Workshop: Advanced JSXGraph

Vol. 7

Alfred Wassermann

UNIVERSITÄT
BAYREUTH

9-06-2021

Contents

Preliminaries

Include JSXGraph

- JSXGraph skeleton page:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>JSXGraph template</title>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <link href="https://cdn.jsdelivr.net/npm/jsxgraph@1.2.3/distrib/jsxgraph.css"
        rel="stylesheet" type="text/css" />
    <script src="https://cdn.jsdelivr.net/npm/jsxgraph@1.2.3/distrib/jsxgraphcore.
        js" type="text/javascript" charset="UTF-8"></script>

    <!-- The next line is optional: MathJax -->
    <script src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js" id="
        MathJax-script" async></script>
  </head>
  <body>

  <div id="jxgbox" class="jxgbox" style="width:500px; height:200px;"></div>

  <script>
    var board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox: [-5, 2, 5, -2]});
  </script>

  </body>
</html>
```

- See JSXGraph handbook (in development): https://ipesek.github.io/jsxgraphbook/

Display of coordinates

There are several approaches to display coordinates of points:

- infobox element
- coordinates as point labels
- text element relative to a point (the *anchor* element)

Infobox element

- There is a demand to change the styling of the infobox element. This can be realized with a mixture of CSS properties and JSXGraph attributes.

- The infobox element is shown whenever a JSXGraph point is highlighted (by e.g. “mouseover”).
- CSS styling of infobox example: https://jsfiddle.net/o7rs5n3u/1/

```
.JXGinfobox {
  padding:10px;
  border: solid black 2px;
  border-radius: 10px;
  background-color: #dddddd;
}
```

```
JXG.Options.infobox.fontSize = 16;
JXG.Options.infobox.strokeColor = 'black';

const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 5, 5, -5], axis: true
});

var p = board.create('point', [1, 1], {
            showInfobox: true,
            infoboxDigits:4
        });
```

- If there is custom information to be displayed, the infobox text can be customized, too. Overwrite `board.highlightInfobox(x, y, el)` or `board.highlightCustomInfobox(text, el)`.
- Custom infobox example: https://jsfiddle.net/25Lbdrkt/

```
JXG.Options.infobox.anchorY = 'bottom';
JXG.Options.infobox.anchorX = 'right';
JXG.Options.infobox.strokeColor = 'blue';

const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 5, 5, -5], axis:true
});

board.infobox.distanceX = 0;
board.infobox.distanceY = 10;

board.highlightInfobox = function(x, y , el) {
        this.infobox.setText( x + '<br>' + y + '<br> digits: ' + el.visProp.
            infoboxdigits );
};

var p = board.create('point', [1, 1], {
            showInfobox: true,
            infoboxDigits:4
        });
```

Show coordinates in labels

- If the coordinates of a point are shown in its label they are visible also without the point being highlighted.
- Example: https://jsfiddle.net/d20gLqs8/1/

```
var p = board.create('point', [1, 1], {
            showInfobox: false
        });
p.setAttribute({name:
    () => '<b>' + p.X().toFixed(3) + ' | ' + p.Y().toFixed(3) + '</b>'
 });

 var q = board.create('point', [-2, -2], {
            showInfobox: false,
            label: {parse: false, useMathJax: true}
        });

q.setAttribute({name:
    () => '\\( \\{' + q.X().toFixed(1) + '\\; | ' + q.Y().toFixed(1) + '\\} \\)'
 });
```

Text relative to a point

- A generalization of the label element is to set the position of text element relative to another element using the attribute `anchor`.
- With `fixed:true/false` the dragging of the text element can be controlled.
- Example: again https://jsfiddle.net/d20gLqs8/1/

```
var s = board.create('text', [-1, -1,
    () => '\\( q = \\{' + q.X().toFixed(1) + '\\; | ' + q.Y().toFixed(1) + '\\} \\)
        '
 ], {
      parse: false,
      useMathJax: true,
      anchor: q,
      fixed: true
 });
```

Traces

- JSXGraph elements can leave a "trace": `trace:true`.
- All traces can be removed with the clearTrace-button, enabled by `showClearTraces:true`.
- Example: https://jsfiddle.net/cvp4qsa6/

```
const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 5, 5, -5], axis: false,
    showClearTraces: true
});

var abc = board.create('polygon', [[-3, -3], [3, -3], [0,2]], {
            fillColor: 'none',
      highlightFillColor: 'none',
      borders: {strokeColor: 'black', strokeWidth: 0.5}
});

abc.vertices[2].on('drag', function(evt) {
  var p = abc.vertices[2];
    p.moveTo([p.X(), 2]);
});

var l1 = board.create('perpendicular', [abc.borders[2], abc.vertices[1]], {
    strokeColor: 'blue', strokeWidth: 0.5
});
var l2 = board.create('perpendicular', [abc.borders[0], abc.vertices[2]], {
    strokeColor: 'blue', strokeWidth: 0.5
});

var l3 = board.create('perpendicular', [abc.borders[1], abc.vertices[0]], {
    strokeColor: 'blue', strokeWidth: 0.5
});

var q = board.create('intersection', [l1, l2], {trace: true, color: 'orange'});
```

Trace curve

- Given a glider point and a point dependent on this glider, the trace curve of that point can be computed.
- Example: https://jsfiddle.net/rqxbhc7d/1/

```
const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 9, 5, -1], axis: true,
    showClearTraces: true
});

var x = board.defaultAxes.x;
var p = board.create('point', [0, 5], {visible: false, withLabel: false});
var l = board.create('parallel', [x, p], {
      fixed: false,
      dash: 2,
      strokeWidth: 0.5
});

var a = board.create('glider', [-3, 0, x]);
var b = board.create('glider', [3, 0, x]);

var c = board.create('glider', [0, 5, l], {name: 'C'});
var abc = board.create('polygon', [a, b, c], {
            fillColor: 'none',
      highlightFillColor: 'none',
      borders: {strokeColor: 'black', strokeWidth: 0.5}
});

var l1 = board.create('perpendicular', [abc.borders[2], abc.vertices[1]], {
    strokeColor: 'blue', strokeWidth: 0.5
});
var l2 = board.create('perpendicular', [abc.borders[0], abc.vertices[2]], {
    strokeColor: 'blue', strokeWidth: 0.5
});

var l3 = board.create('perpendicular', [abc.borders[1], abc.vertices[0]], {
    strokeColor: 'blue', strokeWidth: 0.5
});

var q = board.create('intersection', [l1, l2], {trace: true, color: 'orange'});
var curve = board.create('tracecurve', [c, q]);
```

Extend JSXGraph with new objects

Naive implementation

- 3D projection copied from https://en.wikipedia.org/wiki/3D_projection
- Example: https://jsfiddle.net/gewvn1zo/

```
const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 5, 5, -5], axis: false
});

var max_angle = 2 * Math.PI;
var max_pos = 20;

// Camera angles and position:
var theta_x = board.create('slider', [[1, 4], [3.5, 4],     [0, 0, max_angle]]);
var theta_y = board.create('slider', [[1, 3.5], [3.5, 3.5], [0, 0, max_angle]]);
var theta_z = board.create('slider', [[1, 3], [3.5, 3],     [0, Math.PI, max_angle
    ]]);
var c_x = board.create('slider', [[1, 2.5], [3.5, 2.5], [-max_pos, 0, max_pos]]);
var c_y = board.create('slider', [[1, 2], [3.5, 2],     [-max_pos, 0, max_pos]]);
var c_z = board.create('slider', [[1, 1.5], [3.5, 1.5], [-max_pos, 0, max_pos]]);

// 3D points
var a = board.create('point', [0,0]);
a.coords3D = [1, 1, 1];

var b = board.create('point', [0,0]);
b.coords3D = [1, 3, 1];

var c = board.create('point', [0,0]);
```

```
c.coords3D = [2, 3, 1];

var d = board.create('point', [0,0]);
d.coords3D = [2, 2, 3];

// Segments connecting the points
var attr = {
        strokeWidth: 0.5
    };
var l1 = board.create('segment', [a, b], attr);
var l2 = board.create('segment', [a, c], attr);
var l3 = board.create('segment', [a, d], attr);
var l4 = board.create('segment', [b, c], attr);
var l5 = board.create('segment', [b, d], attr);
var l6 = board.create('segment', [c, d], attr);

// Camera
var cam = {
        c: [0, 0, 0],
        theta: [0, 0, 0],
        e: [0, 0, 1]
    };

// Perspective projection
// see https://en.wikipedia.org/wiki/3D_projection
var project = function(point) {
    var d = [0, 0, 0],
        x = point.coords3D[0] - cam.c[0],
        y = point.coords3D[1] - cam.c[1],
        z = point.coords3D[2] - cam.c[2],
        sx = Math.sin(cam.theta[0]),
        cx = Math.cos(cam.theta[0]),
        sy = Math.sin(cam.theta[1]),
        cy = Math.cos(cam.theta[1]),
        sz = Math.sin(cam.theta[2]),
        cz = Math.cos(cam.theta[2]);

    d[0] = cy * (sz * y + cz * x) - sy * z;
    d[1] = sx * (cy * z + sy * (sz * y + cz * x)) + cx * (cz * y - sz * x);
    d[2] = cx * (cy * z + sy * (sz * y + cz * x)) - sx * (cz * y - sz * x);

    point.coords.setCoordinates(JXG.COORDS_BY_USER,
        [
        cam.e[2] * d[2] + 1 * d[2],
        cam.e[2] * d[0] + cam.e[0] * d[2],
        cam.e[2] * d[1] + cam.e[1] * d[2]
        ], false);
};

// Recompute the 3D scene
var update_cam = function() {
    // Update camera angles and position
    cam.theta = [
            theta_x.Value(),
            theta_y.Value(),
            theta_z.Value()
        ];
    cam.c[0] = c_x.Value();
    cam.c[1] = c_y.Value();
    cam.c[2] = c_z.Value();

    // Project each point
    project(a);
    project(b);
    project(c);
    project(d);

    // Draw the points
    board.update();
};

// Event handlers to trigger the 3D projection
theta_x.on('drag', update_cam);
theta_y.on('drag', update_cam);
theta_z.on('drag', update_cam);
c_x.on('drag', update_cam);
c_y.on('drag', update_cam);
c_z.on('drag', update_cam);

// Initial projection
update_cam();
```

- Approach:
    - Sliders control camera angles and position.
    - Sliders trigger event to update camera and recompute projection of 3D points to the 2D JSXGraph board.
    - In `update_camera()` the camera position is updated and each point is projected to the JSXGraph board “by hand”.
- Cons:
    - 3D coordinates are not encapsuled.
    - Event handling has to be done “by hand”.

Add object type “point3D” to JSXGraph

The goal is to create a new JSXGraph object which resembles a 3D point that can be created by a usual call of

```
    var a = board.create('point3D', [x, y, z, cam], attributes);
```

In this little example, we demonstrate the necessary steps to enable such an object.

> The code efficiency is by no means optimized.

Necessary steps

Here, we will create a new element `'point3D'`. It is advisable to create a new JSXGraph object `camera`, too. To keep the example short, we postpone this as an exercise.

A new JSXGraph object can be created with the following steps:

1. Introduce a new method `JXG.createPoint3D(board, parents, attributes)` with parameters:

   - `board`: board variable
   - `parents`: array `[x, y, z, cam]` to define 3D point
   - `attributes`: JavaScript object containing key-value-pairs like `strokeColor:'blue'` which overwrite the default appearance

   The method has to return the newly created JSXGraph object.

2. Make this method available for `board.create('point3D', parents, attributes);` by calling

```
    JXG.registerElement('point3D', JXG.createPoint3D);
```

3. Define default attributes for `'point3D'` (optional):

```
JXG.Options.point3D = {
    strokeColor: 'blue',
    size: 1
};
```

4. Write documentation …

5. Use the new object, e.g.

```
var a = board.create('point3D', [-4, 1, 1, cam], {name: 'a'});
```

The inner workings of `JXG.createPoint3d()`

As mentioned above, the three parameters of `JXG.createPoint3d()` are

- `board`
- `parents`
- `attribute`

The recommended steps to create the new JSXGraph object are the following:

1. Test the parameters supplied in `parents` whether they are valid inputs. *Here, we skip this step.* Additionally, we would handle here the various allowed inputs, e.g. allowing input for coordinate `x` to be a number or a function returning a number. See the method `JXG.createPoint` in the JSXGraph source code how this can be done.
2. Merge the user supplied attributes with the default attributes.

```
// Merge user supplied attributes with default attributes of '3Dpoint'
attr = JXG.copyAttributes(attributes, board.options, 'point3D');
```

3. Create a JSXGraph object which can be modified to be a 3D point. In our case, this is a regular JSXGraph point. If the new JSXGraph object can not be based on already available JSXGraph objects, the task is much more complicated. Fortunately, for 3D points we can use a regular point:

```
el = board.create('point', [0, 0], attr);
```

4. Add the methods to compute the 3D projection:

```
el.coords3D = [parents[0], parents[1], parents[2]];
cam = parents[3];
el.addConstraint([getProj(el, 0, cam), getProj(el, 1, cam), getProj(el, 2, cam)]);
```

- We store the 3D coordinates in `el.coords3D`.
- With `el.addConstraint( [funcZ, funcX, funcY] )`, functions are registered which compute in each update of the board the 2D coordinates. Remember, JSXGraph uses “homogeneous” 2D coordinates (z, x, y) to be able to handle infinitely far elements. In doubt, just supply `el.addConstraint( [funcX, funcY] )`.
- At the time being, `el.addConstraint` is only available for points. For curves, we could use `updateDataArray`.

5. Return the new object.

The complete code

- Example: https://jsfiddle.net/0suxk3hp/2/

```
// -------------------------------------------------
// Separate file:

/**
  * parents: [x, y, z, cam]
  */
```

```
JXG.createPoint3D = function(board, parents, attributes) {
    var el, attr, cam;
    var getProj = function(point, i, cam) {
        return function() {
            cam.update_cam();
            var d = [0, 0, 0],
                x = point.coords3D[0] - cam.c[0],
                y = point.coords3D[1] - cam.c[1],
                z = point.coords3D[2] - cam.c[2],
                sx = Math.sin(cam.theta[0]),
                cx = Math.cos(cam.theta[0]),
                sy = Math.sin(cam.theta[1]),
                cy = Math.cos(cam.theta[1]),
                sz = Math.sin(cam.theta[2]),
                cz = Math.cos(cam.theta[2]);

            d[0] = cy * (sz * y + cz * x) - sy * z;
            d[1] = sx * (cy * z + sy * (sz * y + cz * x)) + cx * (cz * y - sz * x);
            d[2] = cx * (cy * z + sy * (sz * y + cz * x)) - sx * (cz * y - sz * x);

            var f = [
                    cam.e[2] * d[2] + 1 * d[2],
                    cam.e[2] * d[0] + cam.e[0] * d[2],
                    cam.e[2] * d[1] + cam.e[1] * d[2]
                ];
            return f[i];
        }
    };

    // Merge user supplied attributes with default attributes of '3Dpoint'
    attr = JXG.copyAttributes(attributes, board.options, 'point3D');

    el = board.create('point', [0, 0], attr);

    el.coords3D = [parents[0], parents[1], parents[2]];
    cam = parents[3];
    el.addConstraint([getProj(el, 0, cam), getProj(el, 1, cam), getProj(el, 2, cam)
        ]);

    return el;
};

JXG.registerElement('point3D', JXG.createPoint3D);

// Define default attributes
JXG.Options.point3D = {
    strokeColor: 'blue',
    fillColor: 'blue',
    size: 1
};

// --------------------------------------------------
// Here the new element 'point3D' is used:

var board = JXG.JSXGraph.initBoard("jxgbox", {boundingbox: [-5,5,5,-5],
    axis: false, showCopyright:true, showNavigation:true});
```

```
var max_angle = 2 * Math.PI;
var max_pos = 20;

// Camera angles and position:
var theta_x = board.create('slider', [[1, 4], [3.5, 4],     [0, 0, max_angle]]);
var theta_y = board.create('slider', [[1, 3.5], [3.5, 3.5], [0, 0, max_angle]]);
var theta_z = board.create('slider', [[1, 3], [3.5, 3],     [0, Math.PI, max_angle
    ]]);
var c_x = board.create('slider', [[1, 2.5], [3.5, 2.5], [-max_pos, -max_pos,
    max_pos]]);
var c_y = board.create('slider', [[1, 2], [3.5, 2],     [-max_pos, -max_pos,
    max_pos]]);
var c_z = board.create('slider', [[1, 1.5], [3.5, 1.5], [-max_pos, -3, max_pos]]);

// Camera
var cam = {
        c: [0, 0, 0],
        theta: [0, 0, 0],
        e: [0, 0, 1],
        update_cam: function() {
            cam.theta = [
                theta_x.Value(),
                theta_y.Value(),
                theta_z.Value()
            ];
            cam.c = [
                c_x.Value(),
                c_y.Value(),
                c_z.Value()
            ];
        }
    };

var o  = board.create('point3D', [0, 0, 0, cam], {name:'o'});
var ex = board.create('point3D', [5, 0, 0, cam], {name:'x'});
var ey = board.create('point3D', [0, 5, 0, cam], {name:'y'});
var ez = board.create('point3D', [0, 0, 5, cam], {name:'z'});

var a = board.create('point3D', [-4, 1, 1, cam], {name: 'a'});
var b = board.create('point3D', [6, 3, 1, cam],  {name: 'b'});
var c = board.create('point3D', [1, 8, 1, cam],  {name: 'c'});
var d = board.create('point3D', [2, 6, 4, cam],  {name: 'd'});

// Axes:
var attr1 = {
        strokeWidth: 0.5,
        strokeColor: 'black'
    };

var ax1 = board.create('line', [o, ex], attr1);
attr1.strokeColor = 'orange';
var ax2 = board.create('line', [o, ey], attr1);
attr1.strokeColor = 'red';
var ax3 = board.create('line', [o, ez], attr1);
```

```
// Tetrahedron:
var attr = {
        strokeWidth: 0.8
    };
var l1 = board.create('segment', [a, b], attr);
var l2 = board.create('segment', [a, c], attr);
var l3 = board.create('segment', [a, d], attr);
var l4 = board.create('segment', [b, c], attr);
var l5 = board.create('segment', [b, d], attr);
var l6 = board.create('segment', [c, d], attr);
```

Upcoming events

Next webinar

> Summer break! We will resume – if there is interest - in October at the JSXGraph conference.

2nd international JSXGraph conference

> The 2nd international JSXGraph conference will take place online **October 5th - 7th, 2021**.
> Free registration at https://jsxgraph.org/conf2021. Show your projects!