# VISVESVARAYA TECHNOLOGICAL UNIVERSITY BELGAUM-590014



### A Computer Graphics and Visualization Mini-Project Report

**On**

***“PERIODIC VIEWER”***

*A Mini-project report submitted in partial fulfilment of the requirements for the award of the degree of* ***Bachelor of Engineering in Computer Science and Engineering*** *of Visvesvaraya Technological University, Belgaum.*

Submitted by:

##### ABHAY SUBRAMANIAN K (1DT16CS002)

Under the Guidance of:

##### Mr. ANOOP.G.L

**(Asst. Prof.Dept of CSE)**

**Department of Computer Science and Engineering**

**DAYANADA SAGAR ACADEMY OF TECHNOLOGY AND MANAGAEMENT**

Kanakpura Road,Udayapura, Bangalore

2018-2019



**DAYANADA SAGAR ACADEMY OF TECHNOLOGY AND MANAGAEMENT,**

Kanakpura Road,Udayapura, Bangalore

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CERTIFICATE**

This is to certify that the Mini-Project on Computer Graphics and Visualization work entitled **“PERIODIC VIEWER USING OPENGL”** has been successfully carried out by **ABHAY SUBRAMANIAN K (1DT16CS002)** are bonafide student of **Dayananda Sagar Academy of Technology and Management** in partial fulfilment of the requirements for the award of degree in **Bachelor of Computer Science and Engineering** of **Visvesvaraya Technological University, Belgaum** during academic year 2018-2019. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of project work for the said degree.

##### GUIDES:

|  |  |
| --- | --- |
| **Mr. ANOOP G L** | **Mrs. JAHNAVI SHANKAR** |
| **(Asst. Prof. Dept. of CSE)** | **(Asst. Prof. Dept. of CSE)** |
|  |  |

**Dr. C. NANDINI**

**(Prof &HOD, Dept. of CSE)**

**Examiners:**

**1:**

**2: Signature with Date**

**ABSTRACT**

The Atomizer program allows the user to load any of the elements from the periodic table and view them represented in a 3D model. It provides both 2D (1 plane of rotation) and 3D (3 planes of rotation).

Additionally, the user can toggle between mono and stereo mode. Stereo mode creates an anaglyph image that can be viewed using your standard 3D glasses as those from the old 3D movies, red plastic in one lens and green (or blue) in the other lens. The display() function has two places where colour filtering can be set to either green or blue, depending on your glasses.

The atomic model displayed using Atomizer program depicts real life simulation of an atom. The electrons move in their respective orbits according to their distribution, while the nucleus remains in the centre. The position of electron in each stage is calculated using trigonometric functions to get probable angular position of electron in its respective orbit around all three axes. The entire model can be rotated along y-axis for surrounding view. The speed of rotation can be changed. The model can be further zoom-in zoom-out for detail view. We can focus camera in and out, and can increase and decrease parallax ( adjust 3D image for your eyes).

The user interface has been designed using menus and submenus to load the desired atom for simulation. These menus can be accessed using pointing device like mouse. The key functions are added to provide user interaction during run time. This makes the program a suitable learning tool.

We make use of C with OpenGl for entire coding purpose along with some features of Windows. The OpenGl Utility is a Programming Interface. We use light and material functions to add luster, shade and shininess to graphical objects. The toolkit supports much functionalities like multiple window rendering, callback event driven processing using sophisticated input devices etc.

**ACKNOWLEDGEMENT**

It gives us immense pleasure to present before you our project titled **‘PERIODIC VIEWER USING OPENGL’.** The joy and satisfaction that accompany the successful completion of any task would be incomplete without the mention of those who made it possible. We are glad to express our gratitude towards our prestigious institution **DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT** for providing us with utmost knowledge, encouragement and the maximum facilities in undertaking this project.

We wish to express a sincere thanks to our respected principal **Dr. B. R.Lakshmikantha**

for all their support.

We express our deepest gratitude and special thanks to **Dr.C.Nandini**, **Prof &H.O.D, Dept. Of Computer Science Engineering**, for all her guidance and encouragement.

We sincerely acknowledge the guidance and constant encouragement of our mini-project guides, **Assistant Prof. Mr. Anoop G. L**

ABHAY SUBRAMANIAN K (1DT16CS002)

AND

CHOUTHIK M SHETTY (1DT16CS025)

**TABLE OF CONTENT**

1. INTRODUCTION
   1. Computer Graphics
   2. OpenGL Technology
   3. Project Description
   4. Functions Used

#### REQUIREMENT SPECIFICATION

* 1. Hardware Requirements
  2. Software Requirements

#### INTERFACE AND ARCHITECTURE

* 1. Interface
  2. Architecture

#### IMPLEMENTATION

1. SNAPSHOTS
2. FUTURE ENHANCEMENT
3. CONCLUSION
4. REFERENCES
5. APPENDIX
   1. User Manual
   2. Personal Details

# Chapter-1

**INTRODUCTION**

## Computer Graphics

#### The term **computer graphics** includes almost everything on computers that is not text or sound. Today almost every computer can do some graphics, and people have even come to expect to control their computer through icons and pictures rather than just by typing.

Here in our lab at the Program of Computer Graphics, we think of computer graphics as drawing pictures on computers, also called *rendering*. The pictures can be photographs, drawings, movies, or simulations -- pictures of things which do not yet exist and maybe could never exist. Or they may be pictures from places we cannot see directly, such as medical images from inside your body.

We spend much of our time improving the way computer pictures can simulate real world scenes. We want images on computers to not just look more realistic, but also to BE more realistic in their colors, the way objects and rooms are lighted, and the way different materials appear. We call this work "realistic image synthesis", and the following series of pictures will show some of our techniques in stages from very simple pictures through very realistic ones.

* 1. **OpenGL Technology**

OpenGL is a graphics application programming interface (API) which was originally developed by Silicon Graphics. OpenGL is not in itself a programming language, like C++, but functions as an API which can be used as a software development tool for graphics applications. The term Open is significant in that OpenGL is operating system independent. GL refers to graphics language. OpenGL also contains a standard library referred to as the OpenGL Utilities (GLU). GLU contains routines for setting up viewing projection matrices and describing complex objects with line and polygon approximations.

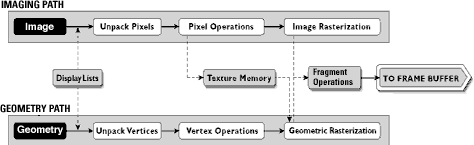
OpenGL gives the programmer an interface with the graphics hardware. OpenGL is a low-level, widely supported modelling and rendering software package, available on all platforms. It can be used in a range of graphics applications, such as games, CAD design,

modelling.

OpenGL is the core graphics rendering option for many 3D games, such as Quake 3. The providing of only low-level rendering routines is fully intentional because this gives the programmer a great control and flexibility in his applications. These routines can easily be used to build high-level rendering and modelling libraries. The OpenGL Utility Library (GLU) does exactly this, and is included in most OpenGL distributions! OpenGL was originally developed in 1992 by Silicon Graphics, Inc, (SGI) as a multi-purpose, platform independent graphics API. Since 1992 all of the development of OpenGL

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you’re using. The OpenGL Visualization Programming Pipeline:-



*fig(1.1) OpenGl Visualization Programming Pipeline*

OpenGL operates on image data as well as geometric primitives.

Simplifies Software Development, Speeds Time-to-Market

Routines simplify the development of graphics software—from rendering a simple geometric point, line, or filled polygon to the creation of the most complex lighted and texture- mapped NURBS curved surface. OpenGL gives software developers access to geometric and image primitives, display lists, modeling transformations, lighting and texturing, anti-aliasing, blending, and many other features. Every conforming OpenGL implementation includes the full complement of OpenGL functions. The well-specified OpenGL standard has language bindings for C, C++, Fortran, Ada, and Java. All licensed OpenGL implementations come from a single specification and language binding document and are required to pass a set of conformance tests. Applications utilizing OpenGL functions are easily portable across a wide array of platforms for maximized programmer productivity and shorter time-to-market.

All elements of the OpenGL state—even the contents of the texture memory and the frame buffer—can be obtained by an OpenGL application. OpenGL also supports visualization applications with 2D images treated as types of primitives that can be manipulated just like 3D geometric objects. As shown in the OpenGL visualization programming pipeline diagram above.

* 1. **PROJECT DESCRIPTION:**

The project is aimed to provide a graphical tool to understand the atomic structure of elements. The program has a wide scope in the educational field to provide an interactive and interesting

mode of learning to the students of chemistry. The stereo mode provides a medium of better understanding of elements at sub-atomic level.

Setting the present program as an example we explain the planar rotation of bodies i.e. rotation along 2D plane and 3D plane.

The program can also inspire other menu based stored data applications to implement a wide variety of applications and learning tool such as- excel sheet to graph generator to study changing market pattern , weather pattern analysis etc.

### Objectives

1. To create a sphere representing nucleus of atom and to create smaller spheres representing electrons which, rotate around nucleus. 2. To load the information about the electron distribution using suitable menus and sub-menus.

1. To provide rotation of electron in 2D plane or 3D plane using the concept rotation of electrons about individual axis.
2. To create anaglyph images in order to provide images visible using 3D glasses in Stereo Mode of display and a suitable interface to change between stereo and mono mode.
3. To provide following controls:-

### Controls

Mouse-Right-Click - Load an Element

Mouse-Left-Click - Toggle between 2D and 3D M – Reset display

S - Display nuclear fission mode P - Pause

U - Unpause

'+' and '-' - Increase and Decrease paralax (adjust the 3D image for your eyes) '[' and ']' - Rotate left and right (around y-axis)

'<' and '>' - Increase and Decrease rotation speed 'a' and 'z' - Fine zoom in and out.

Shift-A and Shift-Z - Larger incremental zoom in and out. 'F' and 'V' - Focus cameras in and out.

### OpenGl Functions Used:

##### glLightfv Function

The glLightfv function returns light source parameter values.

SYNTAX:

void glLightfv(GLenum light, GLenum pname, GLfloat \*params); PARAMETERS:

1. light

The identifier of a light. The number of possible lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL\_LIGHTi where i is a value: 0 to GL\_MAX\_LIGHTS - 1.

1. Pname

A light source parameter for light. The following symbolic names are accepted: GL\_DIFFUSE : The params parameter contains four integer or floating-point values that specify the diffuse RGBA intensity of the light. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative represent able value maps to -

1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default diffuse intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default diffuse intensity of light zero is (1.0, 1.0, 1.0, 1.0).

GL\_SPECULAR : The params parameter contains four integer or floating-point values that specify the specular RGBA intensity of the light. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative represent able value maps to 1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default specular intensity is (0.0, 0.0, 0.0, 1.0) for all lights other than light zero. The default specular intensity of light zero is (1.0, 1.0, 1.0, 1.0).

GL\_AMBIENT : The params contains four integer or floating-point values that specify the ambient RGBA intensity of the light. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative representable value maps to -1.0

. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient light intensity is (0,0, 0, 1). glLightfv(GL\_LIGHT1,GL\_POSITION,pos);

##### glPushMatrix, glPopMatrix Function

The glPushMatrix and glPopMatrix functions push and pop the current matrix stack. SYNTAX:

void glPushMatrix, glPopMatrix(void); PARAMETERS:

This function has no parameters. glPushMatrix();

glPopMatrix();

##### glEnable, glDisable Function

The glEnable and glDisable functions enable or disable OpenGL capabilities. SYNTAX:

void glEnable, glDisable(GLenum cap); PARAMETERS:

cap

A symbolic constant indicating an OpenGL capability. glEnable(GL\_CULL\_FACE); glDisable(GL\_CULL\_FACE);

* + 1. **glColor3f Function** Sets the current color. SYNTAX:

void glColor3f(GLfloat red, GLfloat green, GLfloat blue); PARAMETERS:

1. red: The new red value for the current color.
2. Green: The new green value for the current color.
3. Blue: The new blue value for the current color.

##### glRotatef Function

The glRotated and glRotatef functions multiply the current matrix by a rotation matrix. SYNTAX:

void glRotate( GLfloat angle, GLfloat x, GLfloat y, GLfloat z); PARAMETERS:

angle: The angle of rotation, in degrees. x: The x coordinate of a vector.

y: The y coordinate of a vector. z: The z coordinate of a vector. Example-

glRotatef(xrot,1.0,0.0,0.0);

##### glTranslate Function

The glTranslated and glTranslatef functions multiply the current matrix by a translation matrix. SYNTAX:

void glTranslate( x, y, z); PARAMETERS:

x, y, z

The x, y, and z coordinates of a translation vector. glTranslatef(0.0,0.0,-1.0);

##### glClear Function

The glClear function clears buffers to preset values. SYNTAX:

glClear(GLbitfield mask); PARAMETERS:

mask

Bitwise OR operators of masks that indicate the buffers to be cleared. The four masks are as follows.

Value Meaning

GL\_COLOR\_BUFFER\_BIT The buffers currently enabled for color writing. GL\_DEPTH\_BUFFER\_BIT The depth buffer. glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

##### glMatrixMode Function

The glMatrixMode function specifies which matrix is the current matrix. SYNTAX:

void glMatrixMode(GLenum mode); PARAMETERS:

mode: The matrix stack that is the target for subsequent matrix operations. The mode parameter can assume one of three values:

Value Meaning

GL\_MODELVIEW Applies subsequent matrix operations to the

modelview matrix stack.

glMatrixMode(GL\_MODELVIEW);

##### glLoadIdentity Function

The glLoadIdentity function replaces the current matrix with the identity matrix. SYNTAX:

void glLoadIdentity(void); PARAMETERS:

This function has no parameters. glLoadIdentity();

* + 1. **glColor4f Function** Sets the current color. SYNTAX:

void glColor4f(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha); PARAMETERS:

red

The new red value for the current color. green

The new green value for the current color. blue

The new blue value for the current color. alpha

The new alpha value for the current color. Example-

glColor4f(0.0,0.0,0.0,0.05);

##### glViewport Function

The glViewport function sets the viewport. SYNTAX:

void glViewport(x, y,width, height); PARAMETERS:

x, y

The lower-left corner of the viewport rectangle, in pixels. The default is (0,0). width, height

The width and height, respectively, of the viewport. When an OpenGL context is first attached to a window, width and height are set to the display.

glViewport(0,0,w,h);

##### gluPerspective Function

set up a perspective projection matrix SYNTAX:

void gluPerspective( GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar ); PARAMETERS:

fovy Specifies the field of view angle, in degrees, in the y direction.

aspect Specifies the aspect ratio that determines the field of view in the x direction. The aspect ratio is the ratio of x (width) to y (height).

zNear Specifies the distance from the viewer to the near clipping plane (always positive). zFar Specifies the distance from the viewer to the far clipping plane (always positive). Example-

gluPerspective(50.0,(float)w/(float)h,1.0,20.0);

##### glMaterialfv Function

The glMaterialfv function specifies material parameters for the lighting model.

SYNTAX:

void glMaterialfv(GLenum face, GLenum pname, const GLfloat params); PARAMETERS:

face

The face or faces that are being updated. Must be one of the following: GL\_FRONT, GL\_BACK, or GL\_FRONT and GL\_BACK.

pname

The material parameter of the face or faces being updated. The parameters that can be specified using glMaterialfv, and their interpretations by the lighting equation, are as follows.

GL\_SPECULAR:The params parameter contains four integer or floating-point values that specify the seculars RGBA reflectance of the material. Integer values are mapped linearly such that the most positive represent able value maps to 1.0, and the most negative represent able value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The default specular reflectance for both front-facing and back-facing materials is (0.0, 0.0, 0.0, 1.0).

glMaterialfv(GL\_FRONT,GL\_SPECULAR,white);

##### glutDisplayFunc Function

glutDisplayFunc sets the display callback for the current window. SYNTAX:

void glutDisplayFunc(void (\*func)(void)); PARAMETERS:

func

The new display callback function. glutDisplayFunc(display);

##### glutReshapeFunc Function

glutReshapeFunc sets the reshape callback for the current window. SYNTAX:

void glutReshapeFunc(void (\*func)(int width, int height)); PARAMETERS:

func: The new reshape callback function. glutReshapeFunc(reshape);

##### glutSolidSphere

Syntax :

void glutSolidSphere(GLdouble radius,GLint slices, GLint stacks); 1.radius The radius of the sphere.

2.slices The number of subdivisions around the Z axis (similar to lines of longitude). 3.stacks The number of subdivisions along the Z axis (similar to lines of latitude).

Description: Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

##### glutInit Function

glutInit is used to initialize the GLUT library. SYNTAX:

glutInit(int \*argcp, char \*\*argv); PARAMETERS:

argcp

A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.

Argv

The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

glutInit(&argc,argv);

* + 1. **glutInitDisplayMode Function** glutInitDisplayMode sets the initial display mode. SYNTAX:

void glutInitDisplayMode(unsigned int mode); PARAMETERS:

Mode Display mode, normally the bitwise OR-ing of GLUT display mode bit masks. See values below:

GLUT\_RGB: An alias for GLUT\_RGBA.

GLUT\_DOUBLE: Bit mask to select a double buffered window. This overrides GLUT\_SINGLE if it is also specified.

GLUT\_DEPTH: Bit mask to select a window with a depth buffer. glutInitDisplayMode(GLUT\_RGB|GLUT\_DEPTH|GLUT\_DOUBLE);

##### glutInitWindowPosition, glutInitWindowSize Functions

glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively.

SYNTAX:

void glutInitWindowSize(int width, int height); void glutInitWindowPosition(int x, int y); PARAMETERS:

width

Width in pixels. height

Height in pixels. x

Window X location in pixels. y

Window Y location in pixels. glutInitWindowSize(300,300);

* + 1. **glutCreateWindow Function** glutCreateWindow creates a top-level window. SYNTAX:

int glutCreateWindow(char \*name); PARAMETERS:

name

ASCII character string for use as window name. glutCreateWindow("two pass mirror");

##### glutMainLoop Function

glutMainLoop enters the GLUT event processing loop. SYNTAX:

void glutMainLoop(void); glutMainLoop();

# Chapter-2

**MINIMUM SYSTEM REQUIREMENTS**

## Hardware requirements:

* + - * Pentium or higher processor.
      * 128 MB or more RAM.
      * A standard keyboard, and Microsoft compatible mouse
      * VGA monitor.

## Software requirements:

* + - * The graphics package has been designed for OpenGL; hence the machine must Have Dev C++.
      * Software installed preferably 6.0 or later versions with mouse driver installed.
      * GLUT libraries, Glut utility toolkit must be available.
      * Operating System**:** Windows
      * Version of Operating System**:** Windows XP, Windows NT and Higher
      * Language**:** C

# Chapter-3

**INTERFACE AND ARCHITECTURE**

OpenGL is a software tool for developing the graphics objects. OpenGL library called GLUT i.e. Graphics Library Utility toolkit supports graphics system with the necessary modelling and rendering techniques. The Lighting system is a technique for displaying graphic objects on the monitor and displaying the light effects. It provides the following functionalities.

### Initialization

This function is the initial stage of the system where the system initializes the various aspects of the graphics system based on the user requirements, which include Command line processing, window system initialization and also the initial window creation state is controlled by these routines. This also includes the window and menu management functions which are routines to create and control windows and menus.

### Event Processing

This routine enters GLUT’s event processing loop. This routine never returns, and it continuously calls GLUT callback as and when necessary. This can be achieved with the help of the callback registration functions. These routines register callbacks to be called by the GLUT event processing loop.

### Geometric Shape Rendering

These routines allow the rendering of 3D geometric objects including torus, cylinder, dodecahedrons and teapots. The rendering programmer can also include user defined functions to implement the various techniques such as the animation of the objects, modelling based on the real world objects etc.

### Algorithm

Step 1: Initialize the stage details about electrons of atom in user-defined data structure. Step 2: Initialize parameters of electrons in respective shell.

Step 3: Build stage of electron about X-axis. Step 4: Build stage of electron about Y-axis. Step 5: Build stage of electron about Z-axis.

Step 6: Initialize functions for rotation and redisplay.

Step 7: Colour mask to switch between mono and stereo mode.

Step 8: Create menu and sub-menus for selection of appropriate functions by the user. Step 9: display the objects with effects and options.

Step 10: End

### Data Flow Diagram

START

Stored

Data

MAIN( )

OpenGl Fuction Initialization

|  |  |  |
| --- | --- | --- |
|  | Keyboard Func() |  |

|  |  |  |
| --- | --- | --- |
|  | Mouse Func( ) |  |

|  |  |  |
| --- | --- | --- |
|  | Reshape Func( ) |  |

|  |  |  |
| --- | --- | --- |
|  | Idle Func( ) |  |

|  |  |  |
| --- | --- | --- |
|  | Display( ) |  |

*Fig(3.1) Flow diagram of program*

# Chapter-4

**IMPLEMENTATION**

* 1. **User Defined Structures Used**

We use different user defined structure to store information and state of element.

Structure to store information about element:

struct elements\_info

{

char \*name; char \*symbol; float eno[7];

};

Where, eno[] stores the no of electrons in each shell, symbol stores the atomic symbol and name contains the name of very element.

Structure to store Stage coordinates:

struct element\_stage\_info

{

float elements; float rot\_x; float rot\_y; float rot\_z; float x[120]; float y[120]; float z[120];

};

This structure is used to store the elements electron position in a particular state. Where rot\_x,rot\_y,rot\_z stores the parameters for rotation about respective axis & direction of rotation and x[],y[],z[] stores the coordinate of electron.

### Important Program Modules:

To build a stage of electron around any axis:

Pseudo code to Build a stage of electron around x-axis:

angle\_size = 360/stage[n].elements; radius = n \* stage\_distance;

for (int i=0; i < stage[n].elements; i++)

{

if (angle > 180)

{

}

else

{

}

stage[n].y[i] = (cos((two\_pi\*angle)/360)) \* radius;

stage[n].z[i] = sqrt(( (radius\*radius) - (stage[n].y[i]\*stage[n].y[i]) )); stage[n].x[i] = 0;

stage[n].y[i] = (cos((two\_pi\*angle)/360)) \* radius;

stage[n].z[i] = (-1)\*sqrt(( (radius\*radius) - (stage[n].y[i]\*stage[n].y[i]) )); stage[n].x[i] = 0;

angle = angle + angle\_size;

}//end for

Here the coordinates for electrons in shell is assigned as follows: Y=cosine(angle)\*Radius;

Z=sine(angle)\*Radius or Z=Sqrt(Radius\*Radius-Y\*Y).//Using Pythagoras Theorem Similarly we can build stage for electrons around y or z axis.

To set parameters for rotation for each electron:

if (flat == 1)//for 2D rotation

{

stage[1].rot\_x = 0.0; stage[1].rot\_y = 0.0; stage[1].rot\_z = 1.0;

stage[2].rot\_x = 0.0; stage[2].rot\_y = 0.0; stage[2].rot\_z = -1.0;

stage[3].rot\_x = 0.0; stage[3].rot\_y = 0.0; stage[3].rot\_z = 1.0;

stage[4].rot\_x = 0.0; stage[4].rot\_y = 0.0; stage[4].rot\_z = -1.0;

stage[5].rot\_x = 0.0; stage[5].rot\_y = 0.0; stage[5].rot\_z = 1.0;

stage[6].rot\_x = 0.0; stage[6].rot\_y = 0.0; stage[6].rot\_z = -1.0;

stage[7].rot\_x = 0.0; stage[7].rot\_y = 0.0; stage[7].rot\_z = 1.0;

}

if (flat == 0)//for 3D model

{

stage[1].rot\_x = 0.0; stage[1].rot\_y = 1.0; stage[1].rot\_z = 0.0;

stage[2].rot\_x = 1.0; stage[2].rot\_y = 0.0; stage[2].rot\_z = 0.0;

stage[3].rot\_x = 0.0; stage[3].rot\_y = 0.0; stage[3].rot\_z = 1.0;

stage[4].rot\_x = 0.0; stage[4].rot\_y = 1.0; stage[4].rot\_z = 1.0;

stage[5].rot\_x = 1.0; stage[5].rot\_y = 0.0; stage[5].rot\_z = 1.0;

stage[6].rot\_x = 0.0; stage[6].rot\_y = -1.0; stage[6].rot\_z = -1.0;

stage[7].rot\_x = -1.0; stage[7].rot\_y = 0.0; stage[7].rot\_z = -1.0;

}

Here we assign rotation parameters with value 0= no rotation about the respective axis

1=rotation about respective axis in clockwise direction

-1=rotation about respective axis in anticlockwise direction

* 1. **Header files:** The <stdlib.h> Header File # include <stdlib.h>

The ISO C standard introduced this header file as a place to declare certain standard library functions. These include the Memory management functions( malloc, free) communication with the environment (abort, exit) and others. Not yet all the standard functions of this header file are supported. If a declaration is present in the supplied header file, then CR supports it and will continue to support it. If a function is not there, it will be added in time.

The <stdio.h> Header File # include <stdio.h>

Load the file SIMPLEIO.C for our first look at a file with standard I/O. Standard I/O refers to the most usual places where data is either read from, the keyboard, or written to, the video monitor. Since they are used so much, they are used as the default I/O devices and do not

need to be named in the Input/output instructions. This will make more sense when we actually start to use them so lets look at the file in front of you.

The first things you will notice is the second line of the file, the #include "stdio.h" line. This is very much like the #define we have already studied, except that instead of a simple substitution, an entire file is read in at this point. The system will find the file named "stdio.h" and read its entire contents in, replacing this statement. Obviously then, the file named "stdio.h" must contain valid C source statements that can be compiled as part of a program. This particular file is composed of several standard #defines to define some of the standard I/O operations. The file is called a header file and you will find several different header files on the source disks that came with your C compiler. Each of the header files has a specific purpose and any or all of them can be included in any program.

GL/glut.h header #include <GL/glut.h>

The OpenGL Utility Toolkit (GLUT) is a programming interface with ANSI C and FORTRAN bindings for writing window system independent OpenGL programs. The toolkit supports the following functionality:

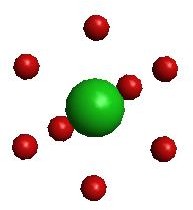
* + 1. Multiple windows for OpenGL rendering.
    2. Callback driven event processing.
    3. Sophisticated input devices.
    4. An ``idle'' routine and timers.
    5. A simple, cascading pop-up menu facility.

Our C compiler uses the double quote marks to indicate that the search for the "include" file will begin in the current directory, and if it not found there, the search will continue in the "include" directory as set up in the environment. It also uses the "less than" and "greater than" signs to indicate that the file search should begin in the directory specified in the environment. Most of the programs in this tutorial have the double quotes in the "include" statements. The next program uses the "<" and ">" to illustrate the usage. Note that this will result is a slightly faster (but probably unnoticeable) compilation because the system will not bother to search the current directory.

# Chapter-5

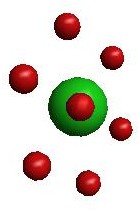
**SNAPSHOTS**

The given fig(5.1) is 2D display of atomic structure of Oxygen(8). The red balls are electrons rotating in their respective shell around the nucleus (the green sphere). This is default display of our program or can be selected by left click->2D Model from menu.



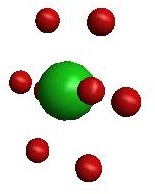
*Fig(5.1) 2D Model of O atom*

The next figure fig(5.2) is 3D display of atomic model of Oxygen. The model can be selected by left click->3D Model in menus.



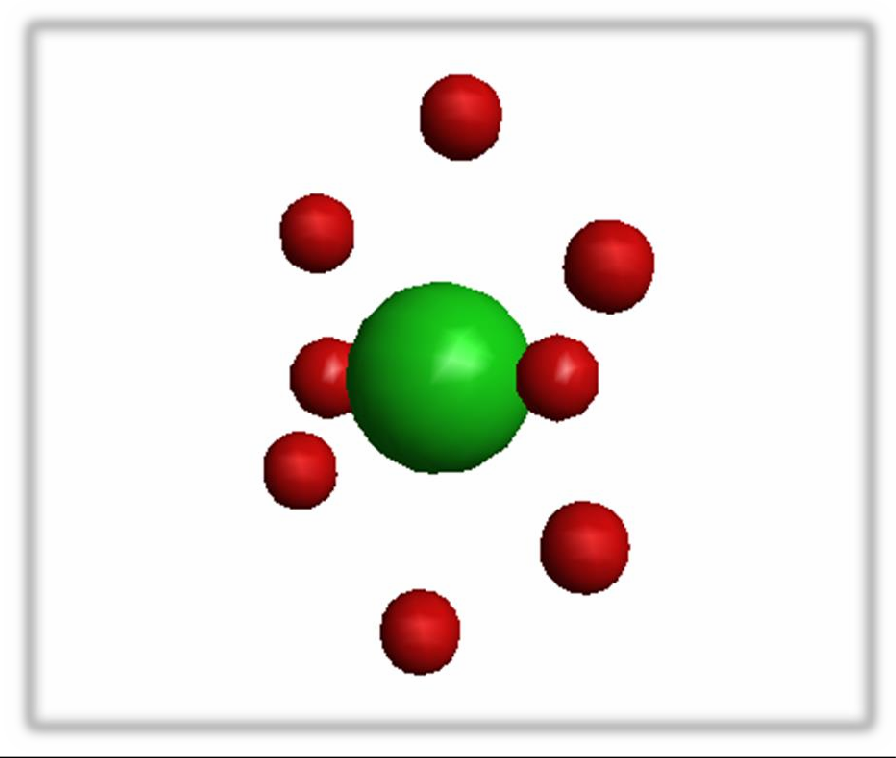
*Fig(5.2)3D Model of O atom*

The given fig(5.3) is 3D model of Oxygen after rotating the model along y-axis in right hand side. The rotation along y-axis can be done using keys [ (for left rotation ),] (for right rotation).



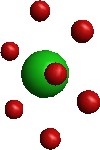
*Fig(5.3)3D Model of O atom after rotation along y-axis*

The given fig(5.4) is 3D model of Oxygen after focusing camera in using key F.



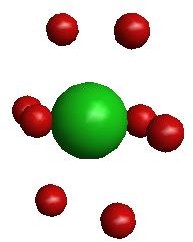
*Fig(5.4) 3D model of O atom after Focussing camera in*

The given fig(5.5) is 3D model of Oxygen after focusing camera out using key V.



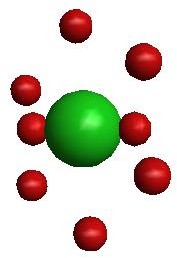
*Fig(5.5) 3D Model of O atom after Focussing camera out*

The given fig(5.6) is 3D model of Oxygen after increasing parallax using ‘+’.



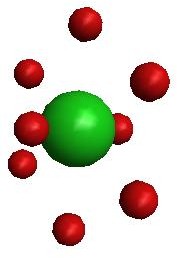
*Fig(5.6) 3D Model of O atom after increasing parallax*

The given fig(5.7) is 3D model of Oxygen after decreasing parallax using key ‘-‘.



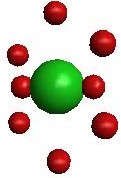
*Fig(5.7) 3D Model of O atom after decreasing parallax*

The given fig(5.8) is 3D model of Oxygen after fine Zoom-in using key A. We can do large incremental Zoom-in using keys Shift+A.



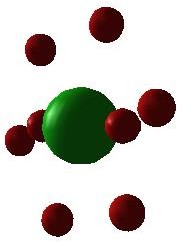
*Fig(5.8) 3D Model of O atom after zoom-in*

The given fig(5.9) is 3D model of Oxygen after fine Zoom-out using key Z. We can do large incremental Zoom-in using keys Shift+Z.



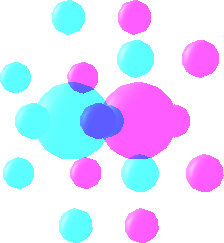
*Fig(5.9) 3D Model of O atom after zoom-out*

The given fig(5.10) is 3D model of an atom in Reset Mode. We can set the figure in Reset Mode using key M.



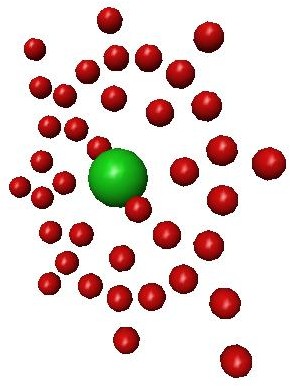
*Fig(5.10) 3D model of atom in Reset mode*

The given fig(5.11) is 3D model of an atom in Stereo Mode. We can display in Fission Mode using key S.



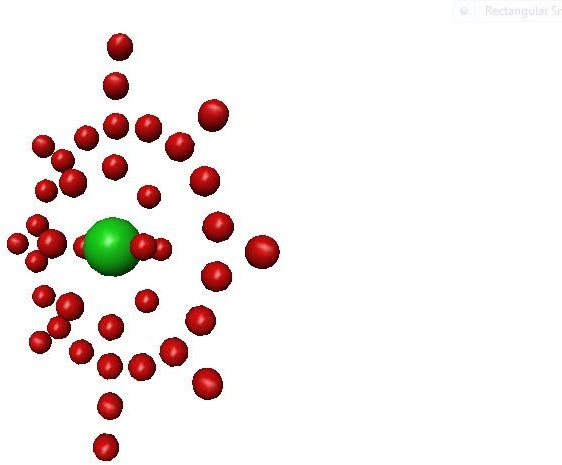
*Fig(5.11) 3D model of atom in Fission mode*

The given fig(5.12) shows 2D atomic model of Strontium(38).



*Fig(5.12) 2D Model of Sr atom*

The given fig(5.13) shows3D atomic model of Strontium(38).



*Fig(5.13) 3D Model of Sr atom*

# Chapter-7

**FUTURE ENHANCEMENT**

The future scope of the project is in the field of simulation of planetary models and particle physics for understanding and observation.

We can place camera at different point of observation to get different views. The rotation along different planes provides us panoramic view of the model. With suitable interface and DBMS tool such as ODBC we can provide a stored data through the file stream. This eliminates the need to store the elements information in user defined structure, which is a lengthy process to store and increases the size of program, thus works slower.

Providing such common data for the program eliminates the necessity of changing the structure in case of alteration of program. The program works faster.

We can add more effects such as Uncertainty Principle in case of atomic model by using more complex scientific theories. Which can be simulated by adding rotation about a referral plane .

# Chapter-8

**CONCLUSION**

Viewing a object projected onto a three dimensional scene is a useful technique for simulating a number efforts, including projecting images, spotlight illumination, and shadows.

The Stereo Mode using Colour Masking and Stencil is useful for generation of Anaglyph image for display. The anaglyph image can be seen using standard 3D glasses as used in old 3D movies.

The perspective view of atom provides a virtual understanding of real physical model of any atom. The 2 –plane rotation provides us front view of the 3D model of atom. Whereas the 3

–plane rotation of the atom provides us with Perspective view of the atomic model.

The described project demonstrates the power of Viewing which is implemented using different modes of viewing. The lighting and material functions of OpenGl library add effect to the objects in animation.

Swap Buffers are implemented to apply a suitable post processing operation to buffer borders to smooth pixel colours towards the buffer edges. This is also useful for giving effect shuddering of object after motion.

# Chapter-9

**REFERENCES**

### Books:

1. The Red Book –OpenGL Programming Guide,6th edition.
2. Rost , Randi J. : OpenGL Shading Language, Addison-Wesley
3. Interactive Computer Graphics-A Top Down Approach Using OpenGL, Edward Angel, Pearson-5th edition.

### Websites:

1. <http://www.opengl.org/documentation>
2. <http://www.nehe.gamedev.net/lesson.asp?index=01>
3. <http://en.wikipedia.org/wiki/OpenGL#Documentation>

# Chapter-10

**APPENDIX**

##### User Manual

**Creating the Project**

**Step 1:** To create an empty console project in CodeBlocks, do the following:

* + 1. Create a new project(FileNewProject)
    2. In the Project Types: pane, select Glut Project, Win32. Then select Win32 Console Application in Templates: pane. Name your project, select the location for the project and click OK.

Click the Application Settings tab on the left, and check the Empty Project box. Then click Finish button.

**Step 2**: Add source code

1. Select Project.
2. In the categories pane, select OpenGL, code. Then select C File(.cpp) in the Templates: pane. Name your file, and then click Add.

**Steps for execution**

1. Compile

From the Visual Studio’s menu Build option(BuildBuild Solution).

1. Execute the program from the Codeblocks’s menu Run option. (Build  Run).

### Controls

Mouse-Right-Click - Load an Element

Mouse-Left-Click - Toggle between 2D and 3D M - Put the display in Mono Mode

S - Put the display in Stereo Mode P - Pause

U- Run

'+' and '-' - Increase and Decrease parallax (adjust the 3D image for your eyes) '[' and ']' - Rotate left and right (around y-axis)

'<' and '>' - Increase and Decrease rotation speed 'a' and 'z' - Fine zoom in and out.

Shift-A and Shift-Z - Larger incremental zoom in and out. 'F' and 'V' - Focus cameras in and out.

##### Personal Details

**NAME:** ABHAY SUBRAMANIAN K

**USN:** 1DT16CS002

**SEMESTER AND SECTION:** VI“A”

**DEPARTMENT:** COMPUTERSCIENCEANDENGINEERING **COLLEGE:** DAYANANDASAGARACADEMYOFTECHNOLOGY

**EMAIL ID:** [**abhaysubramaniank@gmail.com**](mailto:abhaysubramaniank@gmail.com)