Approximating the Generalized Voronoi Diagram of Closely Spaced Objects

John Edwards[1] Eric Daniel[2] Valerio Pascucci[1] Chandrajit Bajaj[3]

[1] SCI Institute, University of Utah

[2] Google, Inc.

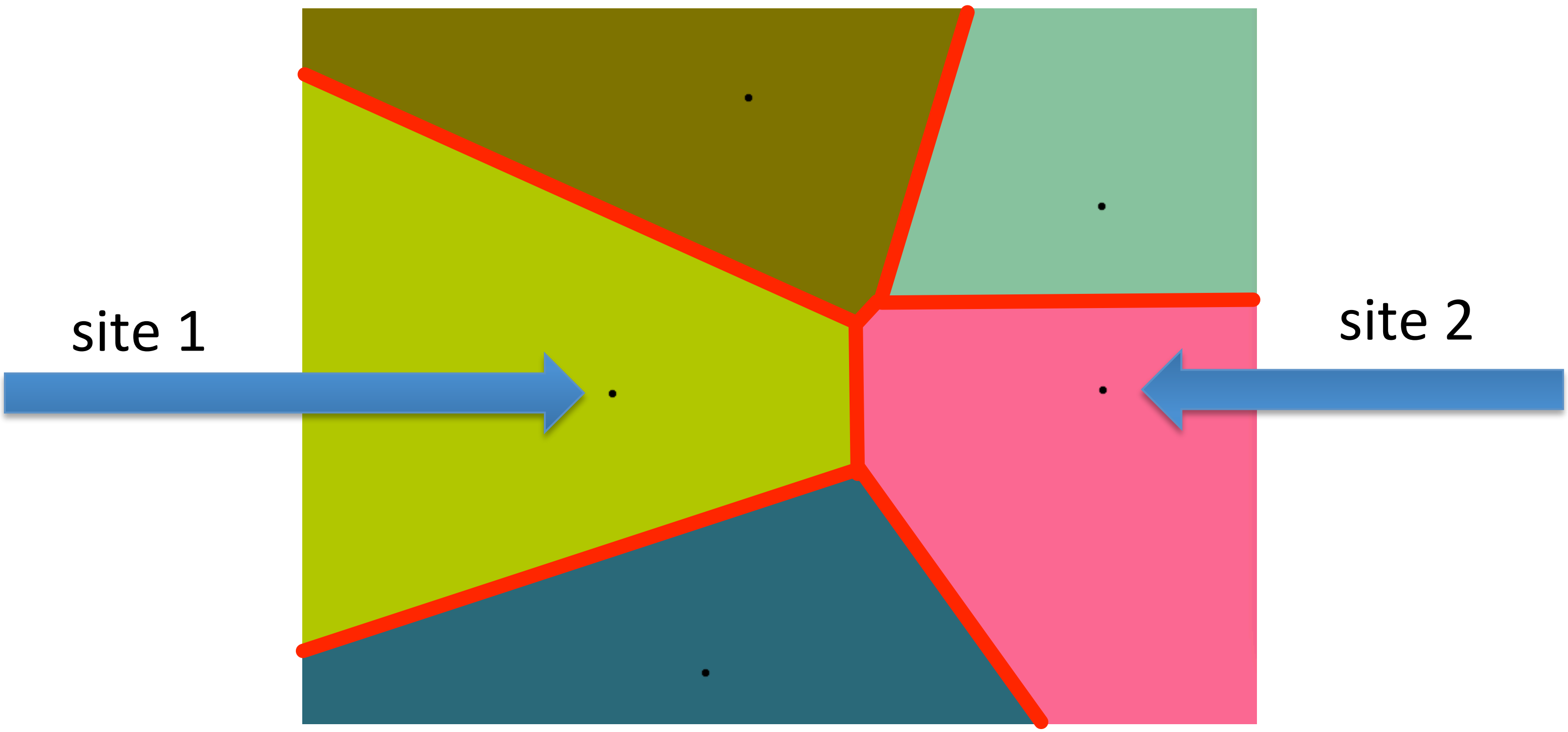
[3] The University of Texas

Eurographics

May 6, 2015

Voronoi Diagram

Sites are points

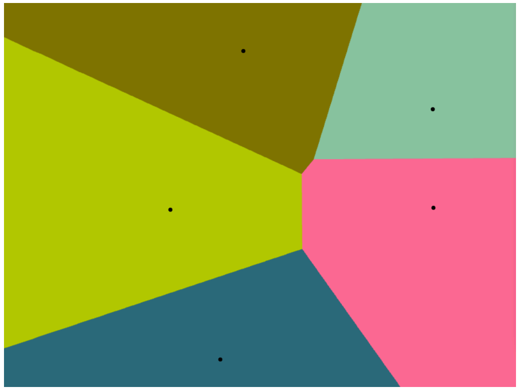


http://alexbeutel.com/webgl/voronoi.html

The Voronoi diagram:

1. is the locus of points equidistant from at least 2 sites
2. is a union of line segments

Voronoi Diagram



- Applications: GIS, biology, geology, physiology, crystallography…
- Exact computation algorithms are simple and fast. Fortune's algorithm:
  - O(N log N) time
  - O(N) space

Generalized Voronoi Diagram (GVD)

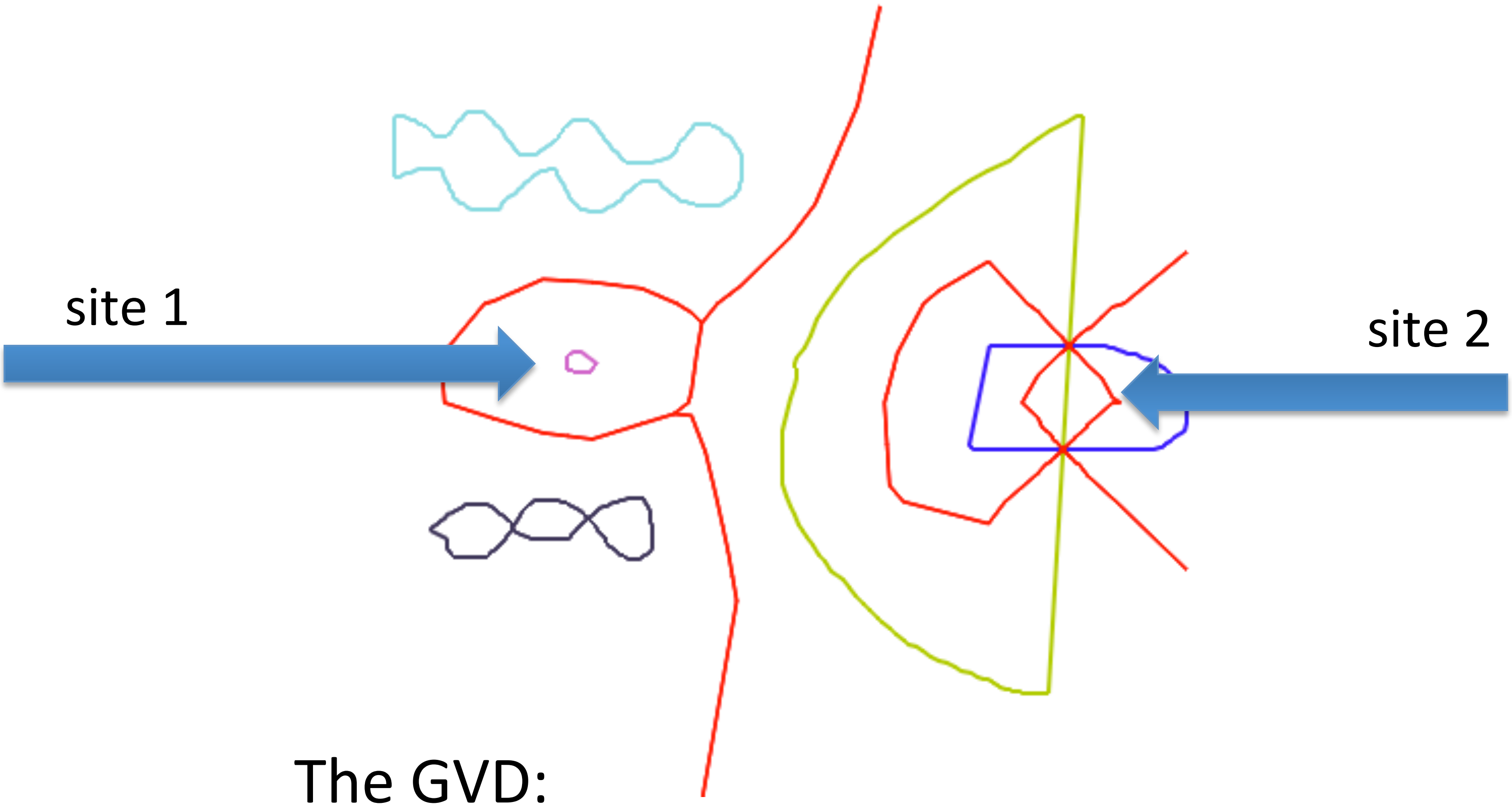
Sites are arbitrary objects

The GVD:

1. is the locus of points equidistant from at least 2 sites
2. is a union of line and (often complex) curve segments

Generalized Voronoi Diagram (GVD)

Sites are arbitrary objects

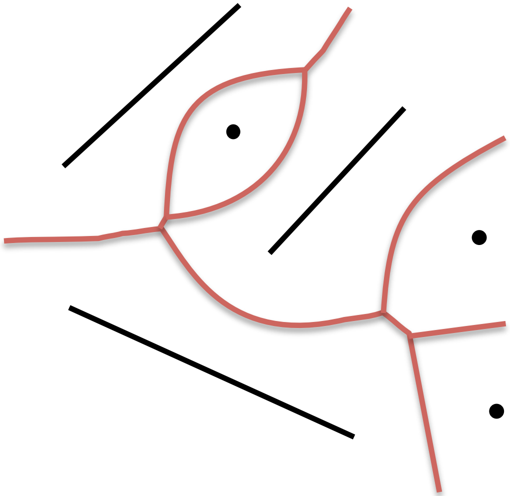


The GVD:

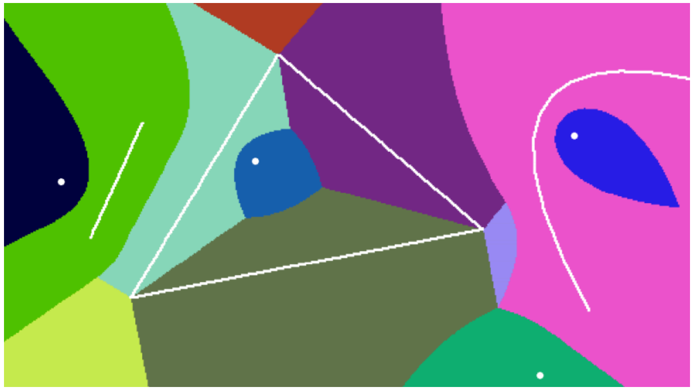
1. is the locus of points equidistant from at least 2 sites
2. is a union of line and (often complex) curve segments

Generalized Voronoi Diagram (GVD)

- Exact computation algorithms
  - Line and point sites only
    - (GVD composed of lines and parabolas)
  - Lee 1982, Karavelas 2004



- Approximation algorithms
  - Arbitrary sites; most are uniformly gridded
  - Hoff et al 1999, Cao et al 2010, etc.

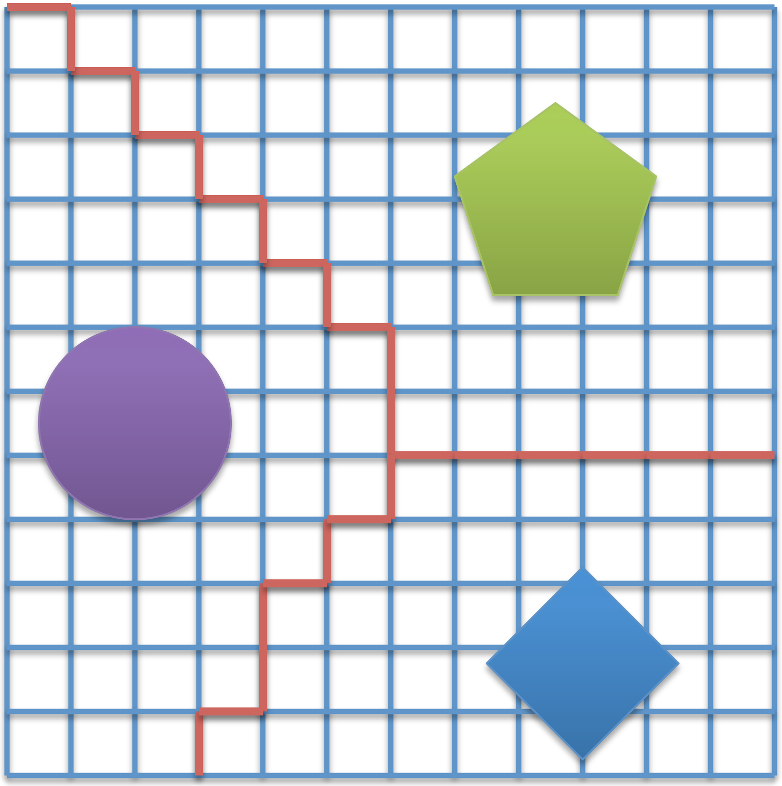


Hoff et al 1999

GVD – uniform gridding

Advantages:

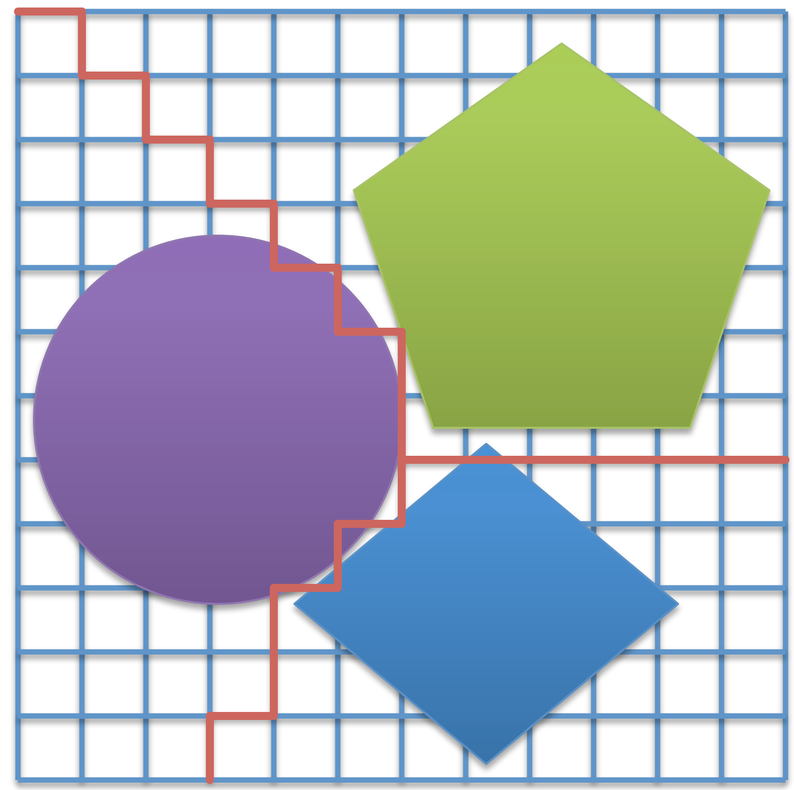
- Simple
- Fast
- Suitable for GPGPU implementations



GVD – uniform gridding

Disadvantages:

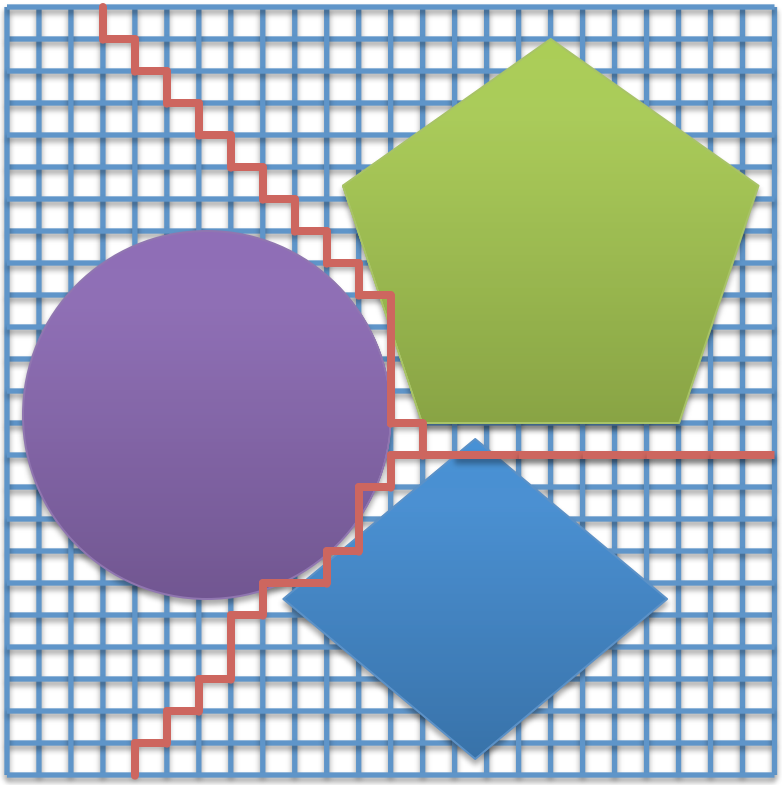
- Object spacings not normally known beforehand
- Resolution may not be high enough



GVD – uniform gridding

Disadvantages:

- Object spacings not normally known beforehand
- Resolution may not be high enough
- Grid may not fit in memory



Uniform vs. Adaptive

- Uniform gridding:
  - Bunny requires 2^{24} cells
- Adaptive gridding:
  - Bunny requires 7K octree cells
  - Previous work
    - Boada et al 2002, 2008 (connected regions only)
    - Teichmann and Teller 1997; Vleugels and Overmars 1998 (convex sites only)

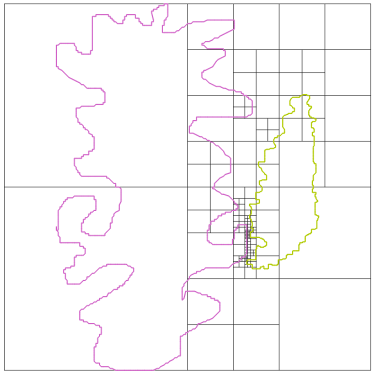
Objective

Our objective is to compute the GVD on datasets…

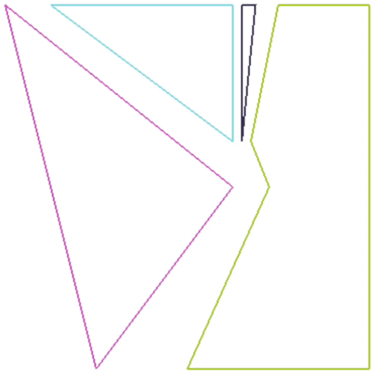
- with closely spaced objects
- with no shape restrictions
  - disconnected
  - non-manifold
  - self-intersections
  - inter-object intersections
- 2D and 3D
- in reasonable time on commodity hardware

Heart model courtesy Jonathan Bronson

Contributions

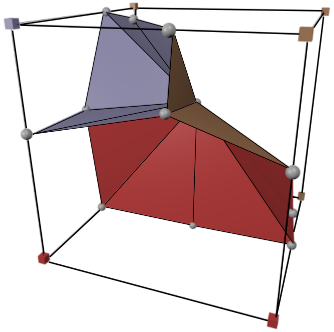


Octree models inter-object space using adjacency structure



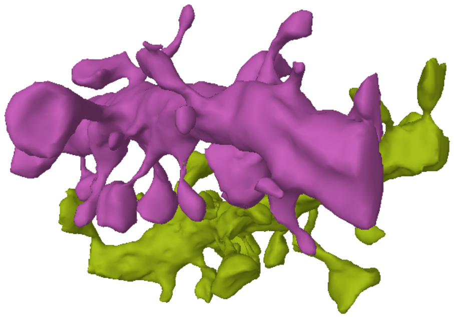
Wavefront distance transform on octree

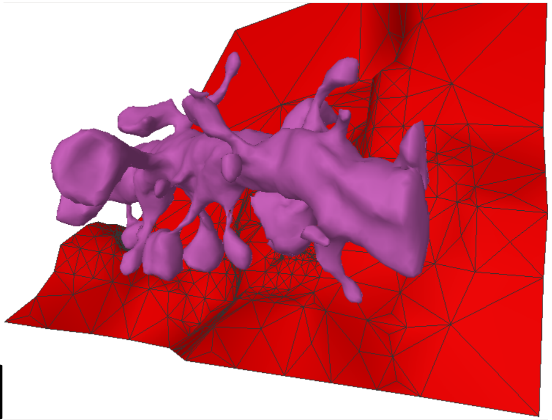
GVD surfacing algorithm on labeled octree



Contributions

- **Octree** decomposition of space
  - Models inter-object space (rather than object features)
  - Adjacency structure (rather than hierarchical) for fast neighbor queries
- **Wavefront distance transform** on octree
  - Conjectured to be 3/2-approximation
- **GVD surfacing** algorithm on labeled octree

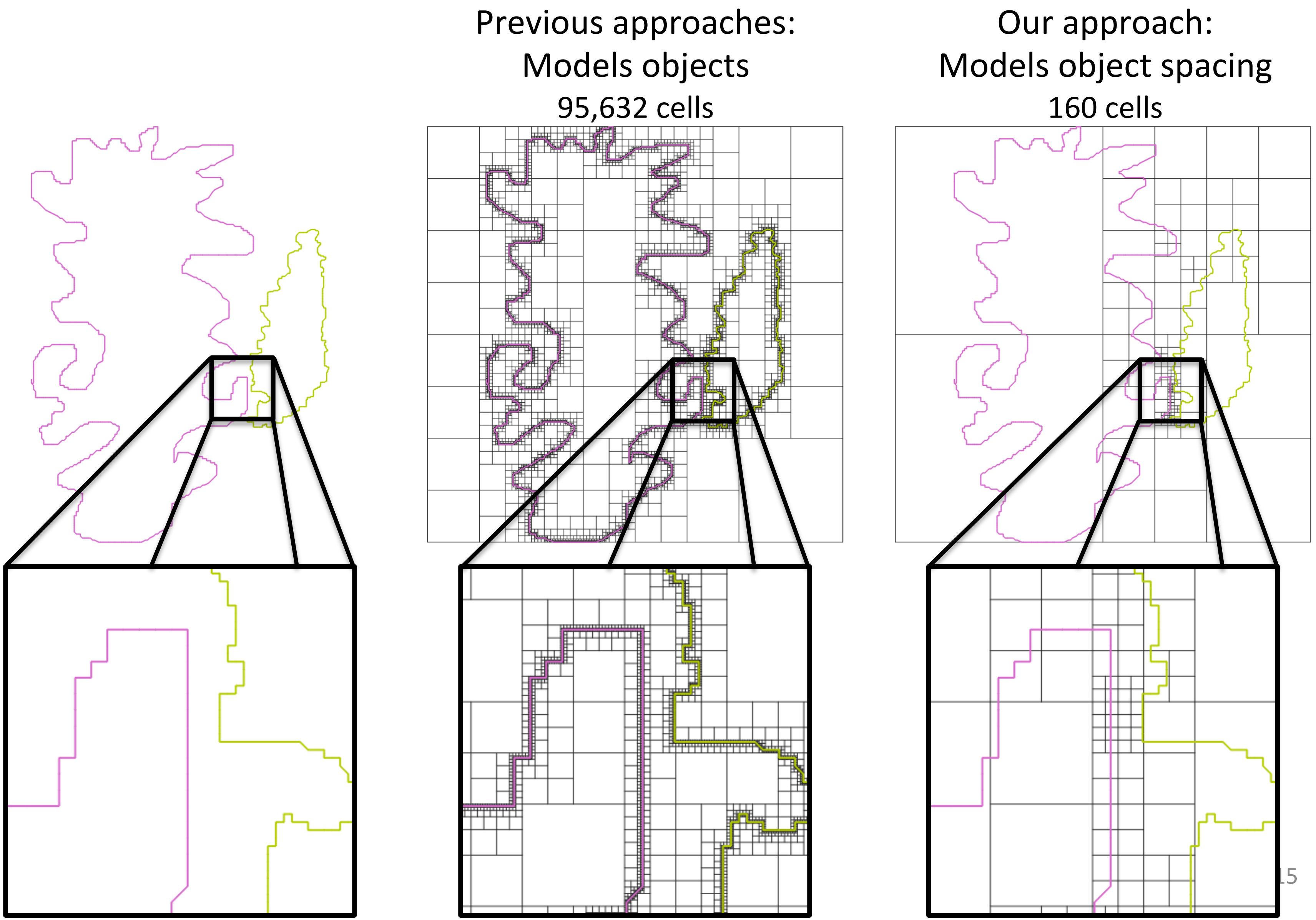




Octree

Note: In this talk I will use “octree” to refer to both quadtree and octree

Octree

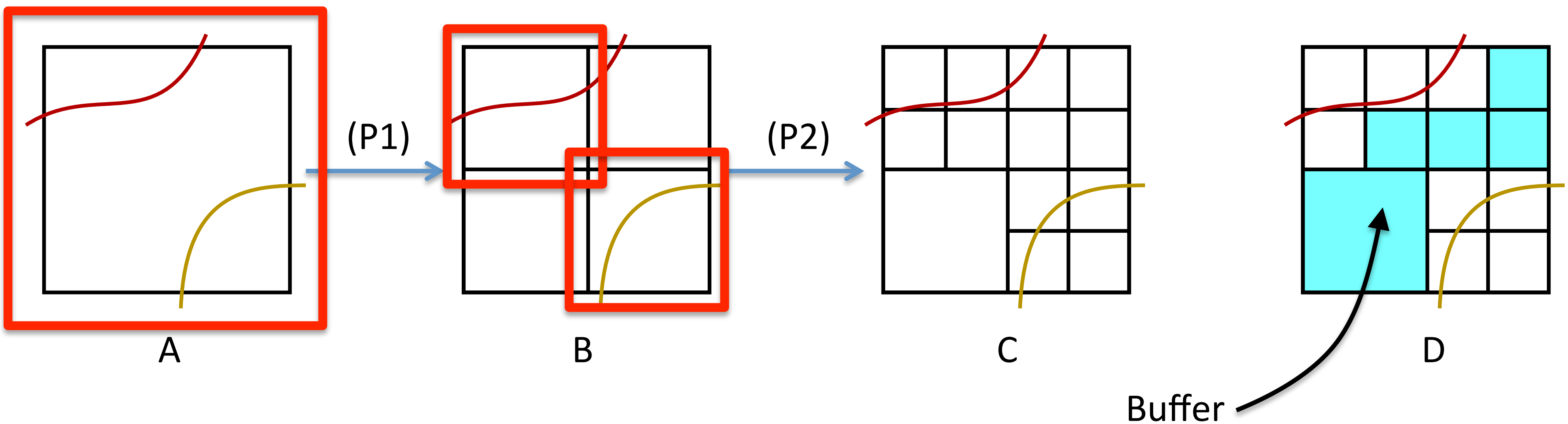

Previous approaches:
Models objects
95,632 cells
Our approach:
Models object spacing
160 cells

Octree – subdivision predicate

Subdivide cell *c* if

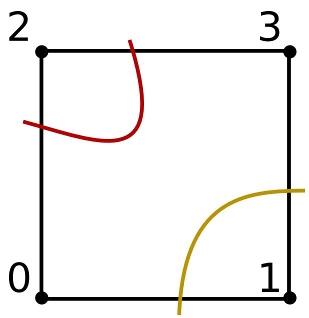
(P1) cell *c* intersects more than one object

(P2) a neighbor of *c* intersects a different object

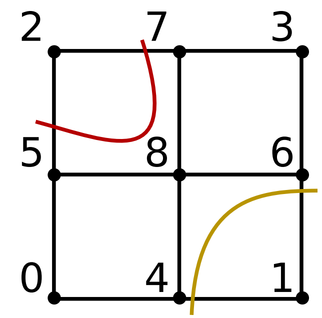


Octree – data structure

Cell vertices store neighbors – no hierarchy



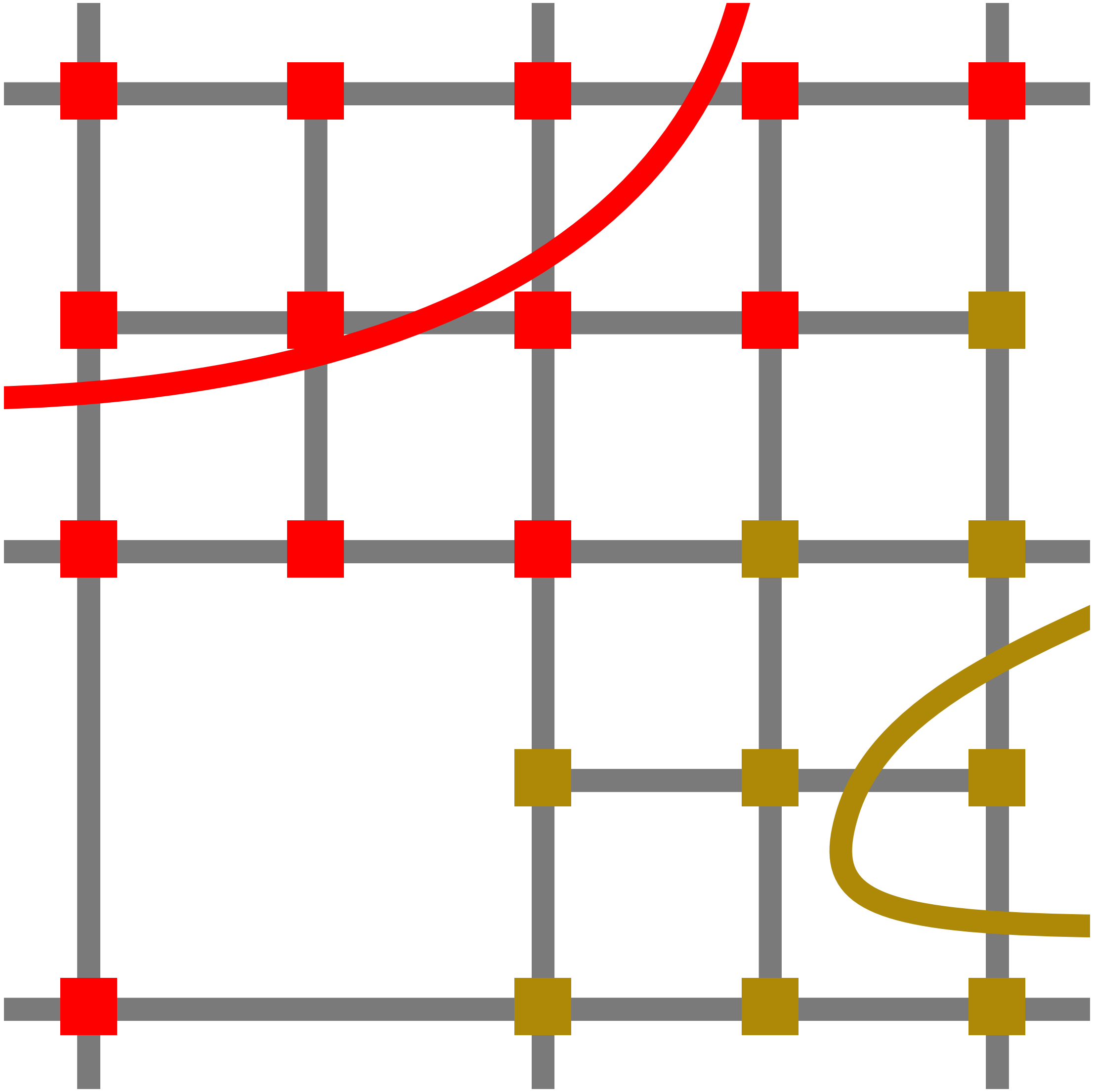
	-x	+x	-y	+y
nbrs(0) =		1		2
nbrs(1) =	0			3
nbrs(2) =		3	0	
nbrs(3) =	2		1	



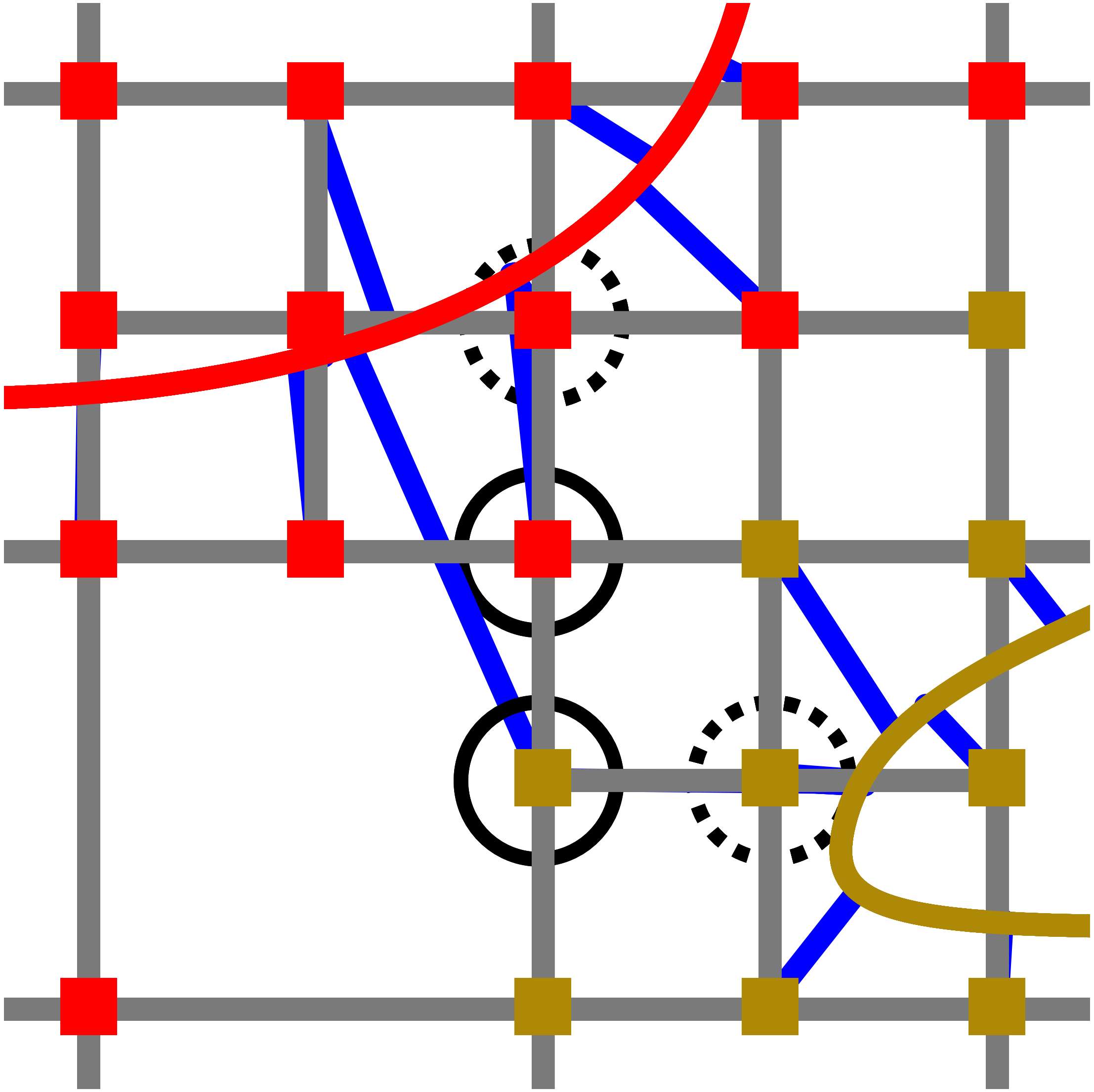
	-x	+x	-y	+y
nbrs(0) =		4		5
nbrs(1) =	4			6
…				
nbrs(8) =	5	6	4	7

	Neighbor finding	**Point location**
Hierarchical	O(log N)	O(log N)
Flat (ours)	O(1)	O(N)

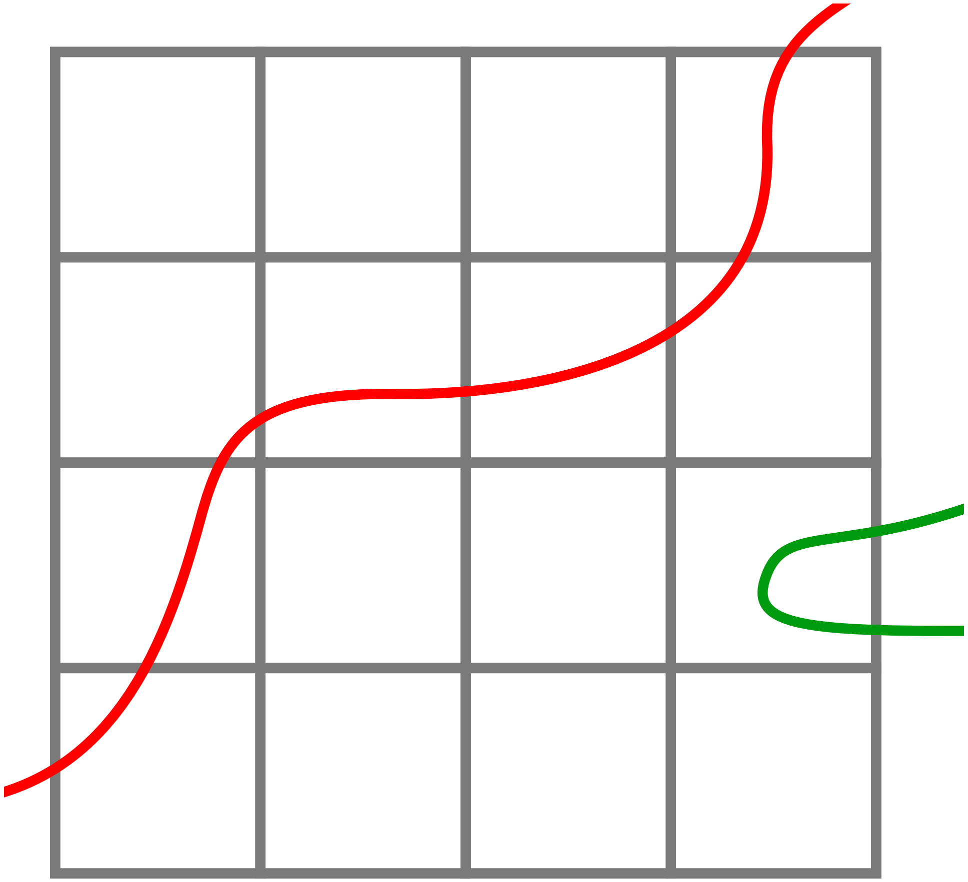
Wavefront distance transform



Wavefront distance transform

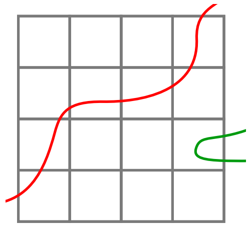


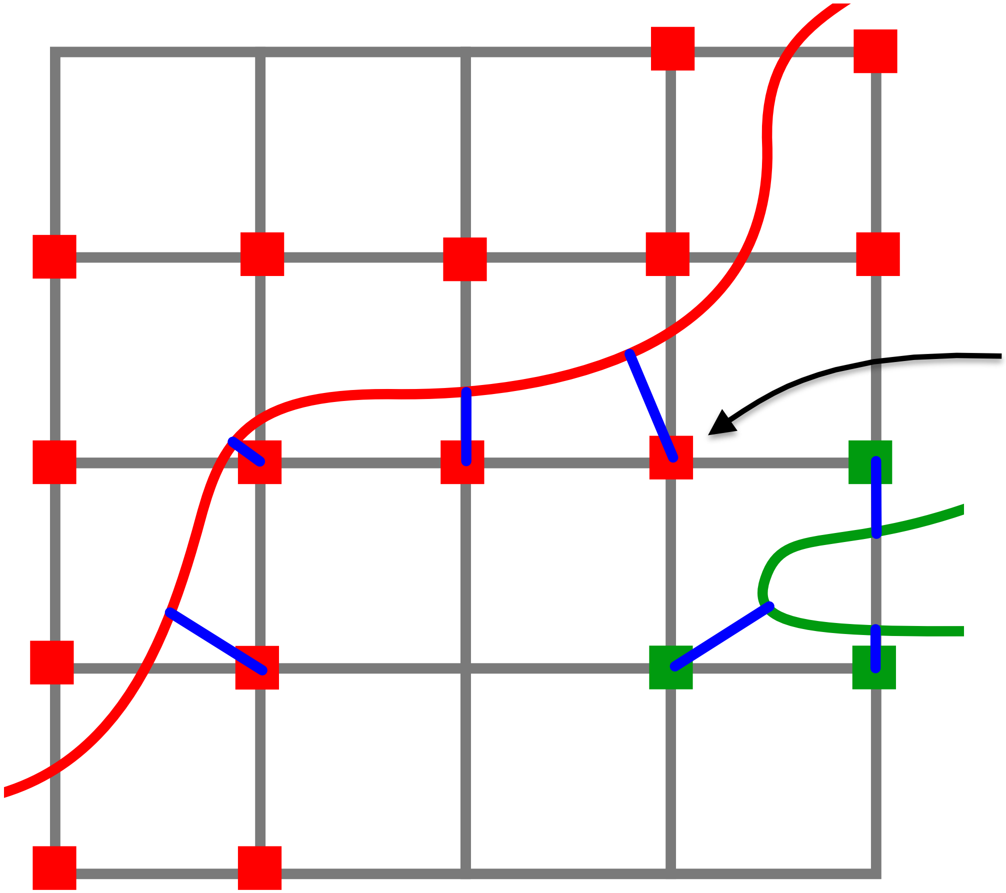
Wavefront distance transform



Example: start with two objects

Wavefront distance transform

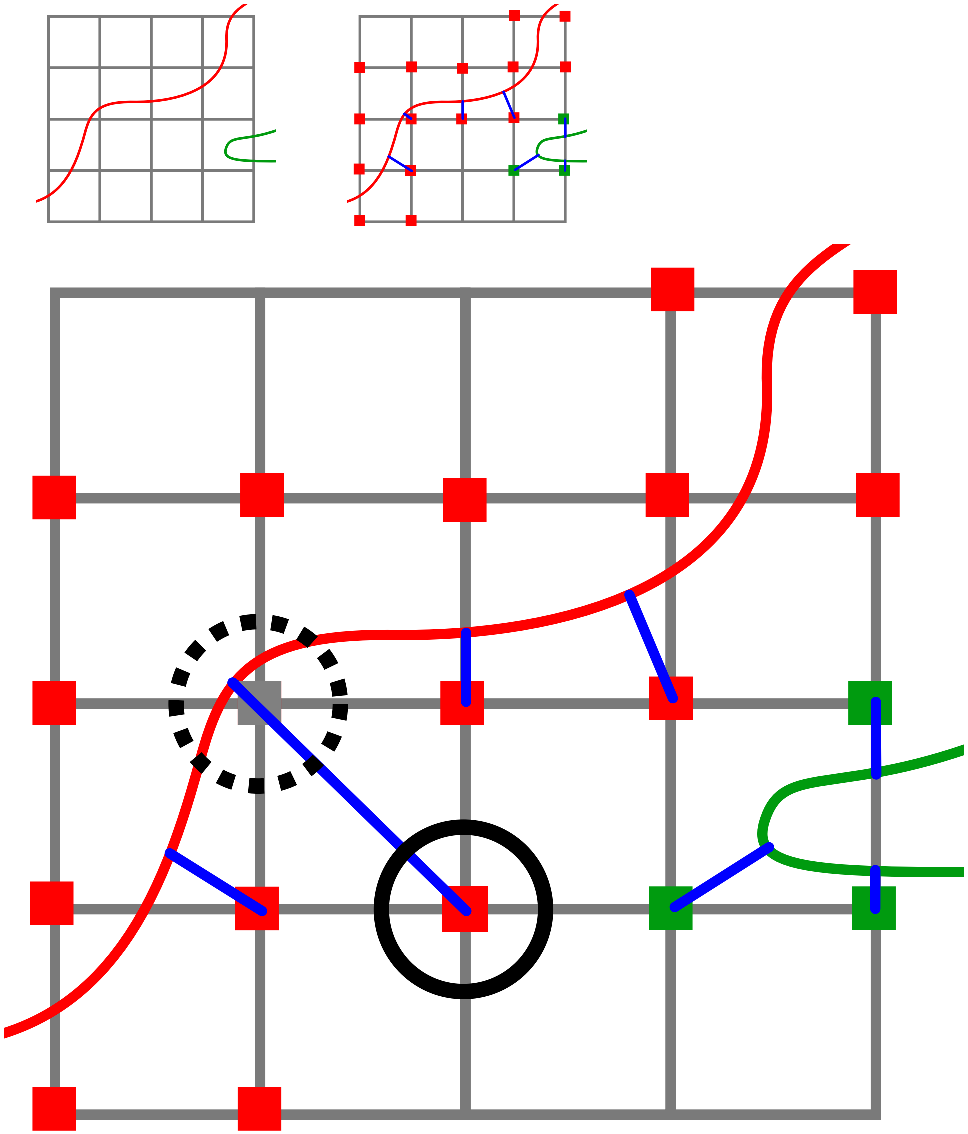




Assign vertices of octree cells that intersect objects

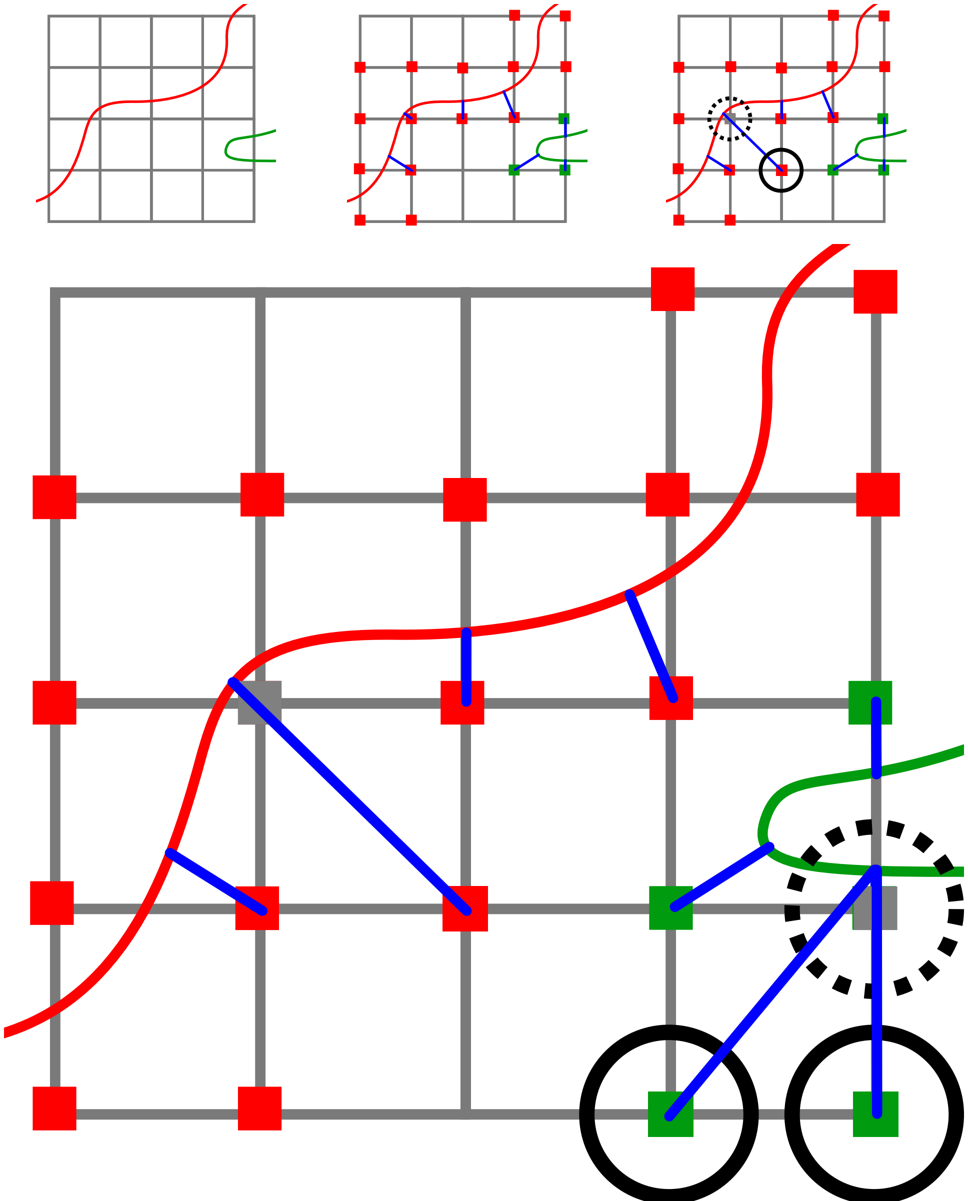
Some vertices belong to two cells

Wavefront distance transform



Propagate closest points to neighbor vertices

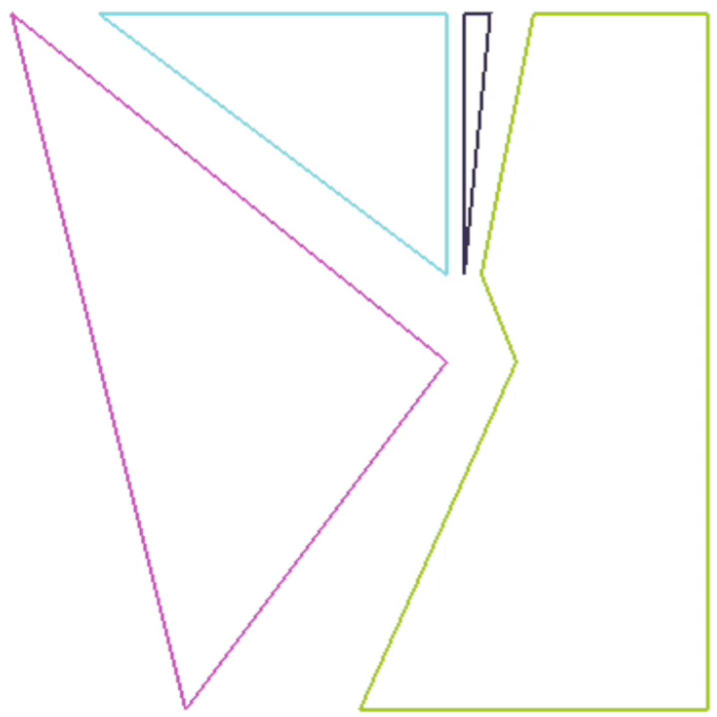
Wavefront distance transform

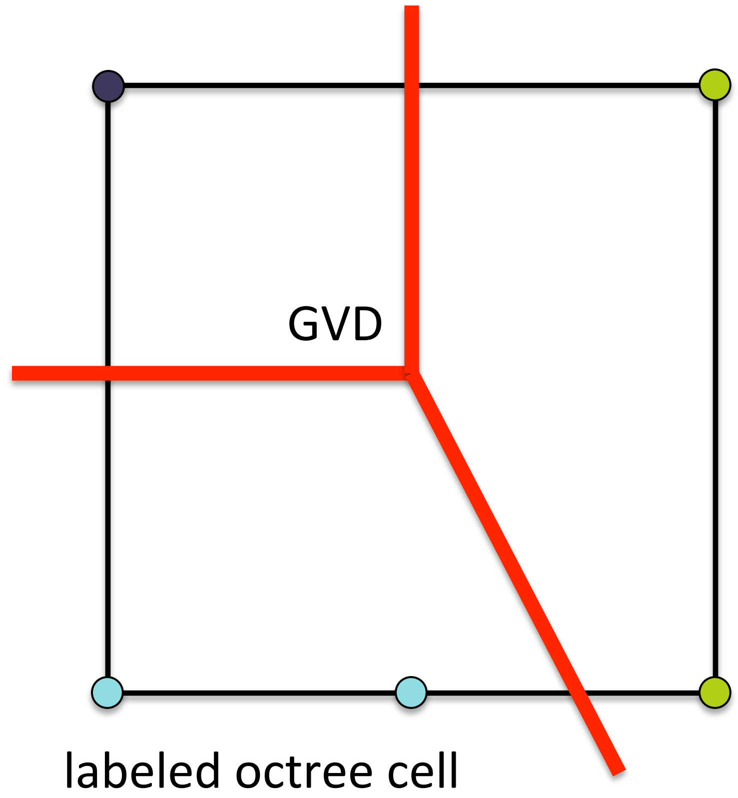


Propagate closest points to neighbor vertices

Wavefront distance transform

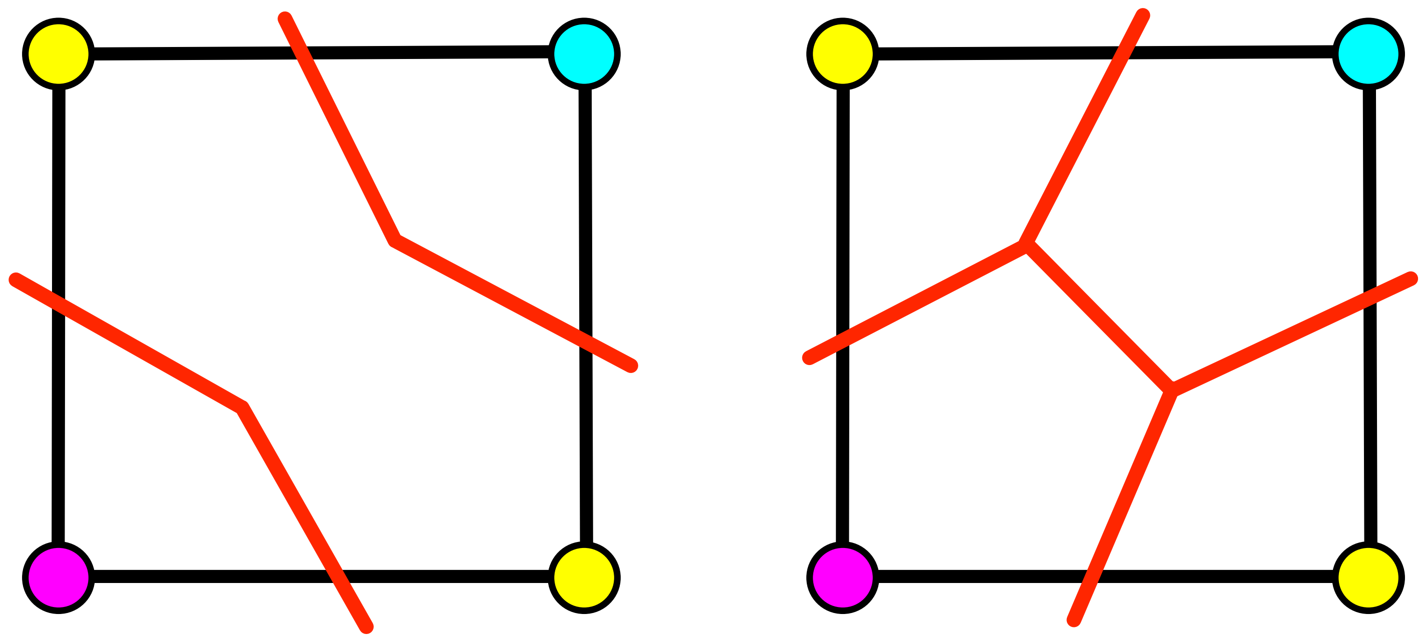
Wavefront propagates until all vertices have been assigned a label



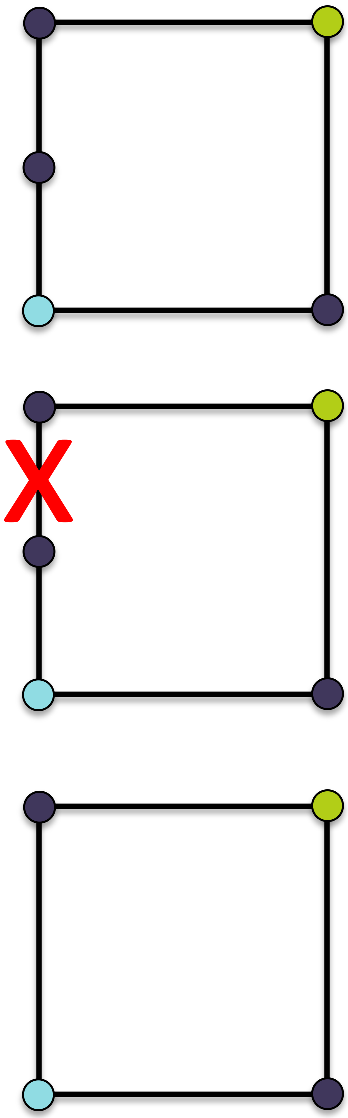


Ambiguity

A cell is ambiguous if there is more than one topology the GVD can take on the interior of the cell



Definition of ambiguity

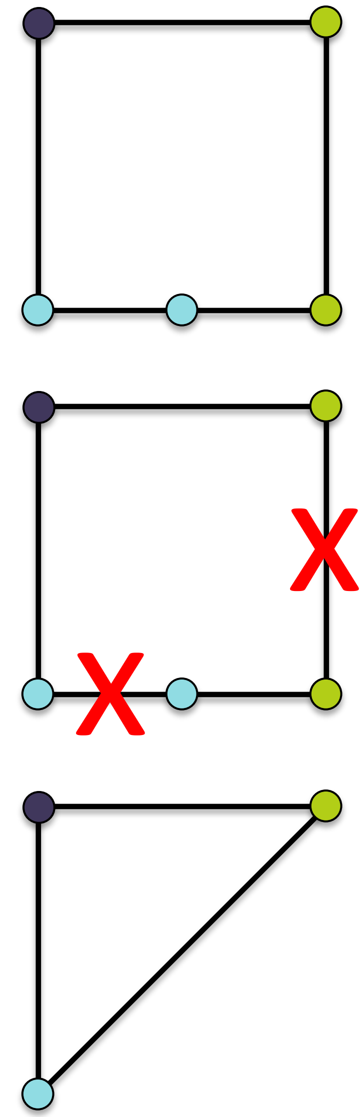


Ambiguous

Consider a D-dimensional labeled octree cell

Collapse edges with same-labeled vertices

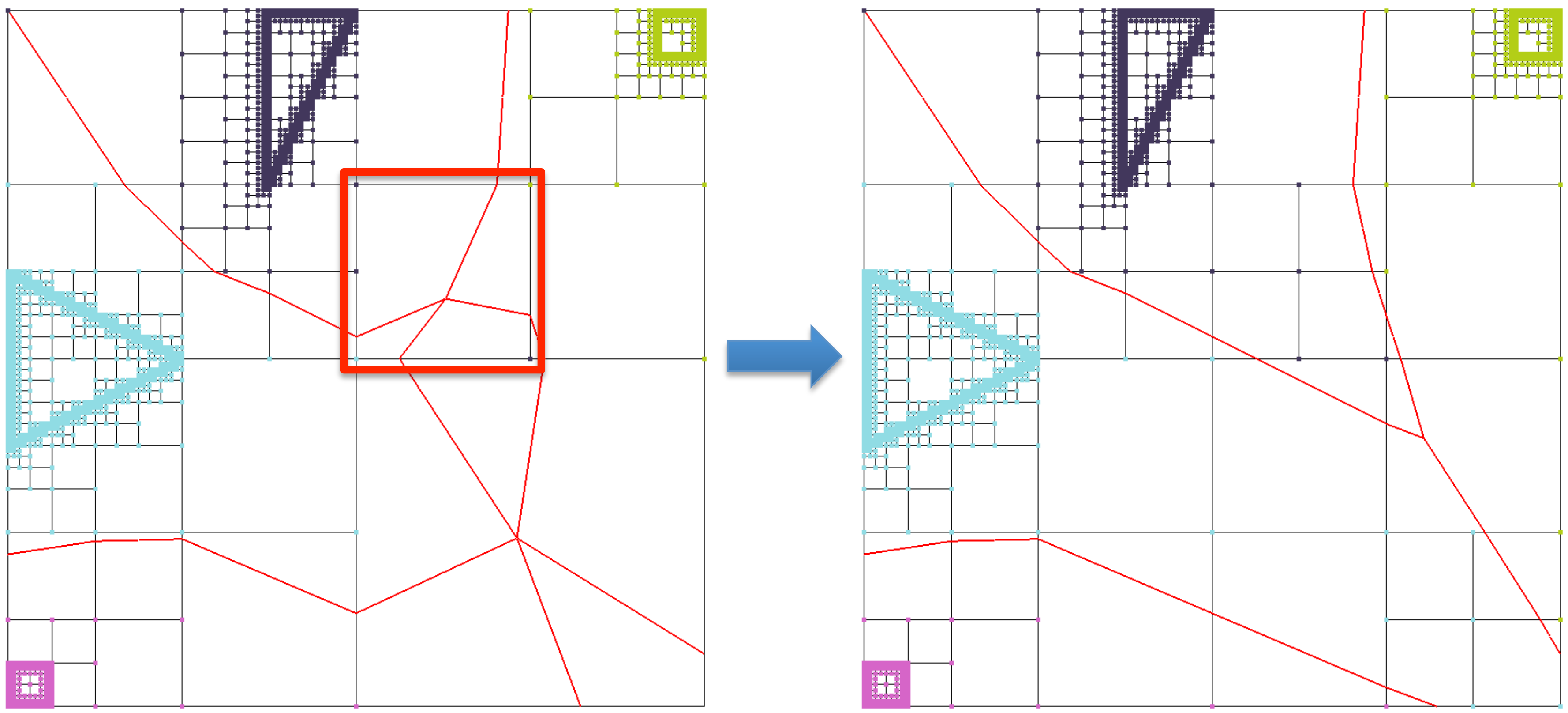
Cell is ambiguous if “reduced” cell is not a (≤D)-dimensional simplex



Not ambiguous

Ambiguity resolution

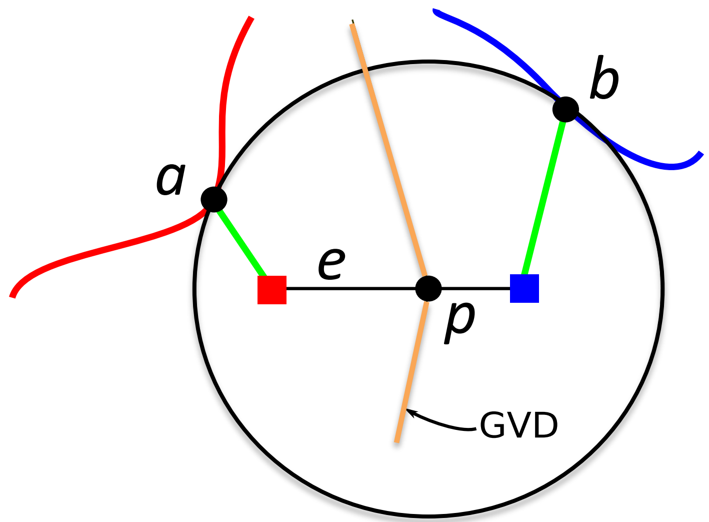
Resolve ambiguities through subdivision



GVD edge intersections

Where on an edge does the GVD reside?

We seek point *p = (x,y,z)* on edge *e.*

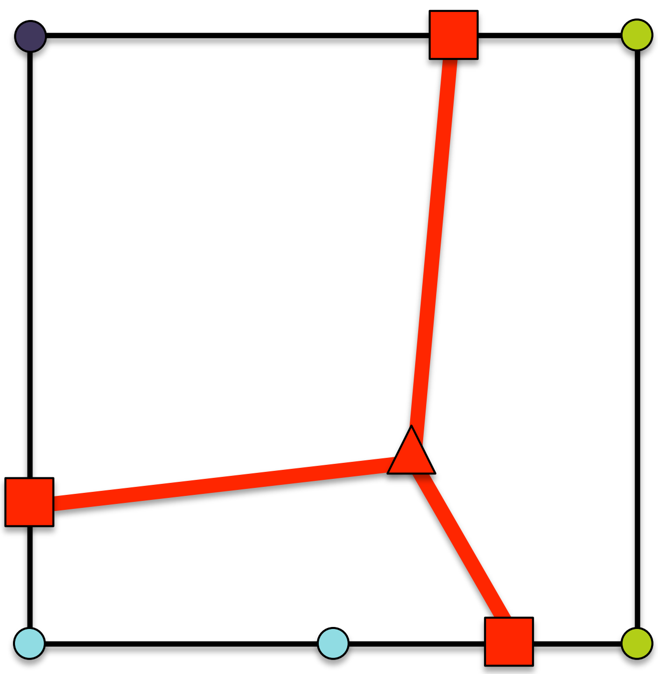


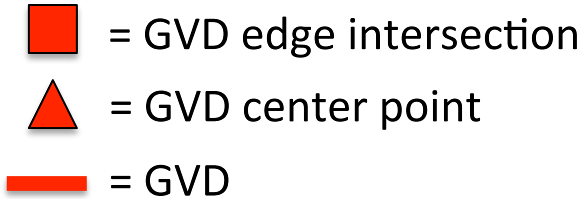
$$x = \frac{2y(a_y - b_y) + 2z(a_z - b_z) + b^T b - a^T a}{2(b_x - a_x)}$$

2D GVD construction

Given a 2D cell with labeled vertices:

1. Compute GVD-edge intersections
2. Compute GVD center point
   – Center point = center of mass of edge intersections
3. Connect edge intersections with center point

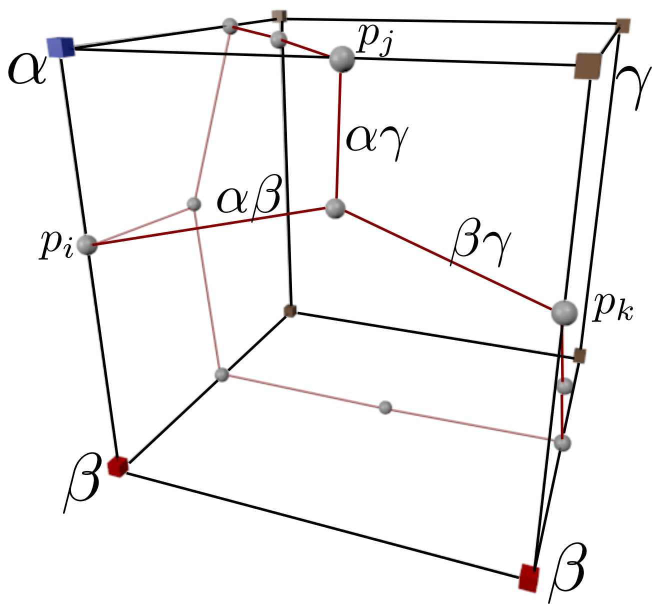


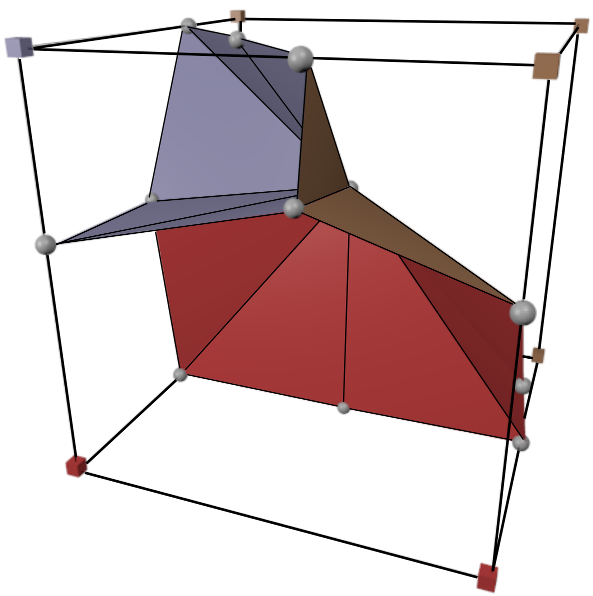


3D GVD construction

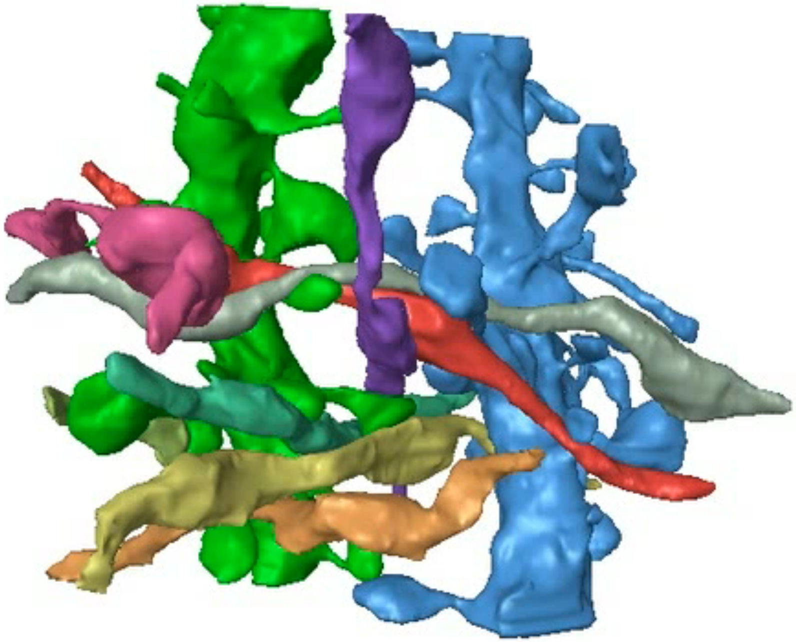
Given a 3D cell with labeled vertices:

1. Compute 2D GVD for each face
2. Triangulate 2D GVDs with cell center

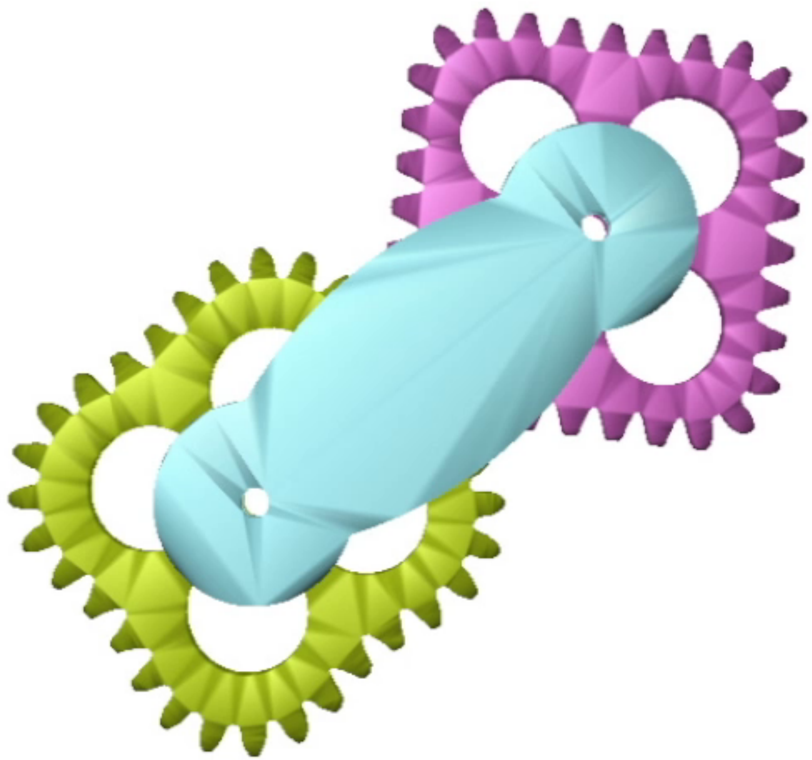




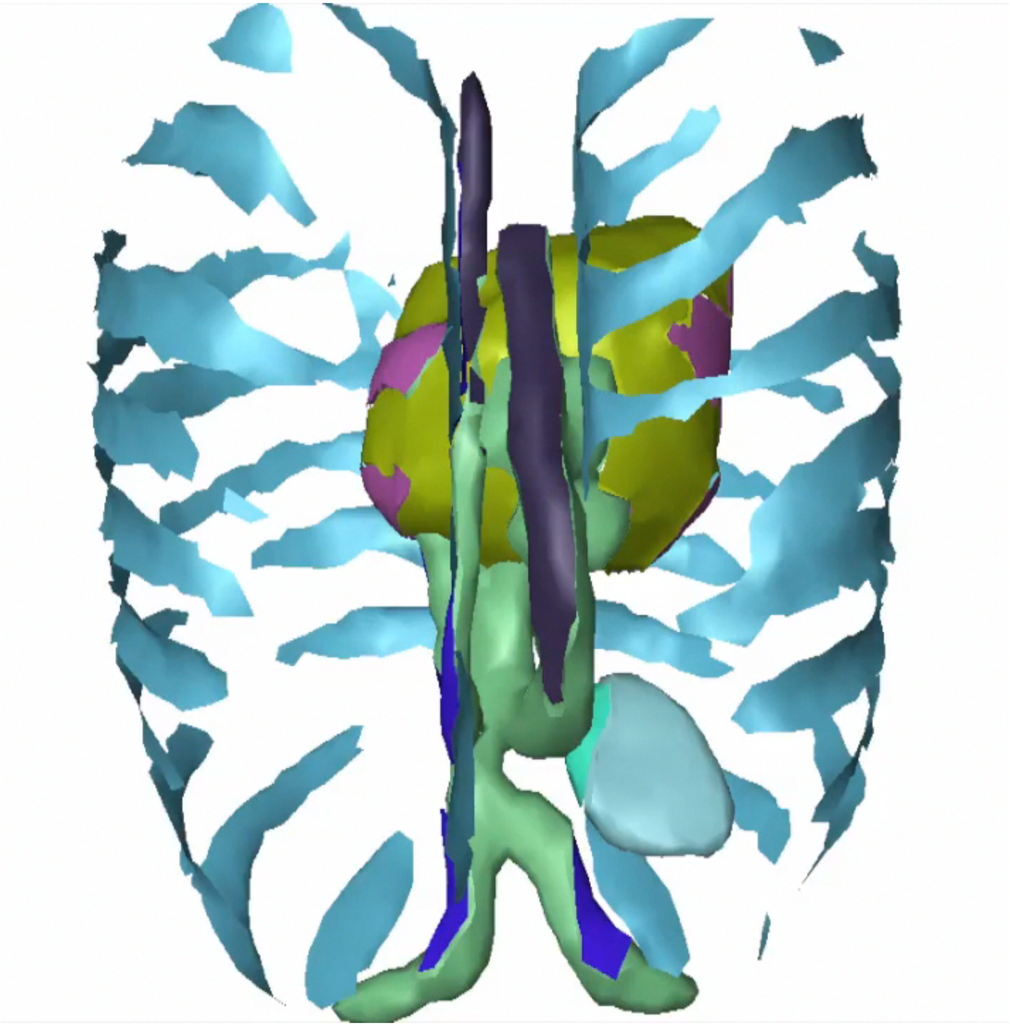
Results



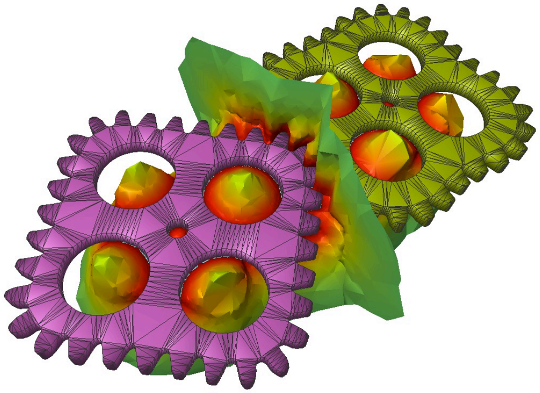
Results



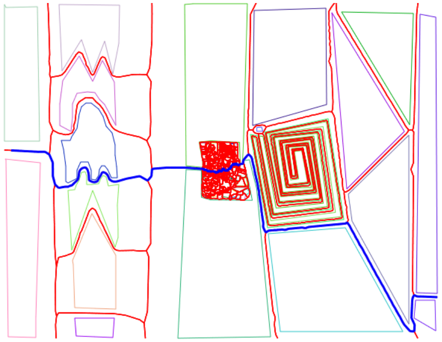
Results



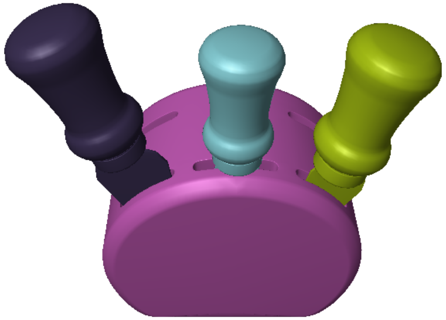
Performance



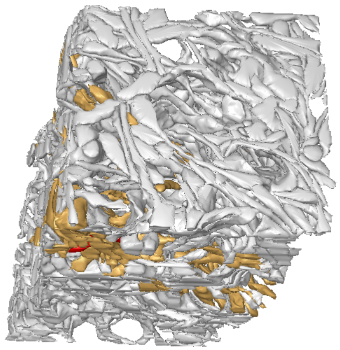
0.9 sec | 3 Mb



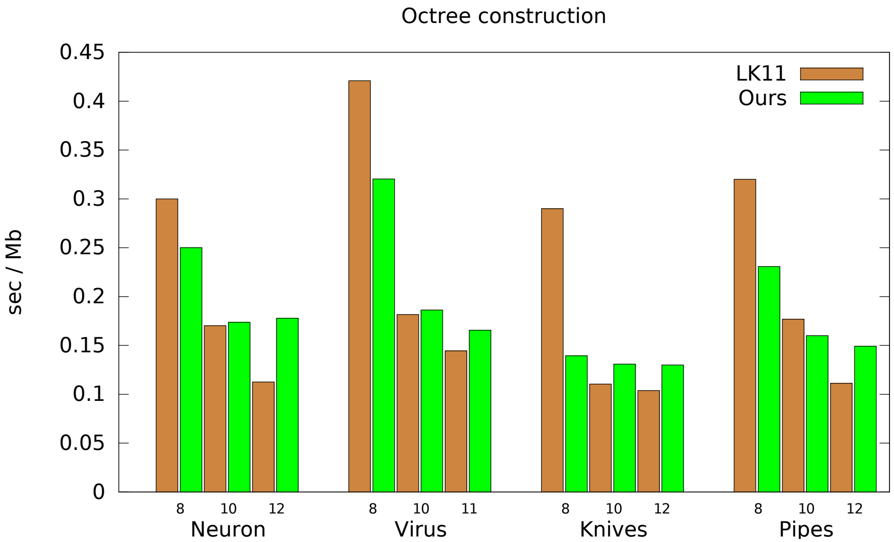
2.0 sec | 8 Mb



3.9 sec | 9 Mb

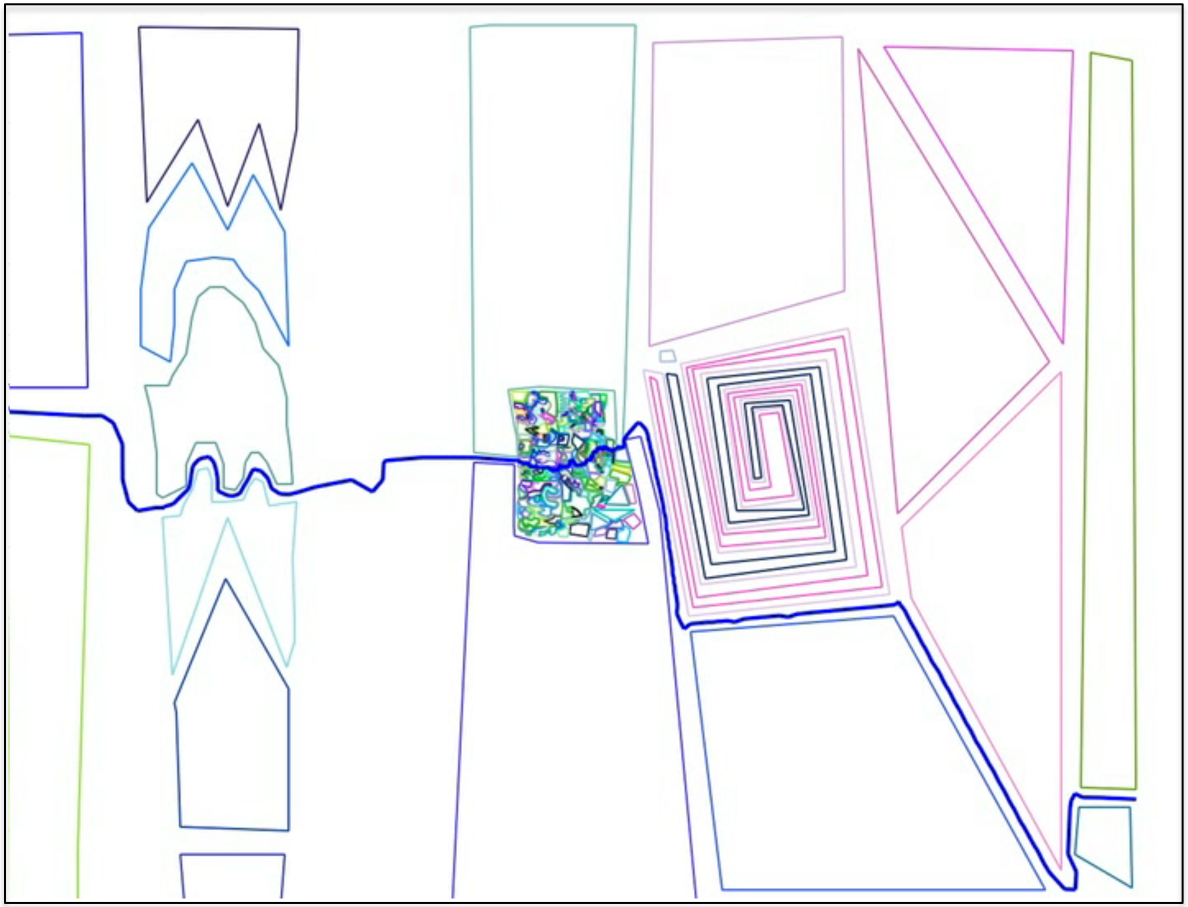


195 sec | 151 Mb

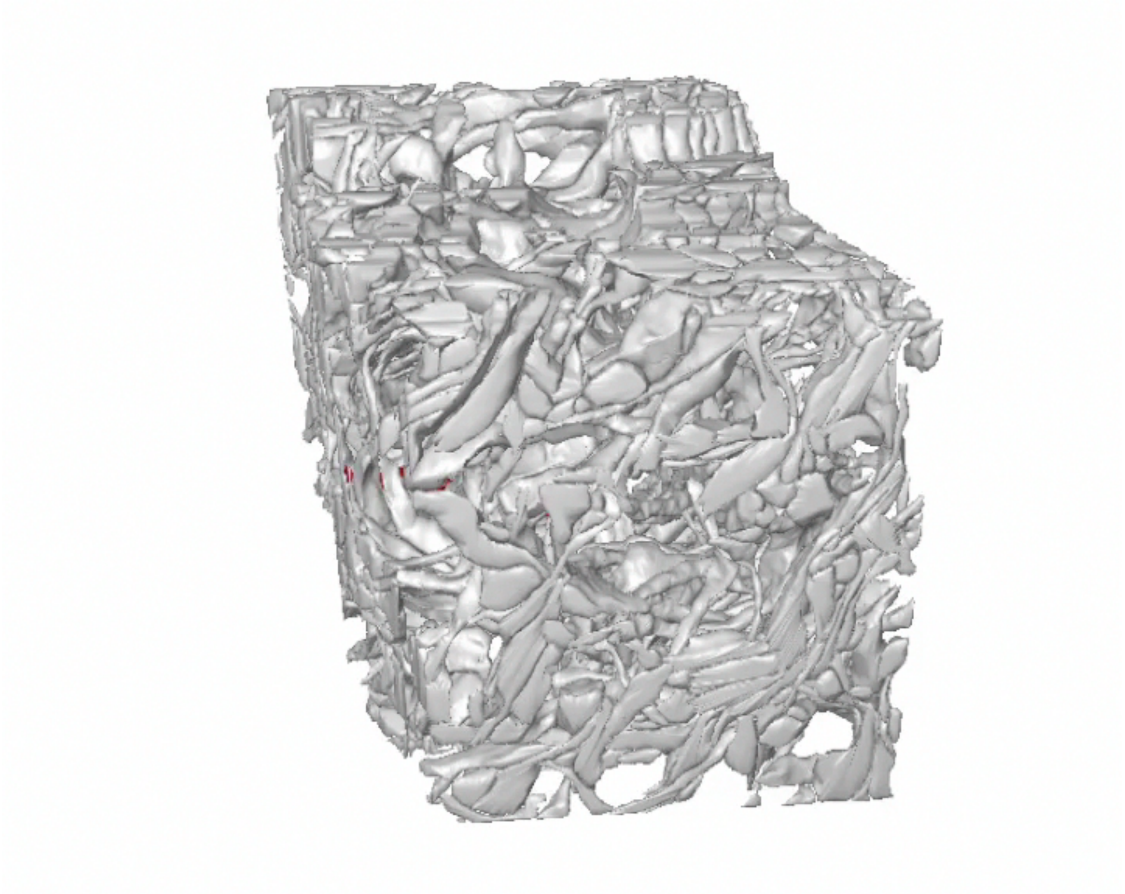


Comparison with Laine and Karras (LK11), which computes an octree that models objects

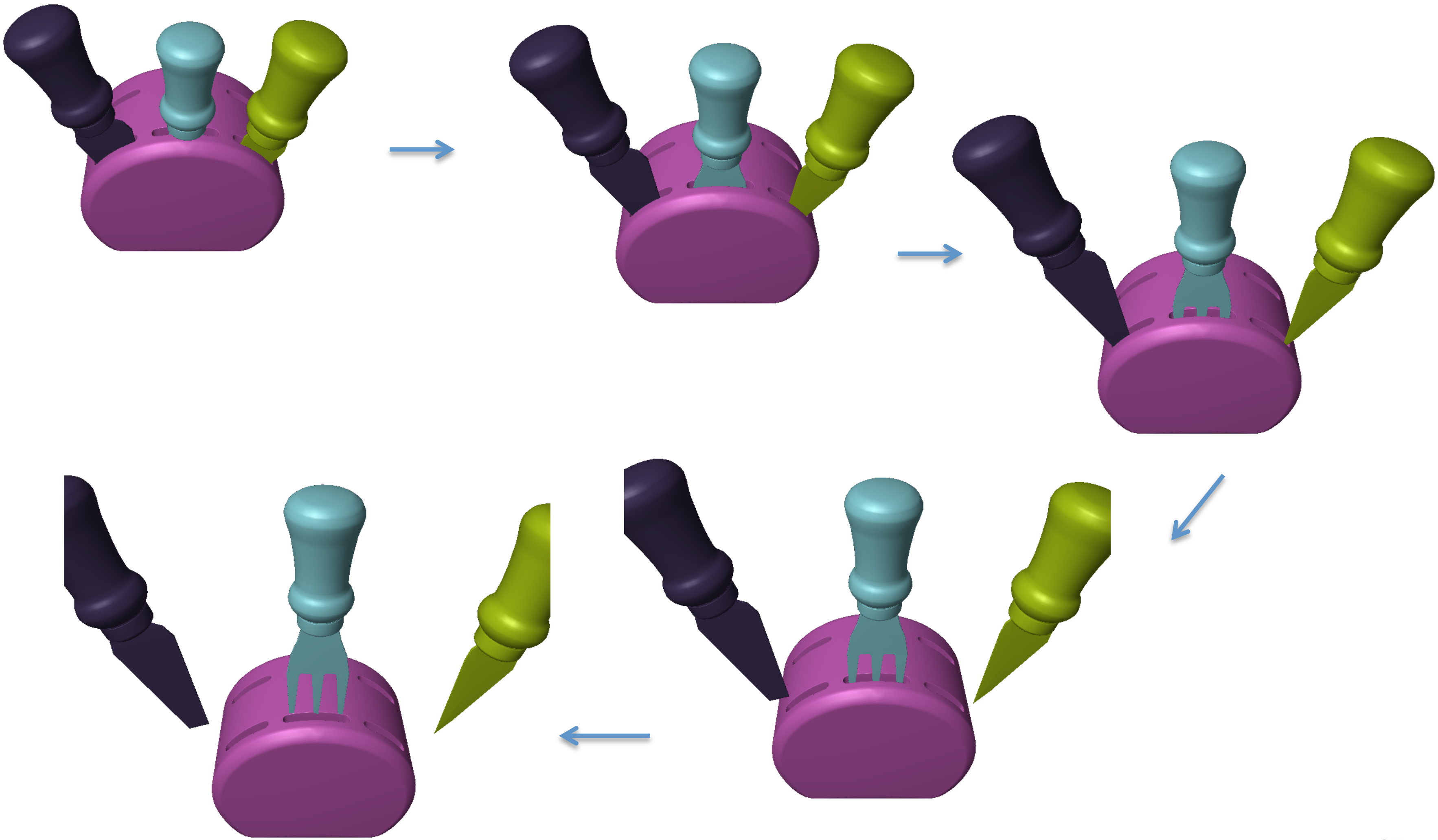
Applications – path finding



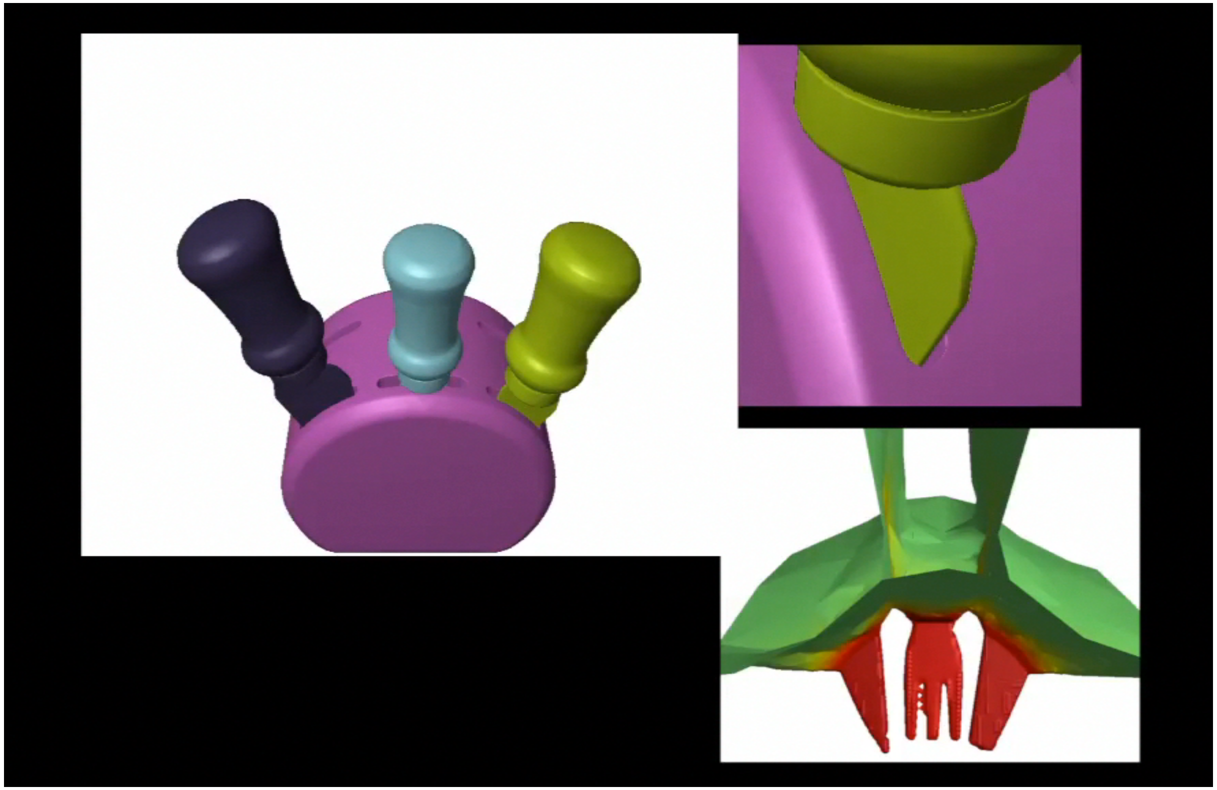
Applications – proximity query



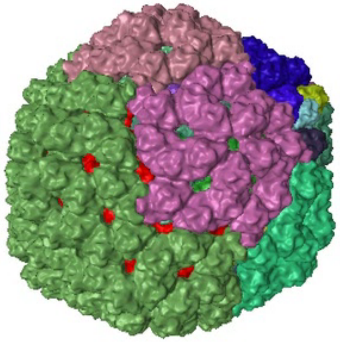
Applications – intersection-free motion



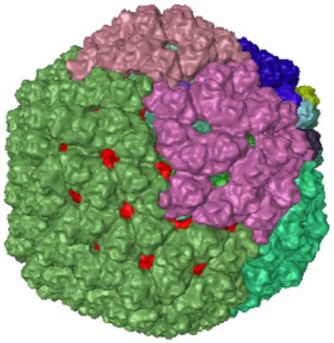
Applications – intersection-free motion



Applications – exploded diagrams



Centroid-based



GVD-based

Conclusion

Before:

```
fun CanComputeGVD(dataset)
  if (grid fits in memory)
    return TRUE
  if (dataset is well-behaved)
    return PROBABLY
  return FALSE
```

Now:

```
fun CanComputeGVD(dataset)
  return TRUE
```

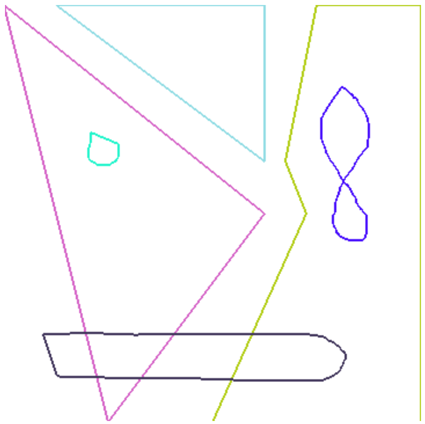
Summary

- GVD can now be computed on arbitrary datasets
- Applications involving difficult datasets are unlocked
- Further work needs to be done for
  - Improved error bound on distance transform
  - Parallelization and other optimizations

Thank you

Code and datasets available at cedmav.org

Relationship to medial axis



Sites

Distance field

GVD

- Medial axis is the locus of critical points of the distance field
- GVD is a subset of the medial axis