r.refine: Scalable Raster to TIN Simplification

Jonathan Todd   Laura Toma
Bowdoin College

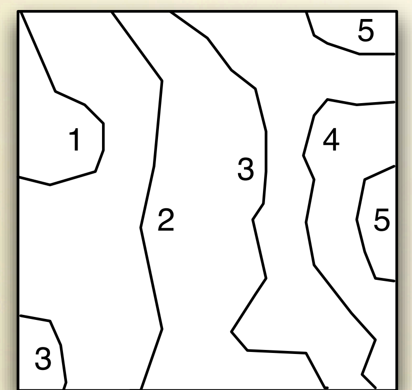
FOSS4G 2006
Lausanne, Switzerland

Data is GROWING!!!

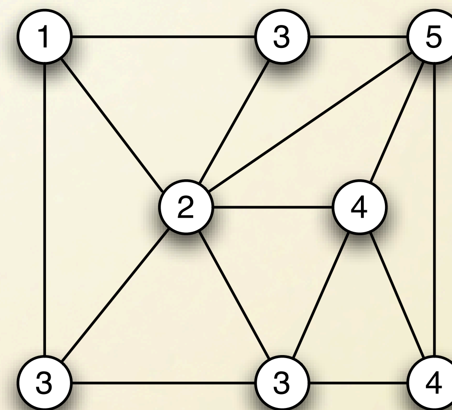
- NASA's SRTM mapped 80% of the earth at 30 meter resolution
    - SRTM data set: 300,000 x 300,000 raster
- USGS & NASA publicly release terabytes of data
- LIDAR data collection produces extremely large data sets at high resolution

DEM Representations

Contour Lines



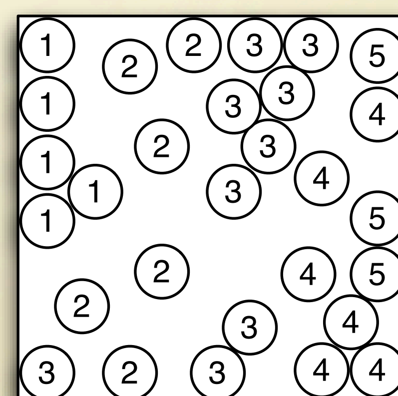
TIN



Raster

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

Sample Points



Raster - TIN Comparison

Rasters

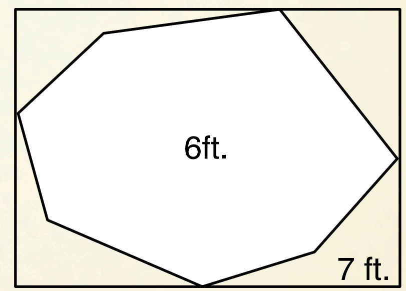
- Fixed Resolution
- Implicit Topology
  - Don't need to store adjacency explicitly
- Simple algorithms
- Large amount of grid data available
- Most Commonly Used

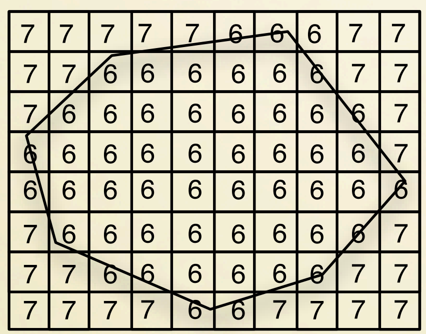
TINs

- 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

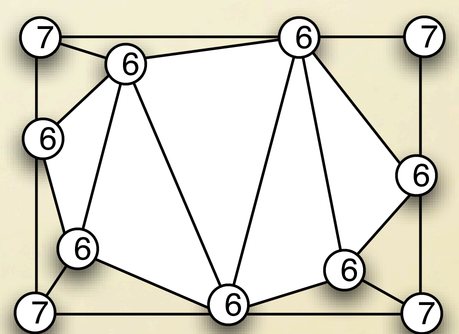
Flat Area





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

R.REFINE

- Scalable raster-to-TIN simplification module
  - Input: raster, error threshold e
  - Output: simplified TIN
- Based on an I/O efficient algorithm

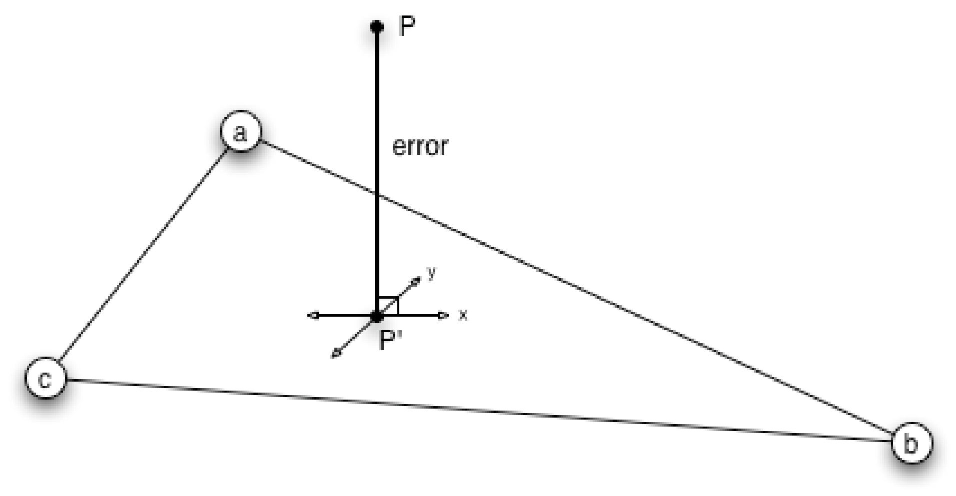
Outline

- [Introduction]
- Raster simplification
- r.refine
- Results
  - Scalability
  - Space efficiency
- Conclusion & Future Work

Raster Simplification

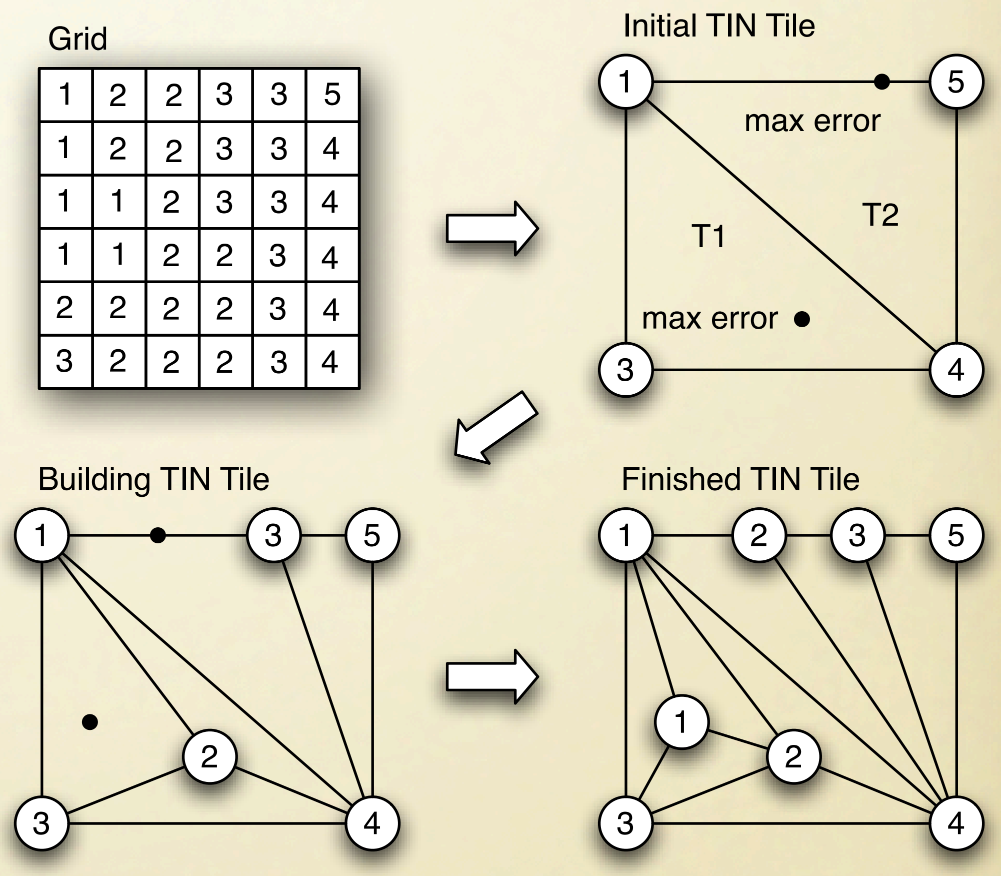
Raster Simplification

- Problem:
  - Given a raster with points ***P*** and an error ε, find ***S*** ∈ ***P*** which approximates ***P*** within ε: that is, every point in ***S*** is within distance ε of ***P***.



Refinement Heuristics

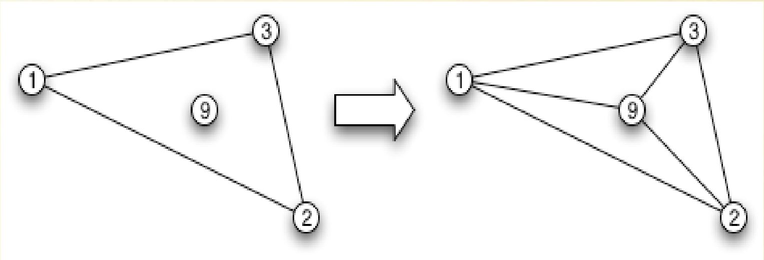
- Start with 4 corner pts of raster
- Repeat:
  - Find point with largest error
  - Add point to triangulation
  - If no more points with error > ε Then break;

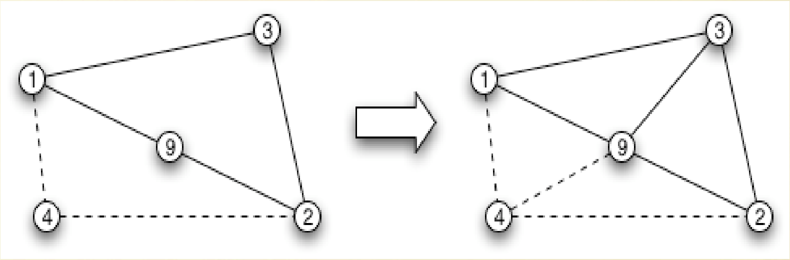


$\varepsilon = 1$

Refinement: Adding Points

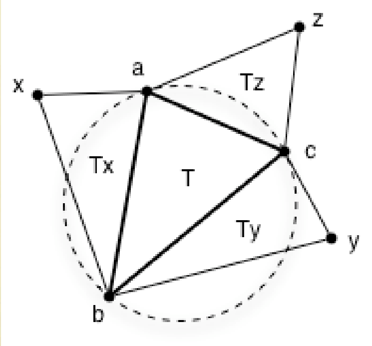
- If point not collinear add 3 triangles
- If collinear add 4 triangles



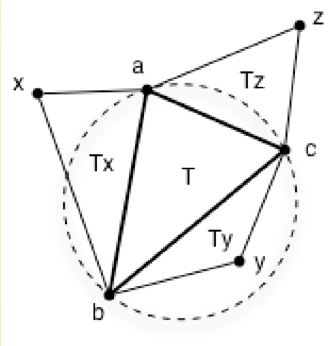


Delaunay Triangulation

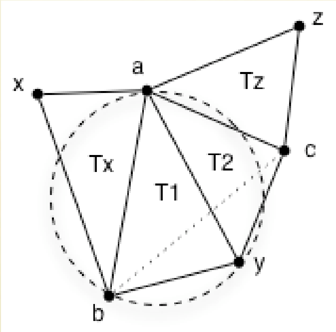
- Delaunay is a type of triangulation which has the property of maximal minimum angle. (Triangles are fat)
- A triangle is locally Delaunay if its circum-circle does not contain any other points in the triangulation
- Delaunay is desirable because it reduces rounding errors and has shown to reduce triangles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



Delaunay



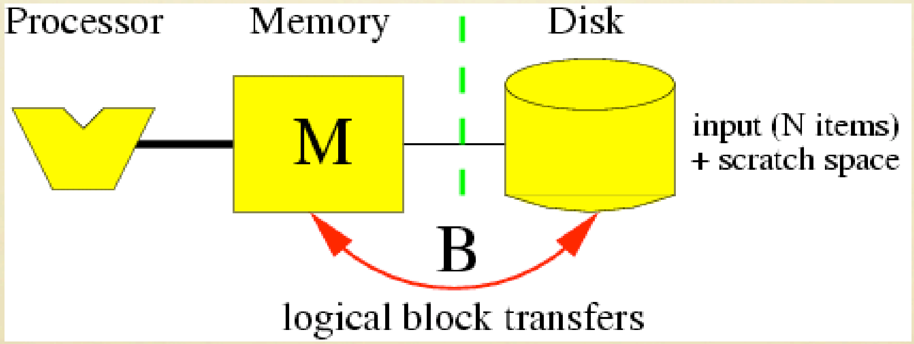
Not Delaunay



Edge flipping

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

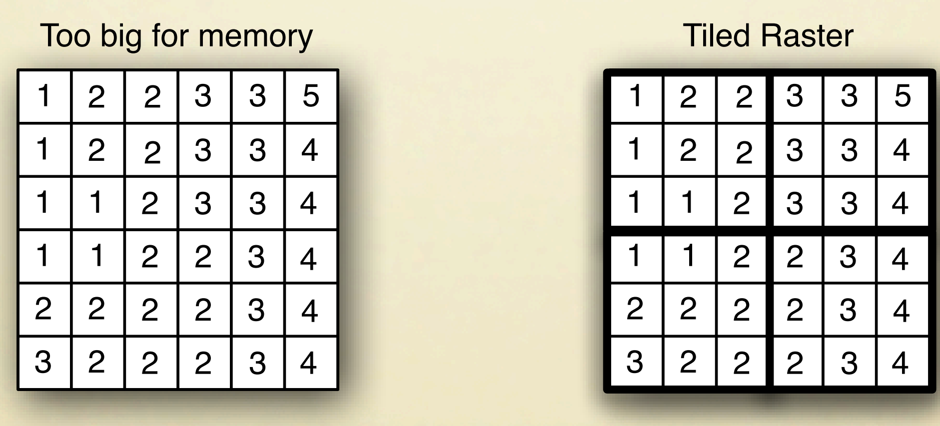


R.REFINE

A scalable approach for raster-to-TIN simplification

Tiling for I/O-Efficiency

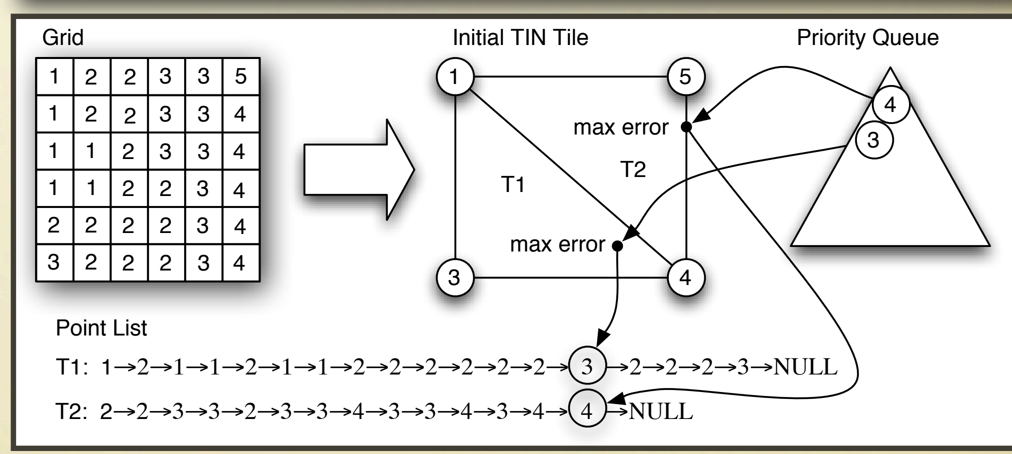
- Tiling is a common I/O optimization technique
  - Take size of memory as parameter
  - Separate large grid into tiles
  - Each tile is small enough to fit in memory
  - Refine each tile individually then write to disk



Our Refinement

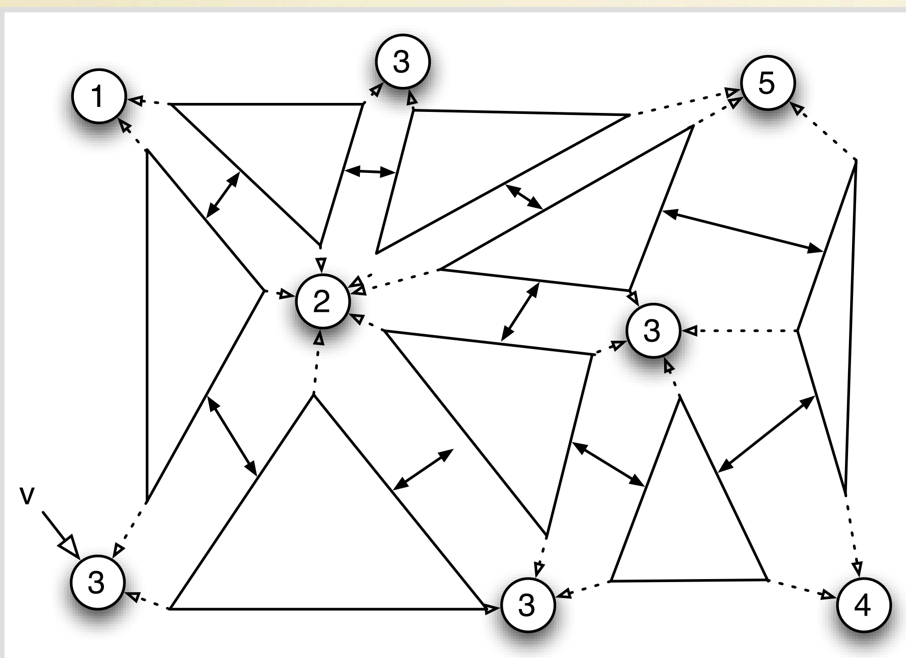
- We use the standard refinement algorithm within each tile
- We maintain Delaunay triangulation while building the TIN

```
refineTinTile(e, tile)
pq ← PQ-init()
tt ← initTinTile(tile, pq)
while s ← PQ-extractMax(pq) and s ≠ nil
do if isCollinear(s)
      then fixCollinear(s)
      else  t1 ← addTri(s, s.p1, s.p2, s.maxError, tt)
            t2 ← addTri(s, s.p1, s.maxError, s.p3, tt)
            t3 ← addTri(s, s.maxError, s.p2, s.p3, tt)
            distributePoints(t1, t2, t3, s, e, pq, tt)
            removeTriangle(s)
            enforceDelaunay(t1, t1.p1, t1.p2, t1.p3, e, tt)
            enforceDelaunay(t2, t1.p1, t1.p3, t1.p2, e, tt)
            enforceDelaunay(t3, t1.p2, t1.p3, t1.p1, e, tt)
return tt
```



TIN Structure

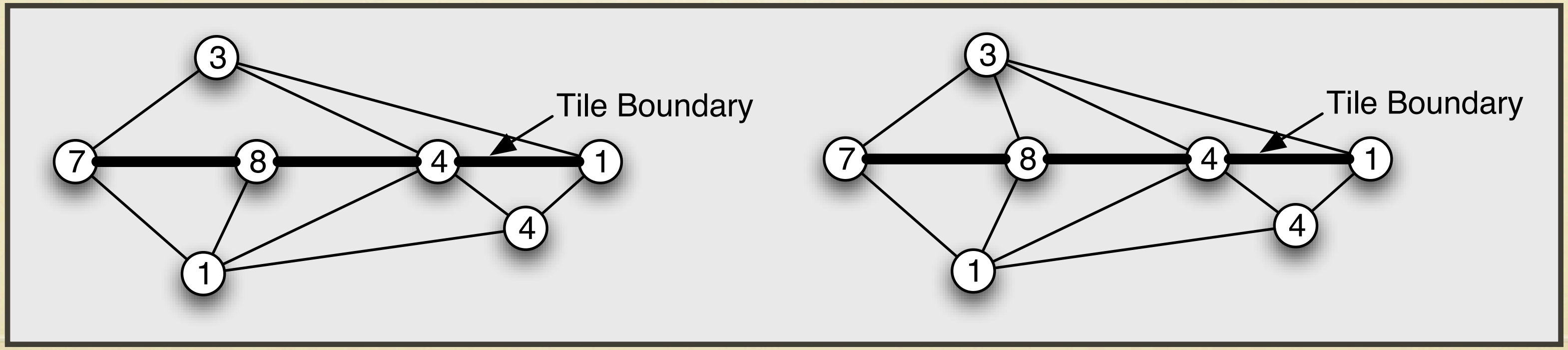
- Two structures:
  - Triangles
  - Vertices



- 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

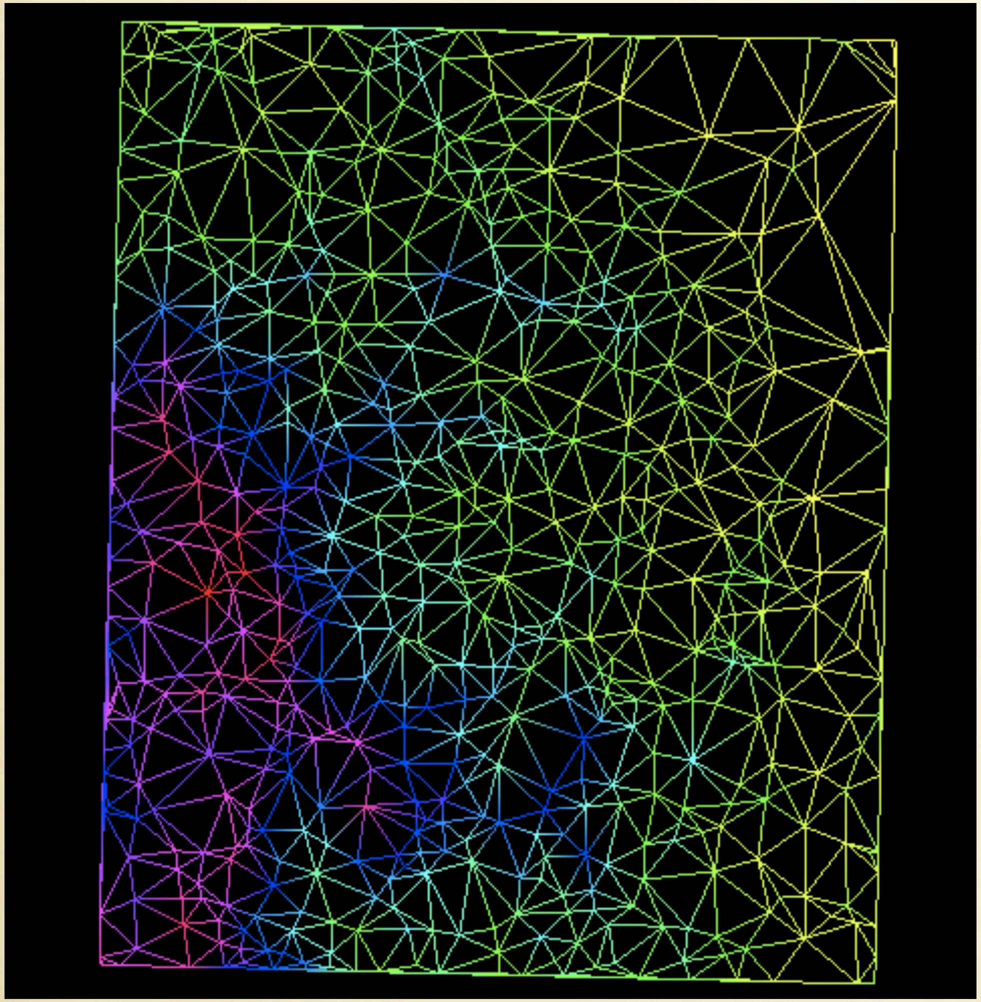
Running r.refine

- 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)

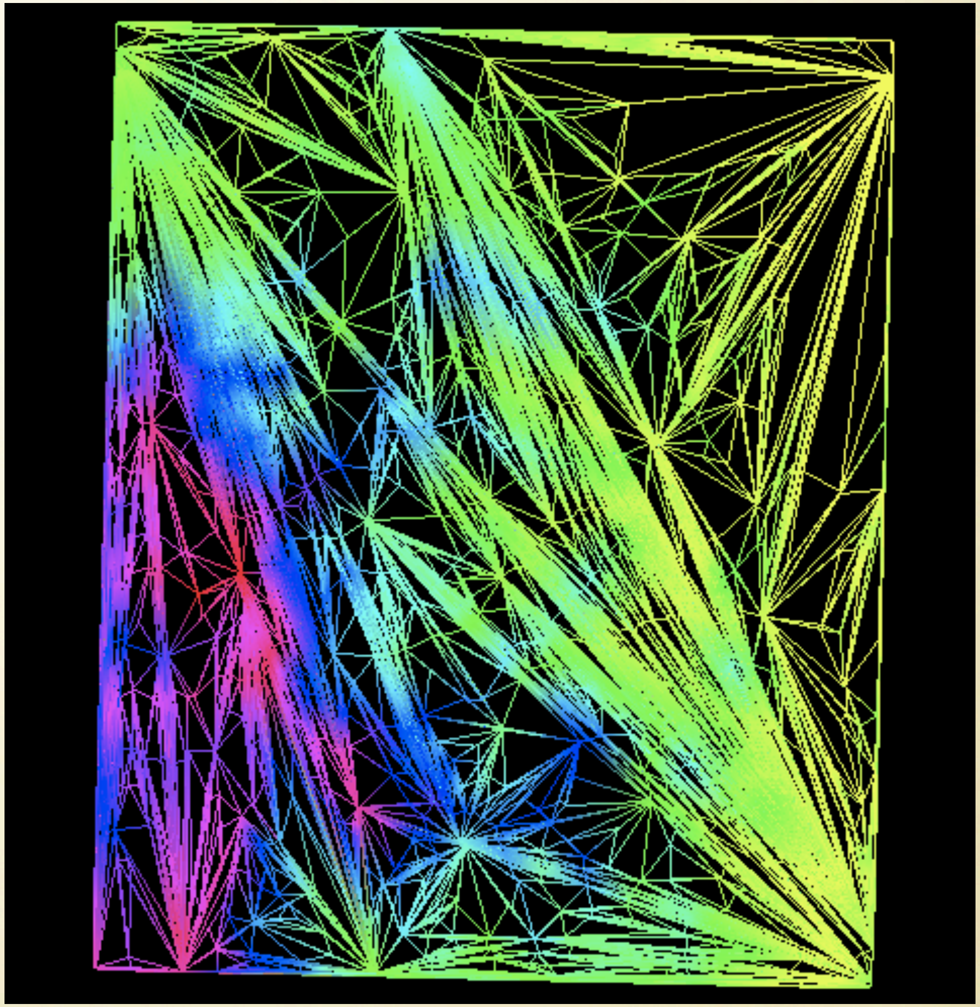
```
Description:
 r.refine: scalable raster-to-TIN simplification.

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%
```



Delaunay



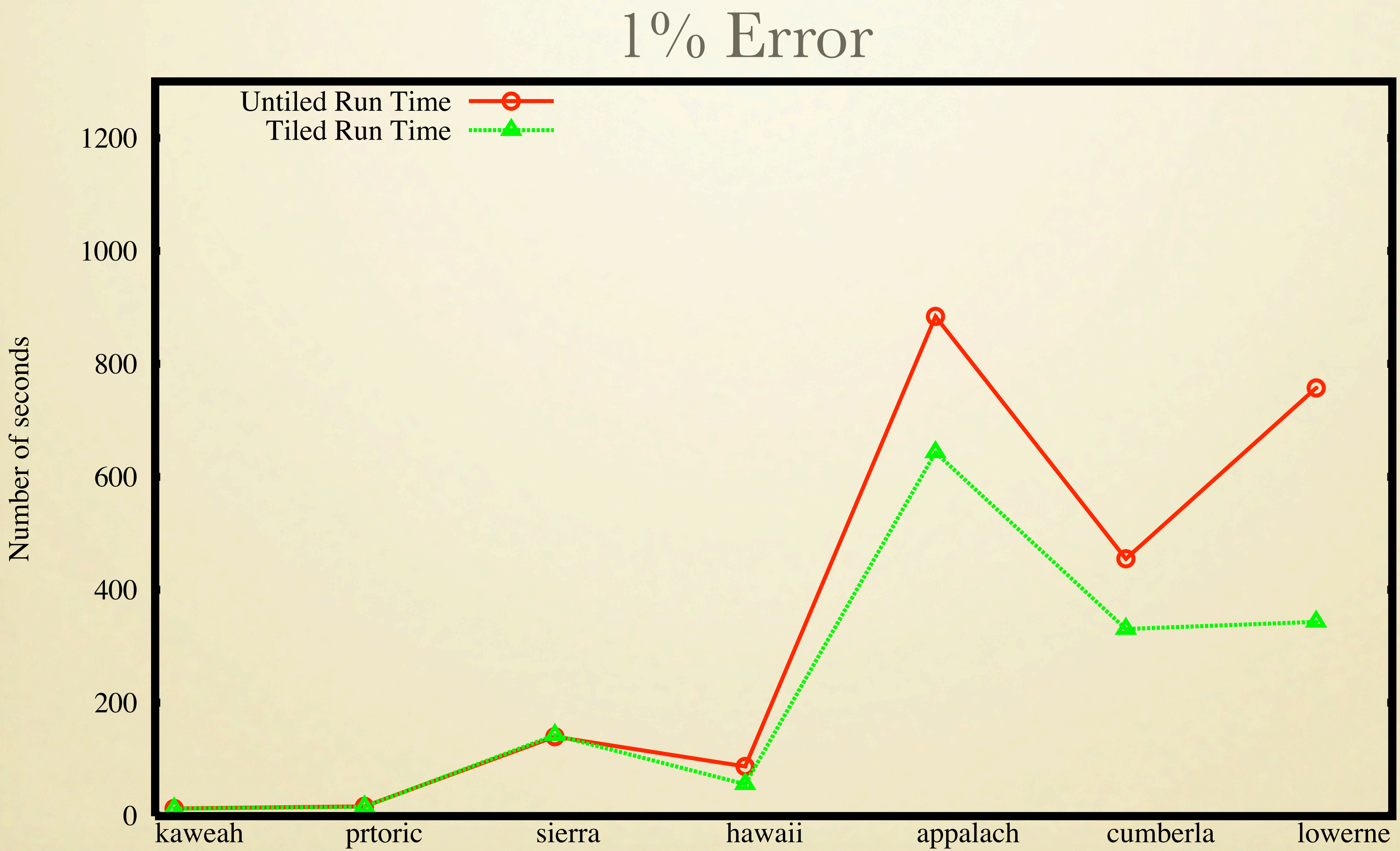
Non-Delaunay

Results

Test Platform

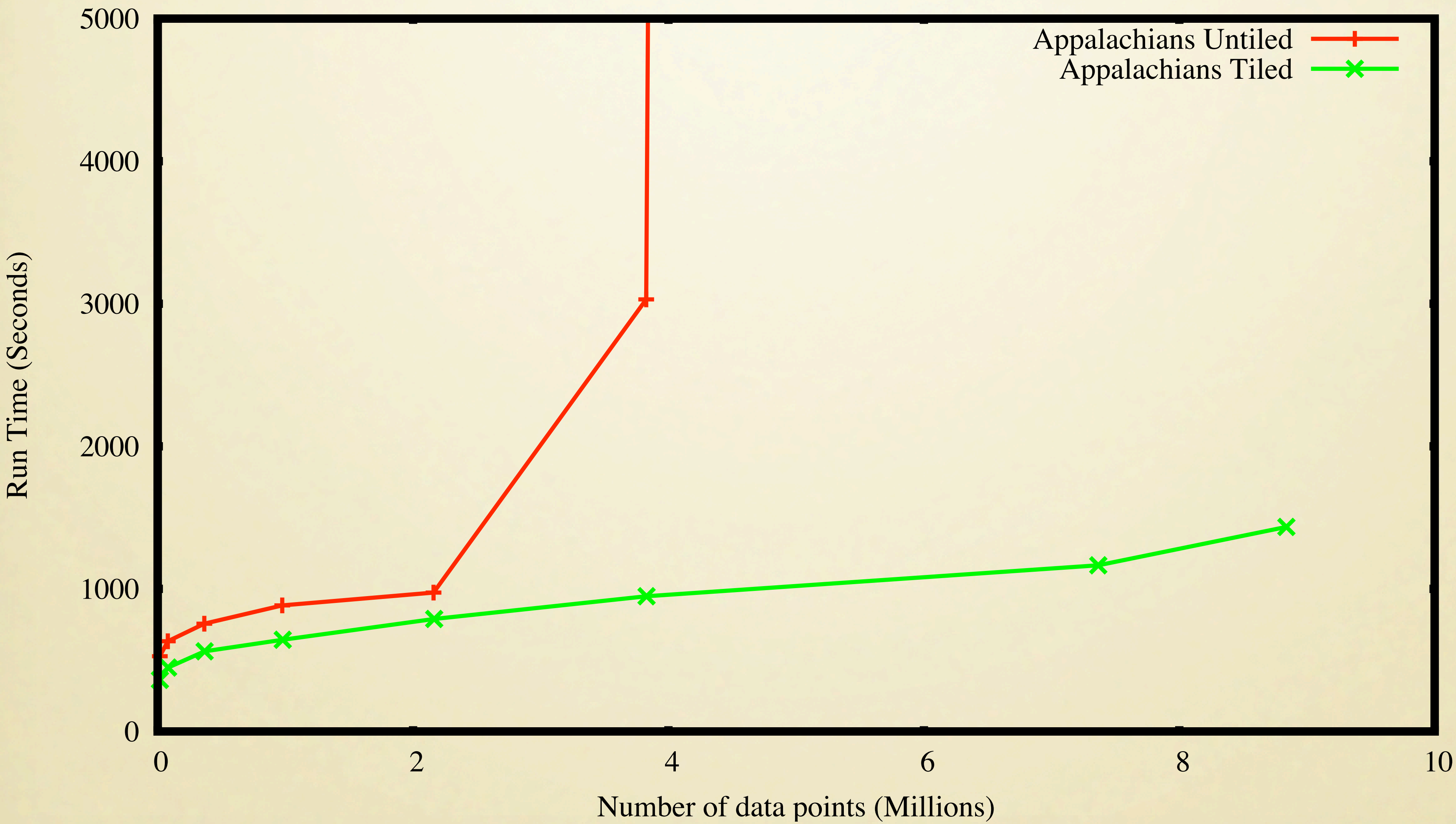
- Apple Dual Processor G5
- 2.5 GHZ CPU
- 1 GB RAM
- Data Sets from 1.6 million to 122 million points

Tiled vs Untiled Runtime Comparison

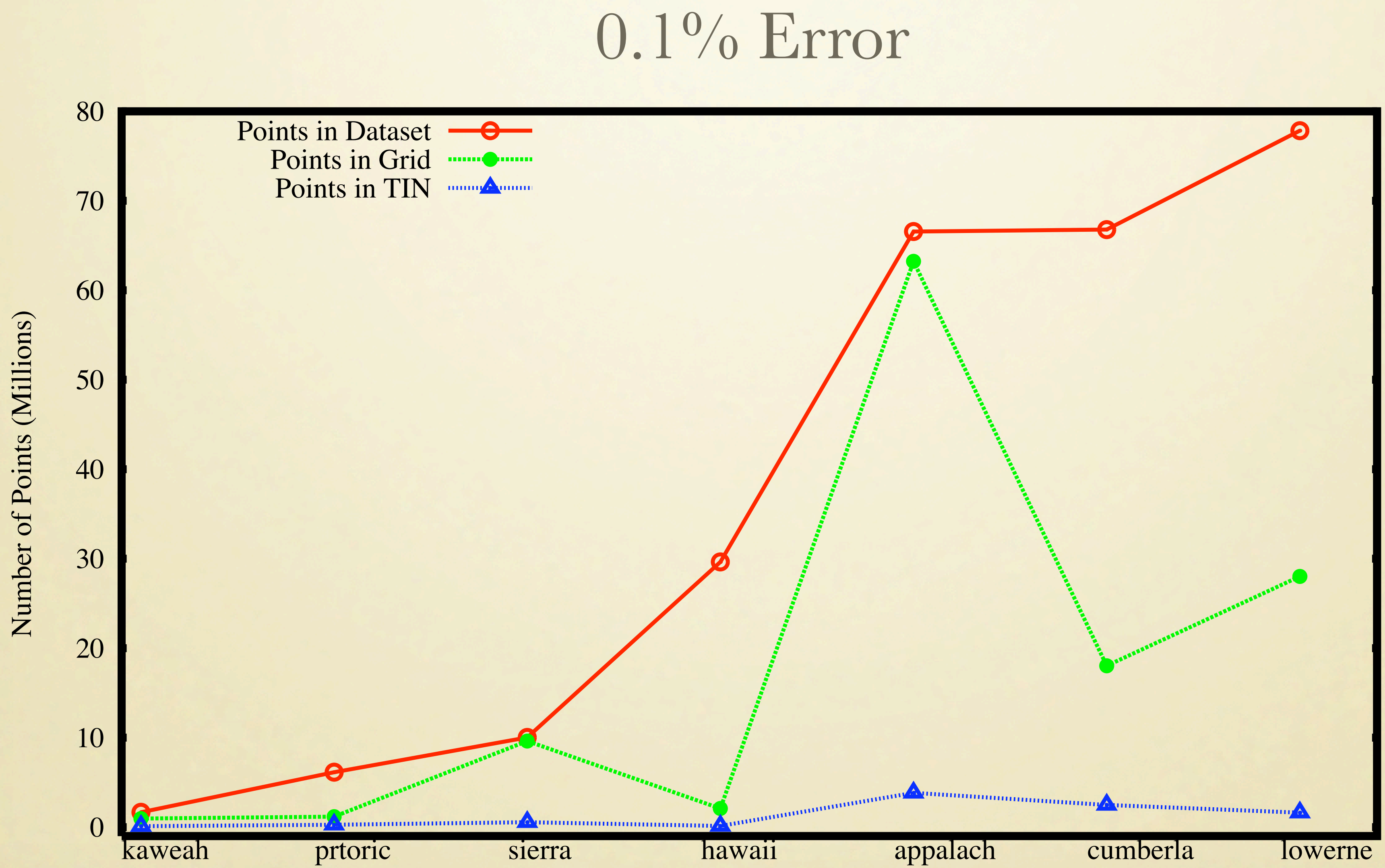


Titled vs Untiled Runtime Comparison

Tiled vs Comparison On Appalachians



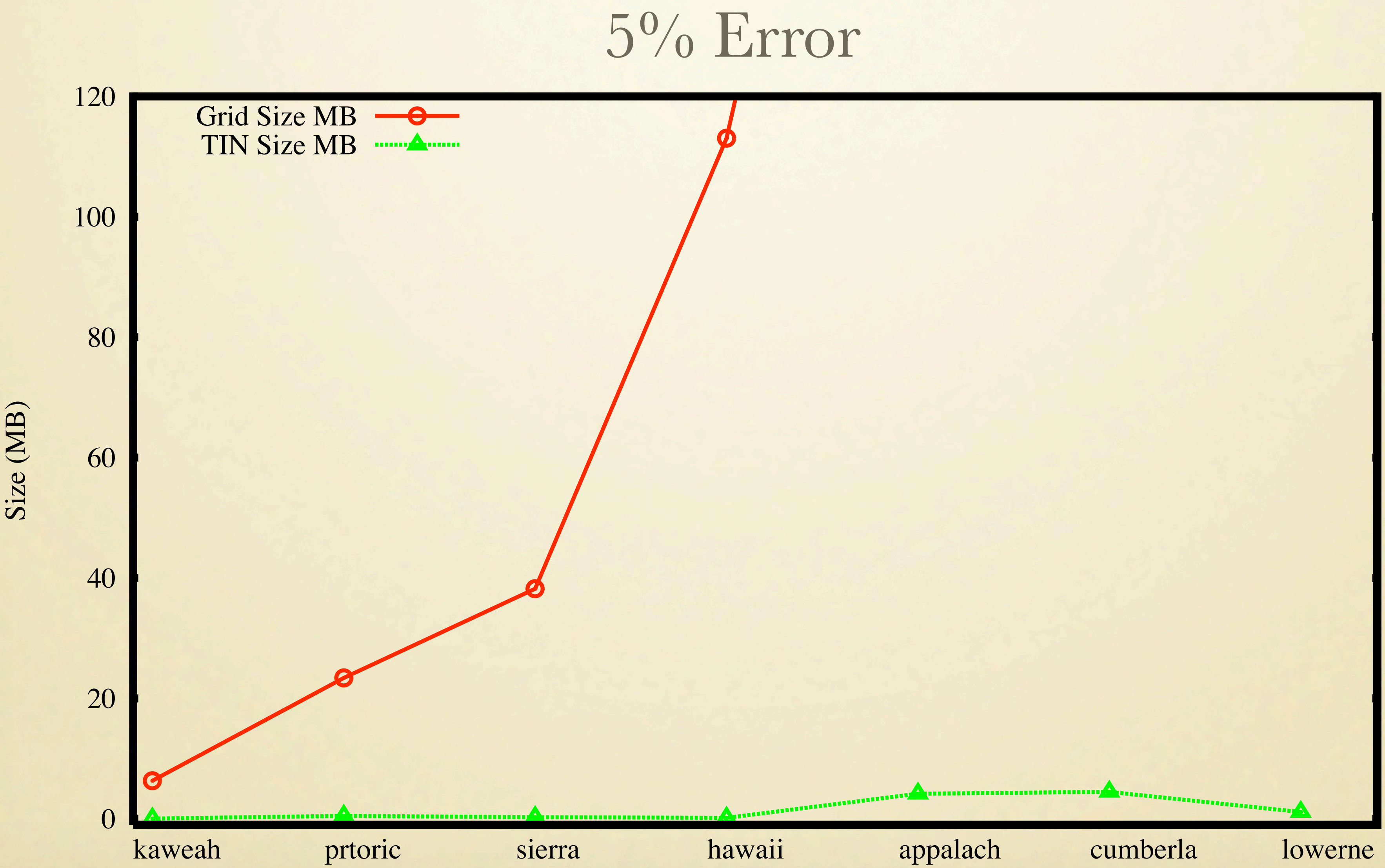
Grid vs TIN Point Comparison



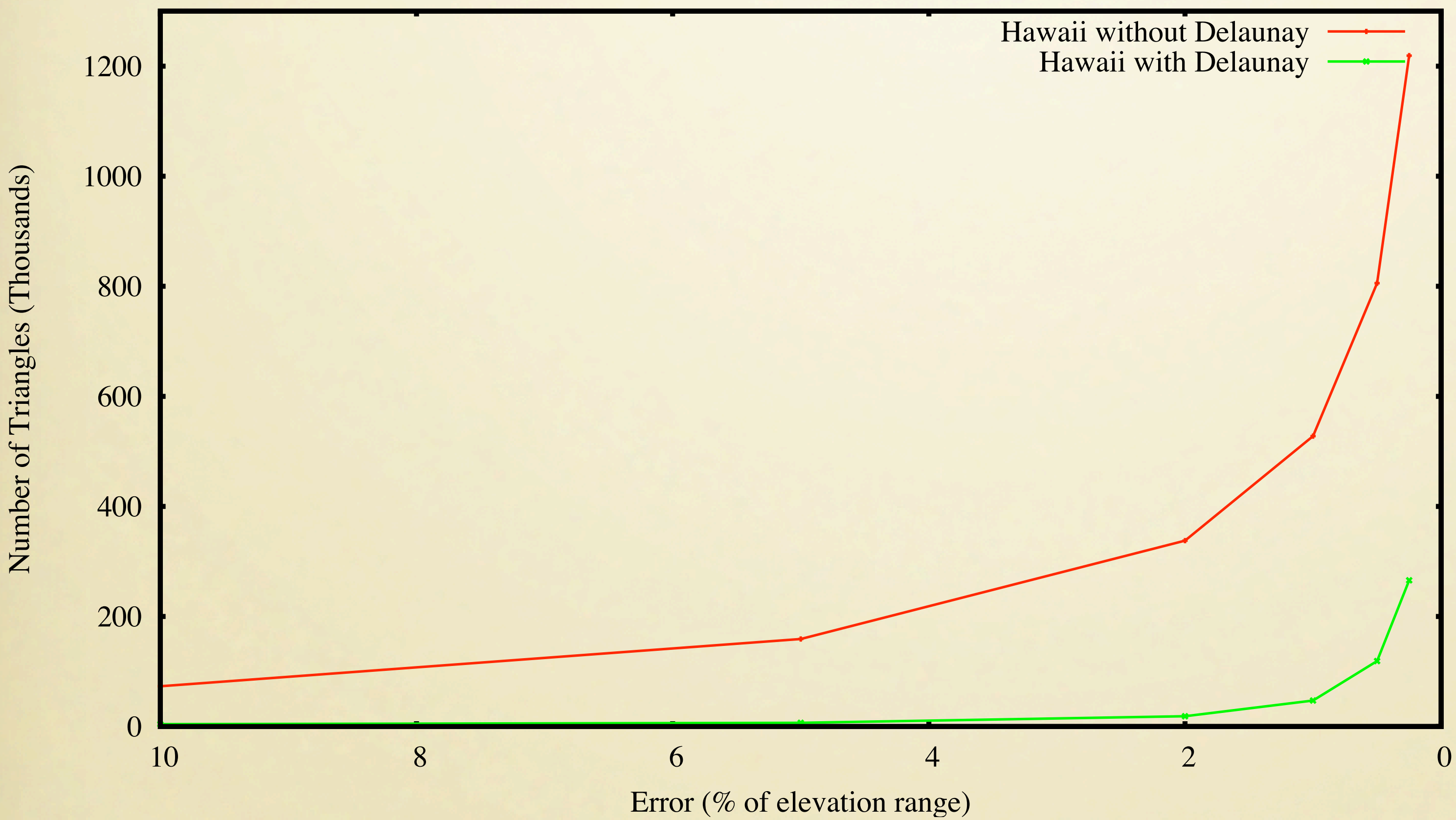
Grid vs TIN Size Comparison

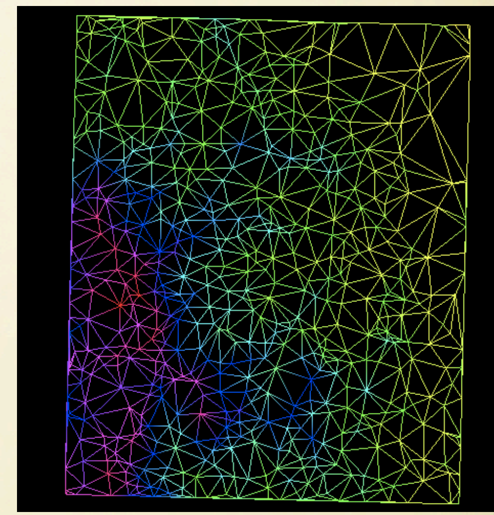
Grid vs TIN Size Comparison

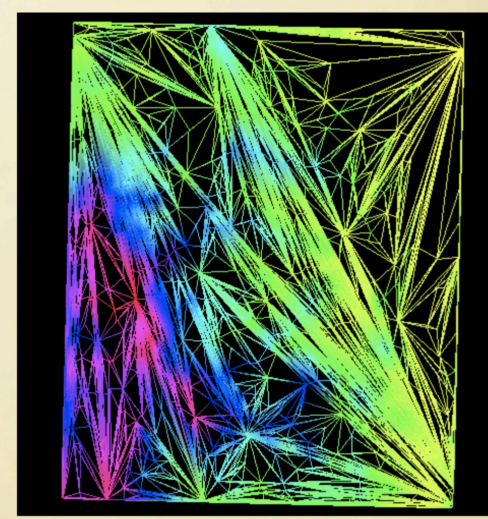
Grid vs TIN Size Comparison



Effects of Delaunay On Number of Triangles







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