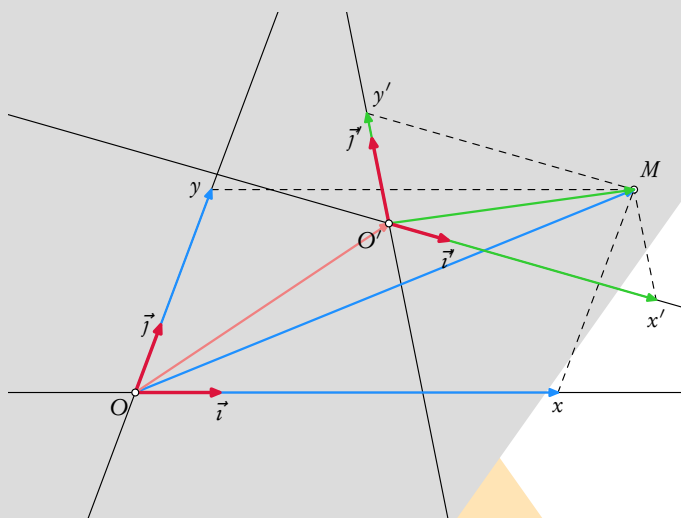


# mp-geom2d

METAPOST package for plane geometry figures



**Contributors**  
Maxime CHUPIN  
Jean-Michel SARLAT

Version 1.4 from 3rd December 2025

<https://gitlab.gutenberg-asso.fr/mchupin/mp-geom2d>

<https://ctan.org/pkg/mp-geom2d>

# Contents

<b>1</b>	<b>Objectif</b>	<b>4</b>
<b>2</b>	<b>Files</b>	<b>4</b>
<b>3</b>	<b>General Operating Principle</b>	<b>5</b>
<b>4</b>	<b>Drawing</b>	<b>5</b>
4.1	Unit . . . . .	5
4.2	In Place . . . . .	5
4.3	Drawing Commands . . . . .	6
4.4	Clipping the drawings . . . . .	10
<b>5</b>	<b>Types</b>	<b>10</b>
5.1	The point type . . . . .	11
5.1.1	Constructor . . . . .	11
5.1.2	Associated Macros . . . . .	12
5.2	The vecteur type . . . . .	16
5.2.1	Constructors . . . . .	16
5.2.2	Associated Macros . . . . .	17
5.3	The segment type . . . . .	18
5.3.1	Constructor . . . . .	19
5.3.2	Associated Macros . . . . .	19
5.4	The droite type . . . . .	20
5.4.1	Constructor . . . . .	20
5.4.2	Associated Macros . . . . .	20
5.5	The cercle type . . . . .	23
5.5.1	Constructors . . . . .	23
5.5.2	Associated Macros . . . . .	24
5.6	The ellipse type . . . . .	28
5.6.1	Constructors . . . . .	28
5.6.2	Associated Macros . . . . .	30
5.7	The parabole type . . . . .	33
5.7.1	Associated Functions . . . . .	34
5.8	The hyperbole type . . . . .	36
5.8.1	Associated Functions . . . . .	38
5.9	The triangle type . . . . .	40
5.9.1	Constructor . . . . .	40
5.9.2	Associated Macros . . . . .	41
5.10	The polygon type . . . . .	42
5.10.1	Constructors . . . . .	43
5.10.2	Associated Macros . . . . .	44
5.11	The chemin type . . . . .	44
5.11.1	Associated Macros . . . . .	45
5.12	The courbe type . . . . .	45
5.12.1	Constructor . . . . .	45
<b>6</b>	<b>Circle Arc</b>	<b>45</b>

<b>7</b>	<b>Some Macros</b>	<b>47</b>
<b>8</b>	<b>Some Transformations</b>	<b>48</b>
8.1	Homothety . . . . .	48
8.2	Axial Symmetry . . . . .	48
8.3	Central Symmetry . . . . .	49
8.4	Inversion . . . . .	50
8.5	Similarity with center (direct) . . . . .	50
<b>9</b>	<b>Annotations and Labels</b>	<b>51</b>
9.1	Right Angle Marks . . . . .	51
9.2	Labels . . . . .	52
<b>10</b>	<b>Coordinate System</b>	<b>54</b>
<b>11</b>	<b>Some Mathematical Constants and Functions</b>	<b>59</b>
<b>12</b>	<b>Representation of Curves and Functions</b>	<b>60</b>
12.1	Function of a Real Variable . . . . .	60
12.2	Parametric Curve . . . . .	60
12.3	Vector Fields . . . . .	62
<b>13</b>	<b>SVG Colors (svgnames)</b>	<b>65</b>
<b>14</b>	<b>Gallery</b>	<b>67</b>
14.1	Coordinate System and External Tangents . . . . .	67
14.2	Vector in a Coordinate System . . . . .	68
14.3	Pascal's Theorem . . . . .	70
14.4	Hyperbola . . . . .	71
14.5	Astroid as an Envelope of Lines . . . . .	72
14.6	Lissajous Curve . . . . .	73
14.7	Function Study . . . . .	74
14.8	Bicorne . . . . .	75
14.9	A function et its derivatives . . . . .	76
14.10	Cardioid . . . . .	77
14.11	Axis of similitude . . . . .	78
14.12	Cubic Drawing . . . . .	81
14.13	Apollonius . . . . .	83
14.14	Ogive Working Drawing . . . . .	85
14.15	Pedal Triangle . . . . .	87
14.16	Pappus Chain . . . . .	89
14.17	The Pythagoras tree using similarity . . . . .	91
<b>15</b>	<b>Historique</b>	<b>92</b>
	<b>Références</b>	<b>92</b>
	<b>Index</b>	<b>94</b>

## 1 Objectif

The `mp-geom2d` package was written with the goal of providing METAPOST macros that allow the creation of geometric figures by closely following an imperative description:

*Let  $A$  be the point with coordinates  $(2,3)$ .*

*Let  $B$  be the point with coordinates  $(4,5)$ .*

*Draw the line  $(A,B)$ .*

....

In this context, geometric objects are most often named ( $A$ ,  $B$ , etc.) or designated by their nature and attributes (line  $(A,B)$ , etc.). To avoid having to go beyond this mode of description, particularly to avoid having to declare the *type* of these objects, the choice was made to identify them by an *index*<sup>1</sup> in tables that specify their characteristics.

**Note** — As of today, the goal has not been fully achieved, and the development is far from complete; it is still necessary to use METAPOST commands or *intermediate macros* to describe a figure. This will likely evolve over time as a satisfactory syntax is found...

## 2 Files

The `mp-geom2d` package is a set of tools for plane geometry with METAPOST [5]. This set is organized into several files:

1. `geom2d.mp` : this is the main file, which loads all the others;
2. `geom2d-main.mp` : contains the general structures and functions;
3. `geom2d-arc.mp` : contains everything related to circle arcs;
4. `geom2d-c2d.mp` : contains everything related to second-degree curves;
5. `geom2d-fct.mp` : contains some usual mathematical functions;
6. `geom2d-lbl.mp` : contains functions related to labels;
7. `geom2d-plt.mp` : contains functions facilitating the representation of mathematical functions;
8. `geom2d-rep.mp` contains various tools for drawing figures in a coordinate system;
9. `geom2d-tra.mp` contains functions for managing transparency (code borrowed from Anthony Phan);
10. `geom2d-svgnames.mp` provides the 150 colors of the SVG specification.

In the following sections, we will describe each of these functions in more detail. Some functions rely on the `graph.mp` extension, which is included in all good T<sub>E</sub>X distributions.

---

<sup>1</sup>The `numeric` type, which is the default type in METAPOST, does not require prior declaration.

### 3 General Operating Principle

mp-geom2d uses tables as the main structure. Each object is numbered via the counter `gdd0`, its type<sup>2</sup> is stored in the table `gddT[]` at the position `gddT[gdd0]`. The properties of the objects are defined in, again, tables of type `numeric` which are `gddA[]`, `gddB[]`, ..., `gddF[]`.

For example, for a `point` (type `mp-geom2d`), the first coordinate is found in `gddA[]` and the second in `gddB[]` (the other tables are not used for such an object).

There are three special tables: `gddP[]` which is of type `path`, `gddPX[][]` which is its extended version, and `gddS[]` which is of type `string`. We will see later what their utility is.

Of course, during a typical use of `mp-geom2d`, calling all these tables is not common practice. The functions we will describe in the rest of this document allow you to avoid having to deal too precisely with this machinery.

### 4 Drawing

mp-geom2d provides macros to draw objects. A command allows you to draw most of the `mp-geom2d` objects (which are described in section 5), as well as some `METAPOST` types. The examples throughout the documentation will illustrate the drawing commands.

To understand how representations work with `mp-geom2d`, we first need to describe the unit management mechanism.

#### 4.1 Unit

mp-geom2d defines a general unit with the global variable `gddU`. This `numeric` is by default 1 cm.

`gddU`

#### 4.2 In Place

When using the drawing commands, `mp-geom2d` uses the macro

`gddInPlace`

This macro specifies the necessary geometric transformations for placing the object, particularly the scaling factor relative to the `gddU` unit.

Internally, this macro is a `scantoken` of the global variable `gddW` (`string`) which contains the transformations.

---

<sup>2</sup>The types are specific to `mp-geom2d` and will be described later.

## 4.3 Drawing Commands

The most common drawing macro is the `trace` macro.

### `gddDraw(object)`

- (*object*):**
- for the METAPOST side, a `path`, a `picture` or a `pair`, in which case `gddDraw` will be equivalent to `draw`;
  - for the mp-geom2d side, any of the 6 objects defined so far: `triangle`, `segment`, `droite`, `cercle`, `ellipse`, `parabole`, `hyperbole`, `polygone`, `chemin`, `courbe` or `vecteur` (for the latter, an arrow will be added to the end of the vector).

When `gddDraw` is used on an mp-geom2d object, the macro calls the METAPOST command `draw` with the placement mechanism described above.

mp-geom2d provides a drawing with an arrow that no longer calls `draw` but `drawarrow`

### `gddArrow(object)`

- (*object*):**
- for the METAPOST side, a `path`, a `picture`, or a `pair`, in which case `trace` will be equivalent to `drawarrow`;
  - for the mp-geom2d side, any of the 6 objects defined so far: `triangle`, `segment`, `droite`, `cercle`, `ellipse`, `polygone`, `chemin`, `courbe`, or `vecteur` (for the latter, an arrow will be added to the end of the vector).

You can also color an object using the following macro.

### `gddFill(object)`

**(*object*):** can be:

- for the METAPOST side, only a `path` and in this case `trace` will be equivalent to `fill`;
- for the mp-geom2d side, any colorable object: `triangle`, `cercle`, `ellipse`, `polygone`, `chemin`, or `courbe`.

The drawing commands of mp-geom2d use the basic macros of METAPOST: `draw` and `fill`. Thus, to specify the color, you can use the `withcolor` command or the `drawoptions` macro.

You can also color with transparency with the following command.

`gddAlphaFill(object , color , transparency coefficient)`

***object***): can be:

- for the METAPOST side, only a `path` and in this case `trace` will be equivalent to `fill`;
- for the mp-geom2d side, any colorable object: `triangle`, `cercle`, `ellipse`, `polygone`, `chemin`, or `courbe`.

***color***): is a `color`.

***transparency coefficient***): is a `numeric` between 0 and 1, 0 for opaque coloring and 1 for invisible coloring.

In fact, with a classic use, transparency is simulated using code borrowed from Anthony Phan: <http://www-math.univ-poitiers.fr/~phan/metalpha.html> . If one uses mp-geom2d with the `luamplib` package, the macro `gddAlphaFill` calls the `withtransparency` macro, which uses PDF transparency and is much lighter in computation. Thus, some code may be compilable with `luatex` but not with `mpost`.

You can also hatch a closed shape<sup>3</sup> with the following command:

`gddHatch(object , angle , spacing , type) → image`

***object***): can be:

- for the METAPOST side, only a `path` and in this case `trace` will be equivalent to `fill`;
- for the mp-geom2d side, any colorable object: `triangle`, `cercle`, `ellipse`, `polygone`, `chemin`, or `courbe`.

***angle***): is the angle (`numeric` in degrees) of the hatching lines.

***spacing***): is the space (`numeric`) between each hatching line (in the unit of mp-geom2d, so no unit should be specified).

***type***): is the type of line. If `<type>= 0`, the lines will be solid, if `<type>= 1`, the lines will be dashed evenly, if `<type>= 2`, the lines will be axis lines, if `<type>= 3`, the lines will be dashed with dots. Finally, if `<type>= 4`, you can define the line pattern with the `HatchPattern` command.

Note that the `gddHatch` macro returns an `image`, so it should be used in conjunction with the `gddDraw` macro.

To use the fourth type of line for hatching, you need to define its pattern using the following command:

---

<sup>3</sup>The code is taken and adapted from the set of macros `geometriesyr16` by Christophe Poulain <https://melusine.eu.org/syracuse/poulecl/geometriesyr16/>.

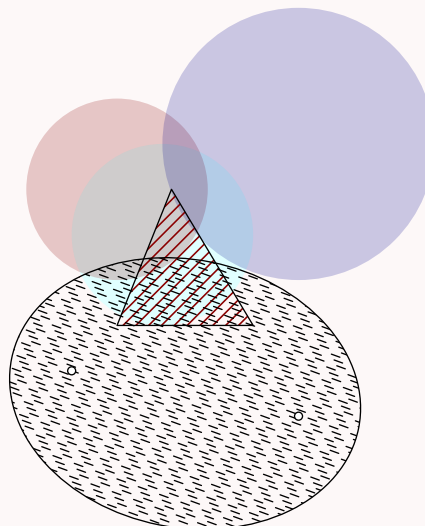
## HatchPattern(*pattern*)

**(*pattern*)**: is the pattern that you would typically provide to the `dashpattern` command. We refer you to the METAPOST documentation. Note that the dimensions provided to the `on` and `off` commands will be scaled to the `gddU` unit of `mp-geom2d`. Therefore, they should be considered as expressed in the `mp-geom2d` unit.

Below, we illustrate some commands for drawing, coloring, and hatching.

### Example 1

```
1 input geom2d;
2
3 beginfig(1);
4 gddU:=0.6cm;
5 A = Point(-1,1); B = Point(3,2);
6 C1= Circle(origine,2);
7 C2 = Circle(A,2); C3 = Circle(B
  ,3);
8
9 gddFill C1 withcolor LightCyan;
10 gddAlphaFill(C2,DarkRed,0.2);
11 gddAlphaFill(C3,DarkBlue,0.2);
12
13 T1 = Triangle((-1,-2),(2,-2)
  ,(0.2,1));
14 gddDraw gddHatch(T1,45,0.2,0)
  withPen(0.6,DarkRed);
15 gddDraw T1;
16 pE1 = Point(-2,-3);
17 pE2 = Point(3,-4);
18 E = EllipseF(pE1,pE2,3.9);
19 HatchPattern(on0.2 off0.3);
20 gddDraw gddHatch(E,-20,0.1,4);
21 gddDraw E;
22 gddDrawPoint pE1; gddDrawPoint pE
  2;
23 endfig;
```



The above macros call a lower-level macro that returns the `path` to be drawn or colored for all `mp-geom2d` objects.

## `gddObjectDrawing(object)` → `path`

**(*object*)**: `triangle`, `segment`, `vecteur`, `cercle`, `ellipse`, `parabole`, `hyperbole`, `polygone`, `chemin`, or `courbe`.

If the object is not closed, `mp-geom2d` provides the following macro that adds a `--`



`cycle` to the end of the path:

`gddClose((object))` → `path` (closed with `--cycle`)

- `(object)`:
- on the *METAPOST* side, `path`;
  - on the *mp-geom2d* side, `triangle`, `segment`, `vecteur`, `cercle`, `chemin`, or `courbe`.

You can specify the *pen* used for drawings with the following command:

`withPen((size), (color))`

`(size)`: scaling factor (`numeric`) of the `pen pencircle` of *METAPOST*;

`(color)`: color `color` to use.

This macro is a shortcut for the *classic* `withpen pencircle scaled ... withcolor ...`. You can, of course, always use the *METAPOST* commands for drawing.

You can represent objects of type `point`. For this, there is a command that draws a small colored circle at the location of the point. The command is as follows:

`gddDrawPoint((point))`

`(point)`: `point` or `pair`

This macro uses three internal parameters of *mp-geom2d* for the size, color, and type of mark used (`gddTaillePoint`, `gddCouleurPoint` and `gddPointeType`). These parameters can be modified with the following commands:

`SetPointSize((n))`

`(n)`: `numeric`, default value 3.

`SetPointColor((c))`

`(c)`: `color`, default value `white`.

There are two other types of point marking: the cross and the disk. To choose another type of marking (using the same `gddDrawPoint` command), you can use the following macro:

### SetPointType(*s*)

*s*: **string**, can be "pointe" (default behavior with a black circle filled with another color, white by default), "croix" (for a simple cross), or "disque" (for a solid disk).

It is often necessary to mark a list of points<sup>4</sup>. For this, the following command is available:

### gddDrawPoints(*pointA*, *pointB*, *pointC*, etc.)

*pointA*, *pointB*, etc.: list of **point** or **pair** separated by commas.

## 4.4 Clipping the drawings

It is often necessary to restrict the drawings to a specific area of the plane  $\mathbb{R}^2$ . For this, mp-geom2d provides the command **Window**:

### Window(*Xmin*, *Ymin*, *Xmax*, *Ymax*)

This function draws a rectangle with two opposite vertices at the points (*Xmin*, *Ymin*) and (*Xmax*, *Ymax*) with the METAPOST color **background** and clips the current picture around this frame (using the METAPOST command **clip currentpicture**).

You can also restrict drawings to the interior of a closed path derived from an mp-geom2d object. To do this, use the following command:

### gddClip(*object*)

*object*: any mp-geom2d "closed" object.

Internally, the macro **Window** modifies the MetaPostextra\_endfig variable. To clear this character string, you can use the following command:

### EndWindow

## 5 Types

mp-geom2d provides several types of objects. The object type is stored in the **gddT[]** table, and the **gddA[]** to **gddF[]** tables, as well as the **gddX** table, contain the properties of the

---

<sup>4</sup>Based on an idea by Alain Matthes.

objects.

Here, we will describe each type of the `mp-geom2d` extension, their respective properties, as well as methods directly associated with the object.

## 5.1 The `point` type

This type corresponds to a point in Euclidean space.

### 5.1.1 Constructor

To be clearer, here is the main function to create such an object:

```
1 vardef Point(expr a,b) =  
2   gddT[incr gdd0] = "point";  
3   gddA[gdd0] = a; gddB[gdd0] = b; gdd0  
4 enddef;
```

`Point(x,y)` → `point`

*x*: `numeric`;

*y*: `numeric`.

This function returns the counter `gdd0` and creates an entry in the type table as `point` and the corresponding attributes (coordinates) `a` and `b` in the tables `gddA` and `gddB`.

With such a type of operation, most manipulations are done on `numeric`s. Indeed, to declare a `point`, it is sufficient to write

```
1 A = Point(2,3);
```

`A` then takes the current value of `gdd0`. This is the identifier of the point.

A `point` can also be defined by its *polar* coordinates with the following command.

`PolarPoint(r,a)` → `point`

*r*: `numeric`, modulus of the point;

*a*: `numeric`, angle with respect to the x-axis.

You can define a point in a coordinate system defined by an origin point and two vectors with the following function:

`PointInBasis( $\langle x \rangle, \langle y \rangle, \langle o \rangle, \langle i \rangle, \langle j \rangle$ )` → `point`

`$\langle x \rangle$` : `numeric` ;  
 `$\langle y \rangle$` : `numeric` ;  
 `$\langle o \rangle$` : `point` origin of the coordinate system ;  
 `$\langle i \rangle$` : `vecteur` (or `point`) defining the x-axis ;  
 `$\langle j \rangle$` : `vecteur` (or `point`) defining the y-axis.

The following code illustrates the use of this command (with `points` instead of `vecteurs`, type described later).

```
1 O = Point(0,0);  
2 I = Point(2,0);  
3 J = PolarPoint(2,Pi/2);  
4 A = PointInBasis(2,3,0,I,J);
```

### 5.1.2 Associated Macros

You can retrieve the x-coordinate and y-coordinate of a point with the following commands:

`Xcoordinate( $\langle a \rangle$ )` → `numeric`

`$\langle a \rangle$` : `point`, `vecteur` or `pair`.

`Ycoordinate( $\langle a \rangle$ )` → `numeric`

`$\langle a \rangle$` : `point`, `vecteur` or `pair`.

You can add the x-coordinates, the y-coordinates, or, like vectors, points together with the following commands.

`XcoordinateSum( $\langle a \rangle, \langle b \rangle$ )` → `numeric`

`$\langle a \rangle$` : `point` or `pair`;

`$\langle b \rangle$` : `point` or `pair`.

`YcoordinateSum( $\langle a \rangle, \langle b \rangle$ )` → numeric

$\langle a \rangle$ : point or pair;

$\langle b \rangle$ : point or pair.

and

`PointSum( $\langle a \rangle, \langle b \rangle$ )` → point

$\langle a \rangle$ : point or pair;

$\langle b \rangle$ : point or pair.

You can also calculate the distance between two points using the following command:

`Length( $\langle a \rangle, \langle b \rangle$ )` → numeric

$\langle a \rangle$ : point or pair;

$\langle b \rangle$ : point or pair.

You can calculate the midpoint between two points:

`MidPoint( $\langle a \rangle, \langle b \rangle$ )` → point

$\langle a \rangle$ : point or pair;

$\langle b \rangle$ : point or pair.

You can perform the rotation of a point around the origin (0,0) with the following command:

`Rotation( $\langle a \rangle, \langle b \rangle$ )` → point

$\langle a \rangle$ : point or pair;

$\langle b \rangle$ : numeric, the angle of rotation in radians.

If you want to rotate a point around another point, you will use the following command:

`CenterRotation( $\langle a \rangle, \langle b \rangle, \langle c \rangle$ )` → point

$\langle a \rangle$ : point or pair;

$\langle b \rangle$ : point or pair, center of rotation;

$\langle c \rangle$ : numeric, the angle of rotation in radians.

We can calculate the isobarycenter of a list of points (or pair of METAPOST) with the following command.

`IsoBarycenter( $\langle a \rangle, \langle b \rangle, \langle c \rangle$ , etc.)` → point

$\langle a \rangle$ : point or pair;

$\langle b \rangle$ : point or pair;

etc.

We can calculate the barycenter of a list of points associated with weights (but here, it is not possible to use METAPOST pairs).

`Barycenter( $\langle A \rangle, \langle a \rangle, \langle B \rangle, \langle b \rangle, \langle C \rangle, \langle c \rangle$ , etc.)` → point

$\langle A \rangle$ : point ;

$\langle a \rangle$ : numeric, weight associated with  $\langle A \rangle$  ;

$\langle B \rangle$ : point ;

$\langle b \rangle$ : numeric, weight associated with  $\langle B \rangle$  ;

etc.

You can determine the bisector of an angular sector defined by three points. The function returns a droite.

`AngleBisector( $\langle A \rangle, \langle B \rangle, \langle C \rangle$ )` → droite

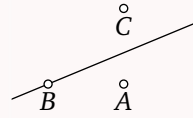
$\langle A \rangle$ : point ;

$\langle B \rangle$ : point ;

$\langle C \rangle$ : point.

## Example 2

```
1 input geom2d;
2 beginfig(1);
3 A = Point(1,0);
4 B = Point(0,0);
5 C = Point(1,1);
6 D = AngleBisector(A,B,C);
7 gddDraw D;
8 gddMark.bot "A";
9 gddMark.bot "B";
10 gddMark.bot "C";
11 Window(-0.5,-0.5,2,2);
12 endfig;
```



The `point` type is obviously related to the METAPOST `pair` type.  
`mp-geom2d` provides macros that allow converting from `point` to `pair` and vice versa.

`PointToPair(a,b)` → `pair`

*a*: `numeric`, x-coordinate;

*b*: `numeric`, y-coordinate.

`PairToPoint(p)` → `point`

*p*: `pair`.

These two commands are complemented by two others that ensure an element is of a given type:

`PairImp(p)` → `pair`

*p*: `point` or `pair`.

`PointImp(p)` → `point`

*p*: `point` or `pair`.

You can also retrieve a point along an `mp-geom2d` object (described in the following sections). The following macro returns a `point` along the path (cyclic or not) of the object, parameterized between 0 and 1.

`PointOf(o,t) → point`

`(o)`: any mp-geom2d object (even a `point` for which it returns itself);

`(t)`: `numeric` between 0 and 1 (which parameterizes the object's path between 0 and 1).

## 5.2 The `vecteur` type

This type corresponds to vectors defined using two coordinates in Euclidean space.

### 5.2.1 Constructors

The constructor function for such an object is:

```
1 vardef Vector(expr a,b) =
2   save n; n = incr gdd0;
3   gddT[n] = "vecteur"; gddA[n] = a; gddB[n] = b; n
4 enddef;
```

`Vector(a,b) → vecteur`

`(a)`: `numeric`;

`(b)`: `numeric`.

This function has the same structure as the corresponding one for the `point`: it returns the current value of `gdd0` after incrementing it, then assigns the type `vecteur` to the corresponding entry in the `gddT` table.

A vector can also be defined from an `pair` in METAPOST.

```
1 vardef VectorP(expr a) =
2   save n; n = incr gdd0;
3   gddT[n] = "vecteur"; gddA[n] = xpart a; gddB[n] = ypart a; n
4 enddef;
```

`VectorP(a) → vecteur`

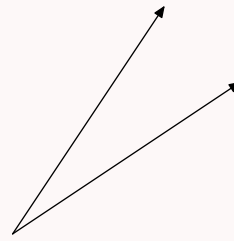
`(a)`: `pair`.

A vector can then be defined as follows:



### Exemple 3

```
1 input geom2d;  
2 beginfig(1);  
3 AB = Vector(2,3);  
4 pair a;  
5 a := (3,2);  
6 A = VectorP(a);  
7 gddDraw AB;  
8 gddDraw A;  
9 endfig;
```



#### 5.2.2 Associated Macros

Since mp-geom2d objects are **numeric**s, the classic operations of the vector space cannot be written with the simple characters +, -, and \*.

**VectorSum**( $\langle a \rangle, \langle b \rangle$ )  $\rightarrow$  vecteur

$\langle a \rangle$ : vecteur ;

$\langle b \rangle$ : vecteur.

**VectorSubtract**( $\langle a \rangle, \langle b \rangle$ )  $\rightarrow$  vecteur

$\langle a \rangle$ : vecteur ;

$\langle b \rangle$ : vecteur.

**ScalarVector**( $\langle k \rangle, \langle b \rangle$ )  $\rightarrow$  vecteur

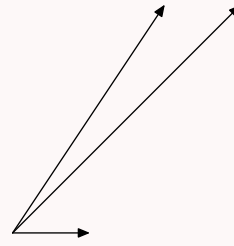
$\langle k \rangle$ : numeric ;

$\langle b \rangle$ : vecteur.

We can then perform the classic operations on vectors.

#### Exemple 4

```
1 input geom2d;  
2 beginfig(1);  
3 AB = Vector(2,3);  
4 BC = Vector(1,0);  
5 AC = VectorSum(AB,BC);  
6 kAC = ScalarVector(3.0,AC);  
7 gddDraw AB;  
8 gddDraw BC;  
9 gddDraw AC;  
10 endfig;
```



We can also calculate the dot product of two vectors with the following command.

`ScalarProduct( $\langle a \rangle$ ,  $\langle b \rangle$ )` → `numeric`

$\langle a \rangle$ : `vecteur` ;

$\langle b \rangle$ : `vecteur`.

We can also calculate the Euclidean norm of a vector.

`Norm( $\langle a \rangle$ )` → `numeric`

$\langle a \rangle$ : `vecteur`.

The following command returns the angle, in radians, between  $[0, \pi]$  formed between two vectors. If we denote  $u$  and  $v$  as the two vectors in  $\mathbf{R}^2$ , the angle calculated

$$\theta = \arccos\left(\frac{u \cdot v}{\|u\| \|v\|}\right).$$

`AngleBetweenVectors( $\langle a \rangle$ ,  $\langle b \rangle$ )` → `numeric` (dans  $[0, \pi]$ )

$\langle a \rangle$ : `vecteur` ;

$\langle b \rangle$ : `vecteur`.

### 5.3 The **segment** type

Segments are defined by two points in  $\mathbf{R}^2$ .

### 5.3.1 Constructor

The constructor function for this object is:

```
1 vardef Segment (expr a,b) =  
2   save n; n = incr gdd0;  
3   gddT[n] = "segment"; gddA[n] = PointImp(a); gddB[n] = PointImp(  
4     b); n  
5 enddef;
```

`Segment(a,b)` → `segment`

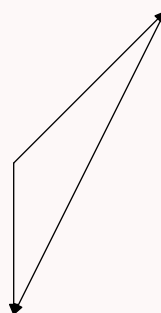
*a*): `point`;

*b*): `point`.

A segment is then defined as follows:

#### Example 5

```
1 input geom2d;  
2 beginfig(1);  
3 A = Point(2,3);  
4 B = Point(4,5);  
5 C = Point(2,1);  
6 AB = Segment(A,B);  
7 BC = Segment(B,C);  
8 AC = Segment(A,C);  
9 gddArrow AB;  
10 gddArrow BC;  
11 gddArrow AC;  
12 endfig;
```



### 5.3.2 Associated Macros

You can calculate the length of a segment with the following macro.

`SegmentLength(a)` → `numeric`

*a*): `segment`.

You can also convert a segment into a vector with the following macro, which calculates the difference in coordinates of the two points defining the segment.

`SegmentToVector(a)` → vecteur

*a*: segment.

## 5.4 The **droite** type

A line is simply defined by two points.

### 5.4.1 Constructor

The constructor function for this object is as follows:

```
1 vardef Line (expr a,b) =
2   save n; n = incr gdd0;
3   gddT[n] = "droite"; gddA[n] = PointImp(a); gddB[n] = PointImp(b
4   ); n
5 enddef;
```

`Line(a,b)` → droite

*a*: point;

*b*: point.

When representing lines (see section 4.3), the *infinite* nature of the line is managed by the global variable `gddExtensionDroite` which defaults to 10.

You can modify it with the following command:

`SetLineExtension(d)`

*d*: numeric (default value 10).

### 5.4.2 Associated Macros

The following macro allows, in the form of a triplet, and thus a **color**, to obtain the coefficients of a given line. For a line (*D*), the macro gives the triplet  $(u, v, w) \in \mathbf{R}^3$  such that  $\forall (x, y) \in (D), ux + vy + w = 0$ .

`LineEquation(a)` → color

`<a>: droite.`

You can calculate the projection of a point onto a line with the following command.

`PointOnLineProjection(<p>,<a>) → point`

`<p>: point ;`

`<a>: droite.`

You can obtain the relative distance of a point on a line with the following macro.

`PointLineRelativeDistance(<p>,<a>) → numeric`

`<p>: point ;`

`<a>: droite.`

The calculation of the distance from a point to a line is done with the following macro.

`PointLineDistance(<p>,<a>) → numeric`

`<p>: point ;`

`<a>: droite.`

The following macro allows obtaining the perpendicular line to a given line passing through a point.

`PerpendicularLine(<a>,<p>) → droite`

`<a>: droite ;`

`<p>: point.`

Similarly, the following macro allows obtaining the parallel line to another passing through a point.

`ParallelLine(<a>,<p>) → droite`

`<a>: droite ;`

$\langle p \rangle$ : point.

You can obtain the **point** of intersection of two lines with the following macro.

**LinesIntersection**( $\langle a \rangle, \langle b \rangle$ )  $\rightarrow$  point

$\langle a \rangle$ : droite ;

$\langle b \rangle$ : droite.

The following macro allows you to plot a point on a line with a certain length (signed), similar to a compass.

**ReportOnLine**( $\langle P \rangle, \langle D \rangle, \langle l \rangle$ )  $\rightarrow$  point

$\langle P \rangle$ : point ;

$\langle D \rangle$ : droite ;

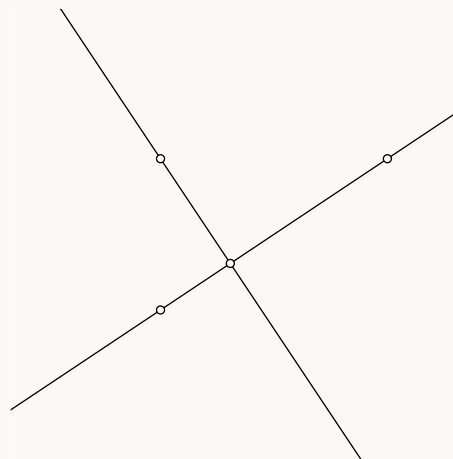
$\langle l \rangle$ : numeric (can be negative).

The direction of the plot relative to point  $\langle P \rangle$  depends on the definition of the line: for a line defined by two points  $A$  and  $B$ , the direction of the plot will follow the vector  $\vec{AB}$ . To change direction, simply multiply the distance parameter  $\langle l \rangle$  by  $-1$ .

Here is an example illustrating the use of some commands related to lines.

#### Exemple 6

```
1 input geom2d;
2 beginfig(1);
3 A = Point(0,0);
4 B = Point(3,2);
5 AB = Line(A,B);
6 gddDraw AB;
7 gddDrawPoint A;
8 gddDrawPoint B;
9 C = Point(0,2);
10 DC = PerpendicularLine(AB,C);
11 gddDraw DC;
12 gddDrawPoint C;
13 D = PointOnLineProjection(C,AB);
14 gddDrawPoint D;
15 Window(-2,-2,4,4);
16 endfig;
```



## 5.5 The **cercle** type

A circle could be defined by a point and a radius.

### 5.5.1 Constructors

The basic constructor function for this object is as follows:

```
1 vardef Circle (expr a,b) =  
2   save n; n = incr gdd0;  
3   gddT[n] = "cercle"; gddA[n] = PointImp(a); gddB[n] = b; n  
4   enddef;
```

**Circle**( $\langle a \rangle, \langle b \rangle$ )  $\rightarrow$  cercle

$\langle a \rangle$ : point or pair, center of the circle;

$\langle b \rangle$ : numeric, radius of the circle.

Two other functions allow defining circles. The **CircleCP** function defines the circle by a center and a point through which the circle passes.

**CircleCP**( $\langle a \rangle, \langle b \rangle$ )  $\rightarrow$  cercle

$\langle a \rangle$ : point or pair, center of the circle;

$\langle b \rangle$ : point, such that  $\langle b \rangle - \langle a \rangle$  is the radius.

We can also define a circle by two points defining its diameter. The corresponding function is as follows:

**CircleD**( $\langle a \rangle, \langle b \rangle$ )  $\rightarrow$  cercle

$\langle a \rangle$ : point or pair ;

$\langle b \rangle$ : point or pair, such that  $(\langle a \rangle - \langle b \rangle)$  is the diameter of the circle.

You can also define a circle as the circle passing through three points with the following macro.

**CircleThreePoints**( $\langle a \rangle, \langle b \rangle, \langle c \rangle$ )  $\rightarrow$  cercle

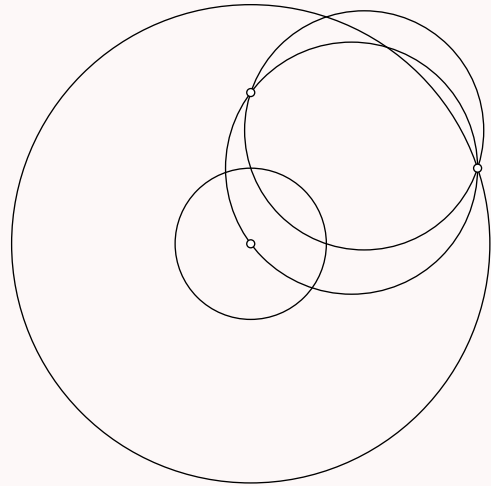
(a): point ou pair ;

(b): point ou pair ;

(c): point ou pair.

### Exemple 7

```
1 input geom2d;
2 beginfig(1);
3 A = Point(3,1);
4 B = Point(0,2);
5 O = Point(0,0);
6 r = 1.0;
7 COr = Circle(0,r);
8 COA = CircleCP(0,A);
9 CAB = CircleD(A,B);
10 COAB = CircleThreePoints(0,A,B);
11 gddDraw COr;
12 gddDraw COA;
13 gddDraw CAB;
14 gddDraw COAB;
15 gddDrawPoint A; gddDrawPoint B;
    gddDrawPoint O;
16 endfig;
```



### 5.5.2 Associated Macros

Many macros are associated with circles.

You can retrieve the radius of a circle with the following command.

**Radius(*a*)** → numeric

(*a*): cercle.

You can obtain the center (**point**) of a circle with the following command:

**Center(*a*)** → point

(*a*): cercle, ellipse or hyperbole.

We can compute the intersection between two circles using the following macro.



`CirclesIntersection(a,b)` → point

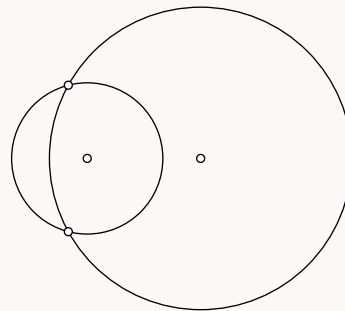
*a*: cercle;

*b*: cercle.

This macro will only give one intersection point. To obtain both intersection points, you will need to reverse the order of the circles in the call, as illustrated in the following example.

#### Exemple 8

```
1 input geom2d;
2 beginfig(1);
3 A = Point(0,0);
4 CA = Circle(A,1);
5 B = Point(1.5,0);
6 CB = Circle(B,2);
7 D1 = CirclesIntersection(CA,CB);
8 D2 = CirclesIntersection(CB,CA);
9 gddDraw CA; gddDraw CB;
10 gddDrawPoint A; gddDrawPoint B;
    gddDrawPoint D1; gddDrawPoint
    D2;
11 endfig;
```



If there is no intersection, the compilation will fail with the error ! Pythagorean subtraction X.XXXX+-+X.XXXX has been replaced by 0. If the two circles are coincident, the error will be ! Division by zero..

mp-geom2d also provides a macro to obtain the intersections between a line and a circle.

`LineCircleIntersection(d,c,n)` → point

*d*: line;

*c*: circle;

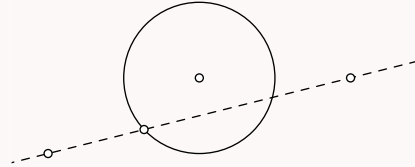
*n*: numeric which is 1 or 2 depending on the intersection point desired (if there is only one intersection point, both values return the same point).

If there is no intersection between the line and the circle, you will get the error ! Intersection between line and circle does not exist.

The following example illustrates this macro.

### Exemple 9

```
1 input geom2d;  
2 beginfig(1);  
3 A = Point(2,2);  
4 C_A = Circle(A,1);  
5 B = Point(0,1);  
6 C = Point(4,2);  
7 BC = Line(B,C);  
8 E1 = LineCircleIntersection(BC,  
   C_A,1);  
9 gddDraw C_A;  
10 gddDraw BC dashed evenly;  
11 gddDrawPoint A; gddDrawPoint B;  
   gddDrawPoint C; gddDrawPoint  
   E1;  
12 Window(-0.5,-0.5,5,4);  
13 endfig;
```



You can obtain the common internal and external tangents to two circles with the following two macros. Again, since there are two tangents, to obtain both, you will reverse the order of the two circles in the call.

**ExternalCommonTangent(*a*,*b*)** → droite

*a*): cercle;

*b*): cercle.

**InternalCommonTangent(*a*,*b*)** → droite

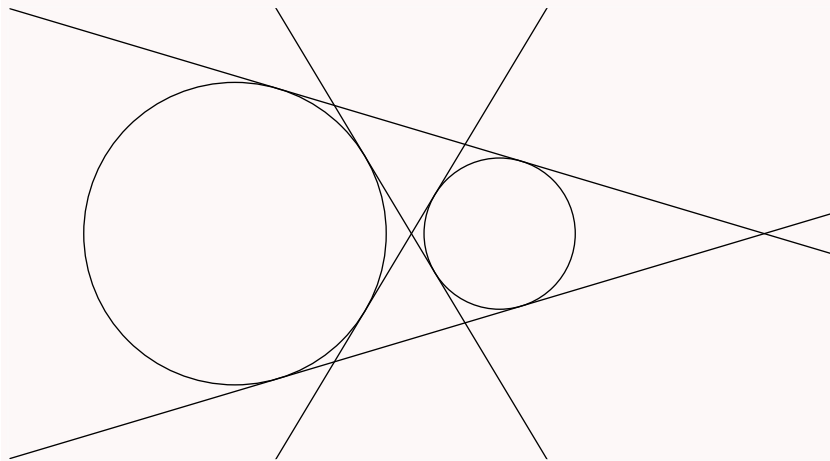
*a*): cercle;

*b*): cercle.

The following example illustrates the use of these two macros.

### Exemple 10

```
1 input geom2d;  
2 beginfig(1);  
3 C1 = Circle((-2,0),2);  
4 C2 = Circle((1.5,0),1);  
5 T1 = ExternalCommonTangent(C1,C2);  
6 T2 = ExternalCommonTangent(C2,C1);  
7 T3 = InternalCommonTangent(C1,C2);  
8 T4 = InternalCommonTangent(C2,C1);  
9 gddDraw C1; gddDraw C2;  
10 gddDraw T1; gddDraw T2;  
11 gddDraw T3; gddDraw T4;  
12  
13 Window(-5,-3,6,3);  
14 endfig;
```



The following macro calculates the radical axis (`droite`) of two circles<sup>5</sup>.

`RadicalAxis(a,b)` → `droite`

`a`: `cercle` ;

`b`: `cercle`.

The following macro calculates the radical center of three circles.

`RadicalCenter(a,b,c)` → `point`

`a`: `cercle` ;

<sup>5</sup>The construction method was largely inspired by T. Thurston's code in his document *Drawing with META-POST* [6].

```
(b): cercle;
```

```
(c): cercle.
```

The following macro calculates the axis of similarity of three circles.

```
AxisOfSimilitude(<a>,<b>,<c>) → droite
```

```
(a): cercle;
```

```
(b): cercle;
```

```
(c): cercle.
```

## 5.6 The **ellipse** type

### 5.6.1 Constructors

Ellipses can be defined in several ways. First, they can be defined with their center, one of the points on the major axis (called the vertex), and one of the points on the minor axis (called the co-vertex). However, when creating an ellipse, many attributes are calculated.

The constructor code is as follows:

```
1 vardef Ellipse(expr C,A,B) =
2   % C : centre
3   % A : vertex
4   % B : co-vertex
5   save n,a,b,c,e,_tmp,slope,_D,_K,_h,_ff;
6   pair _tmp;
7   n = incr gdd0;
8   gddT[n] = "ellipse"; gddA[n] = PointImp(C); gddB[n] = PointImp(
9     A);
10  gddC[n] = PointImp(B);
11  % calcul des deux foyers
12  a = Longueur(C,A);
13  b = Longueur(C,B);
14  c = sqrt(a**2-b**2);
15  e = c/a;
16  _tmp = e*(Pt(A)-Pt(C));
17  % les foyers
18  gddD[n] = PairTOPoint(Pt(C)+_tmp);
19  gddE[n] = PairTOPoint(Pt(C)-_tmp);
20  gddX[n][1] = a;
21  gddX[n][2] = b;
```

```

21 gddX[n][3] = e;
22 % angle du demi grand axe
23 slope = angle(PairImp(A)-PairImp(C));
24 gddX[n][4] = slope;
25 % directrices
26 _h = (a*a-c*c)/c;
27 % prejection sur la directrice
28 _ff = Droite(gddE[n],gddD[n]);
29 _K = ReportSurDroite(gddD[n],_ff,_h);
30 _D = DroitePerpendiculaire(_ff,_K);
31 gddX[n][5] := _D; % on stock la directrice
32 gddX[n][6] := SymetrieCentrale(_D,gddA[n]); % on stock la
    deuxième directrice
33 n
34 enddef;

```

`Ellipse(c,a,b)` → ellipse

*c*): point or pair, the center;

*a*): point or pair, the vertex;

*b*): point or pair, the co-vertex.

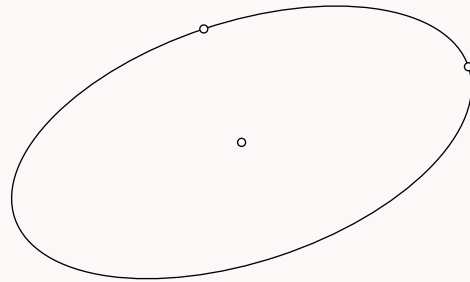
This macro is used as shown in the following example.

#### Exemple 11

```

1 input geom2d;
2 beginfig(1);
3 C = Point(0,0);
4 A = Point(3,1);
5 B = Rotation((1.5,0.5),Pi/2);
6 E = Ellipse(C,A,B);
7 gddDraw E;
8 gddDrawPoint A; gddDrawPoint B;
    gddDrawPoint C;
9 endfig;

```



An ellipse can be defined with its two foci and the semi-major axis using the following command:

`EllipseF(A,B,a)` → ellipse

*A*): point or pair, first focus;

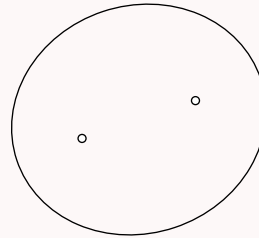
*B*): point or pair, second focus;

**(a): numeric**, the semi-major axis.

This macro is used as shown in the following example.

#### Example 12

```
1 input geom2d;  
2 beginfig(1);  
3 F1 = Point(3,1);  
4 F2 = Point(1.5,0.5);  
5 E = EllipseF(F1,F2,1.7);  
6 gddDraw E;  
7 gddDrawPoint F1; gddDrawPoint F2;  
8 endfig;
```



We can also define an ellipse from a focus, its directrix, and the eccentricity with the following command:

**EllipseFD(*F*,*D*),*e*)** → ellipse

**(F): point** or **pair**, a focus;

**(D): droite**, the associated directrix;

**(e): numeric**, the eccentricity (< 1).

#### 5.6.2 Associated Macros

You can obtain the center (**point**) of an ellipse with the following command:

**Center(*a*)** → point

**(a): cercle, ellipse** or **hyperbole**.

You can obtain the vertex and co-vertex with the following commands:

**Vertex(*a*)** → point

**(a): ellipse**.

`CoVertex(a)` → `point`

`<a>`: `ellipse`.

The following command allows obtaining the two foci based on the integer passed as an argument.

`Focus(a, n)` → `point`

`<a>`: `ellipse` ;

`<n>`: `numeric`, integer which is 1 or 2 to obtain each focus.

You can obtain the semi-major axis and the semi-minor axis with the following commands.

`SemiMajorAxis(a)` → `numeric`

`<a>`: `ellipse`.

`SemiMinorAxis(a)` → `numeric`

`<a>`: `ellipse`.

We can get the excentricity, denoted by  $e$ , with the following command.

`Excentricity(a)` → `numeric`

`<a>`: `ellipse`, `parabole` ou `hyperbole`.

To obtain the inclination (slope of the line passing through the foci of the ellipse), use the following command.

`Slope(a)` → `numeric`

`<a>`: ellipse, parabole, or hyperbole.

You can obtain the directrices of the ellipse with the following command:

`Directrix(<a>,<n>) → droite`

`<a>`: ellipse, parabole, or hyperbole ;

`<n>`: 1 or 2 (numeric), to choose one of the two directrices.

To obtain the tangent (`droite`) at a point on the ellipse, use the following command.

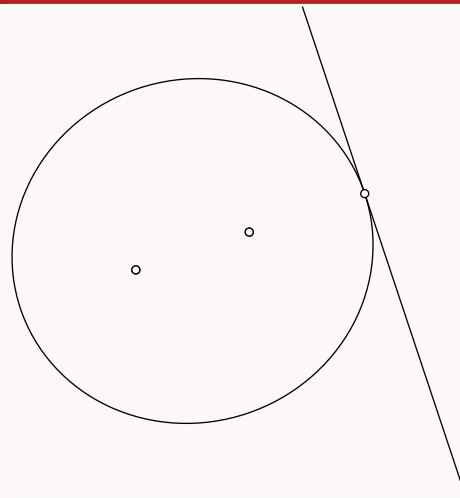
`EllipseTangent(<e>,<p>) → droite`

`<e>`: ellipse;

`<p>`: point or pair.

### Exemple 13

```
1 input geom2d;
2 beginfig(1);
3 F1 = Point(3,1);
4 F2 = Point(1.5,0.5);
5 E = EllipseF(F1,F2,2.4);
6 gddDrawPoints F1, F2;
7 gddDraw E;
8 M' = PointOf(E,0.5);
9 D = EllipseTangent(E,M');
10 gddDraw D;
11 gddDrawPoints M', Foyer(E,1),
    Foyer(E,2);
12 Window(-0.2,-2.5,6,4);
13 endfig;
```



To obtain the tangents (`droite`) passing through a point outside the ellipse, use the following command. If the chosen point is not outside the ellipse, a compilation error will occur.

`ExternalEllipseTangent(<e>,<p>,<n>) → droite`



(*e*): ellipse ;

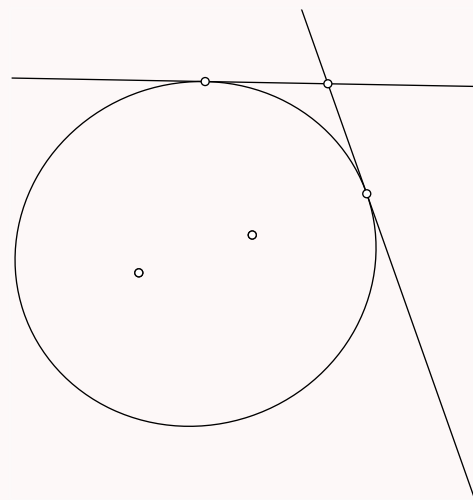
(*p*): point or pair ;

(*n*): numeric which is 1 or 2 to choose the tangent among the two possible ones.

The points of tangency can be accessed as the second point in the definition of the `droite` using the `gddB[]` table.

#### Exemple 14

```
1 input geom2d;
2 beginfig(1);
3 F1 = Point(3,1);
4 F2 = Point(1.5,0.5);
5 E = EllipseF(F1,F2,2.4);
6 gddDrawPoint F1; gddDrawPoint F2;
7 gddDraw E;
8 M = Point(4,3);
9 D1 = ExternalEllipseTangent(E,M
10 ,1);
11 gddDraw D1;
12 gddDrawPoint gddB[D1];
13 D2 = ExternalEllipseTangent(E,M
14 ,2);
15 gddDraw D2;
16 gddDrawPoint gddB[D2];
17 gddDrawPoint M;
18 gddDrawPoint Focus(E,1);
19 gddDrawPoint Focus(E,2);
20 Window(-0.2,-2.5,6,4);
21 endfig;
```



## 5.7 The **parabole** type

The basic constructor for the parabola defines this object using the focus and the directrix.

The constructor code is as follows:

```
1 vardef ParaboleFD(expr F,D) =
2   % F : foyer (point)
3   % D : directrice (droite)
4   save u, v, w, d, i, n,_tmp,slope;
5   pair _tmp;
6   n = incr gdd0;
7   (u,v,w) = EquationDroite(D);
8   % ordonnée relative
9   d := u * gddA[F] + v * gddB[F] + w;
10  gddT[n] := "parabole";
11  gddX[n][1] := D; % on stocke la directrice
```

```

12  gddX[n][2] := D; % on stocke la directrice (compatibilité avec
    hyperbole)
13  % sommet
14  _tmp := (((-d/2)*(u,v)) shifted PairImp(F));
15  gddB[n] = PointImp(_tmp);
16  gddC[n] = PointImp(_tmp);
17  % le foyer (doublé pour compatibilité)
18  gddD[n] := F;
19  gddE[n] := F;
20  gddX[n][3] := 1.0; % excentricité
21  % angle du demi-grand axe
22  slope = angle(PairImp(gddA[D])-PairImp(gddB[D]))+90;
23  gddX[n][4] = slope;
24  i := -gddC2Dparam-1;
25  gddP[n] := (
26    ((i*(v,-u)+((i*i-d*d)/(2d))*(u,v))
27    for i:= -gddC2Dparam upto gddC2Dparam:
28      ..(i*(v,-u)+((i*i-d*d)/(2d))*(u,v))
29    endfor)) shifted PairImp(F);
30  n
31  enddef;

```

**ParabolaFD**( $\langle f \rangle, \langle d \rangle$ )  $\rightarrow$  path

$\langle f \rangle$ : is the focus (**point**) of the parabola;

$\langle d \rangle$ : is the directrice (**droite**) of the parabola.

When representing a parabola, the *infinite* nature is managed by the global variable **gddC2Dparam**, which defaults to 15.

To modify it, use the following command:

**SetConicCoef**( $\langle n \rangle$ )

$\langle n \rangle$ : **numeric**, parameter for drawing hyperbolas and parabolas (default value 15).

### 5.7.1 Associated Functions

The eccentricity, often denoted as  $e$ , can be obtained using the following command:

**Excentricity**( $\langle a \rangle$ )  $\rightarrow$  numeric

$\langle a \rangle$ : **ellipse**, **parabole**, or **hyperbole**.

To obtain the slope (the slope of the axis of symmetry of the parabola), use the following command:

`Slope(a)` → **numeric**

*a*: **ellipse**, **parabole**, or **hyperbole**.

The vertex of the parabola can be obtained using the following command:

`ParabolaVertex(a,n)` → **point**

*a*: **parabole** or **hyperbole**;

*n*: 1 or 2 (**numeric**), an argument useful for the hyperbola which has two vertices. Here, the value of *n* does not matter, as the unique vertex will be returned.

The parabola's directrix can be obtained using the following command:

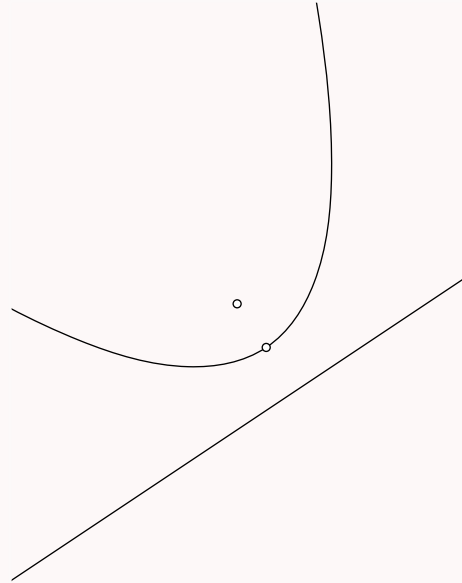
`Directrix(a,n)` → **droite**

*a*: **ellipse**, **parabole**, or **hyperbole**;

*n*: 1 or 2 (**numeric**), an argument useful for the hyperbola which has two directrices. Here, the value of *n* does not matter, as the unique directrix will be returned.

### Exemple 15

```
1 input geom2d;
2 beginfig(1);
3 A = Point(0,0);
4 B = Point(3,2);
5 AB = Line(A,B);
6 F = Point(-1,1);
7 Para = ParabolaFD(F,AB);
8 gddDraw AB;
9 gddDraw Para;
10 gddDrawPoint ParabolaVertex(Para
    ,1);
11 gddDrawPoint F;
12 Window(-4,-5,2,5);
13 endfig;
```



## 5.8 The hyperbole type

The basic constructor for the hyperbola defines this object using the focus and the directrix.

The constructor code is as follows:

```
1 vardef HyperboleFD(expr F,D,e) =
2   % F : foyer (point)
3   % D : directrice (droite)
4   % e : exentricité (numeric)
5   % pm : +1 ou -1 pour les deux branches
6   save u, v, w, d, i, n, _tmp, slope, aa, bb;
7   pair _tmp;
8   n = incr gdd0;
9
10  (u,v,w) = EquationDroite(D);
11  d := u * gddA[F] + v * gddB[F] + w;
12  gddT[n] := "hyperbole";
13
14  % sommets
15  _tmp := -(d+1*sqrt(e*e*d*d))/(e*e-1)*(u,v)shifted (
    ProduitScalaire(F,Vecteur(v,-u))*(v,-u));
16  gddB[n] := PointImp(_tmp);
```

```

17  _tmp := -(d-1*sqrt(e*e*d*d))/(e*e-1)*(u,v)shifted
18  (ProduitScalaire(F,Vecteur(v,-u))*(v,-u));
19  gddC[n] := PointImp(_tmp);
20  % centre comme milieu des deux sommets
21  gddA[n] = Milieu(gddB[n],gddC[n]);
22  % le foyer (doublé pour compatibilité)
23  gddD[n] := F;
24  gddE[n] := RotationCentre(F, gddA[n], Pi);
25  gddX[n][3] := e; % excentricité
26  % directrices
27  gddX[n][1] := D; % on stocke la directrice
28  gddX[n][2] := Droite(RotationCentre(gddA[D], gddA[n], Pi),
    RotationCentre(gddB[D], gddA[n], Pi));
29  % angle de l'axe
30  slope = angle(PairImp(gddA[D])-PairImp(gddB[D]))+90;
31  gddX[n][4] = slope;
32  % cercle principale
33  gddX[n][5] = CercleD(gddB[n],gddC[n]);
34  % asymptotes
35  aa = IntersectionDroiteCercle(D,gddX[n][5],1);
36  bb = IntersectionDroiteCercle(D,gddX[n][5],2);
37  gddX[n][6] = Droite(gddA[n],aa);
38  gddX[n][7] = Droite(gddA[n],bb);
39  % tracés des moitiés
40  i := -gddC2Dparam-1;
41  gddPX[n][1] := (
42  (
43  (i*(v,-u)-(d+sqrt(e*e*d*d+i*i*(e*e-1)))/(e*e-1)*(u,v))
44  for i:= -gddC2Dparam upto gddC2Dparam:
45  ..(i*(v,-u)-(d+sqrt(e*e*d*d+i*i*(e*e-1)))/(e*e-1)*(u,v))
46  endfor
47  ) shifted (ProduitScalaire(F,Vecteur(v,-u))*(v,-u))
48  );
49  gddPX[n][2] := (
50  (
51  (i*(v,-u)-(d-sqrt(e*e*d*d+i*i*(e*e-1)))/(e*e-1)*(u,v))
52  for i:= -gddC2Dparam upto gddC2Dparam:
53  ..(i*(v,-u)-(d-sqrt(e*e*d*d+i*i*(e*e-1)))/(e*e-1)*(u,v))
54  endfor
55  ) shifted (ProduitScalaire(F,Vecteur(v,-u))*(v,-u))
56  );
57  n
58  enddef;

```

In this constructor, we can see the use of the extended `path` table `gddPX[][]`, which allows associating multiple `paths` with a single object.

**HyperbolaFD**( $\langle f \rangle$ ,  $\langle d \rangle$ ,  $\langle e \rangle$ )  $\rightarrow$  hyperbole

$\langle f \rangle$ : is the focus (**point**) of the hyperbola;

$\langle d \rangle$ : is the directrix (**droite**) of the hyperbola;

$\langle e \rangle$ : is the eccentricity of the hyperbola.

You can also define a hyperbola using its two foci and the distance between a vertex and the center of the hyperbola (bifocal definition).

**HyperbolaF**( $\langle f1 \rangle$ ,  $\langle f2 \rangle$ ,  $\langle a \rangle$ )  $\rightarrow$  hyperbole

$\langle f1 \rangle$ : is one focus (**point**) of the hyperbola;

$\langle f2 \rangle$ : is the second focus (**point**) of the hyperbola;

$\langle a \rangle$ : is half the distance between the two vertices of the hyperbola.

### 5.8.1 Associated Functions

You can obtain the center (**point**) of a hyperbola with the following command:

**Center**( $\langle a \rangle$ )  $\rightarrow$  point

$\langle a \rangle$ : **cercle**, **ellipse**, or **hyperbole**.

The eccentricity, often denoted as  $e$ , can be obtained using the following command.

**Eccentricity**( $\langle a \rangle$ )  $\rightarrow$  numeric

$\langle a \rangle$ : **ellipse**, **parabole**, or **hyperbole**.

To obtain the inclination (slope of the line passing through the foci of the hyperbola), use the following command.

**Slope**( $\langle a \rangle$ )  $\rightarrow$  numeric

$\langle a \rangle$ : **ellipse**, **parabole**, or **hyperbole**.

The vertices of the hyperbola can be obtained using the following command:

`HyperbolaVertex( $\langle a \rangle, \langle n \rangle$ )` → `point`

$\langle a \rangle$ : `parabola` or `hyperbola`;

$\langle n \rangle$ : 1 or 2 (`numeric`), to choose one of the two vertices.

You can obtain the directrices of the hyperbola with the following command:

`Directrix( $\langle a \rangle, \langle n \rangle$ )` → `droite`

$\langle a \rangle$ : `ellipse`, `parabole`, or `hyperbole` ;

$\langle n \rangle$ : 1 or 2 (`numeric`), to choose one of the two directrices.

To obtain the *principal circle* of the hyperbola, you can use the following command:

`PrincipalCircle( $\langle h \rangle$ )` → `cercle`

$\langle h \rangle$ : `hyperbole`.

You can also obtain the two asymptotes of the hyperbola with the following command:

`HyperbolaAsymptote( $\langle h \rangle, \langle n \rangle$ )` → `droite`

$\langle h \rangle$ : `hyperbole` ;

$\langle n \rangle$ : 1 or 2 (`numeric`), to choose one of the two asymptotes.

Because a hyperbola consists of two disjoint parts, you cannot directly use the `gddDraw` command on the object as with other `mp-geom2d` objects. Instead, you must use the following command to obtain the `path` associated with one of the two parts of the hyperbola.

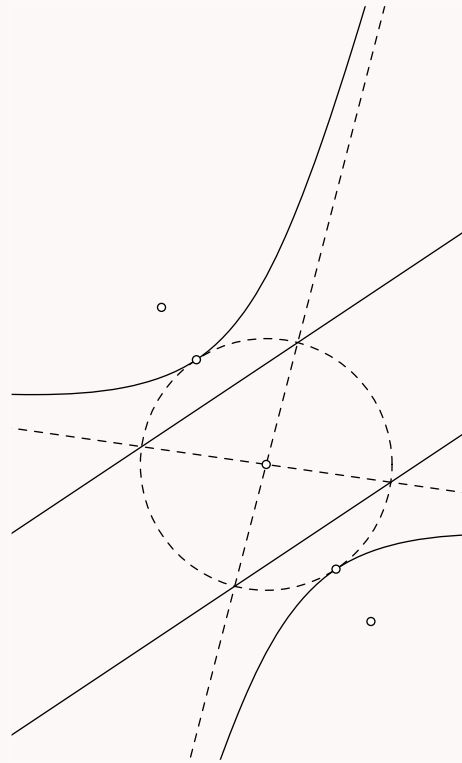
`SemiHyperbola( $\langle h \rangle, \langle n \rangle$ )` → `path`

$\langle h \rangle$ : `hyperbole`;

$\langle n \rangle$ : 1 or 2 (`numeric`), to choose one of the two parts.

### Exemple 16

```
1 input geom2d;
2 beginfig(1);
3 A = Point(0,0);
4 B = Point(3,2);
5 AB = Line(A,B);
6 F = Point(-1,1);
7 Hyper = HyperbolaFD(F,AB,1.5);
8 gddDraw AB;
9 gddDraw SemiHyperbola(Hyper,1);
10 gddDraw SemiHyperbola(Hyper,2);
11 gddDraw Directrix(Hyper,2);
12 gddDraw PrincipalCircle(Hyper)
    dashed evenly;
13 gddDraw HyperbolaAsymptote(Hyper
    ,1) dashed evenly;
14 gddDraw HyperbolaAsymptote(Hyper
    ,2) dashed evenly;
15 gddDrawPoint Center(Hyper);
16 gddDrawPoint Focus(Hyper,1);
17 gddDrawPoint Focus(Hyper,2);
18 gddDrawPoint HyperbolaVertex(
    Hyper,1);
19 gddDrawPoint HyperbolaVertex(
    Hyper,2);
20 Window(-3,-5,3,5);
21 endfig;
```



## 5.9 The **triangle** type

Triangles are defined by three points in  $\mathbb{R}^2$ .

### 5.9.1 Constructor

The constructor function for this object is as follows:

```
1 vardef Triangle (expr a,b,c) =
2   save n; n = incr gddO; gddT[n] = "triangle";
3   gddA[n] = PointImp(a); gddB[n] = PointImp(b); gddC[n] =
    PointImp(c); n
4   enddef;
```

**Triangle**( $\langle a \rangle, \langle b \rangle, \langle c \rangle$ )  $\rightarrow$  **numeric**

$\langle a \rangle$ : point;

$\langle b \rangle$ : point;

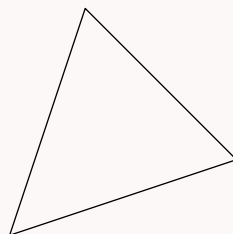
$\langle c \rangle$ : point.



Here, we see the use of the third table `gddC` to store the third point. A triangle is then defined as follows:

#### Example 17

```
1 input geom2d;  
2 beginfig(1);  
3 A = Point(3,1);  
4 B = Point(1,3);  
5 C = Point(0,0);  
6 ABC = Triangle(A,B,C);  
7 gddDraw ABC;  
8 endfig;
```



### 5.9.2 Associated Macros

The following macro calculates the area of a triangle.

`TriangleArea(a)` → `numeric`

*a*: `triangle`.

We can also calculate the orthocenter of a triangle using the following command.

`Orthocenter(a)` → `point`

*a*: `triangle`.

The following macro calculates the inscribed circle of a triangle.

`InscribedCircle(a)` → `cercle`

*a*: `triangle`.

You can also obtain the circumscribed circle with the following macro.

`CircumscribedCircle(a)` → `cercle`

*a*: `triangle`.

We can also calculate the escribed circles. Let  $A$ ,  $B$ , and  $C$  be three points defining a triangle  $ABC$ <sup>6</sup>. The following command allows obtaining one of the three escribed circles,

<sup>6</sup>Of course, the points can have different names.

as desired.

`EscribedCircle( $\langle a \rangle, \langle n \rangle$ )` → `cercle`

$\langle a \rangle$ : `triangle`;

$\langle n \rangle$ : `numeric` which can be 1, 2, or 3. If  $\langle n \rangle = 1$ , it is the escribed circle tangent to the side  $[BC]$  of the triangle; if  $\langle n \rangle = 2$ , it is the one tangent to the side  $[AC]$ ; and if  $\langle n \rangle = 3$ , it is the one tangent to the side  $[AB]$ .

You can also obtain the Euler circle of a triangle with the following command.

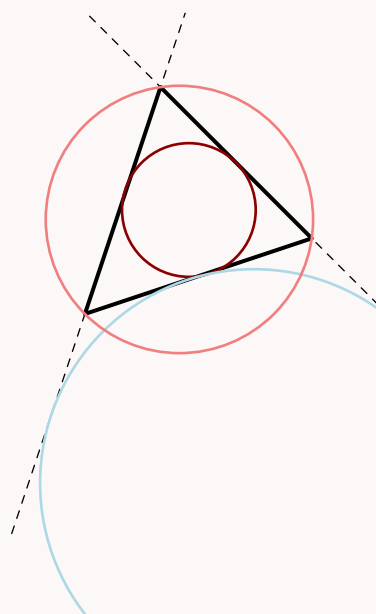
`EulerCircle( $\langle a \rangle$ )` → `cercle`

$\langle a \rangle$ : `triangle`.

Here is an example illustrating some of the macros related to triangles.

#### Exemple 18

```
1 input geom2d;
2 beginfig(1);
3 A = Point(3,1);
4 B = Point(1,3);
5 C = Point(0,0);
6 AB = Line(A,B);
7 BC = Line(B,C);
8 ABC = Triangle(A,B,C);
9 gddDraw AB dashed evenly;
10 gddDraw BC dashed evenly;
11 gddDraw ABC withPen(1.5,black);
12 Euler = EulerCircle(ABC);
13 gddDraw Euler withPen(1,DarkRed);
14 C_E1 = EscribedCircle(ABC,2);
15 gddDraw C_E1 withPen(1,LightBlue)
;
16 C_C = CircumscribedCircle(ABC);
17 gddDraw C_C withPen(1,LightCoral)
;
18 Window(-1,-4,4,4);
19 endfig;
```



## 5.10 The `polygon` type

Polygons are defined by  $N$  points in  $\mathbb{R}^2$ .

### 5.10.1 Constructors

The constructor function for this object is as follows:

```
1 vardef Polygon e(text plist) =
2   save n,_point,i; n = incr gdd0; gddT[n] = "polygone";
3   i:=1;
4   for _point = plist:
5     gddX[gdd0][i] = PointImp(_point);
6     i:=i+1;
7   endfor
8   gddA[n] = i-1; % number of sides
9   gddB[n] = IsoBarycenter(plist); % center
10  n
11 enddef;
```

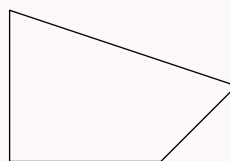
**Polygon**(*<list>*) → **polygone**

*<list>*: is a list of **point** or **pair**.

Here, we see the use of the *double* table **gddX** to store the *N* points.  
A polygon is then defined as follows:

#### Exemple 19

```
1 input geom2d;
2 beginfig(1);
3 A = Point(2,0);
4 B = Point(3,1);
5 C = Point(0,2);
6 D = Point(0,0);
7 P = Polygon(A,B,C,D);
8 gddDraw P;
9 endfig;
```



You can also construct regular polygons with the following command.

**RegularPolygon**(*<N>*,*<radius>*,*<rotation>*,*<translation>*) → **polygone**

*<N>*: integer (**numeric**) indicating the number of points;

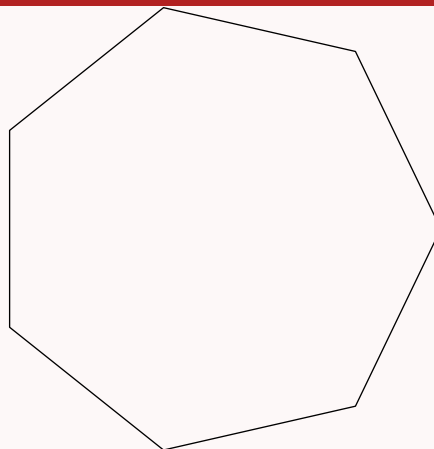
*<radius>*: **numeric**, radius of the circumscribed circle;

*<rotation>*: **numeric**, rotation of the polygon;

*<translation>*: (point) or **vecteur** (or even **pair**), displacement of the polygon's center.

## Exemple 20

```
1 input geom2d;  
2 beginfig(1);  
3 P = RegularPolygon(7,3,0,origine)  
4 ;  
5 gddDraw P;  
6 endfig;
```



### 5.10.2 Associated Macros

You can obtain the number of sides of a polygon with the following command.

`PolygonNumberOfSides(a)` → `numeric`

*a*: `polygone`.

You can access the different vertices of the polygon, numbered starting from 1, using the following macro.

`PolygonPoint(a,i)` → `point`

*a*: `polygone`;

*i*: `numeric`, an integer greater than or equal to 1, allowing access to the *i*<sup>th</sup> point of the polygon.

## 5.11 The `chemin` type

For this particular type (french translation of *path*), `mp-geom2d` stores the `path` in the reserved `gddP` table. Thus, the constructor function for this type of object is as follows:

```
1 vardef Chemin (expr p) =  
2 gddT[incr gddO] = "chemin"; gddP[gddO] = p; gddO  
3 enddef;
```

We defined the english macro `Path` for the english interface.

`Path(p)` → `chemin`

*p*: a `path` from METAPOST

### 5.11.1 Associated Macros

The following macro constructs, from a list of `points`, a polyline of type `chemin`.

`Polyline(list)` → `chemin`

*list*: is a list of `point` or `pair`.

## 5.12 The `courbe` type

For this particular type, `mp-geom2d` stores a string (a filename to be used with the `graph.mp` extension) in the reserved `gddS` table.

### 5.12.1 Constructor

The constructor function for this type of object is as follows:

```
1 vardef CourbeDat (expr s) =  
2   gddT[incr gdd0] = "courbe"; gddS[gdd0] = s; gdd0  
3 enddef;
```

The english interface uses the macro `CurveData`.

`CurveData(s)` → `courbe`

*s*: a filename in the form of a `string` in METAPOST

## 6 Circle Arc

With `mp-geom2d`, several commands allow working with circle arcs. First, you can choose to represent only a circle arc from a previously defined `cercle`. This is done using the following command, which takes as arguments a `cercle` and two angles in radians, and returns a `path` in METAPOST corresponding to the circle arc between the two given angles (angle 0 being parallel to the `Ox` axis).

`gddDrawCircleArc( $\langle C \rangle$ ,  $\langle a \rangle$ ,  $\langle b \rangle$ )` → path

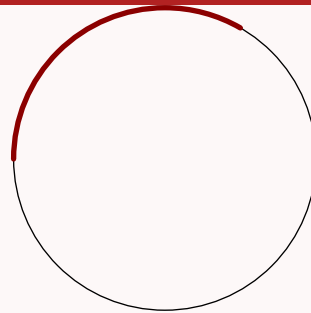
$\langle C \rangle$ : `circle` from which we want to represent an arc;

$\langle a \rangle$ : first angle in radians (`numeric`);

$\langle b \rangle$ : second angle in radians (`numeric`).

### Example 21

```
1 input geom2d;
2 beginfig(1);
3 C = Circle(origin,2);
4 gddDraw C;
5 gddDraw gddDrawCircleArc(C,Pi/3,
6     Pi) withPen(2,DarkRed);
7 endfig;
```



You can also define a `path` object. For this, you can use the following two commands. The first allows you to define a circular arc from a point, a radius, and two angles.

`Arc( $\langle P \rangle$ ,  $\langle r \rangle$ ,  $\langle a \rangle$ ,  $\langle b \rangle$ )` → path

$\langle P \rangle$ : `point` or `pair`, center of the circular arc;

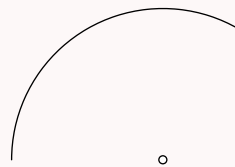
$\langle r \rangle$ : radius of the circular arc (`numeric`);

$\langle a \rangle$ : first angle in radians (`numeric`);

$\langle b \rangle$ : second angle in radians (`numeric`).

### Example 22

```
1 input geom2d;
2 beginfig(1);
3 A = Point(2,2);
4 Ac = Arc(A,2,Pi/3,Pi);
5 gddDrawPoint A;
6 gddDraw Ac;
7 endfig;
```



The second command defining a `chemin` is as follows. It takes as arguments three points, denoted as  $C$ ,  $A$ , and  $B$ : the center of the circular arc  $C$ , and the points  $A$  and  $B$ . It then takes as an argument the radius of the arc defined between the segments  $[C,A]$  and  $[C,B]$ .

`ArcBetweenPoints( $\langle P \rangle$ ,  $\langle r \rangle$ ,  $\langle A \rangle$ ,  $\langle B \rangle$ ,  $\langle s \rangle$ )`  $\rightarrow$  path

$\langle P \rangle$ : point or pair, center of the circular arc;

$\langle r \rangle$ : radius of the circular arc (numeric);

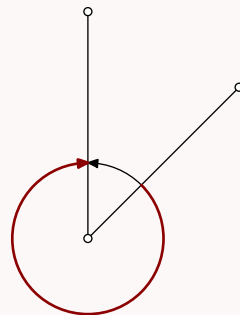
$\langle A \rangle$ : first point or pair;

$\langle B \rangle$ : second point or pair;

$\langle s \rangle$ : numeric, -1 or 1 depending on the chosen direction.

### Example 23

```
1 input geom2d;
2 beginfig(1);
3 C = Point(0,0);
4 A = Point(2,2);
5 B = Point(0,3);
6 Cab = ArcBetweenPoints(C,1,A,B,1)
7   ;
8 Cba = ArcBetweenPoints(C,1,A,B
9   ,-1);
10 gddDraw Segment(C,A);
11 gddDraw Segment(C,B);
12 gddDrawPoint A; gddDrawPoint B;
13   gddDrawPoint C;
14 gddArrow Cab;
15 gddArrow Cba withPen(1,DarkRed);
16 endfig;
```



## 7 Some Macros

This section groups together some macros that are not specific to a type.

The following macro allows calculating an intersection point between two mp-geom2d objects. It is essentially a wrapper around the METAPOST primitive `intersectionpoint` applied to the *drawn* objects (using `gddTraceObjjet`).

`PointIntersection( $\langle o \rangle$ ,  $\langle b \rangle$ )`  $\rightarrow$  point

$\langle o \rangle$ : a mp-geom2d object;

$\langle b \rangle$ : a mp-geom2d object.

## 8 Some Transformations

### 8.1 Homothety

You can perform a homothety on any mp-geom2d object using the following command.

**Homothety**( $\langle p \rangle, \langle o \rangle, \langle k \rangle$ )  $\rightarrow$  same type as  $\langle p \rangle$

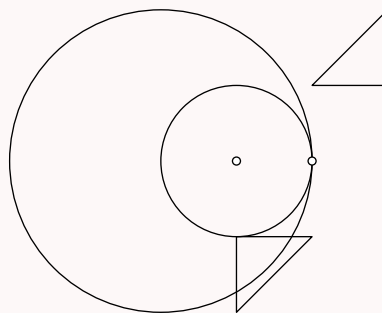
$\langle p \rangle$ : a mp-geom2d object;

$\langle o \rangle$ : **point** or **pair**, center of the homothety;

$\langle k \rangle$ : **numeric**, scaling factor of the homothety.

#### Exemple 24

```
1 input geom2d;
2 beginfig(1);
3 A = Point(1,2);
4 B = Point(0,2);
5 C_B = Circle(B,1);
6 T = Triangle((1,1),(0,0),(0,1));
7 C_H = Homothety(C_B, A, 2);
8 T_H = Homothety(T, A, -1);
9 gddDraw C_B; gddDraw C_H;
10 gddDraw T; gddDraw T_H;
11 gddDrawPoint A; gddDrawPoint B;
12 endfig;
```



### 8.2 Axial Symmetry

You can perform an axial symmetry on any mp-geom2d object using the following command.

**AxialSymmetry**( $\langle p \rangle, \langle d \rangle$ )  $\rightarrow$  same type as  $\langle p \rangle$

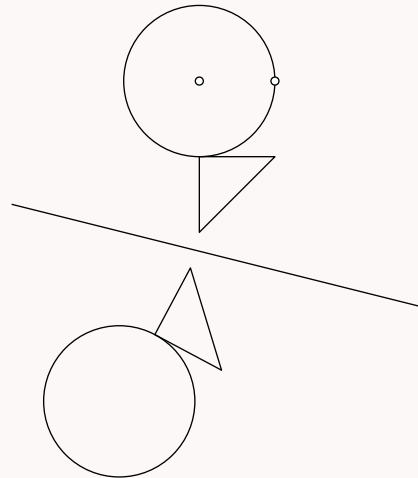
$\langle p \rangle$ : a mp-geom2d object;

$\langle d \rangle$ : **droite** defining the axis of symmetry.



### Exemple 25

```
1 input geom2d;
2 beginfig(1);
3 A = Point(1,2);
4 B = Point(0,2);
5 C_B = Circle(B,1);
6 T = Triangle((1,1),(0,0),(0,1));
7 D = Line((-1,0),(3,-1));
8 C_H = AxialSymmetry(C_B, D);
9 T_H = AxialSymmetry(T, D);
10 gddDraw C_B; gddDraw C_H;
11 gddDraw T; gddDraw T_H;
12 gddDraw D;
13 gddDrawPoint A; gddDrawPoint B;
14 Window(-2.5,-3.5,3,3.2);
15 endfig;
```



## 8.3 Central Symmetry

You can perform a central symmetry on any mp-geom2d object using the following command.

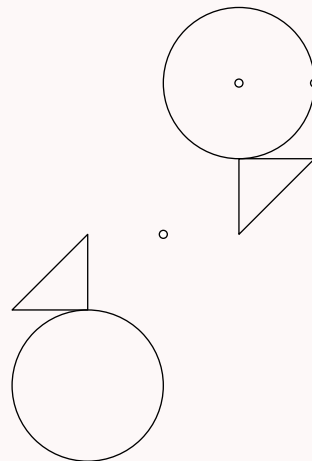
**CentralSymmetry**( $\langle p \rangle$ ,  $\langle d \rangle$ )  $\rightarrow$  same type as  $\langle p \rangle$

$\langle p \rangle$ : a mp-geom2d object;

$\langle d \rangle$ : **point** defining the center of symmetry.

### Exemple 26

```
1 input geom2d;
2 beginfig(1);
3 A = Point(1,2);
4 B = Point(0,2);
5 C_B = Circle(B,1);
6 T = Triangle((1,1),(0,0),(0,1));
7 D = Point(-1,0);
8 C_H = CentralSymmetry(C_B, D);
9 T_H = CentralSymmetry(T, D);
10 gddDraw C_B; gddDraw C_H;
11 gddDraw T; gddDraw T_H;
12 gddDrawPoint D;
13 gddDrawPoint A; gddDrawPoint B;
14 Window(-3.5,-3.5,3,3.2);
15 endfig;
```



## 8.4 Inversion

You can compute the inversion of a point, a circle, or a line with respect to a circle using the following macro:

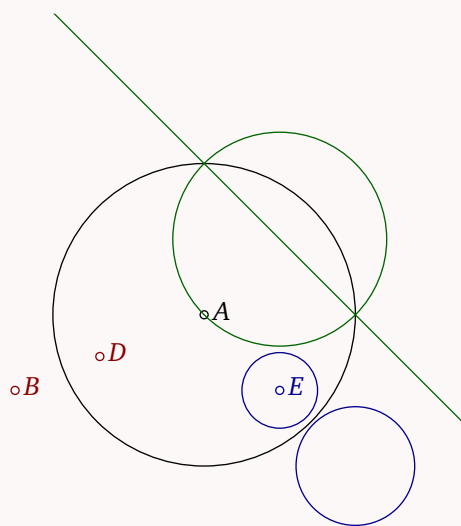
`Inverse( $\langle p \rangle, \langle c \rangle$ )`  $\rightarrow$  point

$\langle p \rangle$ : point, cercle, or droite;

$\langle c \rangle$ : cercle.

### Exemple 27

```
1 input geom2d;
2 beginfig(1);
3   A = Point(2,2);
4   C = Circle(A,2);
5   gddDraw C;
6   gddMark.rt "A";
7   B = Point(-0.5,1);
8   D = Inverse(B,C);
9   drawoptions(withcolor DarkRed);
10  gddMark.rt "B"; gddMark.rt "D";
11
12  drawoptions(withcolor DarkBlue)
13  ;
14  E = Point(3,1);
15  CE = Circle(E,0.5);
16  gddDraw CE;
17  gddMark.rt "E";
18  iCE = Inverse(CE,C);
19  gddDraw iCE;
20
21  drawoptions(withcolor DarkGreen
22  );
23  d = Line((3,3),(4,2));
24  gddDraw d;
25  Cd = Inverse(d,C);
26  gddDraw Cd;
27  Window(-1,-1,5.5,6);
28 endfig;
```



## 8.5 Similarity with center (direct)

One can compute a similarity centered at a point with a given ratio. This is done using the following macro.

`CenterSimilarity( $\langle p \rangle, \langle d \rangle, \langle a \rangle, \langle k \rangle$ )` → same type as  $\langle p \rangle$

$\langle p \rangle$ : a `mp-geom2d` object;

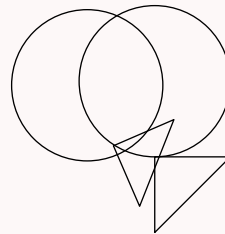
$\langle d \rangle$ : a `point` defining the center of the similarity;

$\langle a \rangle$ : angle (in radians, `numeric`) of the similarity;

$\langle k \rangle$ : similarity factor (`numeric`).

### Example 28

```
1 input geom2d;
2 beginfig(1);
3 A = Point(1,2);
4 B = Point(0,2);
5 C_B = Circle(B,1);
6 ratioSimi = 0.4;
7 angleSimi = Pi/3;
8 T = Triangle((1,1),(0,0),(0,1));
9 D = Point(-1,0);
10 C_H = CenterSimilarity(C_B, D,
11   angleSimi, ratioSimi);
11 T_H = CenterSimilarity(T, D,
12   angleSimi, ratioSimi);
12 gddDraw C_B; gddDraw C_H;
13 gddDraw T; gddDraw T_H;
14 endfig;
```



## 9 Annotations and Labels

### 9.1 Right Angle Marks

`mp-geom2d` provides the following macro to mark a right angle formed by three points:

`OrthoSign( $\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle x \rangle$ )`

$\langle a \rangle, \langle b \rangle, \langle c \rangle$ : are the three `points` forming the right angle  $\widehat{ABC}$ ;

$\langle x \rangle$ : is the “size” (`numeric`) of the right angle mark.

You can also create marks between two points or on a segment, a vector, or any type of object. For this, the first macro is as follows:

`Mark( $\langle a \rangle, \langle b \rangle, \langle n \rangle$ )`

- (a): first **point** forming the segment to be marked;
- (b): second **point** forming the segment to be marked;
- (n): is the type (**numeric**) of mark, there are four  $\langle n \rangle = 1, 2, 3, \text{ or } 4$ .

You can also mark any type of line with the following macro:

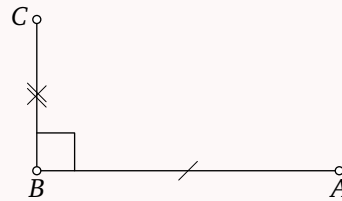
### LineMark(*a*,*n*)

- (a): can be any mp-geom2d object or even a **path** in METAPOST;
- (n): is the type (**numeric**) of mark, there are four types:  $\langle n \rangle = 1, 2, 3, \text{ or } 4$ .

#### Example 29

```

1 input geom2d;
2 beginfig(1);
3 A = Point(4,0);
4 B = Point(0,0);
5 C = Point(0,2);
6 gddDraw Segment(B,A);
7 gddDraw Segment(B,C);
8 gddMark.bot "A";
9 gddMark.bot "B";
10 gddMark.lft "C";
11 gddDraw OrthoSign(A,B,C,0.5);
12 gddDraw Mark(A,B,1);
13 gddDraw LineMark(Segment(B,C),3);
14 endfig;
```



## 9.2 Labels

Since the code produced with mp-geom2d is METAPOST code [5], we can use the classic labeling tools. However, to allow more flexibility (and particularly compatibility with Lua<sub>TEX</sub> and **luamplib** [1] or **minim-mp** [4]), if the code is compiled with METAPOST, then the package **latexmp** [2] is loaded and provides the `textext()` command.

Additionally, mp-geom2d provides some commands to make life easier.

Points can be marked using the following command. This command *marks* the point (using the `gddDrawPoint` command, see section 4.3) and adds a label.

`gddMark.<place> "<name>"`

(*place*): can be the classic placements of METAPOST: **top**, **bot**, **rt**, **lft**, **urt**, **ulft**, **lrt**, **llft**.

(*name*): enclosed in double quotes, must be the name of a point variable. The name will be

typeset in math mode (enclosed in  $\$$ ). If the name contains an `_`, everything after it will be subscripted (e.g., `A_be` will become  $A_{be}$ ). If the name is `alpha`, `beta`, `gamma`, or `delta`, the result will be the corresponding Greek letter typeset in math mode.

`mp-geom2d` also provides an adaptation to the classic `label` of METAPOST.

`gddLabel.<place>(<content>,<point>)`

**<place>**: can be the classic placements of METAPOST: `top`, `bot`, `rt`, `lft`, `urt`, `ulft`, `lrt`, `llft`.

**<content>**: typically what is given to `label`, a string, or a `picture` (which can be produced, for example, with `btex ... etex` or, since `latexmp` is loaded by `mp-geom2d`, `textext()`).

**<point>**: must be a `point` from `mp-geom2d`.

You can also label a `chemin`, a `courbe`, or a `path` using the following macro:

`PathTag.<place>(<content>,<path>,<position>)`

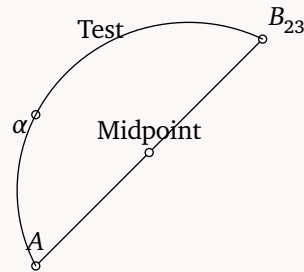
**<place>**: can be the classic placements of METAPOST: `top`, `bot`, `rt`, `lft`, `urt`, `ulft`, `lrt`, `llft`.

**<content>**: typically what is given to `label`, a string, or a `picture` (which can be produced, for example, with `btex ... etex` or, since `latexmp` is loaded by `mp-geom2d`, `textext()`).

**<path>**: must be a `chemin`, a `courbe`, or a `path`.

### Exemple 30

```
1 input geom2d;
2 beginfig(1);
3 A = Point(0,0);
4 B_23 = Point(3,3);
5 alpha = Point(0,2);
6 gddDraw Segment(A,B_23);
7 C = MidPoint(A,B_23);
8 gddDrawPoint C;
9 gddLabel.top(texttext("Midpoint"),
10 C);
11 gddMark.top "A";
12 gddMark.urt "B_23";
13 gddMark.llft "alpha";
14 P = Path(Pt(A)..Pt(alpha)..Pt(B_
15 23));
16 gddDraw P;
17 PathTag.top("Test",P,0.6);
18 endfig;
```



## 10 Coordinate System

mp-geom2d provides a set of commands to facilitate the drawing of coordinate systems.

The main command is the definition of the coordinate system, that is, the box in which the drawing will be represented.

**Frame**( $\langle l \rangle, \langle h \rangle, \langle ox \rangle, \langle oy \rangle, \langle ux \rangle, \langle uy \rangle$ )

$\langle l \rangle$ : **numeric**, width of the coordinate system (in **gddU** units);

$\langle h \rangle$ : **numeric**, height of the coordinate system (in **gddU** units);

$\langle ox \rangle$ : **numeric**, distance (in **gddU** units) from the origin (**point** (0,0)) to the left edge;

$\langle oy \rangle$ : **numeric**, distance (in **gddU** units) from the origin (**point** (0,0)) to the bottom edge;

$\langle ux \rangle$ : **numeric**, unit of the ( $Ox$ ) axis (in **gddU** units);

$\langle uy \rangle$ : **numeric**, unit of the ( $Oy$ ) axis (in **gddU** units).

This command does not return or draw anything. It serves to specify some internal variables for drawing definition. It also modifies the behavior of the **gddInPlace** macro (see page 5). Note that this command requires that the origin (i.e., the **point** (0,0)) is inside the coordinate system.

This macro is accompanied by two others, also *silent*, which serve only to:

- save the current **picture**;
- build a **picture** with the content found between **gddBegin** and **gddEnd**;

- crop (with `clip`) the `picture` to the frame of the constructed coordinate system;
- add the current `picture` to the saved one;
- finally, restore the behavior of `gddInPlace` as it was before using `Frame`.

These two commands are called `gddBegin` and `gddEnd`.

`gddBegin`

`gddEnd`

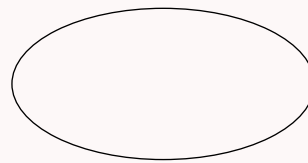
Thus, it is very easy to illustrate that, up to a change of orthogonal basis, a circle is an ellipse.

#### Exemple 31

```

1 input geom2d;
2 beginfig(1);
3 A = Point(1,1);
4 CA = Circle(A,2);
5 Frame(5,5,1,1,1,0.5);
6 gddBegin;
7 gddDraw CA;
8 gddEnd;
9 endfig;

```



There are two commands available to draw the axes of the coordinate system. The first draws the axes passing through the origin.

`Axis`

*(draws a set of elements on the coordinate system)*

This command also adds the labels for the x-axis and y-axis, which are stored in the following dedicated global variables:

`gddXlabel`

*(string, default value "\$x\$")*

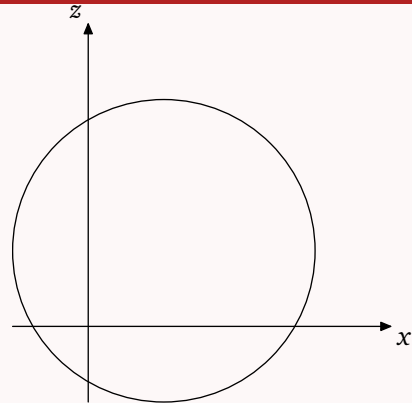
`gddYlabel`

*(string, default value "\$y\$")*

Note, these commands must be used before calling `gddBegin`.  
The following example illustrates how to draw the axes.

### Exemple 32

```
1 input geom2d;  
2 beginfig(1);  
3 A = Point(1,1);  
4 CA = Circle(A,2);  
5 Frame(5,5,1,1,1,1);  
6 gddYlabel:="$z$";  
7 Axis;  
8 gddBegin;  
9 gddDraw CA;  
10 gddEnd;  
11 endfig;
```



You can also define a coordinate system using a syntax that allows you to specify the minimum and maximum abscissas and ordinates.

**FrameMinMax**(*<xmin>*, *<xmax>*, *<ymin>*, *<ymax>*, *<ux>*, *<uy>*)

*<xmin>*: **numeric**, minimum abscissa (in **gddU** units).

*<xmax>*: **numeric**, maximum abscissa (in **gddU** units).

*<ymin>*: **numeric**, minimum ordinate (in **gddU** units).

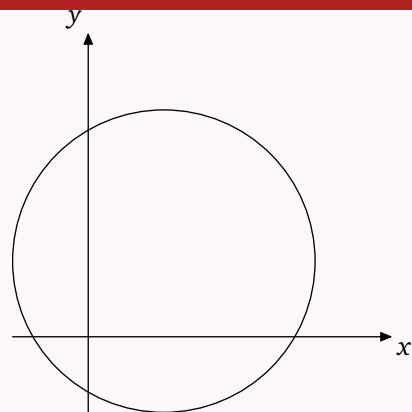
*<ymax>*: **numeric**, maximum ordinate (in **gddU** units).

*<ux>*: **numeric**, unit of the (*Ox*) axis (in **gddU** units).

*<uy>*: **numeric**, unit of the (*Oy*) axis (in **gddU** units).

### Exemple 33

```
1 input geom2d;  
2 beginfig(1);  
3 A = Point(1,1);  
4 CA = Circle(A,2);  
5 FrameMinMax(-1,4,-1,4,1,1);  
6 Axis;  
7 gddBegin;  
8 gddDraw CA;  
9 gddEnd;  
10 endfig;
```



`mp-geom2d` also provides a way to draw axes on the edge of the frame with the following command.

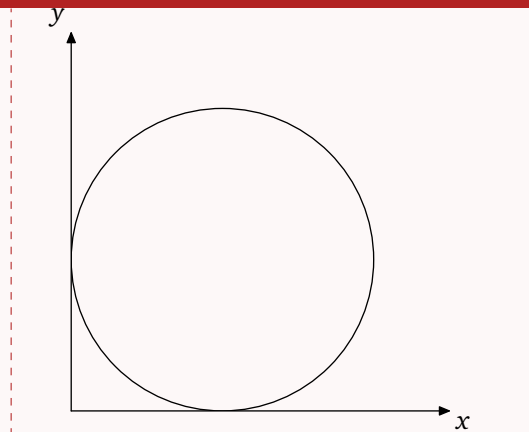


## SideAxis

(draws a set of elements on the coordinate system)

### Exemple 34

```
1 input geom2d;  
2 beginfig(1);  
3 A = Point(1,1);  
4 CA = Circle(A,2);  
5 FrameMinMax(-1,4,-1,4,1,1);  
6 SideAxis;  
7 gddBegin;  
8 gddDraw CA;  
9 gddEnd;  
10 endfig;
```



The following commands allow you to add graduations to the axes (either classic or on the edge). Again, these commands take no arguments, return nothing, and simply draw.

## Graduations

(draws a set of elements on the coordinate system)

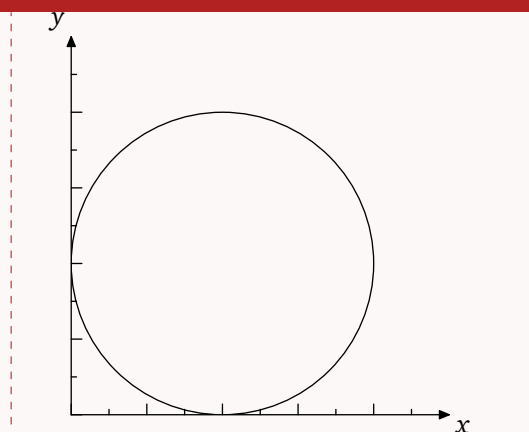
## SideGraduations

(draws a set of elements on the coordinate system)

Use the version consistent with the chosen axes.

### Exemple 35

```
1 input geom2d;  
2 beginfig(1);  
3 A = Point(1,1);  
4 CA = Circle(A,2);  
5 FrameMinMax(-1,4,-1,4,1,1);  
6 SideAxis;  
7 SideGraduations;  
8 gddBegin;  
9 gddDraw CA;  
10 gddEnd;  
11 endfig;
```



If you want to mark the units, mp-geom2d provides the following macro (which can only be used if the axes are not placed on the border).

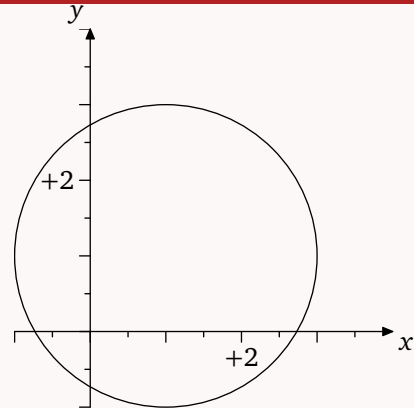
**Units(*unit*)**

(draws a set of elements on the coordinate system)

(*unit*): *numeric*, value to be displayed on both axes.

### Exemple 36

```
1 input geom2d;
2 beginfig(1);
3 A = Point(1,1);
4 CA = Circle(A,2);
5 FrameMinMax(-1,4,-1,4,1,1);
6 Axis;
7 gddBegin;
8 Graduations;
9 Units(2);
10 gddDraw CA;
11 gddEnd;
12 endfig;
```



You can also add a grid to your coordinate system with the following macro.

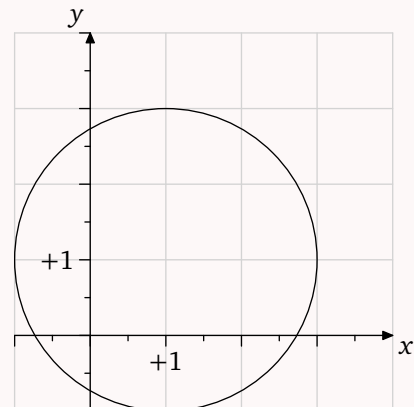
**FrameGrid**

(draws a set of elements on the coordinate system)

In the following example, the color of the grid is set using the MetaPost **drawoptions** command.

### Exemple 37

```
1 input geom2d;
2 beginfig(1);
3 A = Point(1,1);
4 CA = Circle(A,2);
5 FrameMinMax(-1,4,-1,4,1,1);
6 drawoptions(withcolor LightGrey);
7 FrameGrid;
8 drawoptions();
9 Axis;
10 gddBegin;
11 Graduations;
12 Units(1);
13 gddDraw CA;
14 gddEnd;
15 endfig;
```



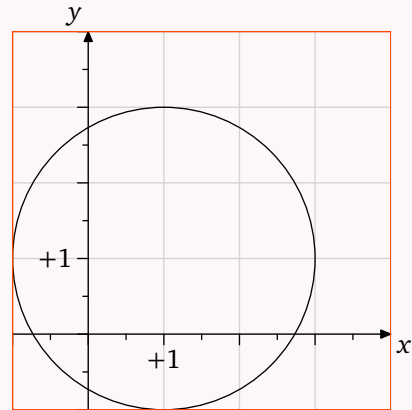
You can also add a frame to the coordinate system with the following macro.

FrameBox → path

In the following example, the color of the grid is set using the MetaPost `drawoptions` command.

### Exemple 38

```
1 input geom2d;
2 beginfig(1);
3 A = Point(1,1);
4 CA = Circle(A,2);
5 FrameMinMax(-1,4,-1,4,1,1);
6 drawoptions(withcolor LightGrey);
7 FrameGrid;
8 drawoptions();
9 Axis;
10 gddBegin;
11 Graduations;
12 Units(1);
13 gddDraw CA;
14 gddDraw FrameBox withcolor
    OrangeRed;
15 gddEnd;
16 endfig;
```



## 11 Some Mathematical Constants and Functions

`mp-geom2d` defines two mathematical constants, `Pi` and `_E`, for the constants  $\pi \approx 3.14159265$  and  $e = 2.71828183$ .

In addition, the package defines some mathematical functions of a real variable:

`sin(x)`

`cos(x)`

`tan(x)`

`exp(x)`

`ln(x)`

`cosh(x)`

`sinh(x)`

`tanh(x)`

`arcsin( $\langle x \rangle$ )`

`arccos( $\langle x \rangle$ )`

`arctan( $\langle x \rangle$ )`

## 12 Representation of Curves and Functions

`mp-geom2d` also provides some macros to facilitate the simple representation of curves and mathematical functions.

### 12.1 Function of a Real Variable

To represent a function of a real variable, use the following macro:

`Plot( $\langle function \rangle$ )( $\langle ti \rangle$ ,  $\langle tf \rangle$ ,  $\langle n \rangle$ )  $\rightarrow$  path`

**$\langle function \rangle$** : is a METAPOST macro that defines the mathematical function of a real variable to be represented;

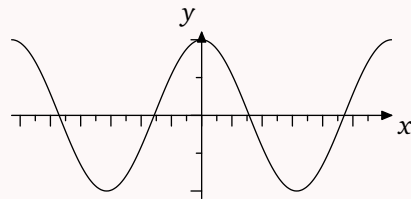
**$\langle ti \rangle$** : is the initial value (**numeric**) of the variable from which to start building the representation of the function;

**$\langle tf \rangle$** : is the final value (**numeric**) of the variable up to which to build the representation of the function;

**$\langle n \rangle$** : is the number of steps (**numeric**) used for the discretization of the representation.

#### Example 39

```
1 input geom2d;
2 beginfig(1);
3 FrameMinMax(-2Pi,2Pi
4             , -1.1,1.1,0.4,1);
5 Axis;
6 gddBegin;
7   Graduations;
8   gddDraw Plot(cos,-2Pi,2Pi,100);
9 gddEnd;
```



### 12.2 Parametric Curve

To represent a plane curve defined by two functions describing the abscissa and ordinate as functions of a parameter, use the following macro:

**Curve**(*abscissa\_function*)(*ordinate\_function*)(*ti*),(*tf*),(*n*) → path

**(abscissa\_function)**: is a METAPOST macro that defines the mathematical function of a real variable describing the evolution of the abscissa of the points of the curve to be represented;

**(ordinate\_function)**: is a METAPOST macro that defines the mathematical function of a real variable describing the evolution of the ordinate of the points of the curve to be represented;

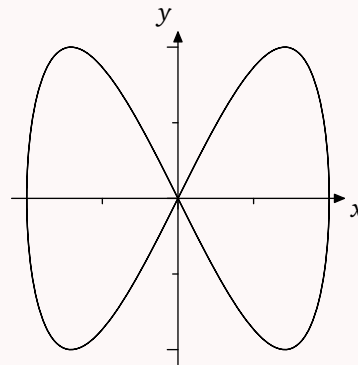
**(ti)**: is the initial value (**numeric**) of the variable from which to start building the representation of the function;

**(tf)**: is the final value (**numeric**) of the variable up to which to build the representation of the function;

**(n)**: is the number of steps (**numeric**) used for the discretization of the representation.

#### Example 40

```
1 input geom2d;
2 vardef f_x(expr t)= cos(t) enddef
3   ;
4   vardef f_y(expr t)= sin(2*t)
5     enddef;
6 beginfig(1);
7   FrameMinMax
8     (-1.1,1.1,-1.1,1.1,2,2);
9   Axis;
10  gddBegin;
11  Graduations;
12  gddDraw Curve(f_x,f_y,-2Pi,2Pi
13    ,100);
14  gddEnd;
15  endfig;
```



To represent a plane curve defined by a function describing the polar coordinates as a function of a parameter, use the following macro:

**PolarCurve**(*function*)(*ti*),(*tf*),(*n*) → path

**(function)**: is a METAPOST macro that defines the mathematical function of a real variable describing the evolution of the polar coordinates of the points of the curve to be represented;

**(ti)**: is the initial value (**numeric**) of the variable from which to start building the representation of the function;

**(tf)**: is the final value (**numeric**) of the variable up to which to build the representation of the function;

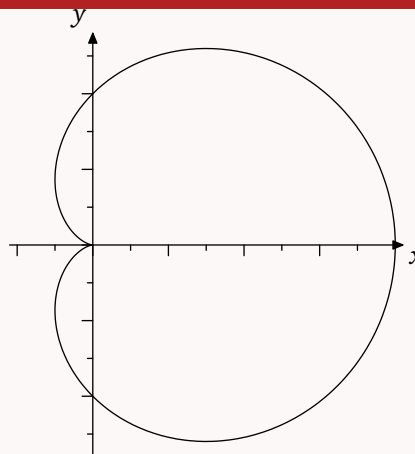
**(n)**: is the number of steps (**numeric**) used for the discretization of the representation.

#### Example 41

```

1 input geom2d;
2 a := 2;
3 vardef cardioid(expr t)= a*(1+cos
  (t)) enddef;
4 beginfig(1);
5 FrameMinMax
  (-1.1,4.1,-2.8,2.8,1,1);
6 Axis;
7 gddBegin;
8 Graduations;
9 gddDraw PolarCurve(cardioid,0,2
  Pi,100);
10 gddEnd;
11 endfig;

```



### 12.3 Vector Fields

mp-geom2d provides macros for representing vector fields.

First, you can plot vector fields associated with a first-order differential equation for a function  $y$  of the variable  $x$ :

$$y' = F(x, y).$$

The associated macro is as follows:

**VectorField**(*function*)(*x*), (*y*), (*px*), (*py*), (*dx*), (*color*) → *path*

**(function)**: is a METAPOST macro that defines a mathematical function from  $\mathbf{R}^2$  to  $\mathbf{R}$ ;

**(x)**: is a value (**numeric**), in **gddU** units, which allows shifting the grid of vectors along the  $x$  direction, with points at all  $\langle x \rangle + i\langle px \rangle$  for integer  $i$ ;

**(y)**: is a value (**numeric**), in **gddU** units, which allows shifting the grid of vectors along the  $y$  direction, with points at all  $\langle y \rangle + i\langle py \rangle$  for integer  $i$ ;

**(px)**: is the value (**numeric**), in **gddU** units, of the grid step along the  $x$ -axis for the representation of vectors;

**(py)**: is the value (**numeric**), in **gddU** units, of the grid step along the  $y$ -axis for the representation of vectors;

**(dx)**: is the norm (**numeric**), in **gddU** units, of the vectors in the vector field;

**(color)**: is the (**color**) used to draw the vectors.

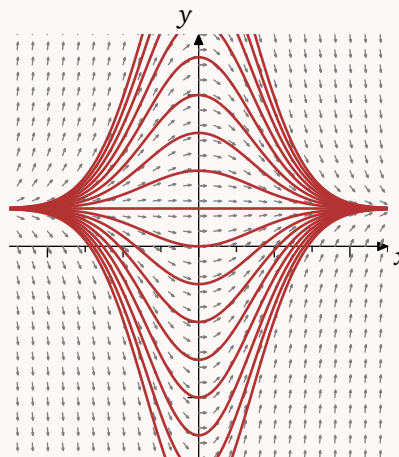
Since this macro uses the `drawarrow` macro from METAPOST, you will need to adjust the `ahlength` parameter to set the size of the arrowhead triangle (see [5]).

#### Example 42

```

1 input geom2d;
2 vardef F(expr x,y) = x - 2 * x *
  y enddef;
3 vardef f(expr x) = 1/2 + a * exp
  (- x*x) enddef;
4 beginfig(1);
5 FrameMinMax
  (-2.5,2.5,-2.8,2.8,1,1);
6 Axis;
7 gddBegin;
8 Graduations;
9 ahlength := 1;
10 VectorField(F,0,0,0.2,0.2,0.1,0.5
  white);
11 % Integral curves
12 for n = 0 upto 16:
13   a := (n/2) - 4;
14   gddDraw Plot(f,-2.5,2.5,50)
15   withPen(1,(0.7,0.2,0.2));
16 endfor
17 gddEnd;
18 endfig;

```



Similarly, you can plot vector fields of a function from  $\mathbf{R}^2$  to  $\mathbf{R}^2$ . The associated macro is as follows:

**VectorFieldDD**(*function*)(*x*),(*y*),(*px*),(*py*),(*dx*),(*color*) → *path*

**(function)**: is a METAPOST macro that defines a mathematical function from  $\mathbf{R}^2$  to  $\mathbf{R}^2$  (thus returning a **pair**);

**(x)**: is a value (**numeric**), in **gddU** units, which shifts the grid of vectors along the *x* direction, with points at all  $\langle x \rangle + i\langle px \rangle$  for integer *i*;

**(y)**: is a value (**numeric**), in **gddU** units, which shifts the grid of vectors along the *y* direction, with points at all  $\langle y \rangle + i\langle py \rangle$  for integer *i*;

**(px)**: is the value (**numeric**), in **gddU** units, of the grid step along the *x*-axis for the representation of vectors;

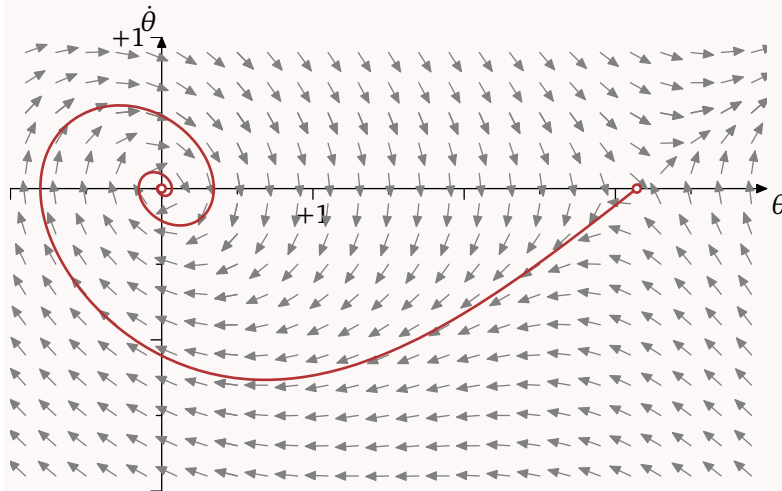
**(py)**: is the value (**numeric**), in **gddU** units, of the grid step along the *y*-axis for the representation of vectors;

**(dx)**: is the norm (**numeric**), in **gddU** units, of the vectors in the vector field;

**(color)**: is the (**color**) used to draw the vectors.

### Exemple 43

```
1 input geom2d;
2 gddXlabel := "\theta";
3 gddYlabel := "\dot{\theta}";
4 numeric b,c;
5 b:=0.5;
6 c=1.0;
7
8 vardef F(expr x,y) = (y,-b*y-c*sin(x)) enddef;
9
10 beginfig(1);
11 Frame(10,6,2,4,2,2);
12 Axis;
13 Units(1);
14 gddBegin;
15 Graduations;
16 trajectory := CurveData("solution0",0);
17 VectorFieldDD(F,0.5,0.5,0.2,0.2,0.15,0.5white);
18 gddDraw trajectory withPen(1,(0.7,0.2,0.2));
19 gddDrawPoint Point(0,0) withPen(1,(0.7,0.2,0.2));
20 gddDrawPoint Point(3.1415,0) withPen(1,(0.7,0.2,0.2));
21 gddEnd;
22 endfig;
```





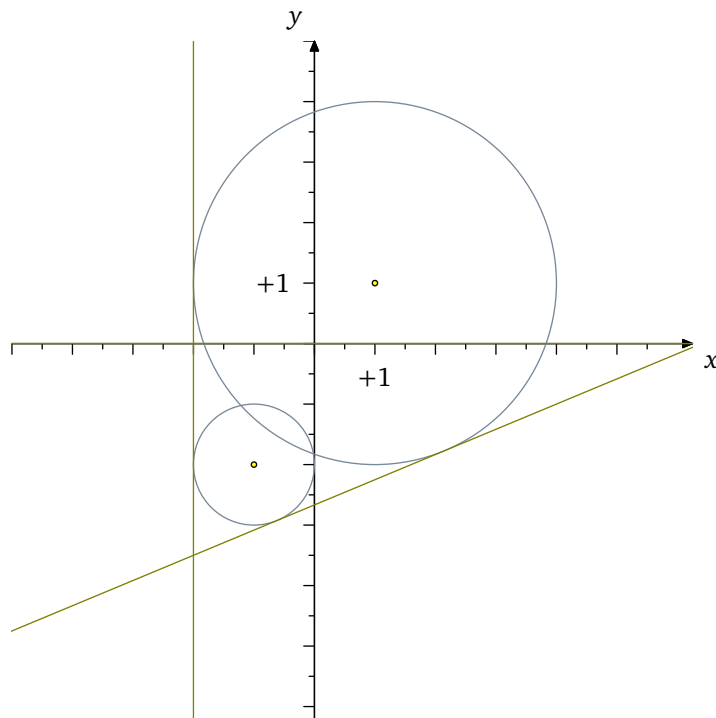
## 13 SVG Colors (svgnames)

 AliceBlue	 AntiqueWhite	 Aqua	 Aquamarine
 Azure	 Beige	 Bisque	 Black
 BlanchedAlmond	 Blue	 BlueViolet	 Brown
 BurlyWood	 CadetBlue	 Chartreuse	 Chocolate
 Coral	 CornflowerBlue	 Cornsilk	 Crimson
 Cyan	 DarkBlue	 DarkCyan	 DarkGoldenrod
 DarkGray	 DarkGreen	 DarkGrey	 DarkKhaki
 DarkMagenta	 DarkOliveGreen	 DarkOrange	 DarkOrchid
 DarkRed	 DarkSalmon	 DarkSeaGreen	 DarkSlateBlue
 DarkSlateGray	 DarkSlateGrey	 DarkTurquoise	 DarkViolet
 DeepPink	 DeepSkyBlue	 DimGray	 DimGrey
 DodgerBlue	 FireBrick	 FloralWhite	 ForestGreen
 Fuchsia	 Gainsboro	 GhostWhite	 Gold
 Goldenrod	 Gray	 Green	 GreenYellow
 Grey	 Honeydew	 HotPink	 IndianRed
 Indigo	 Ivory	 Khaki	 Lavender
 LavenderBlush	 LawnGreen	 LemonChiffon	 LightBlue
 LightCoral	 LightCyan	 LightGoldenrod	 LightGoldenrodYellow
 LightGray	 LightGreen	 LightGrey	 LightPink
 LightSalmon	 LightSeaGreen	 LightSkyBlue	 LightSlateBlue
 LightSlateGray	 LightSlateGrey	 LightSteelBlue	 LightYellow
 Lime	 LimeGreen	 Linen	 Magenta
 Maroon	 MediumAquamarine	 MediumBlue	 MediumOrchid
 MediumPurple	 MediumSeaGreen	 MediumSlateBlue	 MediumSpringGreen
 MediumTurquoise	 MediumVioletRed	 MidnightBlue	 MintCream
 MistyRose	 Moccasin	 NavajoWhite	 Navy
 NavyBlue	 OldLace	 Olive	 OliveDrab
 Orange	 OrangeRed	 Orchid	 PaleGoldenrod
 PaleGreen	 PaleTurquoise	 PaleVioletRed	 PapayaWhip
 PeachPuff	 Peru	 Pink	 Plum
 PowderBlue	 Purple	 Red	 RosyBrown
 RoyalBlue	 SaddleBrown	 Salmon	 SandyBrown
 SeaGreen	 Seashell	 Sienna	 Silver
 SkyBlue	 SlateBlue	 SlateGray	 SlateGrey
 Snow	 SpringGreen	 SteelBlue	 Tan
 Teal	 Thistle	 Tomato	 Turquoise
 Violet	 VioletRed	 Wheat	 White
 WhiteSmoke	 Yellow	 YellowGreen	



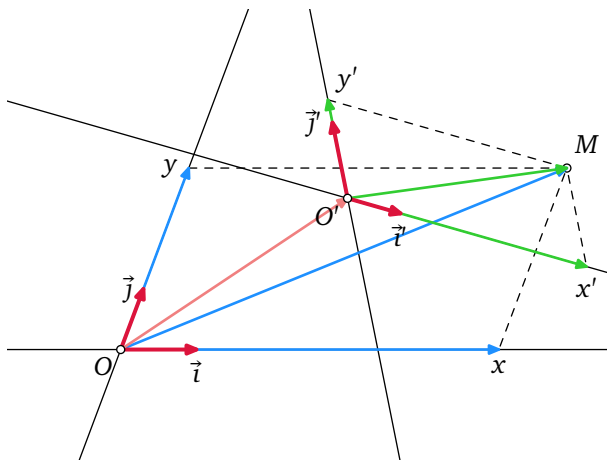
## 14 Gallery

### 14.1 Coordinate System and External Tangents



```
1 input geom2d;
2
3 labeloffset := 6;
4 SetPointSize(2);
5 SetPointColor(Yellow);
6
7 beginfig(1);
8 Frame(9,9,4,5,0.8,0.8);
9 Axis;
10 gddBegin;
11 Axis;
12 Graduations;
13 Units(1);
14
15 C1 = Circle((1,1),3);
16 C2 = Circle((-1,-2),1);
17
18 T1 = ExternalCommonTangent(C1,C2);
19 T2 = ExternalCommonTangent(C2,C1);
20
21 drawoptions(withcolor LightSlateGrey);
22 gddDraw C1;
23 gddDraw C2;
24
25 drawoptions(withcolor Olive);
26 gddDraw T1;
27 gddDraw T2;
28 gddDraw Xaxis;
29
30 drawoptions();
31 gddDrawPoint Point(1,1);
32 gddDrawPoint Point(-1,-2);
33 gddEnd;
34 endfig;
35 end
```

## 14.2 Vector in a Coordinate System



```

1 input geom2d;
2
3 labeloffset := 4;
4 gddU:=1.cm;
5
6 beginfig(1);
7 O = Point(0,0);
8 I = Point(1,0);
9 J = Point(0.3,0.8);
10 M = PointInBasis(5,3,0,I,J);
11 H = PointInBasis(5,0,0,I,J);
12 K = PointInBasis(0,3,0,I,J);
13 O' = Point(3,2);
14 I' = Point(3.7,1.8);
15 J' = Point(2.8,3);
16
17 pair Mt;
18 Mt = CoordinatesInBasis(M,O',I',J');
19 H' = PointInBasis(xpart Mt,0,O',I',J');

```

```

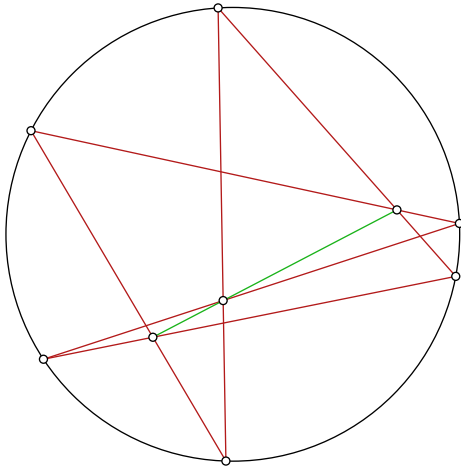
20 K' = PointInBasis(0,ypart Mt,O',I',J');
21
22 Frame(8,6,1.5,1.5,1,1);
23 gddBegin;
24 gddDraw Line(O,I);
25 gddDraw Line(O,J);
26 gddDraw Line(O',I');
27 gddDraw Line(O',J');
28
29 gddDraw Pt(K)--Pt(M)--Pt(H) dashed evenly;
30 gddDraw Pt(K')--Pt(M)--Pt(H') dashed evenly;
31
32 gddMark.urt "M";
33
34 drawoptions(withpen pencircle scaled 1pt withcolor DodgerBlue);
35 gddArrow Segment(O,K);
36 gddArrow Segment(O,H);
37 gddArrow Segment(O,M);
38
39 drawoptions(withpen pencircle scaled 1pt withcolor LimeGreen);
40 gddArrow Segment(O',M);
41 gddArrow Segment(O',H');
42 gddArrow Segment(O',K');
43
44 drawoptions(withpen pencircle scaled 1.5pt withcolor Crimson);
45 gddArrow Segment(O,I);
46 gddArrow Segment(O,J);
47 gddArrow Segment(O',I');
48 gddArrow Segment(O',J');
49
50 drawoptions(withpen pencircle scaled 1pt withcolor LightCoral);
51 gddArrow Segment(O,O');
52
53 drawoptions();
54 gddMark.llft "O";
55 gddMark.llft "O'";
56
57 label.bot(texttext("\(x\)"), PtR(H));
58 label.lft(texttext("\(y\)"), PtR(K));
59 label.bot(texttext("\(\vec{i}\)", PtR(I));
60 label.lft(texttext("\(\vec{j}\)", PtR(J));

```

```
61 label.bot(texttext("\(x'\)"), PtR(H'));
62 label.urt(texttext("\(y'\)"), PtR(K'));
63 label.bot(texttext("\(\vec{\imath}'\)"), PtR(I'));
64 label.lft(texttext("\(\vec{\jmath}'\)"), PtR(J'));
```

```
66 gddEnd;
67 endfig;
68 end
```

## 14.3 Pascal's Theorem



```

1 input geom2d;
2
3 beginfig(1);
4 C = Circle(origine,3);

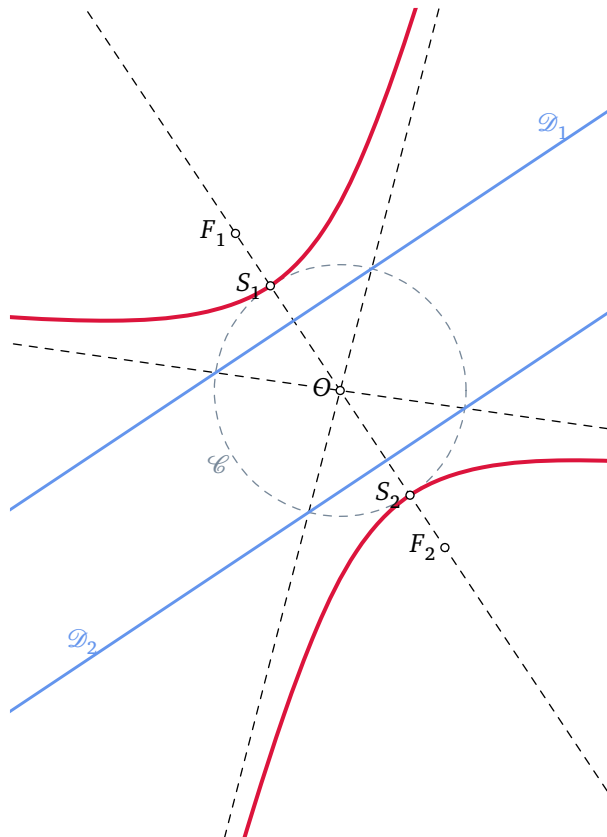
```

```

5 for i:=1 upto 6:
6   rd := uniformdeviate(1.0/6)+(i-1)/6;
7   P[i] := PointOf(C,rd);
8 endfor;
9 D1 = Line(P1,P3); S1 = Segment(P1,P3);
10 D2 = Line(P3,P5); S2 = Segment(P3,P5);
11 D3 = Line(P6,P2); S3 = Segment(P6,P2);
12 D4 = Line(P4,P6); S4 = Segment(P4,P6);
13 D5 = Line(P5,P2); S5 = Segment(P5,P2);
14 D6 = Line(P1,P4); S6 = Segment(P1,P4);
15 I1 = LinesIntersection(D1,D3);
16 I2 = LinesIntersection(D2,D4);
17 I3 = LinesIntersection(D5,D6);
18 PL = Segment(I1,I2);
19 gddDraw C;
20 drawoptions(withcolor (0.7,0.1,0.1));
21 gddDraw S1; gddDraw S2; gddDraw S3; gddDraw S4; gddDraw S5; gddDraw S
22   6;
23 drawoptions(withcolor (0.1,0.7,0.1));
24 gddDraw PL;
25 gddDrawPoint P1; gddDrawPoint P2; gddDrawPoint P3; gddDrawPoint P4;
26   gddDrawPoint P5; gddDrawPoint P6;
27 gddDrawPoint I1; gddDrawPoint I2; gddDrawPoint I3;
28 endfig;
end.

```

## 14.4 Hyperbola



```

1 input geom2d;
2
3 beginfig(1);
4 A = Point(0,0);

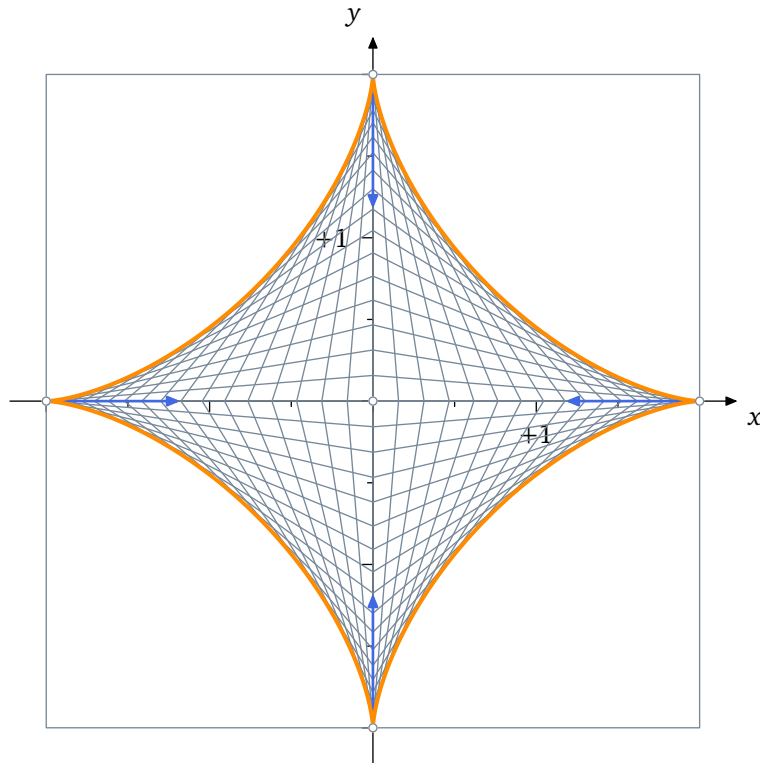
```

```

5 B = Point(3,2);
6 AB = Line(A,B);
7 F_1 = Point(-1,1);
8 Hyper = HyperbolaFD(F_1,AB,1.5);
9 O = Center(Hyper);
10 F_2 = Focus(Hyper,2);
11 Axe = Line(F_1,F_2);
12 S_1 = Sommet(Hyper,1);
13 S_2 = Sommet(Hyper,2);
14
15 gddDraw Axe dashed evenly;
16 C = PrincipalCircle(Hyper) ;
17 gddDraw C withPen(0.5,LightSlateGray) dashed evenly;
18 A_1 = HyperbolaAsymptote(Hyper,1);
19 A_2 = HyperbolaAsymptote(Hyper,2);
20 gddDraw A_1 dashed evenly;
21 gddDraw A_2 dashed evenly;
22
23 D_1 = Directrix(Hyper,1);
24 D_2 = Directrix(Hyper,2);
25
26 gddDraw D_1 withPen(1.1,CornflowerBlue);
27 gddDraw D_2 withPen(1.1,CornflowerBlue);
28 gddDraw SemiHyperbola(Hyper,1) withPen(1.5,Crimson);
29 gddDraw SemiHyperbola(Hyper,2) withPen(1.5,Crimson);
30
31 gddMark.lft "O";
32 gddMark.lft "S_1";
33 gddMark.lft "S_2";
34 gddMark.lft "F_1";
35 gddMark.lft "F_2";
36 gddLabel.lft(texttext("\(\mathcal{C}\)"),PointDe(C,0.6)) withcolor
    LightSlateGray;
37 gddLabel.top(texttext("\(\mathcal{D}_1\)\)"),PointDe(D_1,0.47))
    withcolor CornflowerBlue;
38 gddLabel.top(texttext("\(\mathcal{D}_2\)\)"),PointDe(D_2,0.46))
    withcolor CornflowerBlue;
39 Window(-4,-7,4,4);
40 endfig;
41
42
43 end.

```

## 14.5 Astroid as an Envelope of Lines

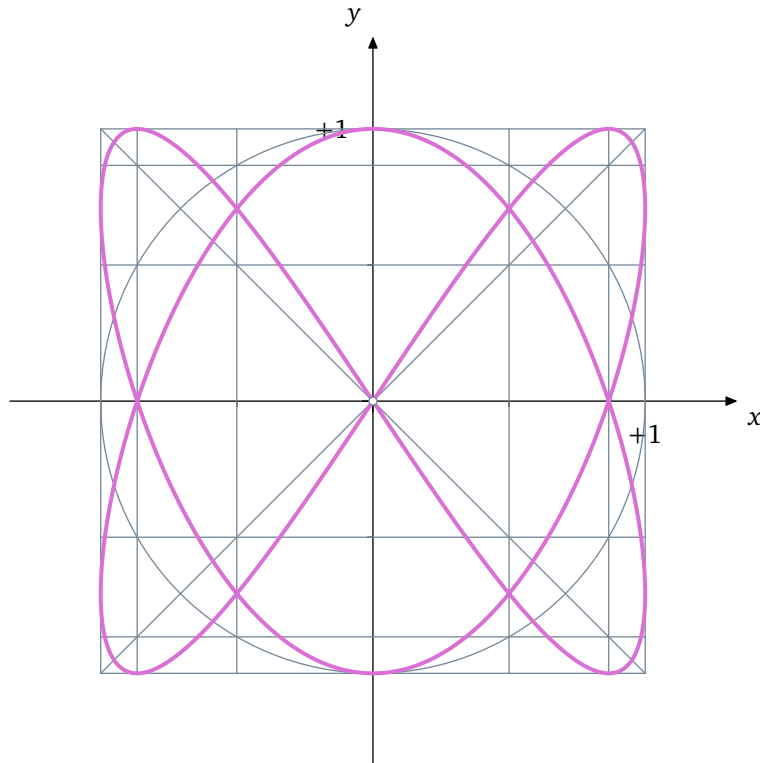


```
1 input geom2d;
2
```

```
3
4 labeloffset := 6;
5 gddU:=1.2cm;
6
7 vardef f(expr t) = 2*cos(t)*cos(t)*cos(t) enddef;
8 vardef g(expr t) = 2*sin(t)*sin(t)*sin(t) enddef;
9
10
11 beginfig(1);
12 Frame(8,8,4,4,1.8,1.8);
13 Axis;
14 gddBegin;
15 Graduations; Units(1);
16 drawoptions(withpen pencircle scaled 0.5 withcolor LightSlateGrey);
17 gddDraw ((-2,-2)--(-2,2)--(2,2)--(2,-2)--cycle);
18
19 nb = 80;
20 pas = 2*Pi/nb;
21 for i=0 upto nb: gddDraw (0,2*sin(i*pas))--(2*cos(i*pas),0); endfor
22 ;
23
24 gddArrow Segment((2,0),(1.2,0)) withPen(1,RoyalBlue);
25 gddArrow Segment((-2,0),(-1.2,0)) withPen(1,RoyalBlue);
26 gddArrow Segment((0,2),(0,1.2)) withPen(1,RoyalBlue);
27 gddArrow Segment((0,-2),(0,-1.2)) withPen(1,RoyalBlue);
28
29 gddDraw Curve(f,g,0,2*Pi,500) withPen(1.5,DarkOrange);
30
31 gddDrawPoint Point(0,0);
32 gddDrawPoint Point(2,0);
33 gddDrawPoint Point(0,2);
34 gddDrawPoint Point(0,-2);
35 gddDrawPoint Point(-2,0);
36
37 gddEnd;
38 endfig;
39 end
```



## 14.6 Lissajous Curve

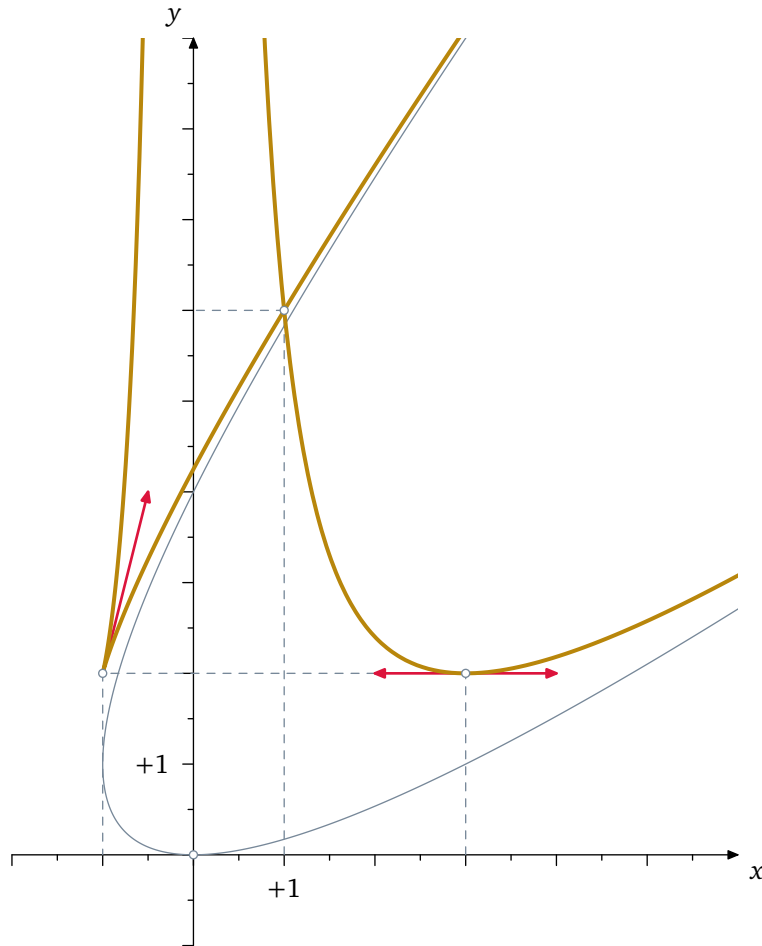


```

1 input geom2d;
2
3 labeloffset := 6;
4 gddU:=1.2cm;
5
6 vardef f(expr t) = sin(2*t+Pi/3) enddef;
7 vardef g(expr t) = sin(3*t) enddef;
8
9 beginfig(1);
10 Frame(8,8,4,4,3,3);
11 Axis;
12 gddBegin;
13 Graduations; Units(1);
14
15 drawoptions(withcolor LightSlateGrey);
16 gddDraw ((-1,-1)--(-1,1)--(1,1)--(1,-1)--cycle);
17 gddDraw ((-1,-1)--(1,1));
18 gddDraw (1,-1)--(-1,1);
19 gddDraw Circle(origine,1);
20 gddDraw ((-1,0.5)--(1,0.5));
21 gddDraw ((-1,-0.5)--(1,-0.5));
22 gddDraw ((0.5,-1)--(0.5,1));
23 gddDraw ((-0.5,-1)--(-0.5,1));
24 gddDraw ((-1,sqrt(3)/2)--(1,sqrt(3)/2));
25 gddDraw ((-1,-sqrt(3)/2)--(1,-sqrt(3)/2));
26 gddDraw ((sqrt(3)/2,-1)--(sqrt(3)/2,1));
27 gddDraw ((-sqrt(3)/2,-1)--(-sqrt(3)/2,1));
28
29 gddDraw Curve(f,g,0,2*Pi,500) withPen(1.5,Orchid);
30
31 pointe Point(0,0);
32
33 gddEnd;
34 endfig;
35 end

```

## 14.7 Function Study

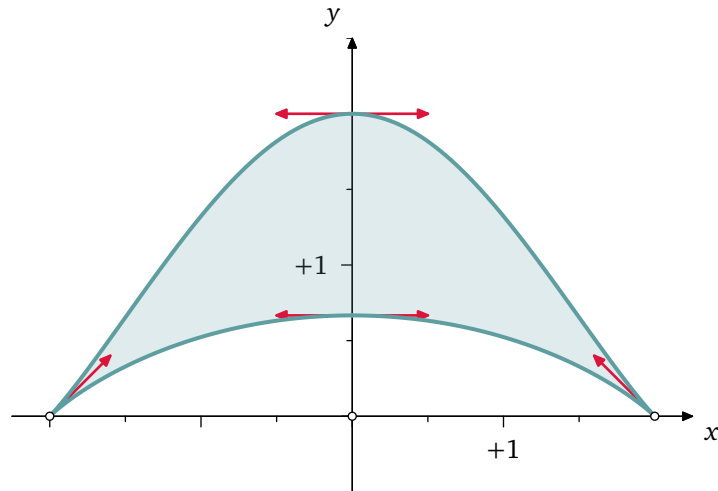


```

1 input geom2d;
2
3 labeloffset := 6;
4 gddU:=1.2cm;
5
6 vardef f(expr t) = t*t-2*t enddef;
7 vardef g(expr t) = t*t + 1/(t*t) enddef;
8 vardef fp(expr t) = 2*t+t*t enddef;
9 vardef gp(expr t) = t*t enddef;
10
11 beginfig(1);
12
13 Frame(8,10,2,1,1,1);
14 Axis;
15 gddBegin;
16 Graduations; Units(1);
17
18 drawoptions(withcolor LightSlateGrey);
19 gddDraw (3,0)--(3,2)--(-1,2)--(-1,0) dashed evenly;
20 gddDraw (1,0)--(1,6)--(0,6) dashed evenly;
21 gddDraw Curve(fp,gp,-5,5,500);
22
23 gddArrow Segment((-1,2),(-1,2)+(0.5,2)) withPen(1,Crimson);
24 gddArrow Segment((3,2),(4,2)) withPen(1,Crimson);
25 gddArrow Segment((3,2),(2,2)) withPen(1,Crimson);
26
27 gddDraw Curve(f,g,-10,-0.05,300) withPen(1.5,DarkGoldenrod);
28 gddDraw Curve(f,g,0.05,10,300) withPen(1.5,DarkGoldenrod);
29
30 gddDrawPoint Point(0,0);
31 gddDrawPoint Point(-1,2);
32 gddDrawPoint Point(3,2);
33 gddDrawPoint Point(1,6);
34
35 gddEnd;
36 endfig;
37 end

```

## 14.8 Bicorne



```

1 input geom2d;
2
3 labeloffset := 6;
4 gddU:=1cm;
5
6 vardef f(expr t) = 2*sin(t) enddef;
7 vardef g(expr t) = 2*cos(t)*cos(t)/(2-cos(t)) enddef;
8 vardef h(expr t) = -t*(1+f(t)) enddef;

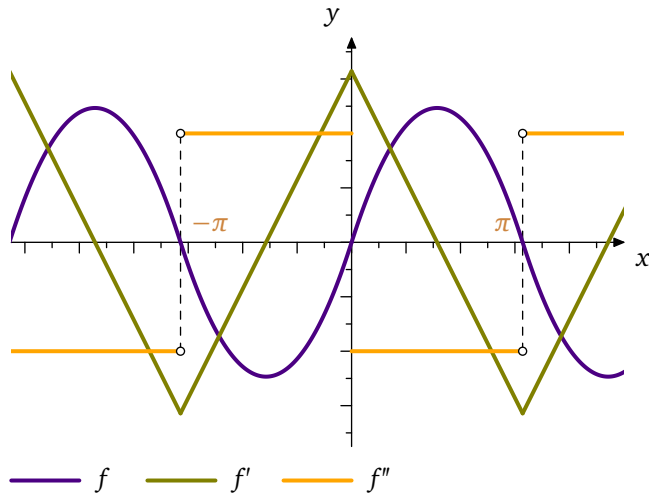
```

```

9
10 def DrawDoubleVector(expr o,d)= drawdblarrow ((o-d)--(o+d))
    gddEnPlace enddef;
11
12 path bicorne;
13 bicorne = Curve(f,g,-Pi,Pi,200)--cycle;
14
15 beginfig(1);
16
17 Frame(9,6,4.5,1,2,2);
18 Axis;
19 gddBegin;
20 Axis;
21 Graduations; Units(1);
22
23 gddAlphaFill(bicorne,CadetBlue,0.2);
24
25 DrawDoubleVector((0,2),(0.5,0)) withPen(1,Crimson);
26 DrawDoubleVector((0,2/3),(0.5,0)) withPen(1,Crimson);
27 gddArrow Segment((2,0),(1.6,0.4)) withPen(1,Crimson);
28 gddArrow Segment((-2,0),(-1.6,0.4)) withPen(1,Crimson);
29
30 gddDraw bicorne withPen(1.5,CadetBlue);
31
32 gddDrawPoint Point(0,0);
33 gddDrawPoint Point(2,0);
34 gddDrawPoint Point(-2,0);
35
36 gddEnd;
37 endfig;
38 end

```

## 14.9 A function et its derivatives



```

1 input geom2d;
2 labeloffset := 6;
3 gddU:=0.9cm;
4
5 vardef f(expr x) = x*(Pi-x) enddef; % f
6 vardef g(expr x) = Pi-2*x enddef; % f'
7 vardef h(expr x) = -2 enddef; % f''
8
9 beginfig(1);
10
11 Frame(9,6,5,3,.8,0.8);
12 Axis;
13 gddBegin;
14 Graduations;
15
16 gddDraw (Pi,-2)--(Pi,2) dashed evenly;

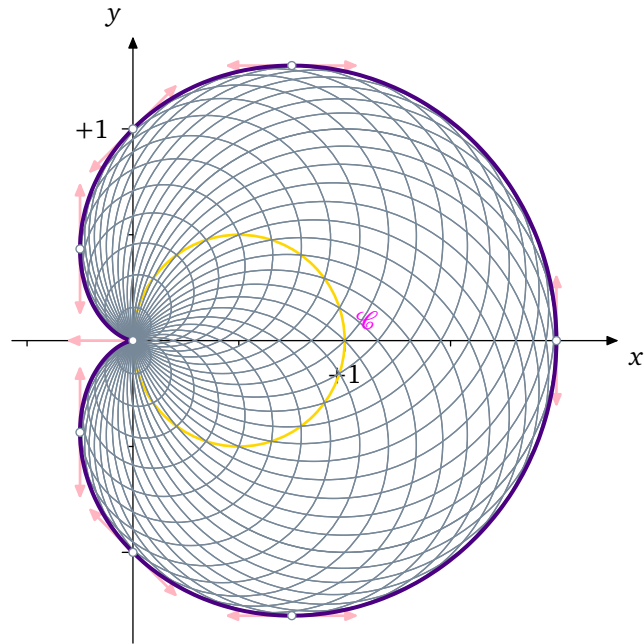
```

```

17 gddDraw (-Pi,-2)--(-Pi,2) dashed evenly;
18
19 gddDraw Plot(f,0,Pi,100) withPen(1.5,Indigo);
20 gddDraw (Plot(f,0,Pi,100) scaled -1) withPen(1.5,Indigo);
21 gddDraw (Plot(f,0,Pi,100) shifted (-2*Pi,0)) withPen(1.5,Indigo);
22 gddDraw (Plot(f,0,Pi,100) scaled -1 shifted (2*Pi,0)) withPen(1.5,
    Indigo);
23
24 gddDraw Plot(g,0,Pi,100) withPen(1.5,Olive);
25 gddDraw (Plot(g,0,Pi,100) xscaled -1) withPen(1.5,Olive);
26 gddDraw (Plot(g,0,Pi,100) shifted (-2*Pi,0)) withPen(1.5,Olive);
27 gddDraw (Plot(g,0,Pi,100) xscaled -1 shifted (2*Pi,0)) withPen(1.5,
    Olive);
28
29 gddDraw Plot(h,0,Pi,100) withPen(1.5,Orange);
30 gddDraw (Plot(h,0,Pi,100) scaled -1) withPen(1.5,Orange);
31 gddDraw (Plot(h,0,Pi,100) shifted (-2*Pi,0)) withPen(1.5,Orange);
32 gddDraw (Plot(h,0,Pi,100) scaled -1 shifted (2*Pi,0)) withPen(1.5,
    Orange);
33
34 gddDrawPoint Point(Pi,2);
35 gddDrawPoint Point(Pi,-2);
36 gddDrawPoint Point(-Pi,2);
37 gddDrawPoint Point(-Pi,-2);
38
39 gddLabel.urt(texttext("\\(-\\pi\\)"), (-Pi,0)) withcolor Peru;
40 gddLabel.ulft(texttext("\\(\\pi\\)"), (Pi,0)) withcolor Peru;
41 gddEnd;
42
43 gddDraw (0,-0.5)--(1,-0.5) withPen(1.5,Indigo);
44 gddLabel.rt(texttext("(f\\)"), (1,-0.5));
45 gddDraw (2,-0.5)--(3,-0.5) withPen(1.5,Olive);
46 gddLabel.rt(texttext("(f'\\)"), (3,-0.5));
47 gddDraw (4,-0.5)--(5,-0.5) withPen(1.5,Orange);
48 gddLabel.rt(texttext("(f''\\)"), (5,-0.5));
49 endfig;
50 end

```

## 14.10 Cardioid



```

1 input geom2d;
2
3 labeloffset := 6;
4 gddU:=0.8cm;
5
6 vardef r(expr t) = 1+cos(t) enddef;
7 vardef rp(expr t) = (r(t)*cos(t),r(t)*sin(t)) enddef;
8
9 def DrawDoubleVector(expr o,d)= drawdblarrow ((o-d)--(o+d))
   gddInPlace enddef;
10
11 beginfig(1);
12 Frame(10,10,2,5,3.5,3.5);

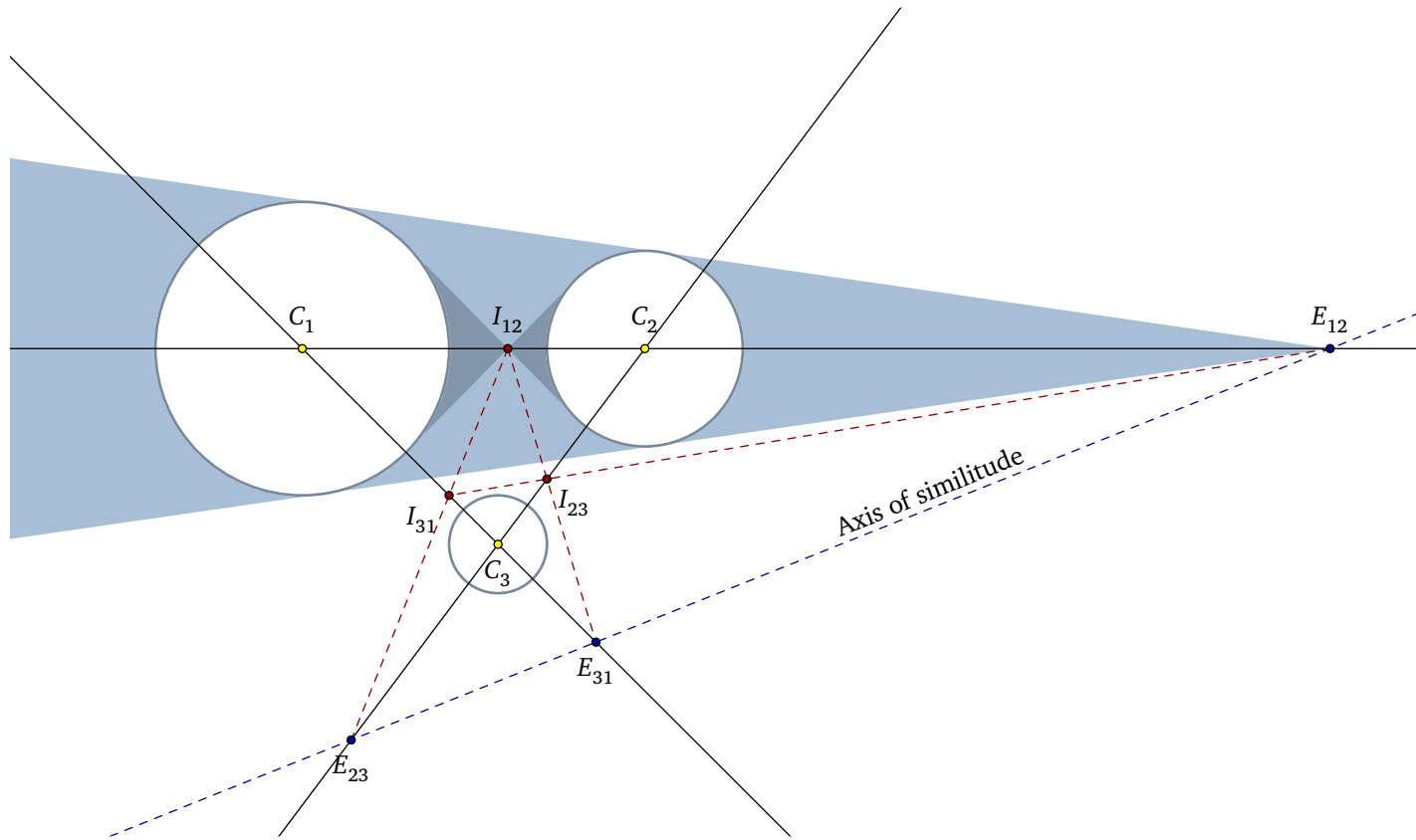
```

```

13 Axis;
14 gddBegin;
15 Graduations; Units(1);
16
17 drawoptions(withcolor LightSlateGrey);
18 draw fullcircle shifted (0.5,0) gddInPlace withPen(1,Gold);
19
20 nb = 80;
21 pas = 2Pi / nb;
22 for i=0 upto nb:
23   theta := i * pas;
24   draw (fullcircle scaled 2cos(theta)
25     shifted (cos(theta)*cos(theta),cos(theta)*sin(theta)))
26     gddInPlace;
27 endfor;
28
29 DrawDoubleVector(rp(0),(0,0.3)) withPen(1,LightPink);
30 DrawDoubleVector(rp(Pi/3),(0.3,0)) withPen(1,LightPink);
31 DrawDoubleVector(rp(Pi/2),(0.2,0.2)) withPen(1,LightPink);
32 DrawDoubleVector(rp(2Pi/3),(0,0.3)) withPen(1,LightPink);
33 DrawDoubleVector(rp(-Pi/3),(0.3,0)) withPen(1,LightPink);
34 DrawDoubleVector(rp(-Pi/2),(0.2,-0.2)) withPen(1,LightPink);
35 DrawDoubleVector(rp(-2Pi/3),(0,0.3)) withPen(1,LightPink);
36
37 gddArrow Segment(origine,(-0.3,0)) withPen(1,LightPink);
38 gddDraw PolarCurve(r,-Pi,Pi,100) withPen(1.5,Indigo);
39
40 gddDrawPoint Point(0,0);
41 gddDrawPoint Point(2,0);
42 gddDrawPoint PairTOPoint(rp(Pi/3));
43 gddDrawPoint PairTOPoint(rp(-Pi/3));
44 gddDrawPoint PairTOPoint(rp(2Pi/3));
45 gddDrawPoint PairTOPoint(rp(-2Pi/3));
46 gddDrawPoint Point(0,1);
47 gddDrawPoint Point(0,-1);
48
49 gddLabel.urt(texttext("\(\mathcal{C}\)"),(1,0)) withcolor Magenta;
50 gddEnd;
51 endfig;
52 end

```

### 14.11 Axis of similitude



```

1 input geom2d;
2 %%% from Drawing with Metapost de Toby Thurston
3 labeloffset := 6;
4 SetPointSize(3);
5 SetPointColor(Yellow);
6 gddU:=0.65cm;
7 beginfig(1);
8 C1 = Circle((-4,0),3);
9 C2 = Circle((3,0),2);
10 C3 = Circle((0,-4),1);
11
12 T1 = ExternalCommonTangent(C1,C2);
13 T2 = ExternalCommonTangent(C2,C1);
14 T3 = InternalCommonTangent(C1,C2);
15 T4 = InternalCommonTangent(C2,C1);
16
17 T5 = ExternalCommonTangent(C2,C3);
18 T6 = ExternalCommonTangent(C3,C2);
19 T7 = InternalCommonTangent(C2,C3);
20 T8 = InternalCommonTangent(C3,C2);
21
22 T9 = ExternalCommonTangent(C1,C3);
23 T10 = ExternalCommonTangent(C3,C1);
24 T11 = InternalCommonTangent(C1,C3);
25 T12 = InternalCommonTangent(C3,C1);
26
27 E12 = LinesIntersection(T1,T2);
28 E23 = LinesIntersection(T5,T6);
29 E31 = LinesIntersection(T9,T10);
30 I12 = LinesIntersection(T3,T4);
31 I23 = LinesIntersection(T7,T8);
32 I31 = LinesIntersection(T11,T12);
33
34 path t[];
35 t1 :=(gddTraceObjet T1) gddEnPlace;
36 t2 := (gddTraceObjet T2) gddEnPlace;
37 t3 := (-10*gddU,-10*gddU)--(-10gddU,10*gddU);
38 fill buildcycle(t1, t3,reverse t2) withcolor 1.4*LightSlateGrey;
39
40 t4 :=(gddTraceObjet T3) gddEnPlace;
41 t5 := (gddTraceObjet T4) gddEnPlace;
42 t6 := (1.5*gddU,-10*gddU)--(1.5*gddU,10*gddU);
43 fill buildcycle(t4, t5,reverse t6) withcolor 1.1*LightSlateGrey;

```

```

44
45 t7 :=(gddTraceObjet T3) gddEnPlace;
46 t8 := (gddTraceObjet T4) gddEnPlace;
47 t9 := (-1.5*gddU,-10*gddU)--(-1.5*gddU,10*gddU);
48 fill buildcycle(t7, t8,reverse t9) withcolor 1.1*LightSlateGrey;
49
50 drawoptions(withpen pencircle scaled 1pt withcolor LightSlateGrey);
51 gddFill C1 withcolor white;
52 gddFill C2 withcolor white;
53 gddFill C3 withcolor white;
54 gddDraw C1; gddDraw C2; gddDraw C3;
55
56 drawoptions();
57 gddDraw Line(Center(C1),Center(C2));
58 gddDraw Line(Center(C3),Center(C2));
59 gddDraw Line(Center(C1),Center(C3));
60
61 D_E = Line(E12,E23);
62 gddDraw D_E dashed evenly withcolor DarkBlue;
63
64 E_I = Segment(E12,I31);
65 gddDraw E_I dashed evenly withcolor DarkRed;
66 I_E1 = Segment(I12,E31);
67 gddDraw I_E1 dashed evenly withcolor DarkRed;
68 I_E2 = Segment(I12,E23);
69 gddDraw I_E2 dashed evenly withcolor DarkRed;
70
71
72 drawoptions();
73 gddDrawPoint Center(C1);
74 gddDrawPoint Center(C2);
75 gddDrawPoint Center(C3);
76 SetPointColor(DarkBlue);
77 gddDrawPoint E12; gddDrawPoint E31; gddDrawPoint E23;
78 SetPointColor(DarkRed);
79 gddDrawPoint I12; gddDrawPoint I31; gddDrawPoint I23;
80
81 gddLabel.top(btex $E_{12}$ etex,E12);
82 gddLabel.bot(btex $E_{31}$ etex,E31);
83 gddLabel.bot(btex $E_{23}$ etex,E23);
84
85 gddLabel.top(btex $I_{12}$ etex,I12);
86 gddLabel.llft(btex $I_{31}$ etex,I31);
87 gddLabel.lrt(btex $I_{23}$ etex,I23);

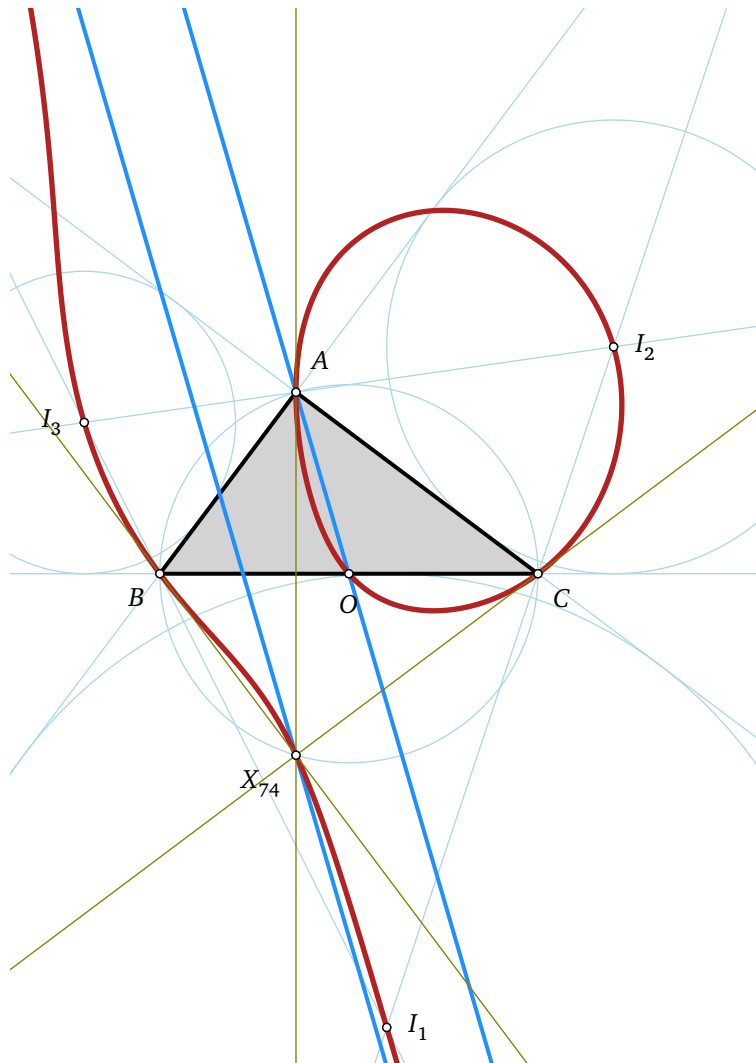
```

```
88
89
90 gddLabel.top(btex  $C_1$  etex, Center(C1));
91 gddLabel.top(btex  $C_2$  etex, Center(C2));
92 gddLabel.bot(btex  $C_3$  etex, Center(C3));
93
```

```
94 draw texttext("Axis of similitude") rotated (22) shifted (0.5[Pt(E
    23),Pt(E12)]
95 gddInPlace +(0,3));
96 Window(-10,-10,19,7);
97 endfig;
98 end.
```



## 14.12 Cubic Drawing



```

1 input geom2d;
2 gddU := 0.5cm;
3 labeloffset := 8pt;
4 gddTailleLabel := 1;
5
6 beginfig(1);
7 A := Point(3.6,4.8);
8 B := Point(0,0);
9 C := Point(10,0);
10 T := Triangle(A,B,C);
11 X_74 := Point(3.6,-4.8);
12
13 C1 := CurveData("K001-1",0);
14 C2 := CurveData("K001-2",1);
15
16 CE1 := EscribedCircle(T,1);
17 CE2 := EscribedCircle(T,2);
18 CE3 := EscribedCircle(T,3);
19 I_1 := Center(CE1);
20 I_2 := Center(CE2);
21 I_3 := Center(CE3);
22 CC := CircumscribedCircle(T);
23 O := Center(CC);
24
25 drawoptions(withcolor LightBlue);
26 gddDraw Line(A,B);
27 gddDraw Line(B,C);
28 gddDraw Line(C,A);
29 gddDraw Line(I_1,I_2);
30 gddDraw Line(I_2,I_3);
31 gddDraw Line(I_3,I_1);
32 gddDraw CE1;
33 gddDraw CE2;
34 gddDraw CE3;
35 gddDraw CC;
36 colorie T withcolor LightGrey;
37 drawoptions(withpen pencircle scaled 1.5);
38 gddDraw T;

```

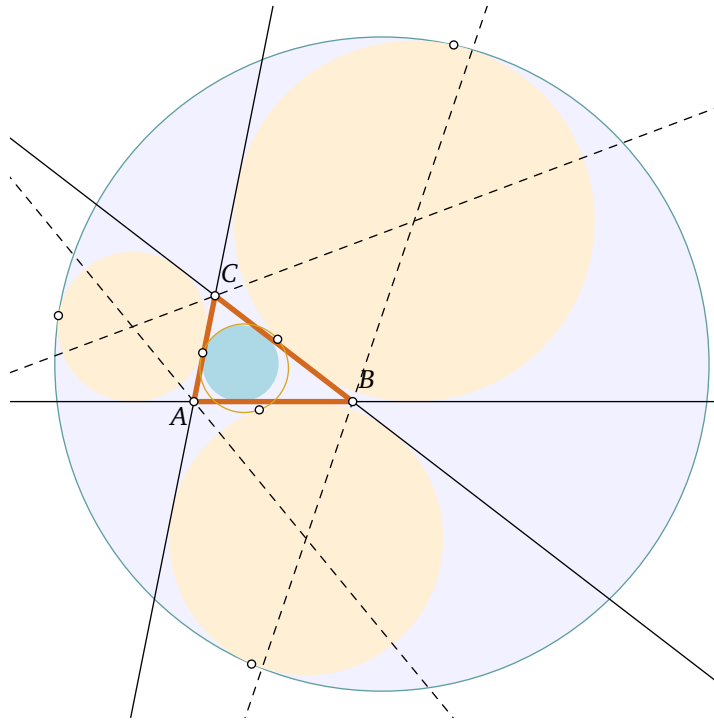
```

39 drawoptions(withcolor DodgerBlue withpen pencircle scaled 1.5);
40 gddDraw Line(0,A);
41 gddDraw Line(X_74, Point(2.2,0));
42 drawoptions(withpen pencircle scaled 2);
43 gddDraw C1 withcolor FireBrick;
44 gddDraw C2 withcolor FireBrick;
45 drawoptions(withcolor Olive);
46 gddDraw Line(A,X_74);
47 gddDraw Line(B,X_74);
48 gddDraw Line(C,X_74);
49 drawoptions();
50 gddDrawPoint A;
51 gddDrawPoint B;
52 gddDrawPoint C;
53 gddDrawPoint I_1;
54 gddDrawPoint I_2;
55 gddDrawPoint I_3;

56 gddDrawPoint 0;
57 gddDrawPoint X_74;
58
59 gddMark.urt "A";
60 gddMark.llft "B";
61 gddMark.lrt "C";
62 gddMark.rt "I_1";
63 gddMark.rt "I_2";
64 gddMark.lft "I_3";
65 gddMark.bot "0";
66 gddMark.llft "X_74";
67
68 Window(-4,-13,16,15);
69 endfig;
70
71 end

```

## 14.13 Apollonius



```

1 input geom2d;
2 gddU:=0.35cm;
3 beginfig(1);
4
5 A = Point(0,0);
6 B = Point(6,0);
7 C = Point(0.8,4);
8 T_ABC = Triangle(A,B,C);

```

```

9
10 C_I = InscribedCircle(T_ABC);
11 C_A = EscribedCircle(T_ABC,2);
12 C_B = EscribedCircle(T_ABC,3);
13 C_C = EscribedCircle(T_ABC,1);
14
15 d_AB = Line(A,B);
16 d_BC = Line(B,C);
17 d_CA = Line(C,A);
18
19 I = Center(C_I);
20 I_C_A = Center(C_A);
21 I_C_B = Center(C_B);
22 I_C_C = Center(C_C);
23 d_CAA = Line(A,I_C_A);
24 d_CAB = Line(B,I_C_B);
25 d_CAC = Line(C,I_C_C);
26
27 A_S = AxisOfSimilitude(C_A,C_B,C_C);
28 P_CA = PointOnLineProjection(I_C_A,A_S);
29 P_CB = PointOnLineProjection(I_C_B,A_S);
30 P_CC = PointOnLineProjection(I_C_C,A_S);
31
32 P_A = Inverse(P_CA,C_A);
33 P_B = Inverse(P_CB,C_B);
34 P_C = Inverse(P_CC,C_C);
35
36 C_R = RadicalCenter(C_A,C_B,C_C);
37 % les neuf points pour les cercles 'deuler (tangent intérieur)
38 % et 'dapollonius (tangent extérieur)
39 D1 = Line(C_R,P_A);
40 P1 = LineCircleIntersection(D1,C_A,1);
41 Q1 = LineCircleIntersection(D1,C_A,2);
42
43 D2 = Line(C_R,P_B);
44 P2 = LineCircleIntersection(D2,C_B,1);
45 Q2 = LineCircleIntersection(D2,C_B,2);
46
47 D3 = Line(C_R,P_C);
48 P3 = LineCircleIntersection(D3,C_C,2);

```

```

49 Q3 = LineCircleIntersection(D3,C_C,1);
50
51 % le cercle 'dapollonius
52 Apol = CircleThreePoints(P1,P2,P3);
53 gddFill Apol withcolor 1.05*Lavender;
54 gddDraw Apol withcolor CadetBlue;
55
56 % cercle inscrit
57 gddFill C_I withcolor LightBlue;
58 % cercles exinscrits
59 gddFill C_A withcolor PapayaWhip;
60 gddFill C_B withcolor PapayaWhip;
61 gddFill C_C withcolor PapayaWhip;
62
63 gddDraw d_AB;
64 gddDraw d_BC;
65 gddDraw d_CA;
66
67 gddDraw d_CAA dashed evenly;
68 gddDraw d_CAB dashed evenly;
69 gddDraw d_CAC dashed evenly;
70

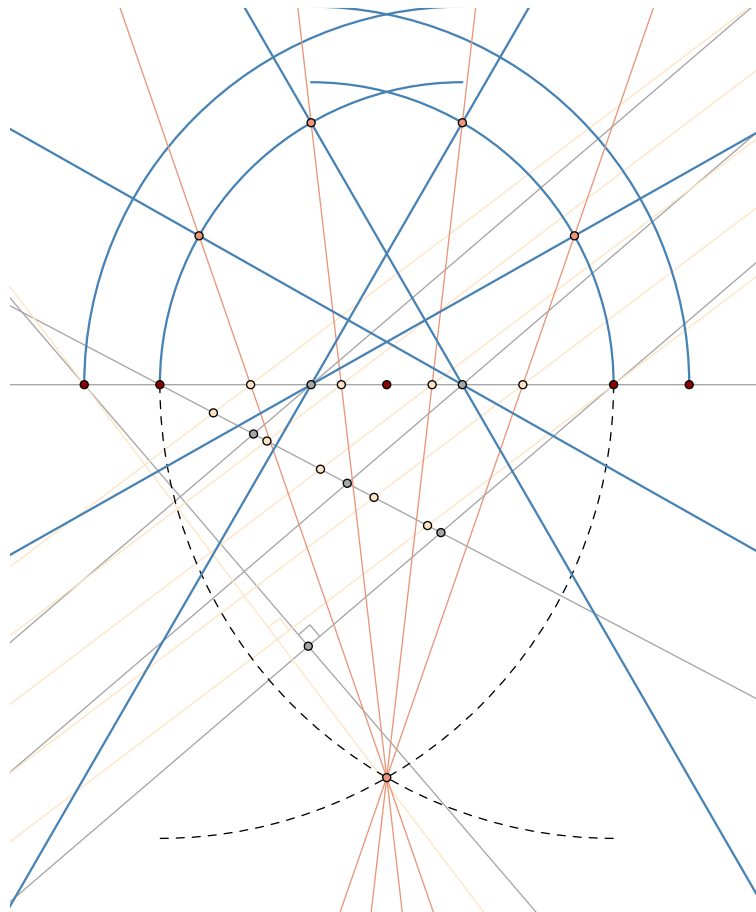
```

```

71 gddDraw T_ABC withpen pencircle scaled 2 withcolor Chocolate;
72
73 C_E = EulerCircle(T_ABC);
74 gddDraw C_E withcolor Goldenrod;
75
76 gddDrawPoint P1;
77 gddDrawPoint P2;
78 gddDrawPoint P3;
79
80 gddDrawPoint Q1;
81 gddDrawPoint Q2;
82 gddDrawPoint Q3;
83
84
85 gddMark.llft "A";
86 gddMark.urrt "B";
87 gddMark.urrt "C";
88 Window(-7,-12,20,15)
89 endfig;
90 end.

```

## 14.14 Ogive Working Drawing



```

1 input geom2d;
2
3 beginfig(1);
4 % base points for the ogive
5 O = Point(0,0);
6 A = Point(-3,0);
7 A2 = Point(-4,0);
8 B = Point(3,0);
9 B2 = Point(4,0);
10 % the base line of the ogive
11 AB = Line(A,B);
12 % we take an arbitrary line
13 Ct = Point(0.8,-2); % arbitrary point
14 AC = Line(A,Ct); % arbitrary line
15 % we take three equal lengths on the line
16 longueurTier := 1.4;
17 C1 = ReportOnLine(A,AC,longueurTier);
18 C2 = ReportOnLine(C1,AC,longueurTier);
19 C3 = ReportOnLine(C2,AC,longueurTier);
20 % we define the line passing through B and the last point
21 BC = Line(B,C3);
22 % we construct a perpendicular line to BC
23 D = Point(-4,0);
24 perpD = PerpendicularLine(BC,D);
25 DD = LinesIntersection(perpD,BC);
26 % we project C2 and C1 onto the line AB to divide [AB] into 3
    equal parts
27 perpC2 = PerpendicularLine(perpD,C2);
28 perpC1 = PerpendicularLine(perpD,C1);
29 D1 = LinesIntersection(AB,perpC1);
30 D2 = LinesIntersection(AB,perpC2);
31 % we construct the envelope of the ogive
32 C_DA = CircleCP(D2,A);
33 C_DB = CircleCP(D1,B);
34 C_DA2 = CircleCP(D2,A2);
35 C_DB2 = CircleCP(D1,B2);
36 % we construct the point P for the final projection
37 C_AB = CircleCP(A,B);
38 C_BA = CircleCP(B,A);
39 P = CirclesIntersection(C_BA,C_AB);
40 % we take 5 equal lengths on the initial arbitrary line

```

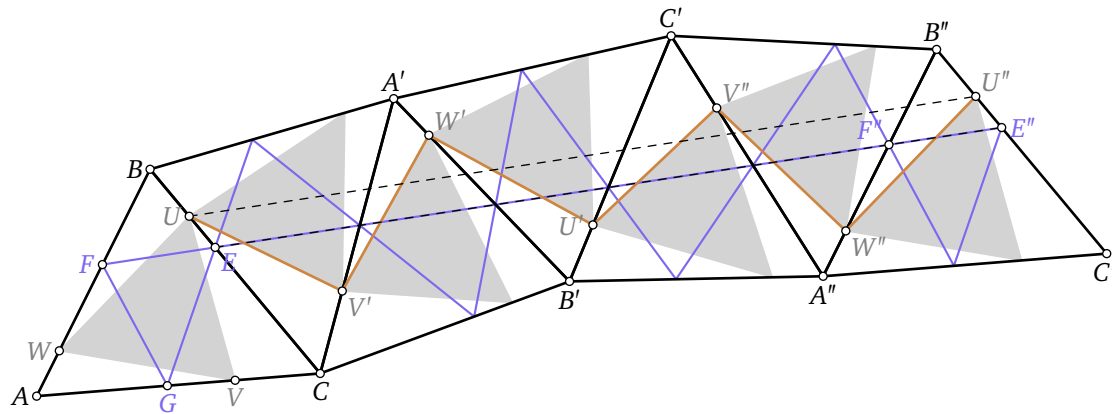
```

41 longueurCinq :=0.8;
42 F1 = ReportOnLine(A,AC,longueurCinq);
43 F2 = ReportOnLine(F1,AC,longueurCinq);
44 F3 = ReportOnLine(F2,AC,longueurCinq);
45 F4 = ReportOnLine(F3,AC,longueurCinq);
46 F5 = ReportOnLine(F4,AC,longueurCinq);
47 % we project the points onto AB to divide [AB] into 5 equal parts
48 BF5 = Line(B,F5);
49 perpF5 = PerpendicularLine(BF5,A2);
50 CC = LinesIntersection(BF5,perpF5);
51 perpF4 = PerpendicularLine(perpF5,F4);
52 perpF3 = PerpendicularLine(perpF5,F3);
53 perpF2 = PerpendicularLine(perpF5,F2);
54 perpF1 = PerpendicularLine(perpF5,F1);
55 G1 = LinesIntersection(AB,perpF1);
56 G2 = LinesIntersection(AB,perpF2);
57 G3 = LinesIntersection(AB,perpF3);
58 G4 = LinesIntersection(AB,perpF4);
59 % we project the Gi points onto the ogive using point P
60 PG1 = Line(P,G1);
61 PG2 = Line(P,G2);
62 PG3 = Line(P,G3);
63 PG4 = Line(P,G4);
64 I1 = LineCircleIntersection(PG1,C_DA,1);
65 I2 = LineCircleIntersection(PG2,C_DA,2);
66 I3 = LineCircleIntersection(PG3,C_DB,2);
67 I4 = LineCircleIntersection(PG4,C_DB,2);
68 % from the 2 points resulting from dividing [AB] into 3 parts
69 % we draw the separations of the stones that make up the ogive
70 Dvoute1 = Line(D2,I1);
71 Dvoute2 = Line(D2,I2);
72 Dvoute3 = Line(D1,I3);
73 Dvoute4 = Line(D1,I4);
74
75 % drawings
76 drawoptions(withcolor 1.3*Grey);
77 gddDraw perpC2; gddDraw perpC1;
78 gddDraw AB;
79 gddDraw AC;
80 gddDraw BC;
81 gddDraw perpD;
82 gddDraw OrthoSign(B,DD,A2,0.2);
83
84 drawoptions(withcolor Bisque);
85
86 gddDraw BF5;
87 gddDraw perpF5;gddDraw perpF4;gddDraw perpF3;gddDraw perpF2;
88 gddDraw perpF1;
89 gddDraw OrthoSign(B,CC,A2,0.2);
90
91 drawoptions(withcolor SteelBlue withpen pencircle scaled 0.85pt);
92 gddDraw gddDrawCircleArc(C_DA,Pi/2,Pi);
93 gddDraw gddDrawCircleArc(C_DA2,Pi/2,Pi);
94 gddDraw gddDrawCircleArc(C_DB2,0,Pi/2);
95 gddDraw gddDrawCircleArc(C_DB,0,Pi/2);
96
97 drawoptions();
98 gddDraw gddDrawCircleArc(C_AB,0,-Pi/2) dashed evenly;
99 gddDraw gddDrawCircleArc(C_BA,Pi,3Pi/2) dashed evenly;
100
101 drawoptions(withcolor DarkSalmon);
102 gddDraw PG1; gddDraw PG2; gddDraw PG3; gddDraw PG4;
103
104 drawoptions(withcolor SteelBlue withpen pencircle scaled 0.85pt);
105 gddDraw Dvoute1; gddDraw Dvoute2; gddDraw Dvoute3; gddDraw Dvoute
106 4;
107 drawoptions();
108 SetPointColor(DarkRed);
109 gddDrawPoint O;
110 gddDrawPoint A;
111 gddDrawPoint B; gddDrawPoint D;
112 gddDrawPoint B2;
113
114 SetPointColor(Bisque);
115 gddDrawPoint F1; gddDrawPoint F2; gddDrawPoint F3; gddDrawPoint F
116 4;
117 gddDrawPoint F5;
118 gddDrawPoint G1; gddDrawPoint G2;gddDrawPoint G3;gddDrawPoint G4;
119 SetPointColor(1.3*Grey);
120 gddDrawPoint DD;
121 gddDrawPoint C1; gddDrawPoint C2; gddDrawPoint C3;
122 gddDrawPoint D1; gddDrawPoint D2;
123
124 SetPointColor(DarkSalmon);
125 gddDrawPoint P;
126 gddDrawPoint I1; gddDrawPoint I2; gddDrawPoint I3; gddDrawPoint I
127 4;
128 Window(-5,-7,5,5);
129 endfig;
130 end.

```

## 14.15 Pedal Triangle

Figure 21 of [3].



```

1 input geom2d;
2 gddU:=1.5cm;
3 beginfig(1);
4 numeric A[],B[],C[],E[],F[],G[],U[],W[],V[],T[],R[],P[];
5 A[1] = Point(0,0);
6 B1 = Point(1,2);
7 C1 = Point(2.5,0.2);
8 W1 = PointOf(Segment(A1,B1),0.2);
9 U1 = PointOf(Segment(B1,C1),0.23);
10 V1 = PointOf(Segment(C1,A1),0.3);
11 G1 = PointOnLineProjection(B1,Line(A1,C1));
12 F1 = PointOnLineProjection(C1,Line(A1,B1));
13 E1 = PointOnLineProjection(A1,Line(B1,C1));
14
15
16
17 for i:=0 step 3 until 3:
18
19 D[1+i] = Line(B[1+i],C[1+i]);
20 A[2+i] = AxialSymmetry(A[1+i],D[1+i]);
21
22 B[2+i] = B[1+i];
23 C[2+i] = C[1+i];
24
25 W[2+i] = AxialSymmetry(W[1+i],D[1+i]);
26 V[2+i] = AxialSymmetry(V[1+i],D[1+i]);
27 U[2+i] = U[1+i];
28
29 G[2+i] = AxialSymmetry(G[1+i],D[1+i]);
30 F[2+i] = AxialSymmetry(F[1+i],D[1+i]);
31 E[2+i] = E[1+i];
32
33 D[2+i] = Line(A[2+i],C[2+i]);
34 B[3+i] = AxialSymmetry(B[2+i],D[2+i]);
35 A[3+i] = A[2+i];
36 C[3+i] = C[2+i];
37 W[3+i] = AxialSymmetry(W[2+i],D[2+i]);
38 U[3+i] = AxialSymmetry(U[2+i],D[2+i]);

```

```

39 V[3+i] = V[2+i];
40 E[3+i] = AxialSymmetry(E[2+i],D[2+i]);
41 F[3+i] = AxialSymmetry(F[2+i],D[2+i]);
42 G[3+i] = G[2+i];
43
44 D[3+i] = Line(A[3+i],B[3+i]);
45 C[4+i] = AxialSymmetry(C[3+i],D[3+i]);
46 A[4+i] = A[3+i];
47 B[4+i] = B[3+i];
48 U[4+i] = AxialSymmetry(U[3+i],D[3+i]);
49 V[4+i] = AxialSymmetry(V[3+i],D[3+i]);
50 W[4+i] = W[3+i];
51 E[4+i] = AxialSymmetry(E[3+i],D[3+i]);
52 G[4+i] = AxialSymmetry(G[3+i],D[3+i]);
53 F[4+i] = F[3+i];
54 endfor;
55
56 for i:=1 upto 7:
57     T[i] = Triangle(A[i],B[i],C[i]);
58     Q[i] = Triangle(U[i],V[i],W[i]);
59     P[i] = Triangle(E[i],F[i],G[i]);
60     gddFill Q[i] withPen(0.5,LightGray);
61     gddDraw P[i] withPen(0.8,MediumSlateBlue);
62 endfor;
63 for i:=1 upto 7:
64     gddDraw T[i] withPen(1,black);
65 endfor;
66
67 gddDraw Polyline(U1,V2,W3,U4,V5,W6,U7) withPen(1,Peru) ;
68 gddDraw Segment(U1,U7) dashed evenly;
69 gddDraw Segment(E1,E7) dashed evenly;
70 gddDrawPoint A1; gddDrawPoint B1; gddDrawPoint C1;
71 gddDrawPoint A2; gddDrawPoint B3; gddDrawPoint C4;
72 gddDrawPoint A5; gddDrawPoint B6; gddDrawPoint C7;
73 gddLabel.lft(texttext("$A$"),A1);

```

```

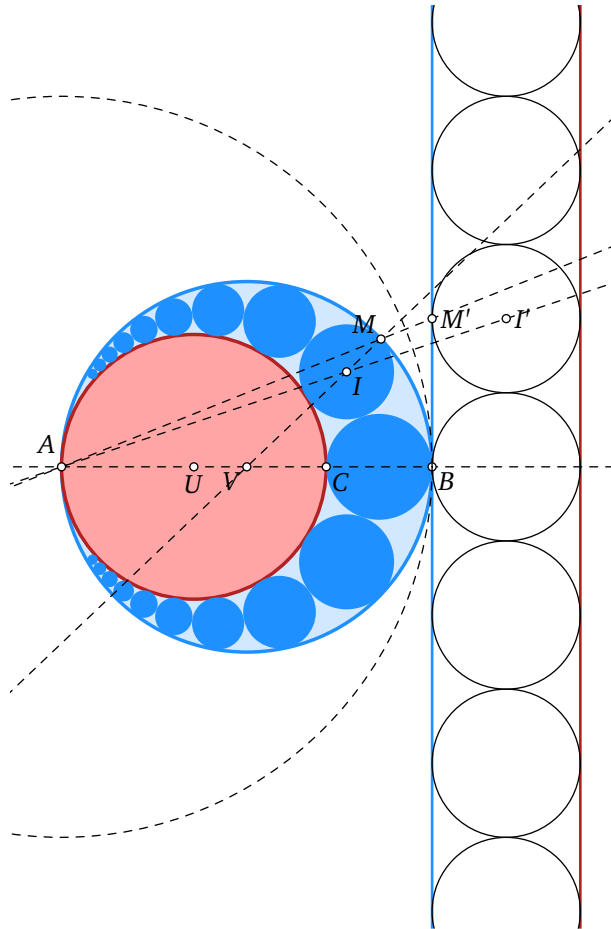
74 gddLabel.lft(texttext("$B$"),B1);
75 gddLabel.bot(texttext("$C$"),C1);
76 gddLabel.top(texttext("$A'$"),A2);
77 gddLabel.bot(texttext("$B'$"),B3);
78 gddLabel.top(texttext("$C'$"),C4);
79 gddLabel.bot(texttext("$A''$"),A5);
80 gddLabel.top(texttext("$B''$"),B7);
81 gddLabel.bot(texttext("$C''$"),C7);
82
83 gddDrawPoint U1; gddDrawPoint V1; gddDrawPoint W1;
84 gddLabel.lft(texttext("$U$"),U1) withcolor Gray;
85 gddLabel.bot(texttext("$V$"),V1) withcolor Gray;
86 gddLabel.lft(texttext("$W$"),W1) withcolor Gray;
87
88 gddLabel.lft(texttext("$U'$"),U4) withcolor Gray;
89 gddLabel.lrt(texttext("$V'$"),V2) withcolor Gray;
90 gddLabel.urt(texttext("$W'$"),W3) withcolor Gray;
91 gddLabel.urt(texttext("$U''$"),U7) withcolor Gray;
92 gddLabel.urt(texttext("$V''$"),V5) withcolor Gray;
93 gddLabel.lrt(texttext("$W''$"),W7) withcolor Gray;
94 gddDrawPoint U4; gddDrawPoint V2; gddDrawPoint W3;
95 gddDrawPoint U7; gddDrawPoint V5; gddDrawPoint W7;
96 gddDrawPoint E1; gddDrawPoint F1; gddDrawPoint G1;
97 gddLabel.lrt(texttext("$E$"),E1) withcolor MediumSlateBlue;
98 gddLabel.lft(texttext("$F$"),F1) withcolor MediumSlateBlue;
99 gddLabel.bot(texttext("$G$"),G1) withcolor MediumSlateBlue;
100
101 gddLabel.ulft(texttext("$F''$"),F7) withcolor MediumSlateBlue;
102 gddDrawPoint F7;
103 gddLabel.rt(texttext("$E''$"),E7) withcolor MediumSlateBlue;
104 gddDrawPoint E7;
105 endfig;
106 end.

```



## 14.16 Pappus Chain

Pappus chain with geometric inverse.



```

1 input geom2d;
2
3 gddU:=0.7cm;
4
5 beginfig(1);
6   r = 2.5;
7   R = 3.5;
8   A = Point(0,0);
9   C_A = Circle(A,2*R);
10  C = Point(2r,0);
11  B = Point(2R,0);
12  C_V = CircleD(A,B);
13  C_U = CircleD(A,C);
14  C_CB=CircleD(C,B);
15  V = Center(C_V);
16  U = Center(C_U);
17  AB = Line(A,B);
18  I_CU = Inverse(C_U,C_A);
19  I_CV = Inverse(C_V,C_A);
20
21  L_AB = Line(A,B);
22  Up = LinesIntersection(L_AB,I_CU);
23  Vp = LinesIntersection(L_AB,I_CV);
24  Cp0 = CircleD(Vp,Up);
25  Rp= Radius(Cp0);
26
27
28  gddAlphaFill(C_V,DodgerBlue,0.2);
29  gddDraw C_V withPen(1.3,DodgerBlue);
30
31  gddFill C_U withcolor 4.9*FireBrick;
32  gddDraw C_U withPen(1.3,FireBrick);
33  gddFill C_CB withcolor DodgerBlue;
34  gddDraw I_CV withPen(1,DodgerBlue);
35  gddDraw I_CU withPen(1,FireBrick);
36
37  Ip[0] = MidPoint(Vp,Up);
38  N=9;
39  gddDraw Cp0;

```

```

40 for i:=-N upto N:
41   if(i<>0):
42     Ip[i] = Addition(Ip[0],Point(0,i*2*Rp));
43     Mp[i] = Addition(Ip[i],Point(-Rp,0));
44     DAMp[i] = Line(A,Mp[i]);
45     DVIp[i] = Line(V,Ip[i]);
46     DAIp[i] = Line(A,Ip[i]);
47     M[i] = LineCircleIntersection(DAMp[i],C_V,2);
48     DVM[i] = Line(V,M[i]);
49     I[i] = LinesIntersection(DAIp[i],DVM[i]);
50     gddFill CircleCP(I[i],M[i]) withcolor DodgerBlue;
51     gddDraw CircleCP(Ip[i],Mp[i]);
52     if(i=1):
53       drawoptions(dashed evenly);
54       gddDraw DAIp[i];gddDraw DVM[i];gddDraw DAMp[i];
55       drawoptions();
56       gddDrawPoint Ip[i]; gddDrawPoint Mp[i];

```

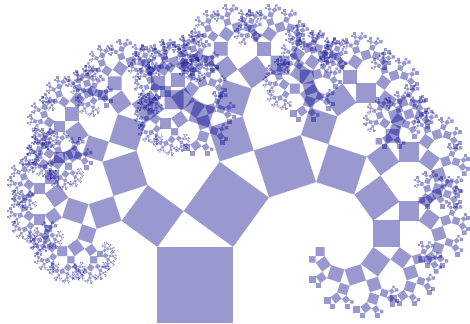
```

57     gddDrawPoint M[i]; gddDrawPoint I[i];
58     gddLabel.rt(texttext("$I'$"),Ip[i]);
59     gddLabel.lrt(texttext("$I$"),I[i]);
60     gddLabel.rt(texttext("$M'$"),Mp[i]);
61     gddLabel.ulft(texttext("$M$"),M[i]);
62   fi
63 fi
64 endfor;
65 gddDraw AB dashed evenly;
66 gddMark.bot "U";
67 gddMark.llft "V";
68 gddMark.ulft "A"; gddMark.lrt "B"; gddMark.lrt "C";
69 gddDraw C_A dashed evenly;
70
71 Window(-1,-2.5R,3*R,2.5R);
72 endfig;
73
74 end.

```

## 14.17 The Pythagoras tree using similarity

Constructing the Pythagoras tree by similarity. This code is inspired by the construction proposed by MicMath on his Twitch channel.



```
1 input geom2d;
2 warningcheck:=0;
3 angleArbre = Pi/5;
4
5 vardef PythagorasTree(expr A,B,n)=
6   save C,D,E;
7   numeric C,D,E;
```

```
8   D = CenterSimilarity(A,B,-Pi/2,1);
9   C = CenterSimilarity(B,A,Pi/2,1);
10  gddAlphaFill(Polygone(A,B,C,D), DarkBlue,0.4);
11  if(n>0):
12    E = CenterSimilarity(C,D,angleArbre,cos(angleArbre));
13    PythagorasTree(D,E,n-1);
14    PythagorasTree(E,C,n-1);
15  fi
16 enddef;
17
18 beginfig(1);
19
20 A = Point(0,0);
21 B = Point(1,0);
22
23 PythagorasTree(A,B,10);
24
25 endfig;
26 end.
```

## 15 Historique

**december 2025:** Added the `EndWindow` macro, added the center-based similarity macros `CenterSimilarity`, added the “Pythagoras Tree” example, fixed bugs, and improved the documentation. Use of `withtransparency` for the command `gddAlphaFill` when `mp-geom2d` is used with LuaTeX and `luamplib`. Version 1.4.

**may 2025:** translation into English of macros, documentation and examples.

## References

- [1] Hans Hagen et al. *The luamplib package. Use LuaTeX’s built-in MetaPost interpreter.* Version 2.34.5. Aug. 3, 2024. URL: <https://ctan.org/pkg/luamplib>.
- [2] Jens-Uwe Morawski. *The latexMP package. Interface for L<sup>A</sup>T<sub>E</sub>X-based typesetting in MetaPost.* Version 1.2.1. June 21, 2020. URL: <https://ctan.org/pkg/latexmp>.
- [3] Hans Rademacher and Otto Toeplitz. *The enjoyment of math. Translated from the German. With a new foreword by Alex Kontorovich.* English. Reprint. Princeton Sci. Libr. Princeton, NJ: Princeton University Press, 2023. ISBN: 978-0-691-24154-8; 978-0-691-24153-1. DOI: [10.1515/9780691241531](https://doi.org/10.1515/9780691241531).
- [4] Esger Renkema. *The minim-mp package. Low-level mplib integration for LuaTeX.* Version 2024/1.6. Apr. 6, 2024. URL: <https://ctan.org/pkg/minim-mp>.
- [5] The MetaPost Team and John Hobby. *The metapost package. A development of Metafont for creating graphics.* Aug. 26, 2021. URL: <https://ctan.org/pkg/metapost>.
- [6] Toby Thurston. *The Drawing-with-Metapost package. How to draw technical diagrams with MetaPost.* Apr. 16, 2023. URL: <https://ctan.org/pkg/drawing-with-metapost>.

## Index

- [\\_E, 59](#)
- [AngleBetweenVectors, 18](#)
- [AngleBisector, 14](#)
- [Arc, 46](#)
- [ArcBetweenPoints, 47](#)
- [arccos, 60](#)
- [arcsin, 60](#)
- [arctan, 60](#)
- [AxialSymmetry, 48](#)
- [Axis, 55](#)
- [AxisOfSimilitude, 28](#)
- [Barycenter, 14](#)
- [Center, 24, 30, 38](#)
- [CenterRotation, 14](#)
- [CenterSimilarity, 51](#)
- [CentralSymmetry, 49](#)
- [ch, 59](#)
- [Circle, 23](#)
- [CircleCP, 23](#)
- [CircleD, 23](#)
- [CirclesIntersection, 25](#)
- [CircleThreePoints, 24](#)
- [CircumscribedCircle, 41](#)
- [cos, 59](#)
- [CoVertex, 31](#)
- [Curve, 61](#)
- [CurveData, 45](#)
- [Directrix, 32, 35, 39](#)
- [Ellipse, 29](#)
- [EllipseF, 30](#)
- [EllipseFD, 30](#)
- [EllipseTangent, 32](#)
- [EndWindow, 10](#)
- [EscribedCircle, 42](#)
- [EulerCircle, 42](#)
- [Excentricity, 31, 34, 38](#)
- [exp, 59](#)
- [ExternalCommonTangent, 26](#)
- [ExternalEllipseTangent, 33](#)
- [Focus, 31](#)
- [Frame, 54](#)
- [FrameBox, 59](#)
- [FrameGrid, 58](#)
- [FrameMinMax, 56](#)
- [gddAlphaFill, 7](#)
- [gddArrow, 6](#)
- [gddBegin, 55](#)
- [gddClip, 10](#)
- [gddClose, 9](#)
- [gddCouleurPoint, 9](#)
- [gddDraw, 6](#)
- [gddDrawCircleArc, 46](#)
- [gddDrawPoint, 9](#)
- [gddDrawPoints, 10](#)
- [gddEnd, 55](#)
- [gddExtensionDroite, 20](#)
- [gddFill, 6](#)
- [gddHatch, 7](#)
- [gddInPlace, 5](#)
- [gddLabel, 53](#)
- [gddMark, 53](#)
- [gddObjectDrawing, 8](#)
- [gddPointeType, 9](#)
- [gddTaillePoint, 9](#)
- [gddU, 5](#)
- [gddW, 5](#)
- [gddXlabel, 55](#)
- [gddYlabel, 55](#)
- [Graduations, 57](#)
- [HatchPattern, 8](#)
- [Homothety, 48](#)
- [HyperbolaAsymptote, 39](#)
- [HyperbolaF, 38](#)
- [HyperbolaFD, 38](#)
- [HyperbolaVertex, 39](#)
- [InscribedCircle, 41](#)
- [InternalCommonTangent, 26](#)
- [Inverse, 50](#)
- [IsoBarycenter, 14](#)
- [Length, 13](#)
- [Line, 20](#)
- [LineCircleIntersection, 25](#)
- [LineEquation, 21](#)
- [LineMark, 52](#)
- [LinesIntersection, 22](#)
- [ln, 59](#)

Mark, 52  
MidPoint, 13  
  
Norm, 18  
  
Orthocenter, 41  
OrthoSign, 51  
  
PairImp, 15  
PairTOPoint, 15  
ParabolaFD, 34  
ParabolaVertex, 35  
ParallelLine, 22  
Path, 45  
PathTag, 53  
PerpendicularLine, 21  
Pi, 59  
Plot, 60  
Point, 11  
PointImp, 15  
PointInBasis, 12  
PointIntersection, 47  
PointLineDistance, 21  
PointLineRelativeDistance, 21  
PointOf, 16  
PointOnLineProjection, 21  
PointSum, 13  
PointTOPair, 15  
PolarCurve, 62  
PolarPoint, 11  
Polygon, 43  
PolygonNumberOfSides, 44  
PolygonPoint, 44  
Polyline, 45  
PrincipalCircle, 39  
  
RadicalAxes, 27  
RadicalCenter, 28  
Radius, 24  
RegularPolygon, 43  
ReportOnLine, 22  
Rotation, 13  
  
ScalarProduct, 18  
ScalarVector, 17  
Segment, 19  
SegmentLength, 19  
SegmentTOVector, 20  
SemiHyperbola, 39  
SemiMajorAxis, 31  
SemiMinorAxis, 31  
SetConicCoef, 34  
SetLineExtension, 20  
SetPointColor, 9  
SetPointSize, 9  
SetPointType, 10  
sh, 59  
SideAxis, 57  
SideGraduations, 57  
sin, 59  
Slope, 32, 35, 38  
  
tan, 59  
th, 59  
Triangle, 40  
TriangleArea, 41  
  
Units, 58  
  
Vector, 16  
VectorField, 62  
VectorFieldDD, 63  
VectorP, 16  
VectorSubtract, 17  
VectorSum, 17  
Vertex, 30  
  
Window, 10  
withPen, 9  
  
Xcoordinate, 12  
XcoordinateSum, 12  
  
Ycoordinate, 12  
YcoordinateSum, 13