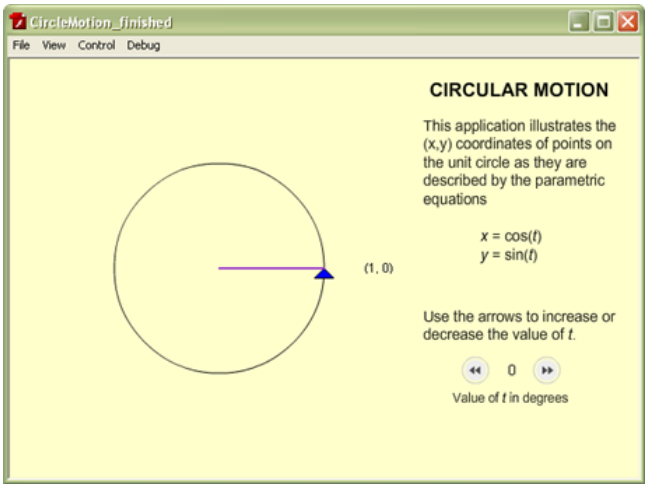
Flash basics for mathematics applets

Circular Motion. Working with movieClips created on the stage

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

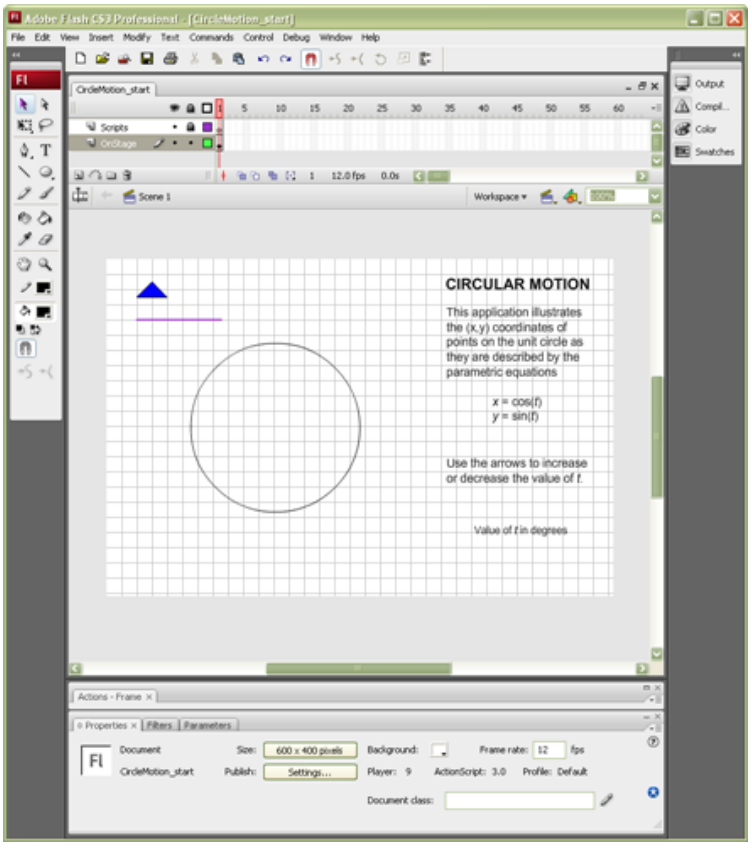


The phrase "movie clip" is by far the most unfortunate term used by *Flash*. In *Flash*, movie clips are simply a class of objects with inherent (but extensible) properties and methods. It is a generic construct that need not involve motion or sound or any other attribute associate with "movies."

In this tutorial, we will make the application shown on the left. This applet shows the coordinates on a unit circle as a parameter *t* is changed by user input. The cursor on the circle and the radius from the circle center to the cursor are *Flash* movie clips. When the user changes the value of *t*, the *x*- and *y*-coordinates of the cursor are changed and both cursor and radius are rotated in a corresponding manner. In addition, we add the keyboard arrow keys as alternative controls for the action on the screen.

Open the Starter file and add dynamic textboxes and control buttons

We will begin with the file **CircleMotion_start.fla**, which has already in place the background text, a static circle in the appropriate position and size, and movieclips with instance names **mcArrow** and **mcLine**. You can click on each of these to see their attributes in the **Property** panel before we continue. At this point, the *Flash* file should look like the screenshot on the right. Save your file with a new name such as **myCircleMotion.fla**.

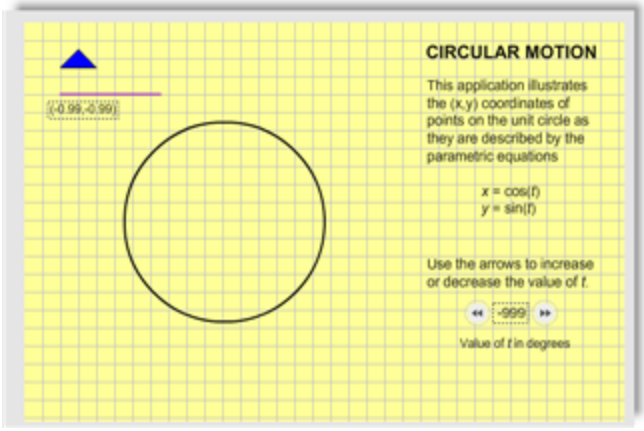


In the **Timeline** panel, you will notice that we have created two layers called "Scripts" and "OnStage." While we continue to add items to the **Stage**, we will leave the Scripts layer locked.

Next create a dynamic textbox with instance name **txtDegrees** and with initial value "-999" to assure that it is the right size for the largest possible text values. Position this textbox directly over the static

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.

text label “Value of t ...” and add to the upper left corner of the stage a dynamic textbox with instance name **txtCoords** and initial value “(-0.99, -0.99),” the widest text that the textbox will have to hold. This is where we will update the current (unit circle) coordinates of the cursor as the value of *t* is changed.



We will now need simple buttons to increase/decrease the value of *t*. Although we can make these from scratch, it is sometimes easier just to use predefined Flash buttons. Under the Windows menu item, select Common Libraries > Buttons. For this applet, we want buttons that reflect “forward” and “back” like those typical of video or music playing programs. Double click on the folder[1] “playback flat,” click on “flat grey back,” drag it onto the stage to the left of the dynamic textbox, and give this instance the name **btnLess** in the **Properties** panel. Similarly, click on “flat grey forward,” drag it onto the stage to the right of the dynamic textbox, and give this instance the name **btnMore.** At this point, your project should look like the screenshot above:

Add the script

To create the scripts for this applet, first click on the layer called **changing** in the **Timeline** panel, and choose Insert > Timeline > Layer. Double click the new layer’s label to give the new layer the name **scripts**. Click Frame 1 of the **scripts** layer, and open the **Actions** panel in preparation of writing some code. Here is the full script for this movie. Obviously, you do not need to type comments in the code. If you do not wish to type *any* of this, you can open the file **CircleMotionScript.txt**, and copy and paste the script into your movie at this time.

```
// Initialize starting value for t (in degrees) and the change in t for each
//  press of the button.

var degree:Number = 0;
var degChange:Number = 1;

// The triangle clip mcArrow can be resized by scaling its
//   height and width separately.

mcArrow.scaleX = 0.5;
mcArrow.scaleY = 0.5;

/* The coordinates of the center of the circle and the radius of the circle
       keep the motion of the clips aligned with the picture on the stage.  The circle
       on stage must have width 200 and coordinates (100,100).
*/

var circleX:Number = 200;
var circleY:Number = 200;
var circleR:Number = 100;

// The radial line segment mcLine should have its registration point
// (i.e., its left endpoint) on the center of circle.

mcLine.x = circleX;
mcLine.y = circleY;

// This function updates the position and rotation of mcArrow, the rotation of
```

[1] If your Buttons library does not have these exact items, shop around to find any forward and backward buttons.

```
//  mcLine, an the position and content of the dynamic textbox txtCoords.

function updateArrow(t:Number):void {

	var radianAngle:Number = t*Math.PI/180.0; // Convert t degrees to radians.

	// Update position of mcArrow

	mcArrow.x = circleX + circleR*Math.cos(radianAngle);
	mcArrow.y = circleY - circleR*Math.sin(radianAngle);

	// Update rotation of mcArrow (note that Flash measures angles clockwise)

	mcArrow.rotation = -t;

	// Update rotation of mcLine

	mcLine.rotation = -t;

	/*	Update position of the dynamic textbox txtCoords.
		Note the adjustment for the height and width of the textbox
		and the larger radius circle in which the textbox must travel.
		The offset and the scaling of the circle require some trial and error
		unless you have been very scientific with your choice of textbox size and
		your placement of the circle on the stage.
	*/

	txtCoords.x = circleX - 35 + 1.5*circleR*Math.cos(radianAngle);
	txtCoords.y = circleY - 10 - 1.5*circleR*Math.sin(radianAngle);

	// Build the string showing the x- and y-coordinates to two decimal places,
	//  and put this into the textbox.

	txtCoords.text = "(" + String(Math.round(100*Math.cos(radianAngle))/100) + ", "
+ String(Math.round(100*Math.sin(radianAngle))/100) + ")";

	// Put the current (degree) value of t into the appropriate textbox.

	txtDegrees.text = String(t);
}

// When btnLess is clicked, reduce that value of t by the preset amount.
//  Make this change mod 360 to keep the angle measures between -359 and 359.

btnLess.addEventListener(MouseEvent.CLICK, decreaseAngle);

function decreaseAngle(evt:MouseEvent):void {
	degree = (degree - degChange) % 360;
	updateArrow(degree);
}

// When btnMore is pressed, increase that value of t by the preset amount.

btnMore.addEventListener(MouseEvent.CLICK, increaseAngle);

function increaseAngle(evt:MouseEvent):void {
	degree = (degree + degChange) % 360;
	updateArrow(degree);
}

// Call this function initially so that the clips are placed in the correct
//  positions original.  This line is not necessary if you manually line up
//  all the clips on the stage before building the swf file.

updateArrow(degree);
```

At this point, you can use Control > Test Movie to test your movie. The buttons for changing the angle have been implemented, but we have not added the option for keyboard input. Go back to the **Actions** panel to add the next few lines of code to add this functionality.

```
	stage.addEventListener(KeyboardEvent.KEY_DOWN,keyPressed);
```

```
function keyPressed(evt:KeyboardEvent):void {
       if (evt.keyCode == Keyboard.LEFT) {
       degree = (degree - degChange) % 360;
              updateArrow(degree);
    }
       if (evt.keyCode == Keyboard.RIGHT) {
       degree = (degree + degChange) % 360;
              updateArrow(degree);
    }
}
```

Save and test your movie, and you will see a full working version of the applet we described at the beginning of this tutorial.

Additional resources and enhancements

To see the source code for the applet we made for this tutorial, open the file **CircleMotion.fla** in Flash. There are several worthwhile enhancements one can make to this project. Here are a few in order of level of difficulty. (The first two enhancements use ideas that you can borrow from the **Introduction** presentation.)

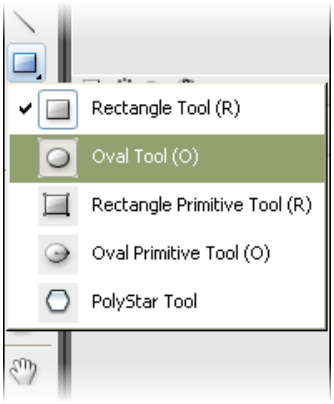
- Add smooth motion of the trace cursor while the buttons are being held.
- Create all of the graphics as sprites at run time. This will alleviate painstaking alignment between stage objects and runtime objects.
- Make the mathematics more interesting. For example, one could add a short tangent line segment and report the slope of the tangent as well as the coordinates of the points – will the students see a connection?

ADDENDUM: Working on the Stage

We give here the instructions for creating the background graphics, textboxes and buttons for this application. In other words, these are the steps involved in creating the file **CircularMotion_start.fla**.

Create background graphics and textboxes

Start with a new project in the *Flash* authoring environment. You will need the **Tools** panel, the **Properties** panel and the **Stage** to be visible. All other panels can be closed if you wish. Click on an empty section of the stage, and in the **Properties** panel choose a nice background color to contrast appropriately with your text. Next add static textboxes to hold the title, the instructions, and the phrase "Value of t (in degrees)," all positioned as shown on the initial screenshot for this activity. Save your file as **myCircleMotion.fla**.



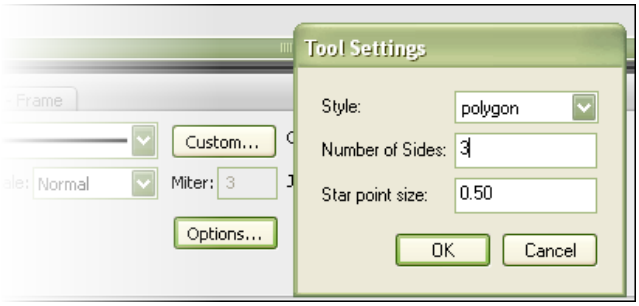
Now draw a circle by selecting the **Oval tool** (shown on the left) from the **Tools** panel and setting the pencil color to black and the fill color to "no fill" as shown on the right. In addition, turn on Object Drawing mode and Snap to Objects by setting the icons at the bottom of the Tools panel to look like the picture on the right. Now clicking and dragging on the stage while holding down the Shift key will produce a circular shape. Open the **Properties** panel to position your circle on the stage. In the sample provided, the height and width of the circle are both 200.0, the *x*-coordinate is 100.0, and the *y*-coordinate is 100.0. These coordinates reflect the leftmost *x*-coordinate and the uppermost *y*-coordinate sof the circle. The important thing is to make note of the radius of your circle (i.e., half of the height/width of the circle) and the coordinates of the

center of your circle, which can be computed by adding the radius of the circle to the coordinates shown in the **Properties** panel.

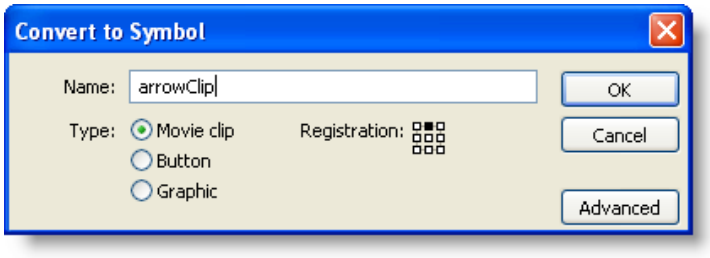
Create the movie clips

In this applet there are two moving objects reacting to user input on the stage. These objects will be positioned by the script dynamically, so we will simply create them in the upper left corner of the stage for now. These two objects are examples of *Flash* movie clips. The term "movie clip" is a very unfortunate name for this very important class of *Flash* objects. One can think of a movie clip as simply a type of object that is to be manipulated on the stage by the script.

The first of our movie clips will be the triangular cursor that traces a point around the circle. We will use the PolyStar Tool. (See the bottom leftmost picture on the previous page to see where this is located.) With this tool selected (and confirm that the Object Drawing mode is still on), go to the **Properties** panel to set the stroke width to 1, stroke color to black and the fill color to blue. In addition, click on the "Options" button to set the number of sides to 3, as shown on the right.



Finally, use the **Selection tool** to (single) click on the triangle and choose Convert to Symbol from the Modify menu. Select "Movie clip" as the type of symbol, assign a descriptive name such as **ArrowClip**, and set the registration point to be the top center of the triangle (i.e., the topmost vertex). The completed dialog box is shown on the left. The registration point of the movie clip are very important since it is this point that is referenced for the location of the clip and it is about this point that the clip will be rotated. Click on the instance of the clip on the stage, and give it the instance name **mcArrow**.



The second moving object on the screen is the radius of the circle that rotates as the parameter t is modified. For this purpose, we will simply draw a horizontal line segment and convert it to a movie clip with registration point on the left end, and then we will position the clip so that this end is at the center of the circle. To accomplish this, use the **Line tool** from the **Tools** panel to draw in a blank region of the stage a horizontal line. Then use the **Properties** panel to give the line segment a contrasting color (like magenta), stroke width 2, and a length that is equal to the radius of the circle. Select this drawing and choose Modify > Convert to Symbol to convert it to a movie clip with name **LineClip** and *registration point on the left end*. Finally, use the **Selection tool** to click on the instance of the clip on the stage, and in the **Properties** panel give it the instance name **mcLine**. Note that you do not need to position the clip relative to the circle as we will do that within the script. The stage should now look like the screenshot for the file **CircleMotion_start.fla** shown on the first page of this guide.

To learn more about Object Drawing mode, we recommend the tutorial "Using the drawing tools" in the Adobe CS3 Video Workshop, http://www.adobe.com/designcenter/video_workshop/about.html