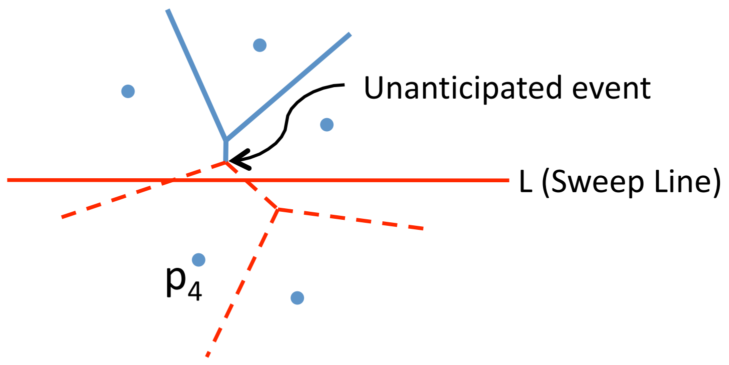
Fortune’s Algorithm

Notes from the book by de Berg, Van Krevald, Overmars, and Schwarzkpf

Fortune's Algorithm

- Based on sweeping the plane with a horizontal line and computing the Voronoi diagram as the line sweeps
- Straight-forward approach won't work
- The unanticipated event can not be recognized until p_4 is known
- Therefore the sweep line technique in a straightforward manner cannot be applied

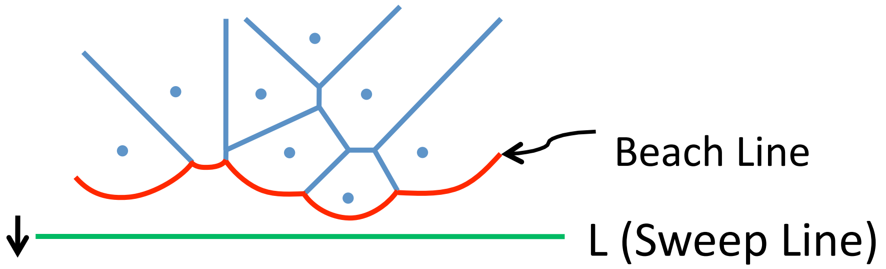


Fortune’s Algorithm

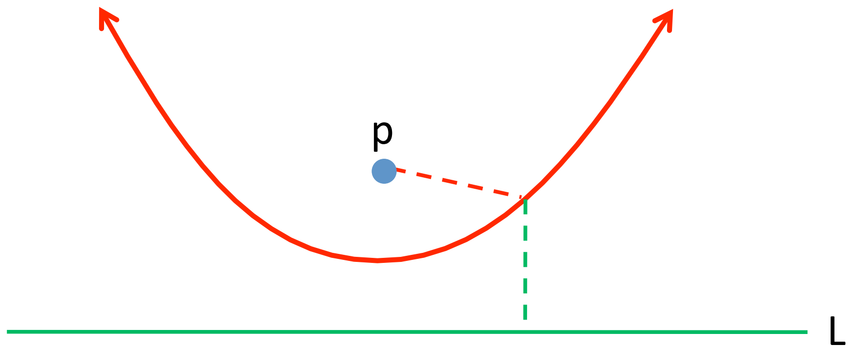
- Fortune applied a transformation that alters the way the distances are measured in the plane.
- The resulting diagram had the same topological structure as the Voronoi diagram but its edges were **parabolic arcs** rather than straight line segments
- Sweeping object was a straight line
- It was an easy matter to “undistort” it to produce the correct Voronoi diagram

Fortune's Algorithm

- Presentation here will be different:
  - Our sweep line is a distorted one. It is called beach line.
  - This distorted sweep line is created by the interactions of the points already swept and a horizontal line moving ahead of the distorted line
  - The distorted sweep line is called the **beach line** and the beach line follows just behind the horizontal sweep line



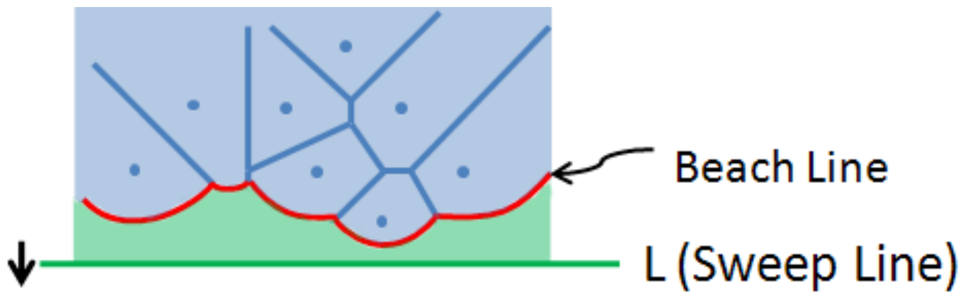
Fortune’s Algorithm



- Point on the parabolic curves are equidistant to p and L

Fortune’s Algorithm

- We make sure that the Voronoi diagram cannot be affected by anything that lies below the horizontal sweep line
- To do this, we will subdivide the half-plane lying above the horizontal sweep line into two regions:
  - All points that are closer to some point above the horizontal sweep line than to the sweep line itself (blue)
  - All points that are closer to the sweep line than any site above the sweep line (green)



Fortune’s Algorithm

- The beach line consists of the lower envelope of parabolas, one for each point. Note that some parabolas will not contribute to the beach line
- **Lemma:** The beach line is an x-monotone curve made up of parabolic arcs. The breakpoints of the beach line lie on Voronoi edges of the final diagram.
  - Parabolas are x-monotone, so is the beach line
  - The breakpoints are equidistant from the two points and the sweep line

Fortune’s Algorithm

- The algorithm consists of simulating the growth of the beach line as the sweep line moves downward and, in particular, tracing the paths of the breakpoints as they travel along the edges of the Voronoi diagram.
- We will maintain:
  - **Sweep line status**:
    - maintain the current location (y-coordinate) of the sweep line.
    - Store, in left-to-right order, the set of points that define the beach line
    - **Note:** the algorithm never stores the beach line itself. The beach line exists solely for visual (conceptual) purposes.

Fortune’s Algorithm

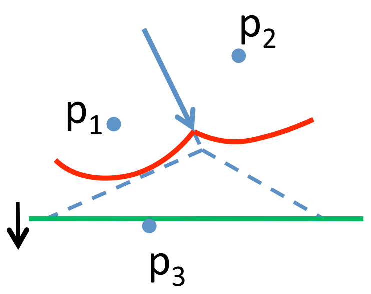
- The set of points that are equidistant from the sweep line to their nearest site above the sweep line is called the beach line
- This implies that the Voronoi diagram above the beach line cannot be affected by any points lying below the sweep line.
- Hence the portion of the Voronoi diagram that lies above the beach line is “safe”. (i.e. we can compute it without knowing about which points are still to appear below the sweep line.)

Fortune’s Algorithm

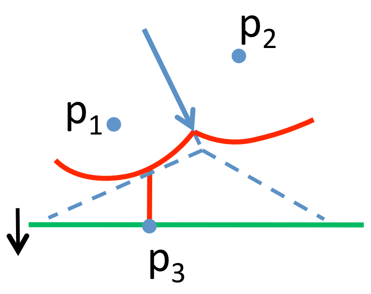
- **Events:** There are two types of events
  - **Point Event:**
    - When the sweep line passes over a new points.
    - A new parabola will be inserted into the beach line.
  - **Vertex Event:** (Circle events in the text)
    - When the length of a parabolic arc shrinks to zero
    - The arc disappears and a new Voronoi vertex will be created at this point

Fortune's Algorithm

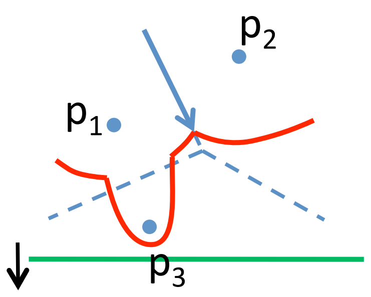
Point Event:



$< p_1, p_2 >$



$< p_1, p_3, p_1, p_2 >$



$< p_1, p_3, p_1, p_2 >$

- To process a point event:
  - Determine the arc of the beach line directly above the new point
  - Split the arc into two by inserting a new infinitesimally small arc at this point
  - As the sweep proceeds this arc will start to widen

Fortune’s Algorithm

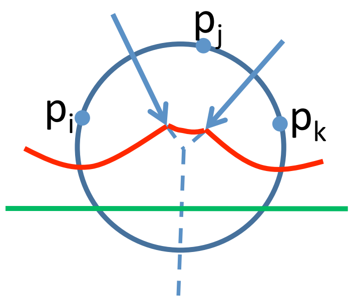
- This (point event) is the only way that new arcs can be introduced to the beach line. (Proof is in the book of Van Krevald et al.)
- Therefore, at most 2n-1 arcs can appear on the beach line. (Net increase: two arcs per point)
- Point events are known in advance
- We sort the points by y-coordinate

Fortune's Algorithm

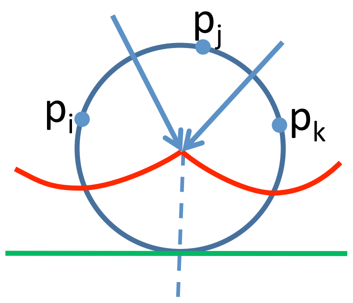
- **Vertex Events:**
  - Vertex events are generated dynamically as the algorithm runs
  - Like the line segment plane sweep algorithm, vertex events are generated by neighbours on the beach line. However, unlike the segment intersection where pairs of consecutive segments generate events, here triples of points generate the event

Fortune's Algorithm

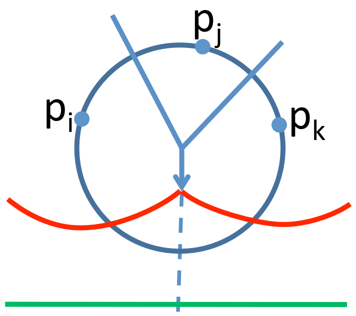
1. P_i, P_j, and P_k whose arcs appear consecutively on the beach line. The circumcircle lies partially below the sweep line



2. Circumcircle is empty and the center is equidistant to p_i, p_j, p_k, and L. The center is a Voronoi vertex.



3. The arc of p_j disappears from the beach line



Fortune’s Algorithm

- **<u>(Partial) Voronoi Diagram:</u>**
  - The partial Voronoi diagram that has been constructed so far will be stored in a DCEL.
  - Unbounded edges are tackled by constructing the Voronoi diagram within a box large enough to include all the Voronoi vertices inside it

Fortune’s Algorithm

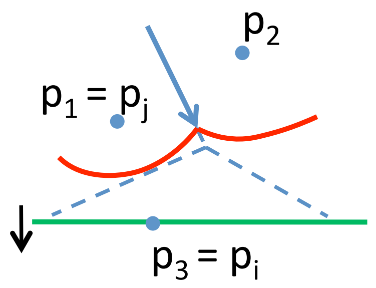
- **<u>Event Queue:</u>**
  - **<u>Point Event</u>**
    - A priority queue of the points. It allows Extract_Max()
  - **<u>Circle (Vertex) Event</u>**
    - For each consecutive triple pi, pj, and pk on the beach line we compute the circumcircle of these events. If the lower endpoint of the circle lies below the sweep line, we create a vertex event whose y-coordinate is the y-coordinate of the bottom endpoint of the circumcircle.

Fortune’s Algorithm

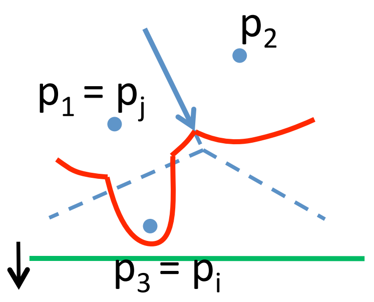
- **Operations to be Performed:**
  - **Point Event**
    - Given a fixed location of the sweep line, determine the arc of the beach line that intersects a given vertical line (O(logn) time search is possible)
    - Computing the predecessor and the successor on the beach line
    - Insert a new arc p_i within a given arc p_j, thus splitting the arc for p_j into two. This creates three arcs: p_j, p_i, p_j (can be done in O(logn) time
    - Delete an arc from the beach line (can be done in O (logn) time)

Fortune's Algorithm

- **<u>Operations to be Performed:</u>**
  - **<u>Point Event:</u>** p_i lies on L
    - Determine the arc of the beach line directly above p_i. Let p_j be the corresponding site.
    - Replace the arc p_j by arcs p_j, p_i, p_j
    - Create new edge (dangling) for the Voronoi diagram which lies on the bisector between pi and p_j
    - Old triples involving p_j is delete and some new triples involving p_i will be inserted



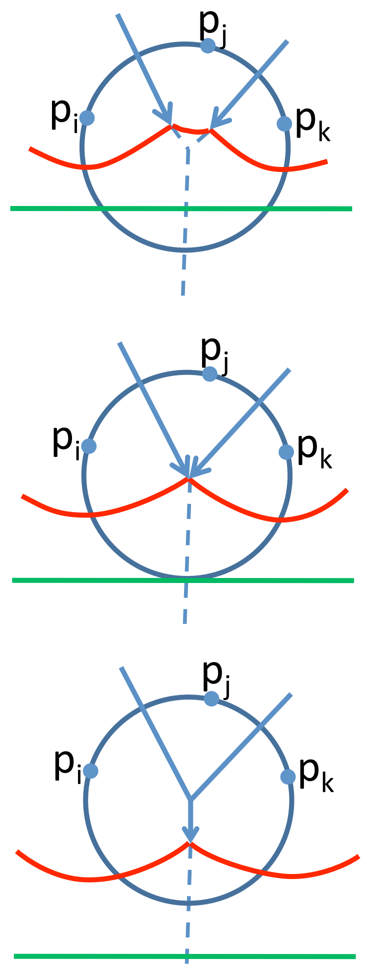
$< p_1, p_2 >$



$< p_1, p_3, p_1, p_2 >$

Fortune's Algorithm

- **<u>Operations to be Performed:</u>**
  - **<u>Vertex Events:</u>**
    - Let p_i, p_j, p_k be the three consecutive points that generate this event (from left to right).
    - We delete the arc for p_j from the beach line
    - We create a new vertex in the Voronoi diagram and tie the edges (p_i, p_j) and (p_j, p_k) to it and start a new bisector (p_i, p_k) that starts growing down below



Fortune’s Algorithm

- **Lemma:** Both the point event and the vertex event can be handled in O(logn) time

- **Theorem:** Fortune’s algorithm computes the Voronoi diagram in O(nlogn) time. The storage space requirement is O(n)

Maximizing Angles and Edge Flippings

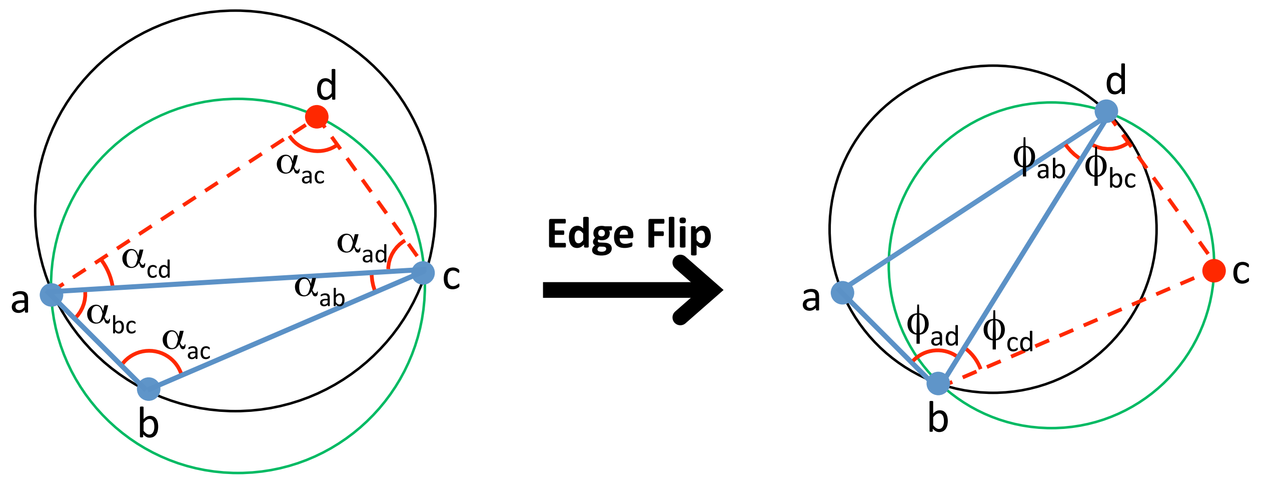
- Among all triangulations, the Delaunay triangulation maximizes the minimum angle.
  - DT avoids skinny triangles
- Much stronger statement: Among all the triangulations with the same smallest angle, the DT maximizes the second smallest angle, and so on.
- In particular, any triangulation can be associated with the sorted angle sequence i.e. the increasing sequence of angles (α_1, α_2, …, α_m) appearing in the triangles of the triangulation.

Maximizing Angles and Edge Flippings

- **Theorem:** Among all triangulations of a given point set, the Delaunay triangulation has the lexicographically largest angle sequence
- **Proof:** Consider an angle sequence (β_1, β_2, …, β_m) of a non-Delaunay triangulation. We want to show that in this case there exists another triangulation (β_1', β_2', …, β_m') which is lexicographically larger.

Maximizing Angles and Edge Flippings

- Since the angle sequence ($\beta_1, \beta_2, \ldots, \beta_m$) comes from a non-Delaunay triangulation, there exists a case like:

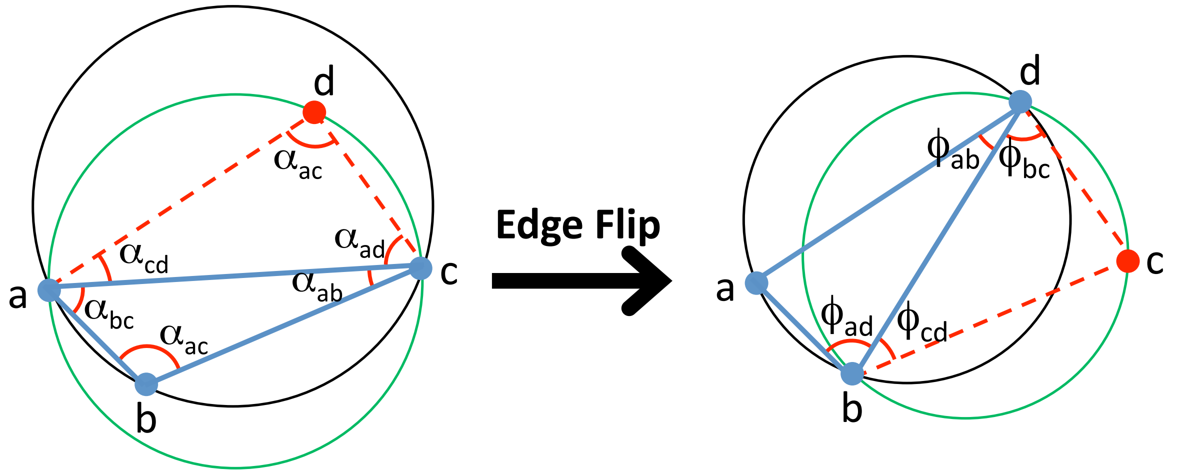


- **D lies inside the circumcircle of Δabc**
- **Implies that b lies inside the circumcircle of Δacd**

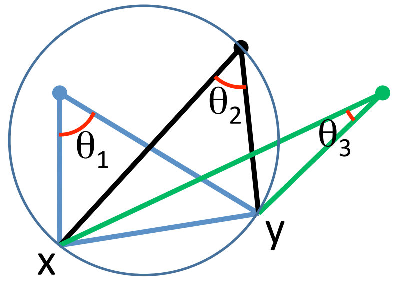
- **Circumcircle of Δabd and Δbcd are empty**

Maximizing Angles and Edge Flippings

- $\phi_{ab} > \alpha_{ab}$
- $\phi_{bc} > \alpha_{bc}$
- $\phi_{cd} > \alpha_{cd}$
- $\phi_{da} > \alpha_{d}$



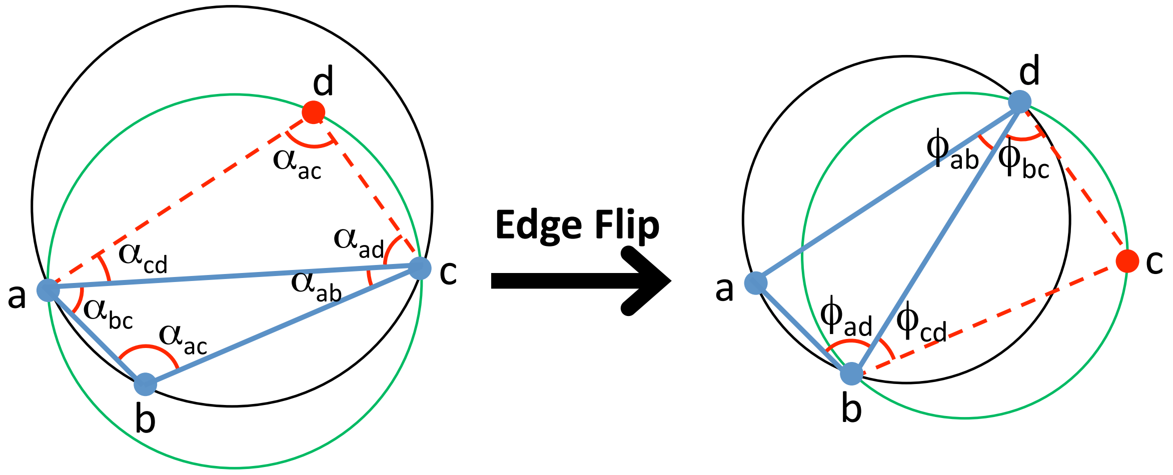
- This comes from the following fact (Thale's Theorem):



$\theta_1 > \theta_2 > \theta_3$

Maximizing Angles and Edge Flippings

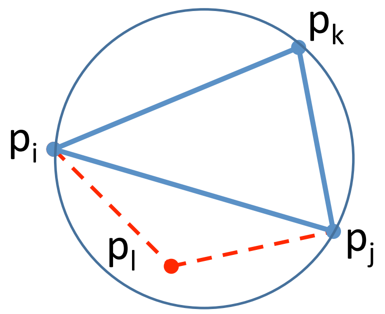
- $\phi_{ab} > \alpha_{ab}$
- $\phi_{bc} > \alpha_{bc}$
- $\phi_{cd} > \alpha_{cd}$
- $\phi_{da} > \alpha_{da}$



- There are two other angles that need to be compared as well. It is not hard to show that, after swapping, these other angles can not be smaller than the minimum of α_{ab} , $\alpha_{bc,}$ $\alpha_{cd,}$ α_{da}
- Since there are only a finite number of triangulations, this process must terminate with the lexicographically macimum triangulation and this triangulation must satisfy the empty circle condition. Hence it is the Delaunay triangulation

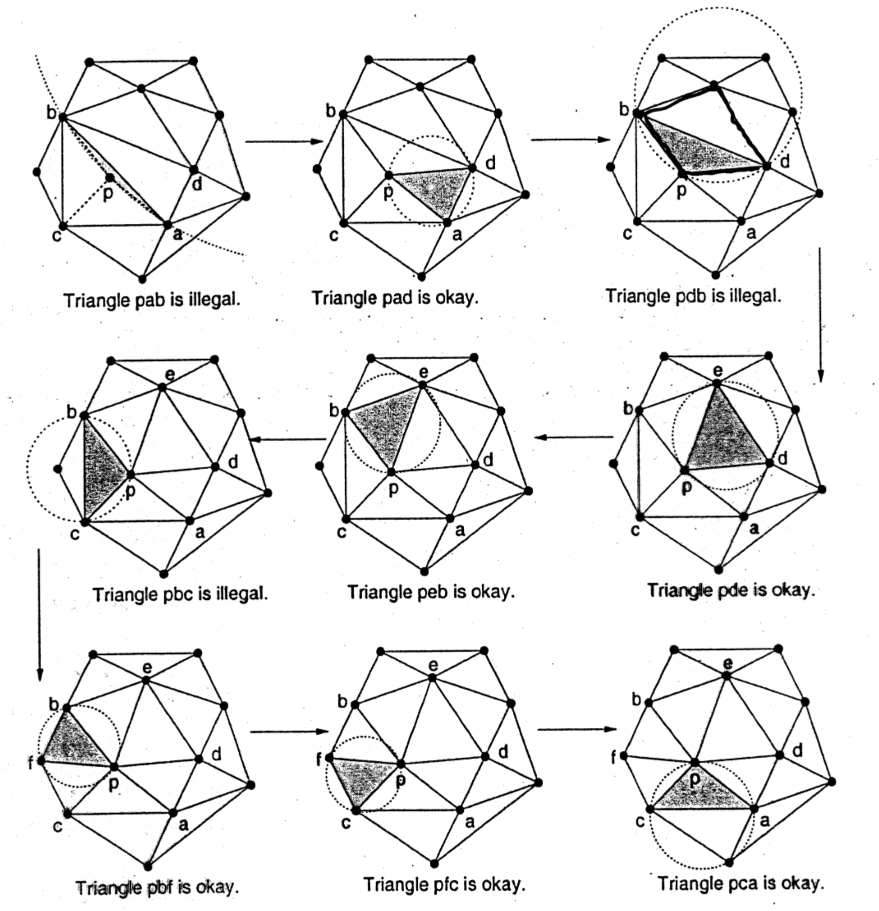
Legal Triangulation

- **Definition:** an edge of a triangulation is illegal if we can locally increase the smallest angle by flipping that edge
- **Definition:** a legal triangulation is one that does not contain any illegal edges
- **Algorithm:** Legal Triangulation (T)
  - Input: Some triangulation of P
  - Output: A legal triangulation of P

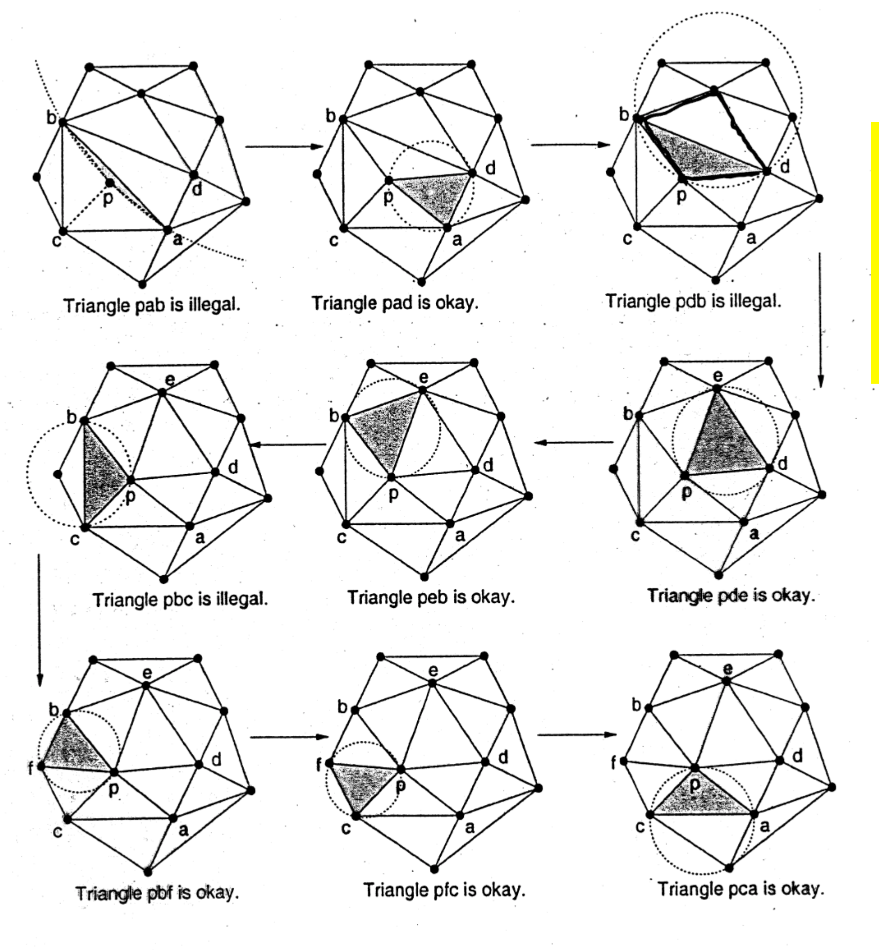


```
While T contains an illegal edge $p_i p_j$
    do    /*flip $p_i p_j$ */
          let $p_i p_j p_k$ and $p_i p_j p_l$ be two triangles adjacent to $p_i p_j$
          remove $p_i p_j$ from T and add $p_k p_l$ instead
Return T
```

Legal Triangulation



Legal Triangulation



We are assuming that prior to adding p the trianlgulation was legal.

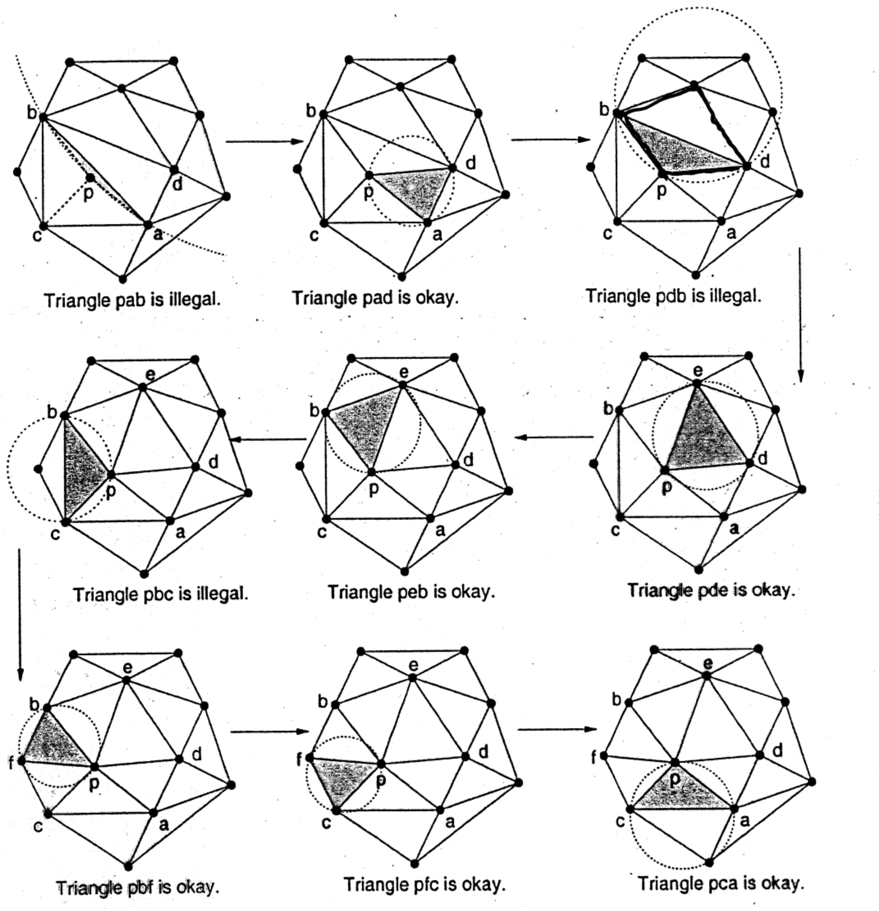
Legal Triangulation

We maintain a stack to store the triangles that need to be tested. After p is added, the three triangles pab, pbc and pca are added to the stack.

Stack

Bottom

pca
pbc
pab



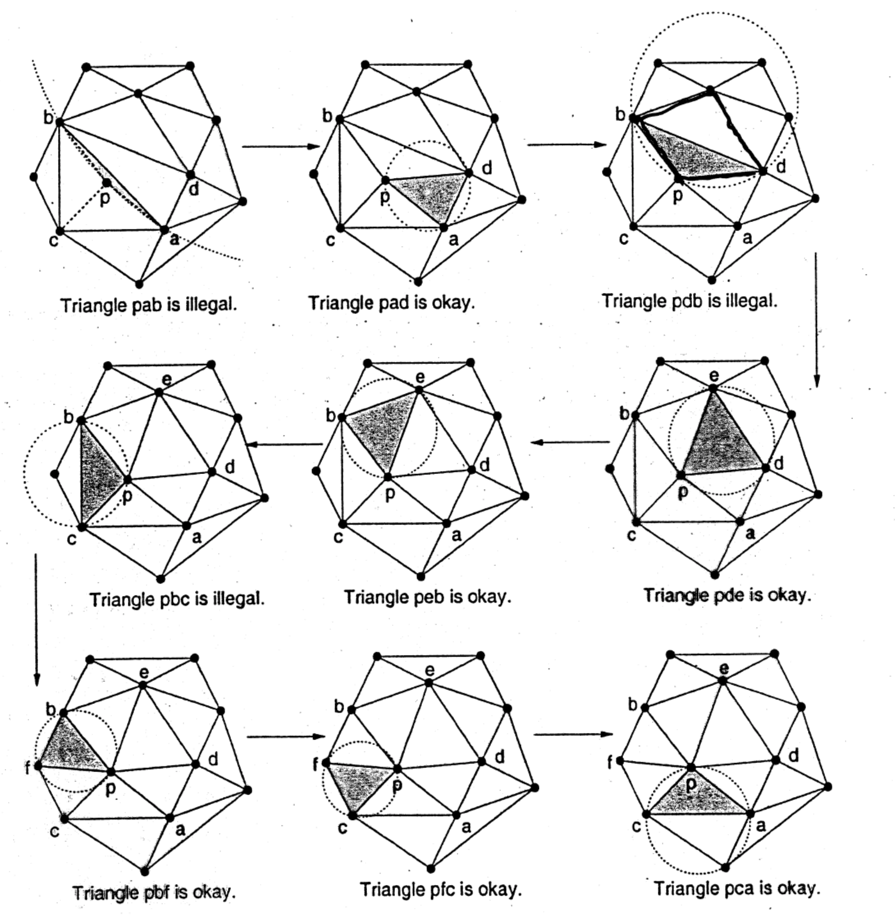
Legal Triangulation

Stack
Bottom

pca
pbc
~~pab~~

Triangle pab is being considered.

It is illegal. Flip the edge ab. Creates two new triangles pad and pdb.



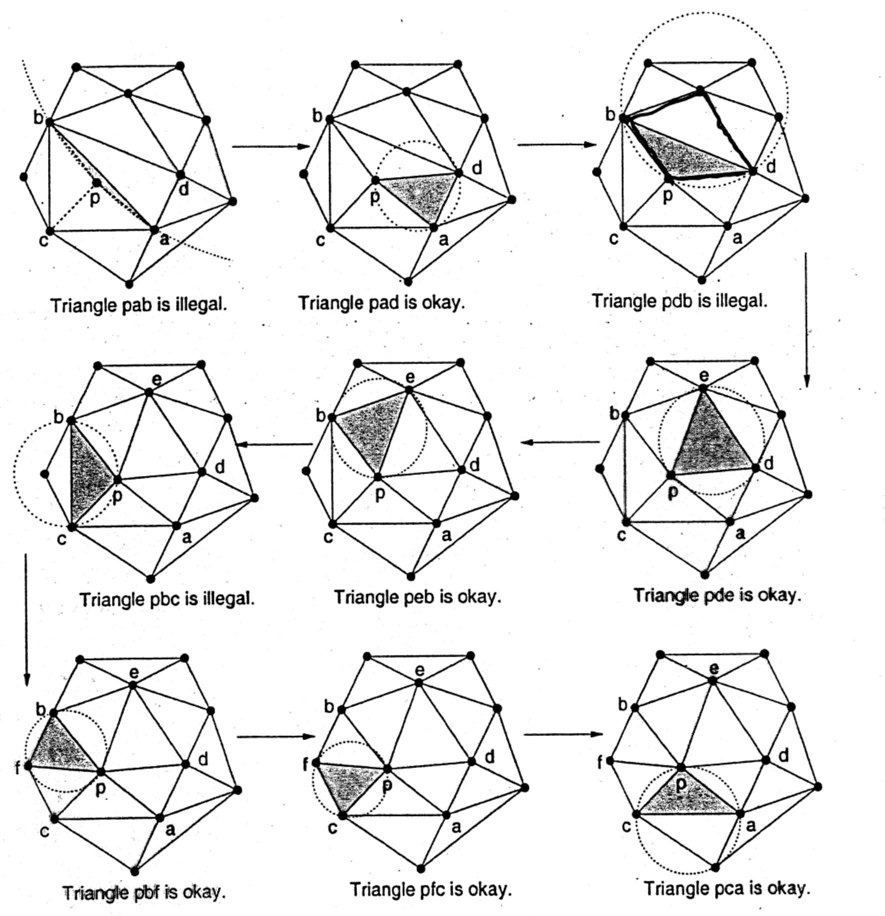
Legal Triangulation

Stack
Bottom

pca
pbc
pdb
pad

Triangle pad is considered next.

It is legal.



Legal Triangulation

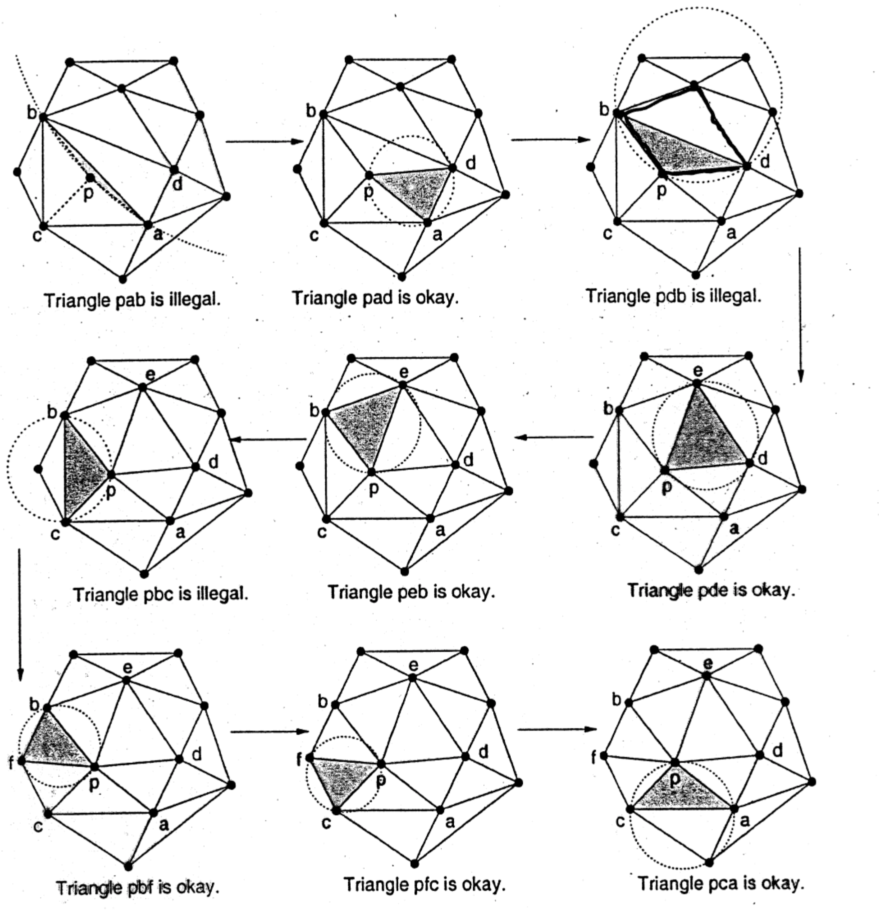
Stack Bottom

pca
pbc
pdb

Triangle pdb is considered next.

It is illegal.

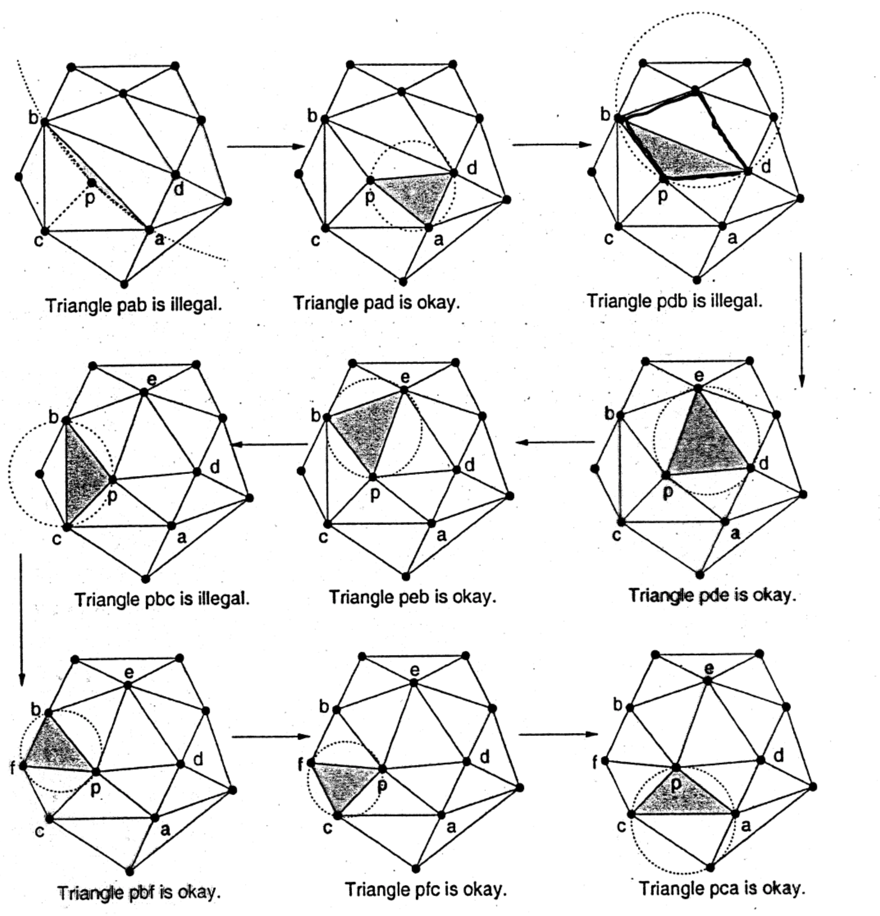
Edge pd is flipped. Two new triangles are created.



Legal Triangulation

Stack Bottom

pca
pbc
peb
pde



Legal Triangulation

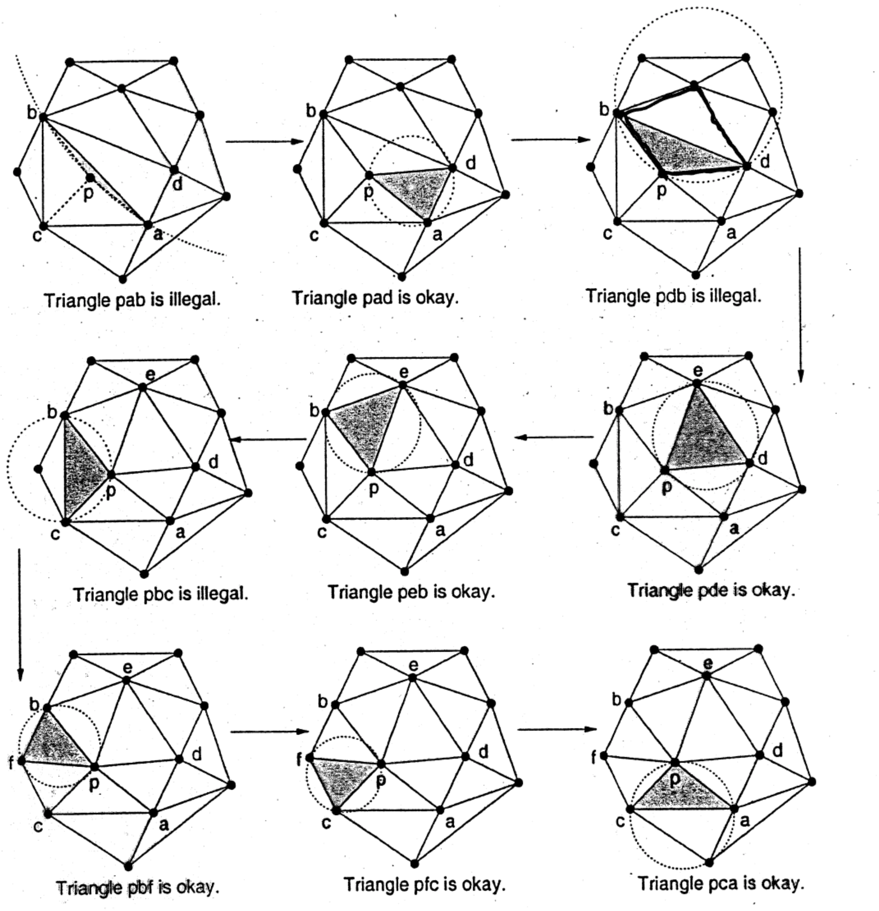
Stack
Bottom

pca
pbc
peb
pde

Triangle pde is legal.

Triangle peb is legal.

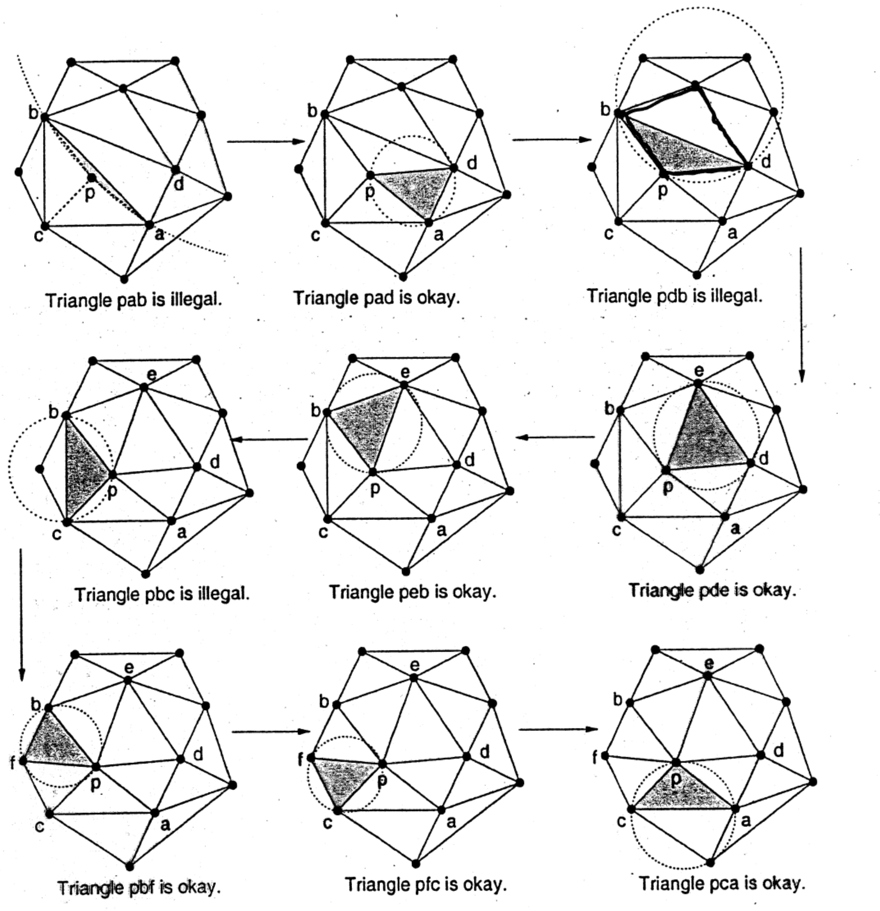
Triangle pbc is illegal.



Legal Triangulation

Stack Bottom

pca
pfc
pbf



Triangle pab is illegal.

Triangle pad is okay.

Triangle pdb is illegal.

Triangle pde is okay.

Triangle peb is okay.

Triangle pbc is illegal.

Triangle pbf is okay.

Triangle pfc is okay.

Triangle pca is okay.

Legal Triangulation

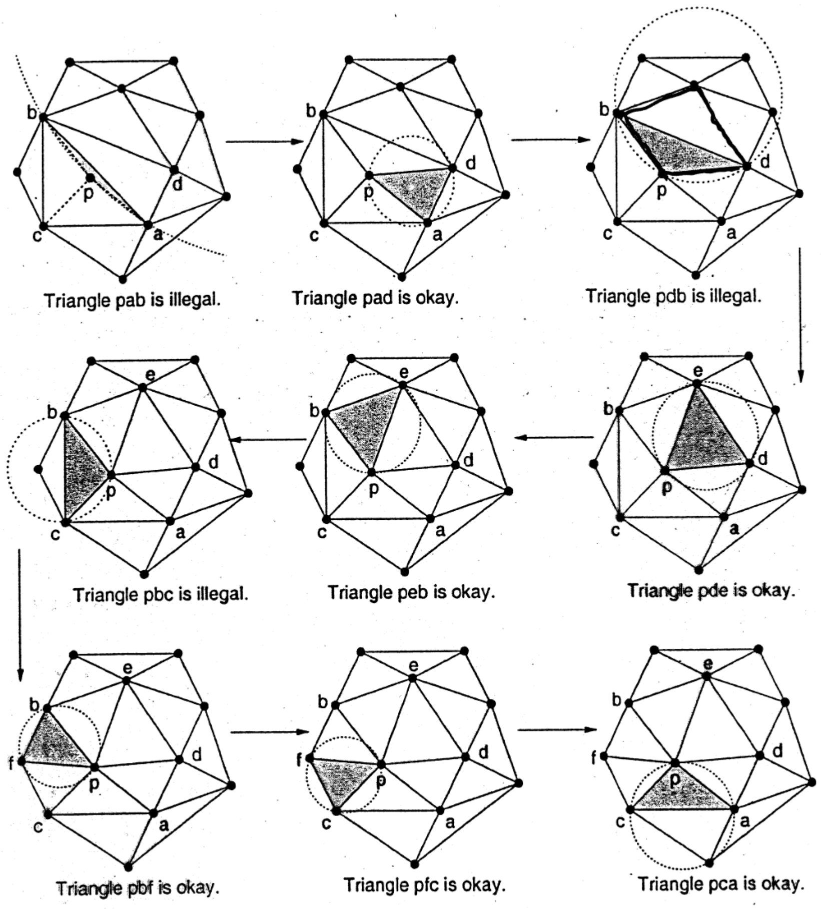
Stack Bottom

pca
pfc
pbf

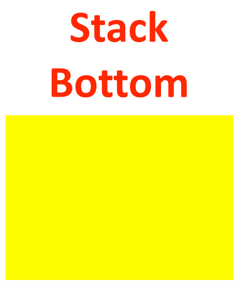
Triangle pbf is legal.

Triangle pfc is legal

Triangle pca is legal.



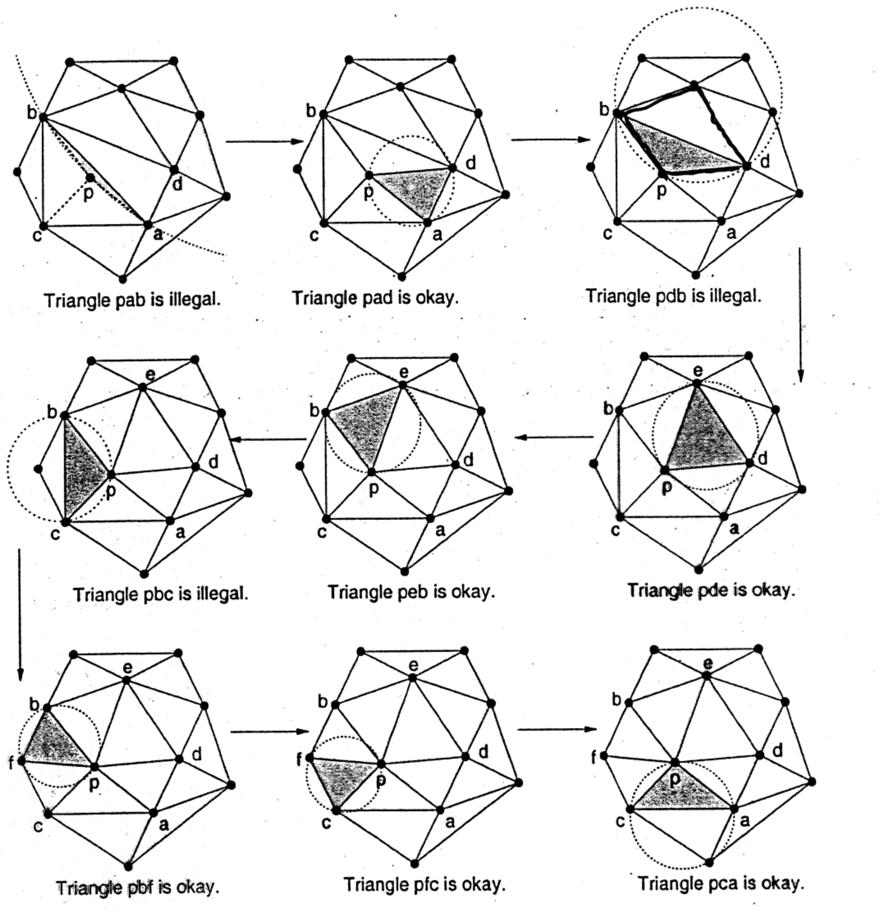
Legal Triangulation



Stack is empty.

The triangulation of the points including p is legal.

Total cost is proportional to the number of triangles that contain p.



Analysis

- Given the initial triangulation of the point set S, the Delaunay triangulation of S can be obtained by simply edge flips in O(n+k) time where k is the number of edge flips.
- We are assuming here that the initial triangulation is stored in a DCEL data structure
- What is k in the worst case?
    - It is $O(n^2)$
- But in practice it is extremely small (expected O(n))
- One can obtain the initial triangulation of S in O(nlogn) time (How?)
- In this case the stack contains all the triangles to start with.