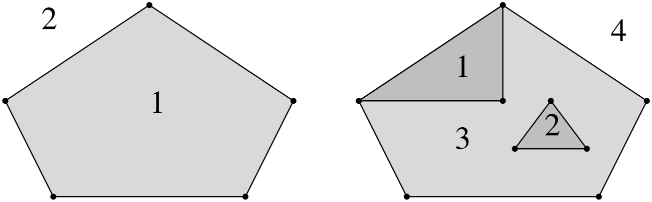
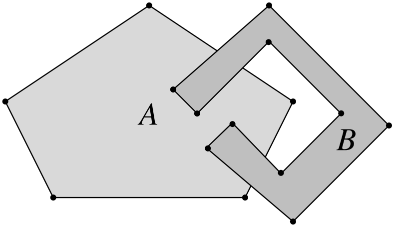
Planar Subdivisions



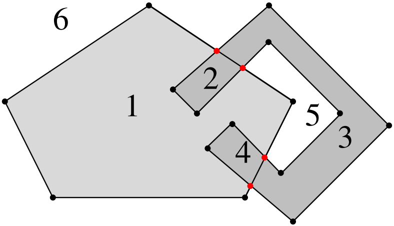
$v = 5, e = 5, f = 2, c = 1$ $\quad$ $v = 9, e = 10, f = 4, c = 2$

- Division of the plane into open regions, called faces.
- Region boundary elements are line segments, called edges.
- Edge endpoints are called vertices.
- Notation: v vertices, e edges, f faces, c components.
- Euler formula: $v - e + f = 1 + c$.

Overlays



two polygons



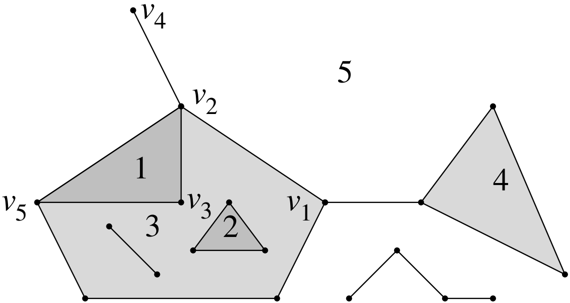
overlay with 6 faces

- Two polygons define a joint subdivision, called an overlay.
- Boolean operations yield sets of faces.
  - $A \cup B = \{1, 2, 3, 4\}$
  - $A \cap B = \{2, 4\}$
  - $A - B = \{1\}$
  - $B - A = \{3\}$

Boundary Representation

- Subdivisions are represented with a boundary representation.
- The elements are vertices, edges, and faces.
- An edge ab has a tail vertex a and a head vertex b.
- The twin of ab is ba.
- Every edge belongs to a single edge loop.
- A vertex stores its coordinates and incident edges.
- An edge stores its tail, twin, and the next edge in its loop.
- A face stores one edge from each of its boundary loops.
- The interior is to the left when a boundary edge is traversed from tail to head.

Example



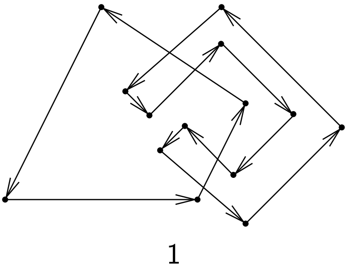
$v = 19, e = 19, f = 5, c = 4$

- The next of edge $v_1 v_2$ is $v_2 v_3$ and both bound face 3.
- The next of edge $v_5 v_2$ is $v_2 v_4$ and both bound face 5.
- The next of edge $v_2 v_4$ is $v_4 v_2$ and both bound face 5.
- This type of edge is called dangling.

Overlay Algorithm

1. Construct the boundary representations of the polygons.
2. Split the edges at their intersection points.
3. Form edge loops.
4. Classify each loop as an outer or an inner boundary.
5. Each outer boundary defines a bounded face.
6. Assign the inner boundaries with point-in-polygon tests.

Overlay Algorithm

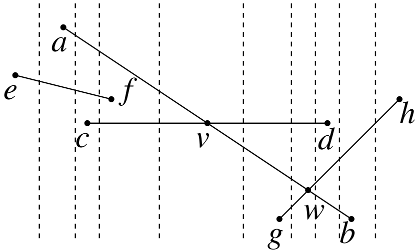


1. Construct the boundary representations of the polygons.
2. Split the edges at their intersection points.

Intersection Point Computation

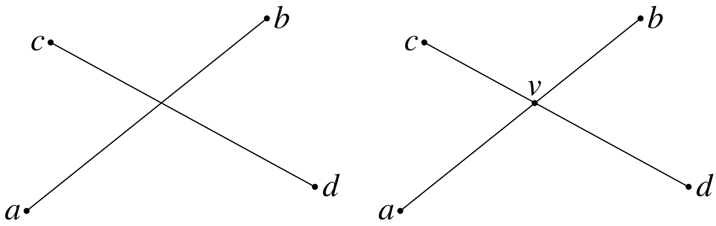
- Input: n edges.
- Output: m intersection points.
- Worst case: $m = O(n^2)$, so running time is $O(n^2)$.
- Brute force algorithm: test every pair of edges.
- Sweep algorithm: test pairs of edges that see each other.
- Output sensitive: $O((n+m)\log n)$.
- Complicated optimal algorithm: $O(n \log n + m)$.

Sweep Algorithm



- Sweep a vertical line through the edges.
- Track the vertical order of the edges that intersect the sweep.
  1. (ef) from e.
  2. (ef, ab) from a.
  3. (cd, ef, ab) from c.
  4. (cd, ab) from f.
  5. (ab, cd) from v.
- Check incident segments for intersection.
  - Check ef and ab at a; no intersection.
  - Check cd and ab at f; compute v.
  - Check gh and ab at v; compute g.

Edge Split

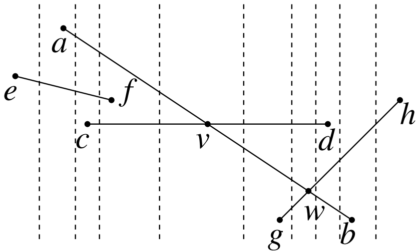


- Add edge *va* with twin *ab* and next *ba.n*.
- Add edge *vb* with twin *ba* and next *ab.n*.
- Add edge *vc* with twin *cd* and next *dc.n*.
- Add edge *vd* with twin *dc* and next *cd.n*.
- Set the twin of *ab* to *va* and the next to *vc*.
- Set the twin of *ba* to *vb* and the next to *vd*.
- Set the twin of *cd* to *vc* and the next to *vb*.
- Set the twin of *dc* to *vd* and the next to *va*.

Implementation

- Sweep list: ordered edges.
- Initialize empty binary tree.
- Events: left endpoint, right endpoint, intersection point.
- Initialize priority queue with endpoint events.
- Queue order is point x coordinate order.
- For the same point, process right endpoint before left.
- Process the next event until the queue is empty.
  - Add or remove edge from sweep list, or swap two edges.
  - Check newly adjacent pairs of edges for intersection.
  - Add intersection points to queue (avoiding duplicates).

Computing the Vertical Order



- The vertical order is computed at the left endpoint a of an edge ab with respect to an edge ef with $e_x < a_x < f_x$.
- a is above ef if aef is a left turn.
- This order is correct until the edges swap.
- The edges are removed from the tree before the swap.
- The edges whose left endpoint is the swap point are inserted.

Complexity

- There are $2n$ endpoint events and m intersection point events.
- Processing an event takes $O(\log n)$ time.
  - $O(\log n)$ to update the queue.
  - $O(\log n)$ to update the sweep.
  - $O(1)$ to check at most two newly adjacent pairs.
- The running time is $O((n + m) \log n)$.

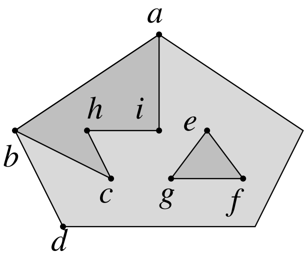
Degenerate Input

- Extra steps are required for degenerate input:
  - three edges that intersect at a point,
  - an intersection point that coincides with an endpoint,
  - a vertical edge,
  - collinear edges.
- The complexity is the same; the proof is a bit harder.
- Getting the program right is tedious.

Overlay Algorithm (reminder)

1. Construct the boundary representations of the polygons.
2. Split the edges at their intersection points.
3. Form edge loops.
4. Classify each loop as an outer or an inner boundary.
5. Each outer boundary defines a bounded face.
6. Assign the inner boundaries with point-in-polygon tests.

Loop Classification

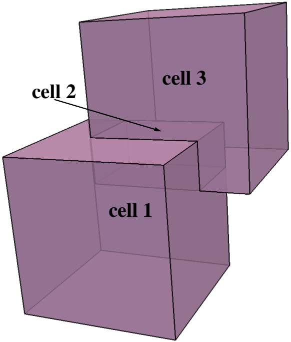


- Classify each loop as an outer or an inner boundary.
- Outer if a left turn occurs at the leftmost vertex.
- Examples: *abc*, *cbd*, and *egf*, but not *fge*.
- Leftmost vertex required, e.g. *chi*.
- Can be extremal in any direction.

Inner Boundary Assignment

- The unbounded face has no outer boundary and one or more inner boundaries.
- The other faces have one outer boundary and zero or more inner boundaries.
- An inner boundary belongs to its closest outer boundary.
  1. Intersect a ray through a vertex of the inner boundary with the outer boundaries.
  2. If there are no intersections, assign the inner boundary to the unbounded face.
  3. Otherwise, assign it to the first intersection.
- Alternately, the closest boundary can be computed by the sweep algorithm in constant time.

Solid Modeling

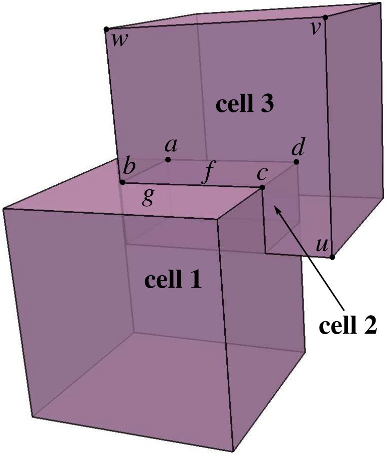


- Construct and manipulate polyhedral models.
- The boundary representation works in 3D.
- The only new ingredient is cells.
- The sweep line algorithm does not generalize well.
- The point-in-polyhedron test is like the point-in-polygon test.

Boundary Representation

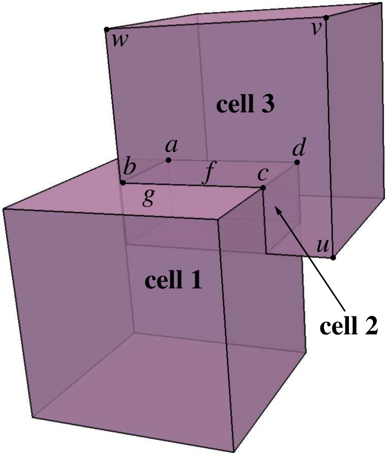
- A vertex v has coordinates (v_x, v_y, v_z) and incident edges vw.
- An edge e has a tail, a twin, a next edge $e.n$, and a facet.
- Edge loops bound facets.
- A facet has one edge per boundary loop, and a shell.
- A triangular facet has one boundary edge e with $e = e.n.n.n$.
- A shell is a closed surface comprised of facets.
- A cell is an open region bounded by shells.
- A bounded cell has an outer shell.
- Every cell has zero or more inner shells.

Example Revisited



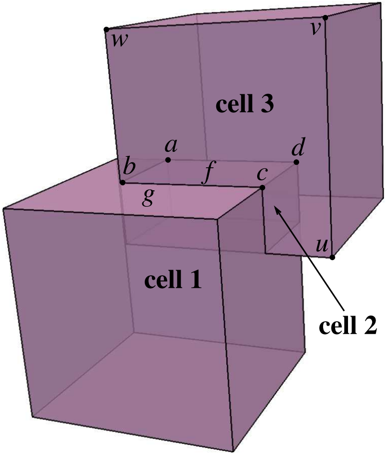
- Edge loop *abcd* bounds facet *f*.
- Edge *cb* bounds facet *g*.
- Facet *f* bounds cells 2 and 3.

Manifold Surfaces



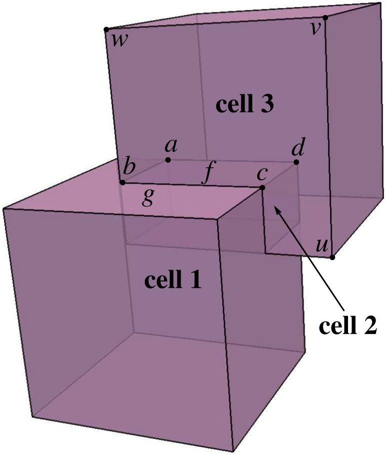
- Manifold surfaces
  - Every edge bounds a single facet.
  - if an edge bounds a facet, so does its twin.
- Are the shells manifold surfaces?

Manifold Surfaces



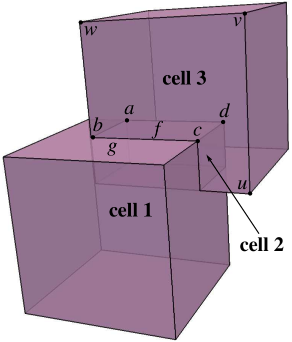
- Manifold surfaces
  - Every edge bounds a single facet.
  - if an edge bounds a facet, so does its twin.
- Are the shells manifold surfaces? Yes.
- Is the entire subdivision a manifold?

Manifold Surfaces



- Manifold surfaces
  - Every edge bounds a single facet.
  - if an edge bounds a facet, so does its twin.
- Are the shells manifold surfaces? Yes.
- Is the entire subdivision a manifold? No.

Shell Orientation



- A shell divides space into bounded and unbounded regions.
- A facet has positive orientation if its normal points into the unbounded region.
- All the facets of a shell have the same orientation.
- Let v be the vertex with the largest z coordinate.
- Let edge vw form the smallest angle with the z axis.
- The shell is positive if vw is convex.

Cell Construction

1. Form the shells by traversing the facets.
2. Compute the orientations of the shells.
3. A positive shell is an outer boundary of the on its interior side and is an inner boundary of the cell on its exterior side.
4. A negative shell is the opposite.
5. Use ray casting to compute the shell nesting.