Degree project

INTERACTIVE GALLERY

Author: Murat BAŞAR
Date: 2012-12-28
Subject: Computer Science
Level: Bachelor
Course code: 2DV00E
Abstract
Today, there are many high-level interactive applications and products around the world which are developed by using programming languages or software. Basically, games, virtual museums, educational applications, interactive architectural products are the simplest examples about these interactive solutions. In this thesis, interactive gallery means moving objects in a three-dimensional room. Objects can be re-placed by using keyboard keys in this three-dimensional room. Thus, user can move these interactive objects in accordance with intended purpose.

Interactive gallery is similar with a room that has furniture, door and windows. Also, this room includes interactive objects in it. The main focal point is about how these objects can be displayed interactively and how different methods are used, when these processes are started to be made. This thesis helps to solve these questions.

Interactive gallery bases on model loading structure and on the following thesis, this process is parted. It starts with 3D modelling, continuous with model loading and it ends with key implementation.

Keywords
Three-dimensional design with Autodesk 3D Studio Max, Exporting .obj files, Wavefront obj files, Model loading, 3D model loading with Java OpenGL (JOGL).
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>2</td>
</tr>
<tr>
<td>1.4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>5</td>
</tr>
<tr>
<td>2.3.1</td>
<td>5</td>
</tr>
<tr>
<td>2.3.2</td>
<td>6</td>
</tr>
<tr>
<td>2.3.3</td>
<td>6</td>
</tr>
<tr>
<td>2.3.4</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>7</td>
</tr>
<tr>
<td>2.4.1</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>8</td>
</tr>
<tr>
<td>2.5.1</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>9</td>
</tr>
<tr>
<td>2.7</td>
<td>10</td>
</tr>
<tr>
<td>2.7.1</td>
<td>11</td>
</tr>
<tr>
<td>2.7.2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>15</td>
</tr>
<tr>
<td>4.2</td>
<td>15</td>
</tr>
<tr>
<td>4.3</td>
<td>15</td>
</tr>
<tr>
<td>4.4</td>
<td>15</td>
</tr>
<tr>
<td>4.5</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>17</td>
</tr>
<tr>
<td>5.1.1</td>
<td>20</td>
</tr>
<tr>
<td>5.1.2</td>
<td>23</td>
</tr>
<tr>
<td>5.1.3</td>
<td>25</td>
</tr>
<tr>
<td>5.1.4</td>
<td>26</td>
</tr>
<tr>
<td>5.1.5</td>
<td>26</td>
</tr>
<tr>
<td>5.1.6</td>
<td>27</td>
</tr>
<tr>
<td>5.1.7</td>
<td>28</td>
</tr>
<tr>
<td>5.2</td>
<td>29</td>
</tr>
<tr>
<td>5.2.1</td>
<td>30</td>
</tr>
<tr>
<td>5.2.2</td>
<td>30</td>
</tr>
</tbody>
</table>
List of Figures
Figure 2.1: The OpenGL Pipeline ................................................................. 6
Figure 2.2: The OpenGL software organization ............................................ 7
Figure 2.3: Autodesk 3D Studio Max Interface ................................................. 9
Figure 2.4: The OBJ file export options .........................................................10
Figure 3.1: Eclipse Architecture.................................................................13
Figure 5.1: Material Editor ..................................................................19
Figure 5.2: The Floor ObjModel1 ..............................................................20
Figure 5.3: The Left Wall (ObjModel3) .........................................................21
Figure 5.4: The Right Wall (ObjModel4) .......................................................22
Figure 5.5: The Back Wall (ObjModel2) .......................................................22
Figure 5.6: The Ceiling (ObjModel) .............................................................23
Figure 5.7: The Left Window (ObjModel7) ....................................................24
Figure 5.8: The Middle and the Right Window (ObjModel7, 8) ......................25
Figure 5.9: The Door (ObjModel15) ............................................................25
Figure 5.10: The Carpet (ObjModel8) ..........................................................26
Figure 5.11: The Television Table (ObjModel6) .............................................27
Figure 5.12: The Ready Models (ObjModel9, 16, 17) .....................................27
Figure 5.13: The Merge Dialog Box .............................................................28
Figure 5.14: The Paintings with Loft Option. (ObjModel5, 11, 12, 13) .............29
Figure 6.1: Result 1 (the loaded objects) ......................................................49
Figure 6.2: Result 2 (The first painting is brought to the left) .........................49
Figure 6.3: Result 3 (the top paintings are replaced) .......................................50
Figure 6.4: Result 4 (the bottom paintings are replaced) ..................................50
Figure 6.5: Result 5 (the bottom and top paintings are replaced on the right side)51
Figure 6.6: Result 6 (the bottom and top paintings are replaced on the left side)51
Figure A.1: Java SE Runtime Environment 7 Options ....................................58
Figure A.2: The Eclipse Platform..............................................................59
Figure A.3: The Implementing JOGL Step 1 .................................................60
Figure A.4: The Implementing JOGL Step 2 ..................................................60
Figure A.5: The Implementing JOGL Step 3 ..................................................61
Figure A.6: The Exporting Process Step 1 ....................................................62
Figure A.7: The Exporting Process Step 2 ....................................................62
List of Schemes

Schema 4.1: The Interactive Gallery Process ......................................................... 16
Schema 5.1: The Class diagram for the Model Loading ........................................... 30
List of Tables
Table 5.1: Reading OBJ Files ................................................................. 31
Table 5.2: Adding Vertex .............................................................. 33
Table 5.3: Storing Shape Statements .................................................. 33
Table 5.4: Reading Tuples ............................................................... 34
Table 5.5: Resizing Model Dimensions ............................................... 36
Table 5.6: Rendering Faces ............................................................... 37
Table 5.7: Finding Materials .............................................................. 39
Table 5.8: Reading and Rendering MTL Files .................................... 40
Table 5.9: Setting Materials ................................................................. 42
Table 5.10: Loading the Models ......................................................... 44
Table 5.11: Key Implementation .......................................................... 46
Table A.1: OBJ Model Class ............................................................... 63
Table A.2: Tuple3 Class ................................................................. 68
Table A.3: ModelDimensions Inner Class ......................................... 69
Table A.4: Faces Inner Class .......................................................... 71
Table A.5: FaceMaterials Inner Class ............................................... 74
Table A.6: Materials Inner Class ........................................................ 75
Table 5.7: Material Inner Class ........................................................ 77
Table 5.8: Model Import Class .......................................................... 80
Abbreviation List

OpenGL: .............................................................. Open Graphic Library
API: ............................................................. Application Program Interface
JRE: .............................................................. Java Runtime Environment
JDK: .............................................................. Java Development Kit
.OBJ: ............................................................. Wavefront Object File
.MTL: ............................................................. Material File
JWM: ............................................................. Java Virtual Machine
GL: .............................................................. Graphic Library
GLU: ............................................................ OpenGL Utility Library
GLUT: ............................................................ OpenGL Utility Toolkit
SGI: .............................................................. Silicon Graphic Library
JNI: .............................................................. Java Native Interface
JDT: .............................................................. Java Development Tool
PDE: ............................................................ Plug-in Developer Environment
SWT: ............................................................. Standard Widget Environment
VRML: ......................................................... Virtual Reality Modelling Language
1. INTRODUCTION

Today, there are high-level interactive galleries around the world. Virtual museums, games, educational applications, scientific applications and many more interactive solutions are examples that will be shown as example for basic interactive galleries. For instance, virtual museum is a museum which is based on structures of virtual realities, databases and web technologies. The designed virtual museum gives to visitor illusion of actual museum. Even if there is an unlimited distance for visitor, visitor can visit a virtual museum by using internet connection [4]. This is benefit of virtual museums because visitor can see historical heritages, historical ruins, famous paintings and information. Also, these virtual museums can be used in other areas. For instance, chemistry is a science. An interactive application can be made by using a virtual museum for educational cases. This can be about elements. Elements can be matched with their abbreviations by using moving objects on an interactive gallery. Thus, interactive gallery can be part of a virtual museum. Due to these reasons, interactive gallery can be an idea about how this process starts and how it can go to a virtual museum with this aspect.

1.1 Motivation

Interactive gallery means moving objects in a three-dimensional room for this thesis. This room has furniture, door and windows like a normal room and chosen objects can be moved in this room by clicking keyboard keys. A programming language can be used for developing yet, there are alternatives and easier ways that give realistic and better results. So, interactive gallery brings some concepts about it. One of the usable and comprehensible ones is model loading which is made by using 3D modelling, graphic libraries and coding. Model loading is using object’s information and objects can be redrawn with same codes. Thus, it provides less coding instead of writing thousands of codes.

There are also alternatives to use these concepts because there a lot of 3D modelling software, model loading methods and programming languages. It means there are a lot of alternatives to use on interactive gallery. As the first step, 3D modelling can be made with 3D software like Autodesk 3D Studio Max, AutoCAD, MilkShape 3D, Maya or Blender because these software support file formats that are used for model loading. Model loading can be made by using these file formats and each software has file formats like .obj, .dae, .blend, .stl, raw. Hence, these options enhance modelling skills to use on interactive gallery and provide learning about different model loading methods that are usable and comprehensible. For instance, Autodesk 3D Studio Max supports .obj file format which stores object’s information. Then, this information is used to be drawn object on development software by using codes. This is also called as model loading.

Java is a good programming language and it is becoming widespread. It is clear because many developers or products encapsulate it. For instance, Java-based games are very popular for computer users who like games. Moreover this, other programming languages or 3D modelling software are following Java and they are trying to find connection for it because it is platform independent, object-oriented and many more. Today, most of the 3D modelling software support file formats that can be used with
Java graphic libraries. JOGL is one of the graphic libraries and modelled 3D objects can be implemented by using this library [16].

When interactive gallery is examined, it shows similarity with virtual museums. Virtual museums can be identified as illusion of real world and it is usually implemented by using Virtual Reality Modelling Language (VRML), which is really hard to code and requires high programming skills. Also, it is placed on the internet by using databases, and web technologies. In one way, interactive gallery takes the initiative for advanced galleries and these possibilities can be settled on interactive gallery for developing advanced galleries.

1.2 Problem/s
There are many methods and platforms to develop an interactive gallery. Here, the main question is how these options can be used for developing and how a gallery can be displayed interactively. Model loading is one of the usable methods and it requires 3D modelling, using graphic libraries and coding. So, 3D modelling software, graphic libraries and programming language are required to make these steps possible.

There are also some requirements about these steps. For example, 3D modelling wants modelling skills to make realistic objects, graphic library and coding are about used programming language and it can be done by using required one (Java OpenGL (JOGL)). Therefore, the purpose of this thesis is to understand 3D interactive objects on Java OpenGL (JOGL) with the model loading way which means using 3D exported models.

1.3 Goals and Criteria
The aim of this research is to understand 3D structures with a three-dimensional building (room) on 3D modelling software and to find out how these structures can be loaded and rendered interactively with Java programming language. But, the programming and designing consists of different steps for the thesis because model loading requires different steps like modelling, using graphic library and coding. So, the main goal is splintered by some sub-goals. These sub-goals:

- **3D Modelling with Autodesk 3D Studio Max**: Three-dimensional objects are designed by using Autodesk 3D Studio Max in this step because this software allows designing 3D models for 3D applications. Also, it supports file formats that are used for model loading and interactive gallery bases on model loading structure. Moreover this, it is easy to learn, options can be found easily on its interface. So, Autodesk 3D Studio Max can give answer for one of the interactive gallery processes.

- **Installing Java and Java Runtime Environments (JRE)**: Java and necessary Java environments are installed because Java runs almost on every operating system and it supports Java-based applications. Also, Java Runtime Environment is used for running Java applications.

- **Setting up Eclipse and Java OpenGL (JOGL) Graphic Library**: Eclipse is a development software. On Eclipse, Java graphic applications can be created with Java OpenGL (JOGL) graphic library and Eclipse supports JOGL files because of its extendable architecture. So, Eclipse is installed for developing. One of the most
important parts is setting up JOGL .jar files and .dll files onto Eclipse. Thus, Java graphic applications can be written easily to be run on OpenGL graphic API.

- **Implementation for Model Loading:** In this part, 3D objects are exported into workspace file which comes with Eclipse. Then, these models are implemented and coded by using Java classes. Model loading is made with other explanation. Thus, 3D objects can be re-drawn.

- **Key Implementations:** In this part of the implementations, three-dimensional objects are integrated with keyboard keys. In this way, when the application runs, three-dimensional loading models get interactive rendering. The most important part is the moving for interactive gallery because just x and y coordinates are used for interaction.

  Each sub goal has been explained with more details in the implementation part. The process works step by step. Each step forms basis to the other step. Also, these steps run on different development platforms. Thus, sub-goals create the main goal of this thesis, at the end of the implementation part. This means, 3D models are started as graphics but, they are finished as interactive objects at the end of the all steps.

**1.4 Outline**

Chapter 2 tries to explain background of Java programming language, its advantages, OpenGL graphic API, Java JOGL graphic library, file formats, and Autodesk 3D Studio Max. Shortly, process steps are introduced here. In chapter 3, Eclipse development software and its architecture are introduced because it creates methodology and technology part for interactive gallery. The usable methods are introduced in chapter 4. Here, it explains which methods are used for developing. In chapter 5, implementations are explained with more details. Designing, exporting, model loading and key implementations are made here. Interactive results are given in chapter 6. They are figured by using pictures. Chapter 7 is discussion part. In this chapter, solutions are discussed. Chapter 8 is the conclusion of thesis and information is given in this chapter for enlarging of these 3D interactive graphic solutions.
2. BACKGROUND
Java programming language has its own graphic libraries. One of them is JOGL and it is used for conversioning OpenGL functions because OpenGL is written with C programming language. JOGL is Java interface and it allows using Java programming language for OpenGL. Thus, this programming language can be used for model loading on OpenGL API by using JOGL graphic library.

Autodesk 3D Studio Max is used for 3D modelling and it supports file formats that are used for model loading. Thus, Java’s flexible structure can be used on 3D models that are developed by using Autodesk 3D Studio Max.

During the ongoing process, this chapter tries to explain clearly what Java, its advantages, OpenGL, Java OpenGL (JOGL) and Autodesk 3D Studio Max are and how they work with their sub-structures.

2.1 Java / Java Programming Language
Today Java is known and one of the most popular programming languages. So, according to java.com; Java is a programming language and data processing platform which was released in 1995 by Sun Microsystems. The utility applications, games, commercial applications and many products are based on this technology. Many applications and websites do not work if Java is not installed on the operating system. It is a fast, reliable and safe technology. This technology is found easily today, it is free and it can be downloaded by www.Java.com web link to support Java-based applications. It shows that Java has found its own way around the world and it is getting bigger every passing day [17].

Java developers tried to find solutions for systems in the past. Java developers did this also for graphic APIs. One of them is OpenGL graphic API. So, Java developers and SGI (Silicon Graphic Library) have brought JOGL for OpenGL because it is written with C language and Java developers attempted to use this API. Today, Java OpenGL (JOGL) library is used to use OpenGL graphic API and its functions.

2.2 Characteristic of Java
Java is a known programming language today and it has the following properties:

- **Simplicity:** Java is easy to learn for people who have experiences with programming languages. Also, written code is simple and understandable because there are no many stars (*) or and (&) signs.
- **Platform independent:** Written codes run on every platform. There is no exerting for transferring system. With this way, even if people have different operating systems, they can play games with the same application, at the same time.
- **Object-orientated programming language:** Objects are created in Java to make new structures. This compels software developers to think object-based and it increases performance.
- **Distributed programming:** Distributed programming comes with language in Java. When it is developed, distribution is calculated for the internet.
- **Power:** Java is not just writing little applications. Big tools can be designed with Java.
• **Multithreaded:** Multithreaded comes with Java programming language. There is no requirement to use a separate library. Thus, little application parts can be run easily at the background, at the same time in application.

• **Stability:** When errors occurred at the working time in Java, they are corrected at the compiling time.

• **Reliability:** Java applications can run to meet requirements with many different safety levels. With this option, Java applets run securely on the internet [14].

2.3 3D Graphic APIs  
This part explains 3D graphic APIs. For this, firstly OpenGL and JOGL are explained with their architectures. In the ongoing process, it explains the interaction between OpenGL and JOGL. Later on, Autodesk 3D Studio Max and file formats are explained for 3D objects that are used for model loading.

2.3.1 OpenGL API  
API (Application Program Interface) is a method to use services that are given by operating system. It can be thought like the calling function from a library. When the request is made with API, application goes on to work.

OpenGL (Open Graphic Library) is a free interface API for applying graphics or developing graphics. OpenGL is used to design two-dimensional or three-dimensional graphics for screen with using advanced hardware support, because OpenGL allows managing of graphic hardware [15]. Also, according to web.cs.wpi.edu; OpenGL API has some features:

- Geometric and raster primitives
- RGBA or colour index mode
- Display list or immediate mode
- Viewing and modelling transformations
- Lighting and Shading
- Hidden surface removal (Depth Buffer)
- Alpha Blending (Translucency)
- Anti-aliasing
- Texture Mapping
- Atmospheric Effects (Fog, Smoke, Haze)
- Feedback and Selection
- Stencil Planes
- Accumulation Buffer
- Depth Cueing
- Wireframes
- Motion Blur [18].

OpenGL has these features and it provides them. For instance; the surface can be covered with texture mapping option. Thus, object is mapped. This is just one of the OpenGL options. These features show OpenGL’s benefits. Thus, OpenGL can always be used for many goals like using of Java JOGL graphic library on this API. Also, OpenGL has some characteristics. They are;
• **Library or API:** It is a library or API to design two-dimensional or three-dimensional graphics and this API allows the using graphic hardware for software developers.

• **Platform independent:** When an application includes OpenGL, firstly used library must be on operating system for running. This library is called as run-time library.

• **Window manager independent:** Written applications, using OpenGL run on Win32, MacOS and X-Window window manager without any problem.

• **Used by many programming languages:** Ada, C, C++, C#, FORTRAN, Python, Perl and Java can get benefits by using OpenGL library.

• **Portable:** This library is operating system and platform independent. With OpenGL, programming is made without graphic card model or processor architecture. Also, programming is made autonomously without operating system. The easy using and being portable make OpenGL popular [18].

2.3.2 OpenGL Pipeline

Figure 2.1 describes OpenGL API calls with various steps. When OpenGL API makes a call; it is placed in command buffer. Then it appears in the line for waiting its turn. When it has got its turn, if there is transformation or lighting, this is done. If there are no calculations, it goes directly for rasterization. The rasterization is done here. Finally, it comes to frame buffer which is memory for the display device [3].

![OpenGL Pipeline Diagram](image)

Figure 2.1: The OpenGL Pipeline [25].

2.3.3 OpenGL Libraries (GLU, GLUT)

GLU provides high-level rendering and modelling for OpenGL. This can be from simple wrapper OpenGL functions to advanced rendering techniques. These features can be 2d image scaling, rendering 3D objects including spheres, cylinders, disks, automatic mipmap generation from a single image [6]. However, some processes can not be made with OpenGL library like taking data from user or plotting window. These are related to platform. This means it against OpenGL’s nature since, OpenGL is platform independent. Fortunately, there is OpenGL Utility Toolkit (GLUT) and it saves software developers to being platform independent. GLUT is an imported library for operating systems. Its aim is creating OpenGL programs windows and the giving data from mouse or keyboard. The using GLUT library, data I/O (Input/output) processes are made without operating system. Figure 2.2 shows OpenGL software organization for OpenGL libraries which are on system [7].
When software developers used these libraries, they provide relation between application and software/hardware. Thus, functional OpenGL library features are used.

### 2.3.4 OpenGL as a State Machine

The state machine controls how to display objects. Object properties can be set up on OpenGL and can be queried by OpenGL. When object is read by OpenGL, it is shown with its GL (graphic library) capabilities. Nevertheless, these properties have stayed until next changes and these properties can be drawing modes, material properties, colours, lights, projection and viewing matrix. Therefore, the state machine decides about objects which are coded with GL capabilities and objects are shown with those properties [3].

### 2.4 JOGL

JOGL is a Java interface to access OpenGL libraries for Java virtual machine. By this way, OpenGL libraries and methods can be used and hardware-accelerated two-dimensional or three-dimensional graphics are designed [8]. The primary difference between 2d and 3D graphics is the third coordinate. It changes with regard to camera angle or viewer position.

Java is the one of the most popular programming languages. In the past, there have been many attempts to make connection between the OpenGL and Java. Thus, JOGL had been brought by Sun (the creators of Java) and SGI (Silicon Graphic Library). JOGL has been developed to use OpenGL libraries and functions. With this way, software developers can make great games or applications which are Java-based [5].

JOGL is easy to use because it provides Java interface to access OpenGL API and its functions. In this manner, Java interface conversions C functions easily. These functions are displayed when application is rendered. Thus, Java-based screens are shown.

### 2.4.1 Interaction with JOGL and OpenGL

JOGL is a programming interface. This programming interface is used to develop graphical applications which are Java-based and supported by OpenGL. The games, interactive education applications, graphic modelling editors are developed by using Java OpenGL (JOGL). But, OpenGL procedures are written in C. At that time, these procedures are called via Java native interface (JNI). This native interface is at the background when JOGL works. Thus, these procedures can work. The JOGL working...
principle has been placed on this framework. So, the used platform should support OpenGL’s work.

There are also some programming interfaces like Java3D, GL4Java to run OpenGL through Java. Contrary to these programming interfaces, JOGL calls OpenGL’s calls as procedural because, there are some specific methods in classes and JOGL allows these calls as procedural. This is JOGL’s approach. Ultimately, this approach contributes to faster graphic visualizations. The necessary specific methods are so important to Java application which uses OpenGL. These methods are absolutely necessary for using this API. They are,

- **Init()**: When application is started, the necessary steps are made here for graphic operations. The window size and type of transformation are made here.
- **Reshape()**: When screen is resized, this method is called. New height and width are given within this method.
- **Display()**: This method is main point of the application. Because the minimum change or when canvas is re-drawn every time, this method is called [9].

In brief, JOGL uses most of the OpenGL functions with its Java interface. It is raw OpenGL binding and far less restrictive than some other Java APIs. It means it uses direct OpenGL calls and some effects can be created like explained in OpenGL specifications. Thus, JOGL exposes many OpenGL extensions more than other Java APIs. Also, modern hardware can be utilized in Java Applications. Here, JOGL is a tool to use OpenGL and OpenGL’s specifications. So, the displaying interactive gallery is made on OpenGL but, commands are given by Java programming language (JOGL). It means when JOGL is used, it is using most of the OpenGL specifications. So, if OpenGL specifications are known well, many interactive gallery or advanced galleries can be created by using JOGL.

### 2.5 Autodesk 3D Studio Max (v. 2011)

Autodesk 3D Studio Max is one of the most popular 3D modelling and animation software. This program is seen as future modelling software for designers with its modelling skills, plug-in support, and easy usage. Also, Autodesk 3D Studio Max has common usage between the 3D modelling software. So, it provides many benefits for designers for modelling three-dimensional models [19].

Actually, Autodesk 3D Studio Max has its own large place in modern-day. It is used for television commercials, designing architectural constructions, analyzing scientific findings, cinema and special effects, space simulation, cartoon animations, medical and commercial areas, and industrial designing. Also, Autodesk 3D Studio Max is seen in computer technology. It makes animator or interactive solutions in computer technology. Moreover this, it provides 3D graphics or models for applications. Objects are designed on Autodesk 3D Studio Max and they run on Java OpenGL (JOGL) as interactive. Thus, 3D objects are moved with hands like games but, the moving is given with coding [12].

### 2.5.1 Autodesk 3D Studio Max (v. 2011) Interface

Autodesk 3D Studio Max has its own menus, toolbars, viewports, command panel, sub-objects, modifier list, material editor and many tools. 3D models are designed with different viewports, dimensions and with many features. Also, it has lighting, mapping,
animating and many more options. Shortly, Autodesk 3D Studio Max has many features for designing 3D models. The next figure shows its interface.

![Image of Autodesk 3D Studio Max Interface](image)

Figure 2.3: Autodesk 3D Studio Max Interface [27].

As is seen from figure 2.3, Autodesk 3D Studio Max has many options to design 3D models. Here, 3D models are designed with standard primitives and with other objects because Autodesk 3D Studio Max has some modelling types and every type have their own sub-tools. Moreover, Autodesk 3D Studio Max collects functions on its interface to access software’s features. In this way, this makes Autodesk 3D Studio Max accessible for its features and functions. It gets easy for designer when designer start to design because Autodesk 3D Studio Max interface gives an intuitive command for beginner users but, it gives direct access for advanced users. In this manner, software’s interface provides faster accessing for designers instead of solution seeking [1].

2.6 Exporting with Autodesk 3D Studio Max

When 3D modelling is finished, the model is exported to use its geometric features on development software by using OpenGL and JOGL interface. To the accessing Export Command by choosing Application Button > Export > Export is used. Thus, the exporting options come to screen. Max can export different file formats. They are:

- Autodesk (FBX)
- 3D Studio (3DS)
- Adobe Illustrator (AI)
- ASCII Scene Export (ASE)
- AutoCAD (DWG, DXF)
- Collada (DAE)
- Initial Graphics Exchange Standard (IGS)
- Flight Studio OpenFlight (FLT)
- JSR-184 (M3G)
- Lightscape Material, Blocks, Parameters, Layers, Preparations and Views (ATR, BLK, DF, LAY, LP, VW)
- Motion Analysis (HTR)
- Publish to (DWF)
- OBJ Material and Object (OBJ)
- ACIS SAT (SAT)
- StereoLithography (STL)
- VRML97 (WRL)

Autodesk 3D Studio Max has different export options to use in other software. In the figure 2.4, the obj file format has been chosen to export 3D model. The geometric features, material features, texture coordinates and other options are used to make interactive models.

![OBJ Export Options](image)

Figure 2.4: The OBJ file export options

As is seen from figure 2.4, there are geometry, material, optimize and other options. For instance geometry data, face type, scale option are used to make 3D object bigger or to make 3D object smaller. These options can be used for object before loading. Also, there are texture coordinates, vertexes, faces, normals, material options on the exporting option. These features are sent after clicking Export Button and sent information is used in coding. Therefore, 3D object’s features are chosen here. Then, this object is exported to use its information on development part. Also, .obj file format is explained in the next section.

### 2.7 OBJ File Format

One of the file formats is OBJ file format and it is supported by many 3D modelling software. It stores model’s information to be used. Basically, according to fileformat.info; OBJ is a file format that is to introduce geometric files. Wavefront OBJ (object) files are used by Wavefront's Advanced Visualizer application to store geometric objects composed of lines, polygons, and free-form curves and surfaces. Wavefront is best known for its high-end computer graphics tools, including modelling,
animation, and image compositing tools. These software run on powerful workstations such as those made by Silicon Graphics, Inc. OpenGL is one of the products that is developed by SG and this means, obj file formats run on OpenGL with Java interface. Therefore, the .obj file format is a bridge to run 3D models on OpenGL [20].

### 2.7.1 OBJ File Format Organization

OBJ file format has some keywords in its documentation. This format uses these keywords to understand which keyword indicates model’s specifications or where it starts and finishes like vertexes, texture coordinates and normals. According to cs.clemson.edu; OBJ files do not require any sort of header, although it is common to begin the file with a comment line of some kind. Comment lines begin with a hash mark (#). Blank space and blank lines can be freely added to the file to aid in formatting and readability. Each non-blank line begins with a keyword and may be followed on the same line with the data for that keyword. Lines are read and processed until the end of the file. Lines can be logically joined with the line continuation character (\) at the end of a line [21]. In the following keywords, keywords are explained with their data types and every keyword is explained with its brief descriptions.

**Vertex data:**

- **v** Geometric vertices: `v x y z`
- **vt** Texture vertices: `vt u v`
- **vn** Vertex normals: `vn dx dy dz`

**Elements:**

- **p** Point: `p v1`
- **l** Line: `l v1 v2 … vn`
- **f** Face: `f v1 v2 … vn`
- **f** Face with texture coordinates: `f v1/t1 v2/t2 … vn/tn`
- **f** Face with texture normals: `f v1/n1 v2/n2 … vn/nn`
- **f** Face with txt and norms: `f v1/t1/n1 v2/t2/n2 … vn/tn/nn`

**Grouping:**

- **g** Group name: `g group name`

**Display/render attributes**

- **usemtl** Material name: `usemtl materialname`
- **mtllib** Material library: `mtllib materiallibname.mtl`

OBJ File format is based on types of definition. For instance, vertexes are signified with a “v” and followed x, y and z coordinates of the point in space. Texture vertexes are signified with a “vt” and this is used for vertex texture. Vertex normals are signified with a “vn” and it defines a vertex normal. “f” character is used for face and it indicates which vertex, vertex texture, vertex normal should be joined to form a face [21].
2.7.2 MTL File Format

The MTL file format is used to show features of material definitions. Each new material definition starts with a newmtl statement and it specifies material’s name. Material properties are defined using material statements. For instance, for a 3D wall model;

```
newmtl wall Default
  Ns 10.0000
  Ni 1.5000
  d 1.0000
  Tr 0.0000
  Tf 1.0000 1.0000 1.0000
  illum 2
  Ka 0.5882 0.5882 0.5882
  Kd 0.5882 0.5882 0.5882
  Ks 0.0000 0.0000 0.0000
  map_Ka wall.jpg
```

- **Ns**: The shininess of the material is set with the s float value.
- **Ni**: The optical specifies for the surface. This is also known as index of refraction.
- **d alpha**: The alpha value specifies the transparency of the material.
- **Tr alpha**: The alpha value specifies the transparency of the material.
- **Tf r g b**: The Tf statement specifies the transmission filter using RGB values.
- **illum**: The illum statement sets the illumination mode for the material.
- **Ka r g b**: The three floats (r, g and b) define the ambient RGB colour of material.
- **Kd r g b**: The three floats specify the diffuse RGB colour of the material.
- **Ks r g b**: The r, g and b floats represent the specular colour of the material.
- **Map_ka file format**: The named file contains a texture for the material [2].

According to cs.clemson.edu; material library files contain one or more material definitions which include colour, texture, and reflection map of individual materials. These are applied for surfaces and vertices of objects. The information is stored arrays when application is rendered. It means, application calls these properties and they go to arrays. Each property (colour, map) is used when it is called [13].
3. TECHNOLOGY AND METHODOLOGY

Java interface (JOGL) is used to access OpenGL functions because objects are displayed on this API. When JOGL is examined, JOGL files are found as JAR extension files. So, there is a development software requirement which accepts JAR files. This development program also will support Java to use this programming language. So, Eclipse has these options when its architecture examined. It has extendable architecture and it has tools to use JAR files and to support Java applications. On this development program, JOGL files can be implemented and Java programming language codes can be written to use OpenGL API. This chapter basically explain this development software and it explains its specifications.

3.1 Eclipse Development Software

Eclipse contains necessary support to build Java applications. One of the specifications is about JAR files because JOGL files are found as extensions and they are JAR files. Eclipse’s extendable architecture allows using these JAR files. Thus, JOGL can be used on Eclipse and these Java files (JOGL) allow to draw graphics or these files can be used for model loading.

Eclipse is an open source embedded software development platform (IDE). The main point for Eclipse is Java and the related technologies however; it can be used for the different programming languages like C/C++, Rubby, Python, Php and JavaScript with its flexible plug-ins-based structure.

Eclipse was released as an open source in 2001. Then the Eclipse Foundation was established in 2004 and it was built on Eclipse 3.0 OSGI service platform. Figure 3.1 shows its architecture;

![Eclipse Architecture](image)

**Figure 3.1: Eclipse architecture [26].**

Eclipse has been developed with extendable architecture. In this architecture, the working system is provided with extensions. Eclipse extension mechanism allows adding new extension onto existing extensions. This makes Eclipse different to usual plug-in-based architectures. Basically, the written extensions always are not core codes. They can extend the features of existing extensions. There are some main features for Eclipse;
- **Workbench**: To the extending Eclipse interface, it includes extension points like viewing tools or toolbar menu.
- **Workspace**: This file includes extension points that make interaction with projects and sources.

Eclipse provides two main properties for plug-in development. They are Java development tools (JDT) and plug-in developer environment (PDE). JDT is Java development platform with all properties. PDE adds special tools that are for the developing plug-ins and extensions like create, develop, test and debug. Also, PDE provides OSGI tools which are for making component programming.

- **Standard Widget Toolkit (SWT)**: A portable widget toolkit
- **JFace**: It brings visual classes to SWT like file buffers, text handling and text editors.
- **Equinox**: It is a run-time module. An application is implemented as bundles (jar) by software developers. Software developers use common services and platform when they made this.
- **Plug-ins**: The structured code/data is set up to help system. They are added on system with extension points and they get mission when they are added on system or they help to another plug-in. Some plug-ins provides apparent properties and some plug-ins is just used as library [10].

On the thesis, JOGL files are moved onto Eclipse and these files are run on Eclipse. In one way, Eclipse provides platform to run all steps on its. So, Eclipse has been preferred to develope interactive gallery process because of its Java, JOGL and platform support.
4. METHODS
Interactive gallery is based on moving objects theory for thesis and this application can be developed by using many development tools. It can be made by using a programming language however; there are easier ways for developing. One of them is model loading method and it is made with 3D modelling, using graphic library and coding. It can be seen hard but, each development step provides better and realistic outputs. Moreover this, less coding is made in this method instead of writing thousands of codes. Thus, an interactive gallery can be obtained [2]. This chapter tries to explain these steps with details which will be used in implementation part.

4.1 Modelling Part
Autodesk 3D Studio Max provides three-dimensional graphic design. On this part, three-dimensional building (the room) is designed with its furniture. Lastly, other objects are placed which will be made interactive. Each three-dimensional structure is named as an object model. The object models can be exported after designing yet, exporting is made after identifying workspace which comes with Eclipse. Thus, models can be called easily from that place.

4.2 Installation of Java and Java Runtime Environment Part
On this part, Java is updated then Java runtime environment is installed because Java runtime environment supports Java applications. It means Java and Java runtime environment provide the necessary supports to run Java applications.

4.3 Installation of Development Software and Implementing JOGL Part
To the creating interactive gallery and interactive objects, development software is needed. So, the Eclipse is downloaded. Thus, obj files (the exported files) are re-created again by using Java OpenGL (JOGL). But, firstly the JOGL .dll files and .jar files are implemented on this development software to use Java programming language.

This step is one of the most important parts because the used JOGL classes are implemented. With this way, written Java codes are supported. Also, the bridge is built to use graphic card functions with other explanation. So, JOGL is implemented on development software.

4.4 Model Loading Part
Model loading codes are implemented on development software in this step. Thus, exported objects are called from workspace and they are re-drawn by using Java OpenGL (JOGL).

4.5 Key Implementation Part
The keyboard keys are integrated on the chosen models for re-placing. Some commands are used for this. The direction and distance are identified. The used directions are identified as x and y to understand this process. The distance is given objects. Thus, when keyboard key is clicked, objects are moved or re-placed.

End of these steps, interactive gallery is obtained with its interactive objects. Also, schema 4.1 tries to explain this process.
Schema 4.1 explains basically interactive gallery process. It is started with modelling. 3D objects are designed in this step. After that, Java and Java runtime environment are installed to run Java projects. Then, Eclipse is downloaded and Java project is created. At this time, 3D models are exported with .OBJ file formats to be loaded on Eclipse by using Java OpenGL (JOGL). In model loading part, 3D models are loaded by using model loading method and chosen objects are made interactive with codes. At the end of the all processes, project is rendered and interactive gallery is obtained.
5. IMPLEMENTATIONS

An interactive gallery can be high-level and illusion of actual one. There are possible methods for developing this. Virtual Reality Modelling Language (VRML) is one of them and it provides virtual reality for interactive galleries. Moreover this, it is supported by Autodesk 3D Studio Max. However, it is really hard to code and requires high programming skills. Also, it is based on virtual reality, database and web technology like VirTools or Vizard [4]. However, interactive gallery is related to its interactive part and it is based on model loading method. So, interactive gallery can be starting point to develop advanced galleries.

In this chapter, interactive gallery processes have been explained with more details. These processes are started to be made with designing, exporting, installation of necessary Java environments. It goes on with Eclipse and JOGL implementation. Later on, models are re-drawn with implementation of codes. At the end of the process, the interactivity is given for models. Thus, the main goal is obtained.

5.1 3D Room Modelling with Autodesk 3D Studio Max (v. 2011)

As is mentioned on chapter 2.5, Autodesk 3D Studio Max is common 3D modelling software around the world. With it, many 3D applications can be developed. This is seen in daily life like television commercials, cinema, computer technology, education, industry, simulations. 3D models are designed firstly to make a gallery because these models are used on development platform. It means the output is these models. When the main goal (the interactive gallery) is scaled up, a 3D room is designed firstly with doors and windows. Then furniture is placed in this room. Lastly, last 3D models are placed in room which will be made interactive. However, 3D models are designed separately. This means, every 3D model wants separate design type and the mapping type. To the creating good graphic solutions and correct results, every 3D model is designed separately. Thus, at the end of the modelling, three-dimensional designs get ready to go to development platform.

The following objects are designed by using Autodesk 3D Studio Max and these objects are named as object models for supplying.

- ObjModel for the ceiling,
- ObjModel1 for the floor,
- ObjModel2 for the back wall,
- ObjModel3 for the left wall,
- ObjModel4 for the right wall,
- ObjModel5 for the painting1,
- ObjModel6 for the television table,
- ObjModel7 for the left window,
- ObjModel8 for the carpet,
- ObjModel9 for the flower,
- ObjModel10 for the middle wall,
- ObjModel11 for the painting2,
- ObjModel12 for the painting3,
- ObjModel13 for the painting4,
- ObjModel14 for the right window,
- ObjModel15 for the door,
- ObjModel16 for the fan,
- ObjModel17 for the television.

There are many tools on Autodesk 3D Studio Max’s interface. Some important elements are indicated which are used for designing. According to Murdock, these are:

- **Title Bar and Menus:** This is the default source for most commands, but also one of the most time-consuming interface methods. The title bar and menus are found along the top edge of the Max window.
  - **Clone (Ctrl+V):** There is couple of ways for cloning. One method is to use the Edit-Clone (Ctrl+V) menu command, and other method is to transform an object while holding down the Shift key.

- **Toolbars:** Max includes several toolbars of icon buttons that provide single-click access to features.
  - These toolbars can float independently or can be docked for an interface edge.
    - The main toolbar is the only toolbar that is visible by default.
  - **Select and Move (W):** Selects an object and allows positional translations.
  - **Select and Rotate (E):** Selects an object and allows rotational transforms.
  - **Select and Uniform Scale, Select and Non-Uniform Scale, Select and Squash (R to cycle):** Selects an object and allows scaling transforms using different methods.
  - **Align (Alt+A), Quick Align, Normal Align (Alt+N), Place Highlight (Ctrl+H), Align to Camera, Align to view:** Opens the alignment dialog box for positioning objects, allows objects to be aligned by their normals, determines the location of highlights, and aligns objects to camera or view.

- **Viewports:** Four separate views into the scene show the Top, Front, Left, and Perspective viewpoints.

- **Command Panel:** The located major control panel to the right of the four viewports, it has six tabbed icons at its top that designer can click to open the various panels. Each panel includes rollouts containing parameters and settings. These rollouts change, depending on the object and tab that is selected.
  - **Boolean:** Created by performing Boolean operations on two or more overlapping objects. The operations include Union, Subtraction, Intersection, and Cut.
  - **Loft:** Sweeps a cross-section shape along a spline path.
  - **Modifier List:** 3D studio max has modifier list to apply on objects. An object can have several modifiers applied to it. Modifiers can be applied using the Modifiers menu or by selecting the modifier from the Modifier List drop-down list located at the top of the Modify panel directly under the object name. Selecting a modifier in the Modifiers menu or from the Modifier List applies the modifier to the current selected object. Modifiers can be applied to multiple objects if several objects are selected.
Doors: 3D Studio Max has ready objects like windows and doors. Doors are under the command panel with standard primitives. This section allows using ready door models.

Editable Poly: The Edit Poly modifier lets designers work with primitive objects using the operators found in the Editable Poly rollouts.

Polygon Mode: The polygon sub object mode. It provides to use sub-buttons (options)

Extrude: It is under the polygon mode. The Extrude button copies and moves the selected sub object perpendicular a given distance and connects the new copy with the original one.

Bevel: The Bevel button extrudes the Polygon sub object selection and then lets designers bevel the edges. To use this feature, select a polygon, click the Bevel button, drag up or down in a viewport to the Extrusion depth, and release the button. Drag again to specify the Bevel amount. The Bevel amount determines the relative size of the extruded face.

Slice Plane: The Slice Plane button lets designers split the poly object along a plane [1].

Materials are used to dress, colour, and paint objects. Just as materials in real life can be described as scaly, soft, smooth, opaque, or blue, materials applied to 3D objects can mimic properties such as colour, texture, transparency, shininess. Figure 5.1 shows this.

Figure 5.1 Material Editor [28]

The material editor has many options. Therefore, some important and used features are indicated. They are;

- **Diffuse**: The surface colour of the object in normal, full, white light. The normal colour of an object is typically defined by its diffuse colour.
- **Opacity**: Opacity is the amount that an object refuses to allow light to pass through it. It is the opposite of transparency and is typically measured as a
percentage. An object with 0 percent opacity is completely transparent, and an object with 100 percent opacity doesn’t let any light through [1].

With these properties, 3D models can be designed easily. Of course, there are many tools and options to get benefit from Autodesk 3D Studio Max yet, these properties also allow designing 3D objects [1].

5.1.1 Walls

Autodesk 3D Studio Max has standard primitives to design new objects like box, cylinder, teapot, sphere, tube, torus, pyramid, plane, cone, geosphere. The new objects are designed with using these geometry objects or using shapes.

![Figure 5.2: The Floor (ObjModel1)](image)

As is seen from figure 5.2, the model is designed for floor but, first of all, the sequences of actions are different for modelling. Therefore, these steps can be itemized. They are;

- The viewport is chosen for modelling.
- The primitive is chosen to make floor. Here, the box primitive is chosen and dimension parameters are designated. Dimensions for the floor; 140x210x3 cm. 140 cm for the length, 210 cm for the width, 3 cm for the height.
- The coordinates are selected. For x, y and z, coordinates are 0, 0, 0. This is more specifically for the loading models.
- The mapping picture is chosen. Autodesk 3D Studio Max has material editor to the designing these 3D models. In the material editor, the mapping type, ambient, diffuse and more features are specified. Here, the diffuse is important for mapping. Diffuse is selected in the material editor and the mapping type is specified. Here, bitmap is selected and the image is found which is used. Lastly, it is applied on the object. The important thing is image dimension. 512x512 dimensions are specified for the images because object can be seen corrupted when it is rendered or when it is loaded with development software.
Lastly, the object is saved as max file but, every object is saved separately on this project and these max files are specified with object model number. After modelling floor, left wall (ObjModel3) is designed. Figure 5.3 shows this.

![Figure 5.3: The Left Wall (ObjModel3)](image)

As is seen from figure 5.3, there are a few differences between the floor and the left wall. The left wall is generated with the using floor. Hence, the main differences are itemized;

- The ObjModel3 is copied by using ObjModel1.
- The dimension parameters are changed (140x140x3cm) for ObjModel3.
- The ObjModel3 is rotated with y direction.
- The ObjModel3 is aligned with the ObjModel1.
- The new map is applied for ObjModel3.

Lastly, the new object is saved as max file. Before saving, these objects are rendered on Autodesk 3D Studio Max. So, the outputs come from rendering results on Autodesk 3D Studio Max. The right wall is generated with the using left wall. On the following figure, it is shown.
As is seen from figure 5.4, the third wall (right wall) is created with the using ObjModel3. It is copied, aligned and saved as max file. But, the right side disappears when it is rendered on Autodesk 3D Studio Max. Autodesk 3D Studio Max provides lighting but, this is made on Java OpenGL (JOGL) with lights.

Also, Autodesk 3D Studio Max provides good option for these objects. When object is copied from other object, the copied object takes other object’s properties. This is good option because there are no material specifications for new object like mapping. On the following figure, the next wall (back wall) is seen after some steps.

There is a demand of some steps to create the wall according to figure 5.5. These steps almost same with previous steps. Firstly, wall is selected. Then it is copied. New
object (back wall) is rotated and aligned with other walls. Lastly, the ObjModel2 is saved as max file.

There are some particular steps for modelling last wall (the ceiling (ObjModel)). These steps are shown on the following figure.

![Image](image.png)

Figure 5.6: The Ceiling (ObjModel).

As is seen from Figure 5.6, the room is almost completed. This output is for Autodesk 3D Studio Max. But, the view is zoomed on Java OpenGL (JOGL). Unlike the previous ObjModel2, the map is changed for the ceiling. This is provided by material editor on Autodesk 3D Studio Max. Lastly, the ObjModel is saved as max file.

5.1.2 Windows

Autodesk 3D Studio Max has ready objects. Some of them are windows. Windows can be used as ready objects in design however; the ready window can not be as it is required. Designer can want to use different designing types on model for better designs. So, designer can do it from scratch instead of re-modelling of ready objects. Thus, designer knows what property can be used on object and where. By this way, new and better window models can be created end of the every design. Therefore, interactive gallery windows are made by using some boxes. Figure 5.7 shows these new changes.
Some options are used for generating a window, in Autodesk 3D Studio Max. One of the options is made with three boxes. To the designing window, the process steps are itemized. These steps;

- Three boxes are drawn by using the standard primitives.
- The dimension parameters are identified as 50x40x3 cm for these boxes.
- The first box is put on wall. The Boolean object type is selected under the compound objects when wall is selected.
- With the Pick Operand B option, the first box is selected and the first box picks the part of wall. Then the empty place is seen on the wall to put frame and glass and first box disappears.
- The second box is used for making a frame. So, the second object is converted to editable poly.
- Under the section, there are several sub options to design this box. One of them is Polygon Mode.
- There is Slice Plane under this section. Some parts are sliced and deleted which will be appeared.
- Lastly, the wood map is applied on this box by using material editor.
- The third box is for making a glass. So, it is scaled to put in frame.
- In the material editor, the Maps option is selected and the Opacity map is identified.
- Here, the Noise mapping type is used for glass.
- The colours are specified as black.
- The map is applied and it is put in frame. When it is rendered, the ObjModel7 is seen on the screen.
- Lastly, the window is saved as max file.

For the other two windows, same process steps can be made or window is just copied. When it is copied, the Boolean option is used to put window on wall. Then windows are
placed on walls. Lastly, the new windows (ObjModel10 and ObjModel14) are saved. The Figure 5.8 shows the final version for the windows.

![The Figure 5.8: The Middle and the Right Window (ObjModel7, 8)](image)

As is seen from Figure 5.8, the same steps are used for the other window. The Clone option brings convenience. Thus, the faster design is made with this option.

### 5.1.3 Door

Autodesk 3D Studio Max has ready objects like windows or doors. These ready objects also can be used for faster design. Also, windows and door are designed with other Autodesk 3D Studio Max options. Here, the new door is designed through ready door. The next figure shows the new object model (ObjModel15).

![Figure 5.9: The Door (ObjModel15)](image)
To the designing door, Autodesk 3D Studio Max door is used. So, this door is developed with some Polygon Mode sub options. These are Extrude, Bevel and Slice Plane. But firstly, door is converted to editable poly. Then the place is sliced with Slice Plane. With the Extrude and Bevel, the door handle is designed. Lastly, the map is applied on this door. Thus, when it is rendered, it is seen like on Figure 5.9.

5.1.4 Carpet
When objects are designed on Autodesk 3D Studio Max, aligning can be needed for these objects. At that time, the Align option is used. Thus, two objects are aligned. The following figure shows this.

![Figure 5.10: The Carpet (ObjModel8)](image)

In this process, new box is used and dimension parameters are specified (115x165x3 cm). The new map is applied on this box. Lastly, it is aligned with floor. When it is rendered, it gives output that is shown on Figure 5.10.

5.1.5 Television Table
The Extrude, the Slice Place and the Bevel options provide new dimensions for objects when they are used together. With an example, the table is designed with these options and the table gets new dimensions. It is like door handle, table’s foot, television table, book shelf. These options are under the Polygon Mode. This means, the object is converted to editable poly for using them. The next figure (5.11) typifies this.
The table is seen on the Figure 5.11 which is designed by using extrude and bevel. For this, the box is converted to editable poly. In the polygon mode, the box is sliced with the Slice plane to know which parts will take dimensions. Later on, the table’s foot is made with the bevel. Lastly, the new distances are gone with extrude. Thus, the television table is generated with just three options.

### 5.1.6 Ready Models

There are many documents on the internet about Autodesk 3D Studio Max and Autodesk 3D Studio Max provides many ready models. In addition to this, many ready models also are placed on the internet by designers [22]. The next figure shows these models.
The ready models allow faster designing for works. To the getting these ready objects on Autodesk 3D Studio Max, the Merge option is used which is under the Import options. This option allows propping design. Another scene is integrated for the current scene. Using this menu command opens a file dialog box that is exactly like the Open File dialog box. Then another scene is selected with clicking the Open button [1]. The next figure shows Merge dialog box.

![Merge Dialog Box](image)

Figure 5.13: The Merge Dialog Box.

After merging, new scenes are integrated from another scene to current scene. Thus, new scene is placed on the current scene. This allows easier designing for 3D modelling. Also, all of the designs can be made with this way.

### 5.1.7 Paintings

Autodesk 3D Studio Max allows using shapes like Line, Rectangle, Circle, Ellipse, Arc, Donut, Ngon, Star, Text, Helix, and Section. To the making paintings, there are several ways. One of them is made with two rectangles and a line. At this point, the Extrude option is used for one of the rectangles. Because of it is used to be background. The other rectangle is used for the getting shape with line. For this, the Loft options is used which is under the Compound Objects. Lastly, the map is applied on backgrounds and output is shown on next figure (5.14).
When scene is rendered, Figure 5.14 is shown on Autodesk 3D Studio Max rendering screen. Consequently, eighteen objects are ready to go to on Java OpenGLE (JOGL). Here, objects are loaded by using codes. But, before coding, Java, Java’s environment, development software, JOGL are implemented. Then, these objects are exported and sent into Eclipse file which is called as workspace (See Appendixes A.1, A.2, and A.3).

5.2 Model Loading
Till here, these steps have been made:
- 3D object models have been designed.
- Java and JRE 7 have been implemented.
- Eclipse has been downloaded and JOGL has been implemented on Eclipse.
- Object models have been exported to...\workspace\Loadobj (the identified Java project name)

   These steps are a substructure for loading models. In this part, the necessary codes are written for loading. It means object models are called with their features, and re-drawn with Java OpenGLE (JOGL) on Eclipse development platform. But, first of all, the main class is specified that is OBJ file loader. Java application runs with main class and main class loads object models and their features like materials, shape statements (v, vt, vn, f). To the making clearer, class diagrams are shown. On the following schema is for OBJ Loader file.
As is seen from Schema 5.1, the necessary classes are classified for the loading. When the main class is run, other classes and other inner classes are entered into the process of the loading. These classes and inner classes are explained with understandable turn in the next sections.

5.2.1 Configurations
LoadObj is a project name that is described to collect all classes. All classes, inner classes and packages are collected under this Java project for the three-dimensional object models loading. As it was explained before, it was named when it was generated with JOGL libraries. Thus, it was made ready to write Java codes on Eclipse. In the ongoing process, it starts writing a main class (ModelImport) but, the main class is used for loading and calling to draw these objects. So, OBJModel class is started firstly to be called. It is seen also on schema 5.1.

5.2.2 Reading OBJ Files
OBJModel class reads the codes line by line and it processes the statements. Thus, OBJModel classes loads models and materials from OBJ and MTL files. The shape statements (v, vt, vn, f) are processed but, grouping transactions are ignored. The MTL file information is processed but, lighting is made with codes (see ModelImport Class).
Textures and colours are not blended because each model is read separately and each model’s information is read line by line.

The scaling can be made when model is exported or it can be scaled when object is loaded. The model has been stored in display list for rendering until the loader.draw() command in main class.

```java
private ArrayList<Tuple3> verts;
private ArrayList<Tuple3> normals;
private ArrayList<Tuple3> texCoords;
private boolean hasTCs3D = true;
private Faces faces;
private FaceMaterials faceMats;
private Materials materials;
private ModelDimensions modelDims;
private String modelNm;
private float maxSize;
private int modelDispList;
public OBJModel(String nm, float sz, GL2 gl, boolean showDetails) {
    modelNm = nm;
    maxSize = sz;
    initModelData(modelNm);
    loadModel(modelNm);
    if (showDetails)
        reportOnModel();
} // end of OBJModel()
private void initModelData(String modelNm) {
    verts = new ArrayList<Tuple3>();
    normals = new ArrayList<Tuple3>();
    texCoords = new ArrayList<Tuple3>();
    faces = new Faces(verts, normals, texCoords);
    faceMats = new FaceMaterials();
    modelDims = new ModelDimensions();
} // end of initModelData()
private void loadModel(String modelNm) {
    String fnm = modelNm + ".obj";
    try {
        FileInputStream fis_model = new FileInputStream(fnm);
        BufferedReader br_model = new BufferedReader(new InputStreamReader(fis_model));
        readModel(br_model);
        br_model.close();
    } catch (IOException e) {
        System.exit(1);
    } // end of loadModel()
private void readModel(BufferedReader br) {
    boolean isLoaded = true;
    int lineNum = 0;
    String line;
    boolean isFirstCoord = true;
    boolean isFirstTC = true;
    int numFaces = 0;
    try {
        while (((line = br.readLine()) != null) && isLoaded) {
            // read model data
        } while (isLoaded);
    } catch (IOException e) {
        System.exit(1);
    } // end of readModel()
```
Table 5.1 Reading OBJ files

Table 5.1 shows, the reading starts with definitions in OBJ file. The shape data is stored in ArrayList of Tuple3. Then OBJModel initiates other package objects like Faces, FaceMaterials, Materials, and ModelDimensions. After definitions, OBJModel class is identified and started. To the reading in OBJ file, a few steps start after identification.

- Model data is placed in variables (verts, normals, texCoords, face, faceMats, ModelDims). Due to searching model features.
- Model is loaded. For this, model is searched on class. If the modelNm.obj is equal to string fnm which is called in ModelImport class, the model is made ready for reading. Otherwise, system.exit() command is run. Actually, the loading is called in the main class but, the searching is made in OBJModel class.
Lastly, the model is read but, OBJ file is parsed for reading because each feature is read line by line and added to variables. Thus, new model is started to be created. Also, in materials and faceMats variables, MTL file and usemtl is used for taking model features.

It is shown in OBJModel class codes, the v, vt, and vn statements are used to add a vertex, a texture coordinate, and a normal Tuple3 object to the verts, texCoords, and normals ArrayLists.

```java
private boolean addVert(String line, boolean isFirstCoord)
{
    Tuple3 vert = readTuple3(line);

    if (vert != null) {
        verts.add(vert);

        if (isFirstCoord)
            modelDims.set(vert);
        else
            modelDims.update(vert);

        return true;
    }

    return false;
}
```

Table 5.2: Adding Vertex

When addVert() command is run addVert() adds a tuple to verts and it updates information which about model dimensions. Table 5.2 shows this with codes.

### 5.2.3 Storing Shape Statements

When OBJModel class is run, it reads codes line by line and it needs also Tuple3 class to get a tuple that is used to store a vertex, normal and texture coordinate.

```java
public class Tuple3
{
    private float x, y, z;

    public Tuple3(float xc, float yc, float zc)
    {
        x = xc;
        y = yc;
        z = zc;
    }

    public String toString()
    {
        return "( " + x + ", " + y + ", " + z + " )";
    }

    public void setX(float xc)
    {
        x = xc;
    }

    public float getX()
    {
        return x;
    }

    public void setY(float yc)
    {
        y = yc;
    }

    public float getY()
    {
        return y;
    }

    public void setZ(float zc)
    {
        z = zc;
    }

    public float getZ()
    {
        return z;
    }
}
```
public float getZ()
{
    return z;
}  // end of Tuple3 class

Table 5.3: Storing Shape Statements

On the table 5.3, a tuple of three elements is identified and it is moved to x, y and z variables to store a vertex, normal or texture coordinate. When OBJModel class needs these contents, it calls Tuple3 class and reads. To the reading tuples:

private Tuple3 readTuple3(String line)
{
    StringTokenizer tokens = new StringTokenizer(line, " ");
tokens.nextToken();
    try {
        float x = Float.parseFloat(tokens.nextToken());
        float y = Float.parseFloat(tokens.nextToken());
        float z = Float.parseFloat(tokens.nextToken());
        return new Tuple3(x,y,z);
    }
    catch (NumberFormatException e)
    {
        System.out.println(e.getMessage());
    }
    return null;
}  // end of readTuple3()
private boolean addTexCoord(String line, boolean isFirstTC)
{
    if (isFirstTC) {
        hasTCs3D = checkTC3D(line);
    }
    Tuple3 texCoord = readTCTuple(line);
    if (texCoord != null) {
        texCoords.add( texCoord );
        return true;
    }
    return false;
}  // end of addTexCoord()
private boolean checkTC3D(String line)
{
    String[] tokens = line.split("\\s+");
    return (tokens.length == 4);
}  // end of checkTC3D()
private Tuple3 readTCTuple(String line)
{
    StringTokenizer tokens = new StringTokenizer(line, " ");
tokens.nextToken();
    try {
        float x = Float.parseFloat(tokens.nextToken());
        float y = Float.parseFloat(tokens.nextToken());
        float z = DUMMY_Z_TC;
        if (hasTCs3D)
            z = Float.parseFloat(tokens.nextToken());
        return new Tuple3(x,y,z);
    }
    catch (NumberFormatException e)
    {
        System.out.println(e.getMessage());
    }
    return null;
}  // end of readTCTuple()
private boolean addNormal(String line)
{
    Tuple3 normCoord = readTuple3(line);
    if (normCoord != null) {
        normals.add( normCoord );
    }
```
return true;
}
return false;
}  // end of addNormal()
private void drawToList(GL2 gl)
{
  modelDispList = gl.glGenLists(1);
  gl.glNewList(modelDispList, GL2.GL_COMPILE);
  gl.glPushMatrix();
  String faceMat;
  for (int i = 0; i < faces.getNumFaces(); i++) {
    faceMat = faceMats.findMaterial(i);
    if (faceMat != null)
      materials.renderWithMaterial(faceMat, gl);
    faces.renderFace(i, gl);
  }
  materials.switchOffTex(gl);
  gl.glPopMatrix();
  gl.glEndList();
}

public void draw(GL gl)
{  ((GL2) gl).glCallList(modelDispList);  }
private void reportOnModel()
{
modelDims.reportDimensions();
if (materials != null)
  faceMats.showUsedMaterials();
}  // end of reportOnModel()
```

Table 5.4: Reading Tuples

On the table 5.4, steps can be itemized. These items;

- **AddTexCoord()**: The texture coordinate is added from the line “vt x y z” to the texCoords ArrayList. There can be two texture coordinates on the line which is specified by looking at first texture coordinate line.
- **CheckTC3D()**: If the line has four tokens which will be “vt” token and three texture coordinates, it checks in this case.
- **readTCTuple()**: The line starts with “vt” OBJ word and x, y or x,y and z floats for texture coordinates are separated by spaces. If there are two coordinates the z value is assigned as dummy value in DUMMY_Z_TC;
- **addNormal()**: The normal is added from line “vn x y z” to the normals ArrayList.
- **drawToList()**: The model is rendered to a display list. Thus, this process can be drawn quicker later. Subsequent calls to OBJModel.draw() will execute the list, greatly improving the drawing speed [2].
- **reportOnModel()**: It shows model dimensions before centering and scaling. Also, shows what materials have been used by faces.

### 5.2.4 Resizing Model Dimensions

This class computes the ‘edge’ coordinates for the model, along its three dimensions. The edge coordinates are used to compute the model’s;

- **Width, height, depth.**
- **Its large dimensions (width, height or depth).**
- **x, y, z center point.**
private float leftPt, rightPt;
private float topPt, bottomPt;
private float farPt, nearPt;
public ModelDimensions()
{
leftPt = 0.0f;  rightPt = 0.0f;
topPt = 0.0f;  bottomPt = 0.0f;
farPt = 0.0f;  nearPt = 0.0f;
}  // end of ModelDimensions()
public void set(Tuple3 vert)
{
rightPt = vert.getX();
leftPt = vert.getX();
topPt = vert.getY();
bottomPt = vert.getY();
nearPt = vert.getZ();
farPt = vert.getZ();
}  // end of set()
public void update(Tuple3 vert)
{
if (vert.getX() > rightPt)
rightPt = vert.getX();
if (vert.getX() < leftPt)
leftPt = vert.getX();
if (vert.getY() > topPt)
topPt = vert.getY();
if (vert.getY() < bottomPt)
bottomPt = vert.getY();
if (vert.getZ() > nearPt)
nearPt = vert.getZ();
if (vert.getZ() < farPt)
farPt = vert.getZ();
}  // end of update()
public float getWidth()
{  return (rightPt - leftPt);  }
public float getHeight()
{  return (topPt - bottomPt);  }
public float getDepth()
{  return (nearPt - farPt);  }
public float getLargest()
{
float height = getHeight();
float depth = getDepth();
float largest = getWidth();
if (height > largest)
largest = height;
if (depth > largest)
largest = depth;
return largest;
}  // end of getLargest()
}  // end of ModelDimensions()

Table 5.5: Resizing Model Dimensions

On the table 5.5:
- The model’s edge coordinates are identified for x, y and z axis with variables (in ModelDimensions()).
- The model’s edge coordinates which come with vertices are initialized and set (in set()).
- The edge coordinates are updated with the using vertices (in update).
The using with the edge coordinates, dimensions are calculated. These are width, height, depth and its large dimensions.

5.2.5 Reading and Rendering Faces
This class stores information about all the face statements. The each data is stored in three arrays of elements which are vertex, texture coordinate and normal indices. For instance, if the statement is:

\[ f \quad 5/4/2 \quad 6/1/2 \quad 7/2/2 \]

the vertex indices array will hold \{5,6,7\}, the texture coordinate indices array will hold \{4,1,2\}, the normal indices array will hold \{2,2,2\}.

All the faces data is held in the three ArrayList that is named faceVertIdxs, faceTexIdxs and faceNormIdxs and these three ArrayList holds all the indices and then these indices are processed with faces.addFaces().

```java
private ArrayList<int[]> facesVertIdxs;
promise ArrayList<int[]> facesTexIdxs;
promise ArrayList<int[]> facesNormIdxs;
promise ArrayList<Tuple3> verts;
promise ArrayList<Tuple3> normals;
promise ArrayList<Tuple3> texCoords;
public Faces(ArrayList<Tuple3> vs,
            ArrayList<Tuple3> ns,
            ArrayList<Tuple3> ts)
{
    verts = vs;
    normals = ns;
    texCoords = ts;
    facesVertIdxs = new ArrayList<int[]>();
    facesTexIdxs = new ArrayList<int[]>();
    facesNormIdxs = new ArrayList<int[]>();
}
public boolean addFace(String line)
{
    try {
        line = line.substring(2);
        StringTokenizer st = new StringTokenizer(line, " ");
        int numTokens = st.countTokens(); // number of v/vt/vn tokens
        int v[] = new int[numTokens];
        int vt[] = new int[numTokens];
        int vn[] = new int[numTokens];
        for (int i = 0; i < numTokens; i++) {
            String faceToken = addFaceVals(st.nextToken());
            StringTokenizer st2 = new StringTokenizer(faceToken, "/");
            int numSeps = st2.countTokens();
            v[i] = Integer.parseInt(st2.nextToken());
            vt[i] = (numSeps > 1) ? Integer.parseInt(st2.nextToken()) : 0;
            vn[i] = (numSeps > 2) ? Integer.parseInt(st2.nextToken()) : 0;
        }
        facesVertIdxs.add(v);
        facesTexIdxs.add(vt);
        facesNormIdxs.add(vn);
    }
    catch (NumberFormatException e) {
        System.out.println(e.getMessage());
        return false;
    }
}
```
return true;
}  // end of addFace()

private String addFaceVals(String faceStr)
{
    char[] chars = faceStr.toCharArray();
    StringBuffer sb = new StringBuffer();
    char prevCh = 'x';
    for (int k = 0; k < chars.length; k++) {
        if (chars[k] == '/' && prevCh == '/')
            sb.append('0');
        prevCh = chars[k];
        sb.append(prevCh);
    }
    return sb.toString();
}  // end of addFaceVals()

public void renderFace(int i, GL2 gl)
{
    if (i >= facesVertIdxs.size())
        return;
    int[] vertIdxs = (int[]) (facesVertIdxs.get(i));
    int polytype;
    if (vertIdxs.length == 3)
        polytype = GL2.GL_TRIANGLES;
    else if (vertIdxs.length == 4)
        polytype = GL2.GL_QUADS;
    else
        polytype = GL2.GL_POLYGON;
    ((GL2) gl).glBegin(polytype);
    int[] normIdxs = (int[]) (facesNormIdxs.get(i));
    int[] texIdxs = (int[]) (facesTexIdxs.get(i));
    Tuple3 vert, norm, texCoord;
    for (int f = 0; f < vertIdxs.length; f++) {
        if (normIdxs[f] != 0) {
            norm = (Tuple3) normals.get(normIdxs[f] - 1);
            ((GL2ES1) gl).glNormal3f(norm.getX(), norm.getY(), norm.getZ());
        }
        if (texIdxs[f] != 0) {
            texCoord = (Tuple3) texCoords.get(texIdxs[f] - 1);
            if (texCoord.getZ() == DUMMY_Z_TC)
                ((GL2) gl).glTexCoord2f(texCoord.getX(), texCoord.getY());
            else
                ((GL2) gl).glTexCoord3f(texCoord.getX(), texCoord.getY(), texCoord.getZ());
        }
        vert = (Tuple3) verts.get(vertIdxs[f] - 1);
        ((GL2) gl).glVertex3f(vert.getX(), vert.getY(), vert.getZ());
    }
    ((GL2) gl).glEnd();
}  // end of renderFace()

public int getNumFaces()
{
    return facesVertIdxs.size();  }
}

Table 5.6: Reading and Rendering Faces

Table 5.6 can be identified with some items:

- **addFace():** It gets face’s indices from line (f v/vt/vn…) with vt or vn index values. If there is no index value, it gets 0 values. Real indices start at 1. So, 0 values is a good choice for this situation.

- **addFaceVals():** A face token (v/vt/vn) can be missing. For instance, vn value can be missing. In this case, 0 values are added.
- **renderFace()**: Here, face is rendered by getting vertex, texture coordinate or normal for face i value. These indices are used to access real vertex, normal or texture coordinate data. Lastly, face is rendered. Each face uses three arrays of indices. As it was explained before, each one is for the normal, texture coordinate and vertex indices. If the model does not use normals or texture coordinates then indices arrays includes 0 values.

### 5.2.6 Finding Materials
The face indices are stored where a material is first used. When a given face needs to be drawn this information is used to change rendering material at the rendering time.

```java
private HashMap<Integer, String> faceMats;
private HashMap<String, Integer> matCount;
public FaceMaterials()
{
    faceMats = new HashMap<Integer, String>();
    matCount = new HashMap<String, Integer>();
    // end of FaceMaterials()

    public void addUse(int faceIdx, String matName)
    {
        if (faceMats.containsKey(faceIdx))
        {
            System.out.println("Face index " + faceIdx + " changed to use material " + matName);
            faceMats.put(faceIdx, matName);
            if (matCount.containsKey(matName)) {
                int i = (Integer) matCount.get(matName) + 1;
                matCount.put(matName, i);
            }
        } else
        {
            matCount.put(matName, 1);
        }
    // end of addUse()

    public String findMaterial(int faceIdx)
    {
        return (String) faceMats.get(faceIdx);
    }

    public void showUsedMaterials()
    {
        Set<String> keys = matCount.keySet();
        Iterator<String> iter = keys.iterator();
        String mName;
        int count;
        while (iter.hasNext()){
            mName = iter.next();
            count = (Integer) matCount.get( mName );
        }
    // end of showUsedMaterials()
}
// end of FaceMaterials class
```

Table 5.7: Finding Materials

Table 5.7 can be specified with some steps:

- Before the addUse() method, faceMats stores a HashMap of face indices mapped to material names. When findMaterial() is called with a face index, looked inside this HashMap. If there is no face index in the HashMap, the method is returned null.
- **addUse()**: This method stores the face index and the material is uses. Also, it stores how many times matName has been used by faces.
- **showUsedMaterials()**: It shows all the material which are used by faces.
5.2.7 Reading and Rendering MTL Files
This class loads the material details from the MTL file and this class stores these details as material objects in the materials ArrayList. Also, it sets up a signified material’s colours or textures to be used, when it is rendered. Inner Materials Class is:

```java
private static final String MODEL_DIR = "";
private ArrayList<Material> materials;
private String renderMatName = null;
private boolean usingTexture = false;
public Materials(String mtlFnm) {
    materials = new ArrayList<Material>();
    String mfnm = MODEL_DIR + mtlFnm;
    try {
        BufferedReader br = new BufferedReader(new FileReader(mfnm));
        readMaterials(br);
        br.close();
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
}
private void readMaterials(BufferedReader br) {
    try {
        String line;
        Material currMaterial = null;
        while (((line = br.readLine()) != null)) {
            line = line.trim();
            if (line.length() == 0) continue;
            if (line.startsWith("newmtl ")) {
                if (currMaterial != null)
                    materials.add(currMaterial);
                currMaterial = new Material(line.substring(7));
            } else if (line.startsWith("map_Kd ")) {
                String fileName = MODEL_DIR + line.substring(7);
                currMaterial.loadTexture(fileName);
            } else if (line.startsWith("Ka "))
                currMaterial.setKa( readTuple3(line) );
            else if (line.startsWith("Kd "))
                currMaterial.setKd( readTuple3(line) );
            else if (line.startsWith("Ks "))
                currMaterial.setKs( readTuple3(line) );
            else if (line.startsWith("Ns ")) {
                float val = Float.valueOf(line.substring(3)).floatValue();
                currMaterial.setNs( val );
            } else if (line.charAt(0) == 'd') {
                float val = Float.valueOf(line.substring(2)).floatValue();
                currMaterial.setD( val );
            } else if (line.startsWith("illum ")) {
                // not implemented
            } else if (line.charAt(0) == '#')
                continue;
```
```java
materials.add(currMaterial);
}
```
As is seen from table 5.8, on Materials class:

- The reading is started with first lines. Firstly, material objects are stored that is built from the MTL file data. For the storing, the current material is used firstly. It is a null value for the first step.
- **readMaterials():** MTL file is parsed line by line. For the new material, it is started to be built with material ArrayList. Because material objects are stored in ArrayList. To the collecting material info, it is started with texture file name, ambient colour, diffuse colour, specular colour, shininess, alpha values. Thus, material is read.
- **readTuple3():** The line starts with MTL words like Ka, Kd or Ks. So, this word is skipped and the three floats (x, y and z) are separated spaces. Then the values are sent to these floats.
- **renderWithMaterial():** Here, the information is rendered with the material. This information is texture or colors. But, it is realized if faceMat is equal to rendering material (renderMatName). If everything is OK and if it is a new material then texture/s and colors are used for the material.
- **switchOffTex():** The texturing is made off and lights get on.
- **switchonText():** The lights are made off and the texturing gets on.
- **getTexture():** it return the texture that is associated with the material name.
- **setMaterialColors():** The rendering starts with using colors by the named material.

### 5.2.8 Setting Materials

A material object keeps texture and colour information for a named material. Also, this object manages the rendering with the using its colours. The texturing is done by material object to the rendering. The codes for this class:

```java
public class Material
{
    private String name;
    private Tuple3 ka, kd, ks;
    private float ns, d;
    private String texFnm;
    private Texture texture;

    public Material(String nm)
    {
        name = nm;
        d = 1.0f; ns = 0.0f;
        ka = null; kd = null; ks = null;
        texFnm = null;
        texture = null;
    }
}
```
public void showMaterial()
{
    System.out.println(name);
    if (ka != null)
        if (kd != null)
            if (ks != null)
                if (ns != 0.0f)
                    if (d != 1.0f)
                        if (texFnm != null)
                            System.out.println(" Texture file: "+ texFnm);
    // end of showMaterial()
}

public boolean hasName(String nm)
{
    return name.equals(nm);
}

public void setD(float val)
{
    d = val;
}

public float getD()
{
    return d;
}

public void setNs(float val)
{
    ns = val;
}

public float getNs()
{
    return ns;
}

public void setKa(Tuple3 t)
{
    ka = t;
}

public Tuple3 getKa()
{
    return ka;
}

public void setKd(Tuple3 t)
{
    kd = t;
}

public Tuple3 getKd()
{
    return kd;
}

public void setKs(Tuple3 t)
{
    ks = t;
}

public Tuple3 getKs()
{
    return ks;
}

public void setMaterialColors(GL2 gl)
{
    if (ka != null) {
        float[] colorKa = { ka.getX(), ka.getY(), ka.getZ(), 1.0f };
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_AMBIENT, colorKa, 0);
    }
    if (kd != null) {
        float[] colorKd = { kd.getX(), kd.getY(), kd.getZ(), 1.0f };
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_DIFFUSE, colorKd, 0);
    }
    if (ks != null) {
        float[] colorKs = { ks.getX(), ks.getY(), ks.getZ(), 1.0f };
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_SPECULAR, colorKs, 0);
    }
    if (ns != 0.0f) {
        gl.glMaterialf(GL.GL_FRONT_AND_BACK, GL2.GL_SHININESS, ns);
    }
    if (d != 1.0f) {
        // not implemented
    }
    // end of setMaterialColors()
}

public void loadTexture(String fnm)
{
    try {
        texFnm = fnm;
        texture = TextureIO.newTexture( new File(texFnm), false);
        texture.setTextureParameteri(GL2.GL_TEXTURE_MAG_FILTER,
As is seen from table 5.9, on this class:

- The name or material, ambient, diffuse, specular colour info, shininess and alpha values and texture info are defined.
- Except the shininess and alpha values, other variables are designated as null values. Because it will be changed when material is shown.
- `showMaterial()`: Here, material is polled. If there is a material, it is shown.
- `set/get methods()`: These methods are used for colour info. Then colour info is placed in variables.
- `setMaterialColors()`: The rendering starts with this material’s colour information.
- `loadTexture()`: The texture information is taken by this method and texture is loaded here when model is rendered.

### 5.2.9 Loading the Models

On this class, the necessary methods are written like init, reshape and display. Also, the object model is called with loading and drawing on this class. But, the fact process is realized on OBJModel class. OBJModel class is just identified on the main class to get the objects. The lights (if there is no light in OBJ and MTL file), GLU, GLUT, fps (speed), Canvas, Animator (for the making interactive), screen size are identified on this class.

To the loading and drawing 3D object models (totally 18 3D objects for the three-dimensional building) on this class with display method;
private OBJModel15 loader15;
private OBJModel16 loader16;
private OBJModel17 loader17;
private final String model_name = "ceiling";
private final String model_name1 = "floor1";
private final String model_name2 = "wall";
private final String model_name3 = "leftwall";
private final String model_name5 = "painting1";
private final String model_name6 = "table";
private final String model_name7 = "glassleft";
private final String model_name8 = "carpet";
private final String model_name9 = "flower";
private final String model_name10 = "glassmiddle";
private final String model_name11 = "painting2";
private final String model_name12 = "painting3";
private final String model_name13 = "painting4";
private final String model_name14 = "glassright";
private final String model_name15 = "door";
private final String model_name16 = "fanofceiling";
private final String model_name17 = "television";
public void display(GLAutoDrawable drawable){
    try {
        Modeldraw(drawable);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private void Modeldraw(GLAutoDrawable drawable) throws IOException {
    final GL2 gl = (GL2) drawable.getGL();
    loader = new OBJModel(model_name, .5f, gl, true);
    loader.draw(gl);
    loader1 = new OBJModel1(model_name1, .5f, gl, true);
    loader1.draw(gl);
    loader2 = new OBJModel2(model_name2, .5f, gl, true);
    loader2.draw(gl);
    loader3 = new OBJModel3(model_name3, .5f, gl, true);
    loader3.draw(gl);
    loader4 = new OBJModel4(model_name4, .5f, gl, true);
    loader4.draw(gl);
    loader5 = new OBJModel5(model_name5, .5f, gl, true);
    loader5.draw(gl);
    loader6 = new OBJModel6(model_name6, .5f, gl, true);
    loader6.draw(gl);
    loader7 = new OBJModel7(model_name7, .5f, gl, true);
    loader7.draw(gl);
    loader8 = new OBJModel8(model_name8, .5f, gl, true);
    loader8.draw(gl);
    loader9 = new OBJModel9(model_name9, .5f, gl, true);
    loader9.draw(gl);
    loader10 = new OBJModel10(model_name10, .5f, gl, true);
    loader10.draw(gl);
    loader11 = new OBJModel11(model_name11, .5f, gl, true);
    loader11.draw(gl);
    loader12 = new OBJModel12(model_name12, .5f, gl, true);
    loader12.draw(gl);
    loader13 = new OBJModel13(model_name13, .5f, gl, true);
    loader13.draw(gl);
    loader14 = new OBJModel14(model_name14, .5f, gl, true);
    loader14.draw(gl);
    loader15 = new OBJModel15(model_name15, .5f, gl, true);
    loader15.draw(gl);
Table 5.10: Loading the Models

Table 5.10 shows that; to the drawing these 3D models, the draw() method is used for callback method. Because the model is rendered to a display list in `drawtoList()` method. It means, it waits to be shown there. When the draw() command is run, it is shown quickly.

As a result, 3D object models are loaded without interactivity when the application is rendered. This means, three-dimensional structure (the room) is shown with 3D model on Java application like it was made with Autodesk 3D Studio Max but, with some features like lights and scaling. Thus, 3D models are ready to get interactivity. This is seen in the last step for the arriving main goal completely.

5.3 Key Implementations

3D objects can be seen well when they are rendered on Java application. This is good option because it provides good graphic solutions yet; interactivity provides better options like the moving. Thus, any object can be replaced. Also, this creates game structure. It means, the player can be used instead for object or objects. Thus, the game is started. So, the interactivity is better option to develop 3D graphic solutions. For this, the connections are made with keyboard keys. To the making connections with keyboard keys, the objects are designated that will be made interactive. To the main goal on this thesis the chosen objects are paintings. It means, OBJModel5 (painting1), OBJModel11 (painting2), OBJModel12 (painting3) and OBJModel13 (painting4). These objects are made interactive when they are loaded and drawn. It means, this process is made again in `Display()` method.

```java
loader16 = new OBJModel16(model_name16, .5f, gl, true);
loader16.draw(gl);
loader17 = new OBJModel17(model_name17, .5f, gl, true);
loader17.draw(gl);
} // end of Display Method
```

```java
// Table 5.10: Loading the Models

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_name16</td>
<td>3D model rendered on Java application</td>
</tr>
<tr>
<td>model_name17</td>
<td>3D model rendered on Java application</td>
</tr>
<tr>
<td>model_name5</td>
<td>3D model rendered on Java application</td>
</tr>
<tr>
<td>model_name11</td>
<td>3D model rendered on Java application</td>
</tr>
<tr>
<td>model_name12</td>
<td>3D model rendered on Java application</td>
</tr>
<tr>
<td>model_name13</td>
<td>3D model rendered on Java application</td>
</tr>
</tbody>
</table>
```

```java
loader5 = new OBJModel5(model_name5, .5f, gl, true);
gl.glPushMatrix();
gl.glTranslatef(Rx5, 0, 0);
gl.glTranslatef(0, Ry5, 0);
loader5.draw(gl);
gl.glPopMatrix();
loader11 = new OBJModel11(model_name11, .5f, gl, true);
```
gl.glPopMatrix();

public void keyPressed(KeyEvent e) {
  switch (e.getKeyCode()) {
    // for painting 1.
    case KeyEvent.VK_RIGHT:
      Rx = Rx + 5;
      System.out.println("go right");
      break;
    case KeyEvent.VK_ESCAPE:
      System.out.println("exit");
      System.exit(0);
      break;
    case KeyEvent.VK_DOWN:
      Ry = Ry - 5;
      System.out.println("go down");
      break;
    case KeyEvent.VK_LEFT:
      Rx = Rx - 5;
      System.out.println("go left");
      break;
    case KeyEvent.VK_UP:
      Ry = Ry + 5;
      System.out.println("go up");
      break;
    // for painting 2
    case KeyEvent.VK_W:
      Ry1 = Ry1 + 5;
      System.out.println("go up");
      break;
    case KeyEvent.VK_A:
      Rx1 = Rx1 - 5;
      System.out.println("go left");
      break;
    case KeyEvent.VK_S:
      Ry1 = Ry1 - 5;
      System.out.println("go down");
      break;
    case KeyEvent.VK_D:
      Rx1 = Rx1 + 5;
      System.out.println("go right");
      break;
    // for painting 3
    case KeyEvent.VK_T:
      Ry2 = Ry2 + 5;
      System.out.println("go up");
      break;
    case KeyEvent.VK_F:
      Rx2 = Rx2 - 5;
      System.out.println("go left");
      break;
    case KeyEvent.VK_H:
      Rx2 = Rx2 + 5;
      System.out.println("go right");
      break;
    case KeyEvent.VK_G:
      Ry2 = Ry2 - 5;
      System.out.println("go down");
      break;
    // for painting 4
    case KeyEvent.VK_1:
      Ry3 = Ry3 + 5;
      System.out.println("go up");
  }
}
```java
break;
case KeyEvent.VK_2:
    Ry3 = Ry3 -5;
    System.out.println("go down");
break;
case KeyEvent.VK_3:
    Rx3 = Rx3 -5;
    System.out.println("go left");
break;
case KeyEvent.VK_4:
    Rx3 = Rx3 +5;
    System.out.println("go right");
break;
}
```

Table 5.11: Key Implementations

On the table 5.11; the chosen objects are implemented with keyboard keys. Also, the variables (Rx, Rx1, Rx2, Rx3, Ry, Ry1, Ry2 and Ry3), animator and the key listener and event listener are identified, when codes are started to be written. As is seen from codes, codes are designated to keyboard keys. But, they are replaced with translatef() commands. PushMatrix() and PopMatrix() commands hold first and last positions of these object. Thus, objects are replaced. At the end of the process, objects get interactivity.
6. RESULTS

From the first steps to the last step:

- 3D models have been designed with Autodesk 3D Studio Max.
- Java and Java SE Runtime Environment 7 have been updated and installed.
- Eclipse has been downloaded and Java OpenGL (JOGL) has been implemented on Eclipse.
- The implementing and the developing have been made to 3D models. (the codes have been written and re-changed)
- The integration has been made with keyboard keys to make interactive.

Every step has been made for the main goal. Thus, application can be run to see 3D results with interactivity. On the following figures, interactive rendering results are shown.

Figure 6.1: Result 1 (the loaded objects)

Figure 6.2: Result 2 (The first painting is brought to the left)
Figure 6.3: Result 3 (the top paintings are replaced)

Figure 6.4: Result 4 (the bottom paintings are replaced)
As is seen from figures (6.1-6.6), 3D objects are loaded and re-drawn when they are rendered. With the keys, they are re-placed. But, x and y coordinates are used for making interactivity. The paintings can be sent to every point with these coordinates on back wall. Of course, it gets the third coordinate (z) but, two coordinates are used to understand basically this process. When every step is combined for loading models, these steps generate a new step for the better 3D graphic solutions. From first steps to last step, each step has been completed with details to understand this process. Thus, every sub-goal is completed and arrived to the main goal. Consequently, the taken every step put a stone for the three-dimensional interactive gallery.
Interactive gallery includes different platforms to be made. On these platforms, different methods and techniques are used. So, there can be some disadvantages as well as advantages for these working environments. For instance, Autodesk 3D Studio Max has big properties and when it is used, it makes better and better three-dimensional graphic solutions yet, it is expensive 3D modelling software and it is hard to find out everything about it [22]. So, Autodesk 3D Studio Max v.2011 which is for students and educators (see www.autodesk.com) product was used to make these 3D models. This means, when Autodesk 3D Studio Max is used completely, it presents new options for designing many realistic outputs. The outputs get effective and like real. Thus, product is achieved. This provides attraction for computer users or customers. For example, the designing 3D environments for a game with Autodesk 3D Studio Max, this can be attractive for game addicted. Because of people who like games so much want to live that time in the game like real, when they play. With other example, the realistic applications can be instructional for education. With this way, students learn quickly and practically.

Java is one of the most popular languages. It runs almost on every operating system. It supports applications that are Java based. With Java, applications can be developed, games are designed, and many applications can be written. On this thesis, Java needed to use Java SE Runtime Environment 7 because Java application needs this environment to be run. The other alternative choice is JDK (Java Development Kit). With this kit, Java applications are developed. The difference is for these two environments, JRE (Java Runtime Environment) is used to run Java applications. The JDK (Java Development Kit) is used to develop Java applications. Also, Java applications are run with JDK. Because of JDK includes also Java Runtime Environment. It means, there is no separate downloading for them.

JOGL is a wrapper library and provides direct access to OpenGL functions. However, there are some other 3D graphic options like Java3D or LWJGL. LWJGL is low-level library like JOGL. Actually, JOGL and LWJGL are pretty much same. There is just so little speed difference and LWJGL has more static methods. LWJGL is oriented toward game development. In addition to 3D graphics, it supports for hardware like gamepads and controllers. JOGL is to the learning 3D graphics with its details and 3D graphics are learnt deeply with JOGL while there is Java programming language. Lastly, Java3D is high-level Java API and it uses object-oriented way to the doing 3D graphics. It allows easily create 3D object and set these 3D objects up for the display. However, JOGL is used to basically understand these 3D object structures and it gives information about drawing these objects with codes. So, JOGL provides wide approach with this direction [24].

When the loading models are examined to the rendering results, there are a lot of OBJModel classes. It means the losing time when it rendered. With another word, application can run slow when every object model is loaded at the same time. Of course, there is a solution for this. For instance, every object model can be exported under an OBJ file. Then the last objects which will be made interactive are exported separately. Thus, five OBJModel classes are used instead of eighteen OBJModel classes. But, if some objects are changed or there is no needing anymore for some objects, the whole file
is designed from the beginning and it can want the scaling or the being editable. So, the models are designed again. However, when all objects are exported separately, the unwanted object can be ignored or closed. It can be little bit slowly but, it obviates re-designing.

With the general structure of interactive gallery, it can be compared with virtual museums in one way. Virtual museums have 3D graphics, interactive solutions, virtual reality and these products are collected on the web technology with databases and necessary environments. When a virtual museum is presented on the web for the purpose of it, it gets one more point with its advanced technological sides. Here, the interactive gallery can be seen as a level in the virtual museum process yet, interactive gallery constitutes the basic step for beginning of this process and it has many solutions to pass sub-steps like designing, exporting, programming, giving interactivity and its theoretical side. So, the interactive galley can be called as an inception for going advanced galleries [4].
8. CONCLUSION AND FUTURE WORKS

An interactive gallery can be created in many ways. In this thesis, interactive gallery has been created by using Autodesk 3D Studio Max, Java OpenGL (JOGL) and Eclipse. Thus, an interactive gallery has been displayed interactively on graphic API (OpenGL). This indicates that an interactive gallery can be developed by using different platforms. Here, the answer is about using method. Model loading has been used to create simple interactive gallery and it encapsulates eighteen objects. This can be also used for a model in this method. After all steps, rendered project calls, objects and graphic API. Thus, objects can be seen as interactive, realistic and effective on graphic API because the used platforms provide this. In this manner, more realistic or effective galleries can be created by using these platforms because of their easy usage, less coding and understandable sides. Finally, developer can create his/her own game, interactive gallery, architectural application or an advanced virtual museum.

As a result, models have been designed and exported then these models have been loaded and the models have been made interactive successfully. This explains how a gallery can be developed and displayed interactively by using model loading method.

As a future work, the planned study can be a virtual museum with web and database support and if there are other necessary environments. The reason can be interactive gallery because 3D modelling software (Autodesk 3D Studio Max) supports Virtual Reality Modelling Language (VRML) files. This means a virtual museum can be created by starting to develop an interactive gallery. Thus, new database and web technology tools like WebGL and XML supervene upon interactive gallery.
REFERENCES


A. APPENDICES

A.1 Installing Java SE Runtime Environment 7 (JRE)
To the running Java application, Java SE Runtime Environment is installed. Because, the loadings objects are run with Java and JOGL is a Java interface. The following webpage shows Java SE Runtime Environment (JRE). http://www.oracle.com/technetwork/Java/javase/downloads/index.html. The following figure shows Java’s webpage and the chosen Java Runtime Environment

![Java SE Runtime Environment](image)

Figure A.1: Java SE Runtime Environment 7 Options [29].

To the updating, Java version is chosen which is suitable for the system. For this, www.Java.com web link can be followed. After the updating, Java SE Runtime Environment is specified with Java version. As is seen from Figure 3.2.1, Java SE Runtime Environment 7 (JRE) is selected. Firstly, the License Agreement is accepted and then the product is selected which is suitable for the operating system. Lastly, the JRE is downloaded and installed (Here, Windows X86 Offline version was selected). Thus, Java platform gets ready for the supporting Java applications.

A.2 Downloading Eclipse and the Implementing Java OpenGL (JOGL)
The Eclipse platform provides interface to develop software with the working tools. Here, the plug-in is JOGL that is Java interface. Thus, 3D objects can be loaded and rendered with the Eclipse platform. Firstly, the Eclipse is downloaded and then it is worked directly with the define file place (workplace). To the downloading Eclipse, http://www.eclipse.org/downloads/ web link can be followed. Then the suitable eclipse package is selected with pursuant to operating system (32 bit/64 bit). Here, Eclipse for Java EE Developers has been selected for the thesis. The download comes with .rar file. Then files are extracted.

After the downloading and extracting, eclipse is run. While it starting to run, the file place is identified for the working with Java files. When the eclipse is opened, the following figure is seen.
Just downloading is not enough for the working with Java. So, JOGL is added as plug-in. Thus, JOGL provides Java interface and graphic libraries. Therefore, JOGL is implemented on eclipse. For this, there are two ways. One of them is before starting, Jogl.dll file and jogl.jar file are placed into JRE 7 file. JRE7 file can be “C:\Program Files\Java\jre7\bin” directory. Jogl.dll is copied into the bin file which is under the JRE7. The jogl.jar file follows “C:\Program Files\Java\jre7\lib\ext” directory. Then it is copied there.

The second way is clearer than first one. For this, these files are added as plug-ins. The following pictures demonstrate this adding process.
1- ) Firstly, “jogamp-windows-i586.7z” library is downloaded and files are extracted.
2- ) The Eclipse is run.
3- ) File > New > Java Project is selected.
4- ) here, the real process starts. The next figure shows this.
Figure A.3: The Implementing JOGL Step 1

Here, the given name for the project and Java environment is identified. Then the next button is clicked and the next page comes to add JOGL files.

Figure A.4: The Implementing JOGL Step 2

5- ) in this step, Libraries sub-menu is selected. JOGL .jar files and .dll files are added. For the .jar files, Libraries > Add External JARs:

.jar Files can be found on the jogamp-windows-i586.jar file then

- Gluegen-rt.jar
- Jogl.all.jar
- Nativewindow.awt.jar
- Newt.awt.jar files are taken.
Also, for .jar files can be used as a user library. It means there is no adding these files every time.

6-) The next step is to add .dll files. After adding .jar files, Native library location: (None) is found. It is under the .jar files which are added in the previous step and in the JRE System Library [JavaSE-1.7]. The next figure makes it clearer.

![Figure A.5: The Implementing JOGL Step 3](image)

After founding Native library location option,
- Edit is clicked.
- In the coming options, External Folder is selected and .dll files are found. These files are in the jogamp-windows-i586\lib file.
- The lib file is selected and then OK is clicked.

Thus, Java OpenGL (JOGL) library is made ready for the writing Java codes. But, firstly, 3D object models are exported individually. The next step explains this.

### A.3 Exporting 3D Object Models with Obj File Format

The Wavefront OBJ file format offers many advanced elements like faces, normals and texture coordinates. So, Wavefront OBJ file format is chosen to load object models. Also, the material information is stored in a separate MTL file. With the information that comes with the exporting is called in Java OpenGL (JOG), when Java application is run. Every object model is exported individually which is designed with Autodesk 3D Studio Max. For the exporting obj models:
- Application Button > Export > Export
- The file place is found to save. The workplace\project name is suitable place for the exporting. Thus, the application calls these models from same file, when it is run.
- OBJ file format is chosen.
- ObjModel is named.
- It is saved as an OBJ file.
After these steps, the following figure comes.

![Image of OBJ Export Options window]

**Figure A.6: The Exporting Process Step 1**

Here, the export options are specified. For instance, if object model will be exported with triangle faces, it is selected. It means, when object model is sent with triangle faces, the f points are identified as triangular. Also, there are quads and polygons options. With other explanations, if texture coordinates are sent, these coordinates are applied on model, when Java application is run. When the export button is clicked, the exporting process starts. The next figure shows this process.

![Image of Export Statistics window]

**Figure A.7: The Exporting Process Step 2**

As is seen from Figure A.7, objmodel was exported. At the exporting time, many features are sent to definite file like geometry vertices (v), texture vertices (vt), vertex normals (vn) and faces (triangles). With this method, each objmodel (totally
18 objmodels) is exported and sent to workplace\project name. Thus, obj models get ready to be loaded.

A.4 Java OpenGL (JOGL) Model Loading Tutorial
According to Christian Lührs; one of the used methods is the importing 3D models for the model loading. With this method, object model is read and loaded [11].

To the loading three-dimensional room, this method is used. But, it does not have interactivity and it can not use object model more than one. So, it has been developed on the implementations part. Here, it has been used as source codes. These codes;

Classes: ModellImport.Java
OBJModel.Java
Tuple3.Java

Inner Classes:
Faces.Java
FaceMaterials.Java
Material.Java
Materials.Java

These classes are used to call and to re-create model which comes with the exporting. This process starts with OBJModel.Java class and it finishes with ModellImport.Java class.

A.4.1 OBJModel Class
This class is the most important class for the model loading. Because of model is read here and it is loaded under this class. This class is seen on table A.1;

```java
import java.io.*;
import java.util.*;
import javax.media.opengl.*;
import com.sun.opengl.util.*;
import com.sun.opengl.util.texture.*;
import java.text.DecimalFormat;

public class OBJModel {
  private static final float DUMMY_Z_TCS = -5.0f;

  // collection of vertices, normals and texture coords for the model
  private ArrayList<Tuple3> verts;
  private ArrayList<Tuple3> normals;
  private ArrayList<Tuple3> texCoords;
  private boolean hasTCs3D = true;

  // whether the model uses 3D or 2D tex coords
  private Faces faces;  // model faces
  private FaceMaterials faceMats; // materials used by faces
  private Materials materials; // materials defined in MTL file
  private ModelDimensions modelDims; // model dimensions
```
private String modelNm;  // without path or ".OBJ" extension
private float maxSize;  // for scaling the model

private int modelDispList;  // the model's display list

public OBJModel(String nm, float sz, GL gl, boolean showDetails) {
    modelNm = nm;
    maxSize = sz;
    initModelData(modelNm);

    loadModel(modelNm);
    centerScale();
    drawToList(gl);

    if (showDetails)
        reportOnModel();
} // end of OBJModel()

private void initModelData(String modelNm) {
    verts = new ArrayList<Tuple3>();
    normals = new ArrayList<Tuple3>();
    texCoords = new ArrayList<Tuple3>();

    faces = new Faces(verts, normals, texCoords);
    faceMats = new FaceMaterials();
    modelDims = new ModelDimensions();
} // end of initModelData()

private void loadModel(String modelNm) {
    String fnm = modelNm + ".obj";
    try {
        System.out.println("Loading model from " + fnm + " ...");
        FileInputStream fis_model = new FileInputStream(fnm);
        BufferedReader br_model = new BufferedReader(
                new InputStreamReader(fis_model));
        readModel(br_model);
        br_model.close();
    } catch (IOException e){
        System.out.println(e.getMessage());
        System.exit(1);
    }
} // end of loadModel()

private void readModel(BufferedReader br) {
    // parse the OBJ file line-by-line
    boolean isLoaded = true;  // hope things will go okay

    int lineNum = 0;
    String line;
    boolean isFirstCoord = true;
    boolean isFirstTC = true;
    int numFaces = 0;

    try {
        while (((line = br.readLine()) != null) && isLoaded) {
            lineNum++;

            // parse line...
if (line.length() > 0) {
    line = line.trim(); //Returns a copy of the string, with leading and
    //trailing whitespace omitted
    if (line.startsWith("v")) { // vertex
        isLoaded = addVert(line, isFirstCoord);
        if (isFirstCoord)
            isFirstCoord = false;
    } else if (line.startsWith("vt")) { // tex coord
        isLoaded = addTexCoord(line, isFirstTC);
        if (isFirstTC)
            isFirstTC = false;
    } else if (line.startsWith("vn")) { // normal
        isLoaded = addNormal(line);
    } else if (line.startsWith("f ")) { // face
        isLoaded = faces.addFace(line);
        numFaces++;
    }
    else if (line.startsWith("mtllib ")) { // load material
        materials = new Materials(line.substring(7));
    } else if (line.startsWith("usemtl ")) { // use material
        faceMats.addUse(numFaces, line.substring(7));
    } else if (line.charAt(0) == 'g') { // group name
        // not implemented
    } else if (line.charAt(0) == 's') { // smoothing group
        // not implemented
    } else if (line.charAt(0) == '#') { // comment line
        continue;
    } else
        System.out.println("Ignoring line " + lineNum + " : " + line);
}
}
catch (IOException e) {
    System.out.println(e.getMessage());
    System.exit(1);
}
if (!isLoaded) {
    System.out.println("Error loading model");
    System.exit(1);
}
// end of readModel()

private boolean addVert(String line, boolean isFirstCoord)
/* Add vertex from line "v x y z" to vert ArrayList, and update the
model dimension's info. */
{
    Tuple3 vert = readTuple3(line);
    if (vert != null) {
        verts.add(vert);
        if (isFirstCoord)
            modelDims.set(vert);
        else
            modelDims.update(vert);
        return true;
    }
    return false;
} // end of addVert()
private Tuple3 readTuple3(String line) {
/* The line starts with an OBJ word ("v" or "vn"), followed by three floats (x, y, z) separated by spaces */
StringTokenizer tokens = new StringTokenizer(line, " ");
tokens.nextToken(); // skip the OBJ word
try {
float x = Float.parseFloat(tokens.nextToken());
float y = Float.parseFloat(tokens.nextToken());
float z = Float.parseFloat(tokens.nextToken());
return new Tuple3(x, y, z);
} catch (NumberFormatException e) {
    System.out.println(e.getMessage());
}
return null; // means an error occurred
} // end of readTuple3()

private boolean addTexCoord(String line, boolean isFirstTC) {
/* Add the texture coordinate from the line "vt x y z" to the texCoords ArrayList. There may only be two tex coords on the line, which is determined by looking at the first tex Coord line. */
if (isFirstTC) {
    hasTCs3D = checkTC3D(line);
    System.out.println("Using 3D tex coords: "+ hasTCs3D);
}
Tuple3 texCoord = readTCTuple(line);
if (texCoord != null) {
texCoords.add(texCoord);
return true;
}
return false;
} // end of addTexCoord()

private boolean checkTC3D(String line) {
/* Check if the line has 4 tokens, which will be the "vt" token and 3 tex coords in this case. */
String[] tokens = line.split("\s+");
return (tokens.length == 4);
} // end of checkTC3D()

private Tuple3 readTCTuple(String line) {
/* The line starts with a "vt" OBJ word and two or three floats (x, y, z) for the tex coords separated by spaces. If there are only two coords, then the z-value is assigned a dummy value, DUMMY_Z_TC. */
StringTokenizer tokens = new StringTokenizer(line, " ");
tokens.nextToken(); // skip "vt" OBJ word
try {
    float x = Float.parseFloat(tokens.nextToken());
    float y = Float.parseFloat(tokens.nextToken());
    float z = DUMMY_Z_TC;
    if (hasTCs3D)
        z = Float.parseFloat(tokens.nextToken());
} catch (NumberFormatException e) {
    System.out.println(e.getMessage());
}
return new Tuple3(x, y, z);
} // end of readTCTuple()
```java
return new Tuple3(x, y, z);
} catch (NumberFormatException e) {
    System.out.println(e.getMessage());
} return null; // means an error occurred
} // end of readTCTuple()

private boolean addNormal(String line)
// add normal from line "vn x y z" to the normals ArrayList
{
    Tuple3 normCoord = readTuple3(line);
    if (normCoord != null) {
        normals.add(normCoord);
        return true;
    }
    return false;
} // end of addNormal()

private void centerScale()
/* Position the model so it's center is at the origin, and scale it so
its longest dimension is no bigger than maxSize. */
{
    // get the model's center point
    Tuple3 center = modelDims.getCenter();

    // calculate a scale factor
    float scaleFactor = 1.0f;
    float largest = modelDims.getLargest();
    // System.out.println("Largest dimension: "+largest);
    if (largest != 0.0f)
        scaleFactor = (maxSize / largest);
    System.out.println("Scale factor: "+scaleFactor);

    // modify the model's vertices
    Tuple3 vert;
    float x, y, z;
    for (int i = 0; i < verts.size(); i++) {
        vert = (Tuple3) verts.get(i);
        x = (vert.getX() - center.getX()) * scaleFactor;
        vert.setX(x);
        y = (vert.getY() - center.getY()) * scaleFactor;
        vert.setY(y);
        z = (vert.getZ() - center.getZ()) * scaleFactor;
        vert.setZ(z);
    }
} // end of centerScale()

private void drawToList(GL gl)
/* render the model to a display list, so it can be drawn quicker
later */
{
    modelDispList = gl.glGenLists(1);
    gl.glNewList(modelDispList, GL.GL_COMPILE);
    gl.glPushMatrix();
    // render the model face-by-face
    String faceMat;
    for (int i = 0; i < faces.getNumFaces(); i++) {
        faceMat = faceMats.findMaterial(i); // get material used by face i
        if (faceMat != null)
```
On the table A.1, model is read and it is loaded. Later on, the normals, texture coordinates, vertexes are added. The model features are taken with other explanation. Then the model is centred and scaled. That is the reason for the using an object model. When second object model is used, the application puts second object on the first one and the application centred all of object models. So, the private void centerScale() command lines and its contents are closed to the using more object models. The scale can be made, when object is exported. That is also another option to the scaling model.

This allows making interactive these objects. On the chapter 4, each object is saved separately. For the interactive gallery, there are five object models in this tutorial. These object are made interactive without private void centerScale() command lines.

### A.4.2 Tuple3 Class
This class is used to the taking normals, texture coordinates and vertexes. This class stores these contents. When the model is rendered, it needs these information and the stored features are sent by using this class.
public String toString()  
{  
    return "(" + x + ", " + y + ", " + z + ")";  
}  

public void setX(float xc)  
{  
    x = xc;  
}  

public float getX()  
{  
    return x;  
}  

public void setY(float yc)  
{  
    y = yc;  
}  

public float getY()  
{  
    return y;  
}  

public void setZ(float zc)  
{  
    z = zc;  
}  

public float getZ()  
{  
    return z;  
}  
}  
// end of Tuple3 class

Table A.2: Tuple3 Class

As is seen from table A.2, these contents are stored in variables. When the model is loaded, the stored content is sent from here.

A.4.3 ModelDimensions Inner Class

In this class, edge coordinates are calculated. The edge coordinates are used to model’s width, height and the largest dimension [11].

public class ModelDimensions  
{  
    // edge coordinates  
    private float leftPt, rightPt;  // on x-axis  
    private float topPt, bottomPt;  // on y-axis  
    private float farPt, nearPt;  // on z-axis  

    // for reporting  
    private DecimalFormat df = new DecimalFormat("0.##");  // 2 dp  

    public ModelDimensions()  
    {  
        leftPt = 0.0f;  rightPt = 0.0f;  
        topPt = 0.0f;  bottomPt = 0.0f;  
        farPt = 0.0f;  nearPt = 0.0f;  
    }  
    // end of ModelDimensions()  

    public void set(Tuple3 vert)  
    // initialize the model's edge coordinates  
    {  
        rightPt = vert.getX();  
        leftPt = vert.getX();  

        topPt = vert.getY();  
        bottomPt = vert.getY();  

        nearPt = vert.getZ();  
        farPt = vert.getZ();  
    }  
}
public void update(Tuple3 vert)
// update the edge coordinates using vert
{
    if (vert.getX() > rightPt)  
        rightPt = vert.getX();
    if (vert.getX() < leftPt)  
        leftPt = vert.getX();

    if (vert.getY() > topPt)  
        topPt = vert.getY();
    if (vert.getY() < bottomPt)  
        bottomPt = vert.getY();

    if (vert.getZ() > nearPt)  
        nearPt = vert.getZ();
    if (vert.getZ() < farPt)  
        farPt = vert.getZ();
}  // end of update()

// ----------- use the edge coordinates -----------------

public float getWidth()
{  return (rightPt - leftPt);  }

public float getHeight()
{  return (topPt - bottomPt);  }

public float getDepth()
{  return (nearPt - farPt);  }

public float getLargest()
{
    float height = getHeight();
    float depth = getDepth();

    float largest = getWidth();
    if (height > largest)
        largest = height;
    if (depth > largest)
        largest = depth;

    return largest;
}  // end of getLargest()

public Tuple3 getCenter()
{
    float xc = (rightPt + leftPt)/2.0f;
    float yc = (topPt + bottomPt)/2.0f;
    float zc = (nearPt + farPt)/2.0f;
    return new Tuple3(xc, yc, zc);
}  // end of getCenter()

public void reportDimensions()
{
    Tuple3 center = getCenter();

    System.out.println("x Coords: " + df.format(leftPt) + " to " + df.format(rightPt));

    System.out.println("  Mid: " + df.format(center.getX()) + "; Width: ");
Table A.3: ModelDimensions Inner Class

On the table A.3, the edge coordinates are calculated. Then the calculated edge coordinates are used to model’s other dimensions.

A.4.4 Faces Inner Class

This class stores the information about each of a model. Later on, these faces are rendered by using the stored information.
StringTokenizer st = new StringTokenizer(line, " ");
int numTokens = st.countTokens(); // number of v/vt/vn tokens
// create arrays to hold the v, vt, vn indicies
int v[] = new int[numTokens];
int vt[] = new int[numTokens];
int vn[] = new int[numTokens];

for (int i = 0; i < numTokens; i++) {
    String faceToken = addFaceVals(st.nextToken()); // get a v/vt/vn token
    // System.out.println(faceToken);
    StringTokenizer st2 = new StringTokenizer(faceToken, "/");
    int numSeps = st2.countTokens(); // how many '/'s are there in the token
    v[i] = Integer.parseInt(st2.nextToken());
    vt[i] = (numSeps > 1) ? Integer.parseInt(st2.nextToken()) : 0;
    vn[i] = (numSeps > 2) ? Integer.parseInt(st2.nextToken()) : 0;
    // add 0's if the vt or vn index values are missing;
    // 0 is a good choice since real indicies start at 1
    // store the indicies for this face
    facesVertIdxs.add(v);
    facesTexIdxs.add(vt);
    facesNormIdxs.add(vn);
}
catch (NumberFormatException e) {
    System.out.println("Incorrect face index");
    System.out.println(e.getMessage());
    return false;
}
return true;
} // end of addFace()

private String addFaceVals(String faceStr)
/* A face token (v/vt/vn) may be missing vt or vn index values; add
0's in those cases. */
{
    char chars[] = faceStr.toCharArray();
    StringBuffer sb = new StringBuffer();
    char prevCh = 'x'; // dummy value

    for (int k = 0; k < chars.length; k++) {
        if (chars[k] == '/' && prevCh == '/') // if no char between '/'s
            sb.append('0'); // add a '0'
        prevCh = chars[k];
        sb.append(prevCh);
    }
    return sb.toString();
} // end of addFaceVals()

public void renderFace(int i, GL gl)
/* Render the _ith face by getting the vertex, normal, and tex coord indicies for face i. Use those indicies to access the actual vertex, normal, and tex coord data, and render the face.

Each face uses 3 array of indicies; one for the vertex indicies, one for the normal indicies, and one for the tex coord indicies.

If the model doesn't use normals or tex coords then the indicies arrays will contain 0's. */
if (i >= facesVertIdxs.size())  // i out of bounds?
  return;

int[] vertIdxs = (int[]) (facesVertIdxs.get(i));
// get the vertex indicies for face i

int polytype;
if (vertIdxs.length == 3)
polytype = gl.GL_TRIANGLES;
else if (vertIdxs.length == 4)
polytype = gl.GL_QUADS;
else
  polytype = gl.GL_POLYGON;

gl.glBegin(polytype);

// get the normal and tex coords indicies for face i
int[] normIdxs = (int[]) (facesNormIdxs.get(i));
int[] texIdxs = (int[]) (facesTexIdxs.get(i));

/* render the normals, tex coords, and vertices for face i by
accessing them using their indicies */
Tuple3 vert, norm, texCoord;
for (int f = 0; f < vertIdxs.length; f++) {
  if (normIdxs[f] != 0) {  // if there are normals, render them
    norm = (Tuple3) normals.get(normIdxs[f] - 1);
    gl.glNormal3f(norm.getX(), norm.getY(), norm.getZ());
  }
  if (texIdxs[f] != 0) { // if there are tex coords, render them
    texCoord = (Tuple3) texCoords.get(texIdxs[f] - 1);
    if (texCoord.getZ() == DUMMY_Z_TC)  // using 2D tex coords
      gl.glTexCoord2f(texCoord.getX(), texCoord.getY());
    else  // 3D tex coords
      gl.glTexCoord3f(texCoord.getX(), texCoord.getY(), texCoord.getZ());
  }
  vert = (Tuple3) verts.get(vertIdxs[f] - 1);  // render the vertices
  gl.glVertex3f(vert.getX(), vert.getY(), vert.getZ());
}

gl.glEnd();
}  // end of renderFace()

public int getNumFaces()
{ return facesVertIdxs.size();  }
}  // end of inner class Faces

Table A.4: Faces Inner Class

On the table A.4, the face information is stored in the variables. But, this is to put this
information. Later on, face values are added. The each face information is taken. Lastly,
this information is rendered.

A.4.5 FaceMaterials Inner Class

When material is used the first time, this class stores the face indices for this material.
When models are rendered, this information is used to change the rendering material
when a given face needs to be drawn [11]. Table A.5 shows the source code.
public class FaceMaterials
{
  private HashMap<Integer, String> faceMats;
  // the face index (integer) where a material is first used
  
  // for reporting
  private HashMap<String, Integer> matCount;
  // how many times a material (string) is used

  public FaceMaterials()
  {
    faceMats = new HashMap<Integer, String>();
    matCount = new HashMap<String, Integer>();
  } // end of FaceMaterials()

  public void addUse(int faceIdx, String matName)
  {
    // store the face index and the material it uses
    if (faceMats.containsKey(faceIdx)) // face index already present
      System.out.println("Face index " + faceIdx + " changed to use material " + matName);
    faceMats.put(faceIdx, matName);
    // store how many times matName has been used by faces
    if (matCount.containsKey(matName)) {
      int i = (Integer) matCount.get(matName) + 1;
      matCount.put(matName, i);
    } else
      matCount.put(matName, 1);
  } // end of addUse()

  public String findMaterial(int faceIdx)
  {
    return (String) faceMats.get(faceIdx);  }

  public void showUsedMaterials()
  /* List all the materials used by faces, and the number of
  faces that have used them. */
  {
    System.out.println("No. of materials used: " + matCount.size());
    // build an iterator of material names
    Set<String> keys = matCount.keySet();
    Iterator<String> iter = keys.iterator();
    // cycle through the hashmap showing the count for each material
    String matName;
    int count;
    while (iter.hasNext()) {
      matName = iter.next();
      count = (Integer) matCount.get(matName);
      System.out.print( matName + ": " + count);
      System.out.println();
    } // end of showUsedMaterials()
  } // end of FaceMaterials class
}

Table A.5: FaceMaterials Inner Class
A.4.6 Materials Inner Class
Materials Inner Class loads the material details from the MTL file. When object is exported as an OBJ file by three-dimensional graphic design software, it brings a MTL file with itself. This MTL file includes some details for object. Also, this class sets up a specified material colours or textures. These features are used at the rendering time. On the table A.6 source codes of this class are seen.

```java
public class Materials {
    private static final String MODEL_DIR = ""
    private ArrayList<Material> materials;
    // stores the Material objects built from the MTL file data
    // for storing the material currently being used for rendering
    private String renderMatName = null;
    private boolean usingTexture = false;
    public Materials(String mtlFnm) {
        materials = new ArrayList<Material>();
        String mfnm = MODEL_DIR + mtlFnm;
        try {
            System.out.println("Loading material from " + mfnm);
            BufferedReader br = new BufferedReader(new FileReader(mfnm));
            readMaterials(br);
            br.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
    private void readMaterials(BufferedReader br) {
        /* Parse the MTL file line-by-line, building Material objects which are collected in the materials ArrayList. */
        try {
            String line;
            Material currMaterial = null; // current material
            while (((line = br.readLine()) != null)) {
                line = line.trim();
                if (line.length() == 0) continue;
                if (line.startsWith("newmtl ")) {
                    // new material
                    if (currMaterial != null) // save previous material
                        materials.add(currMaterial);
                    currMaterial = new Material(line.substring(7));
                } else if (line.startsWith("map_Kd ")) { // texture filename
                    String fileName = MODEL_DIR + line.substring(7);
                    currMaterial.loadTexture(fileName);
                } else if (line.startsWith("Ka ")) { // ambient colour
                    currMaterial.setKa(readTuple3(line));
                } else if (line.startsWith("Kd ")) { // diffuse colour
```
currMaterial.setRd(readTuple3(line));
else if (line.startsWith("Ks ")) // specular colour
currMaterial.setKs(readTuple3(line));
else if (line.startsWith("Ns ")) { // shininess
    float val = Float.valueOf(line.substring(3)).floatValue();
currMaterial.setNs(val);
}
else if (line.charAt(0) == 'd') { // alpha
    float val = Float.valueOf(line.substring(2)).floatValue();
currMaterial.setD(val);
}
else if (line.startsWith("illum ")) { // illumination model
    // not implemented
} else
    System.out.println("Ignoring MTL line: " + line);
materials.add(currMaterial);
}
catch (IOException e) {
    System.out.println(e.getMessage());
}
// end of readMaterials()

private Tuple3 readTuple3(String line)
/* The line starts with an MTL word such as Ka, Kd, Ks, and
the three floats (x, y, z) separated by spaces */
{
    StringTokenizer tokens = new StringTokenizer(line, " ");
tokens.nextToken(); // skip MTL word

    try {
        float x = Float.parseFloat(tokens.nextToken());
        float y = Float.parseFloat(tokens.nextToken());
        float z = Float.parseFloat(tokens.nextToken());
    
        return new Tuple3(x, y, z);
    } catch (NumberFormatException e) {
        System.out.println(e.getMessage());
    }
    return null; // means an error occurred
} // end of readTuple3()

public void showMaterials()
// list all the Material objects
{
    System.out.println("No. of materials: " + materials.size());
    Material m;
    for (int i=0; i < materials.size(); i++) {
        m = (Material) materials.get(i);
m.showMaterial();
        // System.out.println();
    }
} // end of showMaterials()

        // ----------------- using a material at render time -----------------

public void renderWithMaterial(String faceMat, GL gl)
/* Render using the texture or colours associated with the material, faceMat. But only change things if faceMat is different from the current rendering material, whose name is stored in renderMatName. */
{
    if (!faceMat.equals(renderMatName)) { // is faceMat is a new material?
        renderMatName = faceMat;
        switchOffTex(gl); // switch off any previous texturing
    }

    // set up new rendering material
    Texture tex = getTexture(renderMatName);
    if (tex != null) { // use the material's texture
        // System.out.println("Using texture with " + renderMatName);
        switchOnTex(tex, gl);
    } else { // use the material's colours
        setMaterialColors(renderMatName, gl);
    }
} // end of renderWithMaterial()

public void switchOffTex(GL gl)
// switch texturing off and put the lights on;
// also called from ObjModel.drawToList()
{
    if (usingTexture)
    {
        gl.glDisable(GL.GL_TEXTURE_2D);
        usingTexture = false;
        gl.glEnable(GL.GL_LIGHTING);
    }
} // end of resetMaterials()

private void switchOnTex(Texture tex, GL gl)
// switch the lights off, and texturing on
{
    gl.glDisable(GL.GL_LIGHTING);
    gl.glEnable(GL.GL_TEXTURE_2D);
    usingTexture = true;
    tex.bind();
} // end of resetMaterials()

private Texture getTexture(String matName)
// return the texture associated with the material name
{
    Material m;
    for (int i = 0; i < materials.size(); i++) {
        m = (Material) materials.get(i);
        if (m.hasName(matName))
            return m.getTexture();
    }
    return null;
} // end of getTexture()

private void setMaterialColors(String matName, GL gl)
// start rendering using the colours specifies by the named material
{
    Material m;
    for (int i = 0; i < materials.size(); i++) {
        m = (Material) materials.get(i);
        if (m.hasName(matName))
            m.setMaterialColors(gl);
    }
} // end of setMaterialColors()
A.4.7 Material Inner Class
According to Christian Lührs, this class holds colour and material information for a named material. The material objects also manages the rendering using its colours (see setMaterialColors()). The rendering using the texture is done by the material object [11]. On the following table (A.7), source codes are seen.

```java
public class Material {
    private String name;

    // colour info
    private Tuple3 ka, kd, ks; // ambient, diffuse, specular colours
    private float ns, d; // shininess and alpha

    // texture info
    private String texFnm;
    private Texture texture;

    public Material(String nm) {
        name = nm;
        d = 1.0f; ns = 0.0f;
        ka = null; kd = null; ks = null;
        texFnm = null;
        texture = null;
    } // end of Material()

    public void showMaterial() {
        System.out.println(name);
        if (ka != null) System.out.println(" Ka: " + ka.toString());
        if (kd != null) System.out.println(" Kd: " + kd.toString());
        if (ks != null) System.out.println(" Ks: " + ks.toString());
        if (ns != 0.0f) System.out.println(" Ns: " + ns);
        if (d != 1.0f) System.out.println(" d: " + d);
        if (texFnm != null) System.out.println(" Texture file: " + texFnm);
    } // end of showMaterial()

    public boolean hasName(String nm) {
        return name.equals(nm); }

    // -------- set/get methods for colour info ---------

    public void setD(float val) {
        d = val;
    }

    public float getD() {
        return d;
    }

    public void setNs(float val) {
        ns = val;
    }

    public float getNs()
```
```java
public void setKa(Tuple3 t) {
    ka = t;
}

public Tuple3 getKa() {
    return ka;
}

public void setKd(Tuple3 t) {
    kd = t;
}

public Tuple3 getKd() {
    return kd;
}

public void setKs(Tuple3 t) {
    ks = t;
}

public Tuple3 getKs() {
    return ks;
}

public void setMaterialColors(GL gl) {
    // start rendering using this material's colour information
    System.out.println("--- SET MATERIAL COLOR ---");
    if (ka != null) { // ambient color
        float[] colorKa = { ka.getX(), ka.getY(), ka.getZ(), 1.0f }; // not implemented
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT, colorKa, 0);
    }
    if (kd != null) { // diffuse color
        float[] colorKd = { kd.getX(), kd.getY(), kd.getZ(), 1.0f }; // not implemented
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_DIFFUSE, colorKd, 0);
    }
    if (ks != null) { // specular color
        float[] colorKs = { ks.getX(), ks.getY(), ks.getZ(), 1.0f }; // not implemented
        gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_SPECULAR, colorKs, 0);
    }
    if (ns != 0.0f) { // shininess
        gl.glMaterialf(GL.GL_FRONT_AND_BACK, GL.GL_SHININESS, ns);
    }
    if (d != 1.0f) { // alpha
        // not implemented
    }
} // end of setMaterialColors()

// start rendering using this material's colour information

public void loadTexture(String fnm) {
    try {
        texFnm = fnm;
        texture = TextureIO.newTexture(new File(texFnm), false);
        texture.setTextureParameteri(GL.GL_TEXTURE_MAG_FILTER, GL.GL_NEAREST);
        texture.setTextureParameteri(GL.GL_TEXTURE_MIN_FILTER, GL.GL_NEAREST);
    } catch (Exception e) {
        System.out.println("Error loading texture " + texFnm);
    }
} // end of loadTexture()

public void setTexture(Texture t)
```
A.4.8 Model Import Class

This class is the main class for Java project. To the loading models, this class is run. Before running, the main functions are written. But this class does not have interactivity. On the following table (A.8) codes of main functions are seen.

```java
import Java.io.*;
import Javax.swing.*;
import Java.awt.*;
import Java.awt.event.*;
import Javax.media.opengl.*;
import Javax.media.opengl.glu.*;
import com.sun.opengl.util.*; // GLUT, FPSAnimator

public class ModellImport extends JFrame implements GLEventListener{

    // Viewing Window Frame size.
    private int width = 500;
    private int height = 400;

    private OBJModel loader;
    private final String model_name = "barbell";

    // Global canvas for event handling.
    GLCanvas canvas;

    // GUT and GLUT are global objects so that
    // they do not have to be newed in each frame.
    GLUT glut = new GLUT();
    GLU glu = new GLU();

    // Vars for animation.
    FPSAnimator animator;

    // public static void main(String[] args) {
    // ModellImport frame = new ModellImport();
    // }

    // public ModellImport(){
    // super("3D - ModellImport");
    // setSize(width, height);
    // setFocusable(true);
    // setLayout(new BorderLayout());
    // initModellImporter();
```
addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        exit();
    }
});

setVisible(true);
requestFocus();
canvas.requestFocusInWindow();
pack();
}

private void initModellImporter(){
    GLCapabilities caps = new GLCapabilities();
caps.setDoubleBuffered(true);
caps.setHardwareAccelerated(true);

    canvas = new GLCanvas(caps);
canvas.addGLEventListener(this);

    add(canvas, BorderLayout.CENTER);

    //Canvas gets focus whenever frame is activated.
    addWindowListener(new WindowAdapter(){
        public void windowActivated(WindowEvent e){
            canvas.requestFocusInWindow();
        }
    });

    animator = new FPSAnimator(canvas, 20);
animator.setRunAsFastAsPossible(false);
animator.start();
}
}

////////// Methods defined by GLEventListener //////////
public void init(GLAutoDrawable drawable){
    GL gl = drawable.getGL();

drawable.setGL(new DebugGL(drawable.getGL()));
System.out.println("Init GL is " + gl.getClass().getName());

    //On some systems the reshape call does not seem to
    //happen automatically on init.
    // Set the projection and viewport.
    reshape(drawable, 0, 0, width, height);

    gl.glClearColor(0f, 0f, 0f, 0.0f);
gl.glEnable(GL.GL_DEPTH_TEST);
gl.glShadeModel(GL.GL_SMOOTH);

    // add light
    addLight(gl);

    ///// creates a new Object of an ObjectLoader /////
    loader = new OBJModel(model_name, 1.5f, gl, true);
}

public void addLight(GL gl){
    gl.glEnable(GL.GL_LIGHTING);
gl.glEnable(GL.GL_LIGHT0);
final float ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
gl.glLightfv( GL.GL_LIGHT0, GL.GL_AMBIENT, ambient, 0 );
}

// this method is called when the drawable component is moved or resized
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height)
{
    //System.out.println("reshape()");
    GL gl = drawable.getGL();
    gl.glViewport(x, y, width, height); // size of drawing area
    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glLoadIdentity();
    double aspect = (double)width / (double)height;
    // Field of view in y direction [0..180].
    final double fovy = 45.0f;
    glu.gluPerspective( fovy, aspect, 1.0f, 50.0f );
    // If we do not have a rectangular viewport and
    // want to avoid distortion we have to adjust
    // the viewing volume to the viewport.
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
}

public void displayChanged( GLAutoDrawable drawable, boolean modeChanged, boolean deviceChanged)
{
    System.out.println("displayChanged()");
}

public void display(GLAutoDrawable drawable)
{
    GL gl = drawable.getGL();
    // Clear Framebuffer and Z-Buffer.
    gl.glClear( GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT );
    gl.glLoadIdentity();
    // Set camera
    gl.glTranslatef( -0.0f, -0.0f, -3.0f);
    loader.draw(gl);
    gl.glFlush();
}

int getAndPrintError( GL gl )
{
    int errorCode = gl.glGetError();
    System.out.println("errorCode = ", errorCode);
    return errorCode;
}

// Called via user event and on window closing.
void exit()
{
Table A.8: ModelImport Class

The main class calls other classes and inner classes. The model waits under the OBJModel class to be called. When this class is run, the object model is loaded and it is re-drawn.

Also, this tutorial is original model loading tutorial [11] and it has been used to be developed. By this way, more models have been used for interactive gallery and the chosen objects have been made interactive. However, this tutorial does not include object model more than one. The reason is centering and scaling. When the second object model is used, the application loads all of them over and over. When private void centerScale() command lines are taken off, the other OBJModels can be used. Also, there is no interactivity for this tutorial. When keyboard keys are implemented on this class, models can be moved. This tutorial explains basically the model loading process for an object model.