PLY - Polygon File Format

Also known as the Stanford Triangle Format

Source code examples: ply.h, plytest.c, plyfile.c, plydocs.txt

Example of an ascii ply file

Introduction

This document presents the PLY polygon file format, a format for storing graphical objects that are described as a collection of polygons. Our goal is to provide a format that is simple and easy to implement but that is general enough to be useful for a wide range of models. The file format has two sub-formats: an ASCII representation for easily getting started, and a binary version for compact storage and for rapid saving and loading. We hope that this format will promote the exchange of graphical object between programs and also between groups of people.

Overview

Anyone who has worked in the field of computer graphics for even a short time knows about the bewildering array of storage formats for graphical objects. It seems as though every programmer creates a new file format for nearly every new programming project. The way out of this morass of formats is to create a single file format that is both flexible enough to anticipate future needs and that is simple enough so as not to drive away potential users. Once such a format is defined, a suite of utilities (both procedures and entire programs) can be written that are centered around this format. Each new utility that is added to the suite can leverage off the power of the others.

The PLY format describes an object as a collection of vertices, faces and other elements, along with properties such as color and normal direction that can be attached to these elements. A PLY file contains the description of exactly one object. Sources of such objects include: hand-digitized objects, polygon objects from modeling programs, range data, triangles from marching cubes (isosurfaces from volume data), terrain data, radiosity models. Properties that might be stored with the object include: color, surface normals, texture coordinates, transparency, range data confidence, and different properties for the front and back of a polygon.

The PLY format is NOT intended to be a general scene description language, a shading language or a catch-all modeling format. This means that it includes no transformation matrices, object instantiation, modeling hierarchies, or object sub-parts. It does not include parametric patches, quadric surfaces, constructive solid geometry operations, triangle strips, polygons with holes, or texture descriptions (not to be confused with texture coordinates, which it does support!).

A typical PLY object definition is simply a list of (x,y,z) triples for vertices and a list of faces that are described by indices into the list of vertices. Most PLY files include this core information. Vertices and faces are two examples of "elements", and the bulk of a PLY file is its list of elements. Each element in a given file has a fixed number of "properties" that are specified for each element. The typical information in a PLY file contains just two elements, the (x,y,z) triples for vertices and the vertex indices for each face. Applications can create new properties that are attached to elements of an object. For example, the properties red, green and blue are commonly associated with vertex elements. New properties are added in such a way that old programs do not break when these new properties are encountered. Properties that are not understood by a program can either be carried along uninterpreted or can be discarded. In addition, one can create a new element type and define the properties associated with this element. Examples of new elements are edges, cells (lists of pointers to faces) and materials (ambient, diffuse and specular colors and coefficients). New elements can also be carried along or discarded by programs that do not understand them.

File Structure

This is the structure of a typical PLY file:

```
Header
Vertex List
Face List
(lists of other elements)
```

The header is a series of carriage-return terminated lines of text that describe the remainder of the file. The header includes a description of each element type, including the element's name (e.g. "edge"), how many such elements are in the object, and a list of

the various properties associated with the element. The header also tells whether the file is binary or ASCII. Following the header is one list of elements for each element type, presented in the order described in the header.

Below is the complete ASCII description for a cube. The header of a binary version of the same object would differ only in substituting the word "binary_little_endian" or "binary_big_endian" for the word "ascii". The comments in brackets are NOT part of the file, they are annotations to this example. Comments in files are ordinary keyword-identified lines that begin with the word "comment".

```
ply
format ascii 1.0           { ascii/binary, format version number }
comment made by Greg Turk  { comments keyword specified, like all lines }
comment this file is a cube
element vertex 8           { define "vertex" element, 8 of them in file }
property float x           { vertex contains float "x" coordinate }
property float y           { y coordinate is also a vertex property }
property float z           { z coordinate, too }
element face 6             { there are 6 "face" elements in the file }
property list uchar int vertex_index { "vertex_indices" is a list of ints }
end_header                 { delimits the end of the header }
0 0 0                    { start of vertex list }
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3                { start of face list }
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

This example demonstrates the basic components of the header. Each part of the header is a carriage-return terminated ASCII string that begins with a keyword. Even the start and end of the header ("ply" and "end_header") are in this form. The characters "ply" must be the first four characters of the file, since they serve as the file's magic number.

Following the start of the header is the keyword "format" and a specification of either ASCII or binary format, followed by a version number. Next is the description of each of the elements in the polygon file, and within each element description is the specification of the properties. Then generic element description has this form:

```
element
property
property
property
...
```

The properties listed after an "element" line define both the data type of the property and also the order in which the property appears for each element. There are two kinds of data types a property may have: scalar and list. Here is a list of the scalar data types a property may have:

```
name       type        number of bytes
---------------------------------------
char       character                 1
uchar      unsigned character        1
short      short integer             2
ushort     unsigned short integer    2
int        integer                   4
uint       unsigned integer          4
float      single-precision float    4
double     double-precision float    8
```

These byte counts are important and must not vary across implementations in order for these files to be portable. There is a special form of property definitions that uses the list data type:

```
property list
```

An example of this is from the cube file above:

```
property list uchar int vertex_index
```

This means that the property "vertex_index" contains first an unsigned char telling how many indices the property contains, followed by a list containing that many integers. Each integer in this variable-length list is an index to a vertex.

Another Example

Here is another cube definition:

```
ply
format ascii 1.0
comment author: Greg Turk
comment object: another cube
element vertex 8
property float x
property float y
property float z
property uchar red                { start of vertex color }
property uchar green
property uchar blue
element face 7
property list uchar int vertex_index { number of vertices for each face }
element edge 5                    { five edges in object }
property int vertex1              { index to first vertex of edge }
property int vertex2              { index to second vertex }
property uchar red                { start of edge color }
property uchar green
property uchar blue
end_header
0 0 0 255 0 0                     { start of vertex list }
0 0 1 255 0 0
0 1 1 255 0 0
0 1 0 255 0 0
1 0 0 0 0 255
1 0 1 0 0 255
1 1 1 0 0 255
1 1 0 0 0 255
3 0 1 2                   { start of face list, begin with a triangle }
3 0 2 3                   { another triangle }
4 7 6 5 4                 { now some quadrilaterals }
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
0 1 255 255 255               { start of edge list, begin with white edge }
1 2 255 255 255
2 3 255 255 255
3 0 255 255 255
2 0 0 0 0                 { end with a single black line }
```

This file specifies a red, green and blue value for each vertex. To illustrate the variable-length nature of vertex_index, the first two faces of the object are triangles instead of a single square. This means that the number of faces in the object is 7. This object also contains a list of edges. Each edge contains two pointers to the vertices that delinate the edge. Each edge also has a color. The five edges defined above were specified so as to highlight the two triangles in the file. The first four edges are white, and they surround the two triangles. The final edge is black, and it is the edge that separates the triangles.

User-Defined Elements

The examples above showed the use of three elements: vertices, faces and edges. The PLY format allows users to define their own elements as well. The format for defining a new element is exactly the same as for vertices, faces and edges. Here is the section of a header that defines a material property:

```
element material 6
property ambient_red uchar          { ambient color }
property ambient_green uchar
property ambient_blue uchar
property ambient_coeff float
property diffuse_red uchar          { diffuse color }
property diffuse_green uchar
property diffuse_blue uchar
property diffuse_coeff float
property specular_red uchar         { specular color }
property specular_green uchar
property specular_blue uchar
property specular_coeff float
property specular_power float       { Phong power }
```

These lines would appear in the header directly after the specification of vertices, faces and edges. If we want each vertex to have a material specification, we might add this line to the end of the properties for a vertex:

```
property material_index int
```

This integer is now an index into the list of materials contained in the file. It may be tempting for the author of a new application to invent several new elements to be stored in PLY files. This practice should be kept to a minimum. Much better is to try adapting common elements (vertices, faces, edges, materials) to new uses, so that other programs that understand these elements might be useful in manipulating these adapted elements. Take, for example, an application that describes molecules as collections of spheres and cylinders. It would be tempting define sphere and cylinder elements for the PLY files containing the molecules. If, however, we use the vertex and edge elements for this purpose (adding the radius property to each), we can make use of programs that manipulate and display vertices and edges. Clearly one should not create special elements for triangles and quadrilaterals, but instead use the face element. What if a program does not know the adjacency between faces and vertices (so-called unshared vertices)? This is where each triangle (say) is purely a collection of three positions in space, with no notion whether some triangles have common vertices. This is a fairly common situation. Assuming there are N triangles in a given object, then 3N vertices should be written to the file, followed by N faces that simply connect up these vertices. We anticipate that a utility will be written that converts between unshared and shared vertex files.