CHAPTER 7
Intersections

A Sample of Applications

1 The hidden-line and hidden-surface problems

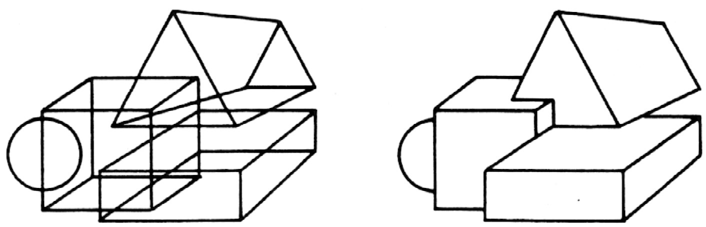


Figure 7.1 Elimination of hidden lines.

• If the projections of two objects are the polygons P_1 and P_2 and P_1 lies nearer to the viewer than P_2, what must be displayed is P_1 and $P_2 \cap \bar{P}_1$ (obviously $P_2 \cap \bar{P}_1$ is the intersection of P_2 and the complement of P_1).

• PROBLEM TYPE I.1 (CONSTRUCT INTERSECTION). Given two geometric objects, form their intersection.

2 Pattern recognition

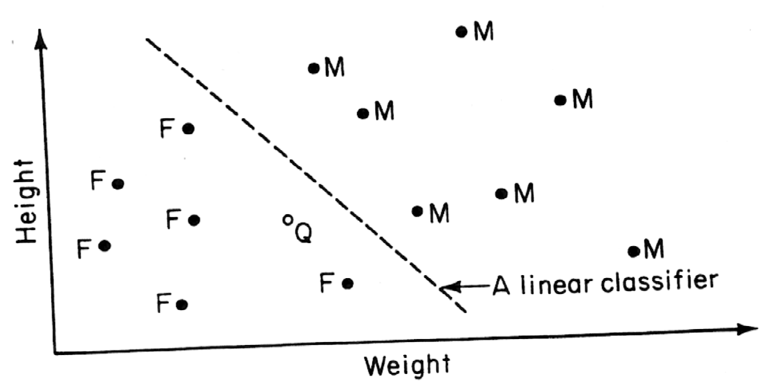


Figure 7.2 A two-variable classification problem.

if $f(x_Q, y_Q) > T$ **then** $Q \in M$ **else** $Q \in F$.

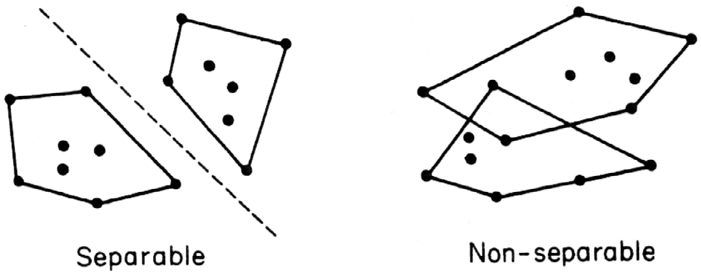


Figure 7.3 Two sets are separable if and only if their convex hulls are disjoint.

- PROBLEM TYPE I.2 (INTERSECTION TEST). Given two geometric objects, do they intersect?

.3 Wire and component layout

- PROBLEM TYPE I.3 (PAIRWISE INTERSECTION). Given N geometric ob jects, determine whether any two intersect.

Intersection of convex polygons

PROBLEM I.1.1 (INTERSECTION OF CONVEX POLYGONS). Given two convex polygons, P with L vertices and Q with M vertices, form their intersection.

Theorem 7.2. *The intersection of a convex L-gon and a convex M-gon is a convex polygon having at most $L + M$ vertices.*

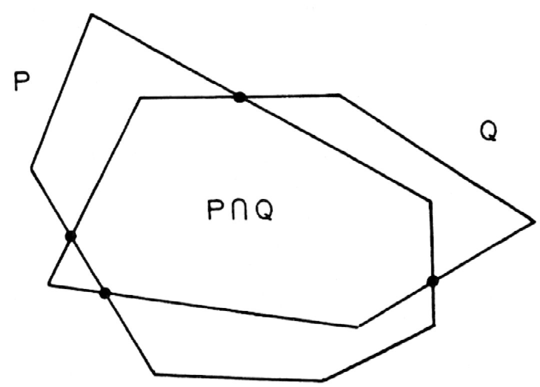


Figure 7.4 Intersection of convex polygons.

Theorem 7.3. *The intersection of a convex L-gon and a convex M-gon can be found in $\theta(L + M)$ time.*

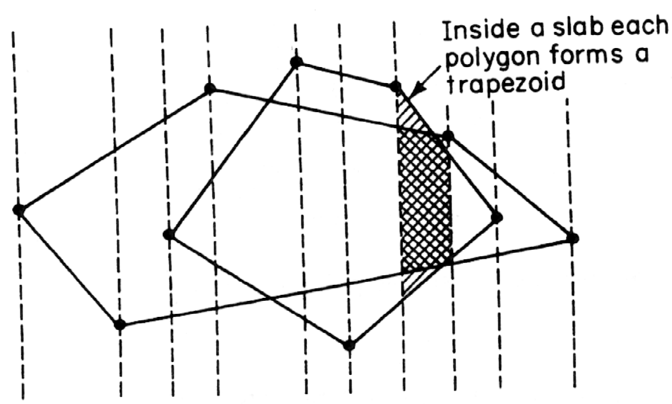


Figure 7.5 Slabs defined by the vertices of two convex polygons.

!!! • Show that the intersection detection can be done in $O(\log(L+M))$ time.

PROBLEM I.2.1 (LINE-SEGMENT INTERSECTION TEST). Given N line segments in the plane, determine whether any two intersect.

Applications

PROBLEM I.2.2 (POLYGON INTERSECTION TEST). Given two simple polygons, P and Q, with M and N vertices, respectively, do they intersect?

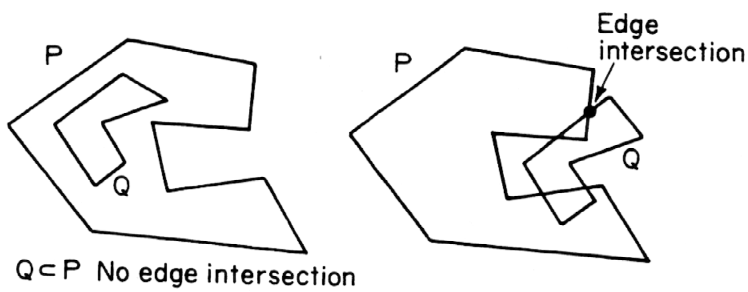


Figure 7.12 Either $P \subset Q$, $Q \subset P$, or there is an edge intersection.

POLYGON INTERSECTION TEST $\propto_N$ LINE-SEGMENT INTERSECTION TEST

PROBLEM I.2.3 (SIMPLICITY TEST). Given a polygon, is it simple?

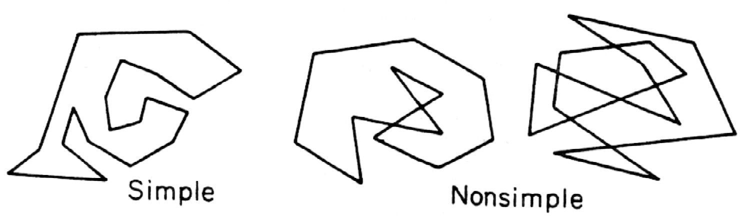


Figure 7.13 Simple and nonsimple polygons.

SIMPLICITY TEST $\propto_N$ LINE-SEGMENT INTERSECTION TEST.

Intersection of star-shaped polygons

Theorem 7.4. *Finding the intersection of two star-shaped polygons requires* $\Omega(N^2)$ *time in the worst case.*

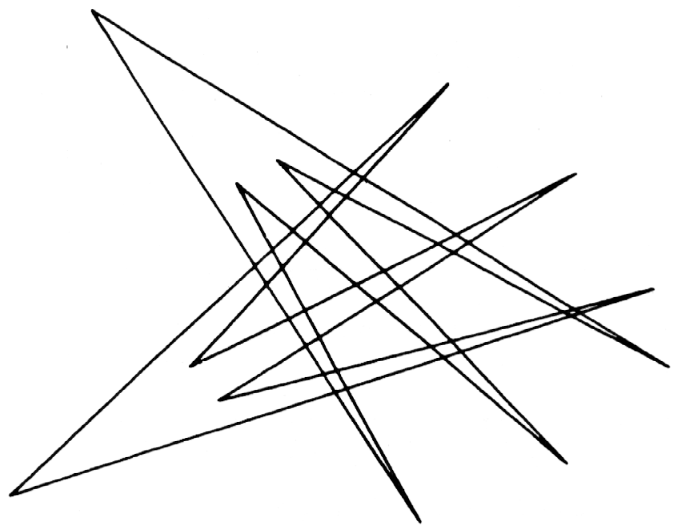
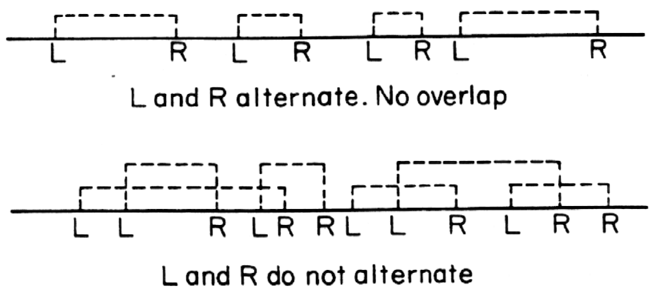


Figure 7.11 The intersection of two star-shaped polygons.

ELEMENT UNIQUENESS $\propto_N$ INTERVAL OVERLAP.

Given a collection of N real numbers x_i, these can be converted in linear time to N intervals $[x_i, x_i]$. These intervals overlap if and only if the original points were not distinct and this proves

Theorem 7.6. $\theta(N \log N)$ *comparisons are necessary and sufficient to determine whether N intervals are disjoint, if only algebraic functions of the input can be computed.*



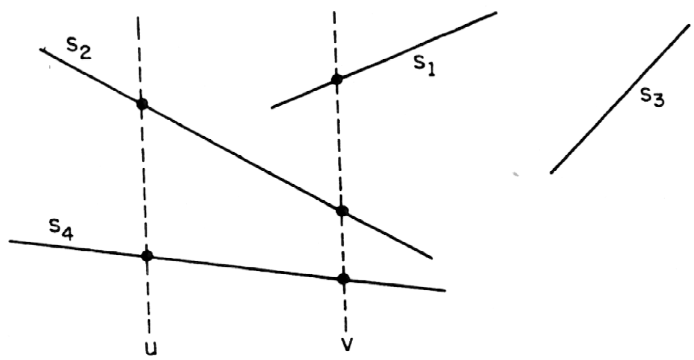


Figure 7.15 An order relation between line segments.

$s_2 >_u s_4$, $s_1 >_v s_2$, $s_2 >_v s_4$, and $s_1 >_v s_4$.

The ordering can change in only three ways:

1. *The left endpoint of segment s is encountered.* In this case s must be added to the ordering.
2. *The right endpoint of s is encountered.* In this case s must be removed from the ordering because it is no longer comparable with any others.
3. *An intersection point of two segments s_1 and s_2 is reached.* Here s_1 and s_2 exchange places in the ordering.

- Sweep-line status: $\mathscr{L}$

a. INSERT($s, \mathscr{L}$). Insert segment s into the total order maintained by $\mathscr{L}$.
b. DELETE($s, \mathscr{L}$). Delete segment s from $\mathscr{L}$.
c. ABOVE($s, \mathscr{L}$). Return the name of the segment immediately above s in the ordering.
d. BELOW($s, \mathscr{L}$). Return the name of the segment immediately below s in the ordering.

⟹ dictionary (use Balanced trees)

- Event-point schedule: $\mathscr{E}$

a. MIN($\mathscr{E}$). Determine the smallest element in $\mathscr{E}$ and delete it.
b. INSERT($x, \mathscr{E}$). Insert abscissa x into the total order maintained by $\mathscr{E}$.

In addition to this essential operation, we also require that $\mathscr{E}$ supports the operation

c. MEMBER($x, \mathscr{E}$). Determine if abscissa x is a member of $\mathscr{E}$.

⟹ priority Queue

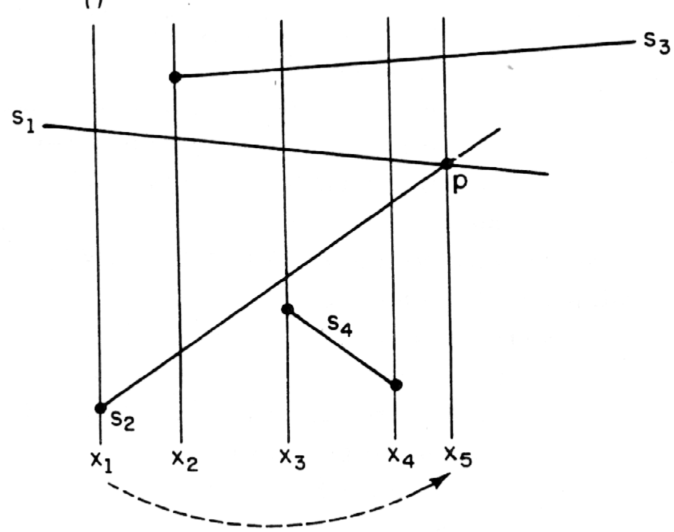


Figure 7.16 Intersection point p is detected for $x = x_1$ when segments s_1 and s_2 are found adjacent for the first time. However event abscissae x_2, x_3, and x_4 must be processed before point p at x_5 is handled.

```
procedure LINE SEGMENT INTERSECTION
begin sort the 2N endpoints lexicographically by x and y and place them
      into priority queue ℰ;
      𝒜 := ∅;
      while (ℰ ≠ ∅) do
         begin p := MIN(ℰ);
               if (p is left endpoint) then
                  begin s := segment of which p is endpoint;
                        INSERT(s, ℒ);
                        s₁ := ABOVE(s, ℒ);
                        s₂ := BELOW(s, ℒ);
                        if (s₁ intersects s) then 𝒜 ⇐ (s₁, s);
                        if (s₂ intersects s) then 𝒜 ⇐ (s, s₂)
                  end
               else if (p is a right endpoint) then
                  begin s := segment of which p is endpoint;
                        s₁ := ABOVE(s, ℒ);
                        s₂ := BELOW(s, ℒ);
                        if (s₁ intersects s₂ to the right of p)
                           then 𝒜 ⇐ (s₁, s₂);
                        DELETE(s, ℒ)
                        end
                  else (*p is an intersection*)
                        begin (s₁, s₂) := segments of which p is intersection
                        (*with s₁ = ABOVE(s₂) to the left of p*)
                              s₃ := ABOVE(s₁, ℒ);
                              s₄ := BELOW(s₂, ℒ);
                              if (s₃ intersects s₂) then 𝒜 ⇐ (s₃, s₂);
                              if (s₁ intersects s₄) then 𝒜 ⇐ (s₁, s₄);
                              interchange s₁ and s₂ in ℒ
                        end;
                  (*the detected intersections must now be processed*)
               while (𝒜 ≠ ∅) do
                  begin (s, s') ⇐ 𝒜;
                        x := common abscissa of s and s';
                        if (MEMBER(x, ℰ) = FALSE) then
                           begin output (s, s');
                                 INSERT(x, ℰ)
                           end
                  end
         end
   end.
```

Theorem 7.7 [Bentley–Ottmann (1979)]. *The K intersections of a set of N line segments can be reported in time $O((N + K)\log N)$.*

• Is not optimal; $\Omega(n \log n + k)$ is a lower bound.

PROBLEM I.1.2 (LINE SEGMENT INTERSECTION). Given N line segments, determine all their intersections.

Theorem 7.9. *Whether any two of N line segments in the plane intersect can be determined in $\theta(N\log N)$ time, and this is optimal.*

An immediate consequence of this result is the following.

Corollary 7.1. *The following problems can be solved in time $O(N\log N)$, in the worst case.*

PROBLEM I.2.2 (POLYGON INTERSECTION TEST). Do two given polygons intersect?

PROBLEM I.2.3 (SIMPLICITY TEST). Is a given polygon simple?

PROBLEM I.2.4 (EMBEDDING TEST). Does a straight-line embedding of a planar graph contain any crossing edges?

Corollary 7.2. *Whether any two of N circles intersect can be determined in $O(N\log N)$ time.*

!!! • Can you show this?

PROBLEM P.13 (MAXIMUM GAP). Given a set S of N real numbers x_1, $x_2, \ldots, x_N$, find the maximum difference between two consecutive members of S. (Two numbers x_i and x_j of S are said to be consecutive if they are such in any permutation of $(x_1, \ldots, x_N)$ that achieves natural ordering.)

Corollary 6.2. *In the algebraic computation tree model, any algorithm for the* MAXIMUM GAP *problem on a set of N real numbers requires $\Omega(N \log N)$ time.*

In a modified computation model, however, Gonzalez (1975) has obtained the most surprising result that the problem can be actually solved in linear time. The modification consists of adding the (nonanalytic) *floor function* "⌊ ⌋" to the usual repertoire. Here is Gonzalez's remarkable algorithm:

```
    procedure MAX GAP
      Input:   N real numbers X[1:N] (unsorted)
      Output: MAXGAP, the length of the largest gap between consecutive
               numbers in sorted order.
begin MIN := min X[i];
      MAX := max X[i];
      (*create N − 1 buckets by dividing the interval from MIN to MAX
      with N − 2 equally-spaced points. In each bucket we will retain
      HIGH[i] and LOW[i], the largest and smallest values in bucket i*)
      for i := 1 until N − 1 do
          begin COUNT[i] := 0;
                LOW[i] := HIGH[i] := Λ
          end; (*the buckets are set up*)
      (*hash into buckets*)
      for i := 1 until N − 1 do
          begin BUCKET := 1 + ⌊(N − 1) × (X[i] − MIN)/
                (MAX − MIN)⌋;
                COUNT[BUCKET] := COUNT[BUCKET] + 1;
                LOW[BUCKET] := min (X[i], LOW[BUCKET]);11
                HIGH[BUCKET] := max (X[i], HIGH[BUCKET])11
          end;
      (*Note that N − 2 points have been placed in N − 1 buckets, so by
      the pigeonhole principle some bucket must be empty. This means that
      the largest gap cannot occur between two points in the same bucket.
      Now we make a single pass through the buckets*)
      MAXGAP := 0;
      LEFT := HIGH[1];
      for i := 2 until N − 1 do
          if (COUNT[i] ≠ 0) then
            begin THISGAP := LOW[i]-LEFT;
                  MAXGAP := max(THISGAP, MAXGAP);
                  LEFT := HIGH[i]
            end
end.
```

This algorithm sheds some light on the computational power of the "floor"