SketchApp – The Möbius interactive spline drawing tool

# Introduction

[Möbius](https://www.digitaled.com/mobius/) is the online STEM learning platform from [DigitalEd](https://www.digitaled.com/). One of its features is the [HTML question type](https://www.digitaled.com/support/help/instructor/Content/INST-AUTHORING/QUESTION-TYPES/Author-HTML-question.htm) that can be used for as a *response area*. When inserting a HTML response area into a question, an HTML <iframe> element is created. An iframe is simply used to embed a HTML page in the current one. Using an iframe ensures that the parent page is not affected.

The SketchApp is built within the inserted iframe and requires three JavaScript scripts:

* RunApp.js This is the actual code that generates the SketchApp
* Paper-full.js Paper.js is used to create an interactive environment
* Cubic\_spline.js This script allows us to draw Cubic Monotone (Hermite) splines

The SketchApp is designed to draw interactive splines on top of a graph with some buttons. An example is given in Figure 1.

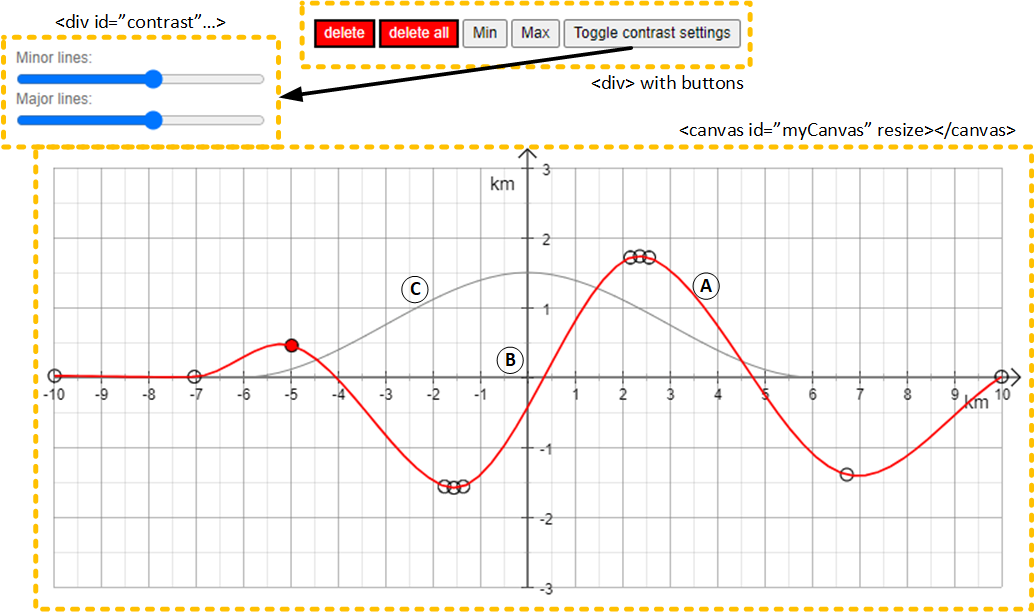


Figure : SketchApp example. (A) Spline (B) Background consisting of axes with labels and a grid (C) Spline as background

The HTML response area requires three functions to work:

* Initialize()
* getFeedback()
* getResponse()

# Basics

## Setting up the iframe

The first step is to create a <div> and <canvas> element in the *Question HTML* part of the response area (Figure 1). In the <div> element, we define the necessary HTML buttons from <input> elements and give the <canvas> element[[1]](#footnote-1) an id, *myCanvas*. In RunApp.js, we use Paper.js to setup a PaperScope object with the canvas and create a Tool object (Figure 2). The Tool object allows us to interact by using mouse and keyboard inputs: *onMouseDown, onMouseUp, onMouseDrag, onMouseMove, onKeyDown* and *onKeyUp*.

<div style="text-align:center">

<input type="button" style="color: white; background-color: red;" value="delete" id="delPoint" />

<input type="button" style="color: white; background-color: red;" value="delete all" id="delAll" />

<input type="button" style="color: black;" value="Min" id="buttonMin" />

<input type="button" style="color: black;" value="Max" id="buttonMax" />

<input type="button" style="color: black;" value="Toggle contrast settings" id="toggleContrast" />

<div id="contrast" style="display: none; text-align:left">

Major lines: <input type="range" style="width: 200px; " id="gridMajor" min="0" max="9" value=""+major\_grid\_lines.lineColor\*10>

Minor lines: <input type="range" style="width: 200px; " id="gridMinor" min="0" max="9" value=""+minor\_grid\_lines.lineColor\*10>

</div>

</div>

<canvas id="myCanvas" resize></canvas>

Figure : Möbius HTML response area - Question HTML part

*/\* Create a PaperScope and Tool object, give the canvas dimensions* *and setup the scope with the canvas \*/*

var scope = new paper.PaperScope();

var tool = new scope.Tool();

$("#myCanvas").width( canvasDef.width );

$("#myCanvas").height( canvasDef.height );

scope.setup($("#myCanvas")[0]);

Figure : RunApp.js code snippet of setting up PaperScope with the canvas and the Tool object

## Background

When the SketchApp is build, an empty graph is created first, from now referred to as background. The background consists of:

* x-axis & y-axis lines
* x-axis label & y-axis labels
* x-axis major & minor tickmarks
* x-axis major & minor tickmark labels
* major & minor gridlines
* a spline (optional)

A Paper.js Group() object called *PermanentElements* is defined. Lines and text are defined using the Paper.js Path() and PointText() objects, respectively. All the objects are added as child objects to *PermanentElements*. When done, *PermanentElements* is added to the *activeLayer* of the scope, drawing the objects to the canvas.

Optionally, a spline can be added to *PermanentElements* using a Path() object. How this is done is explained in section “Drawing the spline”.

## Interaction

Interaction is possible in two ways: (1) *onMouseDown* and (2) *onMouseDrag*. When the user clicks (*onMouseDown*) on the canvas, a Paper.js Point() object is created. The object is then pushed to a *PointsLocation* array, which keeps track of all created points by the user. If a point is created between two already existing points, the point is inserted in the array instead.

If the mouse is dragged when clicked, the position is tracked continuously until released. However, the position of the point in *PointsLocation* is determined by the click event and does not change when dragging. This makes it possible to have a point of which the x-coordinate is larger/smaller than (one of) its neighbour(s). The code for mouse interaction can be found in Appendix A.

## Drawing the spline

So far only the background has been drawn (*PermanentElements*) and the user points tracked (*PointsLocation*). Actual drawing of the spline is done separately and is entirely dependent on the *PointsLocation* array. First two groups are created, *drawnPoints* and *splinePoints*. For each element in *PointsLocation*, Shape.Circle() object is added to *drawnPoints*. Then, a Path(), *lineSpline*, and MonotonicCubicSpline(), *mySpline*, object are created. Interpolation between two elements of *PointsLocation* is done with *mySpline* to build *lineSpline*. Finally, *drawnPoints* and *lineSpline* are added to the *activeLayer*, drawing the points and spline to the canvas.

## Button interaction

The <input> elements with *type=”button”* create simple push buttons to interact with (*click* event handler). The SketchApp uses five buttons:

* Delete Delete the selected point
* Delete all Delete all points
* Min Force the selected point to be a local minimum
* Max Force the selected point to be a local maximum
* Toggle contrast settings Toggle sliders to adjust major and minor gridline opacity

Buttons “Delete” and “Delete all” modify the *PointsLocation* array by removing a single Point() object (“Delete”) or by removing all Point() objects (“Delete all”). The “Min”/”Max” buttons add Point() objects to the *PointsLocation* array, one before and one after the selected element in the array. The added points have a x- and y-offset such that the selected point becomes a local minimum or maximum. The “Toggle contrast settings” shows or hides a nested <div> element with two sliders. The sliders set the *lineColor* attribute of the associated Path() object. The code of all the buttons can be found in Appendix B.

# Designing the background graph

Designing a the background graph is done by adjusting variables in the Algorithm section of a Möbius question. The variables are given and described in Table 1.

Table : Möbius question Algorithm variables

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Appendix B – Mouse interaction

tool.onMouseDown = function(*click*) {

var hitPoint = new scope.Point(*click*.event.offsetX, *click*.event.offsetY);

    var hitResult = drawnPoints.hitTest(hitPoint, hitOptions);

    if (!hitResult) {

    if (selected\_x !== null || selected\_y !== null ) {

        selected\_x = null;

            selected\_y = null;

        }

    else {

*/\* First point \*/*

        if (PointsLocation.length == 0) {

        PointsLocation.push(hitPoint);

            pathsPointsfitsY.push(*click*.event.offsetY);

            pathsPointsfitsX.push(*click*.event.offsetX);

        }

*/\* There are points \*/*

        else {

        for (var i = 0 ; i < PointsLocation.length ; i++) {

*/\* Insert point in array \*/*

                if (*click*.event.offsetX < PointsLocation[i].x) {

                PointsLocation.splice(i, 0, hitPoint);

                    pathsPointsfitsX.splice(i, 0, *click*.event.offsetX);

                    pathsPointsfitsY.splice(i, 0, *click*.event.offsetY);

                    break;

                }

*/\* New point at the end, append point \*/*

            else if (i == PointsLocation.length - 1) {

            PointsLocation.splice(PointsLocation.length, 0, hitPoint);

                pathsPointsfitsX.splice(PointsLocation.length, 0, *click*.event.offsetX);

                pathsPointsfitsY.splice(PointsLocation.length, 0, *click*.event.offsetY);

                break;

            }

        }

    }

}

}

else {

selected\_x = *click*.event.offsetX;

    selected\_y = *click*.event.offsetY ;

    }

    draw\_spline();

};

Appendix A – Button functionality

**Delete button**

var delPoint = $('#delPoint');

*/\* delete point \*/*

delPoint.click(function() {

*/\* loops through all the drawn points \*/*

    for (points = 0 ; points < PointsLocation.length ; points++) {

*/\* if difference between (x,y) coordinate of point and (x,y)*

*coordinate of selected (clicked on screen) is less than 10,*

*point is found and remove that point from array \*/*

        if ((Math.abs(PointsLocation[points].x - selected\_x) < 10) && (Math.abs(PointsLocation[points].y - selected\_y) < 10)) {

        PointsLocation.splice(points, 1);

            pathsPointsfitsX.splice(points, 1);

            pathsPointsfitsY.splice(points, 1);

        }

    }

*/\* deselect point and draw new spline \*/*

    selected\_x = null;

    selected\_y = null ;

    draw\_spline();

});

**Delete all button**

var delAll = $('#delAll');

*/\* delete all points \*/*

delAll.click(function() {

*/\* splice(0, .length) removes all items from array \*/*

    PointsLocation.splice(0, PointsLocation.length);

    pathsPointsfitsX.splice(0, pathsPointsfitsX.length);

    pathsPointsfitsY.splice(0, pathsPointsfitsY.length);

    drawnPoints.removeChildren();

    splinePoints.removeChildren();

*/\* draw spline \*/*

    draw\_spline();

});

**Min button**

var buttonMin = $('#buttonMin');

*/\* min function \*/*

buttonMin.click(function() {

*/\* loops through all the drawn points \*/*

    for (points = 0 ; points < PointsLocation.length ; points++ ){

*/\* if difference between (x,y) coordinate of point and (x,y)*

*coordinate where clicked on screen is less than 10, point is found*

*and remove that point from array \*/*

if ((Math.abs(PointsLocation[points].x - selected\_x) < 10) && (Math.abs(PointsLocation[points].y - selected\_y) < 10)) {

*/\* 'temporarily' save y-coordinate of selected point \*/*

        var tempy = PointsLocation[points].y;

*/\* insert point before selected point, move point w.r.t. selected*

*point by deltax and deltay \*/*

        PointsLocation.splice(points,0, new scope.Point(selected\_x - deltax, tempy - deltay ));

        pathsPointsfitsX.splice(points, 0, selected\_x - deltax );

        pathsPointsfitsY.splice(points, 0, tempy - deltay);

*/\* insert point after selected point, move point w.r.t. selected point*

*by deltax and deltay \*/*

        PointsLocation.splice(points+2, 0, new scope.Point(selected\_x + deltax, tempy - deltay ) );

        pathsPointsfitsX.splice(points+2, 0, selected\_x + deltax);

        pathsPointsfitsY.splice(points+2, 0, tempy - deltay);

        break;

        }

    }

*/\* deselect point and draw new spline \*/*

    selected\_x = null;

    selected\_y = null ;

    draw\_spline();

});

**Max button**

var buttonMax = $('#buttonMax');

*/\* max function \*/*

buttonMax.click(function() {

*/\* loops through all the drawn points \*/*

for (points = 0 ; points < PointsLocation.length ; points++ ){

*/\* if difference between (x,y) coordinate of point and (x,y)*

*coordinate where clicked on screen is less than 10, point is found*

*and remove that point from array \*/*

        if ((Math.abs(PointsLocation[points].x - selected\_x) < 10) && (Math.abs(PointsLocation[points].y - selected\_y) < 10)) {

*/\* 'temporarily' save y-coordinate of selected point \*/*

            var tempy = PointsLocation[points].y;

*/\* insert point before selected point, move point w.r.t. selected*

*point by deltax and deltay \*/*

            PointsLocation.splice(points,0, new scope.Point(selected\_x - deltax, tempy + deltay ));

            pathsPointsfitsX.splice(points, 0, selected\_x - deltax);

            pathsPointsfitsY.splice(points, 0, tempy + deltay);

*/\* insert point after selected point, move point w.r.t. selected*

*point by deltax and deltay \*/*

PointsLocation.splice(points+2, 0, new scope.Point(selected\_x + deltax, tempy + deltay ));

            pathsPointsfitsX.splice(points+2, 0, selected\_x + deltax);

            pathsPointsfitsY.splice(points+2, 0, tempy + deltay);

            break;

        }

    }

*/\* deselect point and draw new spline \*/*

    selected\_x = null;

    selected\_y = null ;

    draw\_spline();

});

**Toggle contrast settings button**

var cont = $('#toggleContrast');

*/\* toggle contrast settings \*/*

cont.click(function() {

*/\* get div element with id: contrast \*/*

    var div\_contrast = document.getElementById('contrast');

*/\* Default style.display is 'none', thus the div is hidden.*

*On click style.display switches between 'block' and 'none'.*

*The 'block' type ensures the display uses the entire line and not*

*inline. \*/*

if (div\_contrast.style.display === 'none') {

    div\_contrast.style.display = 'block';

    }

    else {

    div\_contrast.style.display = 'none';

    }

});

var gridMajor = $('#gridMajor');

*/\* major grid lines slider \*/*

gridMajor.click(function() {

*/\* clears #myCanvas \*/*

    scope.project.clear();

    major\_grid\_lines.lineColor = gridMajor.val()/10;

*/\* redraw axis \*/*

    draw\_axis();

*/\* draw spline \*/*

    draw\_spline();

});

var gridMinor = $('#gridMinor');

*/\* minor grid lines slider \*/*

gridMinor.click(function() {

*/\* clears #myCanvas \*/*

    scope.project.clear();

    minor\_grid\_lines.lineColor = gridMinor.val()/10;

*/\* redraw axis \*/*

    draw\_axis();

*/\* draw spline \*/*

    draw\_spline();

});

1. A *resize* attribute is added to the <canvas> element. While this is not a HTML5 supported attribute, for yet unknown reasons this does ensure the drawn background fits the canvas. [↑](#footnote-ref-1)