Bezier curves

Mathematics, vector graphics and art

Subject:
Mathematics

Teodor Wójcik
Candidate M15-D-000703-0040

33 Copernicus LO Warsaw

Contents

1 Introduction

One of my interest belongs to computer graphics. As my father is a photographer I remember spending a great part of my childhood retouching and tinkering his works in Photoshop under his guidance. With time I became fairly accustomed with the world of pixels. Manipulating their colors and placement slowly stopped to be a challenge. It was about high school when I saw my friend designing illustrations. I was amazed by smoothness of all the lines and infinite scalability of all his works. He wasn't using raster graphics, where all the tiny pixels sum up like puzzles to create an image, but vector graphics, where shapes and curves base on mathematical equations and therefore can be freely altered and reproduced using mathematical tools without quality loss. Simplicity, cleanness and sense of purity of vector graphics reminded me of works of one my favourite abstractionist painter - Pablo Picasso. Raw illustrations produced in vector graphics closely resembled Picasso's sketches, in which using simple curves he captured essence of portrayed animals. From that moment, vector graphics became one of my great interests.

Although I never had problem in achieving certain level of fluency in use of graphic programs, I never really understood foundations that layed behind them. I intuitively predicted that it is mathematics that gives life to the world of graphics and I was always intrigued to examine underlying mathematic formulas. After almost three years of the IB Mathematics program, I feel that I am ready to thoroughly investigate mathematic concepts that underlie various fields of knowledge. In this essay I will analyze mathematics working behind one of fundamental tools of the world of vector graphics - Bezier curves. To prove their power and universality I will try to recreate Pablo Picasso's famous "Bull" painting using just mathematical formulas.

2 Bezier curves

Bezier curves were introduced in 1962 [3] by a french engineer Pierre Bezier. He worked as a car designer Renault and when he needed an effective method of accurately describing his visions, he presented *Bezier curves* and all the corresponding mathematical tools that enabled him to recreate his projections. Although some mathematical foundation were previously laid by a Russian mathematician Sergei Natanovich Bernstein in 1912 in the form of *Bernstein polynomials* and later in 1959 by a french mathematician Paul de Casteljau, who invented *De Casteljau's algorithm* - a method of recursive evaluating *Bernstein polynomials*, it is Pierre Bezier who applied and further developed all these methods into an acclaimed design tool.

As *Bezier curves* differ from functions learnt during IB Mathematics lessons, I'd like to first introduce concept of parametric functions.

2.1 Parametric functions

Functions that were introduced in IB Mathematics curriculum assign certain value to a variable x using notation as below:

$$f(x) = sinx$$

Bezier curves allow to relate two functions by a parameter t as follows[2]:

$$f(a) = sin(a)$$

$$f(b) = cos(b)$$

into:

$$\begin{cases} f_a(t) = sin(t) \\ f_b(t) = cos(t) \end{cases}$$

now when we relate both functions to the Cartesian coordinations, use application of parametric equations slowly becomes apparent:

$$\begin{cases} x = sin(t) \\ y = cos(t) \end{cases}$$

Presented example visualizes following graph generated using Geogebra. Color blue and red illustrates sine and cosine functions plotted for variable x. Color green illustrates sine and cosine functions binded together by parameter t in the form of aforementioned parametric equation.

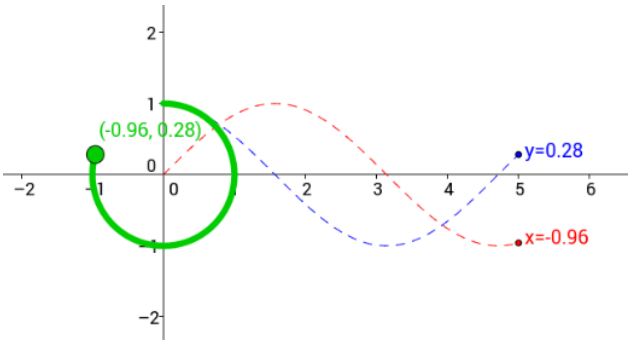

Figure 1: Graph for $t = 5$

2.2 Bezier curve definition

Bezier curves resemble polynomials known from IB Maths classes similar to this [2]:

$$f(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$$

where parameter t replaces variable x. As t can have values between 0 and 1 its consecutive orders will look like this:

$$\begin{aligned} linear &= (1-t) + t \\ square &= (1-t)^2 + 2 \cdot (1-t) \cdot t + t^2 \\ cubic &= (1-t)^3 + 3 \cdot (1-t)^2 \cdot t + 3 \cdot (1-t) \cdot t^2 + t^3 \end{aligned}$$

This structure resembles the *Binomial coefficients* formula and indeed definition for *Bezier curve* is expressed by the equation:

$$Bezier(n,t) = \sum_{i=0}^{n} \underbrace{\binom{n}{i}}_{binomial\ term} \cdot \underbrace{(1-t)^{n-i} \cdot t^i}_{polynomial\ term}$$

Aforementioned formula describes base *Bezier curve*. To effectively manipulate *Bezier curves* into forming desired shapes, points need to be multiple by values that change their strength. These controlling points are conventionally called *weights* as they "draw" the curve toward themselves between other controlling coordinates as the parameter t increases, therefore altering the curves path. Final description of *Bezier curves* states as follows:

$$Bezier(n,t) = \sum_{i=0}^{n} \underbrace{\binom{n}{i}}_{binomial\ term} \cdot \underbrace{(1-t)^{n-i} \cdot t^i}_{polynomial\ term} \cdot \underbrace{w_i}_{weight}$$

To illustrate this concept, example cubic *Bezier curve* will be generated. w_0 represents the starting coordinates equal to $(0,80)$ and w_3 represents the last coordinates equal to $(200,0)$. There are two control coordinates w_1 equal to $(40,20)$ and w_3 equal to $(120,60)$.

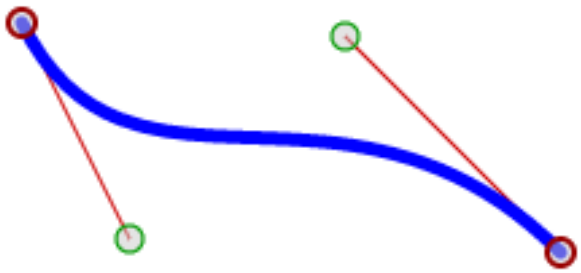

Figure 2: Example *Bezier curve*

Below are presented examples of three *Bezier curves* correspondingly [1] : linear, quadratic and cubic, that further illustrate *Bezier curves*

Figure 3: Sample linear *Bezier curve*

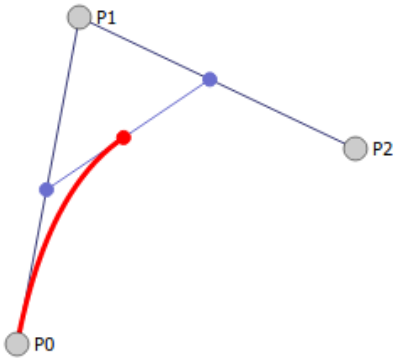

Figure 4: Sample quadratic *Bezier curve*

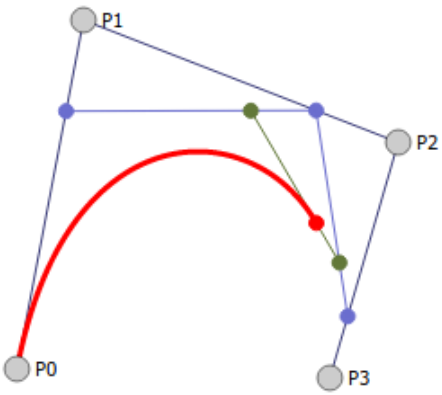

Figure 5: Sample cubic *Bezier curve*

At first theory behind *Bezier curves* might have seemed a bit too unfamiliar for me, but after thorough investigation, *Bezier curves* appeared to be very intuitive. Without being overly complex, they proved to be a powerful tool for designing curves. I was amazed by almost infinite possibilities of plotted shapes of the curves that could be produced by adding and altering consecutive coordinate points.

3 Picasso's painting

Pablo Picasso's animal sketches consist of a few simple curves that together try to capture the "essence" of the animal. From the very moment I saw his works, I was tempted to recreate them with the use of vector graphics to see the curves and mathematics behind them. I'm going to draw Pablo Picasso's most famous "Bull" using nine cubic *Bezier curves*. I will manually set coordinates of every control point of every curve and draw them using Javascript code.

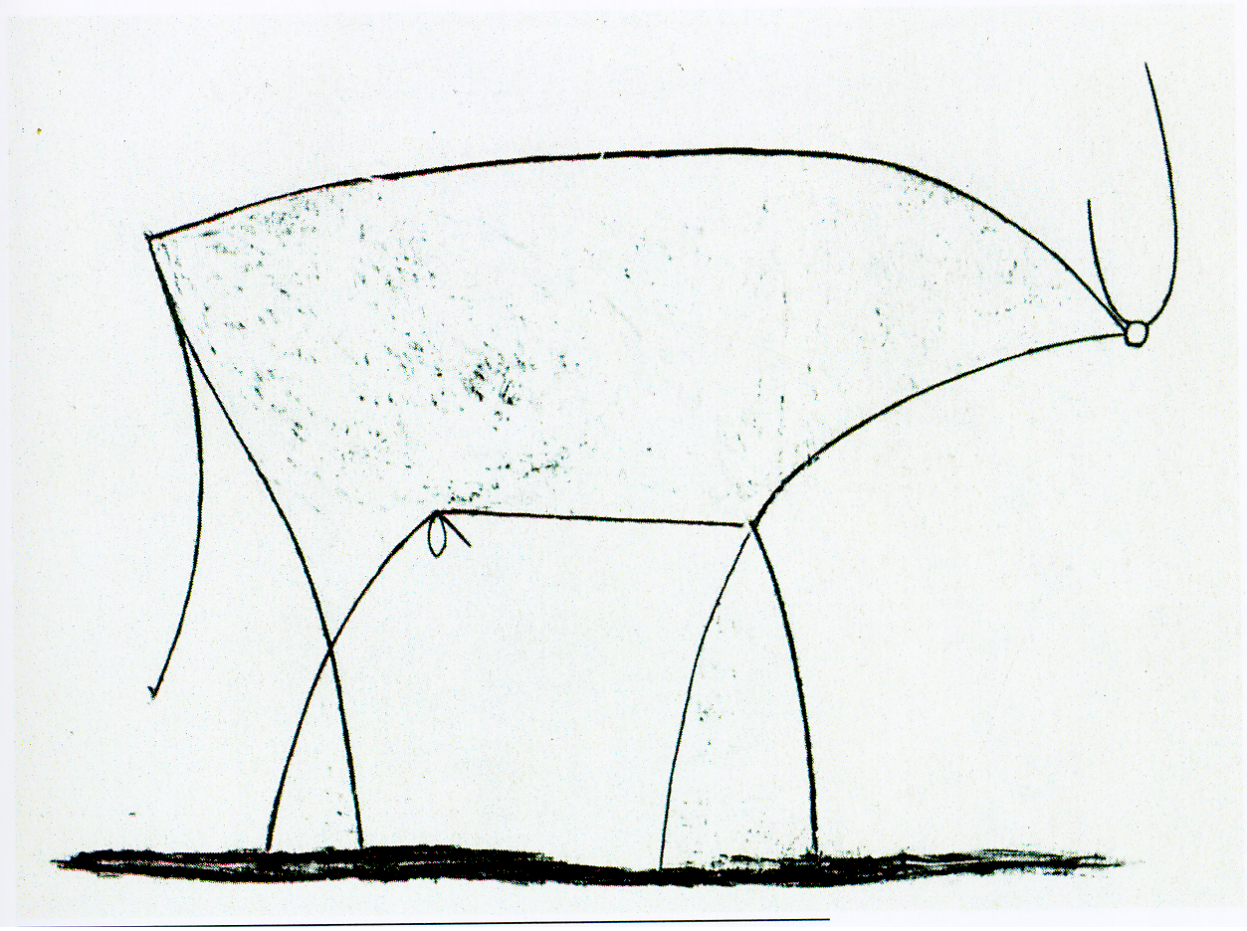

Figure 6: Pablo Picasso's "Bull"

3.1 Curves

This is a list showing which curve represents which part of the bull on the painting:

Curve 1	⇒	ground
Curve 2	⇒	back-left leg and tummy
Curve 3	⇒	back-right leg
Curve 4	⇒	tail
Curve 5	⇒	back
Curve 6	⇒	front-left leg
Curve 7	⇒	front-right leg
Curve 8	⇒	head
Curve 9	⇒	vitals

3.2 Data

These are the coordinates of all control points of plotted *Bezier curves*:

	x_0	y_0	x_1	y_1	x_2	y_2	x_3	y_3
Curve 1	101	312	315	315	253	316	425	309
Curve 2	138	309	144	256	156	193	320	214
Curve 3	114	124	142	216	141	162	191	312
Curve 4	115	125	138	211	96	244	100	242
Curve 5	114	124	187	93	409	72	431	153
Curve 6	277	316	290	258	324	167	431	153
Curve 7	319	213	350	271	349	297	345	314
Curve 8	423	91	417	159	462	204	455	44
Curve 9	192	222	198	219	199	236	206	237

3.3 Equations

$$\text{Curve 1}\begin{cases} x_0 = 101\cdot(1-t)^3 + 315\cdot 3\cdot(1-t)^2\cdot t + 253\cdot 3\cdot(1-t)\cdot t^2 + 425\cdot t^3 \\ y_0 = 312\cdot(1-t)^3 + 315\cdot 3\cdot(1-t)^2\cdot t + 316\cdot 3\cdot(1-t)\cdot t^2 + 309\cdot t^3 \end{cases}$$

$$\text{Curve 2}\begin{cases} x_1 = 138\cdot(1-t)^3 + 144\cdot 3\cdot(1-t)^2\cdot t + 156\cdot 3\cdot(1-t)\cdot t^2 + 320\cdot t^3 \\ y_1 = 309\cdot(1-t)^3 + 256\cdot 3\cdot(1-t)^2\cdot t + 193\cdot 3\cdot(1-t)\cdot t^2 + 214\cdot t^3 \end{cases}$$

$$\text{Curve 3}\begin{cases} x_2 = 114\cdot(1-t)^3 + 142\cdot 3\cdot(1-t)^2\cdot t + 141\cdot 3\cdot(1-t)\cdot t^2 + 191\cdot t^3 \\ y_2 = 124\cdot(1-t)^3 + 216\cdot 3\cdot(1-t)^2\cdot t + 162\cdot 3\cdot(1-t)\cdot t^2 + 312\cdot t^3 \end{cases}$$

$$\text{Curve 4}\begin{cases} x_3 = 115\cdot(1-t)^3 + 138\cdot 3\cdot(1-t)^2\cdot t + 96\cdot 3\cdot(1-t)\cdot t^2 + 100\cdot t^3 \\ y_3 = 125\cdot(1-t)^3 + 211\cdot 3\cdot(1-t)^2\cdot t + 244\cdot 3\cdot(1-t)\cdot t^2 + 242\cdot t^3 \end{cases}$$

$$\text{Curve 5}\begin{cases} x_4 = 114\cdot(1-t)^3 + 187\cdot 3\cdot(1-t)^2\cdot t + 409\cdot 3\cdot(1-t)\cdot t^2 + 431\cdot t^3 \\ y_4 = 124\cdot(1-t)^3 + 93\cdot 3\cdot(1-t)^2\cdot t + 72\cdot 3\cdot(1-t)\cdot t^2 + 153\cdot t^3 \end{cases}$$

$$\text{Curve 6}\begin{cases} x_5 = 277\cdot(1-t)^3 + 290\cdot 3\cdot(1-t)^2\cdot t + 324\cdot 3\cdot(1-t)\cdot t^2 + 431\cdot t^3 \\ y_5 = 316\cdot(1-t)^3 + 258\cdot 3\cdot(1-t)^2\cdot t + 167\cdot 3\cdot(1-t)\cdot t^2 + 153\cdot t^3 \end{cases}$$

$$\text{Curve 7}\begin{cases} x_6 = 319\cdot(1-t)^3 + 350\cdot 3\cdot(1-t)^2\cdot t + 349\cdot 3\cdot(1-t)\cdot t^2 + 345\cdot t^3 \\ y_6 = 213\cdot(1-t)^3 + 271\cdot 3\cdot(1-t)^2\cdot t + 297\cdot 3\cdot(1-t)\cdot t^2 + 314\cdot t^3 \end{cases}$$

$$\text{Curve 8}\begin{cases} x_7 = 423\cdot(1-t)^3 + 417\cdot 3\cdot(1-t)^2\cdot t + 462\cdot 3\cdot(1-t)\cdot t^2 + 455\cdot t^3 \\ y_7 = 91\cdot(1-t)^3 + 159\cdot 3\cdot(1-t)^2\cdot t + 204\cdot 3\cdot(1-t)\cdot t^2 + 44\cdot t^3 \end{cases}$$

$$\text{Curve } 9 \begin{cases} x_8 = 192 \cdot (1-t)^3 + 198 \cdot 3 \cdot (1-t)^2 \cdot t + 199 \cdot 3 \cdot (1-t) \cdot t^2 + 206 \cdot t^3 \\ y_8 = 222 \cdot (1-t)^3 + 219 \cdot 3 \cdot (1-t)^2 \cdot t + 236 \cdot 3 \cdot (1-t) \cdot t^2 + 237 \cdot t^3 \end{cases}$$

3.4 Drawings

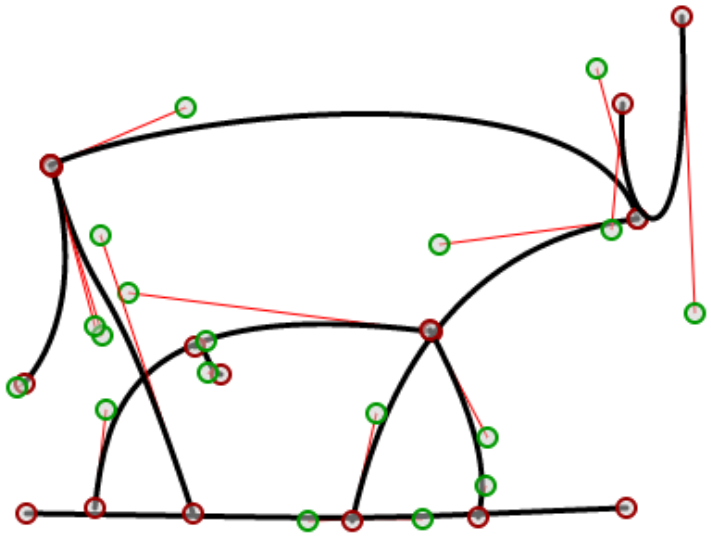

Figure 7: "Bull" generated using 9 *Bezier curves* - with starting and ending coordinate points marked as red and controlling coordinate points marked as green

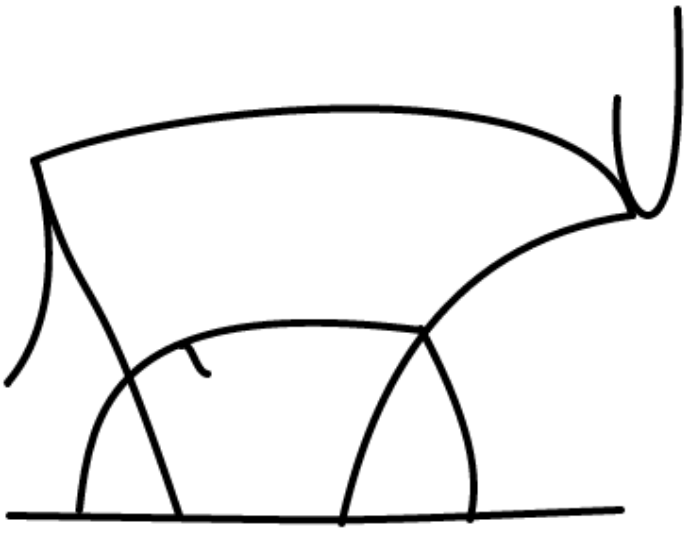

Figure 8: "Bull" generated using 9 *Bezier curves* - clean

4 Conclusion

In this paper I have shown basic maths that underlies drawings of curves in vector graphics. I was surprised that the necessary theories appeared to be comprehensible and understandable for an IB student. Realizing mathematic concepts behind vector graphics gave me broader knowledge of the subject and furthered my graphic skills. Now I can see how my graphic curves are indeed shaped by mathematic formulas, therefore I am able to modify them explicitly, which proves to be an invaluable support during graphic work. I strongly encourage every beginner graphic artist to investigate underlying mathematics of the world of vector graphics, as the mathematic concepts are relatively easy to grasp and benefit every-day graphic work. I also felt a strong sense of satisfaction and fulfillment while generating my first *Bezier curves*. My own projection of Pablo Picasso's "Bull", proved to be in my and my peers opinion, closely similar to the original. It has shown ease of use together with high versatility of *Bezier curves*. Concluding, it was a rewarding experience as it has joined my three great interests: maths, art and graphics.

References

[1] J. Davies. Animated bezier curves. URL <http://www.jasondavies.com/animated-bezier/>. Accessed: 2014-12-30.

[2] M. Kamermans. A primer on bezier curves. URL <http://pomax.github.io/bezierinfo/>. Accessed: 2014-12-30.

[3] J. Kun. Bezier curves and picasso. URL <http://jeremykun.com/2013/05/11/bezier-curves-and-picasso/> Accessed: 2014-12-30.

A Appendix

A.1 Code used to generate Pablo Picasso's "Bull" using *Bezier curves*

Code was intially developed by halfsoft (*http://jsfiddle.net/halfsoft/Gsz2a/*) and further adjusted to the needs of this essay:

```
(function () {
    function curvePoint() {
        this.x = 0;
        this.y = 0;
        this.computeDistance = function (pt) {
            var xd = pt.x - this.x,
                yd = pt.y - this.y;
            return Math.sqrt(xd * xd + yd * yd);
        };
    }
    function curve() {
        this.isValid = false;

        this.p1 = new curvePoint();
        this.p2 = new curvePoint();
        this.cp1 = new curvePoint();
        this.cp2 = new curvePoint();
        this.set = function (params) {
            params = params.replace(/[^0-9,]/g, '');
            params = params.replace(/,,/g, '');
            var split = params.split(",");
            this.p1.x = 1 * split[0];
            this.p1.y = 1 * split[1];
            this.cp1.x = 1 * split[2];
            this.cp1.y = 1 * split[3];
            this.cp2.x = 1 * split[4];
            this.cp2.y = 1 * split[5];
            this.p2.x = 1 * split[6];
            this.p2.y = 1 * split[7];

            this.isValid = (split.length >= 8);
        };
    }

    var current = 0;
    var curves = [];
    var selectionRadius;
    var deleteMode = false;
    var canvas, ctx, code, style, drag = null,
        dPoint;

    function addCurve() {
        var curveParams = prompt("Enter curve params",
                "100,250,150,100,350,100,400,250");
```

```
        var tCurve = new curve();
        tCurve.set(curveParams);

        if (tCurve.isValid) curves.push(tCurve);
        DrawCanvas();
    }

    function delCurve() {
        deleteMode = !deleteMode;
        DrawCanvas();
    }

    // define initial points
    function Init() {
        // default styles
        style = {
            curve: {
                width: 10,
                color: "#000000"
            },
            cpline: {
                width: 0,
                color: "#ffffff"
            },
            point: {
                radius: 0,
                width: 0,
                color: "#900",
                fill: "rgba(200,200,200,0.0)",
                arc1: 0,
                arc2: 2 * Math.PI
            },
            cppoint: {
                radius: 0,
                width: 0,
                color: "#090",
                fill: "rgba(200,200,200,0.0)",
                arc1: 0,
                arc2: 0 * Math.PI
            }
        };
        selectionRadius = style.point.radius * 1.5;
        // line style defaults
        ctx.lineCap = "round";
        ctx.lineJoin = "round";
        // event handlers
        canvas.onmousedown = DragStart;
        canvas.onmousemove = Dragging;
        canvas.onmouseup = canvas.onmouseout = DragEnd;
        DrawCanvas();
    }
    // draw canvas
    function DrawCanvas() {

        style.curve.width = document.getElementById("strokeWidth").value;
        if (deleteMode) {
            ctx.fillStyle = "#F5B8C2";
        } else {
            ctx.fillStyle = "#FFFFFF";
        }
        ctx.fillRect(0, 0, canvas.width, canvas.height);

        if (img && img.src !== "") {
            ctx.drawImage(img, 0, 0);
        }

        for (i = 0; i < curves.length; i++) {
            point = curves[i];
            if (!deleteMode) {
                // control lines
            }
            // curve
            ctx.lineWidth = style.curve.width;
            ctx.strokeStyle = style.curve.color;
            ctx.beginPath();
            ctx.moveTo(point.p1.x, point.p1.y);
            ctx.bezierCurveTo(point.cp1.x, point.cp1.y, point.cp2.x, point.cp2.y,
                    point.p2.x, point.p2.y);
            ctx.stroke();
            // control points
            for (var p in point) {
                if (p == "cp1" || p == "cp2") {
                    ctx.lineWidth = style.cppoint.width;
                    ctx.strokeStyle = style.cppoint.color;
                    ctx.fillStyle = style.cppoint.fill;
                } else {
                    ctx.lineWidth = style.point.width;
                    ctx.strokeStyle = style.point.color;
                    ctx.fillStyle = style.point.fill;
                }
                if (!deleteMode || !(p == "cp1" || p == "cp2")) {
                    ctx.beginPath();
                    ctx.arc(point[p].x, point[p].y, style.point.radius,
                                style.point.arc1, style.point.arc2, true);
                    ctx.fill();
                    ctx.stroke();
                }
            }
        }
    }
    // show canvas code
    function ShowCode() {
        var code =
            "curveWidth:" + style.curve.width + "\n";
        for (i = 0; i < curves.length; i++) {
            point = curves[i];
            code += point.p1.x + ", " + point.p1.y + ", " + point.cp1.x + ", " +
                    point.cp1.y + ", " + point.cp2.x + ", " + point.cp2.y +
                    ", " + point.p2.x + ", " + point.p2.y + ",\n";
        }
        alert(code);
    }
    function LoadCode() {
        $("#dialog").dialog("open");
    }
    // start dragging
    function DragStart(e) {
        var whichButton = e.which;
        if (whichButton == 3 || whichButton == 2) {
            e.preventDefault();
        }
        var ctrlPressed = e.ctrlKey;
        e = MousePos(e);
        var dx, dy;
        for (var i = 0; i < curves.length; i++) {

            selectedPoint = curves[i];

            for (var p in selectedPoint) {
                dx = selectedPoint[p].x - e.x;
                dy = selectedPoint[p].y - e.y;
                if ((dx * dx) + (dy * dy) < selectionRadius * selectionRadius) {
                    if (whichButton == 2 || whichButton == 3) {
                        selectedBezier = curves[i];
                        translatingObject = true;
                        dPoint = e;
                        canvas.style.cursor = "move";
                        return;
                    }

                    if (deleteMode) {
                        if (p == "p1" || p == "p2") {
                            curves.splice(i, 1);
                        } else {
                            return;
                        }
                    }
                    drag = p;
                    dPoint = e;
                    canvas.style.cursor = "move";
                    isSnapped = false;
                    var movepoint = (drag + "").length > 2 ? drag.substring(1) :
                                drag;
```

```
                    for (i = 0; i < curves.length; i++) {
                        tPoint = curves[i];
                        if (tPoint == selectedPoint) continue;
                        if (selectedPoint[movepoint].computeDistance(tPoint.p1)
                                <= selectionRadius) {
                            isSnapped = snapping.checked;
                            selectedBezier = selectedPoint;
                        } else if
                                (selectedPoint[movepoint].computeDistance(tPoint.p2)
                                <= selectionRadius) {
                            isSnapped = snapping.checked;
                            selectedBezier = selectedPoint;
                        }
                    }

                    return;
                }
            }
        }
        if (whichButton == 2 || whichButton == 3) {
            translating = true;
            dPoint = e;
            canvas.style.cursor = "move";
        }
    }
    var translating = false;
    var translatingObject = false;
    var selectedBezier = null;
    var isSnapped = false;
    // dragging
    function Dragging(e) {
        var i, tPoint;
        if (translating) {
            e = MousePos(e);
            for (i = 0; i < curves.length; i++) {
                tPoint = curves[i];
                tPoint.p1.x += e.x - dPoint.x;
                tPoint.p1.y += e.y - dPoint.y;
                tPoint.p2.x += e.x - dPoint.x;
                tPoint.p2.y += e.y - dPoint.y;
                tPoint.cp1.x += e.x - dPoint.x;
                tPoint.cp1.y += e.y - dPoint.y;
                tPoint.cp2.x += e.x - dPoint.x;
                tPoint.cp2.y += e.y - dPoint.y;
            }
            canvas.style.cursor = "move";
            DrawCanvas();
            dPoint = e;
            return;
        }

        if (translatingObject) {
            e = MousePos(e);
            tPoint = selectedBezier;
            tPoint.p1.x += e.x - dPoint.x;
            tPoint.p1.y += e.y - dPoint.y;
            tPoint.p2.x += e.x - dPoint.x;
            tPoint.p2.y += e.y - dPoint.y;
            tPoint.cp1.x += e.x - dPoint.x;
            tPoint.cp1.y += e.y - dPoint.y;
            tPoint.cp2.x += e.x - dPoint.x;
            tPoint.cp2.y += e.y - dPoint.y;
            canvas.style.cursor = "move";
            DrawCanvas();
            dPoint = e;
            return;
        }

        if (drag) {
            e = MousePos(e);
            if (isSnapped) {
                if (drag == "p1" || drag == "p2") {
                    for (i = 0; i < curves.length; i++) {
                        tPoint = curves[i];
                        if (tPoint == selectedPoint) continue;
                        var lockedPoint = "";
                        if (selectedPoint[drag].computeDistance(tPoint.p1) <=
                                selectionRadius) {
                            lockedPoint = "p1";
                        } else if (selectedPoint[drag].computeDistance(tPoint.p2)
                                <= selectionRadius) {
                            lockedPoint = "p2";
                        }

                        if (lockedPoint !== "") {
                            tPoint[lockedPoint].x += e.x - dPoint.x;
                            tPoint[lockedPoint].y += e.y - dPoint.y;
                            if (lockedPoint == "p1" || lockedPoint == "p2") {
                                tPoint["c" + lockedPoint].x += e.x - dPoint.x;
                                tPoint["c" + lockedPoint].y += e.y - dPoint.y;
                            }
                        }
                    }
                }
            }
            selectedPoint[drag].x += e.x - dPoint.x;
            selectedPoint[drag].y += e.y - dPoint.y;
            if (drag == "p1" || drag == "p2") {
                selectedPoint["c" + drag].x += e.x - dPoint.x;
                selectedPoint["c" + drag].y += e.y - dPoint.y;
            }
            dPoint = e;
            DrawCanvas();
        } else {
            e = MousePos(e);
            canvas.style.cursor = "default";

            for (i = 0; i < curves.length; i++) {
                tPoint = curves[i];
                if (tPoint.p1.computeDistance(e) <= selectionRadius ||
                        tPoint.p2.computeDistance(e) <= selectionRadius
                        || (!deleteMode &&
                        (tPoint.cp1.computeDistance(e) <= selectionRadius
                        || tPoint.cp2.computeDistance(e) <=
                        selectionRadius))) {
                    canvas.style.cursor = deleteMode ? "crosshair" : "move";
                }
            }
        }
    }
    // end dragging
    function DragEnd(e) {
        translating = false;
        translatingObject = false;
        if (drag && snapping.checked) {
            if (drag == "p1" || drag == "p2") {
                for (var i = 0; i < curves.length; i++) {
                    tPoint = curves[i];
                    if (tPoint == selectedPoint) continue;
                    if (selectedPoint[drag].computeDistance(tPoint.p1) <=
                            selectionRadius) {
                        selectedPoint[drag].x = tPoint.p1.x;
                        selectedPoint[drag].y = tPoint.p1.y;
                    } else if (selectedPoint[drag].computeDistance(tPoint.p2) <=
                            selectionRadius) {
                        selectedPoint[drag].x = tPoint.p2.x;
                        selectedPoint[drag].y = tPoint.p2.y;
                    }


                }
            }
        }
        drag = null;
        DrawCanvas();
    }
    // event parser
    function MousePos(event) {
        event = (event ? event : window.event);
        return {
            x: event.pageX - canvas.offsetLeft,
            y: event.pageY - canvas.offsetTop
        };
    }
```

```
    var url, img, f, src;
    function fileSelected(event) {
        img = new Image();
        f = document.getElementById("uploadimage").files[0];
        url = window.URL || window.webkitURL;
        src = url.createObjectURL(f);

        img.src = src;
        img.onload = function () {
            DrawCanvas();
            url.revokeObjectURL(src);
        };
    }
    // start
    canvas = document.getElementById("canvas");
    snapping = document.getElementById("snappingEnabled");
    canvas.addEventListener('contextmenu', function (e) {
        e.preventDefault();
        return false;
    }, false);
    document.getElementById("showCodeBut").addEventListener("click",
                    ShowCode, false);
    document.getElementById("loadCodeBut").addEventListener("click",
                    LoadCode, false);
    document.getElementById("addCurveBut").addEventListener("click",
                    addCurve, false);
    document.getElementById("delCurveBut").addEventListener("click",
                    delCurve, false);
    document.getElementById("uploadimage").addEventListener("change",
                    fileSelected, false);
    if (canvas.getContext) {
        ctx = canvas.getContext("2d");
        Init();
    }

    $("#dialog").dialog({
        autoOpen: false,
        buttons: {
            Load: function () {
                curves = [];

                var splitCode = $("#loadData").val().split("\n");

                for (var i = 0; i < splitCode.length; i++) {
                    if (splitCode[i].indexOf("curve:") === 0) {
                        var tCurve = new curve();
                        tCurve.set(splitCode[i]);
                        if (tCurve.isValid) curves.push(tCurve);
                    } else if (splitCode[i].indexOf("curveWidth:") === 0) {
                        document.getElementById("strokeWidth").value =
                                splitCode[i].substring(11);
                    } else {
                        var tCurve = new curve();
                        tCurve.set(splitCode[i]);
                        if (tCurve.isValid) curves.push(tCurve);
                    }

                }
                DrawCanvas();
                $(this).dialog("close");
            },
            Cancel: function () {
                $(this).dialog("close");
            }
        }
    });

})();
```