Notes by Steve Gribble

- N-body continued: Callahan & Kosaraju Algorithm
- Fast Multipole Method (Review)
  1. The Four Ideas
  2. Interaction Lists
- Callahan and Kosaraju's Algorithm
  1. Well Separated Pair Decompositions
  2. Definitions
  3. Algorithm: Tree Decomposition
  4. Algorithm: Well-Separated Realization
  5. Bounding the size of the realization
- N-body Summary

1 Fast Multipole Method (Review)

1.1 The Four Ideas

The four main ideas being the fast multipole method (FMM) were:

- **Multipole expansion** - a Taylor-series like expansion of the potential due to a cluster of particles around an origin. The expansion converges at points outside the cluster so that the potential can be approximated by only keeping the first p terms. We call this the outer or multipole expansion.

- **Error bounds on dropping terms** - the number of terms in the expansion dictates the accuracy of the expansion

- **Dual local expansion** - an expansion of the potential due to particles outside a cluster which converges for points inside the cluster. We call this the inner or local expansion.

- **Translation of expansions** - being able to translate a inner or outer expansion to a new origin, or translate an outer to an inner expansion at a different origin.

A fifth idea (Figure 97) is to use a tree-structure to organize the particles and to determine a strategy for grouping the particles and performing the outer/outer, outer/inner, and inner/inner translations. We call each node of the tree a *cell*.

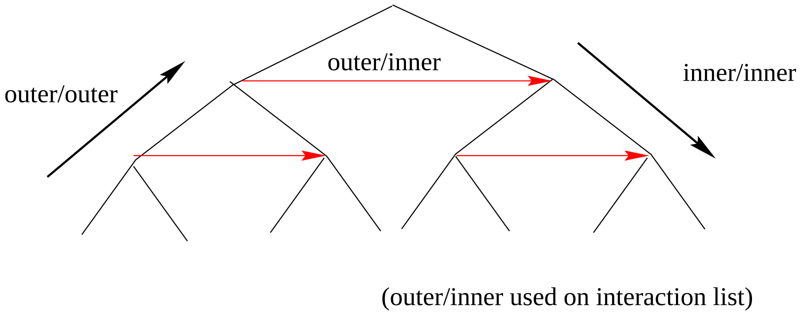

Figure 97: Fifth idea - using a tree for translations

1.2 Interaction Lists

In the fast multipole method, the interaction list (Figure 98) dictates the particles that a given cell will interact with (cross edges in Figure 97). The *interaction list* for a cell c consists of all the cells at the same level of the tree which (1) are well-separated from c and (2) are children of cells that are not well-separated from the parent of c. In Figure 98, for example, we assume a cell is well separated if it is 2 cells away. Clearly all the outer shaded cells are 2 away from the center cell c and are therefore well-separated. Also the parents of these cells are not well-separated from the parent of c (note that parent cells are marked with the thicker lines). One issue is how to find particles in the interaction list. Here are two possible methods:

1. **Uniform distributions:** a hierarchy of meshes can be used in this situation. Simply look at neighbors at an appropriate resolution in the mesh hierarchy.

2. **Non-uniform distributions:** a mesh hierarchy may be very sparsely populated, so a Barnes-Hut tree-like structure may be more appropriate in this case.

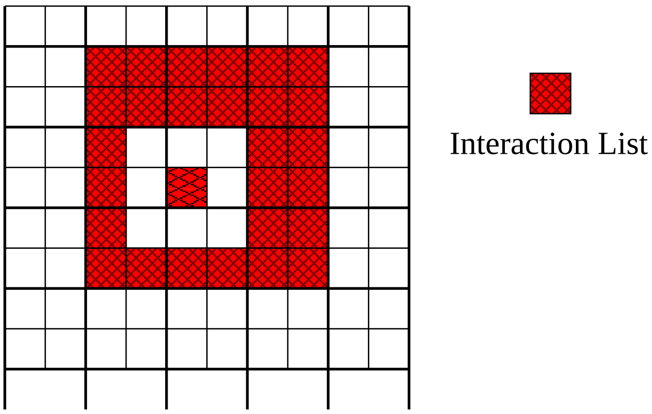

Figure 98: The Interaction List

2 Callahan and Kosaraju's Algorithm

We now discuss an algorithm due to Callahan and Kosaraju for generating a tree and interactions-lists for non-uniform distributions ("A Decomposition of Multi-Dimensional Point-Sets with Applications to k-Nearest-Neighbors and n-Body Potential Fields.", JACM, 42(1), January 1995). The algorithm has the nice property that it is very simple, although the proof of its bounds is a little involved. The particular tree-structure it creates is called a well-separated pair decomposition. It uses a somewhat more general definition of interaction lists than used in the previous section (for example, it allows interactions between cells on different levels). The algorithm works in any dimension, requires at most $O(n \log n)$ time to build the tree for n particles and guarantees that there will be a total of $O(n)$ interactions (size of the interaction lists summed over all cells).

2.1 Well Separated Pair Decompositions

The tree structure used by the algorithm is based on the notion of a tree decomposition of points. Given the set of points {a, b, c, d, e, f, g}, a *tree decomposition* of these points repeatedly splits this set into subsets, as in Figure 99.

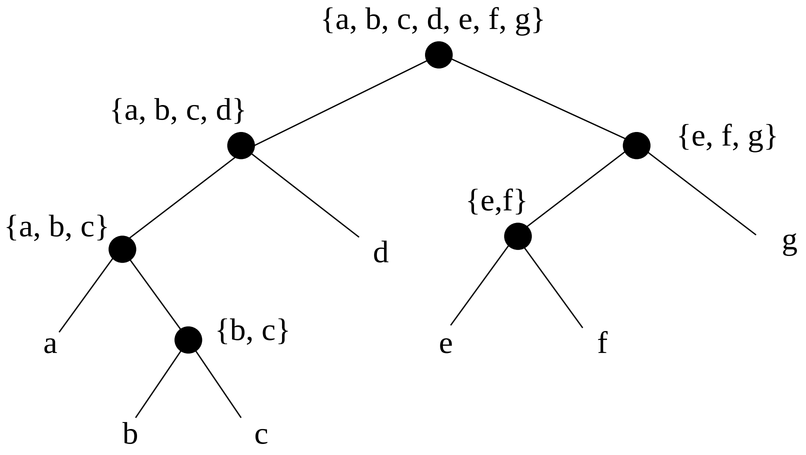

Figure 99: A tree decomposition D of points.

A *well-separated realization* of a tree decomposition is a set of interaction edges between vertices (cells) in the tree (edges represent outer/inner translations) such that (1) the cells at the two ends of the edge are well-separated (we will define well-separated later in these notes) and (2) there is exactly one path between any two leaves that goes up the tree any number of levels (including 0), across a single interaction-edge, and back down the tree any number of levels. The second property guarantees that each particle will interact exactly once with every other particle. Figure 100 shows an example of a well-separated realization for the tree decomposition given in Figure 99. You can verify that there is exactly one path between every leaf. We will define well-separated realizations more formally in Section 2.2.

Together the tree-decomposition and the well-separated realization on that tree are called a *well separated pair decomposition*. The Callahan-Kosaraju algorithm consists of generating

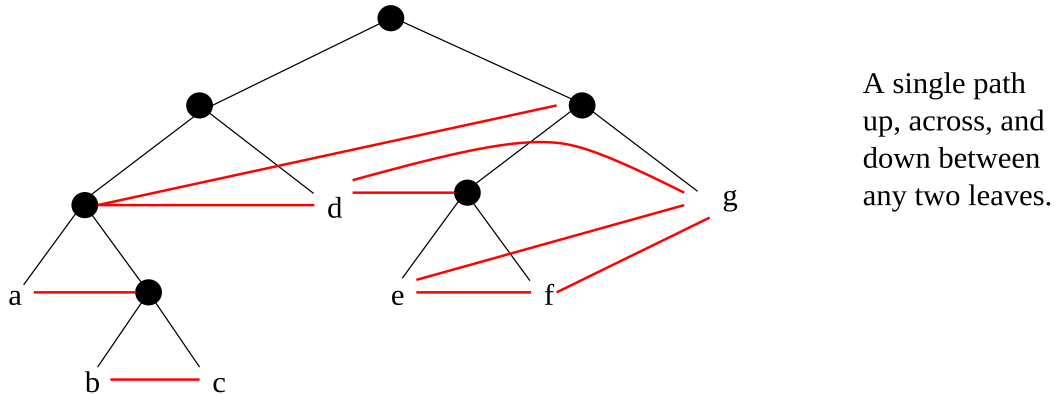

Figure 100: A well-separated "realization" of D.

the tree and the realization in two steps. We will see that we can build the tree-decomposition in $O(n \log n)$ time and find the well-separated realization given the tree in $O(n)$ time. Furthermore we can bound the size of the well-separated realization (total number of interaction edges) by $O(n)$.

The well-separated pair decomposition has applications both to the N-body problem and to computational geometry. For the N-body problem the tree is used for outer/outer translations (going up the tree) and inner/inner translations (going down the tree), and the well-separated realization is used for outer/inner translations. In computational geometry the well-separated pair decomposition can be used to find the k-nearest-neighbors for each point. This requires some augmentation to the tree and is discussed in the Callahan and Kosaraju paper.

For the N-body problem the time taken by one step of building the tree, finding the interaction edges, and doing all the translations to calculate the forces is $k_1 n \log n + k_2 n$, where the first term is for building the tree and the second linear term is for finding the interaction edges and translating the expansions. In practice the constant k_2 is very much larger than k_1. Furthermore in many cases it is possible to use the same tree (or only slightly modified tree) for multiple steps. Therefore from a practical standpoint the algorithm runs in almost linear time.

2.2 Definitions

- **Bounding Rectangle** $R(P)$: the smallest rectangle that contains points P and is aligned with the axes (Figure 101).

- $l_{\max}$: maximum length over the dimensions of a rectangle. $l_{\max}(P)$ is the maximum length of $R(P)$.

- **Well-separated**: Given two rectangles, let r be the smallest radius that contains them both (see Figure 102). For a given *separation constant* s two rectangles are well separated if $d > sr$.

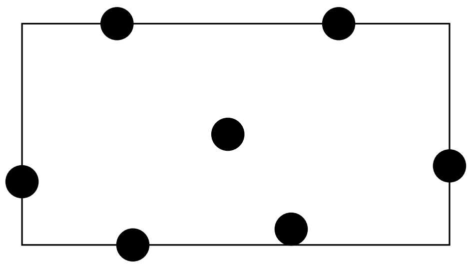

Figure 101: Bounding Rectangle

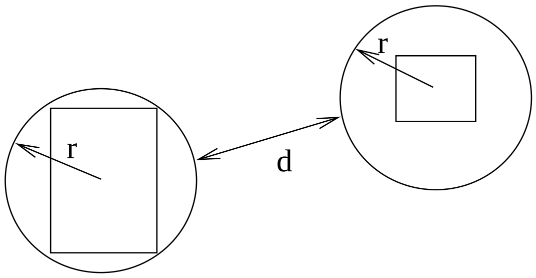

Figure 102: Well Separated

- **Interaction Product** $A \otimes B$:

$$A \otimes B = \{\{p, p'\} | p \in A, p' \in B, p \neq p'\}$$

- **Realization of** $A \otimes B$ is a set $\{\{A_1, B_1\}, \ldots \{A_k, B_k\}\}$ such that:

  1. $A_i \subseteq A, B_i \subseteq B$ for all $i = 1 \ldots k$
  2. $A_i \cap B_i = \emptyset$
  3. $(A_i \otimes B_i) \cap (A_j \otimes B_j) = \emptyset$ for all $i \neq j$, *i.e.*, there are no repeated interactions
  4. $A \otimes B = \cup_{i=1}^{k} A_i \otimes B_i$, *i.e.*, all interactions are included.

  This formalizes the interaction edges in the tree and guarantees that there is only one path between two leaves. In particular rule 2 guarantees that no leaf interacts with itself, rule 3 guarantees that no leaf interacts more than once with another leaf and rule 4 guarantees that every leaf interacts with every other leaf.

- **A well separated realization** is a realization such that for all i, A_i and B_i are well separated.

- **A well-separated pair decomposition** is a tree decomposition plus a well-separated realization $P \otimes P$ where subsets are nodes of the tree (Figure 103).

2.3 Algorithm: Tree Decomposition

The algorithm partitions space by recursively cutting bounding rectangles in half along their longest dimension. It can be defined as follows:

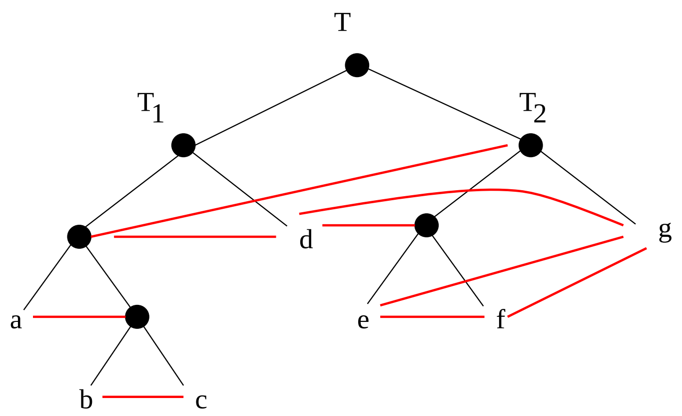

Figure 103: Subsets are nodes of the tree

function Build_Tree(P)
    // P is a set of points
    if $|P| = 1$ **then return** Leaf(P)
    else
        S = plane that splits $R(P)$ perpendicular to its longest dimension
        P_1, P_2 = Split P by the plane S
        return Tree_Node(Build_Tree(P_1), Build_Tree(P_2), $R(P)$)
end

An example of the algorithm on a set of 8 points is given in (Figure 104). The algorithm

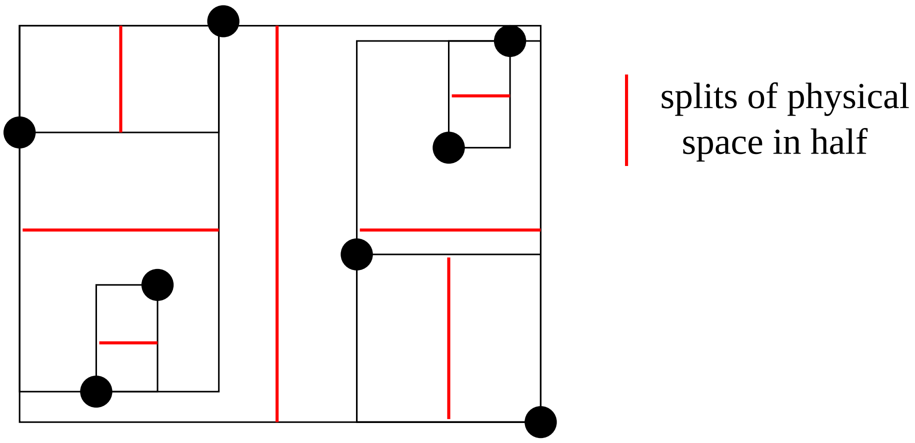

Figure 104: Building the tree

returns a tree with the bounding rectangle stored at each internal node and the original points at the leaves. The bounding rectangles are needed for building the well-separated realization (*e.g.* to determine when nodes are well separated).

For uniformly distributed sets of points the Build_Tree algorithm will run in $O(n \log n)$ time without any special tricks since finding the bounding rectangle and splitting P take $O(n)$ time and the two recursive calls will be on sets of approximately the same size (*i.e.*, we have a standard $T(n) = 2T(n/2) + O(n)$ style recurrence). Furthermore the algorithm is highly parallel since the two recursive calls can be made in parallel and the data can be split in parallel In fact for most "practical" sets of points this will be true.

If the cut does not evenly divide the points, however, the naive algorithm could take $O(n^2)$ time. For example, if there is a sequence of points that are exponentially approaching a fixed point. In this case the algorithm can be slightly modified so that it still runs in $O(n \log n)$ time. This modified version is outlined as follows.

1. Keep the points in linked lists sorted by each dimension
2. In the selected dimension, come in from both sides until the cut is found
3. Remove cut elements and put aside
4. Repeat making cuts until the size of the largest subset is less than $(2/3)n$
5. Create subsets and make recursive calls.

The runtime for building the tree is thus specified by

$$T(n) = \sum_{i=1}^{k} T(n_i) + O(n)$$

such that $\sum_{i=1}^{k} n_i = n$ and $\max_{i=1}^{k}(n_i) = \frac{2}{3}n$.

This solves to $O(n \log n)$ independent of how inbalanced the tree ends up being. A problem is that it does not parallelize easily. Callahan and Kosaraju show how it can be parallelized, but the parallel version is even more complicated and is not work-efficient (the processor-time product is $O(n \log^2 n)$). We will not cover this complicated parallel version since in practice parallelizing the simple version is both more efficient and likely to be highly effective.

2.4 Algorithm: Well-Separated Realization

Once we have a decomposition tree we can use it to build the well-separated realization. The following algorithm returns the set of interaction edges in the well-separated realization. The basic idea is to first generate interaction edges within the left and right children (recursive calls to WSR) and then generate interaction edges between the two children (call to WSR_Pair).

```
function WSR(T)
  // T is a decomposition tree
  if Is_Leaf(T) return ∅
  else
    return WSR(Left(T)) ∪
           WSR(Right(T)) ∪
           WSR_Pair(Left(T),Right(T))
end
```

function WSR_Pair(T_1, T_2)
    // T_1 and T_2 are two disjoint decomposition trees
    if Well_Separated(T_1, T_2) **return** {Edge(T_1, T_2)}
    else
        if ($l_{\max}(T_1) > l_{\max}(T_2)$)
            return WSR_Pair(Left(T_1), T_2) $\cup$ WSR_Pair(Right(T_1), T_2)
        else
            return WSR_Pair(T_1, Left(T_2)) $\cup$ WSR_Pair(T_1, Right(T_2))
end

The idea of WSR_Pair(T_1, T_2) is to return an interaction edge if T_1 and T_2 are well separated. Otherwise it picks the node with a larger $l_{\max}$ and splits that node and calls itself recursively on the smaller and the two halves of the larger.

Figure 105 gives an example of the algorithm along with call tree for WSR_Pair(T_1, T_2). In the example the algorithm determines that $l_{\max}(T_2) > l_{\max}(T_1)$ and therefore splits T_2 in the recursive calls. At the next level it splits T_1, at which point it determines that T_{11} is well separated from T_{21} and therefore makes no more recursive calls on that branch.

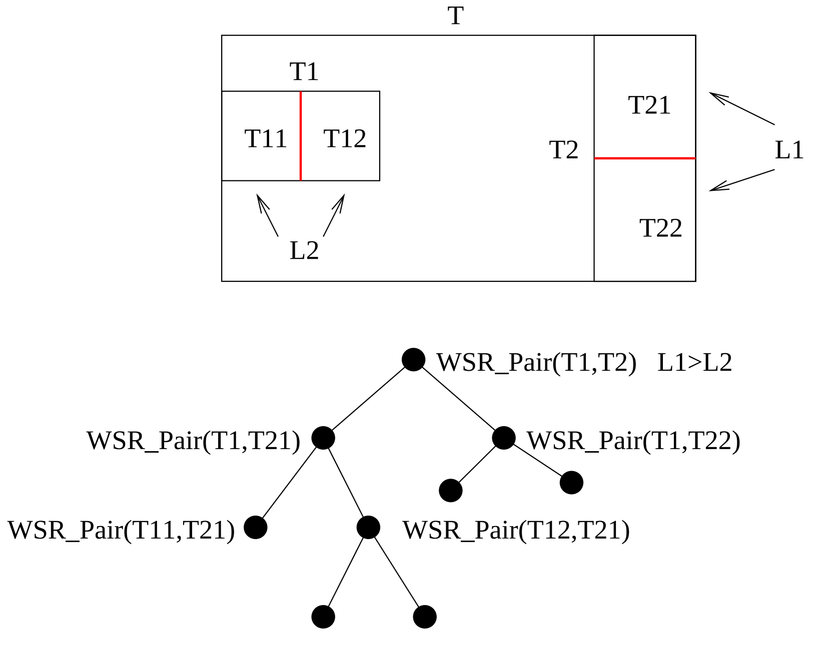

Figure 105: Generating the Realization. The tree represents the call tree for WSR_Pair.

2.5 Bounding the size of the realization

By bounding the size of the realization we both bound the number of interaction edges and also bound the runtime for building the realization since the runtime of WSR is proportional to the number of edges it creates.

We will use the following to bound the total number of interaction edges generated by WSR (size of realization):

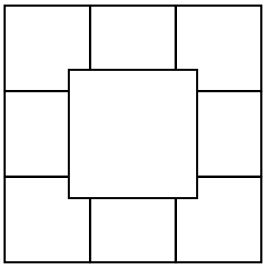

Figure 106: Non-intersecting rectangles

- Properties of the tree nodes - show bounding rectangles are not "too thin"
- Bound # of non-intersecting rectangles that can touch a cube of fixed size (Figure 106). This assumes the rectangles are not too thin.
- Bound # of calls to WSR_Pair(T_1, T_2) for each cell T_1 or T_2.

We define an *outer rectangle* for each bounding rectangle such that these outer rectangles fill space (*i.e.*, the union of all rectangles at a given level of a tree fill the initial space). The outer rectangle for each node in the decomposition tree is defined as follows and illustrated in Figure 107.

$$\hat{R}(P) = \begin{cases} \text{Smallest enclosing cube} & \text{at root} \\ & \\ \text{Use hyperplane that} & \\ \text{divides } R(P) \text{ to} & \text{internally} \\ \text{divide } \hat{R}(P) & \end{cases}$$

Lemma 1 $l_{\min}(\hat{R}(A)) \geq \frac{l_{\max}(p(A))}{2}$, *where* $p(A) =$ *parent of* A *in the tree.*

Proof: By induction.
Base: true when $p(A) =$ root, since $l_{\min}(\hat{R}(A)) = \frac{l_{\max}(p(A))}{2}$.
Induction: if true for $p(A)$ then true for A,

Case 1: $l_{\min}(\hat{R}(A)) = l_{\min}(\hat{R}(p(A)))$. True, since $l_{\max}(p(A)) \leq l_{\max}(p(p(A)))$ and by induction $l_{\min}(\hat{R}(p(A))) = \frac{l_{\max}(p(p(A)))}{2}$.

Case 2: $l_{\min}(\hat{R}(A)) < l_{\min}(\hat{R}(p(A)))$. First we note that $l_{\max}(p(A))$ and $l_{\min}(\hat{R}(A))$ must be along the same dimension since Build_Tree cut perpendicular to $l_{\max}(p(A))$ and if $l_{\min}(\hat{R}(A))$ has decreased it must have been cut. Since $\hat{R}(p(A))$ contains $R(p(A))$ cutting $R(p(A))$ in half will mean that $l_{\min}(\hat{R}(A)) \geq \frac{l_{\max}(p(A))}{2}$.

$\square$

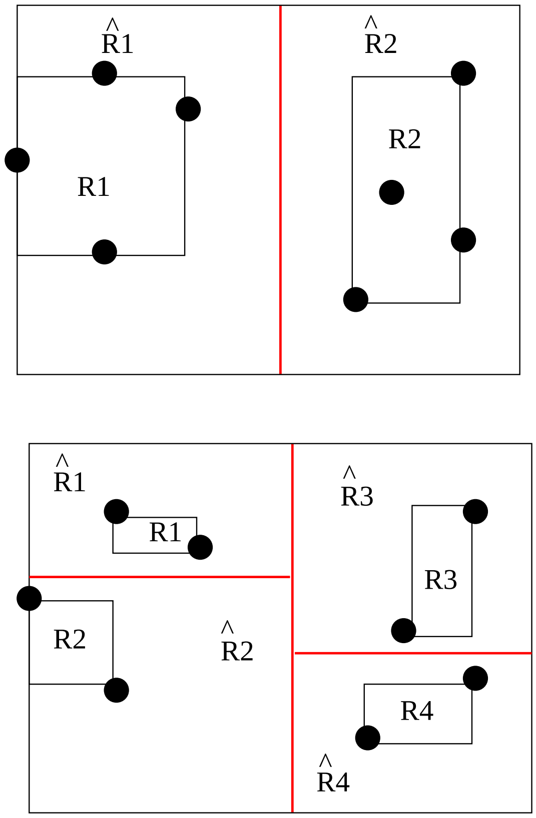

Figure 107: Outer Rectangle

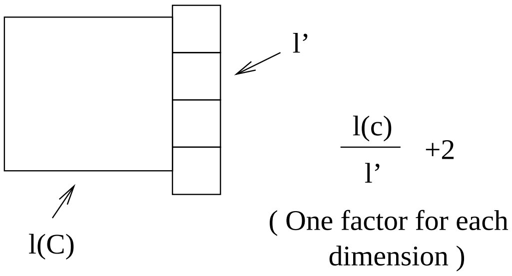

Figure 108: Proof of Lemma 2

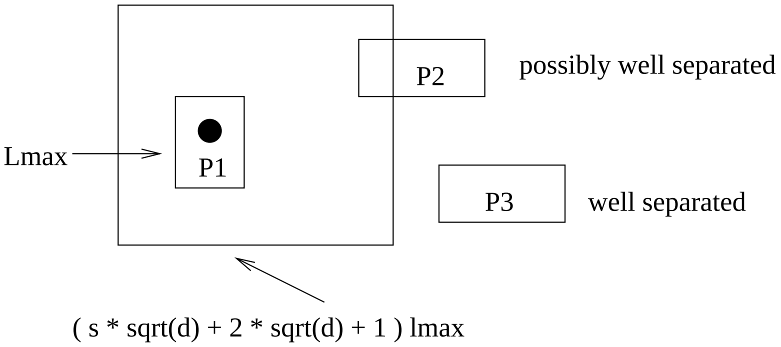

Figure 109: Well separatedness

Lemma 2 *Given a d-cube C and a set S of disjoint d-cubes of length l' that overlap C, then*

$$|S| \leq \left(\frac{l(C)}{l'} + 2\right)^d$$

.

Proof: By picture (Figure 108). □

It's not hard to show that if $l_{\max}(P_1) \geq l_{\max}(P_2)$, then for P_2 not to be well separated from P_1 it must intersect a d-cube of length $(s\sqrt{d} + 2\sqrt{d} + 1)l_{\max}(P_1)$ centered at the center of P_1 (Figure 109).

Note: we will only split P_1 in WSR_Pair(P_1, P_2) if P_1 and P_2 are not well separated.

Putting everything together

Consider for a given node P of the decomposition tree, all invocations WSR_Pair(P,P') such that $l_{\max}(P) \geq l_{\max}(P')$.

1. all P' must be independent (consider recursive structure of the algorithm WSR_Pair)
2. we know $l_{\min}(\hat{R}(P')) \geq \frac{l_{\max}(p(P'))}{2}$ so we can bound the number of P' to

$$|P'| < (2(s\sqrt{d} + 2\sqrt{d} + 1) + 2)^d$$

.

Therefore, the total # of calls to WSR_Pair(P_1,P_2) is bound by

$$2n(2(s\sqrt{d} + 2\sqrt{d} + 1) + 2)^d = O(n)$$

The factor of two at the front comes from the fact that we are counting non-leaf invocations of WSR_Pair(T_1,T_2) while the interactions are created by leaf invocations. However, the total number of leafs is at most twice the number of internal nodes—see Figure 110.

We note that this bound is very conservative and in practice the algorithm actually uses many fewer interactions.

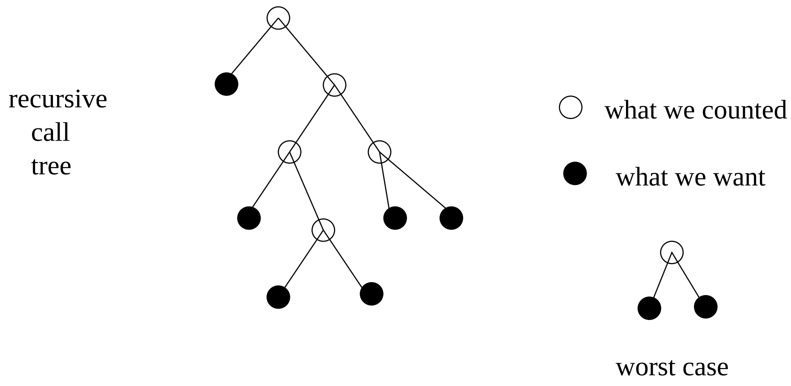

Figure 110: Counting leaves only

3 N-body Summary

- **Algorithms**
  - Particle-Particle
  - Particle-mesh (several variants)
  - Particle-cell (Appel, Barnes-Hut, Press)
  - Cell-cell (FMM, Callahan-Kosaraju)
- **Applications**
  - Astronomy
  - Molecular Dynamics
  - Fluid Dynamics (elementless methods)
  - Plasma Physics