Abstract

The aim of this project will be to design and implement an application in which users can create 3D primitives in a 3D workspace and then produce line drawing projections from arbitrary views, storing them into standard 2D vector graphics which can then be edited if wanted. This is helpful for users like researchers and students who work on areas such as computer graphics and solid geometry.

Using 2D drawing packages to draw 3D diagrams consisting of basic primitives and connecting arrows etc with realistic looking 3D perspective view is hard, unless you are good at drawing. It is one thought to consider using CAD or high end graphics tools, such as Maya, but they are not designed for this purpose, particularly in that they do not produce vector 2D output; they aim at rendering scenes which is not suitable for including in reports, papers etc. A requirement analysis by comparing the software in current market is presented in this report and followed by a reasonable project specification.

The basic background theories for developing such an interactive 3D graphic application are firstly discussed in the report. However, the more important part of the report should be emphasizing on the design and development, and Java3D and Swing are combined to work together for the implementation. To be more specific, detailed implementation by Java3D for real-time 3D display and data process will be included in the report, as well as the user interface and interaction design by Swing.

However, it is difficult to develop an excellent application by just one person or in a short period of time, there must be many shortcomings by the end of the deadline when this package tool is finished. A summary of the current achievements will be listed and illustrated with pictures and figures in the report, followed with project evaluation. In addition, the suggestions of further improvements will also be discussed based on the current achievements.

Contents

1. Introduction

1.1 Field of Computer Graphics

Graphics provides one of the most natural means of communicating with a computer, since human's highly developed 2D and 3D pattern-recognition allow us to perceive and process pictorial data rapidly and efficiently [3]. This is why computer graphics has always been developing fast because people prefer graphics in many cases.

To categorize the areas of computer graphics is not easy, however, most of the graphics practitioners agree with some common major areas which could be field of computer graphics [1]:

- **Modeling** attempts to find a mathematical way of storing the shape and appearance properties of an object to the computer. For example, a glass sphere might be described as a center point and a radius to identify the shape. The appearance might be described as a reflection model about how the light interacts with the sphere.

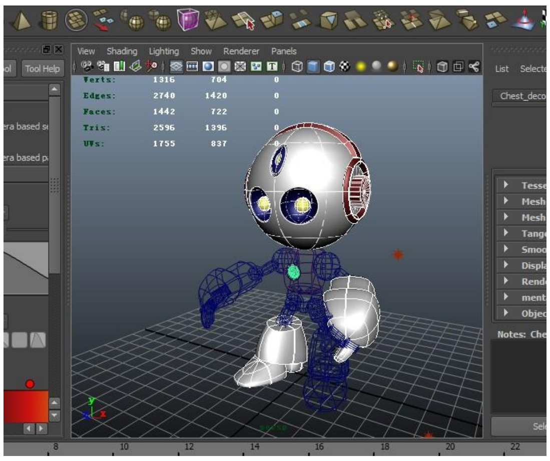


Figure 1.1 Modeling with Autodesk Maya

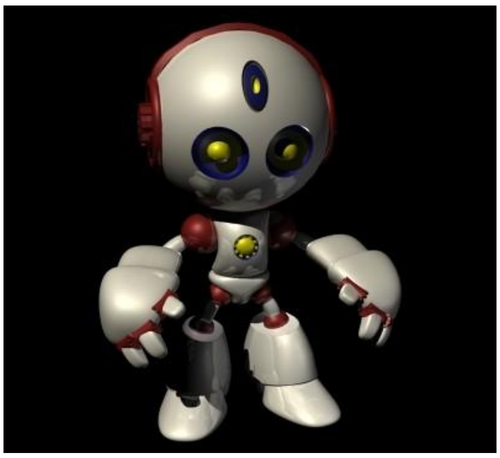


Figure 1.2 Autodesk Maya rendered picture

- **Rendering** is a term inherited from art and can create shaded image which looks realistic from 3D computer graphic models.

- **Animation** is a technique to bring the static images to life. It often means animation technique create an illusion of motion through a sequence of images with a key issue on the time.

Some other areas also related to computer graphics, but it is not a common agreement for all the practitioners that they are the core area of computer graphics. The examples for such areas can be [1]:

- **User interaction** deals with the interface between input/output devices and the user. The most common input devices can be keyboard and mouse while the output device can be the computer screen. The interface here provides visual interactive tools to accomplish the input operations and gives user feedback in imagery.

- **Virtual reality** creates an immersive virtual 3D world. Typically, the user is allowed to interact with the camera view and the viewer's position in the virtual coordinate. So, stereo graphics and the response to the user's interaction are always required.

- **Visualization** attempts to give users insight via visual display. Often there are graphic issues to be addressed in a visualization problem [1].

- **Image processing** deals with the manipulation of 2D images such as noise removing, segmentation, and image sharpening. It is often used in both field of graphics and vision.

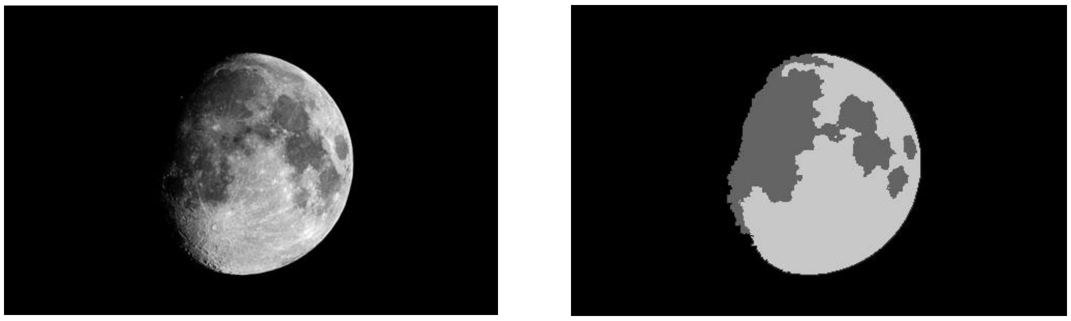


Figure 1.3 Segmentation of image processing

- **3D scanning** uses range-finding technology to create measured 3D models. Such models are useful for creating rich imagery, and the processing of such models often requires graphics algorithms [1].

1.2 Major Applications of Computer Graphics

Computer graphics has been widely applied to many areas in our real world, but the following industries consist of the major consumers of computer graphics technology:

- **Video game** is an electronic game that involves human interaction with a user interface to generate visual feedback on a video device. It increasingly use advanced 3D models, user interface, and rendering technologies.

- **Cartoon** is a form of two-dimensional illustrated visual art. It often rendered directly from 3D models with the handling of a sequence of images called frames on the time axis.

- **Film special effects** are the illusions or tricks of the eyes to simulate the image events in a virtual world. This allows the films to have some scenes which are impossible to be created in the real world. It uses almost all kinds of computer graphics technology like modeling, rendering, and animation.

- **CAD/CAM** is the abbreviation for computer-aided design and computer-aided manufacturing. Computer graphics technology is often to be integrated as a development tools on the computer. These development tools or software can help the designer guide the manufacturing procedure and visually design their products with friendly user interactions. For example, Autodesk 3ds Max is commonly used in architecture design because the user can easily create 3D building models in this software’s workspace by clicking and dragging the mouse.

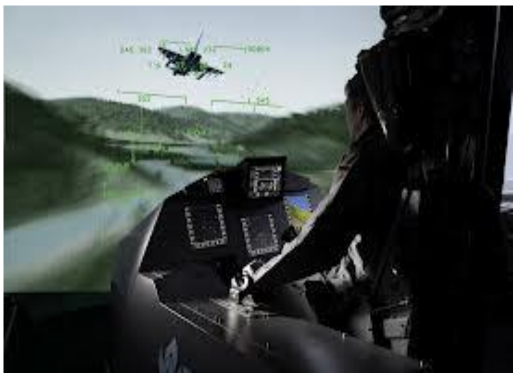


Figure 1.4 Projection system from Antycipe simulation

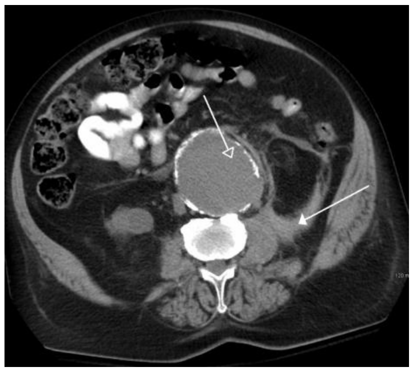


Figure 1.5 A CT scan image showing a ruptured abdominal aortic aneurysm

- **Simulation** is the imitation of the operation of a real-world process or system over time. It can be thought of as an accurate video gaming. For example, a rocket simulator uses very sophisticate computer graphics technology to imitate the route of its flying. Such application can be very useful in initial trainings of some safety critical domains like the specific fire-fighting situations which are too dangerous to create physically.

- **Medical imaging** is the technique and process attempts to create images of the human body for clinical purposes or medical science. Computer graphics is used to create meaningful shaded images with saliency which can be used by the doctor to diagnose or examine diseases.

1.3 Aims and Objectives

The novelty of this project is providing a simple and efficient 3D interactive graphic application to users such as researchers, managers, students etc. who frequently need to insert some 3D structure charts or flow charts into their papers and reports. As we can see, using 2D drawing packages in the current market to draw 3D diagrams consisting of basic primitives and connecting arrows etc with realistic looking 3D perspective is hard, unless you are good at drawing. It is one thought to consider using CAD or high end graphics tools, such as Maya, but they are not designed for this purpose, too complicated for unprofessional users to master them, particularly in that they do not produce vector 2D output; they only render scenes which is not suitable for including in reports, papers etc. As a result, a simple, fast, friendly 3D interactive graphic application will draw such group of users' attention.

The aim of this project is to design and implement a 3D graphic interactive application in which users can easily create 3D diagrams, charts, graphs, etc. Users use the basic interactive devices, the keyboard and the mouse, to accomplish the drawing task by picking up the primitives in the tool box and dragging them into the workspace. The output result of the application is a 2D vector image with perspective view of the 3D primitives created, and the perspective view can be a line drawing projection from arbitrary direction.

This project mainly focuses on application design and development, it is a type I project with the following key deliverables:

- Firstly, this application should be able allow users work in a 3D workspace which is a window to show the 3D model. In the window, the camera can be manipulated to an arbitrary view by moving, rotating and scaling actions. By default, the perspective, top and side views are provided to users.

- In order to create and edit the 3D model, this drawing package should also include a series of tools used to fast create basic elements like cubes, spheres, cones, cylinders and so on. After the elements have been created, they should be allowed to be translated, rotated, and scaled.

- When the 3D figures have been finished, the application can provide function to generate 2D vector images which is the 3D models projection from any perspective view.

2. Fundamental Theories

2.1 Constructing a 3D scene

This project is called ‘3D projection drawing package’ which means implementing 3D drawing is one of the main task. With basic theory about vertices, curves, surfaces, color, etc. in computer graphics, a data structure to consist these basic elements into the required models like cubes, arrows is urgently needed. And the data structure for the whole scene to manage the relationship among the models is useful too.

2.1.1 The Vertex-Face data structure for polyhedral

In 3D graphics application, a 3D model can be very complex which belongs to a large structure and represented by large collection of polygons. The properties for a simple polyhedron have been listed below [2]:

- Each edge connects exactly two vertices, and is the boundary between exactly two faces.
- Each vertex is a meeting point for the at least three edges.
- No two faces intersect, except along their common edge.

The number of edges (E), faces (F), and vertices (V) in the polyhedron always obey the rule of Euler (no hole in this polygon):

$$V - E + F = 2$$

For the polyhedron above, the vertex-face data structure presented a polygon by storing all the vertices into an array. And for each face, points out which vertices contribute to form it. The following figure shows an example of a simple polygon:

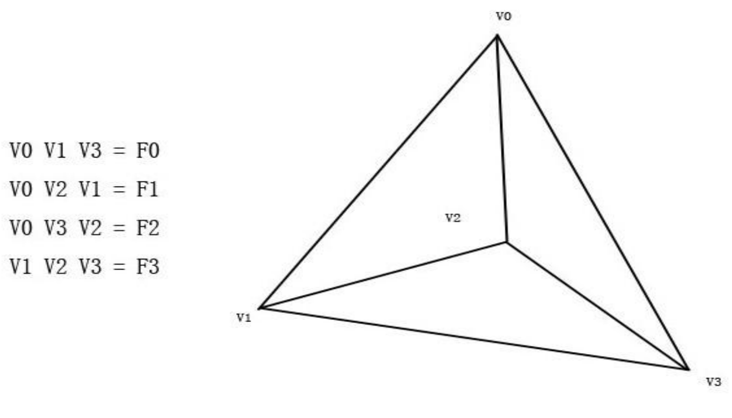


Figure 2.1 Vertex-face data structure

A vertex can be presented by a 3 dimension vector (x, y, z). So that a face can be presented by the vertices contribute to form it just like the front face (F0) in the figure, vertex 0 (V0), vertex 1 (V1), and vertex 3 (V3) in charge of consisting it. The front face (F0) therefore can be presented as: F0 = V0 V1 V3. Notice the order of the vertices for presenting the face, it stands for the normal of the face and decides which side of the face can be seen. With all the faces stored in an array, this data structure demonstrates a straightforward way to implement a polyhedron. The following list shows the data structure for the polyhedron above:

Figure 2.2

Vertices	Faces
V0	V0 V1 V3
V1	V0 V2 V1
V2	V0 V3 V2
V3	V1 V2 V3

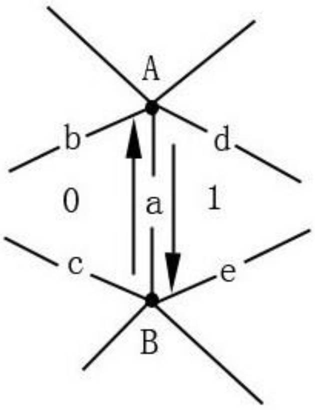
The vertex–face data structure is a simple and easy understanding way for representing a polyhedron. Because the polyhedron is only stored as a collection of separate polygons, this data structure is neither efficient nor helpful in modeling of complex polyhedrons. For example, each vertex would be stored at least three times and each edge twice.

2.1.2 The Winged Edge data structure

The Winged Edge data structure defined by Baumgart (1975) gives some improvements to presenting a polyhedron. It provides the convenience of querying in the follow situations [2]:

- For any face, find all of the edges, traversed in (counter)clockwise order;
- For any face, traverse all of the vertices;
- For any vertex, find all the faces that meet at that vertex;
- For any vertex, find all the edges that meet at that vertex;
- For any edge, find its two faces;
- For any edge, find its two vertices;
- For any edge, find next edge on a face in a certain order.

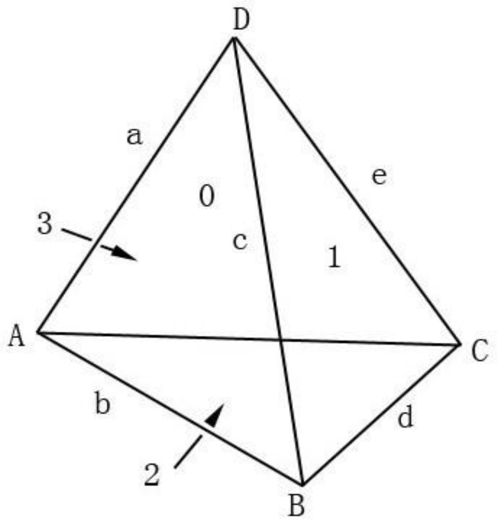
Edges are the first-class citizen of the winged edged data structure. The implementation is illustrated in the following two figures:



Edge	Vertex1	Vertex2	Face left	Face right	Pred left	Succ left	Pred right	Succ right
a	B	A	0	1	c	b	d	e

Figure 2.3 An edge in a winged edge data structure

This is the first figure, shows the elements stored with edge (a) are the face to the left of the edge, the face to the right of the edge, and the previous and successor edged in the traversal of each of those faces. The next figure gives an example of how a polyhedron is represented in this data structure:



Edge	Vertex1	Vertex2	Face left	Face right	Pred left	Succ left	Pred right	Succ right
a	A	D	3	0	f	e	c	b
b	A	B	0	2	a	c	D	f
c	B	D	0	1	b	a	e	d
d	B	C	1	2	c	e	f	b
e	C	D	1	3	d	c	a	f
f	C	A	3	2	e	a	b	d

Vertex	Edge
A	a
B	d
C	d
D	e

Face	Edge
0	a
1	c
2	d
3	a

Figure 2.4 Winged Edge data structure

Above figure and tables demonstrate how a polyhedron is stored using winged edge data structure. The two small tables are not unique. Any associated edge can be stored here, so that searching for edges from vertices or faces becomes quick. In fact, winged-edge data structure makes the desired queries in constant time. Any object stored in this data structure is specified by their geometry (vertices) and topology (relationship between the edges and faces) [2].

2.1.3 The scene hierarchy

With well structured polygon models in the scene, there are also requirements for data structures to organize to relationship among the primitives. A reasonable example for this situation can be a robot's body with different components like arms, legs, head, body, etc. The users dealing with operations on a particular polyhedron always prefer to work on its local coordinates. Think about a translation p, a rotation ø of the whole robot, and addition operation of rotation of the right arm. The transformation matrices:

M1 = translate(p);
M2 = rotate(ø);
M3 = M1 * M2;

Apply M3 to the polyhedron of the body to implement the transformation. However, when the right arm wants to be transformed, its local coordinate has already been changed because of the transformation of the whole body. So the matrices for the arm operation:

M4 = rotate(Θ);
M5 = M3 * M4;

This time, it is matrix M5 should be applied to the right arm. We can easily found the relationship of the body and the right arm from the equation. In the real world, the movement of the whole body has constraint on arms and legs which are body's belonging parts. The relationship can be explained as parents and children, more specifically some certain constraints on transformations such as only constraints on rotation or translation.

It is common requirement for primitives to be transformed frequently. The relationships of different models in the scene can be complex. In order to well organize the scene, a tree structure to store such relationship is needed. And a matrix stack to store the transformation history will bring much convenience to the interactive graphics application. The matrix stack is manipulated by using push and pop operations that added or deleted matrices from the right hand side of the stack. In my example, for the father node model like the body, call push(M1) and push(M2), while for the right arm which is the child node model, call push(M4). While doing the operation of any model, not only the transformation matrices of itself, but also its higher level node model' transformation matrices are always applied to it. As the result of user actions or changes in the application state, the scene structure can be changed by adding or removing nodes [9].

2.2 Projection for Camera Model

Projection is the process of representing 3D scene on a 2D plane, which is a basic and common way of presenting visual output to the user in interactive graphics applications. The two fundamentally different ways of doing projection are orthographic projection and perspective projection.

2.2.1 Orthographic Projection

The orthographic projection is also called orthographic parallel projection, in which the direction of view is always parallel to one of the axe, the projection plane is at right angles to the direction of view, and viewpoint is from infinitely far away. The following figure shows orthographic projection's way of working.

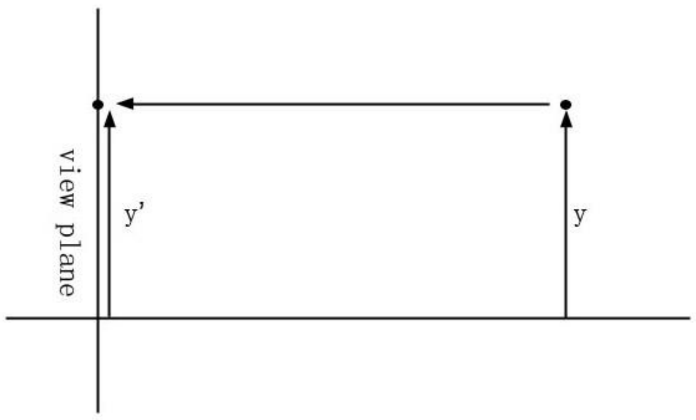


Figure 2.5 orthographic projection

Limit the projection objects within canonical view volume often brings convenience. The canonical view volume is a cube with side of length two centered at the origin. All the coordinates of 3D points in this cube have been specified as (x, y, z) ∈ [-1,1]3. For a screen made up of m by n pixels, x = -1 should be projected to the left side of the screen and y = -1 should be projected to bottom of the screen. The coordinates have 0.5 overshoot form the pixel center.

In orthographic projection, the simplest case will be the canonical view volume is a axis-aligned box [l , r] × [b ,t] ×[n , f]:

x= l is the left plane	x= r is the right plane
y = b is the bottom plane	y = t is the top plane
z = n is the near plane	z = f is the far plane

For any point in the world coordinate, a matrix M0 to map it into the canonical cube should be calculated [1]:

$$M0 = \begin{bmatrix} 2/(r-l) & 0 & 0 & 0 \\ 0 & 2/(t-b) & 0 & 0 \\ 0 & 0 & 2/(n-f) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(l+r)/2 \\ 0 & 1 & 0 & -(b+t)/2 \\ 0 & 0 & 1 & -(n+f)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} \mathrm{xcanonical} \\ \mathrm{ycanonical} \\ \mathrm{zcanonical} \\ 1 \end{bmatrix} = M0 \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (2)$$

From canonical view volume space to the window output, another matrix M1 is presented below [1]:

$$M1 = \begin{bmatrix} m/2 & 0 & 0 & (m-1)/2 \\ 0 & n/2 & 0 & (n-1)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$$\begin{bmatrix} \mathrm{xpixel} \\ \mathrm{ypixel} \\ \mathrm{zcanonical} \\ 1 \end{bmatrix} = M1 \begin{bmatrix} \mathrm{xcanonical} \\ \mathrm{ycanonical} \\ \mathrm{zcanonical} \\ 1 \end{bmatrix} \quad (4)$$

Despite the simple axis-aligned view, an arbitrary view is commonly required in many 3D graphics applications. The conventions to specify viewer' position and orientation can be [1]:

- Eye point e,
- The gaze direction g,
- The view-up vector t.
-

With these parameters, the local coordinate of the camera can be calculated [1]:

$$w = - g/|g|, \quad u = (t \times w)/|t \times w|, \quad v = w \times u.$$

In this situation, the object should be drawn into uvw coordinate. A matrix M2 implement the transform from as moving eye point to the origin and aligning uve to xyz [1]:

$$M2 = \begin{bmatrix} xu & yu & zu & 0 \\ xv & yv & zv & 0 \\ xw & yw & zw & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -xe \\ 0 & 1 & 0 & -ye \\ 0 & 0 & 1 & -ze \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

To summarize the procedure of implementing orthographic projection above, a simple algorithm is presented. The code to draw 3D lines segments with end points a and b in 2D window [1]:

```
Compute M = M1 * M0 * M2;
For each line segment (a , b) do
    p = M * a;
    q = M * b;
    drawline(xp, yp, xq, yq)
```

2.2.2 Perspective Projection

In order to get the projection to look right as the real world, the perspective projection is needed. The perspective projection’s way of working is illustrated in the following figure. The farther the object is from the viewer, the smaller it is. This basic rule will be followed in the perspective projection if the objects are projected directly toward the eye point, and they are displayed where the ray of projection hit the view plane in front of the eye point.

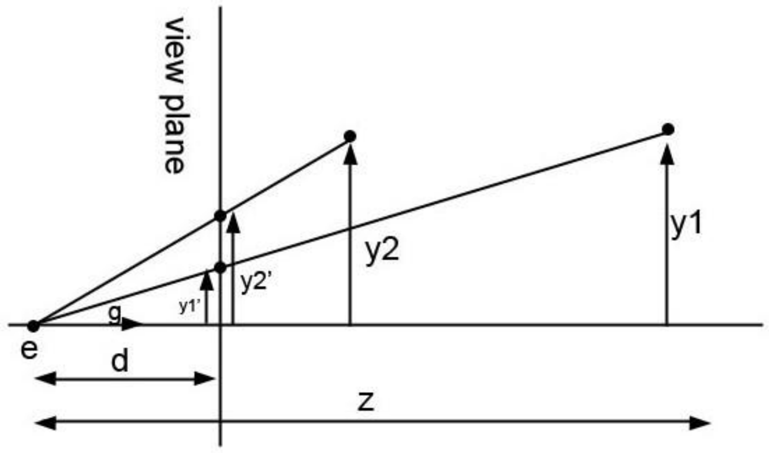


Figure 2.6 perspective projection

To implement perspective projection, it is simple to just multiply another matrix into the composite matrix in the arbitrary orthographic projection. Since the eye point has been specified, to seamlessly implement it based on the work of orthographic projection, the view plane can be set at z = n. One property of this way is the points on the z = n plane remain unchanged during the projection. For other points not on the z = n plane, their x and y position will be scaled by their z position as the theory showed in the figure.

A fourth coordinate as the homogeneous coordinate h is presented to be in charge of encoding how much the other three coordinates have been scaled. The perspective matrix be constructed as [1]:

$$\mathrm{M3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (n+f)/n & -f \\ 0 & 0 & 1/n & 0 \end{bmatrix} \quad (6)$$

$$\mathrm{M3}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \frac{z(n+f)}{n} - f \\ 1 \end{bmatrix} \text{homogenize to be} \begin{bmatrix} nx/z \\ ny/z \\ n+f-fn/z \\ 1 \end{bmatrix} \quad (7)$$

Matrix M3 can be prettier as [1]:

$$\mathrm{M3} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (8)$$

As a result, when the perspective matrix is applied to the procedure of implementing the orthographic projection, it becomes the implementation of perspective projection, the basic algorithm can be stated as [1]:

```
Compute M = M1 * M0 * M3 * M2
   For each line segment (a, b) do
      P = M*a
      Q = M*b
      Drawline(xp/hp, yp/hp, xq/hq, yq/hq)
```

2.3 Interactive Graphic Application Models

Differs from the internal control model (or we can say control-driven model), which the overall control remains interval to the main() function during the entire life time of the program, such kind of interactive graphic applications will be more appropriate to be designed by the model of event-driven programming [1].

The main advantage of control-driven programming is that it is fairly straightforward to translate a verbal description of a solution to a program control structure. A graphic application can be verbalized as following example:

```
While the user does not quit;
Parse and execute the user’s command;
Update all the data;
Update the scene in the viewport;
Waiting for the user’s next command
```

Although, the statements above can be easily translated into code, it has to be concerned with the efficiency issues as well as the potential for increased complexity.

Efficiency Concerns come from when the user interacts with the application in burst of commands and then spend a period of time to examine the output. This is a very common case in graphic applications, for example, the user is working with a word processor, suddenly types in hundreds of words and takes a period of time for reading. The computer will work very fast for this application even though the user has stopped inputting because of the continuous while-loop in this control-driven model. So, in this model, the program is always actively running and wasting machine resources.

Complexity concerns are mainly because of the entire solution is in the main function. One of the situations demonstrates this is that all the relevant user interactions must be parsed and handled. This task can be executed by the switch statement. However, many actions performed by the user in modern software are actually non-application specific. Such kind of increasing requirements rapidly becomes limitations to the dataflow of graphics applications. To implement interactions with the control-driven model also becomes a burden.

Event-driven programming remedies the efficiency and complexity concerns with the GUI system. A function used to loop all the event is provided to define all the central control structure for event-driven model and this function cannot be changed by the user application. Since this means the overall control of the application is external to the program code, the GUI system will automatically takes care of mundane and standard user actions. An efficient GUI should remain idle by default which does not take machine recourses and only become active in the presence of interesting activities [1].

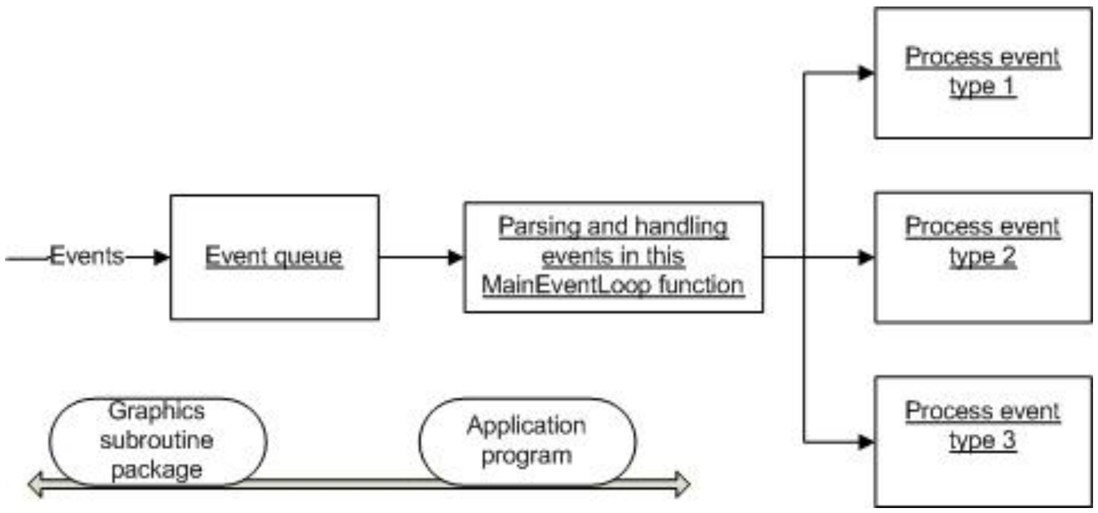


Figure 2.7 Demonstrates the working procedure of MainEventLoop() function

In the event-driven model, the program code handling different types of input actions is written in the function provided by the GUI API. The structure of the code in this MainEventLoop() function seems similar to the main() function in the control-driven model. Here, the difference between these to event handling functions will be discussed [1]:

- System initialization function provided by GUI is a mechanism defined to invoke the user programs within the MainEventLoop() function and it will initialize the application state and to register event service routines.

- Continuous outer loop: All the event-driven programs use this general control structure, user program is required to override appropriate event service routines and terminate the program within the service routine.

- Stop and wait: Instead of actively polling the user for actions which wastes machine resources, the MainEventLoop() provided by GUI stops the entire application process and will only be wake up by operation system calls because the application is in the presence of relevant user actions.

- Event and central parsing switch statement: All the possible actions that a user can perform is included in this statement. Associated with each event is a default behavior and a toggle that allows applications to override the default behavior.

2.4 Frameworks of 3D Graphic Applications

The Modelview-controller (MVC) Architecture mentioned by Peter Shirley in 2005 in his book 'Fundamentals of Computer Graphics' will be discussed in this section. This architecture is about developing strategies for organizing and grouping the functions in the event-control model into components. Different components will collaborate to support the functionality of the interactive graphics applications.

Interactive graphics applications can be described as applications allow user to interactively update and provide real-time visualization of their internal states. In the MVC framework, the model is the application state, the view is responsible for implementing the visualization of the model components to the users, and the controller supports the interaction between the model and the users. Figure shows the relationship between each component and the way they collaborate to work.

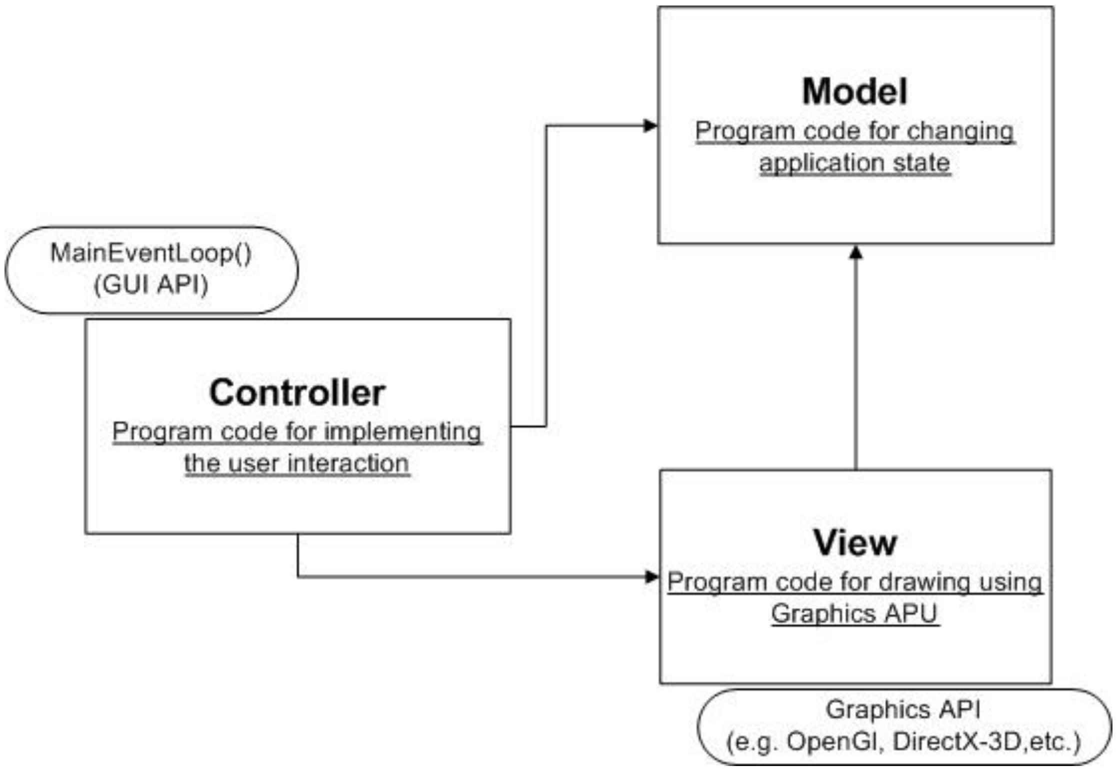

Model
Program code for changing application state
MainEventLoop() (GUI API)
Controller
Program code for implementing the user interaction
View
Program code for drawing using Graphics APU
Graphics API (e.g. OpenGl, DirectX-3D,etc.)

Figure 2.8 Components of an interactive graphics application

The model component defines the application state and implements the interface function for them. Because the graphics primitives represent part of the applications state in graphics application, part of the model component also needs to use graphics API. For the whole system, only the model realize all the details of the application state which means the redraw of the entire application can only be done in the model component.

It is understandable the view model is responsible for drawing the scene which consists of a series of different graphical primitives. Take some of the input from the controller component, translate the input into useful data in supporting with changing and updating the scene is the main task of this component. The arrow from the view to the model indicates that if it is the redraw of the application state, the operation should be executed in the mode component.

The controller component supports the interaction between user and the application. By using the MainEventLoop function which is the function provided by the GUI API, the controller parses and handles the input event queue. The arrow from the controller to the model means some of the output eventually tells the model component to change their application state. The other arrow to the view shows other input events asks the view the update their scene because the build a new primitive or the transformation of the scene may be a requirement from the user.

2.5 Principles and Rules for Interaction Design

T. H. Nelson said in 1977 in his book “The Home Computer Revolution”:

> Designing an object to be simple and clear takes at least twice as long as the usual way. It requires concentration at the outset on how a clear and simple system would work, followed by the steps required to make it come out that way – steps which are often much harder and more complex than ordinary ones. It also requires relentless pursuit of that simplicity even when obstacles appear which would seem to stand in the way of that simplicity [5].

The above words described the human factors of interactive software, and this principles and process is applied by application developers who are producing exciting interactive systems.

Computer-aided design and manufacturing tools are in one of the growing interest of human factors issues. Although practitioners and researchers in many fields make contribution in the improvement of human-computer interaction, software designers and developers are mainly exploring [5]:

- Menu selection techniques
- Command, parametric, and query languages
- Use of graphics, animation, and color
- Direct manipulation
- Natural language facilities
- Error handling, messages, and prevention
- Screen formatting.

2.5.1 Primary Design Goals

A well designed system provides users positive feelings of success, creates an environment that can help users to carried out their task effortlessly. Using such an effective system, the user will not be encumbered by the computer and can predict what will be going on with each step of their operation, hardly feel the existing of the system, totally concentrate on their work. Such a perfect designed system requires not only hard work of the designer, but also supporting theories. Successful designers can go beyond the simple concept of "user friendliness", and have a deep understanding of different kind of users' characteristics.

■ Proper functionality

The first step is to make sure the necessary functionality. Task analysis is the central part to achieve the goals, because inadequate functionality always leads to the frustration of the user no matter how well the human user interface is designed. Tasks frequently used are easy to be included in the checklist, but some occasional tasks, exceptional tasks for emergency situation, and the repair tasks to cope with system errors are easy to be ignored. Opposite to the problems of inadequate functionality, excessive functionality also causes troubles, because the clutter and complexity make implementation, maintenance, learning, and usage more difficult [5], and lead to more mistakes.

■ Reliability, availability, security, and integrity

The second step is ensuring proper system reliability. The software structure and hardware support must ensure high availability, ease of maintenance, and correct performance [5]. The developer should also pay attention to ensuring privacy, security, and information integrity [5]. The system must be able to protect from unwarranted access, inadvertent destruction of data, or malicious tampering [5].

■ Schedules and budgets

The third step is to have a plan which is on schedule and within budget. The delayed delivery of the software may cause the users to choose the alternative software in this high competitive market environment. If the implementation, test, and maintenance of the system are too costly, it causes higher price of the system, and the customer's resistance to accept the price gives chance to the competitor to capture the market.

2.5.2 Rules for Interaction Design

The principles used to guide the design of interactive systems mentioned by Ben Shneiderman in his book 'Designing the User Interface – Strategies for Effective Human-Computer Interaction' are listed and discussed below [5]:

- Strive for consistency.
  This is the most frequently violated principle, yet the easiest principle to repair and avoid. Terminology in the application needs to be explained by using prompts, menus, and help screens. The consistent sequences of actions should be required in similar situation and the consistent commands should also be employed throughout [5].

- Enable frequent users to use short cuts
  As the frequency of use increases, the users are getting familiar with the system and require a higher pace of operations. In such situation, the number of actions in frequent used interactions must be reduced, a more direct and simple way of interaction should be provided, such as using special keys, hidden commands or shortcuts.

- Offer informative feedback
  For every operator action, it is necessary to generate some system feedback. The response of the system let the user knows how the system performs the task. Especially for some infrequent used and important operations, the feedback must be substantial and obvious, while for the frequent used and minor operations, it is better to make the feedback modest.

- Group the sequence of actions
  A sequence of related actions should be organized as a group and generate informative feedbacks to make the user know the progress of the operation. And such feedback makes the user feel the system reliable.

- Offer simple error handling
  The system should be designed to protect and prevent the user from making serious errors. And there should be modules in the system in charge of error detection and handling. When an error is made, the user will be prompted about the error in a comprehensible mechanism. It is better for the error to be repaired in the way that just let the user fix the faulty part but not redo the whole actions

- Permit easy reversal of actions
  As much as possible, the actions should be reversible. Always allow the user to undo their current action or redo their former action, this alleviates the anxiety of repeating the same work, and encourages the user to explore their unfamiliar parts in the system.

2.5.3 Interaction Styles

After the task analysis has been completed, the designer starts to design the interaction of the system. There are many different kinds of interactions, and some of the most common used interaction styles are listed below:

- Menu selection: The users read a list of items, select the one most appropriate to their task, apply the syntax to indicate their selection, confirm the choice, initiate the action, and observe the effect [5].

- Form fill-in: Users see a display of related fields, move a cursor among the fields, and enter data where desired [5].

- Command-language: For expert frequent users who often derive great satisfaction from mastering a complex set of semantics and syntax, command language provides strong feelings of locus of control and initiative [5].

- Natural language: This kind of interaction usually provides little context for issuing the next command, frequently requires "clarification dialog", and may be slower and more cumbersome than the alternatives [5]. But when the user's scope is limited and the command language training is inhibited, the chances for natural language come.

- Direct manipulation: Present a visible set of representations of objects and actions, and the cursor motion devices to select, the users' tasks have been greatly simplified. This style may be the most appropriate for diversity of tasks and users.

The conclusion and summary of the above interaction styles is given by Ben Shneiderman in his book 'Designing the User Interface – Strategies for Effective Human-Computer Interaction'. In the table, designers can easily make choices from the different interaction styles according to their task analysis and requirements.

ADVANTAGES	DISADVANTAGES
Menu Selection	
Shortens leaning Reduces keystrokes Structures decision-making Permits use of dialog management tools	Danger of many menus May slow frequent users Consumes screen space Requires rapid display rate
Form Fill-in	
Simplifies data entry Requires modest training Assistance is convenient Permits use of form Management tools	Consumes screen space
Command Language	
Flexibility Appeals to “power” users Supports user initiative Convenient for creating user defined macros	Poor error handling Requires substantial training and memorization
Natural Language	
Relieves burdens of leaning syntax	Requires clarification dialog May require more keystrokes May not show context unpredictable
Direct Manipulation	
Visually represents task concepts Easy to learn Easy to retain Errors can be avoided Encourages exploration High subjective satisfaction	May be hard to program May require graphics display and pointing devices

3. Requirement Analysis

Computer graphics today is largely interactive: The user controls the contents, structure, and appearance of objects and of their displayed images by using input devices [3]. Interaction between humans and such regular desktop devices involves the participants making actions with the mouse and keyboard that activate a variety of windows, icons, and menus [2].

With the appearance of interactive graphics applications, one of the major applications of computer graphics today is to create 2D and 3D graphs of mathematical, physical, and economic functions; histograms, bar and pie charts; task-scheduling charts; inventory and production charts [3]. As mentioned in the section of ‘aims and objectives’, this 3D projection drawing package is just a typical kind of interactive graphics application for users to interactively build 3D objects and projected into 2D images.

There are many different kinds of 3D graphics software in current market, however, it is still meaningful to develop this 3D drawing package. In the next section of this thesis, the comparison between this project and the existing software will discuss the reason and motivation of developing this 3D projection drawing package.

- Autodesk 3ds Max and Maya

  They mainly aim at making complex 3D models, 3D animation, and high quality rendered 2D images, and it is developed and produced by Autodesk Media and Entertainment. 3ds Max and Maya are powerful on 3D modeling, by providing a variety kind of tools, the user can create very realistic 3D models with this software. For example, the user can create complex model with a start of very simple and basic geometry. To explain this in more details, a curve can be created by several simple clicks in 3ds Max, and then use the powerful editing tools to extrude or spin to get a 3D wine glass model. This 3D model can be textured and shaded, or even some animation can be added. The result output can be rendered in high quality images.

  As commercialized software which has experienced almost 20 years competence in the market, they are sophisticated and have already taken the lead in 3D graphics industry, their way of interaction and user interface design has always been imitated. As showed in the pictures, the 3D model can be viewed from 4 windows and each of them has a different point of view to help the user easily observe every detail of the 3D model.

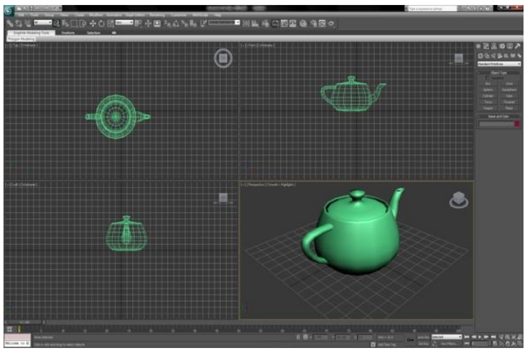


Figure 3.1 Autodesk 3ds Max

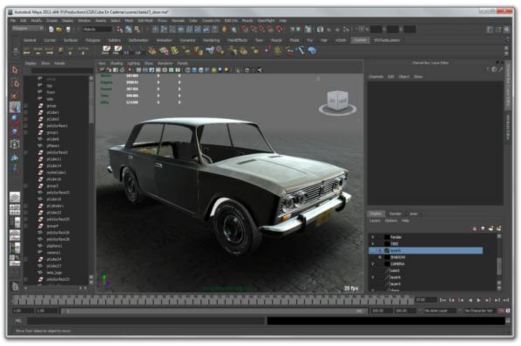


Figure 3.2 Autodesk Maya

The interactions of 3ds Max and Maya can be classic example for this project to imitate. It is easy to use 3ds Max and Maya to create 3D models and have an arbitrary perspective view to output an 2D image, but they are in bitmap file format and their effect will be wired after zooming up the image. From the aims and objectives mentioned in chapter 1, the users need an output of 2D vector images which can solve the problem of zooming. And another serious problem is that 3ds Max and Maya are professional software, too complicated for users to quickly master them without long time of training.

■ Adobe Photoshop and Illustrator

These 2 products developed by Adobe aims at 2D image designers, and they are also leaders in their fields. Photoshop is an expert in 2D image drawing, editing, and composition. Know as the most popular software to modify photos, actually, Photoshop is almost powerful enough to do any creation or modification in 2D graphics field. Compare to Photoshop, illustrator especially aims at 2D vector image drawing.

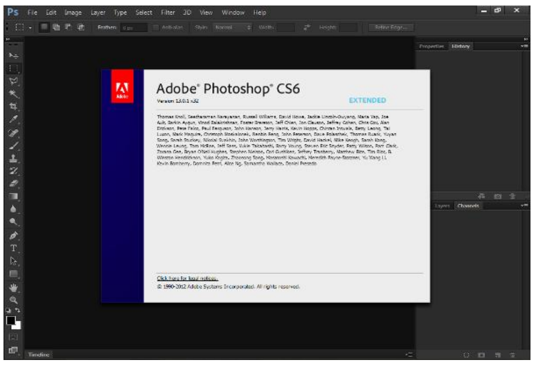


Figure 3.3 Adobe Photoshop

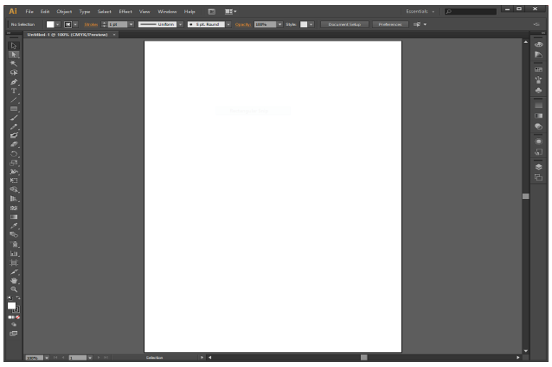


Figure 3.4 Adobe Illustrator

Both of the software provides functions to generate 2D vector images which is the aiming of this 3D projection drawing package, but none of them has any relationship with 3D graphics. If the user would like to create an objective 2D vector image which has the perspective view of 3D shapes by using Photoshop or illustrator, they must be good at 2D drawing. And the same problem as Autodesk 3ds Max and Maya, Photoshop and Illustrator are professional applications, they are powerful based on their complexity, which requires a certain time of training to master the way of using them.

- Microsoft PowerPoint and Visio

Microsoft office's software such PowerPoint and Visio has capability to draw some schematic diagrams consist of basic primitives and lines. They are effective and efficient to generate schematic diagrams or flow charts, and they are also designed in a simple way to use. Especially Microsoft Visio, which aims at drawing a variety kind of 2D schematic diagrams or flow charts, it is an excellent example for this project to imitate the way of interaction.

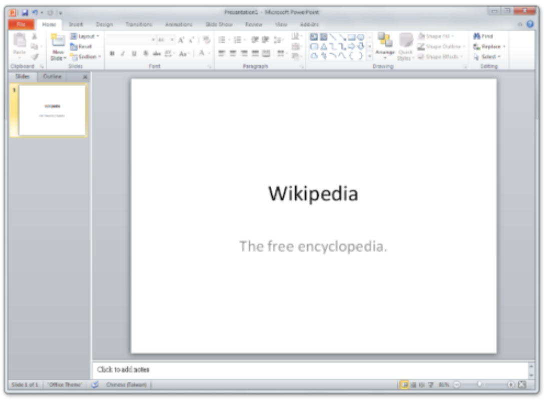


Figure 3.5 Microsoft PowerPoint

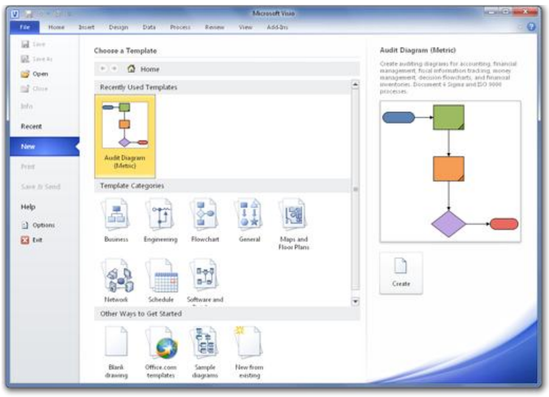


Figure 3.6 Microsoft Visio

However, the problem with both of these two applications is that they can only help users rapidly include 2D primitives in their image, but cannot manipulate the viewport in a 3 dimensional world freely. Although Microsoft Visio provides 3D primitives in its system, the view can only be set from a certain direction without any change.

In conclusion, this 3D projection drawing package will be the most suitable for creating 3D primitives and modifying them to ideal states, then projecting the 3D scene from arbitrary perspective view. The 2D vector image output of diagrams and figures will be included for reports and papers. The user interface, compare to professional software, should be concise to ensure the user can go straight forward to the aiming output.

4. Project Specification

According to the requirement analysis and principle of user interface design, the 3D projection drawing package is a regular 3D interactive graphics application, which means the participants will use keyboard and mouse as input devices and the screen will be the output device. Typical 3D Modeling tools are designed for precise control of complicated shapes, and the interfaces are generally hard for casual users to learn [12], such as Maya and 3Ds Max, but this project aims at a direct way of simple 3D drawing, the user interface should be concise and clean.

As most graphics applications today run on personal computers and workstations, the input and output environment of this project will rely on desktop window systems to manage multiple simultaneous activities, and point-and-click facilities to allow users to select menu items, icons, and objects on the screen; typing is necessary only to input text to be stored and manipulated [3].

Not only the devices should be considered, but also the interactions with user and the use of data for reference and computation should be taken into account when developing this graphics application project [13]. The following content discuss and describe the interaction and function of this project in details.

4.1 Visual display

From an application, camera control can be distinguished on the basis of whether the user exercises some degree of interactive control, or the application assumes full control of the camera itself [7]. Obviously, for an interactive graphics application, the former type control of camera should be applied. A direct control to the camera's translation, rotation and zoom in/out by using keyboard or mouse should be implemented.

A real-time display window will be implemented as the workspace and primary visual output for the user. The window displays the scene which is being created by the user. The camera view can be quickly switched among top view, side view, front/back view and perspective view.

4.2 Creating

To implement the task of 3D drawing, the function of 3D modeling should be provided by this application. A series of basic 3D geometry such cube, pyramid, and cylinder can be created in the workspace by simply clicking the relevant tool buttons.

Because of drawing figures like 3D flow chat, different types of arrows should be implemented in the same way as the geometry mentioned above. Different colors can be applied to the created object by implementing a color palette. All the elements mentioned above can be duplicated or deleted once they have been created based on the rules "Permit easy reversal of actions" in interaction design[5].

4.3 Editing

A panel to show the attributes of the selected object should be implemented so that the user can get the parameter of the object. And it should also be convenient for the user to accurately modify the already created objects by just changing the value of the parameter. Take of cube object as an example, to demonstrate the design of the elements in the attributes panel: width, height, depth, translate x, translate y, translate z (indicate the location of the cube), rotation x, rotation y, rotation z (situation of rotation since the cube has been originally created), scale x, scale y, scale z (situation of scaling since the cube has been originally created).

4.4 Manipulation

The manipulation refers to a specification of object properties (most often position and orientation, but also other attributes) [6]. In this project, manipulation is mainly in terms of selection, translation, rotation and scale. 1. Selection is the ability to look at, point at, or touch an object to indicate that it is the focus of attention [2]. The selection operation can be done by simply clicking the object's name in the object list. 2. Translation, rotation and scale in 3D can be done by using mouse and keyboard or the button on the user interface once the object is in the state of selected.

4.5 Object list

A small window to show the list of all the objects in the current scene is so useful and must be implemented. For example, when the scene becomes complex, it is difficult for the user to select the object he aims at because of the overlapping of objects. Selecting from the outliner becomes efficient at this moment.

4.6 Save and load file

The application can generate a result of 2D image file from the current perspective view of the 3D scene. The viewport of 2D result image should be the same as the 3D perspective display of the workspace and objects in the output image are showed in wire frames. For the situation the user cannot finish working at one time, the working source file can also be saved and loaded for the user to continue the undone work.

5. Implementation

5.1 API Introduction

This project aims at developing a 3D drawing package, dealing with a graphics application program interface (API) is inevitable. An API is a software interface that provides a model for how an application program can access system functionality [1], such as drawing some simple primitives into a window. While graphics libraries fulfill the functions in a mathematical way, it is also common for APIs to have a user-interface toolkit which can process the event of users' input such as press a button and return the output results to the user like display a sphere in the window. Generally speaking, developing a 3D graphics Application mainly focuses on two key issues: handling graphics calls such as "draw cube" and designing ways of interaction such as a button press.

Currently, there are mainly two dominant paradigms for APIs. The first is the integrated approach of Java where the graphics and user-interface toolkit are integrated and portable packages that are fully standardized and supported as part of the language [1]. The second is represented by Direct3D and OpenGL, where the drawing commands are part of a software library tied to a language such as C++, and the user-interface software is an independent entity that might vary from system to system [1].

The programming language chosen for this project is Java, because Java has its 3D graphics library -- 'Java3D', and the graphics user interface development library – 'Swing', and these two libraries are compatible with each other. Other reasons for choosing Java: Many programmer agree that Java is simpler that C++ to learn, write, compile, and debug is because Java has its built-in system in charge of automatic memory allocation and garbage collection while C++ requires programmer to consider about this by themselves. Java is object-oriented which emphasizes on creating and manipulating objects, this allows the programmer to focus more on the module design and make modules work together in a logical way. Another important reason is Java's platform-independent feature, this might be the most outstanding advantage because application programmed by Java has the ability to be easily moved to another computer operating system.

5.1.1 Java3D

The Java3D API is an interface for program on the display and interaction of 3D graphic. It is a collection of high-level constructs for creating and interacting with geometries, camera views, lights and textures and rendered them in a virtual universe. This API is straight forward to use and helps the programmer on interactive 3D graphics application development.

Objects like primitives, lights, and camera views can be created and manipulated by invoking the functions in Java3D, after creation, they are organized in a tree structured scene hierarchy called scene graph (showed in the picture). The whole scene graph is automatically rendered to be displayed. The optimization of the rendering performance is also handled automatically so the programmers can focus more on the modeling and interaction development.

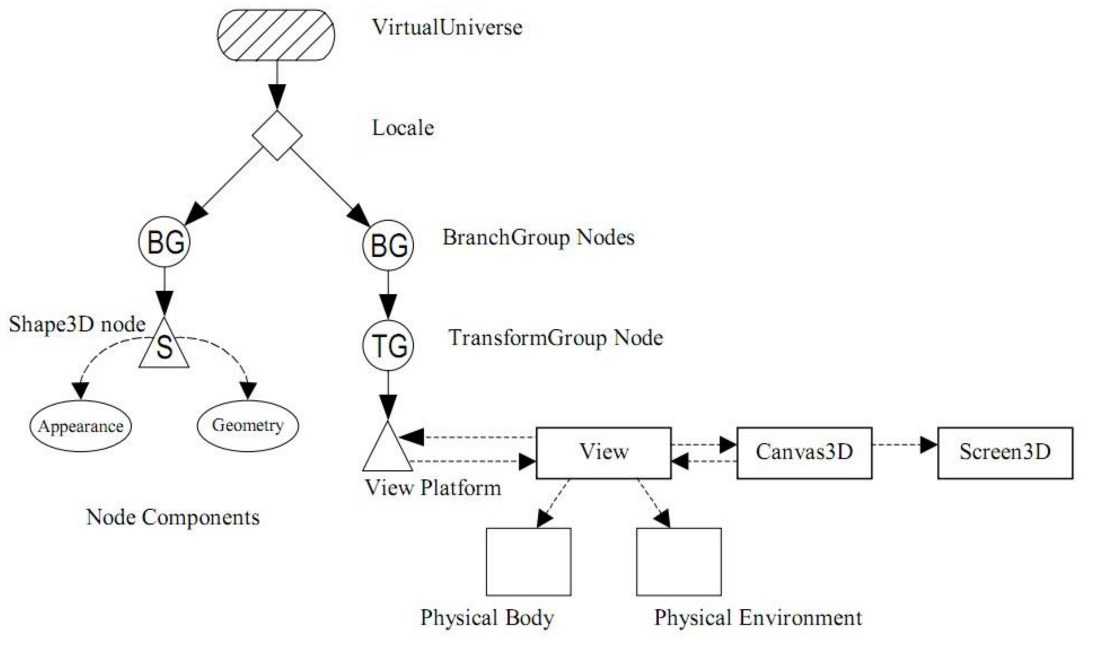


Figure 5.1 Java3D Scene Graph

The picture shows normally Virtual Universe node is the root of the whole scene graph. As a tree structure, a node can have many children but only one parent. Several Locales can be added to the Virtual Universe but only one Locale can be displayed at a time in Java3D. All the content to describe the scene should be organized in branches and add to locale, however, there should always be a branch to include objects such View Platform, View, Canvas3D and Scene3D for the reason that they are used to setup the 3D scene and the way of viewing the 3D virtual universe.

Transform Group node is always added as child of Branch Group node, and the children of Transform Group is normally used to describe single 3D model by defining its shape and appearance. Transform Group used to record all the transformation data of its child, it can be treated as a 4 by 4 matrix which include 3D translation, rotation, and scale.

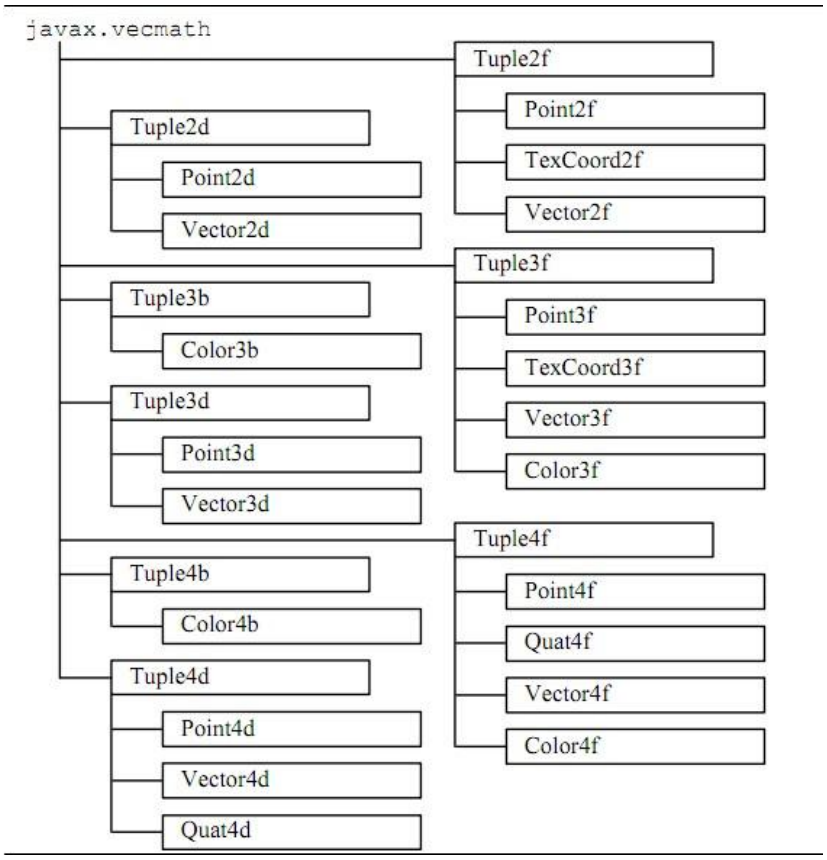


Figure 5.2 Mathematical Foundation Classes

Development on 3D graphics application always stands on some basic mathematical foundation classes. These classes provide direct way of presenting 3D graphics data like point, color, vector, and matrix. Java3D has already built-in a series of comprehensive mathematical foundation classes, they are becoming utilities to provide convenience when programmers try to model a geometries or calculate the 3D transformation. The picture below shows such classes of Java3D in an organized way. The implementation of this project will be presented by detailed explanation of how to use Java3D to achieve the design and algorithms.

5.1.2 Swing

JFC is the abbreviation for Java Foundation Classes, which includes a series of built-in features of creating user interfaces (GUIs), adding rich graphical functionality and interactivity to Java application [14]. Swing GUI components are contained in JFC which includes everything from buttons to split panes to tables [14]. Compare to the earlier GUI components provided by Java – AWT (abstract window toolkit), Swing is more sophisticated, powerful and flexible.

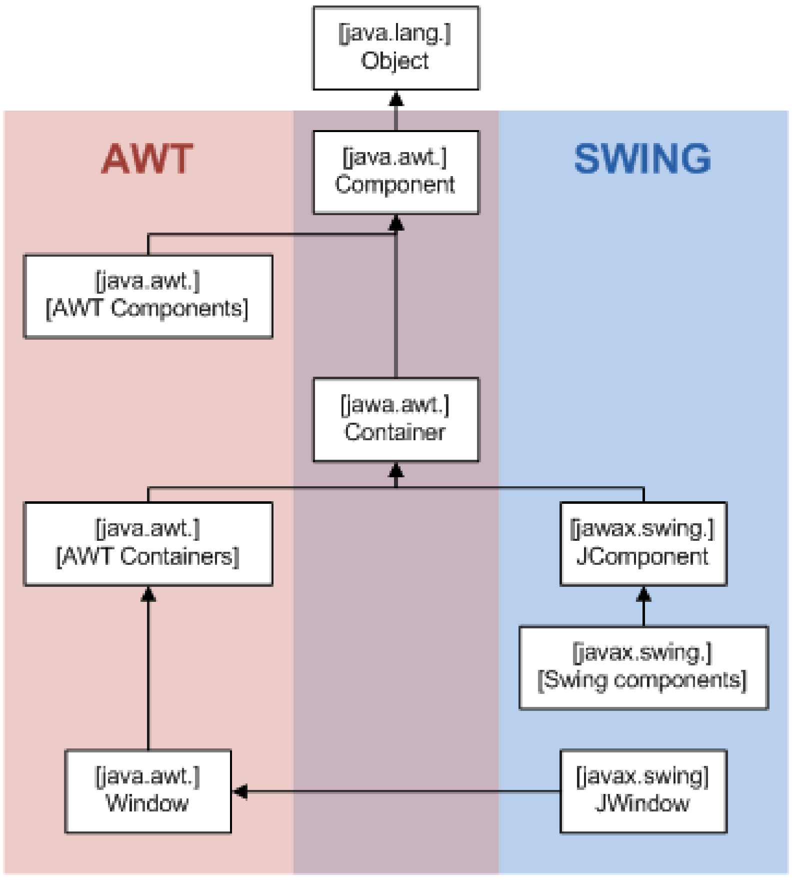


Figure 5.3 AWT Swing Class Hierarchy

The relationship between Swing and AWT is showed in the figure, this illustrates the structure and hierarchy of Swing in Java library. Because the allocation of native resources from windowing toolkit of operating system is not required by Swing, it is often described as ‘lightweight’ while in contrast, AWT is usually described as ‘heavyweight’. It is strongly discouraged to mix the using of heavy and light weight components in GUI development due to the incompatibilities. The reason for using Swing to develop the project is not only because Swing is platform-independent, but also it makes heavy use of Model-View-Controller GUI framework for Java, which follows a single-threaded programming model [15]. As mentioned in the theory section, Swing’s MVC model is quite suitable for the development of this project.

5.2 System Overall Design

As the increase of design complexity in designing multi-media and 3D graphics applications is expected to outgrow the design power with the existent design methodology in the near future, the system-level design methodology gains significant research activities [8]. Based on the framework mentioned in section 2, the following figure shows the design of the program work flow. It is based on the event-driven model and MVC architecture. Once the application has been started, all the initialization should be done first, this include a scene without any object, all tools’ status has been set by default, and a default camera view has been set focusing on the origin of the scene. Using the event- driven model of the graphics API, and the event listener is waiting for the users' action. The user' input devices are keyboard and mouse, all the different kinds of action will be exactly defined later such as a click on the tool button, typing texts into the dialogue window, dragging to move the camera, etc. This is one of the most complex parts in the application implementation which should be done in a patient and logical way. The switch statement in the MainEventLoop() function will direct the program which branch to go.

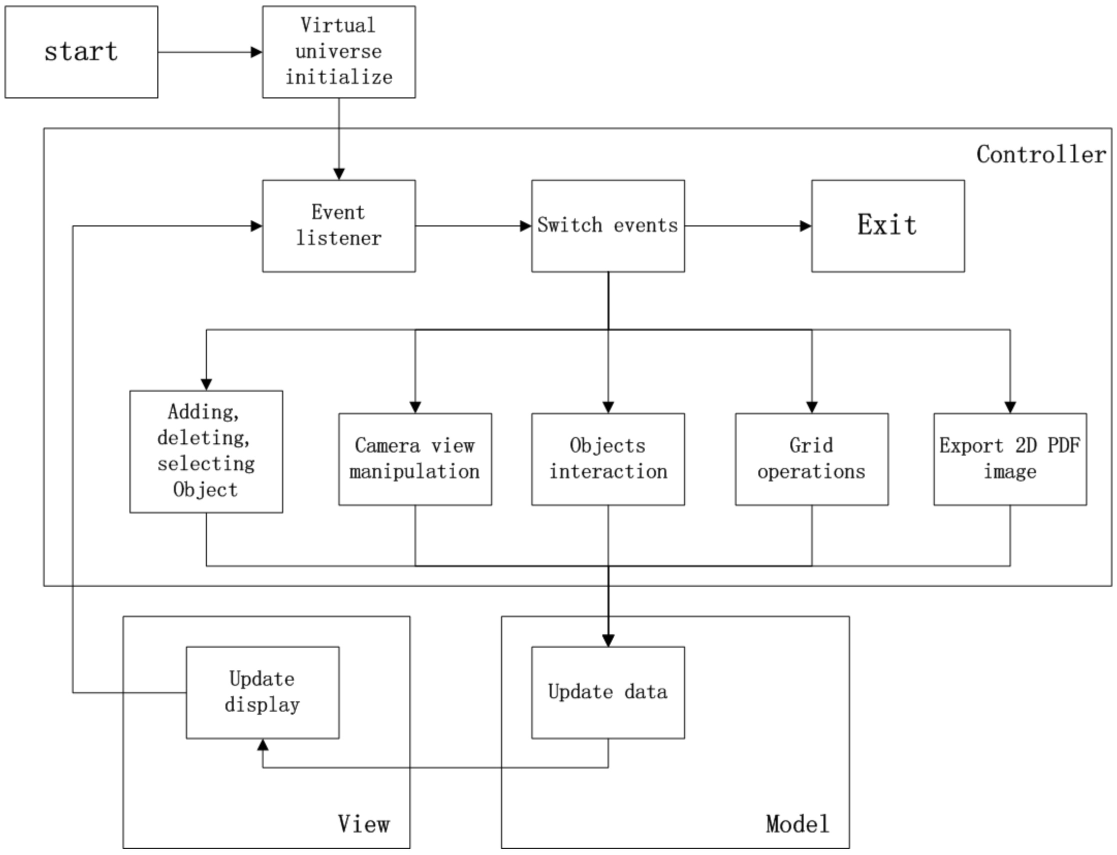


Figure 5.4 program framework

Each module, especially modules of implementing the interactions in this framework can include a series of sub-modules or functions to fulfill a common class of tasks. The details of each module will be presented in later section. The application will not be terminated unless the user clicks the exit button. If the save image or save scene button has been clicked, the relevant task will also be done. As showed in the figure, the event listener and switcher belongs to the controller model which is in charge of the interaction with the user, the update data process controls all the data information of the scene, and update display is responsible for the visual output.

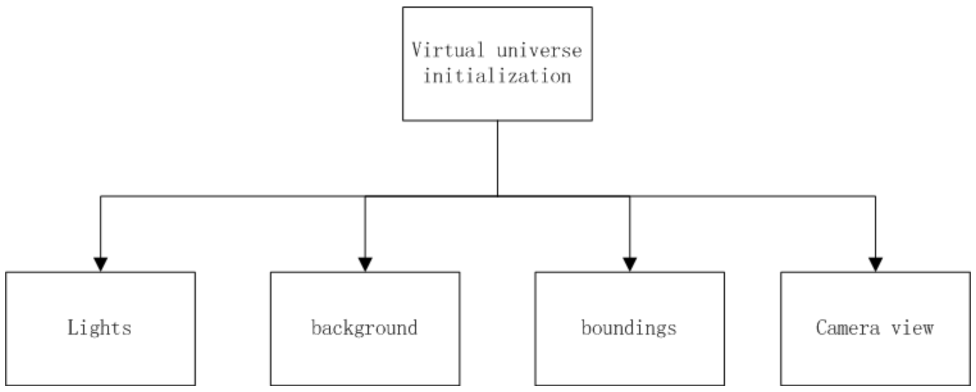


Figure 5.5 Virtual universe module

In the program framework, once the program has been started, the system should firstly set up the virtual universe which means the user can have a virtual 3D world displayed on the screen -- although no object has been created. This module must be done before any input action from the user. As showed in the above figure, the Virtual universe initialization module must finish 4 tasks:

- Camera view is a sub-module in charge of set up the default camera view. As Java3d has already automatically created the camera object in the virtual universe, in this project, this module should set up a group of suitable default parameters of the camera view for the user.

- Lights can be set up at the very beginning, because without lights, the 3D scene will be completely black, and the user can see nothing after they have created an object in the workspace.

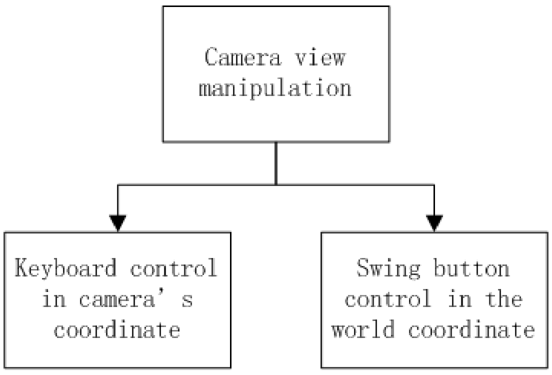
- Background means to set up the background color of the 3D virtual world. By default, it is black in Java3D, but consider about the user are using this application to draw images to be included in papers or reports, which always have white background color, it will be better to change the background color to white.

- Bounding is a sub-module to create geometry like a sphere. This sphere is not for modeling, but used to accelerate the system. Java3D will only do the calculation for the elements in the bounding sphere which should cover user's visible area.

After the initialization, the application's event listeners are activated, this task is done by Java, any input action from the user can be detected. The event can be recognized, and the program's work flow can be switched to the branch module which is responsible for responding the event input from the user. So these modules are actually implementing the interactions of the system.

Based on the common features of responding tasks, the task can be organized in some modules. The camera view manipulation module in the program framework has two sub-modules:

- The first module is to implement the function for the user to manipulate the camera view by using the keyboard. The camera view is moved, rotated according to the coordinate of itself. This is like a bird flying in the real world, it can fly up, down, forward, backward, left, and right, and rotate to change its orientation to view the world.


Camera view manipulation
Keyboard control in camera’s coordinate
Swing button control in the world coordinate
Figure 5.6 Camera view manipulation module

- The second module is a different kind of interaction from the previous one. The user can use button provided on the user interface to manipulate the camera view. But this manipulation is based on the world coordinate, the camera will always look at the origin of the virtual world where ever it moves. The type of interaction helps the user conveniently manipulates the camera view without losing the focus on the objects which are normally not far away from the origin.

If the action from the user is to create a primitive or a line, then according to the button the user clicked, add a box, cone, cylinder, sphere, or line will be displayed in the 3D scene.

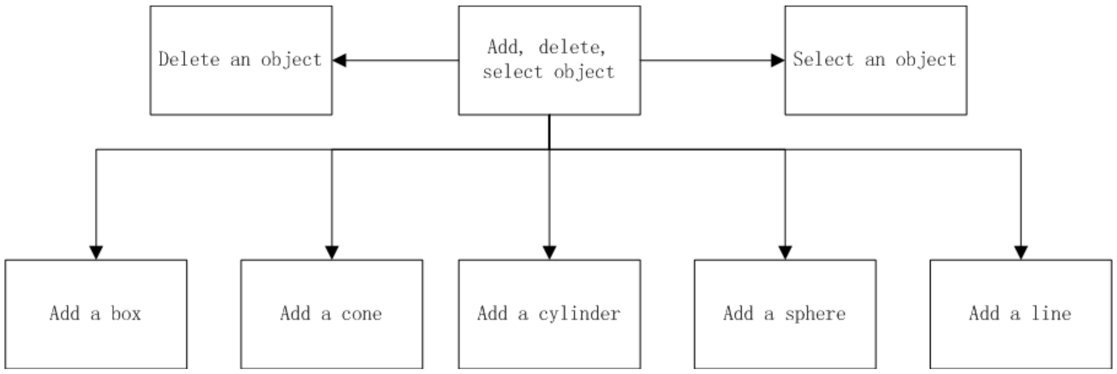


Figure 5.7 Module of Adding, deleting, and selecting an object

It is important to clearly identify which object in the 3D scene is selected, because the user can only be allowed to interact with the object selected. All the objects created in the 3D scene are recorded in a list, with the returning of an index from the list, the interaction will only be applied to the selected object. So once the user clicks the delete button, the selected object will be disappeared from the 3D scene.

The object interaction module controls the selected object's transformation ordered by the user. As showed in the figure below, similar as the camera view manipulation the user can translate, and rotate the selected object. Mouse rotate, mouse translate, and mouse scale sub-module provides a quicker way of interacting with the selected object, this only depends on the user's preference.

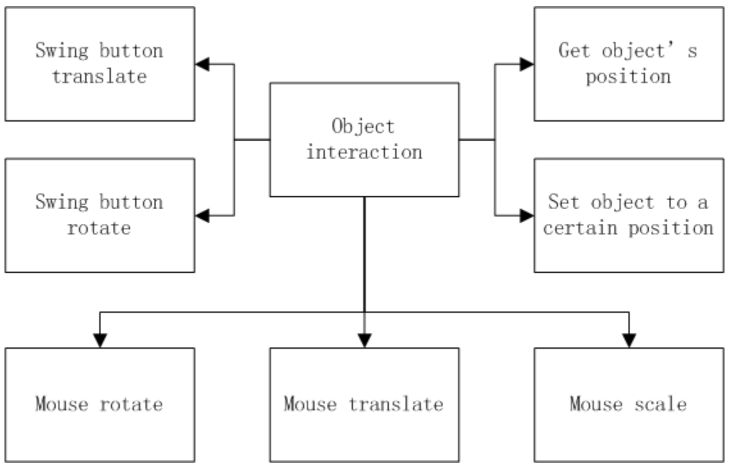


Figure 5.8 Object interaction module

Sometimes, the user wants the created object to be precisely put to a certain position. Object interaction module also provides function to read the user's command from the text input field on the user interface, the user can then specify the object's position. Another function provided by object interaction module is to show the coordinate position data to the user.

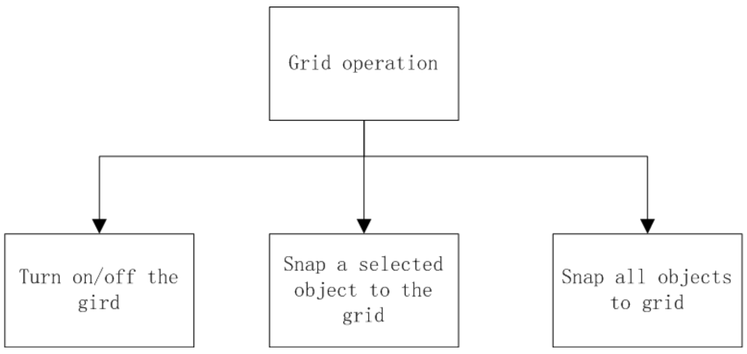


Figure 5.9 Grid operation module

The grid operation module showed in the figure below is used to provide assistance to user. It is a gird ground plane, to be treated as reference object, so that the user can easily get an idea about the position of their objects like the height of a box from the ground floor. The grid can be turn on or off quickly. Snap the selected object function can automatically put the selected object on the ground floor – the grid plane. The user can also perform the action to quickly snap all the objects created in the 3D scene to the grid plane.

Projection module provides ability to export 2D vector image. It is only called by the system once the user has finished their work. In this application, PDF is the 2D vector image file format, the projection module translate the 3D objects' data according to the camera setting into 2D image's data, and the PDF file is then generated according to the translated data.

The program framework and modules are illustrated by following the work flow in the previous paragraphs. Using Java3D as the graphics API to develop this application, the programming work is actually dealing with data and classes. The figure below is the scene graph organized in structure of using Java3D.

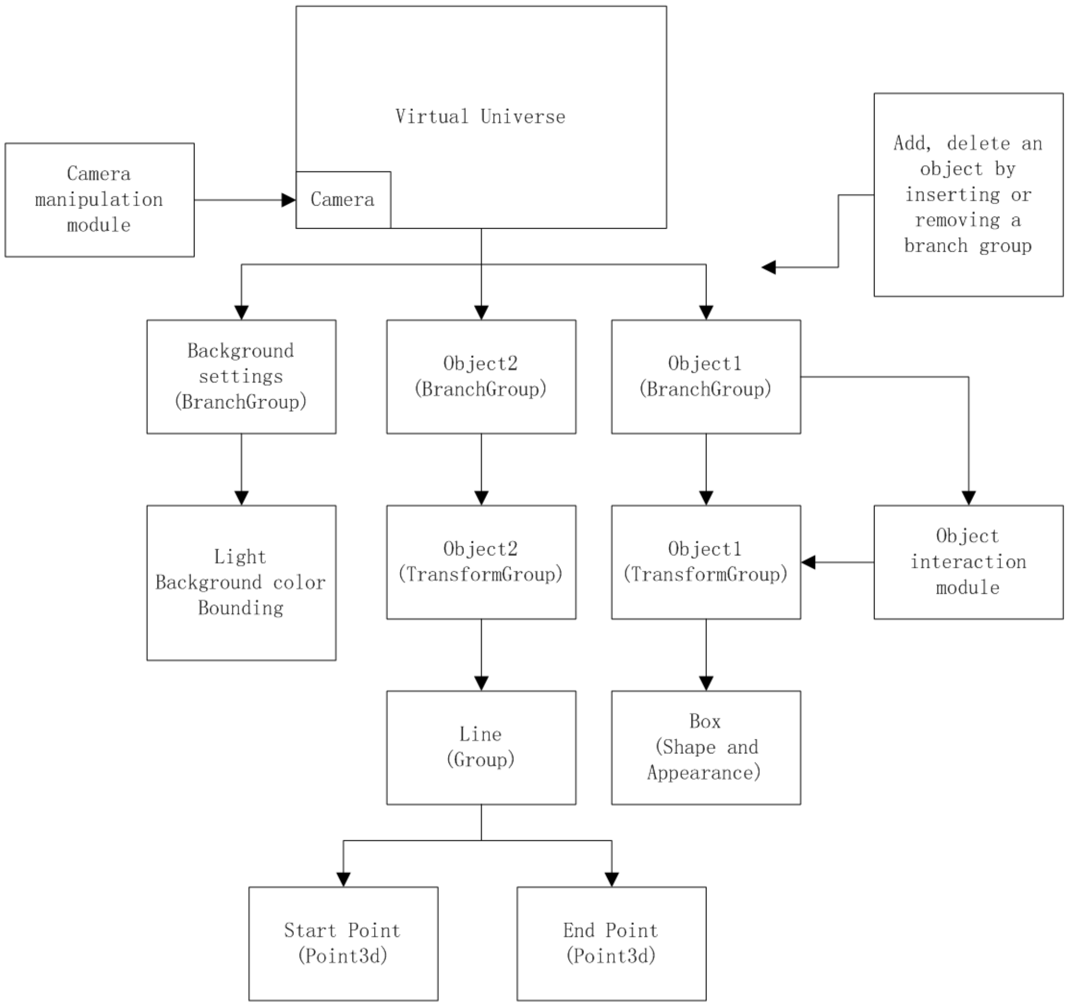

Virtual Universe
Camera manipulation module
Camera
Add, delete an object by inserting or removing a branch group
Background settings (BranchGroup)
Object2 (BranchGroup)
Object1 (BranchGroup)
Light Background color Bounding
Object2 (TransformGroup)
Object1 (TransformGroup)
Object interaction module
Line (Group)
Box (Shape and Appearance)
Start Point (Point3d)
End Point (Point3d)

Figure 5.10 Java3D scene graph of this project

As showed in the figure, the camera view is set up in the virtual universe class, and there are always interaction modules applying on it. The object which cannot interact with the user but still necessary for the 3D scene like bounding, background color, and lights are added to an individual branch group.

Each time the user would like to add an object to the scene, the system inserts a branch group to the 3D scene. The object's branch group always has a transform group child to store the transformation data of the object in this branch. The child of the transform group used to specify the shape of object such as a box, a cylinder, or a sphere, and the appearance of the object such display the object in the way of wireframe or shaded.

The 3D scene knows how many branch group it has, and by returning the index to refer to the selected branch group. Delete a selected object just means to detach the selected object branch group, while object interaction is always added to the selected branch group and apply to the transform group of this branch group. Export the 3D scene as a 2D vector image will gather all the useful data in this scene graph and calculate the data by using a projection algorithm to draw 2D vector image.

5.3 Graphic User Interface and Interaction Design

This section will describe the user interface design from the user's perspective of view. As an interactive 3D graphics application, the most important feature to the user should be real–time 3D scene display. JPanel object from Swing API can include the Canvas3D object which used to show the 3D scene in real-time, so they combined to present the visual output.

The user needs a convenient and direct way to create basic geometries. From other 3D graphics applications example, a group of button has been design to fulfill this task. But before the user clicks the button to create the primitive, the name of the object must be entered into the text field, which is necessary for the organization and management of all the created objects in the scene. With a name entered, and click the creating button, a basic primitive can be created by the default size at the origin of the world coordinate. A grid button should be presented to turn on or off the grid ground plane which is used as a reference object to help the user put the created object.

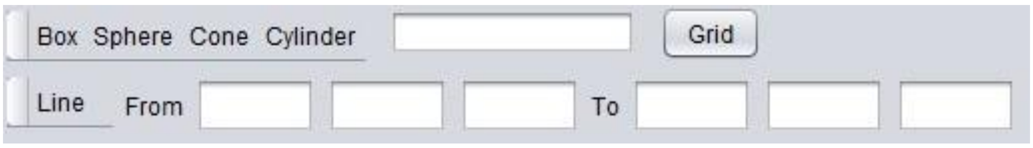


Figure 5.11 Modeling button group

Create a line is different from create other primitives like box, cone, or cylinder. Lines are normally used to connect objects, the start point and end point is more important than the shape and position of the line in most of the cases. So it is more convenient to let the user input the position of the start point and the end point to specify and create a line.

Once the user has created many objects in the scene, to select any one of the created objects is inevitable. This task must be presented a friendly and efficient way to fulfill. According to the experience of using other sophisticated 3D graphics software like Autodesk 3ds Max and Maya, it is a good idea to form an object list which needs to be displayed on the user interface. And this is why the object must be created with a name. The user can select the object by picking any name in the object list, and the interaction can be applied to the selected object. A delete button to be presented not far away from the object list can help the user quickly delete the selected object.

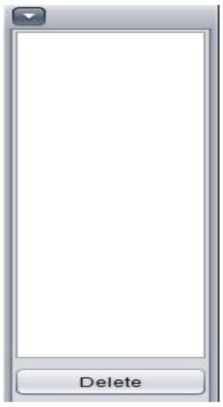


Figure 5.12 (Empty) Object list

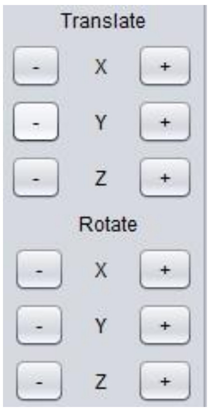


Figure 5.13 Object interaction button group

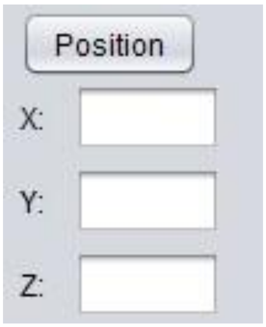


Figure 5.14 Position setting button group

The user can quickly switch the selected object, and interact with this object. The interaction is translation, rotation, and scale, which can be done by using the buttons provided on the user interface and the mouse behavior. Although mouse behavior is quicker than the way of using buttons, the buttons can help beginner to get familiar with this drawing application. And the user sometimes want the precisely put an object to a certain position, so as showed in the picture, the coordinate should be allowed to input. The user can input the x, y, z position and then click the ‘position’ button to set the object to the certain place. And if the text field is empty, clicking the position button can let the text field show the current position of the selected object.

The aim of the project is to allow the user generate 2D vector image from any perspective of view, so the user also need an arbitrary of perspective view when they are working in the 3D scene. Camera manipulation becomes one of the most important tasks. As mentioned before, both camera view manipulation based on world coordinate and the camera coordinate is useful, and result in this application is that the user can use the keyboard navigator buttons to manipulate the camera view in the camera’s coordinate, and can also use the button on the user interface to manipulate the camera in the world coordinate.

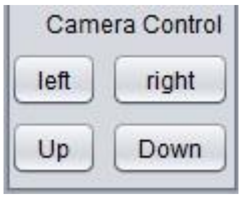


Figure 5.15 Camera manipulation button group

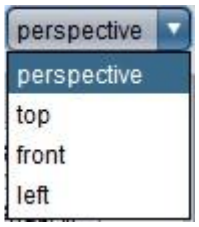


Figure 5.16 Camera view switcher

Some certain camera view position are frequently used, they are perspective view, top view, front view, and side view, so a combo box to quickly switch among different camera views will save the user plenty of time.

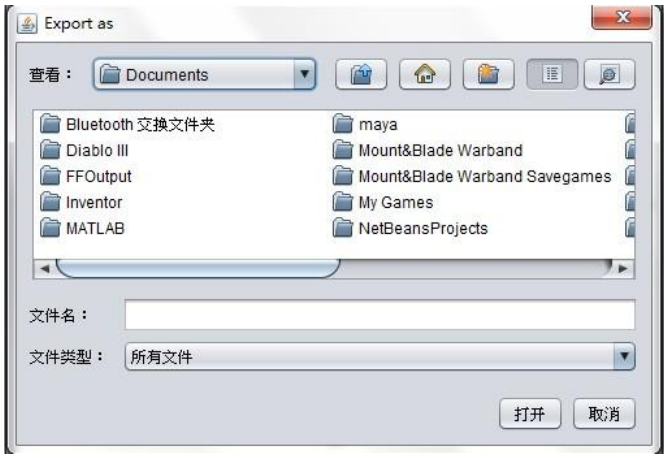


Figure 5.17 File chooser component

Certainly there must be a button to generate and export the 2D vector image. For the user to conveniently choose a directory to put the output file, the file chooser component of the Swing API will appear once the user has clicked the 'done' button. All these interactions have covered the whole pipeline of using this 3D projection drawing application. All the components used above forms the user interface, they are listed below:

JFrame	JPanel
JInternalFrame	JToolBar
Jbutton	JLabel
JTextField	JList
JScrolPane	JComboBox
JFileChooser	

It is one choice to type in the code and manually generate the GUI, this takes more time but allows more flexibility. Another choice to implement the user interface is using the development tools provided by NetBeans IDE. The NetBeans IDE is a free, open-source, cross-platform integrated development environment with built-in support for the Java programming language []. As showed in the picture, in the design view of the NetBeans IDE, the user interface components of Java have been provided as visual tools, developer can simple drag the needed component form the palette into the workspace and then adjust the properties of the component, the code will be automatically generated.

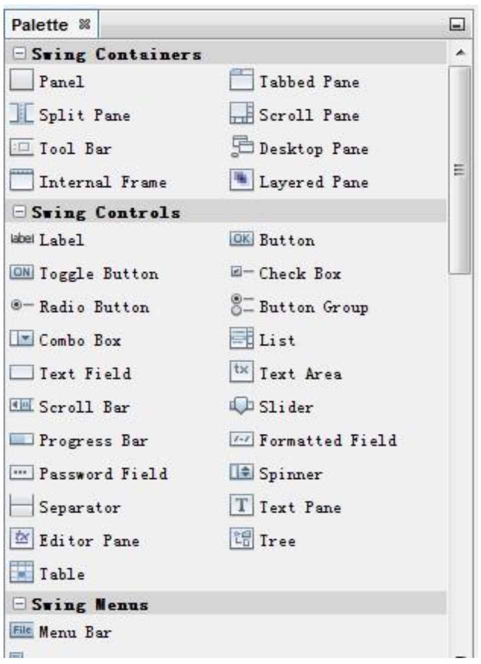
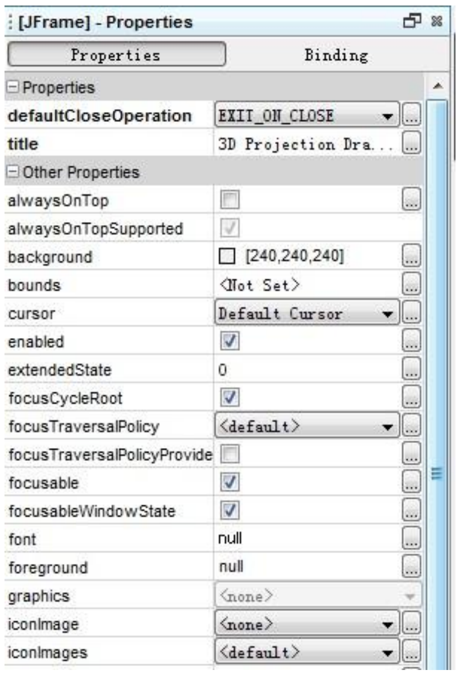


Figure 5.18 Palette



Properties	
defaultCloseOperation	EXIT_ON_CLOSE
title	3D Projection Dra...
Other Properties	
alwaysOnTop	☐
alwaysOnTopSupported	✓
background	☐ [240,240,240]
bounds	<Not Set>
cursor	Default Cursor
enabled	✓
extendedState	0
focusCycleRoot	✓
focusTraversalPolicy	<default>
focusTraversalPolicyProvide	☐
focusable	✓
focusableWindowState	✓
font	null
foreground	null
graphics	<none>
iconImage	<none>
iconImages	<default>

Figure 5.19 Properties

For example, the button used to create the box primitive has been drag from the palette into the user interface, the NetBeans IDE will automatically generate the following piece of Java code:

```
private javax.swing.JButton Box;
Box = new javax.swing.JButton();
Box.setText("Box");
Box.setFocusable(false);
Box.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
Box.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
Box.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        BoxActionPerformed(evt);
    }
});
Create.add(Box);
```

The event listeners have also been added to the components while the function of responding to the event is empty, which need the developer to write their own code to implement the interaction with the GUI component. The whole user interface with event listener can be implemented in this efficient way.

5.4 Module design

This section will describe the modules mentioned in system overall design in details. Code, pseudo-code, and recipe explanation will be the main method to illustrate how these modules are implemented by using Java3D.

5.4.1 Modeling modules

- #### Virtual universe initialization

  Normally, in computer 3D graphics, the camera can be defined as an eye position, a look at position, the distance between the image plate and eye position, and an up vector. In Java 3D, the camera is set up when the user have initialized the virtual universe. The picture below shows the relationship of eye position, image plate, and the virtual world.

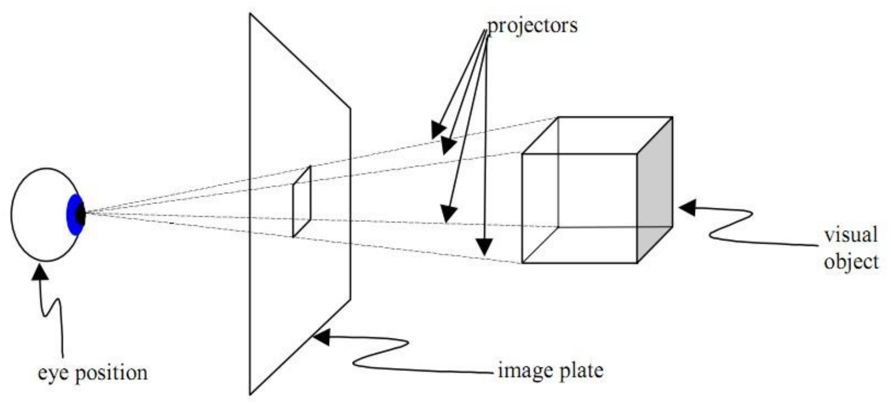


Figure 5.20 Image plate and eye position

When writing code to implement this module, there is a simpler way to initialize the virtual universe, Java3D provides simple universe object, which has been showed in the picture below. Instead of creating instance of each object in the virtual universe in the view branch, creating an instance of simple universe can quickly finish building of the view branch by just setting some parameters.

By default, the camera set up by simple universe has the following features: the image plate is centered at the origin of the world coordinate, the eye position has been put at the point (0, 0, 2.41), the look at point is the origin, the up vector is (0, 1, 0).

The simple universe object has the function of ‘getViewingPlatform()’ which retrieve the object used to manipulate the camera. A transform matrix can be applied to the ‘ViewingPlatform’ object which refers to the camera view, this work can be achieved by using the function of ‘getViewingPlatformTransform().setTransform()’.

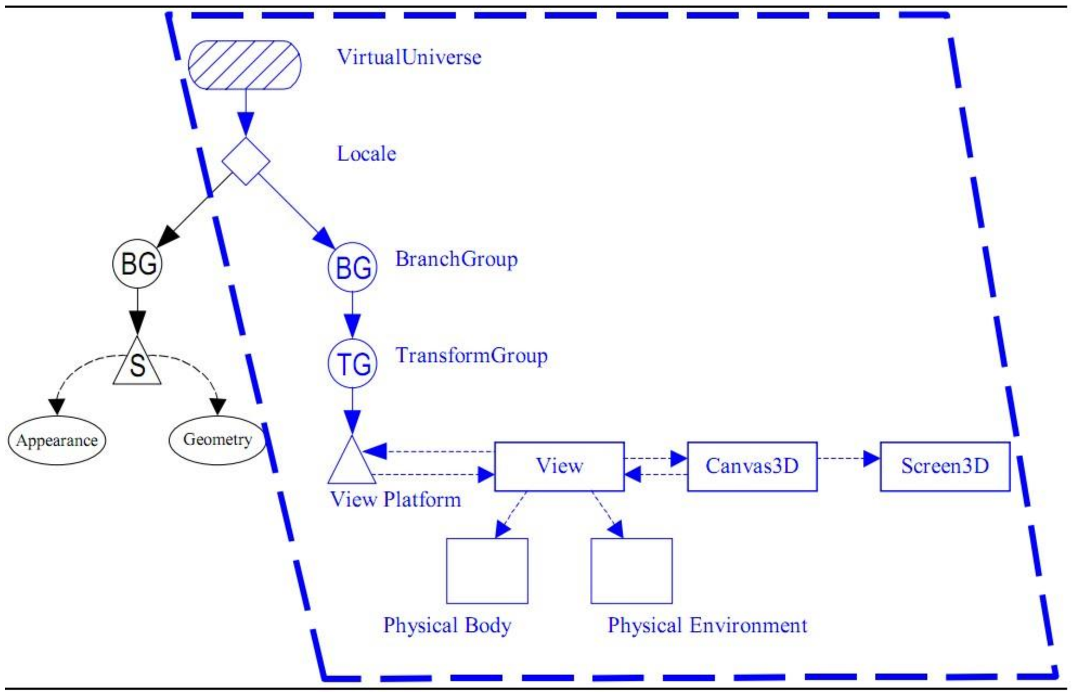


Figure 5.21 Simple Universe

1. Create a Canvas3D Object which is the area to show the 3D scene.
2. Create a Simple Universe object which references the earlier Canvas3D object, and customize the Simple Universe object.
3. Construct a content branch which used to create objects in initialization.
4. Construct another content branch which will allow user interaction to create 3D geometries.
5. Add the above content branches to the children of simple universe.

The recipe above illustrates the specific implementation of this module, and it should be put in the constructor which in charge of initialization of the whole system.

Primitives modeling

In Java3D, models can be presented by calling the function of creating points, lines and faces, then with a proper data structure to organize them. The primitives provided by this application are mainly boxes, cylinders, spheres, and cones, they are regular basic geometries, and Java3D has included functions which can directly generate them.

This figure shows the hierarchy of primitives in Java3D API, but they only provide the shape information of the object, the information of the object's appearance can be implemented by creating an instance of appearance class with proper parameters. The important functions to create these geometries are:

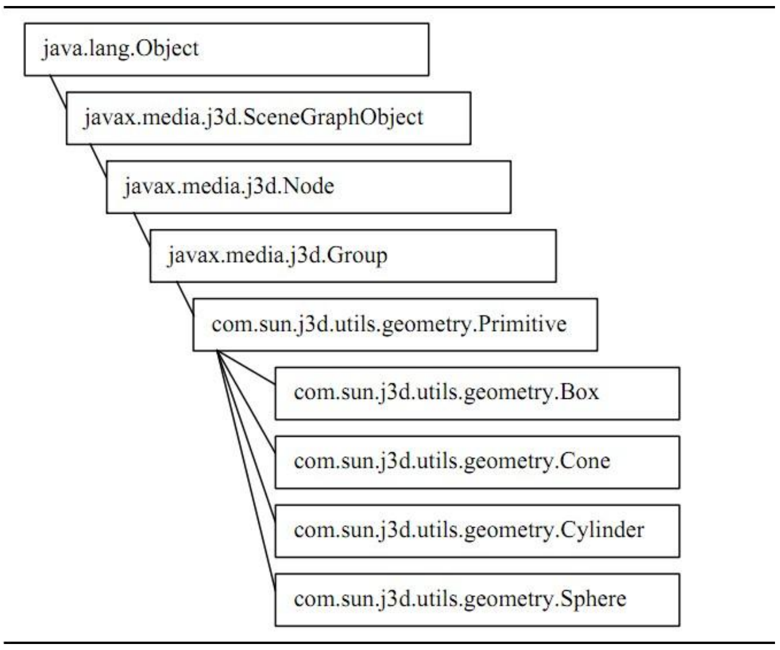

java.lang.Object
javax.media.j3d.SceneGraphObject
javax.media.j3d.Node
javax.media.j3d.Group
com.sun.j3d.utils.geometry.Primitive
com.sun.j3d.utils.geometry.Box
com.sun.j3d.utils.geometry.Cone
com.sun.j3d.utils.geometry.Cylinder
com.sun.j3d.utils.geometry.Sphere

Figure 5.22 Hierarchy of primitives in Java3D API

```
Appearance app = new Appearance();
Color3f ambientColour = new Color3f(0.1f, 0.1f, 0.1f);
Color3f emissiveColour = new Color3f(0.1f, 0.1f, 0.1f);
Color3f specularColour = new Color3f(0.2f, 0.2f, 0.2f);
Color3f diffuseColour = new Color3f(0.5f, 0.5f, 0.5f);
float shininess = 20.0f;
app.setMaterial(new Material(ambientColour, emissiveColour,
    diffuseColour, specularColour, shininess));

Box box = new Box(1.0f, 1.0f, 1.0f, app);
Sphere sphere = new Sphere(1.0f, app);
Cone cone = new Cone(1.0f, 2.0f, app);
Cylinder cylinder = new Cylinder(1.0f, 2.0f, app);
```

The primitives are created at the origin of the world coordinate, while the user need to user the translation function to move them to their ideal place. The constructor refers the created Appearance Class, and set the size of each geometry.

The user can also create lines with specifying the start point and the end point in this application. LineArray Class is used in the modeling work of lines, it is specified by an array points and packaging the points and appearance in the Shape Class. The core code for create a line is:

```
Point3f[] plaPts = new Point3f[2];
plaPts[0] = new Point3f(startx, starty, startz);
plaPts[1] = new Point3f(endx, endy, endz);
LineArray pla = new LineArray(2, LineArray.COORDINATES);
pla.setCoordinates(0, plaPts);
Shape3D plShape = new Shape3D(pla, app);
```

No matter the position of start point and end point, the line array is treated as a single object and its position is set by the function 'setCoordinates'. The grid plane is model by using 40 lines perpendicular with each other.

■ Light, Background Color and Bounding

As mentioned earlier, the bounding is used to accelerate the application's performance and the background color should be set to white to improve user experience. This work can be done by invoking the functions below:

```
bounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0);
Color3f    bgColor = new Color3f(1.0f, 1.0f, 1.0f);
Background bg = new Background(bgColor);
bg.setApplicationBounds(bounds);
```

An ambient light and two directional lights has been added to the background branch to ensure the user can clearly see the objects they created. One of the directional lights is treated as the primary light while the other one is the secondary light with a different direction and lower intensity. And the important functions to invoke are:

```
AmbientLight ambientLight = new AmbientLight(ambientLightColour);
ambientLight.setInfluencingBounds(bounds);
DirectionalLight directionLight = new DirectionalLight(directionLightColour,
directionLightDirection);
directionLight.setInfluencingBounds(bounds);
```

Recipe for building the background content branch which used to create objects in initialization is listed below:

1. Create a background color object with proper parameters and refer the

bounding sphere to it.

2. Create an ambient light object with not very bright white color and refer the bounding sphere to it.
3. Create the primary directional light object with proper direction vector and whit color, and refer the bounding sphere to it.
4. Create the secondary directional light with a different direction vector and a dim color, refer the bounding sphere to it.
5. Add the background object and light objects as the children of the background branch.

5.4.2 Interaction modules

■ Add, Delete and Select Object

The operation of adding and deleting an object can be treated as inserting an object branch or removing an object branch from the scene graph. The following picture shows the different ways of organization of an object branch, at least three ways are presented in the picture. While in this program, all the object branches are organized in the way the middle branch group of the picture shows:

Branch Group – Transform Group – Shape Node – Geometry and Appearance

As the mentioned in the interaction design section, Adding, deleting and selecting operation all has relationship with the object list. So the program of this module must make Swing's JList and Java3D cooperate with each other.

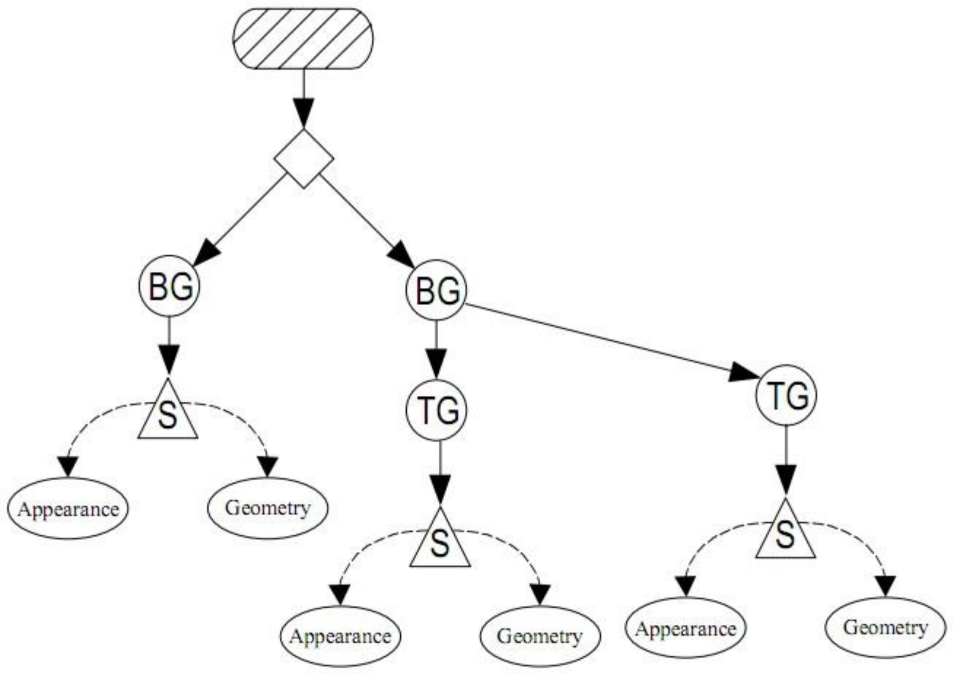


Figure 5.23 Ways of organization of Object Branches

JList is created by dragging the icon in NetBeans IDE into the workspace to implement the object list, and the module of creating any geometry or lines has been discussed in the previous section. This module should fulfill the logic of user interaction to join these two modules together. In order to let the users input the object's name, a TextField object should also be created. The adding interaction should be packaged in the function respond to the creating button pressed, and the procedure is:

1. If the input in the TextField object is not empty or not the same as one of the names already in the JList, go to 3; else, go to 2.
2. Do nothing and return.
3. Insert the input name in the TextField to JList and make the selected index of the JList to current created element.
4. Create a Branch Group and add it as the child of the Simple Universe.
5. Create a Transform Group and add it as child of the Branch Group.
6. Create Shape and Appearance according to the geometry button the user pressed and add it as the child of Transform Group.
7. Interaction with the object switch to this newly created branch.
8. Set selected object as previous selected.

The 7th step will be discussed in details later, while the process above can successfully create a new object and display it in the 3D scene. In a familiar way, the deleting procedure is:

1. If JList length equals to 0, do nothing and return; else, go to 2.
2. Get the current selected index of the JList and remove this element from the JList, also detach the Branch Group identified by this index.
3. Get the current length of the JList, if the selected index now equals to the length which means the selected index exceeds the new length after removed one element, set selected index equals to selected index minus 1; else, do nothing.
4. Interaction with the object switch to currently selected object branch.
5. Set selected object as previous selected.

The deleting process should be packaged in the function respond to the event of clicking 'delete' button. Select process should be implemented in the function respond to the change of JList's index value:

1. Get the selected index and interaction switch to currently selected object branch.
2. Set selected object as previous selected.

Interaction Switching

Only one object is allowed to interact with the user at a time, each time the user change a selected object means the program needs to apply the interaction module to this object and delete the interaction with the object the user select last time. A previous selected index value is needed for this task and by default it equals to -1 which means no previous selected object. All the interactions switching step mentioned above is the process below:

1. If previous selected index equals to -1, do nothing; else, go to 2.
2. Detach the interaction branch.
3. Create a new interaction branch and add it as child a current selected object branch.

Camera View Manipulation

The program provides two different kinds of camera view manipulations as mention in previous section. One is the translation, rotation and zooming operations based on the world coordinate and the other is based on the camera's coordinate. The first kind of camera manipulation is operated by using key board navigation buttons, it is implemented by invoking the Java3D functions.

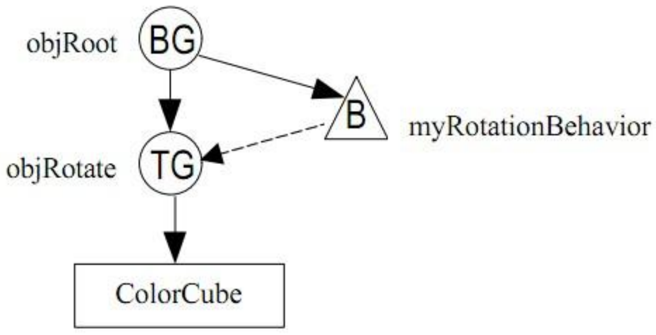


Figure 5.24 Behavior node in the scene graph

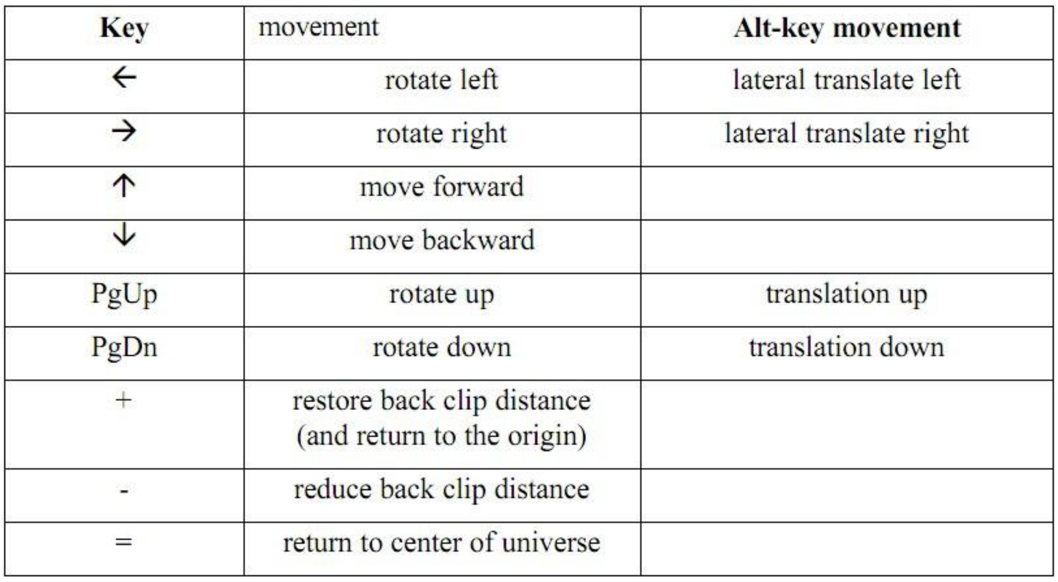
The behavior classes in Java3D like the KeyNavigatorBehavior can be used directly or redefined. Since the KeyNavigatorBehavior has completely fulfill all the requirements for manipulating the camera view based on the camera's coordinate, the program just use it directly. The figure above shows how a behavior node is organized in the scene graph, the behavior node normally should be added as child of a Branch Group, and set the applied object to the Transform Group the behavior would like to control. The core code for implement the KeyNavigatorBehavior is showed below:

```
TransformGroup viewTransformGroup =
                u.getViewingPlatform().getViewPlatformTransform();

  KeyNavigatorBehavior keyInteractor =
                new KeyNavigatorBehavior(viewTransformGroup);

  keyInteractor.setSchedulingBounds(bounds);
  bgRoot.addChild(keyInteractor);
```

In the code, the viewTransformGroup is the transform group of the ViewPlatform object – camera view, an instance of KeyNavigatorBehavior created and applied to the camera's transformation. By invoking the behavior Java3D presents, the camera view can be controlled in the way the following figure shows:



Key	movement	**Alt-key movement**
←	rotate left	lateral translate left
→	rotate right	lateral translate right
↑	move forward	
↓	move backward	
PgUp	rotate up	translation up
PgDn	rotate down	translation down
+	restore back clip distance (and return to the origin)	
-	reduce back clip distance	
=	return to center of universe	

Figure 5.25 Camera manipulation by using keyboard

The other kind of manipulation provided by the program is based on the world coordinate, the camera will always look at the origin. Using four buttons to move the camera up, down, left or right, the manipulation can be implemented by resetting the matrix of the transform group which controls the camera view. In the 3D transformation matrix M, the element $M_{03,}$ M_{13}, and M_{23} is the translation vector (x, y, z). The matrix provided by Java3D can be presented in a 'LookAt' way, which uses an eye point, a look at point, and an up vector to form the matrix. For example, in the function responds to move the camera up but still looks at the origin of the world coordinate:

1. Retrieve the matrix M recorded the data of camera transformation.
2. Create a new matrix by the way 'LookAt'
   a) The eye point ($M_{03,}$ M_{13} + moving interval, M_{23});

    b) Look at point (0, 0, 0);
    c) Up vector (0, 1, 0);
3. Applied this new matrix to the camera view.

The camera can also be quickly switched among perspective view, top view, front view, and side view by using a JComboBox. The JComboBox can have several options, the event listener can detect the user input of changing the selected option. In the function responds to the change of selected value, which means the user may change the selection from top view to front view, the transformation group of the camera view should be reset by new a matrix formed as the 'LookAt' way as follows:

1. Perspective view, eye point (17, 17, 17);
2. Top view, eye point(0, 20, 0);
3. Front view, eye point(0, 0, 20);
4. Left view, eye point (-20, 0, 0).

All different views have the same look at point (0, 0, 0), and up vector (0, 1, 0).

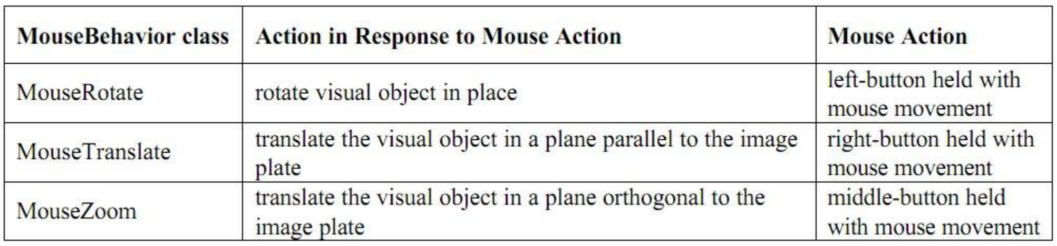
■ Object Interaction

As mentioned in previous section, object can be manipulated in several ways after it has been selected. Use mouse behavior to control the translation, rotation and scale of the object is implemented in the similar way as implementing the keyboard manipulation of the camera view. Java3D provides MouseBehavior class as showed in the figure, which allow the user to control the object with the buttons of his mouse. The important code for implementing a mouse behavior is:

```
MouseRotate uRotate = new MouseRotate();
uRotate.setTransformGroup(RotateGroup);
behaviourBranch.addChild(uRotate);
uRotate.setSchedulingBounds(bounds);

MouseZoom uZoom = new MouseZoom();
uZoom.setTransformGroup(RotateGroup);
behaviourBranch.addChild(uZoom);
uZoom.setSchedulingBounds(bounds);

MouseTranslate uTranslate = new MouseTranslate();
uTranslate.setTransformGroup(RotateGroup);
behaviourBranch.addChild(uTranslate);
uTranslate.setSchedulingBounds(bounds);
```



MouseBehavior class	Action in Response to Mouse Action	Mouse Action
MouseRotate	rotate visual object in place	left-button held with mouse movement
MouseTranslate	translate the visual object in a plane parallel to the image plate	right-button held with mouse movement
MouseZoom	translate the visual object in a plane orthogonal to the image plate	middle-button held with mouse movement

Figure 5.26 Mouse behavior to interact with the selected object

Another way to interact with the object is using the button on the user interface, this is implemented in the similar way as implementing the button control of the camera view. The process of implementing this module:

1. Retrieve the transformation matrix M of the selected object.
2. Create a new matrix T
   a) If it is translation command, use function T.setTranslation(x, y, z) to set the data of matrix T; if it is rotation command, use function T.rotX(angle), T.rotY(angle), or T.rotZ(angle) to set the data; if it is scale command, use T.setScale(scalar) to set the data.
3. Multiply matrix T and matrix M in the order T*M by using function matrix C.mul(T, M);
4. Apply this new matrix C to the selected object.

The program can also get the position of a selected object or set the selected object to a certain position. This can be done in the function responds to the ‘position’ button is clicked:

1. Retrieve the selected object’s transformation matrix M.
2. If the text field is not empty, replace the data of $M_{03,}$ M_{13}, and M_{23} by the data input in the text field; else, display the data of $M_{03,}$ M_{13}, and M_{23} in the text field.

The object can be snap to grid, and this can also be implemented in the button action performed function:

1. retrieve the selected object’s transformation matrix M
2. M_{13} is the value on y-axis in the world coordinate.
3. For different objects, calculate the radius R after they can been scaled.
4. Set M_{13} to the value R.

5.4.3 Projection modules

IText is an open source library for creating and manipulating PDF files in Java. The images in PDF file format are vector images which meet the requirements of this project. In order to draw 2D perspective geometries, this project requires only some of the functions provided by IText:

1. Draw 2D points
2. Draw 2D lines
3. Draw 2D circles
4. Draw 2D ellipses

Once the user has finished the work in the 3D scene, a perspective view will be chosen as the output view port as the 2D vector image, and the button ‘Done’ should be performed to call the program to launch the projection module.

The projection module will go through all the object branches in the scene graph, one by one, translate the 3D data into 2D and draw it on a PDF file:

```
1. Retrieve the transformation matrix $M_c$ of the camera view
2. From the first object branch to the last object branch
3. Retrieve the class name of the shape node in the current branch
4. If class name equals to ‘box’
   a) Project thevbox into 2D and call IText functions to draw it
5. If class name equals to ‘sphere’
   a) Project the sphere into 2D and call IText functions to draw it
6. If class name equals to ‘cylinder’
   a) Project the cylinder into 2D and call IText functions to draw it
7. If class name equals to ‘cone’
   a) Project the cone into 2D and call IText functions to draw it.
8. If class name equals to ‘line’
   a) Project the line into 2D and call IText functions to draw it
```

From the process above, it is obvious that each type of primitives provided by the application requires an individual projection and drawing function based on the primitive’s features. The object P is created by default size at the origin of the world coordinate. After the object has been manipulated by the user, the transformation matrix M can be retrieved. The object’s new status P_w in the world coordinate is:

$$P_w = M * P;$$

According to the perspective projection module described in the background theory section, the data in the world coordinate of any object need to be projected should be firstly transformed into the camera coordinate.

$$P_c = M_c * P_w;$$

For the point p or radius r of any primitive needs to be projected, it has:

$$p_x/p_z = p'_x/d;$$
$$p_y/p_z = p'_y/d;$$
$$r/\mathrm{sqrt}(p_x^2, p_y^2, p_z^2) = r'/\mathrm{sqrt}(p'^2_x, p'^2_y, p'^2_z);$$

The page size of the PDF file can be width and height, in which the origin is located at the left-bottom corner, while the projected 2D object, they are based on the origin at the center of the page. To solve this mismatch problem, the projected point p' (p'_x, p'_y) should be adjusted as p' (p'_x + width/2, p'_y + height/2).

■ Sphere

The projection of the sphere in 2D is always a circle, so the only center point and the radius of the sphere need to be projected according to the mathematical method above. Having the 2D center point and radius, a circle can be easily drawn by invoking the functions IText.

■ Line and Box

The projection of the line and box are also simple. For the line, the start point and the end point need to be projected. By having the 2D start point and end point of the line, the work can be successfully finished by IText's drawing line function. The 8 vertexes of the box need to be projected, and if point 1, 2, 3, 4 forms the bottom face while point 5, 6, 7, 8 forms the top face, the projected 2D point should be connected by lines as the following rules:

1 – 2 – 3 – 4, 5 – 6 – 7 – 8, 1 – 5, 2 – 6, 3 – 7, 4 – 8.

■ Cylinder and Cone

It becomes complex to project cylinder and cone into 2D because the bottom disc will be projected as an ellipse. The first step to project a cylinder into 2D is project the center point of the top face and the bottom face (c1 and c2), then project the radius of the two faces into 2D according to the already projected center points (r1 and r2). By default, the height (H) of the cylinder and cone in 3D is 2, while the projected height (h) can be calculated by the two projected center points. The actual height (d) which is the distance between c1 and c2, can be even shorter than projected height h because of the rotation influence.

The long radius (a1 and a2) of the ellipse will always be equals to the length of the projected radius (r1 and r2), while the short radius (b1 and b2) has the mathematical relationship with the projected height (h) of the cylinder.

$$\cos(\text{angle}) = d/h;$$
$$\text{sine}(\text{angle}) = b/a;$$

By now, the projected center c1and c2, the long radius of the ellipse a1 and a2, and the short radius of the ellipse b1 and b2 are known. Another important parameter to know is the orientation of the ellipse, and it is obvious the orientation of the ellipse (k) can be calculated by using the slope of the line which is the actual height calculated by c1 and c2.

The following work is to call ITex functions to draw two ellipses according to the parameter just calculated. Then if the point 1 and 2 is the start point and end point of the top ellipse's long radius while point 3 and point 4 is the start and end point of the bottom ellipse's long radius. Draw lines to connect point 1 and point 3, point 2 and point 4.

Object cone can be projected in a similar way while it only needs to project one bottom face as an ellipse. After the projected ellipse has been drawn, in 2D coordinate system, two tangents need to be calculated. They are the tangents of the ellipse, but go across the top vertex of the cone. And these two tangent lines should certainly be drawn by using IText.

6. Project Evaluation

The project is mainly on software development, and the final user interface has been showed in the picture. Compare to other software in the market, it still needs many improvements to reach a product level. However, this application can now go through the whole pipeline of generating a 2D vector image and can also provide many useful functions like creating a grid plane as a reference object. As the picture below shows, the user interface is quite clear, the interactions are friendly, and all the functionality provided by the program aims at a single goal, which helps the user with any professional skills to quickly get familiar to use it.

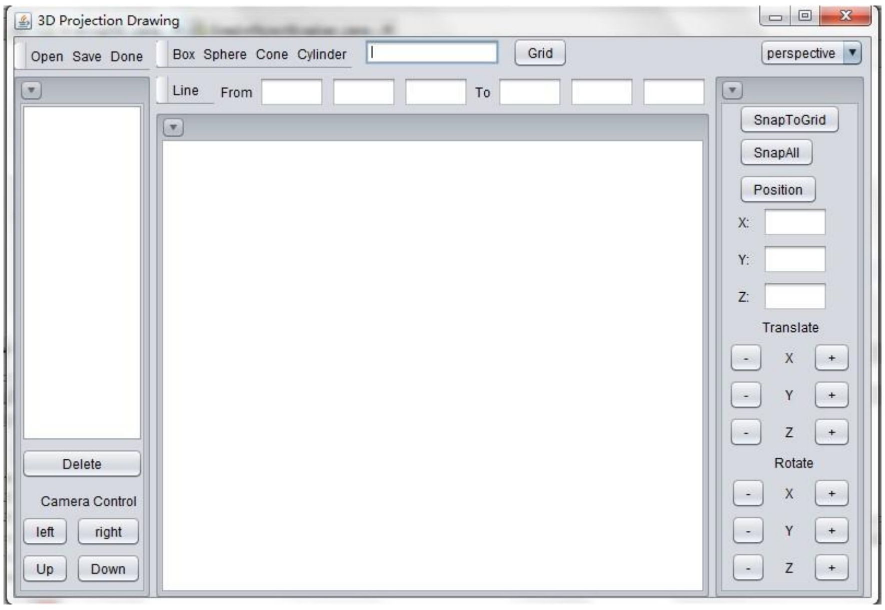


Figure 6.1 The user interface of 3D projection drawing package

The left internal frame of the GUI is the object list, delete button, and camera control button, while the creation button group is located in the top of the user interface. The right side of the user interface is used for the object interaction, like translating or rotating a selected object. The summary of the functionality which the project currently achieved is listed below:

1. Create a box, sphere, cone, or cylinder by entering a name in the text field beside the creation button, then click the creation button.

2. Create a line by entering name, start point, and end point, then click ‘Line’ button.

3. Turn on or turn off the grid by a simple click on the ‘Grid’ button.

4. Quickly switch camera view among top, left, front, and perspective using the

combo box in the top-right corner of the GUI.

5. Change the selection of an existing object by using the object list in the left of the GUI.

6. Delete a selected object by clicking the ‘Delete’ button.

7. Snap a selected object to the grid by clicking the ‘Snap’ button or Snap all objects to grid by clicking the ‘SnapToAll’ button.

8. Manipulate the camera view by clicking the buttons in the left-bottom corner of the GUI or using the keyboard navigation button.

9. Manipulate the selected object by clicking the translate button group, rotate button group, or using the mouse.

10. Get the position of a selected object by clicking the ‘Position’ button when the text field is empty.

11. Set a selected object to a certain position by input data in the text field and click the ‘Position’ button.

12. Generate a PDF image file by clicking the ‘Done’ button to call the file chooser.

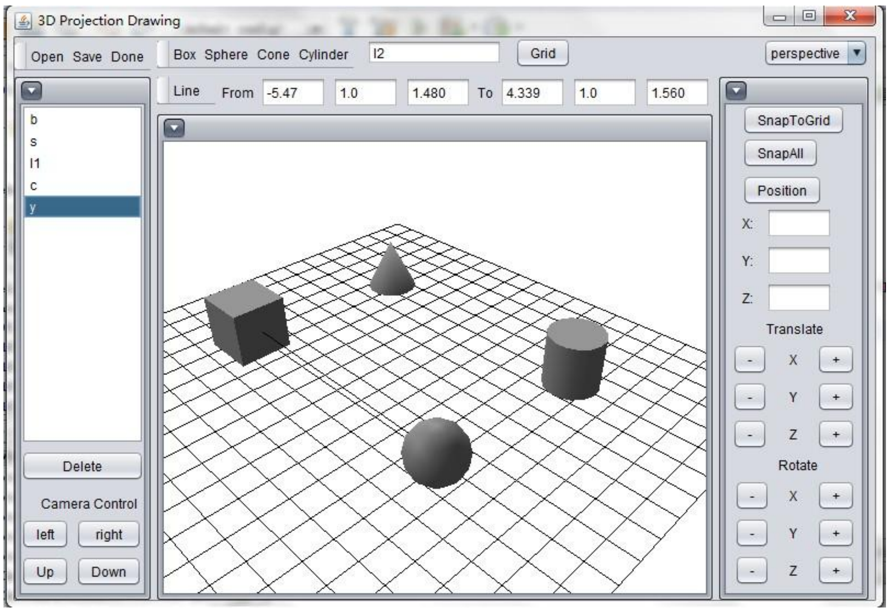


Figure 6.2 The user interface of 3D projection drawing package when the user is working

The above picture shows the GUI when user is using this application, the work space of the GUI demonstrate the 3D scene clearly. From the list of functions, the user will be satisfied by quickly creating 3D primitives in the 3D scene and conveniently interacting with the created objects and the camera view. In terms of result – 2D vector images, the following figures shows the geometries in pairs, the left side pictures are primitives in 3D scene, while the right side pictures are primitives in 2D PDF images.

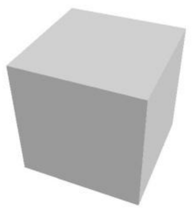


Figure 6.3 Box in 3D scene

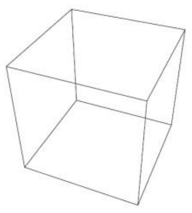


Figure 6.4 Box in vector image

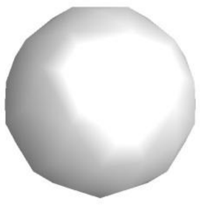


Figure 6.5 Sphere in 3D scene

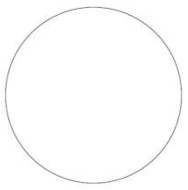


Figure 6.6 Sphere in vector image

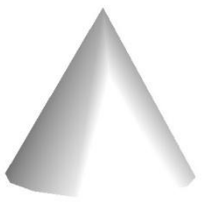


Figure 6.7 Cone in 3D scene

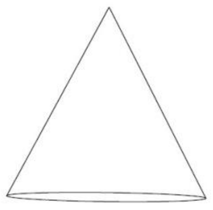


Figure 6.8 Cone in vector image

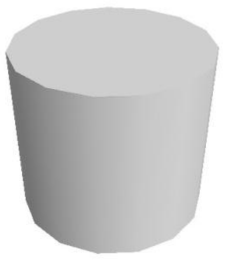


Figure 6.9 Cylinder in 3D scene

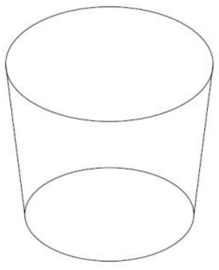
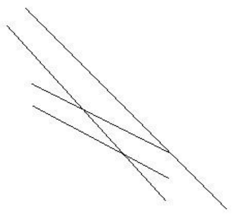


Figure 6.10 Cylinder in vector image



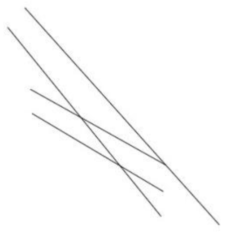


Figure 6.11 Lines in 3D scene

Figure 6.12 Lines in vector image

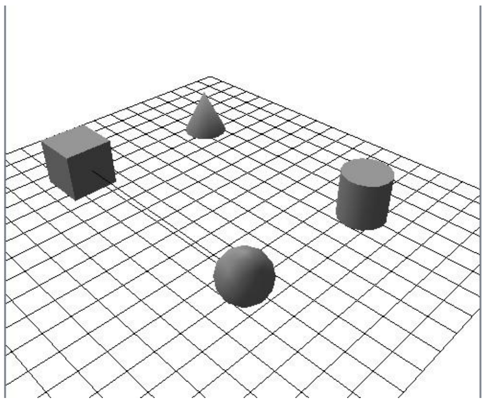


Figure 6.13 Multiple objects in 3D scene

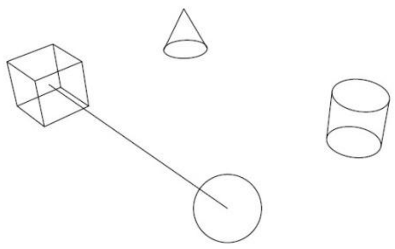


Figure 6.14 Multiple objects in vector image

The comparison between the images in the left and right shows the application can precisely project the object into 2D PDF images as perspective wireframe structure, and the effect is quite acceptable. With such kind of output, user can successfully use this application to create some graphical representations or diagram forms which include 3D geometries.

However, there are also some obvious shortcomings of the application, for example, unable to save and load the unfinished source file, which means the user are forced to finish the work at one time; cannot change a different color the created objects, which makes the user difficult to distinguish different object after many primitives have been created; no quick reversal action in this system which cause the user' impatience because of a mass of effort on redoing or undoing; the object can only be interacted individually while in many cases a group interaction will save time; more primitives should be provided or the more kinds of modification or combination should be allowed since the current use of the project is too narrow.

In conclusion, the current achievements of the project successfully fulfill the requirements in the specification and present an acceptable interactive 3D graphics application. But in terms of time limitation of the project and the time consuming of the software development, the functionalities currently provided may be simple and needs expansion. Although, it still has a long way to go to reach the product level, this simple 3D drawing tool can be an excellent beginning.

7. Further Work

In terms of software development, the application can still be improved in many ways as mentioned in last section. It is common for the most sophisticated software to be along with regular updating, so not only the 7 improvements below can be raised for this project, but they can be reasonable examples. And according to current achievements and evaluation, the following list is a summary of improvement suggestions:

1. The source file of the 3D scene cannot be saved or loaded. This is very useful when the work cannot be done at one time.

2. According to the principles of interaction design, always allow user to do some reversal action. The application now cannot let user quickly undo or redo their action, but it is helpful to make the user easily cancel or recover their wrong actions.

3. The color of the created object should be allowed to be changed. So it is in the projected 2D vector image, the projected object can be demonstrated in different colors.

4. The object in 3D scene cannot only be displayed in shaded mode, but also can be displayed in wireframe structure. And it is better for the user to quickly switch between these 2 modes by using a button.

5. More basic primitives can be provided like pyramid and torus.

6. Not only lines can be drawn, but also arrows and curves.

7. The interaction of line drawing can be improved by simply clicking the mouse in the workspace.

8. Bibliography

[1] P. Shirley et al. Fundamentals of Computer Graphics, 2nd ed. India: A K Peters Wellesley, 2005.

[2] M. Slater, A. Steed, & Y. Chrysanthou. Computer Graphics and Virtual Environments: From Realism to Real-Time. USA: Pearson Education limited, 2002.

[3] J. Foley et al. Computer Graphics Principles and Practice, 2nd ed. USA: Addison-Wesley Publishing Company, 1990.

[4] J. Preece et al. The Systems Programming series: Human-Computer Interaction. UK: Addison-Wesley Publishing Company, 1994.

[5] B. Shneiderman. Designing the User Interface: strategies for Effective Human-Computer Interaction. Canada: Addison-Wesley Publishing Company, 1987.

[6] M. Otaduy, T. Igarashi, & J. LaViola. Lecture Notes, Topic: “Interaction: Interfaces, Algorithms, and Applications”. SIGGRAPH 2009, Course #6, Aug, 2009.

[7] M. Christie & P. Oliver. Course Notes, Topic: “Camera Control in Computer Graphics: Models Techniques and Applications”. SIGGRAPH Asia, 2009.

[8] C. Park, S. Kim, & S. Ha. “A Dataflow Specification for System Level Synthesis of 3D Graphics Applications.” Proceedings of the 2001 Asia and South Pacific Design Automation Conference, 2001, pp. 78.

[9] K. Walczak. “Structure Design of Interactive VR Applications.” Web3D '08 Proceedings of the 13th international symposium on 3D web technology, 2008, pp. 106.

[10] C. Elliott, G. Schechter, R. Yeung, & S. Abi-Ezzi. “TBGA: A High Level Framework for Interactive, Animated 3D Graphics Applications.” Proceedings of the 21st annual conference on Computer graphics and interactive techniques, 1994.

[11] M. Balzer & O. Deussen. “Hierarchy based 3D Visualization of Large Software Structures.” Proceedings of the conference on Visualization '04, 2004, pp. 1.

[12] T. Igarashi & J. Hughes. Course Notes, Topic: “A Suggestive Interface for 3D Drawing.” ACM SIGGRAPH 2006 Courses, 2006, pp. 1.

[13] S. Uno & H. Matsuka. “A General Purpose Graphic System.” Proceedings of the 6th annual conference on Computer graphics and interactive techniques, 1979, pp. 25.

[14] Oracle . “Lesson: Using Swing Components.” Internet: http://docs.oracle.com/javase/tutorial/uiswing/components/index.html, 2012 [21, September, 2012].

[15] John Zukowski: The Definitive Guide to Java Swing, Third Edition, Apress, ISBN 1-59059-447-9

[16] D. Bouvier. “Java 3D API Tutorial.” Internet: http://java.sun.com/developer/onlineTraining/java3d/, September 2000 [21, September, 2012].

9. Appendix: source code

Not all the code are included in the appendix because there will be too many pages, just 3 important parts of the program code is attached here.

- Create background objects:

```
private BranchGroup CreateBgRoot() {
        bgRoot = new BranchGroup();
        Color3f bgColor = new Color3f(1.0f, 1.0f, 1.0f);
        Background bg = new Background(bgColor);
        bg.setApplicationBounds(bounds);
        bgRoot.addChild(bg);
        Color3f ambientLightColour = new Color3f(0.2f, 0.2f, 0.2f);
        AmbientLight ambientLight = new AmbientLight(ambientLightColour);
        ambientLight.setInfluencingBounds(bounds);
        Color3f directionLightColour = new Color3f(0.5f, 0.5f, 0.5f);
        Vector3f directionLightDir = new Vector3f(30.0f, -30.0f, 30.0f);
        DirectionalLight directionLight = new DirectionalLight(directionLightColour,
        directionLightDir);
        directionLight.setInfluencingBounds(bounds);
        Color3f directionLightColour2 = new Color3f(0.5f, 0.5f, 0.5f);
        Vector3f directionLightDir2 = new Vector3f(30.0f, -30.0f, -30.0f);
        DirectionalLight directionLight2 = new DirectionalLight(directionLightColour2,
        directionLightDir2);
        directionLight2.setInfluencingBounds(bounds);
        bgRoot.addChild(ambientLight);
        bgRoot.addChild(directionLight);
        bgRoot.addChild(directionLight2);
        return bgRoot;
}
```

- Create an object like a box, sphere, cone, cylinder, and line. The following code is adding a line and applying mouse interaction behavior to it. Other primitives are coded in a similar way.

```
private void LineActionPerformed(java.awt.event.ActionEvent evt) {
        String name = InputName.getText();
        if (name.equals("") || alreadyInList(name)) {
                Toolkit.getDefaultToolkit().beep();
                InputName.requestFocusInWindow();
                InputName.selectAll();
                return;
        }
        selectedIndex = ObjList.getSelectedIndex(); //get selected index
        if (selectedIndex == -1) { //no selection, so insert at beginning
                selectedIndex = 0;
        } else {                    //add after the selected item
                selectedIndex++;
        }
        ObjListModel.insertElementAt(InputName.getText(), selectedIndex);
        InputName.requestFocusInWindow();
        InputName.setText("");
        ObjList.setSelectedIndex(selectedIndex);
        ObjList.ensureIndexIsVisible(selectedIndex);

        String s_x = SX.getText();
        String s_y = SY.getText();
        String s_z = SZ.getText();
        String e_x = EX.getText();
        String e_y = EY.getText();
        String e_z = EZ.getText();
        float sx = Float.parseFloat(s_x);
        float sy = Float.parseFloat(s_y);
        float sz = Float.parseFloat(s_z);
        float ex = Float.parseFloat(e_x);
        float ey = Float.parseFloat(e_y);
        float ez = Float.parseFloat(e_z);

        BranchGroup extend = new BranchGroup();
        extend.setCapability(BranchGroup.ALLOW_CHILDREN_EXTEND);
        extend.setCapability(BranchGroup.ALLOW_CHILDREN_WRITE);
        extend.setCapability(BranchGroup.ALLOW_DETACH);

        Transform3D Scale = new Transform3D();
        Transform3D Translate = new Transform3D();
        Transform3D Rotate = new Transform3D();
        TransformGroup ScaleGroup = new TransformGroup(Scale);
        TransformGroup TranslateGroup = new TransformGroup(Translate);
        TransformGroup RotateGroup = new TransformGroup(Rotate);
        ScaleGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        TranslateGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        RotateGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        /////////////////////////
        Group lineGroup = new Group();
        Appearance app = new Appearance();
```

```
        ColoringAttributes ca = new ColoringAttributes(black,
        ColoringAttributes.SHADE_FLAT);
        app.setColoringAttributes(ca);

        Point3f[] plaPts = new Point3f[2];
        plaPts[0] = new Point3f(sx, sy, sz);
        plaPts[1] = new Point3f(ex, ey, ez);
        LineArray pla = new LineArray(2, LineArray.COORDINATES);
        pla.setCoordinates(0, plaPts);
        Shape3D plShape = new Shape3D(pla, app);
        lineGroup.addChild(plShape);
        //////////////////////////
        if(prevIndex != -1) {
                BranchGroup lastBG = (BranchGroup) objRoot.getChild(prevIndex);
                BranchGroup behaviourBranch = (BranchGroup) lastBG.getChild(1);
                behaviourBranch.detach();
        }
        BranchGroup behaviourBranch = new BranchGroup();
        behaviourBranch.setCapability(BranchGroup.ALLOW_DETACH);
        MouseRotate uRotate = new MouseRotate();
        uRotate.setTransformGroup(RotateGroup);
        behaviourBranch.addChild(uRotate);
        uRotate.setSchedulingBounds(bounds);
        MouseZoom uZoom = new MouseZoom();
        uZoom.setTransformGroup(RotateGroup);
        behaviourBranch.addChild(uZoom);
        uZoom.setSchedulingBounds(bounds);
        MouseTranslate uTranslate = new MouseTranslate();
        uTranslate.setTransformGroup(RotateGroup);
        behaviourBranch.addChild(uTranslate);
        uTranslate.setSchedulingBounds(bounds);
        Scale scale = new Scale(ScaleGroup);
        behaviourBranch.addChild(scale);
        scale.setSchedulingBounds(bounds);
        ScaleGroup.addChild(lineGroup);
        TranslateGroup.addChild(ScaleGroup);
        RotateGroup.addChild(TranslateGroup);
        extend.addChild(RotateGroup);
        extend.addChild(behaviourBranch);
        prevIndex = ObjList.getSelectedIndex();
        objRoot.addChild(extend);
}
```

- Projection is one of the most important part of the program, because they are coded in a similar way, cylinder projection is given as the example code:

```
private void TAPCylinder(Matrix4d M, Matrix4d Ms, PdfContentByte cb) throws Exception {
    Point3d[] c = new Point3d[3];
    Point3d[] cc = new Point3d[3];
    Point[] cp = new Point[3];
    Point[] pp = new Point[8];
    //////////initialization////////////////////
    c[0] = new Point3d(0.0d, -1.0d, 0.0d);
    c[1] = new Point3d(0.0d, 1.0d, 0.0d);
    c[2] = new Point3d(0.0d, 0.0d, 0.0d);
    ////////////////////////////////////////////
    cc[0] = new Point3d(0.0d, 0.0d, 0.0d);
    cc[1] = new Point3d(0.0d, 0.0d, 0.0d);
    cc[2] = new Point3d(0.0d, 0.0d, 0.0d);
    //////////////////////////////////////////////
    cp[0] = new Point(0.0d, 0.0d);
    cp[1] = new Point(0.0d, 0.0d);
    cp[2] = new Point(0.0d, 0.0d);
    //////////////////////////////////////////////
    pp[0] = new Point(0.0d, 0.0d);
    pp[1] = new Point(0.0d, 0.0d);
    pp[2] = new Point(0.0d, 0.0d);
    pp[3] = new Point(0.0d, 0.0d);
    pp[4] = new Point(0.0d, 0.0d);
    pp[5] = new Point(0.0d, 0.0d);
    pp[6] = new Point(0.0d, 0.0d);
    pp[7] = new Point(0.0d, 0.0d);
    /////////////////////////////////////////////
    double r = 1*Ms.m00;
    double[] rp = new double[3];
    double[] sp = new double[2];
    double angle;
    ////////////////////////////////////////////
    for(int i = 0; i < 3; i++) {
        //calculate the coordinate in camera
        cc[i].x = M.m00*c[i].x + M.m01*c[i].y + M.m02*c[i].z + M.m03;
        cc[i].y = M.m10*c[i].x + M.m11*c[i].y + M.m12*c[i].z + M.m13;
        cc[i].z = M.m20*c[i].x + M.m21*c[i].y + M.m22*c[i].z + M.m23;
        //doing the projection
        cp[i].x = (d*cc[i].x)/cc[i].z;
        cp[i].y = (d*cc[i].y)/cc[i].z;
        rp[i] = r*(Math.sqrt(cp[i].x*cp[i].x + cp[i].y*cp[i].y + d*d))/(Math.sqrt(cc[i].x*cc[i].x +
        cc[i].y*cc[i].y + cc[i].z*cc[i].z));
        //adjust to the pdf page
        cp[i].x = cp[i].x*a + pageWidth/2;
        cp[i].y = cp[i].y*a + pageHeight/2;

        rp[i] = rp[i]*a;
    }
    angle    = Math.acos((cp[0].distance(cp[1]))/(2*rp[2]));
    for(int i = 0; i < 2; i++) {
        sp[i] = rp[i]*Math.sin(angle);
    }
    ///////////////////////////////////////////////////////
    double alpha;
    double a = cp[1].x - cp[0].x;
    double b = cp[1].y - cp[0].y;

    alpha = Math.atan((-1)*a/b);

    pp[0].x = cp[0].x + rp[0]*Math.cos(alpha);
    pp[0].y = cp[0].y + rp[0]*Math.sin(alpha);
    pp[2].x = cp[0].x - rp[0]*Math.cos(alpha);
    pp[2].y = cp[0].y - rp[0]*Math.sin(alpha);

    pp[4].x = cp[1].x + rp[1]*Math.cos(alpha);
    pp[4].y = cp[1].y + rp[1]*Math.sin(alpha);
    pp[6].x = cp[1].x - rp[1]*Math.cos(alpha);
    pp[6].y = cp[1].y - rp[1]*Math.sin(alpha);

    alpha = Math.atan(b/a);

    pp[1].x = cp[0].x + sp[0]*Math.cos(alpha);
    pp[1].y = cp[0].y + sp[0]*Math.sin(alpha);
    pp[3].x = cp[0].x - sp[0]*Math.cos(alpha);
    pp[3].y = cp[0].y - sp[0]*Math.sin(alpha);

    pp[5].x = cp[1].x + sp[1]*Math.cos(alpha);
    pp[5].y = cp[1].y + sp[1]*Math.sin(alpha);
    pp[7].x = cp[1].x - sp[1]*Math.cos(alpha);
    pp[7].y = cp[1].y - sp[1]*Math.sin(alpha);

    drawCylinder2D (pp[0], pp[1], pp[2], pp[3], pp[4], pp[5], pp[6], pp[7], cb);
}
```

```
protected void drawCylinder2D (Point p1, Point p2, Point p3, Point p4,
            Point p5, Point p6, Point p7, Point p8, PdfContentByte cb) throws Exception {
    double k1 = 0;
    double k2 = 0;
    double k3 = 0;
    double k4 = 0;
    double x1 = (p1.x + p3.x)/2;
    double y1 = (p1.y + p3.y)/2;
    double x2 = (p5.x + p7.x)/2;
    double y2 = (p5.y + p7.y)/2;
    double p2x = 0;
    double p2y = 0;
    double p4x = 0;
    double p4y = 0;
    double p6x = 0;
    double p6y = 0;
    double p8x = 0;
    double p8y = 0;
    double angle1;
    double angle2;
    double p_2x;
    double p_2y;
    double p_4x;
    double p_4y;
    double p_6x;
    double p_6y;
    double p_8x;
    double p_8y;

    if(p1.x - p3.x != 0 && p2.x - p4.x != 0) {

        k1 = (p1.y - p3.y)/(p1.x - p3.x);
        k2 = (p2.y - p4.y)/(p2.x - p4.x);

        p2x = (k2*p3.x - k1*p2.x + p2.y - p3.y)/(k2 - k1);
        p2y = k2*(p2x - p3.x) + p3.y;

        p4x = (k2*p1.x - k1*p4.x + p4.y - p1.y)/(k2 - k1);
        p4y = k2*(p4x - p1.x) + p1.y;
    }
    if(p1.x - p3.x == 0 && p2.x - p4.x != 0) {
        p2x = p2.x;
        p2y = p3.y;
        p4x = p4.x;
        p4y = p1.y;
}
if(p1.x - p3.x != 0 && p2.x - p4.x == 0) {
        p2x = p3.x;
        p2y = p2.y;
        p4x = p1.x;
        p4y = p4.y;
}
angle1 = Math.atan(k1);
p_2x = (p2x - x1)*Math.cos(-angle1) - (p2y - y1)*Math.sin(-angle1) + x1;
p_2y = (p2y - y1)*Math.cos(-angle1) + (p2x - x1)*Math.sin(-angle1) + y1;
p_4x = (p4x - x1)*Math.cos(-angle1) - (p4y - y1)*Math.sin(-angle1) + x1;
p_4y = (p4y - y1)*Math.cos(-angle1) + (p4x - x1)*Math.sin(-angle1) + y1;

AffineTransform af1 = new AffineTransform();
cb.saveState();
af1.rotate(angle1, x1, y1);
cb.transform(af1);
cb.ellipse((float)p_2x, (float)p_2y, (float)p_4x, (float)p_4y);
cb.stroke();
cb.restoreState();

if(p5.x - p7.x != 0 && p6.x - p8.x != 0) {

        k3 = (p5.y - p7.y)/(p5.x - p7.x);
        k4 = (p6.y - p8.y)/(p6.x - p8.x);

        p6x = (k4*p7.x - k3*p6.x + p6.y - p7.y)/(k4 - k3);
        p6y = k4*(p6x - p7.x) + p7.y;

        p8x = (k4*p5.x - k3*p8.x + p8.y - p5.y)/(k4 - k3);
        p8y = k4*(p8x - p5.x) + p5.y;
}
if(p5.x - p7.x == 0 && p6.x - p8.x != 0) {
        p6x = p6.x;
        p6y = p7.y;
        p8x = p8.x;
        p8y = p5.y;
}
if(p5.x - p7.x != 0 && p6.x - p8.x == 0) {
        p6x = p7.x;
        p6y = p6.y;
        p8x = p5.x;
        p8y = p8.y;
        }
        angle2 = Math.atan(k3);
        p_6x = (p6x - x2)*Math.cos(-angle2) - (p6y - y2)*Math.sin(-angle2) + x2;
        p_6y = (p6y - y2)*Math.cos(-angle2) + (p6x - x2)*Math.sin(-angle2) + y2;
        p_8x = (p8x - x2)*Math.cos(-angle2) - (p8y - y2)*Math.sin(-angle2) + x2;
        p_8y = (p8y - y2)*Math.cos(-angle2) + (p8x - x2)*Math.sin(-angle2) + y2;

        AffineTransform af2 = new AffineTransform();
        cb.saveState();
        af2.rotate(angle2, x2, y2);
        cb.transform(af2);
        cb.ellipse((float)p_6x, (float)p_6y, (float)p_8x, (float)p_8y);
        cb.stroke();
        cb.restoreState();
        cb.saveState();
        cb.moveTo((float)p1.x, (float)p1.y);
        cb.lineTo((float)p5.x, (float)p5.y);
        cb.moveTo((float)p3.x, (float)p3.y);
        cb.lineTo((float)p7.x, (float)p7.y);
        cb.stroke();
        cb.restoreState();
}
```

- Code for quickly switching among top, front, left, and perspective views:

```
private void ViewItemStateChanged(java.awt.event.ItemEvent evt) {
        if (View.getSelectedIndex() == 0) {
                Transform3D lookAt = new Transform3D();
                lookAt.lookAt(new Point3d(17, 17, 17), new Point3d(0.0, 0.0, 0.0), new Vector3d(0.0,
                1.0, 0.0));
                lookAt.invert();
                u.getViewingPlatform().getViewPlatformTransform().setTransform(lookAt);
        }
        if (View.getSelectedIndex() == 1) {
                Transform3D lookAt = new Transform3D();
                lookAt.lookAt(new Point3d(0.0, 20, 0.0), new Point3d(0.0, 0.0, 0.0), new Vector3d(0.0,
                0.0, -1.0));
                lookAt.invert();
                u.getViewingPlatform().getViewPlatformTransform().setTransform(lookAt);
        }
        if (View.getSelectedIndex() == 2) {
                Transform3D lookAt = new Transform3D();
                lookAt.lookAt(new Point3d(0.0, 0.0, 20), new Point3d(0.0, 0.0, 0.0), new Vector3d(0.0,
                1.0, 0.0));
                lookAt.invert();
                u.getViewingPlatform().getViewPlatformTransform().setTransform(lookAt);
        }
        if (View.getSelectedIndex() == 3) {
                Transform3D lookAt = new Transform3D();
                lookAt.lookAt(new Point3d(-20, 0.0, 0.0), new Point3d(0.0, 0.0, 0.0), new Vector3d(0.0,
                1.0, 0.0));
                lookAt.invert();
                u.getViewingPlatform().getViewPlatformTransform().setTransform(lookAt);
        }
}
```

- Create a KeyNavigatorBehavior instance to make user manipulate the camera view by using keyboard.

```
private void createViewInteractors() {
    TransformGroup viewTransformGroup =
    u.getViewingPlatform().getViewPlatformTransform();
    KeyNavigatorBehavior keyInteractor =
    new KeyNavigatorBehavior(viewTransformGroup);
    keyInteractor.setSchedulingBounds(bounds);
    bgRoot.addChild(keyInteractor);
}
```

- Create a reference object – grid plane.

```
protected Group createGrid() {
    Group lineGroup = new Group();
    Appearance app = new Appearance();
    ColoringAttributes ca = new ColoringAttributes(black,
    ColoringAttributes.SHADE_FLAT);
    app.setColoringAttributes(ca);

    // Plain line
    for(int i = 0; i < 21; i++) {
        Point3f[] plaPts = new Point3f[2];
        float a = (i - 10);
        plaPts[0] = new Point3f(-10.0f, 0.0f, a);
        plaPts[1] = new Point3f(10.0f, 0.0f, a);
        LineArray pla = new LineArray(2, LineArray.COORDINATES);
        pla.setCoordinates(0, plaPts);
```

```
            Shape3D plShape = new Shape3D(pla, app);
            lineGroup.addChild(plShape);

            Point3f[] pla_Pts = new Point3f[2];
            float b = (i - 10);
            pla_Pts[0] = new Point3f(b, 0.0f, -10.0f);
            pla_Pts[1] = new Point3f(b, 0.0f, 10.0f);
            LineArray pla_ = new LineArray(2, LineArray.COORDINATES);
            pla_.setCoordinates(0, pla_Pts);
            Shape3D plShape_ = new Shape3D(pla_, app);
            lineGroup.addChild(plShape_);
        }
        return lineGroup;
}
```

- Use a button to interact with an object or camera view. They are coded in a similar way. The code for move the camera up is given as the example.

```
private void CameraUpActionPerformed(java.awt.event.ActionEvent evt) {
        TransformGroup viewTG =
        u.getViewingPlatform().getViewPlatformTransform();
        Transform3D transform = new Transform3D();
        Matrix4d M = new Matrix4d();
        viewTG.getTransform(transform);
        transform.get(M);
        Transform3D lookAt = new Transform3D();
        lookAt.lookAt(new Point3d(M.m03, M.m13 + 2, M.m23), new Point3d(0.0, 0.0,
        0.0), new Vector3d(0.0, 1.0, 0.0));
        lookAt.invert();
        u.getViewingPlatform().getViewPlatformTransform().setTransform(lookAt);
}
```