



Blender - Python API

#11



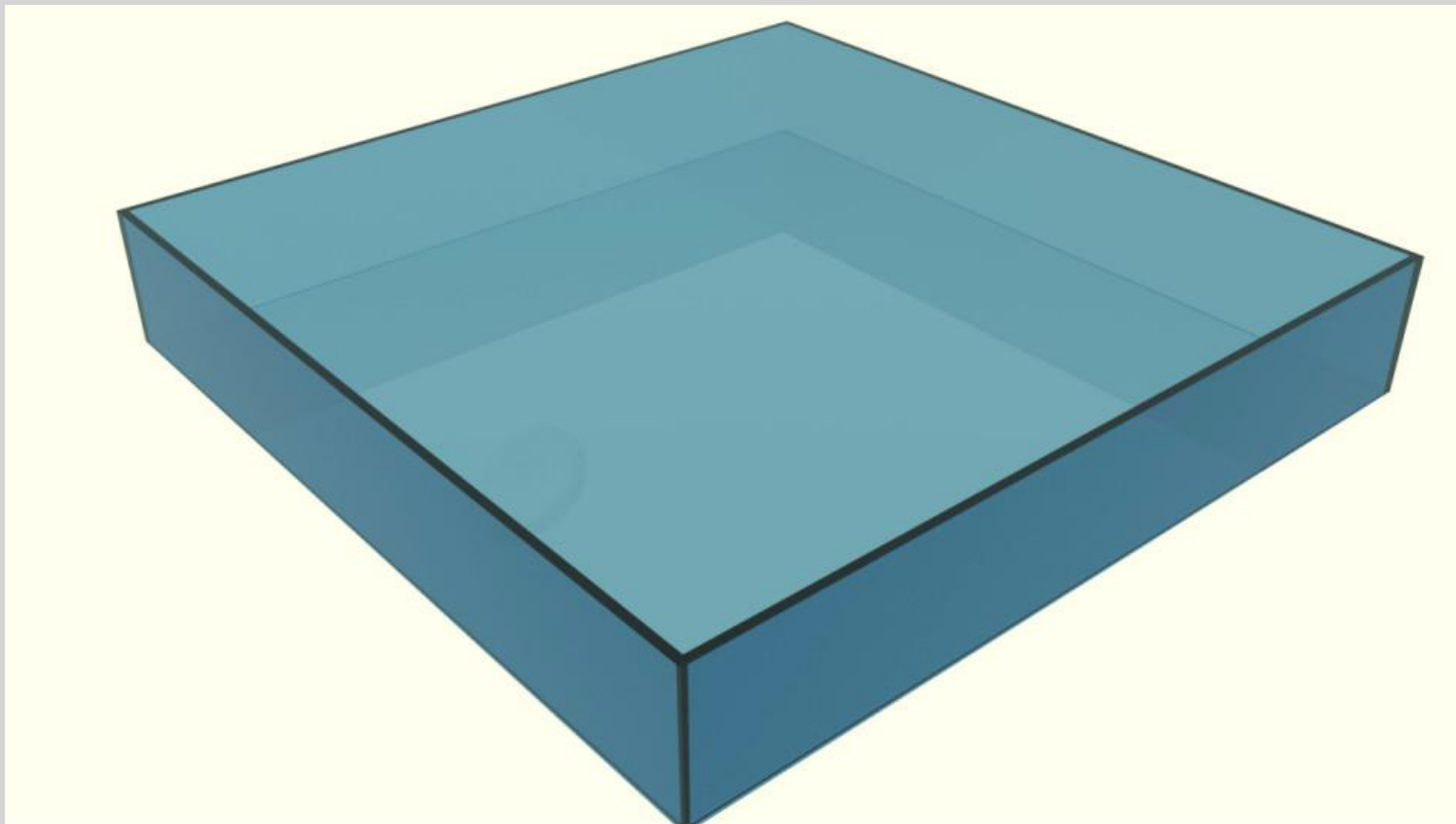
Serdar ARITAN

Department of Computer Graphics
Hacettepe University, Ankara, Turkey



Blender/Python API

Fluid Animation





Blender/Python API

Blender's Physics

Fluid animation can take a lot of time - the better you understand how it works, the easier it will be to estimate how the results will look. The algorithm used for Blender's fluid simulation is the Lattice Boltzmann Method (LBM); other fluid algorithms include Navier-Stokes (NS) solvers and Smoothed Particle Hydrodynamics (SPH) methods. LBM lies somewhere between these two. For Blender's LBM solver, the following things will make the simulation harder to compute:

- Large domains.

- Long duration.

- Low viscosities.

- High velocities.



Blender/Python API Fluid Animation

Workflow

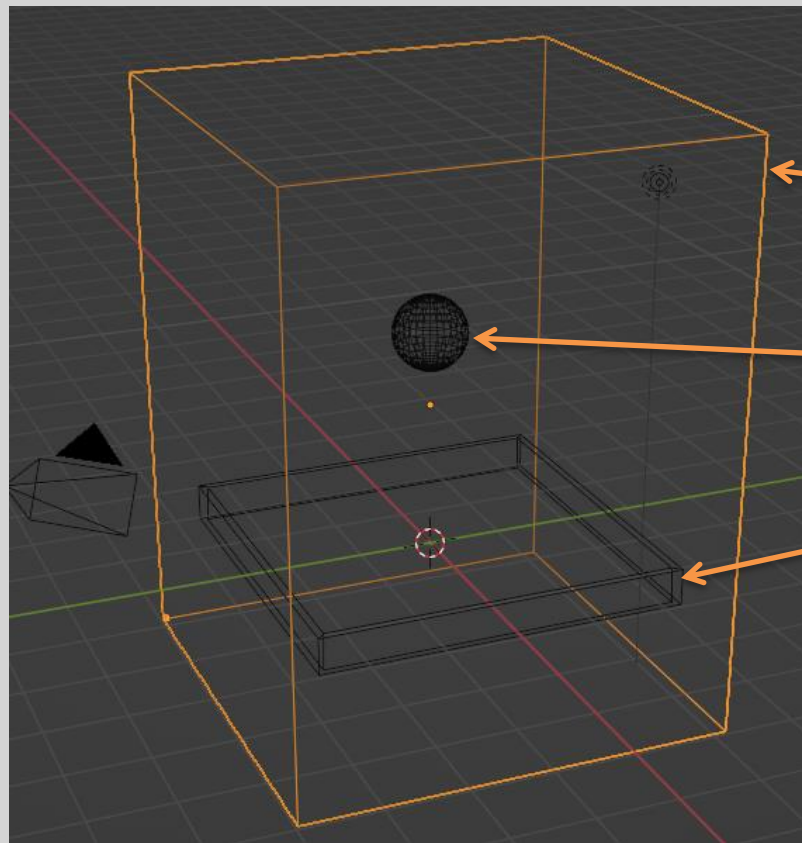
At least a Domain object and one Flow object are required to create a fluid simulation

In general, you follow these steps:

- Create a Domain object that defines the bounds of the simulation volume.
- Set up Flow objects which will emit fluid.
- Set up Effector objects to make the fluid interact with objects in the scene.
- Assign a material to the domain object.
- Save the blend-file.
- Bake the Cache for the simulation.



Blender/Python API Fluid Animation

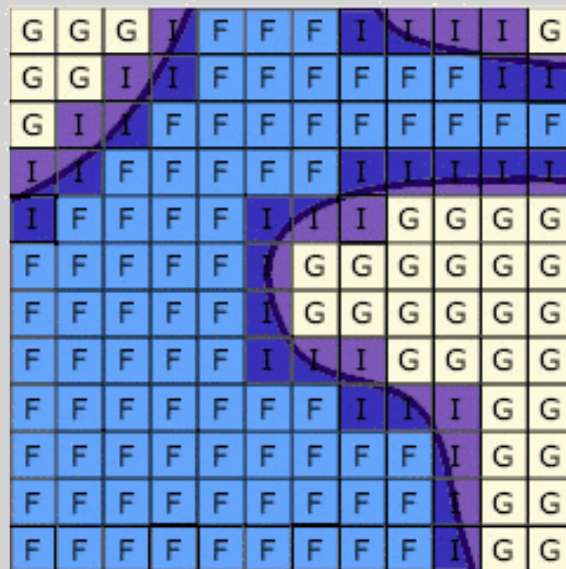
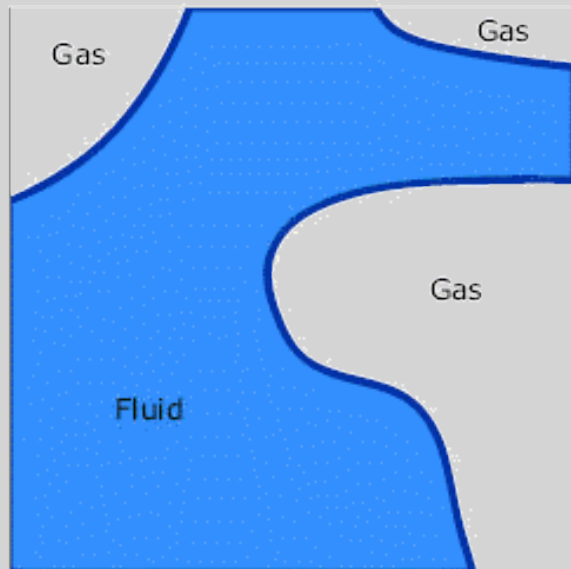


- Create a Domain object that defines the bounds of the simulation volume.
- Set up Flow objects which will emit fluid.
- Set up Effector objects to make the fluid interact with objects in the scene.



Blender/Python API Fluid Animation

Blender's fluid simulation is the Lattice Boltzmann Method (LBM)



G Cell filled with gas

I Interface cell, partly filled with fluid

F Cell filled with fluid



Blender/Python API Fluid Animation

The Lattice Boltzmann Method (LBM) is a numerical method based in kinetic equations formulated on a mesoscopic scale, which simulates fluid dynamics on a macroscopic scale.

The LBM was introduced by McNamara and Zanetti (1988), where the authors show the advantage of extending the Boolean dynamic of cellular automaton to work directly with floating point numbers representing probabilities of particle presence.

VOLUME 61, NUMBER 20

PHYSICAL REVIEW LETTERS

14 NOVEMBER 1988

Use of the Boltzmann Equation to Simulate Lattice-Gas Automata

Guy R. McNamara and Gianluigi Zanetti^(a)

The Research Institutes, The University of Chicago, 5640 South Ellis Avenue, Chicago, Illinois 60637

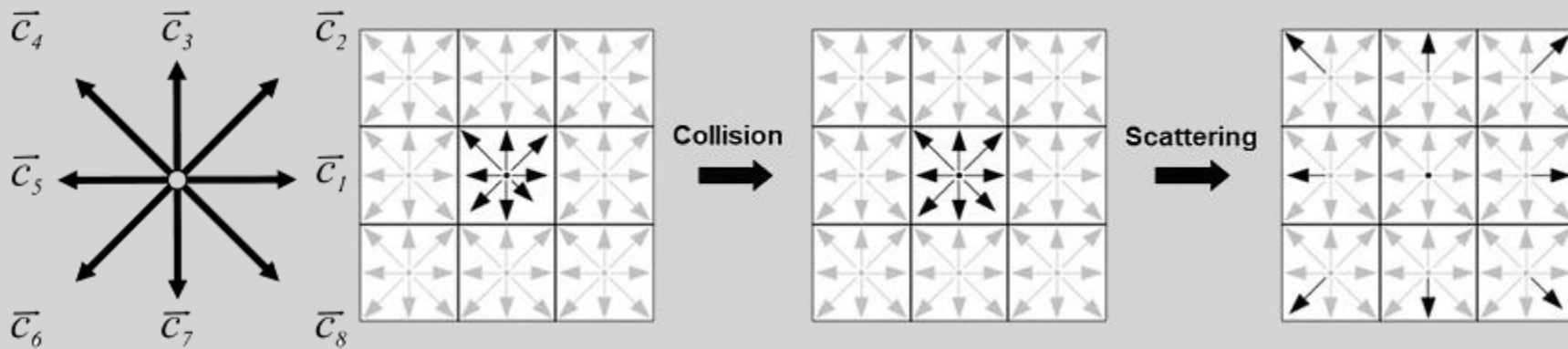
(Received 29 July 1988)

We discuss an alternative technique to the lattice-gas automata for the study of hydrodynamic properties, namely, we propose to model the lattice gas with a Boltzmann equation. This approach completely eliminates the statistical noise that plagues the usual lattice-gas simulations and therefore permits simulations that demand much less computer time. It is estimated to be more efficient than the lattice-gas automata for intermediate to low Reynolds number $R \lesssim 100$.



Blender/Python API Fluid Animation

The lattice Boltzmann model known as D2Q9 has eight nonzero motion directions and the possibility of having a resting particle.





D2Q9 Model algorithm.

```
Dx = 50, Dy = 50, lattice[50][50]. f [9] //each node has 9 fi's
// INITIALIZATION STEP
density = 1.0
velocity[50][50] = 0.0
for each x in [0, Dx - 1] do
  for each y in [0, Dy - 1] do
    for each i in [0, 8] do
      lattice[x][y]. f [i] = f eq(density, velocity[x][y], i)
    end for
  end for
end for
```



```
// SOLVER
repeat
// SAVE VELOCITY FIELD FOR STOP CONDITION
for each x in [0, Dx - 1] do
  for each y in [0, Dy - 1] do
    old[x][y] = velocity[x][y]
  end for
end for
// INNER LATTICE DYNAMICS
for each x in [1, Dx - 2] do
  for each y in [1, Dy - 2] do
    Scattering of node [x][y] for its neighbors
    Collision at node [x][y]
  end for
end for
Treats boundary conditions
// STOP CONDITION
max = 0
for each x in [1, Dx - 2] do
  for each y in [1, Dy - 2] do
    norm = |velocity[x][y] - old[x][y]|
    if (norm > max) then max = norm end if
  end for
end for
until (max >= delta)
```



Blender/Python API

Fluid Animation

The Domain Object

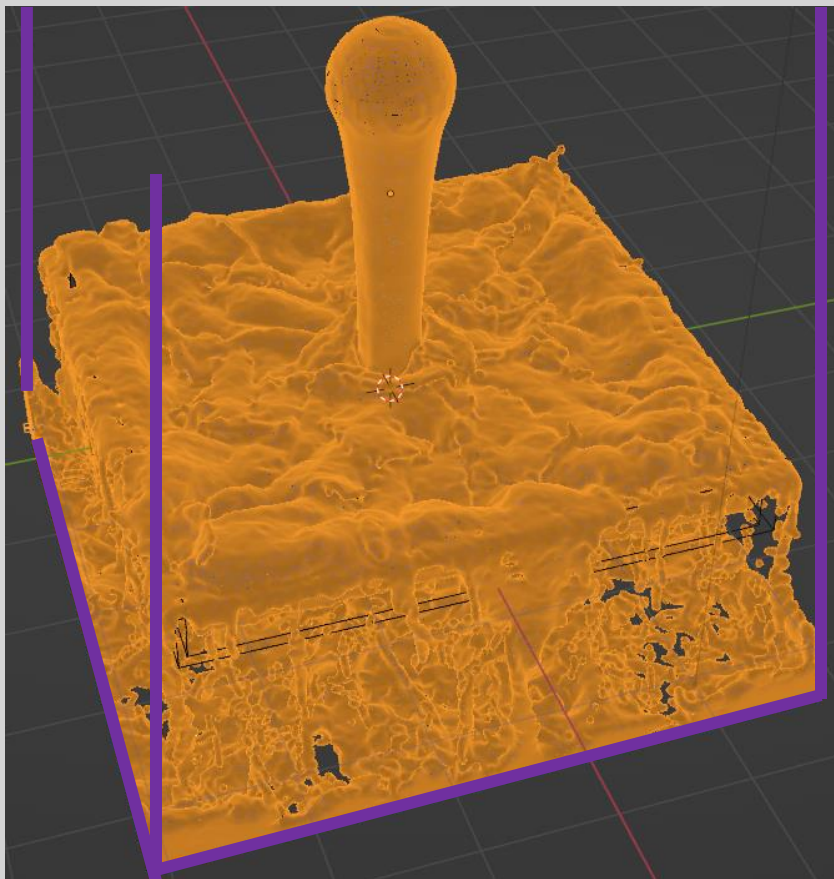
The bounding box of the object serves as **the boundary of the simulation**. All fluid objects must be in the domain. Fluid objects outside the domain will not bake. No tiny droplets can move outside this domain; it's as if the fluid is contained within the 3D space by invisible force fields. There can be only a single fluid simulation domain object in the scene.

The shape of the object does not matter because **it will always be treated like a box** (The lengths of the bounding box sides can be different). So, usually there will not be any reason to use another shape than a box. If you need obstacles or other boundaries than a box to interfere with the fluid flow, you need to insert additional obstacle objects inside the domain boundary.)



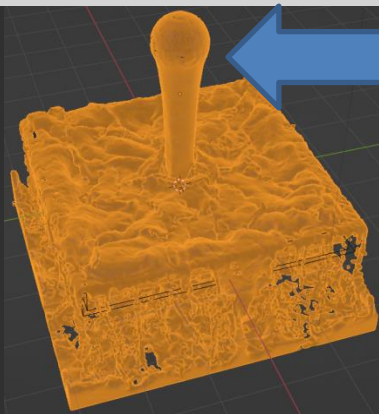
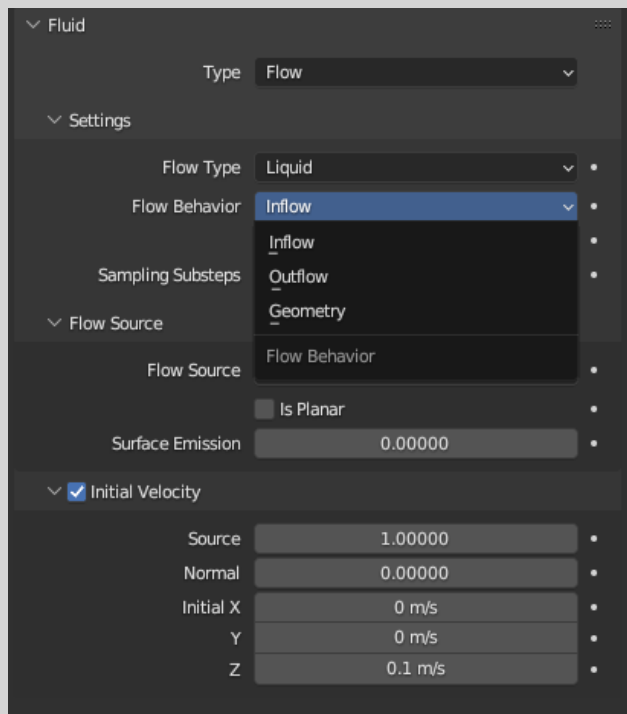
Blender/Python API

Fluid Animation





Blender/Python API Fluid Animation



Inflow: This object will **put fluid** into the simulation, like a water tap.

Initial Velocity

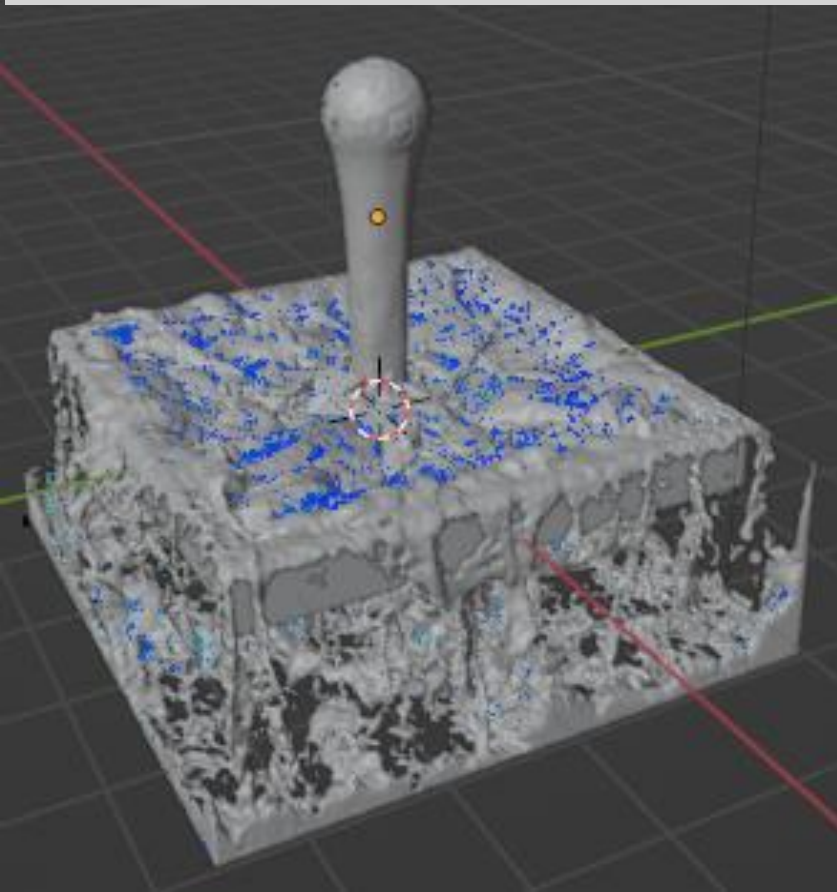
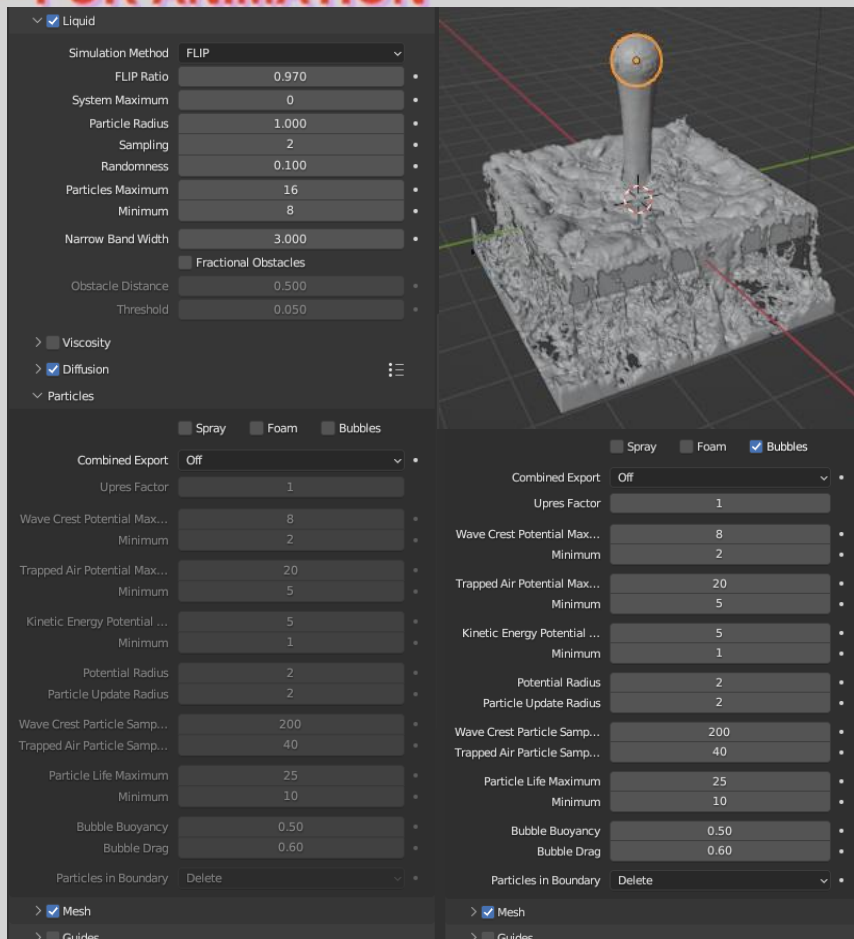
Speed of the fluid that is created inside of the object.

Outflow : Any fluid that enters the region of this object will be **deleted** (think of a drain or a black hole).

This can be useful in combination with an inflow to prevent the whole domain from filling up. When enabled, this is like a tornado (waterspout) or “**wet vac**” vacuum cleaner, and the part where the fluid disappears will follow the object as it moves around.

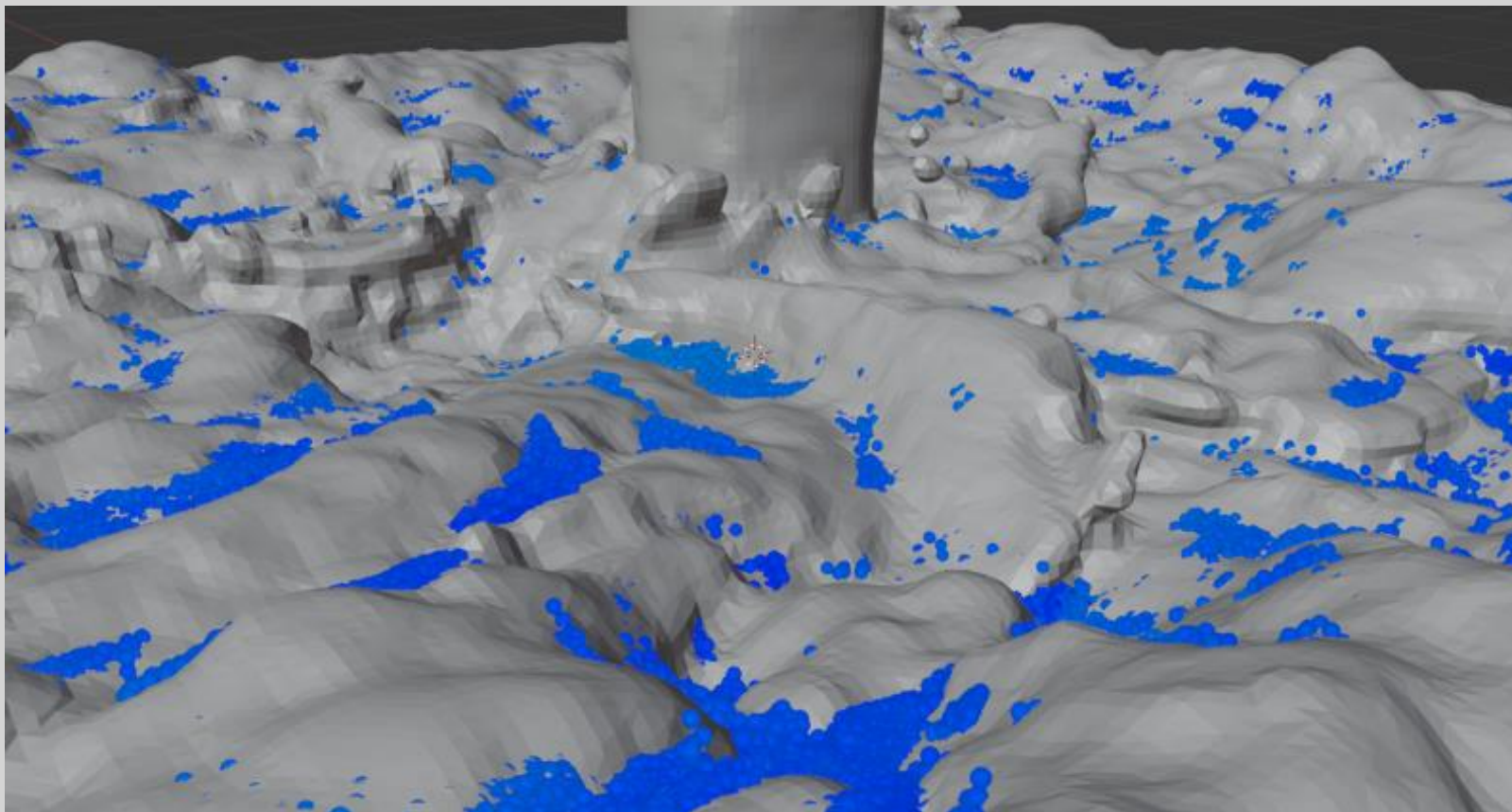


Blender/Python API Fluid Animation





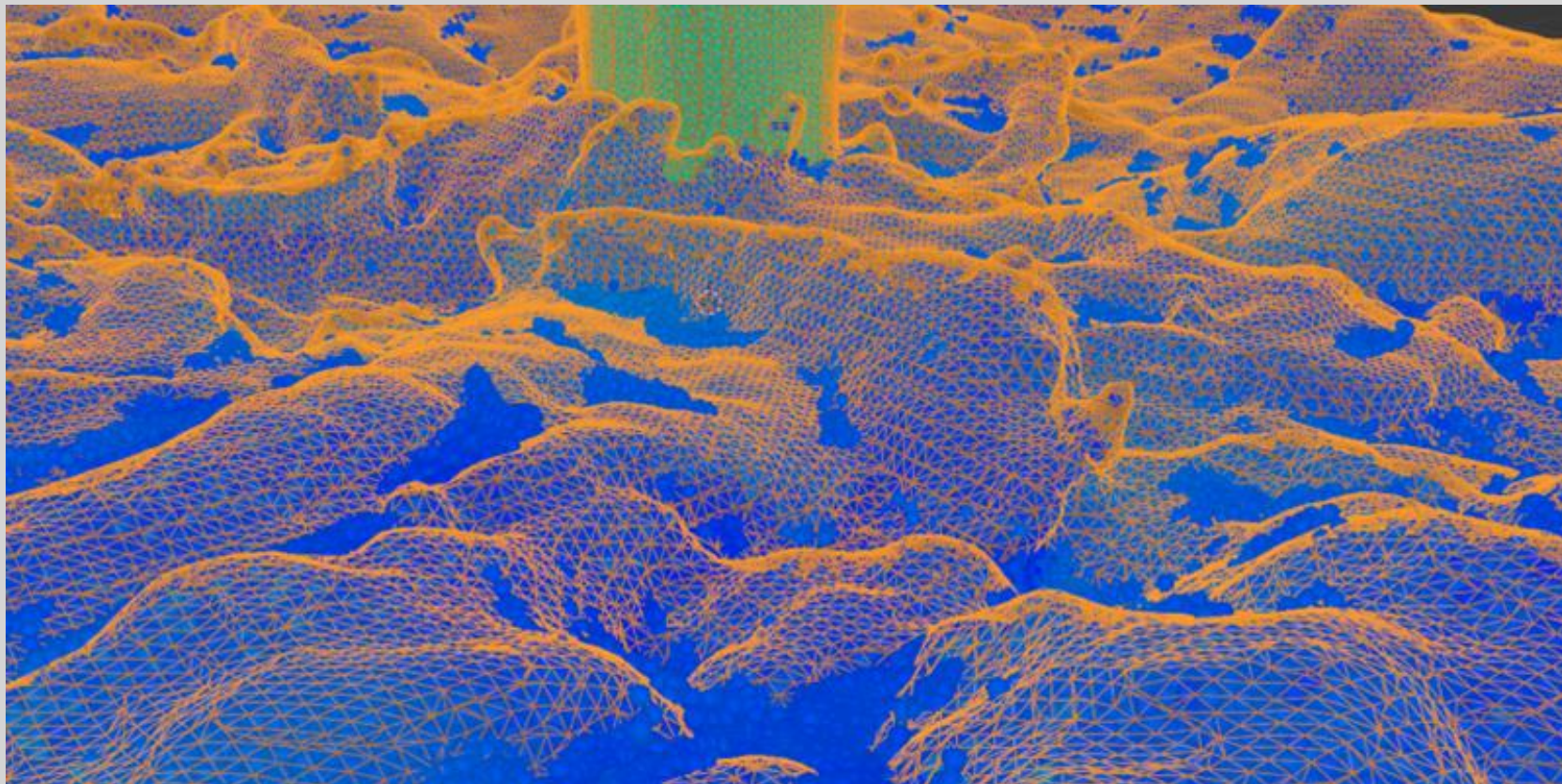
Blender/Python API Fluid Animation



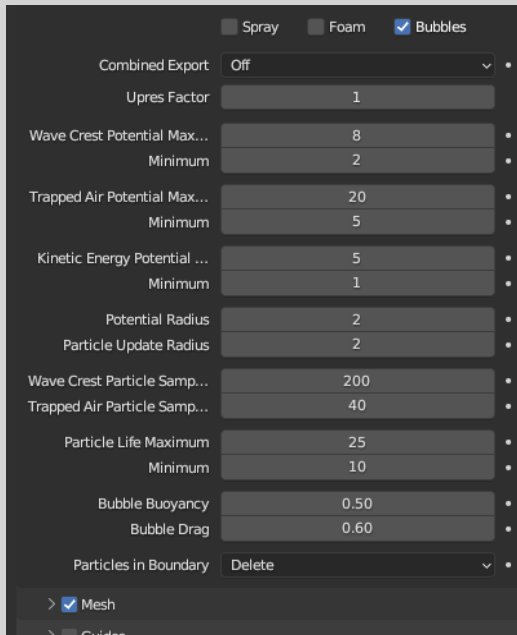


Blender/Python API

Fluid Animation



Blender/Python API Fluid Animation



Home / Physics / Fluid / Type / Domain / Liquid Settings / Particles

Particles

Spray

Create spray particles during the secondary particle simulation. Spray particles are those that appear to fly through the air above the liquid surface when there is a bigger splash.

Foam

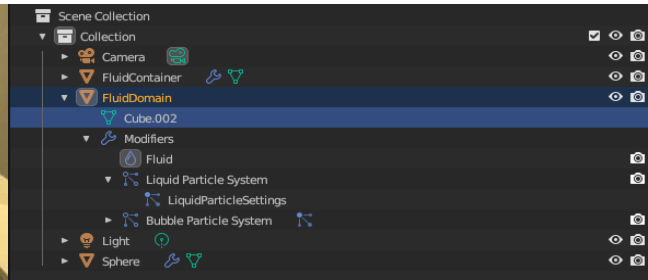
Create foam particles during the secondary particle simulation. Foam particles are those that solely move on the liquid surface.

Bubbles

Create bubble particles during the secondary particle simulation. Bubble particles are those that move below the liquid surface.

Note

Enabling a secondary particle type will also create a particle system for that type of particles. Disabling a particle type will delete this particle system including its settings.





Blender/Python API Fluid Animation

Baking always starts at Frame #1

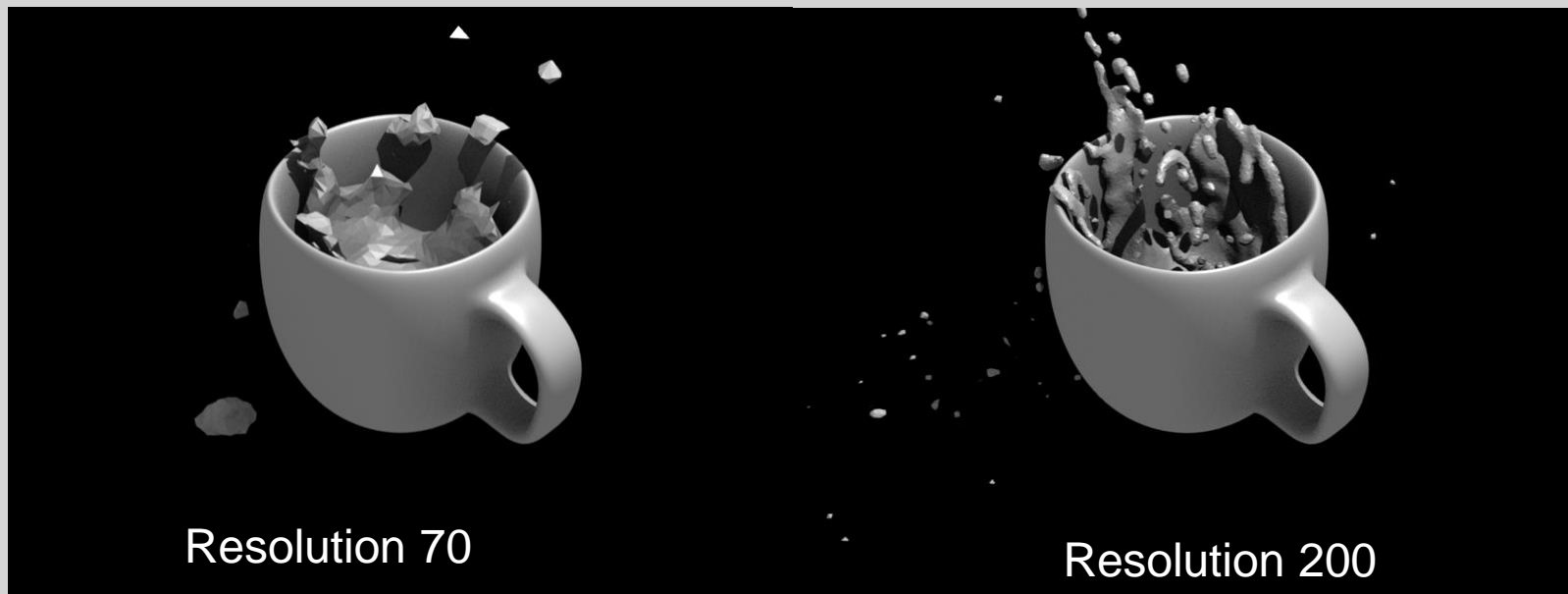
The fluid simulator disregards the Start setting in the Animation panel, it will always bake from frame 1. If you wish the simulation to start later than frame 1, you must key the fluid objects in your domain to be inactive until the frame you desire to start the simulation.

Baking always ends at the End Frame set in the Animation panel

If the frame rate is set to 25 frames per second, and ending time is 4.0 seconds, then you should (if your start time is 0) set your animation to end at frame $4.0 \times 25 = 100$



Blender/Python API Fluid Animation



Render resolution

The granularity at which the actual fluid simulation is performed. This is probably the most important setting for the simulation as it determines the amount of details in the fluid, the memory and disk usage as well as computational time.

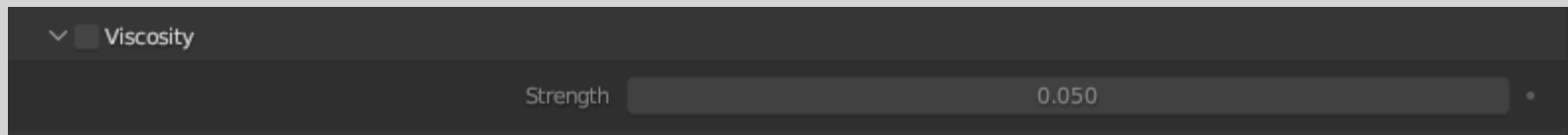


Blender/Python API Fluid Animation

Viscosity

The “***thickness***” of the fluid and actually the force needed to move an object of a certain surface area through it at a certain speed.

For manual entry, please note that the normal real-world viscosity (the so-called dynamic viscosity) is measured in Pascal-seconds (Pa.s), or in Poise units (P, equal to 0.1 Pa.s, pronounced pwaz, from the Frenchman Jean-Louis Poiseuille, who discovered the laws on “the laminar flow of viscous fluids”), and commonly centiPoise units (cP, equal to 0.001 Pa.s, sentipwaz). **Blender**, on the other hand, uses the kinematic viscosity (which is dynamic viscosity in Pa.s, divided by the density in kg.m-3, unit m².s-1).





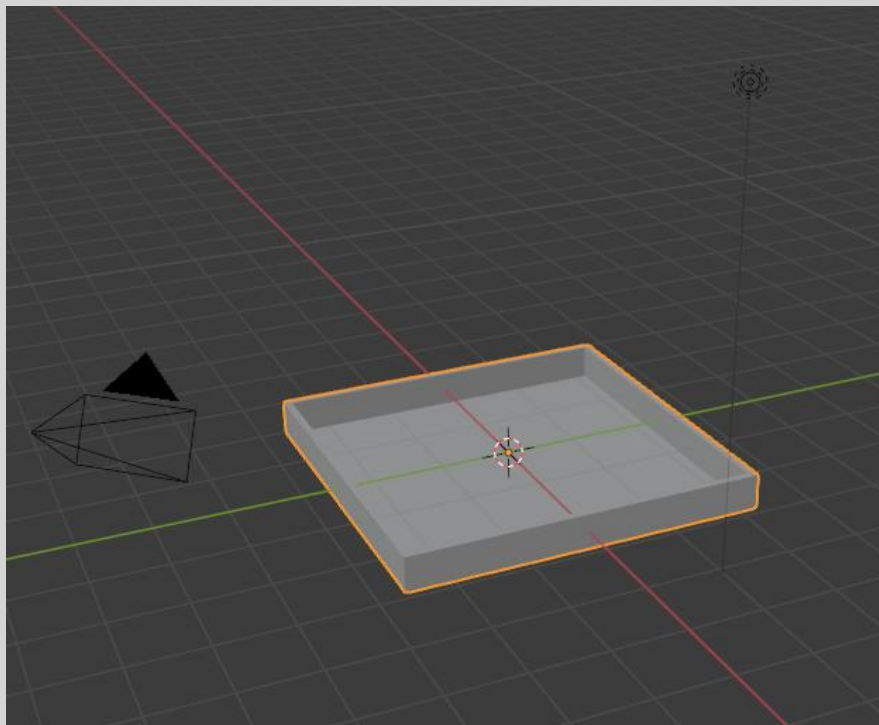
Blender/Python API Fluid Animation

The table below gives some examples of fluids together with their dynamic and kinematic viscosities.

Fluid	Dynamic viscosity (in cP)	Kinematic viscosity (Blender, in $\text{m}^2.\text{s}^{-1}$)
Water (20° C)	1.002×10^0 (1.002)	1.002×10^{-6} (0.000001002)
Oil SAE 50	5.0×10^2 (500)	5.0×10^{-5} (0.00005)
Honey (20° C)	1.0×10^4 (10,000)	2.0×10^{-3} (0.002)
Chocolate Syrup	3.0×10^4 (30,000)	3.0×10^{-3} (0.003)
Ketchup	1.0×10^5 (100,000)	1.0×10^{-1} (0.1)
Melting Glass	1.0×10^{15}	1.0×10^0 (1.0)

Blender/Python API

Fluid Container





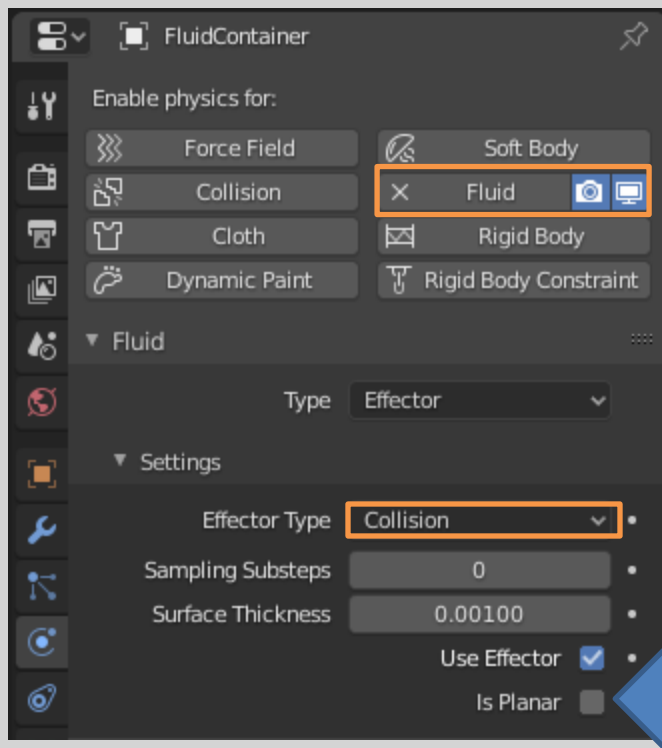
```
# Fluid Container Design START
bpy.ops.mesh.primitive_cube_add(size=1, location=(0, 0, 0))
bpy.context.active_object.name = 'FluidContainer'
# Switching to EDIT mode to read mesh data
bpy.ops.object.mode_set(mode = 'EDIT')
# Deselect all verts, edges, faces
bpy.ops.mesh.select_all(action="DESELECT")
# Collision Cube is a container for the physics
collisionCube = bpy.context.active_object
meColl = collisionCube.data
# Register bmesh object and select various parts
bm = bmesh.from_edit_mesh(meColl)
bm.faces.ensure_lookup_table()
bm.select_mode = {'FACE'}
```



```
# Select the top face of the cube: get the Z coordinates
for f in bm.faces:
    print(f.calc_center_bounds()[2])
    #Z = (collisionCube.matrix_world * face.center)[2]
    if f.calc_center_bounds()[2] > 0.0:
        f.select = True
bm.select_flush_mode()
bpy.ops.mesh.delete(type='FACE')
bpy.ops.object.mode_set(mode = 'OBJECT')
bpy.ops.transform.resize(value=(5, 5, 0.5))
bpy.ops.object.modifier_add(type='SOLIDIFY')
bpy.context.object.modifiers["Solidify"].thickness = 0.03
bpy.ops.object.modifier_apply(modifier="Solidify")
# Fluid Container Design STOP
```



Blender/Python API Effector Collision



Use Effector

Enables or disables the effector object effect on the fluid, this property is useful for animations to selectively enable and disable when the effector affects the fluid.

Is Planar

Defines the effector as either a single dimension object i.e. a plane or the mesh is non-manifold. This ensures that the fluid simulator will give the most accurate results for these types of meshes. A manifold mesh can also be declared as planar. The fluid solver will then ignore the volume inside the mesh and just emit fluid from the mesh sides.



Blender/Python API Effector Collision

```
# Apply Obstacle Modifier to the Fluid Container
```

```
bpy.ops.object.modifier_add(type='FLUID')
```

```
bpy.context.object.modifiers["Fluid"].fluid_type = 'EFFECTOR'
```

```
bpy.context.object.modifiers["Fluid"].effector_settings.effector_type =  
'COLLISION'
```

```
bpy.context.object.modifiers["Fluid"].effector_settings.surface_distance = 0.001
```

```
bpy.context.object.modifiers["Fluid"].effector_settings.use_plane_init = True
```

Effector Type

Collision

Objects of this type will collide with fluid.

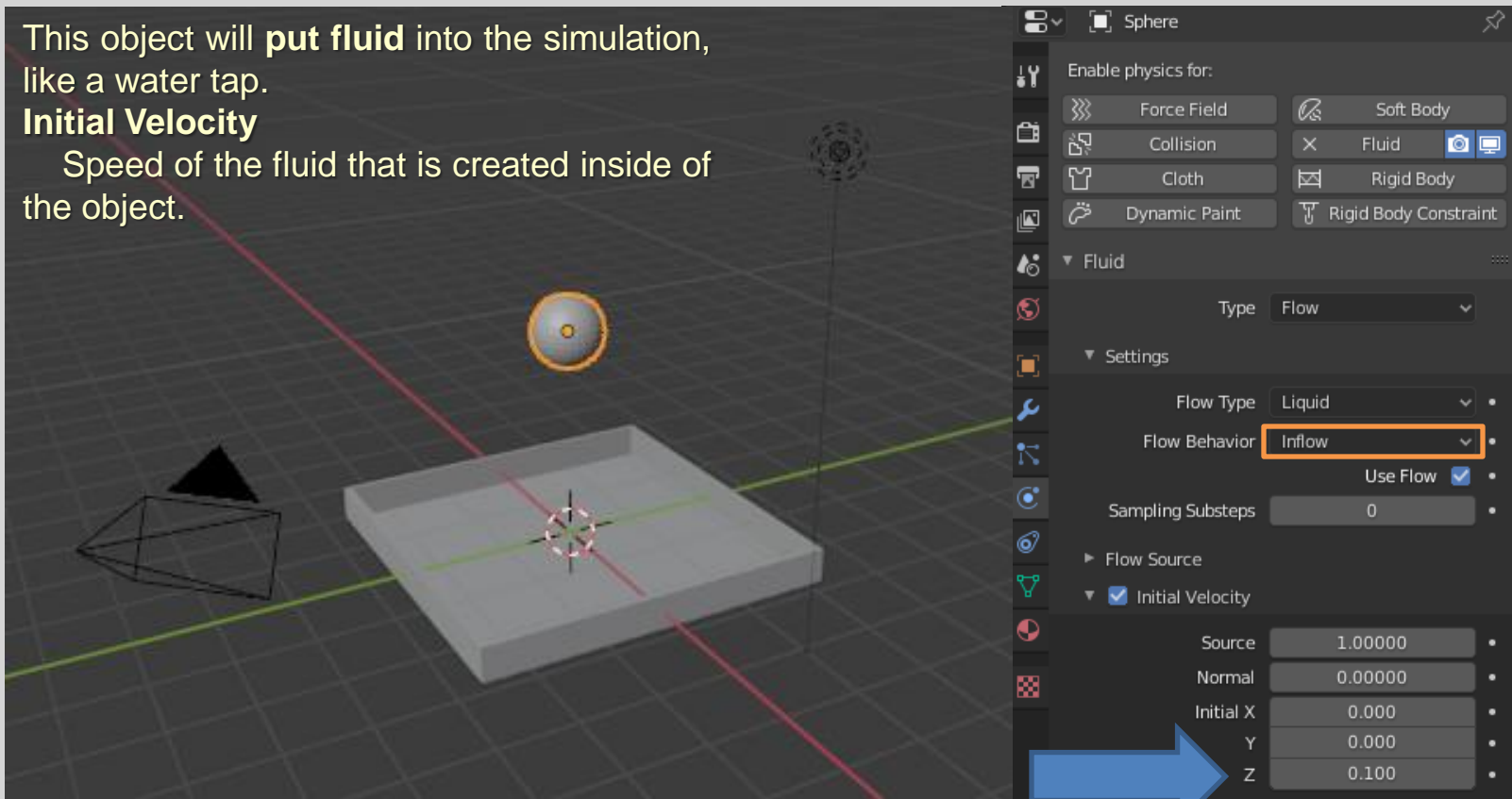


Blender/Python API Fluid Source

This object will **put fluid** into the simulation, like a water tap.

Initial Velocity

Speed of the fluid that is created inside of the object.

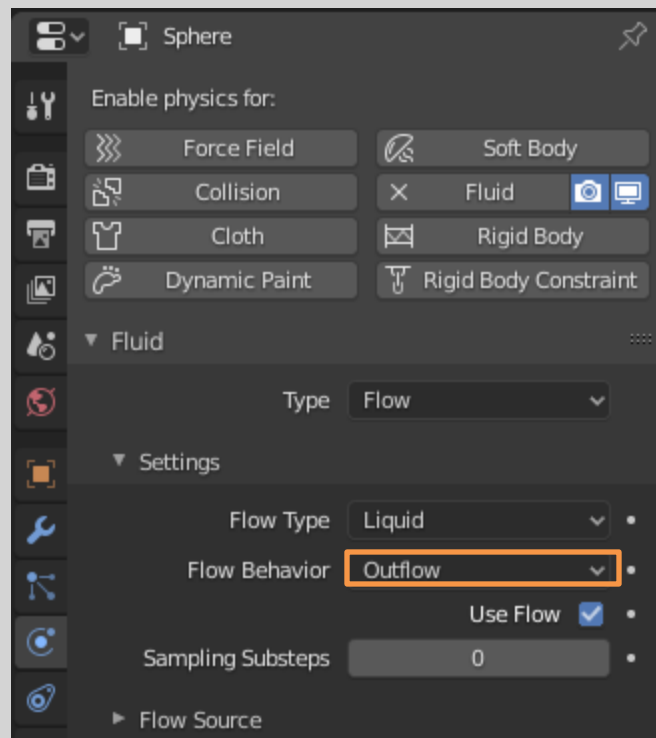




Blender/Python API Fluid Source

Any fluid that enters the region of this object will be **deleted** (think of a drain or a black hole).

This can be useful in combination with an inflow to prevent the whole domain from filling up. When enabled, this is like a tornado (waterspout) or “**wet vac**” vacuum cleaner, and the part where the fluid disappears will follow the object as it moves around.





Blender/Python API Fluid Source

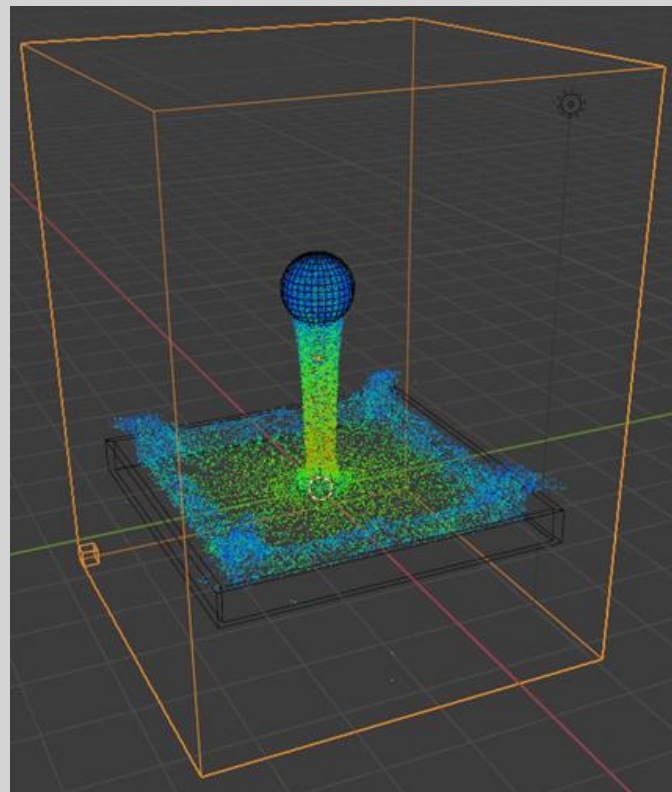
Design Fluid Source

```
bpy.ops.mesh.primitive_uv_sphere_add(radius=0.5, location=(0, 0, 3))  
bpy.ops.object.modifier_add(type='FLUID')  
bpy.context.object.modifiers["Fluid"].fluid_type = 'FLOW'  
bpy.context.object.modifiers["Fluid"].flow_settings.flow_type = 'LIQUID'  
bpy.context.object.modifiers["Fluid"].flow_settings.flow_behavior = 'INFLOW'  
bpy.context.object.modifiers["Fluid"].flow_settings.use_initial_velocity = True  
bpy.context.object.modifiers["Fluid"].flow_settings.velocity_coord[2] = 0.1
```



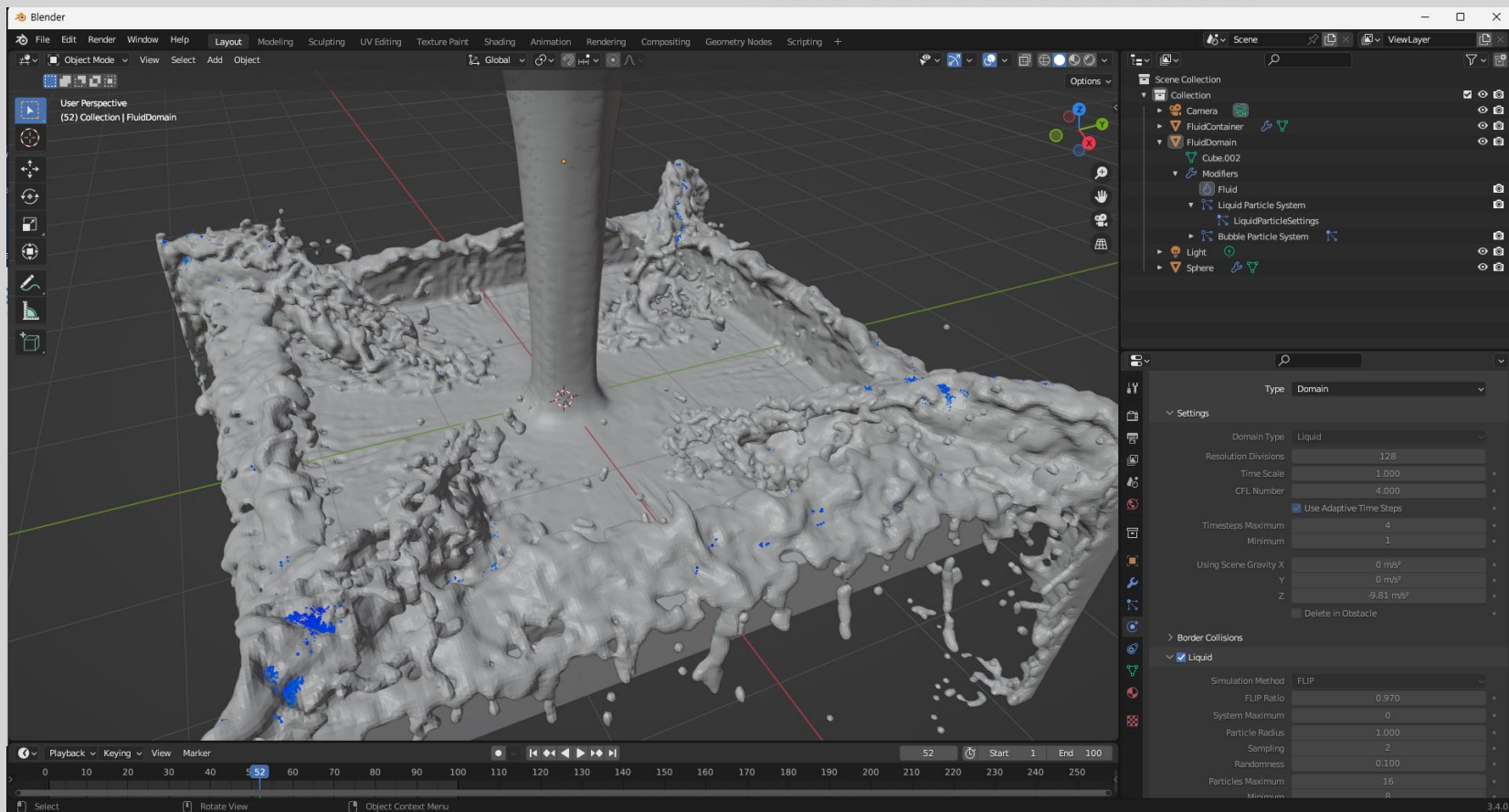
Blender/Python API Fluid Simulation Domain

The bounding box of the object serves as the boundary of the simulation. All fluid objects must be in the domain. Fluid objects outside the domain will not bake. No tiny droplets can move outside this domain; it's as if the fluid is contained within the 3D space by invisible force fields. **There can be only a single fluid simulation domain object in the scene.**



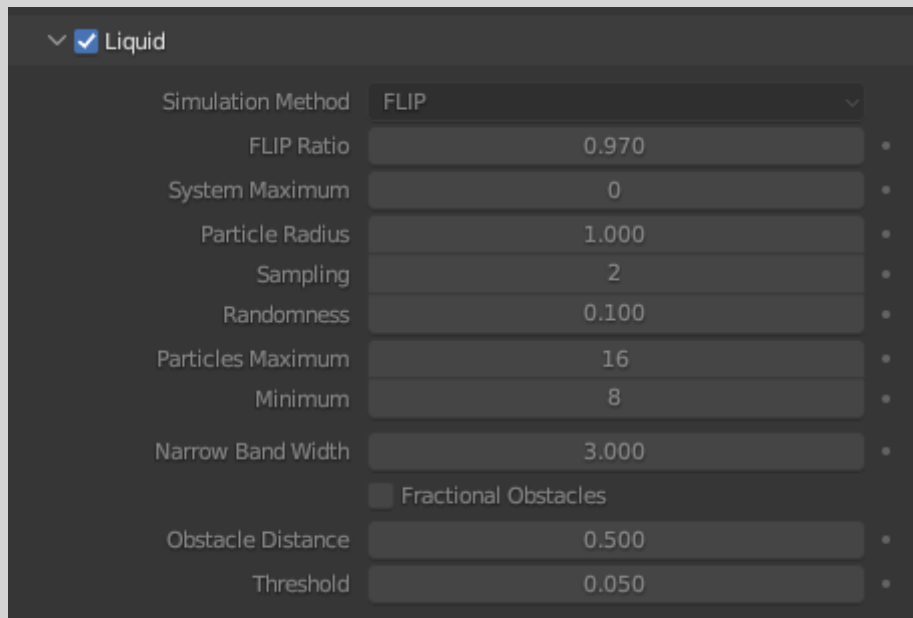
Blender/Python API

Fluid Simulation Domain





Blender/Python API Fluid Simulation Domain



FLIP is an adaptation to fluids of the implicit moment method for simulating plasmas, in which particles carry everything necessary to describe the fluid. Using the particle data, **Lagrangian moment equations are solved on a grid.** The solutions are then used to advance the particle variables from time step to time step.



Blender/Python API Fluid Simulation Domain

FLIP

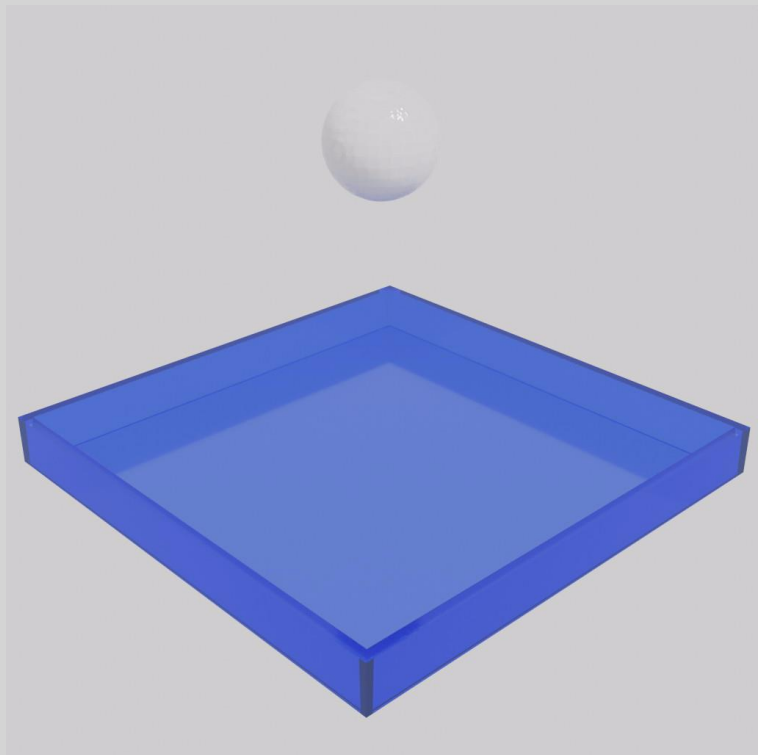
Produces a very **splashy** simulation with lots of particles dispersed in the air.

APIC

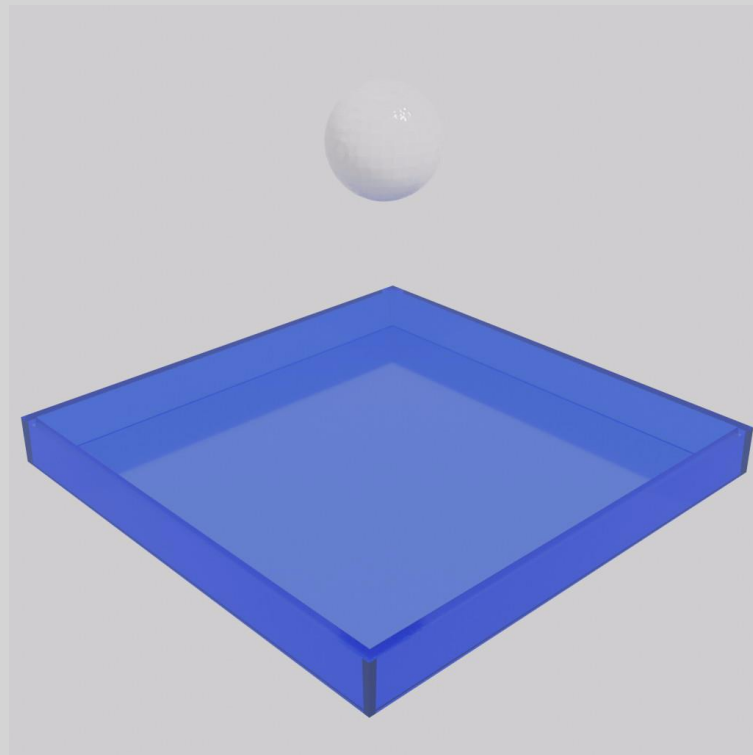
Produces a very energetic but also more stable simulation. Vortices within the liquid will be preserved better than with FLIP.

Blender/Python API

Fluid Simulation Domain



FLIP



APIC

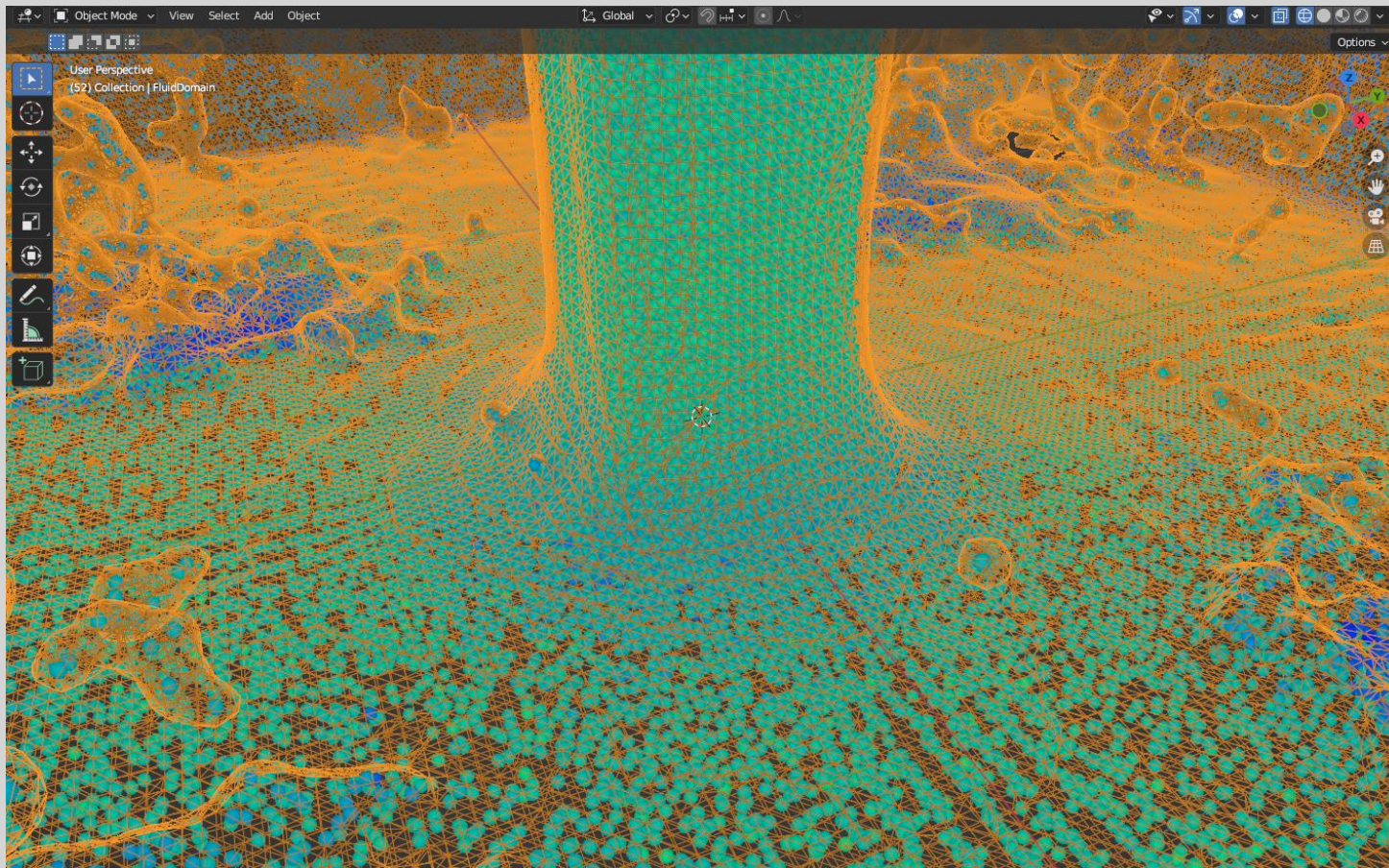


Blender/Python API Fluid Simulation Domain

```
# Set the Fluid Simulation Domain
```

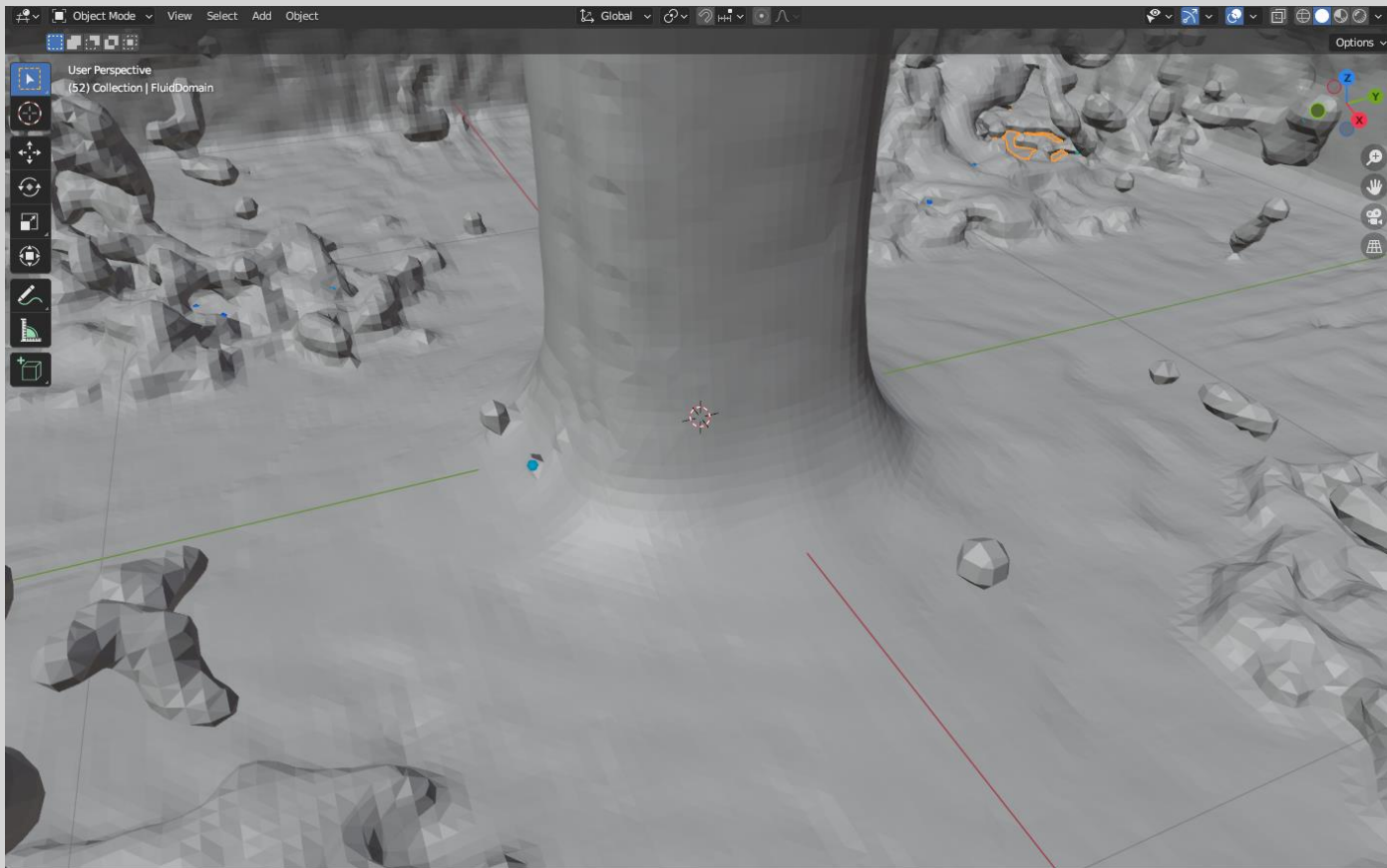
```
bpy.ops.mesh.primitive_cube_add(size=1, location=(0, 0, 2))  
bpy.context.active_object.name = 'FluidDomain'  
bpy.ops.transform.resize(value=(6, 6, 8))  
bpy.context.object.display_type = 'WIREFRAME'  
bpy.ops.object.modifier_add(type='FLUID')  
bpy.context.object.modifiers["Fluid"].fluid_type = 'DOMAIN'  
bpy.context.object.modifiers["Fluid"].domain_settings.domain_type = 'LIQUID'  
bpy.context.object.modifiers["Fluid"].domain_settings.resolution_max = 64 #128  
bpy.context.object.modifiers["Fluid"].domain_settings.use_diffusion = True  
bpy.context.object.modifiers["Fluid"].domain_settings.use_mesh = True  
bpy.context.object.modifiers["Fluid"].domain_settings.mesh_scale = 3
```

Blender/Python API Fluid Simulation



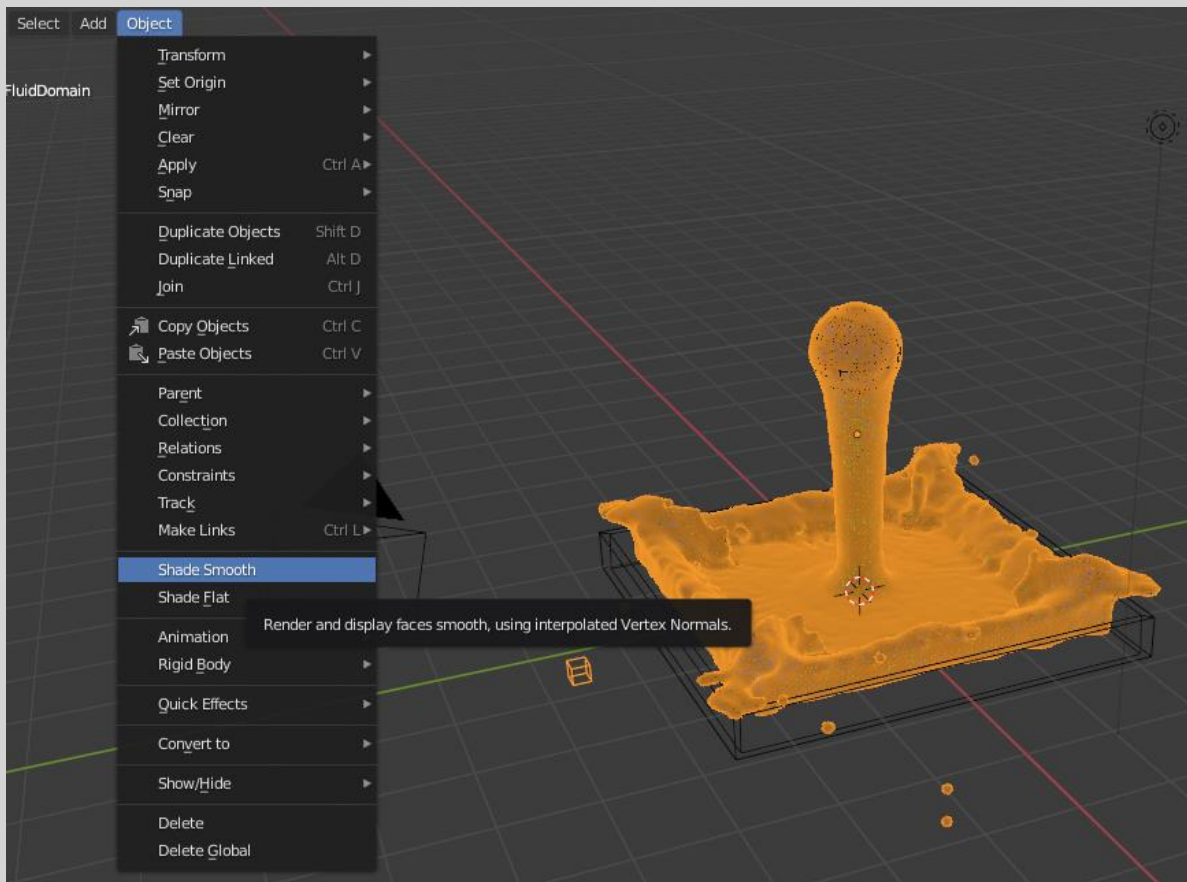


Blender/Python API Fluid Simulation

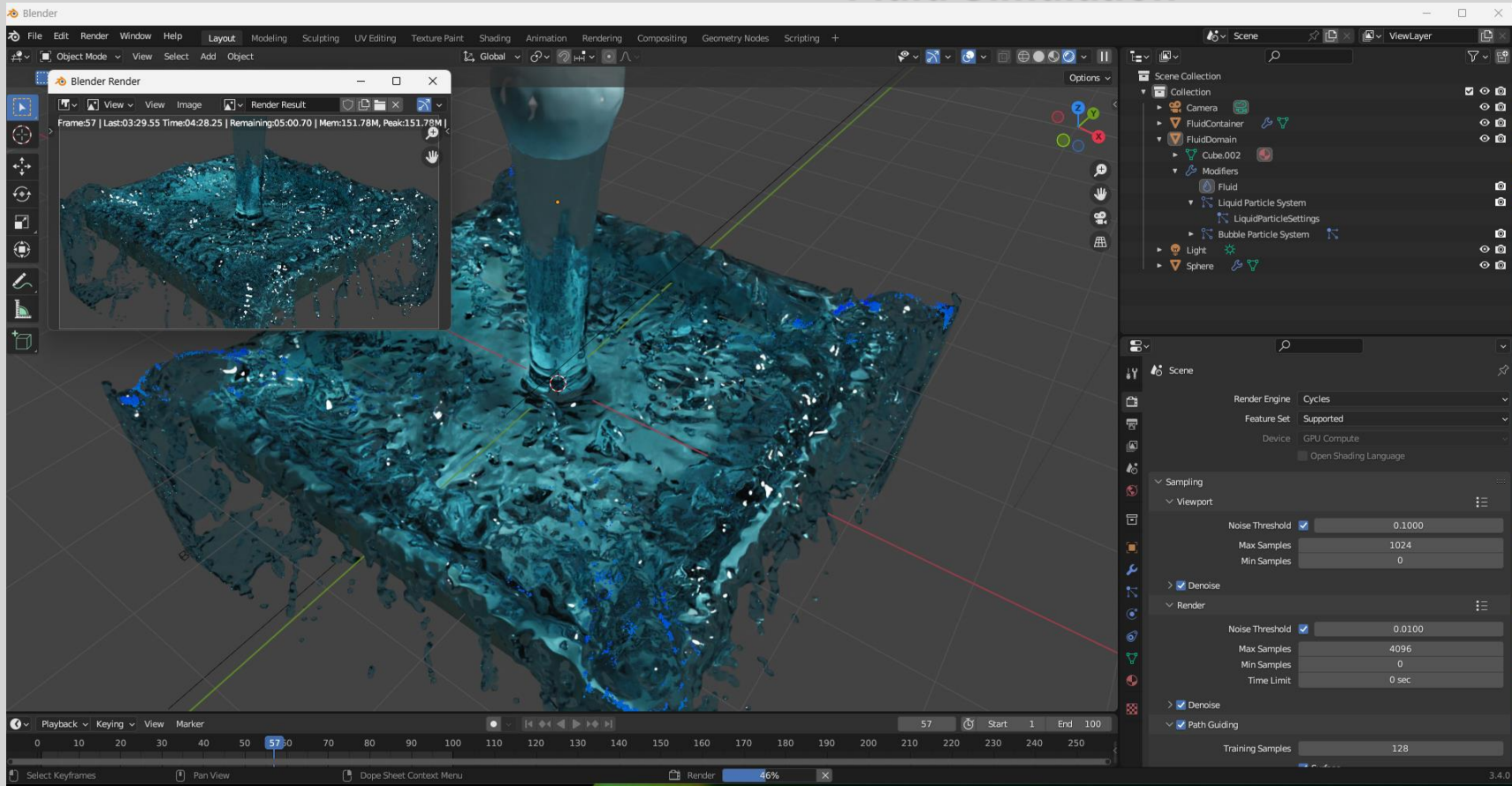




Blender/Python API Fluid Simulation

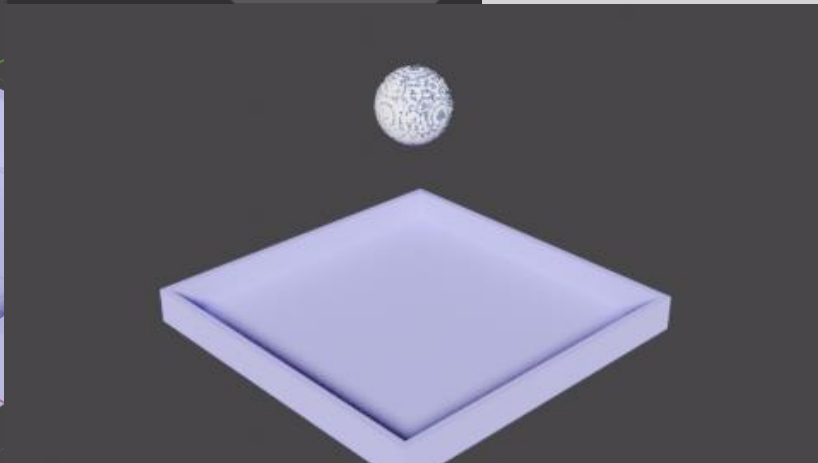
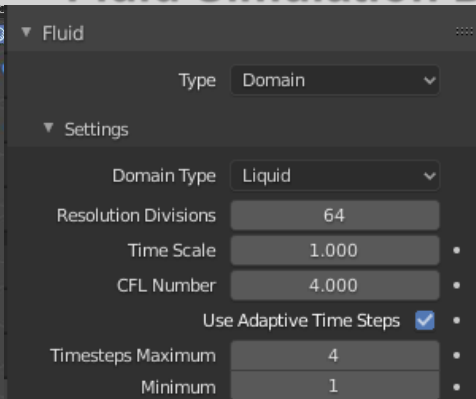
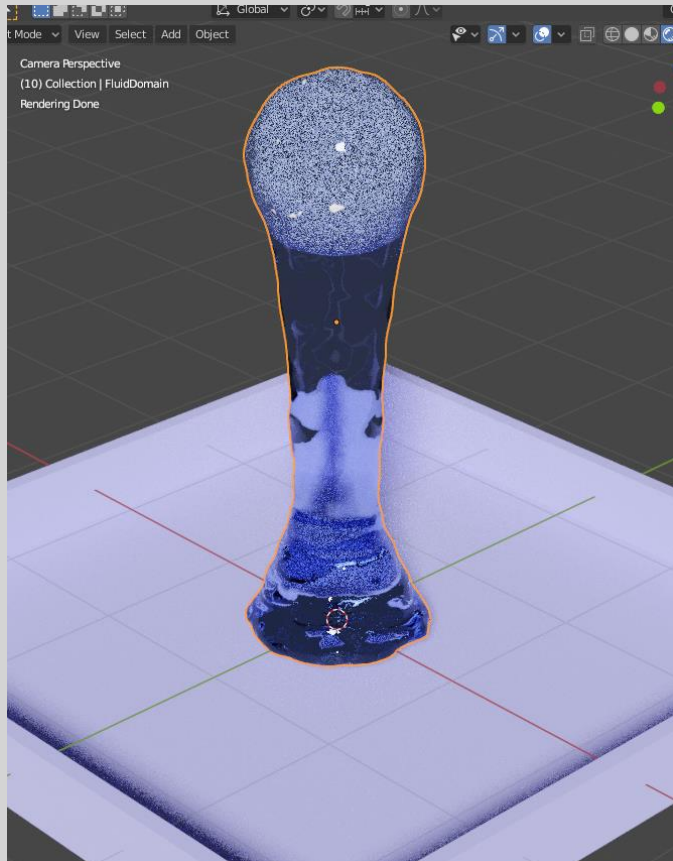


Blender/Python API Fluid Simulation



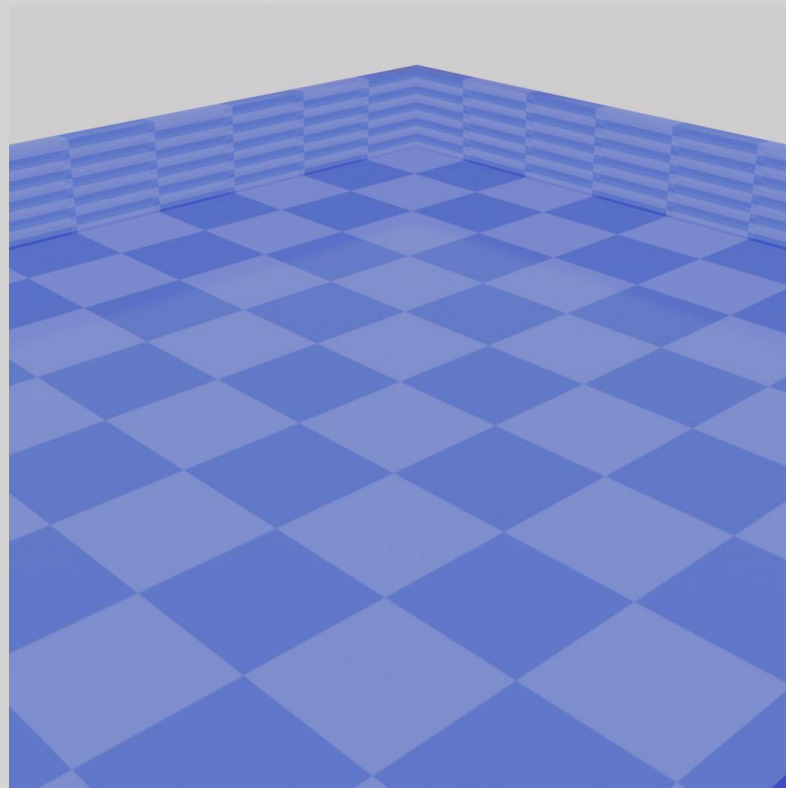
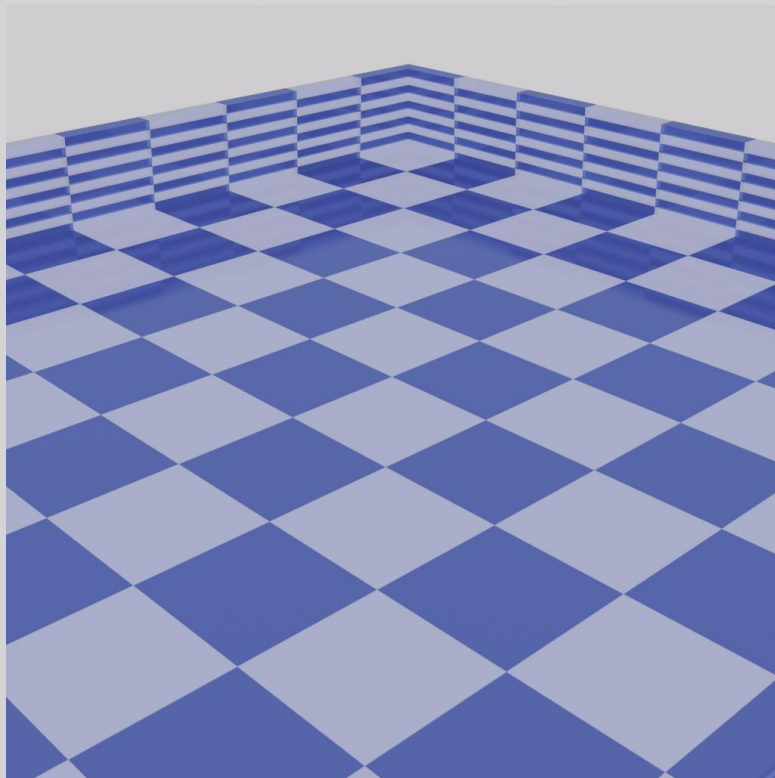


Blender/Python API Fluid Simulation Domain



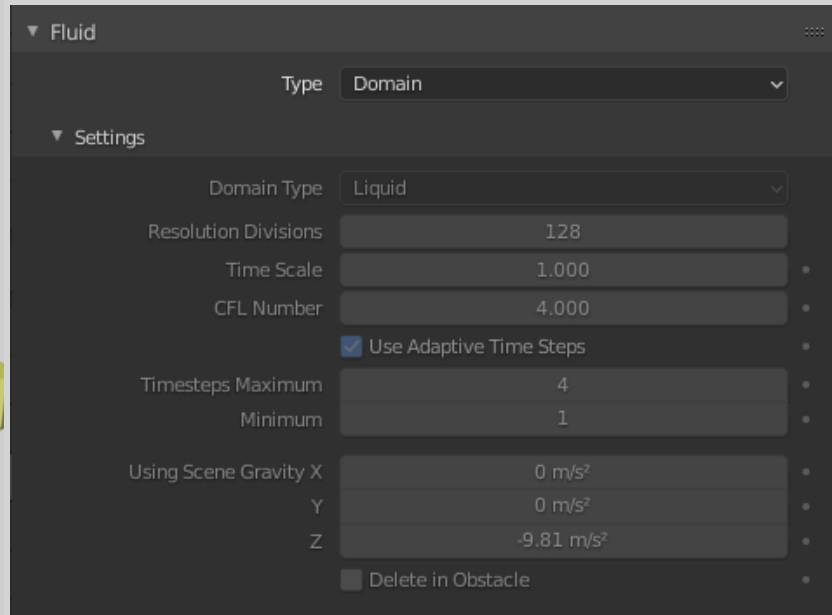
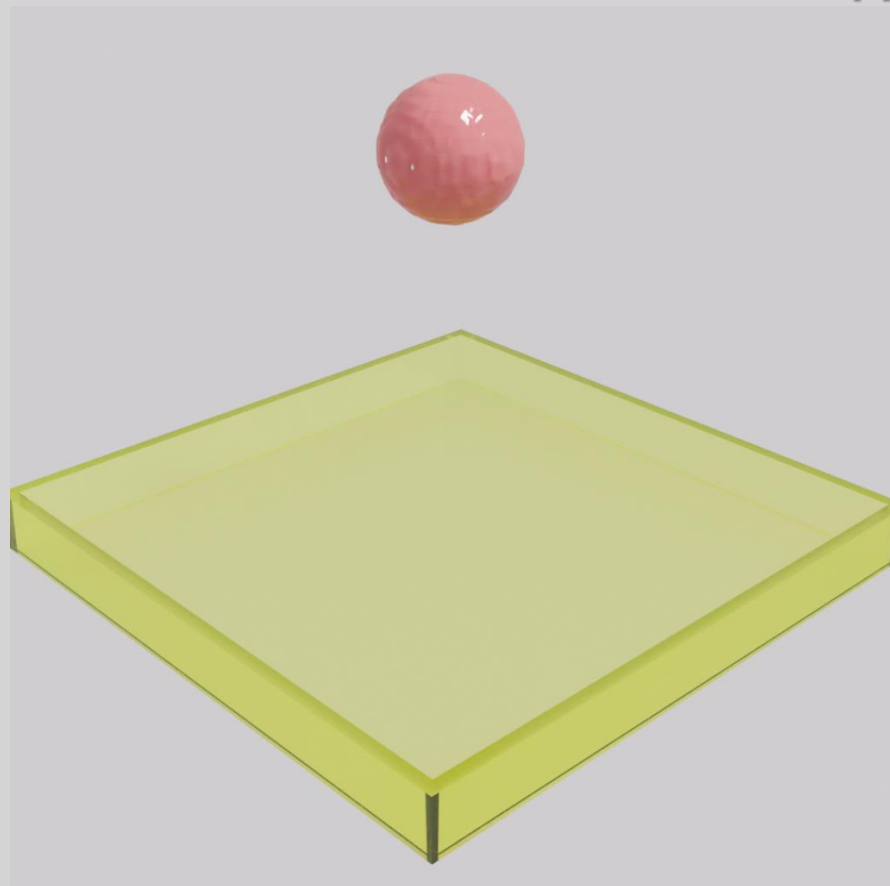


Blender/Python API Fluid Simulation Domain





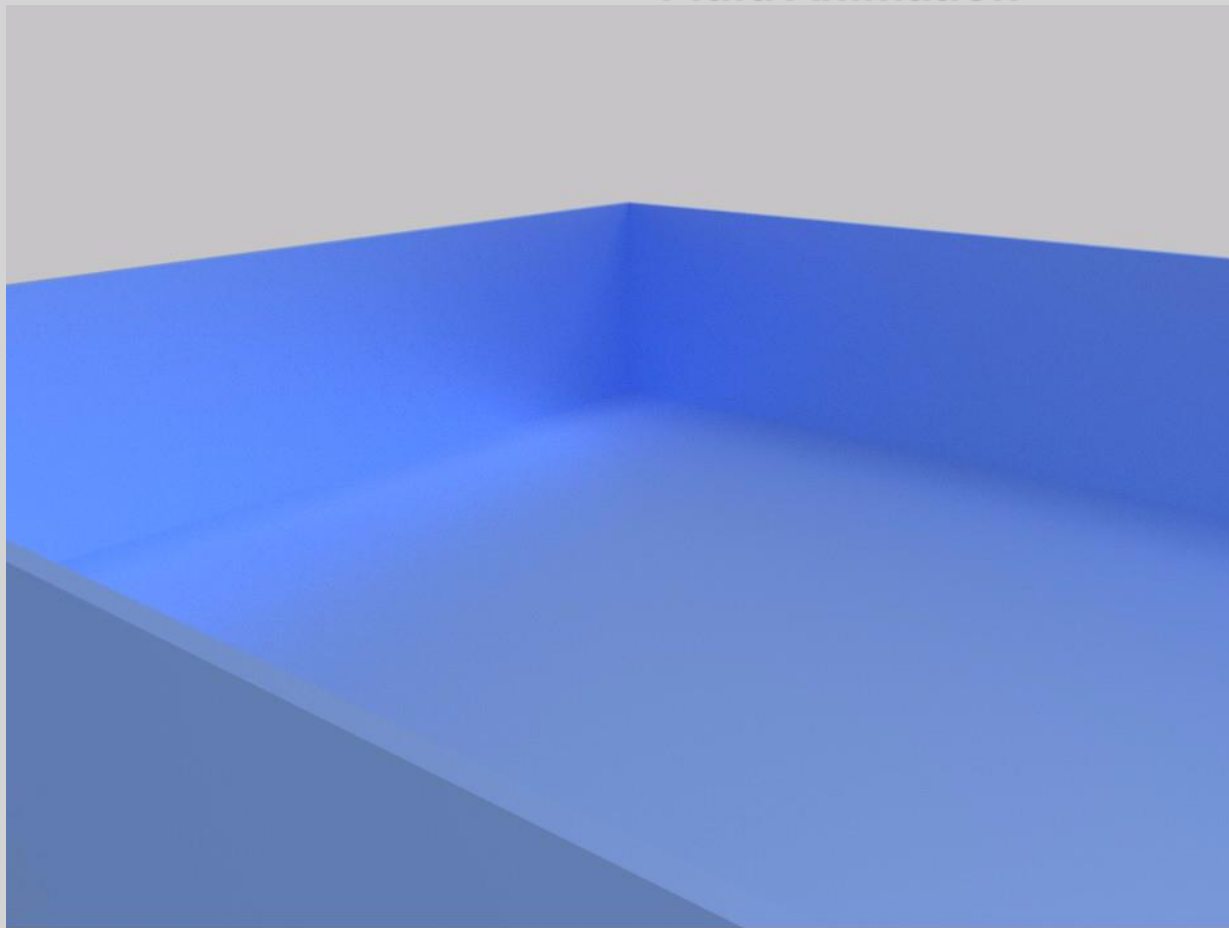
Blender/Python API Fluid Simulation Domain





Blender/Python API

Fluid Animation





Blender/Python API Cloth Simulation

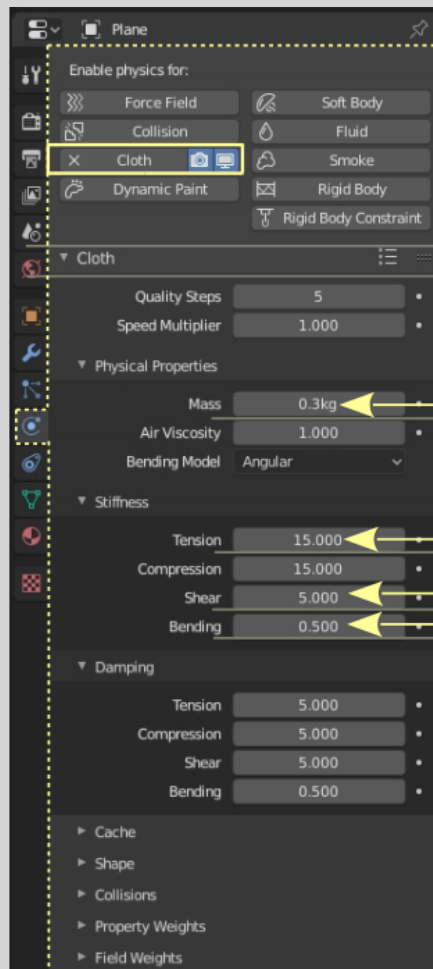
With Cloth Physics applied, an Object exhibits the characteristics of different types of fabric.

With Cloth Physics, controls display in the Properties Editor. Consider a Plane Object in the 3D Viewport Editor in Object Mode, Subdivided ten times (in Edit Mode). With the Plane selected in Object Mode, click Cloth in the Physics buttons.

If you click on the Modifier button in the Properties Editor you will see that a Cloth Modifier has been added to the Plane referring you to the controls in the Physics buttons.



Blender/Python API Cloth Simulation



fabric materials.

Default

	Cotton	Leather	Rubber	Denim	Silk
Mass	0.300	0.4	3.000	1.000	0.150
Tension	15	80	15	40	5
Compression	5	25	25	25	0.0
Shear	0.5	150	25	10	0.05
Bending					



Blender/Python API Cloth Simulation

```
import bpy
```

```
bpy.context.scene.frame_end = 100
```

```
mass = [0.1, 0.5, 0.9]
```

```
for i, sph in enumerate(range(-3, 4, 3)):
```

```
    bpy.ops.mesh.primitive_uv_sphere_add(segments=128, radius=0.5, location=(sph, 0, 1))
```

```
    bpy.ops.object.shade_smooth()
```

```
    bpy.context.active_object.name = 'Sphere_' + str(i)
```

```
    bpy.ops.object.shade_smooth()
```

```
    bpy.ops.object.modifier_add(type='COLLISION')
```



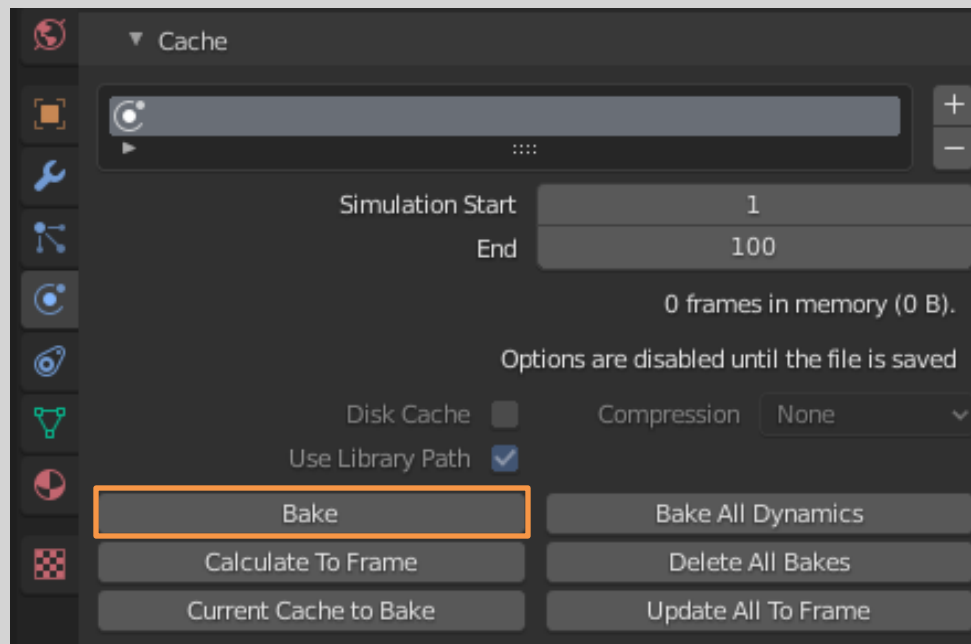
Blender/Python API Cloth Simulation

```
bpy.ops.mesh.primitive_plane_add(size=2.8, location=(sph, 0, 3))
bpy.context.active_object.name = 'myCloth_' + str(i)
# Switching to EDIT mode to subdivide mesh
bpy.ops.object.mode_set(mode = 'EDIT')
bpy.ops.mesh.subdivide(number_cuts=50)
bpy.ops.object.mode_set(mode = 'OBJECT')
bpy.ops.object.shade_smooth()

bpy.ops.object.modifier_add(type='CLOTH')
bpy.context.object.modifiers["Cloth"].settings.quality = 10
bpy.context.object.modifiers["Cloth"].settings.mass = 0.4
bpy.context.object.modifiers["Cloth"].point_cache.frame_end = 100
bpy.context.object.modifiers["Cloth"].settings.mass = mass[i]
bpy.context.object.modifiers["Cloth"].collision_settings.collision_quality = 5
bpy.data.objects['myCloth_' + str(i)].modifiers['Cloth'].collision_settings.use_self_collision = True
```


Blender/Python API

Cloth Simulation





Blender/Python API Cloth Simulation

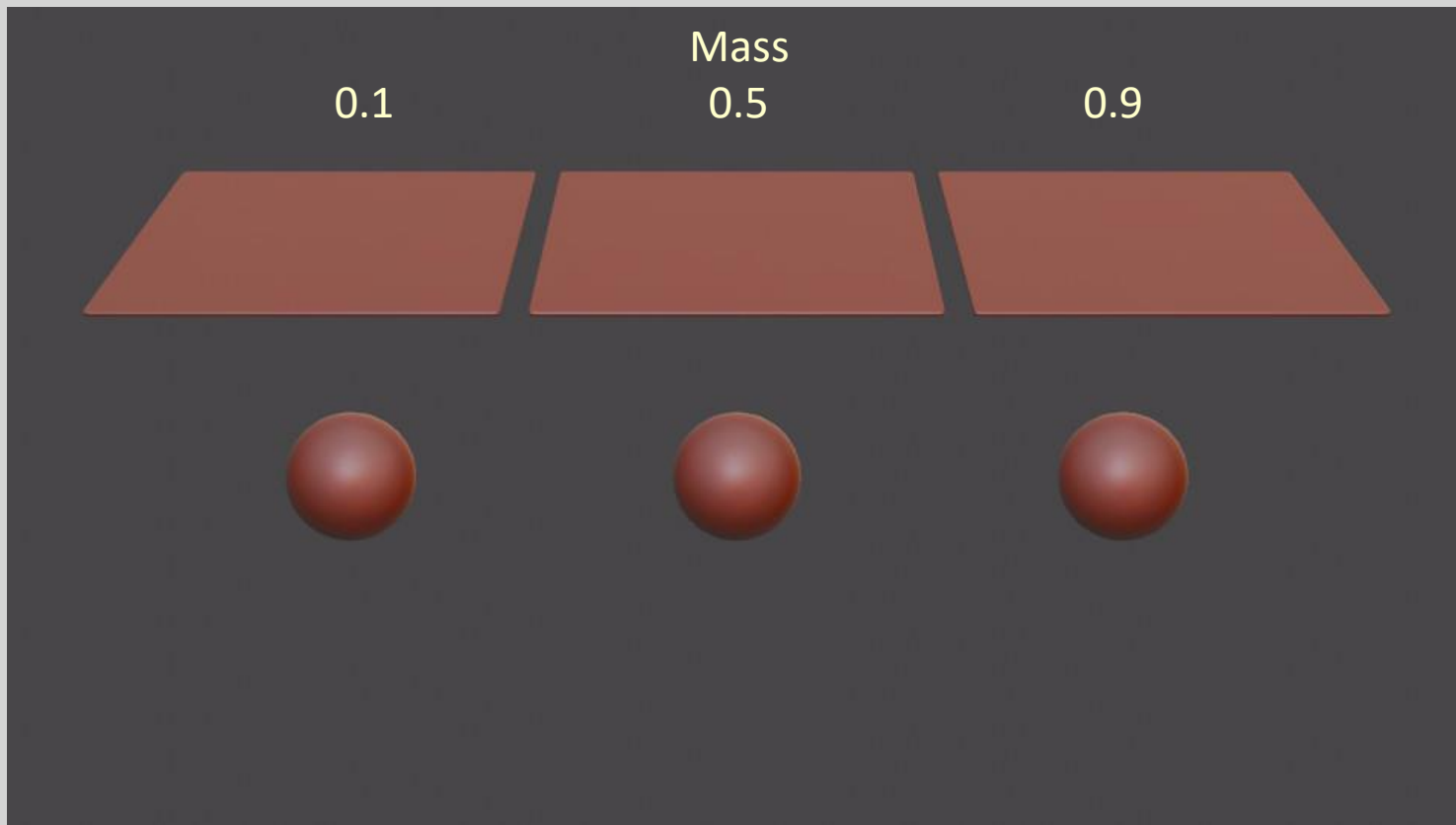
```
# Bake All Physics
#bpy.ops.ptcache.bake_all(bake=True)

for i in range(3):
    ob = bpy.context.scene.objects['myCloth_'+ str(i)]
    bpy.ops.object.select_all(action='DESELECT') # Deselect all objects
    bpy.context.view_layer.objects.active = ob    # Make the cloth the active object
    ob.select_set(True)                          # Select the cloth
    #bpy.data.objects['myCloth_'+ str(i)].select_set(True)
    bpy.ops.object.modifier_add(type='SUBSURF')
    bpy.context.object.modifiers["Subdivision"].render_levels = 3
    bpy.context.object.modifiers["Subdivision"].levels = 3
    bpy.ops.object.modifier_add(type='SOLIDIFY')
    bpy.context.object.modifiers["Solidify"].thickness = 0.03
```



Blender/Python API

Cloth Simulation





Blender/Python API

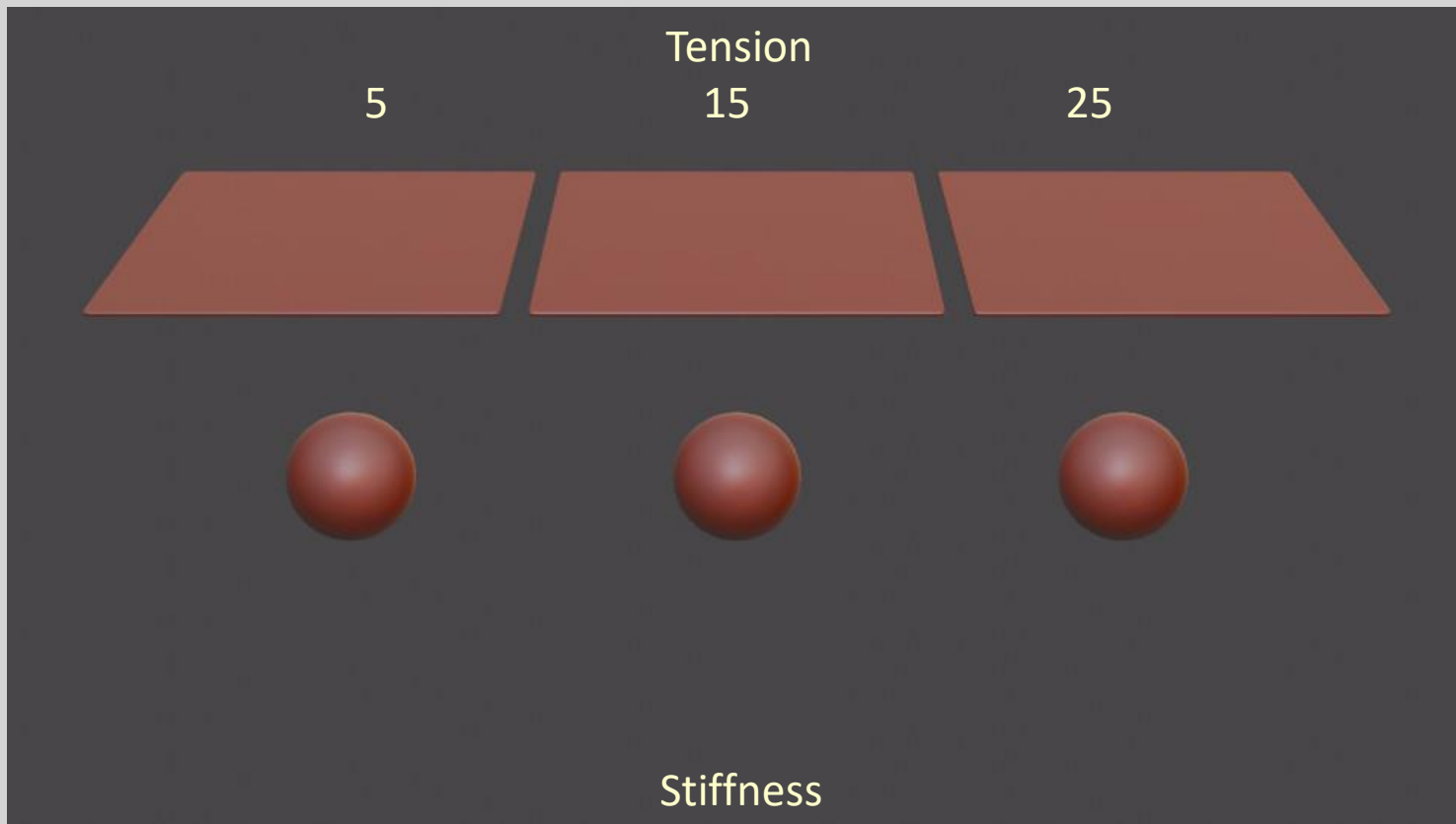
Cloth Simulation





Blender/Python API

Cloth Simulation





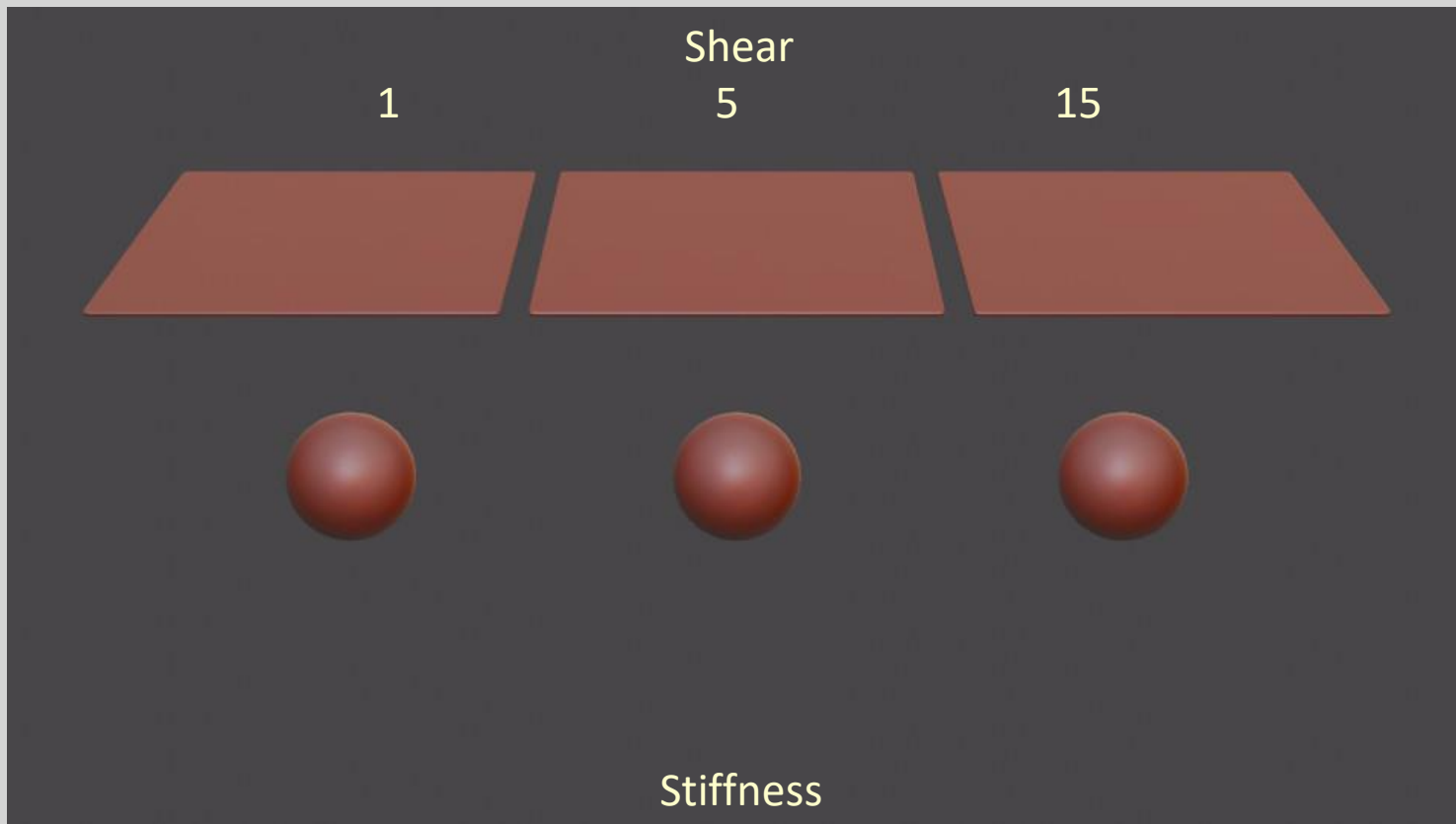
Blender/Python API

Cloth Simulation





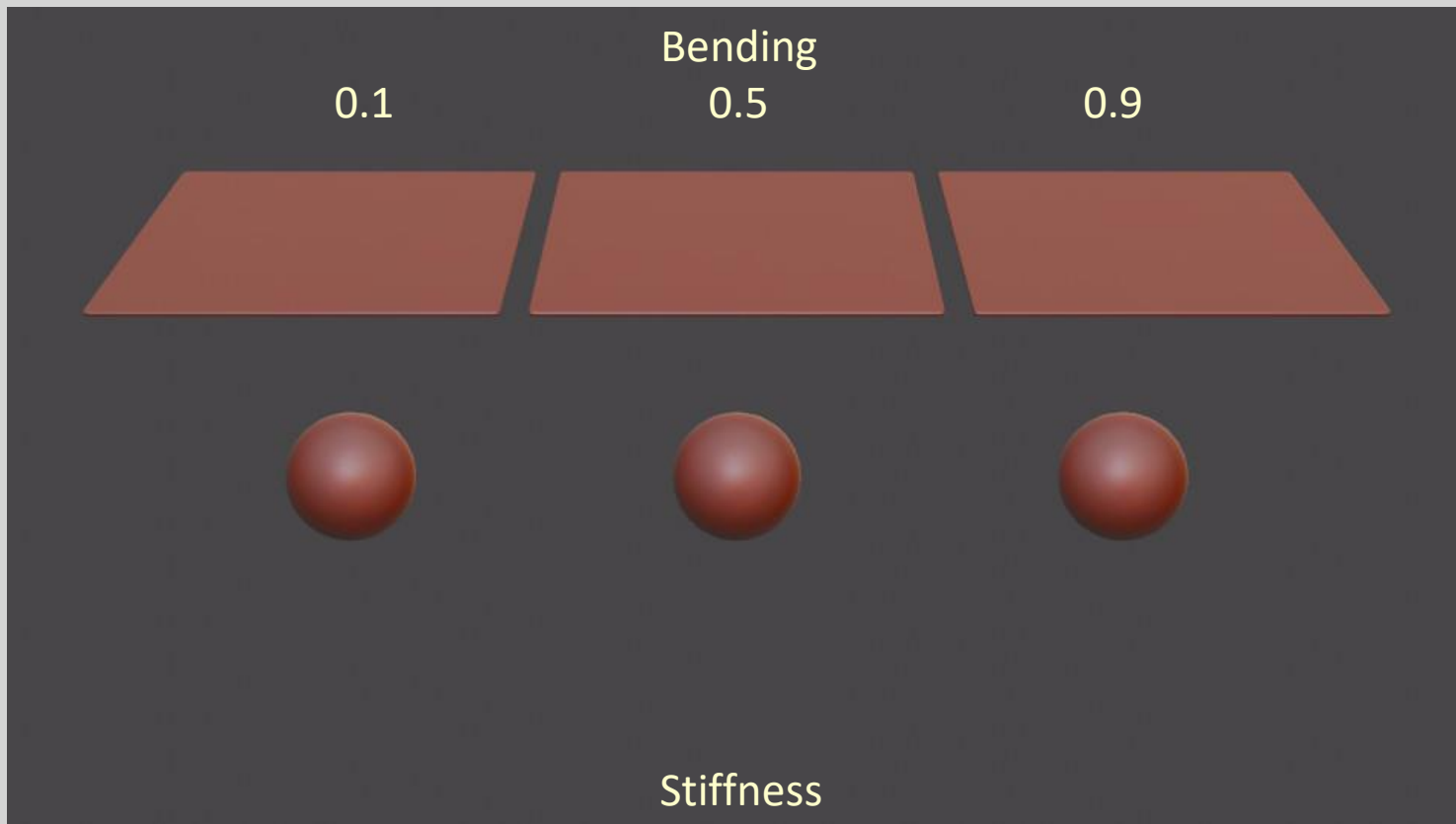
Blender/Python API Cloth Simulation





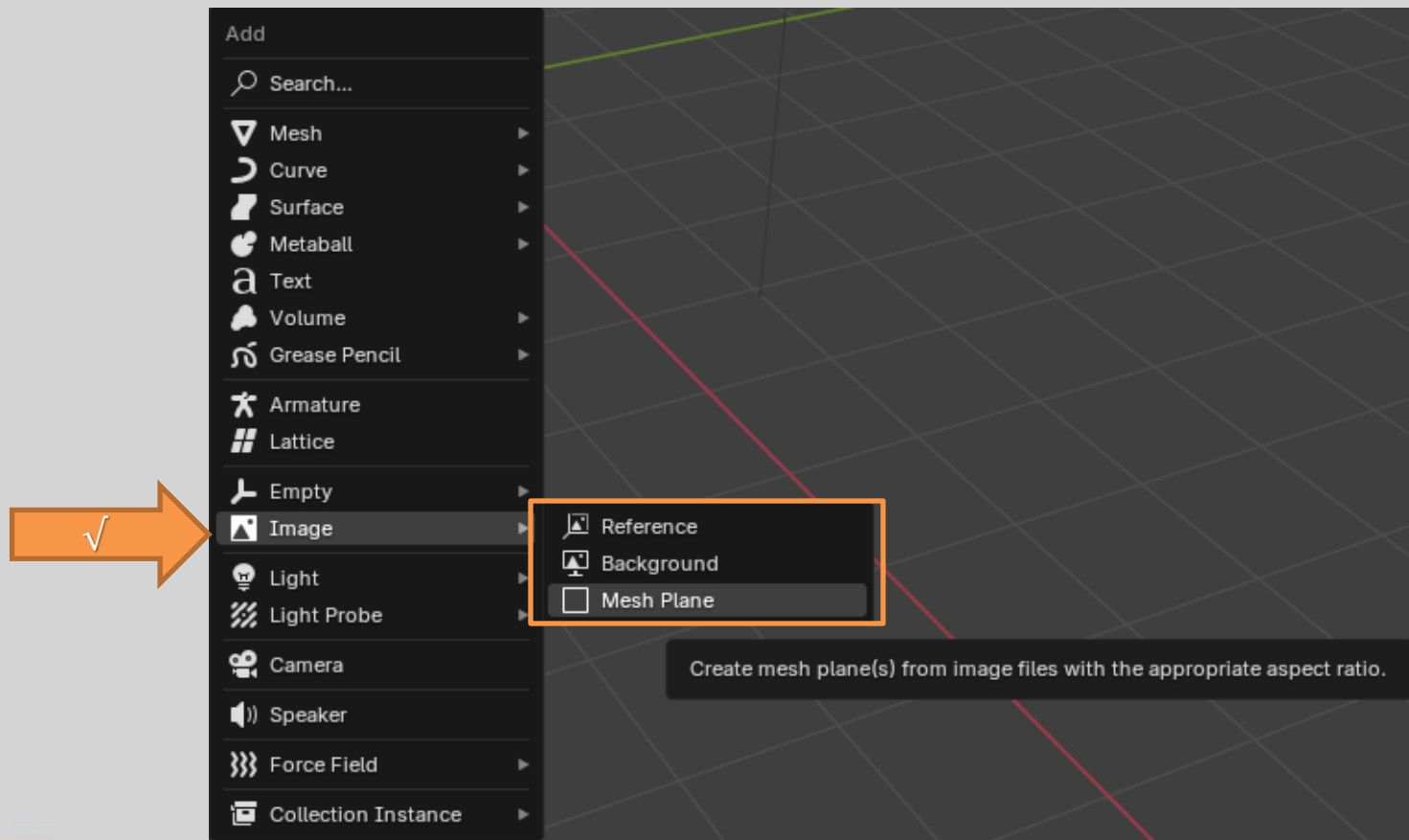
Blender/Python API

Cloth Simulation





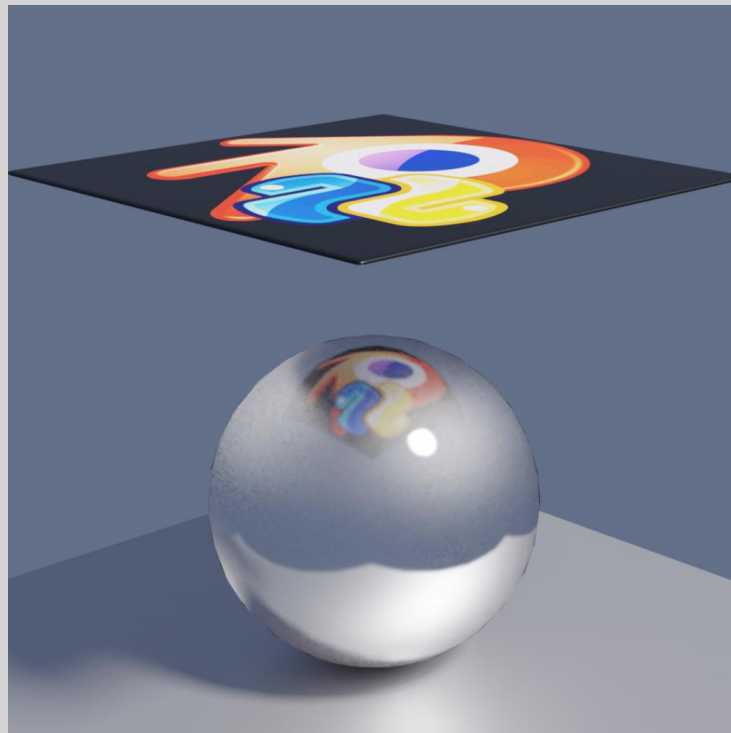
Blender/Python API Cloth Simulation





Blender/Python API Cloth Simulation

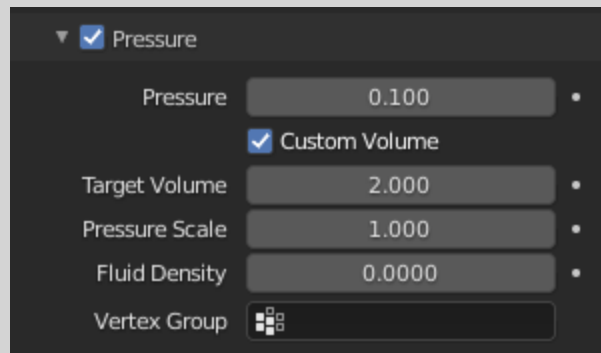
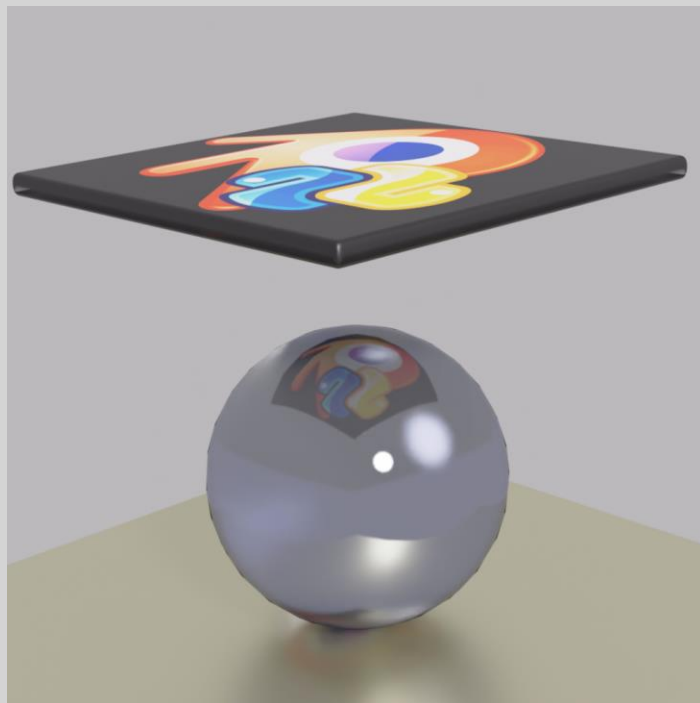
```
bpy.ops.image.import_as_mesh_planes(relative=False, filepath="C:\\... \\Logo_1000.tif",\n                                   files=[{"name": "Logo_1000.tif", "name": "Logo_1000.tif"}],\n                                   directory="C:\\... \\")
```





Blender/Python API Cloth Simulation

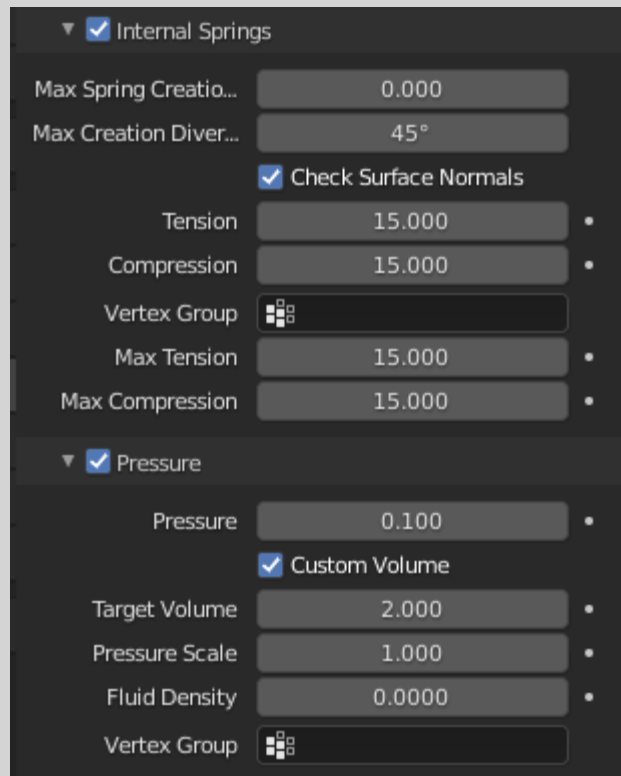
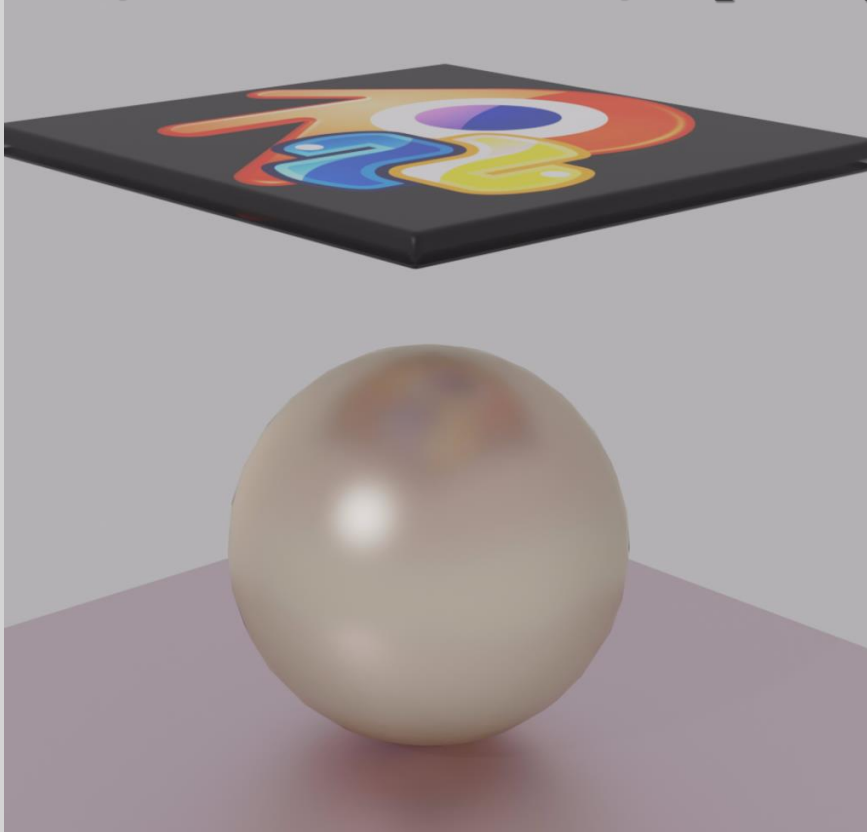
```
bpy.context.object.modifiers["Cloth"].settings.use_pressure = True  
bpy.data.objects['myCloth'].modifiers["Cloth"].settings.uniform_pressure_force = 0.1  
bpy.data.objects['myCloth'].modifiers["Cloth"].settings.use_pressure_volume = True  
bpy.data.objects['myCloth'].modifiers["Cloth"].settings.target_volume = 2
```





Blender/Python API Cloth Simulation

```
bpy.context.object.modifiers["Cloth"].settings.use_internal_springs = True
```





Blender/Python API Cloth Simulation

```
settings.use_internal_springs = False
```



```
settings.use_internal_springs = True
```

