Title
GIS Laboratory Exercises: Volume 2 Technical Issues (91-14)

Author
Veregin, Howard (editor)

Publication Date
1991-05-01

NCGIA

National Center for Geographic Information and Analysis

GIS Laboratory Exercises:

Volume 2, Technical Issues

Edited by

Howard Veregin
University of California, Santa Barbara

Technical Report 91-14

May 1991

Simonett Center for Spatial Analysis
University of California
35 10 Phelps Hall
Santa Barbara, CA 93106-4060
Office (805) 893-8224
Fax (805) 893-8617
ncgia@ncgia.ucsb.edu

State University of New York
301 Wilkeson Quad, Box 610023
Buffalo NY 14261-0001
Office (716) 645-2545
Fax (716) 645-5957
ncgia@ubvms.cc.buffalo.edu

University of Maine
348 Boardman Hall
Orono ME 04469-5711
Office (207) 581-2149
Fax (207) 581-2206
ncgia@spatial.maine.edu

Preface

This set of labs is designed to illustrate and reinforce principles presented in *Volume II: Technical Issues in* GIS of the NCGIA Core Curriculum *in GIS* (units 26 to 50). Since several of these labs were originally prepared in early 1989 for the test version of the Curriculum, they have been presented to students a number of times and have undergone extensive revision. We have also added some new labs to the original set.

While these labs are written for a specific software and hardware configuration, they are presented as models for generic lab development. Please modify them to suit your own configurations and curriculum. To assist with this task, we have included digital versions of the text. See Appendix A for more information.

Several people have contributed to the development of these labs. They were originally written by Michael Goodchild, Karen Kemp and Howard Veregin. We would like to thank the students and faculty at the University of California, Santa Barbara and other universities who participated in the evaluation of the Core Curriculum and provided valuable revision suggestions. The National Science Foundation is also thanked for its contribution through the establishment of the National Center for Geographic Information and Analysis.

Karen K Kemp
Santa Barbara, May 1991

Assignment 1
Coordinate Systems

Objectives: In this assignment you will be using some simple QuickBASIC programs to convert between different coordinate systems, make distance calculations, and draw maps on the screen.

Files: This assignment requires four QuickBASIC programs (LL.BAS, GCDIST.BAS, EUCLID.BAS and MAP.BAS) and a file containing coordinate data for Africa (AFRICA.DAT).

Coordinate System Conversions: Run the QuickBASIC program called LL.BAS to compute the latitude and longitude for each of the points listed in Table 1. Points A through F were derived from 1:25,000 US Geological Survey topographic maps of Maine. Points G and H were derived from a 1:100,000 topographic map of the area near Sydney, Australia. Write your answers (in degrees, minutes and seconds) in the appropriate columns of the Table. Be sure to designate latitudes with an N or an S, and longitudes with an E or a W.

1. a) Do the latitudes and longitudes you computed for the points seem to be correct? (You will need to refer to a map to answer this.) b) Can you account for any discrepancies you observe?

2. Compare your answers for points D and F. Can you account for the difference in UTM Eastings for these two points?

Table 1.

Point	UTM Zone	UTM Easting	UTM Northing	Latitude	Longitude
A	19	416800	4627250		
B	19	413390	4622340		
C	19	254470	4672580		
D	19	252025	4667800		
E	18	737650	4667450		
F	18	747975	4667800		
G	56	330650	6241950		
H	56	315300	6236050		

Distance Calculations: Run the program GCDIST.BAS, which calculates the distance between points on the earth's surface using the great circle distance formula. The program assumes a value of 6371 km for the radius of the earth. Use the latitude and longitude values you entered into Table 1 to calculate the great circle distance between each pair of points in Table 2.

Table 2.

Pair of points	Great circle distance (km)	Euclidean distance (km)	Difference (km)	Difference as a percentage of Euclidean distance
A-B				
C-D				
G-H				
D-F				
C-E				

Now examine the program and answer the following questions.

3. What computation is performed by lines 1700, 1900, 2200 and 2400?

4. a) Are all of the parentheses on line 2600 necessary? b) Write out the line using the minimum number of parentheses needed to preserve the meaning of the equation. c) Why would you want to use parentheses when you don't really need them?

5. If you were to increase the estimate of the earth's radius by 1 percent, how would the great circle distance estimates be affected?

6. a) What happens if you enter latitude and longitude values incorrectly (e.g., enter characters instead of numbers, omit the commas, include too many commas, etc.)? b) What happens if you enter invalid latitude or longitude values (e.g., latitudes greater than 90 or longitudes greater than 180)?

Load the program called EUCLID.BAS, which calculates distances between pairs of points based on the Euclidean distance formula. Before you run this program, you will have to make several changes, as follows:

a) Insert an message between the quotation marks on line 1900.

b) Add the appropriate line number to the GOTO statement in line 2100.

c) Find and correct the error in line 2300.

d) Add a line (number 2400) to convert the computed distance to kilometers. (UTM Northings and Eastings are given in meters, but your answer should be expressed in kilometers.)

Save the program once you have made these changes. Run the program to calculate the Euclidean distance between each pair of points listed in Table 2. Compute (by hand) the difference between the great circle and Euclidean distances, and the difference as a percentage of the Euclidean distance. Enter your answers in the appropriate column in Table 2.

7. Based on your knowledge of the UTM projection, how might you account for the differences in the values in the last column of Table 2?

Make a printout of EUCLID.BAS and hand it in along with the assignment.

Coordinate System Conversions: Now that you have some experience with QuickBASIC programming, open LL.BAS again. Examine the program to answer the following questions.

8. a) What are the # symbols for in line 1700 and others following it?
   b) Why are they used?

9. a) What are the ! symbols for in line 5300 and others following it?
   b) Why are they used?

10. Lines 5000 to 5200 contain a loop that assigns a value to the elements of an array called cmer. These elements are specified as cmer(zone), where zone varies between 1 and 60. Using the equation given on line 5100, calculate the value for cmer(22) and cmer(50).

    cmer(22) =

    cmer(50) =

11. After line 13100 you can observe occurrences of INT(LATITUDE), INT(MIN) and INT(SEC). What does the INT function do and why is it used here?

Graphics Programming: The final step in the assignment is to modify a QuickBASIC program (called MAP.BAS) that displays a map of Africa on the screen using the Mercator projection. Run this program. The coastline of Africa should appear in white. The dashed white horizontal lines are lines of latitude (parallels) ranging from 30 degrees N to 30 degrees S. The central dashed line is the equator.

12. a) Based on the output of the program, would you say that lines of latitude are evenly spaced on Mercator's projection? b. If they are not evenly spaced, then do they get closer or father apart as you move away from the equator?

Now examine MAP.BAS more closely. You need to modify this program so that it displays a second map on top of the Mercator map. This second map will be based on the Lambert cylindrical equal-area projection. Note that the procedure for displaying the Mercator map is broken into three steps in MAP.BAS. Step 1 (lines 3000 to 4000) involves converting the longitude and latitude values for Africa (read in from the file AFRICA.DAT) into x- and y-coordinates. Step 2 (lines 5000 to 5400) draws the Mercator map by connecting adjacent x-y coordinate pairs with straight lines. Step 3 (lines 6000 to 6700) draws the lines of latitude for Mercator's projection.

Modifying the program to display two maps is actually a very simple task. Basically, steps 1 through 3 must be reproduced between lines 6700 and 20000, making a few changes in order to display the Lambert map in a different color. There are only three important changes you need to make:

a) The equation for computing the y-coordinate for Mercator's projection is

y = LOG (TAN (p / 4 + j / 2))

where j is the latitude (in radians). You can see this equation in action on lines 3900 and 6500 of the program. For the Lambert projection, the equation is

y = SIN (j)

b) The white color of the Mercator map is defined by the number 15 in lines 5300 and 6600. To get a different color, use a different number.

Any value between 0 and 15 is acceptable. Experiment or consult the QuickBASIC programming manual.

c) The value 8738 in line 6600 draws the lines of latitude as dashed lines on the Mercator map. You can also use this value for the Lambert map but, since the equator is at the same location on both maps, one of the equators will be completely hidden behind the other. Experiment with other values here or consult the QuickBASIC programming manual.

Once you have made the necessary changes to MAP.BAS, save the program as a text file. You may want to change the name of the file (e.g., MAP1.BAS) to preserve the original program. Run the program after saving it. Make a printout of your program and hand it in along with assignment.

13. Based on the output of your program, what can you say about the relative stretching or flattening of shapes on the two projections as you move away from the equator?

Optional: Modify the MAP.BAS program to display Tissot's Indicatrix for the parallels 0, 10, 20 and 30 degrees S. Display the Indicatrix ellipses on the right side of the map. Superimpose the ellipses for the Lambert projection over those for the Mercator projection. Use the same colors for the ellipses that you used for the maps themselves.

For the Lambert projection, the equations for the major and minor axes of the Indicatrix (k and h, respectively) are

k = SEC (j) = 1 / COS (j)
h = COS (j)

where j is the latitude (in radians). For the Mercator projection, k and h are equivalent:

k = h = SEC (j) = 1 / COS (j)

You will need to use the CIRCLE function to draw the ellipses. The aspect ratio for the function should be defined as h/k and the "radius" of the ellipse should be defined as k/15 (the value of 15 is used as a simple scaling factor).

What does the Indicatrix tell you about relative amounts of angular deformation and areal exaggeration for the two projections?

Hand in a printout of your program.

```
100   'GCDIST.BAS

200   'This program determines the great circle distance between two points.
300   'It assumes a spherical earth with a radius of 6371 kilometers.

400   radius = 6371                        'radius of earth (km)
500   pi = 3.14159                         'pi, the famous constant
600   degrad = (2 * pi) / 360              'to convert degrees to radians

700   CLS
800   DO

1500        'Get latitude and longitude of points and convert to decimal degrees.

1600        INPUT "Enter the latitude of the first point: ", deg, min, sec
1700        lat1 = (deg + (min / 60) + (sec / 3600)) * degrad
1800        INPUT "Enter the longitude of the first point: ", deg, min, sec
1900        lon1 = (deg + (min / 60) + (sec / 3600)) * degrad
2000        PRINT
2100        INPUT "Enter the latitude of the second point: ", deg, min, sec
2200        lat2 = (deg + (min / 60) + (sec / 3600)) * degrad
2300        INPUT "Enter the longitude of the second point: ", deg, min, sec
2400        lon2 = (deg + (min / 60) + (sec / 3600)) * degrad
2500        PRINT

2550        'Compute distance between the two points.

2600        x = (SIN(lat1) * SIN(lat2)) + (COS(lat1) * COS(lat2) * COS(lon1 - lon2))
2700        acosx = ATN(SQR(1 - (x ^ 2)) / x)
2800        dist = radius * acosx

2850        'Print distance.

2900        PRINT "The great circle distance is"; dist; "km."
3000        PRINT

3050        'Get another pair of points?

3100        PRINT "Calculate another distance? [y/n]"
3200        IF INPUT$(1) = "n" THEN EXIT DO

3300  LOOP

3400  END
```

```
100     'LL.BAS

200     'This program converts UTM coordinates to latitude/longitude.
300     'The program handles only north latitude and west longitude.

400     'Dimension arrays.

500     DIM cmer(61), eprm(2), orgn(2), sphin(5), grco(2), spco(2)

600     'Enter the UTM zone number for a point.

700     CLS
800     INPUT "Enter UTM zone (or a 0 to quit): ", utmzone
900     IF utmzone = 0 THEN
1000        STOP
1100    END IF

1150    'Enter coordinates of the point.

1200    INPUT "Enter UTM easting: ", utmx
1300    INPUT "Enter UTM northing: ", utmy

1350    'Some constants...

1400    raddeg = 57.29578
1500    k01 = 0.75
1600    k02 = 0.703125
1700    k03 = 0.68359375#
1800    k04 = 0.67291259765625#
1900    k05 = 0.6661631419939577#
2000    k06 = 0.9375
2100    k07 = 1.025390625#
2200    k08 = 1.07666015625#
2300    k09 = 1.110271903323263#
2400    k10 = 0.234375
2500    k11 = 0.41015625#
2600    k12 = 0.538330078125#
2700    k13 = 0.634460449921875#
2800    k14 = 0.068359375#
2900    k15 = 0.15380859375#
3000    k16 = 0.2379226684570312#
3100    k17 = 0.01922607421875#
3200    k18 = 0.0528717041015625#
3300    k19 = 5.28717041015625D-03
```

```
4000    'Compute central meridians of UTM zones.

5000    FOR zone = 1 TO 60
5100          cmer(zone) = (zone * 6 - 180 - 3) / raddeg
5200    NEXT zone

5250    'Constants for converting coordinates from meters.

5300    eprm(1) = 6378206!
5400    eprm(2) = 6356584!
5500    orgn(1) = 0!
5600    orgn(2) = 500000!
5700    scal = 0.9996
5800    grco(2) = utmx
5900    grco(1) = utmy
6000    utmz% = utmzone

6100    'Calculate ellipsoid parameters.

6200    asem = eprm(1)
6300    bsem = eprm(2)
6400    asq = asem * asem
6500    bsq = bsem * bsem
6600    aux1 = asq - bsq
6700    eccsq = aux1 / asq
6800    eps = aux1 / bsq
6900    cprm = bsq / asem
7000    eccp4 = eccsq * eccsq
7100    eccp6 = eccp4 * eccsq
7200    eccp8 = eccp6 * eccsq
7300    eccp10 = eccp8 * eccsq
7400    a = 1! + k01 * eccsq + k02 * eccp4 + k03 * eccp6 + k04 * eccp8 + k05 * eccp10
7500    b = k01 * eccsq + k06 * eccp4 + k07 * eccp6 + k08 * eccp8 + k09 * eccp10
7600    c = k10 * eccp4 + k11 * eccp6 + k12 * eccp8 + k13 * eccp10
7700    d = k14 * eccp6 + k15 * eccp8 + k16 * eccp10
7800    e = k17 * eccp8 + k18 * eccp10
7900    f = k19 * eccp10
8000    gradm1 = 1! / (cprm * a)

8100    'Return to original plane coordinates.

8200    ncoo = (grco(1) - orgn(1)) / scal
8300    ecoo = (grco(2) - orgn(2)) / scal
8400    ecoosq = ecoo * ecoo
```

```
8500   'Iterate for phipr (latitude of point on central meridian).

8700   phipr = gradm1 * ncoo
8800   cphipr = COS(phipr)
8900   sphipr = SIN(phipr)
9000   aux1 = sphipr
9100   twocos = 2! * cphipr
9200   aux2 = twocos * aux1
9300   sphin(1) = aux2
9400   even = 1
9500   FOR i = 3 TO 10
9600          aux3 = twocos * aux2 - aux1
9700          aux1 = aux2
9800          aux2 = aux3
9900          even = 1 - even
10000         IF even = 1 THEN
10100                sphin(i / 2) = aux3
10200         END IF
10300  NEXT i

10400  'Calculate meridian arc length.

10500  temp = a * phipr - b * sphin(1) / 2! + c * sphin(2) / 4! - d * sphin(3) / 6!
10600  mrdarc = cprm * (temp + e * sphin(4) / 8! - f * sphin(5) / 10!)

10700  'If difference is less than 0.1 mm, stop iterating.

10800  aux1 = ncoo - mrdarc
10900  IF ABS(aux1) < 0.0001 THEN
11000         GOTO 11500
11100  END IF
11200  phipr = phipr + aux1 * gradm1
11300  GOTO 8800

11400  'Calculate phipr- and eprm-dependent values.

11500  tpr = sphipr / cphipr
11600  tprsq = tpr * tpr
11800  etapsq = eps * cphipr * cphipr
11900  npr = asem / SQR(1! - eccsq * sphipr * sphipr)
12000  nprsq = npr * npr
12100  edvn = ecoo / npr
12200  edvnsq = ecoosq / nprsq
```

```
12300   'Calculate spheroidal coordinates.

12400   temp = 5! + 3! * tprsq + 6! * etapsq * (1! - tprsq)
12500   temp = temp + edvnsq / 30! * (61! + 90! * tprsq * (1! + .5 * tprsq))
12600   spco(1) = phipr + tpr * edvnsq * .5 * (-1! - etapsq + edvnsq / 12! * temp)
12700   temp = -1 - 2 * tprsq - etapsq + (edvnsq / 20 * (5 + tprsq * (28 + 24 * tprsq)))
12800   spco(2) = cmer(utmz%) + edvn / cphipr * (1! + edvnsq / 6! * temp)

12850   'Calculate latitude and longitude of point.

12900   latitude = ABS(spco(1) * raddeg)
13000   longitude = ABS(spco(2) * raddeg)
13100   deg = INT(latitude)
13200   min = 60 * (latitude - deg)
13300   sec = 60 * (min - INT(min))
13400   PRINT
13500   PRINT "Latitude = "; deg; "deg. "; INT(min); "min. "; INT(sec); "sec."
13600   deg = INT(longitude)
13700   min = 60 * (longitude - deg)
13800   sec = 60 * (min - INT(min))
13900   PRINT "Longitude = "; deg; "deg. "; INT(min); "min. "; INT(sec); "sec."
14000   PRINT

14200   'Return to beginning of program.

14300   GOTO 800

15000   END
```

```
100   'MAP.BAS

200   'This program draws a map of Africa using Mercator's projection.

300   'Hard-wire the screen display parameters and dimension arrays.

1000  CLS
1100  SCREEN 12
1200  WINDOW (-1, -1)-(1.6667, 1)
1300  DIM dlong(110), dlat(110), x(110), y(110)

1500  pi = 3.14159                'pi, the famous constant

2000  'Read the 110 longitude and latitude values for Africa.

2200  OPEN "AFRICA.DAT" FOR INPUT AS #1
2400  FOR i = 1 TO 110
2500        INPUT #1, dlong(i), dlat(i)
2600  NEXT

3000  'Mercator's projection (Step 1).  Convert longitude and latitude from
3100  'degrees to radians.  Calculate x- and y-coordinates for the projection.

3500  FOR i = 1 TO 110
3600        rlong = dlong(i) * 2 * pi / 360
3700        rlat = dlat(i) * 2 * pi / 360
3800        x(i) = rlong
3900        y(i) = LOG(TAN(pi / 4 + rlat / 2))
4000  NEXT

5000  'Mercator's projection (Step 2).  Draw the map in white (color = 15).

5200  FOR i = 1 TO 109
5300        LINE (x(i), y(i))-(x(i + 1), y(i + 1)), 15
5400  NEXT

6000  'Mercator's projection (Step 3).  Draw lines of latitude as dashed lines.

6300  FOR dparallel = -30 TO 30 STEP 10
6400        rparallel = dparallel * 2 * pi / 360
6500        parallel = LOG(TAN(pi / 4 + rparallel / 2))
6600        LINE (-1, parallel)-(1.6667, parallel), 15, , 8738
6700  NEXT

20000 END
```

Assignment 1
Answer Key

1. **a)** Yes, except for the Australian ones. **b)** The program cannot handle south latitudes or east longitudes.

2. The latitude and longitude of the two points are approximately the same. D and F are the same point. They have different UTM coordinates because they are in two adjacent UTM zones.

Table 1.

Point	UTM Zone	UTM Easting	UTM Northing	Latitude	Longitude
A	19	416800	4627250	41°47'40"N	70°0'4"W
B	19	413390	4622340	41°45'0"N	70°2'30"W
C	19	254470	4672580	42°10'7"N	71°58'20"W
D	19	252025	4667800	42°7'30"N	71°59'59"W
E	18	737650	4667450	41°7'30"N	72°7'30"W
F	18	747975	4667800	42°7'30"N	72°0'0"W
G	56	330650	6241950	56°17'38"S	150°15'48"E
H	56	315300	6236050	56°14'7"S	150°1'12"E

Table 2.

Pair of points	Great circle distance (km)	Euclidean distance (km)	Difference (km)	Difference as a percentage of Euclidean distance
A-B	5.820	5.978	-0.158	-2.645
C-D	5.388	5.369	0.019	0.356
G-H	16.313	16.445	-0.131	-0.799
D-F	0		-- different UTM zone --	
C-E	13.560		-- different UTM zone --	

3. Conversion of degrees, minutes and seconds to decimal degrees.

4. a) No.
b) x = SIN(lat1) * SIN(lat2) + COS(lat1) * COS(lat2) * COS(lon1 - lon2)
c) To make the program easier to read.

5. They would also increase by 1 percent. (See line 2800).

6. a) The program responds with "redo from start" and then prompts for new values. b) The program computes and prints a value for distance, even though the value is invalid.

7. The degree and sign of the error varies over each UTM zone. The lowest error occurs along the standard lines parallel to the central meridian, and increases towards the central meridian and the edges of the zone.

8. a) They designate double precision variables. b) They are used when high precision is required.

9. a) They designate single precision variables. b) They are used to save memory or increase the speed of computation.

10. cmer(22) = -0.890
cmer(50) = 2.042

11. The function returns the largest integer less than or equal to the argument. It is used here to convert decimal degrees to degrees, minutes and seconds.

12. a) Lines of latitude are not evenly spaced on Mercator's projection. b) They get father apart as you move away from the equator.

13. The Mercator projection appears to stretch shapes more, while the Lambert projection seems to squash them. (This is only true in a relative sense, since the Mercator projection is actually conformal.)

Optional: Tissot's Indicatrix shows that angular deformation for the Lambert projection increases as you move away from the equator. However, as the ellipses always have the same area, there is no areal exaggeration (equal-area). For the Mercator projection the ellipses are always circular, indicating that there is no angular deformation (conformal). However, areal exaggeration is apparent since the ellipses get larger as you move away from the equator.

Assignment 1
Program Listing (Answer Key)

```
100  'EUCLID1.BAS

200  'This program determines the Euclidean distance between
300  'two points based on their UTM x- and y-coordinates.
400  'It assumes that the points are in the same UTM zone.
500  'Lines 1900, 2100 and 2300 have been modified.  Line 2400 has been added.

550  CLS
600  DO

800       'Get data for a pair of points.

1000      INPUT "Enter UTM zone of first point: ", zone1
1100      INPUT "Enter UTM easting of first point: ", x1
1200      INPUT "Enter UTM northing of first point: ", y1
1300      PRINT
1400      INPUT "Enter UTM zone of second point: ", zone2
1500      INPUT "Enter UTM easting of second point: ", x2
1600      INPUT "Enter UTM northing of second point: ", y2
1700      PRINT

1750      'If points are in different zones, don't compute distance.

1800      IF zone1 <> zone2 THEN
1900           PRINT "Those points are in different zones."
2000           PRINT
2100           GOTO 2700
2200      END IF

2250      'Compute and print distance.

2300      dist = SQR(((x1 - x2) ^ 2) + ((y1 - y2) ^ 2))
2400      dist = dist / 1000
2500      PRINT "The Euclidean distance is"; dist; "km."
2600      PRINT

2650      'Get data for another pair of points?

2700      PRINT "Calculate another distance? [y/n]"
2800      IF INPUT$(1) = "n" THEN EXIT DO

2900 LOOP
3000 END
```

```
100   'MAP1.BAS

200   'This program draws a map of Africa using Mercator's projection.
210   'Over this it superimposes a second map using Lambert's
220   'cylindrical equal-area projection.

300   'Hard-wire the screen display parameters and dimension arrays.

1000  CLS
1100  SCREEN 12
1200  WINDOW (-1, -1)-(1.6667, 1)
1300  DIM dlong(110), dlat(110), x(110), y(110)

1500  pi = 3.14159                  'pi, the famous constant

2000  'Read the 110 longitude and latitude values for Africa.

2200  OPEN "AFRICA.DAT" FOR INPUT AS #1
2400  FOR i = 1 TO 110
2500       INPUT #1, dlong(i), dlat(i)
2600  NEXT

3000  'Mercator's projection (Step 1).  Convert longitude and latitude from
3100  'degrees to radians.  Calculate x- and y-coordinates for the projection.

3500  FOR i = 1 TO 110
3600       rlong = dlong(i) * 2 * pi / 360
3700       rlat = dlat(i) * 2 * pi / 360
3800       x(i) = rlong
3900       y(i) = LOG(TAN(pi / 4 + rlat / 2))
4000  NEXT

5000  'Mercator's projection (Step 2).  Draw the map in white (color = 15).

5200  FOR i = 1 TO 109
5300       LINE (x(i), y(i))-(x(i + 1), y(i + 1)), 15
5400  NEXT

6000  'Mercator's projection (Step 3).  Draw lines of latitude as dashed lines.

6300  FOR dparallel = -30 TO 30 STEP 10
6400       rparallel = dparallel * 2 * pi / 360
6500       parallel = LOG(TAN(pi / 4 + rparallel / 2))
6600       LINE (-1, parallel)-(1.6667, parallel), 15, , 8738
6700  NEXT
```

```
7000  'Lambert's projection (Step 1).  Convert longitude and latitude from
7100  'degrees to radians.  Calculate x- and y-coordinates for the projection.

7500  FOR i = 1 TO 110
7600        rlong = dlong(i) * 2 * pi / 360
7700        rlat = dlat(i) * 2 * pi / 360
7800        x(i) = rlong
7900        y(i) = SIN(rlat)
8000  NEXT

9000  'Lambert's projection (Step 2).
9100  'Draw the map in red (color = 12) or some other color.

9200  FOR i = 1 TO 109
9300        LINE (x(i), y(i))-(x(i + 1), y(i + 1)), 12
9400  NEXT

10000 'Lambert's projection (Step 3).  Draw lines of latitude as dashed lines.
10100 'Change the value at the end of the LINE statement to get a different
10200 'dashed line than that used for the Mercator projection.

10300 FOR dparallel = -30 TO 30 STEP 10
10400       rparallel = dparallel * 2 * pi / 360
10500       parallel = SIN(rparallel)
10600       LINE (-1, parallel)-(1.6667, parallel), 12, , 21845
10700 NEXT

20000 END
```

```
100   'MAP2.BAS

200   'This program draws a map of Africa using Mercator's projection.
210   'Over this it superimposes a second map using Lambert's
220   'cylindrical equal-area projection.
230   'Next it draws the ellipses for Tissot's Indicatrix for the two
240   'projections (in appropriate colors) for latitudes 0, 10, 20 and 30 S.

300   'Hard-wire the screen display parameters and dimension arrays.

1000  CLS
1100  SCREEN 12
1200  WINDOW (-1, -1)-(1.6667, 1)
1300  DIM dlong(110), dlat(110), x(110), y(110)

1500  pi = 3.14159                  'pi, the famous constant

2000  'Read the 110 longitude and latitude values for Africa.

2200  OPEN "AFRICA.DAT" FOR INPUT AS #1
2400  FOR i = 1 TO 110
2500        INPUT #1, dlong(i), dlat(i)
2600  NEXT

3000  'Mercator's projection (Step 1).  Convert longitude and latitude from
3100  'degrees to radians.  Calculate x- and y-coordinates for the projection.

3500  FOR i = 1 TO 110
3600        rlong = dlong(i) * 2 * pi / 360
3700        rlat = dlat(i) * 2 * pi / 360
3800        x(i) = rlong
3900        y(i) = LOG(TAN(pi / 4 + rlat / 2))
4000  NEXT

5000  'Mercator's projection (Step 2).  Draw the map in white (color = 15).

5200  FOR i = 1 TO 109
5300        LINE (x(i), y(i))-(x(i + 1), y(i + 1)), 15
5400  NEXT
```

```
6000 'Mercator's projection (Step 3). Draw lines of latitude as dashed lines.

6300 FOR dparallel = -30 TO 30 STEP 10
6400       rparallel = dparallel * 2 * pi / 360
6500       parallel = LOG(TAN(pi / 4 + rparallel / 2))
6600       LINE (-1, parallel)-(1.6667, parallel), 15, , 8738
6700 NEXT

6705 'Mercator's projection (Step 4). Calculate Indicatrix axes
6710 'for the required parallels and then plot the ellipses.
6715 'Note that radius of circle must be scaled (in this case,
6720 'divided by 15) to fit on the screen. Note that we use k (not h)
6725 'as the so-called radius of the ellipse since the CIRCLE function
6730 'will use this as the x-radius if aspect is less 1.

6750 FOR dparallel = -30 TO 0 STEP 10
6760         rparallel = dparallel * 2 * pi / 360
6770         parallel = LOG(TAN(pi / 4 + rparallel / 2))
6780         h = 1 / COS(rparallel)
6790         k = h
6800         aspect = h / k
6810         CIRCLE (1.3333, parallel), (k / 15), 15, , , aspect
6820 NEXT

7000 'Lambert's projection (Step 1). Convert longitude and latitude from
7100 'degrees to radians. Calculate x- and y-coordinates for the projection.

7500 FOR i = 1 TO 110
7600       rlong = dlong(i) * 2 * pi / 360
7700       rlat = dlat(i) * 2 * pi / 360
7800       x(i) = rlong
7900       y(i) = SIN(rlat)
8000 NEXT

9000 'Lambert's projection (Step 2).
9100 'Draw the map in red (color = 12) or some other color.

9200 FOR i = 1 TO 109
9300       LINE (x(i), y(i))-(x(i + 1), y(i + 1)), 12
9400 NEXT
```

```
10000 'Lambert's projection (Step 3).  Draw lines of latitude as dashed lines.
10100 'Change the value at the end of the LINE statement to get a different
10200 'dashed line than that used for the Mercator projection.

10300 FOR dparallel = -30 TO 30 STEP 10
10400       rparallel = dparallel * 2 * pi / 360
10500       parallel = SIN(rparallel)
10600       LINE (-1, parallel)-(1.6667, parallel), 12, , 21845
10700 NEXT

10705 'Lambert's projection (Step 4).  Calculate Indicatrix axes
10710 'for the required parallels and then plot the ellipses.

10750 FOR dparallel = -30 TO 0 STEP 10
10760         rparallel = dparallel * 2 * pi / 360
10770         parallel = SIN(rparallel)
10780         h = COS(rparallel)
10790         k = 1 / h
10800         aspect = h / k
10810         CIRCLE (1.3333, parallel), (k / 15), 12, , , aspect
10820 NEXT

20000 END
```

Assignment 2
Vector Data Structures (I)

Objectives: This assignment focuses on the manipulation of vector data. You will be using QuickBASIC to compute polygon areas, perform point-in-polygon tests and locate line intersections.

Files: This assignment requires four QuickBASIC programs (PIP.BAS, DARTS.BAS, INTER.BAS and FRACTAL.BAS) and two data files containing the vector representation of polygons (POLYGON.DAT and POLYGON2.DAT).

Point-in-Polygon Test: The program called PIP.BAS contains a point-in-polygon algorithm that determines whether a specified point falls within a given polygon. Before you can run this program you will have to make the following additions between lines 100 and 5000:

a) Use the CLS statement to clear the screen.

b) Open the polygon data file for reading. The statement to use is:

```
OPEN "POLYGON.DAT" FOR INPUT AS #1
```

c) This data file defines a polygon as a set of x,y-coordinate pairs which, when joined by straight-line segments, describe a closed geometric figure. In order to read in the coordinates, you will first need to know the number of coordinates in the file. This number is given in the first line of the file. Read in this number and assign it to a variable called n. The statement to use is:

```
INPUT #1, n
```

d) Use a DIM statement to dimension two arrays (called x and y) to store the x,y-coordinate pairs. The dimension of these arrays should be n+1 to allow the program to close the polygon (see step f).

e) Read in the x,y-coordinates from the data file and assign them to the arrays called x and y. Use the following "for loop" so that the subscript for these arrays (as defined by variable i) is automatically increased (or "incremented") by a value of 1 each time a new coordinate pair is read in.

```
FOR i = 1 to n
        INPUT #1, x(i), y(i)
NEXT
```

f) Close the polygon by making the last coordinate pair the same as the first, as follows:

```
x(n+1) = x(1)
y(n+1) = y(1)
```

g) Include a line containing a DO statement. This will begin a "do loop" that will be used when prompting the user to enter the x,y-coordinates of a point from the keyboard.

h) Read in the x,y-coordinates of a point from the keyboard. This will be the point for which the point-in-polygon test will be performed. Use a statement something like the following:

INPUT "Enter x-coordinate: ", xpt

This will read in the x-coordinate of the point and assign it to a variable called xpt. Include a similar line to read the y-coordinate and assign it to a variable called ypt.

Lines 5000 through 7700 perform the point-in-polygon test and print the results out to the screen. After line 7700, you will need to include a few more lines.

i) Include a mechanism for performing the point-in-polygon test on another point, should the user want to do that. Use a PRINT statement to print a message asking the user to enter an n (for "no") if another test is not desired. Then include the line:

IF INPUT$(1) = "n" THEN EXIT DO

This will cause the program to exit the do loop should the user enter an n. If any other character is entered, the program will return to the DO statement (see step g).

j) The last two lines of the program should contain LOOP and END statements, respectively. The first of these two lines ends the do loop and the second ends the program.

Now that you have made these modifications, save the program. Run it to determine whether each of the points listed in Table 1 is inside or outside of the polygon. Once the program is running to your satisfaction, make a printout to hand in along with the assignment.

Table 1.

Point	x-coordinate	y-coordinate	Inside or outside?
A	25	100	
B	101	299.7	
C	631	246	
D	387.5	224	
E	97	401	
F	544.1	77	
G	321.3	314	
H	111	49	
I	251	327	
J	118	176	

Area Estimation: A common method of estimating the area of a polygon is to overlay a grid of dots with a known average density and count the number of "hits" (i.e., the number of dots falling within the polygon). The area of the polygon is estimated as the number of hits divided by the average dot density. The area estimate improves as dot density increases.

The program called DARTS.BAS calculates the area of a polygon based on this approach. Open this program and examine it. Note that it reads in the same polygon data you used for the PIP.BAS program.

Lines 340 and 345 of the program are used to generate a point with random x,y-coordinates. The x-coordinate ranges from 0 to 639, and the y-coordinate ranges from 0 to 462. These values reflect the approximate size of the screen, measured in pixels. Lines 510 through 670 should look familiar. This is the same point-in-polygon algorithm used in PIP.BAS. The algorithm is used to determine whether the randomly-generated point is inside or outside the polygon. If it is outside the polygon, the program draws the point as a grey dot (line 820). If it is inside the polygon, the program draws the point as a red dot (line 930). This process is repeated until the user presses a key to temporarily suspend program execution (lines 2000 to the end of the program). After several thousand "trials" (one trial is equal to one random point), the program should give a fairly accurate estimate of polygon area.

Lines 1100 through 1170 are used to print out several statistics -- the number of trials, the number of hits, the number of hits as a percentage of the number of trials, and the estimate of polygon area. These statistics are printed out on the lower left of the screen. Run the program and monitor the statistics printed on the screen, with the goal of filling in Table 2. Press any key to make the program pause temporarily. Perform as many trials as you need in order to identify the shape of the polygon appearing in red on the screen.

1. a) What is the shape of the polygon? b) How many trials are needed to identify this shape?

On a piece of graph paper, graph the estimate of polygon area as a function of the number of trials.

2. a) Does the area estimate appear to be stabilizing as the number of trials increases? b) What is the best estimate of the area of the polygon?

Table 2.

Number of trials	Number of hits	Number of hits as a percentage of number of trials	Polygon area estimate (in pixels)
500			
1000			
1500			
2000			
2500			
3000			
3500			
.			
.			
.			
.			
.			
.			

Other Area Estimates: The area of a polygon can also be calculated from the area of a set of trapezoids defined by the x,y-coordinate pairs of the polygon. Write a QuickBASIC program that implements this calculation. This program will be very similar to of PIP.BAS. Your program should:

a) clear the screen;

b) open the polygon data file called POLYGON.DAT for reading;

c) read in the number on the first line of the data file (the number of x,y-coordinate pairs) and assign it to a variable called n;

d) dimension two arrays (called x and y) for storing the x,y-coordinates from the file (the dimension of the arrays should be n+1);

e) read in the x- and y-coordinates from the file and store these in the two arrays using a for loop;

f) close the polygon;

g) initialize a variable called area (the polygon area estimate) to zero;

h) use the following FOR loop to calculate the area of the polygon

```
FOR  i  =  1  to  n
        area  =  area + (x(i+1) - x(i)) * (y(i+1) + y(i)) / 2
NEXT
```

i) print out the area of the polygon.

Be sure to include comments describing the function of each program section. Run the program to obtain the polygon area estimate. Make a printout of the program to hand in along with the assignment.

3. What is the area estimate you obtained with your program?

On the graph you constructed previously, draw a horizontal line representing the area estimate obtained with your program. Hand in this graph along with your assignment.

4. How does the area estimate you obtained with your program compare to the estimates you obtained with DARTS.BAS?

Modify the program to read data from POLYGON2.DAT rather than POLYGON.DAT. These two data files are identical except that the coordinate pairs in POLYGON.DAT are arranged in clockwise order, while those in POLYGON2.DAT are in counter-clockwise order.

5. a) What is the area estimate you obtained for POLYGON2.DAT? b) How does this estimate compare to that obtained for POLYGON.DAT?

Line Intersection: The program called INTER.BAS finds the intersection of two straight line segments. Run the program to compute the point of intersection for the examples listed in Table 3.

Table 3.

		First line					Second line					
		First end point		Second end point			First end point		Second end point		Point of intersection	
Example		x	y	x	y		x	y	x	y	x	y
A		100	100	300	300		300	100	100	300		
B		400	100	400	300		10	250	600	250		
C		400	100	400	300		400	150	500	150		
D		400	100	400	300		450	150	500	150		
E		100	100	300	300		200	200	400	400		
	F	400	100	200	200	300	150	250	175			
	G	100	100	300	300	300	300	100	100			

6. Examples E, F and G represent a special case of intersection that the program cannot handle. a) Explain what this special case is. b) Explain how E, F and G are each slightly different examples of this special case.

Now open the program called FRACTAL.BAS. You will be using this program to find intersection points for lines composed of multiple straight-line segments. The program first draws a straight line (in blue) across the screen. Then it draws a wiggly line (in green) using the fractal concept. The coordinates of the fractal line are determined randomly, so each time you run the program a different line will be drawn.

Note that the end points of the straight and fractal lines are the same. Also note that the arrays for storing the x,y-coordinates for the fractal line (i.e., x2 and y2) are now dimensioned at 65. This is because the fractal line is composed of 64 individual straight-line segments (65 coordinate pairs).

The variable called w (line 1000) defines the "wiggliness" of the fractal line. Run the program with different values of w and observe how the line changes.

7. a) What is the effect of increasing the value of w? b) What is the effect of decreasing w? c) What happens when w is zero?

Modify the program so that it calculates and displays the intersections between the straight and fractal lines. The easiest way to do this is to insert a modified version of the line intersection algorithm between lines 4100 and 20000 of the program. To do this, open INTER.BAS, highlight lines 4500 to 11200, and select the Copy option from the Edit menu. Now open FRACTAL.BAS, click on line 20000, and select the Paste option from the Edit menu. This procedure will copy the line intersection algorithm from INTER.BAS into FRACTAL.BAS.

Now make the following modifications to the algorithm:

a) Change all occurrences of x2(1) to x2(k), and all occurrences of x2(2) to x2(k+1). Likewise, change all occurrences of y2(1) to y2(k), and all occurrences of y2(2) to y2(k+1).

b) Replace line 10500 with the following:

```
CIRCLE ( xi, yi ), 5, 15
```

This will draw a white circle centered on each intersection point.

c) Delete the PRINT statements in lines 10200 and 11100, and the GOTO statement on line 10600. Do a bit of cleaning up between lines 10100 and 11200, since some of the IF statements are no longer required now that the associated PRINT statements have been deleted.

Save the program and run it several times to generate different fractal lines. Hand in a printout of the program along with your assignment.

8. Explain the rationale for modification a, above.

9. a) Does the program always manage to identify the intersections between the straight line and the fractal line? b) If not, can you explain why the program might be missing some intersections? c) Can you suggest how the program might be modified to fix this problem?

Assignment 2
Program Listing

```
100     'PIP.BAS

5000    'Perform  point-in-polygon  test.

5100    in = 1
5200    FOR i = 1 TO n
5300        IF x(i + 1) <> x(i) THEN
5400            IF (x(i + 1) - xpt) * (xpt - x(i)) >= 0 THEN
5500                IF x(i + 1) <> xpt OR x(i) >= xpt THEN
5600                    IF x(i) <> xpt OR x(i + 1) >= xpt THEN
5700                        b = (y(i + 1) - y(i)) / (x(i + 1) - x(i))
5800                        a = y(i) - b * x(i)
5900                        yi = a + b * xpt
6000                        IF yi > ypt THEN
6100                            in = in * -1
6200                        END IF
6300                    END IF
6400                END IF
6500            END IF
6600        END IF
6700    NEXT

7000    'Print  results.

7100    PRINT
7200    IF in = -1 THEN
7300        PRINT "That point is INSIDE the polygon."
7400    ELSE
7500        PRINT "That point is OUTSIDE the polygon."
7600    END IF
7700    PRINT
```

```
100     'DARTS.BAS

110     'This program estimates the area of a polygon based on the
120     'probability of a randomly-generated point falling within it.

130     CLS
135     SCREEN 12

140     'Open the data file for reading.

145     OPEN "POLYGON.DAT" FOR INPUT AS #1

150     'Read in the number of points defining the outline of the polygon.

160     INPUT #1, n

165     'Dimension the arrays for the x,y-coordinates of the points.
175     'The dimension should be n+1 to allow the polygon to be closed.

180     DIM x(n + 1), y(n + 1)

185     'Read in the x,y-coordinates of the points.

190     FOR i = 1 TO n
200         INPUT #1, x(i), y(i)
205     NEXT

210     'Close the polygon.

215     x(n + 1) = x(1)
220     y(n + 1) = y(1)

225     'Draw a rectangle on the screen.  (The dimensions of the box define
230     'the maximum and minimum coordinates of the random points.)

235     LINE (0, 0)-(639, 462), 7, B

240     'num = the number of trials; nin is the number of hits.
250     'Random generation of points is based on internal clock.

255     num = 0
260     nin = 0
265     RANDOMIZE (TIMER)
320     LOCATE 30, 1
325     PRINT "Hit any key to pause.                    ";
```

```
330     'Generate random x,y-coordinates for a point in the rectangle.

340     xpt = RND * 639
345     ypt = RND * 462

400     'Increase the number of trials by one.

410     num = num + 1

420     'Statistics to print on screen...

425     LOCATE 25, 2
430     PRINT "Trials: ";
435     LOCATE 26, 2
440     PRINT "Hits: ";
445     LOCATE 27, 2
450     PRINT "Percent: ";
455     LOCATE 28, 2
460     PRINT "Area: ";

500     'Point-in-polygon algorithm determines whether
505     'random point is in the polygon (a hit) or not.

510     in = 1
520     FOR i = 1 TO n
530         IF x(i + 1) <> x(i) THEN
540             IF (x(i + 1) - xpt) * (xpt - x(i)) >= 0 THEN
550                 IF x(i + 1) <> xpt OR x(i) >= xpt THEN
560                     IF x(i) <> xpt OR x(i + 1) >= xpt THEN
570                         b = (y(i + 1) - y(i)) / (x(i + 1) - x(i))
580                         a = y(i) - b * x(i)
590                         yi = a + b * xpt
600                         IF yi > ypt THEN
610                             in = in * -1
620                         END IF
630                     END IF
640                 END IF
650             END IF
660         END IF
670     NEXT

800     'If the point is NOT in the polygon, draw a grey dot.

810     IF in = 1 THEN
820         PSET (xpt, ypt), 7
830     END IF
```

```
900     'If the point IS in the polygon, draw a red dot
910     'and increase the number of hits (nin) by 1.

920     IF in = -1 THEN
930         PSET (xpt, ypt), 12
940         nin = nin + 1
950     END IF

1000    'Print out statistics.

1100    LOCATE 25, 11
1110    PRINT num;
1120    LOCATE 26, 11
1130    PRINT nin;
1140    LOCATE 27, 11
1150    PRINT nin / num * 100;
1160    LOCATE 28, 11
1170    PRINT nin / num * 640 * 463;

2000    'Monitor keyboard events to suspend or terminate program.
2010    'If no key is pressed, get another random point...

2100    a$ = INKEY$
2200    IF LEN(a$) = 0 THEN 340

2300    '...otherwise, pause.

2400    LOCATE 30, 1
2500    PRINT "Hit q to quit or any other key to resume.";

2600    'If no other key is pressed, keep waiting...

2700    a$ = INKEY$
2800    IF LEN(a$) = 0 THEN 2700

2900    '...or, if a q is not pressed, get another random point...

3000    IF a$ <> "q" THEN 320

3100    '...otherwise, terminate the program.

5200    END
```

```
100   'INTER.BAS

110   'This program finds the point of intersection
120   'for two straight-line segments.

150   'Dimension the arrays to store the x,y-coordinates of the end
160   'points of the two lines.  Since we are dealing with straight line
170   'segments, the arrays only need to contain two elements.
180   'The arrays called x1 and y1 store the x,y-coordinates
190   'for the end points of the first line segment.
200   'The arrays called x2 and y2 store the x,y-coordinates
210   'for the end points of the second line segment.

250   DIM x1(2), y1(2)
260   DIM x2(2), y2(2)

300   'Begin loop to read in coordinates of end points.

400   CLS
410   DO

505   PRINT
510   PRINT "Enter the x- and y-coordinates of the end points of each line."
520   PRINT "Separate the coordinates with a comma."
530   PRINT
600   INPUT "First end point of first line: ", x1(1), y1(1)
610   INPUT "Second end point of first line: ", x1(2), y1(2)
620   INPUT "First end point of second line: ", x2(1), y2(1)
630   INPUT "Second end point of second line: ", x2(2), y2(2)
640   PRINT

4500  'Initialize flag that will tell us if intersection occurs.

4600  intersect = 1

5000  'Find intersection point. Variables xi and yi
5010  'define x,y-coordinates of the intersection point.

5100  IF x1(1) <> x1(2) THEN
5200      b1 = (y1(2) - y1(1)) / (x1(2) - x1(1))
5300      IF x2(1) <> x2(2) THEN
5400          b2 = (y2(2) - y2(1)) / (x2(2) - x2(1))
5500          a1 = y1(1) - b1 * x1(1)
5600          a2 = y2(1) - b2 * x2(1)
5700          IF b1 = b2 THEN
5800              intersect = 0
```

```
5900            ELSE
6000                xi = -1 * (a1 - a2) / (b1 - b2)
6100                yi = a1 + b1 * xi
6200            END IF
6300        ELSE
6400            xi = x2(1)
6500            a1 = y1(1) - b1 * x1(1)
6600            yi = a1 + b1 * xi
6700        END IF
6800    ELSE
6900        xi = x1(1)
7000        IF x2(1) <> x2(2) THEN
7100            b2 = (y2(2) - y2(1)) / (x2(2) - x2(1))
7200            a2 = y2(1) - b2 * x2(1)
7300            yi = a2 + b2 * xi
7400        ELSE
7500            intersect = 0
7600        END IF
7700    END IF

9000    'Print results.

10100   IF intersect = 0 THEN
10200       PRINT "Lines do not intersect."
10300   ELSE
10400       IF (x1(1) - xi) * (xi - x1(2)) >= 0 THEN
10410           IF (x2(1) - xi) * (xi - x2(2)) >= 0 THEN
10420               IF (y1(1) - yi) * (yi - y1(2)) >= 0 THEN
10430                   IF (y2(1) - yi) * (yi - y2(2)) >= 0 THEN
10500                       PRINT "Lines intersect at ("; xi; ", "; yi; ")"
10600                       GOTO 14100
10700                   END IF
10800               END IF
10900           END IF
11000       END IF
11100       PRINT "Lines do not intersect."
11200   END IF

14000   'Find intersection for another pair of line segments?

14100   PRINT "Find intersection for another pair of line segments? [y/n]"
14200   IF INPUT$(1) = "n" THEN EXIT DO

15000   LOOP

20000   END
```

```
100     'FRACTAL.BAS

200     'This program draws a straight blue line
300     'and a wiggly green "fractal" line.

500     CLS
600     SCREEN 12

700     'Dimension the arrays x1 and y1 for storing the end points of the
720     'straight line (hence dimension of 2).  Arrays x2 and y2 are for
730     'the end points of the fractal line.  The dimension in this case is 65,
740     'as we want 64 segments in the fractal line.  (Note that we ignore
750     'the 0th element of all arrays to avoid confusion.)

800     DIM x1(2), y1(2)
810     DIM x2(65), y2(65)

890     'Random number generation is based on internal clock.

900     RANDOMIZE (TIMER)

990     'Variable called w determines wiggliness of the fractal line.

1000    w = 200

1090    'Define coordinates of end points of straight line.

1100    x1(1) = 20
1110    y1(1) = 250
1120    x1(2) = 620
1130    y1(2) = 250

1140    'Draw the line in blue.

1150    LINE (x1(1), y1(1))-(x1(2), y1(2)), 1

1190    'End points of fractal line are the same as those of straight line.

1200    x2(1) = 20
1210    y2(1) = 250
1220    x2(65) = 620
1230    y2(65) = 250
```

```
1990    'Generate fractal line.

2000    FOR k = 1 TO 6
2100        ka = 64 / 2 ^ k
2200        FOR j = (ka + 1) TO 65 STEP (2 * ka)
2300            x2(j) = (x2(j - ka) + x2(j + ka)) / 2
2310            x2(j) = x2(j) + (1 - 2 * RND) * w / 2 ^ k
2400            y2(j) = (y2(j - ka) + y2(j + ka)) / 2
2410            y2(j) = y2(j) + (1 - 2 * RND) * w / 2 ^ k
2500        NEXT
2600    NEXT

3990    'Draw fractal line in green.

4000    FOR k = 1 TO 64
4100        LINE (x2(k), y2(k))-(x2(k + 1), y2(k + 1)), 2
20000   NEXT

30000   END
```

Assignment 2
Answer Key

Table 1.

Point	x-coordinate	y-coordinate	Inside or outside?
A	25	100	out
B	101	299.7	out
C	631	246	out
D	387.5	224	in
E	97	401	out
F	544.1	77	out
G	321.3	314	in
H	111	49	out
I	251	327	in
J	118	176	in

1. **a)** A maple leaf. **b)** Several thousand at minimum.

2. **a)** Depends on the maximum number of trials performed. **b)** The best estimate is the one for the largest number of trials.

3. Estimate is 90,919.13 pixels.

4. Estimate is very similar to the previous ones.

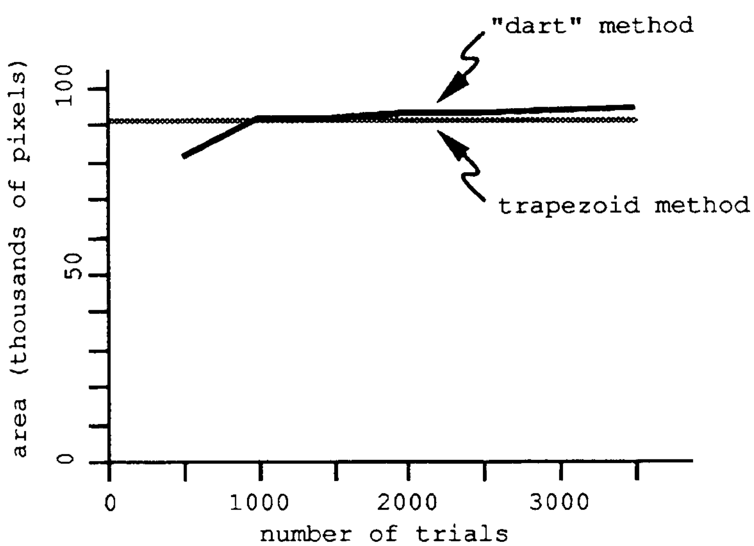
5. **a)** Estimate is -90,919.13 pixels. **b)** Negative of other estimate.

6. **a)** They are all instances of coincident lines. **b)** In E, the lines share a portion of the same line. In F, one line is a subset of the other line. In G, the lines are identical.

7. **a)** Increasing w makes the line more wiggly. **b)** Decreasing w makes the line less wiggly. **c)** When w is zero, the line is straight.

Table 2.
(NOTE: Students' answers will vary slightly.)

Number of trials	Number of hits	Number of hits as a percentage of number of trials	Polygon area estimate (in pixels)
500	137	27.35	81 029.62
1000	312	31.08	92 083.51
1500	468	30.97	91 778.80
2000	628	31.42	93 091.02
2500	785	31.38	92 858.77
3000	945	31.49	93 309.70
3500	1100	31.69	93 888.35



8. The program needs to find the intersections between the straight line and all 64 pieces of the fractal line. Each piece is referenced with a value of k, which is incremented in the loop.

9. **a)** No. **b)** Rounding error means that line 10400 sometimes evaluates as false when it is true. **c)** Change the zeros in line 10400 to a smaller value to implement a fuzzy tolerance. (A value of -1 seems to work.)

Table 3.

	First line				Second line					
	First end point		Second end point		First end point		Second end point		Point of intersection	
Example	x	y	x	y	x	y	x	y	x	y
A	100	100	300	300	300	100	100	300	200	200
B	400	100	400	300	10	250	600	250	400	250
C	400	100	400	300	400	150	500	150	400	150
D	400	100	400	300	450	150	500	150	-- do not intersect --	
E	100	100	300	300	200	200	400	400	-- do not intersect --	
F	400	100	200	200	300	150	250	175	-- do not intersect --	
G	100	100	300	300	300	300	100	100	-- do not intersect --	

Assignment 2
Program Listing (Answer Key)

```
100     'PIP1.BAS

200     'Point-in-polygon  program.

300     CLS

1000    'Open the data file for reading.

1100    OPEN "POLYGON.DAT" FOR INPUT AS #1

1200    'Read in the number of points in the polygon.
1300    '(This is the first line of POLYGON.DAT).

1400    INPUT #1, n

2000    'Dimension the arrays for storing the x,y-coordinates
2100    'of the polygon.  Dimension should be one greater than n
2200    'to allow the program to close the polygon.

2300    DIM x(n + 1), y(n + 1)

3100    'Read in the x,y-coordinates for the n points.

3200    FOR i = 1 TO n
3300        INPUT #1, x(i), y(i)
3400    NEXT

3500    'Close the polygon.

3600    x(n + 1) = x(1)
3700    y(n + 1) = y(1)

4000    'Begin loop for reading in x- and y-coordinates
4100    'of points and performing the test.

4200    DO

4300    'Read in the x- and y-coordinates of the point to test.

4400    INPUT "Enter x-coordinate of the point: ", xpt
4500    INPUT "Enter y-coordinate of the point: ", ypt
```

```
5000    'Perform point-in-polygon test.

5100    in = 1
5200    FOR i = 1 TO n
5300        IF x(i + 1) <> x(i) THEN
5400            IF (x(i + 1) - xpt) * (xpt - x(i)) >= 0 THEN
5500                IF x(i + 1) <> xpt OR x(i) >= xpt THEN
5600                    IF x(i) <> xpt OR x(i + 1) >= xpt THEN
5700                        b = (y(i + 1) - y(i)) / (x(i + 1) - x(i))
5800                        a = y(i) - b * x(i)
5900                        yi = a + b * xpt
6000                        IF yi > ypt THEN
6100                            in = in * -1
6200                        END IF
6300                    END IF
6400                END IF
6500            END IF
6600        END IF
6700    NEXT

7000    'Print results.

7100    PRINT
7200    IF in = -1 THEN
7300        PRINT "That point is INSIDE the polygon."
7400    ELSE
7500        PRINT "That point is OUTSIDE the polygon."
7600    END IF
7700    PRINT

8000    'Perform point in polygon test for another point?

8100    PRINT "Perform the test on another point? [y/n]"
8200    IF INPUT$(1) = "n" THEN EXIT DO

9000    LOOP

9100    END
```

```
100    'AREA.BAS

200    'Program to compute area of a polygon based on trapezoid method.

300    CLS

1000   'Open the data file for reading.

1100   OPEN "POLYGON.DAT" FOR INPUT AS #1

1200   'Read in the number of points in the polygon.
1300   '(This is the first line of POLYGON.DAT).

1400   INPUT #1, n

2000   'Dimension the arrays for storing the x,y-coordinates
2100   'of the polygon.  Dimension should be one greater than n
2200   'to allow the program to close the polygon.

2300   DIM x(n + 1), y(n + 1)

3100   'Read in the x,y-oordinates for the n points.

3200   FOR i = 1 TO n
3300        INPUT #1, x(i), y(i)
3400   NEXT

3500   'Close the polygon.

3600   x(n + 1) = x(1)
3700   y(n + 1) = y(1)

4000   'Initialize area to zero.

4200   area = 0

4300   'Calculate and print out the area of the polygon.

4400   FOR i = 1 to n
4500        area = area + (x(i + 1) - x(i)) * (y(i + 1) + y(i)) / 2
4600   NEXT
4700   PRINT "The polygon has an area of ", area, "pixels."

5000   END
```

```
100     'FRACTAL1.BAS

200     'This program draws a straight blue line
300     'and a wiggly green "fractal" line.
400     'Circles are drawn at points where the two lines intersect.

500     CLS
600     SCREEN 12

700     'Dimension the arrays x1 and y1 for storing the end points of the
720     'straight line (hence dimension of 2). Arrays x2 and y2 are for
730     'the end points of the fractal line. The dimension in this case is 65,
740     'as we want 64 segments in the fractal line. (Note that we ignore
750     'the 0th element of all arrays to avoid confusion.)

800     DIM x1(2), y1(2)
810     DIM x2(65), y2(65)

890     'Random number generation is based on internal clock.

900     RANDOMIZE (TIMER)

990     'Variable called w determines wiggliness of the fractal line.

1000    w = 200

1090    'Define coordinates of end points of straight line.

1100    x1(1) = 20
1110    y1(1) = 250
1120    x1(2) = 620
1130    y1(2) = 250

1140    'Draw the line in blue.

1150    LINE (x1(1), y1(1))-(x1(2), y1(2)), 1

1190    'End points of fractal line are the same as those of straight line.

1200    x2(1) = 20
1210    y2(1) = 250
1220    x2(65) = 620
1230    y2(65) = 250
```

```
1990    'Generate fractal line.

2000    FOR k = 1 TO 6
2100        ka = 64 / 2 ^ k
2200        FOR j = (ka + 1) TO 65 STEP (2 * ka)
2300            x2(j) = (x2(j - ka) + x2(j + ka)) / 2
2310            x2(j) = x2(j) + (1 - 2 * RND) * w / 2 ^ k
2400            y2(j) = (y2(j - ka) + y2(j + ka)) / 2
2410            y2(j) = y2(j) + (1 - 2 * RND) * w / 2 ^ k
2500        NEXT
2600    NEXT

3990    'Draw fractal line in green.

4000    FOR k = 1 TO 64
4100        LINE (x2(k), y2(k))-(x2(k + 1), y2(k + 1)), 2

4500    'Initialize a flag that will tell us when intersection occurs.

4600    intersect = 1

5000    'Find intersection point. Variables xi and yi define
5010    'x,y-coordinates of the intersection point.

5100    IF x1(1) <> x1(2) THEN
5200        b1 = (y1(2) - y1(1)) / (x1(2) - x1(1))
5300        IF x2(k) <> x2(k + 1) THEN
5400            b2 = (y2(k + 1) - y2(k)) / (x2(k + 1) - x2(k))
5500            a1 = y1(1) - b1 * x1(1)
5600            a2 = y2(k) - b2 * x2(k)
5700            IF b1 = b2 THEN
5800                intersect = 0
5900            ELSE
6000                xi = -1 * (a1 - a2) / (b1 - b2)
6100                yi = a1 + b1 * xi
6200            END IF
6300        ELSE
6400            xi = x2(k)
6500            a1 = y1(1) - b1 * x1(1)
6600            yi = a1 + b1 * xi
6700        END IF
6800    ELSE
6900        xi = x1(1)
7000        IF x2(k) <> x2(k + 1) THEN
7100            b2 = (y2(k + 1) - y2(k)) / (x2(k + 1) - x2(k))
7200            a2 = y2(k) - b2 * x2(k)
```

```
7300                yi = a2 + b2 * xi
7400            ELSE
7500                intersect = 0
7600            END IF
7700        END IF

9000        'Draw circles at intersection points.

10100       IF intersect <> 0 THEN
10400           IF (x1(1) - xi) * (xi - x1(2)) >= 0 THEN
10410               IF (x2(1) - xi) * (xi - x2(2)) >= 0 THEN
10420                   IF (y1(1) - yi) * (yi - y1(2)) >= 0 THEN
10430                       IF (y2(1) - yi) * (yi - y2(2)) >= 0 THEN
10500                           CIRCLE (xi, yi), 5, 15
10700                       END IF
10800                   END IF
10900               END IF
11000           END IF
11200       END IF

20000       NEXT

30000       END
```

Assignment 3
Vector Data Structures (II)

Objectives: In this assignment you will be using pcARC/INFO to examine different types of topological overlay.

Files: This assignment requires part of the pcARC/INFO Green River database. You will need the following coverages:

DEVELOP a polygon coverage of areas selected for development

SITES a polygon coverage of ecologically sensitive areas

Examining the Database: Use ARCPLOT to display the two coverages on the screen and examine the features that they contain.

1. Draw a sketch map of the polygons displayed on the screen. Using the IDENTIFY command, label each polygon on your map with the appropriate SITES-ID or DEVELOP-ID code.

In TABLES, select the PAT file for each coverage in turn.

2. What items are contained in SITES.PAT?

3. List the area of each polygon in the SITES coverage along with its SITES-ID code. (Ignore the external polygon.)

4. What items are contained in DEVELOP.PAT?

5. List the area of each polygon in the DEVELOP coverage along with its DEVELOP -ID code.

Topological Overlay: pcARC/INFO provides five different types of topological overlay. The commands are listed below. Each command corresponds to a different combination of Boolean (or logical) operators (i.e., AND, OR and NOT). Each command creates a new output coverage from two existing input coverages.

UNION INTERSECT IDENTITY CLIP ERASECOV

6. Draw a set of sketch maps showing the polygons that would be produced if SITES and DEVELOP were used as the input coverages to each of the five overlay commands.

Perform all five types of overlay using SITES as the first input coverage (i.e., the "in_cover") and DEVELOP as the second input coverage (i.e., the "union_cover", "intersect_cover", etc). Give the output coverage a different name in each case.

Display each of the new coverages in ARCPLOT using the POLYGONSHADES and ARCS commands.

7. Draw a sketch map showing the polygons in each of the coverages. How do these compare to the sketch maps you produced in question 6?

Return to TABLES and select the PAT file for each of the new coverages in turn.

8. List the items contained in the PAT file for each coverage.

9. How are the PAT files for the coverages produced by CLIP and ERASECOV different from the files created with the other three

commands?

10. What items would be contained in these PAT files if you reversed the order of the first and second input coverages for CLIP and ERASECOV (i.e., you used DEVELOP as the "in_cover" and SITES as the "clip_cover" or "erase_cover").

11. Draw a sketch map showing the polygons that would be produced by CLIP and ERASECOV if you reversed the order of the input coverages.

12. Select the PAT file for the coverage produced by the UNION command. What is the total area of the polygons a) common to both SITES and DEVELOP b) found only in SITES c) found only in DEVELOP? (Assume that a value of 0 for any coverage-ID code indicates the external polygon for the associated coverage.)

13. Select the PAT file for the coverage produced by the INTERSECT command. a) What is the total area of the polygons in this coverage? b) How does this correspond to your answer for question 12?

14. Select the PAT file for the coverage produced by the IDENTITY command. What are the similarities and differences between this coverage the SITES coverage?

15. Select the PAT file for the coverage produced by the CLIP command. What are the similarities and differences between this coverage and the coverage produced by the INTERSECT command?

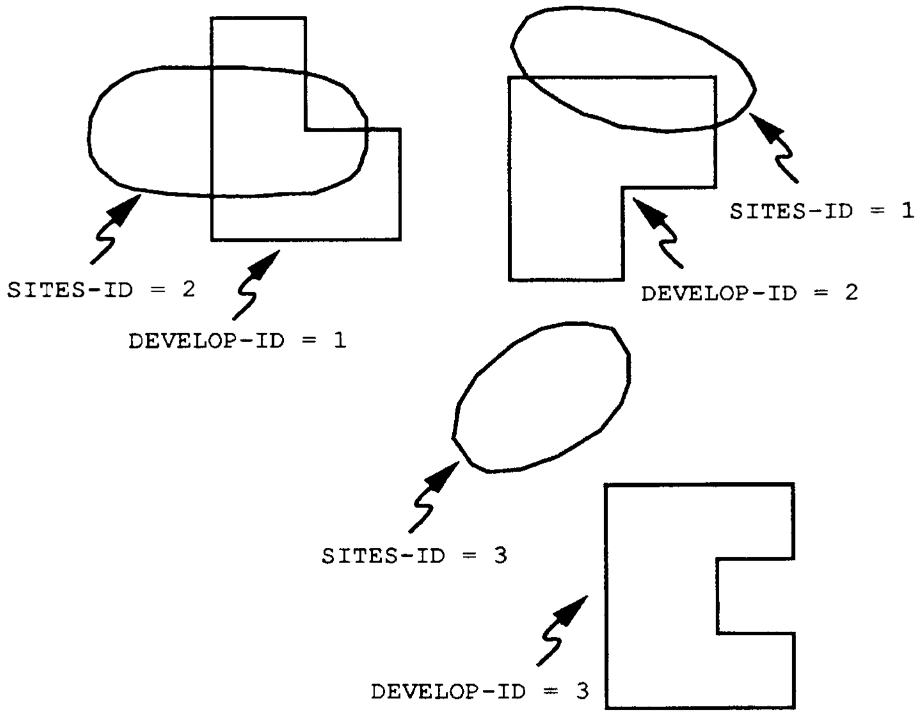
16. Select the PAT file for the coverage produced by the ERASECOV command. What are the similarities and differences between this coverage and the coverage produced by the CLIP command?

17. On each of the sketch maps you drew above (question 7), label the polygons in each of the five new coverages using their appropriate coverage-ID values.

18. Write out a Boolean expression that describes the polygons contained in each of the new coverages (e.g., SITES AND DEVELOP, SITES OR DEVELOP, SITES AND NOT DEVELOP, etc).

Assignment 3
Answer Key

1.



2. AREA, PERIMETER, SITES#, SITES-ID, NUMBER

3.

SITES-ID	AREA
1	134.522
2	164.198
3	141.235

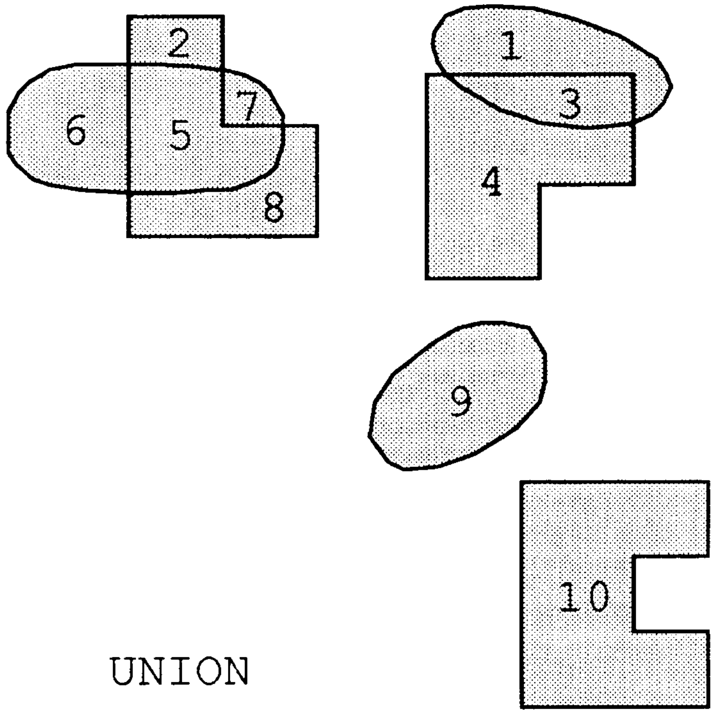
4. AREA, PERIMETER, DEVELOP#, DEVELOP-ID, NAME

5.

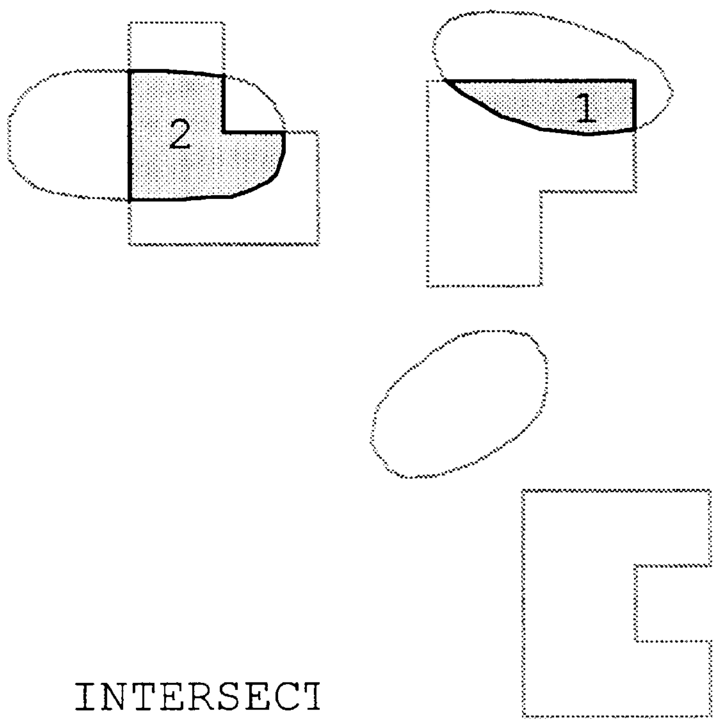
DEVELOP-ID	AREA
1	134.364
2	146.973
3	150.599

6. Answers will vary from student to student.

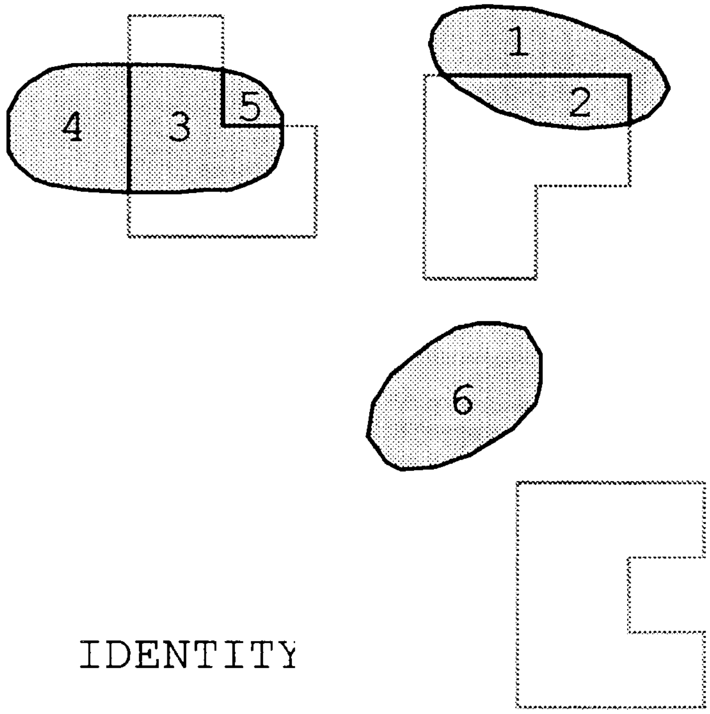
7 & 17.



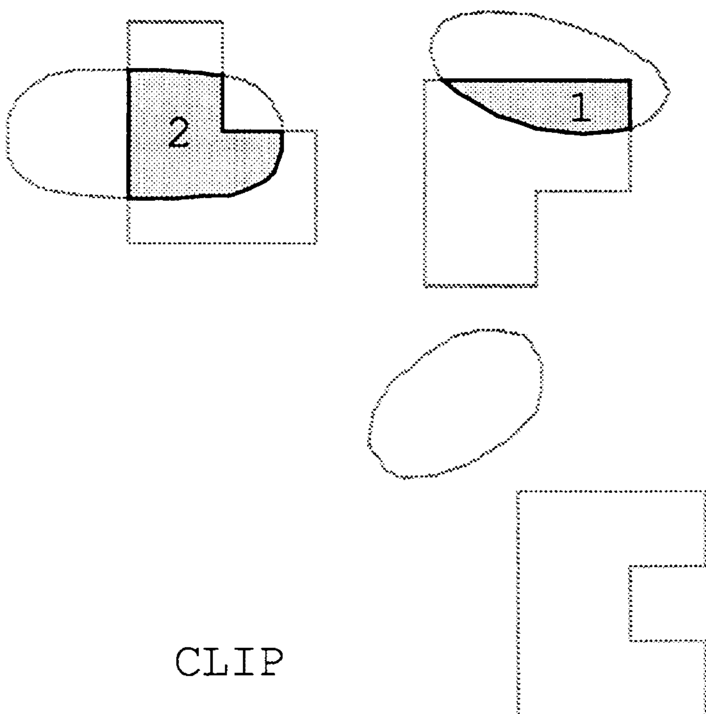
UNION



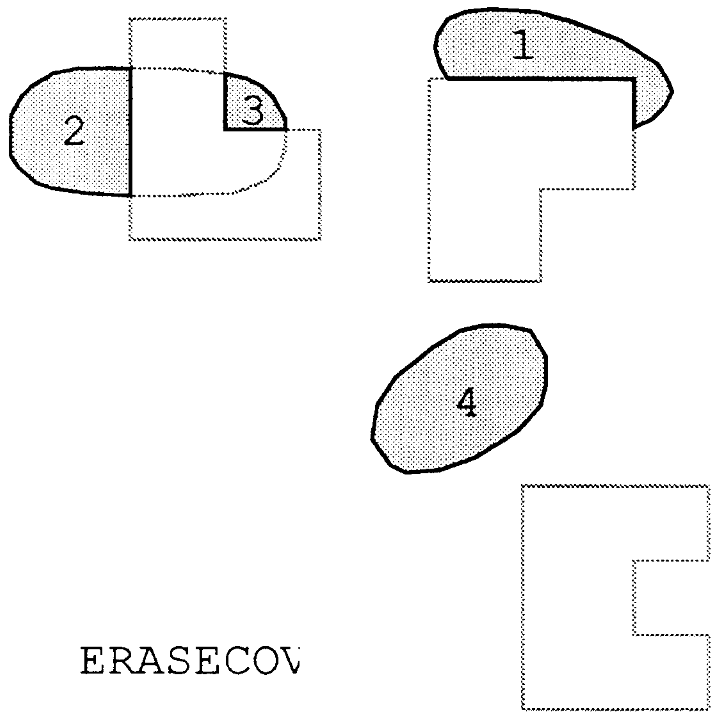
INTERSECT



IDENTITY



CLIP



8. **UNION:** AREA, PERIMETER, COV#, COV-ID, SITES#, SITES-ID, NUMBER, DEVELOP#, DEVELOP-ID, NAME

   INTERSECT: AREA, PERIMETER, COV#, COV-ID, SITES#, SITES-ID, NUMBER, DEVELOP#, DEVELOP-ID, NAME

   IDENTITY: AREA, PERIMETER, COV#, COV-ID, SITES#, SITES-ID, NUMBER, DEVELOP#, DEVELOP-ID, NAME

   CLIP: AREA, PERIMETER, COV#, COV-ID, NUMBER

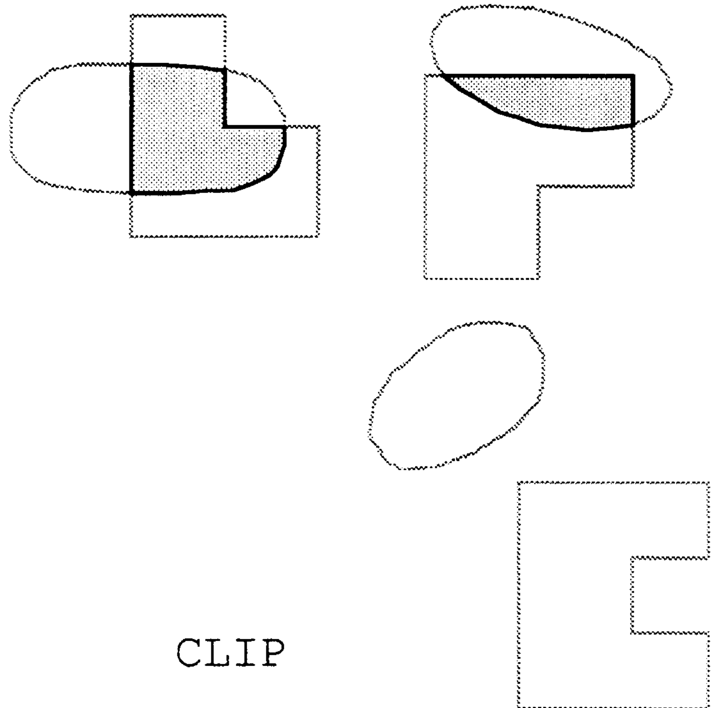
   ERASECOV: AREA, PERIMETER, COV#, COV-ID, NUMBER

9. They contain no items from DEVELOP.

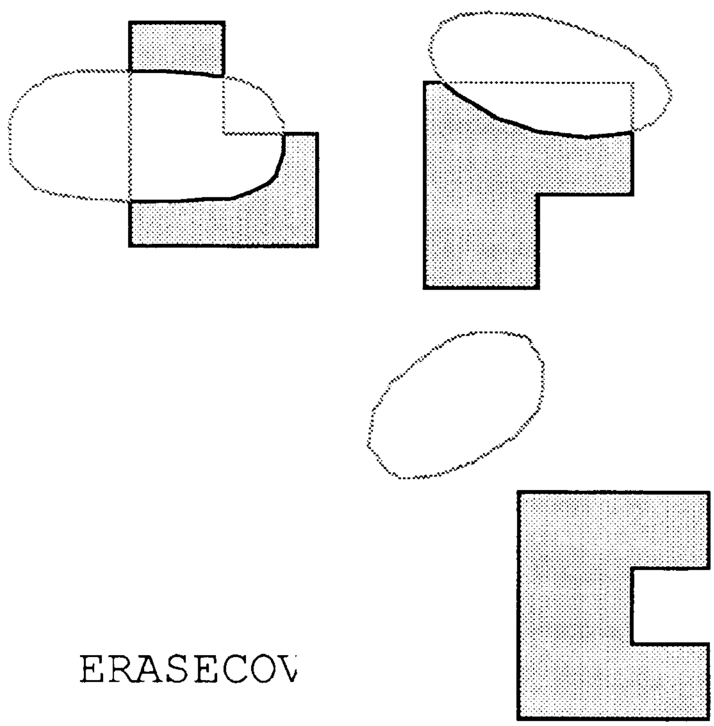
10. **CLIP:** AREA, PERIMETER, COV#, COV-ID, NAME

    ERASECOV: AREA, PERIMETER, COV#, COV-ID, NAME

11.



CLIP



ERASECOV

12. **a)** 33.828 + 69.791 = 103.619
b) 87.094 + 7.313 + 141.235 + 100.694 = 336.336
c) 16.132 + 113.144 + 48.440 + 150.599 = 328.315

13. **a)** 33.828 + 69.791 = 103.619
b) It is the same as 12 a). These are the polygons that SITES and DEVELOP have in common.

14. They have the same total area (i.e., the area of the polygons in SITES). However, in the coverage produced by IDENTITY, the SITES polygons intersecting DEVELOP polygons are now new polygons with attributes from DEVELOP.

15. They both contain the same set of two polygons (i.e., the polygons common to SITES and DEVELOP), but the coverage produced with CLIP has no attributes from DEVELOP.

16. Neither contains any attributes from DEVELOP, but in terms of the polygons retained the two coverages are mirror images of each other.

17. (See question 7.)

18.

UNION:	SITES OR DEVELOP
INTERSECT:	SITES AND DEVELOP
IDENTITY:	SITES OR (SITES AND DEVELOP)
CLIP:	SITES AND DEVELOP
ERASECOV:	SITES AND (NOT DEVELOP)

Assignment 4
Raster Data Structures

Objectives: In this assignment you will be writing a QuickBASIC program to perform run-length encoding on raster data.

Files: This assignment requires two QuickBASIC programs (MAP.BAS and MAPRUN.BAS) and three data files (ELEV.DAT, LAND.DAT and HAWAII.DAT).

Run-Length Encoding: The file called ELEV.DAT is a raster data file of elevations for Africa. Elevation values range from 2 (lowest elevation) to 6 (highest elevation). A value of 1 is used to designate water. The file called LAND.DAT is a raster data file of the same area in which water has been assigned a value of 0 and land a value of 1.

Each file contains 109 rows by 120 columns of cells. The cells are stored in "scan-line" order (i.e., beginning in the upper left corner and proceeding left to right along each row of cells).

Run the QuickBASIC program called MAP.BAS to display each file as a map.

Write a program to run-length encode each file in scan-line order. Each line in the output file should represent a "run" of cell values and should contain two numbers:

a) the length of the run, measured in cells, and

b) the cell value for the run.

The following example shows the output file that would be obtained by performing run-length encoding on an input file containing 4 rows by 5 columns of cells.

```
2 2 2 2 2            5 2
1 1 1 1 1            8 1
1 1 1 3 3    -->     6 3
3 3 3 3 2            1 2
```

Input file → **Output file**

Use your program to perform run-length encoding on the ELEV.DAT data file. Run the QuickBASIC program called MAPRUN.BAS to display a map of the output file. If your program is written correctly, the map should look the same as the one you produced earlier using MAP.BAS. Be sure that the last run of cells is displayed on your map.

Also perform run-length encoding on the LAND.DAT data file.

Hand in a copy of your run-length encoding program once you have it working to your satisfaction. Answer the following questions.

1. In DOS, use DIR to calculate the "compaction ratio" for each of the two output files. This ratio is calculated by dividing the size of the output file in bytes by the size of the corresponding original data file in bytes.

2. Which of the two files has a better compaction ratio? Why?

3. Under what circumstances might you get a compaction ratio greater than 1?

Optional: As the attached article indicates, it is possible to make maps from poems using the Morton cell ordering system. Write a program to perform this task. The data file called HAWAII.DAT contains the data necessary to make the map of Hawaii refered to in the article.

GEO-POESY

Howard Veregin
Department of Geography
University of California,
Santa Barbara, California, 93106

Introduction: Recent theoretical work suggests a strong link between geography and poetry. A simple computation allows any line of poetry to be positioned in two-dimensional space. The entire poem may be mapped as a set of points joined by straight-line segments. By implication, maps are poems and poems are maps.

Terminology: The link between geography and poetry may be explained with reference to prosody, the inexact science of linguistic rhythms. Essentially, a poem is a collection of lines, which are collections of words, which are in turn collections of syllables (Preminger 1986). Prosody shows that these syllables may be differentiated according to the presence or absence of stress. In poems, stress tends to recur in cyclical patterns. Analysis of these patterns is known as scansion, and the taxonomical characterization of patterns is based on meter. Meter is measured in terms of a unit known as a foot. Feet are defined by particular patterns of stressed and unstressed syllables. For instance, an iambic foot contains two syllables, the first unstressed and the second stressed, as in the following example taken from Chaucer:

∪ – ∪ – ∪ – ∪ – ∪ –
That slip / 'ry sci / ence stripped / me down / so bare

Following conventional symbology, a stressed syllable is denoted with a diacritical dash (-), an unstressed syllable with a diacritical cusp (H), a division between two feet with a virgule (/), and a division between two syllables with a hyphen (-) or with a blank space if the division occurs between two words.

In addition to iambic, there are three other common types of feet encountered in English-language poetry (Nims 1974). Trochaic foot is the transpose of iambic. Anapestic foot is characterized by two unstressed syllables followed by a stressed syllable, while in dactylic foot, a stressed syllable is followed by two unstressed syllables.

Meter also depends on the number of feet per line. In the iambic example presented above, there are 5 feet per line. This means that the line is denoted as I. pentameter. Similarly, a trochaic foot with 3 feet per line would be denoted as T. trimeter, an anapestic foot with 1 foot per line would be denoted as A. monometer, etc. Although the number of feet per line is always an integer, it is not uncommon for a poem to contain several feet that lack one or more syllables. Such fractional feet are called catalectic (Bain et al, 1981).

Mathematical Representation: The particular diacritical symbols used to denote whether a syllable is stressed or unstressed are wholly arbitrary. Thus one can safely replace these symbols with any others, so long as they are capable of differentiating between two discrete states. If one assigns a "1" to a stressed syllable and a "0" to an unstressed syllable, then any line of poetry can be represented by a string of bits (binary digits).

For example, the string for I. pentameter is

0101010101

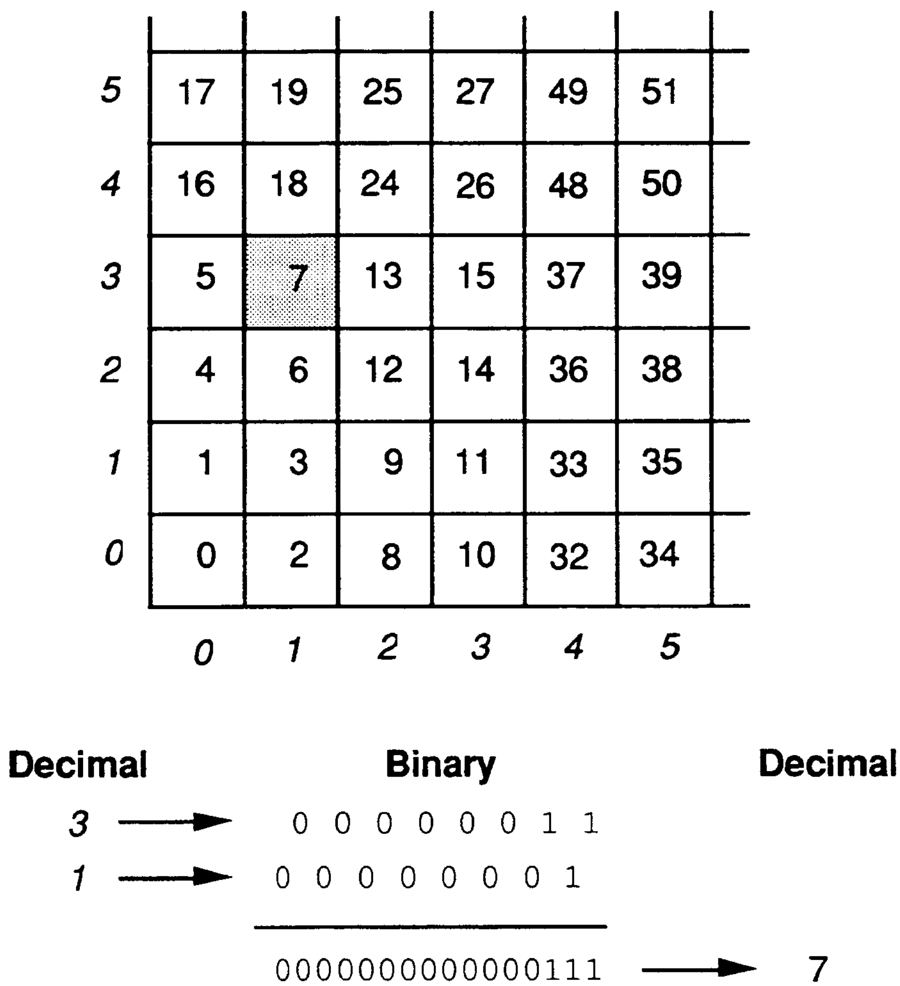
This bit string is the binary (i.e., base 2) representation of a number whose digital (i.e., base 10) representation happens to be 341. Any line of poetry may be represented mathematically in this manner.

The Link with Geography: These observations are closely linked to developments in modern geography pertaining to the tessellation of two-dimensional space. Tessellation refers to the partitioning of space into a set of regular, non-overlapping, spatially exhaustive cells. Typically these cells are square in shape, by other shapes may also be used.

While tessellation itself is a relatively straightforward concept, the optimal method of assigning index numbers to the cells is the subject of debate (Goodchild 1989). A host of different indexing systems have been proposed, but most of these have not been widely applied. The number of potential indexing systems is enormous. If there are r rows and c columns of cells, then there are rc! unique ways to arrange index numbers.

One indexing system that has proved useful, however, is that proposed by G. M. Morton (Morton, no date). In Morton's system, there is an implicit relationship between the location of a cell and its index number. More specifically, the row and column positions of the cell are embedded as interleaved bit strings in the binary representation of the cell index number. The figure below illustrates this relationship for cell number 7 in Morton two-space, for which the row and column positions are 3 and 1 respectively.

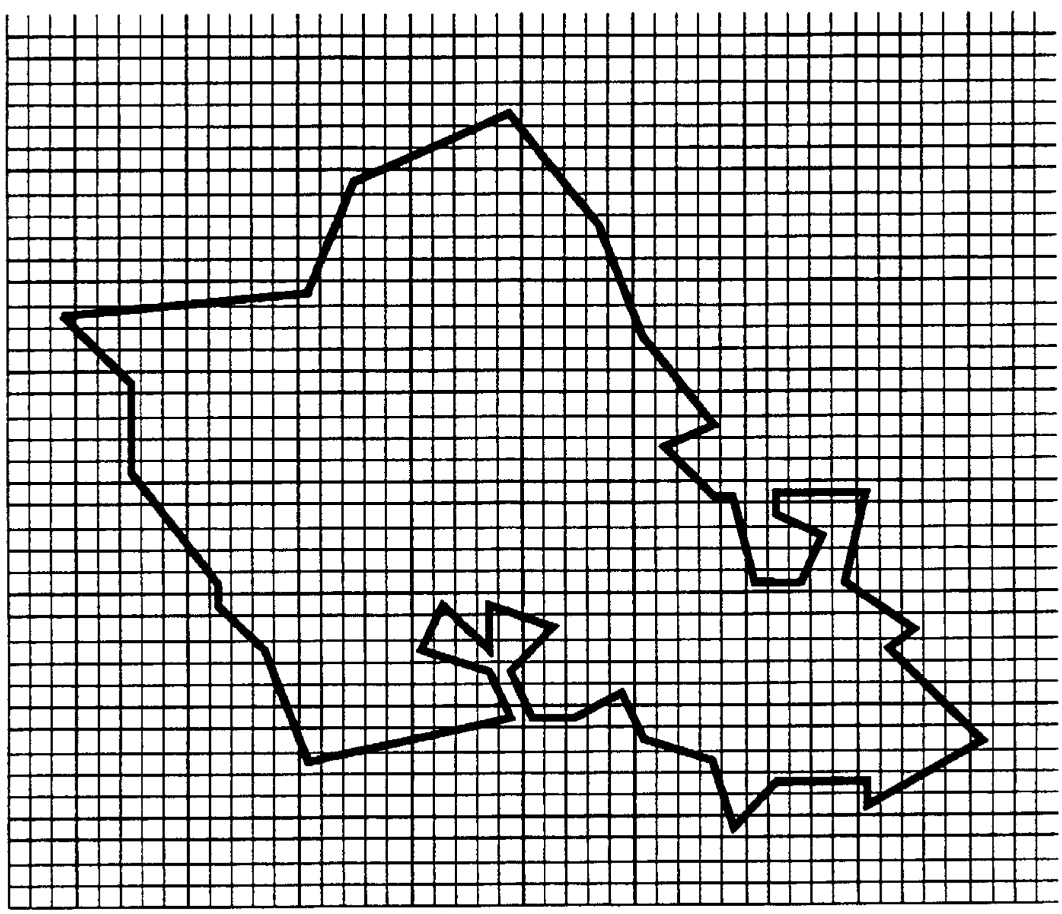
A corollary of this relationship is that any bit string is associated with one and only one cell in Morton two-space. Since any line of poetry can be represented as a bit string, it follows that any such line can be uniquely located in Morton two-space. Furthermore, since a poem is composed of one or more lines, it is possible to locate each line in Morton two-space, join these locations with straight-line segments, and call the resulting product a map.



An Illustration: As an illustration, consider J. Diefenbaker's poem, "The Scientist's Lament", the first six lines of which are reproduced below.

∪ ∪ ∪ – ∪ – ∪ ∪ – – ∪ ∪
I am a / sci-en-tist, / not a Frank / en-stein, Ed.
∪ ∪ ∪ – – – – ∪ ∪ – – –
I'm not an / old sewn-up / corpse with a / re-claimed head,
∪ – ∪ ∪ – ∪ – ∪ – ∪ – ∪
A flat-top / type hair-do, / bolts pro-trud / ing, skin sort
∪ – – ∪ ∪ ∪ – ∪ – – ∪ –
Of green-grey / (or is it / blue?), a big / hair-y wart,
∪ ∪ – – – – ∪ – – – ∪ ∪
And a pet / dog named Boo / Boo. But my / brain, it longs
∪ ∪ – – – – – ∪ ∪ ∪ ∪ –
For a cold, / old, dark, dank / grave. Hey, where / are my thongs?

The entire poem has been mapped into Morton two-space based on the principles described above. The result, as shown in the figure below, is an accurate map of Oahu, Hawaii. The first line of the poem is Kaena Point, the most westerly point on the island. The poem then proceeds in a clockwise direction.



References

Bain, C. E., Beaty, J. & Hunter, J. P. (Eds.) (1981). The Norton Introduction to Literature, 3rd Edition. New York: Norton.

Goodchild, M. F. (1989). Optimal tiling for large cartographic databases. Auto-Carto 9, 444-51.

Morton, G. M. (no date). A Computer Oriented Geodetic Data Base: With a New Technique in File Sequencing. (Unpublished manuscript).

Nims, J. F. (1974). Western Wind. New York: Random House.

Preminger, A. (Ed.) (1986). The Princeton Handbook of Poetic Terms. Princeton: Princeton University Press.

Assignment 4
Program Listing

```
100     'MAP.BAS -- a program to display a raster image on the screen.

190     'Choose data file, set up screen.

200     CLS
300     INPUT "Enter the name of the data file: ", filename$

400     SCREEN 12
500     WINDOW SCREEN (0, 0)-(160, 120)

600     'Input data from file and draw the map.

700     OPEN filename$ FOR INPUT AS #1

800     FOR y = 1 TO 109
810         FOR x = 1 TO 120
820             INPUT #1, z
830             LINE (x, y)-(x + 1, y + 1), z, BF
840         NEXT
850     NEXT

1000    END
```

```
100     'MAPRUN.BAS -- a program to display a run-encoded raster image.

190     'Choose data file, set up screen.

200     CLS
300     INPUT "Enter the name of the data file: ", filename$

400     SCREEN 12
500     WINDOW SCREEN (0, 0)-(160, 120)

600     'Input data from file.

700     OPEN filename$ FOR INPUT AS #1

800     'Hard-wire these guys.

810     nrows = 109
820     ncols = 120
830     ncells = nrows * ncols
840     DIM z(ncells)

900     'Unzip the run-encoded data.

910     endi = 0
920     DO UNTIL EOF(1)
930         starti = endi + 1
940         INPUT #1, frequency, value
950         endi = endi + frequency
960         FOR i = starti TO endi
970             z(i) = value
980         NEXT
990     LOOP

1000    'Display the image.

1010    FOR y = 1 TO nrows
1020        FOR x = 1 TO ncols
1030            cellno = (y - 1) * ncols + x
1040            LINE (x, y)-(x + 1, y + 1), z(cellno), BF
1050        NEXT
1060    NEXT

2000    END
```

Assignment 4
Answer Key

1. Different ratios are possible, depending on the format of the run-length encoded file. **LAND.DAT** and **ELEV.DAT** each contain 26,488 bytes. If a single space is left between the frequency and the cell value, the run-length encoded versions of these files contain 2300 bytes and 10,481 bytes, respectively. The corresponding compaction ratios are 0.087 and 0.396.

2. **LAND.DAT** yields a better compaction ratio since it has longer runs of the same value.

3. You might get a compaction ratio greater than 1 when there is high-frequency variation over space (i.e., short runs).

Assignment 4
Program Listing (Answer Key)

```
100    'RUN.BAS -- program to perform run-length encoding of raster data.

110    'NOTE: Students' actual programs may differ from this example.

200    'Choose data file names, open files.

210    CLS
220    INPUT "Enter the name of the input data file: ", infile$
230    INPUT "Enter the name of the output data file: ", outfile$
240    OPEN infile$ FOR INPUT AS #1
250    OPEN outfile$ FOR OUTPUT AS #2

300    'Hard-wired dimensions...

310    nrows = 109
320    ncols = 120
330    ncells = nrows * ncols
340    DIM z(ncells)

400    'Read data.

410    FOR i = 1 TO ncells
420        INPUT #1, z(i)
430    NEXT

500    'Perform run-length encoding.

510    count = 1
520    FOR i = 1 TO ncells - 1
530        IF z(i) = z(i + 1) THEN
540            count = count + 1
550        ELSE
560            PRINT #2, count; z(i)
570            count = 1
580        END IF
590    NEXT

600    PRINT #2, count; z(i)

700    END
```

```
100  'POEM.BAS -- program to turn a poem into a map.

105  'NOTE: Students' actual programs may differ from this example.

110  CLS
115  PRINT "This program turns poems into maps."
120  PRINT
125  PRINT "The poem is entered as a set of strings of 0s and 1s."
130  PRINT "A 0 is an unstressed syllable and a 1 is a stressed syllable."
135  PRINT
140  PRINT "Each 0 and 1 represents a binary digit (bit)."
145  PRINT "Each line of the poem is entered as a separate string of bits."
150  PRINT "You should leave a space between each bit."
155  PRINT
160  PRINT "Input data include the number of lines in the poem (nlines)"
165  PRINT "and the maximum number of syllables in a line (nsylls)."
170  PRINT "For those lines that have less than nsylls syllables,"
175  PRINT "add extra 0s on the left-hand side of the bit string."
180  PRINT

300  INPUT "How many lines does the poem have? ", nlines
310  INPUT "What's the maximum number of syllables per line? ", nsylls
320  INPUT "What is the name of the input data file? ", filename$

325  'Dimension some arrays and open file for reading.

330  DIM zamboni(nsylls), x(nlines + 1), y(nlines + 1)
340  OPEN filename$ FOR INPUT AS #1

345  'Initialize max and min coordinates.

350  minx = 99999
360  miny = 99999
370  maxx = -99999
380  maxy = -99999

390  'Calculate x and y coordinates for each line of the poem.

400  FOR i = 1 TO nlines
410       x(i) = 0
420       y(i) = 0
440       FOR j = nsylls - 1 TO 0 STEP -1
450            INPUT #1, zamboni(j)
460       NEXT
470       FOR j = nsylls - 1 TO 0 STEP -1
480            factor = 1
```

```
490               basex = j \ 2
500               FOR k = 0 TO basex - 1
510                     factor = factor * 2
520               NEXT
530               IF j MOD 2 = 1 THEN x(i) = x(i) + zamboni(j) * factor
540               IF j MOD 2 = 0 THEN y(i) = y(i) + zamboni(j) * factor
550         NEXT
560         IF x(i) < minx THEN minx = x(i)
570         IF y(i) < miny THEN miny = y(i)
580         IF x(i) > maxx THEN maxx = x(i)
590         IF y(i) > maxy THEN maxy = y(i)
600   NEXT

640   'Last coordinate is the same as the first (if the poem is a polygon).

650   x(nlines + 1) = x(1)
660   y(nlines + 1) = y(1)

690   'Ready to draw the map!

700   SCREEN 12
710   WINDOW (minx - 5, miny - 5)-(maxx + 5, maxy + 5)
800   FOR i = 1 TO nlines
810         LINE (x(i), y(i))-(x(i + 1), y(i + 1))
820   NEXT

1000  END
```

Assignment 5
Surface Modeling with DEMs and TINs

Objectives: In this assignment you will be examining different types of surface models. You will be using QuickBASIC to interpolate and display gridded digital elevation models (DEMs), and pcARC/INFO to examine how surfaces can be modeled using triangulated irregular networks (TINs).

Files: The QuickBASIC part of this assignment requires four QuickBASIC programs (RELIEF.BAS, ILLUMIN.BAS, DRAIN.BAS and INTERPOL.BAS) and two data files (DEM.DAT and SAMPLE.DAT). For the ARC/INFO part of the assignment you will need two coverages:

FACETS a polygon coverage of triangular "facets"

EDGES a coverage containing the arcs defining these facets

Elevation Mapping with Gridded DEMs: Examine the QuickBASIC program called RELIEF.BAS. This program draws a six-color elevation map for a grid, using the elevation data stored in DEM.DAT. The program divides the range of elevation into six classes, and each class is assigned a different color. These colors are defined in the DATA statement on line 540 of the program. The first number defines the color of the lowest elevation class, the second number defines the color of the second-lowest elevation class, and so on.

Run the program several times, changing the values in line 540 to obtain different colors for the elevation ranges. Try to obtain a sequence of colors that conveys a sense of increasing elevation. The colors available to you are:

0 = black	1 = blue	2 = green	3 = cyan
4 = red	5 = magenta	6 = brown	7 = white
8 = dark gray	9 = light blue	10 = light green	11 = light cyan
12 = light red	13 = light magenta	14 = light yellow	15 = white

After running the program, answer the following questions. (You will need to monitor the statistics printed to the screen to answer some of them.)

1. What is the best color sequence you obtained?

2. a) How many cells are in the gridded DEM? b) Assuming that the grid is square (i.e., has the same number of rows and columns), how many rows and columns are in the grid? c) Does your answer agree with the number of elevation values read in by the program from DEM.DAT?

3. a) What are the maximum and minimum cell elevations (z-values)? b) Why does the program need to compute these values?

4. How does the program divide elevations into classes for coloring?

Interpolating to a Grid: Examine the QuickBASIC program called INTERPOL.BAS. This program interpolates the elevations (z-values) for the cells in a gridded DEM using the x,y,z-values for a sample of randomly-selected points. The program uses a distance-weighted interpolation method. For each cell in the grid, the elevation of the cell zc is calculated as:

$$z_c = \frac{\sum_{i=1}^{n} \frac{z_i}{d_{ic}^2}}{\sum_{i=1}^{n} \frac{1}{d_{ic}^2}}$$

where z_i = the elevation of random point i
d_{ic} = the distance between random point i and cell c
n = the number of random points

The program uses a set of random points stored in a file called SAMPLE.DAT to perform the interpolation. This file contains a total of 200 points. When you run the program, you can select any number of points between 1 and 200 as the size of your sample. The program will read this many points from the file.

The program uses only a portion of this sample of points to interpolate the elevation for a given cell. This number of points it uses is defined by the size of the "neighborhood." You can define the neighborhood to contain any number of points between 1 and the total size of your sample.

To illustrate how the program works, imagine that you selected a sample size of 20 points and a neighborhood size of 5 points. The program would read in 20 points from SAMPLE.DAT. For each cell in the grid, it would compute the distance between the cell and each of these points. It would then select the 5 closest points to the cell to interpolate the cell's elevation.

Run the program and create an output file called DEM1.DAT (or anything except DEM.DAT so that you don't destroy the original DEM). To minimize execution time, choose a relatively small value for the sample size (< 25) and the neighborhood size (< 5). Now run RELIEF.BAS after changing the input file name in the program to match the output file you just created.

5. a) In general, how does the appearance of the resulting relief map differ from the maps you created previously? b) How might you account for this difference?

Run INTERPOL.BAS several times, experimenting with different sample and neighborhood sizes. Note that execution time rises dramatically as you increase the values of these variables.

6. How do changes in these variables affect the appearance of the map?

7. a) How do zmin and zmax compare to the values you obtained originally (question 3)? b) How might you account for any differences?

Shaded Relief Mapping with Gridded DEMs: Examine the QuickBASIC program called ILLUMIN.BAS. This program draws a shaded relief map for the same grid you used above (i.e., DEM.DAT). The program creates this map by calculating the slope and aspect of every cell in the grid based on the elevations of the cell's eight neighbors. The illumination of the cell is calculated assuming a light source (i.e., the sun) located 45 degrees above the horizon in the south-west sky. Before computing the slope and aspect, cell elevations are scaled by a vertical exaggeration factor, which effectively stretches the range of illumination values for the grid.

Run the program several times, experimenting with different values for the illumination angle and vertical exaggeration factor.

8. What is the effect on the map of changing these values?

Deriving Drainage Networks from Gridded DEMs: Examine the QuickBASIC program called DRAIN.BAS. This program extracts and maps the drainage network for the same grid you used above (i.e., DEM.DAT). It does this by assuming that surface water can flow into any of the eight neighbors of a given cell. After determining the drainage patterns between cells, the program draws the drainage features for those cells that drain, at minimum, the number of cells defined by a threshold value. For example, if the threshold value was 4, then the drainage patterns would be drawn for every cell that drained at least 4 other cells, but would not be drawn for cells draining 3 or fewer cells.

Run the program several times, experimenting with different threshold values.

9. What effect does changing the threshold value have on the appearance of the drainage network?

10. List at least two problems with the appearance of the drainage network.

The TIN Model: Switch to the directory containing the FACETS and EDGES coverages. Use the ARCS command in ARCPLOT to display EDGES. Create a simple slope map by typing SHADESET COLOR to choose a shadeset file, and POLYGONS FACETS DEGREE_SLOPE SLOPE.LUT to shade in the triangular facets with different shades of grey based on their slopes. (The brighter the shade of grey, the steeper the slope.) Observe that the areas being filled with grey are the same triangular patches defined by the arcs in the EDGES coverage. You can use ARCS to display these arcs again.

Leave ARCPLOT and enter TABLES. Examine the arc attribute file for the EDGES coverage and the polygon attribute file for the FACETS coverage.

11. a) List the items contained in each of these files. b) Based on the names of these items and your knowledge of TINs, what do you think each of these items refers to?

Select the arc attribute file for the EDGES coverage. Now use the RESELECT command to extract only those arcs surrounding polygon # 457 (i.e., only those arcs for which the right polygon # is 457 or the left polygon # is 457). If you use the command correctly, exactly 3 arcs should be reselected.

12. List the command you used to reselect these arcs.

13. Fill in the following data for these arcs.

FNODE#	TNODE#	LPOLY#	RPOLY#	EDGES#	ZFROM	ZTO

14. Draw a diagram illustrating these three arcs and showing:

    a) the polygon # of the triangular facet enclosed by the arcs
    b) the polygon # of each of the three adjacent triangular facets
    c) the node # of each node
    d) the elevation (z-value) of each node
    e) the arc # of each of the three arcs
    f) the direction in which each arc was digitized

Return to ARCPLOT and again display the arcs in the EDGES coverage. Use the RESELECT command as you did before to extract only those arcs surrounding polygon # 457. (Remember that the syntax of the RESELECT command in ARCPLOT is different from that in TABLES.)

Change the line color and display the arcs in EDGES again. (There will of course only be three arcs after you do the reselection.) This will highlight polygon # 457. Now use the IDENTIFY command to obtain the polygon # of each triangular facet adjacent to polygon # 457. Redraw the above diagram, if necessary, correcting for any misorientation.

Use the IDENTIFY command to obtain the slope and aspect of polygon # 457.

15. What slope and aspect values did you obtain?

16. Does the aspect value you obtained make sense in terms of the z-values of the nodes in your diagram? (Aspect varies between 0 and 360 degrees, and is measured in a clockwise direction from the 12 o'clock position.)

Optional: Revise INTERPOL.BAS to perform interpolation based on:

$$z_c = \frac{\sum_{i=1}^{n} \frac{z_i}{d_{ic}^{\alpha}}}{\sum_{i=1}^{n} \frac{1}{d_{ic}^{\alpha}}}$$

where α is a variable entered by the user. Run the program for values of α between 1 and 3. How does changing this variable affect the appearance of the relief map? Hand in a copy of the revised program.

Assignment 5
Program Listing

```
100     'RELIEF.BAS -- A program to display a relief map in 6 colors.

400     DIM z(90, 90)
410     SCREEN 12
420     CLS
430     WINDOW (-10, -10)-(134, 100)

500     'Assign colors for elevation ranges.

510     FOR k = 1 TO 6
520         READ col(k)
530     NEXT
540     DATA 1,2,3,4,5,6

600     'Input elevation raster from file and determine zmax and zmin.

605     OPEN "DEM.DAT" FOR INPUT AS #1
610     zmax = -10000
615     zmin = 100000
630     FOR i = 1 TO 90
635         FOR j = 1 TO 90
640             INPUT #1, z(i, j)
645             IF z(i, j) > zmax THEN zmax = z(i, j)
650             IF z(i, j) < zmin THEN zmin = z(i, j)
655         NEXT
660         LOCATE 21, 60: PRINT "zmin = "; zmin;
670         LOCATE 23, 60: PRINT "zmax = "; zmax
680         LOCATE 25, 60: PRINT "no. of cells = "; i * 90;
690     NEXT

800     'Determine color for each elevation and fill pixel.

820     FOR i = 1 TO 90
830         FOR j = 1 TO 90
840             IF z(i, j) = zmax THEN
850                 c = 6
860             ELSE
870                 c = INT((z(i, j) - zmin) / (zmax - zmin) * 6) + 1
880             END IF
890             LINE (i, j)-(i + 1, j + 1), col(c), BF
900         NEXT
910     NEXT

1000    END
```

```
10      'INTERPOL.BAS

20      'This program interpolates a regular 90x90 grid of elevations
30      'from the x,y,z-coordinates of a set of random points.
40      'The program uses an weighted-distance method (inverse of
50      'square of distance) for a neighborhood of points around each
60      'grid cell. The size of the neighborhood is defined by the user.
70      'The user also defines the number of random points.

100     'Input number of random points (npts) and
105     'number of points in neighborhood (neigh).

190     CLS
200     INPUT "How many random points do you want to use? ", npts
205     IF npts < 1 THEN
210         PRINT
215         PRINT "You must use at least one point."
220         PRINT
225         GOTO 200
230     END IF
250     IF npts > 200 THEN
255         PRINT
260         PRINT "You cannot use more than 200 points."
265         PRINT
270         GOTO 200
275     END IF

300     INPUT "How many points do you want in the neighborhood? ", neighs
305     IF neighs > npts THEN
310         PRINT
315         PRINT "Cannot be greater than the number of random points."
325         PRINT
330         GOTO 300
335     END IF

400     'Dimension necessary arrays.

410     DIM prow(npts), pcol(npts), pz(npts), dist(npts), s(npts)

600     'Open data files.

630     OPEN "SAMPLE.DAT" FOR INPUT AS #1
640     OPEN "DEM1.DAT" FOR OUTPUT AS #2
```

```
700     'Read in x,y,z-coordinates for random points.

710     FOR i = 1 TO npts
720         INPUT #1, prow(i), pcol(i), pz(i)
730     NEXT

900     'Initialize a vector that will hold the numbers 1 through npts, sorted
910     'by the distance between random point i and the current grid cell.

930     FOR i = 1 TO npts
940         s(i) = i
950     NEXT

2000    'Main loop to interpolate value for each grid cell.

2100    FOR r = 1 TO 90
2200        FOR c = 1 TO 90

3000            'Calculate distance between grid cell and each random point.

3200            FOR i = 1 TO npts
3300                distr = r - prow(s(i))
3400                distc = c - pcol(s(i))
3500                dist2(s(i)) = distr * distr + distc * distc
3600            NEXT

4000            'Sort random points by distance from current grid cell.

4100            sorted = 0
4150            DO WHILE sorted = 0
4200                sorted = 1
4250                FOR i = 1 TO (npts - 1)
4300                    IF dist2(s(i)) > dist2(s(i + 1)) THEN
4350                        temp = s(i)
4400                        s(i) = s(i + 1)
4450                        s(i + 1) = temp
4500                        sorted = 0
4550                    END IF
4600                NEXT
4650            LOOP
```

```
5000            'Interpolate the value for the current grid cell.

5200            numer = 0
5250            denom = 0
5300            FOR i = 1 TO neighs
5350                numer = numer + (pz(s(i)) / dist2(s(i)))
5400                denom = denom + (1 / dist2(s(i)))
5450            NEXT
5500            PRINT #2, numer / denom
5550        NEXT
5600    NEXT

10000   END
```

```
100     'ILLUMIN.BAS -- a program to create a shaded relief map.

130     'exag = vertical exaggeration; sun = illumination angle.

150     exag = 20
160     sun = 45
170     tansun = TAN(sun * 3.14159 / 180)

210     DIM z(90, 90)
220     SCREEN 12
230     CLS
240     WINDOW (-10, -10)-(134, 100)

300     'Assign grey scale for illumination.

310     FOR k = 1 TO 4
320         READ col(k)
330     NEXT
340     DATA 0,8,7,15

400     'Input elevation raster from file.

410     OPEN "DEM.DAT" FOR INPUT AS #1
420     FOR i = 1 TO 90
430         FOR j = 1 TO 90
460             INPUT #1, z(i, j)
470             z(i, j) = z(i, j) * exag
480         NEXT
490     NEXT

600     'Step through each cell, omitting boundary cells.

610     FOR i = 2 TO 89
620         FOR j = 2 TO 89
650             b = -z(i - 1, j - 1) - z(i - 1, j) - z(i - 1, j + 1)
660             b = (b + z(i + 1, j - 1) + z(i + 1, j) + z(i + 1, j + 1)) / 9
680             c = -z(i - 1, j - 1) - z(i, j - 1) - z(i + 1, j - 1)
690             c = (c + z(i - 1, j + 1) + z(i, j + 1) + z(i + 1, j + 1)) / 9
710             il = (b + c + tansun)
720             il = il / SQR(b ^ 2 + c ^ 2 + 1) / SQR(2 + tansun ^ 2)
730             IF il < 0 THEN il = 0
740             c = INT(il * 4) + 1
750             LINE (i, j)-(i + 1, j + 1), col(c), BF
760         NEXT
770     NEXT

1000    END
```

```
1100    'DRAIN.BAS -- a program to develop a reduced drainage network.

1120    'Drainage net will be drawn for all cells that drain at minimum the
1130    'number of cells specified by threshold value.  (e.g., if threshold = 2
1140    'then drainage net will be drawn for cells that drain 2 or more cells.)

1160    threshold = 2

1210    DIM z(90, 90), toi(90, 90), toj(90, 90), counter(90, 90)
1220    SCREEN 12
1230    CLS
1240    WINDOW (-10, -10)-(134, 100)

1400    'Input elevation raster from file.

1410    OPEN "DEM.DAT" FOR INPUT AS #1
1420    FOR i = 1 TO 90
1430        FOR j = 1 TO 90
1460            INPUT #1, z(i, j)
1470        NEXT
1480    NEXT

1600    'For each cell, determine which cell it drains into.

1610    FOR i = 2 TO 89
1620        FOR j = 2 TO 89
1630            zmin = 1000
1640            FOR n = -1 TO 1
1650                FOR m = -1 TO 1
1660                    IF z(i + n, j + m) < zmin THEN
1670                        zmin = z(i + n, j + m)
1680                        toi(i, j) = i + n
1690                        toj(i, j) = j + m
1700                    END IF
1710                NEXT
1720            NEXT
1730            IF toi(i, j) = i AND toj(i, j) = j THEN
1740                toi(i, j) = 0
1750                toj(i, j) = 0
1760            END IF
1770        NEXT
1780    NEXT
```

```
1900    'Initialize matrix used to accumulate flow into each cell.

1910    FOR i = 1 TO 90
1920        FOR j = 1 TO 90
1930            counter(i, j) = 0
1940        NEXT
1950    NEXT

11000   'Follow drainage route of each cell.

11010   FOR i = 1 TO 90
11030       FOR j = 1 TO 90
11050           ix = i
11070           jx = j
11090           DO WHILE ix <> 0 AND jx <> 0
11110               counter(ix, jx) = counter(ix, jx) + 1
11130               ix = toi(ix, jx)
11150               jx = toj(ix, jx)
11170           LOOP
11190       NEXT
11210   NEXT

11400   'Draw drainage net.

11410   FOR i = 1 TO 90
11420       FOR j = 1 TO 90
11430           IF counter(i, j) >= threshold THEN
11440               IF toi(i, j) <> 0 AND toj(i, j) <> 0 THEN
11450                   LINE (i, j)-(toi(i, j), toj(i, j))
11460               END IF
11470           END IF
11480       NEXT
11490   NEXT

12000   END
```

Assignment 5
Answer Key

1. (Results will vary here.)

2. **a)** 8100 **b)** 90 rows and 90 columns **c)** Yes, since the program contains a nested loop with i = 1 to 90 and j = 1 to 90.

3. **a)** 100 and 0 **b)** To divide the range of elevations into classes for coloring.

4. Equal interval slicing. Basically, the range of values is divided into six equally-sized classes. (See line 870.)

5. **a)** They are more generalized and contain some artifacts. **b)** The program is estimating the values in 8100 cells from only a few points.

6. Increasing either variable reduces amount of generalization and number of artifacts.

7. **a)** New zmin is larger and new zmax is smaller. **b)** The map has been generalized, so the maximum and minimum values are not as extreme.

8. Changing the illumination angle changes the amount of shadow. The higher the sun, the brighter the map. As the vertical exaggeration factor is increased, surface shape becomes more apparent.

9. An increase in the threshold results in a decrease in the number of drainage features displayed on the map.

10. Many features are unconnected to the network. The network only allows streams to flow in eight directions. In areas of low relief, there are sets of parallel streams very close to each other. (Others...)

11. **FACETS.PAT**

AREA	polygon area
PERIMETER	polygon perimeter
FACETS#	coverage#
FACETS-ID	coverage-id
DEGREE_SLOPE	slope in degrees
ASPECT	aspect
SAREA	surface area of polygon

EDGES.AAT

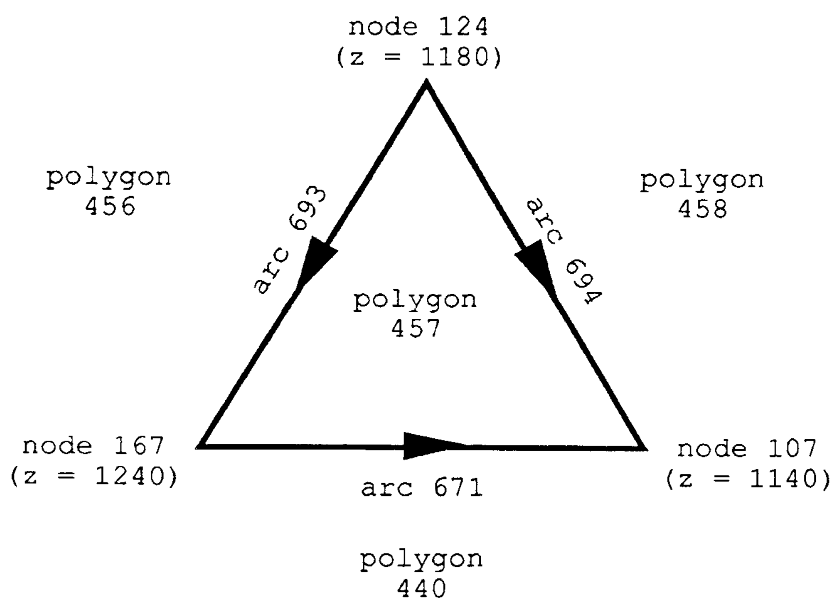
FNODE#	from node #
TNODE#	to node #
LPOLY#	left polygon #
RPOLY#	right polygon #
LENGTH	arc length
EDGES#	coverage#
EDGES-ID	coverage-id
DEGREE_SLOPE	slope in degrees
LSLOPE	slope of left polygon
RSLOPE	slope of right polygon
LASPECT	aspect of left polygon
RASPECT	aspect of right polygon
ZFROM	z-value of from node
ZTO	z-value of to node

12. RESELECT RPOLY# = 457 OR LPOLY# = 457

13.

FNODE#	TNODE#	LPOLY#	RPOLY#	EDGES#	ZFROM	ZTO
167	107	457	440	671	1240	1140
124	167	457	456	693	1180	1240
124	107	458	457	694	1180	1140

14.



15. SLOPE = 7.261 ASPECT = 38.293

16. Yes. (Explain in terms of the direction in which the facet is facing.)

Assignment 5
Program Listing (Answer Key)

```
10      'INTERPOL.BAS

20      'This program interpolates a regular 90x90 grid of elevations
30      'from the x,y,z-coordinates of a set of random points.
40      'The program uses an weighted-distance method (as defined by
50      'the value of alpha) for a neighborhood of points around each
60      'grid cell.  The size of the neighborhood is defined by the user.
70      'The user also defines the number of random points.

100     'Input number of random points (npts) and
105     'number of points in neighborhood (neigh).

190     CLS
200     INPUT "How many random points do you want to use? ", npts
205     IF npts < 1 THEN
210         PRINT
215         PRINT "You must use at least one point."
220         PRINT
225         GOTO 200
230     END IF
250     IF npts > 200 THEN
255         PRINT
260         PRINT "You cannot use more than 200 points."
265         PRINT
270         GOTO 200
275     END IF

300     INPUT "How many points do you want in the neighborhood? ", neighs
305     IF neighs > npts THEN
310         PRINT
315         PRINT "Cannot be greater than the number of random points."
325         PRINT
330         GOTO 300
335     END IF

340     'Input value of alpha.

345     INPUT "What value of alpha do you want? ", alpha
350     IF alpha > 3 or alpha < 1 THEN
355         PRINT
360         PRINT "Alpha should be between 1 and 3."
365         PRINT
370         GOTO 345
375     END IF
```

```
380     'Divide alpha by 2 rather than taking square root of distance in line 3500.

385     alpha = alpha / 2

400     'Dimension arrays.

410     DIM prow(npts), pcol(npts), pz(npts), dist(npts), s(npts)

600     'Open data files.

630     OPEN "SAMPLE.DAT" FOR INPUT AS #1
640     OPEN "DEM1.DAT" FOR OUTPUT AS #2

700     'Read in x,y,z for random points.

710     FOR i = 1 TO npts
720         INPUT #1, prow(i), pcol(i), pz(i)
730     NEXT

900     'Initialize a vector that will hold the numbers 1 through npts, sorted
910     'by the distance between random point i and the current grid cell.

930     FOR i = 1 TO npts
940         s(i) = i
950     NEXT

2000    'Main loop to interpolate value for each grid cell.

2100    FOR r = 1 TO 90
2200        FOR c = 1 TO 90

3000            'Calculate distance between grid cell and each random point.

3200            FOR i = 1 TO npts
3300                distr = r - prow(s(i))
3400                distc = c - pcol(s(i))
3500                dist2(s(i)) = distr * distr + distc * distc
3600            NEXT
```

```
4000            'Sort random points by distance from current grid cell.

4100            sorted = 0
4150            DO WHILE sorted = 0
4200                sorted = 1
4250                FOR i = 1 TO (npts - 1)
4300                    IF dist2(s(i)) > dist2(s(i + 1)) THEN
4350                        temp = s(i)
4400                        s(i) = s(i + 1)
4450                        s(i + 1) = temp
4500                        sorted = 0
4550                    END IF
4600                NEXT
4650            LOOP

5000            'Interpolate the value for the current grid cell.

5200            numer = 0
5250            denom = 0
5300            FOR i = 1 TO neighs
5350                numer = numer + (pz(s(i)) / dist2(s(i)) ^ alpha)
5400                denom = denom + (1 / dist2(s(i)) ^ alpha)
5450            NEXT
5500            PRINT #2, numer / denom
5550        NEXT
5600    NEXT

10000   END
```

Assignment 6
Kriging and the Semivariogram

Objectives: In this assignment you will be using QuickBASIC to construct a semivariogram and perform kriging.

Files: This assignment requires one QuickBASIC program (SEMIVAR.BAS) and two data files (ELEVPTS.DAT and PRECPTS.DAT).

The Semivariogram: Each of the two data files contains the x,y,z-coordinates of 100 random points. In ELEVPTS.DAT the z-coordinates refer to elevations (in meters), and in PRECPTS.DAT they refer to annual precipitation (in millimeters). Both data files were constructed from raster data files of Africa.

The QuickBASIC program called SEMIVAR.BAS computes the information necessary to construct the semivariogram. Each line of the program's output contains two values. The first is the distance (or spatial lag), and the second is the computed semivariance for that spatial lag.

Run the program for each of the two data files and make a printout of each of the resulting output files. Using these printouts, draw the semivariogram for each of the two data files, putting the spatial lag on the x-axis and the semivariance on the y-axis. Hand in a copy of your semivariograms.

1. What is the approximate value of the sill, range and nugget for each of the data files?

2. Is the general form of the semivariogram the same or different for the two data files? What does this say about spatial variation of elevation versus precipitation?

Kriging: The semivariogram can be used to perform kriging, a type of interpolation. The objective of this part of the assignment is to use the semivariogram derived from the elevation data file to interpolate the elevation at a selected point based on the elevations at three neighboring points. In this case the interpolated elevation or z-value (z0) is estimated as

$$z_0 = \lambda_1 z_1 + \lambda_2 z_2 + \lambda_3 z_3 \qquad (1)$$

where z_1 through z_3 are the observed z-values at points 1 through 3, and λ_1 through λ_3 are a set of weights derived from the semivariogram. The observed z-values of the points are as follows:

$$z_1 = 1000 \qquad z_2 = 1500 \qquad z_3 = 2000$$

Computation of the weights in the above equation is most easily achieved using matrix algebra. Most of this computation has been done for you:

$$\lambda_1 = -2.91\text{x}10^{-7} \times \gamma(0,1) + 1.64\text{x}10^{-7} \times \gamma(0,2) + 1.27\text{x}10^{-7} \times \gamma(0,3) + 0.125 \qquad (2)$$

$$\lambda_2 = 1.64\text{x}10^{-7} \times \gamma(0,1) - 1.88\text{x}10^{-7} \times \gamma(0,2) + 2.38\text{x}10^{-8} \times \gamma(0,3) + 0.430 \qquad (3)$$

$$\lambda_3 = 1.27\text{x}10^{-7} \times \gamma(0,1) + 2.38\text{x}10^{-8} \times \gamma(0,2) - 1.51\text{x}10^{-7} \times \gamma(0,3) + 0.446 \qquad (4)$$

In these equations, $\gamma(0,1)$ is the semi-variance for the interpolated point (point 0) and neighboring point 1, $\gamma(0,2)$ is the semi-variance for the interpolated point and neighboring point 2, and $\gamma(0,3)$ is the semi-variance for the interpolated point and neighboring point 3. These values can be obtained directly from your semivariogram data if you know the distance between point 0 and each of the three neighboring points. Use the following distance values:

distance between point 0 and neighboring point 1: 5
distance between point 0 and neighboring point 2: 4
distance between point 0 and neighboring point 3: 3

You're on the right track if you got a value of 5,628,776 for $\gamma(0,1)$.

3. Use equations (2), (3) and (4) to compute the values of λ_1, λ_2 and λ_3. (If the three computed values do not sum to approximately 1.0, something went wrong in your calculations.)

4. Use equation (1) to compute the value of z_0.

Assignment 6
Program Listing

```
100     'SEMIVAR.BAS

150     'This program computes the semivariance from a set of points.

200     CLS
210     INPUT "What is the name of the input file? ", infile$
220     INPUT "What do you want to call the output file? ", outfile$

250     'Open data files for input and output.

260     OPEN infile$ FOR INPUT AS #1
270     OPEN outfile$ FOR OUTPUT AS #2

300     'Hard-wire values for some key variables.

310     npts = 100          'Number of points in input file.
320     maxd = 20           'Approx maximum distance between cells.

400     'Dimension a bunch of arrays.

410     DIM x(npts), y(npts), z(npts)
420     DIM variance(maxd), count(maxd)

500     'Input the x, y and z coordinates for the points.

510     FOR i = 1 TO npts
520         INPUT #1, x(i), y(i), z(i)
530     NEXT

600     'Initialize variance and count to zero for all spatial lags.

610     FOR k = 0 TO maxd
620         variance(k) = 0
630         count(k) = 0
640     NEXT
```

```
700     'Compute variance for all points separated by a given spatial lag.

710     FOR i = 1 TO npts
715          PRINT "Working on point "; i
720          FOR j = 1 TO npts
730               xdist = x(i) - x(j)
740               ydist = y(i) - y(j)
745               zdiff = z(i) - z(j)
750               dist = SQR(xdist ^ 2 + ydist ^ 2)
760               disti = CINT(dist)
770               variance(disti) = variance(disti) + zdiff ^ 2
780               count(disti) = count(disti) + 1
790          NEXT
800     NEXT

900     'Print semivariance and spatial lag.

910     FOR k = 0 TO maxd
920          IF count(k) <> 0 THEN
930               PRINT #2, k; .5 * variance(k) / count(k)
940          END IF
950     NEXT

1000    END
```

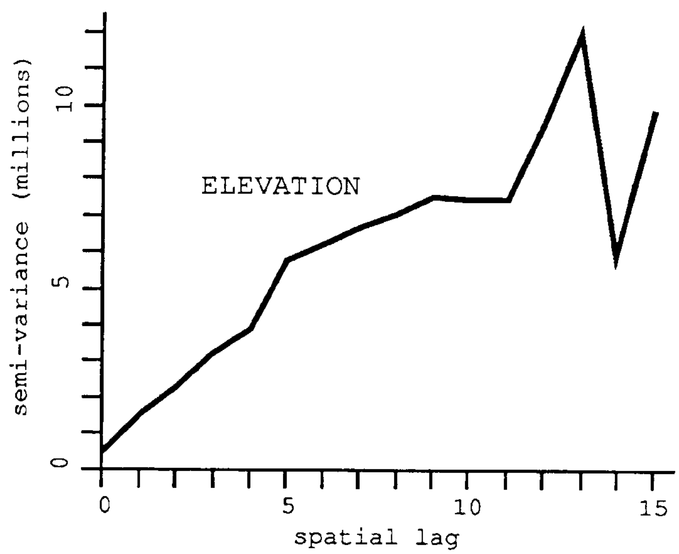
Assignment 6
Answer Key

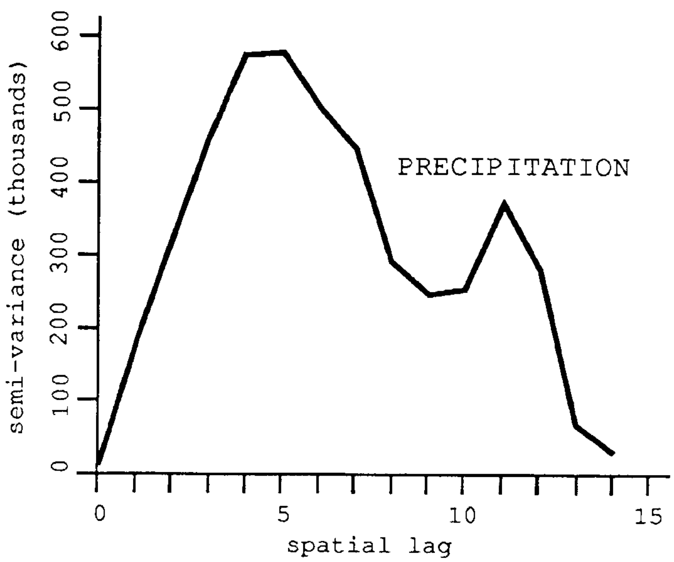
1. Elevation: sill = 11,876,000 range = 13 nugget = 385,000
Precipitation: sill = 578,000 range = 5 nugget = 7,600

2. The form is different. Elevation has a greater range than precipitation, indicating that, over short distances, precipitation is more variable.

3. $\lambda_1 = -0.501$ $\lambda_2 = 0.719$ $\lambda_3 = 0.782$

4. $z_0 = -0.501\ (1000) + 0.719\ (1500) + 0.782\ (2000) = 2141.5$





Assignment 7
Uncertainty, Fuzzy Logic and Relational Databases

Objectives: In this assignment you will be using fuzzy logic in a relational database schema to manipulate uncertainty in spatial data.

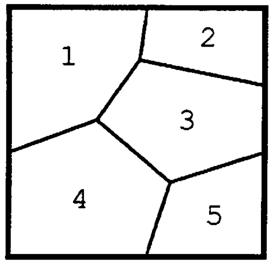
Representing Uncertainty in a Database: Uncertainty in spatial databases occurs because the features contained in these databases are abstractions of real-world phenomena. For example, a database containing the digitized outline of the Devereaux Slough contains uncertainty because the boundaries of the Slough change from season to season. A map of vegetation types in the Los Padres Forest contains uncertainty because it is necessary to generalize vegetation types in order to produce a legible map. A set of polygons representing different soil types in the Goleta Valley contains uncertainty because the boundary between adjacent soil types may actually be a fairly wide zone in which the soil properties of one soil type gradually merge with those of another soil type.

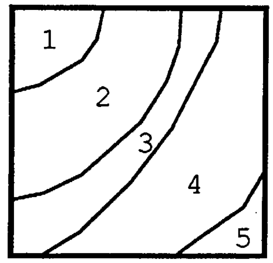
These example illustrate that the spatial databases you work with in a GIS are not accurate "snapshots" of the real world, but instead are imprecise "models" in which the degree and nature of uncertainty depends on the type of data represented and the purposes for which the database was designed.

Fuzzy logic can be used to represent uncertainty in spatial databases and manipulate this uncertainty as data are transformed by GIS functions. One way to implement fuzzy logic is to define a "membership function" for the features in the database. The membership function, which is usually denoted by the letter m, varies between 0.0 and 1.0 for any feature. This value indicates the degree of uncertainty in the feature. For example, assume that you assigned a value of m to each of the polygons in a soil map. A value of m near 1.0 would indicate little uncertainty (i.e., you are reasonable confident that the soil type of the polygon is correct), while a value near 0.0 would indicate a great deal of uncertainty (i.e., you are not very confident that the soil type is correct).

Your first task in this assignment is to examine how the membership function can be used to represent uncertainty. Assume that you have digitized the two maps depicted below. The first (VEG) shows vegetation types and the second (SLOPE) shows slope classes. Both maps cover the same geographical area. You can think of each digitized map as being equivalent to an ARC/INFO coverage.

The polygon attribute tables for the two coverages are also shown below. These tables are named VEG.PAT and SLOPE.PAT respectively. The items called "veg-id" and "slope-id" refer to the polygon numbers on the maps.





VEG.PAT

veg-id	veg-type
1	chaparral
2	sage brush
3	grass
4	sage brush
5	barren

SLOPE.PAT

slope-id	slope-class
1	steep
2	gentle
3	flat
4	gentle
5	steep

Now assume that you have available the following tables (called VEG.MEM and SLOPE.MEM), which indicate the membership function values for each veg-type and slope-class value.

VEG.MEM

veg-type	veg-μ
chaparral	0.8
sage brush	0.7
grass	0.8
barren	0.5

SLOPE.MEM

slope-class	slope-μ
steep	0.9
gentle	0.8
flat	0.6

In order to use these membership function values to represent uncertainty, you will have to perform a relational database operation known as a relational join. This operation involves merging the attributes from two different tables based on a common item (sometimes refered to as the "key" item). For example, if you performed a relational join on VEG.PAT and VEG.MEM, you would obtain a veg-m value of 0.8 for the polygon with a veg-id value of 1, based on the common veg-type value of "chaparral."

Using a relational join, modify the polygon attribute tables for VEG and SLOPE by inserting the appropriate membership function values for each polygon. Record your answers in the table below.

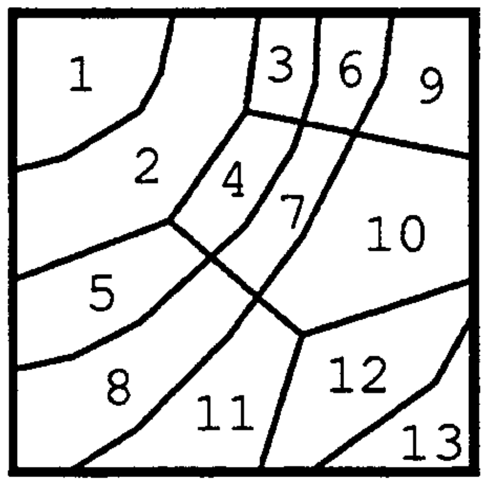
VEG.PAT

veg-id	veg-type	veg-μ
1	chaparral	
2	sage brush	
3	grass	
4	sage brush	
5	barren	

SLOPE.PAT

slope-id	slope-class	slope-μ
1	steep	
2	gentle	
3	flat	
4	gentle	
5	steep	

Manipulating Uncertainty in GIS Operations: Imagine that the two coverages are merged to create the following coverage (called "MERGED").



"MERGED"

How is the membership function manipulated as the coverages are merged in this way? In order to answer that question, you first need to fill in the following polygon attribute table for MERGED based on the modified polygon attribute tables for VEG and SLOPE.

MERGED.PAT

merged-id	veg-id	slope-id	veg-type	slope-class	veg-μ	slope-μ
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						

As you can see from this table, each polygon in MERGED has two membership function values, one from each of the two input coverages. The way in which these membership function values are combined depends on the way in which the attributes from the two input coverages are manipulated. Consider the set of polygons in MERGED for which veg-type = "chaparral" and slope-class = "steep". There is one polygon in this set (i.e., merged-id = 1).

The question that now arises is, "How certain are you that this polygon actually has a veg-type of chaparral and a slope-class of steep?" To answer this question you need to combine the membership function values from the two input coverages. In this case the combined membership function value is equal to the minimum of the veg-m and slope-m values for the polygon. Because the question is concerned with uncertainty in both input attributes (i.e., veg-type and slope-class), the result cannot be any more certain than the least certain of these attributes. In the example, veg-m = 0.8 and slope-m = 0.9 for the selected polygon, and therefore the combined membership function value is 0.8.

One might also ask, "How certain are you that the selected polygon actually has a veg-type of chaparral or a slope-class of steep?" This question is based on the logical operator "or" while the previous question is based on the logical operator "and." In the case of the or operator, the combined membership function value is defined as the maximum (rather than the minimum) of veg-m and slope-m. Because you are concerned with uncertainty in either input attribute (not both) the result is as certain as the most certain of these attributes. Thus the combined membership function value for the selected polygon is 0.9.

For each set of selection criteria below, list the set of polygons that meets the criteria. Then compute the combined membership function value for each polygon based on the specified logical operator.

Selection criteria: veg-type = "barren" and slope-class = "gentle"
Logical operator: and

Selection criteria: veg-type = "sage brush" and slope-class = "flat"
Logical operator: and

Selection criteria: veg-type = "barren" and slope-class = "steep"
Logical operator: or

Selection criteria: veg-type = "sage brush" and slope-class = "gentle"
Logical operator: or

Similarity Relations: Fuzzy logic can also be used to examine how closely a set of selected polygons conform to a particular set of criteria. For example, imagine that you selected all polygons with veg-type = "sage brush" and slope-class =

"flat". Note that several other polygons have similar attributes but do not conform exactly to the criteria you have specified (e.g., polygons with veg-type = "sage brush" and slope-class = "gentle"). These polygons are quite similar to the set of polygons that you have selected, but would not actually be included in the selected set.

Such similarities can be handled using the concept of "similarity relations." A similarity relation is simply a table indicating the degree of similarity between different values of an attribute. Consider the following two similarity relations, the first for veg-type values and the second for slope-class values.

	chaparral	sage brush	grass	barren
chaparral	1.0	0.8	0.4	0.0
sage brush	0.8	1.0	0.6	0.0
grass	0.4	0.6	1.0	0.0
barren	0.0	0.0	0.0	1.0

	steep	gentle	flat
steep	1.0	0.5	0.0
gentle	0.5	1.0	0.5
flat	0.0	0.5	1.0

The interpretation of these tables is relatively straightforward. For example, look at the second table, and follow the row labelled "steep" until you hit the column labelled "gentle." The value of 0.5 indicates the degree of similarity between "steep" and "gentle." There is more similarity between "steep" and "gentle" than between "steep" and "flat," for which the similarity value given in the table is 0.0. Note that the similarity between any value and itself is always 1.0, and that each table is symmetrical.

It is easier to conceptualize the similarities between slope classes than between vegetation types, but, as shown in the second table, there may be context-dependent dimensions along which the similarity between different vegetation types can also be measured.

With the aid of these similarity relations, it is possible to formulate "fuzzy" responses to questions about the attributes of any polygon. For example, you might ask how closely the polygon with merged-id = 1 conforms to the criteria of veg-type = "sage brush" and slope-class = "gentle". From MERGED.PAT you can easily determine that, for this polygon, veg-type = "chaparral" and slope-class = "steep". The above similarity relations show that the similarity between "sage brush" and "chaparral" is 0.8 and that the similarity between "gentle" and "steep" is 0.5. Since you asked a question involving an and (rather than an or), the minimum of the two similarity values is used. Thus the polygon has a similarity of 0.5 to the criteria you specified. By substituting the appropriate attribute values, the same approach can be used for any other polygon.

If the question involves an or, then the maximum of the two similarity values is used. For example, if you asked how closely the polygon with merged-id = 1 conforms to the criteria of veg-type = "sage brush" or slope-class = "gentle", the answer would be 0.8, the maximum of the two similarity values.

Compute how closely each polygon in MERGED conforms to the following criteria:

veg-type = "grass" and slope-class = "steep"

veg-type = "chaparral" or slope-class = "flat"

Optional: Implement the membership function concept in ARC/INFO. You can use the SITES and DEVELOP coverages from Assignment 3. Each item in the polygon attribute table for a coverage may have its own membership function. (You can add items to the table if you want.) Use the relational join capability of INFO to incorporate the membership function values into the polygon attribute tables of each coverage. Use the UNION command to overlay the two coverages and merge their attributes. Show how the membership function values from the input coverages would be combined when manipulating the attribute data in different ways.

Assignment 7
Answer Key

VEG.PAT

veg-id	veg-type	veg-μ
1	chaparral	0.8
2	sage brush	0.7
3	grass	0.8
4	sage brush	0.7
5	barren	0.5

SLOPE.PAT

slope-id	slope-class	slope-μ
1	steep	0.9
2	gentle	0.8
3	flat	0.6
4	gentle	0.8
5	steep	0.9

MERGED.PAT

merged-id	veg-id	slope-id	veg-type	slope-class	veg-μ	slope-μ
1	1	1	chaparral	steep	0.8	0.9
2	1	2	chaparral	gentle	0.8	0.8
3	2	2	sage brush	gentle	0.7	0.8
4	3	2	grass	gentle	0.8	0.8
5	4	2	sage brush	gentle	0.7	0.8
6	2	3	sage brush	flat	0.7	0.6
7	3	3	grass	flat	0.8	0.6
8	4	3	sage brush	flat	0.7	0.6
9	2	4	sage brush	gentle	0.7	0.8
10	3	4	grass	gentle	0.8	0.8
11	4	4	sage brush	gentle	0.7	0.8
12	5	4	barren	gentle	0.5	0.8
13	5	5	barren	steep	0.5	0.9

Selection criteria: veg-type = "barren" and slope-class = "gentle"
Logical operator: and
Selected polygons (merged-id): 12
Membership function values: 0.5

Selection criteria: veg-type = "sage brush" and slope-class = "flat"
Logical operator: and
Selected polygons (merged-id): 6 8
Membership function values: 0.6 0.6

Selection criteria: veg-type = "barren" and slope-class = "steep"
Logical operator: or
Selected polygons (merged-id): 13
Membership function values: 0.9

Selection criteria: veg-type = "sage brush" and slope-class = "gentle"
Logical operator: or

Selected polygons (merged-id):	3	5	9	11
Membership function values:	0.8	0.8	0.8	.8

veg-type = "grass" and slope-class = "steep"

merged-id	similarity
1	min(0.4, 1.0) = 0.4
2	min(0.4, 0.5) = 0.4
3	min(0.6, 0.5) = 0.5
4	min(0.6, 0.5) = 0.5
5	min(0.6, 0.5) = 0.5
6	min(0.6, 0.5) = 0.5
7	min(0.6, 0.5) = 0.5
8	min(0.6, 0.5) = 0.5
9	min(0.6, 0.5) = 0.5
10	min(0.6, 0.5) = 0.5
11	min(0.6, 0.5) = 0.5
12	min(0.6, 0.5) = 0.5
13	min(0.6, 0.5) = 0.5

veg-type = "chaparral" or slope-class = "flat"

merged-id	similarity
1	max(1.0, 0.0) = 1.0
2	max(1.0, 0.5) = 1.0
3	max(0.8, 0.5) = 0.8
4	max(0.4, 0.5) = 0.5
5	max(0.8, 0.5) = 0.8
6	max(0.8, 1.0) = 1.0
7	max(0.4, 1.0) = 1.0
8	max(0.8, 1.0) = 1.0
9	max(0.8, 0.5) = 0.8
10	max(0.4, 0.5) = 0.5
11	max(0.8, 0.5) = 0.8
12	max(0.0, 0.5) = 0.5
13	max(0.0, 0.0) = 0.0