EFFICIENT PARALLEL GRAPH-BASED IMAGE SEGMENTATION
13th International Conference on Computer Analysis of Images and Patterns
September 2nd - September 4th, 2009

Relevant topics:
- Segmentation and grouping
- Graph-based methods
- Object recognition
EFFICIENT PARALLEL GRAPH-BASED IMAGE SEGMENTATION

- MST-based segmentation
  - Introduction
  - Heuristic
  - Results

- Edge-preserving smoothing

- Parallel MST algorithm
Segmentation

Find coherent regions in image – especially buildings

(difficulty: low contrast)

Applications: change detection, ...

Must process 100 MPixel datasets within seconds
Goals (December 2008)

- MSER algorithm (2008) not yet sufficient
  - About 4x slower than satellite and UAV downlink

- MST-based algorithm avoids need for gradient

- Semi-parallel MST algorithm

- Fully independent parallelization
New PHMSF algorithm

Parallel Heuristic for Minimum Spanning Forests:

**Objective**  efficient segmentation (automatically grouping similar pixels into regions).

**Method**  heuristic cuts within a MST (Minimum Spanning Tree) representation.

**Improvement**  parallel processing without truncating objects at tile boundaries.

**Benefit**  accurate and high-performance object extraction from large-scale images.
Image representation

- Edge weights indicate dissimilarity
- No gradient needed
- Allows segmentation via MST
- Caveat: memory use (4 bytes/edge)

Weighted grid graph

© Fraunhofer FOM
Minimum Spanning Tree

- Cheapest set of edges connecting all nodes

- Construct in quasi-linear time:
  NonComparisonSort(edges)
  for (u,v) in edges:
    if not IsCycle(u,v):
      AddToMST(u,v)

- Amenable to parallelization...
MST-based segmentation

- Cut 'suitable' edges (→ next slides)
- Yields several trees, i.e. a forest
- Segment := tree

Minimum Spanning Forest

Segment1  Cut  Segment2
Graph-cut heuristic (1)

- No single threshold for edge weights exists
- \( \rightarrow \) hysteresis idea from Canny’s edge operator
- Connected component finds very homogeneous regions:

Expand by adding nearby edges...
Graph-cut heuristic (2)

Stop when sum of edge weights exceeds region’s credit:

Tends to avoid global over- and undersegmentation!
Graph-cut heuristic (3)

Lower bound on region perimeter:

1. Use initial regions from connected-component output
2. Minimum perimeter via equal-area circle
Results (USC SIPI database)

Input images

Mean Shift

PHMSF (50x as fast)
Results (Quickbird)

Pan-sharpened close-up of Siemens

Label image

3 pixel wide roof structures
Results (Mini-UAV)

Mean Shift (~2500 ms)  Filter + PHMSF (~50 ms)
Results (LUNA 1)

LUNA frame

Segmentation (mean color)
Results (LUNA 2)

LUNA frame

Segmentation (mean color)
Edge-preserving smoothing

Goals:
- Help segmentation algorithm cope with noise (i.e. improve pixel homogeneity)
- Preserve edges

Standard algorithm: anisotropic diffusion
- Computationally expensive
- No general stop criterion

Noisy data (IKONOS)
Bilateral Filter

Bilateral := spatial and radiometric distance

Gauss smoothing while excluding edges:

Non-iterative, can be approximated efficiently (200 MPixel/s)
Parallelization: Motivation

- A highly-tuned implementation can’t keep up with sensor data rates;

- recent CPU advances chiefly benefit multi-threaded programs;

- data-parallel schemes truncate objects at tile boundaries.
Parallelization: Idea

Each processor generates partial MSTs until a cross-border edge is reached:
Parallelization: Algorithm

parallel foreach tile do
  sort edges, merging those with weight < minWeight;

foreach borderEdge do  // connect and mark cross-border regions
  region₁, region₂ := Find(borderEdge.endpoints);
  survivor := Union(region₁, region₂);
  Mark(survivor);
  tile.regions := tile.regions ∪ \{survivor\};

parallel foreach tile do
  foreach r ∈ tile.regions do  r.credit := ComputeCredit(r.size);

parallel foreach tile do
  foreach edge in ascending order of weight do
    region₁, region₂ := Find(edge.endpoints);
    if edge crosses border then Mark(region₁); Mark(region₂);
    else if IsMarked(region₁, region₂) then tile.delayQ.push(edge);
    else EdgeHeuristic(edge);

foreach tile do
  foreach edge ∈ tile.delayQ do EdgeHeuristic(edge);
Microarchitecture-aware sorting

Single-pass (!) variant of counting sort:

Write-combining without cache pollution:

Improves scalability due to decreased memory traffic.
Performance (1)

The new algorithm has quasi-linear complexity.
Constant factors are low due to parallelization and SIMD processing:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MPixel/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSHLK</td>
<td>(out of memory)</td>
</tr>
<tr>
<td>Mean Shift</td>
<td>0.09</td>
</tr>
<tr>
<td>Graph-Based</td>
<td>0.45</td>
</tr>
<tr>
<td>MSER</td>
<td>2.53</td>
</tr>
<tr>
<td><strong>PHMSF</strong></td>
<td><strong>12.80</strong></td>
</tr>
</tbody>
</table>

Performance on an 8.3 MPixel subset (Intel X5365 CPU)
Performance increases with image size (interiors grow faster than borders):

<table>
<thead>
<tr>
<th>Sensor</th>
<th>MPixel</th>
<th>MPixel/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>IKONOS</td>
<td>54</td>
<td>13.5</td>
</tr>
<tr>
<td>Quickbird</td>
<td>219</td>
<td>14.3</td>
</tr>
<tr>
<td>JAS150s</td>
<td>527</td>
<td>24.4</td>
</tr>
</tbody>
</table>

Performance on large datasets (Intel X5365 CPU)

Limited by memory bandwidth; two Intel W5580 CPUs manage 40 MPixel/s!
Software Engineering

- 3500 lines (C++), 9000 including dependencies.
- Parallelization via OpenMP with CPU affinity.
- Vectorization via SSE2 intrinsics.
- Requires 64-bit address space.
- Available in the form of DLLs.
- Theoretically portable to POSIX systems.
Conclusion

- First non-trivial parallel segmentation algorithm; (proper handling of objects on boundaries)

- orders of magnitude faster than existing methods with similar quality;

- results useful for object-based change detection, etc.