Chapter 1

INTRODUCTION

Computer Graphics is one of the most effective and commonly used methods to communicate the processed information to the user. It is widespread today. Computer imagery is found on television, in newspapers, in weather reports, or for example, in all kinds of medical investigation and surgical procedures, displaying the information in the form of graphics instead of simple text.

Computer animation is the art of creating moving images via the use of computers. It is a subfield of computer graphics and animation. Sometimes the target of the animation is the computer itself, but sometimes the target is another medium, such as film, education. It is also referred to as CGI (Computer-generated imagery or computer-generated imaging), especially when used in films.

To create the illusion of movement, an image is displayed on the computer screen then quickly replaced by a new image that is similar to the previous image, but shifted slightly. This technique is identical to the illusion of movement in television and motion pictures.

We are undertaking this mini project is to develop an application of graphics using the concepts that we have learnt in computer graphics theory using the tool like openGL.

Aim : The aim of this project is to implement the animation of the representation of "Bohr's atomic model of lithium atom", using OPENGL as a graphics tool.

Scope : The scope is to use the basic primitives defined in OpenGL library creating geometric objects like sphere of two different size and also provide lighting effect and also providing transformation/rotation. We make use of different concepts such as pushmatrix(), translate(), popmatrix() function to perform Transformation, rotation, scaling and any other manipulations of 3D objects.

Motivation : our motivation is to develop a graphics package to visualize the model proposed by neil'sbohr so that it can be used for teaching aid and we can implement the concepts learnt in computer graphics.

BRIEF DESCRIPTION OF PROJECT :

The Mini Project titled **"BOHR'S ATOMIC MODEL OF LITHIUM ATOM"** is to show that how the model proposed by Neilsbohr is visualized graphically. This model consists of 3 electrons revolving around the nucleus. By pressing the up arrow key the speed of revolution of the electrons is increased and by pressing the down arrow key the speed decreases and also by pressing 'q' it will exit from the output screen

1.1 2D and 3D COMPUTER GRAPHICS

Raster graphic sprites (left) and masks (right) 2D computer graphics are the computer-based generation of digital images—mostly from two-dimensional models, such as 2D geometric models, text, and digital images, and by techniques specific to them. The word may stand for the branch of computer science that comprises such techniques, or for the models themselves.

2D computer graphics are mainly used in applications that were originally developed upon traditional printing and drawing technologies, such as typography, cartography, technical drawing, advertising, etc.. In those applications, the two-dimensional image is not just a representation of a real-world object, but an independent artifact with added semantic value; two-dimensional models are therefore preferred, because they give more direct control of the image than 3D computer graphics, whose approach is more akin to photography than to typography.

3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images. Such images may be for later display or for real-time viewing.

Despite these differences, 3D computer graphics rely on many of the same algorithms as 2D computer vector graphics in the wire frame model and 2D computer raster graphics in the final rendered display. In computer graphics software, the distinction between 2D and 3D is occasionally blurred; 2D applications may use 3D techniques to achieve effects such as lighting, and primarily 3D may use 2D rendering techniques.

3D computer graphics are often referred to as 3D models. Apart from the rendered graphic, the model is contained within the graphical data file. However, there are differences. A 3D model is the mathematical representation of any three-dimensional object (either inanimate or living). A model is not technically a graphic until it is visually displayed. Due to 3D printing, 3D models are not confined to virtual space. A model can be displayed visually as a two-dimensional image through a process called 3D rendering, or used in non-graphical computer simulations and calculations.

1.2 CONCEPTS AND PRINCIPLES

Image

In common usage, an image or picture is an artifact, usually two-dimensional, that has a similar appearance to some subject—usually a physical object or a person. Images may be two-dimensional, such as a photograph, screen display, and as well as a three-dimensional, such as a statue. They may be captured by optical devices—such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water surfaces.

A digital image is a representation of a two-dimensional image using ones and zeros (binary). Depending on whether or not the image resolution is fixed, it may be of vector or raster type. Without qualifications, the term "digital image" usually refers to raster images.

Pixel

The enlarged In portion of the image individual pixels are rendered as squares and can be easily seen.In digital imaging, a pixel is the smallest piece of information in an image. Pixels are normally arranged in a regular 2-dimensional grid, and are often represented using dots or squares. Each pixel is a sample of an original image, where more samples typically provide a more accurate representation of the original. The intensity of each pixel is variable; in color systems, each pixel has typically three or four components such as red, green, and blue, or cyan, magenta, yellow, and black.

Graphics

Graphics are visual presentations on some surface, such as a wall, canvas, computer screen, paper, or stone to brand, inform, illustrate, or entertain. Examples are photographs,

drawings, line art, graphs, diagrams, typography, numbers, symbols, geometric designs, maps, engineering drawings, or other images. Graphics often combine text, illustration, and color. Graphic design may consist of the deliberate selection, creation, or arrangement of typography alone, as in a brochure, flier, poster, web site, or book without any other element. Clarity or effective communication may be the objective, association with other cultural elements may be sought, or merely, the creation of a distinctive style.

Rendering

Rendering is the process of generating an image from a model, by means of computer programs. The model is a description of three dimensional objects in a strictly defined language or data structure. It would contain geometry, viewpoint, texture, lighting, and shading information. The image is a digital image or raster graphics image. The term may be by analogy with an "artist's rendering" of a scene. 'Rendering' is also used to describe the process of calculating effects in a video editing file to produce final video output.

3D Projection: 3D projection is a method of mapping three dimensional points to a two dimensional plane. As most current methods for displaying graphical data are based on planar two dimensional media, the use of this type of projection is widespread, especially in computer graphics, engineering and drafting.

Ray Tracing: Ray Tracing is a technique for generating an image by tracing the path of light through pixels in an image plane. The technique is capable of producing a very high degree of photorealism; usually higher than that of typical scan line rendering methods, but at a greater computational cost.

Shading: Example of Shading

Shading refers to depicting depth in 3D models or illustrations by varying levels of darkness. It is a process used in drawing for depicting levels of darkness on paper by applying media more densely or with a darker shade for darker areas, and less densely or with a lighter shade for lighter areas. There are various techniques of shading including cross hatching where perpendicular lines of varying closeness are drawn in a grid pattern to shade an area. The closer the lines are together, the darker the area appears. Likewise,

the farther apart the lines are, the lighter the area appears. The term has been recently generalized to mean that shades are applied.

Texture Mapping: Texture Mapping is a method for adding detail, surface texture, or color to a computer-generated graphic or 3D model. A texture map is applied (mapped) to the surface of a shape, or polygon. This process is akin to applying patterned paper to a plain white box. Multitexturing is the use of more than one texture at a time on a polygon.

Volume Rendering

Volume rendered CT scan of a forearm with different color schemes for muscle, fat, bone, and blood.

Volume rendering is a technique used to display a 2D projection of a 3D discretely sampled data set. A typical 3D data set is a group of 2D slice images acquired by a CT or MRI scanner.

Usually these are acquired in a regular pattern (e.g., one slice every millimeter) and usually have a regular number of image pixels in a regular pattern. This is an example of a regular volumetric grid, with each volume element, or voxel represented by a single value that is obtained by sampling the immediate area surrounding the voxel.

3D Modeling

3D Modeling is the process of developing a mathematical, wireframe representation of any three-dimensional object via specialized software. The product is called a "3D model". It can be displayed as a two-dimensional image through a process called 3D rendering or used in a computer simulation of physical phenomena. The model can also be physically created using 3D Printing devices. Models may be created automatically or manually. The manual modeling process of preparing geometric data for 3D computer graphics is similar to plastic arts such as sculpting.

Chapter 2

OPENGL

OpenGL provides a set of commands to render a three dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop OpenGL-applications without licensing.

OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation.

Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

2.1 GLUT

GLUT is a complete API written by Mark Kilgard which lets you create windows and handle the messages. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

How OpenGL works

OpenGL bases on the state variables. There are many values, for example the color, that remain after being specified. That means, you can specify a color once and draw several polygons, lines or whatever with this color then. There are no classes like in DirectX. However, it is logically structured. Before we come to the commands themselves, here is another thing: To be hardware independent, OpenGL provides its own data types. They all begin with "GL". For example GLfloat, GLint and so on. There are also many symbolic constants; they all begin with "GL_", like GL_POINTS, GL_POLYGON. Finally the commands have the prefix "gl" like glVertex3f(). There is a utility library called GLU, here the prefixes are "GLU_" and "glu". GLUT commands begin with "glut", it is the same for every library. You want to know which libraries coexist with the ones called before? There are libraries for every system, Windows has the wgl*-Functions, UNIX systems glx* and so on.

A very important thing is to know, that there are two important matrices, which affect the transformation from the 3d-world to the 2d-screen: The projection matrix and the modelview matrix. The projection matrix contains information, how a vertex – let's say a "point" in space – shall be mapped to the screen. This contains, whether the projection shall be isometric or from a perspective, how wide the field of view is and so on. Into the other matrix you put information, how the objects are moved, where the viewer is and so on.

2.2 Rendering Pipeline

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. Although this is not a strict rule of how OpenGL is implemented, it provides a reliable guide for predicting what OpenGL will do. Geometric data (vertices, line, and polygons) follow a path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images and bitmaps) are treated differently for part of the process. Both types of data undergo the same final step (rasterization) before the final pixel data is written to the frame buffer. As shown in **Figure 2.1**.

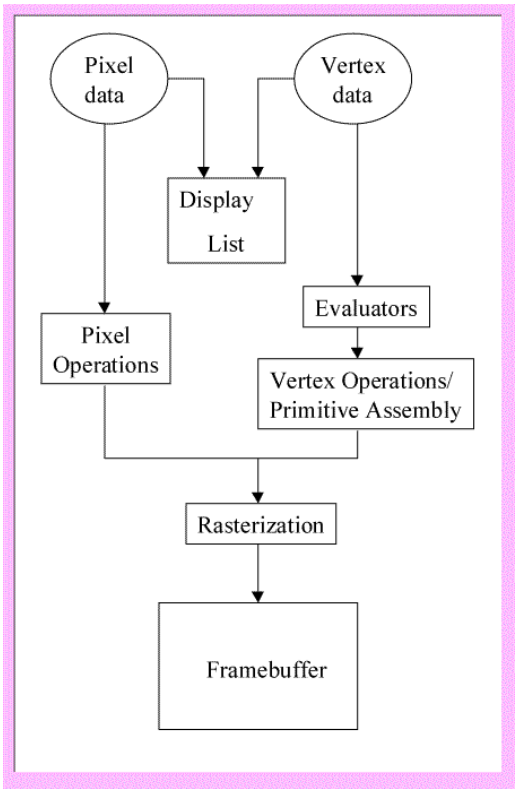

Figure 2.1 Block diagram of Opengl pipeline architecture

2.3 How to use GLUT

GLUT provides some routines for the initialization and creating the window (or full screen mode, if you want to). Those functions are called first in a GLUT application:

In your first line you always write glutInit(&argc, argv). After this, you must tell GLUT, which display mode you want – single or double buffering, color index mode or RGB and so on. This is done by calling glutInitDisplayMode(). The symbolic constants are connected by a logical OR, so you could use glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE). In later tutorials we will use some more constants here.

After the initialization you call glCreateWindow() with the window name as parameter. Then you can (and should) pass some methods for certain events. The most important ones are "reshape" and "display". In reshape you need to (re)define the field of view and specify a new area in the window, where OpenGL is allowed to draw.

Display should clear the so called color buffer – let's say this is the sheet of paper – and draw our objects.You pass the methods by glut*Func(), for example glutDisplayFunc(). At the end of the main function you call glutMainLoop(). This function doesn't return, but calls the several functions passed by glut*Func.

2.4 Double buffering

Double buffering is one of the most basic methods of updating the display. This is often the first buffering technique adopted by new coders to combat flickering.

Double buffering uses a memory bitmap as a *buffer* to draw onto. The buffer is then drawn onto `screen`. If the objects were drawn directly to `screen`, the display could be updated mid-draw leaving some objects out. When the buffer, with all the objects already on it, is drawn onto `screen`, the new image will be drawn over the old one (`screen` should not be cleared). If the display gets updated before the drawing is complete, there may be a noticeable shear in the image, but all objects will be drawn. Shearing can be avoided by using vsync.

In OPENGL, Double Buffering can be enabled by the command

glutInitDisplayMode(GL_DOUBLE);

In order to reap the benefits of double buffering ,i.e. to display the contents of the back buffer, the following instruction must be used.

glutSwapBuffers();

Chapter 3

BOHR'S MODEL OVERVIEW

In atomic physics, the **Bohr model,** devised by Niels Bohr, depicts the atom as a small, positively charged nucleus surrounded by electrons that travel in circular orbits around the nucleus – similar in structure to the solar system, but with electrostatic forces providing attraction, rather than gravity.

Since the Bohr model is quantum physics-based modification of the Rutherford model, many sources combine the two, referring to the **Rutherford-Bohr model**.

He suggested the electrons could only have certain classic motions:

1. The electrons can only travel in certain orbits: at a certain discrete set of distances from the nucleus with specific energies.
2. The electrons of an atom revolve around the nucleus in orbits. These orbits are associated with definite energies and are also called energy shells or energy levels. Thus, the electrons do not continuously lose energy as they travel in a particular orbit. They can only gain and lose energy by jumping from one frequency v determined by the energy difference of the levels according to the Planck relation:

$$\Delta E = E_2 - E_1 = hv$$

where h is Planck's constant.

3. The frequency of the radiation emitted at an orbit of period T is as it would be in classical mechanics; it is the reciprocal of the classical orbit period:

$$v = \frac{1}{T}$$

The significance of the Bohr model is that the laws of classical mechanics applying to the motion of the electron about the nucleus only when restricted by a quantum rule. Although rule 3 is not completely well defined for small orbits, because the emission process involves two orbits with two different periods, Bohr could determine the energy spacing between levels using rule 3

and come to an exactly correct quantum rule: the angular momentum L is restricted to be an integer multiple of a fixed unit:

$$L = n\frac{h}{2\pi} = n\hbar$$

where n = 1, 2, 3, …. is called the principal quantum number, and $\hbar$ = h/2π. The lowest values of *n* is 1; this gives a smallest possible orbital radius of 0.0529nm known as the Bohr radius. Once an electron is in this lowest orbit, it can bet no closer to the proton. Starting from the angular momentum quantum rule Bohr was able to calculate the engines of the allowed orbits of the hydrogen atom and other hydrogen-like atoms and ions.

3.1 Shell model of an atom

Bohr extended the model of Hydrogen to give an approximate model for heavier atoms. Heavier atoms have more protons in the nucleus, and more electrons to cancel the charge. Bohr's idea was that each discrete orbit could only hold a certain number of electrons. After that orbit is full the next level would have to be used. This gives the atom a shell structure, in which each shell corresponds to a Bohr orbit.

In this project, we animate a lithium atom. Lithium has 3 electrons. It has two electrons in the 1 s-subshell and one in the (higher-energy) 2s-subshell, so its configuration is written $1s^2 2s^1$. The pictorial representation of lithium atom is shown below:

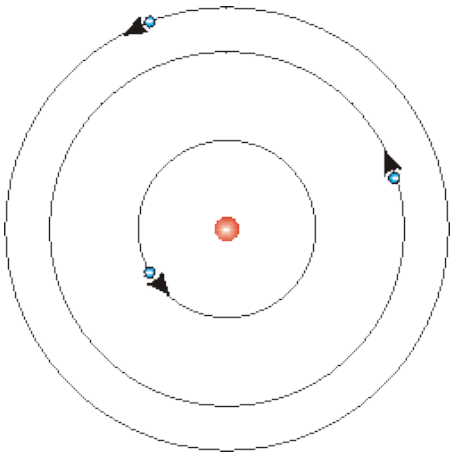

Figure 3.1 Pictorial representation of lithium atom

Chapter 4

SOFTWARE REQUIREMENTS SPECIFICATION

The requirements can be categorized as functional requirement and non-functional requirements

4.1 FUNCTIONAL REQUIREMENTS

Functional requirement defines a function of a software system or its component. A function is described as a set of inputs, the behavior, and outputs. The functional requirements for the successful implementation of the project is that the system must allow the user to control the time of movements of the engine for the respective keys on the keyboard as well as the mouse.

Input: up-arrow key,down-arrow key,'q' are input from keyboard.

Expected Output: creation of Bohr's model along with lighting and shading effect.

I/P Interface: keyboard interfaced.

Behaviour: on pressing of

'up-arrow key': the speed of revolution of electrons can be increased.

'down-arrow key': the speed of revolution of electrons can be decreased.

'q': exit from output screen.

4.2 NON-FUNCTIONAL REQUIREMENTS

Non-Functional Requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. This should be contrasted with functional requirements that define specific behavior or functions.

The minimum Hardware requirements for the project are listed below:

- Processor- Intel or AMD(Advanced Micro Devices)

- RAM- 512MB(minimum)
- Hard Disk-300MB(minimum)
- CPU Frequency : 1.6 GHz
- A Keyboard (Real or Virtual).
- A Mouse (Pointing Device)
- Monitor.

The following operating systems are supported:

- Windows XP Service Pack 2 or above
- Windows Server 2003 Service Pack 1 or above
- Windows Server 2003 R2 or above
- Windows Vista
- Windows Server 2008

The software requirements are:

- Any one of the above mentioned operating systems.
- Microsoft Visual Studio
- Open GL libraries

Chapter 5

DESIGN AND IMPLEMENTATION

5.1 DESIGN

The system is designed using 4 modules which can be represented in the flow diagram.

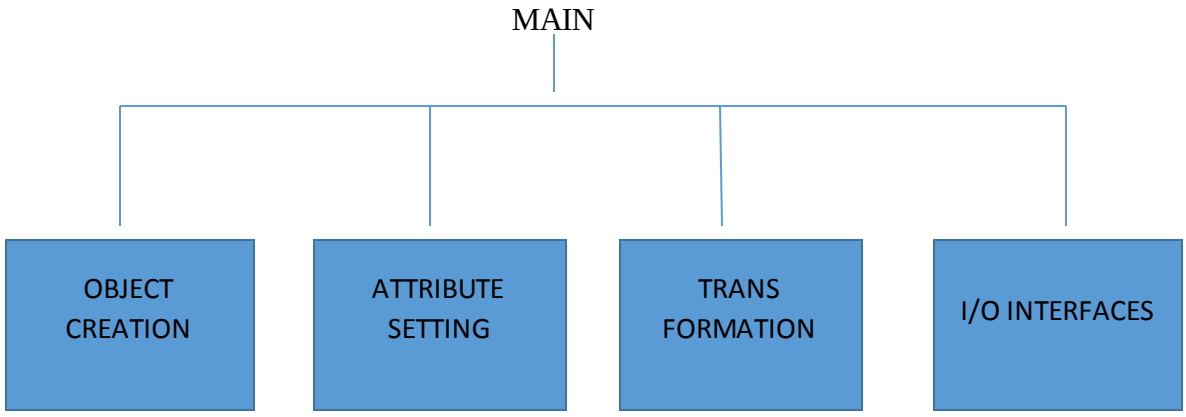

Figure 5.1 Flow diagram of the modules used

Object Creation:

The graphics implemented BOHR'SATOMIC MODDEL OF LITHIUM ATOM is drawn using simple geometric primitive sphere. The primitives are put together at the right co-ordinates to form the complete model.

Attribute Setting:

The attributes govern the way the primitive appears on the display.

Attribute functions allow us to perform operations ranging from choosing the colour with which we display a line segment, to picking a pattern to fill the inside of the polygon, to selecting a type face for the titles on the graph.

Here we use lighting and shading for attribute setting.

Transformation:

One of the characteristics of good API is that it provides the user with a set of transformation function that allows to carry out transformation of the objects such as rotation, translation and scaling.

- **Rotation** rotates the object in the given angle of rotation
- **Translation** is an operation that displaces points by a fixed distance in a given direction
- **Scaling** is an affine non rigid body transformation by which we can make an object bigger and smaller

I/O Interfaces:

One of the most important advances in the computer technology is enabling users to interact with computer displays. The interactions are done by means of keyboard or mouse. The image changes in response to the input. Here keyboard interface is provided and it performs the following functions.

1. 'Up-arrow key': increases the speed of revolution of electrons.
2. 'down-arrow key': decreases the speed of revolution of electrons.
3. 'q': exit from the output screen.

5.2 IMPLEMENTATION

This section provides information about the OpenGL features used, pseudo code and user defined functions used to develop this project.

Graphics features used

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. It is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation, and video games.

OpenGL API uses seven major functions primitive, attribute, viewing, transformation, input, control functions. We make use of most of the above functions in our program. To draw Steam Engine we use primitive functions like points, line, polygons, pixels, circles, line loop, curves and surfaces.

To apply transformations like scaling, translation we use transformation functions provided by openGL and to change the attribute of object like color we used attribute function.All the open gl functions can be found in the header files gl.h, glu.h and glut.h. the header file glut.h is a combination of both gl.h and glu.h and has been included in the project.

Translation: This operation displaces a point by a fixed distance specified by the function call in a given direction and it can be implemented by using the following function.

Scaling: This operation makes the object bigger or smaller and it can be implemented by using the function

Attributes used are:

Coloring: It is the operation in which we assign colors to the objects.OpenGL uses RGB coloring scheme to color the primitives. Each color component is stored separately, usually 8 bits per component. In OpenGL color values range from 0.0 (none) to1.0 (all).

Size: This is used to specifying the size of an object.

Lighting: It is the operation in which simulates how the object reflects the light these feature includes material composition of the object, light's color and its positions and it available in both color such as RGBA and index mode.

Inbuilt functions:

This section presents the OpenGL inbuit functions and user defined functions which are used in our project

voidglRotatef(GLfloat angle, Glfloat x, GLfloat y, GLfloat z)

glRotate produces a rotation of **angle** degrees around the vector (x,y,z).

voidglTranslate (GLfloat x, GLfloat y, GLfloat z)

glTranslate produces a translation of co-ordinate axis to (x, y, z). This function changes the state of the OpenGL.

voidgultBitmapCharacter(void *font, int character)

Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font. Front: Bitmap front to use Character: Character to render (not confined to 8 bits).

voidgluPerspective (float angle, float aspect_ratio, float near, float far)

This function describes the transformation that produces perspective projection. The current matrix is multiplied by this matrix and the result replaces the current matrix.

voidgluSphere (GLuquadric *quadric, GLdouble radius, Glint slices, Glint stacks)

gluSphere creates a sphere of radius radius and divides it into specified number of latitudes and longitudes. This is an quadric object.

User Defined Functions:

This section presents the OpenGL user defined functions which are used in our project

voidKeypressFunc(unsigned char Key, int x, int y)

This function is used to handle all normal key presses. Pressing key 'q' exists the program.

voidSpecialKeyFunc(int Key, int x, int y)

This function is used to handle all special key presses. 'Up Arrow' and 'Down Arrow' keys are the special keys used in this program.

voidKey_up(void)

This function increases the speed of revolution of electrons when 'Up Arrow' key is pressed.

voidKey_down(void)

This function decreases the speed of revolution of electrons when 'Down Arrow' key is pressed.

voidshowMessage(GLfloat x, GLfloat y, GLfloat z, Char *message)

This function is used to display text message on the output screen.

5.3 Pseudo code:

```
beginbohr's{
static void KeyPressFunc( unsigned char Key, int , int )
{
        switch ( Key ) {

        case 'q':          // Escape key
                exit(1);
        }
}
static void SpecialKeyFunc( int Key, int , int )
{
       case 1:up-arrow key
                key_up();
                break;
        case 2: down-arrow key
                Key_down();
                break;
}
static void Key_up(void)
{
```

```
    Double the animation time step
}
static void Key_down(void)
{
Halve the animation time step
}
void showMessage(GLfloat , GLfloat , GLfloat , char *message)
{
 Display the message at the co-ordinates specified
}
static void Display(void)
{
  Set the colour attributes
  Clear the redering window
  Update the animation state
  {
        Orbit += AnimateIncrement/24.0;
                Orbit = Orbit - ((int)(Orbit/365))*365;
  }
 Clear the current matrix (Modelview)
   (rotate the model's plane about the x axis by fifteen degrees)
   Set the material property
    Draw the nucleus as a solid sphere
         Show meaasge "nucleus"
    Show message "li"


     Draw the first electron
         Show message "e-"
     Set the material property
         Position it around the nucleus and determine its position
```

```
        Rotate the first electron on its axis.

Determine the rotation of second electron
Draw the second electron
  Show message "e-"
   Set the material property

Determine rotation of third electron
 Draw the third electron
  Show message "e-"
   Set the material property

}

// Initialize OpenGL's rendering modes
voidOpenGLInit(void)
{
Set the colour property
  Set the material property
  Enable material
}

// ResizeWindow is called when the window is resized
static void ResizeWindow(intw, int h)
{
floataspectRatio;
        h = (h == 0) ?1 : h;
        w = (w == 0) ?1 : w;

        Set up the projection view matrix (not very well!)
         Select the Modelview matrix
```

```
    }

 Main routine
 Set up OpenGL, hook up callbacks, and start the main loop
int main( intargc, char** argv )
{
        Need to double buffer for animation

        Create and position the graphics window

        Initialize OpenGL.

        Set up callback functions for key presses

        Set up the callback function for resizing windows
 Callback for graphics image redrawing


        Start the main loop. glutMainLoop never returns.

return(0);
}
```

OpenGL API's

void glvertex3f (foat x, float y, float z)

glvertex3f inserts a point to the respective co-ordinate specified by (x,y,z).

void glColor3f (int r, int g, int b)

glColor3f specifies the combination of the primary colors Red Green and Blue.

voidglMatrixMode (Glenum mode)

Specifies which matrix stack is the target for subsequent matrix operations. There values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The initial value is GL_MODELVIEW.

voidglutMainLoop(void)

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

voidglViewport(void)

The arguments to glViewPort() describe the origin of the available screen space within the window and the width and height of the available screen area, all measured in pixels on the screen.

Chapter 6

TESTING

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing is actually a series of different test whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work should verify that all system elements have been properly integrated and perform allocated functions.There are several rules that serve as testing objectives

- Testing is a process of executing program with the intent of finding an error.
- A good test case is one that has a high probability of finding an undiscovered error.
- A successful test is one that uncovers an undiscovered error.

The testing has been conducted successfully according to the objectives stated and hence it uncovers error in the software.

The test case of the project is shown below in the table:

Test case ID	Test case Description	Input	Actual Output	Expected Output	Remark
1	To increase speed of revolution of electrons	Press Up-arrow key	Speed is increased	Increases the speed of revolution of electrons	Pass
2	To decrease speed of revolution of electrons	Press Down-arrow key	Speed is decreased Refer **Figure 7.2**	Decreases the speed of revolution of electrons	Pass

3	To exit	'q'	Exit	Exit from the output screen	Pass

Figure 6.1 Test case table

Chapter 7

SNAPSHOTS

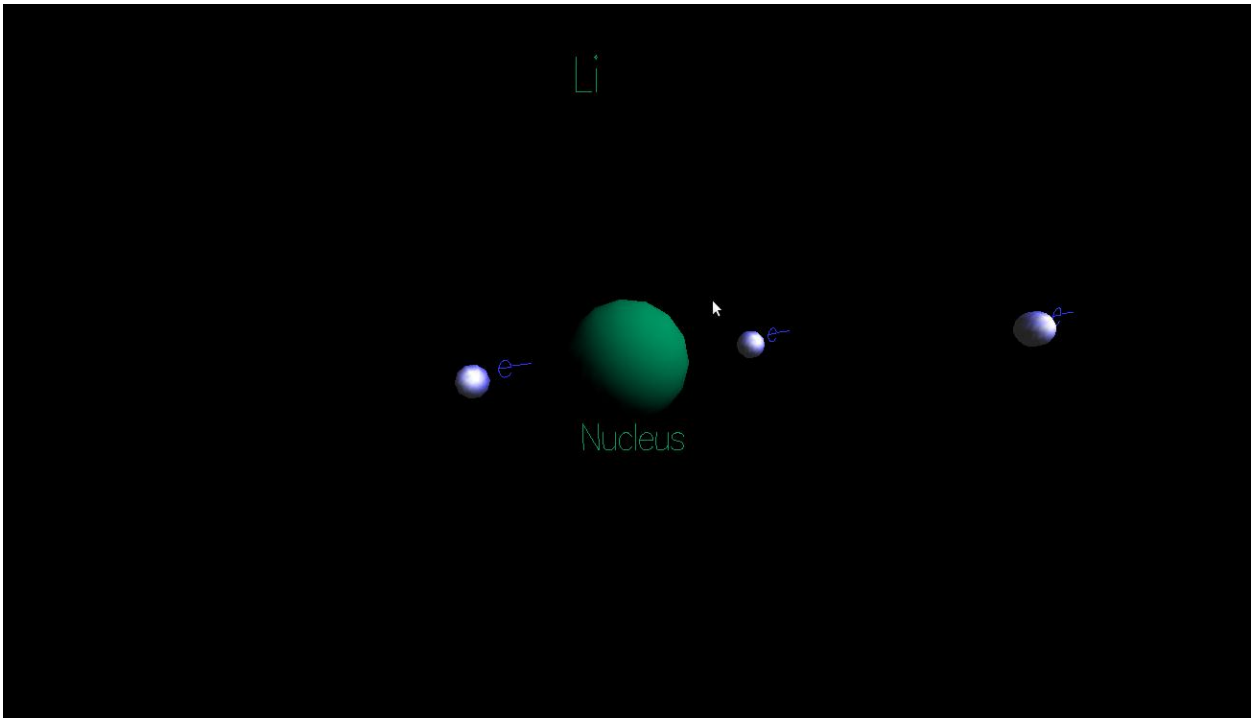

Figure 7.1 Revolution of electrons

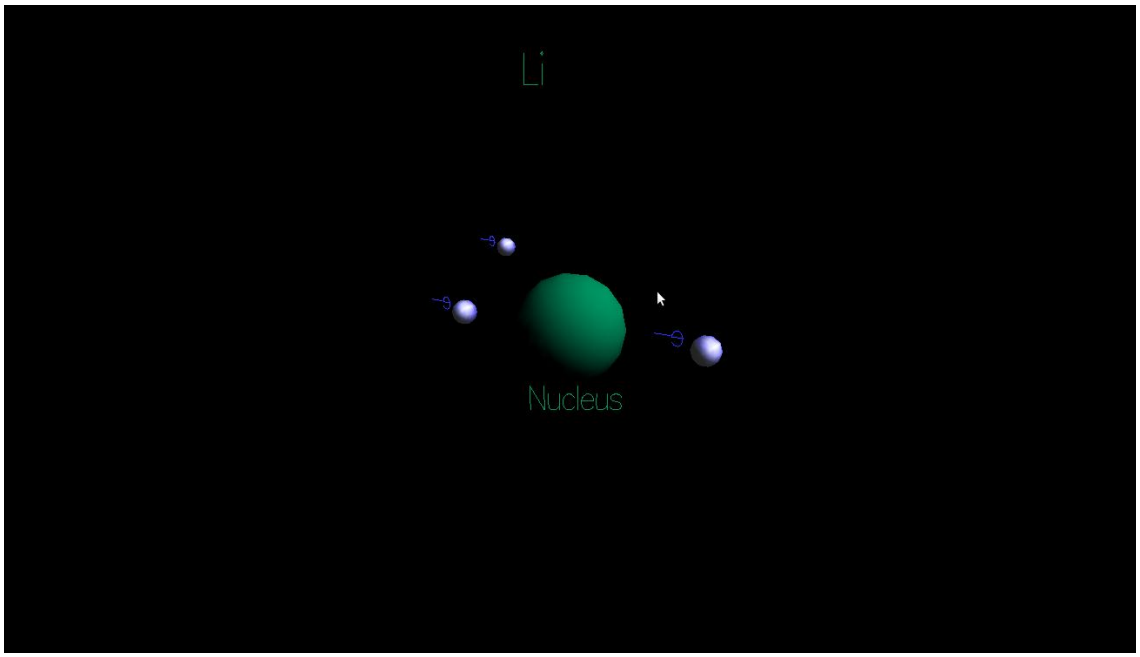

Figure 7.2 Revolution of electrons with decreased speed

CONCLUSION

An attempt has been made to develop an OpenGL graphics package which meets all the necessary requirements that were set out. It is user friendly and provides an easy interaction for the user. The user can very easily use this tool to draw or manipulate a drawing. The interfaces are mouse and keyboard driven and the user can select a function by clicking on an option representing that function or by pressing keys in the keyboard. I finally conclude that this graphics package satisfies all requirements and provides good entertainment.

Working on this project has been a rewarding experience for us. This project was bit time consuming, but gave lots of knowledge, skills for us. While developing this project we assume that this project will serve its needful work at its best with minimum conflicts. On successful implementation of this project, it would provide better understanding of OpenGL functions.

- This project helped us to know in detail of graphics API named OpenGL practically.
- It also helped us to know many functions that are utilized in OpenGL API.
- The output of this project is useful in understanding the concept of Bohr's Atomic Model as it provides a visual animation of the movement of electrons in their orbits.

FUTURE ENHANCEMENT

This project can be implemented to represent the atomic structure of various other elements having more number of electrons.It can be implemented to animate the electronic configuration of various elements. The future versions of this project may be developed in a 3-D platform for a much better user interface, by providing Zooming in and zooming out effects. Menus could also be provided. This project can also be used as a screensaver. This project can be of lot of scope in animation field with more graphical options. Can be used demonstrate tiny OpenGL projects. Interactive animated movies can be developed.

BIBLIOGRAPHY

Text Book References:

- INTERACTIVE COMPUTER GRAPHICS, A Top Down Approach Using OpenGL, Edward Angel, 5th Edition, Pearson Education.
- COMPUTER GRAPHICS PRINCIPLES AND PRACTICE, Foley, Vandam, Feiner, Hughes, 2nd Edition in C, Pearson Education.

Web References:

- www.opengl.org
- www.wikipedia.com