A STRING MODEL ETCHING ALGORITHM

by

Robert Jewett

Memorandum No. UCB/ERL M79/68

SAMPLE Report No. SAMD-3

October 18, 1979

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **18 OCT 1979**	2. REPORT TYPE	3. DATES COVERED **00-00-1979 to 00-00-1979**
4. TITLE AND SUBTITLE **A String Model Etching Algorithm**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Berkeley,Department of Electrical Engineering and Computer Sciences,Berkeley,CA,94720**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The basic algorithm and software implementation of a string model for simulation of surface etching are presented. The algorithm models the time evolution of a line edge profile by advancing nodes or points on a piecewise linear curve representing the profile. The specific formulas for the direction and rate of advance, insertion and deletion of points and deletion of loops are shown. Complete software documentation in the form of parameter definitions and a listing of the FORTRAN code for a CDC 6400 machine are included.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**28**	

A STRING MODEL ETCHING ALGORITHM

by

Robert Jewett

Memorandum No. UCB/ERL M79/68

October 18, 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A String Model Etching Algorithm

Robert Jewett

Electronics Research Laboratory
University of California
Berkeley, California 94720

ABSTRACT

The basic algorithm and software implementation of a string model for simulation of surface etching are presented. The algorithm models the time evolution of a line edge profile by advancing nodes or points on a piecewise linear curve representing the profile. The specific formulas for the direction and rate of advance, insertion and deletion of points and deletion of loops are shown. Complete software documentation in the form of parameter definitions and a listing of the FORTRAN code for a CDC 6400 machine are included.

This research was in collaboration with research sponsored by Joint Services Electronics Program, contract F44620-71-C-0087, Air Force Office of Scientific Research, grant 71-2113 and the U.S. Army Research Office grant DA-ARO-D-31-124-73-G165.

The string development model described here has been incorporated in program SAMPLE (Simulation And Modeling of Profiles in Lithography and Etching) but with several modifications.

A String Model Etching Algorithm

Robert Jewett

Electronics Research Laboratory
University of California
Berkeley, California 94720

Introduction

As the technology of microfabrication is pushed to its limits, it becomes increasingly important to be able to model each step of the fabrication processes, both to predict the ultimate performance of the technology and to understand present techniques. Etching is a step which appears universally in microfabrication. It may occur as the dissolution of a photoresist by an organic solvent, the etching of an oxide by an alkali, or the plasma etching of an electron resist. Whatever its physical details, the etching process can in many cases be modeled as a surface etching phenomenon. The resulting profile is determined by an initial profile which moves through a medium in which the speed of propagation may be a function of position and other variables. Several examples are given in appendix B and [1].

Two major assumptions limit the generality of the present algorithm. Firstly, the pattern to be etched is uniform in one dimension, so the problem can be solved using only two dimensions. For most microfabrication problems, the important cases involve the cross-sections of lines, so the model is directly applicable. In certain other cases, such as round holes, the symmetry of some cross-sections is such that the algorithm can still be applied.

A second major assumption is that the etch rate is a scalar function of position, and is independent of the local direction of etch front motion and the history of the front. In some real situations this does not hold. PMMA for instance has been found to have a gel region at the resist-solvent interface during development, so etch rate is a function of the history of the adjacent regions as well as the exposure. This gel is somewhat swollen compared to the dry resist, so that during certain periods of the development, the surface may actually advance into the solvent. Another case where the second assumption does not hold is in so-called "preferential etching" where etching proceeds more quickly along certain crystal directions, making the etch anisotropic. A third case that is allowed by the algorithm is simultaneous exposure and development, so that etch rate may be a function of time as well as position. Such a case would require the implementation by the user of a special rate algorithm.

Two other algorithms have been reported previously, the cell model of Dill et al.[2] and the ray model of Hagouel[3]. The cell model divides the region under study into rectangles. Each rectangle can be thought of as an ice cube in a tray of ice cubes that is melting. Only the cubes on the exposed surface melt, and the time for a cube to melt is determined by the local rate and the number of sides from which it is exposed. This model is inherently discrete, although continuous contours can be interpolated from cell removal times. It is slightly inaccurate for certain directions in that circular profiles tend to converge to octagons. The only implementations to date appear to require slightly more computation time than the other alternatives.

The ray model works by analogy to Snell's law of optics. The etch rate corresponds to the inverse of the index of refraction. The etch front is inferred by following the paths of "rays" being "refracted" by the nonuniform etch rate. The ray algorithm is suitable for implementation on a desk calculator and for extension to three dimensions. Some user interaction is necessary, especially when the gradient of the rate is not defined everywhere.

The algorithm described in the remainder of this report is known as the "string model". The etch front is simulated by a series of points joined by straight line segments, forming a

string. During each time increment, each point is advanced perpendicularly to the local etch front, as in Fig. 1. A major portion of the algorithm adjusts the number of segments to keep them approximately equal in length. Other subroutines input the data and output the etch front. The algorithm was implemented in FORTRAN and many details were determined by the available syntax, but the following considerations are largely language independent.

Segment length criteria

A major problem in developing the algorithm was in choosing suitable criteria for segment length. Originally it was believed that the segments should be short enough so that any curve that occured would be well defined, i.e., there should be some maximum angle between adjacent segments, perhaps 0.1 radian. It was soon found that this criterion led to a great proliferation of segments in regions where the front was either expanding or contracting. Often adjacent segments would differ in length by several orders of magnitude, which led to apparently insurmountable problems. Eventually it was decided to make segment length uniformity the sole length criterion. While this led to unavoidable position errors of about one-half segment length, the resulting algorithm is much simpler and faster. If necessary, the error can be reduced by decreasing the average segment length with a proportional increase in computation cost.

Time step size

For most cases of interest, the etch rate varies with position. This leads to errors in the position and in the direction of each point on the string. Errors in position arise from the rather simple integration algorithm used. The local rate at the start of each time step is assumed to be constant throughout the step. This can easily lead to position errors as large as the distance covered in one step. Consider for example an etch front in a photoresist approaching an unetchable substrate. A point which is barely outside the substrate at the start of the time step will advance into the substrate at the rate associated with the resist. An example which could spawn huge errors in position is thin layers of alternating fast and slow etch rates. With too large a time step, a point could jump over a slow region.

Errors in direction arise from nonuniform rates along the string, and from certain boundary conditions. During each step, the perpendicular to the front, which is defined below, is assumed to be constant in direction. If two adjacent points have greatly differing rates, the quickly moving point cannot start turning towards the slower point until the end of the time step. This mechanism tends to introduce relatively small errors in position because the error is roughly proportional to the cosine of the angle error.

An example of boundary conditions causing this kind of error is shown in Fig. 2. During the first time step, point 1 moves to the left, giving undercutting. Point 2 moves straight down. The exact solution of the problem is shown by the broken line, which is simply a circular arc joining smoothly with a straight line parallel to the original surface. If the time step were subdivided into smaller steps, the path of point 2 would have bent slightly towards point 1, but the large error in approximating the circular arc would not have been significantly reduced.

Defining the normal to the string

In the example above, it is easy to see what the normal to the string is for point 3, and therefore what its direction of movement should be. It is not so easy to see what direction point 2′ should take. Three alternatives were considered for the algorithm. First, the point could travel perpendicularly to the line segment to its left or right. This approximation is easy to calculate, and can be applied to every point on the string, but it is inherently asymmetrical since the left or right neighbor must be chosen. A second method is to advance the point normal to a line joining the two neighboring points. In the above example, point 2′ would be moved normal to a line joining 1′ and 3′. This is as easy to calculate as the first method, but the directions of the endpoints of the string must be determined separately. This is not a major drawback since the directions of the endpoints are usually set by boundary conditions or symmetry. The third method, which was finally chosen, is to advance the point along the angle

bisector of the two adjacent segments. This requires slightly more computation than the second method, but gives better results when the segments differ in length, especially in regions where the string is contracting.

Endpoint directions

In the present algorithm, the directions of the two endpoints are constants specified by input data. Usually the boundary conditions or symmetry of the problem will determine the directions. In the above example, the endpoints are specified as moving parallel to the resist-oxide interface, which leads to undercutting of the resist. The computation for the example could be roughly halved by noticing that the problem is symmetrical about the vertical centerline. Point 3 therefore always travels along the centerline. The problem can be solved by calculating for the string 1-2-3, with point 3 specified as moving straight down, and then reflecting the results through the centerline to obtain the complete solution. Use of symmetry like this can cause problems when the centerline is not a region of maximum rate. Neureuther and O'Toole[4] have encountered such cases in optical-exposure simulations.

Addition of segments

In the example above, the segment between points 1 and 2 will grow indefinitely due to the nonparallel motion of its endpoints. As it grows, it becomes an increasingly poor approximation to the exact solution, which is just a quarter circle joined to a horizontal segment. Similar cases occur in nearly all simulations since there is usually at least one region in which the front is expanding. One way to solve this problem is to start with very closely spaced points so that sufficient resolution is maintained even with considerable expansion. This becomes expensive for problems with expansion ratios of more than 10:1, as in optical-exposure standing-wave problems. A more economical approach is to break each excessively long segment into two shorter segments. The simplest way to do this is to add the midpoint of the long segment to the string, as shown in Fig. 3a. This placement of the new point does not approximate very well the curved surface of the region of expansion. In the present algorithm the local curvature is estimated by calculating the angles formed by the long segment with the two adjacent segments. The larger these angles are, the more the new point should be offset into the region of expansion, as shown in Fig. 3b.

A reasonable function for the length s can be derived by considering a regular polygon expanding outward, approximating a circle. In that case $\theta=\phi$, and $s=\frac{1}{2}L\tan(\theta/4)$. This is approximated in the algorithm by $s=L\theta/8$, where θ is chosen to be the smaller of the two adjacent angles. This underestimates s for all cases, so the adjustment never overcompensates for curvature. A segment is divided like this whenever its length exceeds some maximum value, such as twice the length of the segments in the original string.

A faster algorithm due to S. N. Nandgaonkar[3] offsets the added point from the midpoint of the long segment by a fraction of the vector difference of the two adjacent segments.

Regions of contraction

Most simulations also have regions of contraction in which the segments become shorter. Consider the development of the idealized exposed line in Fig. 4. The etch rate in the unexposed region is unity, and is three in the exposed region. Contraction occurs at the points A-A'-A'', where the two straight fronts are colliding. The details of the collision are shown in Fig. 4b.

Ideally, the algorithm should move the point at he corner to follow the locus of the intersection of the two straight fronts as they advance. This theoretical point of intersection actually moves faster than the two fronts by a factor of $1/\sin(\theta/2)$, where θ is the angle included between the two fronts, just as the cutting point of a pair of scissors moves faster than the blades. This increase in velocity can only occur for two intersecting fronts, so any algorithm to correct the local rate would have to check three consecutive angles to determine whether the middle vertex was in fact such an intersection.

Rather than perform this additional checking, the corner is allowed to lag behind, as shown by the third string in Fig. 4b. Segments adjacent to the corner decrease in length, and are deleted when they are less than a minimum value, such as 75% of the initial segment length. For some obtuse angles of intersection, this is a satisfactory solution. In some simulations a much sharper angle occurs as in Fig. 5. Segment AB may not contract to less than the minimum allowed length prior to point A crossing the string between B and D, resulting in the situation of Fig. 5b. This process is refered to as "loop budding". Point B will subsequently move to the left rather than the right because the sense of the angle has changed. The loop will start expanding eventually requiring the addition of points such as B′ and B″. Loop formation is beneficial in that the intersection point X can move with increased velocity because it is not actually a calculated point. If the loop grows too large, the additional computation for its segments becomes burdensome.

Loop deletion

Prior to the output of the front, loops are deleted, except for very small loops. This is done by checking each segment against following segments for intersection in two steps. First the coordinates of the left endpoints of the two segments are compared in the x and y directions. If the difference in either dimension is larger than twice the maximum allowed segment length, the pair cannot intersect. Otherwise, the pair may intersect, and a more careful check is performed by the following algorithm.

Suppose complex numbers z_1 and z_2 give the endpoints of segment A, and z_3 and z_4 define segment B. Let

$$Z_A(a)=(1-a)z_1+az_2$$

be any point along the extension of A, determined by a real parameter a. Similarly

$$Z_B(b)=(1-b)z_3+bz_4$$

gives any point along the extension of B. If A and B are nonparallel, $Z_A(a)=Z_B(b)$ has a solution, and solving for a and b gives:

$$a = \frac{\mathrm{Im}((z_3-z_1)(z_4-z_3)^*)}{\mathrm{Im}((z_2-z_1)(z_4-z_3)^*)}$$

$$b = \frac{\mathrm{Im}((z_1-z_3)(z_2-z_1)^*)}{\mathrm{Im}((z_4-z_3)(z_2-z_1)^*)}$$

where * denotes the complex conjugate. If a and b both have values between 0 and 1, the segments A and B intersect at $z_5=(1-a)z_1+az_2=(1-b)z_3+bz_4$, see Fig. 6. When intersection is found to occur, the loop is deleted and the point of intersection is added to the string.

Details of the FORTRAN implementation

The above aspects of the algorithm are fairly independent of the programming language. The following details pertain to the implementation of the algorithm in FORTRAN by the RUN compiler on a CDC 6400 computer[5]. Much of the algorithm is based on the use of complex arithmetic to easily manipulate magnitudes, directions and angles. The x coordinate corresponds to the real part of the complex number, and the y coordinate to the imaginary part. It is important to note that y increases downward, and usually represents depth below the initial surface.

The following sections define each variable, and give a description of each routine in the algorithm. A listing of the program is in appendix A.

Variables in blank COMMON

XY one-dimensional complex array of 200 elements, stores the current coordinates of the etch front

*LABEL one-dimensional integer array of 20 elements, stores the title of the simulation. At 10 characters per word, there are 200 characters in the title.

*XMAX maximum x-dimension, normally used only in plotting programs when no x-dimension is specified by the user. Could be used in RATE to specify RATE = 0 when X > XMAX.

*YMAX maximum y-dimension, similar to XMAX

*NPTS The number of points in the string, not greater than 200. If more points are required, the following must be changed:

All blank common statements (XY)

The line following line 6 in CHKR

Dimensions of XT and YT in PLTOUT

READIN limits the initial value of NPTS to less than 101 and greater than 4.

*CXYL complex variable giving the direction of the left endpoint of the etch front, the cardinal directions could be given as:

left (-1.,0.)

right (1.,0.)

up (0.,-1.)

down (0.,1.)

*CXYR corresponds to CXYL, for the right endpoint

*NBCHK number of times the etch front is stepped forward between checks of segment length by CHKR

*NBOUT number of checks between each output by PLTOUT. The total number of steps between each output is NBCHK times NBOUT.

*TOUT time between outputs

TSTEP time of each minor step, usually TOUT/NBCHK/NBOUT

NSTEP normally equals NBCHK, the number of steps taken by CYCLE. If the user implements a subroutine that changes the value of TSTEP, NSTEP will be adjusted in CYCLE to correspond to the changed time step. No such subroutine currently exists.

ET elapsed time in CYCLE, equals TOUT/NBOUT

TTOT total time of simulation

IFLAG integer variable, equal to 0 if no segment length exceeds the bounds of SMAX and SMIN, and equal to 1 if one does. Value is determined in CYCLE.

SMAX largest allowed segment length

SMIN smallest allowed segment length

*NOUT number of output contours, less than 21

LRT flag to request RTEST, equal to 2HRT for a test, ignored otherwise

---

* denotes user specified variables.

Program ETCH

The main program ETCH calls the various subroutines and cycles through the etching simulation. It initially calls:

READIN to input data cards

LINEAR to initialize the string between the user-specified endpoints

PLTBGN to output the title, XMAX, and YMAX to the output file, TAPE21

RATEF to initialize the function RATE

The program then calculates the upper and lower bounds on segment length and initializes the time parameters. If the user has asked for a test of the rate function, RTEST is called.

The do-loops to lines 4 and 5 advance the etch front with CYCLE, correct segment lengths when necessary with CHKR, remove loops with DELOOP, and output the etch front to TAPE21 with PLTOUT.

Variables in ETCH

DUMMY dummy variable in call to RATEF

M main do-loop counter

N minor do-loop counter

SMID beginning segment length

X dummy input variable to RATEF

Subroutine READIN

This subroutine reads in data cards, checks for appropriate values, and prints the input data. The cards are first read as alphanumerics into the array LINES, so they can be printed out exactly as punched. The input file is then rewound and the variables are read in by appropriate formats:

8A10 for LABEL

F15.7 for floating point numbers

I3 for integers

The data are reprinted as interpreted by the various formats. This allows the user to correct mispunched data without reference to the format statements in the program.

The data are partially checked for reasonable values, and KRASH is called for out-of-range inputs. The allowed ranges are:

NPTS from 5 to 100

NOUT from 1 to 20

NBCHK from 1 to 10

NBOUT from 1 to 50

The directions for the left and right endpoints (CXYL and CXYR), are normalized to unit magnitude.

The first 18 data cards are mandatory. If the user supplies fewer cards, the subroutine prints out the cards as read into LINES and stops. LRT is read on the 19th data card. This card should have "RT" punched in the first two columns if a rate test by RTEST is desired. This is strongly recommended for any new rate function. If there is no 19th data card, LRT is set equal to "NO".

Variables in READIN

LINES an 18x8 array used to store the characters on the first 18 data cards. The total number of characters is 18x80, at 10 characters per word.

Subroutine RTEST

This subroutine checks the rate function in the region of interest. The area between x = 0 and x = XMAX, and y = 0 and y = YMAX is divided into a 30x30 grid. The rate at each grid point is read into the array RTS, and the maximum and minimum rates are kept in RMAX and RMIN. RTS is then normalized so that the maximum value is .99. This allows a compact printing format for the array of rates. The subroutine also looks for the maximum and minimum rates along the starting string. The printed output includes:

1. The minimum and maximum rates observed in the 30x30 grid
2. The minimum and maximum rates along the starting string
3. The "worst case error coefficient", which is found from:

   Q = RMAX*TSTEP/SMIN

   This gives some indication of the accuracy of the simulation.
4. The bounds and dimensions of the test grid
5. The values of the rates in the test grid, normalized to a specified value.

The subroutine also prints out the CPU time required by itself. This gives the user an indication of how quickly RATE is operating.

Variables in RTEST

DT	the smallest time increment in the simulation
DX	the x-distance between grid points
DY	the y-distance between grid points
N	do-loop counter
NX	do-loop counter for the x-direction
NY	do-loop counter for the y-direction
Q	the "worst case error coefficient"
RL	temporary value of the rate along the starting string
RLMAX	maximum rate along the string
RLMIN	minimum rate along the string
RMAX	maximum rate in the test grid
RMIN	minimum rate in the test grid
RNORM	a value 1% greater than RMAX
RTS	a 30x30 real array of rates
RXY	a temporary value of the rate at the current grid point
TIME	The CPU time in seconds spent in subroutine RTEST
TIME1	starting time
TIME2	ending time
X	the real part of the location of the grid point
Y	the imaginary part of the location of the grid point

Subroutine CHKR

This subroutine adds and deletes segments when CYCLE has detected a segment length error. At the start of the subroutine, the current simulated time, the number of points in the string, and the total CPU time are printed out. This allows the user to follow the course of the simulation without a plotted output, and to estimate the calculation time required for more or less resolution and accuracy.

The subroutine next checks to see if the rates at the two left-most points are both zero. If they are, the left endpoint is deleted. This saves computation time when the left end reaches a region of zero rate (i.e. the substrate). To use this feature, the user should arrange to start the left end of the string in a region of maximum rate.

The subroutine then checks each segment length against the limits of SMAX and SMIN. If a segment is out of bounds, its index is stored and the number of changes required is incremented. After all the errors have been located, the subroutine deletes short segments (line 7 to line 10) and splits long segments (line 6 to line 7). If an attempt is made to create more than 200 points, KRASH is called.

Variables in CHKR

ANG an angle used when adding a point near the midpoint of a long segment. It is one-eighth of the smaller convex angle formed by the long segment and its neighboring segments.

ANG1 The angle between the long segment and its left neighbor

ANG2 the angle between the long segment and its right neighbor

C the magnitude of the current segment length

CI the square root of -1, used for 90 degree rotation.

I do-loop counter

INDEX array for indices of segments having length errors

J temporary integer variable

K do-loop counter

M index of current segment to be modified, calculated from the previously found index and the number of segments already added or deleted.

MLAST The last value of M. Checking MLAST prevents the deletion of two adjacent segments on the same pass through CHKR.

N do-loop counter

NADD number of segments added so far, may be negative

NCHNGEnumber of changes needed

NSTOP used to stop one place short of NPTS on some do-loops

T length of the current incorrect segment

TCPU CPU total elapsed time in seconds

Subroutine DELOOP

This subroutine finds and deletes the loops that form in regions of contraction of the string. Because it is a rather slow routine, it is normally called just before each output.

Each segment is checked for intersection with all following segments with a few exceptions. No segment is checked against the two segments following it. While this allows small loops to be output, it saves a considerable amount of time by eliminating needless exact checks. Also, the last four segments are not checked against each other.

A preliminary check is made on each pair of segments to be tested. This is done by looking at the magnitudes of the real and imaginary separations of the left endpoints of the two

segments. If either of these separations is larger than twice the maximum allowed segment length (SMAX), it is impossible for the segments to intersect, and the subroutine goes on to the next pair of segments.

If the left endpoints are closer than the test limit, an exact check is done. This requires the calculation of the point of intersection of the segments if they were extended, and a check to see if that point lies interior to both segments.

If the two segments are found to intersect by the exact check, all the segments on the loop are deleted, and the point of intersection becomes the joint between the two parts of the string. Since the intersection may be near one end of an intersecting segment, one of the resulting segments may be shorter than the minimum allowed segment length. This is not adjusted prior to outputting the string, and will normally be adjusted during the next call to CYCLE.

When a loop is deleted, the subroutine prints the number of points before and after the deletion and the total simulated time.

Variables in DELOOP

AA a parameter that tells how far along the first segment the point of intersection is. If it is between 0 and 1, the intersection occurs within the segment.

ADENOMa temporary variable used in calculating AA

BB similar to AA, for the second segment

BDENOMsimilar to ADENOM

J do-loop counter for segment deletion

JSTART starting value for J

JSTOP stoping value for J

M the index of the left endpoint of the second segment

N the index of the left endpoint of the first segment

NOLD the former value of NPTS, prior to the deletion

S twice SMAX, used for the preliminary check

XN the real part of the the left endpoint of the first segment

YN the imaginary part of the left endpoint of the first segment

Subroutine LINEAR

This subroutine establishes an evenly spaced string of points between specified endpoints. The user supplies the values of NPTS, XY(1), and XY(NPTS) through subroutine READIN.

Variables in LINEAR

N do-loop counter

NSTOP upper limit on N, equal to NPTS-1

XYSTEP complex variable equal to the increment from one point to the next

XYSTRT complex variable giving the starting location of the string, actually one step prior to the start

Subroutine CYCLE

This subroutine advances the string perpendicular to itself according to the local etch rates and the specified time increment. The minimum and maximum segment lengths are retained, so that CHKR can be called to correct any length error.

The subroutine first initializes variables to be used in the do-loop. The number of steps, NSTEP, is calculated from the total time to be simulated, ET, and the time step increment for

each step, TSTEP. CSTEP is a complex variable incorporating both the magnitude of TSTEP and a 90 degree rotation.

For each TSTEP, each point is advanced along the angle bisector of the adjacent segments a distance proportional to the local etch rate. The endpoints are advanced in the directions specified by the user.

At the end of the subroutine, the minimum and maximum segment lengths are checked, and IFLAG is set if an error has occured.

Variables in CYCLE

CI	square root of -1.
CSTEP	complex variable equal to CI*TSTEP
DL	unit vector along segment to the left
DR	unit vector along the segment to the right
DT	DL+DR, giving a direction 90 degrees from the angle bisector
M	do-loop counter for the index of XY
N	do-loop counter for NSTEP
NSTOP	upper limit on M, equal to NPTS-1
T	magnitude of the current segment length
TMAX	observed maximum of T
TMIN	observed minimum of T

Subroutine KRASH

This subroutine prints an error message and stops execution. The input variable is expected to be a Hollerith constant of ten characters, which states the cause of termination.

Variable in KRASH

WORD	integer variable expected to be the error message

Subroutine PLTOUT

This subroutine writes the output on TAPE21. The first call to the subroutine is through entry PLTBGN, which writes LABEL, XMAX, and YMAX on TAPE21 by a binary format. Subsequent calls are through the normal entry, and cause the number of points, NPTS, the x coordinates and the y coordinates to be written on TAPE21 in a binary format.

Variables in PLTOUT

N	do-loop counter
XT	temporary array for the x coordinates
YT	temporary array for the y coordinates

Function RATE(Z)

The RATE function will usually be supplied by the user. The example function simply returns a value of 1.0 for the rate at all positions.

The entry RATEF must appear in the function RATE. It is called by ETCH prior to any direct call to RATE. It is intended to allow the user to calculate constants, establish look-up tables, read in parameters from data cards, etc., prior to using the function. This can often result in the saving of a great deal of computation time.

The input variable, Z, is a complex variable giving the position of the point for which the rate is to be calculated.

Acknowledgements

The author wishes to express his gratitude to A. R. Neureuther, and P. I. Hagouel, who made numerous suggestions during the initial development of the algorithm; to M. O'Toole, who has incorporated the program, with many changes, into an exposure and development simulator, and especially to T. Van Duzer, who has provided continuous encouragement, suggestions, and computer funds.

References

[1] A.R. Neureuther, R.E. Jewett, P.I. Hagouel and T. Van Duzer, "Surface Etching Simulation and Applications in IC Processing", *Kodak Microelectronics Seminar Proceedings,* Monterey, California, October 1975

[2] F.H. Dill, A.R. Neureuther, J.A. Tuttle and E.J. Walker, "Modelling Projection Printing of Positive Photoresists", IBM Research, February 2, 1975, RC 5261

[3] P.I. Hagouel, "X-ray Lithographic Fabrication of Blazed Diffraction Gratings", Ph.D. Dissertation, University of California, Berkeley, 1976

[4] M.M. O'Toole, "Simulation of Optically Formed Image Profiles in Positive Photoresist", Memorandum No. UCB/ERL M79/42, Electronics Research Laboratory, University of California, Berkeley, June 1979

[5] *CAL RUN Fortran Guide,* Computer Center, University of California, Berkeley, Fall 1974

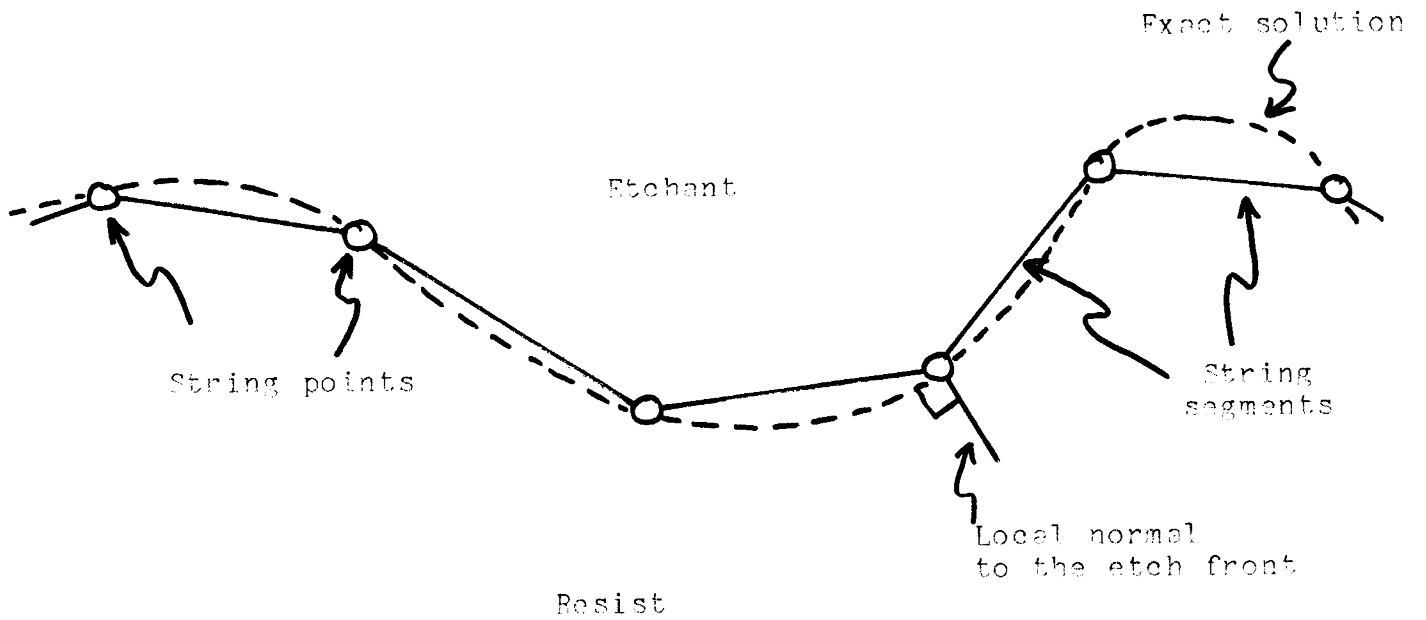


Fig. 1. String model approximation to the etch front.

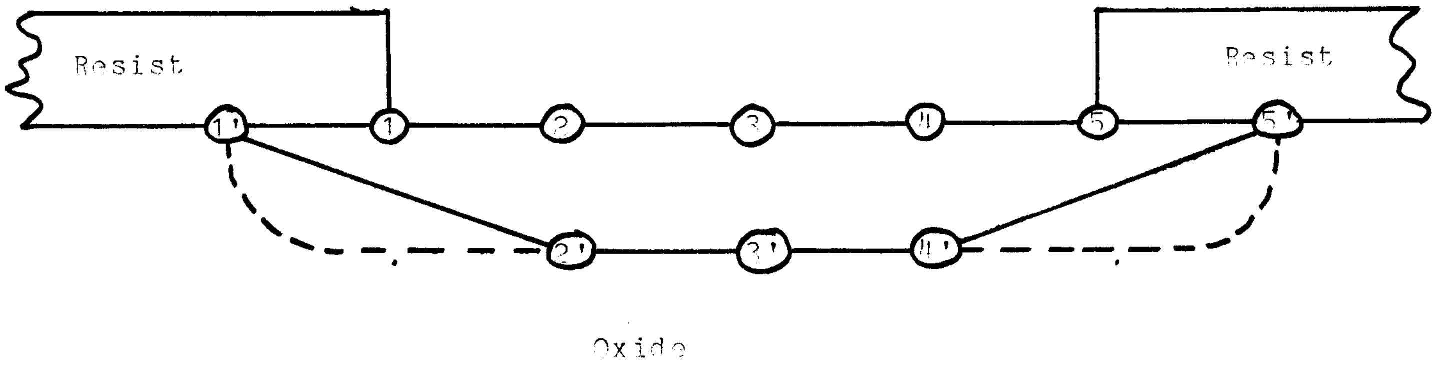
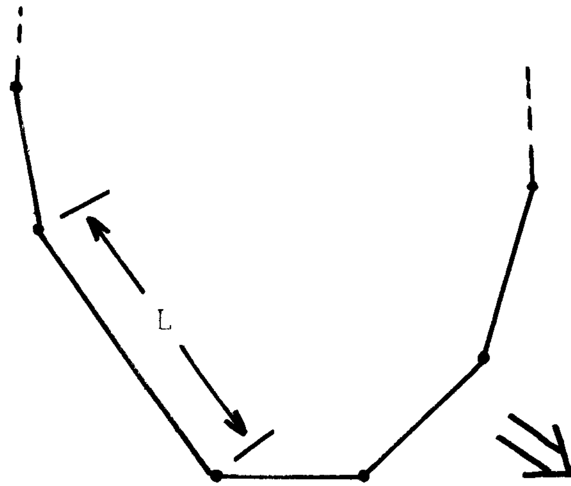


Fig. 2. Uniform rate etching of an oxide with undercutting.



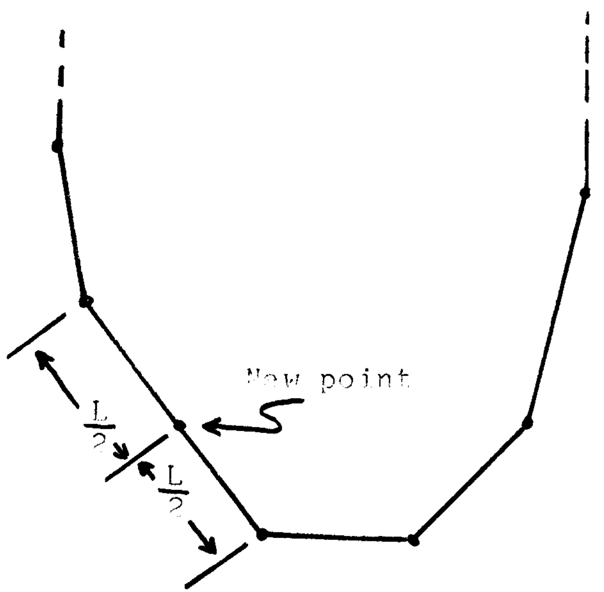


Fig. 3a. Reducing the length of a long segment by bisection.

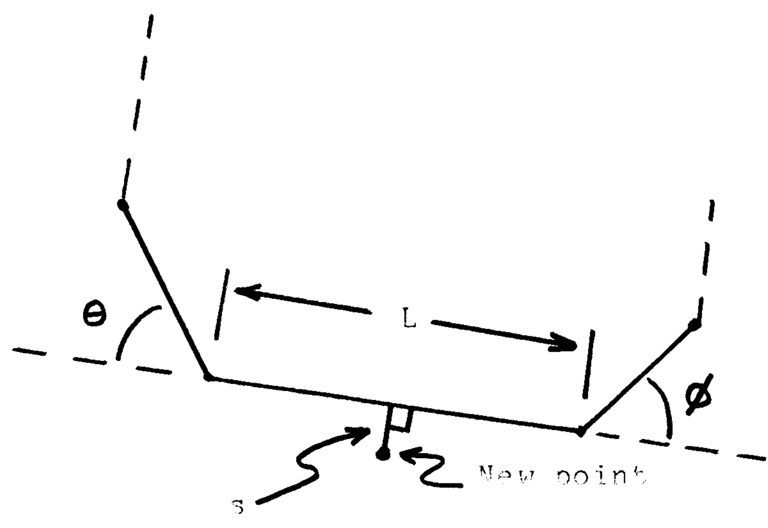


Fig. 3b. Improved position for the new point

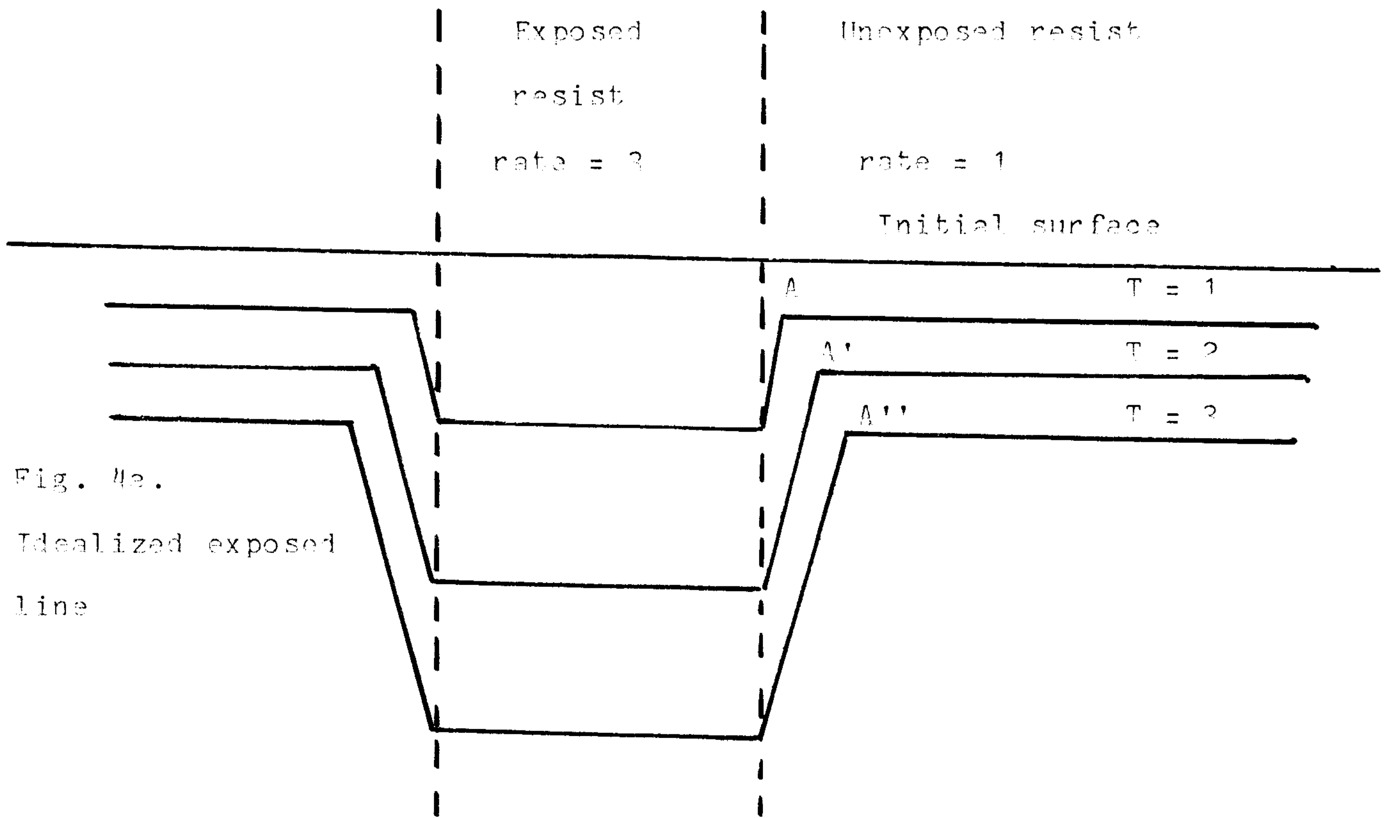


Fig. 4a. Idealized exposed line

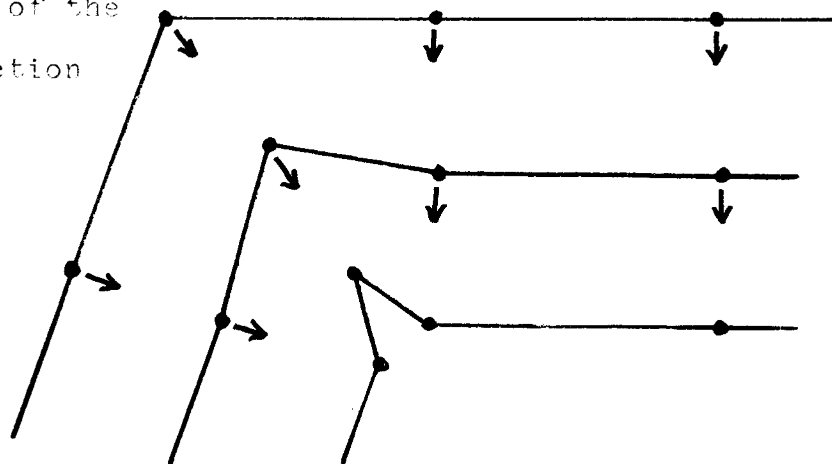


Fig. 4b. Detail of the region of contraction at corner A

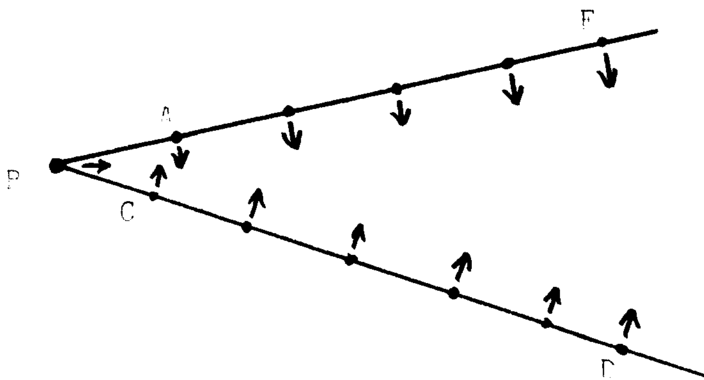


Fig. 5a. Region of contraction with a very acute angle of intersection.

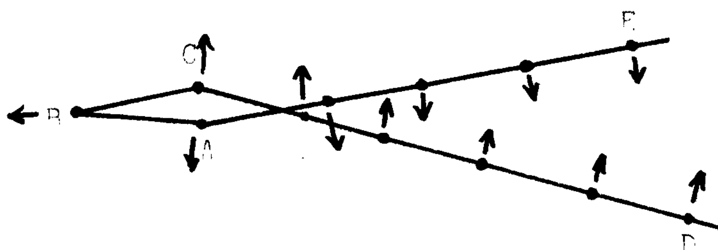


Fig. 5b. Loop budding

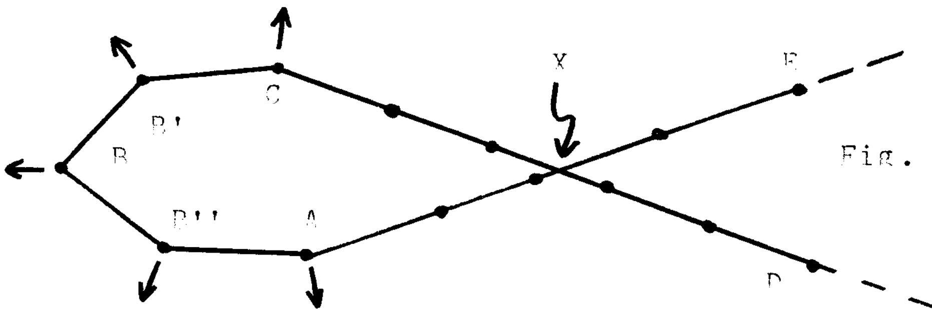
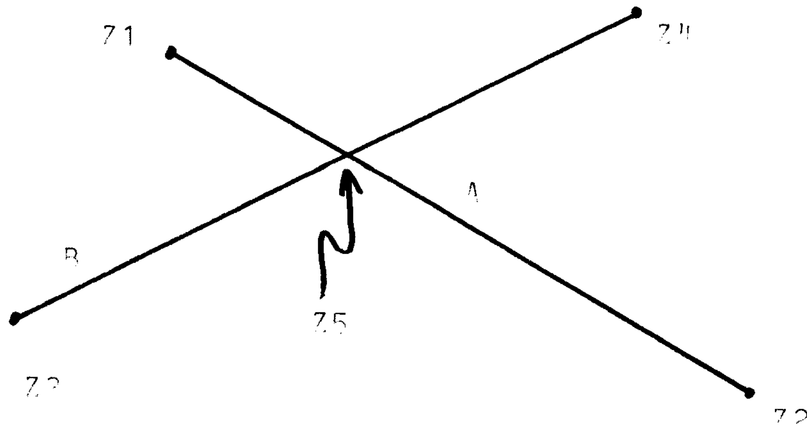


Fig. 5c. Loop expansion



$$Z5 = (1-a)Z1 + aZ2 = (1-b)Z3 + bZ4$$

Fig. 6. Checking segment intersection using parametric equations. Intersection shown for $a = 1/3$, $b = 1/2$.

Appendix A

The following is a listing of the etching algorithm for the RUN compiler on a CDC 6400 computer.

```
 PROGRAM ETCH(INPUT, OUTPUT, TAPE1=INPUT, TAPE20, TAPE21)
 COMMON//XY(200), LABEL(20), XMAX, YMAX, NPTS, CXYL, NBCHK, NBOUT,
C CXYR, TOUT, TSTEP, NSTEP, ET, TTOT, IFLAG, SMAX, SMIN, NOUT
C , LRT
 COMPLEX XY, CXYL, CXYR
 CALL READIN
 CALL LINEAR
 CALL PLTBGN
 DUMMY=RATEF(X)
 SMID=CABS(XY(1)-XY(2))
 SMIN=.75*SMID
 SMAX=2.*SMID
 TTOT=0.
 ET=TOUT/NBOUT
 TSTEP=ET/NBCHK
 IF(LRT.EQ.2HRT) CALL RTEST
 DO 4 M=1,NOUT
 DO 5 N=1,NBOUT
 CALL CYCLE
 IF(IFLAG.EQ.1) CALL CHKR
5 CONTINUE
 CALL DELOOP
 CALL PLTOUT
4 CONTINUE
 END ETCH


 SUBROUTINE READIN
 COMMON//XY(200), LABEL(20), XMAX, YMAX, NPTS, CXYL, NBCHK, NBOUT,
C CXYR, TOUT, TSTEP, NSTEP, ET, TTOT, IFLAG, SMAX, SMIN, NOUT
C , LRT
 COMPLEX XY, CXYL, CXYR
 DIMENSION LINES(18,8)
 REWIND 1
 READ 1, LINES
 IF(EOF,1)6,7
1 FORMAT(8A10)
7 CONTINUE
 PRINT 8, LINES
8 FORMAT(1H1, *INPUT DATA CARDS FOR PROGRAM ETCH AS PUNCHED*,//,
C X, *VARIABLE  *, 10(*DATACARD*), //,
```

```
 C X, *LABEL     *, 8A10, /, X, *LABEL     *, 8A10, /,
 C X, *LABEL     *, 8A10, /,
 C X, *XMAX      *, 8A10, /, X, *YMAX      *, 8A10, /,
 C X, *NPTS      *, 8A10, /, X, *CXYL RE   *, 8A10, /,
 C X, *CXYL IM   *, 8A10, /, X, *CXYR RE   *, 8A10, /,
 C X, *CXYR IM   *, 8A10, /, X, *XY(1) RE  *, 8A10, /,
 C X, *XY(1) IM  *, 8A10, /, X, *XY(NPTS)RE*, 8A10, /,
 C X, *XY(NPTS)IM*, 8A10, /, X, *TOUT      *, 8A10, /,
 C X, *NOUT      *, 8A10, /, X, *NBCHK     *, 8A10, /,
 C X, *NBOUT     *, 8A10, ///)
  REWIND 1
  READ 1, LABEL
2 FORMAT(F15.7)
  READ 2, XMAX, YMAX
3 FORMAT(I3)
  READ 3, NPTS
  READ 2, CXYL, CXYR, XY(1), XY(NPTS), TOUT
  READ 3, NOUT, NBCHK, NBOUT
  PRINT 4, LABEL, XMAX, YMAX, NPTS, CXYL, CXYR, XY(1), XY(NPTS),
 C  TOUT, NOUT, NBCHK, NBOUT
4 FORMAT(X, *INPUT DATA AS INTERPRETED:*, //,
 C  X, *LABEL   *, 8A10, /,
 C  X, *LABEL   *, 8A10, /,
 C  X, *LABEL   *, 4A10, //,
 C  X, *X MAX   *, E12.4,//,
 C  X, *Y MAX   *, E12.4, //,
 C  X, *NUMBER OF POINTS IN STARTING STRING *, I4, //,
 C  X, *DIRECTION OF LEFT ENDPOINT, X AND Y COMPONENTS*, 2F10.4,//,
 C  X, *DIRECTION OF RIGHT ENDPOINT, X AND Y COMPONENTS*,2F10.4,//,
 C  X, *X AND Y OF LEFT ENDPOINT *, 2E12.4, //,
 C  X, *X AND Y OF RIGHT ENDPOINT*, 2E12.4, //,
 C  X, *TIME BETWEEN OUTPUTS *, E12.4, //,
 C  X, *NUMBER OF OUTPUTS *, I3, //,
 C  X, *NUMBER OF ADVANCES BETWEEN CHECKS *, I3, //,
 C  X, *NUMBER OF CHECKS BETWEEN OUTPUTS *, I3, //)
  IF(NPTS.LT.5)    CALL KRASH(10HNPTS < 5   )
  IF(NPTS.GT.100)  CALL KRASH(10HNPTS > 100 )
  IF(NOUT.GT.20)   CALL KRASH(10HNOUT > 20  )
  IF(NOUT.LT.1)    CALL KRASH(10HNOUT < 1   )
  IF(NBCHK.GT.10)  CALL KRASH(10HNBCHK > 10 )
  IF(NBCHK.LT.1)   CALL KRASH(10HNBCHK < 1  )
  IF(NBOUT.GT.50)  CALL KRASH(10HNBOUT > 50 )
  IF(NBOUT.LT.1)   CALL KRASH(10HNBOUT < 1  )
  CXYL=CXYL/CABS(CXYL)
  CXYR=CXYR/CABS(CXYR)
  READ 10,LRT
10 FORMAT(A2)
  IF(EOF,1)11,12
11 LRT=2HNO
12 CONTINUE
  RETURN
6 CONTINUE
  PRINT 8, LINES
  PRINT 9
```

```
9 FORMAT(1H1, *LESS THAN 18 DATA CARDS*)
  STOP
  END READIN


  SUBROUTINE RTEST
  COMMON//XY(200), LABEL(20), XMAX, YMAX, NPTS, CXYL, NBCHK, NBOUT,
 C  CXYR, TOUT, TSTEP, NSTEP, ET, TTOT, IFLAG, SMAX, SMIN, NOUT
 C  , LRT
  COMPLEX XY, CXYL, CXYR
  REAL RTS(30,30)
  CALL SECOND(TIME1)
  PRINT 1,TIME1
1 FORMAT(1H1,   *START OF RTEST, CPU TIME = *, F6.3,//)
  DX=XMAX/29.
  DY=YMAX/29.
  RMAX=RATE(CMPLX(0.,0.))
  RMIN=RMAX
  DO 2 NX=1,30
  X=DX*(NX-1)
  DO 3 NY = 1,30
  Y=DY*(NY-1)
  RXY=RATE(CMPLX(X,Y))
  IF(RXY.LT.0.)GO TO 4
  RMAX=AMAX1(RMAX,RXY)
  RMIN= AMIN1(RMIN,RXY)
  RTS(NX,NY)=RXY
3 CONTINUE
2 CONTINUE
  RNORM=RMAX/.99
  DO 5 N = 1 , 900
  RTS(N)=RTS(N)/RNORM
5 CONTINUE
  RLMAX=RATE(XY(1))
  RLMIN=RLMAX
  DO 6 N=2,NPTS
  RL=RATE(XY(N))
  RLMAX=AMAX1(RLMAX,RL)
  RLMIN=AMIN1(RLMIN,RL)
6 CONTINUE
  DT=TOUT/NBOUT/NBCHK
  Q=DT*RMAX/SMIN
  PRINT 7, RMAX, RMIN, RLMAX, RLMIN, Q
7 FORMAT(1HX,   *MAXIMUM RATE OBSERVED = *, E9.3, /,
 C  X, *MINIMUM RATE OBSERVED = *, E9.3, /,
 C  X, *MAXIMUM RATE ALONG STARTING STRING = *, E9.3, /,
 C  X, *MINIMUM RATE ALONG STARTING STRING = *, E9.3, /,
 C  X, *WORST CASE ERROR COEFF (= MAX RATE X TIME STEP / MIN SEGMENT
 C LENGTH) = *, F10.6,/)
  PRINT 8, XMAX, YMAX, DX, DY, RNORM,((RTS(NX,NY), NX=1,30),NY=1,30)
8 FORMAT(/, X, *X FROM 0 TO *, E10.4, /,
 C  X, *Y FROM 0 TO *, E10.4, /,
 C  X,*X INCREMENT = *, E10.4, /,
 C  X, *Y INCREMENT = *, E10.4, /,
```

```
 C  X, *RATES NORMALIZED TO A VALUE OF *, E10.4,// ,
 C  X, *CAUTION, X AND Y SCALES NOT NECESSARILY EQUAL*,//,
 C  X, *NORMALIZED RATES:*,//,
 C  (1HX, 30F4.2, //))
  CALL SECOND (TIME2)
  TIME = TIME2-TIME1
  PRINT 9, TIME
9 FORMAT(//, X, *TOTAL CPU TIME IN RTEST = *, F7.3, /, 1H1)
  RETURN
4 CONTINUE
  PRINT 10, RXY, X, Y
10 FORMAT(/, X, *NEGATIVE RATE FOUND :*//,
 C  X, *RATE = *, E10.4, /,
 C  X, *   X = *, E10.4, /,
 C  X, *   Y = *, E10.4, //,
 C  X, *JOB ABORTED*)
  STOP
  END RTEST


  SUBROUTINE CHKR
  COMMON//XY(200), LABEL(20), XMAX, YMAX, NPTS, CXYL, NBCHK, NBOUT,
 C  CXYR, TOUT, TSTEP, NSTEP, ET, TTOT, IFLAG, SMAX, SMIN, NOUT
 C  , LRT
  COMPLEX XY, CXYL, CXYR
  COMPLEX CI
  DIMENSION INDEX(200)
  DATA CI/0.,1./
  CALL SECOND(TCPU)
  PRINT 20, TTOT, NPTS, TCPU
20 FORMAT( 1HX, *IN CHKR   TTOT, NPTS, TCPU = *, F20.5, I6, F7.3)
  IF((RATE(XY(1)).EQ.0.).AND.(RATE(XY(2)).EQ.0.)) GO TO 11
12 CONTINUE
  NCHNGE = 0
  NSTOP=NPTS-1
  DO 1 N=1,NSTOP
  C=CABS(XY(N+1)-XY(N))
  IF(C.GT.SMAX.OR.C.LT.SMIN) GO TO 2
3 CONTINUE
1 CONTINUE
  IF(NCHNGE.NE.0) GO TO 4
  RETURN
2 CONTINUE
  NCHNGE = NCHNGE+1
  INDEX(NCHNGE)=N
  GO TO 3
4 CONTINUE
  NADD=0
  MLAST=1
  DO 5 N=1,NCHNGE
  M=INDEX(N)+1+NADD
  T=CABS(XY(M)-XY(M-1))
  IF (T.GT.SMAX) GO TO 6
  IF(T.LT.SMIN) GO TO 7
```

```
 8 CONTINUE
 5 CONTINUE
   RETURN
 6 CONTINUE
   IF(NPTS.GE.200) CALL KRASH(10HNPTS > 200   )
   NADD=NADD+1
   DO 9 I=M,NPTS
   J=NPTS+M-I
   XY(J+1)=XY(J)
 9 CONTINUE
   NPTS=NPTS+1
   ANG1=1.6
   ANG2=1.6
   IF(M.GT.2)ANG1=-AIMAG(CLOG((XY(M+1)-XY(M-1))/(XY(M-1)-XY(M-2))))
   IF(M.LT. NPTS-1)ANG2=-AIMAG(CLOG((XY(M+2)-XY(M+1))/(XY(M+1)-XY(M-1
  C))))
   ANG=AMAX1(AMIN1( ANG1, ANG2),0.)/8.
   XY(M)=(XY(M)+XY(M-1))/2.+ANG*CI*(XY(M)-XY(M-1))
   GO TO 8
 7 CONTINUE
   IF(NPTS.LT.5) CALL KRASH(10HNPTS < 5      )
   IF(MLAST+1.EQ.M) GO TO 8
   NADD=NADD-1
   NPTS=NPTS-1
   MLAST=M
   XY(M-1)=.5*(XY(M-1)+XY(M))
   DO 10 K=M,NPTS
   XY(K)=XY(K+1)
10 CONTINUE
   GO TO 8
11 CONTINUE
   IF(NPTS.LE.10) GO TO 12
   IF(RATE(XY(2)).NE.0.) GO TO 12
   DO 13 N=2,NPTS
   XY(N-1)=XY(N)
13 CONTINUE
   NPTS = NPTS-1
   GO TO 11
   END


   SUBROUTINE DELOOP
   COMMON//XY(200), LABEL(20), XMAX, YMAX, NPTS, CXYL, NBCHK, NBOUT,
  C  CXYR, TOUT, TSTEP, NSTEP, ET, TTOT, IFLAG, SMAX, SMIN, NOUT
  C  , LRT
   COMPLEX XY, CXYL, CXYR
   S=SMAX*2.
   N=0
 1 CONTINUE
   N=N+1
   IF(N.GT.(NPTS-5)) RETURN
   YN=AIMAG(XY(N))
   XN=REAL(XY(N))
   M=N+3
```

```
2 CONTINUE
 M=M+1
 IF(M.GE.NPTS) GO TO 1
 IF(ABS(REAL(XY(M))-XN).GT.S) GO TO 2
 IF(ABS(AIMAG(XY(M))-YN).GT.S) GO TO 2
 ADENOM =AIMAG((XY(N+1)-XY(N))*CONJG(XY(M+1)-XY(M)))
 IF(ADENOM.EQ.0.) GO TO 2
 AA=AIMAG((XY(M)-XY(N))*CONJG(XY(M+1)-XY(M))) / ADENOM
 IF((AA.LE.0.).OR.(AA.GE.1.)) GO TO 2
 BDENOM =AIMAG((XY(M+1)-XY(M))*CONJG(XY(N+1)-XY(N)))
 IF(BDENOM.EQ.0.) GO TO 2
 BB= AIMAG((XY(N)-XY(M))*CONJG(XY(N+1)-XY(N)))/BDENOM
 IF((BB.LE.0.).OR.(BB.GE.1.)) GO TO 2
 XY(N+1)= (1.-AA)*XY(N)+AA*XY(N+1)
 JSTART=N+2
 JSTOP=NPTS-M+N+1
 DO 3 J=JSTART, JSTOP
 XY(J)= XY(J+M-N-1)
3 CONTINUE
 NOLD=NPTS
 NPTS=JSTOP
 PRINT 4, NOLD,NPTS, TTOT
4 FORMAT(/, X, *DELOOP, NOLD = *, I3, *, NPTS = *, I3, *  TIME = *,
 C  F9.3)
 N=N+1
 GO TO 1
 END


 SUBROUTINE LINEAR
 COMMON//XY(200), LABEL(20), XMAX, YMAX, NPTS, CXYL, NBCHK, NBOUT,
C  CXYR, TOUT, TSTEP, NSTEP, ET, TTOT, IFLAG, SMAX, SMIN, NOUT
C  , LRT
 COMPLEX XY, CXYL, CXYR
 COMPLEX XYST RT, XYSTEP
 XYSTEP=(XY(NPTS)-XY(1))/(NPTS-1)
 XYST RT=XY(1)-XYSTEP
 NSTOP=NPTS-1
 DO 1 N=2,NSTOP
 XY(N)=XYST RT+N*XYSTEP
1 CONTINUE
 RETURN
 END


 SUBROUTINE CYCLE
 COMMON//XY(200), LABEL(20), XMAX, YMAX, NPTS, CXYL, NBCHK, NBOUT,
C  CXYR, TOUT, TSTEP, NSTEP, ET, TTOT, IFLAG, SMAX, SMIN, NOUT
C  , LRT
 COMPLEX XY, CXYL, CXYR
 COMPLEX DL,DR,DT
 COMPLEX CI, CSTEP
 DATA CI/0., 1./
 IFLAG=0
```

```
      TMIN=1.E50
      TMAX=0.
      NSTEP=ET/TSTEP+.5
      IF(NSTEP.LT.1.OR.NSTEP.GT.50) CALL KRASH(10HSEE CYCLE1   )
      TSTEP=ET/NSTEP
      CSTEP=CI*TSTEP
      NSTOP=NPTS-1
      DO 1 N=1,NSTEP
      DL=XY(2)-XY(1)
      XY(1)=XY(1)+TSTEP*CXYL*RATE(XY(1))
      T=CABS(DL)
      TMIN=AMIN1(TMIN,T)
      TMAX=AMAX1(TMAX,T)
      DL=DL/T
      DO 2 M=2,NSTOP
      DR=XY(M+1)-XY(M)
      T=CABS(DR)
      TMIN=AMIN1(TMIN,T)
      TMAX=AMAX1(TMAX,T)
      DR=DR/T
      DT=DL+DR
      XY(M)=XY(M)+CSTEP*RATE(XY(M))*DT/CABS(DT)
      DL=DR
    2 CONTINUE
      XY(NPTS)=XY(NPTS)+TSTEP*CXYR*RATE(XY(NPTS))
    1 CONTINUE
      IF((TMAX.GT.SMAX).OR.(TMIN.LT.SMIN))IFLAG=1
      TTOT=TTOT+ET
      RETURN
      END


      SUBROUTINE KRASH(WORD)
      INTEGER WORD
      PRINT 1,WORD
    1 FORMAT(1H1, 132(1H*), ////, T30, *KRASH, YOUR CLUE IS *, A10, /)
      STOP
      END


      SUBROUTINE PLTOUT
      COMMON//XY(200), LABEL(20), XMAX, YMAX, NPTS, CXYL, NBCHK, NBOUT,
     C CXYR, TOUT, TSTEP, NSTEP, ET, TTOT, IFLAG, SMAX, SMIN, NOUT
     C , LRT
      COMPLEX XY, CXYL, CXYR
      REAL XT(200), YT(200)
      DO 1 N=1,NPTS
      XT(N)=REAL(XY(N))
      YT(N)=AIMAG(XY(N))
    1 CONTINUE
      WRITE(21)TTOT, NPTS, (XT(N), N=1,NPTS), (YT(N), N=1,NPTS)
      RETURN
      ENTRY PLTBGN
      WRITE(21)LABEL, XMAX, YMAX
```

```
RETURN
END

FUNCTION RATE(Z)
COMPLEX Z
RATE=1.
RETURN
ENTRY RATEF
RATE=1.
RETURN
END
```