

Blender - Python API

#7



Serdar ARITAN

Department of Computer Graphics
Hacettepe University, Ankara, Turkey



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:

- Hair, grass, and flocks

- Rain

- Smoke and dust

- Water

- Cloth

- Jello

- etc.



Blender/Python API

Blender's Physics

Blender includes advanced physics simulation in the form of the Bullet Physics Engine (Bullet Physics). Most of your work will involve setting the right properties on the objects in your scene, then you can sit back and let the engine take over. The physics simulation can be used for Games, but also for Animation.

When Objects and characters in a Scene move and interact they generally obey the rules of Physics which exist in the real world. Characters jump up and fall down obeying the law of gravity. They collide with each other and with obstacles in the Scene. These actions may or not be exaggerated or strictly adhere to the laws of physics.

Blender includes advanced physics simulation in the form of the Bullet Physics Engine which simulates collision detection, soft and rigid body dynamics. It has been used in video games as well as for visual effects in movies. **Erwin Coumans**, its main author, worked for Sony Computer Entertainment US R&D from 2003 until 2010, for AMD until 2014, then Google and he now works for **NVIDIA**.



(L-R) Nafees Bin Zafar, Stephen Marshall and **Erwin Coumans** attend the Academy Of Motion Picture Arts And Sciences' Scientific And Technical Awards Ceremony at the Beverly Wilshire Four Seasons Hotel on February 7, **2015** in Beverly Hills, California.

Academy Of Motion Picture Arts And Sciences'
Scientific And Technical Awards Ceremony

Blender/Python API Bullet Physics Library



Physics Simulation Team Lead

Sony Computer Entertainment America

May 2003 - Ara 2010 · 7 yıl 8 ay

Build and lead a team to support game physics technology for the Playstation 3 platform. Visit Playstation 3 developers to encourage Cell SPU usage. Present on many conferences on Cell SPU optimizations and physics simulation. Help shipping many AAA games using the Bullet Physics Engine. Help many professional game developers integrate physics technology into their games that exploit Cell SPU effectively.

Also helped optimizing NVIDIA PhysX for Playstation 3 within Playstation US Research...

Daha fazla göster ▾



Senior Physics SDK Engineer

Havok / Telekinesys

Haz 2001 - May 2003 · 2 yıl



Senior Developer

Not a Number / Blender

Oca 2000 - Nis 2001 · 1 yıl 4 ay

Game Engine Architect + R&D. As one of the first company hires, build a game engine team, help hiring and interviewing candidates. Work with the CTO towards a roadmap for the game engine, and execute the plan with the team. Design no-coding logic bricks to allow game developers to create games without traditional coding. Gather feedback from the Blender community and host game engine contests to encourage user generated content (UGC).



Software Developer

Guerilla Games (formerly Orange Games/Epic Games/Lostboys)

1998 - 2000 · 2 yıl

Developed Real-Time CSG (constructive Solid Geometry), based on incrementally updating a CSG operation tree, caching triangles in all internal nodes. Created an in-house level editor using MFC, using this CSG technology, compatible with the Unreal Engine file format.



Distinguished Engineer, Omniverse & Simulation Technology

NVIDIA

Mar 2022 - Halen · 9 ay

<https://www.nvidia.com/en-us/omniverse>



Google Brain Team Researcher

Google

Mar 2014 - Mar 2022 · 8 yıl 1 ay

Founded the Quadruped Locomotion research team in Brain Robotics and responsible for the Physics Simulation of the Everyday Robot project (<https://everydayrobots.com>) and numerous other Robotic Learning projects. Hosted several PhD internships and student researcher positions, and mentored new team members. Presented our research at leading robotics conferences, universities and research labs. Lead the PyBullet effort used for Deep Reinforcement Learning and simulation-to-real robots behavioral...

Daha fazla göster ▾



Creator and Lead Architect

Bullet Physics

Haz 2003 - Mar 2022 · 18 yıl 10 ay

Created the open source Bullet physics library and related activities (community forum, website, software repository, presentations). Encourage its use in film studios such as Dreamworks Animation, Sony Pictures, Digital Domain and game studios such as Rockstar, Disney Interactive and Sony Computer Entertainment and other areas.



Principal Member Of Technical Staff

Advanced Micro Devices

Kas 2010 - Mar 2014 · 3 yıl 5 ay

Leading Bullet physics effects development within the Office of the CTO. Created an OpenCL rigid body pipeline running entirely on GPU. Got Bullet into a Siemens simulation software package and into Autodesk Maya. Implemented higher quality Mixed Linear Complementarity Problem (MLPC) solvers and Featherstone Articulated Body Algorithm. Co-author of the book Multithreading in Visual Effects.



Blender/Python API Bullet Physics Library



Erwin Coumans 🇺🇦 @erwincoumans · Nov 8

PhysX 5 is open source now!

sigmoid.social/web/@Pierre@ma...

or direct github link:

NVIDIA-Omniverse/ PhysX



NVIDIA PhysX SDK

5

Contributors

1

Issue

10

Discussions

1k

Stars

96

Forks



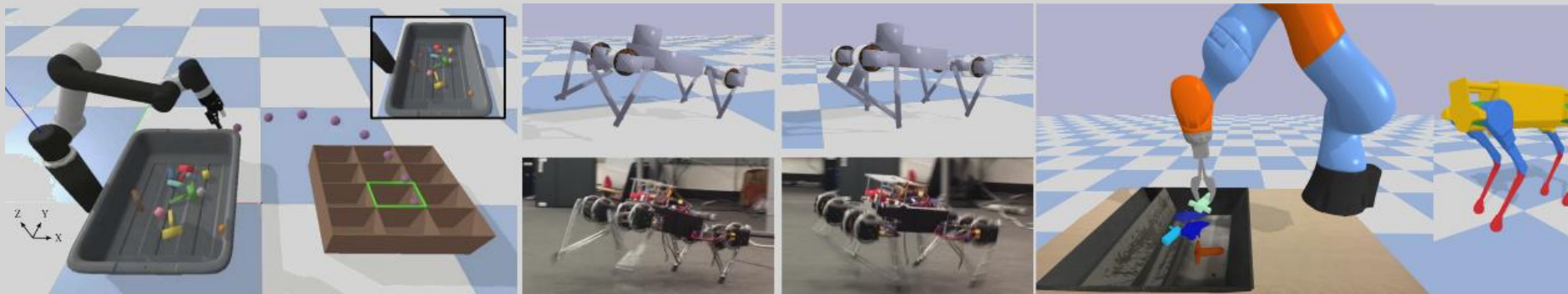
github.com

GitHub - NVIDIA-Omniverse/PhysX: NVIDIA PhysX SDK

NVIDIA PhysX SDK. Contribute to NVIDIA-Omniverse/PhysX development by creating an account on GitHub.

Blender/Python API

Blender's Physics



<https://pybullet.org/wordpress/>

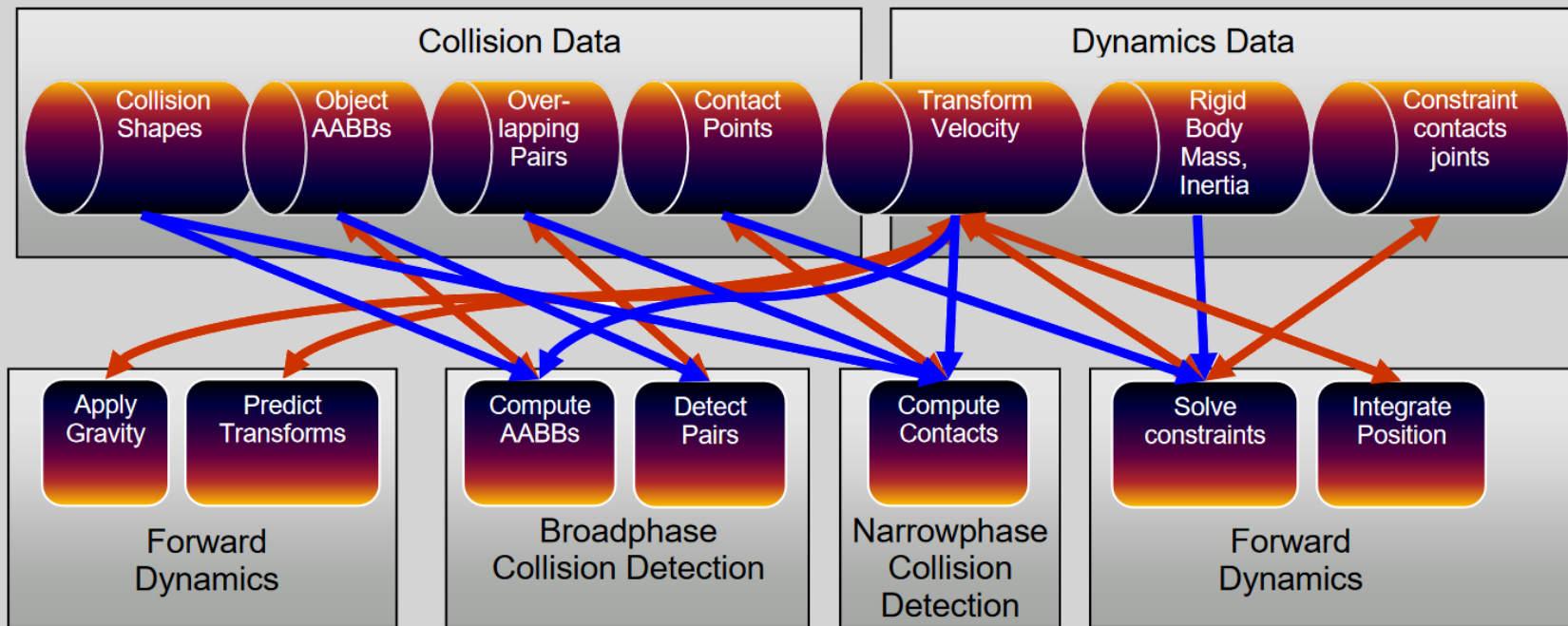
<https://github.com/bulletphysics/bullet3/releases>

Tagged a github release of Bullet Physics and PyBullet, both version 3.05. The release was used for our motion imitation research, and also includes various improvements for the finite-element-method (FEM) deformable simulation. <http://pybullet.org>

Blender/Python API

Rigid Body Physics Pipeline

Data structures and computation stages in the Bullet physics pipeline. This pipeline is executed from left to right, starting by applying gravity, and ending by position integration, updating the world transform.





Blender/Python API pybullet

`pybullet` is an easy to use Python module for physics simulation for robotics, games, visual effects and machine learning. With `pybullet` you can load articulated bodies from **Unified Robot Description Format (URDF)**, **Simulation Description Format (SDF)**, **Multi-Joint dynamics with Contact (MuJoCo - MJCF)** and other file formats. `pybullet` provides forward dynamics simulation, inverse dynamics computation, forward and inverse kinematics, collision detection and ray intersection queries. The Bullet Physics SDK includes `pybullet` robotic examples such as a simulated Minitaur quadruped, humanoids running using `TensorFlow` inference and KUKA arms grasping objects.

<https://youtu.be/tfqCHDoFHRQ>



Blender/Python API

Blender's Physics

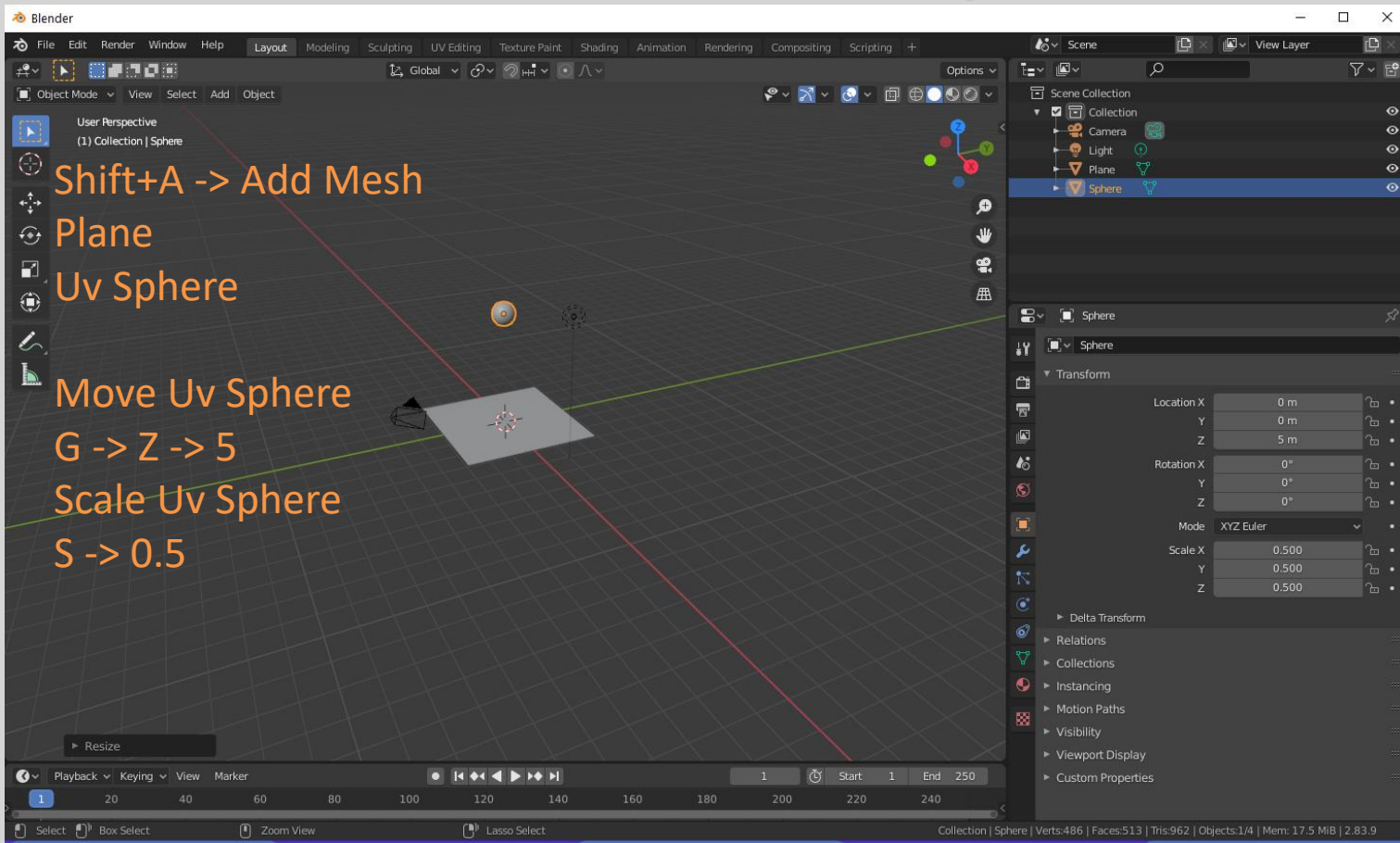
The **rigid body** simulation can be used to simulate the motion of **solid objects**. It affects the position and orientation of objects and does not deform them. Unlike the other simulations in Blender, the rigid body simulation works closer with the animation system. This means that rigid bodies can be used like regular objects and be part of parent-child relationships, animation constraints and drivers.

There are two types of rigid body: active and passive. Active bodies are dynamically simulated, while passive bodies remain static. Both types can be driven by the animation system when using the Animated option.



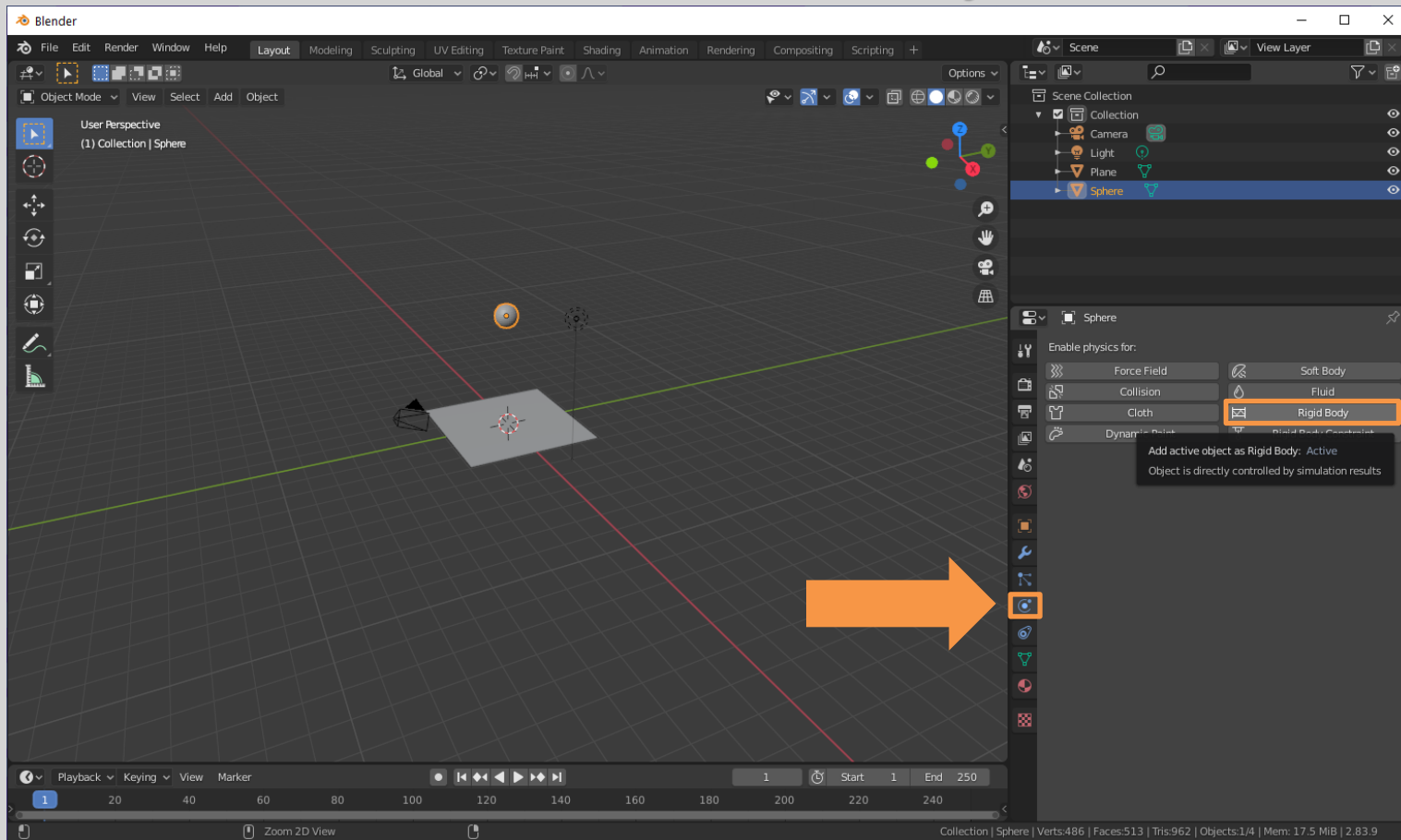
Blender/Python API

Blender's Physics



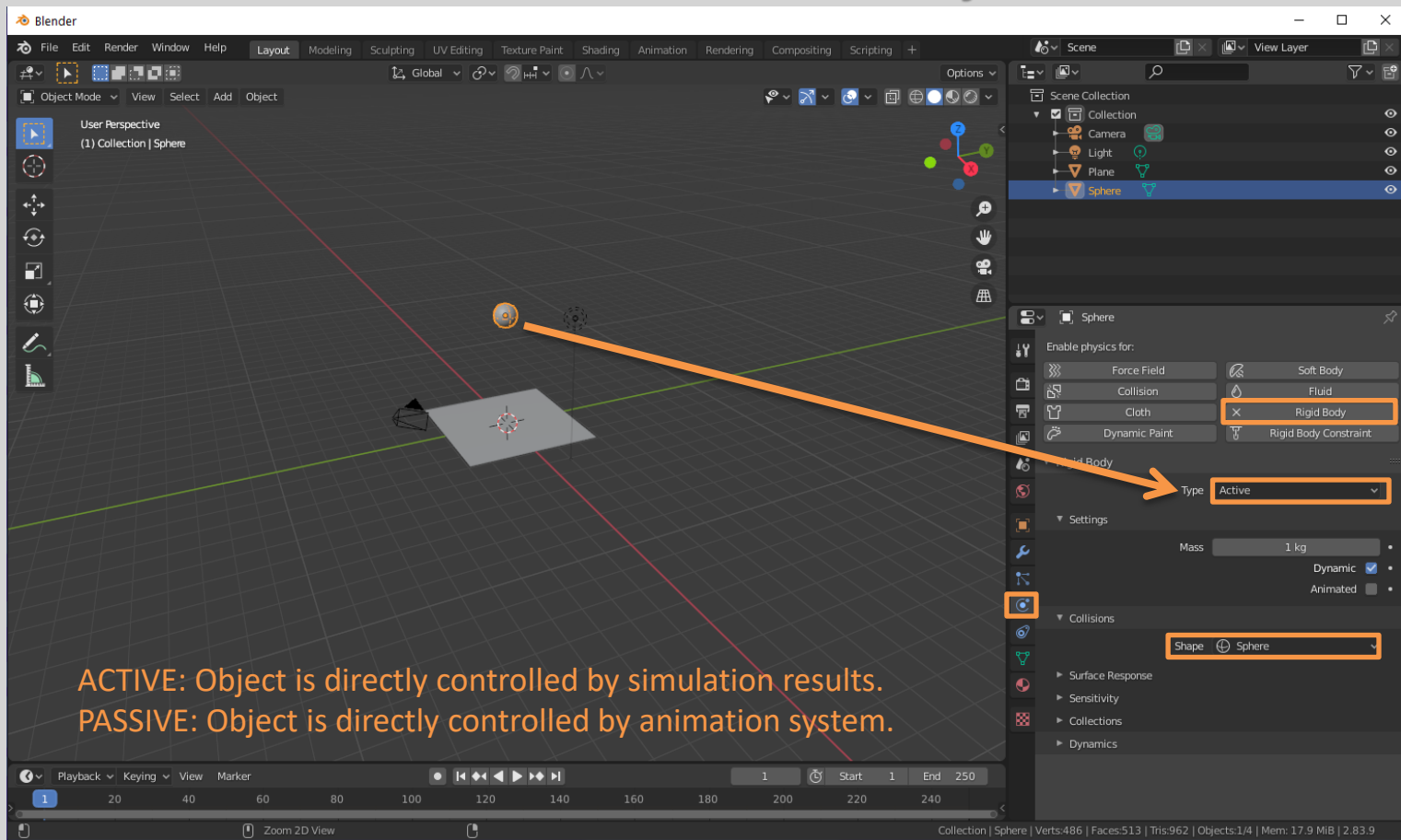
Blender/Python API

Blender's Physics



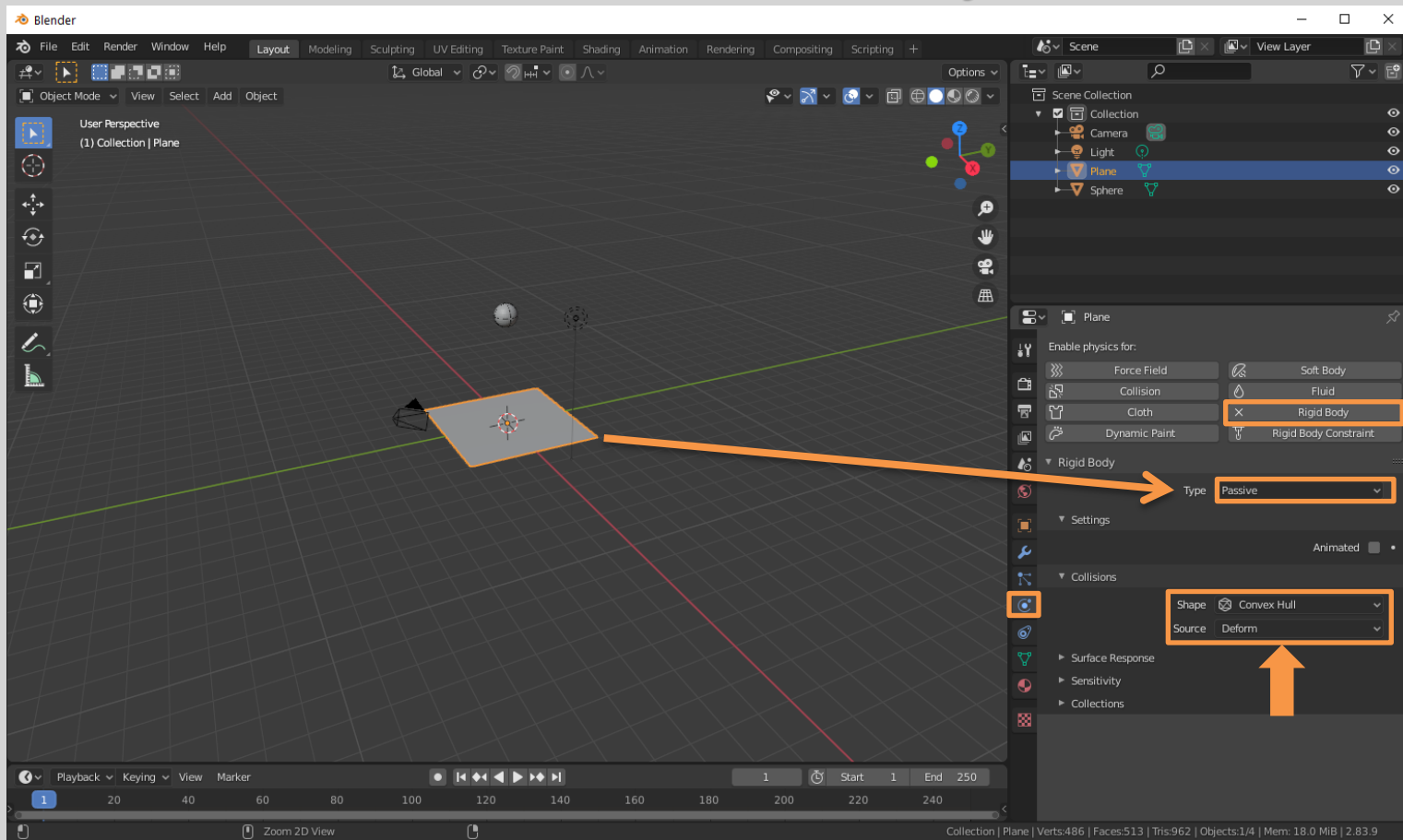
Blender/Python API

Blender's Physics



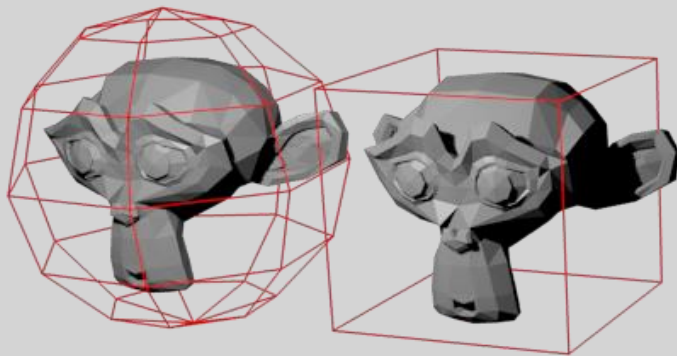
Blender/Python API

Blender's Physics





Blender/Python API Collision Shapes

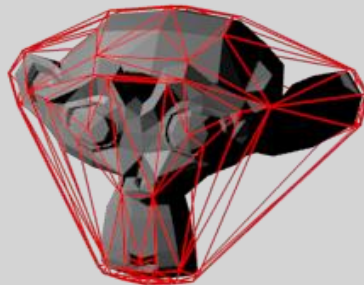


SPHERE

A very fast and simple shape

BOX

A cuboid shape, the length in each dimension can be chosen arbitrarily.



CONVEX_HULL

This is the fastest kind of arbitrary shape. It is defined by a cloud of vertices but the shape formed is the smallest convex shape that encloses the vertices

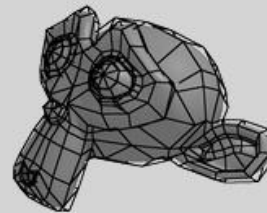
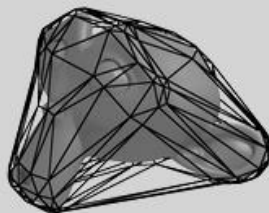
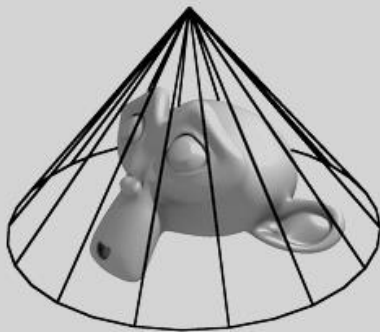
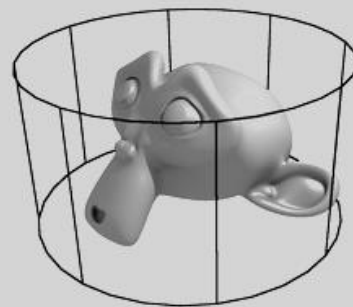
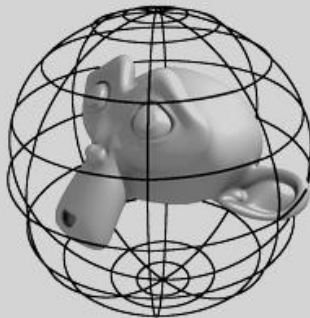
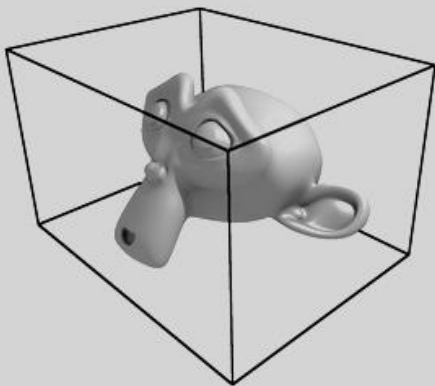


MESH

This is similar to the above but is specified using a conventional mesh composed of vertices and indices. A ConvexHull is more efficient.



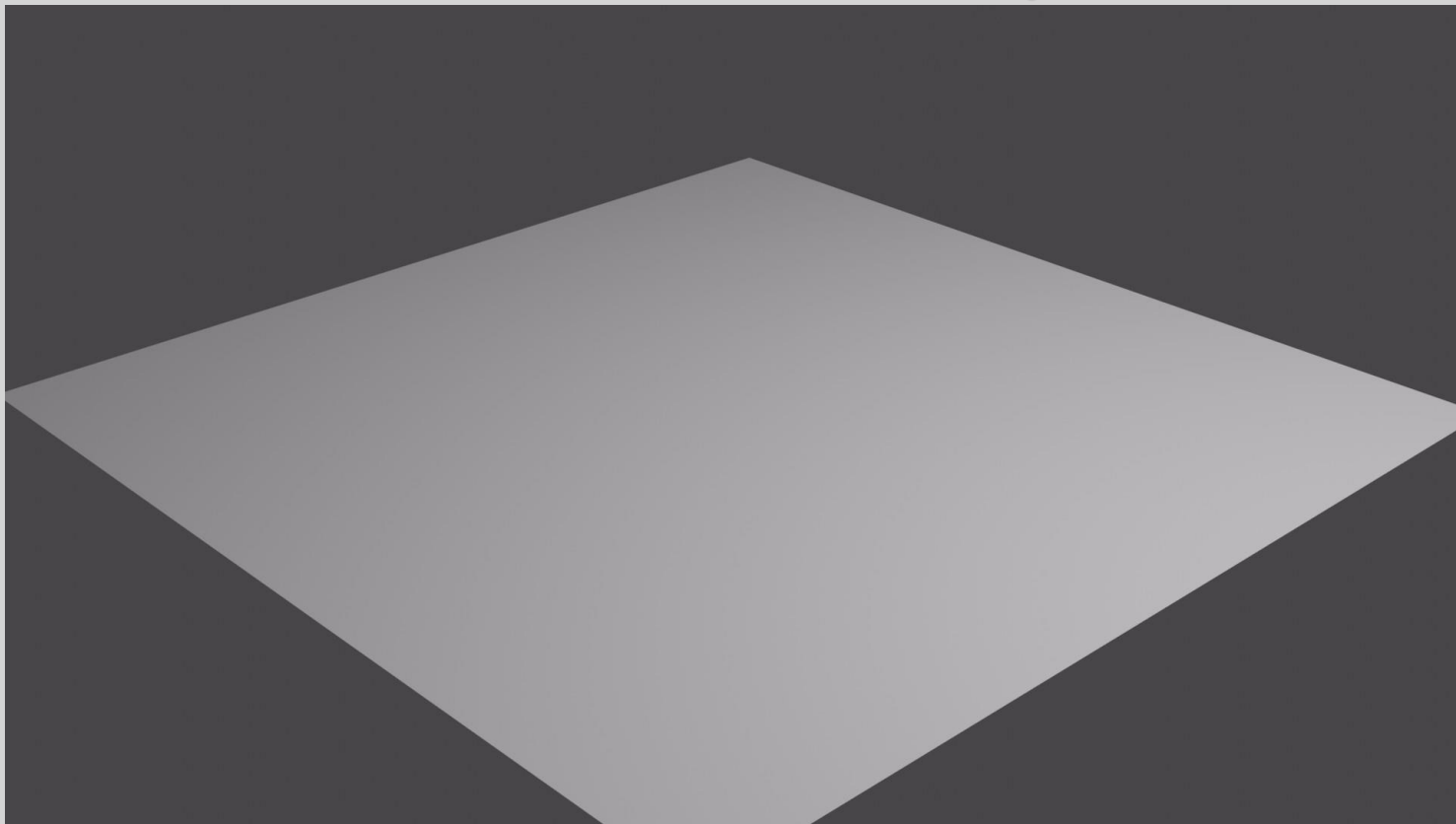
Blender/Python API Collision Shapes





Blender/Python API

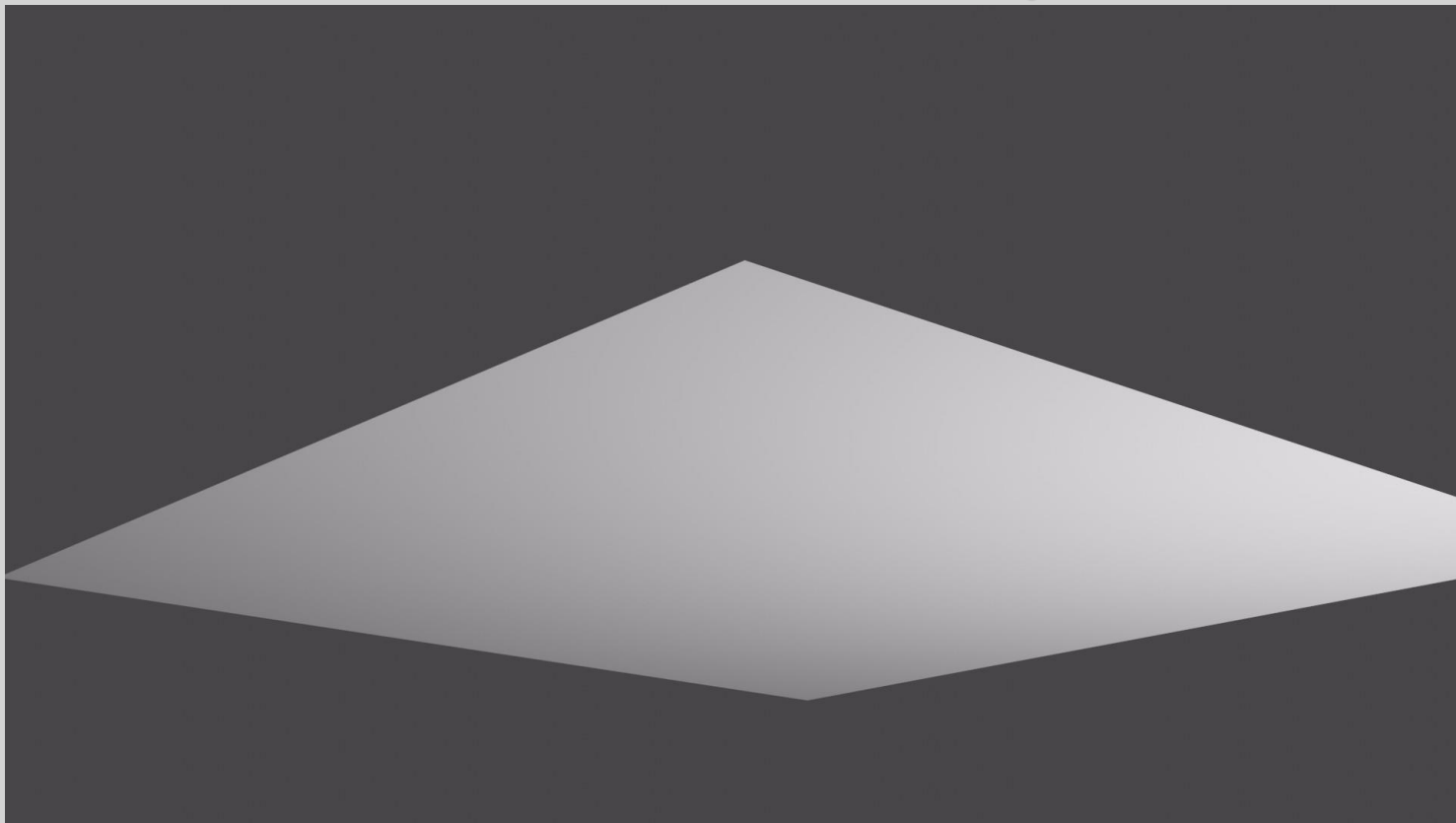
Blender's Physics





Blender/Python API

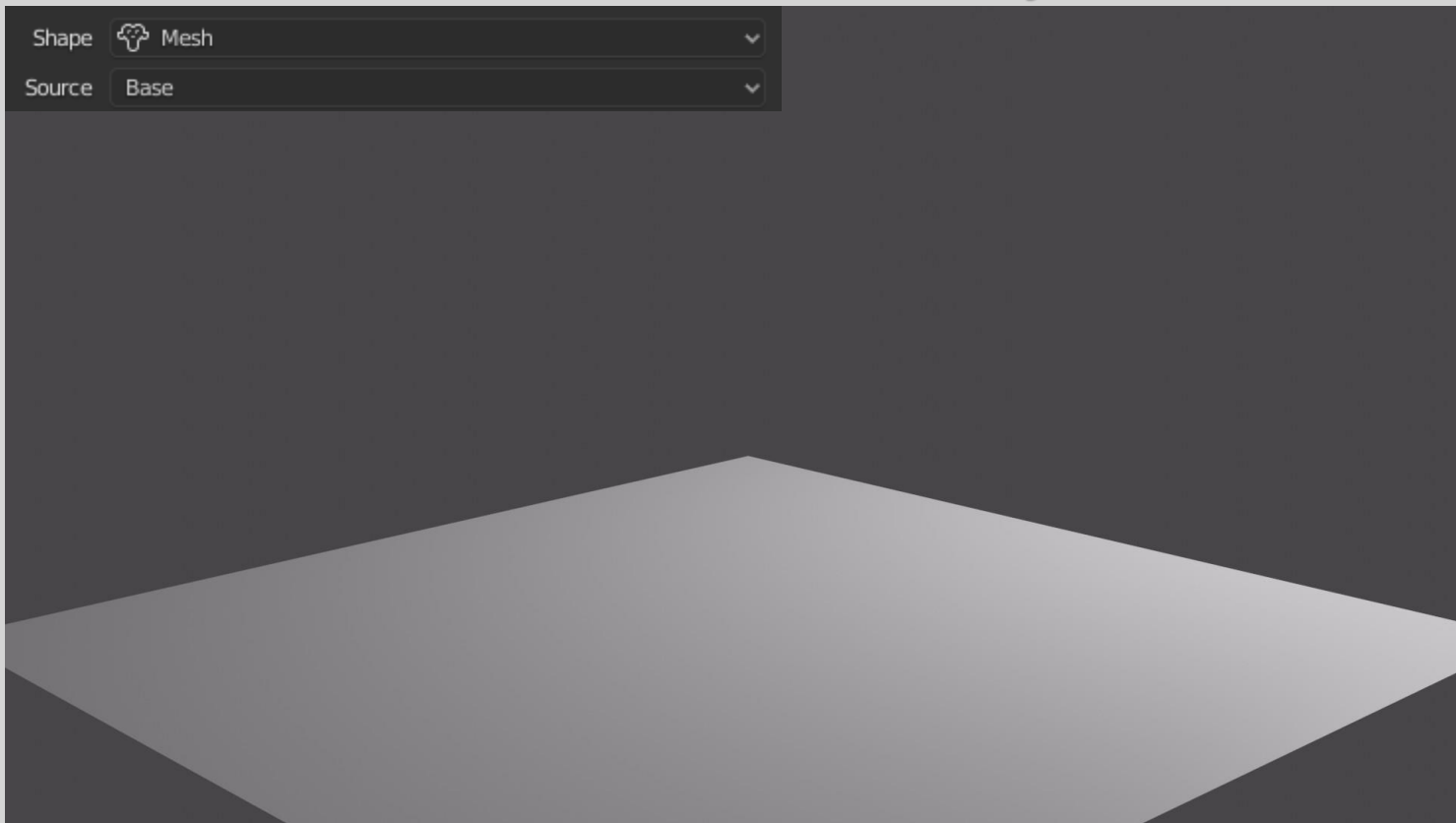
Blender's Physics





Blender/Python API


Blender's Physics





Blender/Python API

Blender's Physics

Shape  Mesh

Source Base

Use custom collision margin (some shapes will have a visible gap around them).

▼ Sensitivity

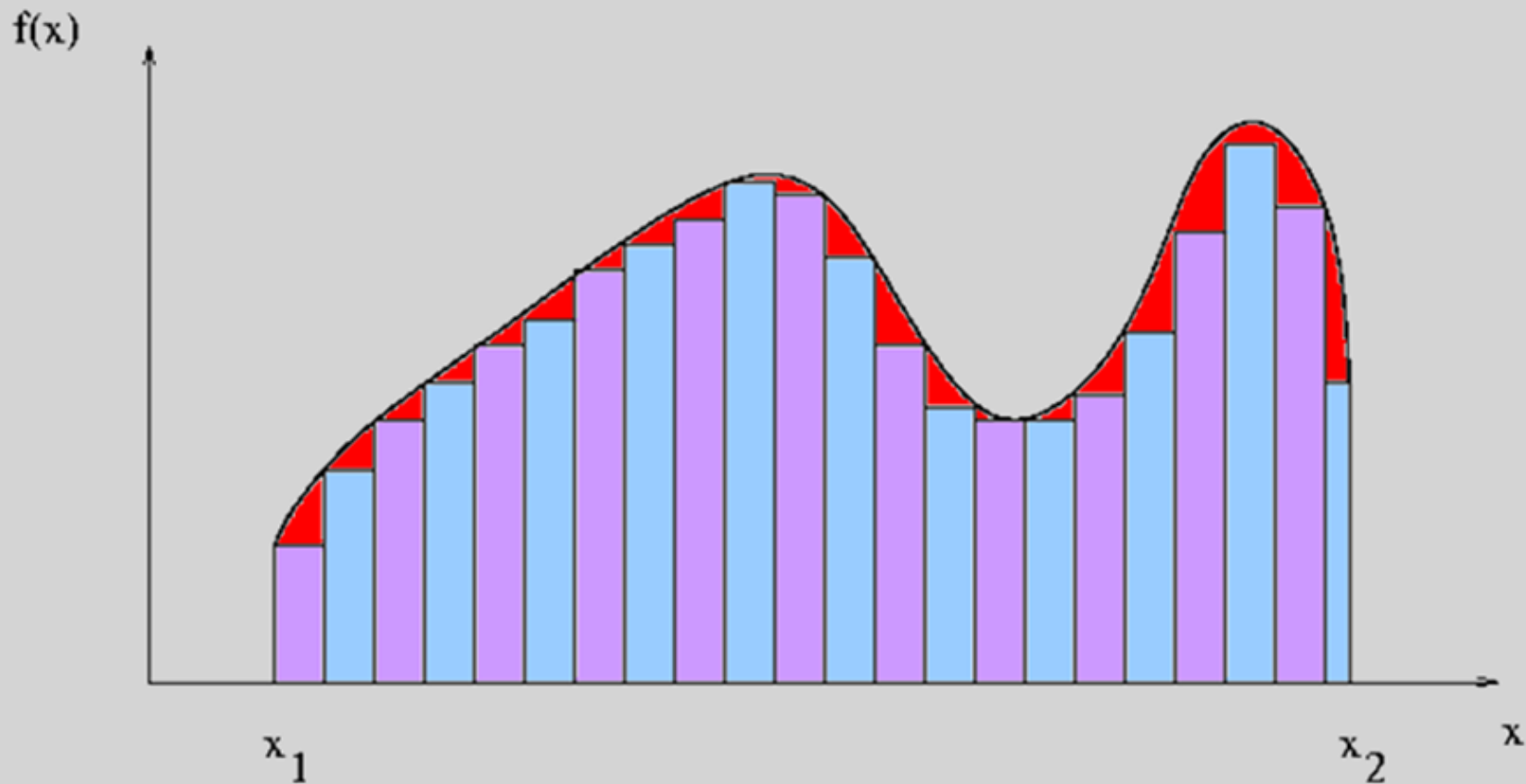
Collision Margin

Margin

0.04 m



Blender/Python API Numerical Integration





Blender/Python API Numerical Integration

Euler : Euler's method is first order method. It is a straight-forward method that estimates the next point based on the rate of change at the current point. It is a single step method. If no dampening is used, particles get more and more energy over time. “**For example, bouncing particles will bounce higher and higher each time**”. Use this integrator for short simulations or simulations with a lot of dampening where speedy calculations are more important than accuracy.

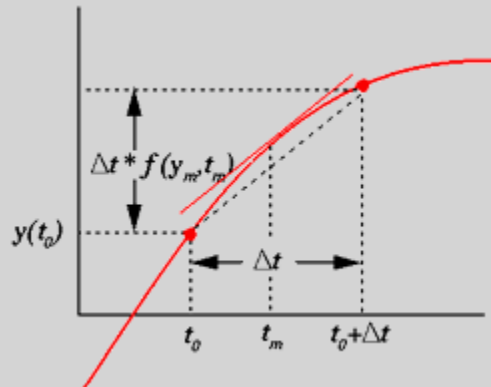
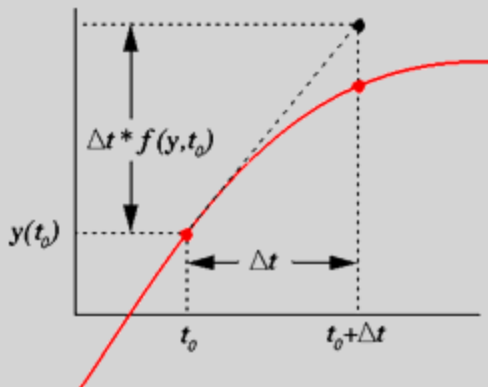
Midpoint : Also known as “**2nd order Runge-Kutta**”. Slower than Euler but much more stable.

RK4 : Short for “**4th order Runge-Kutta**”. Similar to Midpoint but slower and in most cases more accurate. It is energy conservative even if the acceleration is not constant.



Blender/Python API Numerical Integration

Euler method: Also known as “**Forward Euler**”. Simplest integrator. Very fast but also with less exact results. If no damping is used, particles get more and more energy over time. Notably, Forward Euler's method is unconditionally unstable for un-damped oscillating systems (such as a spring-mass system or wave equations) in space discretization.





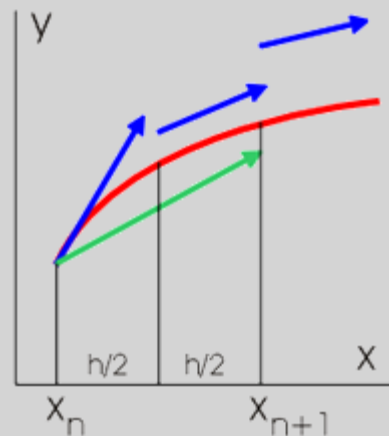
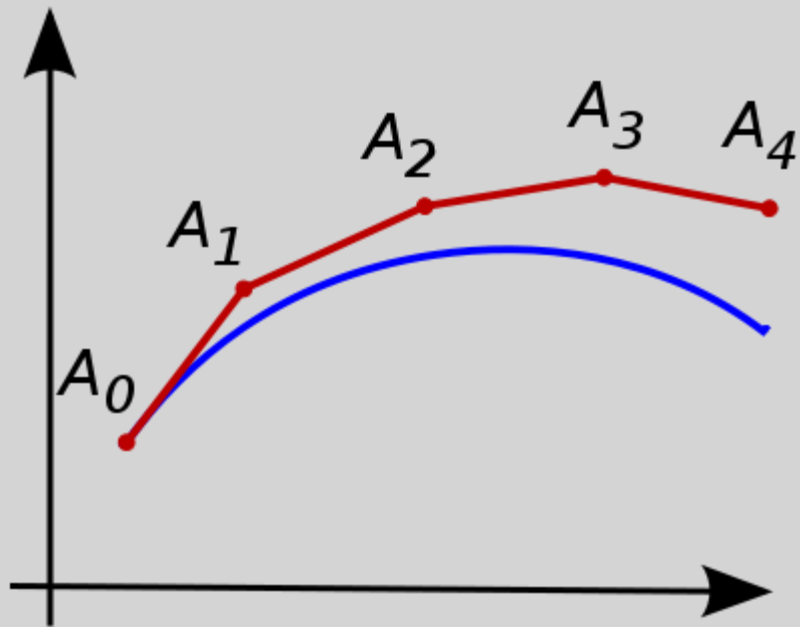
Blender/Python API Numerical Integration

Euler method: Adding physical damping to the model (e.g. Rayleigh damping). In this case the damping will only be applied to the structure and the question is how much damping to introduce when physically it is negligible.

Adding numerical damping. This reduces the numerical oscillations, but also reduces the physical response which should be solved for, and the question is how much numerical damping to introduce in order to obtain acceptable results.

Blender/Python API

Numerical Integration



Blender/Python API

Numerical Integration

Integration

Euler

Also known as “Forward Euler”. Simplest integrator. Very fast but also with less exact results. If no dampening is used, particles get more and more energy over time. For example, bouncing particles will bounce higher and higher each time. Should not be confused with “Backward Euler” (not implemented) which has the opposite feature, the energy decrease over time, even with no dampening. Use this integrator for short simulations or simulations with a lot of dampening where speedy calculations are more important than accuracy.

Verlet

Very fast and stable integrator, energy is conserved over time with very little numerical dissipation.

Midpoint

Also known as “2nd order Runge-Kutta”. Slower than Euler but much more stable. If the acceleration is constant (no drag for example), it is energy conservative. It should be noted that in example of the bouncing particles, the particles might bounce higher than they started once in a while, but this is not a trend. This integrator is a generally good integrator for use in most cases.

RK4

Short for “4th order Runge-Kutta”. Similar to Midpoint but slower and in most cases more accurate. It is energy conservative even if the acceleration is not constant. Only needed in complex simulations where Midpoint is found not to be accurate enough.



Blender 4.2 Manual

Search



Blender/Python API

Numerical Integration

Carl David Tolme **Runge** – Martin Wilhelm **Kutta**



*Carl David
Tolmé Runge*



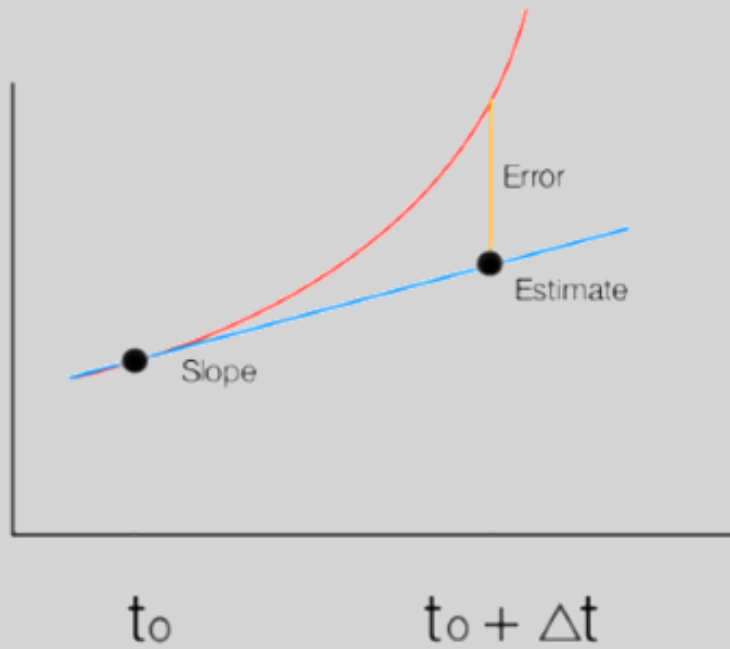
Martin Wilhelm Kutta



Blender/Python API

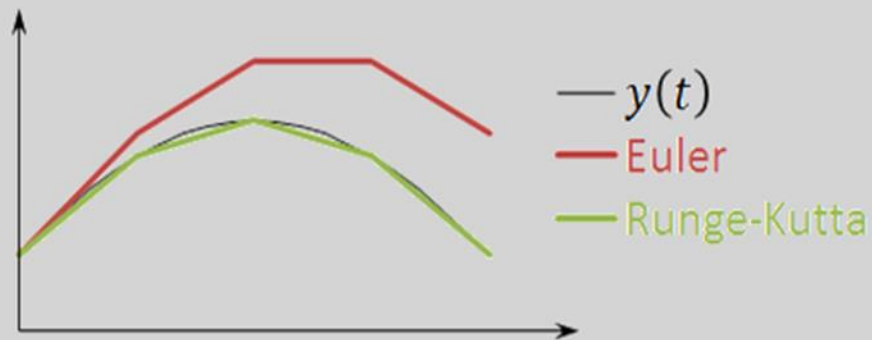
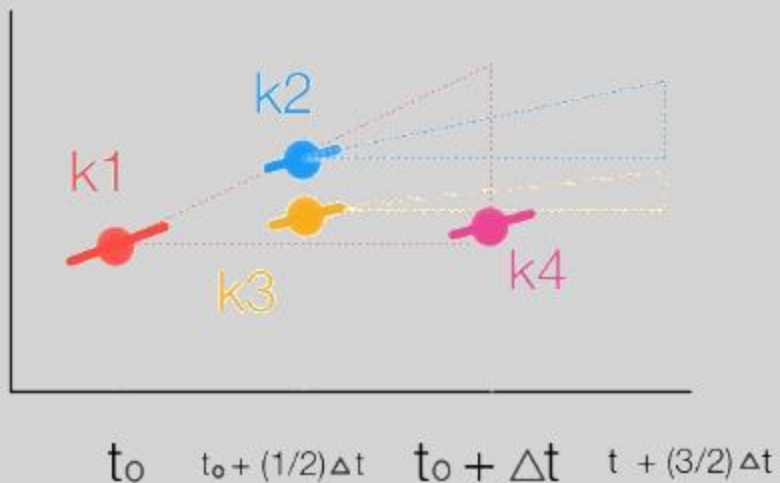
Numerical Integration

Euler's Method





Blender/Python API Numerical Integration



Slopes

$$v_{t+\Delta t} = v_t + \frac{\Delta t}{6} (k1 + 2k2 + 2k3 + k4)$$



Blender/Python API 5 Spheres and 1 Plane

```
import bpy

bpy.ops.mesh.primitive_plane_add(size=5, location=(0, 0, 0))
bpy.ops.rigidbody.object_add()
bpy.context.object.rigidbody.type = 'PASSIVE'
bpy.context.object.rigidbody.collision_shape = 'MESH'
bpy.context.object.rigidbody.mesh_source = 'BASE'

for i in range(5, 11):
    bpy.ops.mesh.primitive_uv_sphere_add(radius=0.5, location=(0, 0, i))
    bpy.ops.rigidbody.object_add()
    bpy.context.object.rigidbody.type = 'ACTIVE'
    bpy.context.object.rigidbody.collision_shape = 'SPHERE'
```

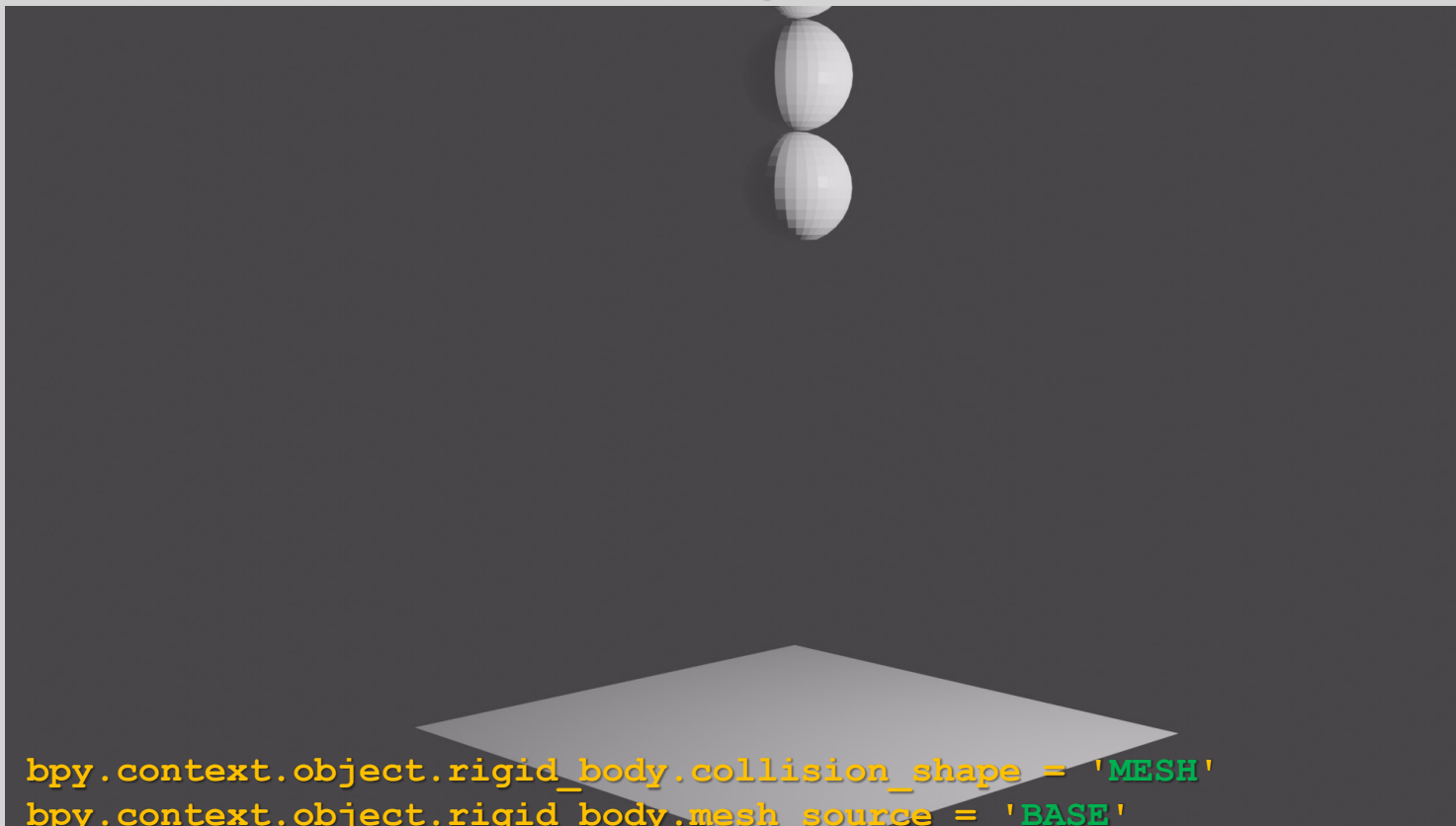


Blender/Python API 5 Spheres and 1 Plane





Blender/Python API 5 Spheres and 1 Plane

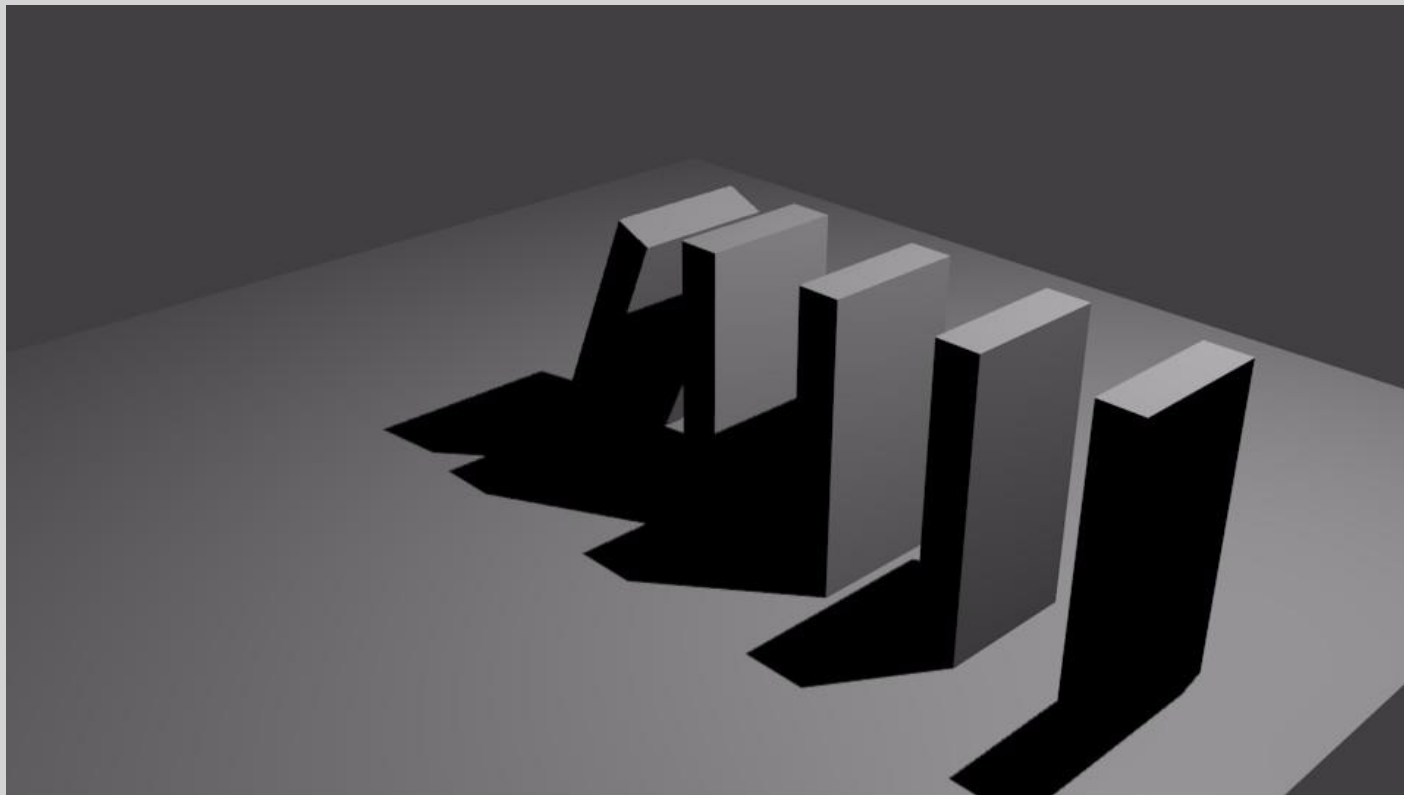




Blender/Python API

5 Bricks and 1 Plane

Homework





Blender/Python API

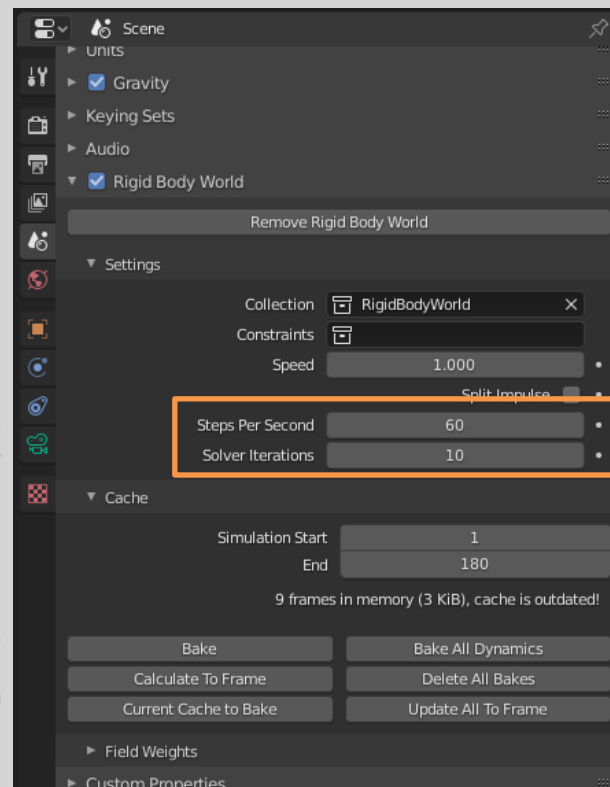
Rigid Body World

Scene ► Rigid Body World

Simulation Stability

The simplest way of improving simulation stability is to increase the **steps per second**. However, care has to be taken since making too many steps can cause problems and make the simulation even less stable (if you need more than 1000 steps, you should look at other ways to improve stability).

Increasing the number of **solver iterations** helps making constraints stronger and also improves object stacking stability.





Blender/Python API

Rigid Body World

Scene ► Rigid Body World

Steps per Second

Number of simulation steps made per second (higher values are more accurate but slower). This only influences the accuracy and not the speed of the simulation.

Solver Iterations

Amount of constraint solver iterations made per simulation step (higher values are more accurate but slower). Increasing this makes constraints and object stacking more stable.



Simulation Stability

It is best to avoid small objects, as they are currently unstable. **Ideally, objects should be at least 20 cm in diameter.** If it is still necessary, setting the collision margin to 0, while generally not recommended, can help making small object behave more naturally.

When objects are small and/or move very fast, they can pass through each other. Besides what is mentioned above it's also good to avoid using mesh shapes in this case. Mesh shapes consist of individual triangles and therefore do not really have any thickness, so objects can pass through more easily. You can give them some thickness by increasing the collision margin.

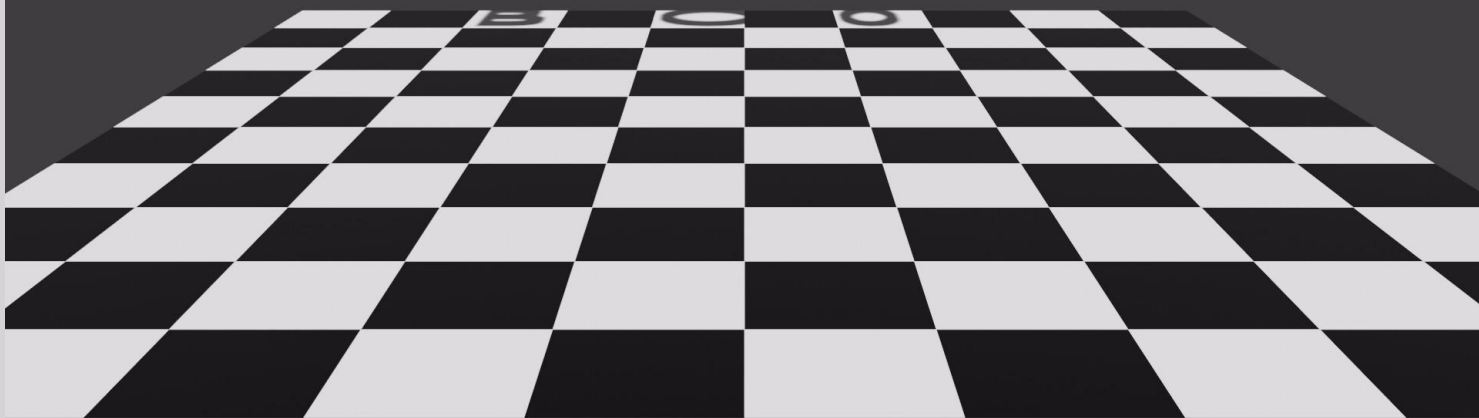


Blender/Python API

Rigid Body World

Scene ► Rigid Body World

```
bpy.context.scene.rigidbody_world.steps_per_second = 60  
bpy.context.scene.rigidbody_world.solver_iterations = 10
```



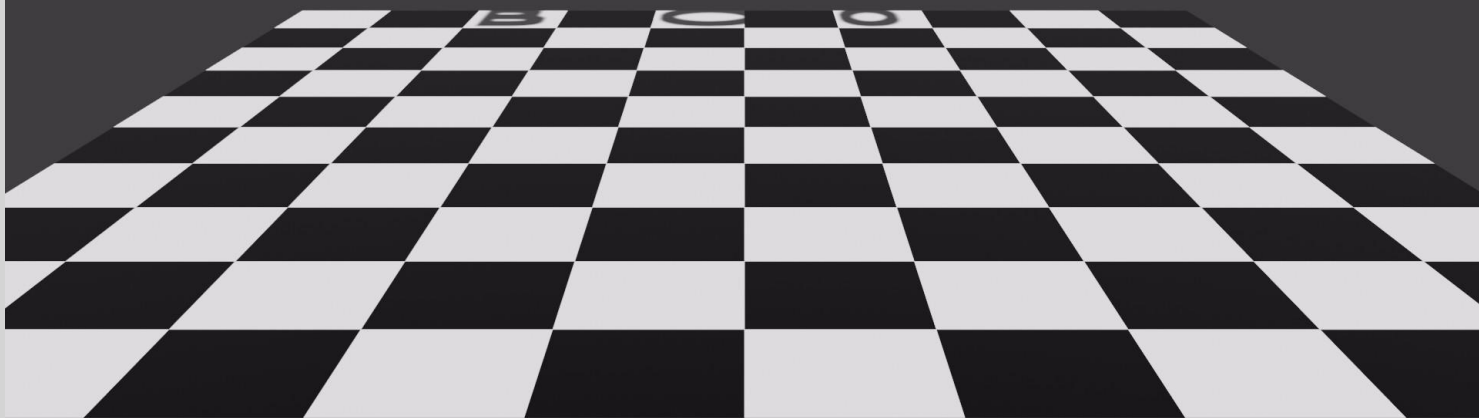


Blender/Python API

Rigid Body World

Scene ► Rigid Body World

```
bpy.context.scene.rigidbody_world.steps_per_second = 120  
bpy.context.scene.rigidbody_world.solver_iterations = 24
```



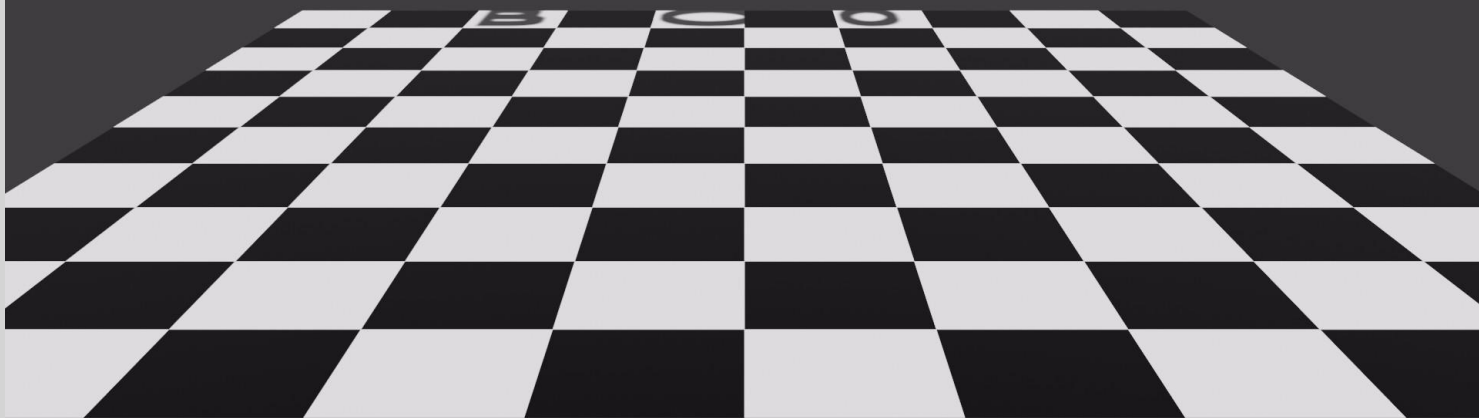


Blender/Python API

Rigid Body World

Scene ► Rigid Body World

```
bpy.context.scene.rigidbody_world.steps_per_second = 240  
bpy.context.scene.rigidbody_world.solver_iterations = 36
```



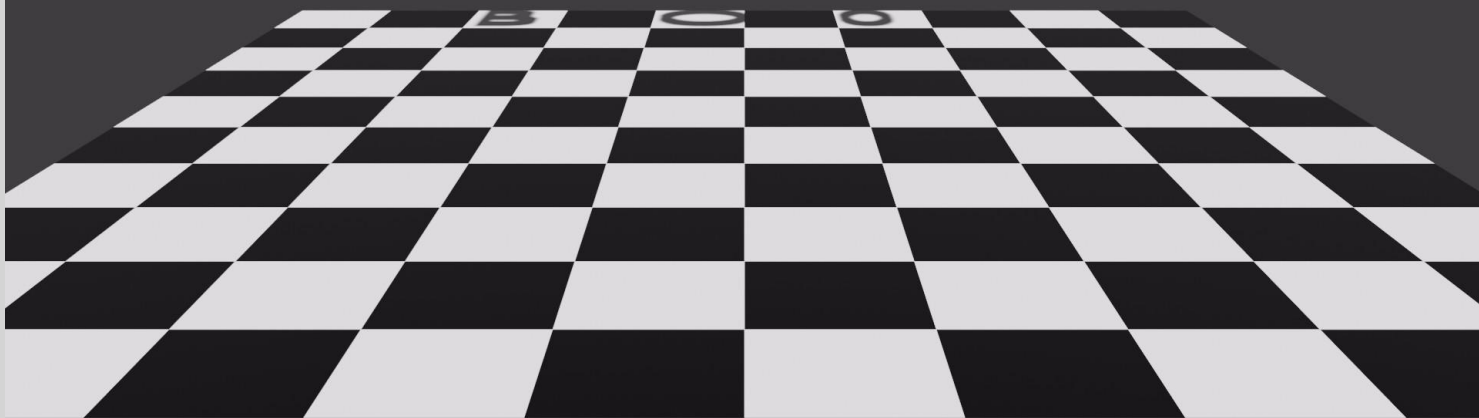


Blender/Python API

Rigid Body World

Scene ► Rigid Body World

```
bpy.context.scene.rigidbody_world.steps_per_second = 360  
bpy.context.scene.rigidbody_world.solver_iterations = 64
```





Blender/Python API Rigid Letters

```
import bpy
from math import radians

# Create and name Text object
bpy.ops.object.text_add(location=(0, 0, 0))
obj = bpy.context.object
obj.name = 'Letter'
obj.data.name = 'LetterData'
# Data attributes
obj.data.body = 'S'
obj.data.font = bpy.data.fonts[0]
obj.data.offset_x = 0
obj.data.offset_y = 0
obj.data.shear = 0
obj.data.space_character = 0
obj.data.size = 3
obj.data.space_word = 0
obj.data.extrude = 0.1
# Rotate 90 degrees
bpy.ops.transform.rotate(value=radians(90), orient_axis='X')
# Convert to a mesh
bpy.ops.object.convert(target='MESH')
```

You can convert a text object, either to a curve, or directly to a mesh, using Convert To in Object Mode.



Blender/Python API Rigid Letters

```
bpy.ops.object.origin_set(type='ORIGIN_CENTER_OF_VOLUME', center='MEDIAN')
```

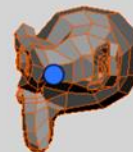
Rigidbody

```
bpy.ops.rigidbody.object_add()  
bpy.context.object.rigid_body.type = 'ACTIVE'  
bpy.context.object.rigid_body.collision_shape = 'MESH'
```

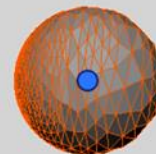
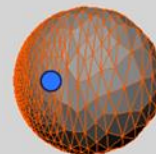
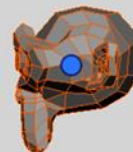
plane

```
bpy.ops.mesh.primitive_plane_add(size=10, location=(0, 0, 0))  
bpy.ops.rigidbody.object_add()  
bpy.context.object.rigid_body.type = 'PASSIVE'  
bpy.context.object.rigid_body.collision_shape = 'MESH'
```

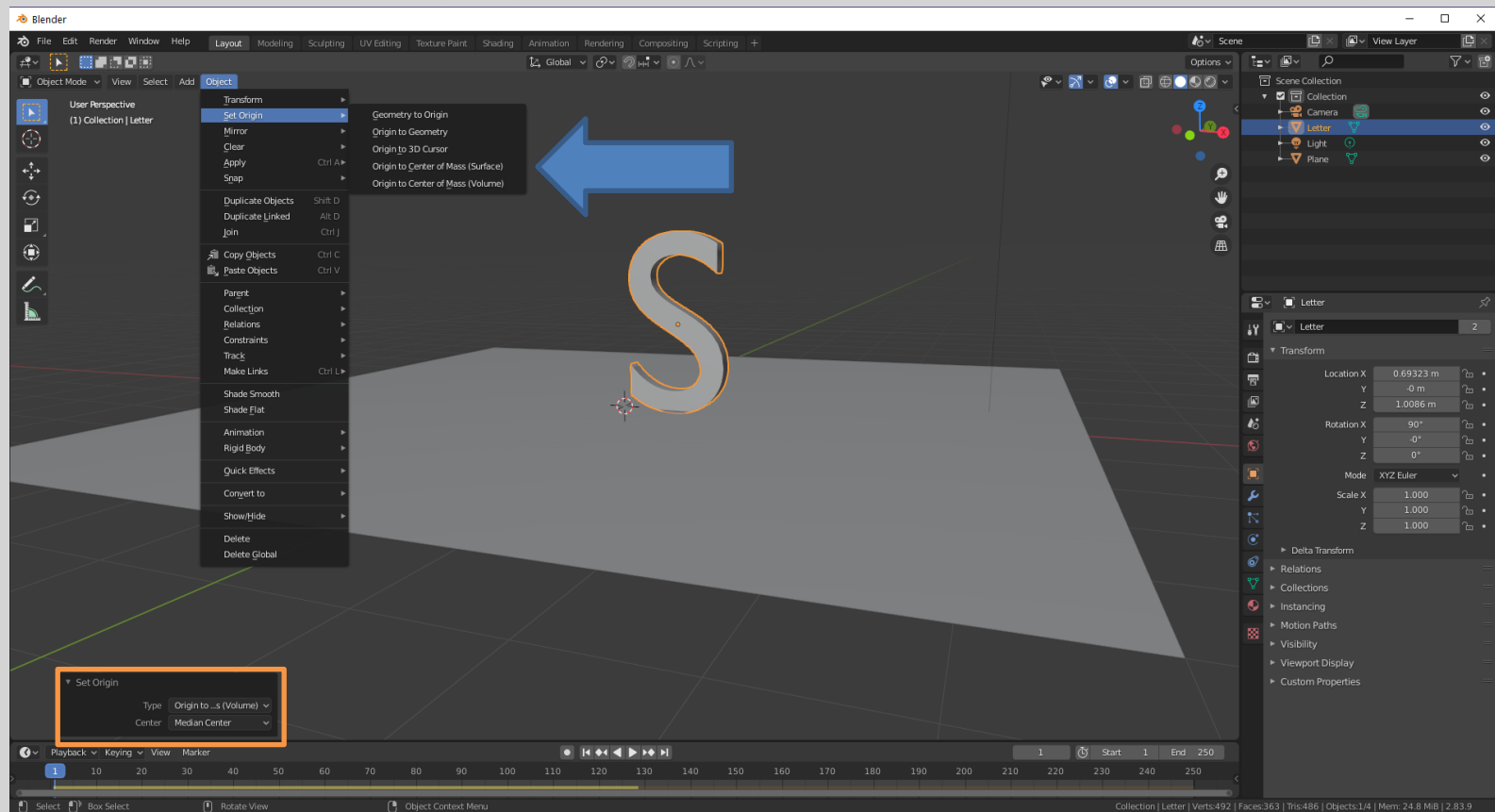
to Geometry



to Mass



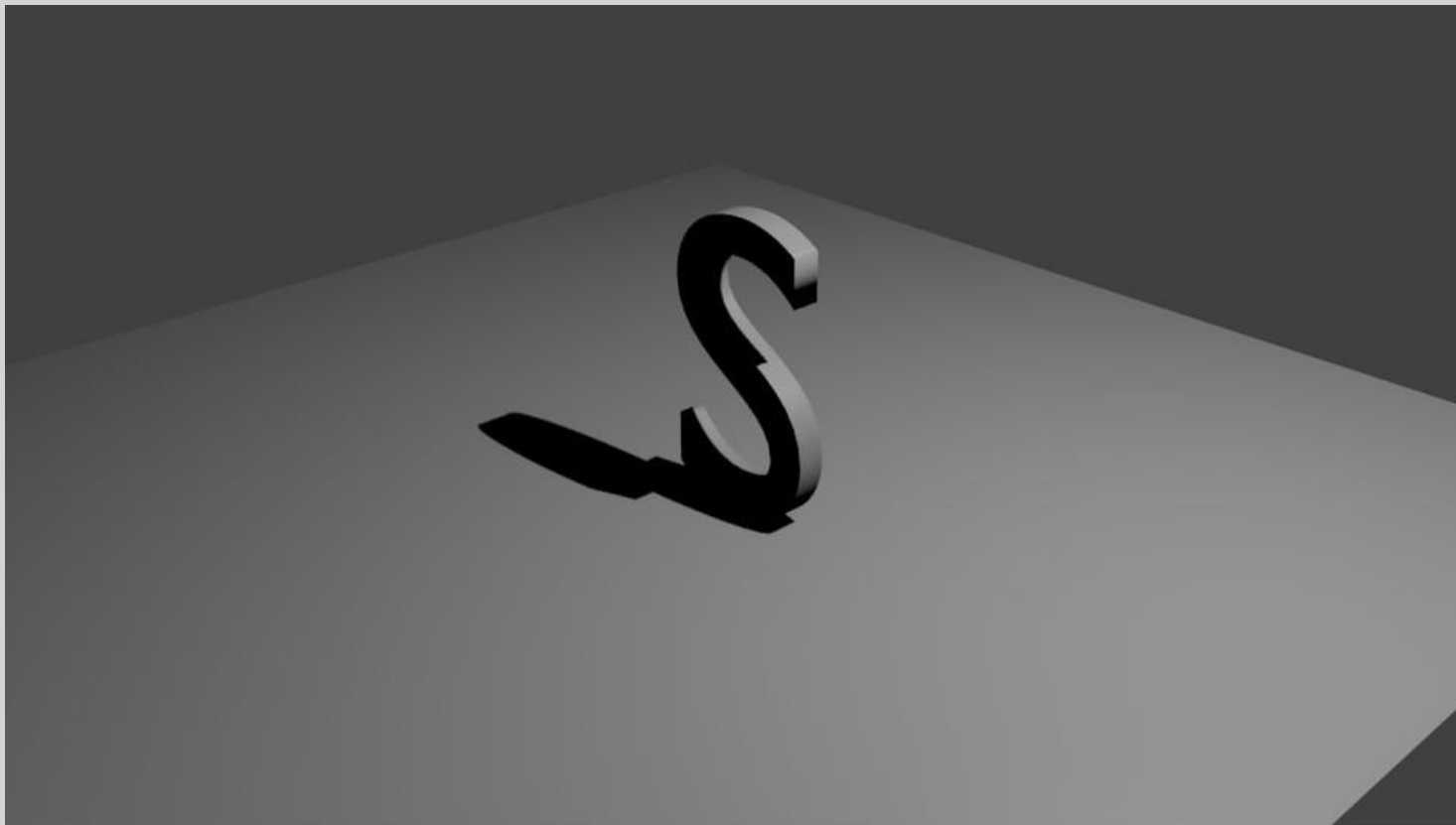
Blender/Python API Rigid Letters





Blender/Python API

Rigid Letters



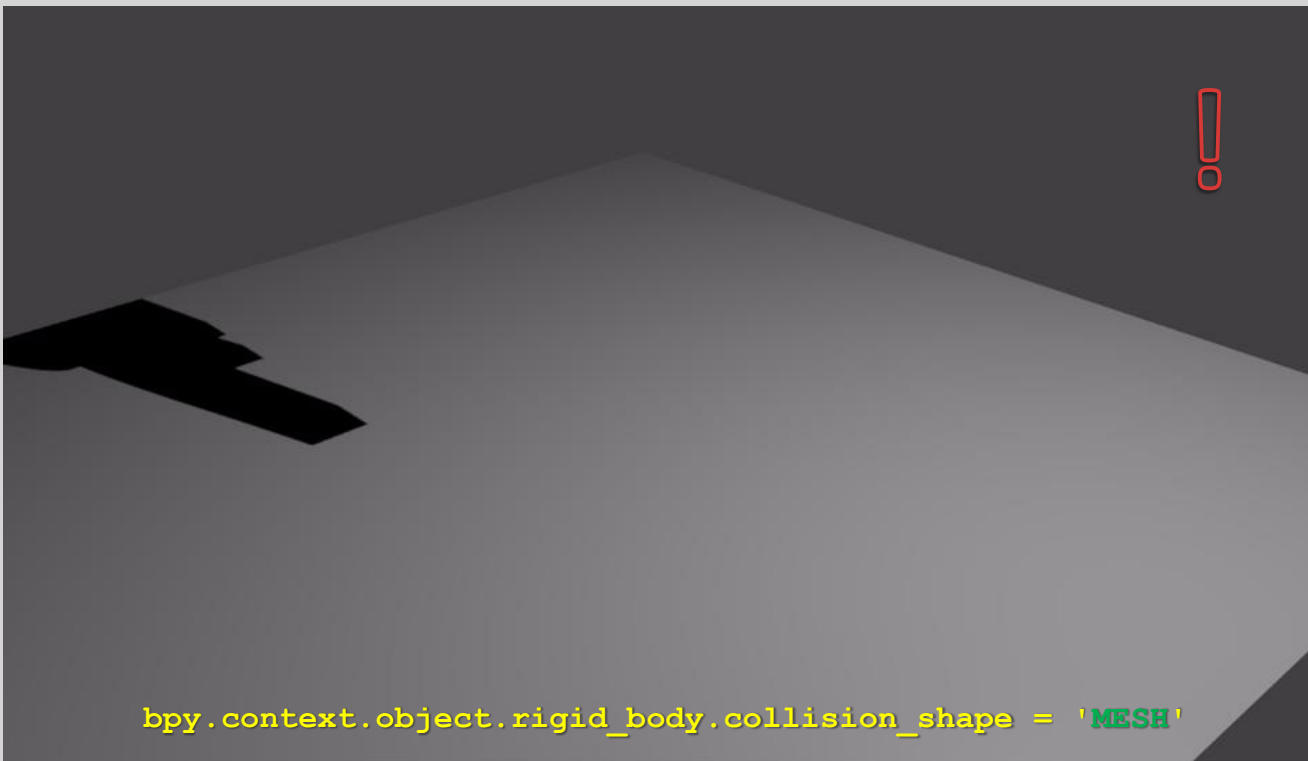


Blender/Python API

Defining a rigid letter function

Homework

```
def rigidLetter(letter='', letterSize=1.0 , loc=(0, 0, 0)):
```

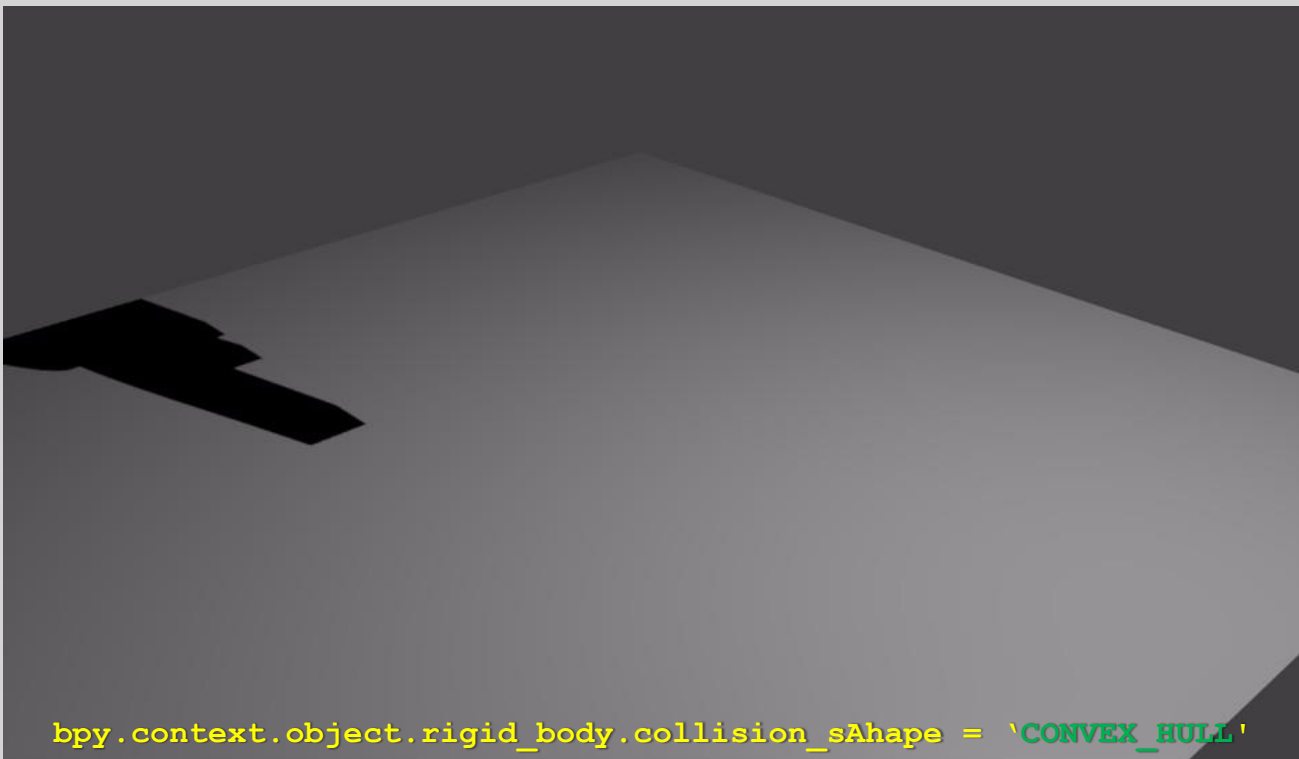




Blender/Python API

Defining a rigid letter function

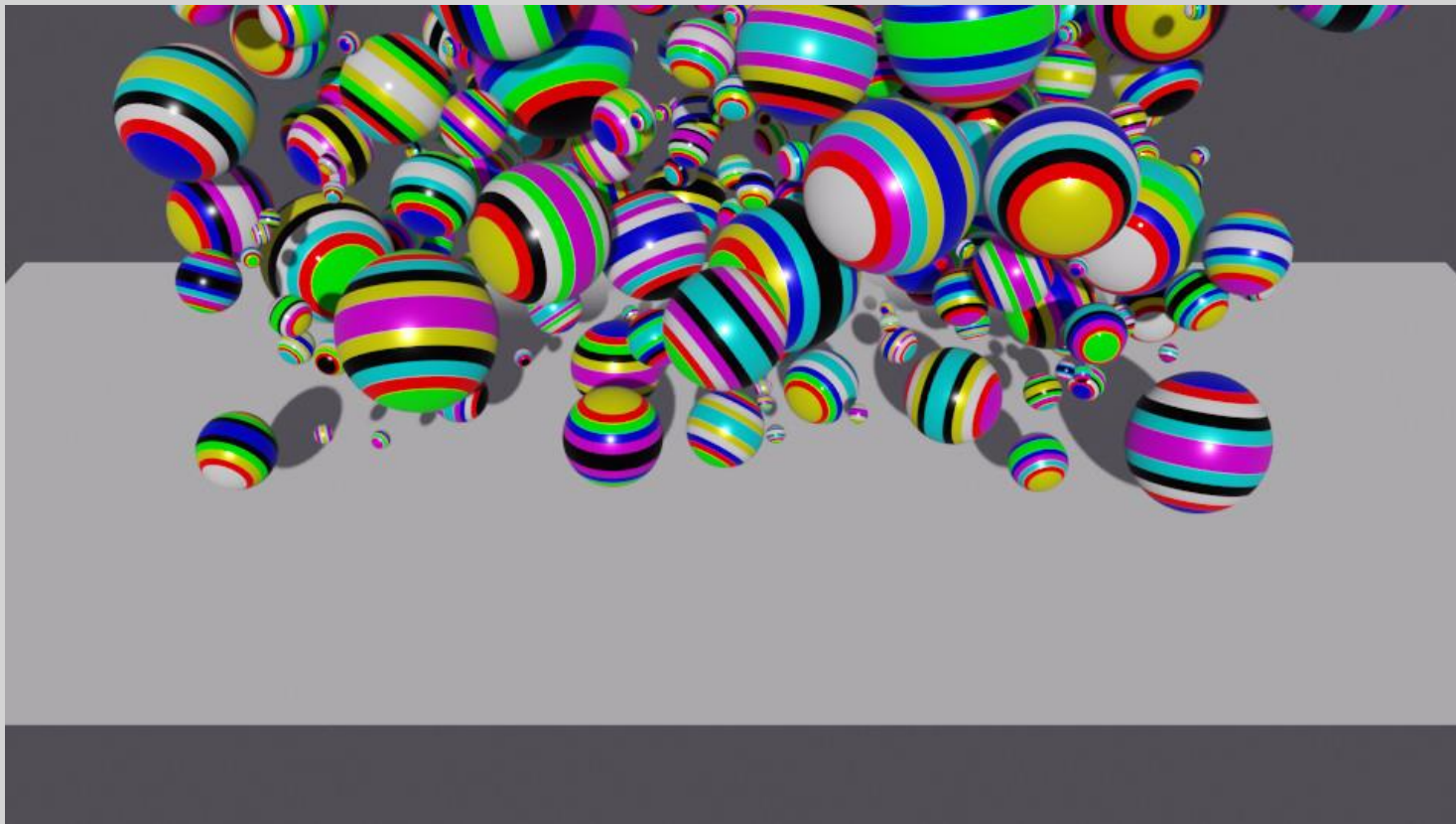
Homework





Blender/Python API

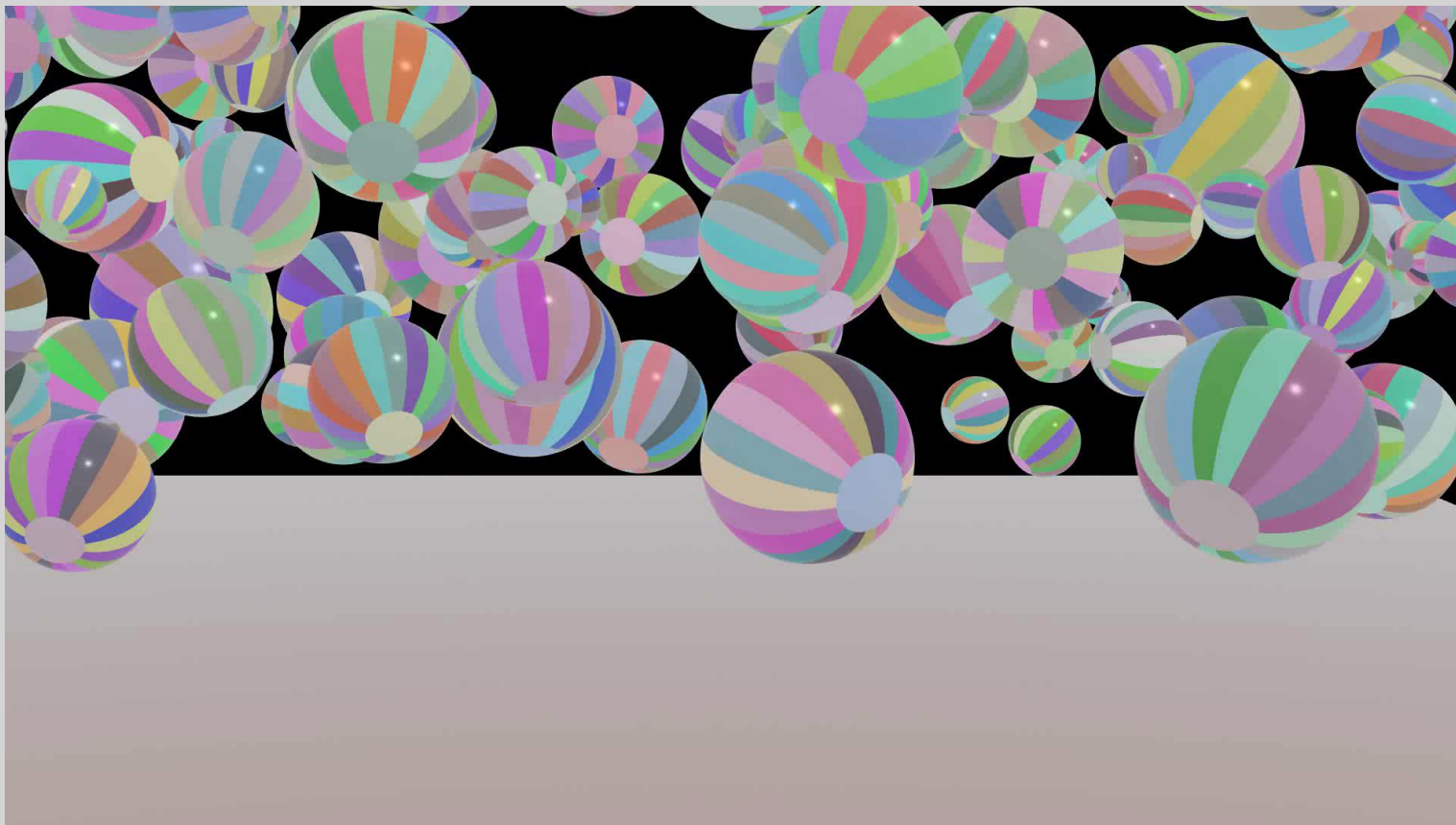
Mid-Term Exam I





Blender/Python API

Mid-Term Exam I



E-posta Sınav teslim tarihi : **13 Kasım 2024 Çarşamba** saat 23:59

Sınav yeri : **14 Kasım 2024 Perşembe** saat 18:30 Animasyon Lab
13 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

Mid-Term Exam I

