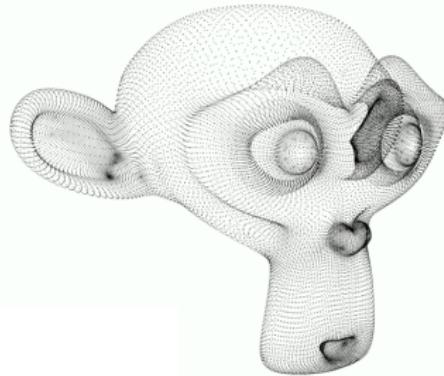


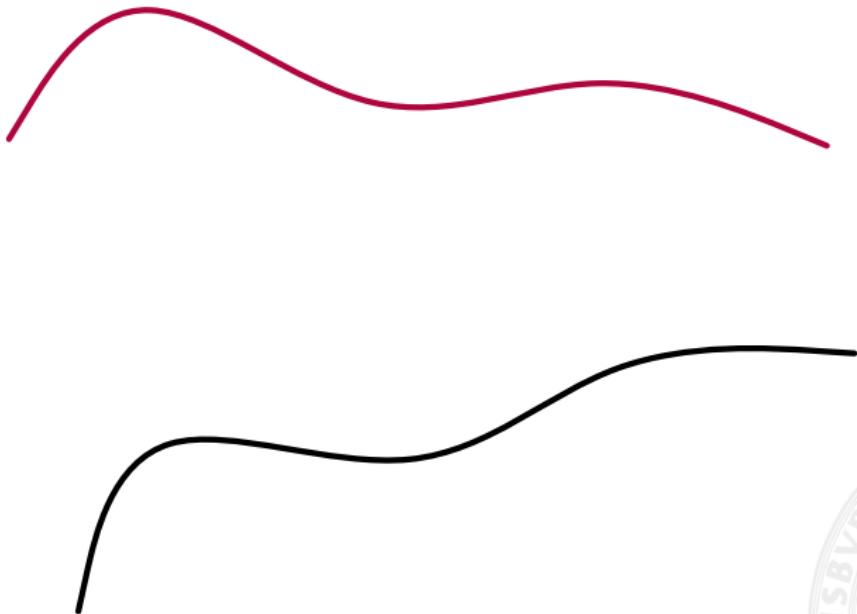
Point Cloud Registration



Least-Squares Fitting & Iterative Closest Point & k-D Trees

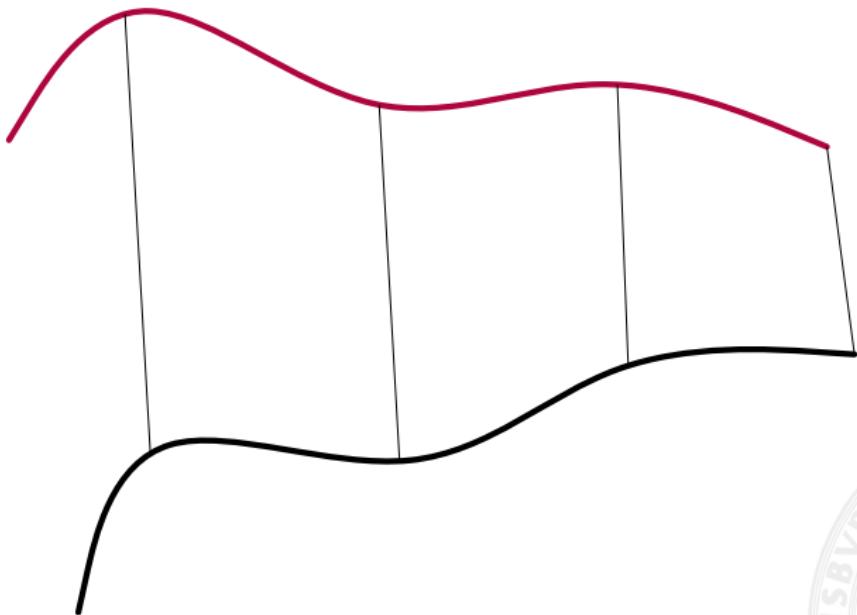
Motivation

Say that two curves are given ...



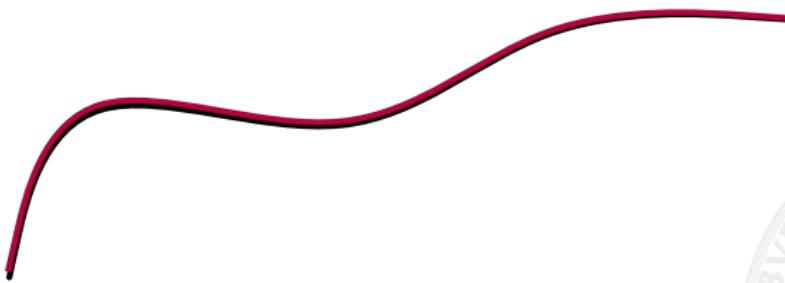
Motivation

... and the (exact) point correspondences are known



Motivation

We can find the correct rotation+translation+scaling to align them!



Motivation

Iterative Closest Point (ICP) Algorithm

If the correspondences are not given, assume that the *closest points* correspond → *Iterative closest point (ICP)* algorithm.

Motivation

A more practical example (3D point cloud of the environment):

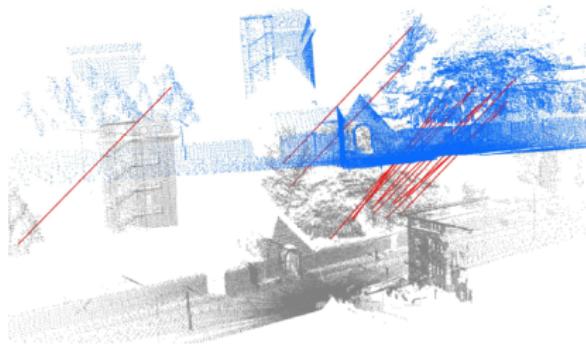


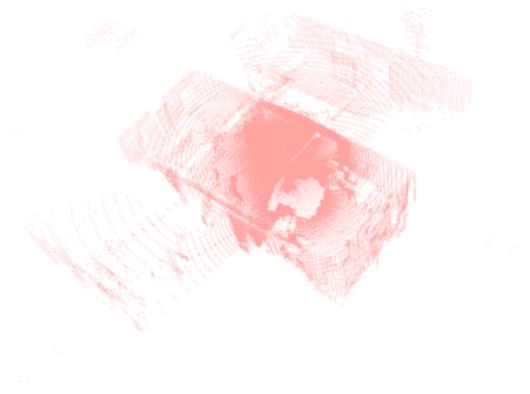
Image source (Point cloud library):



<http://docs.pointclouds.org>

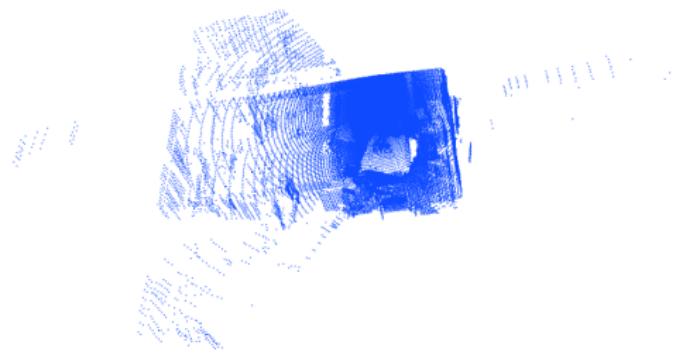
Motivation

Source: Cloud A



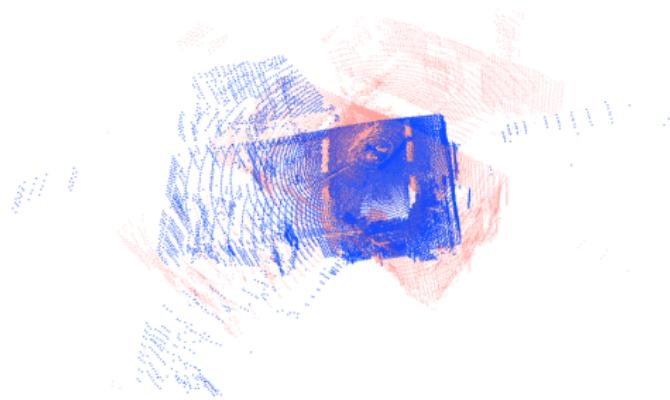
Motivation

Target: Cloud B



Motivation

How can we “align” B with A?



Least-Squares (LS) Fitting of 3-D Point Sets

Assume both point clouds have an **equal number of points**.

Least-Squares (LS) Fitting of 3-D Point Sets

Assume both point clouds have an **equal number of points**.

We have ...

- point sets $\{\mathbf{x}_i\}_{i=1}^N$ and $\{\mathbf{y}_i\}_{i=1}^N$ with $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^3$ (same size)
- define: $\mathbf{x}'_i = \mathbf{x}_i - \bar{\mathbf{x}}$ and $\mathbf{y}'_i = \mathbf{y}_i - \bar{\mathbf{y}}$

Least-Squares (LS) Fitting of 3-D Point Sets

Assume both point clouds have an equal number of points.

We have ...

- point sets $\{\mathbf{x}_i\}_{i=1}^N$ and $\{\mathbf{y}_i\}_{i=1}^N$ with $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^3$ (same size)
- define: $\mathbf{x}'_i = \mathbf{x}_i - \bar{\mathbf{x}}$ and $\mathbf{y}'_i = \mathbf{y}_i - \bar{\mathbf{y}}$

Optimization problem:

$$\min_{\mathbf{R} \in SO(3), s \in \mathbb{R}, \mathbf{t} \in \mathbb{R}^3} E(\mathbf{R}, \mathbf{t}, s) = \sum_{i=1}^N \|\mathbf{y}'_i - s\mathbf{R}\mathbf{x}'_i - \mathbf{t}\|^2$$

where $SO(3) = \{\mathbf{R} \in \mathbb{R}^{3 \times 3} : \mathbf{R}^\top \mathbf{R} = \mathbf{I}_3, \det(\mathbf{R}) = +1\}$, s is scaling and \mathbf{t} translation.

Least-Squares (LS) Fitting of 3-D Point Sets

Part I: Let's expand the error term, i.e.,

$$\|\mathbf{y}_i - s\mathbf{R}\mathbf{x}_i - \mathbf{t}\|^2 = (\mathbf{y}_i - s\mathbf{R}\mathbf{x}_i - \mathbf{t})^\top (\mathbf{y}_i - s\mathbf{R}\mathbf{x}_i - \mathbf{t})$$

Remember, $(a + b + c)^2 = a^2 + 2ab + b^2 + 2ac + 2bc + c^2$.

Least-Squares (LS) Fitting of 3-D Point Sets

Part I: Let's expand the error term, i.e.,

$$\|\mathbf{y}_i - s\mathbf{R}\mathbf{x}_i - \mathbf{t}\|^2 = (\mathbf{y}_i - s\mathbf{R}\mathbf{x}_i - \mathbf{t})^\top (\mathbf{y}_i - s\mathbf{R}\mathbf{x}_i - \mathbf{t})$$

Remember, $(a + b + c)^2 = a^2 + 2ab + b^2 + 2ac + 2bc + c^2$.

Consequently, the energy E can be written as

$$\begin{aligned} \sum_i \|\mathbf{y}_i - s\mathbf{R}\mathbf{x}_i - \mathbf{t}\|^2 &= \sum_i \mathbf{y}_i^\top \mathbf{y}_i - 2s \sum_i \mathbf{x}_i^\top \mathbf{R} \mathbf{y}_i + s^2 \sum_i \underbrace{\mathbf{x}_i^\top \mathbf{R}^\top \mathbf{R} \mathbf{x}_i}_{\mathbf{x}_i^\top \mathbf{x}_i} - \\ &\quad 2 \left(\sum_i \mathbf{y}_i^\top \right) \mathbf{t} + 2s \left(\sum_i \mathbf{x}_i^\top \right) \mathbf{R}^\top \mathbf{t} + N \mathbf{t}^\top \mathbf{t} \end{aligned}$$

(\mathbf{R} is symmetric, hence $\mathbf{R}^\top = \mathbf{R}$)

Least-Squares (LS) Fitting of 3-D Point Sets

Taking dE/dt and setting the result to 0 allows us to solve for the optimal translation t^* , i.e.,

Least-Squares (LS) Fitting of 3-D Point Sets

Taking dE/dt and setting the result to 0 allows us to solve for the optimal translation \mathbf{t}^* , i.e.,

$$\begin{aligned} 2s\mathbf{R} \sum_i \mathbf{x}_i - 2 \left(\sum_i \mathbf{y}_i \right) + 2N\mathbf{t} &= 0 \\ \Rightarrow \frac{1}{N} \sum_i \mathbf{y}_i - s\mathbf{R} \frac{1}{N} \sum_i \mathbf{x}_i &= \mathbf{t}^* \end{aligned}$$

or, equivalently,

$$\mathbf{t}^* = \bar{\mathbf{y}} - s\mathbf{R}\bar{\mathbf{x}}$$

(see definitions from before)

Least-Squares (LS) Fitting of 3-D Point Sets

Let's update the energy functional with this optimal choice \mathbf{t}^* , i.e.,

$$\begin{aligned} E(\mathbf{R}, \mathbf{t}, s) &= \sum_{i=1}^N \|\mathbf{y}_i - \bar{\mathbf{y}} - s\mathbf{R}\mathbf{x}_i + s\mathbf{R}\bar{\mathbf{x}}\|^2 \\ &= \sum_{i=1}^N \|(\underbrace{\mathbf{y}_i - \bar{\mathbf{y}}}_{\mathbf{y}'_i}) - s\mathbf{R}(\underbrace{\mathbf{x}_i - \bar{\mathbf{x}}}_{\mathbf{x}'_i})\|^2 \\ &= \sum_{i=1}^N \|\mathbf{y}'_i - s\mathbf{R}\mathbf{x}'_i\|^2 \end{aligned}$$

Upon expansion, we get

Least-Squares (LS) Fitting of 3-D Point Sets

Let's update the energy functional with this optimal choice \mathbf{t}^* , i.e.,

$$\begin{aligned} E(\mathbf{R}, \mathbf{t}, s) &= \sum_{i=1}^N \|\mathbf{y}_i - \bar{\mathbf{y}} - s\mathbf{R}\mathbf{x}_i + s\mathbf{R}\bar{\mathbf{x}}\|^2 \\ &= \sum_{i=1}^N \|(\underbrace{\mathbf{y}_i - \bar{\mathbf{y}}}_{\mathbf{y}'_i}) - s\mathbf{R}(\underbrace{\mathbf{x}_i - \bar{\mathbf{x}}}_{\mathbf{x}'_i})\|^2 \\ &= \sum_{i=1}^N \|\mathbf{y}'_i - s\mathbf{R}\mathbf{x}'_i\|^2 \end{aligned}$$

Upon expansion, we get

$$E(\mathbf{R}, \mathbf{t}, s) = s^2 \sum_i \mathbf{x}'_i^\top \mathbf{x}'_i - 2s \sum_i \mathbf{x}'_i^\top \mathbf{R}^\top \mathbf{y}'_i + \sum_i \mathbf{y}'_i^\top \mathbf{y}'_i$$

Least-Squares (LS) Fitting of 3-D Point Sets

What about the scale? Taking E/ds and setting the result to 0 gives

$$s^* = \frac{\sum_i (\mathbf{R}\mathbf{x}'_i)^\top \mathbf{y}'_i}{\sum_i \mathbf{x}'_i^\top \mathbf{x}'_i}$$

Horn¹ symmetrizes the error term w.r.t. scale as

$$\left\| \left(\frac{1}{\sqrt{s}} \right) \mathbf{y}'_i - \sqrt{s} \mathbf{R} \mathbf{x}'_i \right\|^2$$

which has the “nice” effect that we obtain

$$s^* = \frac{\sum_i \mathbf{y}'_i^\top \mathbf{y}'_i}{\sum_i \mathbf{x}'_i^\top \mathbf{x}'_i}$$

¹ B. Horn. “Closed-form solution of absolute orientation using unit quaternions”. In: *Journal of the Optical Society of America A* 4 (1987), pp. 629–642.

Least-Squares (LS) Fitting of 3-D Point Sets

From our error expansion (under the optimal choices for t , s), the minimization problem can now be reduced to

$$\arg \max_{R \in SO(3)} E(R, t^*, s^*) = \arg \max_{R \in SO(3)} \sum_i \mathbf{x}'_i^\top R^\top \mathbf{y}'_i$$

with (due to the cyclic property of the trace)

$$\sum_i \mathbf{x}'_i^\top R^\top \mathbf{y}'_i = \sum_i \mathbf{y}'_i^\top R \mathbf{x}'_i = \text{tr} \left(R \underbrace{\sum_i \mathbf{x}'_i \mathbf{y}'_i^\top}_H \right) = \text{tr}(RH)$$

Arun et al.² show that the optimal R is

$$R^* = \mathbf{V} \mathbf{U}^\top \quad \text{with} \quad H := \sum_i \mathbf{x}'_i \mathbf{y}'_i^\top = \underbrace{\mathbf{U} \Sigma \mathbf{V}^\top}_{\text{i.e., SVD of } H}$$

²K.S. Arun, T.S. Huang, and S.D. Blostein. "Least-Squares Fitting of Two 3-D Point Sets". In: *Pattern Analysis and Machine Intelligence* 9.5 (1986), pp. 698–700.

Iterative Closest Point (ICP)

Registration WITHOUT point correspondences

Setup: We have point clouds X and Y

- $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$
- $Y = \{\mathbf{y}_1, \dots, \mathbf{x}_m\}$

Remark

X and Y might have a different number of points!

Iterative Closest Point (ICP)

Registration without point correspondences³

1. Initialize $(\mathbf{R}^*, \mathbf{t}^*, s^*) = (\mathbf{R}_0, \mathbf{t}_0, s_0)$
2. (Select S random points, say $S = 1000$)
3. Use $(\mathbf{R}^*, \mathbf{t}^*, s^*)$ to find matches \mathbf{y}_k for all $n = 1, \dots, S$

$$k = \arg \min_m \|\mathbf{y}_m - s^* \mathbf{R}^* \mathbf{x}_n - \mathbf{t}^*\|^2$$

and collect all corresponding tuple $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^S$

4. Minimize error function (using our least-squares solution)

$$(\mathbf{R}^*, \mathbf{t}^*, s^*) = \arg \min_{\mathbf{R}, s, \mathbf{t}} \sum_{k=1}^S \|\mathbf{y}_k - s \mathbf{R} \mathbf{x}_k - \mathbf{t}\|^2$$

5. Apply transform and repeat steps (2) – (3) until convergence

³P.J. Besl and N.D. McKay. "A Method for Registration of 3-D Shapes". In: *Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256.

Iterative Closest Point (ICP)

Registration without point correspondences

Besl & McKay and Zhang⁴ basically introduce the same ICP algorithm, only that

- Besl & McKay use **all point correspondences**
- Zhang only use correspondences **below a distance threshold** (e.g., using prior knowledge about the data)

⁴Z. Zhang. "Iterative Point Matching for Registration of Free-Form Curves and Surfaces". In: *IJCV* 13.2 (1994), pp. 119–148.

Iterative Closest Point (ICP)

Tools of the trade ...

To start working with point clouds, I would recommend ...

- **Visualization Toolkit (VTK)**; C++/Python



available at: <http://www.vtk.org/>

- **ParaView** (based on VTK) for point cloud visualization (C++/Python)



available at: <http://www.paraview.org>

- **Point Cloud Library (PCL)** for algorithms (C++)



available at: <http://pointclouds.org>

Iterative Closest Point (ICP)

Tools of the trade ...

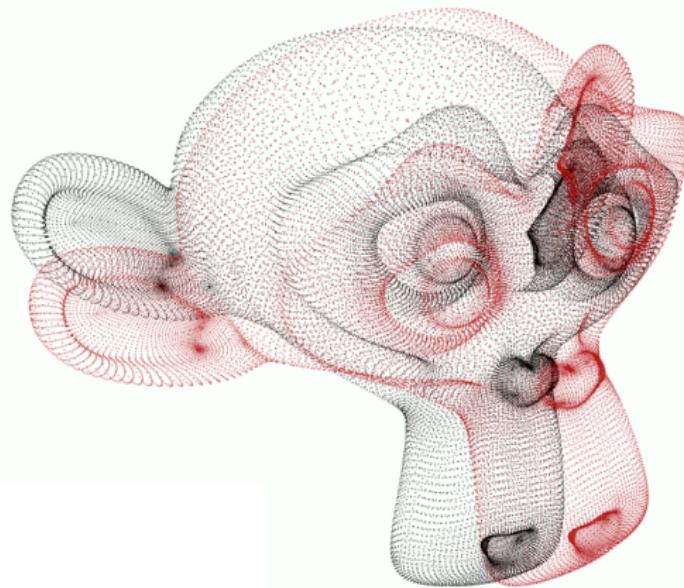
- Open3D; Python



available at: <http://www.open3d.org>

Iterative Closest Point (ICP)

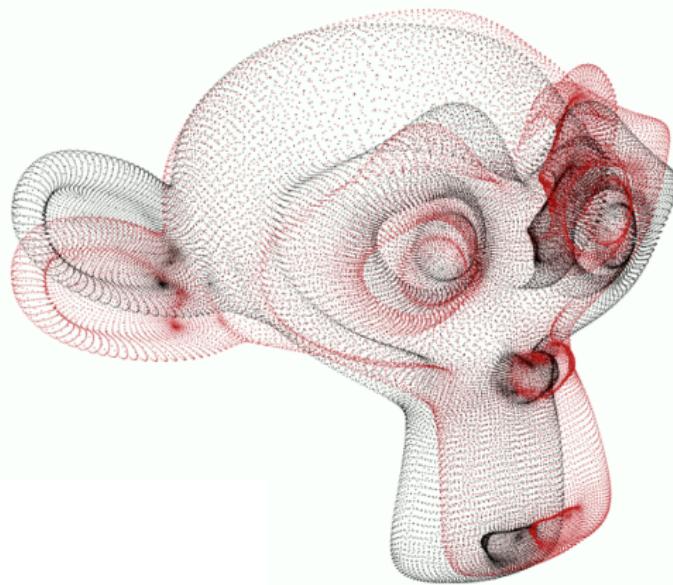
Basic ICP – Example with “Monkey” (using PCL)



Iteration: 0

Iterative Closest Point (ICP)

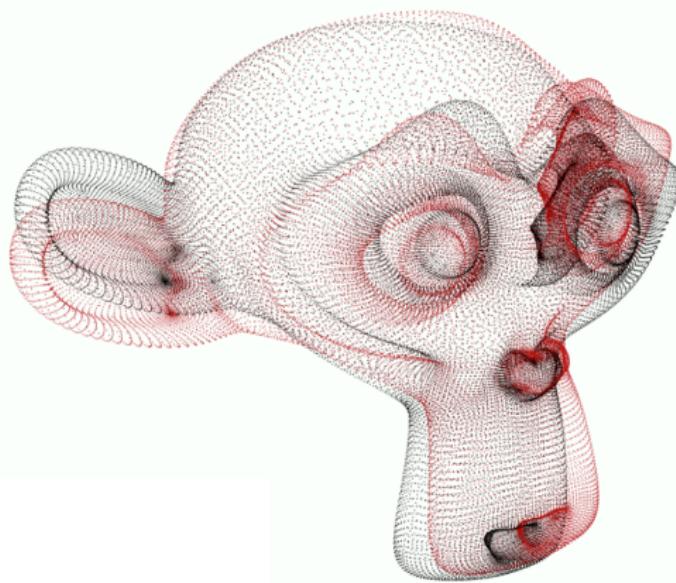
Basic ICP – Example with “Monkey” (using PCL)



Iteration: 3

Iterative Closest Point (ICP)

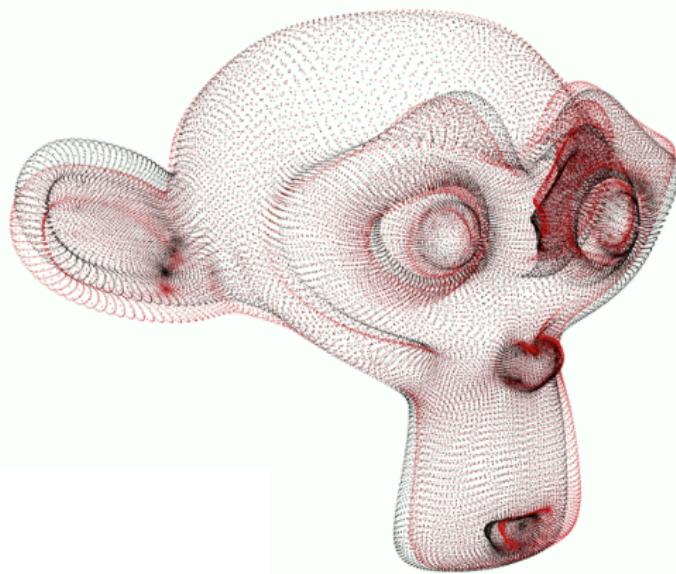
Basic ICP – Example with “Monkey” (using PCL)



Iteration: 5

Iterative Closest Point (ICP)

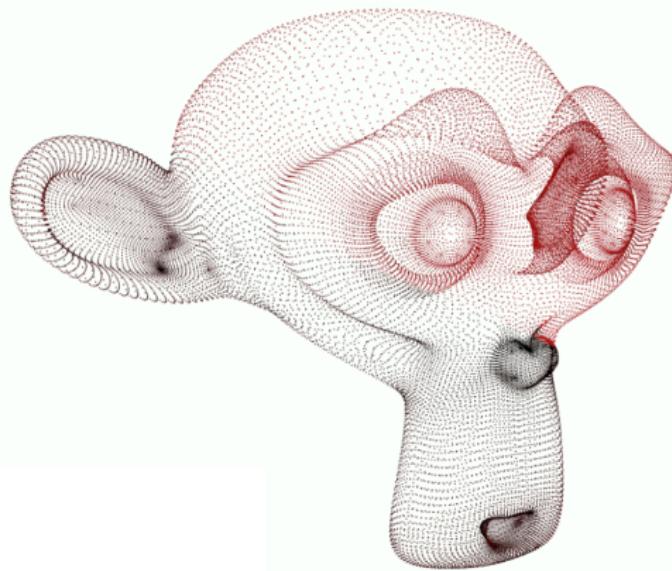
Basic ICP – Example with “Monkey” (using PCL)



Iteration: 9

Iterative Closest Point (ICP)

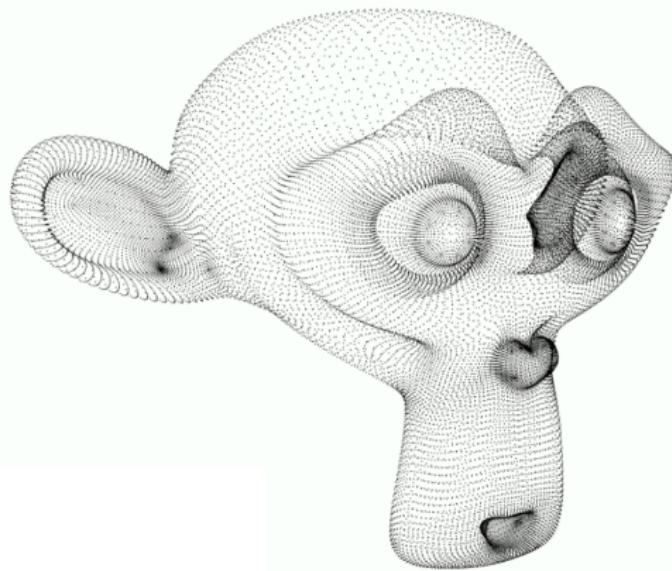
Basic ICP – Example with “Monkey” (using PCL)



Iteration: 25

Iterative Closest Point (ICP)

Basic ICP – Example with “Monkey” (using PCL)

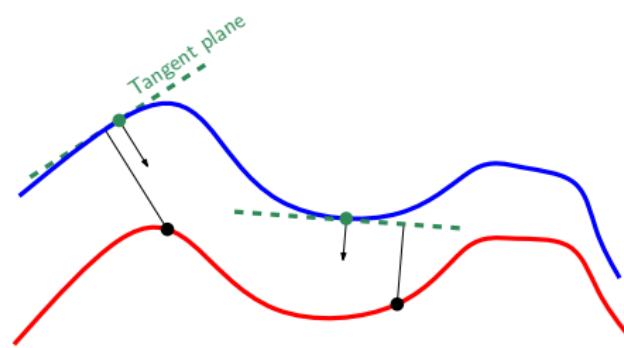


Iteration: Final (Done!)

Iterative Closest Point (ICP)

Variants of ICP with respect to the error metric

One possibility is to replace the point-to-point error metric (as we used before) by a **point-to-plane** error metric⁵



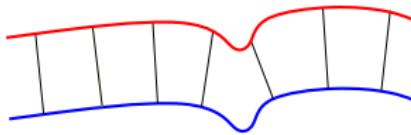
⇒ allows flat regions to “slide” with respect to each other!

⁵ Y. Chen and G. Medioni. "Object Modeling By Registration of Multiple Range Images". In: ICRA. 1991.

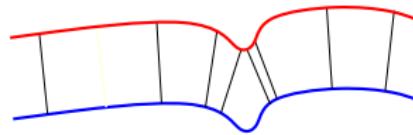
Iterative Closest Point (ICP)

Variants of ICP with respect to the selection of source points

- Use all source points
- Uniform sampling
- Stable sampling⁶
 - Choosing too many points in “featureless regions” is bad
 - Sampling strategy to constrain potentially unstable transformation



Uniform sampling



Stable sampling

We can achieve similar effects as with selection by **weighting**
(however, **pre-processing vs. runtime tradeoff**)

⁶N. Gelfand et al. “Geometrically Stable Sampling for the ICP Algorithm”. In: 3DIM. 2003.

Iterative Closest Point (ICP)

Variants of ICP with respect to the matching of source points

What about speed? Finding the closest point (in a cloud with N points) is the most expensive stage in the ICP algorithm.

- Brute-force $O(N)$
- **Better** to use spatial data structures (e.g., k-D tree)

In general, closest points can be suboptimal as corresponding points! **Alternatives** are ...

- matching of colors, e.g., Godin et al.⁷
- matching of normals, e.g., Pulli⁸
- and many others (e.g., consider curvature, local features, etc.)

⁷ G. Godin, M. Rioux, and R. Baribeau. "Three-dimensional registration using range and intensity information". In: SPIE Videometric III. 1994.

⁸ K. Pulli. "Multiview registration for large data sets". In: 3DIM. 1999.

k-D Trees

Principle

k-D trees are

- spatial data structures introduced by Bentley⁹
- impose a spatial decomposition of *k*-dimensional points that
 - allows efficient search on orthogonal range queries
 - allows efficient nearest neighbor searches

Remark

both properties are achieved by “pruning” the search space!

⁹J. Bentley. “Multidimensional binary search trees used for associative searching”. In: *Commun. ACM* 18 (1975), 509?517.

k-D Trees

Building a (naive) *k*-D tree

Data is inserted as in the case of a regular binary tree, **only that**

k-D Trees

Building a (naive) *k*-D tree

- Data is inserted as in the case of a regular binary tree, **only that**
- the **key** that we use changes at each level

k-D Trees

Building a (naive) *k*-D tree

Data is inserted as in the case of a regular binary tree, **only that**

- the **key** that we use changes at each level
- at each level i , the key is $(i \bmod k)$ where k denotes dimensionality

Example: $a = (1, 2)$, $b = (3, 4)$, $c = (5, 6)$, $d = (7, 8)$

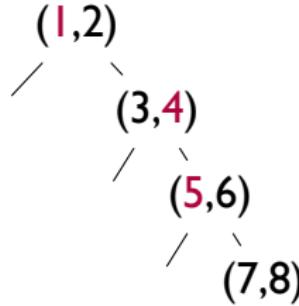
k-D Trees

Building a (naive) *k*-D tree

Data is inserted as in the case of a regular binary tree, **only that**

- the **key** that we use changes at each level
- at each level i , the key is $(i \bmod k)$ where k denotes dimensionality

Example: $a = (1, 2)$, $b = (3, 4)$, $c = (5, 6)$, $d = (7, 8)$

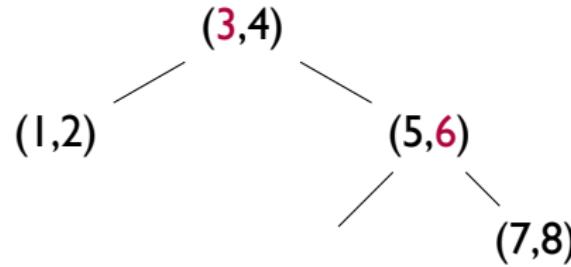


Insertion order: “abcd”

k-D Trees

Building a (naive) *k*-D tree

Same example: $a = (1, 2)$, $b = (3, 4)$, $c = (5, 6)$, $d = (7, 8)$



Insertion order: “bcad”

k-D Trees

Building a (median) *k*-D tree (see Friedman et al.¹⁰)

We are given points $\{\mathbf{p}_i\}_{i=1}^N$, $\mathbf{p}_i \in \mathbb{R}^k$.

Algorithm (Building a median *k*-D tree)

- I. Select split (i.e., “cutting dimension”) coordinate either by
 - I.1 at level i , the key is $(i \bmod k)$ – cycle through coordinate axes
 - I.2 find axis with maximal spread and choose this as key at level i
2. Split data according to the median m_c at the key coordinate c :
 - o left subtree: $\{\mathbf{p}_i : p_{ic} \leq m_c\}$
 - o right subtree: $\{\mathbf{p}_i : p_{ic} > m_c\}$

¹⁰J. Friedman, J. Bentley, and R. Finkel. “An algorithm for finding best matches in logarithmic expected time”. In: ACM Transactions on Mathematical Software 3 (1977), pp. 209–226.

k-D Trees

Median *k*-D tree – Properties

Properties:

- Size: $O(N)$
- Depth: $O(\log N)$
- Construction time: $O(N \log N)$
(Note: the median can be found in $O(N)^{11}$)

¹¹ M. Blum et al. "Time bounds for selection". In: *Journal of Computer and System Sciences* 7.4 (1973), pp. 448–461.

k-D Trees

Some remarks ...

Remarks on the choosing “cutting dimension”

- cycle through (using **level mod dimension**)
 - might produce skinny (elongated) cells
→ bad impact on query times

k-D Trees

Some remarks ...

Remarks on the choosing “cutting dimension”

- cycle through (using **level mod dimension**)
 - might produce skinny (elongated) cells
→ bad impact on query times
- split along dimension of greatest “spread”
 - e.g., use difference between min. and max. value as statistic
 - Bentley refers to such a tree as an *optimized k-D tree*

Remarks on the choosing “cutting value”

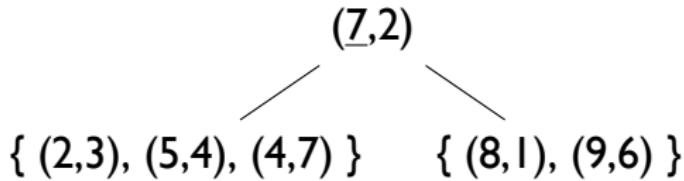
- Choosing the median allows us to guarantee $O(\log N)$ height.

Example from:

http://en.wikipedia.org/wiki/K-d_tree

Data: $(7, 2), (5, 4), (2, 3), (4, 7), (9, 6), (8, 1)$

(our strategy: cycle dimension & split by the upper median $x_{\lceil \frac{n+1}{2} \rceil}$)

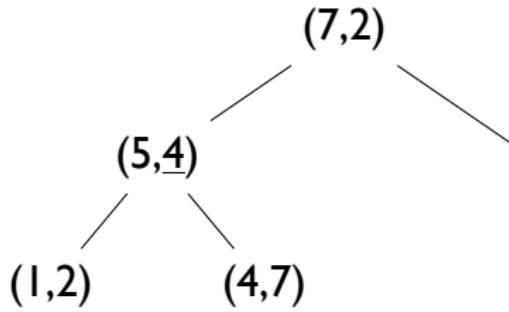


Example from:

http://en.wikipedia.org/wiki/K-d_tree

Data: $(7, 2), (5, 4), (2, 3), (4, 7), (9, 6), (8, 1)$

(our strategy: cycle dimension & split by the upper median $x_{\lceil \frac{n+1}{2} \rceil}$)

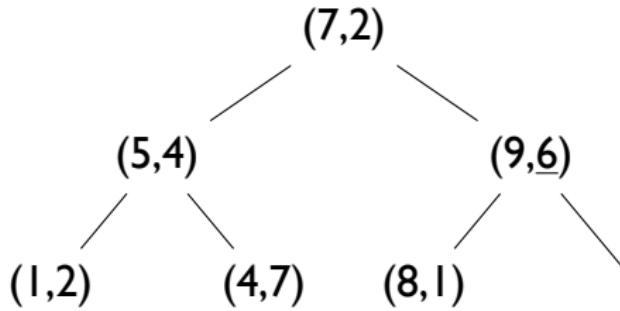


Example from:

http://en.wikipedia.org/wiki/K-d_tree

Data: $(7, 2), (5, 4), (2, 3), (4, 7), (9, \underline{6}), (8, 1)$

(our strategy: cycle dimension & split by the upper median $x_{\lceil \frac{n+1}{2} \rceil}$)



k-D Trees

Nearest neighbor search with a *k*-D tree

Problem: Given a point q , find the point p_i that is closest to q

k-D Trees

Nearest neighbor search with a *k*-D tree

Problem: Given a point q , find the point p_i that is closest to q

First strategy: Find cell that would contain q and return point in it.

k-D Trees

Nearest neighbor search with a *k*-D tree

Problem: Given a point q , find the point p_i that is closest to q

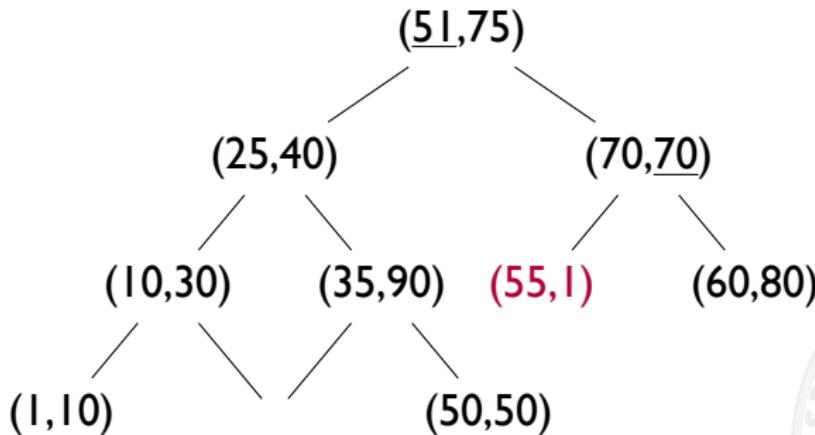
First strategy: Find cell that would contain q and return point in it.
DOES NOT WORK!

k-D Trees

Nearest neighbor search with a *k*-D tree

Problem: Given a point q , find the point p_i that is closest to q

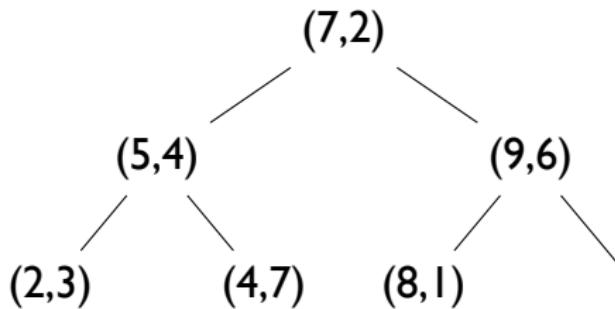
First strategy: Find cell that would contain q and return point in it.
DOES NOT WORK! In the following example, set $q = (52, 52)$



k-D Trees

Nearest neighbor search with a *k*-D tree

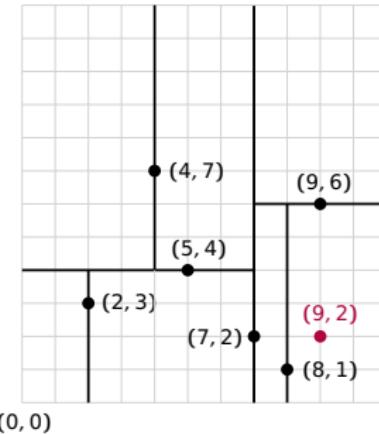
Let's look at our Wikipedia example (median *k*-D tree):



Task: Search for the nearest-neighbor to $\mathbf{q} = (9, 2)$

k-D Trees

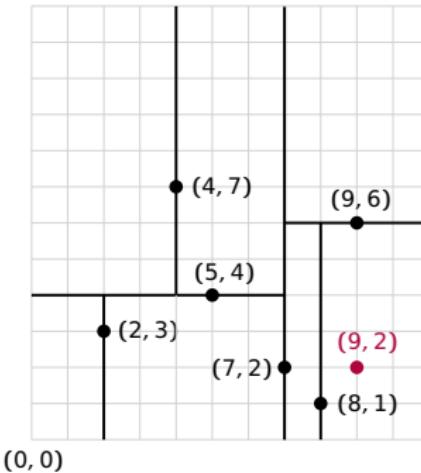
Nearest neighbor search with a *k*-D tree



What's the basic strategy?

- Store the closest point found so far
- Prune subtrees once their bounding boxes cannot contain q
- Search subtrees in most promising order

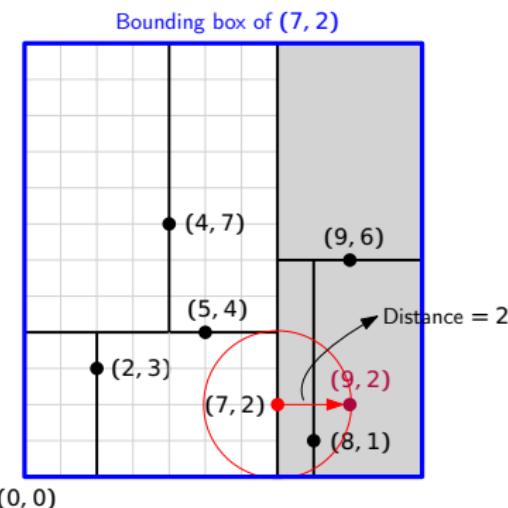
Example



(Symbol legend: → BB dist., → point-to-point dist., ● current point)

Example

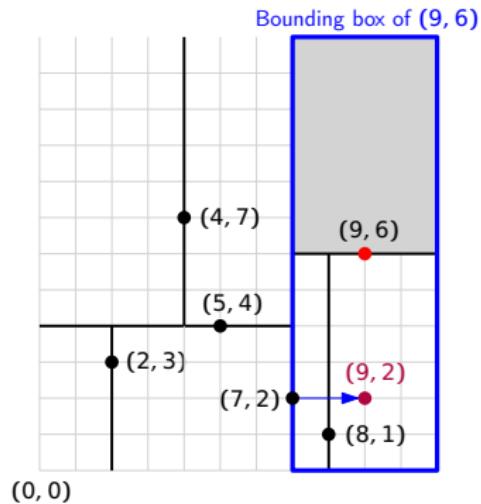
Init. distance at ∞ . Distance of q to BB of $(7, 2)$ is $< \infty$. Store $(7, 2)$ as best point so far. Next, since $9 > 7 \rightarrow$ goto right subtree



(Symbol legend: → BB dist., → point-to-point dist., ● current point)

Example

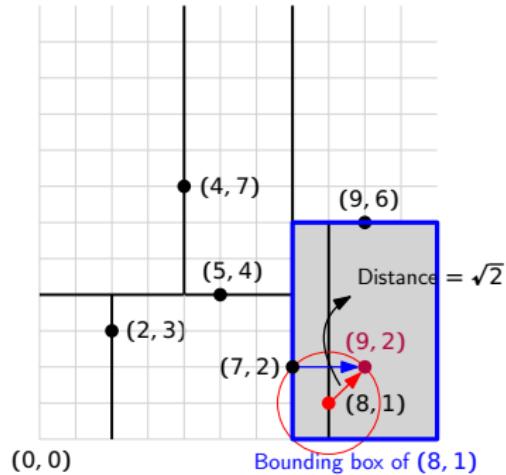
At $(9, 6)$, distance of q to BB of $(9, 6)$ not > 2 , hence, continue;
Distance of $(9, 6)$ to q not best. Next, $2 < 6 \rightarrow$ goto left subtree



(Symbol legend: → BB dist., → point-to-point dist., ● current point)

Example

We are now at the left subtree of $(9, 6)$; Distance of q to BB of $(8, 1)$ not > 2 , hence continue; Distance to $(8, 1)$ is now best!

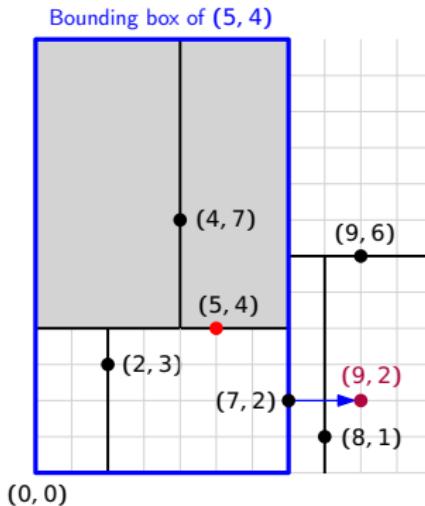


(Also, no need to traverse further since $(8, 1)$ is leaf)

(Symbol legend: → BB dist., → point-to-point dist., ● current point)

Example

We are now in the left subtree of (7, 2) at (5, 4); Now, distance of q to BB of (5, 4) > $\sqrt{2}$; hence, return; **DONE!**



(Symbol legend: → BB dist., → point-to-point dist., ● current point)

More on k -D trees

Curse of dimensionality

Consider n points uniformly in $[0, 1]^d$ and a space partitioning that splits each box in the middle (round robin on axes).

Assume each box is split t times (tree depth t) → each axis is split t/d times (say, t/d is integer) with box dimensions

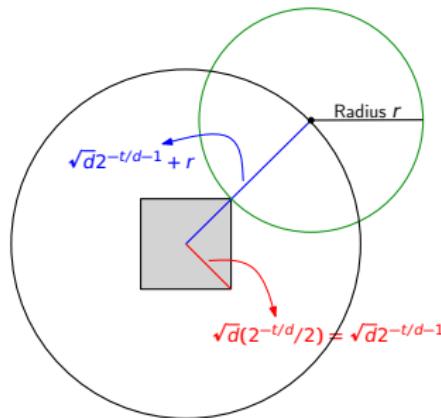
$$[2^{-t/d}, \dots, 2^{-t/d}]$$

What is the probability that ball of radius r will intersect this box?

More on k -D trees

Curse of dimensionality

The length of an r -agonal of an hypercube is $a\sqrt{d}$ (where a denotes the side-length). Hence, we have



The probability that the box intersects a random query sphere is at most the volume of a sphere of radius $\sqrt{d}2^{-t/d-1} + r$.

More on k -D trees

Curse of dimensionality

Assume that $\sqrt{d}2^{-t/d-1} \leq r$.

More on k -D trees

Curse of dimensionality

Assume that $\sqrt{d}2^{-t/d-1} \leq r$.

Hence, the probability is bounded by the volume of a “bounding ball” of radius $2r$. The volume of an \mathbb{R}^d -ball of this radius is

$$V_d(r) = \frac{\pi^{d/2}(2r)^d}{(d/2)!} \text{ (if } d \text{ even)}$$

Hence, the dependency of the volume is in $O(r^d)$.

More on k -D trees

Curse of dimensionality

Assume that $\sqrt{d}2^{-t/d-1} \leq r$.

Hence, the probability is bounded by the volume of a “bounding ball” of radius $2r$. The volume of an \mathbb{R}^d -ball of this radius is

$$V_d(r) = \frac{\pi^{d/2}(2r)^d}{(d/2)!} \text{ (if } d \text{ even)}$$

Hence, the dependency of the volume is in $O(r^d)$.

Consequence

As a result, we expect to inspect only a $O(r^d)$ fraction of boxes and, since $r \leq 1$, this leads to significant speedups!

More on k -D trees

Curse of dimensionality

But, we assumed that $\sqrt{d}2^{-t/d-1} \leq r$. The question is, “how does this behave when d is large”?

- Let's consider $t \approx \log(n)$ (which we have in a balanced tree), which gives boxes of volume $\approx 1/n^{12}$

¹²since $(2 - \log(n)/sd)^d$ (i.e., box vol.) reduces to $2^{-\log(n)}$ and by letting log be approximated by \log_2 , we get $\approx 1/n$

More on k -D trees

Curse of dimensionality

But, we assumed that $\sqrt{d}2^{-t/d-1} \leq r$. The question is, “how does this behave when d is large”?

- Let's consider $t \approx \log(n)$ (which we have in a balanced tree), which gives boxes of volume $\approx 1/n^{12}$
- An analysis shows that requiring $\sqrt{d}2^{-\log(n)/d-1} \leq r$ requires $n \geq 2^d$, i.e., exponential in d !

¹²since $(2^{-\log(n)/sd})^d$ (i.e., box vol.) reduces to $2^{-\log(n)}$ and by letting log be approximated by \log_2 , we get $\approx 1/n$

More on k -D trees

Curse of dimensionality

Summary

We only gain if the number of points n is exponential in the number of dimensions d .

What can we do about that problem?

- Locality-Sensitive-Hashing (LSH)¹³
- Randomized k -D trees¹⁴
- Hierarchical k -means trees (in FLANN)¹⁵

¹³A. Adoni and P. Indyk. "Near-Optimal Hashing Algorithms for Near Neighbor Problem in High Dimensions". In: FOCS. 2006.

¹⁴C. Silpa-Anan and R. Hartley. "Optimised KD-trees for fast image descriptor matching". In: CVPR. 2008.

¹⁵M. Muja and D.G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration". In: VISAPP. 2009.