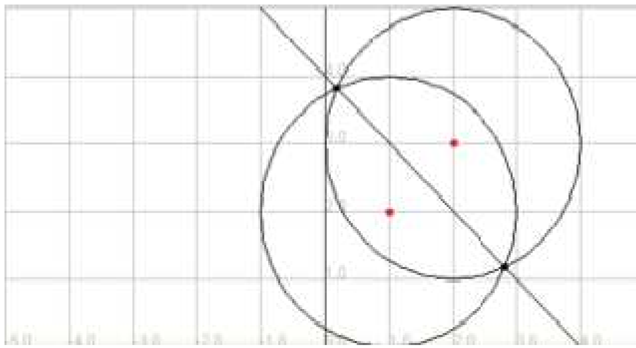
GAALOP Tutorial for Compass Ruler Algebra

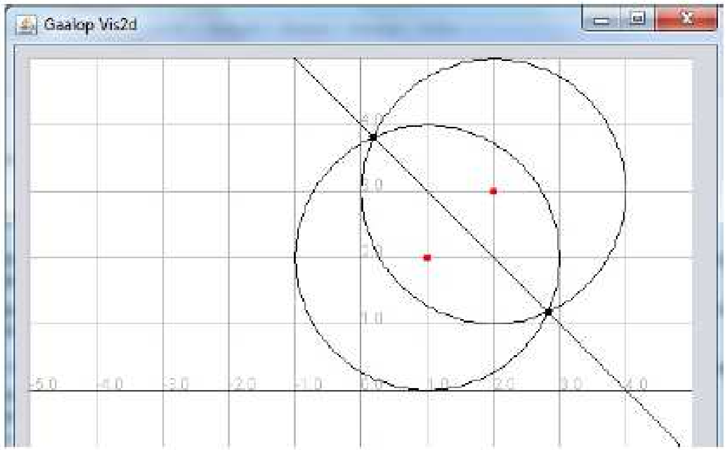
Dr.-Ing. Dietmar Hildenbrand

TU Darmstadt, Germany



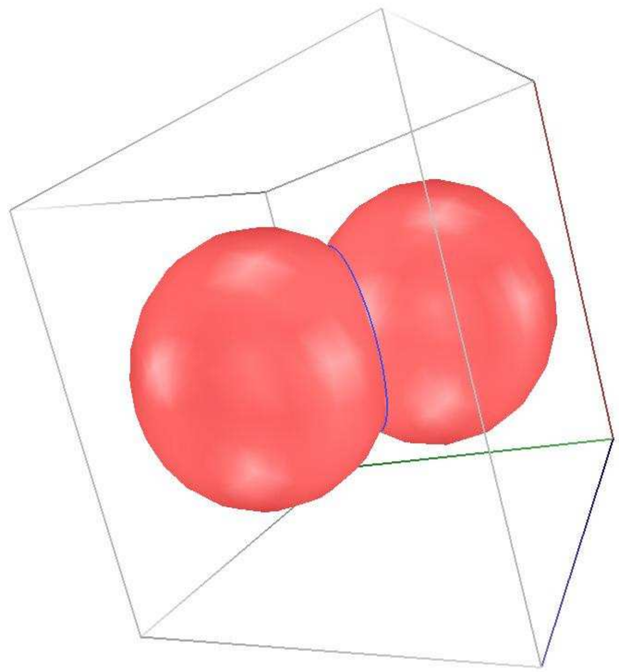
Overview

- Compass Ruler Algebra
- Visualizations with Gaalop



Goal of Geometric Algebra

- Mathematical language close to the geometric intuition combining geometry and algebra



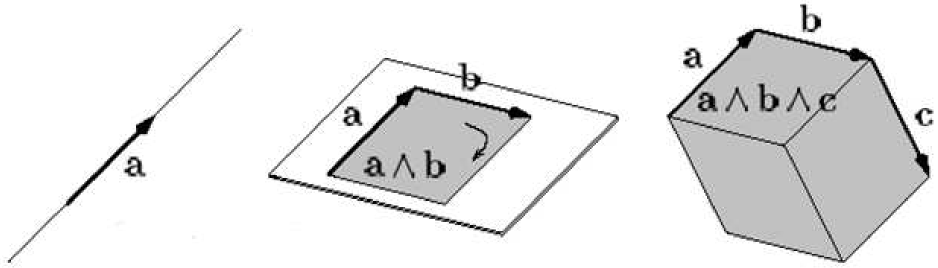
Compass Ruler Algebra

- 4 basis vectors:
  - e_1, e_2
  - e_0 : origin
  - e_∞ : point at infinity

Compass Ruler Algebra

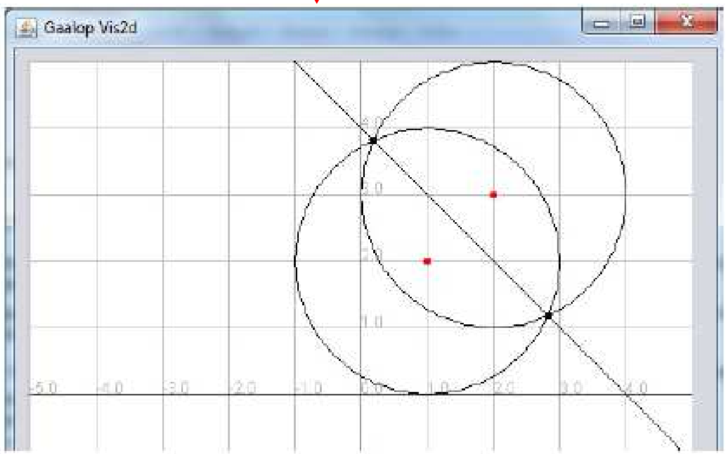
The 16 basis blades of the Compass Ruler Algebra.

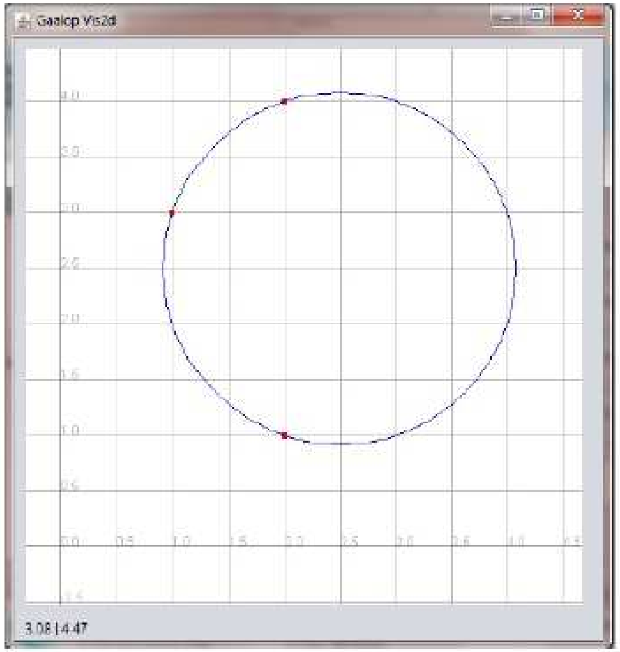
Index	Blade	Dimension
0	1	0
1	e_1	1
2	e_2	1
3	e_∞	1
4	e_0	1
5	$e_1 \wedge e_2$	2
6	$e_1 \wedge e_\infty$	2
7	$e_1 \wedge e_0$	2
8	$e_2 \wedge e_\infty$	2
9	$e_2 \wedge e_0$	2
10	$e_\infty \wedge e_0$	2
11	$e_1 \wedge e_2 \wedge e_\infty$	3
12	$e_1 \wedge e_2 \wedge e_0$	3
13	$e_1 \wedge e_\infty \wedge e_0$	3
14	$e_3 \wedge e_\infty \wedge e_0$	3
15	$e_1 \wedge e_2 \wedge e_\infty \wedge e_0$	4



Compass Ruler Algebra

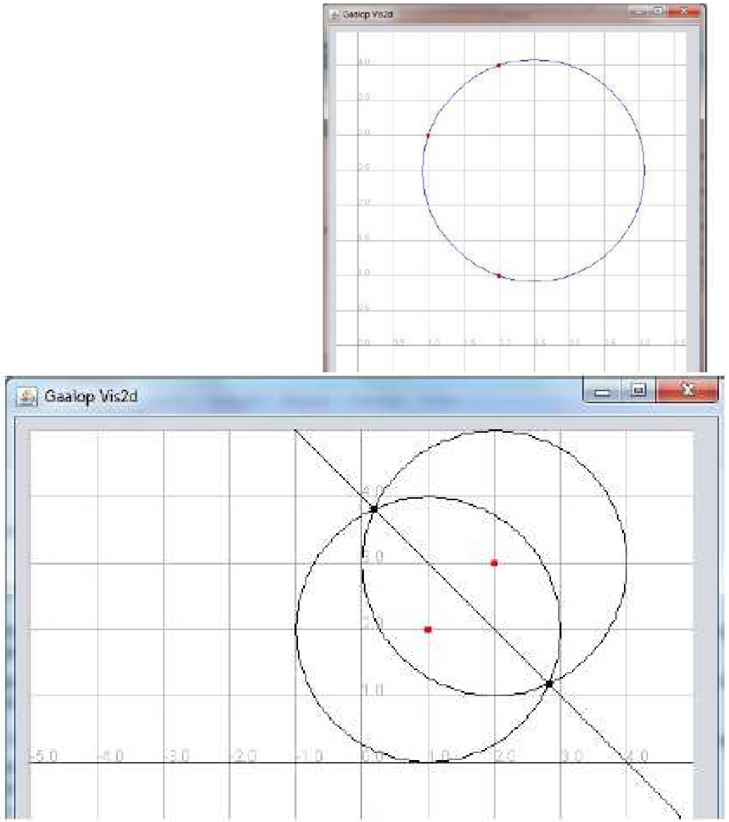
Entity	IPNS representation	OPNS representation
Point	$P = x_1e_1 + x_2e_2 + +\frac{1}{2}(x_1^2 + x_2^2)e_\infty + e_0$	
Circle	$C = P - \frac{1}{2}r^2e_\infty$	$C^* = P_1 \wedge P_2 \wedge P_3$
Line	$L = \mathbf{n} + de_\infty$	$L^* = P_1 \wedge P_2 \wedge e_\infty$
Point pair	$P_p = C_1 \wedge C_2$	$P_p^* = P_1 \wedge P_2$





Compass Ruler Algebra

Meaning of the products:



- Outer Product
    - Generation of geometric objects
    - Intersection
- Inner Product
    - Distance Point-Point
    - Distance Point-Line
    - Angle between Line-Line
    - Distance Point-Circle
    - …
- Geometric Product
    - Rotation
    - Translation
    - Reflection
    - Inversion (Ex. $P = Ce_\infty C$ center of a circle as the inversion of infinity)
    - …

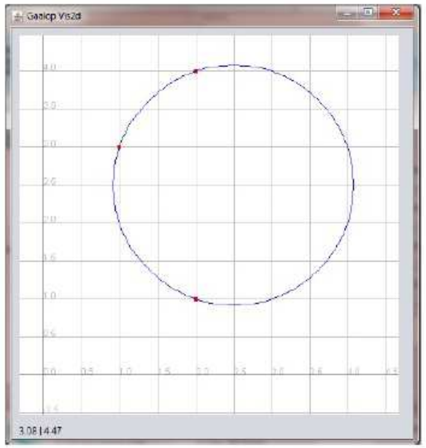
Gaalop


```
:Red;
:P1 = createPoint(2,1);
:P2 = createPoint(1,3);
:P3 = createPoint(2,4);
:Blue;
:C = *(P1^P2^P3);
```

Symbolic Optimization

Latex, C/C++ …

Visualization



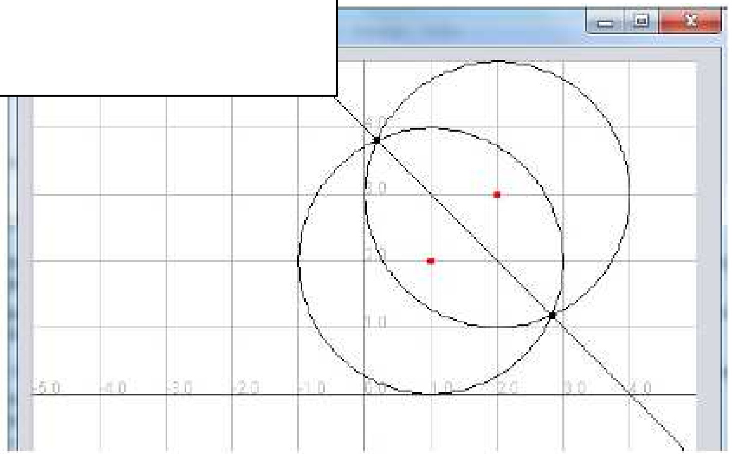
Gaalop


```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);

// intersect two circles with center points P1 and P2 with the same, but arbitrary radius
S1 = P1 - 0.5*r*r*einf;
S2 = P2 - 0.5*r*r*einf;
PP_dual = *(S1^S2);

// the line thru the two points of the resulting point pair
?Bisector = *(PP_dual^einf);
```

Entity	IPNS representation	OPNS representation
Point	$P = x_1e_1 + x_2e_2 + +\frac{1}{2}(x_1^2 + x_2^2)e_\infty + e_0$	
Circle	$C = P - \frac{1}{2}r^2e_\infty$	$C^* = P_1 \wedge P_2 \wedge P_3$
Line	$L = \mathbf{n} + de_\infty$	$L^* = P_1 \wedge P_2 \wedge e_\infty$
Point pair	$P_p = C_1 \wedge C_2$	$P_p^* = P_1 \wedge P_2$



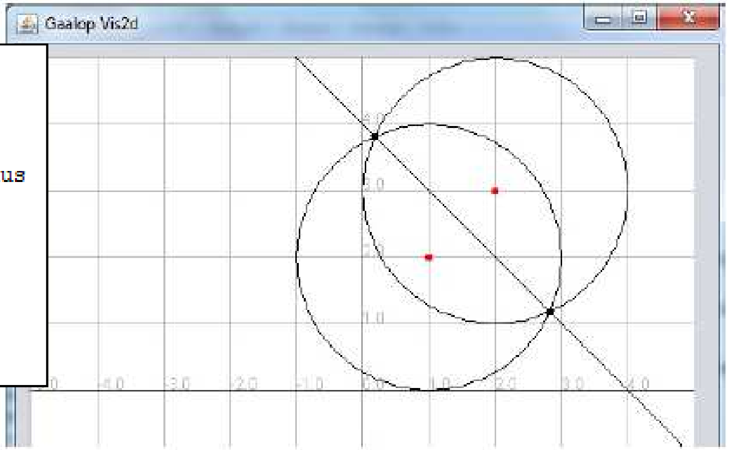
Proofs with Gaalop

Proof, that the perpendicular bisector is equal to the difference of the two points

```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);

// intersect two circles with center points P1 and P2 with the same, but arbitrary radius
S1 = P1 - 0.5*r*r*einf;
S2 = P2 - 0.5*r*r*einf;
PP_dual = *(S1^S2);

// the line thru the two points of the resulting point pair
?Bisector = *(PP_dual^einf);
```



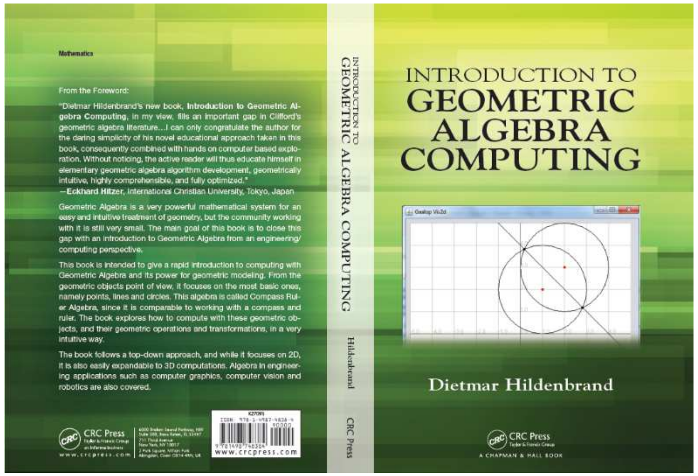

```
void calculate(float x1, float x2, float y1, float y2, float Bisector[16]) {

        Bisector[1] = x2 - x1; // e1
        Bisector[2] = y2 - y1; // e2
        Bisector[3] = ((y2 * y2) / 2.0 - (y1 * y1) / 2.0
                      + (x2 * x2) / 2.0) - (x1 * x1) / 2.0; // einf
}
```

Compass Ruler Algebra

Basic Entities

Entity	IPNS representation	OPNS representation
Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	
Circle	$C = P - \frac{1}{2}r^2 e_\infty$	$C^* = P_1 \wedge P_2 \wedge P_3$
Line	$L = \mathbf{n} + d e_\infty$	$L^* = P_1 \wedge P_2 \wedge e_\infty$
Point pair	$P_p = C_1 \wedge C_2$	$P_p^* = P_1 \wedge P_2$



GAALOP

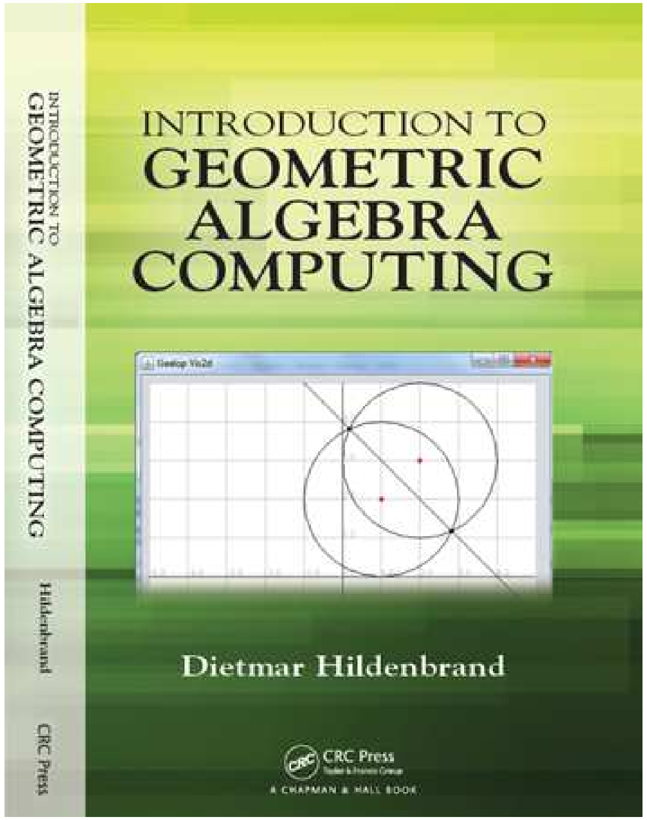
- Software to
  - visualize (2D/3D) Geometric Algebra
  - compute with Geometric Algebra (of arbitrary dimension/signature)
  - generate optimized source code from Geometric Algebra

- GAALOP (**free download** from www.GAALOP.de)

GAALOP reference

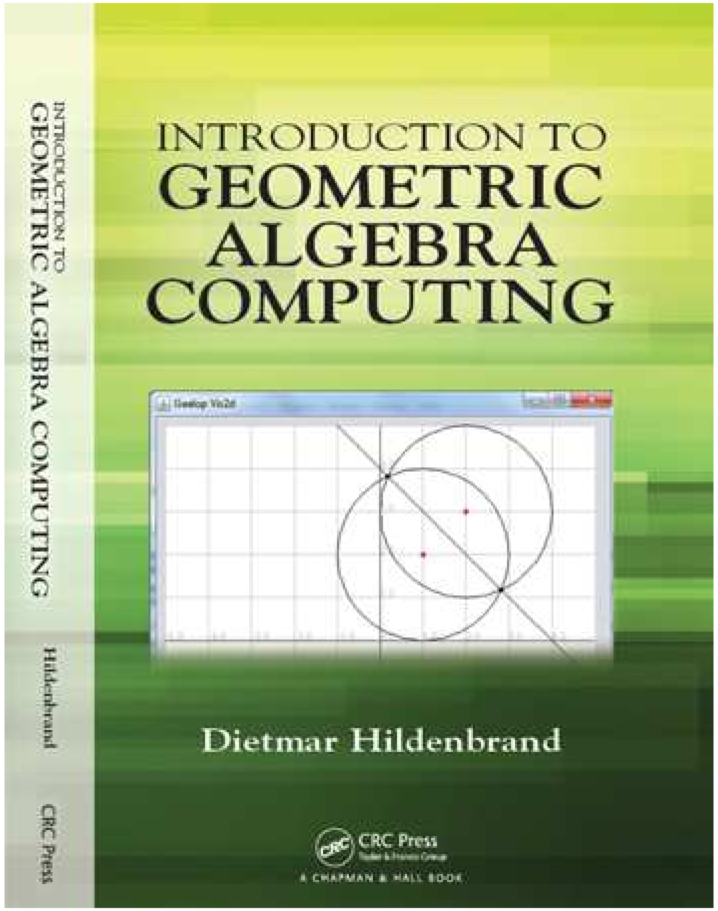
Focus on „Symbolic Geometric Algebra Calculator“

- „Introduction to Geometric Algebra Computing“
- Dietmar Hildenbrand
- CRC Press, 2019



Indices of blades of compass ruler algebra for GAALOP

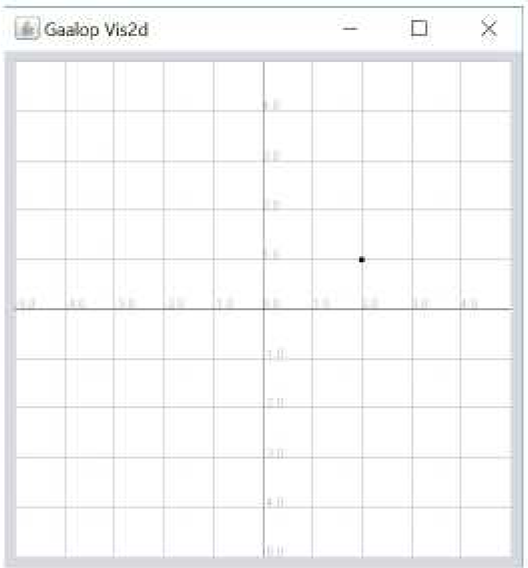
Index	Blade
0	1
1	e_1
2	e_2
3	e_∞
4	e_0
5	$e_1 \wedge e_2$
6	$e_1 \wedge e_\infty$
7	$e_1 \wedge e_0$
8	$e_2 \wedge e_\infty$
9	$e_2 \wedge e_0$
10	$e_\infty \wedge e_0$
11	$e_1 \wedge e_2 \wedge e_\infty$
12	$e_1 \wedge e_2 \wedge e_0$
13	$e_1 \wedge e_\infty \wedge e_0$
14	$e_2 \wedge e_\infty \wedge e_0$
15	$e_1 \wedge e_2 \wedge e_\infty \wedge e_0$



Point

Two alternatives

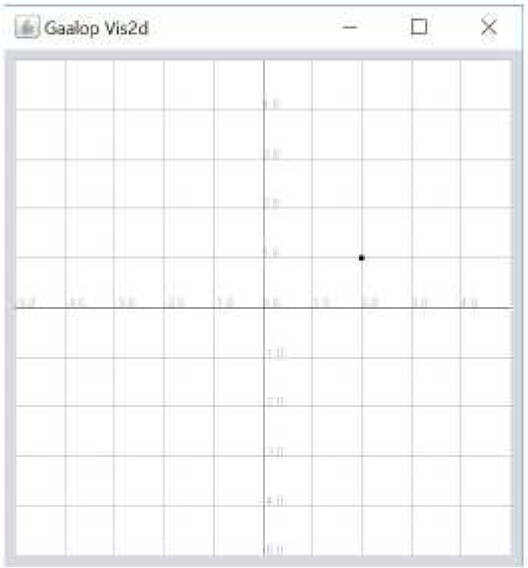
- x1 = 2;
- x2 = 1;
- P1 = x1*e1 + x2*e2 + 0.5*(x1*x1 + x2*x2)*einf + e0;
- P2 = createPoint(x1,x2);
-



Point

Additional GAALOP Code for Visualization

- // visualize the points
- :P1;
- :P2;
-



Remark: Comments with leading //

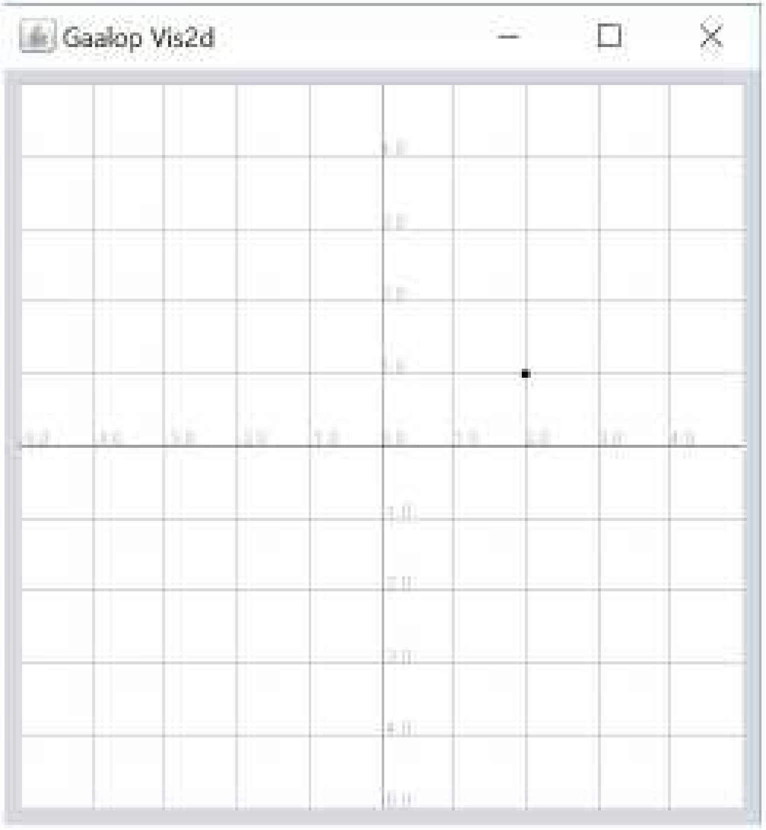
Point

Additional GAALOP Code for numerical output

- ?P1;
- ?P2;

leads to

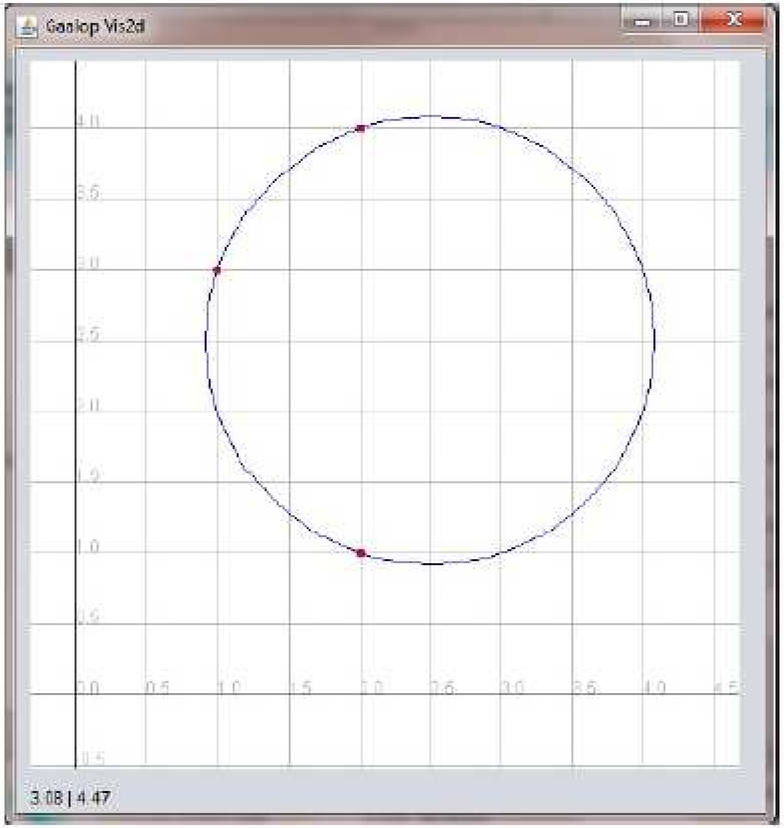
- P1(1) = 2.0 // e1
- P1(2) = 1.0 // e2
- P1(3) = 2.5 // einf
- P1(4) = 1.0 // e0
- P2(1) = 2.0 // e1
- P2(2) = 1.0 // e2
- P2(3) = 2.5 // einf
- P2(4) = 1.0 // e0



Circle

Circle based on the outer product of three points

- :Red;
- :P1 = createPoint(2,1);
- :P2 = createPoint(1,3);
- :P3 = createPoint(2,4);
- :Blue;
- :C = *(P1^P2^P3);
- ?C;

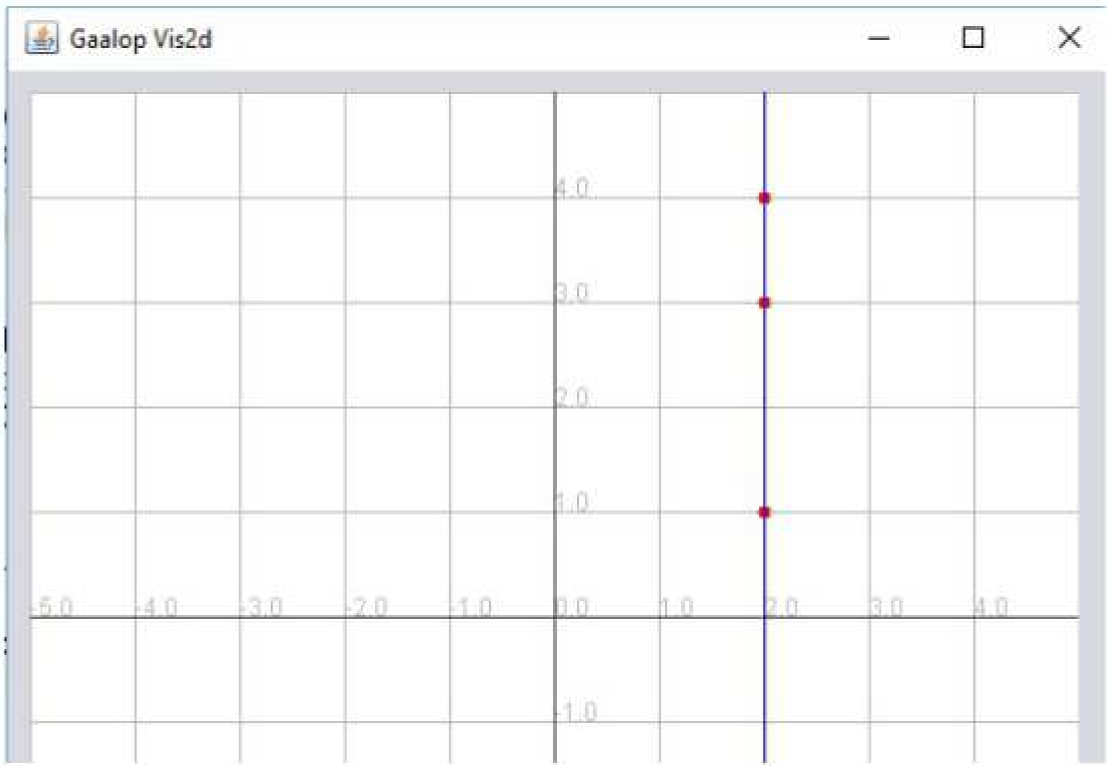


Circle

Circle based on the outer product of three co-linear points

- :Red;
- :P1 = createPoint(2,1);
- :P2 = createPoint(2,3);
- :P3 = createPoint(2,4);
- :Blue;
- :C = *(P1^P2^P3);
- ?C;

Circle with infinite radius



Line

based on the outer product of three points (including point at infinity)

- :Red;
- :P1 = createPoint(2,1);
- :P2 = createPoint(2,3);
- :Blue;
- :L = *(P1^P2^einf);
- ?L;

Circle with infinite radius

Line

based on normal vector and distance to origin

- n1 = sqrt(2)/2;
- n2 = sqrt(2)/2;
-

- n = n1*e1 + n2*e2;
- d = 2;
- :Line = n + d*einf;

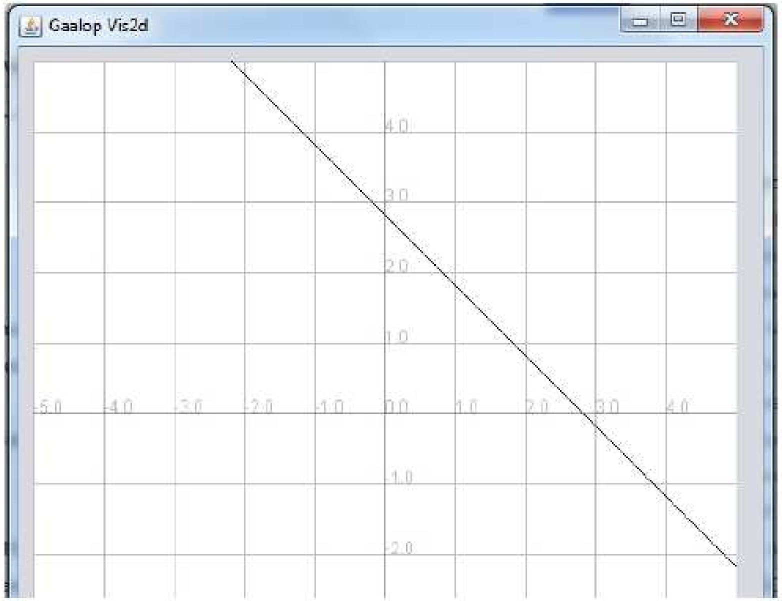
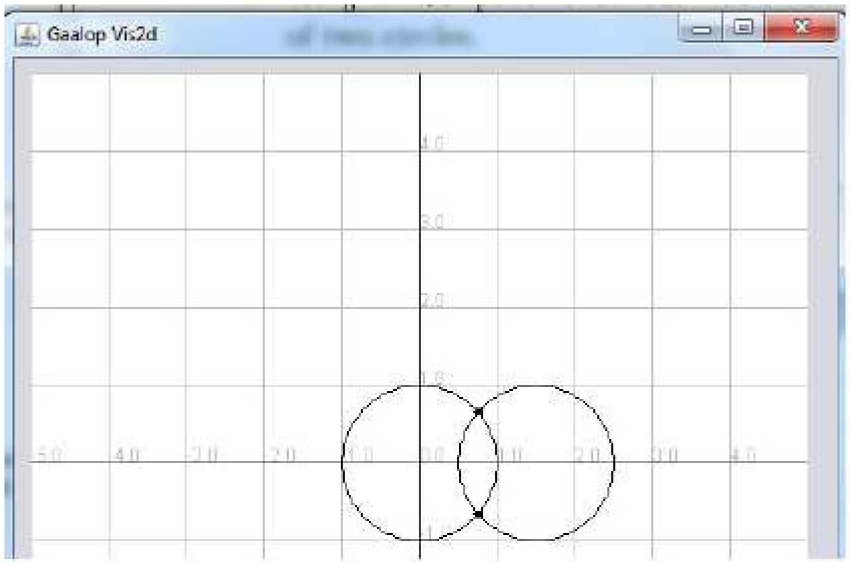


FIGURE 3.7 Visualization of Line.clu: a line based on the normal vector $\frac{1}{2}\sqrt{2} * (1, 1)$ and the distance $d = 2$.

Point pair …

… as the intersection of two circles

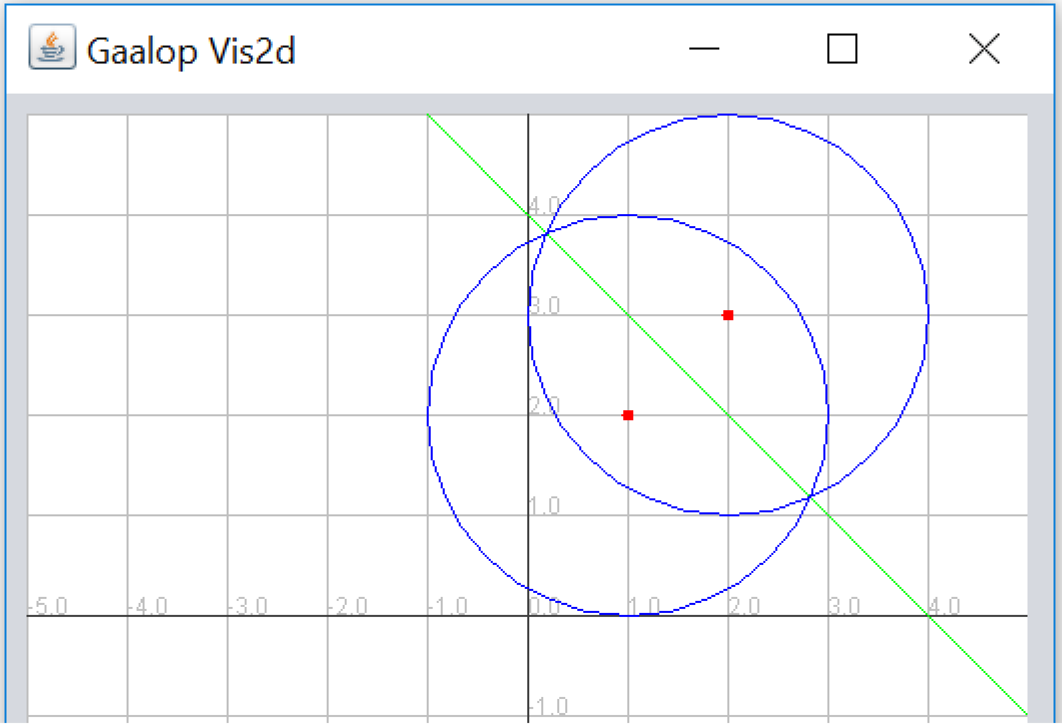
- d = 1;
- r1 = 1;
- r2 = 1;
-
- :C1 = e0-0.5*r1*r1*einf;
- :C2 = createPoint(d,0)-0.5*r2*r2*einf;
- :PP = C1^C2;



Perpendicular Bisector

Line through the intersections of two circles

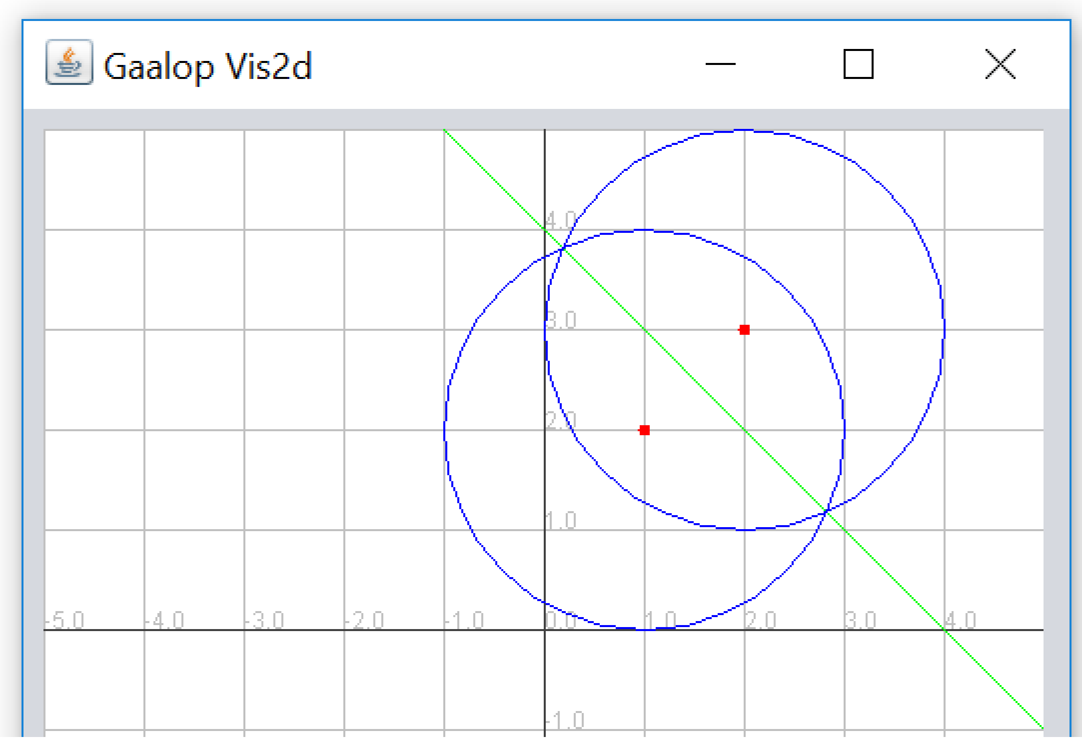
```
P1 = createPoint(x1,y1);
P2 = createPoint(x2,y2);
S1 = P1 - 0.5*r*r*einf;
S2 = P2 - 0.5*r*r*einf;
PP = S1^S2;
?L = *(*PP^einf);
```



Perpendicular Bisector

Additional GAALOP Code for Visualizations

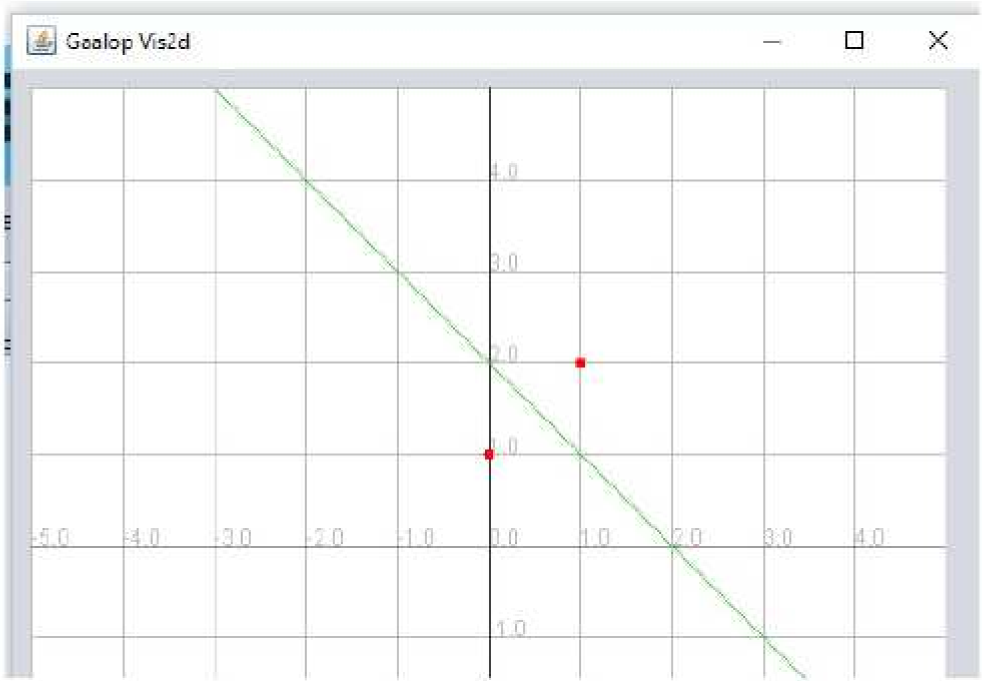
x1=1;
y1=2;
x2=2;
y2=3;



:Red;
:P1;
:P2;
:Blue;
:S1;
:S2;
:Green;
:L;

The difference of two points

- p1 = 1;
- p2 = 2;
- q1 = 0;
- q2 = 1;
-
- P = createPoint(p1,p2);
- Q = createPoint(q1,q2);
- Diff = P-Q;
-



Compass Ruler Algebra

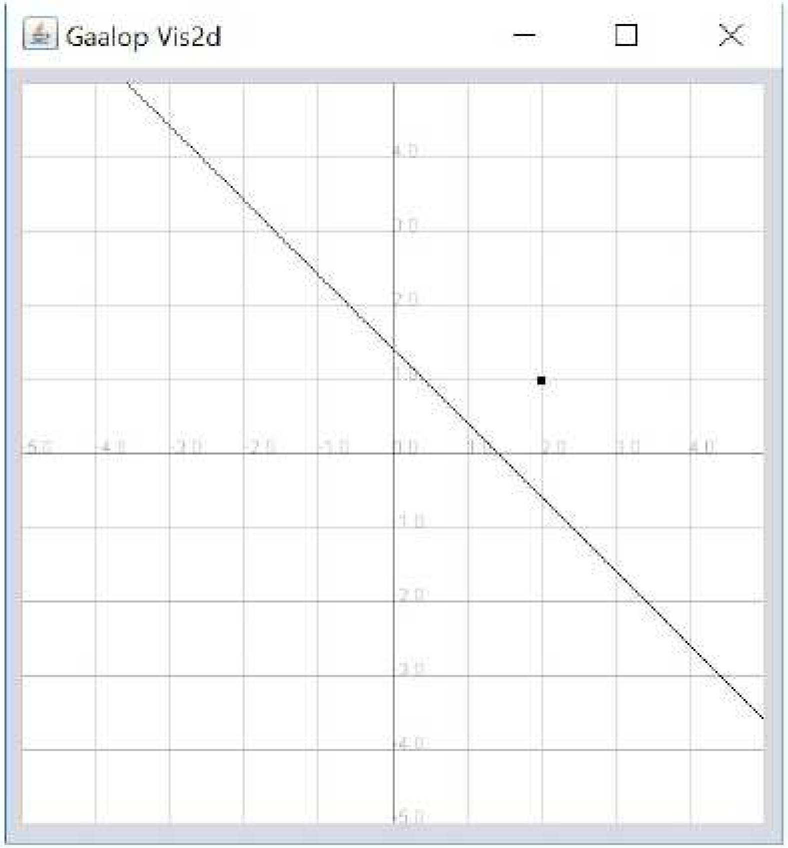
Angles and Distances

TABLE 2.3 Geometric meaning of the inner product of lines, circles and points

$A \cdot B$	B **Line**	B **Circle**	B **Point**
A **Line**	Angle between lines	Euclidean distance from center	Euclidean distance
A **Circle**	Euclidean distance from center	Distance measure	Distance measure
A **Point**	Euclidean distance	Distance measure	Distance

Distance Point-Line

- n1 = sqrt(2)/2;
- n2 = sqrt(2)/2;
- d = 1;
- p1=2;
- p2=1;
- P = createPoint(p1,p2);
- L = n1*e1+n2*e2+d*einf;
- :P;
- :L;

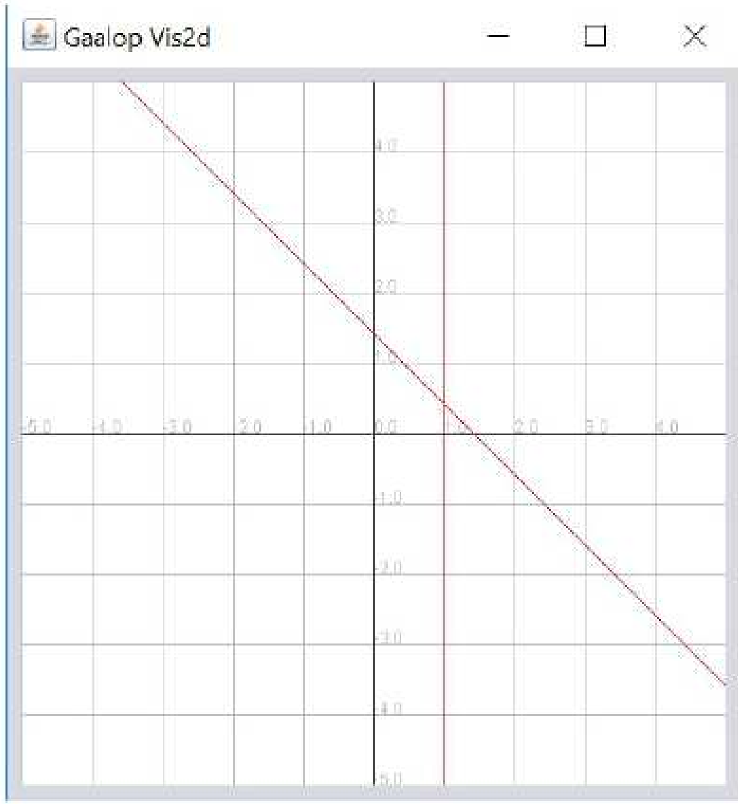


- ?Result = P.L;

Result(0) = 1.121320343559643 // 1.0

Angle between two lines

- n1 = sqrt(2)/2;
- n2 = sqrt(2)/2;
- d = 1;
- L1 = e1+d*einf;
- L2 = n1*e1+n2*e2+d*einf;
- :Red;
- :L1;
- :L2;



- ?Result = L1.L2;
- ?Angle = Acos(Result)*180/3.14159;

Result(0) = 0.7071 // 1.0

Angle(0) = 45.0 // 1.0

Compass Ruler Algebra

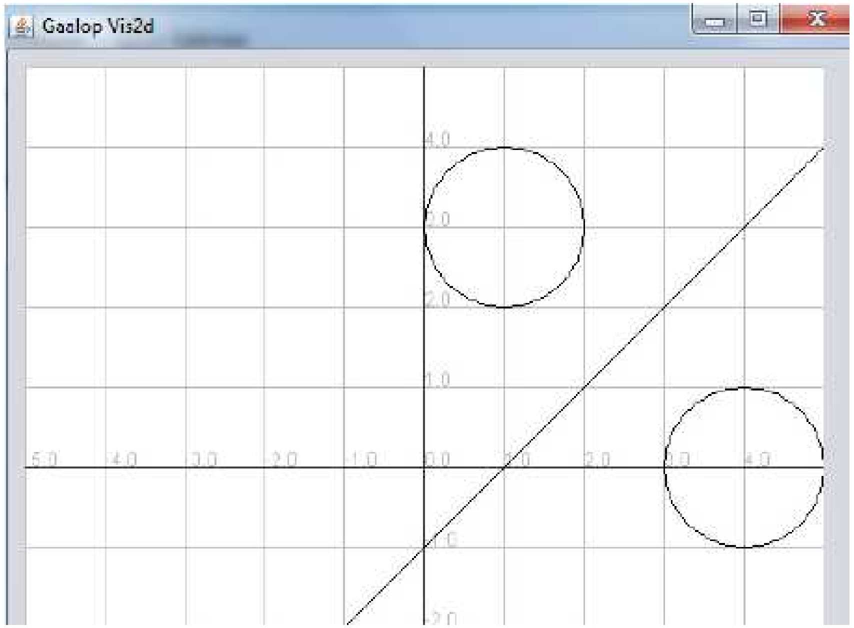
Geometric Transformations

TABLE 3.4 The GAALOPScript description of transformations of a geometric object o in Compass Ruler Algebra (note that e12 is the imaginary unit i).

	operator	**Transformation**
Reflection	L = n1*e1+n2*e2 + d* einf	-L*o*L
Rotation	R = cos (phi/2) - e12 * sin (phi/2)	R*o*(~R)
Translation	T = 1 - 0.5*(t1*e1+t2*e2)*einf	T*o*(~T)

Reflection

… of a circle at a line



- x=1;
- y=3;
- r=1;
- x1=0;
- y1=-1;
- x2=3;
- y2=2;
- :o = createPoint(x,y)-0.5*r*r*einf;
- :L = *(createPoint(x1,y1)^createPoint(x2,y2)^einf);
- :oRefl = - L * o * L;

Reflection

… of a circle at a line

- :o = createPoint(x,y)-0.5*r*r*einf;
- :L = *(createPoint(x1,y1)^createPoint(x2,y2)^einf);
- :oRefl = - L * o * L;
- ?oRefl;

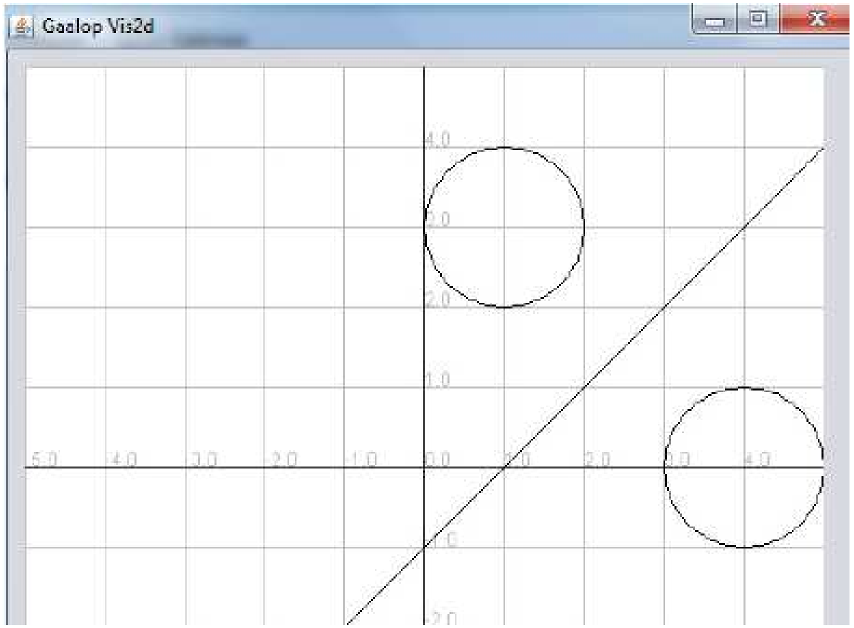
oRefl(1) = 72.0 // e1

oRefl(3) = 135.0 // einf

oRefl(4) = 18.0 // e0

or

$$o_{\text{Refl}} = 72e_1 + 135e_\infty + 18e_0$$



Reflection

… of a circle at a line

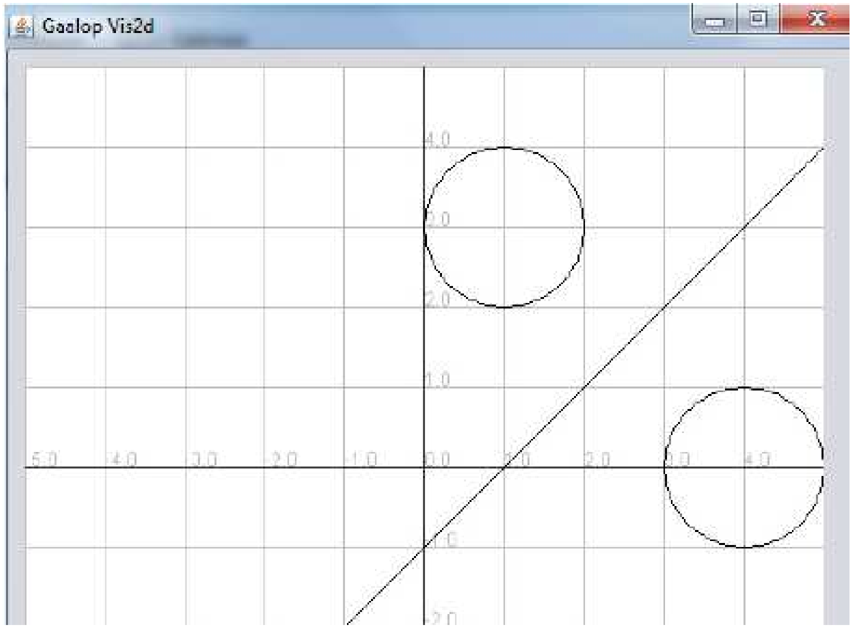
- :o = createPoint(x,y)-0.5*r*r*einf;
- :L = *(createPoint(x1,y1)^createPoint(x2,y2)^einf);
- :oRefl = - L * o * L;
- ?oRefl;

↓

oRefl(1) = 72.0 // e1
oRefl(3) = 135.0 // einf
oRefl(4) = 18.0 // e0

or

$$o_{\mathrm{Refl}} = 72e_1 + 135e_\infty + 18e_0$$



Reflection

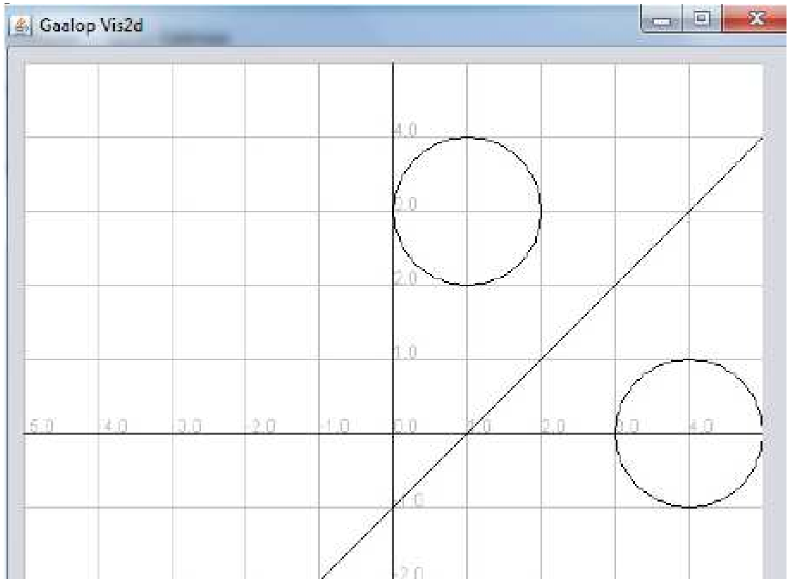
… with normalized objects

- :o = createPoint(x,y)-0.5*r*r*einf;
- L_notnormalized = *(createPoint(x1,y1)^createPoint(x2,y2)^einf);
- :L = L_notnormalized/abs(L_notnormalized);
- :oRefl = - L * o * L;
- ?oRefl;

- oRefl(1) = 4.00 // e1
- oRefl(3) = 7.50 // einf
- oRefl(4) = 1.0 // e0

or

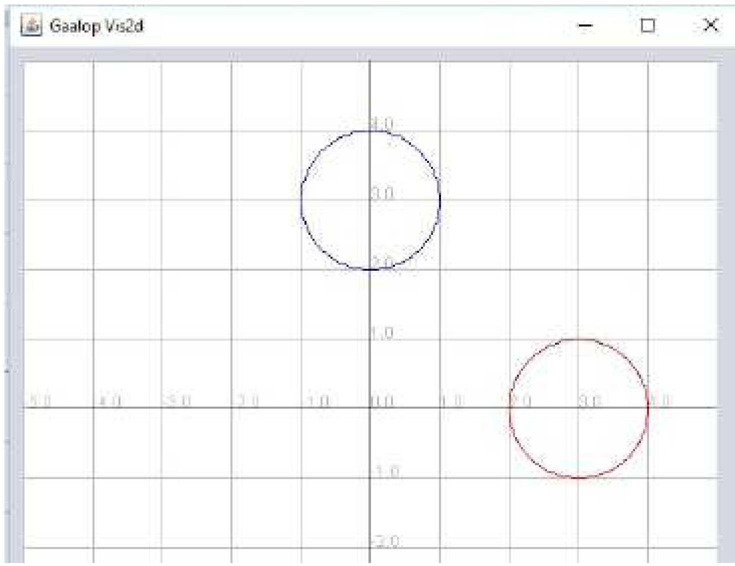
$$o_{\mathrm{Refl}} = 4e_1 + 7.5e_\infty + e_0$$



Rotation

… of a circle

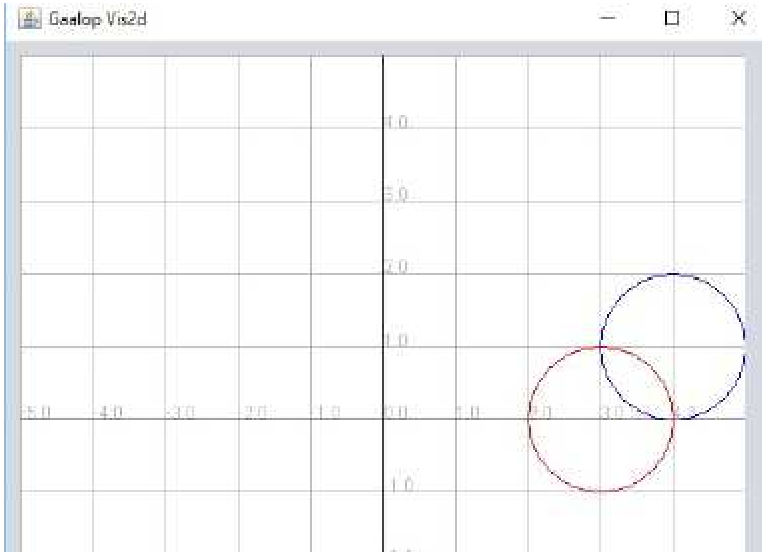
- x=3;
- y=0;
- r=1;
- angle=90;
- alpha=(angle/180)*3.1416;
- i = e1^e2;
- P = createPoint(x,y);
- Circle = P -0.5*r*r*einf;
- Rota = cos(alpha/2) - i* sin(alpha/2);
- Circle_rot = Rota * Circle * ~Rota;



Translation

… of a circle

- x=3;
- y=0;
- t1 = 1;
- t2 = 1;
- r=1;
- P = createPoint(x,y);
- Circle = P -0.5*r*r*einf;

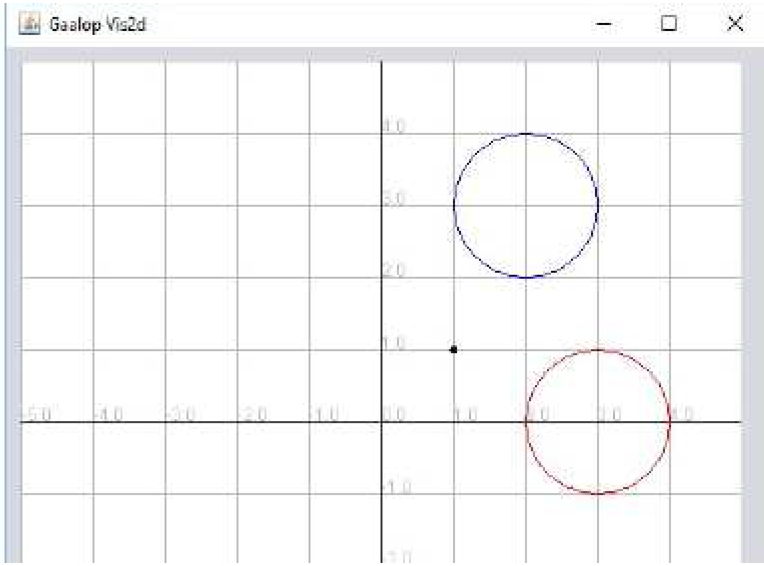


- T = 1-0.5*(t1*e1+t2*e2)^einf;
- Circle_trans = T * Circle * ~T;

Rigid Body Motion

Rotation of a circle around a point

- x=3; y=0; r=1;
- t1 = 1; t2 = 1;
- angle=90;
- alpha=(angle/180)*3.1416;
- i = e1^e2;
- P = createPoint(x,y);
- Circle = P -0.5*r*r*einf;
- Rota = cos(alpha/2) - i* sin(alpha/2);
- T = 1-0.5*(t1*e1+t2*e2)*einf;
- Motor = T * Rota * ~T;
- Circle_rot = Motor * Circle * ~Motor;



Thanks a lot

Google Dietmar Hildenbrand