Automatic calculation of plane loci using Groebner bases and integration into a Dynamic Geometry System

Michael Gerhäuser, Alfred Wassermann

July 24, 2010

UNIVERSITÄT BAYREUTH

Overview

JSXGraph - A short overview

Computing plane loci using Groebner bases

Implementing this algorithm in JSXGraph

Optimizations

Examples

JSXGraph

JSXGraph

What is JSXGraph?

- A library implemented in JavaScript
- Runs in recent versions of all major browsers
- No plugins required
- LGPL-Licensed

Main features

- Dynamic Geometry
- Interactive function plotting
- Turtle Graphics
- Charts

JSXGraph

Supported Hardware

- PC (Windows, Linux, Mac)
- Mobile phones
- "Touchpads" like the Apple iPod and iPad
- Basically everything which runs at least one of the supported browsers

JSXGraph

Supported Browsers

- Firefox
- Chrome/Chromium
- Safari
- Internet Explorer
- Opera

JSXGraph

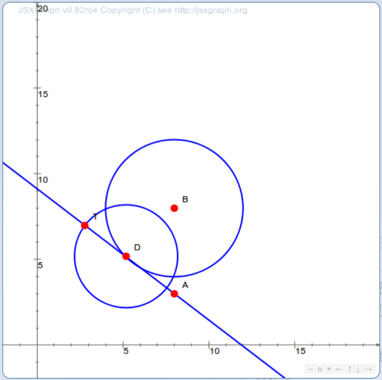
Example/Input

```
<link rel="stylesheet" type="text/css" href="css/jsxgraph.css" />
<script type="text/javascript" src="js/jsxgraphcore.js"></script>
...
<div id="jxgbox" class="jxgbox" style="width:500px; height:500px;"></div>
<script type="text/javascript">
  board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox: [-2, 20, 20, -2], axis:
        true, grid: false, keepaspectratio: true});
  p3 = board.create('point', [8, 3]);
  p4 = board.create('point', [8, 8]);
  c1 = board.create('circle', [p4, 4]);
  p6 = board.create('glider', [0, 0, c1], {name: 'D'});
  g = board.create('line', [p3, p6]);
  c2 = board.create('circle', [p6, 3]);
  p14_1 = board.create('intersection', [c2,g,0], {name: 'T'});
</script>
```

UNIVERSITÄT BAYREUTH

JSXGraph

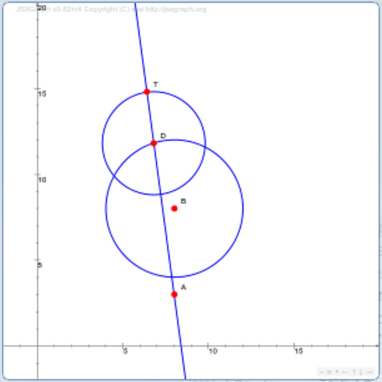
Example/Output



Computing plane loci using Groebner bases (in a nutshell)

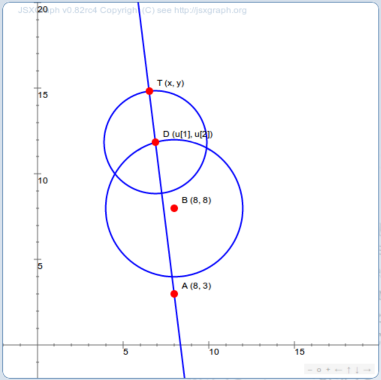
Computing plane loci using Groebner bases

- Given a set of free and dependent points,



Computing plane loci using Groebner bases

- we first choose a coordinate system,



Computing plane loci using Groebner bases

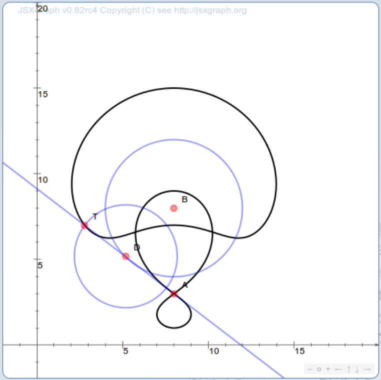
- translate geometric constraints into an algebraic form,
  - $(u[1]-8)^2+(u[2]-8)^2-16=0$
  - $(x-u[1])^2+(y-u[2])^2-9=0$
  - $3x-3u[1]+yu[1]-8y+8u[2]-xu[2]=0$

Computing plane loci using Groebner bases

- calculate the Gröbner basis of the given ideal,
  - $x^6+3x^4y^2+3x^2y^4+y^6-48x^5-38x^4y-96x^3y^2-76x^2y^3-48xy^4-38y^5+1047x^4+1216x^3y+1774x^2y^2+1216xy^3+727y^4-13024x^3-16596x^2y-16096xy^2-8404y^3+97395x^2+109888xy+63535y^2-415536x-300806y+790009=0$

Computing plane loci using Groebner bases

- and finally plot the calculated implicit equation.



Implementing this algorithm in JSXGraph

Implementation

Problems

- No JavaScript implementation of any Gröbner basis algorithm
- Can't use C-libraries directly in JavaScript
- No implicit plotting in JSXGraph by now

Implementation

AJAX

- Transfer data (a)synchronously via HTTP with JavaScript

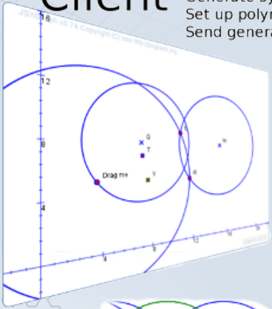
This enables us to

- use a computer algebra system on a (web) server for the expensive Gröbner basis calculations
- use a plotting tool/library for implicit plotting

Implementation

Client

Generate symbolic coordinates for free and dependent points.
Set up polynomials for the dependent ones describing their loci.
Send generated data to webserver for further calculations.



AJAX

Server

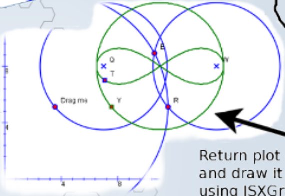
Several software packages are used server side:

python is used to retrieve the data and pass it on to

CoCoA handles the symbolic algebra stuff and returns a set of polynomials which are plotted with the python library

matplotlib

Finally the locus curve is extracted as a list of coordinates from the plots and is sent back to JSXGraph where the data is used to plot the locus directly in the geometric construction.



Return plot data and draw it using JSXGraph.

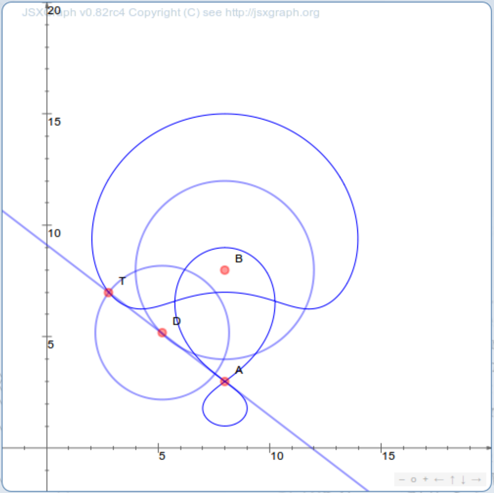
Implementation

Example/Input

```
<link rel="stylesheet" type="text/css" href="css/jsxgraph.css" />
<script type="text/javascript" src="js/jsxgraphcore.js"></script>
...
<div id="jxgbox" class="jxgbox" style="width:500px; height:500px;"></div>
<script type="text/javascript">
  board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox: [-2, 20, 20, -2], axis:
        true, grid: false, keepaspectratio: true});
  p3 = board.create('point', [8, 3]);
  p4 = board.create('point', [8, 8]);
  c1 = board.create('circle', [p4, 4]);
  p6 = board.create('glider', [0, 0, c1], {name: 'D'});
  g = board.create('line', [p3, p6]);
  c2 = board.create('circle', [p6, 3]);
  p14_1 = board.create('intersection', [c2,g,0], {name: 'T'});

  locus = board.create('locus', [p14_1]);
</script>
```

Implementation

Example/Output



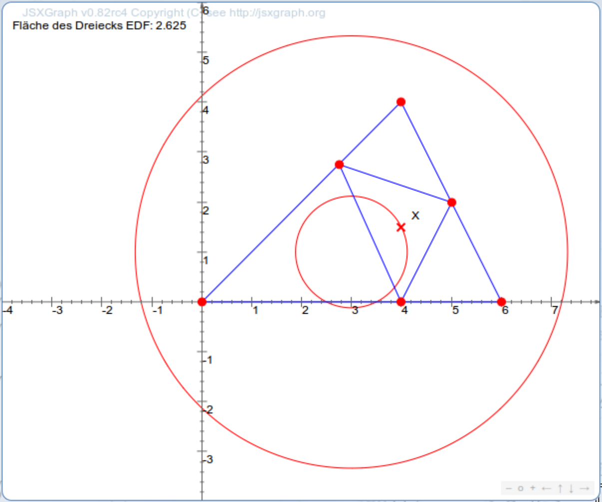
Implementation

Ready-to-use elements

- Glider on circle and line
- Intersection points (circle/circle, circle/line, line/line)
- Midpoint
- Parallel line and point
- Perpendicular line and point
- Circumcircle and circumcenter

Implementation

Easy to extend



Implementation

```
<link rel="stylesheet" type="text/css" href="css/jsxgraph.css" />
<script type="text/javascript" src="js/jsxgraphcore.js"></script>
...
<div id="jxgbox" class="jxgbox" style="width:500px; height:500px;"></div>
<script type="text/javascript">
  board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox:[-4, 6, 8, -4], axis:
        true, grid: false, keepaspectratio: true});
  A = board.create('point', [0, 0]);
  B = board.create('point', [6, 0]);
  C = board.create('point', [4, 4]);

  t1 = board.create('triangle', [A, B, C], {strokeWidth: '1px'});

  X = board.create('point', [4, 1.5], {name:"X"});

  L = board.create('perpendicularpoint', [X, t1.c]);
  M = board.create('perpendicularpoint', [X, t1.a]);
  N = board.create('perpendicularpoint', [X, t1.b]);

  t2 = board.create('triangle', [L, M, N], {strokeWidth: '1px'});
```

Implementation

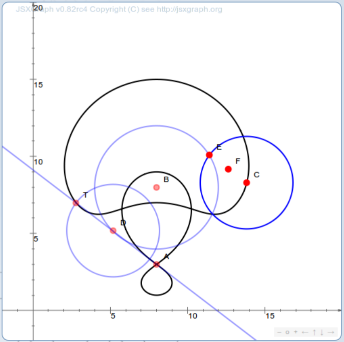
```
  ...

  X.ancestors[L.id] = L;
  X.ancestors[M.id] = M;
  X.ancestors[N.id] = N;
  X.ancestors[A.id] = A;
  X.ancestors[B.id] = B;
  X.ancestors[C.id] = C;

  X.generatePolynomial = function () {
    var as16 = getTriangleArea(L, M, N),
    as = '((('+M.symbolic.x+')-('+N.symbolic.x+'))^2+(('+M.symbolic.y+')-('+N.
         symbolic.y+'))^2)',
    bs = '((('+L.symbolic.x+')-('+N.symbolic.x+'))^2+(('+L.symbolic.y+')-('+N.
         symbolic.y+'))^2)',
    cs = '((('+M.symbolic.x+')-('+L.symbolic.x+'))^2+(('+M.symbolic.y+')-('+L.
         symbolic.y+'))^2)',

    return ['4*'+as+'*'+cs+'-('+as+'+'+cs+'-'+bs+')*('+as+'+'+cs+'-'+bs+')-('+
         as16+')'];
  };

  locus = board.create('locus', [X], {strokeColor: 'red'});
  /* ]]> */
</script>
```

Implementation

Re-using locus data: Discovered loci can be

- intersected with circles, lines, other curves, ...
- used as a base object for gliding points
- used for the discovery of other loci

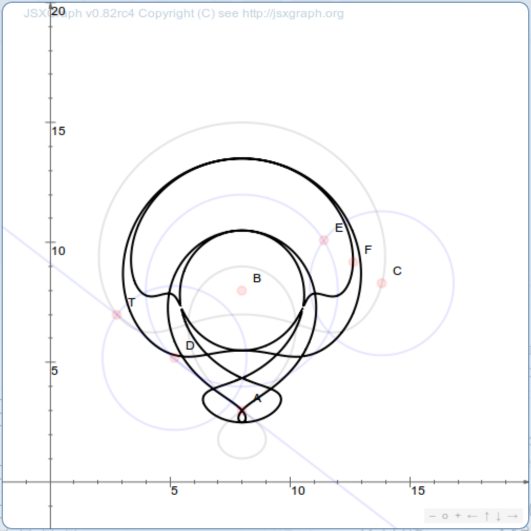
Implementation




```
C = board.create('glider', [loc]);

c2 = board.create('circle', [C, 3]);
E = board.create('intersection', [c1, c2, 0]);
F = board.create('midpoint', [C, E]);
```

UNIVERSITÄT BAYREUTH

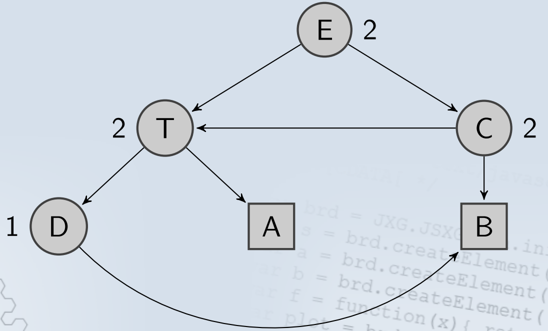
Implementation



Optimization

Optimization

Idea: Divide and conquer



Optimization

Transformations

- Translate the construction moving one point to $(0, 0)$
- Rotate the construction moving another point onto the x-axis
- After the Gröbner basis is calculated, the result is retransformed
- User can choose the two points or
- JSXGraph chooses two points (but sometimes not the best suited ones)

Examples

Last slide

Thank You

- http://jsxgraph.org/
- http://jsxgraph.uni-bayreuth.de/wiki/