15-853:Algorithms in the Real World

Nearest Neighbors and Spatial Decompositions

-Introduction
-Quad/Oct/Kd/BSP trees
-Nearest Neighbor Search
-Metric spaces
-Ball/Cover trees
-Callahan-Kosaraju
-Bounded Volume Hierarchies

Spatial Decompositions

Goal to partition low or medium "dimensional" space into a hierarchy so that various operations can be made efficiently.

Examples:

- Quad/Oct trees
- K-d trees
- Binary space partitioning (BSP)
- Bounded volume hierarchies (BVH)
- Cover trees
- Ball trees
- Well-separated pair decompositions
- R-trees

Applications

Simulation

- N-body simulation in astronomy, molecular dynamics, and solving PDEs
- Collision detection

Machine learning and statistics

- Clustering and nearest neighbors
- Kernel Density estimation
- Classifiers

Graphics

- Ray tracing
- Radiosity
- Occlusion Culling

Applications

Geometry

- Range searching
- All nearest neighbors

Databases

- Range searching
- Spatial indexing and joins

Robotics

- Path finding

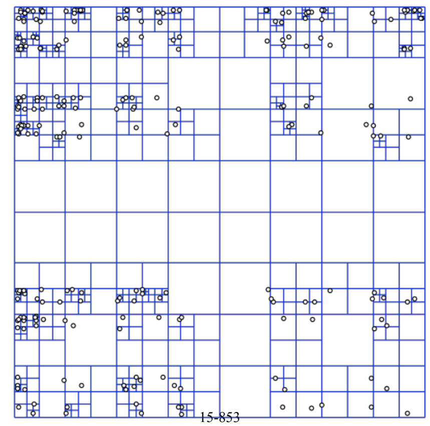
Trees in Euclidean Space: Quad/Oct

Quad (2d) and Oct (3d)Trees:
- Find an axis aligned bounding box for all points
- Recursively cut into 2^d equal parts

If points are "nice" then will be O(log n) deep and will take O(n log n) time to build.

In the worst case can be O(n) deep. With care in how built can still run in O(n log n) time for constant dimension.

Trees in Euclidean Space: Quad/Oct



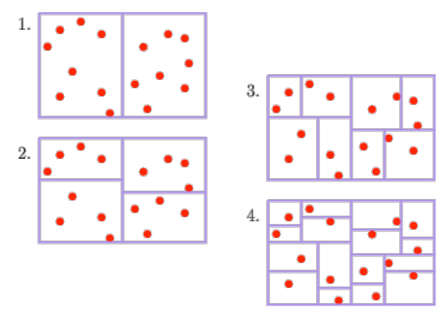
Trees in Euclidean Space: k-d tree

Similar to Quad/Oct but cut only one dimension at a time.
- Typically cut at median along the selected dimension
- Typically pick longest dimension of bounding box
- Could cut same dimension multiple times in a row

If cut along medians then no more than O(log n) deep and will take O(n log n) time to build (although this requires a linear time median, or presorting the data along all dimensions).

But, partitions themselves are much more irregular.

Trees in Euclidean Space: k-d tree

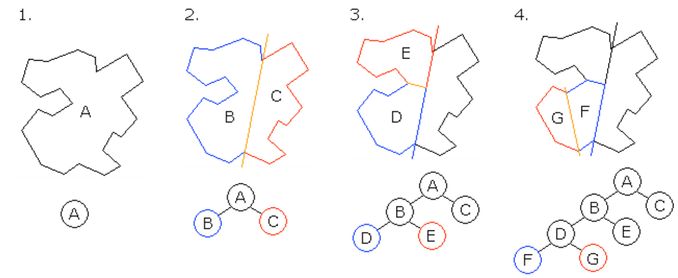


Trees in Euclidean Space: BSP

Binary space partitioning (BSP)
- Cuts are not axis aligned
- Typically pick cut based on a feature, e.g. a line segment in the input

Tree depth and runtime depends on how cuts are selected

Trees in Euclidean Space: BSP

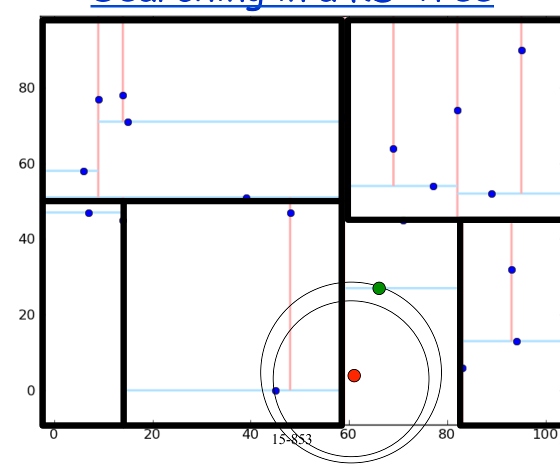


Nearest Neighbors on a Decomposition Tree

```
NearestNeighbor(p,T) =
   N = find leaf p belongs to
   p' = nearest neighbor of p in N (if any, otherwise empty)
   while (N not the root)
       for each child C of P(N) except N
          p' = Nearest(p,p',C)
       N = P(N)
   retun p'

Nearest(p,p',T) =
  if anything in T can be closer to p than p'
    if T is leaf return nearest in leaf or p'
    else for each child C of T
        p' = Nearest(p,p',C)
  return p'
```


Searching in a KD Tree



What about other Metric Spaces?

Consider set of points S in some metric space (X,d)

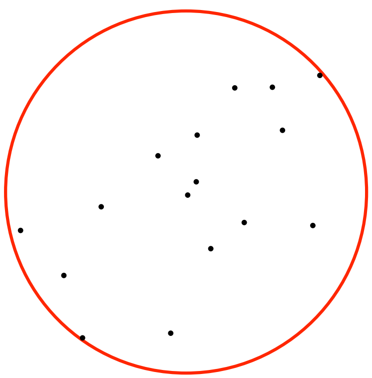
1. $d(x, y) \geq 0$ (*non-negativity*)
2. $d(x, y) = 0$ iff $x = y$ (identity)
3. $d(x, y) = d(y, x)$ (*symmetry*)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Some metric spaces:

- Euclidean metric
- Manhattan distance (l_1)
- Edit distance
- Shortest paths in a graph

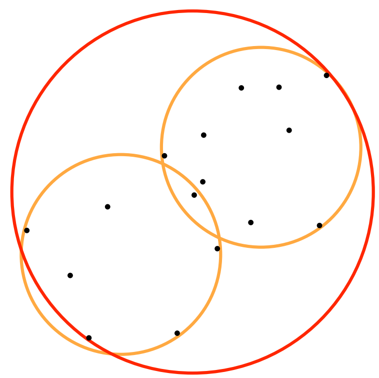
Ball Trees

Divide a metric space into a hierarchy of balls. The union of the balls of the children of a node cover all the elements of the node.

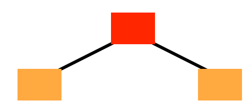


A ball-tree: level 1

Slide from Alex Gray

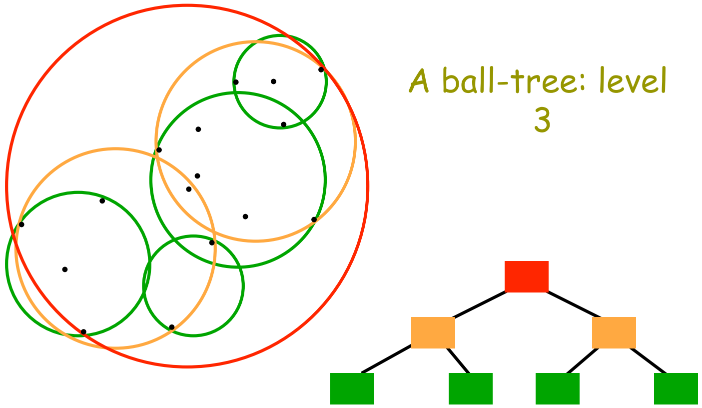


A ball-tree: level 2



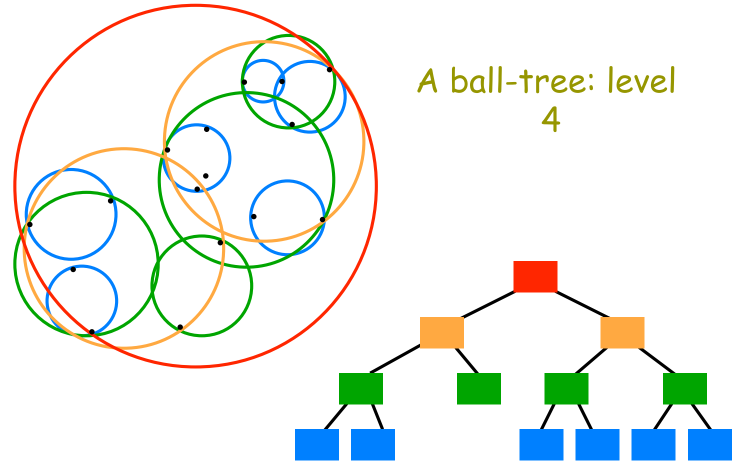
Slide from Alex Gray

A ball-tree: level 3



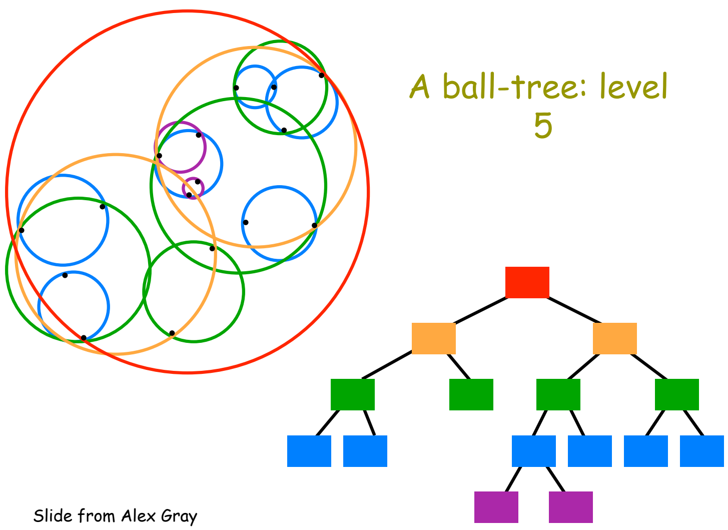
Slide from Alex Gray

A ball-tree: level 4



Slide from Alex Gray

A ball-tree: level 5



Slide from Alex Gray

Ball Trees

Need to decide how to find balls that cover the children.

Can be a lot of waste due to overlap of balls.

Can copy a node to all children it belongs to, or just put in one....depends on application.

Can be used for our nearest neighbor search.

Hard to say anything about costs of ball trees in general for arbitrary metric spaces.

Measures of Dimensionality

Ball of radius r around a point p from a point set S taken from a metric space X

$$B_S(p,r) = \{q \in S : d(p,q) \le r\}$$

A point set has a (t,c)-*Expansion* if
for all p in X and r > 0,
$|B_s(r)| \ge t$ implies $|B_S(p,2r)| \le c|B_S(p,r)|$.
c is called the *Expansion-Constant*,
t is typically O(log |S|)

If S is uniform in Euclidean space then c is proportional to 2^d suggesting that dim(S) = log c
This is referred to as the KR dimension.

Measures of Dimensionality

The *Doubling Constant* for a metric space X is the minimum value c such that every ball in X can be covered by c balls in X of half the radius.

More general than the KR dimension, but harder to prove bounds based on it.

Cover Trees

The following slides are from a presentation of Victoria Choi.

Cover trees work for arbitrary metrics but bounds depend on expansion or doubling constants

Cover Tree Data Structure

A cover tree *T* on a dataset *S* is a leveled tree where each level is indexed by an integer scale *i* which decreases as the tree is descended

C_i denotes the set of nodes at level *I*

d(p,q) denotes the distance between points *p* and *q*

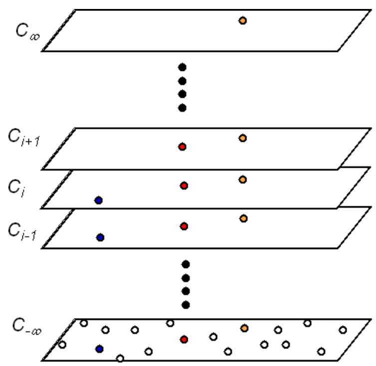
A valid tree satisfies the following properties

- Nesting: $C_i \subset C_{i-1}$
- Covering tree: For every node $p \in C_{i-1}$, there exists a node $q \in C_i$ satisfying $d(p,q) \le 2^i$ and exactly one such *q* is a parent of *p*
- Separation: For all nodes $p,q \in C_i$, $d(p,q) > 2^i$

Nesting

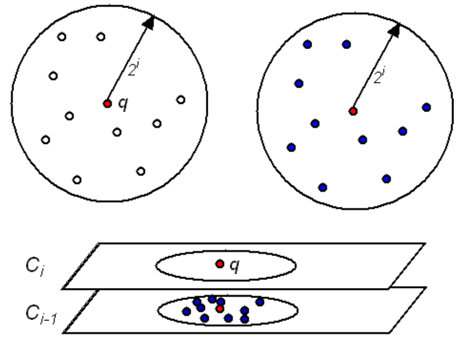
$C_i \subset C_{i-1}$

- Each node in set C_i has a self-child
- All nodes in set C_i are also nodes in sets C_j where j<i
- Set $C_{-\infty}$ contains all the nodes in dataset S



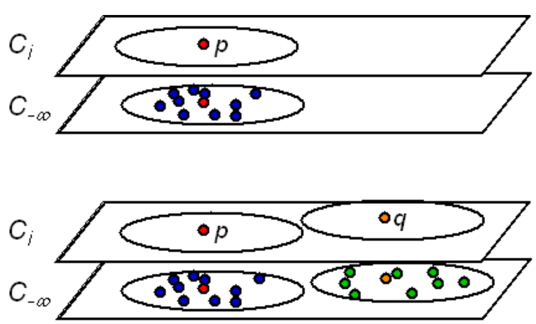
Covering Tree

For every node $p \in C_{i-1}$, there exists a node $q \in C_i$ satisfying $d(p,q) \le 2^i$ and exactly one such q is a parent of p



Separation

For all nodes $p,q \in C_i \quad d(p,q) > 2^i$



Tree Construction

Single Node Insertion (recursive call)

Insert(point p, cover set Q_i, level i)
- set $Q = \{Children(q) : q \in Q_i\}$
- if $d(p,Q) > 2^i$ then return "no parent found"
- else
  - set $Q_{i-1} = \{q \in Q : d(p,q) \le 2^i\}$
  - if Insert($p, Q_{i-1}, i-1$) = "no parent found" and $d(p,Q_i) \le 2^i$
    - pick $q \in Q_i$ satisfying $d(p,q) \le 2^i$
    - insert q into Children(q)
    - return "parent found"
  - else return "no parent found"

Batch insertion algorithm also available

Searching

Iterative method : find p

set $Q_\infty = C_\infty$

for i from ∞ down to $-\infty$

consider the set of children of Q_i:

$$Q = \{Children(q) : q \in Q_i\}$$

form next cover set:

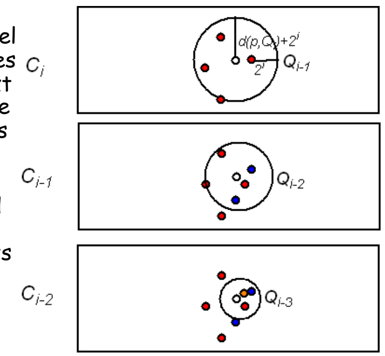
$$Q_{i-1} = \{q \in Q \mid d(p,q) \le d(p,Q) + 2^i\}$$

return $\arg\min_{q \in Q_{-\infty}} d(p,q)$

Why can you always find the nearest neighbour?

When searching for the nearest node at each level i, the bound for the nodes to be included in the next cover set Q_{i-1} is set to be $d(p,Q)+2^i$ where $d(p,Q)$ is the minimum distance from nodes in Q

Q will always center around the query node and will contain at least one of its nearest neighbours



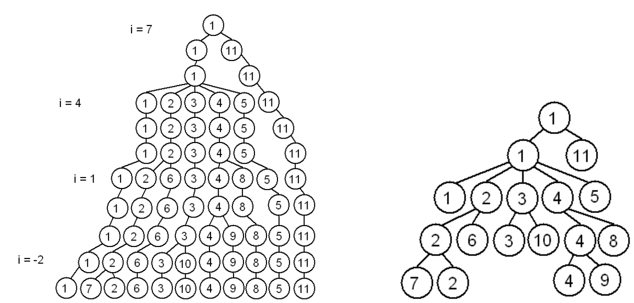
How?

All the descendents of a node q in C_{i-1} are at most 2^i away ($2^{i-1} + 2^{i-2} + 2^{i-3} + \ldots$)

By setting the bound to be $d(p,Q)+2^i$, we have included all the nodes with descendents which might do better than node p in Q_{i-1} and eliminated everything else

Implicit v. Explicit

Theory is based on an implicit implementation, but tree is built with a condensed explicit implementation to preserve O(n) space bound



Bounds on Cover Trees

For an expansion constant c:

The number of children of any node is bounded by c^4 (width bound)

The maximum depth of any point in the explicit tree is $O(c^2 \log n)$ (depth bound)

Runtime for a search is $O(c^{12} \log n)$

Runtime for insertion or deletion is $O(c^6 \log n)$

Callahan-Kosaraju

Well separated pair decompositions

- A decomposition of points in d-dimensional space

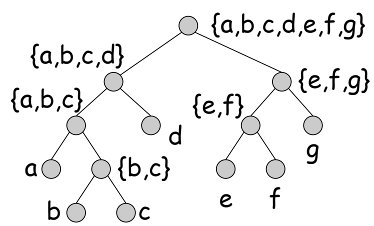
Applications

- N-body codes (calculate interaction forces among n bodys)
- K-nearest-neighbors O(n log n) time

Similar to k-d trees but better theoretical properties

Tree decompositions

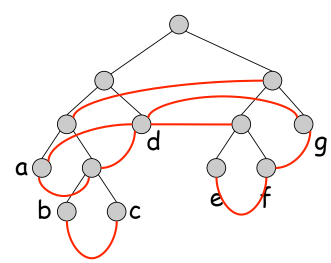
A spatial decomposition of points



A "realization"

A single path between any two leaves consisting of tree edges up, an interaction edge across, and tree edges down.

interaction edge

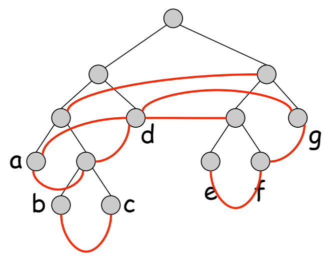


Quiz: 1 edge is missing

A "well-separated realization"

A realization such that the endpoints of each interaction edge is "well separated"

Goal: show that the number of interaction edges is O(n)



Overall approach

Build tree decomposition: O(n log n) time
Build well-separated realization: O(n) time

Depth of tree = O(n) worst case, but not in practice

We can bound number of interaction edges to O(n)

- For both n-body and nearest-neighbors we only need to look at the interaction edges

Callahan Kosaraju Outline

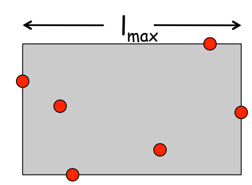
Some definitions
Building the tree
Generating well separated realization
Bounding the size of the realization
Using it for nearest neighbors

Some Definitions

Bounding Rectangle R(P)

Smallest rectangle that contains a set of points P

l_{max} : maximum length of a rectangle

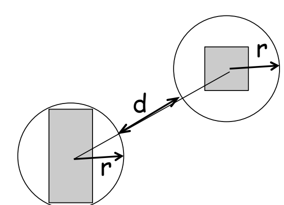


Well Separated:

r = smallest radius that can contain either rectangle

s = separation constant

d > s * r



More Definitions

Interaction Product

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

A **Realization** of $A \otimes B$

Is a set $\{\{A_1,B_1\}, \{A_2,B_2\}, \ldots, \{A_k,B_k\}\}$

such that

1. $A_i \subseteq A,\ B_i \subseteq B \quad i = 1\ldots k$
2. $A_i \cap B_i = \emptyset$
3. $(A_i \otimes B_i) \cap (A_j \otimes B_j) = \emptyset \quad (i \neq j)$
4. $A \otimes B = \cup_{i=1}^{k} A_i \otimes B_i$

This formalize the "cross edges"

A well-separated realization

$\{\{A_1,B_1\}, \{A_2,A_3\}, \ldots, \{A_k,B_k\}\}$

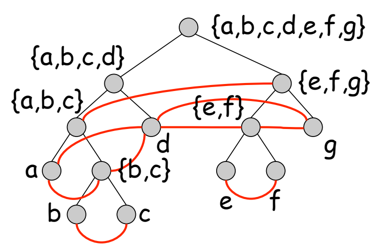
such that $R(A_i)$ and $R(B_i)$ are well separated

A well-separated pair decomposition =

Tree decomposition of P

+ well-separated realization of $P \otimes P$ where the subsets are the nodes of the tree

A well-separated pair decomposition



P = {a,b,c,d,e,f,g}

Realization of $P \otimes P$ =

{{{a,b,c},{e,f,g}}, {{d},{e,f}}, {{d},{b,c}}, {{a},{b,c}}, {{a},{d}}, {{b},{c}}, {{d},{g}}, {{e},{f}}, {{e,f},{g}}}

Algorithm: Build Tree

Function Tree(P)

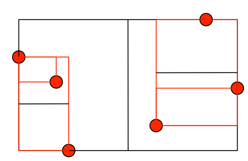
if |P| = 1 then **return** leaf(P)

else

d_{max} = dimension of l_{max}

P_1, P_2 = split P along d_{max} at midpoint

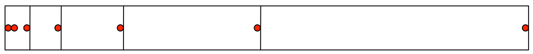
Return Node(Tree(P_1),Tree(P_2), l_{max})



Runtime: naive

Naively:

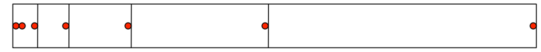
Each cut could remove just one point



$T(n) = T(n-1) + O(n) = O(n^2)$

This is no good!!

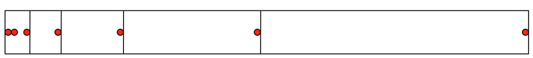
Runtime: better



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

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

Runtime: better



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

$$\text{such that: } \sum_{i=1}^{k} n_i = n$$

$$\text{and: } \max_{i=1}^{k}(n_i) < \frac{2}{3} n$$

$$T(n) = O(n \lg n)$$

Algorithm: Generating the Realization

function wsr(T)

if leaf(T) **return** ∅

else return wsr(left(T)) ∪ wsr(right(T))
∪ wsrP(left(T),right(T))

function wsrP(T_1, T_2)

if wellSep(T_1,T_2) **return** {(T_1,T_2)}

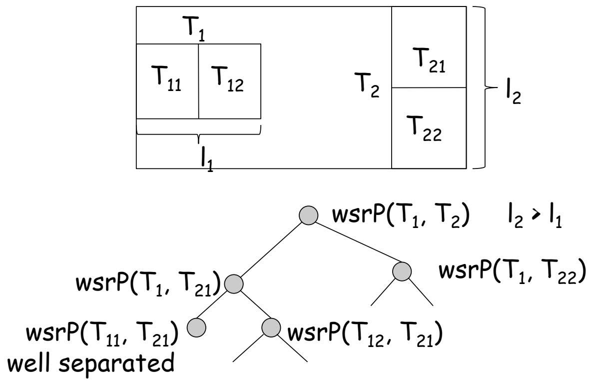
else if $l_{max}(T_1) > l_{max}(T_2)$ **then**

return wsrP(left(T_1), T_2) ∪ wsrP(right(T_1), T_2)

else

return wsrP(T_1, left(T_2)) ∪ wsrP(T_1, right(T_2))

WSR



Bounding Interactions

Just an intuitive outline:

- Can show that tree nodes do not get too thin
- Can bound # of non-overlapping rectangles that can touch a cube of fixed size
- Can bound number of interaction per tree node

Total calls to wsrP is bounded by

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

This bounds both the time for WSR and the number of interaction edges created.

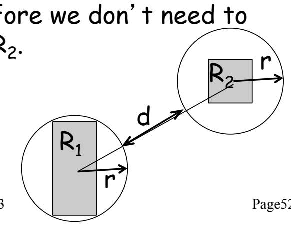
Summary so far

O(n log n) time to build tree
O(n) time to calculate WS Pair Decomposition
O(n) edges in decomposition

Finding everyone's nearest neighbor

Build well-separated pair decomposition with s = 2.

- Recall that d > sr = 2r to be well separated
- The furthest any pair of points can be to each other within one of the rectangles is 2r
- Therefore if d > 2r then for a point in R_1 there must be another point in R_1 that is closer than any point in R_2. Therefore we don't need to consider any points in R_2.



Finding everyone's nearest neighbor

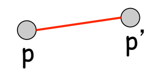
Now consider a point p.

It interacts with all other points p' through an interaction edge that goes from:

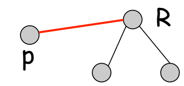
1. p to p'
   (check these distances directly)
2. p to an ancestor R of p'
   (check distance to all descendants of R)
3. an ancestor of p to p' or ancestor of p'
   (p' cannot be closest node)

Step 2 might not be efficient, but efficient in practice and can be made efficient in theory

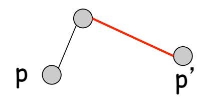
Again: in pictures



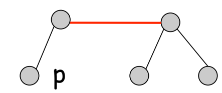
calculate d(p,p')



for p' in R, calculate d(p,p')



ignore for p



ignore

Take minimum of all distances calculated.

The N-body problem

Calculate the forces among n "bodys". Naïve method requires considering all pairs and takes $O(n^2)$ time.

Using Kallahan-Kosaraju can get approximate answer in $O(n)$ time plus the time to build the tree.

Used in astronomy to simulate the motion of starts and other mass

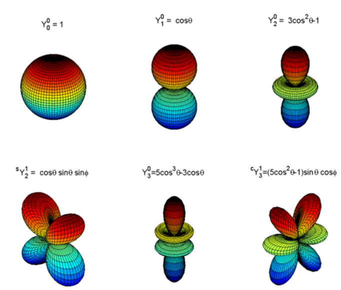
Used in biology to simulate protein folding

Used in engineering to simulate PDEs (can be better than Finite Element Meshes for certain problems)

Used in machine learning to calculate certain Kernels

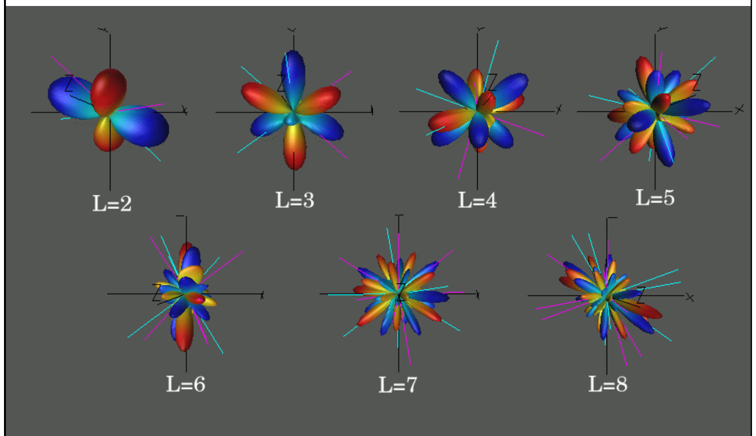
The N-body problem

Can approximate the force/potential due to a set of points by a multipole expansion truncated to a fixed number of terms (sort of like a taylor series).



The spherical harmonics

Potential due to Y_l term goes off as $1/r^{l+1}$ so far away the low terms dominate.



The N-body problem

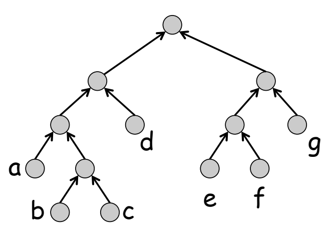
If a set of points in well-separated from p, then can use the approximation instead of all forces.

Need "inverse" expansion to pass potential down from parents to children.

The N-body problem

If a set of points in well-separated from p, then can use the approximation instead of all forces.

Need "inverse" expansion to pass potential down from parents to children.

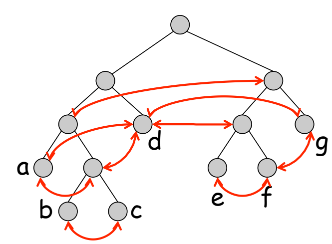


Translate and add "multipole" terms going up the tree. They add linearly.

The N-body problem

If a set of points in well-separated from p, then can use the approximation instead of all forces

Need "inverse" expansion to pass potential down from parents to children.

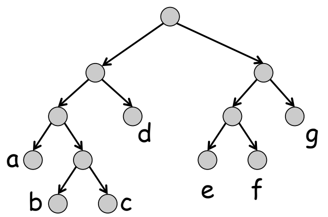


Invert expansions across the interaction edges.

The N-body problem

If a set of points in well-separated from p, then can use the approximation instead of all forces

Need “inverse” expansion to pass potential down from parents to children.



Copy add and translate the inverse expansions down the tree.
Calculate approximate total force at the leaves.

The N-body problem

If a set of points in well-separated from p, then can use the approximation instead of all forces.

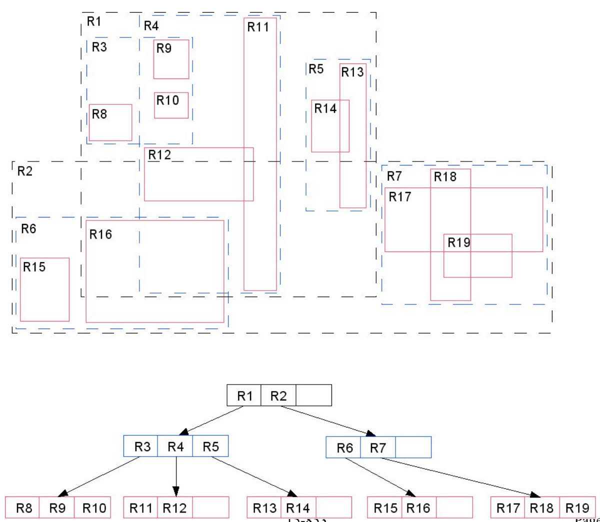
Need “inverse” expansion to pass potential down from parents to children.

Total time is:

- O(n) going up the tree
- O(n) inverting across interaction edges
- O(n) going down the tree

The constant in the big-O and the accuracy depend on the number of terms used. More terms is more costly but more accurate.

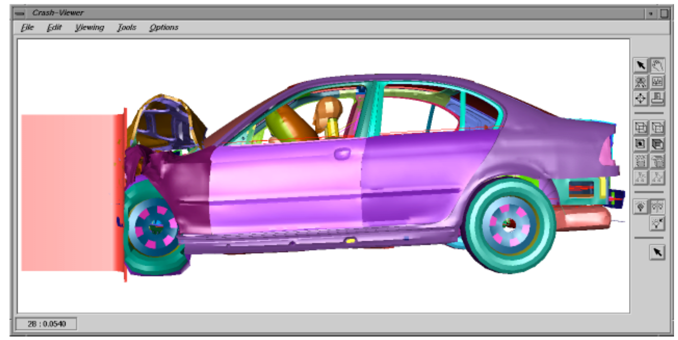
The N-body problem



Bounding Volume Hierarchies

Hierarchically partition objects (instead of points).
Used in applications in graphics and simulation:

- Ray tracing
- Collision detection
- Visibility culling

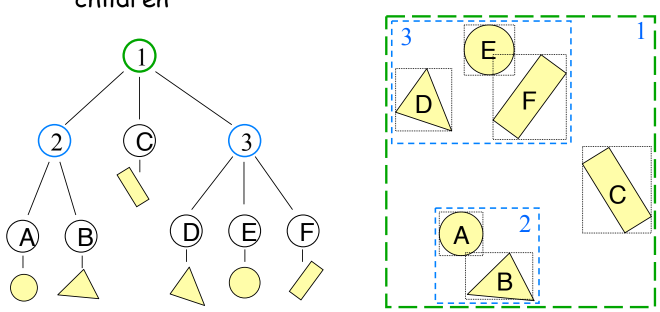


Bounding Volume Hierarchies I

From: Aaron Bloomfield, U. Virginia

Build hierarchy of bounding volumes

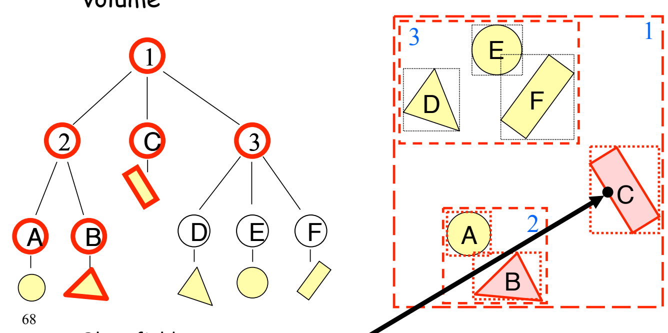
- Bounding volume of interior node contains all children



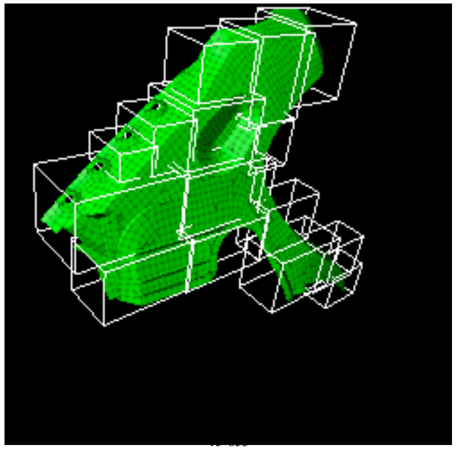
Bounding Volume Hierarchies

Use hierarchy to accelerate ray intersections

- Intersect node contents only if hit bounding volume



A. Bloomfield



Building the Hierarchy

Goals:

- Elements in a subtree should be near each other
- Each node should be of “minimum volume”
- Sum of all bounding volumes should be minimal
- Greater attention should be paid at the root
- Overlap should be small
- Balanced

Two main approaches for construction

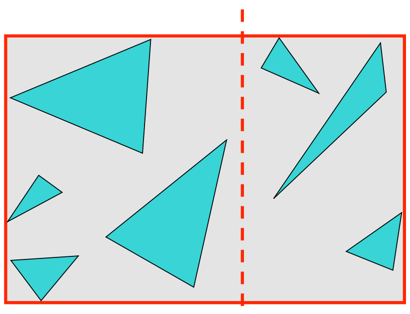
- Top dow
- Bottom up

Bounding Volume Hierarchy: Top Down

Find bounding box of objects

Split objects into two groups

Recurse



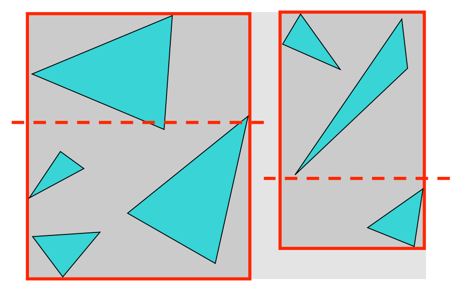
From: Durand and Cutler, MIT

Bounding Volume Hierarchy

Find bounding box of objects

Split objects into two groups

Recurse



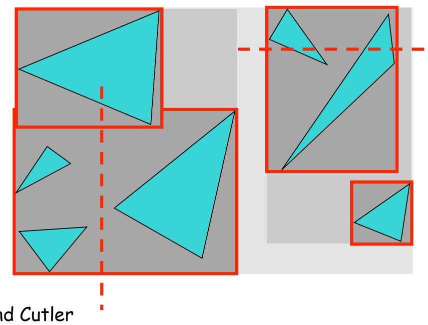
Durand and Cutler

Bounding Volume Hierarchy

Find bounding box of objects

Split objects into two groups

Recurse



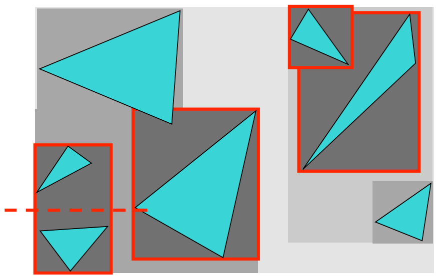
Durand and Cutler

Bounding Volume Hierarchy

Find bounding box of objects

Split objects into two groups

Recurse



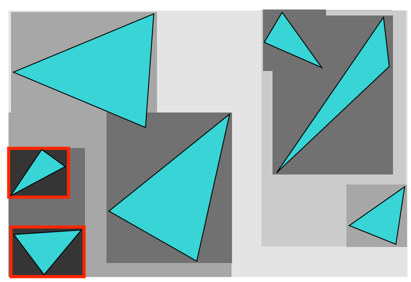
Durand and Cutler

Bounding Volume Hierarchy

Find bounding box of objects

Split objects into two groups

Recurse



Durand and Cutler

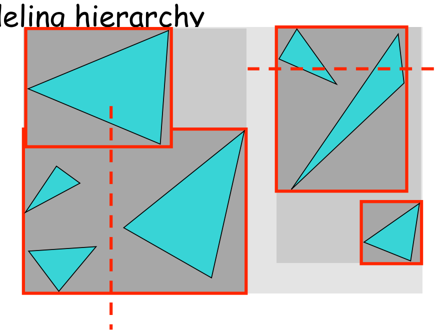
Where to split objects?

At midpoint *OR*

Sort, and put half of the objects on each side

OR

Use modeling hierarchy

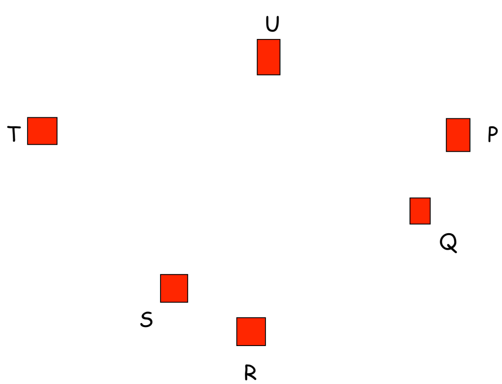


Following slides from:
Bruce Walter, Kavita Bala,
Cornell University
Milind Kulkarni, Keshav Pingali
University of Texas, Austin

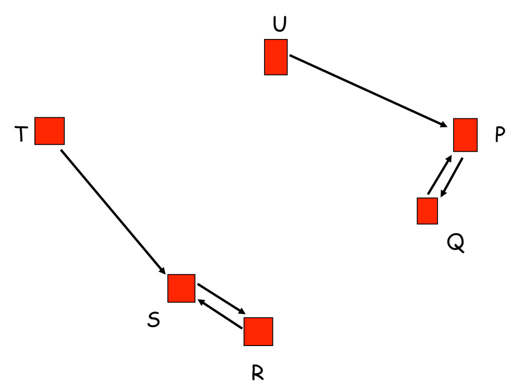
Heap-based Algorithm

Initialize KD-Tree with elements
Initialize heap with best match for each element
Repeat {
  Remove best pair <A,B> from heap
  If A and B are active clusters {
    Create new cluster C = A+B
    Update KD-Tree, removing A and B and inserting C
    Use KD-Tree to find best match for C and insert into heap
  } else if A is active cluster {
    Use KD-Tree to find best match for A and insert into heap}
} until only one active cluster left

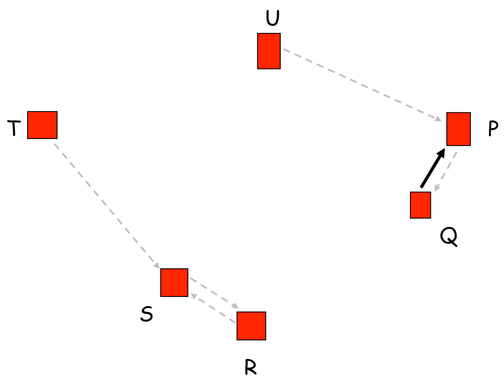
Heap-based Algorithm Example



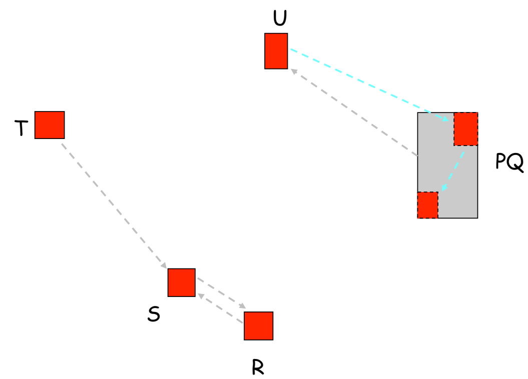
Heap-based Algorithm Example



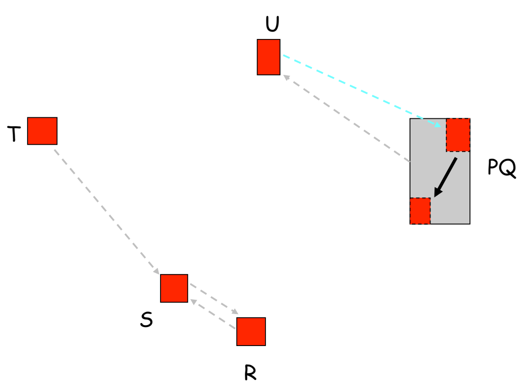
Heap-based Algorithm Example



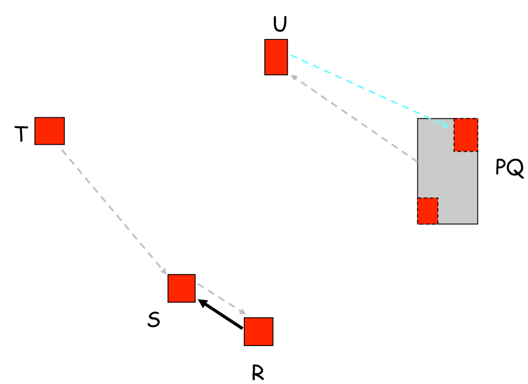
Heap-based Algorithm Example



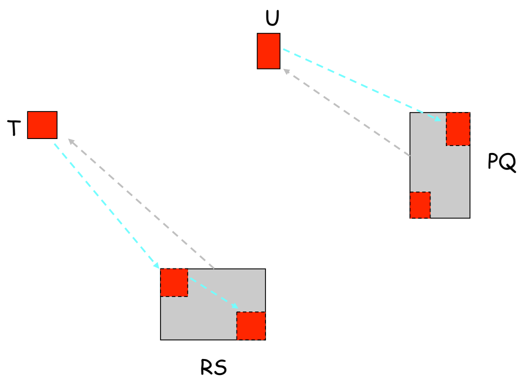
Heap-based Algorithm Example



Heap-based Algorithm Example



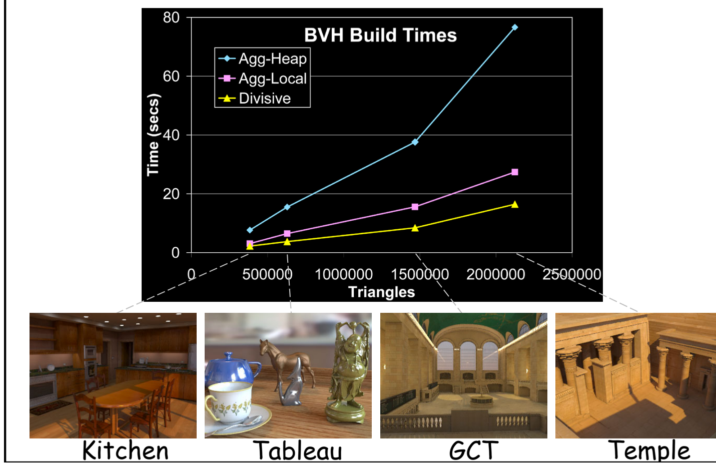
Heap-based Algorithm Example



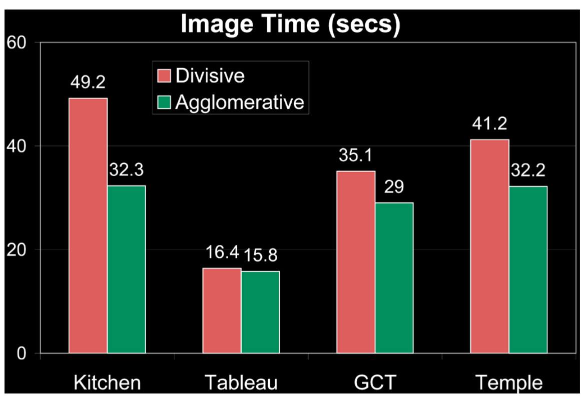
Walter, Bala, Kulkarni, Pingali

Walter, Bala, Kulkarni, Pingali

Results: BVH



Results: BVH



1280x960 Image with 16 eye and 16 shadow rays per pixel, without build ti

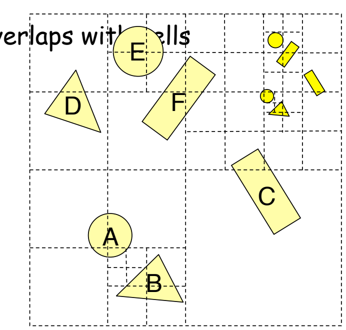
Walter, Bala, Kulkarni, Pingali

Octree

Construct adaptive grid over scene

- Recursively subdivide box-shaped cells into 8 octants
- Index primitives by overlaps with cells

Generally fewer cells



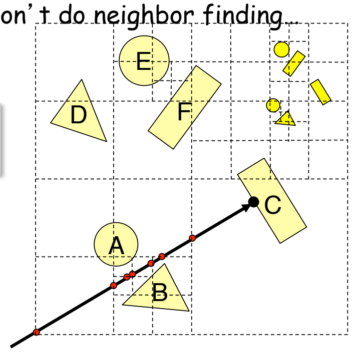
88

A. Bloomfield

Octree

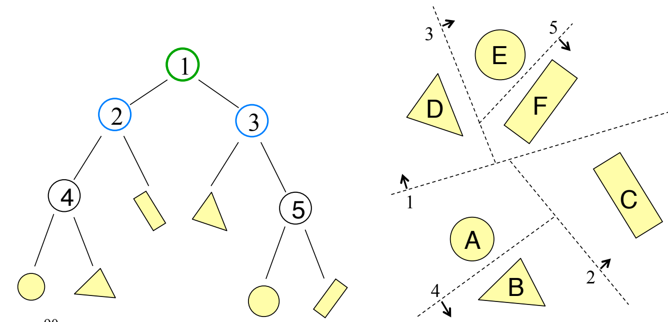
Trace rays through neighbor cells
- Fewer cells
- Recursive descent – don't do neighbor finding…

Trade-off fewer cells for more expensive traversal



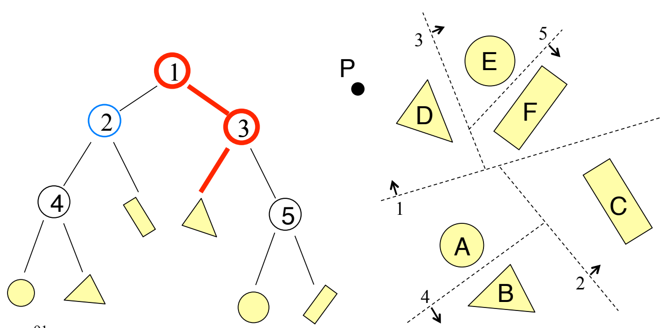
Binary Space Partition (BSP) Tree

Recursively partition space by planes
- Every cell is a convex polyhedron



Binary Space Partition (BSP) Tree

Simple recursive algorithms
- Example: point location



Binary Space Partition (BSP) Tree

Trace rays by recursion on tree
- BSP construction enables simple front-to-back traversal