Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

# A generic approach of integrating 3D models into virtual manufacturing*

Hwa-Jen YAP†[1,2], Zahari TAHA[3], Siti Zawiah Md DAWAL[1,2]

([1]*Department of Engineering Design and Manufacture, Faculty of Engineering, University of Malaya, Kuala Lumpur 50603, Malaysia*)

([2]*Centre for Product Design and Manufacturing, Faculty of Engineering, University of Malaya, Kuala Lumpur 50603, Malaysia*)

([3]*Faculty of Manufacturing Engineering & Technology Management, Universiti Malaysia Pahang, Kuantan26300, Malaysia*)

†E-mail: hjyap737@um.edu.my; jyap737@yahoo.com

Received Mar. 24, 2011; Revision accepted July 28, 2011; Crosschecked Dec. 8, 2011

**Abstract:** Various 3D modeling software has been developed for design and manufacturing. Most of the commercially available software uses native file formats, which may not be able to be read or understood by other software. This paper deals with the development of a generic approach of a 3D model conversion program for virtual manufacturing (VM), using a lexical analyzer generator Lex and the Open Graphic Library (OpenGL). The program is able to convert 3D mesh data between four universal file formats, i.e., Stereolithography (STL), Virtual Reality Modeling Language (VRML), eXtensible Markup Language (XML), and Object (OBJ). Simple assembly functions can be applied to the imported models. The quaternion angle is used for object rotation to overcome the problem of gimbal lock or a loss of one degree of rotational freedom. The program has been validated by importing the neutral format models into the program, applying the transformation, saving the new models with a new coordinate system, and lastly exporting into other commercial software. The results showed that the program is able to render and re-arrange accurately the geometry data from the different universal file formats and that it can be used in VM. Therefore, the output models from a VM system can be transferred or imported to another VM system in a universal file format.

**Key words:** Virtual manufacturing, Lex, OpenGL, Three-dimensional mesh data, Quaternion, Neutral format
**doi:**10.1631/jzus.C11a0077 **Document code:** A **CLC number:** TP391.9; TH122

## 1 Introduction

The rapid growth of advanced technologies has made computer-aided design (CAD) software an essential tool in the field of design, simulation, and optimization. The development of readily affordable desktop based CAD software makes the 3D CAD model indispensable in representing a design of an item. Many types of CAD modeling software have been used in various areas including automotive, manufacturing, design, and construction. Either free or licensed, most of the CAD software may not be able to access native CAD data produced by modeling software.

Three-dimensional model translation software can be used to solve the above problems. The software is categorized mainly based on the functions and supported file types. The typical functions of the software are viewing, translating, rotating, assembling, editing, and file format converting. The application of the software is mainly for design review and model exchange. The software increases the communication efficiency between the designer and manufacturers in collaborative engineering environments. It is able to translate and share 3D models data, and is time saving for manufacturers, suppliers, and consultants.

Conversely, virtual reality is the technology of generating, visualizing, and interacting with a computer generated environment. Unlike CAD systems, there are no standard file formats for virtual reality applications, such as virtual manufacturing (VM).

* Project (No. RG060/09AET) supported by the University of Malaya Research Grant (UMRG)

Most of the virtual reality (VR) packages use native file formats to create their virtual environments. Also, there are no standard methods or algorithms for converting 3D models from CAD modeling software to VM. File format conversion can be done by the translation of CAD models into native file formats using commercial modeling software, a library or database approach (Whyte *et al*., 2000).

### 1.1 Problem statements

Usually, a native file format consists of the algorithms and data structure for that particular software, which may not be readable or visualized by others. Therefore, most of the CAD software cannot access native formats produced by each other. To read different types of files, different software needs to be used to extract the geometric information before being exported into a VM system. This greatly increases the costs of purchasing the software licenses.

Some users may try to make it easier by translating the native file to universal files that can be loaded and opened by most of the CAD software. However, the imported universal files cannot be edited when they are opened in the native system. For example, 3D models in Stereolithography (STL) format can be opened with ProEngineer but cannot be edited using it. Moreover, although multiple universal files can be loaded, the assembly of the universal files cannot be saved in a universal format in a native system. For example, ProEngineer can assemble parts for STL files and Object (OBJ) files, but the assembly file can be saved in a native format only.

There is a different approach to integrating CAD with VR, without undertaking any file conversion. The VR hardware can be connected directly to the commercial CAD software, which was accomplished by the collaboration between Fakespace and Dassault Systèmes for the immersive visualization display devices within the CATIA application (Berta, 1999). In contrast, Schilling *et al.* (2006) introduced a middleware framework to synchronize the geometry and metadata between the VR and CAD system. However, it is applicable to dedicated hardware and software only.

Many researches in VM are using the Virtual Reality Modeling Language (VRML) format and are published on the World Wide Web (Ong *et al.*, 2002; Jezernik and Hren, 2003; Wang and Tian, 2007). However, these researches support only a single format. Multiple formats need to be supported and exchanged, including Stereolithography (STL) from rapid prototyping, eXtensible Markup Language (XML) from arbitrary data structure, and Object (OBJ) files from animation.

Although solid modeling can be performed within the virtual environment (VE), the compatibility and transferability of one VE to another (a different platform) were not discussed (Zhong *et al.*, 2005). Also, it takes a long time to build a new VE from scratch, for CAD modeling, and not import directly from the existing VE (Peng, 2007).

### 1.2 Objectives

In a previous work a design visualization and manipulation concept has been implemented for a universal file format. Furthermore, the CAD native database can be translated to and from the STL, ASCII, VRML, and XML formats with an anaglyph imaging technique to create stereo 3D with red-blue glasses (Yap *et al*., 2008).

In this paper, a generic algorithm has been used to read and extract data from STL, VRML, XML, and OBJ files using a lexical analyzer generator. At the same time, the Open Graphic Library (OpenGL) is used to re-generate a 3D model that can be used in VM. The system also supports the assembly function for the universal file format. Models can be loaded, translated, and rotated. Extracted data, in part or assembly format, can be saved into different universal file formats. Therefore, the output models from one VM system can be transferred or imported to another VM system in a universal file format.

## 2 Standard file formats

File formats can be categorized into standard file formats and native file formats. The native file format of an application is proprietary and these types of files are not meant to be transferred to another format, which is the default file format used by a specific software. The data can be viewed after they have been translated into a universal file or in a neutral format, such as STL, STEP, IGES, OBJ, and VRML. The CAD native database can be translated into a universal CAD geometry file format, which can be opened and read by most of the CAD modeling software (Rule, 1996). Four universal file formats were studied and converted in this work: STL, VRML,

XML, and OBJ.

STL is supported by most of today's CAD software packages and has been widely used for rapid prototyping and computer-aided manufacturing (Jacobs, 1995). It can be divided into two standard data formats, ASCII and binary. The ASCII file is readable and can be modified by a text editor, which makes it easier to spot errors.

VRML is designed to be used on the Internet according to ISO/IEC 14772-1:1997 and 14772-2: 2004, and is intended to be a universal interchange format for integrated 3D graphics and multimedia. It is a file format for describing interactive 3D objects and worlds, containing vertices and edges for a 3D polygon, which can be specified together with the surface color, UV mapped textures, shininess, transparency, etc. (Hartman and Wernecke, 1996).

XML is very famous and popular nowadays. It is a combination of text and extra information including structure, layout, etc. It is categorized as a general-purpose markup language with the primary purpose of facilitating the sharing of structured data across different information systems. The structured data that can be written in XML format include vector graphics, mathematical equations, and server application programming interfaces (APIs). Due to its simplicity and flexibility, XML is widely used for communicating data between applications, especially via the Internet (Williamson, 2008).

The OBJ file is used to transfer geometric data back and forth between the Advanced Visualizer and other applications. OBJ defines the geometry and other properties for objects in Wavefront's Advanced Visualizer. Object files can also be used to transfer geometric data back and forth between the Advanced Visualizer and other applications. Object files can be in ASCII format (*.obj) or binary format (*.mod). The later release of the OBJ file format supports both polygonal objects and free-form objects. Polygonal geometry uses points, lines, and faces to define objects while free-form geometry uses curves and surfaces (Bourke, 2010). In general, there are four types of OBJ file formats. The difference among the formats is the data format in polygon surfaces, where the numbers are indexes into the arrays of vertex positions, texture coordinates, and normal. A number may be omitted if, for example, texture coordinates are not being defined in the model. Table 1 shows the different types of OBJ file formats.

**Table 1 Different types of OBJ file formats**

| Data | OBJ file format |
|---|---|
| Vertices/Texture/ Normal | f 64/64/64 1/1/1 49/49/49<br>f 64/64/64 62/62/62 1/1/1 |
| Vertices/Texture | f 21/21 10/10 20/20<br>f 10/10 21/21 13/13 |
| Vertices/Normal | f 16//16 17//15 18//14<br>f 22//22 23//21 24//20 |
| Vertices | f 5 1 3 or f 5 7 8 6<br>f 5 3 4 f 1 5 6 2 |

## 3 Software development

The file conversion software is able to read four types of 3D models, which are STL, VRML, XML, and OBJ formats. First, the input file is scanned using a lexical analyzer generator. The data extracted from the input file are processed and stored in a container. Then, the model is rendered using the developed algorithms. The user can manipulate the 3D model using the keyboard and the mouse. Lastly, the imported 3D model is converted and saved into a new array. The architecture of the developed software is shown in Fig. 1.

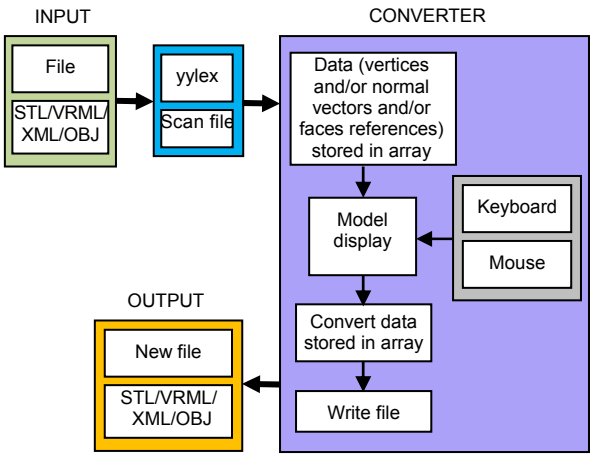


**Fig. 1 The architecture of the developed software**

Basically, the development of the software is categorized into four stages (Fig. 2). First, a generic algorithm is developed to read the 3D models. Second, a graphic rendering algorithm is developed to re-render the imported models. Third, an object transformation algorithm is constructed. Lastly, a generic method is used to convert models into a universal file format.

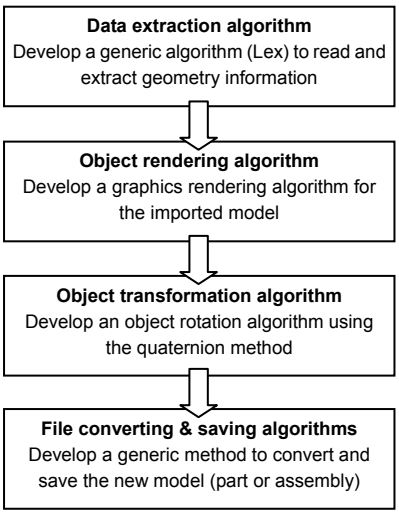


**Fig. 2 Four stages for software development**

## 4 Data extraction using Lex

The lexical analyzer generator Lex is a program generator designed for lexical processing of characteristic input streams. Lex turns the rules into a program, and any source not intercepted by Lex is copied into the generated program (Levine *et al.*, 1992). There are two steps in compiling a Lex source program. First, turn the generated program into the host general-purpose language. Then, the rule is compiled and loaded with a library of Lex subroutines. Lex is easy to use because the user who wants to analyze an input using a string manipulation language will need only to define the string expressions to be matched and the corresponding actions. The user does not need to write an input analyzing program or an often inappropriate string handling language.

Lex is used to obtain the mesh data from 3D model rendering. The data of meshes consist of vertices, textures, and normal. When writing a Lex specification, a set of patterns that Lex matches against the input needs to be created. Lex itself does not produce an executable program; instead, it translates the Lex specification into a file containing a C routine.

Basically, the required data to be extracted for graphic rendering of VRML, XML, and OBJ are the same, i.e., coordinates and point references. However, as discussed in the previous section, OBJ may have a different file format. In contrast, the STL file format has a different data structure, where the data are stored in terms of facet normal and vertices. Fig. 3 shows examples of Lex specification codes for VRML and XML file formats.

```
"#VRML" {
v=0;
t=0; }

"point [" {
point=1;
i=0;
v++; }

"," {
if (point==1) {
i=0;
v++; }
}

"coordIndex [" {
point=0;
coordIndex=1;
j=0;
t++; }

"-1]" {
coordIndex=0;
}
```

(a)

```
"<mesh>" {
v=0;
t=0; }

"<vertex id=\"v-" {
v++;
i=0; }

"<length units=\"cm\">" {
Length=1;
Factor=0.01; }

"<length units=\"m\">" {
Length=1;
Factor=1.0; }

"</length>" {
Length=0; }

"<triangle" {
t++;
j=0; }

"<vertex-ref ref=\"v-" {
Ref=1; }

"\"/>" {
Ref=0; }
```

(b)

**Fig. 3 Examples of Lex specification codes for VRML (a) and XML (b)**

## 5 Object rendering using OpenGL

OpenGL is an API, but not a programming language like C or C++. OpenGL allows programmers to write applications that access graphics hardware; i.e., OpenGL is the software interface to graphics hardware. The interface consists of about 150 distinct commands, which are used to specify the object and operations needed to produce interactive 3D applications (Shreiner *et al.*, 2007). OpenGL's structure is similar to those of most modern APIs, including the Programmer's Hierarchical Interactive Graphics System (PHIGS) and Graphical Kernel System (GKS). In this research, the model is in a mesh form and is rendered using the triangle form/loop. Lighting and texture mapping can be added through the normal and texture information.

### 5.1 Displaying STL model

In the STL file format, the polygon's normal and vertices are organized in a triangle loop. The data are

taken directly from the container and extracted using the lexical analyzer. Fig. 4 shows the general function that is used to render the STL mesh model in the system.

```
for (int i=0; i<stl.size(); i++) {                    // Draw STL file
      Nor[0]=stl[i].xNormal;
      Nor[1]=stl[i].yNormal;
      Nor[2]=stl[i].zNormal;

glBegin(GL_TRIANGLES);
      glColor3f(1.0, 0.0, 0.0);
      glNormal3fv(Nor);
      glVertex3f(stl[i].x1, stl[i].y1. stl[i].z1);
      glVertex3f(stl[i].x2, stl[i].y2. stl[i].z2);
      glVertex3f(stl[i].x3, stl[i].y3. stl[i].z3);
glEnd();
}
```

Vertices to form a basic triangle

**Fig. 4 General function to render the STL mesh model**

### 5.2 Displaying VRML, XML, or OBJ model

The OBJ file does not contain the polygon's normal vectors, and the cloud data are organized vertex by vertex followed by the polygon's point references. To use the OpenGL command, the vertices need to be arranged in triangular form. Hence, the coordinates have to be taken according to the point references.

The cross product of two vectors gives a vector perpendicular to the two vectors, with the orientation given by the right-hand rule. To calculate the normal vector of a plane, two directional vectors in the same plane are described. In this case, the plane is a polygon triangle. Given the three vertices of the polygon $\boldsymbol{p}_1=(x_1, y_1, z_1)$, $\boldsymbol{p}_2=(x_2, y_2, z_2)$, and $\boldsymbol{p}_3=(x_3, y_3, z_3)$, two possible directional vectors representing the triangle are

$$\boldsymbol{d}_1=(x_1-x_2, y_1-y_2, z_1-z_2), \tag{1}$$

$$\boldsymbol{d}_2=(x_1-x_3, y_1-y_3, z_1-z_3). \tag{2}$$

The normal to the triangle can be determined by the cross product of the two possible directional vectors:

$$\boldsymbol{N}=\begin{vmatrix} \mathrm{i} & \mathrm{j} & \mathrm{k} \\ x_1-x_2 & y_1-y_2 & z_1-z_2 \\ x_1-x_3 & y_1-y_3 & z_1-z_3 \end{vmatrix}. \tag{3}$$

Let

$$\boldsymbol{N}=(x_\mathrm{n}, y_\mathrm{n}, z_\mathrm{n}).$$

Then

$$x_\mathrm{n}=(y_1-y_2)(z_1-z_3)-(z_1-z_2)(y_1-y_3),$$

$$y_\mathrm{n}=(z_1-z_2)(x_1-x_3)-(x_1-x_2)(z_1-z_3),$$

$$z_\mathrm{n}=(x_1-x_2)(y_1-y_3)-(y_1-y_2)(x_1-x_3).$$

Fig. 5 shows the functions that are used to render the models in the system.

```
for (int k=0; k<PointRef.size(); k++) {                  // Draw OBJ file
      float x1=(Points[(PointRef[k].p1)-1].x);
      float y1=(Points[(PointRef[k].p1)-1].y);
      float z1=(Points[(PointRef[k].p1)-1].z);

      float x2=(Points[(PointRef[k].p2)-1].x);
      float y2=(Points[(PointRef[k].p2)-1].y);
      float z2=(Points[(PointRef[k].p2)-1].z);

      float x3=(Points[(PointRef[k].p3)-1].x);
      float y3=(Points[(PointRef[k].p3)-1].y);
      float z3=(Points[(PointRef[k].p3)-1].z);

      Nor[0]=(y1-y2)*(z1-z3)-(z1-z2)*(y1-y3);
      Nor[1]=(z1-z2)*(x1-x3)-(x1-x2)*(z1-z3);
      Nor[2]=(x1-x2)*(y1-y3)-(y1-y2)*(x1-x3);
glBegin(GL_TRIANGLES);
      glColor3f(0.9,0.3,0.6);
      glNormal3fv(Nor);
      glVertex3f(x1,y1,z1);
      glVertex3f(x2,y2,z2);
      glVertex3f(x3,y3,z3);
glEnd();
}
```

Coordinate had been taken according to point references

Calculation of the normal vector

Formation of the basic triangle

**Fig. 5 Functions used to render the models in the system**

## 6 Modeling transformation

The OpenGL transformations are used to produce a desired scene for viewing. All transformations are represented as 4×4 matrices. A vertex is always represented as a homogeneous coordinate. The modeling transformation is used to position and orientate the model. It converts the object coordinate systems to a common world coordinate system.

### 6.1 Rotation in arbitrary axis using quaternion

The total rotation can be calculated by multiplying together the representation of the individual rotations, but the order of operands is important and multiplication has to use the same consistent definitions. Hence, in this research, the quaternion method is used to perform the rotating of the object. Quaternions have four dimensions (each quaternion consists of four scalar numbers), one real dimension and three imaginary dimensions, and can be represented as

$$\boldsymbol{Q}=w+x\mathrm{i}+y\mathrm{j}+z\mathrm{k}. \tag{4}$$

The default rotation angle is the Euler angle. Therefore, conversion is required to change the angle into a quaternion angle before multiplying the transformation matrices. Table 2 shows the conversion of the Euler angle to quaternion with a related rotation angle.

**Table 2 Conversion of an Euler angle to a quaternion**

| Rotation about | Rotation angle | Quaternion pure attitude rotation |
|---|---|---|
| $z$ | Attitude angle $a$ | $\boldsymbol{Q}_a=\cos(a/2)+\mathrm{k}\sin(a/2)=c_2+\mathrm{k}s_2$ |
| $y$ | Heading angle $h$ | $\boldsymbol{Q}_h=\cos(h/2)+\mathrm{j}\sin(h/2)=c_1+\mathrm{j}s_1$ |
| $x$ | Bank angle $b$ | $\boldsymbol{Q}_b=\cos(b/2)+\mathrm{i}\sin(b/2)=c_3+\mathrm{i}s_3$ |

Let the order of rotation be about the $z$-, $y$-, and then $x$-axis. Hence, the cross product of quaternion is

$$\boldsymbol{Q}=(\boldsymbol{Q}_h\times\boldsymbol{Q}_a)\times\boldsymbol{Q}_b. \tag{5}$$

As given in Table 2,

$$c_1=\cos(h/2),\ c_2=\cos(a/2),\ c_3=\cos(b/2),$$
$$s_1=\sin(h/2),\ s_2=\sin(a/2),\ s_3=\sin(b/2).$$

Using Eq. (5) and Table 2, it is found that

$$\begin{aligned}\boldsymbol{Q}=&(c_1c_2c_3-s_1s_2s_3)+\mathrm{i}(s_1s_2c_3+c_1c_2s_3)\\&+\mathrm{j}(s_1c_2c_3+c_1s_2s_3)+\mathrm{k}(c_1s_2c_3-s_1c_2s_3).\end{aligned} \tag{6}$$

As stated in Eq. (4),

$$w=c_1c_2c_3-s_1s_2s_3,\ \ x=s_1s_2c_3+c_1c_2s_3,$$
$$y=s_1c_2c_3+c_1s_2s_3,\ \ z=c_1s_2c_3-s_1c_2s_3.$$

The multiplication of two unit quaternions ($\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$) will result in another unit quaternion ($\boldsymbol{Q}_3$) that represents the combined rotation. When modeling the rotation of an object, the algorithm orientation is

$$\boldsymbol{Q}_3=\boldsymbol{Q}_1\times\boldsymbol{Q}_2, \tag{7}$$

where

$$\boldsymbol{Q}_1=(w_1, x_1, y_1, z_1),$$
$$\boldsymbol{Q}_2=(w_2, x_2, y_2, z_2),$$
$$\boldsymbol{Q}_3=(w_3, x_3, y_3, z_3).$$

Therefore,

$$w_3=w_1w_2-x_1x_2-y_1y_2-z_1z_2,$$
$$x_3=w_1x_2+x_1w_2+y_1z_2-z_1y_2,$$
$$y_3=w_1y_2+y_1w_2+z_1x_2-x_1z_2,$$
$$z_3=w_1z_2+z_1w_2+x_1y_2-y_1x_2.$$

### 6.2 Final position and orientation

The final position of the object can be obtained by the calculated combined transformations. The combined transformation consists of a number of successive translations and rotations about a fixed reference frame axis or a moving current frame axis. These translations involve the data stored in the variables of $\boldsymbol{P}_{xyz(\mathrm{new})}$. Once the function is called, the data need to be recalculated to a new set of data by using translation and rotational matrix formulae. Let $\boldsymbol{P}_{xyz}$ be the original position, **Trans**($x_{\mathrm{Loc}}$, $y_{\mathrm{Loc}}$, $z_{\mathrm{Loc}}$) the amount of translation, and **Quat**($x$, $y$, $z$, $\theta$) the angle and axis rotation (quaternion method). We have

$$\boldsymbol{P}_{xyz(\mathrm{new})}=\mathbf{Trans}(x_{\mathrm{Loc}}, y_{\mathrm{Loc}}, z_{\mathrm{Loc}})\cdot\mathbf{Quat}(x, y, z, \theta)\cdot\boldsymbol{P}_{xyz}. \tag{8}$$

Therefore,

$$\boldsymbol{P}_{xyz(\mathrm{new})}=\begin{bmatrix}1&0&0&x_{\mathrm{Loc}}\\0&1&0&y_{\mathrm{Loc}}\\0&0&1&z_{\mathrm{Loc}}\\0&0&0&1\end{bmatrix}$$
$$\cdot\begin{bmatrix}1-2y^2-2z^2&2xy-2wz&2xz+2wy&0\\2xy+2wz&1-2x^2-2z^2&2yz-2wx&0\\2xz-wy&2yz-2wx&1-2x^2-2y^2&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}x_0\\y_0\\z_0\\1\end{bmatrix}.$$

Since

$$\boldsymbol{P}_{xyz(\mathrm{new})}=(x_{\mathrm{new}}, y_{\mathrm{new}}, z_{\mathrm{new}}),$$

we have

$$\begin{aligned}x_{\mathrm{new}}=&(1-2y^2-2z^2)x_0+(2xy-2wz)y_0\\&+(2xz+2wy)z_0+x_{\mathrm{Loc}},\\y_{\mathrm{new}}=&(2xy+2wz)x_0+(1-2x^2-2z^2)y_0\\&+(2yz-2wx)z_0+y_{\mathrm{Loc}},\\z_{\mathrm{new}}=&(2xz-2wy)x_0+(2yz-2wx)y_0\\&+(1-2x^2-2y^2)z_0+z_{\mathrm{Loc}}.\end{aligned}$$

### 6.3 Rotation in the virtual environment

The positions and rotations of the virtual object are controlled by a six-degree-of-freedom (6-DOF) digitizer. It is a pen-shaped input device with no mechanical arm or optical marker. The data (positions and orientations) are triggered and activated through the push button switch mounted on the handle. Fig. 6 shows the figures of the virtual object being rotated in an arbitrary axis.

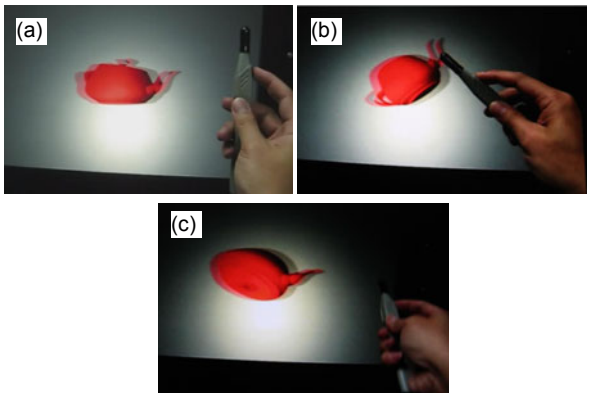


**Fig. 6 Arbitrary axis rotation in a virtual environment**
(a) Default position; (b) Rotation about *z*-axis; (c) Rotation about an arbitrary axis

## 7 Format converting and saving

The data arrangements are different for the STL format and the other three formats. The data structure of the STL format allows numerous repeated vertices. This is because most of the vertices are shared by a number of polygons in mesh, which does not occur for VRML/XML/OBJ. A smaller file size can be obtained and decreases the loading time. Therefore, the repeated points in the original data have to be eliminated by creating the corresponding point references. The following are the steps:

1. Two point lists (PointReference and Corrected PointReference) are created in which the data are taken from the STL file. Both contain the same data.

2. According to the point lists created, the first triangle contains vertices (1, 2, 3), the second triangle contains vertices (4, 5, 6), and so on (Fig. 7).

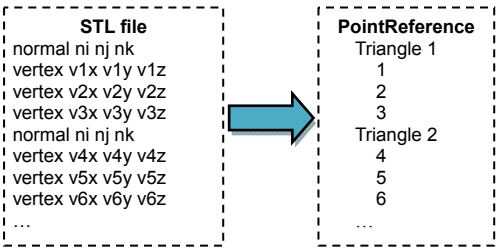


**Fig. 7 Data arrangement for the STL format—triangle loops meshing**

3. All the repeated points are erased from one of the point lists. A new point reference list of the model based on the summarized point list (Corrected PointReference) is generated by comparing the points in the original point list (PointReferenc) based on point reference with all points in the summarized point list. If the points being compared are the same, that particular point reference will be changed to the location of the point in the new point list. Fig. 8 shows the corrected triangle loops with no repeated points.

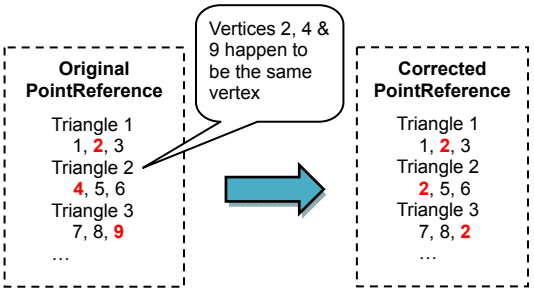


**Fig. 8 Data correction for repeated points**

Another feature of the system is importing several files together and exporting the data into a single file—the assembly file becomes a part file. As an example, the software is able to combine up to four OBJ files into a single part file. Fig. 9 shows an example of assembling two OBJ files using a simple algorithm as follows:

1. Start a loop function and write in the data that had been stored in the array when reading the file.

2. Add the data of vertices extracted from the second file below the data of vertices from the first file. Then, recalculate the total number of vertices.

3. The data of faces depend on the number of vertices. Hence, the data of vertices extracted from the second file will add to the total number of vertices in the first file, and are added below the data of faces from the first file. Then, recalculate the total number of triangles.

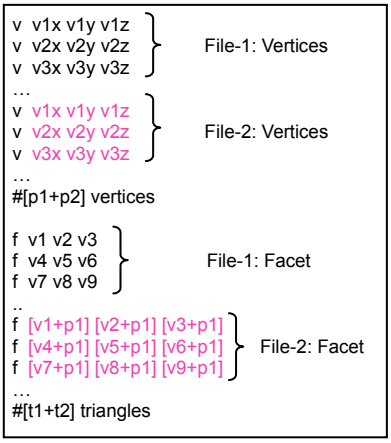


**Fig. 9 Example of assembling two OBJ files**

## 8 Validations

Software validation is essential to ensure that the developed algorithm and programming codes are being executed properly. The functionality of the developed software in this project has been tested and validated. The software validation is divided into four sections: data extracting, 3D model rendering, modeling transformation, and file conversion/saving. Fig. 10 shows the flow of program validation using commercial CAD modeling software. Fig. 11 shows the example 3D assembly model in VRML format, in which the 3D models are being imported and/or rotated, and then being saved and viewed through the Internet browser.

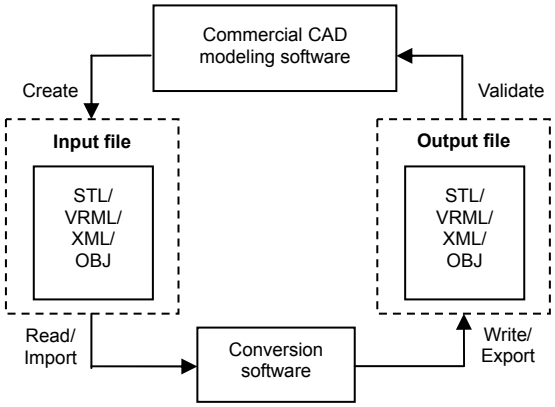


**Fig. 10 Program validation flow using commercial CAD modeling software**

## 9 Conclusions

This paper deals with the design and development of a generic approach of integrating various 3D models into virtual reality including conversion, modeling, and data exchange. This approach has been validated using commercial software. An input file has been scanned and the geometric information is extracted using a lexical analyzer generator.

Rendering algorithms for a universal 3D model are also implemented. These algorithms use the OpenGL API. The system allows model viewing and transformations such as translation, rotation, and scaling. The 3D model can be viewed from different angles and positions. The system can also perform simple assembly of parts and save into one universal part file.

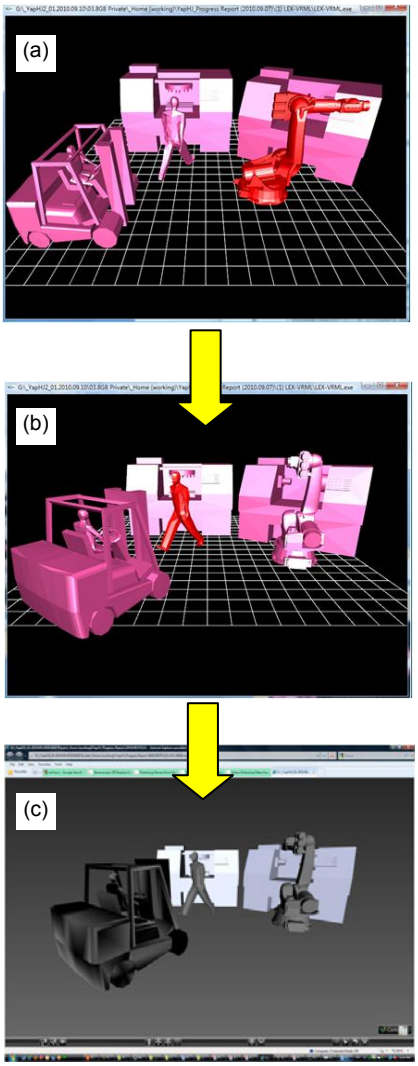


**Fig. 11 Validation of the VRML assembly model**
(a) Import in a virtual environment; (b) Rotation of models; (c) View by the Internet browser

## References

Berta, J., 1999. Integrating VR and CAD. *IEEE Comput. Graph. Appl.*, **19**(5):14-19. [doi:10.1109/38.788793]

Bourke, P., 2010. Object Files (.obj). Available from http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj [Accessed on Dec. 20, 2010].

Hartman, J., Wernecke, J., 1996. The VRML 2.0 Handbook: Building Moving Worlds on the Web. Addison-Wesley, Redwood City, California.

ISO/IEC 14772-1:1997. Virtual Reality Modeling Language (VRML)–Part 1: Base Functionality and Text Encoding.

ISO/IEC 14772-2:2004. Virtual Reality Modeling Language (VRML)–Part 2: Base Functionality and All Bindings for the VRML External Authoring Interface.

Jacobs, P.F., 1995. StereoLithography and Other RP&M Technologies: from Rapid Prototyping to Rapid Tooling.

Society of Manufacturing Engineers, Princeton, New Jersey, USA.

Jezernik, A., Hren, G., 2003. A solution to integrate computer-aided design (CAD) and virtual reality (VR) databases in design and manufacturing processes. *Int. J. Adv. Manuf. Technol.*, **22**(11-12):768-774. [doi:10.1007/s00170-003-1604-3]

Levine, J., Tony, M., Brown, D., 1992. Lex & Yacc. O′Reilly & Associates, Sebastopol, California.

Ong, S.K., Jiang, L., Nee, A.Y.C., 2002. An Internet-based virtual CNC milling system. *Int. J. Adv. Manuf. Technol.*, **20**(1):20-30. [doi:10.1007/s001700200119]

Peng, Q., 2007. Virtual Reality Technology in Product Design and Manufacturing—the Design and Implementation of a Course for the Graduate Study. The Canadian Design Engineering Network (CDEN) and the Canadian Congress on Engineering Education (CCEE).

Rule, K., 1996. 3D Graphic File Formats: a Programmer's Reference. Addison-Wesley, Redwood City, California.

Schilling, A., Kim, S., Weissmann, D., Tang, Z., Choi, S., 2006. CAD-VR geometry and meta data synchronization for design review applications. *J. Zhejiang Univ.-Sci. A*, **7**(9):1482-1491. [doi:10.1631/jzus.2006.A1482]

Shreiner, D., Mason, W., Neider, J., Davis, T., 2007. OpenGL Programming Guide, the Official Guide to Learning OpenGL. Version 2.1, 6th Edition. Addison-Wesley Professional, Boston, Massachusetts.

Wang, Q.Y., Tian, L., 2007. A systematic approach for 3D VRML model-based assembly in Web-based product design. *Int. J. Adv. Manuf. Technol.*, **33**(7-8):819-836. [doi:10.1007/s00170-006-0494-6]

Whyte, J., Bouchlaghem, N., Thorpe, A., McCaffer, R., 2000. From CAD to virtual reality: modelling approaches, data exchange and interactive 3D building design tools. *Autom. Construct.*, **10**(1):43-55. [doi:10.1016/S0926-5805(99)00012-6]

Williamson, H., 2008. XML: the Complete Reference. McGraw-Hill, New York.

Yap, H.J., Taha, Z., Liew, K.S., Ghazilla, R.A.R., Ahmad, N., 2008. Development of a 3D CAD Model Conversion and Visualization System Using Lexical Analyzer Generator and OpenGL. Proc. Asia Pacific Industrial Engineering and Management Society, p.2700-2706.

Zhong, Y., Ma, W., Shirinzadeh, B., 2005. A methodology for solid modelling in a virtual reality environment. *Robot. Comput.-Integr. Manuf.*, **21**(6):528-549. [doi:10.1016/j.rcim.2004.09.003]