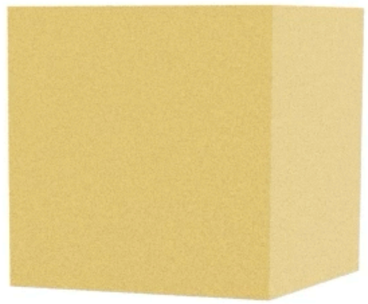
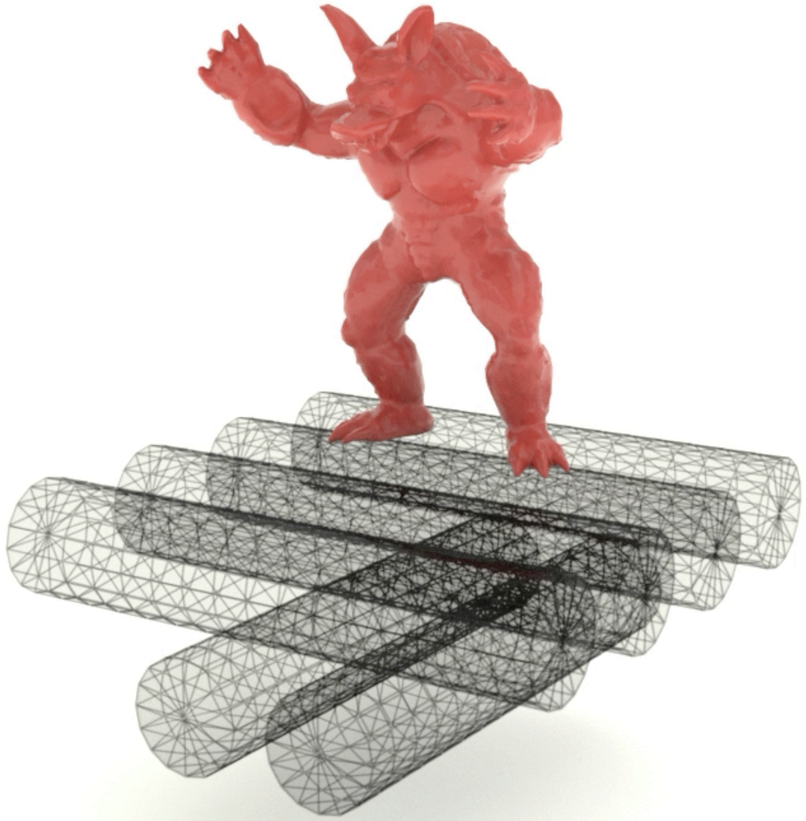
Instructor: Minchen Li

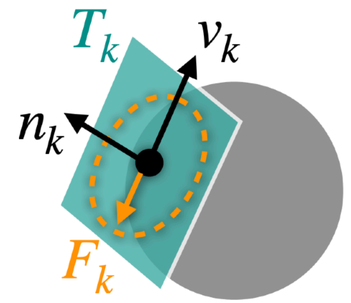




Lec 7: Restitution, Moving Boundary Conditions

15-769: Physically-based Animation of Solids and Fluids (F23)

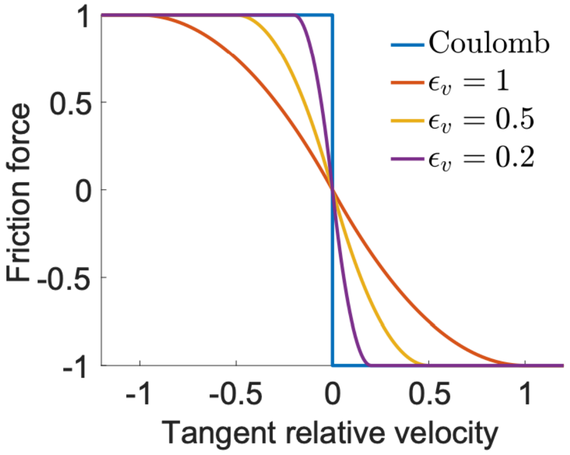
Recap: Frictional Contact



$$F_k(x) = T_k(x) \underset{\boldsymbol{\beta} \in \mathbb{R}^d}{\operatorname{argmin}} \boldsymbol{\beta}^T \mathbf{v}_k \quad \text{s.t.} \quad \|\boldsymbol{\beta}\| \leq \mu \lambda_k \quad \text{and} \quad \boldsymbol{\beta} \cdot \mathbf{n}_k = 0$$

$$F_k(x) = -\mu \lambda_k T_k(x) f(\|\mathbf{v}_k\|) \mathbf{s}(\mathbf{v}_k)$$

Non-smooth!



- Smooth dynamic-static transition
- Semi-implicit discretization $F_k(x) \approx -\mu \lambda_k^n T_k^n f_1(\|\bar{\mathbf{v}}_k\|) \mathbf{s}(\bar{\mathbf{v}}_k)$
- Fixed-point iteration $P_f(x) = \sum_k \mu \lambda_k^n f_0(\|\bar{\mathbf{v}}_k \hat{h}\|)$

Algorithm: alternate between

$$\min_x : E(x, \{\lambda, T\}) = \frac{1}{2}\|x - \tilde{x}^n\|_M^2 + \Delta t^2 \left(P_e(x) + P_b(x) + P_f(x, \{\lambda, T\})\right)$$

$$\text{s.t.} \quad Ax = b,$$

and friction update until convergence

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, Danny M. Kaufman. Incremental Potential Contact: Intersection- and Inversion-free, Large-Deformation Dynamics. ACM ToG (SIGGRAPH), 2020.

More on Traditional Methods for Frictional Contact

Active-Set Methods

$$\min_{x,\, f_n,\, f_t} \frac{1}{2}\|x - (x^n + hv^n)\|_M^2 + h^2 \sum P(x)$$

$$\textbf{s.t.} \quad \forall k, \quad d_k\|f_{n,k}\| = 0, \quad d_k \geq 0, \quad f_{n,k} \cdot n_k \geq 0, \quad (I - n_k n_k^T) f_{n,k} = 0 \qquad \textbf{(normal contact)}$$

$$\|v_k\| \left\| f_{t,k} + \mu \|f_{n,k}\| \frac{v_k}{\|v_k\|} \right\| = 0, \quad \|f_{t,k}\| \leq \mu \|f_{n,k}\|, \quad f_{t,k} \cdot n_k = 0 \qquad \textbf{(friction)}$$

In each iteration:

identify active contact pairs k;

form a KKT system (with approximations) and solve for x, f_n, f_t;

System size grows with k;

No guarantees on

- **Convergence,**
- **Nonpenetration.**

More on Traditional Methods for Frictional Contact

Impulse Methods

For each time step n:

$$x^* = \arg\min_x \frac{1}{2}\|x - (x^n + hv^n)\|_M^2 + h^2 \sum P(x)$$

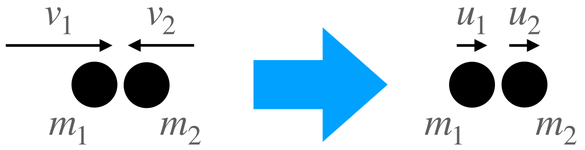
$$v^* \leftarrow (x^* - x^n)/h$$

$$v^{n+1} \leftarrow \text{Resolve collisions and add friction by modifying } v^*$$

$$x^{n+1} \leftarrow x^n + hv^{n+1}$$

Iterate over each pair of contact

e.g. normal contact:



$$u_1 = \frac{m_1 v_1 + m_2 v_2}{m_1 + m_2}$$

$$u_2 = \frac{m_1 v_1 + m_2 v_2}{m_1 + m_2}$$

Fast to compute

Can converge slowly

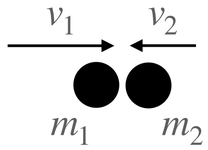
Results depending on the order

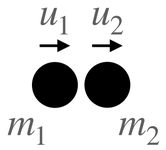
Challenging with simultaneous contact

More on Traditional Methods for Frictional Contact

Coefficient of Restitution

e.g. normal contact:





$$u_1 = \frac{m_1 v_1 + m_2 v_2}{m_1 + m_2} \qquad u_2 = \frac{m_1 v_1 + m_2 v_2}{m_1 + m_2}$$

By solving $\begin{cases} m_1 u_1 + m_2 u_2 = m_1 v_1 + m_2 v_2 \\ \dfrac{|u_1 - u_2|}{|v_1 - v_2|} = 0 \end{cases}$

$$\begin{cases} m_1 u_1 + m_2 u_2 = m_1 v_1 + m_2 v_2 \\ \dfrac{|u_1 - u_2|}{|v_1 - v_2|} = C_R \end{cases} \Longrightarrow \begin{cases} u_1 = \dfrac{m_1 v_1 + m_2 v_2 + m_2 C_R (v_2 - v_1)}{m_1 + m_2} \\ u_2 = \dfrac{m_1 v_1 + m_2 v_2 + m_1 C_R (v_1 - v_2)}{m_1 + m_2} \end{cases}$$

When Coefficient of Restitution $C_R = 1$, energy is conserved after collision

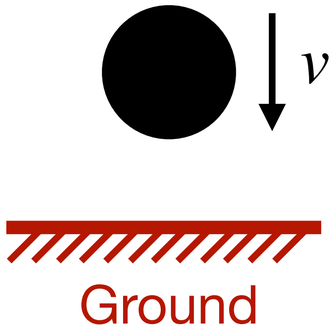
Velocity-based contact formulation offers direct control on C_R

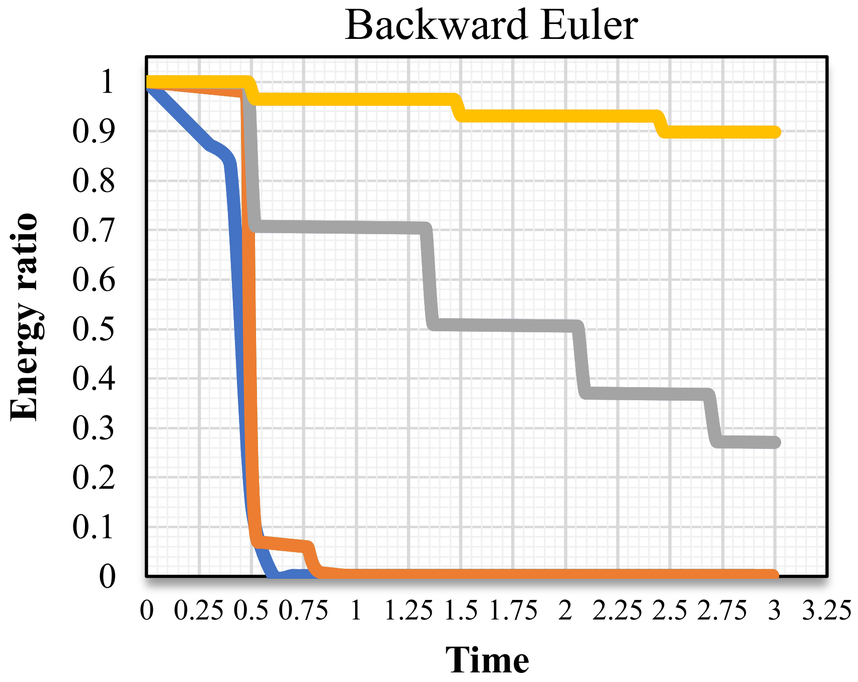
More on Frictional Contact

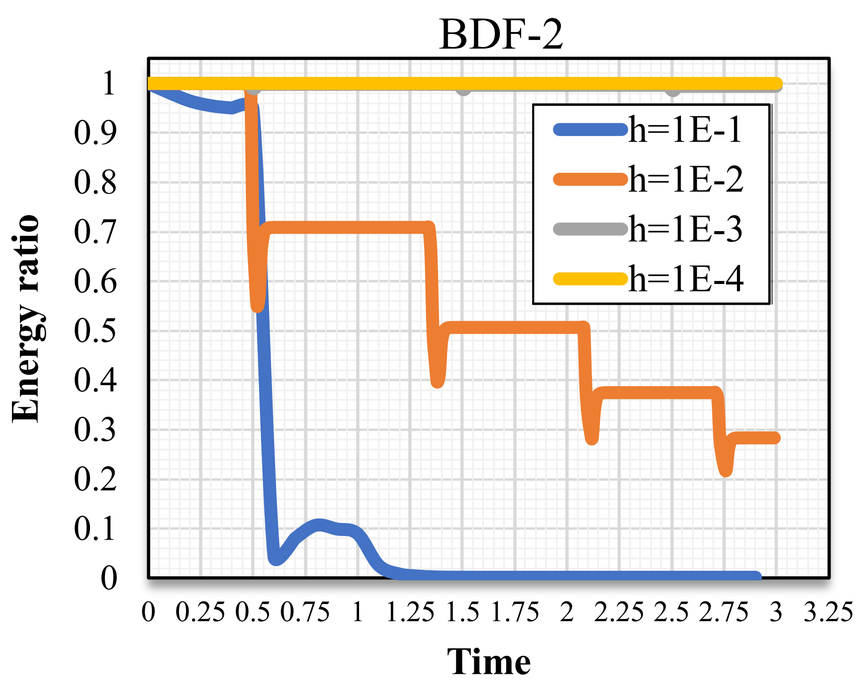
Approximating normal contact as a conservative force with potential energy $P_b(x)$,

Restitution ⇔ Energy Conservation of Time Stepping

e.g.







Use high-order rules, e.g. BDF-2: $\min_{x^{n+1}} \frac{1}{2}\|x^{n+1} - \tilde{x}^n_{BDF2}\|^2_M + \frac{4}{9}h^2 \sum P(x^{n+1})$

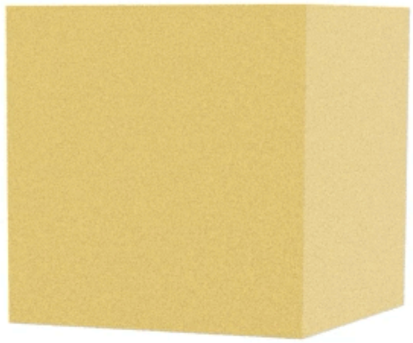
BDF-2 for $y' = f(t, y)$:

$y_{n+2} - \frac{4}{3}y_{n+1} + \frac{1}{3}y_n = \frac{2}{3}hf(t_{n+2}, y_{n+2})$

Yunuo Chen*, Minchen Li*, Lei Lan, Hao Su, Yin Yang, Chenfanfu Jiang. A Unified Newton Barrier Method for Multibody Dynamics. ACM ToG (SIGGRAPH), 2022.

More on Frictional Contact

Restitution ⇔ Energy Conservation of Time Stepping



Use high-order rules, e.g. BDF-2: $\min_{x^{n+1}} \frac{1}{2}\|x^{n+1} - \tilde{x}^n_{BDF2}\|^2_M + \frac{4}{9}h^2 \sum P(x^{n+1})$

Yunuo Chen*, Minchen Li*, Lei Lan, Hao Su, Yin Yang, Chenfanfu Jiang. A Unified Newton Barrier Method for Multibody Dynamics. ACM ToG (SIGGRAPH), 2022.

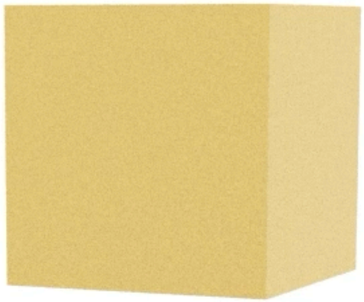
More on Frictional Contact

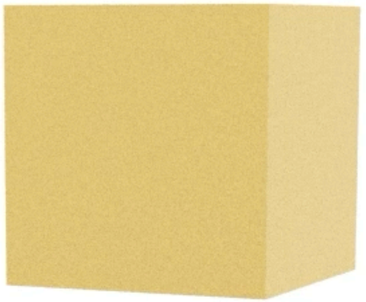
Restitution ⇔ Energy Conservation of Time Stepping

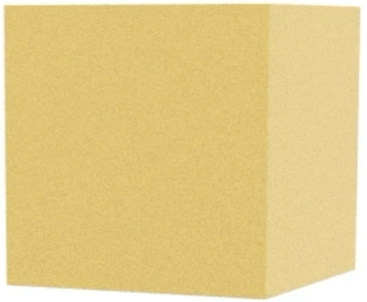
Use Damping energy to control restitution:

$$\frac{k_d}{2} v^T \frac{\partial^2 P_b}{\partial x^2}(x^n) v$$

$$\min_{x^{n+1}} \frac{1}{2} \|x^{n+1} - \tilde{x}^n_{BDF2}\|^2_M + \frac{4}{9} h^2 \sum P(x^{n+1})$$







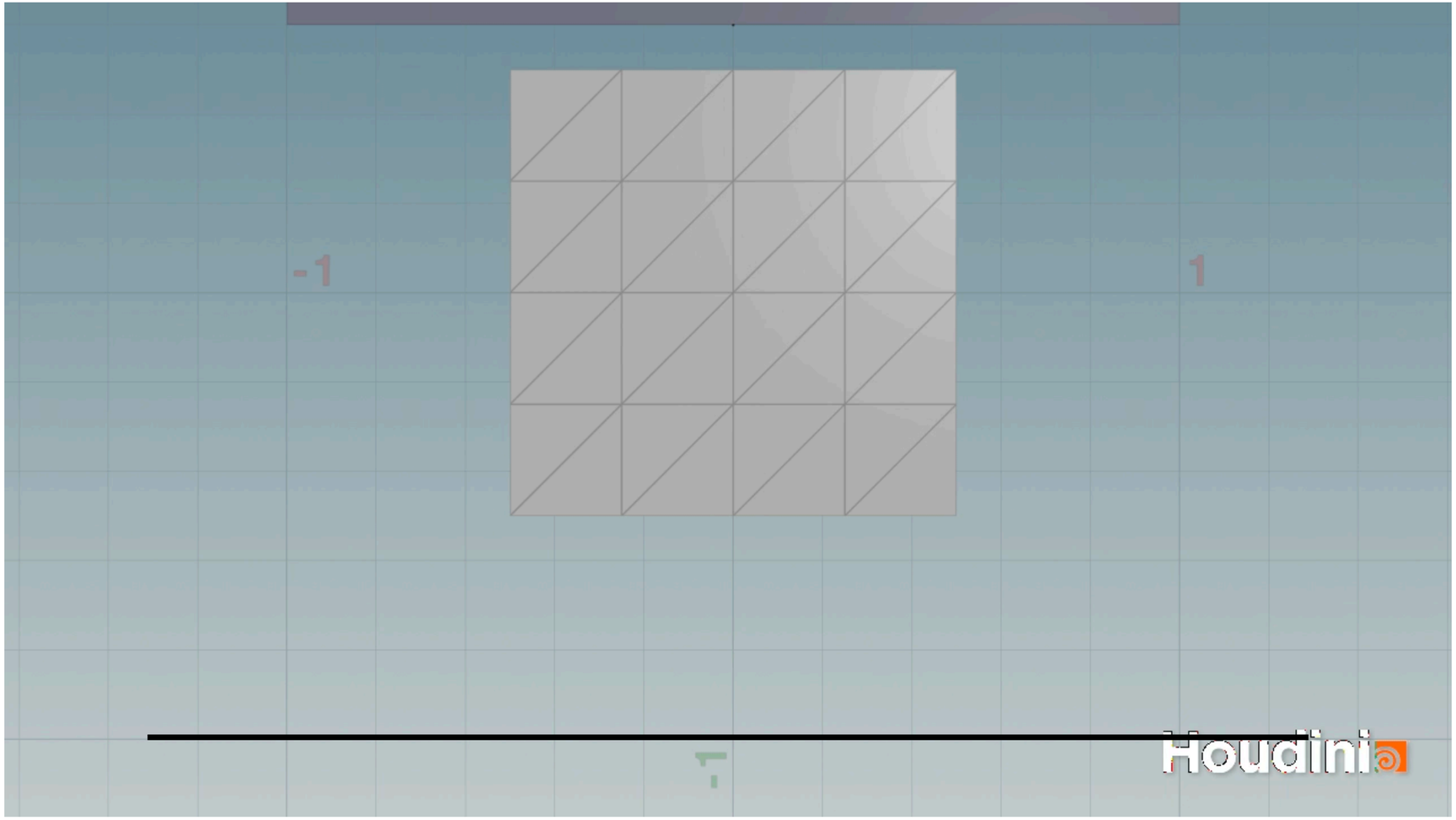
k_d=0

k_d=0.005

k_d=0.1

Yunuo Chen*, Minchen Li*, Lei Lan, Hao Su, Yin Yang, Chenfanfu Jiang. A Unified Newton Barrier Method for Multibody Dynamics. ACM ToG (SIGGRAPH), 2022.

Simulating Moving BC/Obstacles



Formulation

$$\min_x \; E(x) = \frac{1}{2}\|x - (x^n + hv^n)\|_M^2 + h^2 P(x). \qquad \text{s.t.} \quad A^{n+1}x = b^{n+1}$$

Inertia term

Elasticity

Selecting BC DOF

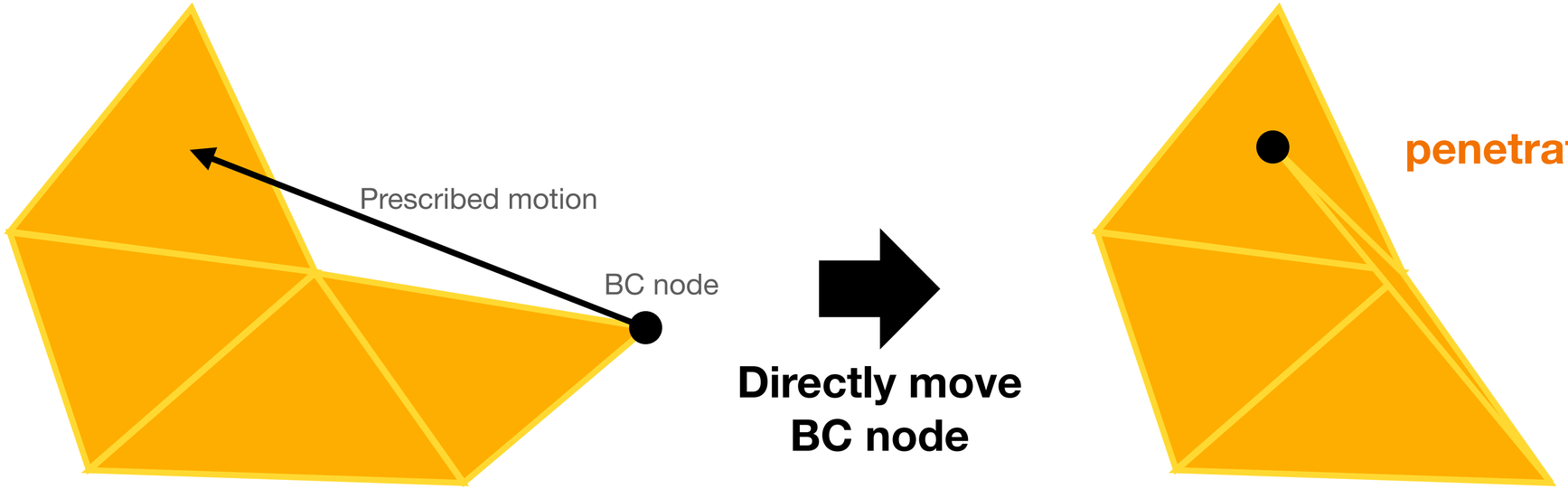
Prescribing BC values

DOF Elimination Method: (if $A^{n+1}x^n = b^{n+1}$)

$$H = \begin{bmatrix} 4 & -1 & -1 & -1 \\ -1 & 4 & -1 & -1 \\ -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 \end{bmatrix}, \quad \text{and} \quad g = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \xrightarrow{\text{Fix node } x_2} \begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_{11} \\ \Delta x_{12} \\ \Delta x_{21} \\ \Delta x_{22} \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ 0 \\ 0 \end{bmatrix}$$

What if $A^{n+1}x^n \neq b^{n+1}$?

Start the optimization at x where $A^{n+1}x = b^{n+1}$?



May also result in slow convergence

Penalty Methods

Formulation

$$\min_x \frac{1}{2}\|x-(x^n+hv^n)\|_M^2 + h^2\sum P(x) \quad \textbf{s.t.} \quad A^{n+1}x = b^{n+1}$$

$$\min_x \frac{1}{2}\|x-(x^n+hv^n)\|_M^2 + h^2\sum P(x) + \frac{\kappa_M}{2}\|A^{n+1}x-b^{n+1}\|_W^2$$

When $\kappa_M \to \infty$,

the solution is accurate

Algorithm:

Initialize κ_M

While solution not accurate enough

$$\min_x E(x) + P_M(x,\kappa_M)$$

$$\kappa_M \leftarrow 2\kappa_M$$

Penalty Methods

Remarks

Algorithm:

Initialize κ_M

While not accurate enough

$$\min_x E(x) + P_M(x, \kappa_M)$$

$$\kappa_M \leftarrow 2\kappa_M$$

- **Measure solution accuracy:**

  Check every constraint $w_i |(A^{n+1}x - b^{n+1})_i| < \epsilon$

- **Don't want super large** κ_M **(ill-conditioning)**

  use Augmented Lagrangian for high accuracy:

$$\max_\lambda \min_x E(x) + \frac{\kappa_M}{2}\|A^{n+1}x - b^{n+1}\|_W^2 + \lambda^T\sqrt{W}(A^{n+1}x - b^{n+1})$$

Solve with primal-dual method, i.e. alternate between

$$\min_x E(x) + \frac{\kappa_M}{2}\|A^{n+1}x - b^{n+1}\|_W^2 + \lambda^T\sqrt{W}(A^{n+1}x - b^{n+1})$$

and $\lambda \leftarrow \lambda + \kappa_M\sqrt{W}(A^{n+1}x - b^{n+1})$

For both methods, can switch to DOF elimination when accurate enough.

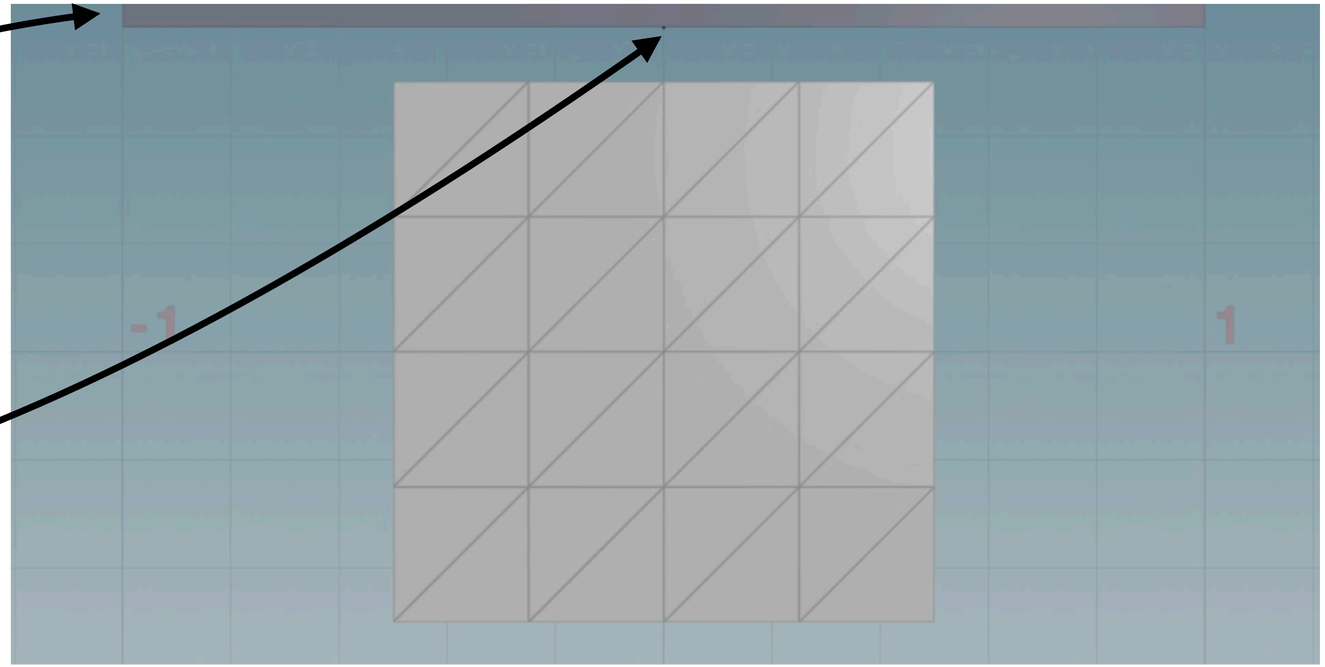
Converges to accurate solution with finite κ_M

Case Study: Compressing Square

New DOF: Ceiling

A moving ceiling with $n = (0, -1)$

Take its origin o as DOF



simulator.py

```
25 [x, e] = square_mesh.generate(side_len, n_seg)       # node
       positions and edge node indices
26 x = np.append(x, [[0.0, side_len * 0.6]], axis=0)    # ceil
       origin (with normal [0.0, -1.0])
```

Case Study: Compressing Square

Normal Contact between Solids and Ceiling

A moving ceiling with $n = (0, -1)$, taking its origin o as DOF

Distance between node x: $d(\mathbf{x}, \boldsymbol{o}) = \mathbf{n}^T(\mathbf{x} - \boldsymbol{o}), \quad \nabla d(\mathbf{x}, \boldsymbol{o}) = \begin{bmatrix} \mathbf{n} \\ -\mathbf{n} \end{bmatrix}, \quad \nabla^2 d(\mathbf{x}, \boldsymbol{o}) = \mathbf{0}$

BarrierEnergy.py

Energy

```
    # ceil:
    n = np.array([0.0, -1.0])
    for i in range(0, len(x) - 1):
        d = n.dot(x[i] - x[-1])
        if d < dhat:
            s = d / dhat
            sum += contact_area[i] * dhat * kappa / 2 * (s - 1) * math.log(s)
```

Gradient

```
    # ceil:
    n = np.array([0.0, -1.0])
    for i in range(0, len(x) - 1):
        d = n.dot(x[i] - x[-1])
        if d < dhat:
            s = d / dhat
            local_grad = contact_area[i] * dhat * (kappa / 2 * (math.log(s) / dhat + (s - 1) / d)) * n
            g[i] += local_grad
            g[-1] -= local_grad
```

Hessian

```
    # ceil:
    n = np.array([0.0, -1.0])
    for i in range(0, len(x) - 1):
        d = n.dot(x[i] - x[-1])
        if d < dhat:
            local_hess = contact_area[i] * dhat * kappa / (2 * d * d * dhat) * (d + dhat) * np.outer(n, n)
            index = [i, len(x) - 1]
            for nI in range(0, 2):
                for nJ in range(0, 2):
                    for c in range(0, 2):
                        for r in range(0, 2):
                            IJV[0].append(index[nI] * 2 + r)
                            IJV[1].append(index[nJ] * 2 + c)
                            IJV[2] = np.append(IJV[2], ((-1) ** (nI != nJ)) * local_hess[r, c])
```

Continuous Collision Detection

```
    # ceil:
    n = np.array([0.0, -1.0])
    for i in range(0, len(x) - 1):
        p_n = (p[i] - p[-1]).dot(n)
        if p_n < 0:
            alpha = min(alpha, 0.9 * n.dot(x[i] - x[-1]) / -p_n)
```

Case Study: Compressing Square

Intermediate Results: Falling Ceiling

Case Study: Compressing Square

Penalty Energy

Moving BC Setup:

```
16 DBC = [(n_seg + 1) * (n_seg + 1)]        # dirichlet node index
17 DBC_v = [np.array([0.0, -0.5])]          # dirichlet node
       velocity
18 DBC_limit = [np.array([0.0, -0.6])]      # dirichlet node limit
        position
```

simulator.py

```
20     DBC_target = [] # target position of each DBC in the
       current time step
21     for i in range(0, len(DBC)):
22         if (DBC_limit[i] - x_n[DBC[i]]).dot(DBC_v[i]) > 0:
23             DBC_target.append(x_n[DBC[i]] + h * DBC_v[i])
24         else:
25             DBC_target.append(x_n[DBC[i]])
26     DBC_stiff = 10  # initialize stiffness for DBC springs
```

time_integrator.py

```
1  import numpy as np
2
3  def val(x, m, DBC, DBC_target, k):
4      sum = 0.0
5      for i in range(0, len(DBC)):
6          diff = x[DBC[i]] - DBC_target[i]
7          sum += 0.5 * k * m[DBC[i]] * diff.dot(diff)
8      return sum
9
10 def grad(x, m, DBC, DBC_target, k):
11     g = np.array([[0.0, 0.0]] * len(x))
12     for i in range(0, len(DBC)):
13         g[DBC[i]] = k * m[DBC[i]] * (x[DBC[i]] - DBC_target[i
       ])
14     return g
15
16 def hess(x, m, DBC, DBC_target, k):
17     IJV = [[0] * 0, [0] * 0, np.array([0.0] * 0)]
18     for i in range(0, len(DBC)):
19         for d in range(0, 2):
20             IJV[0].append(DBC[i] * 2 + d)
21             IJV[1].append(DBC[i] * 2 + d)
22             IJV[2] = np.append(IJV[2], k * m[DBC[i]])
23     return IJV
```

SpringEnergy.py

Penalty term:

$$P_m^{n+1}(x) = \frac{\kappa_M}{2}\|A^{n+1}x - b^{n+1}\|_W^2$$

$$\nabla P_m^{n+1}(x) = \kappa_M (A^{n+1})^T W (A^{n+1}x - b^{n+1})$$

$$\nabla^2 P_m^{n+1}(x) = \kappa_M (A^{n+1})^T W A^{n+1}$$

Case Study: Compressing Square

Stiffness Adjustment and DOF Elimination

Check constraint satisfaction:

```
90      # check whether each DBC is satisfied
91      DBC_satisfied = [False] * len(x)
92      for i in range(0, len(DBC)):
93          if LA.norm(x[DBC[i]] - DBC_target[i]) / h < tol:
94              DBC_satisfied[DBC[i]] = True
```

Use DOF Elimination for satisfied constraints:

```
95      # eliminate DOF if it's a satisfied DBC by modifying
        gradient and Hessian for DBC:
96      for i, j in zip(*projected_hess.nonzero()):
97          if (is_DBC[int(i / 2)] & DBC_satisfied[int(i / 2)]) |
        (is_DBC[int(j / 2)] & DBC_satisfied[int(i / 2)]):
98              projected_hess[i, j] = (i == j)
99      for i in range(0, len(x)):
100         if is_DBC[i] & DBC_satisfied[i]:
101             reshaped_grad[i * 2] = reshaped_grad[i * 2 + 1] =
        0.0
102     return [spsolve(projected_hess, -reshaped_grad).reshape(
        len(x), 2), DBC_satisfied]
```

Adjust stiffness when optimization converged but not all constraints are satisfied:

```
31      [p, DBC_satisfied] = search_dir(x, e, x_tilde, m, l2, k, n
        , o, contact_area, (x - x_n) / h, mu_lambda, is_DBC, DBC,
        DBC_target, DBC_stiff, tol, h)
32      while (LA.norm(p, inf) / h > tol) | (sum(DBC_satisfied) !=
         len(DBC)):   # also check whether all DBCs are satisfied
33          print('Iteration', iter, ':')
34          print('residual =', LA.norm(p, inf) / h)
35
36          if (LA.norm(p, inf) / h <= tol) & (sum(DBC_satisfied)
        != len(DBC)):
37              # increase DBC stiffness and recompute energy
        value record
38              DBC_stiff *= 2
39              E_last = IP_val(x, e, x_tilde, m, l2, k, n, o,
        contact_area, (x - x_n) / h, mu_lambda, DBC, DBC_target,
        DBC_stiff, h)
```

time_integrator.py

Case Study: Compressing Square

Summary

- augmenting the Incremental Potential with extra spring energies on the DBC nodes;
- adaptively scaling up the penalty stiffness when needed;
- eliminating DOFs for those BC nodes already close enough to the target; and
- ensuring all BCs are satisfied at convergence.

Demo!

github.com/liminchen/solid-sim-tutorial /5_mov_dirichlet

A “Quick-Start Guide” Completed!

- We have covered:
  - Discrete Shape Representations
  - Numerical Time Integration
    - Optimization Time Integration Framework
  - Mass-Spring Elasticity
  - (Moving) Sticky/Slip Dirichlet Boundary Conditions
  - Robust and Accurate Frictional Contact
- Next: Continuum Mechanics, Spatial Reduction, Fluids, …

Next Lecture: Hyperelasticity

Image Sources