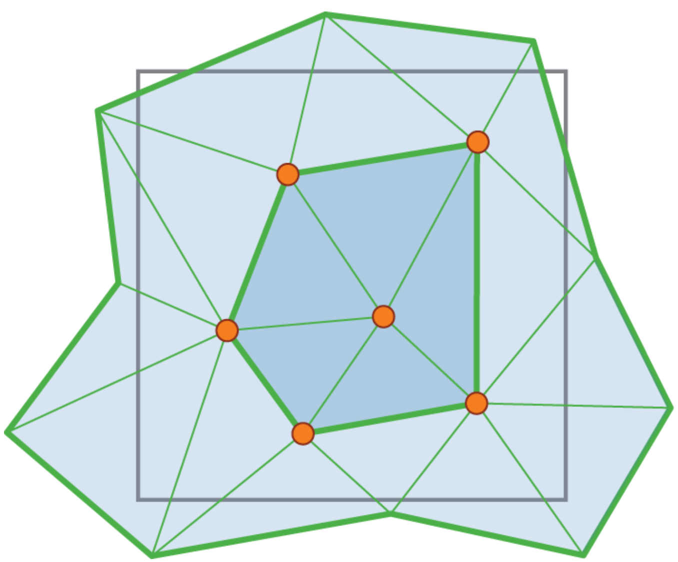
The PR-star Octree:

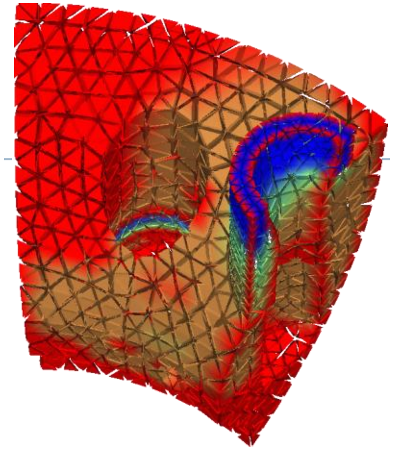
A *spatio-topological* data structure for tetrahedral meshes

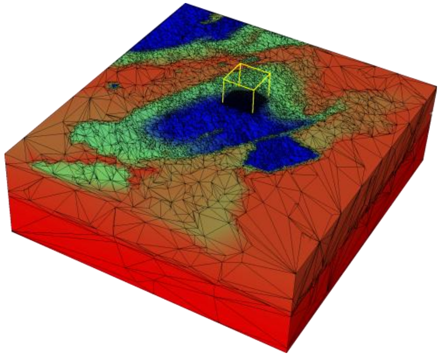


Kenneth Weiss
University of Maryland, College Park
Riccardo Fellegara
University of Genova, IT
Leila De Floriani
University of Genova, IT
University of Maryland, College Park
Marcelo Velloso
University of Maryland, College Park

19th ACM SIGSPATIAL GIS | Chicago, IL | November 1-4, 2011

Motivation





- Tetrahedral meshes
  - Increasingly important for analysis and visualization of scientific datasets
  - Captured/simulated at increasingly fine resolution
- Mesh connectivity
  - Important for many tasks that process the mesh
    - Navigation, visibility, morphology, discrete curvature estimates ray tracing/path following, simplification and repair, etc…
  - Expensive to encode
  - Representations typically are catered to needs of application
- Processing rates (CPU/GPU) increasing faster than memory
  - Favor reductions in memory over those in computing

PR-star Octree

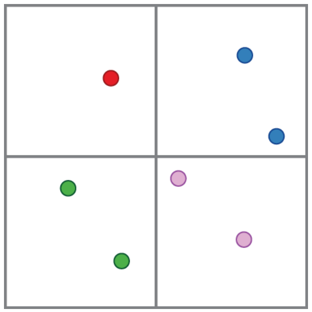
Contributions

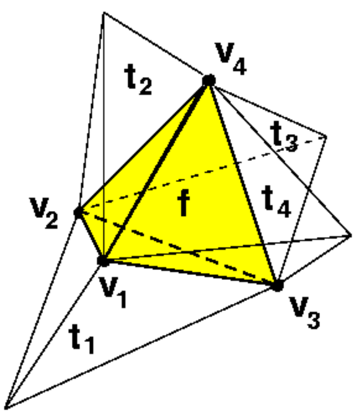
- "Topology through space"
  - Topological connectivity queries
    through spatial index on embedding space
- Encode just enough information to enable efficient reconstruction of all topological relations
  - Allows optimal application-dependent
    local data structures to be generated at runtime
  - Construction costs amortized over multiple coherent queries
- Streaming algorithms over dataset
  - Boundary determination, local curvature estimates, simplification
  - Many more…
- Benefits of this representation increase with dataset size

Related Work

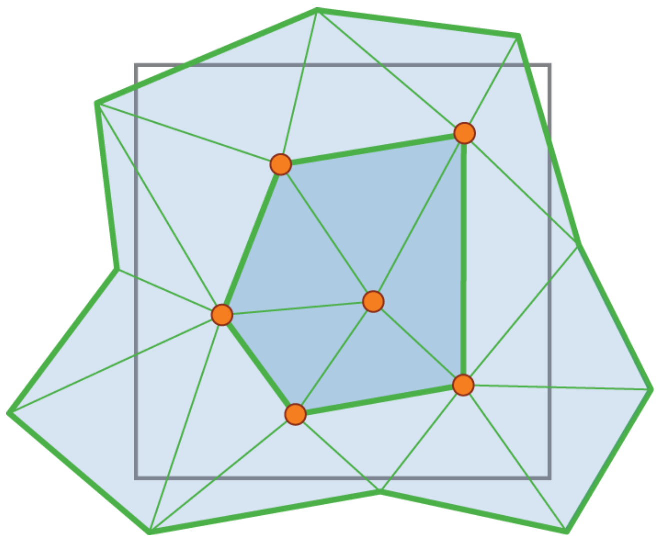
- Spatial data structures
  - Focus is on efficient spatial queries
    - e.g. point location, (*k*)- nearest neighbor query
  - Points:
    - PR- quadtrees, octrees and kd-trees [Samet:2006]
  - Polygons, edges and graphs; Triangles:
    - PM-family of quadtrees – PM1-, PM2-, PM3-, PMR-
  - Tetrahedral meshes [De Floriani *et al.*:2010]
- Topological data structures
  - Focus is on efficient connectivity queries
  - Incidence-based – IG [Edelsbrunner:1987]
  - Adjacency-based – IA [Paoluzzi:1993; Nielson:1997]
- Spatial index on triangle mesh for out-of-core processing [Cignoni:2003] or for expensive processing [Dey et al.: 2010]

Talk overview

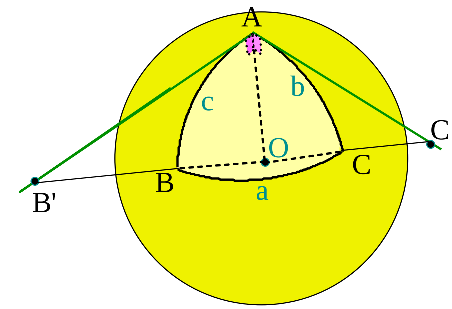


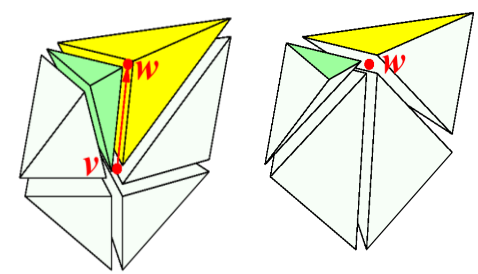


Background



PR-star Octree

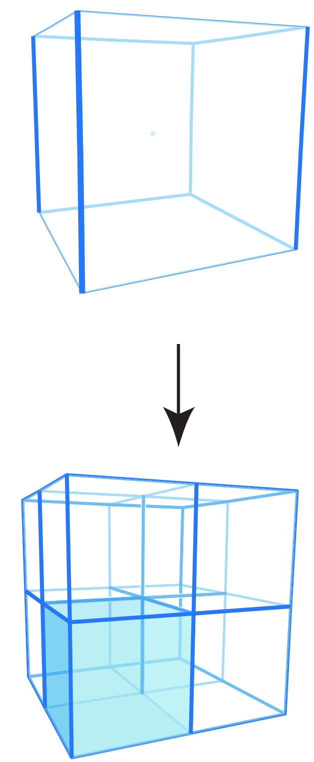


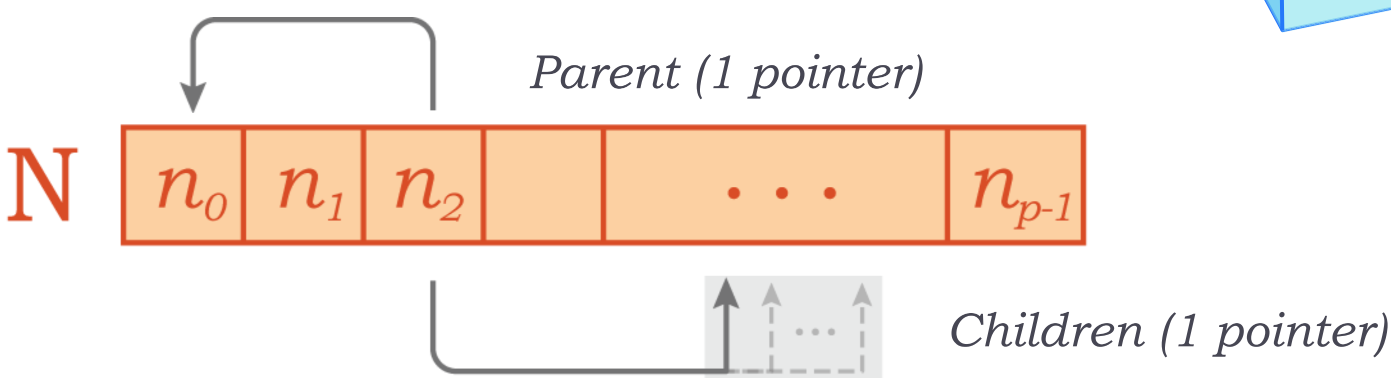


Applications

Region Octrees

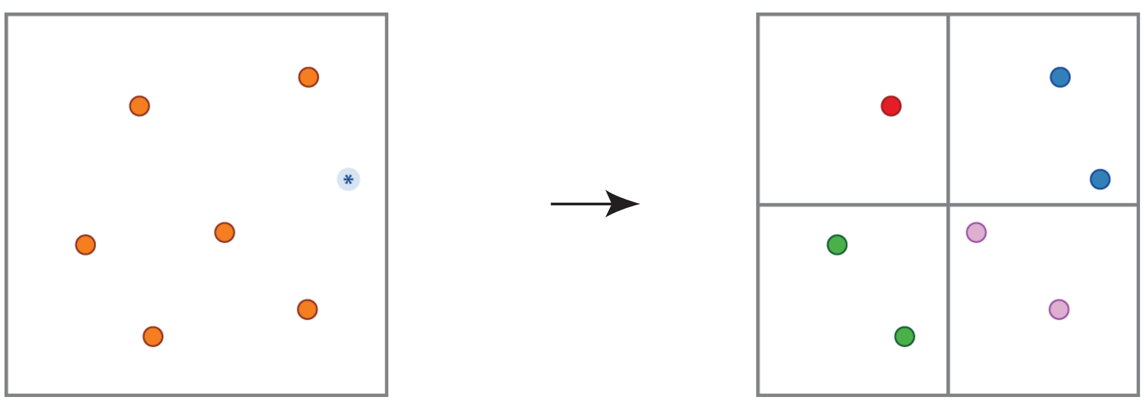
- Hierarchical domain decomposition
- Regular refinement
  - Each cubic parent node is replaced by eight children nodes covering its domain
- Root node
  - Cubic node covering entire domain
- Leaf node
  - Cubic node without children
  - Non-leaf nodes are called *internal nodes*





PR Octree:
Point Region Octree

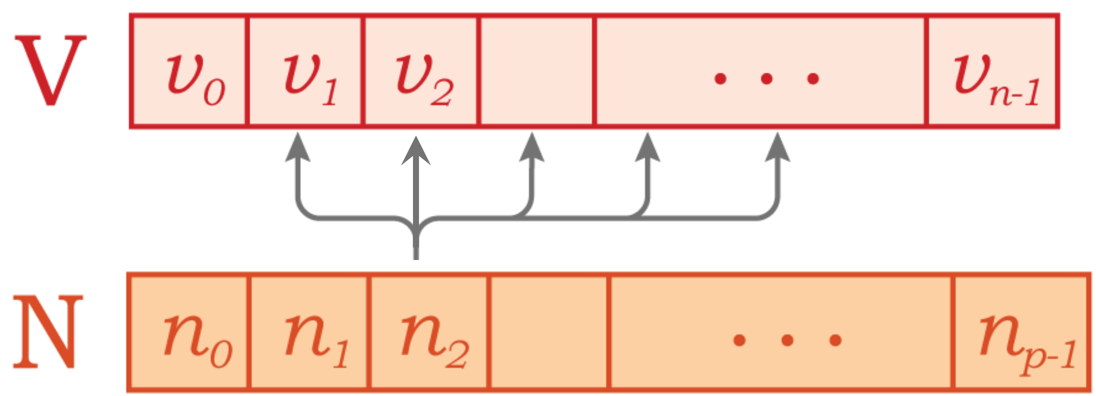
- Region octree used as spatial index on a set of points
  - Points are uniquely indexed by a single leaf node
- Bucket threshold k_v
  - Used to decide when to split a node
    - Decomposition entirely dependent on k_v
- A node is considered *full* when it indexes k_v points
  - Redistribute points to children upon insertion into *full* leaf node



$k_v = 6$

PR Octree:
Representation

- An array of points in R^3 – ***V***
- A set (array) of octree nodes – ***N***
  - Each leaf node ***n*** in ***N*** indexes the set of at most k_v points from ***V*** that lie within its domain

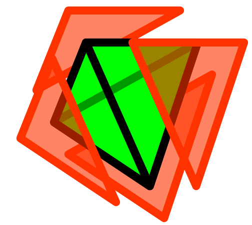


Topological Connectivity Relations

- Fundamental *connectivity* primitives for mesh elements

Boundary relations – $R_{p,q}$ $(p<q)$

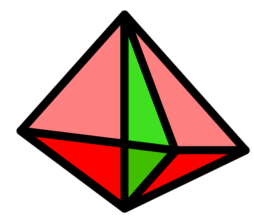
- Set of q-simplices that are a face of a given p-simplex
- e.g. $R_{3,0}$ is the *Tetrahedron-Vertex* relation



$R_{3,2}$

Co-boundary relations – $R_{q,p}$ $(p<q)$

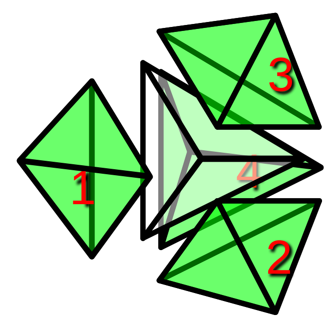
- Set of simplices that have a given simplex as a face
- e.g. $R_{0,3}$ is the *Vertex-Tetrahedron* relation
  - The tetrahedra in the *star* of v



$R_{2,3}$

Adjacency relations – $R_{p,p}$

- Set of p-simplices that adjacent to a given simplex along a p-1 face $(p>0)$ or an edge $(p=0)$
- e.g. $R_{3,3}$ is the *Tetrahedron-Tetrahedron* relation



$R_{3,3}$

Topological Data Structures

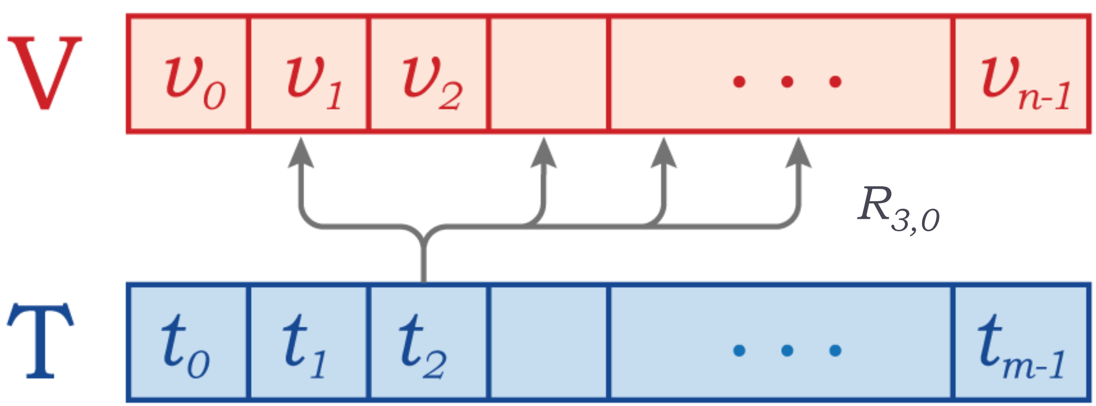
- Explicitly encode a subset of the topological relations
- Implicitly encode a (larger) subset of the relations
  - Reconstruct relevant neighborhoods from encoded relations at runtime

- Application-dependent data formulations
  - *Incidence-based* data structures
    - e.g. Incidence Graph [Edelsbrunner:1987]
  - *Adjacency-based* data structures
    - e.g. Indexed data structure with Adjacency (IA) [Paoluzzi *et al*:1993]

- Adjacency-based data structures more compact when we are mainly interested in *top cells* [DeFloriani and Hui : 2006]

Indexed tetrahedral mesh

- Array of vertices **V**
  - Each vertex v_i encodes a position (x,y,z) and possibly other attributes
- Array of tetrahedra **T**
  - Each tetrahedron t_j encodes the index in **V** of its vertices and possibly other attributes

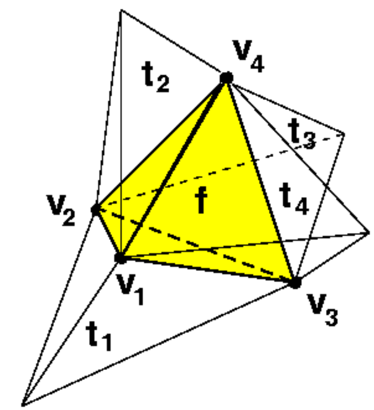


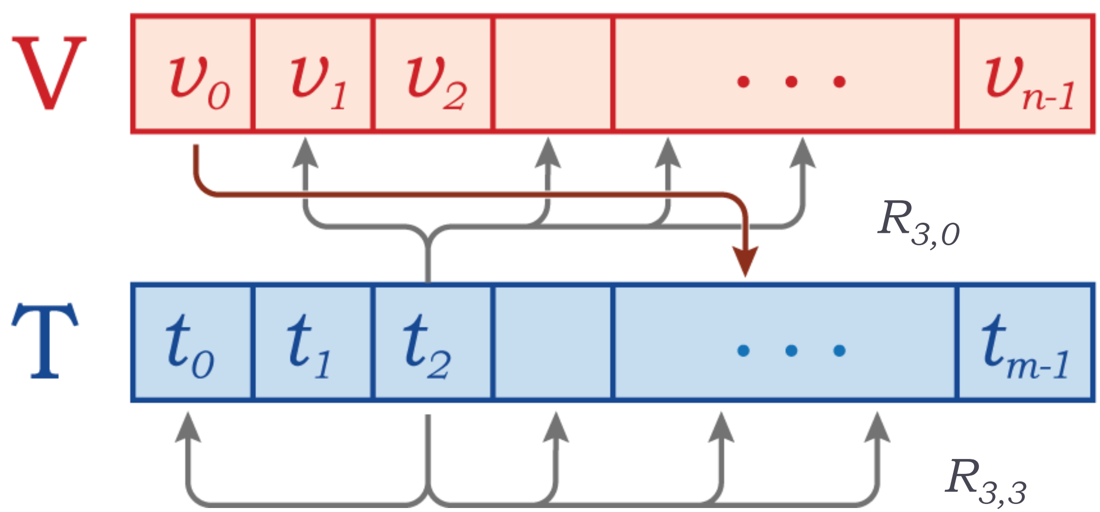
$$v_i = \{x, y, z\}$$

$$t_i = \{i_{v0}, i_{v1}, i_{v2}, i_{v3}\}$$

IA data structure:
Indexed tetrahedral mesh with Adjacencies

- Array of vertices **V**
  - Encodes position of each vertex
  - Encodes a single incident tetrahedron in **T**
- Array of tetrahedra **T**
  - Encodes indices of four vertices in **V**
  - Encodes indices of four adjacent tetrahedra in **T**



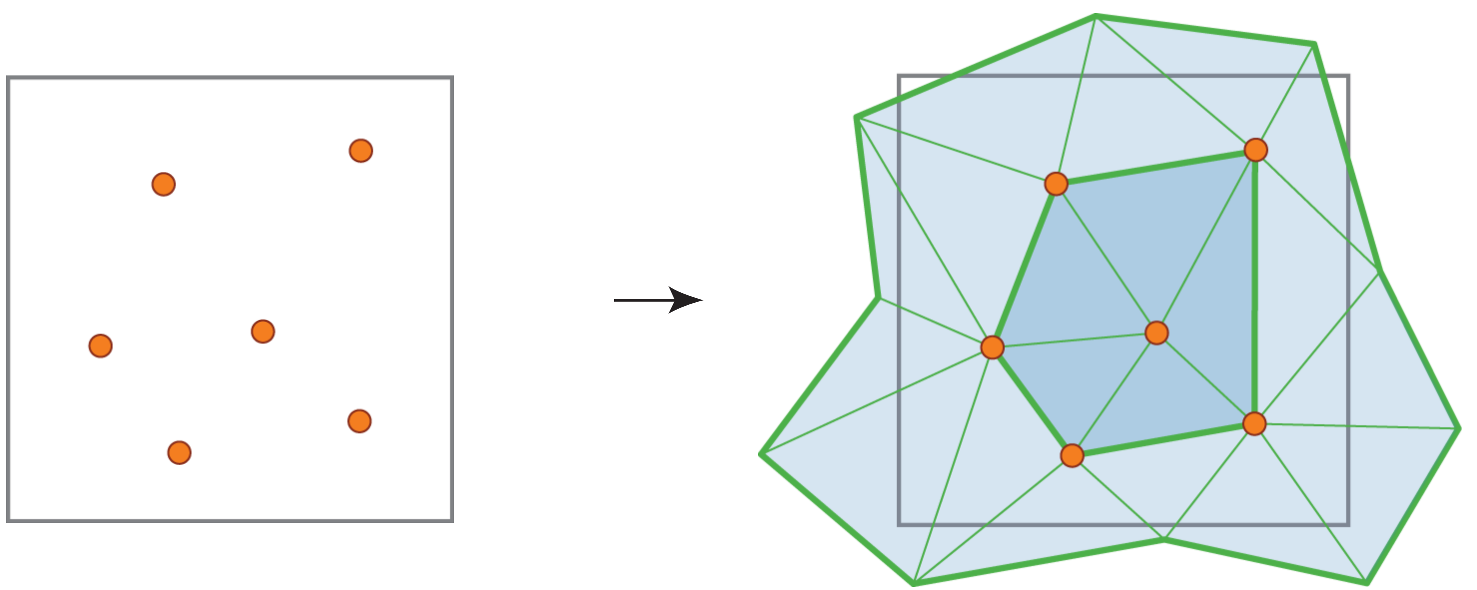


$$v_i = \{x,\ y,\ z,\ i_{t0}\}$$

$$t_i = \left\{ \begin{matrix} i_{v0}, & i_{v1}, & i_{v2}, & i_{v3} \\ i_{t0}, & i_{t1}, & i_{t2}, & i_{t3} \end{matrix} \right\}$$

PR-star Octree

- "Topology through space"
  - A spatial data structure for querying topological connectivity
- Augment PR octree with the set of tetrahedra from the mesh that are incident in its vertices
  - i.e. the tetrahedra in the star of its vertices



Generation of PR-star

Three steps

- Input is soup of tetrahedra defining a tetrahedral mesh Σ

Step 1: Vertices

- Create a PR octree ***N*** on vertices ***V*** of mesh
- Based on user selected bucket threshold k_v

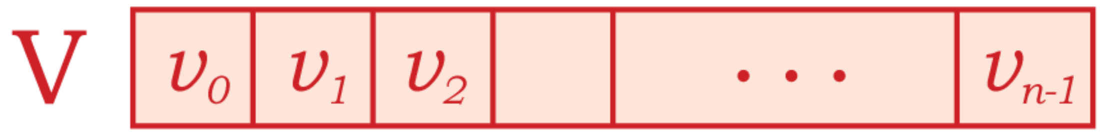
Step 2: Tetrahedra

- Add tetrahedra ***T*** to appropriate leaf nodes of ***N***

Step 3: Spatial sort

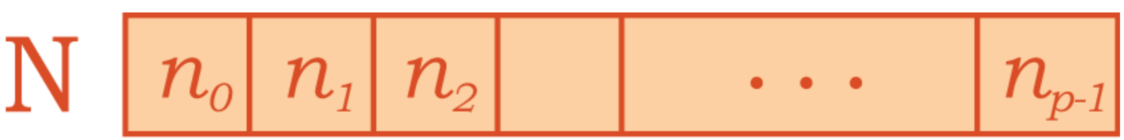
- Reorganize ***V*** and ***T*** based on spatial sorting induced by ***N***
  - Each node in ***N*** indexes a contiguous range of vertices in ***V***
  - Can be encoded via two indices v_{start} and v_{end}
- For ***T*** we store a pointer to a list of tetrahedra indices

PR-star Octree
Representation



Encodes: geometry of the mesh *[3 pointers]*

Encodes: four indices in ***V*** of its vertices *[4 pointers]*



Encodes: hierarchical octree information *[3 pointers]*

range of vertices (v_{start}, v_{end}) *[2 pointers]*

pointer to list of incident tetrahedra *[2 pointers]*

PR-star Octree

Representation

V | v_0 | v_1 | v_2 | | . . . | v_{n-1} |

Encodes: geom... [3 pointers]

T | t_0 | t_1 | t_2 | ...

Encodes: fou... ...ointers]

N | n_0 | n_1 | n_2 | ...

Encodes: hierarch... ...pointers]

range of ve... (v_{start}, v_{end}) [2 pointers]

pointer to list of incident tetrahedra [2 pointers]

Lists of tetrahedra:

Each tetrahedron appears in

- At least one octree node
- At most four octree nodes

χ – Average number of lists in which a tetrahedron appears, where

$$1 \leq \chi \leq 4$$

Evaluation

- Indexed Tetrahedral Mesh Representation
  - Fixed cost of both data structures
  - Total $4|\boldsymbol{T}| + 3|\boldsymbol{V}| \sim 27|\boldsymbol{V}|$

- IA data structure (extended)
  - Topological: $4\,|\boldsymbol{T}| + 3|\boldsymbol{V}| \sim 25|\boldsymbol{V}|$
  - Total: $8\,|\boldsymbol{T}| + 4|\boldsymbol{V}| \sim 52|\boldsymbol{V}|$

- PR-star data structure
  - Topological: $\chi\,|\boldsymbol{T}| + 7|\boldsymbol{N}| \sim 13|\boldsymbol{V}|$
  - Total: $8\,|\boldsymbol{T}| + 4|\boldsymbol{V}| \sim 40|\boldsymbol{V}|$

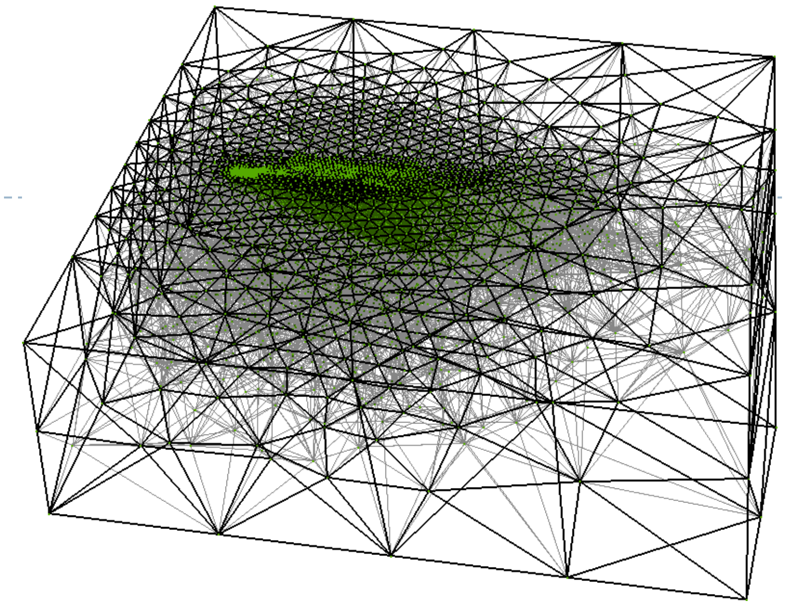
Comparison
~50% topological
~80% total storage

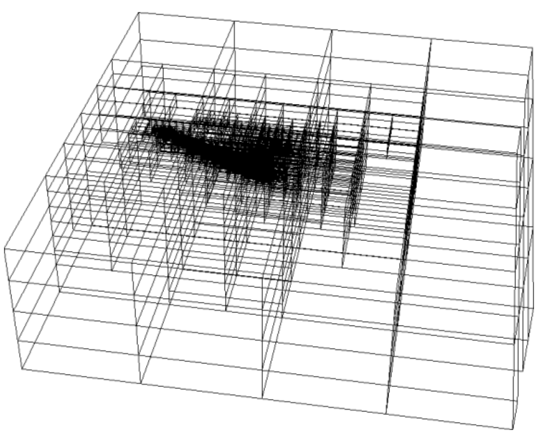
Simplifying assumptions: (see paper for details)

$|\boldsymbol{T}| \sim 6\,|\boldsymbol{V}|$ $|\boldsymbol{N}| \sim |\boldsymbol{V}| / k_v$ $\chi \sim 2$ $k_v \geq 7$

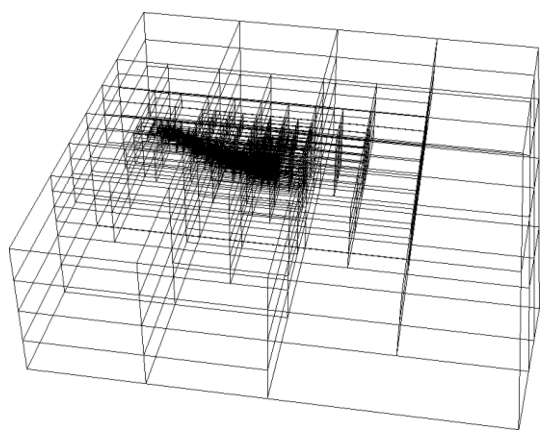
PR-star Octree:
Example

- F117 tetrahedral mesh
  - $|\boldsymbol{V}| = 48.5\ K$
  - $|\boldsymbol{T}| = 240\ K$
  - IA storage: (20.8; 43.6)

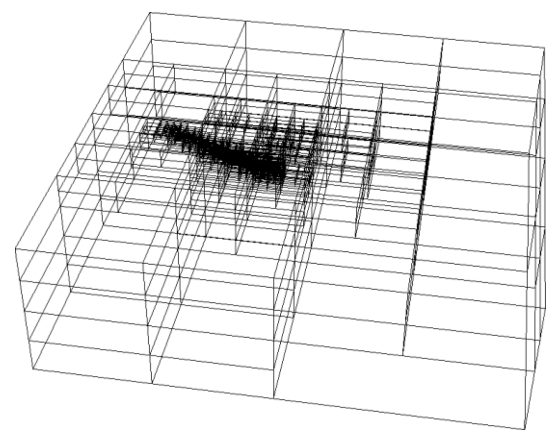




$k_v = 50$
$\chi = 2.6$; $|\boldsymbol{N}| = 4$ K
Storage: (12.8; 35.6)



$k_v = 100$
$\chi = 2.2$; $|\boldsymbol{N}| = 1.9$ K
Storage: (10.9; 33.7)



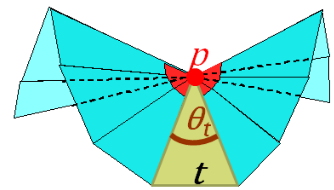
$k_v = 200$
$\chi = 2.0$; $|\boldsymbol{N}| = 1.4$ K
Storage: (10.0 ; 32.8)

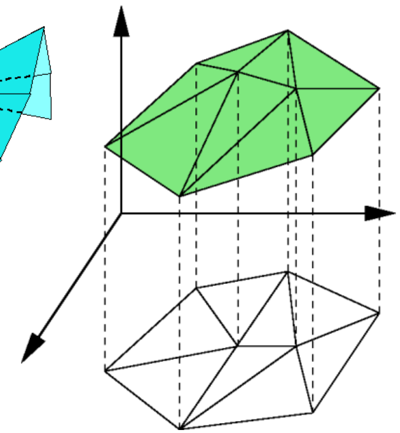
Applications of PR-star
General Strategy

- Streaming algorithm
  - Iterate through octree nodes
- For each leaf octree node
  - Step 1: Build application-dependent local data structure
  - Step 2: Process mesh locally
  - Step 3: Discard local data structure
- Cost of building data structures is amortized over multiple local operations

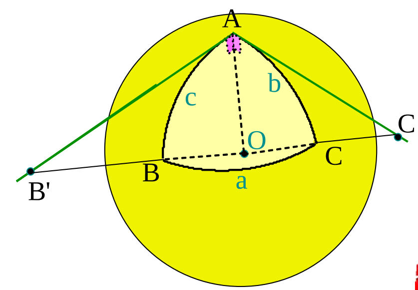
Local discrete curvature estimates

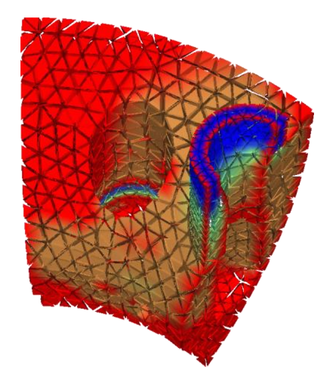
- For terrain
  - Elevations at samples in 2D domain provide embedding as 3D TIN
  - Curvature is concentrated in vertices
  - Depends on geometry of its star
    - e.g. angle deficit between 2D and 3D [Aleksandrov:1957]





- For volume data
  - Scalar values at samples in 3D domain provide embedding as 4D hypersurface
  - Curvature is concentrated in vertices
  - Depends on geometry of its star
    - e.g. angle deficit between 3D and 4D [Mesmoudi et al.:2008]





Results

Timings for generating VT and distortion

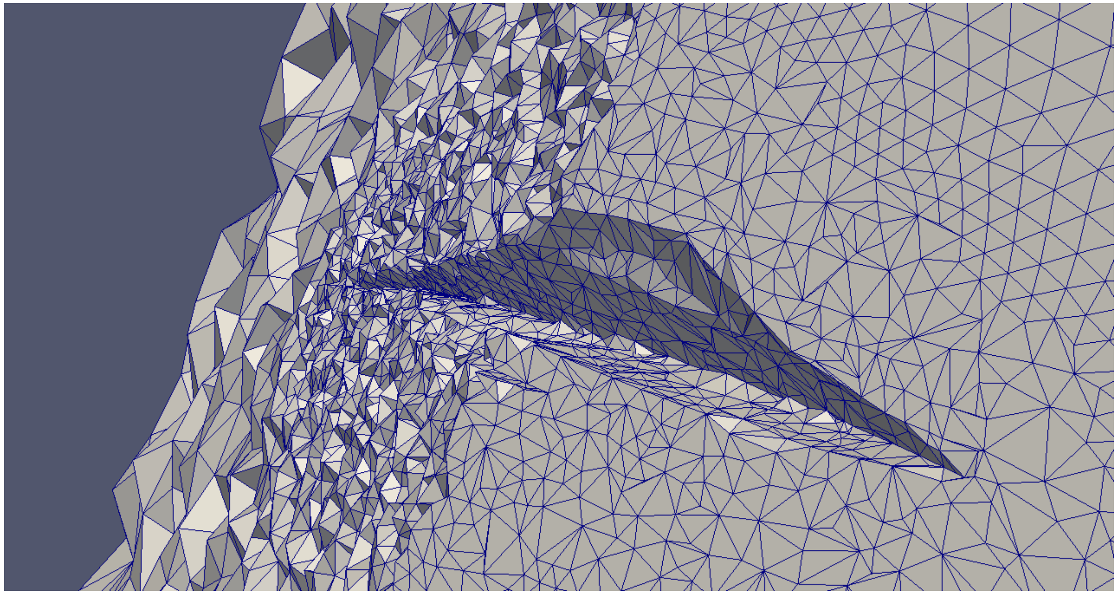
- Compared to IA data structure

- Key observations
  - Building VT is always faster for PR-star
    - Amortized cost over entire mesh
  - For small meshes with small k_v
    - Distortion computation is faster with IA
    - Value of χ plays a dominant role here
  - As mesh size increases, and as k_v increases
    - Distortion is faster with PR-star

- Trend: Effectiveness of PR-star increases with mesh size

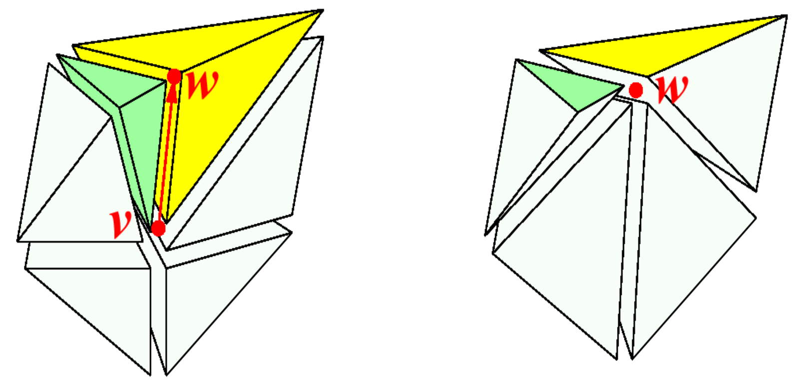
Application
Mesh simplification

- Many mesh generation processes oversample the field
- Simplification algorithms are critical to downstream processing but are resource intensive
  - Local mesh modifications require neighborhoods of vertices
  - Better results are obtained by ordering the simplifications



Local simplification
Half-edge collapse

- Simplify edge e: (w, v)
- Requires:
  - VT relation for vertex v
  - VT relation for vertex w
  - ET relation for edge e



- Steps:
  1. Delete tetrahedra in ET – applies to ***T***
  2. Modify vertices of tetrahedra in VT(v) – applies to ***V***
  3. Delete vertex v – applies to ***V***
  4. Add tetrahedra in VT(v) to VT(w) and remove ET(e)

     applies to local data structure
  5. Remove VT(v) – applies to local data structure

Simplification Algorithm

- Repeat the following until there is not change

- ALGORITHM: SIMPLIFYMESH()
  - for each node n of N
    - Generate VT relation of all vertices v_n
    - Enqueue all edges to be checked for collapse
    - while (queue is not empty)
      - Edge e = top element of queue
      - if (e passes test for simplification)
        - EDGECOLLAPSE (e)
  - SIMPLIFYOCTREE(N)   // by merging sibling leaf nodes

Results

- Compare PR-star with different k_v values
- Special case: $k_v = \infty$
  - Octree only has a single node

- Summary:
  - Similar simplification results
    - Around the same number of tetrahedra removed
    - In around the same amount of time ($\pm$ 20%)
    - using < 1% of the memory

Trend: Better results for larger meshes and larger values of k_v

Discussion

- Introduced PR-star Octree for tetrahedral meshes
  - *Spatio-Topological* approach
- Spatial index "for free"
  - One of the difficulties in topological data structures on spatial data is finding the initial vertices
- Simple global data structure
- Optimal local data structures
  - Not forced to decide in advance which operations (e.g. incidence, adjacency) to optimize
  - Efficiently build the data structure at runtime without worrying (too much) about memory consumption
- Results improve with increased mesh resolution

Limitations

- Only works for spatial meshes
  - Use traditional topological data structure for abstract complexes
- Does not replace spatial data structures
  - Not optimized for general spatial queries
    - E.g. point location (find tetrahedron containing a point)
  - Use PM-family of meshes here
  - But can handle range queries

Future work

- Tuning for parameter k_v
  - Preliminary results: k_v ~ 600-800 appears to be the sweet spot
    - Significantly smaller octrees
    - More time to build the local data structures but less time to traverse the octree
    - Not "too much" extra time to generate the local data structure
- Cache-based algorithms for non-local processing of mesh
  - e.g. simplification of edges spanning two octree nodes
  - Use a cache of expanded nodes
    - Preliminary results: Around 2% of nodes is sufficient for best results
- Exploit inherent parallelism of data structure

Thank you

- Questions? Comments?

- Anonymous reviewers
- Funding Sources
  - NSF Grant IIS-1116747
  - Italian MUIR-PRIN 2009
- Paola Magillo
- Mesh sources
  - AIM@Shape, Volvis
  - C. Silva, R. Haimes