GDC 13

Iterative Rigid Body Solvers

Richard Tonge
Principal Software Engineer, NVIDIA

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 25–29, 2013
EXPO DATES: MARCH 27–29
2013

Arena destruction demo

Demo credits

Matthias Müller-Fischer
Nuttapong Chentanez
Tae-Yong Kim
Aron Zoellner
Kevin Newkirk

Hawken Demo

SIGGRAPH paper

Mass Splitting for Jitter-Free Parallel Rigid Body Simulation

Richard Tonge, Feodor Benevolenski, Andrey Voroshilov

Mass Splitting for Jitter-Free Parallel Rigid Body Simulation

Richard Tonge NVIDIA* Feodor Benevolenski NVIDIA Andrey Voroshilov NVIDIA

Figure 1: Left: 5000 boxes with 40000 contacts coming to rest on a non-convex triangle mesh without jitter, simulated on an NVIDIA GTX580 in over 60 FPS. Middle: Mass splitting used in a real-time fracture simulation. The debris have irregular shapes and large mass ratios. Right: The method allows us to simulate large scale building destruction in real-time in a video game.

Abstract

We present a parallel iterative rigid body solver that avoids common artifacts at low iteration counts. In large or real-time simulations, iteration is often terminated before convergence to maximise scene size. If the distribution of the resulting residual energy varies too much from frame to frame, then bodies close to rest can visibly jitter. Projected Gauss-Seidel (PGS) distributes the residual according to the order in which constraints are processed, so ensuring this order in parallel implementations is very challenging. In contrast, Jacobi-based methods provide order independence, but have slower convergence. We accelerate projected Jacobi by dividing each body mass term in the effective mass by the number of contacts acting on the body, but use the full mass to apply impulses. We further accelerate the method by solving contacts in blocks, providing warm-start performance competitive with PGS while avoiding visible artifacts. We prove convergence to the solution of the underlying linear complementarity problem under standard assumptions. Our implementation, which can simulate a pile of 5000 objects with no visible jittering at over 60 FPS.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

Keywords: rigid bodies, non-smooth dynamics, contact, friction

Links: DL PDF

*e-mail: tonge@nvidia.com

1 Introduction

Rigid body dynamics is widely used in applications ranging from movies to engineering to video games. Piles of objects are particularly common, because ultimately, gravity pulls all rigid bodies to the ground. Some of the most visually interesting simulations involve destruction, such as projectile impacts and explosions, and these can generate large piles of debris. In mechanical engineering some of the most computationally challenging problems involve simulating interaction with large resting systems of soil particles or rocks. Piles require stable simulation of static friction, dynamic friction and resting contact, which presents many challenges.

In large or real-time simulations, the computation budget can be small compared to the number of rigid body contacts. In these cases we must use iterative methods, terminating the iteration before convergence. By stopping early we introduce residual energy into the system, which can cause objects near rest to jitter. See the accompanying video for examples of these artifacts.

A commonly used iterative algorithm is projected Gauss-Seidel (PGS), which solves contacts in sequence. When we terminate the iteration, the last contact solved has no error and the other contacts have non-zero error, resulting in an uneven distribution of the residual energy. On single threaded implementations we can ensure that the ordering of the contacts is consistent from frame to frame, for example, ensuring that collisions are detected in the same order each frame and that addition and deletion of bodies do not disrupt the order. Unfortunately, things are not so straightforward for parallel implementations.

Due to the widespread availability of multi-core CPUs and GPUs, parallel computing is increasingly being used to simulate rigid bodies. PGS has limited parallelism, as updates to bodies having multiple contacts must be serialized to ensure they are not lost and the connectivity between bodies can be complex, especially in piles. This serialization changes the order in which constraints are processed, often changing it dramatically from frame to frame. Operating systems and GPU schedulers can also introduce non-determinism into the order of operations. For these reasons it is very challenging to avoid jittering with parallel PGS, and due to the serialization its performance does not scale well as more threads are added.

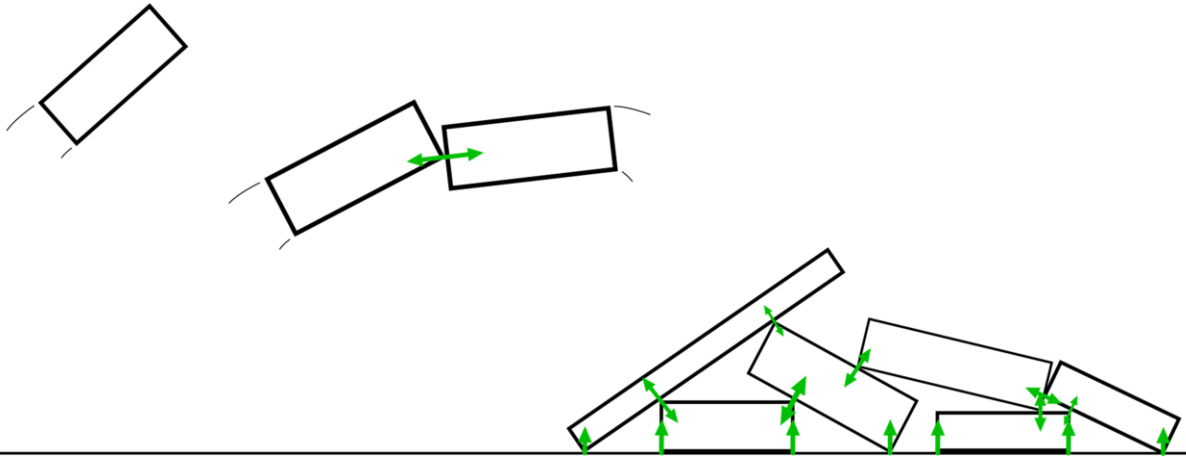
In real-time applications such as games, the designer does not know in advance what the player is going to do and which objects are

Contents

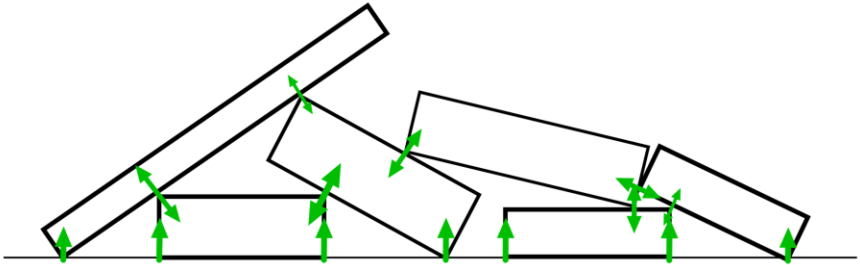
- What do we need rigid bodies to do?
- Single core solver
- Parallel solver
- Results

Section 1

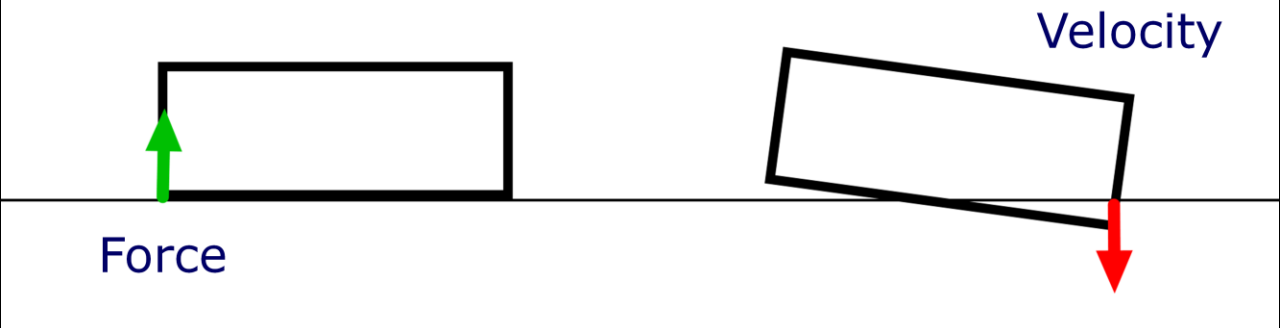
What are rigid bodies supposed to do?



Stable Piles = Balanced forces



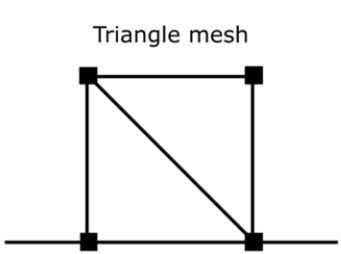
Contacts affect each other

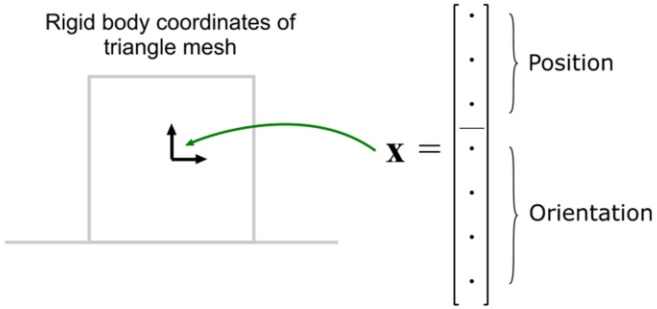


Section 2

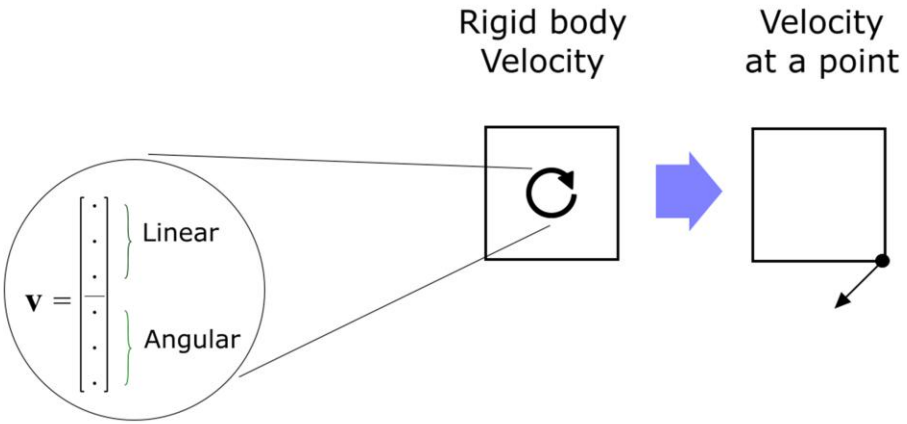
How to make rigid bodies do the right thing using a solver (single core)

Rigid Body Basics



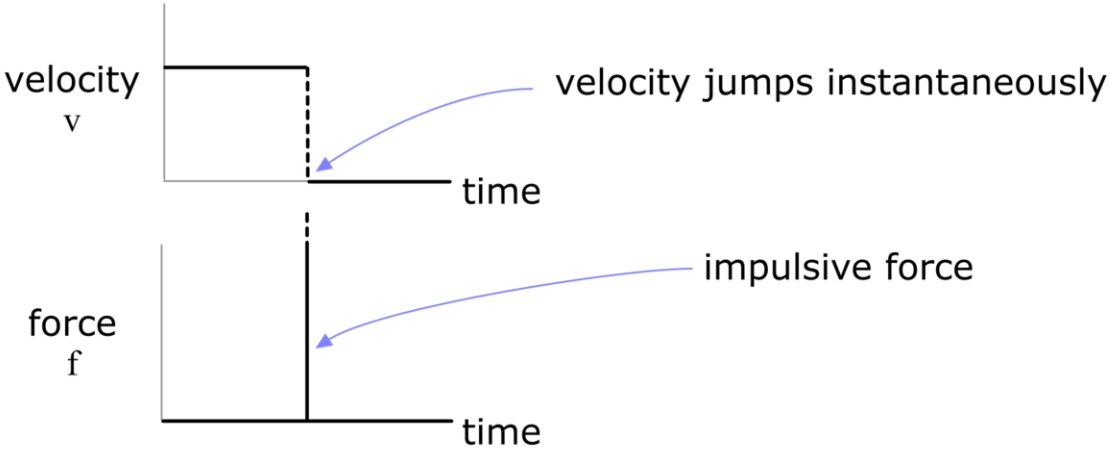


Velocities

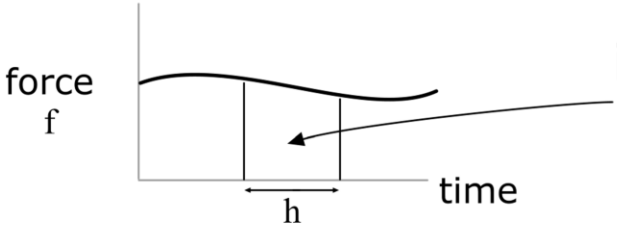


Use impulses and velocities

What is an impulse?



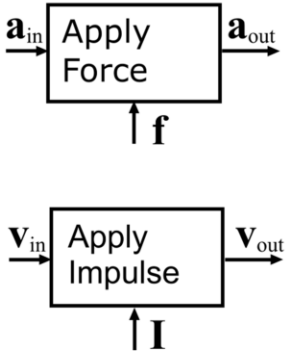
What is an impulse?



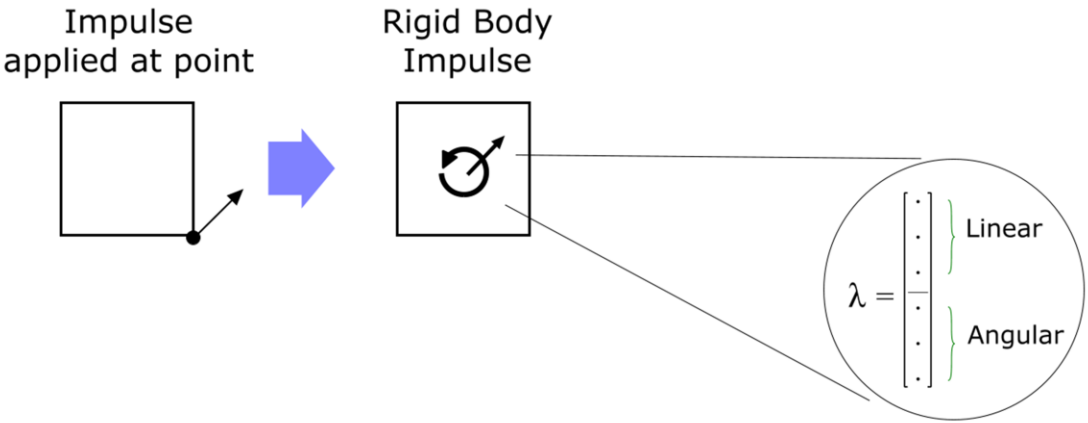
impulse is area under force-time graph

(for constant force, $I = hf$)

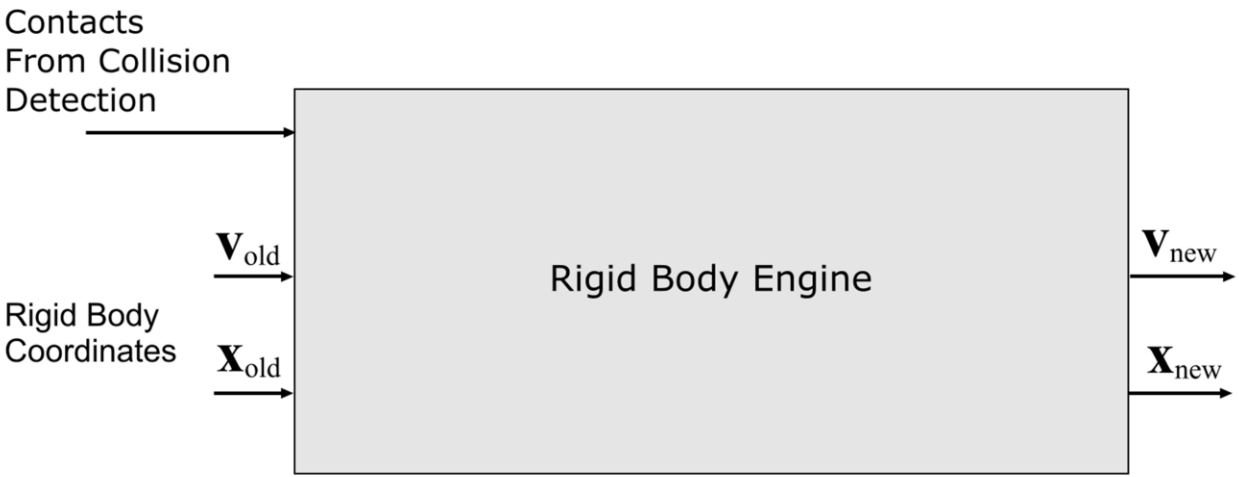
Impulse application



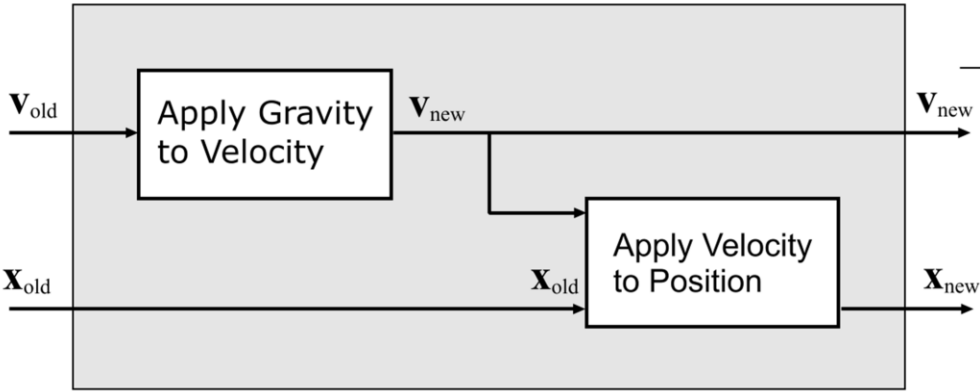
Impulses


Impulse
applied at point
Rigid Body
Impulse
λ =
Linear
Angular

Making a rigid body solver

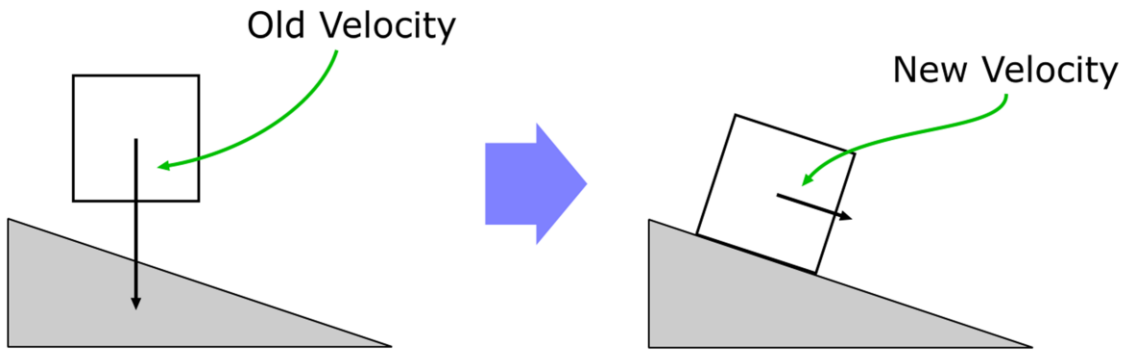


Moving a Body Without Collisions

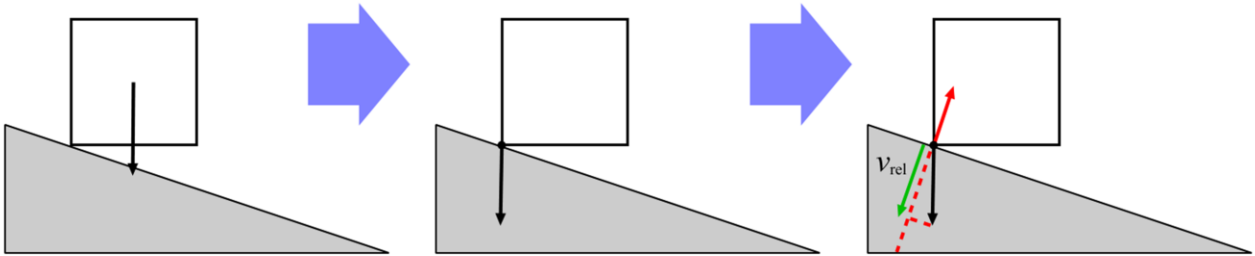


Adding a single contact

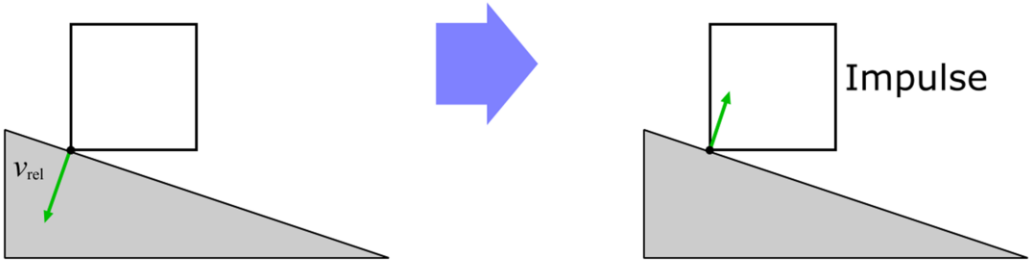
Contact at the velocity level



Velocity at the contact

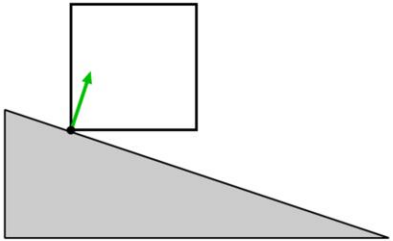


Calculating the impulse

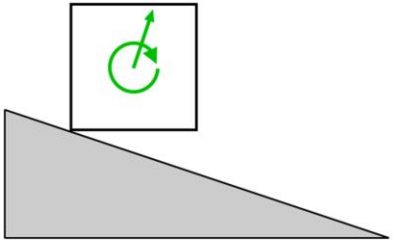


Converting impulse to RB coords

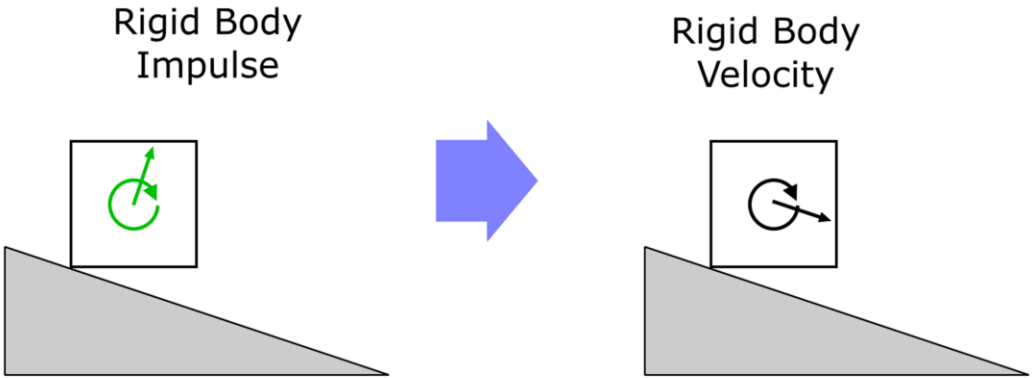
Impulse
At Point



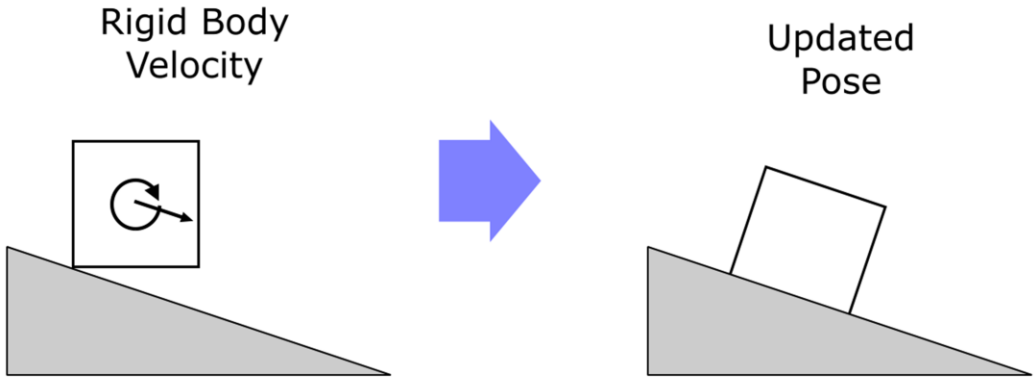
Rigid Body
Impulse



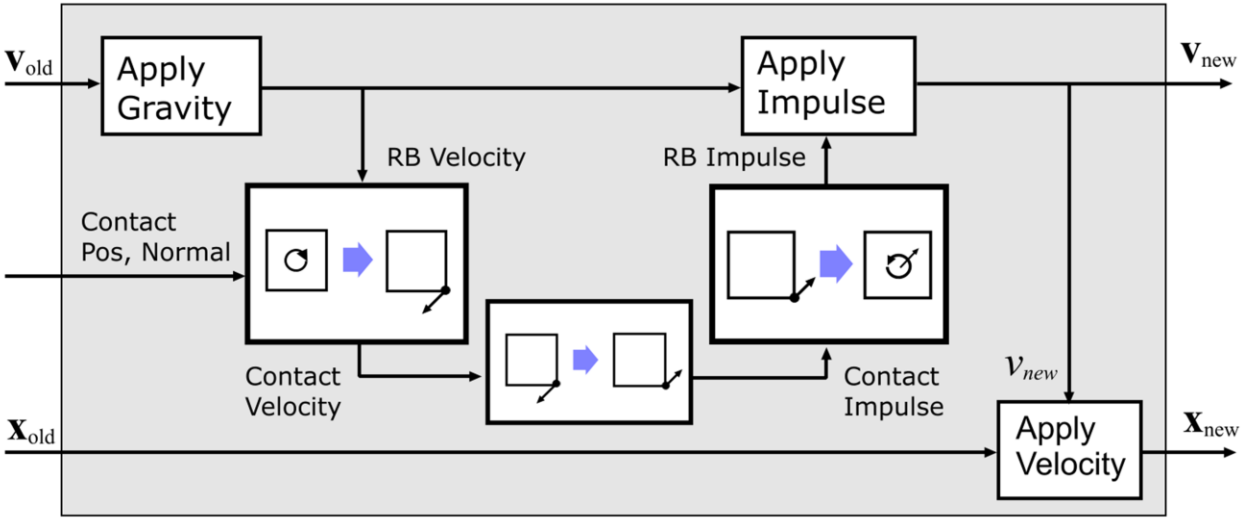
Applying the impulse



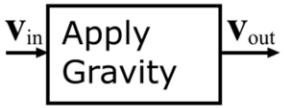
Applying the velocity



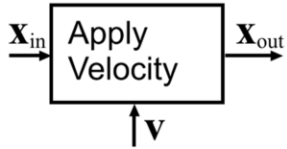
Putting it all together



Summary of Operations 1



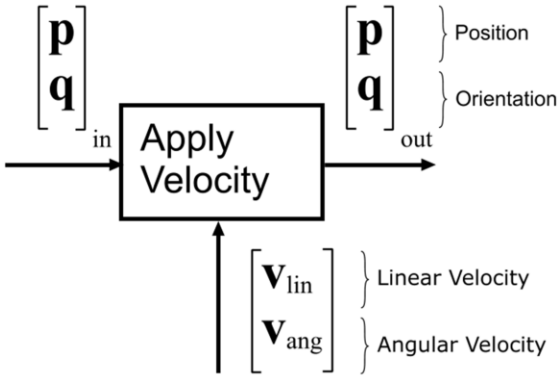
$$\mathbf{v}_{out} = \mathbf{v}_{in} + h\mathbf{g}$$



$$\mathbf{x}_{out} = \mathbf{x}_{in} + h\mathbf{v}$$

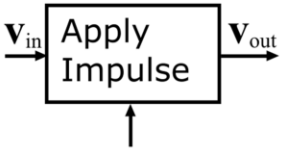
h is the timestep size, 1/60 seconds for 60Hz

Rotation Complicates Things



$$\mathbf{p}_{out} = \mathbf{p}_{in} + h\mathbf{v}_{lin}$$
$$\mathbf{q}_{out} = [\cos(\theta/2), \mathbf{a}\sin(\theta/2)]\mathbf{q}_{in}$$
$$\mathbf{a} = \mathbf{v}_{ang}/|\mathbf{v}_{ang}|$$
$$\theta = |h\mathbf{v}_{ang}|$$

Apply Impulse



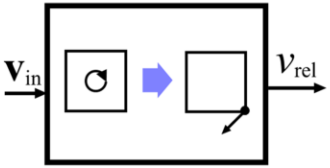
$$\mathbf{v}_{out} = \mathbf{v}_{in} + \mathbf{M}^{-1}\mathbf{I}$$

Rigid Body Impulse, $\mathbf{I}$

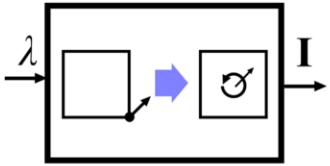
Mass Matrix

$$\mathbf{M} = \begin{bmatrix} m & & & \\ & m & & \\ & & m & \\ & & & \text{Inertia Tensor} \end{bmatrix}$$

Summary of operations 2

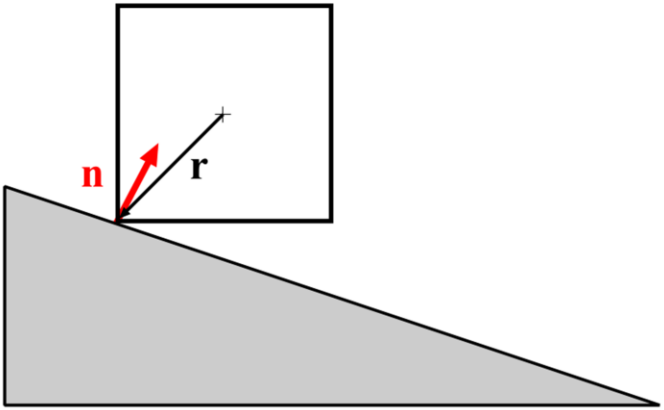


$$v_{\text{rel}} = \mathbf{J}\mathbf{v}_{\text{in}}$$



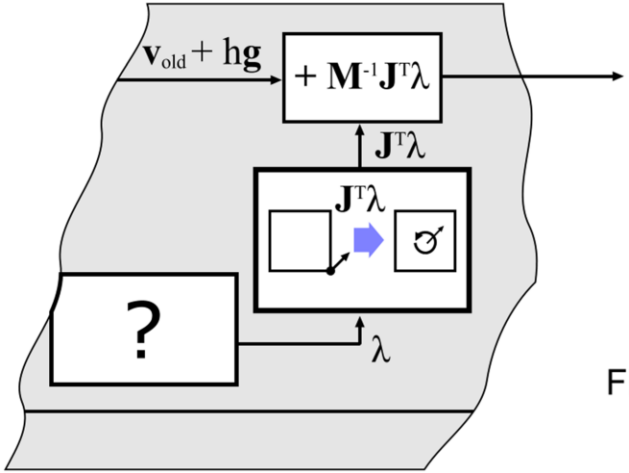
$$\mathbf{I} = \mathbf{J}^T\lambda$$

What is **J**?



$$\mathbf{J} = [\mathbf{n} \mid \mathbf{r} \times \mathbf{n}]$$

Calculating the impulse (λ)



Final velocity in terms of λ

$$\mathbf{v}_{new} = \mathbf{v}_{old} + \mathbf{hg} + \mathbf{M}^{-1}\mathbf{J}^T\lambda$$

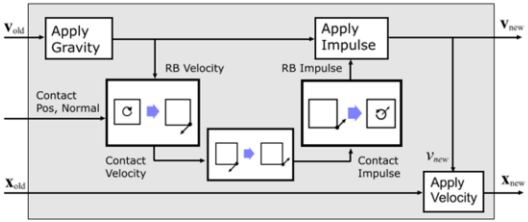
Final relative velocity

$$\mathbf{J}\mathbf{v}_{new} = \mathbf{J}(\mathbf{v}_{old} + \mathbf{hg}) + \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\lambda$$

Final relative velocity is zero when:

$$\lambda = -(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\,\mathbf{J}(\mathbf{v}_{old} + \mathbf{hg})$$

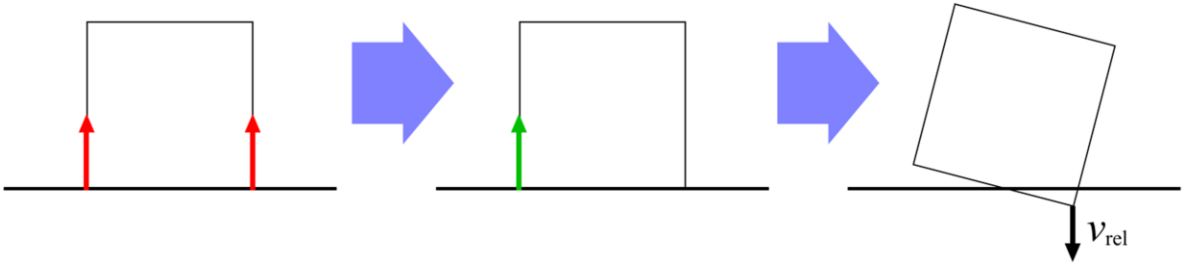
Single Contact Algorithm



=

```
solveSingle(x, v, J, M, h, g)
{
  λ = (-(JM^-1 J^T)^-1 J(v + hg))^+
  v = v + M^-1 J^T λ
  x = x + hv
}
```

Multiple Contacts



Two Contacts

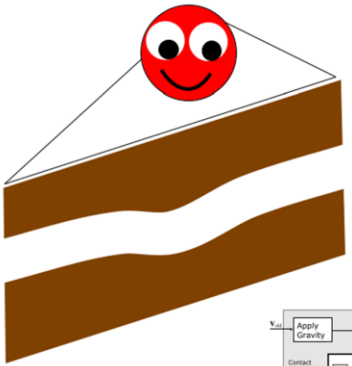
Single Impulse

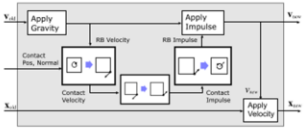
Other Contact Worse

Multiple contacts

$$0 \leq \mathbf{v}_{\text{rel}} \perp 0 \leq \lambda$$

Model first



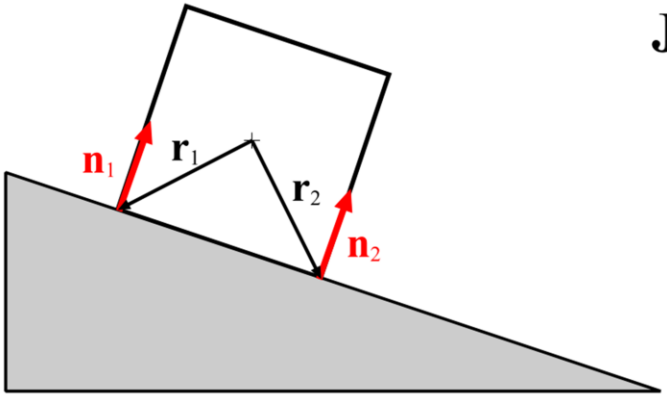


Algorithm second

The value of knowing the model

- Something to test against
- Convergence
- Peace of mind when debugging
- Other solutions for same model

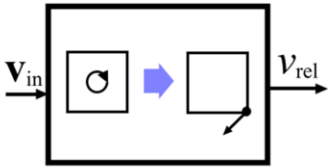
What is **J**?



$$\mathbf{J} = \left[\begin{array}{c|c} \mathbf{n}_1 & \mathbf{r}_1 \times \mathbf{n}_1 \\ \hline \mathbf{n}_2 & \mathbf{r}_2 \times \mathbf{n}_2 \end{array}\right]$$

$$= \left[\begin{array}{ccc|ccc} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array}\right]$$

Operations still work with new **J**



$$\boldsymbol{v}_{\text{rel}} = \mathbf{J}\mathbf{v}_{\text{in}}$$

$$\begin{bmatrix} v_{\text{rel}} \text{ at contact 1} \\ v_{\text{rel}} \text{ at contact 2} \end{bmatrix} = \left[\begin{array}{ccc|ccc} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array}\right] \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \hline \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$\mathbf{J}$ $\mathbf{v}_{\text{in}}$

System is a matrix equation?

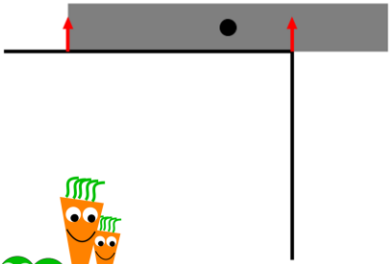
No

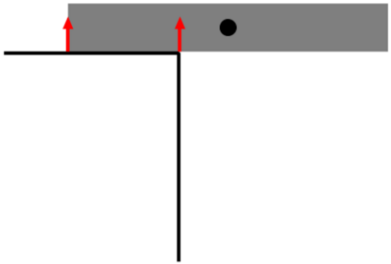
$$\lambda = LCP(A,b)$$

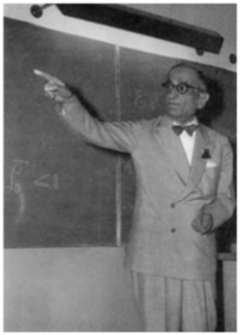
Why?

Contacts can break

When Should Contacts Break?

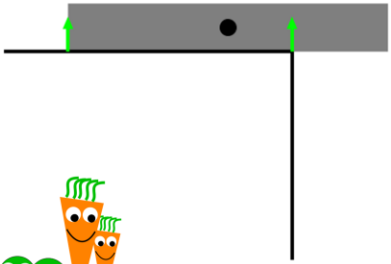


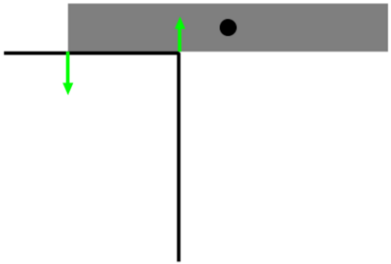




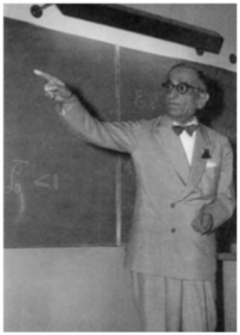
Antonio Signorini

Linear system would eliminate velocity at all contacts





Bar won't fall



Antonio Signorini

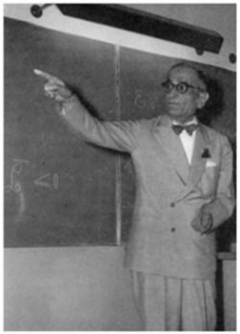
The Signorini Conditions:

$0 \leq \mathbf{v}_{\text{rel}}$ Every relative velocity should be zero or separating

$0 \leq \lambda$ Every contact impulse should be non-attractive

$(\mathbf{v}_{\text{rel}})_i = 0 \quad \text{or} \quad \lambda_i = 0$

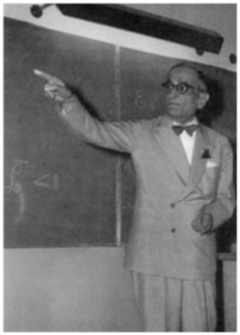
No impulse at separating contacts



Antonio Signorini

A compact way to write the three conditions in one line of math:

$$0 \leq \mathbf{v}_{\text{rel}} \perp 0 \leq \boldsymbol{\lambda}$$



Antonio Signorini

The Final Model

$$\mathbf{M}\ddot{\mathbf{x}} = \mathbf{J}^T\lambda + \mathbf{f}_e$$

$$\dot{\mathbf{x}} = \mathbf{v}$$

$$\mathbf{0} \leq \lambda \perp \mathbf{0} \leq \mathbf{J}\mathbf{v}$$

Discretized model

$$\mathbf{M}(\mathbf{v}_{\text{new}} - \mathbf{v}_{\text{old}}) = \mathbf{J}^T\mathbf{z} + h\mathbf{f}_{\text{e}}$$

$$\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{old}} = h\mathbf{v}_{\text{new}}$$

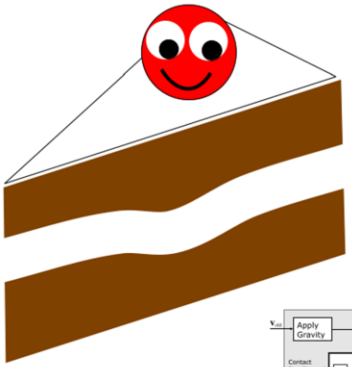
$$\mathbf{z} \geq 0 \perp \mathbf{J}\mathbf{v}_{\text{new}} \geq 0$$

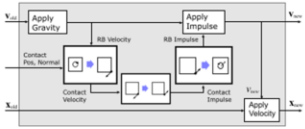
$$\lambda = LCP\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T, \mathbf{J}\left(\mathbf{v}_{\text{old}} + h\mathbf{g}\right)\right)$$

$$\mathbf{v}_{\text{new}} = \mathbf{v}_{\text{old}} + \mathbf{M}^{-1}\mathbf{J}^T\mathbf{z} + h\mathbf{g}$$

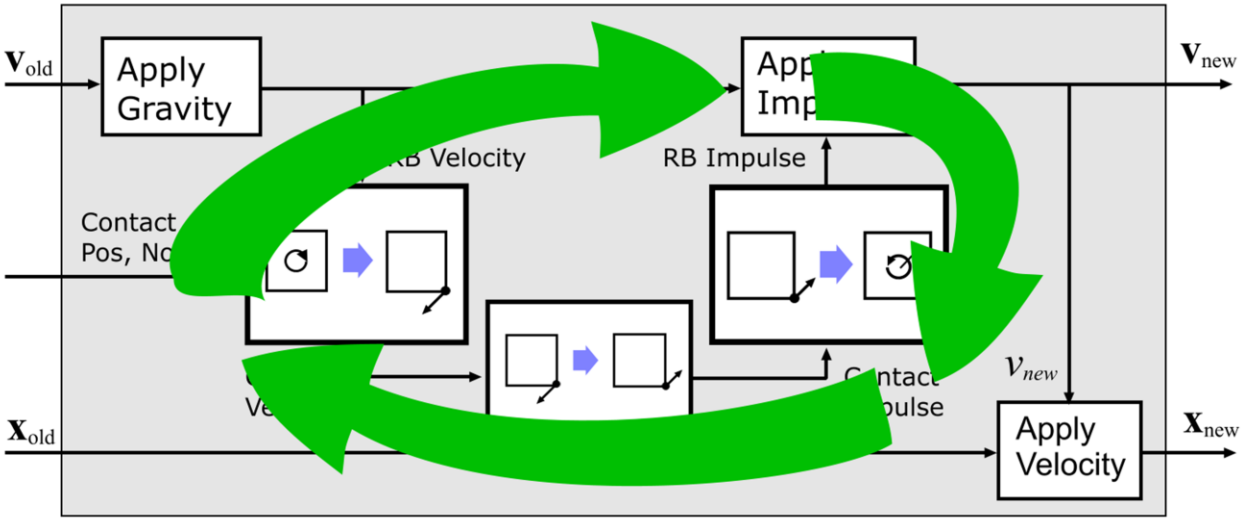
$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} + h\mathbf{v}_{\text{new}}$$

Just Sequentially Apply Impulses





Sequentially Applying Impulses

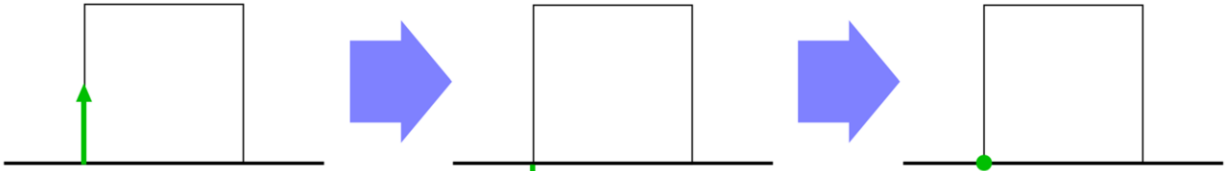


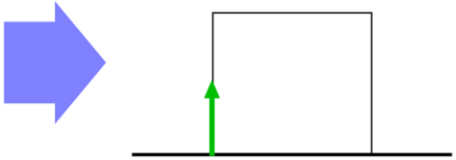
Initial idea to enforce Signorini

At each iteration,

if impulse (λ) is negative, set it to zero

Potential problem



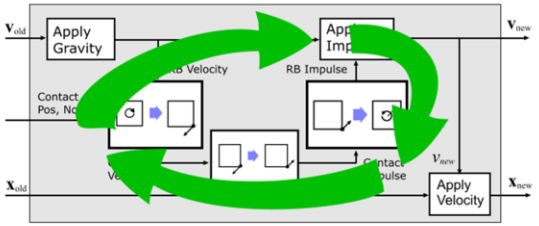


Impulse is always
too large

Solution: Clamp total impulse

- Keep impulse accumulator for each contact
- Clamp the accumulator at iteration, not the added impulse

Multiple Contact Algorithm



=

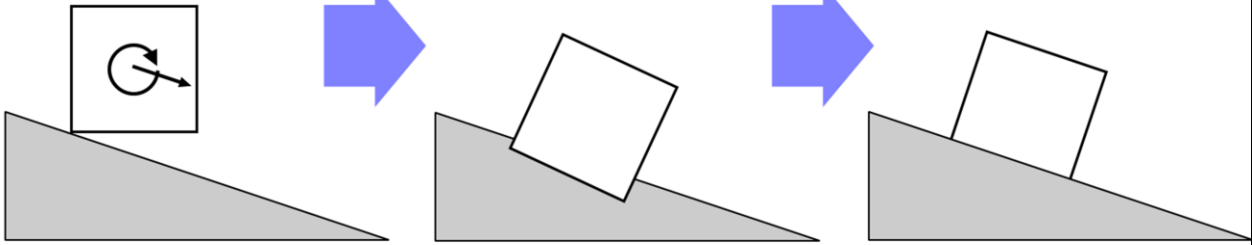
```
solveMultiple(x, v, J, M, h, g)
{
  for j=0; j<iterCnt; j++
  {
    for i=0; i<contactCnt; i++
    {
      t = λi
      m = Ji M^-1 Ji^T
      λi = λi -(1/m) (Jv + b)
      λi = max(0, λi)
      v = v + M^-1 J^T (λi - t)
    }
  }
}
x = x + hv
```

Position projection

Rigid Body Velocity

Fixed timestep, moved into penetration

After position projection



Position Projection

Let ϕ be the penetration.

Instead of requiring: $\mathbf{Jv} \geq \mathbf{0}$

Require: $\mathbf{Jv} + (\gamma/h)\phi \geq \mathbf{0}$

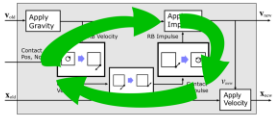
In previous algorithm, set $\mathbf{b} = (\gamma/h)\,\phi$

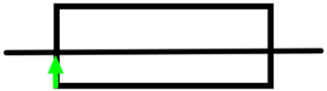
(I use $\gamma = 0.8$)

Section 3

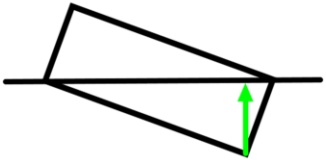
How to make rigid bodies do the right thing using a solver (multicore)

Existing solver method 1:



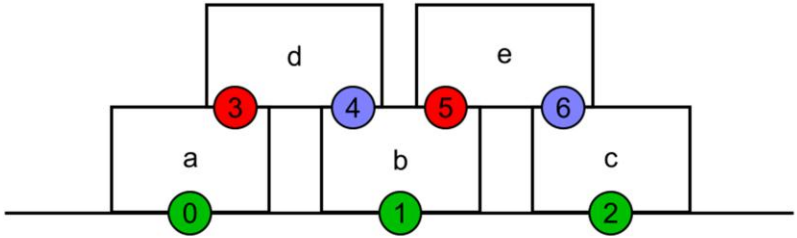


Penetrating configuration

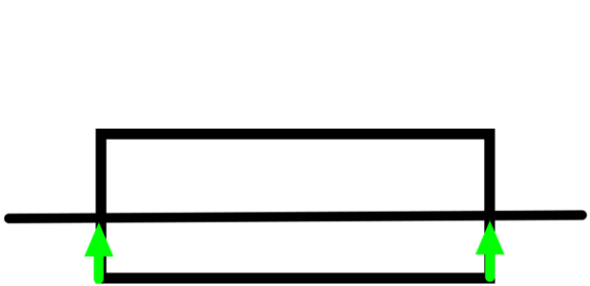


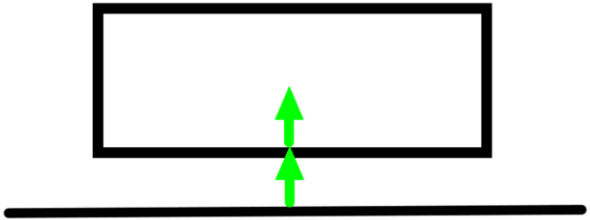
Rendered Frame

PGS coloring



Existing solver method 2:

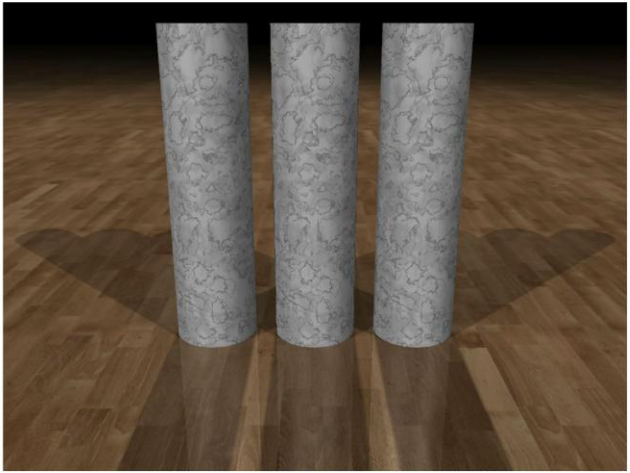




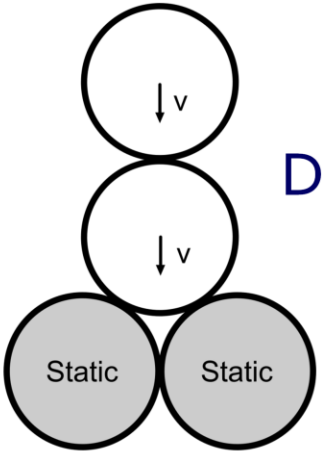
- Method 1 (Projected Gauss Seidel, PGS)
  - Provably convergent ✔
  - Limited parallelism ✘
  - Jitters ✘
  - Widely used

- Method 2 (Projected Jacobi)
  - Maximally parallel ✔
  - Jitter free ✔
  - Non convergent in many cases ✘
  - Converges slowly ✘
  - Unusable in games

Parallel PGS Jitter

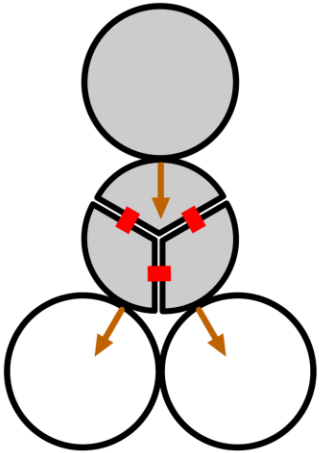


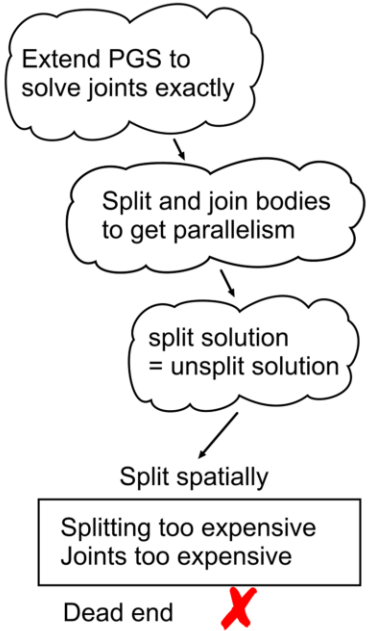
Example



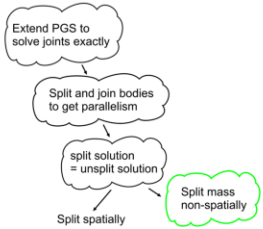
Doesn’t converge with Jacobi

First idea: Spatial splitting

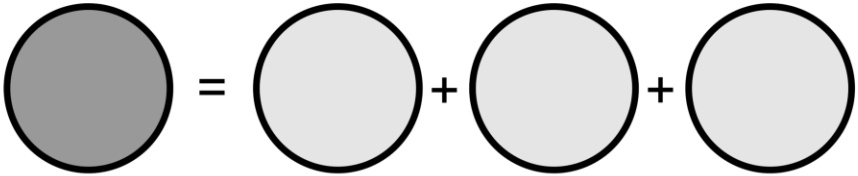



Extend PGS to solve joints exactly
Split and join bodies to get parallelism
split solution = unsplit solution
Split spatially
Splitting too expensive
Joints too expensive
Dead end

New idea: Mass splitting

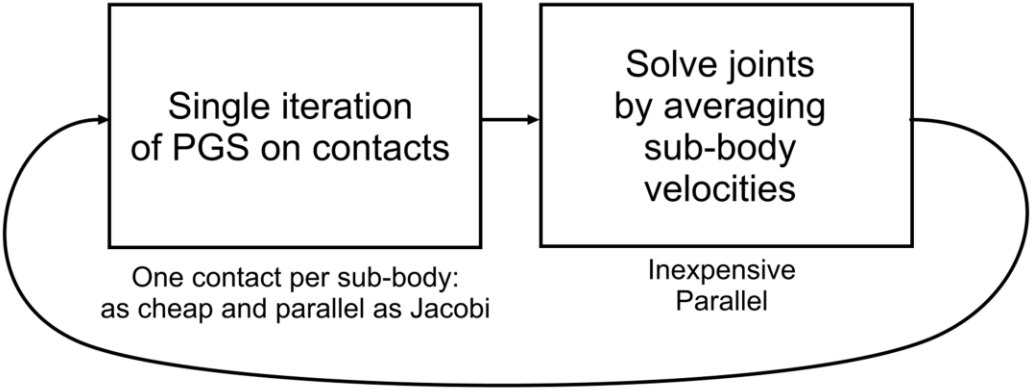


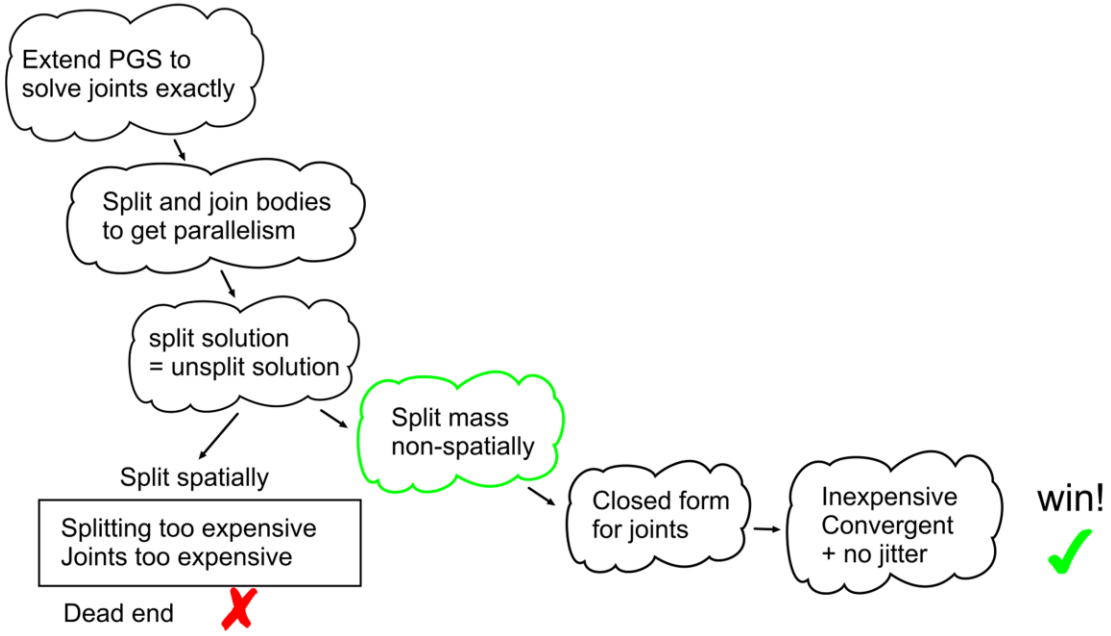
Make sub-bodies have the same spatial extent



+ two fixed joints at C.O.M.

PGS with exact joints

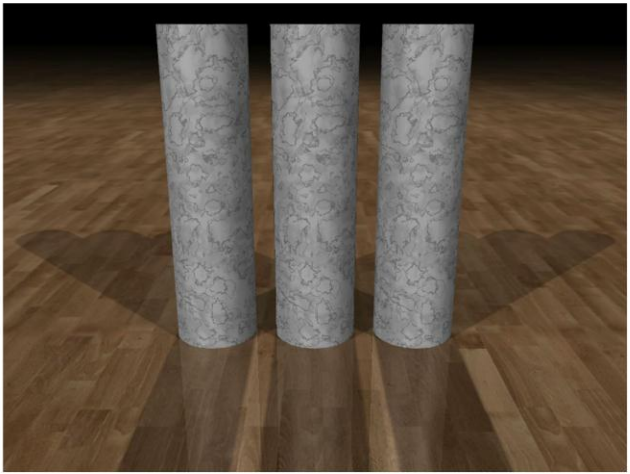



Extend PGS to solve joints exactly
Split and join bodies to get parallelism
split solution = unsplit solution
Split spatially
Splitting too expensive
Joints too expensive
Dead end
Split mass non-spatially
Closed form for joints
Inexpensive
Convergent
+ no jitter
win!

Section 4

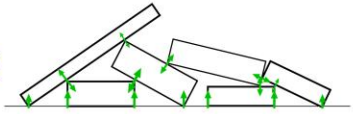
Results

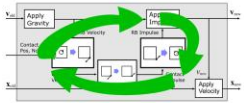
Mass splitting



Summary

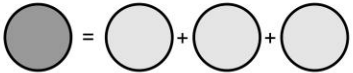
What we need rigid bodies to do

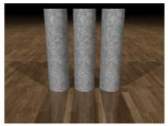




Single core solver

Parallel solver





Jitter-free

Acknowledgments

GPU rigid body technology
- Feodor Benevolenski
- Andrey Voroshilov
- Richard Tonge

Fracture technology and demo
- Matthias Müller-Fischer
- Nuttapong Chentanez
- Tae-Yong Kim
- Aron Zoellner

Thanks also to the PhysX SDK team