DOE/MC/31182 -- 5201
(DE96004482)

Fractal Modeling of Natural Fracture Networks

Final Report
June 1994 - June 1995

Martin V. Ferer
B.H. Dean
Charles Mick

April 1996

Work Performed Under Contract No.: DE-FG21-94MC31182

For
U.S. Department of Energy
Office of Fossil Energy
Morgantown Energy Technology Center
Morgantown, West Virginia

By
West Virginia University Research Corporation
Morgantown, West Virginia

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, 175 Oak Ridge Turnpike, Oak Ridge, TN 37831; prices available at (615) 576-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161; phone orders accepted at (703) 487-4650.

DOE/MC/31182 -- 5201
(DE96004482)
Distribution Category UC-132

Fractal Modeling of Natural Fracture Networks

Final Report
June 1994 - June 1995

Martin V. Ferer
B.H. Dean
Charles Mick

Work Performed Under Contract No.: DE-FG21-94MC31182

For
U.S. Department of Energy
Office of Fossil Energy
Morgantown Energy Technology Center
P.O. Box 880
Morgantown, West Virginia 26507-0880

By
West Virginia University Research Corporation
Department of Chemical Engineering
P.O. Box 6845
Morgantown, West Virginia 26506-6315

April 1996

Objectives

West Virginia University will implement procedures for a fractal analysis of fractures in reservoirs. This procedure will be applied to fracture networks in outcrops and to fractures intersecting horizontal boreholes. The parameters resulting from this analysis will be used to generate synthetic fracture networks with the same fractal characteristics as real networks.

Background

Recovery from naturally fractured, tight-gas reservoirs is controlled by the fracture network.[1] Reliable characterization of the actual fracture network in the reservoir is severely limited. The location and orientation of fractures intersecting the borehole can be determined, but the length of these fractures cannot be unambiguously determined. Fracture networks can be determined for outcrops, but there is little reason to believe that the network in the reservoir should be identical because of the differences in stresses and history. Seismic techniques do provide some large scale (resolution of tens or hundreds of feet) information about the fracture density and average fracture orientation, although there is some controversy about interpretation of the multi-component surface seismic data, especially regarding which layer is being probed.

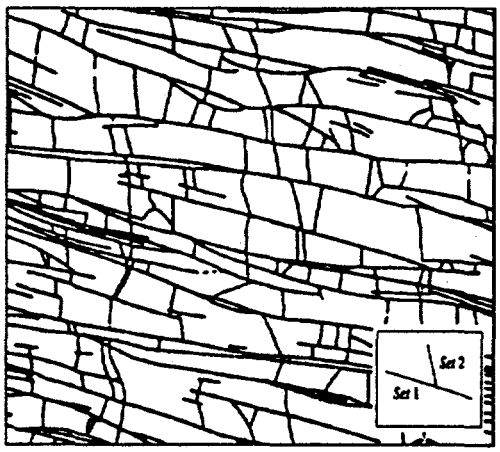

Fig. 1 Outcrop Fractures at MWX site. Shows the primary fractures (Set 1) and the secondary fractures (Set 2).

Furthermore, independent of the assumption of fractal behavior, it is known that typical fractures in the second set should begin and end at fractures of the first set.[2] This effect is commonly observed in real fracture networks from outcrop studies, for example 92% of the secondary fractures in the MWX outcrop (**Fig.** 1) satisfy this criterion.[3] Imposing this constraint upon the secondary

fractures increases the visual similarity between our networks and the real network over simulated networks from other fractal modeling schemes.[4]

Because of the lack of detailed information about the actual fracture network, modeling methods must represent the porosity and permeability associated with the fracture network, as accurately as possible with very little apriori information. Three rather different types of approaches have been used: i) dual porosity simulations, ii)'stochastic' modeling of fracture networks, and iii) fractal modeling of fracture networks. The dual porosity approach is a natural extension of the gridding schemes widely used in describing reservoirs, however in assuming mesoscopic scale (tens or hundreds of feet) averages of fracture porosities and permeabilities, they may be smoothing the very heterogeneities which control the recovery. This may limit reliability for strongly anisotropic fracturing. That is, even if fractures are located randomly throughout the grid-block so that an average porosity may be sensible, the conductivity of similar fractures differ widely invalidating assumptions of an average permeability.

Stochastic models which assume a variety of probability distributions of fracture characteristics have been used with some success in modeling fracture networks.[5-7] The advantage of these stochastic models over the dual porosity simulations is that real fracture heterogeneities are included in the modeling process. On the other hand these stochastic models need information about all features of the actual fracture network to provide the most accurate modeling. In the highest level (most accurate) model for each set of fractures with a given orientation, one needs to determine the probability distribution of i) the location of independent fractures ii) the location of fracture clusters or swarms iii) locations of fractures within clusters, iv) cluster lengths, v) fracture lengths, vi) fracture apertures, and vii) fracture orientations. The less reliable the information determining these probability distributions; the less reliable the fracture network. Reliable information about many aspects of the real fracture network is impossible to determine; the assumption of self-similar fractal behavior (if valid) enables us to predict features of one aspect of the distribution from other aspects of the distribution; i.e. i), ii), and iii) result from the box-counting along the borehole which, in turn, predicts features of the distributions for iv), v), and vi) for self-similar fractal networks.

Aspects of fractal geometry have been applied to mimic the heterogeneity associated with layering in real reservoirs for a number of years. In these cases, the variation in permeability with height at the borehole was found to obey fractal statistics,[8] and the correlations implicit in fractal geometries allowed them to interpolate between the known permeabilities at the borehole in such a way that results from flow models agreed with analyses of production logs and tracer breakthrough.[9] Examples in the open literature reporting the use of fractal geostatistics to treat naturally fractured reservoirs are less common.[4,10] If a set of natural fractures is described by a self-similar fractal geometry, the self-similar, scale invariance of the fracture network implies relationships among the fracture distribution, and the various length scales: clustering or fracture correlation, fracture aperture, and fracture length. Therefore, if fracture networks obey a self-similar fractal geometry, borehole data locating orientational sets of fractures, will enable a determination of the fractal dimension and 'lacunarity'. This along with relatively generic information about the typical aperture size and length of fractures,[1] will allow us to produce a self-similar fractal network. The clustering occurs naturally in the fractal network because of the correlations inherent in fractal geometries. The fractal parts of the aperture size and length distributions (even the fracture shape distributions) should be the same as the fractal parts of the fracture location vs. scale distributions.

In the sections following this introduction, we will i) present 'fractal' analysis of the MWX site, using the box-counting procedure[11,12]; ii) review evidence testing the fractal nature of fracture distributions and discuss the advantages of using our 'fractal' analysis over a stochastic analysis; iii) present an efficient algorithm for producing a self-similar fracture networks which mimic the real MWX outcrop fracture network.

Project Description-Fractal Analysis

Illustrative Example Before analyzing the MWX outcrop (**Fig.** 1), one must understand the box-counting procedure used in these tests as well as our method for generating the fracture networks. As discussed later in this section, the box-counting procedure automatically reproduces the random aspects of the distribution of fractures in addition to reproducing the clustering obvious in **Fig.** 1.

For a simple example of the box-counting procedure consider the distribution of fractures intersecting a length of borehole (**Fig.**2a). To determine the fractal dimension as well as the range of size scales over which the distribution is fractal, one covers the array of fractures by successively smaller and smaller rulers (one-dimensional 'boxes'), and then one counts the number of 'boxes' or rulers covering one or more fractures. If the distribution has a fractal dimension D_f over a range of sizes, then

Fig. 2a Fractures intersecting a borehole.

$$N = A(\Delta)^{D_f}, \tag{1}$$

where N is the number of rulers which cover fractures, the constant A is called the lacunarity, and the scale Δ determines the length of the rulers (L/Δ). If one covers the 24 fractures in **Fig.**2 by a ruler of length L , (shown at the bottom of **Fig.**2b) one ruler covers the fractures; with two rulers of length $L/2$ (near the bottom of **Fig.**2b) both cover fractures; with four rulers of length $L/4$ all 4 cover fractures, but with 8 rulers of length $L/8$, only 6 cover fractures. This is continued down to 128 rulers of length $L/128$ as shown in **Table I**.

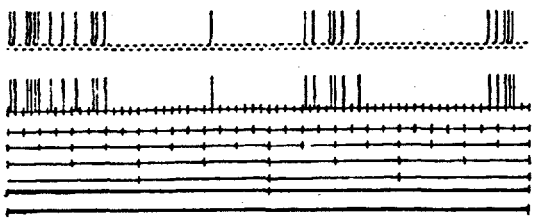

Fig. 2b The lower half of the figure shows the fractures in Fig. 2a with the scale rulers 'covering' the set of fractures from a ruler of length L, proceeding upwards to rulers of length $L/64$ just below the fractures. The top half shows the same set of 'covering' rulers of length $L/128$.

Since there are only 24 fractures, at scales smaller than $L/128$, there will only be 24 rulers covering fractures. A log-log plot of the box-counting for **Fig.** 2 is shown in **Fig.**3 below. The fractal relationship is given by the solid line $N = 2.12\ \Delta^{0.5}$, except at large and small scales for the following reasons. At small Δ, (coarse scales L, $L/2$ and $L/4$), N equals the number of rulers ($N = \Delta$) because all the rulers cover fractures. In later sections we refer to this as the initial covering **I** regime. At very large Δ, (very fine scales $L/256$ and $L/1024$), only 24 rulers are covered because there are only 24 fractures and there is no more detail in the fracture pattern - so that the box-counting 'cuts-off' or

'saturates' at 24. Hence, for this fracture pattern, the pattern is fractal between the initial covering and cutoff regimes (over the range of scales $\Delta = 8$ to 128) with a fractal dimension of 1.5 and a lacunarity of 2.12. The ruler counting for this 1-d slice of the 2-d fracture network gives an exponent: $D_f - 1$, i.e. the actual 2-d fractal dimension minus one.

Table I

Length of Rulers (L/Δ)	Number of Rulers Covering Fractures (N)
L	1
$L/2$	2
$L/4$	4
$L/8$	6
$L/16$	8
$L/32$	13
$L/64$	17
$L/128$	24
$L/256$	24
$L/1024$	24

Before continuing, it should be pointed out that this fracture pattern was generated by our algorithm to have a lacunarity of 2.12 and a fractal dimension of $1+\frac{1}{2}$ over the range of scales from $L/8$ to $L/128$. The algorithm used to generate this pattern is described in a following section.

It is important to realize, however, that if the distribution of fractures in **Fig.** 2a were completely random (i.e., if there were no clustering of fractures) the points from the box-counting would obey a linear relationship ($N = \Delta$) up to cutoff. That is, each box (on the average) would contain one fracture up to the total number of fractures (in this case $N_{total} = 24$); at finer scales the one fracture would randomly occupy one of the smaller boxes. However, because of clustering, groups of fractures are closer together than average. Therefore, when using the box-counting procedure, the linear regime ends before $N = N_{total}$; and one enters the 'fractal' or clustering regime where some boxes are empty and others have several fractures much closer together than average. The box-counting procedure then provides a method for characterizing (and thus for reproducing) this clustering.

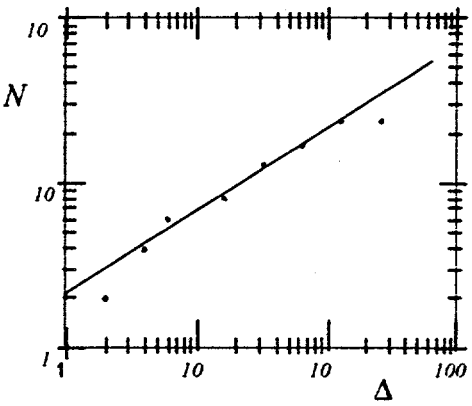

Fig. 3 Fractal Plot for **Fig.**2.

Results

MWX Outcrop First the primary set of fractures in **Fig.**1 were analyzed. A series of eight lines (boreholes) of length L were drawn through the set of primary fractures, and the box-counting procedure was used on each of these boreholes. The results for the number of boxes covering fractures vs. the scale Δ is shown in **Fig.**4.

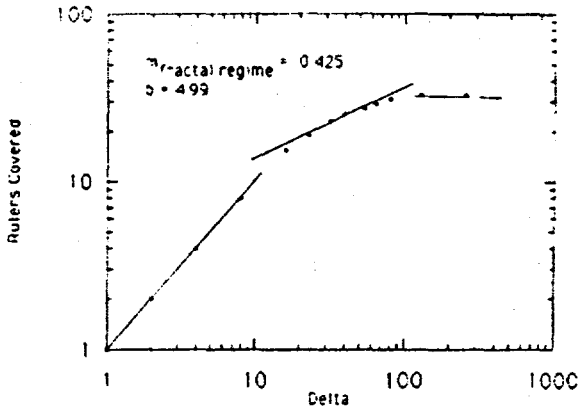

Fig. 4. or the primary fractures, the box-counting from the 'boreholes' on the MWX outcrop (**Fig.**1), shows the initial covering (the linear increase, $N = \Delta$, up to the clustering or fractal regime), the fractal regime, and the cutoff regime.

The initial covering regime persisted until scale 16. The cutoff regime began at scale 80. In-between the data are well represented by the fractal power law $N = 4.9\ \Delta^{0.425}$, indicating a fractal dimension $D_f \approx 1.43$. It should be noted that at intermediate scales the simple doubling rule: $\Delta = 2^n$, (c.f. **Figs.** 2 & 3 and **Table I**) was used to provide more data in the fractal regime.

The secondary set of fractures in **Fig.**1 were analyzed in the same way. A series lines of length L, perpendicular to these secondary fractures, were drawn through the secondary fractures, the box-counting was performed and the values $N(\Delta)$ were averaged. **Fig.**5 shows the plot of N vs. Δ and indicates that for these secondary fractures the initial covering regime persists until scale 6 - and that the cutoff regime begins at scale 40. Between the number of rulers the fractal power law: $N = 3.47\,\Delta^{0.343}$ applies; indicating a fractal dimension $D_f \approx 1.34$. Again, intermediate scales were used to provide more data in the fractal regime.

To determine the length distribution from the data provided by M. McKoy,[3] we plot the total number of fractures with lengths greater than a given length L, $N(L)$, vs. L. It should be noted that this total number $N(L)$ with lengths $\ell \geq L$ is the integral of the number density of fractures $n(\ell)$ with length ℓ integrated from $\ell = L$ up to the one fracture of maximum length ;

$$\text{i.e.} \quad N(L) = \int_L^{L_{max}} n(\ell)\, d\ell .$$

This graph of the data is shown in **Fig.**6. It is fit by the characteristic exponential cutoff for the greatest lengths $L \geq 14$, and by a fractal power law for the smallest lengths ($4 \leq L \leq 14$). For a self-similar fractal fracture network, the number density should be given by $n(\ell) = \eta \ell^{-D_f}$; so the total number is:

$$N(L) = \frac{\eta}{(1 - D_f)} L^{1 - D_f} \quad 11$$

Hence, the data is consistent with a fractal dimension: $D_f = 1.48$.

This data does not determine unambiguously whether or not the clustering regime is rigorously fractal. That is, this data does not favor a strictly power law regime (i.e. fractal behavior) between the linear, initial covering regime, and cutoff. However, the power law assumption used to draw the lines does represent a good fit to the box-counting data. Therefore, at worst, by assuming that the intermediate regime is fractal, we may be

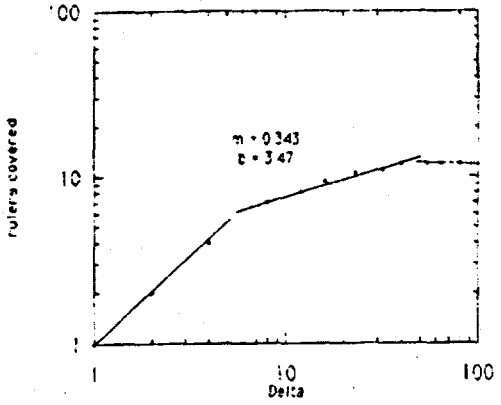

Fig. 5 For the secondary fractures, the box-counting from the 'boreholes' on the MWX outcrop (Fig.1), shows the initial covering (the characteristic linear regime), the fractal regime, and the cutoff regime.

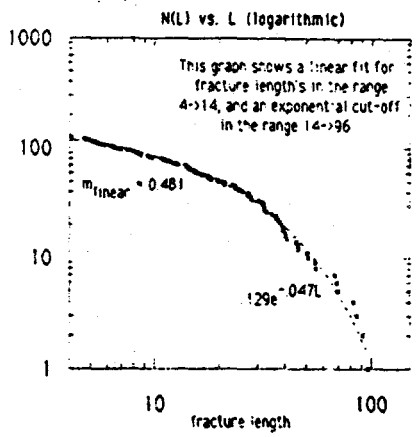

Fig. 6. The number of fractures $N(L)$ with lengths greater than L plotted against L. This shows the exponential cutoff for the larger lengths and the fractal regime for the smaller lengths.

only providing a good approximation to the data. If the assumption of fractal clustering only provides a good approximation to the true clustering, our simulated fracture networks will represent a good approximation to the actual fracture network - which is all that is necessary.

On the other hand, it is encouraging that the power laws from the box counting and length distributions are all consistent with the same fractal dimension: $D_f \approx 1.4$, to within a realistic uncertainty from the data fitting. This equality of fractal dimensions from all length measures is the hallmark of self-similar fracture networks.

A program to carry out the box-counting procedure and return the fractal dimension and lacunarity has been developed in order to process multiple sets of data from various boreholes. To test these programs as well as the routines for simulating the fracture networks, numerous trial runs have been performed to analyze the "borehole fractures" from simulated networks.

Box-counting Results and Analysis - (Robert Pietsch #1) Horizontal Borehole Data

The box-counting program performs three major functions: i) MWX data scan, ii) implements the box-counting algorithm, and iii) performs a least squares analysis. The data scan function is used to read in the data and can be easily modified for different data formats. The least squares analysis is an optional function - and was not used in this report since the box-counting results were transferred to a an external file and processed graphically instead of numerically.

In practice, the initial orientation and ruler length of the borehole sample (from the MWX data set) are specified by the user. These parameters are then modified to carry out the box-counting procedure. The degree of deviation from the borehole is then checked and stored in a generic array. If this value is too low, the corresponding fracture orientation is not included in the final analysis of the box-counting procedure. The ruler increment variable is initialized using the value: $\frac{initial\ ruler\ length}{\Delta}$. Delta is initially set equal to one and then doubled at each iteration. While the ruler increment is less than the modified initial ruler length, a check is performed to see if the number from the fracture

orientation column falls within the specified range. If so, the variable '*rulers_covered*' is incremented by 1. The loop continues until the value of '*rulers_covered*' equals the total number of fractures; indicating that saturation has been reached. These values may then be stored in a separate datafile for further graphical analysis.

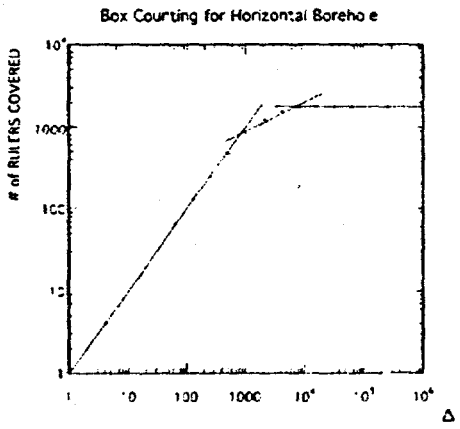

Fig. 7 Box-counting results.

The results for a typical box-counting run using the **Robert Pietsch data** are shown in **Fig.**7. Our analysis shows that in the fractal regime: $\Delta = 800 \rightarrow 7000$, the total number of fractures is given by: $N \approx 76.5\ \Delta^{-0.36}$, implying that $D_f \approx 1.4$.

Are Fracture Networks Fractal?

There is evidence that real fracture networks are fractal both in outcrops where Barton and others found a fractal dimension of $D_f \approx 1.55$, for different fracture systems,[13] as well as from underground data in the Fanay-Augères uranium mine[10] where they found a varying fractal dimension. The variation in their fractal dimension may result from use of too great a range of scales. As we saw for very large scales, all the rulers are covered so their finding a 'fractal dimension' of 2 at large scales is not surprising. Similarly, at very small scales one approaches a limit where the number of 'boxes' covered equals the number of fractures so the fractal dimension approaches 1; this may be an artifact of the neglect of small aperture fractures (micro-cracks which may be significant in determining number at their 0.005 meter scale).

The length of the fractures has been found to be fractal,[14] and the shape of the fractures has also been determined to be fractal[15-17]. This suggests that all features of the fractures may be fractal: distributions of i) centers, ii) lengths, iii)widths, and iv) shapes. The evidence that the shapes are fractal suggests that porosities and permeabilities may also obey fractal statistics. If all geometrical aspects of the fracture distribution are fractal

with the same fractal dimension, the fracture distribution is self-similar. This may seem to be a very unusual occurrence, but in fact many examples of development (or growth) which occur in random media (like the development of fractures in stressed rock formations) have a self-similar geometry. The first level of our geostatistical modeling will assume a self-similar fractal geometry for the fracture distribution. Higher levels of our geostatistical modeling could use actual measurements to determine the fractal distribution of (e.g.) the fracture widths.

The Fracture Generation Algorithm

Here we describe the implementation and design of an algorithm that was developed to generate a 2-d fracture networks. As we have discussed, the primary assumption in our model is that the network geometry is fractal - i.e. has a self-similar or scale invariant geometry. Using this information we have developed a program to generate complete 2-d fracture outcrop networks using only the *lacunarity*, *fractal dimension*, *initial covering*, and *cutoff* parameters obtained from MWX data.

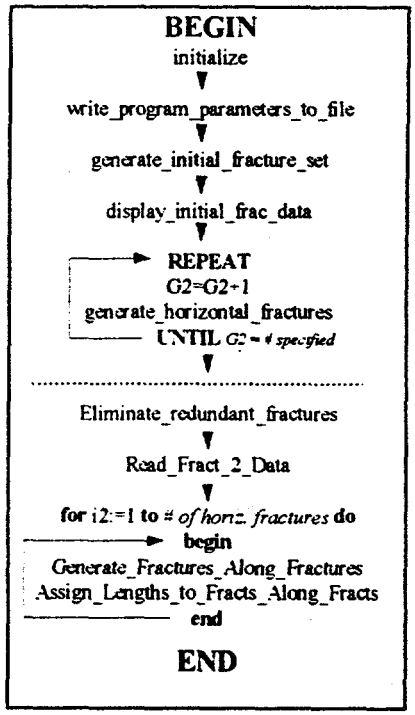

Fig. 8. Procedure Flowchart for *2d_frac.pas*

The PASCAL programming language was chosen to emphasize both modularity and structure in the development of the algorithm. Since the PASCAL syntax is completely analogous to *pseudocode* used in general algorithm descriptions, the program can be easily modified by others or converted to another programming language at a later time.

The algorithm is most clearly described by reference to the procedure flowchart in **Fig.8**. The body of the program (**Appendix A**) consists primarily of variable and procedure declarations whose execution begin on p.36. The procedures listed on p.36

(and in **Fig.**8) define the highest level of program hierarchy. All other procedures declared in the body of the program are called from within these procedures.

The most general description of the program is obtained by examining **Fig.**8. In the broadest sense the program performs 2 tasks (separated by the dotted line):

(1) Generates a horizontal fracture set.

(2) Generates a vertical fracture set - consistent with the fracture set in (1)

To generate the horizontal fracture set the program first generates a 1-dimensional fracture set along a left-justified line extending downward in the vertical direction (see **Fig.**10). This is accomplished by the procedure *GENERATE_INITIAL_FRACTURE_SET* whose flowchart is given in **Fig.**9. The first step in the procedure initializes the first row in the 2-dimensional *ruler* array L[*j*,*k*], where *i*=1,2 and *k* can range from 1 to $2^{13} = 8192$ as declared using the TYPE and VAR clauses at the beginning of the program. The range of the **for** loop given by the variable *Ri* is the initial number of rulers chosen to cover the fracture set in a 1-1 ratio. If a fracture is covered by a ruler, than the value of the array corresponding to this specific ruler is given the value 1. Conversely, an empty ruler site is given the value 0.

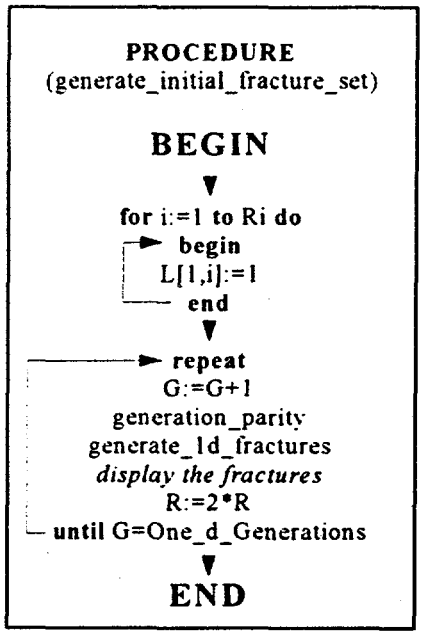

Fig. 9 Flowchart for the *generate_initial_fracture_set* procedure.

Having initialized the *L*[1,*i*] array the *GENERATE_INITIAL_FRACTURE* procedure divides each ruler into two new rulers (by mapping each ruler variable in *L*[1,*i*] to two new ruler variables in *L*[2,*i*]). To accomplish this task the procedure begins the *repeat...until* loop shown in **Fig.**9 and increments the counting variable *G* (initially = 0) to the value of 1. The *GENERATION_PARITY* procedure then determines if *G* is odd or even and assigns the variables *e* and *f* the values (1 and 2) or (2 and 1) respectively, depending on whether G is odd or even.

Next, the procedure randomly chooses one of the 2 new rulers in $L[2,k]$ for each of the rulers in $L[1,i]$ and assigns this ruler a value of 1 - while giving the other ruler a value of 0. In this way, the covered fractures of the initial level are brought down to the next level of $2 \times R_i$ rulers. The remaining rulers are then assigned fractures according to the distribution:

$$N = l\,\Delta^{D_f - 1}, \tag{2}$$

where N is the number of fractures, l is the lacunarity, D_f is the fractal dimension, and $\Delta = \frac{(\textit{total length of fracture set} = 1)}{\textit{number of rulers}}$. The progress of the algorithm is checked by displaying the fracture locations graphically as the program is running. After a single pass through the loop the number of rulers, R, is doubled and the whole process begins again. At the next iteration the parity of G will change, as will the values of e and f according to the previous assignment. Using the mapping:

$$L[e,i] \rightarrow L[f,j],$$

the values of $L[2, j]$ provide input for the next iteration and the values of $L[1, i]$ are replaced with new fracture assignments in the next ruler doubling. The output at this stage of the program is shown in **Fig.**10 and is analogous to **Fig.**2a. Using $D_f = 1.5$, $l = 2.12$, with an initial covering of 4, we obtain 24 fractures with $4 \times 2^5 = 128$ rulers after 5 generations. For the **Robert Pietsch data** (c.f. **Fig.**7), the 1-d fracture output is shown in **Fig.**11 using $D_f = 1.4$, $l = 76.5$, with an initial covering of 800.

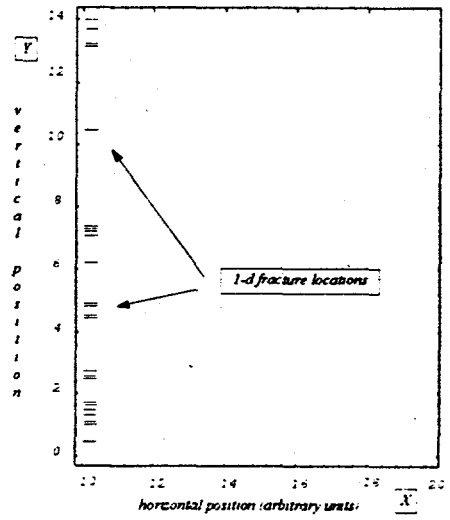

Fig. 10. 1-d fracture generation output. The data is represented graphically by a series of small line segments with extension in the x direction.

Continuing with the generation of the horizontal fracture set the program enters the *repeat...until* loop shown directly above the dotted line in **Fig.**8 and increments the counting variable *G2* (initially = 0) to the value of 1. The loop executes the:

GENERATE_HORIZONTAL_FRACTURES

procedure to produce a vertical fracture set for each value of the grid step in the x direction. **Fig.**12 shows the procedure flowchart:

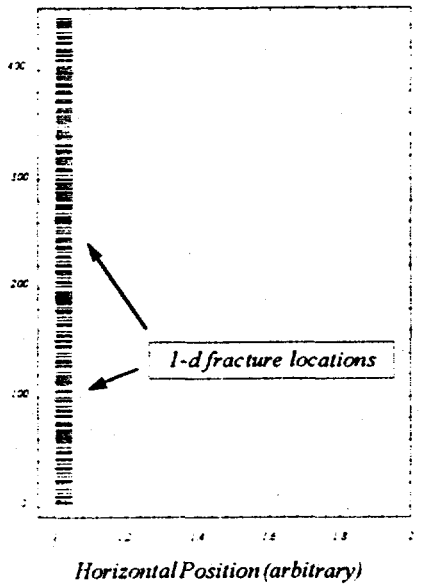

Fig. 11. 1-d fracture generation output using Robert Pietsch data.

In the first iteration (G2=1) the procedure assigns a length (extension in the x-direction) to each fracture site. To obtain the fracture length we assume a probability density function given by

$$p(L) = A\,L^{1-D_f}, \qquad (3)$$

where L is the fracture length and A is a constant. The probability that a given fracture will have a length $\le L'$ (greater than 2 arbitrary units) is then given by the distribution function:

$$P(2 < L \le L') = \int_2^{L'} A\,L^{1-D_f}dL = \frac{A}{2-D_f}\left[L'^{(2-D_f)} - 2^{(2-D_f)}\right]. \qquad (4)$$

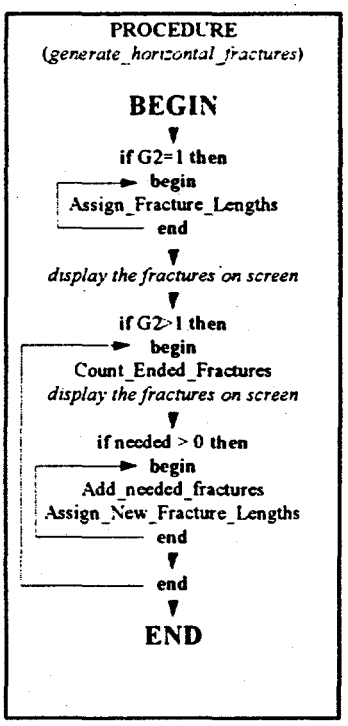

Fig. 12. Procedure flowchart for *generate_horizontal_fractures*

Since the fracture must have a length between 2 and 100 (according to the distribution), the constant A is determined from

$$P(2 < L \le 100) = 1 = \int_2^{100} A\,L^{1-D_f}dL\,, \qquad (5)$$

so that

$$A = \frac{(2-D_f)\,100^{(D_f-2)}}{\left[1-\left(\frac{2}{100}\right)^{(2-D_f)}\right]}\,. \qquad (6)$$

Substituting (6) into (4) then gives

$$P(L') = \left[\frac{1 - \left(\frac{2}{L'}\right)^{(2-D_f)}}{1 - \left(\frac{2}{100}\right)^{(2-D_f)}} \right] \left(\frac{L'}{100} \right)^{(D_f - 2)}. \tag{7}$$

Generating a random number s_3 between $0 \rightarrow 1$ (labeled as *s3* in the *ASSIGN_FRACTURE_LENGTHS* procedure) and then setting this equal to (7), we can solve for the length L' to obtain:

$$L' = 10 \left[s_3 \left(1 - \left(\frac{2}{100}\right)^{(2-D_f)} \right) + \left(\frac{2}{100}\right)^{(2-D_f)} \right]^{\frac{1}{(2-D_f)}}. \tag{8}$$

Next, we generate a second random number s_4 between $0 \rightarrow 1$ and calculate the final fracture length from

$$L = s_4 L'. \tag{9}$$

In this way, the fracture sites are assigned lengths in the horizontal direction. Using the parameters $D_f = 1.5$, $l = 2.7$, and $R_1 = 16$ we obtain 29 fractures with $6 \times 2^3 = 128$ rulers after 3 generations giving the output shown in **Fig.**13.

Referring to **Fig.** 6 we notice that to the left of $L \approx 14$ (*linear regime*) the length assignments may be made using the procedure outlined above. To the right of $L \approx 14$ (*exponential cutoff*) we have a non-linear distribution and so we must use:

$$P_{II}(L) = 129\, e^{-0.047L}. \tag{10}$$

To incorporate the data from region II into our fracture generation program (while avoiding having to solve a non-linear function for L') we are modifying the program by reading in the values from (3) and (10) into an array for each fracture length L' between 2 and 100. Generating a random number between 2 and 100, we can then determine the corresponding fracture length from the array.

At the next iteration, G2 is greater than 1 and the program will step forward by a specified amount in the x-direction ($= (G2 - 1) \times step$) to determine (using the previous length assignments) how many fractures extend past this point. If fractures have ended, new fracture assignments must be made to maintain the distribution in (2). The number of fractures that have not crossed the grid point are counted by the procedure

COUNT_ENDED_FRACTURES and stored in the variable *needed*. If fractures have ended, the procedure *ADD_NEEDED_FRACTURES* is called as shown in **Fig.**12. To guarantee that the new fracture assignments produce a fractal distribution, we must reverse the ruler doubling process and reassign fractures that have crossed the specified grid point to half as many rulers used in the final step of the initial 1-d fracture generation process. The unoccupied fracture sites are then assigned new fractures following the same procedure described for the initial fracture generation.

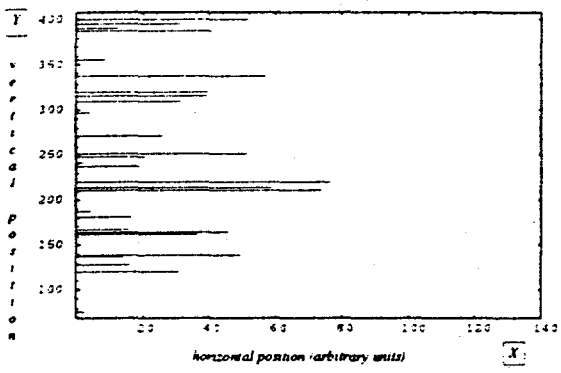

Fig. 13. Lengths are assigned to the initial fracture set.

After new fractures have been added (beginning from $x = (G2 - 1) \times step$) the *ASSIGN NEW FRACTURE LENGTHS* procedure uses (8) and then (9) to determine their length. The x_1 and x_2 coordinates (endpoints) for each of the fractures are stored in the arrays Lfx1[i] and Lfx2[j] and the whole process continues until the distribution is generated for the specified number of horizontal site locations. The endpoints of the fractures along with their vertical position are written to the file FRACT_1.DAT for each value of the gridstep x given above by the *DISPLAY_FRAC_EXTENSIONS* procedure. The output is shown in **Fig.**14 . The parameters used were $D_f = 1.5$, $l = 2.7$, and $R_i = 16$ which were determined from the MWX outcrop using the box counting procedures described in a previous section.

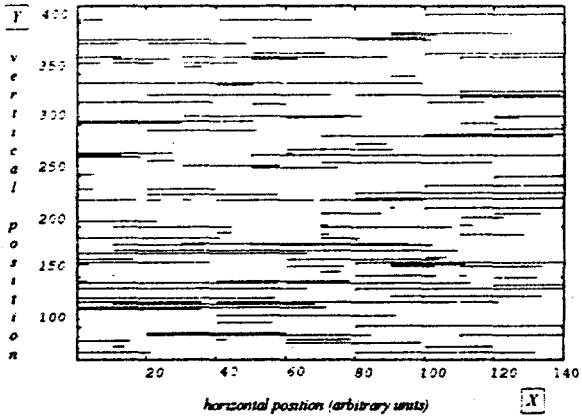

Fig. 14 Horizontal fracture outcrop

To generate the vertical fracture set we first generate a fracture distribution along each of the horizontal fractures by applying our 1-d generation algorithm to each fracture in the datafile FRACT_1.DAT. Since the horizontal fracture positions were previously stored at each value of the gridstep, a fracture crossing *n* gridpoints is stored *n* times by the *DISPLAY_FRAC_EXTENSIONS* procedure. Therefore, before we can assign vertical fractures along each of the horizontal fractures we must first eliminate all duplicate fractures from the data set. This is accomplished by the *ELIMINATE_REDUNDANT_FRACTURES* procedure listed below the dotted line in **Fig.8**. The result of this operation is stored in a new file: FRACT_2.DAT. After obtaining a unique set of horizontal fractures we reinitialize our variables by reading in the FRACT_2.DAT values with the *READ_FRAC_2_DATA* procedure as shown in **Fig.8**.

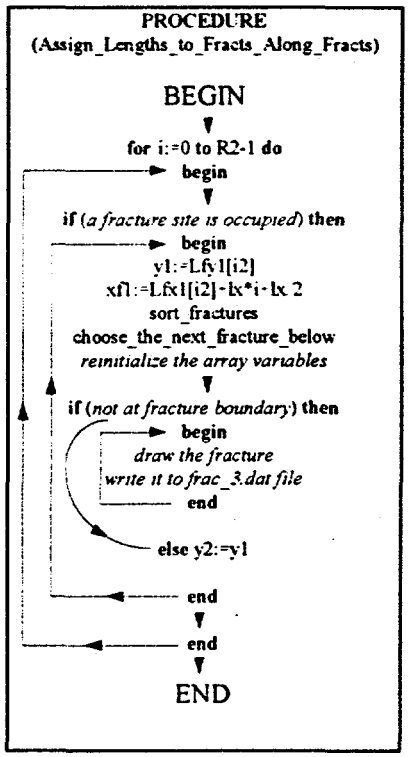

Fig. 15 Procedure flowchart for *ASSIGN_LENGTHS_TO_FRACTURES_ALONG_FRACTURES.*

Starting in the upper left hand corner of **Fig.14** and proceeding downward vertically, the program produces a fractal distribution (using a parameter set determined from the vertical fracture data) along the first fracture in the data set. In our model we assume that vertical fractures can only begin or end along a horizontal fracture. In this case, we need only find the next horizontal fracture below each vertical fracture site to determine the fracture endpoint and therefore its' length. To begin the process the program enters the **for** loop below the *READ_FRAC_2_DATA* procedure in **Fig.8**. If the *i2-th* horizontal fracture has a length greater than a certain number of units, the program executes the *GENERATE_FRACTURES_ALONG_FRACTURES* procedure to generate a fracture set along the *i2-th* fracture. The flowchart for this procedure is completely analogous to the flowchart given in **Fig.9** except that in this case we use a slightly different 1-d fracture

generation procedure (*GENERATE_1D_FRACTURES2*) to incorporate the vertical fracture parameters and new fractal distribution function.

After producing a fracture distribution along the *i2-th* horizontal fracture, the program executes the *ASSIGN_LENGTHS_TO_FRACTURES_ALONG_FRACTURES* procedure whose flowchart is given in **Fig.**15. The outer **for** loop in the procedure scans through all rulers of the fracture distribution just produced by the *GENERATE FRACTURES ALONG FRACTURES* procedure. If a fracture site is occupied then the vertical position of the horizontal fracture is stored in the variable *y1*. The location of the fracture along the *i2-th* horizontal fractures' length is then stored as *xf1*. Now that we have the x and y values of the vertical fractures' starting point - we scan the fracture set (using the SORT FRACTURES and *CHOOSE_THE_NEXT_FRACTURE_BELOW* procedures) to find the vertical position of the next horizontal fracture beneath our given fracture. This position is then stored as *y2*. If the value of *y2* corresponds to a fracture within the boundaries of the network (and not at an adjacent grid site starting at the top of the screen) then the vertical fracture is displayed and its' position stored in the file: *FRAC_3.DAT*. The program terminates when the horizontal fractures have been scanned and vertical fractures are generated along their lengths. Using the identical parameters that were used for **Fig.**14 along with the parameters $D_{f,vertical} = 1.2$, $l_{vertical} = 1$, and $R_{i,vertical} = 4$; we obtain the output shown in **Fig.**16 which may be compared with the MWX data in **Fig.**1.

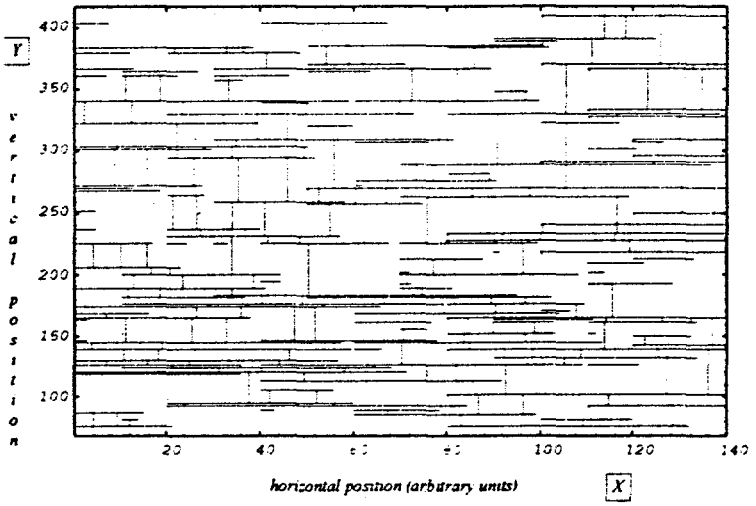

Fig. 16. 2-d Fracture Outcrop Data generated by *2d_frac2.pas*.

Conclusions & Future Work

To model the fracture outcrop networks occurring in naturally fractured tight-gas reservoirs we have taken an approach that incorporates:

A) Fractal Analysis of Available Data:

We characterize the MWX fracture data using four parameters (for the distribution of both horizontal and vertical fractures): i) Lacunarity, ii) Fractal Dimension, iii) Initial Covering Scale, and iv) Cutoff - determined from the distribution of fracture lengths.

B) Fracture Generation:

We generate self-similar fracture networks using data from I.) with an algorithm that incorporates fractal geostatistics.

From our work we have found that there are several advantages in an approach that uses fractal statistics:

i) The networks produced by our model appear to be in agreement with actual fracture networks but do not require extensive apriori knowledge of the network. Using data from isolated borehole sites we can generate entire networks with an algorithm that assumes a self-similar or scale invariant geometry.

ii) We are able to generate horizontal and vertical fractures separately (although not independently) using distinct parameter sets in each case. The fractures can then be analyzed and combined later to produce complete self-consistent networks.

iii) Since the data is generated using a statistical approach, the algorithms require relatively little computer time to produce complete networks

iv) Evidence suggests that real fracture networks obey fractal statistics.

The characterization and analysis of the network data produced by our algorithms is not yet complete. By varying other parameters such as gridsize, fracture length, and the horizontal/vertical orientation of fractures, we believe that it will be possible to generate fracture distribution patterns that are 'optimally similar' in the fractal/statistical sense - to real fracture networks occurring in nature.

Currently we are analyzing the distribution of fractures along horizontal boreholes in the Austin Chalk and fracture lengths from nearby outcrops. The results from this analysis will be used to produce simulated fracture networks.

References

1. Skopec, R. A., JPT. December 1993, 1168, (1993).
2. Davidge, R. W. "Mechanical Behavior of Ceramics." 1979 Cambridge University Press. New York.
3. McKoy, M., private communication. (1994).
4. Xie, H. "Fractals in Rock Mechanics." Geomechanics Research Series. Kwasniewski ed. 1993 A. A. Balkema. Rotterdam.
5. McKoy, M., Development of Stochastic Fracture Porosity Models and Application to the Recovery Efficiency Test (RET #1) Well in Wayne County, West Virginia (1993).
6. Long, J. C. S. and D. M. Billaux, Wat. Resources Res. 23, 1201, (1987).
7. Billaux, D., J. P. Chilès, K. Hestir and J. C. S. Long, Int. J. Rock Mech., Min. Sci. & Geomech. Abstr. 26, 281, (1989).
8. Hewett, T. A. "SPE 15386 Fractal Geostatistics for reservoir hetero's." 1986 Soc. of Pet. Eng. Richardson,TX.
9. Matthews, J. L., A. S. Emanuel and K. A. Edwards, JPT. 1139, (1989).
10. Chilès, J., Math. Geol. 20, 631, (1988).
11. Feder, J. "Fractals." 1988 Plenum Press. New York.
12. Mandelbrot, B. B. "The Fractal Geometry of Nature." 1982 W. H. Freeman Publishers. New York.
13. LaPointe, P. R., Int. J. Rock Mech., Min. Sci. & Geomech. Abstr. . 25, 421, (1988).
14. Heffer, K. J. and T. G. Bevan, fracture length scaling (1990).
15. Roach, D. E., A. D. Fowler and W. K. Fyson, Geology. 21, 759, (1993).
16. Roach, D. E. and A. D. Fowler, Computers & Geosci. 19, 849, (1993).
17. Maloy, K. J., A. Hansen, E. L. Hinrichsen and S. Roux, Phys. Rev. Lett. 213, (1992).

Appendix A: Program Listing: *2d_frac.pas*

(Pascal Source,† 670 lines)

```
PROGRAM td_frac2;   {--- Program declaration ---}

USES crt,graph;   {--- Libraries that will be used ---}

TYPE
 ruler=array[1..2,1..8192] of integer;
 one_d_array=array[1..1600] of real;

VAR          {--- Variables are explained as encountered ---}
 i,
 i1,
 i2,
 c,
 i5,
 j,
 k,
 m,
 s,
 G,
 G2,
 G3,
 e,
 f,
 R,
 Ri,
 R2,
 Ri2,
 count_f;
 c1,
 c2,
 n1,
 needed,
 Magnif,
 step,
```

---

† compiled and developed using *Borland - Turbo Pascal* Version 7.0 under DOS 6.22 on a 486DX2-66 with 16MB of memory. The fracture network figures were produced with *Mathematica* ver. 2.2.1 running under *Windows* 3.11.

```
u,
y_step,
One_d_Generations,
One_d_Generations2,
Two_d_Generations:      integer;
L                       :ruler;
temp,
xf1,
xf2,
cx,
cy,
sx,
sy,
lx,
ly,
Lf,
s3,
s4,
y,
s1,
s2,
y1,
y2,
y3,
Lacunarity,
Lacunarity2,
Fractal_Dim,
Fractal_Dim2,
f_lenth          :real;
Lfx1,
Lfx2,
Lfy1             :one_d_array;
x1,
x2,
Fractures,
Resolution,
Gnumber,
Covered          :string[6];
datafile,
datafile2,
datafile3        :text;
write1                  :string[1];
write            :boolean;
```

```
FUNCTION N(d:integer):integer;  {--- This function gives the distribution of the horizontal fractures---}
 BEGIN
  N:=Round( Lacunarity*Exp( (Fractal_Dim-1)*ln(d) ) );
 END;

FUNCTION N2(d:integer):integer;  {--- This function gives the distribution of the vertical fractures---}
 BEGIN
  N2:=Round( Lacunarity2*Exp( (Fractal_Dim2-1)*ln(d) ) );
 END;

PROCEDURE initialize;  {--- Initialize graphics screen and scaling parameters ---}
 BEGIN
  clrscr;randomize;G:=0;G2:=0;
  Magnif:=3;count_f:=0;
  c1:=detect;c2:=0;initgraph(c1,c2,'c:\tp\BGI');
  setbkcolor(3);
  SetFillStyle(EmptyFill,0);
  n1:=8; write1:='0';write:=false;
  cx:=100;cy:=100*(getmaxy/getmaxx);
  sx:=50;{getmaxx/n1;}sy:=0.8*getmaxy/n1;
  settextjustify(centertext,centertext);
  outtextxy(trunc(getmaxx/2),10,'2-D Fractures');
  delay(1000);
 END;

PROCEDURE display_initial_frac_data;  {--- Display various parameters on-screen ---}
 BEGIN
  k:=0;
  for i:=1 to R do
    begin
     if L[f,i]=1 then k:=k+1;
    end;
  setcolor(4);
  outtextxy(Trunc(0.1*getmaxx),Trunc(0.95*getmaxy),'Rulers =');
  outtextxy(Trunc(0.14*getmaxx+22),Trunc(0.98*getmaxy),'Starting Fractures =');
  outtextxy(Trunc(0.14*getmaxx+13),Trunc(0.92*getmaxy),'Initial Covering =');
  str(R,Resolution);
  str(k,Fractures);
  setcolor(15);
  outtextxy(Trunc(0.175*getmaxx),Trunc(0.95*getmaxy),Resolution);
  outtextxy(Trunc(0.315*getmaxx),Trunc(0.98*getmaxy+0),Fractures);
  str(Ri,Resolution);
```

```
  outtextxy(Trunc(0.295*getmaxx),Trunc(0.92*getmaxy),Resolution);
 END;

PROCEDURE generation_parity;   {-- Determine if G is odd or even --}
 BEGIN
  if (G/2-trunc(g/2)) >0 then
   begin
    e:=1;
    f:=2;
   end;
  if (G/2-trunc(g/2)) =0 then
   begin
    e:=2;
    f:=1;
   end;
 END;

PROCEDURE display_1d_fractures;  {--- Display Fract. along y at a previous Gen. ----}
 BEGIN
  ly:=0.7*getmaxy/R;
  for i:=0 to R-1 do
   begin
     if (i/2-trunc(i/2)) >0 then setcolor(9);
     if (i/2-trunc(i/2)) =0 then setcolor(12);
     line(trunc(sx),trunc(cy+ly*i),
          trunc(sx),trunc(cy+ly*(i+0.95)));
     if (L[e,i+1]=1) then
      begin
       putpixel(trunc(sx),trunc(cy+ly*i+ly/2),15);
      end;
   end;
 END;

PROCEDURE display_1d_fractures2;  {--- Display Fract. along y at the final Gen. ----}
 BEGIN
  ly:=0.7*getmaxy/(2*R);
  for i:=0 to 2*R-1 do
   begin
     if (i/2-trunc(i/2)) >0 then setcolor(9);
     if (i/2-trunc(i/2)) =0 then setcolor(12);
     line(trunc(sx),trunc(cy+ly*i),
          trunc(sx),trunc(cy+ly*(i+0.95)));
     if (L[f,i+1]=1) then
      begin
       putpixel(trunc(sx),trunc(cy+ly*i+ly/2),15);
```

```
      end;
    end;
  END;

PROCEDURE display_1d_fractures3; {-- This backs up to R 2 and shows L[e,i] --}
  BEGIN
   ly:=0.7*getmaxy/(R/2);
   for i:=0 to trunc(R/2)-1 do
    begin
      if (i/2-trunc(i/2)) >0 then setcolor(13);
      if (i/2-trunc(i/2)) =0 then setcolor(4);
      line(trunc(sx-15),trunc(cy+ly*i),
           trunc(sx-15),trunc(cy+ly*(i+0.95)));
      if (L[e,i+1]=1) then
       begin
         putpixel(trunc(sx-15),trunc(cy+ly*i+ly/2),15);
       end;
    end;
  END;

PROCEDURE display_1d_fractures4;  {--- Show fractures by pixel at each Gen. ---}
  BEGIN
   ly:=0.7*getmaxy/(R);
   for j:=0 to R-1 do
    begin
      if (j/2-trunc(j/2)) >0 then setcolor(9);
      if (j/2-trunc(j/2)) =0 then setcolor(12);
      line(trunc(sx+(G2-1)*Step*Magnif),trunc(cy+ly*j),
           trunc(sx+(G2-1)*Step*Magnif),trunc(cy+ly*(j+0.95)));
      if (L[f,j+1]=1) then
       begin
         putpixel(trunc(sx+(G2-1)*Step*Magnif),trunc(cy+ly*j+ly/2),15);
       end;
    end;
  END;
PROCEDURE display_fractures_y1;  {--- Show fractures at a previous stage ---}
  BEGIN
   lx:=Abs(Lfx2[i2]-Lfx1[i2])/(R2);
   for i1:=0 to R2-1 do
    begin
     if (i1/2-trunc(i1/2)) >0 then setcolor(1);
     if (i1/2-trunc(i1/2)) =0 then setcolor(1);
     line(trunc(sx+(Lfx1[i2]+lx*i1)*Magnif+3)      ,trunc(Lfy1[i2]),
          trunc(sx+(Lfx1[i2]+lx*(i1+0.95))*Magnif+3),trunc(Lfy1[i2]));
     if L[e,i1+1]=1 then
```

```
      begin
       putpixel(trunc(sx+(Lfx1[i2]+lx*i1+lx/2)*Magnif+3), trunc(Lfy1[i2]),15);
      end;
    end;
  END;

PROCEDURE display_fractures_y2;
  BEGIN
   lx:=Abs(Lfx2[i2]-Lfx1[i2])/(2*R2);
   for i1:=0 to 2*R2-1 do
    begin
     if (i1/2-trunc(i1/2)) >0 then setcolor(1);
     if (i1/2-trunc(i1/2)) =0 then setcolor(1);
     line(trunc(sx+(Lfx1[i2]+lx*i1)*Magnif+3)      ,trunc(Lfy1[i2]),
          trunc(sx+(Lfx1[i2]+lx*(i1+0.95))*Magnif+3),trunc(Lfy1[i2]));
     if L[f,i1+1]=1 then
      begin
       putpixel(trunc(sx+(Lfx1[i2]+lx*i1+lx/2)*Magnif+3), trunc(Lfy1[i2]),15);
      end;
    end;
  END;

PROCEDURE display_frac_extensions;
  BEGIN
   ly:=0.7*getmaxy/R;              {--This gives the ruler lengths in the y-direc. ----}
   if g2=1 then                 {---Create datafile fract_1.dat ---}
    begin
     assign(datafile,'c:\tp\files\FRACT_1.DAT');
     rewrite(datafile);
    end;
    Bar(trunc(sx+490),trunc(cy), trunc(getmaxx),trunc(cy+340));     {--- Erase old data
from the screen ---}
   for i:=0 to R-1 do
    begin
     if (L[f,i+1]=1) then
      begin
       count_f:=count_f+1;
       str((g2-1),Gnumber);
       Lfy1[i+1]:=cy+ly*i+ly/2;
         writeln(datafile,Lfx1[i+1]:3:2,' ',Lfx2[i+1]:3:2,' ',Lfy1[i+1]:3:2 );     {---write to
fract_1.dat ---}
       setcolor(1);
        line(trunc(sx+Lfx1[i+1]*Magnif+3),trunc(Lfy1[i+1]),    {--- display the fractures
on screen ---}
             trunc(sx+Lfx2[i+1]*Magnif+3),trunc(Lfy1[i+1]));
```

```
        setcolor(8);
        if (G2-1)>0 then              {--- Draw the gridlines on the screen ---}
         line(trunc(sx+(G2-1)*Step*Magnif),trunc(cy),
              trunc(sx+(G2-1)*Step*Magnif),trunc(cy+ly*(R-1)+ly/2));
        if Step>=10 then              {--- Print the grid values on the screen ---}
         begin
          setcolor(8);
          str((g2-1)*step,Gnumber);
          outtextxy(trunc(sx+(G2-1)*Step*Magnif+0),trunc(cy-20),Gnumber);
          str((g2-1),Gnumber);
          outtextxy(trunc(sx+(G2-1)*Step*Magnif+0),trunc(cy+345),Gnumber);
         end;
       end;
     end;
  END;

PROCEDURE generate_1d_fractures;      {--- 1-d Algorithm - Generates Vertical
Slices along x--}
  BEGIN
   for i:=1 to R do   {-- Divide Measuring Scale and Bring down Fractures --}
    begin
     if L[e,i]=1 then  {-- If fractures are present, add fractures below -}
      begin
       for j:=1 to 1 do
        begin
         s:=random(2)+1;
         if s=1 then
          begin
           L[f,2*i-1]:=1;
           L[f,2*i]  :=0;
          end;
         if s=2 then
          begin
           L[f,2*i-1]:=0;
           L[f,2*i]  :=1;
          end;
        end;
      end
     else          {-- If no fractures are present, add spaces ----}
      begin
       L[f,2*i-1]:=0;
       L[f,2*i]  :=0;
      end;
    end;
```

```
  for i:=1 to (N(2*R)-N(R)) do   {------ Add fractures according to distributution ----}
   begin
     repeat
      s:=random(R)+1;
     until (L[e,s]=1) and not ((L[f,2*s-1]=1) and (L[f,2*s]=1));
    if L[f,2*s-1]=1 then L[f,2*s]:=1 else L[f,2*s-1]:=1;
   end;
 END;
PROCEDURE generate_1d_fractures2;
 BEGIN
  for i:=1 to R2 do   {-- Divide Measuring Scale and Bring down Fractures --}
   begin
    if L[e,i]=1 then  {-- If fractures are present, add fractures below -}
     begin
      for j:=1 to 1 do
       begin
        s:=random(2)+1;
        if s=1 then
         begin
          L[f,2*i-1]:=1;
          L[f,2*i]  :=0;
         end;
        if s=2 then
         begin
          L[f,2*i-1]:=0;
          L[f,2*i]  :=1;
         end;
       end;
     end
    else          {-- If no fractures are present, add spaces ----}
     begin
      L[f,2*i-1]:=0;
      L[f,2*i]  :=0;
     end;
   end;

  for i:=1 to (N2(2*R2)-N2(R2)) do   {------ Add fractures ----}
   begin
     repeat
      s:=random(R2)+1;
     until (L[e,s]=1) and not ((L[f,2*s-1]=1) and (L[f,2*s]=1));
    if L[f,2*s-1]=1 then L[f,2*s]:=1 else L[f,2*s-1]:=1;
   end;
 END;
```

```
PROCEDURE Assign_Fracture_Lengths;   {--- Assign lengths to horizontal fractures---}
 BEGIN
  if G2=1 then   {---- Assign initial lengths to fractures ---}
   begin
    for i:=1 to R do
     begin
      if L[f,i]=1 then
       begin
        s3:=(random(100)+1)/100;
        Lf:=Exp((2/3)*Ln( (Exp(1.5*Ln(100))-Exp(1.5*Ln(2)))*s3 ));
        s4:=(random(100)+1)/100;
        Lfx1[i]:=0;
        Lfx2[i]:=s4*Lf;
       end;
     end;
   end;
 END;

PROCEDURE Count_Ended_Fractures;   {--- count horizontal fractures that have
ended---}
 BEGIN
  for i:=1 to R do   {---- If a fracture has ended, count it ---- }
   begin
    if (L[f,i]=1) and (Lfx2[i]<(G2-1)*Step) then
     begin
      needed:=needed+1;
      L[f,i]:=0;
      Lfx1[i]:=(G2-1)*Step;
      Lfx2[i]:=0;
     end;
   end;
 END;

PROCEDURE Display_Grid_info;   {--- display grid values on screen---}
 BEGIN
            str(needed,Gnumber);outtextxy(trunc(sx+(G2-2)*Step*Magnif+7),trunc(cy-
40),Gnumber);
  setcolor(15);outtextxy(trunc(22),trunc(cy-40),'Need:');
  outtextxy(trunc(22),trunc(cy-20),'Grid:');
 END;

PROCEDURE Add_needed_fractures;   {---add horizontal fractures that have ended ---}
 BEGIN
   for i:=1 to needed do
    begin
```

```
      repeat
       Bar(trunc(sx-45),trunc(cy-10),
          trunc(sx-20),trunc(cy+335));
       s:=random(round(R/2))+1;
       str(s,resolution);setcolor(14);
       outtextxy(trunc(sx-30),Trunc(cy-17+ly*2*s),resolution);
      until (L[e,s]=1) and not ((L[f,2*s-1]=1) and (L[f,2*s]=1));
      if L[f,2*s-1]=1 then
        begin
         L[f,2*s]:=1;
         Lfx1[2*s]:=(G2-1)*Step;
        end
      else
       begin
         L[f,2*s-1]:=1;
         Lfx1[2*s-1]:=(G2-1)*Step;
       end;
      display_1d_fractures4;   {--- show the new fracture positions as they are added ---}
     end;
 END;

PROCEDURE Assign_New_Fracture_Lengths; {--- Assign lengths to new fractures ---}
 BEGIN
     for i:=1 to R do
      begin
       if (L[f,i]=1) and (Lfx1[i]=(G2-1)*Step) then
        begin
         s3:=(random(100)+1)/100;
         Lf:=Exp((2/3)*Ln( (Exp(1.5*Ln(100))-Exp(1.5*Ln(2)))*s3 ));
         s4:=(random(100)+1)/100;
         Lfx2[i]:=(G2-1)*Step + s4*Lf;
        end;
      end;
 END;

PROCEDURE generate_horizontal_fractures;    {--- procedure for generating horiz.
fracture set ---}
 BEGIN
  needed:=0; {--- Initialize this variable for the next generation ---}
  if G2=1 then    {---Assign fracture lengths for the initial generation ---}
   Assign_Fracture_Lengths;

  display_frac_extensions;
  display_1d_fractures4;
```

```
  if G2>1 then
   begin
    Count_Ended_Fractures;
    Display_Grid_info;
    display_1d_fractures3;
    display_1d_fractures4;
    if needed>0 then
     begin
      Add_needed_fractures;
      Assign_New_Fracture_Lengths;
     end;
   end;
 END;

PROCEDURE generate_initial_fracture_set;;
 BEGIN
  for i:=1 to Ri do       {--- Set the first Level Fractures --}
   begin
    L[1,i]:=1;
   end;
  G:=0;
  repeat                 {--- start 1-d fracture generation --------}
   G:=G+1;               {----- G Counts the Generations --------}
   generation_parity;    {--- is G odd or even ? ----------------}
   generate_1d_fractures; {---- Algorithm --------------}
   display_1d_fractures;
   display_1d_fractures2;
   R:=2*R;               {-- double the scale resolution --}
  until G=One_d_Generations; {--- end of 1-d loop --}
 END;
PROCEDURE Eliminate_redundant_fractures;
 BEGIN
  reset(datafile);
  m:=0;
  while not Eof(datafile) do  {--- Read in values and count how many from fract_1.dat--
-}
   begin
    m:=m+1;
    readln(datafile,Lfx1[m],Lfx2[m],Lfy1[m]);
   end;
  close(datafile);
  for i:=1 to m do
   begin
    if Lfy1[i]<>(-1) then
     begin
```

```
        for j:=1 to m do
          begin
              if (i<>j) and ((Lfx1[i]=Lfx1[j]) and (Lfx2[i]=Lfx2[j]) and (Lfy1[i]=Lfy1[j]))
then
             Lfy1[j]:=-1;
          end;
        end;
     end;
    Assign(datafile,'c:\tp\files\FRACT_2.DAT');  {---Create file of unique fractures---}
    Rewrite(datafile);
    for i:=1 to m do
     begin
      if Lfy1[i]<>(-1) then
        begin
         writeln(datafile,Lfx1[i]:3:2,' ',Lfy1[i]:3:2,' ',
                    Lfx2[i]:3:2,' ',Lfy1[i]:3:2);
        end;
     end;
    close(datafile);
  END;

PROCEDURE Generate_Fractures_Along_Fractures;  {-- assign fractures along horiz.
fractures--}
  BEGIN
    R2:=Ri2;
    for i:=1 to R do        {--- Set the first Level Fractures --}
      L[1,i]:=1;
    G:=0;
    repeat
     G:=G+1;
     generation_parity;
     generate_1d_fractures2;
     display_fractures_y1;
     display_fractures_y2;
     R2:=2*R2;
    until G=One_d_Generations2;
  END;
```

```
PROCEDURE Switch(Var a,b:Real);  {--- This is used in the sorting procedure ---}
 Var
  c:real;
 BEGIN
  c:=a;
  a:=b;
  b:=c;
 END;

PROCEDURE Sort_fractures;  {--- sort the fractures to assign vertical fractures to next
one below ---}
 Var
  i3,i4:integer;
 BEGIN
  for i3:=2 to u do
   begin
    for i4:=u DownTo i3 do
     begin
      if (Lfy1[i4-1]>Lfy1[i4])  then
       begin
        Switch( Lfy1[i4] , Lfy1[i4-1] );
        Switch( Lfx1[i4] , Lfx1[i4-1] );
        Switch( Lfx2[i4] , Lfx2[i4-1] );
       end;
     end;
   end;
 END;

PROCEDURE Choose_The_Next_Fracture_Below;  {--- Go through sorted list --}
 VAR
  i5:integer;
 BEGIN
  i5:=0;
  repeat
   i5:=i5+1;
  until (Lfy1[i5]>y1) and (xf1>=Lfx1[i5]) and (xf1<=Lfx2[i5]);
  y2:=Lfy1[i5];
 END;

PROCEDURE Read_Fract_2_Data;
 VAR
  i5:integer;
 BEGIN
  i5:=0;
```

```
  reset(datafile);          {---- datafile is Frac_2.dat ----}
  while not Eof(datafile) do
   begin
    i5:=i5+1;
    readln(datafile,Lfx1[i5],Lfy1[i5],Lfx2[i5],Lfy1[i5]);
   end;
  u:=i5;
  close(datafile);
 END;

PROCEDURE Assign_Lengths_to_Fracts_Along_Fracts;    {-- generate and display
vertical fractures --}
 BEGIN
  for i:=0 to R2-1 do    {---- Assign lengths to fractures ---}
   begin
    if L[f,i+1]=1 then
     begin
      y1:=Lfy1[i2];
      lx:=Abs(Lfx2[i2]-Lfx1[i2])/(R2);
      xf1:=Lfx1[i2]+lx*i+lx/2;
      xf2:=Lfx2[i2]+lx*i+lx/2;
      Sort_Fractures;
      Choose_The_Next_Fracture_Below;
      setcolor(1);
      Read_Fract_2_Data;
      if (y2>=0) and (y2<=500) then
       begin
        Line(Trunc(sx+(xf1)*Magnif+3),Trunc(y1),
           Trunc(sx+(xf1)*Magnif+3),Trunc(y2));
        Append(datafile3);
        writeln(datafile3,xf1:3:2,' ',y1:3:2,' ',
                    xf1:3:2,' ',y2:3:2);
        close(datafile3);
       end
       else y2:=y1;
     end;
   end;
 END;

PROCEDURE write_program_parameters_to_file; {---  make  a  datafile  of  the
parameters used ---}
 BEGIN
  Assign(datafile3,'c:\tp\files\fr_text.dat');
  Rewrite(datafile3);
  writeln(datafile3,Lacunarity:3:2,' ',Lacunarity2:3:2,' ',
```

```
                Fractal_Dim:3:2,' ',Fractal_Dim2:3:2,' ',
                Step,' ',One_d_Generations,' ',One_d_Generations2);
 close(datafile3);
END;
```

```
BEGIN  {-------- The Program starts here and executes the procedures----------}
initialize;

 write_program_parameters_to_file;

 generate_initial_fracture_set;

 display_initial_frac_data;     {---- displays initial program info ------}

 G2:=0;                {--- initialize G2 ---}
 REPEAT                  {----- Start 2-d fracture generation ------}
   G2:=G2+1;
   generate_horizontal_fractures;
 UNTIL G2=Two_d_Generations;

 close(datafile); {--- Datafile is Frac_1.dat, containing the endpoints of the horizontal
fractures ---}

 Eliminate_redundant_fractures;  {--Remove duplicate fractures from frac_1.dat and save as frac_2.dat --}

 Read_Fract_2_Data;    {---Reintialize variables with unique fracture values ---}

  Assign(datafile3,'c:\tp\files\fract_3.dat');      {--- Create the datafile: fract_3.dat to store vertical
fractures ---}
 Rewrite(datafile3);
 close(datafile3);

 for i2:=1 to u do   {--- u is the total number of horizontal fractures ---}
  begin
   if Abs(Lfx2[i2]-Lfx1[i2])>=20 then    {- start vertical fractures only along horizon. fractures with length ≥
20 units -}
     begin
      Generate_Fractures_Along_Fractures;
      Assign_Lengths_to_Fracts_Along_Fracts;
     end;
  end;

 setcolor(14);
 outtextxy(trunc(0.8*getmaxx),20,'Done.');     {--- Print a message that the program is
finished ---}
 readln;                {--- wait until a key is pressed ---}
 closegraph;  {--- exit from graphics mode ---}
END.   {-------- Program ends here ----------}
```