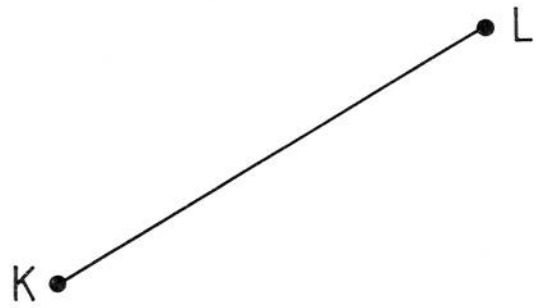
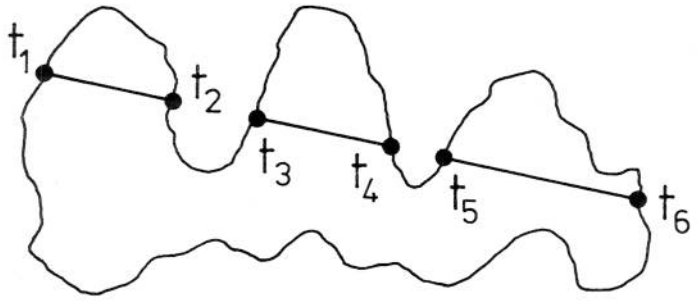
3
Points, line segments and arcs

3.1 Representation of a Line Segment



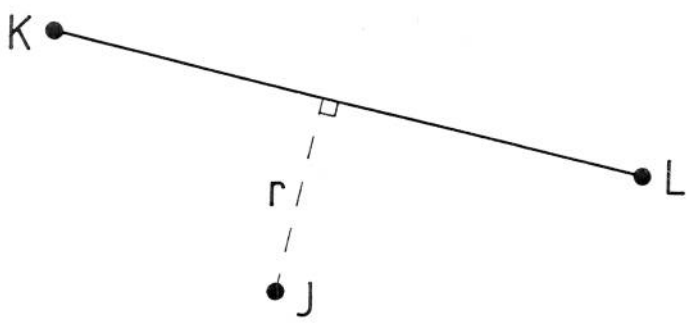
The representation of straight line segments is achieved most simply by storing each segment's endpoints. An alternative is to store the parametric equation of the infinite straight line and the values of the parameter, t, which bound the segment. This is of particular value where a number of segments of the same line are required:



It allows the intersections to be generated in any order and then sorted, segments being then described by consecutive pairs of values.

Both these approaches extend simply to lines in three dimensions. The second is useful for holding the intersections of a ray with the objects in a three-dimensional scene.

3.2 Distance from a Point to a Line Segment



The distance from a point to a line segment is the distance to the line only if the normal from the point to the line strikes the line between the segment endpoints. Otherwise it is the distance from the point to the nearest segment endpoint.

It is convenient to consider the line as the interval t=0 to t=1 of an infinite parametric line:

$$x = x_K + t(x_L - x_K)$$

$$y = y_K + t(y_L - y_K)$$

If we then calculate the value of t where the normal from the point J strikes the line, values between 0 and 1 indicate that the closest point is on the segment, values below 0 indicate that K is the closest point, and values above 1 indicate that L is the closest point. The value of t is found from

$$t = \frac{-[(x_K - x_J)(x_L - x_K) + (y_K - y_J)(y_L - y_K)]}{(x_L - x_K)^2 + (y_L - y_K)^2}$$

and if the nearest point is in the segment, the distance to it is:

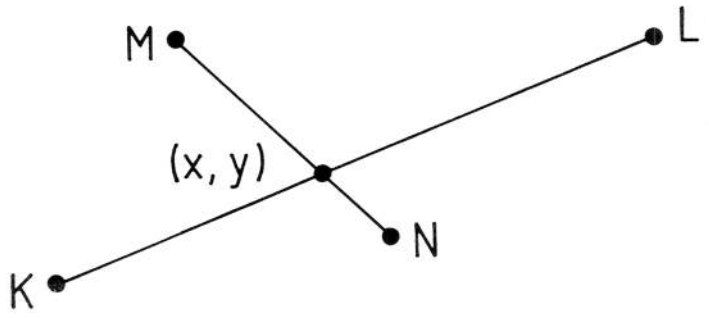
$$r = \sqrt{\{[(x_K - x_J) + t(x_L - x_K)]^2 + [(y_K - y_J) + t(y_L - y_K)]^2\}}$$

By truncating the value of t to be between 0 and 1 we can use this formula in all three cases, coded as follows:

```
XKJ = XK - XJ
YKJ = YK - YJ
XLK = XL - XK
YLK = YL - YK
DENOM = XLK*XLK + YLK*YLK
IF(DENOM.LT.ACCY) THEN
        R = SQRT(XKJ*XKJ + YKJ*YKJ)          Segment ends coincide
ELSE
        T = -(XKJ*XLK + YKJ*YLK)/DENOM       Parameter
        T = AMIN1(AMAX1(T,0.0),1.0)          Truncate to ends
        XFAC = XKJ + T*XLK
        YFAC = YKJ + T*YLK
        R = SQRT(XFAC*XFAC + YFAC*YFAC)
ENDIF
```

Dealing separately with the case where K is the nearest point may be slightly more efficient, but with some loss of simplicity. If it is possible to work with squared distances throughout a problem, then the calls to the costly SQRT function can be avoided.

3.3 Intersection of Two Line Segments



This problem looks deceptively simple. However, the algebra that yields the intersection point directly is inefficient if many of the line segment pairs that are to be compared do not, in fact, intersect. Then it is better to determine whether an intersection is even possible before going on to calculate the coordinates of the intersection. We will first show the direct approach, which in

any case leads to the shortest code.

Consider two line segments KL and MN. These will normally be specified by their endpoint coordinates, though if they are specified by the parametric equations of two infinite lines along with two pairs of parameter values the problem is greatly simplified, as such parametric equations have then to be generated from the endpoint coordinates anyway. The endpoints might, for instance, be part of polygons being processed, of which the endpoints would be vertices. One possible way to find the intersection between them is to generate the equations of the corresponding unbounded lines, and find the intersection between them using the method described in Section 1.5. Unfortunately, it is not then particularly easy to decide whether the intersection of the unbounded lines lies within the segments. Instead we determine the parametric equations of the lines in such a way that the line KL has a parameter s running from 0 at K to 1 at L, and MN has a parameter t running from 0 at M to 1 at N. The solution of the set of simultaneous equations then gives the intersection point as:

$$s = \frac{(x_N - x_M)(y_M - y_K) - (y_N - y_M)(x_M - x_K)}{(x_N - x_M)(y_L - y_K) - (y_N - y_M)(x_L - x_K)}$$

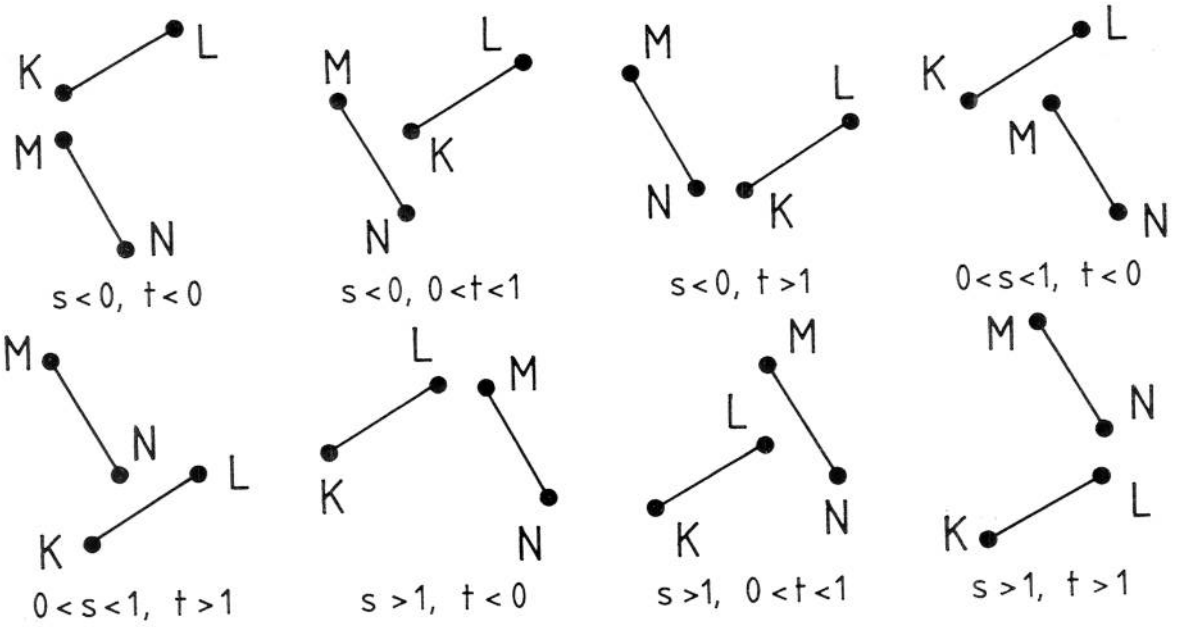
$$t = \frac{(x_L - x_K)(y_M - y_K) - (y_L - y_K)(x_M - x_K)}{(x_N - x_M)(y_L - y_K) - (y_N - y_M)(x_L - x_K)}$$

If the values of both s and t are in the range 0 to 1, then the intersection is within both line segments, and the actual coordinates can be found from either of the parameters. For example:

$$x = x_K + (x_L - x_K)s$$

$$y = y_K + (y_L - y_K)s$$

Even if s and t are outside the range 0 to 1, they may still be useful in some circumstances to classify the relationship between the line segments, of which there are eight possible categories:



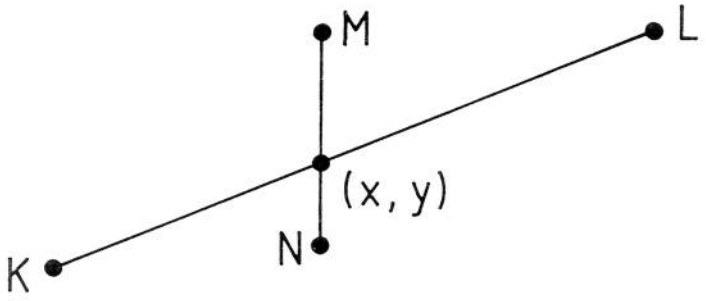
Finding the intersection (if it exists) may be coded:

```
      XLK = XL - XK
      YLK = YL - YK
      XNM = XN - XM
      YNM = YN - YM
      XMK = XM - XK
      YMK = YM - YK

      DET = XNM*YLK - YNM*XLK
      IF(ABS(DET).LT.ACCY) THEN
              ..... The two line segments are parallel

      ELSE
              DETINV = 1.0/DET
              S = (XNM*YMK - YNM*XMK)*DETINV
              T = (XLK*YMK - YLK*XMK)*DETINV
              IF (S.LT.0.0.OR.S.GT.1.0.OR.
     &                T.LT.0.0.OR.T.GT.1.0) THEN
                      ..... Intersection not within line segments

              ELSE
                      X = XK + XLK*S
                      Y = YK + YLK*S
```

```
                ENDIF
        ENDIF
```

If this procedure is being used to find the intersections of many line segments with one single line segment then two of the differences (eg XLK and YLK) need not be recomputed for each comparison.

If an appreciable number of the lines being compared are either horizontal or vertical, then it will be worth branching to much simpler comparisons with such lines. For instance, if the line MN is vertical



then the intersection point is given by

$$x = x_M$$

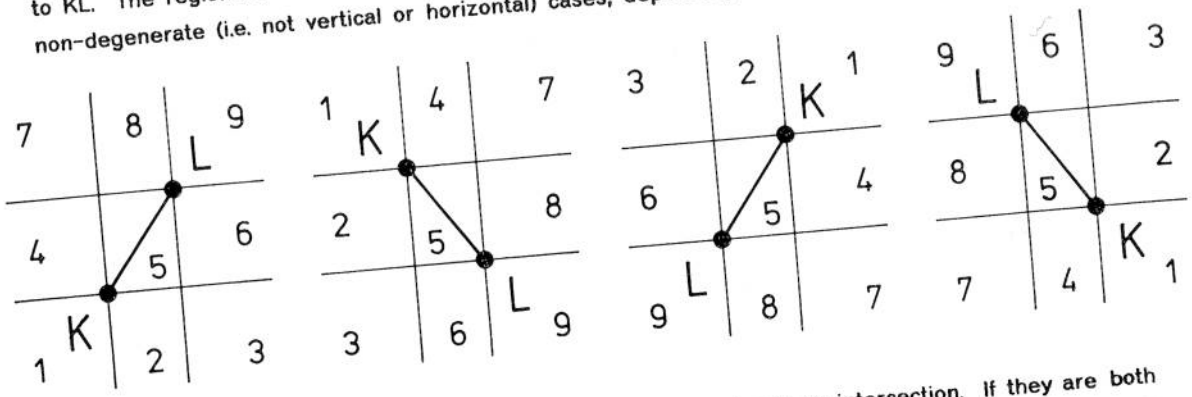
$$y = y_K + \frac{(x_M - x_K)(y_L - y_K)}{(x_L - x_K)}$$

which saves seven multiplications and a division over the general method. The value of y can be used without difficulty to determine whether the intersection point is within the line segments. When coding this check for near-zero values of $(x_L - x_K)$; this corresponds to a near vertical line KL.

Unless the reader has a lot of line segments to compare, and is therefore particularly concerned with efficiency, ignore the following; the preceding methods should be quite adequate. In many cases, such as finding the intersections between two polygons, we are concerned to test a single line (one of the sides of the first polygon) against a set of lines (all of the sides of the second polygon). This was mentioned briefly above. Often, when we do this, we expect the majority of the comparisons to yield no intersections between the segments; the line segments may be very distant. In this case the first method does a great deal of unnecessary arithmetic. We will now show how to cull most of these trivial cases by direct comparisons of the endpoints involving only subtractions. This is, in effect, a variation on the idea of *boxing tests*, discussed later in Section 3.7. The

effectiveness of tests of this sort depends on the orientation of the line segments in the coordinate system. For the method to be described we can only *guarantee* to reject the possibility of intersection when circles drawn with each line segment as a diameter are disjoint.

Consider the lines KL and MN as above. Let KL be the candidate to be compared with a number of different lines MN. The technique is to classify the endpoints of MN into nine regions with respect to KL. The regions are oriented with KL, not with the coordinate directions, and so there are four non-degenerate (i.e. not vertical or horizontal) cases, depending on the orientation of KL:



If both point M and point N fall into region 1, then there is clearly no intersection. If they are both in 5, then they must be tested again. In certain cases, such as M in region 5 and N in region 7, then only one further test need be made (on M in this case). In all cases (even if both M and N are in region 5) we have some information about the situation, and need not start the procedure given at the beginning of this section. In fact, all the cases where further information is required can be satisfied by the test to discover on which side of an unbounded line a point lies. The possible outcomes, and the further tests required to resolve inconclusive outcomes, can be tabulated and hence coded as a lookup table:

Region in which point M lies

1	2	3	4	5	6	7	8	9	
1	1	1	1	12	3	1	4	7	1
1	1	1	4	8	1	4	2	5	2
1	1	1	4	8	1	7	5	1	3
1	3	3	1	9	2	1	1	6	4
12	10	10	11	12	10	11	11	12	5
4	1	1	2	8	1	5	5	1	6
									7

Region in which point N lies

1	lines definitely do not intersect
2	lines definitely intersect
3	lines intersect if K is on left of MN
4	lines intersect if K is on right of MN
5	lines intersect if L is on left of MN
6	lines intersect if L is on right of MN
7	lines intersect if K and L are on opposite sides of MN
8	lines intersect if M is on left of KL
9	lines intersect if M is on right of KL
10	lines intersect if N is on left of KL
11	lines intersect if N is on right of KL
12	lines intersect if M and M are on opposite sides of KL

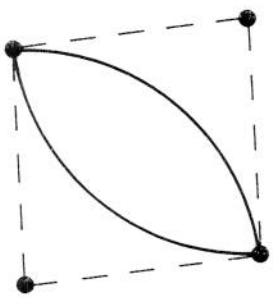
The test to decide which side of a line segment a point is on is derived from the equation for the area of a triangle (Section 4.1). Suppose that it is necessary to find which side of KL the point M lies. This can be found from:

$$r = (x_K - x_M)(y_L - y_M) + (x_L - x_M)(y_K - y_M)$$

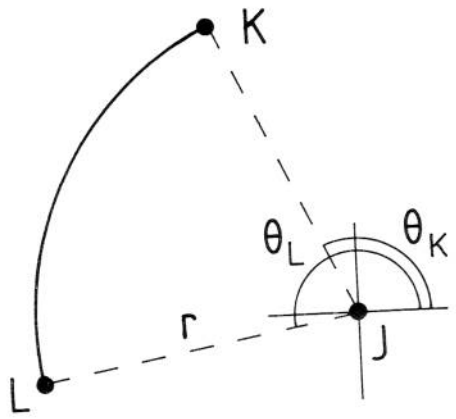
If r is negative, then the point is to the left of the line, if r is positive then the point is to the right. Note that r is not the true distance from the point to the line, but is proportional to it. If we have calculated the two values of r for a point to the left and a point to the right of a line the absolute values of r can be used proportionately to divide the distance between the two points, thus giving the coordinates of the point of intersection.

3.4 Representation of an Arc

Arcs are more complicated to represent than line segments. It is possible to store the endpoints and the radius of an arc. The ambiguity in this representation



can be avoided by a conventional sign attached to the value of the radius. Clearly the points must not be further than twice the radius apart, as then it would be impossible to draw the arc. Also the arc must never subtend an angle greater than 180^{o}, otherwise ambiguity is reintroduced. This representation can be useful, particularly for the input of contours from, say, some type of digitising device such as a tablet. The problem with it is that it does not facilitate the calculation of the centre, which is essential for most applications (see Section 2 .7). It is possible to add the centre to the data held describing the arc. This corresponds well to some numerically controlled machine tool languages, for instance, but it is bulky, and the consistency of an arc's parameters must be checked; the three points need not represent an arc at all.

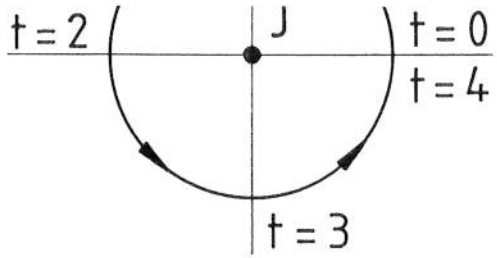


An obvious alternative is to store the arc's centre, radius, and two angles corresponding to the end points. So long as a conventional direction is established (preferably anticlockwise) this allows arcs through any angle with no danger of inconsistency or ambiguity. The drawback is that the sine and cosine functions must be used, at some expense in processor time, to calculate the endpoint coordinates.

$$x_K = x_J + r\cos(\theta_K)$$

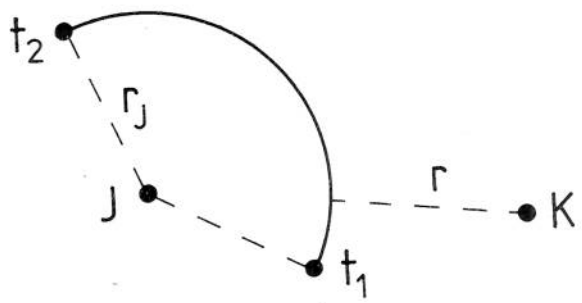
$$y_K = y_J + r\sin(\theta_K)$$

As an alternative, the tangent of the half-angle, $\tan(\theta/2)$, may be stored for each end of the arc. This corresponds to the uneven but computationally efficient circle parameterisation mentioned in Section 2.1. To maintain the parameter $\tan(\theta/2)$ in the range 0 to 1, only one quadrant of the circle can be represented. The other quadrants must be indicated by some convention. A compact scheme is to increment the stored value of $\tan(\theta/2)$ with integer values to indicate which quadrant is meant.



The x and y values are now rather more cheaply calculated by the following code, where TK is the value of tan(θ/2) for the first point, K, added to the integer code for the quadrant:

```
T = AMOD(TK,1.0)                    Remove quadrant information
TSQ = T*T
RFAC = R/(1.0 + TSQ)
DX = (1.0 - TSQ)*RFAC
DY = (T + T)*RFAC
IF(TK.LE.1.0) THEN
        XK = XJ + DX                First quadrant
        YK = YJ + DY
ELSE IF(TK.LE.2.0) THEN
        XK = XJ - DY                Second quadrant
        YK = YJ + DX
ELSE IF(TK.LE.3.0) THEN
        XK = XJ - DX                Third quadrant
        YK = YJ - DY
ELSE
        XK = XJ + DY                Fourth quadrant
        YK = YJ - DX
ENDIF
```



This problem is approached in the same way as the problem of finding the distance to a line segment; the parameter value corresponding to the perpendicular to the whole circle is found and this is compared to the start and finish values of the arc. If the parameterisation is based on tangents of half-angles then trigonometric functions can be avoided by finding the tangent of the angle made by KJ with the x axis and converting this to the half-angle parameterisation using the formula in Section 2.1. Changes in quadrant must also be marked by adding the appropriate integer to the value of t:

```
XKJ = XK - XJ
YKJ = YK - YJ

IF (XKJ.GE.0.0.AND.YKJ.GE.0.0) THEN
        IF (XKJ.GT.ACCY) THEN
                TANT = YKJ/XKJ                          First quadrant
                T = (SQRT(1.0 + TANT*TANT) - 1.0)/TANT
        ELSE
                T = 0.0
        ENDIF

ELSE IF (XKJ.LE.0.0.AND.YKJ.GE.0.0) THEN
        IF (YKJ.GT.ACCY) THEN
                TANT = -XKJ/YKJ                         Second quadrant
                T = 1.0 + (SQRT(1.0 + TANT*TANT) - 1.0)/TANT
        ELSE
                T = 1.0
        ENDIF

ELSE IF (XKJ.LE.0.0.AND.YKJ.LE.0.0) THEN
        IF (XKJ.LT.-ACCY) THEN
                TANT = YKJ/XKJ                          Third quadrant
                T = 2.0 + (SQRT(1.0 + TANT*TANT) - 1.0)/TANT
        ELSE
```

```
                T = 2.0
        ENDIF

ELSE
        IF (YKJ.LT.-ACCY) THEN
                TANT = -XKJ/YKJ                          Fourth quadrant
                T = 3.0 + (SQRT(1.0 + TANT*TANT) - 1.0)/TANT
        ELSE
                T = 3.0
        ENDIF
ENDIF

IF (T2.LT.T1) THEN
        IF (T.GT.T1.OR.T.LT.T2) THEN

                ..... Nearest point is on the arc

        ELSE

                ..... Nearest point is an endpoint

        ENDIF
ELSE
        IF (T.GT.T1.AND.T.LT.T2) THEN

                ..... Nearest point is on the arc

        ELSE

                ..... Nearest point is an endpoint

        ENDIF
ENDIF
```

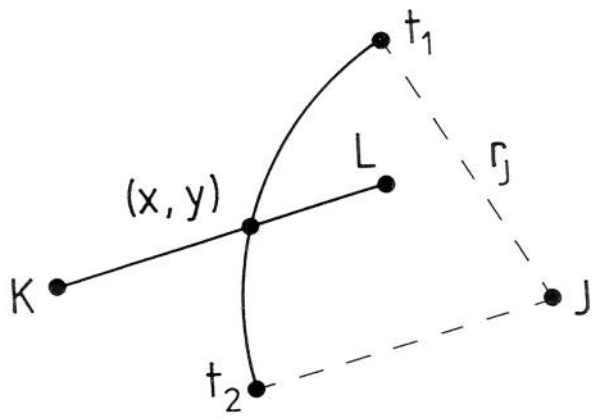
If the nearest point is on the arc then the distance from the point to the arc is simply:

$$r = \sqrt{[(x_K - x_J)^2 + (y_K - y_J)^2]} - r_J$$

A negative distance indicates that J is within the circle.

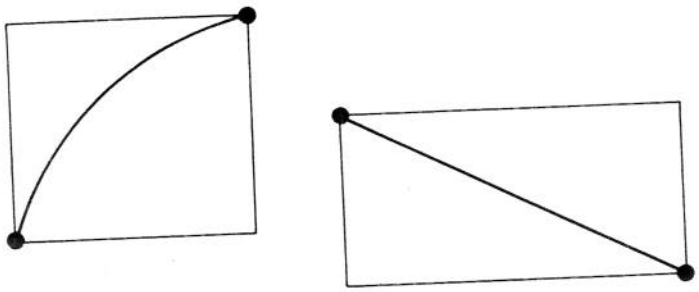
Conversely, if the nearest point to K is an endpoint, then the distance to both endpoints must be calculated using the code in Section 3.4 to find the x and y coordinates corresponding to T1 and T2. The minimum of these two distances is taken as the answer.

3.6 Intersections of a Line Segment and an Arc

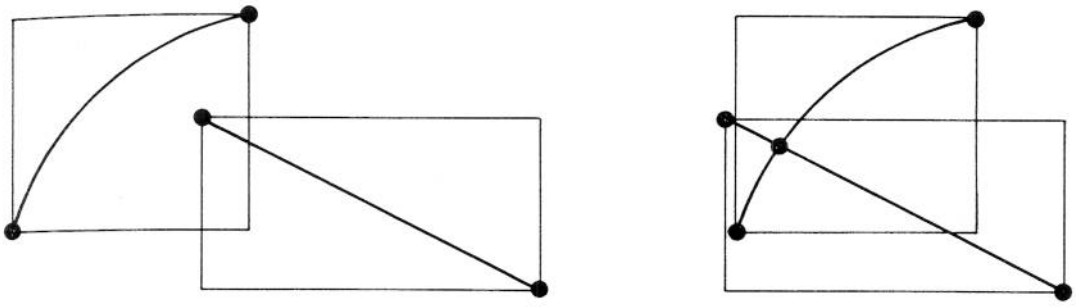


The direct solution of this problem starts with the determination of the intersections of the infinite line of which the segment is a part with the whole circle of which the arc is a part. This operation is covered in Section 2.2. If the infinite line and the whole circle do not intersect, rejection of the arc/segment pair is quick. If, however, the line segment and arc are both short compared to the circle radius, many cases where the segment and the arc are quite distant from each other will remain unrejected until quite late in the computation.

We therefore use a pre-test as we did for segment-segment intersections in Section 3.3; this time an orthodox *boxing test*. Notional rectangles with sides parallel to the axes are constructed and these are compared before comparing the segment and arc that they contain. If they do not intersect the segments may be rejected immediately.

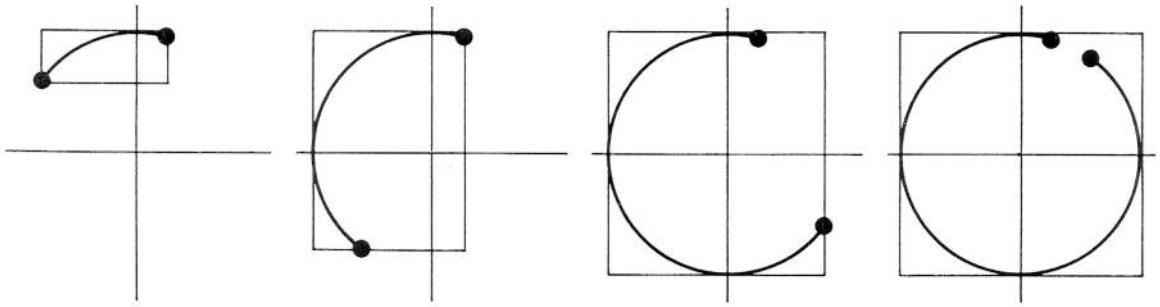


If the boxes do intersect, a more detailed test is necessary to determine whether or not the segment and the arc themselves intersect.

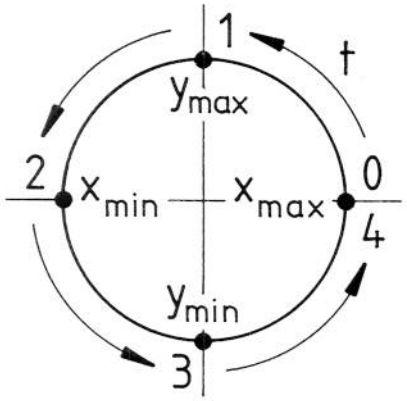


The idea is that most rejections may be made early, especially when segments are small and well separated, which is, of course, the worst case for direct comparison.

For line segments, the corners of the box corresponded to the segment ends. The same is true for arcs if the arc is in a single quadrant. If the arc occupies several quadrants the coordinates of the extreme x and y values of the circle become the x and y coordinates of one or more edges of the box. For example:



The following table shows which values must be used to extend the box for given start and finish quadrants of the arc.



Arc begins

Arc ends	t = 0–1	t = 1–2	t = 2–3	t = 3–4
t = 0–1	XMIN YMIN XMAX YMAX	XMIN YMIN XMAX	YMIN XMAX	XMAX
t = 1–2	YMAX	XMIN YMIN XMAX YMAX	YMIN XMAX YMAX	XMAX YMAX
t = 2–3	XMIN YMAX	XMIN	XMIN YMIN XMAX YMAX	XMIN XMAX YMAX
t = 3–4	XMIN YMIN YMAX	XMIN YMIN	YMIN	XMIN YMIN XMAX YMAX

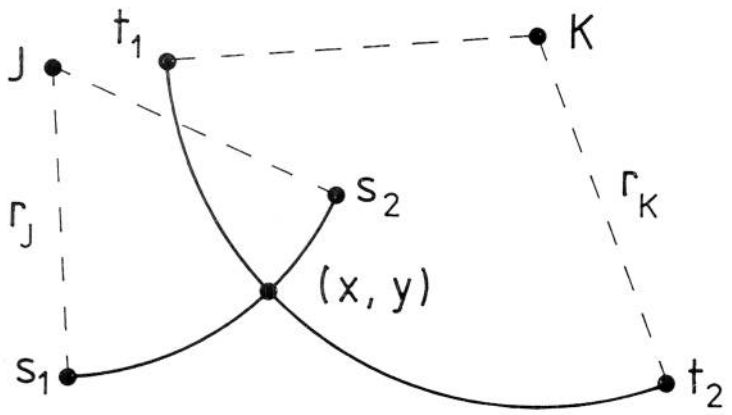
If arcs are to be subjected to repeated boxing tests, then storing their endpoints will save computing time.

If a boxing test fails to eliminate a comparison the next step is to calculate the intersections of the line and the whole circle. Even after a boxing test no intersections may exist. Then it is necessary to parameterise the line segment as described in Section 1.6, and then to work out the intersections (if any) of the resulting parametric line with the *whole* circle, as shown in Section 2.2. Line parameters in the range 0 to 1 at the intersections correspond to intersections inside the segment. The corresponding x and y values are used to calculate the tangent of the angle made by the candidate intersection at the circle centre. This tangent can be converted to a half-angle tangent using the formula:

$$\tan\frac{\theta}{2} = \frac{\sqrt{(1 + \tan^2\theta)} - 1}{\tan\theta}$$

This value can be compared with the values of the arc parameters at the arc ends to see if the intersection lies within the arc or not. The coding of this is identical to the classification of the perpendicular intersection in section 3.5.

3.7 Intersections of Two Arcs



The various operations required for the solution of this problem have already been covered in earlier sections of this and other chapters. If efficiency is at a premium boxing tests may be performed on the two arcs in the same way as for the arc in Section 3.6.

Following any boxing tests, the coordinates of the intersections between the whole circles are found, as described in Section 2.3. If no intersections exist between the whole circles then intersection between the arcs is ruled out. The two sets of intersection coordinates (or only one, if the circles are found to be tangential to each other) are referred to each arc centre in turn to convert them to the half-angle arc parameterisation, following the procedure used in Section 3.5. This section also shows how the values of the parameter thus obtained may be used to determine whether or not the intersections lie on the arcs.

Intersections lying on both arcs are valid: there may be one or two true intersections, or one tangent point.