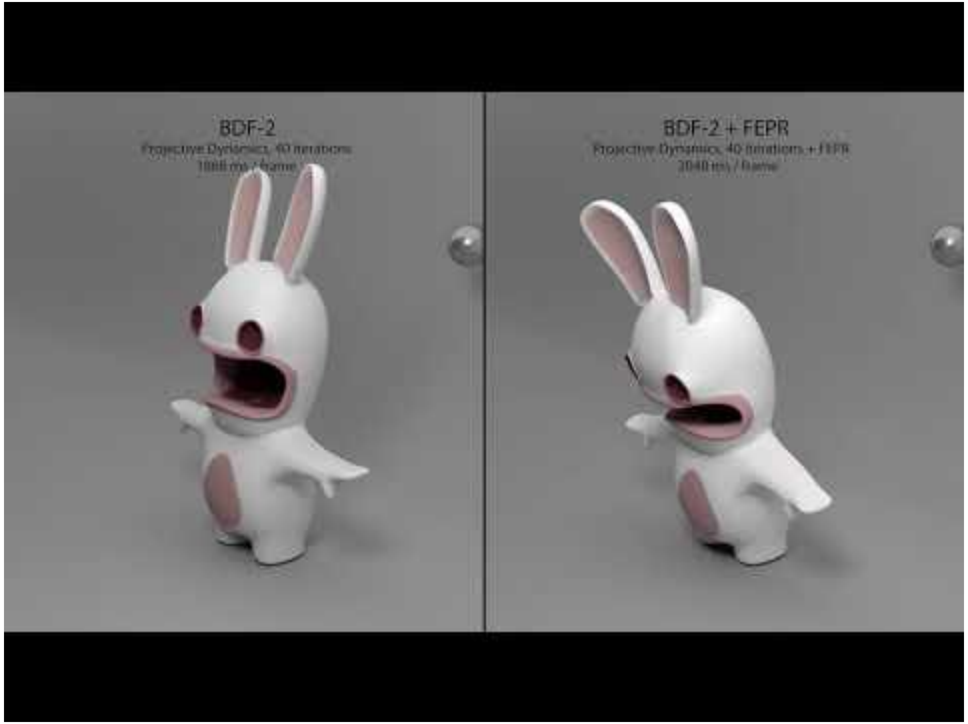
FEPR: Fast Energy Projection for Real-Time Simulation of Deformable Objects

Group 1
Christopher Vogelsanger, Patrick Eigensatz, Véronique Kaufmann

Use Cases / Issues

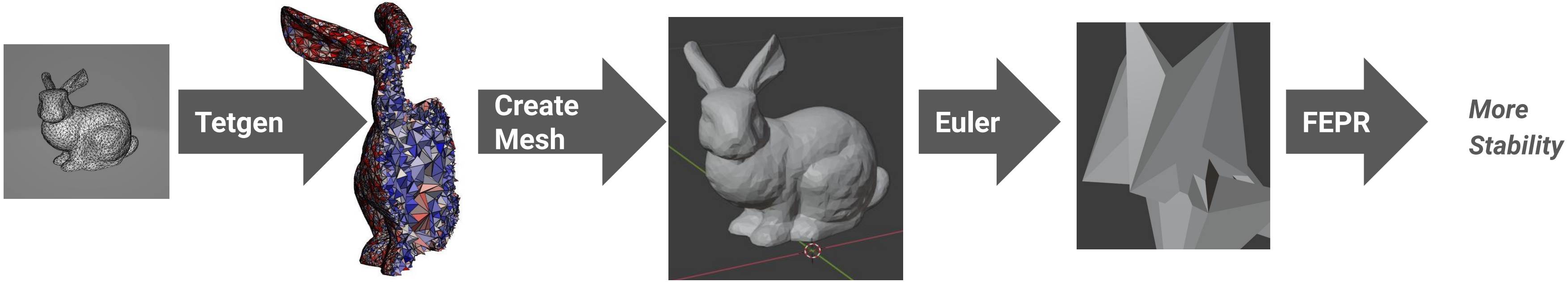
Instability



Algorithm

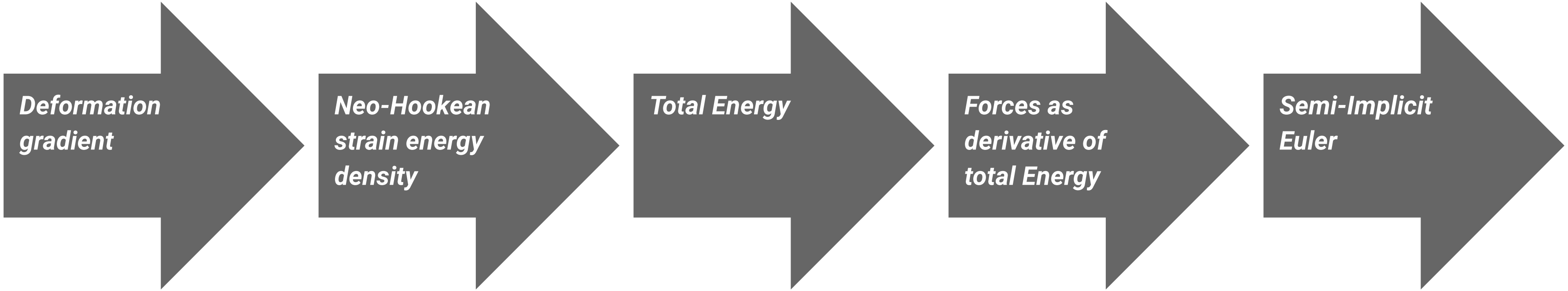
Overview

Overview



Time Integration (Euler)

Overview



Time Integration

Deformation gradient

We compute deformation gradient F using the undeformed tetrahedron edges A and the deformed tetrahedron edges B

$$F(\bar{x}, x) = B(x)A^{-1}(\bar{x}) = \begin{pmatrix} e_x^1 & e_x^2 & e_x^3 \\ e_y^1 & e_y^2 & e_y^3 \\ e_z^1 & e_z^2 & e_z^3 \end{pmatrix} \begin{pmatrix} \bar{e}_x^1 & \bar{e}_x^2 & \bar{e}_x^3 \\ \bar{e}_y^1 & \bar{e}_y^2 & \bar{e}_y^3 \\ \bar{e}_z^1 & \bar{e}_z^2 & \bar{e}_z^3 \end{pmatrix}^{-1} \text{ where } e^i = x_{i+1} - x_1 \text{ for } i \in \{1,2,3,4\}$$

Time Integration

Neo-Hookean strain energy density

Using F we compute the Neo-Hookean strain energy density [6]

$$\Psi(F) = \frac{\mu}{2}\sum_i[(F^T F)_{ii} - 1] - \mu\log(J) + \frac{\lambda}{2}\log^2(J)$$

$$\text{with } J = \det(F)$$

Time Integration

Total Energy

We compute the energy of an element by multiplying its strain energy density with its volume
The total energy is the sum over all element energies

$$E = \sum_e V_e \Psi(F_e)$$

Time Integration

Forces

Using taichi's auto-differentiation function we compute the forces of each node

$$f_i = -\frac{\partial E}{\partial x_i} = -\frac{\sum_e V_e \Psi(F_e)}{\partial x_i}$$

Time Integration

Semi-Implicit Euler

Finally we use the semi-implicit euler to update each nodes velocity and position

$$v_{t+1} = v_t + \Delta t \frac{f_t}{m}$$

$$x_{t+1} = x_t + \Delta t v_{t+1}$$

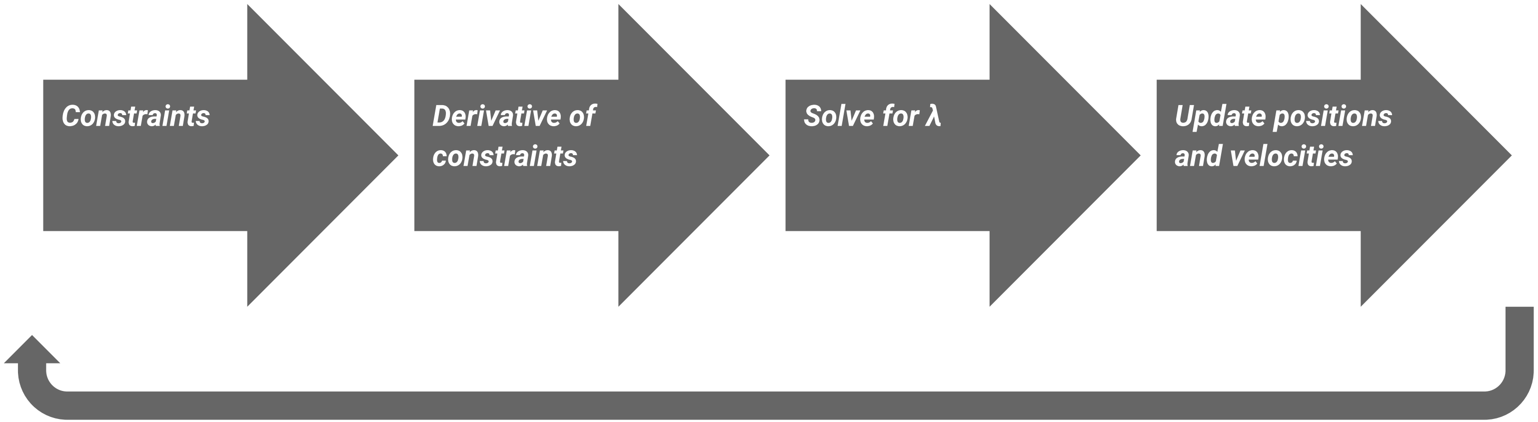
FEPR

- Implement from Paper [7]
- Use simplification that sets variables s,t to 0 and assumes momentum conservation

FEPR

Overview

Repeat until constraints are met



FEPR

Constraints

We compute constraints on total energy H ∈ R, on the total linear momentum P ∈ R^3 and the total angular momentum L ∈ R^3

We stack $c(q) \coloneqq (H, P, L) \in \mathbb{R}^7$ with $q \coloneqq \begin{pmatrix} x \\ v \\ s \\ t \end{pmatrix} \in \mathbb{R}^{6m+2}$ *where* $s = 0, t = 0$

$$H(x, v) = E(x) + \frac{1}{2} v^T M v \text{ with } E(x) = \sum_e V_e \Psi(F_e)$$

$$P(v) = \sum_i m_i v_i$$

$$L(x, v) = \sum_i x_i \times m_i v_i$$

FEPR

Derivative of Constraints

$$\nabla c(q) \coloneqq (\nabla c_1(q), \dots, \nabla c_7(q)) \in \mathbb{R}^7$$

Either use taichi’s auto-differentiation (slow)

or

Compute most derivatives by hand, only use taichi for the derivatives of H (fast)

FEPR

Derivative of Constraints

$$\frac{\partial P(v)}{\partial x_i} = \begin{pmatrix} \frac{\partial P(v)_x}{\partial x_{i,x}} & \frac{\partial P(v)_y}{\partial x_{i,x}} & \frac{\partial P(v)_z}{\partial x_{i,x}} \\ \frac{\partial P(v)_x}{\partial x_{i,y}} & \frac{\partial P(v)_y}{\partial x_{i,y}} & \frac{\partial P(v)_z}{\partial x_{i,y}} \\ \frac{\partial P(v)_x}{\partial x_{i,z}} & \frac{\partial P(v)_y}{\partial x_{i,z}} & \frac{\partial P(v)_z}{\partial x_{i,z}} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\frac{\partial L(x,v)}{\partial x_i} = \begin{pmatrix} \frac{\partial L(x,v)_x}{\partial x_{i,x}} & \frac{\partial L(x,v)_y}{\partial x_{i,x}} & \frac{\partial L(x,v)_z}{\partial x_{i,x}} \\ \frac{\partial L(x,v)_x}{\partial x_{i,y}} & \frac{\partial L(x,v)_y}{\partial x_{i,y}} & \frac{\partial L(x,v)_z}{\partial x_{i,y}} \\ \frac{\partial L(x,v)_x}{\partial x_{i,z}} & \frac{\partial L(x,v)_y}{\partial x_{i,z}} & \frac{\partial L(x,v)_z}{\partial x_{i,z}} \end{pmatrix} = \begin{pmatrix} 0 & -m_i v_{i,z} & m_i v_{i,y} \\ m_i v_{i,z} & 0 & -m_i v_{i,x} \\ -m_i v_{i,y} & m_i v_{i,x} & 0 \end{pmatrix}$$

$$\frac{\partial P(v)}{\partial v_i} = \begin{pmatrix} \frac{\partial P(v)_x}{\partial v_{i,x}} & \frac{\partial P(v)_y}{\partial v_{i,x}} & \frac{\partial P(v)_z}{\partial v_{i,x}} \\ \frac{\partial P(v)_x}{\partial v_{i,y}} & \frac{\partial P(v)_y}{\partial v_{i,y}} & \frac{\partial P(v)_z}{\partial v_{i,y}} \\ \frac{\partial P(v)_x}{\partial v_{i,z}} & \frac{\partial P(v)_y}{\partial v_{i,z}} & \frac{\partial P(v)_z}{\partial v_{i,z}} \end{pmatrix} = \begin{pmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & m_i \end{pmatrix}$$

$$\frac{\partial L(x,v)}{\partial v_i} = \begin{pmatrix} \frac{\partial L(x,v)_x}{\partial v_{i,x}} & \frac{\partial L(x,v)_y}{\partial v_{i,x}} & \frac{\partial L(x,v)_z}{\partial v_{i,x}} \\ \frac{\partial L(x,v)_x}{\partial v_{i,y}} & \frac{\partial L(x,v)_y}{\partial v_{i,y}} & \frac{\partial L(x,v)_z}{\partial v_{i,y}} \\ \frac{\partial L(x,v)_x}{\partial v_{i,z}} & \frac{\partial L(x,v)_y}{\partial v_{i,z}} & \frac{\partial L(x,v)_z}{\partial v_{i,z}} \end{pmatrix} = \begin{pmatrix} 0 & m_i x_{i,z} & -m_i x_{i,y} \\ -m_i x_{i,z} & 0 & m_i x_{i,x} \\ m_i x_{i,y} & -m_i x_{i,x} & 0 \end{pmatrix}$$

FEPR

Solve for λ

We solve the following equation for λ

$$\left(\nabla c\left(q^{(k)}\right)^{T} D^{-1}\nabla c\left(q^{(k)}\right)\right)\lambda^{(k+1)} = \nabla c\left(q^{(k)}\right) \;\; where\ D = diag(M; (dt)^2 M; \epsilon; \epsilon) \;\; and\ \epsilon = 0.001$$

FEPR

Update Positions and Velocities

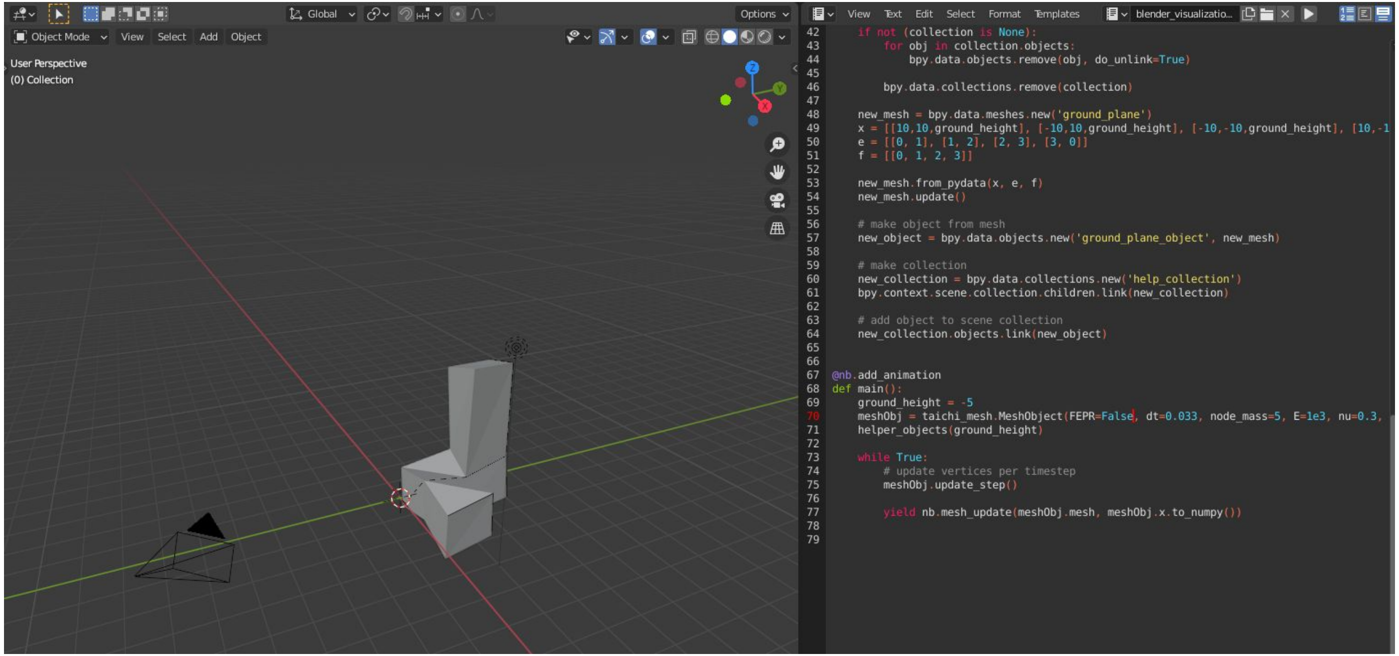
To ensure stability we scale λ and use it to update the positions and velocities

$$q^{(k+1)} = q^{(k)} - D^{-1} \nabla c\left(q^{(k)}\right) \lambda^{(k+1)}$$

With the updated values we recompute the constraints and stop once they are fulfilled

Implementation

Setup


Global
Options
Object Mode
View
Select
Add
Object
User Perspective
(0) Collection
View
Text
Edit
Select
Format
Templates
blender_visualizatio...
    if not (collection is None):
        for obj in collection.objects:
            bpy.data.objects.remove(obj, do_unlink=True)

        bpy.data.collections.remove(collection)

    new_mesh = bpy.data.meshes.new('ground_plane')
    x = [[10,10,ground_height], [-10,10,ground_height], [-10,-10,ground_height], [10,-1
    e = [[0, 1], [1, 2], [2, 3], [3, 0]]
    f = [[0, 1, 2, 3]]

    new_mesh.from_pydata(x, e, f)
    new_mesh.update()

    # make object from mesh
    new_object = bpy.data.objects.new('ground_plane_object', new_mesh)

    # make collection
    new_collection = bpy.data.collections.new('help_collection')
    bpy.context.scene.collection.children.link(new_collection)

    # add object to scene collection
    new_collection.objects.link(new_object)

@nb.add_animation
def main():
    ground_height = -5
    meshObj = taichi_mesh.MeshObject(FEPR=False, dt=0.033, node_mass=5, E=1e3, nu=0.3,
    helper_objects(ground_height)

    while True:
        # update vertices per timestep
        meshObj.update_step()

        yield nb.mesh_update(meshObj.mesh, meshObj.x.to_numpy())

Tools

- Taichi
- Taichi-Blend for visualization of the simulation in blender
- TetGen for Discretization into Tetraeders

Improvements

- Avoiding zero Determinants
- Upper bound on FEPR impact
  - See Demo 2 for example on “almost too much” damping

Issues and Challenges

Issue 1: Maximum Nodes in Taichi

Taichi uses a node structure to store meshes. [1], [2].

- It turns out that Taichi does not support using large meshes in these s-nodes because of a hardcoded upper limit. [3]

Solution: Use small meshes for our demo

- Examples of meshes that were too large: elephant, bunny

Issue 2: Derivatives

Taichi offers an autodiff system with some limitations

- Taichi only supports first-order derivatives [4]
- Taichi only supports tracking of scalar fields [5]
- Using pytorch is incompatible with the taichi scope and kernels

Solution: Track separate fields, treat each entry of the constraints we needed the gradients of as a separate variable. Very error-prone.

Issue 3: Performance

Tracking each values derivative separately causes computation time to grow.

Solution:

Compute most gradients by hand to avoid taichi tape overhead (up to 7 times faster)

Issue 4: Large Vectors

Taichi warning:

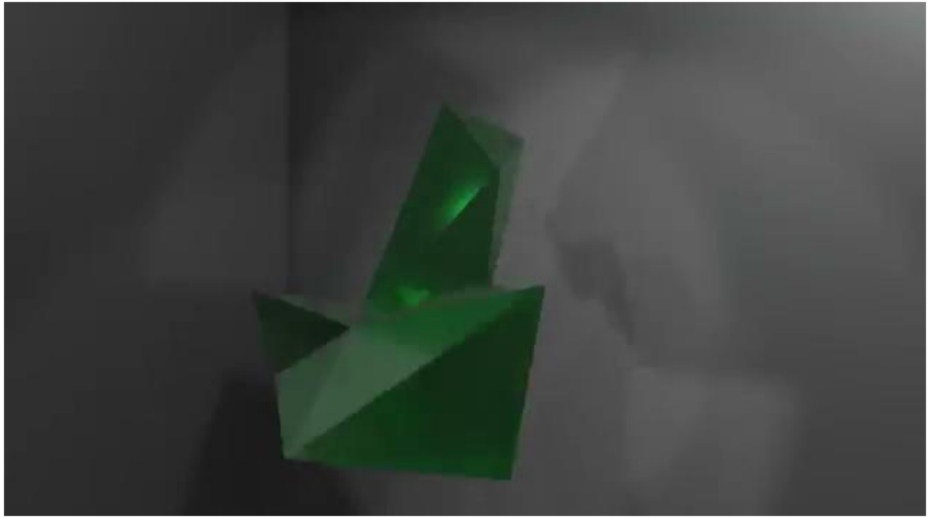
```
UserWarning: Taichi matrices/vectors with 122x1 > 32 entries are not suggested. Matrices/vectors will be automatically unrolled at compile-time for performance. So the compilation time could be extremely long if the matrix size is too big. You may use a field to store a large matrix like this, e.g.:

    x = ti.field(ti.f32, (122, 1)).
```

Solution:

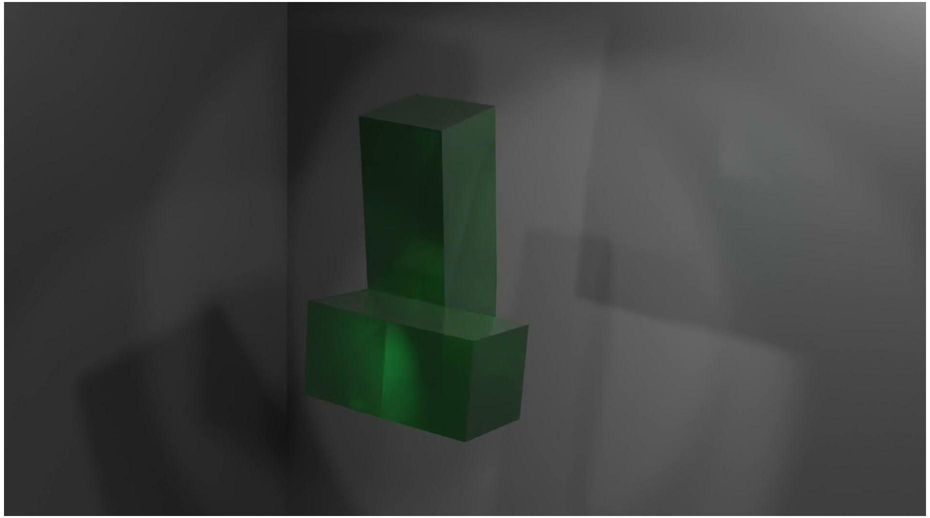
Sacrifice compile-time performance

Demo

Demo

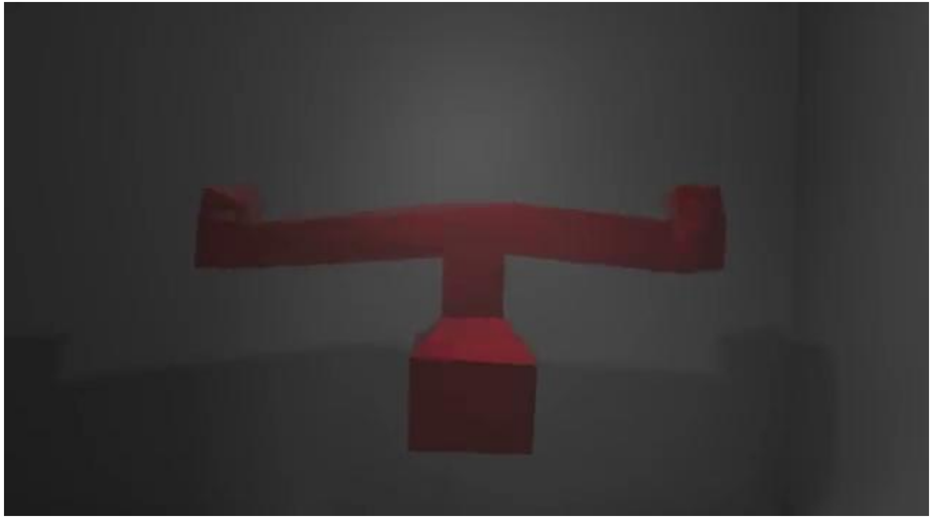


Euler, timestep 1/30s

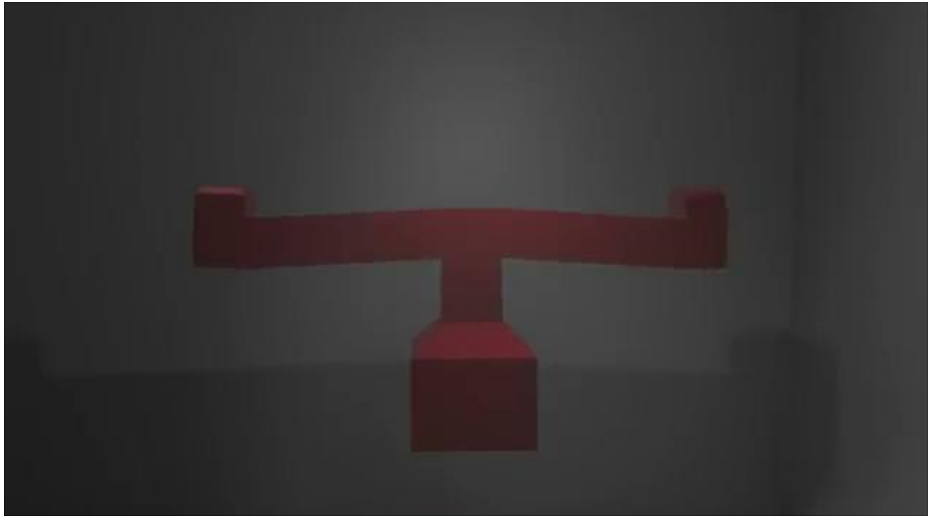


Euler + FEPR, timestep 1/30s

Demo



Euler, timestep 1/45s



Euler + FEPR, timestep 1/30s

Conclusion

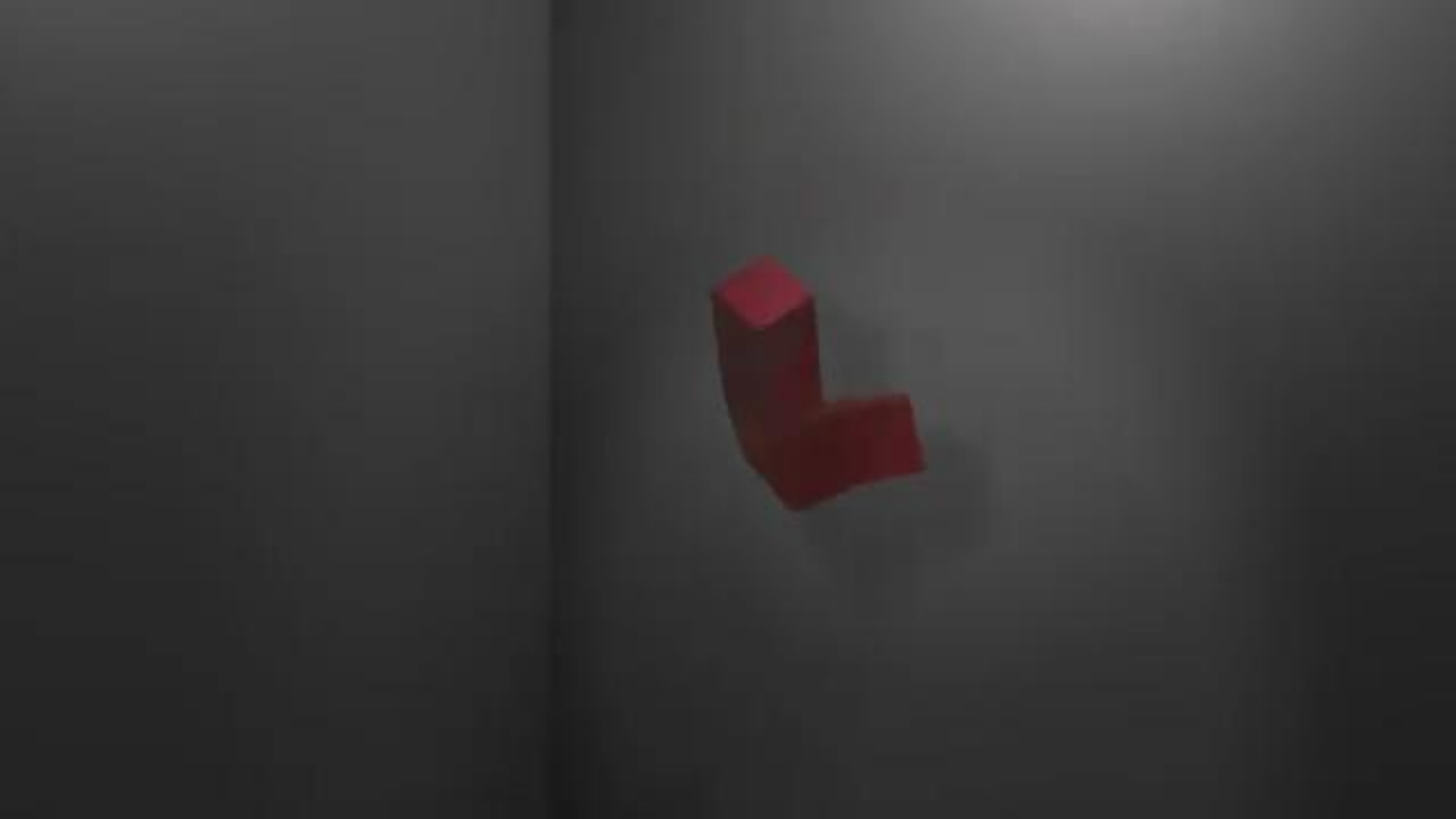
Future Work

- Better Integrators
- Add support for larger meshes (by using a different framework)
- Implement s and t constraints to support non-conserving motion

Main Takeaways

- Interesting to see how much FEPR improves the result

- Next time do not use Taichi
- Use pytorch or a different framework from the beginning
- By the time we figured out all the issues with taichi, it was too late to rewrite all the code.



References

[1] https://github.com/taichi-dev/taichi/blob/ec413c4ea1ed74c4d28fb9a9599f2d85cd164312/taichi/struct/struct_llvm.cpp#L307

[2] https://github.com/taichi-dev/taichi/blob/ec413c4ea1ed74c4d28fb9a9599f2d85cd164312/taichi/inc/constants.h#L12

[3] https://github.com/taichi-dev/taichi/issues/2696#issue-971309657

[4] https://github.com/taichi-dev/taichi/issues/385

[5] https://taichi.readthedocs.io/en/stable/differentiable_programming.html

[6] https://dongqing-wang.com/blog/games201l3/

[7] https://www.cs.utah.edu/~ladislav/dinev18FEPR/dinev18FEPR.pdf