

A Surface Modeling Algorithm Designed for Speed and Ease of Use with all Petroleum Industry Data

Steven Zoraster
Landmark Graphics
Corporation

Abstract

A minimum curvature, multi-grid algorithm for building grid-based surface models is presented. The algorithm executes efficiently against all types of petroleum industry data. This presentation is focused on computational techniques that have been used to make the algorithm run fast, especially in cases where the data includes complex faulting.

1. Introduction

This paper describes a commercial surface modeling algorithm, termed Refinement Gridding, developed to provide the speed and simplicity needed today by geoscientists. This algorithm is an implementation of the familiar minimum curvature, grid-based surface modeling technique often used in earth modeling applications. Refinement Gridding takes full advantage of the large memory on modern computers, incorporates several innovative ideas developed to speed up processing, and makes use of previously unreported computational techniques adapted from other surface modeling algorithms. In addition, it honors uncertainty encoded along with the input data.

Refinement Gridding is especially good at handling the faulted data sets that cause the worst performance problems for surface modeling programs. It will provide reasonable and unbiased results that honor the input data with large or small data sets in the presence of all

industry-standard fault representations. The algorithm is so fast that in some cases it makes sense to save disk storage space and database disorder by building surface models as needed for viewing and then deleting them. Certainly, I find it easier to browse disk files and databases when they are uncluttered by surface models that can be rebuilt quickly from readily available data.

Refinement Gridding takes full advantage of the large memory on modern computers, incorporates several innovative ideas developed to speed up processing, and makes use of previously unreported computational techniques adapted from other surface modeling algorithms.

An example of the quality of the results produced by Refinement Gridding from data with complex faulting is provided in the contour map shown in Fig. 1. The surface model from which the contours are traced has 275 rows and 450 columns. The input data had 42,000 seismic shot

points and 6000 fault segments. The model was created in 4.25 CPU seconds on a 1600 MHz PC with 2.5 gigabyte of memory running Windows 2000.

The rest of this paper is organized into four parts. Section 2 presents timing statistics for several data sets and explains why the algorithm is good for you. Section 3 describes the Refinement Gridding algorithm itself with an emphasis on the computational aspects that make the program fast. Section 4 presents some more results. Conclusions are in Section 5.

2. Why this is of interest

There are three reasons why someone who uses computer-based surface modeling might be interested in this new algorithm. First, it produces grid-based surface models quickly from all types of data, including well picks, two-dimensional (2D) seismic, 3D seismic, and any combination of these data types. Two-dimensional fault data are honored in the form of opaque fault centerlines, opaque fault polygons, and fault centerlines that encode fault vertical separation data.

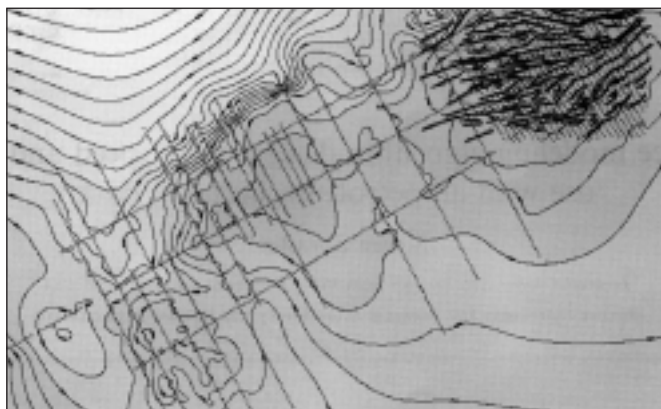


Figure 1.
Contour map created from a particularly troublesome 2D seismic data set with dense faulting and dense seismic coverage in one corner of AOI.

| Data set name | Data type | Number of points | Number of fault vertices | Rows × columns | CPU seconds 1600 MHz Windows 2000 | CPU seconds 750 MHz Solaris |
|----------------|----------------|------------------|--------------------------|----------------|---|--------------------------------|
| Porosity data | Porosity/wells | 240 | 8737 | 137 × 161 | 1.5 | 2.9 |
| Kufpec 2D | 2D seismic | 2476 | 604 | 113 × 113 | 0.5 | 1.8 |
| Seismic 2D | 2D seismic | 42,095 | 6034 | 275 × 445 | 4.4 | 15.1 |
| Loihi | Bathymetric | 1,300,000 | None | 1191 × 1669 | 130.1 | 361.0 |
| Gulf of Mexico | 3D seismic | 2,070,000 | 1480 | 3201 × 3387 | 396.0 | 1537.0 |

Table 1. Data sets and processing times

Table 1 provides information about the speed of this algorithm against various data with point set sizes between 200 and 2,000,000 values, for output models with up to several million grid nodes, on both a PC running Windows 2000 and a Sun Blade 1000 workstation running SUNSV. Modeling the larger faulted data sets with traditional algorithms is computationally impractical in some cases.

The second reason is that it demands few control parameters besides those needed to specify the output surface model extent (area of interest, AOI) and the spacing between grid rows and columns. An innovative AOI default setting program provides detailed analysis of the data to assist in setting these parameters.

The third reason is that the algorithm honors numerical confidence factors for input data either at the data set level or at the level of individual data points. When using multiple sources of data, the user can specify confidence factors for different data sets based on subjective confidence on the input data. With the increased interest in uncertainty measurements in all aspects of the petroleum industry, this will allow better modeling of multiple data sets representing fundamentally different levels of accuracy (for example, wells and seismic). It will also improve modeling based on similar types of data with different levels of accuracy (for example, seismic data of different vintages).

3. Refinement Gridding algorithm

Refinement Gridding is an implementation of the minimum curvature method often used in earth science surface modeling

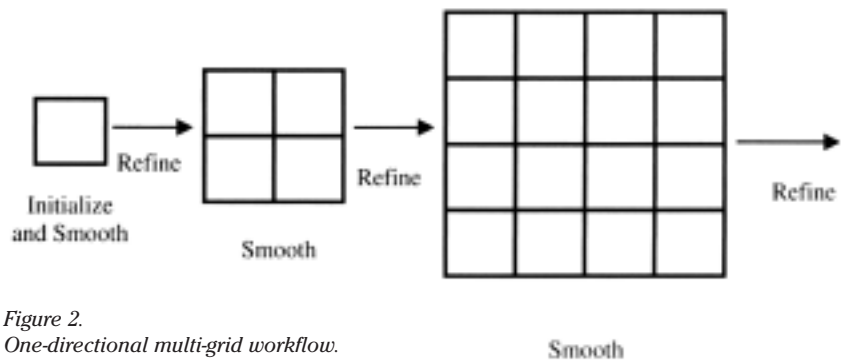


Figure 2. One-directional multi-grid workflow.

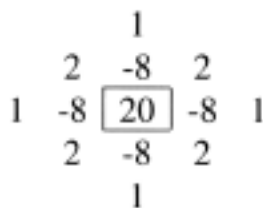


Figure 3. Full 13-node biharmonic operator weights.

(Briggs, 1974; Smith and Wessel, 1990). It is implemented through a one-directional multi-grid technique¹ that moves from an initial coarse grid, initialized in this case to the average of the input data, through a series of finer grids until an approximation of a minimum curvature surface is produced at the desired grid row and column spacing. Working at the coarse spacing imposes the low spatial frequencies of the data on the surface and working at the finer spacing imposes the higher spatial frequencies of the data on the surface. Fig. 2 provides an overview of the general one directional multi-grid method.

At each grid refinement level the current grid-based surface model is treated as an elastic membrane and a convergent linear iterative deformation operator is applied repeatedly at each node to achieve an

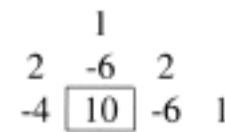


Figure 4. Partial eight-node biharmonic operator used in grid column 2 at bottom of the grid.

approximation to a minimum curvature surface that honors both the input point data and discontinuities encoded in fault traces. The deformation that is applied at each grid node is calculated on the basis of a “molecular summation” (Terzopoulos, 1988) which compares the weighted summation of 12 neighboring nodes with the current value of a central target node to calculate a new value for the target node. Fig. 3 shows the 12 neighboring node weights around a target node used in the complete biharmonic operator.

Near the edge of a grid, near void grid nodes, or near opaque faults a deformation operator using fewer grid nodes must replace a complete 13-node operator. Fig. 4 shows the weights for one such operator. See Terzopoulos (1988) for the simple method to construct all such operators.

¹ “What are Multigrid Methods?”, www.mgnet.org/mgnet/tutorials/xwb/mg.html.

Interlaced between the smoothing of grids at different levels of detail are grid refinement operations. These take an intermediate grid of i rows and j columns and create a new grid with $2*i-1$ rows and $2*j-1$ columns. The refinement process is difficult only because it must honor input faults, which form an important part of the information input to the gridding process. With or without faults, when filling in new grid nodes on a finer grid from a coarser grid, the interpolation process is strictly linear. In Refinement Gridding the serious mathematical work of making a surface that looks nice and honors the data is done by the grid smoothing/deformation operators.

None of this is particularly new, and the reader should review the literature, especially Terzopoulos (1988) to understand the basic mathematical approach applied in Refinement Gridding. What is new here is our full use of the very large amounts of memory available on modern computers, innovative computational techniques developed for this algorithm, and previously unreported techniques adapted from other surface modeling algorithms.

3.1. Pre-computation of molecules

Considering all possible combinations of void nodes or nodes hidden from view by faults there are 4096 (212) biharmonic molecules, each with 13 summation weights. (Weights corresponding to missing nodes are assigned a value of 0.0.) At the start of the algorithm all weight combinations for all molecules are computed and stored in a 2D array with 4096 rows and 13 columns. For each grid refinement, before starting the deformation process the algorithm scans the grid and determines which molecule applies to each node. This information is recorded as an integer between 0 and 4095 which points into the table of deformation weights and is stored in a short integer array of the same size as the grid to be deformed.

The deformation process then proceeds by repeatedly visiting each node in an orderly fashion, looking up which molecule summation operator to apply and applying it. Fourteen multiplications, 15 additions, and one division are required, but no conditional branching and no dynamic creation of summation molecules are necessary.

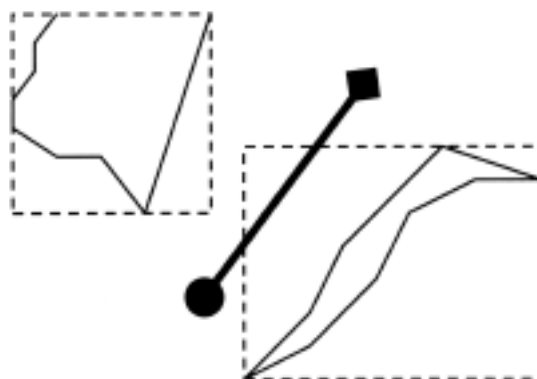


Figure 5.
Heavy line with markers at each end connects a data value to a grid node. Only one of two fault traces needs to be checked to see whether it crosses line because line crosses only one of bounding boxes around fault traces.

There are other important tricks in applying these molecules. Most involve faults and control points. The most important problem is finding out whether there are fault segments nearby.

3.2. Fault processing, fault boxes

Fault traces are input to this algorithm as a sequence of records, one for each fault vertex. Each record is assumed to have four fields, x , y , segment ID, and vertical separation. The segment ID field is a simple integer value whose change indicates the start of a new fault. The vertical separation value may exist or not. If all of them along a single fault are null marker values, then the fault is “opaque”, and surface and data information from one side of the fault usually does not impact the surface shape on the other side of the fault.

Each individual fault trace, defining a single line fault or a fault cutout is stored in working memory in a simple data structure with the coordinates of a bounding box, as shown in Fig. 5. When the algorithm needs to know whether a 2D line joining, say a data point and a grid node, crosses any fault trace the question can be answered relatively efficiently because for most fault traces that line will not cross the bounding box. Therefore, the detailed problem of checking for crossing with individual fault segments can be focused on only a subset of fault traces. This efficiency is one of several used in Refinement Gridding that will be described in this paper.

3.3. Fault processing, fault rasterization

Using fault trace bounding boxes helps to improve the efficiency, but the process of deciding whether faults and given line

segments cross is fundamental to this algorithm, and must be addressed even more efficiently. Part of the answer is “fault rasterization”, which provides needed information about where faults cross rows and columns of the grids.

At each refinement level, all fault segments are processed to determine which grid node connections are crossed by fault segments. It turns out to be computationally inexpensive to determine node-to-node connections crossed by each segment. This information is encoded for each north-south and east-west grid node connection in the form of 2 bit data, which records values “visible”, “opaque”, or “vertical separation”. The first two values tell the algorithm whether an opaque fault intervenes between two nodes. The last code “vertical separation”, tells the algorithm that a fault’ segment encoding vertical separation across the fault intervenes.

The 2 bit data are packed and stored for each grid node in an array, the same size as the grid being smoothed. This information is used to help pre-compute which deformation molecule is to be applied to each grid node.

3.4. Fault processing, distance transformations

Fault rasterization addresses the problem of building deformation molecules efficiently. Still, “seeing” faults in general can be a problem. We really need to know when we are near a fault. This is necessary when the algorithm is processing control points and deciding if nearby faults have to be considered. We would like to do this efficiently. A “distance transformation” is the answer.

A distance transformation (Borgefors, 1986) is an image-processing algorithm developed to convert a 2D raster image consisting of pixels that correspond to features and pixels that do not into a “distance map” where all non-feature pixels hold the approximate distance to the closest feature pixel. This can be done very efficiently using a simple convolution operator. In our case the pixels are the grid, and the “feature pixels” are simply grid nodes that are near faults. (These are available because of the work described above in Section 3.3.) The result of the distance transformation is a new grid in which each node contains a close (accurate within 5%) measurement of the distance to the nearest fault segment. An example is shown in Fig. 6.

We have inverted a standard process in 2D surface modeling. Instead of visiting a grid node in general and then asking whether there is a fault nearby, we have precomputed an approximation to the nearest fault segment. So, when asking if a particular possibly intervening fault trace breaks the direct line connection between a data value and a grid node, we can compare the 2D distance between the data point and the grid node and the pre-computed distance from the grid node to the

nearest fault. If the pre-computed value is larger then no finer grained checks are needed. Since in practice most grid nodes do not have “nearby faults”, in most cases this simple comparison is all that is needed. The algorithm then decides that if the distance transformation measured distance to the nearest fault for a particular grid node is larger than say 3 grid increments there is no reason to resort to the fault box check described earlier.

Distance transformations are useful, see Zoraster (1996) for a previous application.

3.5. Fault blocks

A potential problem with the filter-refine-filter paradigm is that small, and not so small, fault blocks will not be represented at some of the coarser grid spacings. So as the grid is refined the grid nodes in newly revealed fault blocks must be initialized before they can be smoothed. In Refinement Gridding, the fill-in process is a dynamic and fast bleed of data from already filled nodes through open ends of fault blocks if possible. If impossible, then the algorithm simply ignores opaque fault barriers to pick up values from nodes in normally “off-limits” neighboring fault blocks.

The newly “opened” fault blocks revealed during refinement may or may not have data associated with them. If one does, then everything is nice because it is initially, by definition, a small block that should have time to achieve a minimum curvature representation that honors the data as it grows. “Grows” in this usage is a computer science term, meaning the gradual addition of new grid nodes inside the fault block with more refinements, and not some newly revealed dynamic of geology.

If a new fault block does not have data associated with it, then everything is still nice because it is initially, by definition, a small block that should have time to achieve a minimum curvature representation as it grows. This representation will be derived from data values assigned from neighboring fault blocks that provide reasonable starting points for an otherwise intractable problem.

Fig. 7 shows utility of fault block fill-in process on a highly faulted well data set. Porosity measured at well locations is being interpolated and many of smaller fault blocks modeled have only one or two wells in them. Some have no wells.

3.6. Control points

An important part of any surface-modeling algorithm is the ability to match the data. In Refinement Gridding grid nodes near data constantly have their values adjusted to honor nearby data. Our approach is to use the Taylor series expansion-based modification of the local deformation operator at that grid node as described by Smith and Wessel (1990). When multiple data values are within $\frac{1}{2}$ of the grid node spacing, the algorithm averages all such values to obtain a single composite value that is used to constrain the grid node value.

Of course, after composite values are computed we need to assign them to grid nodes. This has a surprising consequence that led to a development detour. The algorithm keeps track of in-memory records for all input data values and these records are assigned an integer pointer tying them to the nearest grid node. After each grid refinement the pointers change and the records are sorted to bring records with the same grid node pointer

| | | | | | | |
|------|------|------|------|------|------|------|
| 2.32 | 1.37 | .96 | 1.37 | 2.32 | 2.74 | 3.70 |
| 1.37 | .96 | 0.0 | .96 | 1.37 | 2.32 | 2.74 |
| .96 | 0.0 | .96 | 0.0 | .96 | 1.37 | 2.37 |
| 1.37 | .96 | 1.37 | .96 | 0.0 | .96 | 1.37 |
| 2.32 | 1.91 | 2.32 | 1.37 | .96 | 0.0 | .96 |
| 3.28 | 2.87 | 2.74 | 2.32 | 1.37 | .96 | 1.37 |
| 4.24 | 3.82 | 3.70 | 2.74 | 2.32 | 1.91 | 2.32 |

Figure 6. Distance transformation applied to a normalized unit grid showing distances from a rasterized fault indicated by shaded 0s in grid.

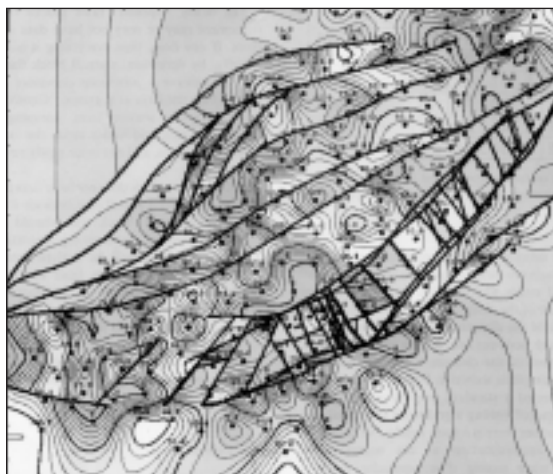


Figure 7. Contour map created from a particularly troublesome 2D seismic data set with dense faulting and dense seismic coverage in one corner of AOI.

together for averaging into a composite value. Initially I simply used the C standard qsort algorithm. I then found that this was the major (embarrassing) bottleneck detected during testing, with sorting taking up to 50 of the algorithm CPU time on Windows 2000. At this point I coded and inserted a version of the Sedgewick (1998, pp. 120-135) combination quicksort-insertion sort algorithm into the algorithm. This brought the algorithm data sorting time down to a more reasonable 10 of total CPU time on Windows 2000.

3.7. Confidence factors

From the algorithm point of view confidence factors are associated with each individual input data value. In fact, they may originally be associated with a survey or data source, and simply be assigned to each data point within a survey before those data are sent to the Refinement Gridding algorithm. In any case, confidence factors are positive numbers that are interpreted relative to each other inside Refinement Gridding. If the confidence values of two input data values must be compared, then it makes no difference whether they are 1 and 2, 50 and 100, or 1000 and 2000; the results are the same. The only expectation is that the relative calculations are valid across all input data.

I chose relative confidence factors for three reasons. First, it was easiest from a programming viewpoint. In application, confidence factors come into play during any stage in the grid smoothing process when two or more data values correspond to the same grid node. In that case the composite control value assigned to that grid node is simply the weighted average of all of those data values and the weights are the confidence factors.

Second, a simple relative confidence scheme is easy to explain to users. And to GUI developers, documentation writers, and program testers within my own company. Plus the wife, but not the kids.

Third much confidence data that have been collected within the petroleum industry is in the form of categorical rankings, such as "poor", "average" "good", or "very good". It is intuitively easy to ask

the user to assign numeric rankings to these categories to get a quick start on the surface modeling application. Given the speed of Refinement Gridding, it is not unreasonable in many cases to ask the user to make several such assignments and to build several surface models while determining exactly which assignment most closely represents his or her best mapping of those category values into numeric values.

3.8. Memory usage

Data input to Refinement Gridding persists in memory. This includes the data quadruplets for each input value (x , y , z , confidence factor), and all fault vertices along with the bounding boxes for each fault trace. As processing continues other data structures move into and sometimes out of memory.

After each refinement there is a new version of the grid-based surface model being smoothed. This is accompanied by node-to-node visibility array made up of those 2 bit fault visibility values produced by fault rasterization. Because these 2 bit packets are needed for all four Manhattan directions from each node they take up $\frac{1}{4}$ the memory required by the grid. Memory is also required to compute and then temporarily hold the composite data values associated with individual grid nodes as described above in Section 3.7. For dense data evenly spaced over the entire AOI there may be almost as many of these composite values as there are grid nodes.

While the composite values are being calculated it is necessary to have access

to information about the distance to the nearest fault from each grid node. This means a distance grid is needed temporarily, produced by the distance transformation described in Section 3.4. The distance transformation output grid is deleted as soon as it is no longer needed as are the composite data values, but naturally the algorithm memory requirements keep on growing as long as it keeps running and the current grid refinement keeps growing. The 3D GOM seismic data set listed in Table 1 required 860 megabytes of memory to create the grid model with its almost 11 million output grid nodes from 2 million shot points. This was approximately 34% of the available memory on my Windows 2000 machine.

As hinted in the preceding paragraphs, the relationship between input data set size, output grid size, and memory usage is not linear. The seemingly much more modest 2 million node surface model created from the 1.3 million sounding Loihi data set required 370 megabytes of memory, 14% of that available (Fig. 8). The large memory demand was caused by the bathymetric data being evenly spread over the target AOI. As the grid size increased the number of composite data values increased, so that there were almost as many composite values at each refinement as there were grid nodes.

3.9. Grid spacing default setting

Several numerical criteria may be applied for deciding the row and column spacing for the grid behind the surface model for use with Refinement Gridding. Several of these criteria analyze input

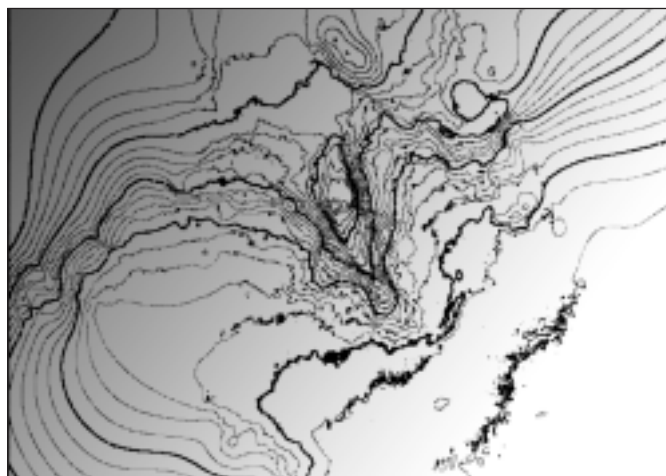


Figure 8.
Sea bottom model created from Loihi data set provided by Naval Oceanographic Office.

data utilizing computational tools developed for use in Refinement Gridding itself. Although computationally efficient and important to good surface modeling, the application of these tools, unfortunately, remains ad hoc.

The most important tool is the ability to assign rapidly each input data value to its nearest grid node, even in the presence of faults (remember the distance transformation). Since we can do this quickly, it is cost effective to try multiple grid spacings, working from coarse to fine, to discover a grid that meets some criteria for a “dense enough” grid.

One fairly standard criterion is that only a small percentage of grid nodes should have data nearby. In sparse well data especially, a rule of thumb is that we normally want an average of 10-16 grid nodes without data for every node that is directly controlled by a nearby data point. This is to allow the surface model room to smooth out away from the hard picks. On the other hand, with denser but more continuous seismic data coverage, this criterion is often relaxed for this inherently noisy data until, for practical computational reasons, we may even accept fewer grid nodes than data.

A second criterion involves the study of the difference between the maximum value of the z coordinate of one of the multiple data values associated with a single grid node and the minimum of those data values. The maximum value of such a range, which of course is always positive, should be small relative to the total z-range of the input data. Otherwise, at the target grid spacing input data with large differences in surface value are collected near at least one grid node.

The “at least one grid node” problem mentioned in the previous paragraph led to another related idea: So what if the data range is large at one location in the AOI, if the “normal” data range is small overall? And this led to establishing another criterion that was dependent on the sample distribution of the z-range statistic across all grid nodes. We look at the sample mean plus several standard deviations of that measurement. If that number is low compared to the range of the input data, then the grid spacing may be good.

Note that the above paragraphs are filled with phrases like “is low compared to”, “should be small”, and “a rule of thumb”. Which means that I do not have the answer on how these decision criteria should be combined. What we have is a decision tool that combines all three criteria using if-then logic and makes decisions on those rules. For example, if the maximum z-range at any grid node is less than say 5 of the data range at some grid spacing the default setter will stop.

4. Discussion

As mentioned in the introduction, Refinement Gridding is an implementation of the familiar minimum curvature, grid-based, surface modeling technology. What makes it most unique are efficiencies achieved by applying various techniques such as pre-computation of summation molecules, fault rasterization, distance transformations, and carrying around pointers to nearby control points for each grid node. All of these are possible because of the large amount of memory now available on just about any computer used in earth modeling.

These efficiencies are important for the data sets on which the algorithm has been tested. I expect it to be even more important in the future as data set sizes increase, with individual 3D surveys being collected that have over 10 million shot points, and with the desire of surface modelers to take any old data set, or any collection of data sets, and simply hurl them at an algorithm to get a reasonable model returned quickly.

Looking back over this paper it should be obvious that most of the individual efficiencies are associated with handling faults, specifically resolving the question of whether a straight 2D line segment joining two adjacent grid nodes or a data value and a grid node cross a fault segment. This is not surprising given that handling faults in 2D or 3D is possibly the major computational problem in earth modeling. For Refinement Gridding the “fault issue” was handled largely by inverting the problem and using a hierarchy of techniques to figure out which grid nodes are not near faults, and then which fault traces could not possibly be between a data value and a grid node.

5. Conclusions

It is possible to make very large surface models using the minimum curvature criteria from very large data sets in the presence of complex faulting. Good algorithm design and the innovative techniques borrowed from other disciplines are required to achieve this goal. The fact that this functionality exists will modify the way people work with large data sets. It will do so by encouraging them to make the large grids necessary to fully honor the high frequency information in those models, and by encouraging them to do so without much worry about disk storage.

References

- Borgefors, G., 1986. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34** (3), 344-371.
- Briggs, I.C., 1974. Machine contouring using minimum curvature. *Geophysics* **39**, 39-48.
- Sedgewick, R., 1998. Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching, 3rd Edition. Addison-Wesley, Reading, MA.
- Smith, W.H.F., Wessel, P., 1990. Gridding with continuous curvature splines in tension. *Geophysics* **55** (3), 293-305.
- Terzopoulos, D., 1988. The computation of visible-surface representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **10** (4), 417-437.
- Zoraster, S., 1996. Imposing geologic interpretations on ; computer-generated contours using distance transformations. *Mathematical Geology* **28** (8), 969-985. ■