Excursions in METAPOST

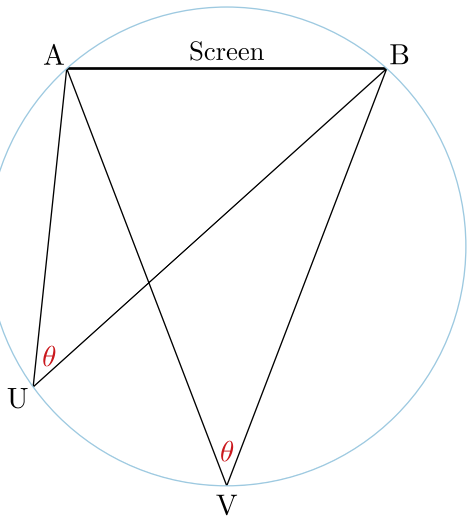
Toby Thurston

October 2022 —

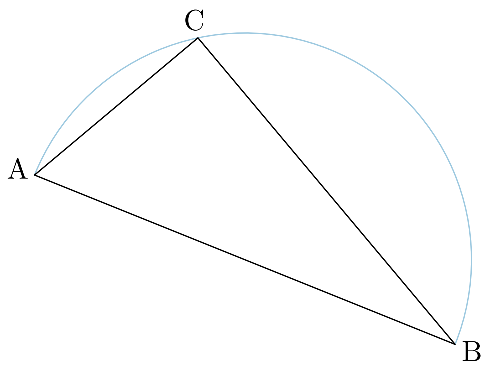
This example document includes geometric illustrations inspired by *Excursions in Geometry*, C. Stanley Ogilvy, OUP 1968. The illustrations are presented roughly in the same order as the book, with notes about how you can use METAPOST to produce similar. The section headings also approximately follow the book. You might like to read the PDF of this document side by side with the source code, so that you can see how each illustration is done. Each illustration is included as in-line METAPOST code, there are no external graphics files used.

1 A bit of background

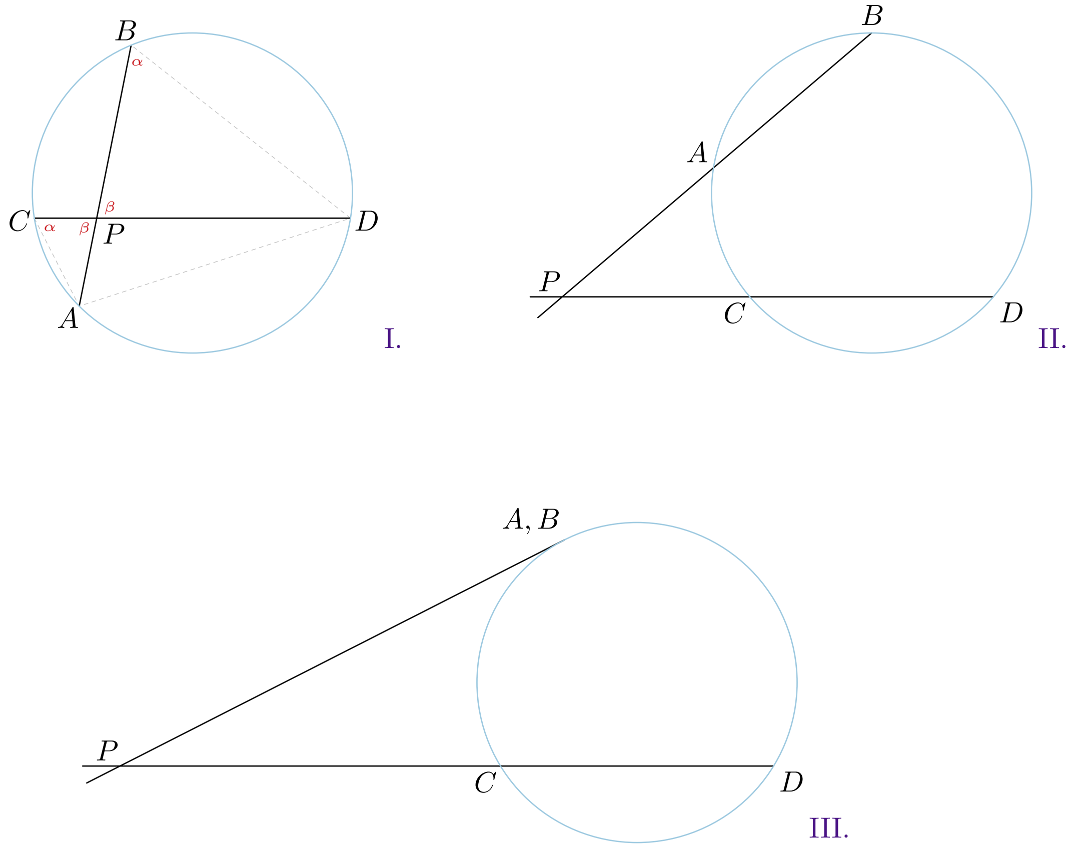
Ogilvie starts with a review of some circle theorems. In fact most of the book is about circles in one way or another. In this first diagram, you are given the width AB of the screen and the ideal viewing angle θ. The METAPOST code works out the rest from that, including a useful routine for a circle through three points.



Here θ is half the angle measured by the intercepted arc, which gives us the useful corollary that any angle inscribed in a semicircle is a right angle, and conversely that if you can show that some angle ACB is a right angle, then the semicircle drawn with AB as a diameter must pass through C.



A basic theorem: If two chords intersect, the product of the lengths of the segments of the one equals the product of the lengths of the segments of the other. In all three cases below you have $PA/PD = PC/PB$ by similar triangles, hence $PA \cdot PB = PC \cdot PD$.

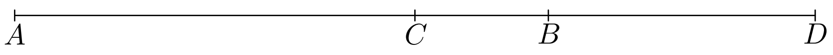


This example also shows how to arrange sub-figures into one.

2 Harmonic division and Apollonian circles

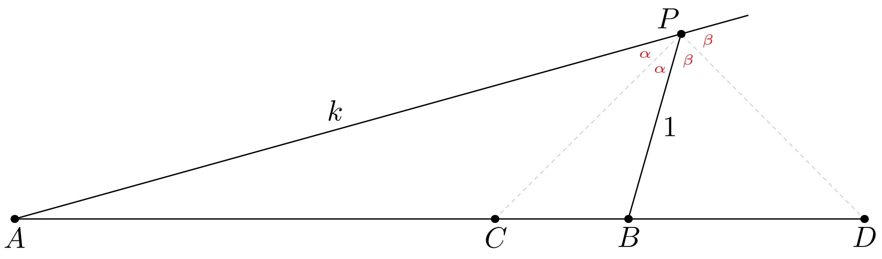
Can we find C and D on the line AB so that $AC/CB = AD/BD$?

Yes:

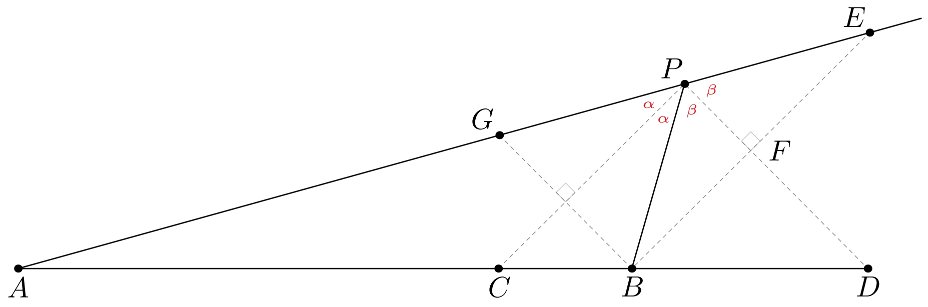


Theorem. The bisector of any angle of a triangle divides the opposite side into parts proportional to the adjacent sides.

So given P, with $AP = k$ and $BP = 1$:



We have $AC/CB = AP/BP = k$ from the interior angles and $AD/BD = AP/BP = k$ from the exterior pair. The proof looks like this:



Since the lines PC and PD are bisectors, then we have $2\alpha + 2\beta = 180°$, hence $\alpha + \beta$ is a right angle, so if we draw BE parallel to PC it will be perpendicular to PD, and hence the two triangles PFE and PFB are congruent, so that $PE = PB$. But the parallel lines cut off proportional segments, so that

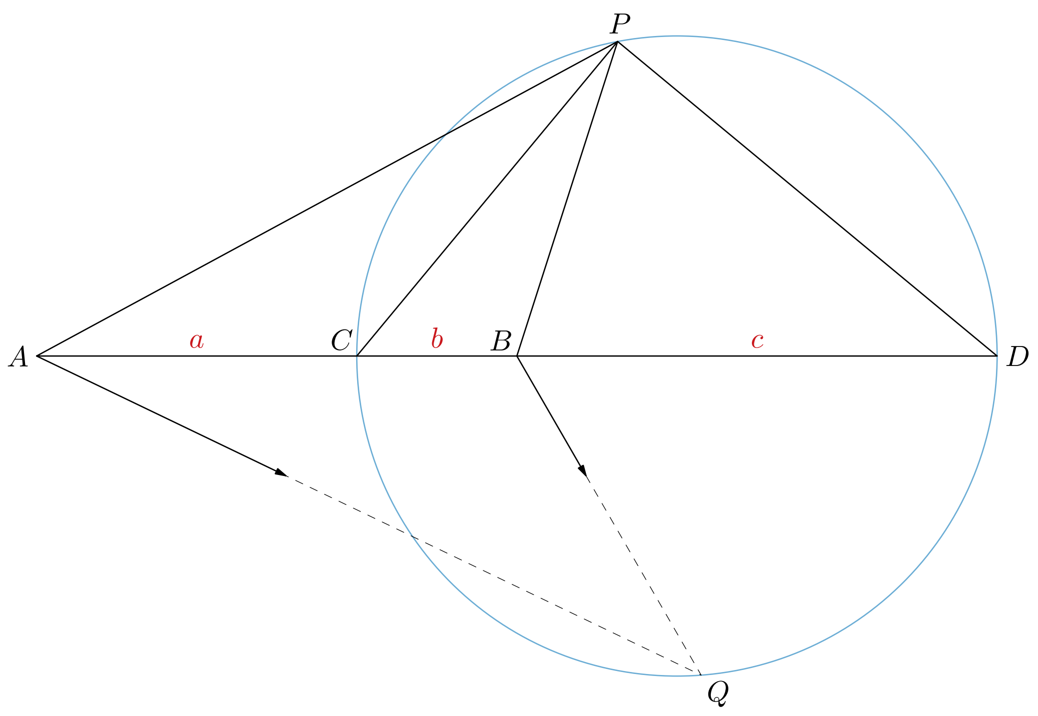
$$\frac{AC}{CB} = \frac{AP}{PE} = \frac{AP}{BP}$$

as required. And using the exterior angle

$$\frac{AD}{BD} = \frac{AP}{GP} = \frac{AP}{BP}$$

2.1 Applied Apollonian example

Given A, B, k, and course of the target, find intersection point Q.



But how can you find C and D in METAPOST given A, B, and k? Using a, b, and c for the lengths as shown above, then using the mediation syntax, point C is `(a/(a+b))[A, B]` and D is `((a+b+c)/(a+b))[A, B]`, so what are these fractions in terms of k? We know that $AC/CB = AD/BD = k$ so we have $a/b = (a+b+c)/c = k$ which we can develop as follows.

Starting with $a/b = k$, add 1 to each side, $a/b + b/b = k + 1$, hence $(a+b)/b = k+1$. We can then divide the first equation by the last to get $a/(a+b) = k/(k+1)$.

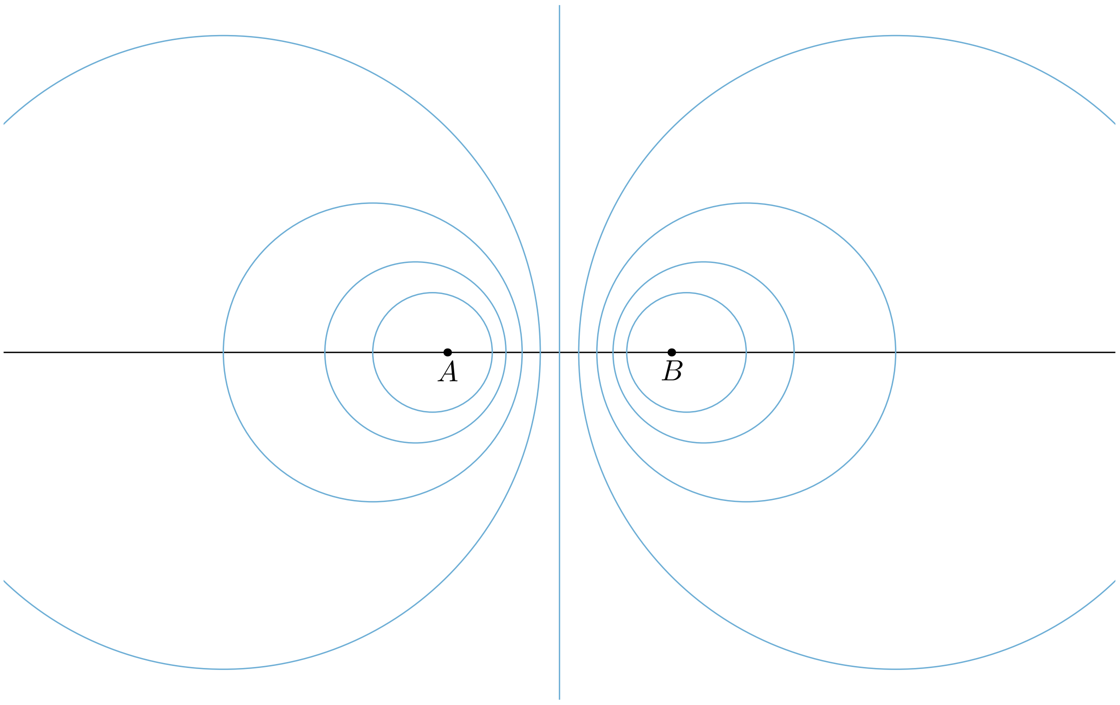
Similarly, starting with $(a+b+c)/c = k$, take 1 away from each side to get $(a+b+c)/c - c/c = k - 1$, hence $(a+b)/c = k-1$. And again dividing the first by the last gives $(a+b+c)/(a+b) = k/(k-1)$.

So our METAPOST expressions for points C and D can be written like this:

```
C = (k/(k+1))[A, B]; D = (k/(k-1))[A, B];
```

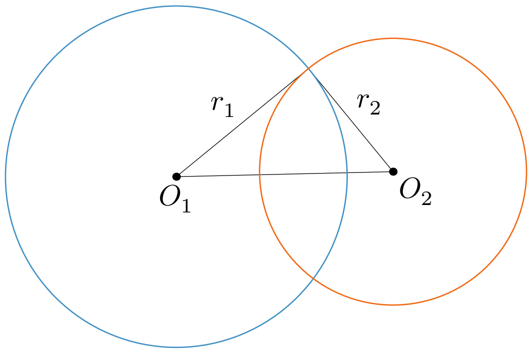
If $k = 1$ you will get a "Division by zero" error from the second expression. This corresponds to D at infinity and C half way from A to B. The circle through these two points can be thought of as the perpendicular bisector of A and B. If you have $0 < k < 1$ then the circle will be round A not B. If $k < 0$ the positions of C and D will be swapped, and there is another error when $k = -1$.

2.2 Coaxial families



The circles on the left are drawn with $0 < k < 1$ and those on the right with $k > 1$. The circles drawn with a given k is the mirror image of one drawn with $1/k$; so the smallest circle around A is drawn with $k = 1/4$ and the corresponding smallest circle around B has $k = 4$.

The tangents at the intersections of orthogonal circles are at right angles to each other.

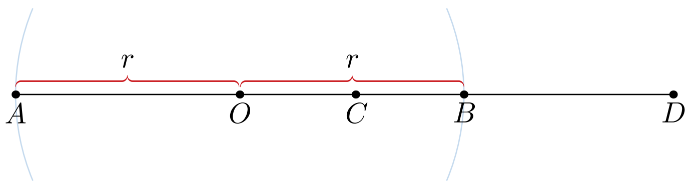


Given O_1, r_1, and r_2, you can find O_2 in METAPOST with:

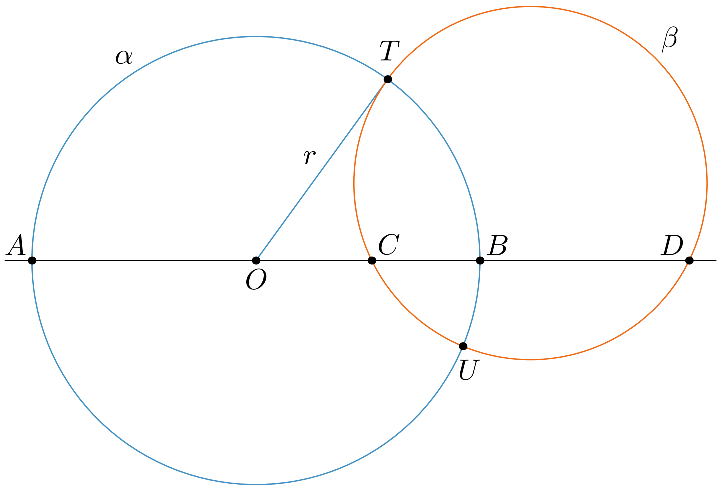
```
c1 = fullcircle scaled 2 r1 shifted o1;
o2 = unitvector (direction t of c1) scaled r2 shifted point t of c1;
```

where t is the desired time on C_1 for a point of intersection.

If you consider a circle with radius r and diameter AB, where AB is divided harmonically then you have (with red braces for the labels):



and so you can write $\frac{AC}{CB} = \frac{AD}{BD}$ as $\frac{r+OC}{r-OC} = \frac{OD+r}{OD-r}$ and hence $r^2 = OC \cdot OD$. And this expression is therefore *equivalent* to the statement that the division is harmonic, and if a circle is drawn with AB as a diameter, *any* circle drawn through C and D will be orthogonal.



Two useful routines are used in this drawing.

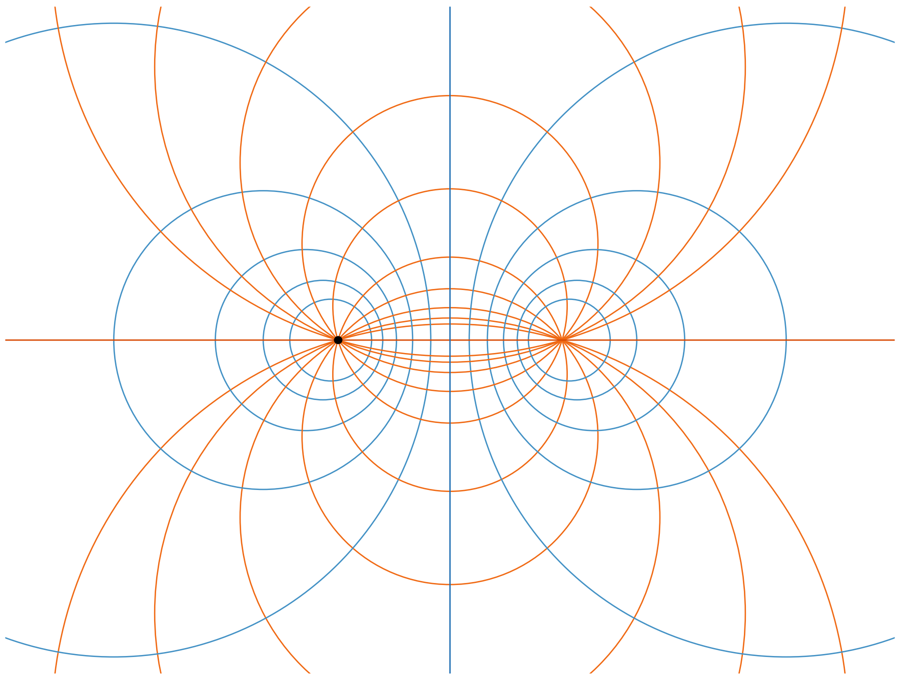
- Return the path of a line through two points, with a small overlap at each end.

```
vardef through(expr a, b, o) = save d; numeric d;
    d = abs(b-a); (1+o/d)[b,a] -- (1+o/d)[a,b]
enddef;
```

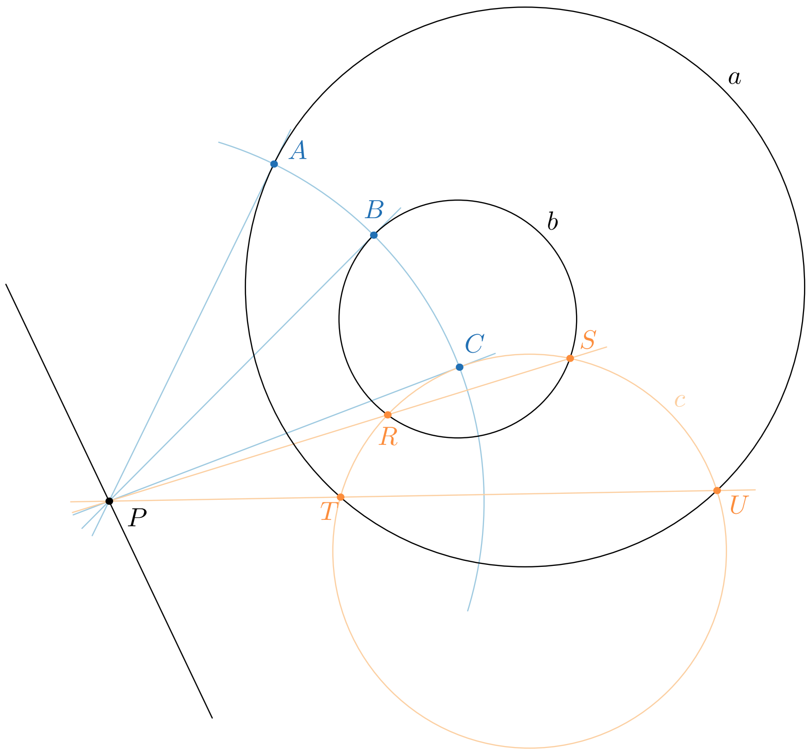
- Return the path of the (unique) circle through three points.

```
vardef circle_through(expr a, b, c) = save o; pair o;
    o = whatever * (b-a) rotated 90 shifted 1/2[a,b]
      = whatever * (c-b) rotated 90 shifted 1/2[b,c];
    fullcircle scaled 2 abs(a-o) shifted o
enddef;
```

Intersecting orthogonal circles.



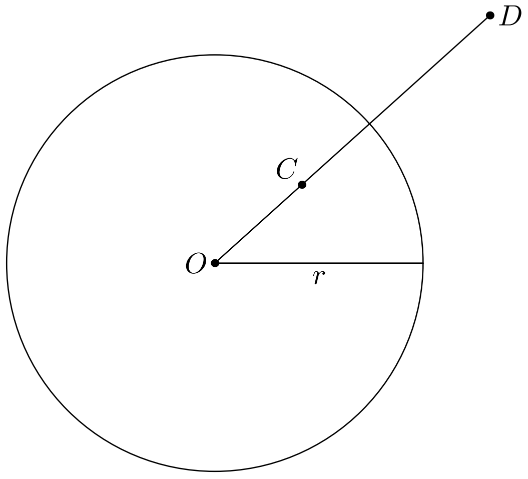
Radical axis of non-intersecting circles



3 Inversive geometry

3.1 Transformations

Given a circle of radius r, let each point C inside the circle be transformed to a point D outside the circle in such a way that the distances comply with $OC \cdot OD = r^2$ and the line OC passes through D.



For METAPOST the direct way to find D given O, C, and r is to calculate

```
D = O + unitvector(C-O) scaled r * r / abs (C-O);
```

although if you keep O at the origin this can be simplified to

```
D = unitvector(C) scaled r * r / abs C;
```

but with the default number system this risks an overflow if $r > \sqrt{2^{15}} \approx 181$, so some safer approach is more helpful for larger circles. Examining `plain.mp` shows that `unitvector` is a macro defined like this:

```
vardef unitvector primary z = z/abs z enddef;
```

which suggests this alternative (safer) approach for inverting C to find D:

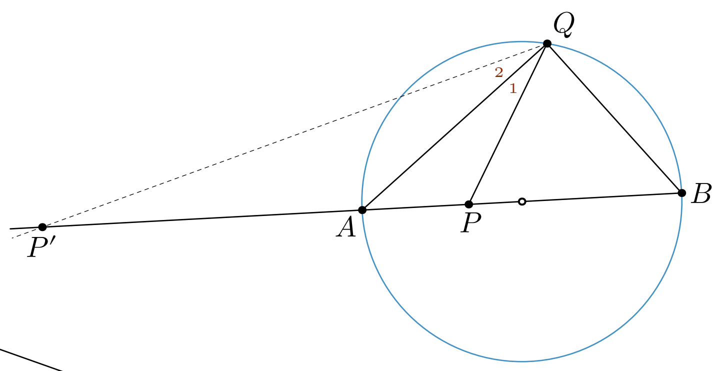
```
numeric s; s = r / abs C; D = C * s * s;
```

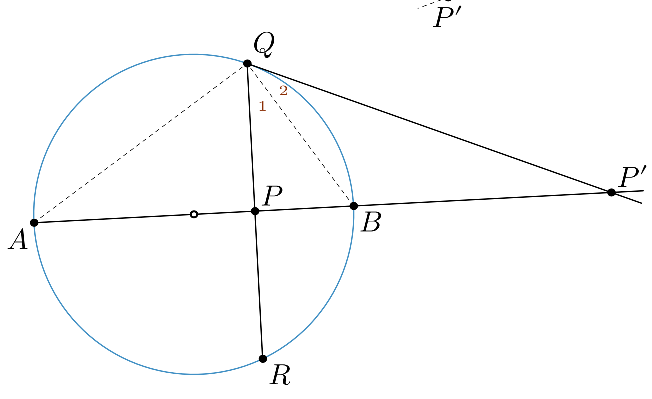
This works well provided that $|C| > \frac{1}{180}r$, which is usually the case. A more general macro to invert a point p in the circle centred at o with radius r might be:

```
vardef invert(expr p, o, r) =
    save t; pair t; t = p - o;
    save s; numeric s; s = r / abs t;
    o + t * s * s
enddef;
```

You could also consider checking that $|t| > 0$ and that s was not too large.

There are other ruler-and-compass constructions to find an inverse point, but they do not always lead to more efficient methods in METAPOST.





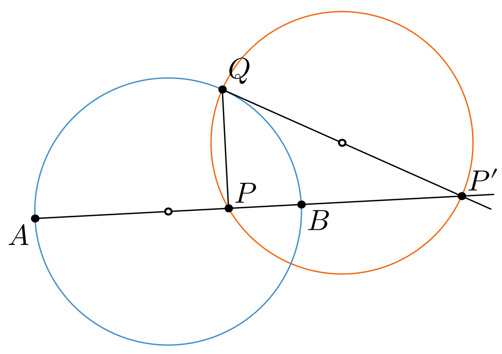
The first method here is to construct $\angle 2$ at Q, such that $\angle 2 = \angle 1$, and the dotted line cuts AB in P'. In METAPOST this is

```
P' = whatever[A, B] = whatever[Q, P reflectedabout(A, Q)];
```

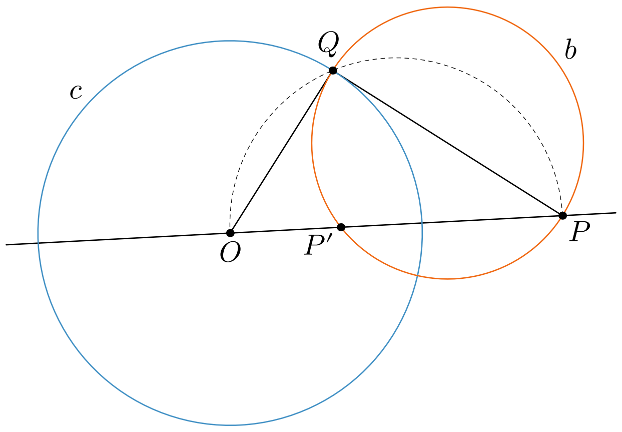
The second is to construct QR perpendicular to the diameter through P, and then the tangent at Q cuts AB in P' which, given the circle c centred at the origin, and the point P, translates into something like this:

```
numeric t; (t, whatever) = c intersectiontimes
                                P -- 200 unitvector(P) rotated 90 shifted P;
P' = whatever[A,B] = whatever * direction t of c shifted point t of c;
```

Drawing a circle through Q, P, and P' shows why this works. The tangent to one circle is the radius of the other, so AB is divided harmonically by P, and P'.



If the point is outside the circle, you need to construct the tangent first, then the circle b cuts OP at the inverse point P':



There is a neat METAPOST idiom to draw the semi-circle between O and P

```
path a; a = halfcircle zscaled (P-O) shifted 1/2[P,O];
```

and if O is really the origin, you can abbreviate that to

```
path a; a = halfcircle zscaled P shifted 1/2 P;
```

Then, assuming path c is the blue circle above, point Q is

```
pair Q; Q = a intersectionpoint c;
```

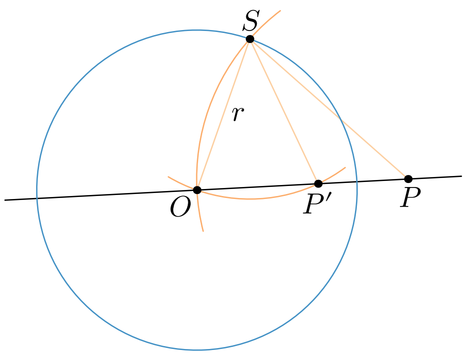
and point P' is:

```
path b; b = fullcircle scaled abs(Q-P) shifted 1/2[Q, P];
pair P'; P' = b intersectionpoint (O--P);
```

although you could do without the circle b by using the normal idiom to find the closest point to Q on OP:

```
pair P'; P' = whatever[O, P]; Q-P' = whatever * (P-O) rotated 90;
```

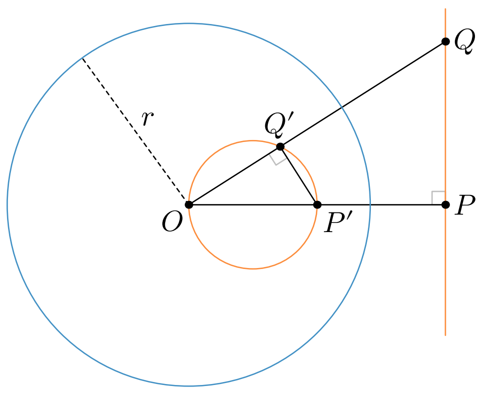
This is all interesting geometry, but if you just want to find the inverse point then the `invert` macro given earlier in this section also works well for points outside the circle.



$\leftarrow$ Similarly, if you are working with ruler-and-compasses, you can find the inverse of an external point P by drawing two arcs as shown, on the left. But this is less convenient in METAPOST, primarily because you need to make sure that you find the correct intersections of the two arcs, so the approach is less "robust" than doing the calculations. The construction works because we have $OP/r = r/OP'$, hence $OP \cdot OP' = r^2$ as before.

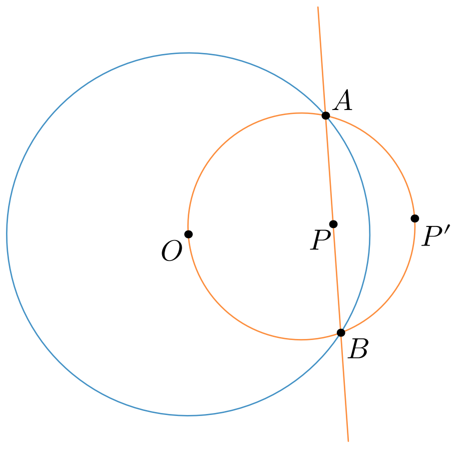
3.2 Invariants

Inversion of a straight line outside the circle of inversion.



Here we have $OP \cdot OP' = r^2$ and $OQ \cdot OQ' = r^2$ so $OP/OQ = OP'/OQ'$, hence the triangles OPQ and $OQ'P'$ are similar and $\angle OQ'P'$ must always be a right angle, since OP is perpendicular to the line, but Q is any point on the line, so the line must invert to a circle through O.

Inversion of a line that cuts the circle of inversion.

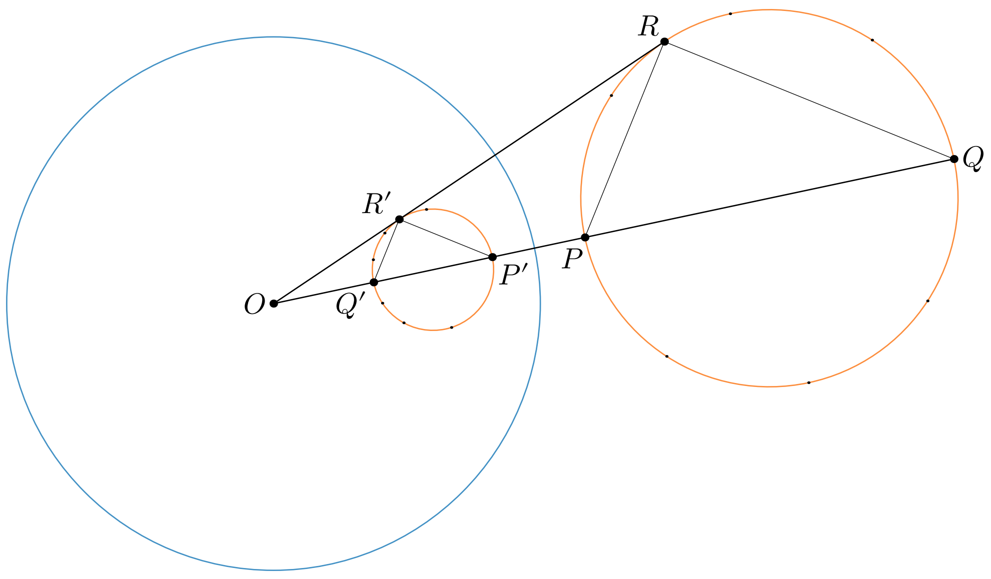


Note that the points A and B are invariant. Using the `invert` macro already described, you can find P' directly and draw the inverted line as a circle with OP' as a diameter. But it is also possible to find the points A and B as the points where the line through P that is perpendicular to OP cuts the circle of inversion, and then draw a circle through O, A, and B using a routine like this:

```
vardef circle_through(expr A, B, C) = save o; pair o;
    o = whatever * (A-B) rotated 90 shifted 1/2 [A,B]
      = whatever * (B-C) rotated 90 shifted 1/2 [B,C];
    fullcircle scaled 2 abs (A-o) shifted o enddef;
```

then P' is the intersection of this circle with the line through OP.

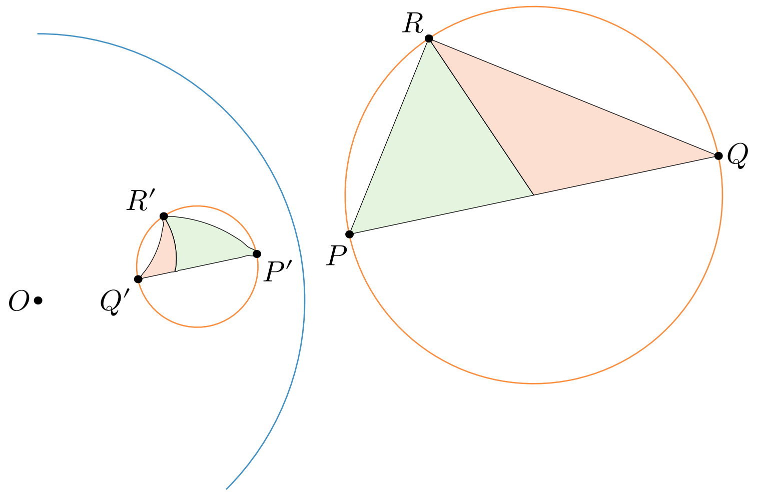
Inversion of a circle outside the circle of inversion



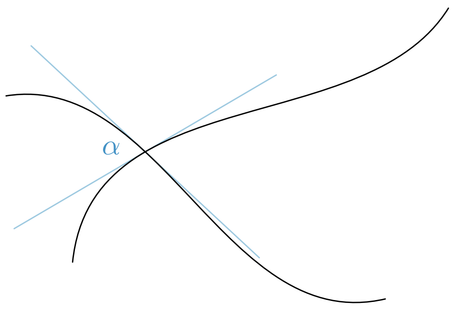
Similar triangles again show that $\angle P'R'Q'$ is a right angle, which proves that the inversion of the circle is also a circle. It is sufficient therefore to find P and Q as the diameter of the circle-to-be-inverted that passes through O (the centre of the circle of inversion), and then the image is the circle drawn with the inverted points P' and Q' as a diameter. However you sould beware that the points of the source circle are not mapped equally to the image. The small dots try to show this in the picture above. With source circle s, and a circle of inversion at z_0 with radius r, you can invert s to path s' like this:

```
s' = for i=0 upto 7: invert(point i of s, z0, r) .. endfor cycle;
```

The resulting path should look circular but will run in the opposite direction to the source path (as you might expect since inversion is a form of reflection). Note also that the triangle PQR does *not* invert to the triangle $P'Q'R'$, and the midpoint of P and Q is not the midpoint of P' and Q'.



And even though you have to look quite hard, you can see from the curved image of the triangle PQR that what do get preserved by inversion, are the angles between curves. At least in the sense of the angles between the tangents at the intersection of two curves, as here:



The METAPOST interest here is drawing the tangents at the point of intersection. `(t, u) = a intersectiontimes b` conveniently gives you the "time" along each path, then the tangent is

```
(- direction t of a -- direction t of a) shifted point t of a
```

except that the length of the direction vectors depends on the exact placement of the control points, so you if you prefer precision over brevity you could try:

```
(unitvector(-direction t of a) -- unitvector(direction t of a))
    scaled 64 shifted point t of a
```