Storage of massive TINs in a DBMS
A comparison and a prototype implementation of the multistar approach
Storage of massive TINs in a DBMS
A comparison and a prototype implementation of the multistar approach

M.Sc. Geomatics P5 presentation
Maarten Pronk
22 June, 2015

Main mentor: Prof. dr. J. (Jantien) Stoter
Second mentor: dr. H. (Hugo) Ledoux
Co-reader: Dr. Ir. M. (Martijn) Meijers
Multistar

- Fast, small and works on massive datasets
- 41291 intersections in 4secs.
Outline

Introduction
Efficient approach?
Current approaches
Multistar approach
Comparison
Conclusion
Introduction
Efficient approach?
Current approaches
Multistar approach
Comparison
Conclusion
TIN datasets

- AHN2 filtered $\rightarrow$ delaunay triangulation with lastools
Pointcloud

- xyzrgbetc
- AHN2
  filtered
Operations

- Reconstruct original surface
- (Delaunay) Triangulation
- TIN:
  Triangulated Irregular Network
Massive?

1m
30 triangles
Massive?

100m
300000 triangles
Massive?

100m
300000 triangles, ~5000 in each bucket
Massive?

1km
30M triangles
2.5km
750M triangles

Massive
Research question

- What are efficient approaches to store massive TINs in a DBMS?
- Efficient as in
  - Storage size of data structure
  - Storage size of index
  - Performance of spatial queries
  - Availability of atomic functions
  - Loading time of the TIN, including construction
Research questions

- What are efficient approaches to store massive TINs in a DBMS?
  - What are reasons for creating patches or block in a DBMS and what are the trade offs?
  - Which spatial index is the most efficient?
  - Which atomic functions are needed for a TIN in a DBMS?
  - How can the topological relationships of a TIN be exploited in these functions?
Methodology

LITERATURE STUDY

EFFICIENT APPROACH

THEORETICAL COMPARISON

CURRENT IMPLEMENTATIONS

DATABASE STUDY

THEORY

PRACTICE

PROTOTYPE

PRACTICAL COMPARISON
Introduction

Efficient approach?

Current approaches

Multistar approach

Comparison

Conclusion
Usage
Atomic functions
Atomic functions
Atomic functions
Data structures

Nodes

Edges

Faces
Triangle based

- SimpleFeatures (OGC 2006)
- triangle s \((a_x \ a_y \ a_z, \ b_x \ b_y \ b_z, \ c_x \ c_y \ c_z, \ a_x \ a_y \ a_z)\)
  - Storing the same coordinates many times over

300n size
Node based

- Vertices array [ vid, x, y, z ]
- Triangle array [ tid, vid1, vid2, vid3 ]
  - Keeps pointcloud

60n size

<table>
<thead>
<tr>
<th>tidd</th>
<th>v1</th>
<th>v2</th>
<th>v3</th>
<th>NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>II</td>
</tr>
<tr>
<td>II</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>I</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>vid</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$x_a$</td>
<td>$y_a$</td>
<td>$z_a$</td>
</tr>
<tr>
<td>b</td>
<td>$x_b$</td>
<td>$y_b$</td>
<td>$z_b$</td>
</tr>
<tr>
<td>c</td>
<td>$x_c$</td>
<td>$y_c$</td>
<td>$z_c$</td>
</tr>
<tr>
<td>d</td>
<td>$x_d$</td>
<td>$y_d$</td>
<td>$z_d$</td>
</tr>
</tbody>
</table>
**Node (star) based**  (Blandford et al 2005)

Stores *neighbours*

52n size

$p[a,b]$

$p[a,b]$

$p[a,b,c,d,e,f,g]$
Buckets

- Massive datasets often split
  - AHN2 comes in tiles
Buckets

- Break a TIN
TIN indexes

• Massive datasets:
  can't index individual components
  – Bounding box of 1 point is 200% increase
  – Bounding box of 1 edge is 100% increase
  – Bounding box of 1 triangle is 66% increase

• Index buckets
Inside bucket
Inside bucket

- Walking
Inside bucket

- Walking
Inside bucket

- Walking
Inside bucket

- Walking
Inside bucket

- Walking
Inside bucket

- Walking
Inside bucket

- Walking
Inside bucket

- Walking
Inside bucket

- Walking
Inside bucket

- Walking
Massive TIN access

- Spatial index on buckets
- Walking or bruteforcing inside a TIN
  - Walking requires convex TIN
Efficient approach!

- Atomic functions
  - Elevation, slope, aspect, local minimum and maximum
- Topological relationships
  - For atomic functions
  - For efficient access
- Buckets
  - For efficient storage
  - For efficient access
- Nodes
  - Smallest data structure
Introduction
Efficient approach?
Current approaches
Multistar approach
Comparison
Conclusion
SF TIN

- Simple Features TIN
  - Well known
  - Subset of Polygon & PolyhedralSurface
  - Needs SFCGAL
  - Only 3D operations
ORACLE SDO_TIN

● Oracle SDO_TIN
  – Partitions of pointclouds / triangles
  – Solves bucket problem
  – Triangle vertices referring to partition and id

<table>
<thead>
<tr>
<th>Patch 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tid</td>
<td>v1</td>
<td>v1 patch</td>
<td>v2</td>
<td>v2 patch</td>
<td>v3</td>
</tr>
<tr>
<td>l</td>
<td>a</td>
<td>1</td>
<td>b</td>
<td>1</td>
<td>c</td>
</tr>
<tr>
<td>ll</td>
<td>a</td>
<td>1</td>
<td>d</td>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>lll</td>
<td>b</td>
<td>1</td>
<td>c</td>
<td>1</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Patch 1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>vid</td>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>a</td>
<td>x_a</td>
<td>y_a</td>
<td>z_a</td>
</tr>
<tr>
<td>b</td>
<td>x_b</td>
<td>y_b</td>
<td>z_b</td>
</tr>
<tr>
<td>c</td>
<td>x_c</td>
<td>y_c</td>
<td>z_c</td>
</tr>
</tbody>
</table>
TIN academics

- PgTIN prototype and paper (Ledoux, 2013)
  - Star based storage
  - Uses walking
  - Comparisons done to single features
  - Bucket index by external structure
# TIN summary

<table>
<thead>
<tr>
<th></th>
<th>Node storage</th>
<th>Buckets</th>
<th>Topological relationships</th>
<th>Atomic functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OGC</strong></td>
<td></td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Postgis</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TIN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Oracle</strong></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td><strong>TIN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pgTIN</strong></td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Introduction
Efficient approach?
Current approaches
Multistar approach
Comparison
Conclusion
Motivation

- PgTIN and Oracle are best and second best approaches
- PgTIN needs a bucket structure
- Oracle needs topology
Multistar

- Star structure based on Blandford 2005 and Ledoux 2013

- Streaming by lastools
- Using buckets based on quadtree
- Index on buckets
- Walking in buckets
Buckets

- By lastools
- 1st run: finalize → delaunay → qtfin.py
- 2nd run finalize → delaunay → smb2db.py → DB
Bucketing strategies

- Not bridging gaps or *islands*
  - *Used for SF comparison, such as MultiPolygonZ*
  - Either gaps | overlap | ruleset
  - *Results in non convex TIN!*

---

53
Bucketing strategies

- Bridging gaps
  - Loose reference
  - *Dangling*
Bucketing strategies

- Bridging gaps directly
  - Relative or
  - Absolute id
  - :-)
Multistar

id 1 bucket 1
star [0, 2, 1, 2, 2, 3, 2]

id 2 bucket 1
star [0, 1, 3]

id 3 bucket 1
star [0, 2, 1, 2, 2, 4]

id 4 bucket 1
star [0, 3, 2, 2, 2, 4]

bucket 1
bucket 2

id 1 bucket 2
star [0, 3, 2, 1, 1]

id 2 bucket 2
star [3, 4, 1, 4, 1, 3, 1, 1, 1]

id 4 bucket 2
star [0, 1, 4, 2, 3]

id 3 bucket 2
star [0, 4, 3, 1]
Multistar

<table>
<thead>
<tr>
<th>id</th>
<th>offsets</th>
<th>points</th>
<th>stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[-4,-5,-8,-13,-17]</td>
<td>xyzxyzxyzxyz</td>
<td>[0,(2,1),(2,2),3,2,0,1,3,0,2,1(2,2),4,0,3,(2,2),(2,4)]</td>
</tr>
<tr>
<td>2</td>
<td>[-4,-4,10,-14,-18]</td>
<td>xyzxyzxyzxyz</td>
<td>[0,3,2,(1,1),3,4,(1,4),(1,3),(1,1),1,0,4,3,1,0,(1,4),2,3]</td>
</tr>
</tbody>
</table>

#
# Multistar

<table>
<thead>
<tr>
<th>id</th>
<th>offsets</th>
<th>points</th>
<th>stars</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>int int[]</td>
<td>bytea</td>
<td>bytea</td>
</tr>
<tr>
<td>1</td>
<td>[-4,-5,-8,-13,-17]</td>
<td>xyzxyzxyzxyz</td>
<td>[0,(2,1),(2,2),3,2,0,1,3,0,2,1(2,2),4,0,3,(2,2),(2,4)]</td>
</tr>
<tr>
<td>2</td>
<td>[-4,-4,10,-14,-18]</td>
<td>xyzxyzxyzxyz</td>
<td>[0,3,2,(1,1),3,4,(1,4),(1,3),(1,1),1,0,4,3,1,0,(1,4),2,3]</td>
</tr>
</tbody>
</table>
### Multistar

<table>
<thead>
<tr>
<th>id</th>
<th>offsets</th>
<th>points</th>
<th>stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>int[]</td>
<td>bytea</td>
<td>bytea</td>
</tr>
<tr>
<td>12</td>
<td>[-4,-5,-8,-13,-17]</td>
<td>xyzxyzxyzxyzxyz</td>
<td>[0,(2,1),(2,2),3,2,0,1,3,0,2,1(2,2),4,0,3,(2,2),(2,4)]</td>
</tr>
<tr>
<td></td>
<td>[-4,-4,10,-14,-18]</td>
<td>1234</td>
<td>[0,3,2,(1,1),3,4,(1,4),(1,3),(1,1),1,0,4,3,1,0,(1,4),(2,3)]</td>
</tr>
</tbody>
</table>

#
Multistar
### Multistar

<table>
<thead>
<tr>
<th>id</th>
<th>offsets</th>
<th>points</th>
<th>stars</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>int int[]</td>
<td>bytea</td>
<td>bytea</td>
</tr>
<tr>
<td>12</td>
<td>[-4,-5,-8,-13,-17]</td>
<td>xyzxyzxyzxyz</td>
<td>[0,(2,1),(2,2),3,2,0,1,3,0,2,1(2,2),4,0,3,(2,2),(2,4)]</td>
</tr>
<tr>
<td></td>
<td>[-4,-4,10,-14,-18]</td>
<td>xyzxyzxyzxyz</td>
<td>[0,3,2,(1,1),3,4,(1,4),(1,3),(1,1),1,0,4,3,1,0,(1,4),2,3]</td>
</tr>
</tbody>
</table>
# Multistar

<table>
<thead>
<tr>
<th>id</th>
<th>offsets</th>
<th>points</th>
<th>stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>int[]</td>
<td>bytea</td>
<td>bytea</td>
</tr>
<tr>
<td>12</td>
<td>[-4,-5,-8,-13,-17]</td>
<td>xyzxyzxyzxyz</td>
<td>[0,(2,1),(2,2),3,2,0,1,3,0,2,1,(2,2),4,0,3,(2,2),(2,4)]</td>
</tr>
<tr>
<td></td>
<td>[-4,-4,10,-14,-18]</td>
<td>xyxxyzxyzxyz</td>
<td>[0,3,2,(1,1),3,4,(1,4),(1,3),(1,1),1,0,4,3,1,0,(1,4),2,3]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
</tr>
</tbody>
</table>
Multistar

<table>
<thead>
<tr>
<th>id</th>
<th>offsets</th>
<th>points</th>
<th>stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>[-4,-5,-8,-13,-17]</td>
<td>xyzxyzxyzxyz</td>
<td>[0,(2,1),(2,2),3,2,0,1,3,0,2,1(2,2),4,0,3,(2,2),(2,4)]</td>
</tr>
<tr>
<td></td>
<td>[-4,-4,10,-14,-18]</td>
<td>xyzxyzxyzxyz</td>
<td>[0,3,2,(1,1),3,4,(1,4),(1,3),(1,1),1,0,4,3,1,0,(1,4),2,3]</td>
</tr>
</tbody>
</table>

#
Drawbacks

- Lookup table for offsets (variable length)
- Indirect triangle storage
Introduction
Efficient approach?
Current approaches
Multistar approach
Comparison
Conclusion
Generation

• Internal
  – Postgis  SP_DelaunayTriangles
  – Oracle   SDO_TIN

• External
  – Lastools  sp_delaunay2d.exe
  – Triangle
Generation

- **Internal**
  - Postgis: `SP_DelaunayTriangles` 3600
  - Oracle: `SDO_TIN` did not run

- **External**
  - Lastools `sp_delaunay2d.exe` 10
  - Triangle 31

*time in seconds for 5M points*
Buckets by Oracle
Buckets by Multistar
Buckets by MultipolygonZ
Table size
Massive Table size
Postgresql specifics

- Row overhead on non bucketed types
- Compression by TOAST on bucketed types
Table size *uncompressed*

![Bar chart showing table size comparison](chart.png)
Performance

Point location

Cold Query Time in [ms] (logarithmic)

Bucket level (higher more buckets, with lesser points)
Performance

Range query

- MultiPolygonZ
- Triangle Array
- Multistar
- TINZ
- pgTIN
- PolygonZ
- TriangleZ

Cold Query Time in [ms] (logarithmic)

Bucket level (higher more buckets, with lesser points)
Summary

- Bucketed types are smaller
- Index on every point or triangle is very large
- Larger buckets slow down queries
- Using topology is fast
Introduction
Efficient approach?
Current approaches
Multistar approach
Comparison
Conclusion
Research question

• What are efficient approaches to store massive TINs in a DBMS?
Efficient approach

- Stores nodes
  - Small data structure
  - Stores pointcloud
Efficient approach

- Has atomic functions
  - Enables more complex functions
  - Prevents transfer of massive data
Efficient approach

• Stores topological relationships
  – For atomic functions
  – For spatial access
Efficient approach

- Uses buckets
  - To split the TIN in non-massive sections
  - For smaller spatial indexes
Conclusion

- An efficient approach with
  - Nodes
  - Atomic functions
  - Topological relationships
  - Buckets

- is validated in the Multistar prototype
Discussion

- Multistar not the only possible implementation
- Other efficient approaches could exist
Recommendation

- Massive TIN
  - Solved in generation & loading
  - Solved in storing
  - NOT solved in output
Future work

- Multiresolution TIN
- Updates
- Triangle Array + topological relationships
Questions?