

Java 3D Scene Engine

Ashutosh A. Morwal

September 2012

Dissertation submitted in partial fulfilment for the degree of
Master of Science in Advanced Computing

Computing Science and Mathematics
University of Stirling

Abstract

Developing user friendly graphical user interface (GUI) which will avail developers to create portable scene graphs which can be used in java3D applications or in applets on websites.

Users should be able to-

- Put primitive shapes given by java3D, as well as external models supplied by third parties in to their scene graphs.
- After adding the objects users should be able to move using interactions, or animate objects.
- Adding textures objects.
- Delete added objects.
- Save/Load scene graphs.

The method that was adopted for the development of this application was spiral development model, throughout the development of the software; the source code was revised many times and modified for the improvement, the core design of the software was also changed to make the application substantially faster than it was at the early stages.

So far, the application performs up to the expectations and requirements which were specified at the beginning. Most of the requirements specified have been fulfilled.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this dissertation reports original work by me during my University project except for the following -

- The idea for detaching the live scene graph, and making changes to it was taken from one of the forum discussions on java.net.[8]
- The PropManager.java class used in loading of external models was taken straight from the book, Killer Game Programming in java. [6]
- Many bits in _basic3DWorld are based on the WrapCheckers3D.java class which was provided in the 3D assignment during the second semester of the course. The basic scene building components such as the building of Canvas3D in _basic3DWorld as well as the lightScene() method, and addBackground() method, were also part of the 3D assignment.
- The move() method in Mover class is based on the doMove() method which was given in the 3D assignment.

Signature

Date

Acknowledgements

This project was not possible without the unduly support of my family, friends and my supervisor Dr. Bruce Graham. During the course of the project, I used to meet Dr. Graham weekly; these meetings helped me improve my project in many ways. Also my sister, Aishwarya, and my dad, helped me a lot; both of them assisted me with the report and suggested valuable improvements. Aishwarya, as an architecture student also helped me test few bits of my code.

I'd also like to thank Dr. Simon Jones, who helped me with choosing this topic as my dissertation, and helped me clear any doubts that I was having in the initial stages of the information gathering.

The folks at java.net were very helpful; while working on the design of the project, the idea of implementing live scene graph was actually suggested by one of the member of the site.

I would also like to show my appreciation to my class mate Sami M'chala who supported and motivated me wherever I had any doubts, not only with the java bit of the code, but also with the report.

Table of Contents

Abstract	i
Attestation	ii
Acknowledgements	iii
Table of Contents	iv
List of Figures	vi
1 Introduction	1
1.2 Scope and Objectives	4
1.3 Achievements	4
1.4 Overview of Dissertation	5
2 State-of-The-Art	6
2.1 Existing components of java3D:	6
2.2 Existing systems:	6
2.2.1 jMonkeyEngine	6
2.2.2 Java3D Editor	8
2.3 Conclusion	9
3 Requirements: Jumping into the 3D World	10
3.1 Problem definition	10
3.1.1 Use case diagram	13
3.1.1.1 Use case 1: creating .3d files	13
3.1.1.2 Use case 2: open saved .3d file <i>Related requirement: 1B</i>	14
3.1.1.3 Use case 3: Adding an object into the scene graph	14
3.1.1.4 Use case 4: Making changes to the added object	14
3.1.1.5 Use case 5: Undo	14
3.1.1.6 Use case 6: Reset	14
3.1.1.7 Use case 7: Show source <i>Related requirement</i>	14
3.1.1.8 Use case 8: Save project <i>Related requirement</i>	14
3.2 Software prototyping	15
3.3 Expected Behavior	18
4 System Design Description	19
4.1 Static Design Model	19
4.1.1 Class descriptions and responsibilities	20
4.1.1.1 <i>3DFrame</i>	20
4.1.1.2 <i>_basic3DWorld</i>	21
4.1.1.3 <i>_3DWorldHandler</i>	24

4.1.1.4	<i>Mover</i>	26
4.1.1.5	<i>_tg</i>	28
4.1.1.6	<i>SourceCreator</i>	29
4.2	Dynamic Design Model	30
4.2.1.1	<i>Sequence diagram for use case 1: creating a new project</i>	30
4.2.1.2	<i>Sequence diagram for use case 2: loading projects</i>	31
4.2.1.3	<i>Sequence diagram for use case 3: adding an object in the scene graph</i> ..	31
4.2.1.4	<i>Sequence diagram for use case 4: editing the added object</i>	32
4.2.1.5	<i>Sequence diagram for use case 4: Undo</i>	33
4.2.1.6	<i>Sequence diagram for use case 6: Reset</i>	34
4.2.1.7	<i>Sequence diagram for use case 7: Show source</i>	34
4.2.1.8	<i>Sequence diagram for use case 8: Save project</i>	35
5	Implementation.....	36
5.1	Program flow	36
5.2	Implementing the code.....	37
5.2.1	The home screen	37
5.2.2	Adding objects in newly created 3D world.....	38
5.2.3	Editing added object into the 3D World.....	40
5.2.4	Undo	44
5.2.5	View Source.....	45

List of Figures

Figure 1: 3D Coordinate system [2].....	2
Figure 2: java3D scene graph [3].....	2
Figure 3: JME scene composer in action	7
Figure 4: Screenshot: java3DEditor	9
Figure 5: Scene graph	10
Figure 6: Use case diagram.....	13
Figure 7: Prototype I.....	15
Figure 8: GUI Prototype II.....	16
Figure 9: Class diagam.....	19
Figure 10: Use case diagram.....	20
Figure 11 Class diagam 2.....	21
Figure 12 showing pick ray and functioning of Picking	24
Figure 13 Class diagam 3.....	25
Figure 14 Class Diagram 4	26
Figure 15 Rotaion around all three axes	27
Figure 16 Class diagam 5.....	28
Figure 17 Class diagam 6	29
Figure 18 Sequence diagram 1	30
Figure 19 Sequence diagram 2.....	31
Figure 20 Sequence diagram 3.....	31
Figure 21 Sequence diagram 4.....	32
Figure 22 Sequence diagram 5.....	33
Figure 23 Sequence diagram 6.....	34
Figure 24 Sequence diagram 7.....	34
Figure 25 Sequence diagram 8.....	35
Figure 26 Program Flow	36
Figure 27 Screenshot: homescreen	37
Figure 28 Screenshot: showing a newly opened project.....	38
Figure 29 Screenshot: dialogBox to add a box into scene graph	38
Figure 30 A box with and without an outline.....	39
Figure 31 Example project in scene builder.....	40
Figure 32 Moving: Position 1 of sphere.....	42
Figure 33 Moving: location of the sphere after being moved.....	43
Figure 34 Undo: it is all there	44

Figure 35: After calling undo() for first time	45
Figure 36 Undo: the state of scene graph after performing undo() twice	45
Figure 37 View Source: showing methods calls made to _basic3DWorld for current scene graph.....	46
Figure 38 Source: After one of the sphere has been removed.....	46
Figure 39 Example of how the code is shown to the user.....	47
Figure 40 .3ds Model added is horizontal to X axes.....	48
Figure 41 Out of proportion models	49
Figure 42 Models processed in order to make them proportionate	Error! Bookmark not defined.
Figure 43: Screenshot 3D world with external models.....	50
Figure 44 .3d file example	50
Figure 45: User Manual dialog box	68

1 Introduction

Java3D Scene Engine is a GUI application designed in java, which enables users to create java3D scene graphs, and make changes to it. A *scene graph* is used to create virtual universe in java3D API, it is made up of different 3D objects such as box, sphere, etc. The scene graph also defines the characteristics of objects such as geometry, appearance, light.

In this introductory section, there will be four subsections- *background and context*, and then the overall *scope, objective*, and some of the *assumptions* that have been made in producing the design of the application, and then the things that have been achieved so far, with an *overview* of the dissertation.

The motivation behind the project came from the fact, that to design any java3d game or application, a lot of time is consumed in designing of scene graphs. This time spent on design is even greater when the developer is new to java3D, therefore, this project was developed in order to minimize the time that is spent in the designing of a 3D application, also giving users a chance to learn how java3D works. The users will be shown the live changes that are being made on the scene graph and they can also see the code behind it at the same time. Suppose, if user adds a box into the scene graph, and user wants to see what change has been made into the code for that, they can see it by clicking select 'view source' option from the menu.

Whenever an object will be added into the scene graph, it will be added into the *3D coordinate system*, every point in 3D coordinate system has three parameters associated with it. Suppose there is a point, say $x [1, 2, -3]$. The first number represents the location of the object along the X-axis, the second number represents location of the object along Y-axis, and the third number represents location of the object along Z-axis. These numbers represent where the point will be placed in 3D coordinate system. The numbers are generally called as, X-coordinate, Y-coordinate and Z-coordinate.

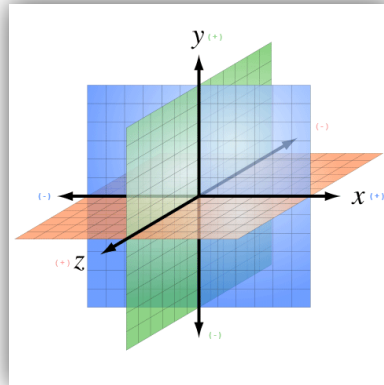


Figure 1: 3D Coordinate system [2]

Every object which will be added in the scene graph will have these coordinate locations. Whenever user chooses to add something in the scene graph, user will be asked for these coordinates. If the coordinates entered by the user are out of the bounds of the rendering area, the user will be notified.

1.1 Background and Context

Java3D scene engine, as the name suggests uses java3D application programming interface developed by Sun. The objects that will be added in java3D are systematized in a parent-child relationship tree structure which is contained in the java3D scene graph, which consists of nodes, and these nodes are made up of java3D objects.

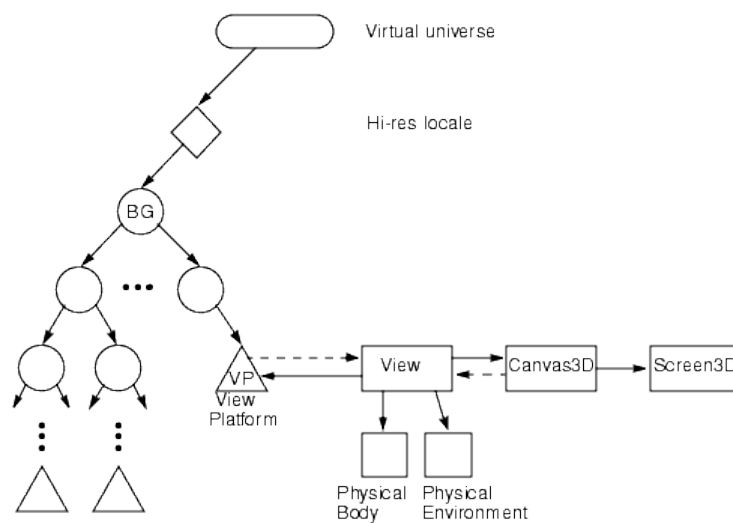


Figure 2: java3D scene graph [3]

The figure above depicts how a scene graph with one shape and behavior looks like; to begin with, every java3D scene graph has two main groups of nodes branching from the locale node. A java3D application can contain one or more VirtualUniverse objects, these VirtualUniverse objects provide grounding for other scene graphs, because in order to be visible in the scene graph, objects need to be added to this object.

The node which is immediately attached to the VirtualUniverse is locale; it defines the origin and coordinates for the branch graphs which are attached to it. The next big thing after locale node is the BranchGraph node; it is single, most important node in any scene graph. It acts as a root for sub-graph which is also known as branch graph. The prime reasons why BranchGraph is so important is because, only this node can be attached to the locale. If user wants to delete any shape from the scene graph, then the shape cannot be directly removed from the scene graph, the BranchGraph to which the shape has been attached needs to be removed in order to remove the shape.

As described earlier, in order to make anything visible, it needs to be added to locale, the programmer does not necessarily have to attach it to locale every time, they can also attach an object to branch graphs which are attached to locale, so effectively they are attaching an object to the locale.

One of the special characteristic of the BranchGroup is all its children nodes will acquire that behavior added to it. Suppose, there is a branch group having four sub graphs (a box, a sphere, a cone, and a cylinder), and if we add the rotation behavior to the topmost branch group of these nodes, then all the four shapes will start rotating. This characteristic of java3D is very useful when defining complex animations such as, movement of arm in human body, or turning of a car. Imagine a car has been added to the scene graph controlling a car, when user presses a key to turn left, the whole car should move, not just the tyres of the car, hence in this case the turning behavior will be added to the node containing all the branch groups of car.

1.2 Scope and Objectives

The idea was to design a user friendly Graphical User Interface (GUI) which will help programmers as well as non-programmers in designing scene graphs in java3D world. This will not only be used in rapid development of 3D applications but will also help programmers, who want to learn java3D. To provide the learning functionality, this application has provision to show users the code generated from the application, highlighting the changes made with every step they take while using the application.

All the requirements in brief are discussed below;

- *Add primitive* shapes into the scene graph.
- *Move and animate* added shapes.
- *Add external 3D models* into the scene graph.
- *Move and animate* the external models.
- Create new scene graphs so they can be designed from scratch.
- Save the scene graphs so they can be used and ported to other applications and websites.
- Load the scene graphs in the application so they can be edited.
- Give users a function to undo the last change made in the scene graph.
- A reset button which will reset the scene graph showing only the axes and no other objects.

Once the users create the scene graphs, they can save it in .3d file and they can use the .3d files in games, or on websites using applets. To load the scene graphs into their game the users will be provided with a loader which can be used with any java3d program, this loader will load the custom project files created from the scene engine.

1.3 Achievements

The required system has been successfully built and the scene engine can do almost all the things it was expected to do. For the purposes of saving the projects, we create a custom file format .3D which saves the information about all the objects that are present in the scene graph. When we load the .3d files, all the objects which are present in the .3d file are added to the scene graph as nodes. So, every object in the .3d file is treated as a node. Along with saving and loading, the project can do all of the following things effectively.

- Undo: if some unwanted shapes or textures are added to the scene graph, users can undo those unwanted changes using this functionality.
- Reset: All the scene graph nodes are removed when this function is used, and a blank project is shown to the user with 3D coordinate system, and a grid which allow users figure out the coordinate system more effectively.
- Adding objects: Users can add primitive shapes in java as well as the external models into the scene graph. While adding these objects users can give their preferences to the object, such as, colour, texture, and coordinates, of the object or users can choose to add an object with default parameters.
- Editing objects: Users can change the properties of the object such as their position, users can rotate the objects around all the three axes, users can also change the size of the object using scaling.

1.4 Overview of Dissertation

The dissertation contains the total of 7 chapters including conclusion. In the next chapter ‘state of the art’ the existing 3d world building system available for java3d will be discussed, along with their advantages and disadvantages, and we will also talk about how some of the features of these applications could be improved.

The next chapter will be ‘requirements gathering’ where the crucial features of the application will be discussed. This chapter will also contain the problem definition and the use case diagram for the system. The software prototyping used during the design phase will also be discussed, and finally we will talk about expected behavior of the application which will explain the typical program flow.

In the next chapter, ‘System Design Description’ we will discuss the more technical bit of the system with UML diagrams such as class diagrams, and sequence diagrams.

The next chapter ‘Implementation’ will contain information about how the solution was implemented and the problems that were faced during the development.

In ‘Testing’ we will talk about the different testing that was carried out during and after the development.

2 State-of-The-Art

In this chapter we will discuss the existing system, that is the programming model of java3D and then the systems which enable users to create similar 3D worlds will be discussed.

2.1 Existing components of java3D:

If one wishes to design 3D shapes in java3d, and add it to the scene graph, they have to write a program to do it. For this the users should know how to program in java and they will have to learn java3D API, and see how the scene graph works in reality.

Even to rotate an added shape or apply any transformations to shapes, users have to define their own behaviour classes which will listen to the user input and implement methods like initialize(), processStimulus() etc. which is clumsy as well as time consuming.

If users want to change or move an object they have to create methods in their behaviour classes to achieve this. To add textures to any shape, users have to create their own appearance objects and point them to texture which are linked to physical image on media.

To save a project in java3d, users have to create different java files for which many components of the code may be changed.

2.2 Existing systems:

2.2.1 jMonkeyEngine

jMonkeyEngine is a game engine designed in java which uses LWJGL (Lightweight Java Game Library) as a default renderer. It is an open source project designed by the JME Core team. First released in 2004, JME team has released three versions since then; the latest version jMonkeyEngine 3.0 was released in 2011. The jMonkeyEngine has its own 3D APIs known as *jMonkeyEngine 3D Scene Graph API*, so the bottom line here is jMonkeyEngine does not use java3D. [4]

Java3D API has some advantage over jMonkeyEngine 3D API, like java3D is compatible with OpenGL as well as Direct3D, jMonkeyEngine is compatible only with OpenGL. Threading model of java3D API is much more sophisticated than the JME 3D API. [5]

jMonkeyEngine SDK SceneComposer: One of the component in jMonkeyEngine lets users create a 3D world, create a land, generate terrains, etc. This component has many predefined 3D world scenes like, a soccer ground, garden, race track, etc. which users can use and edit so that they can use these scenes for their game. Users can also create their own scenes which can be saved. Even though SceneComposer lets you create scenes in 3d, it is a work in progress, and it has few bugs in it.

The aim behind java3D scene builder was to design user friendly GUI experience similar to jMonkeyEngine which will help users build portable scene graphs in java3D API supplied by sun. One of the disadvantages of JME is non-programmers cannot use it, because the users of this software need to know how to code in java and how any java program works.

The Java3D scene builder can be used both by programmers, non-programmers as well as the users who want to learn java3d API by sun. Programmers and java3D users can see the code, which will be modified with every little change that is being made on the scene graph. The non-programmers can choose to hide the code if they want to. All the users of the software will be shown the live scene graph that is being operated, so the users will be well aware of the things going on with their 3D world.

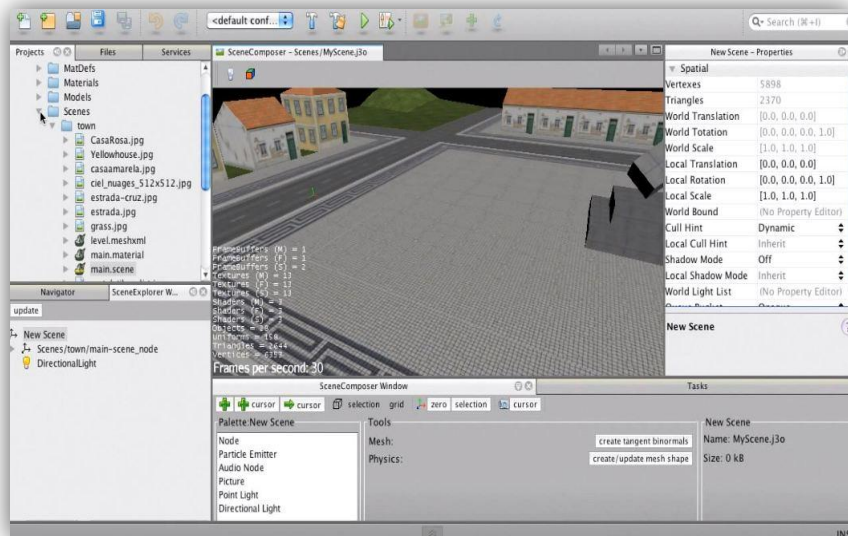


Figure 3: JME scene composer in action

This is a screen shot taken from jMonkeyEngine scene composer, it shows one of the default saved scenes created in JME. Even though it looks impressive to load a scene in JME takes a lot of efforts, and for beginners and non-programmers it is not at all easy to use.

Just to create a simple town scene that you see in the screenshot above, the users have to go through several steps, the users are shown as you see a list of all the projects in the right hand column, they first have to go to their desired scene. This is fine for end users who are familiar with the structure of eclipse IDE but for non-programmers it's a bit clumsy.

2.2.2 Java3D Editor

J3D Editor is one more similar type of GUI based application designed in java and uses java3D. It is used for automatic generation of code; some of its features are highlighted below:

- 3d models supplied are not enough, and not properly processed, they are either not in proportion with the other things in 3D world, or they are placed on screen without rotating.
- GUI is not user friendly; it does not allow users to move the added shapes. In our system we will give users a chance to select any object, and move it using keyboard. Whenever users add any shape onto the scene graph, they are shown a number of dialog boxes one by one, which can be a bit irritating at times, in our system we can show user only one dialog box, users can add all the information onto the dialog box.
- The users can change the colour of the shape or the textures of the shapes after adding.
- The users can see what shape has been selected as the selected shape is highlighted.
- It does not have option to undo or redo the previous changes made onto the scene graph.
- Users can clone their objects in the scene graph.
- The code generation of the java3D Editor is a good feature, and it lets user generate a java file, which can be used in other application.
- Users can reset the viewing platform of the application, it is useful when the screen is rotated by user using mouse, and the users are somehow not able to rotate it back to its original position.

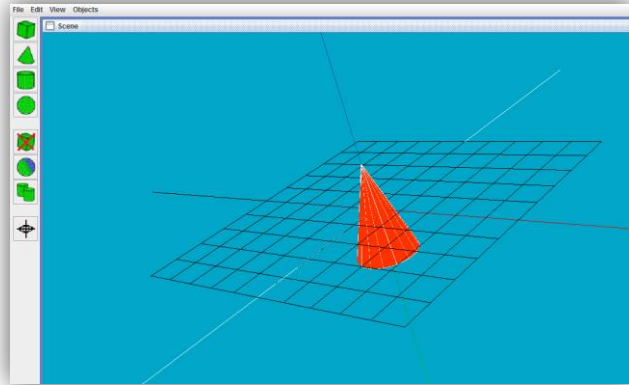


Figure 4: Screenshot: java3DEditor

2.3 Conclusion

In Java3D scene builder we will try to improve these entire drawbacks and add many more of our own features such as undo action and redo action. We will try to keep the GUI as simple as possible, the saving and loading of the program will be very simple. Users will also be allowed to open previously save scene and add their own things to it.

3 Requirements: Jumping into the 3D World

This chapter will describe the requirements of the software; along with characterization of the boundary of the system. This chapter will also include the use case diagram which will be constructed using the requirement stated in section 3.1.

3.1 Problem definition

The graphical user interface (GUI) of the application will be designed in java, with the java3D API to allow users create a java3D scene graph which will be portable and can be used in java3D applications. We have already seen what a scene graph in java3D is and how it can be used in various applications in the introduction, so we will directly start defining the problem that we have in our hands.

System Requirements:

- 1- The fundamental requirement of this application is the ability to construct scene graph, every project that user loads in the system will consists of light and a background. The objects added by the users will be below these nodes and these objects will be affected by both light and the background of the 3D world. The information about the background and lights in the scene graph is stored in the locale node, which is explained in the introduction.

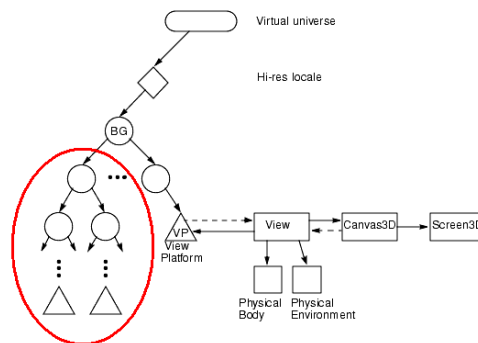


Figure 5: Scene graph

So, to sum up, what the system is expected to build is the circled part in the above scene graph. Along with this, the information of the directional light in the scene graph will also be saved in the .3d file.

- 2- When user starts the system they will be shown a home screen which will have three options, *new*, *open*, *exit*, here users can choose to create a completely new project from the scratch.
 - **2A-** when users will click on new button they should see a black screen with X, Y and Z coordinate and a grid which will show the users how much is 1 unit.
 - **2B-** Users should also be able to open previously saved projects from the /projects directory, once opened users should be able to make changes to the objects presents in these projects. If users open an undesirable project by mistake they should be given a chance to load project which they want to load, for this the software should have an open button, which will show all the projects that are currently present in the projects directory.
- 3- Adding an object into the scene graph.
 - **3A-** The users of the system should be able to add primitive java shapes as well as third party 3D models into the scene graph.
 - **3B-** If user chooses 3D objects, then there should be a provision to add image textures to the primitive shapes.
- 4- Users should be allowed to make changes to the added objects-
 - **4A-** In order to make changes to a particular shape in the scene graph, the users should be able to select that shape using mouse. Java3D has 'picking' libraries which can be used for this purpose. We will need to identify which object was clicked.
 - **4B-** User should be able to *move* the added objects, be it *external models* or *primitive shapes* in the scene graph.
 - **4C-** User should be able to *rotate* the objects.
- 5- User should be able to revert the latest change they made on the scene graph using undo button, this can be easily achieved using the java library **java.util.Stack**, in this particular case, we will save the root scene graph object onto the stack, so whatever changes user makes, they all will be reflected after doing the undo operation. If user loads some project into the scene graph, they should be able to remove every object that has been added in the scene graph using undo.

- 6- Reset function to reset the scene graph to which will show default axes. Reset can be used to remove all the objects that have been added to the system.
- 7- As stated in the introduction section of this paper, this software is mainly useful to student programmers who want to learn java3D, to achieve this bit; the software should have a provision to show the code of the live scene graph on demand, to the users. This code will need to be modified for every little change that is being made on the scene graph.
- 8- The scene graphs will be saved in a physical file on the hard drive, they will be saved in the form of text, but just to avoid confusion between other text files, the file extension given to the scene graphs will be .3d. So, if we have a scene graph named *FirstWorld*, the physical file for this project on the hard disk drive will be, *FirstWorld.3d*. User should be able to save and load the scene graphs in .3d files, create new scene graph projects in .3d files.

3.1.1 Use case diagram

Let us now design a use case diagram for the problem considering the above requirements. After doing the abrupt study of the working of the application, it was realized that there is going to be only one actor, the user of the system in all the use cases, hence we are not going to show the actor in the description.

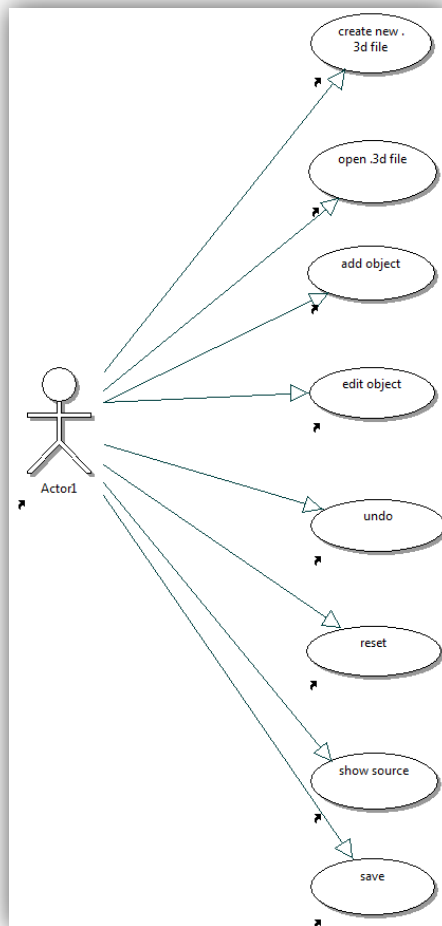


Figure 6: Use case diagram

Let us now describe each and every use case associating it with its related requirement.

3.1.1.1 Use case 1: creating .3d files

Related requirement: 1A

Description: Whenever user runs the program for the first time, an option to create a new project might be selected; user can create a new project even from the 3D editor window.

3.1.1.2 Use case 2: open saved .3d file

Related requirement: 1B

Description: Whenever user runs the program for the first time, an option to open previously saved project might be selected, user can also open project from the 3D editor window.

3.1.1.3 Use case 3: Adding an object into the scene graph.

Related requirement: 2A, 2B

Description: Once the project has been opened in 3D editor window, users may add a 3D object into the scene graph. This 3D object can be a primitive shape with an image texture, or a 3D model.

3.1.1.4 Use case 4: Making changes to the added object

Related requirement: 3A, 3B, 3C.

Description: After adding the 3D objects in the 3D editor, users may choose to change the characteristics of the object. Users can also make changes to the objects which are already present in the scene graph.

3.1.1.5 Use case 5: Undo

Related requirement: 4

Description: If user makes some change to an object, and the change is not desirable, user can undo that change.

3.1.1.6 Use case 6: Reset

Related requirement: 5

Description: Sometimes users may not be happy with whatever they have created; in that case, users may choose to reset the scene graph.

3.1.1.7 Use case 7: Show source

Related requirement: 6

Description: If a user wants to check the source of the current scene graph, they can check it using the menu button 'show source'.

3.1.1.8 Use case 8: Save project

Related requirement: 7

Description: Users should be allowed to save every project in the physical media, so that, the projects can be used later on.

3.2 Software prototyping

Prototyping of modern software is a practise followed by many standard organizations to get a valuable feedback from the potential users of the software early in the project; in the early prototyping stage of software engineering only very few aspects of the systems are considered. Sometimes the prototypes of the software are made on the paper, just to analyse how the system will look like after being developed. This kind of prototype is useful when GUI based software has to be developed.

Now, in software prototyping we will be talking about two different types of prototypes for this application, let us have a look at them one by one.

3.1.2 GUI Prototyping

When it comes to prototyping of software we have different methods, we first will start with the GUI prototyping where we will talk about the two GUI designs which were considered during the beginning stages of the project.

3.1.2.1 GUI Prototype I

In this GUI model, we have a separate window (JFrame) for every feature that we have in the scene engine.

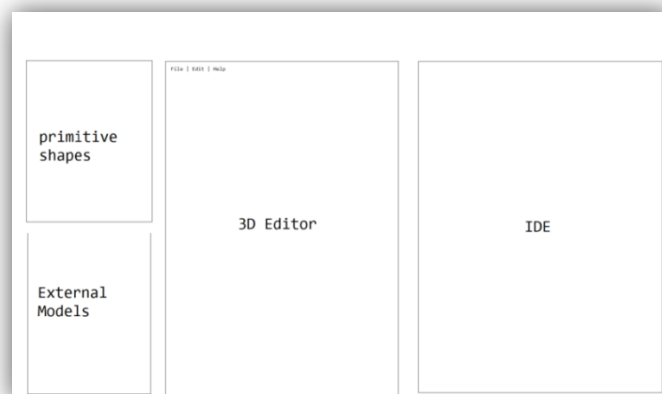


Figure 7: Prototype I

The options to create a new file, load existing file or exit is in the 3D Editor window, users can use JButtons in the primitive shapes window to add shapes onto the scene graph, same with external models. The live scene graph was shown on the 3D Editor window.

3.1.2.2 GUI Prototype II

Instead of having different JFrames for every feature this prototype includes all features on different JPanels in one JFrame.

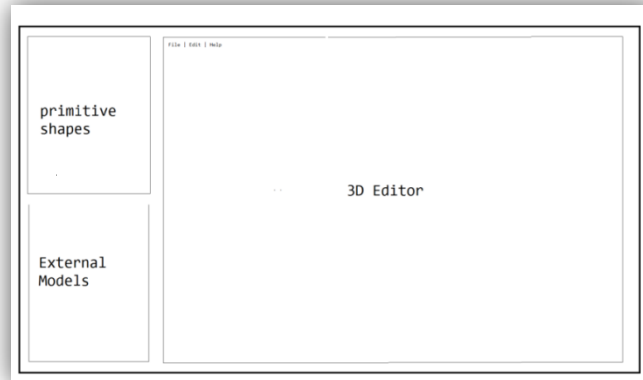


Figure 8: GUI Prototype II

Since there is only one window in this design, the scene engine is a lot easier to manage, there were many other advantages of prototype II over its predecessor, like, it's a lot easier if you want to run the application in full screen, if user want to change the size of the 3D editor window, they can do it easily.

3.1.3 Architectural prototyping:

Now from the GUI prototype of the software, let us talk about the core architectural design of the application, i.e. the actual working of the system.

While working on the early prototype, few issues regarding the architecture of the system were brought up. One of the issues was, how the user triggered changes made to scene graph (like addition of a box to scene graph) will be handled by the system? The following are the two possible solutions using which this issue could have been solved.

3.1.3.1 Compile and load architecture

Whenever user adds a box into the scene graph, we open the main 3D java file *_basic3DWorld.java* we make changes to that file in the method *public void createSceneGraph()*, this change would be nothing but adding an instruction *addBox(..., ..., ..., ...)*; with some parameters describing the box, after doing the file handling, we compile the *_basic3DWorld.java* file, and all other files using this class, and we load it into the memory. It also involved disposing the current window, and creating another JFrame window in order to load the newly created scene graph.

Even though this method works perfectly and was used in the early prototyping of the software, it is a time consuming method, after adding any shape into the 3D world it used to take around 5-6 seconds to compile and load the files back into the memory. Apart from the fact that it was time consuming, the loading of the byte code back into the memory had its own issues such as, the class loader provided by java is buggy sometime it does not load the byte code into the memory for some reason.

3.1.3.2 Changing the live scene graph on the GO

This method of tackling the above discussed problem is straight forward and less time consuming, all we do in this method, is remove the main branch graph from the root node, make the necessary changes to the scene graph like adding of a shape, or undoing previous change, and add the branch graph back to the SimpleUniverse object. This method was not implemented initially, but now the entire software including the loading and saving of projects, uses this method to make changes on scene graph.

3.3 Expected Flow

Before jumping into the object oriented UML diagrams, let us discuss how the software is expected to behave, right from when the user starts the application till it has been closed.

When user runs the software for the first time following sequence of events will happen.

- Users are shown a home screen which has options to-
 - o Create a new project
 - o Open saved projects
 - o Exit the program
- If user selects any one of the first two options the 3D Editor window will be shown to the user, where most of the time of the software will be spent. Here users will be given an option to do following things:
 - o Add object
 - o Edit object
 - o Undo last operation
 - o Reset 3D Frame
 - o Open another project
 - o Create another project
 - o Save project
 - o Exit the program

4 System Design Description

Now that our use case diagram is ready, we can begin with the design phase for our project, here we will determine the essential components of our system and how they relate with each other. The preceding part of this chapter will be about the static design model which will contain class diagram and, dynamic design model which consist of sequence diagrams with their associated use cases and state chart diagrams which will tell us more about the states of the project.

4.1 Static Design Model

The figure below shows class diagram illustrating the static design model for scene engine system. The classes are shown with their attributes and methods, along with different relationship cardinalities between the classes. It should be noted that all the UML diagrams in this document follow standard UML 2.0 syntax.

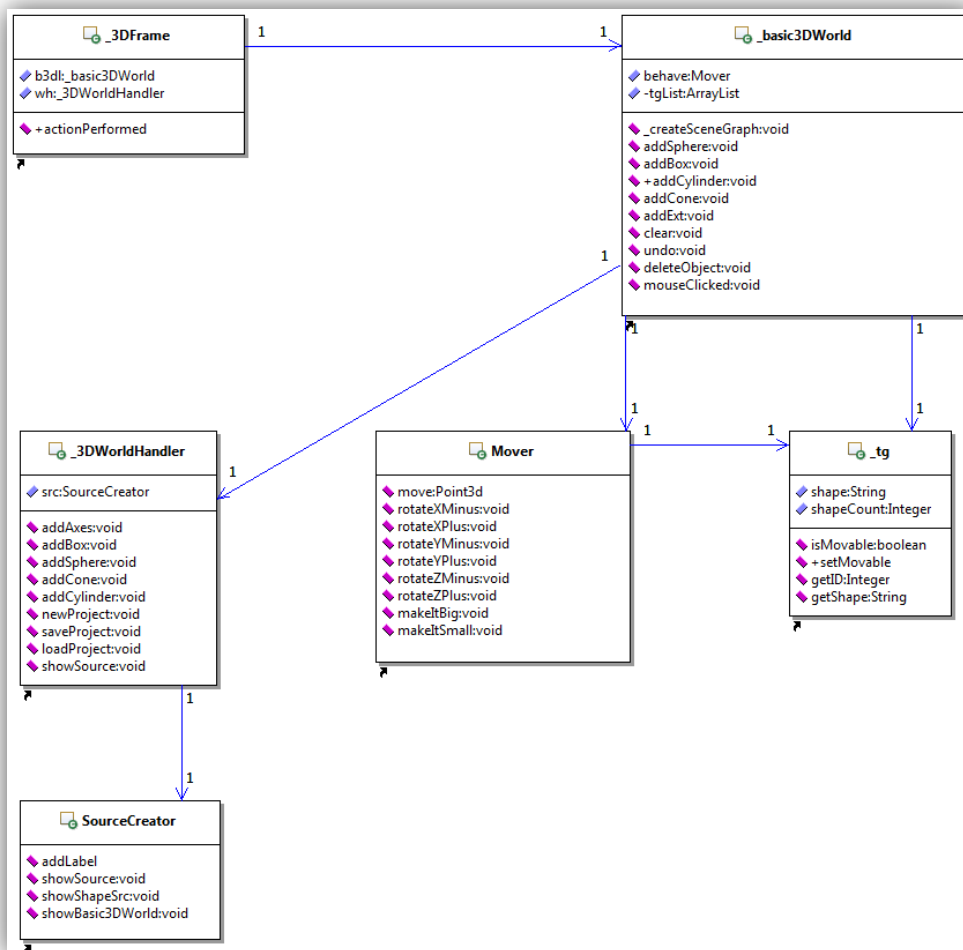


Figure 9: Class diagram

4.1.1 Class descriptions and responsibilities

4.1.1.1 **3DFrame**: This class takes care of the GUI of the whole project, right from displaying all buttons on the main frame to the file, edit, view, and help menus, this class handles it all. This class is also an entry point to the whole program, because home screen is shown to the user in this class. This class extends JFrame which uses the BorderLayout supplied by java swing; this class also makes use of the GraphicEnvironment class to set the JFrame in maximized state. So, it does not matter on what system the application runs, it will always be in the full screen.

Some of the main responsibilities of this class are as follows:

- 1- Generating GUI for the user.
- 2- Showing home screen to the user.
- 3- Listen to all the user inputs on buttons and menu items.

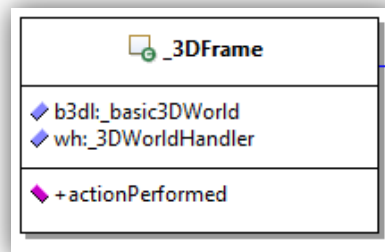


Figure 10: Use case diagram

`_3DFrame` class is associated with `_basic3DWorld`, which represents the whole 3D world shown, this is then added to `_3DFrame` [`_3DFrame` extends `JFrame`]. Another association in this class is `_3DWorldHandler`, this is mostly used in the `actionPerformed()` method, all the calls made by user interactions such as, add box, add sphere, etc, are first gone to `_3DWorldHandler`, which in turn calls method from `_basic3DWorld`.

This class contains only a couple of methods and a constructor, one of the method is **GUIBuilder()** which does all the work of building the GUI, which includes adding buttons, setting their size as well as their location, adding `ActionListeners` to them. All these buttons are later on added to a panel, and that panel is then added to the `JFrame` using `BorderLayout`.

Another important method of this class, **actionPerformed()** which has been defined because of the interface which we have implemented. This method listens to the user interactions not only the buttons but also the menu items. An appropriate

method from `_3DWorldHandler` or `_basic3DWorld` is called when one of the user events takes place.

4.1.1.2 ***_basic3DWorld***: First used in `_3DFrame` this class is the backbone of the whole project, the main activity has to perform is rendering the whole scene graph onto the `_3DFrame`, `_basic3DWorld` manages all the scene graphs which are being added by the user from `_3DFrame`.

```
public class _basic_3DWorld extends JPanel implements MouseListener
```

So, as it can be seen, this class extends `JPanel`, and implements `MouseListener`, all the 3D world is rendered onto the `JPanel`, so what `_3DFrame` shows to the user is this `JPanel`. This class also handles the stack which contains all the changes made by the user, and it is used by the undo function to retrieve data and show it onto the screen.

The main responsibilities of this class are as follows:

- 1- Create scene graph for user interactions
- 2- Provide methods to add shapes and external models onto scene graph, all these objects are added by modifying the live scene graph.
- 3- Providing methods to undo and clear current scene graph.
- 4- Listen to user mouse inputs for selecting a particular object from the scene graph.
- 5- Deleting an object from the scene graph.

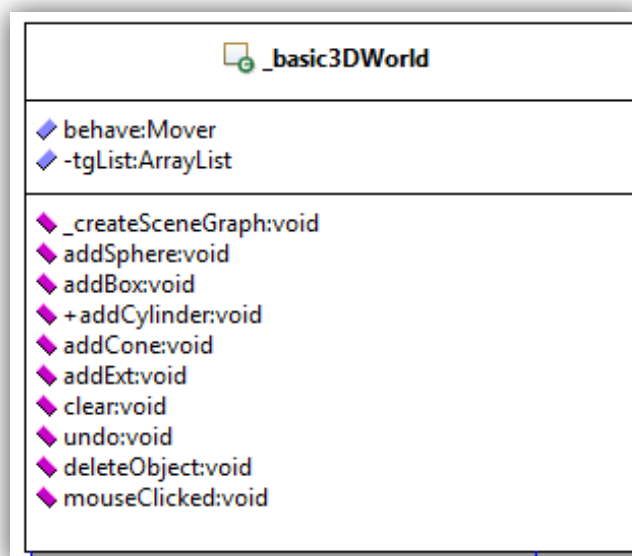


Figure 11 Class diagram 2

The important attributes used by `_basic3DWorld` are `Mover` and `_tgList`, we will be discussing those in the next few pages, but just to get you started, `Mover` is the behavior which handles moving of any object, and `_tgList` is just an `ArrayList` of `_tg` class objects. `_tg` class extends `TransformGroup` to add some extra functionality to it

The constructor of this class does the job of setting up the `JPanel` and adding `MouseListeners`, creating scene graphs, and initiating user location. Besides the constructor, this class contains many important methods which are discussed below.

```
private void _createSceneGraph()
```

This method creates the basic scene graph for the user it contains only the axes, as other objects are added later on by the user. This method contains call to the `addBackground()` method which is pretty self-explanatory it adds a background to the scene graph, the next method called is `lightScene`, which adds all the necessary lights to the scene graph, after `lightScene`, the scene graph is compiled.

Scene graph compilation is necessary as it gets rid of all the unnecessary `BranchGraphs` and `TransformGroups`, if there are any.

```
public void addSphere(float xCoor, float yCoor, float zCoor, float radius, String color, boolean cth)
```

This method adds a sphere onto the live scene graph, for that the arguments provided are as follow.

`xCoor`: X coordinate of the sphere.

`yCoor`: Y coordinate of the sphere.

`zCoor`: Z coordinate of the sphere.

`Cth`: Tells us whether or not we have to show an outline for the added sphere.

The most valuable part of this method, and all the other methods which add an object onto the scene graph is, they all detach the live scene graph, add the shape and load the scene graph back again, as we add an object directly onto the scene graph, the time required to show the object is pretty less.

The other methods which add shapes onto scene graph are similar and self-explanatory.

```
public void clear()
```

Consider a scenario where user creates a new project, adds few objects onto the scene graph, and somehow, the user is not happy with the project and decides to create a new project, in such case, all the previous added objects should be removed from the scene graph, and a clear scene graph should be shown. In such situations

this method is called. This method is also called when user presses reset button on `_3DFrame`.

```
public void undo()
```

Consider one more scenario, where user is working on some project, and deletes an object by mistake, in such situations users can call undo method to recover the previous scene graph that they were working on. As mentioned earlier, this method uses the stack which saves the root scene graph branch after every change made onto the scene graph.

```
public void deleteObject()
```

One of the requirements in the problem definition was, users should be able to delete any object present in the scene graph, and this is the method which provides that functionality to the user. Users can select any object they can see on the scene graph and press key 'D' to delete that object from the scene graph. How the selection of the object using mouse will be discussed further.

```
public void addExt(String target, float xCoor, float yCoor, float zCoor)
```

Whenever users want to add an external model into the scene graph, this method is called, it takes the 3ds object to be added from the user and the coordinates of the object. This method processes 3ds model in java3d, so they are in proportion with the other things in the scene graph.

String target which is supplied to this method is the name of the .3ds file stored on physical media, this method then loads the .3ds model into the scene graph with the help of PropManager.java class.

```
public void mouseClicked(MouseEvent e)
```

Even though this method is defined just because this class implements MouseListener, this is one of the most valuable methods in this class. Whenever user selects any object using mouse, this method gets called, it helps identify which object has been clicked, and accordingly the movable flag for that object is set. Picking API supplied by java3d is used in mouseClicked method to identify which object has been clicked. Picking is used in such a way that it projects a ray onto the scene graph from user's viewpoint, it returns the type of the first object that it intersects.

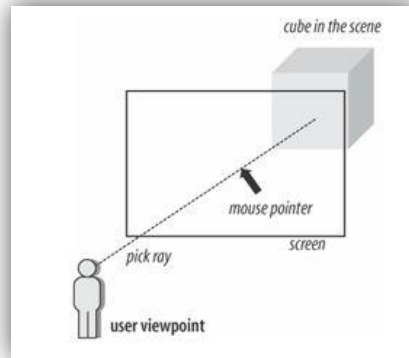


Figure 12 showing pick ray and functioning of Picking

If there are more objects behind the currently visible object, then the object which is visible is picked. Users of this application can use mouse to control the viewing position of the user and selects the objects which were hidden, if they want to.

4.1.1.3 ***_3DWorldHandler***: All the calls to add any primitive shape onto the scene graph are carried out through this class, the main responsibilities of this class are:

- 1- Taking the input from the user describing the characteristics of the shape, by showing a dialog box
- 2- Manage the GUI responsible for showing the dialog box, JTextFields, ComboBoxes, etc which are shown onto the dialog boxes.
- 3- This class also handles the file handling part of the whole application, this class has methods to create a new project, open saved project these methods are called from *_3DFrame*.
- 4- This class adds axes onto the scene graph.
- 5- It converts the current *_tgList* to the labels, which are used by *SourceCreator* class
- 6- *_3DWorldHandler* is associated with *SourceCreator* class, and calls method from *SourceCreator* such as, *addLabel*, and *showSource*.

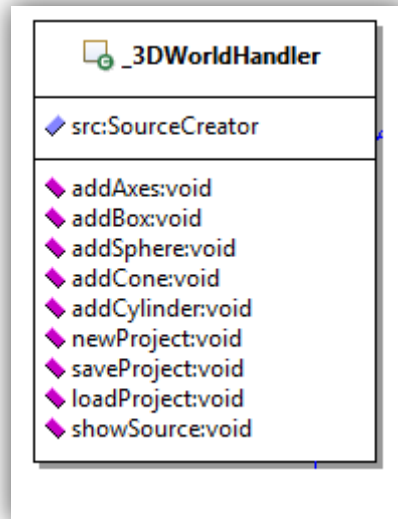


Figure 13 Class diagram 3

Now that we are aware of the activities this class will carry out, let us discuss the methods involved:

public static void addAxes()

Whenever users open a new project, they are shown three axes which represent the 3D Cartesian system of coordinates. This method draws these axes onto the screen.

public void addBox()

This method is called from `_3DFrame` whenever user wants to add a box onto scene graph, it shows a dialog box to the user and takes input from them, then it calls `addBox` method from `_basic3DWorld` to add a box onto scene graph. Other methods to add shapes in this class are similar to this.

public void newProject()

This method is called from `_3DFrame` when a call is made to create a new project, it takes a name of the new project from the user and creates a new file in the default projects directory.

public void loadProject(String filename)

This method is also called from `_3DFrame` class, whenever user presses a button to load previous projects, they are shown a `JFileChooser` in `_3DFrame`, the file name of the user chosen file is then sent to this method. `loadProject()` then opens the file and loads the data into the scene graph.

```
public void saveProject()
```

Saves the opened project into custom .3d file, the filename given to these files is the String stored in 'opened' variable. This method goes through the tgList ArrayList in _basic3DWorld, which is the list of currently present objects in the scene graph, and writes information about these objects into the file. The information stored in file contains the coordinates of the object, the size of the object, the location of the textures, if any are applied, the colour of the object, if textures are not specified, and whether or not the object should be outlined. How the file is saved will be discussed in the next chapter which is the implementation of the system.

4.1.1.4 **Mover**: This class is used by all the _tg class objects throughout the application, it extends abstract class Behavior of java 3D API and has to implement the methods processStimulus and initialization. The behavior provides a foundation for adding user defined actions on the scene graph objects, in the Mover class we have added following behaviours used by every object added into the scene graph.

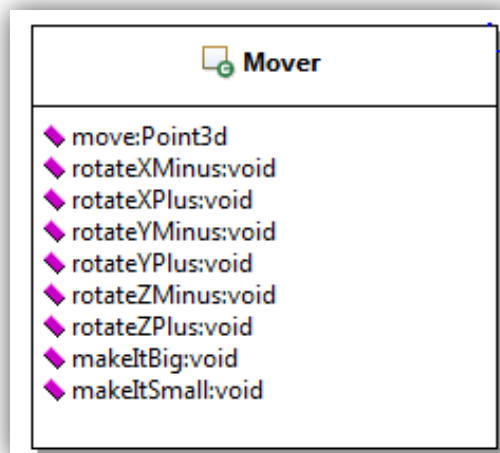


Figure 14 Class Diagram 4

Move the object: Now as I have stated earlier, any object which is visible in the scene graph, can be moved just by clicking onto it, and using the arrow keys. ProcessKeyEvent method calculates in what direction the object will be moving and this calculated value is passed to move method, which does the actual movement of an object on the scene graph using Transform3D. One thing to understand here is, the object can be moved in six different directions, along with regular up, down, left and right, the object can be brought closer to the user in Z axes, (z positive) or away

from the user (z negative). The keys which are used to achieve this are explained below.

Right: X positive

Left: X negative

Up: Y positive

Down: Y negative

Ctrl: Z positive (closer to user's viewpoint)

Shift: Z negative (away from the user.)

Rotate the object: The object can be rotated around all the three axes, in both the directions, so it gives user to turn their object any way they want.

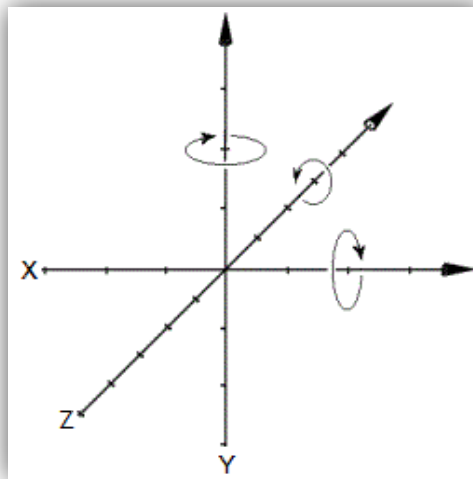


Figure 15 Rotaion around all three axes

The key bindings for rotation are as follows:

Number pad Key 7: Positive rotation around X axis

Number pad Key 8: Negative rotation around X axis

Number pad Key 4: Positive rotation around Y axis

Number pad Key 5: Negative rotation around Y axis

Number pad Key 1: Positive rotation around Z axis

Number pad Key 2: Negative rotation around Z axis

Scaling object: Apart from the regular movement and rotation of the object, this class also implements method to scale the object, it can either make it small or big, users can press key 'Q' to make it big, or 'W' to make it small.

Deleting an object: Users can delete any object in the scene graph which is visible in the scene graph by selecting it using mouse and then pressing 'D' key.

4.1.1.5 **_tg:** As mentioned in the previous sections, **_tg** extends TransformGroup class of java3D; every **_tg** object has a transform and a 3D object associated with it. The 3D object inherits any transform the **_tg** has. Transform3d defines where a particular object will be placed onto the scene graph; it also has information regarding the affine transformations that can be performed on the object. Internally Transform3D is nothing but a 4x4 floating point matrix with double precision.

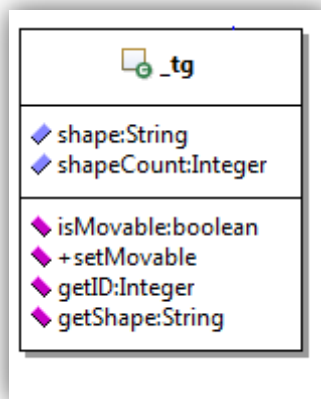


Figure 16 Class diagram 5

The two important attributes of **_tg** are, **shape**, a string which stores which shape it is, and **shapeCount**, which is an integer and acts an ID for every object that is been added. This ID is later on used for picking and identifying every shape.

shapeCount is incremented whenever an object is added into the scene graph.

Another attribute of **_tg** is **movable** this is a Boolean and it determines whether or not we can change the location of a particular object at any given time. The **movable** property of **_tg** is set from **_basic3Dworld** class, precisely when user selects any object using mouse.

4.1.1.6 **SourceCreator:** The last but not the least, this class is used mostly when user wants to see the source code of the projects they are working on, users can select any available project at any time, and see the source code for that.

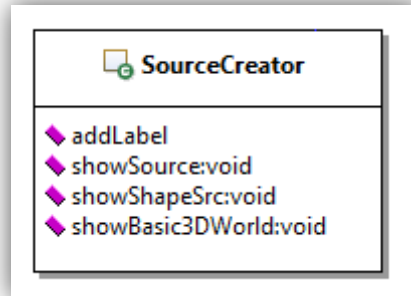


Figure 17 Class diagram 6

Only two methods of this class are public, and are called from `_3DWorldHandler`, and they are, `addLabel`, and `showSource()`. So, whenever user wants to see the source for the project, they can select 'view source' option from the menu, they will be shown a `JFrame` which will contain labels which internally are buttons, these labels represent the calls made to the `_basic3DWorld` to add a particular object. If users want to see the methods upon which the calls are made, then they can click on the method call (as these are nothing but `JButton`). After clicking on this button the content of this `JFrame` will be replaced by `JTextArea` which will display the appropriate method with comments.

[It should be noted that these methods are modified to add comments, and are limited only for educational purpose]. The users can then choose to see the class in which these methods are implemented, and how the building blocks of the 3D world are laid out.

4.2 Dynamic Design Model

The dynamic design model of this system consists of sequence diagrams and state diagrams.

4.2.1 Sequence Diagrams:

Every use case described in problem definition section of this paper, has a UML sequence diagram associated with it.

4.2.1.1 Sequence diagram for use case 1: creating a new project

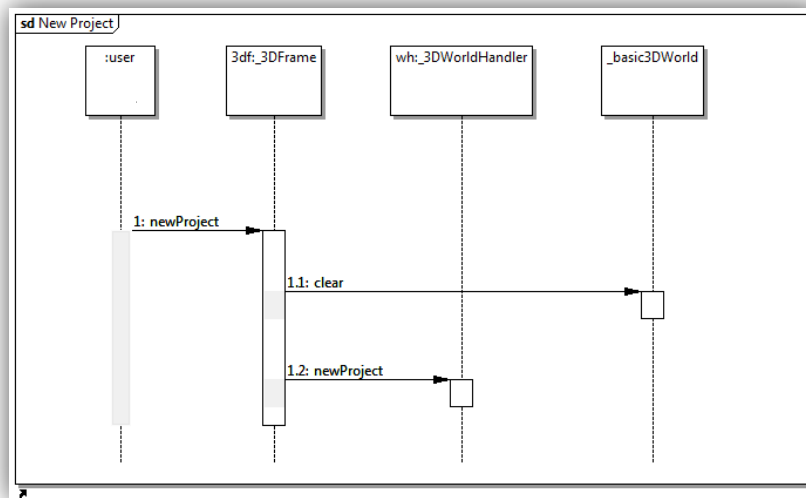


Figure 18 Sequence diagram 1

This sequence diagram shows how user interacts with the system to create a new project, whenever a new project has to be created, we will have to clear anything that is currently in the scene graph. User is just an actor in this case, whenever user interacts with the system, newProject() method in _3DFrame will be called, which clears everything in the scene graph, and then newProject() method from _3DWorldHandler is called which takes care of taking the file name from the user and creating a new file.

4.2.1.2 Sequence diagram for use case 2: loading projects.

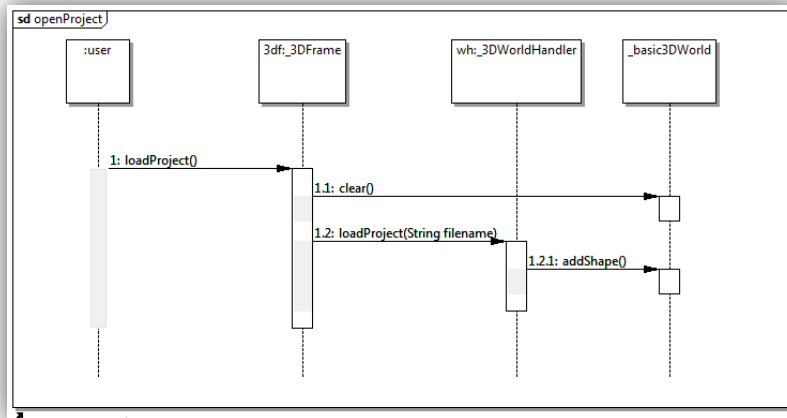


Figure 19 Sequence diagram 2

Just like in the previous sequence diagram, we are calling `clear()` method from `_basic3DWorld` class to clear any previous projects that were in the memory. Then, we show user a `JFileChooser` where users can select the file they want and then that file name is passed to `loadProject()` method in `_3DWorldHandler` class, which opens the file specified by the user and adds appropriate shape in the scene graph. It is to be noted that we do not have any method named `addShape()` in the `_basic3DWorld`, but we have methods such as, `addBox()`, `addSphere`, etc, we have used `addShape()` in the sequence diagram for the sake of simplicity.

4.2.1.3 Sequence diagram for use case 3: adding an object in the scene graph.

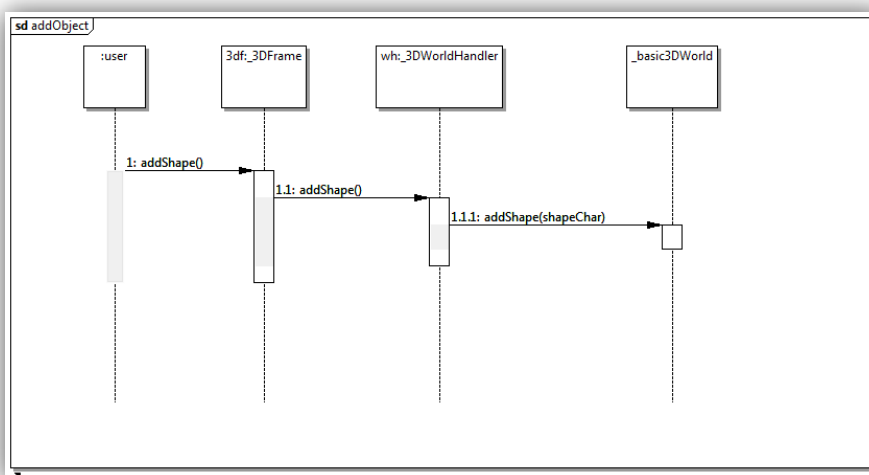


Figure 20 Sequence diagram 3

This use case is straightforward; user presses a button to add a shape onto the scene graph. Then user is shown a dialog box by `_3DWorldHandler` class depending upon the shape. `_3DWorldHandler` then calls `addShape()` method from `_basic3DWorld` passing the parameters accepted from the user.

4.2.1.4 *Sequence diagram for use case 4: editing the added object.*

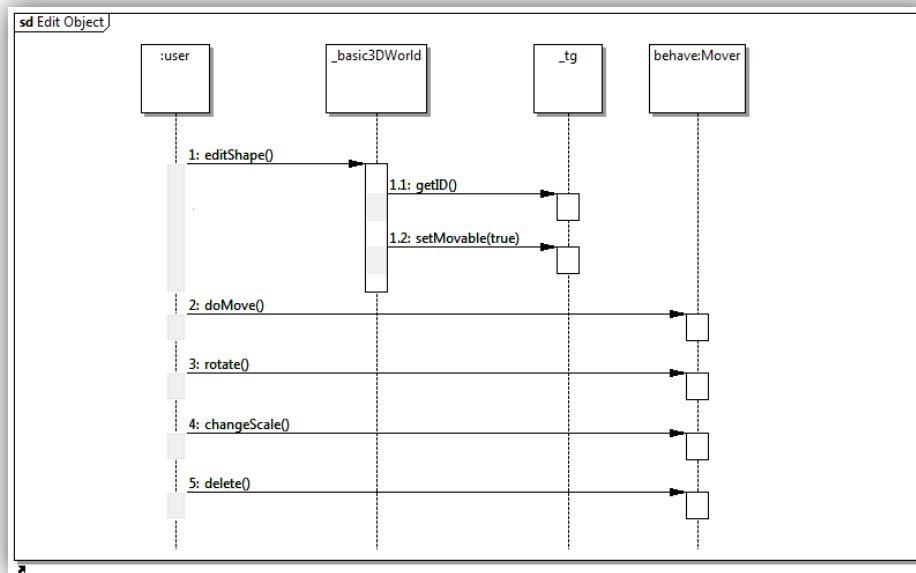


Figure 21 Sequence diagram 4

Users can select any of the visible objects present in the scene graph, and make changes to these objects. Whenever user selects any object using mouse, following events will take place.

- We check if the object user has clicked on is an external object or java3D primitive shape, using Picking classes supplied by java3D.
- If it's a primitive java3D shape, then by checking the IDs set to each shape we identify which shape has been clicked and we set the movable property of that shape as 'true'.

After the shape which is to be moved has been identified, we wait for users to change the attributes of the shape, as it can be seen from the sequence diagram above, users can move shape, users can rotate the shape, they can change the size of the shape, and they can delete the shapes. In this version of scene builder we are only going to implement these features considering the limited time we have. Other features such as adding animations and changing geometry will be added in the future.

To see the key associations with every method in this sequence diagram, please check the appendix for user manual.

4.2.1.5 *Sequence diagram for use case 4: Undo.*

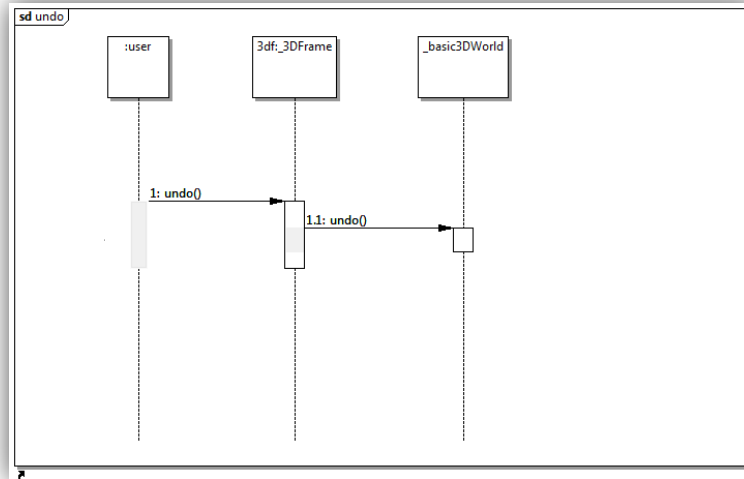


Figure 22 Sequence diagram 5

Users can undo any change they have previously made on the scene graph by pressing undo button on `_3DFrame`. Suppose user adds 3 shapes onto the scene graph, and presses undo the shape which was added latest will be removed, and so on. The `undo()` method in `_basic3DWorld`, pops the top element from the stack, the stack stores previous scene graphs which were modified by the user changes. The stack does not record changes to the shape, i.e. if user rotates a shape, or changes the position of the shape, those changes cannot be retrieved by pressing undo button.

4.2.1.6 Sequence diagram for use case 6: Reset.

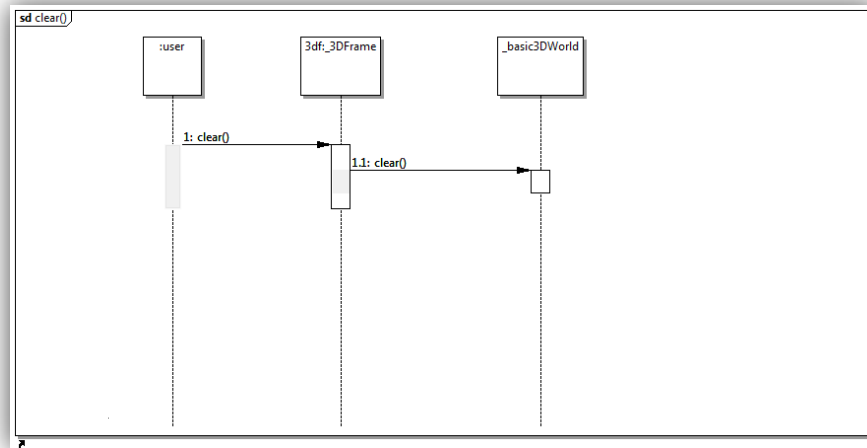


Figure 23 Sequence diagram 6

Users can reset the `_3DFrame` anytime they want, this will remove all the object attached to current scene graph, and will clear the screen. This method will be called from `_3DFrame` by user, and from `_3DFrame` the call will passed to `_basic3DWorld` where the method has been defined.

4.2.1.7 Sequence diagram for use case 7: Show source

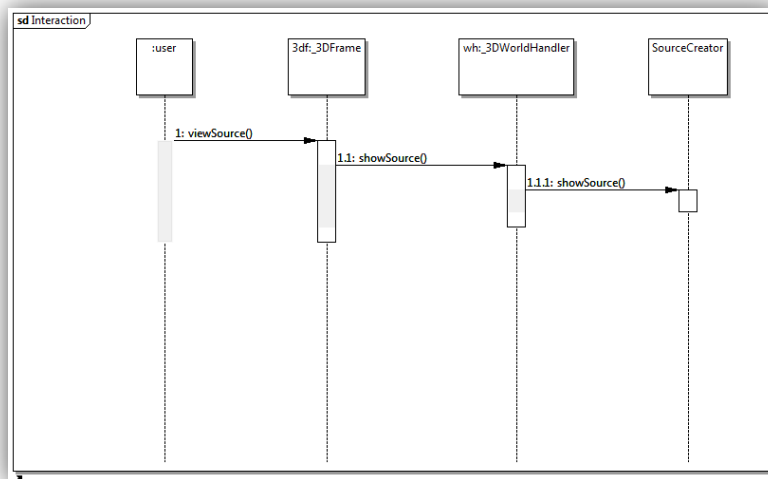


Figure 24 Sequence diagram 7

This method is called from `_3DFrame`, the call is passed to `_3DWorldHandler`, and then `showSource` which is defined in `SourceCreator` is called. This method creates source depending on the current contents of the scene graph which are rendered on the

screen.

So, whenever user adds a shape onto the _3DFrame and click on the view source button, the change which was made by latest added shape can be seen.

4.2.1.8 Sequence diagram for use case 8: Save project

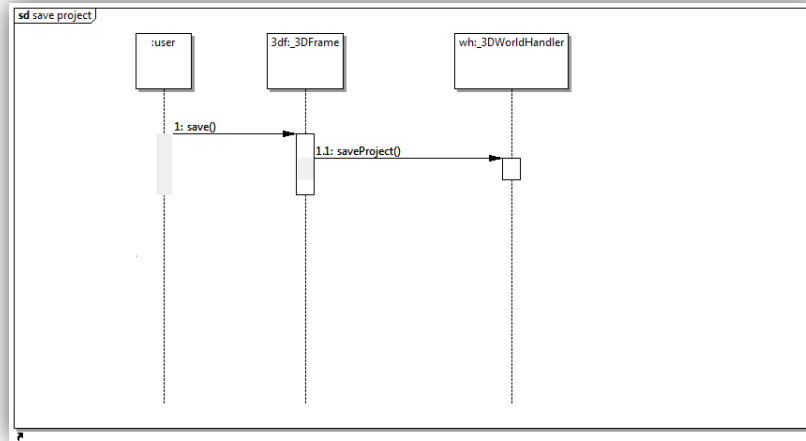


Figure 25 Sequence diagram 8

Whenever user selects save from the menu the _3DFrame will call saveProject() method from _3DWorldHandler, which will scan the whole tgList and put data into already created .3d file.

5 Implementation

This chapter is about how the implementation of each component has been carried out and how every class contributes to the working of the software as a whole. Many screenshots have been included in this chapter which explain more about the working of every component.

5.1 Program flow

To start with, the figure below shows the different states the application will be in and how the different activities in the system will be carried out:

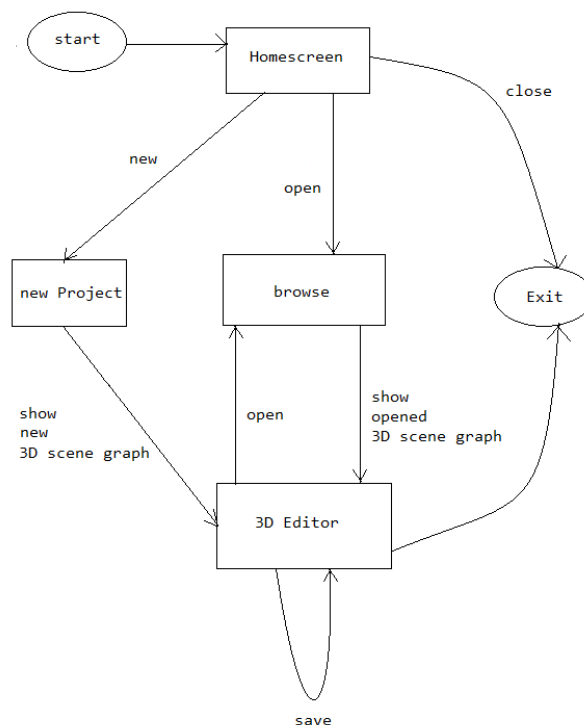


Figure 26 Program Flow

The important states of the software are shown in rectangular shapes; the transitions are shown with arrows. The following are the states in which software will be when it is up and running.

- **Home screen:**

In this state user will be shown a home screen of the application, from here, user can create a new project, or load an existing project, or exit from the system. If user decides to create a new project, then user will be shown a dialog box asking the name

for the project. Similarly, if user chooses to load existing projects, user will be shown a dialog box, where user can browse and check directories in their computer for .3d files.

- **Select project/browse for projects:**

User will be shown the projects in the projects directory; here user can also navigate to other directories to search .3d files. From this state, the users will only be shown the 3D Editor window.

- **Show scene graph/Editor window:**

In this screen, we show user the default X, Y, Z, coordinates along with the grid; or the grid and coordinates with the loaded project, depending on user's former actions. User will make changes to the scene graph in this screen, from this screen users can either close the programs, save current project, or load another project.

5.2 Implementing the code

5.2.1 The home screen

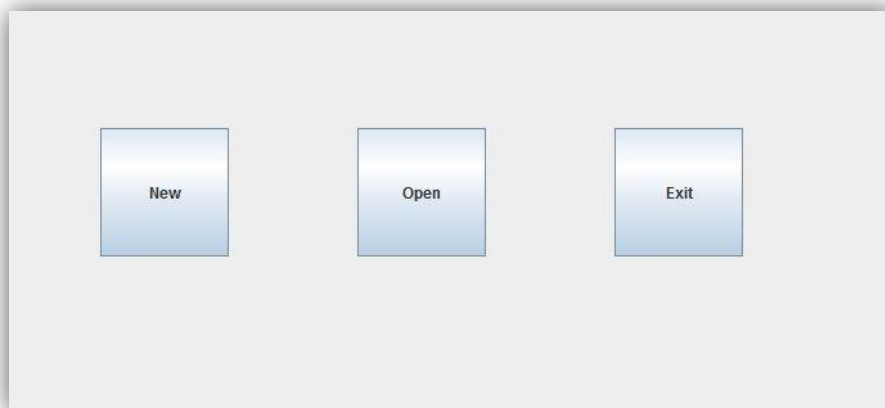


Figure 27 Screenshot: homescreen

Whenever end user runs the application, the following home screen is shown which has three choices. The home screen is linked to the `_3DFrame`, so if user selects any option from the home screen, method from `_3DFrame` is called.

If user selects an option to create a new project, they are shown a dialog box, where the name of the new project can be entered. When the new project is successfully opened, the String attribute in `_3DWorldHandler` called `opened`, stores the name of the

new project. This attribute is useful in saving of a project, because program knows which file to save. 'opened' is also used in creating source for the current project.

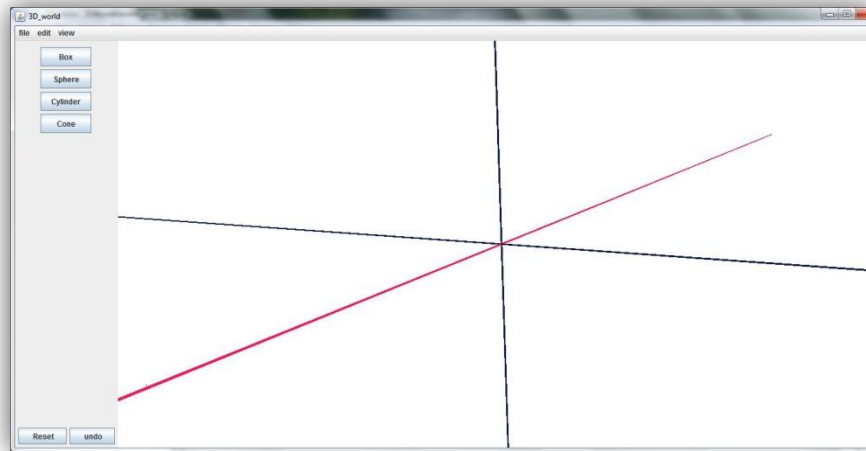


Figure 28 Screenshot: showing a newly opened project

This is how the newly loaded project looks like; Z axis is shown in red and axes X and Y are shown in white. The axes are nothing but java 3D primitive boxes with a very tiny X, Y, or Z length. The axes or any shape for that matter are added to the scene graph from `_3DWorldHandler` class.

5.2.2 Adding objects in newly created 3D world

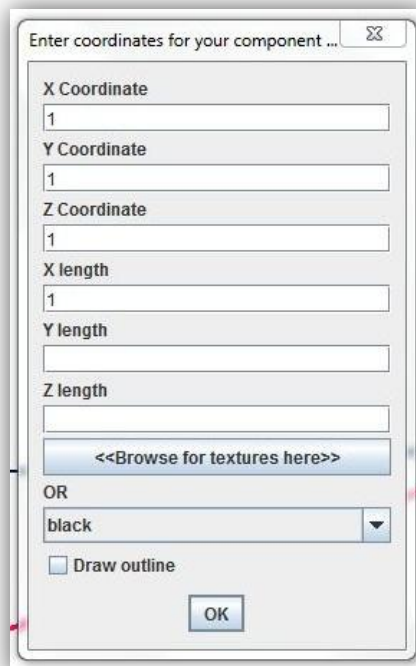


Figure 29 Screenshot: dialogBox to add a box into scene graph

Imagine a scenario where user wants to add a shape onto the scene graph, user clicks on 'box' JButton, after that `_3DWorldHandler` will show dialog 'Box' to the user, which is achieved using the `dialogBox()` method. The data retrieved from the dialog box is sent to `_basic3DWorld`, where java3D box is created according to the user specification and it is placed onto the user specified location.

According to the user data that has been received, the Booleans in the `_3DWorldHandler` will be set, if the user has selected any image for textures, `textured` will be set to true. If the user wants to draw an outline for the added object then `drawOutline` Boolean will be set to true. If the user has not selected any textures then the color selected by the user will be applied to the shape.

Users can also see how each object has been constructed, what other shapes have been used to create that object, to accomplish that for a particular shape users can choose to draw an outline for an object while adding the object onto scene graph.

The figure below shows a box with its outline and the same object without its outline.

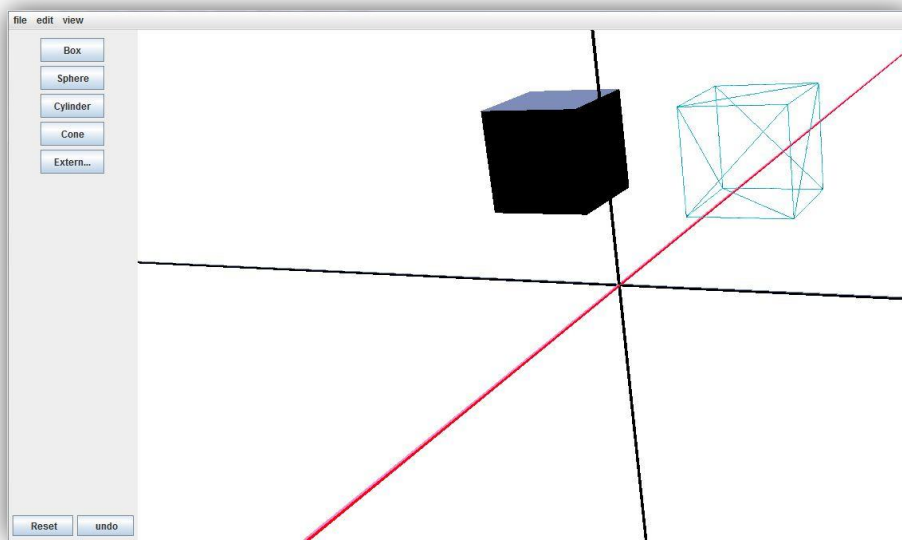
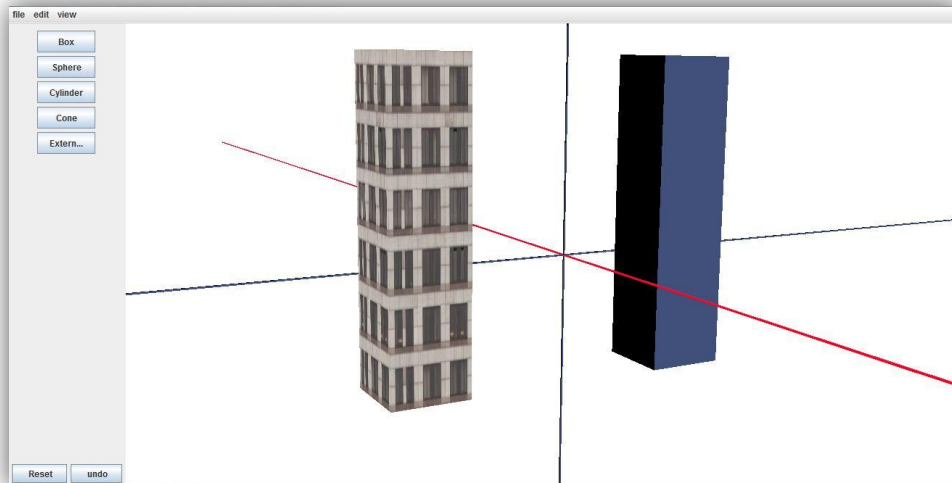


Figure 30 A box with and without an outline

Now if user has selected a texture for an object the scene builder will load that image and put it around the object as a texture. In screenshot below you can see a box has been added with dimension (1x, 4y, 1z), with a texture and without a texture.



The following project was created while testing the application, and users can create 3D world like these using scene builder. What you see in the screenshot are nothing but boxes with different textures.



Figure 31 Example project in scene builder

5.2.3 Editing added object into the 3D World

Many changes could be made to the objects added by the user, such as changing the position of an object, rotating an object, deleting an object from the scene graph, changing size of the object. To accomplish these tasks Mover class which extends Java3D behaviour is used.

Now in order to move an object users first have to select an object, users can do this using mouse. For the selection of a 3D object we use Picking classes supplied by java3D.

```
PickCanvas pickCanvas = new PickCanvas(canvas3D, sceneBG);
```

In order to use picking, you have to create an object of PickCanvas class which is supplied the canvas used by the application to draw 3D elements and the root branch graph as parameters.

```
PickResult result = pickCanvas.pickClosest();
```

Then the PickCanvas object is used in mouseClicked() method where it listens to the user clicks. We use pickClosest() method of PickCanvas to determine the object which is closest to the screen, remember there could be more than one object behind the object user is currently selecting, pickClosest() returns PickResult object which is stored in result.

After getting the result we can retrieve what type of object was selected by the user, but the problem is that is the only information that can be recovered using java3d picking utility.

The above two statements show us how getNode() method of PickResult can be used to retrieve the information that we need, so imagine user clicks on any of the four primitive shapes, then 'p' will not be null it will have some value in it, we use this information to process the selection.

As I mentioned earlier, all that we can recover using picking is the type of object which was clicked. This is not very helpful, especially when there are more than one objects of same type in the scene graph. In order to identify the type of object, we use one property of Primitive and Shape3D class called userData and we have getter and setter methods provided by these classes for this property.

So, whenever an object is added to the scene graph we set a unique ID as its userData. Each time a shape has been added to the scene graph; this ID is incremented and so is unique. This ID is stored in _tg class, and is used by _3DFrame for identification of added shape. Whenever user clicks on any visible object we get an ID of that object, and this ID is then checked with every object that has been added to the tgList, if a match is found, we set the movable property of that _tg to true. So when user presses any key to change the object in Mover class we move only those _tgs whose movable property is true. At any given time, there is going to be only one _tg with its movable property true.

The screenshot below will show how an object can be moved using the arrow keys, the screenshot contains the locations of the object initially, and the location after it was moved.

How the moving of an object actually works:

Below you can see a basic scene graph with a sphere added at location (0x, 0y, 0z), you can see the location of the sphere at the bottom of the screenshot. The program is simply printing the location of sphere every time it is changed.

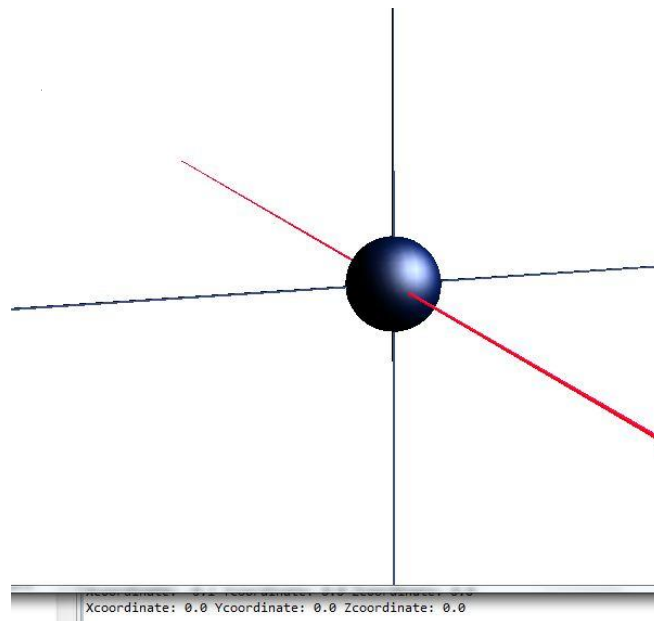


Figure 32 Moving: Position 1 of sphere

The next screenshot shows the same sphere with a change in its position, it can be clearly seen that the object is no more at its default position (0x, 0y, 0z).

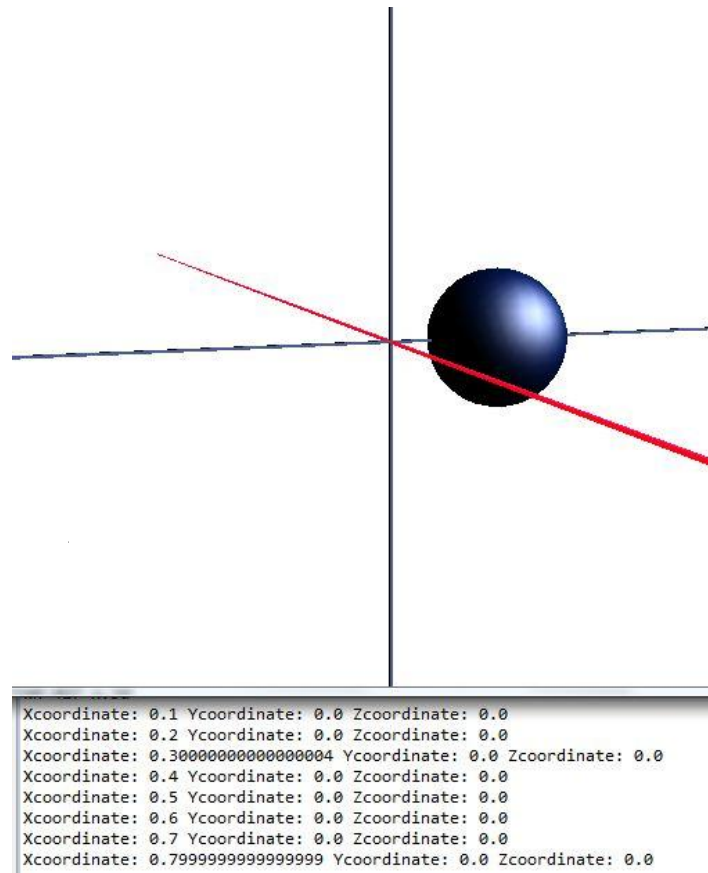


Figure 33 Moving: location of the sphere after being moved

This screenshot also shows how the location was changed, the move() method in Mover was called 8 times, you can figure that out from the X location of the object, it is incremented by .1, and the object is being moved in X positive the number is incremented.

To move an object in 3D world, mover class has a method called move() which is given the location of the new position of the object.

When user presses any key to move an object, we first create a Vector3d for that move, suppose user wants to go in X positive, and then the new Vector3d object that will be generated is:

```
new Vector3d(MOVERATE, 0, 0)
```

This Vector3d is then passed to move() method, where we first check if the object is movable, because it is undesirable to move any other object than what user has pressed. After that we get the Transform3d object from the TransformGroup object of the current shape.

```
Transform3D t3d;
objectTG.getTransform(t3d);
```

objectTG is the TransformGroup of the current object, so we retrieve Transform3D object from objectTG and store it in t3d. As discussed previously Transform3D repre-

sents the current location of a 3D shape and is used for affine transformation. Transform3D has a method 'mul()' which multiplies the affine 3D 4x4 matrix internally stored in Transform3D by Vector3d object 'toMove'. This 'toMove' object has information about where the object will be relocated.

```
t3d.mul(toMove);
```

So, in the end all we do is set the currently altered Transform3D object 't3d' to the objectTG TransformGroup which changes the location of the object.

```
objectTG.setTransform(t3d);
```

5.2.4 Undo

The undo method is defined in _basic3DWorld class, where it retrieves the previous scene graph from the stack maintained by _basic3DWorld, as mentioned earlier, every time some object is added to the scene graph, the stack contents are also updated.

To see the undo() operation in action, look at the figure below, it has 4 different shapes added to the scene graph, they all will be removed using the undo operation.

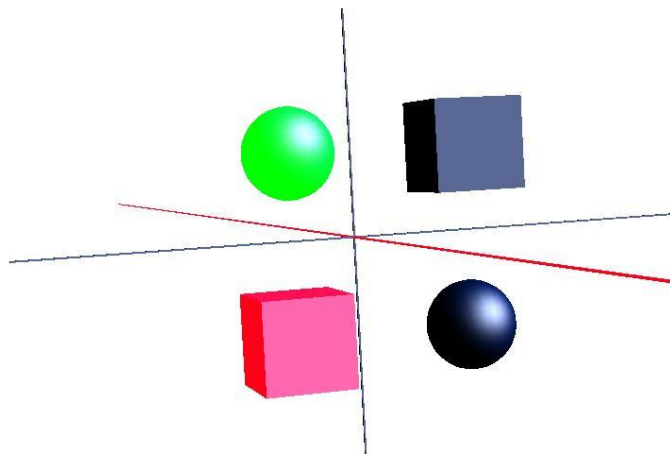


Figure 34 Undo: it is all there

Now the undo method will be called twice, and the two shapes which were added latest will be removed.

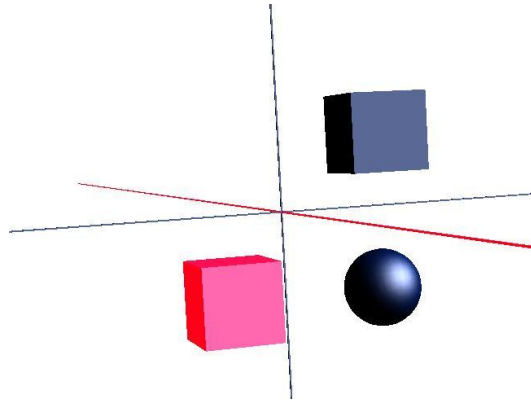


Figure 35: After calling `undo()` for first time

You can see in this image, the green sphere which was added has been deleted. Upon pressing undo again the next shape which was added will be removed.

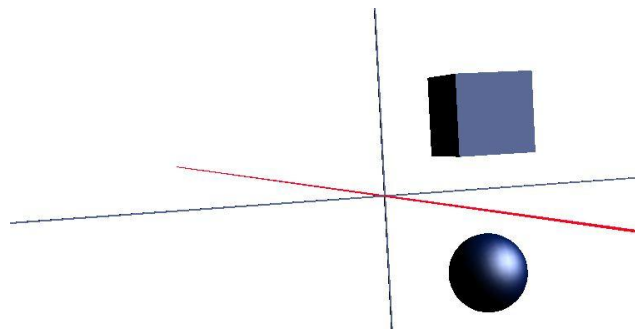


Figure 36 Undo: the state of scene graph after performing `undo()` twice

5.2.5 View Source

When the requirements of the software specified, the idea was to design software which helps user learn how java3D works, even though this feature was only intended for programmers, it can be used by non-programmers to see how the shapes are actually added to the scene graph by java3d, and what parameters are passed to the methods in `_basic3DWorld`.

Now, for the sake of this section, four shapes will be added to the scene graph, and the operation of the view source will be shown.

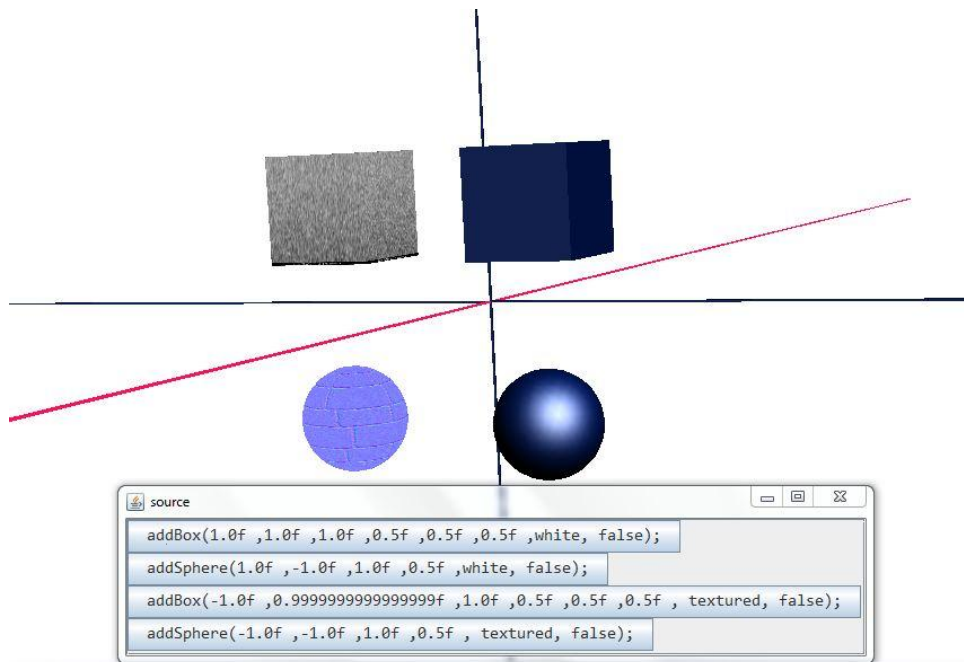


Figure 37 View Source: showing methods calls made to `_basic3DWorld` for current scene graph

As it can be seen, the method calls which are shown are nothing but JButton, and upon clicking these JButtons the actual method definition is shown onto the Source JFrame. If we delete any of the shape that is in the scene graph, and click on the view source again, the method call made for that shape will be removed from the source JFrame.

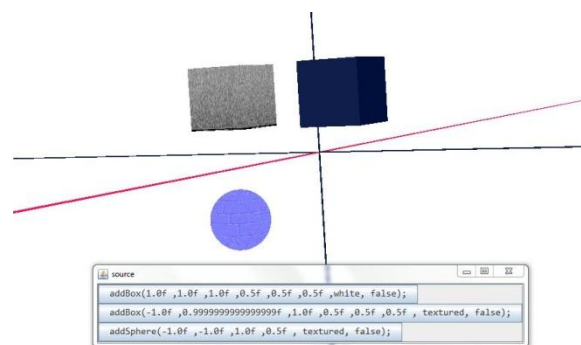
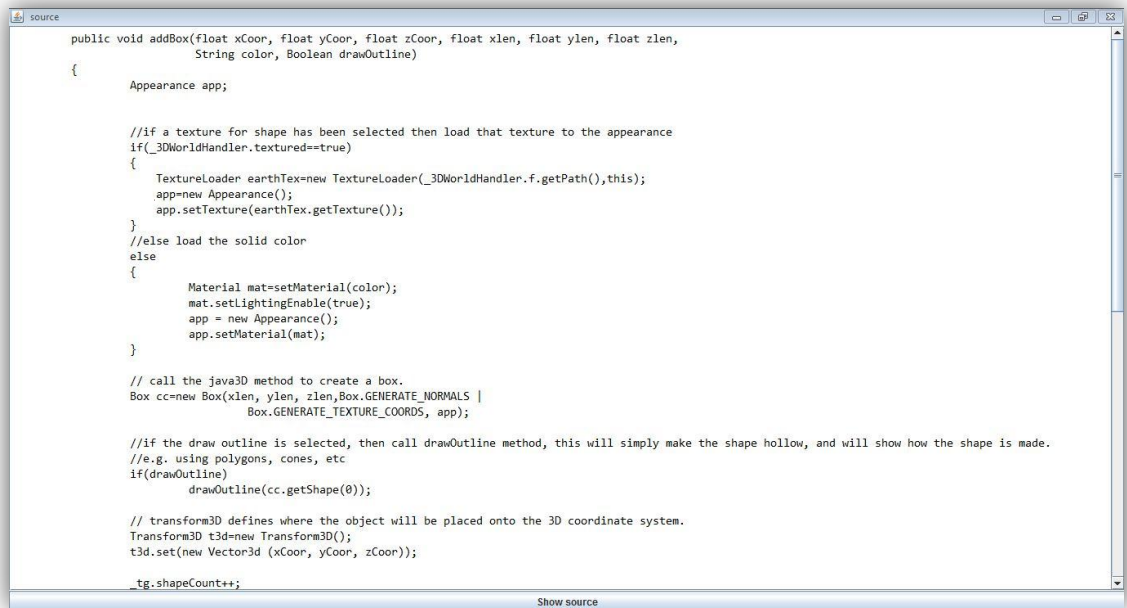


Figure 38 Source: After one of the sphere has been removed.

The way this component of the system works, is the SourceCreator class maintains a list of JButtons which is updated whenever user presses on 'viewSource' menu item. Only those objects are shown to the user which are currently present on the scene graph. The information about the objects which are present in the scene graph is maintained by `tgList` in `_basic3DWorld` class. So, in order to retrieve that information, `_3DWorldHandler` class has a method `addLabelsFromTG()`, which retrieve every `_tg` from the `tgList` and creates a label, and adds a JButton onto the JButton list in

SourceCreator.addLabelsFromTG() method is called every time the ‘viewSource’ button is pressed. So, the method calls which are shown to the user are always updated. As referred earlier, when any of these JButtons are pressed, users are shown the method definition of that call. These methods are different from what is actually used in the program. The components of the method which are used for working of scene builder are removed and many comments are used for better readability.



```
source
public void addBox(float xCoord, float yCoord, float zCoord, float xlen, float ylen, float zlen,
String color, Boolean drawOutline)
{
    Appearance app;

    //if a texture for shape has been selected then load that texture to the appearance
    if(_3DWorldHandler.textured==true)
    {
        TextureLoader earthTex=new TextureLoader(_3DWorldHandler.f.getPath(),this);
        app=new Appearance();
        app.setTexture(earthTex.getTexture());
    }
    //else load the solid color
    else
    {
        Material mat=setMaterial(color);
        mat.setLightingEnable(true);
        app = new Appearance();
        app.setMaterial(mat);
    }

    // call the java3D method to create a box.
    Box cc=new Box(xlen, ylen, zlen,Box.GENERATE_NORMALS |
Box.GENERATE_TEXTURE_COORDS, app);

    //if the draw outline is selected, then call drawOutline method, this will simply make the shape hollow, and will show how the shape is made.
    //e.g. using polygons, cones, etc
    if(drawOutline)
        drawOutline(cc.getShape(0));

    // transform3D defines where the object will be placed onto the 3D coordinate system.
    Transform3D t3d=new Transform3D();
    t3d.set(new Vector3d (xCoord, yCoord, zCoord));

    _tg.shapeCount++;
}
Show source
```

Figure 39 Example of how the code is shown to the user

The above screenshot is an example of how the method definition of any particular object is shown to the users. If users are interested in seeing how the scene graph is generated, what kind of lighting is necessary, or how the background is added to the scene graph, they can click on ‘show 3D world class’ JButton at the bottom, this will load the actual _basic3DWorld class into the JFrame, and users can see the building blocks of java3D.

5.2.6 Adding external models

For the sake of simplicity and considering the limited time provided for the project, only one external model have been included in the application, this model is designed by NCSA. The models are copyrights of ‘The Board of Trustees of the University of Illinois’. There are no legal issues in using the models for the academic purposes; hence we have included them in the system. [7]

In order to use these models in the java3D, we have been provided a class PropManager.java. This class was provided in the book Killer Game Programming by Andrew Davison.

The 3D models used by NCSA are stored in .3ds file extension and they can be created using many third-party softwares. The problem with the 3D models which were provided was, when the model was added into the scene graph, they were out of place, they were either too big for the scene graph, or they were needed to be rotated in order to make them appear as if they are on ground. Also, the textures used on these models were not proper.

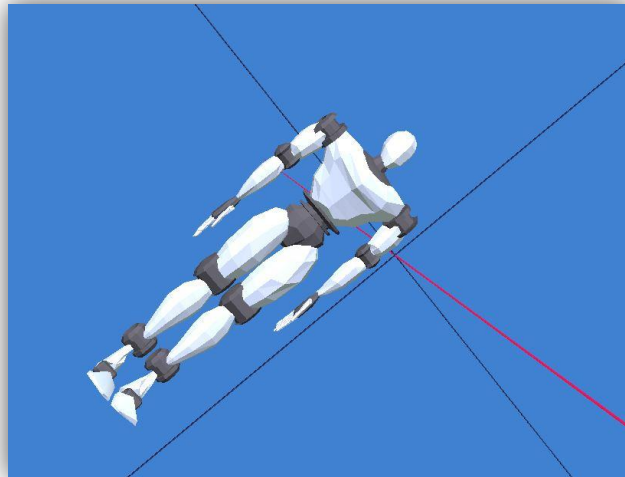


Figure 40 .3ds Model added is horizontal to X axes

As it can be seen in the screenshot, the .3ds model added in the scene graph is not in vertical position which it should be, as it is not desirable, because if a user has added some land in their scene graph, and they add .3ds model, it will be horizontal to the land, which is not something which user wants.

Also, if two different models, for example, a car and a robot are added to the scene graph, they both are not in proportion with each other, which is shown in the next screenshot.

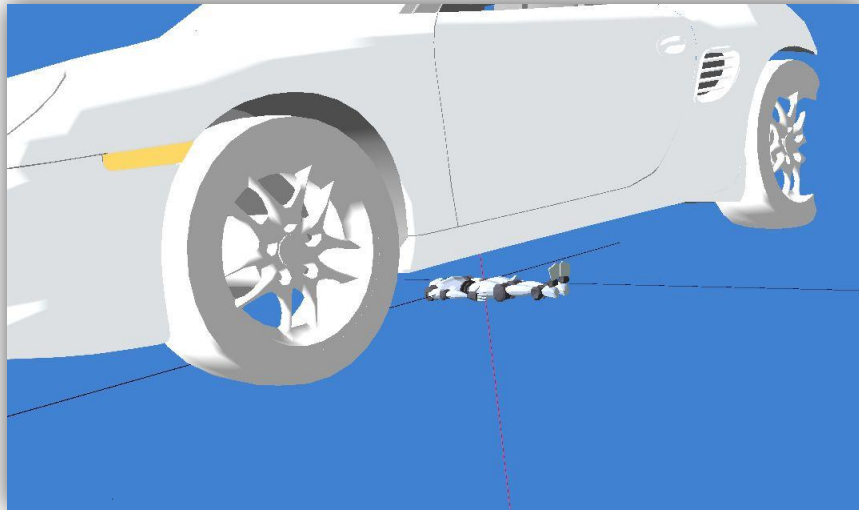


Figure 41 Out of proportion models

The models are just been added without moving them and you can see that they are not in proportion. It can also be seen that the texture of the car is not accurate.

So, in order to make the 3d models in proportion with each other, we had to perform transformations to the TransformGroup to which the 3d models were added.

The next screenshot shows us the same models again; both the models are scaled, and rotated so that they look in proportion with each other.

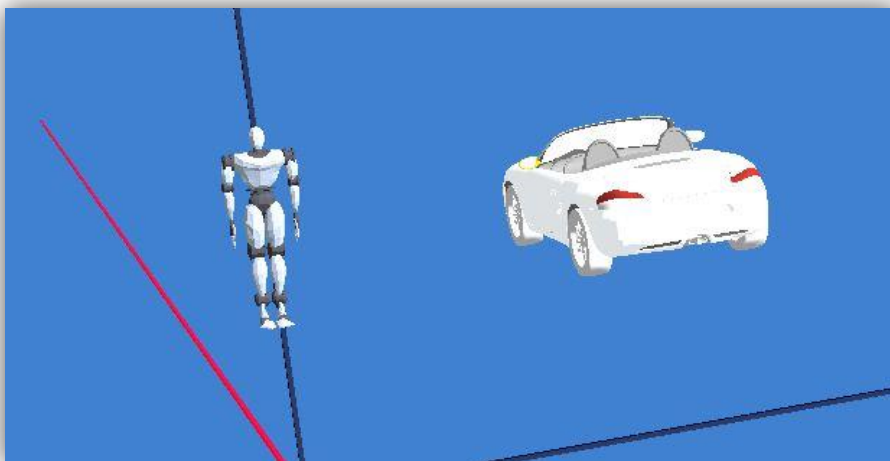


Figure 42: Models processed in order to make them proportionate

The screenshot below shows the 3D world which was constructed during the testing of the application, it contains many 3D models of vehicles and trees, the buildings which are seen in the screenshot are the boxes added with textures.

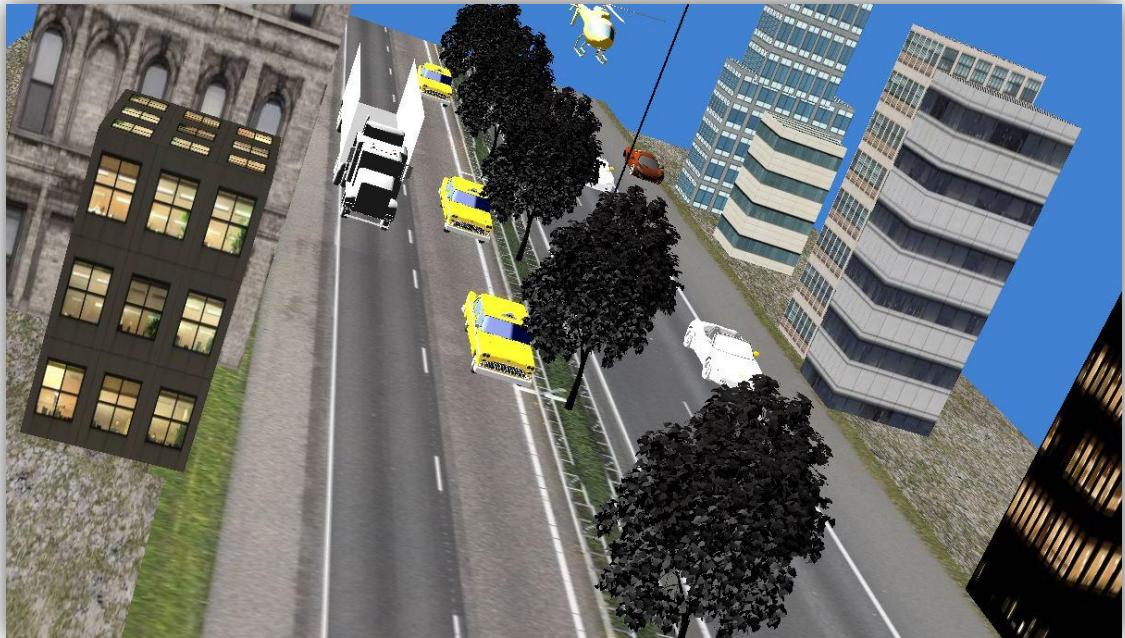


Figure 43: Screenshot 3D world with external models

Every model that you see in the screenshot had to be processed before loading into the scene graph, the application will give uses a list of .3ds models which they can add to the scene graph, whenever user selects any .3ds model, it will first be adjusted and then will get added to the scene graph.

5.2.7 Saving and loading a scene graph in .3d file

As mentioned in previous chapter, the scene graphs are saved in a .3d file format, which is internally a text file storing data about the nodes attached to the scene graph. The .3d file stores details about the light in the scene graph, the location of objects in the scene graph, and their coordinates, their dimensions and any textures attached to them.

```
light      -1.0  -1.0  -1.0
box       -1.4   1.0   1.0   0.5  0.5  0.5  red false notext
Sphere    1.0  -1.0   1.0   0.5
box       -1.0  -1.0   1.0   0.5  0.5  0.5  red true  notext
Cylinder  -1.0  -1.0   1.0   0.5  0.5
ext       1.0   1.0   1.0
          H:\j3DEngine\addExt\textures\blue_jeans.jpg
          taxi.3ds
```

Figure 44 .3d file example

A .3d file with a couple of boxes, a sphere, a cylinder and an external model looks like this. The leftmost column represents the type of object that was added in the scene graph, this part will determine what type of data about the object will be stored.

If it's a sphere we are storing in file we will save its location, radius, color of the sphere, or its texture location, etc. If a sphere is not textured, then 'notext' will be stored in file, which is used as an indicator while doing the loading operation.

While loading we read the whole line from the file, and tokenize it using the java Tokenizer class. Depending on what type of node is stored, which is indicated by the leftmost column in the file, we load data in our scene graph.

Again, the loading logic varies depending upon the shape or object that has been detected at first token.

6 Testing

The core architecture of the system was first prepared and implemented, it was tested rigorously and then, the other components of the system were added to it.

In order to ensure the bug free execution of the software testing was carried out.

Through the whole of the project, unit testing approach was followed, where code was separated from the main program and was tested thoroughly, when all the bugs were removed from the separated code, it was integrated in the main program. The functional testing of the program was carried out after the building of the system, as well as during the implementation of the system.

The application was tested both on windows as well as linux environments, the windows platforms used for the testing were, windows XP, & windows 7, and the linux platform used for testing was Ubuntu distributions by Canonical.

Throughout the development of the system, as well as after the development was finished the testing was carried out. During the development phases white box testing was mostly used to check if the core system is giving the desired results, it is discussed below.

6.1 White box testing

The white box testing of the system was carried out during the development of the core system. The application was tested and improved until the expected result was achieved. The important factor in the white box testing was the data flow, carried out between the files and the scene graph. If the data is not correct then the objects loaded into the scene graph will not be proper, and as per the user expectations. The data was checked either by printing it on the console or by setting breakpoints and checking the values between different variables taking help of eclipse debug utility.

This approach was really helpful when the development of selecting an object for movement using mouse was in progress. The problem as mentioned in the implementation was the picking utility was only detecting the type of shape that was clicked, so in order to identify particular shape, we had to set userData to it.

6.2 Functionality testing

After completion of the system, the application was given many different and extra ordinary inputs to see how system behaves.

In one of the testing many shapes and external models were added into the scene graph, until the system crashed, when the system crashed after overloading of objects JVM gave OutOfMemoryException. This is still acceptable as the number of objects that were added in the scene graph were far good greater than practical use of the application.

6.2.1 User testing

The application was given to users to test; they were also given a questionnaire which was used to evaluate the software, see appendix 2 for the questionnaire. While reviewing the questionnaires following issues were brought up by the users,

- *“If the value of the coordinate supplied to an object is more than 10, sometimes the object is not visible”.*

This clearly means that user has added an object and the object is outside bounds of the application. To resolve this issue, the position of the object entered by the user can be checked, if the position is outside the bounds of the scene graph, users should be given some kind of warning.

- One of the user said that when a scene graph is moved to look around using mouse, but this movement is little too fast to handle.

Apparently the system on which the user tested this application had its mouse motion set to very fast, which resulted in this issue; other users did not find any problem with the mouse.

- Many users complained that when pressing addBox button the dialog box which is shown contains too much information, it is not easy to figure out what to put in the fields on the dialog box.

This can be solved by showing tool tips to the users which will direct them on how to use the software, user can be shown a tooltip box as well, which will help them on what values to input into the fields in the dialog.

User can also be shown an example of where the shapes are added on giving different inputs.

7 Conclusion

7.1 Summary and Evaluation

The Java3D Scene Builder has implemented all the features that were specified in the requirements, the application was tested with few potential users of the system, the application was well received with all the users of the system, many of the users found the software very interesting, and even though initially it was hard to get hold of, once users start building scene graphs and manipulating the objects around it was easy to use.

Many users liked the fact that the application was reliable, fast, and the scene graph generated is portable, and can be used in not only different platforms but also on applets which can be integrated with the browser.

To use the application very little training is required, once users learn the structure of the 3D environment, and are used to the 3D world, it is very efficient to operate.

One of the testers for the application was an architect student who felt this application is very helpful for architects, and with some improvements it can be used for designing purposes by architect students.

Based on this feedback, it was conceded that the Java3D Scene Builder has achieved its primary objectives.

In the section below the critical evaluation of the software will be made, discussing both the strong and weak aspects of the application.

7.1.1 GUI

GUI of this application could be improved in many ways, the GUI uses default JButtons and menus given by java, due to the limitations of time they were not changed, but instead of using text onto buttons images could have been used, which might have helped users improve the user experience of the software.

Many users also complained that help provided in the form of tooltips and dialog boxes is not enough; the application can be improved in this segment.

GUI is responsive and an action such as deleting of an object, or adding of an object, movement of objects, is done fairly quickly.

7.1.2 Objects

Once a shape has been added, you can move it, rotate it, and scale it, but, the color of the shape cannot be changed, also, users cannot add textures or change textures of the added shape.

User can add external 3ds models onto scene graph but once added they cannot make any changes to it; this is simply because using picking it was not possible to identify different external 3ds objects.

One user suggested that the objects added to the scene graph could be moved around using mouse, instead of using the arrow keys, this kind of approach is very fast while interacting with 3d objects in the scene graph.

7.1.3 3D Editor

One of the good features of the 3D Editor is users can move around the editor using mouse, they can also zoom in and zoom out into the 3D world, it helps users to observe where the shapes have been added, and they can change it if they want.

The responsiveness of the application is very good, and the shapes are added to the scene graph in no time, without any lag.

More guidelines are needed to use the 3D editor, such as position markers which will help user figure out the location where they have to add the 3D object.

There should be a provision for users to change the viewing position or the viewing camera, either by using keyboard or mouse, or by manually entering values to change.

7.2 Future work

Several extra features could be added to the application to make it more complete, we will be discussing the features which can help improve the user experience of the software and also some of functionalities which could make the software usable for more serious things such as creating 3D worlds for games in java.

7.2.1 Animation

The application only allows movement of objects through user interactions, the ability to animate objects has not been added. This could be added in future which will help users make their scene graph more intuitive. Users should be able to add animations to make the objects bigger or smaller, or, users should be able to rotate the object.

7.2.2 Changing background

Users should be given an option to change the background of the application, not just the colour of the background but also users should be able to add their custom background image to the scene graph.

7.2.3 Change texture or colour on the go

Once the users add an object they cannot change the colour or the texture of the added object, this functionality could be added which will improve the usability of the application.

7.2.4 Selecting more than one shape for movement

Users should be able to select more than one shape using ctrl key to move them or do any changes to them.

7.2.5 Creating models

Users should be able to create their own 3D models from this application; these models could be used as in other scene builder projects or in any other 3D application.

We can give these models a particular datatype, and which can be used only using the loader.java file which will be provided in future.

7.2.6 More complex shapes

A choice of wider complex shapes will be given to the users, which can help them build their models. A complex shape will be made up of more than one shape, and will be treated as one BranchGroup in the program, so it can be moved easily.

7.2.7 Loading of .3d files in other projects

Users will be given a loader file, which will help them load .3d files in their own project, without doing many efforts. So all users have to do is, run the loader file through their program, and may be passing a .3d file as parameter for the loader constructor. This can of course be done using GUI.

References

- [1] http://jmonkeyengine.org/wiki/doku.php/sdk:scene_composer
- [2] https://wiki.brown.edu/confluence/download/attachments/3735750/3D_coordinate_system.gif?version=1&modificationDate=1195591779000
- [3] <http://download.java.net/media/java3d/javadoc/1.4.0/javafx/media/j3d/doc-files/ViewModel.html>
- [4] http://en.wikipedia.org/wiki/JMonkey_Engine
- [5] <http://jmonkeyengine.org/groups/development-discussion-jme2/forum/topic/java3d-vs-jme>
- [6] <http://fivedots.coe.psu.ac.th/~ad/jg/ch9/index.html>
- [7] <http://cs.gmu.edu/~eclab/projects/mason/ncsaportfolio/portfolioLicense.txt>
- [8] <http://www.java.net/node/647308>

8 Appendix A: Use case description

1: New Project

Related requirement: 1A

Description: Whenever user runs the program for the first time, an option to create a new project might be selected; user can create a new project even from the 3D editor window.

Post-condition: The user should enter valid name for the new .3d file.

Main flow: This use case begins when user selects an option to create a new 3D project. User enters the name of the project, system checks if the project of similar name is present, if it is present, the user is asked to change the name of the project.

Exceptional flow: The 3D file name entered by the user should be a valid name; it should not contain any special symbols, other than ‘_’. The .3d file of the same name should not be present in the current directory.

2: Open project

Related requirement: 1B

Description: Whenever user runs the program for the first time, an option to open previously saved project might be selected, user can also open project from the 3D editor window.

Main flow: this user case begins when user want to open saved projects, and clicks on open button, either from the home screen or from the menu option. So, from here on, the user may find the 3d project in the default directory, or user can browse the other directories in the physical media.

3: Add 3D object.

Related requirement: 2A, 2B

Description: Once the project has been opened in 3D editor window, users may add a 3D object into the scene graph. This 3D object can be a primitive shape with an image texture, or a 3D model.

Pre-condition: A valid project should be running.

Post-condition: Whenever user clicks on add object button, depending on the type of object that user wants to add, a dialog box will be shown to the user; this dialog box will ask user for things like size and coordinates of the object on X, Y, Z coordinate system.

The user should enter valid data for these parameters. By valid data we mean, the size of object can only be in digits, and if user wants to add textures to the object, the image selected for the texture should be a valid image, accepted by java.

Main flow: Use case activated when user wants to add a button, a dialog box is shown in order to get the data regarding the object. If all the data is valid, the object is added in scene graph.

Exceptional flow: As mentioned above, the exceptions in this use case could be the data entered by the user is not valid, in that case, we show the dialog box again, we do not add the object, unless the data is correct.

4: Edit object

Related requirement: 3A, 3B, 3C.

Description: After adding the 3D objects in the 3D editor, users may choose to change the characteristics of the object. Users can also make changes to the objects which are already present in the scene graph.

Pre-condition: A valid project with some objects in them should be present and the '*movable*' property of the object is set to *true*. This is done by clicking onto the object using mouse.

Main flow: Activated when user selects any object and makes some changes to the object.

5: Undo

Related requirement: 4

Description: If user makes some change to an object, and the change is not desirable, user can undo that change.

Main flow: Activated when undo button is pressed, pops the branch graph data from the stack and loads it into the scene graph.

Pre-condition: Stack cannot be empty, i.e. user should have made at least one change to any one of the scene graph object. Because you cannot undo what hasn't been done.

6: Reset

Related requirement: 5

Description: Sometimes users may not be happy with whatever they have created; in that case, users may choose to reset the scene graph.

Main flow: Activated when reset button is pressed, deletes every object from the scene graph.

Exceptional flow: Reset should not be performed on a new or empty scene graph.

7: Show source

Related requirement: 6

Description: If a user wants to check the source of the current scene graph, they can check it using the menu button 'show source'.

Main flow: Activated upon pressing the show source menu button, shows a JFrame window to the user containing the code generated from the changes made to the 3D editor.

Pre-condition: A valid project must be loaded in 3D editor.

8: Save current project

Related requirement: 7

Description: Users should be allowed to save every project in the physical media, so that, the projects can be used later on.

Main flow: When user presses the save button, this use case is activated, and the scene graph will be saved in the default projects directory.

Pre-condition: A valid project must be loaded in 3D editor.

9 Appendix B: User Questionnaire

User Questionnaire: Java3D Scene Builder

1: Is the GUI user friendly?

Yes [] No []

If no, where the improvements could be made? Leave comment below.

2: When you add objects onto the screen, do they appear exactly where you want? Did you have to change the location of the object after adding?

Yes [] No []

3: The program provides default textured, were they enough?

Yes [] No []

If no, what other texture should be added? Leave comment below.

4: Any object added to the scene graph can be edited, it can be moved, rotated and scaled, was it enough?

Yes [] No []

5: Did you come across any crashes while using the system? If so, please describe shortly.

Yes [] No []

6: The application allows you to see the source code of the changes you have made to the scene graph, was it helpful to you? Is the code provided informative enough?

Yes [] No []

7: How would you rate the GUI?

Bad 0 1 2 3 4 5 Excellent

8: how would you rate the responsiveness of the applications??

Bad 0 1 2 3 4 5 Excellent

8: Would you recommend this software to anyone?

Yes [] No []

9: Any comments-

The GUI could use little improvements

Your Name: Yoshit Patel

Signature: 

User Questionnaire: Java3D Scene Builder

1: Is the GUI user friendly?

Yes No

If no, where the improvements could be made? Leave comment below.

2: When you add objects onto the screen, do they appear exactly where you want? Did you have to change the location of the object after adding?

Yes No

3: The program provides default textured, were they enough?

Yes No

If no, what other texture should be added? Leave comment below.

4: Any object added to the scene graph can be edited, it can be moved, rotated and scaled, was it enough?

Yes No

5: Did you come across any crashes while using the system? If so, please describe shortly.

Yes No

6: The application allows you to see the source code of the changes you have made to the scene graph, was it helpful to you? Is the code provided informative enough?

Yes No

7: How would you rate the GUI?

Bad 0 1 2 3 4 5 Excellent

8: how would you rate the responsiveness of the applications??

Bad 0 1 2 3 4 5 Excellent

8: Would you recommend this software to anyone?

Yes No

9: Any comments-

Your Name: *Haipeng Wang*

Signature: *Haipeng Wang*

User Questionnaire: Java3D Scene Builder

1: Is the GUI user friendly?

Yes No

If no, where the improvements could be made? Leave comment below.

2: When you add objects onto the screen, do they appear exactly where you want?
Did you have to change the location of the object after adding?

Yes No

3: The program provides default textured, were they enough?

Yes No

If no, what other texture should be added? Leave comment below.

4: Any object added to the scene graph can be edited, it can be moved, rotated and scaled, was it enough?

Yes No

5: Did you come across any crashes while using the system? If so, please describe shortly.

Yes No

6: The application allows you to see the source code of the changes you have made to the scene graph, was it helpful to you? Is the code provided informative enough?

Yes No

7: How would you rate the GUI?

Bad 0 1 2 3 4 5 Excellent

8: how would you rate the responsiveness of the applications??

Bad 0 1 2 3 4 5 Excellent

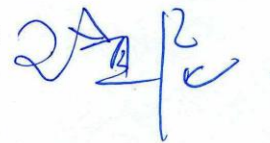
8: Would you recommend this software to anyone?

Yes No

9: Any comments-

Your Name: WANG XUANNI

Signature:



User Questionnaire: Java3D Scene Builder

1: Is the GUI user friendly?

Yes No

If no, where the improvements could be made? Leave comment below.

2: When you add objects onto the screen, do they appear exactly where you want?
Did you have to change the location of the object after adding?

Yes No

3: The program provides default textured, were they enough?

Yes No

If no, what other texture should be added? Leave comment below.

4: Any object added to the scene graph can be edited, it can be moved, rotated and scaled, was it enough?

Yes No

5: Did you come across any crashes while using the system? If so, please describe shortly.

Yes No

6: The application allows you to see the source code of the changes you have made to the scene graph, was it helpful to you? Is the code provided informative enough?

Yes No

7: How would you rate the GUI?

Bad 0 1 2 3 4 5 Excellent

8: how would you rate the responsiveness of the applications??

Bad 0 1 2 3 4 5 Excellent

8: Would you recommend this software to anyone?

Yes No

9: Any comments-

too fast to handle

Your Name: Amy Li-wei

Signature: 吳怡瑋

User Questionnaire: Java3D Scene Builder

1: Is the GUI user friendly?

Yes No

If no, where the improvements could be made? Leave comment below.

2: When you add objects onto the screen, do they appear exactly where you want?
Did you have to change the location of the object after adding?

Yes No

3: The program provides default textured, were they enough?

Yes No

If no, what other texture should be added? Leave comment below.

4: Any object added to the scene graph can be edited, it can be moved, rotated and scaled, was it enough?

Yes No

5: Did you come across any crashes while using the system? If so, please describe shortly.

Yes No

6: The application allows you to see the source code of the changes you have made to the scene graph, was it helpful to you? Is the code provided informative enough?

Yes No

7: How would you rate the GUI?

Bad 0 1 2 3 4 5 Excellent

8: how would you rate the responsiveness of the applications??

Bad 0 1 2 3 4 5 Excellent

8: Would you recommend this software to anyone?

Yes No

9: Any comments-

That's good enough.

Your Name:

Signature:

JIG ZHOU

10 Appendix C: User Manual

10.1 Installation guide:

Make sure your computer has java3D, and all the classpath have been set.

Run the _3DSceneBuilder.jar file to run the application.

10.2 User Manual

Java3D Scene Builder is used to create 3D scene graphs which can be used across various platforms and can be used in a browser in the form of applets as well.

10.2.1 Adding objects

Whenever you will click one of the buttons to add a shape onto the scene graph you will be shown a dialog box, which will look something like this.

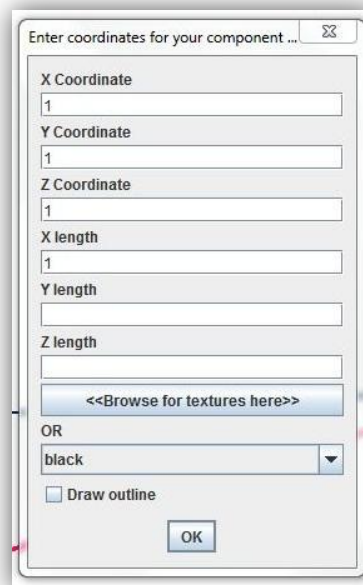


Figure 45: User Manual dialog box

You will have to fill every field that is present in the dialog box, with either a float or a number, once you fill all the fields, you can either choose to add your own textures to the object or you can select a particular colour for your object.

Remember you can either select a colour or you can add the textures, you cannot add both. Once the desired details have been put, press OK, the object will be added in the scene graph.

10.2.2 Editing objects

To edit any object added to the scene graph, click on the object, and move, or rotate the object depending on your requirements, the keys associated with every function are listed below.

10.2.2.1 Movement

Right: X positive

Left: X negative

Up: Y positive

Down: Y negative

Ctrl: Z positive (closer to user's viewpoint)

Shift: Z negative (away from the user.)

10.2.2.2 Rotation

Number pad Key 7: Positive rotation around X axis

Number pad Key 8: Negative rotation around X axis

Number pad Key 4: Positive rotation around Y axis

Number pad Key 5: Negative rotation around Y axis

Number pad Key 1: Positive rotation around Z axis

Number pad Key 2: Negative rotation around Z axis

Delete selected object: D

Increase size of the object: Q

Decrease size of the object: E

10.2.3 View Source

If you want to see how the current scene graph has been created, select View Source menu option from the View Menu, you will be shown a list of buttons, which show how the methods from the 3D world creator class were called.

To see the method definition you can click on any of the button you want, you will be redirected to the method definition.

You can also see how the 3D world creator class has been built, and how the lights or the background were added to the scene graph, to see that you can click 'show source' button at the bottom of the method definition.