

Disk-based gridding for large data sets

Steven Zoraster, Austin Subsurface Modeling, Inc.

Summary

As computer speed and memory increases the amount of data to be processed grows even faster, requiring more sophisticated algorithms to run on the new hardware. Terrain and subsurface formation modeling have experienced this data explosion, along with a demand for larger output models. In this paper I present a grid-based surface modeling algorithm that makes little demand on computer RAM and uses only simple and local mathematical operations while creating a mathematically C^2 continuous solution. The algorithm is competitive with other gridding algorithms on small or medium size data sets. For large problems, problems with tens of millions of input data values and output grids with billions of nodes, this algorithm produces a solution while many other algorithms are just starting or have already crashed.

Introduction

The new gridding algorithm presented here, named Cyclical B-Spline Gridding (CBSG), is a disk-based, mathematical algorithm for gridding large data sets. A 3D view of a terrain model created using CBSG from bare-earth LIDAR collected over the city of Baltimore is shown in Figure 1. Approximately $\frac{1}{2}$ the nodes in this model are the result of interpolation across multi-node gaps in the data. These results are typical of those produced this algorithm

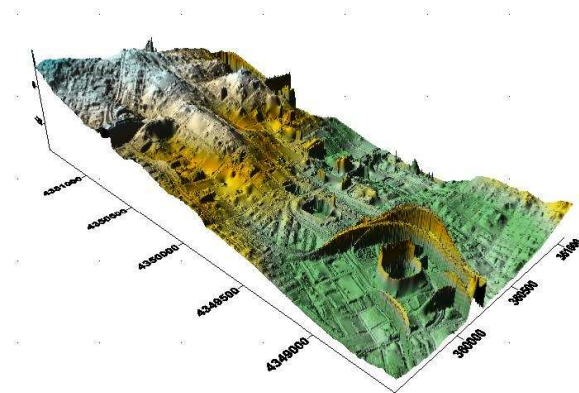


Figure 1. 3D view of a model of part of downtown Baltimore created by Cyclical B-Spline Gridding.

LIDAR (Light Distance and Range) data is the best example of a recently available data type that increases the amounts of data by orders of magnitudes. ***

CBSG is an extension of an in-core, iterative, B-Spline algorithm. (Lee, Wolberg and Shin, 1997) My implementation follows the workflow of the original algorithm, and retains the accuracy of that algorithm. Like that algorithm CBSG implements a one-directional, multilevel solution, working with coarse grids during early iterations and increasingly finer resolution grids during later iterations. The major difference is that CBSG requires minimal RAM, demanding at most space to hold 20 grid columns in core at any point during execution. Because of the conservative use of RAM, CBSG substantially increases: a) the size of data sets that can be modeled and; b) the size of the grids that can be produced.

The price paid for minimal use of RAM is reliance on disk-based storage of intermediate results. Of course, there is time penalty for this use of disk space. But this time penalty is worth paying in exchange for the ability to build large grids from large data sets with a robust mathematical model. In fact, CBSG is simple enough that even for smaller data sets and smaller grids it executes in less time than many other gridding algorithms. CPU and wall clock times for use of this algorithm for various data sets and grid sizes are shown in Table 1.

Disk-based gridding for large data sets

Control Points	Grid Nodes (millions)	CPU (minutes)	Wall Clock (minutes)
89,000	.3	.2	.3
250,000	4.6	.4	.5
250,000	18.5	.5	1
2,700,000	7.7	1.8	3.2
5,700,000	8.3	5.7	11
5,700,000	133.0	17	78
30,000,000	380.0	50	205
145,000,000	4,876	170	600

Table 1: Algorithm CPU and wall clock times

For practical purposes the primary constraint on using this algorithm is the amount of free hard disk space available. To my knowledge, alternative gridding algorithms that solve such large problems with minimal demand on RAM have not been described in detail in a public forum.

Previous Work

The general problem of converting measurements scattered over a 2D surface into a continuous (sampled) model is encountered in many scientific domains with wide variability in data density and data set sizes.

An assumption for all 2D gridding is that the data is sampled from an elevation function F , mapping $R^2 \rightarrow R^1$ and that the derived model approximates F to some degree of accuracy over R^2 . F may have a stochastic component. This problem has been studied for decades. It is possibly impossible, and probably useless, to describe all types of computer gridding algorithms and their individual variations. Reviews can be found in papers and books by Foley and Hagen (1994), Franke and Nielson (1991), Jones, Hamilton and Johnson (1986). A recent, general purpose algorithm for solving large gridding problems using matrix formulations and iterative techniques has been presented by Billings, Beatson, and Newsam (2002). Their algorithm works with many different mathematical models for surface modeling.

It is hard to get descriptions of algorithms that are reported to handle the massive size of data sets and DEM models created from them. This is, I think, because such algorithms have commercial value but are not protected by patents, so that authors are reluctant to provide complete algorithm descriptions. This certainly is true for algorithms provided by GIS companies, which tend to be conservative in revealing algorithm details in public.

A paper by Agarwal, Arge and Danner (2006) does provide a detailed explanation of their algorithm which emphasizes I/O efficiency. They start by using a quad tree (Hjaltason and H. Samet) to partition the data into non-overlapping sets indexed for efficient access. Next, for each segment q , they compute the set of points in q and all segments neighboring q . Finally, they interpolate DEM nodes inside each leaf segment independently using points within the segment and its neighboring ;rsg segments. The use of neighboring segments guarantees a level of DEM continuity across segment boundaries. They do interpolation by an implementation a spline algorithm with a tension parameter (Mitas and Mitasova, 1993). Agarwal, et al, speed up their algorithm by thinning points in each quad tree leaf segment so that it includes only points that are at least a user-specified distance e from all other points within the segment. They report building DEM with as many as 53 billion nodes from data stored in the quad tree with between 66 and 400 million points. On the largest

A tradeoff rarely addressed when evaluating gridding algorithms is between dependence on RAM and dependence on hard disk memory. In related domains, such as computational geometry, this tradeoff is often considered important. For example, sweep-line algorithms for finding intersections between members of a set of line segments are both memory efficient and a natural way to organize computations. For surface modeling, Fortune's (1989) sweep-line based, 2D triangulation is an algorithm that could be used for gridding that would make relatively light demands on RAM. Fortune's algorithm is known to be slower than divide and conquer triangulation algorithms. I do not know if it is being used anywhere for surface modeling instead of divide and conquer because of its small RAM footprint.

For CBSG, the criteria for algorithm selection was an ability to model large data sets and create large surface models in small amounts of RAM, without sacrificing mathematical properties such as accuracy and C^2 continuity. I wanted, at least in theory, an algorithm that could solve any size gridding problem. CBSG is the result. It is an extension of an in-core, hierarchical B-Spline gridding algorithm. Like that algorithm, it is fast and creates a C^2 mathematical surface model. By trading computational time for disk access, I implemented a gridding algorithm with the good qualities of the in-core algorithm with much reduced dependence on computer RAM. While developed for the petroleum industry, other fields of application include meteorology, oceanography, GIS, water quality modeling, engineering, chemistry, medical imaging, and physics.

Disk-based gridding for large data sets

Hierarchical B-Spline Gridding

In-RAM algorithm¹

Creating a grid-based surface model with B-Splines requires the calculation of a lattice of control points over a grid at the same (Δx , Δy) resolution. This is done whether or not a RAM efficient implementation is wanted. Lattice values are derived so that the use of a 16 node set of control lattice values to weight a set of four 3rd order basis functions in both x and y coordinates produces an estimate of surface values in each grid cell. The cubic B-Spline basis functions are

$$B_0(t) = (1-t)^3/6,$$

$$B_1(t) = (3t^3 - 6t^2 + 4)/6,$$

$$B_2(t) = (-3t^3 + 3t^2 + 3t + 1)/6, \text{ and}$$

$$B_3(t) = t^3/6,$$

where t is a normalized grid cell coordinate (0 to 1) in either x or y .

A multi-grid approach is followed, starting with a coarse 4 x 4 grid which is refined at each iteration until the target grid row and column spacing are achieved. Early iterations using, relative coarse grids, provide impose low frequency information in the scattered data. Increasingly finer resolution grids during later iterations are updated to impose high frequency surface information present in the data. At the final grid resolution this compact support avoids global influence from purely local data distributions or local surface roughness. In addition, B-Spline surfaces using 3rd order basis functions are C^2 continuous, which promises high quality interpolation between regions with different data densities.

This general hierarchical B-Spline workflow is shown in Figure 2. This high level workflow is the same for both the original algorithm and the one introduced in this paper. Each rectangular block represents an intermediate grid at some row and column spacing in the multi-grid progression. Numbering the rows and columns of the coarser grid as $nrows$ and $ncols$, each refinement, represented by the ovals, creates a new grid with $2*nrows-3$ rows and $2*ncols-3$ columns at $1/2$ the previous row and column spacing. The refinement process has the feature that - before re-imposing the scattered data z values - the lattice values of the new finer grid represent the same surface model as the lattice grid that was refined. So

¹ The following paragraphs outline more detailed information available in Lee, Wolberg and Shin (1997).

refinement provides a more localized ability to change lattice weights to better fit the available data.

As shown in Figure 2 updating for each data point involves changing local lattice weights stored at the nodes of a 4 row by 4 columns subset of the entire grid centered at the cell holding the data point. Each individual update makes the implicit surface fit that data point's z -value better. For sparse data scattered widely relative to grid row and column spacing, this algorithm interpolates control point z values exactly. However, even at the final, densest grid resolution there may be multiple control points within 2 grid increments of many grid nodes. So the final grid will seldom interpolate the input data exactly. Instead it will pass a smooth, representative surface model through such dense collections of data.

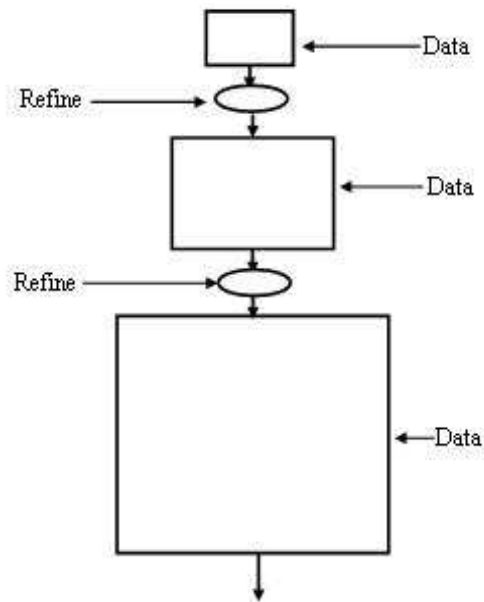


Figure 2: Algorithm workflow for B-Spline gridding, in-memory and disk-based implementations

In fact, the algorithm description above simplifies hierarchical B-Spline data structures and mathematical operations. Because multiple data points may influence the updating of a single grid node, the algorithm performs summation of update information in two auxiliary grids, and only at the end of an iteration updates the grid lattice values. An outline of the more detailed algorithm outline is shown in Figure 3. The two "extra" grids hidden behind the lattice arrays are δ and ω in Lee, et al (1997) in their pseudo code at the start of Section 3.2, while the mathematics to compute individual array values are presented in equations (2) to (5) in their Section 3.1.

Disk-based gridding for large data sets

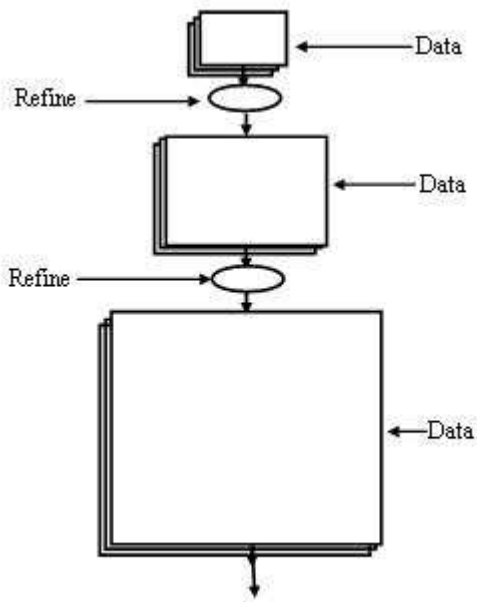


Figure 3. Showing 3 working arrays

CBSG Algorithm

The local, 4 x 4 updating of lattice values is the algorithm feature that allows it store most columns of the current grid model on disk during each iteration. For any one control point updating modifies nodes in only 4 adjacent grid columns. Grid columns to the left of these 4, and columns to their right, might as well be kept on disk. If the data is sorted on its x coordinate then lattice updates will flow across the grid from left to right. As each new control point is reached, the 4 grid columns of interest will be either the 4 currently in memory, or a set of 4 adjacent columns to their right. This local focus of the algorithm combined with sorting of the input data breaks the RAM constraint. The focus on a few columns is indicated by the narrow rectangles inserted in each grid block in Figure 2. These rectangles represent grid columns that are in memory at any one time. As indicated by the arrows, these processing rectangles move across the grid from left to right. Columns of the grid to the left of the window have been updated and can be written to disk. Columns to the right are still on disk, not yet updated, waiting to be read into memory.

The “moving window” just described, with a focus on grid column updating, is also applied to the “refinement” step. Refinement also proceeds from left to right. At any instant

a few columns of the grid at the “input” resolution will be in memory while being used to generate higher density columns of the output grid. The output grid columns have twice as many nodes as the input columns. This is not a problem because only three input columns need to be read into memory to create two output columns. Again, the output columns are written to disk after creation, the refinement process moves to the right, and another input grid column is read into memory.

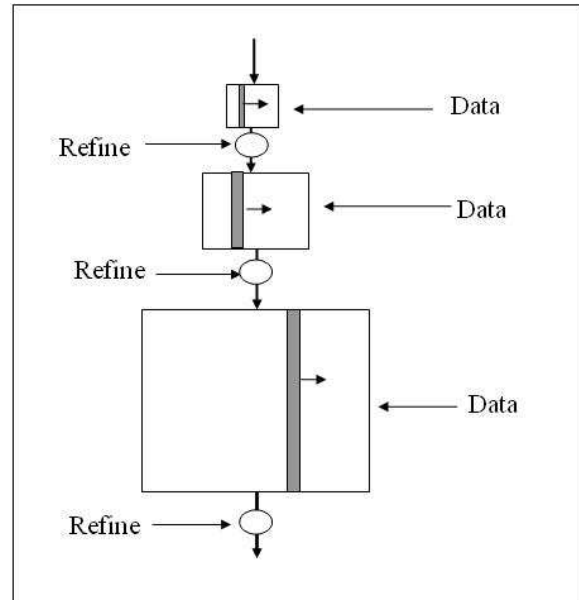


Figure 4: Disk-based workflow for CBSG. Computational window moves from left to right.

Now, in fact, the algorithm description above simplifies algorithm data structures and mathematical operations. Because multiple data points may influence the updating of a single grid node, the original in-core algorithm performs summation of update information in two auxiliary grids, and only at the end of an iteration updates the grid lattice values. As shown in Figure 5, CBSG keeps 4 columns from 3 grids in memory at one time. The actual updating of the lattice values in a left-hand column of the lattice grid is computed from the left hand column of the two auxiliary grids just as the examination of a new control lattice reveals that one or more new – right side – columns of the control lattice need to be read from disk, and left side lattice columns updated and written to disk. New columns for the auxiliary grids are created as needed and node values in those two columns start as zeros. Left hand columns of the auxiliary grids that are no longer needed are simply deleted.

Disk-based gridding for large data sets

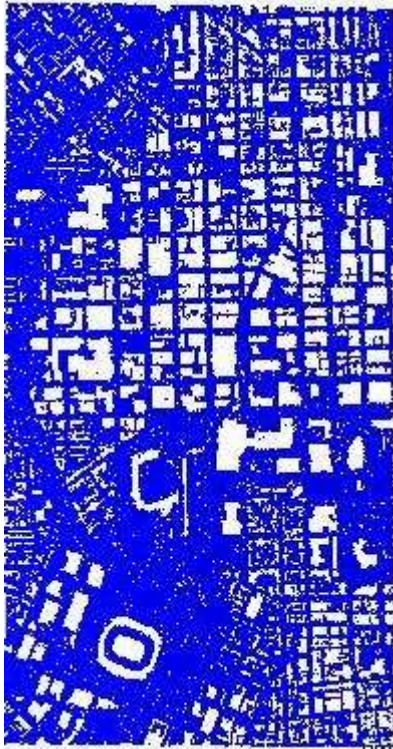


Figure 6. Bare earth view of city center



Figure 7. Map view of interpolated data

LIDAR data from Mount Saint Helens is the source of a second example. I use LIDAR data covering much of the northwest corner of the Smith Creek 7.5-minute, USGS quadrangle, itself located east of the Mount Saint Helens quadrangle. Figure 8 shows data coverage from a small (()) part of the north west quadrangle. The processing statistics for this model are shown in row 4 of Table 1.

Disk-based gridding for large data sets

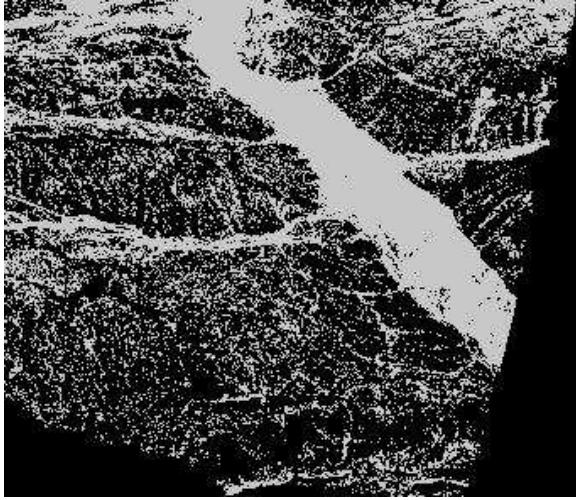


Figure 8. 3D view of bare earth point cloud data

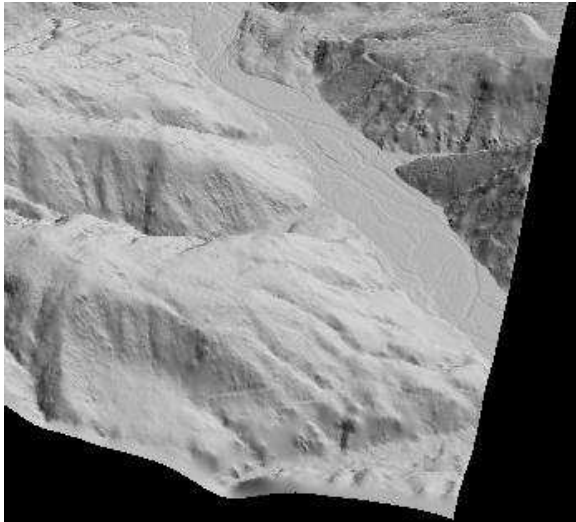


Figure 9

Discussion/Conclusion

An algorithm has been presented for creating rectangular grid-based surface models using disk storage to increase the size of the data sets that can be modeled and the size of the models that can be produced. The method applies multi-grid B-Spline gridding in a novel way such that internal storage space is required for only a few columns of the target grid and one x,y, z triple of data at any one instance. The general workflow and mathematical details for memory-based hierarchical B-Spline gridding are honored, so there is no loss of accuracy from this use of disk storage.

Disk-based gridding for large data sets

References

- Billings, S. D., R. K. Beastson, and G.N. Newsam, 2002, Interpolation of geophysical data using continuous global surfaces, *Geophysics*, 67, 1810-18.
- Franke, R., and G. M. Nielson, 1991, Scattered data interpolation of large sets of scattered data, *International Journal of Numerical Methods in Engineering*, 15, 1,691-1,704.
- Foley, T. , and H. Hagen, 1994, Advances in scattered data interpolation: *Surv. Math. Ind.* 4, 71-84.
- Fortune, S., 1987, A sweepline algorithm for Voronoi diagrams, *Algorithmica*, 2, 153-174.
- G. R. Hjaltason and H. Samet. Speeding up construction of quadtrees for spatial indexing. *VLDB*, 11(2):109–137, 2002.
- H. Mitsova and L. Mitas. Interpolation by regularized spline with tension: I. theory and implementation. *Mathematical Geology*, 25:641–655, 1993.
- Jones, T. A., D. E. Hamilton, and D. E. Johnson, 1986, *Contouring geological surfaces with the computer*, Van Nostrand Reinhold.
- Lee, S., G. Wolberg, G., S. Y. Shin, S. Y., 1997, Scattered data interpolation with multilevel b-splines, *IEEE Transactions on Visualization and Computer Graphics*, 3, 228–244.
- Wolberg, G., 1997, Nonuniform Image Reconstruction Using Multilevel Surface Interpolation, *International Conference on Image Processing (ICIP'97)*, 1007, Volume 1, pages 909-912
- [Agarwal, P., Arge, L., and Danner, A., 2006, From Point Cloud to Grid DEM: A Scalable Approach, *Proc. International Symposium on Spatial Data Handling*, 2006.
- Hodgson, M. E., and Bresnahan, P., 2004, Accuracy of Airborne Lidar-Derived Elevation: Empirical Assessment and Error Budget”, *Photogrammetric Engineering & Remote Sensing*, 331-340.
- Washington State Geospatial Data Archive, Mount Saint Helens – Lidar Data,
http://wagda.lib.washington.edu/data/type/elevation/lidar/st_helens/toutle03.html