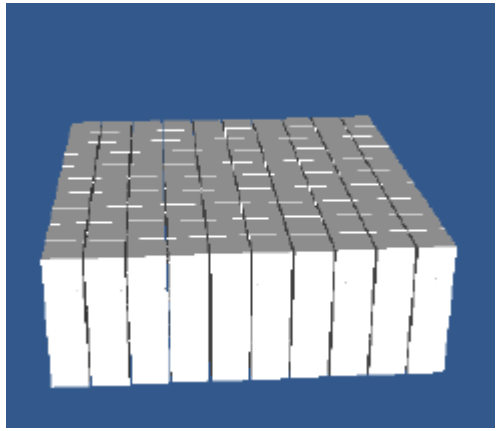
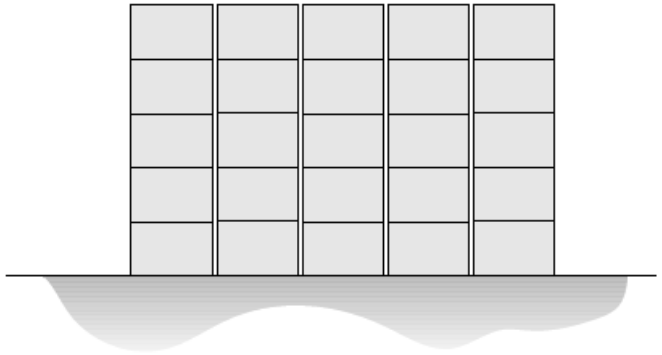
Velocity-Based Shock Propagation for Multibody Dynamics Animation

Presented by Patrick O' Halloran

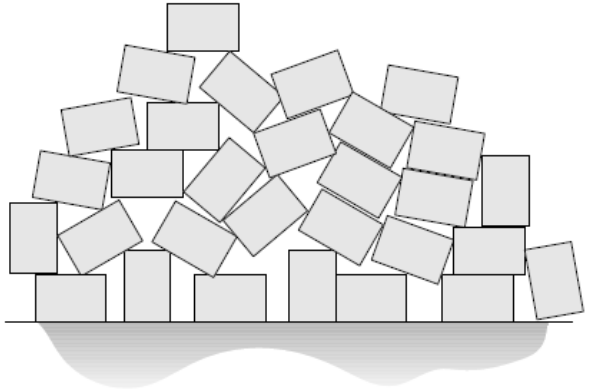
Intro

- Explicit time-stepping scheme
- Velocity based complementarity formulations
- Constraint formulations are solved by an LCP solver.
- Optimised for multibody dynamics
- Shock Propagation

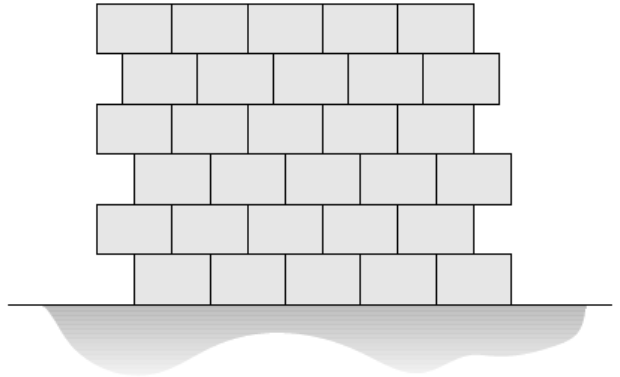




Sparse Stacking

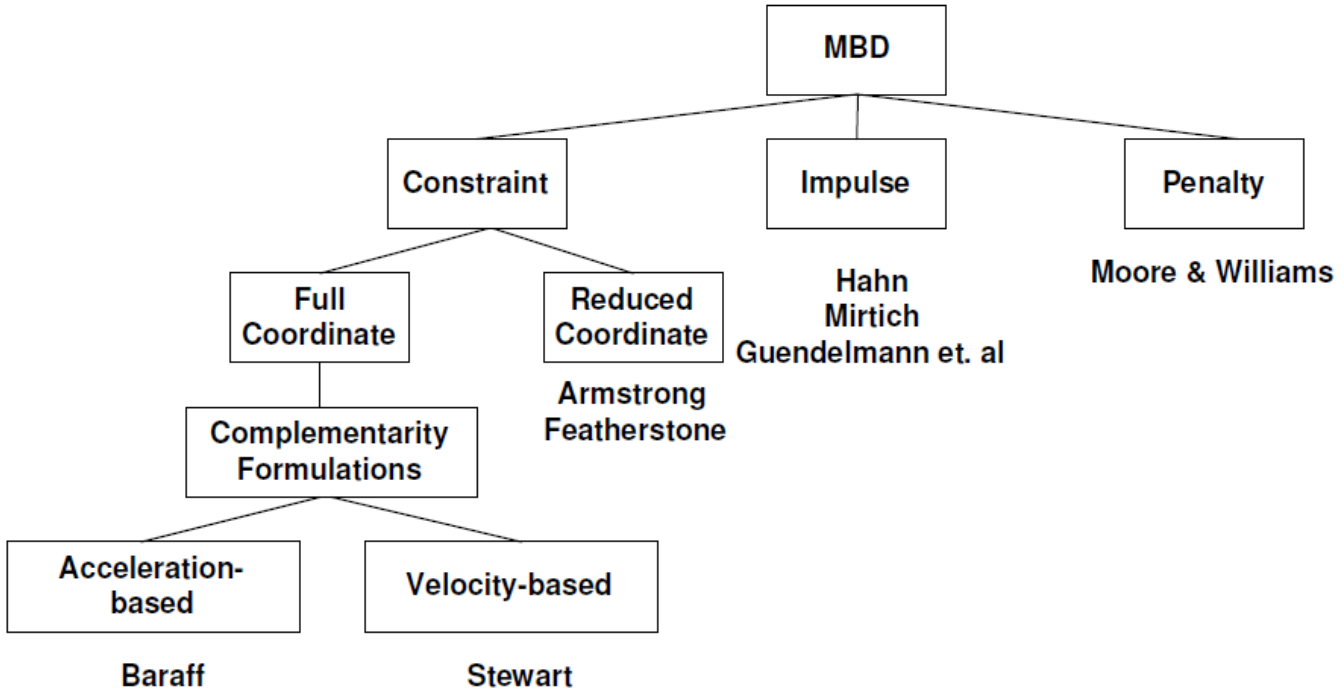


Dense random stacking

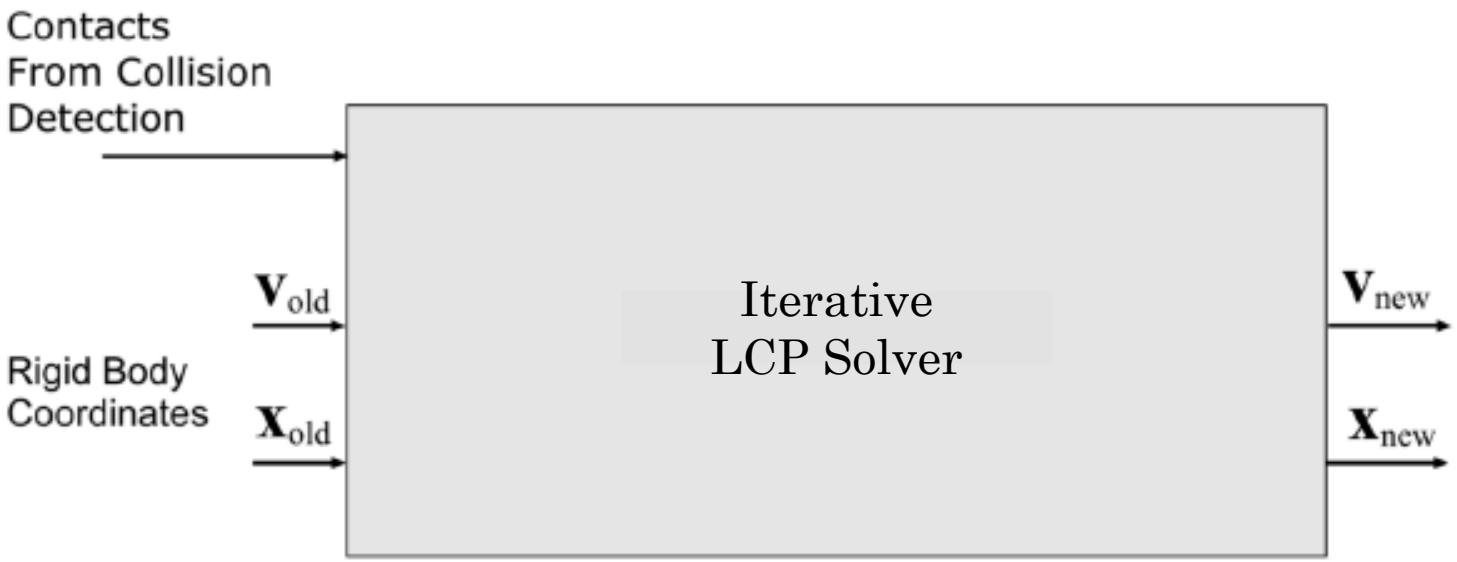


Dense structured stacking

Background


MBD
Constraint
Impulse
Penalty
Full Coordinate
Reduced Coordinate
Hahn
Mirtich
Guendelmann et. al
Moore & Williams
Armstrong
Featherstone
Complementarity Formulations
Acceleration-based
Velocity-based
Baraff
Stewart

System overview



Constraints

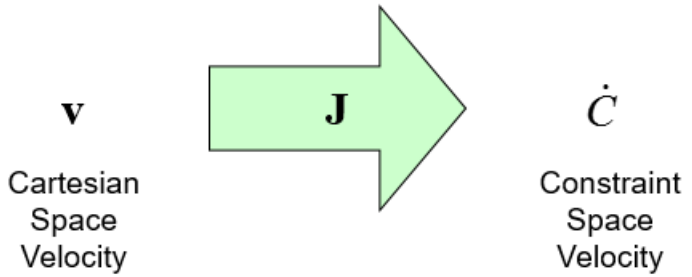
Advantages

- Penetration is handled
- Joints and contacts are handled the same way

Velocity Constraints

- Derivative of position constraint w.r.t. time
- Define the allowed motion
- Satisfied by applying impulses
- Typically involves two bodies
- Are linear (The row vector J depends on position)

$$\dot{C} = \mathbf{J}\mathbf{v} = 0$$

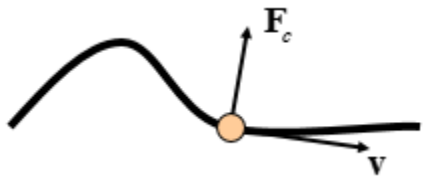


$$\vec{u} = [\vec{v}_1, \vec{\omega}_1, \vec{v}_2, \vec{\omega}_2, \cdots, \vec{v}_N, \vec{\omega}_N]^T .$$

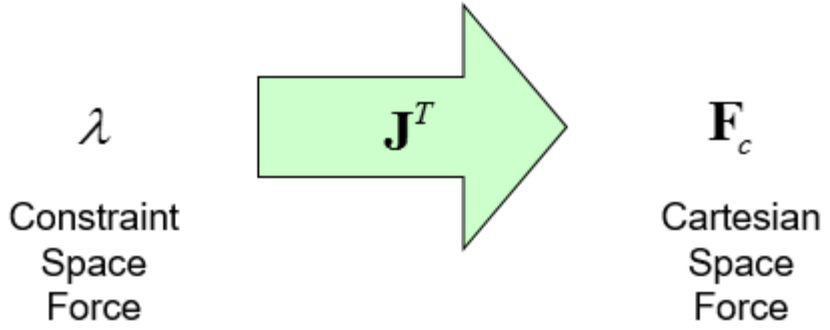
$$J_k = \begin{bmatrix} \mathbf{J}^i_{\text{lin}_k} & \mathbf{J}^i_{\text{ang}_k} & \mathbf{J}^j_{\text{lin}_k} & \mathbf{J}^j_{\text{ang}_k} \end{bmatrix}$$

Constraint force

- Constraint forces do no work
  - Therefore constraint force is orthogonal to the constraint (the allowed velocity).



$$\mathbf{F}_c = \mathbf{J}^T \lambda$$



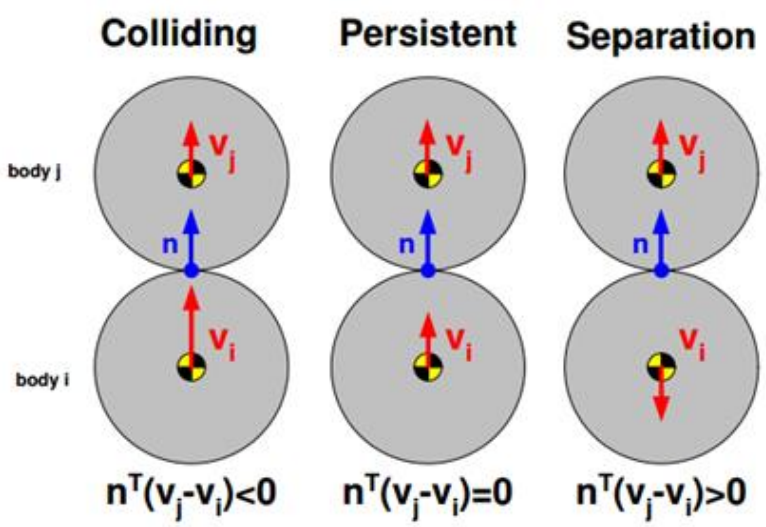
$$P_c = \mathbf{F}_c^T \mathbf{v} = \left(\mathbf{J}^T \lambda\right)^T \mathbf{v} = \lambda \mathbf{J} \mathbf{v} = 0$$

Lagrange Multiplier (λ)

$$\vec{f}_1 = \mathbf{J}_{\text{row}_1}^T \lambda_1,$$

- The Lagrange multiplier is the constraint force magnitude
- Finding the right lambda is the job of the LCP solver.

Formulating Constraints



$$\vec{n} \cdot (\vec{v}_j - \vec{v}_i) \geq 0$$

$$\vec{n} \cdot (\vec{v}_j + \vec{\omega}_j \times \vec{r}_j - (\vec{v}_i + \vec{\omega}_i \times \vec{r}_i)) \geq 0$$

$$\underbrace{\begin{bmatrix} -\vec{n}^{\,t} & -\left(\tilde{\mathbf{r}}_\mathbf{i}^{\times}\vec{n}\right)^T & \vec{n}^t & \left(\tilde{\mathbf{r}}_\mathbf{j}^{\times}\vec{n}\right)^T \end{bmatrix}}_{J_{\text{row}_1}} \underbrace{\begin{bmatrix} \vec{v}_i \\ \vec{\omega}_i \\ \vec{v}_j \\ \vec{\omega}_j \end{bmatrix}}_{\vec{u}} = J_{\text{row}_1}\vec{u} \geq 0.$$

Merging Constraints

$$J_1 V + b_1 = 0$$
$$J_2 V + b_2 = 0$$
$$J_3 V + b_3 = 0$$

$$JV + b = 0$$

Types: Contact, Friction, Ragdolls, Particles / Cloth, Distance, Joints, Motors

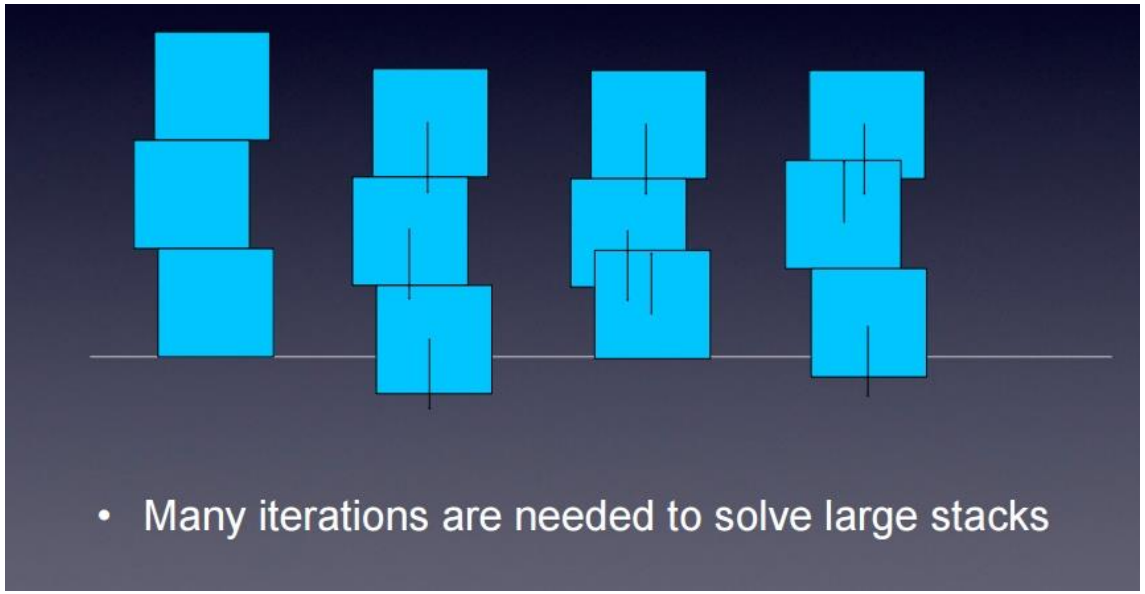
Solving Constraints

- Solving a single constraint is easy.
- Solving globally is hard (inefficient)
- Alternative: solve constraints iteratively
  - Solve all constraints, one at a time, multiple times
  - Solutions will invalidate each other
  - But over many iterations each constraint will converge and a global solution achieved.
- Once a solution is achieved, an impulse can be applied to both bodies in the constraint in order to enforce the constraint
- Spare Matrix Representation used
- Can be solved in linear time wrt. the number of constraints
- Constraint Drift

$$\lambda = \frac{-(JV + b)}{JM^{-1}J^T}$$

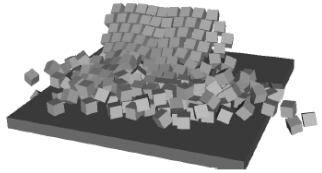
$$M^{-1} = \begin{bmatrix} m_1^{-1} & 0 & 0 & 0 \\ 0 & I_1^{-1} & 0 & 0 \\ 0 & 0 & m_2^{-1} & 0 \\ 0 & 0 & 0 & I_2^{-1} \end{bmatrix}$$

Convergence

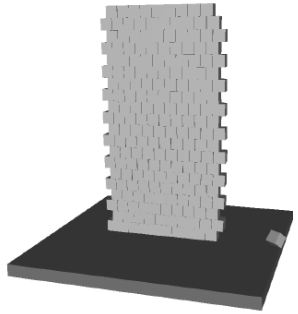


Convergence

Convergence results (200 brick wall) of iterative LCP solver

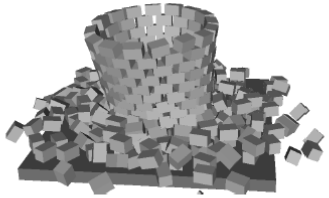


10 iterations after 4 secs.

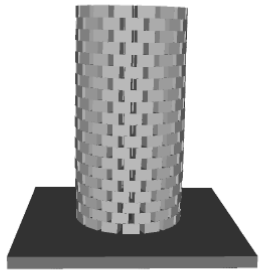


100 iterations after 4 secs.

Convergence results (320 brick tower) of iterative LCP solver



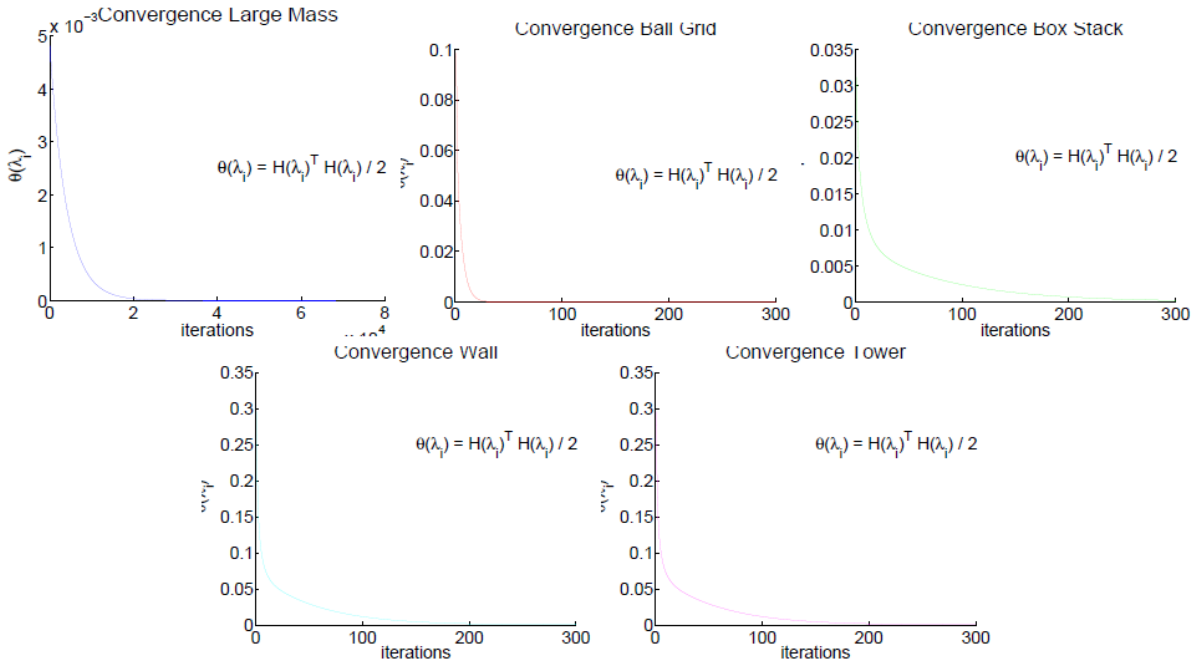
10 iterations after 4 secs.



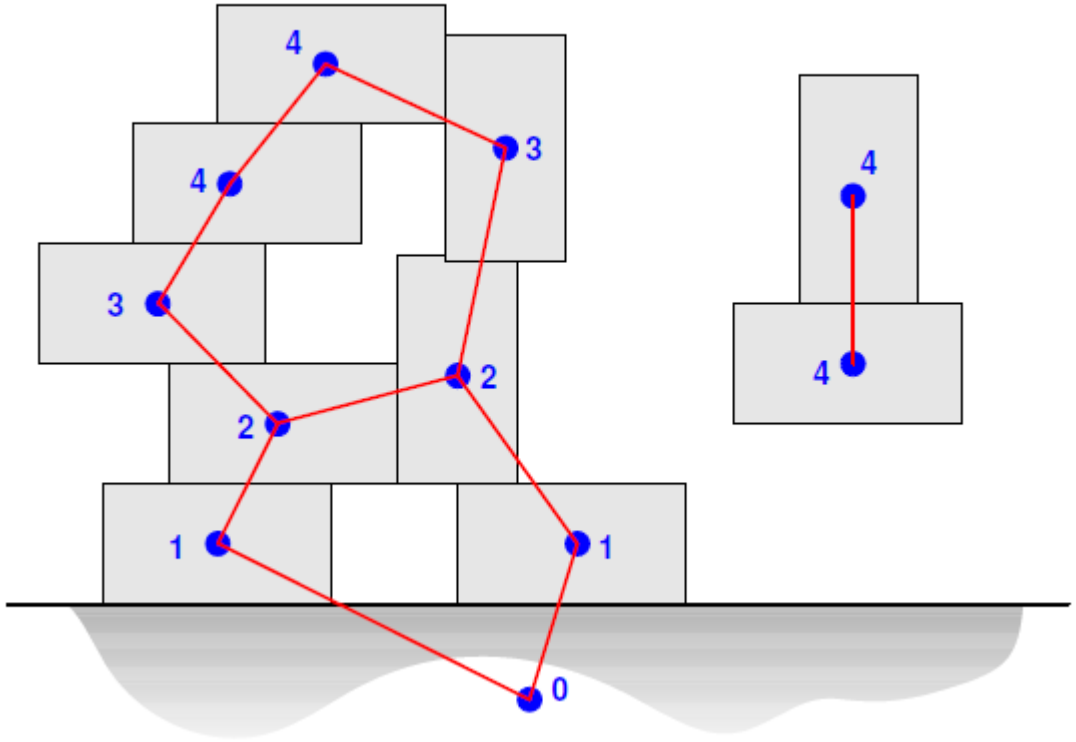
100 iterations after 4 secs.

Rule of thumb

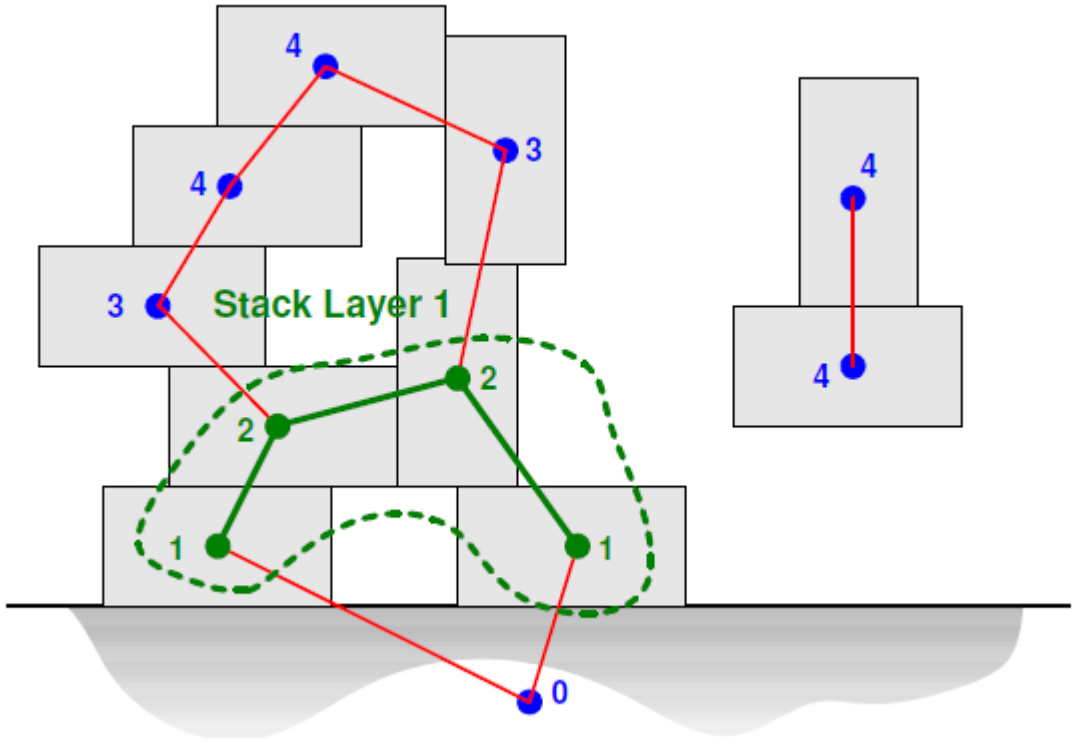
- Convergence is related to the structure of the stack
- The number of iterations needed is O(kn^2)
  - Where k is some constant
  - n is the number of constraints on the 'longest' path



Contact Graph

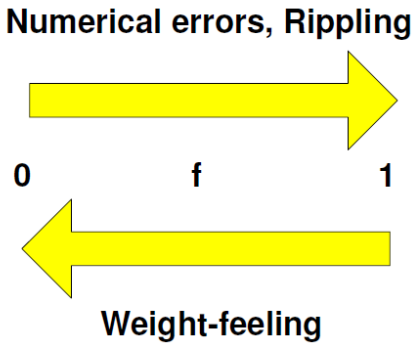


Stack Layers

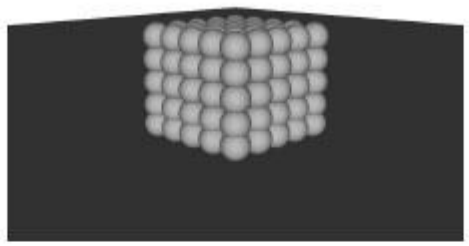



```
simple-scheme(f,dt)
  collision detection at time t
  dynamics(f*dt)
  shock-propagration(
    dynamics((1-f)*dt), correction()
  )
  t = t + dt
```

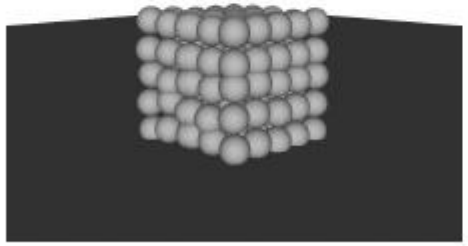
```
shock-propagation(algorithm A)
  compute contact graph
  for each stack layer in bottom up order
    fixate bottom-most objects of layer
    apply algorithm A to layer
    un-fixate bottom-most objects of layer
  next layer
```



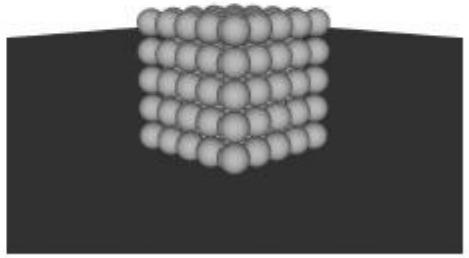
Convergence



1’st frame



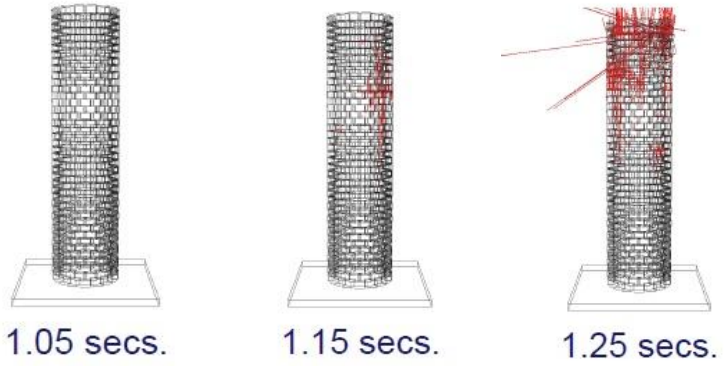
6’th frame w/o. shock-propagation

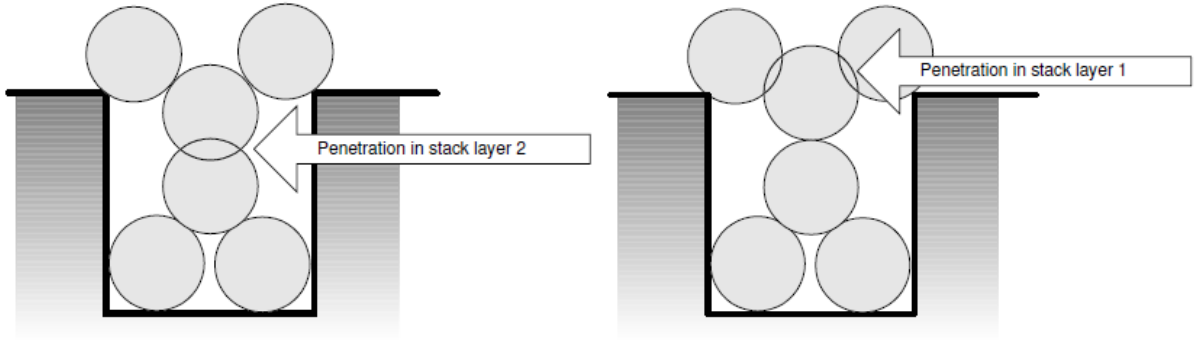


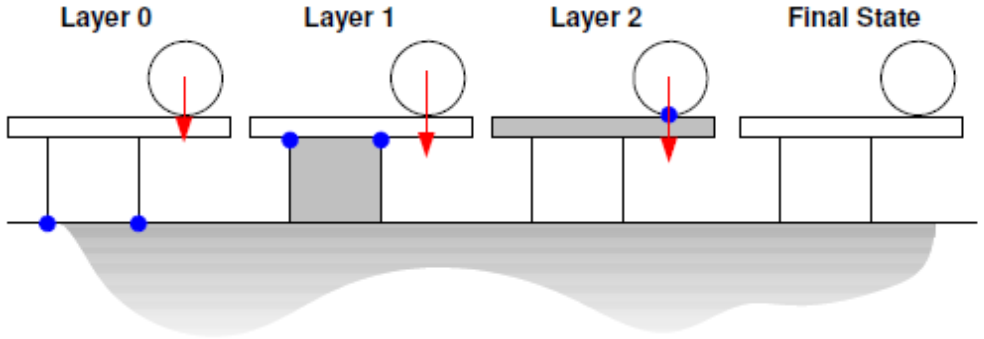
6’th w. shock-propagation

Still some problems..

- Rippling due to numerical errors
- Cyclical dependencies
- ‘No Weight feeling’ problem







Fix: Add a second correction phase.

Final algorithm

```
velocity-based-shock-propagation(f,dt)
  collision detection at time t
  dynamics(f*dt)
  shock-propagration(
    dynamics((1-f)*dt), correction()
  )
  collision detection at time t + dt
  correction()
  t = t + dt
```

where $0 \leq f \leq 1$ is a weighting value.

Things working