

# CHAPTER 1

## INTRODUCTION

### 1.1 Aim

The aim of this project is to develop a 2-D atom simulator, which contains options like selecting the user desired element, simulating the selected element. And also stopping the simulation when user wants. The interface should be user friendly and should use mouse and keyboard interface for the interaction with the user. The main goal is to show the users how an element structure is and how the electrons revolves around the nucleus so that one can easily get the knowledge of atom.

### 1.2 Introduction to OpenGL

OpenGL is an open specification for an applications program interface for defining 2D and 3D objects. The specification is cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. It renders 3D objects to the screen, providing the same set of instructions on different computers and graphics adapters. Thus it allows us to write an application that can create the same effects in any operating system using any OpenGL-adhering graphics adapter.

Computer graphics, a 3-dimensional primitive can be anything from a single point to an n-sided polygon. From the software standpoint, primitives utilize the basic 3-dimensional rasterization algorithms such as Bresenham's line drawing algorithm, polygon scan line fill, texture mapping and so forth. OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels. This is done by a graphics pipeline known as the OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline, or configure how the pipeline processes these primitives.

OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. OpenGL's low-level design requires programmers to have a good

knowledge of the graphics pipeline, but also gives a certain amount of freedom to implement novel rendering algorithms.

## **GLUT**

GLUT, short for OpenGL Utility Toolkit, is a set of support libraries available on every major platform. OpenGL does not directly support any form of windowing, menus, or input. That's where GLUT comes in. It provides basic functionality in all of those areas, while remaining platform independent, so that you can easily move GLUT-based applications from, for example, Windows to UNIX with few, if any, changes.

### **1.3 Project Related Concepts**

The objective is to build an atom simulator which can convince the audience about the structure of an element. The coding is implemented for the atoms from Hydrogen to Neon, that is for 10 elements. In this simulation importance is given on a structure of an element and how the electrons revolve around the nucleus.

The basic requirements of the atom simulator are analyzed to be:

1. User Interface- User should be able to select an element and start the simulation on their own. They can start, stop and change the elements of their choice and after this they can exit the simulation.
2. Element Selection- User can select the element from Hydrogen to Neon for the simulation, that is from atomic number 1 to atomic number 10.
3. Start/ Stop Simulation- User after selecting an element from the mentioned list he/she can start the simulation. As soon as they select start, the electrons around the nucleus starts revolving around the nucleus within their orbit. If they select the stop simulation option, the simulation will be stopped.

## 1.4 INTERFACES

### Mouse interface

- **Select element:**  
When the user clicks the right click button on the mouse, the screen will be prompted with the list of options. First option is to select the user desired element from the list. The list contains elements from Hydrogen to Neon for the simulation, that is from atomic number 1 to atomic number 10.
- **Simulate:**  
After user selecting an element, when he/she clicks on the simulate option, the electrons around the nucleus starts revolving around the nucleus within their orbit.
- **Stop simulation:**  
If a user selects this option, the simulation will be paused.
- **Exit:**  
The program execution will be terminated and the window will be destroyed after selecting this option.

### Keyboard Interface

Three functionalities are implemented using the keyboard function.

- After selecting an element, if a user presses spacebar the simulation will be started.
- After starting the simulation if the user clicks on 'S' key, simulation will be paused.
- If the user clicks on the 'Q' key, program execution will be terminated and the window will be destroyed.

## CHAPTER 2

### REQUIREMENTS SPECIFICATION

Visual Studio 2005 delivers on Microsoft's vision of smart client applications by letting developers quickly create connected applications that deliver the highest quality rich user experiences. This new version lets any size organization create more secure, more manageable, and more reliable applications that take advantage of Windows Vista, windows7, 2007 Office System and the Web. By building these new types of applications, organizations will find it easier than ever to capture and analyze information so that they can make effective business decisions.

#### 2.1 Software Requirements

- An MS-DOS based operating system like Windows 98, Windows 2000 or Windows XP, vista, windows 7 is the platform required to develop the 2D and 3D graphics applications.
- A Visual C/C++ compiler is required for compiling the source code to make the executable file which can then be directly executed.
- A built in graphics library like glut and glut32, and header file like GL\glut.h and also dynamic link libraries like glut and glut32 are required for creating the 3D layout.

#### 2.2 Hardware Requirements

The hardware requirements are very minimal and the software can run on most of the machines.

- Processor - Intel 486/Pentium processor or above.
- Processor Speed - 500 MHz or above
- RAM - 64MB or above Storage Space - 2 MB or above, hard disk -10MB.
- Monitor resolution - A color monitor with a minimum resolution of 1000\*700
- Supports both single &double buffering.

## CHAPTER 3

### DESIGN

#### 3.1 Window design

Atom simulation uses only one window. That is

- **Atom simulation (Main window):** This window contains all the contents that is menu bar and simulation display. This is window used for all the events and functions in this project. In this window we display simulation of first 10 atoms in the periodic table. And all mouse and keyboard events triggered in this window. All the labels and information about the model will be displayed on this window.

#### 3.2 Menu Bar

Menu bar is designed so that one can easily access the various options like selecting the elements or starting the simulation or stopping the simulation etc.

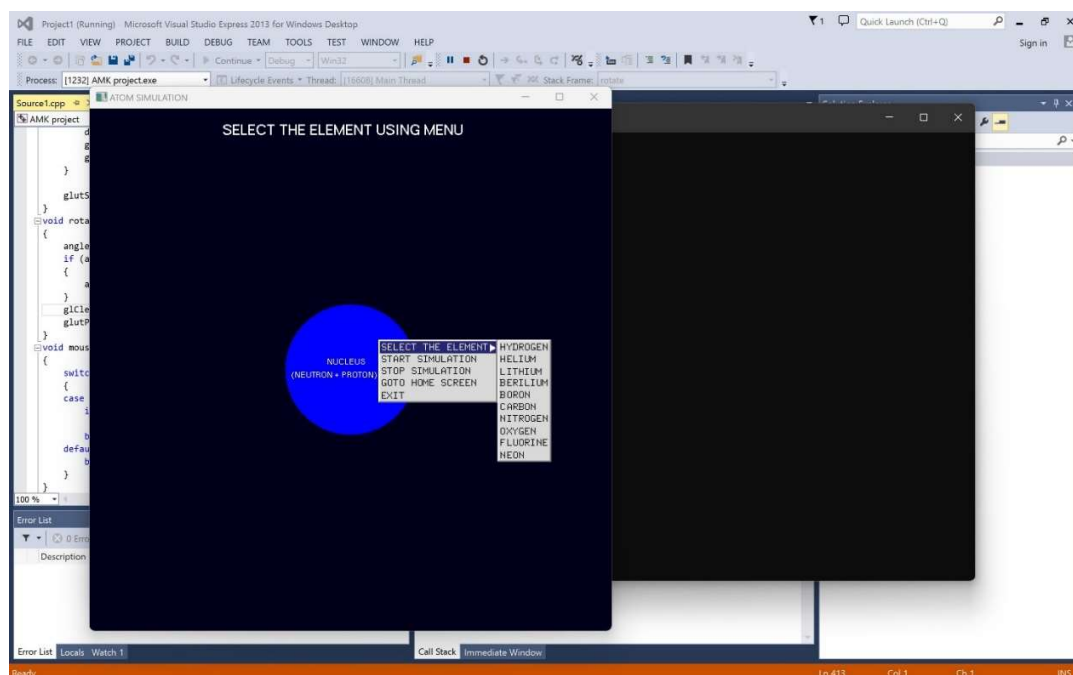


Figure 3.1: Menu Bar

### 3.3 Simulation display

As soon as user selects the element and click on the simulate option, the electrons around the nucleus starts revolving around the nucleus within their orbit.

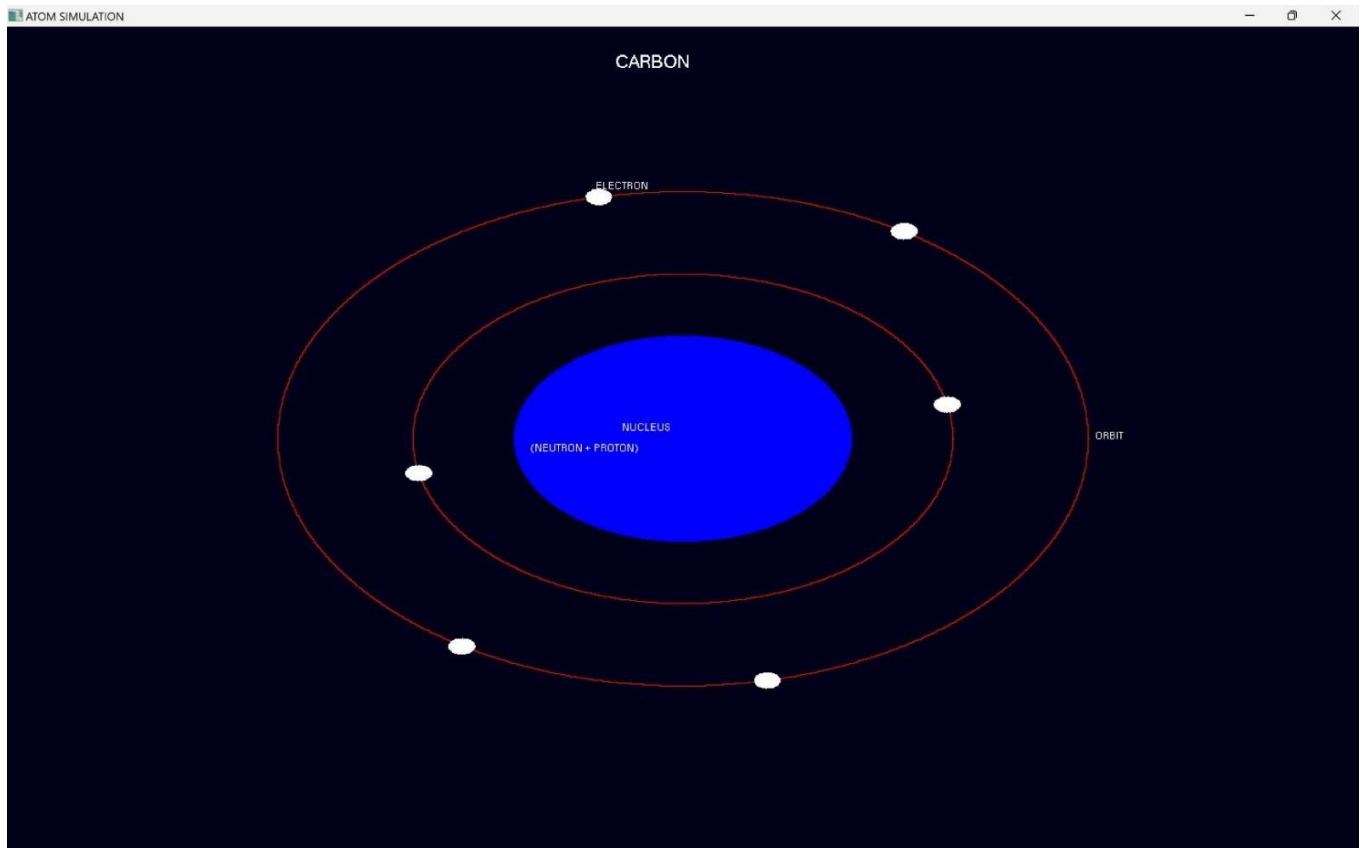


Figure 3.2: Simulation display

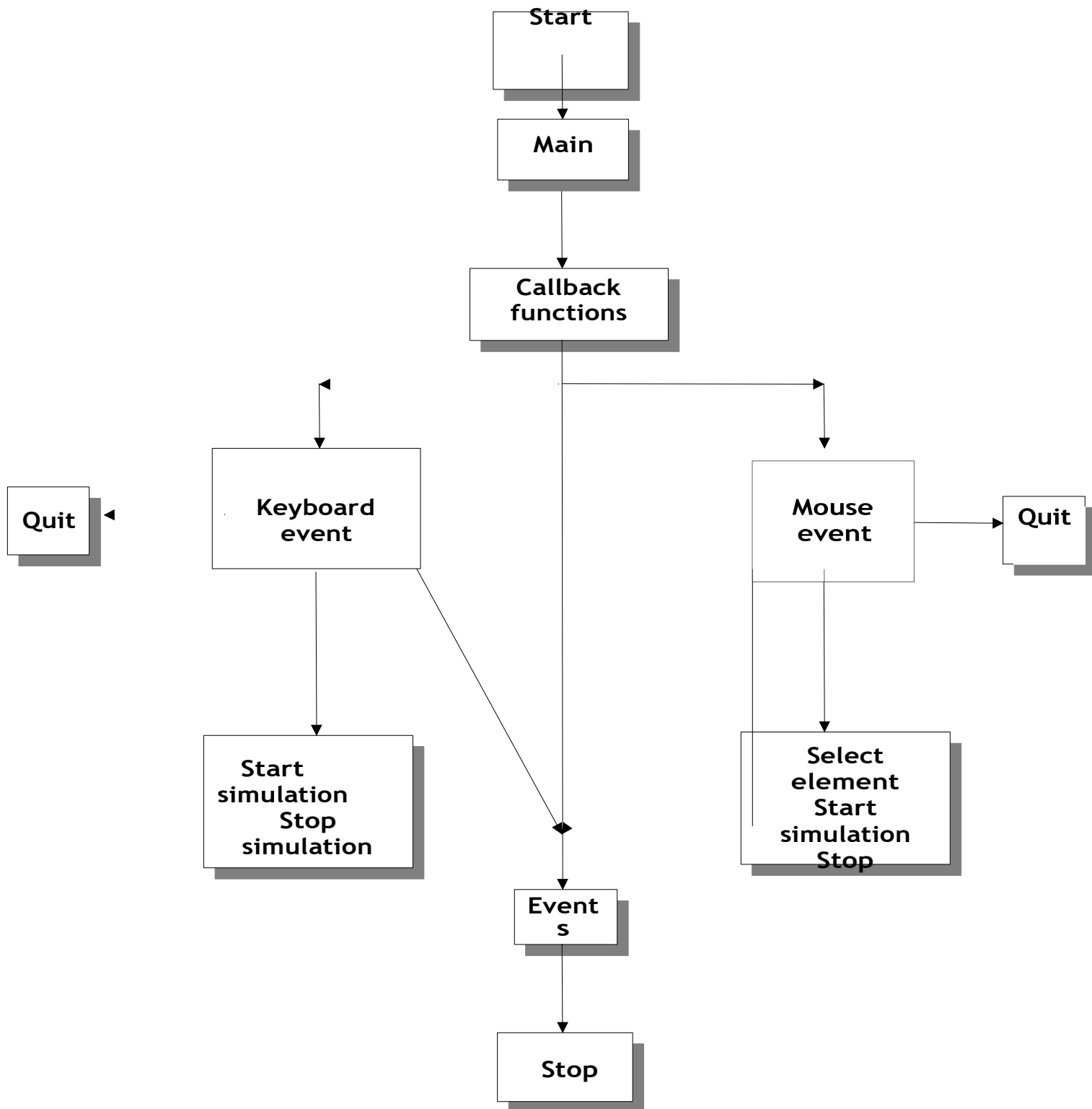


Figure3.3 Flow of control

## CHAPTER 4

## IMPLEMENTATION

### 4.1 Functions used

**glRasterPos3f(x, y, z):** OpenGL maintains a 3-D position in window coordinates. This position, called the raster position, is maintained with subpixel accuracy. The current raster position consists of three window coordinates (x, y, z), a clip coordinate *w* value, an eye coordinate distance, a valid bit, and associated color data and texture coordinates.

**glutCreateMenu(menu):** glutCreateMenu creates a new pop-up menu and returns a unique small integer identifier. The range of allocated identifiers starts at one. The menu identifier range is separate from the window identifier range. Implicitly, the current menu is set to the newly created menu. This menu identifier can be used when calling glutSetMenu.

**glutAddMenuEntry(args):** glutAddMenuEntry adds a menu entry to the bottom of the current menu. The string name will be displayed for the newly added menu entry. If the menu entry is selected by the user, the menu's callback will be called passing value as the callback's parameter.

**glutAttachMenu(button):** glutAttachMenu attaches a mouse button for the current window to the identifier of the current menu; glutDetachMenu detaches an attached mouse button from the current window. By attaching a menu identifier to a button, the named menu will be popped up when the user presses the specified button. button should be one of GLUT\_LEFT\_BUTTON, GLUT\_MIDDLE\_BUTTON, and GLUT\_RIGHT\_BUTTON. Note that the menu is attached to the button by identifier, not by reference.

**glutMouseFunc(args):** glutMouseFunc sets the mouse callback for the current window. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of GLUT\_LEFT\_BUTTON,



GLUT\_MIDDLE\_BUTTON, or GLUT\_RIGHT\_BUTTON. For systems with only two mouse buttons, it may not be possible to generate GLUT\_MIDDLE\_BUTTON callback. For systems with a single mouse button, it may be possible to generate only a GLUT\_LEFT\_BUTTON callback. The state parameter is either GLUT\_UP or GLUT\_DOWN indicating whether the callback was due to a release or press respectively. The x and y callback parameters indicate the window relative coordinates when the mouse button state changed. If a GLUT\_DOWN callback for a specific button is triggered, the program can assume a GLUT\_UP callback for the same button will be generated (assuming the window still has a mouse callback registered) when the mouse button is released even if the mouse has moved outside the window.

**glutKeyboardFunc(args):** glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks. During a keyboard callback, glutGetModifiers may be called to determine the state of modifier keys when the keystroke generating the callback occurred.

## CHAPTER 5

### SOURCE CODE

```
#include <stdio.h>

#include <math.h>

#include <GL/glut.h>

#define pi 3.142

static GLfloat angle = 0;

static int submenu;

static int mainmenu;

static int value = -1;

void init()
{
    gluOrtho2D(-1000, 1000, -1000, 1000);
}

void circle(float rad)
{
    glBegin(GL_POINTS);

    glColor3f(1, 0, 0);

    for (float i = 0; i < (2 * pi); i += 0.00001)
    {
        glVertex2i(rad * cos(i), rad * sin(i));
    }

    glEnd();
}

void drawString(float x, float y, float z, char *string)
{

```

```
    glColor3f(1, 1, 1);

    glRasterPos3f(x, y, z);

    for (char *c = string; *c != '\0'; c++)

    {

        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_10, *c);

    }

}

void drawhead(float x, float y, float z, char *string)

{

    glColor3f(1, 1, 1);

    glRasterPos3f(x, y, z);


    for (char *c = string; *c != '\0'; c++)

    {

        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *c);

    }

}

void drawsubhead(float x, float y, float z, char *string)

{

    glColor3f(1, 1, 1);

    glRasterPos3f(x, y, z);

    for (char *c = string; *c != '\0'; c++)

    {

        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, *c);

    }

}
```

```
void nuc(float rad)
{
    glBegin(GL_POLYGON);
    glColor3f(0, 0, 1);
    for (float i = 0; i < (2 * pi); i = i + 0.00001)
    {
        glVertex2f(rad * cos(i), rad * sin(i));
    }
    glEnd();
}
```

```
void eleright(float rad)
{
    glBegin(GL_POLYGON);
    glColor3f(1, 1, 1);
    for (float i = 0; i < (2 * pi); i += 0.00001)
    {
        glVertex2i(rad + 20 * cos(i), 20 * sin(i));
    }
    glEnd();
}
```

```
void eleleft(float rad)
{
    glBegin(GL_POLYGON);
    glColor3f(1, 1, 1);
    for (float i = 0; i < (2 * pi); i += 0.00001)
    {
```

```
        glVertex2i(-(rad + 20 * cos(i)), 20 * sin(i));
    }

    glEnd();
}

void eletop(float rad)
{
    glBegin(GL_POLYGON);
    glColor3f(1, 1, 1);
    for (float i = 0; i < (2 * pi); i += 0.00001)
    {
        glVertex2i(20 * cos(i), rad + 20 * sin(i));
    }

    glEnd();
}

void eledown(float rad)
{
    glBegin(GL_POLYGON);
    glColor3f(1, 1, 1);
    for (float i = 0; i < (2 * pi); i += 0.00001)
    {
        glVertex2i(20 * cos(i), -(rad + 20 * sin(i)));
    }

    glEnd();
}

void eletr(float rad)
{

```

```
glBegin(GL_POLYGON);
glColor3f(1, 1, 1);
for (float i = 0; i < (2 * pi); i += 0.00001)
{
    glVertex2i(((rad - 175) + 20 * cos(i)), ((rad - 175) + 20 * sin(i)));
}
glEnd();
}

void eletl(float rad)
{
    glBegin(GL_POLYGON);
    glColor3f(1, 1, 1);
    for (float i = 0; i < (2 * pi); i += 0.00001)
    {
        glVertex2i(-(rad - 175) + 20 * cos(i), ((rad - 175) + 20 * sin(i)));
    }
    glEnd();
}

void ededl(float rad)
{
    glBegin(GL_POLYGON);
    glColor3f(1, 1, 1);
    for (float i = 0; i < (2 * pi); i += 0.00001)
    {
        glVertex2i(-(rad - 175) + 20 * cos(i), -((rad - 175) + 20 * sin(i)));
    }
}
```

```
    glEnd();
}

void eledr(float rad)
{
    glBegin(GL_POLYGON);
    glColor3f(1, 1, 1);
    for (float i = 0; i < (2 * pi); i += 0.00001)
    {
        glVertex2i(((rad - 175) + 20 * cos(i)), -((rad - 175) + 20 * sin(i)));
    }
    glEnd();
}

void display()
{
    glClearColor(0, 0, 0.1, 0.9);

    if (value == -1)
    {
        char cn[] = "GHOUSIA COLLEGE OF ENGINEERING";
        drawhead(-490, 900, 0, cn);

        char pn[] = "Ramanagaram, Karnataka-562259";
        drawsubhead(-250, 850, 0, pn);

        char dn[] = "DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING";
        drawhead(-690, 650, 0, dn);

        char prn[] = "A Mini Project On";
        drawsubhead(-150, 450, 0, prn);
    }
}
```

```
char pro[] = "ATOM SIMULATION";  
drawhead(-250, 350, 0, pro);  
char pb[] = "PROJECT BY: ";  
drawhead(-690, -150, 0, pb);  
char p1[] = "SAHANA B R";  
drawhead(-690, -250, 0, p1);  
char p1u[] = "1GC21CS086";  
drawsubhead(-690, -300, 0, p1u);  
char gb[] = "GUIDED BY: ";  
drawhead(290, -150, 0, gb);  
char g1[] = "Mrs MEENAZ  
TABASSUM";  
drawhead(290, -250, 0, g1);  
char d1[] = " Professor, Dept. Of CSE, GCE";  
drawsubhead(290, -300, 0, d1);  
char in[] = "Press enter to Continue";  
drawhead(-250, -700, 0, in);  
glutSwapBuffers();  
glutDetachMenu(GLUT_RIGHT_BUTTON);
```



```
}  
  
if (value != -1)  
{  
    nuc(250);  
    char n[] = "NUCLEUS";  
    drawString(-90, 20, 0, n);  
    char d[] = "(NEUTRON + PROTON)";  
    drawString(-225, -30, 0, d);  
    if (value == 0)  
    {  
        char nu[] = "SELECT THE ELEMENT USING MENU";  
        drawhead(-490, 900, 0, nu);  
    }  
}  
  
if (value == 1)  
{  
    char n[] = "HYDROGEN";  
    drawhead(-100, 900, 0, n);  
    circle(400);  
    char o[] = "ORBIT";  
    drawString(410, 0, 0, o);  
    glPushMatrix();  
    glRotatef(angle, 0, 0, 1);  
    eleright(400);  
    char e[] = "ELECTRON";  
    drawString(420, 0, 0, e);  
}
```

```
        glPopMatrix();
        glutSwapBuffers();
    }
    if (value == 2)
    {
        char n[] = "HELIUM";
        drawhead(-100, 900, 0, n);
        circle(400);
        char o[] = "ORBIT";
        drawString(410, 0, 0, o);
        glPushMatrix();
        glRotatef(angle, 0, 0, 1);
        eleright(400);
        eleleft(400);
        char e[] = "ELECTRON";
        drawString(420, 0, 0, e);
        glPopMatrix();
        glutSwapBuffers();
    }
    if (value == 3)
    {
        char n[] = "LITHIUM";
        drawhead(-100, 900, 0, n);
        circle(400);
        circle(600);
        char o[] = "ORBIT";
```

```
drawString(610, 0, 0, o);

glPushMatrix();

glRotatef(angle, 0, 0, 1);

eleright(400);

eleleft(400);

eletop(600);

char e[] = "ELECTRON";

drawString(0, 620, 0, e);

glPopMatrix();

glutSwapBuffers();
}

if (value == 4)
{
    char n[] = "BERYLLIUM";

    drawhead(-100, 900, 0, n);

    circle(400);

    circle(600);

    char o[] = "ORBIT";

    drawString(610, 0, 0, o);

    glPushMatrix();

    glRotatef(angle, 0, 0, 1);

    eleright(400);

    eleleft(400);

    eletop(600);

    eledown(600);

    char e[] = "ELECTRON";
```

```
    drawString(0, 620, 0, e);

    glPopMatrix();

    glutSwapBuffers();
}

if (value == 5)
{
    char n[] = "BORON";

    drawhead(-100, 900, 0, n);

    circle(400);

    circle(600);

    char o[] = "ORBIT";

    drawString(610, 0, 0, o);

    glPushMatrix();

    glRotatef(angle, 0, 0, 1);

    eleright(400);

    eleleft(400);

    eletop(600);

    eledown(600);

    eletr(600);

    char e[] = "ELECTRON";

    drawString(0, 620, 0, e);

    glPopMatrix();

    glutSwapBuffers();
}

if (value == 6)
{
```

```
char n[] = "CARBON";  
drawhead(-100, 900, 0, n);  
circle(400);  
circle(600);  
char o[] = "ORBIT";  
drawString(610, 0, 0, o);  
glPushMatrix();  
glRotatef(angle, 0, 0, 1);  
eleright(400);  
eleleft(400);  
eletop(600);  
eledown(600);  
eletr(600);  
eledl(600);  
char e[] = "ELECTRON";  
drawString(0, 620, 0, e);  
glPopMatrix();  
glutSwapBuffers();  
}  
if (value == 7)  
{  
    char n[] = "NITROGEN";  
    drawhead(-100, 900, 0, n);  
    circle(400);  
    circle(600);  
    char o[] = "ORBIT";
```

```
drawString(610, 0, 0, o);

glPushMatrix();

glRotatef(angle, 0, 0, 1);

eleright(400);

eleleft(400);

eletop(600);

eledown(600);

eletr(600);

eledl(600);

eletl(600);

char e[] = "ELECTRON";

drawString(0, 620, 0, e);

glPopMatrix();

glutSwapBuffers();

}

if (value == 8)

{

char n[] = "OXYGEN";

drawhead(-100, 900, 0, n);

circle(400);

circle(600);

char o[] = "ORBIT";

drawString(610, 0, 0, o);

glPushMatrix();

glRotatef(angle, 0, 0, 1);

eleright(400);
```

```
    eleleft(400);

    eletop(600);

    eledown(600);

    eletr(600);

    eledl(600);

    eletl(600);

    eledr(600);

    char e[] = "ELECTRON";

    drawString(0, 620, 0, e);

    glPopMatrix();

    glutSwapBuffers();
}

if (value == 9)
{
    char n[] = "FLUORINE";

    drawhead(-100, 900, 0, n);

    circle(400);

    circle(600);

    char o[] = "ORBIT";

    drawString(610, 0, 0, o);

    glPushMatrix();

    glRotatef(angle, 0, 0, 1);

    eleright(400);

    eleleft(400);

    eletop(600);

    eledown(600);
```

```
    eletr(600);
    eledl(600);
    eletl(600);
    eledr(600);
    eleleft(600);
    char e[] = "ELECTRON";
    drawString(0, 620, 0, e);
    glPopMatrix();
    glutSwapBuffers();
}
if (value == 10)
{
    char n[] = "NEON";
    drawhead(-100, 900, 0, n);
    circle(400);
    circle(600);
    char o[] = "ORBIT";
    drawString(610, 0, 0, o);
    glPushMatrix();
    glRotatef(angle, 0, 0, 1);
    eleright(400);
    eleleft(400);
    eletop(600);
    eledown(600);
    eletr(600);
    eledl(600);
```



```
        eletl(600);

        eledr(600);

        eleleft(600);

        eleright(600);

        char e[] = "ELECTRON";

        drawString(0, 620, 0, e);

        glPopMatrix();

        glutSwapBuffers();

    }

    glutSwapBuffers();
}

void rotate()
{
    angle = angle + 1.0;
    if (angle > 360)
    {
        angle = angle - 360;
    }

    glClear(GL_COLOR_BUFFER_BIT);

    glutPostRedisplay();
}

void mouseControl(int button, int state, int x, int y)
{
    switch (button)
    {
```

```
case GLUT_LEFT_BUTTON:
    if (state == GLUT_DOWN)
        glutIdleFunc(rotate);
    break;
default:
    break;
}
}

void keyboard(unsigned char key, int x, int y)
{
    if (key == 13)
    {
        value = 0;
        glClear(GL_COLOR_BUFFER_BIT);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutPostRedisplay();
    }
    else if (key == 's')
    {
        glutIdleFunc(NULL);
    }
    else if (key == 32)
    {
        glutIdleFunc(rotate);
    }
    else if (key == 'q' || key == 'Q')
```

```
{
    exit(0);
}

else if (key == 27)
{
    glutReshapeWindow(700, 700);
}
}

void fkey(int key, int x, int y)
{

    if (key == GLUT_KEY_F10)
    {
        glutReshapeWindow(glutGet(GLUT_SCREEN_WIDTH),
        glutGet(GLUT_SCREEN_HEIGHT));
    }
}

void menu(int option)
{
    if (option == 13)
    {
        exit(0);
    }

    else if (option == 11)
    {
        glutIdleFunc(rotate);
    }
}
```

```
}  
  
else if (option == 12)  
{  
    glutIdleFunc(NULL);  
}  
  
else  
{  
    value = option;  
}  
  
glClear(GL_COLOR_BUFFER_BIT);  
  
glutPostRedisplay();  
}  
  
void createMenu(void)  
{  
    submenu = glutCreateMenu(menu);  
    glutAddMenuEntry("HYDROGEN", 1);  
    glutAddMenuEntry("HELIUM", 2);  
    glutAddMenuEntry("LITHIUM", 3);  
    glutAddMenuEntry("BERILIUM", 4);  
    glutAddMenuEntry("BORON", 5);  
    glutAddMenuEntry("CARBON", 6);  
    glutAddMenuEntry("NITROGEN", 7);  
    glutAddMenuEntry("OXYGEN", 8);  
    glutAddMenuEntry("FLUORINE", 9);  
    glutAddMenuEntry("NEON", 10);
```

```
mainmenu = glutCreateMenu(menu);

glutAddSubMenu("SELECT THE ELEMENT", submenu);

glutAddMenuEntry("SIMULATE", 11);

glutAddMenuEntry("STOP SIMULATION", 12);

glutAddMenuEntry("EXIT", 13);


glutAttachMenu(GLUT_RIGHT_BUTTON);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(700, 700);
    glutCreateWindow("ATOM SIMULATION");
    init();
    glutDisplayFunc(display);
    glutMouseFunc(mouseControl);
    glutKeyboardFunc(keyboard);
    glutSpecialFunc(fkey);
    createMenu();
    glutMainLoop();
    return 0;
}
```

**CHAPTER 6****TESTING****6.1 TEST CASES****Table 6.1 : Test cases for Mouse interface**

<b>Sl. No</b>	<b>Functionality</b>	<b>Comments</b>	<b>Remarks</b>
1.	Mouse right click	It shows menu bar to user.	Pass
2.	Selecting the options 1. Select element 2. Simulate 3. Stop simulation 4. Exit 5. Goto Home	It shows the list of option to user from which they can select an option. Selects the element in the given list. Starts the simulation. Stops the simulation. Exits from the window. Display starting window	Pass

**Table 6.2 : Test cases for Keyboard interface**

<b>Sl. No</b>	<b>Functionality</b>	<b>Comments</b>	<b>Remarks</b>
1.	Choosing the options 1. Simulate 2. Stop simulation 3. Exit	It shows the list of option to user from which they can select an option. Starts the simulation. ( Space bar ) Stops the simulation. ('S') Exits from the window. ('Q')	Pass

## CHAPTER 8

### SNAPSHOTS

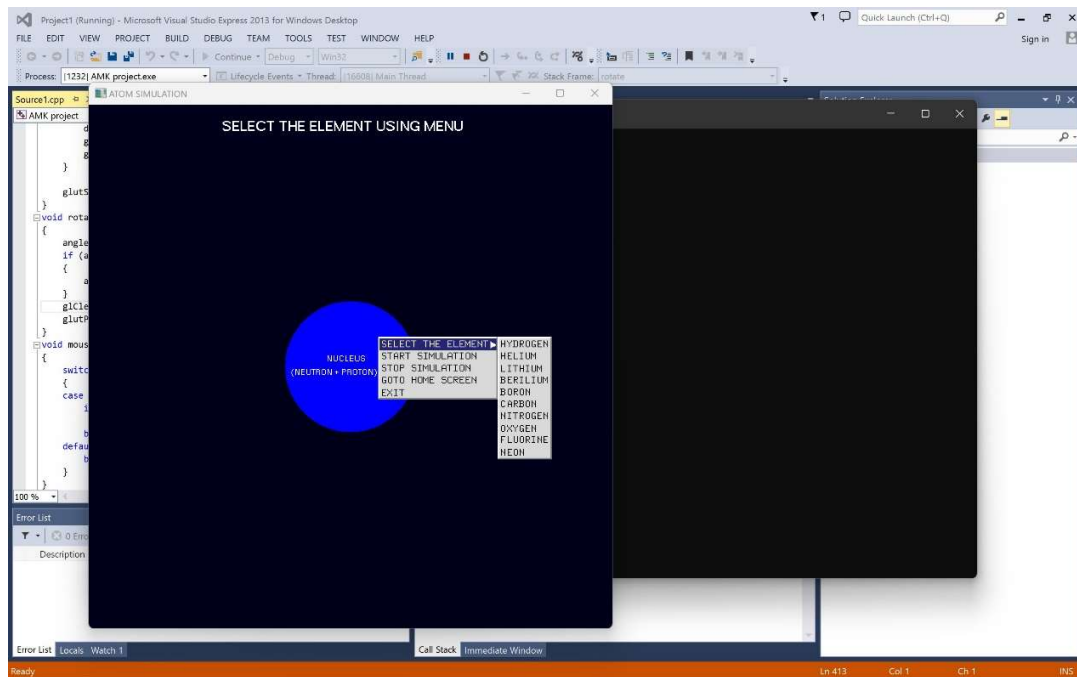


Figure 8.1: Snapshot of Starting Screen

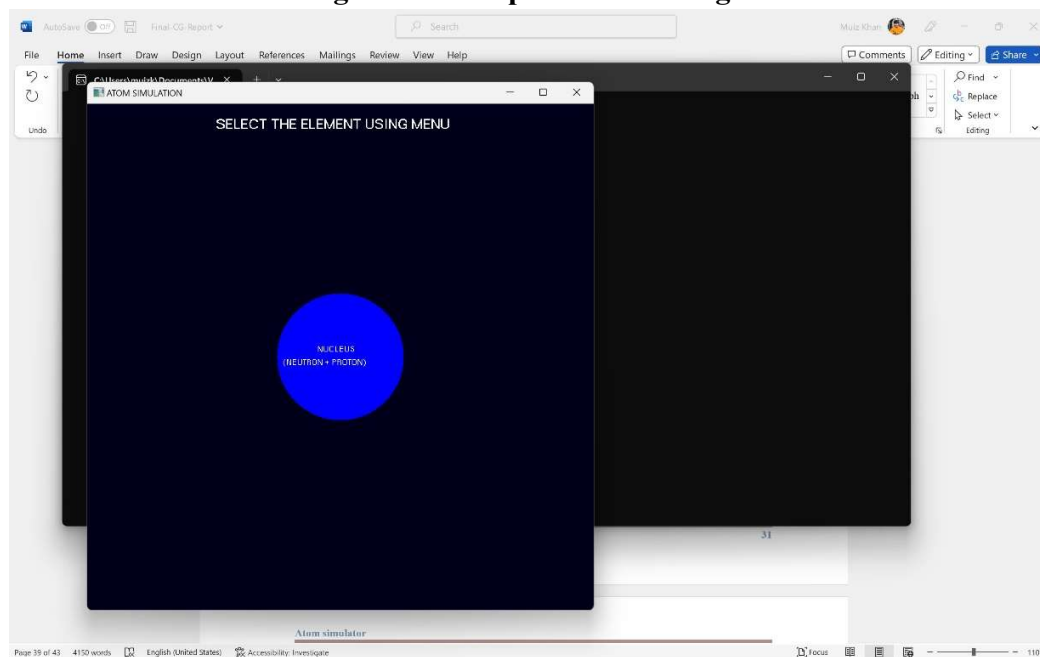
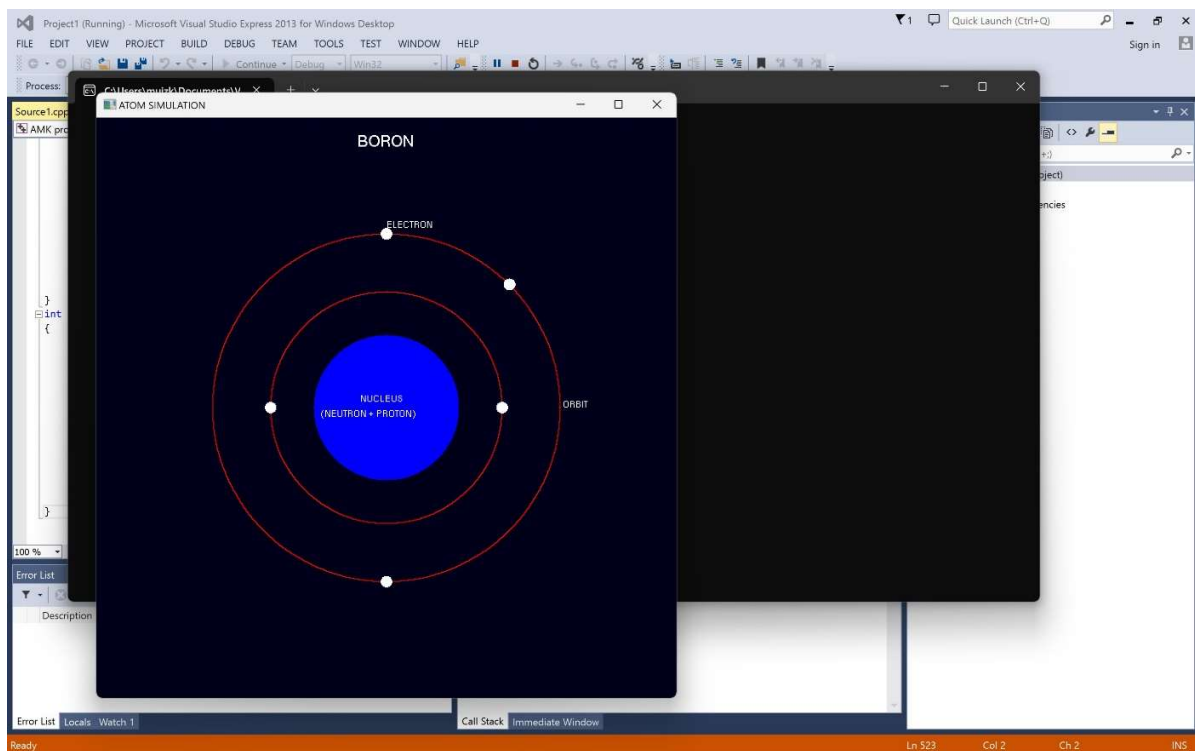
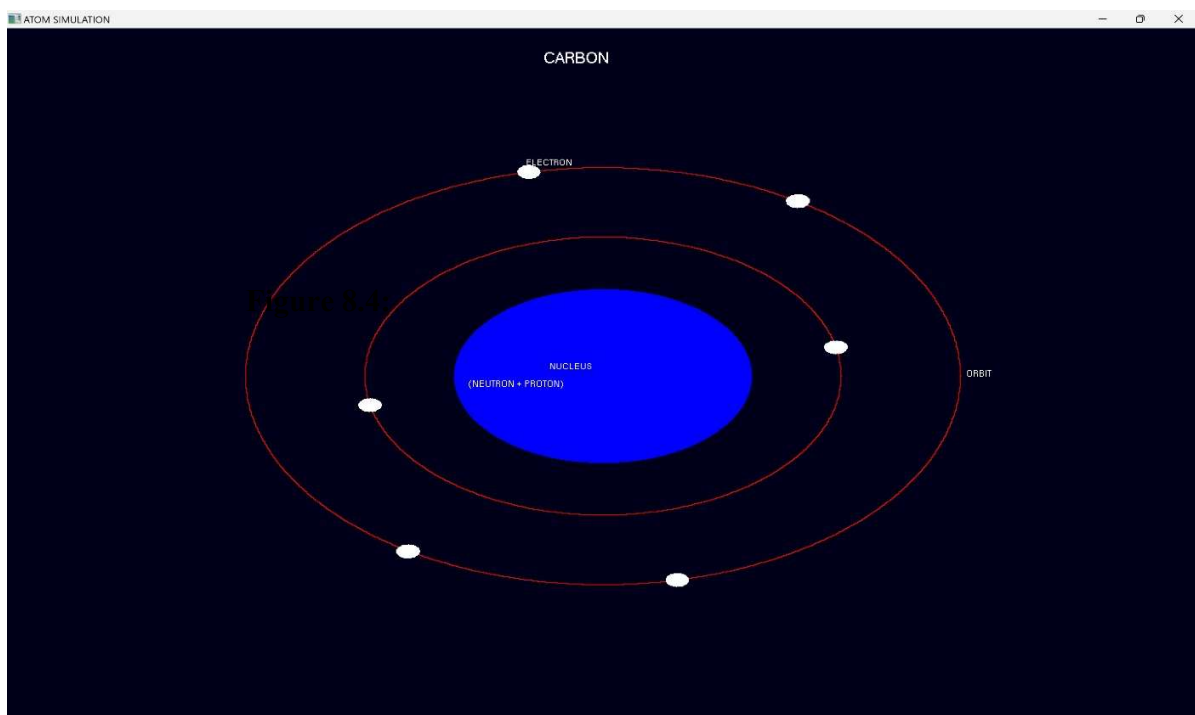


Figure 8.2: Snapshot of Menu Interface



**Figure 8.3: Snapshot of Atom simulation**



**Figure 8.4: Snapshot of Full Screen Atom simulation**



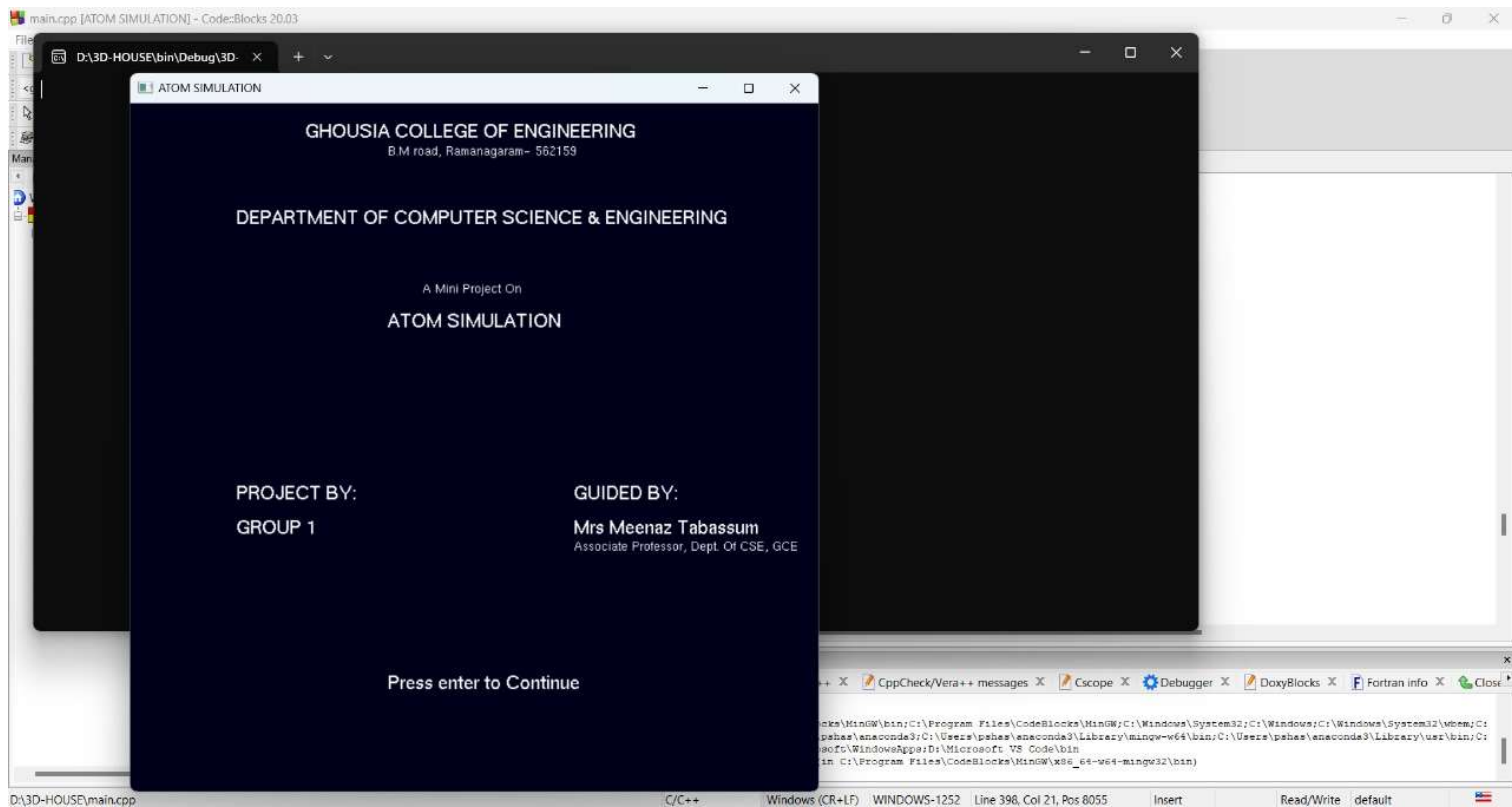


Figure 8.5: Snapshot of Home Screen

## CHAPTER 8

### CONCLUSION

This atom simulation is very good project. Users can very easily understand the structure of an element. The interface is mouse driven and the user can select a function by clicking. And also, the interface supports keyboard interface. We have tried our best to make this simulator very realistic, so that user can easily understand the concepts of electrons, orbits, atoms and nucleus etc.

### FUTURE ENHANCEMENTS

The following are some of the features that are planned to be supported in the future versions of the atom simulator.

- Adding all the elements from the periodic table.
- Features like showing the simulation with all the important details of an element.
- Adding a search bar for selecting an element from the list of all the elements.
- Making the simulation in 3-D.

## REFERENCES

- [1] Interactive Computer Graphics A Top-Down Approach with OpenGL -Edward Angel, 5th Edition, Addison-Wesley, 2008.
- [2] Computer Graphics Using OpenGL – F.S. Hill Jr. 2nd Edition, Pearson Education, 2001.
- [3] Computer Graphics – James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes, Addison-Wesley 1997.
- [4] Computer Graphics - OpenGL Version – Donald Hearn and
- [5] Pauline Baker, 2nd Edition, Pearson Education, 2003.
- [6] [www.google.com](http://www.google.com)
- [7] [www.openglforum.org](http://www.openglforum.org)