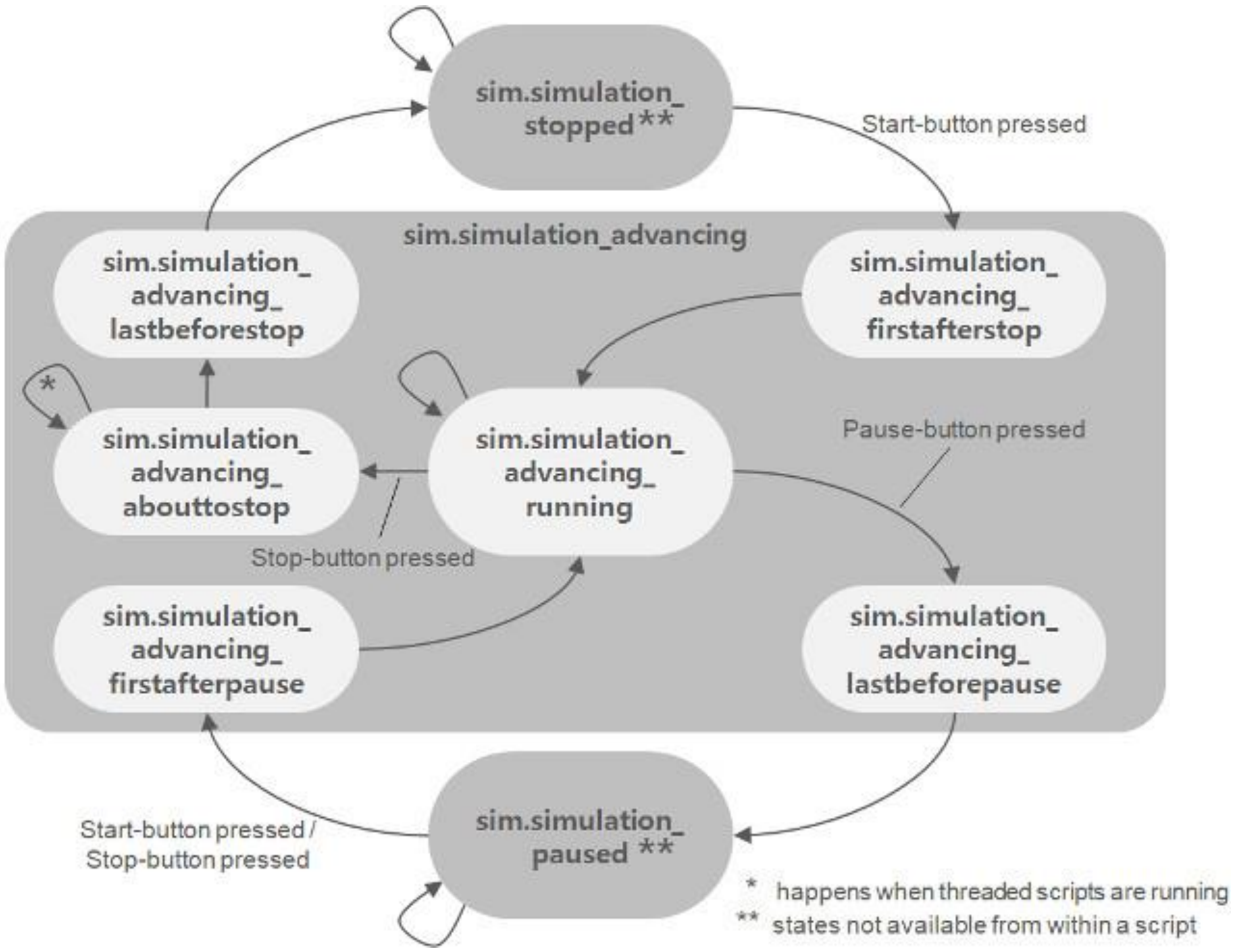
Simulation

A simulation in CoppeliaSim can be started, paused and stopped with [Menu bar --> Simulation --> Start/Pause/Stop simulation] or through the related toolbar buttons:

[Simulation start/pause/stop toolbar buttons]

Internally, the simulator will use additional intermediate states in order to correctly inform scripts or programs about what will happen next. Following state diagram illustrates the simulator's internal states:


sim.simulation_stopped**
Start-button pressed
sim.simulation_advancing
sim.simulation_advancing_lastbeforestop
sim.simulation_advancing_firstafterstop
*
sim.simulation_advancing_abouttostop
sim.simulation_advancing_running
Pause-button pressed
Stop-button pressed
sim.simulation_advancing_firstafterpause
sim.simulation_advancing_lastbeforepause
Start-button pressed / Stop-button pressed
sim.simulation_paused **
* happens when threaded scripts are running
** states not available from within a script

[Simulation state diagram]

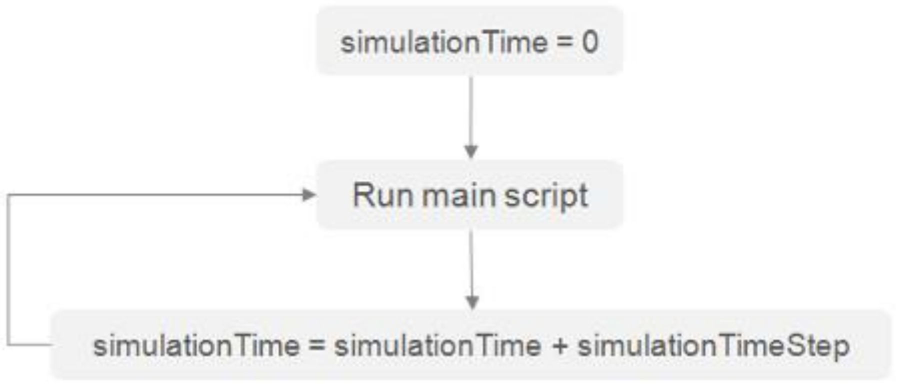
Scripts and programs should alwaysreact according to the current system call function and possibly the simulation state in order to behave correctly. It is good practice to divide each control code into at least 4 system call functions (e.g. for non-threaded child scripts):

- **Initialization function**: **sysCall_init**: the function is called only when the script is initialized.
- **Actuation function**: **sysCall_actuation**: the function is called when actuation should happen.
- **Sensing function**: **sysCall_sensing**: this function is called when sensing should happen.
- **Clean-up function**: **sysCall_cleanup**: the function is called just before the script is de-initialized (e.g. at simulation end, or when the script is destroyed).

For examples on how to arrange a typical script, refer to the main script, the child scripts and customization scripts pages.

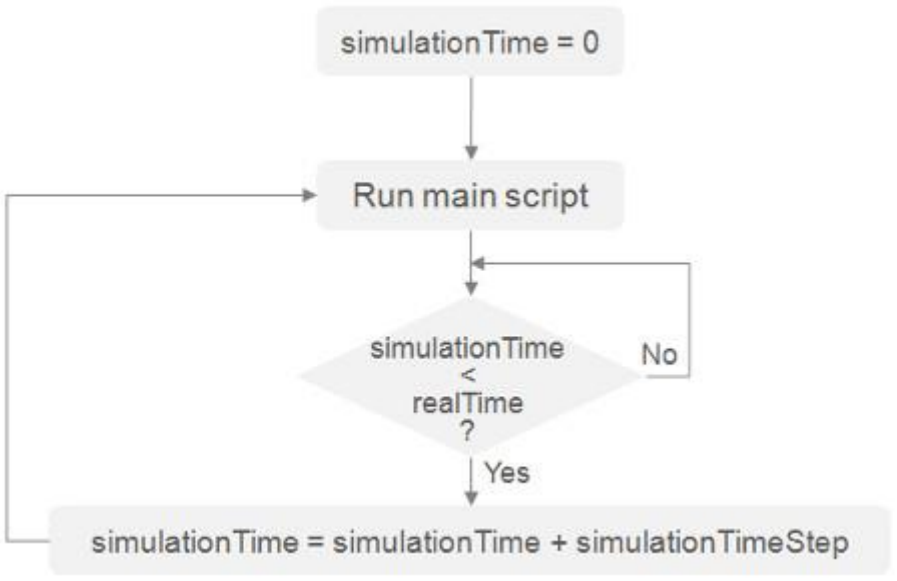
Simulation loop

The simulator operates by advancing the simulation time at constant time steps. Following figure illustrates the main simulation loop:



[Main simulation loop]

Real-time simulation is supported by trying to keep the simulation time synchronized with the real time:


simulationTime = 0
Run main script
simulationTime < realTime ?
No
Yes
simulationTime = simulationTime + simulationTimeStep

[Real-time simulation loop]

Following represents a very simplified main client application (messaging, plugin handling and other details have been omitted for clarity purpose):

```
void initializationCallback
{
    // do some initialization here
}

void loopCallback
{
    if ( (simGetSimulationState()&sim_simulation_advancing)!=0 )
    {
        if
( (simGetRealTimeSimulation()!=1)||(simIsRealTimeSimulationStepNeed
ed()==1) )
        {
            if
((simHandleMainScript()&sim_script_main_script_not_called)==0)
                simAdvanceSimulationByOneStep();
        }
    }
}

void deinitializationCallback
{
    // do some clean-up here
}
```

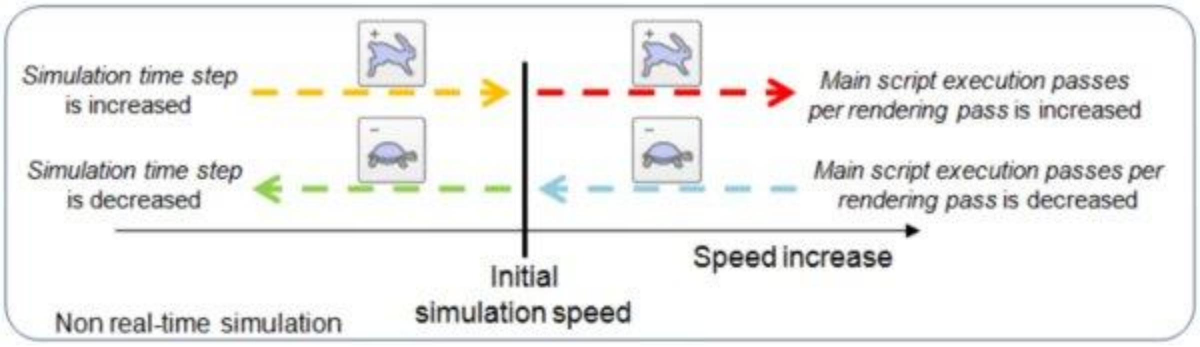
Depending on the simulation complexity, performance of the computer and simulation settings, real-time simulation might not always be possible.

Simulation speed

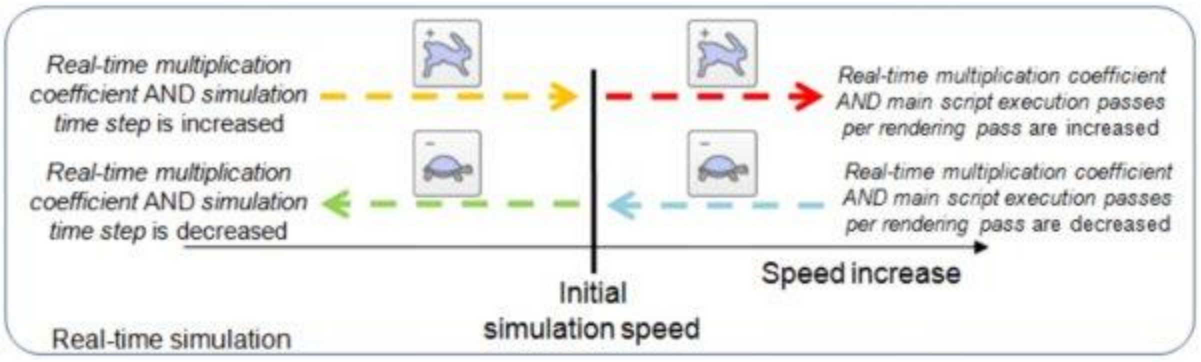
In non real-time simulations, the simulation speed (i.e. the perceived speed) is mainly dependent on two factors: the simulation time step and the number of simulation passes for one rendering pass (see the simulation dialog for more details). In the case of a real-time simulation, the simulation speed mainly depends on the real-time multiplication coefficient, but also to a certain degree of the simulation time step (a too small simulation time step might not be compatible with the real-time character of a simulation because of the limited calculation power of the computer). During simulation, the simulation speed can be adjusted with following toolbar buttons:

[Simulation speed adjustment toolbar buttons]

The simulation speed is adjusted in a way so that the initial simulation time step is never increased (because this might have as consequence the breaking of a mechanism for example). Following two figures illustrate the simulation speed adjustment mechanisms:


Simulation time step is increased
Main script execution passes per rendering pass is increased
Simulation time step is decreased
Main script execution passes per rendering pass is decreased
Speed increase
Initial simulation speed
Non real-time simulation

[Simulation speed adjustment mechanism for **non real-time simulations**]


Real-time multiplication coefficient AND simulation time step is increased
Real-time multiplication coefficient AND main script execution passes per rendering pass are increased
Real-time multiplication coefficient AND simulation time step is decreased
Real-time multiplication coefficient AND main script execution passes per rendering pass are decreased
Speed increase
Initial simulation speed
Real-time simulation

[Simulation speed adjustment mechanism for **real-time simulations**]

By default, each simulation cycle is composed by following **sequential** operations:

- Executing the main script
- Rendering the scene

Threaded rendering

The rendering operation will always increase the simulation cycle duration, thus also slowing down simulation speed. The number of main script executions per scene rendering can be defined (see further up), but this is not enough in some situations, because rendering will still slow down every xth simulation cycle (which can be handicapping with real-time requirements). For those situations, a threaded rendering mode can be activated via the user settings, or via the following toolbar button:

[Threaded rendering toolbar button]

When the threaded rendering mode is activated, a simulation cycle will only consist in execution of the main script, thus simulations will run at maximum speed. Rendering will happen via a different thread, and not slow down the simulation task. The drawbacks have however to be considered. When threaded rendering is activated, then:

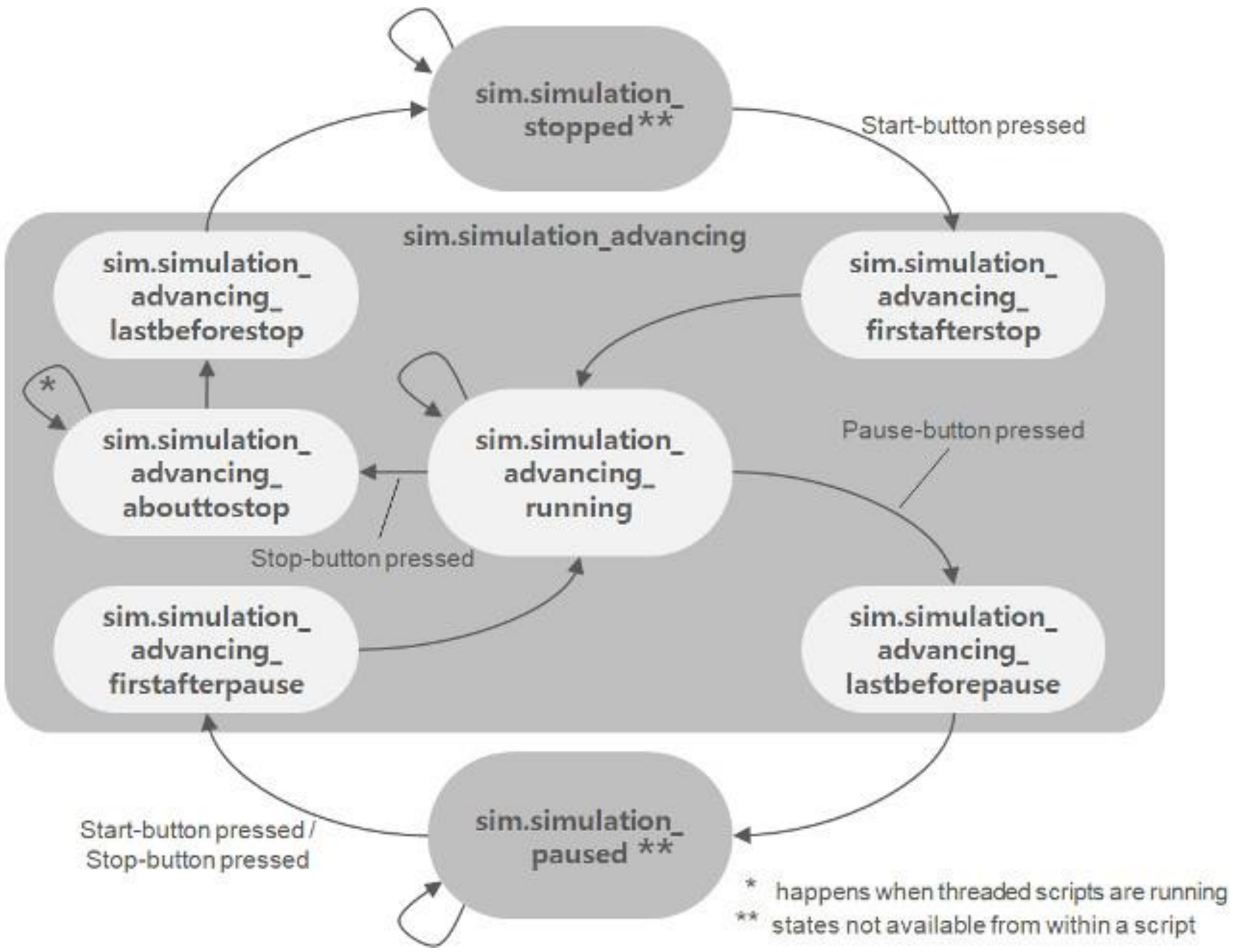
- Rendering will happen asynchronously to the simulation loop, and visual glitches might appear
- The video recorder will not operate at constant speed (some frames might get skipped)
- The stability of the application might be reduced
- Some operations (e.g. erasing an object, etc.) require to wait for the rendering thread to finish work, before being able to execute, and vice-versa. In those situations, cycles could take more time than in the sequential rendering mode.

模擬

在 coppeliaSim 中的模擬可透過[選單-->模擬-->開始/暫停/停止模擬]或相關工具欄按鈕來啟動

[Simulation start/pause/stop toolbar buttons]

為了能正確告知下一步的狀態，在程序中模擬器將會使用其他內部狀態，以下為模擬器內部狀態


sim.simulation_stopped**
Start-button pressed
sim.simulation_advancing
sim.simulation_advancing_lastbeforestop
sim.simulation_advancing_firstafterstop
*
sim.simulation_advancing_abouttostop
sim.simulation_advancing_running
Pause-button pressed
Stop-button pressed
sim.simulation_advancing_firstafterpause
sim.simulation_advancing_lastbeforepause
Start-button pressed / Stop-button pressed
sim.simulation_paused **
* happens when threaded scripts are running
** states not available from within a script

[Simulation state diagram]

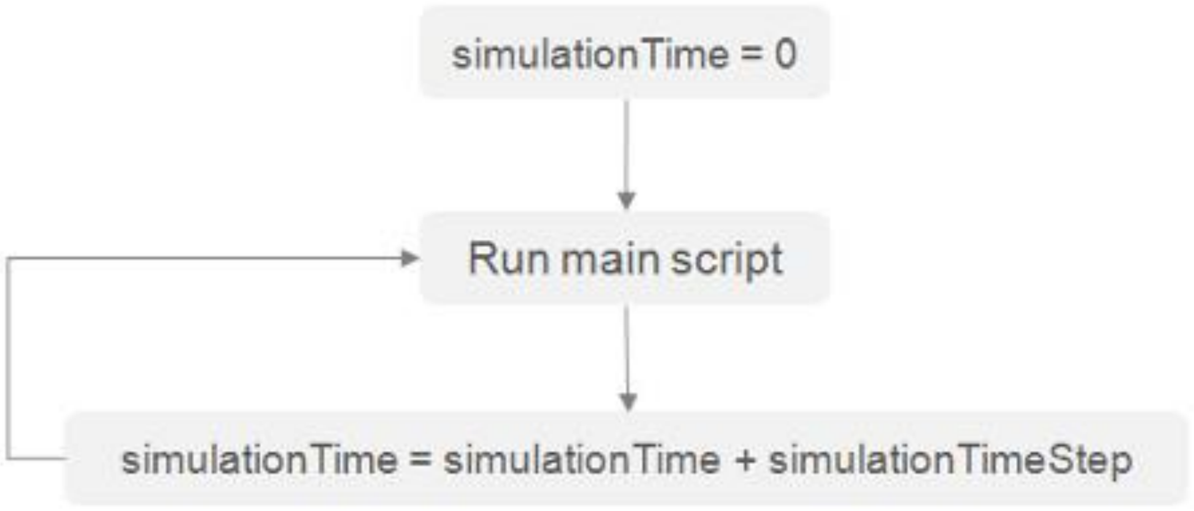
為了能正確執行，腳本與程序必須根據當前系統及模擬型態進行修正，較佳的作法為將每個控制代碼分成 4 個系統函數

- **初始函數: 僅在腳本初始化使用**
- **激活函數: 僅在激活時使用.**
- **感知函數: 僅在感應發生使用.**
- **清除函數:僅在函數為初始化之前使用**

有關如何操作腳本的例子請參考 main script, child scripts 和 customization scripts

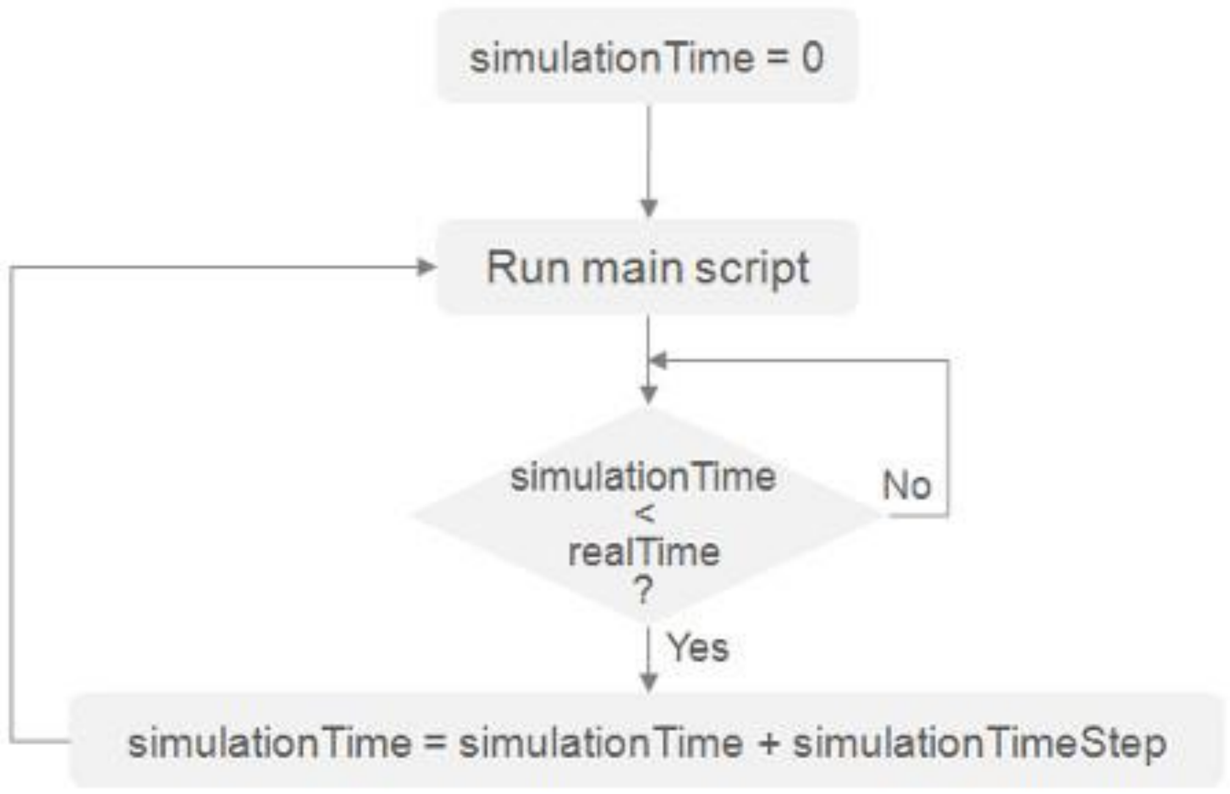
Simulation loop

模擬器透過縮短時間提高仿真的操作，下圖說明了主要仿真循環



[Main simulation loop]

Real-time 模擬支援模擬同步時間



[Real-time simulation loop]

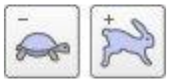
下面是已經精簡過的客戶端應用程序（為了淺顯易懂，下列程式碼已經省略了訊息、插件處理及其他詳細信息):

```
void initializationCallback
{
    // do some initialization here
}

void loopCallback
{
    if
( (simGetSimulationState()&sim_simulation_advancing)!=0 )
    {
        if
( (simGetRealTimeSimulation()!=1)||(simIsRealTimeSimulationSt
epNeeded()==1) )
        {
            if
((simHandleMainScript()&sim_script_main_script_not_called)==0
)
                simAdvanceSimulationByOneStep();
        }
    }
}

void deinitializationCallback
{
    // do some clean-up here
}
```

取決於仿真困難度，有時電腦不一定可以執行。

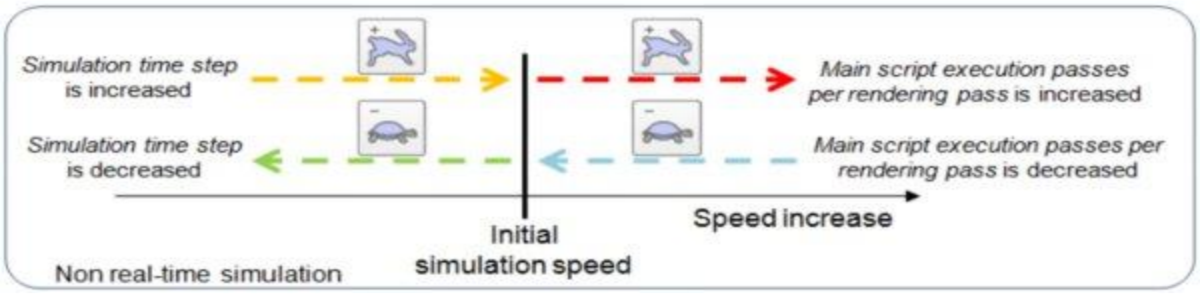
Simulation speed

在 non real-time 模擬，模擬速度取決於兩個要素 : 模擬時間長短和一個渲染的模擬遍數。 在 real-time 仿真的情況下，模擬速度主要取決於實時乘法係數，而且在一定程度上取決於模擬時間長短（太小的仿真時間可能與 real time 時間不兼容）。 由於電腦的計算能力有限，因此無法進行模擬。 在模擬過程中，可以使用以下工具欄按鈕來調整模擬速度

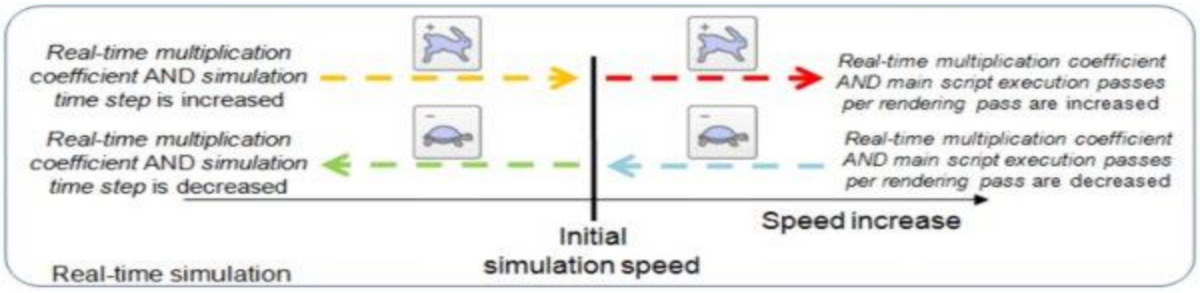


[Simulation speed adjustment toolbar buttons]

以特殊方式調整模擬速度使初始模擬時間永遠不會增加，下圖說明了模擬速度調整機制


Simulation time step is increased
Main script execution passes per rendering pass is increased
Simulation time step is decreased
Main script execution passes per rendering pass is decreased
Speed increase
Initial simulation speed
Non real-time simulation

[Simulation speed adjustment mechanism for **non real-time simulations**]


Real-time multiplication coefficient AND simulation time step is increased
Real-time multiplication coefficient AND main script execution passes per rendering pass are increased
Real-time multiplication coefficient AND simulation time step is decreased
Real-time multiplication coefficient AND main script execution passes per rendering pass are decreased
Speed increase
Initial simulation speed
Real-time simulation

[Simulation speed adjustment mechanism for **real-time simulations**]

在默認情況下，每個周期由以下順序組成

- **執行 main script**
- **渲染場景**

Threaded rendering

渲染將會增加模擬時間，因此我們可以定義每個場景腳本執行次數，但有些特殊情形下這些步驟會不管用，因為渲染仍然會減慢每個第 x 個模擬週期的時間。 在這種情況下，可以通過用戶設置或以下工具欄按鈕激活線程渲染模式

[Threaded rendering toolbar button]

激活線程渲染模式後，模擬週期將包含在執行主腳本中因此模擬將以最大速度運行。渲染將通過不同的線程進行，並且不會減慢模擬任務的速度。然而必須考慮以下缺點

- **渲染將與模擬循環異步發生，並且可能會出現視覺故障**
- **錄像機無法以恆定速度運行**
- **應用程序的穩定性可能會降低**
- **某些操作需要等待渲染線程完成工作才能執行，反之亦然，在那些情況下，循環可能比順序渲染模式花費更多的時間。**