FeatureLego: Volume Exploration Using Exhaustive Clustering of Super-Voxels

Shreeraj Jadhav, Saad Nadeems, and Arie Kaufman

Stony Brook Univerisity

Introduction

- Motivation
    - Typical approaches
        - Organize regions in a tree / hierarchy
        - Fewer choices for user
        - Modify region boundaries or re-compute with different input parameters

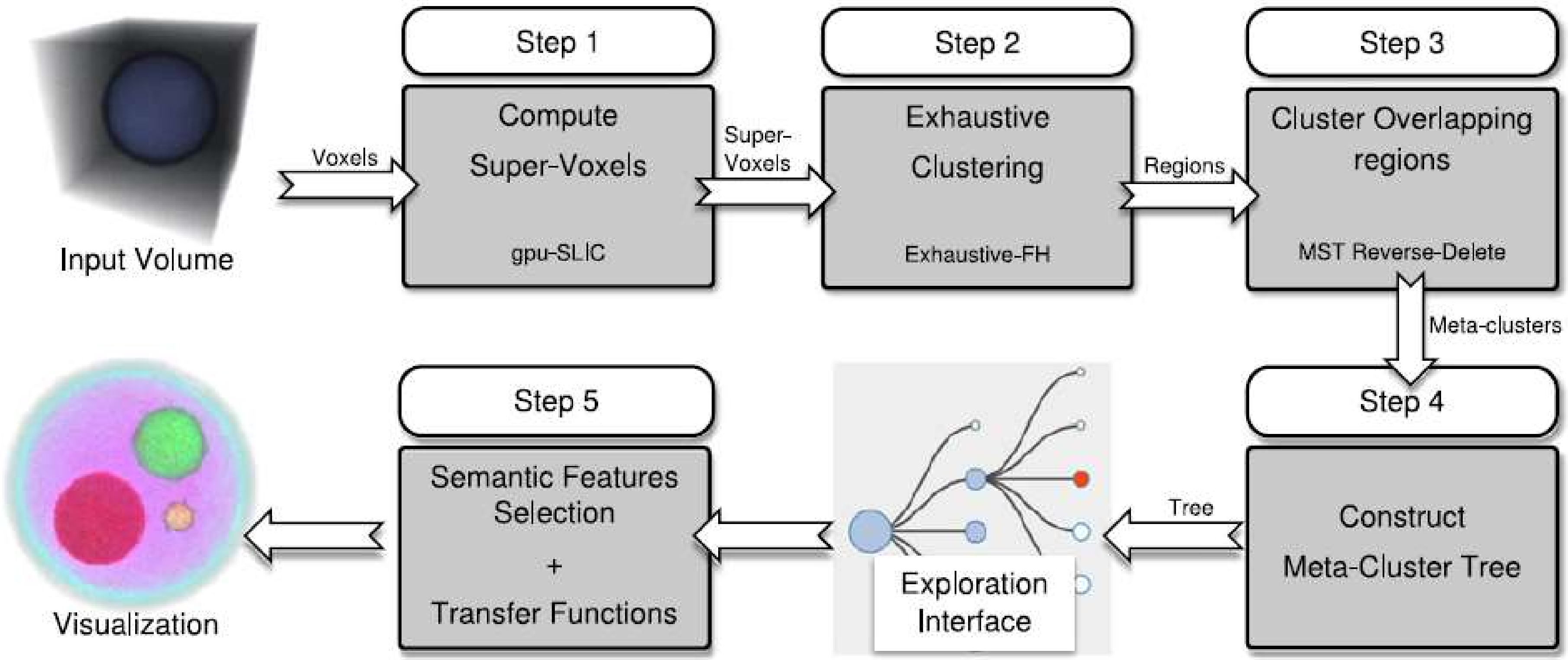
    - Pamameter sampling
        - Remains arbitray
        - Computationally expensive (brute force)
        - How much sampling is adequate

- Their solution
    - Exhaustive clustering

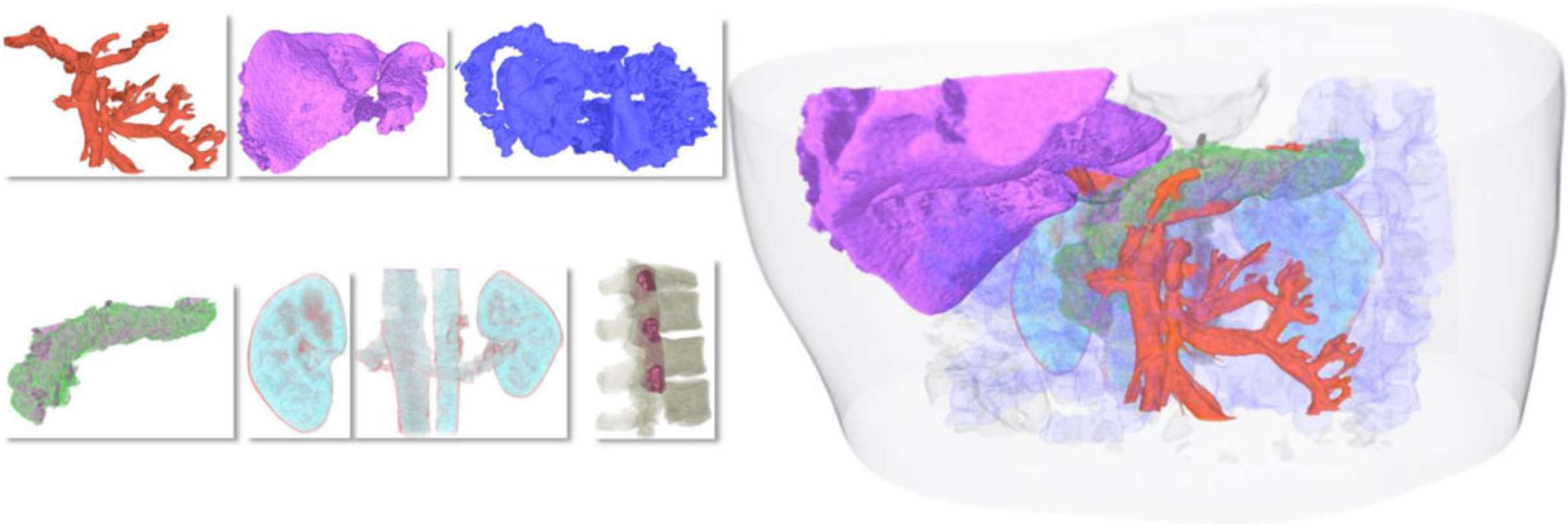
Approach

- Finest selection granularity -> super-voxels
  - Local compactness (reduce fragmentation)
  - Effciency for following steps
- Efficient exhaustive clustering of super-voxels
  - Extend a well-known image segmentation technique to 3D
    - Felzenszwalb and Huttenlocher **(FH) method**
- Meta-cluster tree
  - Efficiently manage and explore large number of regions

FeatureLego Pipeline Overview



FeatureLego



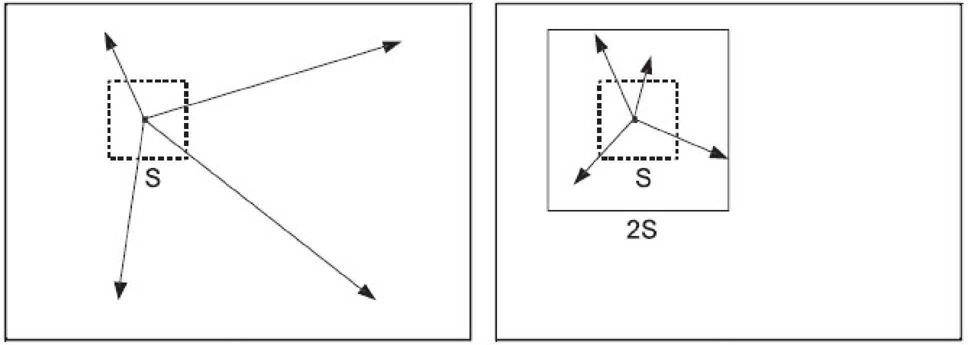
Step 1: 3D SLIC

- Goal: computation of compact super-voxels

- Benefit:
  - Local compactness (reduce fragmentation)
  - make the exhaustive clustering more efficient

Images segmented using SLIC

Slic algorithm



(a) standard k-means searches the entire image

(b) SLIC searches a limited region

Algorithm 1. SLIC superpixel segmentation

/* *Initialization* */

Initialize cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ by sampling pixels at regular grid steps S.

Move cluster centers to the lowest gradient position in a 3×3 neighborhood.

Set label $l(i) = -1$ for each pixel i.

Set distance $d(i) = \infty$ for each pixel i.

repeat

/* *Assignment* */

for each cluster center C_k **do**

for each pixel i in a $2S \times 2S$ region around C_k **do**

Compute the distance D between C_k and i.

if $D < d(i)$ **then**

set $d(i) = D$

set $l(i) = k$

end if

end for

end for

/* *Update* */

Compute new cluster centers.

Compute residual error E.

until $E \leq$ threshold

$$d_c = \sqrt{(l_j - l_i)^2 + (a_j - a_i)^2 + (b_j - b_i)^2},$$
$$d_s = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2},$$
$$D' = \sqrt{\left(\frac{d_c}{N_c}\right)^2 + \left(\frac{d_s}{N_s}\right)^2}.$$

- Dc – distance in color space
- Ds – distance in spatial space
- Ns and Nc, respective maximum distances within a cluster

$$D' = \sqrt{\left(\frac{d_c}{m}\right)^2 + \left(\frac{d_s}{S}\right)^2},$$

- maximum spatial distance expection Ns

  $$N_S = S = \sqrt{(N/K)}.$$

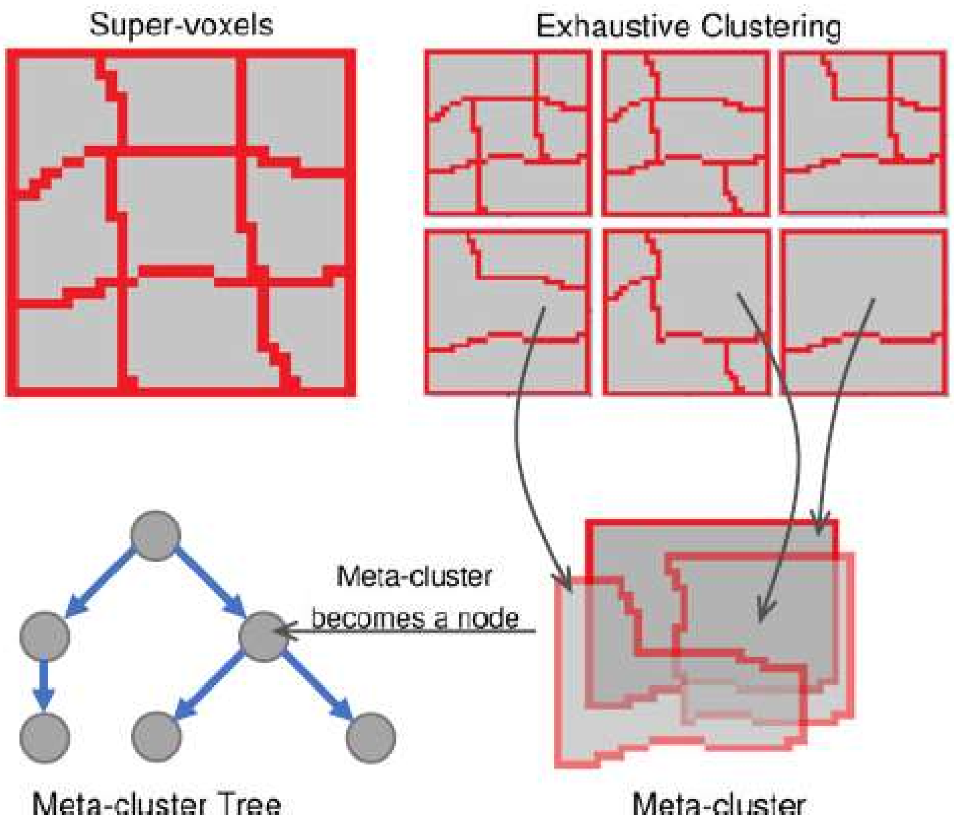
- Nc not straightforward – fix to a constant m.

Algorithm 1. SLIC superpixel segmentation

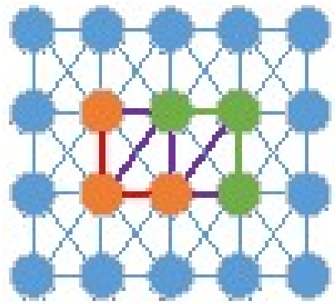
/* *Initialization* */
Initialize cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ by sampling pixels at regular grid steps S.
Move cluster centers to the lowest gradient position in a 3×3 neighborhood.
Set label $l(i) = -1$ for each pixel i.
Set distance $d(i) = \infty$ for each pixel i.

repeat
  /* *Assignment* */
  for each cluster center C_k **do**
    for each pixel i in a $2S \times 2S$ region around C_k **do**
      Compute the distance D between C_k and i.
      if $D < d(i)$ **then**
        set $d(i) = D$
        set $l(i) = k$
      end if
    end for
  end for
  /* *Update* */
  Compute new cluster centers.
  Compute residual error E.
until $E \leq$ threshold

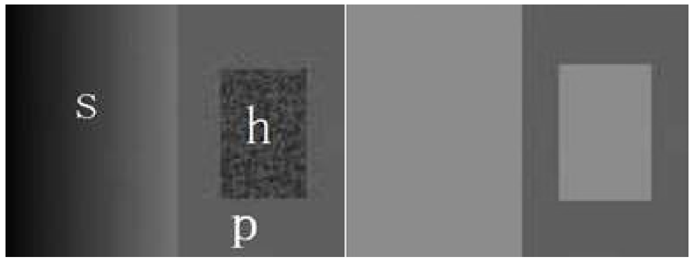
Step 2: Exhaustive-FH Clustering – overview



Step 2: Exhaustive-FH Clustering – overview



- Graph based Segmentation algorithm
- construct adjanceny graph
  - Pixels –> nodes
  - intensity difference -> edge weights
  - Clusters -> MST



- Try to combine clusters to new clusters
- Global threshold? No
- Adapative threshold? Yes
  - p small, s a little bigger, h super big

Step 2: Exhaustive-FH Clustering – overview

- Internal variation

$$Int(C) = \max_{e \in MST(C,E)} w(e) \ .$$

- Difference between clusters – the most similar place
  - w(C1, C2) is the minimum edge weight connecting regions C1 and C2

- the way decide whether combine or not

$$w(C_1, C_2) \leq \min\Big(\mathrm{Int}(C_1) + \tau(C_1), \mathrm{Int}(C_2) + \tau(C_2)\Big). \quad (1)$$

- Special case
  - Initially Int(C1) = 0 – zero tolerance
  - Over-segmentation
  - Add more tolerance

$$\tau(C) = k/|C|$$

  - |C| denote size of C, k ∈ [0, +∞) is contant number (the only hyperparmeter)

Algorithm steps

- 1. construct adjanceny graph
  - Pixels –> nodes intensity difference -> edge weights
- 2. sort edge-list by weight
- 3. collapse edge e (merge adjancent regions C1 and C2)

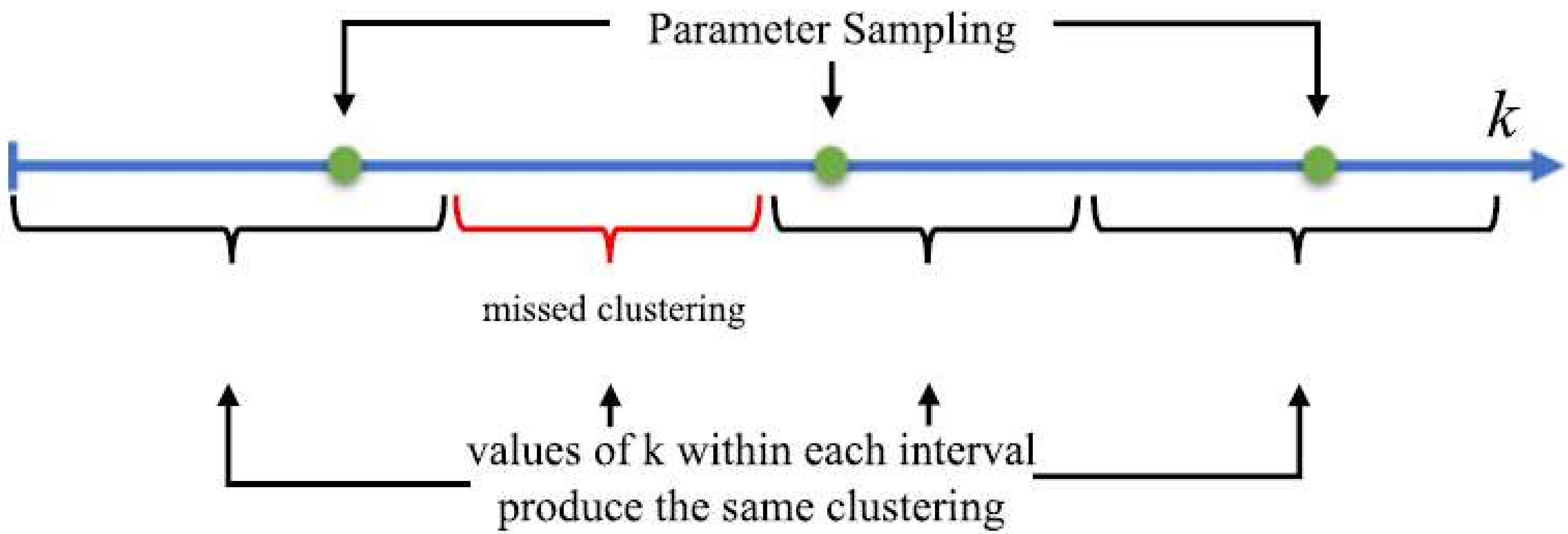
$$Int(C) = \max_{e \in MST(C,E)} w(e) \ .$$

$$w(C_1, C_2) \leq \min\Big(\mathrm{Int}(C_1) + \tau(C_1), \mathrm{Int}(C_2) + \tau(C_2)\Big). \quad (1)$$

$$\tau(C) = k/|C|$$

- Bigger k -> larger clusters

Interval tracking



FH method in the paper

- 1. construct adjanceny graph
  - **Super-pixels** –> nodes edge weights computed based on super-pixels
- 2. sort edge-list by weight
- 3. collapse edge e (merge adjancent regions C1 and C2)

- Instead of **parameter sampling**, they use **interval tracking**
- Use properties to prove the correctness

compute distance between super-voxels

- Using the chi-squared distance between 1D intensity histograms
- Each histogram:
  - A total of 64 bins across the entire scalar range of the input volume
- Chi-squared distance

$$d(j, j') = \sqrt{\sum_{i=1}^{n} \left(\frac{f_{ij}}{f_{.j}} - \frac{f_{ij'}}{f_{.j'}} \right)^2 \cdot \frac{1}{f_{i.}}} \ ,$$

$f_{i.}$	is the sum of the components of the ith row;
$f_{.j}$	is the sum of the components of the jth column;

Chi-squared distance – example

	Business 1	Business 2	Business 3	Total
X_1	0.1	0.275	0.15	0.525
X_2	0.09	0.2	0.075	0.365
X_3	0.06	0.025	0.025	0.11
Total	0.25	0.5	0.25	1

Frequency

$$
\begin{aligned}
d^2(1,2) &= \frac{1}{0.525} \cdot (0.4 - 0.55)^2 \\
&\quad + \frac{1}{0.365} \cdot (0.36 - 0.4)^2 \\
&\quad + \frac{1}{0.11} \cdot (0.24 - 0.05)^2 \\
&= 0.375423 \\
d(1,2) &= 0.613
\end{aligned}
$$

The way calculating the distance

$$d(j, j') = \sqrt{\sum_{i=1}^{n} \left(\frac{f_{ij}}{f_{.j}} - \frac{f_{ij'}}{f_{.j'}} \right)^2 \cdot \frac{1}{f_{i.}}},$$

$f_{i.}$ is the sum of the components of the ith row;

$f_{.j}$ is the sum of the components of the jth column;

Some term

$$w(C_1, C_2) \le \min\Big(\mathrm{Int}(C_1) + \tau(C_1), \mathrm{Int}(C_2) + \tau(C_2)\Big). \quad (1)$$

- Edge satisfy condition (1) -> this edge is merged
  - Called *edge collapse*

- During every operation, the order of edge list is always the same
  - Can be done by pre-computing the list of edges and pass the same list to all exections

Property1

$$w(C_1, C_2) \le \min\Big(\mathrm{Int}(C_1) + \tau(C_1), \mathrm{Int}(C_2) + \tau(C_2)\Big). \quad (1)$$

- If *k=a* collapse an edge -> all *k⩾a* will also collapse that edge
- If *k=b* does not collapse an edge -> all *k⩽b* will also not collapse that edge
- If an edge is collapsed for *k = ks* -> all values of *k > ks* will also collapse that edge
- If an edge is not collapsed for *k = ks,* -> exist some ke can collapse that edge

$$k_e = \min\Big((w(C_1, C_2) - Int(C_1)) \cdot |C_1|, \\ (w(C_1, C_2) - Int(C_2)) \cdot |C_2| \Big). \quad (2)$$

Property2

- For $ki \neq kj$, both executions have made the same decisions up to an edge $e_n \in E$ -> both ki, kj encounter the same value of $|C1|$ and $|C2|$ for e_{n+1}
- My opinion: not obvious, since this condition do not guarantee if both executions have made the same decisions up to e_1 to e_{n-1}
- Can modidfy the condition to
  - have made the same decisions up to edges e_1 to e_n
- P1 & modified p2 -> for all values $k \in [ks, ke)$, collapse the same edges in E, and in the same order

Property3

- For $ki \neq kj$, all decisions to collapse edges $e \in E$ are the same -> the resulting clusterings of both executions are equivalent
- P3 -> all values of k in the final interval $[ks, ke)$ produce the same clustering

- If one edge not collapse -> update the interval

Algorithm 1. Exhaustive FH Clustering

```
Construct compact super-voxels B;
Construct adjacency graph G(V, E) for super-voxels;
Sort E by non-decreasing order of edge-weights;
Initialize k ← {0, ∞};
do
  S ← B;
  foreach e : E do
    if e.weight ≤ k[0] then
      Merge regions connected by e in S;
    else
      Calculate k_e using Eq. (2);
      if k_e < k[1] then
        k[1] ← k_e
  OutputList.insert( S );
  k ← {k[1], ∞};
while S.RegionCount > 1;
return OutputList;
```

$$k_e = \min\Big((w(C_1, C_2) - Int(C_1)) \cdot |C_1|, \; (w(C_1, C_2) - Int(C_2)) \cdot |C_2| \Big). \tag{2}$$

Step3: Construction of Meta-Cluster Hierarchy

- Computing Meta-Clusters
  - MST-based clustering algorithm called reverse-delete
  - distance measure the overlap between different clusters
  - Jaccard distance

$$d_J(r_i, r_j) = 1 - \frac{|r_i \cap r_j|}{|r_i \cup r_j|},$$

    - Voxel level (not super-voxel) -- quantify the actual sizes of regions in the volume.
  - Delete edge repeatedly until dJ < t
    - t – user-provided dissimilarity threshold = 0.3

Step3: Construction of Meta-Cluster Hierarchy

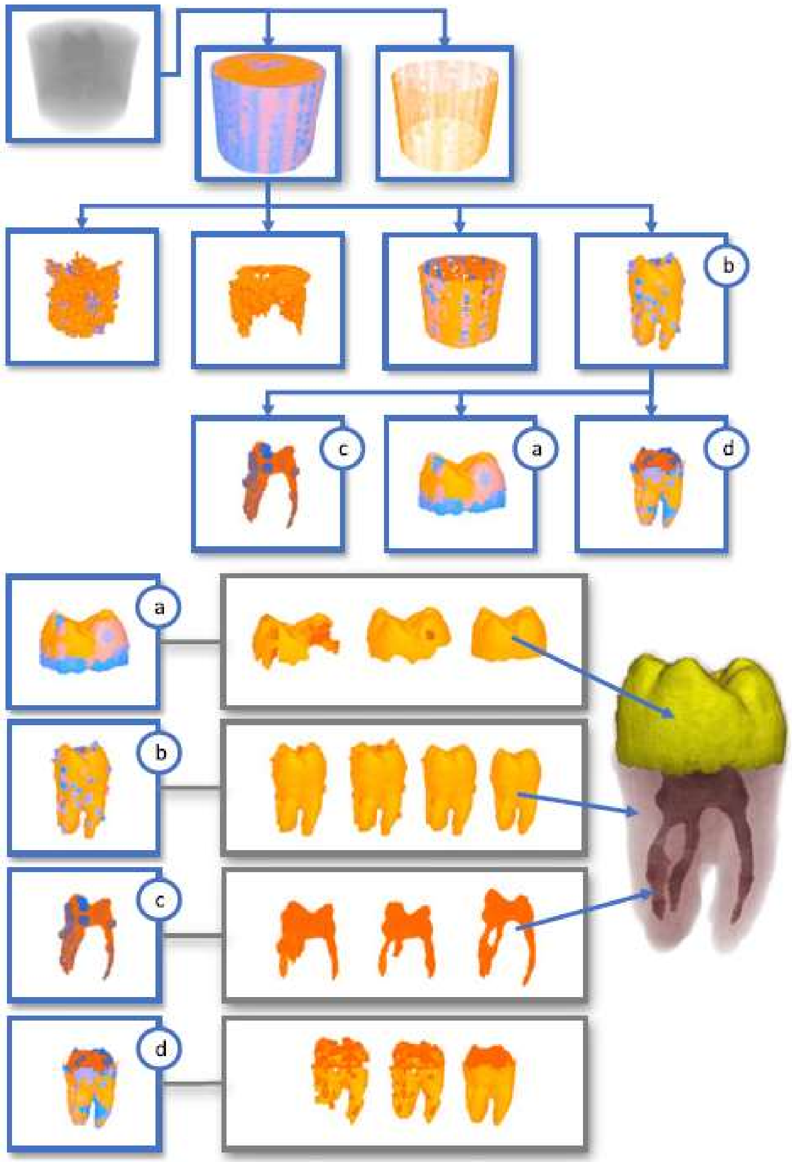
- Tree construction

Algorithm 2. Meta-Cluster Tree Construction

Compute meta-clusters $\{v_i\}$
for each v_i **do**
  Construct superset list $\{s_j\}$ such that $v_i \subseteq s_j$
  Sort superset list by increasing sizes
  if $\{s_j\}$.*NotEmpty* **then**
    Find smallest superset s_f
  else
    $s_f = root$
  Construct edge (v_i, s_f)
for each v_i **do**
  for each superset s_j **do**
    if s_j is not an ancestor of v_i **then**
      Create node v_i' as duplicate of v_i
      Construct edge (v_i', s_j)
for each v_i **do**
  Sort child nodes by size in descending order.

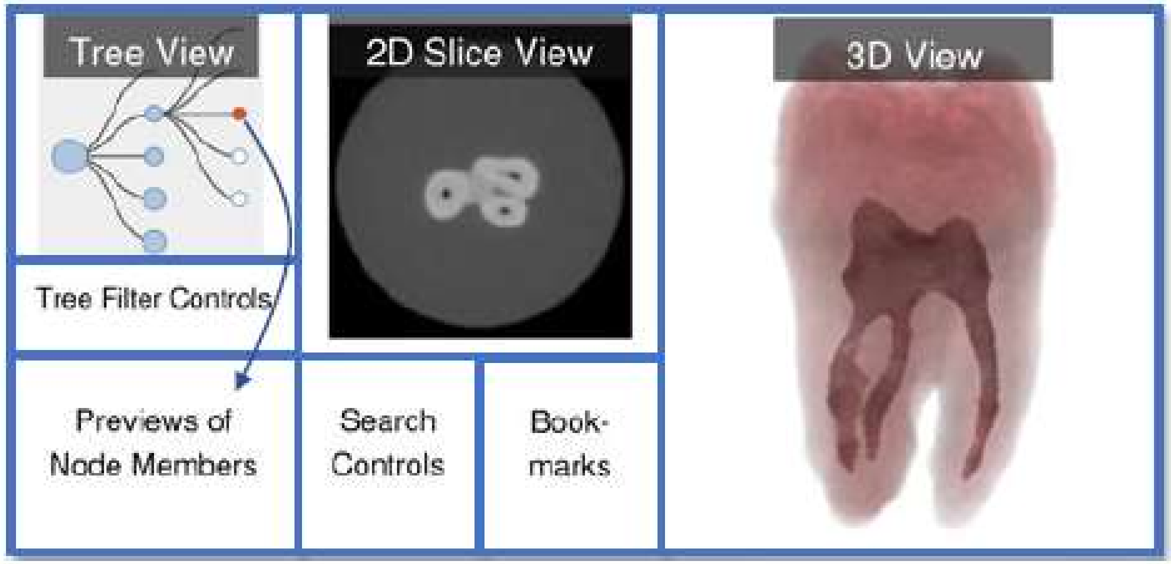
meta-cluster tree for the Tooth dataset

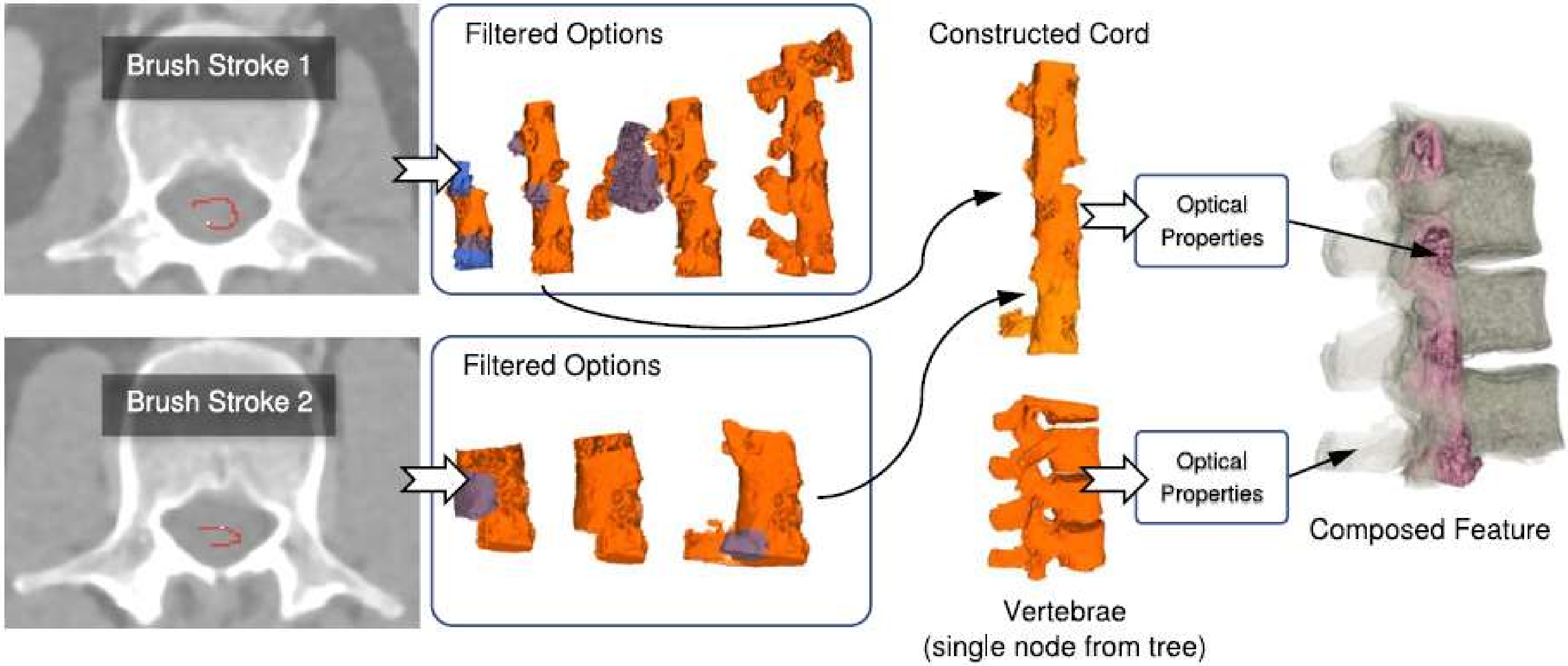
- Root – entire volume
- first level nodes – broad seperatioin between ROI & noise
- (b) the entire tooth
  - (c) dentine,
  - (a) root canals
  - (d) the crown.
- Further shows clusters of (a)(b)(c)(d)



User interface

- Tree view – meta-cluster tree
- Node member view – show selected node
- Tree prune control & search control
  - dynamically prune the tree based on min meta-tree size & max branch factor
- Bookmarks – store selected results
- Search Controls
  - Search features based through brushing & size constraints

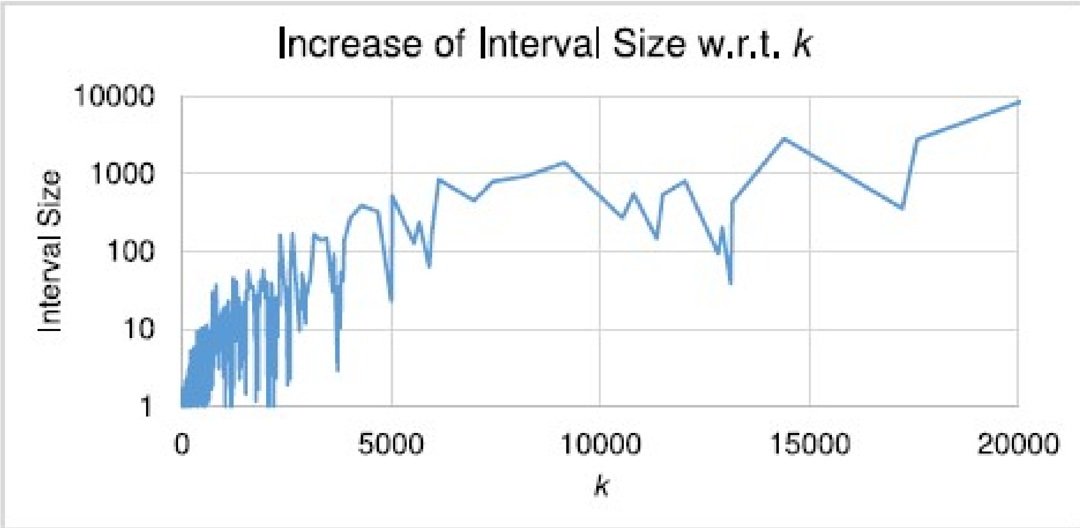




- Brush select regions
- Botton-up search
- Filter based on meta-cluster size (eg. 1,000 to 100,000)

Implementation

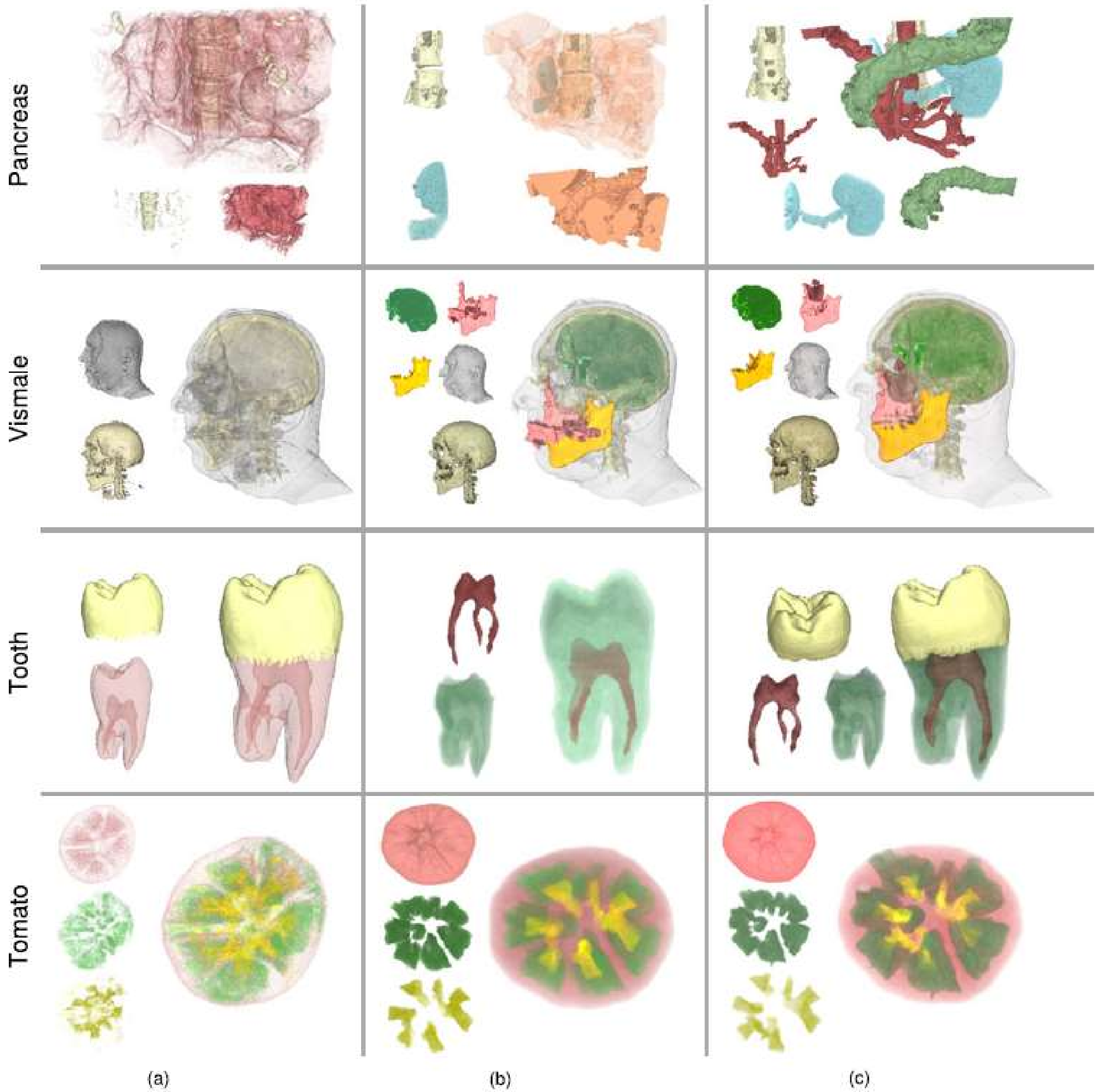
- SLIC – extending the GPU implementation of SLIC to 3D volumes
- FH cluster
  - Multi-threading
  - Each thread pass a range of k
  - Based on the figure
    - Smaller contiguous intervals when k close to 0 -> Require more iterations
  - goal: workload balance
  - Pass ranges in increasing size
  - E.g thread 1 [0, 50), thread2 [50, 125),



Qualitative Evaluation

- (a) voxel-clustering through normalized-cut of intensity-gradient histograms
- (b) FeatureLego with parameter sampling
- (c) FeatureLego with exhaustive clustering.

- (a) limited in separating features & features are fragmented
- (b) miss out on some features



the effect of base granularity

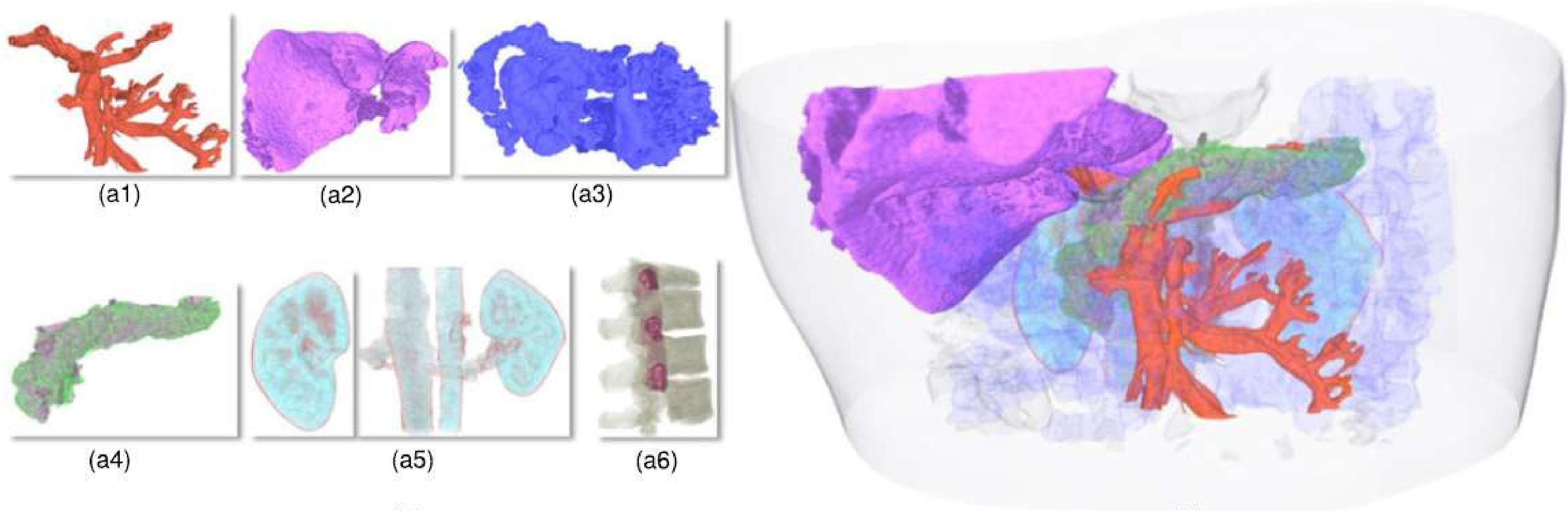
(a)
SV Size = 512
Regions = 1

(b)
SV Size = 2197
Regions = 6

(c)
SV Size = 4096
Regions = 2

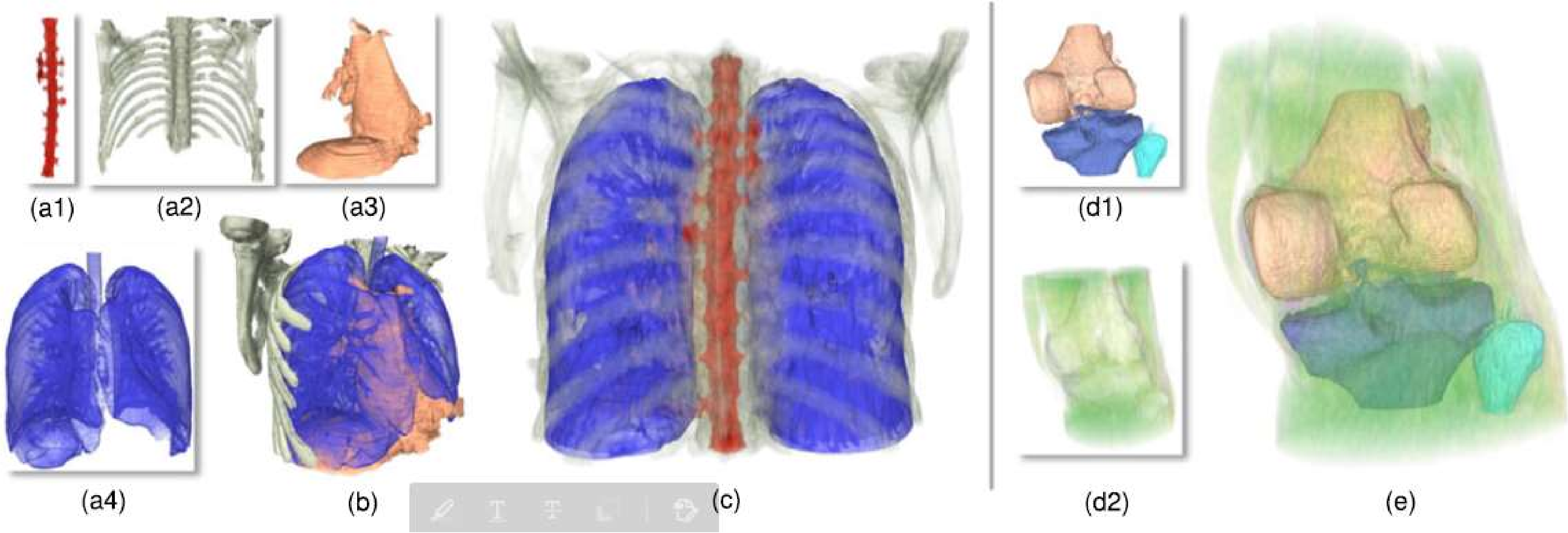
- root canals are impossible to separate cleanly (b and c) until super-voxel size is lowered to 512 (a).

Abdominal CT scan



(a) (b)

Chest CTand Knee MRI scans.



Performace

Pre-processing time required at each stage

Dataset	Dimensions	Super-Voxels	Exhaustive Clustering	Meta-Cluster Tree	Total Time
Tomato	256 x 256 x 64	17 sec	17 sec	3 sec	37 sec
Pancreas	235 x 153 x 210	30 sec	38 sec	22 sec	1.5 min
Vismale	128 x 256 x 256	33 sec	2.1 min	4.4 min	7 min
Tooth	256 x 256 x 161	42 sec	74 sec	86 sec	3.4 min
Knee MRI	512 x 512 x 120	2 min	7.3 min	7.9 min	17.2 min
Chest CT	384 x 384 x 240	2.4 min	4.1 min	4.8 min	11.3 min
Abdominal CT	504 x 416 x 243	3.4 min	6.7 min	10.2 min	20.3 min

Summary

- Novel exhaustive clustering pipeline
- Provides more choices for selecting / visualization
- Exhaustive clustering is performed as pre-processing
- Meta-cluster tree -> efficiently explore volume