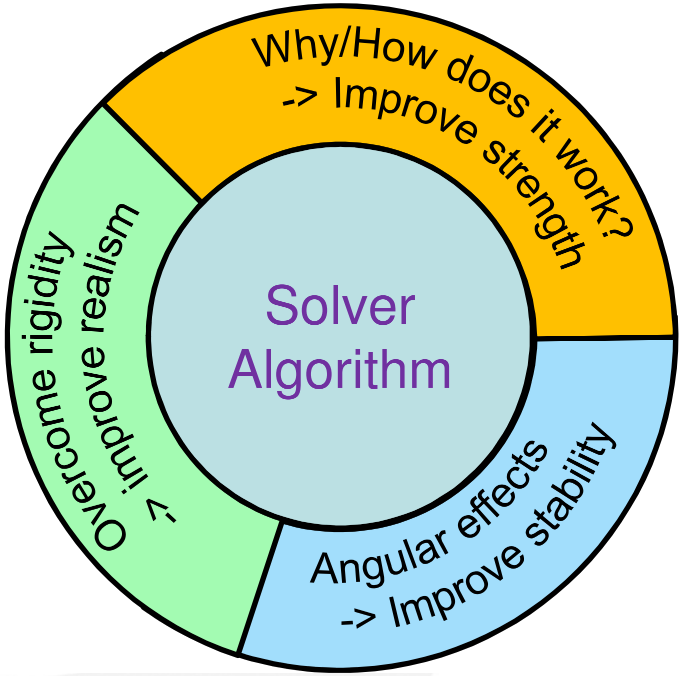
Stop my Constraints from Blowing Up!

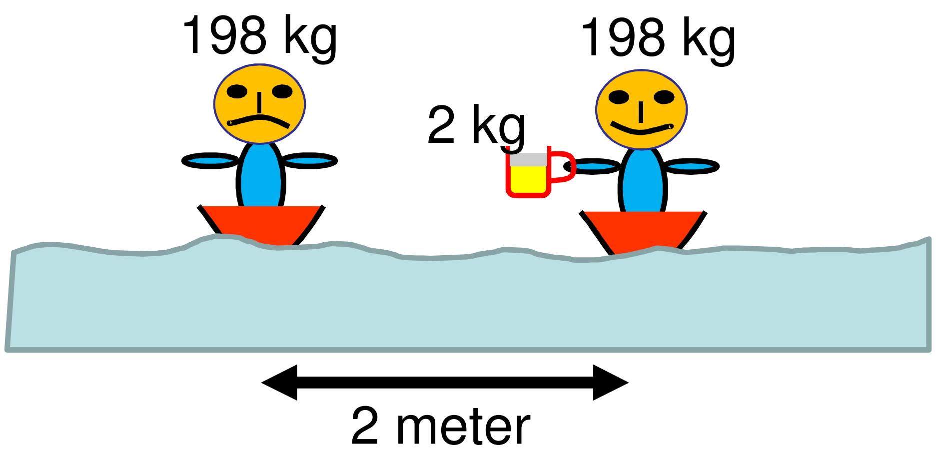
Oliver Strunk
(Havok)

Overview

Just understanding a rigid-body solver algorithm is not enough.
(google for ‘ragdoll glitch’ and enjoy)



Chapter I: How does a solver work?



‘Position’ only Solver

Each Bavarian moves the mug to his boat every other second.

- The center of mass of ‘Bavarian + Beer’ must stay constant:

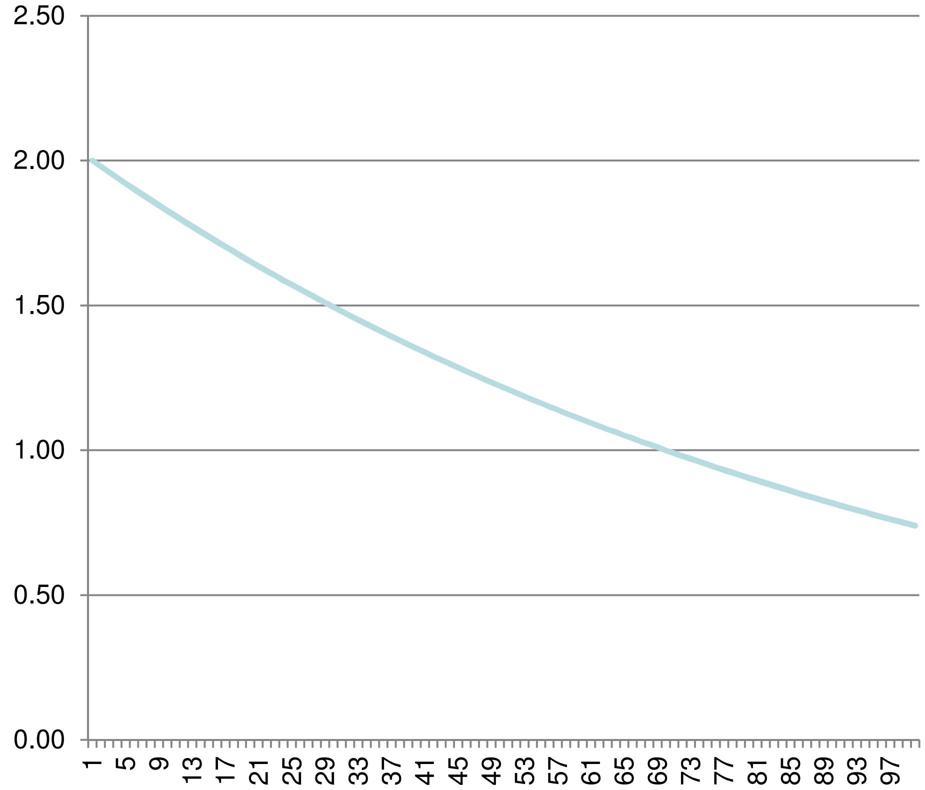
  the mass ratio   mug : Bavarian = 1:99

  -> the mug will move 198cm and the Bavarian 2cm.
  In the end the mug and the Bavarian moved and all velocities are 0.

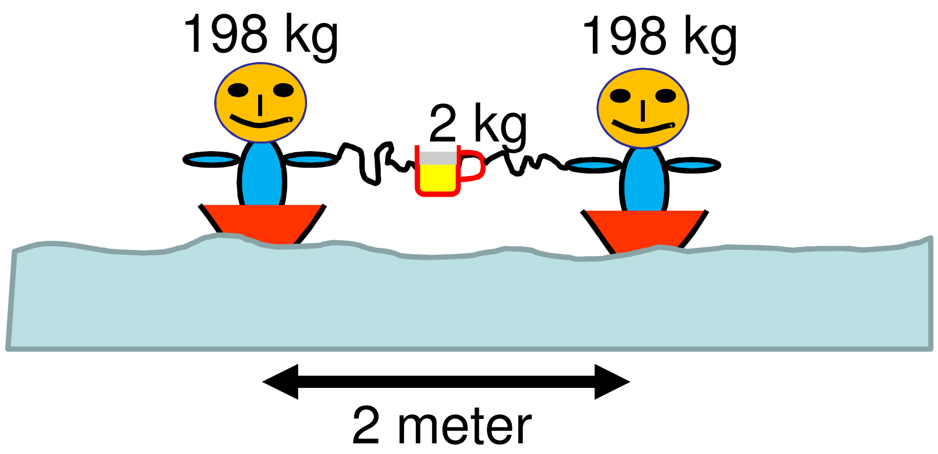
- In the next second the other Bavarian will do the same.
- Repeat

Lets iterate

Step	Distance	
1	2.00	
2	1.98	
3	1.96	
4	1.94	
5	1.92	
10	1.83	
100	.74	~2/e



Now use velocities



‘Position and Velocity’ Solver

Each Bavarian now pulls the rope every other second such that the mug arrives at his boat after one second.

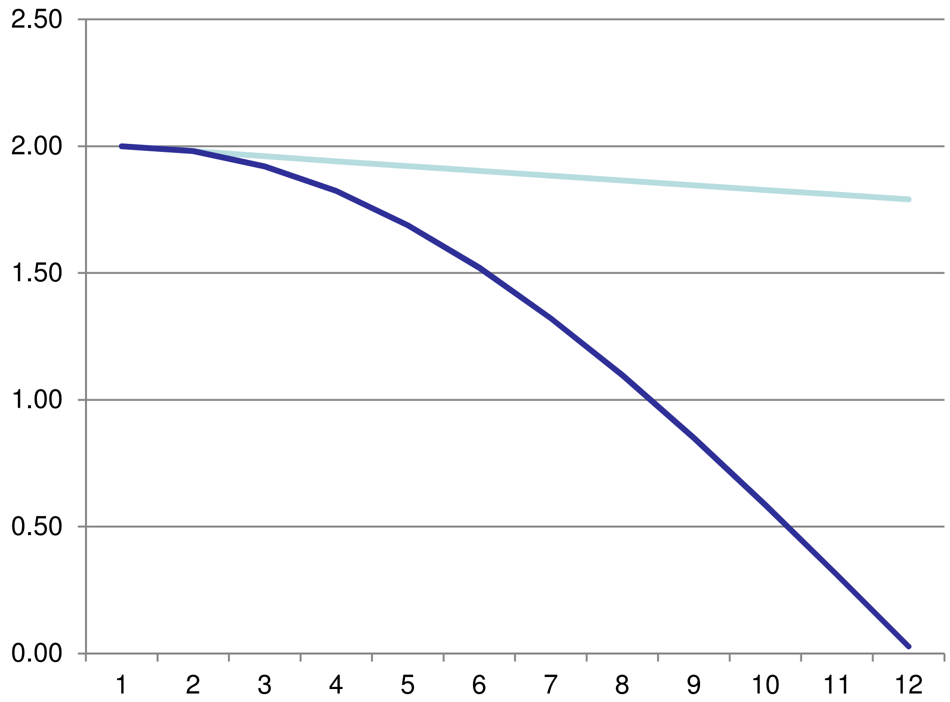
- Mass * velocity (moment) must stay constant:

  -> If the beer is accelerated to 1.98 meters/second, the Bavarian will be accelerated to 2.0cm/second.
  After one second: the mug reaches the boat and *both* mug and Bavarian have some velocity.

- In the next second the other Bavarian will pull the rope and reverse the velocity of the mug.

Let's iterate

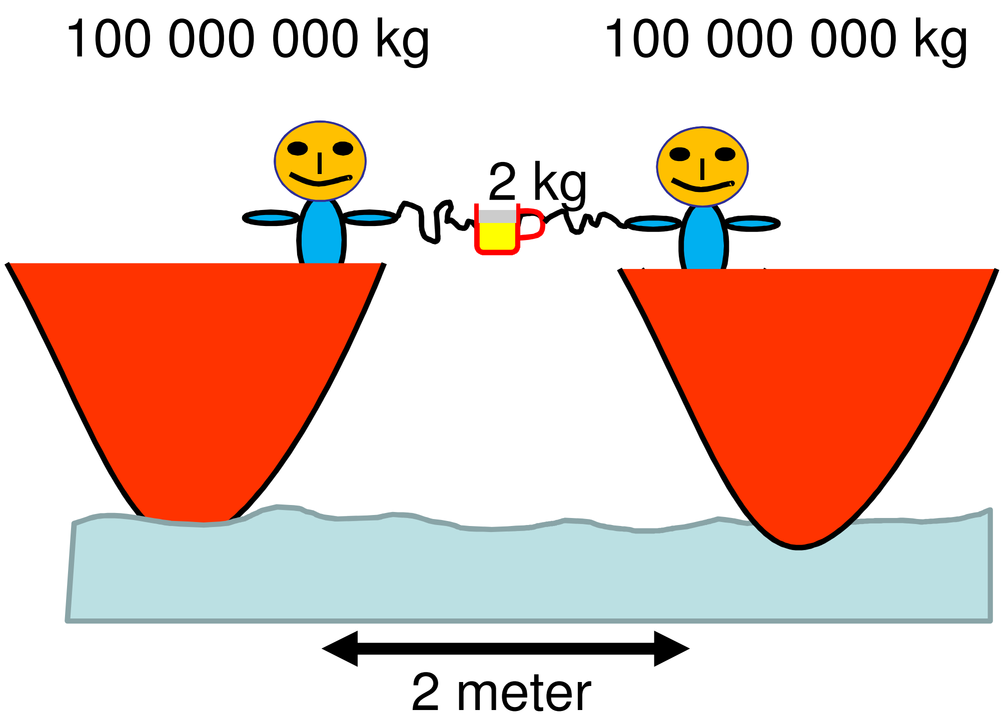
Step	Distance	Relative Velocity	Distance Prev. Solver
1	2	0.020	2.00
2	1.98	0.060	1.98
3	1.92	0.098	1.96
4	1.82	0.134	1.94
5	1.69	0.168	1.92
6	1.52	0.199	1.90
7	1.32	0.225	1.88
8	1.10	0.247	1.86
9	0.85	0.264	1.85
10	0.59	0.276	1.83
11	0.31	0.282	1.81
12	0.03	0.282	1.79



Lessons Learned

- 2 constraints fighting each other iteratively solves the global problem (=bringing the boats together).
- We saw 2 types of iterative solvers:
  - Strength ~ num iterations (convergence)
  - Strength ~ num iterations^2, but
    - this adds energy

Can We Pull an Ocean-Liner Using a Mug?



7000 iterations of our beer-mug solver will move this ship by 1 meter.

Lets look into details.

Solver ‘strength’ depends on:

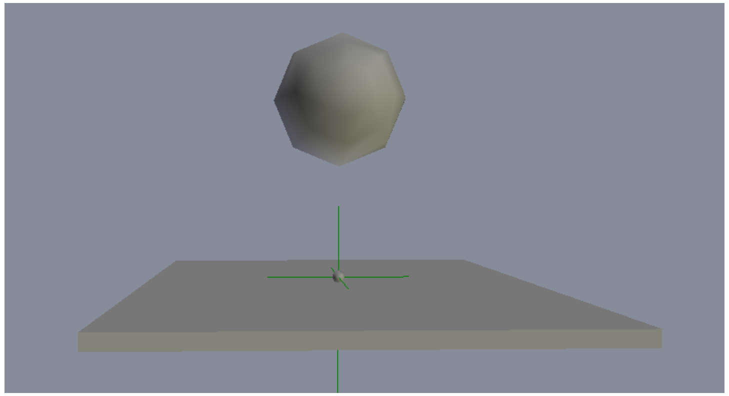
- Solver type (e.g. linear/quadratic)
  - We want fast convergence without the risk of instability.
- Number of solver iterations (per second)
  - We want as few as possible to save CPU.
- Mass ratio of the objects involved
  - Needs to be as small as possible to allow for small number of iterations.

Solver Type

Statement:

- All **quadratic/linear** convergence solvers have similar ‘strength’.

- Solver variants found in the literature:
  - Position/velocity based solver.
  - Error correction by post-projection, split impulse or Baumgarte-stabilization.



Number of Solver Iterations

- Say n=iterations/sec, d=distance
  - > we accelerate the body n-times per second to d*n velocity -> acceleration = n^2 *d

n [Hz]	d [meter]	acceleration	acc/gravity
30	0.05	45	4.5
120	0.05	720	72
240	0.01	576	57.6
1000	0.10	100000	10000

Number of Solver Iterations

Observations:

- Running one solver iteration per frame (30Hz) is not good enough.
- Running a pure quadratic convergent solver higher than frame frequency can lead to instability.
- Most game physics engine solvers use quadratic convergence using frame frequency (30Hz) and use linear convergence using sub-iterations (4-10).

Mass Ratio:

High mass ratios require lots of solver iterations, so keep mass ratio low!

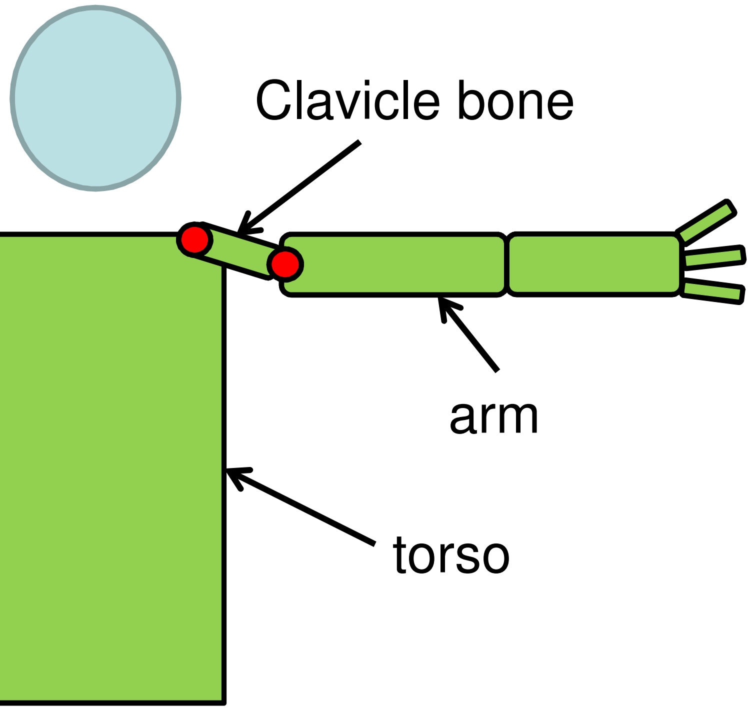
Avoid calculating the mass from density automatically:

- Guessing density is tricky:
  - What is the density of a car / a 747 ?
  - Problem: solid vs. hollow objects
  - -> So let your artist set the mass not the density.
- A tank driving either over a 10cm metal or a wooden box makes no difference:
  - Increase your masses on small debris objects
  - Exceptions: bullets and rockets
- Advice: Don’t tweak mass if you don’t have a problem yet.

Mass-Ratio in a 3d-World

- In a 1d-universe, the mass ratio just depends on the mass of the objects.
- But games are not 1d ☹, so 2d/3d rigid bodies can rotate. As a result the mass ratio depends on mass **and** inertia (=‘angular mass’).

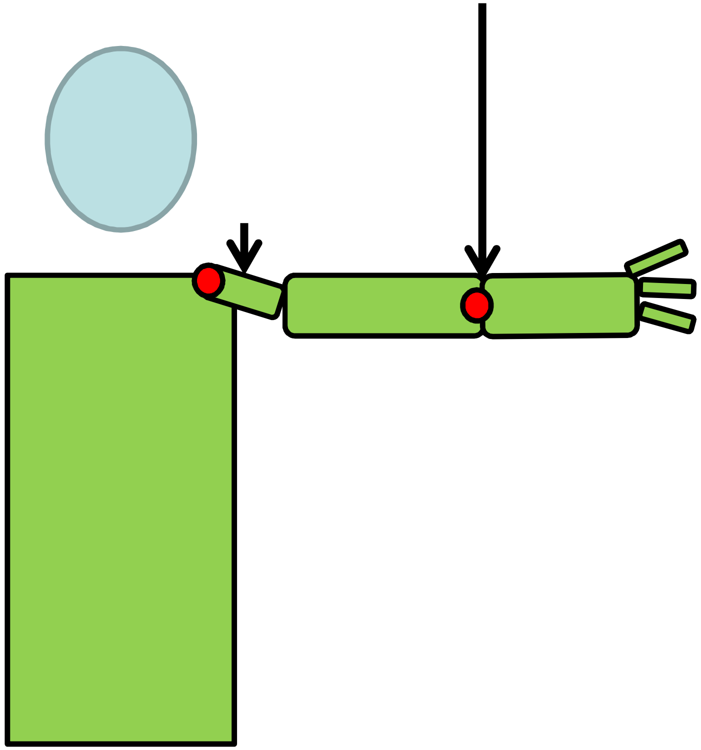
Typical 'Bad' Example



Lets assume:

- All joints are limited.
- All joints have reached their limit.
- Shoulder bone is 4x smaller than arm.
- Mass is calculated from density.

Let’s simplify



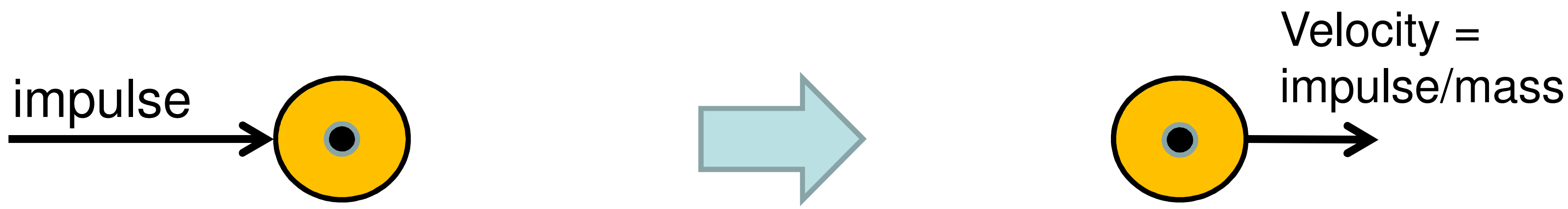
Lets move the pivot to the mass center of the combined arm. (in a fixed constraint we can do this without changing behavior).

We see: We virtually apply a force at the shoulder bone way outside its shape.

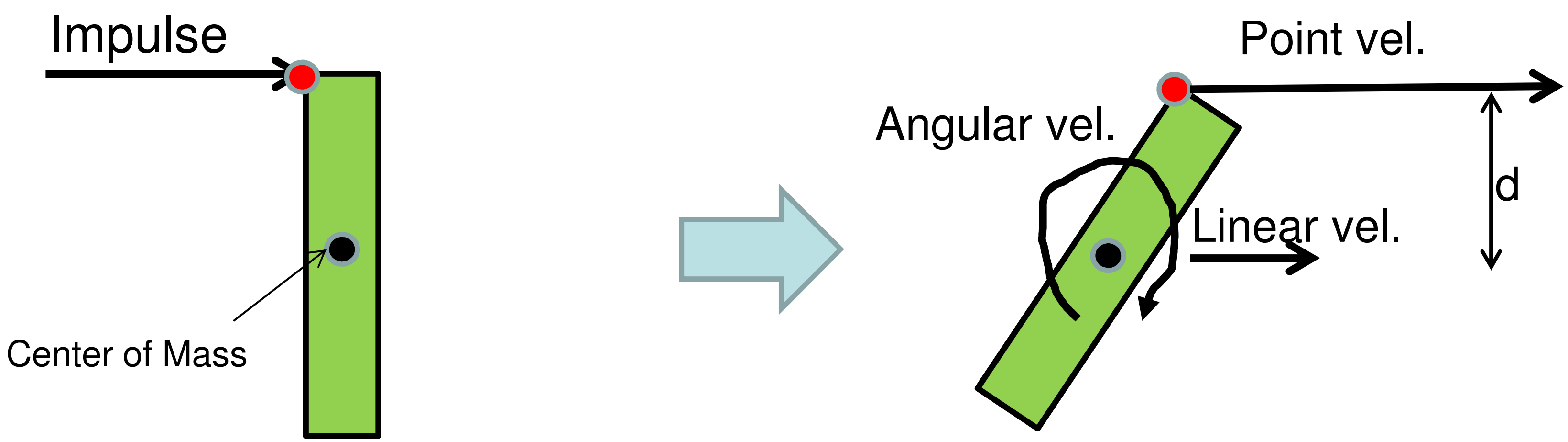
Is this bad?

Yes, it results in very poor solver strength because the **‘effective’** mass ratio gets extremely high (1 : 3000)

Effective Mass



Effective Mass



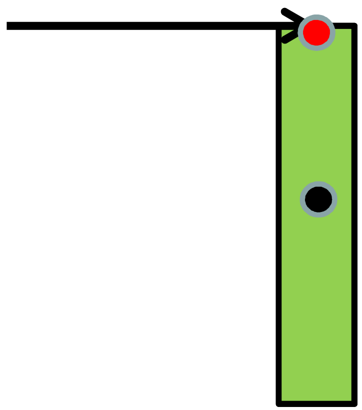
velocity at ● = linearVel + d*angularVel
= impulse/mass + d*impulse/angularMass(d)

angularMass(d) = inertia/d

velocity = impulse / effMass

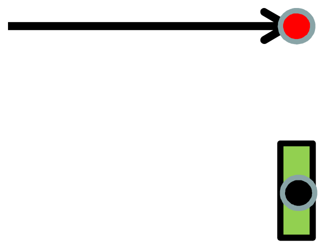
effMass = 1 / (1/mass + d^2/inertia)

Effective Mass Example



- Dimensions: 1meter * 20cm * 20cm
- Mass: 100kg (= density: 2.5kg/litre)
- Distance: ↔ = .5 meter.
- Inertia: 100/12* (1*1 + .2*.2) = 8.6 kg m^2
- Effmass at = $1/(1/100 + .5^2/8.6)$ = 25kg

Lets make the cube 4x smaller



- Dimensions: .25meter * 5cm * 5cm
- Mass: 1.56kg (= density: 2.5kg/litre)
- Distance: ↔ = .5 meter.
- Inertia: 1/12*1.56*(.25*.25 + .05*.05) = 8.46e-5

- Total effMass at = 33.13g

cube 4 times smaller -> eff.Mass drops by 750 !!!

Summary Chapter 1

- Solver strength mainly depends on:
  - Number of solver iterations
  - Mass ratio
- Rotations are bad!
  - Angular movement reduces our effective mass and therefore our solver strength.

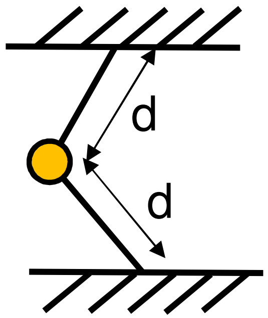
Chapter 2: Angular Effects

Angular movement not only reduces effective mass but also leads to instability!

Demo

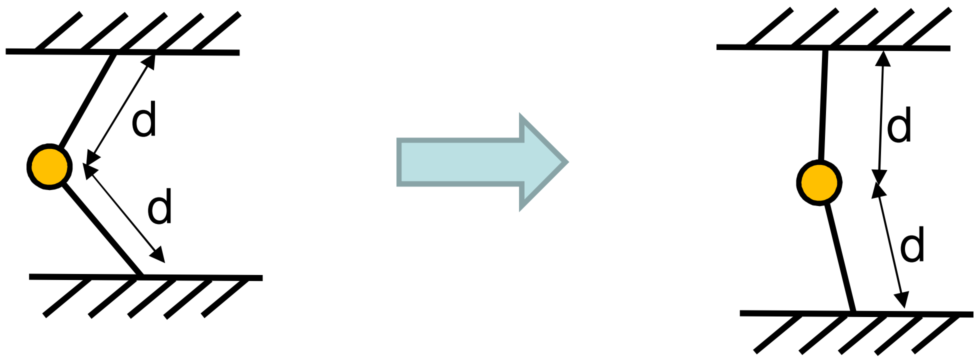
Angular Effects

We have 2 distance constraints (distance d) between a ball and 2 fixed walls.



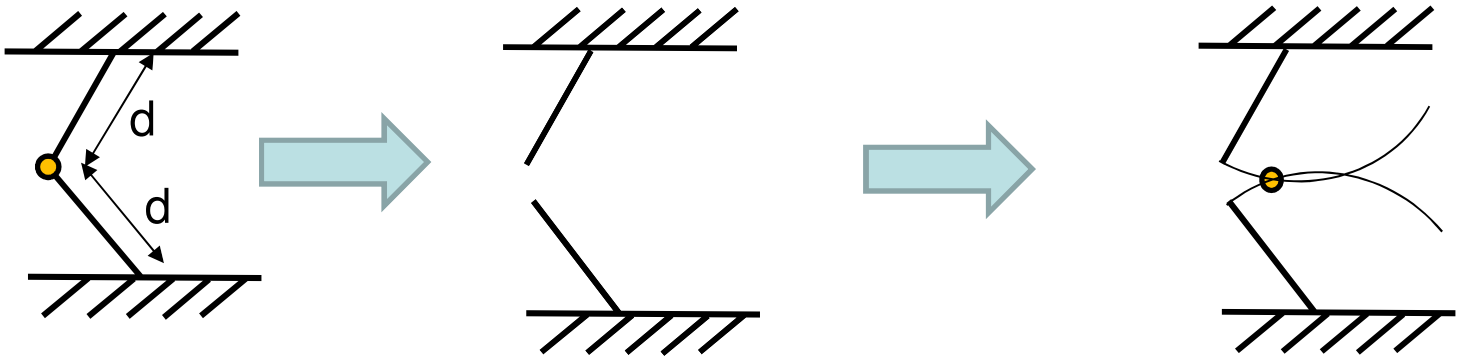
Angular Effects

- If we move the walls apart, we'll have to move the ball to satisfy the constraints:



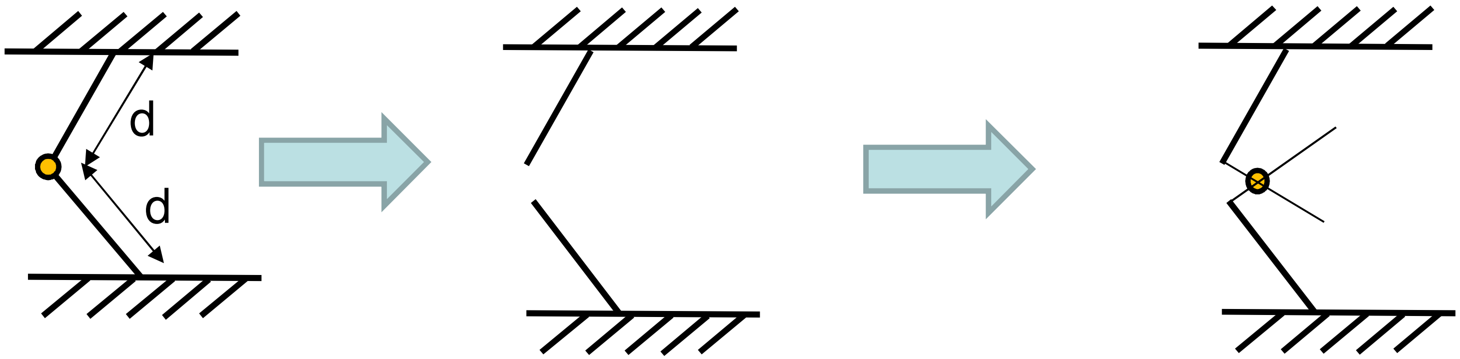
Angular Effects

How will a perfect algorithm solve this?



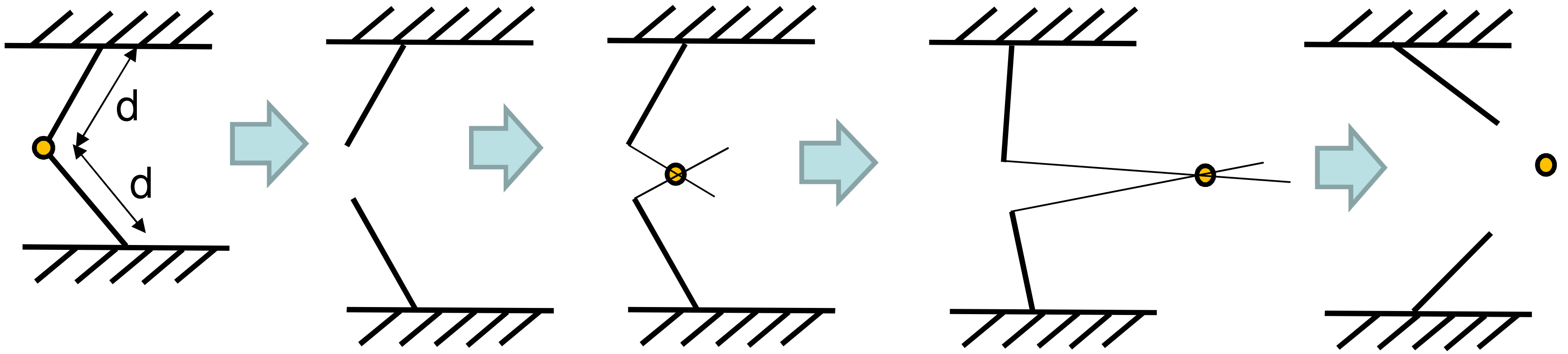
Angular Effects

We are forced to linearize the equations of motion.



Angular Effects

- Lets move the walls a little bit further:



Angular Effects

- A solver will gain energy if it “overshoots” too much.
- The likelihood of overshoots increases if the solver ‘miss-predicts’ the angular movement:
  - Because of linearization, the force direction (=Jacobian) is not optimally chosen, especially when running ‘building the jacobian’-algorithm at low frequency.
  - Angular velocity is high compared to linear velocity.
  - Effective angular mass is much less than the body mass and high forces are applied:

$$\textbf{inertia}/\textbf{d}^2 \;\; / \;\; \textbf{mass} \;\; \ll 1.0$$

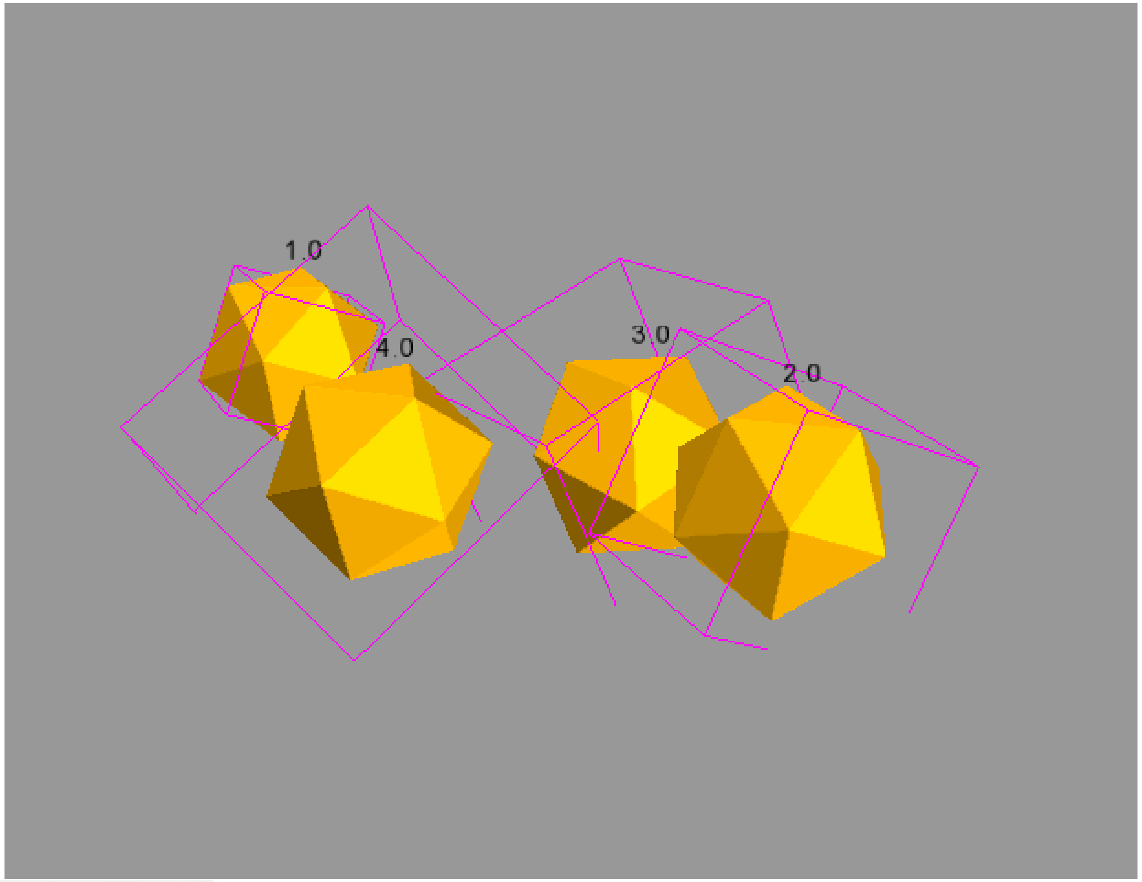
Lessons Learned

- Low inertias are the main problem for bad solver behavior!!!
  - They increase the effective mass ratios between bodies
  - They lead to high angular velocities, which lead to ‘explosions’/jitter

- Solution:
  - Increase inertias

Increasing Inertia

- Demo



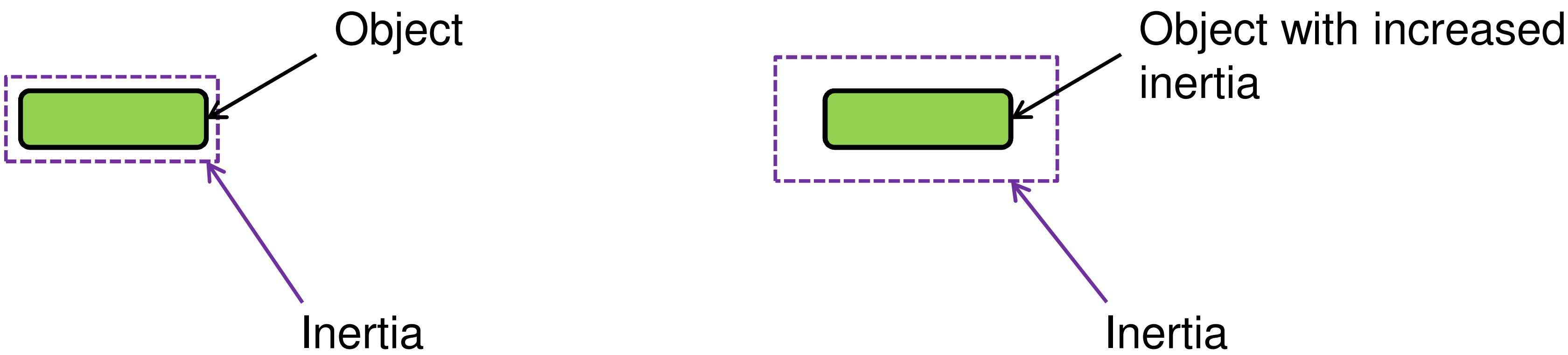
Increasing Inertia

Lessons learned:

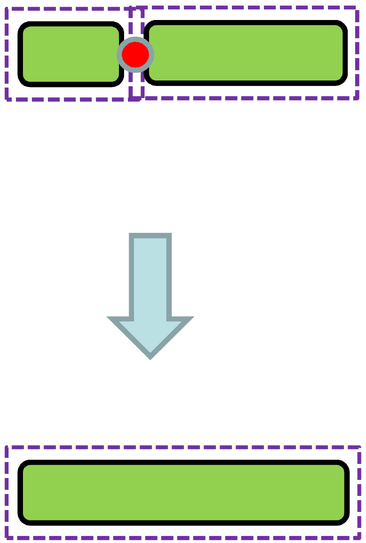
- We can increase inertia of selected single objects easily by factor of 2 - 4 before users spot serious artifacts.

Inertia Visualizations

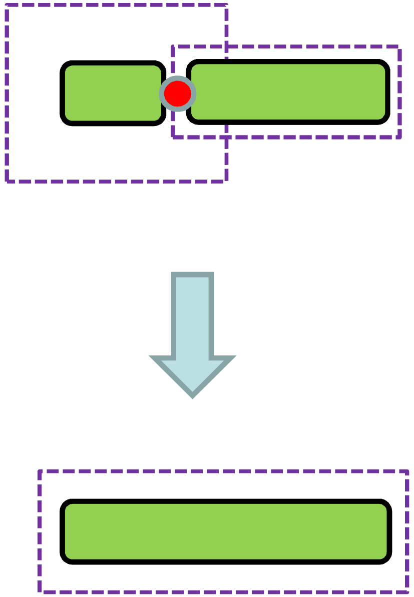
We can visualize the inertia by drawing a box which would have the same inertia.



Combining Inertias

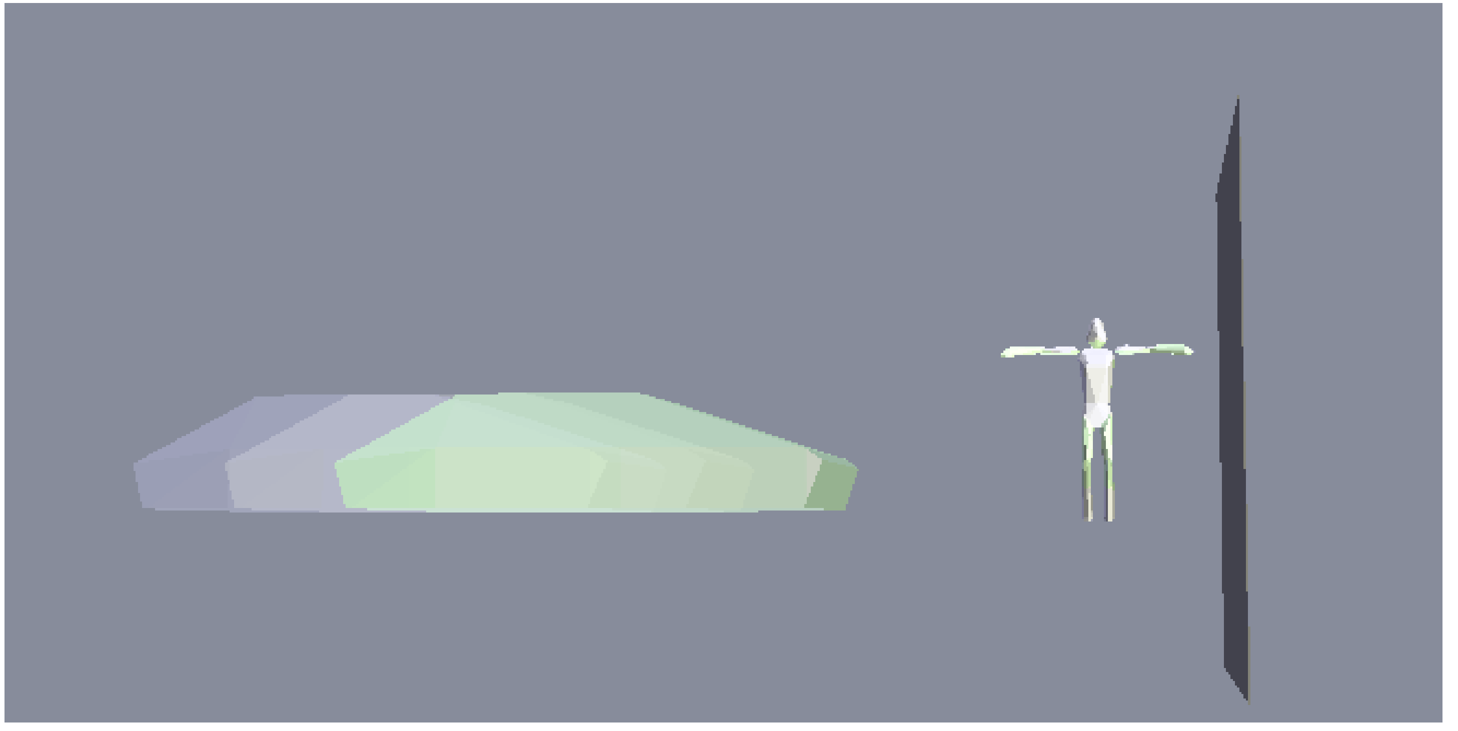


Combining Inertias



Demo Of Ragdoll With Increased Inertia

- Demo

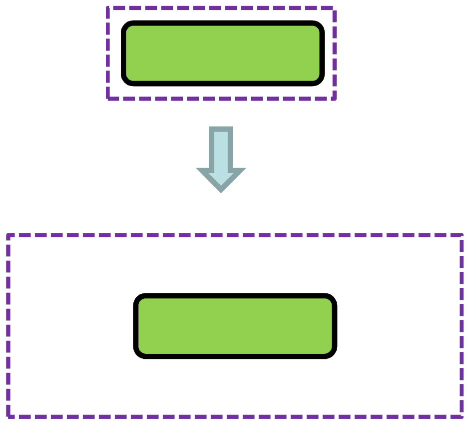


Lessons Learned

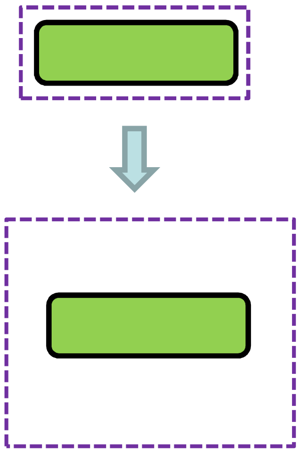
- Increasing inertia matrix is very often quite acceptable in a game environment and artifacts are hardly noticeable.
  - Especially true for small bodies inside a chain of constraint bodies (like shoulder bone).

How to Increase the Inertia Matrix

- Multiply by a factor
  (single bodies)



- Add a constant to the diagonal
  (bodies in a constraint chain)



Summary Chapter 2

To improve solver stability and ‘strength’:

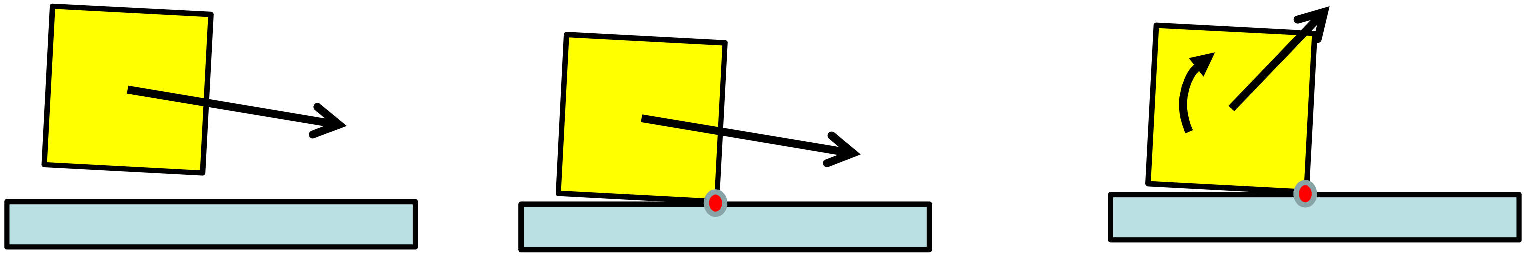
- Increase inertia
- Increase simulation frequency
- Decrease mass ratio

Advice: Don’t try to damp rigid bodies or the solver

Chapter 3: Reducing Rigidity

Our physics engine uses rigid bodies. Rigid means 100% rigid.

– Bodies never deform.
– Bodies never break.
– Impulses and friction forces have no limit:
  -> Physics engine can feel ‘cute and bouncy’



Implementing Soft Contact

- Reduce the maximum contact impulse to allow for some penetration.
  - E.g. clip the impulse.
- Ensure that the penetration recovery happens slowly to avoid springy behavior.
  - E.g. reduce the Baumgarte stabilization for this contact.

Soft Contact

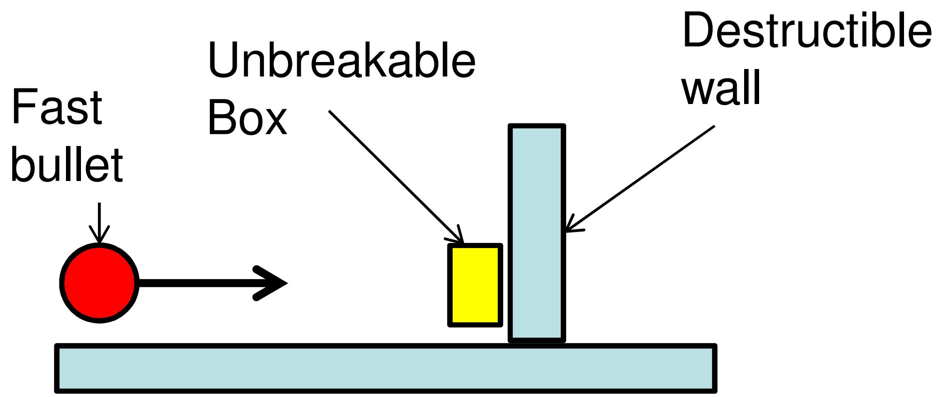
- Demo:

Destruction

Idea: destroy bodies if impulses get too high.

Destruction

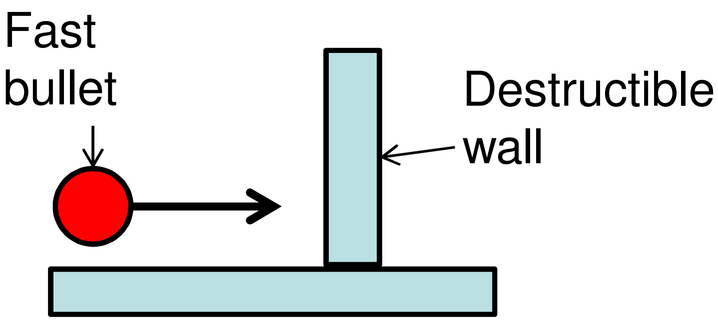
- Solution 1: Estimate the contact impulse and destruct the object **before** running the solver:
  - Impulse = relativeVelocity * effectiveMass
  - This works pretty well, except if the breakable object is blocked by an unbreakable object.



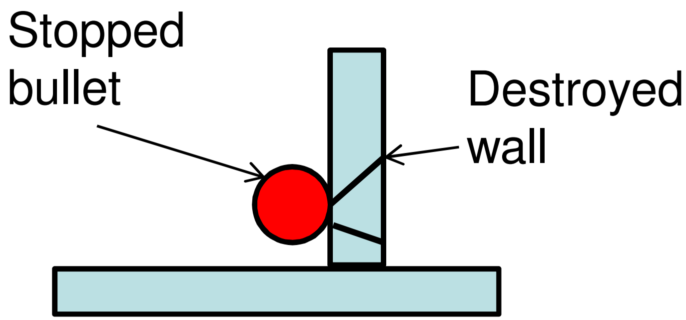
Destruction

- Solution 2: **After** running the solver:
  If the contact impulse exceeds a limit, break the object
  – If implemented naively, breakable fixed objects will stop any moving object.

1st frame:



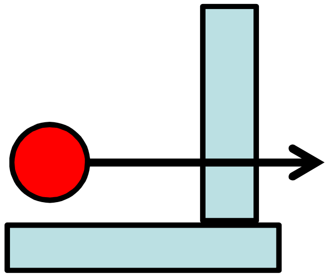
2nd frame:



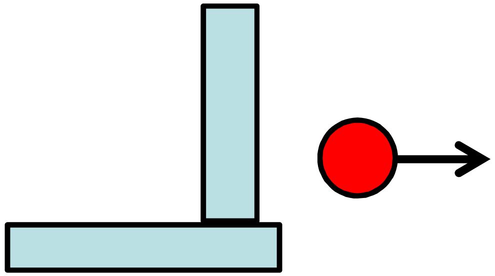
Destruction

- Solution 2b: Extend solution 2 with the following modifications:
  - The solver also clips the impulses to the breaking limit.
  - If an object breaks, move the two objects involved to the previous position.

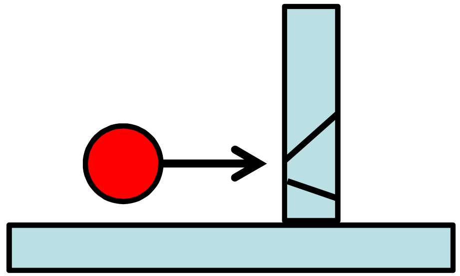
1st frame:



2nd frame
Just after solve:



2nd frame after destruct
and position move:



Destruction

- Demo

Summary Chapter 3

- Impulse clipping in the solver allows to emulate soft, deformable or breakable materials.

The End

Thanks for listening, questions welcome ☺