

Flash Simulation for Developers: Simple Beam Uniformly Distributed Load

By Michael Lively and Seyed Allameh
Northern Kentucky University

This tutorial demonstrates how Macromedia Flash can be used to create simulations for engineering technology. The Simple Beam (Uniformly Distributed Load) simulation was developed for the course Strength of Materials. The simulation's unique graphical interface was created to enhance student learning, and is presently being tested in the classroom. In this tutorial, we demonstrate a number of programming techniques not commonly found in any one place in the literature. We have assumed that the reader is proficient with the Flash interface and basic action scripting.

Design and Graphical Elements

An important aspect driving the complexity of our design is the task of creating a simulation that gives real physical numbers and a visual experience to represent those physical numbers. Taking this into consideration requires that some error handling be created to handle the nonlinearity of the cubic deflection equation.

Our design creates an interactive environment which graphs deflection, shear, and bending moment as a function of beam length for a simple beam with a uniformly distributed load.

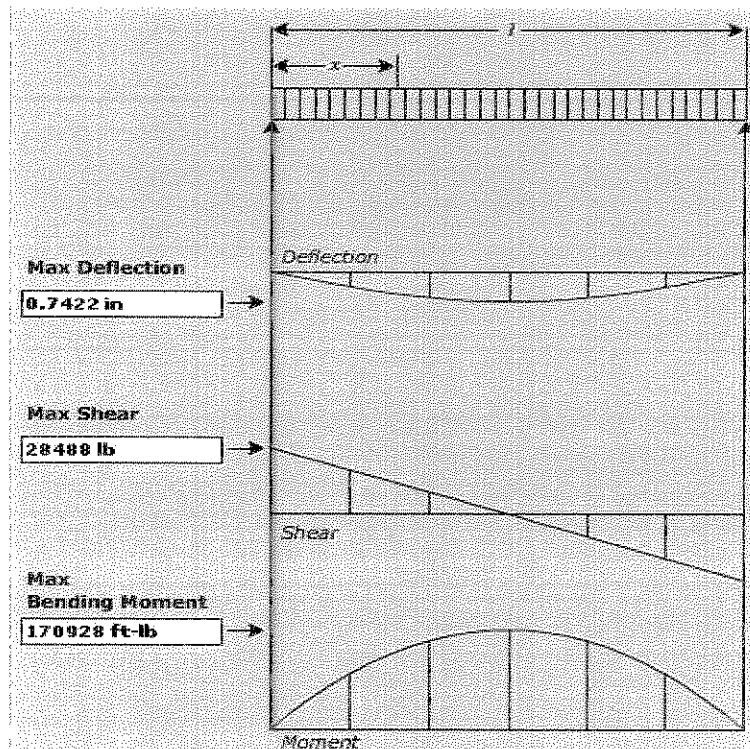


Figure 1. Plot of Deflection, Shear, and Bending Moment.

The calculations are based on four inputs; load, beam length, Young's Modulus, and area moment. Selection of English or Metric units is possible and accomplished using radio buttons.

Inputs

☒ English ☐ Metric (SI)

Load Rate lb/ft

Length ft

Young's Modulus psi

Area Moment in⁴

Figure 2. Inputs

The simulation outputs maximum values of deflection, shear, and bending moment, and has the ability to calculate these values as a function of x (or beam position) dynamically.

Beam x position

ft

Deflection(x)

in

Shear(x)

lb

Bending Moment (x)

ft-lb

Figure 3. Deflection, Shear, and Bending Moment as a function of x.

The dynamic calculations are run in three ways; by using the arrow buttons shown in Figure 3, by using the keyboard left and right arrow keys, or by dragging the vertical red bar in Figure 1.

The interval of calculation is set by the increment input box shown in Figure 4 below. This interval can be changed by inputting a new number. For example, an interval input size of 24 means that 24 points are calculated along the beam for plotting purposes when the simulation is run. Inputting a value of 50 changes it to 50 calculated points along the beam.

Increment

Figure 4. Set Interval

The simulation is run by pressing the Run Graph Calculation button, pressing the enter key, or sliding the vertical red bar in Figure 1.



Figure 5. Run Graph Calculation button

Resources

During the simulation development, we used Macromedia Flash 8 Professional, the Flash help dictionary found in Flash 8, and the class textbook Applied Statics and Strength of Materials found in reference 1.

Getting Started

Download the simple_beam.zip file and unzip it in a folder on your computer. In the simple_beam folder you will find two Flash fla files (simpleBeam_starter fla and simpleBeam_finished fla). The simpleBeam_starter fla contains all the graphical elements that you will need to complete this tutorial; and the simpleBeam_finished fla is the completed working simulation.

Open the simpleBeam_finished fla in Flash 8 and test it. Familiarize yourself with the program and how it works. Close this fla and open the simpleBeam_starter fla and then open the library which contains all the graphical elements that you will need to complete the simulation.

Creating the Graphical Interface

Start by opening up the simpleBeam_starter fla file in Flash 8. The file contains only one layer labeled Background Elements which contains various static text fields, a border, an input text field with the instance name myincrtxt, and a dynamic text field with the instance name my_status. The dynamic text field is used to send status messages to the screen to inform the user of invalid input or graphical corrections made due to low input values.

Add five layers to the timeline and label them top down actions, myInput_mc, runGraph_btn, beamPos_mc, and myPlots_mc as shown in Figure 6.

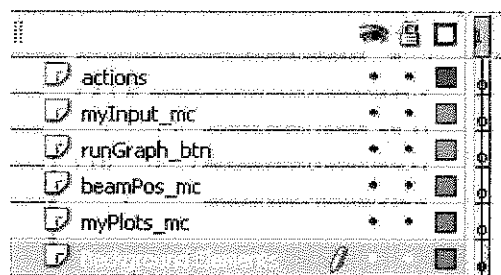


Figure 6. Simulation Layers

After creating the layers, complete the following steps to build the graphical interface:

1. Drag the myInput_mc movie clip from the library and position it on state at $x = 6$, $y = 69$ and give it the instance name myInput_mc.
2. Drag a button from the library and position it on state at $x = 18$, $y = 248$ and give it the instance name runGraph_btn. Open up properties panel and in the parameters tab type in the label box “Run Graph Calculation”.
3. Drag the beamPos_mc movie clip from the library and position it on state at $x = 64$, $y = 380$ and give it the instance name beamPos_mc.
4. Drag the myPlots_mc movie clip from the library and position it on state at $x = 405$, $y = 89$ and give it the instance name myPlots_mc.

Using Dot Syntax

The movie clips myInput_mc, beamPos_mc, and myPlots_mc rely heavily on the use of dot syntax since they are composites or movies within movies, buttons within movies, or text fields within movies. The dot syntax is very easy to use. Each time you want to reference a movie, button, or text field within a movie clip you simply put a dot and then the instance name you want to reference. For example, if you want to reference the right arrow key within the beamPos_mc movie clip of Figure 3 you would use a dot between the instance name of the movie clip and the instance name of the arrow button. An example of how this would be used is in the code snippet below, where pressing the arrow key with the instance name goRight_btn contained in the beamPos_mc movie clip produces a trace to the Flash output screen.

```
beamPos_mc.goRight_btn.onPress = function():Void{  
    trace(“pressed right arrow button”);  
}
```

We use dot syntax extensively throughout this tutorial to drill down into movie clips and reference other movies, buttons, and text fields. It is an important tool when dealing with complex movie interactions.

Simulation Architecture

The simulation has six sections and the code for each section is given and discussed in detail. Below is a short description of each section:

1. Section 1: Declare Initial Parameters - declares the parameters used throughout the calculations.
2. Section 2: Max Calculations – calculates the max deflection, shear, and bending moment.
3. Section 3: Graph Plotting – plots deflection, shear, and bending moment as a function of beam position in the three graphs found in Figure 1.

4. Section 4: Beam Position Calculations – codes the arrow buttons, keyboard left and right arrows, and graph slider to dynamically calculate deflection, shear, and bending moment as a function of beam position.
5. Section 5: Key Board Listener – creates a listener function which listens for input from the left arrow, right arrow, and enter keys.
6. Section 6: Radio Button Code – creates the listener for the radio buttons which determines if Metric or English values are accepted, converted, and outputted.

Section 1: Declare Initial Parameters

Start by opening up the actions panel of the simpleBeam_starter fla file. All code will be entered into the top layer of the Flash timeline labeled “actions” (which you created when setting up the graphical interface). You may choose to manually enter the code from this tutorial or cut and paste it from the completed tutorial.

The first statement of the code imports the necessary classes to run the English/Metric conversion radio buttons found in Figure 2.

```
//Import mx.controls for radio button  
import mx.controls.RadioButton;
```

The code below declares the constants necessary for Metric to English conversion.

```
//Conversion constants for English to Metric  
var convPtoN:Number = 4.448;  
var convFtoM:Number = 0.3048;  
var convPSItoPA:Number = 6894.7573;  
var convItoNmm:Number = 25.4;  
var convI3toNmm3:Number = 16390;  
var convI4toNmm4:Number = 416200;  
var myConv:Number = Math.pow(12,3);
```

In both Figures 2 and 3 we have overlapping Metric and English unit movies. Depending on the radio button selection, the English or Metric labels are made visible. We start by setting the metric labels to a visibility of “false” by setting the _visible command to “false”, leaving the English unit labels visible on the stage. Since both sets of units labels overlap, setting one invisible reveals the other clearly. Figure 7 below shows the overlapping unit labels. Since the labels are in the myInput_mc and beamPos_mc movie clips we must use dot syntax to drill down to them.

```
//Visibility of SI and English unit labels
myInput_mc.met_mc._visible = false;
beamPos_mc.met_2._visible = false;
```



Figure 7. Overlapping English and Metric unit labels.

In simple cases, like our overlapping labels of Figure 7, using the `_visible` command works fine. However, when dealing with more complex data transfer, it works better to reposition elements above the screen and bring them in and out as needed as opposed to using visibility.

Declare the five input variables

```
//Declare Input variables
var sInt:Number; //Interval Size
var beamL:Number; //Beam Length
var modulusE:Number; //Young's Modulus
var densityW:Number; //Load Rate
var ibeamI:Number; //Area Moment
```

We use short straight line segments to create the graphs for deflection, shear, and bending moment. The flash drawing API we use requires that we declare double values to plot the segments. These double values are obtained from the FOR loop of Section 3.

Declare the FOR loop variables

```
//Declare FOR loop variables
var myX_1:Number; //1st x position for plotting segment
var myX_2:Number; //2nd x position for plotting segment
var defByX_1:Number; //1st y deflection for plotting segment
var defByX_2:Number; //2nd y deflection for plotting segment
var shearV_1:Number; //1st y shear for plotting segment
var shearV_2:Number; //2nd y shear for plotting segment
var bendingM_1:Number; //1st y bending moment for plotting segment
var bendingM_2:Number; //2nd y bending moment for plotting segment
```

By setting the `xIncrement` variable equal to zero, we start the program at the zero value of `x` on the beam. Setting `convBool` equal to zero tells the radio button code that we are using English units.

```
//Initial x increment parameter and conversion Boolean
var xIncrement:Number = 0;
var convBool:Number = 0;
```

Section 2: Max Calculations

This portion of the simulations calculates the maximum beam deflection, shear and bending moment. Using the onRelease command we create a function that calls the error checking function.

```
//Run calculations button  
runGraph_btn.onRelease = function():Void{  
  errorChecking();  
}
```

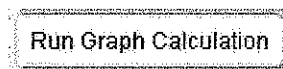


Figure 8. Run Graph Calculation Button.

Simple error checking has been set up using the isNaN and IF commands which check to see if the input is a number. If any of the four inputs are not numbers an error message is sent to a dynamic text field on the stage which has the instance name my_status.text.

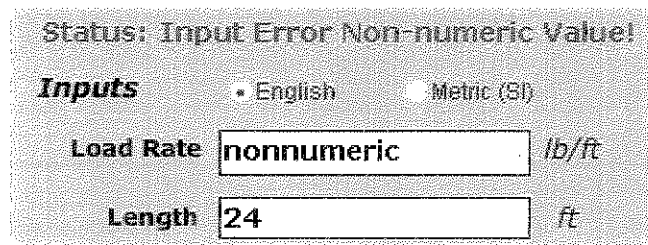


Figure 8. Error handling of a non-numeric input.

If all the input values are numeric then the plotting function createGraph is called.

```
//Simple error checking  
function errorChecking() {  
  if(isNaN(myInput_mc.w_txt.text)||isNaN(myInput_mc.l_txt.text)||isNaN(  
    myInput_mc.e_txt.text)||isNaN(myInput_mc.i_txt.text)){  
    my_status.text = "Status: Input Error Non-numeric Value!";  
  }else{  
    my_status.text = "";  
    createGraphs();  
  }  
}
```

In order, for the createGraph function to run it must grab the input parameters from the input text boxes. This is done using the init_param function. Since the text boxes are in the myInput_mc movie clip we must use dot syntax to drill down to them.

```
//Grab Input parameters
function init_param(){
densityW = myInput_mc.w_txt.text;
beamL = myInput_mc.l_txt.text;
modulasE = myInput_mc.e_txt.text;
ibeamI = myInput_mc.i_txt.text;
sInt = myincre_txt.text;}

//Plotting function
function createGraphs(){
//Initiate parameters
init_param(); //Grabs the text inputs from the text boxes
```

Each time a graph is created the previous graph needs to be cleared since the graphing API does not erase what it has already drawn. All three graphs (deflection, shear, and moment) exist in three different movie clips (mc.defl_mc, shear_mc, and momnt_mc) and must be cleared using the clear() command. Once again, we must apply dot syntax to reference these movies since they are in the myPlots_mc movie clip.

```
//Clear plots deflection, shear, and moment
myPlots_mc.defl_mc.clear(); //Clear the Deflection Graph
myPlots_mc.shear_mc.clear(); //Clear the Shear Graph
myPlots_mc.momnt_mc.clear(); //Clear the Bending Moment Graph
```

The equations for Max deflection, shear and bending moment of the beam were taken from the class text book and will not be discussed in detail in this tutorial. Those wanting to learn more about the physics can consult reference 1.

This portion of code takes the input parameters and calculates the maximum deflection.

```
//Calculate Max Deflection
var deflectionD = 5*densityW*Math.pow(beamL, 4)*myConv/(384*modulasE*ibeamI);
```

We use this small segment of code to truncate our deflection to four decimal places. The Math.ceil command rounds the number to its ceiling.

```
//Round decimal to 4 places
deflectionD = Math.ceil(deflectionD*10000)/10000;
```

The radio buttons in Figure 2 set the convBool variable and the IF command. If convBool is set equal to zero then the unit “in” is added to the answer for English units and if convBool is equal to 1 then the unit “mm” is added to the deflection answer for the Metric units. This is

accomplished using the + command which adds the string to the number. Adding units directly to the output text is a nice touch and avoids the use of visibility commands.

```
//End tag function depending on units
if(convBool == 0){
  //Add English units to the output text field
  myPlots_mc.defl_mc.maxDeflection.def_txt.text = deflectionD+" in";
}else{
  //Add Metric units to the output text field
  myPlots_mc.defl_mc.maxDeflection.def_txt.text = deflectionD+" mm";
}
```

The three tab pointers in Figure 1 point to the max value of the curves. We normalized our curves using the Math.log command which enabled us to illustrate change over a large range of values. Using the Math.log command saved us the difficulty of creating zoom functions.

As shown in Figure 9 below, the Max Deflection pointer is positioned using dot syntax to drill down to the pointer and the the _y command.

```
//Position max tab and pointer
myPlots_mc.defl_mc.maxDeflection._y =
Math.log(100*deflectionD)*Math.log(100*
deflectionD)*((densityW*myConv)/(24*modulasE*ibeamI))*(beamL/2)*(
Math.pow(beamL,3) - 2*beamL*Math.pow((beamL/2),2)+
Math.pow((beamL/2),3))/deflectionD;
myPlots_mc.defl_mc.maxDeflection._x =-130;
```

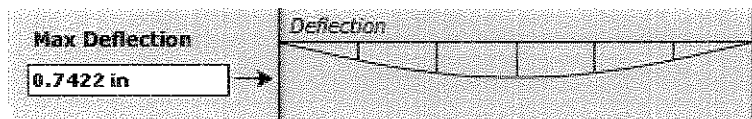


Figure 9. Deflection pointer pointing to max deflection of the graph.

Here we calculate Max Shear and limit it to two decimal places using the Math.ceil command.

```
//Calculate Max Shear
var maxShear = densityW*beamL/2;
maxShear = Math.ceil(maxShear*100)/100;
```

As was done for deflection, we set our max shear units using the convBool variable and the IF command.

```
//End tag function depending on units
if(convBool == 0){
```

```

myPlots_mc.shear_mc.maxShear_mc.shr_txt.text = maxShear + " lb";
}else{
myPlots_mc.shear_mc.maxShear_mc.shr_txt.text = maxShear + " N";
}

```

Here we move the Max Shear pointer to the correct y position using dot syntax and the `_y` command.

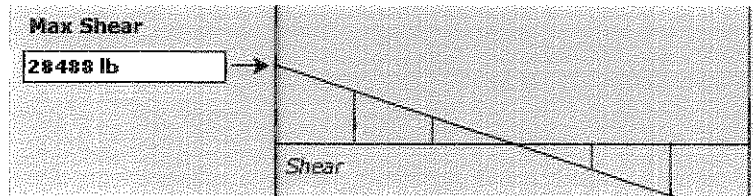


Figure 10. Shear pointer pointing to max shear value of the graph.

```

//Position max tab and pointer
myPlots_mc.shear_mc.maxShear_mc._y = Math.log(maxShear)
*4*(myPlots_mc.shear_mc._y-densityW*(beamL/2))/maxShear;

```

In this segment of code, we have the shear pointer move to the left (using dot syntax and the `_x` command) as the deflection pointer moves down as deflection increases. This keeps the two pointers from overlapping and interfering with one another.

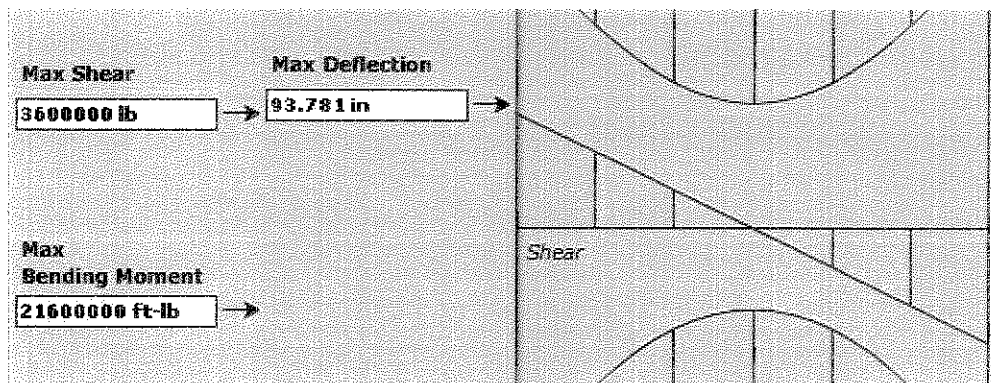


Figure 11. Shifting pointers as Deflection descends.

If deflection is greater than 7.5 shift the max shear pointer.

```

if(deflectionD>7.5){myPlots_mc.shear_mc.maxShear_mc._x = -260; }
else{myPlots_mc.shear_mc.maxShear_mc._x =-130;}

```

Calculate max bending moment and limit it to two decimal places using the `Math.ceil` command.

```
//Calculate Max Bending Moment
var maxBendingM = (densityW/8)*Math.pow(beamL,2);
maxBendingM = Math.ceil(maxBendingM*100)/100;
```

As was done for deflection, we set our max bending moment units using the convBool variable and the IF command.

```
//End tag function depending on units
if(convBool == 0){
  myPlots_mc.momnt_mc.maxMoment_mc.mnt_txt.text = maxBendingM + " ft-lb";
}else{myPlots_mc.momnt_mc.maxMoment_mc.mnt_txt.text = maxBendingM + " m-N";}
```

Here we move the Max Bending Moment pointer to the correct y position using dot syntax and the `_y` command.

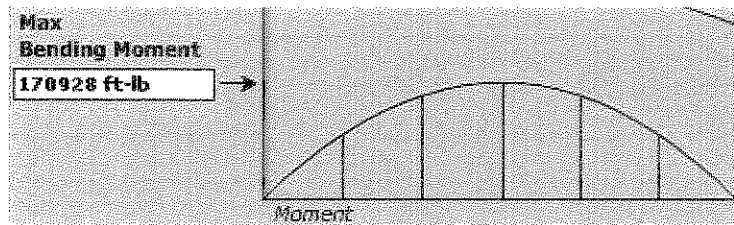


Figure 12. Bending Moment pointer pointing to max Bending Moment value of the graph.

```
//Position max tab and pointer
myPlots_mc.momnt_mc.maxMoment_mc._y = Math.log(maxShear)*6*
(myPlots_mc.momnt_mc._y-(densityW/2)*(beamL/2)*(beamL-(beamL/2)))
/maxBendingM;
```

Here we move the Max Bending moment to the left so it does not overlap the deflection as shown in Figure 11. If deflection is greater than 17 pixels we move it to the left using the `_x` position command.

```
if(deflectionD>17){myPlots_mc.momnt_mc.maxMoment_mc._x = -260; }
else{myPlots_mc.momnt_mc.maxMoment_mc._x =-130;}
```

The physical equations are nonlinear and as a result experience a large range of variation. . During certain input values the graph drops below the line of origin for shear and bending moment (see Figure 1). So the following code adjusts the plotting axis to stay with the center y position of the function. The condition only occurs for low input values and causes the graph to err in its representation by shifting the center line of the function. If there is a shift of more than 4 pixels, a status message is sent stating that the axis has been adjusted.

Status: Adjusted Axis for small input value no hatch plotting

Figure 13. Bending Moment pointer pointing to max Bending Moment value of the graph.

```
//Shear line Adjust for low numbers & Calculate Shear Endpoint Values
var myFirstNum:Number = Math.log(maxShear)*4*
(myPlots_mc.shear_mc._y- densityW*(beamL/2 - 0))/maxShear;
var myLastNum:Number = Math.log(maxShear)*4*
(myPlots_mc.shear_mc._y- densityW*(beamL/2 - beamL))/maxShear;
var myAdjustShear = (myFirstNum+myLastNum)/2; //Shear axis amount to shift
myPlots_mc.myGraphic.shearLine_mc._y = 253.0 + myAdjustShear;
```

Adjust Bending Moment Axis.

```
//Bending Moment line Adjust for low numbers
var myFirstNum2:Number = Math.log(maxShear)*6*(myPlots_mc.momnt_mc._y)/
maxBendingM;
var myLastNum2:Number = Math.log(maxShear)*6*(myPlots_mc.momnt_mc._y)
/maxBendingM;
var myAdjustMoment = (myFirstNum2+myLastNum2)/2; //Moment axis shift
myPlots_mc.myGraphic.momentLine_mc._y = 386 + myAdjustMoment;
```

Send status message that the bending moment axis has been shifted.

```
//Send Status message if shift of more that 4 pixels occurs
if(myAdjustShear>4){my_status.text = "Status: Adjusted Axis for small input value
no hatch plotting";} //Change the graph if it shifts more than 4 pixels.
else{my_status.text = ""};
```

Two important aspects driving the complexity of the code above arise from the design task of creating a simulation that gives real physical numbers and a visual experience to represent those physical numbers. Thus, in the creation of the simulation both aspects drove the need for error handling of the highly fluctuating number values of a nonlinear equation.

Section 3: Graph Plotting

In this section, we discuss the code that creates the graphs for beam deflection, shear, and bending moment. For simplicity, we use short straight line segments to plot the graphs. The drawing API used requires that we declare double values to plot the segments. These double values are easily obtained from the FOR loop by calculating the *ith* and *ith+1* values and plotting a segment for each iteration.

Notice that the FOR loop iterates through one less than the total number of iterations (*sInt-1*). Since we are calculating the *ith* and *ith+1* values for each iteration, the final value is calculated on the *ith-1* iteration.

```
for (i = 0; i <= sInt-1; i++) {
```

At each value of i of the FOR loop, we calculate two values of position on the beam by calculating the i th and $i+1$ values.

```
//X points
myX_1 = i*beamL/sInt;
myX_2 = (i+1)*beamL/sInt;
```

The Figure 14 below shows six straight line segments generated by an interval (sInt) value of six. In this case, 12 beam position values (6 myX_1 and 6 myX_2) are generated to create this curve.

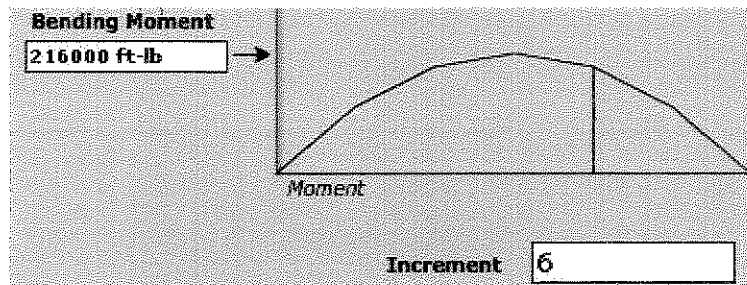


Figure 14. Bending Moment pointer pointing to max Bending Moment value of the graph.

Calculate the 1st deflection value for the 1st point of the line segment.

```
//Calculate Deflection Values
defByX_1 = Math.log(100*deflectionD)*Math.log(100*deflectionD)*((densityW*myConv)/
(24*modulasE*ibeamI))*myX_1*(Math.pow(beamL,3) -
2*beamL*Math.pow(myX_1,2)+Math.pow(myX_1,3))/deflectionD;
```

Calculate the 2nd deflection value for the 2nd point of the line segment.

```
defByX_2 = Math.log(100*deflectionD)*Math.log(100*deflectionD)*((densityW*myConv)/
(24*modulasE*ibeamI))*myX_2*(Math.pow(beamL,3) -
2*beamL*Math.pow(myX_2,2)+Math.pow(myX_2,3))/deflectionD;
```

Plot the i th deflection line segment using the Flash drawing API. The drawing API allows you to create vector graphics. Using ActionScript we can draw lines, curves, shapes, fills, and gradients and display them on the screen. You can draw with ActionScript on a MovieClip instance. In our case, we define three movie clip instances for our three graphs (mc.defl_mc, shear_mc, and momnt_mc). These movie clips were placed inside another movie clip named myPlots_mc and dot syntax is used to reference the graphs. It takes two points to draw each line segment. The moveTo command sets the beginning point of the drawing API (sets the pen down) and the lineTo command draws a straight line from the first point to the second point.

```
//Plot deflection values
myPlots_mc.defl_mc.lineStyle(1, 0x000000, 100); //Define line size, line color, line
```

```

transparency
myPlots_mc.defl_mc.moveTo(myX_1*248/beamL, defByX_1); //First point of the line
segment
myPlots_mc.defl_mc.lineTo(myX_2*248/beamL, defByX_2); //Second point of the line
segment

```

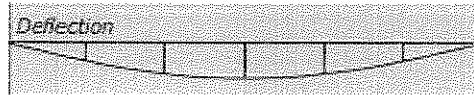


Figure 15. Deflection Graph.

In reference 1, the textbook graphics have vertical hatch lines. We are able to reproduce this effect using the modulus command (%). Using the modulus and IF command, we draw the vertical hatch lines on the graph every 4th value of i. This occurs since the modulus is the remainder of “i” divided by 4. Every multiple of 4 has a value of zero and a vertical hatch line is drawn.

```

//Vertical Grid Lines
if(i%4==0){
myPlots_mc.defl_mc.lineStyle(1, 0x000000, 100);
myPlots_mc.defl_mc.moveTo(myX_1*248/beamL, 0);
myPlots_mc.defl_mc.lineTo(myX_1*248/beamL, defByX_1);
}

```

Calculate the 1st deflection value for the 1st point of the line segment.

```

//Calculate Shear Values
shearV_1 = Math.log(maxShear)*4*(myPlots_mc.shear_mc._y-densityW*
(beamL/2 - myX_1))/maxShear;

```

Calculate the 2nd shear value for the 2nd point of the line segment.

```

shearV_2 = Math.log(maxShear)*4*(myPlots_mc.shear_mc._y-densityW*
(beamL/2 - myX_2))/maxShear;

```

Plot the ith shear line segment using the Flash drawing API.

```

//Plot shear values
myPlots_mc.shear_mc.lineStyle(1, 0x000000, 100); //Define line size, line color, line
transparency
myPlots_mc.shear_mc.moveTo(myX_1*248/beamL, shearV_1); //First point of line segment
myPlots_mc.shear_mc.lineTo(myX_2*248/beamL, shearV_2);

```

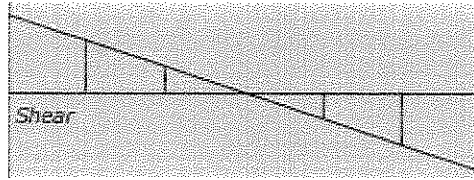


Figure 16. Shear Graph.

Draw the vertical hatch lines on the graph every 4th value of i using the modulus function and commands.

```
//Vertical Grid Lines
if(i%4==0&&myAdjustShear<4){
myPlots_mc.shear_mc.lineStyle(1, 0x000000, 100);
myPlots_mc.shear_mc.moveTo(myX_1*248/beamL, 0);
myPlots_mc.shear_mc.lineTo(myX_1*248/beamL, shearV_1);
}
```

Calculate the 1st bending moment value for the 1st point of the line segment.

```
//Calculate Bending Mommment values
bendingM_1 = Math.log(maxShear)*6*(myPlots_mc.momnt_mc._y-
(densityW/2)*myX_1*(beamL-myX_1))/maxBendingM;
```

Calculate the 2nd bending moment value for the 2nd point of the line segment.

```
bendingM_2 = Math.log(maxShear)*6*(myPlots_mc.momnt_mc._y-
(densityW/2)*myX_2*(beamL-myX_2))/maxBendingM;
```

Plot the i th bending moment line segment using the flash drawing API.

```
//Plot bending moment values
myPlots_mc.momnt_mc.lineStyle(1, 0x000000, 100); //Define line size, line color, line
transparency
myPlots_mc.momnt_mc.moveTo(myX_1*248/beamL, bendingM_1); //first point of line
segment
myPlots_mc.momnt_mc.lineTo(myX_2*248/beamL, bendingM_2);
```

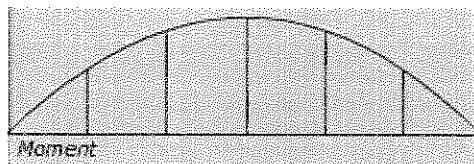


Figure 17. Bending Moment Graph.

Draw the vertical hatch lines on the graph every 4th value of i using the modulus and IF commands.

```
//Vertical Grid Lines
if(i%4==0&&myAdjustMoment<4){
myPlots_mc.momnt_mc.lineStyle(1, 0x000000, 100);
myPlots_mc.momnt_mc.moveTo(myX_1*248/beamL, 0);
myPlots_mc.momnt_mc.lineTo(myX_1*248/beamL, bendingM_1);
}}
//End of for loop
}
```

Section 4: Beam Position Calculations

In addition to calculating max values and plotting graphs of deflection, shear, and bending moment, we developed a graphical interface which allows for dynamic calculations of these values anywhere on the beam by using arrow buttons, arrows on the keyboard, or a slider bar.

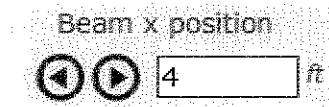


Figure 18. Beam Position Arrow Buttons

Upon pressing the right arrow key the errorChecking function is run which calculates the present max values and creates our three graphs in Figure 1.

```
//Increment Right
beamPos_mc.goRight_btn.onPress = function():Void{
errorChecking();
```

The onEnterFrame command is started and executes its function at the current frame rate which increments the xIncrement beam position value along the beam and calculates dynamically the values of deflection, shear, and bending moment by executing the findValues function. The IF statement stops the iteration of xIncrement if xIncrement is greater than or equal to the beam length beamL. It does this since the ith+1 value is calculated in the FOR loop and there is no need to calculate further. If the final ith+1 value is greater than the beam length, then a second IF statement changes the final value to the beam length.

```
myPlots_mc.myGraphic.onEnterFrame = function():Void{
if(xIncrement<beamL){
xIncrement=xIncrement+beamL/sInt;
if(xIncrement>beamL){xIncrement=beamL;}
beamPos_mc.x_pos.text = xIncrement;
```



```

    findValues(xIncrement);
  }}
}

```

Upon release of the right arrow key the onEnterFrame command is deleted and the dynamic calculations cease.

```

beamPos_mc.goRight_btn.onRelease = function():Void{
delete myPlots_mc.myGraphic.onEnterFrame;
}

```

Upon mouse release outside of the right arrow key the onEnterFrame is deleted and the dynamic calculations cease. It is important to handle this condition as well since accidentally sliding away from the button and releasing will cause the calculation to continue if not corrected using action scripting.

```

beamPos_mc.goRight_btn.onReleaseOutside = function():Void{
delete myPlots_mc.myGraphic.onEnterFrame;
}

```

Upon pressing the left arrow key the errorChecking function is run which calculates the present Max values and creates our three graphs.

```

//Decrement Left
beamPos_mc.goLeft_btn.onPress = function():Void{
errorChecking();
}

```

The onEnterFrame command is started and executes its function at the current frame rate which decrements the value along the beam and calculates dynamically the values of deflection, shear, and bending moment by executing the findValues function. The IF statement stops the iteration of xIncrement if xIncrement is less than or equal to zero. If the value is less than the value of zero, then a second IF statement changes the final value to zero.

```

myPlots_mc.myGraphic.onEnterFrame = function():Void{
if(xIncrement>0){
xIncrement=xIncrement-beamL/sInt;
if(xIncrement<0){xIncrement=0;}
beamPos_mc.x_pos.text = xIncrement;
findValues(xIncrement);
}}
}

```

Upon release of the right arrow key the onEnterFrame is deleted and the dynamic calculations cease.

```

beamPos_mc.goLeft_btn.onRelease = function():Void{
delete myPlots_mc.myGraphic.onEnterFrame;
}

```

Upon mouse release outside of the left arrow key the onEnterFrame is deleted and the dynamic calculations cease. It is important to handle this condition as well since accidentally sliding away from the button and releasing it will cause the calculation to continue.

```

beamPos_mc.goLeft_btn.onReleaseOutside = function():Void{
delete myPlots_mc.myGraphic.onEnterFrame;
}

```

The red line of Figure 19 below can be dragged along the beam to calculate deflection, shear, and bending moment as a function of beam position.

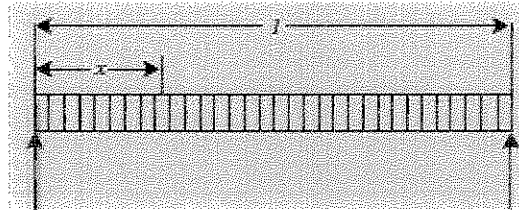


Figure 19. Drag able red line

```

myPlots_mc.myGraphic.redLine_mx.onPress = function():Void {
errorChecking();
}

```

Upon pressing the red line the erroChecking function is executed which calculates the max values and plots the graphs. Then the startDrag function is executed. Its parameters (0,0,0,247) restricts the drag of the red line vertically, and horizontally along the beam which is 247 pixels long. In addition, the onMouseMove command is executed which dynamically calculates the values of deflection, shear, and bending moment as the mouse moves by executing the findValues function.

```

this.startDrag(0,0,0,247);
this.onMouseMove = function():Void{
findValues(this._x*(beamL/247));
trace(this._x);
beamPos_mc.x_pos.text = xIncrement= this._x*(beamL/247);
xIncrement = (beamL/sInt)*Math.floor(xIncrement/(beamL/sInt));
}
}

```

Upon release of the mouse the onMouseMove is set to Null which stops the calculations and the stopDrag function is executed which releases the red line from the mouse. Once again, we are using dot syntax to drill down to the red line.

```

myPlots_mc.myGraphic.redLine_mx.onRelease = function():Void {
this.onMouseMove = null;
this.stopDrag();
}

```

It is important to execute an on release outside function, otherwise the red line will stick to the mouse if it is slid off the graph without release. Thus upon return to the graph the red line will stick to the mouse and slide without clicking.

```

myPlots_mc.myGraphic.redLine_mx.onReleaseOutside = function():Void {
this.onMouseMove = null;
this.stopDrag();
}

```

The purpose of the findValues function is to receive the new value of xIncrement and find the new values of deflection, shear, and bending moment and send them to the appropriate dynamic text boxes for display shown in Figure 20.

Figure 20. Deflection, Shear, and Bending Moment output

```

//Main function for calculating beam deflection, shear, and bending moment as a function of x
function findValues(inputX):Void{
var dx_txtFull:Number =
((densityW*myConv)/(24*modulasE*ibeamI))*inputX*(Math.pow(beamL,3) -
2*beamL*Math.pow(inputX,2)+Math.pow(inputX,3));

//Truncation to four decimal places
beamPos_mc.dx_txt.text = Math.ceil(dx_txtFull*10000)/10000;
beamPos_mc.sx_txt.text = densityW*(beamL/2 - inputX);
beamPos_mc.bx_txt.text = (densityW/2)*inputX*(beamL-inputX);
myPlots_mc.myGraphic.redLine_mx._x = inputX*(247/beamL);
}

```

Section 5: Keyboard Listener

It is a nice touch to be able to enter commands from the keyboard. It saves time and is intuitive to the user. However, from experience, we have learned that the keyboard listener function in

Flash can sometime interfere with other code. So once implemented, it needs to be thoroughly tested and the appropriate error handling code inserted.

Create the key listener object.

```
//Create a listener function which listens for input from the left & right arrow, and enter keys.  
var keyListener:Object = new Object();
```

Using the onKeyDown function determines which key is being pressed.

```
keyListener.onKeyDown = function():Void {
```

Use the following code snippet to determining various key code values without having to use a keyboard conversion table. The values are sent directly to the output window using the trace command.

```
//trace code for finding Key Code, Ascii, and Character Ascii  
trace("DOWN -> Code: " + Key.getCode() + "\tASCII: " + Key.getAscii() + "\tKey: " +  
chr(Key.getAscii()));
```

The key code value of 13 is the Enter key. Upon pressing the Enter key the function erroChecking is executed which determines the max values of deflection, shear, and bending moment and then plots the graphs.

```
//Enter Key to Run Graph Calculation  
if(Key.getCode()==13){  
xIncrement = 0;  
myPlots_mc.myGraphic.redLine_mx._x = 0;  
erroChecking();  
}
```

The key code value of 37 is the Left arrow key. Upon pressing the Left arrow key on the keyboard the function erroChecking is executed which determines the max values of deflection, shear, and bending moment and then plots the graphs.

```
//Left Arrow Key  
if(Key.getCode()==37){
```

We found that the key code was interacting with our radio buttons when pressing the left or right arrow keys. So we disabled the radio buttons during key press and enabled them upon release using the enabled command equal to false or true. This cleared up the problem. But it illustrates that using the keyboard in Flash for complex programs can be cumbersome.

```
myInput_mc.radioGroup.enabled = false;  
erroChecking();
```

Next, the onEnterFrame command is started and executes its function at the current frame rate which decrements the value along the beam and calculates dynamically the values of deflection, shear, and bending moment by executing the findValues function.

```
myPlots_mc.myGraphic.onEnterFrame = function():Void{
  if(xIncrement>0){
    xIncrement=xIncrement-beamL/sInt;
    if(xIncrement<0){xIncrement=0;}
    beamPos_mc.x_pos.text = xIncrement;
    findValues(xIncrement);
  }}
}
```

The key code value of 39 is the right arrow key. Upon pressing the right arrow key on the keyboard the function errorChecking is executed which determines the max values of deflection, shear, and bending moment and then plots the graphs.

```
//Right Arrow Key
if(Key.getCode()==39){
```

By disabling the radio buttons during key press we cleared up the interaction of the keyboard arrow keys with the radio buttons. On key release, we enable the radio buttons.

```
myInput_mc.radioGroup.enabled = false;
errorChecking();
```

As before, the onEnterFrame function is called and executes its function at the current frame rate which decrements the value along the beam and calculates dynamically the values of deflection, shear, and bending moment by executing the findValues function.

```
myPlots_mc.myGraphic.onEnterFrame = function():Void{
  if(xIncrement<beamL){
    xIncrement=xIncrement+beamL/sInt;
    if(xIncrement>beamL){xIncrement=beamL;}
    beamPos_mc.x_pos.text = xIncrement;
    findValues(xIncrement);
  }}}
}
```

It is important to stop the onEnterFrame function upon releasing the error keys. This is done using the onKeyUp function and deleting the onEnterFrame.

```
keyListener.onKeyUp = function():Void {
```

This enables the radio buttons on the onKeyUp command.

```

myInput_mc.radioGroup.enabled = true;
delete myPlots_mc.myGraphic.onEnterFrame;
}

```

Finally, the keyListener object (created at the very beginning) is added. One of the most common errors in coding listener objects is to fail to create a listener object or to forget to copy and paste it from a code bank.

```

//Key listener
Key.addListener(keyListener);

```

Section 6: Radio Button Code

In this final portion of the code, we want to select English or Metric units using radio buttons. The radio button code was taken directly from the Flash help library within the Flash dictionary library inside of Flash. To this code we added the English and Metric conversion coding.

First, we create the listener object.

```

// Create listener object.
var rbListener:Object = new Object();

```

This function is executed when the user clicks a radio button.

```

rbListener.click = function(evt_obj:Object){

```

In many coding situations, we want to know the instance name of what we are pressing. This is accomplished by using the `_name` command. So in our case if the radio button pressed has the instance name `met_radio` then we run the Metric conversion code.

```

if(evt_obj.target.selection._name == "met_radio"){

```

We set the `convBool` equal to 1 so that the correct Metric units are inserted into the max pointer dynamic text boxes discussed earlier.

```

convBool = 1;

```

In this segment, we set the visibility of the labels by making the Metric labels visible and English labels invisible using the `_visible` command.

```

myInput_mc.met_mc._visible = true;
beamPos_mc.met_2._visible = true;
myInput_mc.eng_mc._visible = false;

```

```
beamPos_mc.eng_2._visible = false;
```

This code converts the input values to Metric and places the Metric values back into the input text boxes.

```
myInput_mc.w_txt.text=myInput_mc.w_txt.text*convPtoN/convFtoM;  
myInput_mc.l_txt.text=myInput_mc.l_txt.text*convFtoM;  
myInput_mc.e_txt.text=myInput_mc.e_txt.text*convPSItoPA;  
myInput_mc.i_txt.text=myInput_mc.i_txt.text*convI4tomm4;  
sInt = myincre_txt.text;
```

Set the deflection conversion factor.

```
myConv = myConv*convI3tomm3*10000000;
```

If the radio instance name is not equal to met_radio then we run the English conversion code.

```
}else{
```

We set the convBool equal to 0 so that the correct English units are inserted into the max pointer dynamic text boxes discussed earlier.

```
convBool = 0;
```

In this segment we set the visibility of the labels making English labels visible and Metric labels invisible using the _visible command.

```
myInput_mc.met_mc._visible = false;  
beamPos_mc.met_2._visible = false;  
myInput_mc.eng_mc._visible = true;  
beamPos_mc.eng_2._visible = true;
```

This code converts the input values to English and places the English values back into the input text boxes.

```
myInput_mc.w_txt.text=myInput_mc.w_txt.text*convFtoM/convPtoN;  
myInput_mc.l_txt.text=myInput_mc.l_txt.text/convFtoM;  
myInput_mc.e_txt.text=myInput_mc.e_txt.text/convPSItoPA;  
myInput_mc.i_txt.text=myInput_mc.i_txt.text/convI4tomm4;
```

Set the equations conversion factor.

```
myConv = myConv/(10000000*convI3tomm3);  
myConv = Math.pow(12,3);  
}
```

```
}
```

Finally, create the click listener and connect it to our rbListener object.

```
// Radio Button listener.  
myInput_mc.radioGroup.addEventListener("click", rbListener);
```

Summary

The greatest advantage of using Macromedia Flash in building scientific simulations is Flash's ability to combine high level programming with a rich user environment. In this tutorial, we demonstrated a number of programming techniques not commonly found in any one place in the literature. There are few (if any) good Flash books for programming engineering interfaces for the classroom. It is our hope that paper helps fill that gap.

With Adobe's purchase of Macromedia we have seen the architecture of Flash extend considerably. Given Flash's cross platform and rapid deployment ability, it is the tool of choice for academic simulation.

Additional Flash Resources

For the reader who wants to pursue a vigorous Flash training program, we have found the following resources useful in our study of Flash for beginner to advanced programmer.

1. lynda.com - a video library of very pertinent multimedia topics – stays current on many topics.
2. Community MX - professionals in the Flash community publish articles, and how-to tutorials.
3. Flash Kit - an extensive resource of Flash tutorials and example movies.
4. Safari Online - entire library of online books on various computer related topics - very current.
5. Adobe.com - extensive tech docs and some example code (& Flash internal code help).
6. kirupa.com - extensive set of online tutorials which deal with the use of PHP, MYSQL, and Flash.
7. trainsimpleonline.com – a video library geared towards the Flash professional treating up-to-date and relevant topics. Top notch site!

Acknowledgement

Many thanks to our Dean of Arts and Sciences Kevin Corcoran whose generous support made this paper possible.

References

1. Leonard Spiegel and George F. Limbrunner, Applied Statics and Strength of Materials 4th Edition, Pearson Education, Inc., 2004