OFF - A 3D Object File Format

Randi J. Rost
6-November-1986
Updated 12-October-1989

Digital Equipment Corporation
Workstation Systems Engineering
100 Hamilton Ave.
Palo Alto, Ca. 94301

This document describes the data format developed by WSE for the interchange and archiving of three-dimensional objects. This format, called OFF (for Object File Format), is general, flexible, and extensible. It supports ASCII text versions of objects for the purpose of interchange, and binary versions for efficiency of reading and writing. It is assumed that applications will develop their own, more efficient format for internal storage and operation on three-dimensional objects.

1. Introduction

One of the most time-consuming tasks in computer animation projects is designing the 3D models that will be used. Many computer animation houses have found that owning a large number of databases makes it easier for them to take on new projects at a lower cost (time and $$$). The cost of initially creating an object can be amortized over the number of times it can be re-used. It is our intention to promote the use of OFF files within (and perhaps even outside of) Digital in an effort to build up our collection of useful 3D models.

The file format itself is not limiting: OFF files can be used for a wide variety of object types. None of the "policy decisions" are hard-wired in the design of the file format, or in the support library routines that allow reading and writing of OFF files. Rather, the policy decisions have been left up to the designers since the format supports "generic" object definitions. We have developed specific conventions for objects that are defined by polygons for use within WSE, and we'd encourage others to adopt these conventions as well in order to promote the interchange of useful object data bases. The *dxmodel* application (also developed by WSE) is an example of an application that permits reading and writing of OFF files.

This paper describes the Object File Format itself, the conventions we've adopted at WSE, and the library of support routines that can be used to read and write OFF files.

2. Design Goals and Non-Goals

Design goals for the Object File Format include:

1) **Simple.** Simple cases should be simple. It should be possible to type in all the data required for a simple object (such as a cube) by hand.

2) **Powerful.** The Object File Format should be capable of accomodating complicated objects (many vertices, polygons, and a wide range of attributes).

3) **Portable.** ASCII text file representation required for portability across operating systems and hardware. This also allows operations on OFF files by familiar system utilities (text editors and the like).

4) **Efficient.** Binary text file representation required to allow efficient reading and writing of OFF files. It is assumed that reading/writing an object is a costly operation, but reading and writing ASCII data is just *too* slow.

5) **General.** The format should address the general case of the three-dimensional object, not a single particular case.

6) **Extensibile.** Make sure the format can be easily extended to eventually support other primitives such as bezier patches.

7) **No Favors.** Avoid hard-wiring policy decisions. Rather, provide generic building blocks capable of supporting several styles and document a set of strongly encouraged conventions that we have adopted.

There are also things that were specifically non-goals in the design of the Object File Format.

1) **Internal Format.** The Object File Format is not intended to be forced upon applications as an internal format that must be used. Rather, it should be considered a means for inputting and outputting object descriptions in a device-, language-, and operating system-independent format. Applications should feel free to develop and maintain the most useful/efficient data format they can, and only convert to/from OFF when input or output of a standardized object is desired.

2) **Object Conventions.** OFF conventions are documented only for objects in polygonal form at this point. It is anticipated that the Object File Format can be easily extended to handle bezier surface patches and other primitives in the future.

3. Objects

For the purposes of the Object File Format, we'll adopt a very general definition of an *object*. An *object* is simply a list of properties (name, description, author, copyright information, geometry data, colors, etc.)

The most important information about the object can be found in the *header file* for the object. The header file is always an ASCII text file that, by convention, is named *name*.aoff where *name* is the object name. See Appendix B for an example of an OFF object header file.

A few of these properties (name, description, author, copyright, type) are common to every type of 3D object and are considered standard properties. The standard properties are built into the routines that manipulate 3D objects. The rest of the properties may vary with the type of object, and so are defined by convention only.

The *name* of an object is used to concisely describe the object itself. For example, we have objects named "x29", "banana" and "vw". By convention, this name also becomes the prefix for OFF data filenames when an object is read or written, so it is best to keep it fairly short.

The *description* is used to more fully describe the object itself. It may contain the time and date of creation or more prose describing the object.

The *author* should be the name of the person (or company, or utility) that created the object. We should always try to give credit where credit is due. This field tells you who to thank for spiffy objects or whose cage to rattle when a problem with an OFF file is discovered.

The *copyright* field contains information dealing with the distribution of the object data. Some object databases will be regarded by a company as proprietary. These objects should not be copied or distributed without consent. Other objects (vw, x29) were developed by companies or individuals and can be copied or used as long as the copyright notice appears and proper credit is given. Still other objects (cube, sphere, etc.) have been placed in the public domain. We have tried to be as careful as possible in preserving copyright and author information for the objects we have collected, but sometimes the information was lost or unavailable. Be sure and honor copyright notices. If you don't, you (or your company) could end up in big trouble.

The *type* field contains the type of the object. For now, only one type of object is supported: polygon objects. It is anticipated that polyline and surface patch-type objects will be supported in the future as well.

Also contained in the object header file are lines that describe the various properties of the object. Each line in the object header file that describes an attribute of the object other than a standard attribute must contain the property name, the property type, the data format and either a file name or a string containing default data, depending on the property type. Each of these four items is an ASCII string, separated by white space.

The *property name* uniquely describes the property. Property names for which we have defined conventions (see Appendix A) include geometry, polygon_colors, vertex_colors, back_faces, vertex_order, polygon_normals, vertex_normals, diffuse_coef, specular_coef, and specular_power.

OFF currently supports four *property types*: *default*, *generic*, *indexed*, and *indexed_poly*. If a property is indicated to be of type *default*, the part of the line after the data format is assumed to contain some default data that will be applied to the entire object. For instance, it may make sense to give the entire object a default color and default diffuse and specular coefficients.

If the property type is either generic, indexed, or indexed_poly (described more fully below), the remainder of the line is taken to be a file name that can be opened and read to obtain the information for the property.

The data format indicates what type of data will be found on the remainder of the line if the property type is default, otherwise what kind of data will be found in the specified file. The data format is a string of characters (no spaces) that indicate the order and type of the data. Supported primitive data types are:

f - A number stored internally as a 32-bit floating point number

d - A number stored internally as a 64-bit double-precision floating point number

i - A number stored internally as a 32-bit integer value

h - A number stored internally as a 16-bit integer value

b - A number stored internally as an 8-bit integer value

s - A 32-bit pointer to a null-terminated string of characters

If, for instance, you were interested in using 32-bit floating values for r, g, and b default color values, you might have a line in the object header file that reads polygon_colors default fff 1.0 0.8 0.0

It is important to understand that in all cases, the "string" (s) data primitive will indicate a pointer to a string that is stored internally in the data block for an object and not the string itself.

4. ASCII Property Files

OFF supports ASCII text files as a way of providing for language-, hardware-, and operating system-independent object data files. The three types of data files currently supported by OFF are generic, indexed, and indexed_poly.

4.1. Generic Files

Generic files contain only a *count* value followed by *count* data items of the type specified by the data format. Each data item can be comprised of some combination of the primitive data types described in Section 3. Generic data files are useful for storing attributes which are unique at every vertex or polygon (such as color or normals).

String data items in ASCII generic files may not contain spaces or other white space. 8-bit integers must be listed in the range 0-255.

See Appendix D for an example of an ASCII generic data file.

4.2. Indexed Files

Indexed files make use of a list of indices in order to reduce the amount of data required to store a property, and to provide a useful level of indirection. For instance, indexed files are commonly used to maintain per-polygon color information. If an object has just five colors, the indexed data file would contain the list of the five colors followed by an index from one to five for each of the polygons in the object. If an application maintains the indirection, it is possible for the user to easily select five different colors to be used on the model.

An indexed file begins with two integers separated by white space: the number of data items and the number of indices that will be provided. Following these two values is the list of data items that is to be used. Each data item in this list can be some combination of the primitive types described in Section 3. Following the data items is a list of indices each of which is a pointer to one of the items in the list of data items. The list of data items is assumed to begin starting at one, not zero.

4.3. Indexed_Poly Files

Indexed_poly files take advantage of a connectivity list to reduce the amount of information needed to store a list of polylines, polygons, or normals. The unique geometry items (e.g., vertices) are listed in the first part of the file. Following this list is a connectivity list. Each line in the connectivity list contains a *count* value followed by *count* indices (pointers) to information in the geometry list. (Items in the geometry list are indexed starting from one, not zero.)

The first line of an indexed_poly data file contains three integers, separated by white space. The first number on this line indicates the number of data items (vertices/normals) that follow, the second number indicates the number of polylines/polygons that follow the data list, and the third indicates the total number of edges that are contained in the polyline/polygon connectivity list.

String data items in ASCII indexed_poly files may not contain spaces or other white space. 8-bit integers must be listed in the range 0-255.

See Appendix C for an example of an ASCII indexed_poly file.

5. Binary OFF Files

The same three types of data files described above are also supported in binary format. There are a few minor differences.

5.1. Generic Files

Binary generic files begin with the first 32-bit word equal to OFF_GENERIC_MAGIC as defined in the include file *off.h*. The second word in the file is the *count* (number of data items in the file). Following the *count* is the data itself.

The data format in the header file describes the primitives that make up each data item in the list. Within each data item, floats, doubles, 32-bit integers, and string pointers will all begin on a word boundary. 16-bit integers will all begin on a half-word boundary. Thus, if your data format for the data items in a generic data file is "bbb" (three byte values), each data item will be stored as three bytes followed by a null byte so that each data item will begin on a word boundary. Strings begin with a 32-bit *count* followed by *count* characters followed by a null character. The string is null-padded so it will end on a word boundary.

(It is assumed that for strings, the length will be read and then the necessary memory will be allocated and the string read in. This eliminates a problem with having variable-length data in the data files. Anyway, strings in files are really only there for symmetry with default values, where strings are really useful. The performance implications for files containing strings will probably be enough to prevent people from using them.)

5.2. Indexed Files

Binary indexed files begin with the first 32-bit word equal to OFF_INDEXED_MAGIC as defined in the include file *off.h*. The second word in the file is the *count* (number of data items in the file). The third word in the file is *num_indices* (number of indices in the index list).

Following these two integers is the data for the data item list. Each item in the data item list will begin on a word boundary. The data format in the header file describes the primitives that make up each data item in the list. Within each data item, floats, doubles, 32-bit integers, and string pointers will all begin on a word boundary. 16-bit integers will all begin on a half-word boundary. Thus, if your data format for the data items in an indexed data file is "bbb" (three byte values), each data item will be stored as three bytes followed by a null byte so that each data item will begin on a word boundary. Strings begin with a 32-bit *count* followed by *count* characters followed by a null character. The string is null-padded so it will end on a word boundary.

Following the data item list is a list of index values. This list will also begin on a word boundary, however, the index values within this list are short integers and will be packed two to each 32-bit word.

5.3. Indexed_Poly Files

Binary indexed_poly files begin with the first 32-bit word equal to OFF_INDEXED_POLY_MAGIC as defined in the include file off.h. The second word is the number of data items in the vertex list (*npts*), the third word is the number of polylines/polygons in the list (*npolys*), and the fourth word is the number of edges contained in the connectivity list (*nconnects*).

Starting at the fifth word in the file is a list of *npts* data items, followed by *npolys* short integers containing polyline/polygon vertex counts, followed by *nconnects* short integers which are indices into the array of data items. (This arrangement is slightly different than that used for indexed_poly files in ASCII format for efficiency reasons.)

The same restrictions that apply to the data types for generic binary files apply to indexed_poly binary files as well. In addition, the vertex count array which follows the geometry data in an indexed_poly file will always begin on a word boundary. The connectivity array that follows the vertex count array will not necessarily start on a word boundary, but will always begin *npolys * sizeof(short)* bytes after the start of the vertex count array.

6. Off.a and Objects.h

An include file and a library of routines has been provided for UNIX/C programmers to more easily manipulate OFF files. The basic concepts of "reading" and "writing" OFF files are supported in this library of routines. The library is a software layer on top of the operating system file I/O interface, with special knowledge of OFF files. This subroutine library provides a mechanism for accessing the syntactical elements of an object file, but makes no attempt to understand the semantics. Higher level interfaces can be layered on top.

The subroutine library refers to an object as a pointer to an *OFFObjDesc*. This structure contains a pointer to the first property in the property list. It is defined as follows:

```
typedef struct
   {
   OFFProperty     *FirstProp;  /* Pointer to first property in list */
   } OFFObjDesc;
```

The information that describes the object is contained in a linked list of property structures. The first such structure in the list is pointed at by an *OFFObjDesc* structure. The property structures have the form:

```
typedef struct _OFFProp
   {
   char        PropName[40];
   int         PropType;
   char        PropFileName[256];
   int         PropCount;
   char        DataFormat[40];
   char        *PropData;
   struct _OFFProp *NextProp;
   } OFFProperty;
```

PropName contains a string defining one of the property types for which a convention has been defined. This includes the property names "name", "author", "description", "copyright", "comment", "geometry", "polygon_colors", "polygon_normal", etc. For a complete list of property names, see Appendix A. (The special attribute type "comment" is supported so that blank lines and comment lines can be preserved if an object file is read and then written.)

The *PropType* field contains a value equal to *OFF_DEFAULT_DATA*, *OFF_GENERIC_DATA*, *OFF_INDEXED_DATA*, or *OFF_INDEXED_POLY_DATA* which defines the basic type for the property.

The *PropFileName* is required if *PropType* is something other than *OFF_DEFAULT_DATA*. It contains a string representing the name of the file to be read/written for this attribute. This file name should *not* contain a path leading up to the file itself, only the actual file name. The object search path mechanism (see Section 7) should be used instead.

The *PropCount* indicates the actual number of data items associated with this particular attribute. After reading in an object, properties of type *OFF_DEFAULT_DATA* will have a *PropCount* of one, properties of type *OFF_GENERIC_DATA* will have a *PropCount* equal to the number of generic data items in the list, properties of type *OFF_INDEXED_DATA* will have a *PropCount* equal to the number of items in the data item list, and properties of type *OFF_INDEXED_POLY_DATA* will have a *PropCount* equal to the number of data items in the geometry list.

The *DataFormat* field contains a string of characters corresponding to primitive data items. The composite type of the data for this property can then be deduced by looking at this field and applying the rules for padding to word and half-word boundaries.

The *PropData* field contains a pointer to a block of memory containing the actual data for this property. This data will have the same data alignment restrictions as a binary file has, with the exception of strings. As strings are read in, memory is malloc'ed to hold them and a pointer to the string is stored in the appropriate field in the data list. This means that all primitive data types will have a fixed size and lengths and alignments can be computed more easily.

The *NextProp* field contains a pointer to the next property structure in the property list.

The routines contained in the subroutine library are defined below.

#include "off.h"

int **OFFReadObj**(Obj, FileName)
    OFFObjDesc *Obj;
    char *FileName;

int **OFFWriteObj**(Obj, FileName, Directory, FileType);
    OFFObjDesc *Obj;
    char *FileName;
    char *Directory;
    int FileType;

void **OFFPackObj**(Obj)
    OFFObjDesc *Obj;

void **OFFPackProperty**(Property)
    OFFProperty *Property;

int **OFFReadGeneric**(Property, FileName)
    OFFProperty *Property;
    char *FileName;

int **OFFWriteGeneric**(Property, FileName, FileType)
    OFFProperty *Property;
    char *FileName;
    int FileType;

int **OFFReadIndexed**(Property, FileName)
    OFFProperty *Property;
    char *FileName;

int **OFFWriteIndexed**(Property, FileName, FileType)
    OFFProperty *Property;
    char *FileName;
    int FileType;

int **OFFReadIndexedPoly**(Property, FileName)
    OFFProperty *Property;
    char *FileName;

int **OFFWriteIndexedPoly**(Property, FileName, FileType)
    OFFProperty *Property;
    char *FileName;
    int FileType;

OFFObjDesc ***OFFCreateObj**()

int **OFFDestroyObj**(Obj)
    OFFObjDesc *Obj;

OFFProperty ***OFFAddProperty**(Obj)
    OFFObjDesc *Obj;

int **OFFRemoveProperty**(Obj, PropertyName)
    OFFObjDesc *Obj;
    char *PropertyName;

int **OFFFreeProperty**(Property)
    OFFProperty *Property;

OFFReadObj will attempt to open the object header file named *FileName* and read the object data it contains. A pointer to the constructed object structure will be returned in *Obj* when the object has been read. An attempt will be made to open the specified file first as given, then concatenated in turn with each of the directories specified by the environment search path variable *OBJ_PATH*. The property list for the object is built as the file is read. Upon return, the client need only traverse the property list and select the data it needs. This routine calls *OFFReadGeneric* and *OFFReadIndexedPoly* in order to read associated data files. *OFFReadObj* will return 0 if the read operation was successful, -1 otherwise.

OFFWriteObj will attempt to write the object pointed at by *Obj* using the filename specified by *FileName*. The file will be written in the directory indicated by *Directory*. If *FileType* is *OFF_ASCII*, the file will be written as an ASCII text OFF file. If *FileType* is *OFF_BINARY*, the file will be written as a binary OFF file. The property list for the object is traversed and each property of the object is written out in turn. This routine calls *OFFWriteGeneric* and *OFFWriteIndexedPoly* in order to write associated data files. *OFFWriteObj* will return 0 if the write operation was successful, -1 otherwise.

OFFPackObj attempts to *pack* the object pointed at by *Obj*. Packing only applies to geometry and normals stored in indexed_polygon format. See *OFFPackProperty* below.

OFFPackProperty packs the property pointed at by *Property*. Packing can only be applied to indexed_polygon properties with format ‘‘fff’’. (This is normally the case for geometry and normals properties.) Packing works by sharing common data (vertex or normal) values. Since each vertex and vertex normal of an object is usually shared by three or more polygons, this can save a great deal of space and rendering time. If *Property* could not be packed, it is not modified; otherwise the property data is changed in-place.

OFFReadGeneric will read the generic data file named *FileName* (here *FileName* contains the full path name) into the property structure pointed at by *Property*. This routine will allocate the space it needs in order to read in the data. A pointer to this allocated data space will be stored in the *PropData* field of the specified *property* as described earlier. The entire object, including all allocated memory resources can later be deallocated by calling *OFFDestroyObj*. This routine will not typically be called directly by applications. *OFFReadGeneric* will return 0 if the read operation was successful, -1 otherwise.

OFFWriteGeneric will write the generic data associated with *Property* into the file *FileName* (here *FileName* contains the full path name of the file to be written). If *FileType* is *OFF_ASCII*, the file will be written as an ASCII text generic data file. If *FileType* is *OFF_BINARY*, the file will be written as a binary generic data file. This routine will not typically be called directly by applications. *OFFWriteGeneric* will return 0 if the write operation was successful, -1 otherwise.

OFFReadIndexed will read the indexed data file named *FileName* (here *FileName* contains the full path name) into the property structure pointed at by *Property*. This routine will allocate the space it needs in order to read in the data. A pointer to this allocated data space will be stored in the *PropData* field of the specified *property* as described earlier. The entire object, including all allocated memory resources can later be deallocated by calling *OFFDestroyObj*. This routine will not typically be called directly by applications. *OFFReadIndexed* will return 0 if the read operation was successful, -1 otherwise.

OFFWriteIndexed will write the indexed data associated with *Property* into the file *FileName* (here *FileName* contains the full path name of the file to be written). If *FileType* is *OFF_ASCII*, the file will be written as an ASCII text indexed data file. If *FileType* is *OFF_BINARY*, the file will be written as a binary indexed data file. This routine will not typically be called directly by applications. *OFFWriteIndexed* will return 0 if the write operation was successful, -1 otherwise.

OFFReadIndexedPoly will read the indexed_poly data file named *FileName* (here *FileName* contains the full path name) into the property structure pointed at by *Property*. This routine will allocate the space it needs in order to read in the data. A pointer to this allocated data space will be stored in the *PropData* field of the specified *property* as described earlier. The entire object, including all allocated memory resources can later be deallocated by calling *OFFDestroyObj*. This routine will not typically be called directly by applications. *OFFReadIndexedPoly* will return 0 if the read operation was successful, -1 otherwise.

OFFWriteIndexedPoly will write the indexed_poly data associated with *Property* into the file *FileName* (here *FileName* contains the full path name of the file to be written). If *FileType* is *OFF_ASCII*, the file will be written as an ASCII text indexed_poly data file. If *FileType* is *OFF_BINARY*, the file will be written as a binary indexed_poly data file. This routine will not typically be called directly by applications. *OFFWriteIndexedPoly* will return 0 if the write operation was successful, -1 otherwise.

OFFCreateObj allocates and initializes an *OFFObjDesc* structure A pointer to the newly-created structure is returned. The null pointer is returned if the operation was unsuccessful.

OFFDestroyObj deallocates all memory resources associated with the object pointed at by *Obj*. It works by calling *OFFFreeProperty* for each property in the property list for the specified object.

OFFAddProperty adds a property structure to the property list associated with the object pointed at by *Obj*, initializes it, and returns a pointer to it. The null pointer is returned if the operation was unsuccessful.

OFFRemoveProperty deletes the named property from the object pointed at by *Obj*. This routine returns -1 if the named property is not found in the property list for the specified object.

OFFFreeProperty frees all the memory resources allocated to the property structure specified by *Property* as well as the property structure itself. This routine will not typically be called directly by applications.

7. Object Search Path

It is important to avoid embedding path names in object files. When an object is transported to another system, chances are slim that the same directory structure will exist. The *OFFReadObj* routine in libobj.a knows about an environment variable named *OBJ_PATH* that is used to overcome this problem.

When an object is read, an attempt is first made to open it in the current working directory. If that attempt fails, the directories specified in the *OBJ_PATH* environment variable are tried in turn until the file is successfully opened or the directory list is exhausted. *OBJ_PATH* contains a list of directories, separated by spaces, that are to be searched for the named objects.

The name of the directory where a successful open operation occurred is used for opening associated data files as well. This means that all of the data files for a particular object must reside in the same directory.

It is hoped that in this way, users will be able to draw on one or more collections of "standard" objects in addition to their own private collections of objects.

8. Appendix A: Conventions for Polygonal Objects

This list contains the conventions we have adopted for describing 3D polygonal objects which are defined in some three-dimensional model coordinate system. Items in regular type are string literal, printed as they would appear in an OFF file, and item in italics indicate data values that will vary from object to object. By convention, the header for an ASCII OFF file is suffixed with ".aoff" and the header for a binary OFF file is suffixed with ".off". There are two choices for how colors may be stored. If they are stored as generic data, the suffixes used are ".{b}pcol" for polygon colors and ".{b}vcol" for vertex colors. If they are stored as indexed data, the suffixes used are ".{b}ipcol" and ".{b}ivcol".

Property	Type	Format	Defaults	ASCII filename	Binary Filename
name	*****	*****	*objname*	*****	*****
author	*****	*****	*author*	*****	*****
description	*****	*****	*description*	*****	*****
copyright	*****	*****	*copyright*	*****	*****
type	*****	*****	polyline	*****	*****
	*****	*****	polygon	*****	*****
geometry	indexed_poly	fff	*****	*name*.geom	*name*.bgeom
polygon_colors	generic	fff	*****	*name*.pcol	*name*.bpcol
vertex_colors	generic	fff	*****	*name*.vcol	*name*.bvcol
polygon_colors	indexed	fff	*****	*name*.ipcol	*name*.bipcol
vertex_colors	indexed	fff	*****	*name*.ivcol	*name*.bivcol
back_faces	default	s	cull	*****	*****
			display	*****	*****
			reverse	*****	*****
vertex_order	default	s	clockwise	*****	*****
			counter-clockwise	*****	*****
			counterclockwise	*****	*****
polygon_normals	generic	fff	*****	*name*.pnorm	*name*.bpnorm
vertex_normals	generic	fff	*****	*name*.vnorm	*name*.bvnorm
diffuse_coef	default	f	*value*	*****	*****
specular_coef	default	f	*value*	*****	*****
specular_power	default	f	*value*	*****	*****
bounding_box	default	ffffff	*value*	*****	*****

9. Appendix B: OFF Header File For a Cube (cube.aoff)

```
name          cube
author        Randi J. Rost
description   cube with sides of red, green, blue, cyan, yellow, magenta
copyright     public domain
type          polygon

# Prop.            data type      format    filename or default data
#________          __________     ______    _________________________

geometry           indexed_poly   fff       cube.geom
vertex_order       default        s         clockwise
polygon_colors     generic        fff       cube.pcol
back_faces         default        s         cull
```

10. Appendix C: Listing of cube.geom

```
 8      6      24
-1.0   -1.0    1.0
-1.0    1.0    1.0
 1.0    1.0    1.0
 1.0   -1.0    1.0
-1.0   -1.0   -1.0
-1.0    1.0   -1.0
 1.0    1.0   -1.0
 1.0   -1.0   -1.0
 4      1      2      3      4
 4      5      6      2      1
 4      3      2      6      7
 4      3      7      8      4
 4      1      4      8      5
 4      8      7      6      5
```

11. Appendix D: Listing of cube.pcol

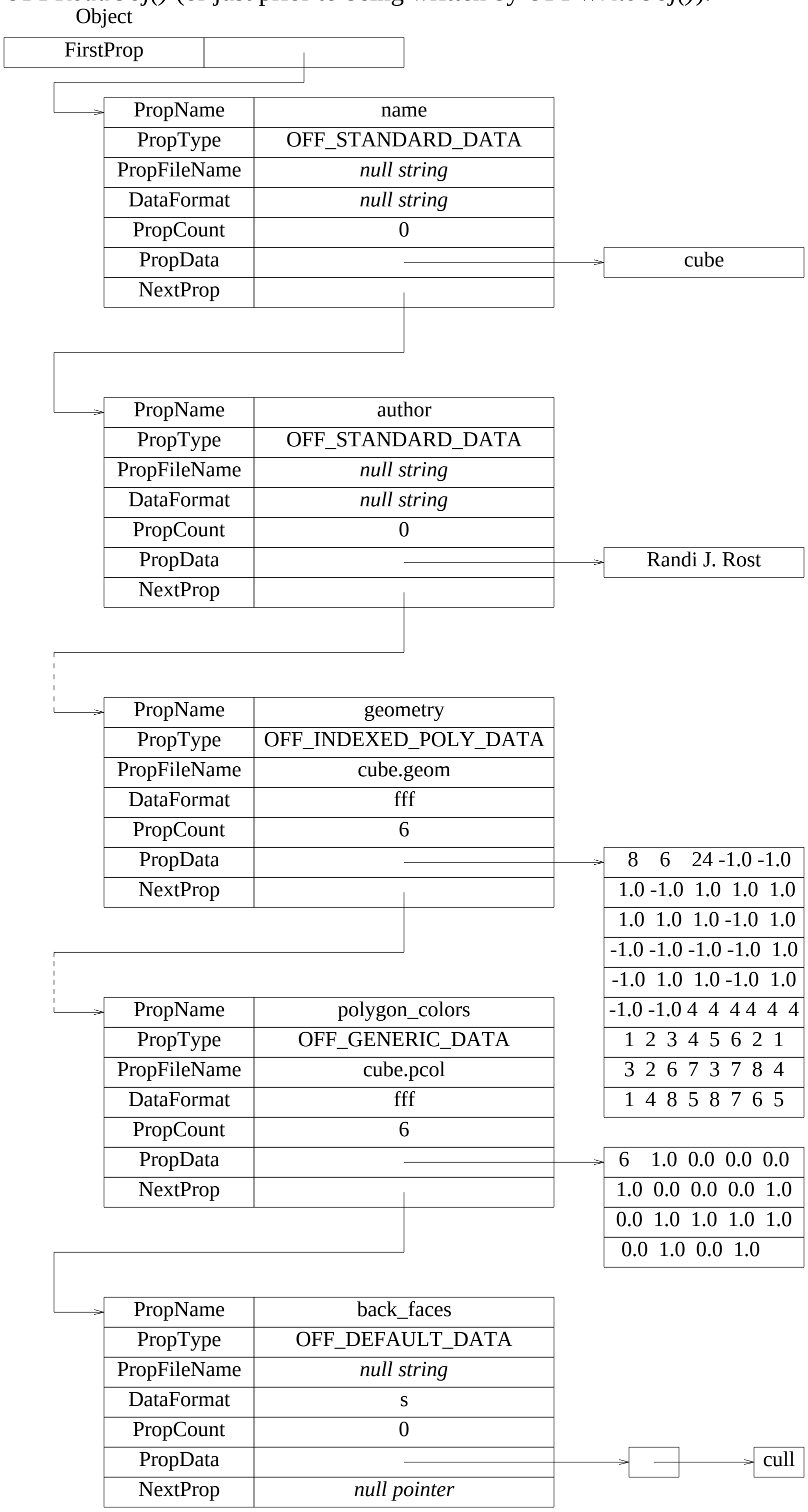
```
6
1.0     0.0     0.0
0.0     1.0     0.0
0.0     0.0     1.0
0.0     1.0     1.0
1.0     1.0     0.0
1.0     0.0     1.0
```

12. Appendix E: Listing of off.h

```
#define OFF_INDEXED_POLY_MAGIC  0xFEEDFEEDL
#define OFF_GENERIC_MAGIC       0xBEEFBEEFL
#define OFF_INDEXED_MAGIC       0xBADBADBAL

#define OFF_BIGSTR              256
#define OFF_SMSTR               40

#define  OFF_ASCII               0
#define  OFF_BINARY              1


/* Types of data for object properties  */

#define OFF_UNKNOWN_TYPE_DATA   0
#define OFF_STANDARD_DATA       1
#define OFF_COMMENT_DATA        2
#define OFF_DEFAULT_DATA        3
#define OFF_GENERIC_DATA        4
#define OFF_INDEXED_POLY_DATA   5
#define OFF_INDEXED_DATA        6


typedef struct _OFFProp
    {
    char        PropName[OFF_SMSTR];     /* Name of property (or attribute)  */
    int         PropType;                /* Type of data for property        */
    char        PropFileName[OFF_BIGSTR];/* Name of file that has prop data  */
    char        DataFormat[OFF_SMSTR];   /* Pointer to property data format  */
    int         PropCount;               /* Number of data items for property*/
    char        *PropData;               /* Pointer to property data         */
    struct _OFFProp *NextProp;           /* Pointer to next property in list */
    } OFFProperty;

typedef struct
    {
    OFFProperty *FirstProp;              /* Pointer to first property in list*/
    } OFFObjDesc;
```

13. Appendix F: Data Structure Format

The following diagram depicts some of the data structures for the object *cube.aoff* after being read by *OFFReadObj()* (or just prior to being written by *OFFWriteObj()*).



14. Acknowledgements

OFF is a derivative of an object file format used at Ohio State University. Special thanks to Allen Akin of WSE for helpful ideas and suggestions. Thanks also to Jeff Friedberg of Digital's High-Performance Workstation (HPWS) group and Shaun Ho of WSE who also contributed to the design. Danny Shapiro of WSE provided suggestions for additional enhancements and conventions.