SANDIA REPORT

SAND2003-3300
Unlimited Release
Printed September 2003

Three-Dimensional Representations of Salt-Dome Margins at Four Active Strategic Petroleum Reserve Sites

Christopher A. Rautman and Joshua S. Stein

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy
under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:(865)576-8401
Facsimile:(865)576-5728
E-Mail:reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:(800)553-6847
Facsimile:(703)605-6900
E-Mail:orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2003-3300
Unlimited Release
Printed September 2003

Three-Dimensional Representations of Salt-Dome Margins at Four Active Strategic Petroleum Reserve Sites

Christopher A. Rautman
Underground Storage Technology Department
Sandia National Laboratories
P. O. Box 5800
Albuquerque, New Mexico 87185-0706

Joshua S. Stein
Performance Assessment and Decision Analysis Department
4100 National Parks Highway
Sandia National Laboratories
Carlsbad, New Mexico 88220

ABSTRACT

Existing paper-based site characterization models of salt domes at the four active U.S. Strategic Petroleum Reserve sites have been converted to digital format and visualized using modern computer software. The four sites are the Bayou Choctaw dome in Iberville Parish, Louisiana; the Big Hill dome in Jefferson County, Texas; the Bryan Mound dome in Brazoria County, Texas; and the West Hackberry dome in Cameron Parish, Louisiana.

A new modeling algorithm has been developed to overcome limitations of many standard geological modeling software packages in order to deal with structurally overhanging salt margins that are typical of many salt domes. This algorithm, and the implementing computer program, make use of the existing interpretive modeling conducted manually using professional geological judgement and presented in two dimensions in the original site characterization reports as structure contour maps on the top of salt. The algorithm makes use of concepts of finite-element meshes of general engineering usage. Although the specific implementation of the algorithm described in this report and the resulting output files are tailored to the modeling and visualization software used to construct the figures contained herein, the algorithm itself is generic and other implementations and output formats are possible.

The graphical visualizations of the salt domes at the four Strategic Petroleum Reserve sites are believed to be major improvements over the previously available two-dimensional representations of the domes via conventional geologic drawings (cross sections and contour maps). Additionally, the numerical mesh files produced by this modeling activity are available for import into and display by other software routines. The mesh data are not explicitly tabulated in this report; however an electronic version in simple ASCII format is included on a PC-based compact disk.

ACKNOWLEDGMENTS

This work was supported by the U.S. Department of Energy, Strategic Petroleum Reserve Project Management Office, in New Orleans, Louisiana.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

CONTENTS

FIGURES

TABLES

Three-Dimensional Representations of Salt-Dome Margins at Four Active Strategic Petroleum Reserve Sites

INTRODUCTION

This report presents the results of three-dimensional computer modeling of the outer margins of salt domes at the four currently active (2003) U.S. Strategic Petroleum Reserve (SPR) sites in the Gulf Coast region of Louisiana and Texas. These models are presented in numerical form for potential use in other applications, and they are visualized for reference purposes using a variety of graphical representations.

These models are essentially *direct transformations* of the paper-based existing site-characterization models, or understandings of the external geometries, of the four active SPR salt domes to a computer-graphical format. This model conversion has been undertaken as-is, with the full realization that the existing site characterization reports are modestly dated and the interpretations therein are one to two decades old (Hogan, 1980a, 1980b; Whiting et al., 1980; Hart et al., 1981; Magorian and Neal, 1988; Magorian et al., 1991; Neal et al., 1993, 1994).

The geologic description and interpretation of the four salt domes and their contained Strategic Petroleum Reserve sites is being updated separately as a part of ongoing SPR Project site characterization activities. These newest updates will be published over the course of the next one to three years. The geology will be updated and evaluated critically in light of new (approximately post-1990) data including new drilling and hydrocarbon development activities, cavern-operational experience, seismic exploration data, and other information that may have been acquired or otherwise made available since the completion of the older formal site-characterization update reports. Additionally, inconsistencies recognized via fully three-dimensional modeling of the older data, which may have been difficult or impossible to recognize in a two-dimensional compilation environment, will be identified, described, and reconciled through new interpretations and/or model-construction approaches.

MODELING

Computer-based numerical modeling inevitably is closely tied to the modeling software employed in the task. The models described in this report were generated using the 3-D modeling capabilities of ***M***ining ***V***isualization ***S***ystem (MVS)[1], a high-end geologic modeling software product marketed by C-Tech Development Corporation, of Huntington Beach, California. MVS originated as an environmental-modeling product, but has undergone substantial expansion of capabilities during the past 5-6 years. Additional information regarding C-Tech Development Corporation and its software products may be accessed over the internet at **`http://www.ctech.com`**.

DATA SOURCES

The fundamental sources of data used to generate the three-dimensional representations of the four SPR salt domes are the original site-characterization reports and the corresponding update reports as listed in table 1. No new data have been used. Also, reinterpretation of the existing data and models has been minimal and restricted to the cases where some additional interpretation was required simply to generate the necessary data for the model conversion.

[1] The use of trade, product, industry, or firm names is for descriptive purposes only and does not imply endorsement by Sandia National Laboratories or the U. S. Government.

Table 1. Geologic Site Characterization Reports for the Strategic Petroleum Reserve

SPR Site, version		Report Number	Citation	Structure Contour Map (figure in original report)
Bayou Choctaw	original	SAND80-7140	Hogan (ed.), 1980a	Figure 6.19
	update	SAND92-2284	Neal et. al., 1993	Figure 6b
Big Hill	original	SAND81-1045	Hart and others, 1981	Figure 5-1
	update	SAND88-2267	Magorian and Neal, 1988	Figure 1
Bryan Mound	original	SAND80-7111	Hogan (ed.), 1980b	Figures 6-1, 6-2
	update	SAND94-2331	Neal et. al., 1994	Figure 4b
West Hackberry	original	SAND80-7131	Whiting (ed.), 1980	Figure 4.9
	update	SAND90-0224	Magorian et. al., 1991	Figure 7

MODELING TECHNIQUES

The natural geometry of salt domes presents certain problems for many, if not most, general-purpose geological modeling software programs. Specifically, the generic geometry of a salt dome is a quasi-cylindrical mass with steeply dipping flanks that may exhibit local overhanging configurations. In addition, the crest, or upper surface, of a salt dome is typically near horizontal because of dissolution of the salt mass over geologic time by laterally moving near-surface ground waters. Subtle variations in the specific dissolution conditions at a particular site may cause local complexities in this upper surface. The consequence of the occurrence of salt domes as vertically extensive steeply dipping features with an upper boundary at nearly 90 degrees to the dome flanks is that numerical modeling of these differing *geometric* features, in effect, requires two separate approaches.

Confounding the mechanics of numerical modeling is that, in most instances, the spatial arrangement of data available from which to develop the model is one of near-vertical "strings" of measurements or observations: i.e., data obtained from drill holes such as oil and gas wells. Although the flat-lying upper surface of most salt domes is readily described by vertical penetrations of the top-of-salt surface, the flanks of a salt dome are rarely well defined except in locations of closely spaced drilling. Although many such regions of "close" drilling are associated with hydrocarbon development in the upturned sediments along the flanks of numerous salt domes, the extent of such drilling is not necessarily uniform around the circumference of a given dome. This irregular spacing of well control may induce artifacts into surfaces generated by numerical algorithms. Because seismic data was not available at the time that the original site characterization reports were compiled, the geometry of the SPR salt domes as originally characterized was constrained solely by drill hole information.

The suite of numerical algorithms generally available in geologic modeling software is particularly ill-suited for the representation of near-vertical surfaces. Typically, the principal interest of many geologic studies is sedimentary basins and other environments in which the geometry of the geologic units is "layer-cake" in nature and relatively uncomplicated structurally. More specifically, structural overhangs of salt pose severe difficulties because most modeling algorithms are predicated upon a single-valued ("functional") relationship of the surface in two-dimensional (plane) space. Interpolation-type algorithms are designed to move "smoothly" from the observed elevation at one *x,y* position to the observed elevation at another *x,y* position along a grid of *x,y* locations at which the elevation of the geologic surface is unknown. Typically this is accomplished by weighting the different nearby *measured* elevation values in some manner that produces a visually

pleasing resultant surface. The introduction of the possibility of multiple elevations at a single (known or unknown) 2-D spatial position leads to a complete breakdown of the computational algorithm and further processing is either impossible (leading to system crashes) or produces absurdly impossible representations of the geology.

Modeling of the Salt Dome Flanks

One method to deal with the problem of geological overhangs and multiple elevation values for a given horizontal position associated with the geometric "flank" of a salt dome is to abandon (partially; see below) the assumptions inherent in most geological modeling software and to adopt the spatial principles involved in engineering finite-element meshes. An engineering *mesh*, as that term is used in this report, involves the complete three-dimensional spatial specification of nodes and an explicit specification of how those nodes are connected together to form the mesh. Although most computational meshes are "three dimensional" and involve the specification of volumetric elements, the bounding surface of a salt dome may be fully specified by a two-dimensional (surface) mesh in three spatial dimensions.

Conceptually, the transition from a near-vertical geologic surface with potential recumbent overhangs to a 2-D finite-element-type surface mesh in three dimensions is straightforward. The difficulty in actually constructing such a mesh comes in that — unlike the classical engineering application of mesh construction in which the geometry and connectivity of the mesh is pre-specified by the numerical analyst — the application of these techniques in the geologic environment involves inference of the unknown full geometry and connectivity from sparse, spatially scattered measurements. This is precisely the reason that the generally available modeling algorithms adopt the *implicit* connectivity specified by the interpolation-onto-a-horizontal-grid approach.

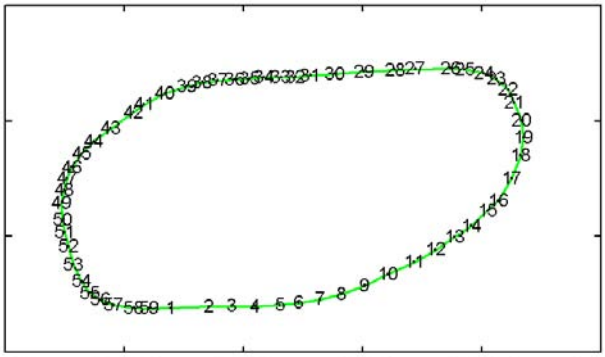
All that is required to generate a 2-D surface mesh representing the margins of the salt domes is the spatial positions of a large number of points on that bounding surface and a means of knowing how those points should be connected to define a number of polygons to approximate that geologic contact. The interpretive structure contour maps drawn on the top-of-salt surface and presented in the various site characterization reports for the Strategic Petroleum Reserve project (table 1) provide exactly this mechanism.

Although a structure contour map is a strictly two-dimensional representation on a flat piece of paper, it represents a three-dimensional object: a model. The vintage of the site-characterization reports (late 1970s and published in 1980 and 1981) strongly suggests that the models represented by the contour maps were constructed by hand using geological judgment constrained by the available distribution of observations of the salt-sediment interface in the boreholes existing at that time. Nevertheless, two-dimensional contours can be discretized (digitized) and converted to a large number of points in 3-D space by calibrating the horizontal position of each discrete point to some reference grid and assigning a *z*-value equal to the indicated elevation/depth of the specific contour line. Maintaining a sequential order of digitization, complemented by a depth-ordering of individual sequences of digitized points representing the different contour lines allows us to specify the connections between 3-D points defining the outer surface of the salt dome. All digitization work for this modeling effort was conducted in the state plane coordinate system (NAD-27) appropriate to the location of the salt dome in question.

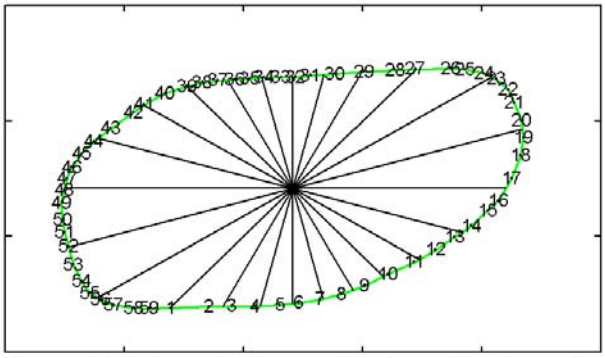
In practice and for convenience in automated numerical construction of the actual meshes, the process described in the preceding paragraph is slightly adapted, as follows. This logic is implemented numerically in program **`ctr2evs`**. The source code for this program is presented in Appendix A, and the Fortran code itself in straightforward ASCII format is contained on the compact disk included with this report.

1. Each structure contour drawn on the top-of-salt surface is digitized sequentially at a horizontal interval that allows the resulting line segment to approximate the original contour line [fig. 1(a)]. Each *x*-*y* spatial position along the *closed* contour is assigned the indicated *z*- (elevation-) value. Manipulation of *open* contours is not currently supported.

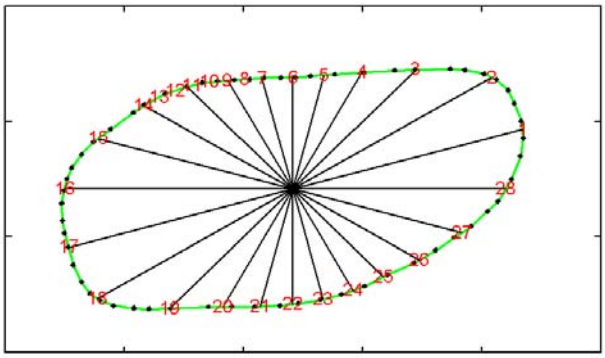
2. The resulting sequential arrays of *x*-*y*-*z* coordinates are sorted by elevation (z) value, maintaining the internal sequential order in *x*-*y*.



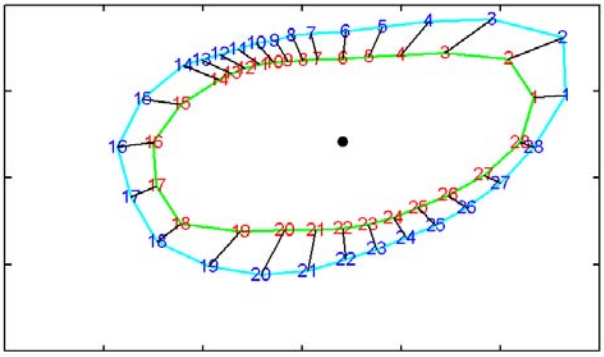
(a) Example of a digitized structure contour. The digitized points are numbered sequentially (beneath the center of the number) and they are connected by straight-line segments.



(b) A number of rays at a constant angular increment (here, 15 degrees; typically 2 degrees in practice) are passed from the centroid of the discretized contour to intersect the line segments defined by the digitized points.



(c) The intersection points defined by the rays in the previous step form the new nodal points of the modeling mesh. Note that additional nodes may need to be defined if a ray has multiple intersections with a discretized contour (of any elevation). In this illustration, nodes numbered 8 through 13 have been modified because of multiple intersections on one or more other contours. Note also that the nodes have been renumbered for consistency starting from due east.



(d) A second structure contour at some elevation below the current contour has been processed in the same manner and added to the illustration. The modeling mesh is generated by connecting correspondingly numbered nodes, as indicated. The process is repeated for all available closed contours. One method of creating the top of the salt dome is simply to connect the nodes of the highest elevation contour to the contour centroid (black dot); see discussion in text.

Figure 1. Conceptual representation of the procedure involved in creating a mesh from digitized structure contours. Reentrant contours add significant complexity and have been omitted from this simplistic explanation.

3. Each digitized contour is first resampled at a constant angular increment, as measured from the centroid of the closed contour [fig. 1(b)]. In effect, a ray is projected from the centroid until it intersects the appropriate discrete line segment. Multiple intersections of the same angular ray with other line segments are identified and tracked. The resolution of the output mesh is determined to a first order by the angular increment selected at this stage.
4. The contour line with the greatest number of ray intersections is identified, and a second resampling of the original line segments is performed to generate an identical number of spatial nodes for each contour digitized; these nodes are identified in counter-clockwise (trigonometric-angular sequence) from *1* through *N* [fig. 1(c)].
5. A mesh describing the geometry of the portion of the salt dome represented by the structure-contour map is then constructed very simply by connecting sequential nodes for one contour with the corresponding sequentially numbered nodes for the contours immediately "above" and "below" in elevation [fig. 1(d)].
6. The resulting nodal values and connectivity definitions are written to an ASCII text file in the format desired for use (visualization) by a specified software package, here MVS.

Note that because of the typical geometry of a salt dome as a flat-topped, steeply dipping quasi-cylindrical mass, the uppermost structure contour is generally located relatively close to the near-vertical margins of the salt. The modeling algorithm described above, being limited to the vertical extents of the interpreted structure contours, thus produces an open-ended quasi-cylindrical surface representing the flanks of the dome. Some structure contour maps may utilize a vertically varying contour interval near the top of the dome for increased resolution (i.e, 500 ft vertically over most of the extent of the dome supplemented by 100-ft contours near the top surface). However, the usual situation is that the very top of the dome is not well represented by the structure contours.

Modeling of the Salt Dome Crest

To provide a more precise representation of the geometry of the top portion of the salt domes, drill hole penetrations of the nearly flat-lying top-of-salt surface can be modeled using a conventional geological modeling software package. The MVS modeling software uses a proprietary variant of the standard ordinary kriging algorithm (Journel and Huijbregts, 1978) to model subhorizontal or gently dipping geological surfaces, and the software outputs the resulting undulating surface in a specific variant of the finite-element mesh format (nodal coordinates plus explicit connectivity among nodes). The MVS software provides capabilities for merging two or more sets of meshes for direct visualization within the MVS environment. The software can also be programmed to write-out a file containing the combined meshes. If the crest and flanks of the dome are modeled properly and filtered to avoid overlapping mesh nodes and surfaces, the final output file will consist of a complete salt dome representation in which the two component surfaces merge essentially as a single surface.

For situations where either a "quick-look" is desired or where existing well control is sufficiently sparse or poorly located that the kriging approach seems inappropriate, an alternative modeling approach for the flat-lying top-of-salt surface on the crest of the dome was developed. This methodology simply involves determining the centroid of the highest-elevation digitized structure contour, identifying the difference in elevation between that contour and the next lowest, and closing the connect-the-dots mesh described in item (5), above, to the uniform central point [see also fig. 1(d)]. The elevation assigned to this single point of closure is arbitrarily assigned as one-half of the difference in elevation between the two highest structural contours. The justification for this value is that the actual elevation should be no higher than the full contour-elevation difference (or there should have been another contour represented in the paper model). This alternative approach is coded into the `ctr2evs` program, and is selected as an option at run time.

Discussion of Alternative Modeling Approaches. Conceptually and theoretically, it is apparent that the preferred mechanism for modeling the top of a salt dome would be to use actual, measured elevations for the relevant contact(s) and interpolate/model these in some manner. Such measured elevations will generally be stratigraphic

tops from drilling (well logs, etc.), but might also include depth-converted stratigraphic picks from seismic (2-D or 3-D). Only seismic data are likely to be sufficiently abundant (if they exist at all) to provide high-resolution modeling of the dome crest. Well spacings on top of salt masses, absent site-specific drivers such as the presence of sulphur deposits or Strategic Petroleum Reserve caverns, are generally quite wide. Additionally, wells that were drilled in an attempt to delineate the simple positions of salt domes are generally quite old (straight wildcat drilling for salt having been supplanted by other exploration techniques early in the twentieth century), and the data from these early drilling efforts may be rather unreliable.

Clearly, the more spatially distributed data values that are available from which to create a model, the more geologically accurate will be the resulting model, all else remaining equal. The precise shape of the salt dome crest — and certainly of the top of caprock — may yield important information regarding the interaction of faulting or other evidence of differential movement of salt spines at depth with the subject surface(s). Only approaches using actual geologic data can reveal these types of features and insights. The requirement for "densely spaced" data to identify subtle features is a principal driver toward the use of seismic, and particularly 3-D seismic data. However, seismic data were not available during site characterization for the Strategic Petroleum Reserve, and hence, were not used in constructing any of the four site characterization models

The secondary method for modeling the top-of-salt surface described above, was developed and is presented here as an option for two principal reasons. First, it was extremely simple to implement and it allows a numerical model to be developed based *solely* on the interpretive contour maps and *without reference* to any drill hole data. By itself, this latter attribute may be desirable in future potential applications of the modeling methodology. Second, there may be downstream applications of these salt dome models that focus on the flanks, as opposed to on the crest of the dome. In such applications (for example, investigation of the closest approach of a cavern to the salt-sediment interface at depth), the precise configuration of the top-of-salt surface may be irrelevant and thus not worth the investment of effort in modeling. Note that the actual elevation differences implied between the two modeling approaches are generally very small as a fraction of the modeled vertical extent of the salt domes.

Because the additional logistical and numerical complexity of modeling the top surface of a salt dome in detail is nontrivial, and because the resulting numerical model must necessarily be larger (simply in terms of file size) to contain that detail, the simplistic method of simply closing the top surface to an arbitrary-but-representative location may be an attractive alternative. Additionally, future updated modeling of SPR and other salt domes (as envisaged for the SPR project) will probably require the use of interpretive structure contour techniques. The closure-to-centroid modeling algorithm will allow rapid visualization of potentially many such interpretive models and the selection (based upon expert judgment) of the one(s) that presumably reflect the actual subsurface geology. A detailed (kriged) model of the top-of-salt surface may then be substituted for the arbitrary crestal configuration once the best representation of the salt dome flanks has been determined.

RESULTS

This section presents the results of modeling the existing salt-dome models as captured in the original site characterization reports and in selected updates to those original site characterization investigations. Reasons for converting or not converting a specific model are discussed in the relevant section for each SPR site.

The presentation of results by site is somewhat involved. First, in general there are models corresponding to both the original characterization report and an update characterization report, dated roughly 10 years later (table 1). Second, there are two methods available for modeling the uppermost, crestal portion of the salt dome, as discussed in the section on *Modeling of the Salt Dome Crest*. Although in the larger scheme of things, the amount of relief on the flat-lying portion of a salt dome is minimal in comparison to the total amount of structural relief exhibited by the known extent of the diapir, the two different modeling approaches yield rather different detailed configurations of the top of salt. Both methods are typically presented for illustrative purposes. Unless there are compel-

ling reasons otherwise, the "best" model should always be considered to be the one that makes use of the maximum geologic data.

The salt dome models are presented in a consistent manner and to a common depth across all sites, with additional illustrations and discussion as appropriate to local conditions. Note, however, that the West Hackberry salt dome is significantly larger than the other three domes in horizontal plan view. The views for this dome are affected accordingly. Four principal views are presented: one each nominally from the east, south, west, and north. The views are perspective in nature, and reflect a consistent viewer elevation of 20 degrees above the horizontal. To assist in conveying the perspective view, the actual angle of the view associated with each cardinal direction is actually from 15 degrees less (in azimuth) than the nominal direction. Thus, the southerly view is actually from an angle of 180–15 = 165 degrees. Structure contours have been placed on the salt surface at selected depth increments to aid in the visualization.

BAYOU CHOCTAW SITE

The principal configuration of the Bayou Choctaw salt dome is presented in figures 2 and 3. The model is based upon the original site characterization report (table 1: Hogan, 1980a). Figure 2 presents the salt-dome top as modeled using actual drill hole data, whereas figure 3 presents the model generated simply by closing the top of the dome arbitrarily to the centroid of the highest structural contour.

The graphical visualizations of the Bayou Choctaw dome represent the salt diapir as a relatively cylindrical mass that gradually expands laterally at depths below approximately 5,000 ft. The dome exhibits overhangs on the southern and western sides. Data are available to delineate the dome to a depth of roughly 9,000 ft below sea level, although Hogan (1980a) presents partial structure contours to a markedly greater depth on the northwestern flank of the dome. These depths are generally below the zone of SPR interest. Departures from a mostly circular plan view are minimal at shallow depths, but the dome exhibits a somewhat elongated, rounded point to the south at depths greater than 7,000 ft below sea level. A slight reentrant is also evident on the western flank of the dome beginning at approximately this same depth.

The highest structural levels of the Bayou Choctaw dome exhibit somewhat more complex contour patterns, suggestive of shallow "valleys" on the top of salt (see Hogan, 1980a, fig. 6-1) on both the northwestern and southeastern margins of the top of salt. Because these features were presented by Hogan as supplemental structure contours at 100-ft vertical intervals, they have been captured and are represented in the numerical models of the Bayou Choctaw dome in both figures 2 and 3. Hogan's figure 6-1 indicates sufficient well control from the numerous caverns developed in this dome to suggest that these are likely to be real geologic features.

Differences in the detailed form of the dome between the two alternative modeling methods (merged-surfaces vs. closure-to-centroid) are minimal because of the high-resolution structure contours and because the highest two contours are separated by a mere 100 ft. vertically. Reference to figure 3 suggests that the closure on the highest portion of the dome is somewhat more circular in form than indicated by the modeling of the many actual well penetrations. These differences are inferred to be of little practical consequence at this time, given that we have not evaluated the spatial configuration of the available well control nor the quality of the "picks" for the top-of-salt surface. In a more thorough remodeling and evaluation of the Bayou Choctaw dome, the "valleys" suggested in figure 2, plus the elongation of the uppermost closed contour may be interpretable in terms of differential salt movements (see also next paragraph).

Note that only the salt dome model corresponding to the original site characterization report (table 1: Hogan, 1980) has been included in this document. Neal et al. (table 1: 1993) presented a slightly revised structure contour map of the Bayou Choctaw salt dome (their fig. 6B). However, the differences concerning the main vertical extent of the dome are virtually negligible, with most of the variation involving the introduction of faulted displacements of the top-of-salt surface. Because introduction of discontinuities in a modeled surface creates various numerical problems in an interpolation-type algorithm, we decided not to pursue creation of an updated numerical representation at this time, given the relatively small changes in the con-

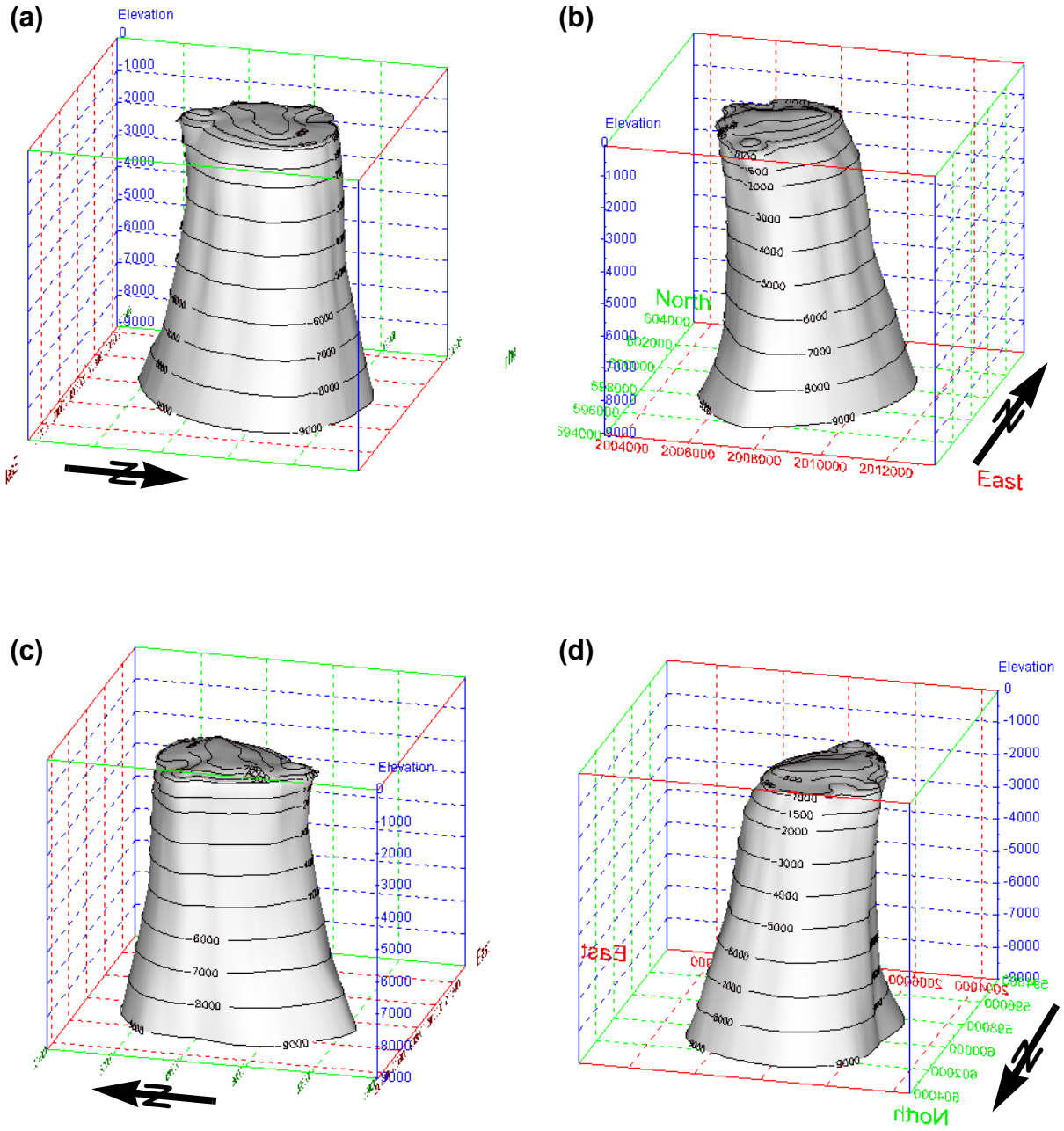


Figure 2. Four representative views of the Bayou Choctaw salt dome, as generated by the two-part, merged-surface technique for the original site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration.

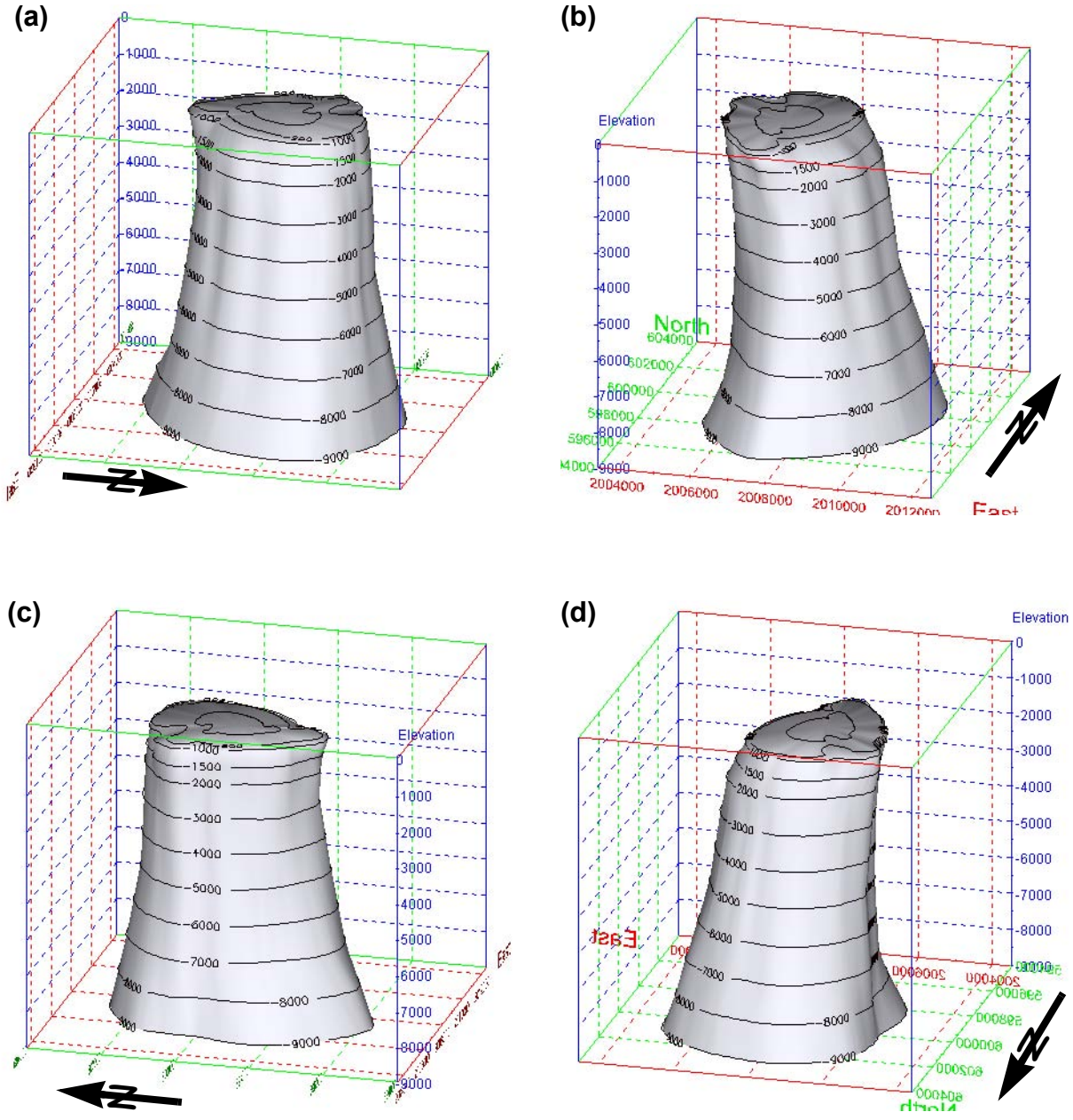


Figure 3. Four representative views of the Bayou Choctaw salt dome, as generated by the one-part, closed-to-centroid technique for the original site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration.

figuration of the top of salt between the two characterization reports. Rather, effort has been dedicated to creating an original and an updated numerical model for those sites where the overall configuration of the salt dome changed to a larger degree in the site characterization update report. Resolution of these additional details regarding implications for faulting on the top of the Bayou Choctaw dome is beyond the scope of this report. Additionally, a primary driving factor in the preparation of this report was documentation of the modeling algorithm itself, together with presentation of representative illustrations of the algorithm's results.

BIG HILL SITE

The geologic configuration of the Big Hill salt dome is presented in figures 4 and 5. The models are based on the original site characterization report (table 1: Hart et al., 1981), but selected information from the site characterization update report (table 1 (Magorian and Neal,1988) have been incorporated as described below. The model generated by creating and merging two separate geologic surfaces is shown in figure 4, and the model constructed simply by closing the top of the dome to the centroid of the highest structural contour is presented in figure 5.

These visualizations present the Big Hill salt dome as a mass of relatively circular plan view, but one for which the cylindrical shape is tilted strongly towards the south. Significant salt overhang exists on this southern margin, along with a rather pronounced smaller protuberance to the south. This particular portion of the Big Hill dome is relatively well characterized by numerous well penetrations in a well developed oil field.

In contrast to the well-characterized nature of the southern and southwestern flanks of the Big Hill dome (some 50-plus wells locally), the top of the salt mass is defined essentially only by wells associated with the 14 SPR caverns, the remainder of the few wells (perhaps a dozen) over the top of the dome having been shallow sulphur exploration tests that typically did not penetrate to the depth of the top of salt. A consequence of this relatively sparse but locally concentrated well coverage in the model shown in figure 4 is that the top of salt on the dome is represented as a very shallow basin (fig. 4). In other words, the salt penetrations observed in the cavern wells indicate the top of salt at a slightly *greater* depth (between roughly 20 to 50 feet) than the elevation of the highest structural contour inferred in the site characterization report (at 1,500 ft below sea level). Whether this is an accurate reflection of the actual geology is unknown because the structure-contour map is interpretive, but in any event the central "depression" implied by the inferred structure contour and the Big Hill cavern wells is quite shallow.

The precise opposite effect is visualized in figure 5, wherein the top of salt is generated by identifying the centroid of the –1500-ft structural contour and simply (and arbitrarily) closing the modeled surface to this location and at an elevation one-half of the last contour interval higher. In this case because of the coarse structure-contour interval used in the site characterization report of Hart et al. (1981), the dome-top elevation is represented at 1,250 ft below sea level, in contrast to the known cavern intersections of the top-of-salt surface in roughly the same horizontal locations of some –1,550 to –1,600 ft. The implication is that the –1,250 -ft dome-top elevation is purely an artifact of the simplified, non-geology-based modeling technique, and that it is preferable to use the approach based on actual geologic data when it is available (see next paragraph). Figure 6 shows the differences in the configuration of the top-of-salt surface produced by the two different techniques. For both cases, however, the broad geometric forms suggest that the top of the Big Hill dome is not particularly well characterized for details. Compare, either of figures 4 or 5 to the inferred more detailed configuration of the Bayou Choctaw dome in figure 2.

It is important to note that the 14 SPR caverns at the Big Hill site were leached *after* publication of the original site characterization report of Hart and others (1981). Therefore, the 14 "picks" for the top of salt in the center of the Big Hill dome (actually, there are 28 physical wells, because each cavern comprises two closely space drilled holes) that were used to produce the model of figure 4 were not available to generate the paper model of Hart and others. Thus, the only possible conversion of the Hart and others model is that represented by figure 5, which was generated by closing the crest of the dome to the centroid of the uppermost pub-

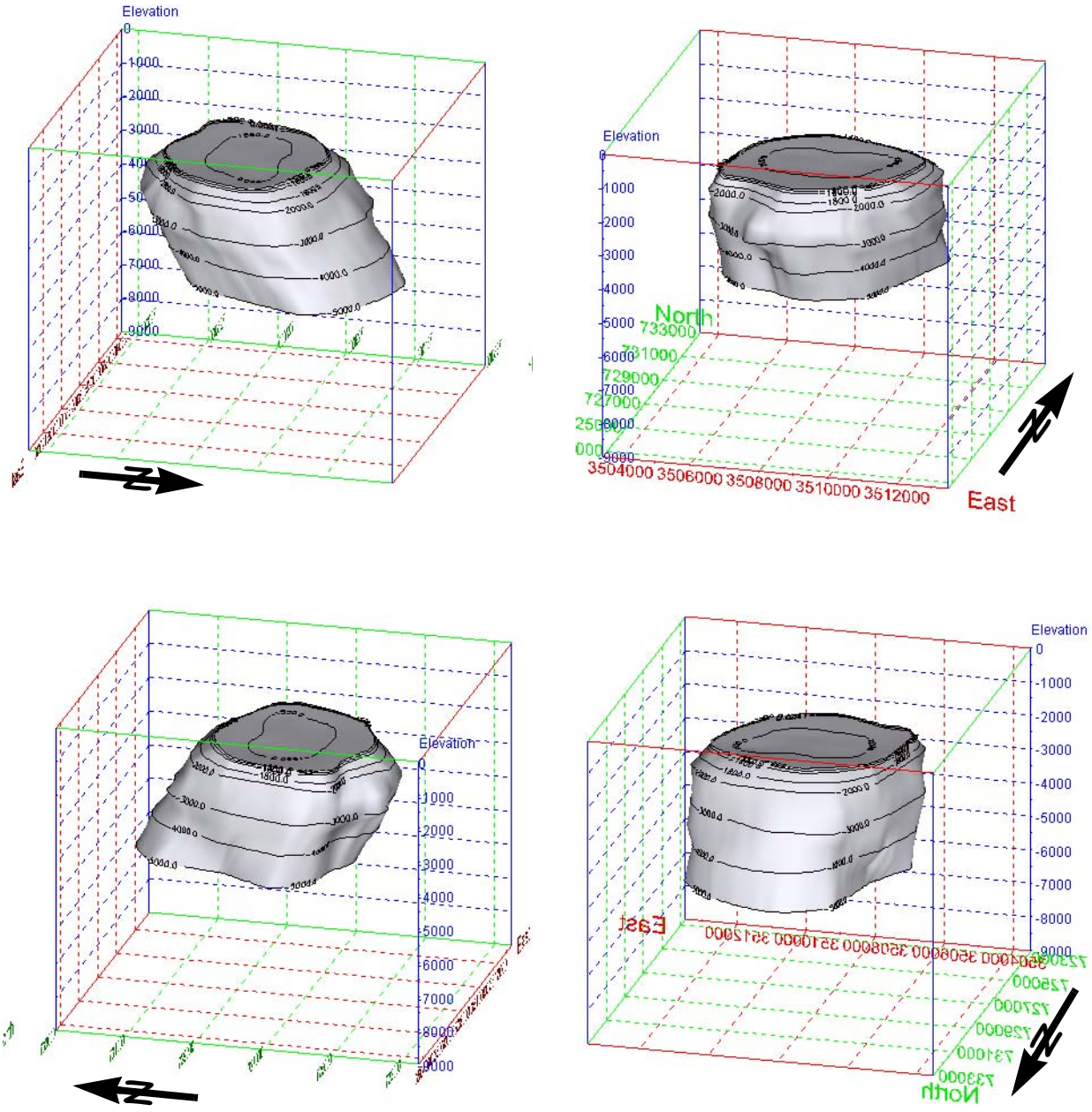


Figure 4. Four representative views of the Big Hill salt dome, as generated by the two-part, merged-surface technique for the original site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration.

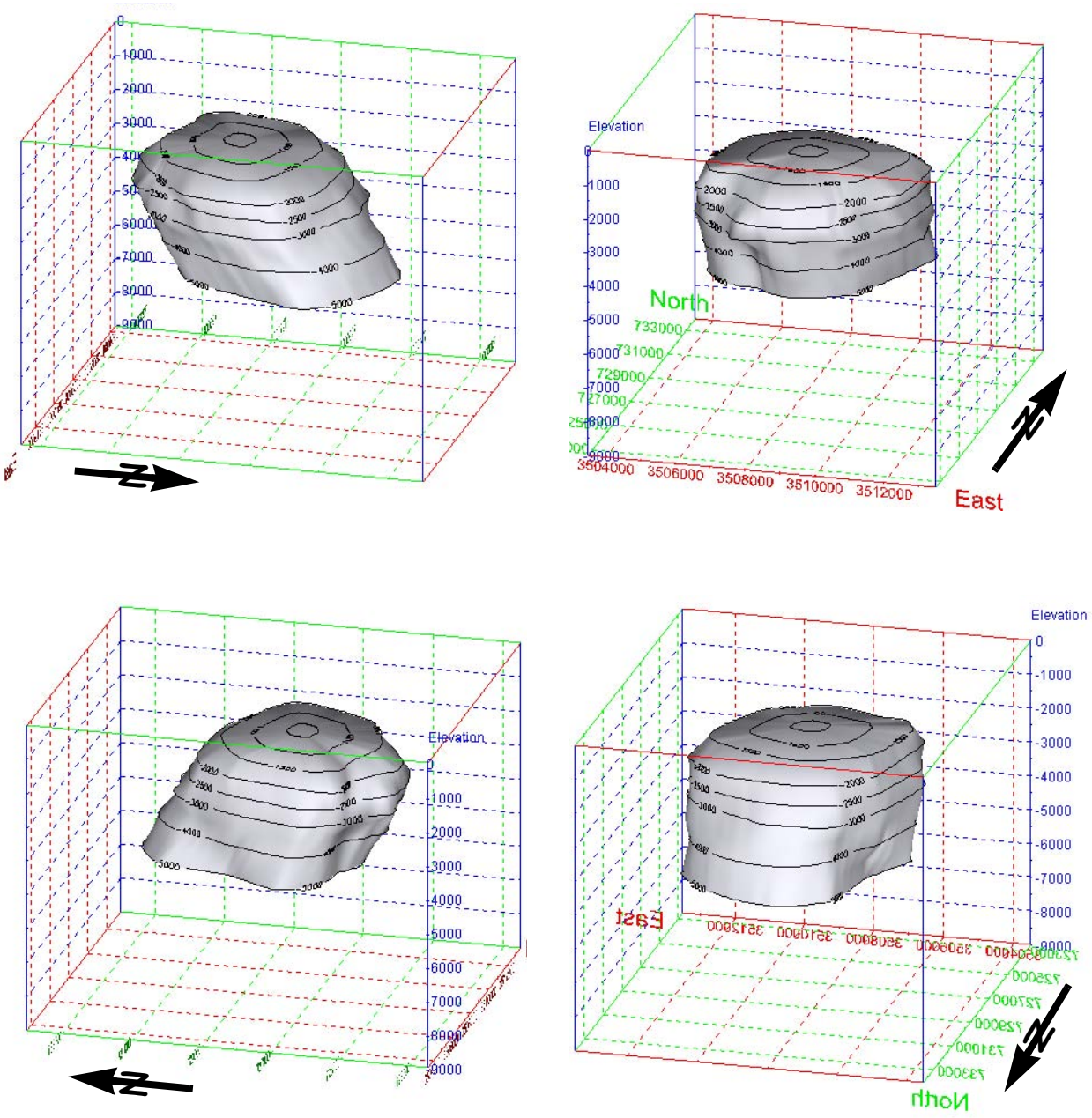


Figure 5. Four representative views of the Big Hill salt dome, as generated by the one-part, closed-to-centroid technique for the original site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration.

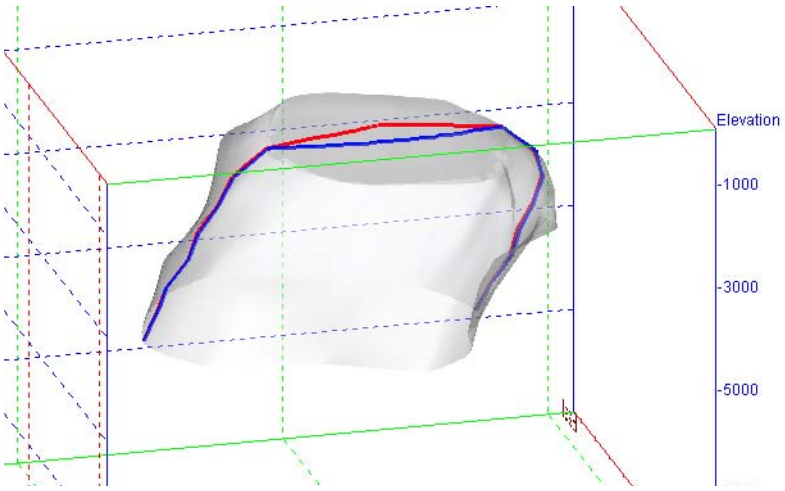


Figure 6. Comparison of mid-crest cross sections produced by the two alternative modeling techniques for the Big Hill salt dome. The blue profile is cut through the two-step, merged geologic surfaces model, shown completely in figure 4, and shown in the ghostly transparent shape. The red profile is cut at the same location through the single-step, closure-to-centroid geologic model, shown completely in figure 5. The "basinal" configuration of the model in figure 4 vs. the "domal" configuration of the model in figure 5 is clearly represented. The elevation shown as blue profile is always below that of the red profile.

lished structure contour. A geologically based model of the dome crest (i.e., fig. 4) is possible only for the site-characterization update model based on the additional information contained in Magorian and Neal (1988). This is one example of the utility of and necessity for the short-cut algorithmic approach described on page 11.

This difference illustrated by the two different modeling approaches for the Big Hill dome is probably a generally relevant conclusion. Measured data incorporated into a model should take precedence over an arbitrary assumption. Exceptions to this rule may occur if the specific locations of the "measured data" on the crest of a salt dome are sufficiently biased spatially that numerical artifacts result from the "blind" application of a particular mathematical algorithm. For this reason, it is good practice to examine the configuration of the top-of-salt surface generated by both approaches and to judge the geological relevance of both models.

BRYAN MOUND SITE

The geometric configuration of the Bryan Mound salt dome is presented in figures 7 through 10. Figures 7 and 8 represent the structural model contained in the original site characterization report (table 1: Hogan, 1980b), whereas figures 9 and 10 capture the updated model contained in the site characterization update report by Neal et al. (table 1: 1994). There are a number of differences between the models, not the least of which is that the updated version (figs. 9 and 10) extends the depth extent of the salt diapir 2,000 additional feet to a total depth of 7,000 ft below sea level. Figures 7 and 9 represent the merged, two-surface modeling approach, whereas figures 8 and 10 represent the simpler closure of the dome top to the centroid of the highest available structure contour.

Comparison of the drill-hole-based domal top-of-salt surfaces shown in figures 7 and 9 illustrates the apparent differences in domal configuration. Because the highest reliable structure contour available in the site characterization model was at −1,500 ft, a fairly large portion of the areal extent

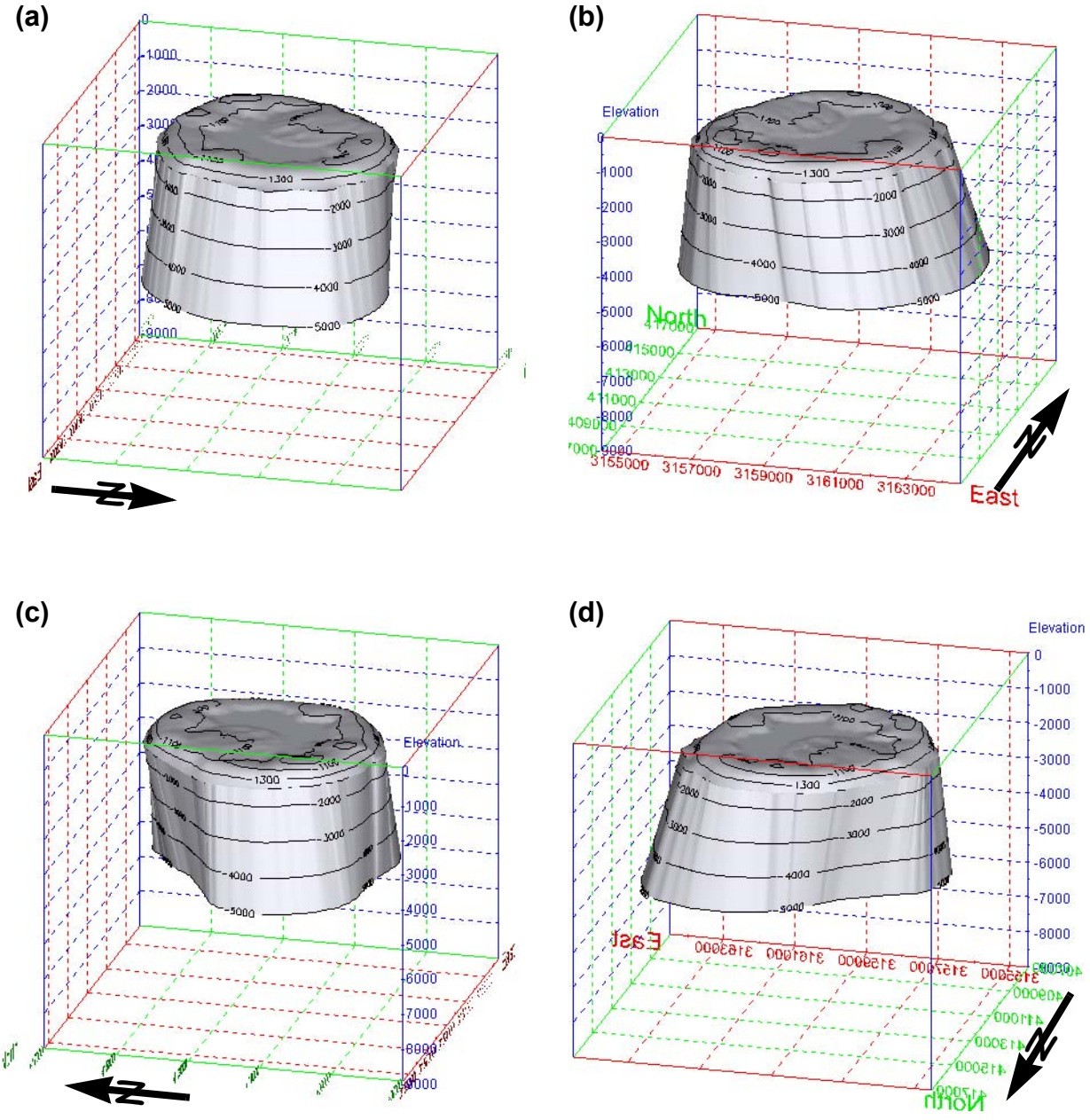


Figure 7. Four representative views of the Bryan Mound salt dome, as generated by the two-part, merged-surfaces technique for the original site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration.

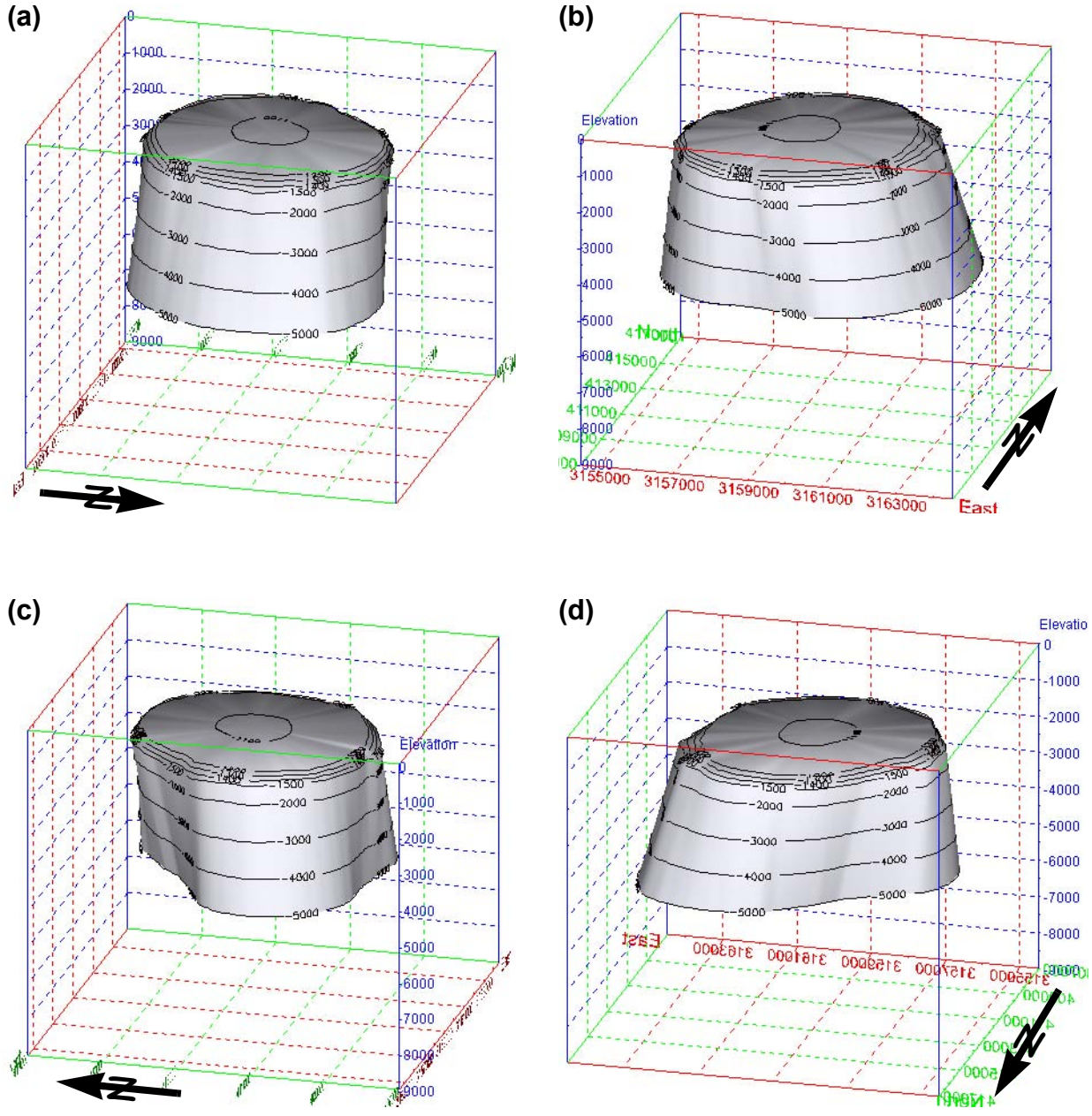


Figure 8. Four representative views of the Bryan Mound salt dome, as generated by the one-part, closed-to-centroid technique for the original site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration.

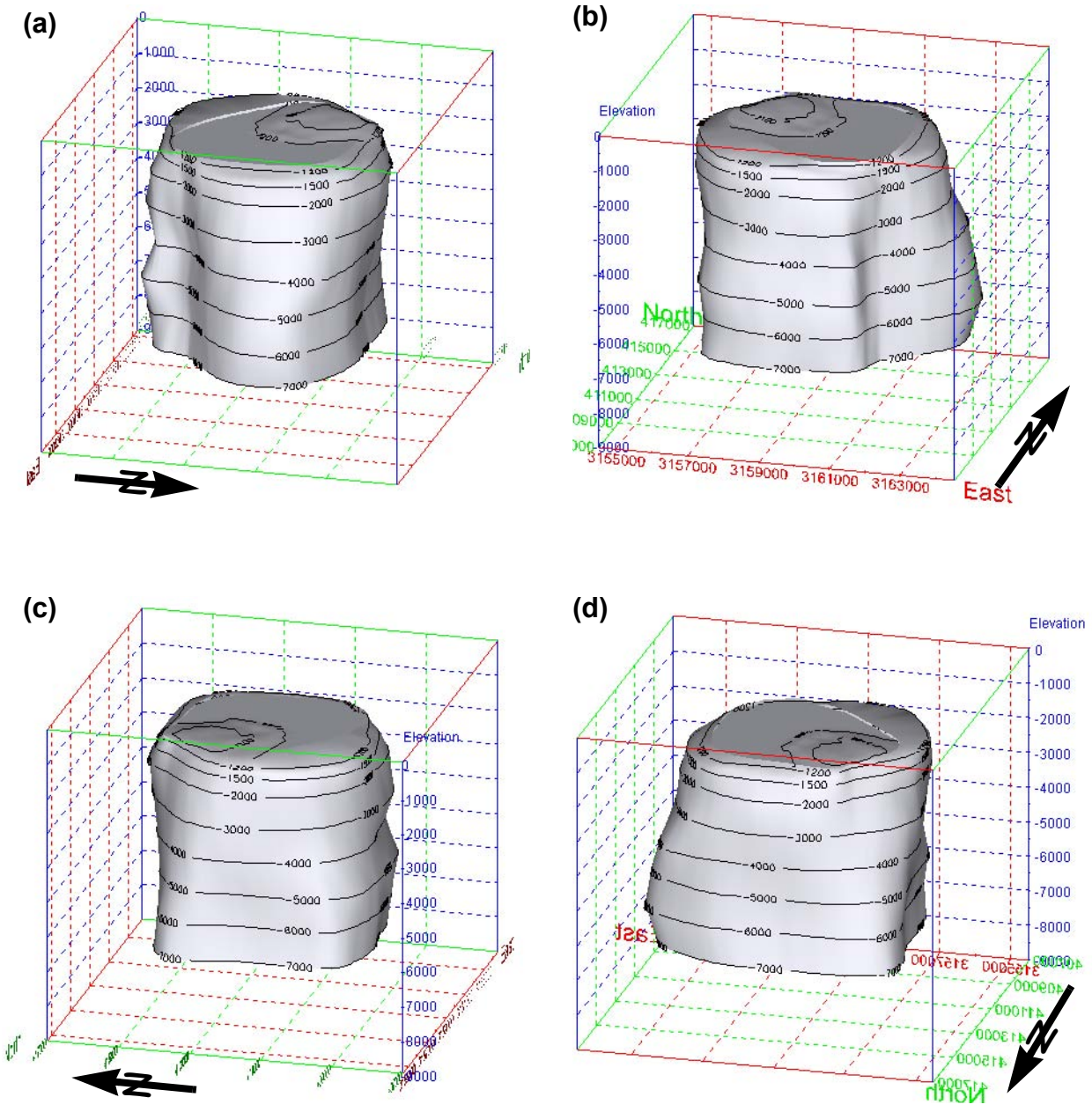


Figure 9. Four representative views of the Bryan Mound salt dome, as generated by the two-part, merged-surfaces technique for the updated site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration.

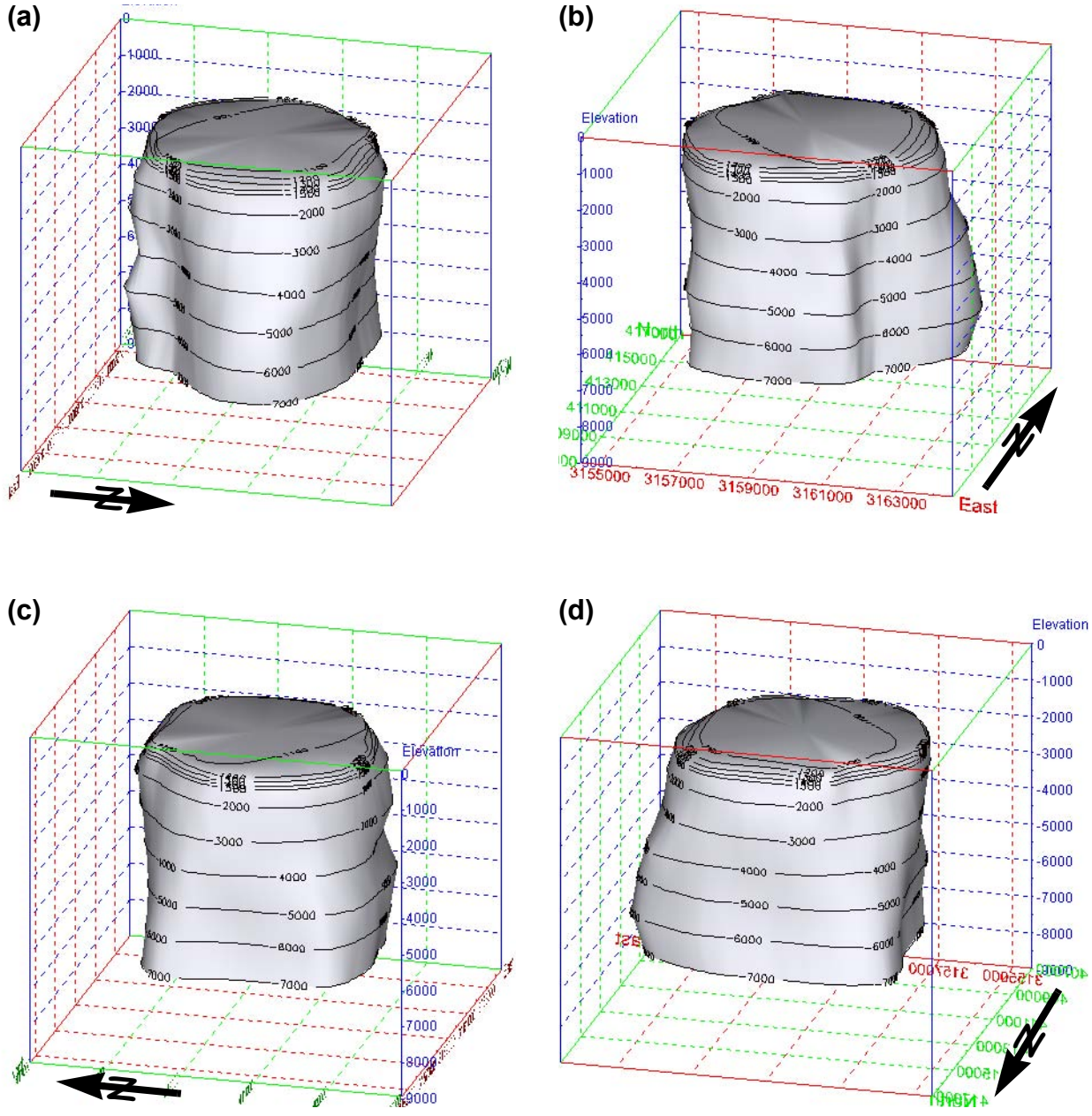


Figure 10. Four representative views of the Bryan Mound salt dome, as generated by the one-part, closed-to-centroid technique for the updated site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration.

of the dome was modeled using well control (and the digitized points lying along the –1500-ft contour to ensure accurate matching of the two surfaces). For the updated model, shown in figure 9, the highest reliable structure contour was judged to be the –1200 ft contour, which is located significantly inboard of the –1500-ft contour in the earlier model. Thus, essentially the only well control used to model the crestal portion of the dome in the updated site characterization model are the various cavern wells (plus the digitized –1200-ft contour points). In contrast, for the original site characterization model, the upper "flat" top-of-salt surface is much less well controlled, given that the same cavern-well salt elevations are available, and the resulting model appears much more strongly dominated by the points associated with the minus-1,500-ft structure contour.

Comparison of the two arbitrary-closure models of figures 8 and 10 reflect the obvious limitation of this particular modeling technique. Particularly in comparison to the well-contact-based models of figures 7 and 9, the arbitrary closure of the dome to the centroid of the last available structure contour inevitably places the highest point on the dome at essentially the center of the near-circular salt mass. This is a quite different configuration than that suggested by the modeling of the actual top-of-salt penetrations in the available wells, which suggest an irregular basinal configuration in the original site characterization model and an ill-defined elevated region in the northwestern portion of the dome in the site characterization update model. Again, the configuration and reliability of the geologic data used to create the kriged top-of-salt surfaces for both the original and updated site characterization models has not been evaluated as part of this modeling effort. Nevertheless, the suggestion of subtle geometric features — and the differences between those features as illustrated by figures 7 and 9 — indicates that updated geologic data and more refined future modeling may contain important information regarding possible differential internal salt movements in the Bryan Mound dome.

A comparison of the "footprint" of the Bryan Mound salt dome at various depths for both the original and updated site characterization models is presented in figure 11. The comparison is in the form of a series of level plans at elevations of 1500, 2500, 3500, and 4500 ft below sea level. This is essentially the vertical zone of overlap of the two models. Comparisons at depths shallower than about 1500 ft below sea level would emphasize only the differences in the methods used to generate the top of salt. In contrast, the differences in the horizontal positions of the near-vertical salt-dome flanks represent the major differences between the original and the updated models (table 1: Hogan, 1980(b), Neal et al., 1994).

The differences in the salt outlines near the top of the dome are quite minor and represent relatively small interpretive differences between the two site characterization reports. The fact that the –1,500 -ft contour for the original site-characterization model [shown in blue in fig. 11(a)] is indicated by an intermittent line reflects an artifact that this contour is precisely at the position of the merger between the two different surfaces used to generate the model itself.

The differences in position of the salt dome flanks at the –3500 and –4500 ft levels are more significant — from approximately 200–500 ft [fig. 11(c)] to *as much as 1,000 ft* laterally [fig. 11(d)]. Because we did not investigate precisely what additional control(s) were used to influence the interpretive modeling for the site characterization update report [the contours shown in red in figs. 11(c) and (d)] as part of this modeling effort, it is not possible to make definitive statements regarding the degree of confidence that can be placed in either model at these greater depths. A systematic reevaluation of the drilling control used to construct the site-characterization models (and their updates) was not undertaken as part of this model-*conversion* exercise. Clearly, however, the precise position of the salt-sediment interface for the Bryan Mound salt dome is quite uncertain at depths corresponding to the approximately lowermost extent of the oil-storage caverns. The newer site-characterization update studies that are now being undertaken (in fiscal years 2003 and beyond) will address the type(s) (e.g., drill-hole based vs. seismic) and extent of geologic control available to constrain the margin of the salt mass at various depths and horizontal positions.

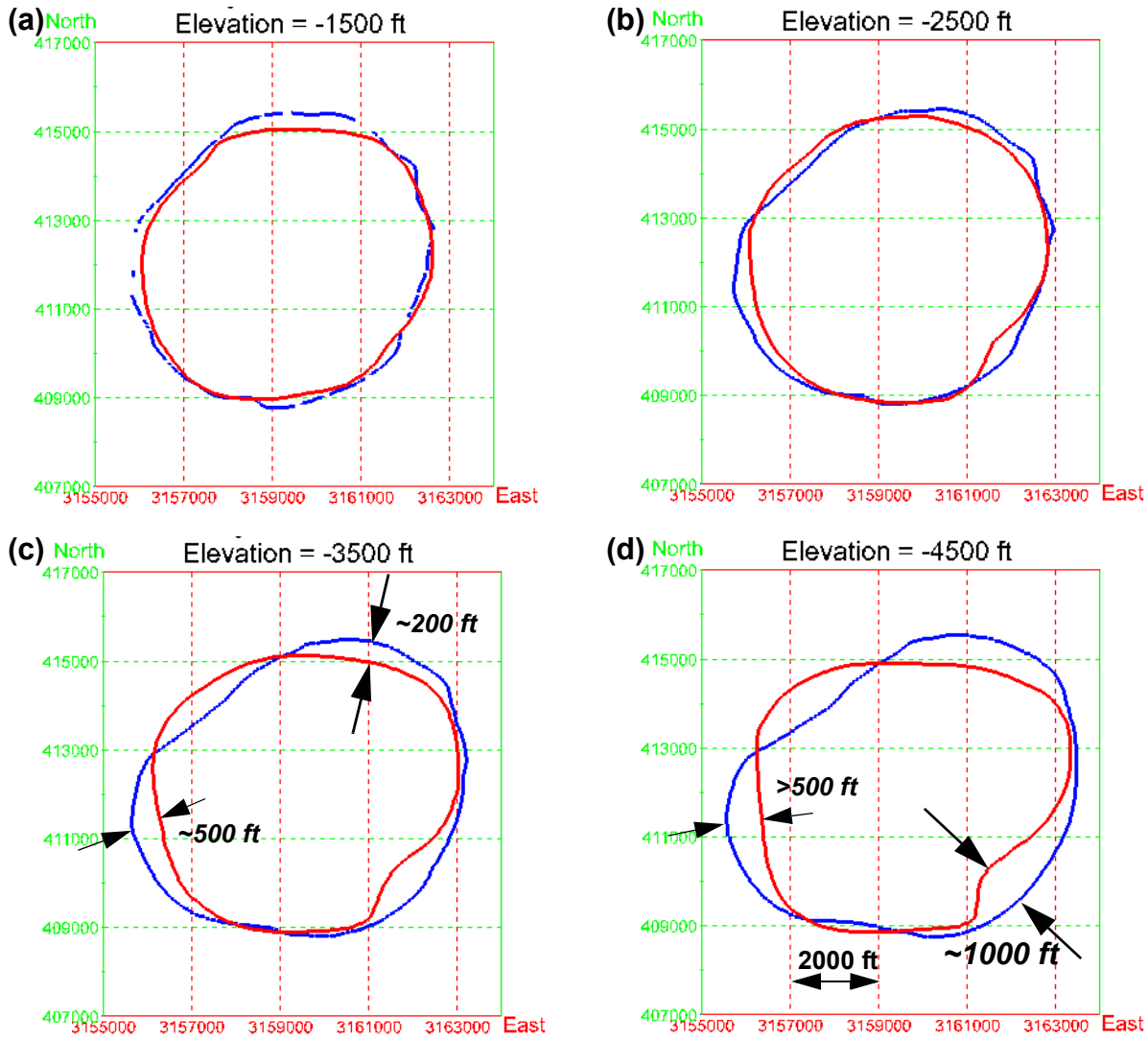


Figure 11. Comparison in plan view of the original (blue) and updated (red) site characterization models for the Bryan Mound salt dome. Elevations shown are below sea level. Note the increasing discordance between the two models at greater depths where there is greater uncertainty as to the precise position of the salt margin. View is from directly overhead.

WEST HACKBERRY SITE

Visualizations of the salt dome hosting the West Hackberry SPR site are presented in figures 12 through 15. The original site characterization model versions (table 1: Whiting, 1980) are presented in the first two figures, numbered 12 and 13; figures 14 and 15 are associated with the updated version of Magorian et al. (table 1: 1991). The sequence of presentation of the modeling methodologies for the top of the salt dome is similar, with figures 12 and 14 representing the two merged surfaces and figures 13 and 15 representing the closure-to-centroid approach.

A major difference between the West Hackberry salt dome and the domes at the other SPR sites is that the dome that hosts West Hackberry represents a much larger mass of salt. In fact, the east-west dimension of the West Hackberry dome is almost twice as large as any of the others. The

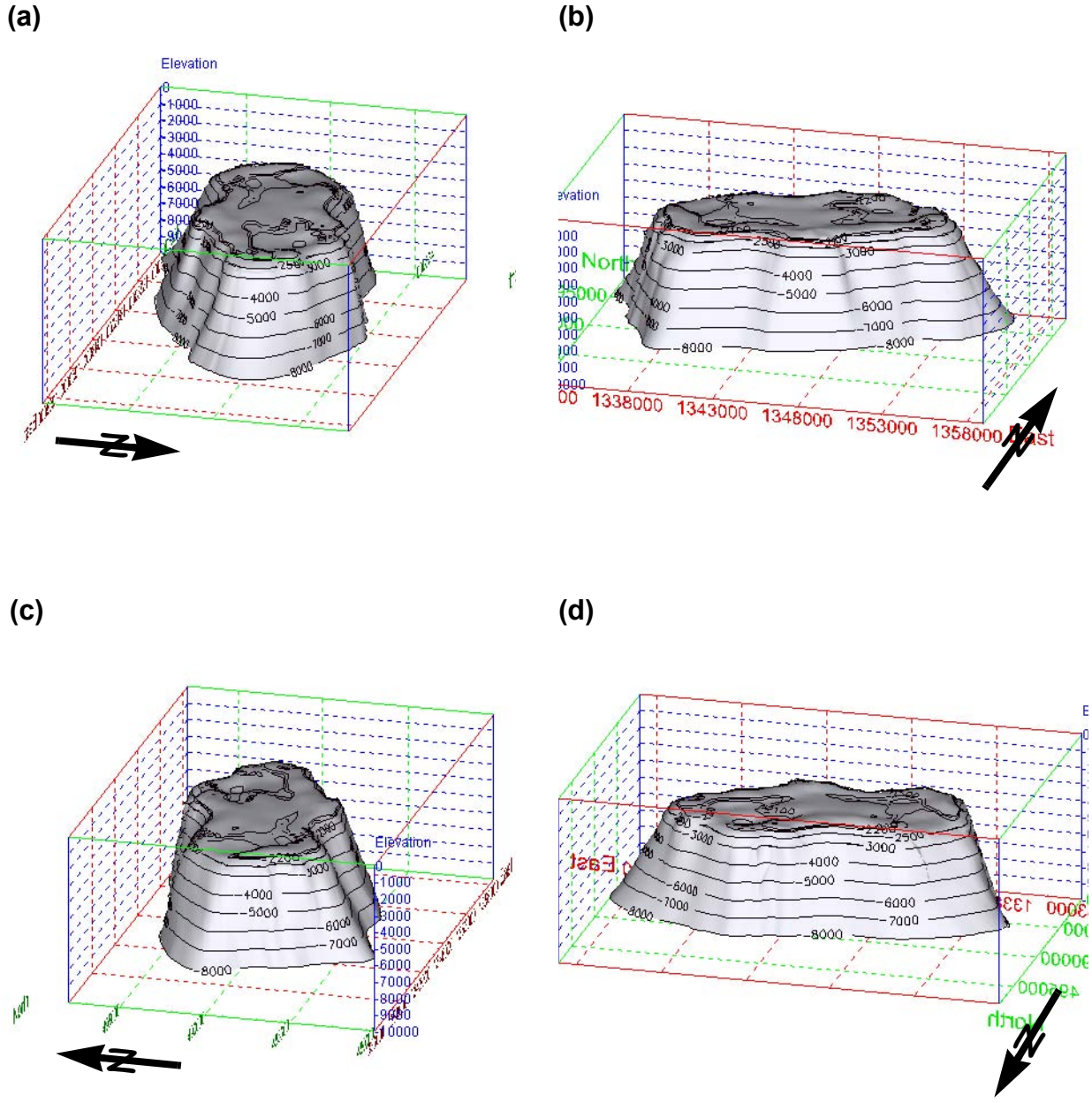


Figure 12. Four representative views of the West Hackberry salt dome, as generated by the two-part, merged-surfaces technique using the original site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration. Note the change in horizontal dimensions from the views of the other salt domes.

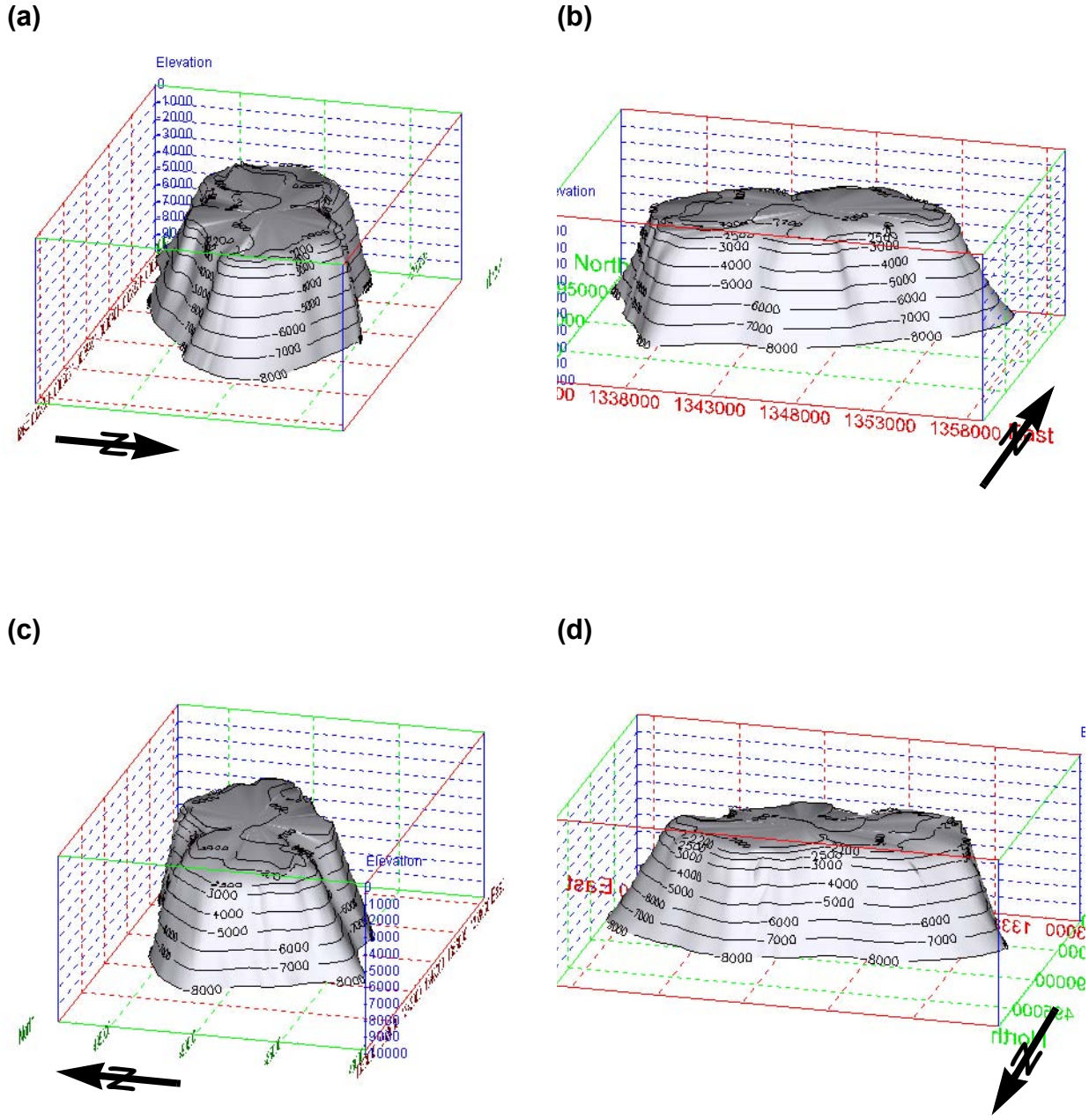


Figure 13. Four representative views of the West Hackberry salt dome, as generated by the one-part, closed-to-centroid technique using the original site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration. Note the change in horizontal dimensions from the views of the other salt domes.

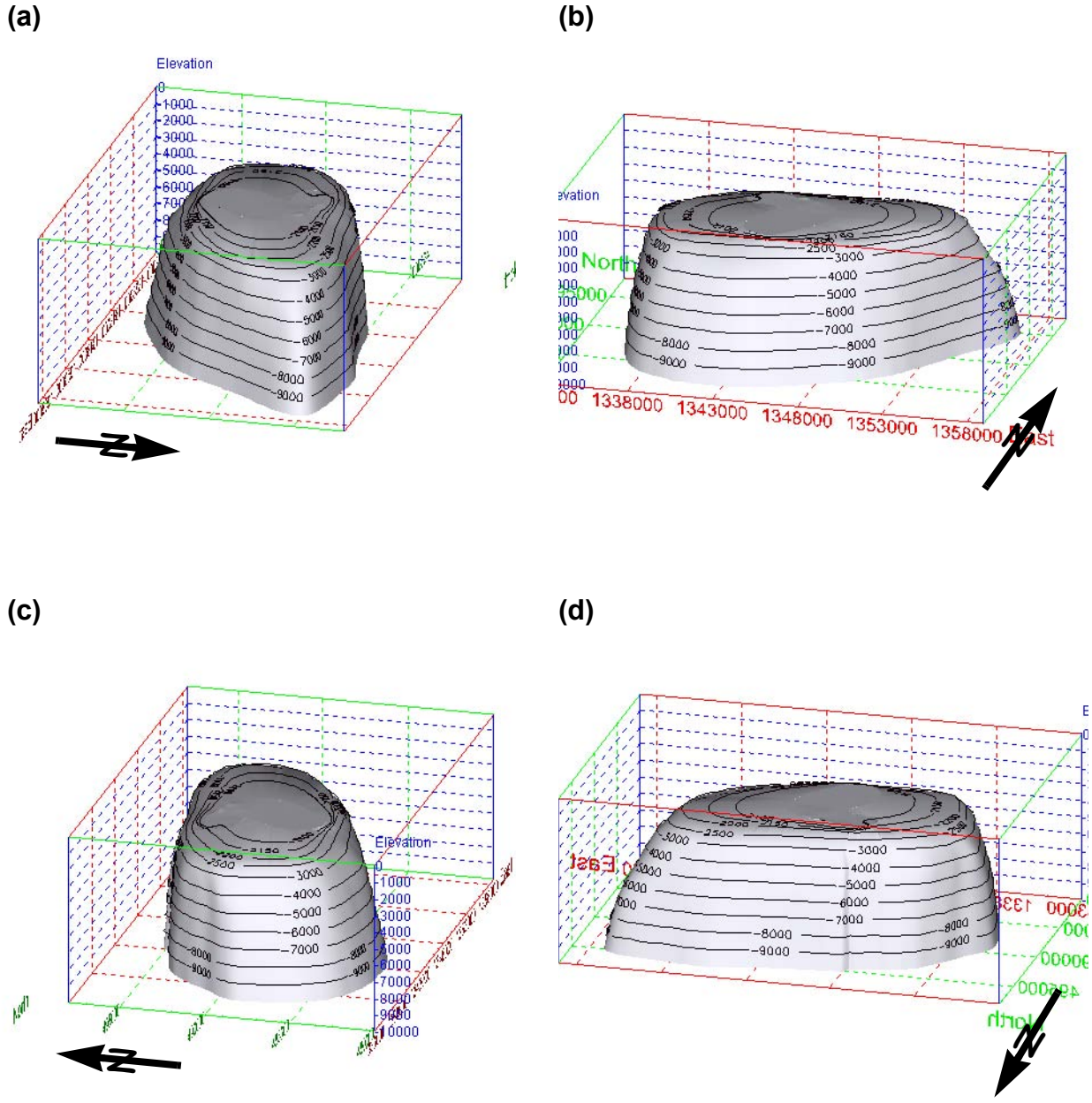


Figure 14. Four representative views of the West Hackberry salt dome, as generated by the two-part, merged-surfaces technique using the updated site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration. Note the change in horizontal dimensions from the views of the other salt domes.

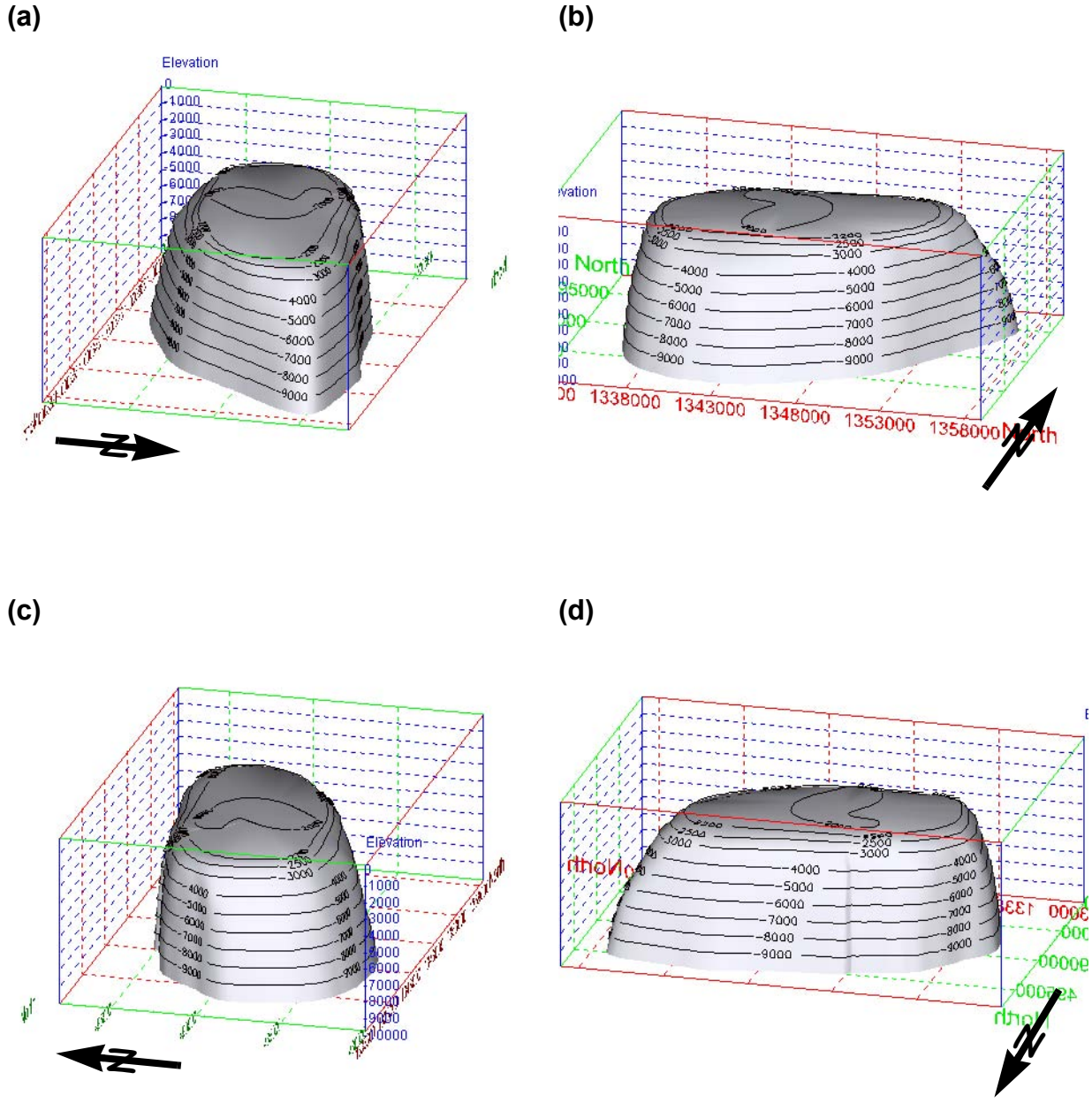


Figure 15. Four representative views of the West Hackberry salt dome, as generated by the one-part, closed-to-centroid technique using the updated site characterization data. (a) azimuth = 75°; (b) azimuth = 165°; (c) azimuth = 255°; (d) azimuth = 345°; view is from 20° above the horizontal. No vertical exaggeration. Note the change in horizontal dimensions from the views of the other salt domes.

West Hackberry dome is also far less circular in plan view, being almost twice as long as it is wide. The flanks of the West Hackberry dome, at least according to the original site characterization report, are much more irregular than those of any of the other SPR salt domes. These vertical crenulations are much reduced in the updated version; compare figure 12 versus figure 14, for example. The dome also lacks salt overhangs, according to both the original site characterization and site characterization update interpretations.

The upper, flat-lying top-of salt for this dome is different in detail between the models constructed using the merged-surfaces approach and the closure-to-centroid approach, as expected. However, the absolute differences implied in the elevation of the top of the salt are minimal.

Figure 16 presents a comparison of the original and updated site characterization models of the flanks of the West Hackberry salt dome, similar to the comparison of figure 11 for the Bryan Mound site. Only selected structure contours at 1,000-ft depth increments are presented. However, for the West Hackberry site, the dome has been modeled during both phases (table 1) of the site characterization to a depth of 8000+ feet.

In similar fashion to the Bryan Mound dome, the salt outlines of the two different-vintage models at the West Hackberry site are generally more similar at shallower depths and more divergent at greater depths where the intensity of drilling is less. However, there are two major differences between the two models. First, the original site characterization model (table 1: Whiting, 1980) appears to have attempted to represent a greater degree of complexity in the margin of the salt, whereas the updated site characterization model (table 1: Magorian et al., 1991) appears to have been modeled either on a much more generalized basis or with an overt effort to filter what may have been believed to be noise in the data. The overall shape of the updated dome, as shown in red in figure 16, exhibits a relatively smooth outline, whereas the original model, shown in blue, exhibits much more surface complexity. Second, the sharp eastern extension of the dome — which is, in fact, substantially east of the SPR facilities of primary interest — has been revised extensively. The more recent interpretation of Magorian et al. (1991), the extreme eastern "tip" of the West Hackberry dome points consistently east-northeast instead of being "bent" towards the southeast as in the earlier interpretation. No attempt has been made during this model-conversion process to evaluated the well control and data underlying this difference. However, this difference appears to be one of the more potentially significant deviations between the original site-characterization reports and the updated versions.

Discussion of Visualization Technology

Computer technology has changed significantly the ability to *visualize* a three-dimensional model. Examples of such visualizations are shown in the figures presented in the *Results* section of this report for the four active SPR salt domes. The computer-based visualization technology allows viewing of the identical model from many different perspectives, as well as the extraction of only portions of the model, such as the plan-view outlines presented, for example, in figure 16.

However, a computer-based three dimensional model may be "viewed" in the same manner as a more classical paper-based geologic model. For example, consider figure 17. Part (a) of the figure reproduces (via a scanned image) the structure contour map of the top-of-salt horizon presented in Magorian and others (1993, fig. 7) for the West Hackberry salt dome. This representation is essentially "the" model of the West Hackberry dome. Part (b) of figure 17 is the same type of view — a structure contour map on the top of salt — extracted (visualized) by the current computer software. Comparison of the two illustrations indicates that the computer model has reproduced the original figure/model essentially as-is.

That the computer version of the structure contour map for the West Hackberry SPR site is essentially identical to the original paper version is not surprising, as the digitized structure contours from figure 17(a) are, in fact, the raw material from which the model of figure 17(b) was constructed. However, alternative visualization of the same model, such as the four different views shown in figure 15, more intuitively present the subtleties of the flanks of the West Hackberry salt dome in a way not possible from the straightforward structure contour map of the original.

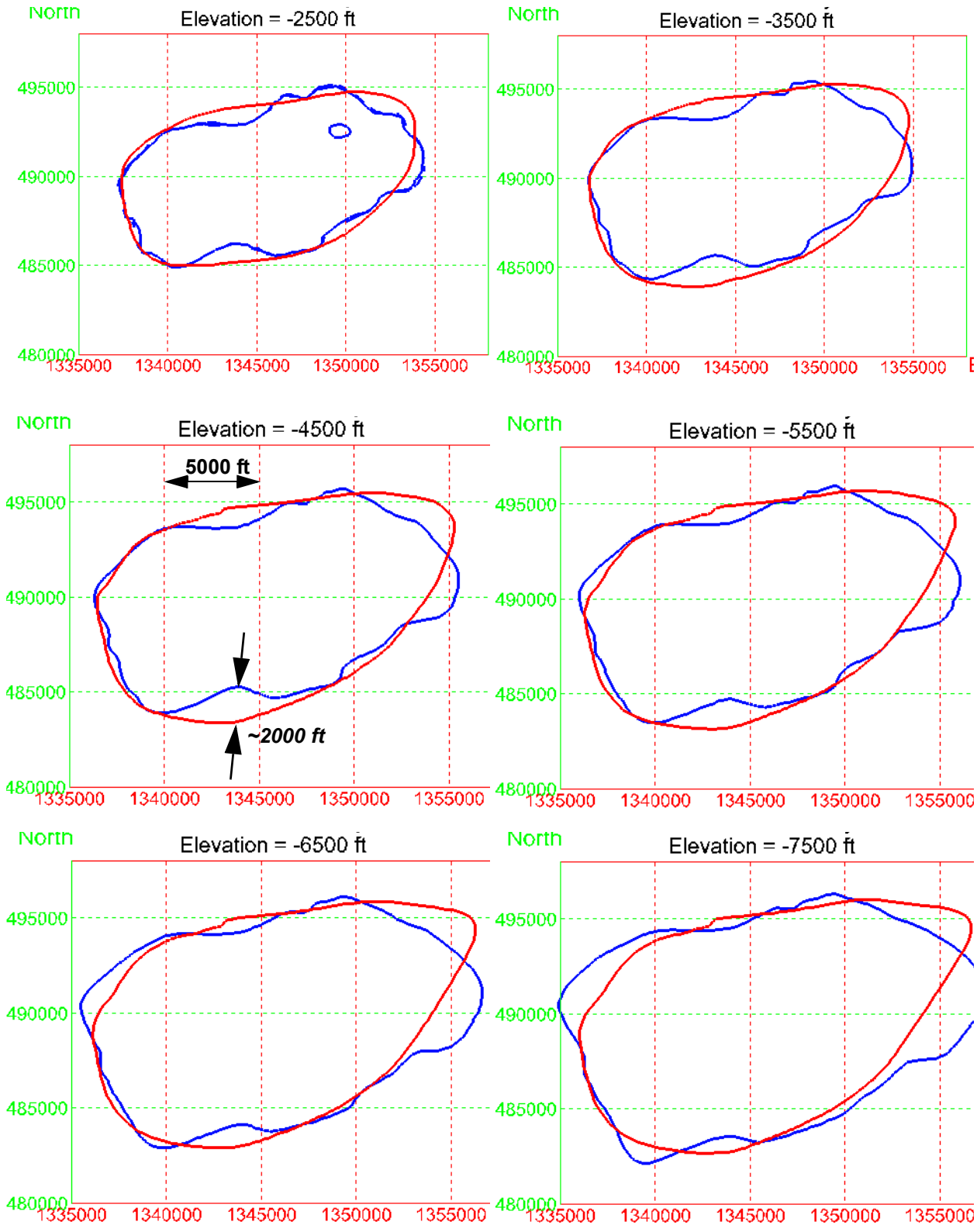


Figure 16. Comparison in plan view of the original (blue) and updated (red) site characterization models for the West Hackberry salt dome. Elevations shown are below sea level. View is from directly overhead.

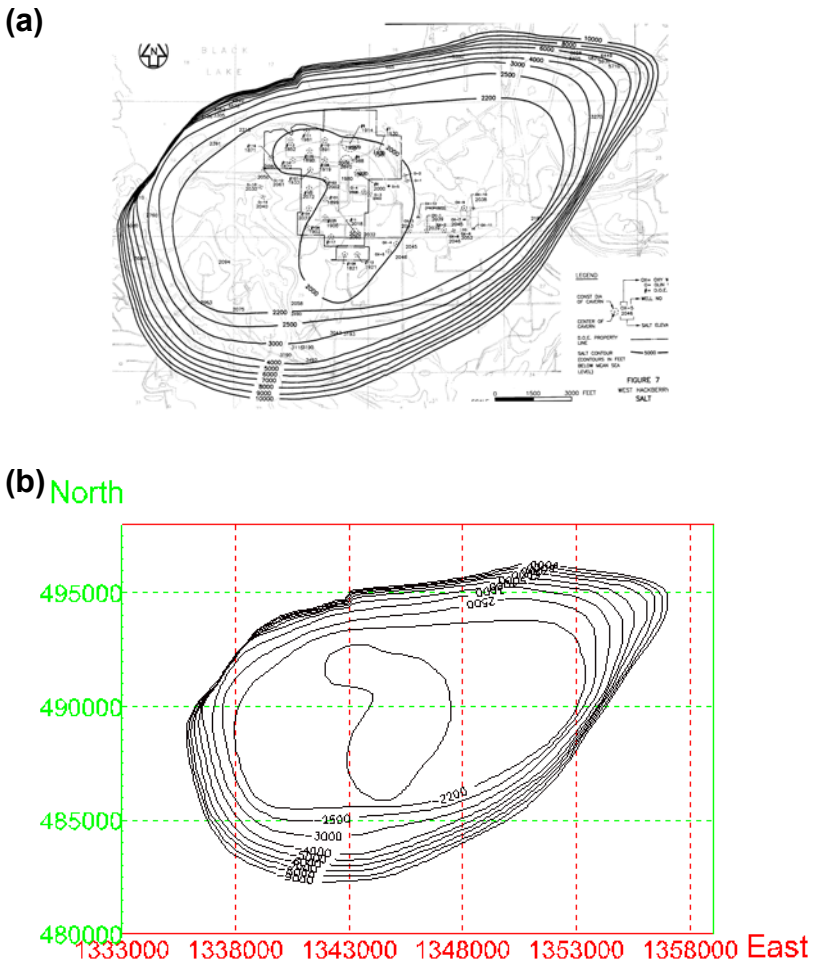


Figure 17. Comparison of visualizations of the West Hackberry salt dome. (a) Scanned bitmap image of the structure contour map showing the top of salt from site-characterization update report (Magorian and others, 1991). (b) Identical visualization of MVS model showing structure contours.

The advantages of computer-generated visualizations over a simple paper model are inferred to be more significant as the geometry involved in a specific model becomes more complex. The West Hackberry dome is a relatively simple "mound" of salt. However, the Bayou Choctaw salt dome (fig. 2) exhibits a pronounced structural overhang on the western and northwestern margins. The precise spatial configuration of the Bayou Choctaw dome is somewhat obscure in the structure-contour visualization of the dome, presented in figure 18 as a scanned (and reduced) replicate of the paper-report figure. This is regardless of whether those structure contours are the original paper model [fig. 18(a)] or generated from the computer version [fig. 18(b)]. In contrast, the flexibility provided by the alternative visualizations in figure 2 clearly — and more intuitively — display the "real" geometric configuration of the overhang.

Summary and Conclusions

The existing site characterization models of the salt-dome margins at the four active Strategic Petroleum Reserve sites have been converted to digital format and visualized using modern computer software. A new modeling algorithm has been developed to overcome limitations of many

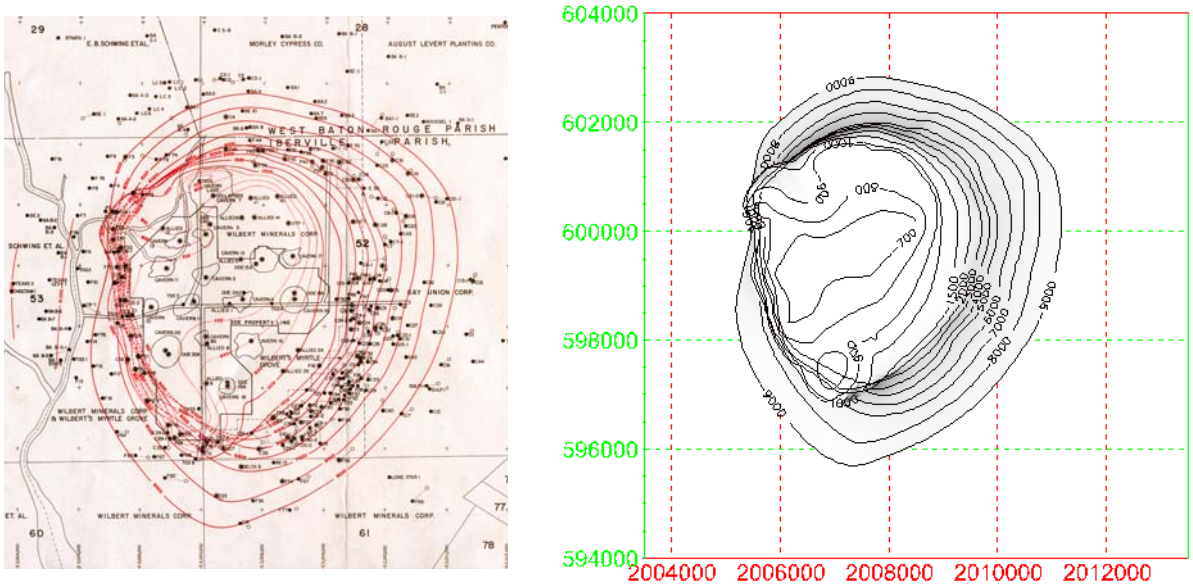


Figure 18. Comparison of visualizations of the Bayou Choctaw salt dome. (a) Scanned image of the structure contour map showing the top of salt from site-characterization update report (Neal and others, 1993). (b) Identical visualization of MVS model showing structure contours.

geological modeling software packages in order to deal with structurally overhanging salt margins that are typical of many salt domes. This algorithm, and the implementing computer program, make use of existing interpretive modeling conducted manually using professional geological judgement and presented in two dimensions in the site characterization reports as structure contour maps on the top of salt. The algorithm makes use of concepts of finite-element meshes of general engineering usage. Although the specific implementation of the algorithm described in this report and the resulting output files are tailored to the modeling and visualization software used to construct the figures contained herein, the algorithm itself is generic and other implementations and output formats are possible.

The graphical visualizations of the salt domes at the four SPR Sites, Bayou Choctaw (La.), Big Hill (Tex.), Bryan Mound (Tex.), and West Hackberry (La.) are believed to be major improvements over the previously available two dimensional representations of the domes via conventional geologic drawings (cross sections and contour maps). The newer computer renditions are more intuitive and explicit than the older paper drawings. Note that a sequence of level plans, such as those illustrated in figures 11 and 16, can be combined and presented on a single figure, thus essentially producing the same type of paper structure contour drawing as conventionally used to display geologic features. Additionally, the numerical mesh files produced by the current modeling activity are available for import into and display by other software routines. Although the mesh data are not explicitly tabulated in this report, an electronic version in simple ASCII format is included on a CD-R (Appendix B).

The Bayou Choctaw dome is shown to be a generally cylindrical mass, with a slight elongation toward the east, and a significant structural overhang on the southern and western sides. Essentially the only differences between the original and updated site characterization models is the detailed configuration of the top-of-salt surface. The updated model (table 1: Neal et al., 1993) was developed to present a modestly complex pattern of small-scale faulting on the top of salt. We did not attempt to capture this faulted geometry.

The Big Hill dome, as originally modeled during initial site characterization, is shown to be an irregularly cylindrical mass that is tilted strongly toward the south. The site-characterization update model (table 1: Magorian and Neal, 1988) did not attempt to change the overall configuration of the salt dome. Instead, this update report attempted to describe some of the internal structure of the Big Hill dome based on data resulting from drilling and leaching of the 14 SPR caverns. Because initial characterization of the Big Hill site was conducted before construction of the 14 SPR caverns provided numerous reliable stratigraphic elevations for the top-of-salt surface, only the closure-to-centroid modeling method (fig. 5) is possible using strictly the original information (table 1: Hart and others, 1981). Construction of the merged-surface model (fig. 4) is only possible through incorporation of cavern-well data from the update report information of Magorian and Neal (1988).

Visualizations have been produced for both the original site characterization model and an updated version of the Bryan Mound salt dome. The models are generally similar in representing an essentially upright, irregularly circular, cylindrical mass of salt. Potentially significant differences in the geometry and lateral position of the salt margins at depth exist between the two versions, and it is apparent that considerable variation in our understanding of the precise position of the salt-sediment interface exists at present. The lateral positions of the salt-sediment interface may vary by distances of up to approximately 1000 ft for the deepest (modeled) portions of the salt mass.

Two different versions of the West Hackberry salt dome were also modeled and visualized. The West Hackberry dome is shown to be a quite elliptical body, elongated east-northeast to west-southwest. There are numerous differences in the details of the salt flanks between the original site characterization report and the updated model. Some of these differences may be of potential significance to the SPR program, whereas others involve portions of the salt dome far removed from the West Hackberry SPR facilities.

A full-scale remodeling of the Strategic Petroleum Reserve salt domes has not been performed as part of this algorithm-development and model-conversion activity. Neither has an evaluation of the data and interpretations contained in the original vs. updated site characterization reports (table 1) been attempted. The converted models, particularly those that rely on geologically based top-of-salt surface to represent the crest of the salt domes, are suggestive that potentially meaningful geologic information regarding internal salt movements may exist and might be usefully extracted by yet-another characterization update.

For some sites (e.g., Big Hill; see Rautman, 2001), it is known that significant new geologic data (specifically three dimensional seismic data) have been obtained since completion of the first site characterization update report (Magorian and Neal, 1988). Other geologic data — seismic, drill hole, and other — may have become available both at Big Hill and for the other SPR salt domes. Therefore, this report, and particularly the modeling algorithm and approaches described herein, should be considered as part of a future planned program of updated site characterization.

REFERENCES

Hart, R.J., Ortiz, T.S., and Magorian, T.R., 1981, *Strategic Petroleum Reserve (SPR) geological site characterization report, Big Hill salt dome*, Sandia Report SAND81-1045, Sandia National Laboratories, Albuquerque, N. Mex.

Hogan, R.G. (ed.), 1980a, *Strategic Petroleum Reserve (SPR) geological site characterization report, Bayou Choctaw salt dome*, Sandia Report SAND80-7140, Sandia National Laboratories, Albuquerque, N. Mex.

Hogan, R.G. (ed.), 1980b, *Strategic Petroleum Reserve (SPR) geological site characterization report, Bryan Mound salt dome*, Sandia Report SAND80-7111, Sandia National Laboratories, Albuquerque, N. Mex.

Journel, A.G., and Huijbregts, C.J., 1978, *Mining Geostatistics*, New York: Academic Press, 600 p.

Magorian, T.R., Neal, J.T., Perkins, S., Xiao, Q.J., and Byrne, K.O., 1991, *Strategic Petroleum Reserve (SPR) additional geologic site characterization studies, West Hackberry salt dome, Louisiana*, Sandia Report SAND90-0224, Sandia National Laboratories, Albuquerque, N. Mex.

Neal, J.T., Magorian, T.R., Byrne, K.O., and Denzler, S., 1993, *Strategic Petroleum Reserve (SPR) additional geologic site characterization studies, Bayou Choctaw salt dome, Louisiana*, Sandia Report SAND92-2284, Sandia National Laboratories, Albuquerque, N. Mex.

Neal, J.T., Magorian, T.R., and Ahmad, S., 1994, *Strategic Petroleum Reserve (SPR) additional site characterization studies, Bryan Mound salt dome, Texas*, Sandia Report SAND94-2331, Sandia National Laboratories, Albuquerque, N. Mex.

Rautman, C.A., (ed.), Weisenburger, K., Uden, R., and Williams, C.V., 2001, *Gravity-based 3-D modeling of the Big Hill Strategic Petroleum Reserve Site, Texas*, Sandia Report SAND2001-1938, Sandia National Laboratories, Albuquerque, N. Mex. (Official Use Only).

Walker, Jr., Cameron, 1992, *1991-1992 high-resolution seismic survey at Big Hill salt dome*, unpublished contractor report by Walker Geophysical Company, Essex, Iowa, to Sandia National Laboratories; contained in Sandia National Laboratory project files.

Whiting, G.H., 1980, *Strategic Petroleum Reserve (SPR) geological site characterization report, West Hackberry salt dome*, Sandia Report SAND80-7131, Sandia National Laboratories, Albuquerque, N. Mex.

This page intentionally left blank

Appendix A: Fortran Program `ctr2evs`

This page intentionally left blank.

INTRODUCTION

This appendix contains a listing of the Fortran program, **`ctr2evs`**, that was written to read digitized contours and to convert them into a two-dimensional mesh suitable for input into Mining Visualization System (or any of the other C-Tech Development Corporation's) geologic modeling and visualization software packages. The program is written using generally Fortran-90 conventions and using structured programming techniques. The program has been successfully compiled and executed using Microsoft Corporation's "Fortran Powerstation 4.0™" running under Microsoft's Windows 2000™ operating system. The original ASCII text source code is included on the CD-R that is part of this report.

The program consists of a main program segment that reads the digitized data and performs most of the operations necessary to produce the output mesh file, including computation of the initial ray-intersections for resampling the digital contours. The program uses Fortran-90 dynamic array allocation procedures; array specifications are contained in a separate **`module`** file and storage is allocated at runtime to meet the requirements of the data. The actual array allocation, as well as computation of a number of frequently used customized function calls are contained in a separate **`functions`** file. A major subroutine, named **`trace`**, deals with multiple ray intersections for the same angular increment, and it literally "traces" the various line segments involved for all contours and distributes the required number of points equally along that trace.

Note that although care has been taken to ensure proper formatting of the Fortran source code that follows, it is possible that the conversion of the original ASCII text to a word-processing format may have resulted in the loss of original tabs or word-wrapping of lines longer than the width of this page. Such non-allowable Fortran occurrences will cause compile errors if the following source code is simply copied and pasted into a Fortran-type editor and compiled. First, there is no need for such a procedure to use this computer program as the original Fortran source files are included in digital form on the included CD-ROM. Second, any such errors should be quite evident to an experienced Fortran programmer and correctable quite easily.

FORTRAN MAIN PROGRAM LISTING

```
program ctr2evs
!program to create a "UCD" format file of a "surface" enclosing a set of contour lines
!  contours must be closed, as for a salt dome; written for SPR purposes.
!  produces EVS quadrilateral elements with equal numbers of points around
!  the circumference.

!written by Chris Rautman, SNL 6113                              13 May 2001
!  modified to deal with re-entrant contours                     15 June 2001
!  note: there is still some sort of bug that causes array deallocation
!  failures for certain angular increments; Fortran is not trapping errors
!  as advertised and the source of this error is unknown --      15 June 2001
!  major modifications by Chris Rautman, SNL 6113                22 November 2001
!    subroutine trace rewritten to deal with "cross-over" situations where
!    there are multiple intersections (re-entrant contours) at the ray-angle
!    equal 360 degrees/ray number 1 boundary
!    also mods to main program to restructure contour line-segment array to
!    number all line segments from x-axis through center of contour
!    (may have corrected problem of 15 June 2001)
!  modified by CAR to provide for output of absolute or          10 December 2001
!     relative coordinates; relative coords are centered at
!     (0,0) of overall bounding box for all contours.
!  modified by CAR to provide automatic closure to center of     26 April 2002
!     uppermost contour at elevation 1/2 above last two contours
!     fixed bust in writing elements for the "not closed" case   9 May 2002

!*************************************************************************
!                        Input Variables
!fname      filename with digitized contour data in (x,y,z) triplets
!v2         contour z-value read from file for counting number of contours.
!vr()       contour points (x,y,z) triplets as read from file: fname
!x(),y(),z()   assigned coordinate values of contour points
!incr       angular increment on which to compute rays
!ans        character answer to input question(s)
!dbg        debugging flag; must be set via data statement at compile time

!                        Output Variables
!px(),py()     points of intersections of rays from center of contour with perimeter
!z()        elevation of nodal coordinates
!nodes      number of intersection points; nodes for EVS
!elem       number of quadrilateral elements defined by the nodes, for EVS

!                        Internal Variables
!fname2     synthesized filenames for debugging files
!v1         comparison variable for counting number of contours contained in file
!nj         number of contours identified from input
!ni         count of number of contour points in each contour polyline
!nimax      maximum number of points across all contour polylines, used to
!           allocate arrays; also redefined to count number of records for EVS
!dist       distance from last point on a contour to the first point; used
!           to determine existence of a closed contour polyline
!sumd       sum of distances around contour; used to estimate closed or open contour
!nmiss      number of non-closed contours omitted from calculations
!n(j)       number of contour points for contour j
!nn         placeholder for n(j) in call to subroutine trace to prevent resetting
!npts()     number of points to be inserted by subroutine trace for each ray
!cx(),cy()  center points of closed contours, from which rays radiate
```

```
!minx,miny  values of the bounding box for the closed contour, from
!maxx,maxy   which cx() and cy() are calculated
!bbminx,bbminy    values of the bounding box for the entire structure; similar
!bbmaxx,bbmaxx     to minx,maxx, etc.
!maxj1,maxj2   indices of the highest and second-highest contours (z-values)
!segno()    number associated with each line segment (currently not used)
!seg0       line segment number for positive x-axis intercept, used to
!            determine sense of line numbering for ccw()
!b_ray,e_ray   starting and stopping line segment numbers in call to trace
!ni()       number of intersections for a particular ray, k, for contour j
!nintsect() number of intersection points for a contour, j
!nk         number of angular rays to use; computed from incr
!factor     used in computing EVS node indices
!m1, m2     slopes of the ray and line segment, used to calculate intersection
!d()        distances from ends of line segment to px,py, and total line segment
!Ix,Iy      points of intersection before being identified as within line segment
!presvrX()     set equal to Ix, Iy, Seg0 for the very first ray intersection;
!presrvY()      used to restore a starting intersection in subroutine trace for cases
!presrvSeg()    in which px,py are undefined because of multiple intersections on ray 1
!temp       used to store a temporary value of Iy prior to final assignment
!indx()     index of final intersections points, px(),py(), accounting for
!            multiple intersections [nintsect()]
!maxnodes   maximum number of intersection points; first of any contour, reset
!            later to be across all contours for writing UCD file
!tempx(),tempy()  temporary arrays used to resort line segments for ccw ordering
!zprime()   temporary array to resort contour depths in descending order
!i,j,k,jj,m index variables: i-line segments,j-contours,k-rays; also generic
!iperm()    array of indices for sorted array, z(); original order
!ierr       return error value from various Fortran calls
!flag       generic logical indicator
!ccw(j)     logical indicator of line-segment numbering counter-clockwise
!closed()   logical indicator of closed contours
!probray()  logical indicator for a ray with multiple intersections
!epsilon    "small" value for comparison of values for match

!                      External Routines
!dsvrgp()   sorts a double precision real vector and returns the permutation
!             vector (IMSL)
!cosd       cosine function in degrees (intrinsic)
!dtand      double-precision tangent (intrinsic)
!Dpy        double-precision pythagorean function (custom)
!angle      double-precision 360-degree angle determination from slope (custom)
!define_arrays    module to allow run-time allocation of certain arrays, specifically
!                  x(),y(),z(),px(),py(),segno(),presvrX,presrvY,presrvSeg;  (custom)

!************************************************************************
use msimsl
use msflib
use define_arrays
implicit none
logical flag,dbg,test
logical,allocatable::closed(:),probray(:),ccw(:)
character fname*40,ans*1,fname2*40
integer i,j,k,jj,nj,nk,nimax,maxnodes,nn,ierr,seg0,maxj1,maxj2
integer nmiss,nodes,elem,factor,b_ray,e_ray
integer,allocatable::n(:),ni(:,:),nintsect(:),iperm(:),npts(:)
real*4 incr,ang,minx,miny,maxx,maxy,bbminx,bbmaxx,bbminy,bbmaxy,sumd
real*8 vr(4),v1,v2,epsilon,m1,m2,dist,d(5),Ix,Iy,temp,Dpy
real*8,allocatable::cx(:),cy(:),tempx(:),tempy(:),zprime(:)
```

```
data epsilon /1e-6/ , dbg /.false./
data bbminx,bbminy,bbmaxx,bbmaxy /1.0e21,1.0e21,-1.0e21,-1.0e21/

!*****************************************************************************
!get file name and open files; user input
flag = .false.
do while( .not. flag )
   write(*,'(//,a,\)') ' Enter name of the file with the contour data: '
   read(*,'(a)') fname
   inquire(file=fname,exist=flag)
end do
write(*,*)
write(*,'(a,\)') ' Enter the angular increment to use, in degrees: '
read(*,*) incr

open(2,file=fname,action='read')
open(21,file='documentation.txt')
if(dbg) open(35,file='temp.dat')
!open(45,file='ChemDataFormat.csv')

!-----------------------------------------------------------------------------
!start documentation file for debugging purposes if needed
write(21,'(2a,/)') ' Opening contour data file: ',fname
write(21,'(a,f6.3,a,/)') ' Angular increment is:',incr,' degrees'
if( mod( 360.0,incr ) .ne. 0 ) then
   write( *,'(a)') ' Warning: angular increment does not give even number of rays'
   write(21,'(a)') ' Warning: angular increment does not give even number of rays'
end if
nk = int(360.0/incr)
write( *,'(a,i4,a,/)') ' Computations will be performed with',nk,' rays'
write(21,'(a,i4,a,/)') ' Computations will be performed with',nk,' rays'

!*****************************************************************************
!determine number of contour levels and maximum storage required
v1=0.0
nj=0
nimax=0
write(21,'(a)') ' Contours in original input-file sequence:'
do while( .not. eof(2) )
   read(2,*) (v2,k=1,3)
   !write(*,'(2f8.1)') v1,v2
   if( v2 .ne. v1 ) then
      nj = nj+1
      v1 = v2
      i = 1
      write(21,'(i3,f10.1)') nj,v2
   else
      i = i + 1
      nimax = max(nimax,i)
   end if
end do
rewind(2)
v1=0.0
write( *,'(2(a,i5),/)') ' No. of contours = ',nj,'; Maximum number of points = ',nimax
write(21,'(2(a,i5),/)') ' No. of contours = ',nj,'; Maximum number of points = ',nimax
write(21,*)

!-----------------------------------------------------------------------------
```

```
!note use of "nimax+1" in array allocation to allow for closing of contours by
!  program; allocation of certain  arrays performed in subroutine: allocate_arrays
allocate( probray(nk),closed(nj+1), stat=ierr )
if( ierr .ne. 0 ) stop 'Array allocation error 1'
allocate( iperm(nj+1),nintsect(nk),ni(nk,nj+1),npts(nk), stat=ierr )
if( ierr .ne. 0 ) stop 'Array allocation error 2'
allocate( cx(nj+1+1),cy(nj+1+1),n(nj+1),ccw(nj+1), stat=ierr)
call allocate_arrays(nimax,nj+1,nk)
if( ierr .ne. 0 ) stop 'Array allocation error 2'
x = -999.0;y = -999.0;z = -999.0

!-------------------------------------------------------------------------------
!read and store all the data for each contour, j
write(21,'(a)')' Checking for duplicate records....'
v1=0.0
nimax = 0            !nimax used here to count total number of records for MVS file
flag = .false.
do j=0,nj
   v2=0.0
   do while( .not. eof(2) )
      read(2,*) (vr(k),k=1,3)
      !write(*,*)(vr(k),k=1,3)
      if( vr(3) .ne. v1 ) then            !first record for a new contour
         backspace(2)
         v1 = vr(3)
         flag = .true.
         exit                             !this step increments j
      else if( flag .eqv. .true. ) then   !read the first record for a contour
         i=1
         x(i,j) = vr(1)                   !store data for first record of this contour
         y(i,j) = vr(2)
         z(j)   = vr(3)
         !nimax = nimax + 1
         write(21,'(a,i3,a,f10.1)')' Contour',j,' elevation:',z(j)
         flag = .false.
      else                              !read all remaining records; delete exact dupli-
cates
         if( vr(3) .ne. v1 ) exit      !identify new contour
         if( vr(1) .eq. x(i,j) .and. vr(2) .eq. y(i,j) ) then
            write(21,'(5x,a,i3,2i5,a)') ' Duplicate records, contour',j,i,i+1,&
             ';  deleting the second ...'
            v2 = v2 + 1                !keep count of deleted records
            cycle
         end if
         i=i+1                         !store all remaining records for this contour
         x(i,j) = vr(1)
         y(i,j) = vr(2)
         if( z(j) .ne. vr(3) ) then
            write(*,'(4i5,6f10.1)') j,i,nimax,nimax+int(v2),x(i,j),y(i,j),z(j),&
             vr(1),vr(2),vr(3)
            stop '>>> Irresolvable problem with depth values'
         end if
         v1 = vr(3)
         segno(i,j) = i
         n(j) = i
         nimax = nimax + 1             !nimax used here to count records for MVS file
      end if
   end do
   if( j .gt. 0 ) then
```

```
      write(21,'(2(a,i3,a,i5,/))') ' Original number of nodes in contour',j,':',&
       n(j)+nint(v2),'   Number of unique nodes in contour',j,':',n(j)
   end if
end do
close(2)             !close input file

!summarize input information
if(dbg) open(41,file='points.dat')
write(21,'(/,a,/,a)') ' Number of unique points per contour:',' Contour  Elevation
Npts'
do j=1,nj
   write(21,'(i6,f8.1,i6)') j,z(j),n(j)
   if(dbg) write(41,'(a,i4,f8.1,i5)') ' Contour',j,z(j),n(j)
   if(dbg) then
      do i=1,n(j)
         write(41,'(i5,3f12.2)') i,x(i,j),y(i,j),z(j)
      end do
   end if
end do
write(21,*)
close(41)

!------------------------------------------------------------------------------
!check for relict duplicate points along contour
!write( *,'(a,/)')' Checking for remaining near-identical points'
write(21,'(a)')  ' Checking for remaining near-identical points'
flag = .false.
do j=1,nj
   do i=1,n(j)-1
      if( abs(x(i,j)-x(i+1,j)) .lt. epsilon) then
         write(21,'(a,i3,a,2i5,a)') ' j=',j,'   i=',i,i+1,'  X-coordinates are identi-
cal'
         write(21,'(t10,2f12.4)') x(i,j),x(i+1,j)
         flag = .true.
      end if
      if( abs(y(i,j)-y(i+1,j)) .lt. epsilon) then
         write(21,'(a,i3,a,2i5,a)') ' j=',j,'   i=',i,i+1,'  Y-coordinates are identi-
cal'
         write(21,'(t10,2f12.4)') y(i,j),y(i+1,j)
         flag = .true.
      end if
   end do
end do
if( flag .eqv. .true. ) then
   write(*,'(/,a,/,a,//)')' Duplicate points have been found; cannot continue.', &
   '   Check file: documentation.text for errors'
   write(21,'(/,a,//)')' Near-duplicate points have been found; cannot continue.'
   stop
end if
write( *,'(a,/)') ' Checking duplicate points: Done'
write(21,'(a,/)') ' Checking duplicate points: Done'

!------------------------------------------------------------------------------
!check for closed contours
!note that this makes the first and last points in a contour identical <<<<<
write( *,'(/,a)')  ' Checking for closed contours...'
write(21,'(/,a,/)')' Checking for closed contours; last distance should be zero'
nmiss = 0            !nmiss is the number of non-closed contours
do j=1,nj
```

```
   closed(j) = .true.
   !if the distance between first and last points = 0, contour is already closed
   dist = Dpy( x(1,j),y(1,j),x(n(j),j),y(n(j),j) )
   write( *,'(a,i3,f12.1a,f12.2)') ' For contour',j,z(j),', the last distance =',dist
   write(21,'(a,i3,f12.1a,f12.2)') ' For contour',j,z(j),', the last distance =',dist
   if( dist .ne. 0.0 ) closed(j) = .false.
   if( closed(j) ) then
      write(21,'(a,/)') '     assuming contour is closed and continuing...'
   else
      !if last distance <> 0, find out if user wants to consider the contour closed
      ! but just lacking a duplicate last point, based on similarity of last distance
      ! to the average or "typical" segment-length distance
      sumd = 0.0
      do i=1,n(j)-1
         dist = Dpy( x(i,j),y(i,j),x(i+1,j),y(i+1,j) )
         sumd = sumd + dist
      end do
      dist = sumd/float(n(j))
      write( *,'(a,f8.2)')'     The average segment length is =',dist
      write(21,'(a,f8.2)')'     The average segment length is =',dist
      write(*,'(a,\)')'        Assume that this contour is closed? (y/n): '
      read(*,'(a)') ans
      if( ans .eq. 'y' .or. ans .eq. 'Y' ) then
         write( *,'(a,i3,a,/)')'     ... closing contour',j,' ...'
         write(21,'(a,i3,a,/)')'     ... closing contour',j,' ...'
         !add a "final" point equal to the first point and increment n(j) for consis-
tency
         n(j) = n(j) + 1
         x(n(j),j) = x(1,j)
         y(n(j),j) = y(1,j)
         closed(j) = .true.
      else
         !leave line as NOT closed and write all line-segment data for debugging
         nmiss = nmiss + 1
         do i=1,n(j)-1
            dist = Dpy( x(i,j),y(i,j),x(i+1,j),y(i+1,j) )
            if(dbg) write(21,'(a,i5,a,f8.2,a,3(f10.1,a))') ' Distance',i,' =',dist,&
             ':   ',x(i,j),',',y(i,j),',',z(j)
         end do
         write(21,'(a,i3,f10.1,a,/)')' Contour:',j,z(j),' not closed; omitting'
         write( *,'(a,i3,f10.1,a,/)')' Contour:',j,z(j),' not closed; omitting'
         write(21,*)
      end if
   end if
end do
write(*,'(a,/)')'    Checking closed contours: Done'

!-------------------------------------------------------------------------------
!set up a file for MVS Map_Spheres without duplicate points
open(43,file='UniqueContourPts.csv')
write(43,'(2a)')'#Unique contour data points from file: ',fname
write(43,'(a)')' 1'
write(43,'(i6,a)') nimax,',1'       !number of records and variables for MVS
fname2='                                '
do j=1,nj
   if( dbg ) then          !only if seriously debugging
      !also write individual contour files with synthetic names for debugging
      !if( closed(j) .eqv. .false. ) cycle
      i=len_trim(fname)
```

```
      write(fname2(1:i-3),'(a)') fname(1:i-3)
      write(fname2(i-2:i+2),'(i5.5)') -nint(z(j))
      write(fname2(i+3:i+6),'(a)') '.dat'
      !write(*,'(a,i3,2a)') ' writing debugging file for contour',j,': ',fname2
      open(33,file=fname2,status='unknown')
      do i=1,n(j)-1           !omit the duplicate of the starting point
         write(33,'(3(f15.3,a))') x(i,j),',',y(i,j),',',z(j)
      end do
      close(33)
   end if
   do i=1,n(j)-1           !omit the duplicate of the starting point
      write(43,'(4(f15.3,a))') x(i,j),',',y(i,j),',',z(j),',',z(j)
   end do
end do
test=commitqq(43)
close(43)
write(21,'(a,/)')' Have successfully written EVS Map_Spheres file: UniqueCon-
tourPts.csv'
write( *,'(a,/)')' Have successfully written EVS Map_Spheres file: UniqueCon-
tourPts.csv'
write(*,*)
test=commitqq(21)

!------------------------------------------------------------------------------
!find approximate center points for radial computations
! bounding-box x- and y-values initialized using a data statement
write( *,'(a)')' Finding approximate centers of each valid contour...'
write(21,'(a)')' Finding approximate centers of each valid contour...'
do j=1,nj
   if( closed(j) .eqv. .false. ) cycle
   !find max and min x- and y-values
   maxx = -1e+21
   maxy = -1e+21
   minx =  1e+21
   miny =  1e+21
   do i=1,n(j)
      maxx = max( maxx,x(i,j) )
      maxy = max( maxy,y(i,j) )
      minx = min( minx,x(i,j) )
      miny = min( miny,y(i,j) )
   end do
   cx(j) = minx + (maxx - minx)/2.0    !approximate center as midpoint
   cy(j) = miny + (maxy - miny)/2.0
   !write( *,'(a,i3,a,2f10.1)')' Center of contour',j,':',cx(j),cy(j)
   write(21,'(a,i3,a,2f10.1)')' Center of contour',j,':',cx(j),cy(j)
   write(21,'(a,2f10.1,5x,2f10.1)')' Bounding box: ',minx,miny,maxx,maxy

   !determine sense of line-segment numbering (cw vs. ccw)
   m2 = 0.0                            !for angle = 0 (positive x-axis)
   do i=1,n(j)-1                       !iterate over all line segments of this contour
      if( x(i,j) .gt. cx(j) ) then     !work only with x-values to right of center
         !compute the slopes of the line segments, m1
         m1 = ( y(i+1,j) - y(i,j) )/( x(i+1,j) - x(i,j) )
         if( abs(m1 - m2) .lt. epsilon ) cycle
         !compute the point of intersection
         Ix = ( y(i,j)-m1*x(i,j)+m2*cx(j)-cy(j) ) / (m2-m1)
         Iy = y(i,j) + m1*Ix - m1*x(i,j)
         !determine if intersection point falls between line segment ends
         ! if intersection is within this line segment, d(1) + d(2) = d(3) +/- epsilon
```

```
         d(1) = Dpy( x(i,j),y(i,j),Ix,Iy )
         d(2) = Dpy( x(i+1,j),y(i+1,j),Ix,Iy )
         d(3) = Dpy( x(i,j),y(i,j),x(i+1,j),y(i+1,j) )
         !THIS is the critical test for intersection
         if( (d(1) + d(2) .le. d(3) + epsilon) .and. (d(1) + d(2) .ge. d(3) - epsilon)
) then
            if( y(i+1,j) - y(i,j) .gt. 0 ) ccw(j) = .true.
            if(dbg) write( *,'(a,i3,a,l1)')' Contour',j,' runs counter-clockwise:
',ccw(j)
            write(21,'(a,i3,a,l1,/)')' Contour',j,' runs counter-clockwise: ',ccw(j)
            !seg0(j) = i                  !preserve the x-axis intercept segment no.
            exit
         end if
      end if
   end do

   !find the overall center of the bounding box for all contours
   bbmaxx = max( bbmaxx,maxx )
   bbmaxy = max( bbmaxy,maxy )
   bbminx = min( bbminx,minx )
   bbminy = min( bbminy,miny )
end do

!--------------------------------------------------------------------------------
!need to invert line segment order for clockwise contours
do j=1,nj
   if( closed(j) .eqv. .false. ) cycle
   if( ccw(j) .eqv. .true. ) cycle
   write(21,'(a,i3,a)')' inverting x- and y-values for contour',j,';  now ccw'
   allocate( tempx(n(j)),tempy(n(j)) )
   do i=1,n(j)
   !write(*,'(3i5)') n(j),i,n(j)-i+1
      tempx(n(j)-i+1) = x(i,j)
      tempy(n(j)-i+1) = y(i,j)
   end do
   do i=1,n(j)
      x(i,j) = tempx(i)
      y(i,j) = tempy(i)
   end do
   deallocate( tempx,tempy)
   test=commitqq(21)
end do
write(21,*)

!--------------------------------------------------------------------------------
!write( *,'(/,a,2(2f10.1,3x))')' Overall bounding box is:',bbminx,bbminy,bbmaxx,bbmaxy
write(21,'(/,a,2(2f10.1,3x))')' Overall bounding box is:',bbminx,bbminy,bbmaxx,bbmaxy
!note use of index "nj+1" to store bounding-box center
cx(nj+1) = bbminx + (bbmaxx - bbminx)/2.0
cy(nj+1) = bbminy + (bbmaxy - bbminy)/2.0
!increase bounding box by 20 percent for chem-data-type calculations along rays
bbminx = bbminx - (bbmaxx - bbminx)*0.20
bbmaxx = bbmaxx + (bbmaxx - bbminx)*0.20
bbminy = bbminy - (bbmaxx - bbminx)*0.20
bbmaxy = bbmaxy + (bbmaxx - bbminx)*0.20
if(dbg) write( *,'(/,6x,a,2f10.1)')' Overall center is:',cx(nj+1),cy(nj+1)
if(dbg) write( *,*)
write(21,'(/,6x,a,2f10.1)')' Overall center is:',cx(nj+1),cy(nj+1)
write(21,*)
```

```
test=commitqq(21)

!-------------------------------------------------------------------------------
!this code reorganizes the x and y arrays so that x(1),y(1) is at end of x-axis
write( *,'(a)') Restructuring contour line segments ...'
if(dbg) open(55,file='newarray.dat')
do j=1,nj
   if( closed(j) .eqv. .false. ) cycle
   if(dbg) write(55,'(a,i3,a,f8.1a,i5)') working on contour',j,' elevation:', &
    z(j),' # points:',n(j)
   allocate( tempx(n(j)+10),tempy(n(j)+10) )
   tempx = -999.; tempy=-999.
   !find +X-axis intercept
   do i=1,n(j)
      if( x(i,j) .lt. cx(j) ) cycle       !work only to +X side of center
      if( (y(i,j) .le. cy(j)) .and. (y(i+1,j) .ge. cy(j)) ) then
         seg0 = i
         exit
      end if
   end do
   k = 0
   write(21,'(a,i3,a,i5)') Restructuring contour',j,' starting at segment',seg0
   write(*,*)'Segment Zero =',seg0, '  contour',j,'  n(j)=',n(j)

   do i=seg0,n(j)-1
      k = k + 1
      tempx(k) = x(i,j)
      tempy(k) = y(i,j)
      if(dbg) write(55,'(2(i5,2f12.2))') i,x(i,j),y(i,j),k,tempx(k),tempy(k)
   end do
   do i=1,seg0-1
      k = k + 1
      tempx(k) = x(i,j)
      tempy(k) = y(i,j)
      if(dbg) write(55,'(2(i5,2f12.2))') i,x(i,j),y(i,j),k,tempx(k),tempy(k)
   end do
   !now refill the original arrays
   do i=1,n(j)
      x(i,j) = tempx(i)
      y(i,j) = tempy(i)
      !write(*,'(2(i5,2f12.2))') i,x(i,j),y(i,j),k,tempx(k),tempy(k)
   end do
   x(n(j),j) = x(1,j)
   y(n(j),j) = y(1,j)
   !write(55,'(3f12.2,a)') x(i,j),y(i,j),z(j),' end of data'

   !do i=1,n(j)
   !  write(55,'(3f12.2)') x(i,j),y(i,j),z(j)
   !end do
   deallocate( tempx,tempy )
   if(dbg) test = commitqq(55)
end do
if(dbg) close(55)
write(21,'(a,/)')'     done renumbering contours from the east'
test=commitqq(21)

write(*,*)
pause '   >>>  Data input complete; press <Enter> to continue'
```

```
!*************************************************************************
!compute and assign the intersection points for rays with line segments

maxnodes = 0                  !maxnodes is the total (maximum) number of nodes required
                              ! for equalization of numbers across all contours
!initialize probray() array   !all rays are OK initially
do k=1,nk
   probray = .false.
end do
do j=1,nj                     !iterate over all contour levels
nintsect(j) = 0
if( closed(j) .eqv. .false. ) cycle    !skip non-closed contours `
if( dbg ) then                !only if seriously debugging
   fname2='RayIntersects.dat                    `
   i=len_trim(fname2)
   write(fname2(i-2:i+2),'(i5.5)') -nint(z(j))
   write(fname2(i+3:i+6),'(a)') '.dat'
   write( *,'(/,a,i3,2a)') ` writing ray-intersection file for contour',j,': ',fname2
   write(21,'(/,a,i3,2a)') ` writing ray-intersection file for contour',j,': ',fname2
   open(33,file=fname2,status='unknown')
end if
do k=1,nk                          !iterate over all angles
   !compute angle and slope of the radial line, m2
   ang = float(k)*incr
   if( ang .eq. 90.0 .or. ang .eq. 270.0 ) then
      flag = .true.
      !write(*,'(i3,a,i4,a,f10.5,a,\)') j,' angle** =', k,'  m2 =',ang, `  no slope'
   else
      flag = .false.
      m2 = dtand(dfloat(ang))
      !write(*,'(i3,a,i4,a,2f10.5,/)') j,' angle   =', k,'  m2 =',ang, m2
   end if

   ni(k,j)=0                          !ni(k,j) is number of intersections for this ray
   do i=1,n(j)-1                      !iterate over all line segments of this contour
      !compute the slopes of the contour-line segments, m1
      m1 = ( y(i+1,j) - y(i,j) )/( x(i+1,j) - x(i,j) )
      if( abs(m1 - m2) .lt. epsilon ) then
         !lines are parallel; no intersection
         write(21,'(a,i4,f10.5)') ` line segment delta slope:', i,  m2-m1
         write(*,'(a,i5,a,2f10.5)') ` line segment: ',i,' -- lines are parallel',m1,m2
         pause
         Ix = cx(j)              !if parallel, set intersection to center
         Iy = cy(j)
      else
         !compute the point of intersection; use of cosine keeps track of quadrant
         if( flag .eqv. .false. ) then
            Ix = ( y(i,j)-m1*x(i,j)+m2*cx(j)-cy(j) ) / (m2-m1)
            if( Ix .ge. cx(j) .and. cosd(ang) .ge. 0.0 ) then
               Iy = y(i,j) + m1*Ix - m1*x(i,j)
            else if( Ix .le. cx(j) .and. cosd(ang) .le. 0.0 ) then
               Iy = y(i,j) + m1*Ix - m1*x(i,j)
            else
               cycle
            end if
         else                     !ang=90 or 270 where tan(ang) undefined
            Ix = cx(j)
            Iy = y(i,j) + m1*Ix - m1*x(i,j)
            if( ang .eq. 90.0 .and. Iy .le. cy(j) ) then
```

```
            temp = Iy
            cycle
         else if( ang .eq. 270.0 .and. Iy .ge. cy(j) ) then
            Iy = temp
         end if
      end if
   end if
   !determine if intersection point falls between line segment ends
   ! if intersection is within this line segment, d(1) + d(2) = d(3) +/- epsilon
   d(1) = Dpy( x(i,j),y(i,j),Ix,Iy )
   d(2) = Dpy( x(i+1,j),y(i+1,j),Ix,Iy )
   d(3) = Dpy( x(i,j),y(i,j),x(i+1,j),y(i+1,j) )

   !THIS is the critical test for intersection
   if( d(1) + d(2) .le. d(3) + epsilon .and. d(1) + d(2) .ge. d(3) - epsilon ) then
      !Ix,Iy is a true intersection of the ray and the line segment
      !now ASSIGN the intersection location for storage
      px(k,j) = Ix                  !preserve the nodal locations
      py(k,j) = Iy
      segno(k,j) = i                !track line segment associated with intersection
      ni(k,j)=ni(k,j)+1             !count the number of intersections for this ray
      nintsect(j) = nintsect(j) + 1 !count the total number of intersections
      if( (k .eq. 1) .and. (ni(k,j) .eq. 1) ) then
         !preserve very first intersection in case of problems later
         presrvX(j) = Ix
         presrvY(j) = Iy
         presrvSeg(j) = i
         write(21,'(a,2f12.1,2(a,i4))')' Preserving first intercept: (',&
          presrvX(j),presrvY(j),')',presrvSeg(j),'  contour',j
      end if
      if( ni(k,j) .gt. 2 ) then
         write(21,'(a,f8.1,a,f5.0,2(a,i4))')'    >>> contour',z(j),' angle',ang, &
          ' ray',k,' has multiple intersections:',ni(k,j)
         write(*,'(a,f8.1,a,f5.0,a,i3)')' contour',z(j),' angle',ang, &
          ' has multiple intersections:',ni(k,j)
         probray(k) = .true.
         px(k,j) = -999.              !replace intersection with dummy coordinates
         py(k,j) = -999.
         segno(k,j) = -9
      end if
      maxnodes = max( maxnodes,nintsect(j) )!nkement count of maximum number of nodes
      if( y(i+1,j) - y(i,j) .gt. 0 ) ccw(j) = .true.
      !write list of rays and intersections for debugging
      if(dbg) write(35,'(2f10.1,f5.0,i5,f8.0,i5,l1,4i5,2f10.1)') Ix,Iy,ang, &
        j,z(j),k,probray(k),indx(j),i,ni(k,j),segno(indx(j),j),x(i,j),y(i,j)
      !write files with intersection points for debugging using Sigma Plot
      if( dbg ) then       !serious debugging output
         write(33,'(2f12.1)') cx(j),cy(j)
         write(33,'(2f12.1,i5,f8.0)') Ix,Iy,k,z(j)
         write(33,*)
      end if
   end if
 end do
end do
write(21,'(a,f8.1,a,i5)')' Contour',z(j),',  total no. of intersections:',nintsect(j)
write( *,'(a,f8.1,a,i5)')' Contour',z(j),',  total no. of intersections:',nintsect(j)
close(33)
end do
write(*,*)
```

```
flag=commitqq(21)
flag=commitqq(35)

write(21,'(/,a)')' Preserved first segment numbers and intersection points'
write(21,'(a)')' Contour Segment   X        Y'
do j=1,nj
   if( closed(j) .eqv. .false. ) cycle
   write(21,'(2i5,2f12.1)') j,presrvSeg(j),presrvX(j),presrvY(j)
end do
write(21,*)

!-------------------------------------------------------------------------------
!list sense of structure for each contour
!do j=1,nj
!  if( closed(j) .eqv. .false. ) cycle
!  write( *,'(a,i3,a,l1)')' Contour',j,' runs counter-clockwise: ',ccw(j)
!  write(21,'(a,i3,a,l1)')' Contour',j,' runs counter-clockwise: ',ccw(j)
!end do
!write(*,*)

!-------------------------------------------------------------------------------
!determine the number of evs nodes to insert for each angular increment (if any)
do k=1,nk                  !initialize
   npts(k)=0
end do
maxnodes = 0
i=0
do k=1,nk                  !count needed nodes
   if( probray(k) .eqv. .true. ) i = i + 1
   do j=1,nj
      if(closed(j) .eqv. .false. ) cycle
      npts(k) = max( npts(k),ni(k,j) )
   end do
   maxnodes = maxnodes + npts(k)
end do
write( *,'(/,a,i5,/)')' Number of required points per contour:',maxnodes
write(21,'(/,a,i5,/)')' Number of required points per contour:',maxnodes

!-------------------------------------------------------------------------------
!write a temporary file with the identified intersection points for debugging only
if( dbg ) then                   !serious debugging only
   open(42,file='nodes.dat')
   write(42,'(a)')'k  Intersection_pX   Intersection_pY  Contour_Z   segno  ProbRay
npts'
   do j=1,nj
      if( closed(j) .eqv. .false. ) cycle    !skip non-closed contours
      do k=1,nk
         write(42,'(i4,3f15.3,i5,5x,l1,i3)') k,px(k,j),py(k,j),z(j),&
          segno(k,j),probray(k),npts(k)
      end do
      write(42,'(17x,a,i5,/)') 'Total number of line segments = ',n(j)
   end do
   close(42)
   write( *,'(a,/)')' Have successfully written nodal-point file: nodes.dat'
   write(21,'(a,/)')' Have successfully written nodal-point file: nodes.dat'
end if

!-------------------------------------------------------------------------------
!find intersections before and after problem rays; compute fill-in points
```

```
if(dbg) open(44,file='finaldata.dat')
if( .not. dbg ) open(44)
do j=1,nj
   if( closed(j) .eqv. .false. ) cycle
   k = 1
   b_ray = k
   do while( k .le. nk )
      if( probray(k) .eqv. .false. ) then
         write(44,'(3f15.3,2i5,3x,l1)') px(k,j),py(k,j),z(j), j,  k,probray(k)
         !write( *,'(3f15.3,2i5,3x,l1)') px(k,j),py(k,j),z(j), j,  k,probray(k)
         b_ray = k
         k = k + 1
      else
         i = 0
         do while( k .le. nk )
            if( probray(k) .eqv. .true. ) then
               i = i + npts(k)
               k = k + 1
            else
               e_ray = k
               exit
            end if
         end do
         if( (k .ge. nk) .and. (probray(nk) .eqv. .true.) ) then
            !the final ray had a problem; wrap-around case
            e_ray = 1
            !note: e-ray is set to one because problem rays from 1 on
             !have already been dealt with at the beginning of this section
             !i.e., ray 1 (pX(1),pY(1)) will ALWAYS be an anchor point
            write(*,'(a,2i5,a,i5)')' >>> Wrap-around situation with rays:', &
             b_ray,e_ray,' >>> j =', j
         end if
         write(21,'(a,i3,a,f8.1,a)')' ***************  generating',i,&
          ' new points for contour',z(j),'  ***************'
         if(dbg) write(*,'(a,i3,a,f8.1)')' generating',i,' new points for contour',z(j)
         nn = n(j)         !avoid resetting n(j)
         if(dbg) write(*,'(2(a,i4))')' calling subroutine trace:  Beginning ray
=',b_ray,&
          ';  Ending ray =',e_ray
         call trace( j,b_ray,e_ray,i,nn,dbg )
         if(dbg) write(*,*)'back from subroutine trace'
      end if
   end do
   test=commitqq(21)
end do
!write(*,*)'rewinding file'
rewind(44)
!write(*,*)'   file rewound; beginning deallocation'

!------------------------------------------------------------------------------
!release unused memory
ierr = -999
deallocate( probray,ccw,n,ni,nintsect,npts,segno,x,y,indx, stat=ierr )
if( ierr .ne. 0 ) stop 'Array allocation error'

deallocate( px,stat=ierr )
if( ierr .ne. 0 ) stop 'Array allocation error'
!write(*,*) 'deallocated px'
!write(*,*)'deallocation error code:', ierr
```

```
!flag = allocated(py)
!write(*,*)'py flag = ',flag
deallocate( py,STAT = ierr )
if( ierr .ne. 0 ) stop 'Array allocation error'
!if( ierr .ne. 0 ) stop 'Array DEallocation error'
!write(*,*)'deallocation error code:', ierr
!write(*,*)'   successful deallocation'

!reallocate storage for final EVS nodes
allocate(px(maxnodes,nj),py(maxnodes,nj),stat=ierr )
if( ierr .ne. 0 ) stop 'Array allocation error'

write(*,'(//,a)')' Rereading temporary file'
do j=1,nj
   if( closed(j) .eqv. .false. ) cycle
   if( dbg ) then                   !for serious debugging only
      fname2 = 'finaldata.dat               '
      k=len_trim(fname2)
      write(fname2(k-2:k+2),'(i5.5)') -nint(z(j))
      write(fname2(k+3:k+6),'(a)') '.dat'
      open(33,file=fname2,status='unknown')
   end if
   do i=1,maxnodes
      read(44,*) px(i,j),py(i,j)
      if(dbg) write(33,'(3f15.3)') px(i,j),py(i,j),z(j)
   end do
   close(33)
end do
write(*,'(a)')' have reread the finaldata array '
test=commitqq(21)


!*****************************************************************************
!sort z-array by increasing depth so that UCD file is written in vertical sequence
! note that ALL references to "(i,j)" from here on out MUST be to "(i,iperm(j)))"
! in order to make use of rearranged z-array

do j=1,nj                         !must first initialize iperm
   iperm(j) = j
end do
!sort the depth (z-value) array
!create a dummy z-array to avoid messing up z; all we want is the permutation vector
allocate( zprime(nj) )
call dsvrgp(nj,z,zprime,iperm)
deallocate( zprime )
write(21,'(/,a)')' Have resorted the array of contours'
write( *,'(/,a)')' Have resorted the array of contours'

!need to invert iperm array order for UCD output; sort bottom to top
allocate( tempx(nj) )
do i=1,nj
   tempx(nj-i+1) = iperm(i)
   !write(*,'(2(i5,f10.1))') i,z(i),iperm(i),z(iperm(i))
end do
do i=1,nj
   iperm(i) = tempx(i)
end do
deallocate( tempx )
```

```
do j=1,nj
   !write( *,'(2i5,f10.1,3x,l1)') j,iperm(j),z(iperm(j)),closed(iperm(j))
   write(21,'(2i5,f10.1,3x,l1)') j,iperm(j),z(iperm(j)),closed(iperm(j))
end do
write(21,'(a,/)')' Depth array sorted OK'
write( *,'(a,/)')' Depth array sorted OK'
write(*,*)

!find two highest contour z-values
do j=1,nj
   if( closed(iperm(j)) .eqv. .false. ) cycle
   maxj1 = iperm(j)
   exit
end do
do j=2,nj
   if( closed(iperm(j)) .eqv. .false. ) cycle
   maxj2 = iperm(j)
   exit
end do
write(*,'(/,a,2f10.1)')' The two highest-elevation contours are:', &
 z(maxj1),z(maxj2)
write(*,'(24x,a,2i5)')'Contour indices:',maxj1,maxj2
write(21,'(/,a,2f10.1)')' The two highest-elevation contours are:', &
 z(maxj1),z(maxj2)
write(21,'(24x,a,2i5)')'Contour indices:',maxj1,maxj2

flag = .false.
do while( .not. flag )
   write(*,'(a,/,2(3x,a,/))')' Write UCD file in','1. absolute or','2. relative coordi-
nates?'
   write(*,'(a,\)')' Enter (1) or (2): '
   read(*,*) nn
   if( nn .eq. 1 .or. nn .eq. 2 ) flag = .true.
end do
write(*,*)
write(*,'(a,\)')' Close the salt-dome shell to the centroid? (y/n): '
read(*,*) ans

!*******************************************************************************
!NOW write the UCD mesh file
write(*,'(a,/)')' Writing final unstructured-mesh data file...'
i= len_trim(fname)
write(fname(i-3:i),'(a)')'.inp'
open(50,file=fname)
write(50,'(2a)')'#ucd mesh file generated from file: ',fname
if( nn .eq. 2 ) then
   write(50,'(2(a,f12.3))')'#Note: RELATIVE coordinates: Xcen=',cx(nj+1),'
Ycen=',cy(nj+1)
   write(50,'(a)')'#Note: Z-coordinates increase DOWNward for use by DirecX'
else
   write(50,'(2(a,f12.3))')'#Note: ABSOLUTE coordinates: Xcen=',cx(nj+1),'
Ycen=',cy(nj+1)
end if
if( ans .eq. 'y' .or. ans .eq. 'Y' ) then
   write(50,'(a)')'#ucd shell is closed to overall centroid'
end if

nodes= maxnodes*(nj-nmiss)
elem = maxnodes*(nj-nmiss-1)
```

```
if( ans .eq. 'y' .or. ans .eq. 'Y') then
   write(*,'(a,i10)')'      Total number of nodes    =',nodes + 1
   write(*,'(a,i10)')'      Total number of elements =',elem + maxnodes
   write(50,'(2i10,a)')nodes+1,elem+maxnodes,' 1 0 0'
else
   write(*,'(a,i10)')'      Total number of nodes    =',nodes
   write(*,'(a,i10)')'      Total number of elements =',elem
   write(50,'(2i10,a)')nodes,elem,' 1 0 0'
end if

!-------------------------------------------------------------------------------
!write the coordinates of each node
jj=0
do j=1,nj
   if( closed(iperm(j)) .eqv. .false. ) cycle         !skip non-closed contours
   jj = jj + 1
   do k=1,maxnodes
      if( nn .eq. 1 ) then
         write(50,'(i10,3f15.3)') k+(jj-1)*maxnodes,px(k,iperm(j)), &
          py(k,iperm(j)),z(iperm(j))
      else
         write(50,'(i10,3f15.3)') k+(jj-1)*maxnodes,px(k,iperm(j))-cx(nj+1), &
          py(k,iperm(j))-cy(nj+1),-z(iperm(j))
      end if
   end do
end do
!write the centroid for closure
if( ans .eq. 'y' .or. ans .eq. 'Y') then
   write(50,'(i10,3f15.3)') k+(jj-1)*maxnodes,cx(maxj1),cy(maxj1),z(maxj1) &
    + (z(maxj1)-z(maxj2))/2
end if

write(21,'(//,a,7x,f15.3)')' Z-max elevation is:',z(maxj1)
write(21,'(a,f15.3)')' Z-second-max elevation is:',z(maxj2)
write(21,'(a,5x,f15.3,//)')' Closure elevation is:',z(maxj1) &
    + (z(maxj1)-z(maxj2))/2

!-------------------------------------------------------------------------------
!compute & write the element descriptions: element #, material #, element type, node #s
do j=1,nj-nmiss - 1
   do k=1,maxnodes
      if( k .eq. maxnodes ) then
         factor = maxnodes
      else
         factor = 0
      end if
      elem= k+(j-1)*maxnodes
      write(50,'(i10,a,4i10)') elem,' 5 quad ', &
       k+(j-1)*maxnodes, &
       k+j*maxnodes,&
       k+1+j*maxnodes - factor, &
       k+1+(j-1)*maxnodes - factor
   end do
end do
do k=1,maxnodes
   if( k .eq. maxnodes ) then
      factor = maxnodes
   else
      factor = 0
```

```
   end if
   if( ans .eq. 'y' .or. ans .eq. 'Y' ) then
      elem= k+(nj-nmiss-1)*maxnodes
      write(50,'(i10,a,4i10)') elem,' 5 tri  ', &
       k, &
       k+1-factor, &
       (nj-nmiss)*maxnodes +1
   end if
end do


!-------------------------------------------------------------------------------
!write "data values" for each node (use elevation of contour for now)
write(50,'(a)')'1 1'
write(50,'(a)')'Elevation, ft'
jj = 0
do j=1,nj
   if( closed(iperm(j)) .eqv. .false. ) cycle
   jj = jj + 1
   do k=1,maxnodes
      if( nn .eq. 1 ) then
         write(50,'(i10,f15.3)') k+(jj-1)*maxnodes,z(iperm(j))
      else
         write(50,'(i10,f15.3)') k+(jj-1)*maxnodes,-z(iperm(j))
      end if
   end do
end do
!write the centroid for closure
if( ans .eq. 'y' .or. ans .eq. 'Y') then
   write(50,'(i10,f15.3)') maxnodes+(jj-1)*maxnodes+1, &
    z(maxj1) + (z(maxj1)-z(maxj2))/2
end if

close(50)
write(21,'(/,3a)')' UCD file: ',trim(fname),' written successfully'
write( *,'(/,3a)')' UCD file: ',trim(fname),' written successfully'
write(*,*)

!*******************************************************************************
close(21)
deallocate( px,py,z,cx,cy,iperm, stat=ierr)
if( ierr .ne. 0 ) stop 'Deallocation error at end'
stop
end
```

ARRAY-DECLARATION MODULE

```
module define_arrays
   !part of program ctr2evs
   !written by Chris Rautman, SNL, 6113
   !defines certain arrays dynamically for use across main program and subroutine
   !                Variables (see also main program)
   !x(),y(),z()   coordinates of line segments defining contours
   !px(),py()  coordinates of intersections of line segments with angular rays
   !segno() identifier of line segment associated with px and py
```

```
   !indx()     ??used only in initial debugging???
   !-------------------------------------------------------------------------
   integer,allocatable,save::segno(:,:),indx(:),presrvSeg(:)
   real*8,allocatable,save:: presrvX(:),presrvY(:)
   real*8,allocatable,save::x(:,:),y(:,:),z(:),px(:,:),py(:,:)
end module define_arrays
```

FUNCTIONS

```
!miscellaneous functions and small subroutines for program "ctr2evs"
!written by Chris Rautman, SNL 6113                      11 June 2001

!*****************************************************************************
subroutine allocate_arrays(nimax,nj,nk)
   !provides for dynamic allocation of various arrays; requires "module"
   !note use of nimax+1 to allow closing of contours within program
   use define_arrays
   integer nimax,nj,nk
   allocate( x(nimax+1,nj+1),y(nimax+1,nj+1),z(nj+1),segno(nk,nj+1) )
   allocate( px(nk,nj+1),py(nk,nj+1),indx(nj+1) )
   allocate( presrvX(nj),presrvY(nj),presrvSeg(nj) )
end subroutine allocate_arrays


!*****************************************************************************
real*8 function Dpy( x1,y1,x2,y2 )
!uses Pythagorean Theorem to find a distance, Dpy
!                    Variables
!x1,y1   starting coords of line segment
!x2,y2   ending coordinates of line segment
implicit none
real*8 x1,y1,x2,y2
Dpy = dsqrt( (y2-y1)*(y2-y1) + (x2-x1)*(x2-x1) )
end function


!*****************************************************************************
real*8 function angle( m,x,x1)
!computes a 360-degree angle from a slope, m, and
! corrects for quadrant.  arcsine function [-pi,+pi]
!                    Variables
!m    slope value
!x,x1 starting and ending coordinates of contour line segment intersecting
!       the positive x-axis. Function REQUIRES ccw numbering of segments
!datand  double-precision arctangent function in degrees (intrinsic)
implicit none
real*8 m,x,x1

!perform some error checking
if( x1 .eq. x ) then
   write(*,'(a,f15.3)')' points X and X1 are identical; =',x
   write(*,'(a)')' Implicit no-slope m passed to function: angle'
   write(*,*) 'm = ', m
   stop
```

```
end if
if( m .eq. 0.0 ) stop 'Explicit no-slope m passed to function: angle'

!compute the angle
!write(*,'(a,f8.3)')' base angle =',datand(m)
if( m .ge. 0 ) then
   if( x1 .gt. x ) angle = datand(m)            !1st Quadrant
   if( x1 .lt. x ) angle = datand(m) + 180.0    !3rd Quadrant
else
   if( x1 .gt. x ) angle = datand(m) + 360.0    !4th Quadrant
   if( x1 .lt. x ) angle = datand(m) + 180.0    !2nd Quadrant
end if

end function
```

SUBROUTINE TRACE

```
subroutine trace( ctr,istart,istop,npts,n,dbg)
!written by Chris Rautman, SNL 6113
!  major modifications by CAR  11/22/2001
!  to account for "problem" (i.e., multiple-intersection) rays at the
!  start of the ray-processing sequence (angle = 0 degrees trigonometric)
!  issues with "wrap-around" or "cross-over" of line-segment numbering fixed
!part of program ctr2evs

!*************************************************************************
!                 Input Variables
!ctr     index of the current contour
!istart  last no-problem intersection node
!istop   next no-problem intersection node
!npts    number of new nodes to generate
!n       total number of nodes on this contour
!dbg     debugging flag (controls amount of output)
!segno() line-segment numbers; passed through define_arrays
!x(),y(),z()   line-segment coordinates; passed through define_arrays
!px(),py()  intersections of rays with the contour; passed through define_arrays
!            starting and stopping points for trace calculations
!presrvX()  very first intersection point on the contour with ray 1; substituted
!presrvY()   for undefined ray-intersection point when ray 1 has multiple intersects
!presrvSeg()   line-segment number assoc. w/ PresrvX,PresrvY; passed via define_arrays

!                 Output Variables
!ptX,ptY,ptZ() new nodal coordinates; written to file for later use
!               by main program (not passed in memory)

!                 Internal Variables
!ist, isp   starting and stopping line-segment numbers
!nsegs   number of line segments involved
!d()     distances along each line segment
!dtarget required distance between new nodal points
!dist    cumulative distance along contour to reach dtarget
!m       slope of a given line segment
!ang     angle in [0,360] associated with slope, m
!hyp     hypoteneuse distance for computation of dx, dy
```

```
!dx,dy   delta-x and -y from current line-segment starting point, used to
!          compute ptX, ptY
!flag    generic logical indicator
!xover   logical flag indicating cross-over of the line-segment numbering boundary
!         (positive x-axis intercept from center of contour bounding box)

!                 External Routines
!Dpy     Pythagorean distance function (custom)
!angle   determines ang in [0,360] from slope m based on starting and ending
!          points of the line segment (custom)

!**************************************************************************
use define_arrays
use msflib
implicit none
logical flag,xover,dbg
integer i,j,n,ctr,istart,istop,ist,isp,nsegs,npts
real*8 dist,dtarget,Dpy,m,ang,dx,dy,ptX,ptY,angle,hyp
real*8,allocatable::d(:)
xover = .false.

!**************************************************************************
write(21,'(2(a,i5),/,a,f8.1,a,i5)')' Trace;  starting ray:',istart, &
  ',  ending ray:',istop,'    Contour',z(ctr),'  Total # line segments in contour =',n
if( istop .lt. istart ) then
   write(21,'(a)')' >>> Possible problems: ending ray is lower number than starting ray'
   write(21,'(3(a,i3))')'     Contour:',ctr,'     istart=',istart,'    istop=',istop
end if
ist = segno(istart,ctr)
if( ist .lt. 1 ) then               !undefined, use original first intersection
   ist = presrvSeg(ctr)
   px(istart,ctr) = presrvX(ctr)
   py(istart,ctr) = presrvY(ctr)
   write(21,'(a,i4,a,2f12.1,a)')' restoring preserved first intercept for ISTART:', &
    ist,'  (',presrvX(ctr),presrvY(ctr),')'
end if
isp = segno(istop,ctr)
if( isp .lt. 1 ) then               !undefined, use original first intersection
   isp = presrvSeg(ctr)
   px(istop,ctr) = presrvX(ctr)
   py(istop,ctr) = presrvY(ctr)
   write(21,'(a,i4,a,2f12.1,a)')' restoring preserved first intercept for ISTOP:', &
    isp,'  (',presrvX(ctr),presrvY(ctr),')'
end if
nsegs = isp - ist + 1
if( nsegs .lt. 0 ) then
   !write( *,'(a)')'      warning: have crossed line-segment array boundary'
   write(21,'(a)')'      warning: have crossed line-segment array boundary'
   !we have crossed over the end of the line-segment array
   xover = .true.                   !for this call only!
   nsegs = n-ist+isp
end if
allocate( d(nsegs) )

!**************************************************************************
IF( xover .eqv. .false. ) THEN               !NO CROSS-OVER IS INVOLVED
!**************************************************************************
!This section of the subroutine is for no-crossover normal processing
```

```
!list the lengths of all involved line segments for comparison and reference
write(21,'(/,a,i5,a,2i5,3x,l1)')' there are',nsegs,' line segments; ist,isp:',ist,isp,
xover
if(dbg) then
   do i=ist,isp
      dist = Dpy( x(i,ctr),y(i,ctr),x(i+1,ctr),y(i+1,ctr) )
      write(21,'(a,i5,f12.3)')'    Total line-segment length: segment',i,dist
   end do
end if

!-------------------------------------------------------------------------------
!compute total distance along contour from last non-problem point to next non-problem
point
dist = 0.0
!first find the total distance along line segments from starting point to stopping point
if(dbg) write(21,'(/,2(a,2i5))')' checking between intersections, rays:',&
  istart,istop,';  line segments:',ist,isp

!compute the distances available for reallocation of points on contours
if( nsegs .eq. 1 ) then
   !distance between starting and stopping points that are on same line segment
   d(1) = Dpy( px(istart,ctr),py(istart,ctr),px(istop,ctr),py(istop,ctr) )
   dist = d(1)
   if(dbg) write(21,'(3x,a,f8.3)')' total distance between starting and stopping points
is',dist
else if( nsegs .eq. 2 ) then
   !starting and stopping points are on two successive line segments
   !distance from first point to end of associated line segment
   d(1) = Dpy( px(istart,ctr),py(istart,ctr),x(ist+1,ctr),y(ist+1,ctr) )
   if(dbg) write(21,'(3x,a,2f10.1,a,i4,f8.3)') 'distance from (',px(istart,ctr), &
     py(istart,ctr),') to end of segment',ist,d(1)
   dist = d(1)
   !distance from line-segment boundary to stopping point
   d(2) = Dpy( x(ist+1,ctr),y(ist+1,ctr),px(istop,ctr),py(istop,ctr) )
   if(dbg) write(21,'(3x,a,i4,a,2f10.1,t60,a,t65,f8.3)')' distance from start seg-
ment',&
     ist+1,' to (',px(istop,ctr),py(istop,ctr),')',d(2)
   dist = dist + d(2)
else if( nsegs .gt. 2) then
   !starting and stopping points are on line segments with intervening whole segments
   !compute distance from starting point to end of first line segment
   d(1) = Dpy( px(istart,ctr),py(istart,ctr),x(ist+1,ctr),y(ist+1,ctr) )
   if(dbg) write(21,'(3x,a,2f10.1,a,t60,i4,t65,f8.3)')' distance from
('',px(istart,ctr),&
     py(istart,ctr),') to end of segment',ist,d(1)
   dist = d(1)
   !measure along all full line segments
   do j=2,nsegs-1
      d(j) = Dpy( x(ist+j-1,ctr),y(ist+j-1,ctr),x(ist+j,ctr),y(ist+j,ctr) )
      if(dbg) write(21,'(3x,a,t60,i4,t65,f8.3)')' length of full line segment',ist+j-
1,d(j)
      dist = dist + d(j)
   end do
   !compute distance from beginning of associated line segment to stopping point
   d(nsegs) = Dpy( x(isp,ctr),y(isp,ctr), px(istop,ctr),py(istop,ctr) )
   if(dbg) write(21,'(3x,a,i5,a,2f10.1,a,t65,f8.3)')' distance from segment',isp,' to
('',&
    px(istop,ctr),py(istop,ctr),')',d(nsegs)
   dist = dist + d(nsegs)
```

```
else
   write(21,*)'Non-crossover processing: bad NSEGS number; should never get here'
   flag = commitqq(21)
   stop 'Abnormal termination, check documentation file'
end if

write(21,'(a,f12.3)')' Total linear distance from start to stop = ',dist
!allocate total distance to incremental distances between required points
dtarget = dist/float(npts+1)
write(21,'(a,f12.3,a,i3,a,//)')' Individual distances  = ',dtarget,'  times',&
  npts+1,' new intervals'

!*****************************************************************************
ELSE                                          !CROSS-OVER PROCESSING
!*****************************************************************************
!This section of the subroutine is for CROSSOVER processing

!list the lengths of all involved line segments for comparison and reference
write(21,'(/,a,i5,a,2i5,3x,l1)')' there are',nsegs,' line segments: ist,isp',ist,isp,
xover
if(dbg) then
   do i= ist,n-1                    !complete the "normal-sequence" listing of segments
      dist = Dpy( x(i,ctr),y(i,ctr),x(i+1,ctr),y(i+1,ctr) )
      write(21,'(a,i5,f12.3)')'    Total line-segment length: segment',i,dist
   end do
   do i = 1,isp                     !generate list from first line-segment to end of
problem
      dist = Dpy( x(i,ctr),y(i,ctr),x(i+1,ctr),y(i+1,ctr) )
      write(21,'(a,i5,f12.3)')'    Total line-segment length: segment',i,dist
   end do
end if

!-----------------------------------------------------------------------------
!compute total distance along contour from last non-problem point to next non-problem
point
dist = 0.0
if(dbg) write(21,'(/,2(a,2i5),a,i5)')' checking between intersections, rays:',&
  istart,istop,';  line segments:',ist,n-1,' and line segments    1',isp

if( nsegs .eq. 1 ) then          !crossover cannot be an issue on one line segment
   !distance between starting and stopping points that are on same line segment
   d(1) = Dpy( px(istart,ctr),py(istart,ctr),px(istop,ctr),py(istop,ctr) )
   dist = d(1)
   if(dbg) write(21,'(3x,a,f8.3)')' total distance between starting and stopping points
is',dist
else if( nsegs .eq. 2 ) then
   !starting and stopping points are on successive line segments that cross line-number-
ing bdry
   !compute distance from first point to end of the last-numbered line segment
   d(1) = Dpy( px(istart,ctr),py(istart,ctr),x(ist+1,ctr),y(ist+1,ctr) )
   if(dbg) write(21,'(3x,a,2f10.1,a,t60,i4,t65,f8.3)') ' distance from
(',px(istart,ctr), &
     py(istart,ctr),') to end of segment',ist,d(1)
   dist = d(1)
   !compute distance from first-numbered segment boundary to stopping point
   d(2) = Dpy( x(1,ctr),y(1,ctr),px(istop,ctr),py(istop,ctr) )
   if(dbg) write(21,'(3x,a,i4,a,2f10.1,a,t65,f8.3)')' distance from start segment',&
     1,' to (',px(istop,ctr),py(istop,ctr),')',d(2)
   dist = dist + d(2)
```

```
else if( nsegs .gt. 2) then
   !compute distance from starting point to end of first line segment
   d(1) = Dpy( px(istart,ctr),py(istart,ctr),x(ist+1,ctr),y(ist+1,ctr) )
   if(dbg) write(21,'(3x,a,2f10.1,a,t60,i4,t65,f8.3)')' distance from
('',px(istart,ctr),&
    py(istart,ctr),') to end of segment',ist,d(1)
   dist = d(1)
   !measure along all full line segments
   do j=2,nsegs-1
      if( ist+j .le. n ) then    !this is for the "normal" part of the processing
         d(j) = Dpy( x(ist+j-1,ctr),y(ist+j-1,ctr),x(ist+j,ctr),y(ist+j,ctr) )
        if(dbg) write(21,'(3x,a,t60,i4,t65,f8.3)')' length of full line segment',ist+j-
1,d(j)
         dist = dist + d(j)
      else                          !this is for processing the crossover and later
segments
         d(j) = Dpy( x(ist+j-n,ctr),y(ist+j-n,ctr),x(ist+j+1-n,ctr),y(ist+j+1-n,ctr) )
        if(dbg) write(21,'(3x,a,t60,i4,t65,f8.3)')' length of full line segment',ist+j-
n,d(j)
         dist = dist + d(j)
      end if
   end do
   !compute distance from beginning of associated line segment to last point
   d(nsegs) = Dpy( x(isp,ctr),y(isp,ctr), px(istop,ctr),py(istop,ctr) )
   if(dbg) write(21,'(3x,a,i5,a,2f10.1,a,t65,f8.3)')' distance from segment',isp,' to
('',&
    px(istop,ctr),py(istop,ctr),')',d(nsegs)
   dist = dist + d(nsegs)
else
   write(21,*)'CROSSOVER processing section: bad NSEGS number; should never get here'
   flag = commitqq(21)
   stop 'Abnormal termination, check documentation file'
end if

write(21,'(a,f12.3)')' Total linear distance = ',dist
!allocate total distance to incremental distances
dtarget = dist/float(npts+1)
write(21,'(a,f12.3,a,i3,a,//)')' Individual distances  = ',dtarget,'  times',&
  npts+1,' new intervals'

!*************************************************************************
END IF
!*************************************************************************

!-------------------------------------------------------------------------
!now compute individual distances to reach the required target distance and
! generate the new point(s) equally distributed along the original contour
j=1            !j tracks the number of line portions between required points
!               called "segments" but different than contour "line segments"
flag = .false.
do i=1,npts
   dist = 0.0
   !document initialization of distance tracking
   if(dbg) write(21,'(a,i3,28x,a,f12.3)')' point',i,' cumulative distance =',dist

   do while( j .le. nsegs )
      if( dist + d(j) .lt. dtarget ) then    !dtarget is the required distance between
points
```

```
         !we are not yet to the next point; add the current line-segment distance to
"dist"
          !and iterate to next line segment
         dist = dist + d(j)               !cumulative distance along contour
        if(dbg) write(21,'(2(a,i3),a,f8.3,a,f12.3)')' point',i,' segment',j,'    d =',&
          d(j),'   cumulative distance =',dist
         j=j+1
         flag = .false.
      else
         !solve for a point on current line segment
         hyp= dtarget - dist              !this is the remaining distance on this line
portion
         if( (x(ist+j,ctr)-x(ist+j-1,ctr)) .eq. 0.0 ) then     !trap no-slope lines
            write(21,'(a,i5)') ' >>> line segment with no slope: segment',ist+j-1
            write(21,'(a,2f12.1)') '     starting point:',x(ist+j-1,ctr),y(ist+j-1,ctr)
            write(21,'(a,2f12.1)') '     ending point:  ',x(ist+j,ctr),  y(ist+j,ctr)
            dx = 0.0
            dy = hyp
         else                                                  !we have a valid slope
           if( (ist + j) .le. n ) then                       !this is "normal" processing
               if(dbg) write(21,'(2(a,i4))') 'ist+j-1 =', ist+j-1,', ist+j =', ist+j
               m = (y(ist+j,ctr)-y(ist+j-1,ctr))/(x(ist+j,ctr)-x(ist+j-1,ctr))
               ang = angle(m,x(ist+j-1,ctr),x(ist+j,ctr))
            else                                                 !this is "crossover"
processing
               if(dbg) write(21,'(2(a,i4))') 'ist+j-n =',ist+j-n,', ist+j+1-n
=',ist+j+1-n
               m = (y(ist+j+1-n,ctr)-y(ist+j-n,ctr))/(x(ist+j+1-n,ctr)-x(ist+j-n,ctr))
               ang = angle(m,x(ist+j-n,ctr),x(ist+j+1-n,ctr))
            end if
            if(dbg) write(21,'(a,f8.3,a,f8.3)')' slope: ',m,',  angle: ',ang !,x(ist+j-
1,ctr),x(ist+j,ctr)
            dx = hyp*dcosd(ang)
            dy = hyp*dsind(ang)
         end if

         if( (j .eq. 1) .and. (flag .eqv. .false.) ) then
            !working from P(start)
            ptX = px(istart,ctr) + dx
            ptY = py(istart,ctr) + dy
            flag = .true.
         else if( flag .eqv. .false. ) then
            !working from start of preceding line segment
            if( ist+j .gt. n ) then     !crossover of line-segment numbering boundary
               ptX = x(ist+j-n,ctr) + dx
               ptY = y(ist+j-n,ctr) + dy
            else
               ptX = x(ist+j-1,ctr) + dx
               ptY = y(ist+j-1,ctr) + dy
            end if
         else
            !working on the same segment as previously
            ptX = ptX + dx
            ptY = ptY + dy
         end if
         !reset the available line-segment length
         d(j) = d(j) - hyp
        if(dbg) write(21,'(2(a,i3),a,f8.3,a,f12.3)')' point',i,' segment',j,'  hyp =',&
          hyp,'   cumulative distance =',dist+hyp
```

```
         if(dbg) write(21,'(a,i3,a,2f10.3)') point',i,' dx,dy:',dx,dy
         if(dbg) write(21,'(a,i3,a,2f10.1,2(a,f8.3),/)')' point',i,' is (',ptX,ptY,&
          '); dist available = ',d(j),'; hypoteneuse dist used =',hyp
         if( .not. dbg) write(21,'(a,i3,a,2f10.1,2(a,f8.3))')' point',i,' is
(',ptX,ptY,&
          '); dist available = ',d(j),'; hypoteneuse dist used =',hyp
         write(44,'(3f15.3,i5,a,i3,a)') ptX,ptY,z(ctr),ctr,' (',i,')'
         flag = .true.
         exit
      end if
   end do
end do

!-------------------------------------------------------------------------------
deallocate(d)
flag = commitqq(44)
write(21,'(a,/)')' end subroutine trace'
flag = commitqq(21)
return
end
```

Appendix B: Description of Dome-Margin File Format

This page intentionally left blank.

INTRODUCTION

The CD-ROM included with this report contains simple ASCII files produced by the MVS modeling software that describe the surface meshes defining the margins of the four SPR salt domes. These files should be viewable using any standard ASCII text editor, or be importable into a standard word processor.

Two different sets of files are included. First are the open-ended, quasi-cylindrical files produced by the software program **`ctr2evs`** directly from the digitized structure contour maps of the site characterization reports. These files demonstrate the output of the computer code implementing the newly developed modeling algorithm. The second set of files are the "final," or complete models of the various SPR salt domes. These files were generated by combining the **`ctr2evs`** output files describing the salt-dome flanks with a crestal surface generated by one or the other of the two algorithmic variants (page 11) for generating the upper portion of the salt dome in question. The two separately generated surfaces have been trimmed and merged using the capabilities of the MVS software, prior to be written-out as a single file containing the full representation of the salt-dome margin. It is these latter files that may be of greatest utility for use in other SPR project software to visualize the salt-dome geometries as modeled during site characterization.

The format of these files is a variant of a fairly typical finite-element-mesh description. Specifically, an initial section gives the spatial coordinates of the nodes of the mesh and each node is numbered. A second section of the file contains a description of the connectivity of those nodes (node 1 is connected to node 2, which is connected to node 3, which is connected to node 4, etc.). Only the node numbers forming a mesh element (here a quadrilateral) need be specified in this manner. Each mesh element is also given a unique number. A third section contains the numerical "property values" that are associated with each nodal position. Various interspersed lines specify the number of nodes, the number of mesh elements, and the names and units of measure for the material property values.

Because this report is concerned only with the shape of the dome-margin mesh, the material-property section of the files is somewhat superfluous. Accordingly, the only entries present in this section of the included files are the node number (referenced back to the nodal coordinates in section 1) and the elevation of those nodes. Thus the format of the mesh files is complete even though the elevation values in the material property section are redundant with the z-coordinate of the nodes themselves. Conceivably other material properties of interest could be included associated with the nodal locations (or with the polygonal mesh elements) for visualization.

The material that follows is the description of the *u*nstructured-*c*ell-*d*ata (UCD) file format, taken directly from the MVS software system's "help" files. The interested reader is also referred to the dome-margin UCD files that are included on the CD-ROM, although as "real" files, these are rather lengthy for illustrative purposes.

UCD File Format[1]

The format of UCD files is very similar to that commonly employed by node-based finite element models. Essentially, UCD files contain three different sections with a header line that describes the type and amount of data that is contained in each section. A UCD file cannot contain blank lines or lines with leading blanks. Comments can be present at the top of the file, but are not allowed in other parts of the file. Leading comment lines must begin with the "#" symbol. An example UCD file is presented below, with explanation lines describing the contents of the lines (which can not be present in actual UCD files) placed in brackets "{}" shaded differently, *and italicized*..... This UCD file consists of one hexahedral cell with 8 nodes, and each node has a single scalar data value.

***** *Begin Example UCD File* *****

```
# Lines with leading "#" characters in a UCD file can be comments.
# The user can include any number of comment lines.
```

*{The **first section** of data in the file begins with a header line, which describes the number of nodes in the model, the number of cells in the model, the number of nodal data components, the number of cell data components, and the number of model data components (cell and model data components are not currently used in EVS). The first section of the file looks like this:}*

```
8 1 1 0 0    {8 nodes, 1 cell, 1 nodal data component, 0 cell data, 0 model data}
   1  0.000   0.000   1.000  {node 1, xcoord = 0, ycoord = 0, zcoord = 1]
   2  1.000   0.000   1.000  {node 2, xcoord = 1, ycoord = 0, zcoord = 1]
   3  1.000   1.000   1.000  {node 3, xcoord = 1, ycoord = 1, zcoord = 1]
   4  0.000   1.000   1.000  {and so forth for the 8 nodes in this hexahedron]
   5  0.000   0.000   0.000
   6  1.000   0.000   0.000
   7  1.000   1.000   0.000
   8  0.000   1.000   0.000
```

*{The **second section** of the file contains the list of elements in the model, the material type (or layer) of each element, the type of element (EVS usually uses quads and hexahedrons), and the nodes that make up the vertices of the element, listed in order of connectivity starting from the upper left vertex, and progressing counterclockwise for each layer of vertices in the element (see the diagram below). Remember these lines starting with brackets are for explanation only and cannot actually be present in the file.}*

```
1  1   hex 1 2 3 4 5 6 7 8{Element 1, Material 1, Hexahedral Element Type,Node numbers
for nodes at vertices of element. This element has 8 vertices, at nodes 1 through 8.
Note that the nodes were identified by number in the first section of the file, and
along with their X, Y, Z, coordinates}
```

*{The **third section** of the file contains a first line that lists the number of data components for each node, and then a list of the number of properties belonging to each component. Most EVS modules output "scalar" properties, which by definition have only one property per data component.*

[1] This entire section is taken virtually verbatim from the MVS help system, ©1994-2002 C Tech Development Corporation, under the topic "UCD File Format." It has been edited only slightly to fit the current circumstances. Some text has wrapped to a second or additional lines. Remember that blank lines are not allowed in an actual UCD file.

However, the MT3D model outputs a vector component describing the velocity of fluid flow at the model nodes, which has the three properties of X, Y, Z velocity present in one data component. An example of this type of "vector" data component is presented below.}

```
   1 1    {One data component, One property in the first component, "a scalar" The next
n lines of the third section of the file contain text entries separated by commas, that
describe the name and units of the property in each data component.  There are as many
of these text lines as there are data components.}
stress, lb/in**2  {Data Component 1 is the Stress property, Units are lb/in**2}
```

{The last n lines in the third section of the file list the node numbers, and the values for the properties in each data component. There are as many lines as there are nodes in the model.}

```
   1  4999.9999      {Node 1, 4999.9999 (Stress Property Value in lb/in**2)}
   2  18749.9999     {Node 2, 18749.9999 (Stress Property Value in lb/in**2)}
   3  37500.0000     {Node 3, and so on ...}
   4  56250.0000
   5  74999.9999
   6  93750.0001
   7  107500.0003
   8  5000.0001
```

{The last node number and list of data component property values is the last line of the file.}

***** End Example UCD File *****

A more complex example of a UCD file is provided below. … Note that this UCD file has no header lines, and has a vector property in the last nodal data component (in the third section of the file). Again, some explanations of the file structure are provided, which are enclosed in brackets "{}" and colored magenta. These are not in the actual UCD file.

***** Begin Example UCD File *****

```
22134   10500      12       0       0    {Line 1 of Section 1; 22134 nodes, 10500
                                           elements, 12 nodal data components}
     1  9625.112 10786.380     6.275    {Node 1, 9625.112 X Coord, 10786.380 Y Coord,
                                           6.275 Z Coordinate}
     2  9714.992 10742.550     6.367
     3  9804.871 10698.710     6.581
     4  9894.751 10654.870     6.939
     5  9984.630 10611.030     7.581
```

{Data lines for nodes 6 through 22130 omitted from this example}

```
 22131 12534.330  6029.657   -62.443
 22132 12624.210  5985.820   -62.879
 22133 12714.090  5941.983   -63.324
 22134 12803.970  5898.146   -63.318       {Node 22134 is last in model}
   1   0 hex      1    52    53     2  1582  1633  1634  1583{Line 1 of Section 2;
   2   0 hex      2    53    54     3  1583  1634  1635  1584 Element 1, Material 0,
   3   0 hex      3    54    55     4  1584  1635  1636  1585 Hex Elem., Nodes at
   4   0 hex      4    55    56     5  1585  1636  1637  1586 the vertices of element
```

```
   5   0 hex      5    56    57     6  1586  1637  1638  1587 1 are 1,52,53,2,1582,
                                                              1633,1634,1583}
```

Data lines for elements 6 through 10494 omitted from this example}

```
 10495   6 hex  20496 20547 20548 20497 22077 22128 22129 22078
 10496   6 hex  20497 20548 20549 20498 22078 22129 22130 22079
 10497   6 hex  20498 20549 20550 20499 22079 22130 22131 22080
 10498   6 hex  20499 20550 20551 20500 22080 22131 22132 22081
 10499   6 hex  20500 20551 20552 20501 22081 22132 22133 22082
 10500   6 hex  20501 20552 20553 20502 22082 22133 22134 22083  {Element 10500,
                                                                 Material 6...
                                                                 Last in model}
10 1 1 1 1 1 1 1 1 1 3  {Line 1 of Section 3; 10 Data components, Comp.1 through 9 have
                          1 property, Comp 9 has 3}
head, feet           {First data component is head in feet (a scalar property)}
drawdown, feet       {Second data component is drawdown in feet}
thickness, feet      {Third data component is layer thickness in feet}
Geolayer, number     {Fourth data component is Geolayer number, which specifies which
                      element material it beongs to}
Elevation, feet      {And so forth...}
Conductivity, feet/day
Storage, unitless or 1/feet
Concentration, M/L^3
Change in Conc, M/L^3
Velocity, L/t        {Velocity has three components, Vel in X, Vel in Y, Vel in Z, in
                      L/t, a vector data component}
   1  .890  .000  19.1  0  6.28  15.0  .250  .742E-07  .000     .000     .750E-02  .000
{Node 1, properties as listed above}
   2  .890  .000  19.6  0  6.37  15.0  .250  .000      .000     .000     .590E-02 .000
   3  .895  .000  20.6  0  6.58  15.0  .250  .286E-03 -.143E-03 -.750E-02 .115E-01 .000
   4  .900  .000  21.2  0  6.94  15.0  .250  .341E-03 -.314E-03  .000     .111E-01 .000
   5  .900  .000  21.2  0  7.58  15.0  .250  .756E-05 -.171E-03  .000     .101E-01 .000
```

Data lines for nodes 6 through 22129 omitted from this example}

```
 22130 .735 .000 3.77  6 -63.0 105. .100E-05 .295E-02 -.667  -.525E-01 -.732E-01  .000
 22131 .740 .000 3.85  6 -62.4 105. .100E-05 .504E-03 -.947   .000     -.330E-01  .000
 22132 .740 .000 4.14  6 -62.9 105. .100E-05 .352E-06 -.733   .000      .501E-02  .000
 22133 .740 .000 4.44  6 -63.3 105. .100E-05 .000     -.644   .000      .525E-01  .000
 22134 .740 .000 4.45  6 -63.3 105. .100E-05 .000     -.643   .000      .525E-01  .000
{Node 22134, last in model}
```

***** End Example UCD File *****

DISTRIBUTION:

U.S. Department of Energy (via CD-R only)
Strategic Petroleum Reserve Project Management Office
900 Commerce Road East
New Orleans, LA 70123

U.S. Department of Energy (3)
Strategic Petroleum Reserve
1000 Independence Avenue, SW
Washington, D.C. 20585
Attn: D. Johnson, FE-421

Sandia Internal:

MS 0701 P.B. Davies, 6100
MS 0741 Margie Tatro, 6200
MS 0706 D.J. Borns, 6113
MS 0706 B.L. Ehgartner, 6113
MS 0706 B.L. Levin, 6113
MS 0706 C.A. Rautman, 6113 (5)
MS 0706 A.R. Sattler, 6113
MS-0706 S. Wallace, 6113, for SPR library
MS-0735 R.E. Finley, 6115
MS 0750 T.E. Hinkebein, 6118
MS-1395 J.S. Stein, 6821 (5)
MS 9018 Central Tech. Files, 8945-1
MS 0899 Technical Library, 9616 (2)