Workshop II – 3D API

3. International JSXGraph Conference

Alfred Wassermann

UNIVERSITÄT BAYREUTH

05-10-2022

Contents

1 Introduction

In this workshop we introduce the new – and quite preliminary – 3D API in JSXGraph.

1.1 Include JSXGraph

```
<!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/distrib/jsxgraph.css" rel="
        stylesheet" type="text/css" />
    <script src="https://cdn.jsdelivr.net/npm/jsxgraph/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>
```

2 3D Programming - Overview

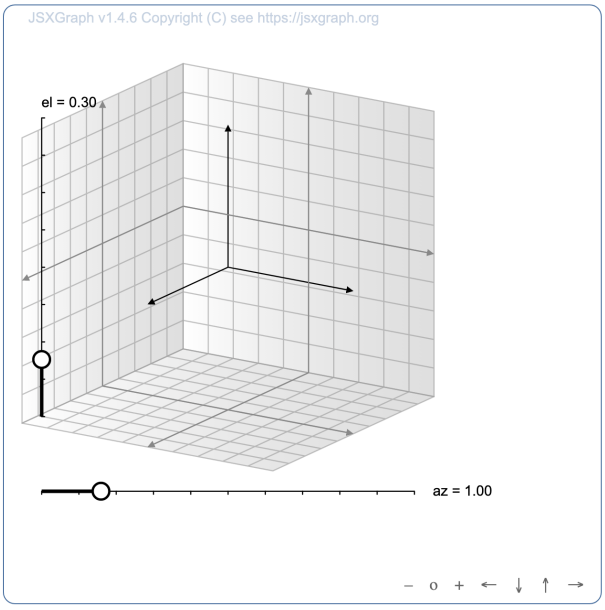
> Thanks to Juha-Matti Huusko. His talk at the JSXGraph conference in 2021 inspired the 3D support.

Some words of caution:

- As of v1.4.6: preliminary support for 3D in JSXGraph, the API may still change
- JSXGraph's main graphics engine is SVG. Thus, 3D support can not compete with OpenGL / WebGL
- Many attributes, options have to be added
- So far, only parallel projection, perspective projection will come
- Handling will also be improved
- Already usable in calculus and analysis, less so in geometry

3 3D view

3.1 Default 3D view

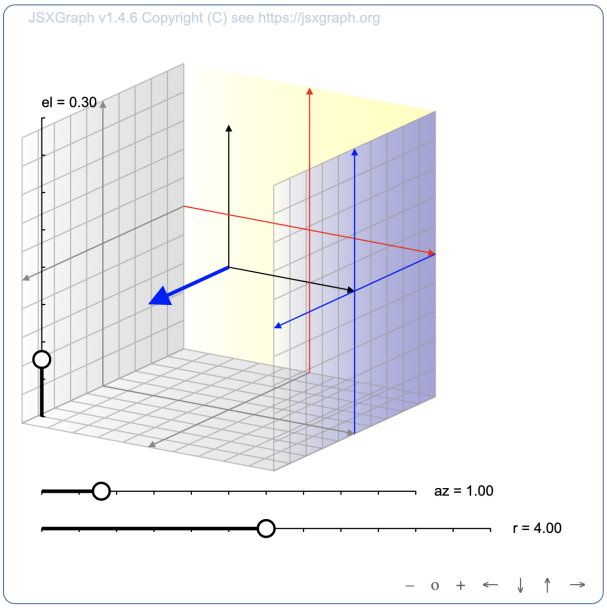


- A 3D view is displayed in a JSXGraph board
- A board may contain multiple 3D views
- https://jsfiddle.net/zsjtnedw/
- Sliders for azimuth and elevation are displayed by default

```
<div id="jxgbox" class="jxgbox construct"></div>
<script>
```

```
    const board = JXG.JSXGraph.initBoard('jxgbox', {
        boundingbox: [-8, 8, 8, -8],
        keepaspectratio: false,
        axis: false
    });
    var bound = [-5, 5];
    var view = board.create('view3d',
        [[-6, -3], [8, 8],
        [bound, bound, bound]],
        {});
</script>
```

3.2 Customize the 3D view



- Naming convention:
  - e.g. `xPlaneRear` is the plane orthogonal to the direction `x` at position `x = bound[0]`,
  - `yPlaneFront` is the plane orthogonal to the direction `y` at position `y = bound[1]`
- https://jsfiddle.net/1tj54db6/

```
var bound = [-5, 5];
var view = board.create('view3d',
    [[-6, -3], [8, 8],
    [bound, bound, bound]],
    {
        xAxis: { strokeColor: 'blue', strokeWidth: 3},

        // Planes
        xPlaneRear: { fillColor: 'yellow',  mesh3d: {visible: false}},
        yPlaneFront: { visible: true, fillColor: 'blue'},
```

```
        // Axes on planes
        xPlaneRearYAxis: {strokeColor: 'red'},
        xPlaneRearZAxis: {strokeColor: 'red'},

        yPlaneFrontXAxis: {strokeColor: 'blue'},
        yPlaneFrontZAxis: {strokeColor: 'blue'},

        zPlaneFrontXAxis: {visible: false},
        zPlaneFrontYAxis: {visible: false}
 });
var r = board.create('slider', [[-7, -6], [5, -6], [1, 4, 7]], { name: 'r' });
```

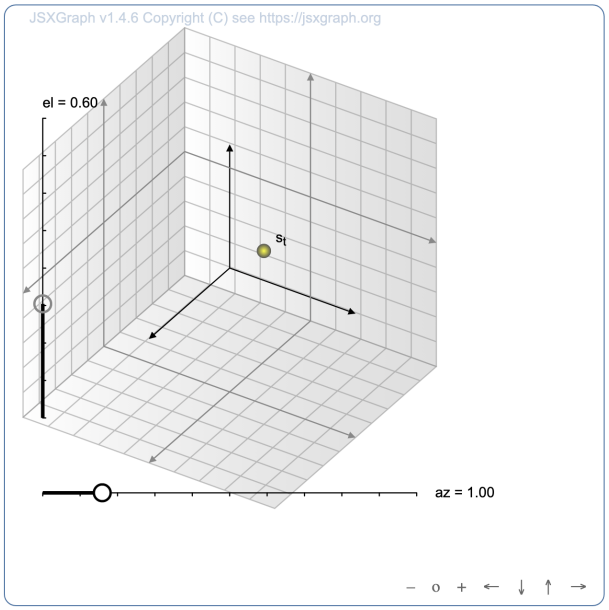
3.3 3D elements

- All element creators end with `'3d'`, e.g. `'point3d'`, `'line3d'`
- So far we have:
  - `point3d`
  - `line3d`
  - `plane3d`
  - `parametricsurface3d`
  - `functiongraph3d`
  - `view3d`
  - `axis3d`
- Instead of `board.create()` we use `view.create()` to create 3D elements.
- Also 2D elements can be created via `view.create()`, e.g. if 2D elements are connecting 3D points.

All 3D elements in JSXGraph possess a subobject `element2D` which contains the projection of the 3D object to the 2D board.

4 3D Points

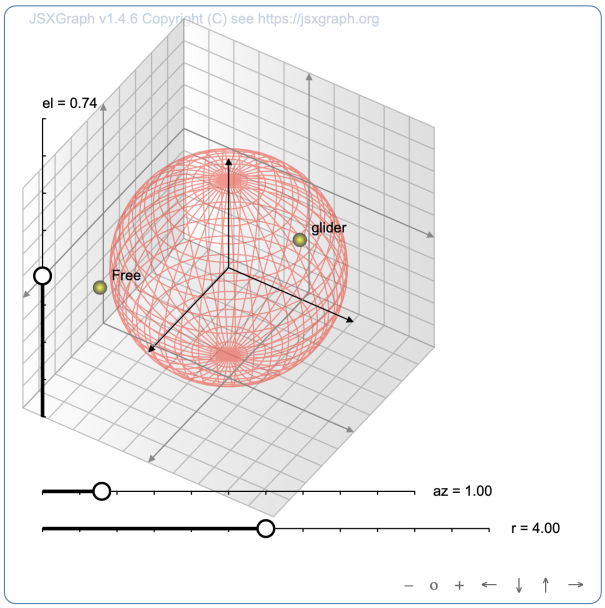
4.1 Free point



- Free 3D points can be dragged in their initial z plane
- More freedom to dragging is planned
- See https://jsfiddle.net/khm01ctj/1/

```
var bound = [-5, 5];
var view = board.create('view3d',
    [[-6, -3], [8, 8],
    [bound, bound, bound]],
    {});

// 3D point
var p = view.create('point3d', [1, 2, 2], { size: 5, name: 'A' });
```

4.2 Glider



- *Gliders* are bound to a 3D surface (curved or plane), see point `q` below
- Name is still `point3d`, may change to `glider3d`
- See https://jsfiddle.net/kqsytdh4/3/

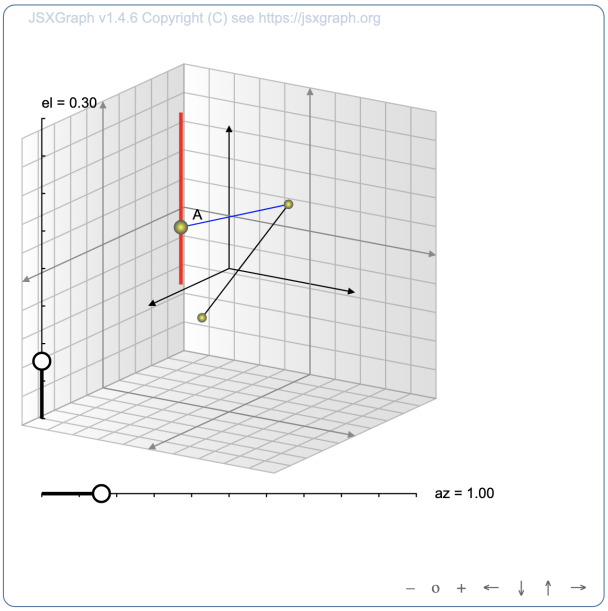
```
var r = board.create('slider', [[-7, -6], [5, -6], [1, 4, 7]], { name: 'r' });

// Free 3D point
var p = view.create('point3d', [1, 2, 2], { name: 'Free', size: 5 });

// Sphere
var sphere = [
    (u, v) => r.Value() * Math.sin(u) * Math.cos(v),
    (u, v) => r.Value() * Math.sin(u) * Math.sin(v),
    (u, v) => r.Value() * Math.cos(u)
];
var c = view.create('parametricsurface3d', sphere.concat(
    [
        [0, 2 * Math.PI],
        [0, 2 * Math.PI]
    ]), { strokeColor: '#ff0000', strokeOpacity: 0.5, stepsU: 40, stepsV: 40 });

// 3D glider
var u = Math.PI * 0.4,
    v = Math.PI * 0.3;

var q = view.create('point3d', [
        sphere[0](u, v), sphere[1](u, v), sphere[2](u, v),  // Initial coordinates
        c                                                   // Object, the point is
             bound to
    ],
    {
```

```
      name: 'glider', size: 5
    });
```

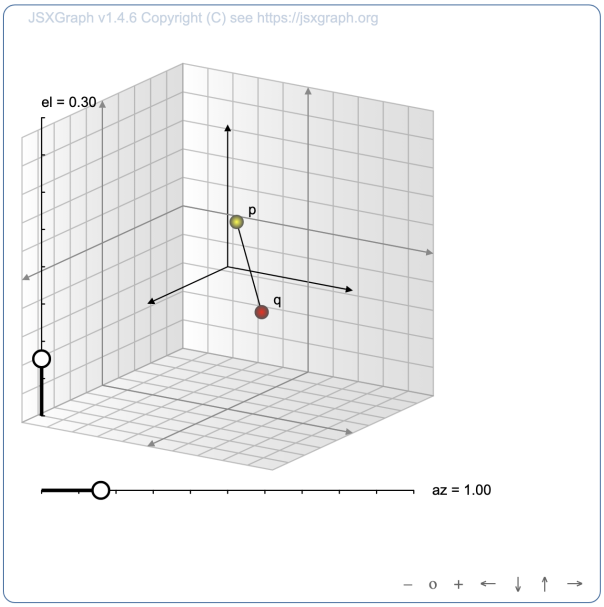
5 3D lines and curves

5.1 Lines

- Two variants:
  - Segment defined by two points, see `l1` and `l2` below
  - Line defined by point / direction and range, see `l3?` below
- See https://jsfiddle.net/dwfp5tnx/
- 3D lines have a subobjects `point1` and `point2`




```
// Lines through 2 points
var l1 = view.create('line3d', [[1, 3, 3], [-3, -3, -3]], {
    point: { visible: true }, point2: { visible: true }
});
var p = view.create('point3d', [1, 2, 2], { name: 'A', size: 5 });
var l2 = view.create('line3d', [p, l1.point1], { strokeColor: 'blue' });

// Line by point A, direction, range
var l3 = view.create('line3d', [p, [0, 0, 1], [-2, 4]], {
    strokeColor: 'red', strokeWidth: 3,
    point1: { visible: false }, point2: { visible: false }
});
```



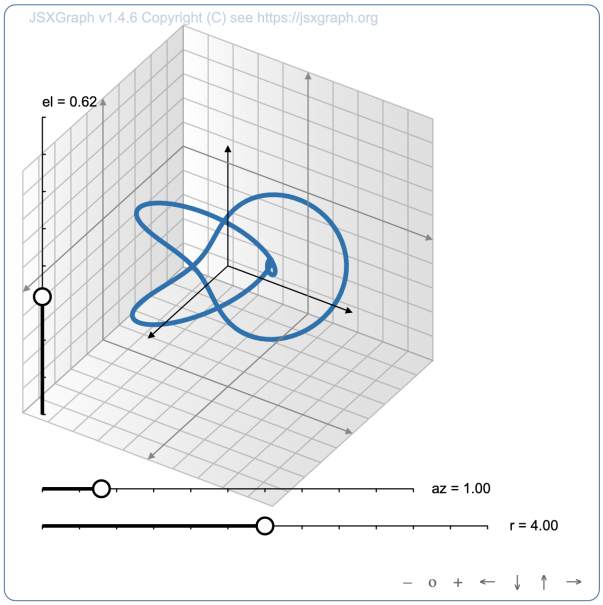
- 3D points may depend on other elements, see point `q` below
- See https://jsfiddle.net/5hs6pemr/
- For now, 3D lines defined by two points are 3D segments. Infinite 3D lines through two points have to be implemented.

```
var p = view.create('point3d', [1, 1, 2], { size: 5, name: 'p' });
// Dependent point:
var q = view.create('point3d', [
  () => [p.X(), p.Y() + 1, p.Z() - 3]
  ], {
    name: 'q', fillColor: 'red', size: 5
  });
// Line defined by two points
var l2 = view.create('line3d', [p, q], { fixed: false });
```

5.2 Curves

A *parametric curve* in 3D is a map

$$\mathbb{R} \to \mathbb{R}^3.$$



- Displays the graph of a parametric curve $\mathbb{R} \to \mathbb{R}^3$
- Supply 3 functions returning a number
- See https://jsfiddle.net/7dj805ba/

```
var r = board.create('slider', [[-7, -6], [5, -6], [1, 4, 7]], { name: 'r' });

var curve = view.create('curve3d', [
        (t) => r.Value() / 3 * (2 + Math.cos(3 * t)) * Math.cos(2 * t),
        (t) => r.Value() / 3 * (2 + Math.cos(3 * t)) * Math.sin(2 * t),
        (t) => r.Value() / 3 * Math.sin(3 * t),
        [-Math.PI, Math.PI]
    ], { strokeWidth: 4 });
```

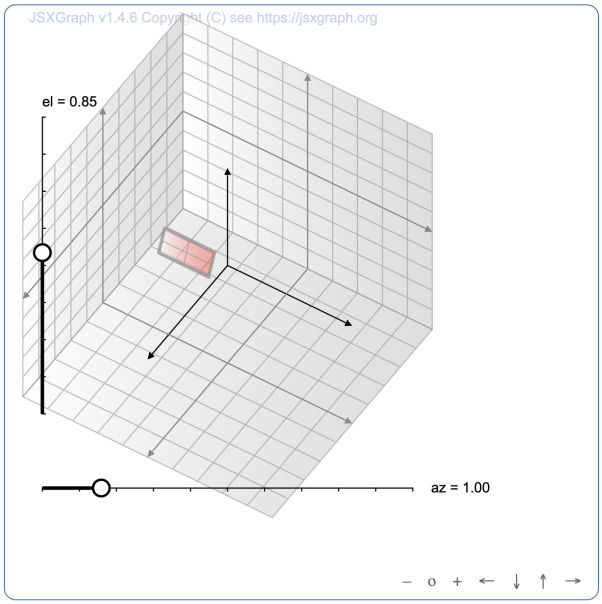
- Alternatively, supply 1 function returning an array of 3 numbers
- See https://jsfiddle.net/z2gbaews/

```
var r = board.create('slider', [[-7, -6], [5, -6], [1, 4, 7]], { name: 'r' });

var curve = view.create('curve3d', [
    (t) => [Math.cos(t), r.Value() * Math.sin(t), 0.5 * t],
        [-Math.PI, Math.PI]
    ], { strokeWidth: 4 });
```

5.3 Planes

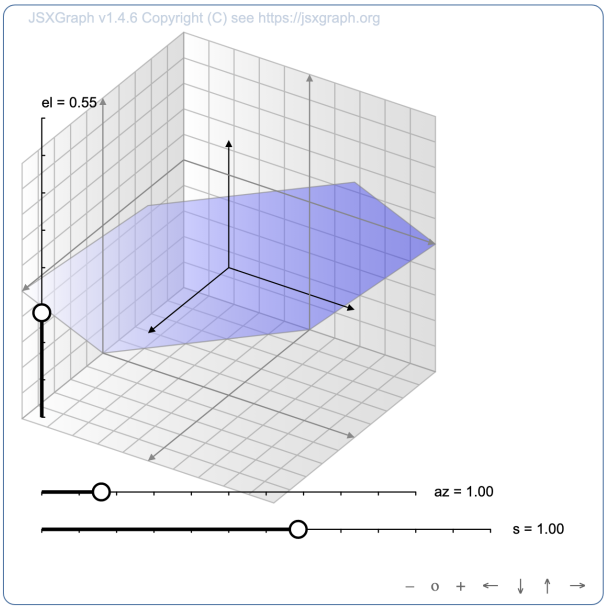
- There are two variants: finite facets and infinite planes.
- First, a finite **facet**, e.g. a rectangle
- See https://jsfiddle.net/y7czf3hL/

- Cut off at bounding cube still to be implemented



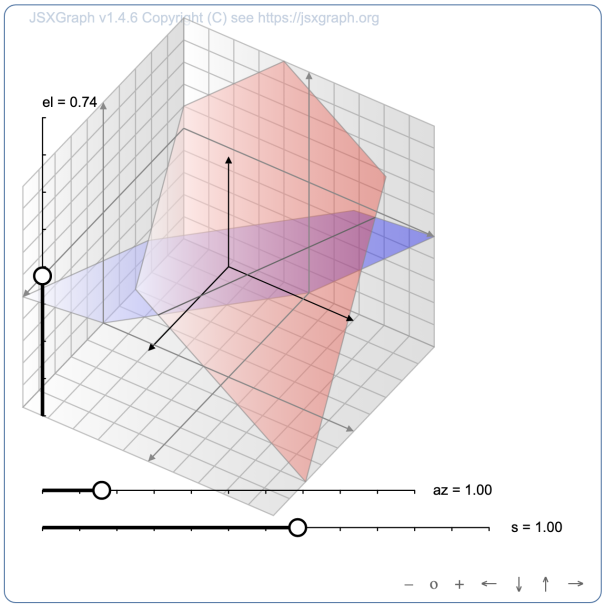

```
var v11 = [0, 1, 0],
    v12 = [2, 1, 1];

var rectangle = view.create('plane3d', [[1, -1, 1], v11, v12, [-1, 1], [-1, 1]], {
    fillColor: 'red', strokeWidth: 1, strokeColor: '#888888',
    strokeOpacity: 0.6,
    mesh3d: { visible: true }
});
```

- Second, an infinite plane, cut off at the bounding cube
- (Infinite) planes – affine flats – are defined in the style of *analytic geometry*: 1 point, 2 directions and 2 ranges
- See https://jsfiddle.net/p4vcnfbg/1/




```
var s = board.create('slider', [[-7, -6], [5, -6], [-3, 1, 4]], { name: 's' });

var v1 = [1, 0, () => s.Value()],
    v2 = [2, -1, () => s.Value()];

var plane = view.create('plane3d', [
        [2, -1, 1],             // Point
        v1, v2,                 // Two directions
        [-Infinity, Infinity],  // Range 1
        [-Infinity, Infinity]   // Range 2
    ], {
    fillColor: 'blue', strokeWidth: 1, strokeColor: '#888888',
    strokeOpacity: 0.6,
    mesh3d: { visible: false } // not yet
});
```

5.4 Intersection of planes



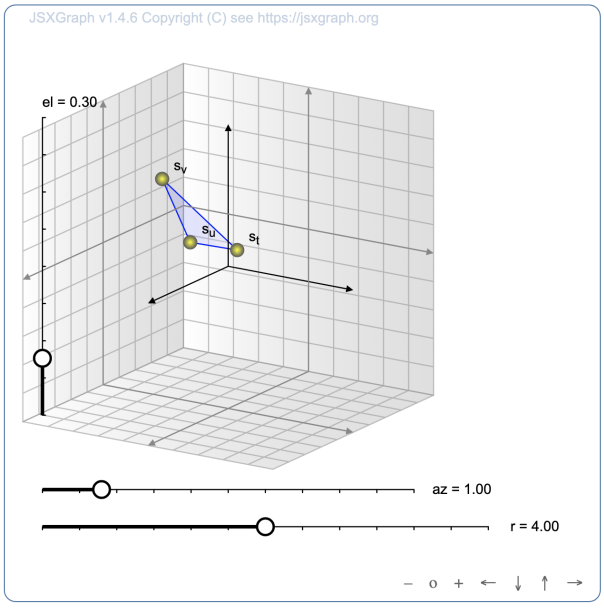
- See https://jsfiddle.net/8vat4hqn/

```
var v1 = [1, 0, () => s.Value()],
    v2 = [2, -1, () => s.Value()];

var plane1 = view.create('plane3d', [
        [2, -1, 1],             // Point
        v1, v2,                 // Two directions
        [-Infinity, Infinity],  // Range 1
        [-Infinity, Infinity]   // Range 2
    ], {
        fillColor: 'blue', strokeWidth: 1, strokeColor: '#888888',
        strokeOpacity: 0.6,
    });

 var plane2 = view.create('plane3d', [
        [-1, 2, 0],             // Point
        [1, 2, 0], [0, 1, 1],   // Two directions
        [-Infinity, Infinity],  // Range 1
        [-Infinity, Infinity]   // Range 2
    ], {
        fillColor: 'red', strokeWidth: 1, strokeColor: '#888888',
        strokeOpacity: 0.6
    });

var p1, p2, li;
p1 = view.create('point3d', [() => view.intersectionPlanePlane(plane1, plane2)[0]],
     { visible: false });
p2 = view.create('point3d', [() => view.intersectionPlanePlane(plane1, plane2)[1]],
     { visible: false });
li = view.create('line3d', [p1, p2], {strokeColor: '#666666'});
```

5.5 2D polygons in 3D space

- 3D finite facets can be handled by 2D polygons (so far: must be handled)
- After all, a projective transformation maps lines to lines
- Use the subobject `element2D` of a 3D point to access the 2D point

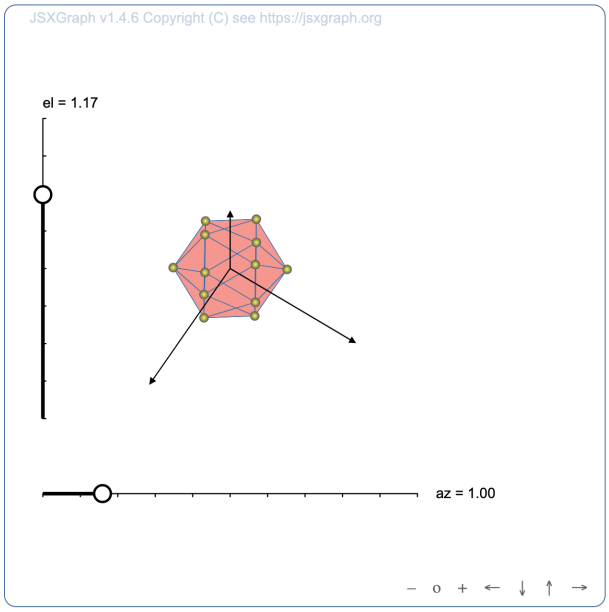
5.5.1 A triangle



- The triangle below is just a 2D polygon using 2D points
- See https://jsfiddle.net/pyc41nwv/

```
var view = board.create('view3d',
    [[-6, -3], [8, 8],              // 2D box of view
    [[-5, 5], [-5, 5], [-5, 5]]],  // 3D bounding cube
    {});

var A = view.create('point3d', [1, 1, 1], { size: 5 });  // Point A
var B = view.create('point3d', [3, 1, 2], { size: 5 });  // Point B
var C = view.create('point3d', [1, -2, 3], { size: 5 });  // Point C

view.create('polygon', [A.element2D, B.element2D, C.element2D], {
    highlight: false,
    gradient: 'linear',
    gradientSecondColor: 'white',
    fillColor: 'blue',
    borders: { strokecolor: 'blue', strokeWidth: 1 }
});
```

5.5.2 Solids



- E.g. an icosahedron
- See https://jsfiddle.net/my04azxv/

```
var view = board.create('view3d',
    [[-6, -3], [8, 8],
    [[-5, 5], [-5, 5], [-5, 5]]],
    {
        xPlaneRear: {visible: false},
        yPlaneRear: {visible: false},
        zPlaneRear: {visible: false}
    });

var point_attr = { fixed: true, withLabel: false, label: { offset: [5, 5] } },
    i, j,
    phi = (1 + Math.sqrt(5)) * 0.5,
    pol_attr = { borders: { strokeWidth: 0.5 }, fillColor: 'red' },
    q = [],
    f = [];

// Icosahedron
let cnt = 0;
point_attr.withLabel = false;
for (i = -1; i < 2; i += 2) {
    for (j = -1; j < 2; j += 2) {
        point_attr.name = cnt++;
        q.push(view.create('point3d', [0, i, j * phi], point_attr).element2D);
        point_attr.name = cnt++;
        q.push(view.create('point3d', [i, j * phi, 0], point_attr).element2D);
        point_attr.name = cnt++;
        q.push(view.create('point3d', [j * phi, 0, i], point_attr).element2D);
    }
```

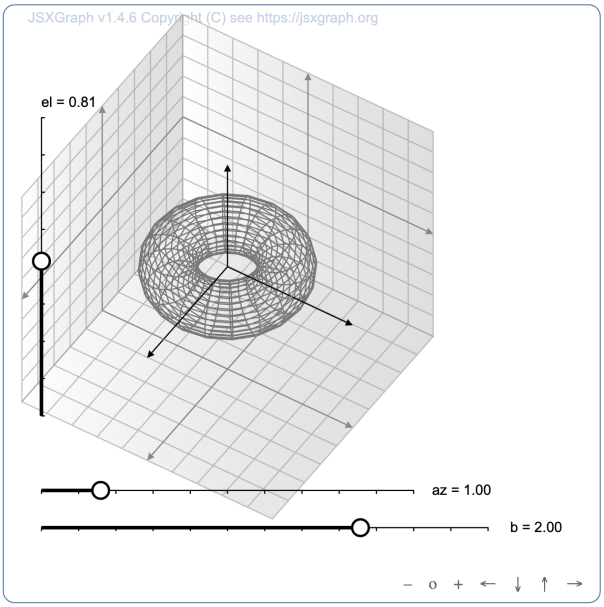
```
}
f.push(board.create('polygon', [q[0], q[1], q[2]], pol_attr));
f.push(board.create('polygon', [q[0], q[1], q[7]], pol_attr));
f.push(board.create('polygon', [q[1], q[2], q[8]], pol_attr));
f.push(board.create('polygon', [q[0], q[5], q[7]], pol_attr));
f.push(board.create('polygon', [q[5], q[7], q[11]], pol_attr));
f.push(board.create('polygon', [q[1], q[7], q[3]], pol_attr));
f.push(board.create('polygon', [q[1], q[8], q[3]], pol_attr));
f.push(board.create('polygon', [q[3], q[7], q[11]], pol_attr));
f.push(board.create('polygon', [q[3], q[8], q[9]], pol_attr));
f.push(board.create('polygon', [q[2], q[8], q[4]], pol_attr));
f.push(board.create('polygon', [q[4], q[8], q[9]], pol_attr));
f.push(board.create('polygon', [q[0], q[2], q[6]], pol_attr));
f.push(board.create('polygon', [q[0], q[5], q[6]], pol_attr));
f.push(board.create('polygon', [q[5], q[10], q[11]], pol_attr));
f.push(board.create('polygon', [q[3], q[9], q[11]], pol_attr));
f.push(board.create('polygon', [q[9], q[10], q[11]], pol_attr));
f.push(board.create('polygon', [q[5], q[6], q[10]], pol_attr));
f.push(board.create('polygon', [q[4], q[9], q[10]], pol_attr));
f.push(board.create('polygon', [q[2], q[4], q[6]], pol_attr));
f.push(board.create('polygon', [q[4], q[6], q[10]], pol_attr));
```

6 Parametric surfaces

A *parametric surface* is a map

$$\mathbb{R}^2 \to \mathbb{R}^3$$

- Parameters are:
  - Three functions $\mathbb{R}^2 \to \mathbb{R},\ (u, v) \mapsto f(u, v)$
  - Ranges for u and v
- The attributes `stepsU: 30, stepsV: 20` define the mesh
- See https://jsfiddle.net/9kL23tmh/

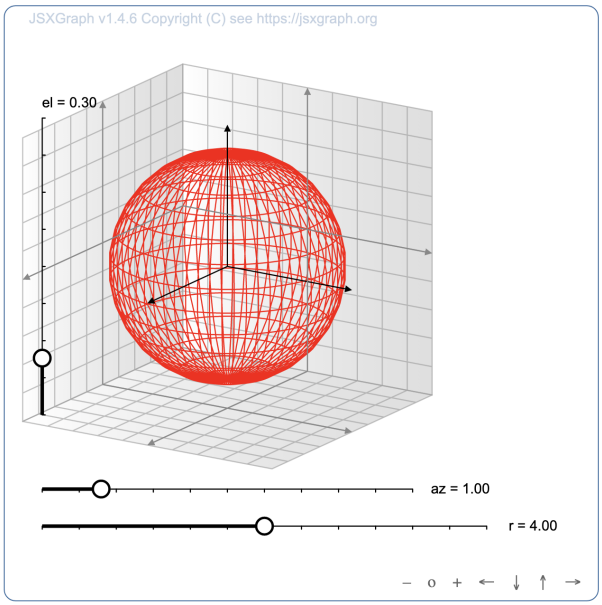



```
var b = board.create('slider', [[-7, -6], [5, -6], [-3, 2, 4]], { name: 'b' });

// Torus
var c = view.create('parametricsurface3d', [
    (u, v) => (Math.cos(u) + b.Value()) * Math.cos(v),
    (u, v) => (Math.cos(u) + b.Value()) * Math.sin(v),
    (u, v) => Math.sin(u),
    [0, 2 * Math.PI],
    [0, 2 * Math.PI]
], { strokeColor: '#777777', stepsU: 30, stepsV: 20, visible: true });
```

- Alternative parameters are one function

$$\mathbb{R}^2 \to \mathbb{R}^3,\ (u,v) \mapsto [f_x(u,v), f_y(u,v), f_z(u,v)]$$

  plus ranges
- Example: sphere as parametric surface
- See https://jsfiddle.net/96zpwmgx/1/



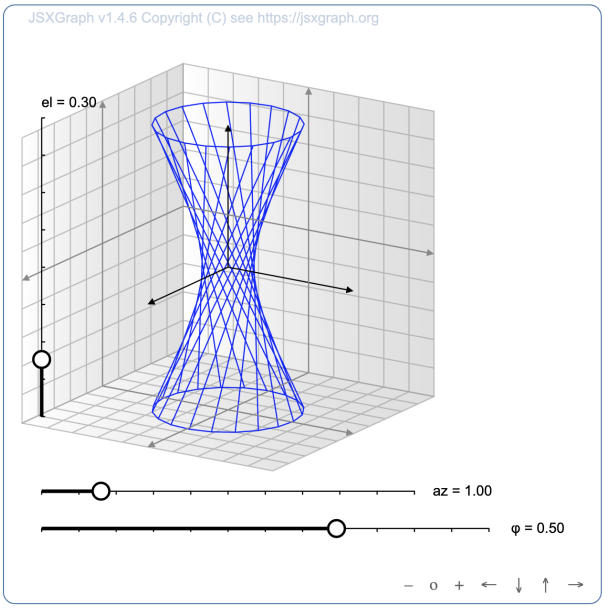

```
var r = board.create('slider', [[-7, -6], [5, -6], [1, 4, 7]], { name: 'r' });

// Sphere
var c = view.create('parametricsurface3d', [
    (u, v) => [
                r.Value() * Math.sin(u) * Math.cos(v),
                r.Value() * Math.sin(u) * Math.sin(v),
                r.Value() * Math.cos(u)
            ],
    [0, 2 * Math.PI],
    [0, Math.PI]
], { strokeColor: '#ff0000', stepsU: 30, stepsV: 30 });
```

6.1 Ruled surfaces

Given two maps $f, g : \mathbb{R}^2 \to \mathbb{R}^3$, a *ruled surface* is a linear combination

$$(u, v) \mapsto (1 - v) \cdot f(u, v) + v \cdot g(u, v).$$

Usually it is a convex combination of the functions f and g, i.e. $0 \leq v \leq 1$.



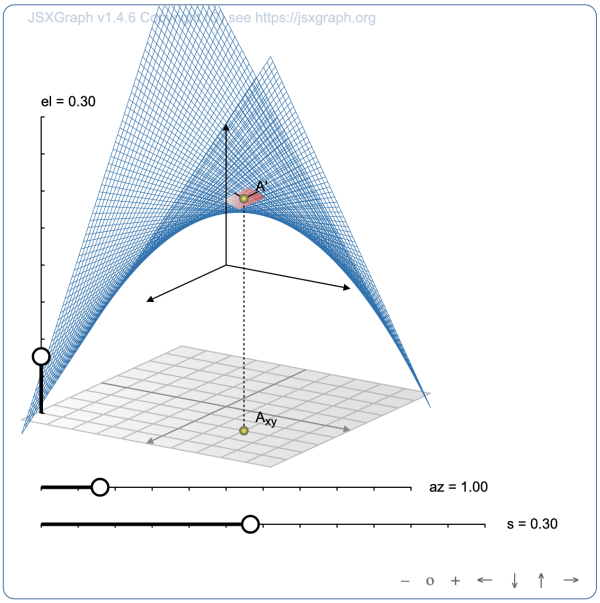
- Example: hyperboloid
- Just set `stepsV:1` in `parametricsurface3d`
- See https://jsfiddle.net/bgjkh47e//

```
var phi = board.create('slider', [[-7, -6], [5, -6], [-Math.PI / 2, 0.5, Math.PI /
    2]], {
    name: '&phi;', snapWidth: 0.1
});
var c = view.create('parametricsurface3d', [
    (u, v) => (1 - v) * Math.cos(u - phi.Value()) + v * Math.cos(u + phi.Value()),
    (u, v) => (1 - v) * Math.sin(u - phi.Value()) + v * Math.sin(u + phi.Value()),
    (u, v) => 2 * v - 1,
    [0, 2 * Math.PI],
    [-2, 3]
], { strokeColor: '#0000ff',
     strokeWidth: 1, strokeOpacity: 0.9,
     stepsU: 20, stepsV: 1
});
```

6.2 3D function graphs

A *3D function graph* is the visualization of a map of the form

$$\mathbb{R}^2 \to \mathbb{R}.$$



- See https://jsfiddle.net/yL4adbxe//

```
var box = [-5, 5];
var view = board.create('view3d',
    [
        [-6, -3], [8, 8],
        [box, box, box]
    ],
    {
        xPlaneRear: {visible: false},
        yPlaneRear: {visible: false},
    });

var s = board.create('slider', [[-7, -6], [5, -6], [-3, 0.3, 4]], { name: 's' });

// Function F to be plotted
var F = (x, y) => s.Value() * x * y + 2;
// 3D surface
var c = view.create('functiongraph3d', [
    F,
    box, // () => [-s.Value()*5, s.Value() * 5],
    box, // () => [-s.Value()*5, s.Value() * 5],
], { strokeWidth: 0.5, stepsU: 70, stepsV: 70 });

// 3D points:
// Point on xy plane, i.e. zPlaneRear
var Axy = view.create('point3d', [2, 2, -5], { name: 'A_{xy}' });

// Project Axy to the surface
var A = view.create('point3d', [
    () => [Axy.X(), Axy.Y(), F(Axy.X(), Axy.Y())]
], { name: "A'" });
view.create('line3d', [Axy, A], { dash: 1 })
```

```
// Partial derivatives of F in point A
// "supplied manually"
var dFx = () => s.Value() * A.Y(),
    dFy = () => s.Value() * A.X(),
    dFx_vec = [1, 0, dFx],
    dFy_vec = [0, 1, dFy];

// Gradient plane
var plane1 = view.create('plane3d', [
    A,
    dFx_vec, dFy_vec,
    [() => -dFx(), dFx], [() => -dFy(), dFy]
], {
    fillOpacity: 0.8, fillColor: 'red'
});
var a = view.create('line3d', [A, dFx_vec, [0, dFx]]);
var b = view.create('line3d', [A, dFy_vec, [0, dFy]]);
```

7 Final words

- Some more examples: https://jsxgraph.uni-bayreuth.de/~alfred/jsxdev/3D/demo3d.html
- API docs: https://jsxgraph.org/docs

Missing are

- Many options, methods, special cases
- Complete documentation
- Facets, hidden surface removal
- Axes in matlab style
- Various intersection methods
- …

Please send us your suggestions, bug reports, or even patches!