emNewton

(version v.2.200 Last edited on December 21st, 2017)

INTRODUCTION

This plugin is an implementation of Sir Isaac Newton's "*Law of universal Gravity*" that describes the behavior of massive bodies that move and attract each other via gravity. Newton's law can roughly be stated as following:

> "*each particle attracts each other particle with a force that is proportional to the product of their masses and inversely proportional to the square of the distance between them.*"

On the emNewton2 (../../plugin/emnewton2.html) web page you can:

- download the plugin.
- download demo and tutorial scenes.

Note: If you feel that something is not well explained (or not explained at all) then please write me a short e-mail. I will then update the documentation and/or make a demo scene as quickly as possible.

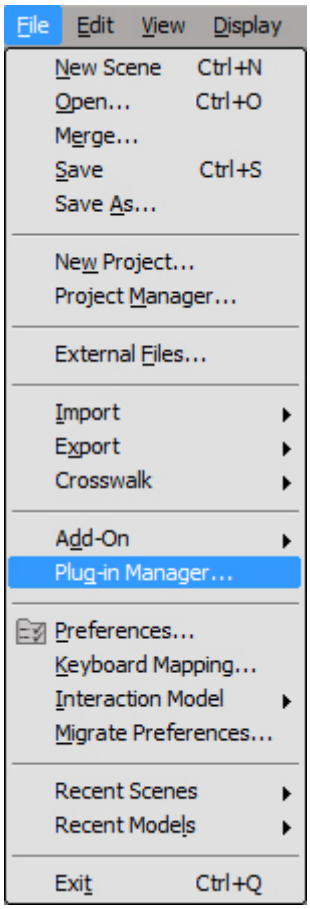
Tip: Check out the following sites for more information, videos and discussions:
si-community.com - the unofficial Softimage community (http://www.si-community.com/)
softimage.tv - the Softimage user based video library (http://softimage.tv/)

INSTALLATION

Important:
- This plugin will only work correctly if the addon emTools (../../plugin/emtools.html) is also installed, so please make sure you have the most recent emTools (../../plugin/emtools.html) installed!
- Before installing this plugin you must uninstall any previous version.

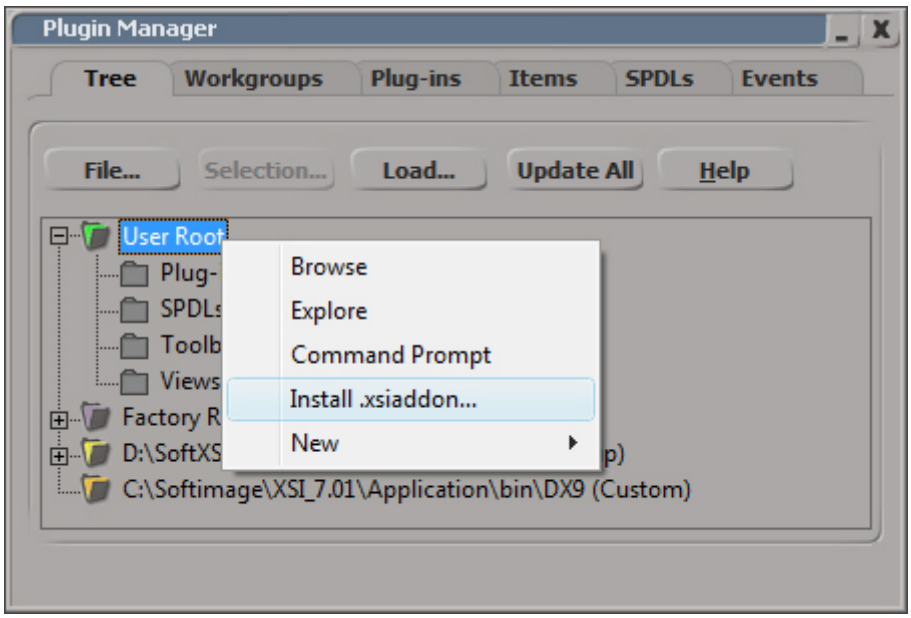
Installing the addon:

1. Open the "Plugin Manager". It is located under "File -> Plugin Manager":



2. As mentioned above: please remove / uninstall any previous version of this plugin. This is really important, because if Softimage finds the same plugin twice then one (maybe even both) might not work.

3. To install the plugin into the user directory simply right-click onto the folder "User Root" and choose "Install .xsiaddon...":



4. A browser dialogue will be displayed: go to where you copied the .xsiaddon file, select it and click on "OK". The addon is then automatically installed.

   Close Softimage. The plugin is now fully installed and ready to be used. For more information concerning addons and how to install / uninstall them please check the chapter "Working with Addons" in the Softimage documentation.

DEMO VERSION RESTRICTIONS

NOTE: IF YOU ARE USING THE LATEST VERSION OF THIS PLUGIN THEN THERE ARE NO RESTRICTIONS AND YOU CAN SKIP THIS CHAPTER.

If you are using the full version of this plug-in then you can skip this chapter. If not, please read the following list of restrictions of the demo version:

- After 600 frames the plugin will not simulate any more, so you must start playback again from the first frame.
- Only up to 25,000 particles simultaneously are allowed. If the number of particles exceeds 25,000 then the plugin has no effect on the particles.

QUICK SETUPS

To quickly create an emNewton2 setup simply get one of the presets from the emNewton2 menu, for example:

"Get -> Primitive -> emNewton2 -> Sun with Spiral"

NOTATION AND UNITS: PHY, XSI, MANTISSA, EXPONENT...

When working with emNewton2 one will often have to deal simultaneously with very small and very large numbers. This can quickly become a bit of a problem, here an example:

Let's say we want to have a little rock orbit around the sun. The rock shall have a diameter of 1 metre, a mass of 3 metric tons and its orbit radius shall be between the Earth's orbit and Mars' orbit. So far so good. Now we need to enter values for all involved objects (e.g. when creating particles for the rock and the sun), so what units to use? Let's try two different versions:

1. *We use "m (metres)" and "kg (kilograms)":* The rock's mass is then 3000 kg, its diameter 1 m and its orbit radius about 187,000,000,000 m.
   The sun has a diameter of 1,392,000,000 m and a mass of 1,989,100,000,000,000,000,000,000,000,000 kg.

2. *We use "AU (Astronomical Unit)" and "Pg (Petagram)":* The rock's mass is then 0.000000003 Pg, its diameter 0.000000000006685 AU and its orbit radius about 1.25 AU.
   The sun has a diameter of 0.0093 AU and a mass of 1,989,100,000,000,000 Pg.

In both cases we have some very impressive numbers, just try entering those in a scalar node in ICE!

A more practical notation is required, the so-called scientific notation in which values are split into a mantissa and an exponent. The above mass of the sun for example would look like this: 1.9891*10^30 kg with 1.9891 being the mantissa and 30 being the exponent. This is already much better (and emNewton2 uses this internally), but still not the ideal solution for our rock-sun example. It would be nice to simply use units that are best suited depending on what is to be described. For the rock a normal person would say: "the rock has a diameter of 1 m and weighs 3 tons". And for the sun something like this would be better: "the sun has a diameter of 1.392 Gm (Gigametre) and weighs 1 Solar mass". It is possible to do exactly that in emNewton2 with the help of a set of compounds called "Convert phy <foo>". These compounds will convert an input (as for example "1.2 AU") into the internal representation, so that we need not worry about that. Those compounds are available for lengths, masses, velocities, speeds and times (or durations).

The emNewton2 naming convention:

In order to avoid confusion when dealing with the units the emNewton2 compounds have a "xsi" or a "phy" before (or after) the parameter names. "xsi" means that it is a normal Softimage value and "phy" means that it is a value with a real physical unit. For example the compound Convert phy Mass will convert the physical inputs into an "ordinary" Softimage value that can be used with a "Set Particle Mass" compound.

The problem with 32 bit float values:

The ICE scalar values are 32 bit floating point numbers. These are great because they have a high enough precision for the common usage and require only 4 bytes of memory per number. The precision is however not high enough for what emNewton2 does. The combination of very small and very big numbers results in rounding errors that get worse the longer the simulation goes. In order to deal with that problem emNewton2 uses double float values for the internal calculations and for the ICE calculations it uses (wherever possible) not a single scalar but two scalars, one for the mantissa and one for the exponent, very similar to the scientific notation. One compound that demonstrates that quite well is Calculate Orbit Velocity. If the compound calculated the orbit velocity the "conventional way" then the output values would be mainly rubbish (zero, -infinity or +infinity) due to the enormous rounding errors.

THE COMPOUNDS

All emNewton2 compounds can be accessed in the ICE Tree viewer.
They are located in **"Task -> Mootzoid - Newton2"**.

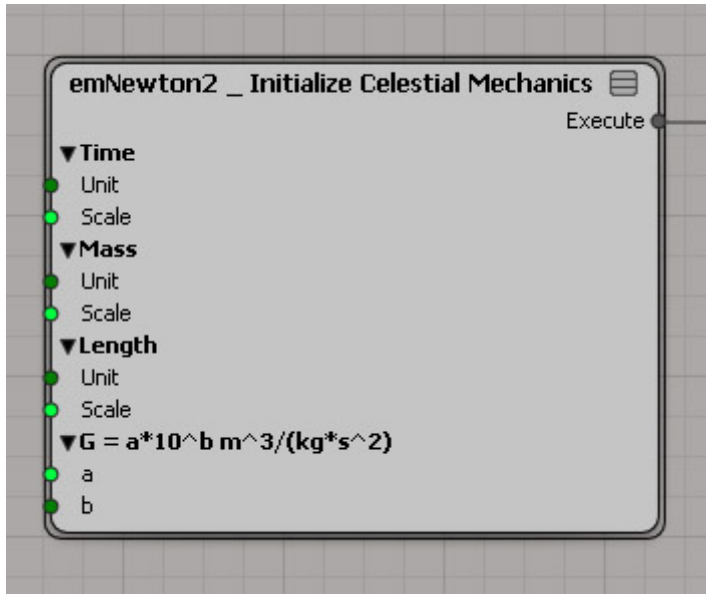
*Tip: all compounds contain the word "emNew**ton2**" in their names, so simply enter "**ton2**" in the search field to see all the available compounds.*

Initialize Celestial Mechanics

This compound must be used in all emNewton2 setups. It initializes the emNewton2 physics and the units of time, mass and length.

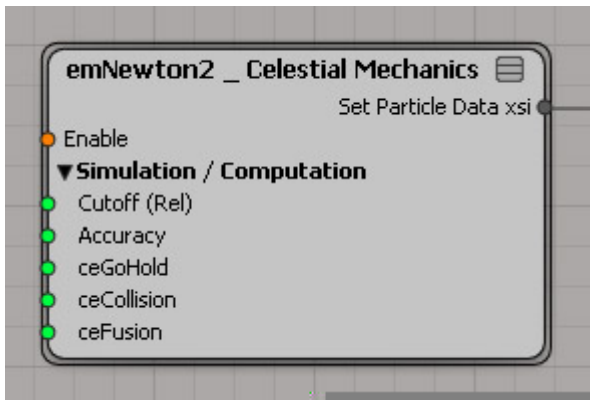
In general one could say that emNewton2 does "its own thing" internally. However it is required to define what exactly "1 Softimage Unit" represents so that emNewton2 will be able to make the correct conversions.

It should be used once in an ICE Tree in the modeling stack, but it can also be used in an ICE Tree in the Simulation stack, for example if you want to animate the gravitational constant over time. However modifying the Time, Mass or Length unit over time is not recommended!



- *The Input Port(s) and Parameter(s):*
  - *Time*
    - **Unit**
      Defines how much physical time elapses in one Softimage second.
    - **Scale**
      An additional scaling factor for the above unit. If for example you would like to work with a time unit of "1 month" then you could set "Unit" equal "Week" and "Scale" equal "4".
  - *Mass*
    - **Unit**
      Defines how much physical mass a particle with "1 Softimage mass" has.
      If for example you set this to "Earth Mass" and emit a particle with a mass equal "0.1" then emNewton2 will consider the particle to have a physical mass of 10% of the Earth's mass.
    - **Scale**
      An additional scaling factor for the mass.
  - *Length*
    - **Unit**
      Defines the physical length that is represented by "1 Softimage Unit".
      If for example this is set to "Astronomical Unit" (=average distance between the Sun and the Earth) and you have a particle that is located at (2, 0, 0) then emNewton2 will consider this particle to be twice the distance Sun-Earth from the origin.
    - **Scale**
      An additional scaling factor for the length.
  - *G = a*10^b m^3/(kg*s^2)*
    - **a**
      The mantissa of the gravitational constant.
      Using a negative value will produce negative gravity.
    - **b**
      The exponent of the gravitational constant.
- *The Output Port(s):*
  - **Execute**
    Plug this into an execute port.

Celestial Mechanics



This is the compound that performs the main calculations: based on the current particle data (positions, velocity, size, mass,..) it calculates and sets the new particle data.
It also deletes any particles that have a size less than or equal zero.

- *The Input Port(s) and Parameter(s):*
  - **Enable**
    Enables/disables this compound.

- *Simulation / Computation*
  - **CutOff (Rel)**
    A cutoff distance (relative to the particle size) that defines a volume around a particle in which brute force calculations are to be performed (instead of optimized calculations).
    This simply means that when particles are relatively close to each other (and contained in the cutoff distance) then accurate calculations are done.
    Increasing this value results in longer simulation times.
  - **Accuracy**
    The accuracy of the simulation.
    Increasing this value results in longer simulation times.
  - **ceGoHold**
    This is a special parameter that controls the behavior of particles that a very close to each other. When that is the case then "bad" things can sometimes happen. For example you could get some huge gravitational acceleration values and your particles just fly away with triple light speed.
    To prevent these things from happening you can define a so-called "Go-Hold" radius around a particle. When another particle enters this radius then gravity is calculated in a different way in order to prevent illegal computations and unwanted forces and velocities.
    The Go-Hold radius is relative to the particle size.
  - **ceCollision**
    The amount of collision when particles intersect (0=0%, 1=100%).
    When using fusion (see next parameter) then this value is typically smaller than 1 so that particles can penetrate themselves (for a value >= 1 there would hardly be any penetration and therefore no fusion).
    Note that you can also use values greater than 1.
  - **ceFusion**
    The amount of fusion (or melting) per second when particles intersect.
    This value represents the percentage (0=0%, 1=100%) of the intersection volume that will "flow" from the lighter particle to the heavier particle.
    Example:
    Let's assume that a small particle is entirely contained in a larger and heavier particle. Furthermore our frame rate is 25 fps and ceFusion is equal 0.5. Then the bigger particle would completely absorb the smaller particle in 2 seconds (or 50 frames). Note that in practice this might take a shorter or longer time, because both particles change their sizes during the process.

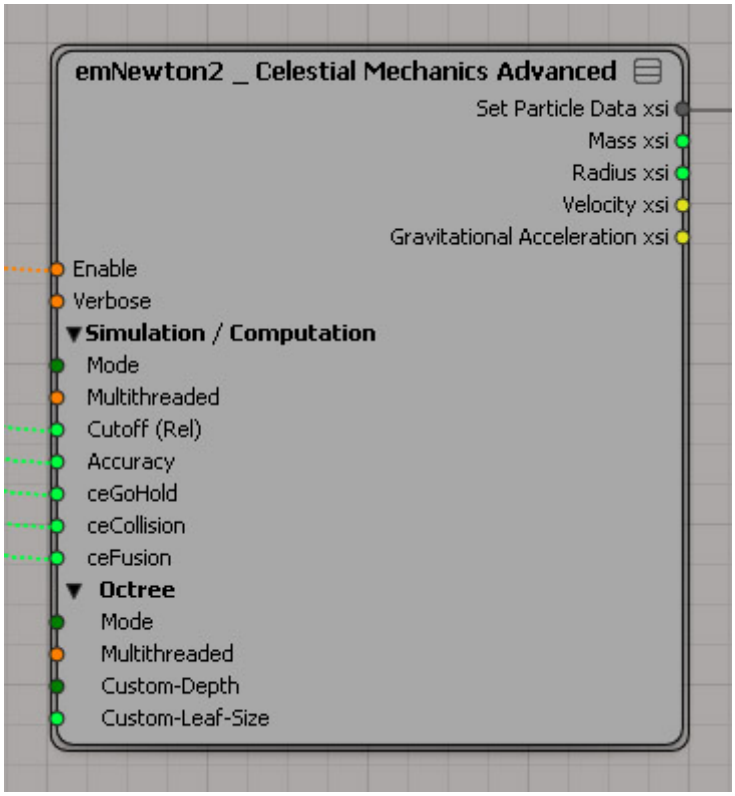
- *The Output Port(s):*
  - **Set Particle Data xsi**
    Sets the new particle size, mass and velocity.

Celestial Mechanics Advanced

This is the compound that performs the main calculations (it is embedded in the compound Celestial Mechanics): based on the current particle data (positions, velocity, size, mass,..) it calculates the new size and position as well as the gravitational acceleration for each particle. These values are available through the output ports of this compound.

- *The Input Port(s) and Parameter(s):*
  - **Enable**
    Enables/disables this compound.
  - **Verbose**
    Enable this to have some information outputted into the history log.

- *Simulation / Computation*
  - **Mode**
    The simulation mode:
  - **Octree - Runtime O(n*log(n))**
    The recommended default mode. Before any calculations are performed a special octree is created in order to speed up subsequent calculations.
    Example:
    You have a huge cluster of particles and a single particle that is far away from that cluster. To calculate the gravitational force that the cluster exerts on the single particle it is not necessary to check each particle contained in the cluster. Instead the cluster is considered a single big particle thus reducing calculation.
    In short: *accurate calculations are only performed when particles are relatively close to each other.*



  - **Brute force - Runtime O(n^2)**
    In this mode emNewton2 simply compares each particle with all the others. This is the most accurate way of calculating the gravitation force, but also the slowest. Things will take forever once the particle count reaches a certain amount, which means that this mode should only be used with small amount of particles.
    Note: emNewton2 automatically switches to this mode when the particle count is low (< 500).
  - **Multithreaded**
    Enables/disables multithreading when calculating the gravitation acceleration, the collisions, etc.
  - **CutOff (Rel)**
    See Celestial Mechanics.
  - **Accuracy**
    See Celestial Mechanics.
  - **ceGoHold**
    See Celestial Mechanics.
  - **ceCollision**
    See Celestial Mechanics.
  - **ceFusion**
    See Celestial Mechanics.
- *Octree*
  - **Mode**
    The octree mode:
    - Automatic
    - Use Custom Depth
    - Use Custom Leaf Size

    The automatic mode is the default mode and well suited for most scenarios. However it sometimes can be necessary to use a fixed (custom) depth or leaf size, for example with very large setups that have several millions of very small particles.
    As a general rule one can say:

a.) The bigger the leaf size the less memory is used (but calculations can take longer).
b.) The smaller the depth the less memory is used (but calculations can take longer).

- **Multithreaded**
  True to use multithreading when building the octree.
- **Custom-Depth**
  The custom depth of the octree.
  This is only used if mode is set to "Use Custom Depth".
  Note: bigger depth values require more memory.
- **Custom-Leaf-Size**
  The custom leaf size of the octree.
  This is only used if mode is set to "Use Custom Leaf Size".
  Note: smaller leaf sizes require more memory.

- *The Output Port(s):*
  - **Set Particle Data xsi**
    Sets the new particle size, mass and velocity.
  - **Mass xsi**
    The new particle mass.
  - **Radius xsi**
    The new particle radius (= particle size).
  - **Velocity xsi**
    The new particle velocity.
  - **Gravitational Acceleration xsi**
    The gravitational acceleration in Softimage units per square seconds.

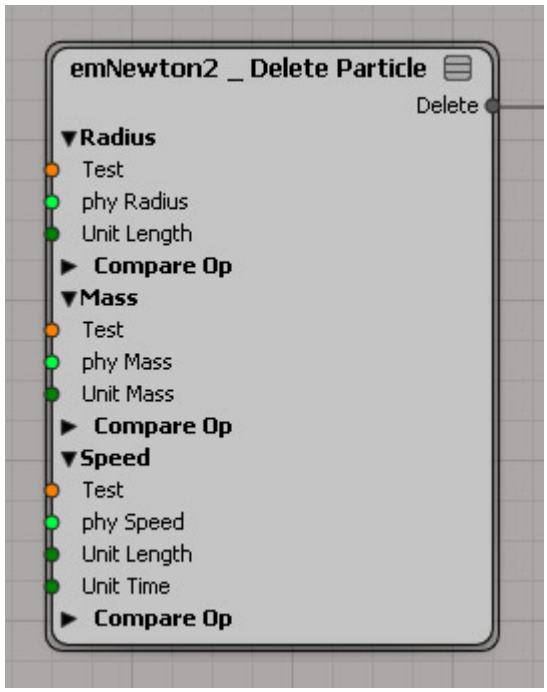
Delete Particle

Tests a particle's size and/or mass and/or speed and possibly deletes it.

All values are defined via the typical scalar and unit, so that one can for example easily delete particles that are...

"...smaller than 250 metres or lighter than 0.1 metric tonnes or faster than 10% of the speed of light".

Deleting unwanted particles is something one generally wants to do, especially when using particle fusion (melting), because often some teensy-weensy particles will remain. These little particles do no harm but they increase the simulation time.



Apply Acceleration

Sets the velocity of a particle equal the input velocity plus the input acceleration.

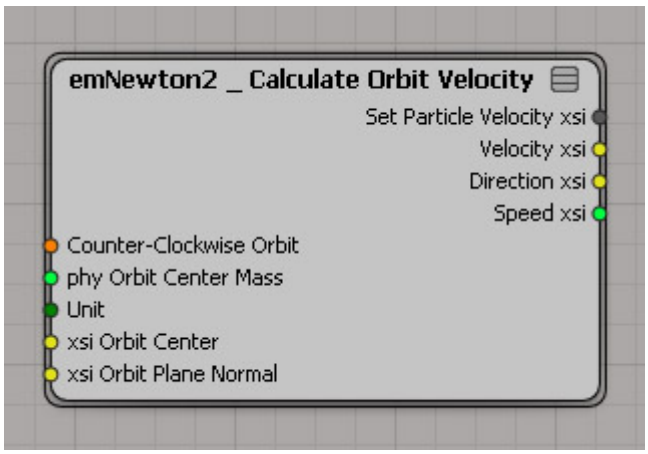
- *The Input Port(s) and Parameter(s):*
  - **xsi Velocity**
    The Softimage velocity.

- **xsi Acceleration**
  The Softimage acceleration that is added to the above velocity.

- *The Output Port(s):*
  - **Set Velocity xsi**
    Plug this into an execute port.

Calculate Orbit Velocity

Use this compound to calculate the orbit velocity for a particle.

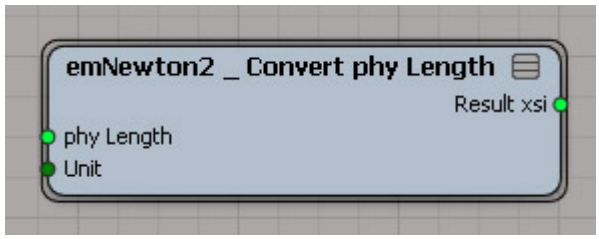


This compound is typically used when creating particles, but it can also be used differently. For example one could calculate the orbit velocity of a particle and compare it to the current particle's velocity and - if they differ to much - correct the particle velocity a bit. This is a nice and easy way of preventing particles to "go wild".

- *The Input Port(s) and Parameter(s):*
  - **Counter-Clockwise Orbit**
    True to orbit counter-clockwise.
  - **phy Orbit Center Mass**
    The mass of the center around which the particle will orbit.
  - **Unit**
    The physical unit of the orbit center mass.
  - **xsi Orbit Center**
    The orbit center.
  - **xsi Orbit Plane Normal**
    The plane normal of the orbit plane.
- *The Output Port(s):*
  - **Set Particle Velocity xsi**
    Sets the particle velocity equal the orbit velocity.
  - **Velocity xsi**
    The orbit velocity.
  - **Direction xsi**
    The normalized direction vector of the orbit velocity.
  - **Speed xsi**
    The orbit speed.

Convert phy Length

Converts a physical length (e.g. "0.14 AU") into Softimage units.



- *The Input Port(s) and Parameter(s):*
  - **phy Length**
    The length.

- **Unit**
  The length's unit.

- *The Output Port(s):*
  - **Result xsi**
    The result in Softimage units.

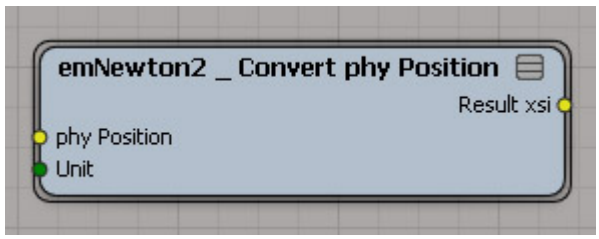
Convert phy Mass

Converts a physical mass (e.g. "7.1 kg") into Softimage units.

- *The Input Port(s) and Parameter(s):*
  - **phy Mass**
    The mass.
  - **Unit**
    The masses unit.

- *The Output Port(s):*
  - **Result xsi**
    The result in Softimage units.

Convert phy Position



Converts a physical position into Softimage units.

- *The Input Port(s) and Parameter(s):*
  - **phy Position**
    The position.
  - **Unit**
    The position's (or length's) unit.

- *The Output Port(s):*
  - **Result xsi**
    The result in Softimage units.

Convert phy Speed

Converts a physical speed (e.g. "600 km/s") into Softimage units.

- *The Input Port(s) and Parameter(s):*
  - **phy Speed**
    The speed.
  - **Unit Length**
    The speed's length unit.
  - **Unit Time**
    The speed's time unit.

- *The Output Port(s):*

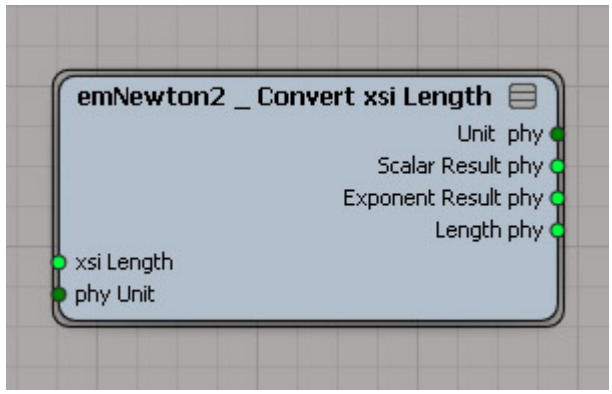
- **Result xsi**
  The result in Softimage units.

Convert phy Time

Converts a physical time (e.g. "3 days") into Softimage units.

- *The Input Port(s) and Parameter(s):*
  - **phy Time**
    The time (or duration).
  - **Unit**
    The time's unit.
- *The Output Port(s):*
  - **Result xsi**
    The result in Softimage units.

Convert xsi Length



Converts a Softimage length into a physical length.

- *The Input Port(s) and Parameter(s):*
  - **xsi Length**
    The Softimage length to be converted.
  - **phy Unit**
    The unit into which the Softimage length is to be converted.
- *The Output Port(s):*
  - **Unit phy**
    The unit of the results (this is equal the "phy Unit" input parameter).
  - **Scalar Result phy**
    The mantissa of the result.
  - **Exponent Result phy**
    The exponent of the result.
  - **Length phy**
    The result (= the two above values combined = scalar * 10^exponent).

phy Length/Mass/Time

These compounds are a pass-through for physical length, mass and time.

- *The Input Port(s) and Parameter(s):*
  - **phy Length/Mass/Time**
    Amount.
  - **Unit**
    Physical Unit.
- *The Output Port(s):*

- **phy Length/Mass/Time**
  Amount.
- **Unit**
  Physical Unit.

Set Size From Mass

This compound sets the size of a particle depending on its mass and the input mass-size interval.
Being able to specify not one but two mass-size pairs enables one to make objects denser (=more mass per volume) when they have lots of mass.

Example:
Let's say one creates a simulation that involves a cluster of 10,000 suns that orbit around the cluster's barycenter. Fusion is enabled and as the suns intersect they melt together to produce bigger and heavier suns. But the heavier a sun gets the denser it should get.
With this compound you can do exactly that. You can define a radius (or size) for the not-so-heavy suns and another radius for the heavy suns.

- *The Input Port(s) and Parameter(s):*
  - *Start*
    - **phy Mass**
      The mass.
    - **Unit**
      The masses unit.
    - **phy Length**
      The length.
    - **Unit**
      The length's unit.
  - *End*
    - **phy Mass**
      The mass.
    - **Unit**
      The masses unit.
    - **phy Length**
      The length.
    - **Unit**
      The length's unit.

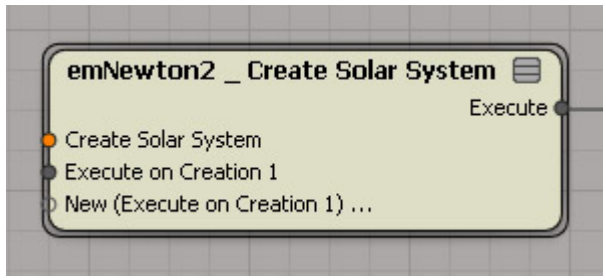
- *The Output Port(s):*
  - **Execute**
    Plug this into an execute port.

Test phy Length/Mass/Time

These compounds compare the particle's current physical values with the input values.
One could for example test if a particle is "smaller than 100 metres" or "heavier than the Sun".

- *The Input Port(s) and Parameter(s):*
  - **phy Length/Mass/Time**
    The value.
  - **Unit**
    The unit of the above value.
- *The Output Port(s):*
  - **Result**
    The result of the test.

Create Solar System



Creates a particle for each solar system object that is contained in the database.

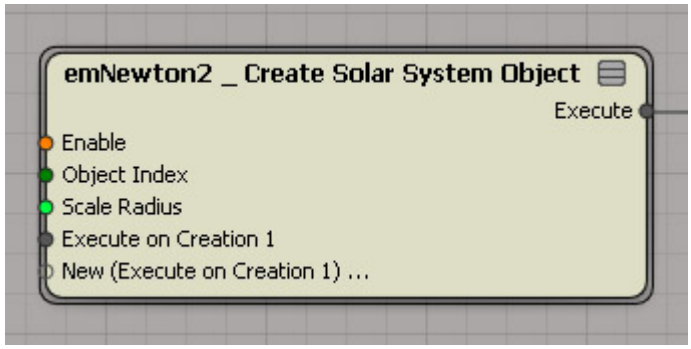
Note: please read
The Solar System and HORIZONS before using this compound.

- *The Input Port(s) and Parameter(s):*
  - **Create Solar System**
    Enables/disables the compound.
  - **Execute on Creation**
    Plug any additional compounds in here, for example a "Set Particle Color".
- *The Output Port(s):*
  - **Execute**
    Plug this into an execute port.

Create Solar System Object

Creates a particle based on the index of a solar system object.

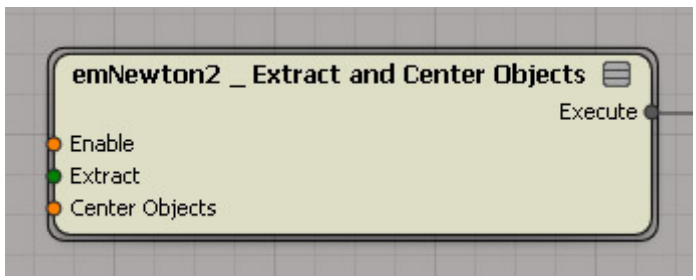
Note:
please read The Solar System and HORIZONS before using this compound.



- *The Input Port(s) and Parameter(s):*
  - **Enable**
    Enables/disables the compound.
  - **Object Index**
    The solar system object index.
  - **Scale Radius**
    A scale that is applied to the freshly created particle. Our solar system is really very big and the planets and moons contained in it are, in relation, extremely small, so most of the time you will increase the size in order to see something.
  - **Execute on Creation**
    Plug any additional compounds in here, for example a "Set Particle Color".
- *The Output Port(s):*
  - **Execute**
    Plug this into an execute port.

Extract and Center Objects

This compound extracts a given set of objects (e.g. Jupiter and its moons) from the solar system.
Typically you create and simulate the entire solar system and use this compound in a Post-Simulation ICE Tree, as for example in the setup that can be created here:



Get -> Primitive -> emNewton2 -> Solar System (Post-Extracted)

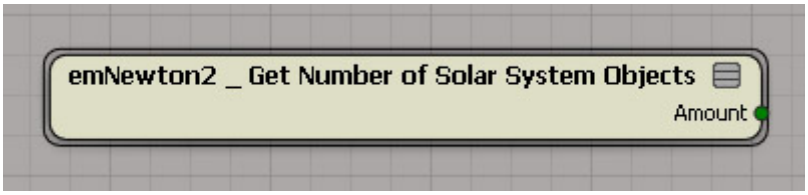
Note:
please read The Solar System and HORIZONS before using this compound.

- *The Input Port(s) and Parameter(s):*
  - **Enable**
    Enables/disables the compound.
  - **Extract**
    Defines what shall be extracted.
  - **Center Objects**
    If enabled then the extracted objects are moved in such a way that the barycenter (=center of mass) lays in the world's origin.
- *The Output Port(s):*
  - **Execute**
    Plug this into an execute port.

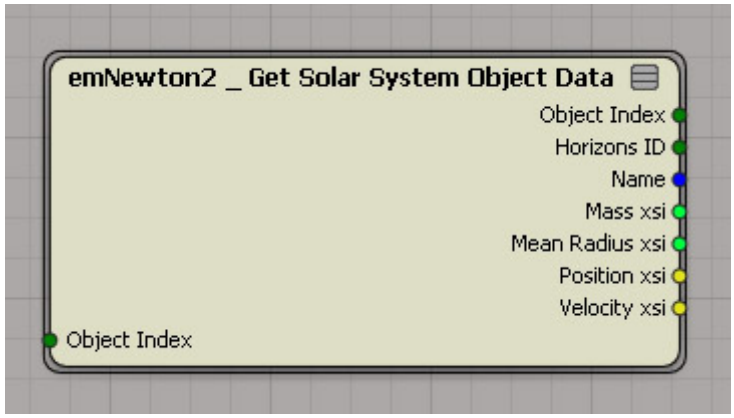
Get Number of Solar System Objects

Returns the amount of elements that are contained in emNewton2's solar system database.

Note:
please read The Solar System and HORIZONS before using this compound.



Get Solar System Object Data

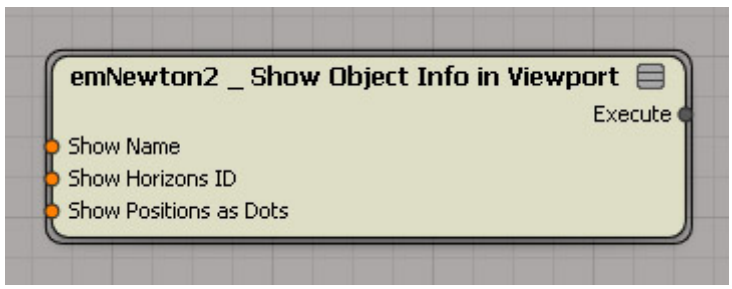


Outputs the position, velocity, mass, etc. of a solar system object, based on the input index.

Note:
please read The Solar System and HORIZONS before using this compound.

- *The Input Port(s) and Parameter(s):*
  - **Object Index**
    The index of the solar system object.
- *The Output Port(s):*
  - **Object Index**
    The index of the solar system object (this has the same value as the input parameter "Object Index").
  - **Horizons ID**
    The HORIZONS object ID.
    For example Earth has the ID 399.
  - **Name**
    The official name of the object.
  - **Mass xsi**
    The mass (in Softimage units). It is possible to get the physical mass by entering the compound.
  - **Mean Radius xsi**
    The object's mean radius (or size) in Softimage units. It is possible to get the physical radius by entering the compound.
  - **Position xsi**
    The position (in Softimage units). It is possible to get the physical position by entering the compound.
  - **Velocity xsi**
    The velocity (in Softimage units). It is possible to get the physical velocity by entering the compound.

Show Object Info in Viewport



This compound is used in a Post-Simulation ICE Tree to display some information in the viewports.

Note:
please read The Solar System and HORIZONS before using this compound.

- *The Input Port(s) and Parameter(s):*

- **Show Name**
  Display the object's name.
- **Show Horizons ID**
  Display the HORIZONS ID.
- **Show Positions as Dots.**
  Display the particle positions as dots.
  Note: when creating the solar system one cannot fail to notice that it is *very big* and *very empty*! The objects and thus the particles are very, very small. Because of their small sizes they are often not visible in the viewports. By enabling "Show Positions as Dots" you will see the objects no matter how small they are.

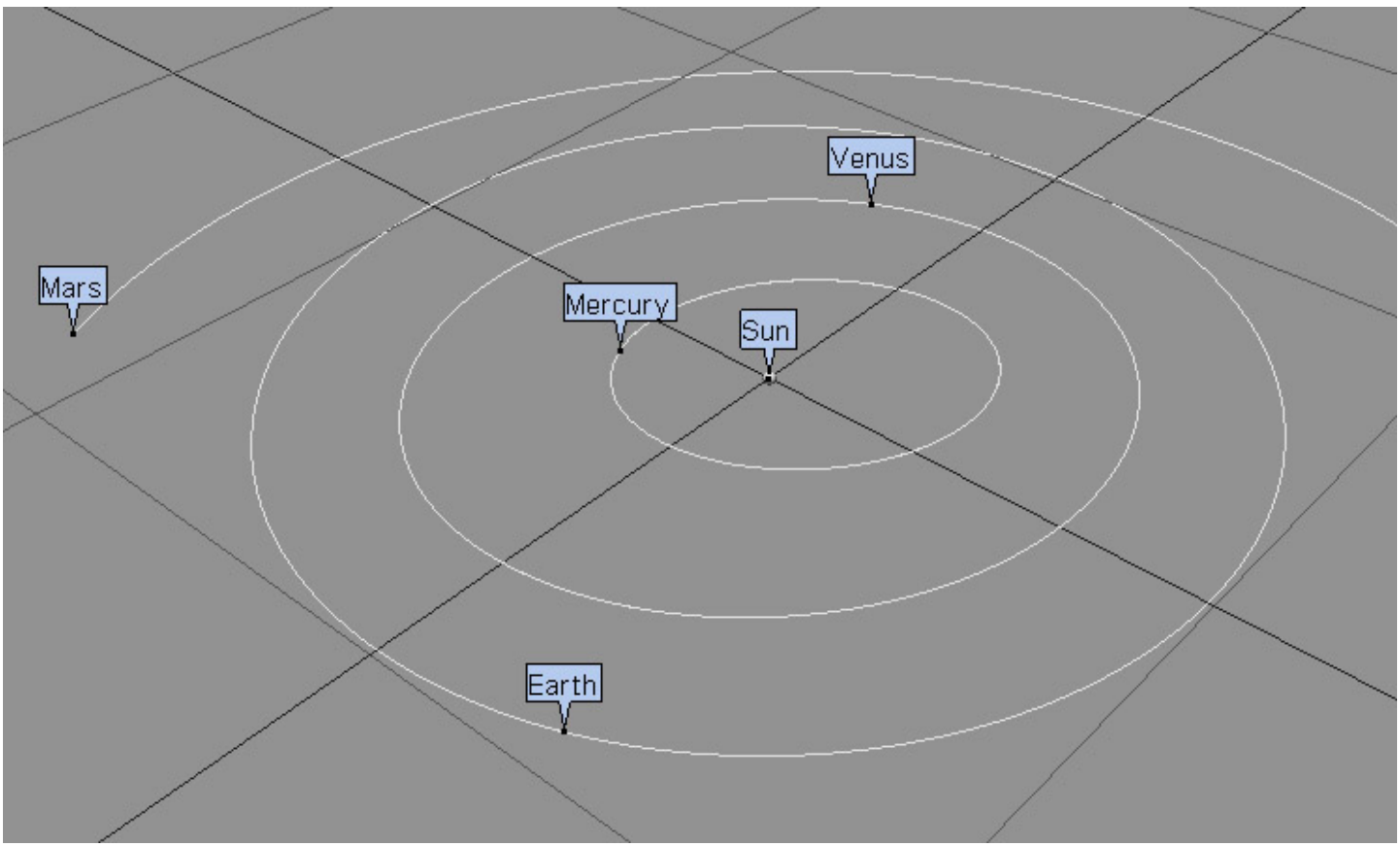
- *The Output Port(s):*
  - **Execute**
    Plug this into an execute port.

The Solar System and HORIZONS

emNewton2 has a little database that contains some of our solar system's objects. The data of the Sun, Earth and its Moon as well as the other planets and some of their moons is accessible through the compound Get Solar System Object Data.
Other compounds let you create particles for each solar system object contained in the database or you might just create a point cloud with all objects contained in the database by doing the following (note: you will have to increase the *Subframe Sampling* in the point cloud's *Simulation Settings* in order to get an accurate simulation of the smaller objects like Mars' moon Phobos, Jupiter's moon Io or nearly all of the moons from Saturn):

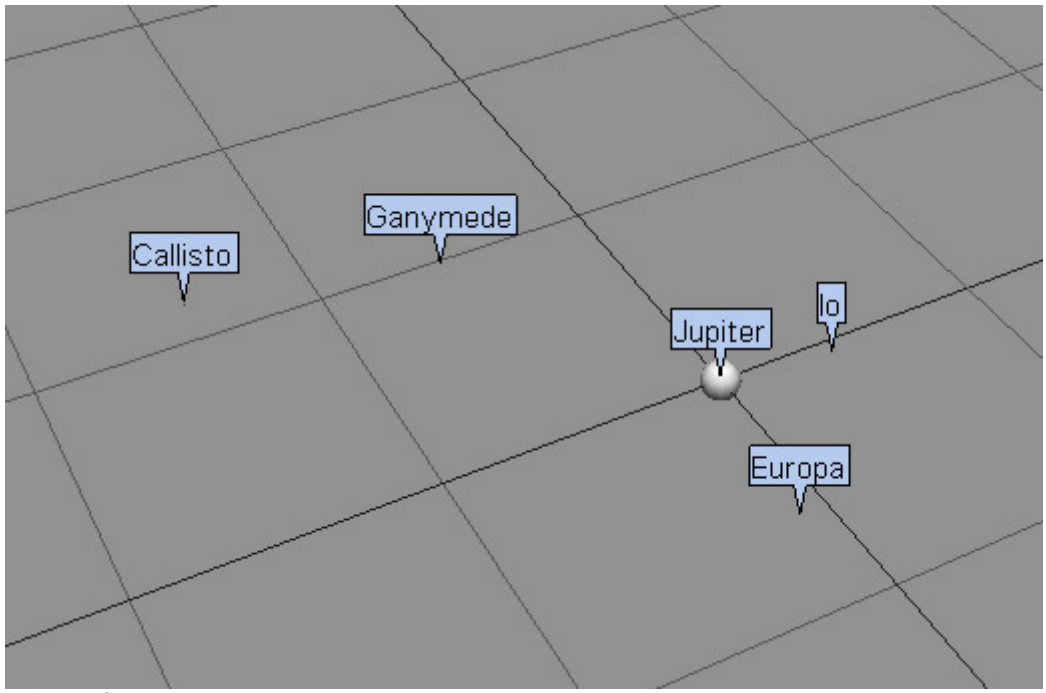
Get -> Primitive -> emNewton2 -> Solar System (Complete)

Here an example of the Sun and the inner planets Mercury, Venus, Earth and Mars.



On the right is a further example showing Jupiter with the Galilean moons.

The data for the database was created using the telnet and email interface of HORIZONS solar system data and ephemeris computation service

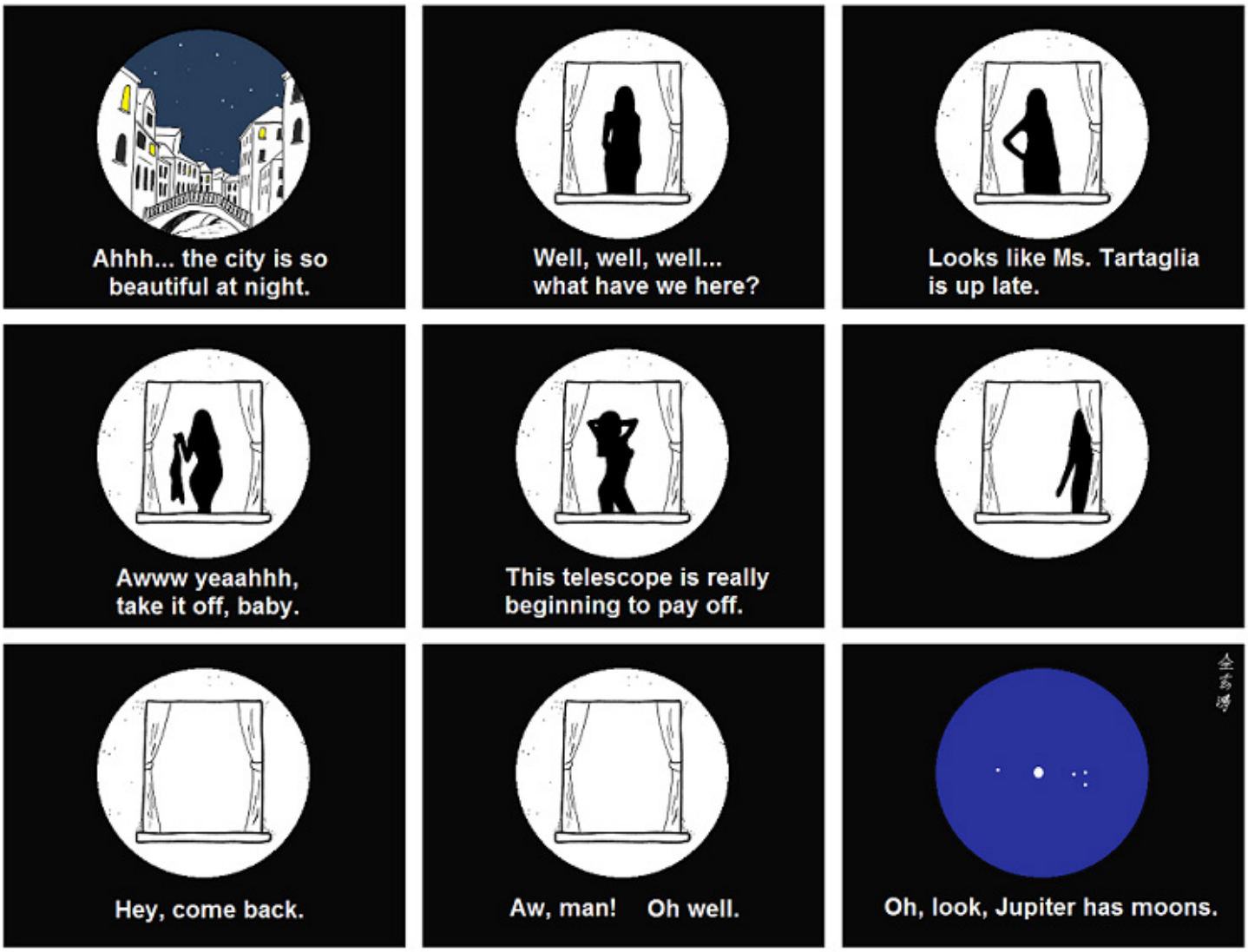


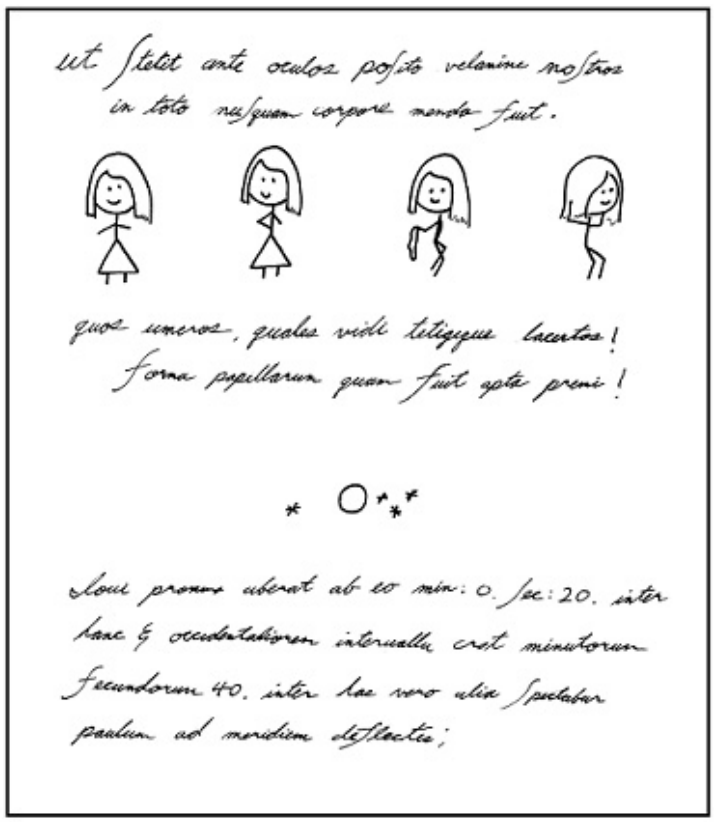
(http://ssd.jpl.nasa.gov/?horizons).
This great tool from Jet Propulsion Laboratory gives anybody access to very accurate data.

The emNewton2 database contains the positions and velocities that the object had on *the 1st of January 2001*.

It must be said that emNewton2 is by no means as accurate or elaborate as for example Chris Laurel's Celestia (http://www.shatters.net/celestia/), Volker Springel's GADGET (http://www.mpa-garching.mpg.de/galform/gadget/index.shtml) or Vladimir Romanyuk's Space Engine (http://en.spaceengine.org/).
But it is much fun, especially because it is available in ICE!

Speaking of Galileo Galilei:
Here is a great web comic from **"Abstruse Goose"** (http://abstrusegoose.com/) called **Rear Window** (Galileo's lost notebook) (http://abstrusegoose.com/414)





Galileo's lost notebook

TIPS AND TRICKS, TROUBLE SHOOTING

- **Tips and Tricks:**
  - *Don't use the Softimage* **"Simulation Root"** *compound***.**
    It is recommended to not use the above compound.
  - **Octree's memory usage.**
    The one thing that uses most of the memory is the octree. Its default mode is "Automatic". This mode is well suited for most scenarios, but when using really large amounts of particles (more than a few million) it can be wiser to use a fixed leaf size.
- **Trouble Shooting:**
  - *KNOWN ISSUE (Softimage 2011 only): the names of the solar system objects do weird things.*
    CAUSE: there is some sort of bug in Softimage 2011 SP1 and 2011 SAP. Sorry, there is nothing I can do about it. You might consider switching to SI 2012 SP1 or above.
  - *KNOWN ISSUE (***"emTools _ Create 3D Point Primitive"***): Softimage crashes when creating millions of particles.*
    CAUSE: unfortunately there is a bug in the factory Softimage node "Sort Array with Key".
    This bug results in crashes when using the "Filter by Sphere/Cylinder/Profile" feature together with high amount of particles (around 32 million).
    This bug has been reported and Softimage will hopefully fix it for the next version (Softimage 2013).
  - *PROBLEM: most of the emNewton2 compounds are red.*
    PROBABLE CAUSE: You forgot to add the Initialize Celestial Mechanics compound.
  - *PROBLEM: I have my solar system all right, but the moons keep flying away!*
    SOLUTION: You need to either increase subframe sampling or to use a smaller time unit in the Initialize Celestial Mechanics compound (the default is "Week", so you might try "Day" or even "Hour").
  - *PROBLEM: the compounds somehow don't work at all.*
    PROBABLE CAUSE: You forgot to install the most recent emTools (../../plugin/emtools.html) addon.
  - *PROBLEM: emNewton2 uses massive amounts of RAM... too much!*
    PROBABLE CAUSE: You have many *very small* particles (several million).
    SOLUTION: Set the octree mode from "Automatic" to "Use Custom Depth / Leaf Size".
    Using a custom depth of 8 or 9 should definitely solve the problem.

VERSION HISTORY

- *New in Version 2.000:*
  - new and specialized multithreaded octree for quick neighboring and calculations.
  - fusion (or melting) of intersecting particles.
  - compounds to automatically calculate the orbit velocity.
  - compounds for converting between real physical values and Softimage units.
  - database with real physical values of planets and moons of our solar system.
- *New in Version 2.100:*

- now supports the new "emBatch" render node licensing.
- the addon is now built separately for Softimage 2012, 2013, etc. using the respective SDK.

- *New in Version 2.150:*
  - the addon is now available for Softimage 2012, 2013, 2014 and 2015.
  - now using the newest mootzoid libs.
  - a few minor bug fixes.
- *New in Version 2.180:*
  - a few changes and optimizations in the internal octree class.
  - now uses RLM v.11.2.
  - revised multithreading.
  - the version for Softimage 2015 no longer has a memory leak (due to a bug fix in the Softimage 2015 ICE SDK).
- *New in Version 2.200:*
  - the plugin is now freeware and no longer requires a license.
  - [Softimage] Linux is no longer supported.

LIMITATIONS AND REMARKS

- emNewton2 is only available for Softimage 64 bit (Windows).
