Metric Spaces: Distances modeled as metric space \( (X, f) : f : X \times X \rightarrow \mathbb{R}^{\geq 0} \), s.t. for all \( p, q, r \in X \):

- **Symmetry**: \( f(p, q) = f(q, p) \)
- **Positivity**: \( f(p, q) \geq 0 \) and \( f(p, q) = 0 \) iff \( p = q \)
- **Triangle Inequality**: \( f(p, q) \leq f(p, r) + f(r, q) \)

**Euclidean Distance**: for \( p, q \in \mathbb{R}^d \):

\[
\| p - q \| = \left[ \sum (p_i - q_i)^2 \right]^{\frac{1}{2}}
\]

**Voronoi Diagram**: A fundamental structure for metric spaces.

Given a point set \( P = \{p_1, \ldots, p_n\} \) in \( \mathbb{R}^d \) called sites, we want to subdivide space based on each site's "region of influence"
**Def:** Voronoi cell for site \( p_i \)

\[
V_\mathcal{P}(p_i) = \{ q \in \mathbb{R}^d \mid \| p_i - q \| < \| p_j - q \|, \forall j \neq i \}
\]

**Obs:**
- Voronoi cells are **disjoint**
- For Euclidean dist, Voronoi cells are (possibly unbounded) **convex polyhedra**

Let \( h(i, j) = \{ q \mid \| p_i - q \| < \| p_j - q \| \} \)

\( h(i, j) \) - halfspace bounded by perpendicular bisector between \( p_i \) and \( p_j \)

\[
\text{Vor}(p_i) = \bigcap_{j \neq i} h(i, j)
\]

- intersection of halfspaces \( \Rightarrow \text{polytope} \)
**Def:** \( \text{Vor}(P) \) is the subdivision (cell complex) induced by \( P \)'s Voronoi cells.

- \( \text{Vor}(P) \) covers \( \mathbb{R}^d \)
- Has \( n \) cells (faces of dim \( d \))
- Polyhedral subdivision (for Euclidean dist)
- Combinatorial complexity:
  - \( \mathbb{R}^2: O(n) \) edges + vertices
  - \( \mathbb{R}^d: O(n^{[d/2]+1}) \) size [Closely related to convex polytopes in \( \mathbb{R}^{d+1} \)]

**Many applications:**

**Nearest neighbor search:**

Preprocess a set of sites \( P = \{ p_1, \ldots, p_n \} \subset \mathbb{R}^d \)

s.t. given any query point \( q \in \mathbb{R}^d \)

can find \( q \)'s nearest site

**How?**

- Compute \( \text{Vor}(P) \)
- Build a point-location data structure for \( \text{Vor}(P) \)
  
  [Optimal in \( \mathbb{R}^2 \). Not as good in \( \mathbb{R}^d \).]
**Point-based Clustering:**
- Given set $T$ of training points, group them into $k$ clusters
- Clusters are defined by $k$ cluster centers $\{c_1, \ldots, c_k\}$
- Cluster membership based on closest center
- $k$-means clustering

**Variations:**
- Other metrics: $L_1$-Vor diagram (Manhattan distance)
- Weighted pts:
  - Multiplicative: $\text{dist}(q, p_i) = \alpha_i \| p_i - q \|
  - Additive: $\text{dist}(q, p_i) = \| p_i - q \| + \omega_i$
- $k^{th}$ Nearest:
  - $\text{Vor}_k(P) = \text{subdivide based on } k^{th}$ closest

[Diagram showing clustering with centroids $c_1, c_2, c_3, c_4$ and points assigned to clusters]
\( \text{Vor}_n(P) = \text{farthest point Vor. diag} \)
- Other shapes:
  - Voronoi diagram of line segments
  - Medial axis of polygon centers of maximal disks

**Properties of the Voronoi Diagram:**

**Empty-circle Property:**

A pt \( q \) is on an edge of the Vor. diag iff there is a circle centered at \( q \) that passes through 2 sites \( p_i \) and is otherwise empty.

**Circumcircle Property:** A pt \( v \) is a vertex of the diagram iff it is the center of a circle passing through 3 sites \( p_i \) and is otherwise empty.
Hull Property: A site \( p_i \) has an unbounded Voronoi cell iff \( p_i \) is on boundary of convex hull of \( P \).

Constructing Voronoi Diagrams in \( \mathbb{R}^2 \):

- **Incremental** - add a site; update (best if randomized)

- **Divide + Conquer** - \( O(n \log n) \)

- **Plane Sweep** (this lecture)
  - Fortune's Algorithm - \( O(n \log n) \)

**Difficulty with Plane Sweep:**

How can you process these when you haven't discovered \( p_i \) yet?
Clever twist: We’ll maintain two sweeping structures: sweep line + beach line

Def: Given a set of pts $R$ and pt $q$, define

$$\text{dist}(q, R) = \min_{p \in R} ||p - q||$$

Given a sweep line $l$ (horizontal + moving down) define

$P^+(l)$ to be sites lying above $l$

$P^-(l)$ to be sites lying below $l$

Given sweep line $l$, define the beach line to be the set of pts $q \in \mathbb{R}^2$ that are equidistant from $P^+(l)$ and $l$

$$\text{beach}(l) = \{ q \in \mathbb{R}^2 \mid \text{dist}(q, P^+(l)) = \text{dist}(q, l) \}$$
Beach-line Structure:
The points equidistant to a site \( p \) and line \( l \) form a parabola (wider as \( p \) is higher).

The beach line is the lower envelope of these parabolas for all sites in \( P^*(l) \).

- Beach line is \( x \)-monotone
- A single site may contribute 0, 1, or multiple arcs

Total complexity is \( \mathcal{O}(|P^*(l)|) = \mathcal{O}(n) \)  
[Proof: Exercise]
Key: The portion of $\text{Vor}(S)$ above the beach line is "safe" from sites lying below $l$.

Fortune's Algorithm:

Sweep-line status:
- $y$-coord of sweep line
- seq. of sites (left to right) that contribute arc to beach line (e.g. $<2,1,2,3,2>$)
- Parabolic arcs not computed
- Breakpoints generated as needed

$P_i$ cannot affect $\text{Vor}(S)$ above beach line ($l$)
Voronoi diagram: Portion of Voronoi diagram (rep. as DCEL) above beach line is stored/updated.

Events:

Site event: Sweep line passes over a site

Vertex event (circle event):
- A new Voronoi vertex is discovered
- An arc on beach line vanishes

Priority Queue: Stores y-coords for sweep line at events.

Site events: Easy—just y-coord of site (static)

Vertex events: Tricky! (see below)
Scheduling vertex events:

- For each consecutive triple \(<... p_i, p_j, p_k...>\) on beach line compute lowest y-coord of circumcircle \((p_i, p_j, p_k)\).
- Schedule vertex event when sweep line reaches this y-coord.

Site Event: for site \(p_i\):  
(1) Find arc of beach line above \(p_i\).  
(2) Split this arc: \(<... p_j, ...> \Rightarrow <... p_j, p_i, p_j, ...>\).  
(3) Create a new "dangling" edge (between \(p_i + p_j\)) add to Voronoi diagram.
(4) Update priority queue vertex events (below)

Vertex Event: for triple \( \langle p_i, p_j, p_k \rangle \)

(1) Delete \( p_j \)'s arc from beach line

\[ \langle \ldots p_i p_j p_k \ldots \rangle \Rightarrow \langle \ldots p_i p_k \ldots \rangle \]

(2) Create new Voronoi vertex joining edges \( p_i p_j + p_j p_k \) in diagram

(3) Start new (partial) Voronoi edge for \( p_i p_k \)

(4) Update priority queue vertex events

Analysis: - \( O(n) \) events
- \( O(\log n) \) per event
- \( O(n\log n) \) total time