)^T \end{bmatrix}$$

- We approximate the limits on our λ based on a realistic approximation of the force an object acting under gravity would experience as friction (meaning mg, multiplied by some constant μ) such that

$$-\mu mg \leq \lambda \leq \mu mg$$

- This applies to both constraints, and we can experiment with μ to find a value which suits our simulation

Constraint Drift and Baumgarte

Constraint Drift

- We introduced the correction factor for Constraint Drift in Tutorial 4, using the equation

$$\mathbf{JV} = -\beta \mathrm{C}$$

- This is the Baumgarte scheme, and acts to compensate for Constraint Drift

- Constraint Drift is, in part, a consequence of our time-stepped physics simulation – more on this later
- Because our results for time step t_{n+1} are based on erroneous results from time step t_n, etc.

Constraint Drift

- When we first introduced constraints, we stated they shouldn't add or remove energy from the system, e.g.

$$\dot{C} = 0$$

- Once we consider the error that's slowly entering the system, it's more accurate to say that

$$\dot{C} \approx 0$$

Baumgarte Offset

$$\dot{C} \approx 0$$

- If the error is related to time, and the constraint itself, then we can reason (or prove, if you want to read the paper), the following

$$\dot{C}(t) + \beta C(t) = 0$$

- We clamp the Baumgarte constant to lie in the region

$$0 < \beta < \frac{1}{\Delta t}$$

Baumgarte Offset

$$0 < \beta < \frac{1}{\Delta t}$$

- This is one reason for us using a fixed time step, as it means the boundaries of our Baumgarte offset are static
- If we varied time step, we would need to investigate the derivation of the Baumgarte offset and recompute it with each time step
- The time taken to do so would, by necessity, make the resulting computation inaccurate in any case

Summary

- Introduced three methods of collision response
- Discussed mathematical underpinnings of Impulse-based collision response
- Introduced the spring equation as the basis for Penalty-based collision response
- Discussed soft-body representation as a mesh of nodes connected by constraints

- Introduced contact constraints
- Discussed Friction
- Discussed error-correction (Baumgarte)