R.REFINE: Scalable Raster to TIN Simplification

Jonathan Todd   Laura Toma
Bowdoin College

FOSS4G 2006
Lausanne, Switzerland
Data is GROWING!!!

- NASA’s SRTM mapped 80% of the earth at 30 meter resolution
  - SRTM data set: 300,000 x 300,000 raster
- USGS & NASA publicly release terabytes of data
- LIDAR data collection produces extremely large data sets at high resolution
**DEM Representations**

**Contour Lines**

**Raster**

**TIN**

**Sample Points**
**Raster - TIN Comparison**

**Rasters**
- Fixed Resolution
- Implicit Topology
  - Don’t need to store adjacency explicitly
- Simple algorithms
- Large amount of grid data available
- Most Commonly Used

**TINs**
- Variable resolution
- Topology needs to be stored explicitly
- Algorithms are more complex
- Data needs to be converted into a TIN
- Somewhat less popular than grids
Variable Resolution

Flat Area

Raster - 80 pts

TIN - 11 pts, 12 tris
Representing Massive data

- With rasters, the same amount of space is used to represent
  - a mountainous region (Himalayas)
  - a flat area (Mohave desert)

- **Space efficiency becomes more important for massive data!**
- **Increased space efficiency can significantly reduce run time**
Scalable raster-to-TIN Simplification

- raster-to-TIN simplification
  - simplify raster to TIN which approximates the raster within a user specified error threshold
  - intuitively: drop points in the raster that are redundant
- **Scalable** raster to TIN simplification
  - efficient when size of input raster becomes very large
Scalable raster-to-TIN simplification module
  • Input: raster, error threshold $e$
  • Output: simplified TIN
Based on an I/O efficient algorithm
Outline

- [Introduction]
- Raster simplification
- r.refine
- Results
  - Scalability
  - Space efficiency
- Conclusion & Future Work
Raster Simplification
Problem:
- Given a raster with points $P$ and an error $\varepsilon$, find $S \subseteq P$ which approximates $P$ within $\varepsilon$: that is, every point in $S$ is within distance $\varepsilon$ of $P$. 
Refinement Heuristics

- Start with 4 corner pts of raster
- Repeat:
  - Find point with largest error
  - Add point to triangulation
  - If no more points with error > $\varepsilon$ Then break;

\[ \varepsilon = 1 \]
Refinement: Adding Points

- If point not collinear add 3 triangles
- If collinear add 4 triangles
Delaunay Triangulation

- Delaunay is a type of triangulation which has the property of maximal minimum angle. (Triangles are fat)
- A triangle is locally Delaunay if its circum-circle does not contain any other points in the triangulation
- Delaunay is desirable because it reduces rounding errors and has shown to reduce triangles in a TIN

![Delaunay](image1)

![Not Delaunay](image2)

![Edge flipping](image3)
Scalability

- Refinement is not scalable
- Refinement requires random access to data
  - If data-size > mem-size run time is very long
  - GRASS segment library does not fix this
- Large data-sets necessitate scalability
R.REFINE

A scalable approach for raster-to-TIN simplification
Tiling for I/O-Efficiency

- Tiling is a common I/O optimization technique
  - Take size of memory as parameter
  - Separate large grid into tiles
  - Each tile is small enough to fit in memory
  - Refine each tile individually then write to disk

Too big for memory

```
1 2 2 3 3 5
1 2 2 3 3 4
1 1 2 3 3 4
1 1 2 2 3 4
2 2 2 2 3 4
3 2 2 2 3 4
```

Tiled Raster

```
1 2 2 3 3 5
1 2 2 3 3 4
1 1 2 3 3 4
1 1 2 2 3 4
1 1 2 2 3 4
2 2 2 2 3 4
3 2 2 2 3 4
```
Our Refinement

- We use the standard refinement algorithm within each tile
- We maintain Delaunay triangulation while building the TIN

```
REFINETIN TILE(e, tile)
1 pq ← PQ-INIT()
2 tt ← INIT TIN TILE(tile, pq)
3 while s ← PQ-EXTRACTMAX(pq) and s ≠ NIL
4 do if isCOLLINEAR(s)
5 then FIXCOLLINEAR(s)
6 else t1 ← ADDTRI(s, s.p1, s.p2, s.maxError, tt)
7 t2 ← ADDTRI(s, s.p1, s.maxError, s.p3, tt)
8 t3 ← ADDTRI(s, s.maxError, s.p2, s.p3, tt)
9 DISTRIBUTE POINTS(t1, t2, t3, s, e, pq, tt)
10 REMOVE TRIANGLE(s)
11 ENFORCE DELAUNAY(t1, t1.p1, t1.p2, t1.p3, e, tt)
12 ENFORCE DELAUNAY(t2, t1.p1, t1.p3, t1.p2, e, tt)
13 ENFORCE DELAUNAY(t3, t1.p2, t1.p3, t1.p1, e, tt)
14 return tt
```
Two structures:
- Triangles
- Vertices

Triangles store:
- Pointer to adjacent triangles
- Pointer to vertices
- List of points inside

Points store:
- Location (x, y, z)

TIN is accessed through lower left vertex V
Combining Tiles

- Need to combine tiles such that boundary points are consistent
- We refine one tile at a time starting with the upper left tile. We maintain consistency by adding points to right and bottom neighbors.
- There is no known way to maintain Delaunay globally and I/O-efficiently.
Using r.refine
Running r.refine

- Flags
  -d  Don’t use Delaunay
  -n  Include nodata points
  -r  Render

- Parameters
  - Input grid
  - Epsilon (% of Max Elevation)
  - Output TIN
  - Output sites
  - Output vector
  - Memory (Default 500 MB)

Description:
r.refine: scalable raster-to-TIN simplification.

Usage:
r.refine [-dnr] grid=name [epsilon=value] [tin=name]
  [output_sites=name] [output_vect=name] [memory=value]

Flags:
  -d  Do NOT use Delaunay triangulation
  -n  Include nodata points (more points, better boundaries)
  -r  Render TIN in OpenGL

Parameters:
  grid  Input raster
  epsilon  Error threshold, in percentage of max elevation
    default: 1.0
  tin  Output TIN file
    default: output.tin
  output_sites  Name of output sites file.
    default: NULL
  output_vect  Name of output vector file.
    default: NULL
  memory  Main memory size (in MB)
    default: 500
r.refine Output

GRASS:~/nfs-gis/> r.refine grid=elev eps=3 output_sites=eleve3 output_vect=eleve3
region size is 472 x 391
r.refine grid=elev output=output.tin output=sites=eleve3 outputVect=eleve3
error=3.00 mem=500.00 delaunay=1 no_data=0 render=0
raster2grid: reading raster elev....done
refining
write TIN tile to sites file eleve3
  100%
write TIN tile to vect file eleve3
done refining
........DONE........
err=3.00% absErr=27.48 mem=500.00MB numTiles=1
raster: 184552 points
TIN: triangles=2350 points=1183
total time: 1.70      99.9%
Figure 15: Effect of Mountain in a Delaunay Triangulation
Figure 16: Simplifying the same raster: (a) Delaunay is maintained (b) Delaunay is ignored.
Results
Test Platform

- Apple Dual Processor G5
- 2.5 GHZ CPU
- 1 GB RAM
- Data Sets from 1.6 million to 122 million points
Tile vs Untiled Runtime Comparison

1% Error

Figure 8: Tile vs Untiled Runtime Comparison on Datasets with 0% Error

Figure 9: Tile vs Untiled Runtime Comparison on Datasets with 1% Error
Figure 8: Tile vs Untiled Runtime Comparison on Datasets with 1% Error

Figure 9: Tile vs Untiled Runtime Comparison on Datasets with 0.1% Error
**Tiled vs Comparison On Appalachians**

![Graph showing the comparison between Tiled and Untiled versions of the Appalachians dataset. The x-axis represents the number of data points in millions, ranging from 0 to 10. The y-axis represents run time in seconds, ranging from 0 to 5,000. Two lines are plotted: one for the Untiled version and one for the Tiled version. The Untiled version shows a significant increase in run time as the number of data points increases, while the Tiled version maintains a relatively flat line, indicating better performance.](image-url)
Grid vs TIN Point Comparison

0.1% Error

Figure 11: Grid vs. TIN Point Comparison (0.1% Error)
Grid vs TIN Size Comparison

0.1% Error

Grid Size MB
TIN Size MB
Grid vs TIN Size Comparison

1% Error

Size (MB)

Grid Size MB  TIN Size MB

kaweah   prtoric   sierra   hawaii   appalach   cumberla   lowerne
Grid vs TIN Size Comparison

5% Error

![Graph showing Grid Size MB and TIN Size MB for different locations. The graph indicates a significant 5% error in size comparison.](image-url)
Effects of Delaunay on Number of Triangles

Figure 15: Effect of Mountain in a Delaunay Triangulation.

Figure 16: Simplifying the same raster: (a) Delaunay is maintained (b) Delaunay is ignored.
Conclusions & Future Work
Future Work

- Assure quality of data. (No artificial dams or ridges)
- Apply flow modeling to TINs
- Parallelize code
- Take sample points (LIDAR) as input
In Conclusion

- r.refine provides a starting point for work on TINs
  - Available in GRASS
- TINs can represent rasters of any kind
  - Any module for raster can be done on TIN