

User manual search (offline, online), forum search

- Dynamics
 - Designing dynamic simulations
 - Physics engine differences
 - simMujoco API reference
 - Data visualization/output
 - Web-browser based front-end
 - Data manipulation/transformation
 - Messaging/interfaces/connectivity
 - Remote API
 - ZeroMQ remote API
 - WebSocket remote API
 - ROS Interfaces
 - ROS Interface
 - simROS API reference
 - ROS 2 Interface
 - simROS2 API reference
 - simZMQ API reference
 - simWS API reference
 - Paths/trajectories
 - Path planning
 - simOMPL API reference
 - Synthetic vision
 - simIM API reference
 - simVision API reference
 - Custom user interfaces
 - simUI API reference
 - simUI XML syntax
 - simQML API reference
 - Import/export
 - XML format
 - URDF format
 - simURDF API reference
 - SDF format
 - simSDF API reference
 - Video exporter
 - simAssimp API reference



Dynamics

CoppeliaSim currently supports five different physics engines: the Bullet physics library, the Open Dynamics Engine, the MuJoCo physics engine, the Vortex Studio engine and the Newton Dynamics engine. There are smaller differences between the engines, but one can switch at any time from one to another, depending on one's needs. The reason for the large diversity in physics engine support in CoppeliaSim is that physics simulation is a complex task, that can be achieved by emphasising one or another aspect of a simulation, such as precision/realism, speed, or with support of diverse features:

- Bullet physics library:** an open source physics engine featuring 3D collision detection, rigid body dynamics, and soft body dynamics. It is used in games, and in visual effects in movies. It is often considered as a *game physics engine*.



[Bullet Physics]

- Open Dynamics Engine (ODE):** an open source physics engine with two main components: rigid body dynamics and collision detection. It has been used in many applications and games. It is often considered as a *game physics engine*.



[Open Dynamics Engine]

- MuJoCo:** an open source state-of-the-art physics engine, producing high fidelity physics simulations. It supports a large variety of features such as soft body physics, ropes, strings/elastics, tendons, various specific constraints (e.g. dependency constraints between individual joints), etc. Via [XML injection](#), one is able to additionally cover many MuJoCo specificities.



[MuJoCo]

- Vortex® Studio:** a closed source, commercial physics engine producing high fidelity physics simulations. Vortex offers real-world parameters (i.e. corresponding to physical units) for a large number of physical properties, making this engine both realistic and precise. Vortex is mainly used in high performance/precision industrial and research applications. The Vortex plugin for CoppeliaSim is based on [Vortex Studio](#), which requires each user to [register with CM Labs](#), for a free license key.



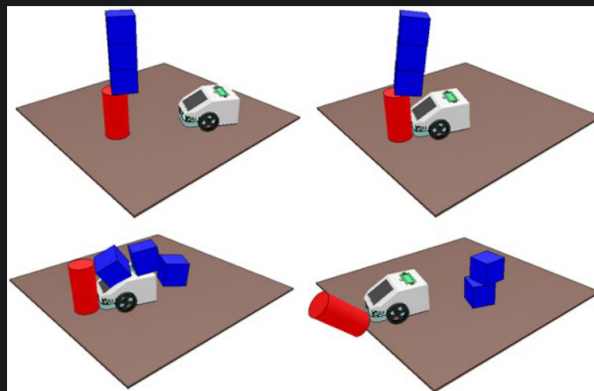
[Vortex Studio]

- Newton Dynamics:** Newton Dynamics is a cross-platform life-like physics simulation library. It implements a deterministic solver, which is not based on traditional LCP or iterative methods, but possesses the stability and speed of both respectively. This feature makes Newton Dynamics a tool not only for games, but also for any real-time physics simulation.



[Newton Dynamics]

A physics engine allows simulating interactions between [objects](#) that are near to real-world object interactions. It allows objects to fall, collide, bounce back, but it also allows a manipulator to grasp objects, a conveyor belt to drive parts forward, or a vehicle to roll in a realistic fashion over uneven terrain. Following figures illustrates such a simulation:



[Dynamics simulation]

Unlike many other simulation software packages, CoppeliaSim is not a *pure* physics engine centered

simulator. It can rather be seen as a hybrid simulator, that combines kinematics and dynamics in order to obtain the best performance for various simulation scenarios. Nowadays, physics engines are still relying on many approximations and are relatively imprecise and slow, and wherever possible you should try to use kinematics instead (e.g. for robotic manipulators) and only rely on dynamics where otherwise not feasible (e.g. a robotic manipulator's gripper, mobile robots).