The Beauty of Parametric Curves
A Guide

This guide is a companion to the article by Barbara Kaskosz "The Beauty of Parametric Curves" at the MathDL Flash Forum. This guide explains how to customize the gallery of parametric curves through simple editing of an XML file. It also gives a description of all the custom ActionScript 3 classes in the package bkde.as3. needed for the application.*

The article stemmed from a tutorial at http://www.flashandmath.com/ titled "Parametric Art Gallery - A Visual Experiment in ActionScript 3".

The AS3 classes in the package bkde.as3. come from an earlier article by Doug Ensley and Barbara Kaskosz "Flash Tools for Developers: Graphing Curves in the Plane" at the Math DL Flash Forum. The new classes are in the package are: ParamArtBoard and ParamGallery. The classes HorizontalSlider and VerticalSlider have been modified and improved.*

How to Modify the Parametric Art Gallery

The zipped folder param_art_dcr.zip contains all the source files ('fla' and 'as' files): gallery.fla, choose.fla, and the nested sequence of folders bkde.as3.* with the AS3 class files. However, if all you want to do is to customize the selection of parametric families for your applet, you do not need those source files. All you need are the runtime files; that is, the html files, the swf files, and the xml file.

The application presented in this article consists of two components: a gallery of parametric families (gallery.swf, gallery.html, and paramart.xml), and a general parametric families plotter (choose.swf and choose.html). The plotter allows you to graph and arbitrary parametric family and to find parametric families that you like and may want to include in your gallery.

For the gallery of parametric families to work, you need to place the files: gallery.html, gallery.swf, and paramart.xml in the same directory. The gallery.swf file is embedded in the gallery.html file through the html code. The file gallery.swf pulls data at runtime from paramart.xml. The XML file, paramart.xml, contains definitions of the parametric families included in the gallery. Therefore, all you need to do in order to change the selection of parametric families included in your gallery is to edit the paramart.xml file. The latter is essentially a text file. You can edit it with Notepad or any similar text editor.

You do not need to know much about XML to add, remove, or change families in the gallery. When you open paramart.xml, you will see that the structure of this xml file is very simple and self-explanatory. Every parametric family:

```
x = x(t,a,b) , y = (t,a,b),          (1)
```

for example,

```
x = b*sin(3*t),  y = a*cos(7*t),
```

where t is the parameter, and a, b are constants, is represented in the XML file by data enclosed between <plot …> and </plot> tags:

```
<plot func1="b*sin(3*t)" func2="a*cos(7*t)" >

<xRange min= "-11" max="11" />

<yRange min= "-11" max="11" />

<tRange min= "0" max="2*pi" />

<aRange min= "2" max="10" />

<bRange min= "2" max="10" />

</plot>
```

You can see clearly the equations for x and y, and then the ranges. The ranges for x and y determine the x and y ranges for the black graphing board; the t range is the range for the parameter t. For a given family, the vertical sliders change the values for the constants a, b, and display the corresponding graphs, 20 graphs at a time. Thus, you need to provide ranges for the constants a and b as well.

That is all there is to it. To change a family, you edit the above data. To remove a family, delete <plot …> and </plot> tags and everything in between for the family you don't want. To add a family add a new <plot …> … </plot> block containing the new data.

The applet choose.swf embedded in choose.html page allows you to enter an arbitrary parametric family of the form (1) into the input boxes placed within the applet, enter all the ranges as well in the appropriate boxes, and then play with the sliders to see if you like the corresponding images. If you do, add the new family to your gallery by editing paramart.xml.

In case, you are interested in the source code, source files are included in the package and you can find below full documentation of AS3 classes.

Classes in the Package bkde.as3.*

bkde.as3.parsers.CompiledObject

Description

CompiledObject is a helper class for MathParser. It creates a convenient datatype to be returned by doCompile method of MathParser. This datatype is an object with three properties listed below which comprise the results of compiling a mathematical formula into a form suitable for evaluation.

Constructor

The constructor is evoked by the keyword "`new`" and takes no parameters:

- **`new CompiledObject();`**

You shouldn't encounter the need to use the constructor since the only instances of CompiledObject that can conceivably be useful are those returned by doCompile method of MathParser.

Public Methods

None.

Public Properties

CompiledObject has three public instance properties.

- ***instance*`.PolishArray : Array`**

When the instance is returned by doCompile method of MathParser, the array represents a mathematical formula in the Polish notation.

- ***instance*`.errorStatus : Number`**

When the instance is returned by doCompile method of MathParser, the property has value 1 if an error is found and 0 otherwise.

- ***instance*`.errorMes : String`**

When the instance is returned by doCompile method of MathParser, the string contains a message to the user indicating where in the input a mistake was found.

bkde.as3.parsers.MathParser

Description

An instance of MathParser (you create an instance using the class's constructor described below) will compile a string that represents a mathematical formula (usually the user's input) and then calculate the values of the compiled formula for given values of variables that are recognized by the instance. "Compiling" consists of rewriting a formula in a form suitable for evaluation; that is, in the Polish notation. Compiling will be successful if the user obeys by the simple syntax rules described at the end of this section.

Constructor

The constructor is evoked with the word "`new`":

- **`new MathParser(`*`parameter1`*`:Array)`**

The constructor takes one parameter which is an array. For the MathParser's instance to do what you want it to do, the parameter has to be an array of strings. The strings represent the names of variables that the instance will recognize. For example:

```
var procFun:MathParser = new MathParser(["t", "a", "b"]);
```

The instance "procFun" will recognize the variables t, a and b.

```
var procFormula:MathParser = new MathParser(["t"]);
```

The instance "`procFormula`" will recognize `t` as a variable. MathParser knows the constants e and pi. Do not enter them into the constructor.

Note: Variables have to be entered as strings. It is:

```
procFormula:MathParser =  new MathParser(["x"]);
```

and not:

```
procFormula:MathParser = new MathParser([x]);
```

Variables can be comprised of more than one letter, e.g.:

```
procFormula:MathParser = new MathParser(["tension"]);
```

It is important to remember the order in which you pass your variables to the constructor since the evaluator method of the parser will expect values for those variables in the same order.

Public Methods

MathParser has two public instance methods.

- ***instance*.doCompile(*parameter1*:String): CompiledObject**

The method takes a string (typically a mathematical formula entered by the user) and returns an instance of CompiledObject. If no mistakes in syntax are found, the PolishArray property of the returned CompiledObject instance represents the formula in the Polish notation, errorStatus=0, errorMes= " ". If a mistake is found, errorStatus=1, errorMes contains a message indicating where the mistake was found, PolishArray=[];

- ***instance*.doEval(*parameter1*:Array,*parameter2*:Array): Number**

The method takes two parameters, both arrays. For the method to be useful, the first array must be the PolishArray property of a CompiledObject returned by doCompile method. The second parameter represents an array of numerical values for the variables recognized by the instance of MathParser. (The same variables that you passed to the constructor of your MathParser instance.) Under these conditions, doEval will return the value of the formula represented by the PolishArray for the specified values of the variables.

Public Properties

None.

Syntax Accepted by MathParser

The parser expects calculator-like syntax: e.g.:

sin(2*x^2)-e^-x+tan(pi*x)/2

Multiplication must be entered as *. Arguments of functions must be enclosed in parentheses. The parser is case-insensitive and blind to white spaces. It recognizes the constants e and pi. Here is the list of functions that the parser knows:

sin(·), cos(·), tan(·), asin(·), acos(·), atan(·), ln(·), sqrt(·), abs(·), ceil(·), floor(·), round(·), max(·,·), min(·,·).

Addition, multiplication, division and exponentiation are denoted by the usual symbols +,*, /, ^ , subtraction and unary minus by - .

The class MathParser in our tutorial is used by the class ParamArtBoard to process the user's input for x(t,a,b) and y(t,a,b) in choose.fla, or the corresponding data from an xml file in gallery.fla.

bkde.as3.parsers.RangeObject

Description

RangeObject is a helper class for RangeParser. It creates a convenient datatype to be returned by parseRangeTwo and parseRangeFour methods of RangeParser. This datatype is an object with three properties listed below which comprise the results of compiling the user's range input for two variables, say x and y, or for one variable, say a parameter t. In most of our applets the range boxes allow for numerical entries as well as for entries containing pi, like pi/4, 2*pi etc.. Such entries have to be parsed and checked for validity.

Constructor

The constructor is evoked by the keyword "`new`" and takes no parameters:

- **`new RangeObject();`**

You shouldn't encounter the need to use the constructor since the only instances of RangeObject that can conceivably be useful are those returned by parseRangeTwo or parseRangeFour methods of RangeParser.

Public Methods

None.

Public Properties

RangeObject has three public instance properties.

- ***instance*.Values : Array**

When the instance is returned by one of the methods of RangeParser, the array represents four range values (if ranges for two variables are being parsed) or two range values if the range for one variable is being parsed.

- ***instance*.errorStatus : Number**

When the instance is returned by one of the RangeParser methods, the property has value 1 if an error is found and 0 otherwise.

- ***instance*.errorMes : String**

When the instance is returned by one of the RangeParser methods, the string contains a message to the user indicating where in the input a mistake was found.

bkde.as3.parsers.RangeParser

Description

In most of our applets the range boxes for x and y or for a parameter, say t, allow for numerical entries as well as for entries containing pi, like pi/4, 2*pi etc.. Such entries have to be parsed and checked for validity.

Constructor

The constructor is evoked with the word "`new`" and takes no parameters:

- **`new RangeParser();`**

For example

```
var procRange:RangeParser = new RangeParser();
```

Public Methods

RangeParser has two public instance methods.

- ***instance*`.parseRangeTwo(`*`parameter1`*`:String, `*`parameter2`*`:String):`**
  `RangeObject`

The method takes two strings (typically the user's entries in range boxes for a parameter t, for example) and returns an instance of RangeObject. If the entries are found to be valid numerical entries (or valid entries containing pi), and the first entry is less than the second entry, the Values property of the returned RangeObject contains the two range values. In that case, errorStatus=0, errorMes= " ". If a mistake was found, errorStatus=1, errorMes contains a message indicating where the mistake was found, Values=[];

- ***instance*`.parseRangeFour(`*`parameter1`*`:String, `*`parameter2`*`:String,`**
  ***`parameter3`*`:String, `*`parameter4`*`:String):RangeObject`**

The method takes four strings (typically the user's entries for x and y ranges) and returns an instance of RangeObject. If the entries are found to be valid numerical entries (or valid entries containing pi), the first entry is less than the second entry, the third entry is less than the fourth entry, then the Values property of the returned RangeObject contains the corresponding four range values. In that case, errorStatus=0, errorMes= " ". If a mistake was found, errorStatus=1, errorMes contains a message indicating where the mistake was found, Values=[];

Public Properties

None.

The class RangeParser in our tutorial is used by the class ParamArtBoard to process the user's input for the x, y, t, a, and b ranges in choose.fla, or the corresponding data from an xml file in gallery.fla.

bkde.as3.boards.ParamArtBoard

Description

ParamArtBoard performs many tasks in our parametric art applets. The class parses formulas for x and y coordinates which are in terms of t, a, and b. It parses the range entries for x, y, t, a, and b. The formulas and the entries come either from the user's input, as in "Parametric Art – Choosing a Family" applet, or from an external xml file as in "Parametric Art – A Gallery" applet. The class then plots the curves corresponding to given values for t, a, and b. The class is also responsible for drawing the square graphing board in which our plots reside. The class contains and controls the error display box. The layout, the colors, and the sizes of all elements of an instance of ParamArtBoard are all easily customizable via instance methods of the class.

ParamArtBoard extends Sprite. Thus, it inherits from Sprite. In particular, you can control the position of your instance of ParamArtBoard within the main movie with the Sprite methods and properties:

```
instance.x

instance.y
```

These properties set the x and the y coordinates in pixels of the upper left corner of your instance of ParamArtBoard with respect to the upper left corner of the parent movie. Recall that in Flash, the x coordinate increases to the right, the y coordinate increases as you go down.

Constructor

The constructor is evoked with the word `"new"` and takes two numerical parameters. The parameters are the width and the height, in pixels, of the rectangular graphing board which will be drawn:

- **`new GraphingBoard(w:Number,h:Number);`**

Public Methods – ParamArtBoard Appearance

- ***`instance`*`.changeBackColor(h:Number): void`**

The method controls the background color of the graphing board created by the instance. The numerical parameter should be the desired color in the hexadecimal form.

Default: white.

- ***instance*.changeBorderColorAndThick(h:Number,t:Number): void**

The method controls the color and the thickness of the border of a graphing board created by the instance. The first parameter passed to the method should be the hexadecimal form for the desired color.

Default: black, 1.

Public Methods – Error Display

An instance of ParamArtBoard controls the text field for displaying error messages to the user. You can control the appearance and the position of the error text field with the following methods.

- ***instance*.setErrorBoxSizeAndPos(w:Number,h:Number,xpos:Number, ypos:Number): void**

The parameters determine: the width, the height (in pixels) of the error text field, and its x and y position relative to your instance of ParamArtBoard.

Default: The text field is positioned over the upper half of the graphing board created by the instance.

You can set visual attributes of the error box with the method:

- ***instance*.setErrorBoxFormat(c1:Number,c1:Number,c3:Number,s:Number): void**

The parameters determine: the background color, the border color, the text color, and the text size. All colors should be passed in hex.

Default values: white, white, black, 12.

Note: The error display text field is a public property of ParamArtBoard:

- ***instance*.ErrorBox**

Hence, you can control it directly through methods of the TextField class. In particular, you can assign text to the ErrorBox directly via:

- ***instance*.ErrorBox.text="message to the user";**

We made ErrorBox property public to give you easy control over the visibility of ErrorBox and the text displayed in it. (The error box is visible when the user made an error; its text is an error message to the user determined by the kind of error found.)

Public Methods – Parsing Input

- ***instance*.procInput(f1:String,f2:String,x1:String,x2:String,**
  y1:String,y2:String,t1:String,t2:String,
  a1:String,a2:String,b1:String,
  b2:String,):void

The method uses an instance of MathParser and an instance of RangeParser to parse strings corresponding to formulas for x(t,a,b), y(t,a,b) and the range entries for x, y, t, a, and b. If an error is found, the ErrorBox becomes visible and a message is displayed. Otherwise, the formulas and values for all ranges are properly stored and the ParamArtBoard is ready to plot curves.

Public Methods – Graphing

For every instance of ParamArtBoard the maximum number of graphs that can be displayed simultaneously should be set via the method:

- ***instance*.setMaxNumGraphs(a:int): void**

Default: 5.

In our applets, we set the value to 20.

Graphs of curves are drawn using the method:

- ***instance*.drawGraph(num:int,thick:Number,a:Number,b:Number):void**

The method takes four parameters. The first, an integer, is the number of the graph being drawn. This integer must not exceed the maximum number of graphs to be displayed at one time (set by *instance*.setMaxNumGraphs(..) method. The second parameter will determine the thickness of your graph, in pixels. The third and the fourth parameters correspond to current values for the constants a and b. When the values for a and b are given, the graph of a curve is produced by evaluating the formulas for x(t) and y(t) at many consecutive values of t throughout the range for t. The points (x(t),y(t)) obtained in this manner are joint by lineal elements. At how many points will the evaluation takes place is determined by the public property *instance*.numPoints. In our applets, numPoints is set to 200.

Public Methods – Clearing the Graphing Board

To clear the graphs you use the method:

- ***instance*.resetBoard(): void**

The method erases all graphs and sets the read-only public property *instance*.isReady to false. This means the board is not ready to graph. The method is evoked before loading a new example.

- ***instance*.eraseGraphs(): void**

This method erases curves but keeps all the information about a current example so drawing can continue. The method is called by the Clear button in our applets.

Public Methods – Other

Here are some other possibly useful methods of ParamArtBoard class.

- ***instance*.getMaxNumGraphs(): int**

Use this method if you are not sure what the maximum number of graphs is set to. The next two methods are self-explanatory:

- ***instance*.getBoardWidth(): Number**

- ***instance*.getBoardHeight(): Number**

The next four methods allow you to convert functional coordinates to their pixel equivalents and vice versa. Note: these methods will work only if the ranges for your horizontal and vertical variables are set. We do not use those methods in our applets.

- ***instance*.xtoPix(a:Number): Number**

- ***instance*.ytoPix(a:Number): Number**

- ***instance*.xtoFun(a:Number): Number**

- ***instance*.ytoFun(a:Number): Number**

A testing method, which, again, is not used by our applets:

- ***instance*.isDrawable(a:*): Boolean**

The method returns “true” if “a” is of the numerical datatype and it is a finite number, and its absolute value does not exceed 5000. Otherwise, the method returns “false”. The reason you may want to have a test of such kind is that an attempt to draw an object, a portion of a graph or a cursor, located very far away from the graphing board (in pixels), you may encounter unexpected results.

Finally, if you want to remove an instance of ParamArtBoard at runtime, you should call

- ***instance*.destroy(): void**

The method removes all listeners set by your instance of ParamArtBoard, clears all drawings, and sets all the Sprites created by the instance to null.

Public Properties

The only two read-write public properties of ParamArtBoard (except for those inherited from Sprite) are ErrorBox and numPoints.

- ***instance*.ErrorBox**

It is a dynamic text field in which error messages to the user can be displayed. As we described above, you can set the size, the position, and the formatting for the text field by using ParamArtBoard methods. You can also apply the TextField class properties and methods to ErrorBox, e.g.:

```
board.ErrorBox.visible=true;

board.ErrorBox.text="Error somewhere.";
```

- ***instance*.numPoints:Number**

The property determines the number of points along each curve at which the coordinates are going to be evaluated to produce the graph. More points give smoother graphs but too many points may slow our applets down. We set the property to the default 200.

ParamArtBoard class has many read-only properties.

- ***instance*.isReady:Boolean**

The default value of 'false' is set to true after the method procInput has run successfully and the instance is ready to draw curves. In that case, the values of the following read-only properties are assigned. The meaning of those properties is clear from their names.

- ***instance*.xmin:Number**
- ***instance*.xmax:Number**
- ***instance*.ymin:Number**
- ***instance*.ymax:Number**
- ***instance*.tmin:Number**
- ***instance*.tmax:Number**
- ***instance*.amin:Number**
- ***instance*.amax:Number**
- ***instance*.bmin:Number**
- ***instance*.bmax:Number**

In our applets, we use all read-only properties of the class.

bkde.as3.boards.ParamGallery

The class extends the EventDispatcher class. Its main purpose is to load examples for a gallery from an external xml file and prepare them to be processed by ParamArtBoard class.

Constructor

The constructor is evoked with the word "`new`" and takes no parameters:

- **`new ParamGallery();`**

Public Static Constants

- **`DATA_LOADED:String="dataLoaded"`**

- **`LOAD_ERROR:String="loadError"`**

These two constants correspond to two custom events dispatched by an instance of the class while loading an external xml file. The first is dispatched when the file has loaded successfully and the data corresponding to examples has been stored in arrays. The second event is dispatched if a loading error is encountered (other than Flash Player security errors. The class does not listen to those.)

Please look at the well-commented code in gallery.fla to see how to use these events.

Public Methods

- ***`instance`*`.loadExamples(str:String): void`**

The string parameter represents the address of an xml file containing data for items in our gallery. The file has to be structured as our paramart.xml for the method to work properly. The method dispatches the DATA_LOADED event and prepares the data if loading was successful; it dispatches the LOAD_ERROR event otherwise.

- ***`instance`*`.getExample(n:int): Array`**

The method returns the data corresponding to the example number n as an array of strings that represent formulas for x and y, and x, y, t, a, and b ranges. The array can be then processed by the procInput method od ParamArtBoard class.

Public Properties

- ***`instance`*`.examplesTotal: int`**

This read-only property returns the number of items found in the xml file.

bkde.as3.utilities.HorizontalSlider

Description

Flash CS3 has a slider component. Our class provides a light-weight, easily customizable alternative. The class HorizontalSlider extends Sprite. Hence, it inherits from Sprite. Each instance of HorizontalSlider consists of a track, 3 pixels wide, with four tick marks, and a draggable knob. You can control the length, the style, the colors, and the appearance of the slider using the methods listed below.

Constructor

The constructor is evoked with the word "`new`" and takes two parameters. The first parameter is the length, in pixels, of the slider to be created. The second, a String parameter, determines the style of the draggable knob:

- **`new HorizontalSlider(len:Number,style:String);`**

The available styles for the knob are" "triangle" and "rectangle".

Public Static Constants

- **`SLIDER_CHANGE:String = "sliderChange"`**

The constant contains the name of the custom event that is diaptched when the knob is being dragged by the user. Consult gallery.fla file to see the event in action.

Public Methods

Most methods of the class are meant to give you control over your slider's visual attributes. The names of the properties are quite self-explanatory.

- ***instance*`.changeKnobColor(c:Number): void`**

Default: dark gray.

- ***instance*`.changeKnobSize(c:Number): void`**

Default: 8 (in pixels).

- ***instance*`.changeKnobOpacity(n:Number): void`**

Default: 1.0 – completely opaque.

You may find this method useful with a rectangular knob if you want the tick marks underneath to show.

- ***instance*.changeKnobLeftLine(c:Number): void**

- ***instance*.changeKnobRightLine(c:Number): void**

Default: white, black.

The methods change the colors of the left and the right outline of the knob to create a 3D effect with your coloring scheme.

- ***instance*.changeTrackOutColor(c:Number): void**

- ***instance*.changeTrackInColor(c:Number): void**

Default: dark gray, white.

The methods change the colors of the outline of the track and the line inside to match your coloring scheme.

- ***instance*.setKnobPos(p:Number): void**

The method adjusts the x coordinate of the knob along the horizontal track. 0 corresponds to the left end of the slider. The method is usually used to set the initial position of the knob.

- ***instance*.getKnobPos(): Number**

This extremely important method allows you to make your applet respond to the changing horizontal position of the knob as the user drags the knob along the track.

Here are the last two methods of lesser importance.

- ***instance*.getSliderLen(): Number**

The method returns the slider's length.

Finally, if you want to remove an instance of HorizontalSlider at runtime, you should call

- ***instance*.destroy(): void**

The method removes all listeners set by your instance of HorizontalSlider, clears all drawings, and sets all the Sprites created by the instance to null.

Public Properties

Each instance of HorizontalSlider has one public property (except for those inherited from Sprite). This property, "isPressed", is a read-only property:

- ***instance*.isPressed: Boolean**

The property is set by the class to "true" if the user presses the mouse button over the knob. The property is reset to the default – "false" when the user releases the mouse button over or outside the knob.

bkde.as3.utilities.VerticalSlider

Description

The class is nearly identical to the HorizontalSlider class. It has the same methods and properties. The only difference is that the track is drawn vertically and all references in the description above to the x (or horizontal) coordinate should be replaced by references to the y (or vertical) coordinate. Since HorizontalSlider inherits from Sprite, any instance of HorizontalSlider can be positioned vertically via Sprite.rotation property. The reason we have a separate class has to do with slight differences in the way the vertical slider is drawn which give it a more pleasing appearance.

Constructor

The constructor is evoked with the word "new" and takes two parameters. The first parameter is the length, in pixels, of the slider to be created. The second, a String parameter, determines the style of the draggable knob:

- **new VerticalSlider(len:Number,style:String);**

The available styles for the knob are" "triangle" and "rectangle".

Public Methods

Same as for HorizontalSlider.

Public Properties

Same as for HorizontalSlider.

Note: The package bkde.as3.boards contains the class GraphingBoard. That class is not used in our parametric art applets.

May 1, 2008