MNI .obj file format

Robert D. Vincent

April 4, 2005

Contents

1 Introduction

The MNI .obj file format was designed as a simple storage format for representations of geometric objects. Overall, an MNI .obj file may have either an ASCII and a binary representation. In ASCII files, space and newline characters serve as separators for data fields, but have no other significance to the format.

All of the original design and implementation of this format was the work of David MacDonald. This document borrows extensively from David's existing comments and documentation.

2 Basics

A .obj file consists of a simple list of object records which may be in one of seven classes. Each object record is introduced by a single ASCII alphabetic character that determines the class of the object record. For binary records, the character is in lower case. For ASCII records, the character is in upper case. The size of the record itself is variable, but is typically determined by certain key fields that define the number of vertices or points in the object.

2.1 Basic types

File data consists of five fundamental types: *char* (alphabetic object class specifiers), *float*, *integer*, *string*, or *boolean*.

Because the ASCII library is implemented using standard "C" functions such as `fscanf` and `fprintf`, the textual representations of numeric types are quite standard. *Integer* values consist of an optional sign ('+' or '-') followed by one or more decimal digits. *Float* values will consist of an optional sign followed by an integer part consisting of one or more decimal digits, followed by an optional decimal point, followed by an optional fractional part consisting of one or more decimal digits, and finally an optional exponent specifier of the form 'e' or 'E' followed by an optional sign and an integer exponent. Either the integer part or the fractional part must have a non-zero length.

In the case of binary files, there is no clear specification of the format or "endianness" of the data. It can be assumed that *integer* values will be 32-bit, 2's complement integers, and that *float* numbers will be 32-bit IEEE floating point values.

In ASCII files, a *string* is represented as a sequence of zero or more characters surrounded by either single quotes, double quotes, or back quotes.

In binary files, a *string* is represented as a 32-bit integer length followed by the specified number of 8-bit bytes.

In ASCII files, *boolean* values are represented by the ASCII character 'f' or 'F' for FALSE, and 't' or 'T' for TRUE.

In binary files, *boolean* values are simply integer values. Any non-zero value is interpreted as representing logical TRUE, zero is interpreted as FALSE.

2.2 Compound types

The MNI .obj format defines four compound types: *point*, *colour*, *vector*, and *surfprop*, a surface property record.

A *point* is defined by three floating point numbers corresponding to the point's location along the X, Y, and Z axis respectively. The range of these values is arbitrary.

The second compound type defined by this format is the *colour* object. A colour is defined by four floating-point numbers in the interval 0..1, representing red, blue, green, and alpha values. The alpha value is typically one.

A *vector* is similar to a point object in structure, consisting of three values for the X, Y, and Z components of the vector. Vectors are assumed to be normalized unit vectors, therefore all values should be in the range -1.0...+1.0.

A *surfprop* structure is composed of five floating point parameters which define the appearance of a surface material. The definition of these values was based largely on the SGI IRIS Graphics Library and OpenGL specifications. The five members of a surface property structure are defined as follows:

- Ambient colour (A)

  This value in the range 0...1 defines the intensity of the surface's ambient colour, as a proportion of the surface's specified colour.

- Diffuse reflectivity (D)

  This value in the range 0...1 sets the diffuse reflectivity of the object's surface, as a proportion of the surface's specified colour.

- Specular reflectance (S)

  This value in the range 0...1 specifies the specular reflectance of the object surface. Specular reflectance is assumed to be uniform over the colour spectrum.

- Specular scattering exponent (SE)

  This value in the range 0...128 specifies the specular scattering exponent of the material. The higher the value, the smoother the surface appearance and the more focused the specular highlight.

- Transparency (T)

  This value in the range 0...1 specifies the transparency of the surface. This value may be ignored in many implementations, depending on the graphics library in use and the capabilities of the graphics controller.

2.3 Object classes

The following eight object classes are defined by the MNI .obj file format.

- 'L' (Lines) - A group of line segments.
- 'M' (Marker) - A tag or marker point.

- 'F' (Model) - A reference to an external .obj file.
- 'X' (Pixels) - An 2-dimensional bitmap.
- 'P' (Polygons) - A set of closed polygons.
- 'Q' (Quadmesh) - A quadrilateral mesh.
- 'T' (Text) - A text label.

The 'V' (Volume) character is reserved but not utilized.

3 Object record details

3.1 Lines

A lines object is used to represent a series of line segments.

- thickness (*float*)

  This value specifies the overall thickness of the line segments. It must be within the range 0.001...20.0.

- npoints (*integer*)

  This specifies the total number of distinct vertices in the lines object

- point_array (array of *point*)

  A list of the coordinates for each distinct vertex in the lines object. Note that vertices may be reused if the end_indices and indices fields are set appropriately.

- nitems (*integer*)

  The total number of line segments

- colour_flag (*integer*)

  A flag indicating the number of colours allocated in this object. A value of zero specifies that single colour applies to all line segments. A value of one specifies that colours are specified on a per-line basis. A value of two specifies that colours are specified on a per-vertex basis.

- colour_table (array of *colour*)

  1The RGB colour values to be associated with the line segments. The length of this section may be either 1 (if **colour_flag** is 0), **nitems** (if **colour_flag** is 1) or **npoints** (if **colour_flag** is 2).

- end_indices (array of *integer*)

  This is a list of length **nitems** that specifies the index of the last element in the **indices** array for each group of line segments in the lines object.

- indices (array of *integer*)

  The integer index into the point_array that specifies how each of the vertices is assigned to each line segment. The length of this array must be equal to the greatest value in the **end_indices** array plus one.

3.2 Marker

A marker object is used to assign a special meaning to a position in the object. The marker can be associated with a specific patient, anatomical structure, and text comment.

- type (*integer*) A value of zero indicates this marker should be displayed as a box, while a value of one indicates a sphere.
- size (*float*) The desired size of the marker. The units and interpretation are not clearly specified.
- colour (*colour*) The desired colour for the marker.
- position (*point*) The location of the maker.
- structure_id (*integer*) An integer value associating this marker with a specific anatomical structure.
- patient_id (*integer*) An integer value associating this marker with a particular patient.
- label (*string*) A string to be displayed along with the marker.

3.3 Model

A model record is used to incorporate the contents of another .obj file by reference. Processing of model records is *not* automatically handled by the library, it is up to the calling program to read the external file.

- filename (*string*)

3.4 Pixels

A pixels record stores a two-dimensional bitmap object.

- pixel_type (*integer*)

  This field specifies the type of the pixel_array field. The pixel type can take a value of either 0, 1 or 2. A value of zero indicates that the pixel array values are eight bit color indices of type *integer*. A value of one indicates that pixel values are sixteen bit color indices of type *integer*. A value of two indicates that pixel values are of RGB values of type *colour*.

- x_size (*integer*)

  The extent of the bitmap in the horizontal direction.

- y_size (*integer*)

  The extent of the bitmap in the vertical direction.

- pixel_array (see pixel_type field)

  The array of pixel values, with a total of x_size * y_size elements. Pixels are stored such that the upper left corner is the first pixel in the array, and the lower right corner is the last pixel in the array.

3.5 Polygons

Polygon records may be in either compressed or uncompressed format. Compressed format is reserved for sets of polygons with tetrahedral topology.

The uncompressed format is as follows:

- surfprop (*surfprop*)

  Surface properties for the polygons.

- npoints (*integer*)

  Number of distinct vertices in the aggregate polygon object.

- point_array (array of *point*)

  List of distinct vertices that define this group of polygons. Note that vertices may be reused if the end_indices and indices fields are set appropriately.

- normals (array of *vector*)

  List of point normals for each point.

- nitems (*integer*)

  Number of polygons defined.

- colour_flag (*integer*)

  A flag indicating the number of colours allocated in this object. A value of zero specifies that single colour applies to all line segments. A value of one specifies that colours are specified on a per-item basis. A value of two specifies that colours are specified on a per-vertex basis.

- colour_table (array of *colour*)

  The RGB colour values to be associated with the polygons. The length of this section may be either 1 (if **colour_flag** is 0), **nitems** (if **colour_flag** is 1) or **npoints** (if **colour_flag** is 2).

- end_indices (array of *integer*)

  This is a list of length **nitems** that specifies the index of the element in the indices list associated with each successive polygon.

- indices (array of *integer*)

  A list of integer indices into the **point_array** that specifies how each of the vertices is assigned to each polygon. The length of this array must be equal to the greatest value in the **end_indices** array plus one.

The compressed format is distinguised by the sign of the field after the surface properties, which should always be negative. Use of the compressed format is reserved for polygons with tetrahedral topology.

- surfprop (*surfprop*)

  Surface properties for the polygons.

- -nitems (*integer*)

  Number of polygons, multiplied by -1.

- point_array (array of *point*)

  List of vertices that define this group of polygons.

- colour_flag (*integer*)

  A flag indicating the number of colours allocated in this object. A value of zero specifies that single colour applies to all line segments. A value of one specifies that colours are specified on a per-item basis. A value of two specifies that colours are specified on a per-vertex basis.

- colour_table (array of *colour*)

  The RGB colour values to be associated with the polygons. The length of this section may be either 1 (if **colour_flag** is 0), nitems (if **colour_flag** is 1) or npoints (if **colour_flag** is 2).

3.6 Quadmesh

Quadrilateral mesh.

- surfprop (*surfprop*)

  Surface properties for the mesh.

- m (*integer*)

  The number of rows in the mesh.

- n (*integer*)

  The number of columns in the mesh.

- m_closed (*boolean*)

  A boolean specifying whether the rows have their extreme points joined by an edge.

- n_closed (*boolean*)

  A boolean specifying whether the columns have their extreme points joined by an edge.

- colour_flag (*integer*)

  A flag indicating the number of colours allocated in this object. A value of zero specifies that single colour applies to all line segments. A value of one specifies that colours are specified on a per-item basis. A value of two specifies that colours are specified on a per-vertex basis.

- colour_table (array of *colour*)

  The RGB colour values to be associated with the mesh. The length of this section may be either 1 (if **colour_flag** is 0), $(m-1)*(n-1)$ (if **colour_flag** is 1) or $m*n$ (if **colour_flag** is 2).

- point_array (array of *point*)

  A list of points, of total length $n*m$.

- normals (array of *vector*)

  A list of point normals, of total length $n*m$.

3.7 Text

A text object specifies a text label.

- font_type (*integer*)

  This integer flag is set to zero to specify a fixed-width font, or to one for proportional spaced font.

- text_size (*float*)

  The desired font size.

- colour (*colour*)

  The desired text colour.

- point (*point*)

  The 3D position at which to display the text.

- text (*string*)

  The text to display.

4 Implementation

The object file format is implemented by the "BIC programmer's library" (BICPL). The low-level I/O functions for basic types are defined in the "volume_io" library.

The following BICPL functions are used to implement high-level object file I/O:

```
Status input_graphics_file(char *filename,
                           File_formats *format,
                           int *n_objects,
                           object_struct ***object_list);

Status output_graphics_file(char *filename,
                            File_formats format,
                            int n_objects,
                            object_struct *object_list[]);
```