

Thematic Map Overlay

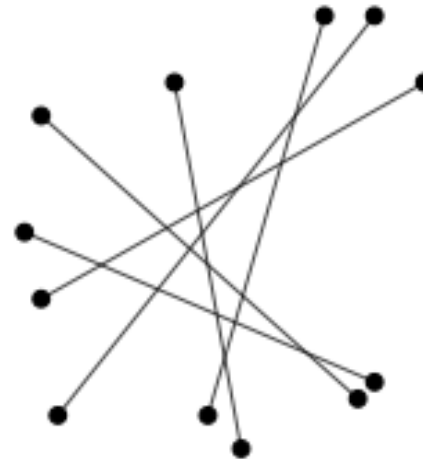


**Computational Geometry
2 : Line Segment Intersection
pages 19-41**

Compute all intersections among the segments !

Brute force algorithm :

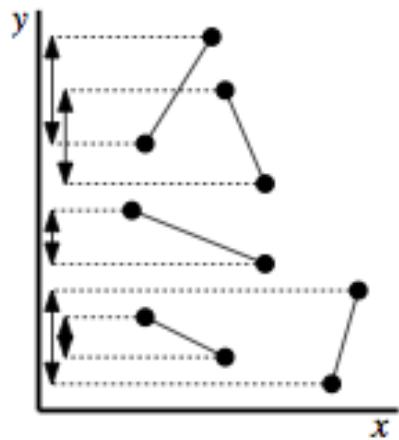
*Simply take each pair of segments,
compute whether they intersect,
and, if so, report their intersection point.*



The brute-force algorithm clearly requires $O(n^2)$ time.

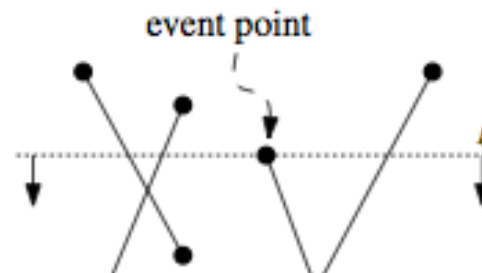
In a sense this is optimal here !
When each pair of segments intersects,
any algorithm must take $\Omega(n^2)$ time,
because it has to report all intersections.c

An output sensitive algorithm ?



How can we avoid testing all pairs of segments for intersection?

Plane sweep algorithm

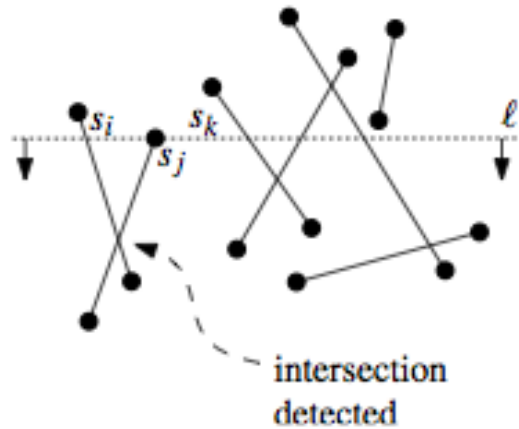


The moments at which the sweep line reaches an event point are the only moments when we do something:

- Update the status of the sweep line.
- Performs some intersection tests.

The status corresponds to the **ordered** sequence of segments intersecting the sweep line.

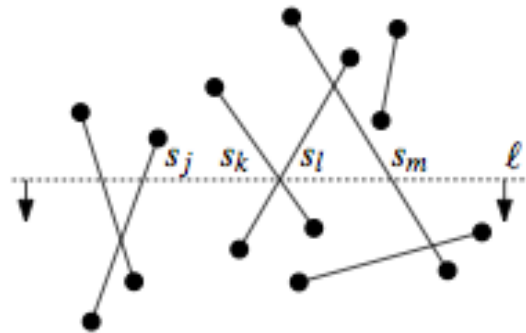
Event point = upper endpoint



This segment must be tested for intersection against its two neighbors along the sweep line.

Only intersection points below the sweep line are important

Event point = intersection

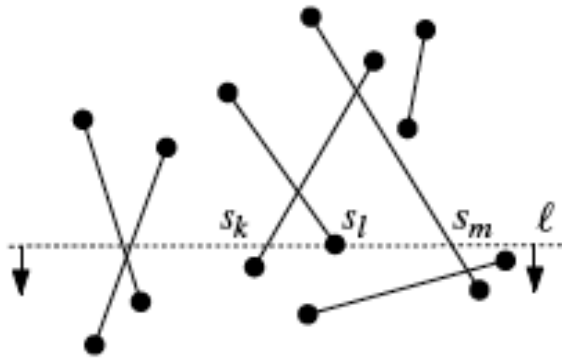


The two segments that intersect change their order.

Each of them gets (at most) one new neighbor against which it is tested for intersection.

Again, only intersections below the sweep line are still interesting.

Event point = lower point



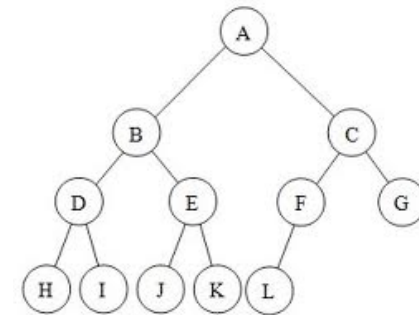
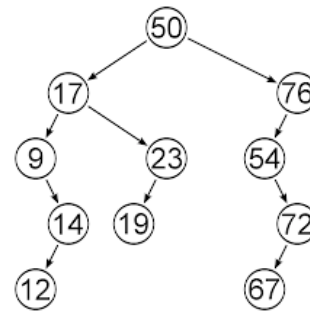
Assume three segments s_k , s_l and s_m appear in this order on the sweep line when the lower endpoint of s_l is encountered.

Then s_k and s_m will become adjacent and we test them for intersection.

Its two neighbors now become adjacent and must be tested for intersection.

If they intersect below the sweep line, then their intersection point is an event point. (Again, this event could have been detected already.)

Event queue



We store the events in a **balanced binary search tree**, ordered according to the altitude.

Fetching the next event and inserting an event take $O(\log m)$ time, where m is the number of events in Q .

We do not use a heap to implement the event queue, because we have to be able to test whether a given event is already present in Q .



The **event queue** stores the events.
We denote the event queue by Q .

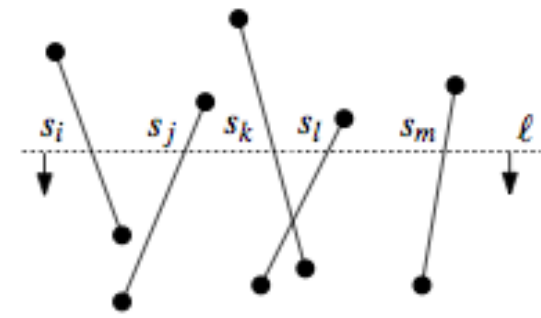
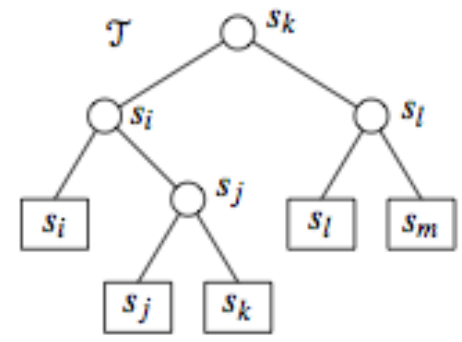
We need an operation that removes the next event that will occur from Q , and returns it so that it can be treated.

Status tree



We also use a **balanced binary search tree** !

Each update and neighbor search operation takes $O(\log n)$ time.



The **status tree** stores the ordered sequence of segments intersecting the sweep line.
We denote the status tree by T .

We need an operation that removes the next event that will occur from Q , and returns it so that it can be treated.

The algorithm

Algorithm FINDINTERSECTIONS(S)

Input. A set S of line segments in the plane.

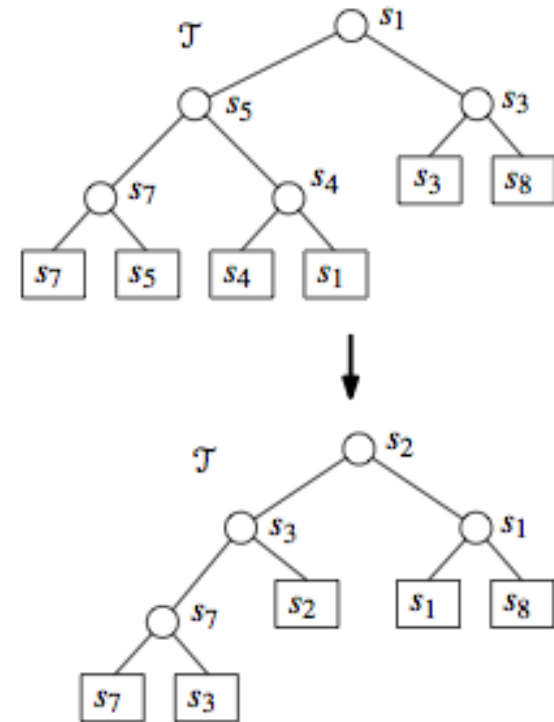
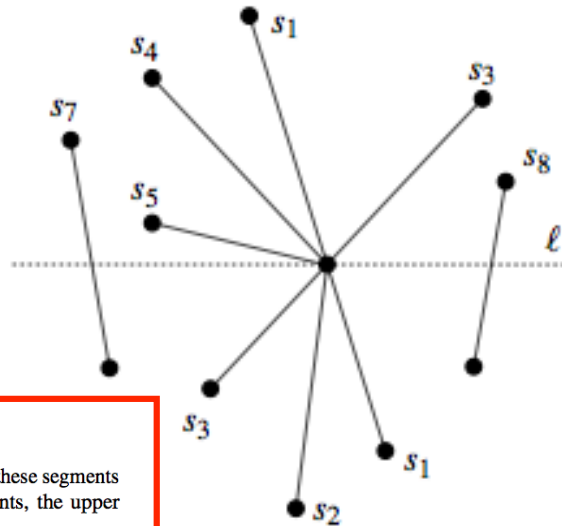
Output. The set of intersection points among the segments in S , with for each intersection point the segments that contain it.

1. Initialize an empty event queue \mathcal{Q} . Next, insert the segment endpoints into \mathcal{Q} ; when an upper endpoint is inserted, the corresponding segment should be stored with it.
2. Initialize an empty status structure \mathcal{T} .
3. **while** \mathcal{Q} is not empty
4. **do** Determine the next event point p in \mathcal{Q} and delete it.
5. HANDLEEVENTPOINT(p)

HANDLEEVENTPOINT(p)

1. Let $U(p)$ be the set of segments whose upper endpoint is p ; these segments are stored with the event point p . (For horizontal segments, the upper endpoint is by definition the left endpoint.)
2. Find all segments stored in \mathcal{T} that contain p ; they are adjacent in \mathcal{T} . Let $L(p)$ denote the subset of segments found whose lower endpoint is p , and let $C(p)$ denote the subset of segments found that contain p in their interior.
3. **if** $L(p) \cup U(p) \cup C(p)$ contains more than one segment
4. **then** Report p as an intersection, together with $L(p)$, $U(p)$, and $C(p)$.
5. Delete the segments in $L(p) \cup C(p)$ from \mathcal{T} .
6. Insert the segments in $U(p) \cup C(p)$ into \mathcal{T} . The order of the segments in \mathcal{T} should correspond to the order in which they are intersected by a sweep line just below p . If there is a horizontal segment, it comes last among all segments containing p .
7. (* Deleting and re-inserting the segments of $C(p)$ reverses their order. *)
8. **if** $U(p) \cup C(p) = \emptyset$
9. **then** Let s_l and s_r be the left and right neighbors of p in \mathcal{T} .
10. FINDNEWEVENT(s_l, s_r, p)
11. **else** Let s' be the leftmost segment of $U(p) \cup C(p)$ in \mathcal{T} .
12. Let s_l be the left neighbor of s' in \mathcal{T} .
13. FINDNEWEVENT(s_l, s', p)
14. Let s'' be the rightmost segment of $U(p) \cup C(p)$ in \mathcal{T} .
15. Let s_r be the right neighbor of s'' in \mathcal{T} .
16. FINDNEWEVENT(s'', s_r, p)

A a little bit more tricky !



HANDLEEVENTPOINT(p)

1. Let $U(p)$ be the set of segments whose upper endpoint is p ; these segments are stored with the event point p . (For horizontal segments, the upper endpoint is by definition the left endpoint.)
2. Find all segments stored in \mathcal{T} that contain p ; they are adjacent in \mathcal{T} . Let $L(p)$ denote the subset of segments found whose lower endpoint is p , and let $C(p)$ denote the subset of segments found that contain p in their interior.
3. **if** $L(p) \cup U(p) \cup C(p)$ contains more than one segment
4. **then** Report p as an intersection, together with $L(p)$, $U(p)$, and $C(p)$.
5. Delete the segments in $L(p) \cup C(p)$ from \mathcal{T} .
6. Insert the segments in $U(p) \cup C(p)$ into \mathcal{T} . The order of the segments in \mathcal{T} should correspond to the order in which they are intersected by a sweep line just below p . If there is a horizontal segment, it comes last among all segments containing p .
7. (* Deleting and re-inserting the segments of $C(p)$ reverses their order. *)
8. **if** $U(p) \cup C(p) = \emptyset$
9. **then** Let s_l and s_r be the left and right neighbors of p in \mathcal{T} .
10. FINDNEWEVENT(s_l, s_r, p)
11. **else** Let s' be the leftmost segment of $U(p) \cup C(p)$ in \mathcal{T} .
12. Let s_l be the left neighbor of s' in \mathcal{T} .
13. FINDNEWEVENT(s_l, s', p)
14. Let s'' be the rightmost segment of $U(p) \cup C(p)$ in \mathcal{T} .
15. Let s_r be the right neighbor of s'' in \mathcal{T} .
16. FINDNEWEVENT(s'', s_r, p)

FINDNEWEVENT(s_l, s_r, p)

1. **if** s_l and s_r intersect below the sweep line, or on it and to the right of the current event point p , and the intersection is not yet present as an event in \mathcal{Q}
2. **then** Insert the intersection point as an event into \mathcal{Q} .

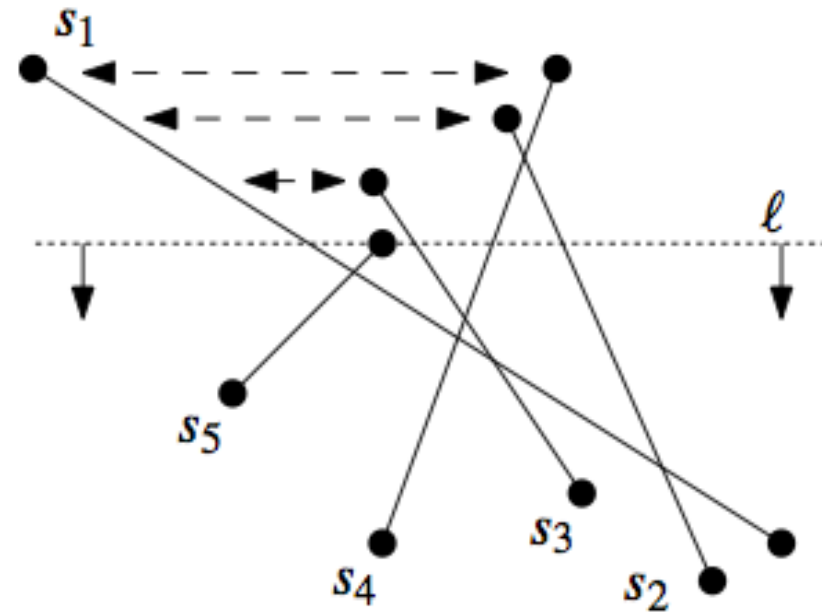
HANDLEEVENTPOINT(p)

1. Let $U(p)$ be the set of segments whose upper endpoint is p ; these segments are stored with the event point p . (For horizontal segments, the upper endpoint is by definition the left endpoint.)
2. Find all segments stored in \mathcal{T} that contain p ; they are adjacent in \mathcal{T} . Let $L(p)$ denote the subset of segments found whose lower endpoint is p , and let $C(p)$ denote the subset of segments found that contain p in their interior.
3. **if** $L(p) \cup U(p) \cup C(p)$ contains more than one segment
4. **then** Report p as an intersection, together with $L(p)$, $U(p)$, and $C(p)$.
5. Delete the segments in $L(p) \cup C(p)$ from \mathcal{T} .
6. Insert the segments in $U(p) \cup C(p)$ into \mathcal{T} . The order of the segments in \mathcal{T} should correspond to the order in which they are intersected by a sweep line just below p . If there is a horizontal segment, it comes last among all segments containing p .
7. (* Deleting and re-inserting the segments of $C(p)$ reverses their order. *)
8. **if** $U(p) \cup C(p) = \emptyset$
9. **then** Let s_l and s_r be the left and right neighbors of p in \mathcal{T} .
10. FINDNEWEVENT(s_l, s_r, p)
11. **else** Let s' be the leftmost segment of $U(p) \cup C(p)$ in \mathcal{T} .
12. Let s_l be the left neighbor of s' in \mathcal{T} .
13. FINDNEWEVENT(s_l, s', p)
14. Let s'' be the rightmost segment of $U(p) \cup C(p)$ in \mathcal{T} .
15. Let s_r be the right neighbor of s'' in \mathcal{T} .
16. FINDNEWEVENT(s'', s_r, p)

FINDNEWEVENT(s_l, s_r, p)

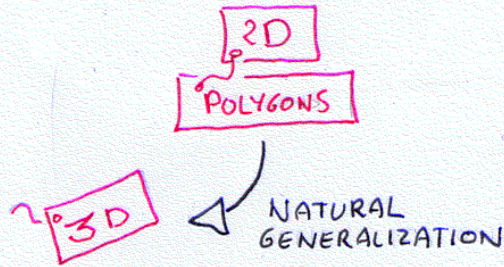
1. **if** s_l and s_r intersect below the sweep line, or on it and to the right of the current event point p , and the intersection is not yet present as an event in \mathcal{Q}
2. **then** Insert the intersection point as an event into \mathcal{Q} .

Complexity of the sweep algorithm

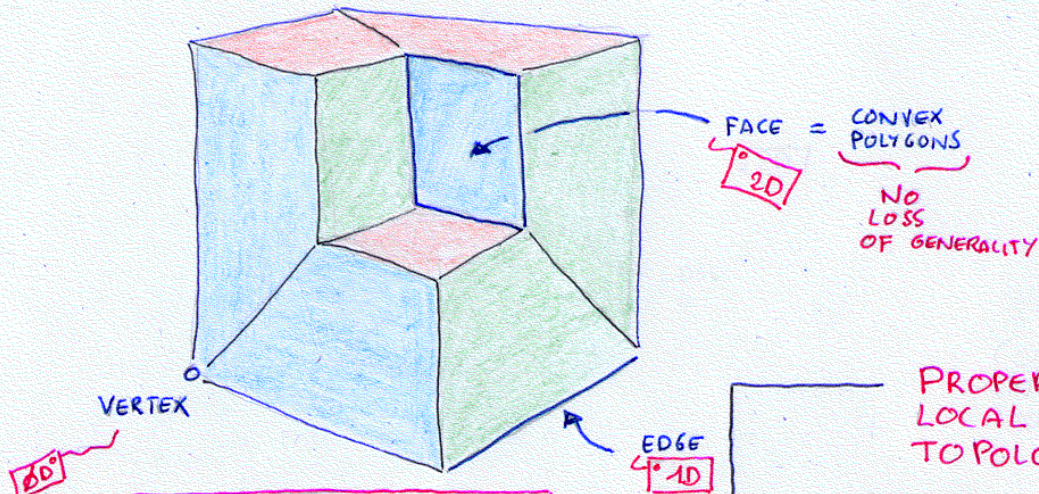


Theorem 2.4 *Let S be a set of n line segments in the plane. All intersection points in S , with for each intersection point the segments involved in it, can be reported in $O(n \log n + I \log n)$ time and $O(n)$ space, where I is the number of intersection points.*

POLYHEDRA

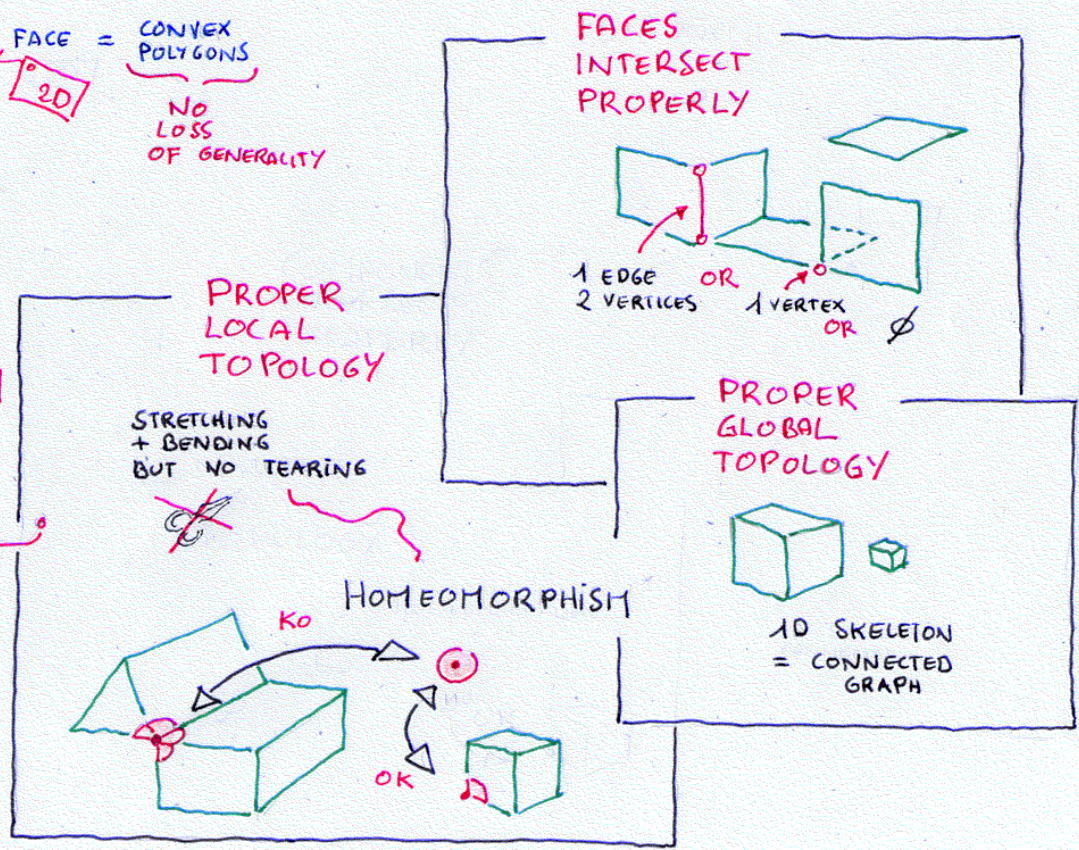


3 VALIDITY CONDITIONS



3 GEOMETRIC COMPONENTS

2-MANIFOLD




POLYTOPES

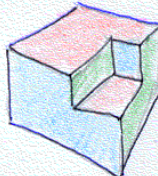
CLOSED CURVE
 POLYGONS
 CONVEX POLYGONS
 REGULAR POLYGONS

2D ANALOGY

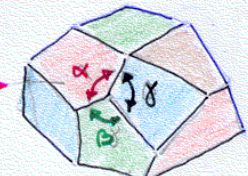
2-MANIFOLDS



POLYHEDRA = SPECIAL CASE OF 2-MANIFOLDS



POLYTOPES ≅ CONVEX POLYHEDRA

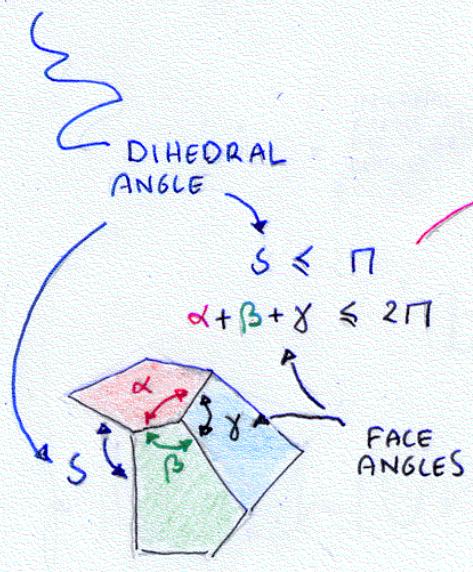


NO STANDARD NOTATIONS
 POLYTOPES
 POLYHEDRA

CONVEX BOUNDED

BE CAREFUL!

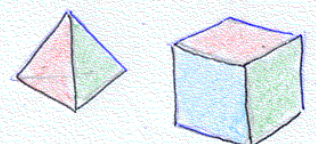
INTERNAL ANGLE IN SPACE AT THE EDGE BETWEEN PLANES CONTAINING ASSOCIATED FACES



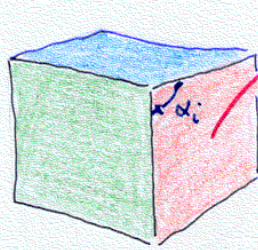
REGULAR POLYTOPES

FACES = REGULAR POLYGONS
 SAME NUMBER OF FACES AT EACH VERTEX

EQUAL ANGLES
 EQUAL SIDES



PLATONIC SOLIDS INEQUALITY



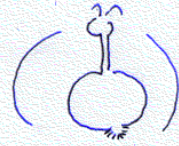
$m =$ NUMBER OF VERTICES PER FACE

$v =$ NUMBER OF FACES PER VERTEX

$m\alpha$

$$\sum_{i=1}^m \alpha_i = (m-2)\pi$$

m VERTICES
 $m-3$ DIAGONALS
 $m-2$ TRIANGLES



CONVEX POLYHEDRA CONDITION

$$\sum_{i=1}^v \alpha_i < 2\pi$$

$v\alpha$



$$\cancel{v \frac{(m-2)\pi}{m}} < \cancel{2\pi}$$

$$\begin{aligned} v m - 2v - 2m &< 0 \\ v m - 2v - 2m + 4 &< 4 \\ \hline (v-2)(m-2) &< 4 \end{aligned}$$

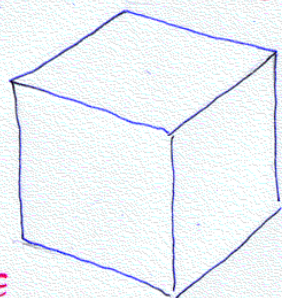
PLATONIC SOLIDS



4
TETRAHEDRON

3 TRIANGLES AT EACH VERTEX
 $(v-2)(n-2) = 1$

4 VERTICES
6 EDGES

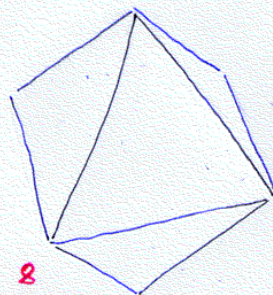


CUBE

8 VERTICES
12 EDGES

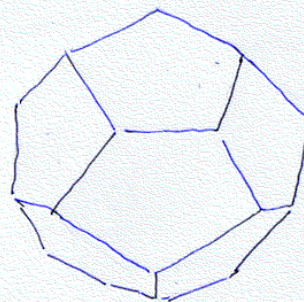
6
HEXAHEDRON

3 SQUARES AT EACH VERTEX
 $(v-2)(n-2) = 2$



8
OCTAHEDRON

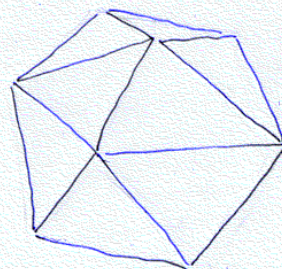
4 TRIANGLES AT EACH VERTEX
6 VERTICES
12 EDGES
 $(v-2)(n-2) = 2$



12
DODECAHEDRON

3 PENTAGONS AT EACH VERTEX
 $(v-2)(n-2) = 3$

20 VERTICES
30 EDGES



20
ICOSAHEDRON

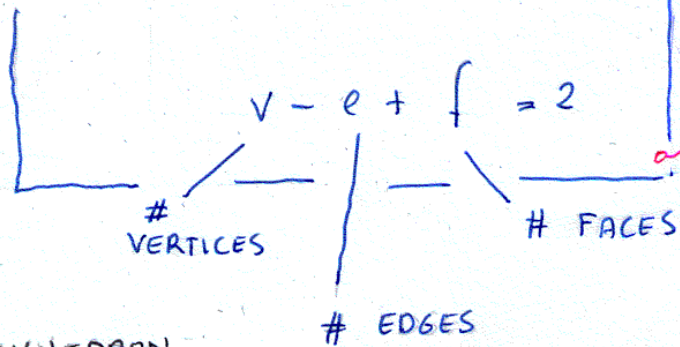
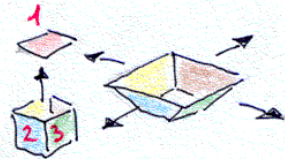
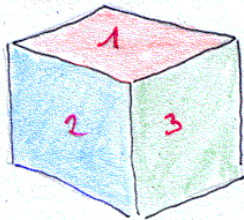
5 TRIANGLES AT EACH VERTEX
 $(v-2)(n-2) = 3$

12 VERTICES
30 EDGES

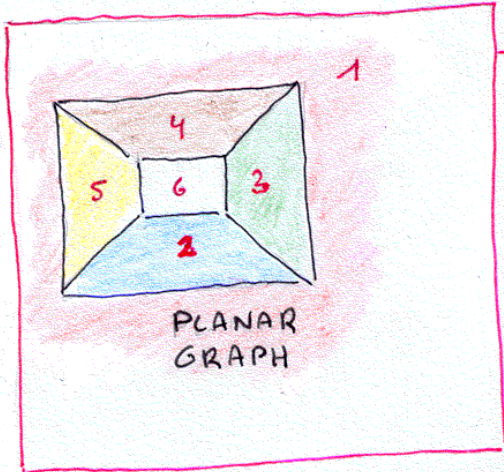
EULER'S FORMULA

PROOF

POLYHEDRON

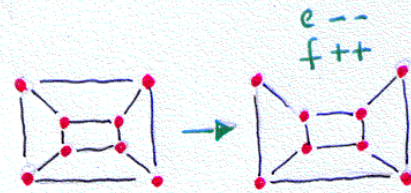


LEONARD EULER 1758

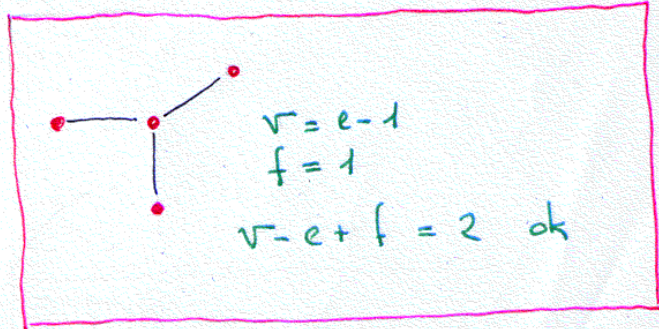


THE GRAPH IS NOT A TREE

TRANSFORMING IT IN A TREE CAN BE PERFORMED PRESERVING $v - e + f$



$v - e + f = 2$
TRUE FOR TREE



Exercise 2

- 2.1 Let S be a set of n disjoint line segments whose upper endpoints lie on the line $y = 1$ and whose lower endpoints lie on the line $y = 0$. These segments partition the horizontal strip $[-\infty : \infty] \times [0 : 1]$ into $n + 1$ regions. Give an $O(n \log n)$ time algorithm to build a binary search tree on the segments in S such that the region containing a query point can be determined in $O(\log n)$ time. Also describe the query algorithm in detail.

