# A Catalogue of Geometric Primitive Operations

Ulrich Kortenkamp, Lena Polke, Jürgen Richter-Gebert, Marc Schneider

## About this document

This document is considered to be a living document. It should describe all relevant primitive operations and elementary geometric objects that are necessary to build a powerful dynamic geometry system. For each object suitable coordinate settings are given in a projective framework. For each operations formulas and algorithms are presented. There may be one or many elegant ways to perform the calculation. Design criterion for this document is that all operations are given in a way that they make sense in a general projective framework. Geometries with measurement (like Euclidean Geometry or Hyperbolic Geometry) are represented via fundamental objects in the sense of Cayley-Klein Geometries. The calculations of the primitive operation should be given in a way that refers to these fundamental objects to make them as general as possible. In special situations where the calculations becomes significantly simpler in a certain coordinate framework this may be given. But a general formula is always required for this document.

For each operation also a rough piece of code should be provided. Code preferably in CindyScript language since this document will form the basis for embedding The Cinderella Geometry kernel in CindyScript. In addition Mathematica or JavaScript code may be given when appropriate.

# 1 Primitive geometric objects

## 1.1 Absolute projective Objects

### 1.1.1 Point

*What:* A point in the projective plane.

*Representation:* A homogeneous vector $p = (x, y, z) \in \mathbb{C}^3$. Usually this vector will be real, but complex values are explicitly admitted.

*CindyScript:* `[x,y,z]`. Use `point([x,y,z])` to force drawing as a point.

*Interpretation:* In standard embedding $p = (x, y, z) \in \mathbb{R}^3$ will be interpreted as the euclidean point $(x/z, y/z)$. Points of the form $p = (x, y, 0)$ lie on the line at infinity. Scalar multiples represent the same point.

### 1.1.2 Line

*What:* A line in the projective plane.

*Representation:* A homogeneous vector $l = (a, b, c) \in \mathbb{C}^3$. Usually this vector will be real, but complex values are explicitly admitted.

*CindyScript:* `[a,b,c]`. Use `line([a,b,c])` to force drawing as a line.

*Interpretation:* In standard embedding $p = (a, b, c) \in \mathbb{R}^3$ will be interpreted as the euclidean line $\{(x, y) \mid ax + by + c = 0\}$. A line of the form $l = (0, 0, 1)$ is the line at infinity. Scalar multiples represent the same line.

### 1.1.3 Conic

*What:* A conic in the projective plane. May degenerate into two lines or even into a double line

*Representation:* A homogeneous symmetric matrix $A \in \mathbb{C}^{3\cdot3}$. Usually this matrix will be real, but complex values are explicitly admitted.

*CindyScript:* `[[a11,a12,a13],[a12,a22,a23][a13,a23,a33]]`. Sometimes also just a vector `[a,b,c,d,e,f]`

*Interpretation:* The matrix

$$\begin{pmatrix} a & d/2 & e/2 \\ d/2 & b & f/2 \\ e/2 & f/2 & c \end{pmatrix}$$

Represents the conic (in homogeneous coordinates)

$$ax^2 + by^2 + cz^2 + dxy + exz + fyz = 0$$

Scalar multiples represent the same conic.

*Remarks:* Circles in in a geometry will be a special case of a general conic. Perhaps store as a pair of matrizes (Primal/Dual Pair in the sense of Perspectives of Projective Geometry).

#### 1.1.4 Transformation

*What:* A projective transformation. Many other transformations will be just be special cases of this (like rotations, translations, reflections etc).

*Representation:* A homogeneous matrix $A \in \mathbb{C}^{3\cdot 3}$. Determinant should not vanish. Usually this matrix will be real, but complex values are explicitly admitted.

*CindyScript:* `[[a,b,c],[d,e,f][g,h,i]]`.

*Interpretation:* The matrix $A \in \mathbb{C}^{3\cdot 3}$ acts via $p \mapsto A \cdot p$ on points. The rest is a consequence of this.

*Remarks:* It may be useful to store both $A$ and $A^{-1}$ (alternatively the adjoint $A^{\Delta}$). Many operations will make use of the Inverse.

### 1.2 Objects depending on a Geometry

#### 1.2.1 Geometry

This should be a longer section explaining *fundamental objects* of a Cayley-Klein geometry (TODO). They will be stored as a pair of matrices (Primal Dual Pair) $(\mathcal{F}, \mathcal{F}')$ satisfying $\mathcal{F} \cdot \mathcal{F}' = \lambda E$. Depending on the choice one gets very difficult geometries. Standard choices are:

**Euclidean Geometry** Here one needs a conic that degenerates into a double line and two complex conjugate points on it. Standard choice is:

$$\mathcal{F} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathcal{F}' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The line at infinity is $l = (0, 0, 1)$. The two special points are $\mathtt{I} = (-i, 1, 0)$ and $\mathtt{J} = (i, 1, 0)$. Operations may refer to these two points.

**Euclidean Geometry** Here one needs a real non-degenerate conic. Standard choice is:

$$\mathcal{F} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \qquad \mathcal{F}' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

The fundamental object here corresponds to a unit circle.

**Elliptic Geometry** Here one needs a fully complex non-degenerate conic. Standard choice is:

$$\mathcal{F} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathcal{F}' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

With this choice this is a geometry representing points on a unit sphere with antipodal points identified.

### 1.2.2 Circle

A circle is a special conic. It always refers to the fundamental conic of a geometry. It must be in a special relation to the fundamental conic. The crucial property is that it has two touching points to the fundamental conic. These touching points may be complex.

(TODO: Welche Formel charakterisiert das?)

**Euclidean Geometry** Euclidean circles may be characterized as those conics that pass through $\mathtt{I} = (-i, 1, 0)$ and $\mathtt{J} = (i, 1, 0)$.

### 1.2.3 Segment

### 1.2.4 Polygon

### 1.2.5 Angle

### 1.2.6 Distance

### 1.2.7 Area

# 2 Primitive geometric operations

## 2.1 Absolute projective Operations

### 2.1.1 Free and semifree operations

#### 2.1.1.1 Free Point

#### 2.1.1.2 Point on Line

#### 2.1.1.3 Point on Conic

#### 2.1.1.4 Point on Cubic

#### 2.1.1.5 Line Through a point

### 2.1.2 Determined operations

#### 2.1.2.1 Join Point Point

*Input:* Two points $q$ and $q$. *Output:* a line $l$.

*Calculation:* $l = p \times q$

*CindyScript:*

```
join(p,q):=
  line(cross(p,q));
);
```

*Remarks:* The join operation is already defined in CindyScript.

#### 2.1.2.2 Interesection Line Line

*Input:* Two points $l$ and $m$. *Output:* a line $p$.

*Calculation:* $p = l \times m$

*CindyScript:*

```
meet(l,m):=
  point(cross(l,m));
);
```

*Remarks:* The meet operation is already defined in CindyScript.

#### 2.1.2.3 Intersection Line Conic

*Input:* A line $l$ and a conic $A$. *Output:* a pair of points $p_1, p_2$.

*Calculation:* First calculate a matrix $M_l$ from the line $l$.

$$M_l = \begin{pmatrix} 0 & l_3 & -l_2 \\ -l_3 & 0 & l_1 \\ l_2 & -l_1 & 0 \end{pmatrix}$$

This matrix represents the operator of taking the vector product with $l$. This means $l \times g = M_l \cdot g$. The matrix $B = M_l^T A M_l$ is degenerate and represents a dual conic that has the two special points that are the intersection of $l$ with $A$. This matrix must now be decomposed into the two points. This can be done by finding a suitable $\alpha$ such that

$$M_l^T A M_l + \alpha M_l$$

is a rank one matrix. This $\alpha$ can be calculated as

$$\frac{1}{l_3}\sqrt{-\det\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}}$$

However this only works as long as $l_3 \neq 0$. Alternatively one can calculate it as

$$\alpha = \frac{1}{l_2}\sqrt{-\det\begin{pmatrix} b_{33} & b_{31} \\ b_{13} & b_{11} \end{pmatrix}} \quad \text{or} \quad \frac{1}{l_1}\sqrt{-\det\begin{pmatrix} b_{22} & b_{23} \\ b_{32} & b_{33} \end{pmatrix}}$$

One of the entries of $l$ will always be non-zero. By this calculation one only needs to evaluate one square root.

After calculating $M_l^T A M_l + \alpha M_l$ a non-zero row and a non-zero column represent the two points.

*CindyScript:*

```
matop(p):=(
  (0,p_3,-p_2),(-p_3,0,p_1),(p_2,-p_1,0)
);

intersectCL(c,l):=(
  regional(mat1,mat2,de,erg);
  mat1=matop(l);;
  mat2=mat1*c*transpose(mat1);
  de=det([mat2_1_[1,2],mat2_2_[1,2]]);
  erg=mat2-sqrt(-de)/l_3*mat1;
  [point(erg_1),point(transpose(erg)_1)];
);
```

This code does not take care of picking non-zero entries of $l$ and does not make sure that the picked row and column are non-zero. This needs some extra care. Maybe a special choice is better, even numerically.

This code would choose the index of maximal absolute value. It also normalises the objects first. However this may interfere with tracinf of roots. (TODO Does it?)

```
maxind(v):=sort(1..3,-|v_#|)_1;

matop(p):=(
  (0,p_3,-p_2),(-p_3,0,p_1),(p_2,-p_1,0)
);

intersectCL(c,l):=(
  c=c/|c|;
  l=l/|l|;
  regional(mat1,mat2,de,erg,j,j1,j2);
  mat1=matop(l);;
  mat2=mat1*c*transpose(mat1);
  j=maxind(l);
  j1=j+1;if(j1>3,j1=j1-3);
  j2=j+2;if(j2>3,j2=j2-3);
  de=det([mat2_j1_[j1,j2],mat2_j2_[j1,j2]]);

  erg=mat2-sqrt(-de)/l_j*mat1;
  [point(erg_1),point(transpose(erg)_1)];
);
```

*Tracing:* This operation must be traced. This can either happen on the level of the square root operation (Cindy in Cindy approach). Or on the level of the calculated points (CindyJS kernel). The expression under the square root is insensitive under sign flip of the conic or the line. Thus the input elements can be savely rescaled when one only considers the complex directions to trace the square roots.

#### 2.1.2.4 Find root of cubic polynomial

*Reason:* This is needed as a subroutine for the intersection cubic with cubic-

*Input:* Four complex numbers $a, b, c, d$. *Output:* three pairs complex numbers $(\lambda_1, \mu_1), (\lambda_2, \mu_2), (\lambda_3, \mu_3)$, which are three solutions of

$$a \cdot \lambda^3 + b \cdot \lambda^2\mu + c \cdot \lambda\mu^2 + d \cdot \mu^3 = 0$$

The solutions are considered to be homogeneous.

*Calculation:* The following step by step procedure produces the solution by taking exactly one square root and one cube root. First let:

$$\omega_0 = 0, \quad \omega_1 = \frac{-1 + i \cdot \sqrt{3}}{2}, \quad \omega_2 = \frac{-1 - i \cdot \sqrt{3}}{2}.$$

These are the third roots of unity. Now let

$$\begin{aligned} W &= -2b^3 + 9abc - 27a^2d \\ D &= 27(-b^2c^2 + 4ac^3 + 4b^3d - 18abcd + 27a^2d^2) \\ Q &= \sqrt[3]{4(W + a\sqrt{D})} \\ K &= 2b^2 - 6ac \end{aligned}$$

The value $Q$ is the the root required. We consider the set of all three complex roots $Q_i\omega_i Q$. For what comes the roots do not have to be distinguished. From these values the three solutions can be derived as the pairs

$$\big( - bQ_i + K + Q_i^2/2, 3aQ_i \big).$$

If one prefers so have a non homogeneous solution (i.e. a solution for $a\cdot\lambda^3+b\cdot\lambda^2+c\cdot\lambda+d = 0$) then the following formula is also nice

$$\lambda_i = \frac{b}{3a} - \frac{K}{3aQ_i} + \frac{Q_i}{6a}$$

*Remark:* In the Perspectives book there are several smaller bugs in the corresponding formula. (Sorry! It must have been late when I wrote this). The above formula is directly copied from running and tested code. So it should be correct. Still I do not entirely like the formula because of its strong break of symmetries w.r.t. the homogeneous coordinates. One should check with Jim Blinn's how to solve a cubic to learn more about cubic solving.

*CindyScript:* The following code is tested and works:

```
solvecubic(a,b,c,d):=(
 regional(w0,w1,W,D,Q,K);
 w1=-1/2+i*sqrt(3/4);
 w2=-1/2-i*sqrt(3/4);
 W=-2*b^3+9*a*b*c-27*a^2*d;
 D=(-b^2*c^2+4*a*c^3+4*b^3*d-18*a*b*c*d+27*a^2*d^2)*27;
 Q=(4*(W+a*sqrt(D)))^(1/3);
 K=2*b^2-6*a*c;
 (
   ((-b*Q+K+(Q^2)/2),(3*a*Q)),
   ((-b*(Q*w1)+K+((Q*w1)^2)/2),(3*a*(Q*w1))),
   ((-b*(Q*w2)+K+((Q*w2)^2)/2),(3*a*(Q*w2)))
 )
);
```

#### 2.1.2.5 Split a degenerate conic in two lines

*Reason:* This is needed as a subroutine for the intersection cubic with cubic-

*Input:* A symmetric matrix $A$ of a conic that is assumed to be degenerate. For what follows we require that $A$ has rank 2. If $A$ has rank 1 then each non-zero column or row of $A$ represents the desired double-line.

*Output:* A pair of lines that form the conic.

*Remark:* At several points this routine requires picking a non-zero element. For that some `if`-statements are necessary.

*Calculation:* Assume that $A$ is a symmetric and degenerate matrix. Thus it represents a degenerate conic. We are looking for lines $l$ and $m$ with $A \sim lm^T + ml^T$. Step 1: Calculate the adjoint $B = A^{\Delta}$. This matrix will be of rank 1. Each non-zero column $B_i$ represents a point $p = B_i$ that is at the intersection of the two lines of the degenerate conic. For a suitable $i$ pick such a point, such that $B_{i,i} \neq 0$ Let $d = B_{i,i}$ be the corresponding diagonal entry of that matrix. Calculate the matrix

$$M_p = \begin{pmatrix} 0 & p_3 & -p_2 \\ -p_3 & 0 & p_1 \\ p_2 & -p_1 & 0 \end{pmatrix}$$

Now calculate the matrix

$$D = A - \beta M_p \quad \text{with} \quad \beta = 1/\sqrt{-B_{i,i}}$$

A non-zero column, and non-zero row correspond to the two lines.

*CindyScript:* In the following routine it is not yet certified that the finally picked row and column are non-zero.

```
adjoint(m):=(cross(m_2,m_3),cross(m_3,m_1),cross(m_1,m_2));

splitconic(deg):=(
 regional(j,p,b,rank1);
  adj=adjoint(deg);
  j=1;if(|adj_2_2|>|adj_1_1|,j=2);if(|adj_3_3|>|adj_j_j|,j=3);
  p=adj_j;
  b=sqrt(-p_j);
  p=p/b;
  rank1=deg-matop(p);
  //TODO certify to pick non-zero row and column
  (line(rank1_2),line(transpose(rank1)_1));
);
```

*Remark:* The routine requires that the matrix is definitely rank 2. Otherwise the results are meaningless. Is there a more generic way that also gives some reasonable results even if $A$ is non-degenerate. (Most probably not).

#### 2.1.2.6 Intersection Conic Conic

*Strategy:* First consider the pencil of conics $\lambda A + \mu B$. Find $\lambda, \mu$ such that this linear combination is a degenerate conic. This requires to solve a cubic equation. After finding these parameters split the conic $\lambda A + \mu B$ into two lines. Finally intersect each of these lines with the conic $A$.

*Calculation:*

There are two bigger subroutines needed for calculating the intersection of two conics. One is finding the roots of a cubical polynomial, the other ist splitting a degenerate conic into two lines. Also the intersection of a conic and a line is required. These routines are describe at a prior place in this document.

Let $A_1, A_2, A_3$ be the colums of $A$ and $B_1, B_2, B_3$ be the colums of $B$. with

$$\begin{aligned}
\alpha &= \det(A_1, A_2, A_3) \\
\beta &= \det(B_1, A_2, A_3) + \det(A_1, B_2, A_3) + \det(A_1, A_2, B_3) \\
\gamma &= \det(B_1, B_2, A_3) + \det(A_1, B_2, B_3) + \det(B_1, A_2, B_3) \\
\delta &= \det(B_1, B_2, B_3)
\end{aligned}$$

consider

$$\alpha \cdot \lambda^3 + \beta \cdot \lambda^2 \mu + \gamma \cdot \lambda \mu^2 + \delta \cdot \mu^3 = 0$$

and find a solution $(\lambda, \mu)$ by using the `solvecubic` routine. Then consider the the Matrix $C = \lambda \cdot A + \mu \cdot B$. This matrix is degenerate and can be split into two lines $l, m$ by using the `splitconic` routine. Finally, intersect these two lines with one of the conics $A$ or $B$.

*CindyScript:*

```
intersectCC(A,B):=(
  A=A/|A|;B=B/|B|;
  aa=det(A);
  bb=det((A_1,A_2,B_3))+det((A_1,B_2,A_3))+det((B_1,A_2,A_3));
  cc=det((A_1,B_2,B_3))+det((B_1,A_2,B_3))+det((B_1,B_2,A_3));
  dd=det(B);
  roots=solvecubic(aa,bb,cc,dd);
  deg1=roots_3_1*A+roots_3_2*B;
  erg=splitconic(deg1);
  intersectCL(B,erg_1)++intersectCL(B,erg_2);
);
```

This routine produces four solutions. The may be complex. If some of the intersections are already known there may be simpler routines for this (see below). In this implementation the matrices are normalized. This might be a problem with tracing.

*Tracing:* The final four objects are traced by the the usual Cinderella approaches. It would be desirable to trace this operation entirely on the level of tracing square and cube roots. This might be tricky but doable. All in all during the calculation there are (in order of appearance) a square root and a cube root while solving the cubic. Then a square root when splitting the conic. And two more square roots when intersecting the lines with a conic. (one of them could be avoided by splitting a second solution and only intersect lines).

*Remarks:* I'm not entirely happy with this computation. It breaks symmetry at several places. Tracing may be difficult on the level of roots (but doable) If one calculates the solution of a quartic on Mathematica it seems that only a square root and a cube root are involved. So there is space for improvement.

#### 2.1.2.7 Intersection Conic Conic with three known points

```
intersectionCC2known(A,B,X,Y):=(
  regional(l,p,C,aa,bb,cc);
  p=X+Y;
  l=join(X,Y);
  C=p*A*p*B-p*B*p*A;
  aa=if(l_1!=0,C_1_1/l_1,if(l_2!=0,C_1_2/l_2*2,C_1_3/l_3*2));
  bb=if(l_2!=0,C_2_2/l_2,if(l_3!=0,C_2_3/l_3*2,C_2_1/l_1*2));
  cc=if(l_3!=0,C_3_3/l_3,if(l_1!=0,C_3_1/l_1*2,C_3_2/l_2*2));

  line((aa,bb,cc));
);
```

#### 2.1.2.8 Intersection Conic Conic with three known points

```
intersectionCC3known(A,B,X,Y,Z):=(
lxz=cross(X,Z);
lyz=cross(Y,Z);

lx=join(
    meet(
     join(inverse(A)*lxz,inverse(B)*lxz),
     join(Z,Y);
    ),X
);
```

```
ly=join(
    meet(
     join(inverse(A)*lyz,inverse(B)*lyz),
     join(Z,X);
    ),Y
);

meet(lx,ly);

);
```

#### 2.1.2.9 Intersection Line Cubic

#### 2.1.2.10 Intersection Conic Cubic

#### 2.1.2.11 Intersection Cubic Cubic

#### 2.1.2.12 Conic by five points

#### 2.1.2.13 Polar to Conic from Point

#### 2.1.2.14 Polar to Conic from Line

#### 2.1.2.15 Conic by 4 Points and 1 Line

#### 2.1.2.16 Conic by 4 Lines and 1 Point

#### 2.1.2.17 Conic by 5 Lines

#### 2.1.2.18 Projective trafo

#### 2.1.2.19 Composition

#### 2.1.2.20 Inverse

#### 2.1.2.21 Transform by Function

#### 2.1.2.22 IFS

#### 2.1.2.23 Locus

#### 2.1.2.24 Animation

#### 2.1.2.25 Map Point by Trafo

#### 2.1.2.26 Map Line by Trafo

#### 2.1.2.27 Map Conic by Trafo