UNIT 9: SPATIAL DATA CONVERSION

Written by Rustin Dodson, Santa Barbara, California

Context

Spatial datasets are produced and distributed in a variety of formats: vector, raster, point, line, polygon, image, etc. Often datasets are designed for certain computer systems (Unix, DOS/Windows, Macintosh) or software programs (GRASS, Idrisi, Arc/Info). In the likely event that an important dataset is available, but in the wrong format, the GIS analyst must be aware of the issues, methods, and tools for converting spatial data to a format which is compatible with the current GIS project.

Learning Outcomes

Awareness:

Students should be aware of common data formats, computer systems, and spatial data handling software programs that are likely to be encountered. Students should be able to convert between data formats using the built-in conversion tools of a GIS program.

Competency:

Students should have a working knowledge of image data conversion issues, including image formats, data types, and byte swapping.

Mastery:

Students should be fluent in a programming or scripting language which allows custom creation of data conversion tools.

Example Application

As part of a study in mapping potential ranges of plant species, a GIS analyst has been assigned the task of identifying locations in the USA which are not subject to frost (sub-

freezing temperatures), based on the past 20 years of temperature observations. The analyst has located a set of daily temperature measurements made by the National Climatic Data Center (NCDC). The temperature measurements are stored as simple text files which contain the measurement station ID, the date and time of measurement, and the temperature observations themselves. Another text file contains the ID and location for each measurement station.

After downloading the necessary data files, the GIS analyst needs to

1. **Extract data records with the desired temperature variable (daily minimum temperature) and for the desired time period (the past 20 years).**
2. **Identify and account for missing values in the temperature records.**
3. **Convert daily minimum temperatures to a measure of mean frost days per year (FROSTDAYS).**
4. **Obtain the geographic locations of the measurement stations, and attach them to FROSTDAYS for each station.**
5. **Convert the station locations and FROSTDAYS into a GIS dataset (a point layer).**
6. **Attempt to derive a continuous surface of FROSTDAYS from the point dataset in the previous step.**

Preparatory Units

Recommended:

- Data acquisition (UNIT 1)
- Using and interpreting metadata (UNIT 7)
- Projecting data (UNIT 10)

Complementary:

- Error checking (UNIT 8)
- Registration and Conflation (UNIT 11)
- Planning a digitizing project (UNIT 12)
- Planning a scanning project (UNIT 16)

Awareness

Learning Objectives:
After completing this section you should be able to:

- Describe some common spatial data formats.
- Identify some common GIS software packages and describe some of their data formats.

Vocabulary:

- Import

- Export
- USGS
- Spatial data transfer standard (SDTS)
- USGS digital line graph (DLG) format
- USGS digital elevation model (DEM) format
- US Bureau of the Census TIGER files
- DXF file format
- Image band
- Band sequential (BSQ) image format
- Band interleaved by pixel (BIP) image format
- Band interleaved by line (BIL) image format
- Scripting languages
    - AWK
    - PERL
- Byte-swapping
- Big endian
- Little endian

Basic Knowledge/Skills:

Generic spatial data formats

Imagine a simple map of the contiguous 48 United States. Such a map could be stored in a variety of formats:

1. Vector polygons, where each state is stored as a continuous chain of points and indexed by a unique ID number.
2. Vector lines, where state boundaries are stored as chains of points, but no polygon IDs are present.
3. Raster polygons, where each pixel of a raster image stores the ID number of state or a background value such as zero.
4. Raster lines, where pixels which fall on a state boundary have non-zero values while the rest of the image contains zeros.

Each format above contains information on the shapes and locations of the 48 states. Some formats are better suited for analysis in a GIS, while others are useful for display only. Sometimes data are available in only one format and must be converted in order to be usable within a given GIS project. For example, if a hard-copy map is scanned, the resulting dataset is often in format 4 above (raster lines). To be useful for analysis, the dataset may have to be converted to one of the other formats (1-3).

Advanced Knowledge/Skills:

Published spatial data formats The following formats are used by government agencies who publish spatial datasets. Many GIS packages contain built-in functionality for importing and/or exporting these common formats.

SDTS - Spatial Data Transfer Standard.

Created by the Federal Geographic Data Committee (FGDC) over a period of several years, this is a detailed and exhaustive standard which includes vector and raster data. Federal agencies are required to support SDTS, and are in the process of converting their spatial datasets to SDTS format. Most GIS software producers have completed or are in the process of creating SDTS translators for their GIS products. In addition, a number of public-domain SDTS conversion programs are available:

- SDTS-to-MapInfo and SDTS-to-DXF translators are available at *[link removed]*
- A freeware program, MICRODEM, manipulates DEM files in DTED, USGS DEM, and SDTS formats. MICRODEM runs on Windows 95/NT, and is available at *[link removed]*

- **DLG - Digital Line Graph**.

DLGs are vector data files created and maintained by the USGS. They are available at several scales, and typically include:

1. Political and physical boundaries
2. Transportation, including roads, trails, railroads, pipelines, and airports
3. Hydrography, including streams, rivers, and lakes
4. Man-made features, including built-up areas, capitals, county seats, populated places, and population range
5. Public Land Survey System, including land grants, township, range, and subdivisions of the public lands.

DLG files were originally provided in their own format, known as DLG format. Many GIS packages contain functions for importing DLG format data. Since the adoption of the SDTS, however, the USGS has been creating SDTS versions of the national DLG dataset. At some point in the future, the DLG format may disappear completely, but for the time being there is quite a bit of DLG format data available.

- **DEM - Digital Elevation Model format**.

In GIS jargon, "DEM" is often used to generically refer to "a set of gridded elevation data". However, "DEM" can also refer to the format used by the USGS for distributing its digital elevation model data. USGS DEMs are available at several scales over most of the United States (coverage is not complete for the finest resolution DEM data). DEM conversion tools exist within many commercial GIS packages. The USGS is in the process of converting DEM format data to the SDTS format.

A freeware program, MICRODEM, manipulates DEM files in DTED, USGS DEM, and SDTS formats. MICRODEM runs on Windows 95/NT, and is available at *[link removed]*

- **TIGER - Topologically Integrated Geographic Encoding and Referencing**.

TIGER files are produced and maintained by the US Census Bureau, and contain vector boundary data designed for use with population and demographic data collected by the Census Bureau. TIGER files contain:

1. Polygon data, including census tracts and blocks, congressional districts, and ZIP code boundaries
2. Line data, including highways and streets
3. Attribute data for street segments, including address ranges and ZIP codes
4. Point data, including landmarks and "key geographic features"

- **DXF - Drawing eXchange Format.**

DXF is a vector format used by the AutoCAD software package. Many GIS programs offer functions to import and/or export the DXF format.

GIS-specific formats

The following list describes the data formats used by a number of popular GIS packages. Most full-function GIS programs, like the ones listed below, can import and export data from a number of standard data formats. In addition, most can export data to simple text files so that data can be manipulated with user-written programs. For example, the GRASS GIS has the commands `r.out.ascii` and `v.out.ascii` which convert raster and vector data to text files. Arc/Info has the commands `gridascii` and `ungenerate`, which do the same thing, but create text files of a slightly different format.

NOTE: The selection of GIS programs below reflects those which have been used extensively by the author. This is not intended to be an exhaustive list of GIS software, nor is it meant to be a list of the best or most popular GIS programs.

Data formats supported by some common GIS packages:

- **GRASS**

GRASS primarily supports raster map layers. Vector point, line, and polygon layers are available at a slightly lower level of functionality. When raster maps are uncompressed (using the `r.compress` command), raster data are stored as standard flat image data, with an associated text file containing the image header information. (Flat image format and other image data issues are discussed in the "Competency" section of this document.)

- **Arc/Info**

Arc/Info supports vector point, line, and polygon coverages; shape files (a vector format which allows overlapping polygons); and raster grids. Arc/Info provides a proprietary export format which allows data to be moved between different versions of Arc/Info. This format is used, for example, when converting Arc/Info data from the Unix operating system to the DOS/Windows operating system. Raster data may be converted to flat image format with the `gridimage` command.

- **Idrisi**

Idrisi supports primarily raster images. Vector point, line, and polygon layers are available at a slightly lower level of functionality. Idrisi supports flat image data as one of its native formats. Image data are stored in `.img` files, and header data are provided in `.doc` files.

- **Erdas Imagine**

  Imagine is more of an image processing system than a GIS, and supports primarily raster images. It can, however, read Arc/Info vector files directly.

Software Example (Arc/Info):

Convert a text file of point locations into an Arc/Info point coverage: The Arc/Info GENERATE command reads text files of point, line, and polygon data. In this example, you will import a text file containing point data into Arc/Info, verify the data conversion, and then convert the point data into Arc/Info GRID format.

1) For point data, the GENERATE command expects a text file with three fields per line: An integer point identifier (ID), followed by an X-coordinate, followed by a Y-coordinate. The final line of the text file should contain the word "end". We will use the following text file as the starting point for this exercise:

Text file: foo.gen

```
1 12.1  15.3
2  9.5  23.0
3 99.4  66.99
end
```

2) The following dialog generates a coverage called FOOPTS, using data from the text file "foo.gen":

```
Arc: generate foopts
Copyright (C) 1982-1997 Environmental Systems Research Institute, Inc.
All rights reserved.
GENERATE Version 7.1.1  (Thu Feb  6 23:26:50 PST 1997)

Generate: input foo.gen
Generate: points
Creating points with coordinates loaded from foo.gen
Generate: quit

Externalling BND and TIC...
```

3) Next, we'll build point topology and add the X/Y coordinates to the Point Attribute Table (PAT):

```
Arc: build foopts point
```

```
 Building points...
Arc: addxy foopts
Adding X,Y Coordinates to foopts.PAT
```

4) To verify that the data were correctly converted, list the PAT:

```
Arc: list foopts.pat
Record          AREA    PERIMETER FOOPTS# FOOPTS-ID      X-COORD      Y-
COORD
     1         0.000        0.000     1       1           12.100
 15.300
     2         0.000        0.000     2       2            9.500
 23.000
     3         0.000        0.000     3       3           99.400
 66.990
```

Note that we have three points, with IDs of 1, 2, and 3, and that the X and Y coordinates match those in the "foo.gen" file. A visual check like this a good idea in order to catch bugs that might have been in the original data file, such as missing fields, extra fields, or extraneous characters. Even if you are converting a large amount of data, you should spot-check at least a handful of data values from the beginning, middle, and end of the data file.

5) Suppose that the three points you've converted to Arc/Info represent locations of archaeological dig sites, and that you wish to use these point locations in a raster-based model which runs in the Arc/Info GRID environment. You'll need to convert the point coverage FOOPTS into a grid called FOOGRID. Let's assume that your X/Y coordinate units are meters, and that you want your resulting grid to have a 1-meter cell resolution.

```
Arc: pointgrid foopts foogrid foopts-id
 Converting points from foopts to grid foogrid
Cell Size (square cell): 1
Convert the Entire Coverage(Y/N)?: y
Enter background value (NODATA | ZERO): nodata
Number of Rows    = 53
Number of Columns = 91
```

6) Now verify the grid by listing its Value Attribute Table (VAT):

```
Arc: list foogrid.vat
Record       VALUE      COUNT
     1           1          1
     2           2          1
     3           3          1
```

Note that the grid FOOGRID contains three cells with values 1, 2, and 3. The COUNT field above indicates that there is one instance of each grid cell. The remainder of the grid's cells have the value NODATA.

Competency

Learning Objectives:

After completing this section you should be able to:

- Describe several types of image file formats and data types.
- Understand the concept of byte swapping.

Image data

Image data are published in several different file formats and data types. The *file format* determines the manner in which image pixels are organized. For example, given a two-band image, one file format might store all pixels for band 1 together, followed by all pixels for band 2. Another file format might alternate between band 1 pixels and band 2 pixels. The *data type* of an image determines the manner in which pixel values are stored. For example, one image might store each pixel as a one-byte integer, while another image might store each pixel as a four-byte floating point number. **Image file formats** In general, image data are stored starting at the top-left image corner and following a left-to-right scan order of each row of the image. For single-band images, this format is often called **flat image data**, or a **flat image file**. When there is more than one image band, the bands are organized in one of three ways: band sequential (BSQ), band interleaved by line (BIL), and band interleaved by pixel (BIP).

- **Flat image data (single-band)**

  Pixel data are stored starting at the top-left image corner and following a left-to-right scan order of each row of the image.
- **Band sequential**

  Pixel data for each image band are contiguous: all data for band one are stored first, followed by all of band two, etc.
- **Band interleaved by line**

  Pixel data for each row of each band are contiguous: data for row one of band one are followed by row one of band two, etc. until row one of all bands has been stored. Data for subsequent rows are then stored, band-by-band as for row one.
- **Band interleaved by pixel**

  Pixel one of band one is followed by pixel one of band two, pixel one of band three, etc. No pixels of a given band are stored contiguously.

Image data types

The image data type refers to the manner in which each image pixel is stored in the computer. In general one wants to use the most compact data type possible in order to minimize the storage size of an image. However, the more compact the data type, the smaller the range of values that can be stored. For example, a typical one-byte image uses one byte (8 bits) per pixel. A one-byte number has a range of 2^8, or 256. Thus pixels in a one-byte image

can take on values ranging from 0 to 255. Since these values are all positive numbers, 0-255 is called the *unsigned* range. If a one-byte image needs to store positive and negative numbers, the 256 possible values are split into the *signed* range: -128 to 127.

Ranges of possible pixel values for various image data types

Image data type	Bits per pixel	Unsigned range	Signed range
One-byte integer	8	0 to 255	-128 to 127
Two-byte integer	16	0 to 65,535	-32,768 to 32,767
Four-byte integer	32	0 to 4,294,967,295	-2,147,483,648 to 2,147,483,647
Floating-point (single precision)	32	Not applicable	System-dependent; typically +/- 1^{37} with 6 digits of precision
Floating-point (double precision)	64	Not applicable	System-dependent; typically +/- 1^{128} with 15 digits of precision

Transferring image data to other computer systems

The format of the floating-point data type often varies from one computer system to another. For example, floating-point data on a Macintosh system are typically not readable on a Windows or Unix system. When transferring image data to another computer system, it is best to use integer data types. One-byte integer data tends to be the most stable image format. Two- and four-byte integer data can often be read directly from computer to computer, however certain computer architectures use different formats for storing multi-byte integer data. These formats are known as the **byte order**, and refer to the order in which the individual bytes for a multi-byte integer are stored. The terms **big-endian** and **little-endian** are often used to describe the two types of byte ordering. DOS/Windows systems are typically little-endian while many Unix systems are big-endian. Software tools exist for switching between the two byte orderings, or **byte-swapping**. For example, the Unix system has a built-in command called `dd` which has an option for swapping data bytes. Similarly, the Idrisi GIS has a command called SWAP.

The origin of "endian"

The concept of "big-endian" and "little-endian" comes from Jonathan Swift's book

Gulliver's Travels, in which certain people ate their hard-boiled eggs starting at the small end, and some ate their eggs starting at the big end. The ordering of bytes within an integer is similarly arbitrary.

Image header formats

In addition to the image data, a set of related data is required which stores important attributes of the image such as the number of rows and columns, the data type and file format, and the map projection information. This attribute information is usually known as *header data*, because it is often found at the beginning of an image file. Header data can also be stored as a separate file.

Below is an example of the header data created by the Arc/Info `gridascii` command. The header data contains enough information for a GIS to correctly interpret the number of image rows and columns. The `xllcorner`, `yllcorner`, and `cellsize` parameters allow a GIS to georeference the image to other spatial data.

Example of image header data

```
ncols         1530
nrows         1769
xllcorner     -2416518.9111918
yllcorner     1961603.4472114
cellsize      1000
NODATA_value  -9999
-9999 -9999 -9999 -9999 -9999 ...
```

The `gridascii` command also creates a separate file with map projection information. This information is required to correctly interpret an image's X/Y coordinate data. Here is an example `.prj` file:

Example of image projection information

```
Projection    ALBERS
Zunits        NO
Units         METERS
Spheroid      CLARKE1866
Xshift        0.0000000000
Yshift        0.0000000000
Parameters
 29 30  0.000 /* 1st standard parallel
 45 30  0.000 /* 2nd standard parallel
-96  0  0.000 /* central meridian
 23  0  0.000 /* latitude of projection's origin
0.00000       /* false easting (meters)
0.00000       /* false northing (meters)
```

Software Example (Arc/Info):

Application:

You've just created a floating-point grid which contains the average January temperature over the USA in degrees Celsius. You want to make these data available over the internet, and you don't want to exclude those without access to Arc/Info. You decide to publish the data in a generic flat image format. You'll need to round the floating-point data to integer values. Assume that the floating-point image is called JANFLOAT:

```
janint = con(janfloat >= 0, int(janfloat + 0.5), int(janfloat - 0.5))
```

Since the grid function `int()` truncates values rather than rounding them, we add 0.5 to positive values of JANFLOAT and subtract 0.5 from negative values of JANFLOAT. The above conversion has properly rounded the January temperatures to the nearest integer. However, is this what we want? All JANFLOAT cells with values from 4.500 to 5.499 are given a value of 5 in the JANINT grid. We don't want to lose all of JANFLOAT's floating-point precision when we convert to integer, but we don't necessarily need the full six digits of precision either. Let's assume that our temperature grid is accurate to the nearest one-tenth degree. In order to keep this information in the integer grid, we'll multiply JANFLOAT by 10 before rounding:

```
janint2 = con(janfloat >= 0, int(janfloat * 10 + 0.5), int(janfloat * 10
 - 0.5))
```

Now JANINT2 contains our desired level of precision. Note that the JANINT2 grid has units of tenths-of-a-degree Celsius, while JANINT and JANFLOAT have units of degrees Celsius. The final step is to use the GRIDIMAGE command to convert JANINT2 to a flat image format.

Mastery

Learning Objectives:

After completing this section you should:

- Be aware of some scripting languages for creating custom data conversion tools.
- Have a rudimentary knowledge of the AWK language.

Tools for custom conversion

Sometimes a GIS has no built-in conversion command for a given dataset. When this occurs, the best solution is often to create a custom conversion tool. This can be done with a standard

programming language such as C, C++, Pascal, or Java. These languages allow a maximum amount of freedom and flexibility for highly complex tasks, however they can be cumbersome when used for less complicated tasks. Many data conversion tasks can be done with much simpler scripting languages, which offer less programming flexibility but result in programs (scripts) that are shorter, easier to maintain, and easier to debug. **Scripting languages**

The following are some common scripting languages which are available on most hardware platforms:

- **AWK**

  The AWK program was created by Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan. AWK is a simple scripting language which is designed to process text files on a line-by-line basis. Spatial data are often available in formats which are perfectly suited for the AWK language, such as data from the National Climatic Data Center in which each line of a data file contains information for one measurement station.

  AWK is most commonly found on the Unix operating system, however there are several versions available for DOS/Windows. There is an online awk manual at: *[link removed]*

  An AWK script is a simple text file, where each line consists of a *pattern* followed by an *action*. The pattern determines whether or not to act on the current line of the input file. If the pattern matches the current input line, then the action is performed on that line. Patterns are AWK expressions or Unix *regular expressions* (which are beyond the scope of this document), and actions are AWK expressions which are enclosed in curly braces: {}. When an AWK script is invoked on a given input file, AWK automatically splits each field of the input line into the variables $1, $2, $3, etc.

  Let's say you have a text file with a list of longitude, latitude coordinates:

```
Longitude  Latitude
-120.05     79.99
 123.11     81.23
  20.01    -11.45
 179.88     -0.21
 235.64     22.50
 -37.22     91.00
-111.11     87.23
```

  The following AWK script could be used to extract all points south of the equator:

```
 $2 < 0 {print}
```

The first part of the script, "$2 < 0" is the pattern. It matches all lines of the input file where the second field (latitude, which is automatically set to $2) is less than zero. The action of the script is "{print}", which means to print the current line of the input file. So the output of the script would contain the lines "`20.01 -11.45`" and "`179.88 -0.21`" .

Note that the text file above contains some illegal values for longitude or latitude. The following script would find and print all lines containing illegal values: (Note: "`||`" is the "or" operator.)

```
$1 > 180 || $1 < -180 || $2 > 90 || $2 < -90 {print}
```

Suppose your GIS software imports files only in the lon, lat order rather than lat, lon. The following AWK script will reverse the order of the longitude, latitude fields. Note that by omitting the pattern in the script below, we have told AWK to process every line of the file. The script contains only an action:

```
{print $2, $1}
```

- **PERL - Practical Extraction and Report Language**

Written by Larry Wall, PERL is "an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information." PERL is similar to AWK, but contains far more flexibility and functionality, including complex data structures, the ability to handle binary data, numerous libraries of special functions, and object orientation. Like AWK, PERL was designed for the Unix operating system, however versions are available for DOS/Windows systems. PERL software and documentation are available at *[link removed]*. A detailed introduction to the PERL language is beyond the scope of this document.

Tasks:>

Write an AWK script which converts longitude/latitude coordinates from degrees/minutes/seconds to decimal degrees:

dms2dd.awk

```
# AWK script for converting deg min sec coordinates to decimal degrees.

# Written for NCGIA CCTP By Rusty Dodson, 1/21/98.

# If input file not specified, print usage and exit

BEGIN {                                    # "begin" block executes
 once
```

```
    if (ARGC < 2) {
        printf("\n Usage: awk -f dms2dd.awk infile \n")
        printf("\n Input file format:  lonD lonM lonS latD latM latS
 ID\n")
        exit 1
    }
}

# The following statements execute once for each line of the input
 file.
# Input fields are automatically split into $1, $2, $3 ...

# For readability, assign names to input fields 1-7

  { Xd = $1; Xm = $2; Xs = $3;
    Yd = $4; Ym = $5; Ys = $6;
    id = $7
  }

# Convert longitude; account for negative degrees

  { if (Xd >= 0)
      lon = Xd + (Xm / 60) + (Xs / 3600)
  }
  { if (Xd < 0)
      lon = Xd - (Xm / 60) - (Xs / 3600)
  }

# Convert latitude; account for negative degrees

  { if (Yd >= 0)
      lat = Yd + (Ym / 60) + (Ys / 3600)
  }
  { if (Yd < 0)
      lat = Yd - (Ym / 60) - (Ys / 3600)
  }

# Print the results

  { printf("%12.6f %12.6f %15s\n", lon, lat, $7) }
```

Given the following text file (dms.dat) of degree/minute/second coordinate data:

dms.dat

```
-149 54  1 61 13  5 Anchorage
  80 11 38 25 46 26 Miami
-119 41 50 34 25 15 SantaBarbara
```

Running the conversion script:

```
awk -f dms2dd.awk dms.dat
```

produces the following output:

```
-149.900278    61.218056       Anchorage
  80.193889    25.773889           Miami
-119.697222    34.420833    SantaBarbara
```

NOTE: the script can be written concisely as:

Concise version of `dms2dd.awk`

```
# dms2dd.awk
  { printf("%12.6f %12.6f %15s\n",
      $1 >= 0 ?  $1 + ($2 / 60) + ($3 / 3600) :  $1 - ($2 / 60) - ($3 /
 3600),
      $4 >= 0 ?  $4 + ($5 / 60) + ($6 / 3600) :  $4 - ($5 / 60) - ($6 /
 3600),
      $7)
  }
```

Write a PERL script to convert signed integer image data to floating-point:

The following PERL script was written because, at the time, Arc/Info could not properly read signed image data. The `int2flt.pl` script was a quick-and-dirty solution to that problem. By converting signed integer image data to floating-point, the Arc/Info command FLOATGRID could be used to import the signed image data.

Extra comments were added to the PERL code below, yet much of it is cryptic to those unfamiliar with the language. The point of including it here is to demonstrate that a fairly complex data conversion can be implemented with just 9 lines of PERL code.

`int2flt.pl`

```
#!/usr/local/bin/perl5 -w
# By Rusty Dodson, 08/27/96.
#
# Convert signed 2-byte integer image to 4-byte float:

# set buffer size, initialize input buffer:

$bufsiz = 1024;
$inbuf = "";

while (1) {                                              # infinite
 loop
```

```
   # read a chunk of data from the input data stream, remembering
number
   # of bytes read ($n_in):

   $n_in = read(STDIN, $inbuf, $bufsiz);

   # use "unpack" to convert integer data to the array "@inval":

   @inval = unpack("s*", $inbuf);   # s = signed short integer

   # "pack the @inval array into a chunk of floating-point data:

   $outbuf = pack("f*", @inval);    # pack as float

   # write the float chunk to the output stream:

   $n_out = syswrite(STDOUT, $outbuf, $n_in * 2);      # outbytes is
2*inbytes

   print STDERR "read $n_in, wrote $n_out bytes...\n";

   # when end-of-file is reached, break the infinite loop:

   exit 0 if(eof(STDIN));
}
```

Follow-up Units

- Project management (UNIT 52)
- Communicating about and distributing GIS products (UNIT 53)

Resources

[Outdated links have been removed.]

- On-line AWK manual
- PERL site
- USGS EROS Data Center
  Contains information on DLG data, DEM data, land use/land cover data, hydrologic data, and more.
- SDTS frequently asked questions list
- US Census Bureau
- National Spatial Data Infrastructure

Created: May 14, 1997. Last updated: October 5, 1998.