STUDY OF ONE- AND TWO-DIMENSIONAL FILTERING AND DECONVOLUTION ALGORITHMS FOR A STREAMING ARRAY COMPUTER

NASA GRANT NO. NSG-1648

FINAL REPORT

Appendix 3

Dr. George E. Ioup, Principal Investigator
Department of Physics
University of New Orleans
New Orleans, LA 70148

A STUDY OF DERIVATIVE FILTERS USING THE DISCRETE FOURIER TRANSFORM

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of
Master of Science

in

The Department of Physics

by
Kathleen Acomb Whitehorn
B. S., University of New Orleans, 1977
May, 1980

ACKNOWLEDGEMENTS

I would like to express sincere appreciation to Dr. George E. Ioup for his guidance and his encouragement, without which this thesis would never have come to fruition. Thanks also go to Dr. Charles E. Head and Dr. Joseph E. Murphy for their helpful suggestions. And last but not least I would like to thank my husband, Mark, for his patient understanding and productive conversations.

Part of the work associated with this thesis was funded by a NASA grant, NSG 1648.

The three-dimensional plots were drawn using the ASPEX plotting package obtained from the Laboratory for Computer Graphics and Spatial Analysis, Harvard University.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Important properties of derivative (difference) filters using the discrete Fourier transform are investigated. The filters are designed using the derivative theorem of Fourier analysis.

Because physical data are generally degraded by noise, the derivative filter is modified to diminish the effects of the noise, especially the noise amplification which normally occurs while differencing. The basis for these modifications is the reduction of those Fourier components for which the noise most dominates the data.

The various filters are tested by applying them to find differences of two-dimensional data to which various amounts of signal dependent noise, as measured by a root mean square value, have been added. The modifications, circular and square ideal low-pass filters and a cut-off pyramid filter, are all found to reduce noise in the derivative without significantly degrading the result. And the last also reduces Gibbs oscillations for those data sets for which these oscillations are present with low-pass filtering.

The FORTRAN programs which perform the filtering (DERIV4.FOR and FILT.FOR) and the program which adds the noise (GNOISE.FOR) are given and discussed.

INTRODUCTION

Filtering is the process of multiplying the Fourier transform of data with some function. The mathematical form of the multiplying function depends on the desired outcome. One type of filter that is used to locate peaks, to determine the position of boundaries and locate edges, (and to determine the derivative) is the derivative filter. The drawback to using this filter is the sensitivity of the derivative operation to noise, especially high frequency noise. This problem can be alleviated by modifying the derivative filter.

In this study a discrete approximation to the derivative of the input data is obtained by operating in the transform domain using the derivative theorem. For data with noise the transform is filtered a second time to reduce the effects of noise. Results of these operations with and without noise are compared in the function domain. Computer programs are used to perform these manipulations of the data.

DERIV4.FOR is the FORTRAN program that outputs the transform of the input data (two-dimensional data), the transform of the x-derivative, the y-derivative, or the second derivative with respect to x and y. DERIV4.FOR uses the one-dimensional FFT program listed in Higgins' article (Higgins,1976). And DERIV4.FOR accepts real (one input file) input data or complex (two input files) input data,

while it outputs complex data. The program that adds the Gaussian noise to the data is GNOISE.FOR, and it accepts only real data. GNOISE.FOR is based on a noise program written by Bill Bivens (Bivens,1976) which has been modified for two-dimensional data. FILT.FOR is the FORTRAN program that performs the filtering, and was written by the author. The three filter choices are a circular filter, a square filter (called a rect filter), and a flat-topped pyramid filter. The filter consists of the multiplication of one of three filter functions after the transform has been multiplied by the derivative filter. The circular and rect filter functions have the value of one inside the boundaries of the respective geometric figures after which the filters are named, and have the value zero outside the boundaries. The flat-topped filter function is one inside the square and slopes linearly (with negative slope) from the edges of the square to the end of the data matrix where the value of the function is zero. FILT.FOR also allows for the size of the filter function (the size of the square or circle) to be varied.

This thesis is separated into three chapters with a summary and an appendix. The three chapters are entitled FOURIER TRANSFORMS, TAKING the DERIVATIVE, and NOISE and FILTERS. The appendix includes the FORTRAN code for the programs that were used.

CHAPTER 1

FOURIER TRANSFORMS

1 1 SPECIAL FUNCTIONS AND THEOREMS

In one dimension the Fourier transform of a continuous function, f(x), is :

$$\int_{-\infty}^{\infty} f(x)\, e^{-2\pi i x s}\, dx$$

This integral is a function of s and will be called F(s). The independent variables in the function and transform domains are x and s, respectively. Here the term frequency will be used to represent the independent variable in the transform domain, regardless of the units of the function domain independent variable. The inverse transform of F(s) is :

$$f(x) = \int_{-\infty}^{\infty} F(s)\, e^{2\pi i x s}\, ds$$

The sign reversal in the exponential is necessary to ensure that two successive transformations result in the original function (Bracewell, 1965).

The Fourier transform of a continuous function in two dimensions is:

$$F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y)\, e^{-2\pi i (ux + vy)}\, dx\, dy$$

where u is the independent variable in the transform domain associated with the independent variable x of the function domain and v of the transform domain is associated with y of the function domain. The inverse transform is:

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u,v)\, e^{2\pi i (ux + vy)}\, du\, dv$$

First I will define some common functions associated with Fourier transforms and give their transforms (Bracewell, 1965).

In one dimension some common functions are

rect(x) = 1 abs(x) < 1/2
= 0 abs(x) > 1/2
= 1/2 abs(x) = 1/2

tri(x) = 1 - abs(x) abs(x) < 1
= 0 abs(x) > 1

the impulse function

$d(x)$: $1 = \int_{-\infty}^{\infty} d(x)dx$; $d(x) = 0,\ x \neq 0$

the sinc function

$\mathrm{sinc}(x) = \sin(\pi x)/(\pi x)$

the replicating or shah function

$III(x) = \sum_{n=-\infty}^{\infty} d(x-n)$

even impulse pair

$II(x) = 1/2d(x+1/2) + 1/2d(x-1/2)$

odd impulse pair

$Ii(x) = 1/2d(x+1/2) - 1/2d(x-1/2)$

Gaussian

$\exp(-\pi x^2)$

COMMON TRANSFORM PAIRS

f(x)	F(s)
rect(x)	sinc(s)
tri(x)	$\mathrm{sinc}^2(s)$
d(x)	1
III(x)	III(s)
II(x)	cos(πs)

Ii(x)	isin(πs)
exp(-πx^2)	exp(-πs^2)

In two dimensions the common functions are

rect(x,y)=	rect(x)rect(y)
sinc(x,y)=	sinc(x)sinc(y)
d(x,y)=	d(x)d(y)
gaussian=	exp(-π(x^2+ y^2))

COMMON TRANSFORM PAIRS

f(x,y)	F(u,v)
rect(x,y)	sinc(u,v)
exp(-π(x^2+ y^2))	exp(-π(u^2+ v^2))
cos(πx)	II(u)d(v)

A few theorems play a basic role when dealing with Fourier transforms. Therefore I will state these theorems here for those unfamiliar with Fourier transforms, but will not give the proofs, which may be found in Bracewell, 1965. A few of the basic theorems:

THE SIMILARITY THEOREM

If f(x) has the Fourier transform F(s) then f(ax) has the Fourier transform $|a|^{-1}$F(s/a). Conceptually this states that broadening a function in the function domain causes

contraction of the transform and growth of its ordinate in the transform domain, and vice versa.

THE ADDITION THEOREM

If f(x) and g(x) have the Fourier transforms F(s) and G(s), respectively, then f(x) + g(x) has the Fourier transform F(s) + G(s).

THE SHIFT THEOREM

If f(x) has the Fourier transform F(s) then f(x-a) has the Fourier transform $\exp(-2\pi ias)F(s)$. This simply states that shifting a function in the function domain is analogous to giving the transform a frequency dependent phase shift.

THE CONVOLUTION THEOREM

If $h(x) = \int_{-\infty}^{\infty} f(u)g(x-u)dx$ ($h=f*g$, f convolved with g) and f(x) has Fourier transform F(s) and g(x) has Fourier transform G(s), then H(s) is the transform of h(x) where $H(s) = F(s)G(s)$. The convolution theorem is used quite often since it is usually easier to perform a multiplication or division than it is to perform a convolution or deconvolution.

THE DERIVATIVE THEOREM

If f(x) has the Fourier transform F(s) then f'(x),the derivative of f(x) has the Fourier transform $2\pi isF(s)$. Here is a method for obtaining the derivative of a function that is not an approximation technique. And by successively applying this theorem higher order derivatives can be obtained.

Another theorem, although not a basic theorem in continuous Fourier transform analysis, is important in going from continuous transforms to discrete transforms. This theorem, known as the sampling theorem, states that a function whose transform is zero for $s > s_c$ (where s_c is the cutoff frequency in the transform domain) is fully specified by samples taken at equal intervals not exceeding $1/(2s_c)$ save for any harmonic terms with zeroes at the sampling points. Thus it is possible to reconstruct a function from its samples if the sampling interval is less than or equal to $1/(2s_c)$. See Fig. 1.1 for graphical detail.

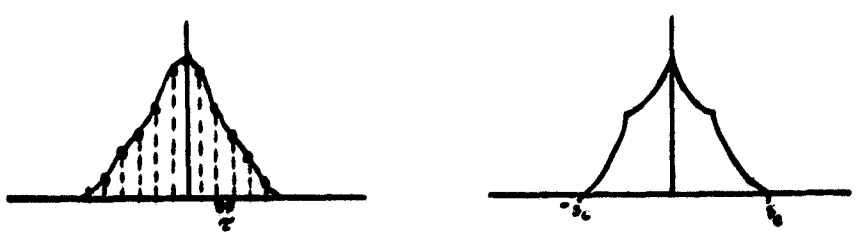

Fig. 1.1 Stating the relationship of t and s_c

The shah function is useful in representing sampled data. For when one multiplies a function f(x), by the shah function, III(x), one is effectively sampling that continuous function at evenly spaced intervals. The values of the function, f(x), at integral values of x are preserved (by the deltas) whereas information of the function between the intervals of the deltas is not kept. Symbolically this sampling property of the shah function is represented as:

$$\text{III}(x)\, f(x) = \sum_{n=-\infty}^{\infty} f(n)\, \delta(x-n)$$

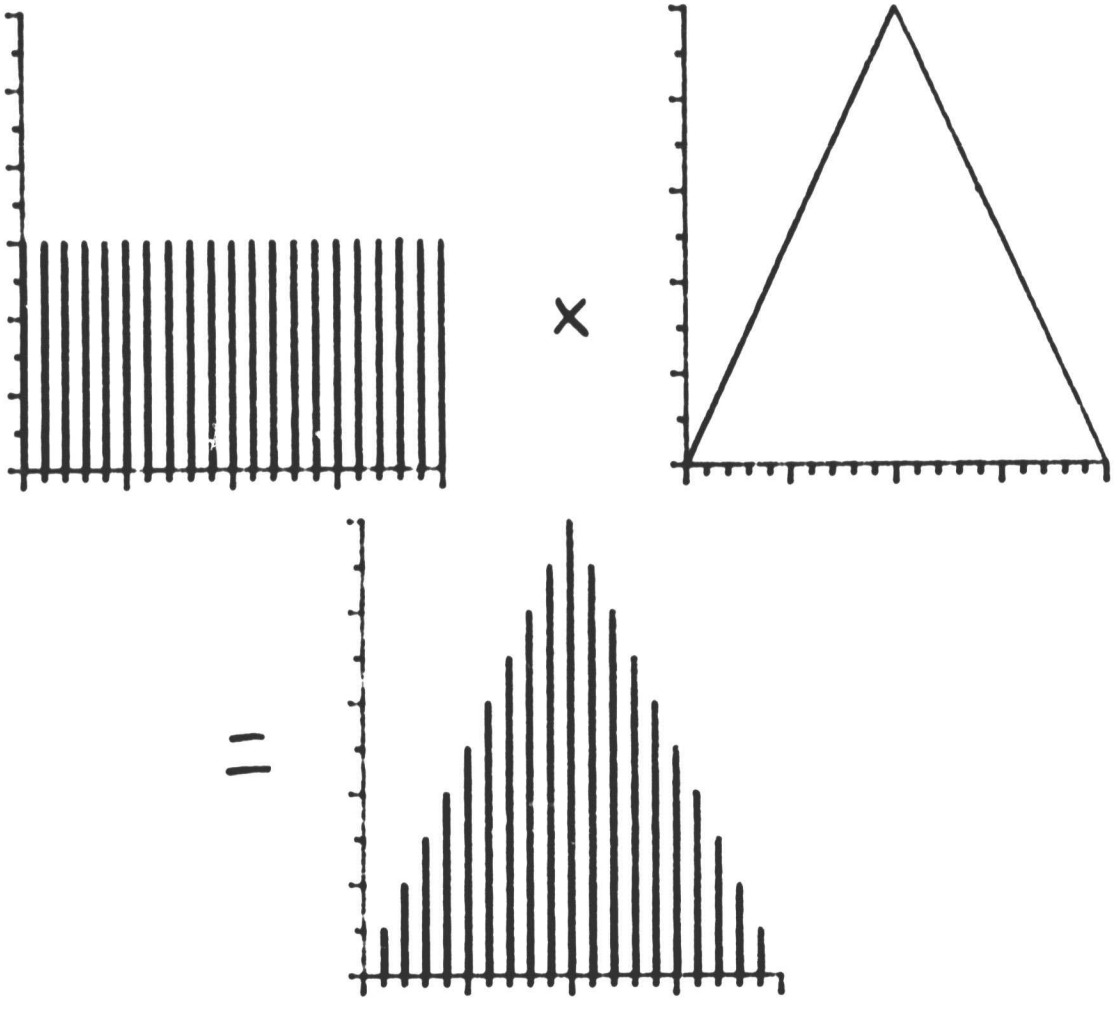

Fig. 1.2 The sampling property of III(x)

And this is graphically stated in Fig. 1.2.

Another important property of the shah function that is closely associated with the sampling property is that of replication. When the shah function is convolved with another function the result is the function being replicated at unit intervals to infinity in both directions. If the function is wider than one unit interval (or wider than the replication interval when this is not unity) then there is overlapping of the replications. When this occurs in the transform domain it is known as aliasing and can be a serious problem. The replicating property symbolically stated is:

$$\text{Ш}(x) * f(x) = \sum_{n=-\infty}^{\infty} f(x-n)$$

The replicating property of the shah is shown graphically in Fig. 1.3.

Discrete functions (data) can be viewed as sampled continuous functions. That is, if the continuous function is represented by f(x) then the discrete (or sampled) version can be represented by III(x/t)f(x), where t is the sampling interval. And as a consequence of sampling, the transform of a discrete function is |t|·III(ts)*F(s), where

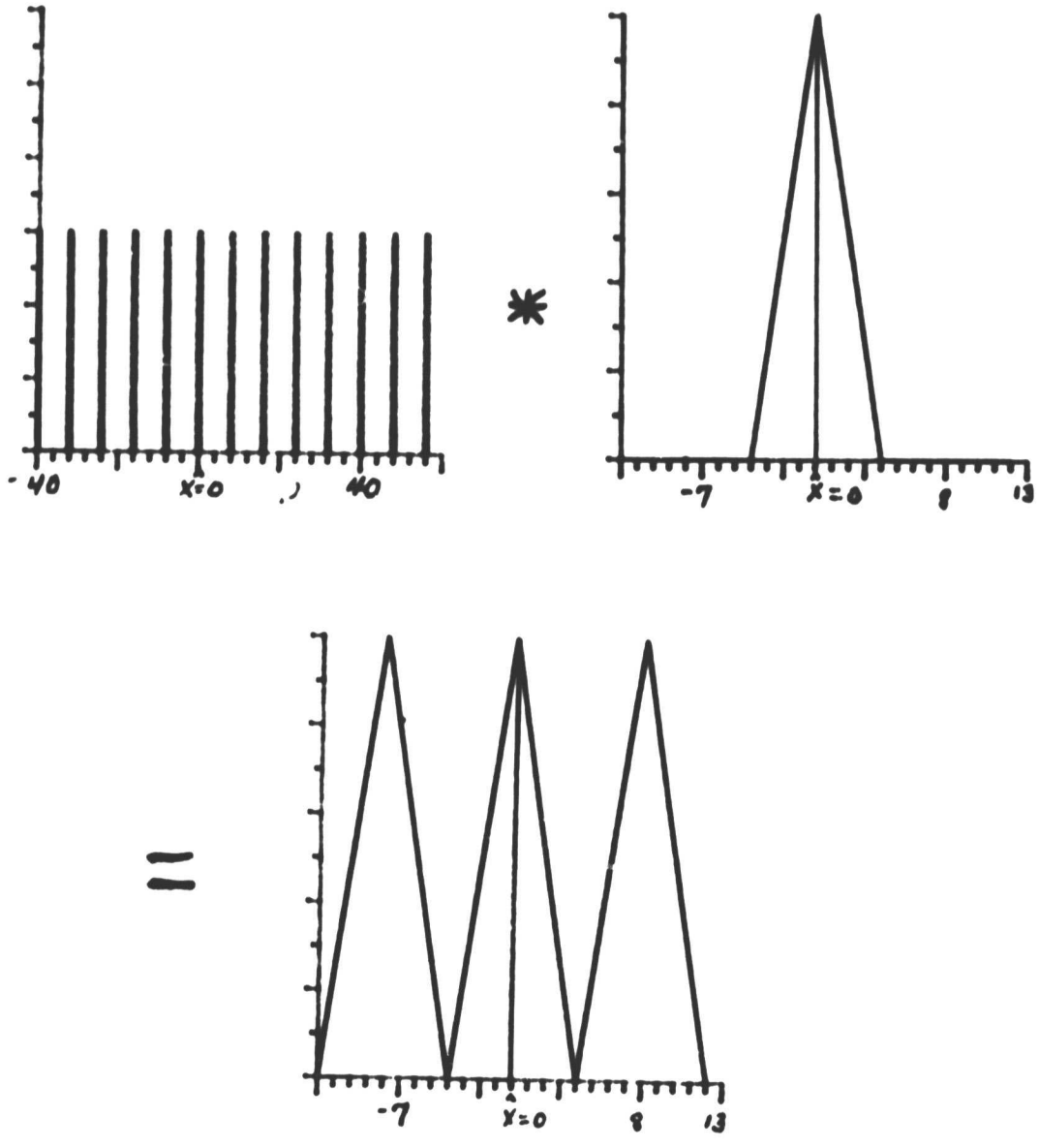

Fig. 1.3 The replicating property of III(x)

F(s) is the transform of the continuous function. Thus we see that sampling in the function domain causes replication in the transform domain. This can also be applied in the reverse direction- sampling in the transform domain causes replication in the function domain.

1.2 THE DISCRETE FOURIER TRANSFORM

The Fourier transform as previously stated operates on continuous functions, whereas physical data are normally discrete. Therefore one must shift gears and begin thinking in terms of discrete functions. One method is to construct discrete functions from continuous functions using the sampling function, and then to use the continuous Fourier transform to obtain a representation for the discrete Fourier transform (known as the DFT).

Thus a discrete function can be represented by III(x/t)f(x), where f(x) is a continuous function that corresponds to the discrete function. Also since the discrete function is taken to have finite extent one can represent it as the continuous function times a rect function. The rect function is often called a window since when it multiplies a continuous function only that portion of the function which falls under the rect is left non-zero. Thus the discrete function can be represented by the product III(x/t)rect((x-a)/c)f(x). The parameter t in the shah

function determines the sampling interval, the parameter c in the rect function determines the width of the window, and a determines the position of the center of the rect. From the convolution theorem the transform of a discrete function can be viewed as $|tc|\cdot\exp(-2\pi ias)\mathrm{III}(ts)*\mathrm{sinc}(cs)*F(s)$. Thus the DFT of a discrete function is not exactly equal to a sampling of the Fourier transform of the analogous continuous function. Due to the processes of sampling and windowing the original function is altered somewhat and thus its transform is affected also (Bracewell,1965).

Convolving the continuous Fourier transform with a sinc broadens it, and the convolution with the shah function causes replication. If any portion of a function that is replicated extends beyond the cutoff frequency then these high frequencies mask as lower frequencies and aliasing occurs. Since convolving with the sinc broadens the transform this contributes further to aliasing. Convolution with the sinc also causes Gibbs oscillations about any rapid change in the transform (Bracewell,1965).

The following is a derivation of the DFT using the representation of discrete functions by continuous functions multiplied with the shah function (Ioup,1978).

discrete function = $f_D(x)$

$$f_D(x) = \sum_{k=0}^{N-1} f(k\Delta x)\, \delta(x - k\Delta x)\, \Delta x$$

$$F_D(s) = \int_{-\infty}^{\infty} f_D(x)\, e^{-2\pi i x s}\, dx$$

$$= \int_{-\infty}^{\infty} \left[\sum f(k\Delta x)\, \delta(x - k\Delta x)\, \Delta x \right] e^{-2\pi i x s}\, dx$$

$$= \sum_{k=0}^{N-1} f(k\Delta x)\, \Delta x \int_{-\infty}^{\infty} \delta(x - k\Delta x)\, e^{-2\pi i x s}\, dx$$

$$= \sum_{k=0}^{N-1} f(k\Delta x)\, \Delta x\, e^{-2\pi i (k\Delta x) s}$$

Sampling $F_D(s)$ at critical sampling interval gives

$$F_D(r\Delta s) = \sum' f(k\Delta x)\, \Delta x\, e^{-2\pi i (k\Delta x)(r\Delta s)}$$

where $\Delta s = \dfrac{1}{N\Delta x}$

The discrete Fourier transform of discrete data is a sampled function. The transform is sampled to allow representation on, and calculation using a digital computer. Thus the data in the function domain are replicated. The fast Fourier transform technique is just a quick method for obtaining the DFT of a function. Therefore the fast Fourier transform (FFT) and the DFT (both are the same transform-one is just a faster approach to calculation for large data sets) view the data from the function domain as one period of the replication with the period of replication equal to the width of the window.

As a result of this if the period of periodic input data is incommensurate with that of the window, then it is better to add zeroes to the end of one period of the function till the periodicity of the function plus the appended zeroes is commensurate with the window. Function domain replication, implicit in the use of a sampled transform, joins incomplete periods of the function together causing discontinuities to be generated when the data is incommensurate with the window. See Fig. 1.4.

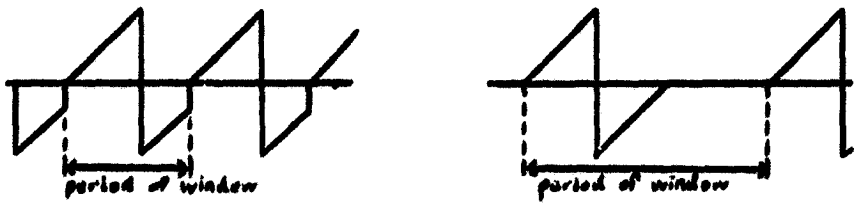

Fig. 1.4 Incommensurate function without and with zeroes

The discontinuities could introduce Gibbs oscillations, in a DFT representation. These will be discussed in more detail in the chapter on noise and filters.

The cutoff frequency in the transform domain is equal to one-half the inverse of the sampling interval in the function domain (i.e., $=1/(2t)$). Thus to include higher frequencies and reduce aliasing the sampling interval is made as small as possible.

1.3 THE FAST FOURIER TRANSFORM

The FFT algorithm reduces the number of operations performed in the calculation of the DFT of a sequence. The algorithm was rediscovered by Cooley and Tukey in 1964 (Cochran et al). The significance of this algorithm is that it reduces the time required to calculate the DFT of a sequence. It takes N multiplications to compute the DFT in the straightforward method, whereas the number of multiplications performed using the FFT algorithm is approximately $2N\log_2 N$. As N gets large the savings in computation time becomes great. For one-dimensional data

N=1024 is common; for this example the savings amount to a factor of one hundred reduction in the number of operations required (Higgins,1976; Cochran et al,1967).

Basically the FFT algorithm can be understood by taking an N point transform and splitting it into two N/2 point transforms. Then these two transforms are each split in half, and this process repeats itself until there are N one point transforms. There are other FFT algoritms that work on sequences of N points (N not prime) for N not two raised to an integral power, but because the reduction to N one point transforms can not be completed they are not as efficient. The FFT algorithm also uses the periodicity of the exponential function to eliminate redundant operations.

To understand this process let A(r) be the value of the transform of X(k) at the frequency r=rΔs, where r =0,1,2,...N-1. N is the number of points in the sequence X(k). Then from the DFT

$$A(r) = \sum_{k=0}^{N-1} X(k)\exp(-2\pi irk/N)$$

Then the data set is split into even and odd sequences, Y(k) and Z(k).

$Y(k) = X(2k)$ $\qquad k=0,1,2,\ldots,(N/2)-1.$

$$Z(k)=X(2k+1) \qquad k=0,1,2,\ldots,(N/2)-1.$$

And let

$$A(r)=\sum_{k=0}^{N/2-1}[Y(k)\exp(-4\pi irk/N) + Z(k)\exp(-2\pi i(2k+1)r/N)]$$

$$= \sum_{k=0}^{N/2-1} Y(k)\exp(-4\pi irk/N) + \exp(-2\pi ir/N) \sum_{k=0}^{N/2-1} Z(k)\exp(-4\pi irk/N)$$

where r= 0,1,2,...,N-1

if $K(r)=\sum_{k=0}^{N/2-1} Y(k)\exp(-4\pi irk/N)$

$$L(r)=\sum_{k=0}^{N/2-1} Z(k)\exp(-4\pi irk/N)$$

$$M(r)= \exp(-2\pi ir/N)$$

then

$$A(r)= K(r) + M(r)L(r) \qquad r=0,1,2,\ldots,(N/2)-1.$$

Since K(r) and L(r) are periodic in the half interval ($0 \le r < N/2$), A(r) can be generated for the second half using the values of K(r) and L(r) for the first half.

$$A(r+(N/2))= K(r) - M(r)L(r)$$

The minus sign comes from $\exp(-2\pi ir/N)$ as $0 \le r < N/2$ being

opposite in sign to $\exp(-2\pi ir/N)$ as $N/2<=r<N$, (i.e., $\exp(-2\pi ir/N) = -\exp(-2\pi i(r+(N/2))/N)$). Thus the N point transform has gone to two N/2 point transforms.

The FFT algorithm used (Higgins,1976) assumes that the first data point is the value of the function at the origin, and that any values associated with negative x (abscissa) are placed beyond the function value of the last positive x. See Fig. 1.5. The data must either input to the FFT program in this form or a portion of the program must be devoted to rearranging the data into the format required by the FFT subroutine. The program DERIV.FOR which performs a one-dimensional FFT of a sequence does the latter. The output of DERIV.FOR is in the same format as the input. Though instead of rearranging the transform the input data is multiplied by a phase factor (the input data can be viewed as having been shifted to the center of the matrix). The result is the same as if the rearrangement had been performed (Andrews,1970).

When the number of data points is a power of two one runs into difficulty in representing even functions that have their origin sampled. Since the function is represented by an even number of points the window can not be symmetric about the origin. Thus it is best to have the function go to zero at both ends within the width of the window. If this is not possible then the asymmetry and the replication in the function domain which results from having

Fig. 1.6 Rearrangement of data

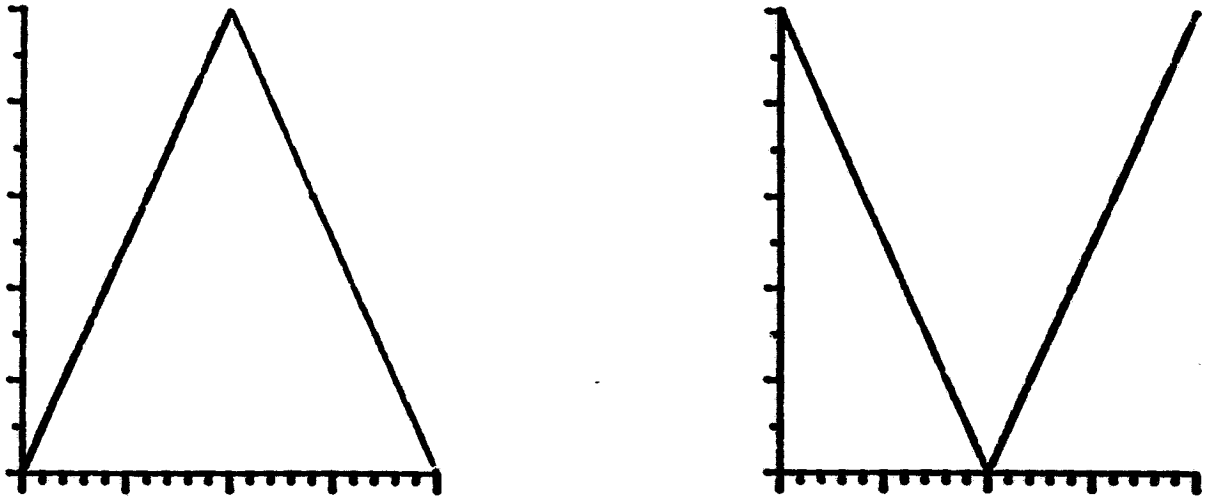

Fig. 1.5 Rearrangement of data

a sampled transform must be carefully considered.

In two dimensions the DFT is defined as:

$$F(m,n)=\sum_{k=0}^{N-1}\sum_{j=0}^{M-1}X(j,k)\exp(-2\pi i((mj/M)+(nk/N)))$$

which can be rewritten as:

$$F(m,n)=\sum_{k=0}^{N-1}\left[\sum_{j=0}^{M-1}X(j,k)\exp(-2\pi imj/M)\right]\exp(-2\pi ink/N)$$

The term in the brackets is the transform of row (or column) k. And the outer sum transforms the columns of the above result. Thus a two-dimensional DFT can be obtained by first performing a one-dimensional DFT on each row (this amounts to M one-dimensional DFT operations) and then executing a one dimensional DFT on each column of the matrix of transformed rows (this amounts to N one-dimensional DFT operations). This is equivalent to the result obtained when columns and rows are interchanged in the above procedure.

The two-dimensional discrete Fourier transform is also sampled. The input data are now a matrix of values, where the first index corresponds to the y coordinate values and the second to the x coordinate values.

Since the one-dimensional FFT algorithm used in the two-dimensional FFT is of the type discussed previously, the two-dimensional FFT routine also expects data in a different format than what might be expected. In addition, the

transform is not arranged as expected. Instead of the origin being located at or near the center of the transform matrix (at point N/2 + 1, N/2 + 1) for an even matrix (where N is the number of rows or columns in the matrix) the FFT results in the origin being located at the top left corner of the matrix with all the low frequencies in the corners and the high frequencies in the center. The same idea applies to data in the function domain. See Fig. 1.6.

The two- dimensional FFT program used, DERIV4.FOR, expects data with the origin located at the center and rearranges it into the format expected by the two-dimensional FFT subroutine. The two-dimensional transform is also rearranged such that the origin is at the center by the phase multiplication method mentioned previously in the section on the one-dimensional FFT.

Since the number of points is normally even, and in our case N where N is a power of two, there is a missing bottom row and a rightmost column if data with even symmetry and with the origin sampled are used. Since the transform is sampled the input data are viewed as one period of the replication. The one-dimensional discussion generalizes to two-dimensions.

As in the one-dimensional case sampling in the function domain causes replication in the transform domain. Though now the replication is in two dimensions with the top row of one period adjacent to the bottom row of another period and

the same for the left and right most columns.

CHAPTER 2

TAKING THE DERIVATIVE

From the derivative theorem the transform of the derivative of a function is just $2\pi is$ times the transform of the function to be differentiated. For higher order derivatives the relationship is: $\mathcal{F}(f^{(n)}(x)) = (2\pi is)^n F(s)$, where $\mathcal{F}(f(x)) = F(s)$.

Thus the derivative theorem provides a method for obtaining the derivative of continuous and discrete functions without approximation other than any approximation already made in treating a continuous function discretely. This method, like any derivative technique, is sensitive to noise, especially since there is no smoothing due to the approximations of normally used numerical techniques. But if one is already using the FFT and the data are relatively noise-free then this method is definitely a viable alternative to derivative approximation methods. Also, if the data are noisy very effective filters may be used as part of the derivative process.

Since I have concentrated this investigation on two-dimensional functions most of the programs operate on two-dimensional data with most of the results for two dimensional data. When it comes to explaining the theory I will use one-dimensional functions for simplicity, reverting to two-dimensional functions only to illustrate an important point or a veiled implication.

The FORTRAN program DERIV4.FOR takes as input two-dimensional data in the form of a two-dimensional square matrix, with the maximum size of the matrix being 64 X 64. Most of the time though, the size of the data matrix was 32 X 32, a compromise between the number of points desired and the length of time required to run the program. Using 32 X 32 matrices DERIV4.FOR ran in approximately one third of the time compared to when 64 X 64 matrices were used.

DERIV4.FOR gives the user four choices of how the transform will be manipulated. Once the transform is obtained it is multiplied by $2\pi iu$, or by $2\pi iv$, or by $-4\pi^2 uv$, or it is untouched. Thus DERIV4.FOR can give the transform of the x-derivative of the input data, the transform of the y-derivative of the input data, the transform of the second derivative with respect to x and y of the input data, or the transform of the data. To obtain the transform of the second derivative with respect to x or y DERIV4.FOR is just run twice, with the output from the first run being the input for the second run.

Also DERIV4.FOR performs either the minus-i or the plus-i transform. The minus-i transform has a negative i in the argument of the exponential whereas the plus-i transform has a positive i in the argument of the exponential.

In multiplying by $2\pi iu$ or $2\pi iv$, etc., the transform is "centered" (as centered as can be using an even number of rows and columns) about the origin. Thus in some cases the

sign of the u or v is negative. That is, although any replication could be used, we use the one centered about the origin.

The sampled functions that were used to check DERIV4.FOR were a two-dimensional gaussian and a cosine wave. The gaussian used was $\exp[((17-I)^2 + (J-17)^2)/(4.0)]$. Figure 1.7 is the gaussian and Fig. 1.8 is the x-derivative of the gaussian. First DERIV4.FOR was run to obtain the minus-i transform of the x-derivative of the gaussian. This was then plus-i transformed to obtain the x-derivative of the gaussian. The same sequence of events were followed to obtain the x-derivative of the cosine wave. The relation used for the cosine wave was $1 + \cos(2\pi(J-17)/16)$ where the addition of one was to produce non-negative data. Figures 1.9 and 1.10 are respectively plots of the cosine wave and its derivative.

CHAPTER 3

NOISE AND FILTERS

3.1 NOISE

Since noise in signals is very common, whether it be background noise, instrument noise, or another unwanted signal, developing the method of taking the derivative using the derivative theorem of Fourier transform analysis would not be complete without including noisy data. The type of noise chosen was ordinant dependent Gaussian additive noise, i.e., noise with a Gaussian probability density function. This choice was made since the noise associated with most imaging sensors can be modeled as a Gaussian distributed random process.

GNOISE.FOR is the FORTRAN program that adds noise to the input data. The output is noise added onto the data. If f(y) is the probability density function then $f(y) = e^{-y^2/\alpha^2 2}$. Now f(y) is the probability density function of the noise. To determine the amplitude of the noise at a particular point the relation between the amplitude and probability density function must be determined.

If $f(y) = e^{-y^2/2\alpha^2} \Rightarrow$

$$\ln[f(y)] = \frac{-y^2}{2\alpha^2}$$

$$2\alpha^2 \ln[f(y)] = -y^2$$

$$y = \sqrt{2\alpha^2 \ln\left[\frac{1}{f(y)}\right]}$$

Since f(y) is evenly distributed, f(y) can be represented by a uniformly distributed random number. Thus, if P is a random number between zero and one, $y = \sqrt{2\alpha^2 \ln\left[\frac{1}{P}\right]}$.

To describe ordinant dependent noise α is not a constant but is equal to a scale factor times the ordinant of the data point in question (α =SF·A(I,J), where A(I,J) is the value of the function associated with the point ((J-17),(17-I))). The scale factor allows the root mean square value (RMS) of the noise to be varied.

The amplitude of the noise is y. This is added to the ordinant (or the value of the function associated with the point) by GNOISE.FOR. Additive noise was used since it is common and the simplest to deal with mathematically.

The synthetic data sequences to which noise was added were the gaussian and cosine wave used before. The scale factors used were 0.00001 and 0.0001. The following plots show the data with noise and then its derivative. For the

Gaussian the derivative is real, thus the imaginary part gives an idea of the round-off error. For example, the magnitude of the maximum and minimum values of the imaginary part of the x-derivative of the Gaussian data are 42.349 and -42.349 respectively for the scale factor equal to 0.00001. For the maximum and minimum values associated with the other x-derivatives of the Gaussian data and the cosine wave data with the associated scale factors of the noise; see Table I.

TABLE I

d/dx of	maximum	minimum	scale factor	Re/Im
Gauss.	42.3	- 42.3	.00001	Im
Gauss.	12492.0	-12655.5	.00001	Re
Gauss.	124.5	- 124.5	.0001	Im
Gauss.	12018.8	-13681.5	.0001	Re
cosine	274.2	- 274.2	.00001	Im
cosine	7589.0	- 7439.5	.00001	Re
cosine	772.4	- 772.4	.0001	Im
cosine	11306.4	- 9536.4	.0001	Re

The RMS is the root mean square of the noise amplitude. That is

$$\text{RMS} = \sqrt{\frac{\sum_{i,j} (n_{i,j})^2}{N^2}}$$

; $n_{i,j}$ = noise amplitude at point i,j

; N^2 = number of points

The SNR is the signal to noise ratio, which is

$$SNR = \frac{\text{PEAK SIGNAL VALUE}}{\text{RMS of NOISE}}$$

Each filter operated on data with the noise scale factor equal to 0.0001 and 0.00001. Also the SNR and RMS for the Gaussian and cosine waves with noise scale factors of 0.0001 and 0.00001 are listed in Table II.

TABLE II

function	scale factor	SNR	RMS
Gauss.	.00001	814.3	.123 E-2
Gauss.	.0001	285.6	.350 E-2
cosine	.00001	188.1	.106 E-1
cosine	.0001	66.7	.300 E-1

3.2 FILTERING

The aim of any filtering is to reduce the unwanted effects in data and enhance the wanted ones. We wish to minimize the effects of the noise on the derivative of the data. We accomplish this by filtering in the transform domain, because if any of the characteristics of the noise in the transform domain are known then it is simpler to design the filter in the transform domain. Most often the only characteristics of the noise that are known are those which are simply given in the transform (frequency) domain.

In the transform domain the transform of the derivative of the function is $2\pi i s F(s)$, where $F(s)$ is the transform of the function itself. The multiplication by s amplifies high frequency noise, therefore any filter should decrease this effect. One method would be to cut off any frequencies above a certain value. This is analogous to multiplying the transform of the derivative with a two-dimensional rect function, a two-dimensional circular function, or some other geometric shaped plateau function. The drawback with this type of filtering is the introduction of Gibbs oscillations due to the abrupt windowing in the frequency domain by the filter. To reduce any Gibbs oscillations a tail can be added to the plateau filter. though this also increases the effect of high frequency noise on the transform.

Gibbs oscillations are the oscillations that result around rapid changes in the function domain when the function is represented by a transform that has been truncated (multiplied by a rect function in the simplest case) (Bracewell,1965). A truncated transform translates in the function domain to convolving the function with a sinc function, if the region of a discontinuity in the time domain is to be examined. The discontinuity can be approximated with the sgn function($\mathrm{sgn}(x)= 1,\ x>0;\ = -1,\ x<0$). Now $\mathrm{sgn}(x)*\mathrm{sinc}(x) = 2\int_0^x \mathrm{sinc}(t)dt$, where $2\int_0^x \mathrm{sinc}(t)dt = (2/\pi)\mathrm{Si}(\pi x)$ (Si is the sine integral). This function oscillates about -1 for large negative x values. As the origin is approached the amplitude of the

oscillations increases, passes though zero at x=0, shoots up to a maximum of 1.18 and then oscillates about 1 as x increases, with the oscillations dying out as x increases. The amplitude of the oscillations about -1 and 1 remains the same if the sinc function is compressed by a factor of N and strengthened by a factor of N (to preserve unit area) and only the frequency of the oscillations is altered. It is increased. Thus changing N does not change the amount of overshoot which is approximately nine per cent of the amount of the discontinuity.

Thus to reduce Gibbs oscillations a linear tail was added to the rect function to form a flat-topped pyramid function. This flat-topped pyramid filter function, the circular filter function and the rect filter function were used on noisy gaussian and cosine wave functions.

Dealing first with the x derivative of the Gaussian data, the worst filters (worst in terms of affecting the presence of the noise) were the circular filter with the radius, R, equal to four and the rect filter with the length of a side equal to nine (=2M+1, where M=4), with the circular filter being worse. (Because the derivative of the Gaussian should be real, even with noise, the non-zero imaginary parts reflect round-off error in the calculations).

For the circular filter with R=4, the oscillations in the derivative were quite large and were not dying down within the period. While the oscillations in the derivative from the rect filter with M=4 were still rather large, they were dying off some as x and y varied from zero.

With R=8 the oscillations in the result using the circular filter are less and show some circular symmetry. The result with the rect filter with M=8 has oscillations which are less than for M=4, and die off more rapidly for values of x and y off axis. Also the oscillations are greatest in the x direction. This is the direction in which the derivative is taken, and the derivative process tends to amplify the effects of any type of noise.

The pyramid filter adds a linear ramp to the rect function that extends from the edge of replication (or edge of the square containing all the data- not including the extra row or column) to the edge of the rect. The derivatives using the pyramid filter show no perceptable oscillations.

For the cosine wave data the rect and circular filters will work the best since the transform of a cosine wave is the even pair situated on the u axis with their separation determined by the period of the cosine. The transform of the cosine wave chosen for this work is non-zero only close to the origin. Therefore the rect and circular filters can cutoff much of the transform (and thus much of the noise)

but still have only a small effect on the even pair (actually the even pair convolved with a sinc because of the windowing in the function domain).

This was seen in the results of the circular, rect, and pyramid filters. The pyramid filter was the worst since it let in more noise. And as expected as R or M decreased in the circular, rect or pyramid filters the results showed less effects of noise.

The cosine wave is representative of functions whose spectrum is centered closely about the origin and such drastic measures (M=4,R=4) would eliminate important information in the transform domain for functions which are not so concentrated. Thus, since the Gaussian had a spectrum which spread out over the entire period (two-dimensional) in the transform domain, the pyramid filter was better as it allowed more information of the transform through.

SUMMARY

A study of derivative filters using the discrete Fourier transform has been performed. As has been discussed a filter multiplies the transform of the data to be filtered with some function. Thus the derivative filter multiplies the transform of the data with $2\pi is$, where s is the independent variable in the transform domain. This result is the derivative theorem of Fourier transform analysis.

But because the derivative process is sensitive to noise, the filter must be modified to reduce the noise effects. This noise filtering can be done before, after, or combined with the derivative filtering since the multiplication is commutative. In this case the noise filtering was done after the derivative filters.

The input data were a two-dimensional cosine wave and a two-dimensional Gaussian wave. The input data were entirely two- dimensional, though in the theoretical portions of this thesis reference was made to one-dimensional functions for simplicity. And since the data were two-dimensional all the computer programs were written to operate on two-dimensional data.

The Gaussian ordinant dependent noise was added to the input data by the program GNOISE.FOR. This program uses the random number generator in FORTRAN to determine the size of the noise.

DERIV4.FOR was used to take the FFT of the input. The filter program used was FILT.FOR.

After a set of noisy data was filtered by DERIV4.FOR and FILT.FOR the resultant output was transformed back to the function domain in order to examine the derivative of the noisy data after filtering.

Of the three filters, the one employing a tail (the function sloped down to zero rather than abruptly cutting off) worked the best on data having a transform not concentrated about the origin. Gibbs oscillations in the function domain are reduced by such filters and the trade-off in letting more noise through to allow the transform to go gradually to zero is definitely beneficial.

For data with transform information centered about the origin, the filters that cut off abruptly worked better than the filter that employed a tail. This is due to being able to get rid of the high frequency noise without destroying any important transform information. Thus the appropriate filter depends on the type of data. Since the information in the transform domain is generally not concentrated about the origin, filter functions that are to be used for common data need tails.

Future work in this area might be to use different tails such as a Gaussian tail for the circular filter or a cosine wave tail for the rect filter. Also the filter could

be tested using another type of noise, rather than additive Gaussian ordinant dependent. Along these lines of thought the noise could be ordinant dependent for ordinants larger than a certain value, while for ordinants less than a certain value the noise could be independent of the ordinant or depend on some other parameter.

Perspective Plots

The following are plots of the final results. SF is the scale factor used to determine the amplitude of the noise. Cir, Pyr, and Rect are abbreviations for the circular, flat-topped, and square filter functions, respectively.

1.7 Gaussian input data

1.8 x-derivative of Gaussian data

1.9 Cosine wave input data

1.10 x-derivative of Cosine data

1.11 x-derivative of Gauss. SF=1.OE-4, no filter

1.12 x-derivative of Gauss. SF=1.OE-3, no filter

1.13 x-derivative of Gauss. SF=1.OE-4, Pyr, M=4

1.14 x-derivative of Gauss. SF=1.OE-4, Pyr, M=4 imag

1.15 x-derivative of Gauss. SF=1.OE-3, Pyr, M=4

1.16 x-derivative of Gauss. SF=1.OE-4, Pyr, M=8

1.17 x-derivative of Gauss. SF=1.OE-3, Pyr, M=8

1.18 x-derivative of Gauss. SF=1.OE-4, Cir, R=4

1.19 x-derivative of Gauss. SF=1.OE-3, Cir, R=4

1.20 x-derivative of Gauss. SF=1.OE-4, Cir, R=8

1.21 x-derivative of Gauss. SF=1.OE-3, Cir, R=8

1.22 x-derivative of Gauss. SF=1.OE-4, Rect M=4

1.23 x-derivative of Gauss. SF=1.OE-3, Rect M=4

1.24 x-derivative of Gauss. SF=1.OE-4, Rect M=8

1.25 x-derivative of Gauss. SF=1.OE-3, Rect M=8

1.26	x-derivative of cosine	SF=1.0E-4, Rect M=8
1.27	x-derivative of cosine	SF=1.0E-3, Rect M=8
1.28	x-derivative of cosine	SF=1.0E-4, Cir, R=8
1.29	x-derivative of cosine	SF=1.0E-3, Cir, R=8
1.30	x-derivative of cosine	SF=1.0E-4, Cir, R=4
1.31	x-derivative of cosine	SF=1.0E-3, Cir, R=4
1.32	x-derivative of cosine	SF=1.0E-4, Pyr, M=8
1.33	x-derivative of cosine	SF=1.0E-3, Pyr, M=8
1.34	x-derivative of cosine	SF=1.0E-4, Pyr, M=4
1.35	x-derivative of cosine	SF=1.0E-3, Pyr, M=4
1.36	x-derivative of cosine	SF=1.0E-4, no filter
1.37	x-derivative of cosine	SF=1.0E-3, no filter

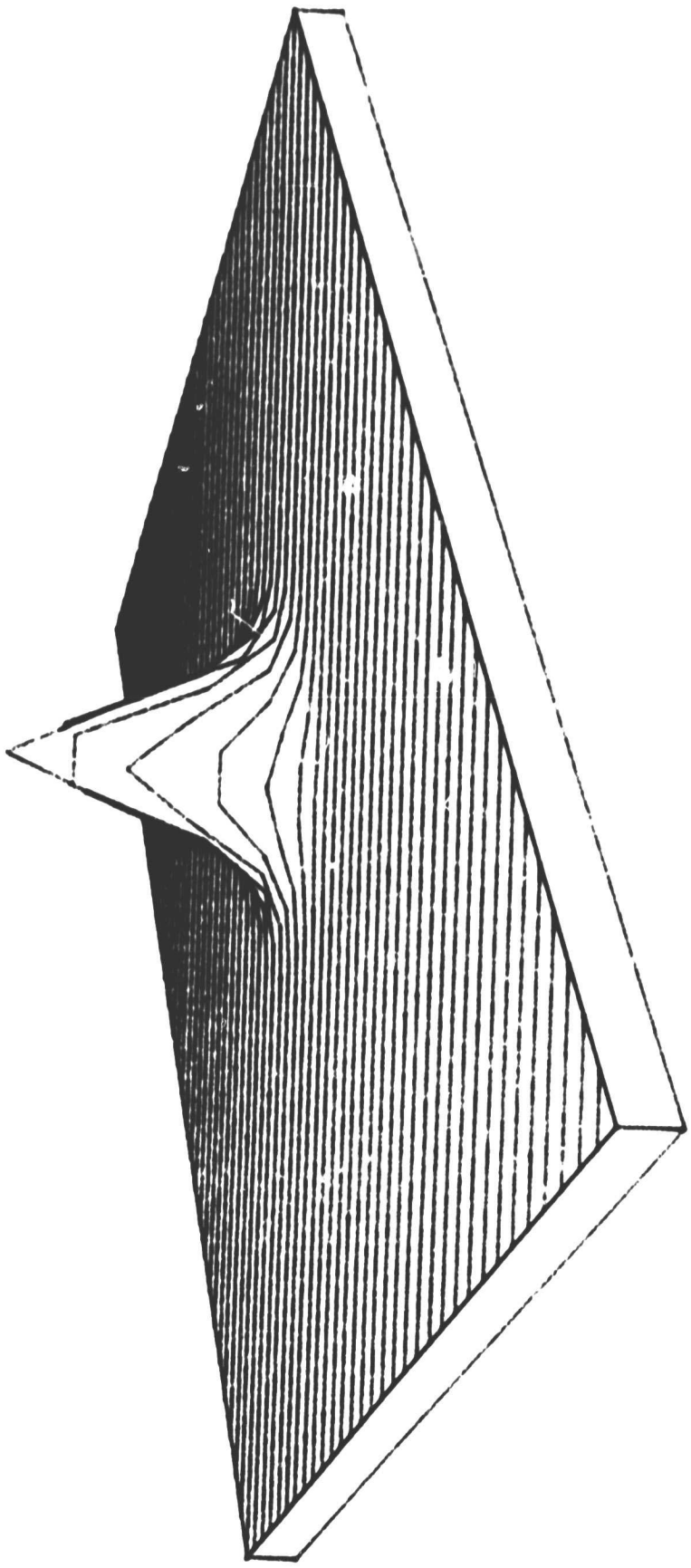

Fig. 1.7

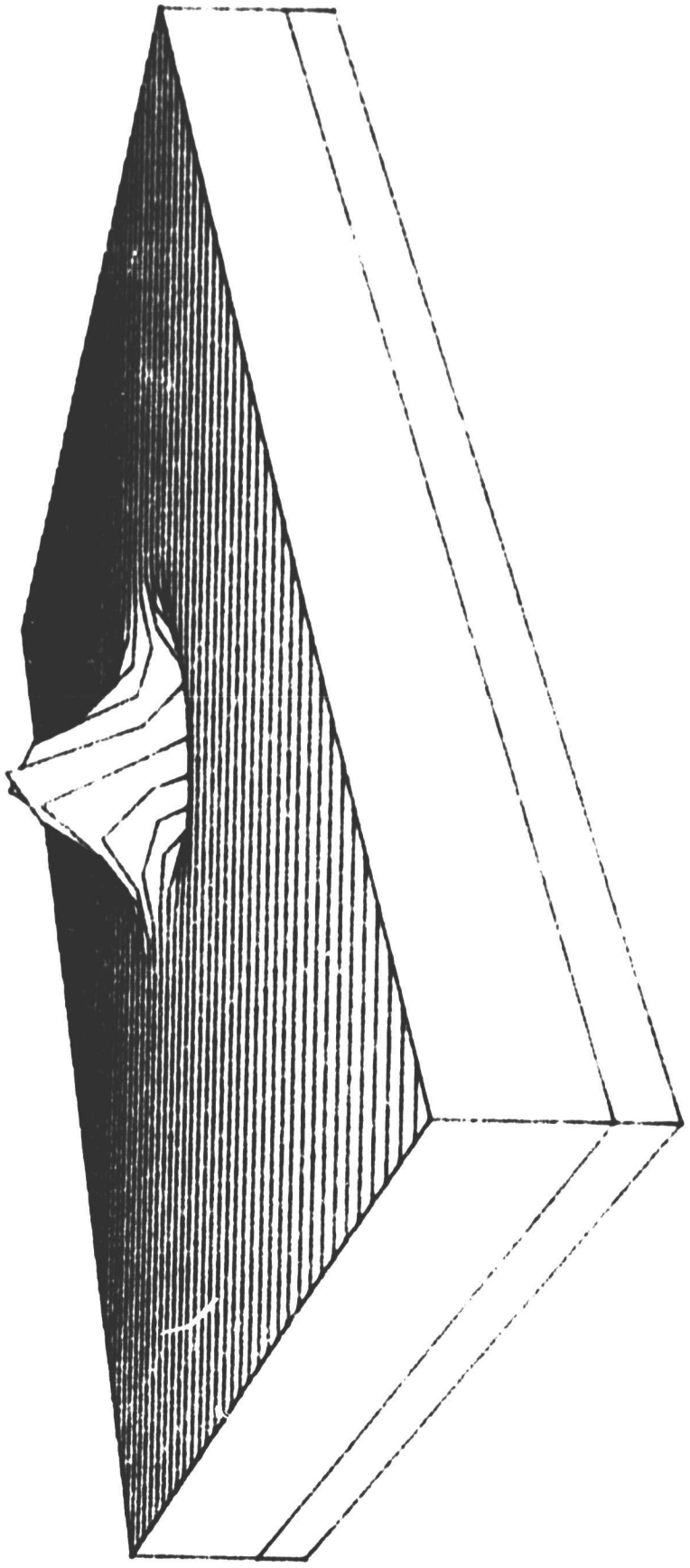

Fig. 1.8

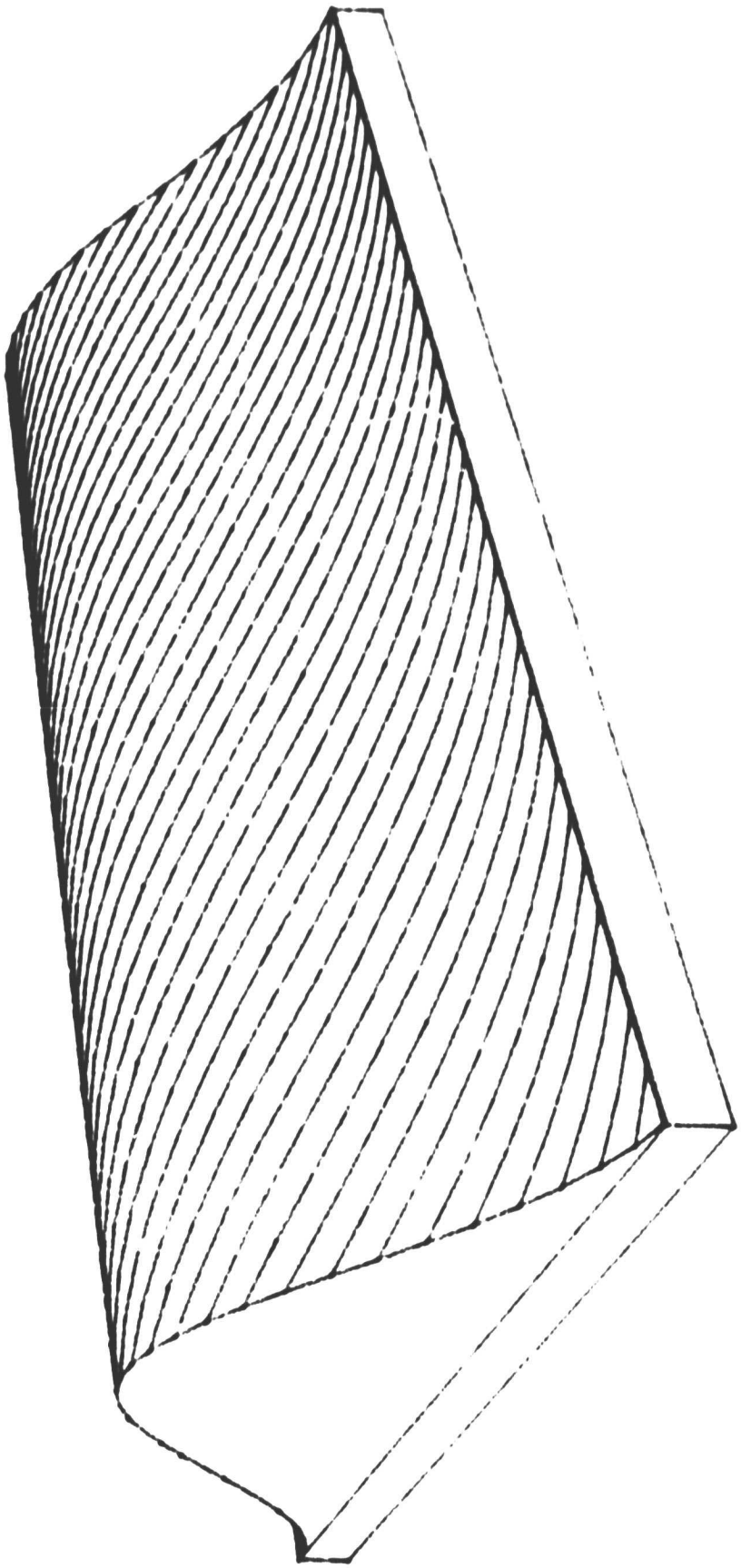

Fig. 1.9

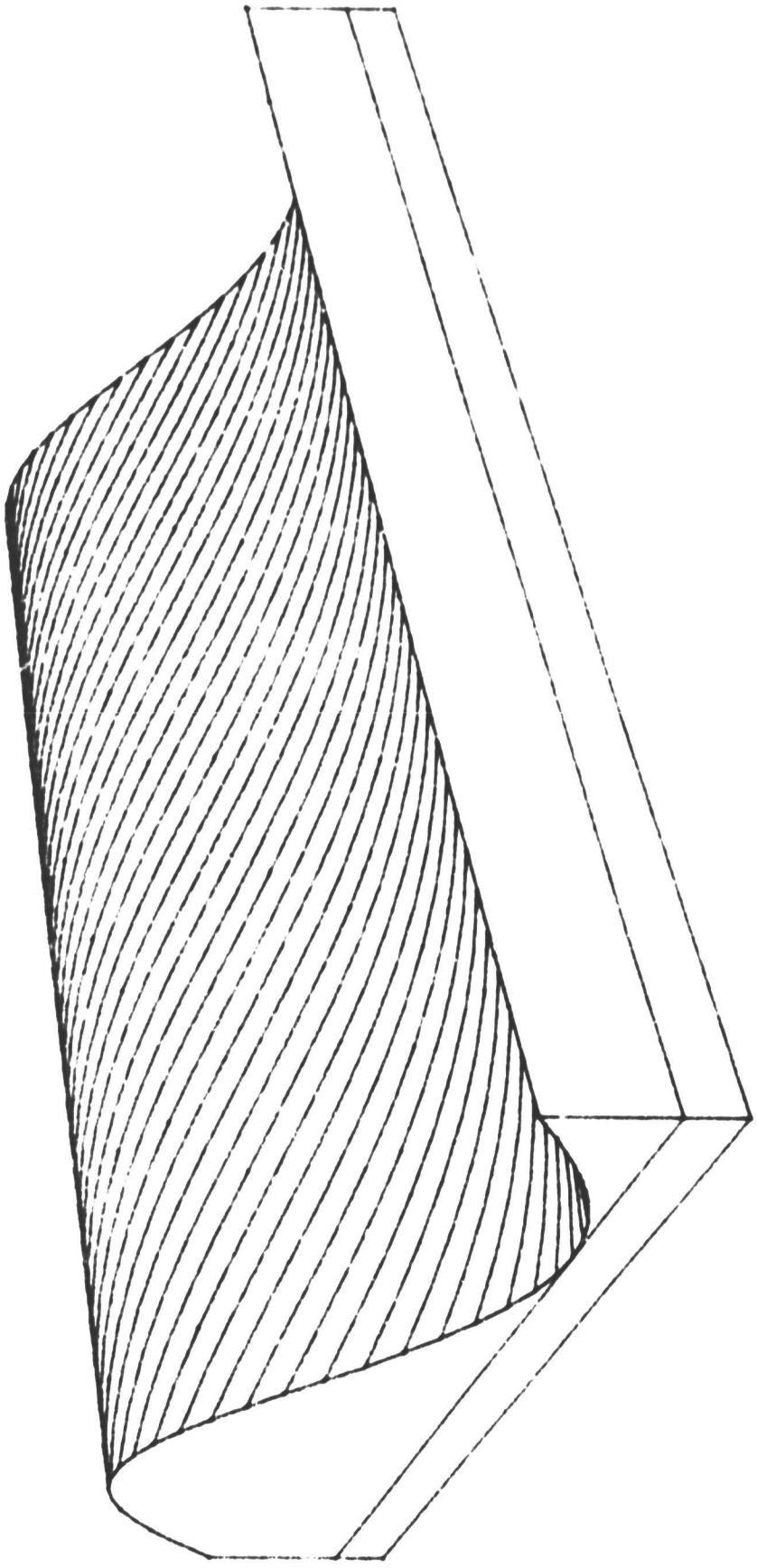

Fig. 1.10

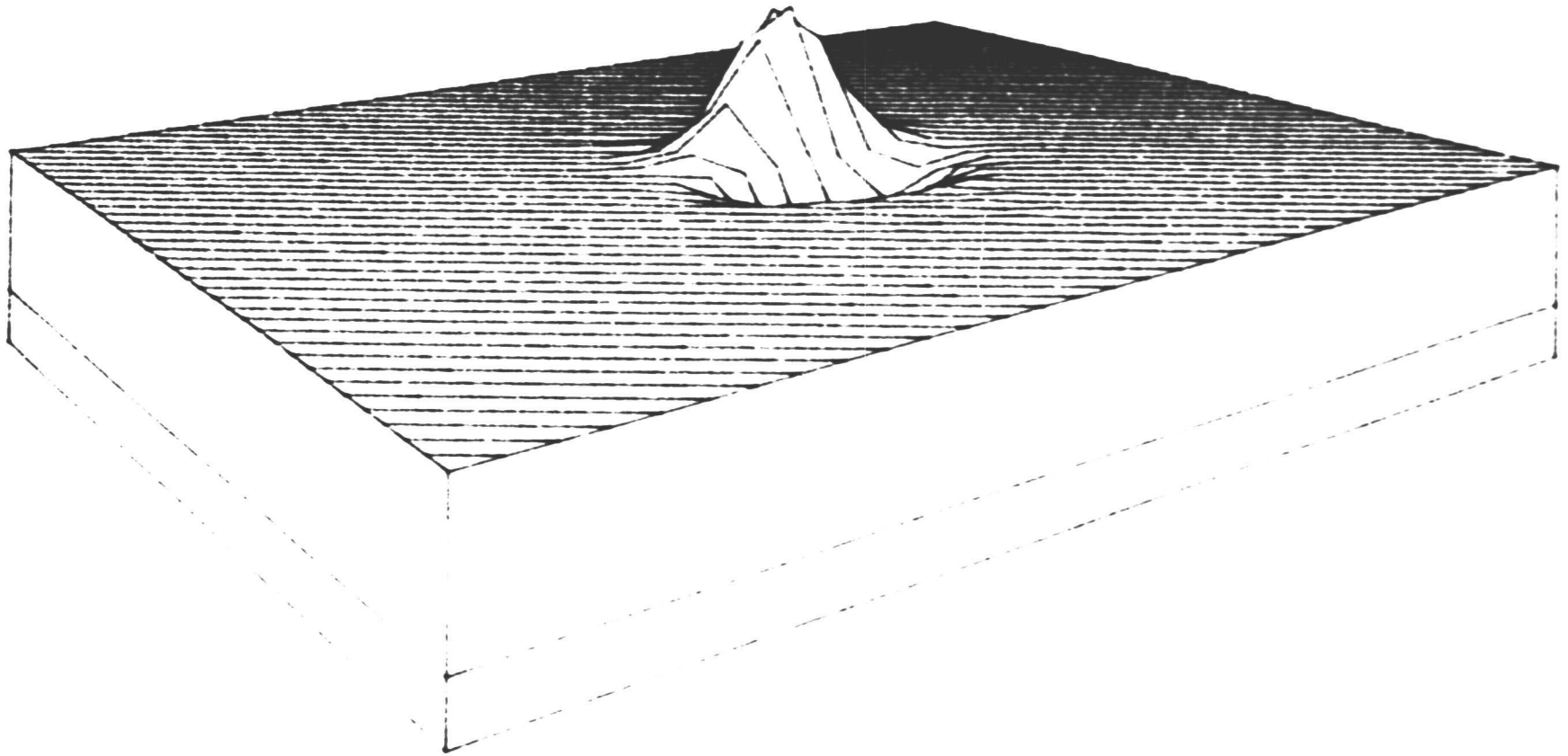

Fig. 1.11

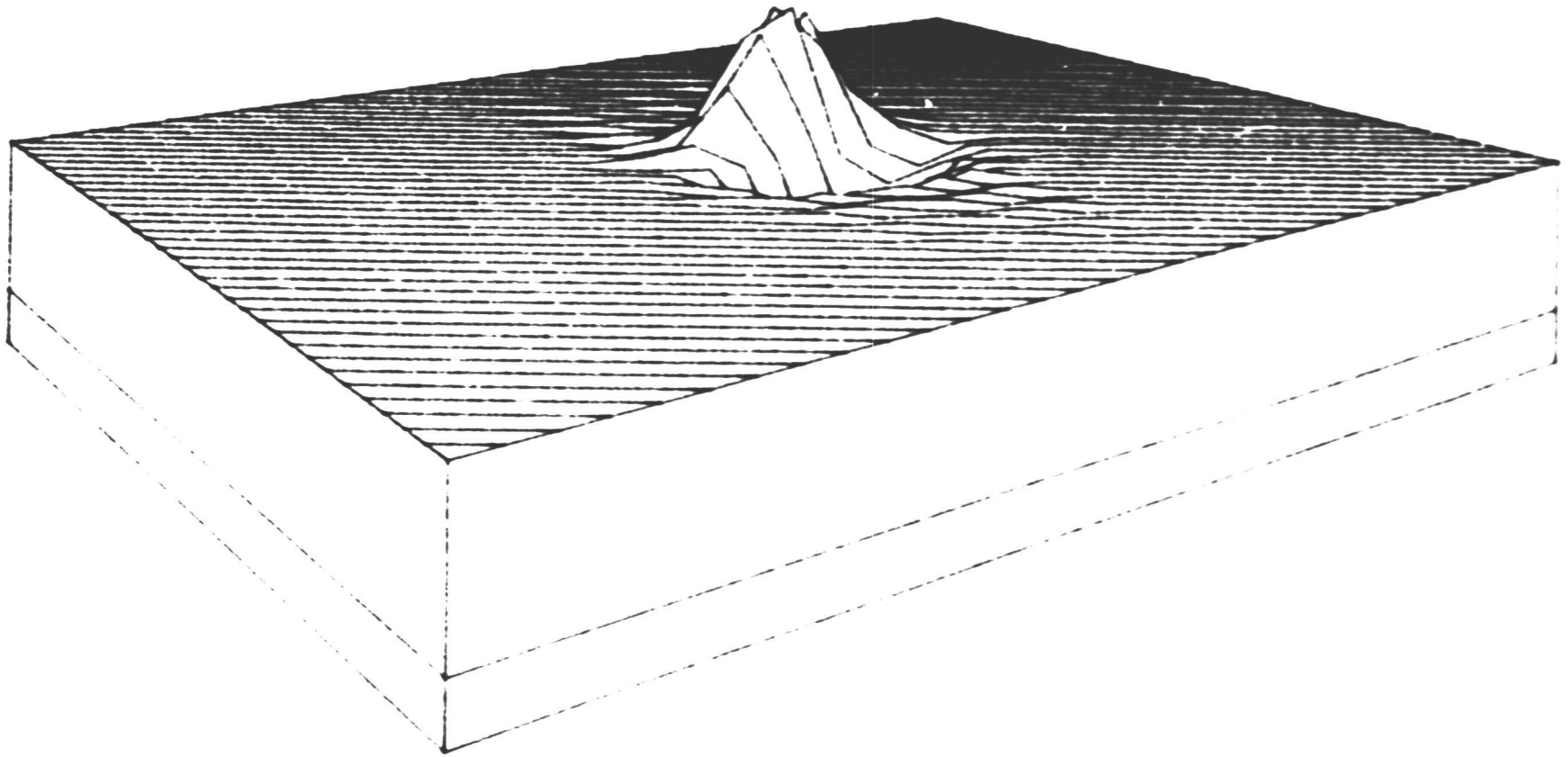

Fig. 1.12

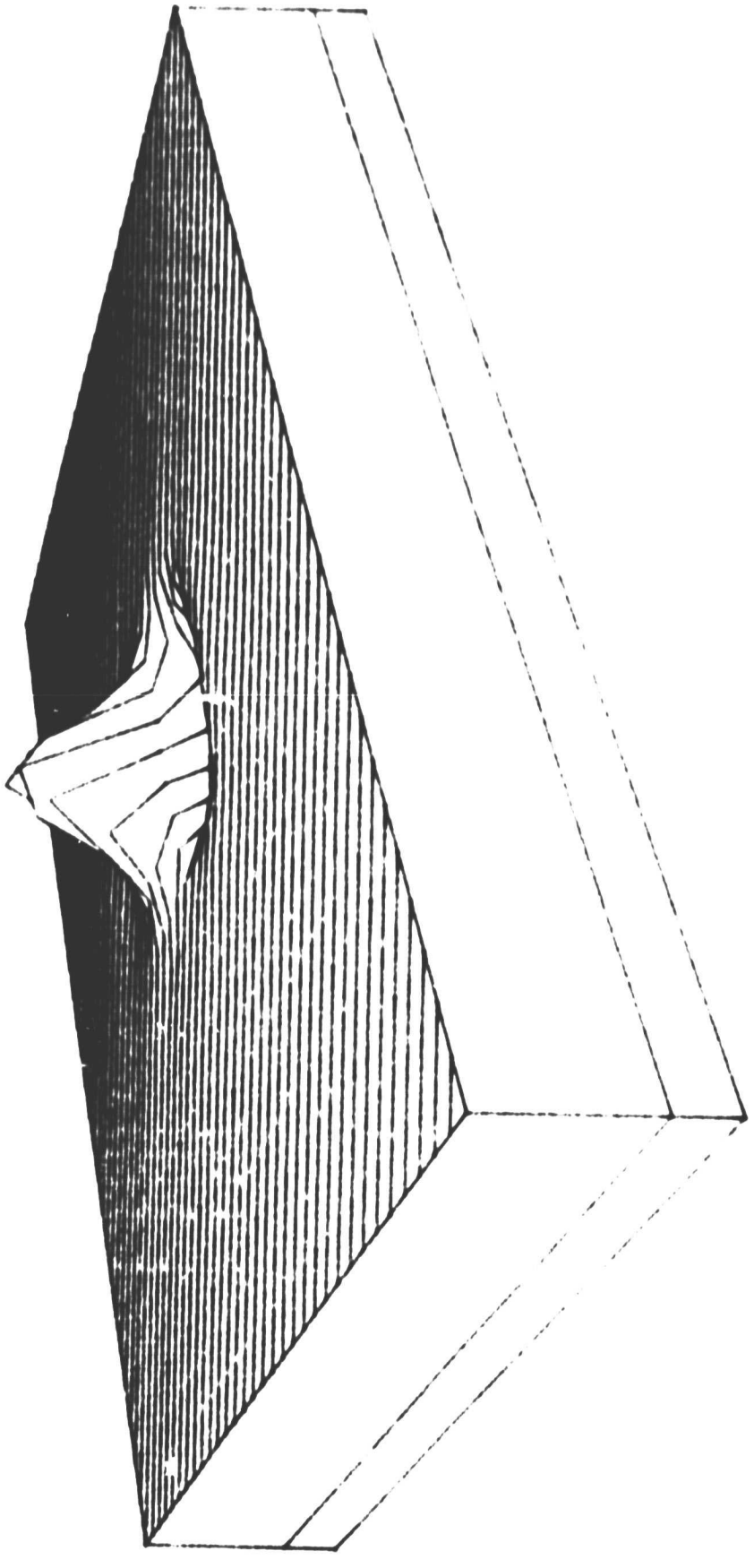

Fig. 1.13

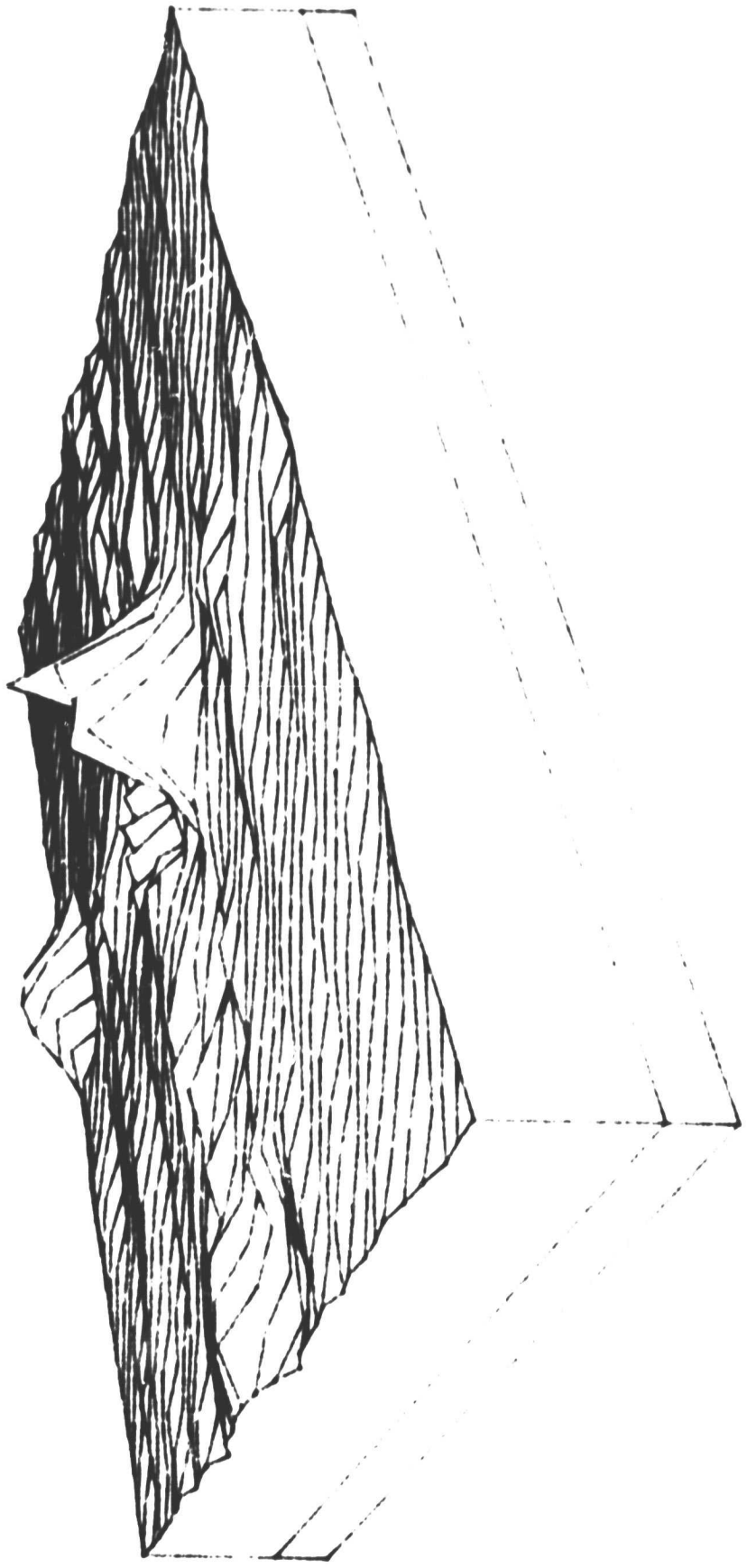

Fig. 1.14

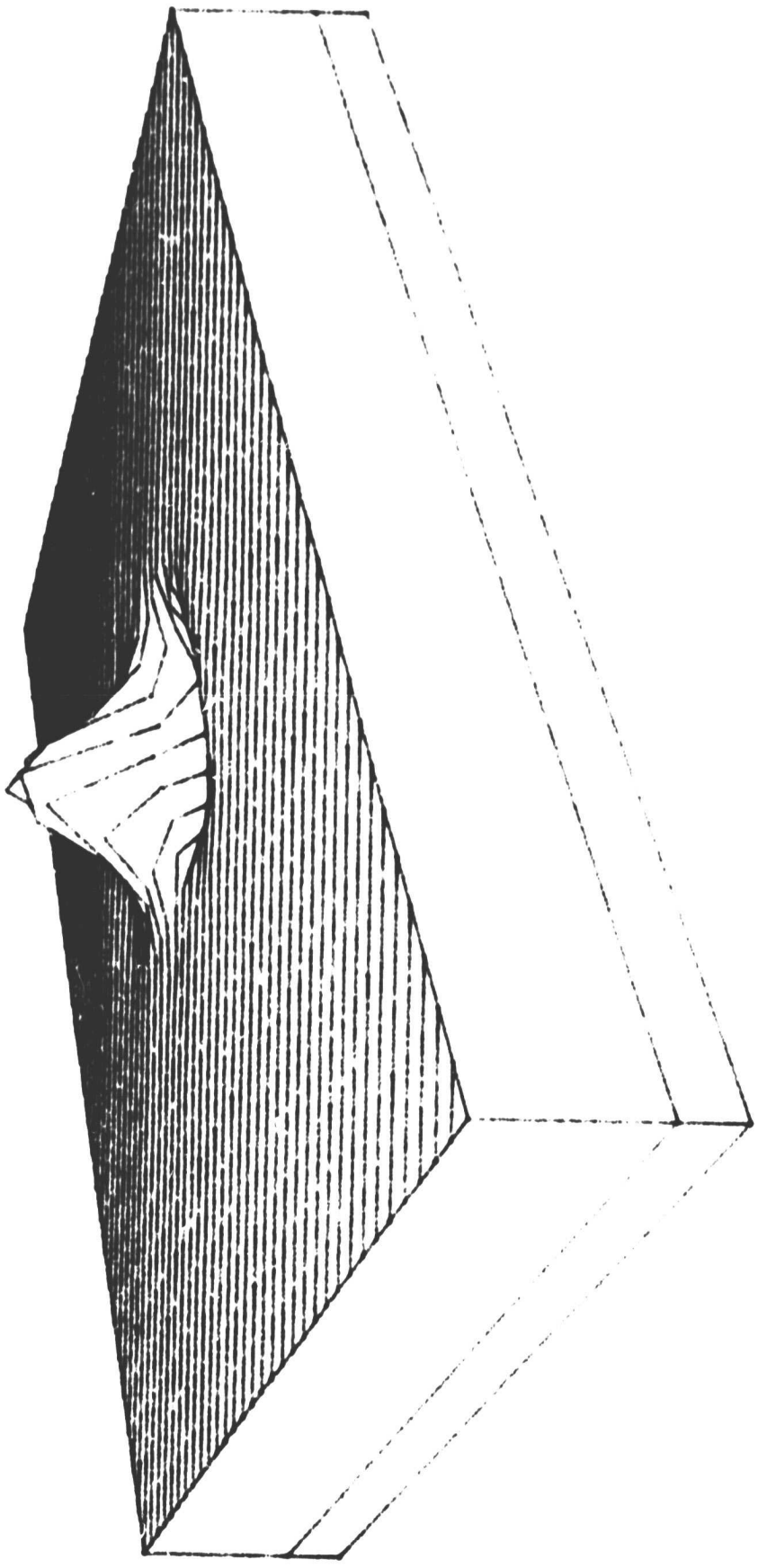

Fig. 1.15

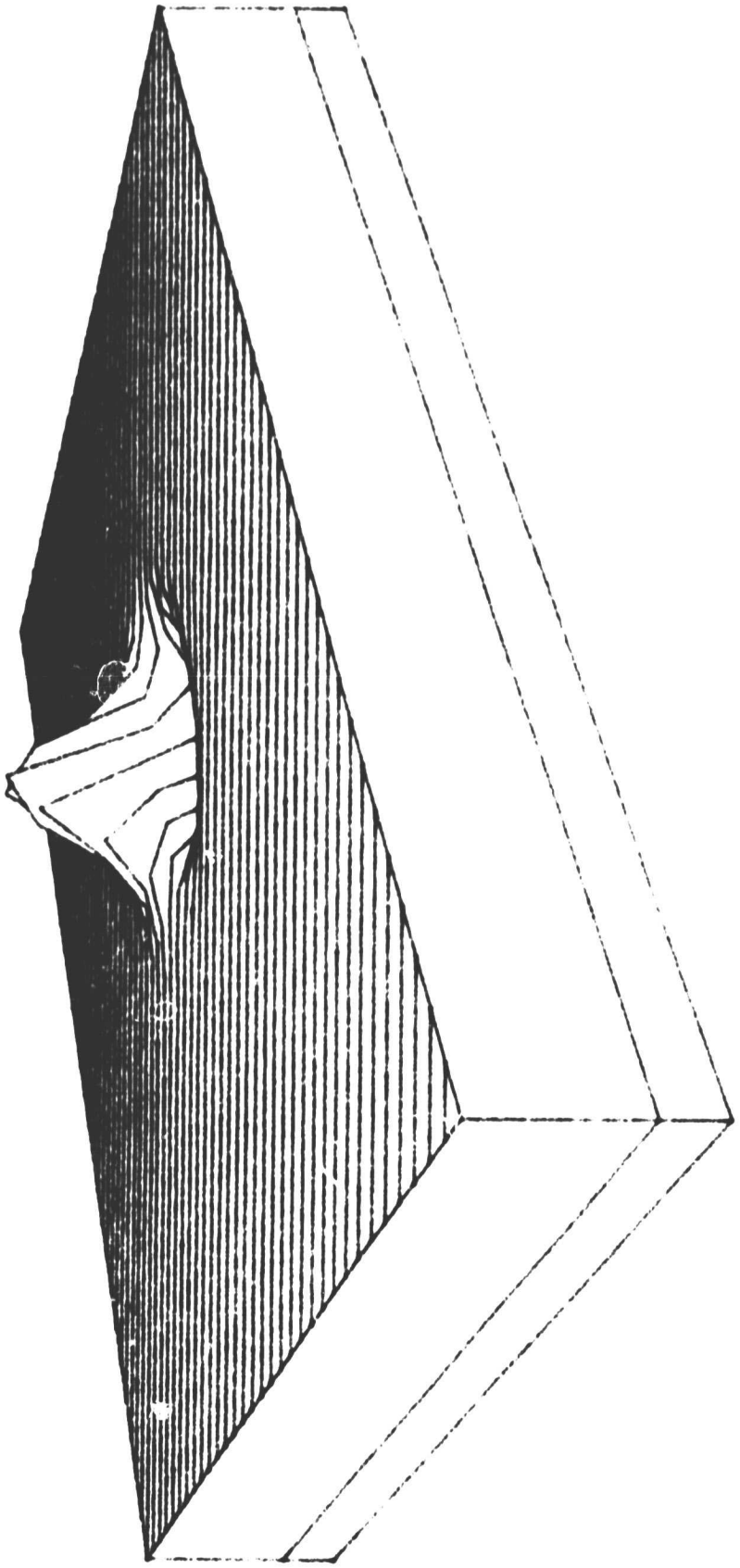

Fig. 1.16

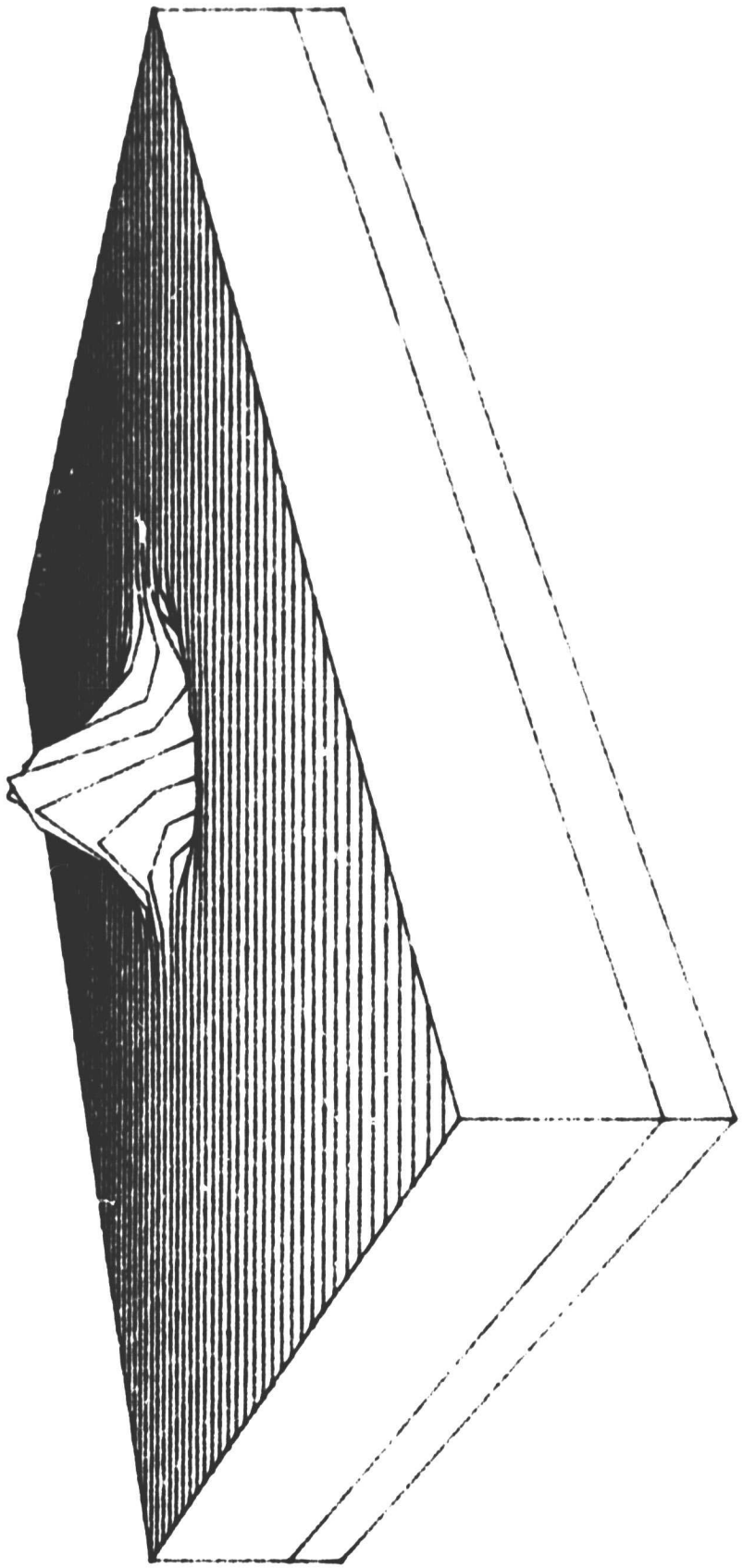

Fig. 1.17

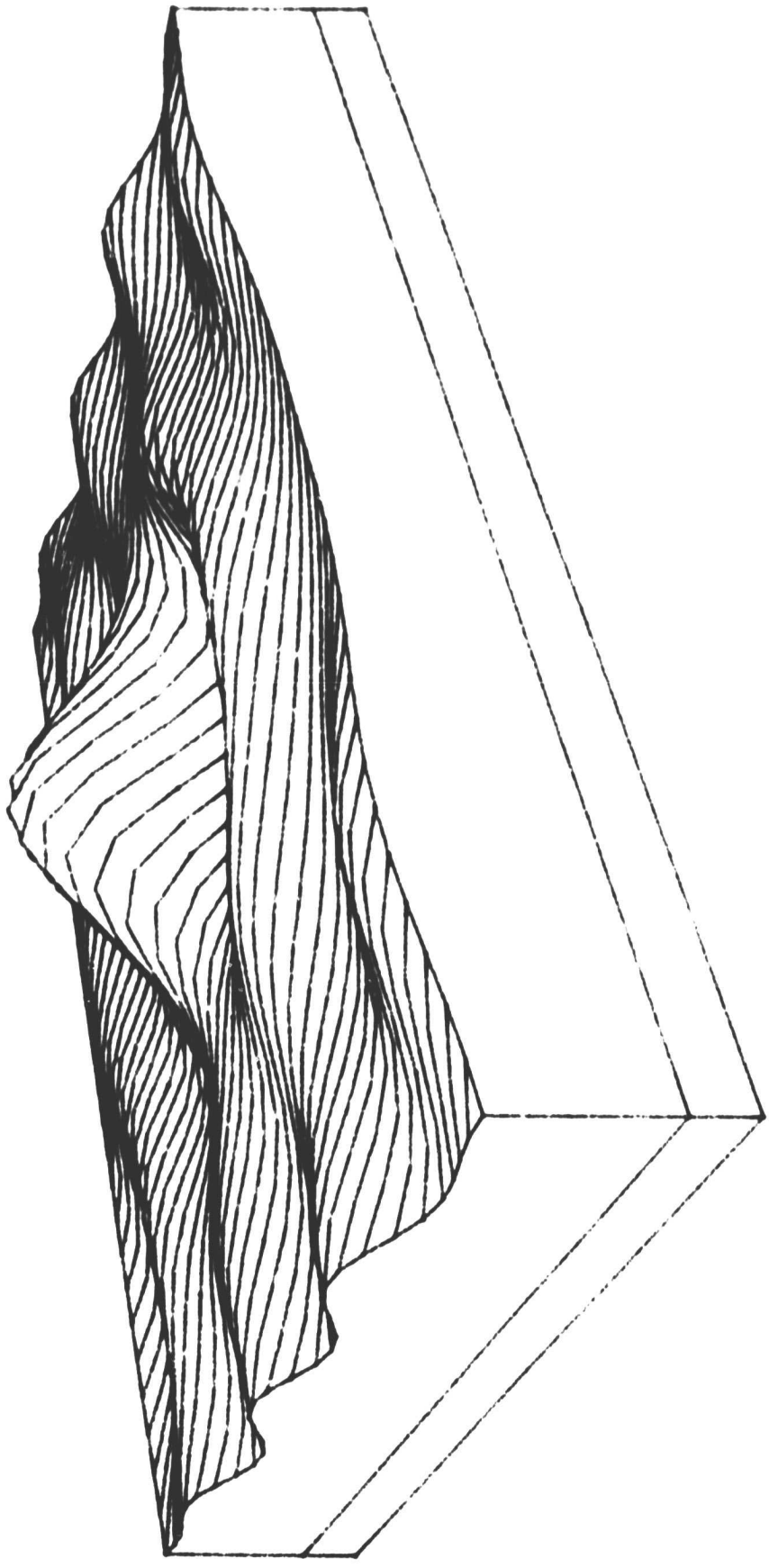

Fig. 1.18

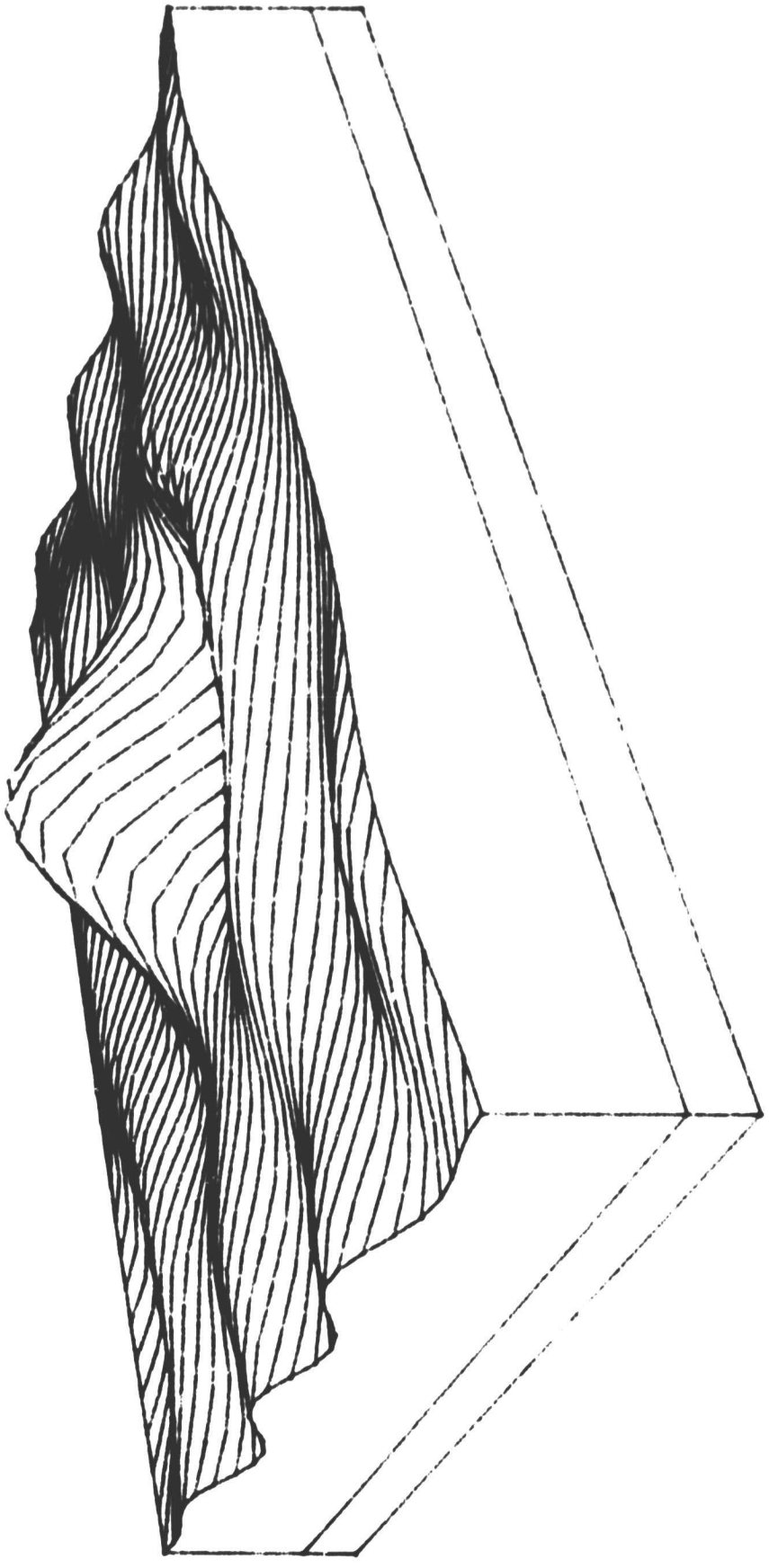

Fig. 1.19

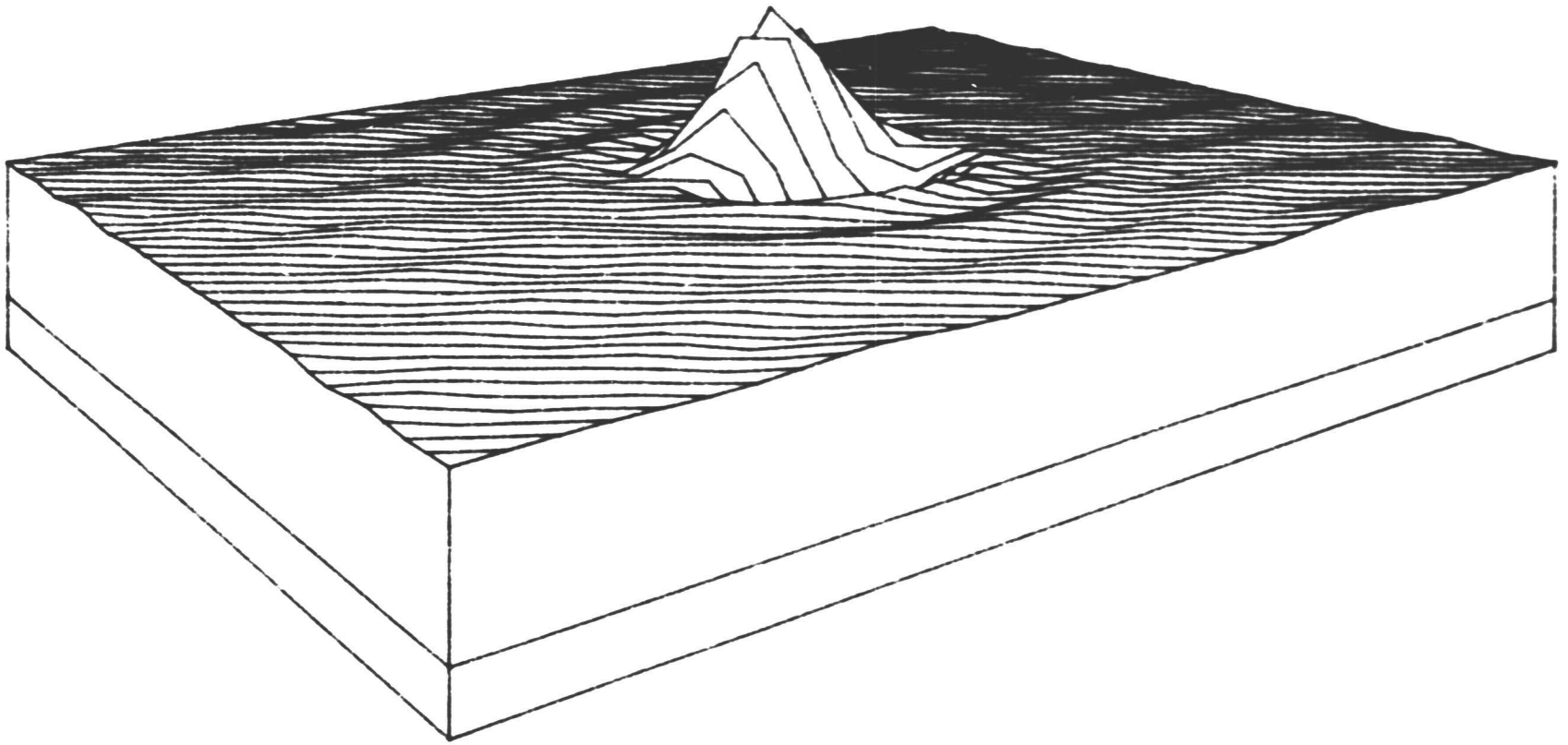

Fig. 1.20

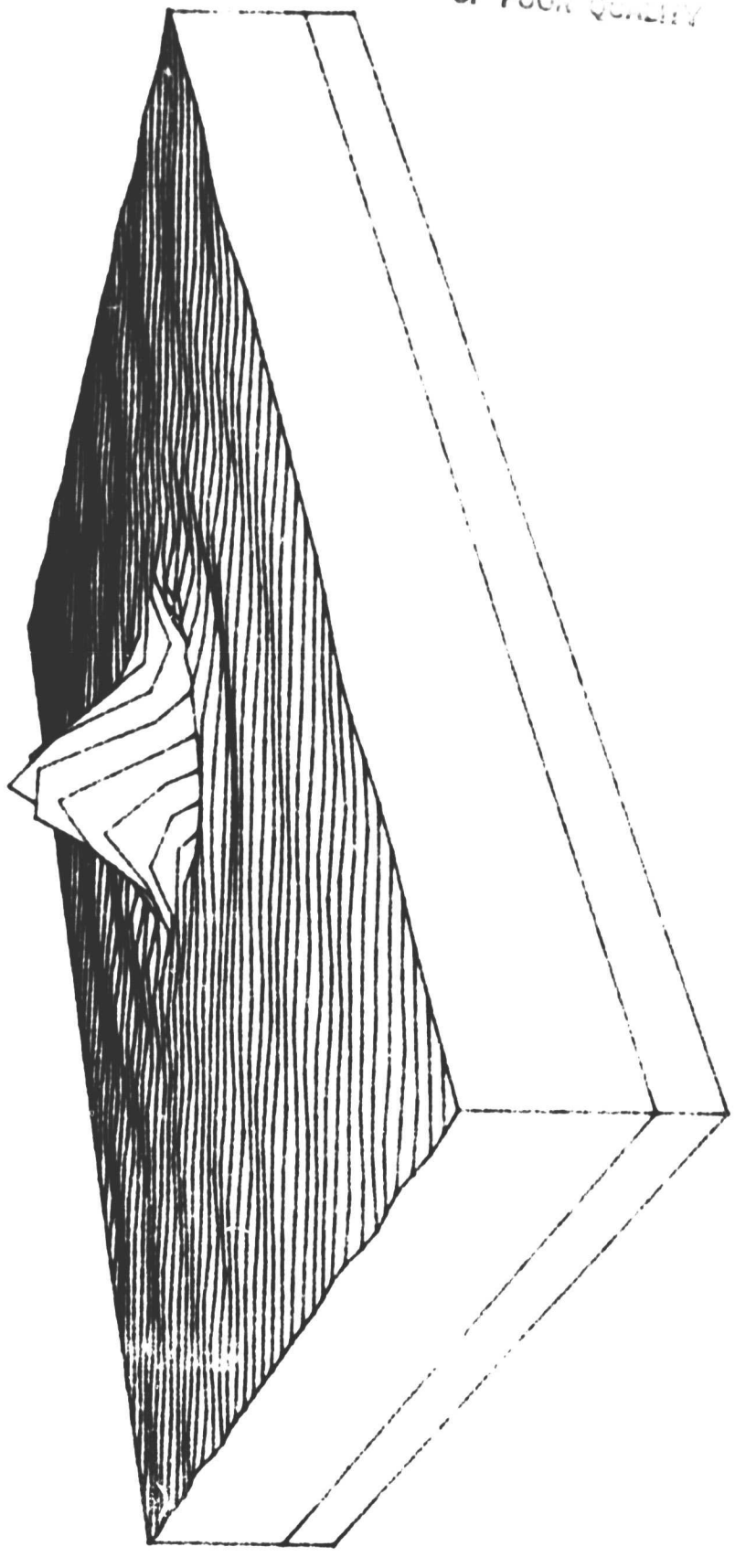

Fig. 1.21

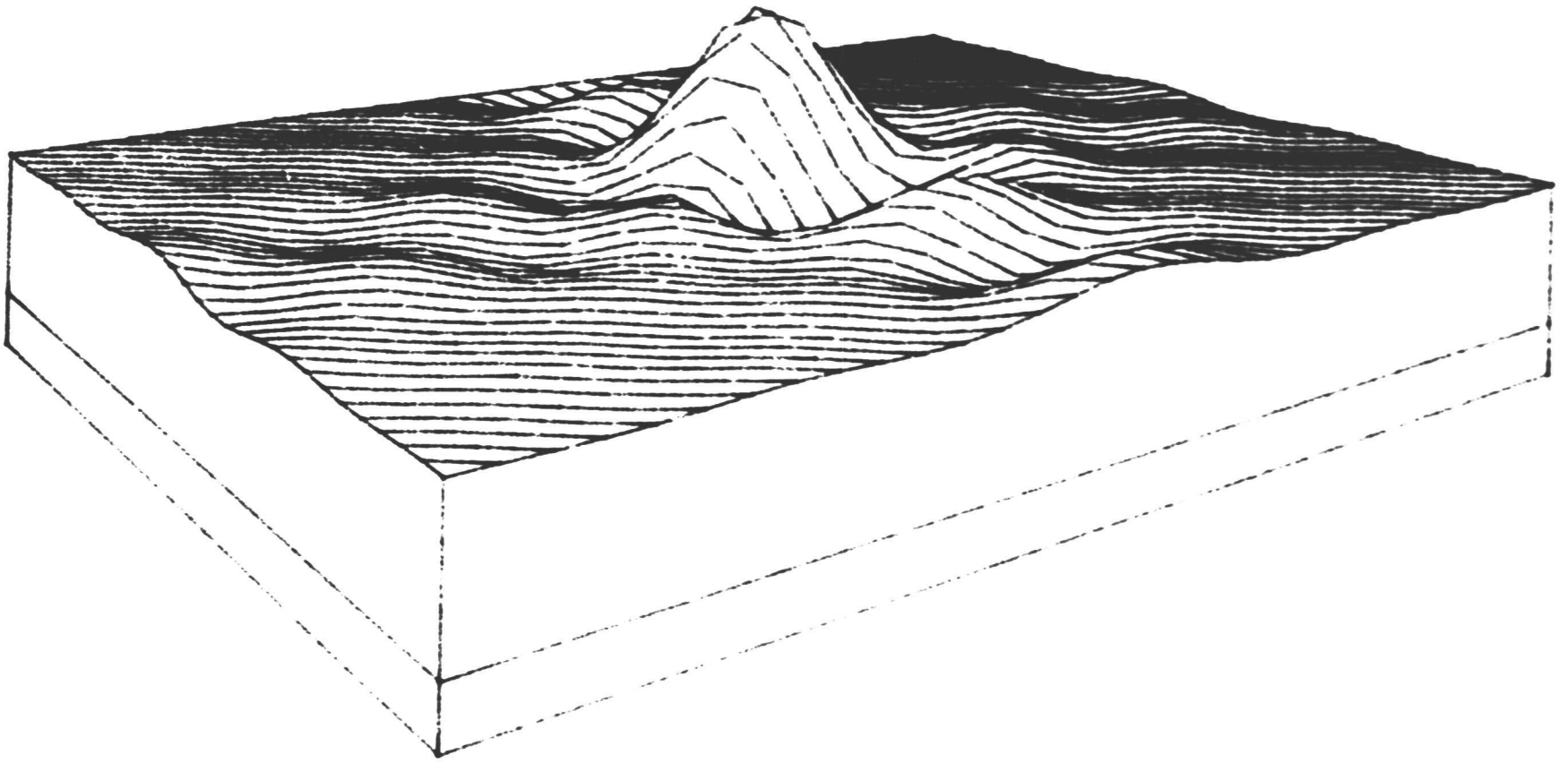

Fig. 1.22

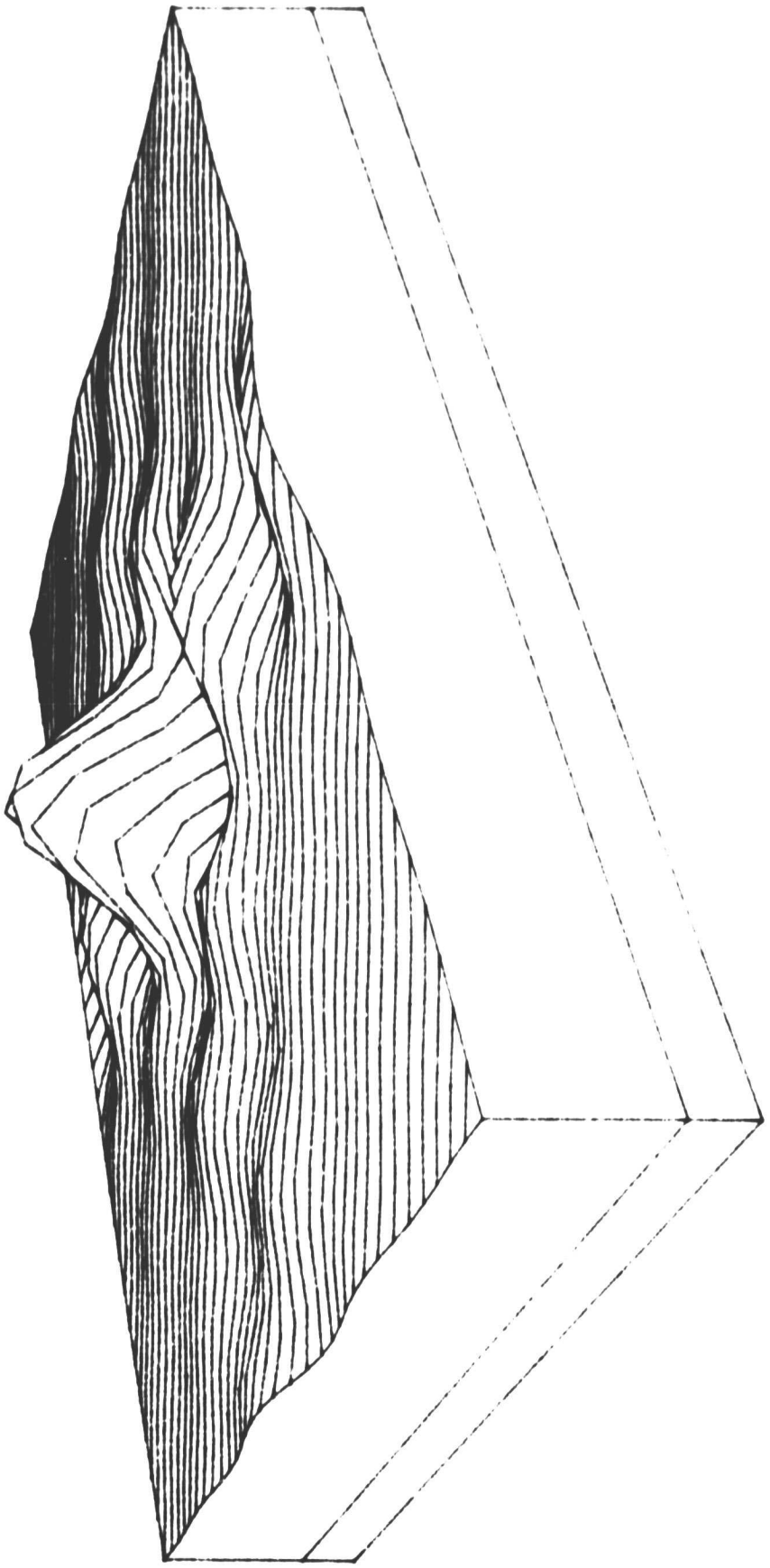

Fig. 1.23

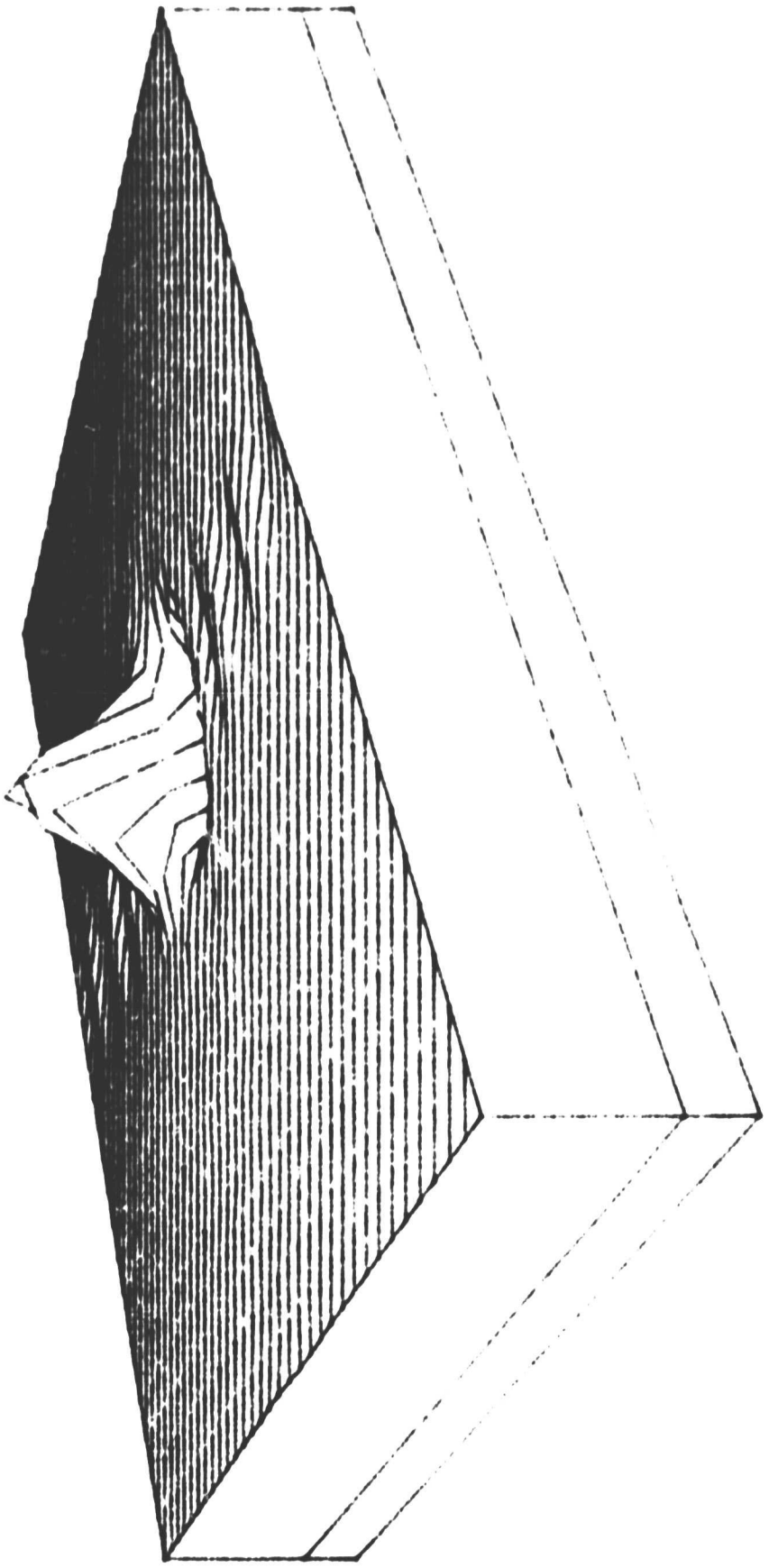

Fig. 1.24

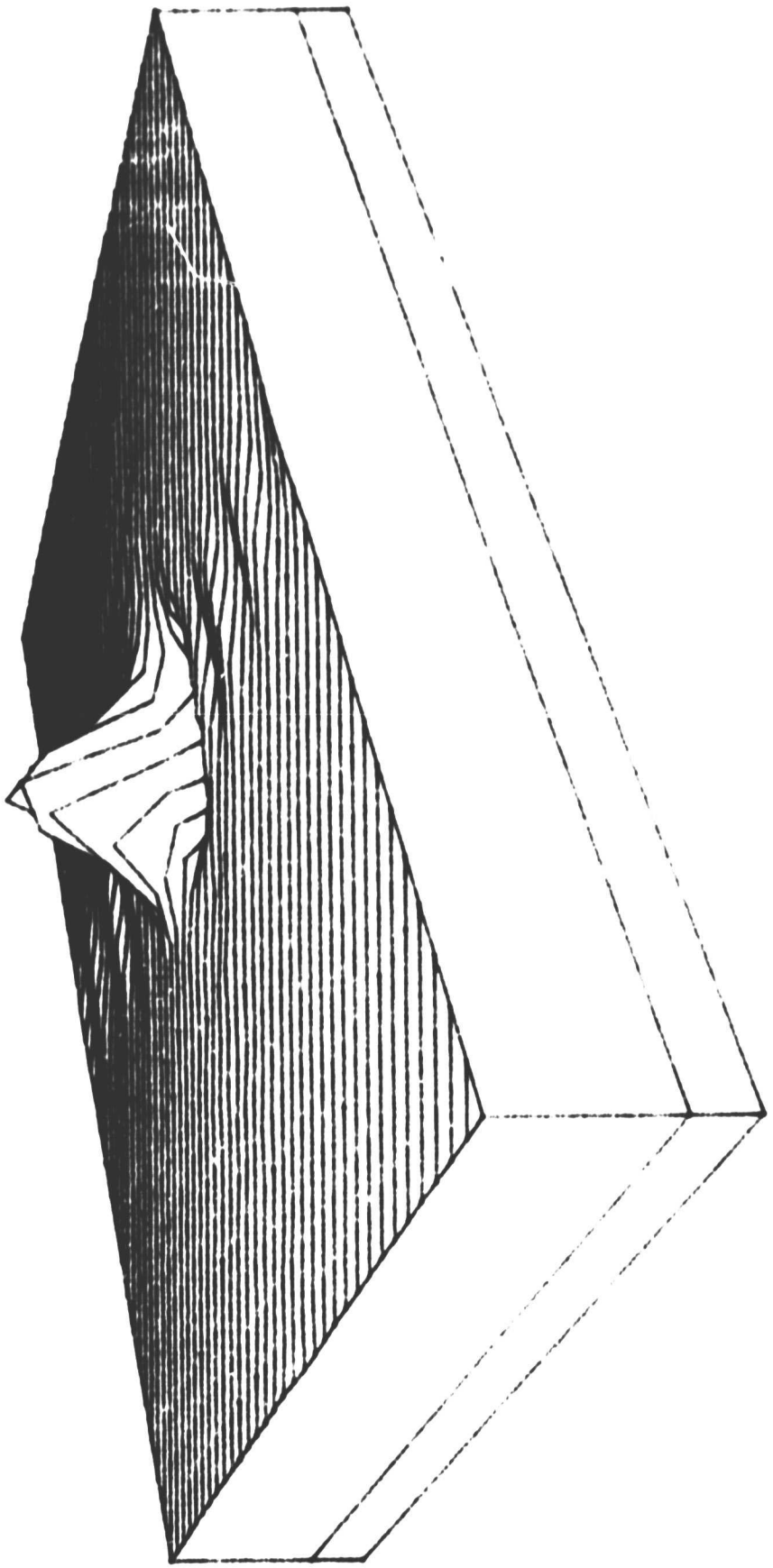

Fig. 1.25

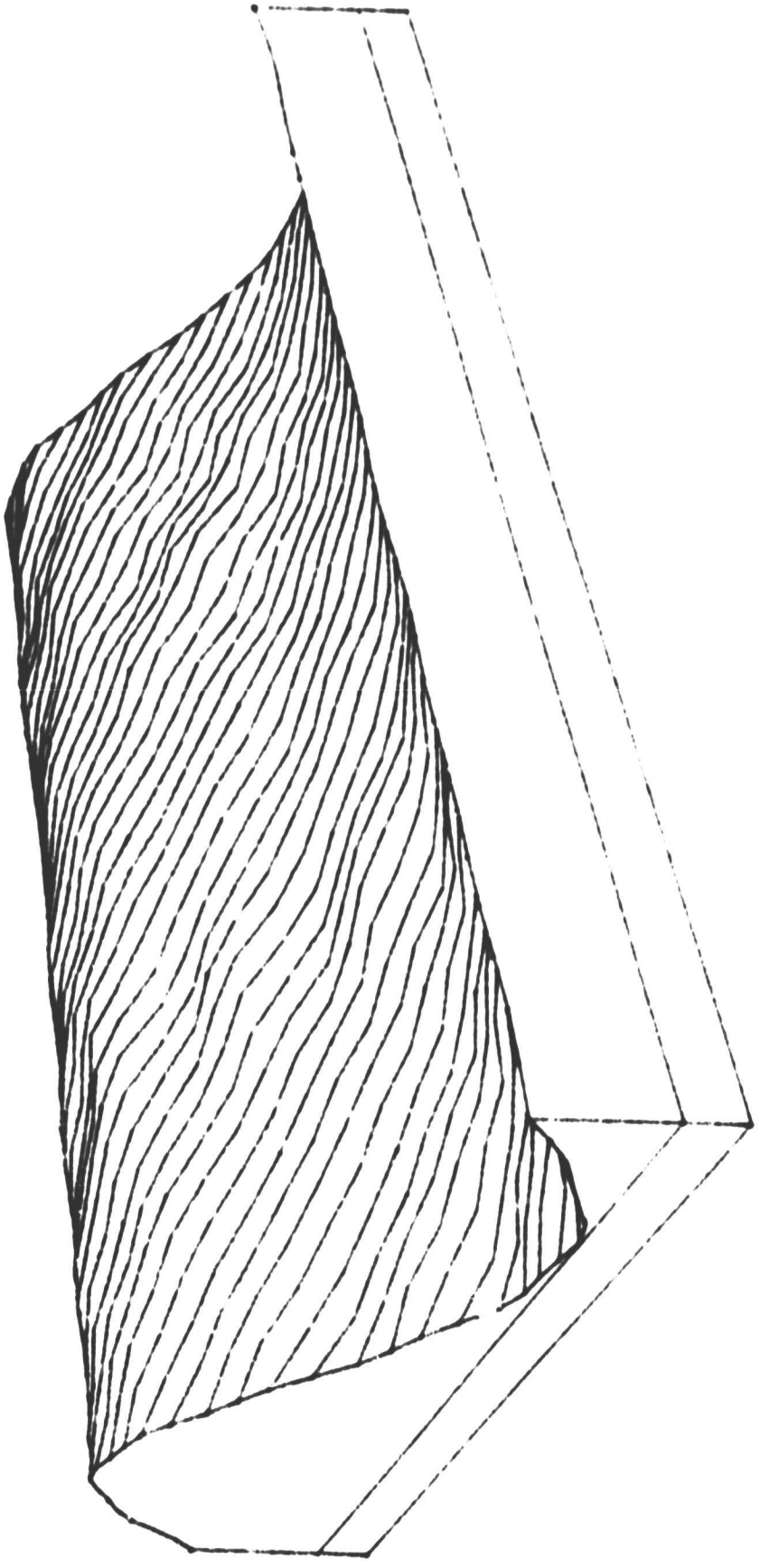

Fig. 1.26

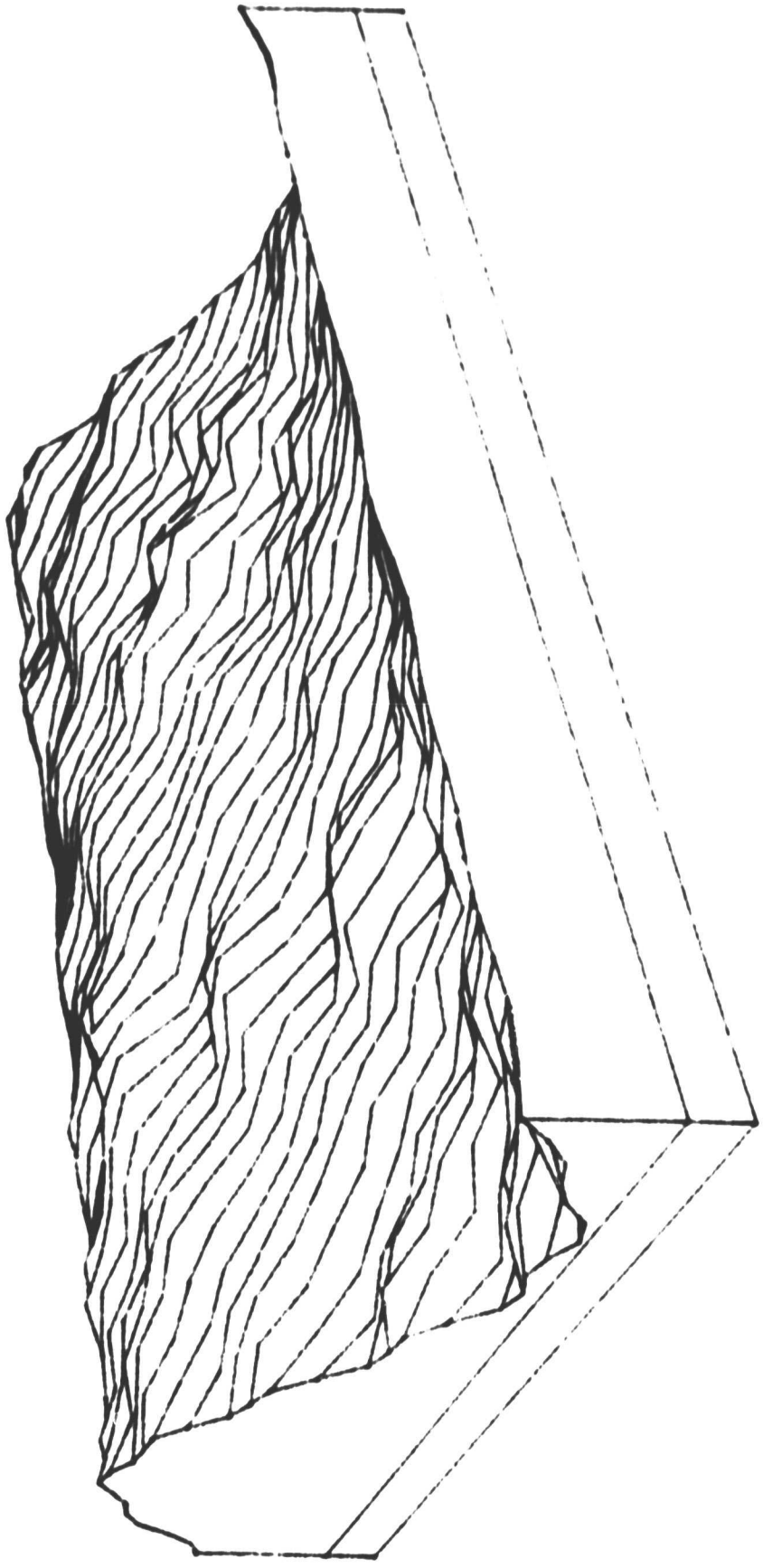

Fig. 1.27

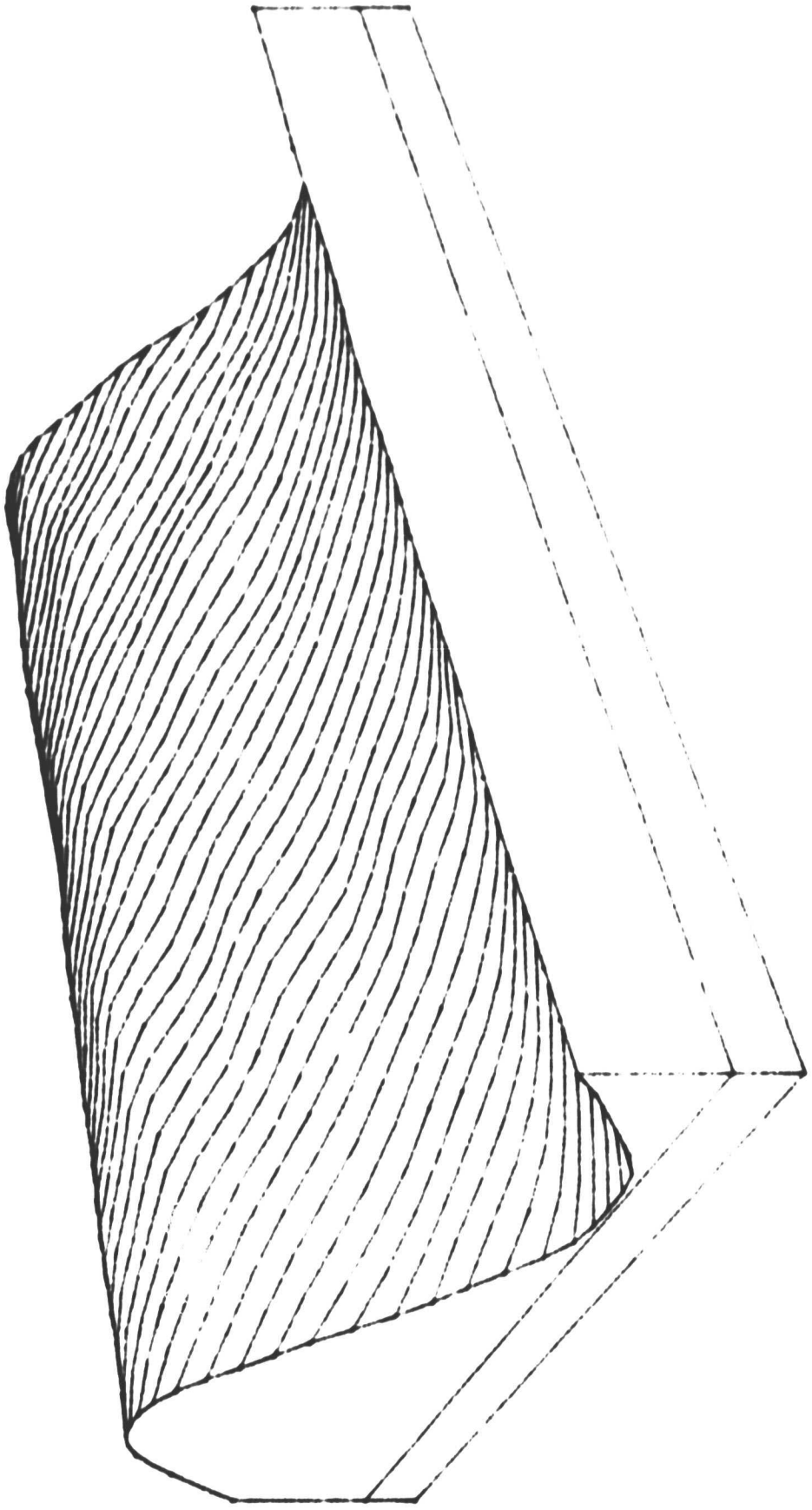

Fig. 1.28

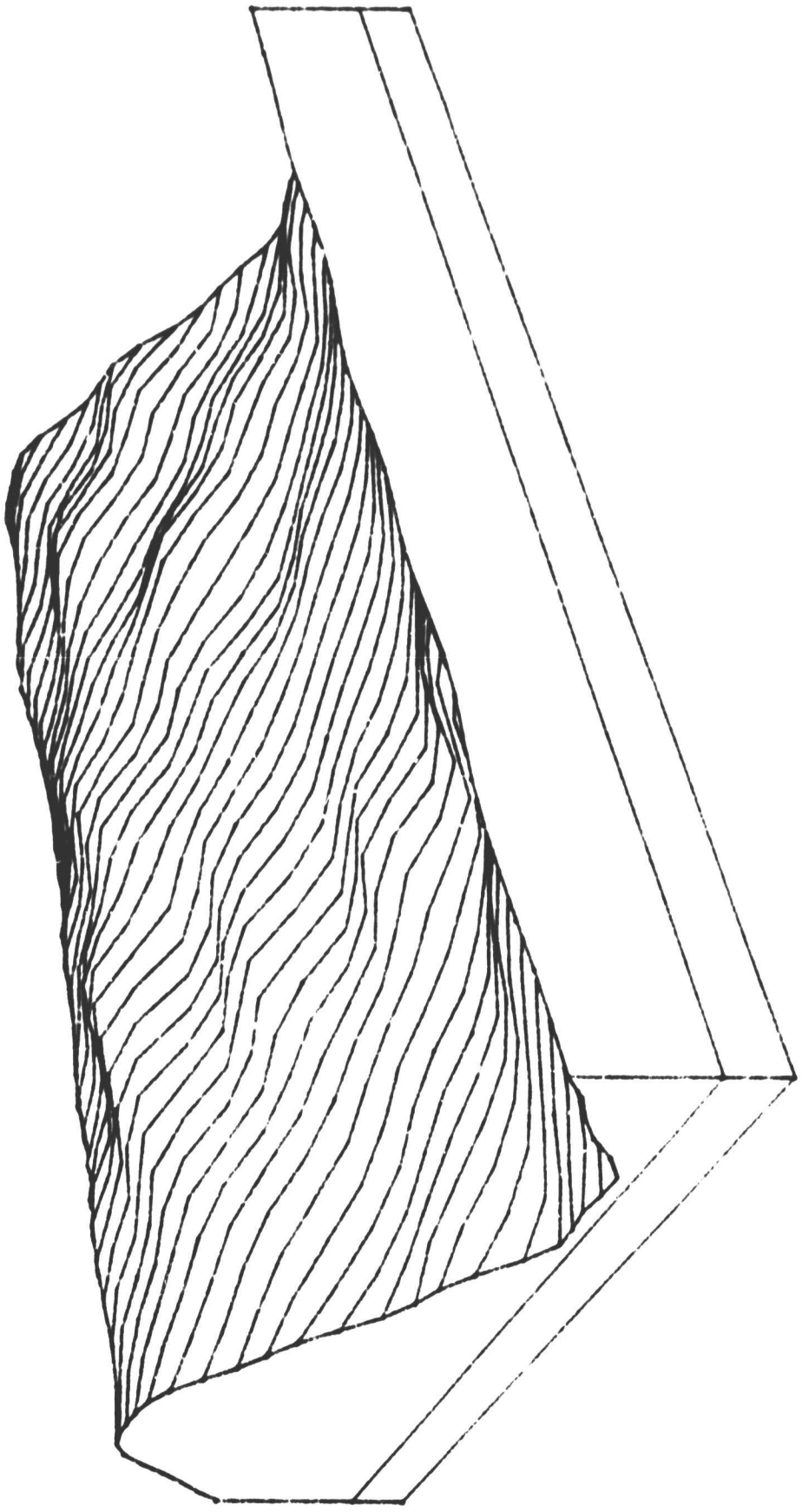

Fig. 1.29

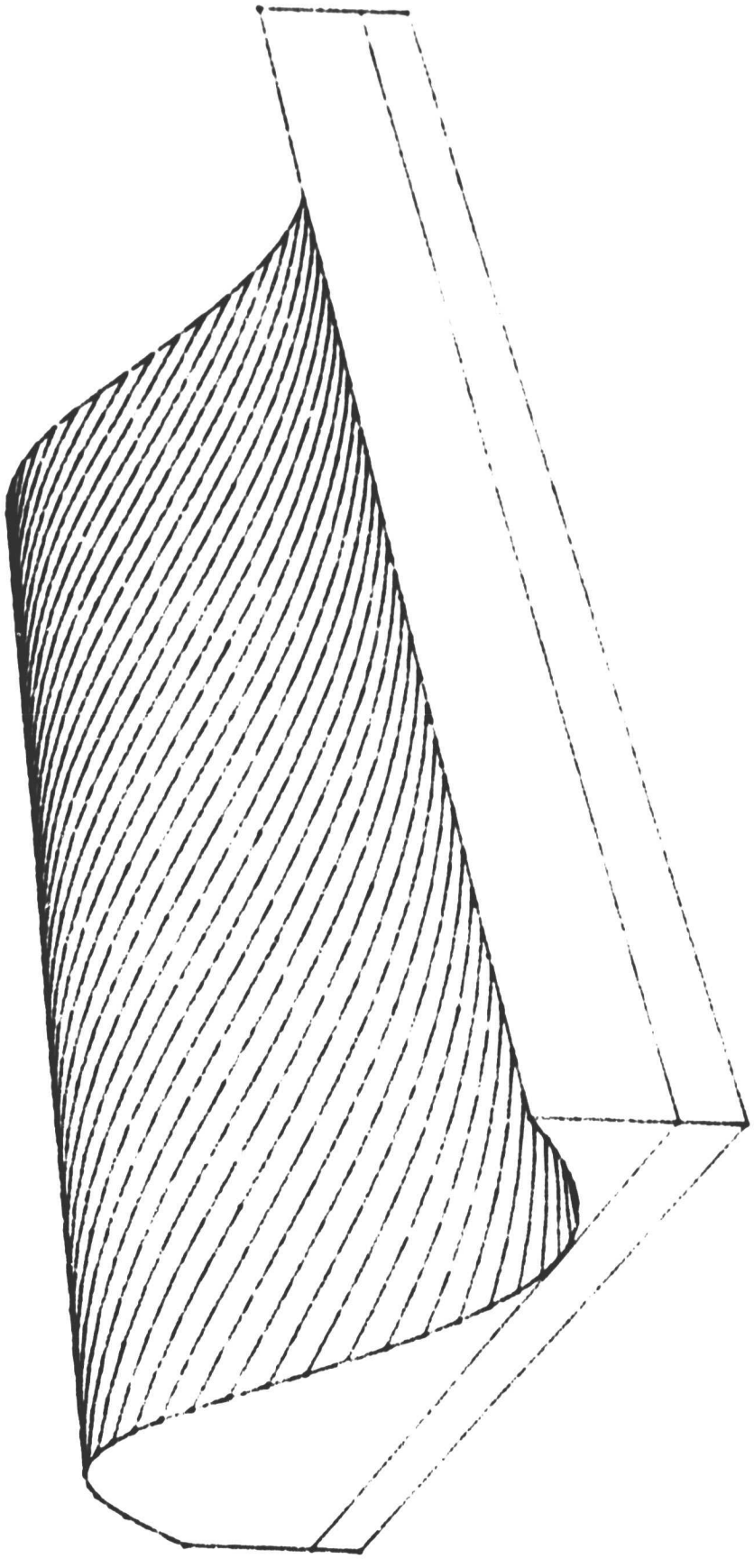

Fig. 1.30

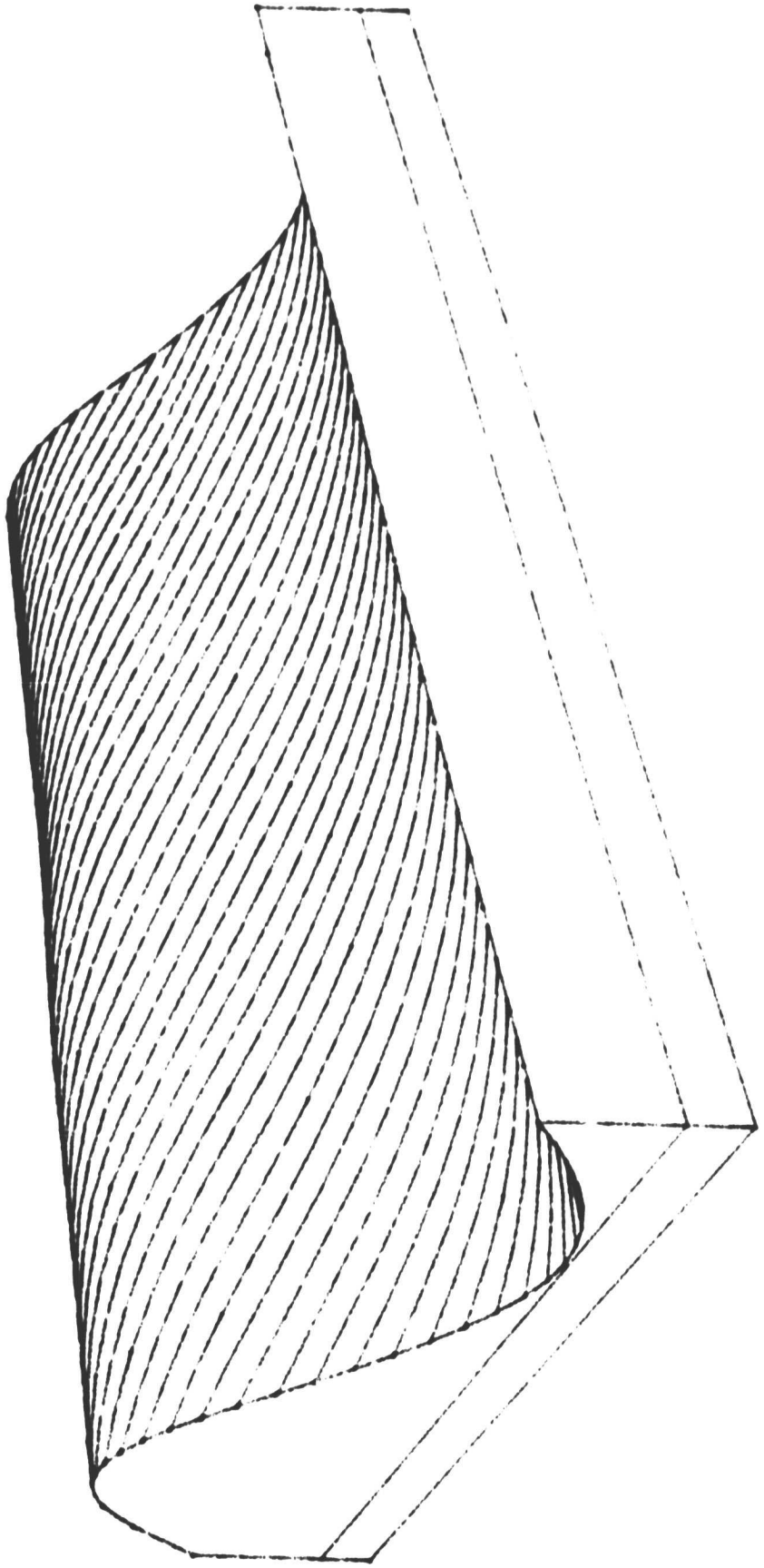

Fig. 1.31

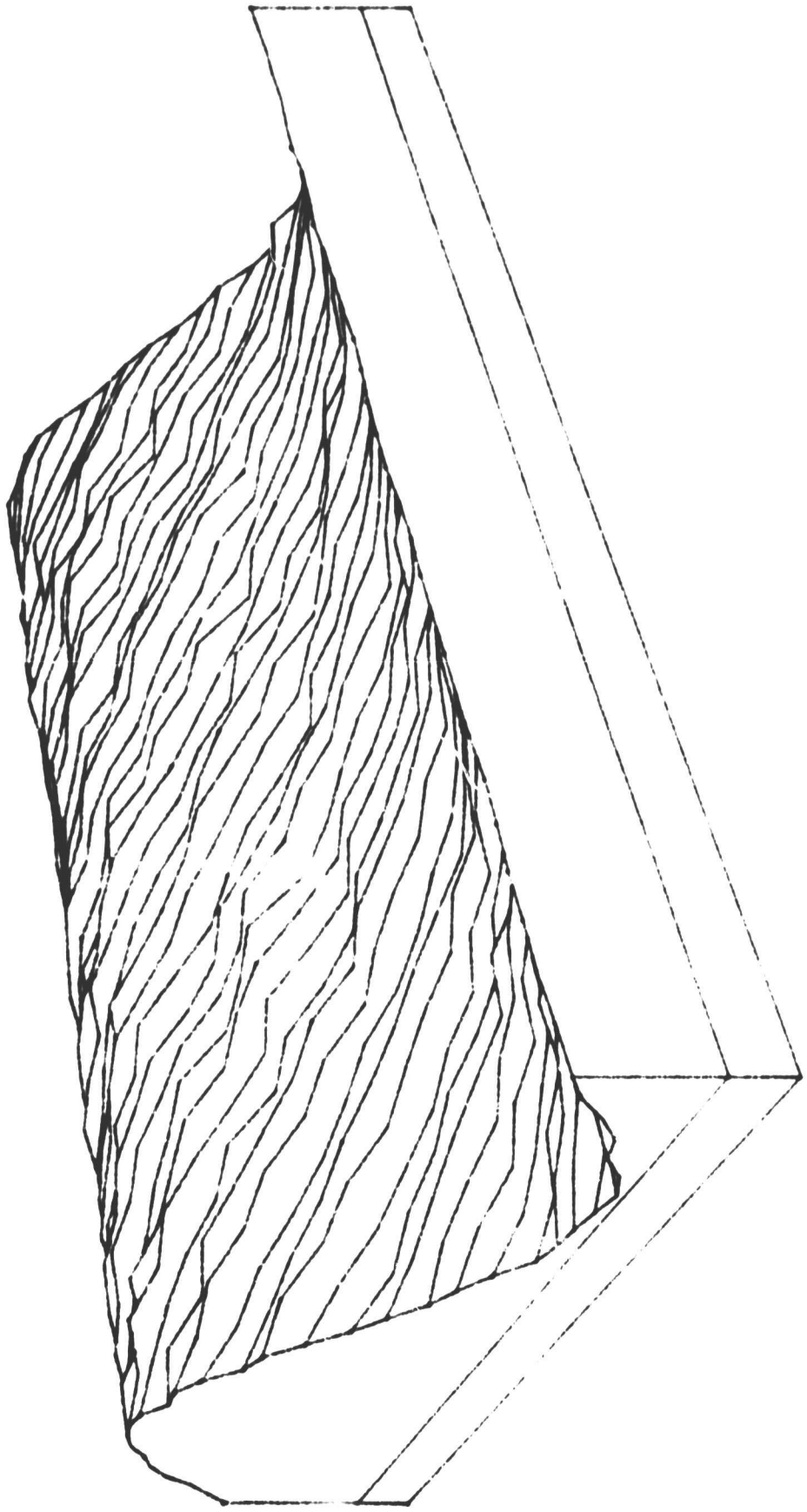

Fig. 1.32

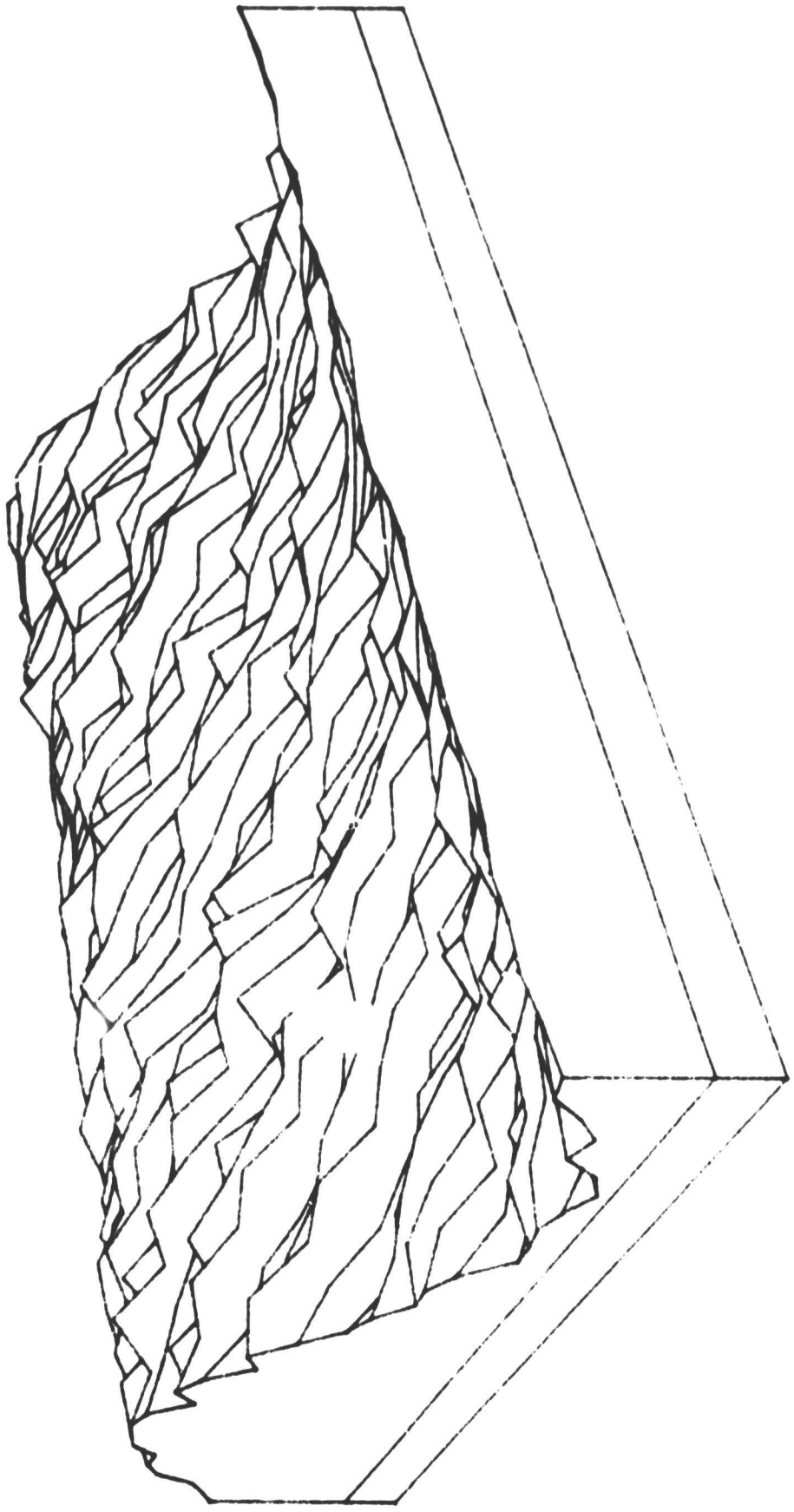

Fig. 1.33

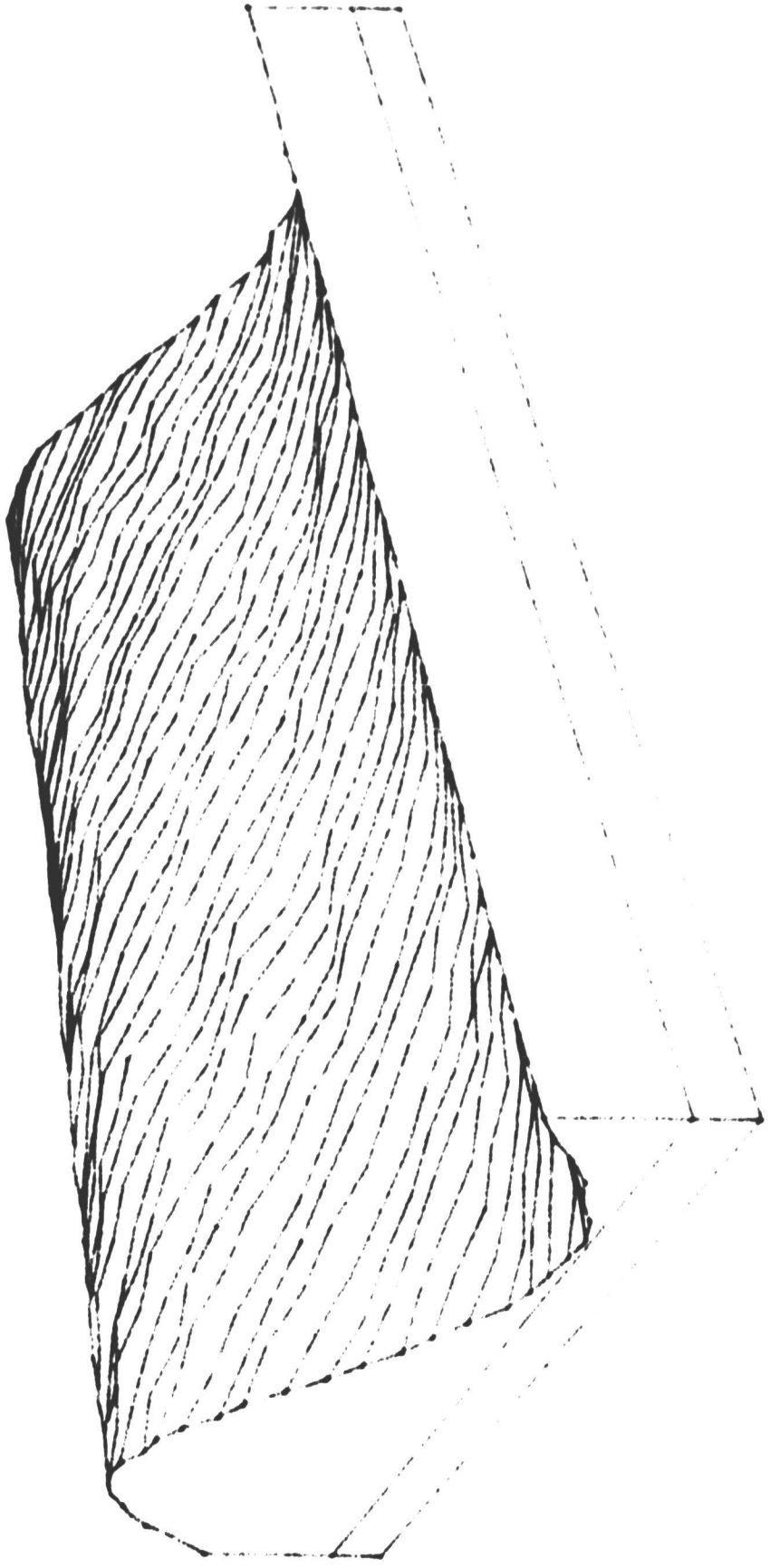

Fig. 1.34

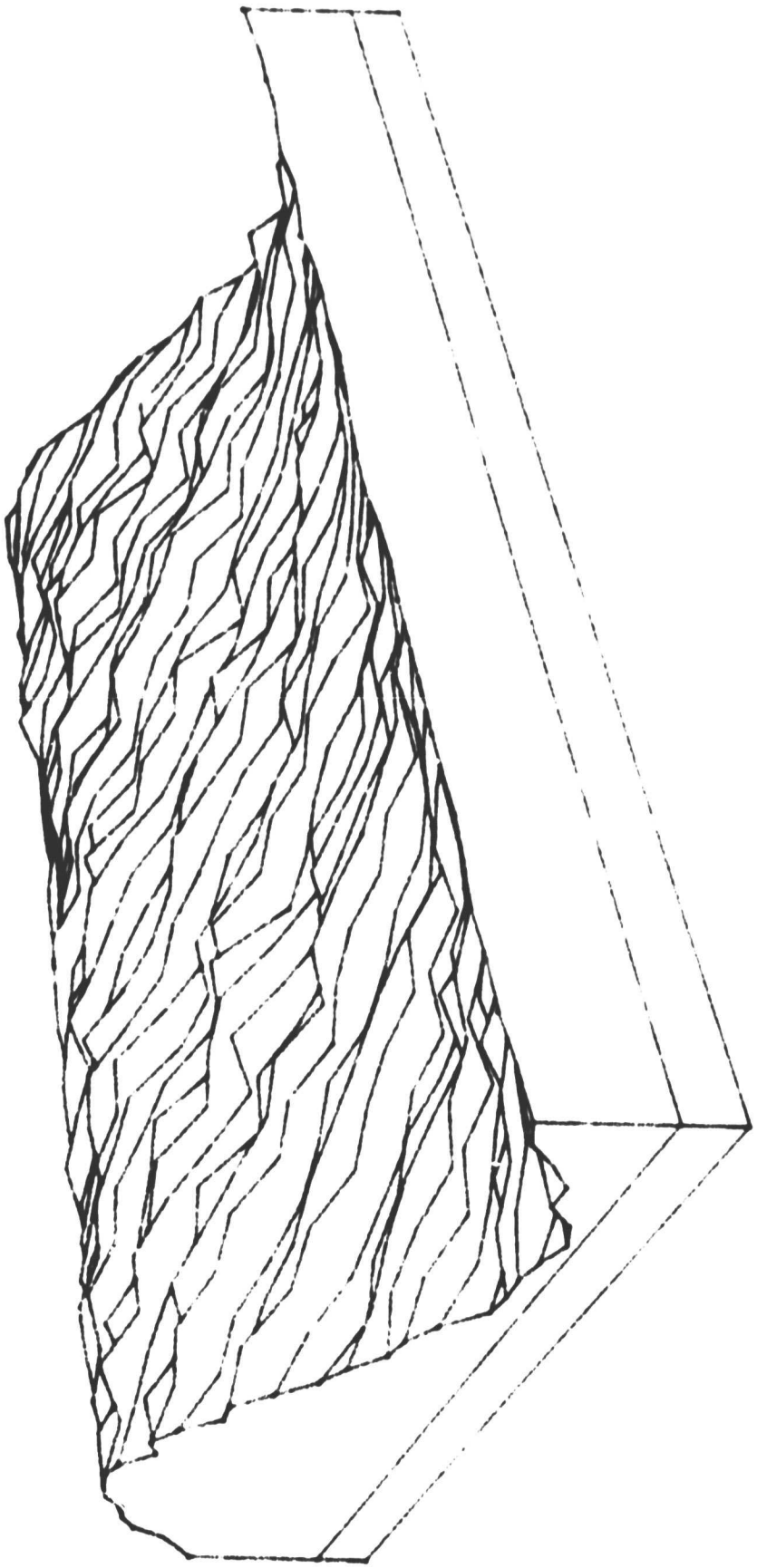

Fig. 1.35

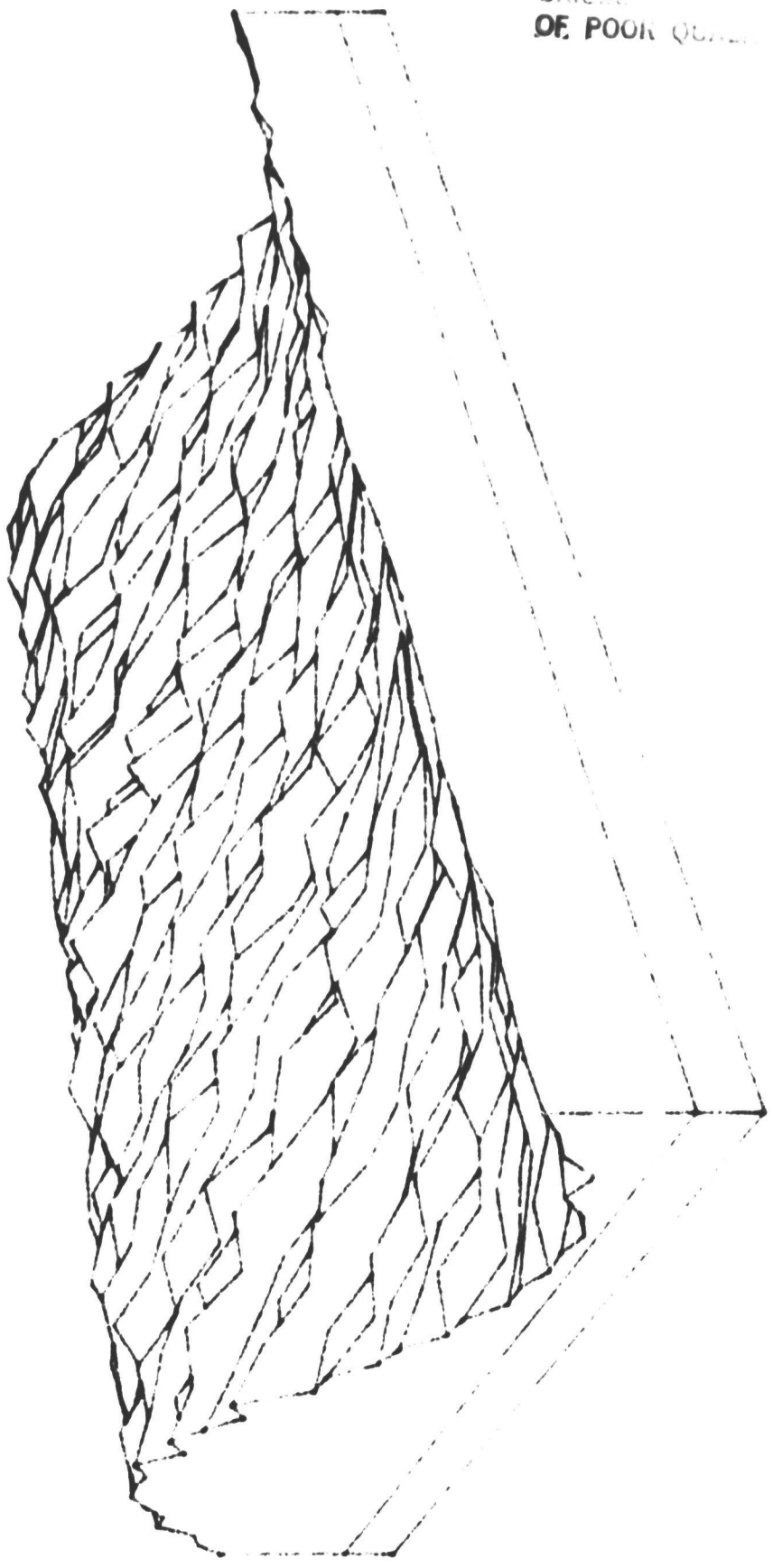

Fig. 1.36

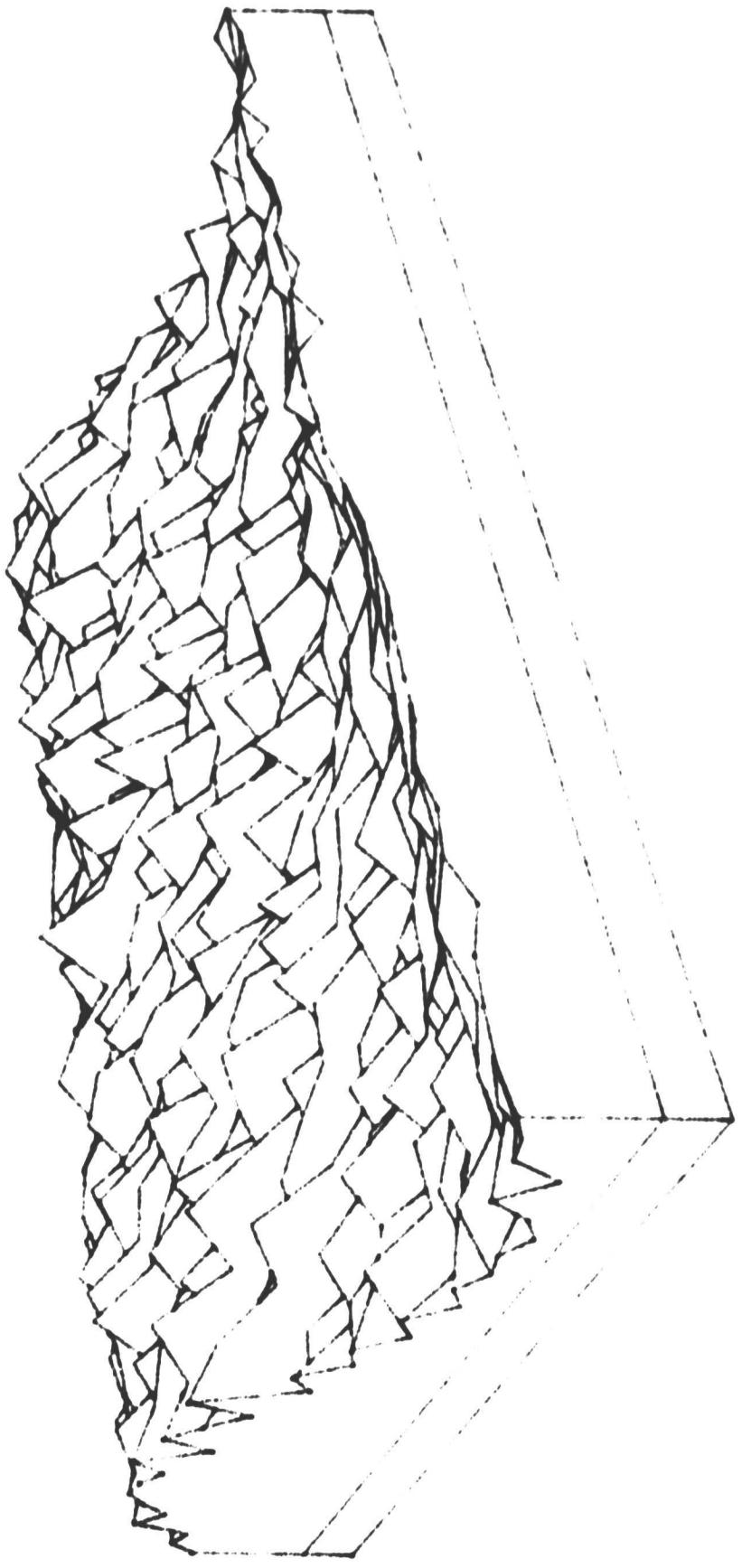

Fig. 1.37

Appendix

Following is the code for the programs discussed in this thesis. The order of the execution of these programs is: step one- input the data to GNOISE.FOR and choose the scale factor of the noise; step two- input the noisy data to DERIV4.FOR and choose the minus-i transform and the x, second xy, or y derivative; step three- filter the output of DERIV4.FOR using FILT.FOR with one of the three filter functions after choosing its size; step four- plus-i transform this result to obtain the x, second xy, or y derivative of the original data using DERIV4.FOR.

The data files used by DERIV4.FOR, GNOISE.FOR, FILT.FOR, are all unformatted binary random access files. The size of the two dimensional input and output arrays necessitated the use of such files. Disc I/O time was reduced drastically and thus execution time was greatly cut. The record length of these data files is equal to the number of data points.

DERIV4.FOR is the FORTRAN program that performs the transformation of the x-derivative of the data, second xy-derivative of the data, y-derivative of the data, or the transform of the data. The run-time parameters are:

1. The size of the matrices that hold the input and output data- the limit is 64 (for 64 X 64 matrices). Must input an integer that is an integral power of two.

2. The operation to be performed. Enter 1 for the transform of x-derivative of the input, 2 for the transform of the second xy-derivative of the input, 3 for the transform of the y-derivative of the input, 4 for the transform of the input.

3. The type of data- real or complex. Enter 1 for real data, a 2 for complex data.

4. The sign of the transform- minus-i or plus-i. Enter -1

for the minus-i transform, 1 for the plus-i transform.

5. The output file unit numbers. Enter the unit number for the real part data file first, then the unit number for the imaginary part data file.

6. The input file unit number(s). Enter the unit number for the real part. If the data is complex follow with the unit number for the imaginary part.

DERIV4.FOR

```
      DIMENSION A1(64,64),B1(64,64),DATA1(128)
      DIMENSION C1(64,64)
      TYPE 556
556   FORMAT('  ENTER SIZE OF MATRIX, LIMIT IS 64 ',$)
      ACCEPT 557,IZ1
557   FORMAT(I)
      TYPE 558
558   FORMAT('  ENTER D/DU, D2/DUDV, D/DV, OR F(X) ',$)
      ACCEPT 557,IZ2
      TYPE 559
55    FORMAT('  ENTER REAL(1) OR COMPLEX(2) DATA TYPE ',$)
      ACCEPT 557,IZ3
      TYPE 560
560   FORMAT('  ENTER SIGN OF TRANSFORM (1, -1) ',$)
      ACCEPT 557,IZ5
      IZ4=2*IZ1
563   FORMAT(2I)
      CALL DERIV(IZ1,IZ2,IZ3,IZ4,IZ5,A1,B1,DATA1,C1)
      STOP
      END
      SUBROUTINE DERIV(IN1,IK,IL1,IN3,ISN1,A,B,DATA,C1)
      DIMENSION A(IN1,IN1),DATA(IN3),B(IN1,IN1)
      DIMENSION C1(IN1,IN1)
      LOGICAL FLAG,FLAG2
48    FORMAT(2I)
      IN2=IN1/2
```

```
        IL2=IN1*IN1
        TYPE 562
562     FORMAT('  ENTER OUTPUT FILE # S(RE,IM) ',$)
        ACCEPT 48,IF1,IF2
        FLAG2=.TRUE.
        FLAG=.TRUE.
14      FORMAT (I)
        GO TO (220,230) IL1
220     TYPE 565
565     FORMAT('  ENTER FILE # ',$)
        ACCEPT 14,IFC
        CALL DEFINE FILE(IFC,IL2,LOC3,0,0,0)
666     READ (IFC#1) A
667     CALL REFRMT(IN1,IN2,A,C1)
        GO TO 333
230     TYPE 566
566     FORMAT('  ENTER FILE # S(RE,IM)',$)
        ACCEPT 48, IFA, IFB
        CALL DEFINE FILE (IFA,IL2,LOC1,0,0,0)
        CALL DEFINE FILE (IFB,IL2,LOC2,0,0,0)
        READ (IFA#1) A
        READ (IFB#1) B
        CALL REFRMT(IN1,IN2,A,C1)
        CALL REFRMT(IN1,IN2,B,C1)
333     DO 1 I=1,IN1
        DO 3 I1=1,IN3
3       DATA(I1)=0
```

```
11      IF (FLAG) GOTO 100
        DO 110 K=1,IN1,2
        A(I,K)=-A(I,K)
110     CONTINUE
        GOTO 111
100     DO 101 K=2,IN1,2
        A(I,K)=-A(I,K)
101     CONTINUE
111     FLAG=.NOT.FLAG
        DO 4 J=1,IN1
4       DATA(2*J-1)=A(I,J)
        IF (IL1.EQ.1) GO TO 240
        IF (FLAG2) GOTO 200
        DO 210 K=1,IN1,2
        B(I,K)=-B(I,K)
210     CONTINUE
        GOTO 211
200     DO 201 K=2,IN1,2
        B(I,K)=-B(I,K)
201     CONTINUE
211     FLAG2=.NOT.FLAG2
        DO 241 J=1,IN1
241     DATA(2*J)=B(I,J)
240     CALL MRKFFT(IN1,ISN1,IN3,DATA)
        DO 5 J=1,IN1
        A(I,J)=DATA(2*J-1)
5       B(I,J)=DATA(2*J)
```

```
1       CONTINUE
        DO 60 I2=1,IN1-1
        DO 60 J=I2+1,IN1
        TMP=A(I2,J)
        A(I2,J)=A(J,I2)
        A(J,I2)=TMP
60      CONTINUE
        DO 61 I3=1,IN1-1
        DO 61 J=I3+1,IN1
        TMP=B(I3,J)
        B(I3,J)=B(J,I3)
        B(J,I3)=TMP
61      CONTINUE
        DO 2 I=1,IN1
        DO 6 J=1,IN1
        DATA(2*J-1)=A(I,J)
6       DATA(2*J)=B(I,J)
        CALL MRKFFT(IN1,ISN1,IN3,DATA)
        DO 7 J=1,IN1
        A(I,J)=DATA(2*J-1)
7       B(I,J)=DATA(2*J)
12      CONTINUE
2       CONTINUE
        DO 70 I2=1,IN1-1
        DO 70 J=I2+1,IN1
        TMP=A(I2,J)
        A(I2,J)=A(J,I2)
```

```
      A(J,I2)=TMP
70    CONTINUE
      DO 71 I3=1,IN1-1
      DO 71 J=I3+1,IN1
      TMP=B(I3,J)
      B(I3,J)=B(J,I3)
      B(J,I3)=TMP
71    CONTINUE
      GO TO(49,39,59,777)IK
49    DO 41 I=2,IN2+1
      DO 41 J=IN2+1,IN1
      A(I,J)=A(I,J)*6.28318531*(J-(IN2+1))
41    B(I,J)=B(I,J)*6.28318531*(J-(IN2+1))
      DO 42 I=2,IN2+1
      DO 42 J=2,IN2
      A(I,J)=A(I,J)*6.28318531*(J-(IN2+1))
42    B(I,J)=B(I,J)*6.28318531*(J-(IN2+1))
      DO 43 I=IN2+2,IN1
      DO 43 J=2,IN2+1
      A(I,J)=A(I,J)*6.28318531*(J-(IN2+1))
43    B(I,J)=B(I,J)*6.28318531*(J-(IN2+1))
      DO 44 I=IN2+2,IN1
      DO 44 J=IN2+2,IN1
      A(I,J)=A(I,J)*6.28318531*(J-(IN2+1))
44    B(I,J)=B(I,J)*6.28318531*(J-(IN2+1))
      A(1,1)=A(1,1)*(-(IN2+1))*1.57079633
      B(1,1)=B(1,1)*(-(IN2+1))*1.57079633
```

```
      DO 68 J=2,IN1
      A(1,J)=A(1,J)*(J-(IN2+1))*3.14159265
68    B(1,J)=B(1,J)*(J-(IN2+1))*3.14159265
      DO 69 I=2,IN1
      A(I,1)=A(I,1)*3.14159265*(-(IN2+1))
69    B(I,1)=B(I,1)*3.14159265*(-(IN2+1))
      GO TO 777
39    DO 31 I=2,IN2+1
      DO 31 J=IN2+1,IN1
      A(I,J)=A(I,J)*6.28318531*
     ! (J-(IN2+1))*(ABS(I-(IN2+1)))
31    B(I,J)=B(I,J)*6.28318531*(ABS(I-(IN2+1)))
      DO 32 I=2,IN2+1
      DO 32 J=2,IN2
      A(I,J)=A(I,J)*6.28318531*
     ! (J-(IN2+1))*(ABS(I-(IN2+1)))
32    B(I,J)=B(I,J)*6.28318531
     ! *(J-(IN2+1))*(ABS(I-(IN2+1)))
      DO 33 I=IN2+2,IN1
      DO 33 J=2,IN2+1
      A(I,J)=A(I,J)*6.28318531*(J-(IN2+1))*((IN2+1)-I)
33    B(I,J)=B(I,J)*6.28318531*(J-(IN2+1))*((IN2+1)-I)
      DO 34 I=IN2+2,IN1
      DO 34 J=IN2+2,IN1
      A(I,J)=A(I,J)*6.28318531*(J-(IN2+1))*((IN2+1)-I)
34    B(I,J)=B(I,J)*6.28318531*(J-(IN2+1))*((IN2+1)-I)
      DO 67 J=2,IN1
```

```
      A(1,J)=A(1,J)*3.14159265*(IN2+1)*(J-(IN2+1))
67    B(1,J)=B(1,J)*3.14159265*(IN2+1)*(J-(IN2+1))
      DO 66 I=2,IN1
      A(I,1)=A(I,1)*3.14159265*(I-(IN2+1))*(IN2+1)
66    B(I,1)=B(I,1)*3.14159265*(I-(IN2+1))*(IN2+1)
      A(1,1)=A(1,1)*(IN2+1)*(-(IN2+1))*1.57079633
      B(1,1)=B(1,1)*(IN2+1)*(-(IN2+1))*1.57079633
      GO TO 777
59    DO 51 I=2,IN2+1
      DO 51 J=IN2+1,IN1
      A(I,J)=A(I,J)*6.28318531*(ABS(I-(IN2+1)))
51    B(I,J)=B(I,J)*6.28318531*(ABS(I-(IN2+1)))
      DO 52 I=2,IN2+1
      DO 52 J=2,IN2
      A(I,J)=A(I,J)*6.28318531*ABS(I-(IN2+1))
52    B(I,J)=B(I,J)*6.28318531*(ABS(I-(IN2+1)))
      DO 53 I=IN2+2,IN1
      DO 53 J=2,IN2+1
      A(I,J)=A(I,J)*6.28318531*((IN2+1)-I)
53    B(I,J)=B(I,J)*6.28318531*((IN2+1)-I)
      DO 54 I=IN2+2,IN1
      DO 54 J=IN2+2,IN1
      A(I,J)=A(I,J)*6.28318531*((IN2+1)-I)
54    B(I,J)=B(I,J)*6.28318531*((IN2+1)-I)
      A(1,1)=A(1,1)*(IN2+1)*1.57079633
      B(1,1)=B(1,1)*(IN2+1)*1.57079633
      DO 65 J=2,IN1
```

```
      A(1,J)=A(1,J)*3.14159265*((IN2+1))
65    B(1,J)=B(1,J)*3.14159265*(IN2+1)
      DO 64 I=2,IN1
      A(I,1)=A(I,1)*3.14159625*(I-(IN2+1))
64    B(I,1)=B(I,1)*3.14159625*(I-(IN2+1))
C     THE REAL PART OF THE TRANSFORM IS IN 21
C     THE IMAG. PART OF THE TRANSFORM IS IN 22
777   CALL DEFINE FILE(IF1,IL2,LOC4,0,0,0)
      CALL DEFINE FILE(IF2,IL2,LOC5,0,0,0)
      WRITE (IF2#1) A
      WRITE (IF1#1) B
700   RETURN
      END
      SUBROUTINE MRKFFT(NN,ISIGN,IQ1,DATA)
      DIMENSION DATA(IQ1)
      N=2*NN
C
C
C
C
C     FAST FOURIER TRANSFORM ROUTINE
C
C
      J=1
      DO 5 I=1,N,2
      IF(I-J)1,2,2
1     TEMPR=DATA(J)
```

```
      TEMPI=DATA(J+1)
      DATA(J)=DATA(I)
      DATA(J+1)=DATA(I+1)
      DATA(I)=TEMPR
      DATA(I+1)=TEMPI
2     M=N/2
3     IF(J-M)5,5,4
4     J=J-M
      M=M/2
      IF(M-2)5,3,3
5     J=J+M
      MMAX=2
6     IF(MMAX-N)7,10,10
7     ISTEP=2*MMAX
      THETA=6.2831853/FLOAT(ISIGN*MMAX)
      SINTH=SIN(THETA/2)
      WSTPR=-2*SINTH*SINTH
      WSTPI=SIN(THETA)
      WR=1
      WI=0
      DO 9 M=1,MMAX,2
      DO 8 I=M,N,ISTEP
      J=I+MMAX
      TEMPR=WR*DATA(J)-WI*DATA(J+1)
      TEMPI=WR*DATA(J+1)+WI*DATA(J)
      DATA(J)=DATA(I)-TEMPR
      DATA(J+1)=DATA(I+1)-TEMPI
```

```
        DATA(I)=DATA(I)+TEMPR
8       DATA(I+1)=DATA(I+1)+TEMPI
        TEMPR=WR
        WR=WR*WSTPR-WI*WSTPI+WR
9       WI=WI*WSTPR+TEMPR*WSTPI+WI
        MMAX=ISTEP
        GO TO 6
C
10      RETURN
        END
        SUBROUTINE REFRMT(IS,IS2,A,C)
        DIMENSION C(IS2,IS2),A(IS,IS)
39      FORMAT(I)
21      DO 60 I=1,IS2
        DO 60 J=1,IS2
60      C(I,J)=A(I,J)
        DO 61 I=IS2+1,IS
        DO 61 J=IS2+1,IS
        A(I-IS2,J-IS2)=A(I,J)
61      A(I,J)=C(I-IS2,J-IS2)
        DO 62 I=1,IS2
        DO 62 J=1,IS2
62      C(I,J)=A(I,J+IS2)
        DO 63 I=1,IS2
        DO 63 J=1,IS2
        A(I,J+IS2)=A(I+IS2,J)
63      A(I+IS2,J)=C(I,J)
```

```
50      FORMAT (8F)
        RETURN
        END
```

FILT.FOR is the FORTRAN filter program. The choice of filter functions is ideal square and circular low-pass filter functions, and a flat-topped filter function. The extent of the filter function can be varied by changing the value of M for the square and flat-topped filter function and R (radius) for the circular filter function. The run-time parameters are:

1. The size of the square matrices- same conditions hold as in DERIV4.FOR.

2. The input and output files. The data type is assumed to be complex, therefore a total of four logical unit numbers must be entered. The input unit numbers must be first, and for each pair of unit numbers the real unit number is first.

3. The filter function desired. Enter 1 for the square filter function, 2 for the circular filter function, 3 for the flat-topped pyramid filter function.

4. The size or extent of the filter function- limit is one-half the size of the matrices used by the program to hold data.

FILT.FOR

```
      DIMENSION A1(64,64),B1(64,64)
      TYPE 10
10    FORMAT('  ENTER SIZE OF MATRIX ',$)
      ACCEPT 11,ISZ1
11    FORMAT(I)
      ISZ2=ISZ1/2+1
      CAIL FILTR(A1,B1,ISZ1,ISZ2)
      STOP
      END
      SUBROUTINE FILTR(A,B,IN1,IORIG)
      DIMENSION A(IN1,IN1),B(IN1,IN1)
      IL2=IN1*IN1
401   FORMAT(I)
402   FORMAT(2I)
      TYPE 460
460   FORMAT('  ENTER INFILES,OUTFILES(RE,IM) ',$)
      ACCEPT 461,IF1,IF2,IF3,IF4
461   FORMAT(4I)
      CALL DEFINE FILE(IF1,IL2,LOC1,0,0,0)
      CALL DEFINE FILE(IF2,IL2,LOC2,0,0,0)
      CALL DEFINE FILE(IF3,IL2,LOC3,0,0,0)
      CALL DEFINE FILE(IF4,IL2,LOC4,0,0,0)
      READ(IF1#1) A
      READ(IF2#1) B
      TYPE 410
410   FORMAT(' IS FILTER SQ(1),CIRC(2),CUT PYRAM(3) ',$)
```

```
      ACCEPT 401,ISHAPE
      GO TO (420,430,450) ISHAPE
420   TYPE 421
421   FORMAT('  ENTER M(SIZE=2*M+1) ',$)
      ACCEPT 401,IMM2
      GO TO 440
430   TYPE 431
431   FORMAT('  ENTER RAD  ',$)
      ACCEPT 401,IRAD
440   IF (ISHAPE.EQ.1) IMID=IMM2
      IF (ISHAPE.EQ.2) IMID=IRAD
      DO 441 I=1,IORIG-IMID-1
      DO 441 J=1,IN1
      A(I,J)=0.0
      B(I,J)=0.0
441   CONTINUE
      DO 442 I=IORIG+IMID+1,IN1
      DO 442 J=1,IN1
      A(I,J)=0.0
      B(I,J)=0.0
442   CONTINUE
      DO 443 I=IORIG-IMID,IORIG+IMID
      DO 443 J=1,IORIG-IMID-1
      A(I,J)=0.0
      B(I,J)=0.0
443   CONTINUE
      DO 444 I=IORIG-IMID,IORIG+IMID
```

```
      DO 444 J=IORIG+IMID+1,IN1
      A(I,J)=0.0
      B(I,J)=0.0
444   CONTINUE
      IF (ISHAPE.EQ.1) GO TO 490
      K1=IN1/2+IMID+2
      J3=(IN1/2-IMID)*2
      K2=IN1/2-IMID+1
      J4=J3+4*IMID+2
      K3=K1-1
      J5=J3+1
      J6=J4/2
      RIGN=IORIG
      DO 140 I2=IORIG-IMID,IORIG+IMID
      DO 140 I3=IORIG-IMID,IORIG+IMID
      L1=I2-IN1/2-1
      L2=(I3+1)/2-IN1/2-1
      XR=(I2-RIGN)**2
      YR=(I3-RIGN)**2
      R=IMID
      IF (SQRT(XR+YR).LE.R) GOTO 140
      A(I2,I3)=0.0
      B(I2,I3)=0.0
140   CONTINUE
      GO TO 490
C     CIRCLE FILTER IS FINISHED
C
```

```
450     TYPE 451
451     FORMAT('  ENTER MID(PLATEAU=2*MID+1) ',$)
        ACCEPT 401,IMID
        IRMP=(IORIG-1)-(IMID+1)
        XDEL=1.0
        DO 22 I=3,IORIG-IMID-1
        DO 23 J=I,IN1-(I-2)
        A(I,J)=(XDEL/IRMP)*A(I,J)
        B(I,J)=(XDEL/IRMP)*B(I,J)
23      CONTINUE
        XDEL=XDEL+1.0
22      CONTINUE
        XDEL=1.0
        K=0
        DO 24 I=IN1-1,IORIG+IMID+1,-1
        DO 25 J=3+K,IN1-1-K
        A(I,J)=(XDEL/IRMP)*A(I,J)
        B(I,J)=(XDEL/IRMP)*B(I,J)
25      CONTINUE
        XDEL=XDEL+1.0
        K=K+1
24      CONTINUE
        XDEL=1.0
        DO 26 J=3,IORIG-IMID-1
        DO 27 I=J+1,IN1-J+1
        A(I,J)=(XDEL/IRMP)*A(I,J)
        B(I,J)=(XDEL/IRMP)*B(I,J)
```

```
27      CONTINUE
        XDEL=XDEL+1.0
26      CONTINUE
        XDEL=1.0
        K=0
        DO 28 J=IN1-1,IORIG+IMID+1,-1
        DO 29 I=4+K,IN1-2-K
        A(I,J)=(XDEL/IRMP)*A(I,J)
        B(I,J)=(XDEL/IRMP)*B(I,J)
29      CONTINUE
        K=K+1
        XDEL=XDEL+1.0
28      CONTINUE
        DO 30 J=1,IN1
        A(IN1,J)=0.0
        B(IN1,J)=0.0
30      CONTINUE
        DO 31 I=1,IN1
        A(I,IN1)=0.0
        B(I,IN1)=0.0
31      CONTINUE
        DO 20 I=1,2
        DO 20 J=1,IN1
        A(I,J)=0.0
        B(I,J)=0.0
20      CONTINUE
        DO 21 J=1,2
```

(-2

```
      DO 21 I=1,IN1
      A(I,J)=0.0
      B(I,J)=0.0
21    CONTINUE
490   WRITE(IF3#1) A
      WRITE(IF4#1) B
500   RETURN
      END
```

GNOISE.FOR is the FORTRAN program that adds Gaussian ordinant dependent noise to the input data. The input data is not destroyed. The run-time parameters are:

1. The size of the matrices- the same conditions hold as for DERIV4.FOR.

2. The scale factor of the noise.

3. The type of data- real or complex. Enter 1 for real, 2 for complex.

4. The input logical unit number(s), the unit number(s) that is (are) to contain signal plus noise, the unit number(s) that is (are) to contain the noise only. If the data is complex two unit numbers are required for each case, with the real unit number being the first in all cases.

GNOISE.FOR

```
      DIMENSION A(64,64),B(64,64)
      REAL SF
      TYPE 10
10    FORMAT('  ENTER SIZE OF MATRIX ',$)
      ACCEPT 11,ISZ
11    FORMAT(I)
      TYPE 12
12    FORMAT('  ENTER SCALE FACTOR ',$)
      ACCEPT 13,SF
13    FORMAT(G)
      CALL NOIS(A,B,SF,ISZ)
      STOP
      END
      SUBROUTINE NOIS(A1,B1,SF1,ISZ1)
      DIMENSION A1(ISZ1,ISZ1),B1(ISZ1,ISZ1)
      REAL SF1
      IL2=ISZ1*ISZ1
      KOUNT=0
      TYPE 100
100   FORMAT('  REAL(1) OR COMPLEX(2) DATA TYPE ',$)
      ACCEPT 110,IK1
111   FORMAT(2I)
110   FORMAT(I)
112   FORMAT(2G)
113   FORMAT(G)
114   FORMAT(3I)
```

```
115     FORMAT(6I)
        GO TO(120,130) IK1
120     TYPE 121
121     FORMAT('  ENTER INFILE,S+N,N FILE ',$)
        ACCEPT 114,IF1,IFS1,IFN1
        CALL DEFINE FILE(IF1,IL2,LOC1,0,0,0)
        CALL DEFINE FILE(IFS1,IL2,LOC2,0,0,0)
        CALL DEFINE FILE(IFN1,IL2,LOC3,0,0,0)
        READ(IF1#1) A1
        GO TO 200
130     TYPE 131
131     FORMAT('  ENTER INPUT RE,IM,(S+N)
     ! RE,IM,(N) RE,IM',$)
        ACCEPT 115,IF1,IF2,IFS1,IFS2,IFN1,IFN2
        CALL DEFINE FILE(IF1,IL2,LOC1,0,0,0)
        CALL DEFINE FILE(IF2,IL2,LOC2,0,0,0)
        CALL DEFINE FILE(IFS1,IL2,LOC3,0,0,0)
        CALL DEFINE FILE(IFS2,IL2,LOC4,0,0,0)
        CALL DEFINE FILE(IFN1,IL2,LOC5,0,0,0)
        CALL DEFINE FILE(IFN2,IL2,LOC6,0,0,0)
        READ (IF1#1) A1
200     RMS=0.0
        KOUNT=KOUNT+1
        DO 300  I=1,ISZ1
        DO 300 J=1,ISZ1
        K=I
        P=RAN(5)
```

```
      S=RAN(10)
      IF (A1(I,J).LT.1.0E-20) XN=2.*P*A1(I,J)
      IF (A1(I,J).LT.1.0E-20) GO TO 250
      XN=ABS(A1(I,J))*
     ! (-ALOG(P*SQRT(6.28318*SF1*ABS(A1(I,J)))))
      XN=SQRT(2.0*SF1*XN)
250   IF (S.GT.0.5) XN=-XN
      TEMP=A1(I,J)
      A1(I,J)=A1(I,J)+XN
      B1(I,J)=XN
      IF (A1(I,J).LT.0.0) A1(I,J)=0.0
      RMS=(A1(I,J)-TEMP)**2+RMS
300   CONTINUE
      RMS=SQRT(RMS/IL2)
      TYPE 113,RMS
      IF (KOUNT .EQ. 2) GOTO 500
      WRITE (IFS1#1) A1
      WRITE (IFN1#1) B1
      IF (IK1 .EQ. 1) GOTO 400
      IF (KOUNT .EQ. 1) READ (IF2#1) A1
      IF (KOUNT .EQ. 1) GOTO 200
500   WRITE (IFS2#1) A1
      WRITE (IFN2#1) B1
400   RETURN
      END
```

Bibliography

Andrews, H.C. (1970), Computer Techniques in Image Processing, Academic Press, New York, New York.

Bivens, William C. (1976), "Resolution Enhancement for Non-fixed Linear Systems," (Unpublished Master's Thesis, Department of Physics, University of New Orleans).

Bracewell, Ron(1965), The Fourier Transform and its Applications, McGraw Hill, New York, New York.

Cochran, William T., et al (1967), "What is the Fast Fourier Transform?," Proceedings of the IEEE, vol. 55, no. 10, pp. 1664-1674 (October).

Higgins, R.J. (1976), "Fast Fourier Transform: An introduction with some minicomputer experiments," American Journal of Physics, vol. 44, no. 8, pp. 766,773 (August).

Ioup, George E, private communications (1978).

VITA

Kathleen Simons Acomb Whitehorn was born in [redacted] on [redacted] to Bettie Ann [redacted] and Cyril Lloyd Acomb. She graduated from Walters Preparatory High School in May 1974 and the following fall entered the University of New Orleans. On August 6, 1977 she married Mark Alan Whitehorn, also a graduate of the University of New Orleans in physics. She received her Bachelor of Science degree in physics with honors in December of 1977 from the University of New Orleans. The following spring she accepted a graduate assistantship from the Physics Department of the University of New Orleans. Her last two semesters she worked with the Louisiana State University Eye Center.