Gridding vs Simplification in Extracting Performance from High Density Data

Simon Greener,
The SpatialDB Advisor

Abstract

- The storage of high density data is becoming more commonplace due to the use of GPS and other high-resolution data capture devices.
- The storage of this data is straightforward for Oracle practitioners, but the myriad uses of data can cause significant performance problems that must be, and can be, addressed using a range of techniques.
- This presentation outlines the use of simplification and tiling for improving the performance of selection, visualisation and geo-processing operations.
- Provides recommendations as to choice and methods of implementation.

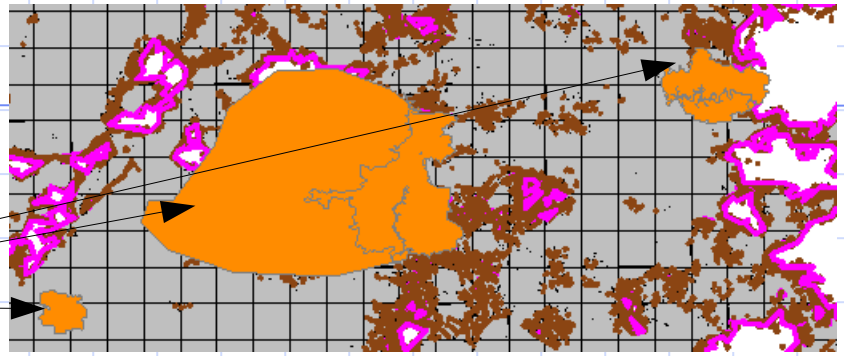
Business Situation

- Data:
  - US cellular mobile phone coverage data;
  - Vector data generated from large (national cellular coverage) raster dataset (0.0025 degree pixel size) using various dB (e.g., -98) levels;
  - Original dataset has about 300,000 polygons.
- Use:
  - Visualization at various levels from national down to street level via a web application;
  - Trade Area polygon coverage used to clip cellular data and return a variety of tabular info (demographics) and cell coverage (area);
  - Trade Area is of random size roughly an aggregation of 10-20 US Zip code polygons.

Trade Area Clipping ...

- When used for Trade Area clipping:
  - Untreated data is unusable in real time to do clipping, etc. via Oracle through a web interface.
  - Target performance is sub-second;
    - Current performance is 8+ seconds when using SDO_GEOM.SDO_INTERSECTION().
- Note, some polygons in source data break Oracle Sdo_Ordinate_Array limit!
  - Needs to be handled;

Example "Trade Areas"



Thinning, Generalisation or Simplification (what's in a name?)

- Current approach performs thinning before loading into Oracle:
  - Note: SDO_GEOM.SIMPLIFY() cannot be used as thinning is not “topologically constrained”;
  - FME and Manifold GIS used to perform thinning;
    - 300,000 polygons reduced to around 10,000.
  - Simplified data checked with ArcGIS before loading;

After loading

- **SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT()** is run after loading;
  - Reports multiple self-intersecting vs self-crossing polygons!
  - SDO_UTIL.RECTIFY_GEOMETRY used to correct geometries.
    - Correction takes many days to run.

Problems

- Current thinning approach:
  - Raster data are not provided;
  - Provided vector data is densely coordinated;
  - Preprocessing times are long, narrowing update processing windows;
  - Generates Oracle validation errors;
  - Error correction takes days to run.

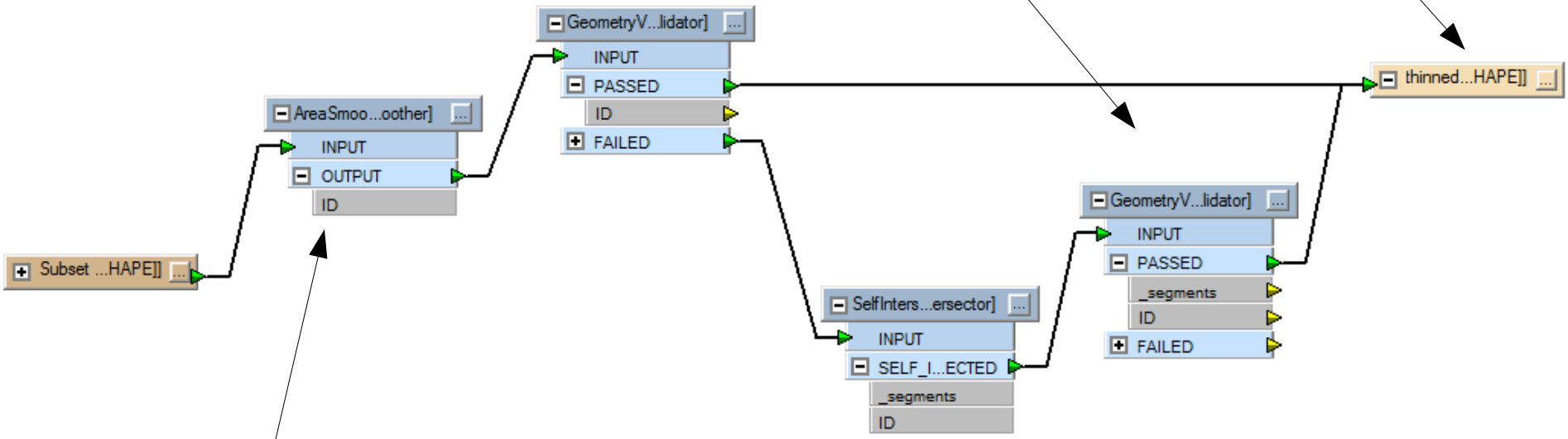
Brief...

- Brief of consultancy is to look at:
  1. Methods for ensuring valid, thinned, data using FME and Manifold GIS;
  2. The nature of the errors being reported by Oracle and their effect on processing;
  3. Alternate methods for providing the desired performance for Trade Area analysis.

FME Process

Validators added to ensure loaded data is correct.

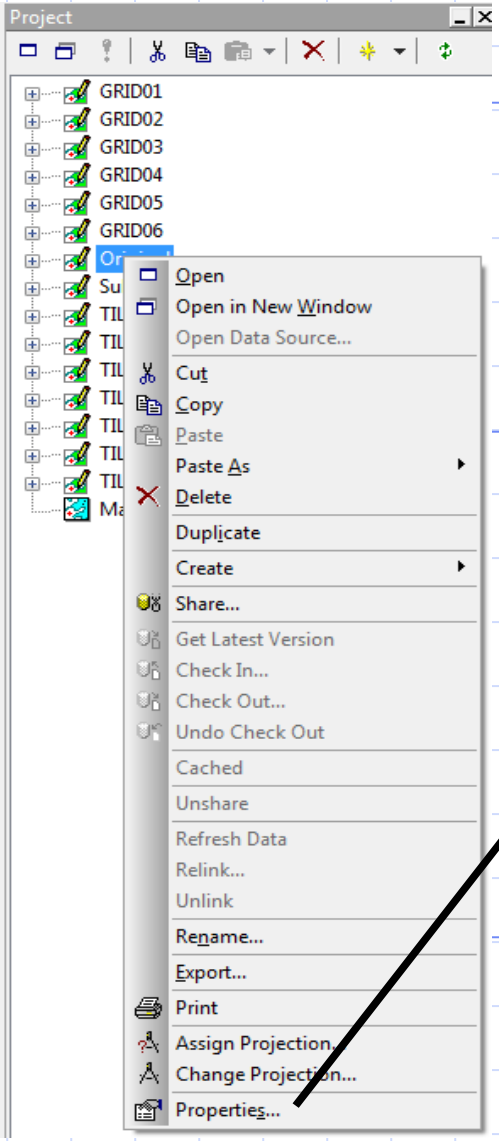
Data actually goes to Oracle not a shapefile



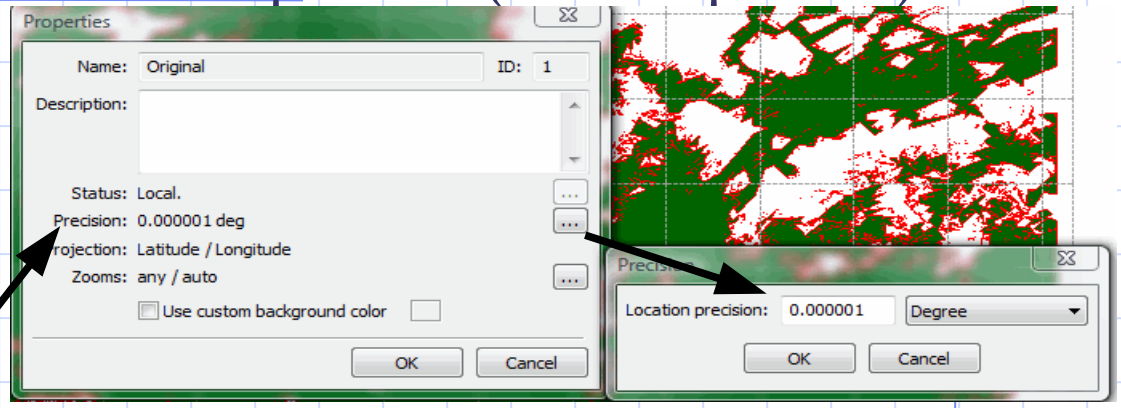
In the case of adjacent polygons, collinear portions of their boundaries will be smoothed together.

Self-Intersection trick added to deal with potential 13349 errors

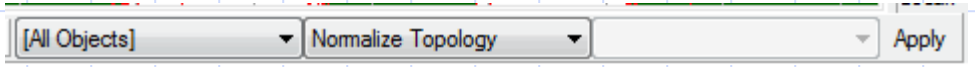
Manifold GIS



- Manifold GIS simplification is done by:
  1. Set Precision of Drawing Component (via Properties)

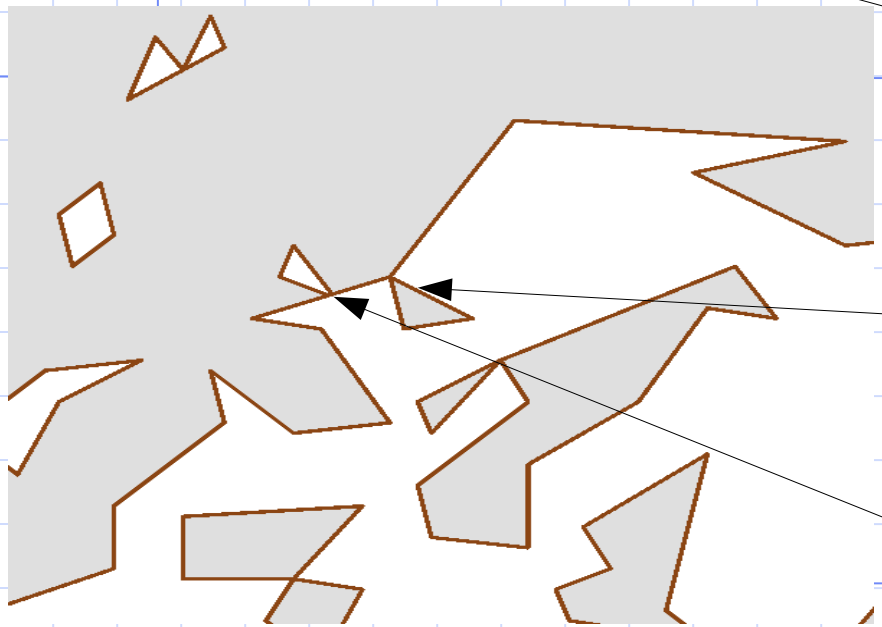


  2. Use "Normalize Topology" transform and not Drawing>Simplify

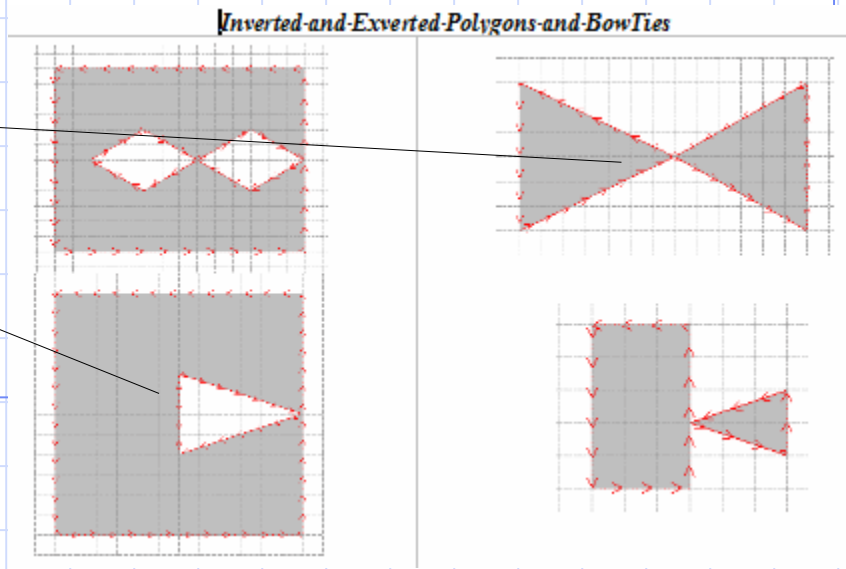


Simplification Results

- Oracle validation after simplification show ORA-13349 errors;
- All are Inverted/Exverted polygons:



ORA-13349: polygon boundary crosses itself
Cause: The boundary of a polygon intersects itself.
Action: Correct the geometric definition of the object.



- Manifold GIS does not produce Exverted polygons but FME does.

ORA-13349

- OGC/SQLMM standards do not allow exverted/inverted polygons yet most GIS allow one or both of them;
  - Exverted polygons, once corrected by Oracle, become separate polygons;
    - But end up with many more polygons for analysis;
  - Inverted polygons, once corrected by Oracle, become holes (2003) within polygon outer shell (1003).
  - All then pass Oracle validation.

Are these a problem?

- Given the observed slowness in correcting ORA-13349, and experience with GIS data, we can ask the questions:
  - "Are these a problem?"
  - "Do we need to fix them?"
- Inverted/Exverted polygons:
  - Are correctly indexed and happily geo-processed (SDO_GEOM.SDO_INTERSECTION etc) by Oracle;
  - Will be written, read and displayed by most GIS vendor products;
  - Do not appear to impair Oracle performance:
    - There is a school of thought that invalid data in an indexed sdo_geometry column impairs performance;
    - No empirical studies show this to be the case.

Inverted/Exverted Summary

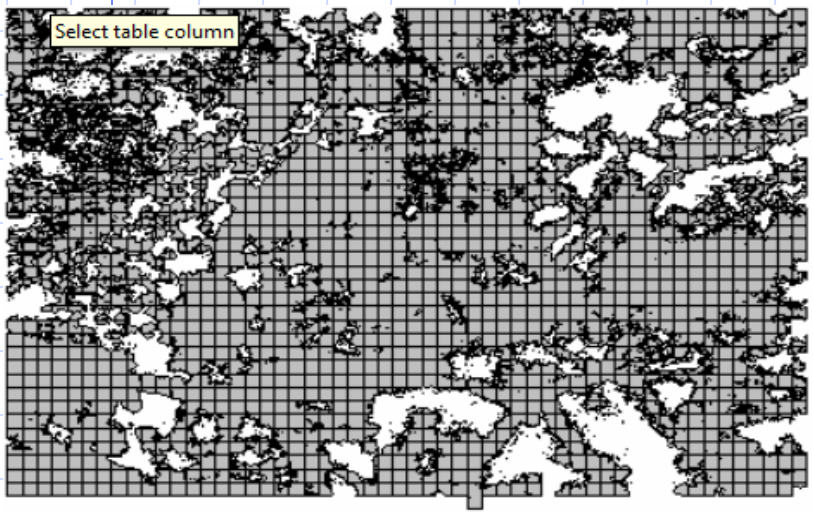
- In short:
  - During consultancy, Oracle correction of "invalid" geometries was skipped because:
    - No statistically observable performance degradation was observed in Trade Area geo-processing;
    - Three GIS products all "passed" the data before loading;
    - Along with Oracle itself, all GIS products could consume data after loading into Oracle Spatial (with no SDO_UTIL.RECTIFY_GEOMETRY processing);
    - Significant end-to-end processing improvements in the simplification process were extracted.

Simplification by Rasterisation

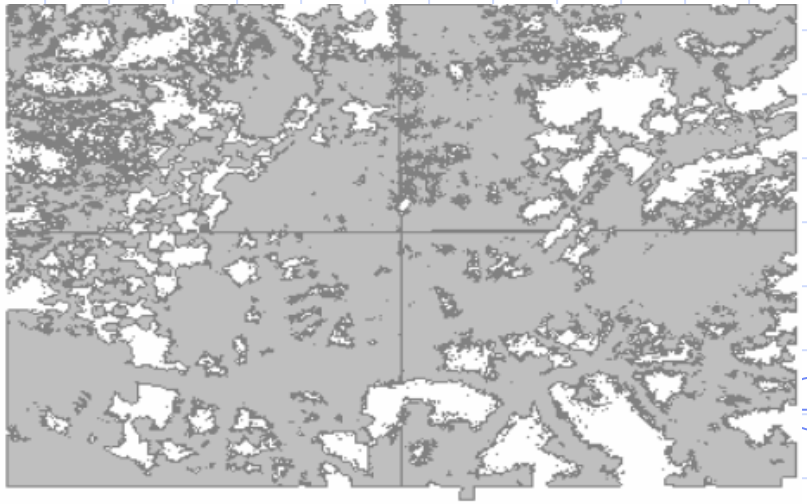
- Traditional vector simplification requires:
  - Choice of best "simplification" value via costly trial-and-error processing;
  - Costly "topologically constrained" processing;
  - High chance of creating invalid geometries.
- An alternate approach was tried that used rasterisation as the method for simplification.
  - Choice of "simplification" value easier here as data was, originally, from a 0.0025 degree raster (so smallest pixel size is determined).
  - Sadly, original raster is not available so derivative vector product much be "reverse rasterised".

Rasterisation Steps

- All processing was done in Manifold GIS as FME processing is more complicated and slower.
  - Original shapefile (for -98db level) was loaded into Manifold GIS and the grid lines removed.



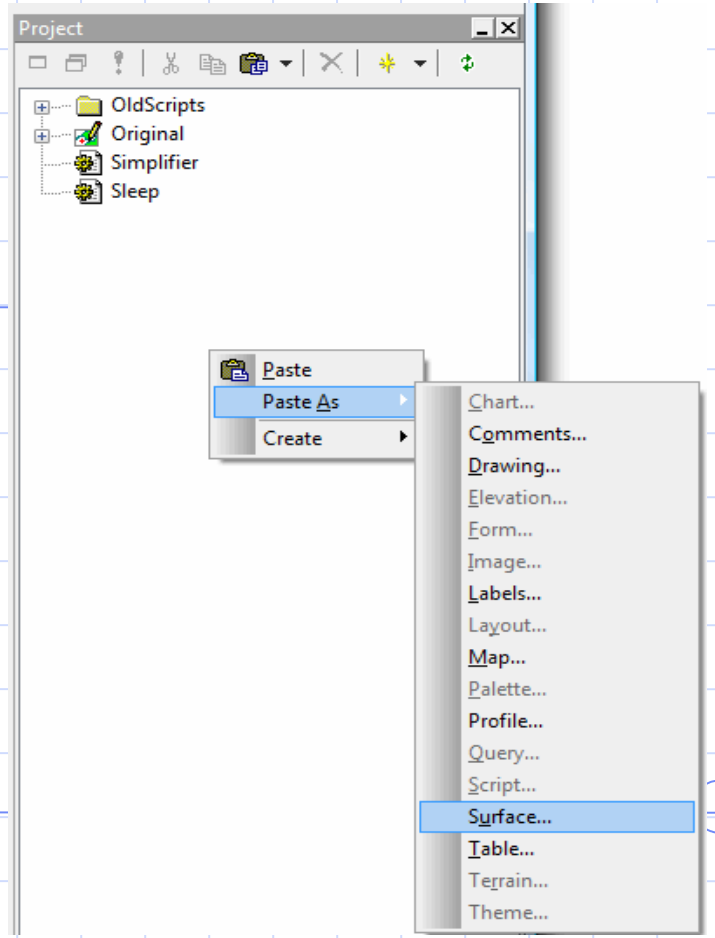
Original·Data·Showing·0.2·Degree·Grids



Dissolved·and·Loaded·Original

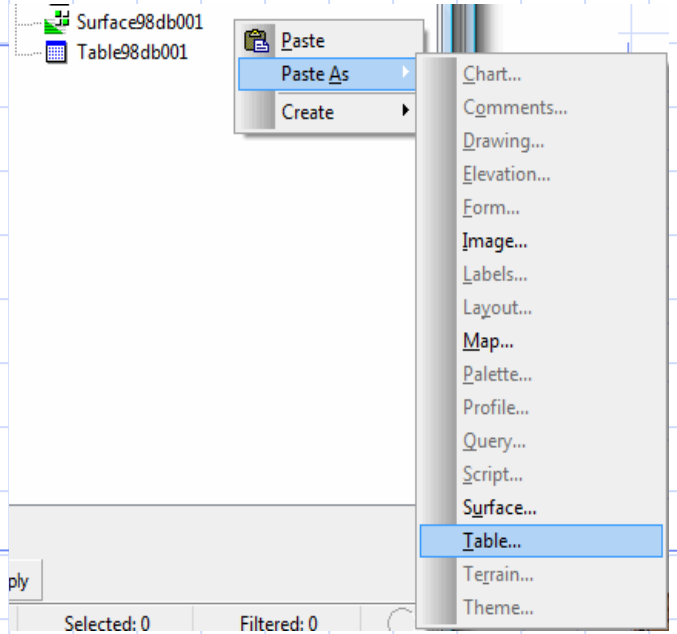
Rasterisation Steps: 2

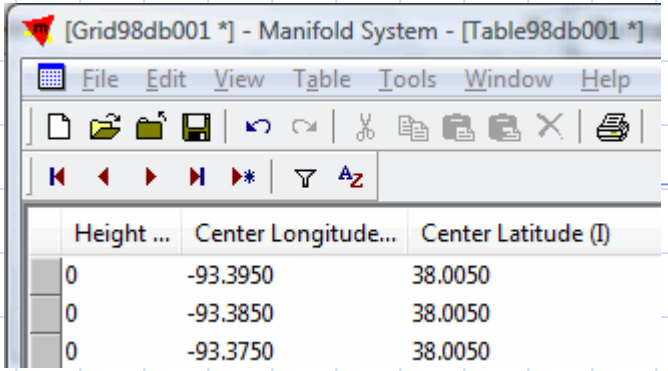
- Vector drawings were then created by rasterisation as Manifold GIS Surfaces (Copy Drawing>Paste As Surface) using different pixel (or grid) sizes.
- The resultant rasters contains only 2 values: 0 (no data) and -98 (the DB value).



Rasterisation Steps: 3

- The surface was then converted to a Table via:
  - Copy <surface> / Paste As Table
  - Only three "intrinsic" attributes pasted:
    - Height (I)
    - Center Longitude (I)
    - Center Latitude (I)





Rasterisation Steps: 4

- A vector drawing with one attribute, Height (I), is created.
- Drawing is then populated with individual square polygons representing each pixel:

```
sSQLStmt = "INSERT INTO ["&sComponent&"] ([DB],[Geom (I)])" & vbCrLf
sSQLStmt = sSQLStmt & "SELECT [Height (I)]," & vbCrLf
sSQLStmt = sSQLStmt & "       AssignCoordSys(" & vbCrLf
sSQLStmt = sSQLStmt & "           CGeom(CGeomWKB(" & vbCrLf
sSQLStmt = sSQLStmt & "                     ""POLYGON((""&CSTR([Center Longitude (I)] - " & iHalfPixel & ")&"" ""&"  & vbCrLf
sSQLStmt = sSQLStmt & "                                CSTR([Center Latitude (I)]  - " & iHalfPixel & ")&"", ""&" & vbCrLf
sSQLStmt = sSQLStmt & "                                CSTR([Center Longitude (I)] + " & iHalfPixel & ")&"" ""&"  & vbCrLf
sSQLStmt = sSQLStmt & "                                CSTR([Center Latitude (I)]  - " & iHalfPixel & ")&"", ""&" & vbCrLf
sSQLStmt = sSQLStmt & "                                CSTR([Center Longitude (I)] + " & iHalfPixel & ")&"" ""&"  & vbCrLf
sSQLStmt = sSQLStmt & "                                CSTR([Center Latitude (I)]  + " & iHalfPixel & ")&"", ""&" & vbCrLf
sSQLStmt = sSQLStmt & "                                CSTR([Center Longitude (I)] - " & iHalfPixel & ")&"" ""&"  & vbCrLf
sSQLStmt = sSQLStmt & "                                CSTR([Center Latitude (I)]  + " & iHalfPixel & ")&"", ""&" & vbCrLf
sSQLStmt = sSQLStmt & "                                CSTR([Center Longitude (I)] - " & iHalfPixel & ")&"" ""&"  & vbCrLf
sSQLStmt = sSQLStmt & "                                CSTR([Center Latitude (I)]  - " & iHalfPixel & ")&""))""")),"  & vbCrLf
sSQLStmt = sSQLStmt & "           CoordSys(""" & sComponent & """ AS Component))" & vbCrLf
sSQLStmt = sSQLStmt & "  FROM ["&sTableComponent&"]"
CreateGridQuery.Text = sSQLStmt
CreateGridQuery.Run()
```

Manifold GIS's Spatial SQL is awesome (a natural for an Oracle Spatial practitioner)!

Rasterisation Steps: 5

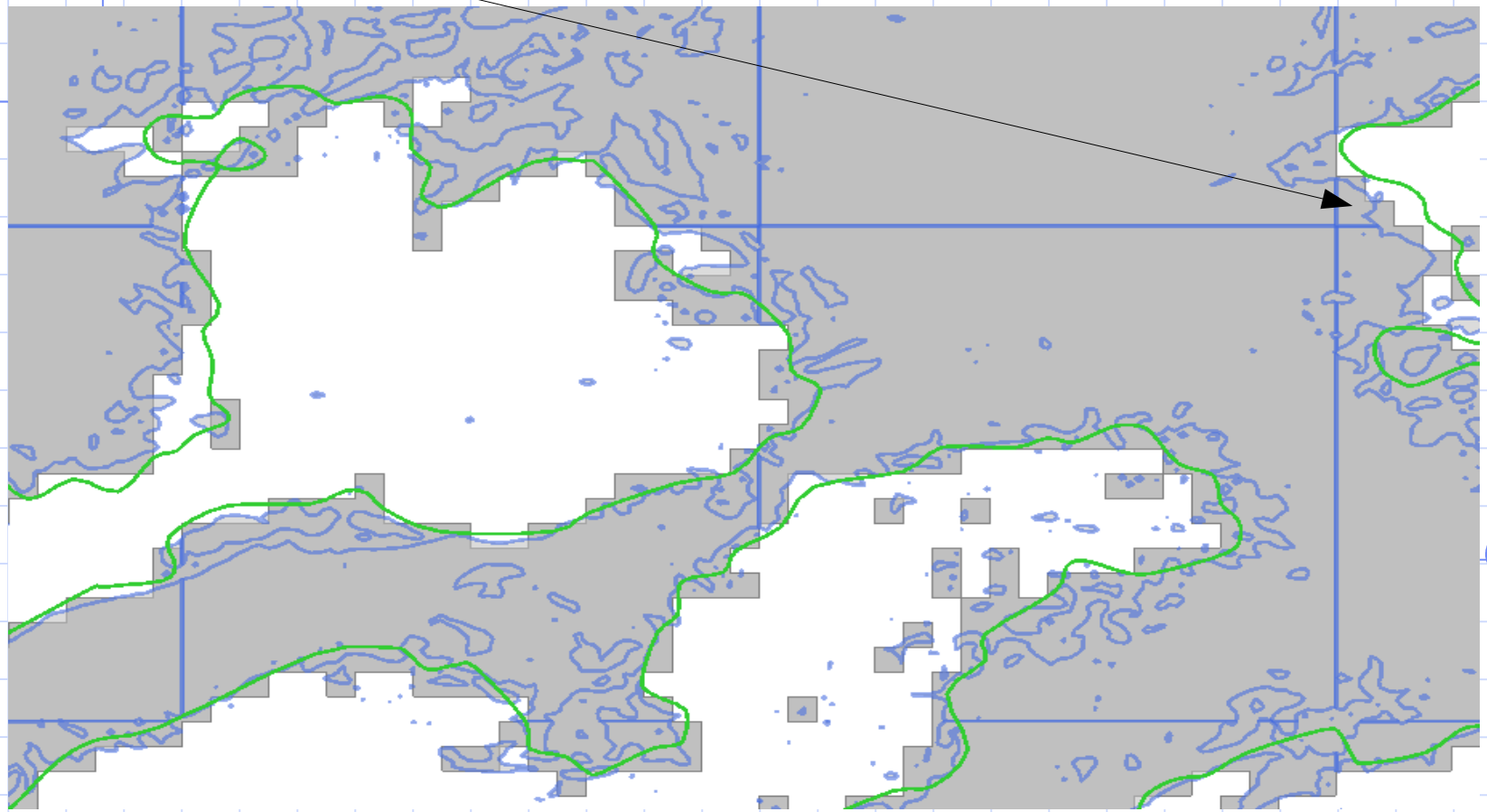
- All pixel polygons are then unioned together to create two output polygons (“GROUP BY Height (I)” with Height (I) = 0; one for -98) and written to a new drawing:

```
Sub UnionTilesQuery( sComponent, sDestComponent, LL, UR )
    Dim SelectTilesQuery
    Set SelectTilesQuery = Application.ActiveDocument.NewQuery("SelectTilesQuery")
    Dim sWKT
    sWKT = "POLYGON(("&LL.X&" "&LL.Y&", "&LL.X&" "&UR.Y&", "&UR.X&" "&UR.Y&", "&UR.X&" "&LL.Y&", "&LL.X&" "&LL.Y&"))"
    Dim sSQLStmt
    sSQLStmt = "INSERT INTO ["&sDestComponent&"] ([DB],[Geom (I)])" & vbCrLf
    sSQLStmt = sSQLStmt & "SELECT [" & sComponent & "].[DB],UnionALL([" & sComponent & "].[ID])" & vbCrLf
    sSQLStmt = sSQLStmt & "  FROM ["&sComponent&"]" & vbCrLf
    sSQLStmt = sSQLStmt & " WHERE Contains(AssignCoordSys(CGeom(CGeomWKB(""" & sWKT & """)),CoordSys("""&sComponent&""" As COMPONENT)),["&sComponent&"].[ID])" & vbCrLf
    sSQLStmt = sSQLStmt & " GROUP BY [" & sComponent & "].[DB]"
    SelectTilesQuery.Text = sSQLStmt
    SelectTilesQuery.Run()
    Set SelectTilesQuery = Nothing
    DeleteComponent "SelectTilesQuery"
End Sub
```

- “Normalise Topology” is then run on the final Drawing;

Rasterisation: Result

- Example of vector output using 0.01 degree intermediate pixel (blue is original, green is vector simplified data).
- Note "staircase effect" in final rasterised vector boundary.



Rasterisation: Summary

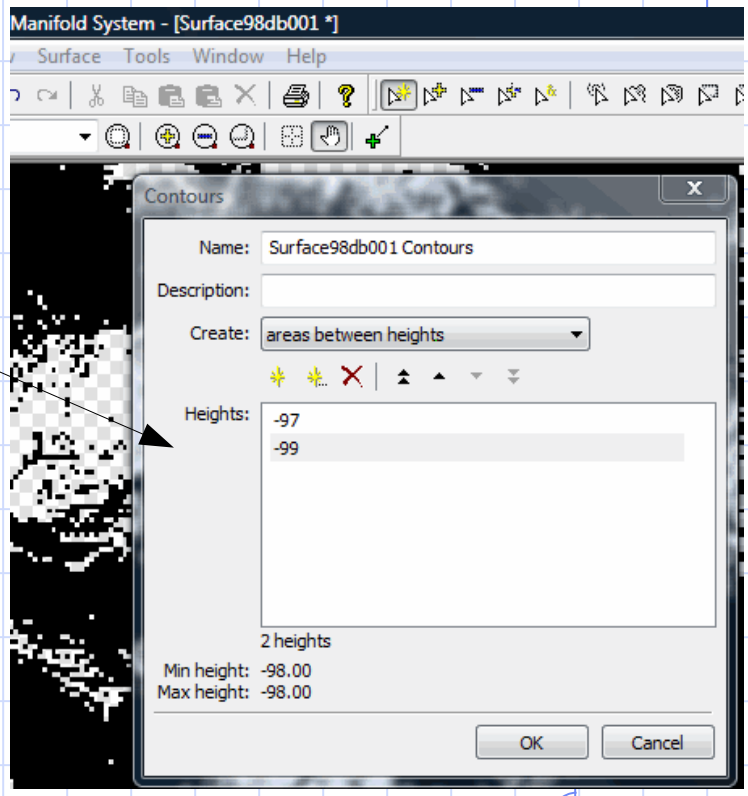
- While steps look complicated, the processing is, actually, very simple and fast;
  - All processing was scripted (including Copy/Paste) making it easy to try different pixel values;
- Benefits:
  - Speed of pre-processing;
  - No issues with ORA-13349 errors;
  - Pixel size choice easy as original data was raster;
- Downside:
  - Still requires pre-processing of supplied data before loading into Oracle;
  - Not for visualisation!

Vectorisation

- There is a variation on the Rasterisation approach that is:
  - Simpler;
  - Provides more aesthetically pleasing output.
  - I call it Vectorisation.

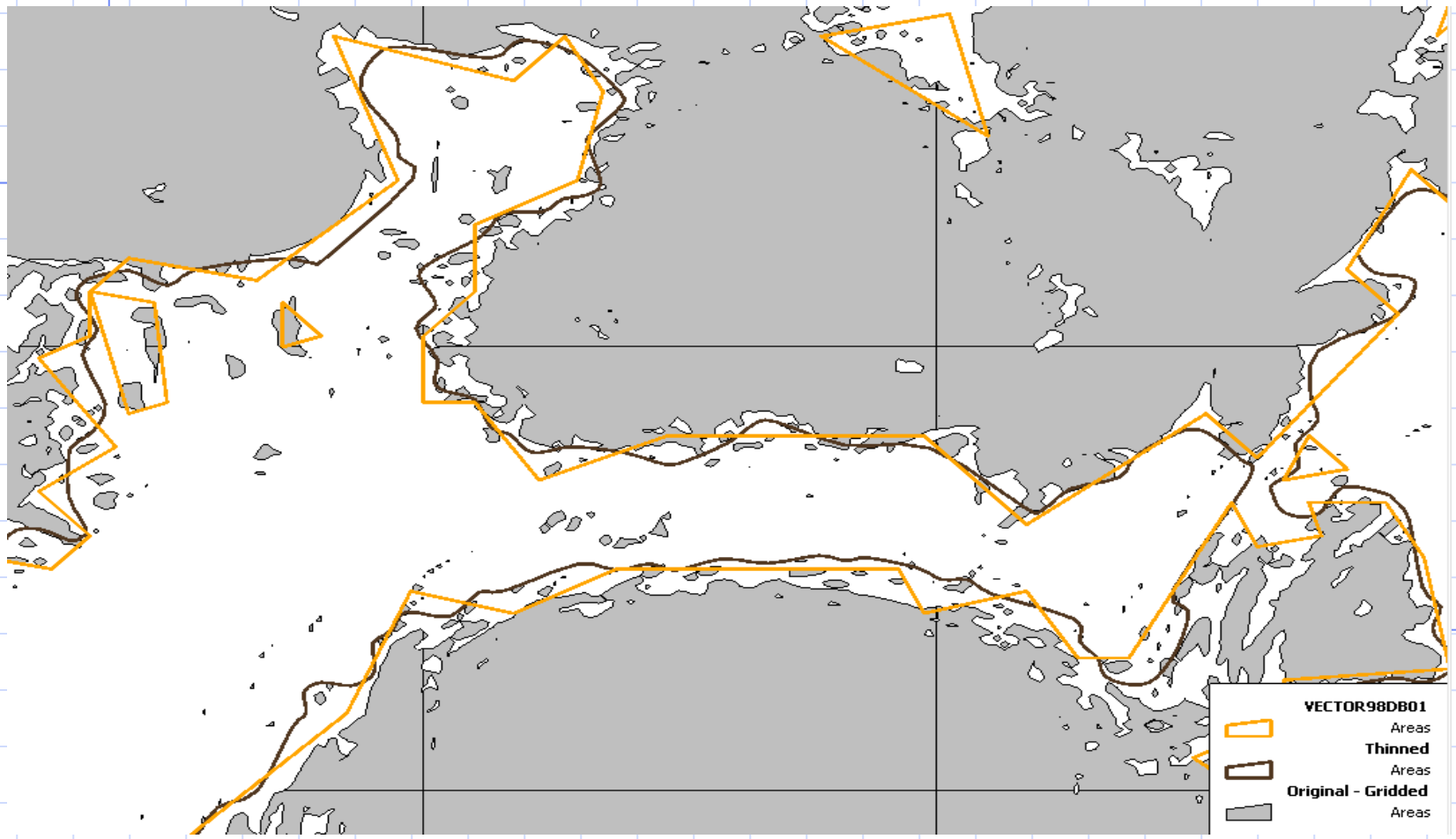
Vectorisation Steps

- Follow Rasterisation Steps up to and including Step 2.
- The raster is then contoured (Surface>Contours) to produce a polygon vector data set.
- Finally, "Normalize Topology" (usually half the pixel size) is used to "explode" the single polygon produced by contouring, into its composite polygons *and* to execute some simplification (thinning) of the vector data.



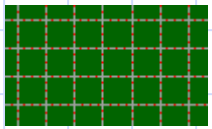
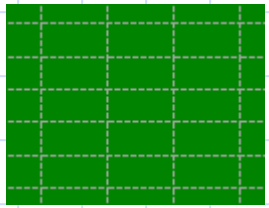
Vectorisation: Result

- Note no harsh "staircase effect" in vectorised output:



Tiling

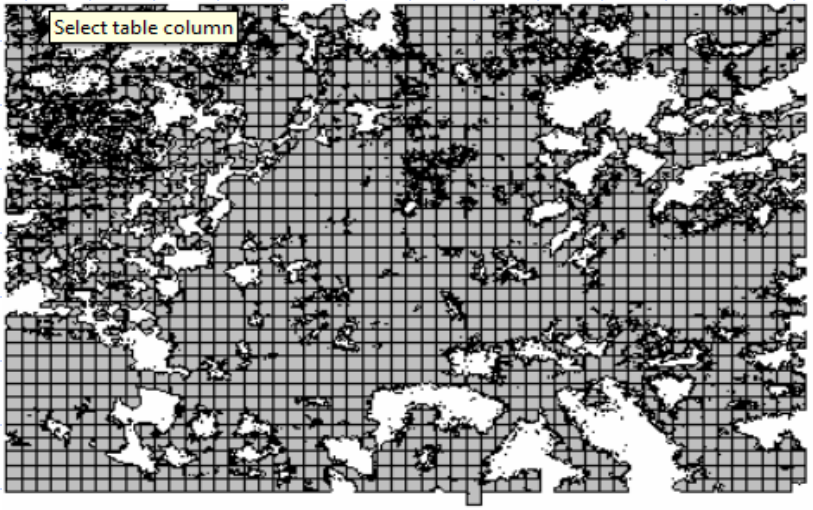
- One, final, alternative, is available to improve performance: Tiling;
- Tiling is the simple clipping of a vector dataset with a tessellated surface (grid).
  - Tessellated surface can be:
    - Square grid;
    - Rectangular grid;
    - Hexagonal grid;

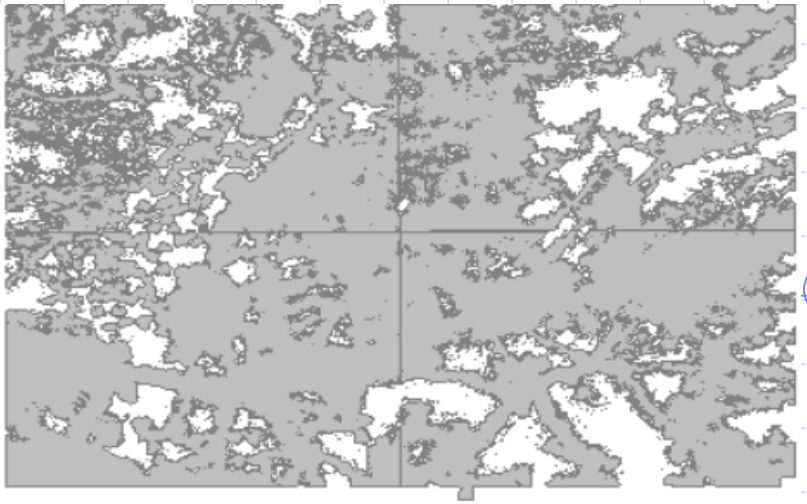

  - And can be regular (all tiles same size) or irregular (as in a quad tree).

Why Tiling?

- Original data included 0.2 degree tiles;
- Oracle Sdo_Ordinate_Array limit requires large polygons to be broken up via tiling (4 tiles) anyway;
- Technique used successfully at other sites to improve geo-processing performance;
- No simplification problems;
- Simpler pre-processing;



Original·Data·Showing·0.2·Degree·Grids



Dissolved·and·Loaded·Original

Tiling Methodology

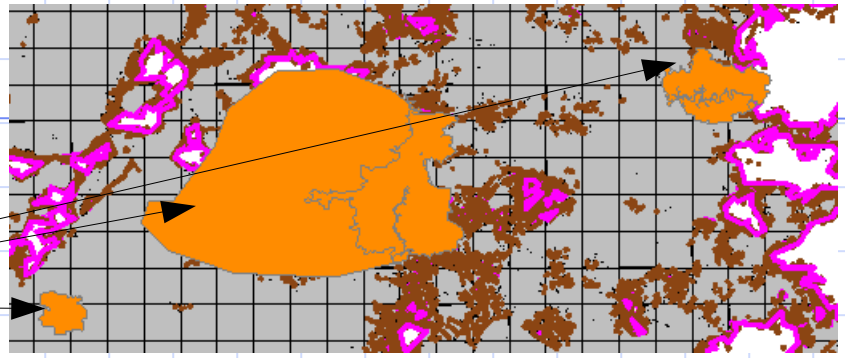
- The technique starts with the original vector data with the (0.2 Degree) grid lines removed.
- Then vector grids containing non-overlapping, square polygons are generated.
  - A number of grid sizes are used: 0.1, 0.2, 0.3, 0.4, 0.5 and 0.6.
- Then the original data is overlaid ("cookie cut") with each grid and saved as a new vector table in Oracle;
  - eg TILE98DB01 is the overlay of the original data and the 0.1 vector grid.

Comparison of Methods: Trade Area Analysis

- Trade Area analysis conducted against only Vectorised, Tiled (different pixel sizes), Original and Thinned (supplied) data using 4 different Trade Areas:
  - SIMPLE, CENSUS 1, CENSUS 2, CENSUS 2

```
SELECT /*+ORDERED*/
       SUM(sdo_geom.sdo_area(
                sdo_geom.sdo_intersection(a.geom, ta.geom, 0.05),0.05))
         / 10000 As area_in_ha
  FROM TradeArea ta,
       <table_name> a
 WHERE ta.detail = ':1'
   AND sdo_anyinteract(a.geom,ta.geom) = 'TRUE'
```

Example "Trade Areas"



Comparative Results

Note resulting area from query compared to original data

- Results (for "Simple" Trade Area) are:

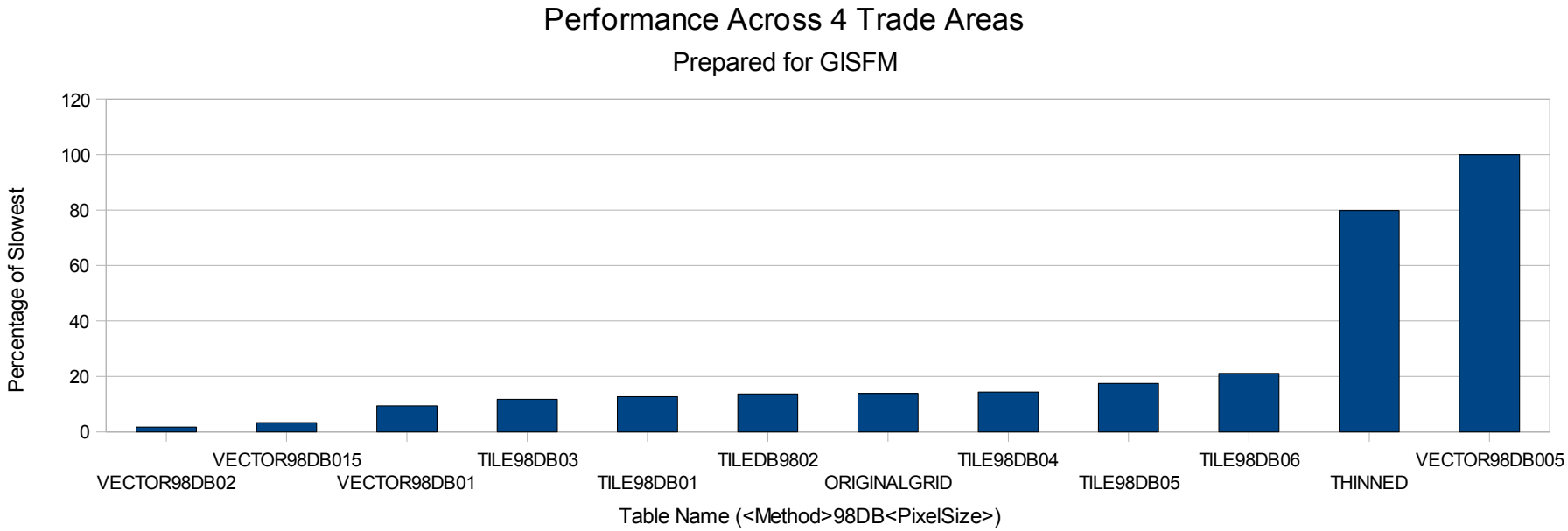
TradeArea	Table	Area	Percentage Area	Time
SIMPLE	VECTOR98DB02	1185449.48	108.34	0.49
SIMPLE	VECTOR98DB015	1178365.29	107.69	0.8
SIMPLE	TILE98DB01	1094134.23	99.99	1.95
SIMPLE	VECTOR98DB01	1167646.92	106.71	2.04
SIMPLE	TILEDB9802	1094132.54	99.99	2.34
SIMPLE	ORIGINALGRID	1094136.55	99.99	2.45
SIMPLE	TILE98DB03	1094145.69	100	2.65
SIMPLE	TILE98DB04	1094134.96	99.99	3.7
SIMPLE	TILE98DB05	1094146.07	100	5.47
SIMPLE	TILE98DB06	1094148.12	100	7.03
SIMPLE	THINNED	1154580.54	105.52	17.05
SIMPLE	VECTOR98DB005	1145514.07	104.69	22.08
SIMPLE	VECTOR98DB0025	1123145.45	102.65	211.1
SIMPLE	ORIGINALORA	1094198.48	100	1541.01

- In almost all situations (vectorised simplification or tiling) performance significantly faster than the supplied thinned data.
- Original gridded (0.2 degree) data performed far better than expected (it is equivalent to TILE98DB02).

Target time to beat

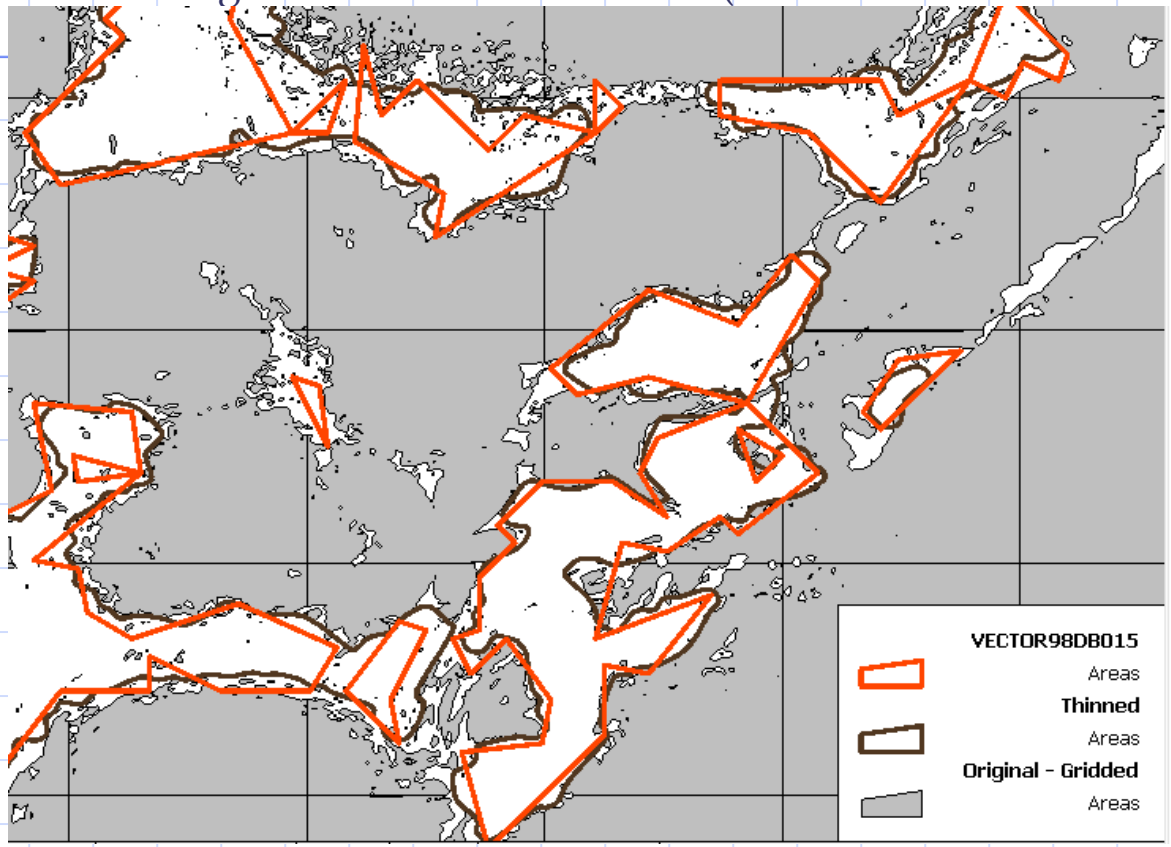
Results Charted

- Table is graphed via one simple chart in which two slowest representations of the original data have been removed: VECTOR98DB0025 and ORIGINALORA.
- The average of the four runs was taken and graphed as a percentage.



Results Discussed

- Clearly the three tables generated via vector-to-raster simplification were fastest.
- However, the size of each raster cell used to generate the three tables – 0.02, 0.015 and 0.01 Degrees – results in significant variation from the actual cellular coverage as can be seen in below (and in % Area differences).



Results Discussed 2

- The tiled original vector data performed incredibly well.
  - TILE98DB03 performance roughly on par with VECTOR98DB01.
  - While TILE98DB03 includes a small amount of simplification (99.99% area difference) but this simplification could be removed without much “pain” being experienced (see performance of ORIGINALGRID).
  - In short, the tiled data performed to a more than acceptable level.
    - Potential for more improvements with even smaller tiles eg 0.05 degree pixel size.
    - Potential mixing of tiles and vectorisation;

Conclusion

- There are more simplification methods than one might think:
  - Simplification;
  - Rasterisation;
  - Vectorisation;
- All methods of simplification involve some data loss;
  - What is acceptable depends on use of data;
- However, simple tiling of original data can provide best result:
  - Very simple preprocessing;
  - No loss of data precision;
  - Potential multiple use of data;
  - Some “light” simplification could also be applied.

Thanks and Questions...

- My thanks to Bill Ten Broek of GISFM in the USA for the opportunity to work on this problem.

- Any questions?