CSE520 Computational Geometry
Lecture 23
Quadtrees and Approximate Nearest Neighbor Searching

Antoine Vigneron

Ulsan National Institute of Science and Technology

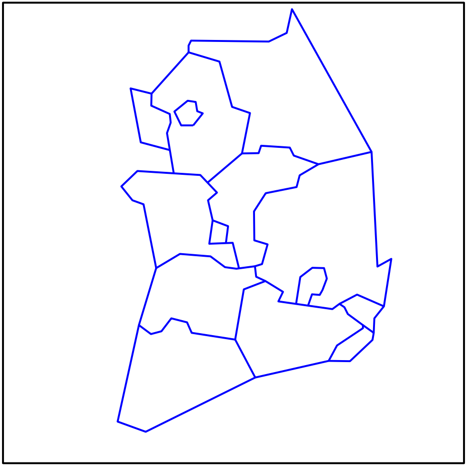
June 16, 2020

Course Organization

- Today, I will will present *quadtrees*.
- A quadtree is a tree data structure that is suitable for geometric data in low dimension.
- I will also present an application to near-neighbor searching.

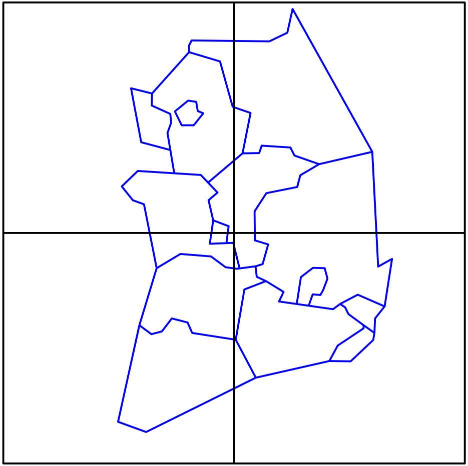
- References:
- Sariel Har Peled's book, chapters 2 and 17.

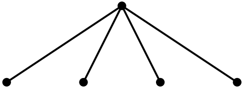
Example



- Find a suitable square box containing the input subdivision.

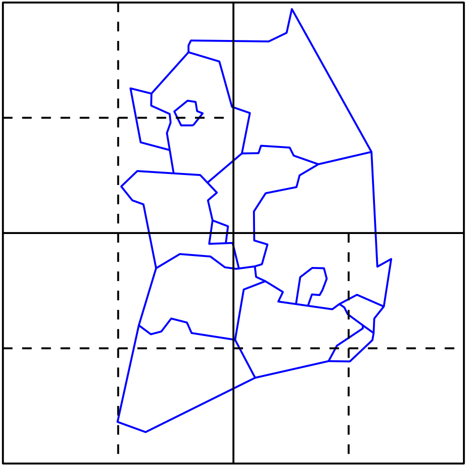
Example

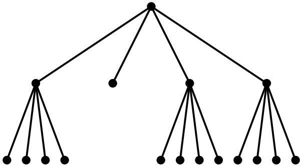




- Partition the outer box into 4 boxes.

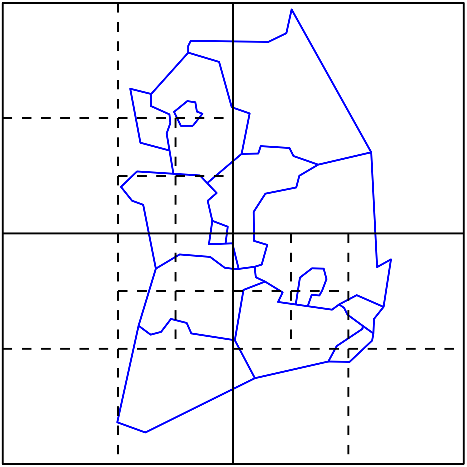
Example

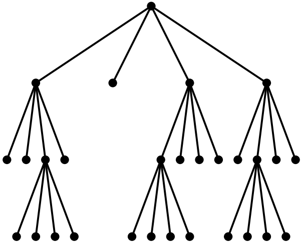




- Keep subdividing any box that overlaps with 5 regions or more.
- Remark: We could have used any other constant instead of 5.

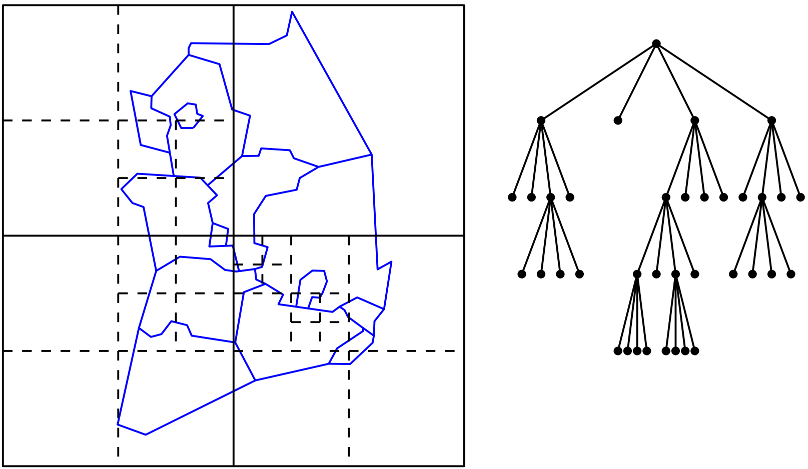
Example





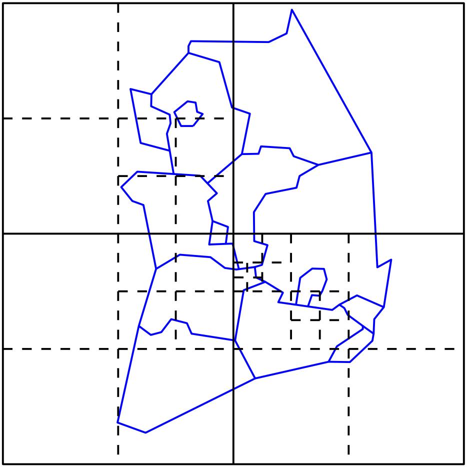
- 3 regions are subdivided.

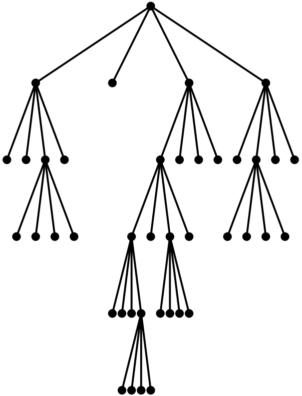
Example



- 2 regions are subdivided.

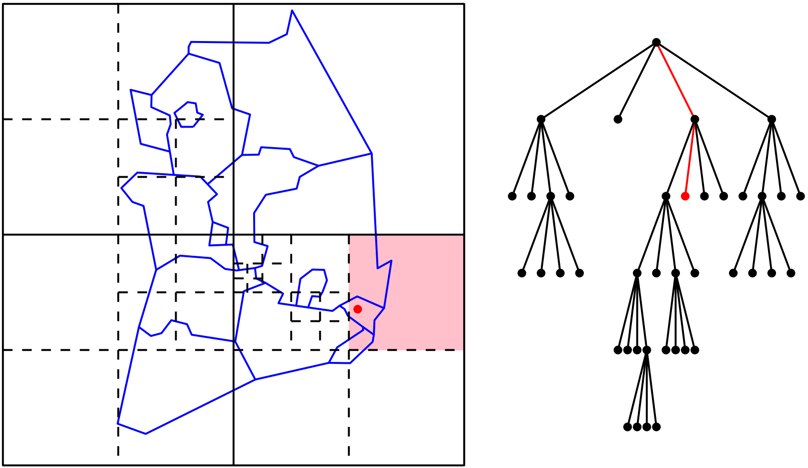
Example





- Last step of the construction.

Example



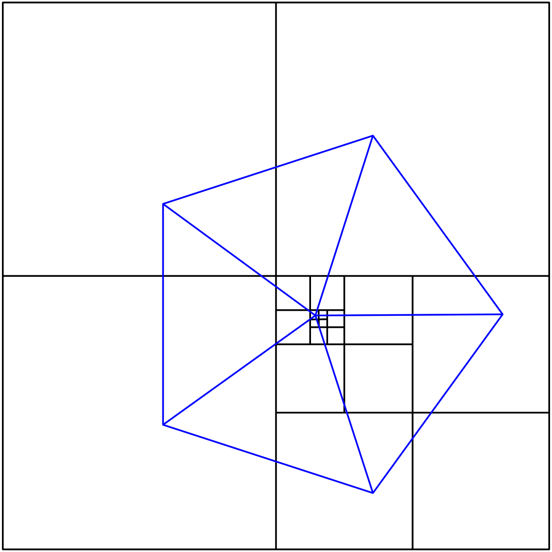
- In order to locate the red point, we go down to the leaf containing it, and check each of the region overlapping with this leaf box.

Quadtrees

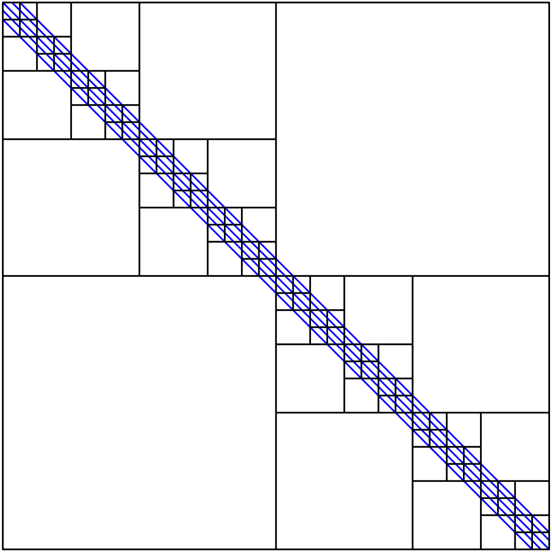
- Each node of the graph is associated with a conflict list.
- The conflict list of a node u contains all the input regions that overlap with the box corresponding to u.
- Each box that overlaps with more than 4 input regions is subdivided into 4 boxes of half the size.
- We only need to record the conflict list of the leaves.
- This data structure is a *quadtree*.

- In practice, this data structure can be good if the regions have a nice shape.
- But for some input subdivision it may have a very large size. (See next two slides.)

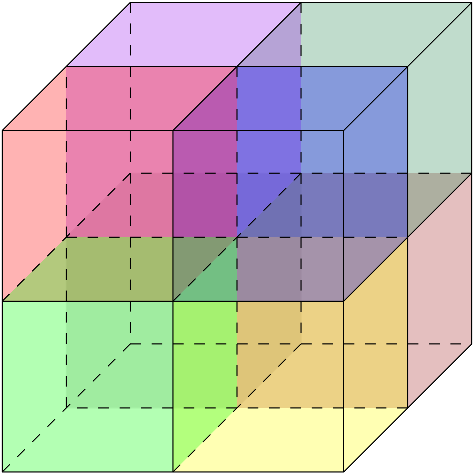
Quadtrees

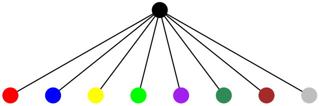


Quadtrees



Quadtrees



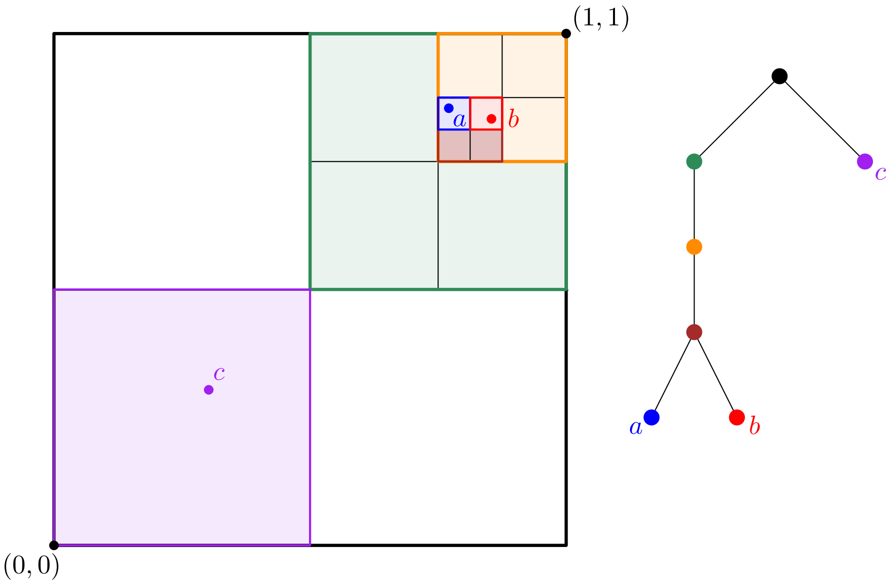


Quadtrees

- Quadtrees generalize to arbitrary dimension.
- A box is split according to each dimension.
- A node can have up to 2^d children.
- So it is suitable for *fixed* dimension, i.e. $d = O(1)$.

- From now on, we will assume $d = O(1)$ and we will focus on quadtrees for set of points.
- We will subdivide a box whenever it contains more than one input point.

Recording a Point-Set in a Quadtree



Recording a Point-Set in a Quadtree

- Let P be a set of n points in $\mathbb{R}^d$.
- We assume that P is contained in the box $[0, 1]^d$.
- A *quadtree box* is a box obtained by recursively partitioning the box $[0, 1]^d$ into 2^d boxes. In other words, a quadtree box is of the form

$$\left[\frac{k_1}{2^i}, \frac{k_1+1}{2^i}\right] \times \cdots \times \left[\frac{k_d}{2^i}, \frac{k_d+1}{2^i}\right]$$

  where $k_1, \dots, k_d \in \{0, \dots, 2^i\}$ and $i \in \mathbb{N}$.
- The *level* of this box is i.

Recording a Point-Set in a Quadtree

- Each node v of the quadtree records a quadtree box denoted $\mathrm{box}(v)$.
- The root node is associated with the box $[0, 1]^d$.
- Each leaf corresponds to exactly one point of P.
- Whenever $\mathrm{box}(v)$ contains more than one point of P, we create 2^d children of v corresponding to the 2^d boxes obtained by partitioning $\mathrm{box}(v)$.

- In order to save space, we delete the nodes v such that $\mathrm{box}(v) \cap P = \emptyset$. (See Slide 15.)
- For instance, we replace them with a null pointer.

Recording a Point-Set in a Quadtree

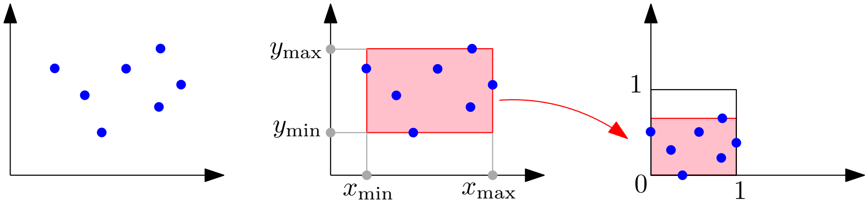
- The *spread* of P is
$$\Phi(P) = \frac{\mathsf{diam}(P)}{\mathrm{dmin}(P)}$$
where
$$\mathsf{diam}(P) = \max_{p,q \in P} d(p,q)$$
is the diameter of P and
$$\mathrm{dmin}(P) = \min_{\substack{p,q \in P \\ p \neq q}} d(p,q).$$
- When there is no ambiguity, we write Φ instead of $\Phi(P)$.

Recording a Point-Set in a Quadtree

Theorem

Let P be a set of n points contained in $[0,1]^d$, and such that $\operatorname{diam}(P) \geqslant 1$. *Then the quadtree recording P has height $O(\log \Phi)$, size $O(n \log \Phi)$, and can be constructed in $O(n \log \Phi)$ time.*

- We can always enforce the conditions $P \subset [0,1]^d$ and $\operatorname{diam}(P) \geqslant 1$ by scaling and translating P appropriately. It can be done in $O(n)$ time:



Proof

- We first prove that the height of the quadtree is $O(\log \Phi)$.
- Let u be an internal node at level i.
- Then $\mathrm{box}(u)$ contains at least 2 points of P, so

$$\begin{aligned}\mathrm{dmin}(P) &\leqslant \mathrm{diam}(\mathrm{box}(u)) \\ &= \frac{\sqrt{d}}{2^i} \leqslant \mathrm{diam}(P)\frac{\sqrt{d}}{2^i}\end{aligned}$$

  so $2^i \leqslant \Phi\sqrt{d}$, and thus $i \leqslant \log(\Phi) + \log(d)/2$.
- As $d = O(1)$, it means that $i = O(\log \Phi)$.

- So all internal nodes are at level $O(\log \Phi)$.
- It implies that the height of the quadtree is $O(\log \Phi)$.

Proof

- As the quadtree has height $O(\log \Phi)$ and has n leaves, it has at most n nodes per level, and thus it has $O(n \log \Phi)$ nodes.
- As each node records at most one point, this data structure uses $O(n \log \Phi)$ space.
- It can be constructed in $O(n \log \Phi)$ time by calling the procedure below with $Q = P$, $b = [0, 1]^d$ and $v =$root.

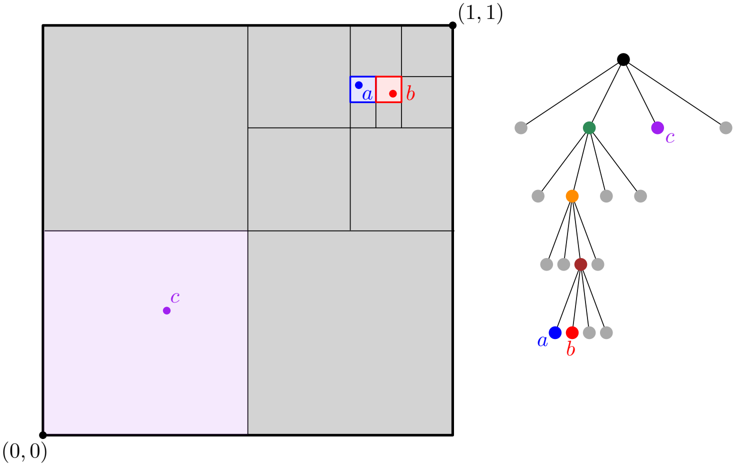
Proof

Pseudocode

procedure $\textsc{ConstructQuadtree}(Q,b,v)$
    if $|Q| = 1$ **then**
        record the point $q \in Q$ at node v
        return
    let $b_1, \dots, b_m$ be the $m = 2^d$ sub-boxes of b
    for $i \leftarrow 1, m$ **do**
        if $Q \cap b_i \neq \emptyset$ **then**
            create a child v_i of v
            $\textsc{ConstructQuadtree}(Q \cap b_i, b_i, v_i)$

- As $d = O(1)$, each level of the quadtree contains at most n nodes, and the height of the quadtree is $O(\log \Phi)$, the total construction time is $O(n \log \Phi)$.

Point Location in a Quadtree



- In this section, we include in the quadtree the empty leaf nodes (grey).
- It increases the number of nodes by a factor less than 4.

Point Location in a Quadtree

Theorem

Given a quadtree of height h, we can preprocess it in linear time so that given a query point q, we can find the leaf containing q in $O(\log h)$ time.

- Remark: We will need to be able to perform hashing in $O(1)$ time per query, which is a common assumption in algorithms design.
- Remember that a quadtree box is of the form

$$\left[\frac{k_1}{2^i}, \frac{k_1+1}{2^i}\right] \times \cdots \times \left[\frac{k_d}{2^i}, \frac{k_d+1}{2^i}\right]$$

where $k_1, \ldots, k_d \in \{0, \ldots, 2^i\}$ and $i \in \mathbb{N}$.

Point Location in a Quadtree

- We record $\mathrm{box}(u)$ for all u in a hash table, using as keys the integers $k_1, \ldots, k_d$ described above.
- As $d = O(1)$, we can construct this data structure in linear time, and each query takes $O(1)$ time.

- If $q = (q_1, \ldots, q_d)$, the quadtree box containing q at level i satisfies $k_i = \lfloor q_i / 2^i \rfloor$ for all i.
- If there is a node u at level i containing q, we can find it in $O(1)$ time by hashing.
- So we can find the leaf node containing u in $O(h)$ time by trying each level separately.

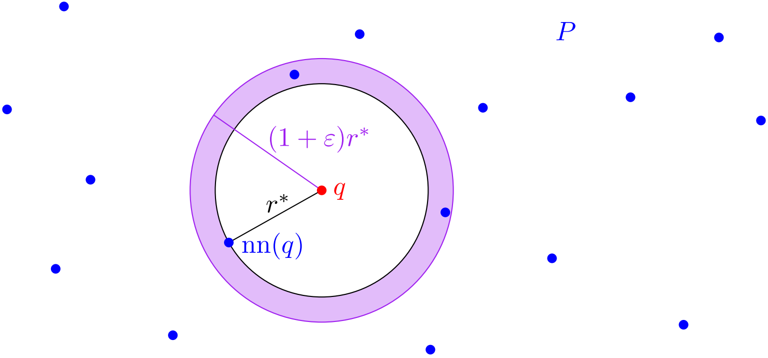
Point Location in a Quadtree

- Better approach: Do binary search on the level of the leaf node containing q.

- So we check whether there is a node u at level $\lfloor h/2 \rfloor$ such that $q \in \mathrm{box}(u)$.
- If it is not the case, recurse on the first $\lfloor h/2 \rfloor$ levels.
- If it is the case, and if this node is a leaf, return it.
- If it is an internal node, recurse on levels deeper than $\lfloor h/2 \rfloor$.

- This approach takes $O(\log h)$ time.

Approximate Nearest Neighbor Searching



- A $(1+\varepsilon)$-ANN query for q may return any of the points in the purple annulus.

Approximate Nearest Neighbor Searching

- P is still a set of n points in $\mathbb{R}^d$.
- Given a query point q, the distance from q to a closest point in P is denoted
$$r^* = \min_{p \in P} d(q,p).$$
- We denote by $\mathrm{nn}(q)$ a closest point $p \in P$ such that $d(q, \mathrm{nn}(q)) = r^*)$. This point p is also called a *nearest neighbor*.
- Let $0 < \varepsilon \leqslant 1$ be a relative error ratio.
- A $(1+\varepsilon)$-*approximate nearest neighbor* of q is a point $p \in P$ such that $d(q,p) \leqslant (1+\varepsilon)r^*$.
- We also write it $(1+\varepsilon)$-ANN.

Approximate Nearest Neighbor Searching

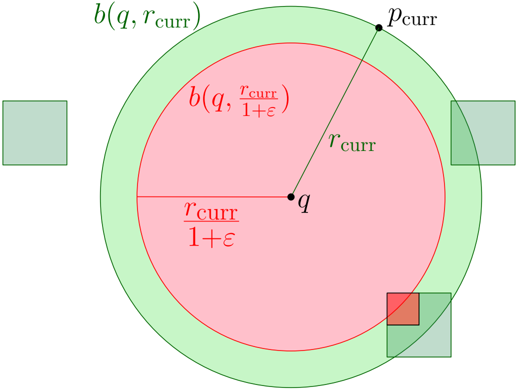
- Why should we use ANN searching?
- Alternative: exact near neighbor using Voronoi diagrams.
- Problem: in dimension d, the Voronoi diagram has combinatorial complexity $\Theta(n^{\lfloor d/2 \rfloor})$.
- In practice, we only compute it in dimension 2 or 3.
- But ANN queries can be answered efficiently using quadtrees in any fixed dimension:

Theorem

The quadtree described above allows us to answer $(1+\varepsilon)$-ANN queries in time $O(1/\varepsilon^d + \log \Phi)$.

- Remember we can compute this quadtree in $O(n \log \Phi)$ time.
- We will first describe the algorithm, and then analyze it.

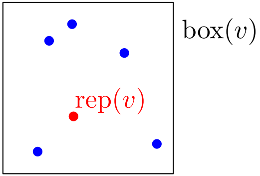
Algorithm

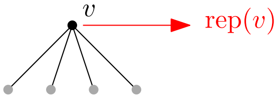


- Maintain a collection A_i of quadtree nodes at level i. (Green boxes.)
- Maintain the closest point found so far. (Point p_{curr}.)
- Recurse to the children of nodes in A_i whose box overlaps with the red ball. (Here, only the red box.)

Algorithm

- We denote by $\mathcal{T}$ the quadtree recording P.
- An *empty* node v of $\mathcal{T}$ is a node such that $\mathrm{box}(v) \cap P = \emptyset$, i.e. the corresponding box contains no input point.





- For each non-empty node v of $\mathcal{T}$, record a *representative point* $\mathrm{rep}(v) \in \mathrm{box}(v) \cap P$.
- We can compute these representative points in time $O(n \log \Phi)$ for the whole quadtree by a bottom-up construction.

Algorithm

Pseudocode

procedure ANN(quadtree $\mathcal{T}$, query point q, ε)
  $A_0 \leftarrow \{$root of $\mathcal{T}\}$, $p_{\mathrm{curr}} \leftarrow \mathrm{rep}$(root of $\mathcal{T}$), $i \leftarrow 0$
  while $A_i \neq \emptyset$ **do**
    $A_{i+1} \leftarrow \emptyset$
    for each node $u \in A_i$ **do**
      for each non-empty child v of u **do**
        if $d(q, \mathrm{rep}(v)) \leqslant r_{\mathrm{curr}}$ **then**
          $p_{\mathrm{curr}} \leftarrow \mathrm{rep}(v)$, $r_{\mathrm{curr}} \leftarrow d(q, \mathrm{rep}(v))$
    for each node $u \in A_i$ **do**
      for each non-empty child v of u **do**
        if $\mathrm{box}(v) \cap b(q, r_{\mathrm{curr}}/(1+\varepsilon)) \neq \emptyset$ **then**
          insert v into A_{i+1}
    $i \leftarrow i + 1$
  return p_{curr}

Proof of Correctness

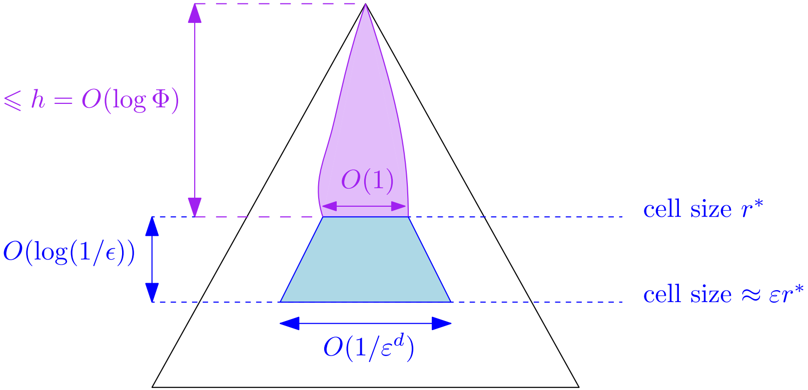
- During our traversal of $\mathcal{T}$, we discard a non-empty node v, and the subtree rooted at it, if $\mathrm{box}(v)$ does not overlap with the ball $b(q, r_{\mathrm{curr}}/(1+\varepsilon))$.
- Let w be the node discarded by the algorithm such that $\mathrm{nn}(q) \in \mathrm{box}(w)$.
- Let c be the point of $\mathrm{box}(w)$ that is closest to q.

$$\begin{aligned} r^* = d(q, \mathrm{nn}(q)) &\geqslant d(q,c) \\ &\geqslant \frac{r_{\mathrm{curr}}}{1+\varepsilon} \end{aligned}$$

  so p_{curr} is a $(1+\varepsilon)$-ANN.
- After that, p_{curr} may be updated, but it only gets closer to q, so it is still a $(1+\varepsilon)$-ANN.

Analysis

- Idea: At lower levels i, the set A_i contains $O(1)$ nodes.
- At later stages, it contains $O(1/\varepsilon^d)$ nodes in total.



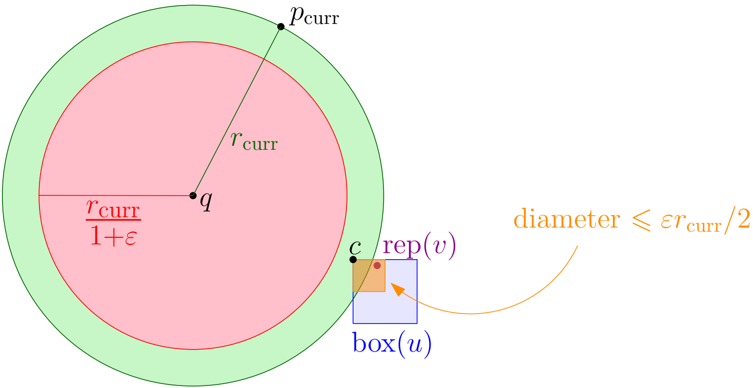
Analysis

Lemma

During the course of the algorithm, the size of the quadtree boxes of the nodes we visit is $\Omega(\varepsilon)$. More precisely, we have $\dfrac{1}{2^i} \geqslant \dfrac{\varepsilon r^*}{4\sqrt{d}}$.

- Suppose that, at line 9 and 12 of the algorithm, we visit a child v of a node $u \in A_i$.
- Suppose that $\mathrm{box}(v)$ has side length $1/2^{i+1} < \varepsilon r^* / (4\sqrt{d})$.
- Then $\mathrm{box}(u)$ has side length $1/2^i < \varepsilon r^* / (2\sqrt{d})$.
- As $r^* \leqslant r_{\mathrm{curr}}$, it implies $1/2^i \leqslant \varepsilon r_{\mathrm{curr}} / (2\sqrt{d})$.

Analysis



- Therefore the diameter $\sqrt{d}/2^i$ of $\text{box}(u)$ satisfies $\dfrac{\sqrt{d}}{2^i} \leqslant \dfrac{\varepsilon r_{\text{curr}}}{2}$.
- As v is visited at lines 5–8, we must have $d(q, \text{rep}(v)) \geqslant r_{\text{curr}}$.

Analysis

- Let c be the point of $\mathrm{box}(v)$ that is closes to q.

$$\begin{aligned} d(q,c) &\geqslant d(q,\mathrm{rep}(v)) - \mathsf{diam}(\mathrm{box}(v)) && \text{by triangle inequality} \\ &\geqslant r_{\mathrm{curr}} - \frac{\varepsilon r_{\mathrm{curr}}}{2} \\ &= \left(1 - \frac{\varepsilon}{2}\right) r_{\mathrm{curr}} \\ &\geqslant \frac{r_{\mathrm{curr}}}{1+\varepsilon} && \text{because } 0 < \varepsilon \leqslant 1 \end{aligned}$$

- It means that v will not be inserted in A_{i+1}.
- So the algorithm does not reach stage $i+1$, which completes the proof of the lemma.

Analysis

Lemma

For any node $v \in A_i$, we have $\mathrm{box}(v) \subset b(q, \ell_i)$ *where*

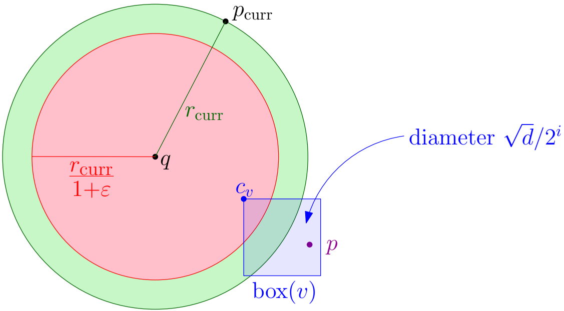
$$\ell_i = \frac{2\sqrt{d}}{2^i} + (1+\varepsilon)r^*.$$

- We now prove this lemma.
- There are two cases.

- **Case 1:** There exists $w \in A_i$ such that $\mathrm{nn}(q) \in \mathrm{box}(w)$.
- Since $\mathrm{diam}(\mathrm{box}(w)) = \sqrt{d}/2^i$, we have $d(\mathrm{nn}(q), \mathrm{rep}(w)) \leqslant \sqrt{d}/2^i$.
- It follows that

$$d(q, \mathrm{rep}(w)) \leqslant d(q, \mathrm{nn}(q)) + d(\mathrm{nn}(q), \mathrm{rep}(w)) \leqslant r^* + \frac{\sqrt{d}}{2^i}$$

Analysis



- So at Line 5 of the $i-1$th step, just before we construct A_i, we have

$$r_{\text{curr}} \leqslant d(q, \text{rep}(w)) \leqslant r^* + \frac{\sqrt{d}}{2^i}$$

- For any node $v \in A_i$, the the ball $b(q, r_{\text{curr}}/(1+\varepsilon))$ must intersect $\text{box}(v)$, so there is a point $c_v \in \text{box}(v)$ such that $d(q, c_v) \leqslant r_{\text{curr}}$.

Analysis

- Then any point $p \in \mathrm{box}(v)$ satisfies

$$\begin{aligned} d(q,p) &\leqslant d(q,c_v) + d(c_v,p) && \text{by the triangle inequality} \\ &\leqslant r_{\mathrm{curr}} + \sqrt{d}/2^i && \text{see previous slide} \\ &\leqslant r^* + 2\sqrt{d}/2^i \\ &\leqslant \ell_i \end{aligned}$$

- In other words, $\mathrm{box}(v) \subset b(q,\ell_i)$, which completes the proof of Case 1.

Analysis

- **Case 2:** For all $v \in A_i$, we have $\mathrm{nn}(q) \notin \mathrm{box}(v)$.
- So we have discarded the box containing $\mathrm{nn}(q)$, and thus by the argument on Slide 33, we have $r_{\mathrm{curr}} \leqslant (1+\varepsilon)r^*$.
- It follows that $\mathrm{box}(v) \subset b(q, (1+\varepsilon)r^* + \sqrt{d}/2^i) \subset b(q, \ell_i)$.
- This completes the proof of the lemma on Slide 38.

Analysis

Lemma

There are two constants α_d, β_d depending on d only such that

(a) *If $1/2^i > r^*$, then $|A_i| \leqslant \alpha_d$.*

(b) *If $1/2^i \leqslant r^*$, then $|A_i| \leqslant \beta_d (2^i r^*)^d$.*

- **Proof of (a)**: By the lemma on Slide 38, for any $v \in A_i$, we have $\mathrm{box}(v) \subset b(q, \ell_i)$ where $\ell_i = \frac{2\sqrt{d}}{2^i} + (1+\varepsilon) r^*$.
- As $\varepsilon < 1$ and $1/2^i > r^*$, we have $\ell_i < 2(1+\sqrt{d})/2^i$.
- The volume of $b(q, \ell_i)$ is $C_d \ell_i^d$ for some constant C_d depending on d only. The volume of $\mathrm{box}(v)$ for any $v \in A_i$ is $(1/2^i)^d$.
- As these boxes are disjoint, their number is at most

$$\frac{C_d \ell_i^d}{(1/2^i)^d} < C_d (2(1+\sqrt{d}))^d.$$

- So we can just choose $\alpha_d = C_d (2(1+\sqrt{d}))^d$.

Analysis

- **Proof of (b)**: By the lemma on Slide 38, for any $v \in A_i$, we have $\mathrm{box}(v) \subset b(q, \ell_i)$ where $\ell_i = \dfrac{2\sqrt{d}}{2^i} + (1+\varepsilon)r^*$.
- Since $1/2^i \leqslant r^*$ and $\varepsilon < 1$, we have $\ell_i < 2(\sqrt{d}+1)r^*$.
- So the volume of each $\mathrm{box}(v)$ where $v \in A_i$ is at most $(1/2^i)^d$.
- So the number of these boxes is at most

$$\frac{C_d(2(\sqrt{d}+1)r^*)^d}{(1/2^i)^d} = \beta_d(2^i r^*)^d$$

where β_d is a constant depending only on d.

Analysis

Lemma

Our algorithm returns a $(1+\varepsilon)$*-ANN in time* $O(1/\varepsilon^d + \log(1/r^*))$

- In the first part of the traversal, when $1/2^i > r^*$, we have $i < \log(1/r^*)$, so we traverse at most $\lceil \log(1/r^*) \rceil$ levels. By previous lemma (a), we visit $O(1)$ nodes per level, so it takes time $O(\log(1/r^*))$.
- In the second part of the traversal, by the lemma on Slide 35, we have

$$\frac{\varepsilon r^*}{4\sqrt{d}} \leqslant \frac{1}{2^i} \leqslant r^*$$

and at each level we visit at most $\beta_d (2^i r^*)^d$ nodes.

Analysis

- So the number of nodes we visit is at most $\sum_{i=0}^{m} \beta_d (r^*)^d (2^d)^i$ where $m = \lfloor \log(4\sqrt{d}/(\varepsilon r^*)) \rfloor$.
- This is the sum of a geometric series with common ratio 2^d, so it is dominated by its last term. Therefore, this sum is

$$O\left((r^*)^d (2^m)^d\right) = O\left((r^*)^d (4\sqrt{d}/(\varepsilon r^*))^d\right) = O(1/\varepsilon^d).$$

- So overall, the algorithm visits $O(1/\varepsilon^d + \log(1/r^*))$ nodes, which completes the proof of the lemma.

- We prove Theorem 29 by using the upper bound $i \leqslant h = O(\log \Phi)$ for the first part of the traversal.