

Blender - Python API

#8



Serdar ARITAN

Department of Computer Graphics
Hacettepe University, Ankara, Turkey



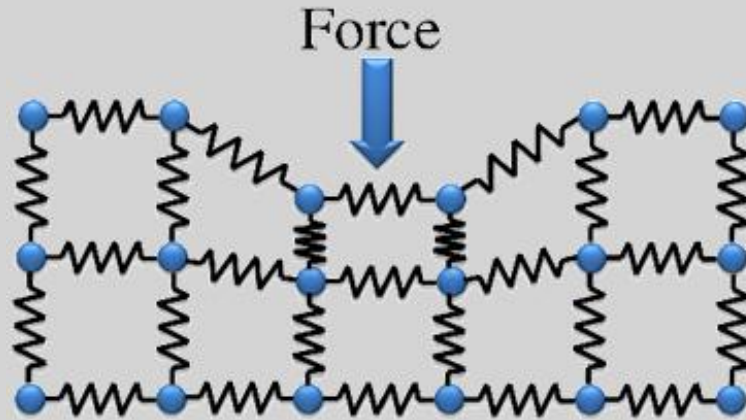
Blender/Python API

Soft-body dynamics

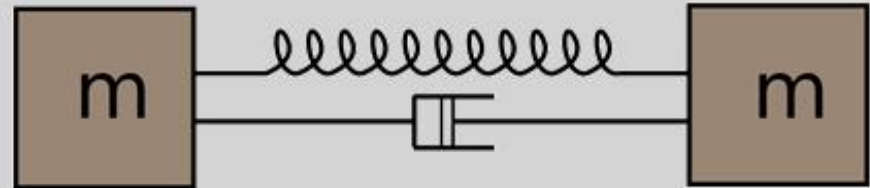
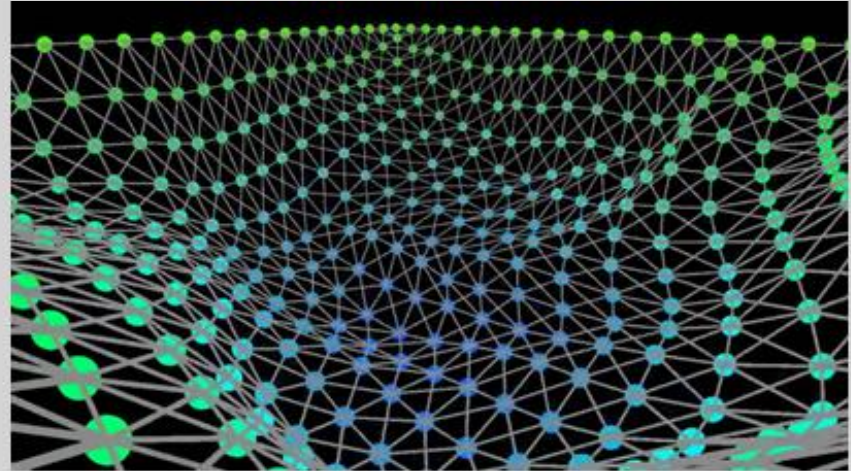
Soft-body dynamics is a field of computer graphics that focuses on visually realistic physical simulations of the motion and properties of deformable objects (or soft bodies). Unlike in simulation of rigid bodies, the shape of soft bodies can change, meaning that the relative distance of two points on the object is not fixed. While the relative distances of points are not fixed, the body is expected to retain its shape to some degree (unlike a fluid). The scope of soft body dynamics is quite broad, including simulation of soft organic materials such as muscle, fat, hair and vegetation, as well as other deformable materials such as clothing and fabric. **Generally, these methods only provide visually plausible emulations rather than accurate scientific/engineering simulations**, though there is some crossover with scientific methods, particularly in the case of **finite element simulations**

Blender/Python API

Soft-body dynamics



● Mass
~ Spring



Mass-Spring-Damper



Blender/Python API

Soft-body dynamics

Spring: an ideal elastic element

Will immediately change and return to its original shape upon loading and unloading.

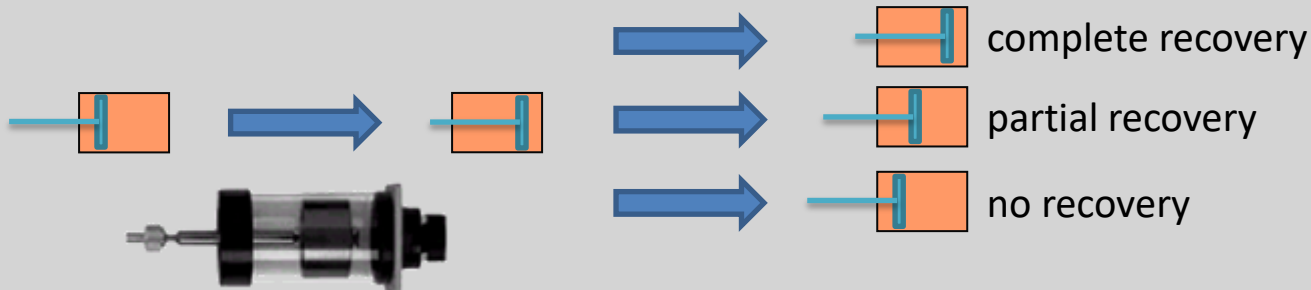


Damper: a viscous fluid

Will change its original shape upon loading, depending on time (and temperature).

May slowly return to or may not return to its original shape upon unloading.

partial or complete recovery

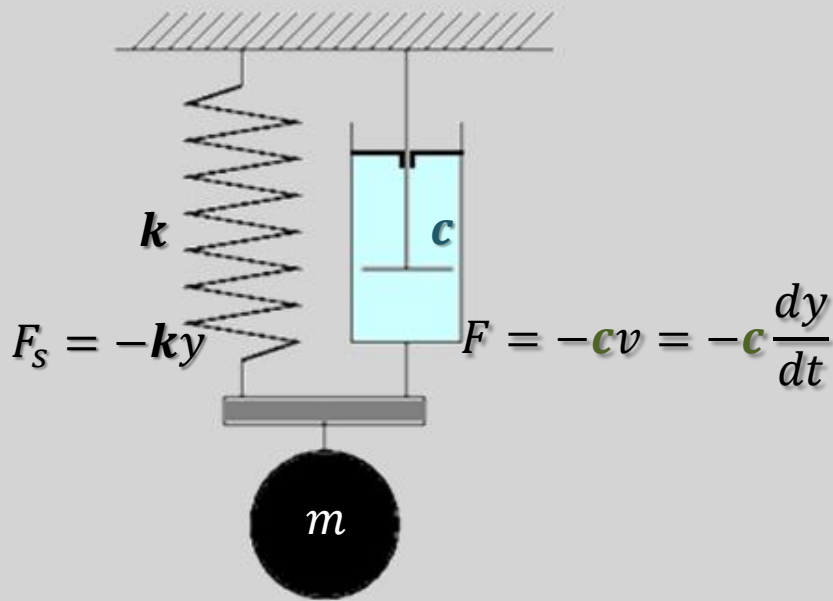




Blender/Python API

Soft-body dynamics

An ideal mass–spring–damper system with mass m , spring constant k , and viscous damper of damping coefficient c is subject to an oscillatory force.



$$F^{net} = -ky + \left(-c \frac{dy}{dt}\right)$$

$$ma = -ky - c \frac{dy}{dt}$$

$$a = -\frac{k}{m}y - \frac{c}{m} \frac{dy}{dt}$$

$$\frac{d^2y}{dt^2} + \frac{c}{m} \frac{dy}{dt} + \frac{k}{m}y = 0$$

2. Order Differential Equation



Blender/Python API

Blender's Physics

Blender's physics system allows you to simulate a number of different real world physical phenomena. You can use these systems to create a variety of static and dynamic effects such as: Soft Body, Cloth and Smoke

A Soft Body in general, is a simulation of a soft or rigid deformable object. It is useful for everything that tends to bend, deform, in reaction to forces like gravity or wind, or when colliding with other objects.

In Blender, this system is best for simple cloth objects and closed meshes e.g. for skin or rubber. There is **dedicated** Cloth Simulation physics that use a different solver, and is better for cloth.



Blender/Python API

Blender's Physics

Soft body simulation is done by applying forces to the vertices or control points of the object. There are exterior forces like gravity or force fields and interior forces that hold the vertices together. This way you can simulate the shapes that an object would take on in reality if it had volume, was filled with something, and was acted on by real forces.

Soft Bodies can interact with other objects through Collision. **They can interact with themselves through Self-Collision.**

The result of the Soft Body simulation can be converted to a static object. You can also bake edit the simulation, i.e. edit intermediate results and run the simulation from there.



Blender/Python API

Blender's Physics

Soft Bodies are well suited for:

- Elastic objects with or without collision.
- Flags, fabric reacting to forces.
- Certain modeling tasks, like a cushion or a table cloth over an object.
- Blender has another simulation system for clothing. But you can sometimes use Soft Bodies for certain parts of clothing, like wide sleeves.
- Hair (as long as you minimize collision).
- Animation of swinging ropes, chains and the like.

Blender/Python API

Blender's Physics

Soft Body simulation works for all objects that have vertices or control points:

- Meshes.

- Curves.

- Surfaces.

- Lattices.

To activate the Soft Body simulation for an object: In the Properties editor, go to the Physics tab (it is all the way on the right, and looks like a bouncing ball). Activate the Soft Body button. A lot of options appear.

Soft Body Object

Friction

The friction of the surrounding medium. Generally friction dampens a movement.

Mass

Mass value for vertices. Larger mass slows down acceleration, except for gravity where the motion is constant regardless of mass. Larger mass means larger inertia, so also braking a Soft Body is more difficult.

Soft Body Goal

Soft Body Goal acts like a pin on a chosen set of vertices; controlling how much of an effect soft body has on them. Enabling this tells Blender to use the position of a vertex in the simulation. Animating the vertices can be done in all the usual ways before the Soft Body simulation is applied. The goal is the desired end-position for vertices. How a soft body tries to achieve this goal can be defined using stiffness forces and damping. A Goal value of 1.0 means no Soft Body simulation, the vertex stays at its original (animated) position. When setting Goal to 0.0, the object is only influenced by physical laws.

Goal Settings

Stiffness

The spring stiffness for Goal. A low value creates very weak springs (more flexible “attachment” to the goal), a high value creates a strong spring (a stiffer “attachment” to the goal).

Damping

The friction for Goal. Higher values dampen the effect of the goal on the soft body.

Use Edges

The **edges** in a Mesh Object can act as **springs** as well.

Springs

Pull 

The spring stiffness for edges. A low value means very weak springs, a high value is a strong spring that resists being pulled apart. 0.5 is latex, 0.9 is like a sweater, 0.999 is a highly-starched napkin or leather.

Push 

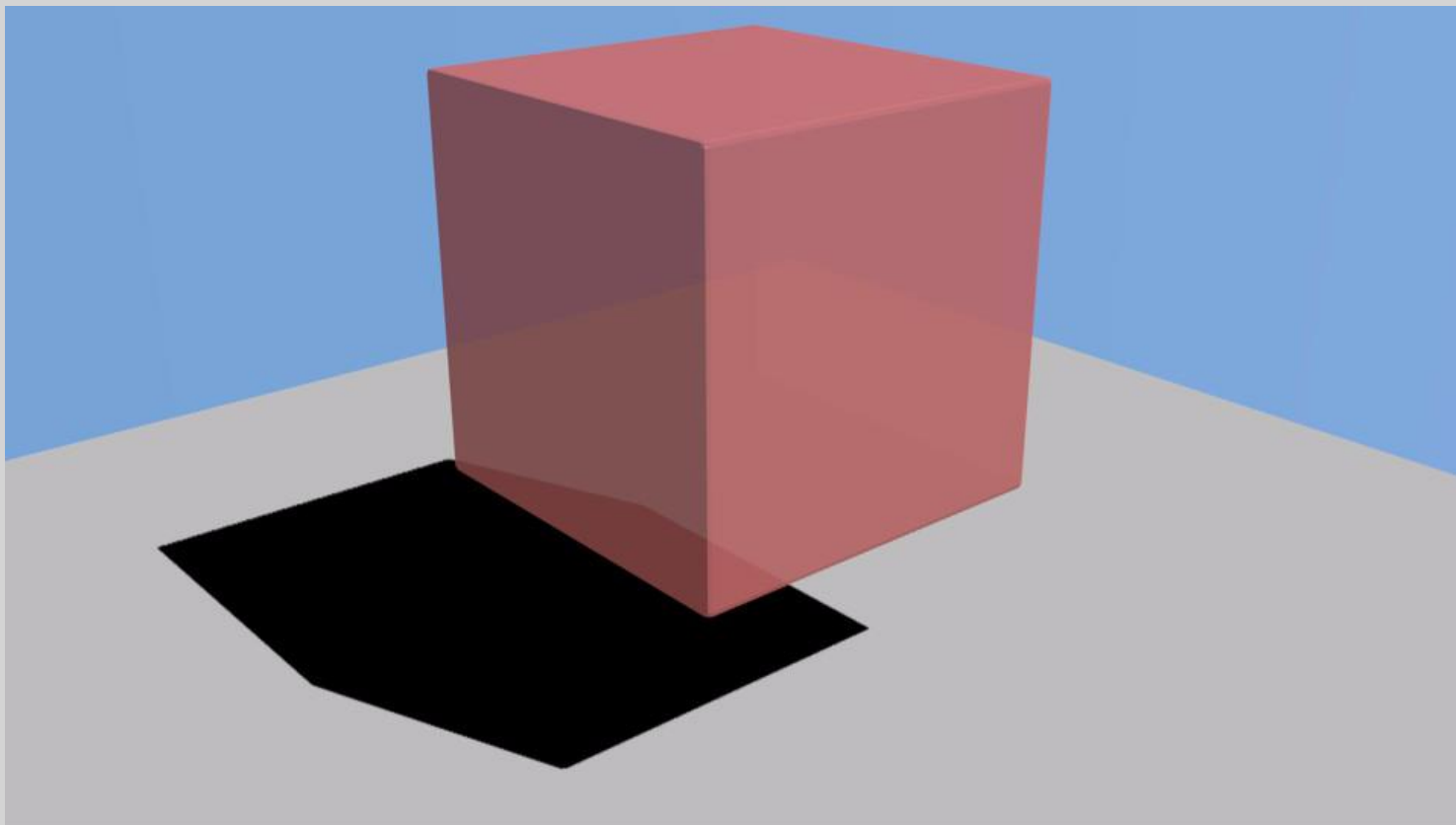
How much the soft body resist being scrunched together, like a compression spring. Low values for fabric, high values for inflated objects and stiff material.



Blender/Python API

Blender's Physics

Pull: 0.5
Push: 0.5
Damp: 0.5
Bending: 0.4
Shear: 0.4
Mass: 1

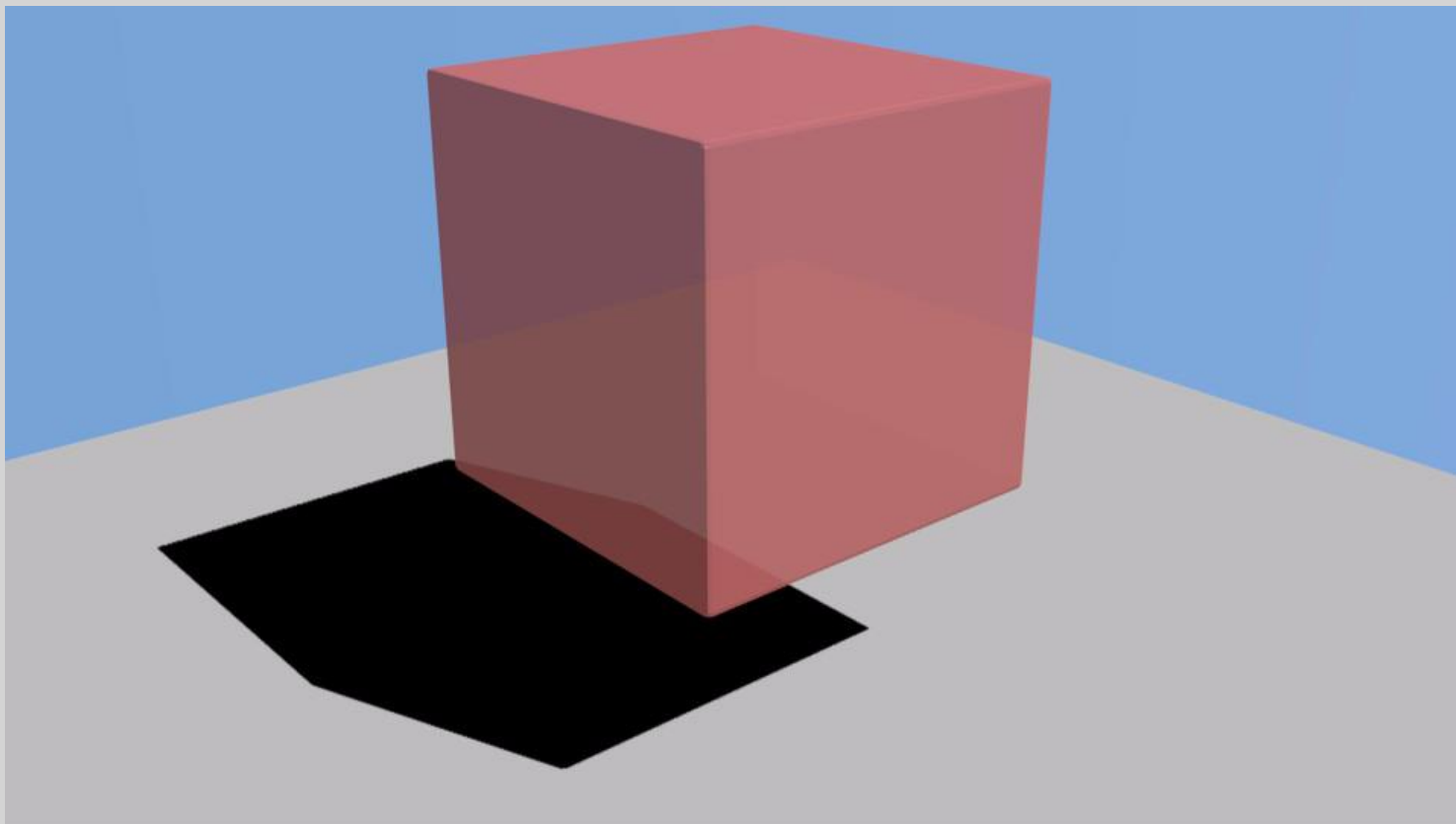




Blender/Python API

Blender's Physics

Pull: 0.3
Push: 0.3
Damp: 0.3
Bending: 0.2
Shear: 0.2
Mass: 1

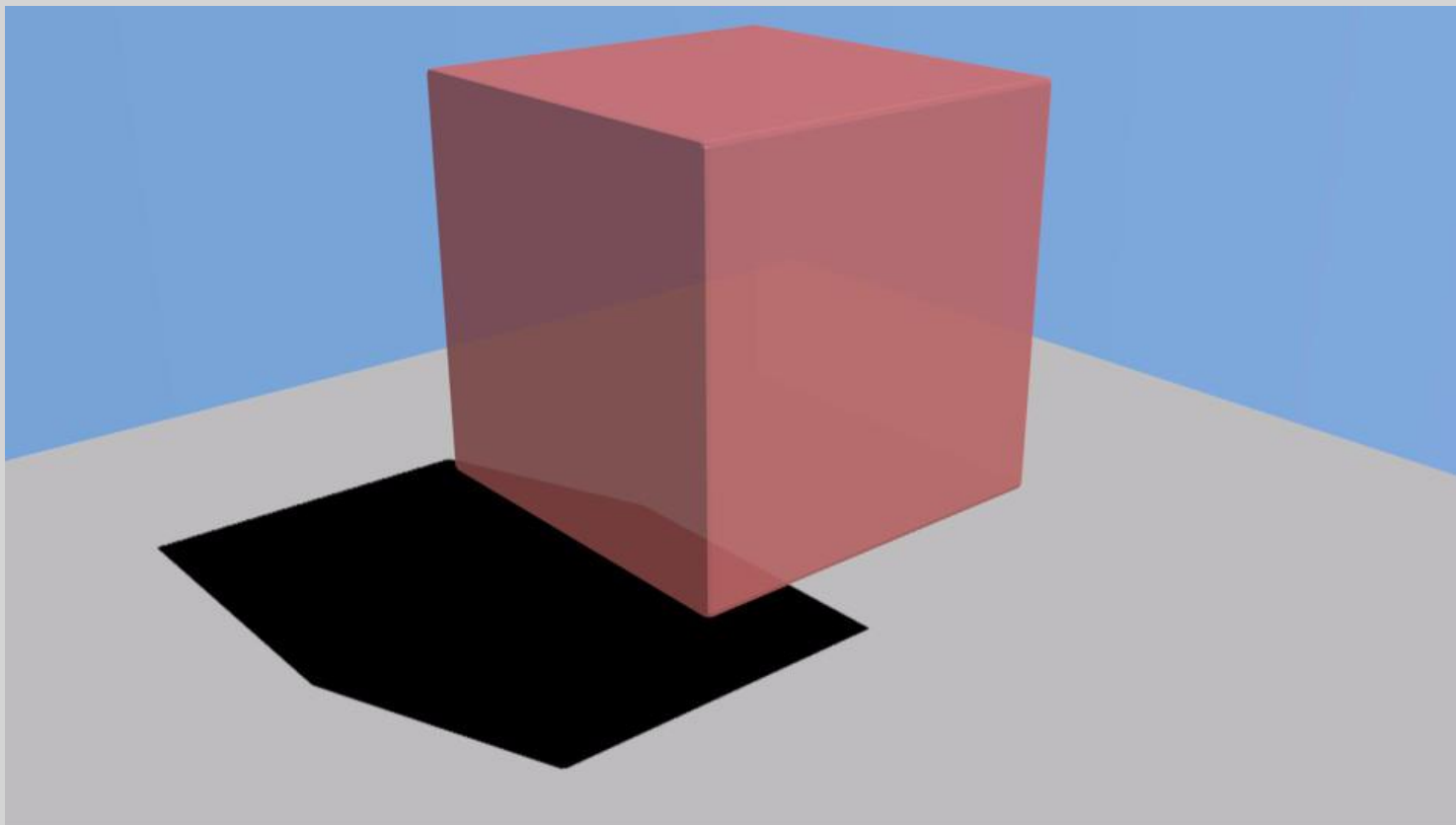




Blender/Python API

Blender's Physics

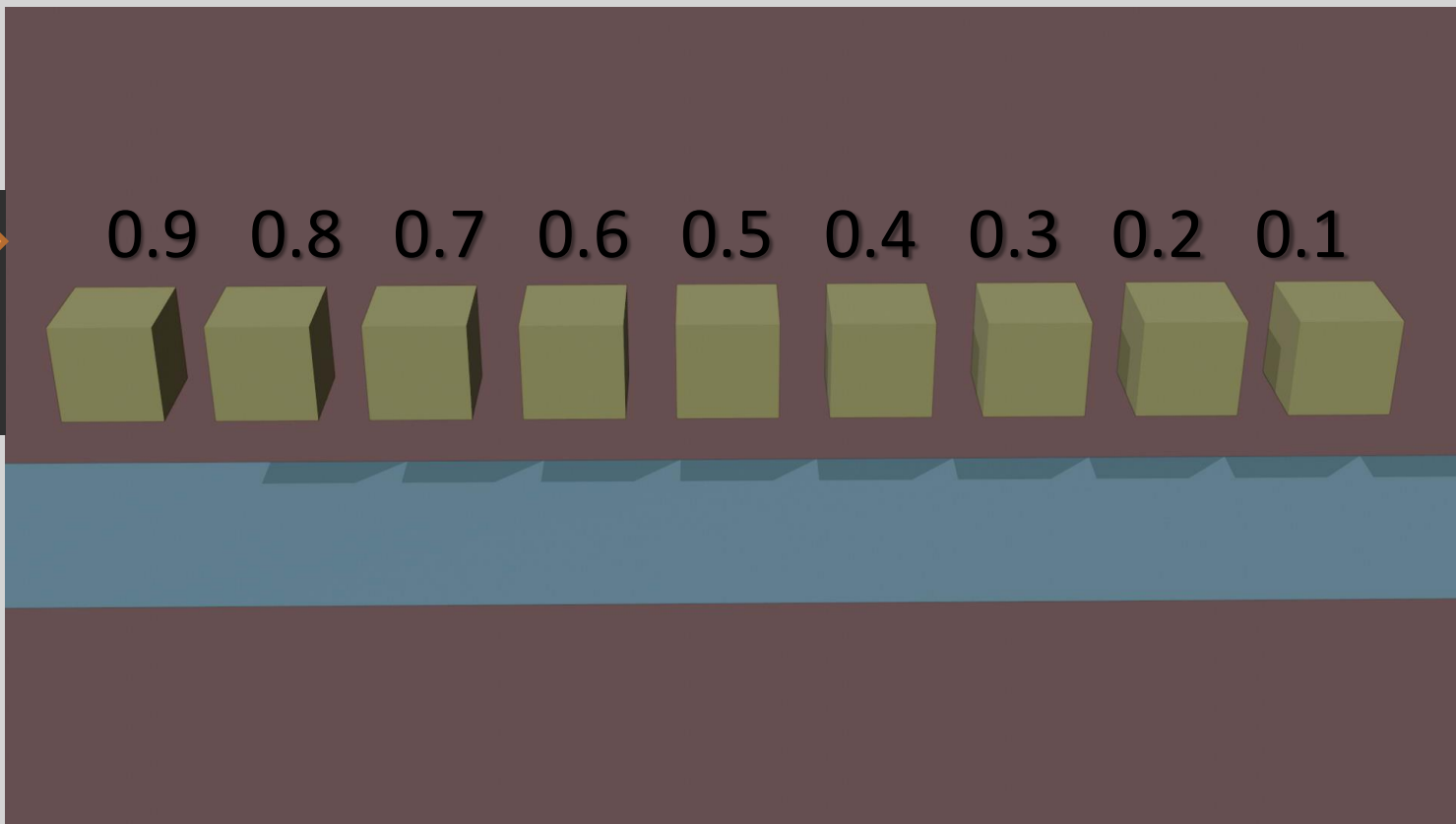
Pull: 0.1
Push: 0.1
Damp: 0.1
Bending: 0.1
Shear: 0.1
Mass: 1





Blender/Python API Blender's Physics

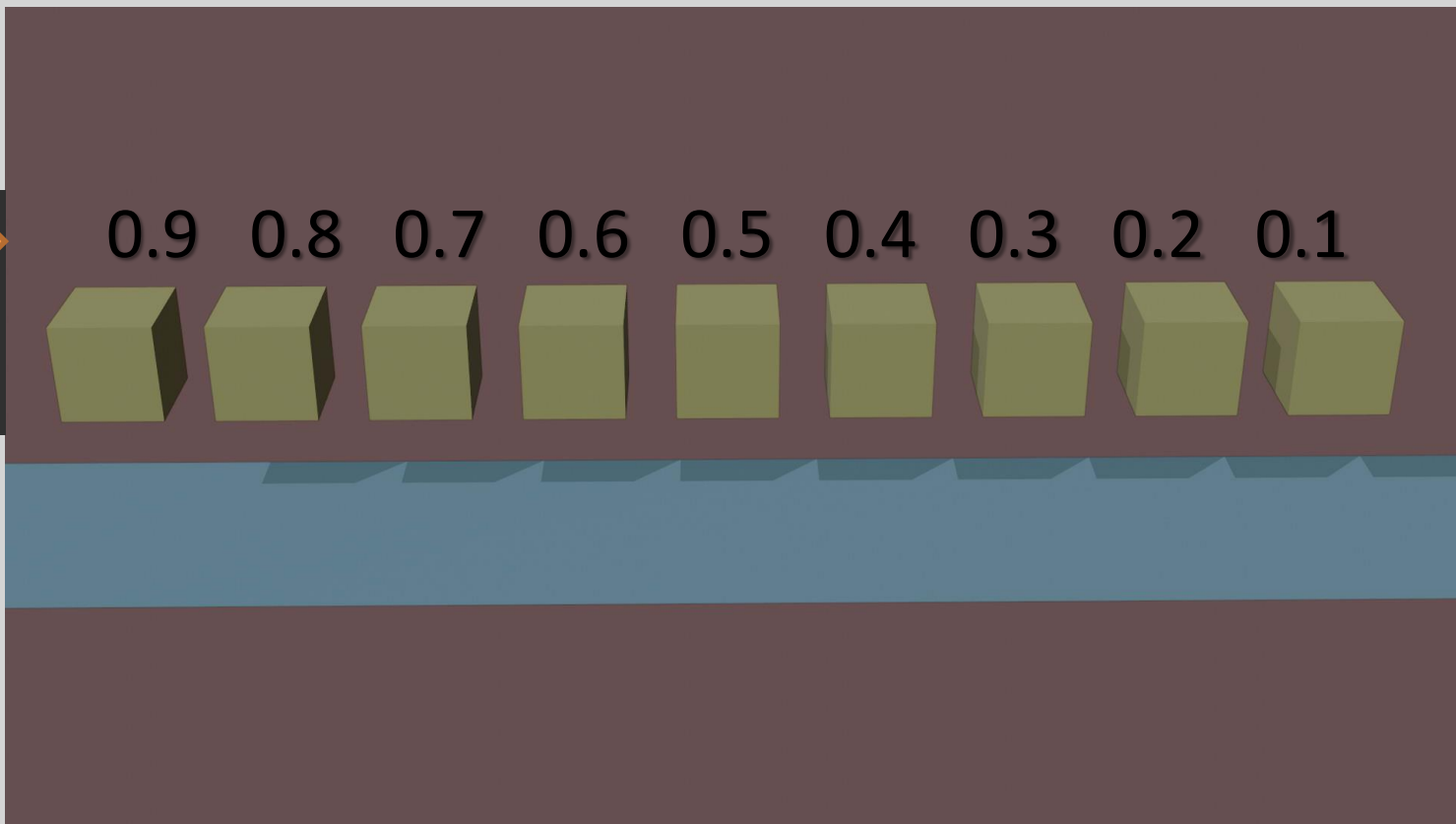
Pull is how much the edges are allowed to stretch





Blender/Python API Blender's Physics

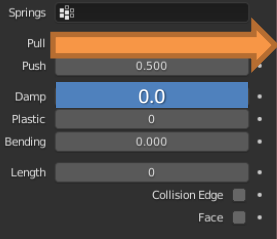
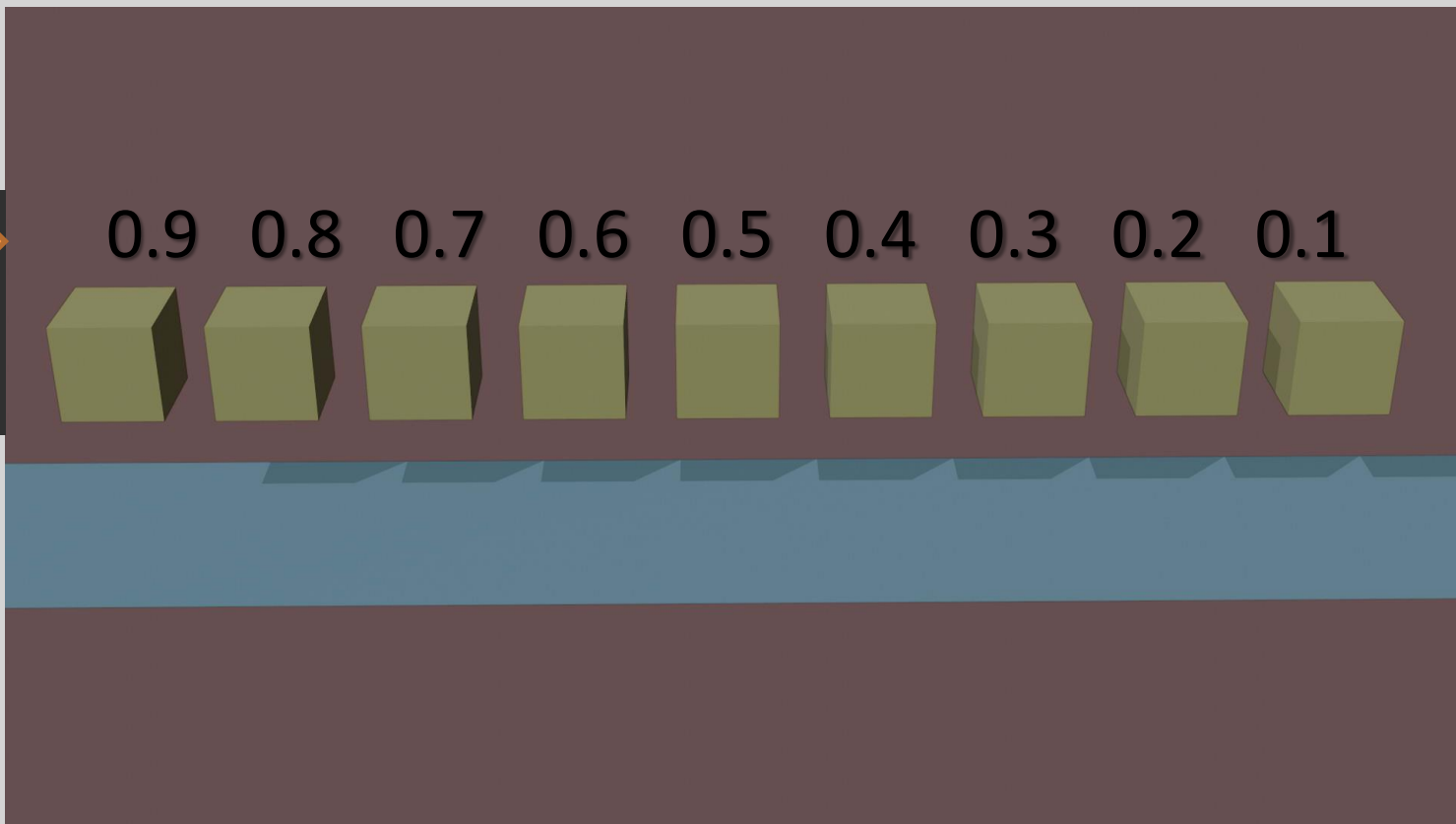
Pull is how much the edges are allowed to stretch





Blender/Python API Blender's Physics

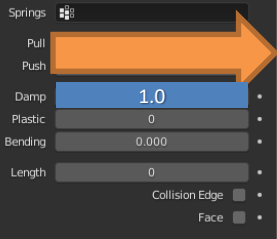
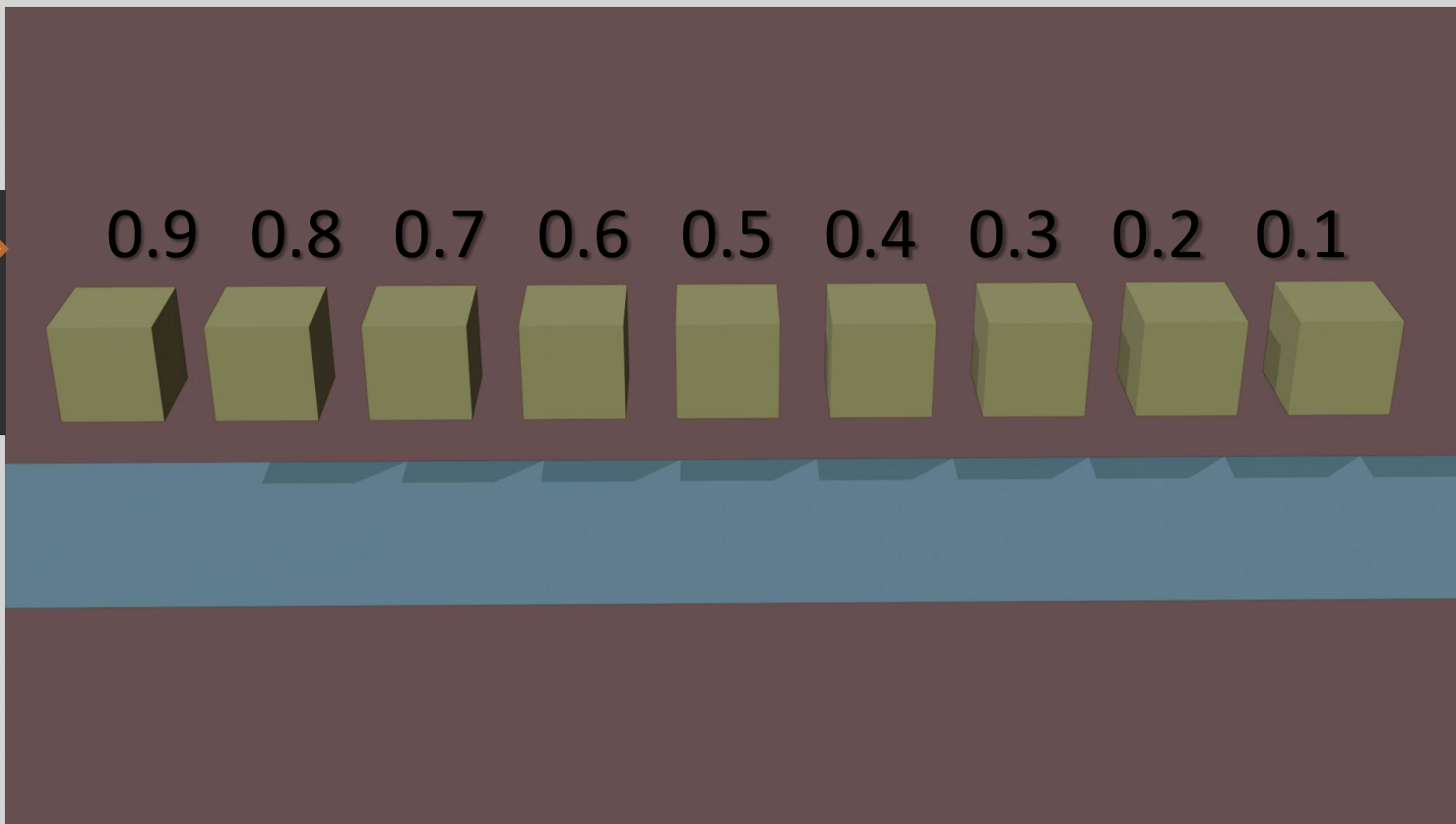
Pull is how much the edges are allowed to stretch





Blender/Python API Blender's Physics

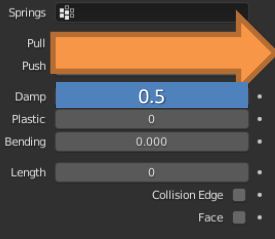
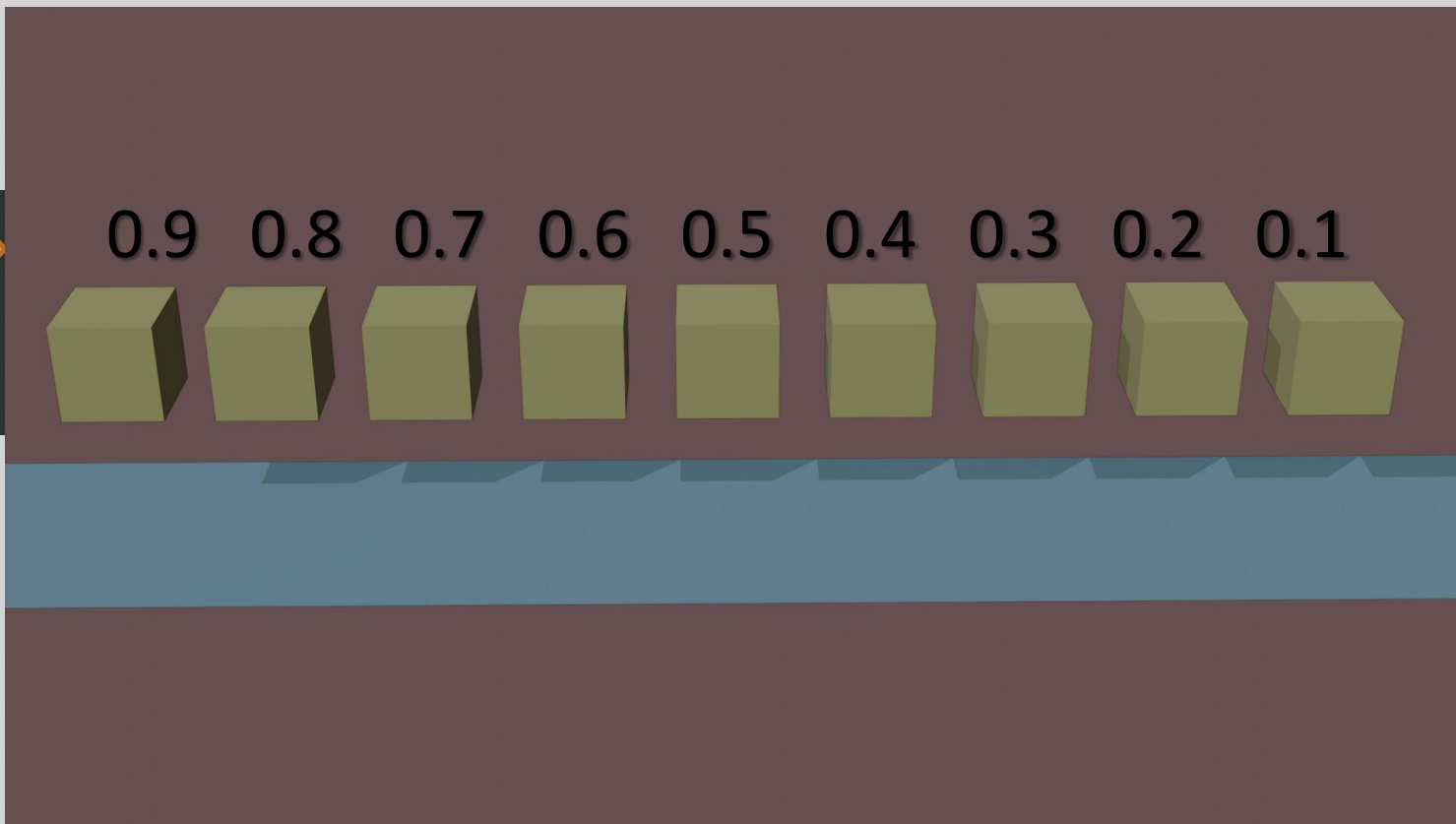
Push is how much the soft body resists being cramped together





Blender/Python API Blender's Physics

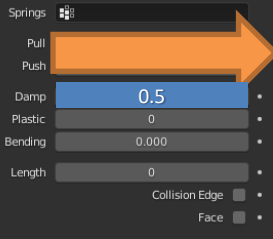
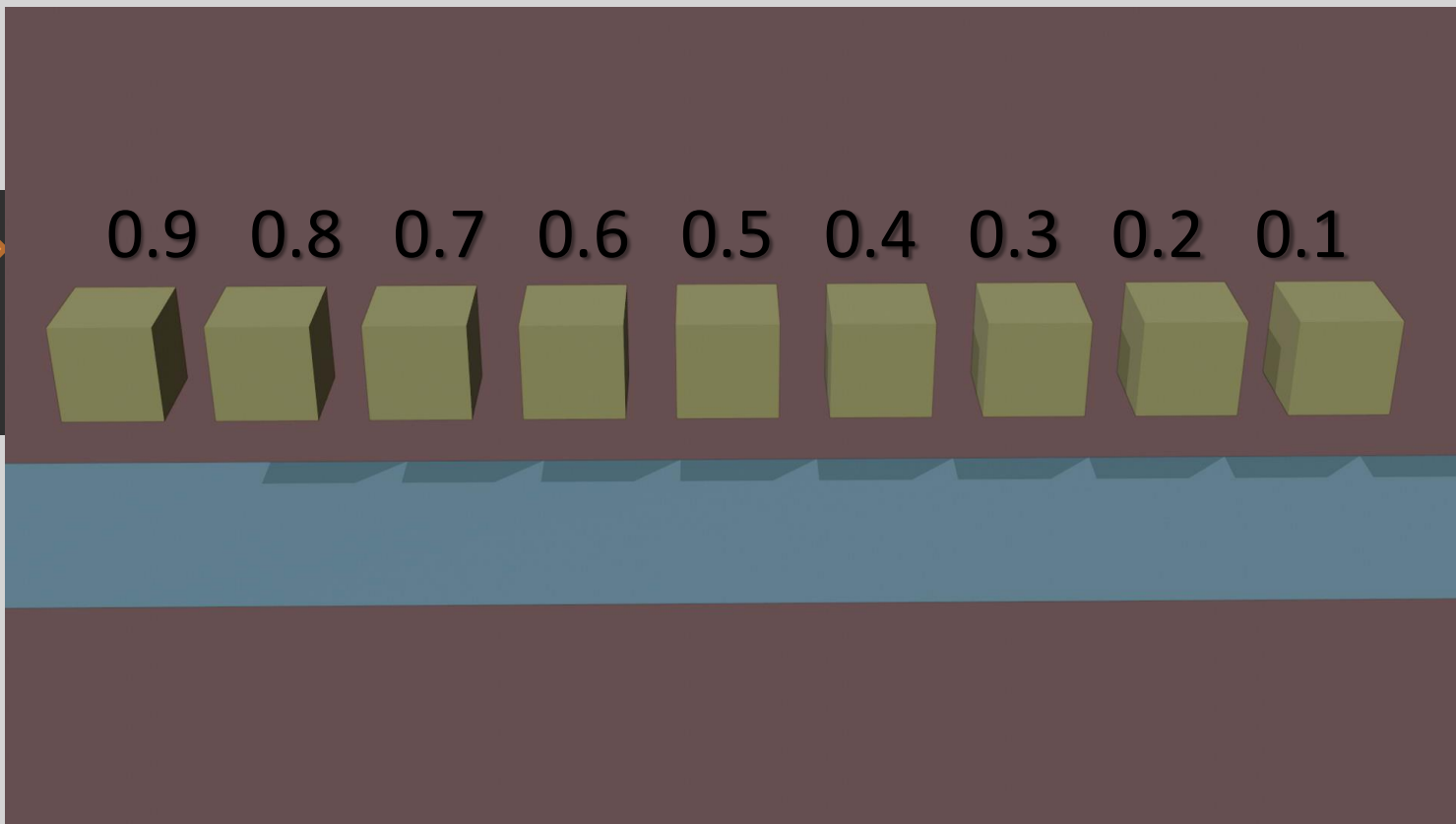
Push is how much the soft body resists being cramped together





Blender/Python API Blender's Physics

Push is how much the soft body resists being cramped together





Blender/Python API

Blender's Physics

Damp

The friction for edge springs. High values dampen the edge stiffness effect and calm down the body.

Plasticity

Permanent deformation of the object.

Bending

This option creates virtual connections between a vertex and the one after the next. This includes diagonal edges. Damping applies also to these connections.

Length

The edges can shrink or been blown up. This value is given in percent, 0 disables this function. 100% means no change, the body keeps 100% of its size.



Blender/Python API

Blender's Physics



Without Stiff Quads enabled.

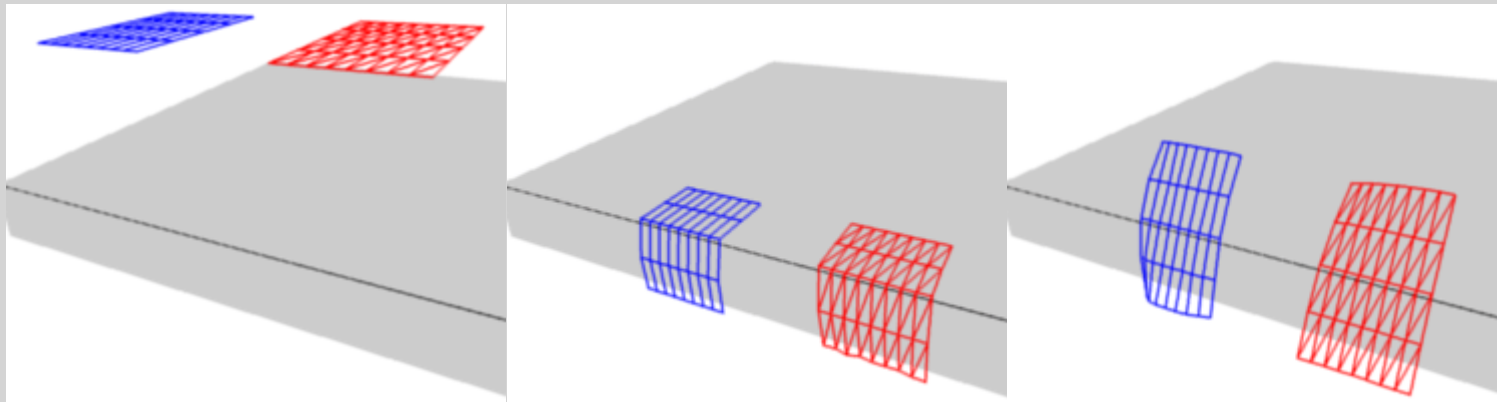


Stiff Quads is activated (for both cubes).





Blender/Python API Blender's Physics



No bending stiffness High bending stiffness (10)

Bending stiffness can also be used if you want to make a subdivided plane more plank like.



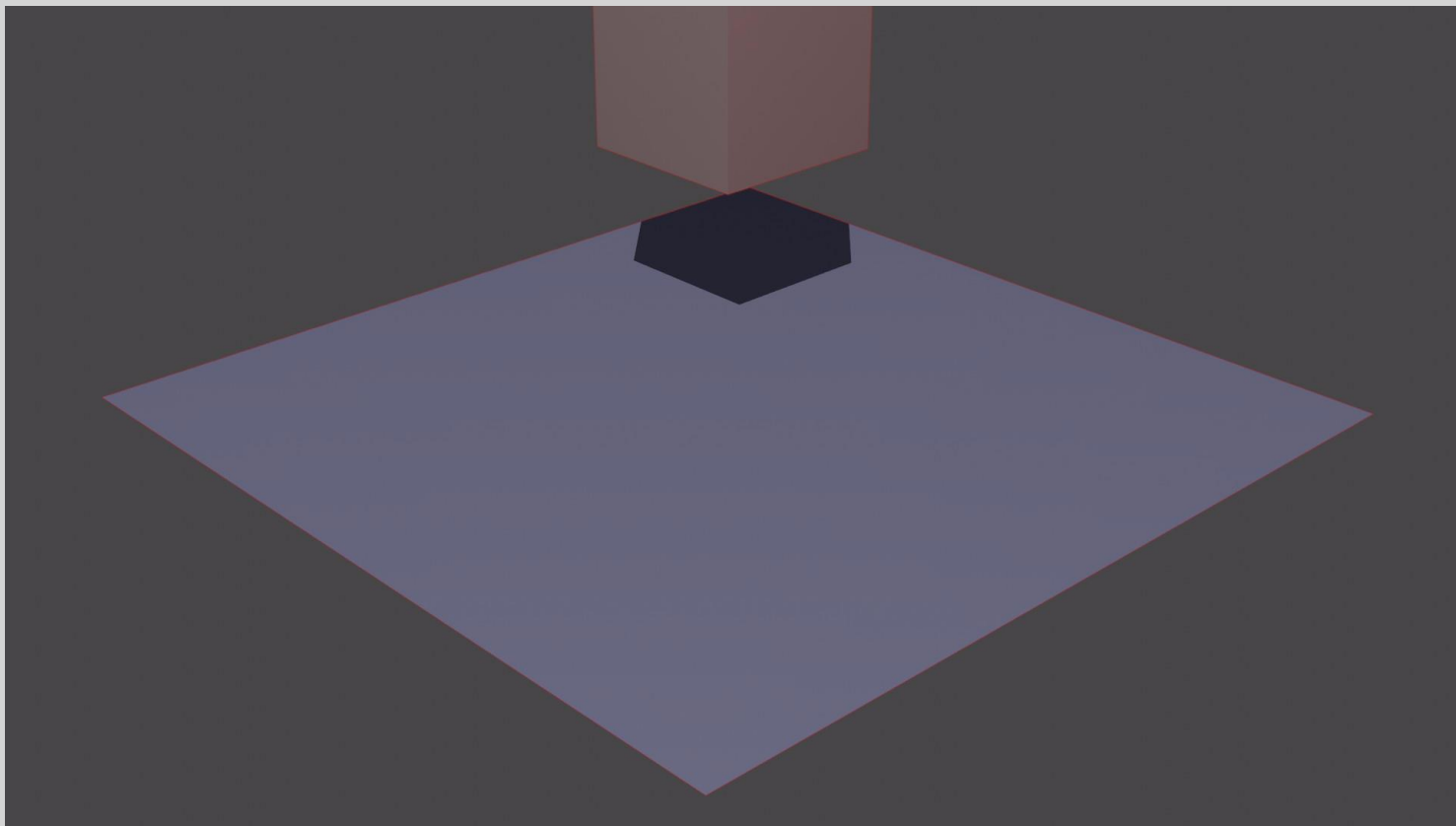
Blender/Python API Blender's Physics

```
import bpy
# Add a Cube
bpy.ops.mesh.primitive_cube_add(size=1, location=(0, 0, 2))
# Subdivide the Mesh for softbody calculations
bpy.ops.object.mode_set(mode = 'EDIT')
bpy.ops.mesh.subdivide(number_cuts = 4, smoothness = 0)
# Modify the cube as a SoftBody
bpy.ops.object.mode_set(mode='OBJECT')
bpy.ops.object.modifier_add(type='SOFT_BODY')
bpy.context.object.modifiers["Softbody"].settings.friction = 2
bpy.context.object.modifiers["Softbody"].settings.use_goal = False
bpy.context.object.modifiers["Softbody"].settings.use_self_collision = True
bpy.context.object.modifiers["Softbody"].settings.use_stiff_quads = False
bpy.context.object.modifiers["Softbody"].settings.pull = 0.5
bpy.context.object.modifiers["Softbody"].settings.push = 0.5
bpy.context.object.modifiers["Softbody"].settings.damping = 0.5
bpy.context.object.modifiers["Softbody"].settings.shear = 0.4
bpy.context.object.modifiers["Softbody"].settings.bend = 0.4
bpy.ops.object.modifier_add(type='COLLISION')
```



Blender/Python API Blender's Physics

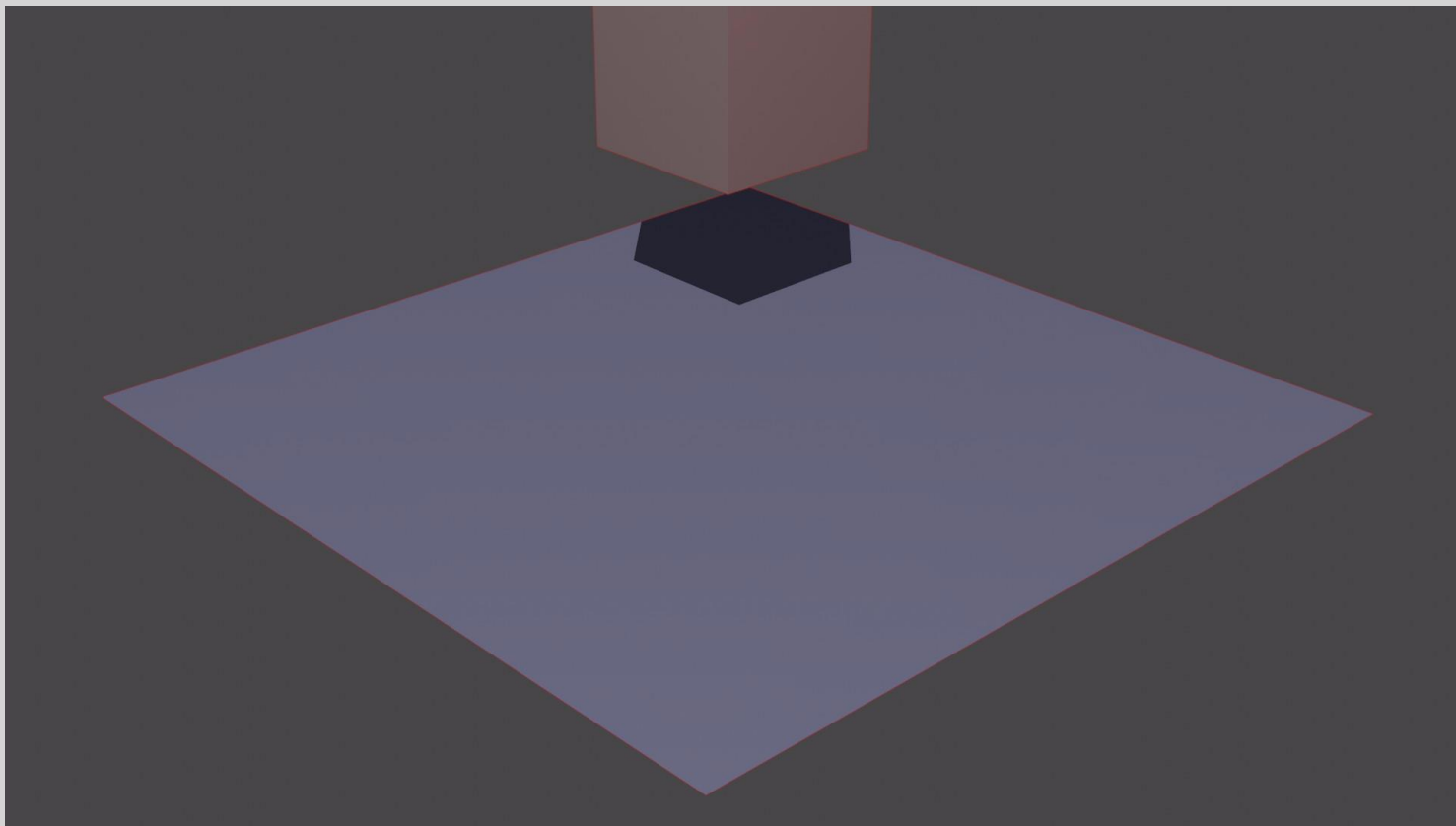
```
bpy.ops.mesh.subdivide(number_cuts = 4, smoothness = 0)
```





Blender/Python API Blender's Physics

```
bpy.ops.mesh.subdivide(number_cuts = 8, smoothness = 0)
```

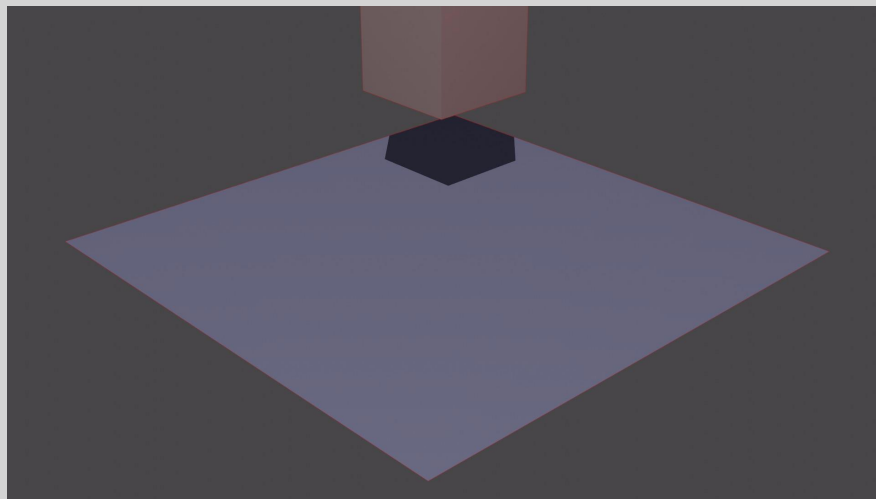




Blender/Python API Blender's Physics

```
bpy.ops.mesh.subdivide(number_cuts = 8, smoothness = 0)
```

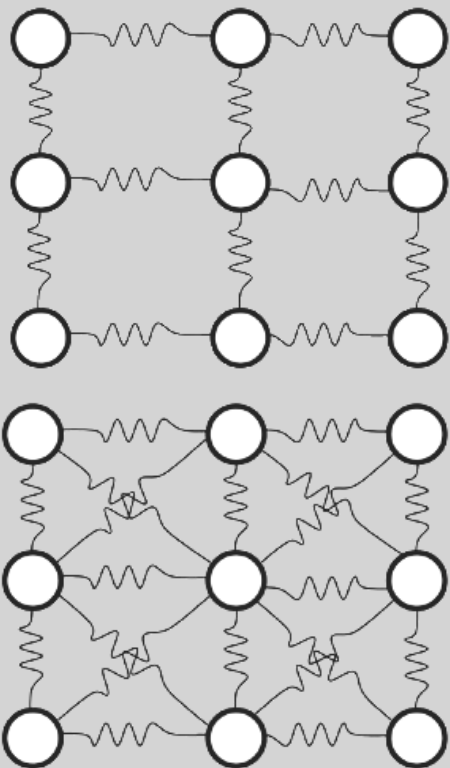
The **specific strength** is a material's (or muscle's) strength (force per unit area at failure) divided by its density. It is also known as the **strength-to-weight ratio** or **strength/weight ratio** or **strength-to-mass ratio**.



Another way to describe specific strength is **breaking length**, also known as **self support length**: the maximum length of a vertical column of the material (assuming a fixed cross-section) that could suspend its own weight when supported only at the top.

Blender/Python API

Blender's Physics



To create a connection between the vertices of a Soft Body object there have to be forces that hold the vertices together. These forces are effective along the edges in a mesh, the connections between the vertices. The forces act like a spring.

Additional forces with Stiff Quads enabled.

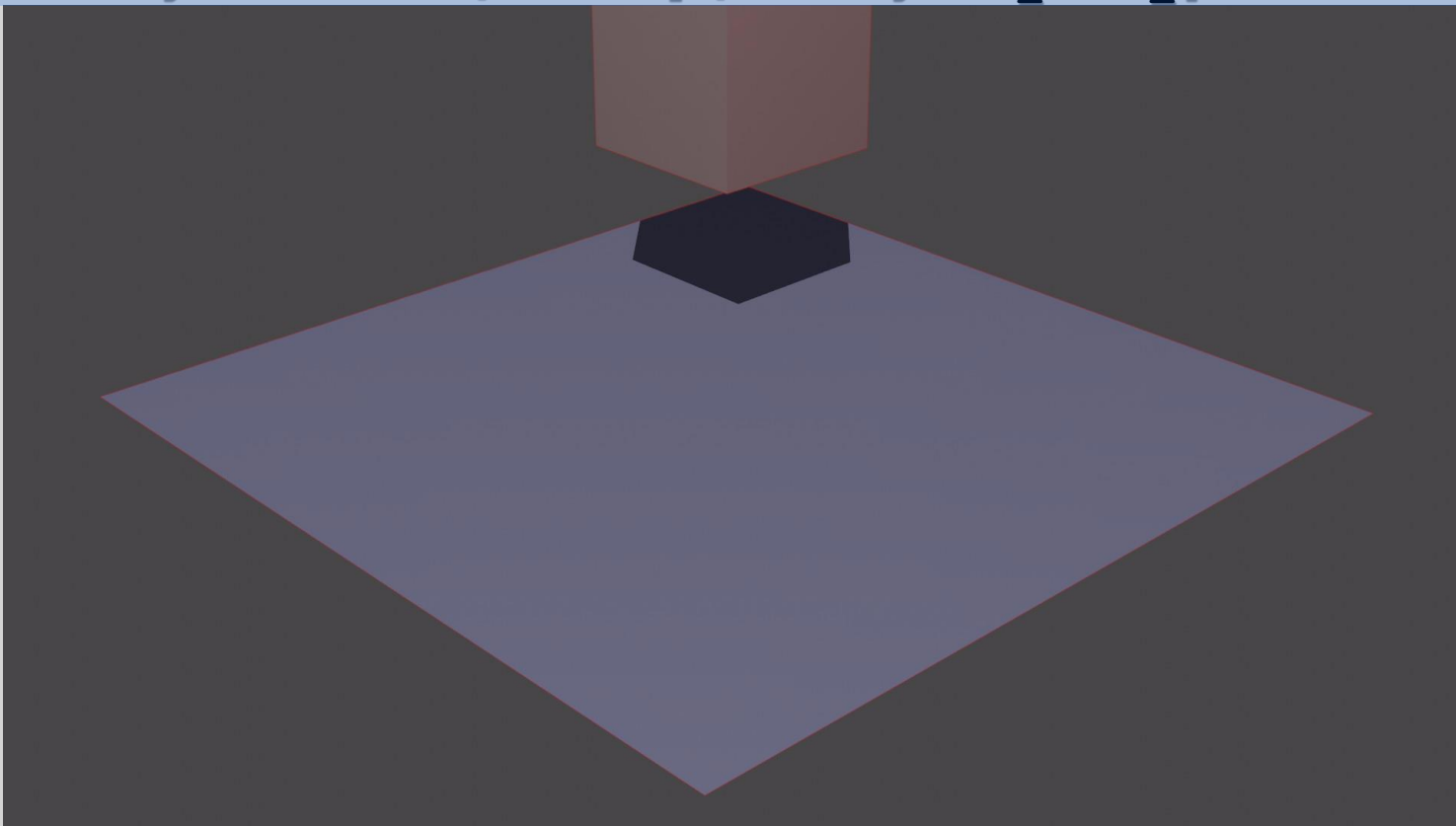


Blender/Python API

Blender's Physics

```
bpy.ops.mesh.subdivide(number_cuts = 8, smoothness = 0)
```

```
bpy.context.object.modifiers["Softbody"].settings.use_stiff_quads = True
```





Blender/Python API Soft Body Cube

```
import bpy  
import bmesh
```

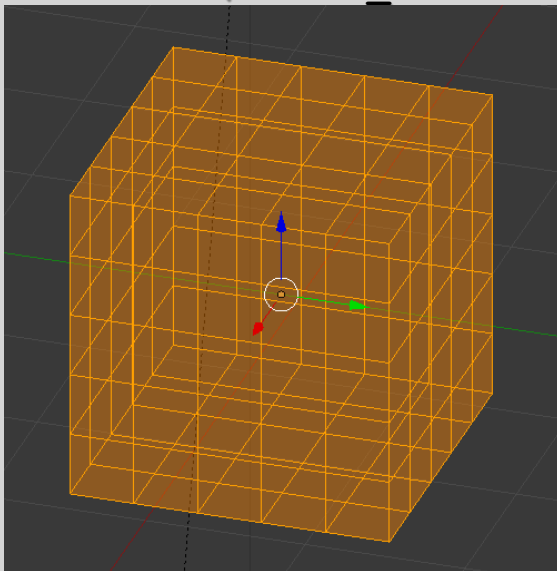
```
# Add a Cube
```

```
bpy.ops.mesh.primitive_cube_add(radius=1, location=(0, 0, 2))
```

```
# Subdivide the Mesh for softbody calculations
```

```
bpy.ops.object.mode_set(mode = 'EDIT')
```

```
bpy.ops.mesh.subdivide(number_cuts = 4, smoothness = 0)
```



This **subdivision** is for **physics calculations**; if you have a powerful computer you can increase the number of subdivision.
Do NOT smooth!



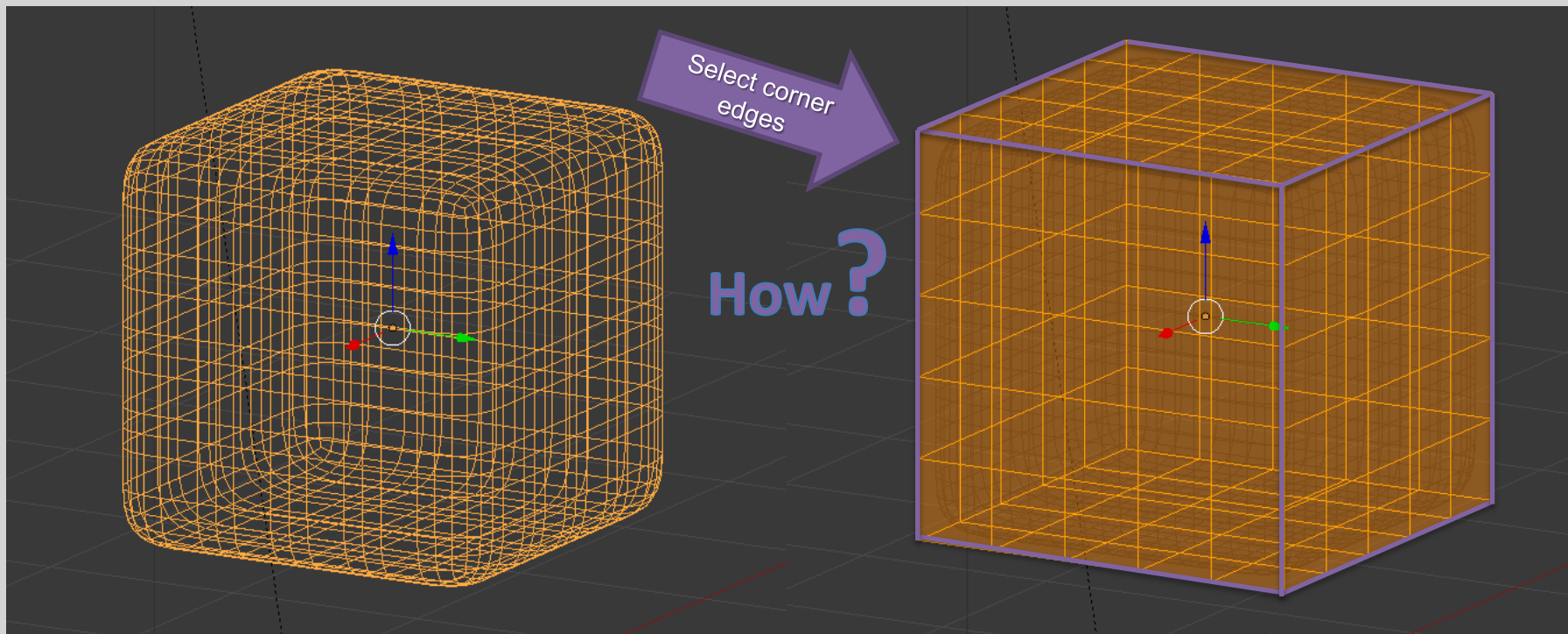
Blender/Python API Soft Body Cube

```
# Modify the cube as a SoftBody
bpy.ops.object.mode_set(mode='OBJECT')
bpy.ops.object.modifier_add(type='SOFT_BODY')
bpy.context.object.modifiers["Softbody"].settings.friction = 2
bpy.context.object.modifiers["Softbody"].settings.use_goal = False
bpy.context.object.modifiers["Softbody"].settings.use_self_collision = True
bpy.context.object.modifiers["Softbody"].settings.use_stiff_quads = True
bpy.context.object.modifiers["Softbody"].settings.pull = 0.5
bpy.context.object.modifiers["Softbody"].settings.push = 0.5
bpy.context.object.modifiers["Softbody"].settings.damping = 0.5
bpy.context.object.modifiers["Softbody"].settings.shear = 0.4
bpy.context.object.modifiers["Softbody"].settings.bend = 0.4
# Subsurface the cube for better render result
bpy.ops.object.modifier_add(type='SUBSURF')
bpy.context.object.modifiers["Subsurf"].levels = 2
bpy.context.object.modifiers["Subsurf"].render_levels = 4
```

Increase if you have a
powerful computer



Blender/Python API Soft Body Cube



Object Mode

Edit Mode



Blender/Python API Soft Body Cube

```
class bmesh.types.BMEdge
```

The BMesh edge connecting 2 verts

```
calc_face_angle(fallback=None)
```

Parameters: **fallback** (*any*) – return this when the edge doesn't have 2 faces (instead of raising a `ValueError`).

Returns: The angle between 2 connected faces in radians.

Return type: float

```
calc_face_angle_signed(fallback=None)
```

Parameters: **fallback** (*any*) – return this when the edge doesn't have 2 faces (instead of raising a `ValueError`).

Returns: The angle between 2 connected faces in radians (negative for concave join).

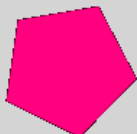
Return type: float



Convex



Concave



Convex

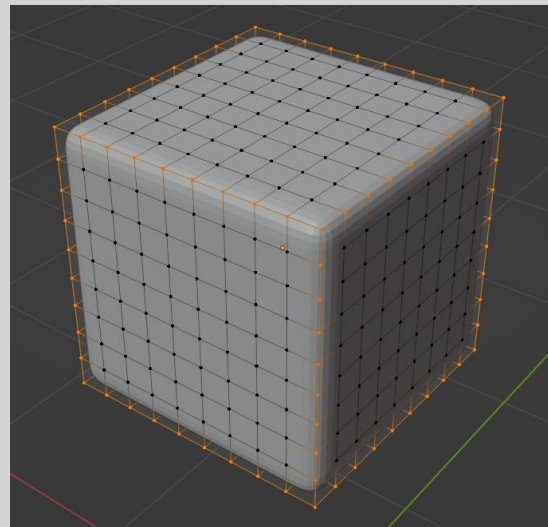


Concave



Blender/Python API Soft Body Cube

```
# Switching to EDIT mode to read mesh data
bpy.ops.object.mode_set(mode = 'EDIT')
# Deselect all verts, edges, faces
bpy.ops.mesh.select_all(action="DESELECT")
# Register bmesh object and select various parts
me = bpy.context.object.data
bm = bmesh.from_edit_mesh(me)
bm.edges.ensure_lookup_table()
bm.select_mode = {'EDGE'}
# Select the corner edges
for e in bm.edges:
    if e.calc_face_angle_signed() >= 1.57: # in radians 90 degree 1.5708 rad
        e.select = True
bm.select_flush_mode()
```





Blender/Python API Soft Body Cube

Each edge in a Blender model has a **crease value** associated with it, which is used to tell the Subsurf modifier how sharp we want that edge to be.

By default, all **edges** have a crease of **0**, which is why our cube has lost all its sharp edges.

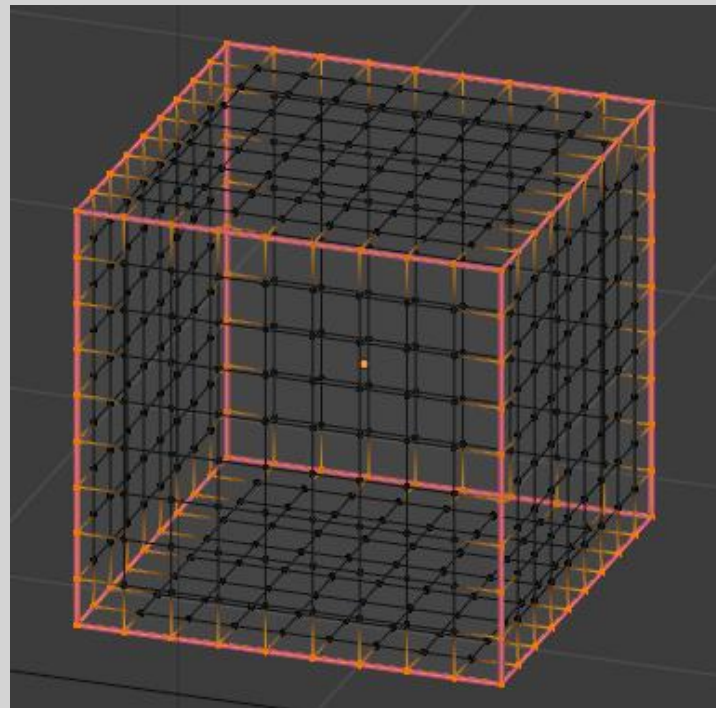
To Crease or Not to Crease?



*Yes to creases on
dress pants*

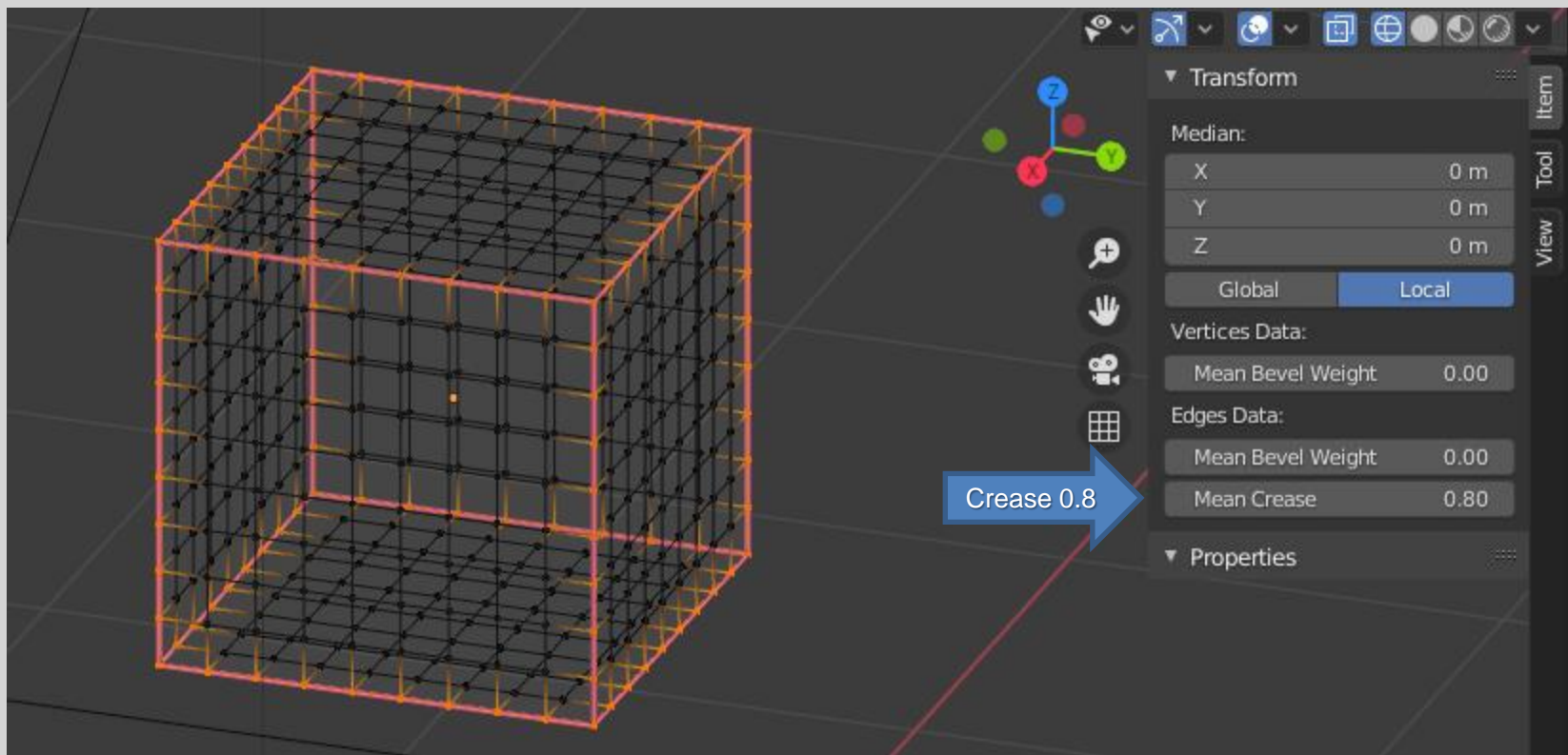


*No to creases
on jeans*



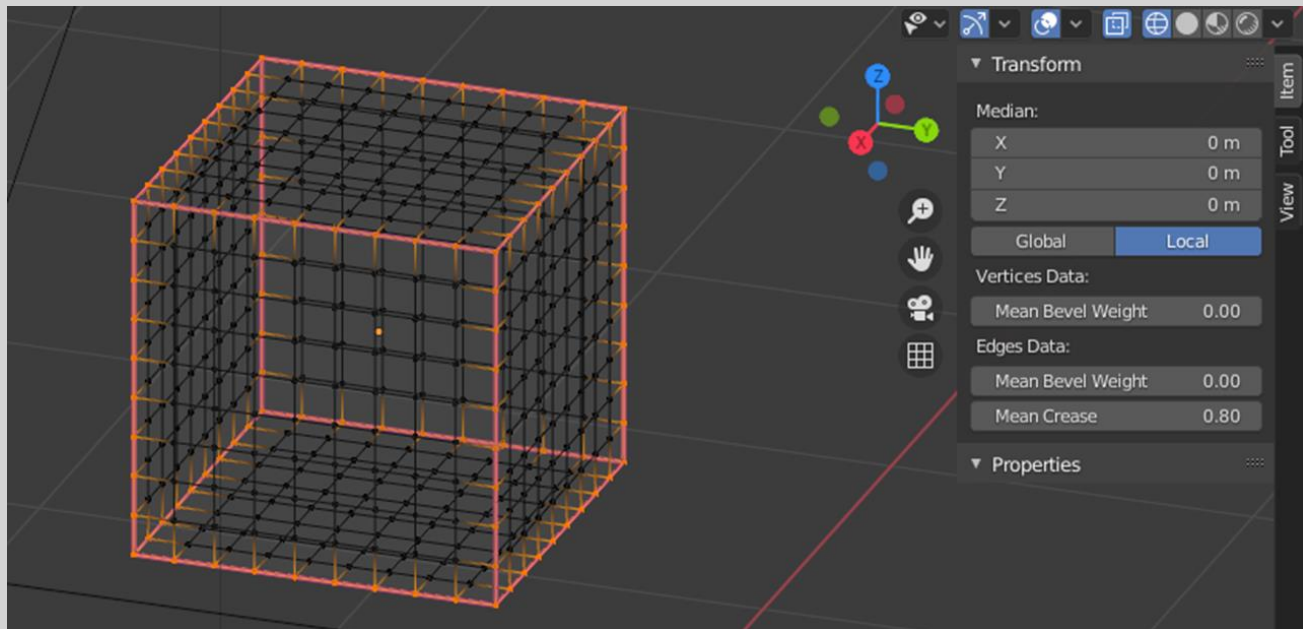


Blender/Python API Soft Body Cube





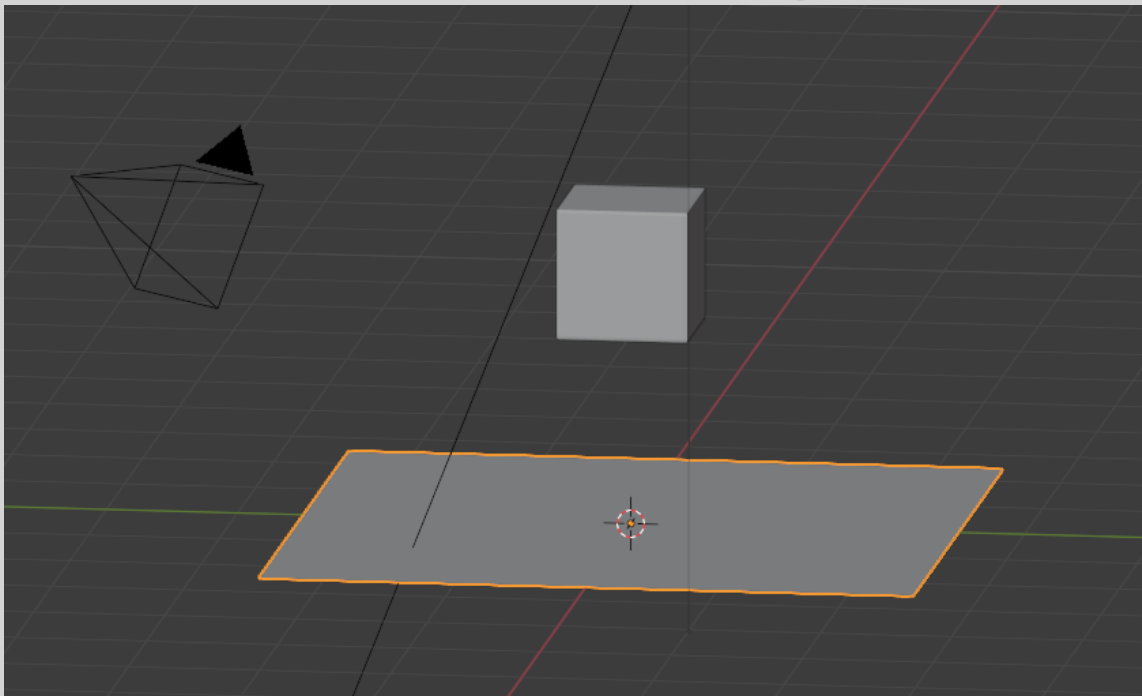
Blender/Python API Soft Body Cube



```
cr = bm.edges.layers.crease.verify()  
selectedEdges = [e for e in bm.edges if e.select]  
for e in selectedEdges: e[cr] = 0.8  
bmesh.update_edit_mesh(me)
```




Blender/Python API Soft Body Cube



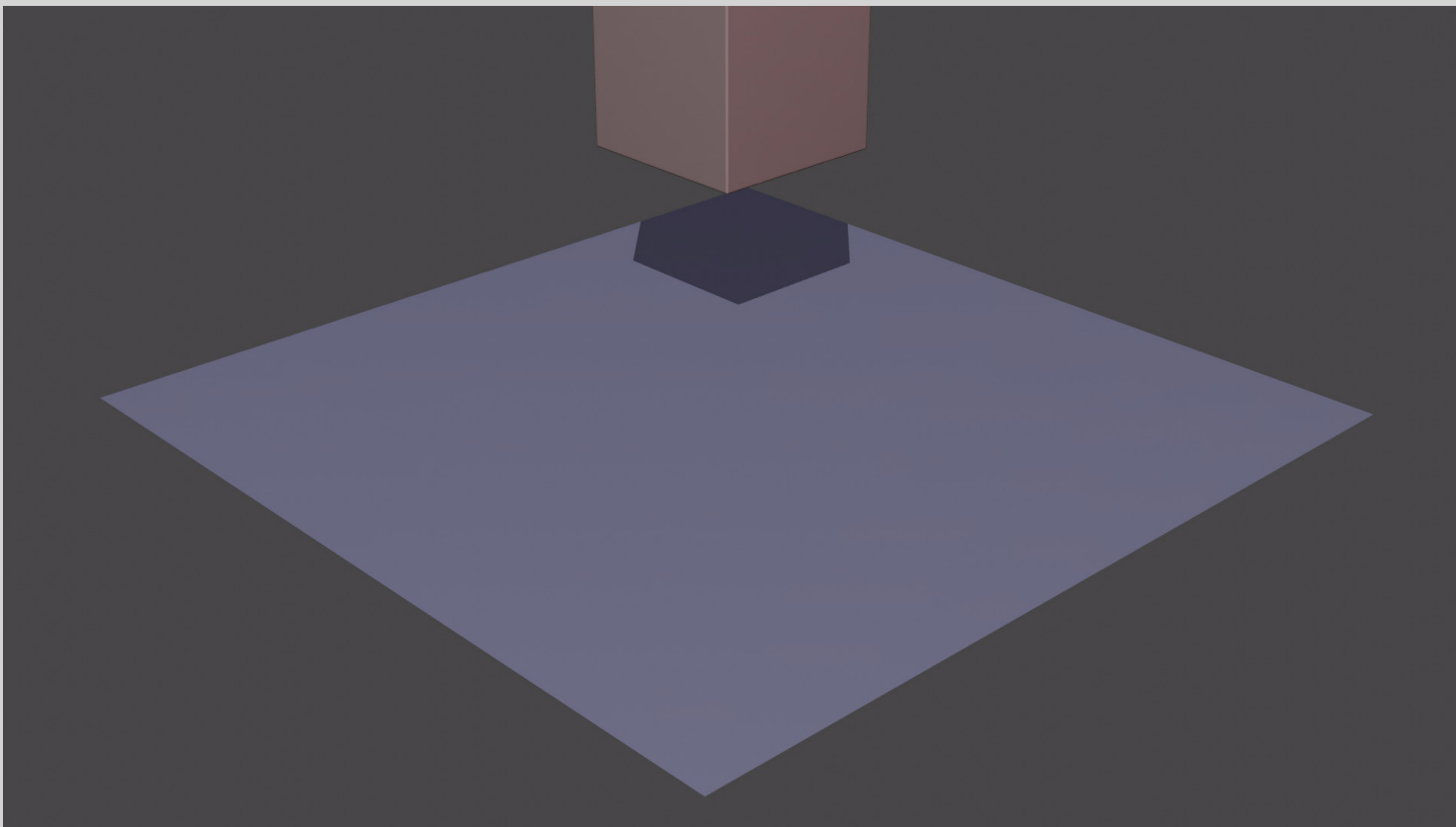
Add a Plane

```
bpy.ops.object.mode_set(mode='OBJECT')  
bpy.ops.mesh.primitive_plane_add(size=5, location=(0, 0, 0))  
bpy.ops.object.modifier_add(type='COLLISION')
```



Blender/Python API

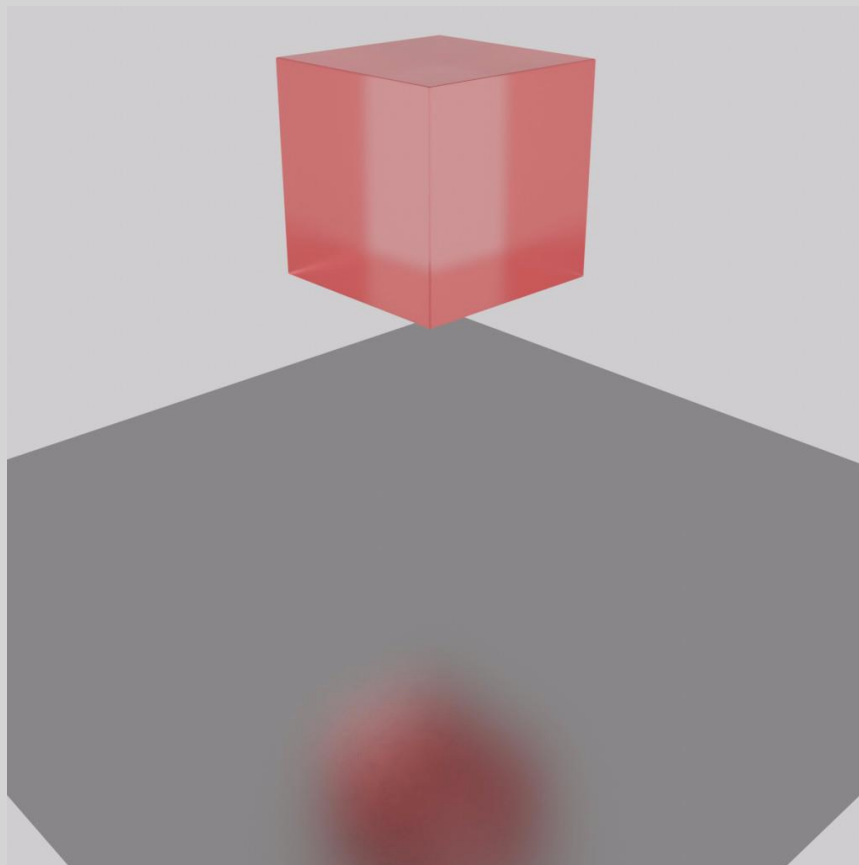
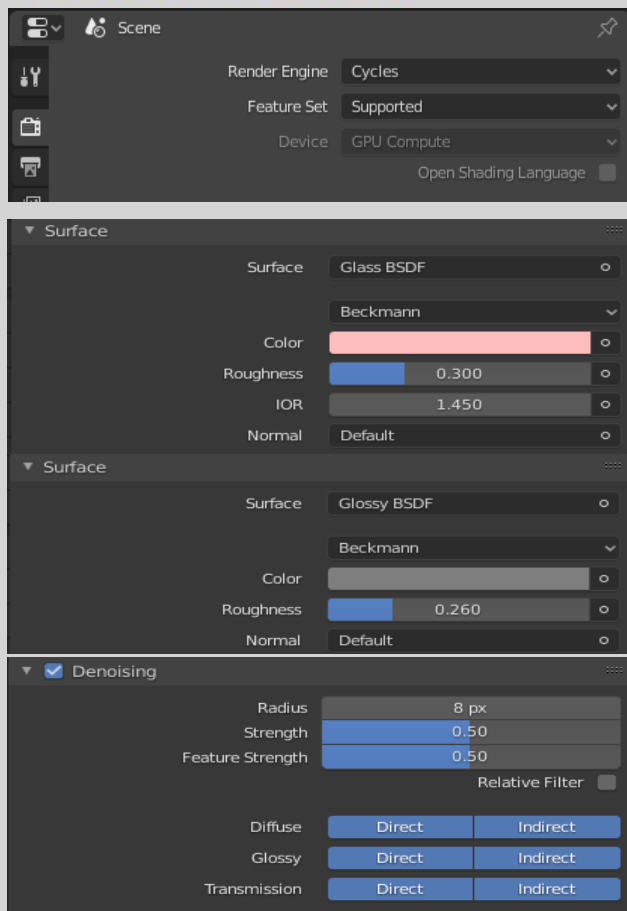
Soft Body Cube





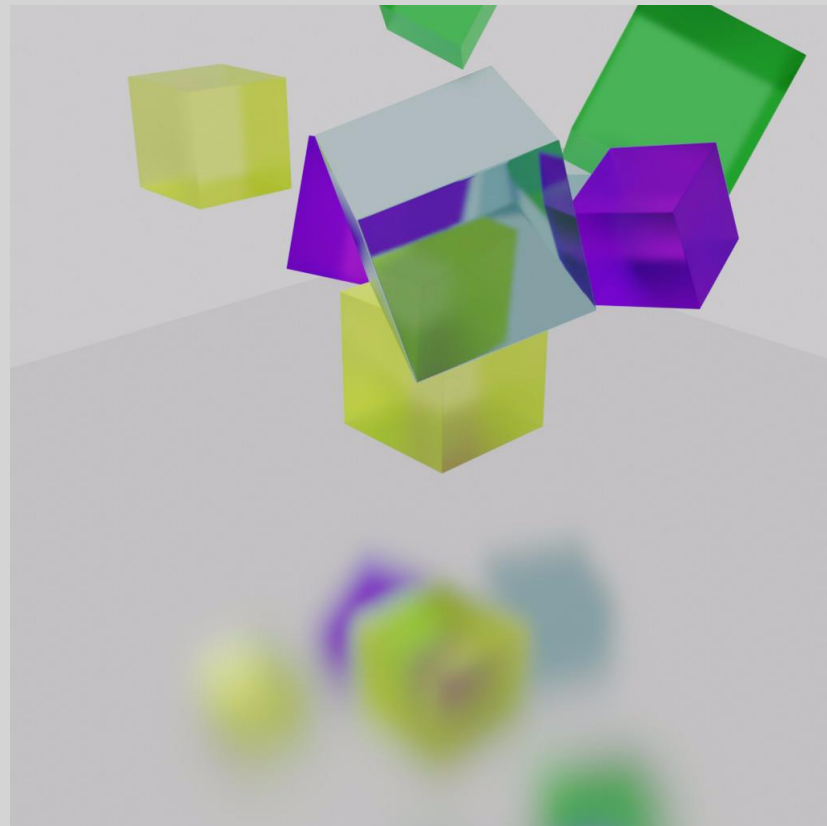
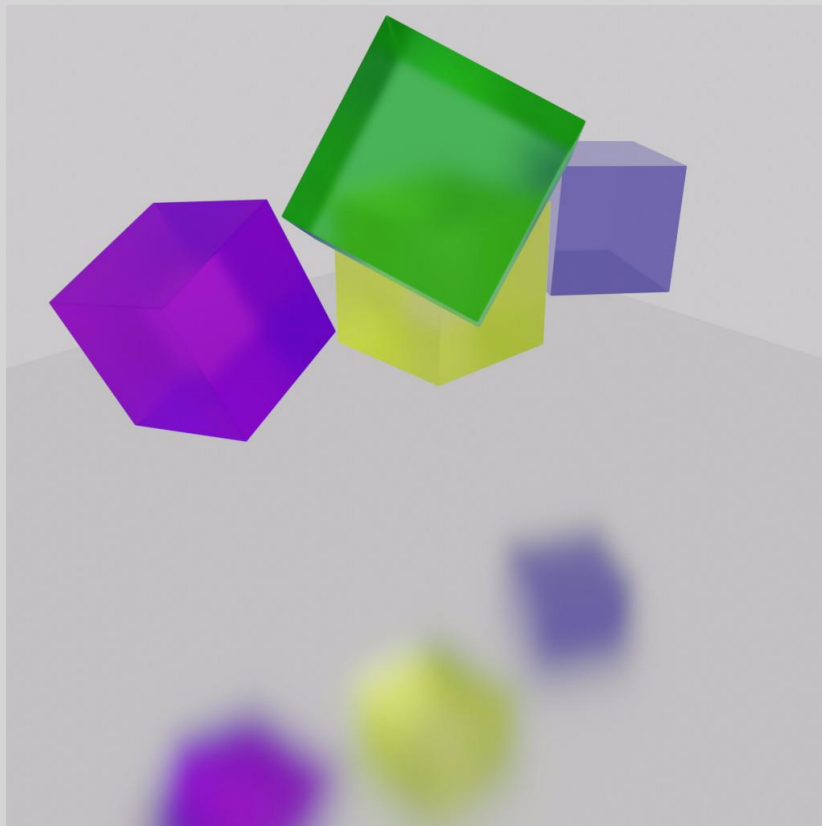
SCRIPT LANGUAGES FOR ANIMATION

Blender/Python API Soft Body Cube





Blender/Python API Soft Body Cube

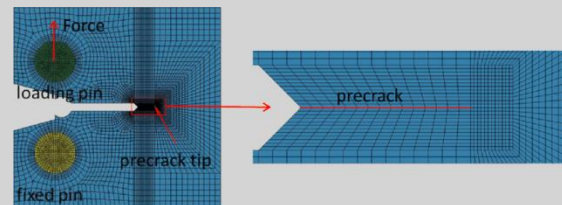




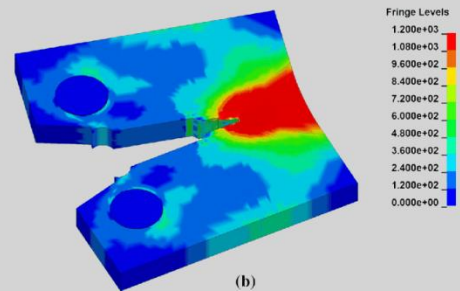
Blender/Python API Finite Element Method

Normally 3D models consist of just the outer shell of an object, and techniques like voronoi fracturing "**fake**" internal matter by creating extra surfaces between pieces when needed.

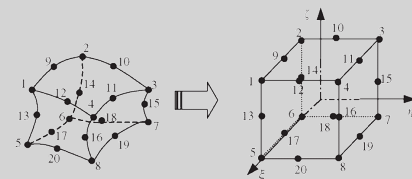
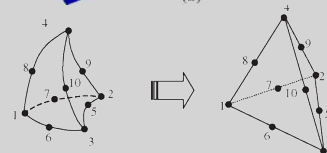
In contrast, finite element solids represent an object as **a solid mass of small elements**. This lets the solver realistically simulate bending, elasticity, internal mass, chipping, crumbling, and shattering. Solid objects can simulate stiff materials (like metal or wood), or elastic, rubbery, fluid, and floppy objects (like muscle and fat).



(a)



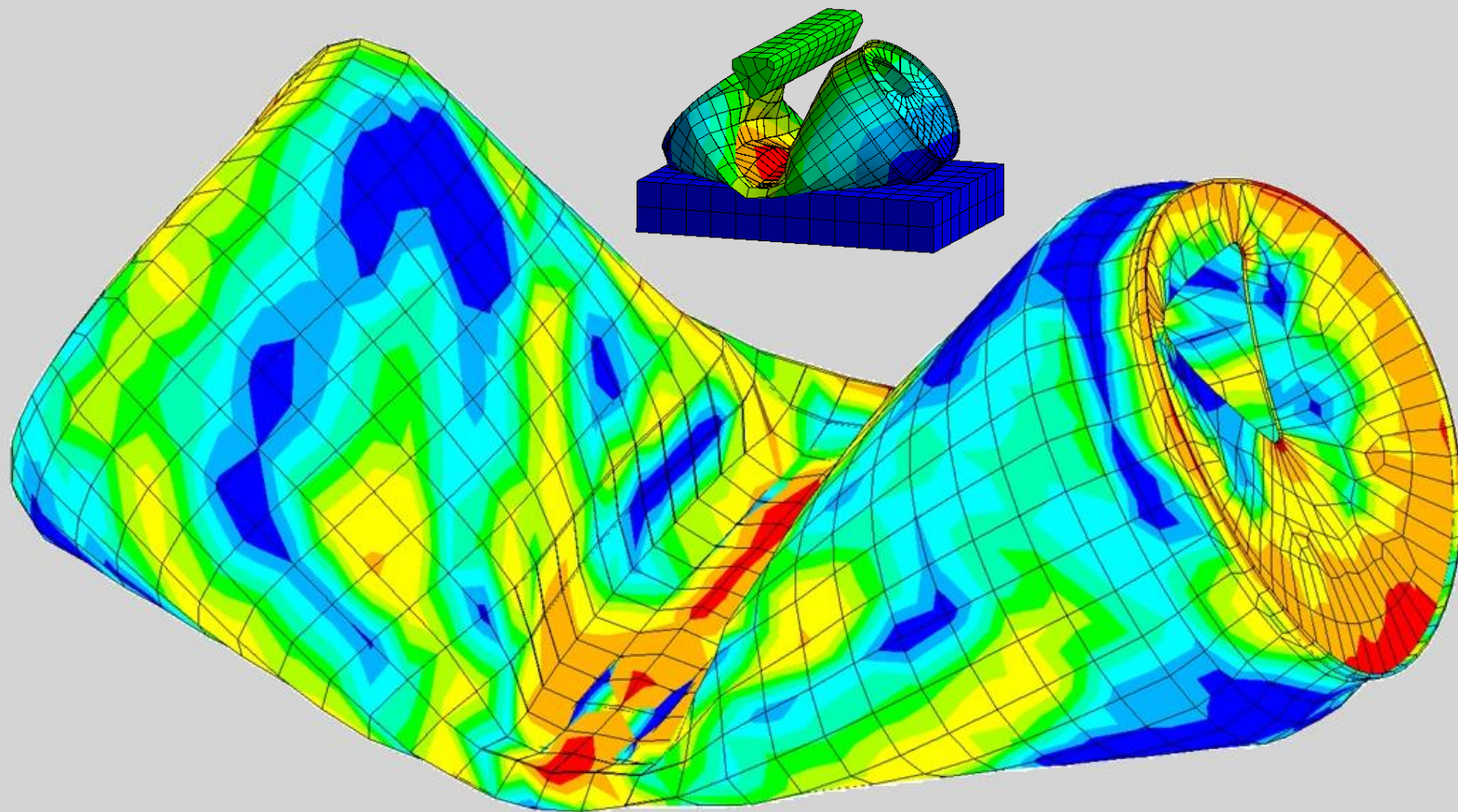
(b)





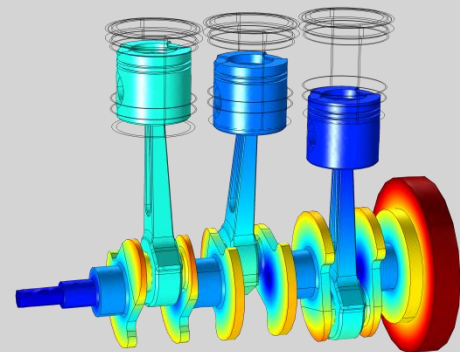
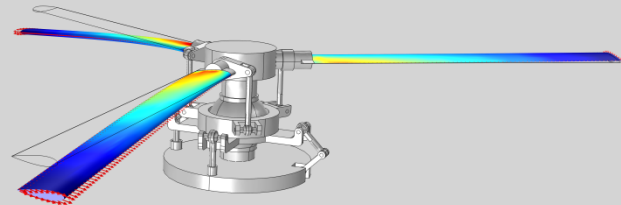
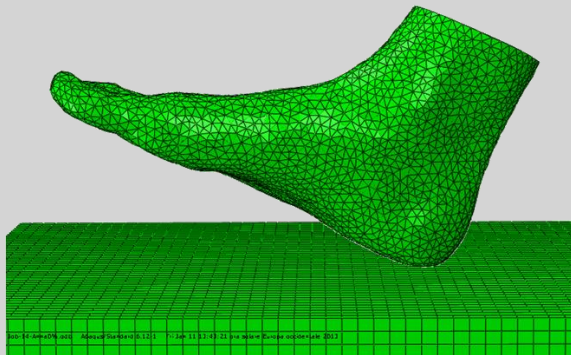
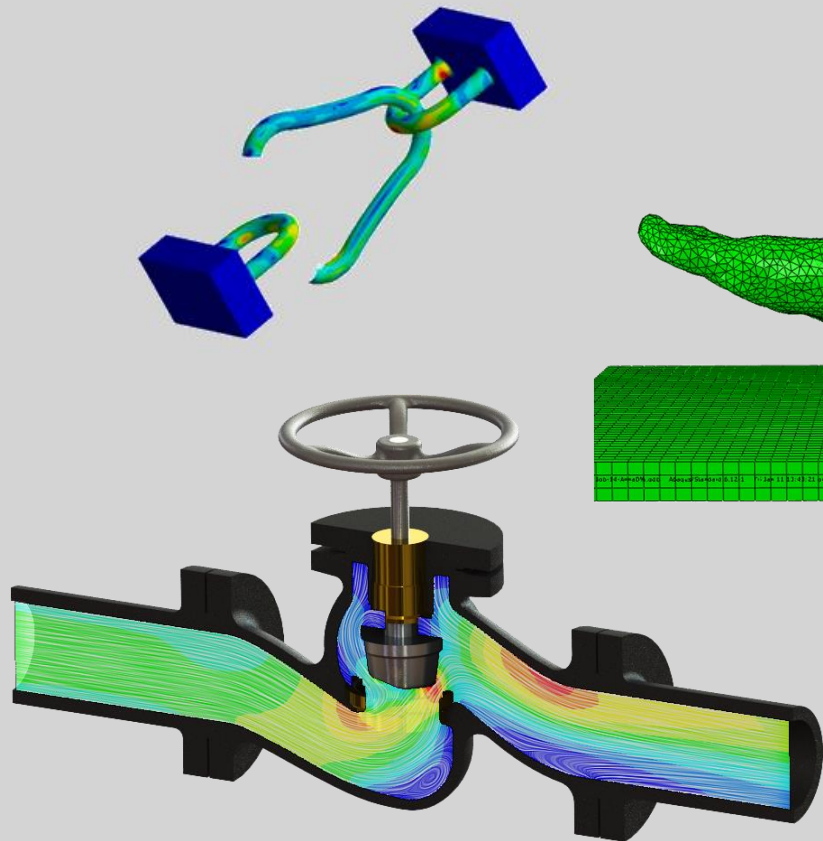
Blender/Python API

Finite Element Method



Blender/Python API

Finite Element Method



Houdini

Finite Element Method

Houdini 20.0 > Nodes > Dynamics nodes >



FEM Solver^{2.0} dynamics node

Since

16.0

PARAMETERS

Solve Method	Choose the solve method: GSL or GNL. The method GNL (global nonlinear) is more accurate than GSL (global single linearization) and can work at lower Substeps.
Simulation	Choose the simulation type: quasi-static or dynamic.
Integration	Choose the integration scheme: First Order or Second Order.
Substeps	The number of substeps per frame.
Collision Passes	The maximum number of collision detection & resolution passes.
Allow Collisions	Turn off to disable all support for collisions.
Allow Changing Rest	If turned on, the solver will consider changes in the constraints parameters: you can, for example, animate the strength of soft constraints to zero and “disable” them this way.
Allow Dynamic System	Turn off to optimize for constant topology, material coefficients and constraints.
Allow Fracturing	Turn off to disable all fracturing.



[Houdini 20.0](#) > [Nodes](#) > [Dynamics nodes](#) >



FEM Solid Object^{2.0} dynamics node

Creates a simulated FEM solid from geometry.

On this page

- [Overview](#)
- [Solid Object shelf tool](#)
- [Organic Mass shelf tool](#)
- [Parameters](#)
 - [Model](#)
 - [Deformation](#)
 - [Collisions](#)
 - [Fracturing](#)
 - [Drag](#)
 - [Attributes](#)
 - [Visualization](#)
 - [Creation](#)
 - [Attributes](#)
- [Locals](#)

Since


14.0

Blender/Python API

Finite Element Method

HoudiniTM
3D ANIMATION TOOLS

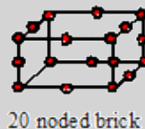
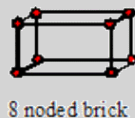
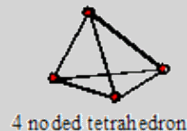
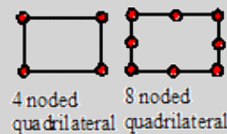
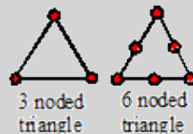
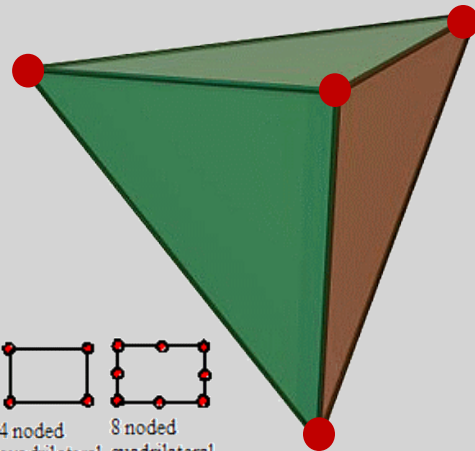
Houdini 16.5 > Nodes > Dynamics nodes >

 **Finite Element Solver** ^{1.0} dynamics node

Sets and configures a Finite Element solver.

Houdini uses meshes of tetrahedrons or tets (four-sided pyramids) to represent finite element solids.

In contrast to many other solvers, Houdini's finite element solver is resolution independent: the way an object moves and deforms is to a large degree independent of the density of tetrahedrons. In particular, the same settings on a Solid Object have similar results when applied to meshes of increasingly high resolution.





Finite elements vs. Springs

The Finite Element solver simulates the motion of deformable solids. The solver treats the simulated shape as a solid continuum, as opposed to a collision of particles. This means the internal mass and the internal forces are distributed over the entire shape, including the interiors of the tetrahedra.

The **finite-element simulation is based on internal stresses and strains** that occur within the simulated body. This contrasts with other types of solvers, such as particle-based mass-spring systems, where the mass is concentrated in the vertices and where internal forces are generated by springs between these vertices.

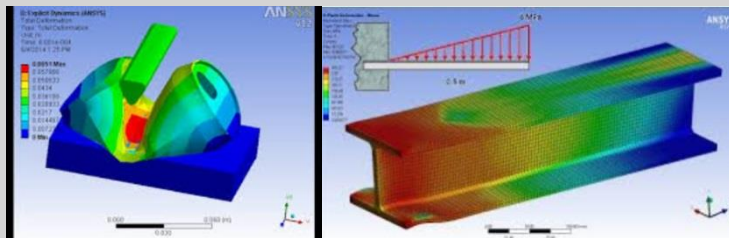


Blender/Python API Finite Element Method

The solid continuum approach of the finite element solver has several advantages over particle-based solvers:

The simulation behavior stays very consistent if the mesh resolution changes. This makes presets predictable. A simulation with a low-res mesh (cheap) is a good prediction of a simulation with a higher-res mesh.

The results of finite element simulation are realistic. The results of a finite element simulation is based on solid mechanics. Finite element systems tend to be more efficient at simulating stiff objects than position-based particle systems.





Blender/Python API Finite Element Method



FEniCSx is a popular open-source computing platform for solving partial differential equations (PDEs). FEniCSx enables users to quickly translate scientific models into efficient finite element code. With the high-level **Python** and C++ interfaces to FEniCSx, it is easy to get started, but FEniCSx offers also powerful capabilities for more experienced programmers.

NUMFOCUS
OPEN CODE = BETTER SCIENCE

Blender/Python API

Finite Element Method

Solving a PDE in FEniCS

As an illustration of how to program a simple PDE model with FEniCS, consider the Stokes equations in variational form:

$$\int_{\Omega} \text{grad } u : \text{grad } v \, dx - \int_{\Omega} p \, \text{div } v \, dx + \int_{\Omega} \text{div } u \, q \, dx = \int_{\Omega} f \cdot v \, dx.$$

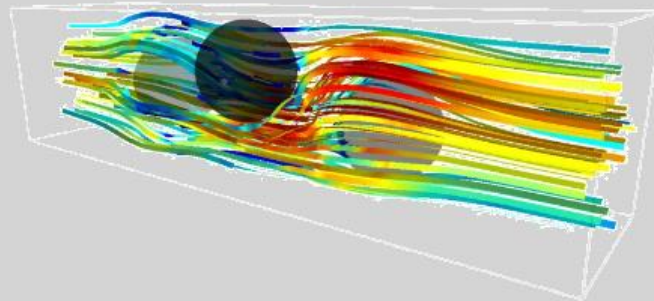
The variational problem is easily transcribed into Python using mathematical operators in FEniCS:

```
# Define function space
P2 = VectorElement('P', tetrahedron, 2)
P1 = FiniteElement('P', tetrahedron, 1)
TH = P2 * P1
W = FunctionSpace(mesh, TH)

# Define variational problem
(u, p) = TrialFunctions(W)
(v, q) = TestFunctions(W)
a = inner(grad(u), grad(v))*dx - p*div(v)*dx + div(u)*q*dx
L = dot(f, v)*dx

# Compute solution
w = Function(W)
solve(a == L, w, [bc1, bc0])
```

The above code snippet also shows how to define a suitable finite element function space, using continuous piecewise quadratic vector-valued functions for the velocity and continuous piecewise linear functions for the pressure (Taylor-Hood). The computational domain and mesh are also easily created with FEniCS, here defined by three spheres immersed in a 3D channel.

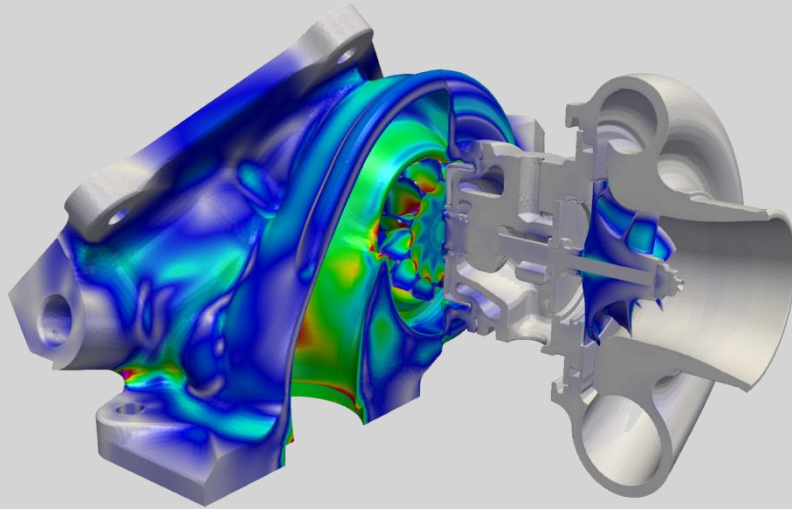


```
# Define domain
h = 0.25
r = 0.3*h
box = Box(Point(0, 0, 0), Point(1, h, h))
s0 = Sphere(Point(0.3, 0.50*h, 0.50*h), r)
s1 = Sphere(Point(0.5, 0.65*h, 0.65*h), r)
s2 = Sphere(Point(0.7, 0.35*h, 0.35*h), r)
domain = box - s0 - s1 - s2

# Generate mesh
mesh = generate_mesh(domain, 32)
```


Blender/Python API

Finite Element Method



Each component of the FEniCSx platform has been fundamentally designed for parallel processing. Executing a FEniCSx script in parallel is as simple as calling `mpirun -np 64 python script.py`



Solving a PDE in FEniCS

As an illustration of how to program a simple PDE model with FEniCS, consider the Stokes equations in variational form:

$$\int_{\Omega} \text{grad } \mathbf{u} : \text{grad } \mathbf{v} \, dx - \int_{\Omega} p \, \text{div } \mathbf{v} \, dx + \int_{\Omega} \text{div } \mathbf{u} \, q \, dx = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} \, dx.$$

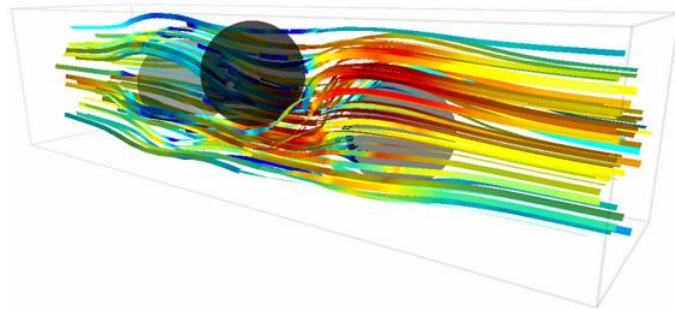
The variational problem is easily transcribed into Python using mathematical operators in FEniCS:

```
# Define function space
P2 = VectorElement('P', tetrahedron, 2)
P1 = FiniteElement('P', tetrahedron, 1)
VH = P2 * P1
W = FunctionSpace(mesh, VH)

# Define variational problem
(u, p) = TrialFunctions(W)
(v, q) = TestFunctions(W)
a = inner(grad(u), grad(v))*dx - p*div(v)*dx + div(u)*q*dx
L = dot(f, v)*dx

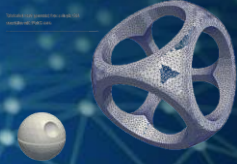
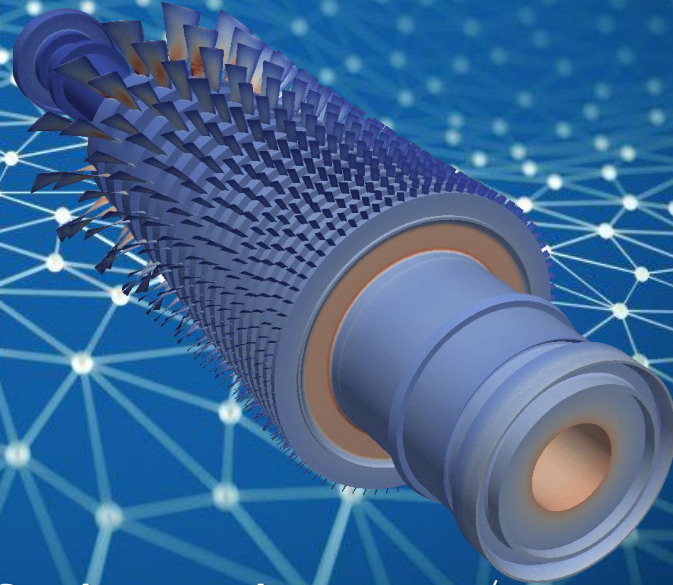
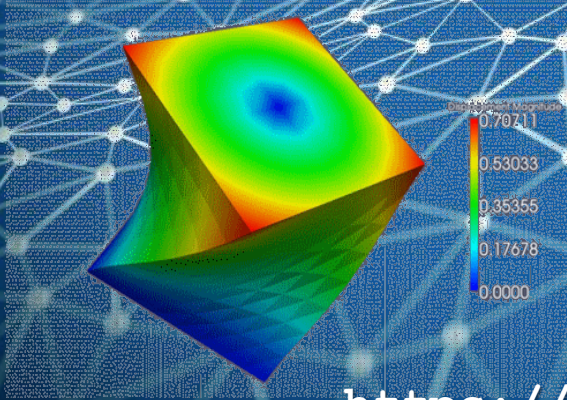
# Compute solution
w = Function(W)
solve(a == L, w, [bc1, bc0])
```

The above code snippet also shows how to define a suitable finite element function space, using continuous piecewise quadratic vector-valued functions for the velocity and continuous piecewise linear functions for the pressure (Taylor-Hood). The computational domain and mesh are also easily created with FEniCS, here defined by three spheres immersed in a 3D channel.





FENICS PROJECT



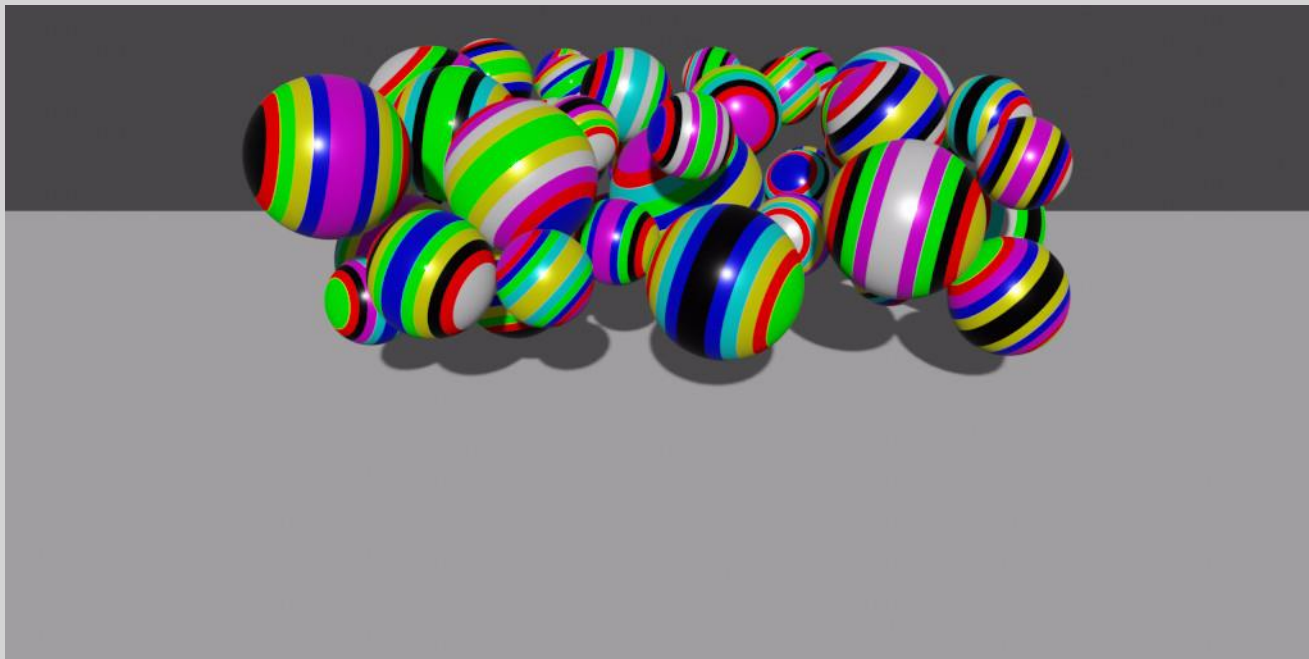
<https://fenicsproject.org/>

<http://femwiki.wikidot.com/fenics-intro:fenics-introduction>



Blender/Python API

Midterm Exam II: Soft Body Sphere



This is the second part of mid-term exam. Write a Blender script that simulates soft body balls dropping a flat plane. You need to render your result as a video.

Blender/Python API

Midterm Exam II: Soft Body Sphere

E-posta Sınav teslim tarihi : **20 Kasım 2024 Çarşamba** saat 23:59

Sınav yeri : **21 Kasım 2024 Perşembe** saat 18:30 Animasyon Lab
20 Kasım Çarşamba günü e-posta ile gönderdiğiniz programlar çalıştırılacak.

Blender Python'da yazılan programınızın gönderileceği e-posta adresi : **serdar.aritan@hacettepe.edu.tr**
serdar.aritan@gmail.com

Konu: BCO 602 Animasyon İçin Betik Diller <**Öğrenci No**>
İçerik: Blender da yazılmış programınız ZIP dosyası olarak (sıkıştırılmış)



Blender/Python API

Next week : Particle Physics

