PHY Software Documentation

By Kovács Hunor (group 3022, 2009.02.09)

## Overview

*PHY* is a simulator program that describes the action, behavior and motion of different types of physical elements: bodies, force fields and spaces. Using the software, one can easily study the effects created by systems of forces that are hard to understand in the real world.

Points like objects are drawn on the screen into a frame of reference and are animated properly as it would happen in real, in the same conditions. Different types of forces can be cast to act on these points, like homogeneous force fields (similar to gravity on Earth from close range); radial force fields (similar to gravity of the Sun from far range) either with homogeneous, linear or quadratic magnitude; vortices (simultaneous circular and attracting forces); drag forces (resisting forces in function of speed) and so on. One can also create spaces, defining their position and shape, which isolate everything lying in them, setting a limit to specific forces whose point of origin is positioned inside the given area.

The user can easily play the simulation back several times by pressing the Play, Pause and Stop buttons, or the user can pause the states of all objects whenever he likes to. In addition, a time speed slider provides a solution for slowing down or speeding up the animation with hundreds of times slower or faster allowing us to observe very fast events such as turbulences, oscillations and explosions.

## Display

The application incorporates a toolbar, a drawing panel and a menu area. An edit window serves the editing and creation of an object.

### The toolbar

The toolbar is on the top, displaying information about the timing, and possessing the two playback buttons: the Play and Stop button. The first value is the time speed multiplier, it shows us how faster flows the simulation than in real.

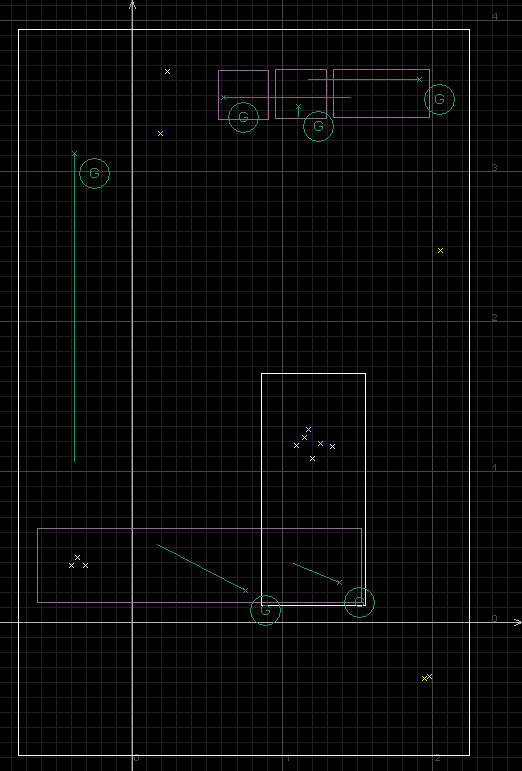
toolbar.jpg

The second value is the amount of time elapsed since the simulation has started measuring with time speed that was described previously.

The play button starts the simulation and changes into a pause button. Pressing again, it pauses the simulation process and inverts back to play button.

The stop button resets everything, independently of the playback’s state, the simulation stops, and the objects will be drawn back to their initial state, when they were declared or last edited by user. Whenever the simulation is on, all other buttons in the menu area are disabled, forbidding the user to create or edit objects while playing.

### The drawing panel

The drawing panel is the main part of the program, because this is where the objects are displayed and where the whole simulation meets the observer. On a black background we can see a frame of reference, a pair of white axes, a grey grid while numbers, coordinates appear on the bottom and on the right side. One can easily move in the system by dragging the canvas with the mouse in the desired direction. Zooming is also possible using the mouse scroll. When the view is sufficiently zoomed up, gridlines may become thin. By zooming in a bit, a new set of gridlines will appear with smaller order of magnitude. This makes hard to get lost, in addition the actual precise coordinates will always be there on the sides.

In this viewport will appear all our objects. For example a *PointlikeBody* is represented by a small yellow X, and a *RectangularIsolatorSpace* is represented by a purple rectangle with a size corresponding with the attributes of the space.

### The menu

The menu area placed on the left side holds various user controls. The first on the top is a slider for adjusting time speed. It sets values exponentially leaving the user to choose between detailed decimal values around one and leaving the user to make hundred values reachable in the same scale in the same time. Using this we can slow down our simulation to one thousand times slower.

The second in line is a list containing the names of placed objects. By selecting one or more of them with the Shift or Ctrl button, they become highlighted in the viewport too, shown as white instead of their own color. The selected elements can be deleted or edited with the buttons below the list. Note that the edit button will always refer to the first selected element on the list, because only one editing window is allowed at a time.

At last on the bottom we have the creation area where we can choose between buttons named by the types of objects they create. Pressing them makes an edit window to pop up.

### Popup edit windows

In the popup window we can declare the initial state of our object. For faster declaration use the mouse’s right button, click, hold and drag the drawing canvas to specify the main attributes. The corresponding values in numbers will immediately appear in the small window’s text fields.

## Elements

In this first version of this program there are only tree objects to create. Each of them represents one from the tree main types: bodies, forces, spaces. They are the *PointlikeBody*, the *GravityField* and the *RectangularIsolatorSpace*.

### PointlikeBody

A *PointlikeBody* possesses four attributes. It has mass, position, speed and a name. It has no dimension.

In the viewport it is represented by a small yellow X.

When editing, by clicking the right mouse button the user sets the position of the body and by dragging one can set the velocity vector relative to the position.

### GravityField

A *GravityField* is a homogeneous force field, which acts on all objects having a mass greater than zero, with a constant magnitude m\*g, and a constant direction. It has mass, position, speed, name, and constant g as attributes.

In the viewport it is represented by an X like a *PointlikeBody* and a line representing the vector g. There is also a circled capital letter G near the small X. Its color is green.

When editing, by clicking with the right mouse button the user sets the position and by dragging the user can set the g vector.

### RectangularIsolatorSpace

A *RectangularIsolatorSpace* is a finite area of a rectangular shape. It isolates forces that are inside and outside. A body in a set of spaces like this will be affected only by forces that are in the same set of spaces.

Note that in case of two nested spaces, points in the inner space will be affected only by forces in the inner space, and points in the outer space will be affected only by forces in the outer space without the effect of the inner space’s forces.

In case of two intersecting spaces, the intersection is treated as a separate third space, and because it is bounded on all sides, the forces that are positioned in the intersection area will affect points only there.

Points in a certain set of spaces are affected by forces positioned in the same set of spaces.

In the viewport it appears as a purple rectangle.

When editing, by clicking the mouse, the user sets the position, and by dragging sets the dimension.

## Operation

Like many other programs this one is built on the Model – View – Controller pattern. The objects we can create and a class that connects theese to each other (namely the *PHYModel*) take place in the model part. In the view part takes place the main window *PHYView*, the special type of panel it uses, the *DrawPanel*, and the small editing windows. The controller is made up by the *PHYController* class that controls the main window, some classes extending the *EditXController* that controls the small editing views and a *Runner* class that provides the command of drawing the animation.

### Threadding

The objects, we create by pushing the create buttons, are all separate threads. Each one of them has its own goal: calculating its own actual position and speed from its previous position, speed and from forces acting on it from time to time.

For calculating the position the object needs to know its speed and time elapsed from the last calculation. Let’s say, my *PointlikeBody* was sleeping for 50 milliseconds at the position p= {10; 10} m with a velocity vector v= {1; 0} m/s. In the next cycle it determines its new position by adding v\*t meters to it, its new position resulting at {10.05; 10} m. Velocity is a function of acceleration and time, and acceleration is a function of force and mass. So if our *PointlikeBody* weighs 5 kilograms and was in a space where a gravitational force acts with g= {0; -10} N\*kg, after 50 milliseconds wait we get an acceleration of a= {0; -10} m/s2 and a new velocity vector v= {1; -0.5} m/s.

Classes *PointlikeBody*, *GravityField* and *RectangularIsolatorSpace* are extensions of the *Thread* class, and in their *run()* method they use their *calculateNewPosition(double, Dimension)*, *calculateNewVelocity(double, Dimension, Dimension)* and *calculateActingForces(Dimension)* methods to assign new values for their position and velocity. After that they go to a sleep and when the sleep is over everything starts again.

### MVC pattern

The *Runner* class is also an extension of *Thread*, and in its *run()* method it calls for redrawing the *DrawPanel* of the *PHYView* from time to time. Based on the sleep time t in milliseconds, we get a 1000/t frame per second animation.

*PHYControl* and other control classes transfer button pushes or other changes that the user makes to the *PHYModel* .

*PHYView* and other views call for *get()* functions in the *PHYModel* and this way display information on the screen.

## Simulation vs. Calculation

The main purpose of *PHY* is to simulate physical phenomena. There is a big difference between simulation and calculation.

By calculation, a program could determine the path of an object, and by replacing the time variable with a given value in the resulting formula, it could calculate the precise position of that point in that given time. The process requires no playback, it is much more difficult to calculate, in some cases it could even be impossible, but the result is more accurate.

By simulation, a program determines the position of an object, by calculating a sequence of very small displacements and adding them together. This process requires a playback. There is no possibility of jumping to different times. Each step is followed by another one, and the combination of these is the result. Steps are easy to determine from previous steps, but the results are not as accurate as the calculation method, because of the rounding in continuous additions.

*PHY* is a simulator. The timer in the toolbar is completely independent of the moving objects and the timer is used for measuring. The objects are also independent of each other: there is no restriction that under the action of the same forces two *PointlikeBody*s in the same spaces should move identically.

Some calculation errors may come up when the animation is too fast. During a cycle of a thread a given amount of time passes. In our case I defined that to be 33 milliseconds. The object assumes that the conditions during that sleep time are constant. What if during that time some new force would start to act on the body? It will observe that later, when the sleep time passes, and maybe too late. Its speed does not change in the right time, and in the following, its position will not be precisely correct. Some ‘perpetum mobile’ cases may collapse easily if simulation is played back too fast or is not slowed down to a clearly visible continuous animation.

## Extensions

The first version holds only the three basic physical elements. There are no radial force fields, dampening forces and neither are different shaped spaces, but the program is created in a way that it can be extended very easily. Adding other types of elements requires some work, but the basic idea stays and the simulator will work the same way.

New types of forces could be introduced, as previously mentioned in the first paragraph of the Overview, by only extending the class PointlikeBody and defining an act() function depending on how the force acts on bodies and in function of which properties.

New types of spaces can be made, for example a circular or oval space, to be able to isolate not only rectangular areas.

New types of bodies could be defined, for example a *Hair* type which is made of a chain of *PointlikeBody*s and can bend in any direction maintaining its length.

## Post scriptum

I hope you like it.