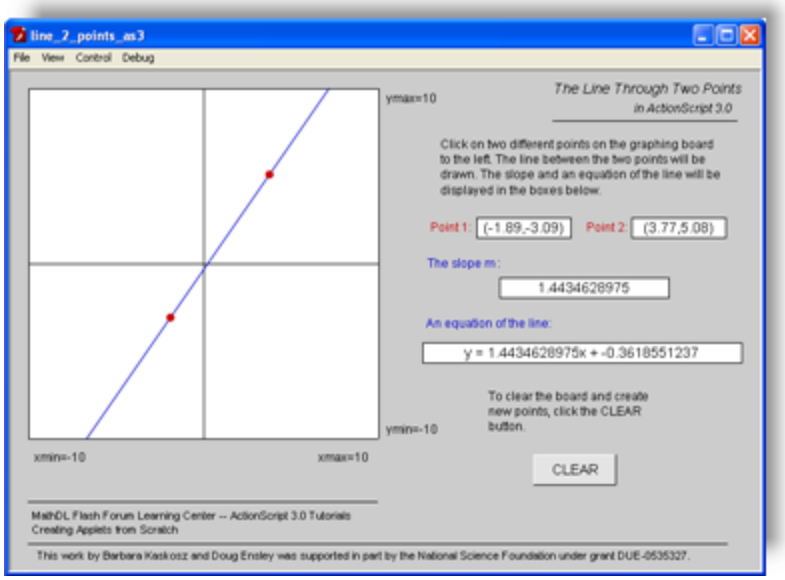
Mathematics applets from scratch

Drawing in a window at runtime

by Doug Ensley, Shippensburg University
and Barbara Kaskosz, University of Rhode Island



In this tutorial, we will make the application shown on the right. This applet allows the user to click on two points on a coordinate grid and places dots at each point clicked. On the click of the second point, the program draws the line determined by the two points and displays the equation of the line. This applet extends the basic functionality introduced in the "LineSegment" tutorial and furthermore provides an illustration of the pedagogical value to this type of interface.

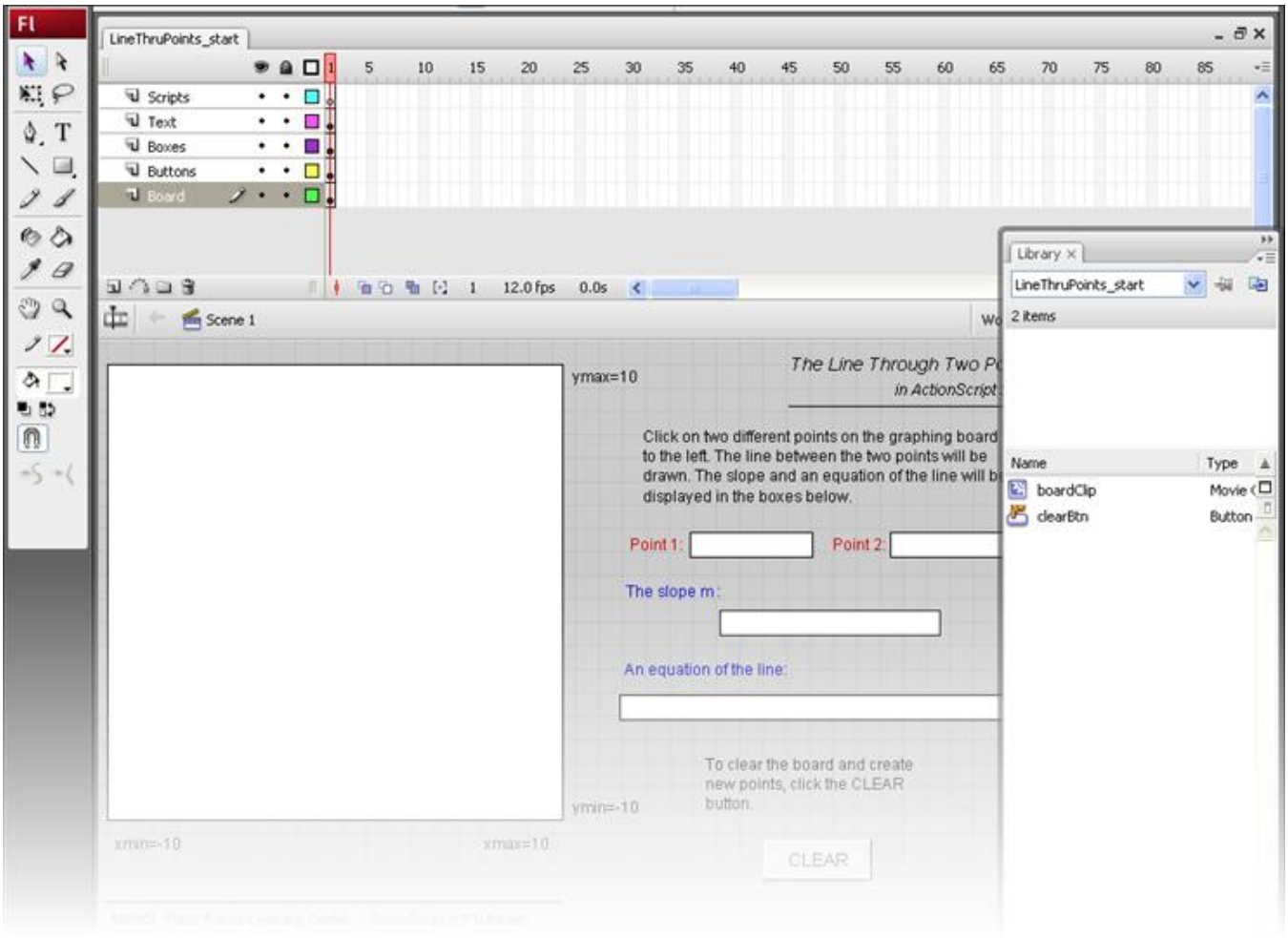
Step 1: Create the stuff on the stage. (Note that if you do not feel excited about creating all of these objects on your stage, you can skip this step by opening the file **LineThruPoints_start.fla**.)

Place some static text to label the button and output boxes as shown in the example above. To be consistent with our script, we need to set up the following elements on the stage: (Note that our starter file uses several layers on the timeline for organizational purposes. This is optional, of course, but the list below reflects exactly how the starter file is set up.)

- All of the static text shown in the example above in a layer called **Text**.
- All of the dynamic textboxes shown in the example above in a layer called **Boxes**. Use the instance names **point1Box**, **point2Box**, **slopeBox**, and **equationBox** for consistency with the scripts to be discussed later.
- A single button with instance name **butClear** as shown. (Copy a button from a previous application or make your own!) For consistency, place this button on a layer called **Buttons**.
- A 350 by 350 square with black border and white fill positioned roughly as shown. Convert this graphic to a movie clip (using Modify > Convert to Symbol or F8) namedboardClip. Give the clip the instance name **mcBoard**, and for consistency, place it on a layer called **Board**.
- In preparation for our subsequent work, create one more layer and call it **Scripts**.

At this point, the screen should look more or less like the screen shot below. Note that the button and the graphing board movie clip are the only two entries in the **Library** panel at this time. Save your file using a descriptive name like **myLineThruPoints.fla** before going on.

This activity is based on material posted on the *Math Flash Forum* at the MathDL Digital Classroom Resources. It is created with partial of NSF DUE-0535327 and the Mathematical Association of America.



Step 2. Create Shapes for axes and line and Sprites for points. Open the **Actions** panel for frame 1 of the "Scripts" layer, and type the following code. (If you do not have the time or inclination to type, you will find the code in the **LineThruPointsScript.txt** file in the **RuntimeLines** folder.)

The first five lines set the range of the coordinate system and the size of the board. We put these at the top so they are easy to modify to correspond to any changes on the stage. After that, we create the coordinate axis and the line to be drawn as Shapes and the two points to be plotted as Sprites (in case we want to interact with the points in future extensions). These are created in order so that the automatic depth assignments (low to high) agree with the layout (back to front) we desire. Note that only the coordinate axis is drawn (i.e., assignments made to the graphics property) at this time. The other objects, although attached to the graphing board, have no graphics yet and so do not show up at runtime. This is beneficial since clearing and editing graphics is much easier than managing an objects list of children. We finally note that we use a function to draw the coordinate axis simply because there are some natural extensions we will address later.

```
var nXmin:Number=-10;
var nXmax:Number=10;
var nYmin:Number=-10;
var nYmax:Number=10;
```

```
var nSize:Number = mcBoard.width;

var shAxis:Shape = new Shape();
mcBoard.addChild(shAxis);
drawAxis();

var shLine:Shape = new Shape();
mcBoard.addChild(shLine);

var spPoint1:Sprite = new Sprite();
mcBoard.addChild(spPoint1);

var spPoint2:Sprite = new Sprite();
mcBoard.addChild(spPoint2);

function drawAxis():void {
      shAxis.graphics.lineStyle(2, 0x999999);
      shAxis.graphics.moveTo(1, mcBoard.height/2);
      shAxis.graphics.lineTo(mcBoard.width - 1, mcBoard.height/2);
      shAxis.graphics.moveTo(mcBoard.width/2, 1);
      shAxis.graphics.lineTo(mcBoard.width/2, mcBoard.height - 1);
}
```

Step 3. Add listener for mcBoard to draw points. We next add code that asks the **mcBoard** to listen for mouse clicks and draw either the `spPoint1` or the `spPoint2` sprite. To know which point to place, we simply count the points placed with the `numPoints` variable. After the second point is drawn, we draw the line using a function to be defined in the next step.

```
var numPoints:Number = 0;

mcBoard.addEventListener(MouseEvent.CLICK, placePoint);

function placePoint(evt:MouseEvent):void {

      var x1fun:Number;
      var x2fun:Number;
      var y1fun:Number;
      var y2fun:Number;

      if (numPoints == 0)     {

            spPoint1.graphics.lineStyle(0,0x000000);
            spPoint1.graphics.beginFill(0xCC3333);
            spPoint1.graphics.drawCircle(0,0,5);
            spPoint1.graphics.endFill();

            spPoint1.x = mcBoard.mouseX;
            spPoint1.y = mcBoard.mouseY;

      // Convert from pixels to coordinate system & display on stage.
            x1fun=Math.floor(xtoFun(spPoint1.x)*100)/100;
            y1fun=Math.floor(ytoFun(spPoint1.y)*100)/100;
```

```
            point1Box.text="("+String(x1fun)+","+String(y1fun)+")";

            numPoints = 1;    // Now there's one point placed.
        }
        else if(numPoints == 1)
        {
            spPoint2.graphics.lineStyle(0,0x000000);
            spPoint2.graphics.beginFill(0xCC3333);
            spPoint2.graphics.drawCircle(0,0,5);
            spPoint2.graphics.endFill();

            spPoint2.x = mcBoard.mouseX;
            spPoint2.y = mcBoard.mouseY;

        // Convert from pixels to coordinate system & display on stage.
            x2fun=Math.floor(xtoFun(spPoint2.x)*100)/100;
            y2fun=Math.floor(ytoFun(spPoint2.y)*100)/100;
            point2Box.text="("+String(x2fun)+","+String(y2fun)+")";

            numPoints = 2;    // Now there are two points placed,

            drawLine(); //   so we can draw the line.
        }
    }
```

Note that the function above uses two auxiliary functions called `xtoFun` and `ytoFun`. This illustrates an important issue with this type of application.

The definitions of these two functions follow. They simply take a pixel value, *a*, for *x* or *y* relative to **mcBoard** and convert it to the corresponding coordinate system value in the range [`nXmin`,`nXmax`] or [`nYmin`, `nYmax`].

```
function xtoFun(a:Number):Number {
      var xconv:Number=nSize/(nXmax-nXmin);
      return a/xconv+nXmin;
}

function ytoFun(a:Number):Number {
      var yconv:Number=nSize/(nYmax-nYmin);
      return nYmax-a/yconv;
}
```

Step 4: Write the `drawLine` function. This function does all of the "heavy lifting" for this application. The issue here is more difficult than the simple line segment for two reasons. First, we must run the line (a Shape named `shLine`) to the edges of **mcBoard**, so there is a mathematics problem to solve to find the two points on the boarder of the box (we call these `(xbeg,ybeg)` and `(xend,yend))` that we can use to construct a line that will also go through the two user-clicked points. The second issue is a purely pedagogical one: We would like the user to see the equation of the line in terms of the normal coordinate system, which is not what Flash uses for its stage coordinates. Hence, there are some conversions involved. To keep matters straight, we use

the variablenames `display_m` and `display_b` for the slope and y-intercept we display to the user. You will notice that we do not use these values for anything else.

```
function drawLine():void {
      // Set line thickness and color of the line to be drawn.
      shLine.graphics.lineStyle(2, 0x33333FF);

      // Local variables for the points on mcBoard to save typing.
      var x1:Number = spPoint1.x;
      var x2:Number = spPoint2.x;
      var y1:Number = spPoint1.y;
      var y2:Number = spPoint2.y;

      var m:Number;
      var display_m:Number;
      var display_b:Number;

      var xbeg:Number;
      var xend:Number;
      var ybeg:Number;
      var yend:Number;

      if(x1==x2 && y1==y2){
            slopeBox.text="";
            equationBox.text="One point doesn't determine a line.";
            return;
      }

      if(x1==x2){
            slopeBox.text="undefined";
            equationBox.text="x"+" = "+String(xtoFun(x1));
            shLine.graphics.moveTo(x1,0);
            shLine.graphics.lineTo(x1,nSize);
            return;
      }

      if(y1==y2){
            slopeBox.text="0";
            equationBox.text="y"+" = "+String(ytoFun(y1));
            shLine.graphics.moveTo(0,y1);
            shLine.graphics.lineTo(nSize,y1);
            return;
      }

      m = (y1 - y2)/(x1 - x2);      // The slope in stage coordinates.

      display_m = -m;
      display_b = ytoFun(y1) + m * xtoFun(x1);

      slopeBox.text = String(Math.floor(10000*display_m)/10000);
      equationBox.text = "y = " + slopeBox.text + " x  +  " +
String(Math.floor(10000*display_b)/10000);
```

```
        /* The next block of code finds the y-coordinates that the line
        would have at x=0 and x=nSize.  We then test to see if these
        y-coordinates fit on the board.  If not, we know the correct
        points to use are along the top or bottom of the board.
        */

        xbeg = 0;
        ybeg = y1 + m * (xbeg - x1);
        xend = nSize;
        yend = y1 + m * (xend - x1);

        if(ybeg>nSize){
              ybeg = nSize;
              xbeg = x1 + (ybeg - y1)/m;
        }

        if(ybeg<0){
              ybeg=0;
              xbeg=x1 + (ybeg - y1)/m;
        }

        if(yend>nSize){
              yend = nSize;
              xend = x1 + (yend - y1)/m;
        }

        if(yend<0){
              yend=0;
              xend=x1 + (yend - y1)/m;
        }

        shLine.graphics.moveTo(xbeg,ybeg);        // We draw the line!
        shLine.graphics.lineTo(xend,yend);
  }
```

Step 5 . Add functionality for the clear button. This is a simple piece of code that clears all of the graphics attached to our sprites and shapes. When the CLEAR button (instance name **btnClear**) is clicked, the function `clearLine` is called. That is, the dots are removed and all the text boxes cleared. The global variable `numPoints`, which counts the number of clicks, is reset to 0.

```
  btnClear.addEventListener(MouseEvent.CLICK, clearLine);

  function clearLine(evt:MouseEvent):void {

        numPoints=0;

        shLine.graphics.clear();
        spPoint1.graphics.clear();
        spPoint2.graphics.clear();

        point1Box.text="";
        point2Box.text="";
```

```
        slopeBox.text="";
        equationBox.text="";
    }
```

Files and Enhancements

The complete source file for this project is located in the **RuntimeLines** folder under the name **LineThruPoints.fla**.

Here are some suggestions for enhancements.

1. Change this to a quiz format in which two random points are selected by the script and the student must enter the correct equation or some variation of this.
2. Add a "grid" to the drawing board and make points snap to the grid.
3. Make the applet start with two points already selected and graphed. First place this "initialization data" at the beginning of the fla file and then try to pull it from an external XML file.
4. Make the points draggable once they are placed.