Appendix B

POL: A Binary File Format for Polygonal Models

InnovMetric Software has devised POL, a binary file format supported by IMCompress that allows the easy description of polygonal models. A key feature of the POL format is that it supports the description of multi-contour, simple planar polygons. This appendix gives a complete description of the POL format.

B.1 Fundamentals

Multi-contour polygons

POL supports the description of multi-contour polygons. A multi-contour polygon is defined by one external contour and a series of internal contours that trim the area enclosed by the external contour.

If no internal contours are defined, the area enclosed by the external contour is a direct facet of the polygonal model. An example of a single contour polygon is shown in Fig. B.1. When internal contours are defined, they either remove or add an area to the facet represented by the polygon. Internal contours that are not enclosed in any other internal contours are considered holes. In this way, they remove the area they enclose from the polygonal model. Moreover, internal contours can be defined inside a hole in order to create an “island”. These island contours cancel the hole and add the area they enclose to the polygonal model. Thereafter, holes can be defined inside islands and so on. Up to 128 levels of contour enclosing can be specified in the POL format. Fig. B.2 shows an example of a multi-contour polygon with two holes and one island.

Simple polygon constraint

Only simple polygons should be described in the POL format since IMCompress does not support complex polygons. A polygon is simple if its edges intersect at vertices only, if there are no duplicate vertices, and if exactly two edges meet at any vertex.

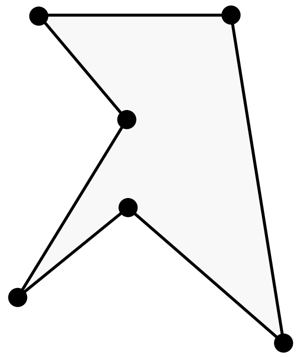

Fig. B.1 A single contour polygon

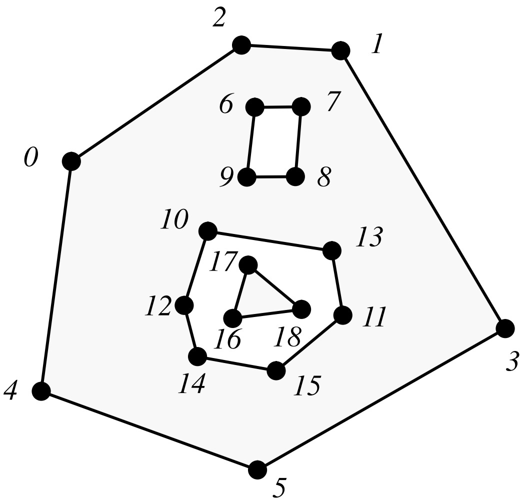

Fig. B.2 A multi-contour polygon

Planar polygon constraint

Polygons described in the POL format should be planar polygons. A planar polygon is a polygon for which all vertices lie on a common plane. When a polygon is planar, it is possible to fit a plane to its vertices and use this fitted plane to triangulate the interior of the polygon. IMCompress does not guarantee the triangulation results if non-planar polygons are defined.

Counterclockwise order constraint for the external contour

In order to determine the orientation of the polygons, the external contour of every multi-contour polygon description must be specified in counterclockwise order according to the right-hand rule.

B.2 Specification of the POL format

POL format files are identified by the ".pol" extension. A POL file may comprise up to four blocks of information. The first block is a header that contains useful information about the file structure. Following the header, a second block of information contains the set of 3-D geometric vertices of the polygonal model. A third block then contains the description of the model polygons. The first three blocks of information must be present in a POL file. The fourth block of a POL file contains color or texture information and is optional. The rest of this section explains the structure of each block of information in a POL file.

B.2.1 Block 1: Header

The header part of a POL file is 512 bytes long and contains useful information for reading the rest of the file. The following structure can be used to read the header of a POL file (C language syntax):

```
struct header
   {
   char  format_version[64];
   char  user_comments[128];
   char  dummy1[4];
   long  nv;
   long  vblock_length;
   long  np;
   long  pblock_length;
   long  color_texture_flag;
   long  ctblock_length;
   long  dummy2[73];
   };
```

`format_version:`	A 64-byte field reserved for an identifier string which gives the name of the format and the version number. IMCompress 1.0 currently writes “POL Format v1.0” in this field.
`user_comments:`	A 128-byte field available for the user. Users can write any information there about the model.
`dummy1:`	A 4-byte field that may be used for special characters.
`nv:`	Number of vertices in the file.
`vblock_length:`	Length in bytes of the block containing the vertices.
`np:`	Number of polygons described in the file.
`pblock_length:`	Length in bytes of the block containing the polygons.
`color_texture_flag:`	A flag indicating color or texture information. All color or texture information is given in the optional fourth block following the block of polygons. If 0, there is no color or texture information; If 1, RGB colors are defined for each vertex of the polygonal model; If 2, RGBA colors are defined for each vertex of the polygonal model; If 3, two texture coordinates are defined for each polygon vertex.
`ctblock_length:`	Length of the color or texture block in bytes.
`dummy2:`	Space reserved for future use.

B.2.2 Block 2: List of vertices

This block contains `vblock_length` bytes of information. It contains a list of `nv` vertices. Each vertex is specified by 3 floating point numbers representing its *x, y, z* coordinates. Reading the list of vertices can be done by reading a block of `nv` vertex structures (C language syntax):

```
struct vertex
      {
      float  x;
      float  y;
      float  z;
      };
```

B.2.3 Block 3: List of polygons

This block contains `pblock_length` bytes of information. It can be read as a block of `pblock_length/4` integer numbers of type `long`. This section of a POL file consists of a list of `np` polygons. Each polygon vertex is represented by an integer number which indicates a vertex position in the list of vertices. Therefore, a polygon vertex can take values from 0 to `nv`-1.

The first parameter in the description of a multi-contour polygon is the number of contours:

```
long  nc;  /* number of contours */
```

The number of contours must be larger than or equal to 1. After this comes a list of `nc` contour descriptions. The first contour must be the external contour. The order in which the internal contours are specified is not important. A contour description is made of two parts. The first part is an integer number representing the number of contour vertices:

```
long  ncv;  /* number of contour vertices */
```

The second part of a contour description consists of a list of `ncv` contour vertices. The vertices of the external contour must be given in counterclockwise order. As for the internal contours, they can be either expressed in clockwise or counterclockwise order. A vertex is an integer number:

```
long v;  /* a given vertex */
```

Example of a multi-contour polygon

Let us consider the multi-contour polygon of Fig. B-2. Assuming that the indices next to the polygon vertices correspond to positions in the list of vertices, this polygon would be described in a POL file as follows:

```
4    /* Number of polygon contours */
6    /* Number of vertices in external contour */
4    /* External contour vertices in counterclockwise order */
5
3
1
2
0
3    /* Number of vertices in internal contour 1 */
16   /* List of contour vertices */
17
18
4    /* Number of vertices in internal contour 2 */
6    /* List of contour vertices */
7
8
9
6    /* Number of vertices in internal contour 3 */
10   /* List of contour vertices */
12
14
15
11
13
```

The data type long is used for all integer numbers in a POL file.

B.2.4 Block 4: Optional color or texture information

This optional block contains `ctblock_length` bytes of information. The content of the block depends on the value of `color_texture_flag`. If this flag is set to *0*, the block is empty.

- If `color_texture_flag` is set to *1*, the block contains three bytes of color information for each geometric vertex described in block 2. The three bytes represent the RGB components of the vertex color. RBG components contained in the block are interlaced as follows:

  RGBRGBRGBRGB...

- If `color_texture_flag` is set to *2*, the block contains four bytes of color information for each geometric vertex described in block 2. The four bytes represent the RGBA components of the vertex color. RBGA components contained in the block are interlaced as follows:

  RGBARGBARGBARGBA...

- If `color_texture_flag` is set to *3*, the block contains 2-D texture coordinates for each polygon vertex specified in block 3. The block then contains `ctblock_length/4` floating point numbers. For each polygon vertex, two floating point numbers give *x* and *y* texture coordinates. Texture coordinates must be normalized between *0* and *1* for both axes. The texture image is specified as an external file in the POL format. Use **`imformat`** to get the list of image formats supported by IMCompress and IMTexture.