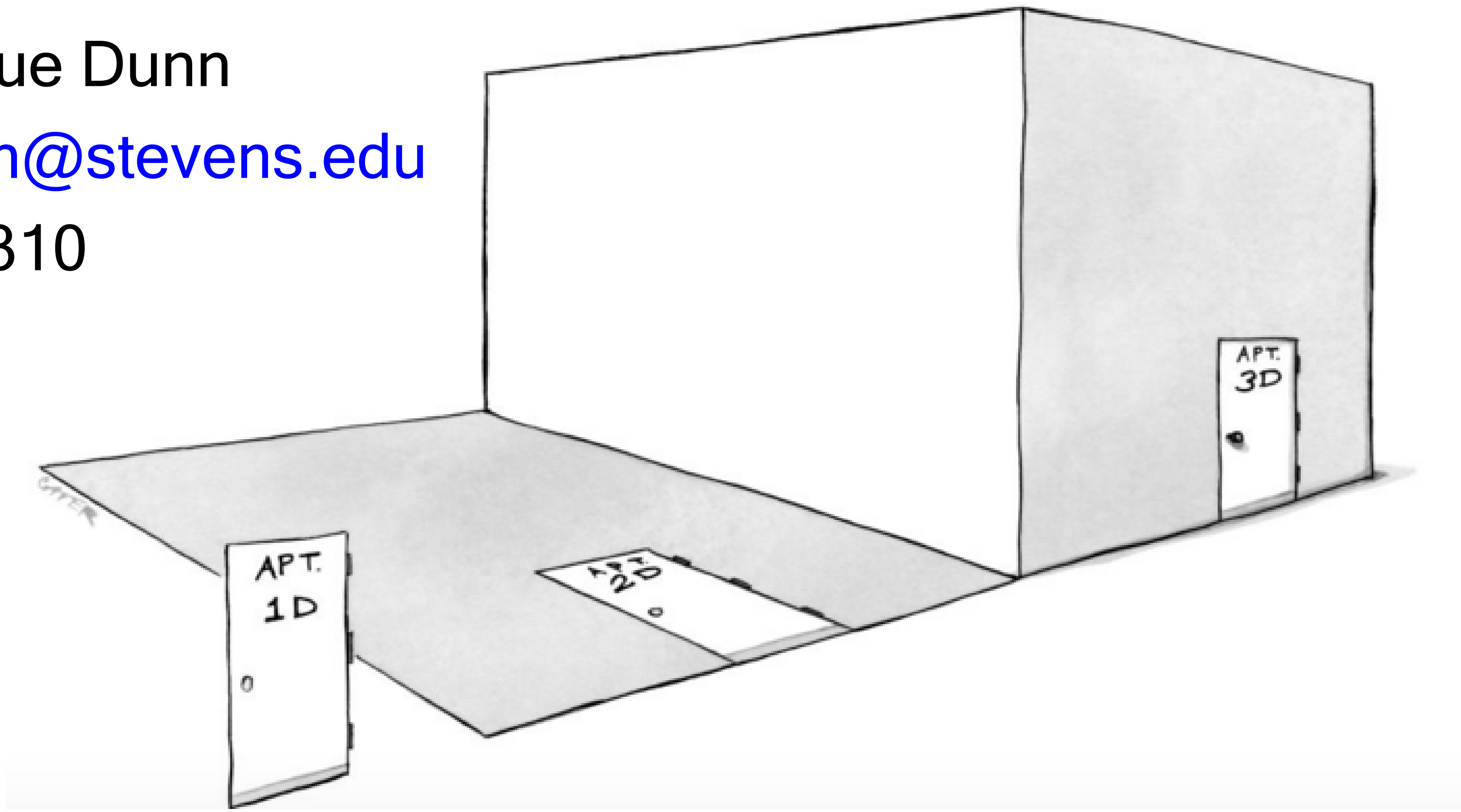
CS 532: 3D Computer Vision
Lecture 11

Enrique Dunn
edunn@stevens.edu
Lieb 310

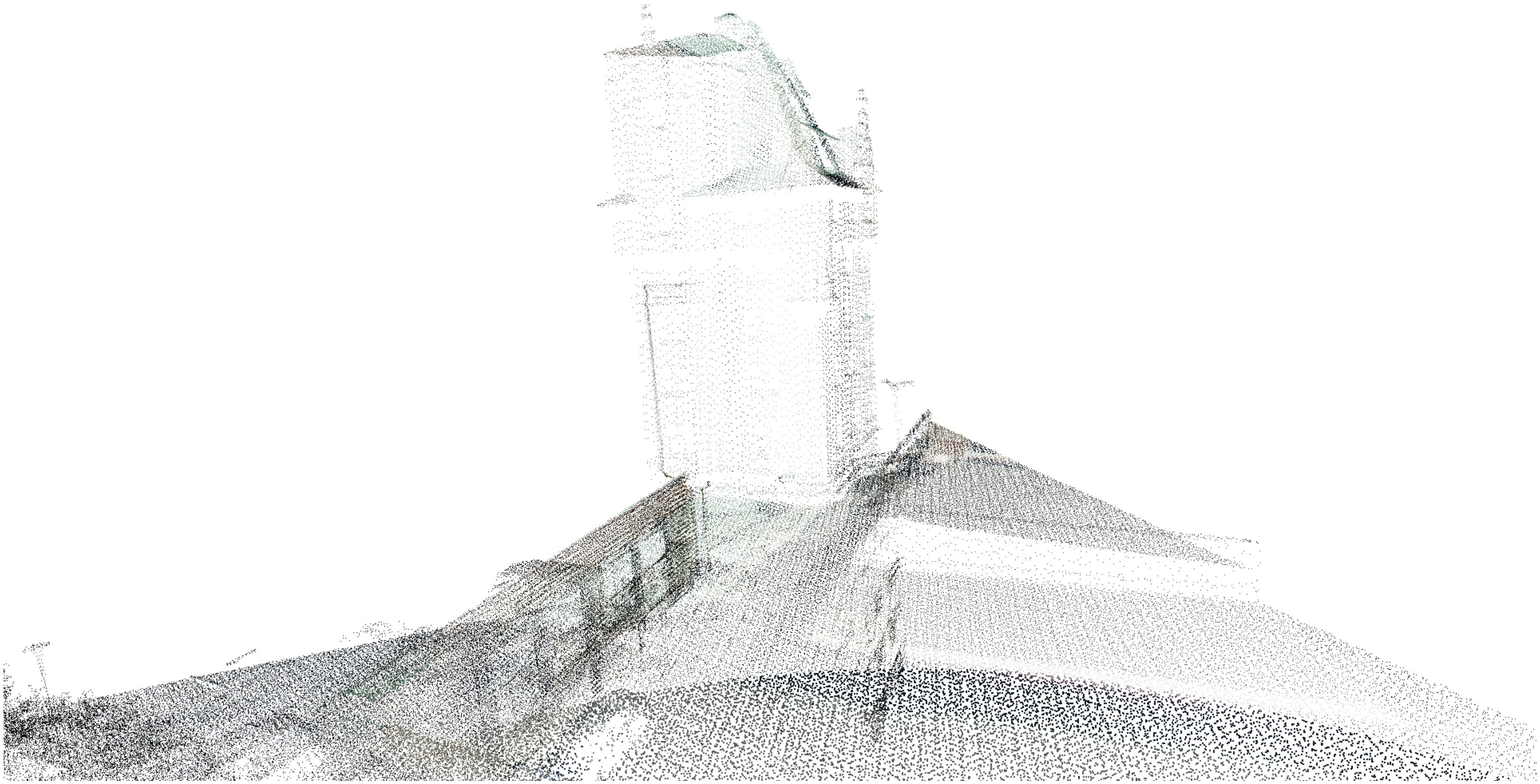


Lecture Outline

- Unorganized point clouds
  - Range queries and kd-trees
- Descriptors for 3D point clouds

- David M. Mount, CMSC 754: Computational Geometry lecture notes, Department of Computer Science, University of Maryland, Spring 2012
  - Lecture 16

Point Clouds



Unorganized Point Clouds

- In many cases, data come as unorganized point clouds
  - Not trivial to generate meshes even given a 2D reference frame (see Homework 6.2)
  - Even more complicated to fuse multiple viewpoint-based meshes
  - Some sensors (moving LIDAR) measure only points

What can be done?

- While mesh generation is possible, it is not guaranteed to be correct
  - Even watertight manifolds can be generated using Poisson surface reconstruction (see previous notes)
  - It is often hard to discriminate forward and backward facing points
  - It is hard to determine whether there are gaps between points

What can be done?

- Process the data by assuming possible connections between points
  - I.e. assuming nearby points belong to the same smooth surface
- Do not make hard decisions

Key Parameter: Scale

- Trade-off between fidelity to data and robustness to noise
- If analysis is performed at small scale, details (high curvature) can be preserved
- If data are noisy, high curvature typically corresponds to noise…

Nearest Neighbors

- Number one enabling technology for working with unorganized point clouds: Nearest Neighbor Search
- Algorithms cannot be quadratic
- It can be shown that kd-trees can be generated in O(n logn) time
- Queries take O(logn) per point
  - k queries: find k nearest neighbors of query point
  - epsilon queries: find neighbors within an ε-ball centered at the query point

Tools

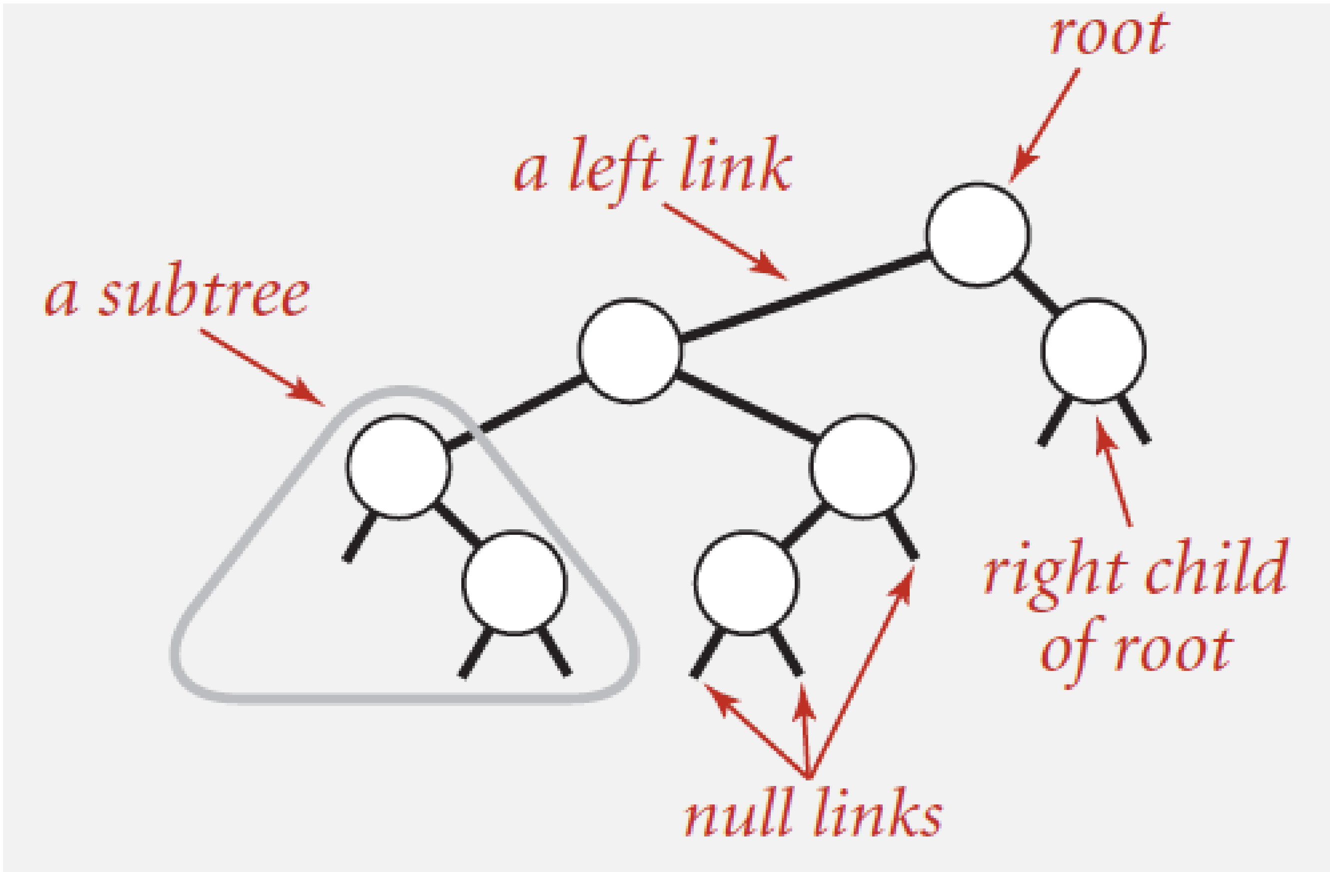
- knnsearch() in Matlab
  - http://www.mathworks.com/help/stats/knnsearch.html
  - Do multiple queries with one call
- FLANN library
  - http://www.cs.ubc.ca/research/flann/
  - Bindings for C/C++, Matlab and python, part of PCL
- CvKNearest in OpenCV (which has bindings for C/C++, Python and Java)
- KDTree in scikit

- All these are not limited to 3D data

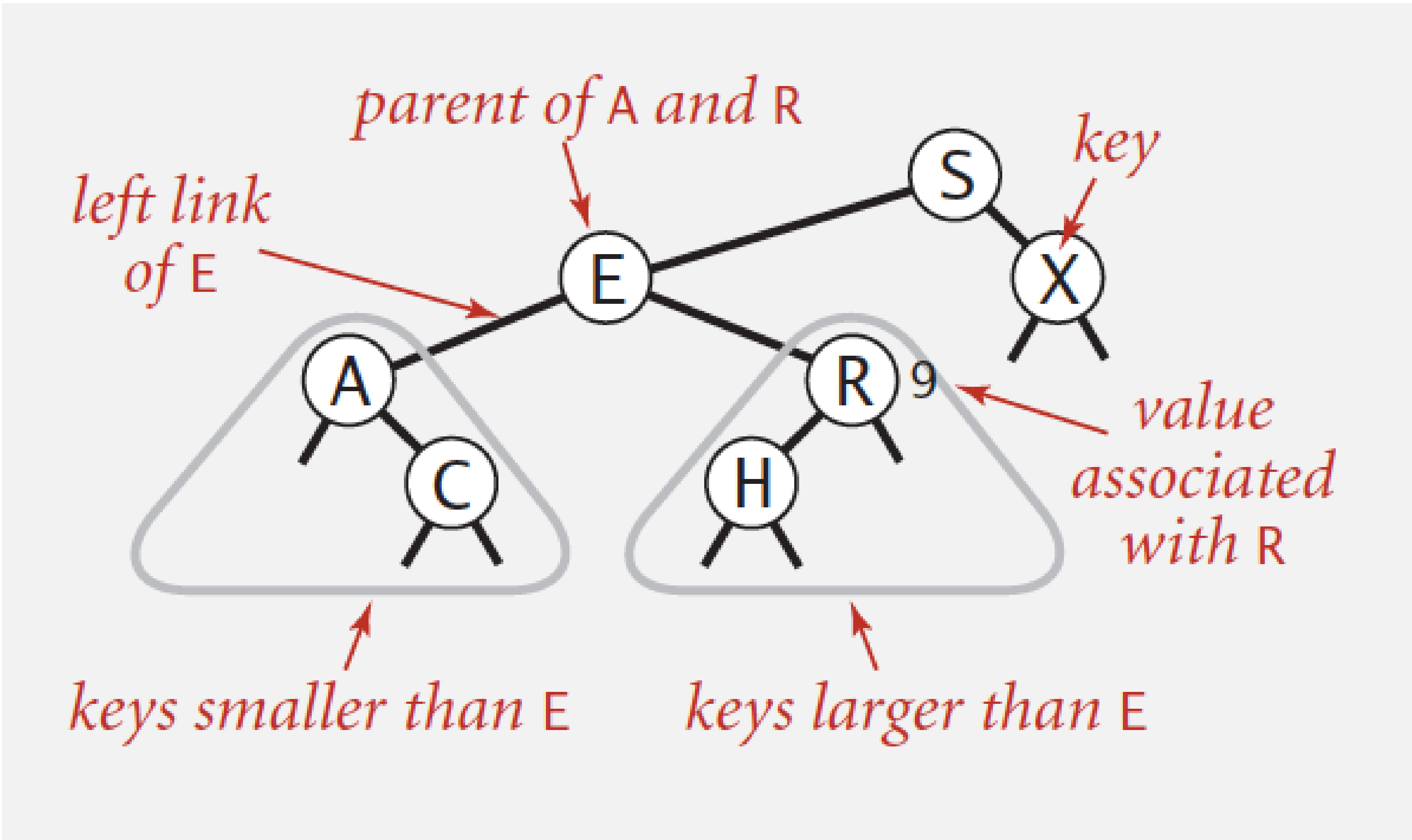
1D Range Queries

- 1D range query problem: Preprocess a set of n points on the real line such that the ones inside a 1D query range (interval) can be reported fast
- The points $p_1, \ldots, p_n$ are known beforehand, the query [x, x’] only later
- A solution to a query problem is a data structure description, a query algorithm, and a construction algorithm

Binary Search Tree

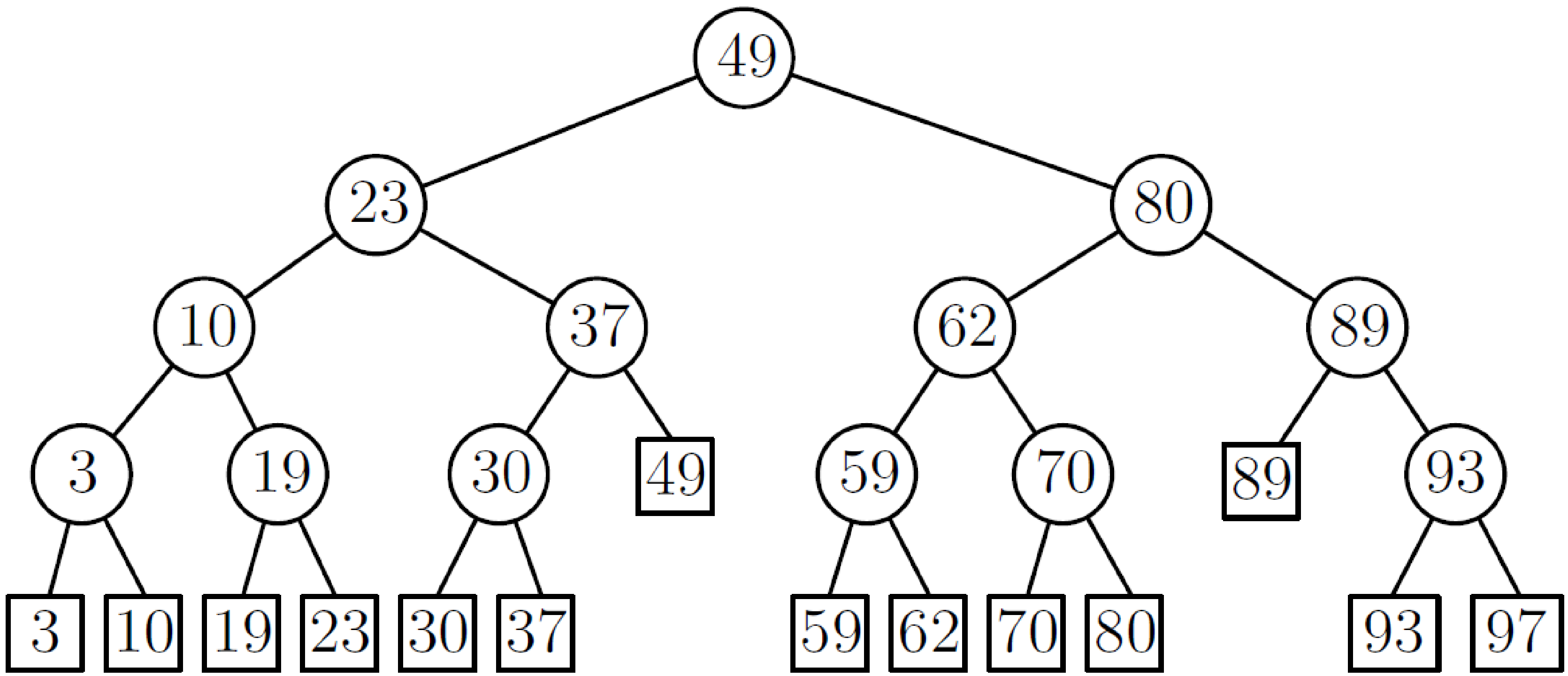


Binary Search Tree



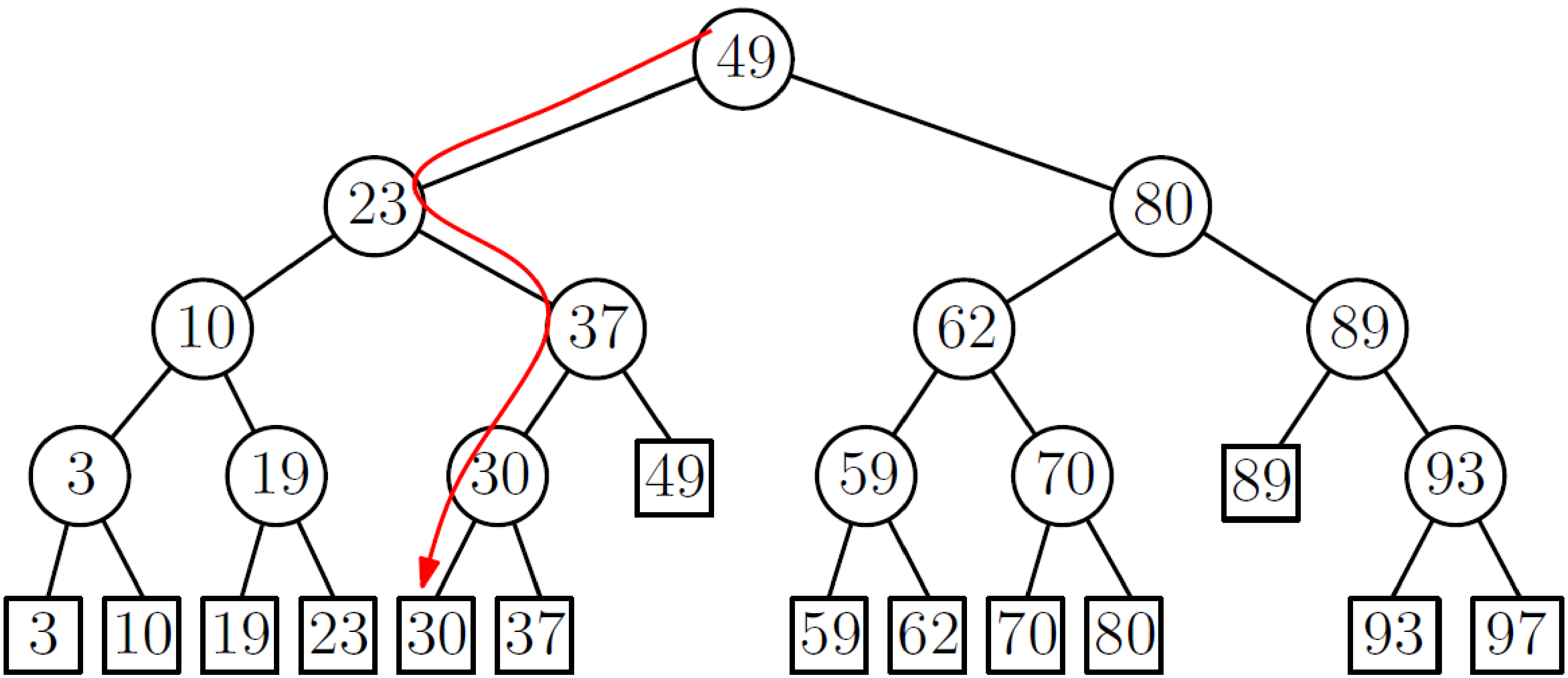
Balanced Binary Search Trees

- A balanced binary search tree with the points in the leaves



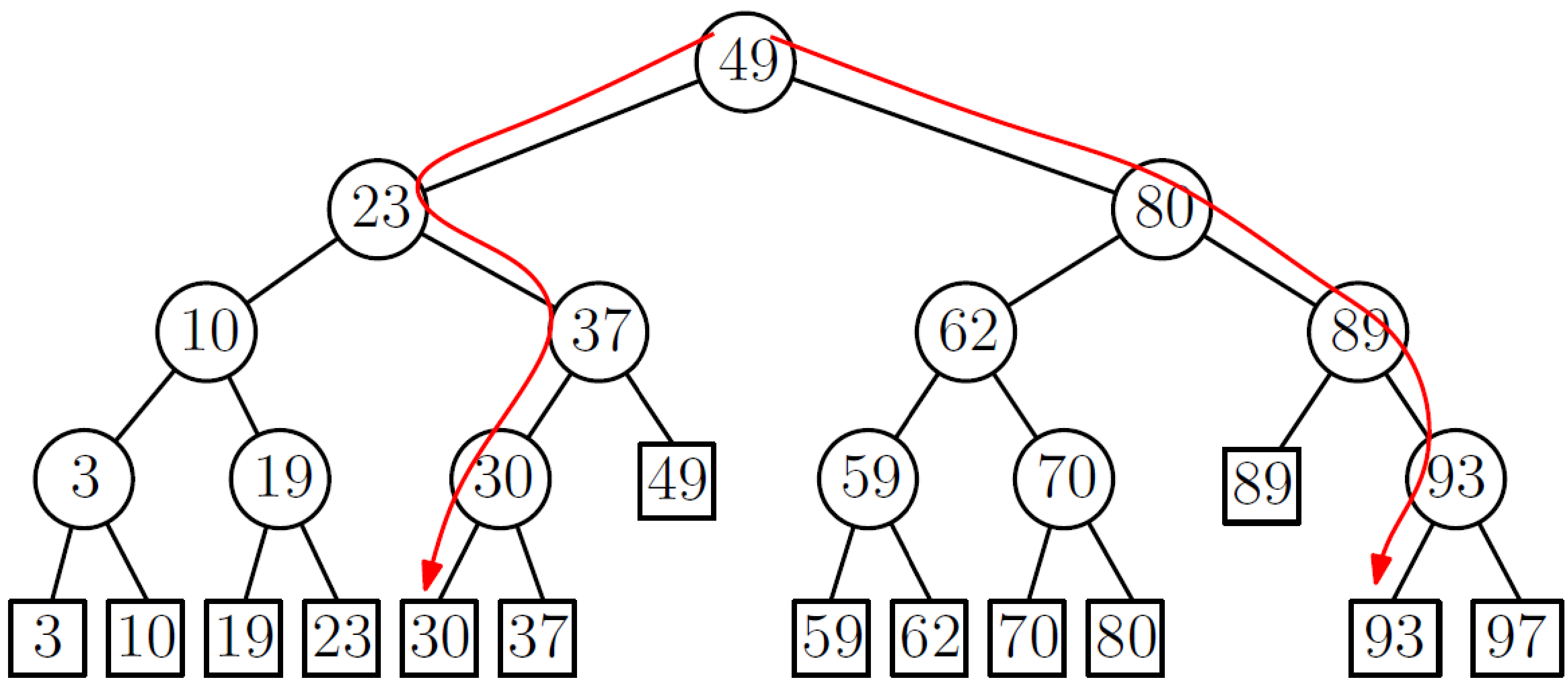
Balanced Binary Search Trees

- The search path for 25



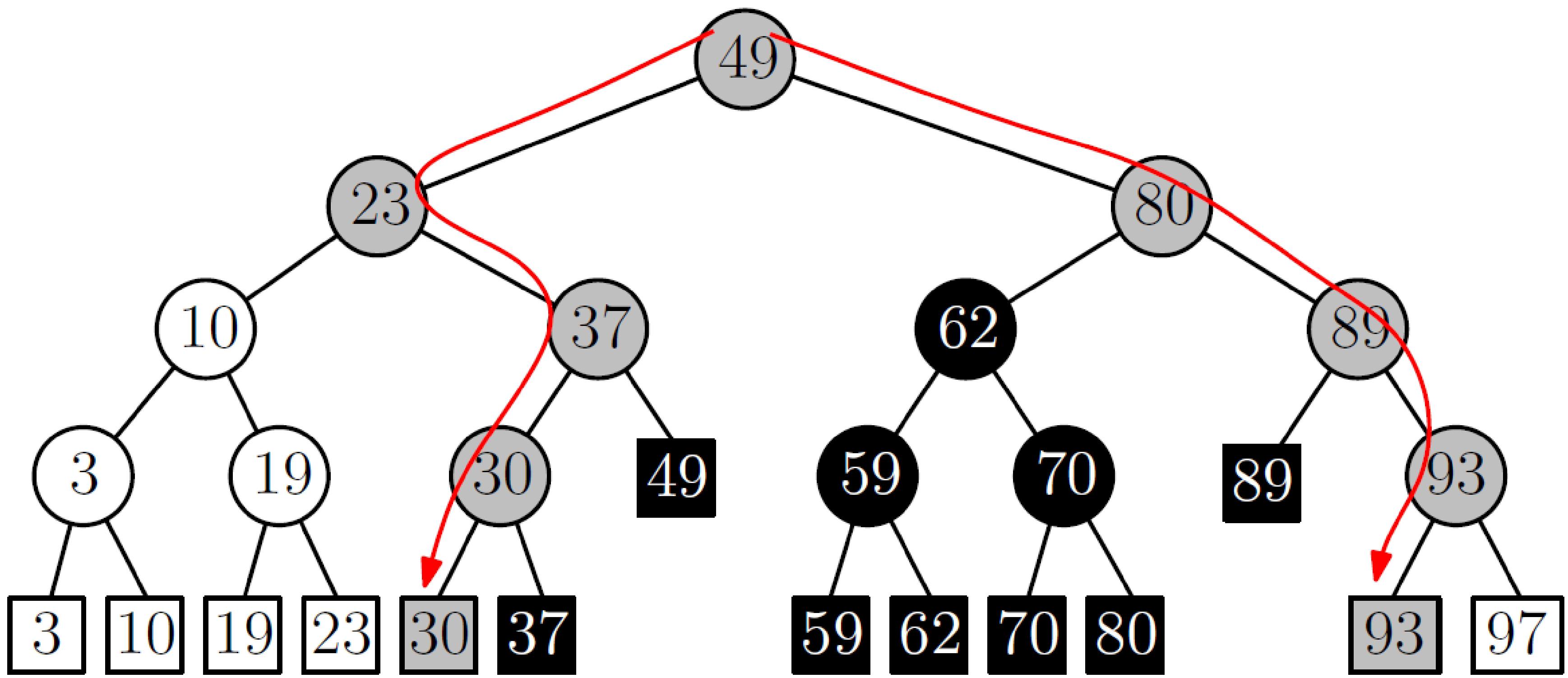
Balanced Binary Search Trees

- The search paths for 25 and 90



Example 1D Range Query

- A 1D range query with [25, 90]



Node Types for a Query

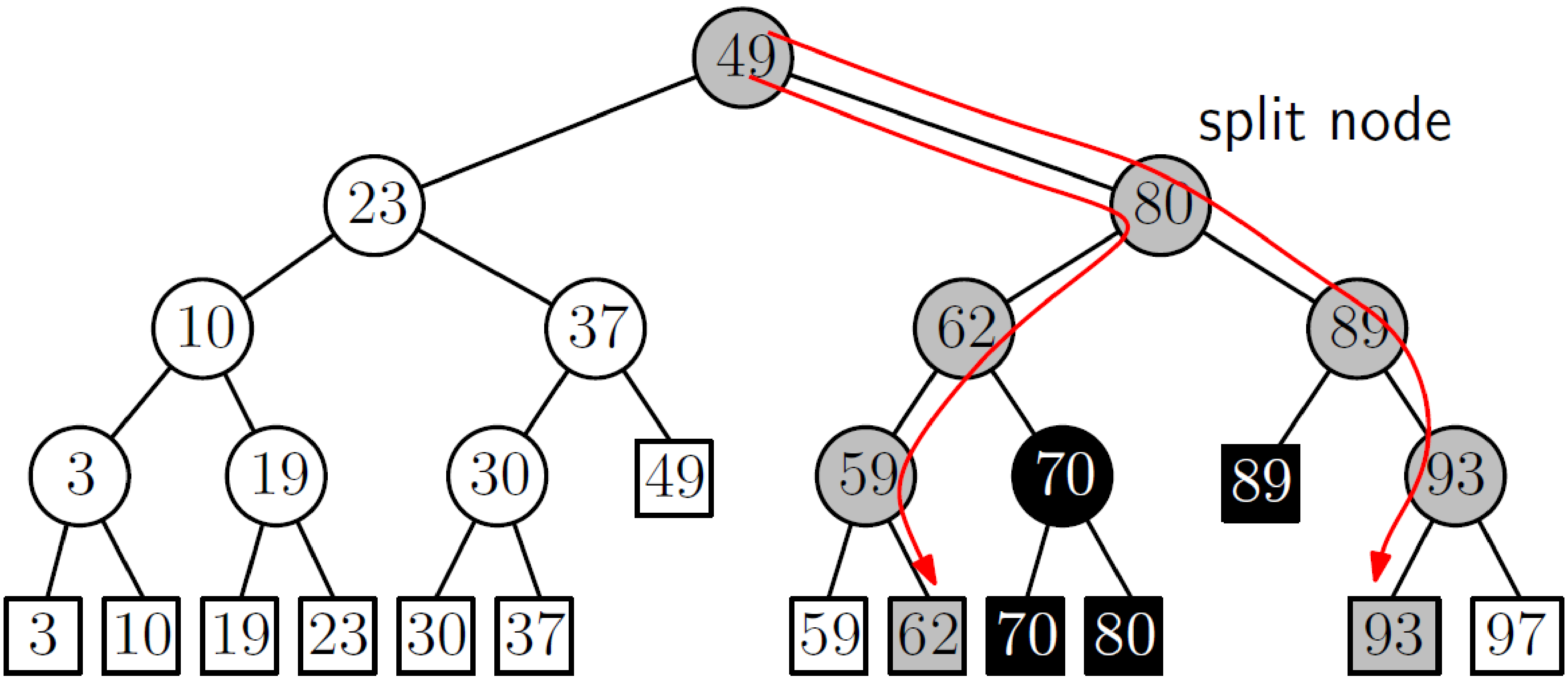
- Three types of nodes for a given query:
  - White nodes: never visited by the query
  - Grey nodes: visited by the query, unclear if they lead to output
  - Black nodes: visited by the query, whole subtree is output

Node Types for a Query

- The query algorithm comes down to what we do at each type of node
- Grey nodes: use query range to decide how to proceed: to not visit a subtree (pruning), to report a complete subtree, or just continue
- Black nodes: traverse and enumerate all points in the leaves

Example 1D Range Query

- A 1D range query with [61, 90]



1D Range Query Algorithm

Algorithm $\textsc{1DRangeQuery}(\mathcal{T}, [x : x'])$

$\nu_{\text{split}} \leftarrow \textsc{FindSplitNode}(\mathcal{T}, x, x')$

if ν_{split} is a leaf

  then Check if the point in ν_{split} must be reported.

  else $\nu \leftarrow lc(\nu_{\text{split}})$

    while ν is not a leaf

      do if $x \leq x_\nu$

        then $\textsc{ReportSubtree}(rc(\nu))$

          $\nu \leftarrow lc(\nu)$

        else $\nu \leftarrow rc(\nu)$

    Check if the point stored in ν must be reported.

    $\nu \leftarrow rc(\nu_{\text{split}})$

    Similarly, follow the path to x', and ...

Query Time Analysis

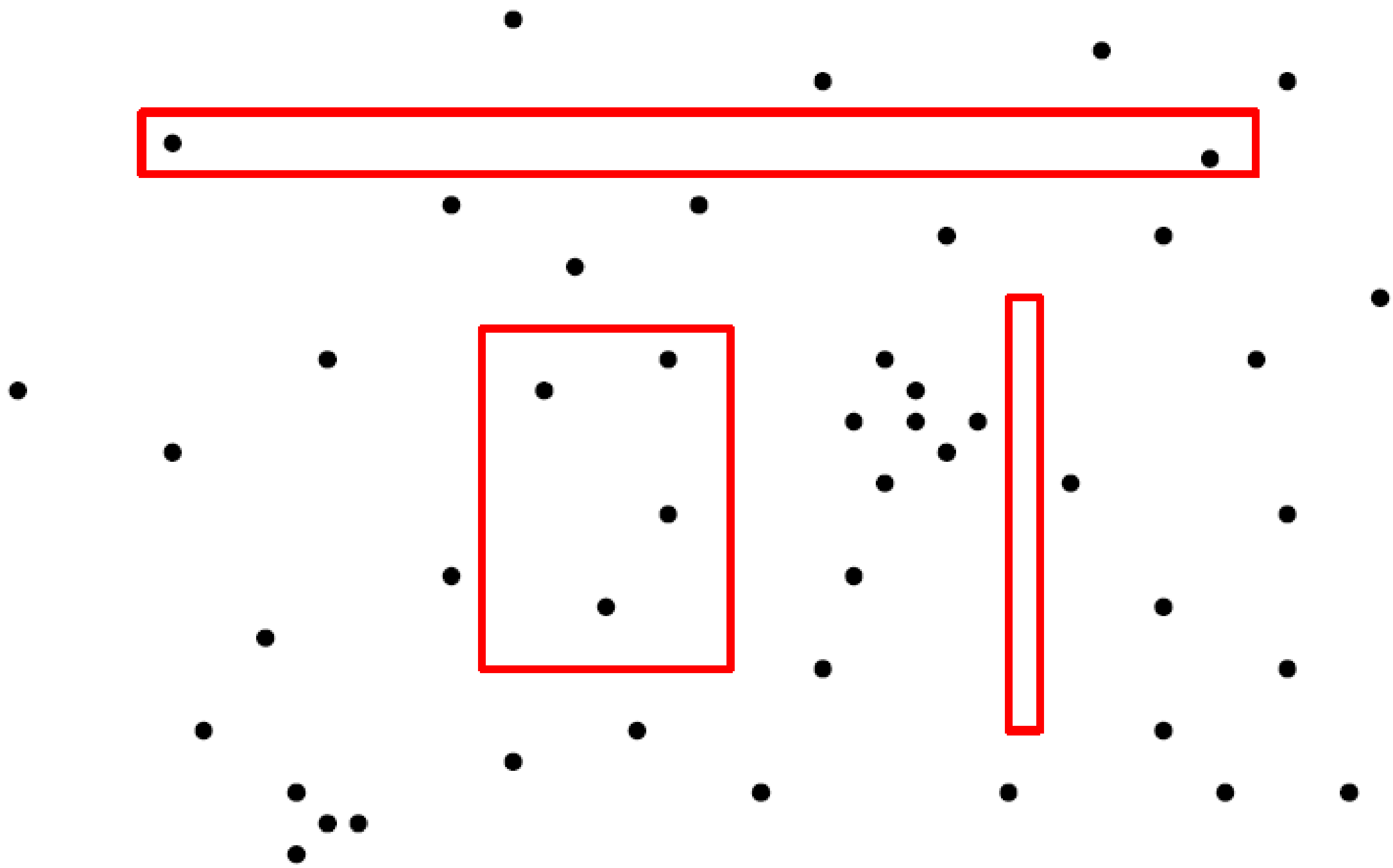
- The efficiency analysis is based on counting the numbers of nodes visited for each type
  - White nodes: never visited by the query; no time spent
  - Grey nodes: visited by the query, unclear if they lead to output; time complexity depends on n
  - Black nodes: visited by the query, whole subtree is output; time complexity depends on k, the output size

Query Time Analysis

- Grey nodes: they occur on only two paths in the tree, and since the tree is balanced, its depth is O(logn)
- Black nodes: a (sub)tree with m leaves has m-1 internal nodes; traversal visits O(m) nodes and finds m points for the output
- The time spent at each node is O(1)

=>O(logn+k) query time

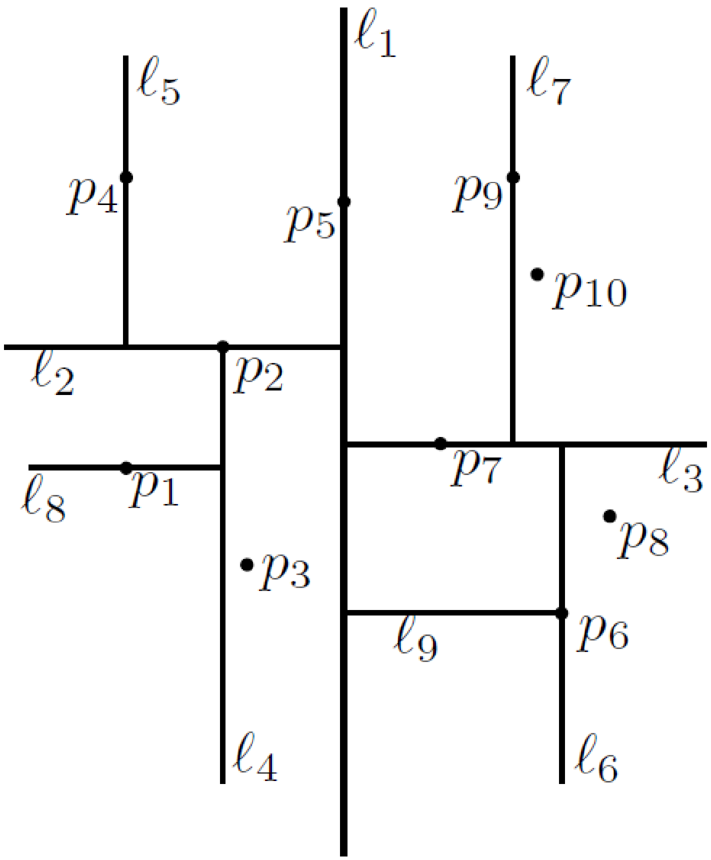
Range Queries in 2D

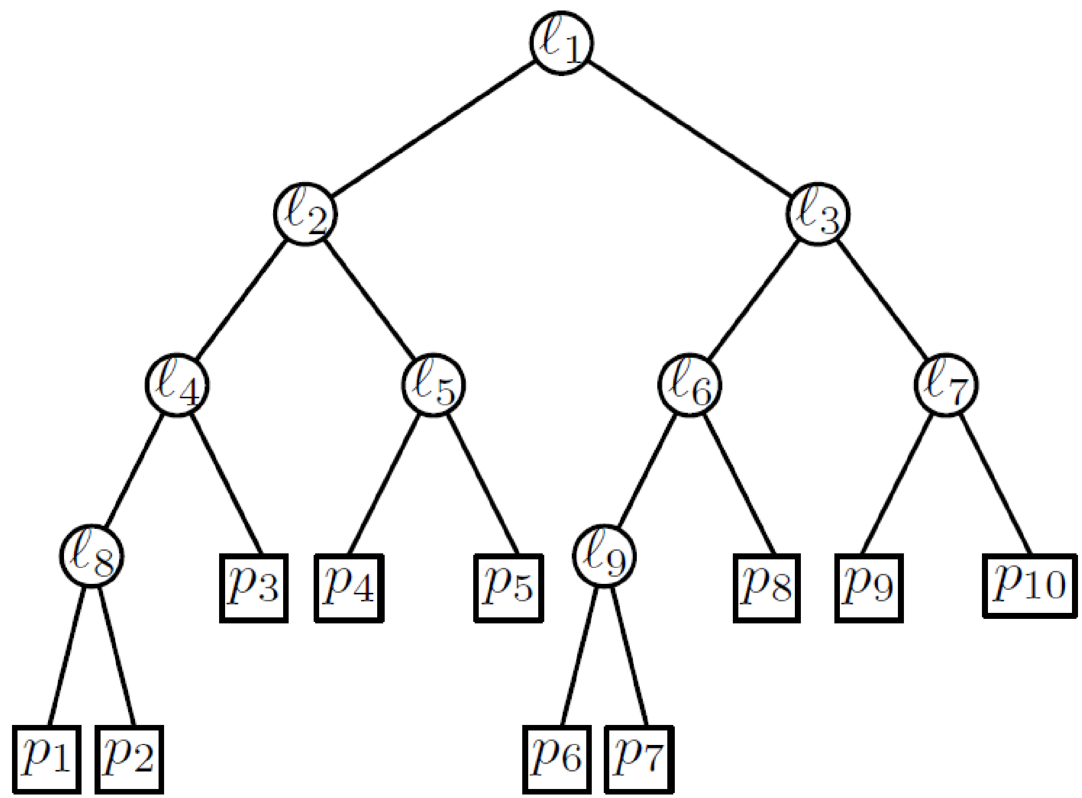


Kd-Trees

- Kd-trees, the idea: Split the point set alternatingly by x-coordinate and by y-coordinate
- Split by x-coordinate: split by a vertical line that has half the points to its left or on it, and half to its right
- Split by y-coordinate: split by a horizontal line that has half the points below or on it, and half above it

Kd-Trees


ℓ_1
ℓ_5
ℓ_7
p_4
p_5
p_9
p_{10}
ℓ_2
p_2
p_7
ℓ_3
ℓ_8
p_1
p_8
p_3
ℓ_9
p_6
ℓ_4
ℓ_6

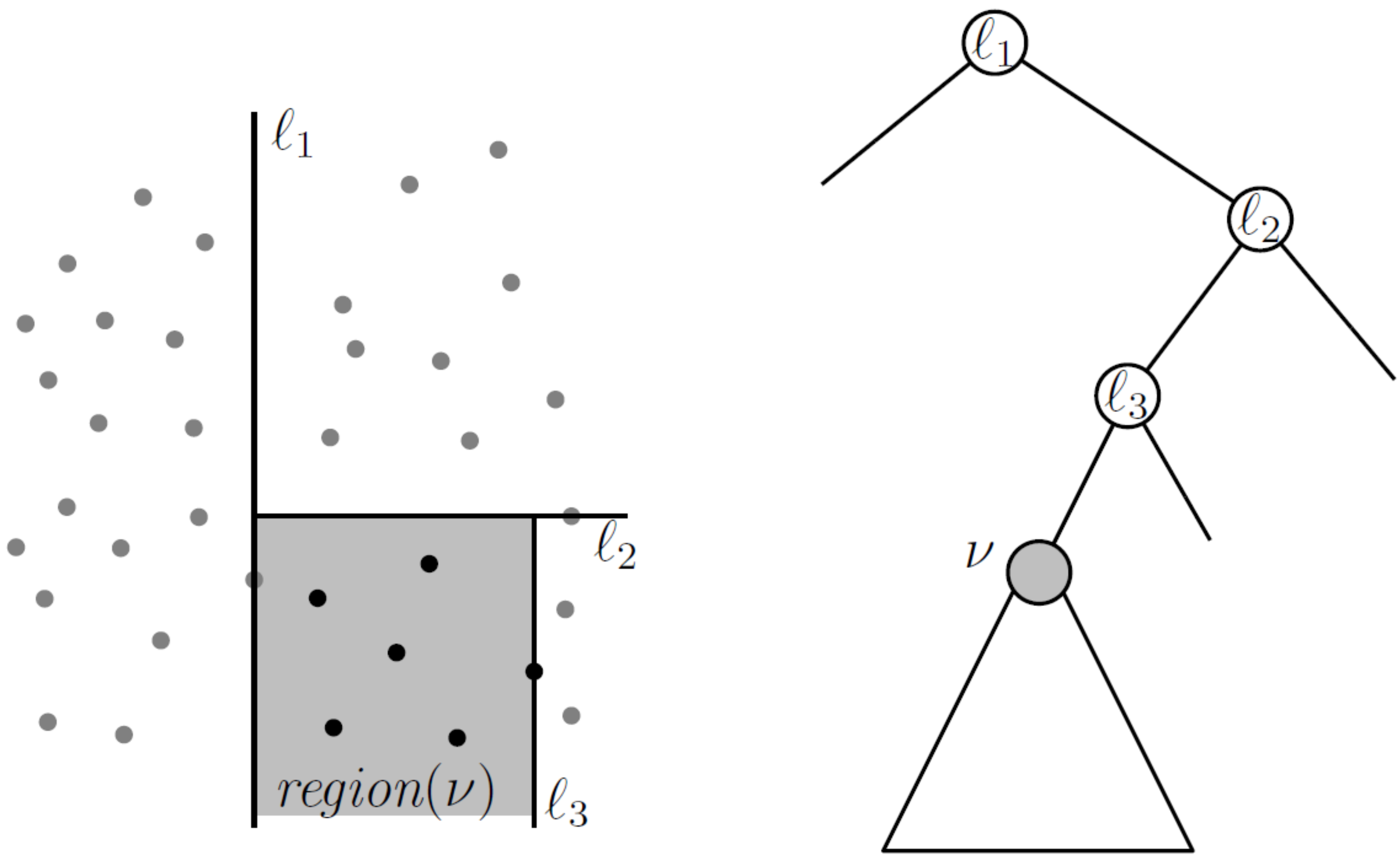

ℓ_1
ℓ_2
ℓ_3
ℓ_4
ℓ_5
ℓ_6
ℓ_7
ℓ_8
p_3
p_4
p_5
ℓ_9
p_8
p_9
p_{10}
p_1
p_2
p_6
p_7

Kd-Tree Construction

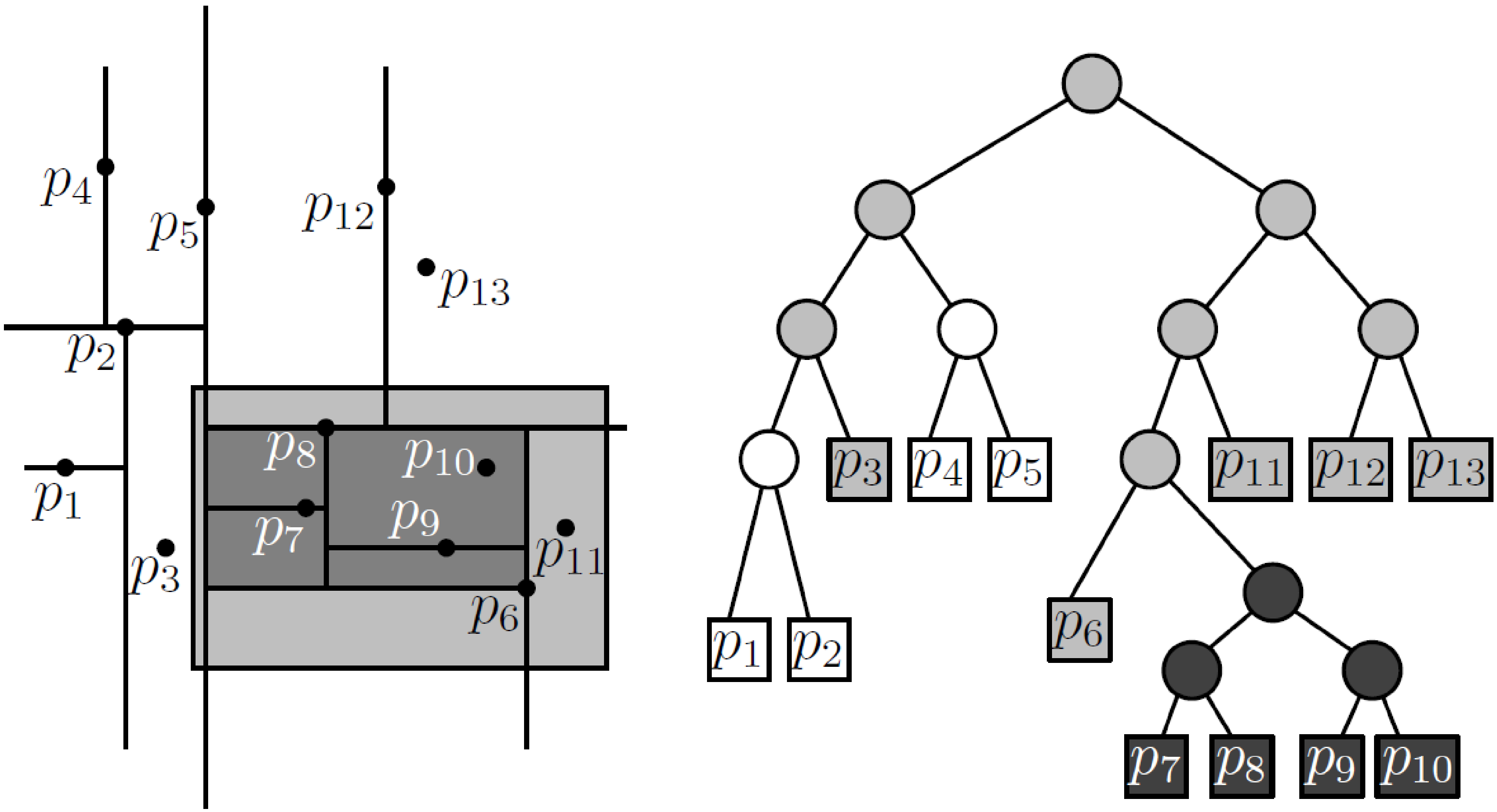
Algorithm $\textsc{BuildKdTree}(P, \mathit{depth})$

1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** depth is even
4. **then** Split P with a vertical line ℓ through the median x-coordinate into P_1 (left of or on ℓ) and P_2 (right of ℓ)
5. **else** Split P with a horizontal line ℓ through the median y-coordinate into P_1 (below or on ℓ) and P_2 (above ℓ)
6. $\nu_{\text{left}} \leftarrow \textsc{BuildKdTree}(P_1, \mathit{depth}+1)$
7. $\nu_{\text{right}} \leftarrow \textsc{BuildKdTree}(P_2, \mathit{depth}+1)$
8. Create a node ν storing ℓ, make ν_{left} the left child of ν, and make ν_{right} the right child of ν.
9. **return** ν

Kd-Tree Region of Nodes



Kd-Tree Querying



Kd-Tree Querying

Algorithm SEARCHKDTREE(ν, R)

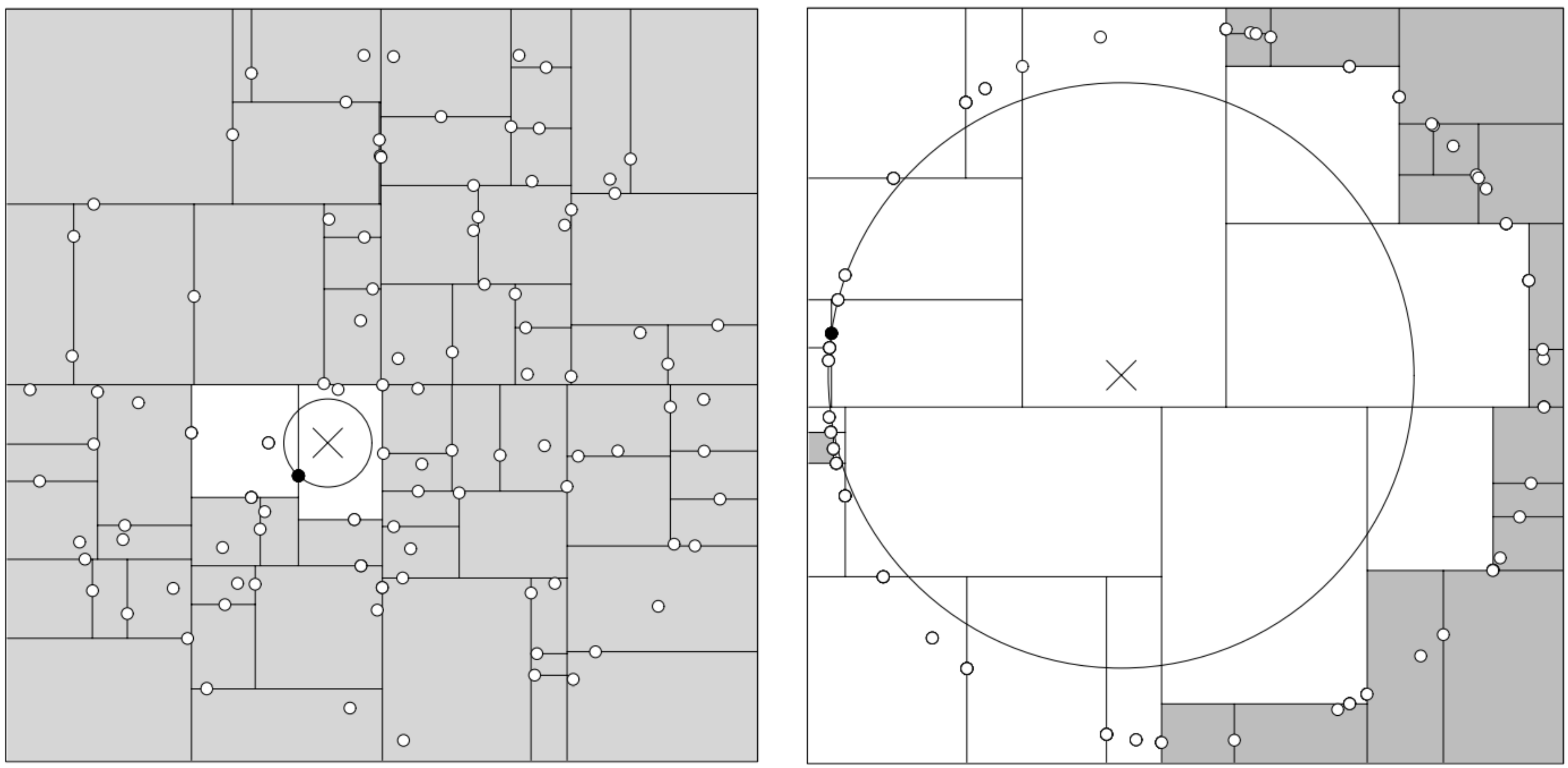
Input. The root of (a subtree of) a kd-tree, and a range R

Output. All points at leaves below ν that lie in the range.

```
if ν is a leaf
   then Report the point stored at ν if it lies in R
   else if region(lc(ν)) is fully contained in R
           then REPORTSUBTREE(lc(ν))
           else if region(lc(ν)) intersects R
                   then SEARCHKDTREE(lc(ν), R)
        if region(rc(ν)) is fully contained in R
           then REPORTSUBTREE(rc(ν))
           else if region(rc(ν)) intersects R
                   then SEARCHKDTREE(rc(ν), R)
```

Nearest Neighbor Search

- Effects of input distribution on search



Kd-Trees in Higher Dimensions

Theorem: A set of n points in d-space

can be preprocessed in O(n logn) time into a data structure of O(n) size

so that any d-dimensional range query can be answered in $O(n^{1-1/d} + k)$ time, where k is the number of answers reported

FLANN

- Adaptively selects between
  - Randomized kd-trees
  - Hierarchical k-means trees
- Criteria
  - Dimensionality
  - Relative weights on tree construction vs. search optimization

Normal Estimation in Point Clouds

- Important task in itself
- Useful for encoding data in representations enabling classification, matching and indexing
- Assumption: each point lies on locally linear (planar) patch along with some (many) of its neighbors

Normal Estimation in Point Clouds

- Find reference point’s nearest neighbors
  - Forming triplets (ref. point + 2 NNs) increases complexity with dubious benefits
  - Form pairs and accumulate partial information, since two points do not define a surface
- Accumulate information for reference point’s tangent plane
  - Vector connecting ref. point and each NN belongs to tangent plane

Normal Estimation in Point Clouds

- Form scatter (covariance) matrix

$$\mathrm{M} = \frac{1}{k}\sum_{i=0}^{k}(\mathrm{p}_i - \hat{\mathrm{p}})(\mathrm{p}_i - \hat{\mathrm{p}})^T,\ \hat{\mathrm{p}} = \frac{1}{k}\sum_{i=0}^{k}\mathrm{p}_i$$

- Normal: eigenvector with smallest eigenvalue (0 eigenvalue if perfect plane)

Normal Estimation in Point Clouds

- Form scatter (covariance) matrix
- Normal: eigenvector with smallest eigenvalue (0 eigenvalue if perfect plane)

- Tombari et al. ECCV 2010:

$$M = \frac{1}{\sum_{i:d_i<R}(R-d_i)} \sum_{i:d_i<R}(R - d_i)(p_i\text{-}p)(p_i\text{-}p)^T$$

where, R is the radius of the ball around p and d_i is the distance from p to p_i

Normal Estimation in Point Clouds

- Tombari et al. resolve sign of eigenvectors (x, y, z) by making them consistent with majority of neighborhood points

$$S_x^+ \doteq \left\{ i : d_i \leq R \wedge (\mathbf{p}_i - \mathbf{p}) \cdot \mathbf{x}^+ \geq 0 \right\}$$
$$S_x^- \doteq \left\{ i : d_i \leq R \wedge (\mathbf{p}_i - \mathbf{p}) \cdot \mathbf{x}^- > 0 \right\}$$
$$\mathbf{x} = \begin{cases} \mathbf{x}^+, & |S_x^+| \geq |S_x^-| \\ \mathbf{x}^-, & \text{otherwise} \end{cases}$$

- Where does the surface normal point for convex shapes?

The Right Way

- Count nearest neighbors more, attenuate influence of remote points
  - At the very least, normalize vectors that contribute to covariance matrix
  - Use weight function on outer products that decreases with distance

Normal Estimation

- My recommendation

$$M = \sum_{\mathrm{i:d_i<R}} e^{-\frac{d_i^2}{2\sigma^2}} \frac{(p_\mathrm{i}\text{-}\mathrm{p})(\mathrm{p_i}\text{-}\mathrm{p})^\mathrm{T}}{||\mathrm{p_i} - \mathrm{p}||^2}$$

with σ a parameter representing scale.

Descriptors for 3D Point Clouds

Invariant Descriptors

- Objective: represent point cloud (or surface) in a way that it can be compared and matched with other point clouds
- Trade-off between Uniqueness and Repeatability
- Invariance to:
  - Rigid transformation of the object
  - Viewpoint change of the scanner(s)
  - Sampling variations
  - Noise
- Local and global descriptors have been proposed
  - Focus on local here

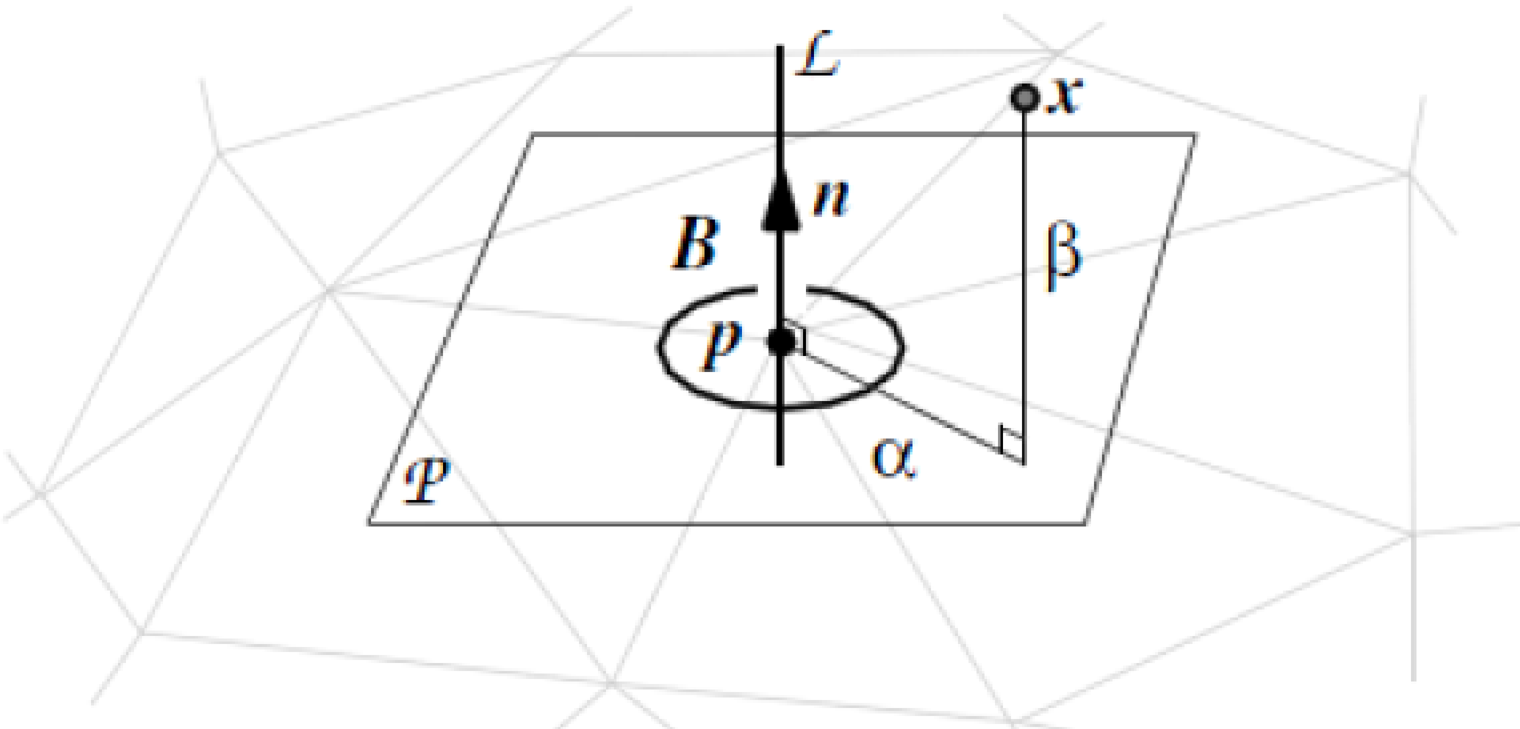
Spin Images

- Johnson and Hebert, PAMI 1999
- Arguably most popular 3D shape descriptor, used in several recognition engines
- Fast to compute and very fast to compare

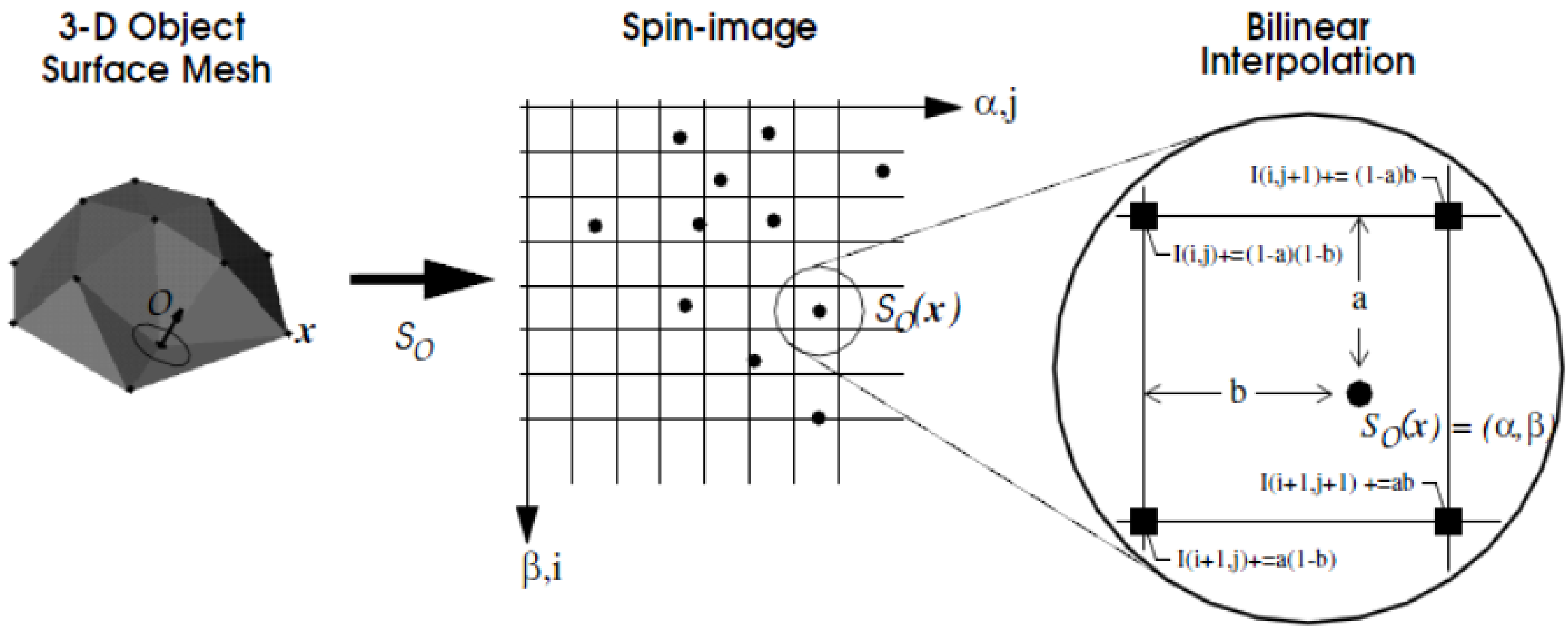
Spin Images

- Computed in a cylindrical coordinate system defined by a reference point and its corresponding normal
- All points within this region are transformed by computing:
  - the distance from the reference normal ray α
  - the height above the reference normal plane β
- A 2D histogram of α and β is used as the descriptor
- Due to integration around the normal of the reference point, spin images are invariant to rotations about the normal

Spin Images

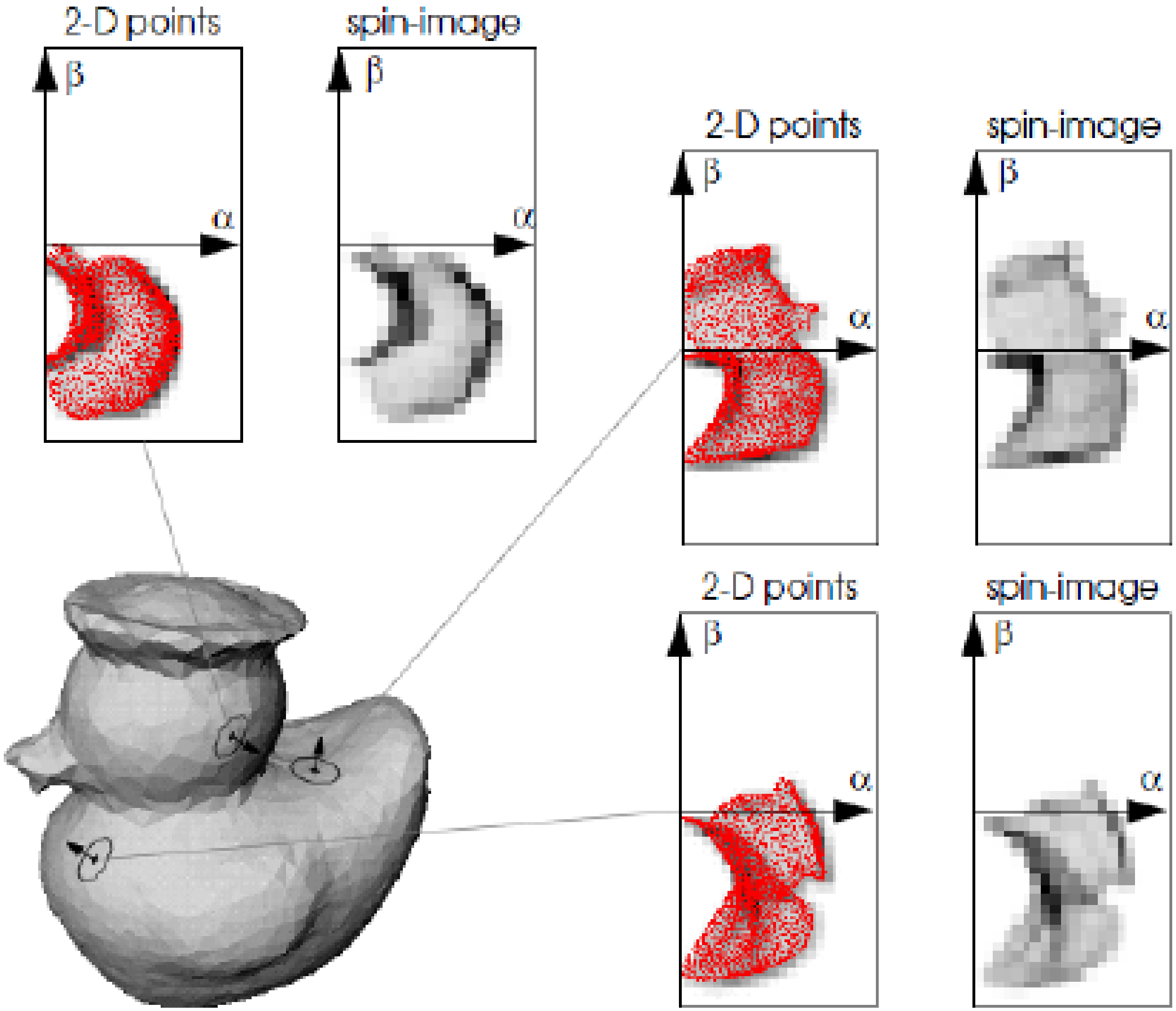


Spin Images



3-D Object
Surface Mesh
O
x
S_O
Spin-image
α,j
β,i
$S_O(x)$
Bilinear
Interpolation
I(i,j+1)+= (1-a)b
I(i,j)+=(1-a)(1-b)
a
b
$S_O(x) = (\alpha, \beta)$
I(i+1,j+1) +=ab
I(i+1,j)+=a(1-b)

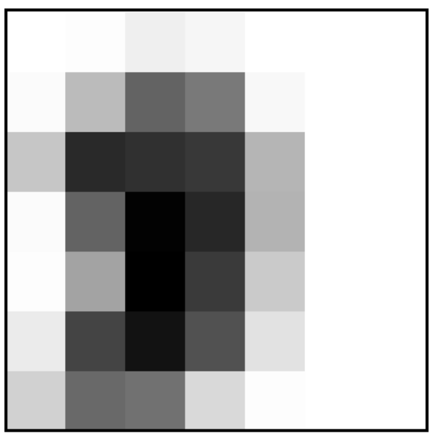
Spin Images

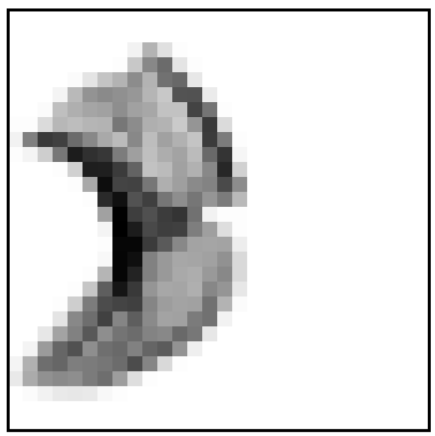


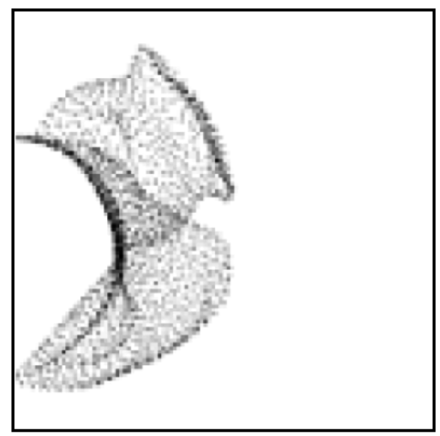
2-D points
spin-image
β
α
2-D points
spin-image
β
α
2-D points
spin-image
β
α

Effects of Parameters

- Parameters of a spin image
  - Number of bins horizontally and vertically
    - Make vertical number of bins odd
  - Bin size
  - Support angle (max angle between reference normal and neighboring point normal to be included)
    - Default 60 degrees

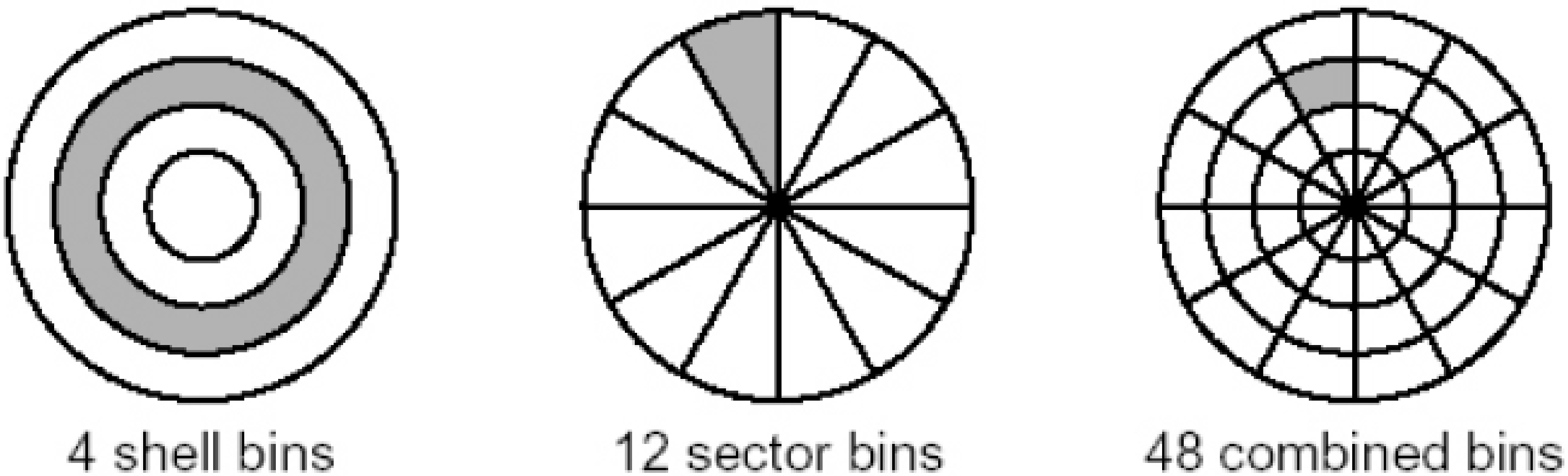






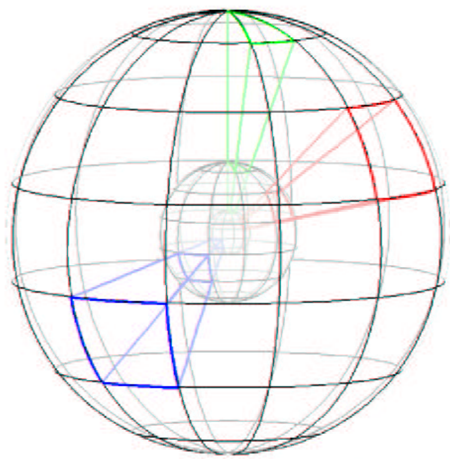
3D Shape Contexts

- Frome et al., ECCV 2004
- Histogram of neighboring points in sphere



3D Shape Contexts

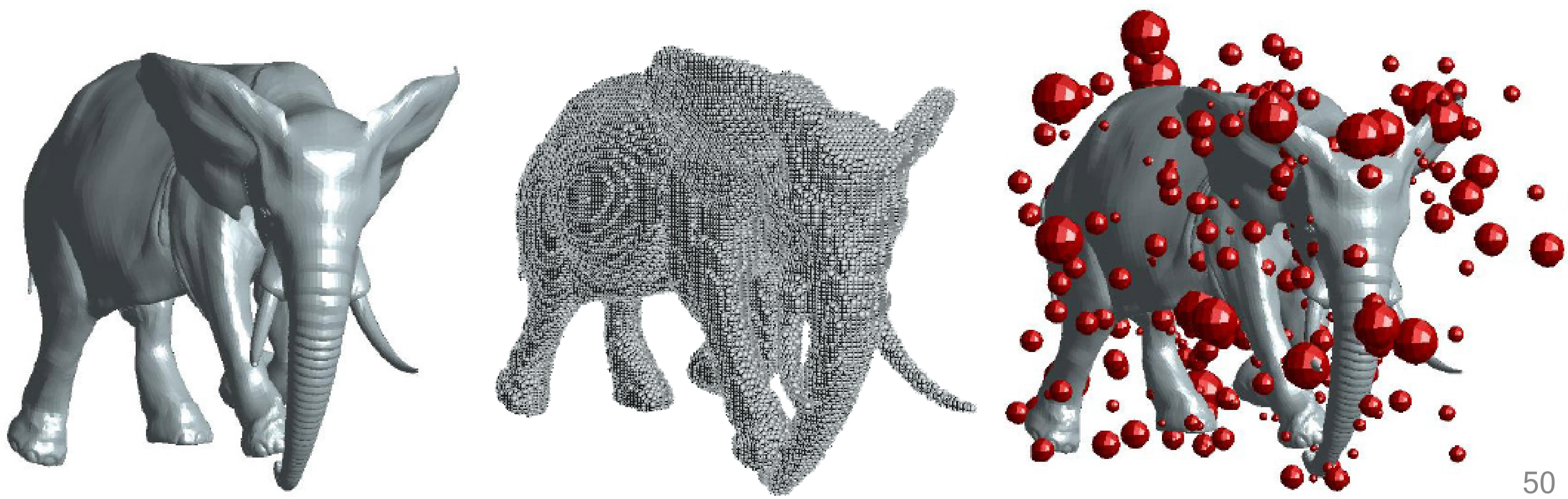
- Frome et al., ECCV 2004
- Histogram of neighboring points in sphere



- Challenge: matching requires rotations

3D SURF

- Knopp et al., ECCV 2010
- Detector and descriptor
- Convert surface into voxel representation
- Compute second-order derivatives at several scales (3 octaves)

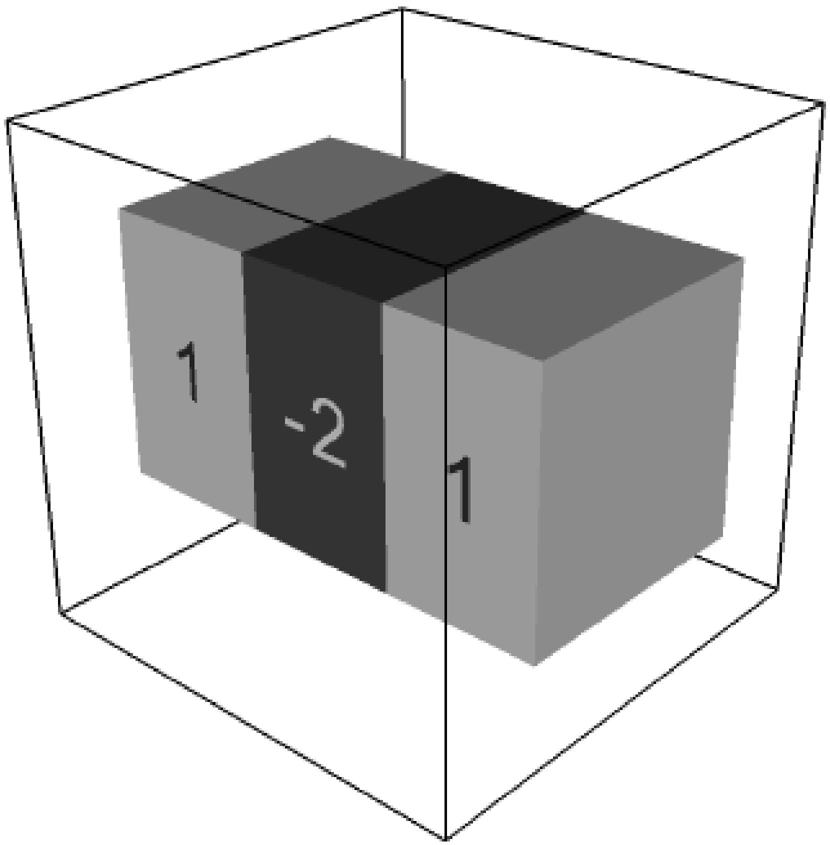


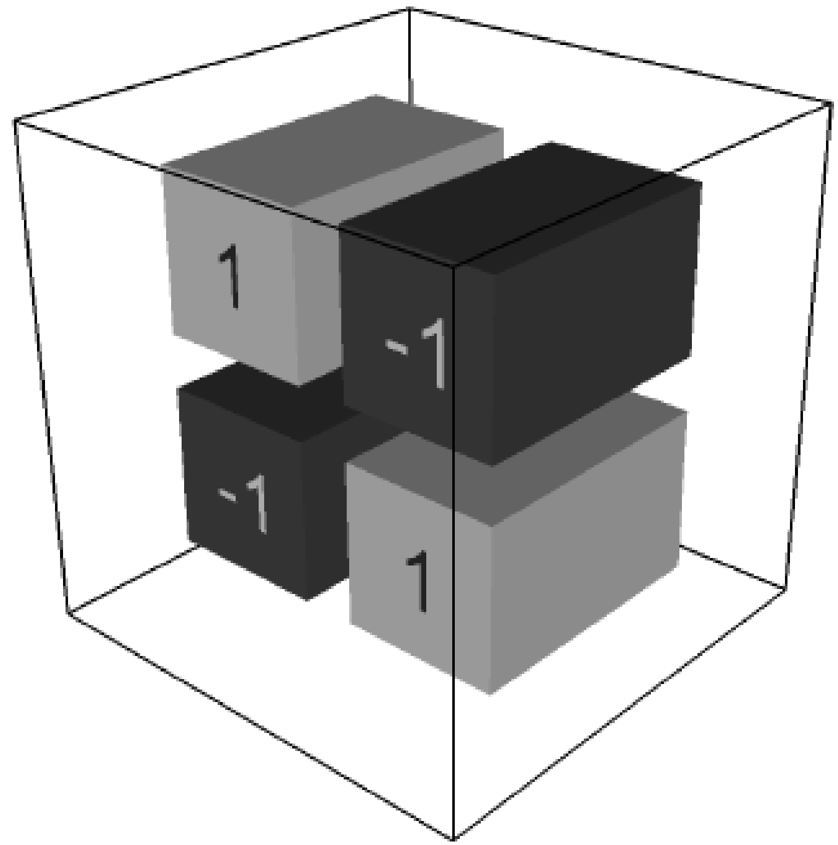
3D SURF

$$S(\boldsymbol{x},\sigma) = \left|H(\boldsymbol{x},\sigma)\right| = \left| \begin{pmatrix} L_{xx}(\boldsymbol{x},\sigma) & L_{xy}(\boldsymbol{x},\sigma) & L_{xz}(\boldsymbol{x},\sigma) \\ L_{yx}(\boldsymbol{x},\sigma) & L_{yy}(\boldsymbol{x},\sigma) & L_{yz}(\boldsymbol{x},\sigma) \\ L_{zx}(\boldsymbol{x},\sigma) & L_{zy}(\boldsymbol{x},\sigma) & L_{zz}(\boldsymbol{x},\sigma) \end{pmatrix} \right|$$

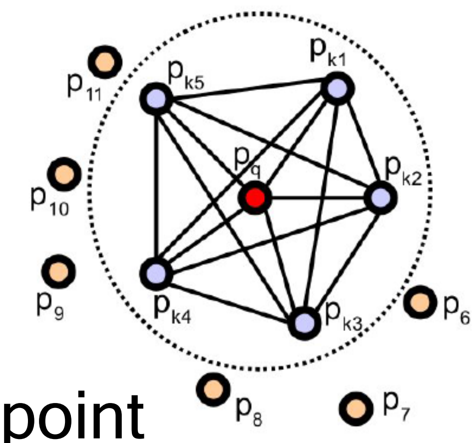
- Saliency function: absolute value of the determinant of the Hessian matrix at each point
- Select keypoints after non-max suppression
- Compute invariant local coordinate frame
- Descriptor:
  - N×N×N grid around the feature
  - At each grid cell, store a 6-dimensional description vector of Haar wavelet responses
  - Default N=3

3D Haar Filters





Fast Point Feature Histograms



- Rusu et al., ICRA 2009 (other variations exist)
- PFH: Given points and normal:
  - Find all pairs of neighbors of reference point and define local frame
  - $u=n_i$
  - $v=(p_j-p_i)\times u$
  - $w=u\times v$
  - Compute properties of frame

$$\alpha = v \cdot n_j$$
$$\phi = (u \cdot (p_j - p_i))/||p_j - p_i||$$
$$\theta = \arctan(w \cdot n_j, u \cdot n_j)$$

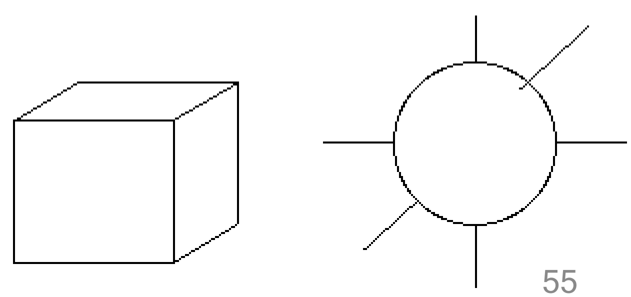
Fast Point Feature Histograms

- PFH (cont)
  - Perform persistence analysis to determine which features are salient at a given scale
- FPFH:
  - Do not compute over all pairs of neighbors, but only between reference point and its k nearest neighbors
  - Then, blend Simplified Point Feature Histograms (SPFH) with weights inversely proportional to distances between points (typically 5 bins per dimension, 125-D descriptor)
  - More optimizations in paper

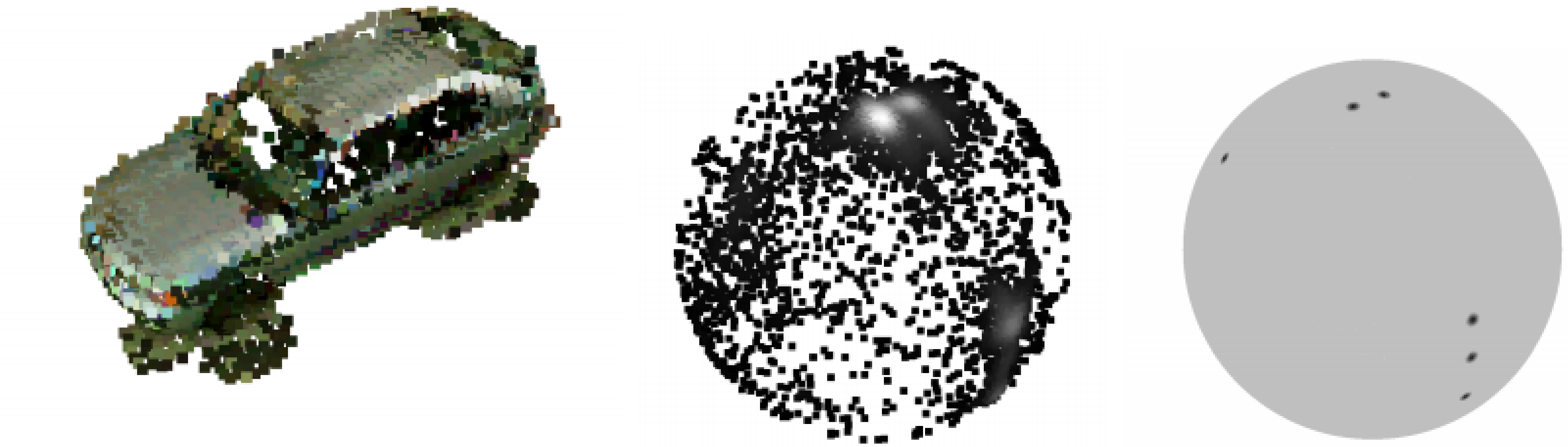
$$FPFH(p) = SPF(p) + \frac{1}{k} \sum_{i=1}^{k} \frac{1}{\omega_k} \cdot SPF(p_k)$$

Extended Gaussian Images

- Horn, Proc. of the IEEE 1984
- Represent shape by mapping the normal of each point on unit sphere
  - Surface normal has two degrees of freedom
- Convex shapes can be uniquely reconstructed given EGI
  - Non-convex shapes can be described by EGI with some loss of information



Extended Gaussian Images

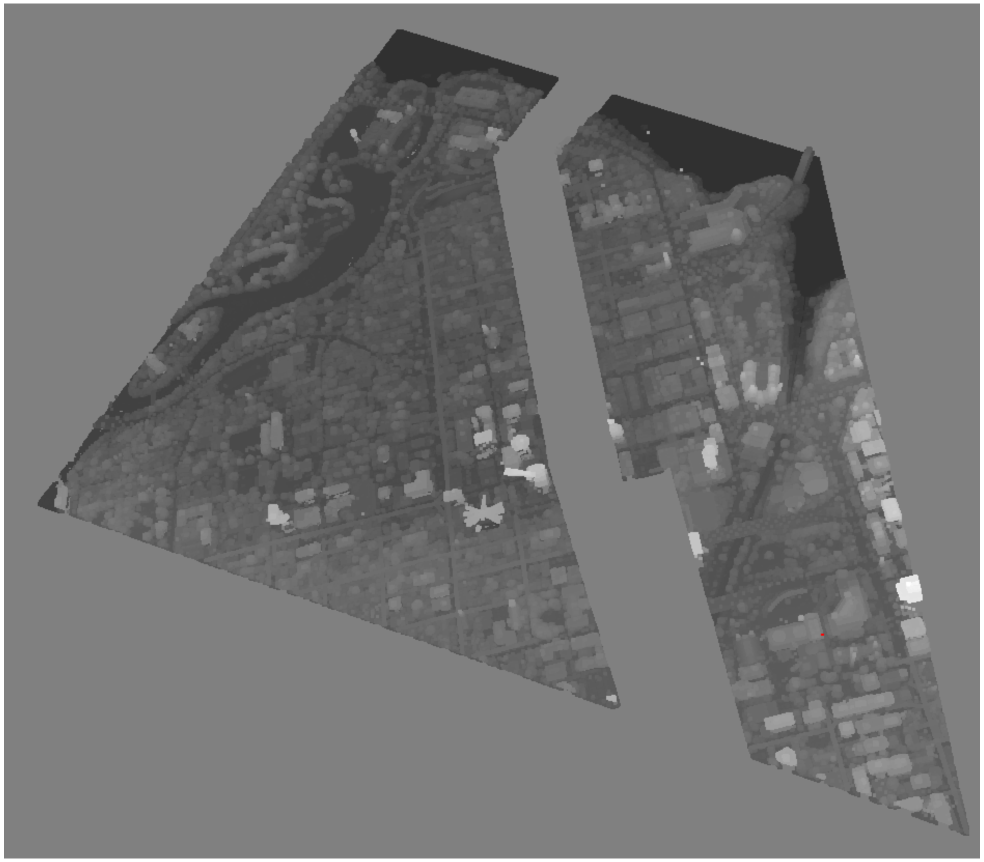


- Above: input point cloud, EGI, constellation EGI
- Matching requires alignment of two spherical histograms
  - Unpleasant, at best

Analysis of Large-Scale 3D Point Clouds

Segmentation of Large-scale 3D Datasets

- Input: colored point clouds collected by terrestrial and airborne LIDAR sensors
- Goal: detect and recognize more than 100 objects classes
  - Stadiums and power plants
  - Mailboxes and parking meters
  - Powerlines



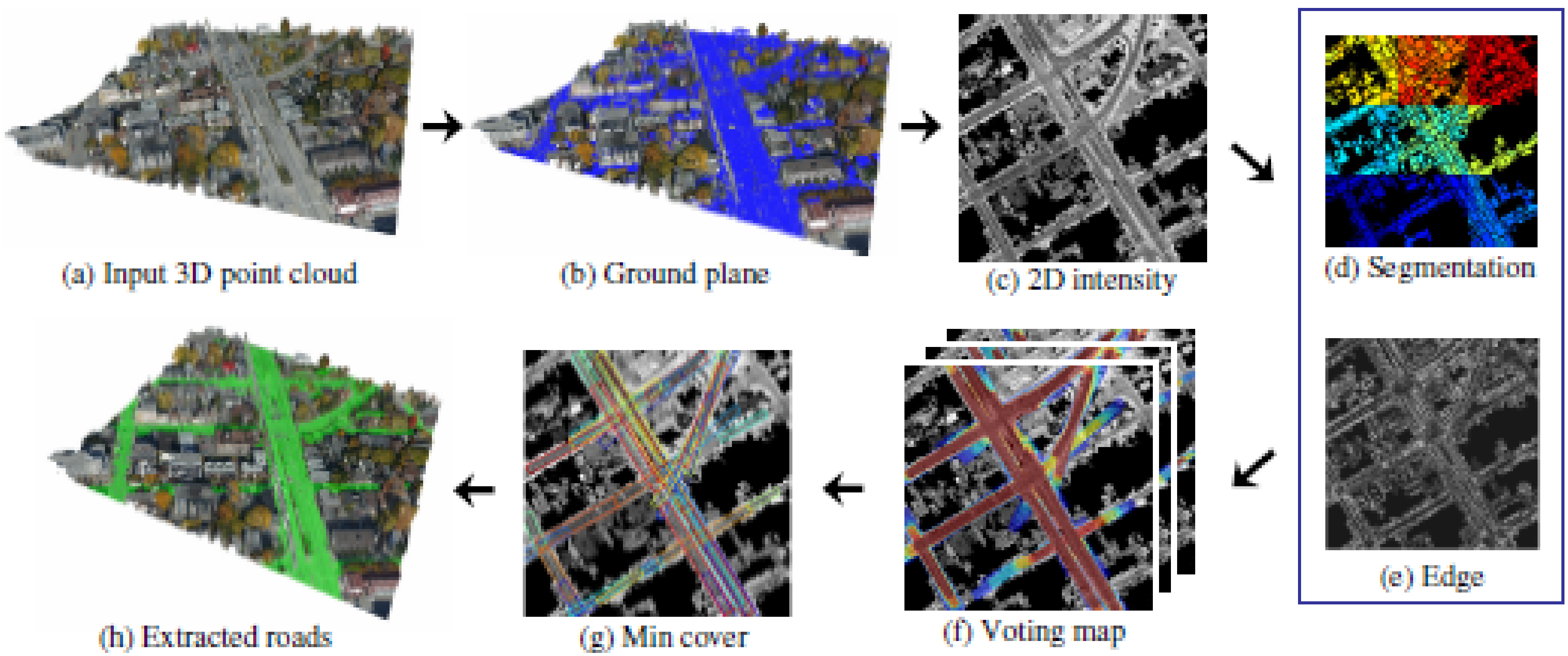
975 million points

A Minimum Cover Approach for Extracting the Road Network from Airborne LIDAR Data

- Combine local edge and region information to estimate road likelihood
- Pose road extraction as minimum cover problem
  - Explain likelihood maps by rectangular road segments with strong preference for elongated segments

[Zhu and Mordohai, 3DIM 2009]

Overview of the Algorithm



Hypothesis Generation

- Sample hypotheses from region boundaries returned by NCut segmentation on intensity image
  - Captures low-contrast boundaries
  - Test multiple values for width (10-25m) and length (40-300m)
- Output likelihood maps L for each width by combining boundary and interior feature strengths
  - Each likelihood map covers all orientations resulting in good performance at intersections

$$\mathcal{L}_k(x,y) = \sum_{H_i \in \mathcal{H}_k^{valid}} \beta_{bd} S_i^{bd}(x,y) + \beta_{int} S_i^{int}(x,y)$$

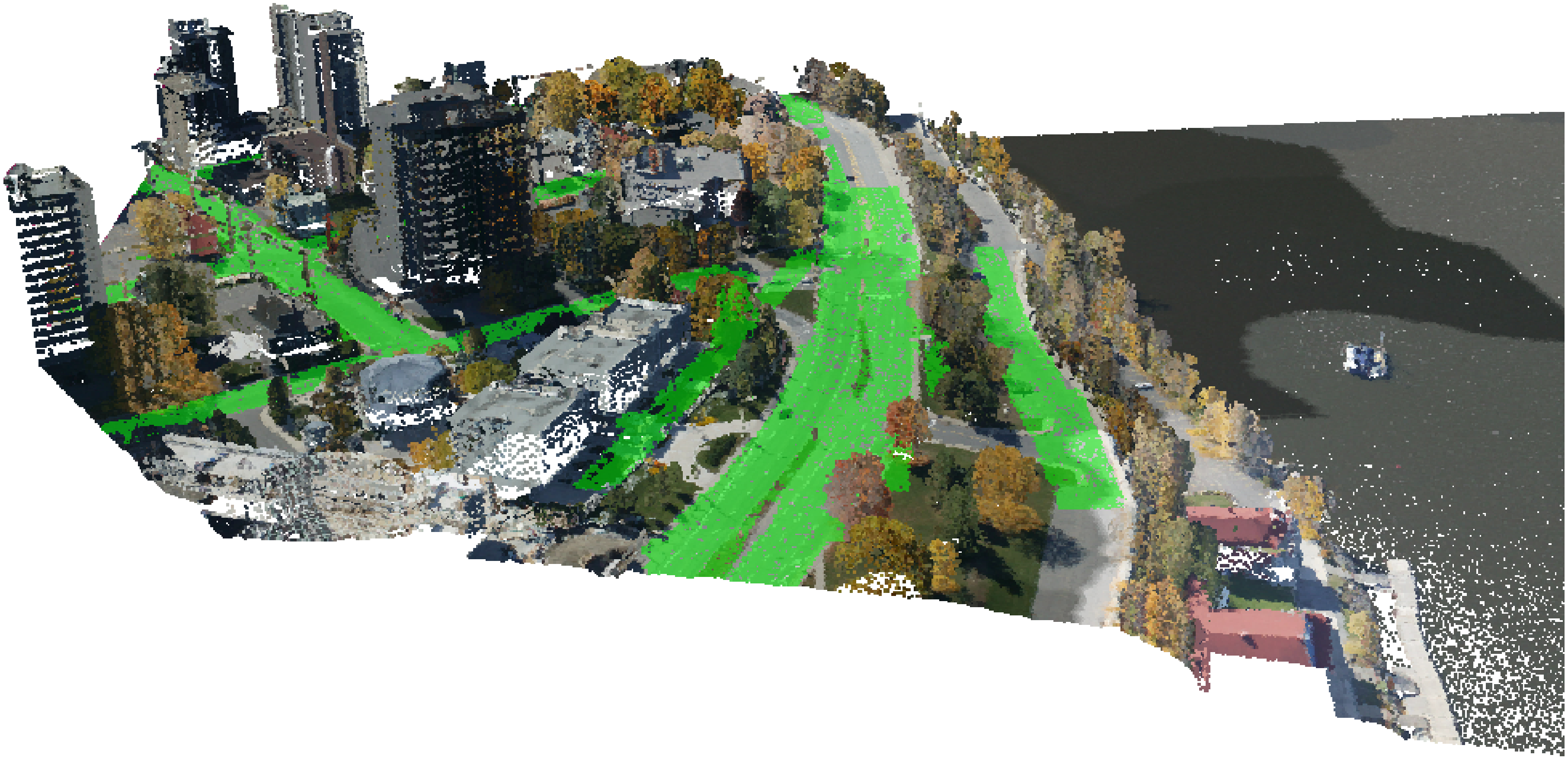
Detection as Minimum Cover

- Explain likelihood maps by sparsest set of road segments
  - Penalize uncovered parts according to road likelihood
  - Penalize covered parts according to bg likelihood
  - Penalize for additional components
- NP-hard in general, but greedy approximation with theoretical guarantees is effective [Felzenszwalb and McAllester, 2006]

$$Cost(R_i) = \sum_{s_j \in R_i} C(s_j) + A$$

$$Cost_{cover}(\mathcal{R}_\mathcal{S}) = \sum_{R_i \in \mathcal{R}_\mathcal{S}} Cost(R_i) + \sum_{s_j \notin \mathcal{R}_\mathcal{S}} \frac{\mathcal{L}(s_j)}{\mathcal{L}_{max}}$$

Road Detection

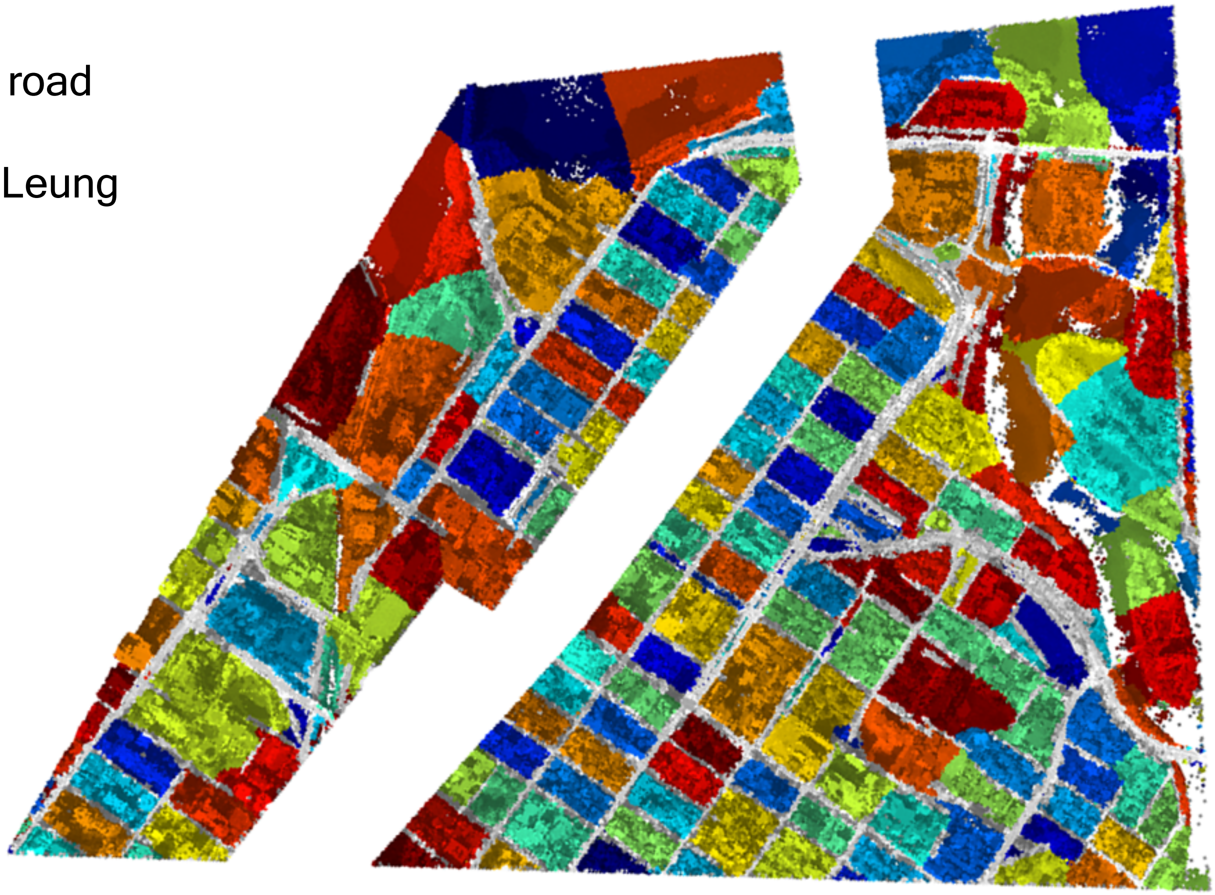


Road Detection

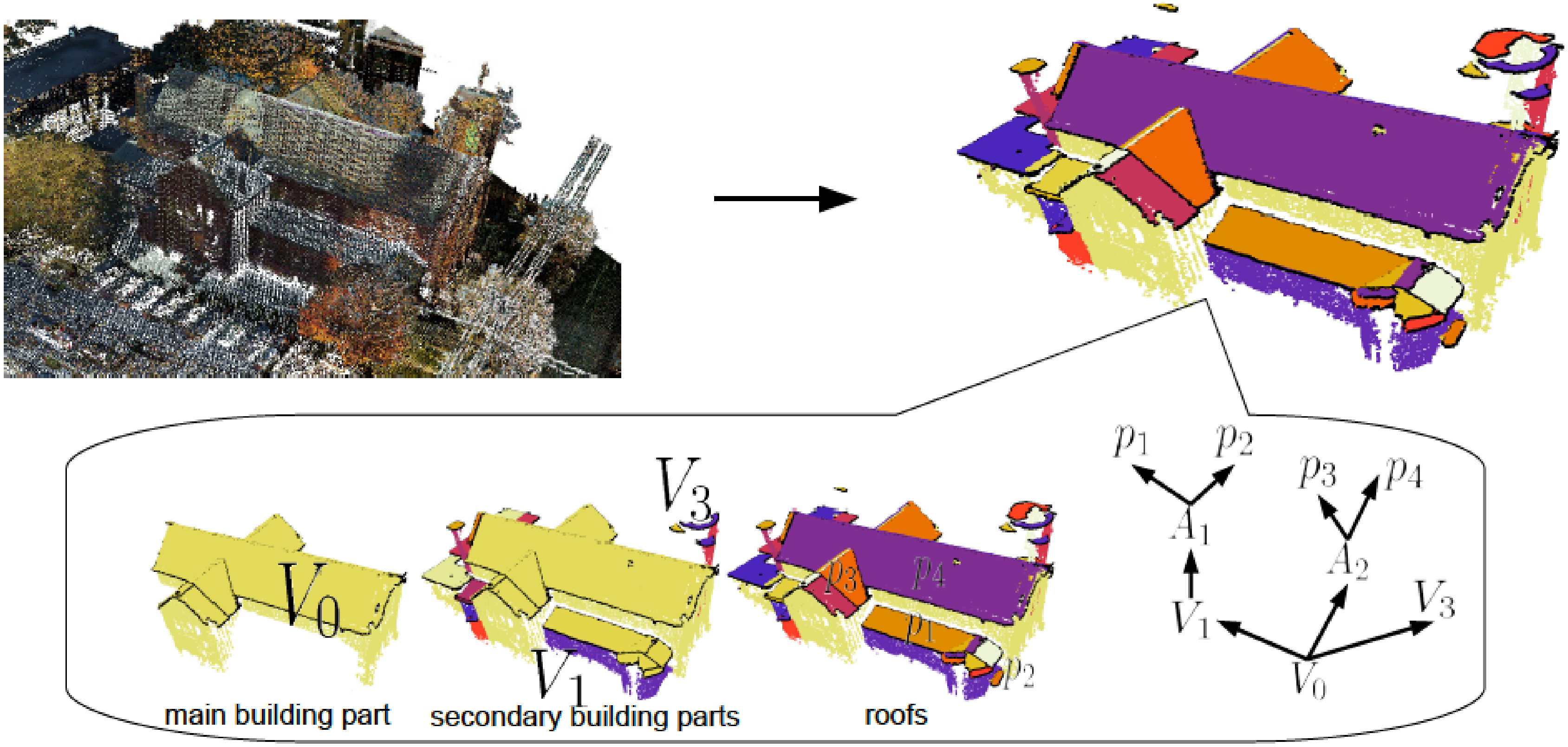
Road Detection

City Blocks

Segment city blocks from road likelihood map using the intervening contour idea (Leung and Malik, 1998)



Building Detection and Parsing



- Detection of buildings from unorganized range data
- Parsing of the buildings into an hierarchical, semantic representation

[Toshev, Mordohai and Taskar, CVPR 2010]

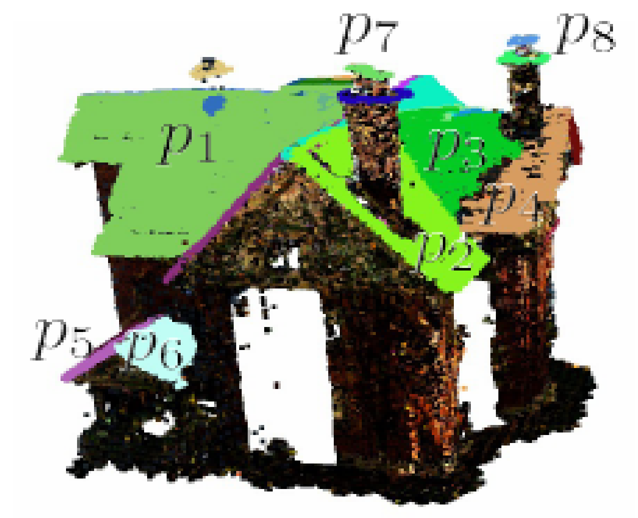
Approach

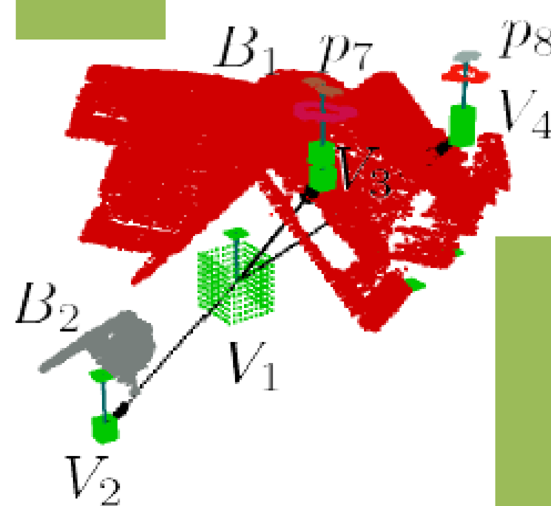
- Use a generic grammar based on simple geometric rules
- Apply dependency parsing for efficient inference
- Define parsing and detection in a single framework

- Primitives:
  - Planes and planar patches – explicitly detectable from point clouds
  - Volumes – represent building parts and are enclosed by planar patches

Grammar

- Terminals: planar patches extracted from point cloud.
- Non terminals:
  - Roof components
  - Volumes enclosed by the roofs
  - Supernodes : a global “building” and a global “non-building” node used for detection

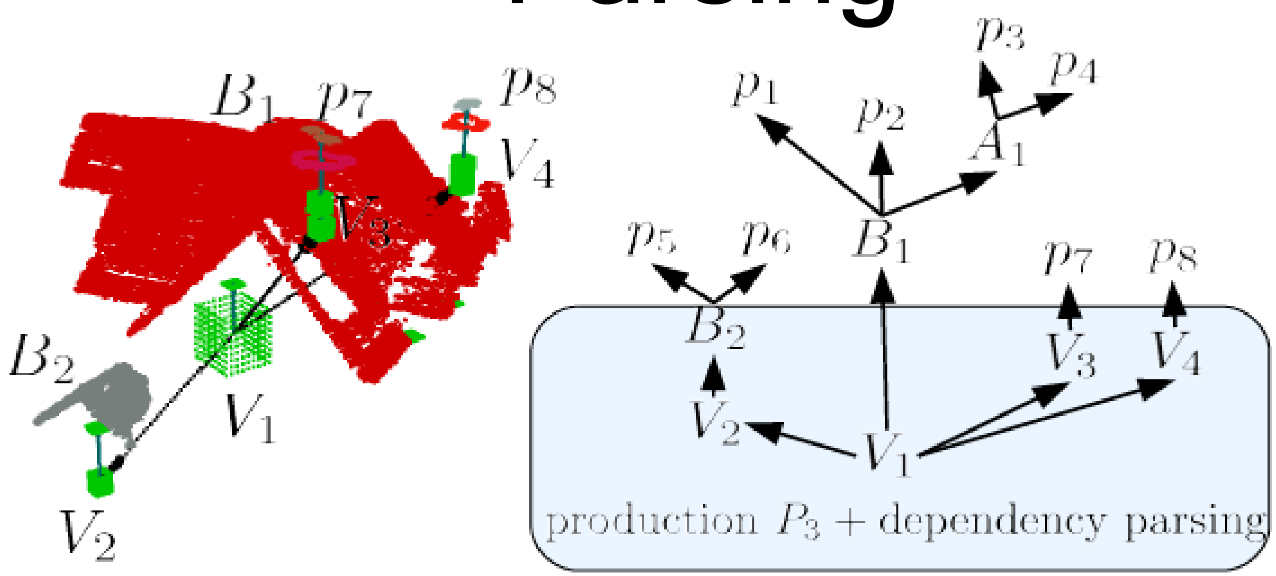




Classification

- Classification of a volume using a linear SVM
- Features are extracted from planar patches enclosing the volume:
  - Elevation
  - Distance to the nearest ground point
  - Convexity of the upper volume surface
  - Scatter of point cloud in the volume
  - Area and aspect ratio
  - Degree of enclosure by empty space
  - Fitting error

Parsing



- Productions for parsing planar patches are deterministic
- Hierarchy among volumes is not deterministic – dependency parsing
- For set of planar patches, a sequence of productions generates a parse tree

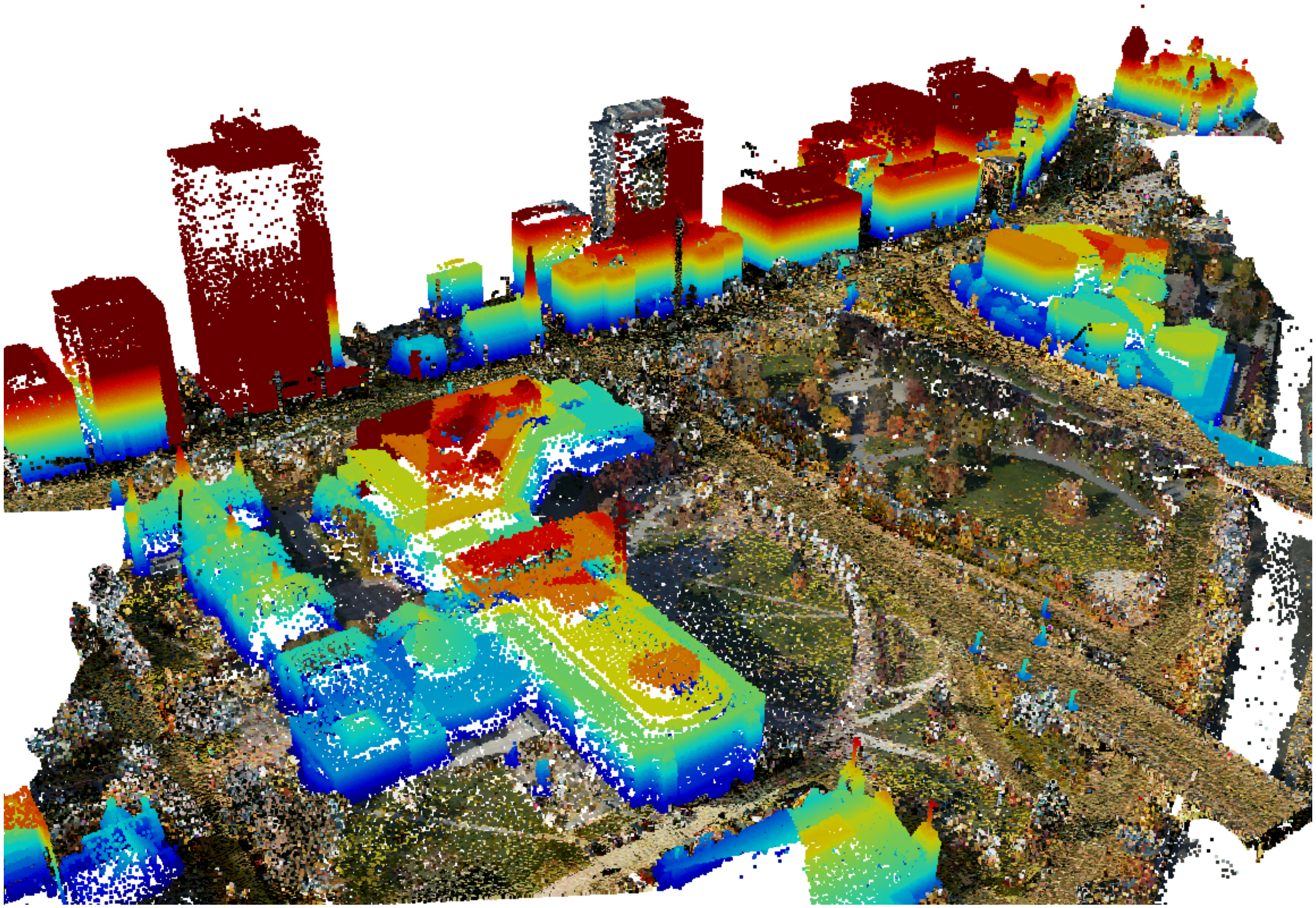
Inference

- Construct a directed graph G consisting of the volumes
- Edge weights represent:
  - –Hierarchy based on the area
  - –How likely they belong to the same class
  - –Whether they are children of the supernodes
- A parse tree T is a maximum spanning tree in G
- Use Chi-Liu/Edmonds algorithm to compute MST

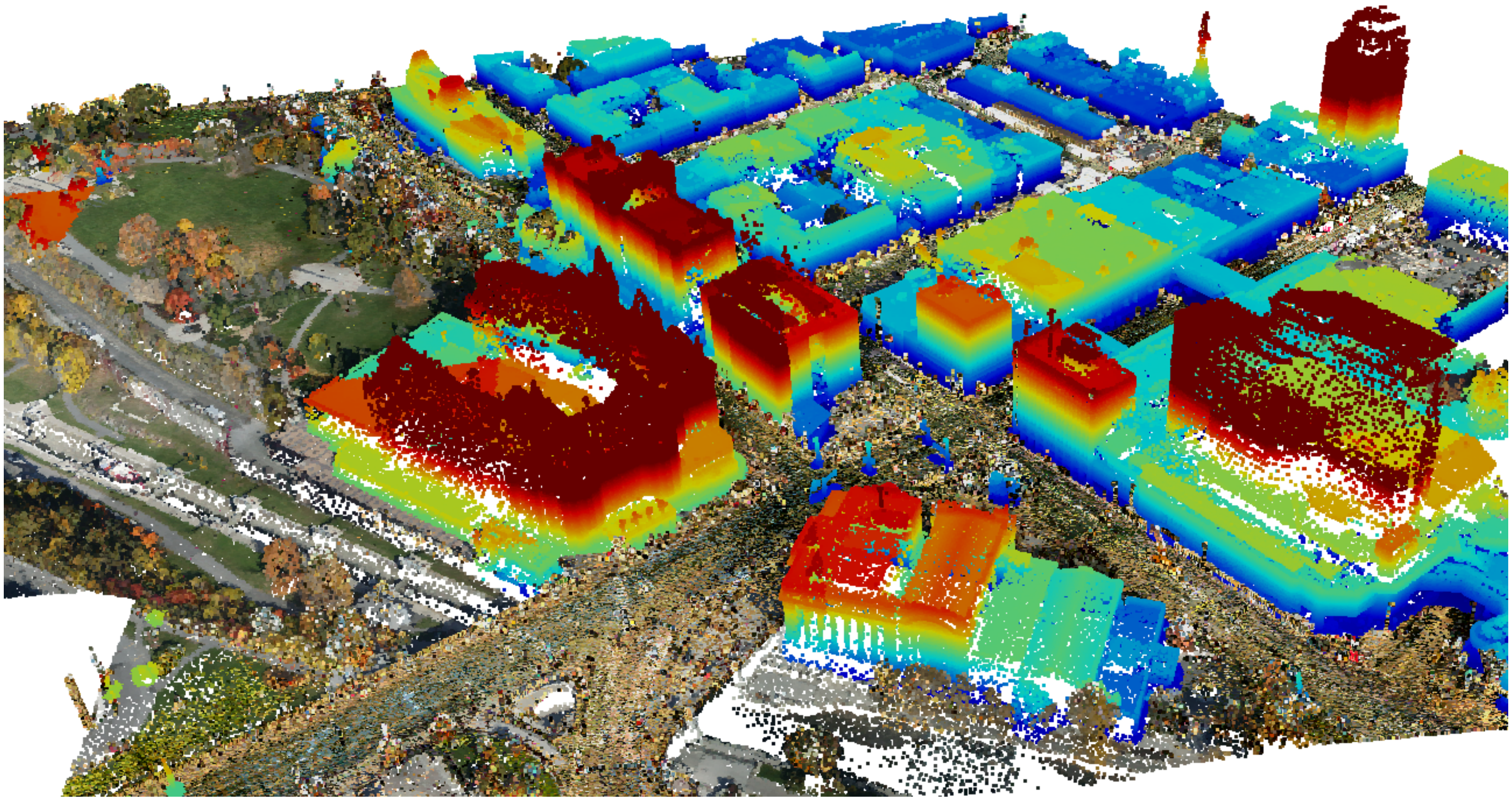
Quantitative Results

- 9 blocks used for training (buildings and their parses are labeled)
- 78 blocks used for testing (only buildings are labeled)
- Detection results (accuracy of patch classification): 89.3%
- Parsing accuracy (3-fold cross-validation on the 9 blocks): 76.2 %

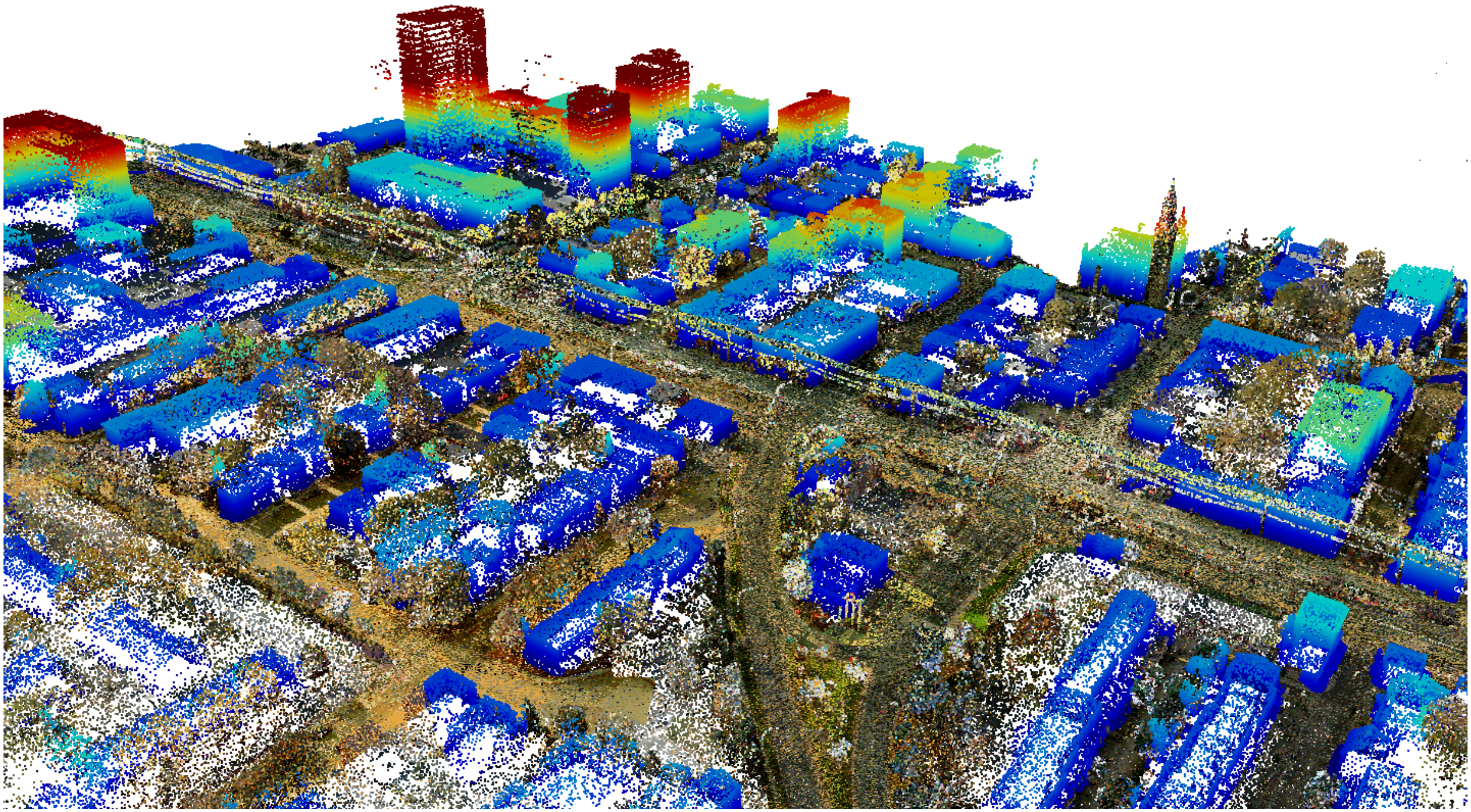
Detection Results



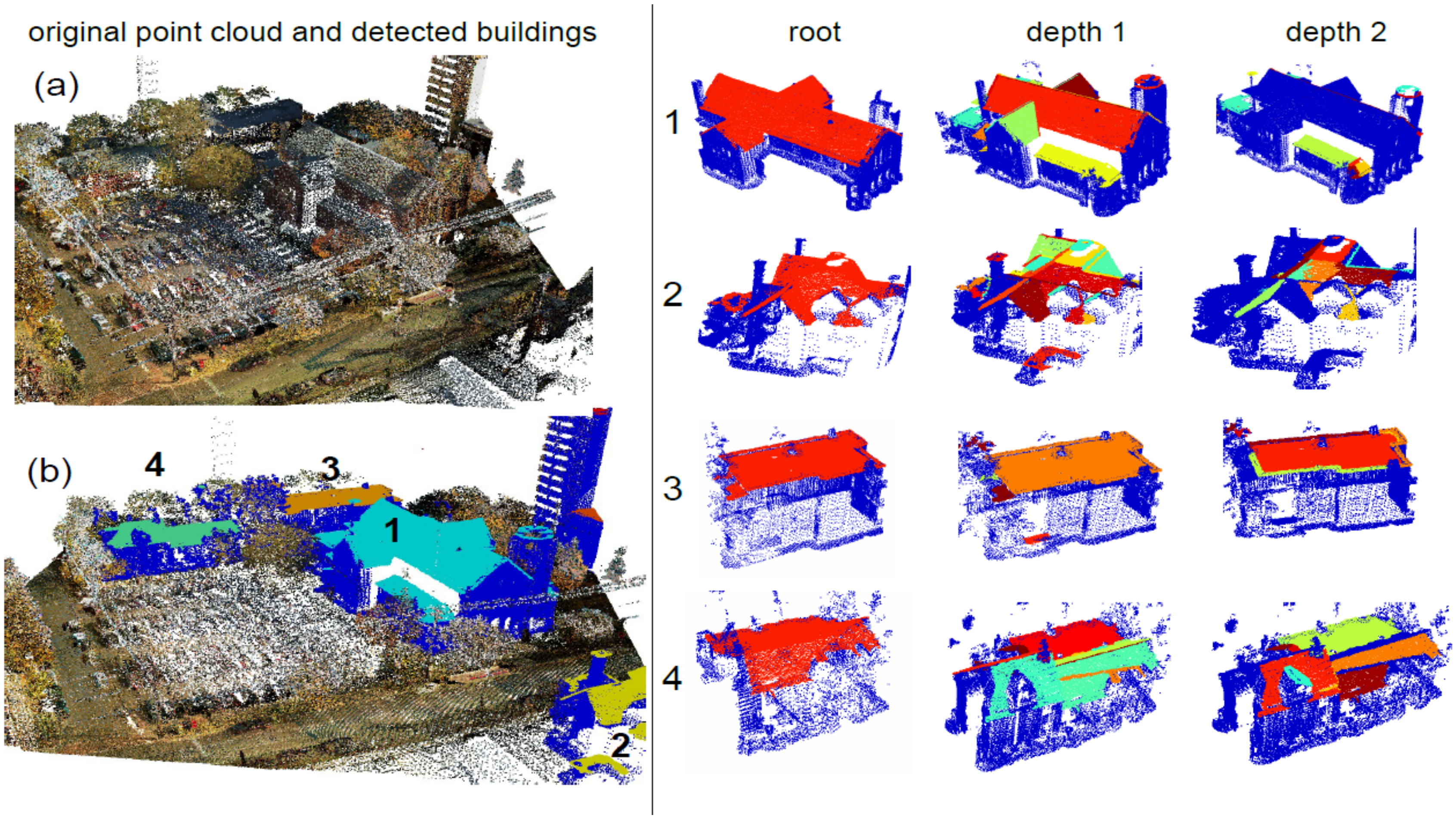
Detection Results



Detection Results



Parsing Results



3D Object Detection using Bottom-up and Top-down Descriptors

- Detect objects in large-scale 3D datasets
- Requirements:
  - Precision
  - Recall
  - Speed

[Patterson, Mordohai and Daniilidis, ECCV 2008]

Shape Descriptors

- Different descriptors provide different trade-off between speed and accuracy
- More types of invariance => faster, less discriminative
  - E.g. spin images vs. 3D shape contexts
- Global descriptors are more accurate, but are sensitive to occlusion (and deformation)
  - Require only one comparison per target
  - Need segmentation hypotheses

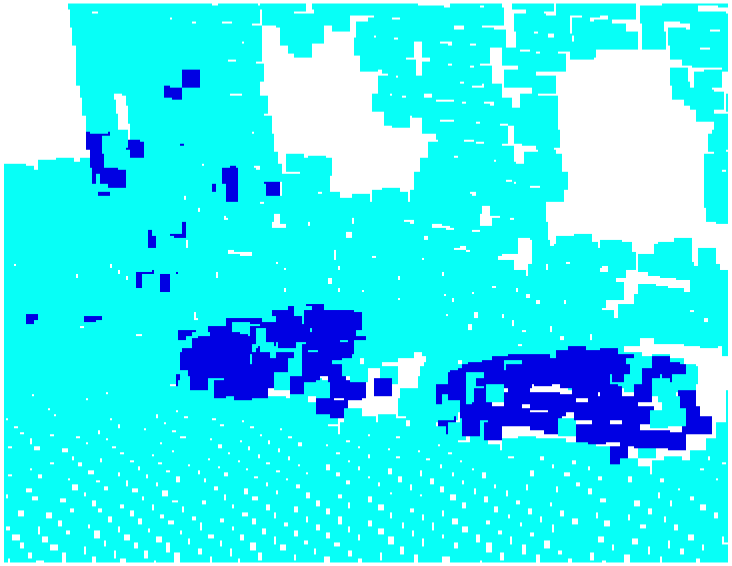
Approach

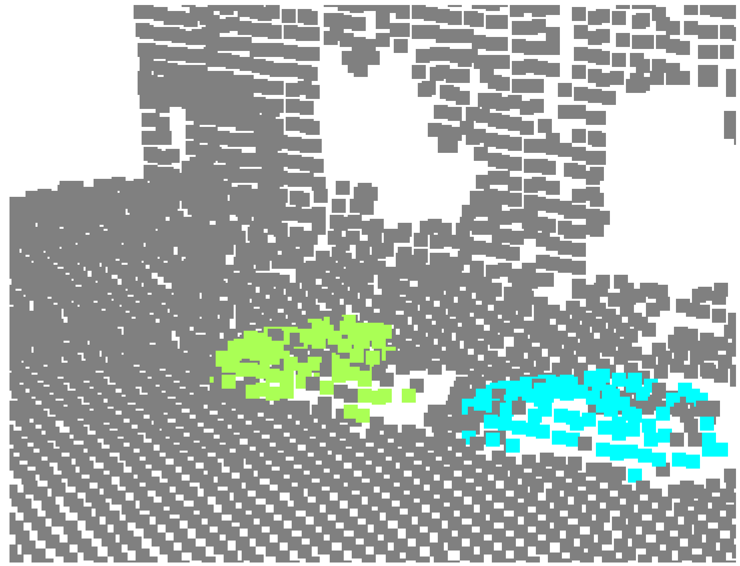
- Spin images for fast bottom-up detection of regions of interest
  - Objective: maximize recall
- Extended Gaussian Images (EGIs) as global top-down descriptors to verify hypotheses
  - Objective: prune wrong hypotheses
  - Side product: best alignment with most similar model

Experiments

- Dataset: 220 million points collected from terrestrial sensors
- 2.2 million used as training set for spin images
  - 81,000 spin images in DB_{SI}, 2600 positive
  - 17 cars in DB_{EGI}

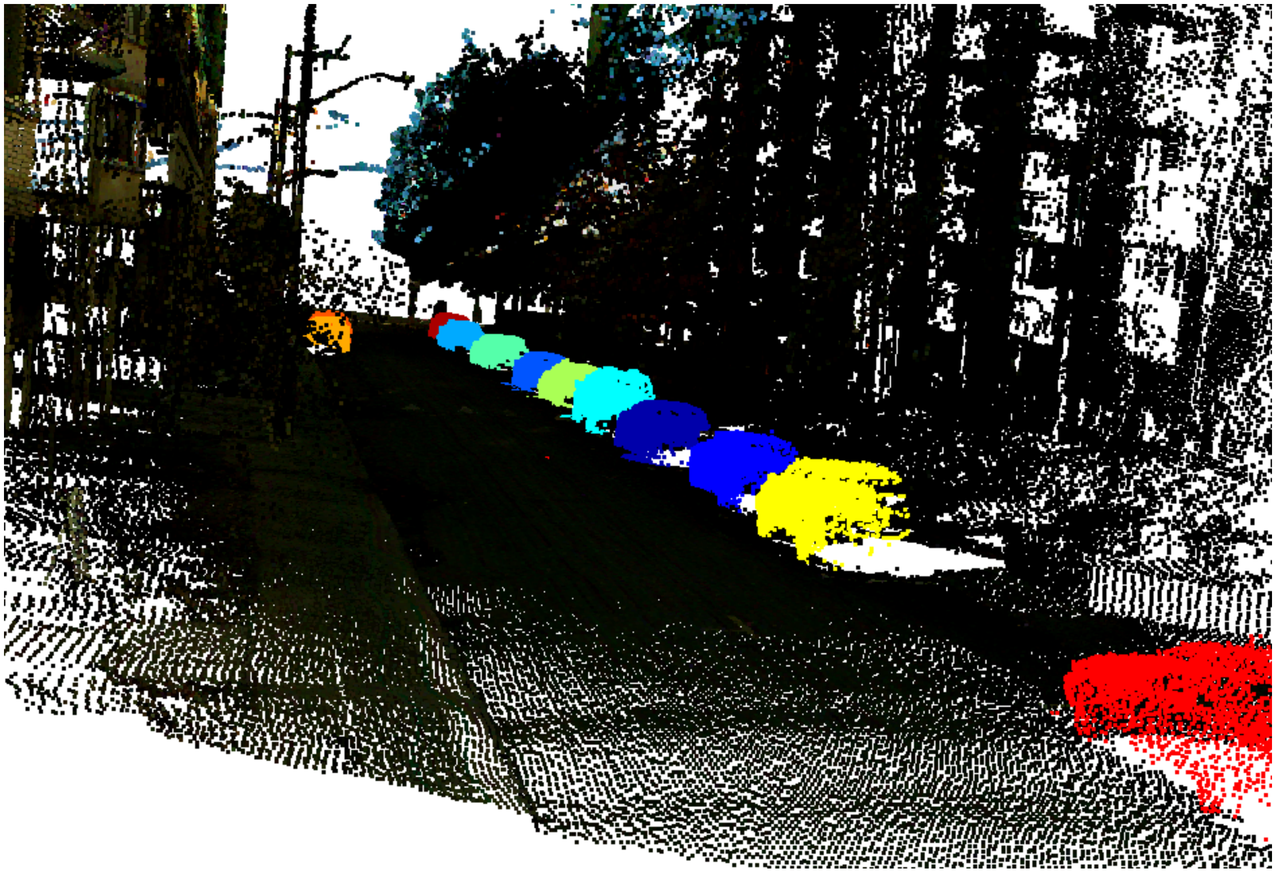
Bottom-up Detection



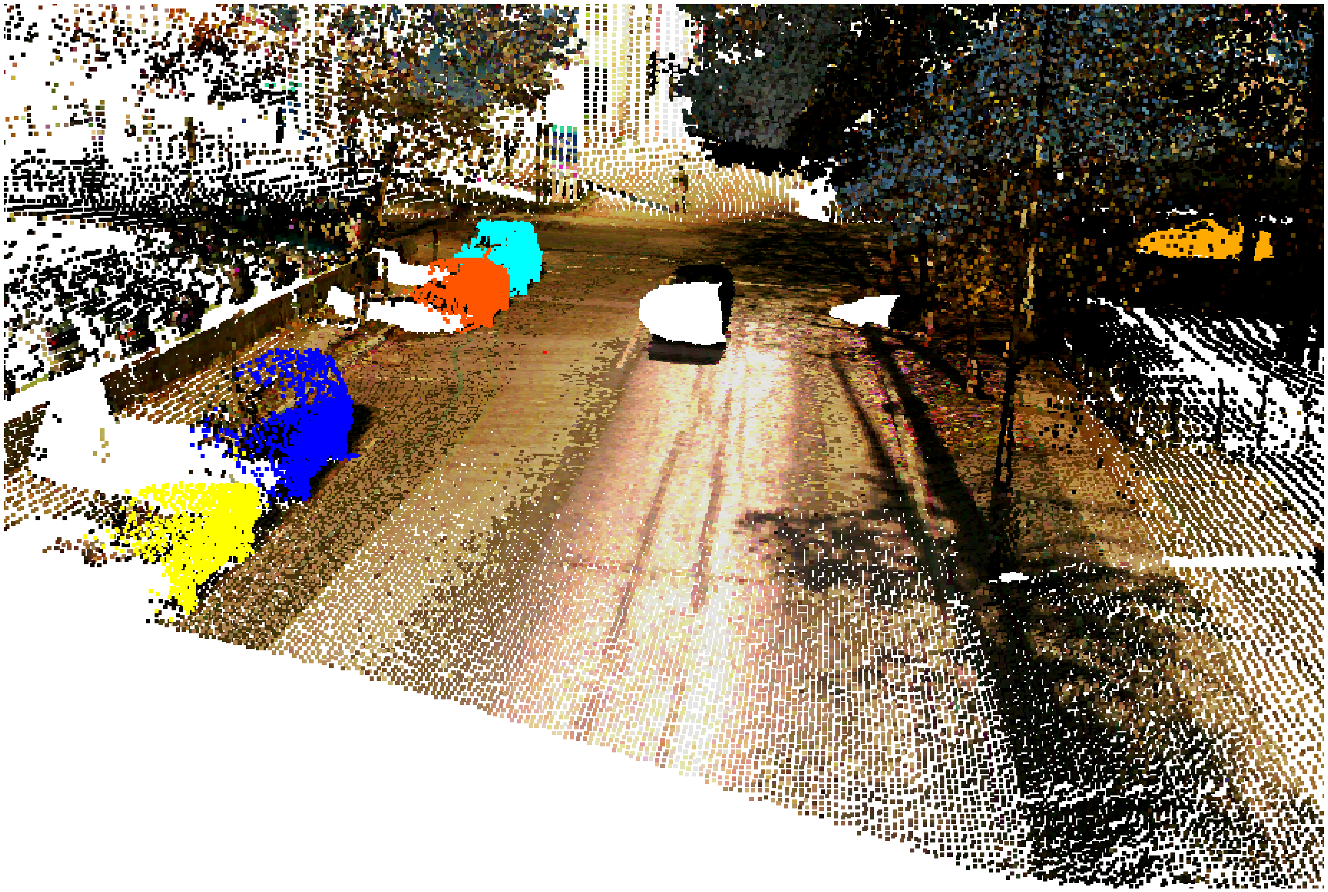


- Spin images of unknown scene classified as positive or negative
- Positive spin images clustered to form hypotheses
  - Minimum number required for hypothesis

Results

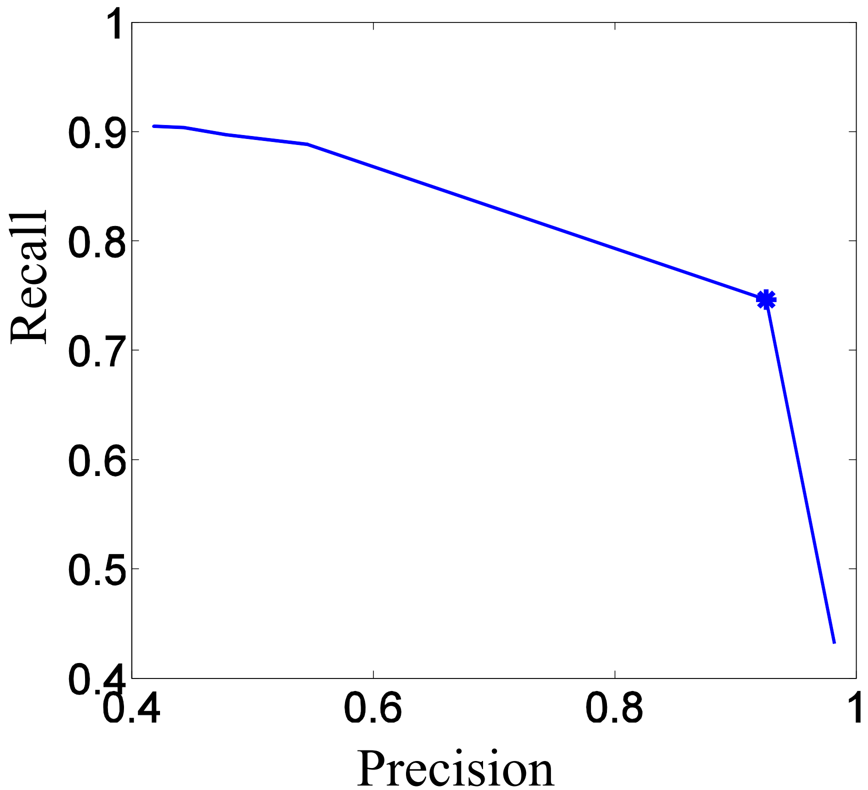


Results



Results

Precision-Recall Curve



- Data: 1221 cars
- Bottom-up stage:
  - 2200 hypotheses, 1100 correct
- Top-down stage *:
  - 905 true positives
  - 74 false alarms
  - 316 missed detections

Car Detection Video