EU IST - 2001 - 032641

VIRTUAL HERITAGE: HIGH QUALITY ACQUISITION AND PRESENTATION 3D

ViHAP3D

Representation Tools

Project Acronym: ViHAP3D	Project number: IST 2001-32641
Deliverable Date of Delivery: August/03	Workpackage: WP4.4
Contractual Date of Delivery: August/03	Actual Date of Delivery: July/03
Nature: Report	Authors: ISTI-CNR

PLY format description plus VCG extensions

Abstract

VIHAP3D application exchange data using different file formats, one of the most used is the PLY file format. The PLY is a format used to store 3D polygonal models created by Stanford; the format itself is quite customizable to specific needs. VCG introduced some extensions to the format in order to better work with models from 3D scanning.

This document, **Deliverable 4.4.2** covers the basic of PLY format plus the VCG extensions.

Deliverable **4.4, First Release of Representation Tools** is composed by

- **4.4.1** Simplification Tool: MeshSimplify v.1 with User Manual
- **4.4.2** PLY fileformat description

Intro

This document presents the Stanford standard PLY polygon format with VCG extensions. The PLY is a file format for storing 3D objects models that are described as a collection of polygons. A formal description of the standard can be found at Stanford website (www.graphics.stanford.edu). VCG, following the standard guidelines, introduced some extensions to deal with custom data associated to the 3D model, mainly regarding 3D scanning.

PLY Philosophy

The main idea of PLY is to give a flexible data storage format, easy to read and write. To obtain this, the PLY file structure contains a header area and a data area. In the header are described which kind and how many "elements" are used to represent the object in the data area. The data area contains all the elements specified in the header that represents the object.

Base of the PLY format are "elements" and "properties"; the 3D model is composed by a number of elements, each one bearing some properties. Vertices and faces are basic examples of elements: a vertex element, for example, would contain its own coordinates and its own color as properties.
There can be other elements and additional properties for each element but an application able to use them correctly is required: each application, following its own needing, decide what to save to (and also what to load from) PLY files.

Examples of new elements could be edges, cells (lists of pointers to faces) and materials (ambient, diffuse and specular colors and coefficients).

File Structure

A PLY file is composed of:

- HEADER
  - declarations
- DATA
  - Vertex List
  - Face List
  - (List of other elements)

The header contains all information necessary to the file parsing; the header is always in ASCII format. The header includes a description of each element type, including the element name (e.g. **`vertex`**), 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.

Each line of the header is a carriage-return terminated ASCII string that begins with a keyword. The header begins with the string **`ply<cr>`** and it's terminated with the string **`end_header<cr>`**; the characters **`ply<cr>`** must be the first four characters of the file, since they serve as the fileOS 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 used in the data area, and within each element description is the specification of the "properties". Then generic element description has this form:

element *<element-name> <number-in-file>*
property *<data-type> <property-name-1>*
property *<data-type> <property-name-2>*
property *<data-type> <property-name-3>*
...

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	Alternative name	Type	Num of bytes
char	***int8***	*Character*	*1*
uchar	***uint8***	*unsigned character*	*1*
short	***int16***	*short integer*	*2*
ushort	***uint16***	*unsigned short integer*	*2*
int	***int32***	*Integer*	*4*
uint	***uint32***	*unsigned integer*	*4*
float	***float32***	*single-precision float*	*4*
double	***float64***	*double-precision float*	*8*

These byte counts are important and must not vary across implementations in order for these files to be portable. In some ply files, mainly the ones coming University of North Carolina group, the name in the first column are substitued by the ones in the second column. VCG files by defaul follow use the first column naming.

There is a special form of property definitions to specify the list data type:

property list *<numerical-type> <numerical-type> <property-name>*

An example of this is:

property list uchar int vertex_index

This row defines a property called **vertex_index.** It specifies how many and the kind of values we'll find in the list. In this case we'll have a list composed by a number, stored as *<uchar>*, of values and each value is stored as an integer. In this example, each integer is a vertex index.

A simple cube

Below is the complete ASCII description for a cube. The header of a binary version of the same object is simply obtained substituting the keyword ascii with **binary_little_endian** or **binary_big_endian.** The comments in brackets are just annotations to this example.

```
ply
format ascii 1.0                    {ascii/binary, format version number}
comment created by HandWrite        {comments keyword specified, like all lines}
element vertex 8                    {define "vertex" element, 8 of them in file}
property float x                    {vertex contains float "x" coordinate}
property float y                    {vertex contains float "y" coordinate}
property float z                    {vertex contains float "z" coordinate}
element face 12                     {there are 12 "face" elements in the file}
property list uchar int vertex_indices  {"vertex_indices" is a list of ints}
end_header                          {delimits the end of the header}
-1 -1 -1                            {start of vertex list}
1 -1 -1
1 1 -1
-1 1 -1
-1 -1 1
1 -1 1
1 1 1
-1 1 1                              {end of vertex list}
3 0 2 1                             {start of face list}
3 0 3 2
3 4 5 6
3 4 6 7
3 1 6 5
3 1 2 6
3 0 4 3
3 3 4 7
3 0 1 5
3 0 5 4
3 2 3 7
3 2 7 6                             {end of face list}
```

Keywords List

Below is a complete list, alphabetically sorted, listing the key-words used in standard PLY format file:

Keyword	Description	Example of usage
ascii	The object is stored in ascii format	*format **ascii** 1.0*
binary_big_endian	The object is stored in binary format (Big Endian)	*format **binary_big_endian** 1.0*
binary_little_endian	The object is stored in binary format (Little Endian)	*format **binary_little_endian** 1.0*
comment	Simple comment string	***comment** Hello world*
element	Starts an element definition	***element** vertex 5* *property …* *…*
end_header	Delimits the end of the header	*ply* *…* *…* ***end_header***
format	Starts file format definition	***format** ascii 1.0*
list	Special property type	*element face 5* *Property **list** uchar int vertex_index*
obj_info	Reserved but not used	
ply	First characters in a PLY file. Delimits the beginning of the header	***ply*** *format …* *…*
property	Define one property for an element	*element vertex 5* ***property** float x* ***property** float y* ***property** float z*

VCG PLY Extensions

According with the PLY philosophy, VCG added some extension and applied some variations to the standard PLY format. There are new key-words and new standard elements that are parsed by VCG applications.

There is a restriction in the polygon type: VCG programs are able to work on triangles only, instead of generic polygon; this means that faces are always made up by three vertices.

Texture managing

Texturing it's not directly supported by PLY format: some addings have been made to the standard.
Texture coordinates are stored using a face property (**`texcoord`**); coordinates are defined for each wedge, in a triangle face are defined three coordinate pairs (one for each vertex). In the header a line like this should be used:

```
property list uchar float texcoord
```

The texture file is specified using a line in the header that begins with the **`comment`** keyword. Standard applications just ignore the **`comment`** keyword, while VCG applications parse **`comment`** searching for **`TextureFile`** or **`TextureNormalFile`** keywords.

`comment` [**`TextureFile`** | **`TextureNormalFile`**] <*imagefile*>

- **`TextureFile`** means that imagefile is a texture that will be attached to the mesh,
- **`TextureNormalFile`** means that imagefile is a normal map where each texel encodes a normal in RGB space.

In both cases textures coordinates are to be supplied for each triangle and the file is assumed to be in the same dir of the ply file. To allow a easy renaming of ply files and textures, the special imagefile name **"<this>"** is interpreted as a placeholder of the name of the ply file without the .ply extension. For example, in a ply file called "**dummy.ply**" the line

```
comment TextureFile <this>.png
```

means that the texture called **dummy.png** has to be loaded.

Elements and properties

Here is the list of PLY elements and properties used by VCG applications:

Vertex element:

Property Name	Datatypes Allowed	Description
`x`	float	Vertex coordinates
`y`		
`z`		
`flags`	int	Flags are used in VCG applications to

		mark vertices. For a complete list of flags see section FLAGS
quality	float	Can be used to give quality information for a vertex
red	uchar	RGB components and Alpha channel
green		
blue		
alpha		

Face element:

Please note that there exists an alternate spelling for the vertex_indices property, defined in the standard, that reads **vertex_index**. All the most recent VCG applications are able to work with both formats. VCG applications by default save using **vertex_indices**

Property Name	Datatypes Allowed	Description
vertex_indices	list uchar int	Vertex list, VCG extensions support only lists of 3 vertices (triangles)
flags	int	Flags are used in VCG applications to mark faces. For a complete list of flags see section FLAGS
quality	float	Can be used to give quality information for a face
texcoord	list uchar float	List of texture coordinates; there are 3 pairs for each face. Texture coordinates are referred to wedges
color	list uchar float	face color
texnumber	int	index of texture used for that face (valid if multiple texture have been specified in the header)

Tristrip Element:

In some of the ply files coming from the Stanford University, the 'face' element is substituted by a 'tristrips' element, that describe all the triangular face of the model as a set of triangle strips. Usually just a single strip is present and a '-1' index inside the strip is used to restart the strip. Supported only in loading.

Property Name	Datatypes Allowed	Description
vertex_indices	list int int	The list of vertexes composing the strip.

Camera Element:

A new element has been defined to include in the file information on the 3D acquisition. These kinds of information (position and parameters of the 3D camera) are significant only for single range map models.
To include a camera in a PLY file, in the header should be added a line like:

```
element camera 1
```

followed by camera properties; it is mandatory to specify all properties.

Property Name	Datatypes Allowed	Description
view_px	float	x, y, z coordinates of a vector which represent the camera position
view_py		
view_pz		
x_axisx	float	x, y, z coordinates which represent the camera right vector position
x_axisy		
x_axisz		
y_axisx	float	x, y, z coordinates which represent the camera up vector position
y_axisy		
y_axisz		
z_axisx	float	x, y, z coordinates which represent the camera front vector position
z_axisy		
z_axisz		
focal	float	Focal length
scalex	float	Image scaling factor
scaley		
scalez		
centerx	float	Pin-hole position
centery		
viewportx	int	Viewport dimension
viewporty		
k1	float	Radial lens distortion
k2		
k3		
k4		

Meaning of Flags property:

The **flags** property is used in VCG applications to mark vertices or faces. Flags are a bit field which represent the status of a vertex or face. The flags are used for several purposes by applications.
There can be 8 differents flags for each vertex and 10 differents flags for each face.

Vertices Flags	Hex Value	Bit Description
DELETED	0x0001	The vertex is deleted from the mesh
NOTREAD	0x0002	The vertex of the mesh is not readable
NOTWRITE	0x0004	The vertex is not modifiable
MODIFIED	0x0008	The vertex is modified
VISITED	0x0010	Can be used to mark the visited vertex
BORDER	0x0020	Indicates that the vertex lies in a border face
NOTMANIFOLD	0x0100	The vertex belongs to an edge which is shared by three or more faces

Faces Flags	Hex Value	Bit Description
DELETED	0x00000001	The face is deleted from the mesh
NOTREAD	0x00000002	The face of the mesh is not readable
NOTWRITE	0x00000004	The face is not modifiable
MODIFIED	0x00000008	The face is modified
VISITED	0x00000010	Can be used to mark the visited face
SELECTED	0x00000020	The face is selected
BORDER0	0x00000040	Edge from vertex 0 to vertex 1 is in the mesh border
BORDER1	0x00000080	Edge from vertex 1 to vertex 2 is in the mesh border
BORDER2	0x00000100	Edge from vertex 2 to vertex 0 is in the mesh border
NORMX	0x00000200	This flag is on if the face normal is closer to the x axis with respect to orientation
NORMY	0x00000400	This flag is on if the face normal is closer to the y axis with respect to orientation
NORMZ	0x00000800	This flag is on if the face normal is closer to the z axis with respect to orientation
COMPLEX0	0x00001000	Edge from vertex 0 to vertex 1 contains a vertex with the flag NOTMANIFOLD
COMPLEX1	0x00002000	Edge from vertex 1 to vertex 2 contains a vertex with the flag NOTMANIFOLD
COMPLEX2	0x00004000	Edge from vertex 2 to vertex 0 contains a vertex with the flag NOTMANIFOLD
FEATURE0	0x00008000	UNUSED
FEATURE1	0x00010000	UNUSED
FEATURE2	0x00020000	UNUSED
MHEDGE0	0x00040000	UNUSED
MHEDGE1	0x00080000	UNUSED
MHEDGE2	0x00100000	UNUSED

Examples

The following examples will show you how to use VCG properties and flags in PLY files.

File	Comments	Description
ply format ascii 1.0 comment created by VCG element vertex 8 property float x property float y property float z element face 12 property list uchar int vertex_indices end_header -1 -1 -1 … -1 1 1 3 0 2 1 … 3 2 7 6	begin of file ply data format and version info simple comment element declaration(8 vertices) property declaration element declaration(12 faces) property declaration end of header delimiter x y z coordinates of vertex 0 x y z coordinates of vertex 7 face 0, 3 vertices, vertex 0 vertex 2 vertex 1 face 11, 3 vertices, vertex 2 vertex 7 vertex 6	A simple cube
ply format ascii 1.0 comment created by VCG element vertex 8 property float x property float y property float z property uchar red property uchar green property uchar blue element face 12 property list uchar int vertex_indices end_header -1 -1 -1 255 0 0 … -1 1 1 0 0 0 3 0 2 1 … 3 2 7 6	property declaration red component property declaration green component property declaration blue component x y z coordinates of vertex 0 r g b components of vertex 0 x y z coordinates of vertex 7 r g b components of vertex 7	A simple cube with colored vertices
ply format ascii 1.0 comment created by platoply element vertex 8 property float x property float y property float z property uchar red property uchar green property uchar blue property uchar alpha element face 12 property list uchar int vertex_indices end_header -1 -1 -1 255 0 0 50 … -1 1 1 0 0 0 50 3 0 2 1 … 3 2 7 6	property declaration alpha component x y z coordinates of vertex 0 r g b and alpha components of vertex 0 x y z coordinates of vertex 7 r g b and alpha components of vertex 7	A simple cube with colored vertices and each with alpha component.

ply format ascii 1.0 comment created by VCG element vertex 8 property float x property float y property float z element face 12 property list uchar int vertex_indices property list uchar float color end_header -1 -1 -1 … -1 1 1 3 0 2 1 3 255 0 0 … 3 2 7 6 3 255 127 0	property declaration color (list) face 0, 3 vertices, vertex 0 vertex 2 vertex 1, r g b components face 11, 3 vertices, vertex 2 vertex 7 vertex 6, r g b components	A simple cube with colored faces.
ply format ascii 1.0 comment created by VCG element vertex 8 property float x property float y property float z property float quality element face 12 property list uchar int vertex_indices end_header -1 -1 -1 1.0 … -1 1 1 0.09 3 0 2 1 … 3 2 7 6	property declaration quality x y z coordinates of vertex 0, quality x y z coordinates of vertex 7, quality	A simple cube with quality float in each vertex.
ply format ascii 1.0 comment created by VCG element vertex 8 property float x property float y property float z element face 12 property list uchar int vertex_indices property float quality end_header -1 -1 -1 … -1 1 1 3 0 2 1 0.02 … 3 2 7 6 0.04303		A simple cube with quality float in each face.
ply format ascii 1.0 comment created by VCG element vertex 8 property float x property float y property float z property int flags element face 12 property list uchar int vertex_indices end_header -1 -1 -1 32 … -1 1 1 0 3 0 2 1 … 3 2 7 6	property flags declaration x y z coordinates of vertex 0, flag SELECTED on x y z coordinates of vertex 7, flag SELECTED on	A simple cube with SELECTED flag on in vertex 0.

ply format ascii 1.0 comment created by VCG element vertex 8 property float x property float y property float z property int flags element face 12 property list uchar int vertex_indices end_header -1 -1 -1 2 1 -1 -1 2 1 1 -1 2 -1 1 -1 2 … -1 1 1 0 3 0 2 1 … 3 2 7 6	property flags declaration x y z coordinates of vertex 0, flag SELECTED on x y z coordinates of vertex 1, flag SELECTED on x y z coordinates of vertex 2, flag SELECTED on x y z coordinates of vertex 3, flag SELECTED on x y z coordinates of vertex 7, flag SELECTED off (each flag is off)	A simple cube with NOTREADABLE flag on in vertices 0, 1, 2, 3.
ply format ascii 1.0 comment created by VCG element vertex 8 property float x property float y property float z element face 12 property list uchar int vertex_indices property int flags end_header -1 -1 -1 … -1 1 1 3 0 2 1 64 3 0 3 2 64 3 4 5 6 64 … 3 2 7 6 0	property flags declaration face 0, 3 vertices, vertex 0 vertex 2 vertex 1, flag BORDER0 on face 1, 3 vertices, vertex 0 vertex 3 vertex 2, flag BORDER0 on face 2, 3 vertices, vertex 4 vertex 5 vertex 6, flag BORDER0 on face 2, 3 vertices, vertex 4 vertex 5 vertex 6, flag BORDER0 off (each flag is off)	A simple cube with BORDER0 flag on in faces 0, 1, 2.
ply format ascii 1.0 comment created by VCG comment TextureFile torrepisa.png element vertex 8 property float x property float y property float z element face 12 property list uchar int vertex_indices property list uchar float texcoord end_header -1 -1 -1 … -1 1 1 3 0 2 1 6 1 0 0 1 0 0 3 0 3 2 6 1 0 1 1 0 1 … 3 2 7 6 6 0 0 0 0 0 0	torrepisa.png will be used as a texture property declaration texcoord(list) face 0, 3 vertices, vertex 0 vertex 2 vertex 1, texture coordinates face 1, 3 vertices, vertex 0 vertex 3 vertex 2, texture coordinates face 7, 3 vertices, vertex 0 vertex 2 vertex 1, texture coordinates (only one texel will be attached on this face)	A simple cube with 1 texture mapped in 2 faces.

ply format ascii 1.0 comment created by VCG comment TextureFile torrepisa.png comment TextureFile other.png element vertex 8 property float x property float y property float z element face 12 property list uchar int vertex_indices property list uchar float texcoord property int texnumber end_header -1 -1 -1 … -1 1 1 3 0 2 1 6 1 0 0 1 0 0 0 3 0 3 2 6 1 0 1 1 0 1 0 3 4 5 6 6 1 0 0 0 0 1 1 3 4 6 7 6 1 0 0 1 1 1 1 … 3 2 7 6 6 0 0 0 0 0 0 0	torrepisa.png will be used as texture0 other.png will be used as a texture1 property declaration texcoord(list) face 0, 3 vertices, vertex 0 vertex 2 vertex 1, texture coordinates (referred to texture0) face 1, 3 vertices, vertex 0 vertex 3 vertex 2, texture coordinates (referred to texture0) face 2, 3 vertices, vertex 4 vertex 5 vertex 6, texture coordinates (referred to texture1) face 3, 3 vertices, vertex 4 vertex 6 vertex 7, texture coordinates (referred to texture1) face 11, 3 vertices, vertex 2 vertex 7 vertex 6, texture coordinates (referred to texture0)	A simple cube with 2 texture mapped in 4 faces. Each texture is mapped in two triangles.
ply format ascii 1.0 comment created by VCG comment TextureNormalFile bunny.png element vertex 150 property float x property float y property float z element face 251 property list uchar int vertex_indices property list uchar float texcoord end_header -9.27384 12.2733 0.927055 … 6.07459 6.23413 0.91502 3 1 0 5 6 0.179245 0.748047 0.127081 0.748047 0.154828 0.501953 3 149 144 145 6 0.347947 0.251953 0.389012 0.251953 0.352386 0.373047	bunny.png will be used as map to generate face normals property declaration texcoord(list)	A simple “bunny” with 1 texture mapped in the whole mesh. Texture color is used for generate faces normals

ply format binary_little_endian 1.0 comment VCGLIB generated element camera 1 property float view_px property float view_py property float view_pz property float x_axisx property float x_axisy property float x_axisz property float y_axisx property float y_axisy property float y_axisz property float z_axisx property float z_axisy property float z_axisz property float focal property float scalex property float scaley property float centerx property float centery property int viewportx property int viewporty property float k1 property float k2 property float k3 property float k4 element vertex 28894 property float x property float y property float z property int flags property uchar red property uchar green property uchar blue property uchar alpha property float quality element face 54435 property list uchar int vertex_indices property int flags property list uchar float color end_header	camera element declaration	Here is an example of camera use: camera element declaration must be placed before other elements declaration. All properties are required and must be specified in the correct order, as shown.