User Guide

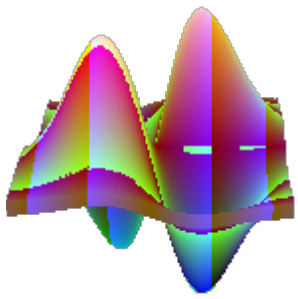
For:

Rhyscitlema Graph Plotter 3D

By:

Rhyscitlema

http://www.rhyscitlema.com/applications



15 April 2017

Contents

1 General Information

Rhyscitlema Graph Plotter 3D is an application to draw any graph in a 3D virtual space, and with everything **fully** defined in text. A graph is a 3D object in space, having a position and a direction of orientation. It is viewed through a virtual camera which is yet another object in space.

Everything about an object is defined in a single block of text, through variables and functions defined with their associated mathematical expressions, in a text format called Math Function Expression Text (MFET). Multiple objects are defined collectively, in a block of text called Rhyscitlema Objects Definition Text (RODT).

Currently, an object can be:

- a Surface: this is the graph of an equation of a surface given in the form f(x,y,z)=0.
- a Camera: this is a plane-surface used to view the Rhyscitlema 3D virtual space.

By drawing appropriate graphs, colouring them with the appropriate image files, and making them to vary with time, the software can be used for more than just graph plotting. It can be used to design, develop, view and use complex systems like cars, houses, roads and even classic games.

Important: everything is based on the Math Function Expression Text (MFET), which is only explained in the user guide for the Rhyscitlema MFET Calculator software, available at http://rhyscitlema.com/applications/mfet-calculator. This must be followed and understood first.

For now only technical information is provided.

Rhyscitlema products are provided for free, under the simple usage agreement: **use at your own risk**. In order to encourage improvement as well as release of more content you may please consider donating at http://rhyscitlema.com/donate.

2 Rhyscitlema Objects Definition Text (RODT)

An RODT is simply a block of text containing a collection of objects definitions.

Every object is defined as an MFET inner-container found inside an RODT parent-container. Therefore essentially, everything is an MFET container. It is necessary to first understand the MFET language, which is explained in details in the User Guide for Rhyscitlema MFET Calculator.

2.1 The RODT *base* container

There are currently two types of objects: Camera and Surface. An object is defined simply by defining an MFET container that inherits directly or indirectly from a corresponding *base* container, the later which has already been defined inside the RODT *base* container as show below:

```
0;
name = "Rhyscitlema_Objects_Definition_Text" ;

private
message = "
        left-click on a graph

    or  right-click on a camera

 More at http://rhyscitlema.com/applications
";

private
\mfet{0;
   name = "Object";
   origin =  (0,0,0) ;
   axes =  (1,0,0),(0,1,0),(0,0,1) ;
}

\mfet{0;
   type = "Object";
   name = "Camera";
   rectangle = {0,0,400,300,0,0};
   zoom =  1;
}

\mfet{0;
   type = "Object";
   name = "Surface";
   function(x,y,z) =  0;
   colour(x,y,z) =  {x,y,z,1};
   boundary =  {0,1,0,1,0,1};
   accuracy =  1;
}
```

As can be noticed, the container with name “Object” cannot be inherited since it has private access.

The *message* component is optional. It must evaluate to a string. This message is shown on the user interface when there is no object currently selected by the user. It can be used so to provide some information on how to interact with the scene created by the loaded RODT.

In the 3D virtual space, a unit of length corresponds to a certain number of pixels. This is specified through a variable called **PixelSize** – the size of a pixel, through the User Interface Definition Text (UIDT). The UIDT is an MFET container loaded at the launch of the software. By default it defines PixelSize = 1/1000 units-per-pixel; so 1 unit of length equals 1000 pixels.

2.2 For Any Object

Any object in space has a position and a direction of orientation, given by the following variables:

2.2.1 origin

This is the position vector (x,y,z) of the local origin (0,0,0) of an object, given in terms of world coordinates. World coordinates refer to the coordinates with respect to global space. This is in contrasts to local coordinates, which refer to coordinates from a particular object's point of view.

Below is an example that specifies the origin of an object to be located at +10 units relative to the world's x-axis, +20 units relative to the world's y-axis, and -10 units relative to the worlds z-axis.

- origin = (10, 20, -10) ;

2.2.2 axes

This is a 3x3 matrix for which:

- The 1st row is the direction vector (x,y,z) of the local x-axis (1,0,0) of an object
- The 2nd row is the direction vector (x,y,z) of the local y-axis (0,1,0) of an object
- The 3rd row is the direction vector (x,y,z) of the local z-axis (0,0,1) of an object

Below is an example that defines the local x-axis of an object to point to the world's negative x-axis, the local y-axis to point to the world's z-axis, and the local z-axis to point to the world's y-axis.

- Axes = (-1,0,0), (0,0,1), (0,1,0) ;

2.3 For a Surface Object

2.3.1 function(x,y,z)

A point (x,y,z) satisfying the equation of a graph is said to be a point on that graph. *function(x,y,z)* is the function that defines a 3D graph surface to be drawn, based on the following equation of the surface: *function(x,y,z)=0*. So a point (x,y,z) that satisfies this equation is a point on the graph.

Below is an example that will plot the graph of "*z=sin(x)*", or precisely, the graph of "*z-sin(x)=0*":

- function(x,y,z) = z-sin(x) ;

Note: even 0 is a valid function expression, in which case every point (x,y,z) of the entire 3D space will lie on the graph. Certainly the plotted graph will not be a typical "graph of a surface".

2.3.2 colour(x,y,z)

For a point (x,y,z) satisfying the equation of a graph, that is a point on the graph, this function tells the colour of that point. Therefore it determines the image (on the screen) of the plotted graph.

The result is a vector of 4 values, corresponding to the colour format {blue, green, red, alpha}, or more precisely {blue-intensity, green-intensity, red-intensity, point-opacity}. Each value ranges from 0 to 1 inclusive. A value outside this range will create undefined behaviour.

- alpha=0 => zero opacity => full transparency
- alpha=1 => full opacity => zero transparency

Note: if the colour is exactly {0,0,0,0} then the particular point on the graph is completely ignored.

Below is an example that defines the colour of a graph to be entirely green, as well as constrains the graph to be inside a cylinder of 1 unit radius:

- colour(x,y,z) = (x^2 + y^2 <= 1) ? {0,1,0,1} : {0,0,0,0} ;

2.3.3 boundary

The boundary defines the limits on the *local* x, y and z axes of a graph – it is a box that contains the graph. It is provided in the form: {lower x limit, higher x limit, lower y limit, higher y limit, lower z limit, higher z limit}. And thus the result of evaluation must be a vector of 6 values.

2.3.4 accuracy

This is an integer value that determines the accuracy in drawing a graph. It hugely affects the time taken to draw the graph. The smaller it is the better. Typical values are 1 (lowest), 10, 20, 100, 200.

The highest value is the largest difference between lower and higher boundary values, divided by PixelSize. This gives pixel-level accuracy to the plotted graph, and so any higher is unnecessary. The smaller the difference between lower and higher boundary values the smaller the accuracy value can become.

This surface accuracy feature exists due to the following unsolved/impossible Math problem:
Given a function f(x) differentiable over an interval [a, b], find if there exists, the smallest real value of x in that interval such that f(x)=0, or tell in case no such value of x exists.
There seem to be a general solution but which will take very long and unavailable time to develop!

2.4 For a Camera Object

2.4.1 rectangle

A camera appears on the screen as a rectangle with position given by its left-most and top-most coordinates on the device's screen, and with size given by its width and height. This information is provided through the 6-value vector:

- rectangle = {left, top, width, height, 0, 0}

The values are given in *number of pixels*. This is unlike any other value which is given in number of units of length. For example a camera may have “rectangle = {0, 0, 400, 300, 0, 0}” and “zoom = 0.5”. This zoom value (explained soon) corresponds to 0.5units / PixelSize = 0.5/1000 = 500 pixels.

Consider the following example:

- rectangle = (100, 100, 400, 300, 0, 0) := current + CameraResize ;

We have:

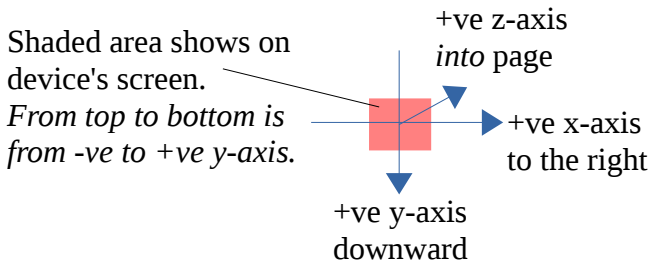
- (100, 100) is the (x,y) position on the device's screen.
- (400, 300) is the (width, height) of the camera rectangle.
- ':=' is the replacement operator (refer to the user guide for the calculator software).
- 'current' is used along with the replacement operator; it is the value on the left-hand-side.
- CameraResize is a software-defined variable for obtaining user input (more later). Here, the user input is the change in the camera rectangle upon moving and/or resizing the window.

2.4.2 zoom

A camera can be zoomed in or out. zoom-in is done by increasing the magnitude of the zoom value while zoom-out is done by decreasing it. A positive or negative value of zoom will view what is in-front or behind the camera – respectively.

2.5 Camera Orientation

It is important to note how a camera object views the Rhyscitlema 3D virtual space. The image below shows the orientation of the local x,y,z axes of a camera.



The shaded area is the rectangular portion of the camera's plane of view that is considered. This is specified by the *rectangle* vector of 6 values. It is therefore important to note that:

- The camera views what is in front of it, that is what is on its *positive z-axis*.
- The camera's y-axis goes downward. As a consequence, what is on the negative y-axis shows at the top of the rectangle (on the device's screen), and what is on the positive y-axis shows at the bottom of the rectangle.

This design of the camera orientation causes a lot of position-orientation troubles as you will notice yourself! But then since the very beginning of the development of this software, exactly which camera orientation to use has remained a very difficult design problem!

3 Obtaining and Using User Inputs

There is a group of software-defined variables for obtaining user inputs. These are used along with the replacement operator := in order to obtain and apply the user inputs.

3.1 Obtaining User Inputs

Below is the list of all software-defined variables meant for obtaining user inputs. They will always evaluate to 0 (or 0s) when not defined or not available or not set:

- CameraResize: The change in position and size of the camera window. A 6-value vector.
- CameraDistance: The distance from the selected object to the selected camera along the z-axis of the camera, scaled by the corresponding camera_zoom value. A single value.
- CameraPosition: The position (x,y,z) of the selected camera object.
- CameraAxes: The {x-axis, y-axis, z-axis} of the selected camera object. A 3x3 matrix.
- LocalPointedPoint: The point (x,y,z) on, and *local* to, the object pointed-to by the mouse.
- LocalPointedPixel: The pixel (x,y,0) on the camera window pointed-on by the mouse.
- MouseMotion: The (x,y,z) displacement of the mouse, given in number of pixels.

- Mouse_Left, Mouse_Middle, Mouse_Right
- Key_Ctrl, Key_Shift
- Key_Up, Key_Down, Key_Left, Key_Right
- Key_F1, Key_F2, …, Key_F12
- Key_0, Key_1, …, Key_9
- Key_A, Key_B, …, Key_Z

3.2 Using User Inputs

The use of user inputs is essentially based on the replacement operator '**:=**'. Actually the operator was originally developed for this purpose! Consider the following expression:

- zoom = 2 := current + Mouse_Right * MouseMotion[2] ;

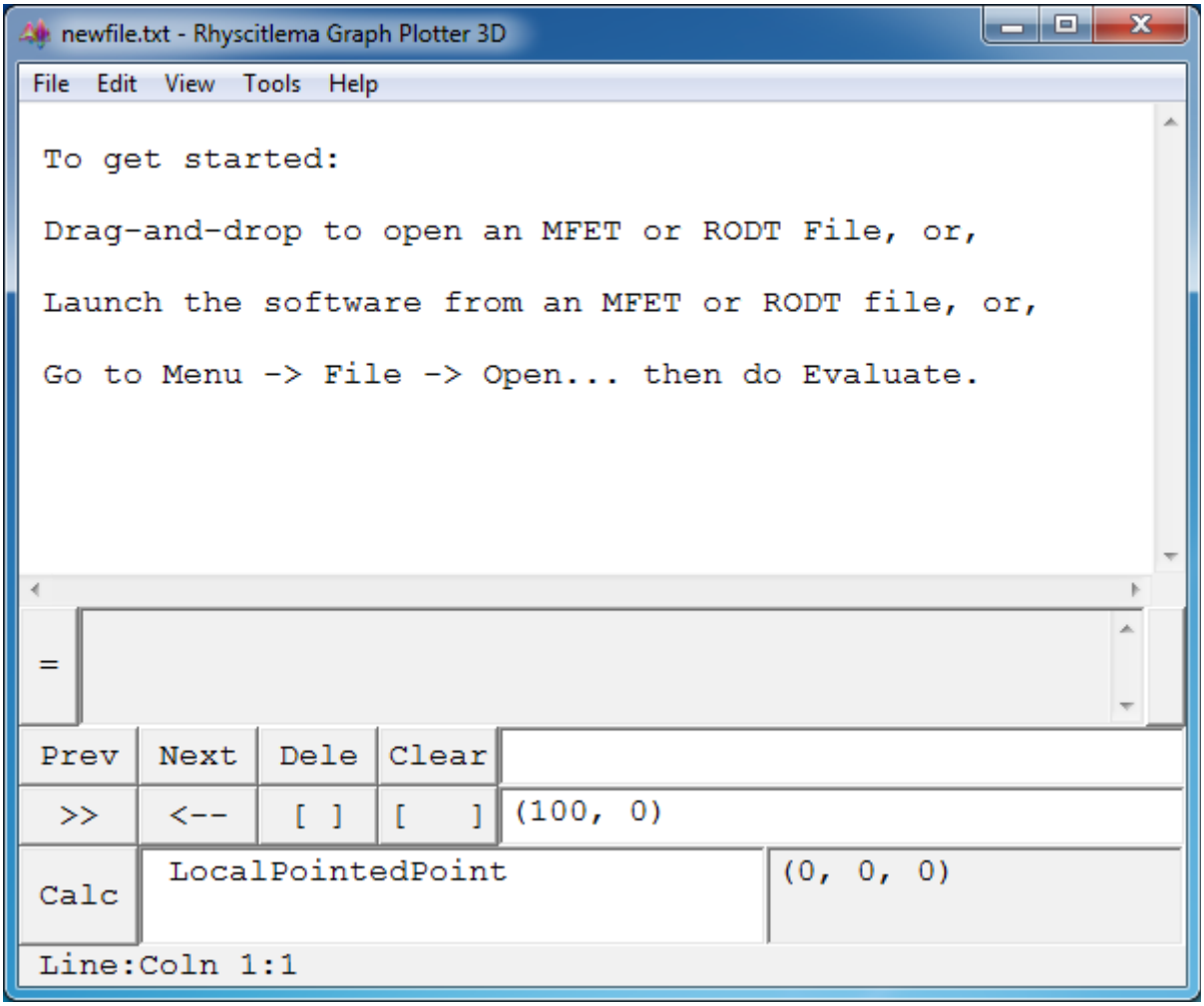
The replacement operator **:=** evaluates what is on its right-hand-side (RHS), then replaces what is on its left-hand-side (LHS) with the result. '*current*' is a special variable associated with the replacement operator. It is considered special only when there is an associated replacement operator. It simply is the value of the current LHS. This causes the replacement to be an update.

In the example above, current = 2. If the mouse right-button is clicked then Mouse_Right = 1. If the user has scrolled the mouse then MouseMotion[2] is non-zero. Suppose it is -1. So the RHS of the operator := will evaluate as (2 + 1 * -1) equal to 1. The expression will therefore change to:

- zoom = 1 := current + Mouse_Right * MouseMotion[2] ;

The replacement of the LHS is reflected or updated on the MFET text expression only when there is a **button press or release**. By pressing the **Escape key** this update is **canceled.** The changes made are also canceled. Therefore one can left-click, hold, move the mouse and observe changes, press the Escape key to cancel the changes, then release the left-click.

4 User Interface



A viewed object can be selected by clicking on it using the mouse **left-click**.
A displayed camera can be selected by clicking on it using the mouse **right-click**.
The MFET container text of the selected object will appear in the large user entry text field.

Note: As far as text file editing, the 1st and the 3rd buttons layers are concerned, the user interface is the exact same as that of the Rhyscitlema MFET calculator software. Refer to its user guide. Especially note that the Evaluate/Equal button can be used so to commit changes made to an object as well as create new objects. A new object is created when a newly created MFET container is ***detected*** to inherit (directly or indirectly) from the *base* object MFET container (as was discussed).

4.1 Software-specific Menu Features

- Menu → Tools → Save All Objects:
  - Saves all existing objects to a specified RODT file. It is recommended to use this feature as often as possible so to ensure that the entire scene is always saved, for in case the software fails or your computer fails.

- Menu → Tools → Remove All Objects:
  - Removes all existing objects (after a request for confirmation to proceed).

- Menu → Tools → Take Camera Picture:
  - After selecting a camera, this feature allows you to save a picture of it. Some image formats like PNG and BMP will save the alpha (opacity) component of the pixels colours.

- Menu → Toots → Prev/Next/Delete/Clear:
  - Do the same as their corresponding buttons found in the 2nd layer of buttons.

4.2 2nd layer of buttons and text field

At the far right of the 2nd layer is the *path* text field that displays the full path of the currently selected container. If empty then no container is selected (equivalent to the *Root* container selected).

The *Prev* and *Next* buttons are used to scroll through all containers, each time selecting the previous or next container – respectively. This includes inner-containers – only variables and functions are not selected this way. The *Dele* button deletes the selected container. If that is an object then the object is removed from the 3D virtual space. The *Clear* button clears the path as well as the message text fields. The cleared path means no container is selected.

A container can be explicitly selected by writing its full path inside the path text field. As it is a full path it must therefore start with a '|'. The method of applying the changes made may vary depending on the platform in which the software has been implemented. For Windows the changes are applied when keyboard focus is moved out of the text field.

Important: the Evaluate/Equal button is performed with respect to the selected container. Precisely, if what is evaluated does not have a name then the name of the currently selected container is used, thereby committing changes to this selected container. But if a name is provided and it is different from that of the current container, then the target container is searched for (or added if not found) as thought it was a ***direct sibling*** to the selected container.

Important: changes made to an inner-container does not reflect in its parent container. Therefore if this parent container is to be evaluated again then that inner-container will go back to its original state. This may not be wanted especially for objects that process user inputs.

4.3 4th layer of buttons and text field

The "Calc" button (found on the left) evaluates the content of the calculator input text field (found in the middle), and displays the result in the calculator result text field (found on the right).

In order to call a container from here, the component calling techniques as specified by the MFET language must be used. There are typically two methods:

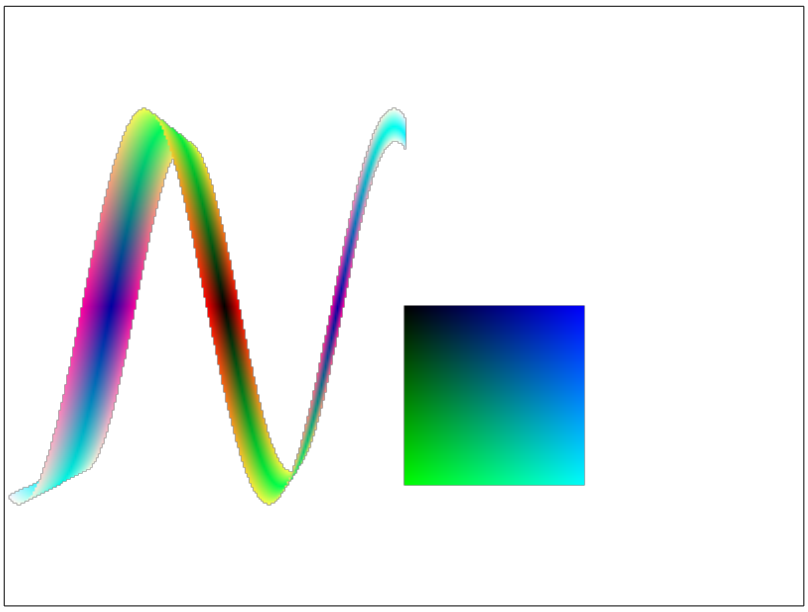
1. To inherit from a parent container so to be able to directly call its inner components.
2. To use the generic component calling mechanism: *"path".name*

5 A Basic Example

In this section a simple RODT is presented and discussed. In the discussion it is important to note how the limitations of certain objects are not found in others because of extra information provided. A very typical example of this is a plotted graph that cannot be moved due to its lack of processing user inputs.

Refer to the website http://www.rhyscitlema.com/applications/graph-plotter-3d for more examples. There you will find interesting features including colouring of objects using loaded images.

To load, copy the entire RODT into the large user entry text field, then click on Evaluate. The software will detect that the newly created container inherits from the RODT *base* container and will detect the inner-containers that are objects. The result is the transparent 400x300 image below:



5.1 Example RODT text

```
0;

type = "Rhyscitlema_Objects_Definition_Text" ;

message = "
           left-click on a graph

      or  right-click on a camera


 click, maintain and move the mouse to turn

 press ctrl, maintain and move the mouse to shift


 http://www.rhyscitlema.com/applications/graph-plotter-3d

";
```

```
\mfet{0;
 type =  "Camera" ;
 name =  "Camera0" ;

 origin =  {0,0,0} ;
 axes =  {1,0,0},{0,1,0},{0,0,1} ;

 rectangle =  {100, 100, 400, 300, 0, 0};
 zoom =  1 ;
}

\mfet{0;
 type =  "Surface" ;
 name =  "Graph1" ;

 origin =  (0, 0, 10) :=
            current + Mouse_Left *
           (MouseMotion * PixelSize .* (10, 10, 10));

 axes =  {1,0,0},{0,1,0},{0,0,1} ;

 function(x,y,z) =  0 ;
 colour(x,y,z) =  {x,y,z,1} ;

 boundary =  {0,1,0,1,0,1} ;
 accuracy =  1 ;
}

\mfet{0;
 type =  "Surface" ;
 name =  "Graph2" ;

 origin =  (-1, 0, 10) := "|file.mfet".moveObject(current);

 axes =  {1,0,0},{0,1,0},{0,0,1} ;

 function(x,y,z) =  y - sin(5x) ;
 colour(x,y,z) =  cabs(x,y,z,1) ;

 boundary =  {-1,1,-1,1,-1,1} ;
 accuracy =  10 ;
}
```

5.2 The camera object "Camera0"

The object with name "Camera0" has the simplest definition of a camera object. It does nothing but override the inner components of its base container. However, it does not process user inputs and is therefore completely static.

The camera origin position is (0,0,0), which is the same as the origin position of the 3D virtual space, referred to as the world's origin. Similarly to its position, the camera's local x, y and z axes are all the same as the world's x, y and z axes: (1,0,0), (0,1,0) and (0,0,1).

The camera rectangle provided will create a window positioned at the coordinates (100, 100) and

with the width and height equal to 400 and 300 pixels. Since this variable does not process any user input the window will not change when moved or resized. The typical way to define it is:

- rectangle = (100, 100, 400, 300, 0, 0) := current + CameraResize ;

Because the camera zoom is a positive value, the camera views what is in front of it (i.e: what is on its positive z-axis). The other objects should therefore be positioned such that they will turn out to be on the camera's positive z-axis. However it is typically best to first determine the position of the objects of a scene before determining that of the camera that views the scene – we cannot possibly afford to have a situation where *an observer of a system determines how that system will be*!

5.3 The surface object "Graph 1"

The object with name "Graph1" has a simple definition of a surface object. Even the function that defines the graph always evaluates to 0, thereby satisfying the surface equation for all points (x,y,z). Additionally the object processes user inputs, which enables the user to change the object's position using the mouse.

Notice how the origin position of the graph is on the world's positive z-axis. This enables the camera to view it. Also notice, from the position and shape of the graph, that the camera rectangle is centered on the origin of its plane of view (i.e. its local xy-plane).

5.4 The surface object "Graph2"

The object with name "Graph2" has a typical definition of a surface object. It is usually difficult to find a good *colour(x,y,z)* and a good *boundary* when plotting a graph. An idea is to express certain constants in terms of time 't', then watch the change in the plotted graph, and finally pause and record the value of these constants. The values can be obtained by calling ObjectName.VariableName in the calculator section of the user interface.

It is important to note how the method to obtain and use user inputs has been abstracted inside a file called file.mfet. This is an MFET file used as a library. Its full content may simply be as follows:

```
0;
 public moveObject(current) = current + Mouse_Left *
                              (MouseMotion * PixelSize) .* (10, 10, 10) ;
```

Note:

- '0;' is the very first MFET statement as required by the language.
- *public* is important because the call "|file.mfet".moveObject starts with *public* access level.
- *current* is the function parameter, a 3-value vector.
- *Mouse_Left* evaluates to 1 or 0, a single value.
- *MouseMotion* is the (x,y,z) displacement of the mouse in pixels, a 3-value vector.
- *PixelSize* is the size of a pixel in units-of-length per pixel, a single value.
- (10, 10, 10) is a 3-value vector. 10 is the distance from the camera to the graph.
- '.*' is the multiplication operator to perform *per-value* operation. If '*' was used instead then the operation performed would have been vector cross-product.
- All operators here are doing per-value operation.

For more about the MFET language and all its operators and operations please refer to the user guide for the Rhyscitlema MFET Calculator software.