Subject: Re: about DXF format
Posted by Vince Hradil on Mon, 18 Jun 2007 14:55:57 GMT
View Forum Message <> Reply to Message

On Jun 18, 7:23 am, JMZawo...@gmail.com wrote:
> On Jun 16, 10:20 pm, airy.ji...@gmail.com wrote:
>
>> no more people would like to discuss this topic?what a pity!
>
> I have had nothing but trouble trying to export DXF files from AutoCAD
> and read them with IDLffDXF. Some (most) objects never appear and
> others are improperly positioned or rotated. My limited investigations
> led me to conclude that while DXF may be an open standard to exchange
> CAD models, it also allows for the inclusion of proprietary formatting
> and objects. True, IDL does not support all object types that may
> occur in DXF files, but this is not the primary problem. If you read
> the DXF file directly (it's ASCII) you'll note a lot of AutoCAD
> specific stuff in there that I gather tells AutoCAD more about how to
> position and orient objects in the model. It would be much more useful
> to me if IDL could read/write either IGES or STEP files as these are
> really designed to exchange model geometries. I currently export these
> types from AutoCAD and translate them to IDL-compatible DXF files
> using 3rd party software from TechnoSoft (AML).

In my experience, it IS possible to parse a dxf file. You just have
to read the docs that describe the format, then parse the file
correctly. The trick is that some entities contain other entities and
lines and they all have different local and global origins and scale
factors. Yeah, it complicated, but I've written a parser to parse a
few dxf files, and works (most of the time).

Here's my very crude code. Just try >plot_dxf, "file.dxf"

```
function resolve_inserts, innow, inserts, plines

  ;; plines
  nbdx = where( plines.blockname ne '', nbcnt )
  if nbcnt gt 0 then begin
     nbplines = plines[nbdx]
     bdx = where( nbplines.blockname eq innow.blockref, bcnt )
     for b=0l, bcnt-1 do begin
        pnow = plines[nbdx[bdx[b]]]
        *(pnow.vertices) = *(pnow.vertices) + rebin(innow.coord,
3,pnow.nvert,/sample)
        plines[nbdx[bdx[b]]] = pnow
     endfor
  endif
```

```
  ;; inserts
  nbdx = where( inserts.blockname ne '', nbcnt )
  if nbcnt gt 0 then begin
     nbinserts = inserts[nbdx]
     bdx = where( nbinserts.blockname eq innow.blockref, bcnt )
     for b=0l, bcnt-1 do begin
        ininside = inserts[nbdx[bdx[b]]]
        ininside.coord = ininside.coord + innow.coord
        if size(plines,/type) eq 8 then plines =
resolve_inserts(ininside,inserts, plines)
     endfor

     return, plines
  endif
end

function eval_spline, ncp, controlpts, nsegs

  tarray = findgen(nsegs)/(nsegs)
  np = (ncp-1)/3

  sval = fltarr(2,nsegs*np+1)
  for i=0l, np-1 do begin
     p0 = controlpts[*,3*i]
     p1 = controlpts[*,3*i+1]
     p2 = controlpts[*,3*i+2]
     p3 = controlpts[*,3*i+3]

     sval[*,nsegs*i] = p0
     for j=1l, nsegs-1 do begin
        t = tarray[j]
        vert = p0*(1-t)*(1-t)*(1-t) + p1*3.0*t*(1-t)*(1-t) +
p2*3.0*t*t*(1-t) + p3*t*t*t
        sval[*,nsegs*i+j] = vert
     endfor
  endfor
  sval[*,nsegs*np] = controlpts[*,ncp-1]

  return, sval
end

function decodetext, instring

  upos = strpos(instring,'\U+')
  if upos lt 0 then return, ''

  usplit = strsplit( strmid( instring,upos ), '\U+', /extract )
  outstring = bytarr(n_elements(usplit))
```

```
  reads, usplit, outstring, format='(Z)'

  return, string(outstring)
end

function read_dxf, fname

  nlines = file_lines(fname)
  print, 'NLINES: ', nlines

  openr, lun, fname, /get_lun
  line = ''
  adxf = strarr(nlines)
  for i=0l, nlines-1 do begin
     readf, lun, line
     adxf[i] = line
  endfor
  free_lun, lun

  return, adxf
end

function dxf_plines, fname, layer=layer, nsegs=nsegs

  if n_elements(layer) eq 0 then layer='ALL'
  if n_elements(nsegs) eq 0 then nsegs=4L

  adxf = read_dxf(fname)

  hdx = where( adxf eq 'HEADER')
  cdx = where( adxf eq 'CLASSES')
  tdx = where( adxf eq 'TABLES')
  bdx = where( adxf eq 'BLOCKS')
  edx = where( adxf eq 'ENTITIES')
  odx = where( adxf eq 'OBJECTS')
  eofdx = where( adxf eq 'EOF' )

  header = adxf[hdx:cdx-2]
  classes = adxf[cdx:tdx-2]
  tables = adxf[tdx:bdx-2]
  blocks = adxf[bdx:edx-2]
  entities = adxf[edx:odx-2]
  objects = adxf[odx:eofdx-2]

;   hfdx = where(header eq layer, hfcnt)
;   cfdx = where(classes eq layer, cfcnt)
;   tfdx = where(tables eq layer, tfcnt)
;   bfdx = where(blocks eq layer, bfcnt)
```

```
;   efdx = where(entities eq layer, efcnt)
;   ofdx = where(objects eq layer, ofcnt)

;   print, 'HEADER: ', hfcnt
;   print, 'CLASSES: ', cfcnt
;   print, 'TABLES: ', tfcnt
;   print, 'BLOCKS: ', bfcnt
;   print, 'ENTITIES: ', efcnt
;   print, 'OBJECTS: ', ofcnt

  ;; blocks
  iblocks = where( blocks eq 'BLOCK', nblocks )
  eblocks = where( blocks eq 'ENDBLK', neblocks )
  print, 'NBLOCKS: ', nblocks

  inserts = {blockname:'', blockref:'', coord:fltarr(3)}
  mtexts = {blockname:'', coord:fltarr(3), height:0.0, mstring:''}
  plines = {blockname:'', nvert:0L, vertices:ptr_new() }

  for i=0l, nblocks-1 do begin
     ibnow = iblocks[i]
     ebnow = eblocks[i]
     bnow = blocks[ibnow:ebnow]
     boffset = ( where( bnow eq 'AcDbBlockBegin' ) )[0]
     blockptr = boffset + 2l
     blockname = bnow[blockptr]
     blockptr = boffset + 16l
     if layer ne 'ALL' then begin
        bfdx = where( bnow eq layer, bfcnt )
     endif else begin
        bfcnt = 1l
     endelse
     if bfcnt gt 0 then begin
        print, 'BLOCK: ', blockname, i+1, format='(8A,12A,8I)'
        blen = ebnow-ibnow+1
        bent = bnow[blockptr]
        while blockptr lt blen and bent ne 'ENDBLK' do begin
           bent = bnow[blockptr]
           case bent of
              'INSERT': begin
                 boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbBlockReference' ) )[0]
                 blockptr = blockptr + boffset + 2l
                 blockref = bnow[blockptr]
                 blockptr = blockptr + 2l
                 xinsert = float(bnow[blockptr])
                 blockptr = blockptr + 2l
                 yinsert = float(bnow[blockptr])
```

```
            blockptr = blockptr + 2l
            zinsert = float(bnow[blockptr])
            print, ' INSERT: ', blockref, xinsert, yinsert,
zinsert
            inserts = [ inserts, {blockname:blockname,
blockref:blockref, coord:[xinsert,yinsert,zinsert]} ]
            blockptr = blockptr + 2l
          end
          'LINE': begin
            boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbLine' ) )[0]
            blockptr = blockptr + boffset + 2l
            startpt = fltarr(3)
            endpt = fltarr(3)
            for p=0l, 2 do begin
              startpt[p] = float(bnow[blockptr])
              blockptr = blockptr+2l
            endfor
            for p=0l, 2 do begin
              endpt[p] = float(bnow[blockptr])
              blockptr = blockptr+2l
            endfor
            print, ' LINE: ', strtrim(startpt) ;, strtrim(endpt)
            plines = [ plines, {blockname:blockname, nvert:2l,
vertices:ptr_new([[startpt],[endpt]])} ]
          end
          'SPLINE': begin
            boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbSpline' ) )[0]
            blockptr = blockptr + boffset + 2l
            splineflag = long(bnow[blockptr])

            blockptr = blockptr + 8l
            degree = long(bnow[blockptr])

            blockptr = blockptr + 2l
            nknots = long(bnow[blockptr])
            knots = fltarr(nknots)

            blockptr = blockptr + 2l
            ncontrolpts = long(bnow[blockptr])
;             controlpts = fltarr(3,ncontrolpts)
            controlpts = fltarr(2,ncontrolpts)

            blockptr = blockptr + 2l
            nfitpoints = long(bnow[blockptr])
            if nfitpoints gt 0 then fitpoints =
fltarr(3,nfitpoints)
```

```
            print, ' SPLINE: ', splineflag, degree, nknots,
ncontrolpts, nfitpoints

            blockptr = blockptr + 2l
            blockptr = blockptr +
( where( strtrim(bnow[blockptr:blen-1],2) eq '40' ) )[0] + 1
            for p=0l, nknots-1 do begin
               knots[p] = float(bnow[blockptr])
               blockptr = blockptr + 2l
            endfor
            for p=0l, ncontrolpts-1 do begin
               controlpts[0,p] = float(bnow[blockptr])
               blockptr = blockptr + 2l
               controlpts[1,p] = float(bnow[blockptr])
               blockptr = blockptr + 2l
;               controlpts[2,p] = float(bnow[blockptr])
               blockptr = blockptr + 2l
            endfor
;            sx = sort( controlpts[0,*] )
;            controlpts = controlpts[*,sx]
            for p=0l, nfitpoints-1 do begin
               fitpoints[0,p] = float(bnow[blockptr])
               blockptr = blockptr + 2l
               fitpoints[1,p] = float(bnow[blockptr])
               blockptr = blockptr + 2l
               fitpoints[2,p] = float(bnow[blockptr])
               blockptr = blockptr + 2l
            endfor
;            print, 'DONE SPLINE'
            sval = eval_spline(ncontrolpts,controlpts,nsegs)
            np = nsegs*(ncontrolpts-1)/3+1
            sval = transpose( [ [transpose([sval])],
[replicate(0.0,np)] ] )
            plines = [ plines,
{blockname:blockname,nvert:ncontrolpts*nsegs
+1L,vertices:ptr_new(sval)} ]
         end
         'HATCH': begin
            boffset = ( where( strtrim(bnow[blockptr:blen-1],2)
eq '2' ) )[0]
            blockptr = blockptr + boffset + 1L
            pattern = bnow[blockptr]

            blockptr = blockptr + 6l
            nloops = long(bnow[blockptr])
            print, ' HATCH: ', pattern, nloops, format="(8A,12A,
8I,$)"
```

```
            for p=0l, nloops-1 do begin
              blockptr = blockptr + 2l
              btype = long(bnow[blockptr])

              blockptr = blockptr + 2l
              nedge = long(bnow[blockptr])

              blockptr = blockptr + 2l
              edgetype = long(bnow[blockptr])
              case edgetype of
                1: begin ; line
                end
                2: begin ; circular arc
                end
                3: begin ; elliptical arc
                end
                4: begin ; spline
                  blockptr = blockptr + 2l
                  degree = long(bnow[blockptr])

                  blockptr = blockptr + 6l
                  nknots = long(bnow[blockptr])
                  knots = fltarr(nknots)

                  blockptr = blockptr + 2l
                  ncontrolpts = long(bnow[blockptr])
                  controlpts = fltarr(2,ncontrolpts)

                  blockptr = blockptr + 2l

                  print, ' SPLINE: ', degree, nknots,
ncontrolpts
                  for p=0l, nknots-1 do begin
                    knots[p] = float(bnow[blockptr])
                    blockptr = blockptr + 2l
                  endfor
                  for p=0l, ncontrolpts-1 do begin
                    controlpts[0,p] = float(bnow[blockptr])
                    blockptr = blockptr + 2l
                    controlpts[1,p] = float(bnow[blockptr])
                    blockptr = blockptr + 2l
                  endfor
;                  print, 'DONE HATCH SPLINE'
                  sval =
eval_spline(ncontrolpts,controlpts,nsegs)
                  np = nsegs*(ncontrolpts-1)/3+1
                  sval = transpose( [ [transpose([sval])],
[replicate(0.0,np)] ] )
```

```
                    plines = [ plines,
{blockname:blockname,nvert:ncontrolpts*nsegs
+1L,vertices:ptr_new(sval)} ]
                  end
                endcase
              endfor
              blockptr = blockptr + 12l
            end
            'MTEXT' : begin
              boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbMText' ) )[0]
              blockptr = blockptr + boffset + 2l
              position = fltarr(3)
              for p=0l, 2 do begin
                position[p] = float(bnow[blockptr])
                blockptr = blockptr+2l
              endfor
              height = float(bnow[blockptr])
              blockptr = blockptr+8l
              tstring = bnow[blockptr]
              mstring = decodetext(tstring)
              print, ' MTEXT: ', mstring
              mtexts = [ mtexts, {blockname:blockname,
coord:position, height:height, mstring:mstring} ]
              blockptr = blockptr + 14l
            end
            'POLYLINE' : begin
              seqend = ( where( bnow[blockptr:blen-1] eq
'SEQEND' ) )[0]
              vdx = where( bnow[blockptr:blockptr+seqend] eq
'AcDb2dVertex', vcnt )
              if vcnt gt 0 then begin
                vertices = fltarr(3,vcnt)
                for v=0l, vcnt-1 do begin
                  boffset = ( where( bnow[blockptr:blen-1] eq
'AcDb2dVertex' ) )[0]
                  blockptr = blockptr + boffset + 2l
                  for p=0l, 2 do begin
                    vertices[p,v] = float(bnow[blockptr])
                    blockptr = blockptr+2l
                  endfor
                endfor
                print, ' PLINE: ', strtrim(vertices[*,0])
                plines = [ plines, {blockname:blockname,
nvert:vcnt, vertices:ptr_new(vertices)} ]
              endif
              blockptr = blockptr + 10l
            end
```

```
            'ENDBLK': break
            else: begin       ; unknown block
              print, 'Unknown Block: ', bent
              blockptr = blockptr + 1l
            end
          endcase
        endwhile
      endif else begin
;       print, 'BLOCK: ', blockname, i+1, format="(8A,12A,8I,$)"
;       print, ' (NOT FIGURE)'
      endelse
    endfor

    bnow = entities
    blen = odx-2-edx

    blockptr = 2l
    if layer ne 'ALL' then begin
      bfdx = where( bnow eq layer, bfcnt )
    endif else begin
      bfcnt = 1l
    endelse
    if bfcnt gt 0 then begin
      bent = bnow[blockptr]
      repeat begin
        bent = bnow[blockptr]
        case bent of
          'INSERT': begin
            boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbEntity' ) )[0]
            blockptr = blockptr + boffset + 2l
            if layer eq 'ALL' or layer eq bnow[blockptr] then begin
              boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbBlockReference' ) )[0]
              blockptr = blockptr + boffset + 2l
              blockref = bnow[blockptr]
              blockptr = blockptr + 2l
              xinsert = float(bnow[blockptr])
              blockptr = blockptr + 2l
              yinsert = float(bnow[blockptr])
              blockptr = blockptr + 2l
              zinsert = float(bnow[blockptr])
              print, ' INSERT: ', blockref, xinsert, yinsert,
zinsert
              inserts = [ inserts, {blockname:'',
blockref:blockref, coord:[xinsert,yinsert,zinsert]} ]
              blockptr = blockptr + 2l
            endif else begin
```

```
            blockptr = blockptr + ( where( bnow[blockptr:blen-1]
eq '  0' ) )[0] + 1
            endelse
          end
          'LINE': begin
            boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbEntity' ) )[0]
            blockptr = blockptr + boffset + 2l
            if layer eq 'ALL' or layer eq bnow[blockptr] then begin
              boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbLine' ) )[0]
              blockptr = blockptr + boffset + 2l
              startpt = fltarr(3)
              endpt = fltarr(3)
              for p=0l, 2 do begin
                startpt[p] = float(bnow[blockptr])
                blockptr = blockptr+2l
              endfor
              for p=0l, 2 do begin
                endpt[p] = float(bnow[blockptr])
                blockptr = blockptr+2l
              endfor
              print, ' LINE: ', strtrim(startpt) ;, strtrim(endpt)
              plines = [ plines, {blockname:'', nvert:2l,
vertices:ptr_new([[startpt],[endpt]])} ]
            endif else begin
              blockptr = blockptr + ( where( bnow[blockptr:blen-1]
eq '  0' ) )[0] + 1
            endelse
          end
          'SPLINE': begin
            boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbEntity' ) )[0]
            blockptr = blockptr + boffset + 2l
            if layer eq 'ALL' or layer eq bnow[blockptr] then begin
              boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbSpline' ) )[0]
              blockptr = blockptr + boffset + 2l
              splineflag = long(bnow[blockptr])

              blockptr = blockptr + 8l
              degree = long(bnow[blockptr])

              blockptr = blockptr + 2l
              nknots = long(bnow[blockptr])
              knots = fltarr(nknots)

              blockptr = blockptr + 2l
```

```
                ncontrolpts = long(bnow[blockptr])
;                controlpts = fltarr(3,ncontrolpts)
                controlpts = fltarr(2,ncontrolpts)

                blockptr = blockptr + 2l
                nfitpoints = long(bnow[blockptr])
                if nfitpoints gt 0 then fitpoints =
fltarr(3,nfitpoints)

                print, ' SPLINE: ', splineflag, degree, nknots,
ncontrolpts, nfitpoints

                blockptr = blockptr + 2l
                blockptr = blockptr +
( where( strtrim(bnow[blockptr:blen-1],2) eq '40' ) )[0] + 1
                for p=0l, nknots-1 do begin
                   knots[p] = float(bnow[blockptr])
                   blockptr = blockptr + 2l
                endfor
                for p=0l, ncontrolpts-1 do begin
                   controlpts[0,p] = float(bnow[blockptr])
                   blockptr = blockptr + 2l
                   controlpts[1,p] = float(bnow[blockptr])
                   blockptr = blockptr + 2l
;                   controlpts[2,p] = float(bnow[blockptr])
                   blockptr = blockptr + 2l
                endfor
;                sx = sort( controlpts[0,*] )
;                controlpts = controlpts[*,sx]
                for p=0l, nfitpoints-1 do begin
                   fitpoints[0,p] = float(bnow[blockptr])
                   blockptr = blockptr + 2l
                   fitpoints[1,p] = float(bnow[blockptr])
                   blockptr = blockptr + 2l
                   fitpoints[2,p] = float(bnow[blockptr])
                   blockptr = blockptr + 2l
                endfor
;                print, 'DONE SPLINE'
                sval = eval_spline(ncontrolpts,controlpts,nsegs)
                np = nsegs*(ncontrolpts-1)/3+1
                sval = transpose( [ [transpose([sval])],
[replicate(0.0,np)] ] )
                plines = [ plines,
 {blockname:'',nvert:ncontrolpts*nsegs+1L,vertices:ptr_new(sv al)} ]
             endif else begin
                blockptr = blockptr + ( where( bnow[blockptr:blen-1]
eq '  0' ) )[0] + 1
             endelse
```

```
        end
        'HATCH': begin
          boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbEntity' ) )[0]
          blockptr = blockptr + boffset + 2l
          if layer eq 'ALL' or layer eq bnow[blockptr] then begin
            boffset = ( where( strtrim(bnow[blockptr:blen-1],2)
eq '2' ) )[0]
            blockptr = blockptr + boffset + 1L
            pattern = bnow[blockptr]

            blockptr = blockptr + 6l
            nloops = long(bnow[blockptr])
            print, ' HATCH: ', pattern, nloops, format="(8A,12A,
8I,$)"
            for p=0l, nloops-1 do begin
              blockptr = blockptr + 2l
              btype = long(bnow[blockptr])

              blockptr = blockptr + 2l
              nedge = long(bnow[blockptr])

              blockptr = blockptr + 2l
              edgetype = long(bnow[blockptr])
              case edgetype of
                1: begin ; line
                end
                2: begin ; circular arc
                end
                3: begin ; elliptical arc
                end
                4: begin ; spline
                  blockptr = blockptr + 2l
                  degree = long(bnow[blockptr])

                  blockptr = blockptr + 6l
                  nknots = long(bnow[blockptr])
                  knots = fltarr(nknots)

                  blockptr = blockptr + 2l
                  ncontrolpts = long(bnow[blockptr])
                  controlpts = fltarr(2,ncontrolpts)

                  blockptr = blockptr + 2l

                  print, ' SPLINE: ', degree, nknots,
ncontrolpts
                  for p=0l, nknots-1 do begin
```

```
                    knots[p] = float(bnow[blockptr])
                    blockptr = blockptr + 2l
                  endfor
                  for p=0l, ncontrolpts-1 do begin
                    controlpts[0,p] = float(bnow[blockptr])
                    blockptr = blockptr + 2l
                    controlpts[1,p] = float(bnow[blockptr])
                    blockptr = blockptr + 2l
                  endfor
;                  print, 'DONE HATCH SPLINE'
                  sval =
eval_spline(ncontrolpts,controlpts,nsegs)
                  np = nsegs*(ncontrolpts-1)/3+1
                  sval = transpose( [ [transpose([sval])],
[replicate(0.0,np)] ] )
                  plines = [ plines,
 {blockname:'',nvert:ncontrolpts*nsegs+1L,vertices:ptr_new(sv al)} ]
                end
              endcase
            endfor
            blockptr = blockptr + 12l
          endif else begin
            blockptr = blockptr + ( where( bnow[blockptr:blen-1]
eq '  0' ) )[0] + 1
          endelse
        end
        'MTEXT' : begin
          boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbEntity' ) )[0]
          blockptr = blockptr + boffset + 2l
          if layer eq 'ALL' or layer eq bnow[blockptr] then begin
            boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbMText' ) )[0]
            blockptr = blockptr + boffset + 2l
            position = fltarr(3)
            for p=0l, 2 do begin
              position[p] = float(bnow[blockptr])
              blockptr = blockptr+2l
            endfor
            height = float(bnow[blockptr])
            blockptr = blockptr+8l
            tstring = bnow[blockptr]
            mstring = decodetext(tstring)
            print, ' MTEXT: ', mstring
            mtexts = [ mtexts, {blockname:'', coord:position,
height:height, mstring:mstring} ]
            blockptr = blockptr + 14l
          endif else begin
```

```
            blockptr = blockptr + ( where( bnow[blockptr:blen-1]
eq '  0' ) )[0] + 1
          endelse
        end
        'POLYLINE' : begin
          boffset = ( where( bnow[blockptr:blen-1] eq
'AcDbEntity' ) )[0]
          blockptr = blockptr + boffset + 2l
          if layer eq 'ALL' or layer eq bnow[blockptr] then begin
            seqend = ( where( bnow[blockptr:blen-1] eq
'SEQEND' ) )[0]
            vdx = where( bnow[blockptr:blockptr+seqend] eq
'AcDb2dVertex', vcnt )
            if vcnt gt 0 then begin
              vertices = fltarr(3,vcnt)
              for v=0l, vcnt-1 do begin
                boffset = ( where( bnow[blockptr:blen-1] eq
'AcDb2dVertex' ) )[0]
                blockptr = blockptr + boffset + 2l
                for p=0l, 2 do begin
                  vertices[p,v] = float(bnow[blockptr])
                  blockptr = blockptr+2l
                endfor
              endfor
              print, ' PLINE: ', strtrim(vertices[*,0])
              plines = [ plines, {blockname:'', nvert:vcnt,
vertices:ptr_new(vertices)} ]
            endif
            blockptr = blockptr + 10l
          endif else begin
            blockptr = blockptr + ( where( bnow[blockptr:blen-1]
eq '  0' ) )[0] + 1
          endelse
        end
        'ENDSEC': break
        else: begin          ; unknown block
          print, 'Unknown Block: ', bent
          blockptr = blockptr + ( where( bnow[blockptr:blen-1] eq
'  0' ) )[0] + 1
        end
      endcase
    endrep until blockptr ge blen or bent eq 'ENDSEC'
  endif else begin
;       print, 'BLOCK: ', blockname, i+1, format="(8A,12A,8I,$)"
;       print, ' (NOT FIGURE)'
  endelse

  ninserts = ( size(inserts,/dimensions) )[0]
```

```
  if ninserts gt 1 then begin
    ninserts = ninserts - 1
    inserts = inserts[1:ninserts]
  endif else begin
    inserts = (-1)
  endelse

  nmtexts = ( size(mtexts,/dimensions) )[0]
  if nmtexts gt 1 then begin
    nmtexts = nmtexts - 1
    mtexts = mtexts[1:nmtexts]
  endif else begin
    mtexts = (-1)
  endelse

  nplines = ( size(plines,/dimensions) )[0]
  ptr_free, (plines[0]).vertices
  if nplines gt 1 then begin
    nplines = nplines - 1
    plines = plines[1:nplines]
  endif else begin
    plines = (-1)
  endelse

  ;; resolve inserts
  if size(inserts,/type) eq 8 then begin
    idx = where( inserts.blockname eq '', cnt )
    for i=0L, cnt-1 do begin
;       print, i+1, ' of ', cnt
;      ans = ''
;      read, ans, prompt='Enter something:'
       innow = inserts[idx[i]]
       if size(plines,/type) eq 8 then plines =
resolve_inserts(innow,inserts, plines)
    endfor
  endif

  return, plines
end

pro plot_dxf, fname, nsegs=nsegs, layer=layer

  colors
  bignum = 9999999.9
  minx = bignum
  maxx = -bignum
  miny = bignum
  maxy = -bignum
```

```
  if n_elements(nsegs) eq 0 then nsegs=10L

  plines = dxf_plines(fname, layer=layer, nsegs=nsegs)
  if size(plines,/type) ne 8 then return
  nplines = n_elements(plines)
  for i=0L, nplines-1 do begin
     minx = min( (*(plines[i].vertices))[0,*] ) < minx
     miny = min( (*(plines[i].vertices))[1,*] ) < miny
     maxx = max( (*(plines[i].vertices))[0,*] ) > maxx
     maxy = max( (*(plines[i].vertices))[1,*] ) > maxy
  endfor

  mmx = [minx,maxx]
  mmy = [miny,maxy]

  print, mmx
  print, mmy

  mmx = [0,300]
  mmy = [0,300]

  plot, [0,0], [1,1], /nodata, /noerase, xstyle=5, ystyle=5, /
isotropic, xrange=mmx, yrange=mmy

  for j=0l, nplines-1 do plots, *(plines[j].vertices)

  return
end
```