A Conversion Pipeline: From Laser-scanned Data to High Fidelity Rendering

Gavin Ellis*

Computer Graphics Group
University of Bristol
Bristol, UK

Abstract

Accurate, textured, 3D computer models of objects are of increasing interest in many disciplines, e.g. archeology. Laser scanners provide a very precise way of gathering data for such models; however, a number of steps are necessary before this new laser data can be imported into a computer model that is ready for rendering.

This work discusses the processes undertaken to develop a simple to use, seamless method of transforming raw laser-scanned data (with associated texture information) to the three dimensional modeling system, *Maya*, for further editing, or directly to high fidelity rendering in the *Radiance* lighting visualisation suite.

Keywords: Laser Scanning, 3D file formats, texture mapping

1 Introduction

It is clear that computer graphics can help us better understand the past by visually recreating archaeological sites under accurate and authentic conditions [8]. Such graphics are only scientifically valid if the components of the scene are created in an exact, scientific manner. Methods such as 3D scanning and rendering with high fidelity graphics offer a faithful way to recreate scenes from the past as shown in Figure 1 [4][3]. However, the images they produce are limited by the data they work with: in many cases the modeling of complex objects and surfaces using modeling packages and artists alone is not sufficiently accurate. Laser scanners provide a method of capturing accurate information about object's surfaces. The Minolta 910 laser scanner [13], which we used, has an accuracy of under a millimetre. It also captures colour texture information for the model by way of the CCD[1]

Alias-Wavefront's Maya is used as a powerful 3D modeling package which allows the scanned models to be manipulated after scanning [12]. Radiance is used because it is a renderer capable of producing high fidelity images by accurate lighting visualisation [6].

*ellis@cs.bris.ac.uk

[1]*Charge Coupled Device* - photosensitive grid, as used in digital cameras.

There was, however, no existing way to import the data acquired by the laser scanner into Radiance, nor a way to import the captured textures into Maya. In addition, writing texture images to Radiance `PIC` files directly saves time and complication of converting via a number of intermediate stages.

This paper is concerned with facilitating and simplifying the process necessary for conversion from the VMRL 1.0 format files produced by a Minolta 910 laser scanner to formats that can be understood by Maya and Radiance. In Section 2 we look at the existing software and the file formats under consideration. Section 3 explains how the sample data was collected and then section 4 describes the conversion process employed to convert VRML files to Maya and Radiance files. Results of the work are in Section 5 and our conclusions are presented along with avenues of further research in Section 6.

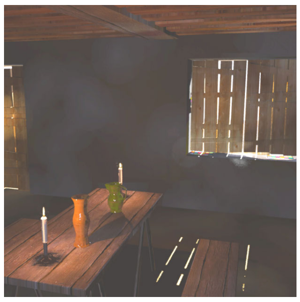

Figure 1: A High Fidelity Render of a Medieval House

2 Previous Work

Technical improvements have greatly increased the accuracy of laser scanners and the speed with which they capture data [5]. Many scanners now also capture texture information about the target's surface which can be applied to the 3D model [2]. The increased accuracy makes 3D scanning a useful tool in archeology. It allows accurate records of artefacts to be made and also allows them to be put in the context of their day; for example, lighting rooms with contemporary light sources may cast different shadows and cause different contrasts on a subject compared with modern lights [7].

The VRML standard was proposed in 1994 [17]. One facet of the language is that it provided a standard for storing 3D models. VRML version 1.0 is exported by the scanner. It is a human-readable ASCII based file standard. Many 3D modeling programs are capable of exporting to VRML files, but only a very small number are capable of reading in. The two applications we were interested in, Maya and Radiance, were not able to read the VRML 1.0 files generated by the scanner software.

Maya has a `wrl2ma` conversion program which converted VRML 2.0 files, but not VRML 1.0. *Polytrans* and *SoftImage* would convert VRML 1.0 to Maya but were both financially prohibitively expensive and would not convert to Radiance's file formats [14][15].

This work made extensive use of the VRML 1.0 and Radiance's `PIC` file specifications[17][10][9].

3 Acquiring Data

The quality of the image produced by a renderer is only as good as the information given to describe the scene. The scanner captures 3D information about the shape of a surface of a model by measuring the distance from the scanner to points on the surface. With an accuracy of less than a millimetre the laser scanner can create realistic meshes from objects. After measuring the 3D depth data, the Vivid 910 uses its CCD to capture a 2D image in the same way a digital camera does. The CCD relies on ambient light to illuminate the target.

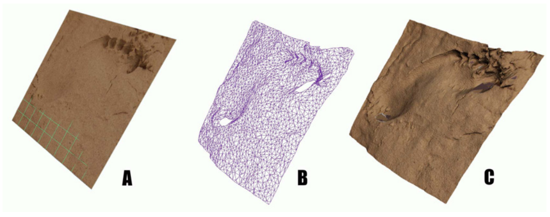

Figure 2: The components of a scanned model

The scanner software then matches points on the photograph to points in the surface mesh and exports the data as a VRML file which contains both the mesh and bitmap. Figure 2 shows an exploded view of the components, a 2D image(A) is mapped onto a 3D mesh(B) and the combined textured model(C).

4 Separating Data

The VRML files which are output from the laser scanner contain all the texture and mesh data in one file. Our aim is to separate these into three independent files.

VRML files have two formats: the binary format, which has a smaller file size but is only machine readable and ASCII, which is larger but can be viewed and edited in a standard text editor. Fortunately the scanner uses the VRML 1.0 ASCII format. VRML files typically have the extension .wrl. All comments within a VRML file are prefixed with a '#'. The first line (the header) of a VRML 1.0 file is a comment line:

#VRML V1.0 ascii

The software that generates the VRML files we will be converting then adds a second comment:

# Polygon Editing Tool

These two lines allow a simple check to be made to ensure the file is valid for further processing. The third line contains a comment identifying the name of the object. The important fields we are concerned with are:

- `Texture2`
- `TextureCoordinate2`
- `Coordinate3`
- `coordIndex`
- `textureCoordIndex`

The fields are case sensitive. If there is no texture information saved in the file then the Texture2, TextureCoordinate2 and textureCoordIndex fields will not be present.

The conversion from VRML requires the texture to be separated from the mesh. VRML files are separated into sections relating to texture, vertex and face information. Three components are created from the VRML file:

- The 2D texture (stored as a bitmap).
- A mesh file (stored in OBJ format).
- A material file (used by Radiance and Maya to tie the texture and mesh together).

4.1 Texture Data

4.1.1 Extracting the VRML Texture

Texture is saved in the `Texture2` field. The VRML 1.0 specification allows the file to store all the image data directly inside the file. To identify this, the first line in the Texture2 field is:

```
image <width> <height> <format>
```

The `<format>` value determines the format of the texture data that follows. The Minolta 910 always uses format '3' which is a three channel (red, green and blue), one byte per channel, hexadecimal layout:

`0xRRGGBB`
e.g. `0XFF0000` would be pure red.

The scanner exports VRML files with texture embedded in them. The texture is a merged composite of all the photographs taken during the scans that make up the model. The number of these values is dependant on the size of the image, the total given by:

$$< \texttt{width} > \times < \texttt{height} >$$

The first entry in this list is the bottom left because this represents the origin (0, 0) in the U V coordinate system. The scan lines then go left to right, bottom to top.

4.1.2 Creating Traditional Bitmap Files

The texture information is read into an array. This can then be used to create texture files in traditional bitmap formats e.g. `PNG, TIFF`. Our application was written in Java [16], so it was straightforward to save these files using the basic Java API.

4.1.3 Creating `PIC` Files

`PIC` files are the way Radiance stores all images, including those of the scenes it has rendered. Because the main aim of the work was to smooth the pipeline from scanning an object to rendering it, it made sense to be able to export `PIC` files directly. This means that the images are ready to be mapped directly onto the object's surface.

Because the `PIC` file is being created from a file without any gamma information, a value of 2.2 is assumed. This is the common value used in PCs [1] and is also the default value used in the Radiance programs (ra_ppm, ra_tiff to convert `TIFF` and `PPM` bitmaps to `PIC`) [9].

In a high fidelity image generation, the colour and illumination values are at least as important as the content, if not more so. So, rather than just recording one byte for each of the red, green and blue values, the `PIC` file system uses 32 bits. One byte for each of the RGB channels, and a fourth 1-byte component E, the Exponent. This means that the 'dominant' colour is described most accurately. As with the Radiance utilities, a pre-calculated table of exponents is stored in memory to save computation time.

4.2 3D Data

4.2.1 Vertices

The computer model of the object that has been scanned is made up of vertices. Each vertex is a point in three-dimensional space, so is described by three orthogonal coordinates. In VRML, vertices are collected in the 'Coordinate3' field, in a list

e.g. `x1, y1, z1, x2, y2, z2, ...`

In `OBJ` vertices are identified separately with a 'v' at the start of the vertex:

e.g.
```
v x1 y1 z1
v x2 y2 z2
```

`Texture2`

$$\begin{tikzcd} & \texttt{.mtl}(material file) \\ \texttt{.png}(texture file) \end{tikzcd}$$

$$\begin{tikzcd} 0xRRGGBB \arrow[r] \arrow[dr, "image"'] & \left.\begin{matrix}\texttt{.mtl}(materialfile)\\ \texttt{.png}(texturefile)\end{matrix}\right\} Maya \\ & \left.\begin{matrix}\texttt{.mat}(materialfile)\\ \texttt{.pic}(texturefile)\end{matrix}\right\} Radiance \end{tikzcd}$$

VRML format $\longrightarrow$ .OBJ format

`TextureCoordinate2`

$$uv \xrightarrow{Indextopointinimage} \texttt{vt}\,uv$$

`Coordinate3`

$$xyz, \xrightarrow{Vertices} \texttt{v}\,xyz$$

`IndexedFaceSet`

`coordIndex`

$$a, b, c, -1 \xrightarrow{Faces} \texttt{f}\,abc$$

`textureCoordindex`

$$i, j, k, -1 \xrightarrow{Textureonface} \texttt{f}\,a/ib/jc/k$$

Figure 3: The Conversion Process From VRML to Maya `OBJ`

4.2.2 Texture Mapping

The modeling program needs to know what texture should be applied to the surface of the model. VRML uses UV texture mapping (described earlier). In VRML files textures are listed in the TextureCoordinate2 field as a list :

e.g. *u1 v1,*
u2 v2, ...

Each of these coordinates refers to a colour on the texture map. The order of this list is important because the colour of the faces is given in the 'textureCoordIndex' by referring to the point in this list where the coordinates are to describe the point on the texture map where the colour information is.

The `OBJ` format also uses UV mapping. Unlike the VRML format it does not have the capability to store the texture in-line in the file. At the start of the file the keyword `loadmtl` *mtl_file* tells the program that the material file identifying the materials used in that object is called *mtl_file*. Then, another keyword, `usemtl` *material* indicates that material *material* from that material file should be used to colour the vertices that follow appear on the lines that follow (until another `usemtl` or the end of file is reached). `OBJ` files use the same principle to define the texture of a vertex. A list of vertex textures is made and then referenced later in the face description. In `OBJ` files the vertices are separated by `vt` at the start of the reference:

e.g. `vt` *u1 v1*
`vt` *u2 v2*

In the files exported by the scanner this section is missing. This means that there is no accurate way of fitting a texture to the object. For example, if the texture was exported as a spherical projection and it was then applied to the object in Maya with the default method, say cylindrical projection, then the resulting rendered picture would be incorrect.

4.2.3 Faces

So that the object appears as solid rather than a cluster of points, the points are joined into faces. Any number of vertices can be joined into a face, but fortunately the scanner software program only uses between three and five points per face which simplifies the conversion process. Faces also have a texture associated with them. This texture is derived from the textures recorded at each of the vertices that comprise the face and is then 'stretched' over the face. In both VRML and `.OBJ` files the vertices and textures are referred to by their position in the list of vertices given at the start of the file. The order in which the vertices are listed for each face determines which side of the face is 'inside' and which is 'outside'. VRML files have a field called `IndexedFaceSet` which contains the information about faces. It is then separated into at least two parts (the files exported originally only used the two sub-fields discussed here). The `CoordIndex` field contains the basic face data. It lists the vertices making up each face. The list of vertices for each face is terminated with a `-1` (because the vertex is referenced according to its position in a list there can obviously be no `-1`). The 'textureCoordIndex' field contains the texture data for the faces described in the previous `CoordIndex` field. Consequently there must be the same number of faces and the same number of vertices in each face. Order is important again because the first face in this list refers to the first entry in the `CoordIndex` list. This time, instead of vertices, textures are listed (i.e. referring to the UV texture listed in the TextureCoordinate2 field. Again, -1 denotes the end of a face. In `OBJ` files the vertices and textures are referred to by their position in the list of vertices given earlier in the file. Again, the the order determines which faces are 'inside' and 'outside'. `OBJ` files describe faces by keeping all the information together. Each face starts with 'f' and then the vertices are listed. If texture information exists, then the reference is placed next to the vertex separated by a '/'.

e.g.
`f` *vertex1/texture1 vertex2/texture2 vertex3/texture3 ...*

In `OBJ` files exported by the scanner, no texture information associated with the face.

4.3 Material Files

Although material files are used by both Maya and Radiance to describe how a texture should be applied to a model, the format of the two files differ.

4.3.1 Maya `.MTL` files

Maya stores the information about how a material should be applied to an object in a file separate from the definition of the object itself. The `OBJ` file has within it the parameters `loadmtl` and `usemtl` which tell Maya which material file to use. Comment lines begin with a '#', and the file can define several textures within it, each with the form:

`newmtl` *texture name*
`Kd` *R G B*
`Ka` *R G B*
`Ks` *R G B*
`map_Kd` *bitmap file name*

Where:
`Ks` is the specular coefficient (colour of highlight).
`Ka` is the ambient coefficient (background or fill light).
`Kd` is the diffuse coefficient (base colour).
`map_Kd` provides the name of the bitmap file containing the texture map.

4.3.2 Radiance `.MAT` files

Radiance can import `.OBJ` files and makes use of the `usemtl` and `loadmtl` keywords to load its own material file. `.MAT` files are the material description files used by Radiance. Again, they are are simple human-readable ASCII files. Comment lines begin with a '#'.

Just as the `.MTL` files describes how a bitmap should be applied as a texture in Maya, `.MAT` describes how the bitmap should be applied in Radiance. However, Radiance only accepts `.PIC` format files as textures.

```
# Section 1
void colorpict patt
7 clip_r clip_g clip_b \
           texture.pic  .  Lu Lv
0
```

```
0

# Section 2
patt plastic material_name
0
0
5 0.25 0.25 0.25 0 0
```

The first line of the first section defines canvas as a material with a texture applied to it. The second line starts with a number identifying the number of arguments on the line (as is usual in Radiance). The `clip_r/g/b` limit the intensity of the texture map to prevent it from glowing in the scene. The *picfile* is the `PIC` file of the texture which we extract from the VRML file. `Lu` and `Lv` specify the mapping system to use, i.e. UV (as described in section 3.2) rather than tiling the texture over the mesh.

The second section defines the material *material_name*, which is identified in the `.OBJ` file by the `usemtl` keyword, as being an opaque (plastic) material of the type canvas, i.e. having the texture projected onto it. The last line of this section identifies the colour of this material. This is largely irrelevant because the texture takes the place of any colour value.

5 Results

Once the texture map has been extracted using our approach it can be mapped onto a model. The scanner software can export Maya `OBJ` meshes, but they have no texture mapping information. Consequently when Maya tries to map the texture onto the mesh it does not 'fit', as shown in Figure 4. Our software extracts the texture mapping information (the `TextureCoordinate2` and `vt` data described earlier).

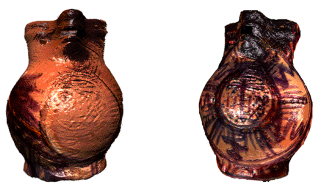

Figure 4: A Jug Model Rendered In Maya Using Scanner Software *[left]* and with Texture Map Using our method *[right]*

Loading the `OBJ` file and texture in Maya revealed a minor error in the scanner's software. Originally the conversion program was written according to the VRML specification. Comparing the output bitmap to the texture displayed by the scanner showed that the bitmap was upside-down. Adjustments were made to our program to allow the user to select which way up to export the texture. The default setting is now correct for Maya and Radiance.

When this technique was being developed, Radiance (version 3.5) was not capable of mapping textures onto

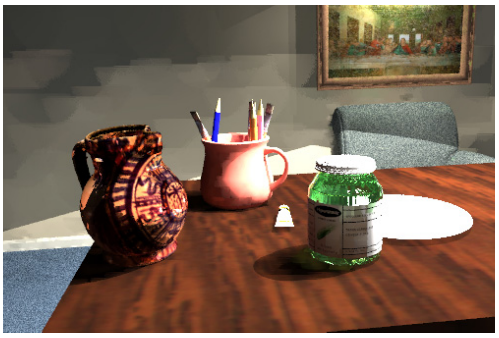

Figure 5: The High Fidelity Render of the Jug Model in a Scene Generated in Radiance

complex polygon surfaces. Mapping onto simple planes worked correctly, but not when applied to more complicated polygons. At the September 2003 Radiance conference version 3.6beta was released with an improved `obj2mesh` program. This new release fixes the problem and so, when used in conjunction with out software, allows high fidelity rendering of scanned objects, as shown in Figure 5.

6 Conclusions & Further Work

The conversion pipeline developed enables texture and 3D data captured by a Minolta scanner to be converted to formats which can be used with Maya and soon with Radiance, enabling the data to be combined with other complex models and rendered in high fidelity scenes.

Maya has a function to reduce the number of vertices in a polygon model. This function works with polygons created in Maya, but does not work with the polygon meshes from the scanner. A variety of methods were investigated using a small model (with 100 vertices) to try and resolve this problem: Neither the `OBJ` files exported by the scanner or our software can be reduced. Vertex normals were calculated and included in a file (these are not normally included by either program because they were deemed unnecessary) in case this made a difference, but to no avail. Currently the only way to reduce the size of a model is to load the original file in the scanner's software and reduce it there. It would be more convenient to include polygon reduction in our software or write a plug-in for Maya to do this.

In many archaeological reconstructions it is necessary to consider the period lighting that would illuminate a scene. This means that the effects of modern, artificial lights captured by the CCD must removed. The complete model is formed by stitching the separate scans together. The problem is that the edges that are stitched at different distances from the spotlight used to illuminate the object. Consequently, when the texture is merged a tide mark forms at the join. In Figure 6 the left half is more shaded

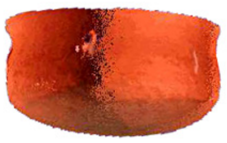

Figure 6: A bowl showing lighting boundary

than the right and the join is clearly visible. For the most part, this problem can be overcome by careful, equal illumination of the target. However, as the finished composite model comprises several scans, selecting which scan's texture image should be applied to the final model could improve the results.

A similar problem occurs with highlights on the objects' surface. If two scans are taken of an object which is rotated between the scans, one highlight will appear on each texture map. If the scans are carefully organised to overlap then the area with the highlight from the first scan can be replaced by the same area from the second scan without the highlight. The highlight from the second scan would then be replaced by the same region in the third scan and so on. This would be a very useful feature but it would be hard to implement because by the time the texture has been merged by the scanner software it is too late to modify the individual scans. One possible way to approach this problem would be to export the unmerged image from each scan then choose the parts of the image that you want to apply to the mesh (i.e. those images without the highlights).

7 Acknowledgments

Thanks go to my supervisor, Alan Chalmers for his help throughout this project. To Patrick Ledda and Veronica Sundstedt for use of their models (Figures 1 & 5) and Richard Gillibrand for help with Radiance and Maya.

References

[1] D. Blatner and B. Fraser, *Real World Photoshop 6* Peachpit Press 2001.

[2] M. Callieri, P. Cignoni, F. Ganovelli, C. Montani, P. Pingi, R. Scopigno ”VCLab's Tools for 3D range data processing.” In *Proceedings of VAST 2003*, 2003

[3] A. Chalmers, K. Devlin, D. Brown, P. Debevec, P. Martinez, G. Ward, ”Recreating the Past”, *SIGGRAPH 2002 Course, 27*, ACM SIGGRAPH, July 2002.

[4] G. Godin, J.-Angelo Beraldin, J. Taylor, L. Cournoyer, M. Rioux, S. El-Hakin, R. Baribeau, F. Blais, P. Boulanger, J. Domey and M. Picard, ”Active Optical 3D Imaging for Heritage Applications”, In *Computer Graphics And Applications 'Art History and Archaeology'* pp. 38-50 September/October 2002

[5] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Shade and D. Fulk, ”The Digital Michelangelo Project: 3D Scanning of Large Statues”, In *Siggraph Proceedings 2000*, 2000

[6] Ann McNamara, A. Chalmers, T. Troscianko and I. Gilchrist, ”Comparing Real and Synthetic Scenes using Human Judgements of Lightness.”, In *Proceedings of the Eurographics Workshop in Brno*, B Piroche and H Rushmeier, editors, Eurographics, June 2000.

[7] Ann McNamara, A. Chalmers and D. Brown, ”Light and the Culture of Medieval Pottery”, In *Proceedings of the International Conference on Medival Archaeology*, 1997.

[8] E. Vote, D. Acevedo Feliz, D. Laidlaw and M. Sharp Joukowsky, ”Discovering Petra: Archaeological Analysis in VR” In *Computer Graphics And Applications 'Art History and Archaeology'* September/October 2002

[9] G. Ward, Radiance source code. *http://radsite.lbl.gov/radiance/HOME.html*

[10] G. Ward, ”Real Pixels”. In *Graphics Gems II* pp.80-83 Edited by James Arvo Academic Press - 1991

[11] G. Ward Larson and R. Shakespeare, *Rendering with Radiance The Art and Science of Lighting Visualisation*, Morgan Kaufman 1998.

[12] Alias-Wavefront website. (2003) *http://www.alias.com/eng/index.shtml*

[13] Konica-Minolta website. (2003) *http://www.minolta-3d.com*

[14] Okino Computer Graphics, Polytrans website (2003) *http://www.okino.com/conv/conv.htm*

[15] Softimage website (2003) *http://www.softimage.com/home/*

[16] Sun Microsystems website. (2003) *http://www.java.com/en/index.jsp*

[17] VRML 1.0 File Specification. (2003) *http://www.web3d.org/technicalinfo /specifications/VRML1.0/index.html*