Point Line Poly

A Technical Friend for PC ARC/INFO® Users
And Other Desktop GIS Software

Volume **7** Number **5**

Inside:

Icon Key

PC ARC/INFO

ArcView

Idrisi

ArcView 3.2

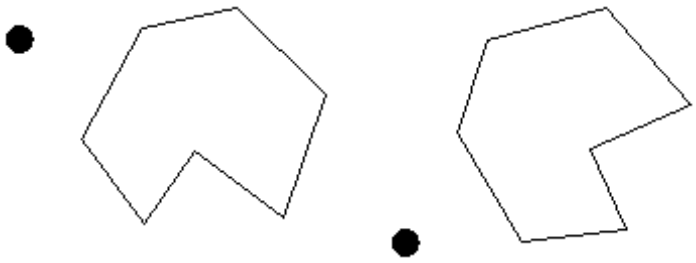
Version 3.2 has been shipping since September. In addition to fixing a number of nasty bugs[1], some significant new functionality has been added.

New Avenue Classes

The Transform2D class allows moving, scaling, rotating, and mirroring shapes. For example, the following code rotates a shape about an arbitrary axis point:

```
theTrans = Transform2D.Make
theTrans.Move(0@0 - AxisPoint)
theTrans.Rotate(theAngle)
theTrans.Move(AxisPoint)
newShape = oldShape.Transform(theTrans)
```

Before: After:



Note that the rotation angle is in degrees. Use oldShape.ReturnCenter as the axis point to rotate the shape about its center. The online help under "Transform2D (Class)" contains additional examples.

Another new Avenue class is VTabSort, which allows you to sort tables on multiple columns and export the results:

```
vs = VTabSort.Make(theVTab,flist,slist,false,true)
if (ExportSorted) then
   vs.Export(out_fn,dBASE)
   return nil
end
```

See "sorttab.ave" in the PLP OnLine Code Pack for a more complete example.

(Continued on page 2)

Other Avenue improvements include additional geocoding requests, storing environment variables with projects, and setting densification parameters for reprojection.

Projection Utility

This standalone utility, developed using VB and MapObjects[2], supports a number of standard projections and datum transformation methods including NADCON (sorry, no CNT). Users also have the ability to define custom projections: see the utility's online help FAQ for an example of defining Albers.

One useful feature is the ability to associate projection definition (.prj) files with a shapefile, thus providing an essential source of metadata commonly lacking in shapefile datasets.

The utility has a wizard interface, but a "quiet" option is available which allows running the application from a command prompt. For example, the following command will convert a shapefile from State Plane NAD27 feet to UTM NAD83 meters[3]:

```
projutil -Q -ID . -OD . -IF az_sp.shp -
OF az_utm.shp -IC
NAD_1927_Arizona_Central -OC
NAD_1983_UTM_Zone_12N -IG 108001
```

Although very slow and cumbersome in execution, the utility should be useful for AV users lacking access to PC ARC/INFO or DAK.

SDTS Translators

The SDTS Raster to Grid translator allows the creation of grids from DEM files. I haven't been able to get the interface to work with full pathnames, but the application may be run from AV's BIN32 directory at the command prompt (having the BIN32 directory in PATH doesn't suffice):

```
sdtsr2g c:\0home\test\plp\dem\7332 c:
\0home\test\plp\demgrid
```

SDTSP2A converts NGS control point data in SDTS format[4] to an ARC/INFO coverage:

(Continued on page 3)

POINT LINE POLY is published by Pierssen Publishing, 3125 West Wilson Drive, Flagstaff, AZ USA 86001. Electronic subscriptions: 6 issues $10 in US funds. Single Issues $2. E-mail and Web access is required. Send subscriptions and fulfillment questions to *PLP*, piersen@primenet.com. Phone/FAX (520) 774-7905. Or mail to the above address. Submit material and technical questions to The Editor, at one of the above addresses.

ISSN: 1099-2324

POSTMASTER: Send address changes to: Point Line Poly
Pierssen Publishing
3125 West Wilson Drive
Flagstaff, AZ USA 86001
e-mail: piersen@primenet.com

Say, what? "It's possible for one never to transgress a single law and still be a bastard. And vice versa. Actually it's only a question of convenience." Hermann Hesse, *Demian*

Binary File I/O

HACKER'S CORNER

Because Avenue does not support binary file I/O, it is necessary to develop external code to do so. The most convenient method is to create a DLL to handle requests from Avenue. "Avbinio.dll", in the PLP OnLine code pack, includes the following routines:

```
FILE * OpenRead(char * path);
FILE * OpenWrite(char * path);
int ReadByte(FILE * stream);
int ReadInt16(FILE * stream);
int ReadInt32(FILE * stream);
int ReadFloat(FILE * stream);
int ReadDoubleAsFloat(FILE * stream);
int GetIntVal();
float GetFloatVal();
int WriteByte(FILE * stream, int val);
int WriteInt16(FILE * stream, int val);
int WriteInt32(FILE * stream, int val);
int WriteFloat(FILE * stream, float
val);
int WriteFloatAsDouble(FILE * stream,
float val);
int Close(FILE * stream);
```

OpenRead and OpenWrite return file handles:

```
FILE * OpenRead(char * path)
{
   return fopen(path,"rb");
}
```

Routines to read data return a status value (0 if successful, -1 if null file handle, -2 if EOF, -9 if read error) and store the value read in a global variable:

```
int ReadByte(FILE * stream)
{
   unsigned char B[1];
   size_t f_read;
   if (stream == NULL)
      return -1;
   f_read = fread(B,sizeof(unsigned
char),1,stream);
   if (ferror(stream))
      return -9;
   if (feof(stream))
```

(Continued on page 4)

(Continued from page 2)

```
sdtsp2a c:\0home\test\plp\ngs\plp1 c:
\0home\test\plp\ngscov
```

Note that attribute tables are not automatically joined to the PAT.

Conclusion

Other features of 3.2 include an upgrade of Crystal Reports to version 7, some new data format viewers, and upgrades and patches to certain extensions[5]. Also, the ESRI Data & Maps for the USA have been converted from NAD27 to NAD83. Additional information is available in the online help, under "What's new in ArcView GIS 3.2".

Overall, 3.2 offers a fair amount of new functionality for a modest price. If you have 3.1, the upgrade is worthwhile for the bug fixes alone. And if you've been holding off upgrading from 3.0, now's the time to do so.

[1]The line symbol cap and join problem has not yet been fixed, though ESRI has identified it as a known issue. Also, Polygon.ReturnCenter still doesn't return the true centroid, though the online help now explains the method used to obtain the point.
[2]The projection utility is only available to Windows users. Some utilities, such as the SDTS translators, are available to Windows and Solaris users.
[3]An Access table containing all the EPSG names and codes may be downloaded at this site: **http://www.petroconsultants.com/products/geodetic.html**
[4]Available at **http://www.ngs.noaa.gov/datasheet.html**
[5]The CAD reader upgrade to support AutoCAD 2000 format wasn't shipped with the CD, but should be made available for download when ready. Check **www.esri.com** for more info.

PLP

(Continued from page 3)

```
      return -2;
   IntVal = (int) B[0];
   return 0;
}
```

To retrieve the value read, use GetIntVal or GetFloatVal:

```
int GetIntVal()
{
   return IntVal;
}
```

The Write routines return 0 if successful, -1 if null file handle, and -9 if write error:

```
int WriteByte(FILE * stream, int val)
{
   unsigned char B[1];
   B[0] = (unsigned char) val;
   size_t f_write;
   if (stream == NULL)
      return -1;
   f_write = fwrite(B,sizeof(unsigned
char),1,stream);
   if (ferror(stream))
      return -9;
   return 0;
}
```

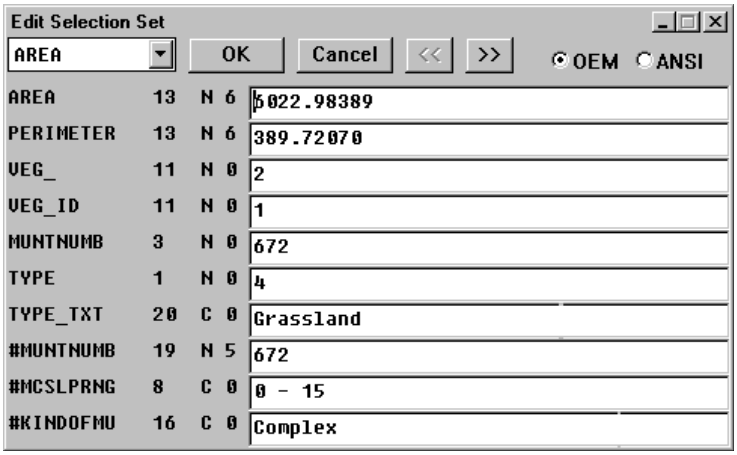
Note the ReadDoubleAsFloat and WriteFloatAsDouble routines; these exist because Avenue does not support a Double DLLPROC_TYPE, and should be used with care. The following script is an example of reading and writing byte values in Avenue using the avbinio DLL:

```
'**** define DLLProcs

bioDLLName = "$AVBIN\avbinio.dll".
AsFileName
bioDLL = DLL.Make(bioDLLName)
bioOpenRead = DLLProc.Make
(bioDLL,"OpenRead",
#DLLPROC_TYPE_POINTER,
   {#DLLPROC_TYPE_STR})
bioOpenWrite = DLLProc.Make
(bioDLL,"OpenWrite",
#DLLPROC_TYPE_POINTER,
   {#DLLPROC_TYPE_STR})
bioReadByte = DLLProc.Make
(bioDLL,"ReadByte",#DLLPROC_TYPE_INT32,
   {#DLLPROC_TYPE_POINTER})
bioGetIntVal = DLLProc.Make
(bioDLL,"GetIntVal",
#DLLPROC_TYPE_INT32,{})
bioWriteByte = DLLProc.Make
(bioDLL,"WriteByte",
#DLLPROC_TYPE_INT32,
   {#DLLPROC_TYPE_POINTER,
#DLLPROC_TYPE_INT32})
bioClose = DLLProc.Make(bioDLL,"Close",
#DLLPROC_TYPE_INT32,
   {#DLLPROC_TYPE_POINTER})

'**** write test file

vlist = {37,42,45,0,3}
tname = FileName.GetTmpDir.MakeTmp
("test","bin").GetFullName

fh = bioOpenWrite.Call({tname})
for each v in vlist
   result = bioWriteByte.Call({fh,v})
   if (result = -1) then
      MsgBox.Error("Error opening file
for write","")
      return nil
   elseif (result = -9) then
      MsgBox.Error("File write
error","")
      return nil
   end
end
result = bioClose.Call({fh})
if (result <> 0) then
   MsgBox.Error("Error closing write
file","")
   return nil
end

'**** read test file

vlist2 = List.Make

fh = bioOpenRead.Call({tname})
while (true)
   result = bioReadByte.Call({fh})
   if (result = -1) then
      MsgBox.Error("Error opening file
for read","")
      return nil
```

Cursors and Forms

In **V7N4** we discussed emulating cursor functionality with the aid of an external application that stores record numbers in memory. Now let's look at how we can apply this concept to a version of forms that scrolls through a selection set in ARCEDITW or ARCPLOTW. First we need to design a dialog box:

```
BEGIN    0  4.0  0 17.0 60 v Edit
Selection Set
DROP  -101  0.0  0 11.0 14 f
PBUT  -102  0.0 15  1.5  8 v OK
PBUT  -103  0.0 24  1.5  8 v
Cancel
PBUT  -104  0.0 33  1.5  4 v <<
PBUT  -105  0.0 38  1.5  4 v >>
RBUT  -106  0.5 45  1.0  6 v OEM
RBUT  -107  0.5 52  1.0  6 v ANSI
LTEXT  111  2.0  0  1.5 20 f
EBOX   112  2.0 20  1.5 40 f
LTEXT  113  3.5  0  1.5 20 f
EBOX   114  3.5 20  1.5 40 f
LTEXT  115  5.0  0  1.5 20 f
EBOX   116  5.0 20  1.5 40 f
LTEXT  117  6.5  0  1.5 20 f
EBOX   118  6.5 20  1.5 40 f
LTEXT  119  8.0  0  1.5 20 f
EBOX   120  8.0 20  1.5 40 f
LTEXT  121  9.5  0  1.5 20 f
EBOX   122  9.5 20  1.5 40 f
LTEXT  123 11.0  0  1.5 20 f
EBOX   124 11.0 20  1.5 40 f
LTEXT  125 12.5  0  1.5 20 f
EBOX   126 12.5 20  1.5 40 f
LTEXT  127 14.0  0  1.5 20 f
EBOX   128 14.0 20  1.5 40 f
LTEXT  129 15.5  0  1.5 20 f
EBOX   130 15.5 20  1.5 40 f
```



Note the addition of arrow buttons that allow the user to navigate through a selection set. The example routines in the PLP OnLine Code Pack have been developed using GUITOOL directives[1], and make certain assumptions about variables set beforehand. File "cforms.inc" contains variable declarations common to all routines:

(Continued on page 6)

(Continued from page 4)

```
   elseif (result = -2) then
      '**** EOF reached
      break
   elseif (result = -9) then
      MsgBox.Error("File read
error","")
      return nil
   end
   result = bioGetIntVal.Call({})
   vlist2.Add(result)
end
result = bioClose.Call({fh})
if (result <> 0) then
   MsgBox.Error("Error closing read
file","")
   return nil
end

theMsg = "Values read:"
for each v in vlist2
   theMsg = theMsg + NL + v.AsString
end
MsgBox.Report(theMsg,"")
File.Delete(tname.AsFileName)
```

Examples of reading and writing other value types are in the PLP OnLine Code Pack.

PLP

(Continued from page 5)

```
&define i -8 &var
&define j -9 &var
&define wksp -18 &var
&define temp -19 &var
&define opt 2 &var
&define program 3 &var
&define cover 4 &var
&define feature 5 &var
&define arraynum 6 &var
&define curpos 7 &var
&define pointer 8 &var
&define numrec 9 &var
&define numit 500 &var
&define origin_it 501
&define origin_val 601
```

Routine "cforms", the main routine for the application, assumes that variable [opt] has been set to GLOBAL (for calculating values of all records in the set) or CURSOR (to edit values one record at a time). Variable [program] is set to ARCEDITW or ARCPLOTW; if ARCPLOTW, then [cover] must be set to the name of the coverage and [feature] to the appropriate feature type (POINTS, ARCS, or POLYS). Variable [numit] contains the number of items, and [origin_it] is the start variable for the list of item definitions[2].

```
*AUTHOR PLP
*ROUTINE cforms
c_forms.dlg 1

&define oemtoggle -11 &var

&if &eq CURSOR [opt] &do
   &r cf_open
   &rv [temp]
   &if &eq .FALSE. [temp] &do
      &return .FALSE.
   &end
   &sv [curpos] 0
&end

&rem **** initialize dialog box

&r cf_load
&openw [winfile]
*W L 101 ,
&sv [i] 1
&while &rn [i] 1 [numit] &do
   &cv [j] [i] + [origin_it] - 1
   &extract [temp] %[j] 1
   *W [temp]
   &inc [i]
&end
*W ,
&sv [pointer] [origin_it]
&extract [temp] %[pointer] 1
*W S 101 [temp]
&r cf_init 106
&closew

&rem **** put up dialog box and process
resp

&sv [oemtoggle] 0
*OPEN c_forms.dlg
*PICK 101
   &r cf_store
   &sv [i] 1
   &while &rn [i] 1 [numit] &do
      &extract [temp] %<[i] +
[origin_it] - 1> 1
      &if &eq [temp] %101 &do
         &cv -1 [i] + [origin_it] - 1
         &break
      &end
      &inc [i]
   &end
   &value [pointer] -1
*PICK 102 50
   &r cf_calc
   *CLOSE
*PICK 103 49
   *CLOSE
*PICK 104
   &r cf_calc
   &r cf_curs DEC
   &dec [curpos]
   &r cf_load
*PICK 105
   &r cf_calc
   &r cf_curs INC
   &inc [curpos]
   &r cf_load
*PICK 106
   &r cf_store
   &sv [oemtoggle] 106
*PICK 107
   &r cf_store
   &sv [oemtoggle] 107
*ENDPICK ALL
   &if &ne [oemtoggle] 0 &do
      WIN DB P
```

```
      &value [temp] [winrtn]
      WIN DB D 1
      *OPEN c_forms
      WIN DB [temp]
   &end
   &openw [winfile]
   &r cf_init [oemtoggle]
   &closew
   &sv [oemtoggle] 0
*ENDPICK
&if &eq CURSOR [opt] &do
   &r cf_close
&end
&return .TRUE.
```

The first step, if the CURSOR option is desired, is to set up the cursor. The steps used in routine "cf_open" are similar to those developed in the previous issue:

```
*ROUTINE cf_open

&rem **** create cursor

&value [wksp] WKSP
&if &eq ARCEDITW [program] &do
   SHOW NUMBER SELECT [numrec]
&else
   &value [temp] [feature] 1 %<len
[feature] - 1>
   WIN SEL W [cover] [temp] [wksp]t
$sel.lis
   SHOW RESELECT [numrec] 0
&end
&if &eq [numrec] 0 &do
   &type "ERROR: No records in
selection set"
   &if &eq ARCPLOTW [program] &do
      & DEL [wksp]t$sel.lis
   &end
   &return .FALSE.
&end
&value [temp] ARC
WIN PATH [temp]\APPS
&type "Starting server..."
WIN RUN ah
&type "Initializing cursor..."
WIN RUNW arrayf U
WIN RUNW arrayf A [numrec]
WIN CB R
&value [arraynum] 1
&if &eq ARCEDITW [program] &do
   &sv [i] 1
   &openw [wksp]t$sel.lis
   &while &rn [i] 1 [numrec] &do
      SHOW SELECT %[i] [temp]
      &write [temp]
      &inc [i]
   &end
   &closew
   WIN RUNW arrayf F [arraynum] [wksp]t
$sel.lis
   & DEL [wksp]t$sel.lis
   SHOW SELECT 1 [i]
   SEL $RECNO IN {[i]}
&else
   WIN RUNW arrayf W [arraynum] [wksp]t
$sel.lis
   WIN RUNW arrayf G [arraynum] 0
   WIN CB R
   &value [i] 1
   RESELECT [cover] [feature] $RECNO IN
{[i]}
   &r cf_flash SELECT
&end
&return .TRUE.
```

Executables "ah.exe" and "arrayf.exe" are assumed to reside in the %ARC%\apps directory. The array server is started, an array is allocated, the array is populated with record numbers, and the first record in the set is selected.

The next step is to initialize the dialog box. Routine "cf_load" loads item values into a storage area starting at [origin_val]:

```
*ROUTINE cf_load

&rem **** load item values

&if &eq ARCEDITW [program] &do
   SHOW SELECT 1 -11
   SHOW EDITFEATURE -12
&end
&sv [i] 1
&while &rn [i] 1 [numit] &do
   &extract -1 %<[i] + [origin_it] - 1>
1
   &extract -2 %<[i] + [origin_it] - 1>
3
   &if &eq ARCEDITW [program] &do
      SHOW %-12 %-11 ITEM %-1 [temp]
   &else
      &if &eq %-2 C &do
         MOVEITEM %[cover] %[feature]
```

```
%-1 TO [temp]
      &else
         CALC %[cover] %[feature]
[temp] = %-1
      &end
   &end
   &if &eq %-2 D &do
      &rem **** reformat date value
      &value -1 [temp] 3 4
      &value -2 [temp] 5 6
      &value -3 [temp] 7 8
      &sv [temp] "%-2/%-3/%-1"
   &end
   &value %<[i] + [origin_val] - 1>
[temp]
   &inc [i]
&end
&return
```

DROP widget 101 is loaded with a list of item names, terminated with a comma, and is set to the first item. Routine "cf_init" sets the remaining widgets:

```
*ROUTINE cf_init

&define oemtoggle -11 &var
&value [oemtoggle] -1

&rem **** enable/disable arrows

&if &eq [opt] CURSOR &do
   &if &eq [curpos] 0 &do
      *W G 104
      *W E 105
   &elseif &eq [curpos] %<[numrec] - 1>
&do
      *W E 104
      *W G 105
   &else
      *W E 104
      *W E 105
   &end
&else
   *W G 104
   *W G 105
&end

&rem *** OEM/ANSI toggle

&if &ne [oemtoggle] 0 &do
   *W S [oemtoggle] 1
&end

&rem **** set LTEXT and EBOX values

&sv [i] 1
&while &rn [i] 1 10 &do
   &cv [temp] [origin_it] + [numit] - 1
   &cv [j] [i] + [pointer] - 1
   &cv -5 109 + ( [i] * 2 )
   &cv -6 %-5 + 1
   &if &rn [j] [pointer] [temp] &do
      &value [temp] %[j]
      &extract -13 [temp] 1
      &extract -14 [temp] 2
      &extract -15 [temp] 3
      &extract -16 [temp] 4
      &sv [temp] 0 "(I1,T1,'%-13',
T13,'%-14',T17,'%-15',T19,'%-16')"
      *W S %-5 [temp]
      &if &eq %-15 C &do
         *W B %-6 %-14 2
      &elseif &eq %-15 D &do
         *W B %-6 %-14 1
      &elseif &eq %-16 0 &do
         &if &rn %-14 1 10 &do
            *W B %-6 %-14 3
         &else
            *W B %-6 10 3
         &end
      &else
         &if &rn %-14 3 20 &do
            *W B %-6 %-14 4
         &else
            *W B %-6 20 4
         &end
      &end
      &value -18 %<[j] + [origin_val] -
[origin_it]>
      &if &eq [oemtoggle] 106 &do
         *W T %-6 OEM
      &elseif &eq [oemtoggle] 107 &do
         *W T %-6 ANSI
      &end
      *W S %-6 %-18
      *W E %-6
   &else
      *W S %-5
      *W S %-6
      *W G %-6
   &end
   &inc [i]
&end
*W F 112
&return
```

The arrow button widgets are enabled or disabled depending on the forms option and the position in the selection set. The OEM/ANSI toggle is set if needed, and finally the LTEXT and EBOX widgets are populated from the item definition and item value lists.

For most return widget options, values in the EBOX widgets are stored back to the value list before anything further is done:

```
*ROUTINE cf_store

&rem **** store dialog attribute values

&cv -1 [origin_val] - [origin_it]
&sv [i] 1
&while &rn [i] 1 10 &do
   &cv [temp] [origin_it] + [numit] - 1
   &cv [j] [i] + [pointer] - 1
   &if &rn [j] [pointer] [temp] &do
      &value %<[j] + %-1> %<110 + ( [i]
* 2 )>
   &end
   &inc [i]
&end
&return
```

If an item is picked in the DROP widget, variable [pointer] is altered accordingly and the dialog reinitialized. Clicking OK or the <Enter> key performs a calculation before closing the dialog, whereas the CANCEL or <Esc> key skips the calculation step. Routine "cf_calc" recalculates the items based on the values put back into the value list:

```
*ROUTINE cf_calc

&rem **** calculate attributes

&r cf_store
&sv [i] 1
&while &rn [i] 1 [numit] &do
   &value -1 %<[i] + [origin_it] - 1>
   &value -2 %<[i] + [origin_val] - 1>
   &extract -3 -1 1
   &extract -4 -1 3
   &value -5 -2
   &if &eq ARCEDITW [program] &do
      &if &eq "%-4" "C" &do
         MOVEITEM '%-5' TO %-3
      &else
         CALC %-3 = %-5
      &end
   &else
      &if &eq "%-4" "C" &do
         MOVEITEM %[cover] %[feature]
'%-5' TO %-3
      &else
         CALC %[cover] %[feature] %-3 =
%-5
      &end
   &end
   &inc [i]
&end
&return
```

If an arrow button is clicked, the current record is recalculated and the next or previous record selected and item values reloaded. Routine "cf_curs" performs the cursor operation:

```
*ROUTINE cf_curs

&define cur -1 &var

&rem **** select next or previous
record

&if &eq DEC [cur] &do
   &cv [temp] [curpos] - 1
&else
   &cv [temp] [curpos] + 1
&end
WIN RUNW arrayf G [arraynum] [temp]
WIN CB R
&value [i] 1
&if &eq ARCEDITW [program] &do
   SHOW SELECT 1 [temp]
   WIN RUNW arrayf S [arraynum]
[curpos] [temp]
   SEL $RECNO IN {[i]}
&else
   &r cf_flash UNSELECT
   ASEL [cover] [feature]
   RESELECT [cover] [feature] $RECNO IN
{[i]}
   &r cf_flash SELECT
&end
&return
```

Note that in the case of ARCEDITW the updated record number is put back in the array before retrieving the next or previous record number.

The OEM/ANSI toggle requires that the dialog box be destroyed and recreated. Note that pin mode is temporarily set to preserve the position of the dialog box. Finally, routine "cf_close" restores the selection set and closes and removes the cursor:

```
*ROUTINE cf_close

&rem **** restore selection set

&if &eq ARCEDITW [program] &do
   &sv [i] 0
   &while &rn [i] 0 %<[numrec] - 1> &do
      WIN RUNW arrayf G [arraynum] [i]
      WIN CB R
      &value [j] 1
      ASEL $RECNO IN {[j]}
      &inc [i]
   &end
&else
   &value [wksp] WKSP
   &value [temp] [feature] 1 %<len
[feature] - 1>
   WIN SEL R [cover] [temp] [wksp]t
$sel.lis
   & DEL [wksp]t$sel.lis
   &r cf_flash SELECT
&end

&rem **** remove cursor

&type "Removing cursor..."
WIN RUNW arrayf R [arraynum]
WIN RUNW arrayf D
WIN RUNW arrayf N
WIN CB R
&value [i] 1
&if &eq [i] 0 &do
   &type "Closing server..."
   WIN RUNW arrayf C
&end
&return
```

In ARCPLOTW, routine "cf_flash" is used to indicate the currently selected feature:

```
*ROUTINE cf_flash

&rem **** draw selected records

&define option -1 &var
&define symbol -2 &var
&if &eq SELECT [option] &do
   &sv [symbol] 5
&else
   &sv [symbol] 8
&end
&if &eq POINTS [feature] &do
   MARKERSET COLOR.MRK
   POINTMARKERS [cover] [symbol]
&elseif &eq ARCS [feature] &do
   LINESET COLOR.LIN
   ARCLINES [cover] [symbol]
&elseif &eq POLYS [feature] &do
   SHADESET COLOR.SHD
   POLYGONSHADES [cover] [symbol]
&end
&return
```

Additional routines in "cforms.gui" — "aeforms", "apforms", and "cf_items" — are used to implement command line versions of the "cforms" routine. See the PLP OnLine Code Pack for the code.

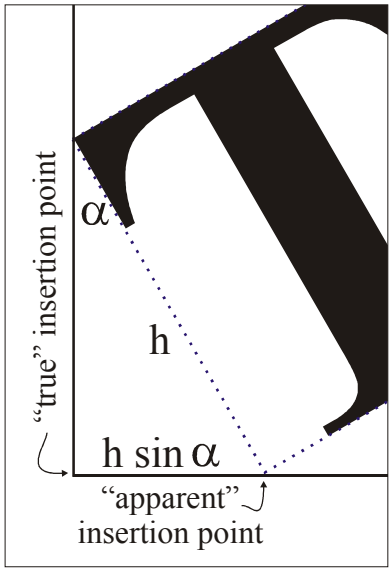
---

[1]See **V6N4** and **V6N5** for articles discussing GUITOOL.
[2]Routine GETITEM in **V6N5** and cf_items in the PLP Online Code Pack for this issue give examples of routines to populate the item list.

PLP

Rotating GraphicText

Rotating GraphicText objects in ArcView is not as simple a task as it may seem. As the following figure illustrates, when a GraphicText object has a nonzero angle its "apparent" insertion point is not the same as the actual one:



Thus, when rotating graphic text, it's important to determine the difference between the origin of the apparent (rotated) bounding box and that of the actual bounding box. The following script will calculate the difference to a sufficiently accurate degree for placement:

```
'GTOffset
'Get offset correction for GraphicText

'Arguments:
' g = theGraphicText
' sf = text scale factor
' (display units per point of text)
'Returns:  point

g = SELF.Get(0)
sf = SELF.Get(1)

a0 = g.GetAngle
while (a0 >= 180)
   a0 = a0 - 360
end
while (a0 <= -180)
   a0 = a0 + 360
end
if (a0 = 0) then return (0@0) end

x0 = g.GetBounds.GetLeft
y0 = g.GetBounds.GetBottom
sz = g.GetSymbol.GetSize * sf
nml = g.GetText.AsTokens(nl).Count
sp = g.GetSpacing
sz = sz + ((nml - 1) * sp * sz)
dx = 0
dy = 0
if ((a0 >= 90) or (a0 < -90)) then
   dx = g.GetBounds.GetWidth.Negate
end
if (a0 < 0) then
   dy = g.GetBounds.GetHeight.Negate
end
SinOffset = sz * a0.AsRadians.Sin
CosOffset = sz * a0.AsRadians.Cos
if ((a0 > 0) and (a0 < 90)) then
   dx = dx - SinOffset
elseif (a0 > 90) then
   dy = dy + CosOffset
elseif ((a0 < 0) and (a0 > -90)) then
   dy = dy + CosOffset
elseif (a0 < -90) then
   dx = dx - SinOffset
end
return (dx@dy)
```

Note the requirement of a scale factor equal to the number of display units per text point size. For layouts this should be 1/72, but for views this may be calculated as:

```
sf = theView.ReturnScale / ppu
```

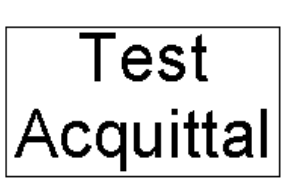
Where "ppu" is the number of points per View unit. The following script will rotate selected GraphicShape and GraphicText objects about a user-defined point in either a View or Layout:

```
theTitle = "Rotate Graphic Shapes and
Text"
d = av.GetActiveDoc
AxisPoint = d.GetDisplay.
ReturnUserPoint
if (d.Is(View)) then
   theUnits = d.GetUnits.AsString.
```

```
Substitute("_"," ").Extract(2)
   if (theUnits = "UNKNOWN") then
      MsgBox.Info("View map units must
be defined",theTitle)
      exit
   elseif (theUnits = "FEET") then
      ppu = 72 * 12
   elseif (theUnits = "METERS") then
      ppu = 72 * 100 / 2.54
   else
      theMsg = "Your map units are not
supported."+NL+
               "Please add a points per
unit"+NL+
               "conversion to this
script."
      MsgBox.Info(theMsg, theTitle)
      exit
   end
   sf = d.ReturnScale / ppu
elseif (d.Is(Layout)) then
   sf = 1 / 72
else
   theMsg = "Active document must be
Layout or View."
   MsgBox.Info(theMsg, theTitle)
   return nil
end
a = MsgBox.Input("Input angle:",
theTitle,"0")
if (a = nil) then
   return nil
elseif (a = "0") then
   return nil
elseif (a.IsNumber.Not) then
   return nil
else
   theAngle = a.AsNumber
end
theTrans = Transform2D.Make
theTrans.Move(0@0 - AxisPoint)
theTrans.Rotate(theAngle)
theTrans.Move(AxisPoint)

gl = d.GetGraphics
for each g in gl.GetSelected
   g.Invalidate
   if (g.Is(GraphicShape)) then
      s1 = g.GetShape
      s2 = s1.Transform(theTrans)
      g.SetShape(s2)
   elseif (g.Is(SplineText)) then
      '**** not supported ****
   elseif (g.Is(GraphicText)) then
      x = g.GetBounds.GetLeft
      y = g.GetBounds.GetBottom
      a0 = g.GetAngle
      d = av.Run("GTOffset",{g,sf})
      p0 = x@y - d
      s = g.GetSymbol.Clone
      sp = g.GetSpacing
      al = g.GetAlignment
      t = g.GetText
      a1 = theAngle + a0
      gl.RemoveGraphic(g)
      p1 = p0.Transform(theTrans)
      g = GraphicText.Make(t,p1)
      g.SetSpacing(sp)
      g.SetAlignment(al)
      g.SetSymbol(s)
      g.SetAngle(a1)
      gl.Add(g)
      d = av.Run("GTOffset",{g,sf})
      g.Offset(d)
      g.SetSelected(true)
   end
   g.Invalidate
end
```

Note the use of the Transform2D class, which is new in ArcView 3.2. Also note that the script does not rotate SplineText entities; this is because the defining PolyLine objects are not retrievable in Avenue.

Before:



After:

PLP

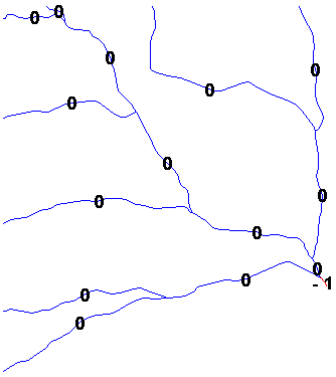
Stream Ordering

Routine "st_order" may be executed in ARCPLOT/W to calculate the stream order of an arc coverage.

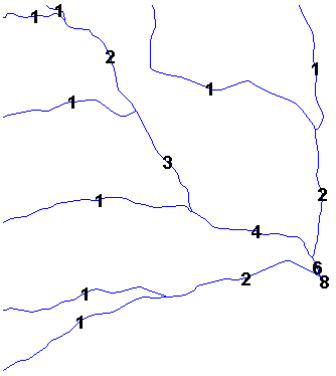
```
Usage: &r ST_ORDER [cover] [order_item]
{SHREVE/STRAHLER}
```

By default, Shreve's method is used:

Before:



After:



In order for the routine to work, certain conditions must be met:

1) the coverage has clean arc and node topology (no overlaps)
2) every arc's to_node is downstream from its from_node
3) the only dangling from_nodes are on order 1 streams
4) the only dangling to_node is on the arc furthest downstream
5) no braids or splits exist
6) the arc furthest downstream has its order set to -1
7) all other arcs have their order set to some other value

The routine uses a few global variables:

st_order.inc:

```
&define cover 2 &var
&define order_it 3 &var
&define option 4 &var
&define node 5 &var
&define order 6 &var
&define first -17 &var
&define wksp -18 &var
&define temp -19 &var
&define opt1 SHREVE
&define opt2 STRAHLER
```

The first part of the routine finds the arc furthest downstream:

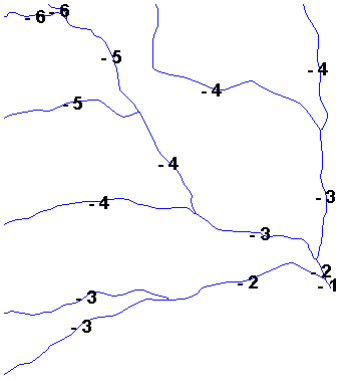
```
&routine st_order

&include st_order.inc

&define i -11 &var
&define rec -12 &var

&if &eq "x%-1" "x" &do
   &delim < >
   &type "Usage: &r ST_ORDER [cover]
[order_item] {<opt1>/<opt2>}"
   &delim [ ]
   &return
&end
&extract [cover] -1 1
&extract [order_it] -2 1
&if &eq "x%-3" "x" &do
   &sv [option] [opt1]
&else
   &extract [option] -2 1
&end
&value [wksp] WKSP
LINESET COLOR.LIN
CLEARSELECT
RES [cover] ARC [order_it] = -1
SHOW RESELECT [temp] 0
```

```
&if &ne [temp] 1 &do
   &type "One arc in stream network
(the furthest downstream)"
   &type "must have its order set to -
1"
   &return
&end
```

Then, the network is traced upstream. As each arc is encountered, a negative number is assigned to its level in the tree:

```
&type "Tracing stream network..."
LINESYMBOL 5
&sv [i] -1
&while &do
   ARCS [cover]
   CALC %[cover] ARC %[order_it] = %[i]
   &openw [wksp]t$node.lis
   LIST [cover] ARC FNODE_
   &closew
   CLEARSELECT
   &sv [first] .TRUE.
   &open [wksp]t$node.lis error
   &while &do
      &read [temp] [break]
      &extract [node] [temp] 1
      &if &nm [node] &do
         &if &eq [first] .TRUE. &do
            RES [cover] ARC TNODE_ =
[node]
            &sv [first] .FALSE.
         &else
            ASEL [cover] ARC TNODE_ =
[node]
         &end
      &end
   &end
   &close
   & DEL [wksp]t$node.lis
   SHOW RESELECT [temp] 0
   &if &eq [temp] 0 &do
      &break
   &end
   &dec [i]
&end
```

For each level in the tree, from_nodes are listed to a file and arcs with corresponding to_nodes are selected; this becomes the next level in the tree until no more arcs are selected:



In the next section we work our way back down through the levels in the tree. For each arc in the current level, the stream order is calculated:

```
&type "Generating stream order..."
LINESYMBOL 4
&while &ne [i] 0 &do
   CLEARSELECT
   RES [cover] ARC [order_it] = [i]
   &openw [wksp]t$rec.lis
   LIST [cover] ARC $RECNO
   &closew
   &open [wksp]t$rec.lis error
   &while &do
      &read [temp] [break]
      &if &nm [temp] &do
         &cv [rec] [temp]
         CLEARSELECT
         RES [cover] ARC $RECNO IN
{[rec]}
         CALC %[cover] ARC [node] =
FNODE_
         &r st_calc
         &rv [temp]
         &if &eq [temp] .FALSE. &do
            &return
         &end
         CLEARSELECT
         RES [cover] ARC $RECNO IN
{[rec]}
         CALC %[cover] ARC %[order_it]
= %[order]
         ARCS [cover]
      &end
   &end
   &close
   & DEL [wksp]t$rec.lis
   &inc [i]
&end
```

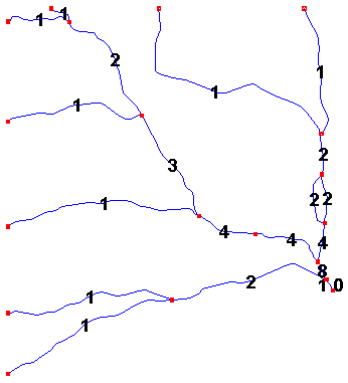
```
CLEARSELECT
&return

&label error
&type "I/O error"
&close &all
&closew &all
CLEARSELECT
&return
```

Routine "st_calc" performs the actual stream order calculation:

```
&routine st_calc

&include st_order.inc

&define o1 -11 &var
&define o2 -12 &var
&define delta -13 &var
&define sum -14 &var

&value [wksp] WKSP
CLEARSELECT
RES [cover] ARC TNODE_ = [node]
SHOW RESELECT [temp] 0
&if &eq [temp] 0 &do
   &sv [order] 1
   &return .TRUE.
&end
&if &eq [temp] 1 &do
   CALC %[cover] ARC [order] = %
[order_it]
   &return .TRUE.
&end
&openw [wksp]t$order.lis
LIST [cover] ARC [order_it]
&closew
&sv [first] .TRUE.
&sv [delta] .FALSE.
&open [wksp]t$order.lis error 2
&while &do
   &read [temp] [break] 2
   &if &nm [temp] &do
      &cv [o2] [temp]
      &if &eq [first] .TRUE. &do
         &value [o1] [o2]
         &value [sum] [o2]
         &sv [first] .FALSE.
         &continue
      &end
      &if &ne [o1] [o2] &do
         &cv [o1] [o1] max [o2]
         &sv [delta] .TRUE.
      &end
      &cv [sum] [sum] + [o2]
   &end
&end
&close 2
& DEL [wksp]t$order.lis
&if &eq [option] [opt1] &do
   &value [order] [sum]
&else
   &if &eq [delta] .TRUE. &do
      &value [order] [o1]
   &else
      &cv [order] [o1] + 1
   &end
&end
&return .TRUE.

&label error
&type "I/O error: st_calc"
&close &all
&closew &all
CLEARSELECT
&return .FALSE.
```

Arcs having to_nodes coresponding to current arc's fron_node are selected. If none are found, the order is set to 1, and if only one is selected, the order is preserved. Otherwise, the order is set according to the current option.

Use "compsml st_order n" to generate st_order.sml and st_calc.sml. If a steam has a split or braid, the routine won't blow up, but it will most likely lead to undesirable results:



PLP

Blank Variables

As of 3.5.2, and possibly earlier, local variables that are blank are filled with spaces after executing an ARC-level command:

```
&sv -1
&sv -2 #

&type "Before: x%-1x"
&type "Before: x%-2x"

IDENTITY LINECOV POLYCOV XX LINE

&type "After: x%-1x"
&type "After: x%-2x"
```

After executing IDENTITY, variable -2 will still contain "#" but -1 will contain 80 spaces. Similarly, local variables in a routine called by a module are filled with spaces:

```
TABLES tabtest.sml
```

tabtest.sml:

```
&type "x%-1x"
```

Variable -1 will contain 80 spaces.

If a variable hasn't been used and a blank value is needed, you can initialize the variable before you use it:

```
&sv -1
DEFINE [outfile]
[item1]
1
C
%-1
```

Otherwise, wherever possible, assign a non-blank values:

```
&if &eq "x%-5" "x" &do
   &sv -10 #
&else
   &sv -10 %-5
&end
IDENTITY [cover1] [cover2] [temp] POLY
%-10
INTERSECT [temp] [cover3] [outcover]
POLY %-10
```

PLP

CA/HI/NV/Guam ESRI Regional Users Conference

The Group's next annual conference will held at the Ilikai Hotel on Ala Moana Bvld in Honolulu, Hawaii on February 16-18 (Wed-Fri), 2000 with some pre-conference events planned as well. For more info see:

http://www.mp.usbr.gov/mp400/cahinv/cahinvpg.html

PLP

PLP OnLine

http://www.primenet.com/~piersen/PLP

User Name: plpv7n5
Password: z8f49rav