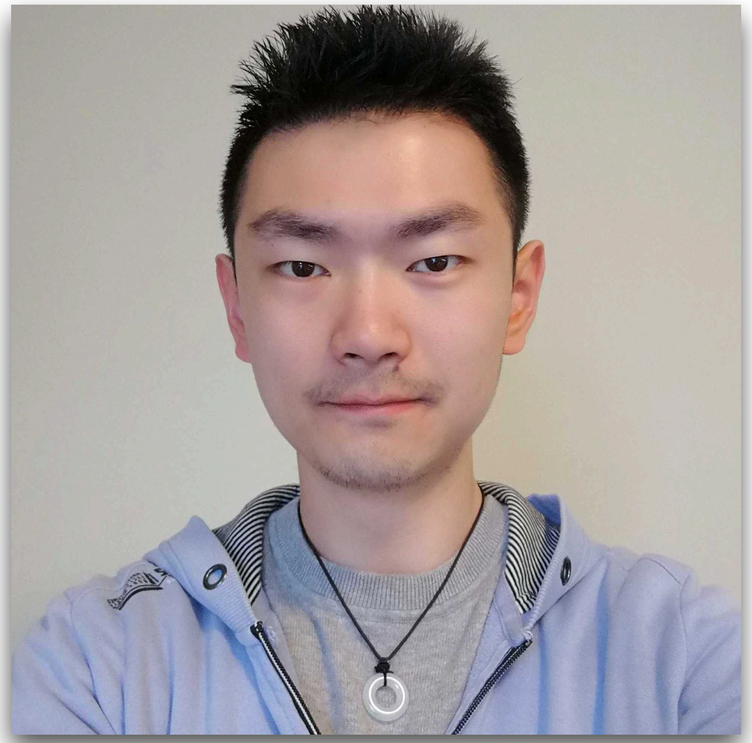
Introduction to Optimization Time Integration for Solids and Fluids

Eurographics 2026 Tutorial

Speakers: Minchen Li (CMU), Jiayi (Eris) Zhang (Stanford)

Speakers



Minchen Li
Assistant Professor
Carnegie Mellon University

Jiayi (Eris) Zhang
PhD student
Stanford University

Topics Today

Part I: Fundamentals and The Big Picture (Minchen)

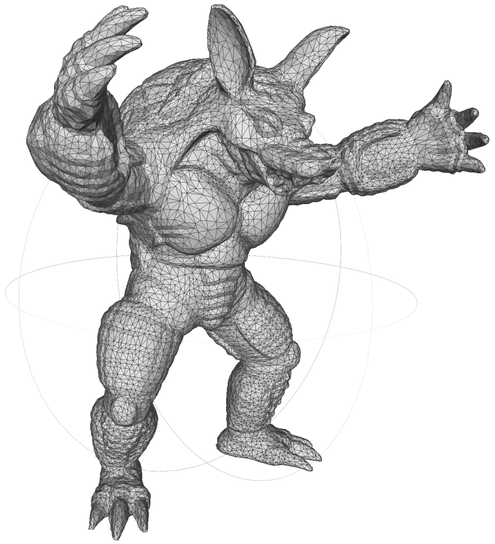
- Distortion Minimization
- Elastodynamics Simulation via Optimization Time Integration
- Case Study: Mass-Spring System
- More Topics on Optimization Time Integration

Part II: Advanced Topics (Eris)

- Constrained System
- Reduced-Order System
- Multilevel System
- Adaptive Simulation

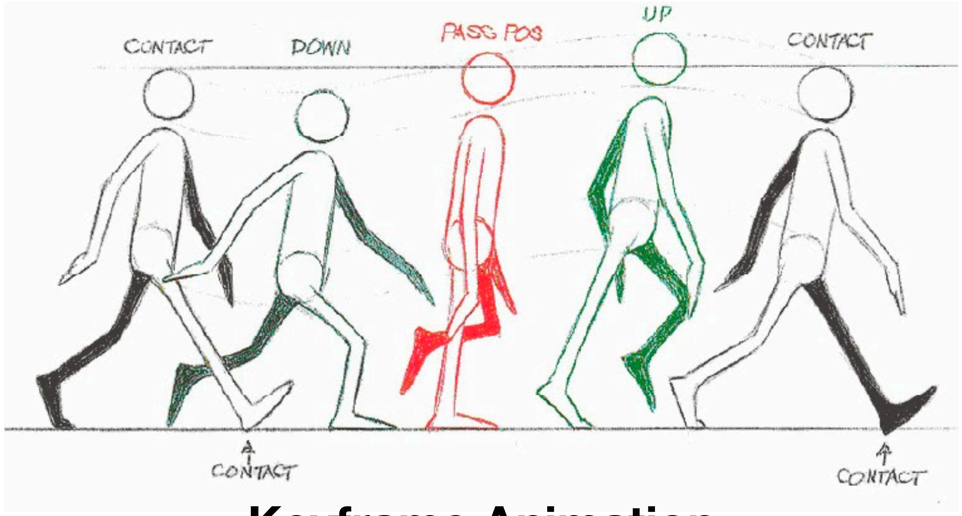
Computer Graphics:

Generating Realistic Visual Effects via Computing

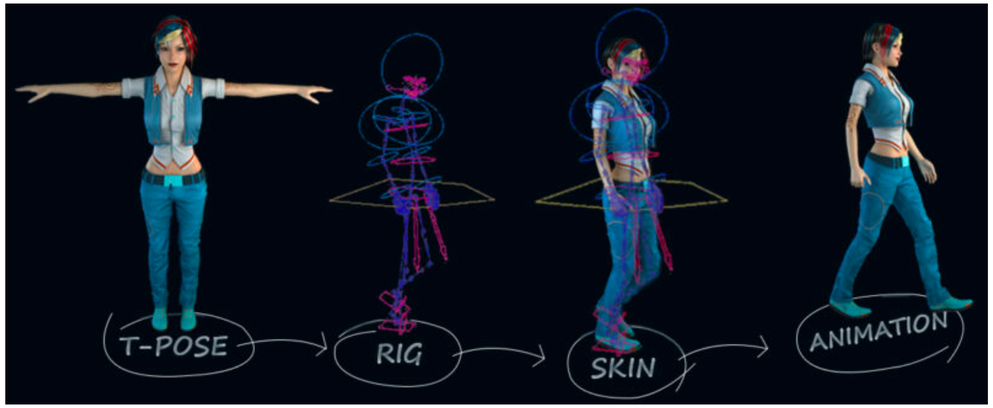


Geometry → Appearance → Animation

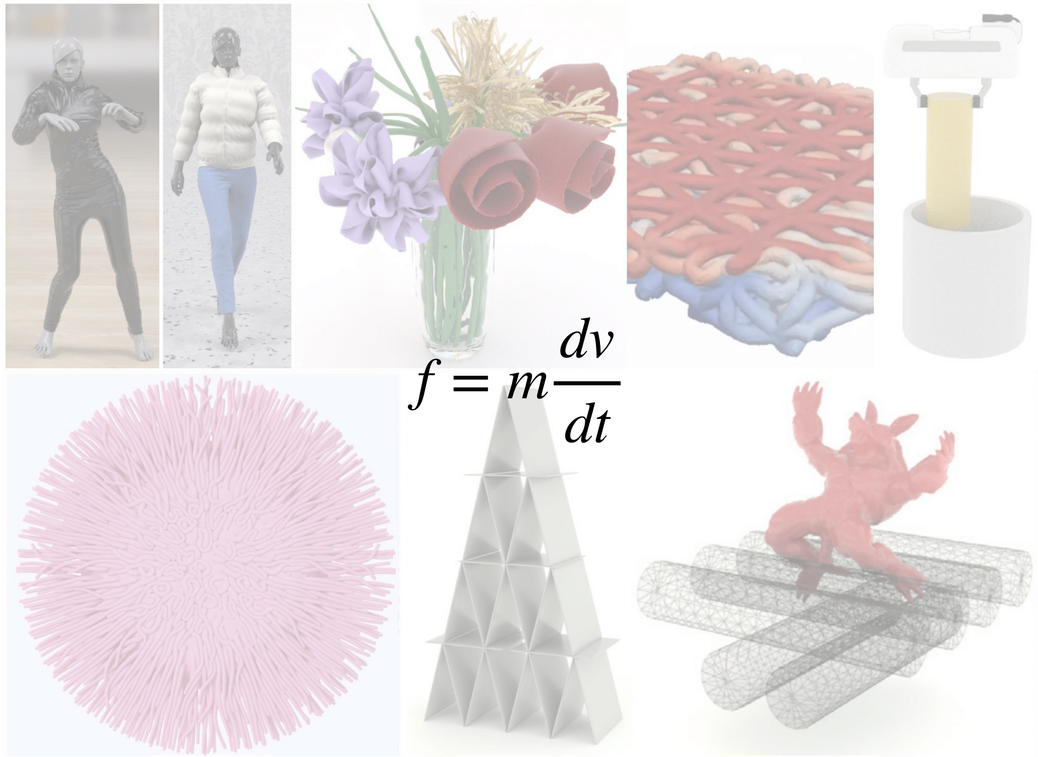
Animation



Keyframe Animation

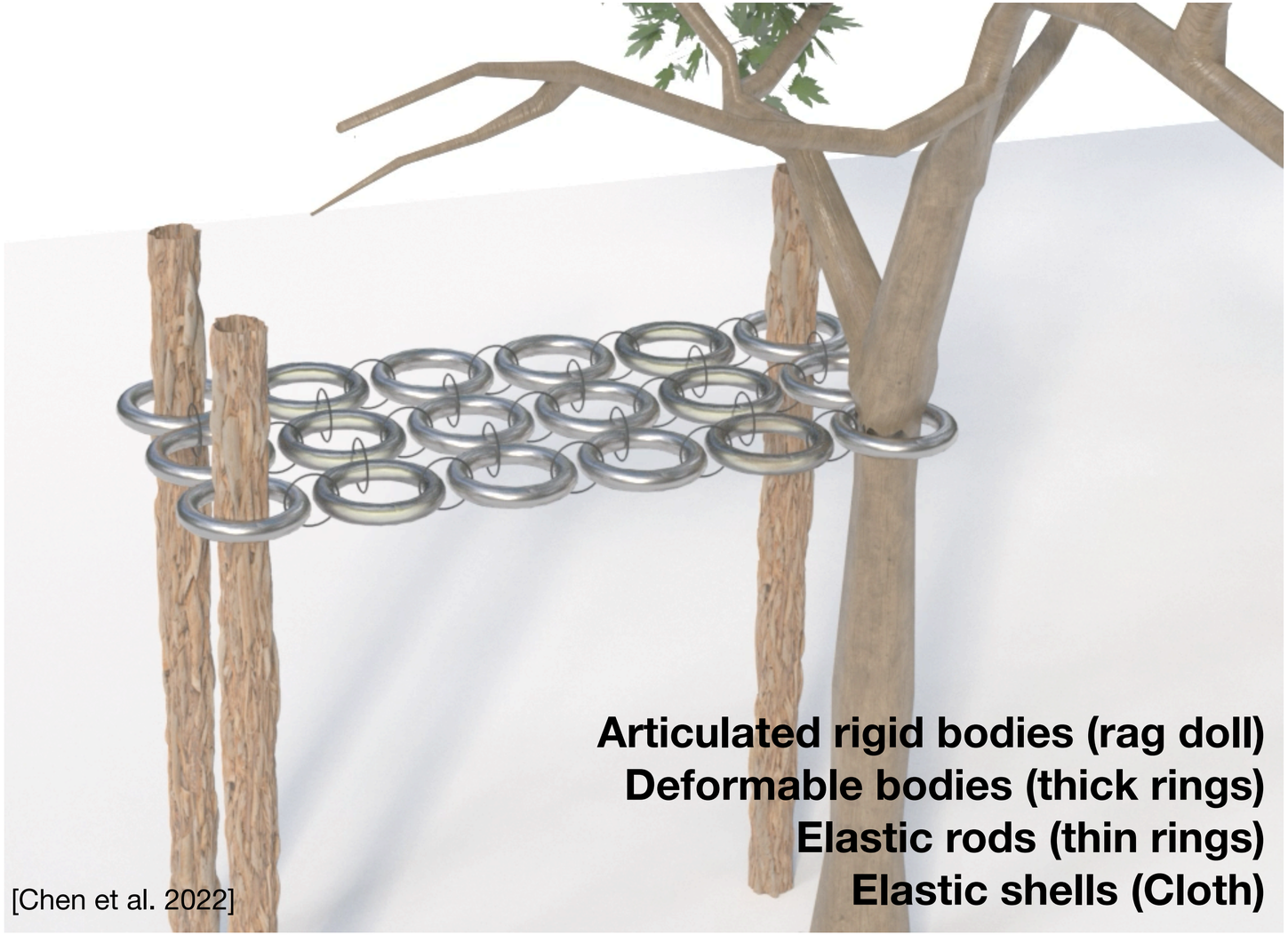


Skinning Animation



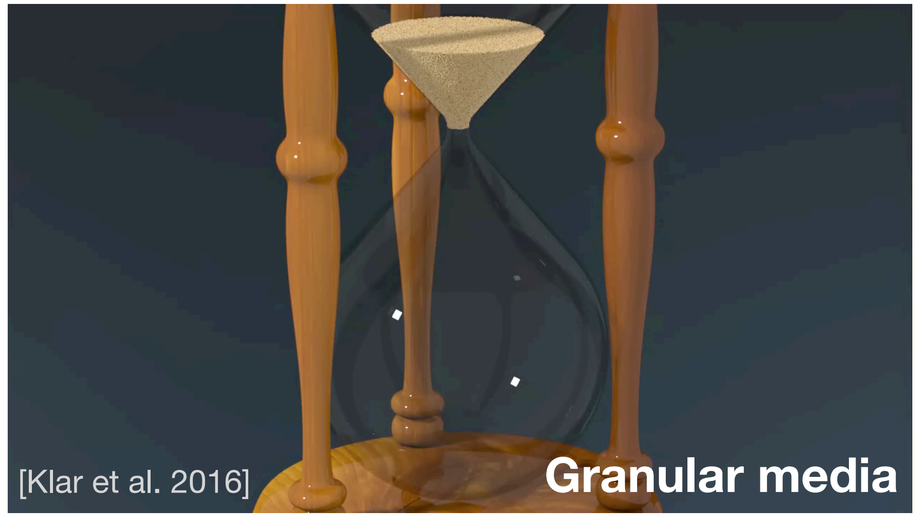
Physics-based Animation

Physics-based Animation



Articulated rigid bodies (rag doll)
Deformable bodies (thick rings)
Elastic rods (thin rings)
Elastic shells (Cloth)

[Chen et al. 2022]



[Klar et al. 2016]

Granular media

[Xie et al. 2023]

Fluids

Optimization Time Integration

A Reliable Simulation Approach based on Numerical Optimization

Topics Today

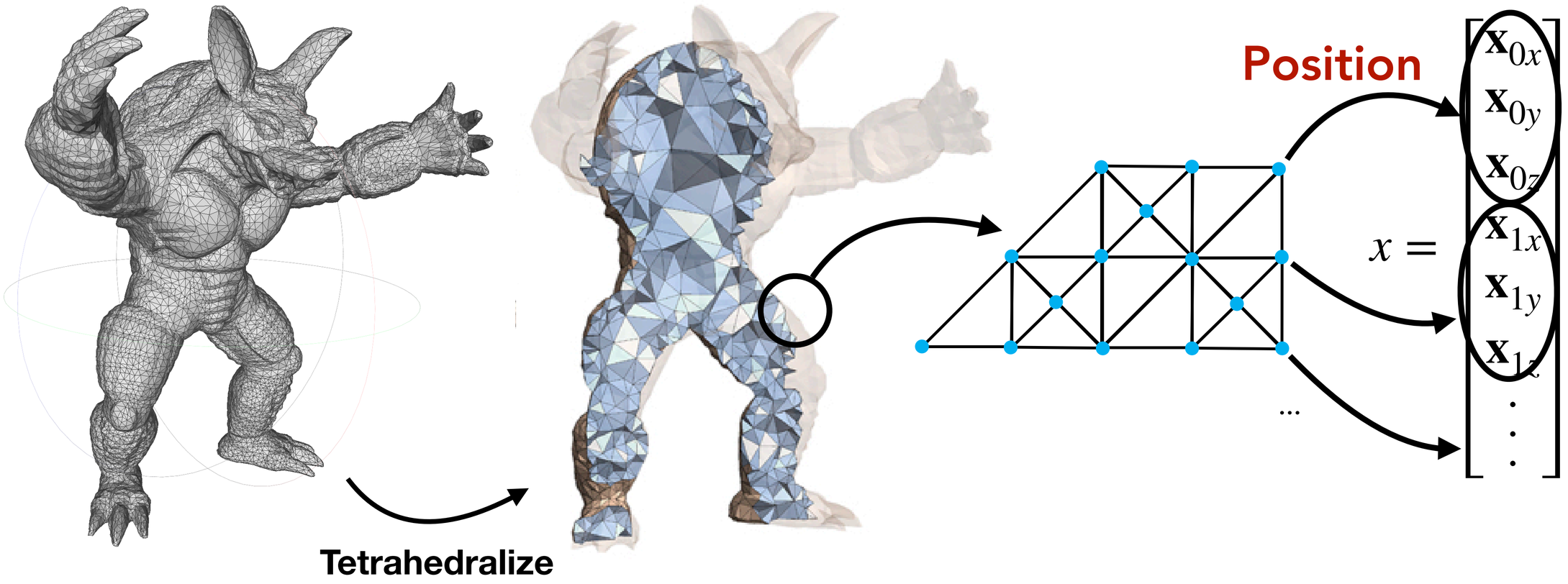
Part I: Fundamentals and The Big Picture (Minchen)

- Distortion Minimization
- Elastodynamics Simulation via Optimization Time Integration
- Case Study: Mass-Spring System
- More Topics on Optimization Time Integration

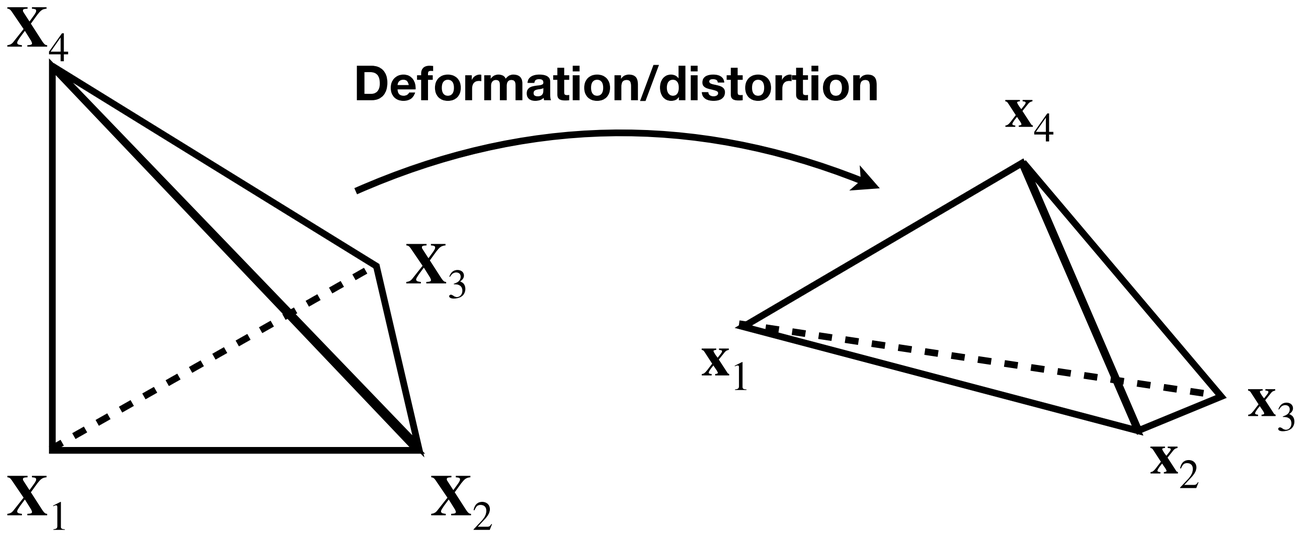
Part II: Advanced Topics (Eris)

- Constrained System
- Reduced-Order System
- Multilevel System
- Adaptive Simulation

Spatial Discretization



Measure Distortion Locally



Distortion energy, e.g. $\Psi(\mathbf{F}) = \mu\|\mathbf{F} - \mathbf{R}\|_F^2$

- $\mathbf{R} = \mathbf{U}\mathbf{V}^T$ **is the closest rotation to** $\mathbf{F}$
- $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ **is the SVD**

— As-Rigid-As-Possible (ARAP) [Igarashi et al. 2005]

Assume uniform deformation

Distortion over the whole mesh: $\sum_e V_e \Psi(\mathbf{F}_e)$

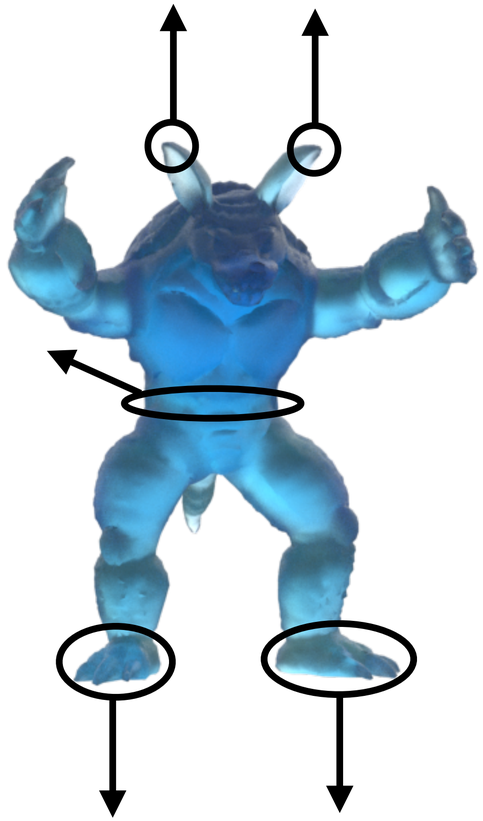
Affine transformation:

$$\mathbf{F} = [\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_1, \mathbf{x}_4 - \mathbf{x}_1][\mathbf{X}_2 - \mathbf{X}_1, \mathbf{X}_3 - \mathbf{X}_1, \mathbf{X}_4 - \mathbf{X}_1]^{-1}$$

$$\forall \mathbf{X},\ \mathbf{x}(\mathbf{X}) = \mathbf{F}(\mathbf{X} - \mathbf{X}_1) + \mathbf{x}_1$$

(V_e **is the volume of tetrahedron element e)**

Mesh Deformation via Distortion Minimization



- S selects the anchors
- $\hat{x}$ contains the target position

$$\min_x \sum_e V_e \Psi(\mathbf{F}_e) \quad s.t. \quad Sx = \hat{x}$$

We could move the anchors to the target position, and only optimize for the free vertices

$$\min_x \sum_e V_e \Psi(\mathbf{F}_e)$$

Newton's Method for Distortion Minimization

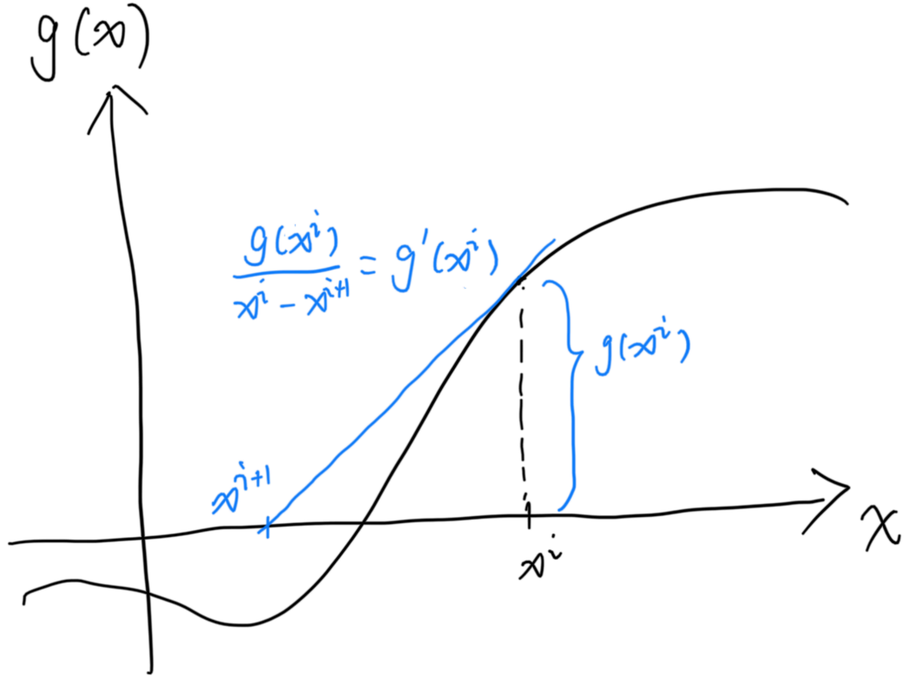
Formulation

Let $g(x) = \nabla E(x)$, $E(x) = \sum_e V_e \Psi(\mathbf{F}_e)$

We want to solve $g(x) = 0$

Newton's method in 1D:

- **Start from initial guess** x^0
- **For each iteration (until convergence)**
  - $x^{i+1} \leftarrow x^i - g(x^i)/g'(x^i)$



Newton's Method for Distortion Minimization

Formulation

Let $g(x) = \nabla E(x)$, $E(x) = \sum_e V_e \Psi(\mathbf{F}_e)$

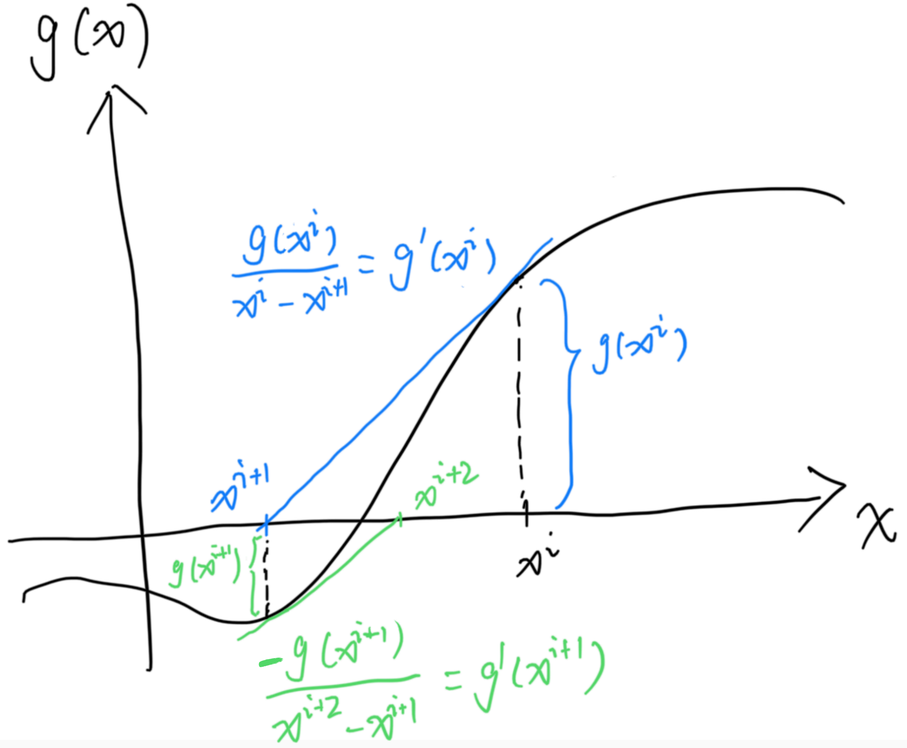
We want to solve $g(x) = 0$

Newton's method in 1D:

- **Start from initial guess** x^0
- **For each iteration (until convergence)**
  - $x^{i+1} \leftarrow x^i - g(x^i)/g'(x^i)$

In higher dimensions:

$$x^{i+1} \leftarrow x^i - (\nabla g(x^i))^{-1} g(x^i)$$



Derivation:

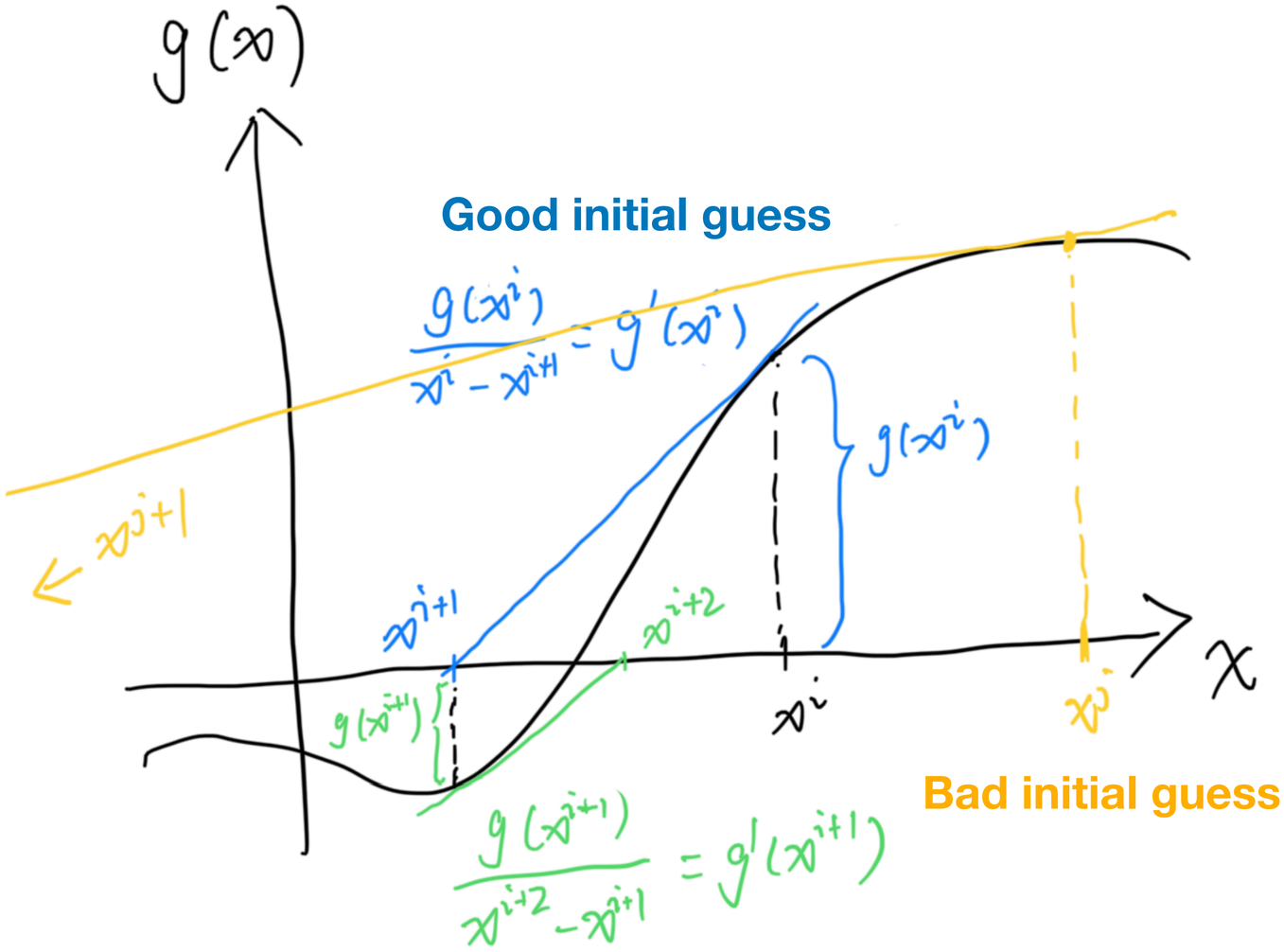
Linearly approximate $g(x) = 0$ **at** x^i **:**

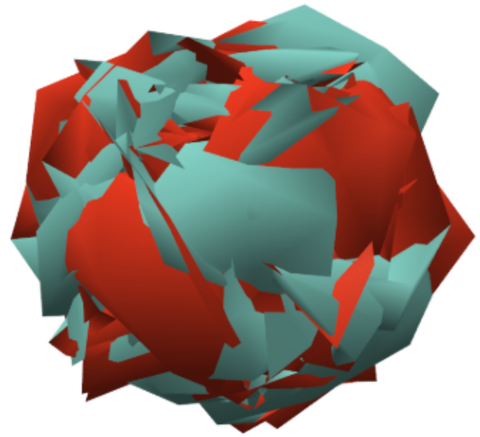
$$g(x) = g(x^i) + \nabla g(x^i)(x - x^i)$$

$$g(x^{i+1}) \approx g(x^i) + \nabla g(x^i)(x^{i+1} - x^i) = 0$$

Convergence Issue of Newton's Method

Over-shooting





Bad initial guess → Numerical explosion!

Robust Distortion Minimization

Newton's Method with Line Search

We want to solve $\nabla E(x) = 0$

Newton's method:

- **Start from initial guess** x^0
- **For each iteration (until convergence)**
  - $x^{i+1} \leftarrow x^i - (\nabla^2 E(x^i))^{-1} \nabla E(x^i)$

Let $p = -(\nabla^2 E(x^i))^{-1} \nabla E(x^i)$

Line Search along direction p**:**

$$\min_{\alpha} E(x^i + \alpha p)$$

$$x^{i+1} \leftarrow x^i + \alpha p$$

Theory:

If p **is a descent direction at** $x = x^i$ **(like** $-\nabla E(x^i)$**),**

$$\exists \alpha > 0,\ s.t.\ E(x^i + \alpha p) < E(x^i)$$

— need $\nabla^2 E(x)$ **to be symmetric positive-definite**

Idea:

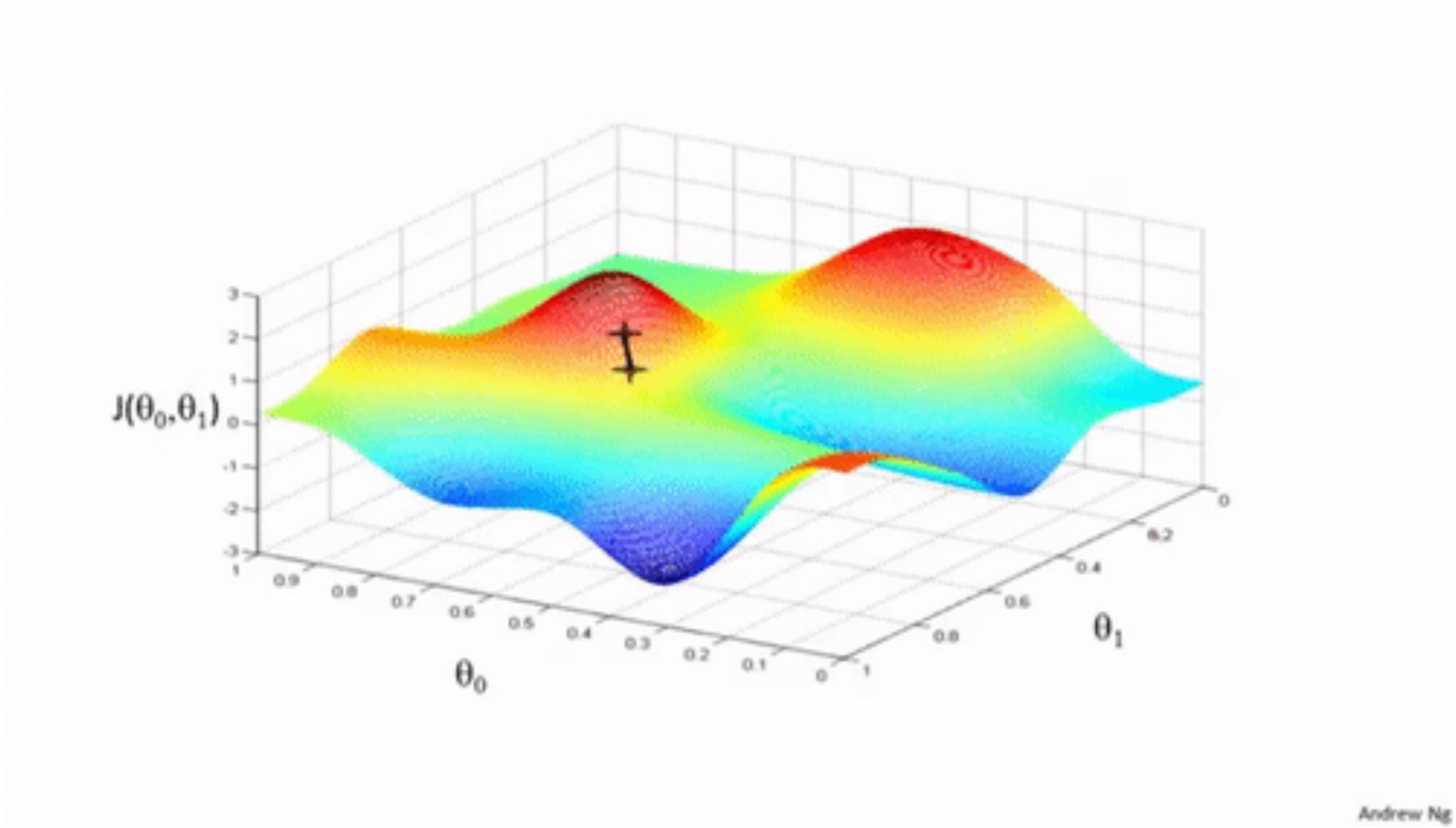
We can project $\nabla^2 E(x)$ **to a nearby SPD matrix for computing** p

Then we can ensure $E(x^{i+1}) < E(x^i)\ \forall i$

— no explosion!

Robust Distortion Minimization

Newton's Method with Line Search, 2D Illustration



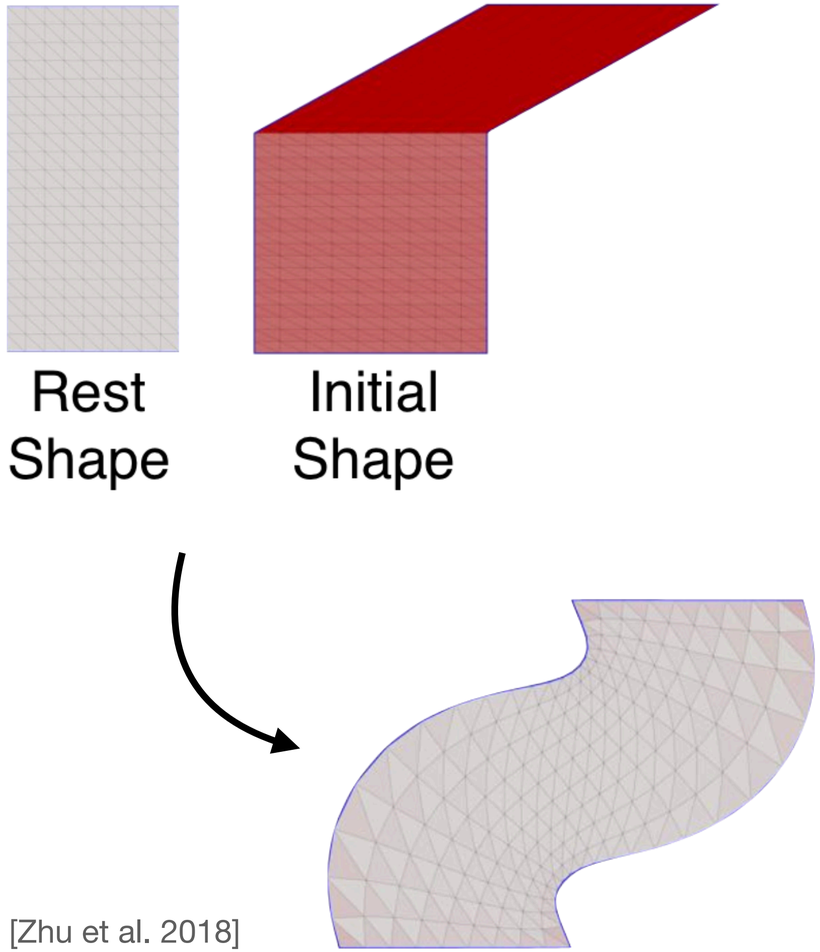
Andrew Ng

Robust Distortion Minimization

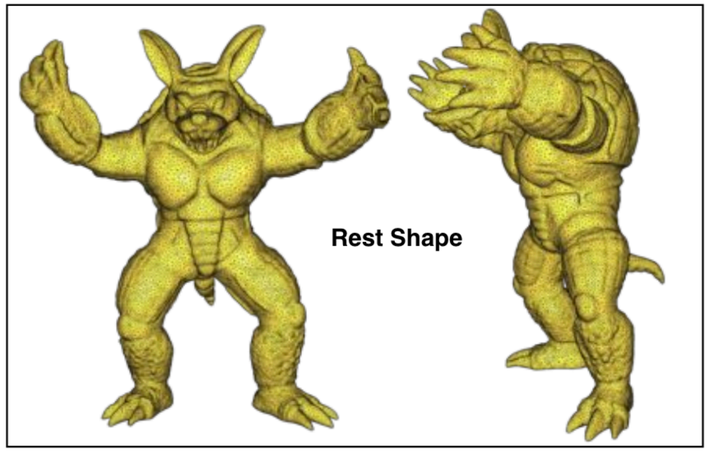
Pseudo-code

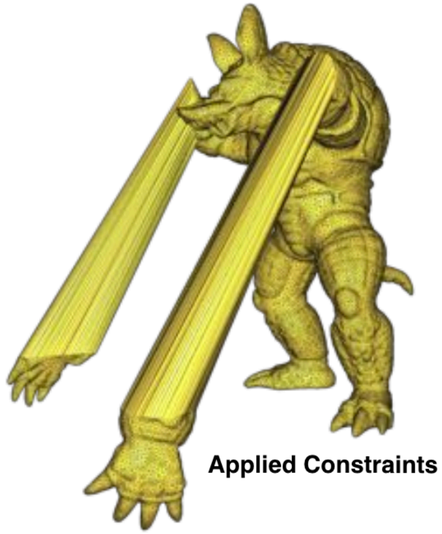
- $x \leftarrow$ initial guess
- Do
  - $P \leftarrow$ projectSPD($\nabla^2 E(x)$)
  - $p \leftarrow P^{-1} \nabla E(x)$
  - $\alpha \leftarrow 1$
  - While $E(x + \alpha p) > E(x)$
    - $\alpha \leftarrow \alpha/2$
  - $x \leftarrow x + \alpha p$
- While $\|p\| < \epsilon$

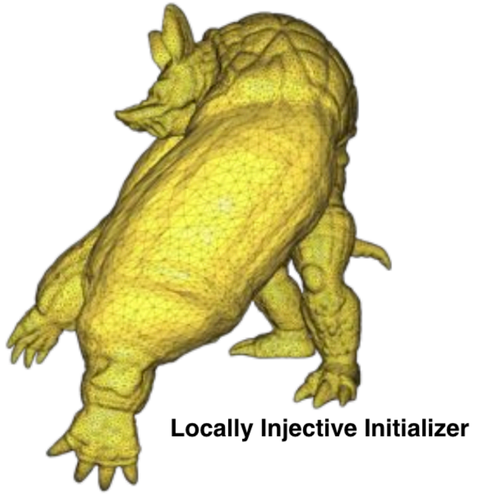
Distortion Minimization Examples



[Zhu et al. 2018]







Topics Today

Part I: Fundamentals and The Big Picture (Minchen)

- Distortion Minimization
- Elastodynamics Simulation via Optimization Time Integration
- Case Study: Mass-Spring System
- More Topics on Optimization Time Integration

Part II: Advanced Topics (Eris)

- Constrained System
- Reduced-Order System
- Multilevel System
- Adaptive Simulation

Adding Dynamics

Distortion minimization: $\min_x E(x)$ $\quad E(x) = \sum_e V_e \Psi(\mathbf{F}_e)$

Elastodynamic simulation: $\min_x E(x)$ $\quad E(x) = \sum_e V_e \Psi(\mathbf{F}_e) + \boxed{\frac{1}{2\Delta t^2} \|x - (x^n + \Delta t v^n)\|_M^2}$

Inertia term

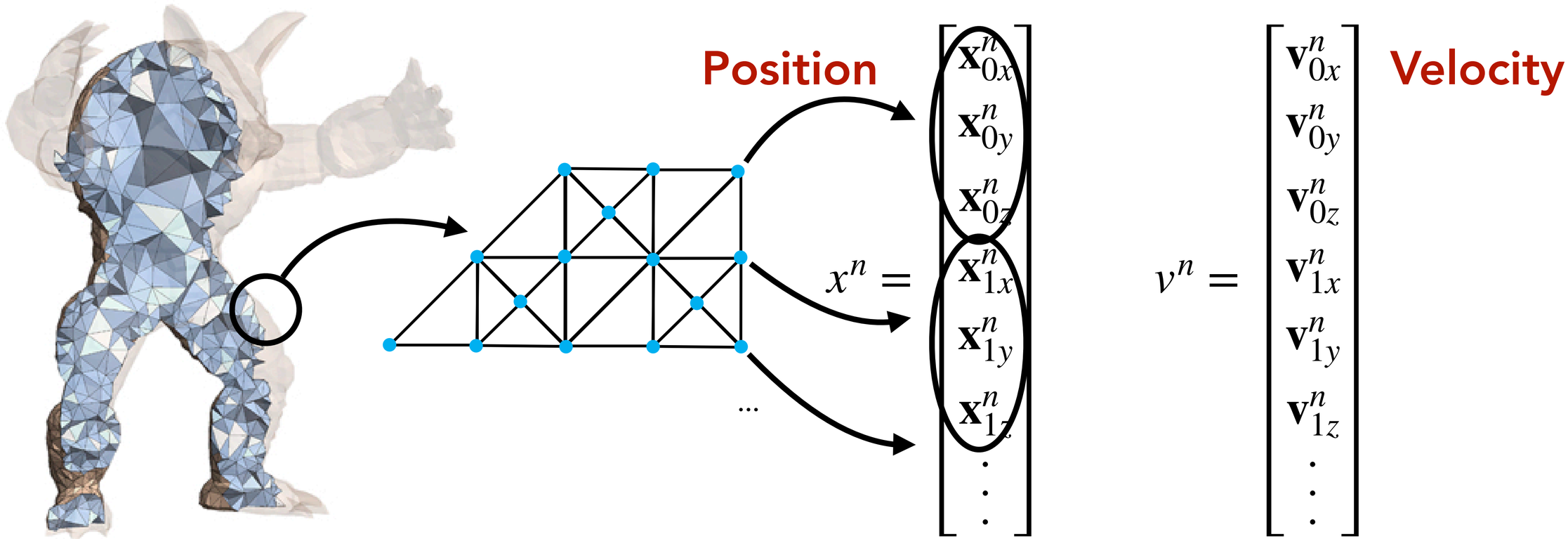
Δt: time step size

x^n: vertex positions at previous time step

v^n: velocity at previous time step

M: mass matrix

Spatial Discretization with Dynamics



Governing Equation (Conservation of Momentum)

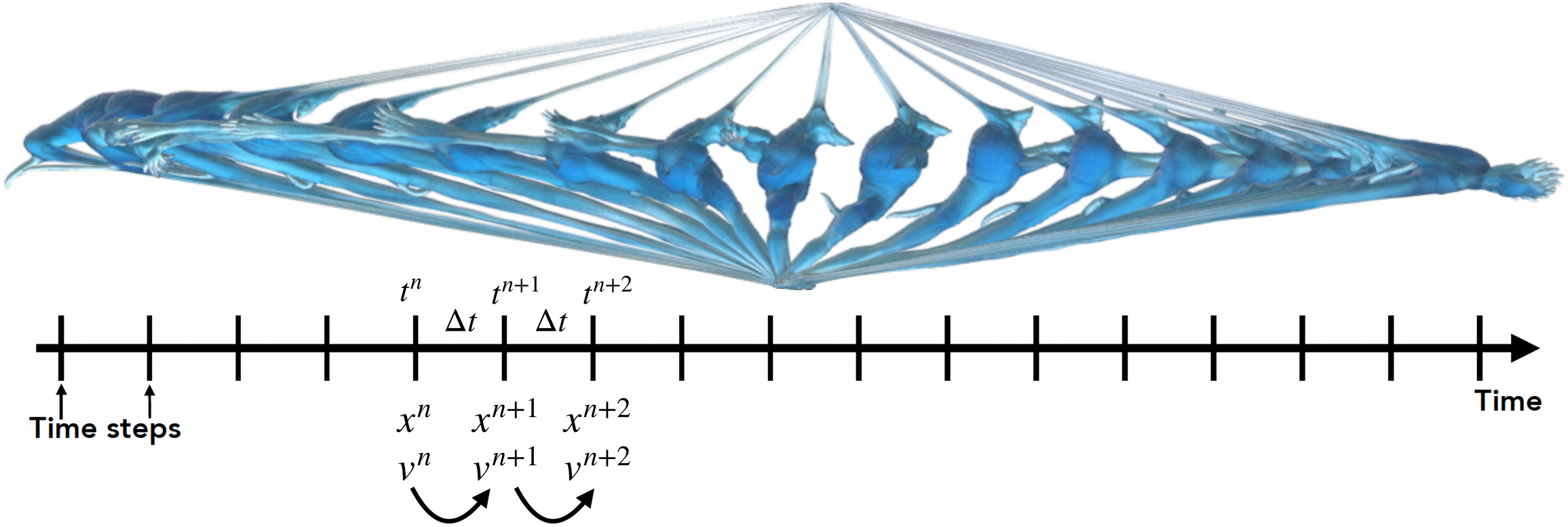
- The spatially discrete, temporally continuous form

$$\frac{dx}{dt} = v,$$
$$M\frac{dv}{dt} = f.$$

- Mass matrix (for now)

$$M = \begin{pmatrix} m_1 & & & \\ & m_1 & & \\ & & m_2 & \\ & & & m_2 \end{pmatrix}$$

Time Stepping (Time Integration)



Governing Equation (Temporally Discrete)

Forward Difference, Forward Euler

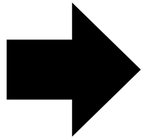
- Forward difference approximation on velocity and acceleration

$$\left(\frac{\mathbf{d}x}{\mathbf{d}t}\right)^n \approx \frac{x^{n+1}-x^n}{\Delta t} \qquad \left(\frac{\mathbf{d}v}{\mathbf{d}t}\right)^n \approx \frac{v^{n+1}-v^n}{\Delta t}$$

$$\left(f(t^n+\Delta t) = f(t^n) + \frac{\mathbf{d}f}{\mathbf{d}t}(t^n)\Delta t + O(\Delta t^2)\right)$$

Taylor's expansion

$$\frac{x^{n+1}-x^n}{\Delta t} = v^n,$$
$$M\frac{v^{n+1}-v^n}{\Delta t} = f^n.$$



$$x^{n+1} = x^n + \Delta t v^n,$$
$$v^{n+1} = v^n + \Delta t M^{-1} f^n$$

Newton's 2nd Law (Temporally Discrete)

Forward and Backward Difference, Symplectic Euler

- Forward difference on acceleration, backward difference on velocity

$$x^{n+1} = x^n + \Delta t v^{n+1}$$

$$v^{n+1} = v^n + \Delta t M^{-1} f^n$$

Newton's 2nd Law (Temporally Discrete)

Backward Difference, Backward Euler (or Implicit Euler)

- Backward difference approximation on velocity and acceleration

$$x^{n+1} = x^n + \Delta t v^{n+1},$$
$$v^{n+1} = v^n + \Delta t M^{-1} f^{n+1}$$

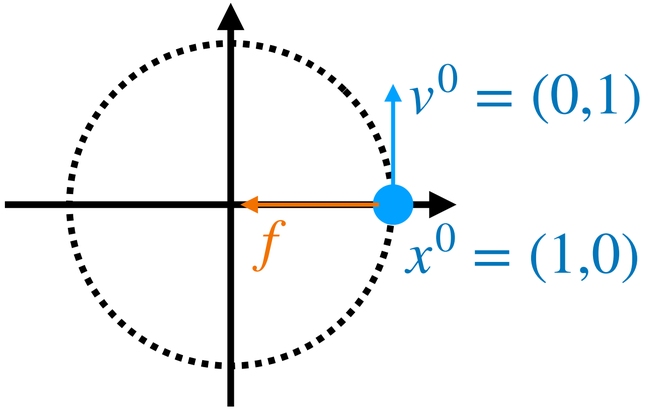
$$f^{n+1} = f(x^{n+1})$$

Needs to solve a system of equations:

$$M(x^{n+1} - (x^n + \Delta t v^n)) - \Delta t^2 f(x^{n+1}) = 0.$$

Stability of Forward, Symplectic, and Backward Euler

Example on a uniform circular motion

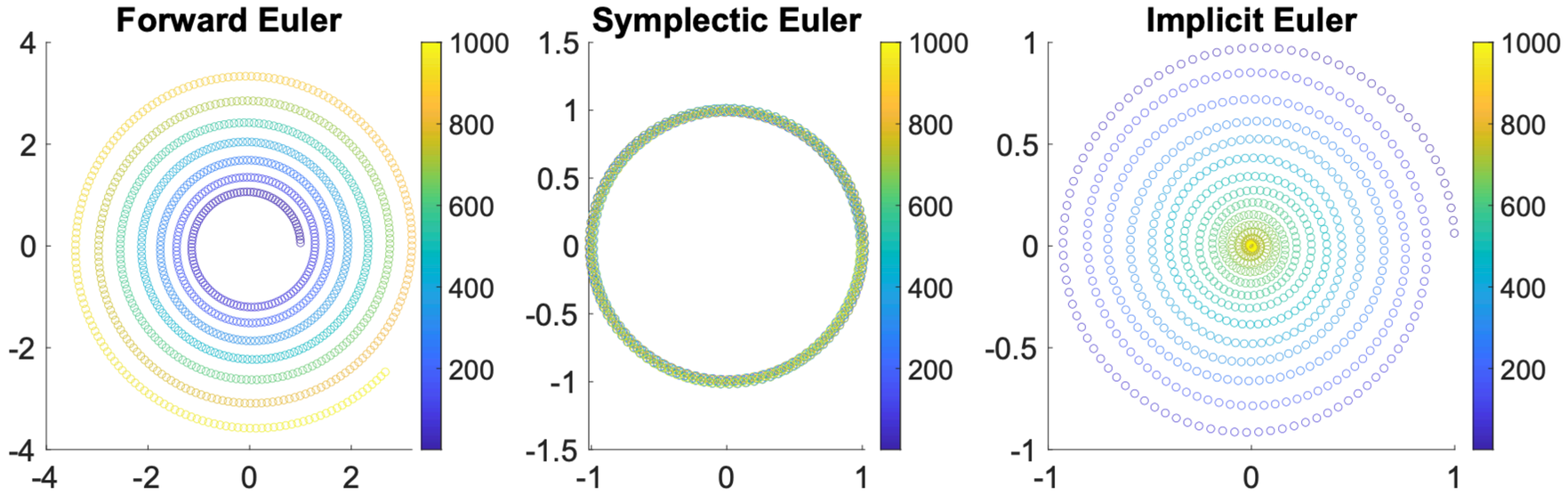


Problem Setup

$$x^{n+1} = x^n + \Delta t v^n,$$
$$v^{n+1} = v^n + \Delta t M^{-1} f^n$$

$$x^{n+1} = x^n + \Delta t v^{n+1}$$
$$v^{n+1} = v^n + \Delta t M^{-1} f^n$$

$$x^{n+1} = x^n + \Delta t v^{n+1},$$
$$v^{n+1} = v^n + \Delta t M^{-1} f^{n+1}$$



Newton's Method for Backward Euler

Formulation

Let $g(x) = M(x - (x^n + \Delta t v^n)) - \Delta t^2 f(x)$

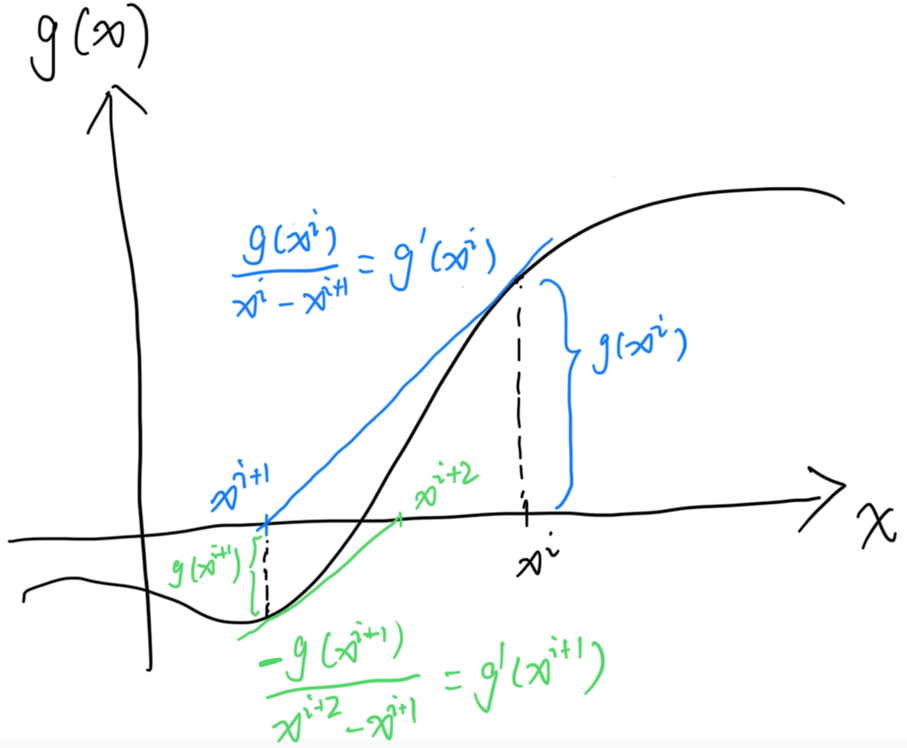
We want to solve $g(x) = 0$

Newton's method in 1D:

- **Start from initial guess** x^0
- **For each iteration (until convergence)**
  - $x^{i+1} \leftarrow x^i - g(x^i)/g'(x^i)$

In higher dimensions:

$$x^{i+1} \leftarrow x^i - (\nabla g(x^i))^{-1} g(x^i)$$



Derivation:

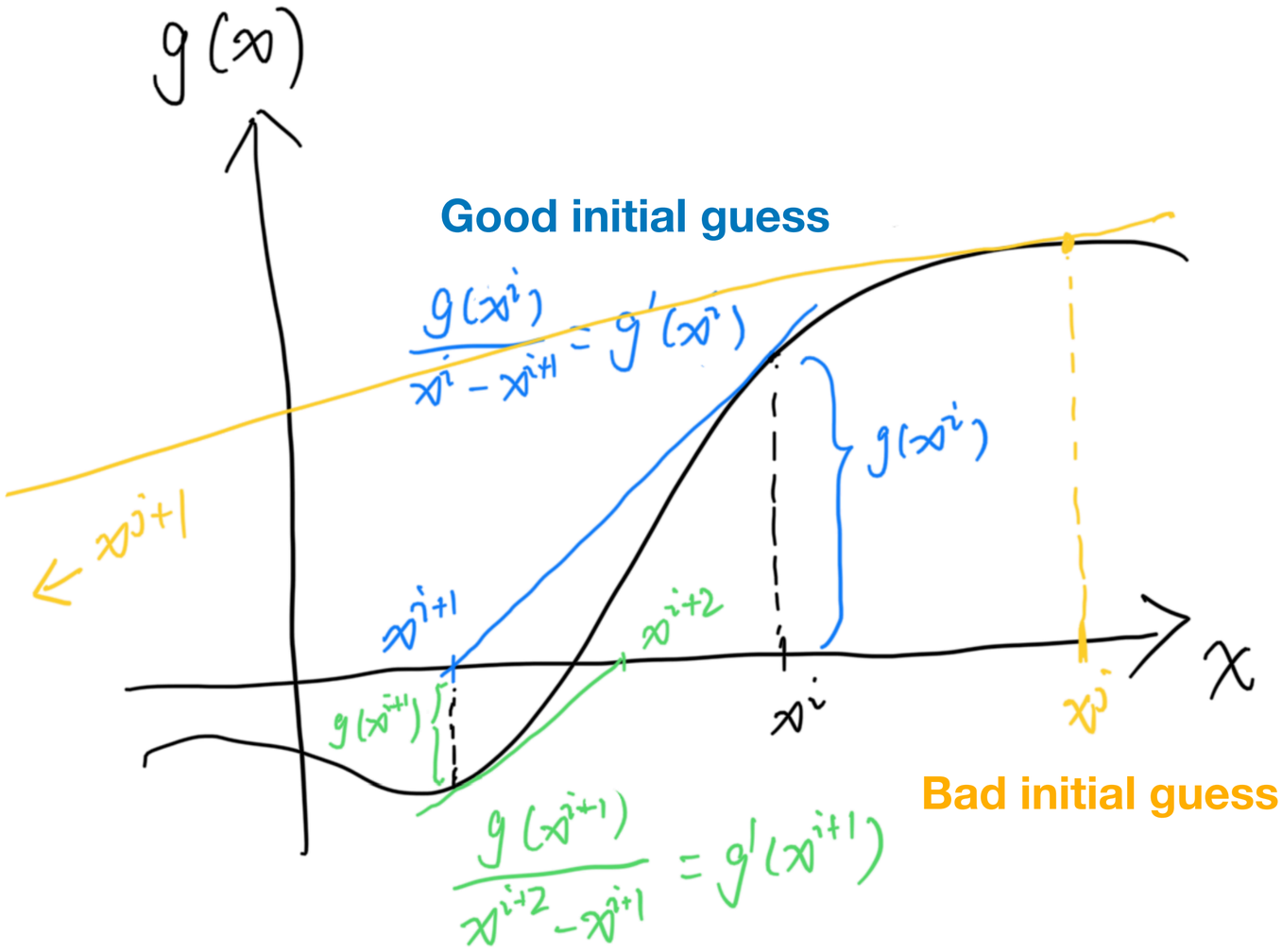
Linearly approximate $g(x) = 0$ **at** x^i **:**

$$g(x) = g(x^i) + \nabla g(x^i)(x - x^i)$$

$$g(x^{i+1}) \approx g(x^i) + \nabla g(x^i)(x^{i+1} - x^i) = 0$$

Convergence Issue of Newton's Method

Over-shooting



Simulation explodes!

Optimization Time Integration

$$x^{n+1} = \arg\min_x E(x)$$

$$\text{where } E(x) = \frac{1}{2}\|x - \tilde{x}^n\|_M^2 + \Delta t^2 P(x).$$

$$\tilde{x}^n = x^n + \Delta t v^n$$

$$\tfrac{1}{2}\|x - \tilde{x}^n\|_M^2 = \tfrac{1}{2}(x - \tilde{x}^n)^T M (x - \tilde{x}^n)$$

$$\tfrac{\partial P}{\partial x}(x) = -f(x)$$

At the local minimum of $E(x)$, $\tfrac{\partial E}{\partial x}(x^{n+1}) = 0$

$$M(x^{n+1} - (x^n + \Delta t v^n)) - \Delta t^2 f(x^{n+1}) = 0.$$

Global Convergence with Line Search

Pseudo-code

Algorithm 3: Projected Newton Method for Backward Euler Time Integration

Result: x^{n+1}, v^{n+1}

$x^i \leftarrow x^n$;
do
  $P \leftarrow \text{SPDProjection}(\nabla^2 E(x^i))$;
  $p \leftarrow -P^{-1}\nabla E(x^i)$;
  $\alpha \leftarrow \text{BackTrackingLineSearch}(x^i, p)$; //
  $x^i \leftarrow x^i + \alpha p$;
while $\|p\|_\infty / h > \epsilon$;
$x^{n+1} \leftarrow x^i$;
$v^{n+1} \leftarrow (x^{n+1} - x^n)/\Delta t$;

Algorithm 2: Backtracking Line Search

Result: α

$\alpha \leftarrow 1$;
while $E(x^i + \alpha p) > E(x^i)$ **do**
  $\alpha \leftarrow \alpha/2$;

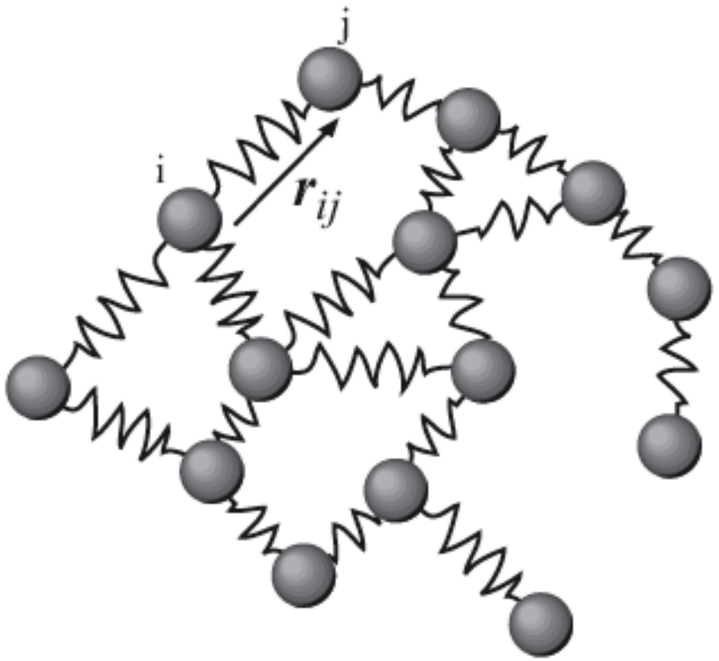
Topics Today

Part I: Fundamentals and The Big Picture (Minchen)

- Distortion Minimization
- Elastodynamics Simulation via Optimization Time Integration
- Case Study: Mass-Spring System
- More Topics on Optimization Time Integration

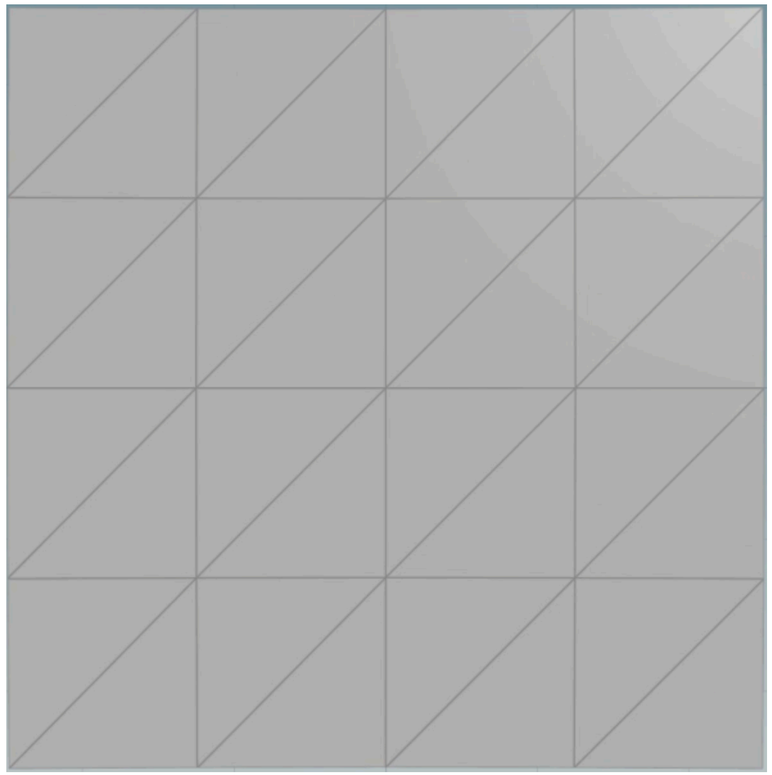
Part II: Advanced Topics (Eris)

- Constrained System
- Reduced-Order System
- Multilevel System
- Adaptive Simulation

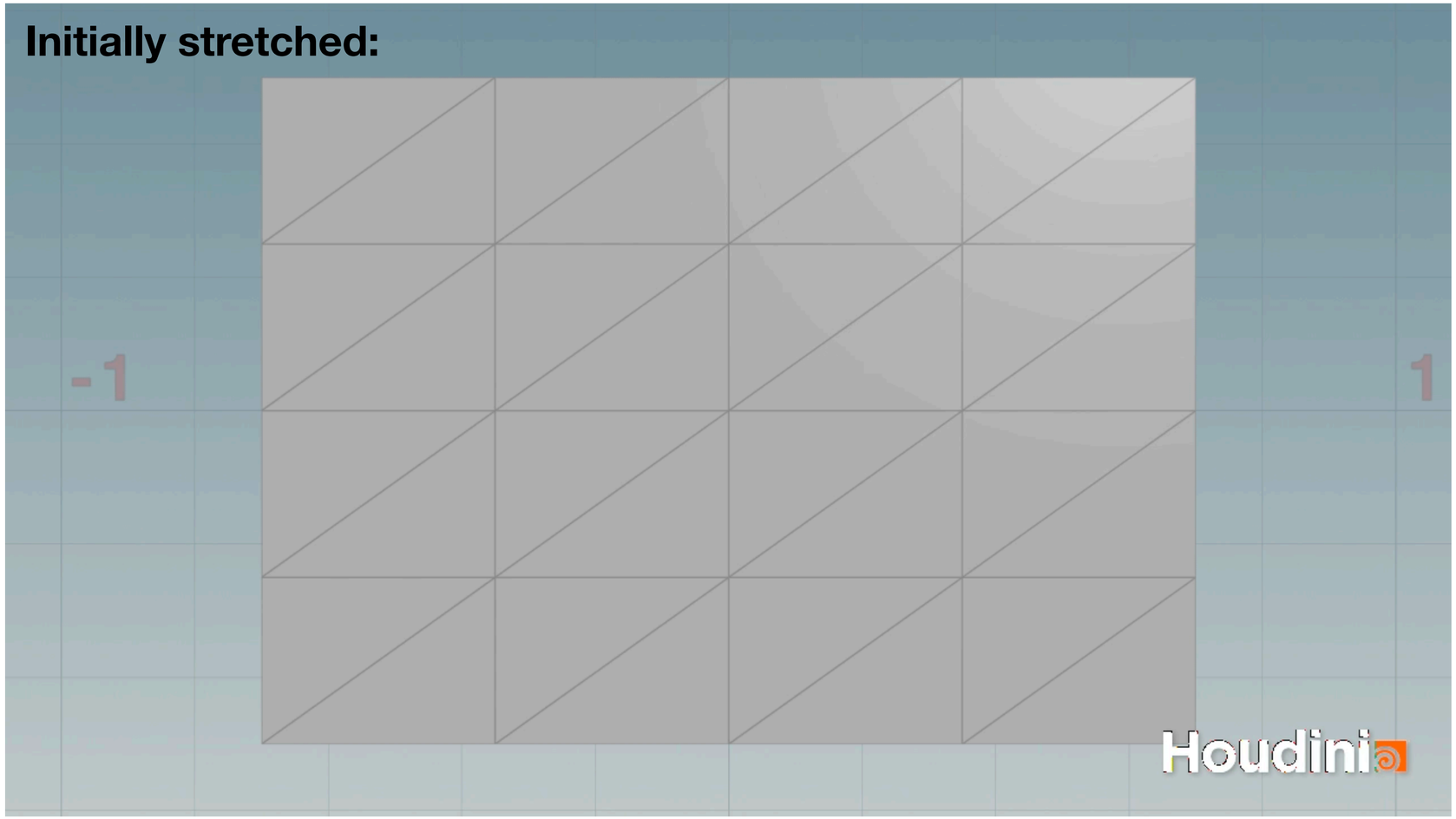


Case Study — Mass-Spring Simulation

An Initially Stretched Elastic Square

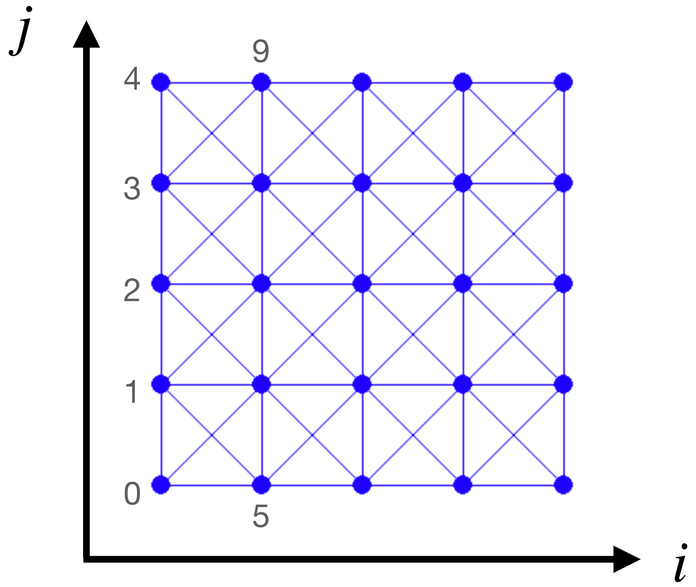


Rest shape



Mass-Spring Representation of Solids

- Mass particles connected by springs
  - square_mesh.py




```python
import numpy as np

def generate(side_length, n_seg):
    # sample nodes uniformly on a square
    x = np.array([[0.0, 0.0]] * ((n_seg + 1) ** 2))
    step = side_length / n_seg
    for i in range(0, n_seg + 1):
        for j in range(0, n_seg + 1):
            x[i * (n_seg + 1) + j] = [-side_length / 2 + i *
    step, -side_length / 2 + j * step]
```

```python
    # connect the nodes with edges
    e = []
    # horizontal edges
    for i in range(0, n_seg):
        for j in range(0, n_seg + 1):
            e.append([i * (n_seg + 1) + j, (i + 1) * (n_seg +
    1) + j])
```

```python
    # vertical edges
    for i in range(0, n_seg + 1):
        for j in range(0, n_seg):
            e.append([i * (n_seg + 1) + j, i * (n_seg + 1) + j
     + 1])
```

```python
    # diagonals
    for i in range(0, n_seg):
        for j in range(0, n_seg):
            e.append([i * (n_seg + 1) + j, (i + 1) * (n_seg +
    1) + j + 1])
            e.append([(i + 1) * (n_seg + 1) + j, i * (n_seg +
    1) + j + 1])

    return [x, e]
```

Time Integration

Optimization-based Implicit Euler

$$x^{n+1} = x^n + \Delta t v^{n+1},$$
$$v^{n+1} = v^n + \Delta t M^{-1} f^{n+1}$$

$\Longleftrightarrow$

Inertia term — Elasticity

$$E(x) = \frac{1}{2}\|x - (x^n + h v^n)\|_M^2 + h^2 P(x).$$

Incremental Potential

$$\frac{\partial P}{\partial x}(x) = -f(x)$$

Algorithm 3: Projected Newton Method for Backward Euler Time Integration

Result: x^{n+1}, v^{n+1}

$x^i \leftarrow x^n$;
do
  $P \leftarrow \text{SPDProjection}(\nabla^2 E(x^i))$; // Energy Hessian
  $p \leftarrow -P^{-1}\nabla E(x^i)$; // Energy Gradient
  $\alpha \leftarrow \text{BackTrackingLineSearch}(x^i, p)$;
  $x^i \leftarrow x^i + \alpha p$;
while $\|p\|_\infty / h > \epsilon$;
$x^{n+1} \leftarrow x^i$;
$v^{n+1} \leftarrow (x^{n+1} - x^n)/\Delta t$;

Algorithm 2: Backtracking Line Search

Result: α

$\alpha \leftarrow 1$;
while $E(x^i + \alpha p) > E(x^i)$ **do** // Energy Value
  $\alpha \leftarrow \alpha/2$;

Incremental Potential

Inertia Term

$$\text{with } \tilde{x}^n = x^n + hv^n$$

$$E_I(x) = \frac{1}{2}\|x - \tilde{x}^n\|_M^2$$

$$\nabla E_I(x) = M(x - \tilde{x}^n)$$

$$\nabla^2 E_I(x) = M \quad \text{– SPD}$$

InertiaEnergy.py

```python
import numpy as np

def val(x, x_tilde, m):
    sum = 0.0
    for i in range(0, len(x)):
        diff = x[i] - x_tilde[i]
        sum += 0.5 * m[i] * diff.dot(diff)
    return sum

def grad(x, x_tilde, m):
    g = np.array([[0.0, 0.0]] * len(x))
    for i in range(0, len(x)):
        g[i] = m[i] * (x[i] - x_tilde[i])
    return g

def hess(x, x_tilde, m):
    IJV = [[0] * (len(x) * 2), [0] * (len(x) * 2), np.array
    ([0.0] * (len(x) * 2))]
    for i in range(0, len(x)):
        for d in range(0, 2):
            IJV[0][i * 2 + d] = i * 2 + d
            IJV[1][i * 2 + d] = i * 2 + d
            IJV[2][i * 2 + d] = m[i]
    return IJV
```

Incremental Potential

Mass-Spring Elasticity Energy

- Hooke's Law in 1D:
  - $E = \frac{1}{2} k (\Delta x)^2$

Spring stiffness

Spring displacement

- In higher dimensions:

  - $\frac{1}{2} k (\|x_1 - x_2\| - l)^2$ or $l^2 \frac{1}{2} k (\frac{\|x_1 - x_2\|}{l} - 1)^2$

Current length

Rest length

A strain measure

- To avoid computing square root, we define

Area weighting

$$P_e(x) = l^2 \frac{1}{2} k (\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}{l^2} - 1)^2$$

Elasticity energy density
(elasticity energy per unit area)

Continuous setting:

$$P = \int_{\Omega^0} \Psi dX$$

Incremental Potential

Mass-Spring Elasticity Energy Gradient and Hessian

$$P_e(x) = l^2 \frac{1}{2} k (\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}{l^2} - 1)^2$$

$$\frac{\partial P_e}{\partial \boldsymbol{x}_1}(x) = -\frac{\partial P_e}{\partial \boldsymbol{x}_2}(x) = 2k(\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}{l^2} - 1)(\boldsymbol{x}_1 - \boldsymbol{x}_2)$$

$$\begin{aligned}
\frac{\partial^2 P_e}{\partial \boldsymbol{x}_1^2}(x) = \frac{\partial^2 P_e}{\partial \boldsymbol{x}_2^2}(x) = -\frac{\partial^2 P_e}{\partial \boldsymbol{x}_1 \boldsymbol{x}_2}(x) = -\frac{\partial^2 P_e}{\partial \boldsymbol{x}_2 \boldsymbol{x}_1}(x) \\
= \frac{4k}{l^2}(\boldsymbol{x}_1 - \boldsymbol{x}_2)(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T + 2k(\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}{l^2} - 1)\boldsymbol{I} \\
= \frac{2k}{l^2}(2(\boldsymbol{x}_1 - \boldsymbol{x}_2)(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T + (\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2 - l^2)\boldsymbol{I})
\end{aligned}$$

MassSpringEnergy.py

```python
import numpy as np
import utils

def val(x, e, l2, k):
    sum = 0.0
    for i in range(0, len(e)):
        diff = x[e[i][0]] - x[e[i][1]]
        sum += l2[i] * 0.5 * k[i] * (diff.dot(diff) / l2[i] - 1) ** 2
    return sum

def grad(x, e, l2, k):
    g = np.array([[0.0, 0.0]] * len(x))
    for i in range(0, len(e)):
        diff = x[e[i][0]] - x[e[i][1]]
        g_diff = 2 * k[i] * (diff.dot(diff) / l2[i] - 1) * diff
        g[e[i][0]] += g_diff
        g[e[i][1]] -= g_diff
    return g
```

Incremental Potential

Mass-Spring Elasticity Energy Hessian Implementation

$$\begin{aligned}\frac{\partial^2 P_e}{\partial \boldsymbol{x}_1^2}(x) &= \frac{\partial^2 P_e}{\partial \boldsymbol{x}_2^2}(x) = -\frac{\partial^2 P_e}{\partial \boldsymbol{x}_1 \boldsymbol{x}_2}(x) = -\frac{\partial^2 P_e}{\partial \boldsymbol{x}_2 \boldsymbol{x}_1}(x) \\ &= \frac{4k}{l^2}(\boldsymbol{x}_1 - \boldsymbol{x}_2)(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T + 2k(\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}{l^2} - 1)\boldsymbol{I} \\ &= \frac{2k}{l^2}(2(\boldsymbol{x}_1 - \boldsymbol{x}_2)(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T + (\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2 - l^2)\boldsymbol{I})\end{aligned}$$

MassSpringEnergy.py

```
20 def hess(x, e, l2, k):
21     IJV = [[0] * (len(e) * 16), [0] * (len(e) * 16), np.array
       ([0.0] * (len(e) * 16))]
22     for i in range(0, len(e)):
23         diff = x[e[i][0]] - x[e[i][1]]
24         H_diff = 2 * k[i] / l2[i] * (2 * np.outer(diff, diff)
       + (diff.dot(diff) - l2[i]) * np.identity(2))
25         H_local = utils.make_PD(np.block([[H_diff, -H_diff],
       [-H_diff, H_diff]]))
26         # add to global matrix
27         for nI in range(0, 2):
28             for nJ in range(0, 2):
29                 indStart = i * 16 + (nI * 2 + nJ) * 4
30                 for r in range(0, 2):
31                     for c in range(0, 2):
32                         IJV[0][indStart + r * 2 + c] = e[i][nI
       ] * 2 + r
33                         IJV[1][indStart + r * 2 + c] = e[i][nJ
       ] * 2 + c
34                         IJV[2][indStart + r * 2 + c] = H_local
       [nI * 2 + r, nJ * 2 + c]
35     return IJV
```

Incremental Potential

Mass-Spring Elasticity Energy Hessian Projection (make_PSD)

$$\min_P \|P - \nabla^2 E(x^i)\|_{\mathrm{F}} \quad s.t. \quad v^T P v \geq 0 \;\; \forall v \neq 0$$

Solution: $\hat{A} = Q\hat{\Lambda}Q^{-1}, \quad \hat{\Lambda}_{ij} = \Lambda_{ij} > 0 \; ? \; \Lambda_{ij} : 0$

Definition (Eigendecomposition). The eigendecomposition of a square matrix $A \in R^{n \times n}$ is

$$A = Q\Lambda Q^{-1}$$

where $Q = [q_1, q_2, ..., q_n]$ is composed of the eigenvectors q_i of A, $\|q_i\| = 1$; $\Lambda = [\lambda_1, \lambda_2, ..., \lambda_n]$, $\lambda_1 \geq \lambda_2 \geq ..., \lambda_n$ are the eigenvalues of A; and $Aq_i = \lambda_i q_i$.

utils.py

```python
import numpy as np
import numpy.linalg as LA

def make_PD(hess):
    [lam, V] = LA.eigh(hess)    # Eigen decomposition on symmetric matrix
    # set all negative Eigenvalues to 0
    for i in range(0, len(lam)):
        lam[i] = max(0, lam[i])
    return np.matmul(np.matmul(V, np.diag(lam)), np.transpose(V))
```

Incremental Potential

Gradient and Hessian

time_integrator.py

```
38 def IP_val(x, e, x_tilde, m, l2, k, h):
39     return InertiaEnergy.val(x, x_tilde, m) + h * h *
       MassSpringEnergy.val(x, e, l2, k)      # implicit Euler
40
41 def IP_grad(x, e, x_tilde, m, l2, k, h):
42     return InertiaEnergy.grad(x, x_tilde, m) + h * h *
       MassSpringEnergy.grad(x, e, l2, k)    # implicit Euler
43
44 def IP_hess(x, e, x_tilde, m, l2, k, h):
45     IJV_In = InertiaEnergy.hess(x, x_tilde, m)
46     IJV_MS = MassSpringEnergy.hess(x, e, l2, k)
47     IJV_MS[2] *= h * h     # implicit Euler
48     IJV = np.append(IJV_In, IJV_MS, axis=1)
49     H = sparse.coo_matrix((IJV[2], (IJV[0], IJV[1])), shape=(
       len(x) * 2, len(x) * 2)).tocsr()
50     return H
```

Time Integration

Algorithm 3: Projected Newton Method for Backward Euler Time Integration

Result: x^{n+1}, v^{n+1}

$x^i \leftarrow x^n$;
do
  $P \leftarrow \text{SPDProjection}(\nabla^2 E(x^i))$;
  $p \leftarrow -P^{-1}\nabla E(x^i)$;
  $\alpha \leftarrow \text{BackTrackingLineSearch}(x^i, p)$; //
  $x^i \leftarrow x^i + \alpha p$;
while $\|p\|_\infty / h > \epsilon$;
$x^{n+1} \leftarrow x^i$;
$v^{n+1} \leftarrow (x^{n+1} - x^n)/\Delta t$;

Algorithm 2: Backtracking Line Search

Result: α

$\alpha \leftarrow 1$;
while $E(x^i + \alpha p) > E(x^i)$ **do**
  $\alpha \leftarrow \alpha/2$;

time_integrator.py

```python
import copy
from cmath import inf

import numpy as np
import numpy.linalg as LA
import scipy.sparse as sparse
from scipy.sparse.linalg import spsolve

import InertiaEnergy
import MassSpringEnergy
```

```python
def step_forward(x, e, v, m, l2, k, h, tol):
    x_tilde = x + v * h     # implicit Euler predictive position
    x_n = copy.deepcopy(x)

    # Newton loop
    iter = 0
    E_last = IP_val(x, e, x_tilde, m, l2, k, h)
    p = search_dir(x, e, x_tilde, m, l2, k, h)
    while LA.norm(p, inf) / h > tol:
        print('Iteration', iter, ':')
        print('residual =', LA.norm(p, inf) / h)

        # line search
        alpha = 1
        while IP_val(x + alpha * p, e, x_tilde, m, l2, k, h) > E_last:
            alpha /= 2
        print('step size =', alpha)

        x += alpha * p
        E_last = IP_val(x, e, x_tilde, m, l2, k, h)
        p = search_dir(x, e, x_tilde, m, l2, k, h)
        iter += 1

    v = (x - x_n) / h   # implicit Euler velocity update
    return [x, v]
```

```python
def search_dir(x, e, x_tilde, m, l2, k, h):
    projected_hess = IP_hess(x, e, x_tilde, m, l2, k, h)
    reshaped_grad = IP_grad(x, e, x_tilde, m, l2, k, h).reshape(len(x) * 2, 1)
    return spsolve(projected_hess, -reshaped_grad).reshape(len(x), 2)
```

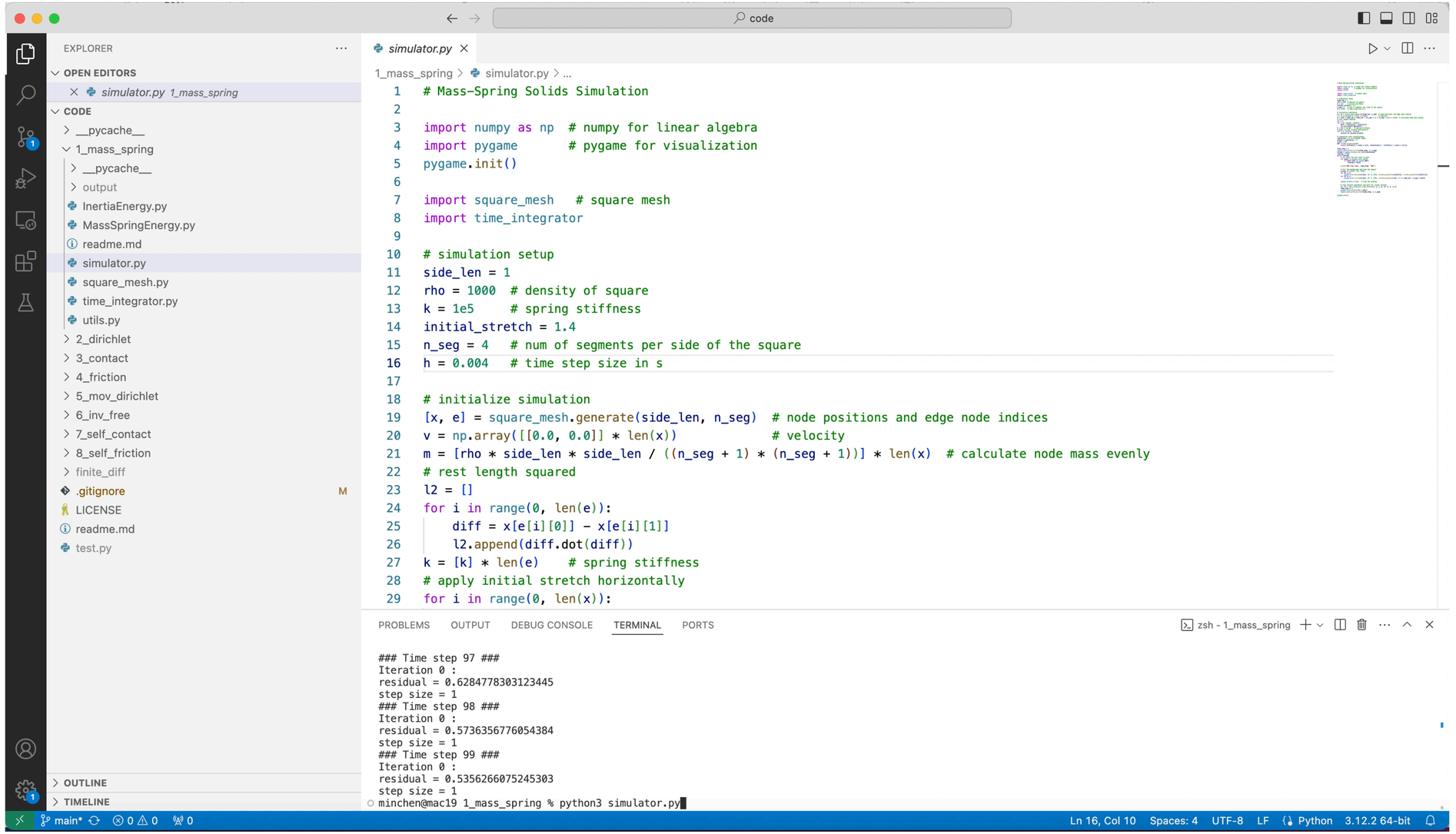
Simulator with Visualization

Simulator.py

```python
# Mass-Spring Solids Simulation

import numpy as np  # numpy for linear algebra
import pygame       # pygame for visualization
pygame.init()

import square_mesh   # square mesh
import time_integrator

# simulation setup
side_len = 1
rho = 1000  # density of square
k = 1e5     # spring stiffness
initial_stretch = 1.4
n_seg = 4   # num of segments per side of the square
h = 0.004   # time step size in s

# initialize simulation
[x, e] = square_mesh.generate(side_len, n_seg)  # node positions and edge node indices
v = np.array([[0.0, 0.0]] * len(x))             # velocity
m = [rho * side_len * side_len / ((n_seg + 1) * (n_seg + 1))] * len(x)  # calculate node mass evenly
# rest length squared
l2 = []
for i in range(0, len(e)):
    diff = x[e[i][0]] - x[e[i][1]]
    l2.append(diff.dot(diff))
k = [k] * len(e)    # spring stiffness
# apply initial stretch horizontally
for i in range(0, len(x)):
    x[i][0] *= initial_stretch
```

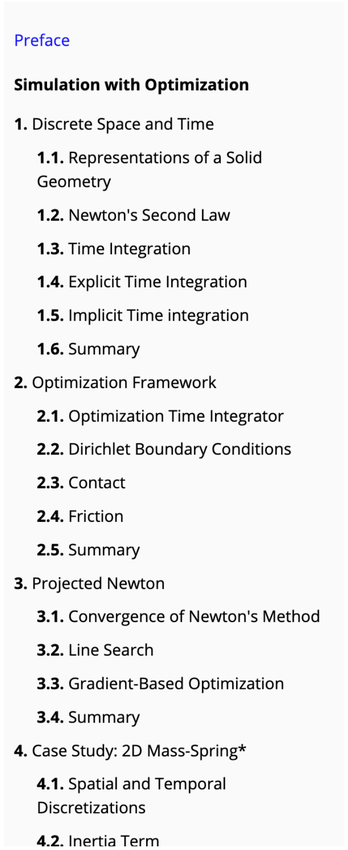
```python
# simulation with visualization
resolution = np.array([900, 900])
offset = resolution / 2
scale = 200
def screen_projection(x):
    return [offset[0] + scale * x[0], resolution[1] - (offset[1] + scale * x[1])]

time_step = 0
screen = pygame.display.set_mode(resolution)
running = True
while running:
    # run until the user asks to quit
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    print('### Time step', time_step, '###')

    # fill the background and draw the square
    screen.fill((255, 255, 255))
    for eI in e:
        pygame.draw.aaline(screen, (0, 0, 255), screen_projection(x[eI[0]]), screen_projection(x[eI[1]]))
    for xI in x:
        pygame.draw.circle(screen, (0, 0, 255), screen_projection(xI), 0.1 * side_len / n_seg * scale)

    pygame.display.flip()   # flip the display

    # step forward simulation and wait for screen refresh
    [x, v] = time_integrator.step_forward(x, e, v, m, l2, k, h, 1e-2)
    time_step += 1
    pygame.time.wait(int(h * 1000))

pygame.quit()
```

Demo


code
EXPLORER
OPEN EDITORS
simulator.py 1_mass_spring
CODE
__pycache__
1_mass_spring
__pycache__
output
InertiaEnergy.py
MassSpringEnergy.py
readme.md
simulator.py
square_mesh.py
time_integrator.py
utils.py
2_dirichlet
3_contact
4_friction
5_mov_dirichlet
6_inv_free
7_self_contact
8_self_friction
finite_diff
.gitignore
M
LICENSE
readme.md
test.py
OUTLINE
TIMELINE
simulator.py
1_mass_spring > simulator.py > ...
1 # Mass-Spring Solids Simulation
2
3 import numpy as np # numpy for linear algebra
4 import pygame # pygame for visualization
5 pygame.init()
6
7 import square_mesh # square mesh
8 import time_integrator
9
10 # simulation setup
11 side_len = 1
12 rho = 1000 # density of square
13 k = 1e5 # spring stiffness
14 initial_stretch = 1.4
15 n_seg = 4 # num of segments per side of the square
16 h = 0.004 # time step size in s
17
18 # initialize simulation
19 [x, e] = square_mesh.generate(side_len, n_seg) # node positions and edge node indices
20 v = np.array([[0.0, 0.0]] * len(x)) # velocity
21 m = [rho * side_len * side_len / ((n_seg + 1) * (n_seg + 1))] * len(x) # calculate node mass evenly
22 # rest length squared
23 l2 = []
24 for i in range(0, len(e)):
25 diff = x[e[i][0]] - x[e[i][1]]
26 l2.append(diff.dot(diff))
27 k = [k] * len(e) # spring stiffness
28 # apply initial stretch horizontally
29 for i in range(0, len(x)):
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
zsh - 1_mass_spring
Time step 97
Iteration 0 :
residual = 0.6284778303123445
step size = 1
Time step 98
Iteration 0 :
residual = 0.5736356776054384
step size = 1
Time step 99
Iteration 0 :
residual = 0.5356266075245303
step size = 1
minchen@mac19 1_mass_spring % python3 simulator.py
main*
0 0 0
Ln 16, Col 10
Spaces: 4
UTF-8
LF
Python
3.12.2 64-bit

Free Online Book and Python Tutorial

Code: github.com/phys-sim-book/solid-sim-tutorial



Preface

Simulation with Optimization

1. Discrete Space and Time

1.1. Representations of a Solid Geometry

1.2. Newton's Second Law

1.3. Time Integration

1.4. Explicit Time Integration

1.5. Implicit Time integration

1.6. Summary

2. Optimization Framework

2.1. Optimization Time Integrator

2.2. Dirichlet Boundary Conditions

2.3. Contact

2.4. Friction

2.5. Summary

3. Projected Newton

3.1. Convergence of Newton's Method

3.2. Line Search

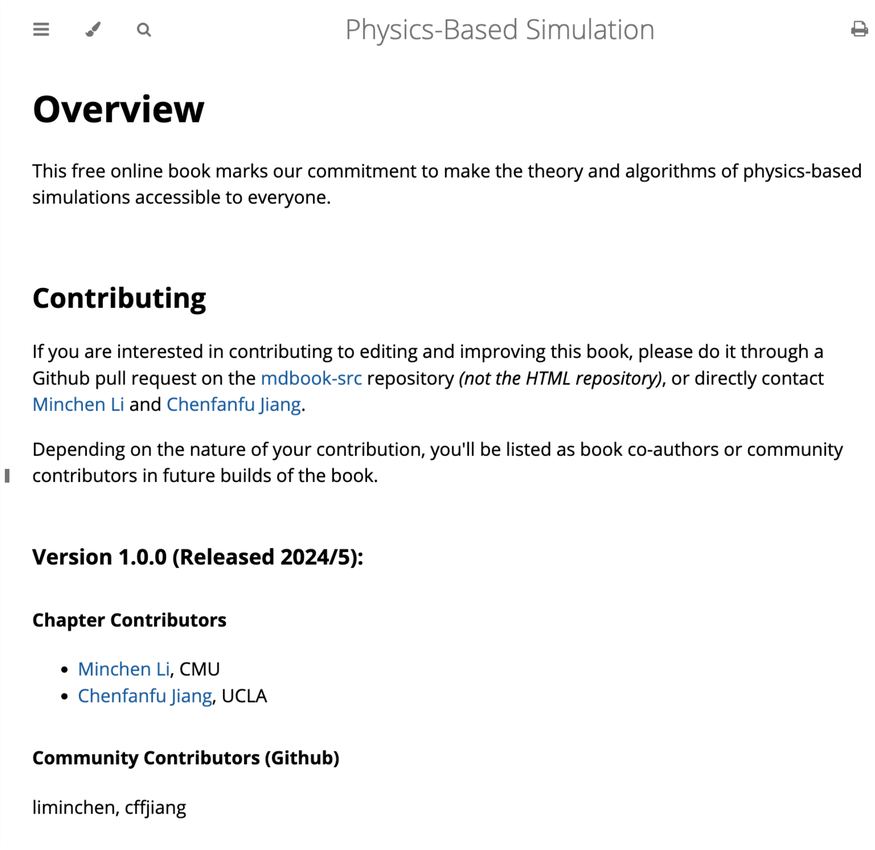
3.3. Gradient-Based Optimization

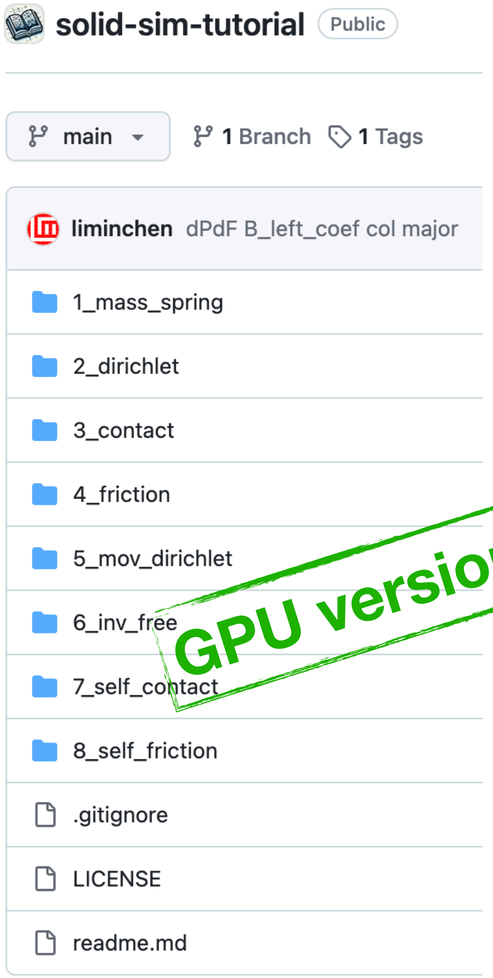
3.4. Summary

4. Case Study: 2D Mass-Spring*

4.1. Spatial and Temporal Discretizations

4.2. Inertia Term





GPU version coming!

Free online book: phys-sim-book.github.io

Topics Today

Part I: Fundamentals and The Big Picture (Minchen)

- Distortion Minimization
- Elastodynamics Simulation via Optimization Time Integration
- Case Study: Mass-Spring System
- More Topics on Optimization Time Integration

Part II: Advanced Topics (Eris)

- Constrained System
- Reduced-Order System
- Multilevel System
- Adaptive Simulation

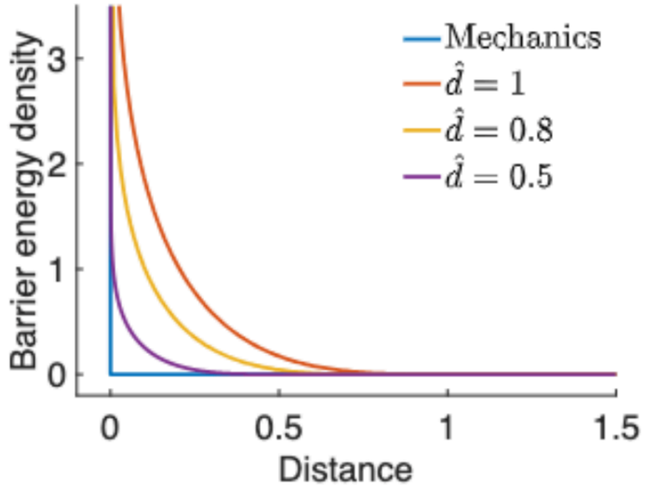
Self-Contact (Briefly)

$$x^{n+1} = arg\min_{x} E(x)$$
$$s.t. \quad \forall ij,\ d_{ij}(x) > 0$$

ij iterates non-adjacent surface primitive pairs

$$x^{n+1} = arg\min_{x} E(x) + \Delta t^2 \sum_{ij} w_{ij} b(d_{ij}(x))$$

$$b(d_{ij}(x)) = \begin{cases} \frac{\kappa}{2}\left(\frac{d_{ij}}{\hat{d}} - 1\right)\ln\frac{d_{ij}}{\hat{d}} & d_{ij} < \hat{d} \\ 0 & d_{ij} \geq \hat{d} \end{cases}$$



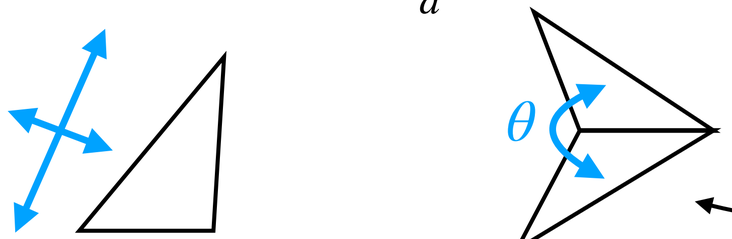
[Li et al. 2020]

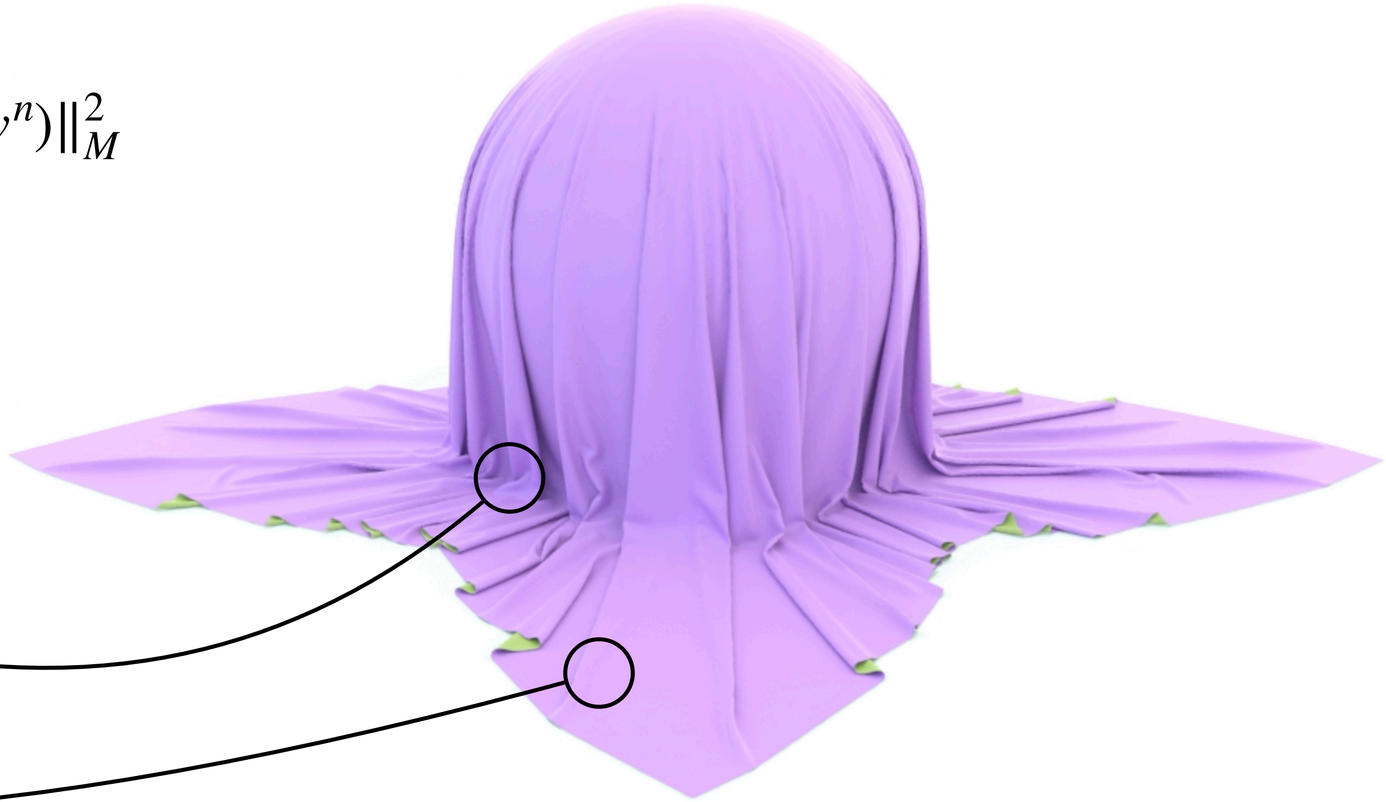
Thin Objects (Briefly)

$$x^{n+1} = \arg\min_x E(x)$$

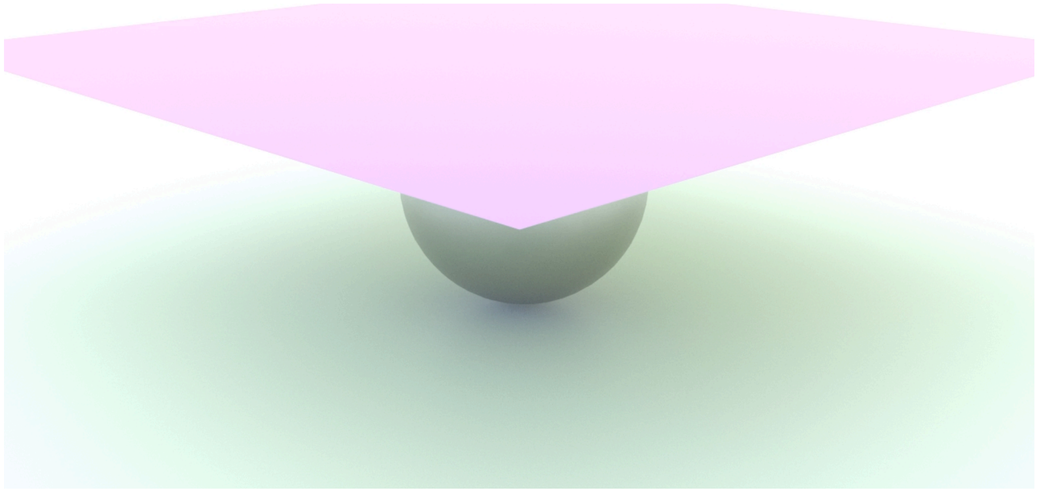
$$E(x) = \sum_e V_e \Psi(\mathbf{F}_e) + \frac{1}{2\Delta t^2}\|x - (x^n + \Delta t v^n)\|_M^2$$

$$\sum_e V_e \Psi_{\textbf{stretch}}(\mathbf{F}_e) + \sum_d w_d \Psi_{\textbf{bend}}(\theta_d)$$





Thin Objects

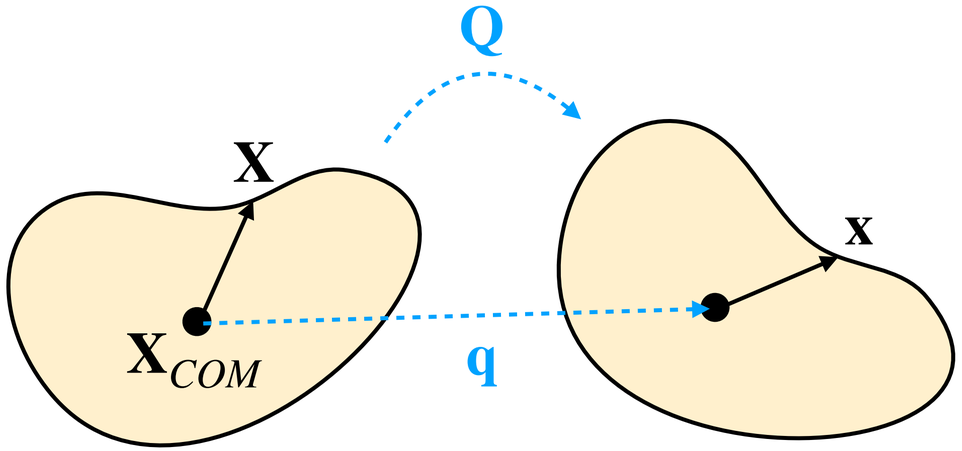


[Li et al. 2021]

Nearly Rigid Effects (Briefly)

If don't care about the tiny deformations,

Can simply track rotation\affine transformation $\mathbf{Q}$ and translation $\mathbf{q}$ per body!



$$\mathbf{x} = \mathbf{Q}(\mathbf{X} - \mathbf{X}_{COM}) + \mathbf{X}_{COM} + \mathbf{q}$$

Constant deformation gradient per body,

$$x^{n+1} = arg\min_{x} E(x)$$

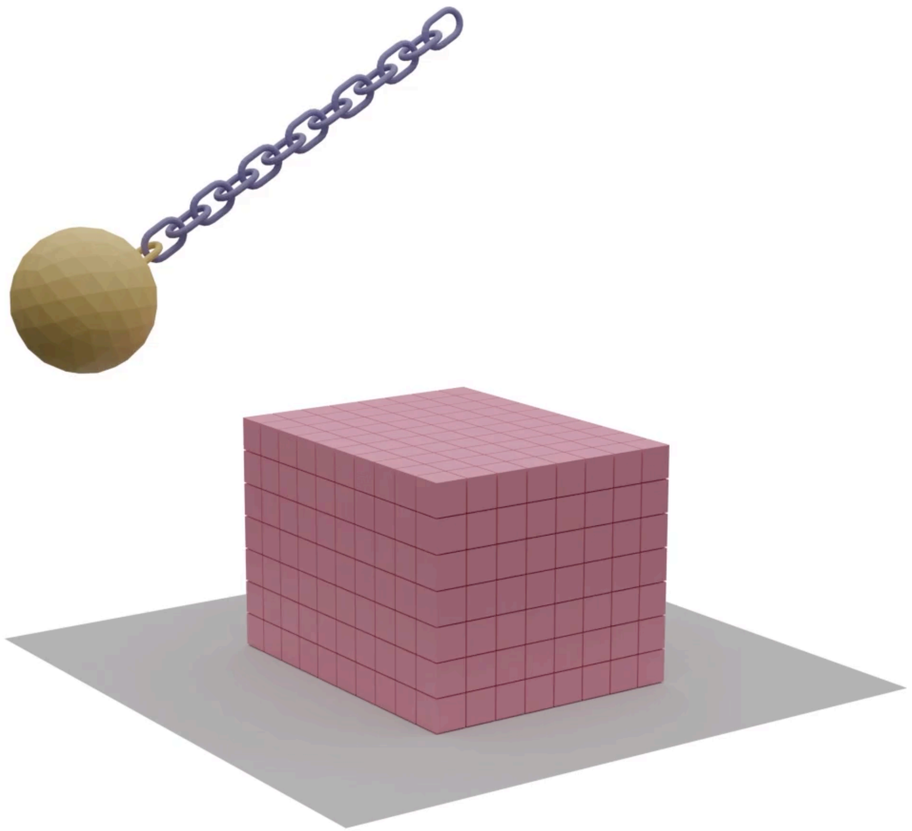
$$E(x) = \sum_{e} V_e \Psi(\mathbf{F}_e) + \frac{1}{2\Delta t^2}\|x - (x^n + \Delta t v^n)\|_M^2$$

Minimize in the subspace
(Let $x = \phi(Q, q)$)

$$Q^{n+1}, q^{n+1} = arg\min_{Q,q} E(\phi(Q, q))$$

$$E(\phi(Q, q)) = \sum_{e} V_e \Psi(\mathbf{F}_e(Q, q)) + \frac{1}{2\Delta t^2}\|\phi(Q, q) - (x^n + \Delta t v^n)\|_M^2$$

Nearly Rigid Objects



[Ferguson et al. 2021]

[Lan et al. 2022]

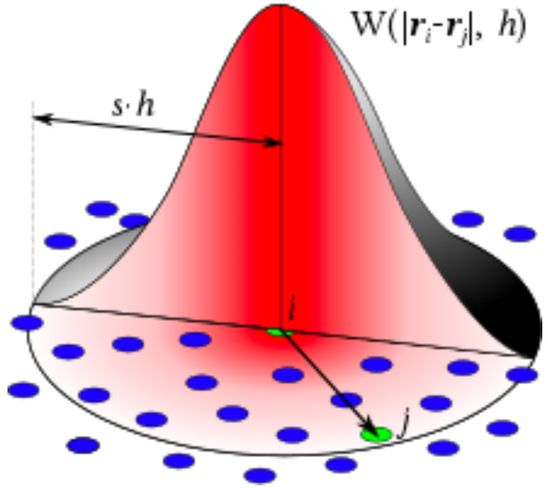
Fluids (Briefly)

$$x^{n+1} = arg\min_{x} E(x)$$

$$E(x) = \sum_{e} V_e \Psi(\mathbf{F}_e) + \frac{1}{2\Delta t^2}\|x - (x^n + \Delta t v^n)\|_M^2$$

$$\Psi(\mathbf{F}_e) = \mu(\det(\mathbf{F}_e) - 1)^2$$

No penalty on shearing,
only penalizes volume change



Use particles and neighbors to compute $\mathbf{F}$
via e.g. SPH, MLS

Fluids

[Xie et al. 2023]

Conclusion and Future Works

Optimization time integration

enables reliable and versatile physics-based simulation

$$x^{n+1} = arg\min_{x} E(x)$$

$$E(x) = \sum_{e} V_e \Psi(\mathbf{F}_e) + \frac{1}{2\Delta t^2}\|x - (x^n + \Delta t v^n)\|_M^2$$

Improving efficiency and scalability;

Incorporating path-dependent effects, e.g. damping, plasticity, friction, etc;

Application to inverse problems, e.g. computational design, robot learning, etc.

Topics Today

Part I: Fundamentals and The Big Picture (Minchen)

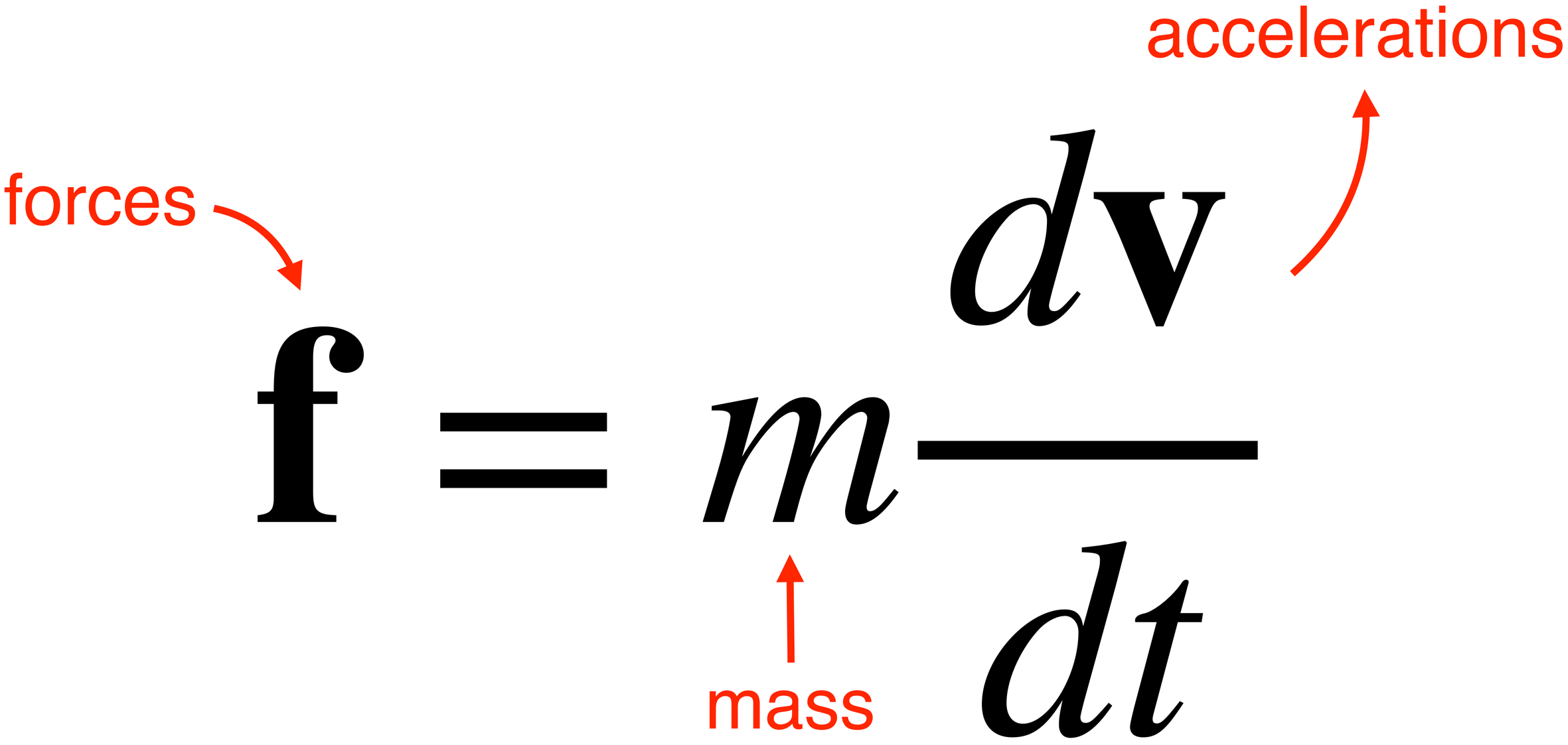
- Distortion Minimization
- Elastodynamics Simulation via Optimization Time Integration
- Case Study: Mass-Spring System
- More Topics on Optimization Time Integration

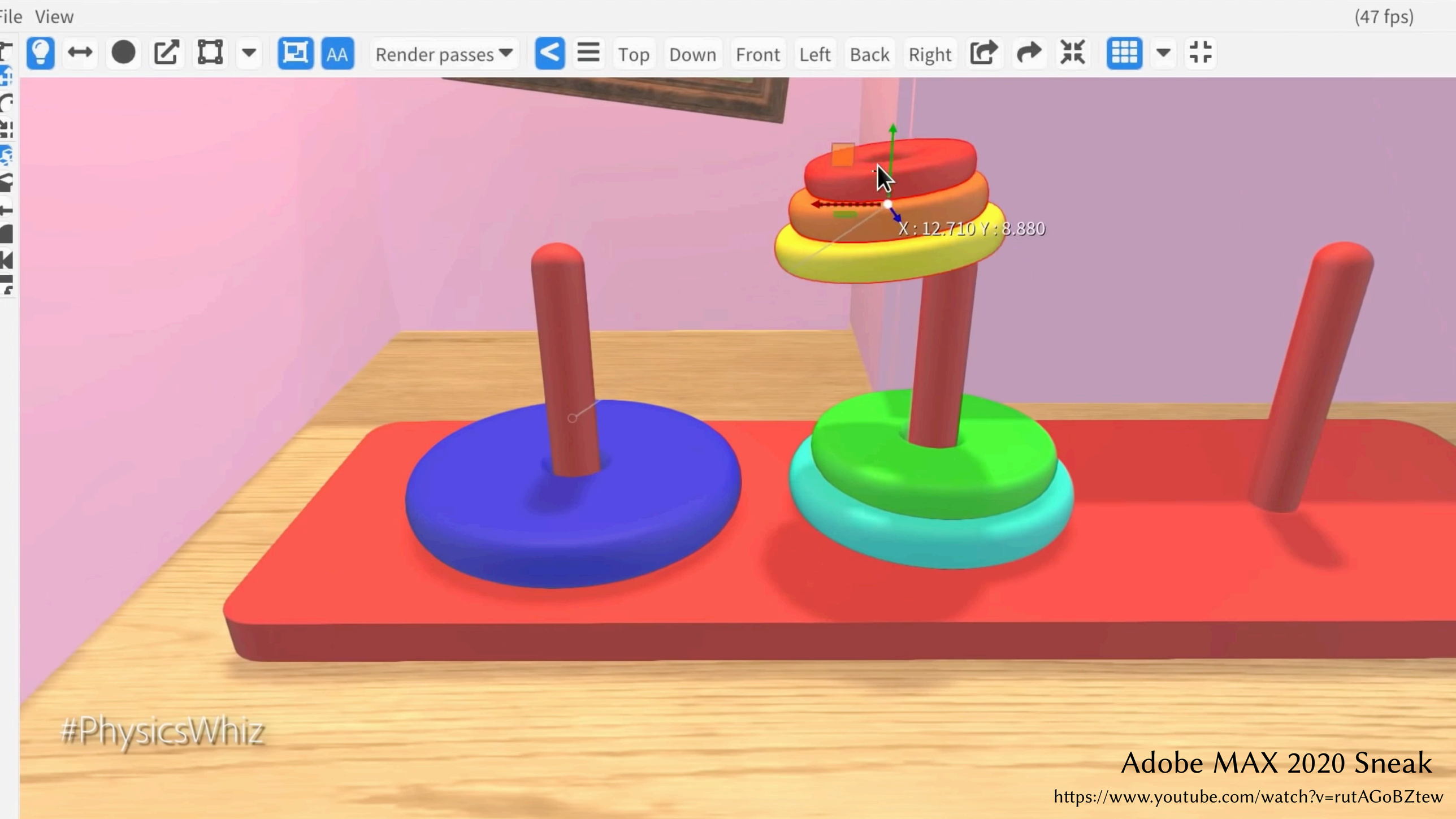
Part II: Advanced Topics (Eris)

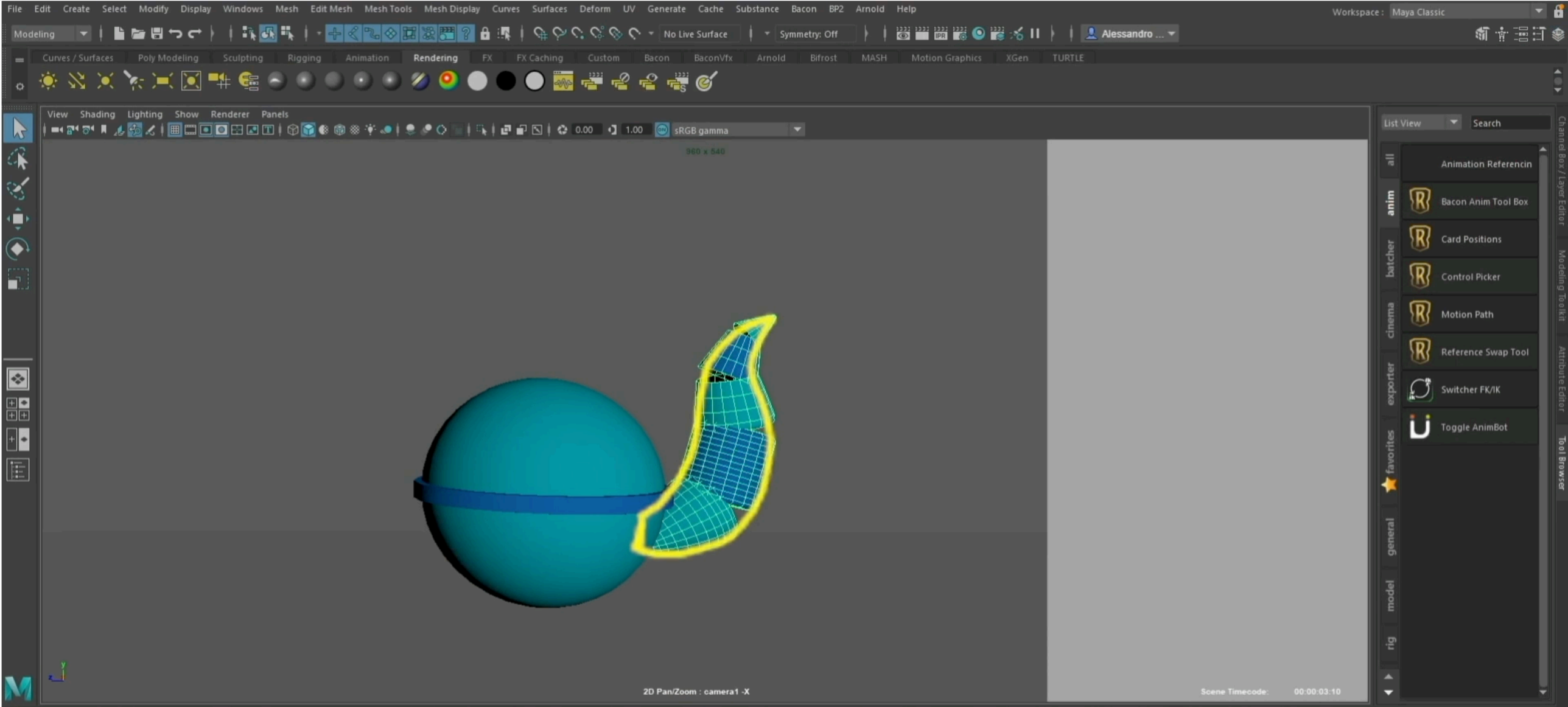
- Constrained System
- Reduced-Order System
- Multilevel System
- Adaptive Simulation

Introduction to Optimization Time Integration for Solids and Fluids (Part II)

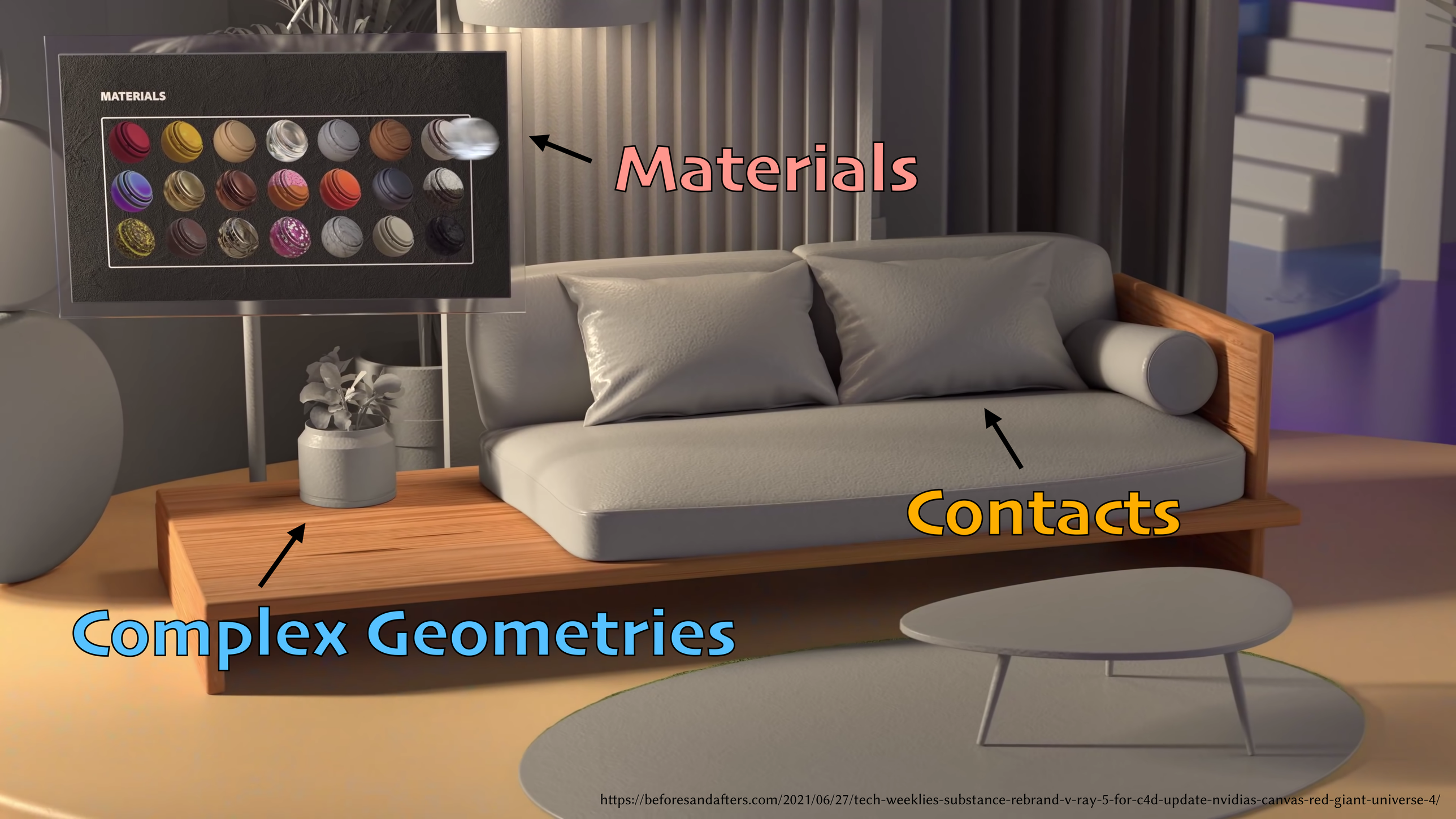
Minchen Li & **Jiayi Eris Zhang**


accelerations
forces
$\mathbf{f} = m\frac{d\mathbf{v}}{dt}$
mass


File View
(47 fps)
AA
Render passes
Top
Down
Front
Left
Back
Right
X : 12.710 Y : 8.880
#PhysicsWhiz
Adobe MAX 2020 Sneak
https://www.youtube.com/watch?v=rutAGoBZtew

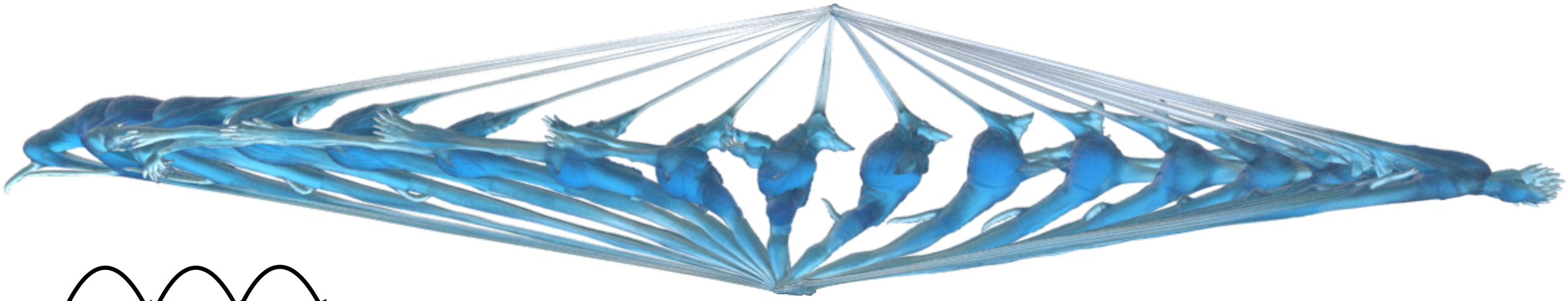


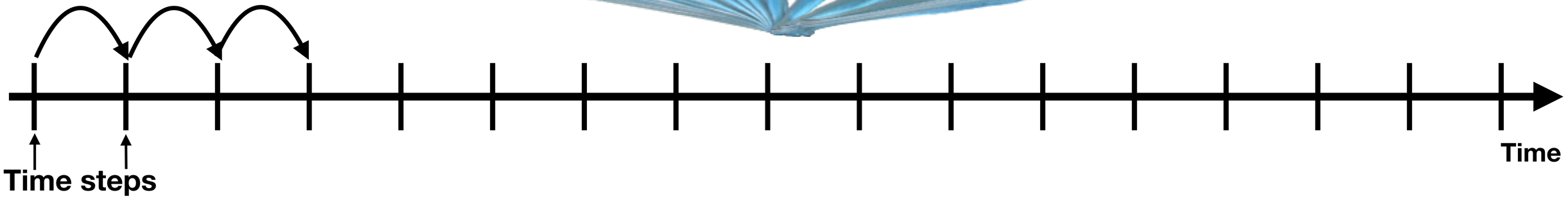
https://www.youtube.com/watch?v=0fnOBuTG6kU



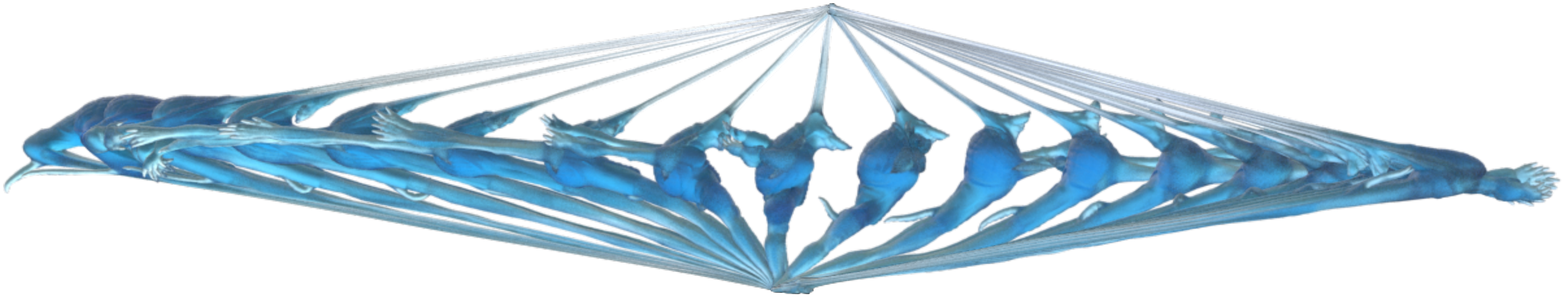
https://beforesandafters.com/2021/06/27/tech-weeklies-substance-rebrand-v-ray-5-for-c4d-update-nvidias-canvas-red-giant-universe-4/

Time Stepping





Li et al, SIGGRAPH 2019



Li et al, SIGGRAPH 2019

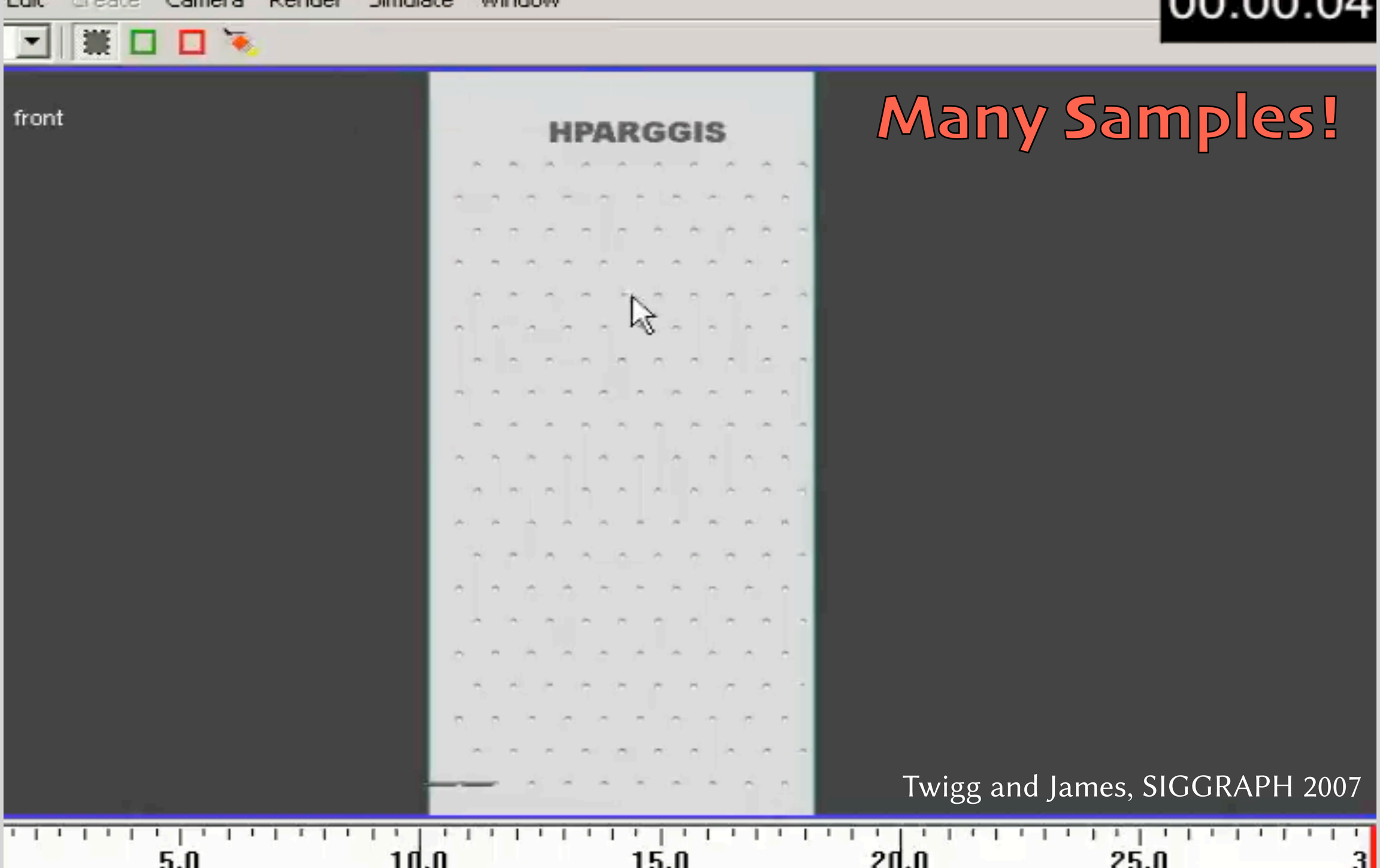
Time Stepping

Li et al, SIGGRAPH 2019

Time Stepping

Direct Simulation?

Li et al, SIGGRAPH 2019


front
HPARGGIS
Many Samples!
Twigg and James, SIGGRAPH 2007

Art-directable Simulation!

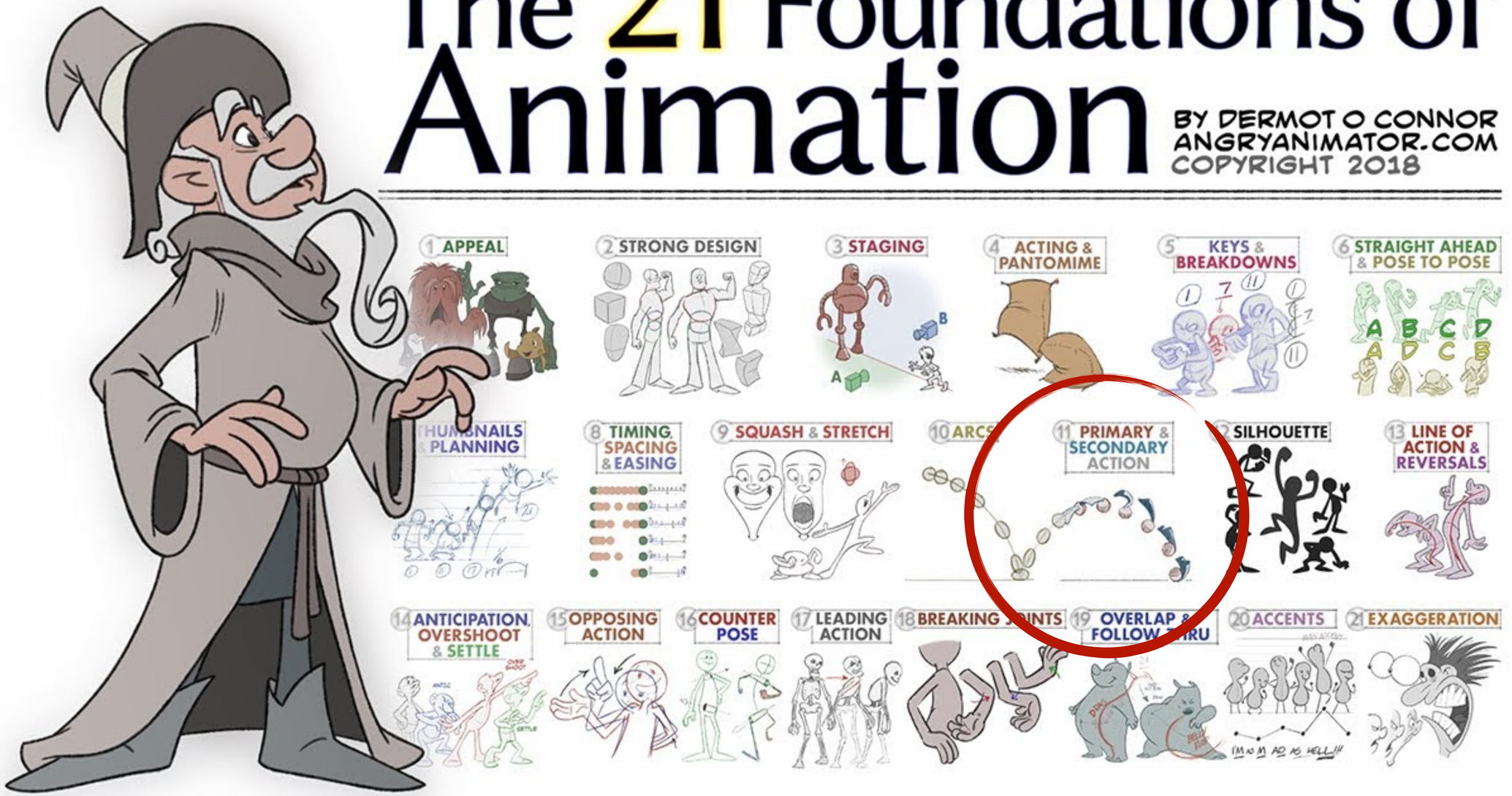
Rigging!
Control!
...

Physics!
Automation!
...

F=ma

The 21 Foundations of Animation

BY DERMOT O CONNOR
ANGRYANIMATOR.COM
COPYRIGHT 2018

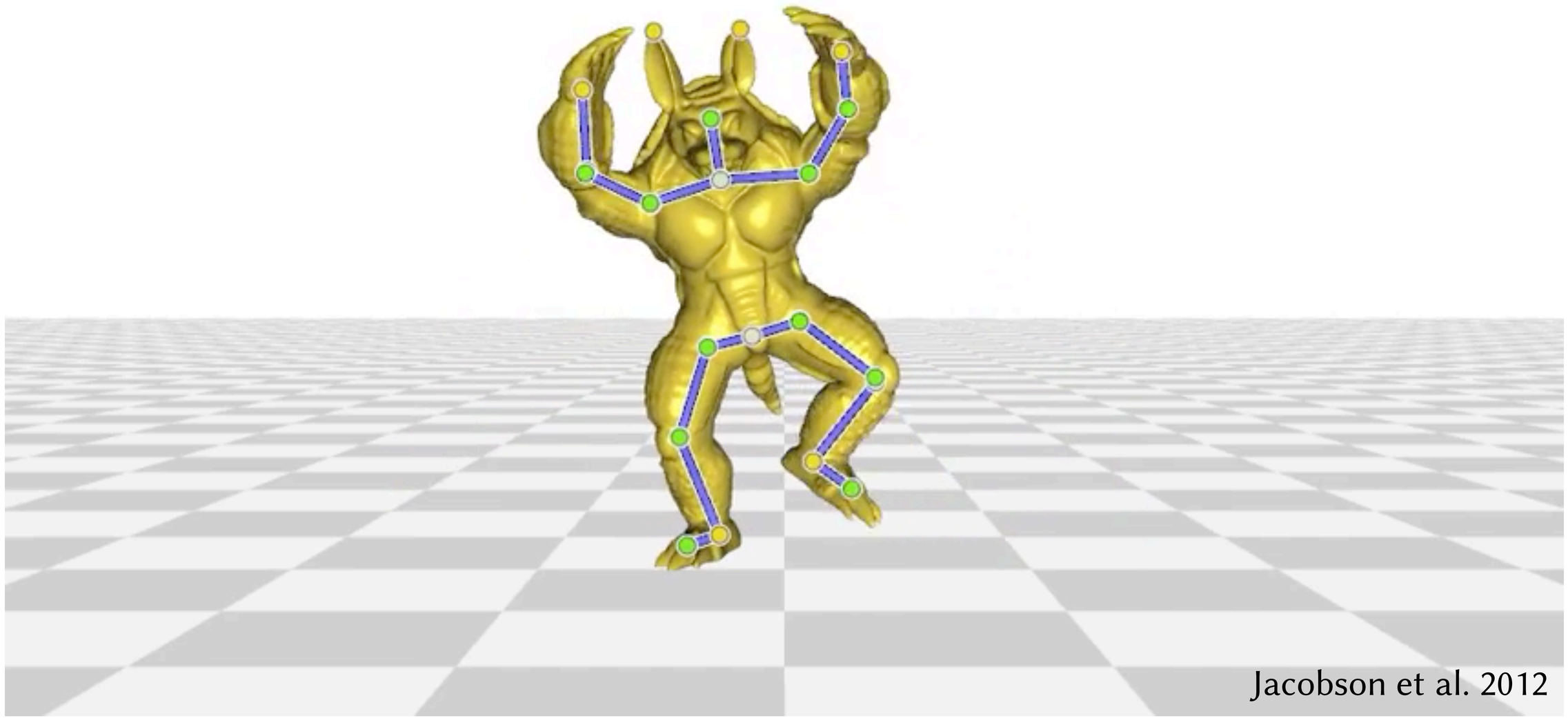


With Secondary Motion

Primary Motion Only

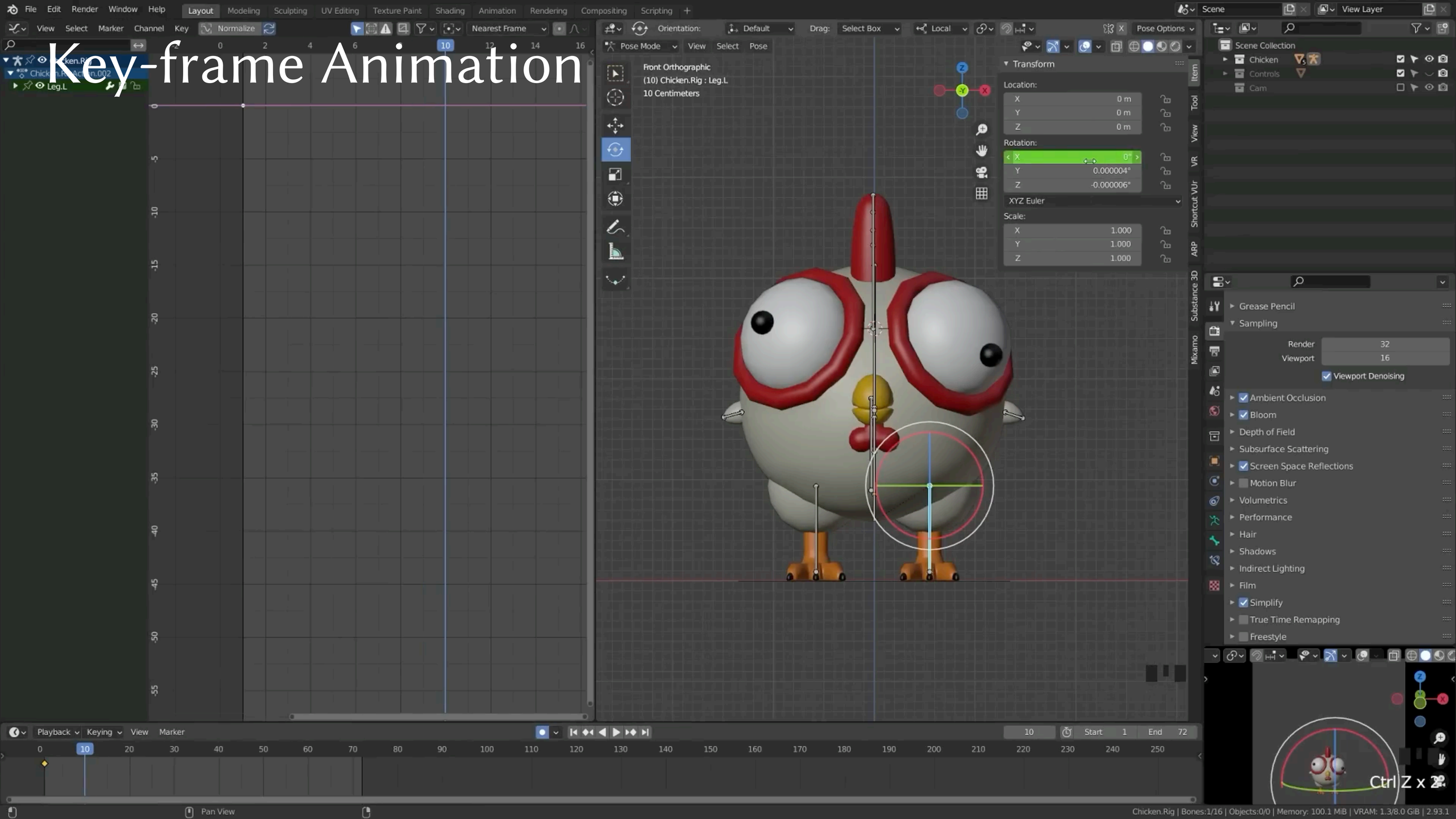
https://www.youtube.com/watch?v=4OxphYV8W3E&t=1s

Skinning



Jacobson et al. 2012

Key-frame Animation

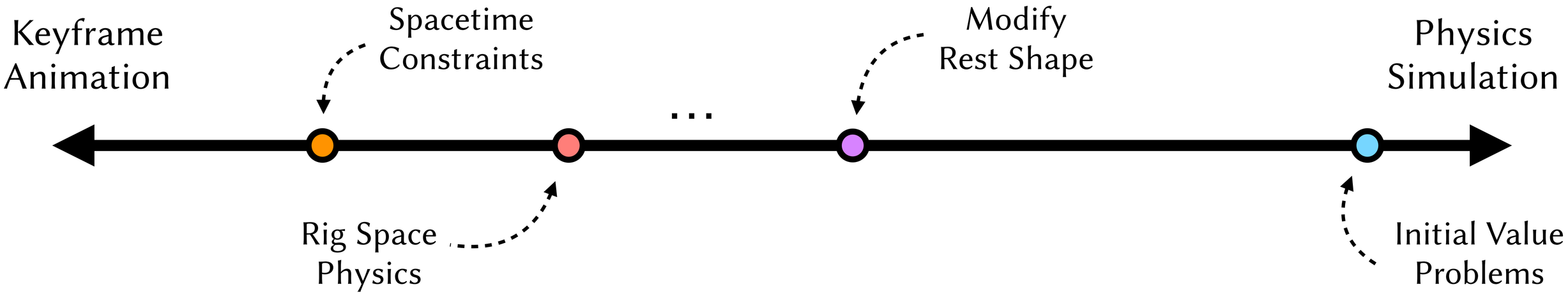


The *False* Linear Spectrum

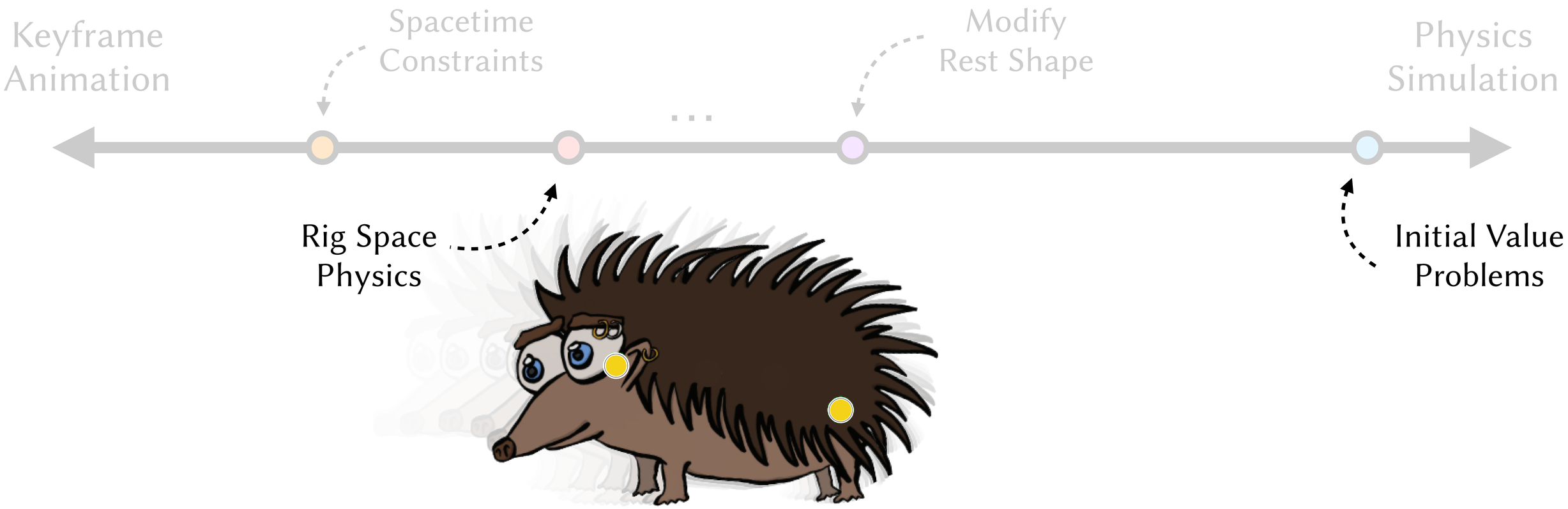
Keyframe Animation

Physics Simulation

The *False* Linear Spectrum



The *False* Linear Spectrum



The *False* Linear Spectrum

Keyframe Animation

Spacetime Constraints

Modify Rest Shape

Physics Simulation

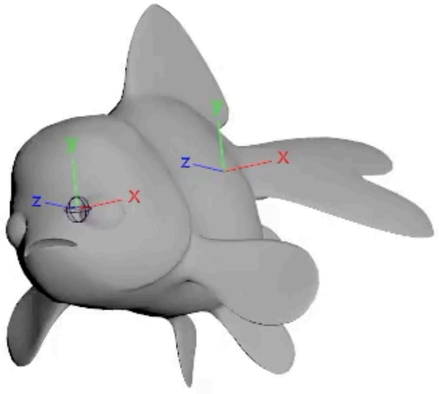
...

Rig Space Physics

Initial Value Problems

“Creative primary effects and physical secondary effects are not contradictory, but rather they are *complementary*”

Primary Motion + Our Complementary Secondary Motion

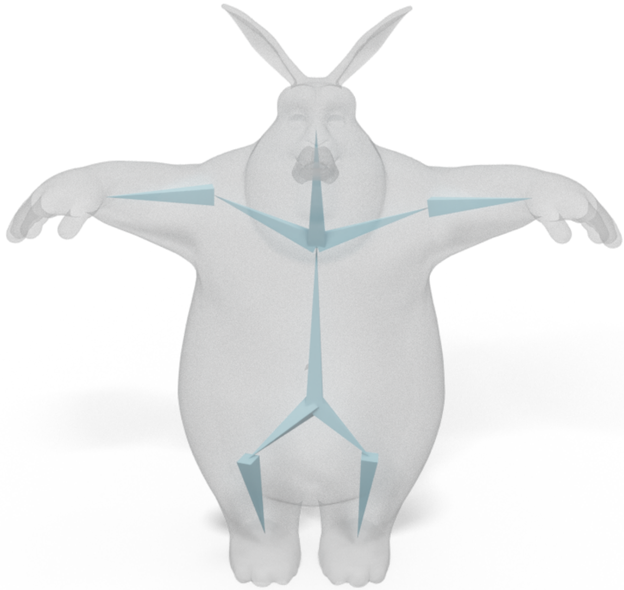


Input Rig Function

$\mathbf{u^r}$:

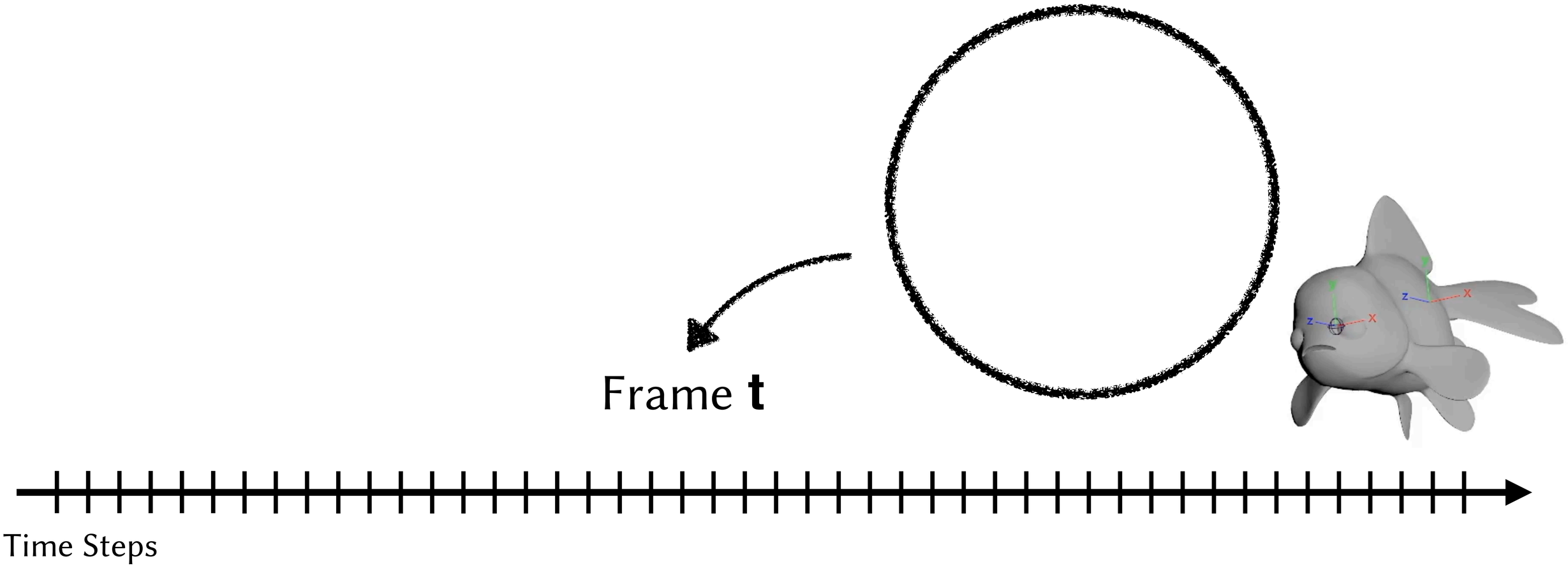
Any Rig Type

p_t

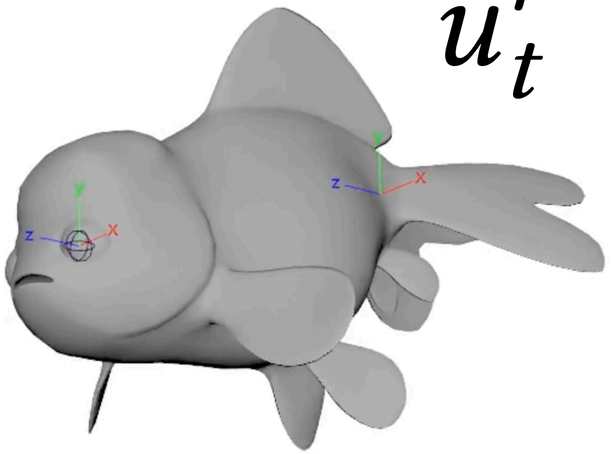


u_t^r

Input Rig Parameters Sequence



u_t = u_t^r + u_t^c



Final Displacement

Rig Displacement

Complementary Displacement

Time Integration

Any Physical Model

$$E(u_t) = \frac{1}{2}\|u_t - \tilde{u}\|_M^2 + h^2\Psi(u_t)$$

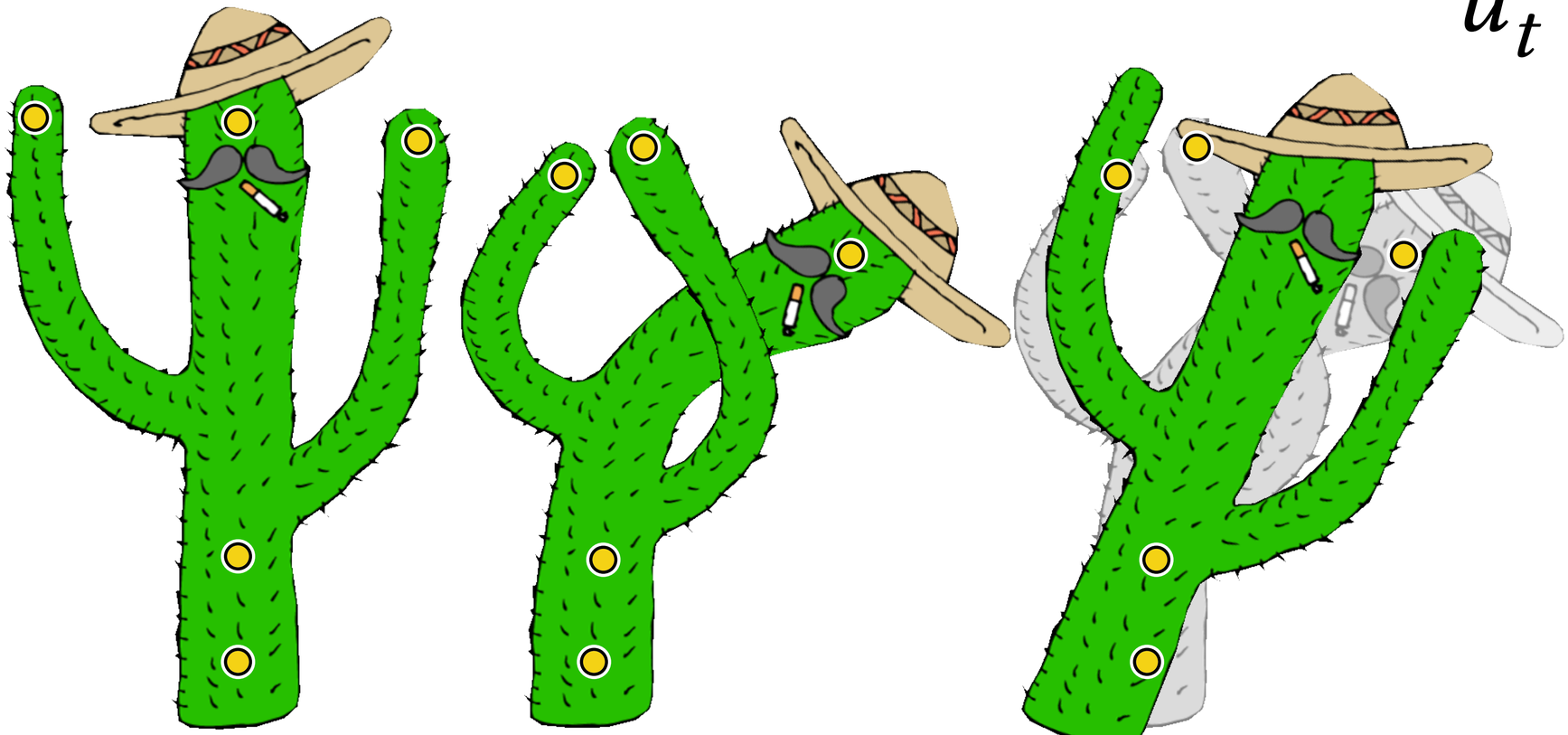
Inertia Term

Elasticity Potential

$$u_t^c = \underset{u_t^c}{\operatorname{argmin}}\ E_t(u_t^r + u_t^c)$$

Unconstrained

$$u_t^c = -u_t^r$$

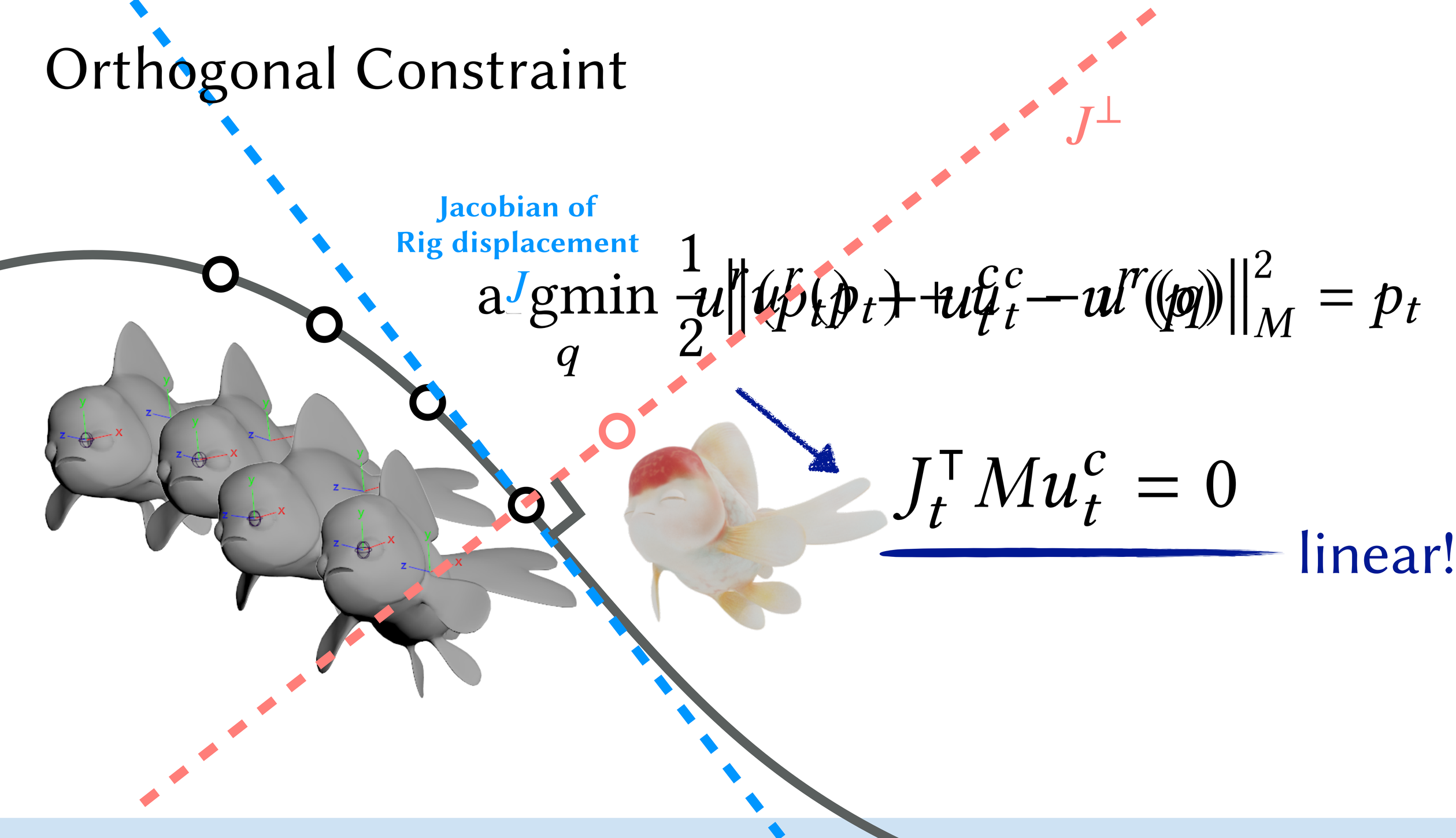


rest pose

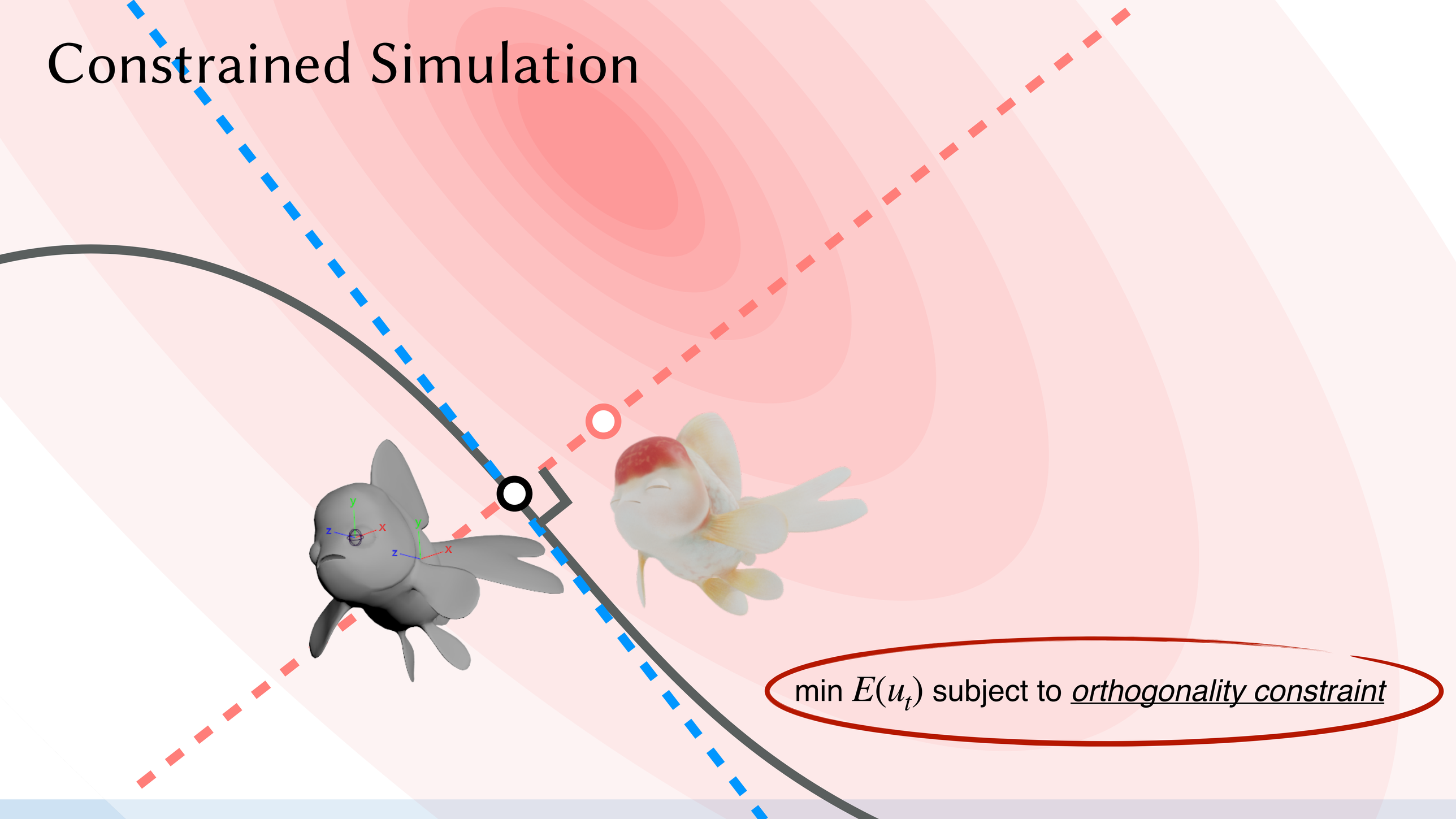
rig pose

physics *undoes* rig pose

Orthogonal Constraint



Constrained Simulation

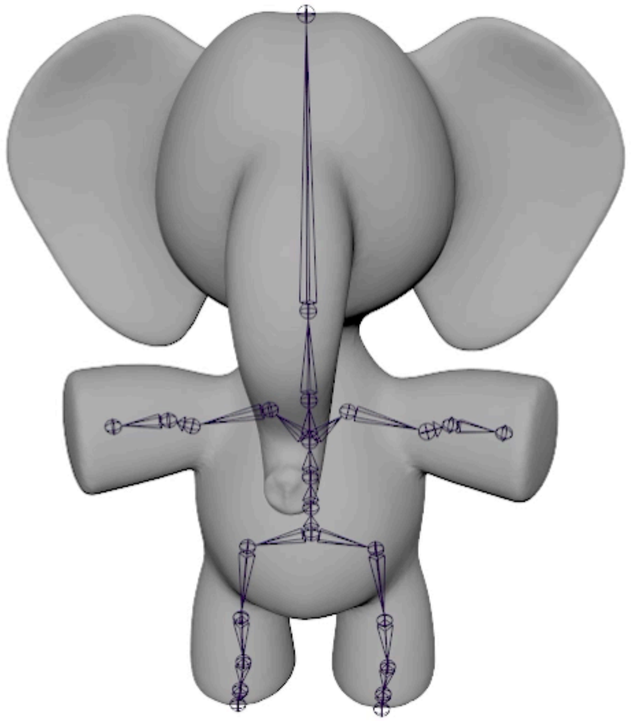


min $E(u_t)$ subject to *orthogonality constraint*

Complementary Dynamics [Zhang et al. 2020]

$$u_t^c = \underset{u_t^c}{\operatorname{argmin}} \; E_t(u_t^r + u_t^c)$$

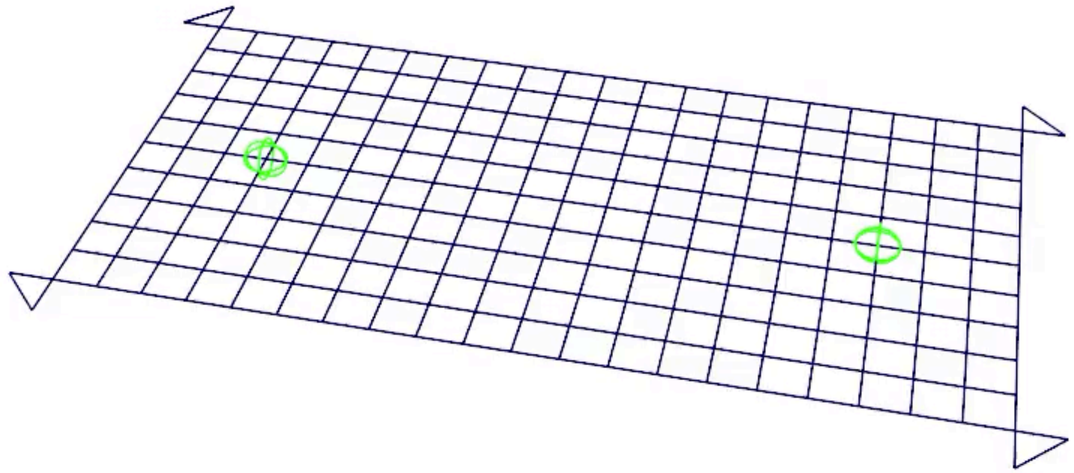
subject to $J_t^\top M u_t^c = 0$



Input

Output

Cloth Simulation (Mass-Spring System)

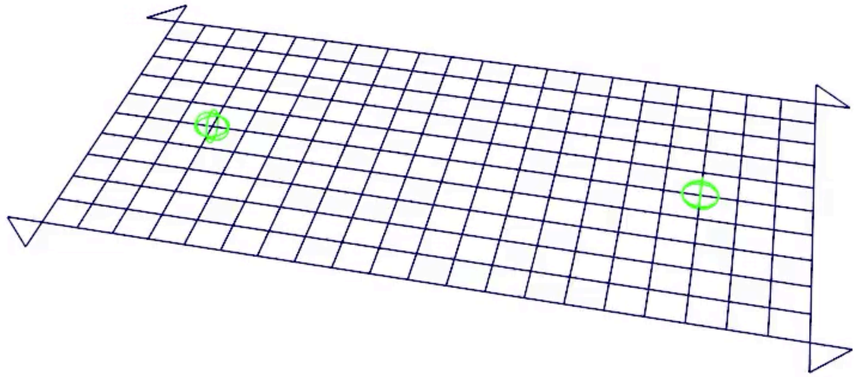


Primary Motion

Cloth Simulation (Mass-Spring System)

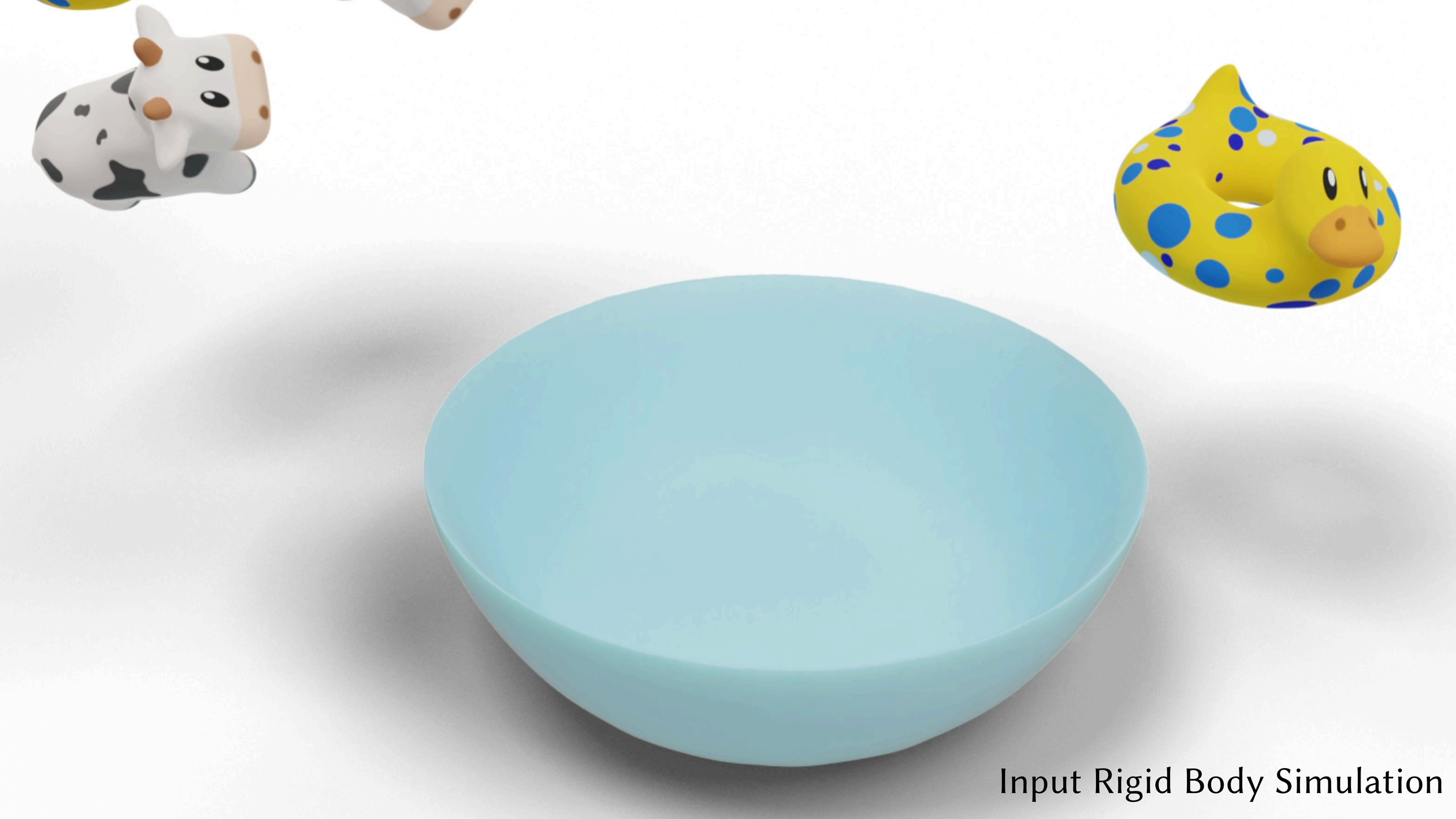
+ Our Complementary Secondary Motion

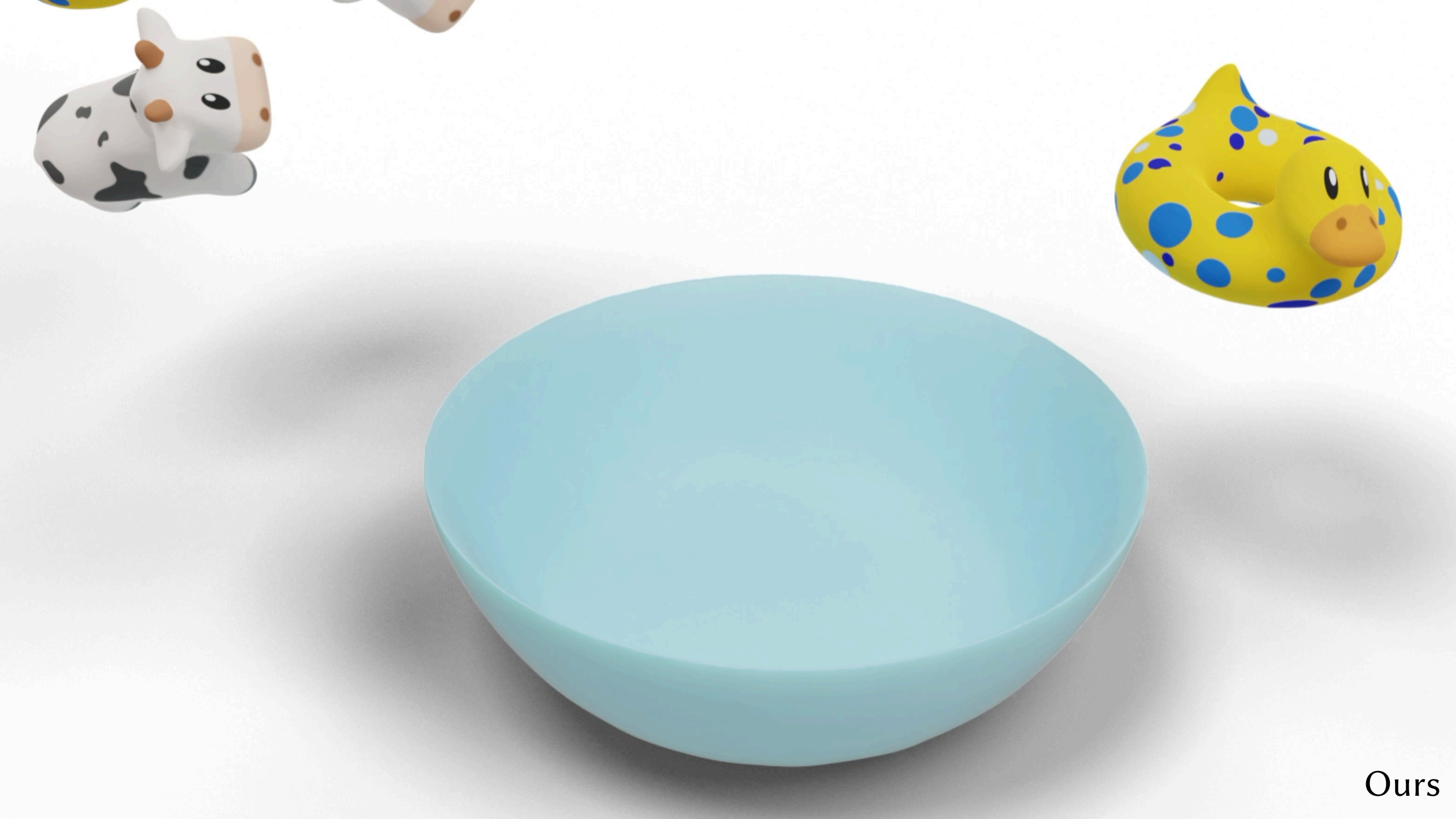
Cloth Simulation (Mass-Spring System)



Input

Output


Input Rigid Body Simulation



Ours

Complementary Dynamics [Zhang et al. 2020]

$$u_t^c = \operatorname{argmin}\ E_t(u_t^r + u_t^c)$$

u_t^c Cost increases with mesh resolution!

subject to $J_t^\top M u_t^c = 0$

Constraint matrix is dense!

Model Reduction

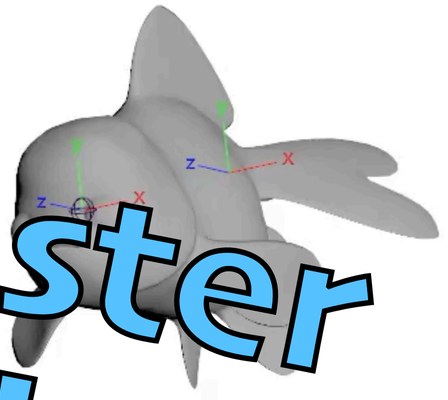
dim(U) >> dim(z)

$$z_t^c = \underset{z_t^c}{\operatorname{argmin}} \; E_t\left(u_t^r + b(z_t^c)\right)$$

subject to $J_t^\top M u_t^c = 0$

Primary Motion + **Complementary Dynamics**

100X faster now!



Our Skinning Modes Are Rotation Equivariant

ARAP Energy : -7.9463e-16

ARAP Energy : -1.5569e-16

ARAP Energy : -2.8129e-16

ARAP Energy : -4.8503e-16

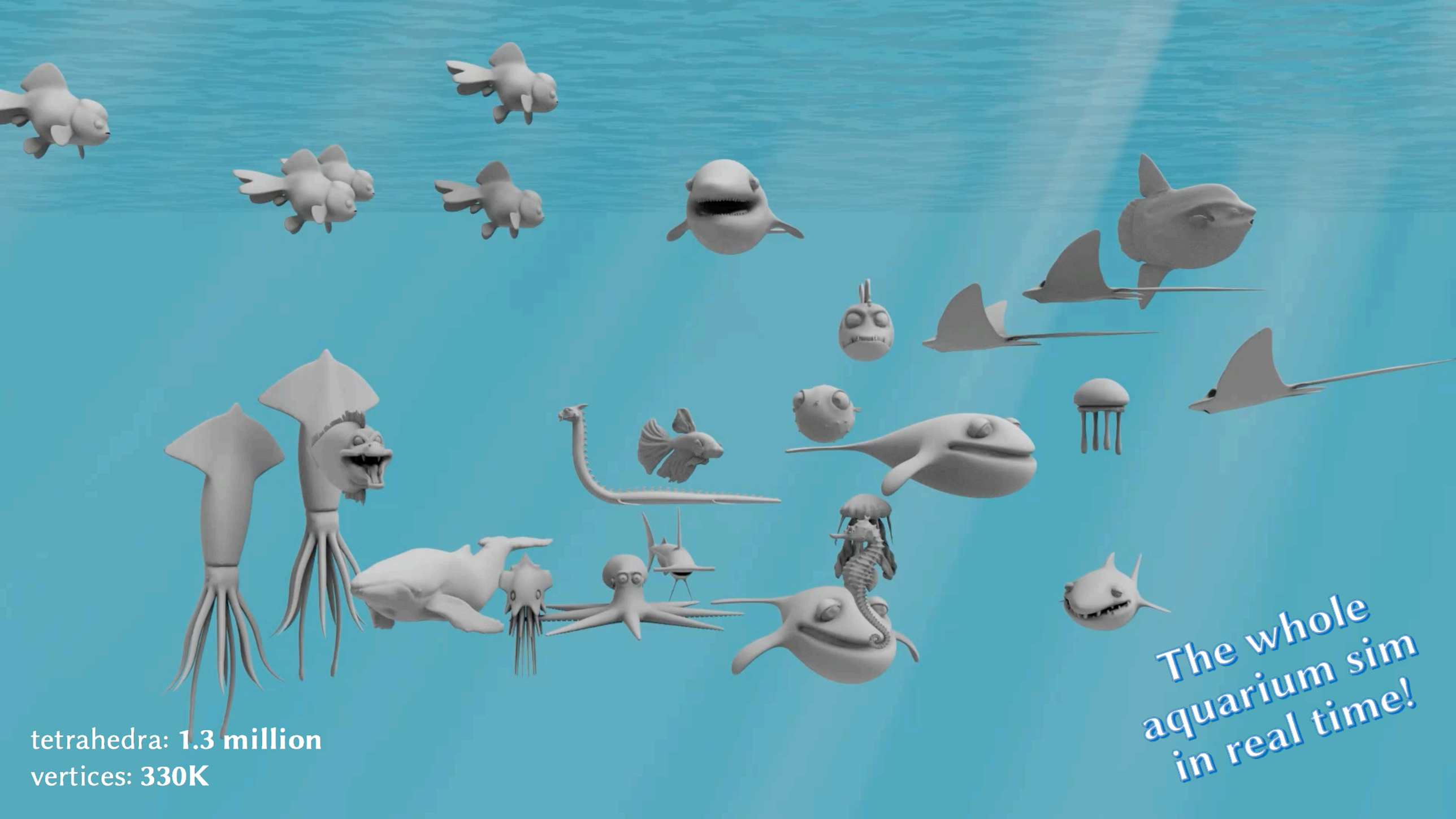
ARAP Energy : -2.8155e-16

ARAP Energy : -7.3058e-16

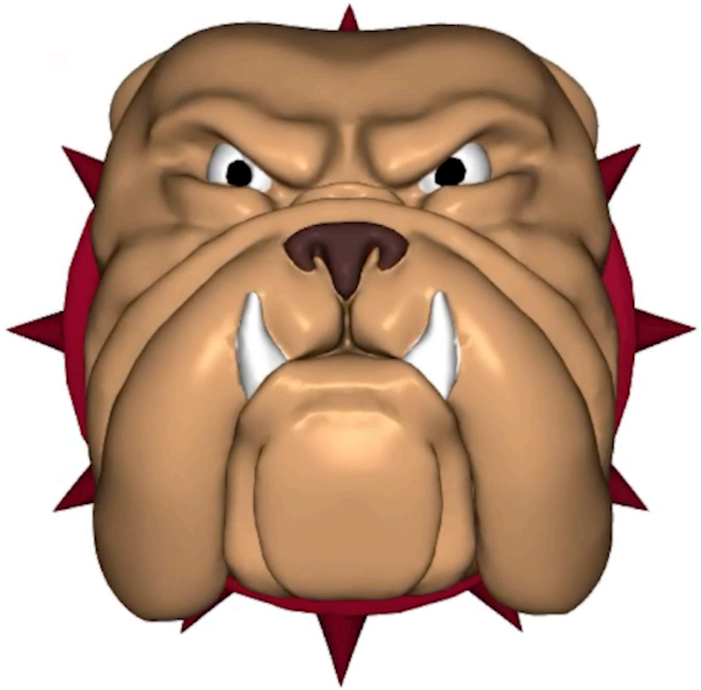
Displacement modes

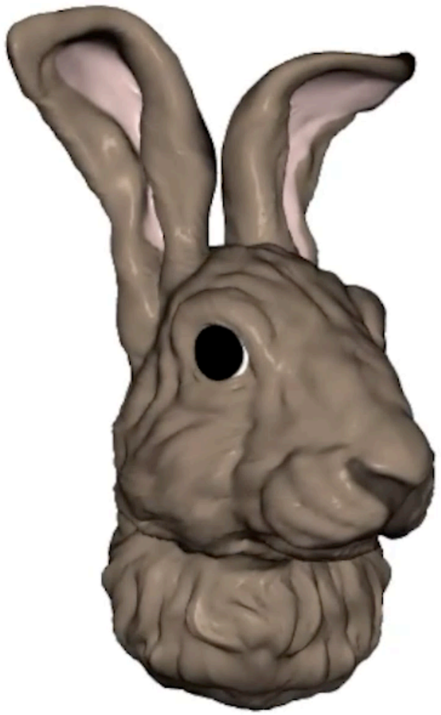
Modal derivatives

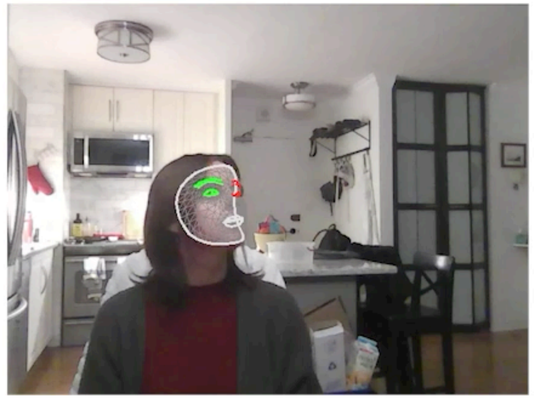
Our skinning modes


tetrahedra: 1.3 million
vertices: 330K
The whole aquarium sim in real time!

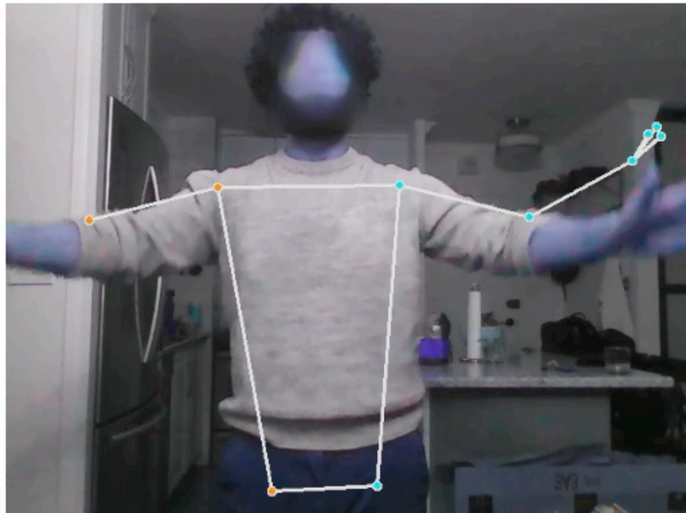
Face Tracking Augmentation







Skeleton Tracking Augmentation



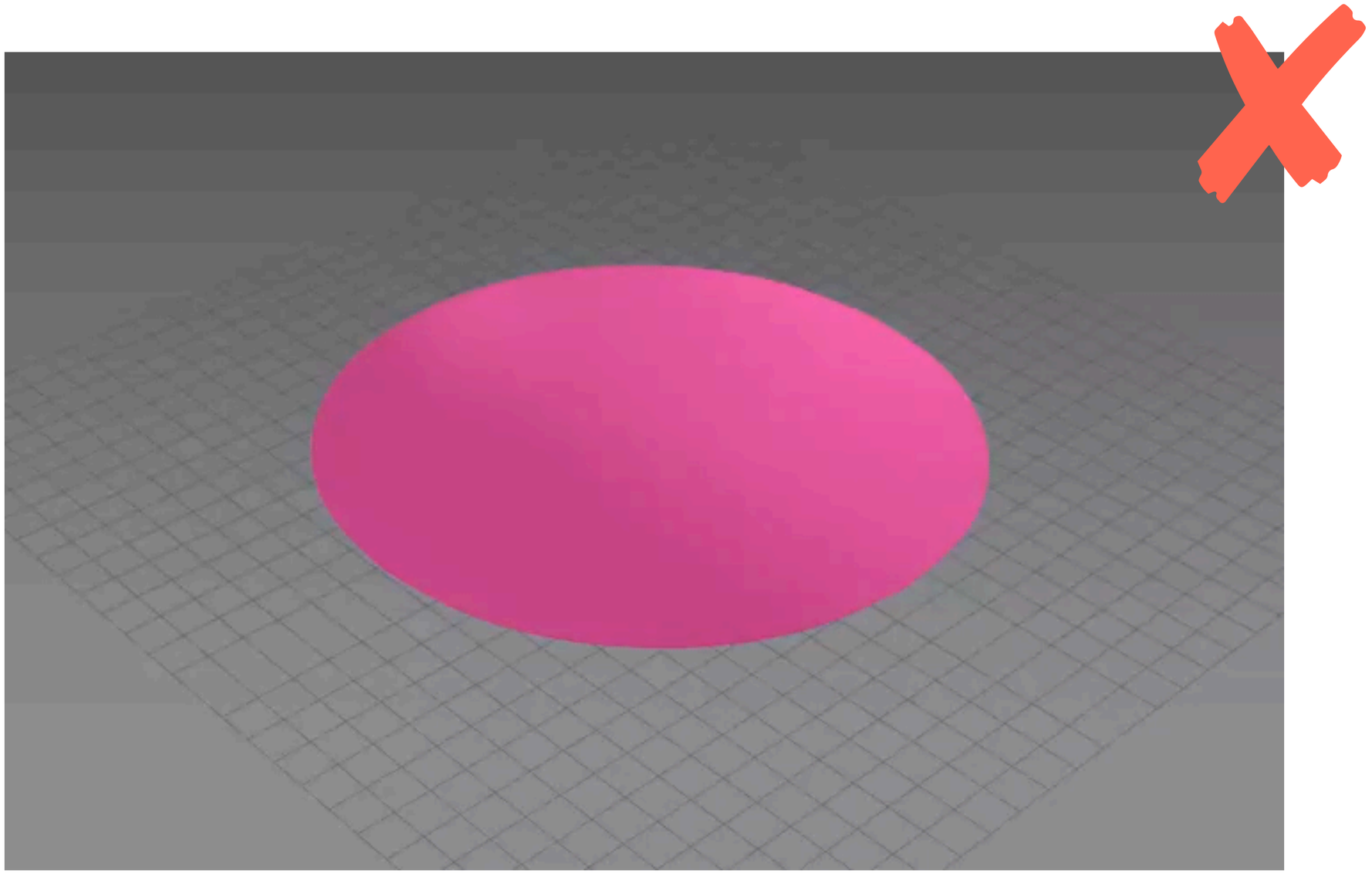
In physical simulation..

Speed

v.s.

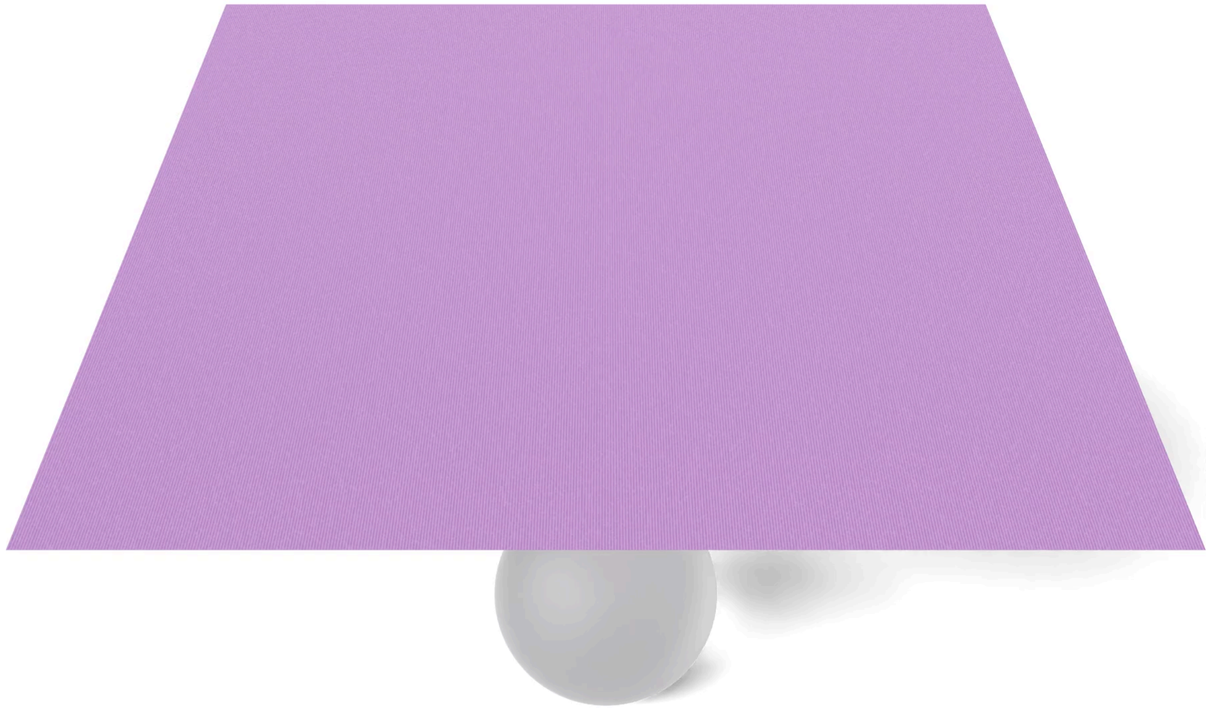
Fidelity

High-res sim in real time?

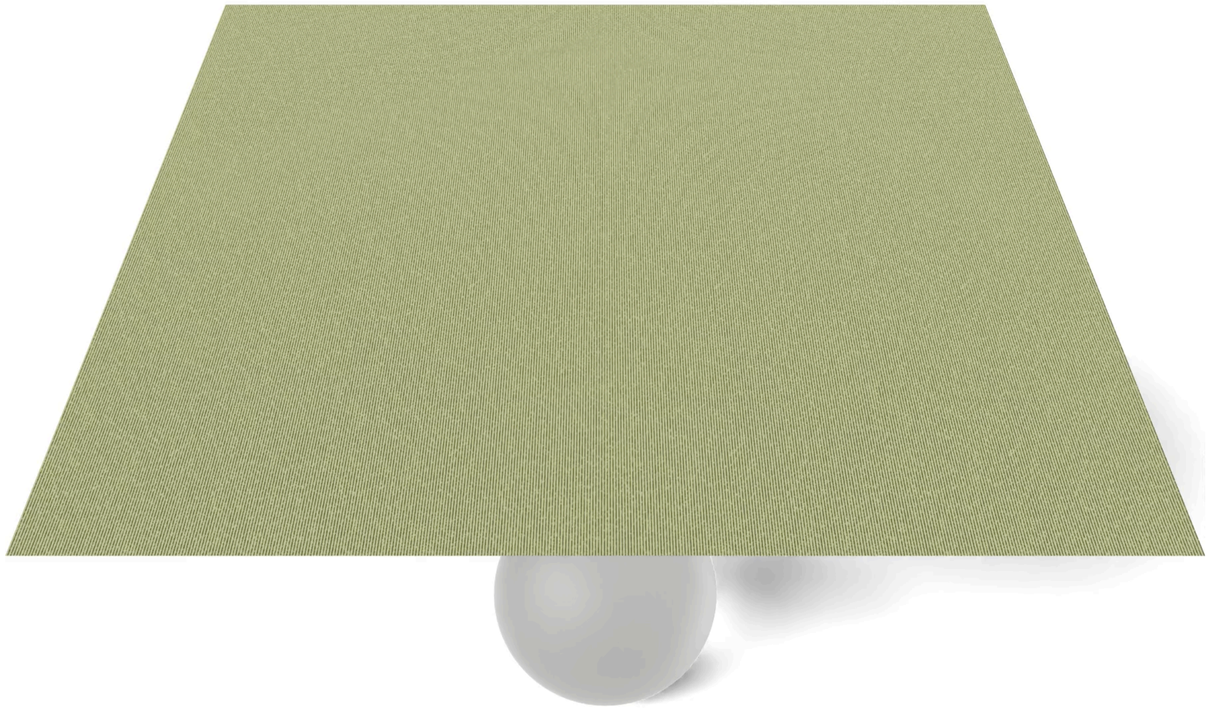


Marvelous Designer - https://twitter.com/paarsec/status/1412440542202499073

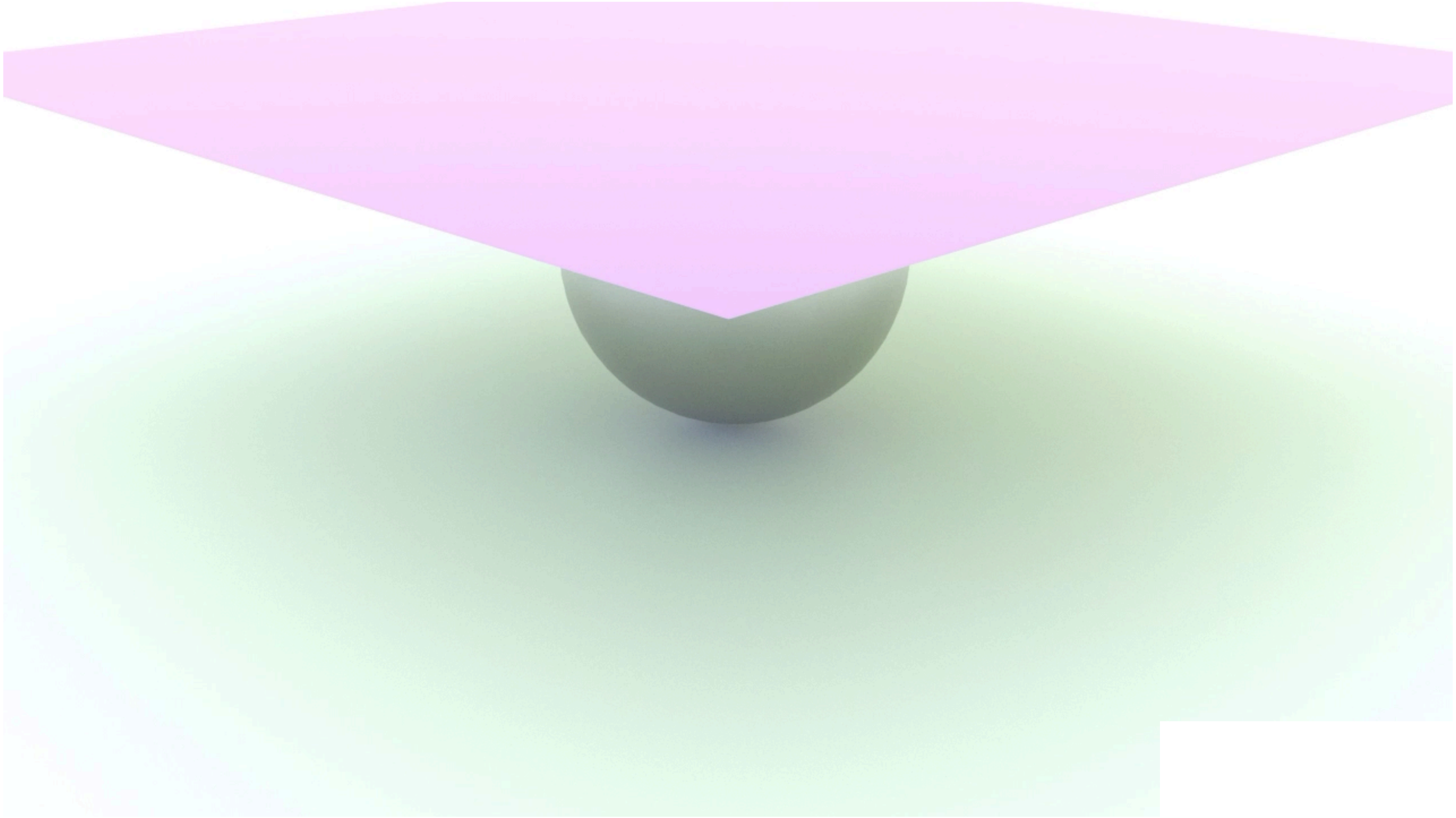
Intersection and Explosion



Houdini Vellum



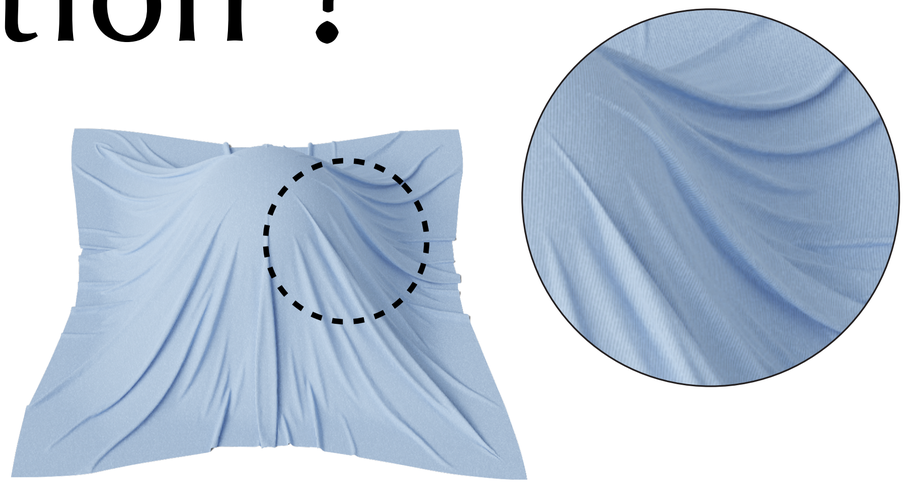
Marvelous Designer



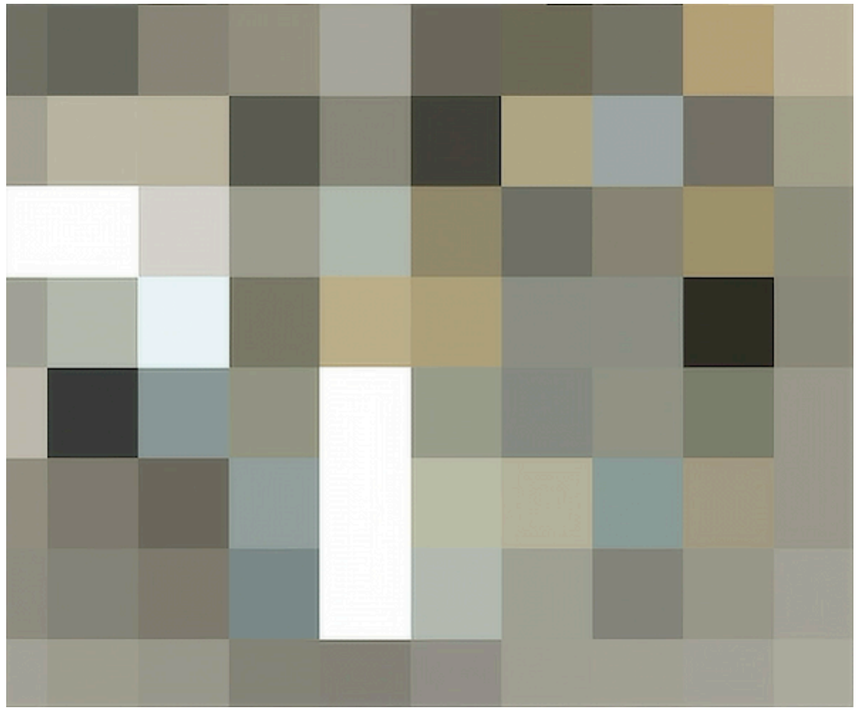
IPC [Li et al. 2021]

Interactive + High-quality

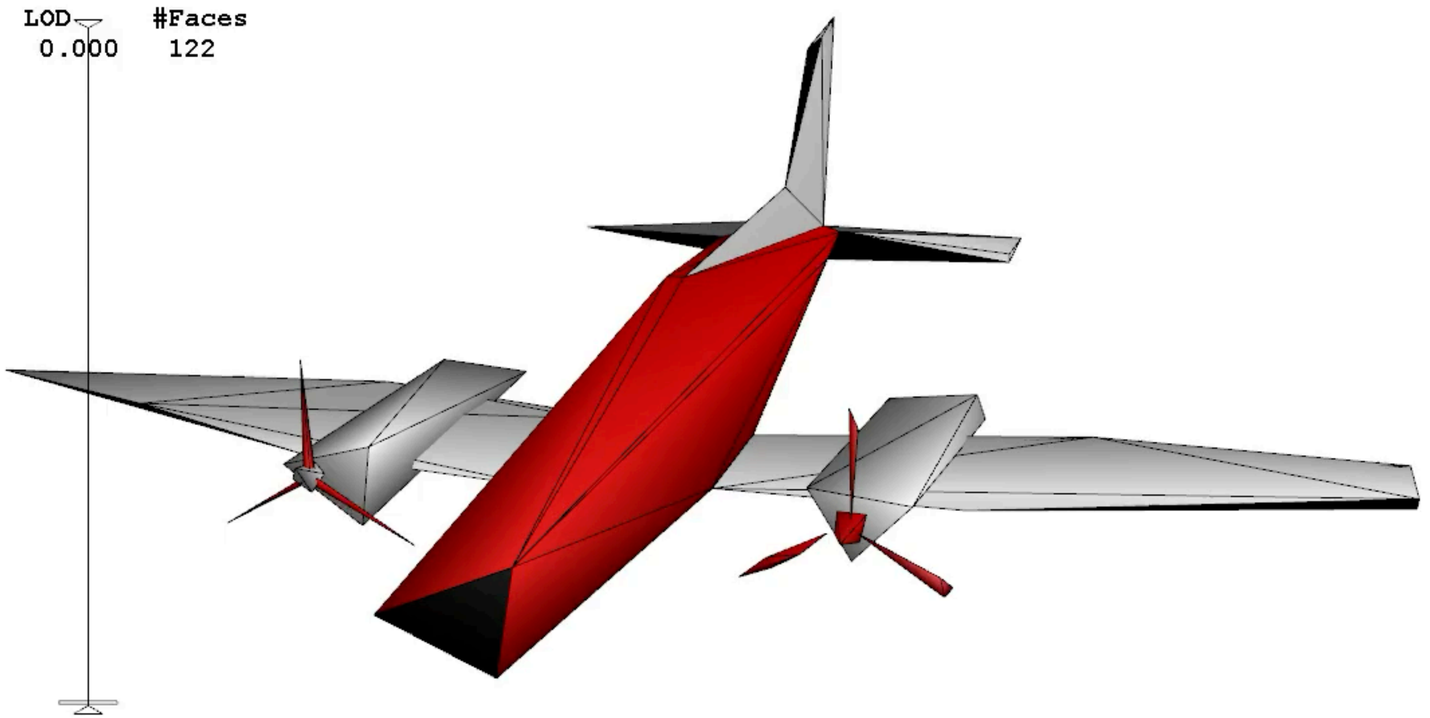
Cloth Simulation ?



Progressive Refinement in Rendering and Geometry

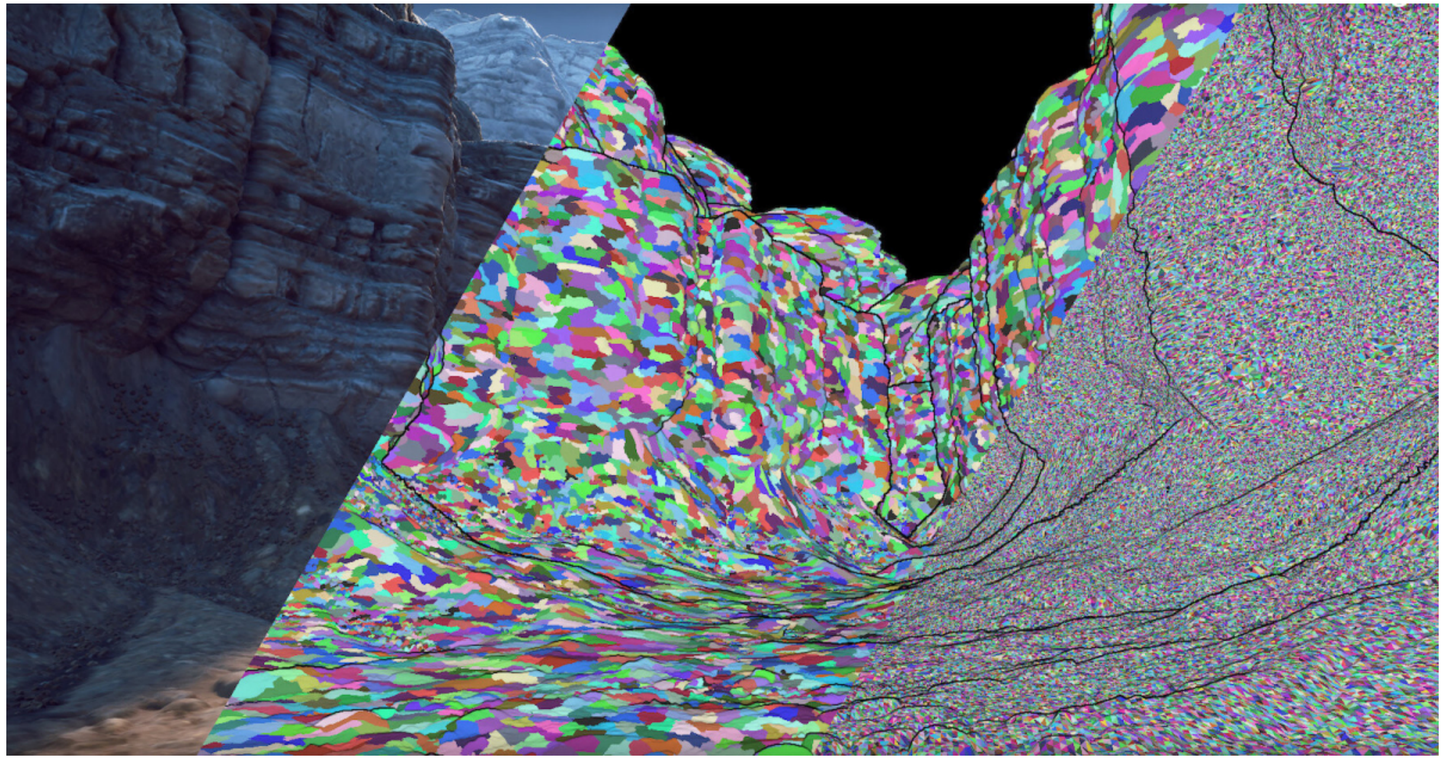


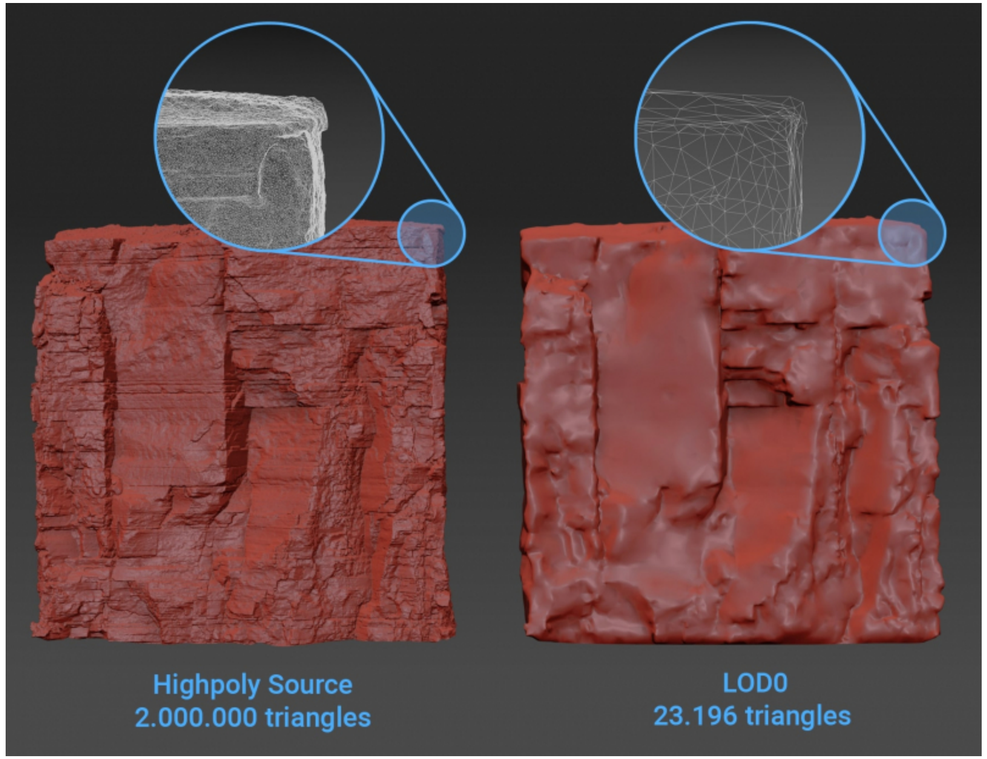
[Redway3D 2021]



[Hoppe 1996]

Progressive Refinement in Rendering and Geometry



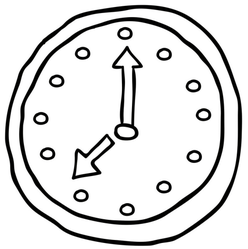


UE5 Nanite

https://80.lv/articles/discussing-the-possibilities-and-drawbacks-of-unreal-engine-5-s-nanite/

How can we map the progressive approach to cloth simulation

Challenge #1: Inconsistencies in Direct Simulation



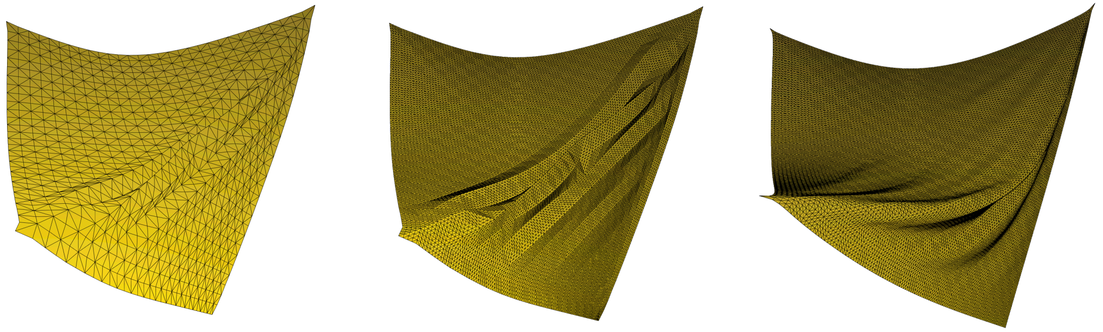
Not similar!

Low-res direct sim

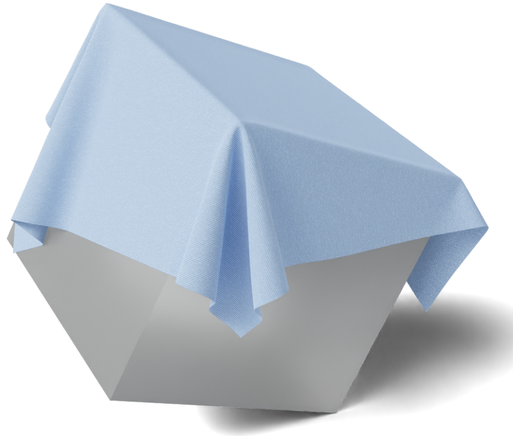
High-res direct sim

Challenge #2: Artifacts in Cascadic Methods

Sensitive Couture [Umetani et al. 2011]



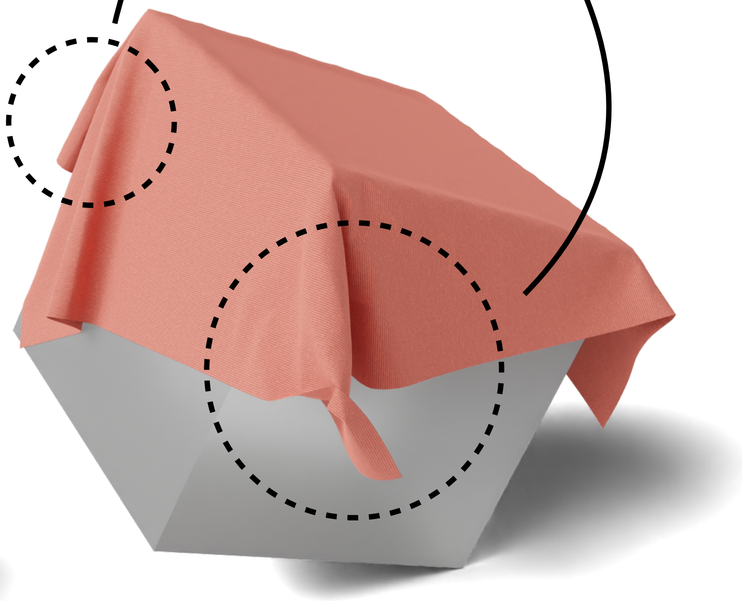
Coarse-level Simulation | Prolongated Solution | Fine-level Simulation



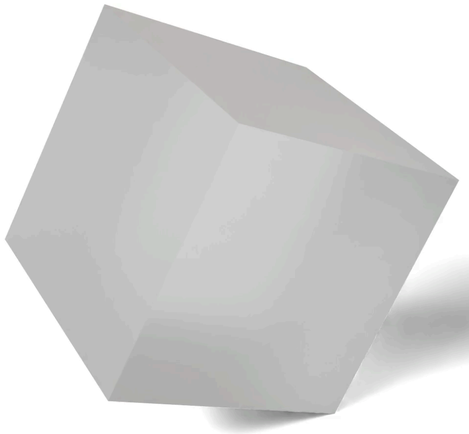
Direct simulation

Membrane Locking!

Sharp creases and kinks



Level 3



Level 0

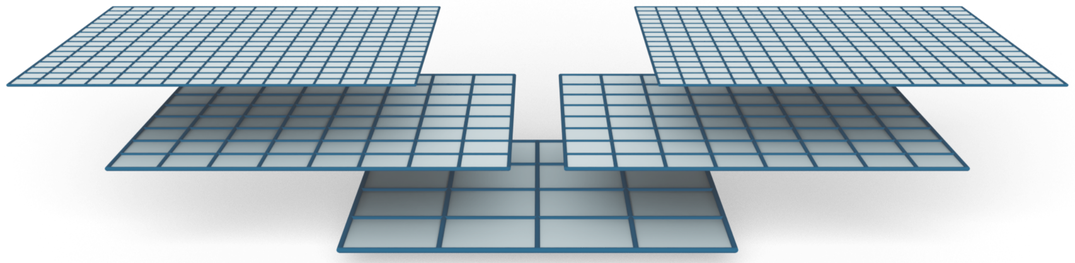
Level 1

Level 2

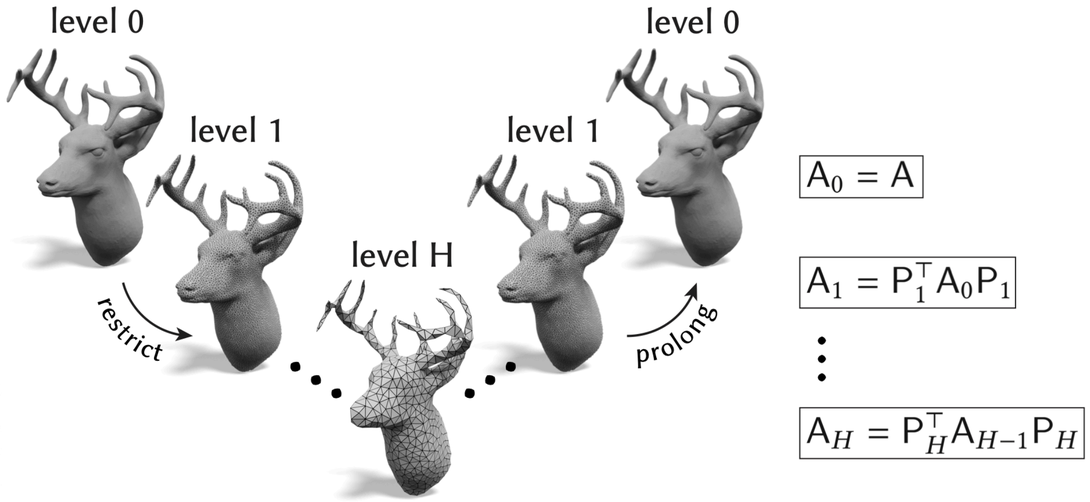
Challenge #3: Linear Multigrid Doesn't Apply

$$Ax = b$$

Applies for solving linear systems!



Regular Grid



Unstructured Domain

[Liu et al. 2021]

Progressive!

$\neq$

~~Interactive + High-quality~~

Cloth Simulation ?

Progressive Cloth Simulation

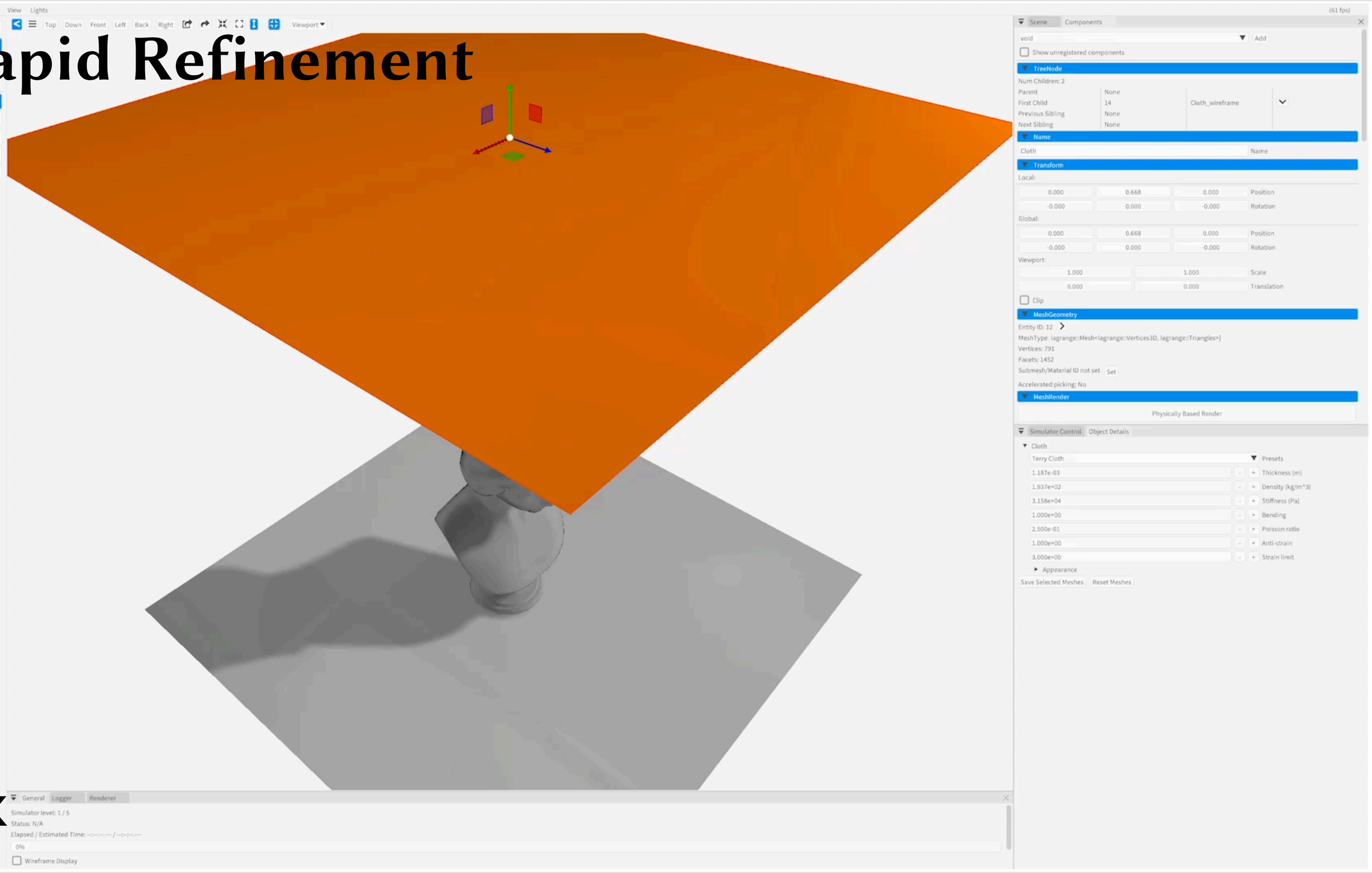
Consistent progression across resolutions

Reliable stable, intersection-free simulation

Interruptibility at any point for parameter changes

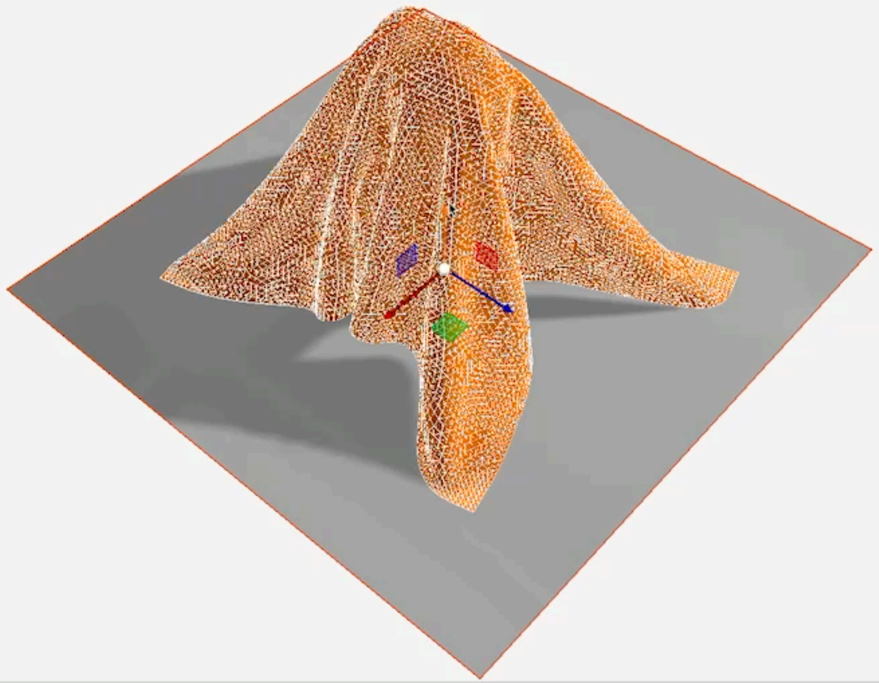
Interactive preview with consistent coarse-level simulation

Rapid Refinement



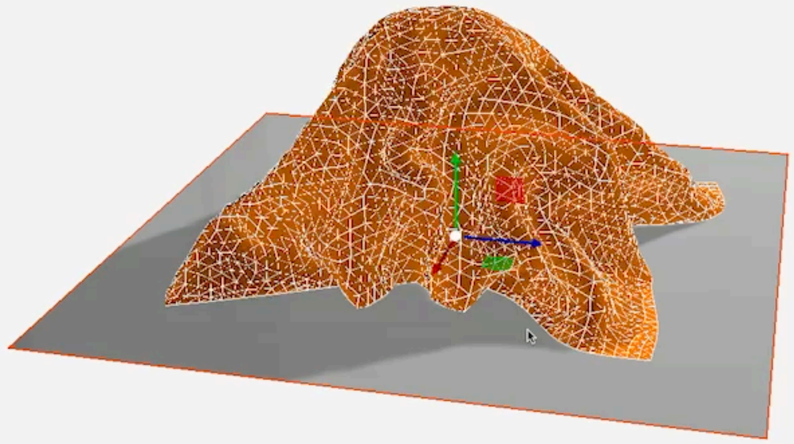
3x

Asynchronous Collision Resolution



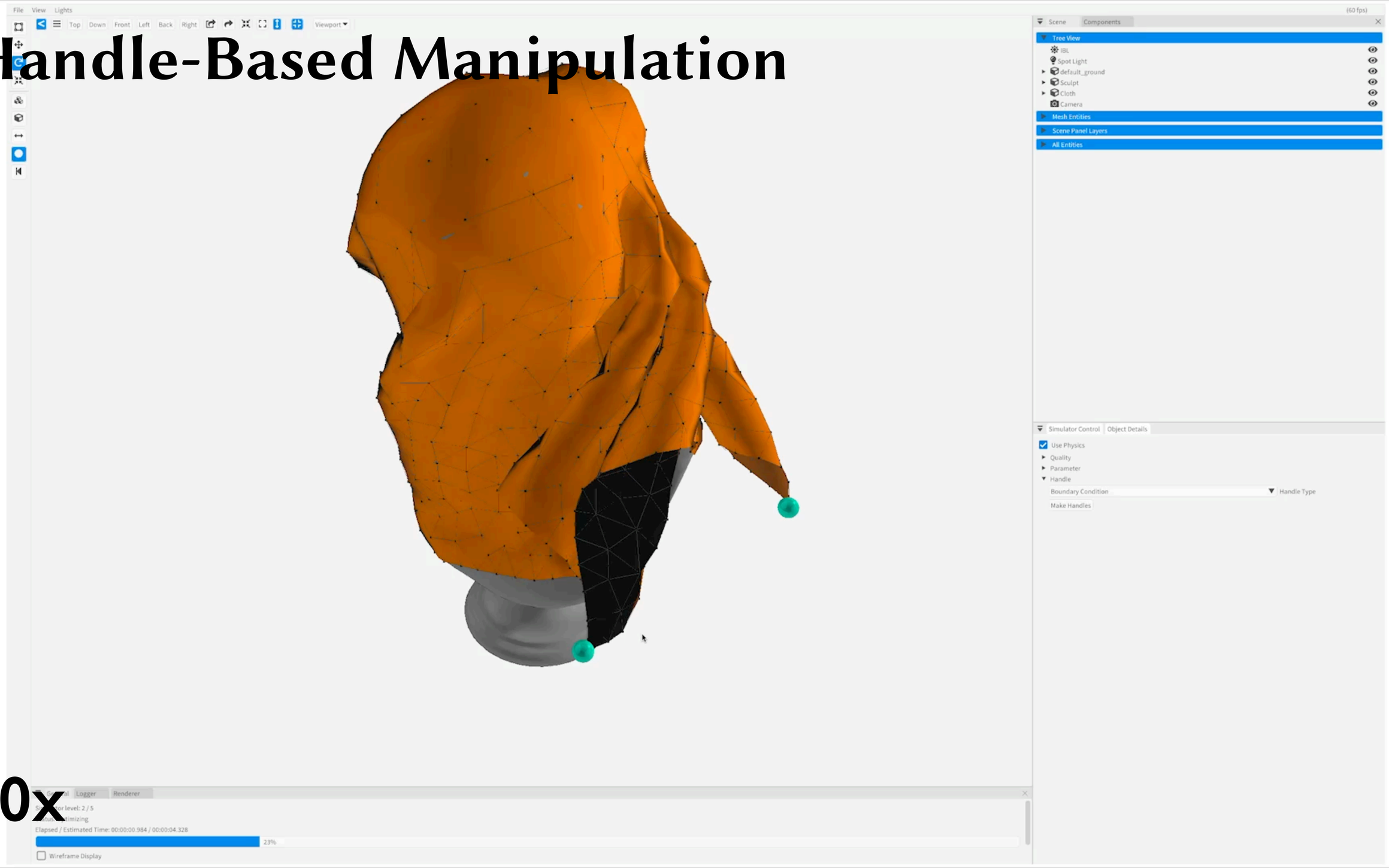
3x

Material Exploration



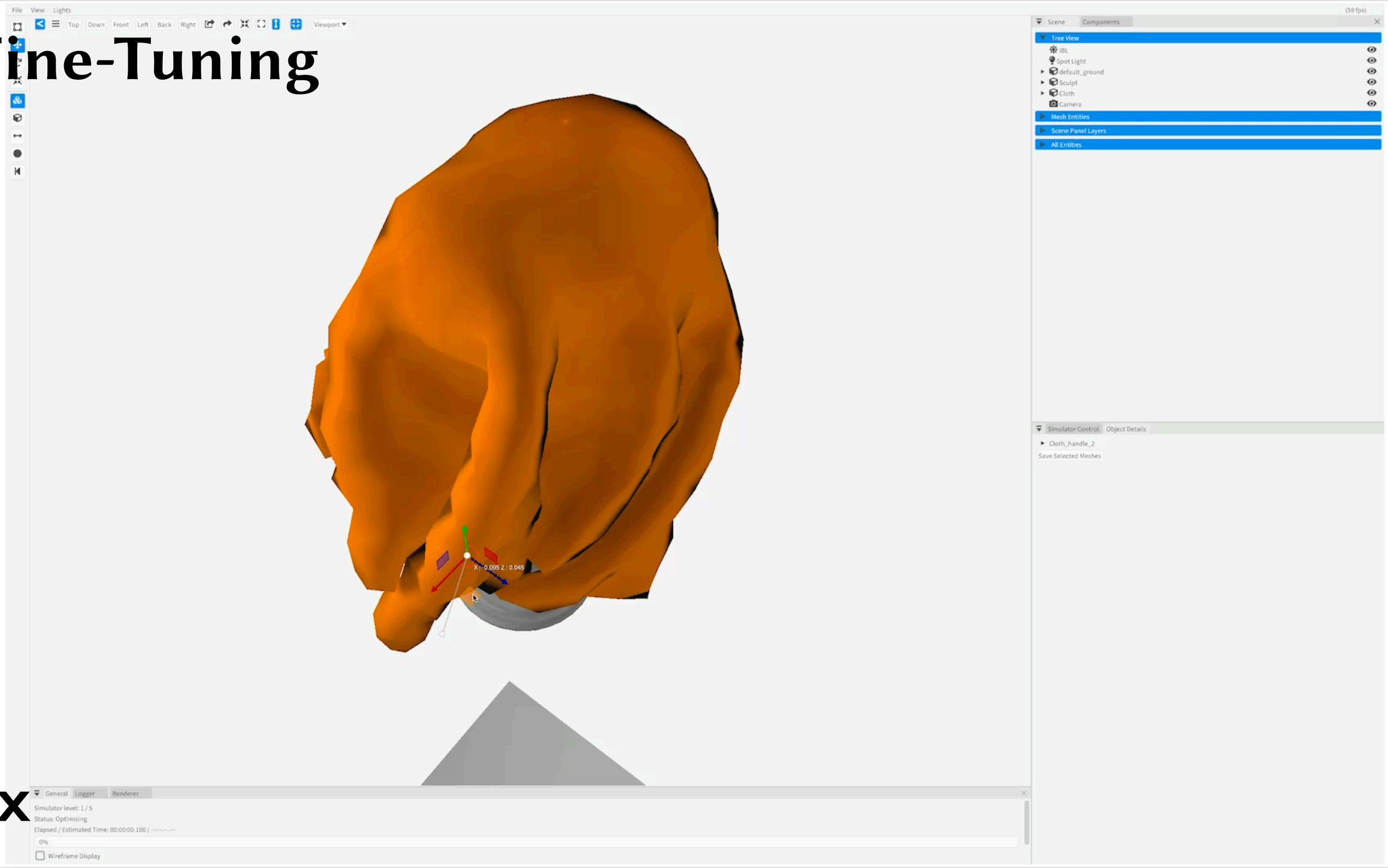
4x

Handle-Based Manipulation



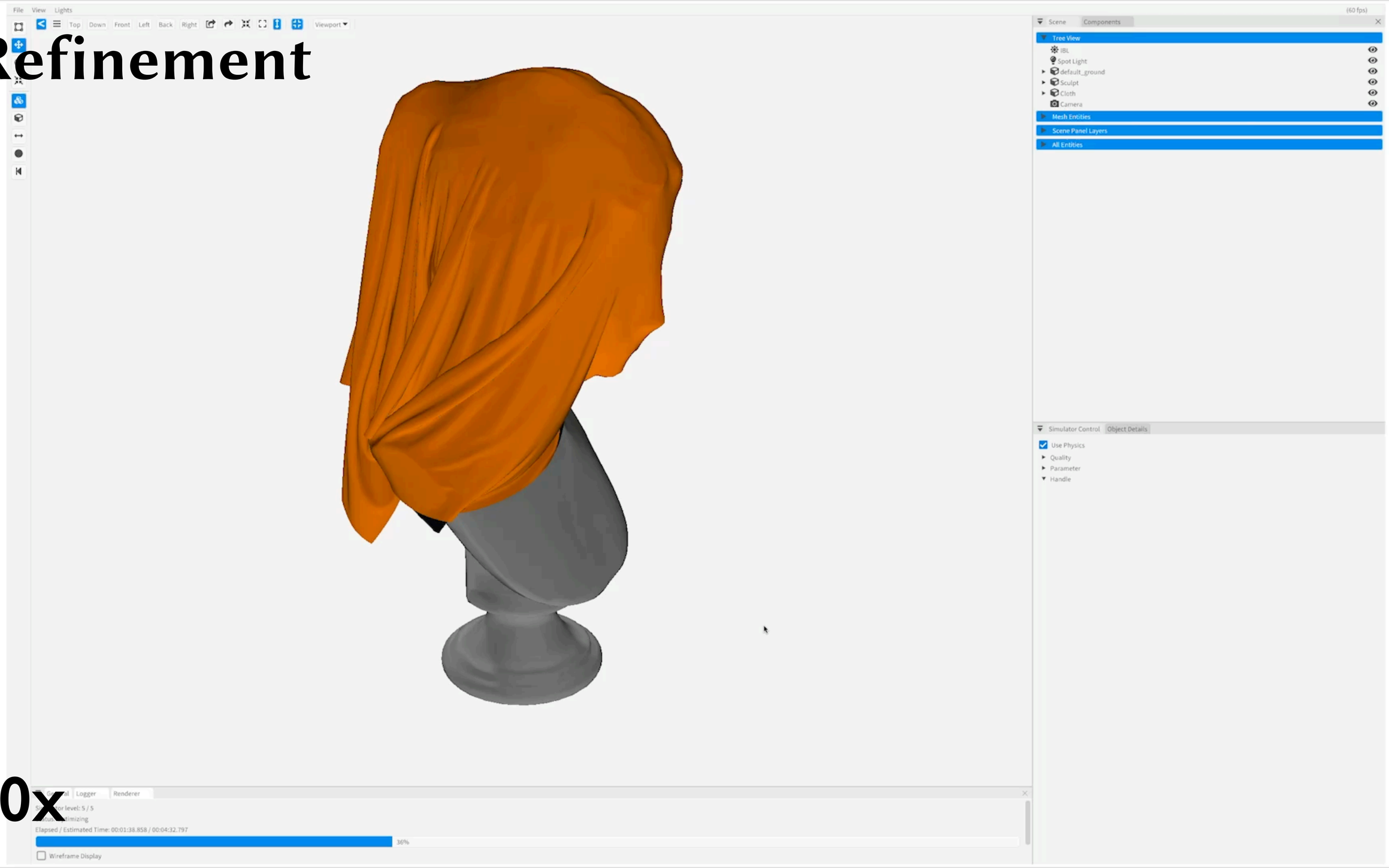
10x

Fine-Tuning



3x

Refinement



40x

1.5K
6K
24K
92K
372K
Progressive Refinement

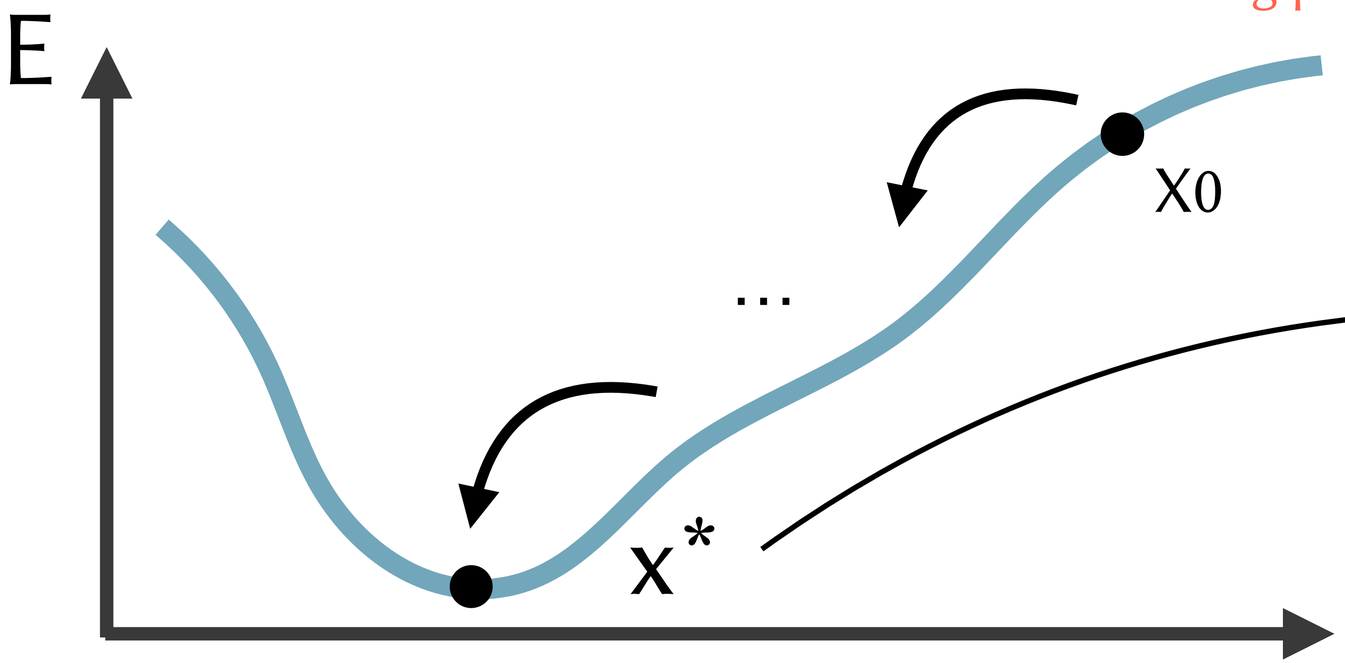
Incremental Potential Problem

$$x^* = \underset{x}{\operatorname{argmin}}\, E(x, \bar{x})$$

where $E = \Psi + \Phi + B + D + S$

Membrane and bending energies

Contact, friction and strain limiting potentials

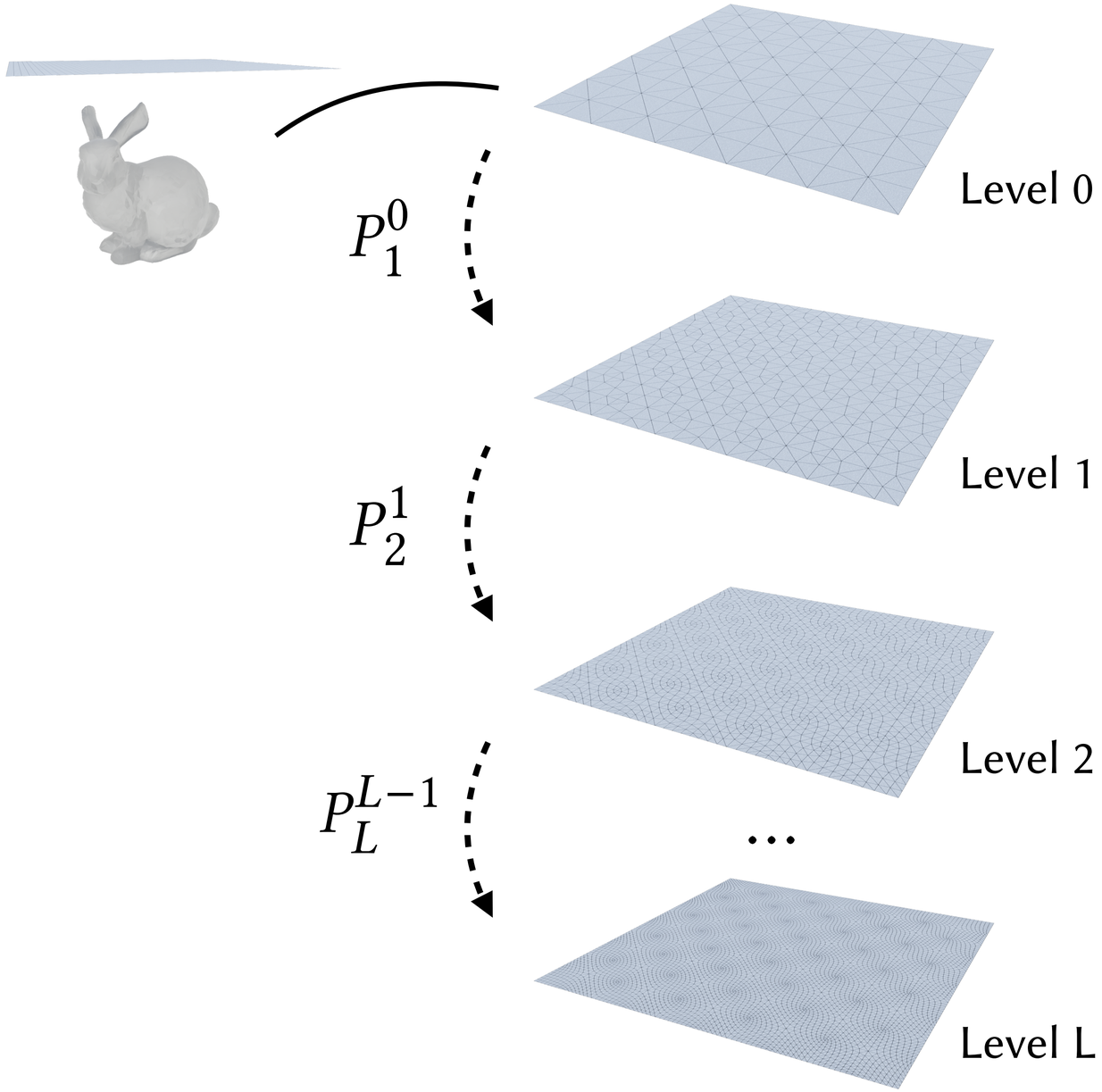


Initial state

[Li et al. 2021]

Final converged state

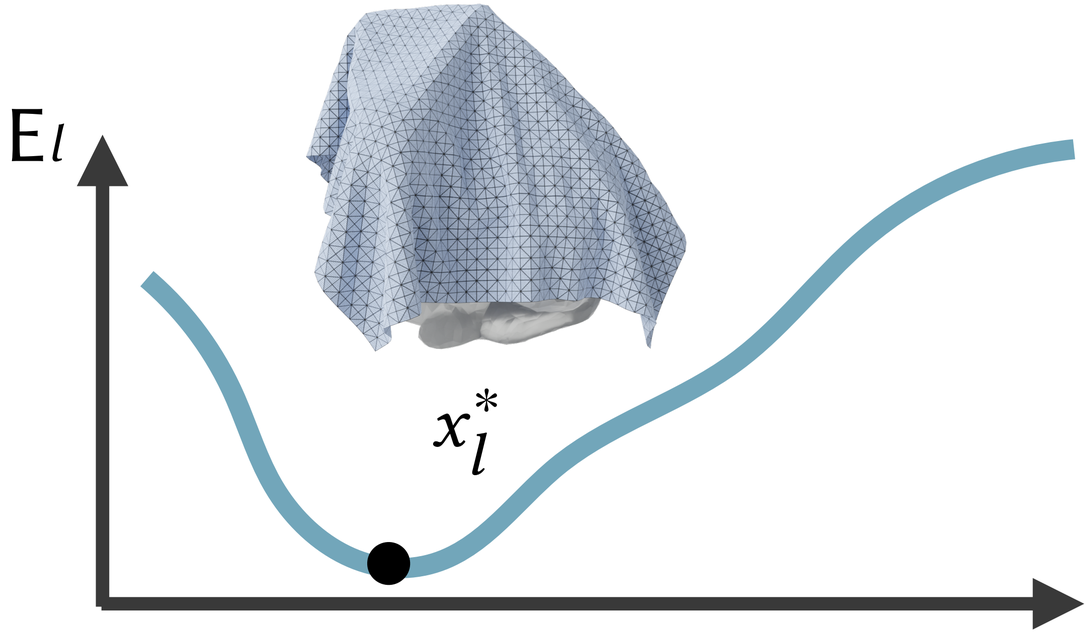
Prolongation and Multi-resolution IP Problem



★ **For each level** *l = 0, 1, 2, .., L*

$$x^* = \underset{x}{\operatorname{argmin}}\, E_l(x, \bar{x})$$

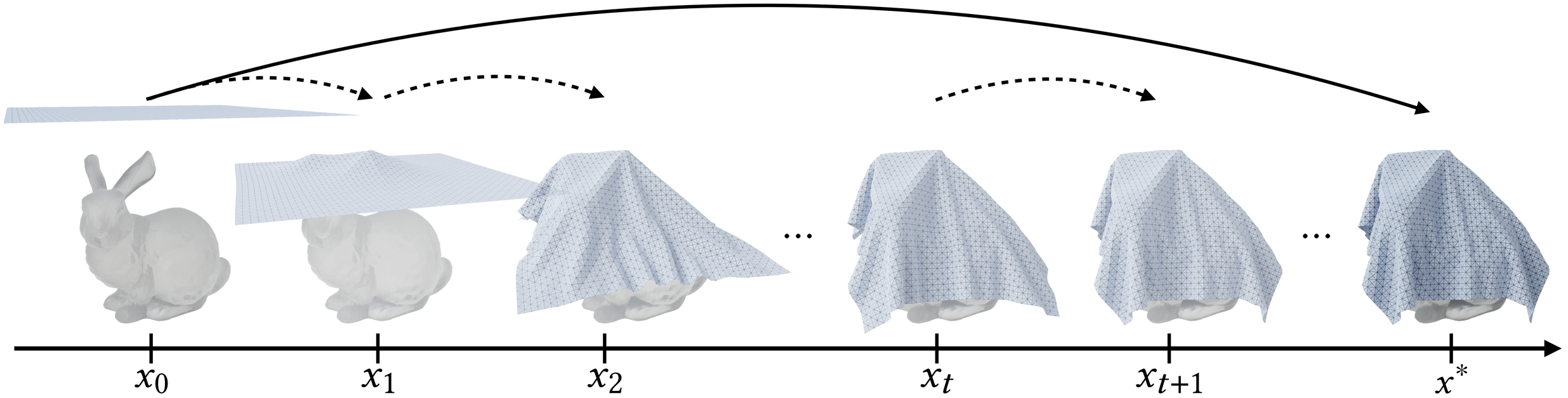
where $E_l = \Psi_l + \Phi_l + B_l + D_l + S_l$



Quasistatic Stepping

For each level
l = 0, 1, 2, … , L

$$x^* = \underset{x}{\operatorname{argmin}}\, E_l(x, \bar{x})$$



Artificial time stepping (implicit Euler)

$$x^{t+1} = \underset{x}{\operatorname{argmin}}\, E_l(x, \bar{x})$$

Coarse-level Proxy Energies

For each level
$l = 0, 1, 2, \ldots, L$

$$x^* = \operatorname*{argmin}_x E_l(x, \tilde{x})$$

where $E_l = \Psi_l(x_l) + \Phi_l(x_l) + B_l(x_l) + D_l(x_l) + S_l(x_l)$

Coarse-level Proxy Energies

For each level
$l = 0, 1, 2, \ldots, L$

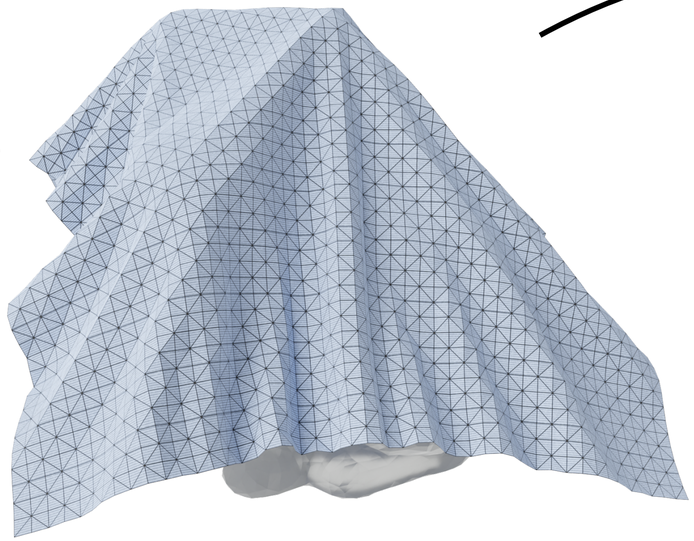
$$x_l^* = \underset{x_l}{\operatorname{argmin}}\, F_l(x_l, \bar{x}_l, P^l)$$

$\star P^l = P_L^{L-1} \cdots P_{l+1}^l$

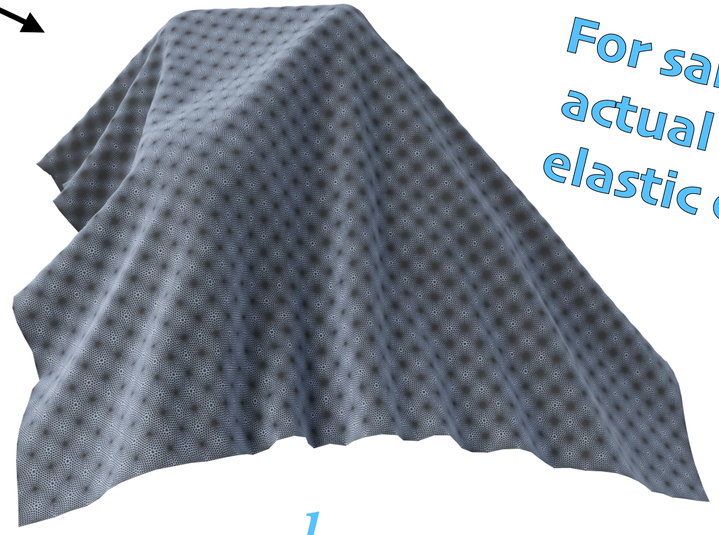
where $F_l = B_l(x_l) + D_l(x_l) + S_l(x_l) + \Psi_L(P^l x_l) + \Phi_L(P^l x_l)$

Prolong to the fine level

For handling contact constraints!



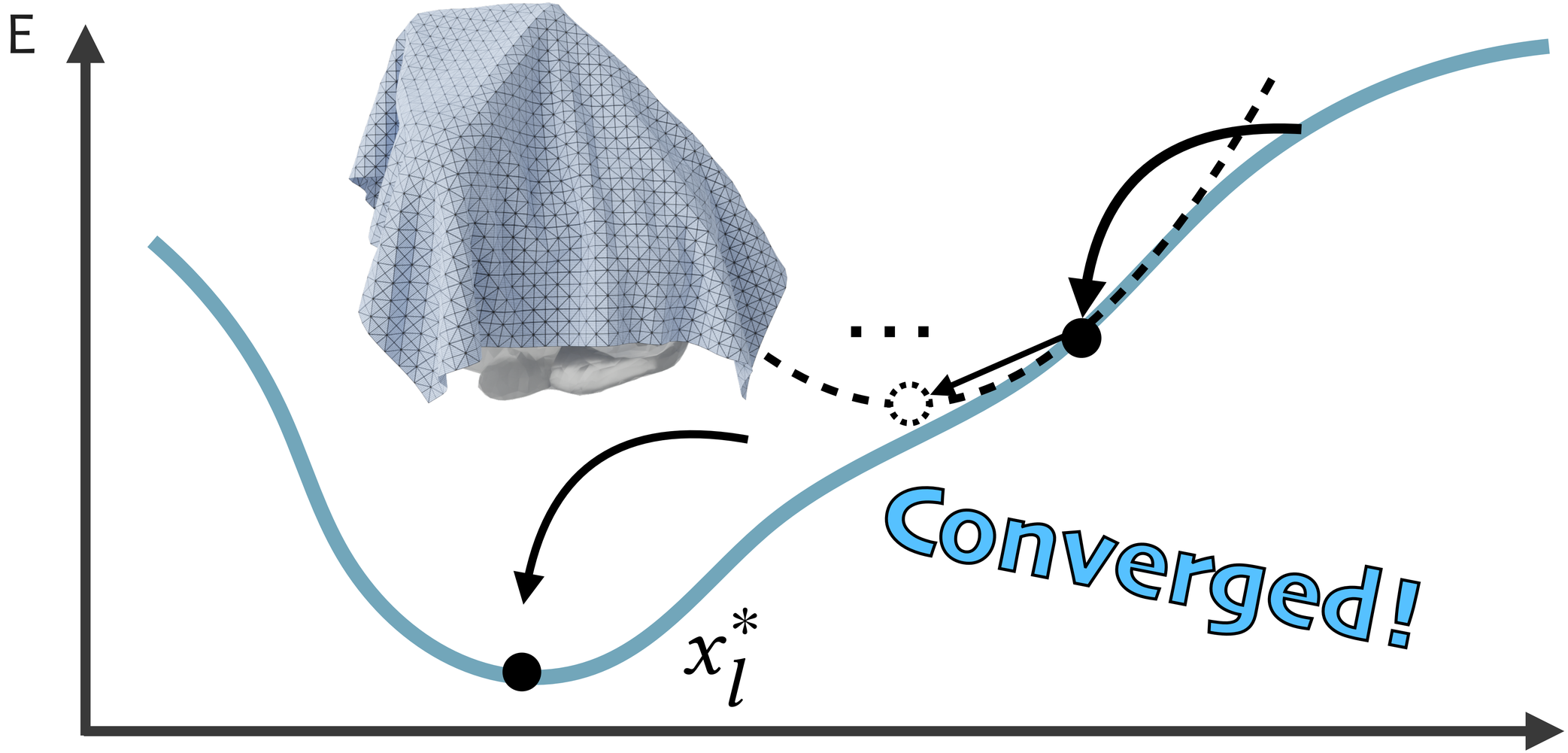
x_l



$P^l x_l$

For sampling the actual fine-level elastic energies!

Newton-type Descent Optimizer



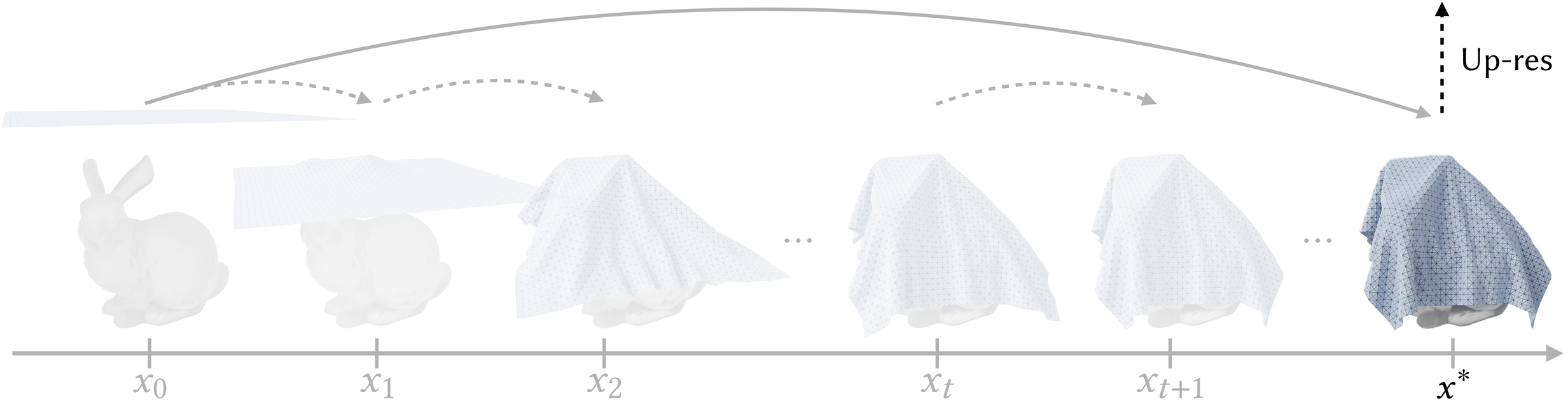
Quasistatic Preview Stepping

intersections

For each level
$l = 0, 1, 2, \ldots, L$

$$x_l^* = \underset{x_l}{\operatorname{argmin}}\, F_l(x_l, \bar{x}_l, P^l)$$

Up-res



Artificial time stepping (implicit Euler)

$$x_l^{t+1} = \underset{x_l}{\operatorname{argmin}}\, F_l(x_l, \bar{x}_l, P^l) + \frac{1}{2}\|x_l - x_l^t\|_M^2$$

Safe Prolongation for Contacts

$$x_{l+1}^0 = U_{l+1}^l x_l^* + \alpha(P_{l+1}^l x_l^* - U_{l+1}^l x_l^*)$$

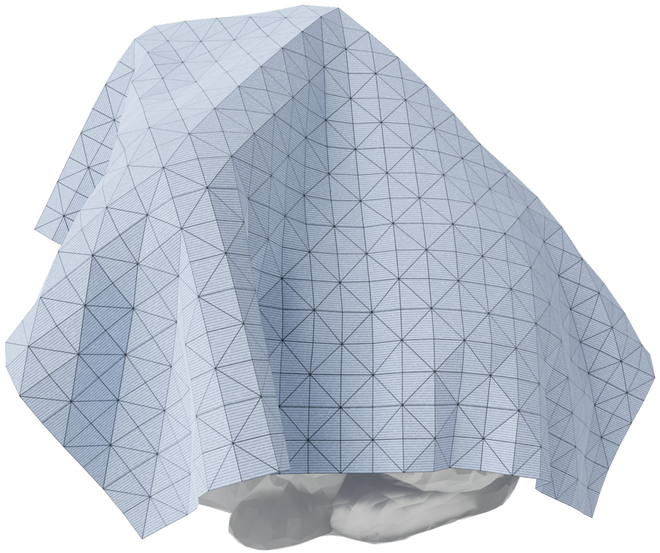
Prolong to the next level

In-plane upsampling

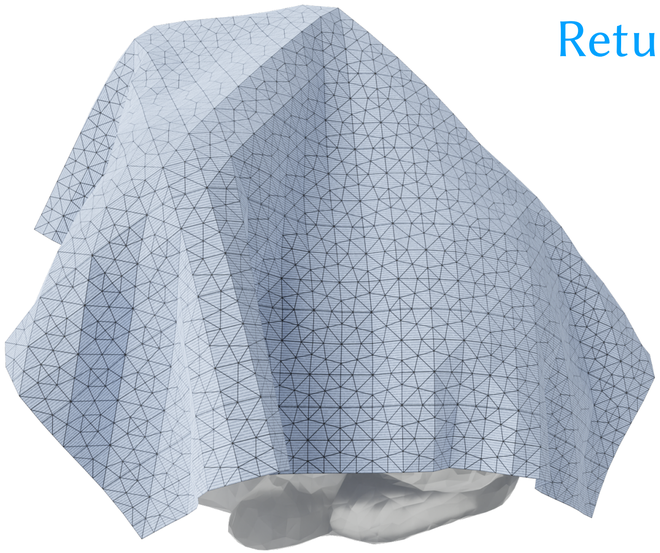
CCD search

Return α

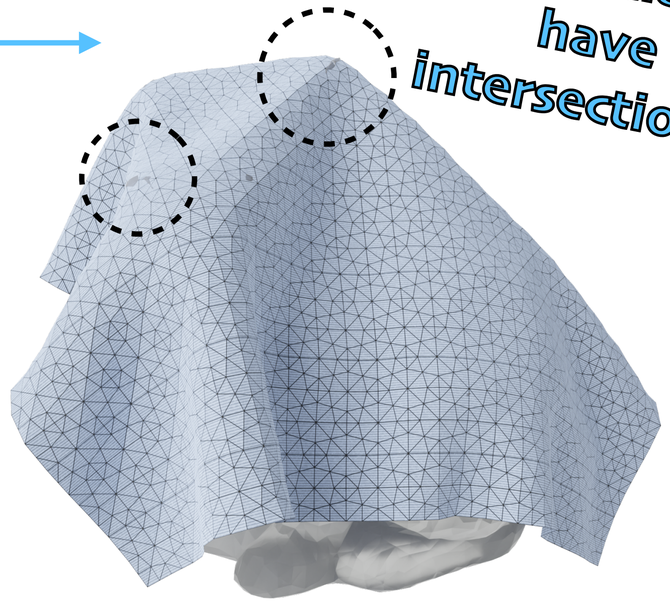
Potentially have intersections!



Intersection-free converged state at level l x_l^*



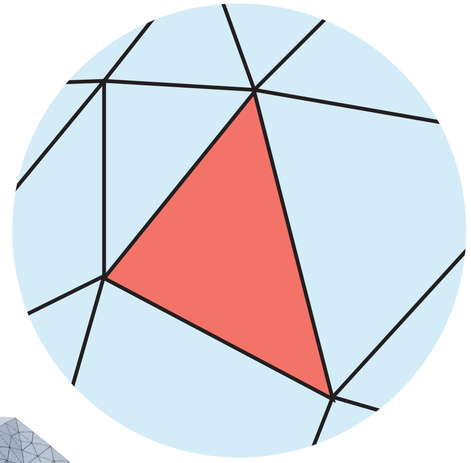
Safe initial (intersection-free) shape at level $l+1$ $U_{l+1}^l x_l^*$



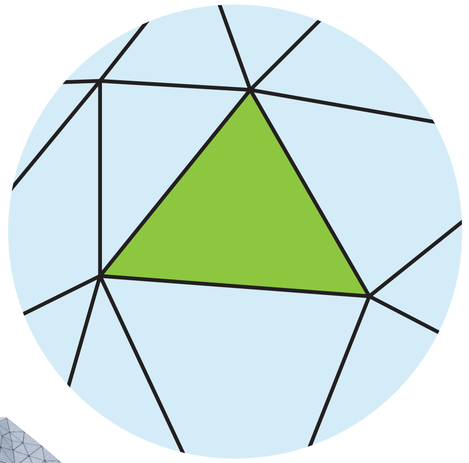
Target shape at level $l+1$ $P_{l+1}^l x_l^*$

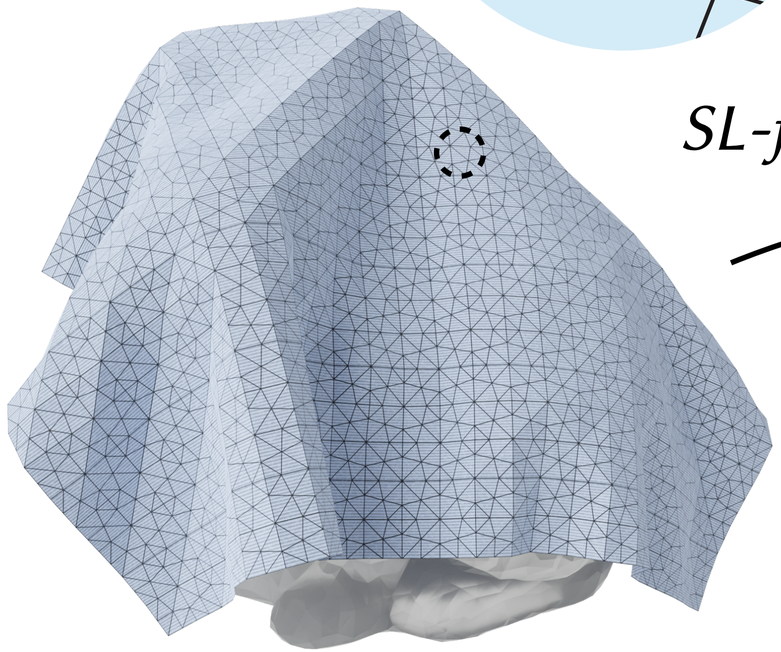
Safe Prolongation for Strain Limiting

Principle stretches are over the limits!

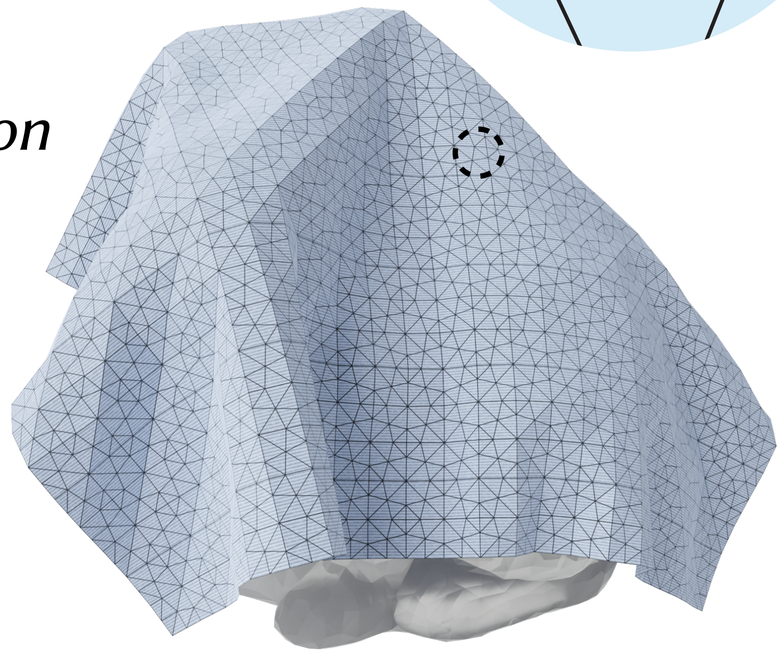


Principle stretches are under the limits!



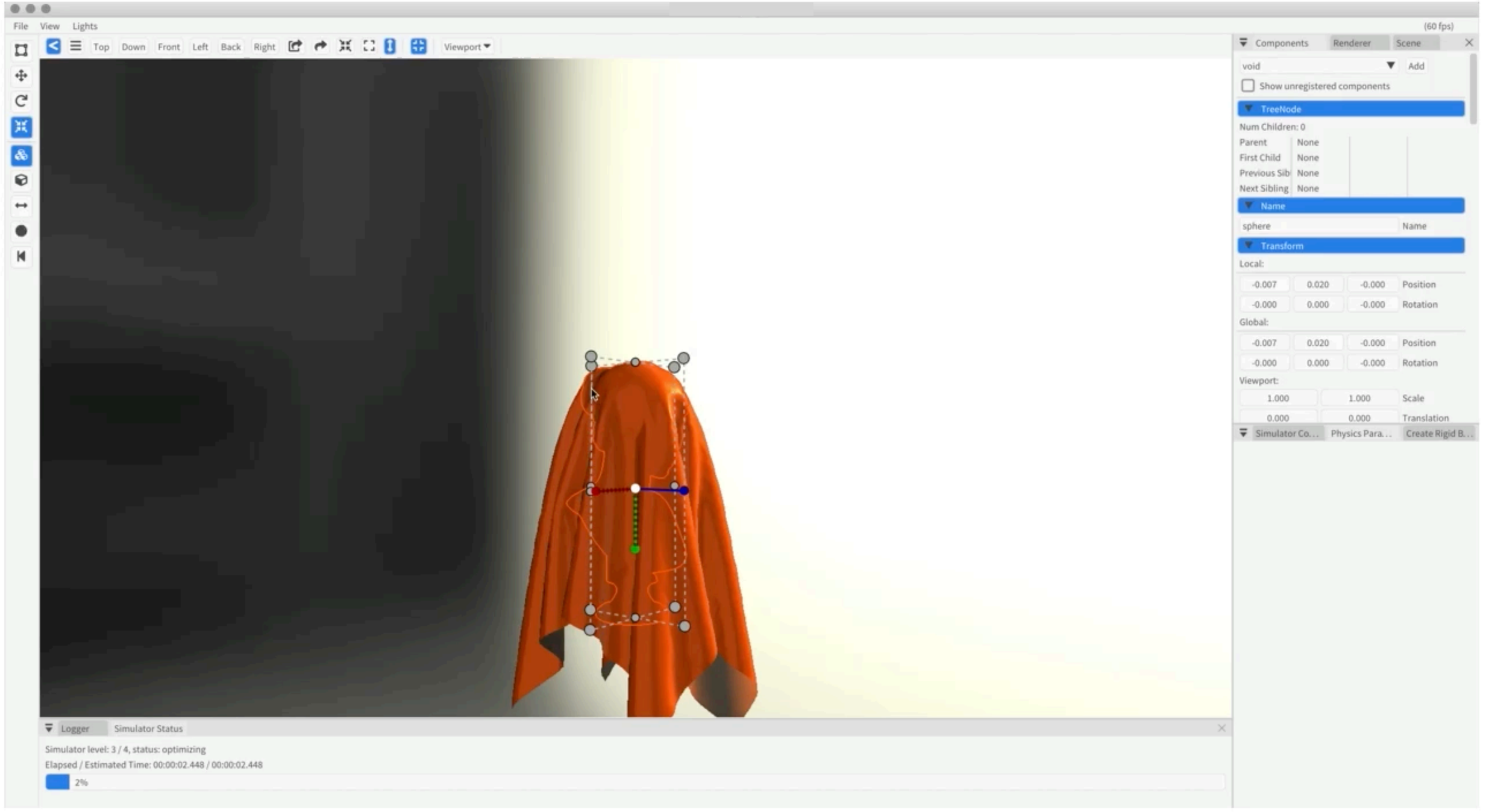


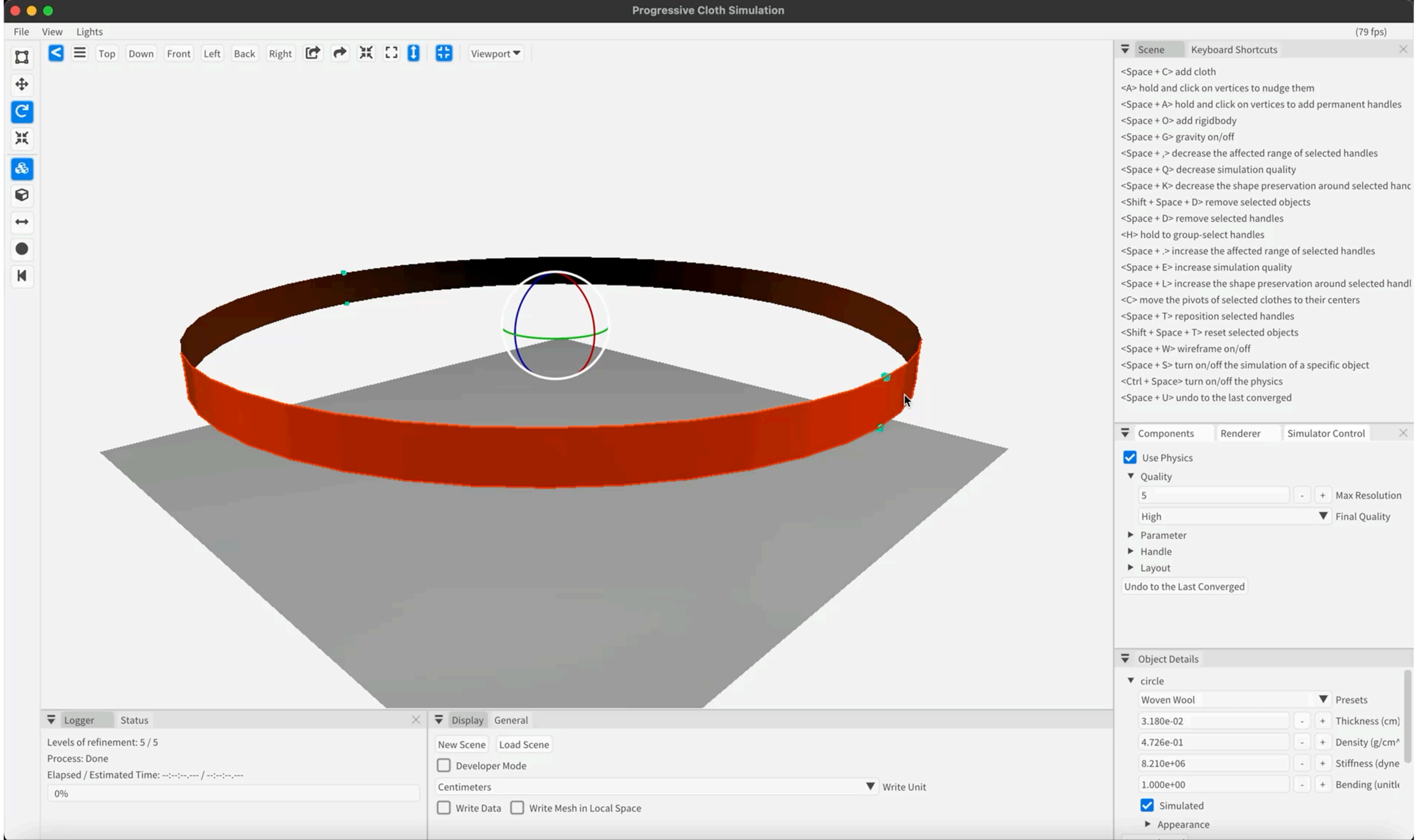
SL-feasible Optimization



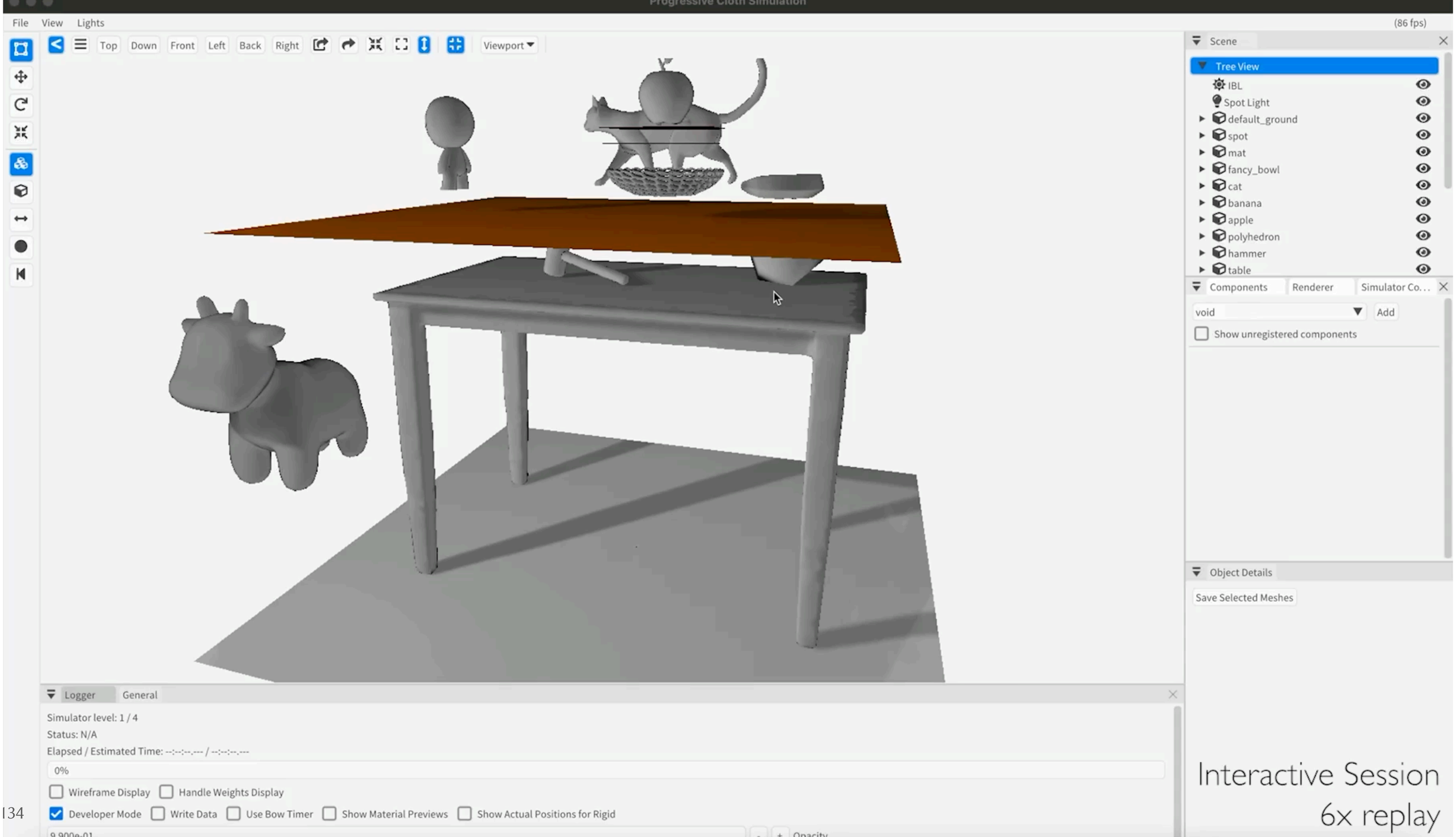
Contact-Safe State

Contact and Strain Limiting-Safe State

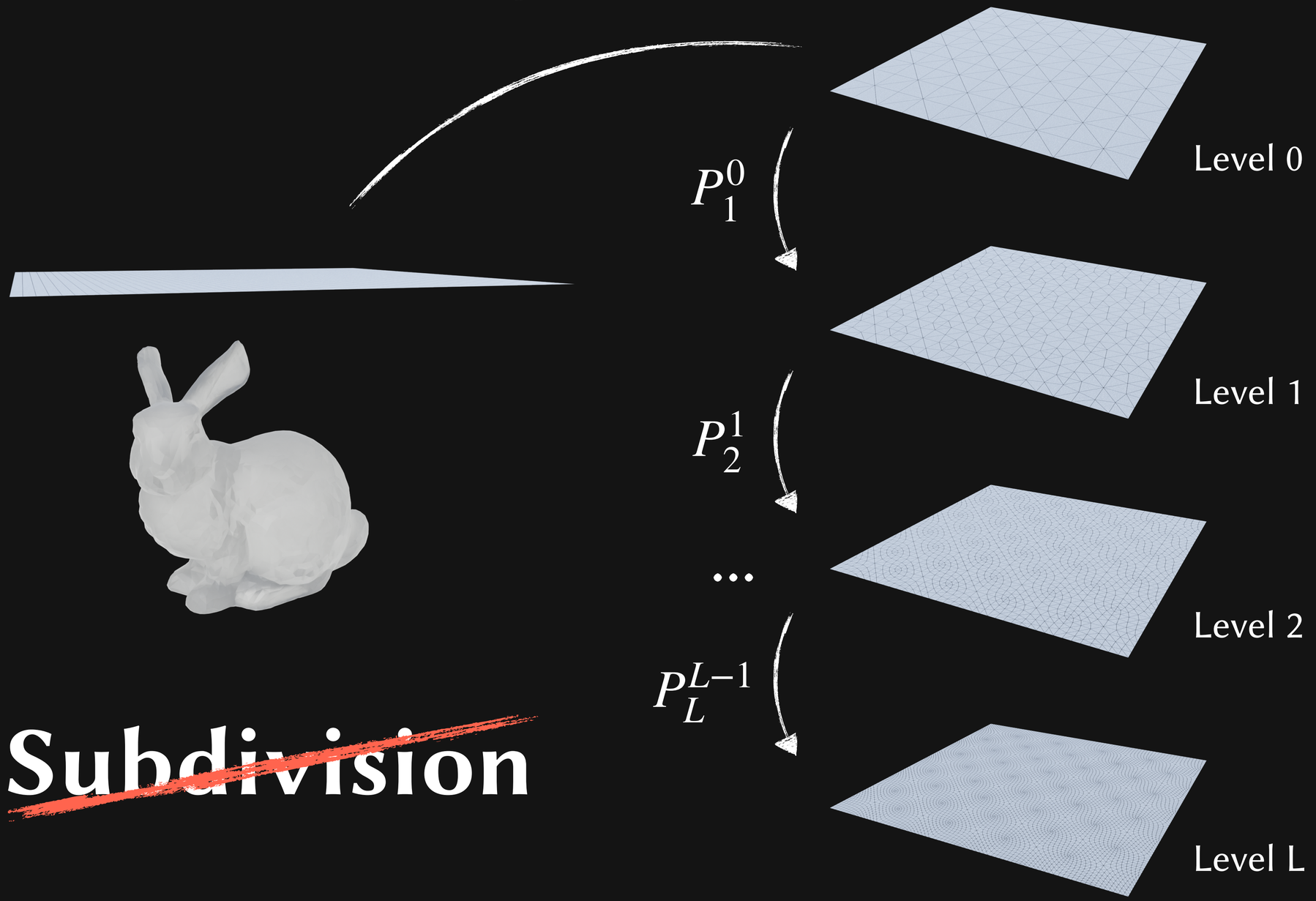

File
View
Lights
(60 fps)
Top
Down
Front
Left
Back
Right
Viewport
Components
Renderer
Scene
void
Add
Show unregistered components
TreeNode
Num Children: 0
Parent
None
First Child
None
Previous Sib
None
Next Sibling
None
Name
sphere
Name
Transform
Local:
-0.007
0.020
-0.000
Position
-0.000
0.000
-0.000
Rotation
Global:
-0.007
0.020
-0.000
Position
-0.000
0.000
-0.000
Rotation
Viewport:
1.000
1.000
Scale
0.000
0.000
Translation
Simulator Co...
Physics Para...
Create Rigid B...
Logger
Simulator Status
Simulator level: 3 / 4, status: optimizing
Elapsed / Estimated Time: 00:00:02.448 / 00:00:02.448
2%


Progressive Cloth Simulation
File
View
Lights
(79 fps)
Top
Down
Front
Left
Back
Right
Viewport
Scene
Keyboard Shortcuts
<Space + C> add cloth
<A> hold and click on vertices to nudge them
<Space + A> hold and click on vertices to add permanent handles
<Space + O> add rigidbody
<Space + G> gravity on/off
<Space + ,> decrease the affected range of selected handles
<Space + Q> decrease simulation quality
<Space + K> decrease the shape preservation around selected hanc
<Shift + Space + D> remove selected objects
<Space + D> remove selected handles
<H> hold to group-select handles
<Space + .> increase the affected range of selected handles
<Space + E> increase simulation quality
<Space + L> increase the shape preservation around selected handl
<C> move the pivots of selected clothes to their centers
<Space + T> reposition selected handles
<Shift + Space + T> reset selected objects
<Space + W> wireframe on/off
<Space + S> turn on/off the simulation of a specific object
<Ctrl + Space> turn on/off the physics
<Space + U> undo to the last converged
Components
Renderer
Simulator Control
Use Physics
Quality
5
Max Resolution
High
Final Quality
Parameter
Handle
Layout
Undo to the Last Converged
Object Details
circle
Woven Wool
Presets
3.180e-02
Thickness (cm)
4.726e-01
Density (g/cm^
8.210e+06
Stiffness (dyne
1.000e+00
Bending (unitl
Simulated
Appearance
Logger
Status
Levels of refinement: 5 / 5
Process: Done
Elapsed / Estimated Time: --:--:--.--- / --:--:--.---
0%
Display
General
New Scene
Load Scene
Developer Mode
Centimeters
Write Unit
Write Data
Write Mesh in Local Space

Twisted Ribbon


Progressive Cloth Simulation
File
View
Lights
(86 fps)
Top
Down
Front
Left
Back
Right
Viewport
Scene
Tree View
IBL
Spot Light
default_ground
spot
mat
fancy_bowl
cat
banana
apple
polyhedron
hammer
table
Components
Renderer
Simulator Co...
void
Add
Show unregistered components
Object Details
Save Selected Meshes
Logger
General
Simulator level: 1 / 4
Status: N/A
Elapsed / Estimated Time: --:--:--.--- / --:--:--.---
0%
Wireframe Display
Handle Weights Display
Developer Mode
Write Data
Use Bow Timer
Show Material Previews
Show Actual Positions for Rigid
Opacity
Interactive Session
6x replay

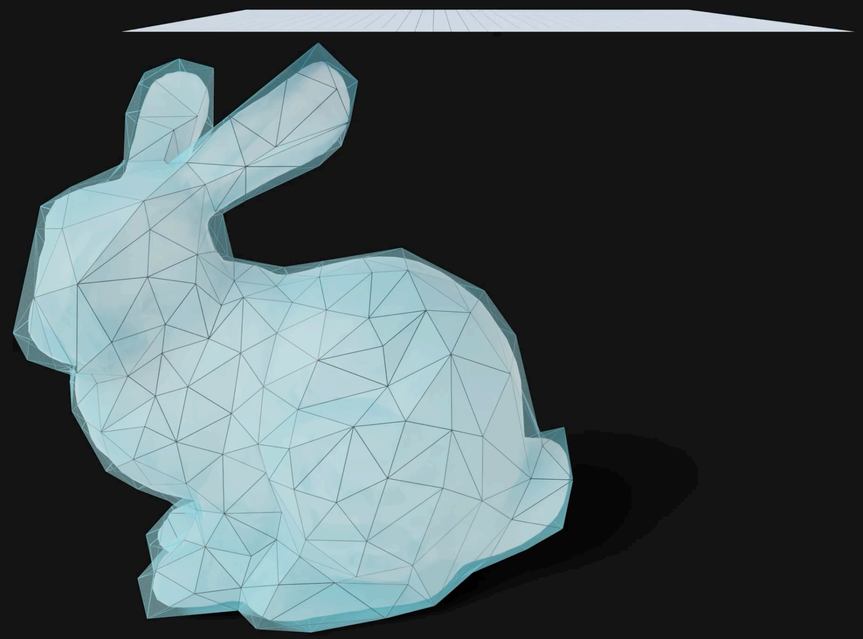
Coarse-to-Fine Hierarchy



Subdivision

Coarse-to-Fine Hierarchy

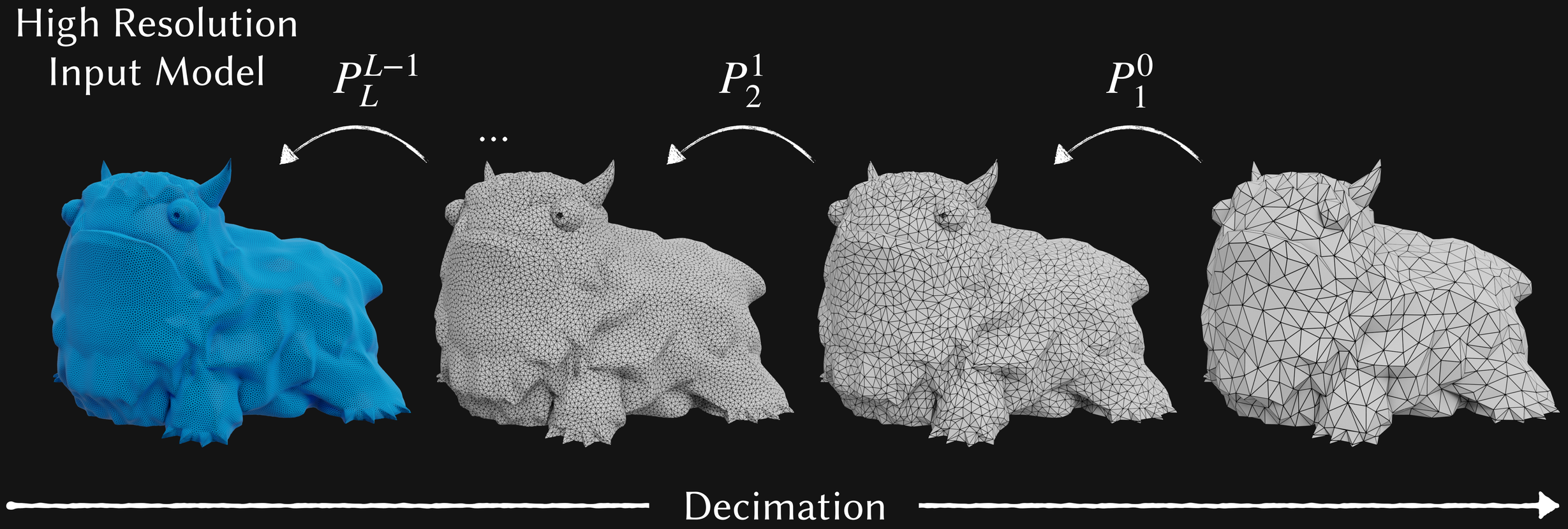
Surface **not** representable by repeated loop subdivision of a base mesh



Nested Collision Objects

Progressive Shell Quasistatics

Fine-to-Coarse Hierarchy



Two Key Ingredients

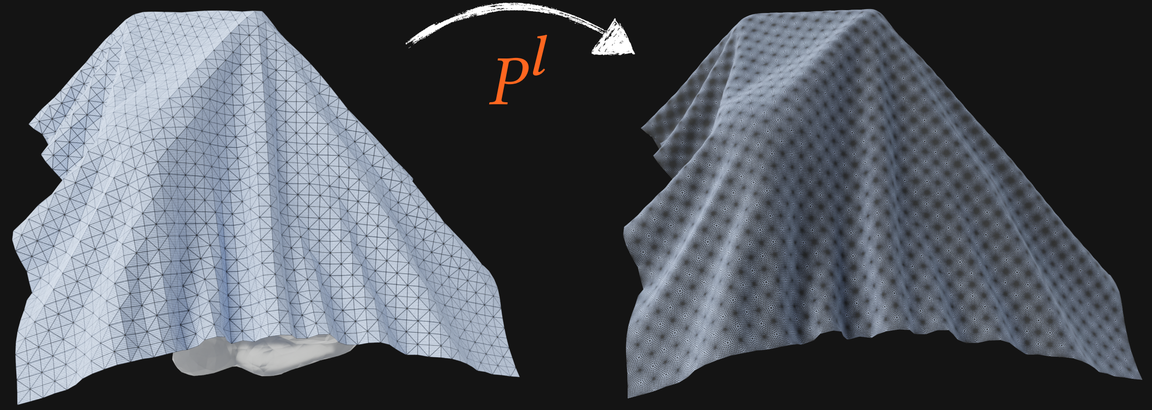
$$x_l^* = \underset{x_l}{\operatorname{argmin}}\, F_l(x_l, \bar{x}_l, P^l)$$

where $F_l = C_l(x_L) + \Psi_l(P^l(x_l))$

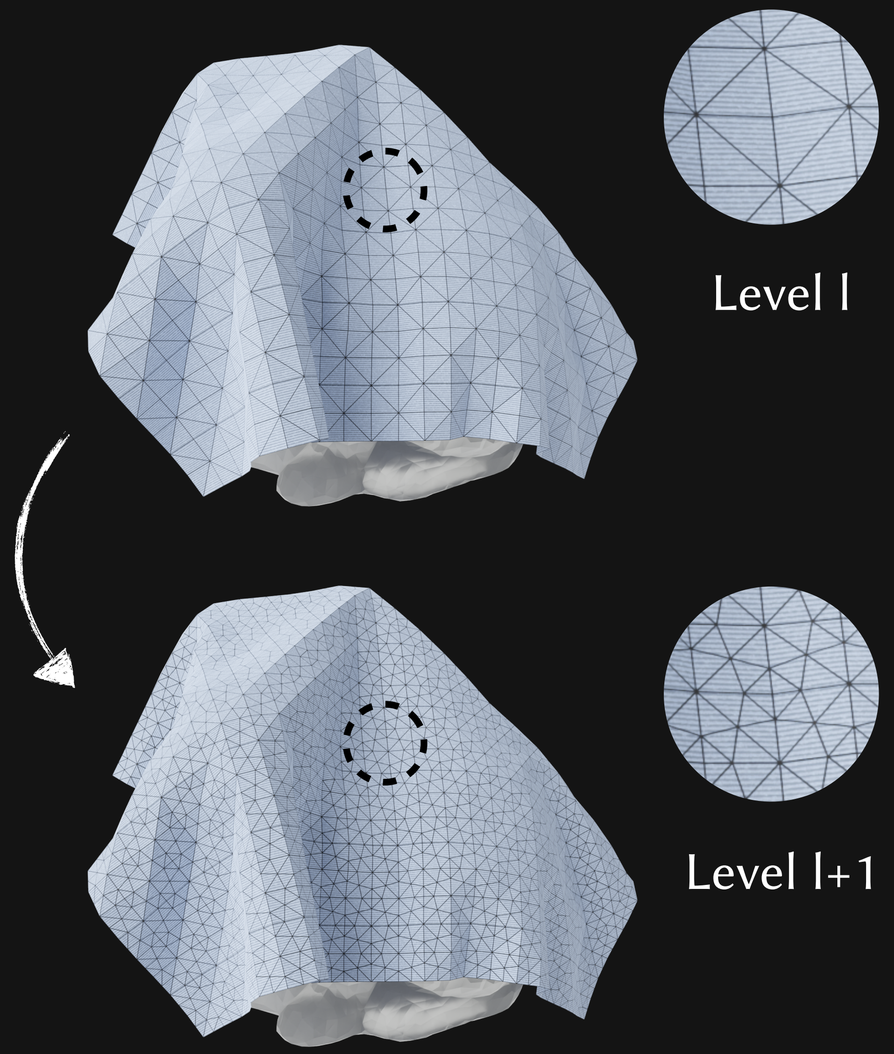
Barrier-Based Terms Shell Elastic Energies

Prolong

P^l

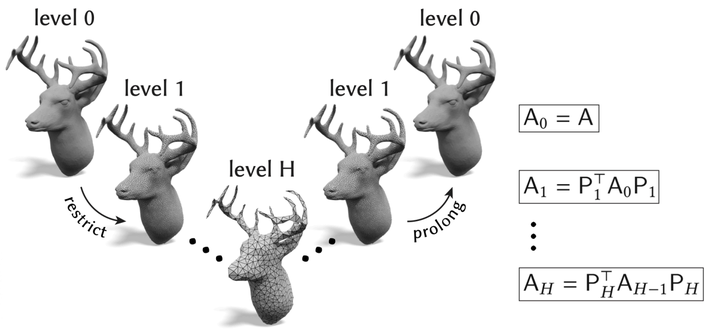


Coarse-Level Proxy Energies

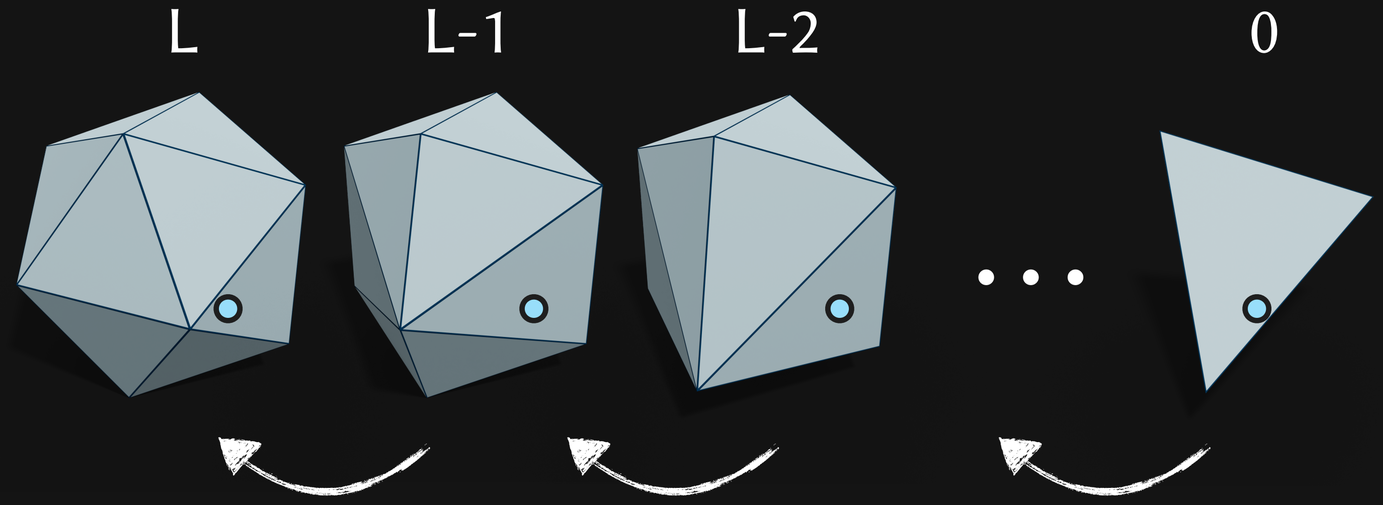


Safe Initialization

Intrinsic Prolongation



Intrinsic Prolongation
[Liu et al. 2021]



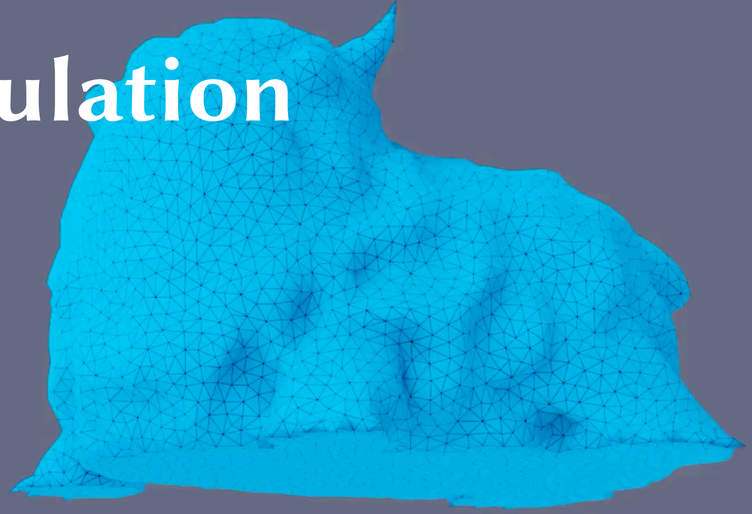
Forward decimation

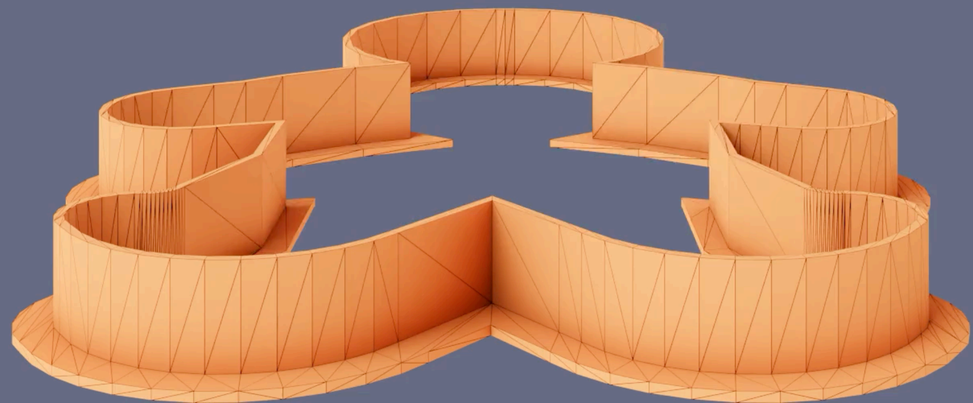
Bijective map

#V: 50K

#V: 500

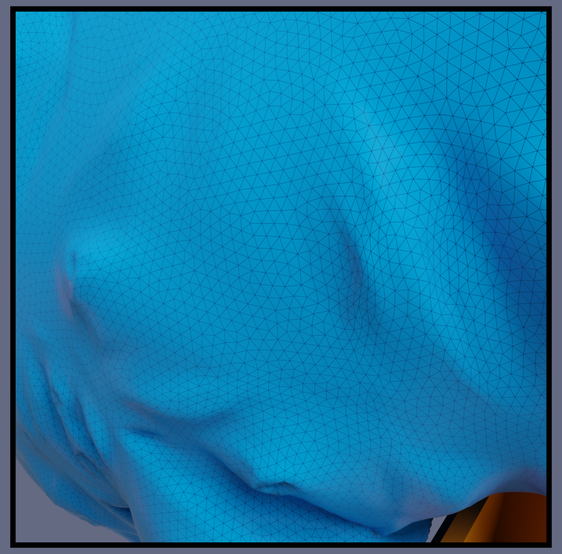
Quasistatic Preview Simulation

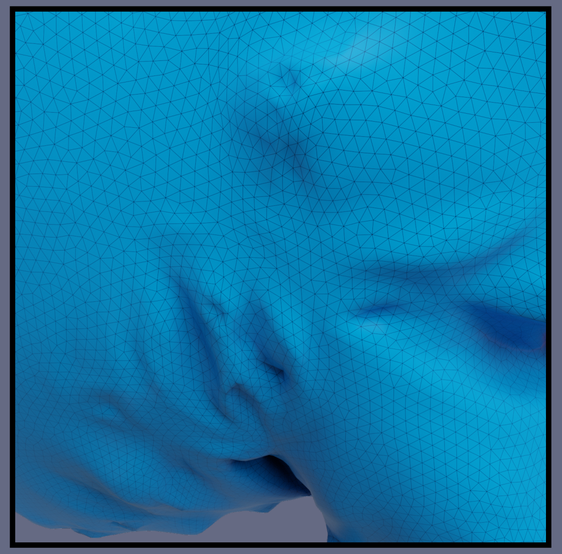


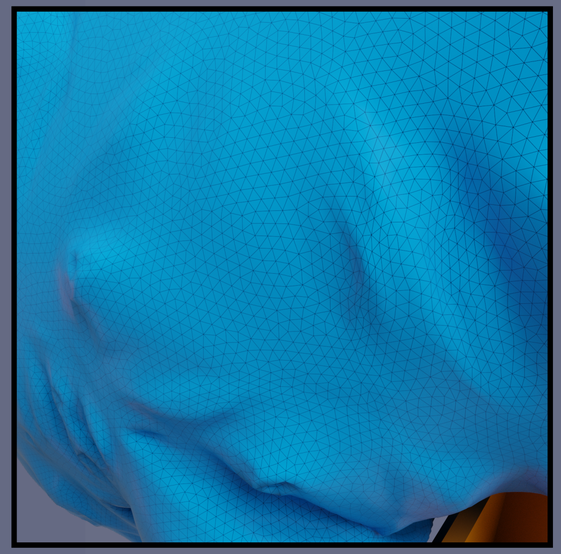


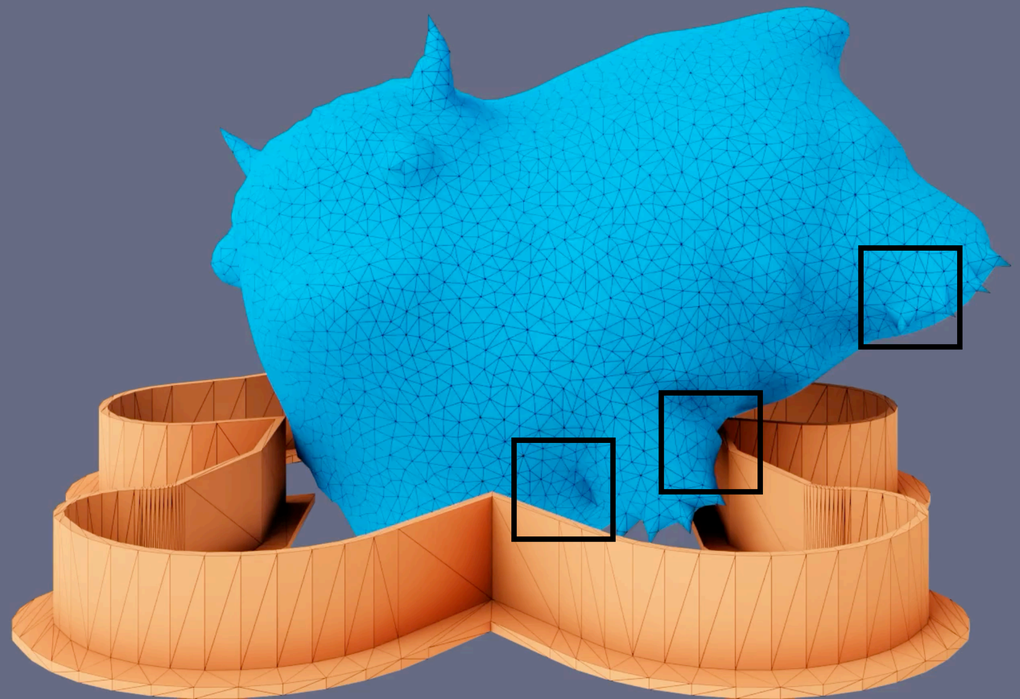
4X Playback

Progressive Refinement

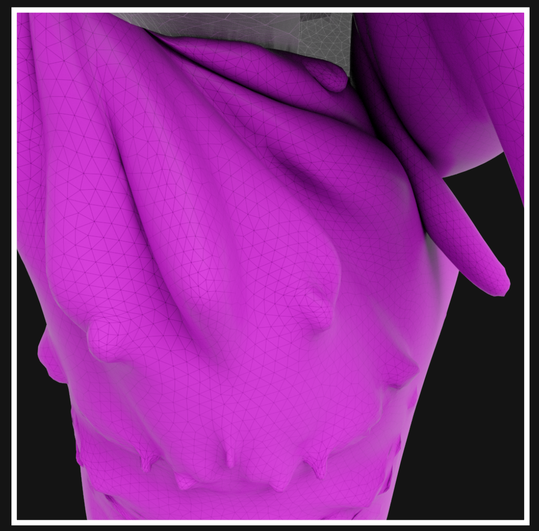


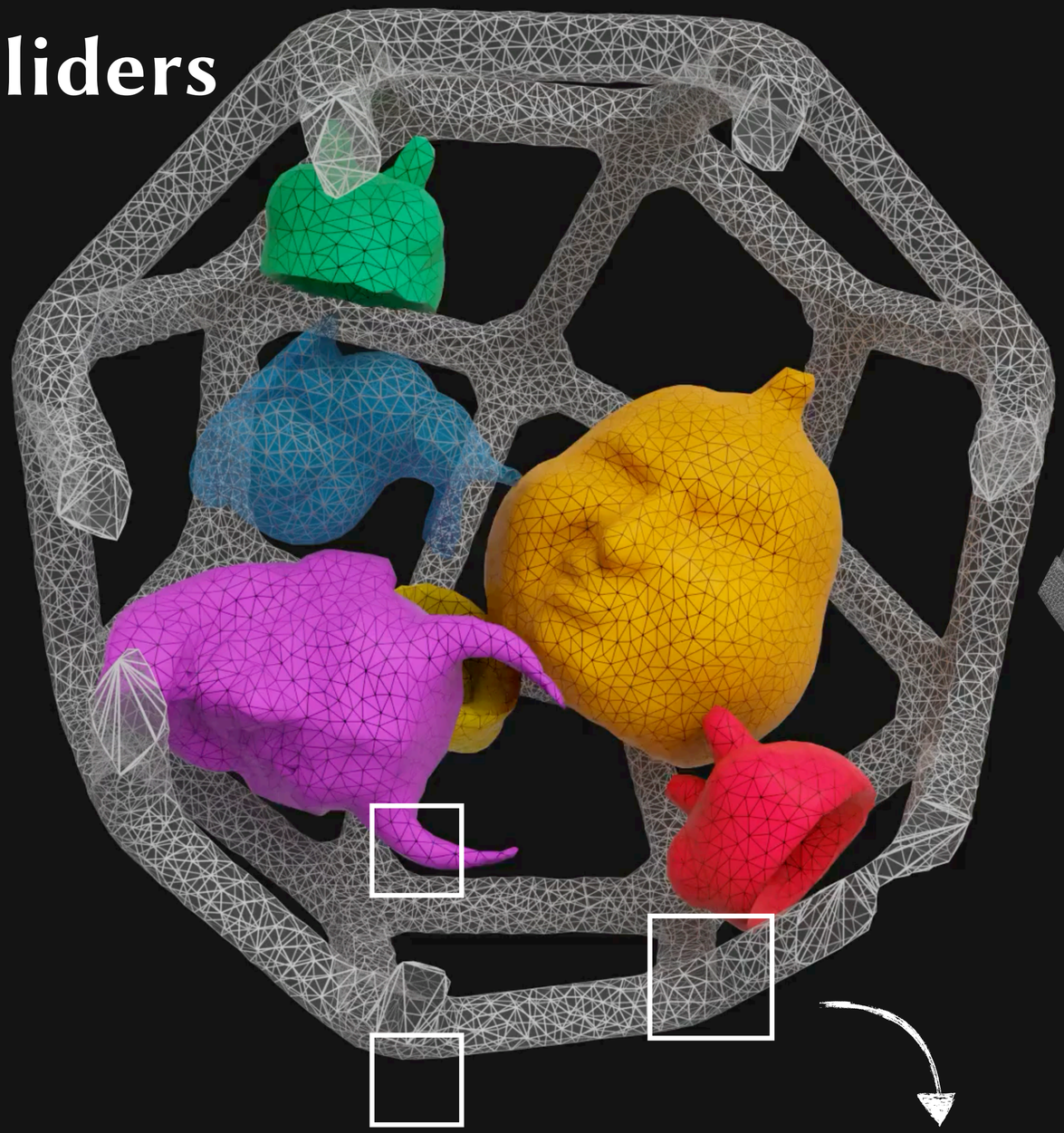


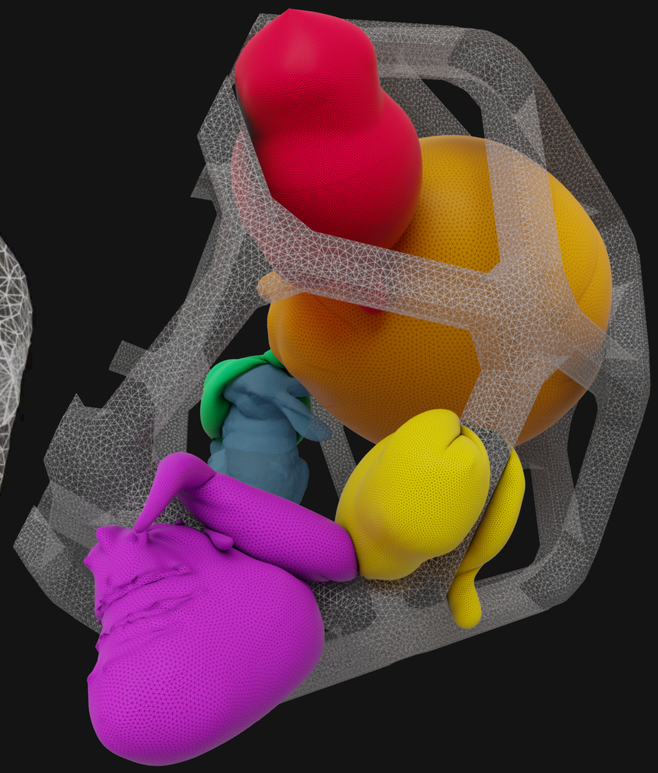




Progressive Colliders



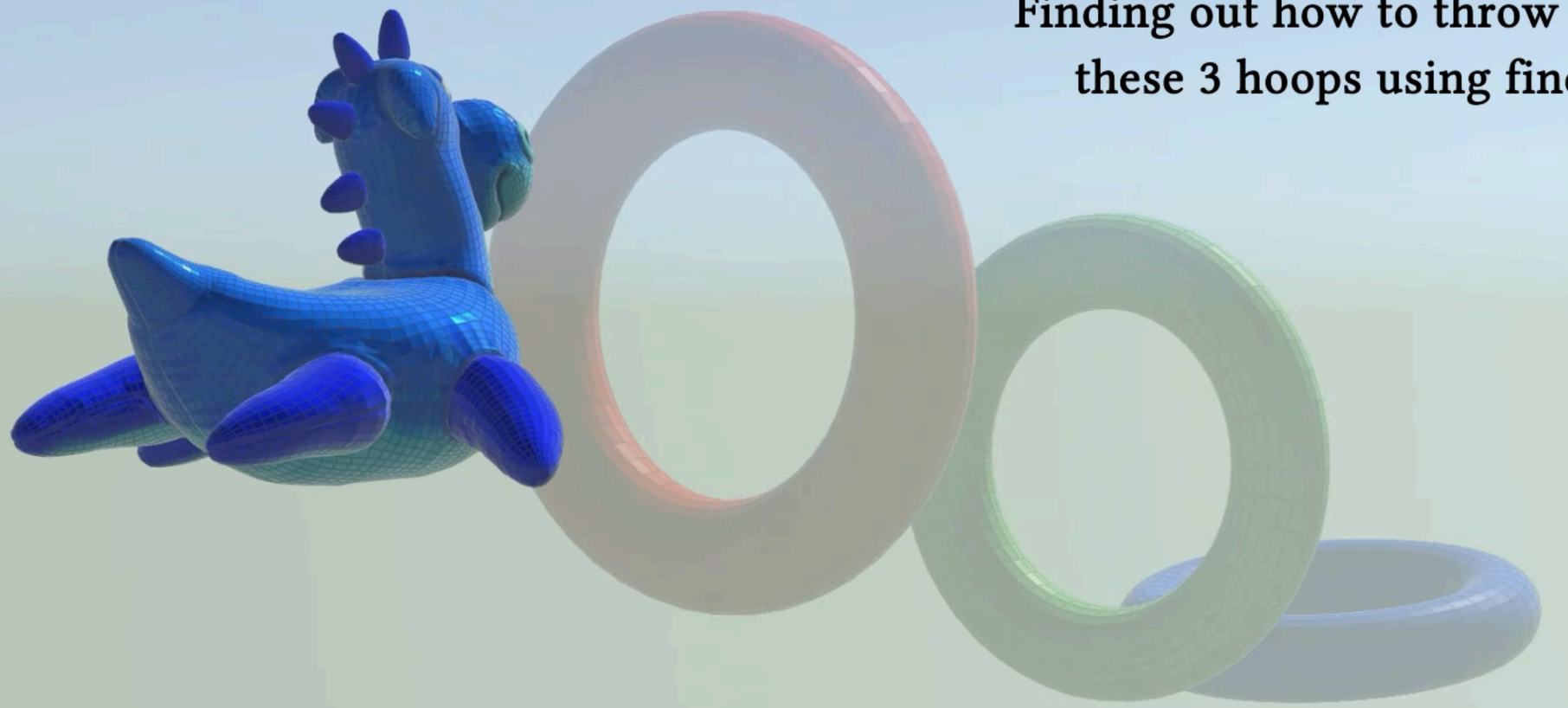




Back View

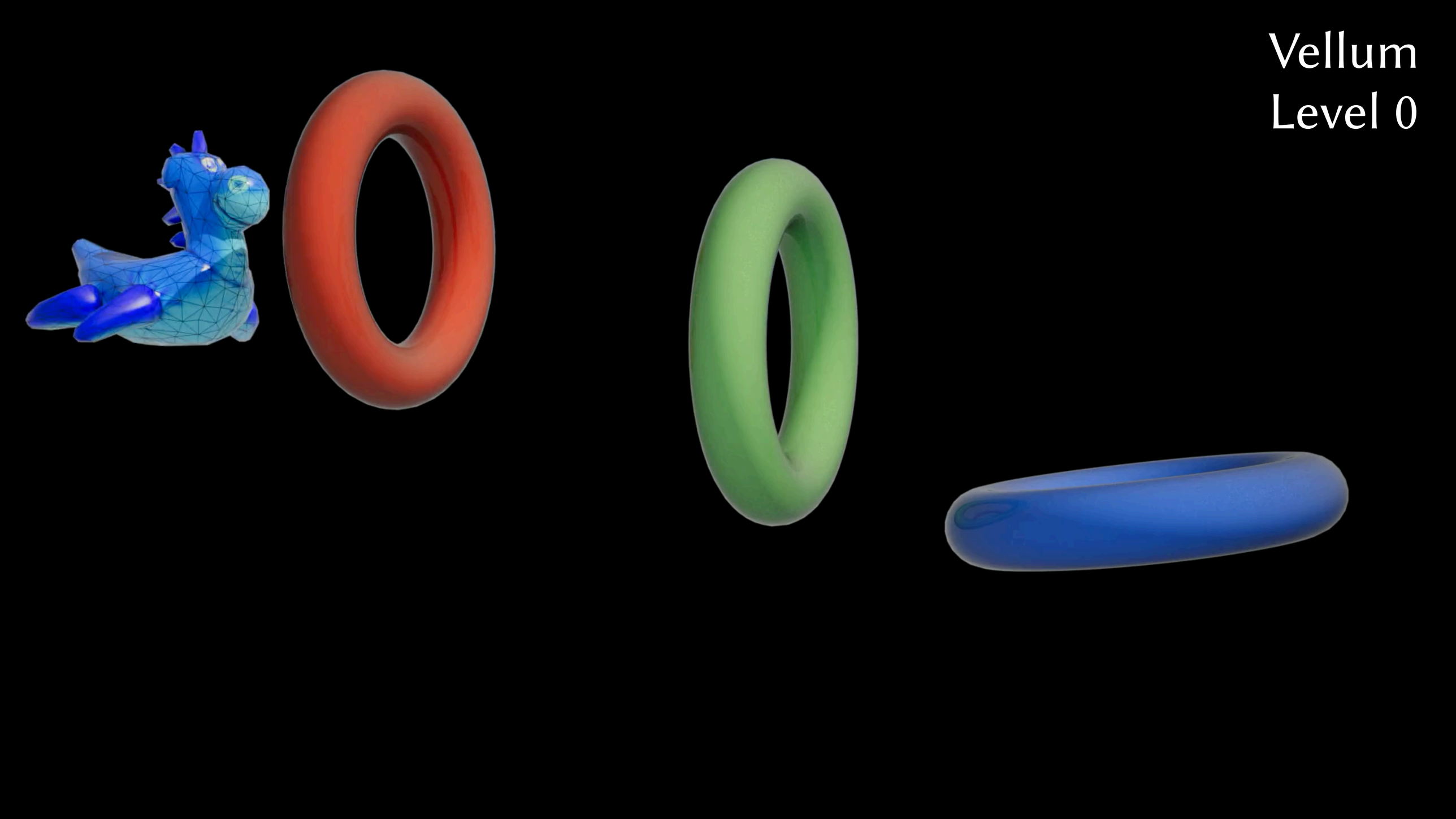
Refining fixed collision geometries!

Motivation: Motion Design

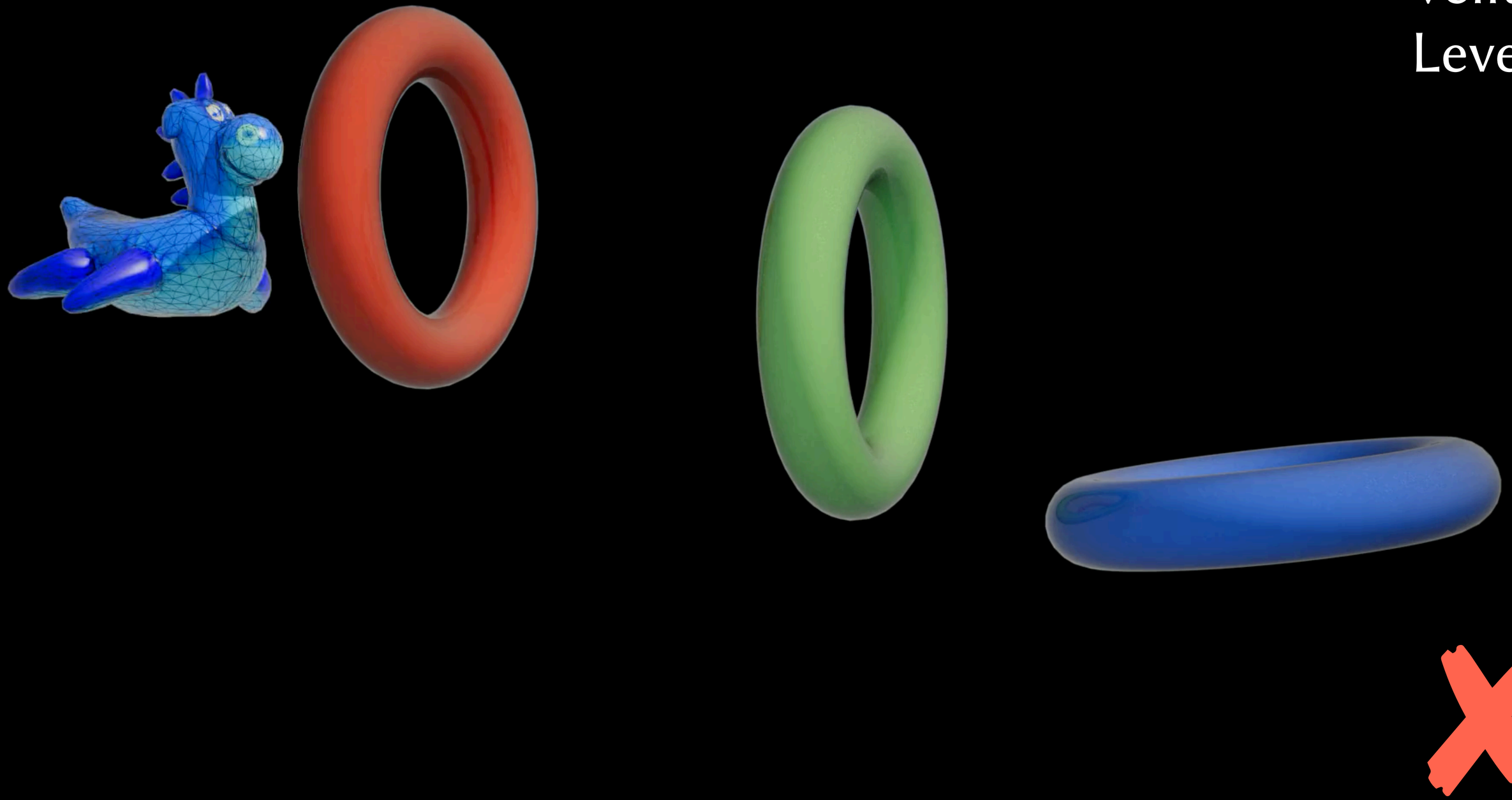


Finding out how to throw this inflatable toy through these 3 hoops using fine-res simulation is ***tedious***.

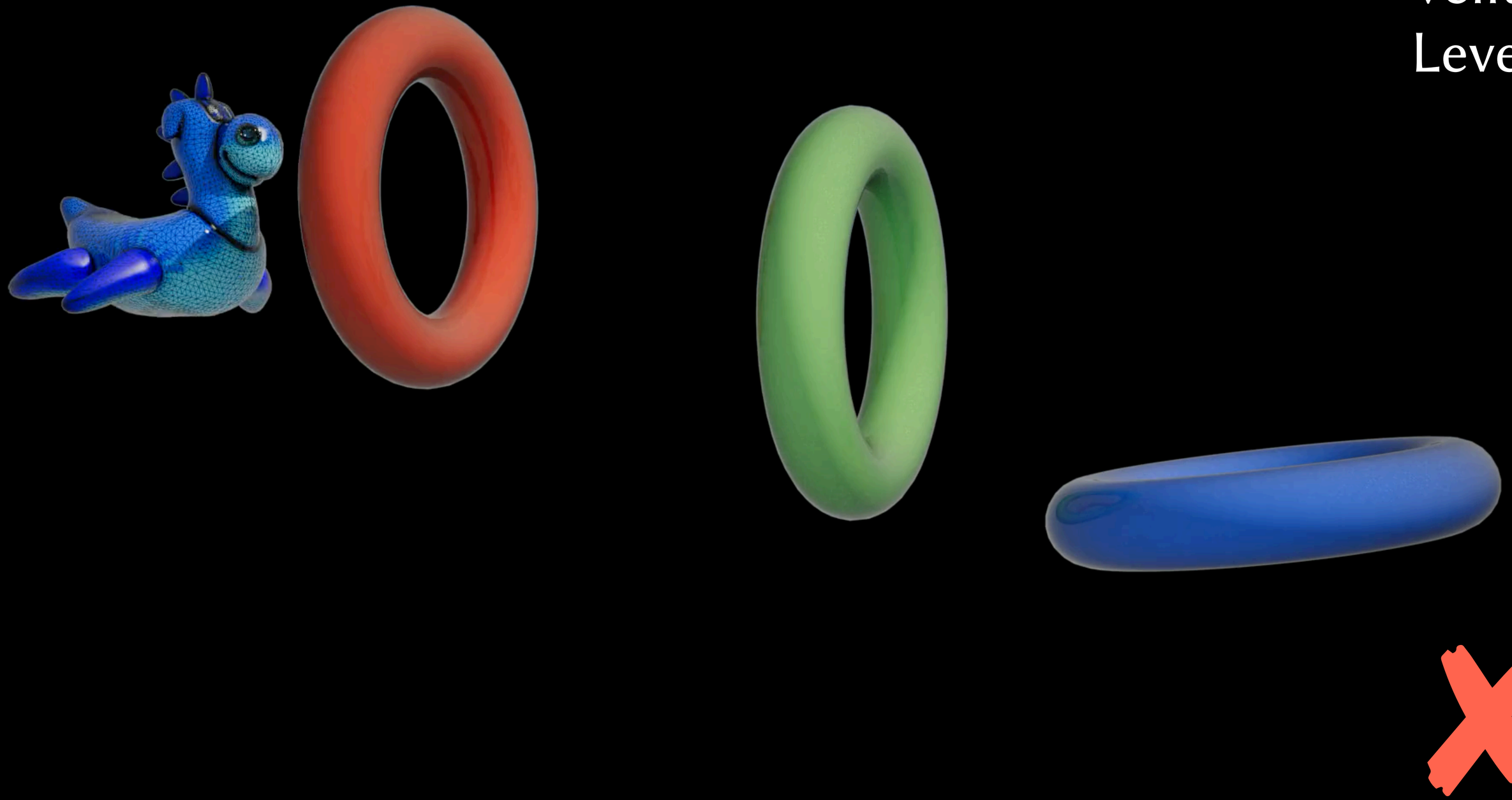
Houdini Vellum


Vellum
Level 0

Vellum
Level 1



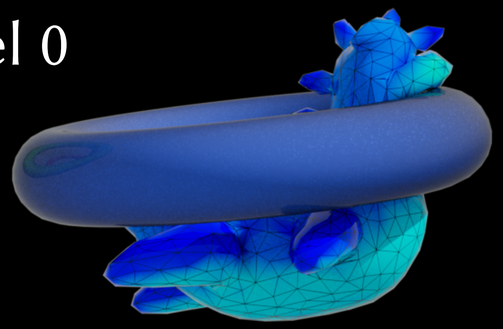
Vellum
Level 2



Time

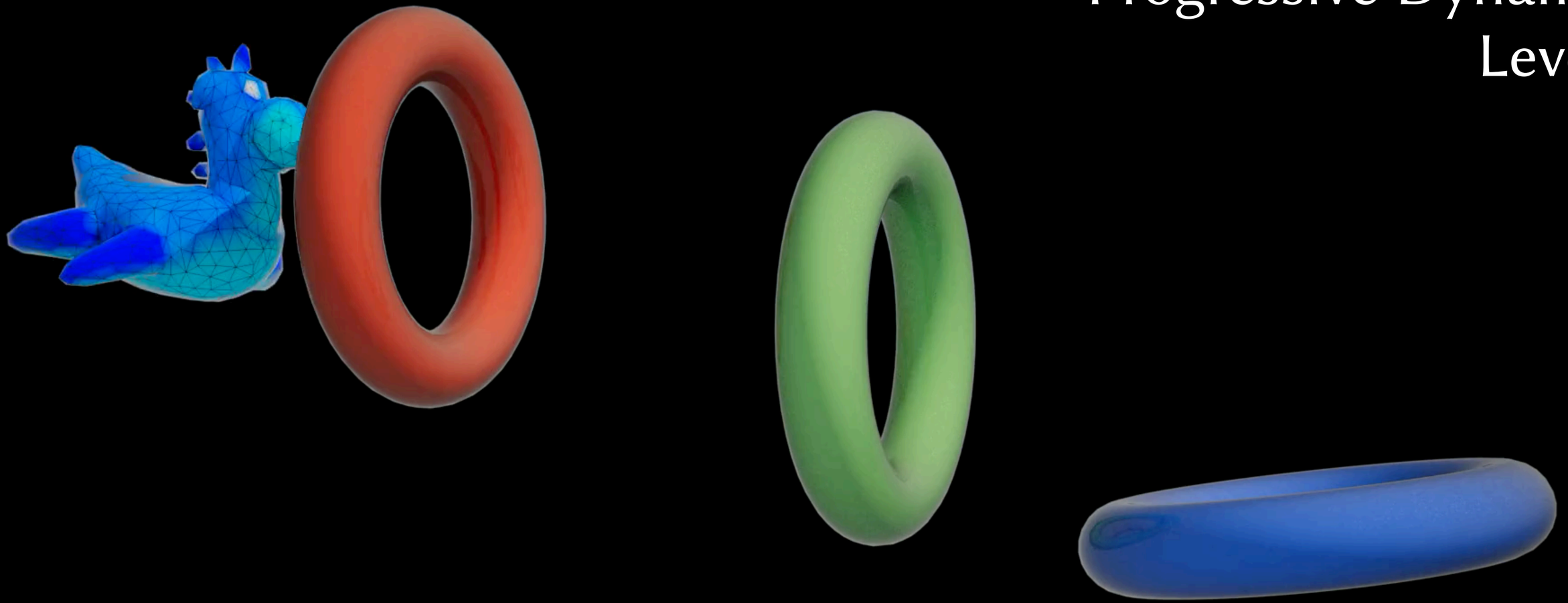
?

Level 0

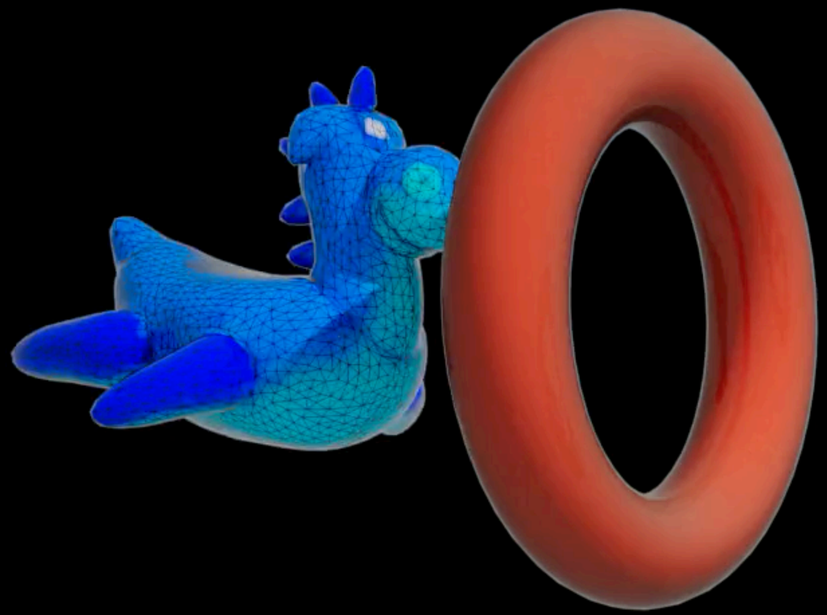


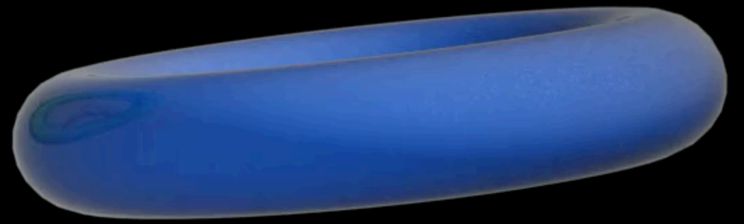
Zhang et al. 2022, 2023

Progressive Dynamics
Level 0

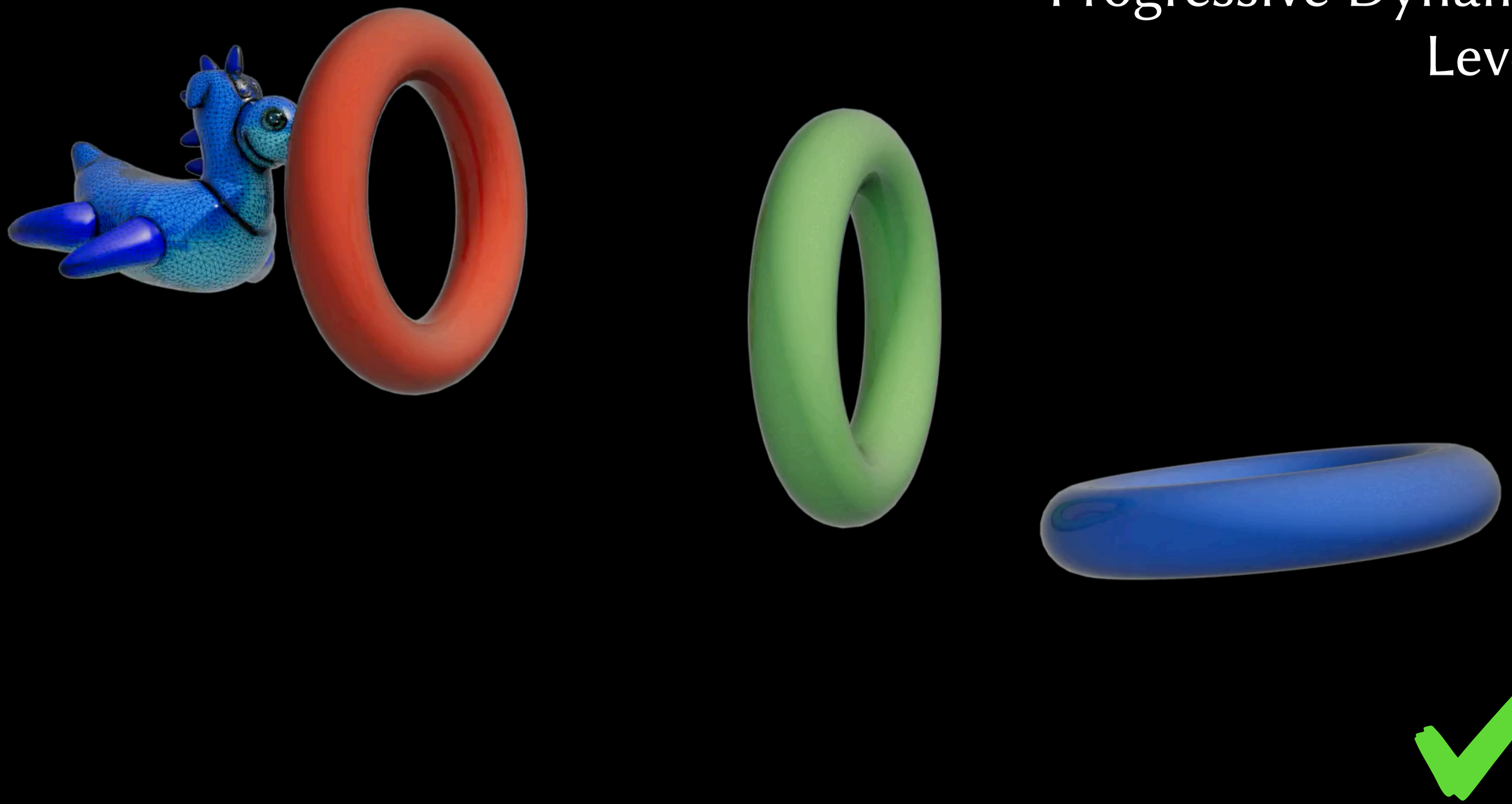


Progressive Dynamics
Level 1

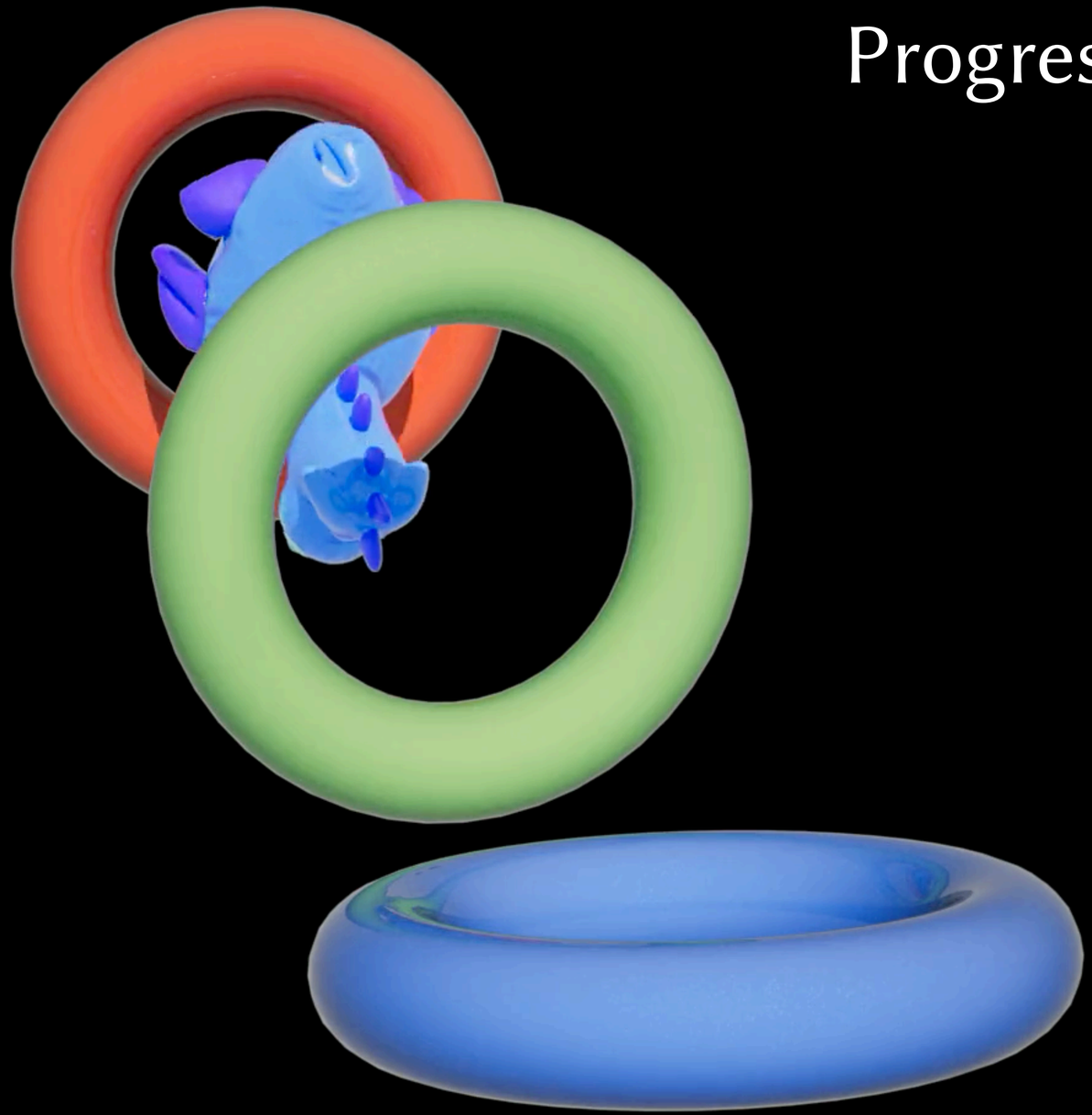




Progressive Dynamics
Level 2



Progressive Dynamics
Level 2



Zhang et al. 2024

And use coarse previews to rapidly explore layout and many physical parameters to arrive at their Final Result.

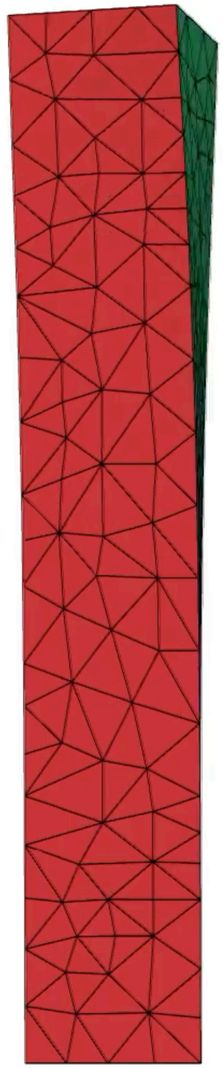
In physical simulation..

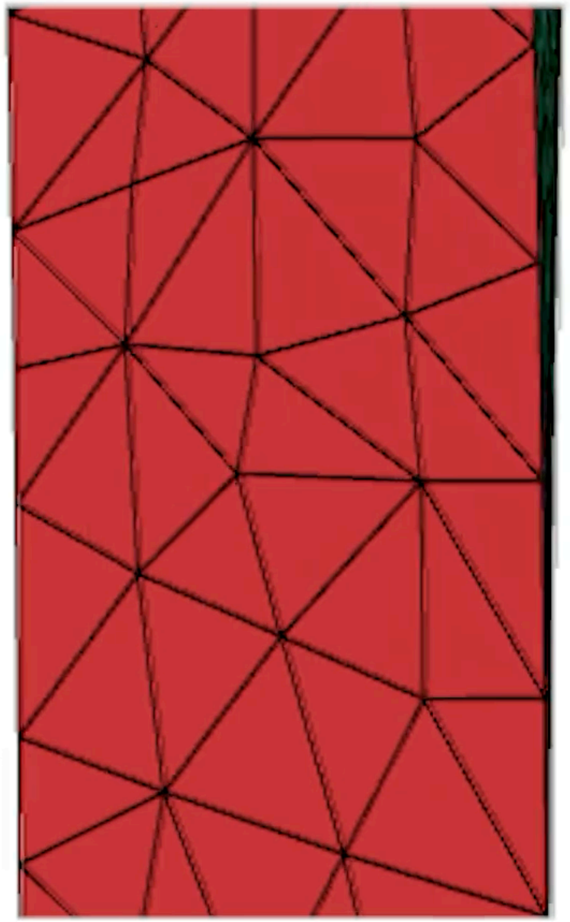
Speed

v.s.

Fidelity

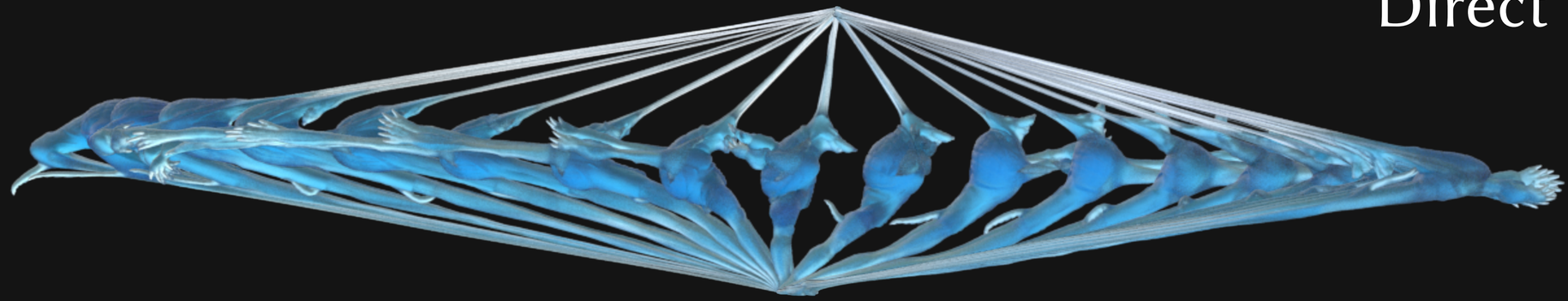
Adaptive Simulation





Ferguson et al. 2023

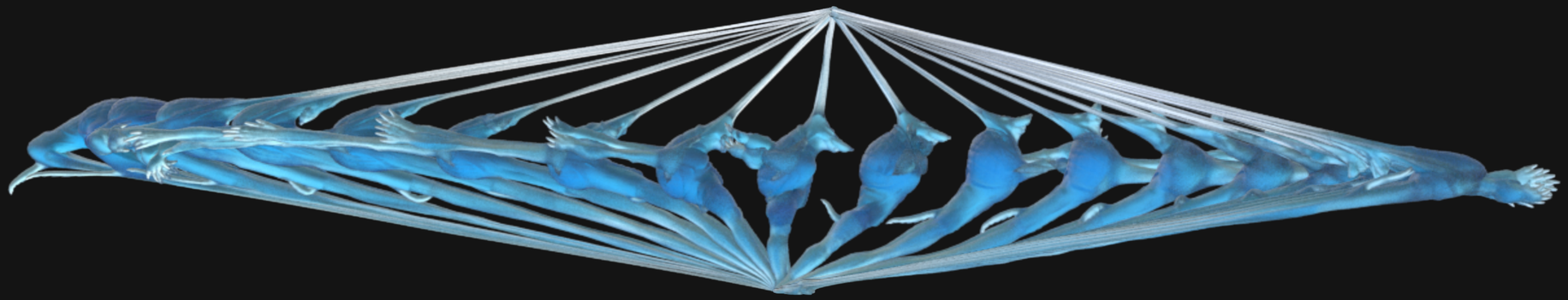
Direct Simulation



Time

Art-directability!

- Constrained System
- Reduced-Order System
- Multilevel System
- Adaptive System
- ...



Time

Thank You!

Questions?

Image sources:

- https://r-wong253249-sp.blogspot.com/2013/11/pose-to-pose-and-frame-by-frame-research.html
- https://dreamfarmstudios.com/blog/what-is-3d-rigging/
- https://drive.google.com/file/d/1oxeQ9L_DX_u_3nig-DMoW_GHZGO1Sva9/preview
- https://medium.com/@jaleeladejumo/gradient-descent-from-scratch-batch-gradient-descent-stochastic-gradient-descent-and-mini-batch-def681187473
- https://academic-accelerator.com/encyclopedia/spring-system
- https://en.wikipedia.org/wiki/Smoothed-particle_hydrodynamics
- https://duxingyi-charles.github.io/publication/lifting-simplices-to-find-injectivity/
- https://www-users.cse.umn.edu/~narain/files/admm-pd.pdf