# Flash Tools for Developers: 3D Surface and Function Graphers in ActionScript 3

August 16, 2007

*This paper is a companion to the online article at the MathDL Digital Classroom Resources "Flash Tools for Developers: 3D Surface and Function Graphers in ActionScript 3" by Barbara Kaskosz and Doug Ensley. We give a short tour of the templates featured in the article in the first part of this paper. In the second part, we provide complete documentation of the custom classes in the package bkde.as3.*. We include documentation for the classes used in this article as well as for those used in our previous ActionScript 3 articles. This way each of our consecutive AS3 articles will contain the latest versions of all the classes in the package. The classes will undergo modifications based on the readers' suggestions.*

*All files, including fla files and class files, are contained in param_surf_as3.zip*

## The Templates

In this article, you will find five applets which can be viewed as easily customizable templates. All of them are based on the new GraphingBoard3D class. They also use MathParser and several other classes from the bkde.as3.* package.

*Parametric Surfaces in ActionScript 3 – Rectangular Coordinates*

In this applet, surf_graph_rectan.swf, the user enters parametric formulas for the rectangular coordinates x(t,s), y(t,s), and z(t,s) and for the ranges for t and s. The applet draws the corresponding surface which then can be rotated dynamically. The user can choose the opacity of the surface and scaling constrained option. A gallery of ten pre-programmed examples is provided. In this template, the examples are defined in the fla file. The code in the corresponding fla file contains detailed comments.

*Parametric Surfaces in ActionScript 3 – Cylindrical Coordinates*

In the template surf_graph_cylin.swf, the user enters parametric formulas for the cylindrical coordinates r(t,s), θ(t,s), and z(t,s) and for the ranges for t and s. Other options are the same as in the previous template. A gallery of ten pre-programmed examples is provided. In this template, the examples are defined in an external XML file and loaded at runtime. This makes refreshing the gallery of examples easy without recompiling the fla file. For the runtime loading to work, the XML file, cylindrical.xml, must reside in the same directory as param_surf_cylin_as3.swf. In the fla file, we comment only the parts that are different from the first template.

*Parametric Surfaces in ActionScript 3 – Spherical Coordinates*

In this applet, surf_graph_sphere.swf, the user enters parametric formulas for the spherical coordinates $\rho(t,s)$, $\theta(t,s)$, and $\phi(t,s)$ and for the ranges for t and s. Other options are the same as in the previous templates. Again, we provide a gallery of ten pre-programmed examples contained in an external XML file, spherical.xml. In the fla file, we comment only the parts that are specific to this template.

*3D Function Grapher in ActionScript 3 – on White*

In this template, fun_graph3d_white.swf, we present a grapher for functions of two variables. In many ways, it is a special case of a parametric grapher although we use a different coloring method and simplify some parts of the code for that case.

*3D Function Grapher in Action Script 3 – on Black*

The purpose of this template, fun_graph3d_black.swf, is to illustrate the methods for customizing appearance of the grapher using the methods of GraphingBoard3D class.

# Classes in the Package bkde.as3.*

Note: Classes that are new in this article are GraphingBoard3D, MatrixUtils, and StringUtils. All other classes have not changed since our last article. Hence, except for the three new classes, all examples of code below and references to templates concern planar graphers templates from our previous MathDL Flash Forum article: *"Flash Tools for Developers (AS3): Graphing Curves on the Plane"*. Download planar_tools_as3.zip from the latter article if you want to see the code in context.

## 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(["x", "y"]);
```

The instance "procFun" will recognize the variables x and y.

```
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.

***Examples of Use***

In our *Function Grapher in ActionScript 3.0 – Template 1*, we use MathParser as follows:

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

We created an instance that will recognize x as a variable. The user enters text into the input text field, InputBox1, on the Stage. The text entered represents a formula for the first function to be graphed. Within the function makeGraphs, we retrieve the user's input and store in a string variable sFunction1; we declare other variables related to parsing:

```
var sFunction1:String= "";
……
var compObj1:CompiledObject;
……

sFunction1=InputBox1.text;
```

If the input is not empty, we apply to it doCompile method of the parser and store the result in the variable compObj1. (The method returns an instance of CompiledObject.)

```
compObj1=procFun.doCompile(sFunction1);
```

If compObj1.errorStatus=0, the user entered a formula properly. The formula is now stored in the form ready for evaluation (that is, rewritten in the Polish notation) in compObj1.PolishArray. To evaluate the formula for a given value of x, say xmin which represents a number defined previously in the script, we use doEval method of the parser as follows:

```
procFun.doEval(compObj1.PolishArray,[xmin]);
```

In makeGraphs function the latter value as well as the values of the formula for other values of x between xmin and xmax are stored as elements of an array, f1Array.

***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 - .

## 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.

***Examples of Use***

In our *Function Grapher in ActionScript 3.0 – Template 1*, we use RangeParser as follows:

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

Then inside the function makeGraphs:

```
      sXmin=XminBox.text;
      sXmax=XmaxBox.text;
      sYmin=YminBox.text;
      sYmax=YmaxBox.text;
      oRange=procRange.parseRangeFour(sXmin,sXmax,sYmin,sYmax);
    if(oRange.errorStatus==1){
            board.ErrorBox.visible=true;
            board.ErrorBox.text="Error. "+oRange.errorMes;
            return;
      }
      xmin=oRange.Values[0];
      xmax=oRange.Values[1];
      ymin=oRange.Values[2];
      ymax=oRange.Values[3];
```

"board" above is an instance of GraphingBoard which controls the error display box board.ErrorBox.

## bkde.as3.boards.GraphingBoard

---

***Description***

GraphingBoard is the main visual class for creating customizable planar graphers: function graphers, parametric curves graphers, etc.. An instance of GraphingBoard draws a rectangular graphing board (at runtime), the vertical and horizontal coordinate axes as well as graphs of functions or parametric curves. Any instance of GraphingBoard contains and controls an error display text field where messages to the user can be displayed. It also contains a coordinate display text field in which the values of the horizontal and the vertical coordinates are displayed when the user mouses over the graphing board. An instance of GraphingBoard can enable the user to draw within the graphing board with the mouse.

The layout, the colors, and the sizes of all elements of an instance of GraphingBoard are all easily customizable via instance methods of the class.

An instance of GraphingBoard sets x and y ranges in functional terms (usually based on the user's input), and provides public methods for translating pixel coordinates of a point into its functional coordinates and vice versa. For simplicity of this presentation, we will assume most of the time that the horizontal and vertical variables in your applet are named "x" and "y' although you may choose different names using the method "enableCoordsDisp(…,…)" described later. The pixel coordinates in the Flash's coordinate system are, however, always x and y.

GraphingBoard extends Sprite. Thus, it inherits from Sprite. In particular, you can control the position of your instance of GraphingBoard 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 GraphingBoard 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.

In our *Function Grapher in ActionScript 3.0 – Template 1*, we have an instance of GraphingBoard named "board", a child of the main movie. We position "board" within the main movie with:

```
board.x=20;

board.y=20;
```

***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);`**

In our *Function Grapher in ActionScript 3.0 – Template 1*, we create a 350 by 350 graphing board as follows:

```
var board:GraphingBoard = new GraphingBoard(350,350);
```

We store our instance of GraphingBoard in a variable called "board" whose datatype is GraphingBoard. We pass to the constructor the size of our graphing board; all other attributes will be set using the methods of the class.

***Public Methods – Graphing Board 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.

To continue with examples from *Function Grapher in ActionScript 3.0 – Template 1* where the instance of GraphingBoard is stored in "board":

```
board.changeBackColor(0x000000);
```

We set the background to black.

- ***`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.

In *Function Grapher in ActionScript 3.0 – Template 1* where the instance of GraphingBoard is stored in "board" we use:

```
board.changeBorderColor(0xFFFFFF,1);
```

We set the border color to white, its thickness to 1.

***Public Methods – Coordinate Axes Appearance***

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

The method sets the color and the thickness, in pixels, of the x and y axes. The color should be passed to the method in its hexadecimal form.

Default: black, 0.

(Thickness set to 0 produces a line 1 pixel wide whose thickness will not change with rescaling.)

In *Function Grapher in ActionScript 3.0 – Template 1,* we have:

```
board.setAxesColorAndThick(0xCCCCCC,0);
```

We set the color of the coordinate axes to light gray, their thickness to 0.

Note: Colors and thickness of graphs of functions or curves are passed directly to the graphing method of GraphingBoard as shown later in this guide. That way those attributes can vary from graph to graph.

***Public Methods – Drawing by the User***

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

The method enables the user to draw on the graphing board with the mouse. It sets the color and the thickness, in pixels, of the user's drawing.

Default: enabled in red with thickness 0.

In *Function Grapher in ActionScript 3.0 – Template 1,* we have:

```
board.enableUserDraw(0xFFFF00,1);
```

We enable the user to draw in yellow with thickness 1.

If you want to disable the drawing capability, use the method:

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

***Public Methods – Error Display***

An instance of GraphingBoard 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 GraphingBoard.

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

In *Function Grapher in ActionScript 3.0 – Template 1,* we have:

```
board.setErrorBoxSizeAndPos(310,120,20,20);
```

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.

In *Function Grapher in ActionScript 3.0 – Template 1,* we have:

```
board.setErrorBoxFormat(0x000000,0x000000,0xCCCCCC,12);
```

We chose black background, black border, and gray text of size 12.

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

- ***instance*.ErrorBox**

Hence, you can control its attributes directly through methods of Flash's TextField class. It is easier, however, to use the methods of GraphingBoard to set properties of the error box.

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.)

Note: The initial visibility of the error box is set to false.

In *Function Grapher in ActionScript 3.0 – Template 1,* we have the following code within makeGraphs function which displays an error massage to the user if a mistake in range entries is found:

```
    sXmin=XminBox.text;
```

```
    sXmax=XmaxBox.text;

    sYmin=YminBox.text;

    sYmax=YmaxBox.text;

    oRange=procRange.parseRangeFour(sXmin,sXmax,sYmin,sYmax);

  if(oRange.errorStatus==1){

          board.ErrorBox.visible=true;

          board.ErrorBox.text="Error. "+oRange.errorMes;

          return;

    }
```

***Public Methods – Coordinates Display***

An instance of GraphingBoard controls the text field for displaying the values of the horizontal and vertical variables (in functional terms) when the user mouses over the graphing board. You can control the appearance and the position of the coordinate text field with the following methods.

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

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

Default: The text field is positioned in the lower left corner a graphing board created by the instance.

In *Function Grapher in ActionScript 3.0 – Template 1*, you can see:

```
board.setCoordsBoxSizeAndPos(60,40,20,300);
```

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

- ***instance*.setCoordsBoxFormat(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.

In *Function Grapher in ActionScript 3.0 – Template 1*, we use the method:

```
board.setCoordsBoxFormat(0x000000,0x000000,0xCCCCCC,12);
```

We chose black background, black border, and gray text of size 12.

You can disable the coordinate display box using the method:

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

Default: enabled.

You can use the "enableCoordsDisp" method to enable coordinates display and to change the names of your horizontal and vertical variables displayed in the coordinates display box from the default "x" and "y" :

- ***instance*.enableCoordsDisp(h:String,v:String): void**

The coordinates displayed in the box will be named according to the string parameters h and v that you pass to the method.

Default: enabled with coordinates named "x" and "y".

We do not use the latter method in our templates. If you want your coordinates to be named, for example, "t" and "s", and your instance of GraphingBoard is "board" you call:

```
board.enableCoordsDisp("t","s");
```

***Public Methods – Tracing Cursors***

GraphingBoard provides methods helpful for building graph-tracing mechanisms into your applet. The class allows two styles for the tracing cursor: "cross" and "arrow". The corresponding method is

- ***instance*.setTraceStyle(s:String): void**

The string parameter "s" has two values that will produce a cursor: "cross" and "arrow".

Default value: "cross" .

In our template *Parametric Curves in ActionScript 3.0 – Basic,* we chose the arrow:

```
board.setTraceStyle("arrow");
```

For each of the two styles you have much control over the size and the color of the tracing cursor. Below are the corresponding methods.

- ***instance*.setCrossSizeAndThick(s:Number,t:Number): void**

The method sets the size and the line thickness for the cross cursor.

Default: 6, 1.

In *Function Grapher in ActionScript 3.0 – Template 2*, you see:

```
board.setCrossSizeAndThick(6,2);
```

We wanted the cursor thicker than the default value.

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

The method sets the color for the cross cursor. The color should be passed in its hex form.

Default: black.

In *Function Grapher in ActionScript 3.0 – Template 2,* we have:

```
board.setCrossColor(0x9900FF);
```

We chose magenta for our cursor.

- ***instance*.setCrossPos(s:Number,t:Number): void**

The method sets the position of the cross cursor, in pixels, relative to the instance's upper left corner.

Default: The upper left corner.

- ***instance*.crossVisible(b:Boolean): void**

The method sets the visibility of the cross cursor to "false" or "true" as in:

```
board.crossVisible(true);
```

Default: false.

The two last methods are used to move the cursor along a given graph and make it visible or invisible depending on the user's actions. You will find tracing functionality in *Function Grapher in ActionScript 3.0 – Template 2.*

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

The method returns cross' size in case you need it for positioning purposes. We have similar methods for the arrow cursor.

- ***instance*.setArrowSize(s:Number): void**

The method sets the size for the arrow cursor.

Default: 10.

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

The method sets the color for the arrow cursor. The color should be passed in its hex form.

Default: black.

- ***instance*.setArrowPos(s:Number,t:Number,r:Number): void**

The method sets the x and y coordinates, in pixels, of the arrow cursor relative to the upper left corner an instance, x=s, y=t. The last parameter represents the rotation of the arrow, counterclockwise, in degrees. The rotation parameter allows the arrow to move along a curve and rotate accordingly.

Default: The upper left corner, pointing upwards.

The last method is used to build a tracing mechanism in which an arrow traces a parametric curve when the user slides a slider. You will find an example of such mechanism in *Parametric Curves in ActionScript 3.0 – Basic.*

- ***instance*.arrowVisible(b:Boolean): void**

The method sets the visibility of the arrow cursor to "false" or "true".

Default: false.

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

The method returns arrow's size in case you need it for positioning purposes.

***Public Methods – Graphing***

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

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

Default: 5.

For example, in *Function Grapher in ActionScript 3.0 – Template 1,* we do want to graph three functions so we don't have to call the method. If you want to display more than five graphs at a time, you should call the method with an appropriate parameter.

Before you can make any of the graphing methods work, you have to set x and y ranges (or, in general, the ranges for your horizontal and vertical variables), in order for your instance of GraphingBoard to know how to translate functional values to pixel values and vice versa. Once you decided on the ranges for x and y (based on the user's input or your own assignment), you have to call the method:

- ***instance*.setVarsRanges(a:Number,b:Number,c:Number,d:Number): void**

Default: no range is set until you call the method.

In *Function Grapher in ActionScript 3.0 – Template 1,* we want to open with the standard range -10, 10, -10, 10. Hence, we call for our instance of GraphingBoard named "board":

```
board.setVarsRanges(-10,10,-10,10);
```

Later, within the function makeGraphs after we parse the user's input for x and y ranges and assign values to the range variables xmin, xmax, ymin, ymax, we call:

```
board.setVarsRanges(xmin,xmax,ymin,ymax);
```

Once the variables ranges are set, we can draw the horizontal and the vertical axes via the method:

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

For example in any of the templates we use:

```
board.drawAxes();
```

Graphs of functions or curves are drawn using the method:

- ***instance*.drawGraph(num:int,thick:Number,aVals:Array,c:Number):Array**

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 integer will also determine the depth of the graph being drawn in the internal stacking order of your instance of GraphingBoard. (A graph with a higher number will appear in front of a graph with lower number.) The second parameter will determine the thickness of your graph, in pixels. The third parameter should be an array whose elements are two-element arrays. Each of the two-element arrays represents the x and y coordinates of a point within (or outside) the graphing board. These coordinates are in functional terms; they will be translated to pixel values by the method. For the method to work properly, this array, denoted above by aVals, should consist of consecutive points along a graph or a curve which will be joined by lineal elements to form the actual graph. (Although, the method will join the consecutive points in the array by lineal elements regardless what the points represent.) The last parameter is responsible for the color of the graph. The value for a color should be passed in hex.

Below we illustrate how the method is used in *Function Grapher in ActionScript 3.0 – Template 1*. The portion of the script below comes from within makeGraphs function. Earlier within the function, the x and y ranges, xmin, xmax, ymin, ymax, have been already determined by parsing the user's input. A string variable, sFunction1, and a CompiledObject variable, compObj1, have been declared already.

```
board.setVarsRanges(xmin,xmax,ymin,ymax);

board.drawAxes();

/*
Determing the value of xstep used later for creating an array
of points to be plotted.
*/

xstep=(xmax-xmin)/points;

/*
We are retrieving the formulas that our user
entered for the first function to be graphed.
*/

sFunction1=InputBox1.text;
……

/*
We will compile the formula  using MathParser,
if the user entered it.
*/

if(sFunction1.length>0){

/*
Evoking our MathParser "doCompile" method
to compile the first formula entered by the user.
Recall that the instance of MathParser created above is called
procFun. If an error is found during compiling,
a message is sent to board.ErrorBox and the function quits.
*/

compObj1=procFun.doCompile(sFunction1);

if(compObj1.errorStatus==1){

      board.ErrorBox.visible=true;

      board.ErrorBox.text="Error in f1(x). "+compObj1.errorMes;

      return;

}

/*
If no error is found we create an array of points, f1Array, to be
```

```
      plotted. Each entry of the array consists of a pair of [x,y] values.
      x values are determined by starting from xmin and adding step-by-step
      the value which brings us to the next point on the x axis.
      This value is xstep and depends on the number of points chosen above.
      To obtain to corresponding values of y, we evoke the procFun.doEval
      method.
      */

      for(i=0;i<=points;i++){

f1Array[i]=[xmin+i*xstep,procFun.doEval(compObj1.PolishArray,[xmin+xstep*i]);

      }

            /*
            Observe, that the array of points created above contains
            functional values. Those values will be converted to pixel values
            by "board". board.drawGraph method evoked below will do it for
            us.
             */

          board.drawGraph(1,1,f1Array,0xFF0000);

      }
```

The drawGraph method returns an array. If the tracing cursor is "cross" (the default), the returned array is the original aVals array which was passed to the method with all x and y coordinates of all points translated to their pixel equivalents. Being able to retrieve this translated array in your applet is valuable for tracing purposes. The returned array gives you the consecutive positions for the cross cursor when tracing the graph. If you do not have a tracing mechanism in your applet, you can ignore the array returned by the method. In *Function Grapher in ActionScript 3.0 – Template 2,* we use the returned array to login positions for tracing. Note the following lines within makeGraphs function:

```
f1PixArray=board.drawGraph(1,1,f1FunArray,0xFF0000);

f2PixArray=board.drawGraph(2,1,f2FunArray,0x0000FF);
```

If the cursor is "arrow", drawGraph returns an array of three-element arrays. Each of the three-element arrays gives you a position of the tracing arrow along the curve as well as the arrow's rotation. Again, the returned array serves a possible tracing mechanism. In our template *Parametric Curves in ActionScript 3.0 – Basic,* we used the returned array. Note the line in drawCurve function:

```
arrowPos=board.drawGraph(1,1,fArray,0xFF0000);
```

"arrowPos" is a global array variable used later to trace a curve with the arrow cursor.

### *Public Methods – Clearing the Graphing Board*

To clear the graphs you use the method:

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

The method erases all graphs, the x and y axes, and resets the x and y ranges to undefined. The method does not erase the user's drawing. The latter is accomplished by

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

We separated these two methods since, typically, you want the GRAPH button to clean the graphing board but not to erase the user's drawing. (The user may possibly be experimenting with drawing functions.)

### *Public Methods – Other*

Here are some other possibly useful methods of GraphingBoard class.

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

Use this method if you are not sure what the maximum number of graphs is set to. Similarly:

- ***instance*.getBoardWidth(): Number**
- ***instance*.getBoardHeight(): Number**
- ***instance*.getVarsRanges(): Array**

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. They will work correctly regardless what names you gave your horizontal and vertical variables.

- ***instance*.xtoPix(a:Number): Number**
- ***instance*.ytoPix(a:Number): Number**
- ***instance*.xtoFun(a:Number): Number**
- ***instance*.ytoFun(a:Number): Number**

A couple of testing methods:

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

The method returns "true" if "a" is of the numerical datatype and it is a finite number. Otherwise, the method returns "false".

- ***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. In our templates, you can draw outside the graphing board as the board is masked. But you shouldn't draw objects which are too far away.

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

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

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

### *Public Properties*

The only public property of GraphingBoard (except for those inherited from Sprite) is

- ***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 GraphingBoard methods. You can also apply Flash's TextField class properties and methods to ErrorBox, e.g.:

```
board.ErrorBox.visible=true;

board.ErrorBox.text="Error in f1(x). "+compObj1.errorMes;
```

## bkde.as3.boards.GraphingBoard3D

---

### *Description*

GraphingBoard3D is the main visual class for creating customizable surface graphers: 3D function graphers, parametric surface graphers in rectangular, cylindrical and spherical coordinates. An instance of GraphingBoard3D draws a square graphing board (at runtime), the boxed-style coordinate axes as well as graphs of functions or parametric surfaces. Any instance of GraphingBoard3D contains and controls an error display text field in which messages to the user are displayed. It also contains display text fields in which the ranges of x, y and z can be displayed.

The layout, the colors, and the sizes of all elements in an instance of GraphingBoard3D are easily customizable via instance methods of the class.

GraphingBoard3D extends Sprite. Thus, it inherits from Sprite. In particular, you can control the position of your instance of GraphingBoard3D within the main movie with the Sprite 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 GraphingBoard3D 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.

In *ParametricSurfaces in ActionScript 3.0 – Rectangular Coordinates*, we have an instance of GraphingBoard3D named "board", a child of the main movie. We position "board" within the main movie with:

```
board.x=15;

board.y=55;
```

***Constructor***

The constructor is evoked with the word `"new"` and takes one numerical parameter. The parameter is the size, in pixels, of the square graphing board which will be drawn:

- **new GraphingBoard3D(s:Number);**

In *Parametric Surfaces in ActionScript 3.0 – Rectangular Coordinates*, we create a 330 by 330 graphing board as follows:

```
var board:GraphingBoard3D = new GraphingBoard(330);
```

We store our instance of GraphingBoard3D in a variable called "board" whose datatype is GraphingBoard3D. We pass to the constructor the size of our square graphing board; all other attributes will be set using the methods of the class.

***Public Methods – Graphing Board Appearance***

- ***instance*.setBackground(b:Boolean,c:Number=0xFFFFFF): void**

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

Default: true, white. The default value for the second parameter is supplied in the definition of the method. Thus, you can simply use:

```
instance.setBackground(false);
```

to have no background and create a transparent board. In *3D Function Grapher in ActionScript 3.0 – on Black,* where the instance of GraphingBoard3D is stored in "board", we have:

```
board.setBackground(true,0x000000);
```

We set the background to black.

- ***instance*.setBorder(b:Boolean,c:Number=0x000000,t:Number=1):void**

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

Default (the default values for the last two parameters are supplied in the definition): true, black, 1.

In *3D Function Grapher in ActionScript 3.0 – on Black*, we use:

```
board.setBorder(true,0xFFFFFF,3);
```

We set the border color to white, its thickness to 3. If you want no border you can type:

```
instance.setBorder(false);
```

***Public Methods – Coordinate Axes Appearance***

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

The method sets the color of the axes that will appear in front of a surface. The color should be passed to the method in its hexadecimal form.

Default: black. The default is set by the constructor but it is not supplied in the method's definition. Hence, when you call the method, you need to supply a value for the parameter.

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

The method sets the color of the axes that will appear in the back of a surface.

Default: gray.

In *3D Function Grapher in ActionScript 3.0 – on Black,* we have:

```
board.setFrontAxesColor(0xFFFFFF);

board.setBackAxesColor(0x666666);
```

We set the colors to white and gray, respectively.

***Public Methods – Error Display Text Field***

An instance of GraphingBoard3D controls the text field for displaying error messages to the user. You can enable, disable, and control the appearance, the position, and the format of the error text field with the following methods.

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

The method adds the error display field to the Display List.

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

The method removes the error display field from the Display List.

Default: enabled.

- ***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 GraphingBoard3D; that is, relative to the upper left corner of the square graphing board.

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

Note: The error field does not have to be placed on top of the square serving as our graphing board. The field can be placed anywhere in the main movie although it will remain a child of the GraphingBoard3D instance.

In *3D Function Grapher in ActionScript 3.0 – on Black,* we have:

```
board.setErrorBoxSizeAndPos(290,100,20,30);
```

You can set visual attributes of the error box:

- ***instance*.setErrorBoxBorder(b:Boolean,c:Number=0x000000): void**

The method sets the existence and the color of the border.

Default: true, black. The default value of the second parameter is supplied in the definition of the method.

In *3D Function Grapher in ActionScript 3.0 – on Black* , we have:

```
board.setErrorBoxBorder(false);
```

We opted for no border.

- ***instance*.setErrorBoxBackground(b:Boolean,c:Number=0xFFFFFF):void**

The method sets the existence and the color of the background.

Default: true, white. The default for the latter is supplied in the definition of the method.

In *3D Function Grapher in ActionScript 3.0 – on Black,* we have:

```
board.setErrorBoxBackground(true,0x000000);
```

We chose black background.

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

The parameters determine: the font color and the font size. (For all display fields the font family is Arial.)

Default values: black, 12.

In *3D Function Grapher in ActionScript 3.0 – on Black,* we have:

```
board.setErrorBoxFormat(0xCCCCCC,12);
```

We chose light gray text of size 12.

- ***instance*.setErrorBoxVisible(b:Boolean):void**

The method controls the visibility of the error box. The initial visibility of the error box is set to false. Subsequently, the box is made visible by showError method and invisible by resetBoard method. Thus, the method is rarely used.

- ***instance*.showError(mes:String):void**

The method will make the error box visible and display the message “mes”.

In *3D Function Grapher in ActionScript 3.0 – on Black,* we have the following code within prepGraph function (procFun is an instance of MathParser):

```
compObj=procFun.doCompile(inpString);
if(compObj.errorStatus==1){
      board.showError(compObj.errorMes);
      bIsError=true;
      return;}
```

The code evokes the showError method if a mistake in the user's input is found and the function prepGraph quits.

***Public Methods – Other Display Boxes***

Besides the error display box, an instance of GraphingBoard3D controls the text fields for displaying the current ranges of x, y, and z variables, the labels for the boxed x, y, and z axes, and the box with a message to the user to wait while input is being processed. Neither of the boxes has a border or a background. Otherwise, they can be controlled with similar methods as the error box.

Here are the methods for the axes' label boxes. (The labels read "x", "y", "z", and "xmin", "xmax", "ymin", "ymax", "zmin", "zmax".)

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

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

Default: enabled.

- ***instance*.setLabelsFormat(c:Number,s:Number,b:String="bold"):void**

The first two parameters determine the color and the size of the text. The last which can be "bold" or "normal" determines if the text is in bold. The supplied default is "bold". Default values for the first two are: black, 11. You have to provide values for these two parameters when evoking the method.

In *3D Function Grapher in ActionScript 3.0 – on Black*, you can see:

```
board.setLabelsFormat(0xCCCCCC,11,"normal");
```

The labels are moving dynamically as the surface is rotated so their positions are set by an instance of GraphingBoard3D. Also, their sizes adjust automatically to accommodate the font.

The x, y, and z range display boxes are disabled by default. In a function grapher, for example, you don't need them as the x, y ranges, and possibly the z range, are entered by the user. You enable or disable them by:

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

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

Default: disabled.

The methods add or remove the range boxes from the Display List.

The range display boxes, xRangeBox, yRangeBox, zRangeBox are public properties of an instance of GraphingBoard3D, so you can send text to them as the ranges are determined dynamically based on the user's input. You can set their format via:

- ***instance*.setRangeBoxesFormat(c:Number,s:Number):void**

The method sets the color and the font size for the range boxes.

Default: black, 11.

The range boxes are positioned by default at the bottom corners and at the left upper corner of the graphing board as you see them in *Parametric Surfaces in ActionScript 3 – Rectangular Coordinates* (and other templates for parametric surfaces). If you want to reposition and resize them you can use the following methods that set the boxes positions (relative to the upper left corner of the graphing board), their width and height:

- ***instance*.setXRangeBoxPosAndSize(a:Number,b:Number,w:Number,**
  **h:Number):void**

- ***instance*.setYRangeBoxPosAndSize(a:Number,b:Number,w:Number,**
  **h:Number):void**

- ***instance*.setZRangeBoxPosAndSize(a:Number,b:Number,w:Number,**
  **h:Number):void**

In all parametric surface templates, we use the default positions of the range boxes.

The last of the display text fields controlled by GraphingBoard3D is the box displaying a message to the user to wait while the applet is busy processing input. You can set the text, the position, the format of the box as well as enable or disable it via the following self-explanatory methods.

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

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

The methods add or remove the box from the Display List.

Default: enabled.

- ***instance*.setWaitBoxFormat(c:Number,s:Number,mes:String):void**

The method sets the color of the text, the size of the font, and the specific message.

Default: magenta, 12, "Processing…".

In *3D Function Grapher in ActionScript 3.0 – on Black* you find:

```
board.setWaitBoxFormat(0xFFFF00,12,"Please wait...");
```

- ***instance*.setWaitBoxVisible(b:Boolean):void**

Default: false.

- ***instance*.setWaitBoxPos(a:Number,b:Number):void**

Default: upper right corner.

The size of the box adjusts to the font and message chosen.

***Methods and Properties Used for Graphing***

GraphingBoard3D has two public graphing methods: drawAxes and drawSurface.

- ***instance*.drawAxes(M:Array): void**

The parameter M has to be an array of three-elements arrays; that is, a 3 by 3 matrix which represents a rotation matrix. drawGraph will then draw boxed-style coordinate axes centered within the square graphing board. The method will also control the positions and the visibility of axes' labels. The method can be evoked right after an instance of GraphingBoard3D is created and added to the Display List. The drawAxes method erases a previous view of the axes before drawing a new view.

The size of the cube composed of the boxed axes is calculated by an instance of GraphingBoard3D and is roughly equal to one third of the graphing board size passed to the constructor. You can retrieve the size of the cube via:

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

The second method, drawSurface, will draw a surface only after you decided on a grid or a mesh for your graph and supplied your instance of GraphingBoard3D with pixel coordinates of the surface's nodes. To illustrate the process, we use fragments of code from *Parametric Surfaces in ActionScript 3 – Rectangular Coordinates.* We create there an instance of GraphingBoard3D and store it in a variable "board":

```
var board:GraphingBoard3D=new GraphingBoard3D(330);

this.addChild(board);
……
var size:Number;
……
size=board.getCubeSize();
……
board.nMesh=30;
```

To draw a surface, we begin with an imaginary rectangle whose sides represent the ranges for the parameters t and s. We subdivide the rectangle into a grid of sub-rectangles. The fineness of the grid is determined by the public property

- ***instance*.nMesh: uint**

Default: 20.

In our templates, the mesh is initially set to 30. At each vertex of the grid we will calculate the corresponding functional values x(t,s), y(t,s), z(t,s) (based on the compiled input by the user) and check if all three of them are legal numbers.

These calculations are done by the prepGraph function in the main script. The function parses the user's input and then calculates and logs in with "board" (via board.setPixArray method) the array of nodes for the surface that was entered by the user. After a surface is known to "board", drawSurface method can render different views of the surface and combined with drawAxes will render synchronized views of the whole assembly.

Within the prepGraph function, the local variable "mesh" is set equal to board.nMesh. We see the following code there:

```
        function prepGraph():void {

                ……
                var mesh:Number=board.nMesh;
                ……

                board.resetBoard();

                //……Compiling the user's input.

  /*
  If no mistake was found when compiling the user's input, we use doEval
  method of MathParser to calculate the functional coordinates of the
  nodes. The functional coordinates of the nodes are stored in
  fArray.
  ……
  */

  for(j=0; j<=mesh;j++){

        fArray[j]=[];

        curt=tmin+j*(tmax-tmin)/mesh;

for(i=0; i<=mesh; i++){

        curs=smin+i*(smax-smin)/mesh;

        cury=procFun.doEval(compObj3.PolishArray,[curt,curs]);
        curz=procFun.doEval(compObj1.PolishArray,[curt,curs]);
        curx=procFun.doEval(compObj2.PolishArray,[curt,curs]);
```

```
            fArray[j][i]=[curx,cury,curz];
            ……

  }

      }
   ……

/*

We calculated functional coordinates of the nodes. Now we are going to
compute the pixel coordinates of the nodes and store them in pArray. We will
also check which nodes have coordinates which are all legal numbers.
This information -- 0 for yes, 1 for no -- will also be stored in
pArray as the fourth coordinate of each node.
The pArray will be then passed to "board" via board.setPixArray.

(The conversion formulas, xtoPix, ytoPix, ztoPix, that you can see later in
the script depend, of course, on the size of the cube; that is, on the value
of our variable “size”.)
*/

         for(j=0; j<=mesh;j++){

               pArray[j]=[];

              for(i=0; i<=mesh;i++){

                    pArray[j][i]=[];

                    pArray[j][i][0]=xtoPix(fArray[j][i][0]);

                    pArray[j][i][1]=ytoPix(fArray[j][i][1]);

                    pArray[j][i][2]=ztoPix(fArray[j][i][2]);

      if(isLegal(fArray[j][i][0]) && isLegal(fArray[j][i][1]) &&
                                                    isLegal(fArray[j][i][2])){

                                  pArray[j][i][3]=0;

                    } else { pArray[j][i][3]=1;}

              }

         }

         fArray=[];

      }

      board.setPixArray(pArray);

      ……

       }
```

In the last line of this code fragment, we used the method of GraphingBoard3D, setPixArray, and passed to board an array of nodes in their pixel values as well as the information which node has all three coordinates that are legal numbers. This information is necessary for the drawSurface method to render the surface that you want and then different views of the surface while rotating it.

- ***instance*.setPixArray(A:Array): void**

"A" should be an array of nMesh+1 arrays, each of those nMesh+1 long. Each element of the latter array is a four element array. The first three elements represent coordinates of a node in pixels; the fourth element is 0 or 1 depending if the node's coordinates are legal or not.

Once the setPixArray method has run, we can successfully call:

- ***instance*.drawSurface(M:Array): void**

The parameter M represents a 3 by 3 rotation matrix. In order for the rotation of a surface and the boxed axes to be synchronized, drawAxes and drawSurface should be called together with the same rotation matrix. The drawSurface method erases a previous view of the current surface before drawing a new view.

In procInput function, we see:

```
function procInput():void {

              prepGraph();

              if(bIsError){return;}

              drawGraph(iniMatrix);

}
```

where drawGraph is:

```
   function drawGraph(M:Array):void {

       board.drawSurface(M);

       board.drawAxes(M);

       ……

}
```

To clear the graphs you use the method:

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

The method erases all graphs, makes all display boxes invisible, and resets all arrays to empty. You should call the method before drawing a new surface entered by the user. You shouldn't use it before rendering new views of a previously compiled surface as the method will erase the array of nodes. In our templates, the method resetBoard is called by RESET button and by prepGraph function.

A few important public properties and methods related to graphing:

- ***instance*.nOpacity: Number**

A number between 0 and 1 which determines the opacity of the surface rendered. In our templates, the value is initially set to 1 (the default). The opacity buttons change the value of the property and redraw a surface.

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

The method sets the color of the wireframe on your surface.

Default: dark gray.

- ***instance*.bShowFrame: Boolean**

The property determines if the wireframe is shown or not.

Default: true.

It is an easy exercise to augment one of the templates by adding a button which toggles the visibility of the wireframe.

- ***instance*.setColorType(t:String,c:Number=0xFFFF00): void**

The method determines the coloring type for your graphs. Available choices are: "parametric" (default) where tiles are colored based on their position in the 3D space; "function" appropriate for graphs of functions – coloring is based on the x and y position of the tile in the rectangular domain of a function, unrelated to the value of the function; "solid" – all tiles are of the same color. If you choose "solid" as your first parameter, the second parameter sets the solid color of your choice. (Default: yellow.) In *3D Function Grapher in ActionScript 3.0 – on Black,* we have:

```
board.setColorType("function");
```

- ***instance*.fLength: Number**

The distance of the camera from the surface.

Default: 2000. The higher the value the less distortion for perspective.

### *Other Public Methods*

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

The method returns the size of the square graphing board, the same number that was passed to the constructor.

- ***instance*.combineRGB(red:Number,green:Number,blue:Number): Number**

This method eventually will be moved to an upcoming class ColorUtils as a static method. For now, it resides in GraphingBoard3D as an instance method. You pass three values between 0 and 255 to the method which then returns the number representing the corresponding color that Flash understands.

Finally, if you want to remove an instance of GraphingBoard3D at runtime, you should call

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

The method removes all drawings in children of your instance of GraphingBoard3D, and sets all the objects created by the instance to null.

### *Public Properties*

We have discussed all the public properties above. Let's just list them for the record. They are:

- ***instance*.xRangeBox**
- ***instance*.yRangeBox**
- ***instance*.zRangeBox**

These text fields are public so you can easily send text to them. In *Parametric Surfaces in AcctionScript 3 – Rectangular Coordinates,* within prepGraph function, we send current values of the x, y, and z ranges to be displayed in the boxes (It should be pointed out that x, y and z coordinates for the user translate into z, x, and y coordinates internally to make them consistent with Flash's coordinate names.)

```
board.yRangeBox.text="ymin="+String(Math.round(xmin*100)/100)+
                              "\n"+"ymax="+String(Math.round(xmax*100)/100);

board.xRangeBox.text="xmin="+String(Math.round(zmin*100)/100)+
                              "\n"+"xmax="+String(Math.round(zmax*100)/100);

board.zRangeBox.text="zmin="+String(Math.round(ymin*100)/100)+
                              "\n"+"zmax="+String(Math.round(ymax*100)/100);
```

You can format, position and size the range display boxes using the GraphingBoard3D methods discussed above or directly, using TextField class methods.

- ***instance*.nMesh: uint**

Default: 20.

- ***instance*.bShowFrame: Boolean**

The property determines if the wireframe is shown or not.

Default: true.

- ***instance*.nOpacity: Number**

A number between 0 and 1 which determines the opacity of the surface rendered. In our templates the value is initially set to 1 (the default). The opacity buttons change the value of the property and redraw a surface.

- ***instance*.fLength: Number**

The distance of the camera from the surface. Default: 2000.

## bkde.as3.utilities.MatrixUtils

---

The class will be enriched by more methods. For now, it contains a few static methods directly related to our 3D drawing.

- **MatrixUtils.MatrixByVector(A:Array,B:Array): Array**

The method performs matrix multiplication of a 3 by 3 matrix A and a 3D vector B. That is, A is an array of three-element arrays (representing rows), B is a three element array. The method returns a 3D vector; that is, a three-element array.

- **MatrixUtils.MatrixByMatrix(A:Array,B:Array): Array**

The method performs matrix multiplication of a 3 by 3 matrix A and a 3 by 3 matrix B. That is, A and B are both arrays of three-element arrays (representing rows). The method returns their matrix product.

- **MatrixUtils.projectPoint(p:Array,flen:Number): Array**

p is a three-element array; that is, a point in xyz-space. The method returns a projection of the point p onto the xy-plane assuming the camera is located in front of the xy-plane, on the positive z axis, at the distance flen.

- **MatrixUtils.rotMatrix(a:Number,b:Number,c:Number,theta:Number): Array**

The method calculates and returns the rotation matrix about the axis [a,b,c] by the angle theta (in degrees).

## bkde.as3.utilities.StringUtils

---

Similarly, this class will be enriched by more methods. For now, it contains two static methods which are helpful for processing the user's input.

- **`StringUtils.isBlankText(s:String): Boolean`**

The method returns "true" if the string s is empty or consists of blank spaces only. Otherwise, the method returns "false".

- **`StringUtils.removeSpaces(s:String): String`**

The method removes all blank spaces from s and returns the resulting string.

## 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".

In our template *Parametric Curves in ActionScript 3.0 – Basic*, we create a slider that is 250 pixels long and has a triangular knob:

```
var hsSlider:HorizontalSlider=new HorizontalSlider(250,"triangle");

addChild(hsSlider);

hsSlider.x=30;
```

```
hsSlider.y=435;
```

We store our instance of HorizontalSlider in a variable called "hsSlider" whose datatype is HorizontalSlider. As with all display objects, we have to add our slider to be a child of an object already on the Display List, in this case to the main movie. We set the position of our slider using properties inherited from Sprite.

***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.

The color should be passed in hex, as in *Parametric Curves in ActionScript 3.0 – Basic* :

```
hsSlider.changeKnobColor(0xCC0000);
```

- ***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 is an excerpt from *Parametric Curves in ActionScript 3.0 – Basic.* (The function KnobPosToFun translates the horizontal position of the knob in pixels to the equivalent value of the parameter "t". The function has been defined earlier in the script. The same is true for the function getArrowPos. KnobBox is a dynamic text field residing on the Stage in which the current value of "t" is displayed as the knob moves.) The fragment of code below uses the property of HorizontalSlider, isPressed discussed in Properties section.

```
/*
The event listener is assigned to the Stage which listens to the
movement of the mouse and responds to it provided the slider's knob
is pressed. This event listener causes the arrow to move along
a curve.
*/

stage.addEventListener(MouseEvent.MOUSE_MOVE,handleMove);

function handleMove(e:MouseEvent):void {

      if(hsSlider.isPressed){

         var curKnobPos:Number;

         var curArrowPos:Array;

            curKnobPos=KnobPosToFun(hsSlider.getKnobPos());

            if(isTRangeSet){

                  if(isGraphToTrace){

            curArrowPos=getArrowPos(curKnobPos);

            board.setArrowPos(curArrowPos[0],curArrowPos[1],curArrowPos[2]);

            board.arrowVisible(true);

                  }

            KnobBox.text=String(Math.round(curKnobPos*1000)/1000);

            } else {

                  KnobBox.text="";
```

```
            }
            e.updateAfterEvent();
        }
}
```

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.

## bkde.as3.utilities.VerticalSlider

---

***Description***

The class is nearly identical to 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.

---

*August 16, 2007*