CMSC 754: Lecture 16
Well-Separated Pair Decompositions

Reading: This material is not covered in our text. My presentation is taken from the book "Geometric Approximation Algorithms" by S. Har-Peled. The original paper on WSPDs is "A Decomposition of Multidimensional Point Sets with Applications to k-Nearest-Neighbors and n-Body Potential Fields," by P. B. Callahan and S. Rao Kosaraju, *J. ACM*, 42, 67–90, 1995.

Approximation Algorithms in Computational Geometry: Although we have seen many efficient techniques for solving fundamental problems in computational geometry, there are many problems for which the complexity of finding an exact solution is unacceptably high. Geometric approximation arises as a useful alternative in such cases. Approximations arise in a number of contexts. One is when solving a hard optimization problem. A famous example is the *Euclidean traveling salesman problem*, in which the objective is to find a minimum length path that visits each of n given points (see Fig. 1(a)). (This is an NP-hard problem, but there exists a polynomial time algorithm that achieves an approximation factor of $1 + \varepsilon$ for any $\varepsilon > 0$.) Another source arises when approximating geometric structures. For example, early this semester we mentioned that the convex hull of n points in $\mathbb{R}^d$ could have combinatorial complexity $\Omega(n^{\lfloor d/2 \rfloor})$. Rather than computing the exact convex hull, it may be satisfactory to compute a convex polytope, which has much lower complexity, and whose boundary is within a small distance ε from the actual hull (see Fig. 1(b)).

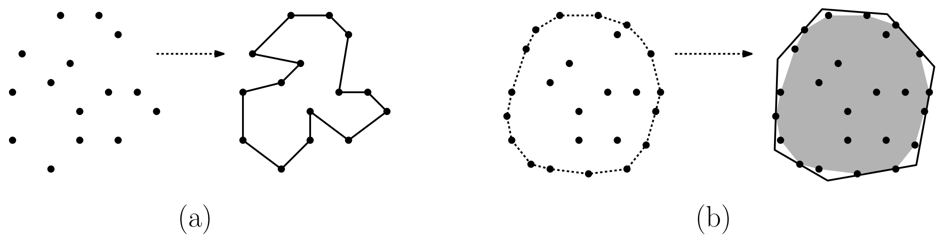

Fig. 1: Geometric approximations: (a) Euclidean traveling salesman, (b) approximate convex hull.

Another important motivations for geometric approximations is that geometric inputs are typically the results of sensed measurements, which are subject to limited precision. There is no good reason to solve a problem to a degree of accuracy that exceeds the precision of the inputs themselves.

Motivation: The n-Body Problem: We begin our discussion of approximation algorithms in geometry with a simple and powerful example. To motivate this example, consider an application in physics involving the simulation of the motions of a large collection of bodies (e.g., planets or stars) subject to their own mutual gravitational forces. In physics, such a simulation is often called the *n-body problem.* Exact analytical solutions are known to exist in only extremely small special cases. Even determining a good numerical solution is relative costly. In order to determine the motion of a single object in the simulation, we need to know the gravitational force induced by the other $n - 1$ bodies of the system. In order to compute this force, it would seem that at a minimum we would need $\Omega(n)$ computations per point, for a total of $\Omega(n^2)$ total computations. The question is whether there is a way to do this faster?

What we seek is a structure that allows us to encode the distance information of $\Omega(n^2)$ pairs in a structure of size only $O(n)$. While this may seem to be an impossible task, a clever approximate answer to this question was discovered by Greengard and Rokhlin in the mid 1980's, and forms the basis of a technique called the *fast multipole method*[1] (or FMM for short). We will not discuss the FMM, since it would take us out of the way, but will instead discuss the geometric structure that encodes much of the information that made the FMM such a popular technique.

Well Separated Pairs: A set of n points in space defines a set of $\binom{n}{2} = \Theta(n^2)$ distinct pairs. To see how to encode this set approximately, let us return briefly to the n-body problem. Suppose that we wish to determine the gravitational effect of a large number of stars in a one galaxy on the stars of distant galaxy. Assuming that the two galaxies are far enough away from each other relative to their respective sizes, the individual influences of the bodies in each galaxy can be aggregated into a single physical force. If there are n_1 and n_2 points in the respective galaxies, the interactions due to all $n_1 \cdot n_2$ pairs can be well approximated by a single *interaction pair* involving the centers of the two galaxies.

To make this more precise, assume that we are given an n-element point set P in $\mathbb{R}^d$, and a separation factor $s > 0$. We say that two disjoint sets of A and B are *s-well separated* if the sets A and B can be enclosed within two Euclidean balls of radius r such that the closest distance between these balls is at least sr (see Fig. 2).

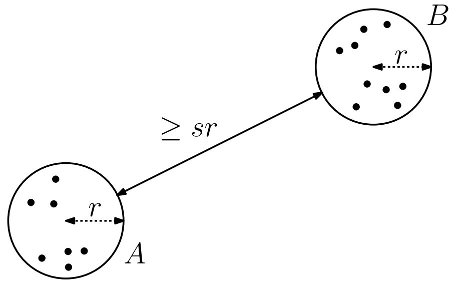

Fig. 2: A well separated pair with separation factor s.

Observe that if a pair of points is s-well separated, it is also s'-well separated for all $s' < s$. Of course, since any point lies within a (degenerate) ball of radius 0, it follows that a pair of singleton sets, $\{\{a\}, \{b\}\}$, for $a \neq b$, is well-separated for any $s > 0$.

Well Separated Pair Decomposition: Okay, distant galaxies are well separated, but if you were given an *arbitrary* set of n points in $\mathbb{R}^d$ (which may not be as nicely clustered as the stars in galaxies) and a fixed separation factor $s > 0$, can you concisely approximate all $\binom{n}{2}$ pairs? We will show that such a decomposition exists, and its size is $O(n)$. The decomposition is called a *well separated pair decomposition*. Of course, we would expect the complexity to depend on s and d as well. The constant factor hidden by the asymptotic notion grows as $O(s^d)$.

Let's make this more formal. Given arbitrary sets A and B, define $A \otimes B$ to be the set of all

[1]As an indication of how important this algorithm is, it was listed among the top-10 algorithms of the 20th century, along with quicksort, the fast Fourier transform, and the simplex algorithm for linear programming.

distinct (unordered) pairs from these sets, that is

$$A \otimes B \ = \ \{\{a,b\} \mid a \in A,\ b \in B,\ a \neq b\}.$$

Observe that $A \otimes A$ consists of all the $\binom{n}{2}$ distinct pairs of A. Given a point set P and separation factor $s > 0$, we define an *s-well separated pair decomposition* (s-WSPD) to be a collection of pairs of subsets of P, denoted $\{\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}\}$, such that

(1) $A_i, B_i \subseteq P$, for $1 \leq i \leq m$

(2) $A_i \cap B_i = \emptyset$, for $1 \leq i \leq m$

(3) $\bigcup_{i=1}^{m} A_i \otimes B_i = P \otimes P$

(4) A_i and B_i are s-well separated, for $1 \leq i \leq m$

Conditions (1)–(3) assert we have a cover of all the unordered pairs of P, and (4) asserts that the pairs are well separated. Although these conditions alone do not imply that every unordered pair from P occurs in a unique pair $A_i \otimes B_i$ (that is, the cover of $P \otimes P$ is actually a partition), our construction will have this further property. An example is shown in Fig. 3. (Although there appears to be some sort of hierarchical structure here, note that the pairs are not properly nested within one another.)

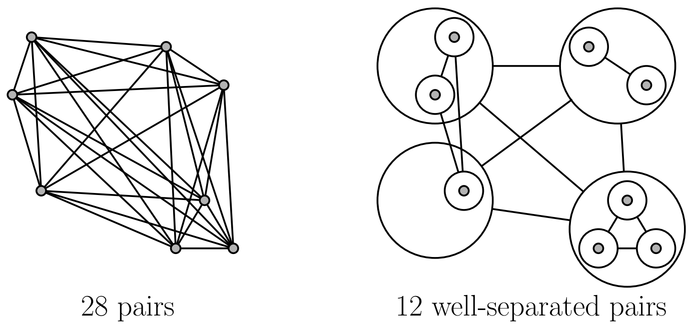

Fig. 3: A point set and a well separated pair decomposition for separation $s = 1$.

Trivially, there exists a WSPD of size $O(n^2)$ by setting the $\{A_i, B_i\}$ pairs to each of the distinct pair singletons of P. Our goal is to show that, given an n-element point set P in $\mathbb{R}^d$ and any $s > 0$, there exists a s-WSPD of size $O(n)$ (where the constant depends on s and d). Before doing this, we must make a brief digression to discuss the quadtree data structure, on which our construction is based.

Quadtrees: A *quadtree* is a hierarchical subdivision of space into regions, called *cells*, that are hypercubes. The decomposition begins by assuming that the points of P lie within a bounding hypercube. For simplicity we may assume that P has been scaled and translated so it lies within the unit hypercube $[0,1]^d$.

The initial cell, associated with the *root* of the tree, is the unit hypercube. The following process is then repeated recursively. Consider any unprocessed cell and its associated node u in the current tree. If this cell contains either zero or one point of P, then this is declared a leaf node of the quadtree, and the subdivision process terminates for this cell. Otherwise, the cell is subdivided into 2^d hypercubes whose side lengths are exactly half that of the original hypercube. For each of these 2^d cells we create a node of the tree, which is then made a child of u in the quadtree. (The process is illustrated in Fig. 4. The points are shown in Fig. 4(a), the node structure in Fig. 4(b), and the final tree in Fig. 4(c).) Quadtrees can be used to store various types of data. Formally, the structure we have just described in called a *PR-quadtree* (for "point-region quadtree").

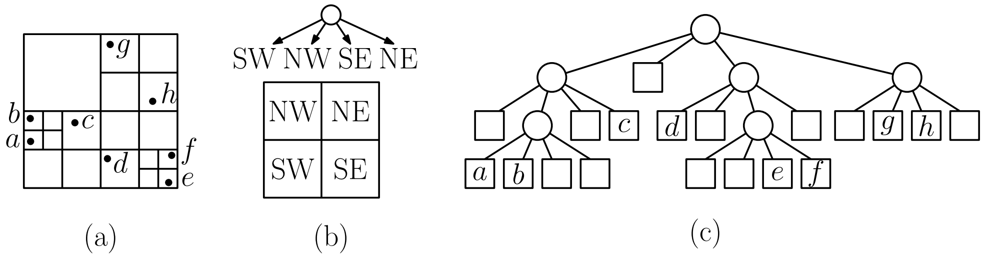

Fig. 4: The quadtree for a set of eight points.

Although in practice, quadtrees as described above tend to be reasonably efficient in fairly small dimensions, there are a number of important issues in their efficient implementation in the worst case. The first is that a quadtree containing n points may have many more than $O(n)$ nodes. The reason is that, if a group of points are extremely close to one another relative to their surroundings, there may be an arbitrarily long *trivial path* in the tree leading to this cluster, in which only one of the 2^d children of each node is an internal node (see Fig. 5(a)).

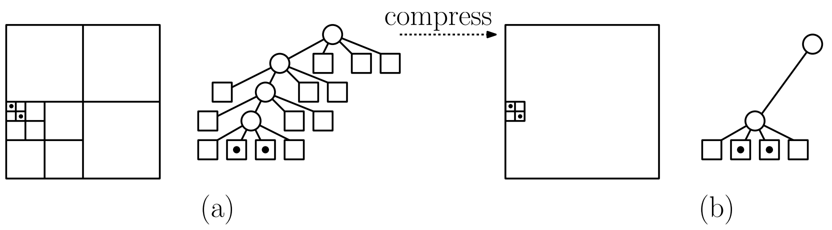

Fig. 5: Compressed quadtree: (a) The original quadtree, (b) after path compression.

This issue is easily remedied by a process called *path compression.* Every such trivial path is compressed into a single link. This link is labeled with the coordinates of the smallest quadtree box that contains the cluster (see Fig. 5(b)). The resulting data structure is called a *compressed quadtree.* Observe that each internal node of the resulting tree separates at least two points into separate subtrees. Thus, there can be no more than $n-1$ internal nodes, and hence the total number of nodes is $O(n)$.

A second issue involves the efficient computation of the quadtree. It is well known that the tree can be computed in time $O(hn)$, where h is the height of the tree. However, even for a compressed quadtree the tree height can be as high as n, which would imply an $O(n^2)$ construction time. We will not discuss it here, but it can be shown that in any fixed dimension it is possible to construct the quadtree of an n-element point set in $O(n \log n)$ time. (The key is handling uneven splits efficiently. Such splits arise when one child contains almost all of the points, and all the others contain only a small constant number.)

The key facts that we will use about quadtrees below are:

(a) Given an n-element point set P in a space of fixed dimension d, a compressed quadtree for P of size $O(n)$ can be constructed in $O(n \log n)$ time.

(b) Each internal node has a constant number (2^d) children.

(c) The cell associated with each node of the quadtree is a d-dimensional hypercube, and as we descend from the parent to a child (in the uncompressed quadtree), the size (side length) of the cells decreases by a factor of 2.

(d) The cells associated with any level of the tree (where tree levels are interpreted relative to the uncompressed tree) are of the same size and all have pairwise disjoint interiors.

An important consequence stemming from (c) and (d) is the following lemma, which provides an upper bound on the number of quadtree disjoint quadtree cells of size at least x that can overlap a ball of radius r.

Packing Lemma: Consider a ball b of radius r in any fixed dimension d, and consider any collection X of pairwise disjoint quadtree cells of side lengths at least x that overlap b. Then

$$|X| \;\le\; \left(1 + \left\lceil \frac{2r}{x} \right\rceil\right)^d \;\le\; O\left(\max\left(2,\; \frac{r}{x}\right)^d\right)$$

Proof: We may assume that all the cells of X are of side length exactly equal to x, since making cells larger only reduces the number of overlapping cells (see Fig. 6(b)).

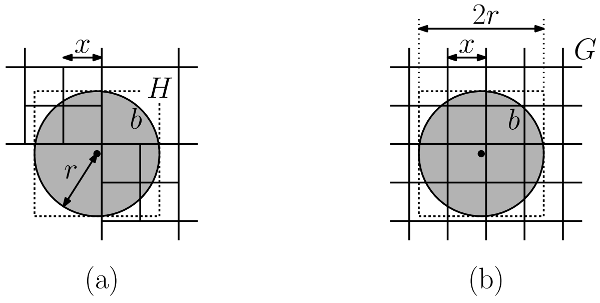

Fig. 6: Proof of the Packing Lemma.

By the nature of a quadtree decomposition, the cells of side length x form a hypercube grid G of side length x. Consider a hypercube H of side length $2r$ that encloses b (see Fig. 6). Clearly every cell of X overlaps this hypercube. Along each dimension, the number of cells of G that can overlap an interval of side length $2r$ is at most $1 + \lceil 2r/x \rceil$. Thus, the number of grid cubes of G that overlap H is at most $(1 + \lceil 2r/x \rceil)^d$. If $2r < x$, this quantity is at most 2^d, and otherwise it is $O((r/x)^d)$.

For the construction of the WSPD, we need to make a small augmentation to the quadtree structure. We wish to associate each node of the tree, both leaves and internal nodes, with a point that lies within its cell (if such a point exists). Given a node u, we will call this point u's *representative* and denote this as $\text{rep}(u)$. We do this recursively as follows. If u is a leaf node that contains a point p, then $\mathrm{rep}(u) = \{p\}$. If u is a leaf node that contains no point, then $\mathrm{rep}(u) = \emptyset$. Otherwise, if u is an internal node, then it must have at least one child v that is not an empty leaf. (If there are multiple nonempty children, we may select any one.) Set $\mathrm{rep}(u) = \mathrm{rep}(v)$.

Given a node u in the tree, let P_u denote the points that lie within the subtree rooted at u. We will assume that each node u is associated with its *level* in the tree, denoted $\mathrm{level}(u)$. Assuming that the original point set lies within a unit hypercube, the side lengths of the cells are of the form $1/2^i$, for $i \geq 0$. We define $\mathrm{level}(u)$ to be $-\log_2 x$, where x is the side length of u's cell. Thus, $\mathrm{level}(u)$ is just the depth of u in the (uncompressed) quadtree, where the root has depth 0. The key feature of level is that $\mathrm{level}(u) \leq \mathrm{level}(v)$ holds if and only if the sidelength of u's cell at least as large as that of v's cell.

We will treat leaf nodes differently from internal nodes. If a leaf node u contains no point at all, then we may ignore it, since it cannot participate in any well-separated pair. If it does contain a point, then we think of the leaf node conceptually as an infinitesimally small quadtree cell that contains this point. We do this by defining $\mathrm{level}(u) = +\infty$ for such a node. We will see later why this is useful.

Constructing a WSPD: We now have the tools needed to to show that, given an n-element point set P in $\mathbb{R}^d$ and any $s > 0$, there exists a s-WSPD of size $O(s^d n)$, and furthermore, this WSPD can be computed in time that is roughly proportional to its size. In particular, the construction will take $O(n \log n + s^d n)$ time. We will show that the final WSPD can be encoded in $O(s^d n)$ total space. Under the assumption that s and d are fixed (independent of n) then the space is $O(n)$ and the construction time is $O(n \log n)$.

The construction operates as follows. Recall the conditions (1)–(4) given above for a WSPD. We will maintain a collection of sets that satisfy properties (1) and (3), but in general they may violate conditions (2) and (4), since they may not be disjoint and may not be well separated. When the algorithm terminates, all the pairs will be well-separated, and this will imply that they are disjoint. Each set $\{A_i, B_i\}$ of the pair decomposition will be encoded as a pair of nodes $\{u, v\}$ in the quadtree. Implicitly, this pair represents the pairs $P_u \otimes P_v$, that is, the set of pairs generated from all the points descended from u and all the points descended from v. This is particularly nice, because it implies that the total storage requirement is proportional to the number of pairs in the decomposition.

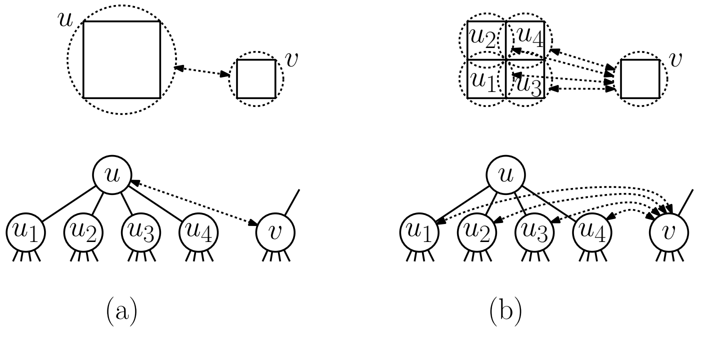

Fig. 7: WSPD recursive decomposition step.

The algorithm is based on a recursive subdivision process. Consider a pair of nodes $\{u, v\}$ that arise in the decomposition process. If either of the nodes is an empty leaf, then we may ignore this pair. If both of the nodes are leaves, then they are clearly well-separated (irrespective of the value of s), and we may output this pair. Otherwise, let us assume that u's cell is least as large as v's. That is, u's level number is not greater than v's. (Recall that a leaf node is treated as an infinitesimally small quadtree cell that contains the node's point, and its level is defined to be $+\infty$. So if an internal node and a leaf node are compared, the internal node is always deemed to have the larger cell.) Consider the two smallest Euclidean balls of equal radius that enclose u's cell and v's cell (see Fig. 7(a)). If these balls are well separated, then we can report $\{u, v\}$ as (the encoding of) a well separated pair. Otherwise, we subdivide u by considering its children, and apply the procedure recursively to the pairs $\{u_i, v\}$, for each child of u_i of u (see Fig. 7(b)).

A more formal presentation of the algorithm is presented in the following code block. The procedure is called ws-pairs(u, v, s), where u and v are the current nodes of a compressed quadtree for the point set, and s is the separation factor. The procedure returns a set node pairs, encoding the well separated pairs of the WSPD. The initial call is ws-pairs(u_0, u_0, s), where u_0 is the root of the compressed quadtree.

Construction of a Well Separated Pair Decomposition

```
ws-pairs($u, v, s$) {
    if ($u$ and $v$ are leaves and $u = v$) return;
    if (rep($u$) or rep($v$) is empty) return $\emptyset$;   // no pairs to report
    else if ($u$ and $v$ are $s$-well separated)       // (see remark below)
        return $\{\{u, v\}\}$;                          // return the WSP $\{P_u, P_v\}$
    else {                                              // subdivide
        if (level($u$) > level($v$)) swap $u$ and $v$;// swap so that $u$'s cell is at least as large as $v$'s
        Let $u_1, \ldots, u_m$ denote the children of $u$;
        return $\bigcup_{i=1}^{m}$ ws-pairs($u_i, v, s$);          // recurse on children
    }
}
```

How do we test whether two nodes u and v are s well separated? For each internal node, consider the smallest Euclidean balls enclosing the associated quadtree cells. For each leaf node, consider a degenerate ball of radius zero that contains the point. In $O(1)$ time, we can determine whether these balls are s well separated. Note that a pair of leaf cells will always pass this test (since the radius is zero), so the algorithm will eventually terminate.

Remark: Due to its symmetry, this procedure will generally produce duplicate pairs $\{P_u, P_v\}$ and $\{P_v, P_u\}$. A simple disambiguation rule can be applied to eliminate this issue.

Analysis: How many pairs are generated by this recursive procedure? It will simplify our proof to assume that the quadtree is not compressed (and yet it has size $O(n)$). This allows us to assume that the children of each node all have cell sizes that are exactly half the size of their parent's cell. (We leave the general case as an exercise.)

From this assumption, it follows that whenever a call is made to the procedure ws-pairs(), the sizes of the cells of the two nodes u and v differ by at most a factor of two (because we always split the larger of the two cells). It will also simplify the proof to assume that $s \geq 1$ (if not, replace all occurrences of s below with $\max(s, 1)$).

To evaluate the number of well separated pairs, we will count calls to the procedure ws-pairs(). We say that a call to ws-pairs is *terminal* if it does not make it to the final "else" clause. Each terminal call generates at most one new well separated pair, and so it suffices to count the number of terminal calls to ws-pairs. In order to do this, we will instead bound the number of nonterminal calls. Each nonterminal call generates at most 2^d recursive calls (and this is the only way that terminal calls may arise). Thus, the total number of well separated pairs is at most 2^d times the number of nonterminal calls to ws-pairs.

To count the number of nonterminal calls to ws-pairs, we will apply a charging argument to the nodes of the compressed quadtree. Each time we make it to the final "else" clause and split the cell u, we assign a charge to the "unsplit" cell v. Recall that u is generally the larger of the two, and thus the smaller node receives the charge. We assert that the total number of charges assigned to any node v is $O(s^d)$. Because there are $O(n)$ nodes in the quadtree, the total number of nonterminal calls will be $O(s^d n)$, as desired. Thus, to complete the proof, it suffices to establish this assertion about the charging scheme.

A charge is assessed to node v only if the call is nonterminal, which implies that u and v are not s-well separated. Let x denote the side length of v's cell and let $r_v = x\sqrt{d}/2$ denote the radius of the ball enclosing this cell. As mentioned earlier, because we are dealing with an uncompressed quadtree, and the construction always splits the larger cell first, we may assume that u's cell has a side length of either x or $2x$. Therefore, the ball enclosing u's cell is of radius $r_u \leq 2r_v$. Since u and v are not well separated, it follows that the distance between their enclosing balls is at most $s \cdot \max(r_u, r_v) \leq 2sr_v = sx\sqrt{d}$. The centers of their enclosing balls are therefore within distance

$$r_v + r_u + sx\sqrt{d} \;\leq\; \left(\frac{1}{2} + 1 + s\right) x\sqrt{d} \;\leq\; 3sx\sqrt{d} \qquad \text{(since } s \geq 1\text{)},$$

which we denote by R_v (see Fig. 8(a)).

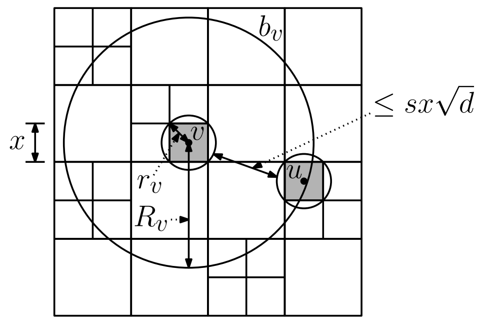

Fig. 8: WSPD analysis.

Let b_v be a Euclidean ball centered at v's cell of radius R_v. Summarizing the above discussion, we know that the set of quadtree nodes u that can assess a charge to v have cell sizes of either x or $2x$ and overlap b_v. Clearly the cells of side length x are disjoint from one another and the cells of side length $2x$ are disjoint from one another. Thus, by the Packing Lemma, the total number of nodes that can assess a charge to node v is at most C, where

$$\begin{aligned} C &\leq \left(1 + \left\lceil \frac{2R_v}{x} \right\rceil\right)^d + \left(1 + \left\lceil \frac{2R_v}{2x} \right\rceil\right)^d \;\leq\; 2\left(1 + \left\lceil \frac{2R_v}{x} \right\rceil\right)^d \\ &\leq 2\left(1 + \left\lceil \frac{6sx\sqrt{d}}{x} \right\rceil\right)^d \;\leq\; 2(2 + 6s\sqrt{d})^d. \end{aligned}$$

(In the last inequality, we used the fact that $\lceil z \rceil \leq 1 + z$.) Since the dimension d is assumed to be a constant and $s \geq 1$, this is $O(s^d)$.

Putting this all together, we recall that there are $O(n)$ nodes in the compressed quadtree and $O(s^d)$ charges assigned to any node of the tree, which implies that there are a total of $O(s^d n)$ total nonterminal calls to ws-pairs. As observed earlier, the total number of well separated pairs is larger by a factor of $O(2^d)$, which is just $O(1)$ since d is a constant. Together with the $O(n \log n)$ time to build the quadtree, this gives an overall running time of $O((n \log n) + s^d n)$ and $O(s^d n)$ total well separated pairs. In summary we have the following result.

Theorem: Given a point set P in $\mathbb{R}^d$, and a fixed separation factor $s \geq 1$, in $O(n \log n + s^d n)$ time it is possible to build an s-WSPD for P consisting of $O(s^d n)$ pairs.

As mentioned earlier, if $0 < s < 1$, then replace s with $\max(s, 1)$. Next time we will consider applications of WSPDs to solving a number of geometric approximation problems.