CHAPTER - 1

INTRODUCTION

1.1 Aim

The aim of this atom simulation project in the computer graphics is to develop an interactive and visually engaging 2D representation of atoms revolving around a nucleus in a real-time environment using the open GL library.

The system will incorporate a user-friendly menu that allows for the selection of different atoms, and it will provide essential information such as atomic number, atomic mass and other related data. The project aims to create an educational tool that promotes an intuitive understanding of atomic structure, behavior, and chemical properties.

1.2 Overview

This project aims to develop an atom simulation system within the context of a computer graphics. The focus is on creating a visually appealing and interactive 2D representation of atoms, such as oxygen and hydrogen, revolving around a nucleus in a real-time environment. The system also provides essential information about the atoms, including atomic number, atomic mass, oxidation state, density, boiling point, and melting point.

The simulation system incorporates computer graphics techniques to render the atoms and their movements. The 2D representation of atoms rotating around the nucleus provides an intuitive visualization of atomic structure and behavior using the OpenGL libraries of some APIs of GL functions.

In addition to the visual representation, the system offers essential atomic information. Each atom is accompanied by its corresponding atomic number, atomic mass, oxidation state, density, boiling point, and melting point. This data is dynamically updated as the user interacts with the simulation, providing real-time information about the selected atoms. The system promotes an intuitive understanding of atomic structure and behavior, facilitating the exploration of chemical elements and their characteristics.

1.3 Outcome

The outcome is a fully functional and visually engaging 2D atom simulation system. It features interactive atom representation, a user-friendly menu for changing atom types, real-time information display, and smooth rendering. It serves as an educational tool for exploring atomic structures and properties in an interactive and visually captivating manner.

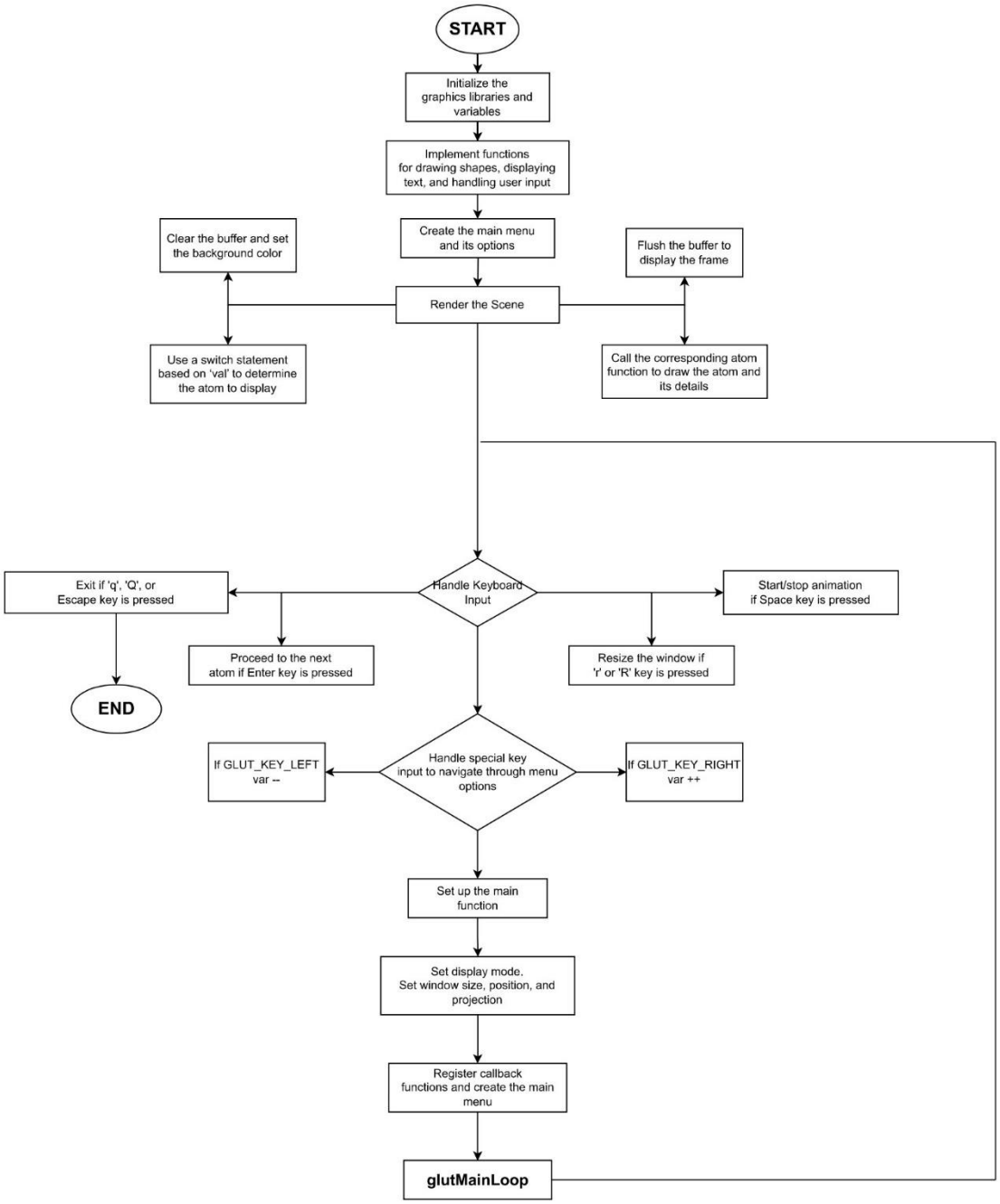
CHAPTER – 2

DESIGN AND IMPLEMENTATION

2.1 Algorithm

1. Initialize the graphics libraries and variables.
2. Implement functions for drawing shapes, displaying text, and handling user input.
3. Create the main menu and its options.
4. Render the scene:
    - Clear the buffer and set the background color.
    - Use a switch statement based on ‘val’ to determine the atom to display.
    - Call the corresponding atom function to draw the atom and its details.
    - Flush the buffer to display the frame.
5. Handle keyboard input:
    - Exit if 'q', 'Q', or Escape key is pressed.
    - Proceed to the next atom if Enter key is pressed.
    - Go back to the home screen if 'h' or 'H' key is pressed.
    - Start/stop animation if Space key is pressed.
    - Resize the window if 'r' or 'R' key is pressed.
6. Handle special key input to navigate through menu options.
7. Initialize and attach the menu to the right mouse button.
8. Set up the main function:
    - Initialize libraries, create the window, and set display mode.
    - Set window size, position, and projection.
    - Register callback functions and create the main menu.
    - Enter the main event loop using 'glutMainLoop’.

2.2 Flow Chart



2.3 OpenGL API's Used with Description

glClearColor: This API sets the color used to clear the color buffer. It takes red, green, blue, and alpha values as parameters to specify the desired color.

glClear: This API clears the buffers specified by the mask parameter. In the code, glClear(GL_COLOR_BUFFER_BIT) is used to clear the color buffer.

glFlush: This API ensures that all commands queued in the buffer are executed immediately, rather than waiting for them to be processed in the background.

gluOrtho2D: This API sets up a two-dimensional orthographic viewing region for the OpenGL window. It defines the coordinate system used for rendering.

glBegin and glEnd: These APIs delimit the vertices of geometric primitives to be drawn. They mark the beginning and end of a sequence of vertices that define a primitive shape.

glVertex2f: This API specifies a 2D vertex with floating-point coordinates. It is used to define the position of points, lines, and other geometric primitives.

glColor3f: This API sets the current color for drawing operations. It takes red, green, and blue values as parameters, each ranging from 0.0 to 1.0, to specify the desired color.

glRasterPos2i: This API sets the raster position for bitmap and pixel operations. It takes the x and y coordinates of the raster position as integer values.

glutBitmapCharacter: This API renders a bitmap character at the current raster position. It takes a font identifier (such as GLUT_BITMAP_HELVETICA_18 or GLUT_BITMAP_TIMES_ROMAN_24) and a character as parameters.

glutInit, glutInitDisplayMode, glutInitWindowSize, glutInitWindowPosition: These APIs initialize the GLUT library and set various properties of the window, such as display mode, size, and position.

glutCreateWindow: This API creates a top-level window and associates it with the current OpenGL context.

glutDisplayFunc: These APIs register callback functions to handle display.

glutKeyboardFunc: These APIs register callback functions to handle keyboard.

glutSpecialFunc: These APIs register callback functions to handle special key events.

glutMainLoop: This API enters the GLUT event processing loop, which continuously processes events and dispatches them to the appropriate callback functions.

2.4 Source Code

Functions.h

```
#include <Windows.h>
#include <gl/glut.h>
#include <cmath>


float angle = 0;


// Function to draw a filled circle
void FilledCircle(float cenX, float cenY, float rad)
{
   glColor3f(1.0f, 0.0f, 0.0f);   // Red

   glBegin(GL_TRIANGLE_FAN);
      glVertex2f(cenX, cenY);

      for (int i = 0; i <= 360; i++)
      {
         float theta = 2.0f * 3.14159f * float(i) / 360;
         float x = cenX + rad * cos(theta);
         float y = cenY + rad * sin(theta);

         glVertex2f(x, y);
      }
   glEnd();
}

// Function to draw a circle
void Circle(float cenX, float cenY, float rad)
{
   glColor3f(0.0f, 0.0f, 1.0f);   //Blue
```

```
    glBegin(GL_LINE_LOOP);
        for (int i = 0; i <= 360; i++)
        {
            float theta = 2.0f * 3.14159f * float(i) / 360;
            float x = cenX + rad * cos(theta);
            float y = cenY + rad * sin(theta);

            glVertex2f(x, y);
        }
    glEnd();
}

// Function to draw a filled circle

void rotatingCircle(float radX, float radY, float Rad, float ang)
{
    glColor3f(0.0f, 1.0f, 0.0f);  // Green

    // Calculate the position of the first rotating circle
    float centerX = radX + Rad * cos(angle + ang);
    float centerY = radY + Rad * sin(angle + ang);

    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(centerX, centerY);
    for (int i = 0; i <= 360; i++)
    {
        float theta = 2.0f * 3.14159f * float(i) / 360;
        float x = centerX + 10.0 * cos(theta);
        float y = centerY + 10.0 * sin(theta);
        glVertex2f(x, y);
    }
    glEnd();
}



// Spinning Angle

void Spin()
{
    angle = angle + 0.005;
    if(angle > 360)
        angle = 0;

    Sleep(1);
    glutPostRedisplay();    // Mark the window for redrawing
}
```

Details.h

```
#include <windows.h>
#include <GL/glut.h>
#include <cmath>
#include <stdio.h>
#include <string.h>


const char* atNo[] = {"", "Atomic Number: 01",
                      "Atomic Number: 02",
                      "Atomic Number: 03",
                      "Atomic Number: 04",
                      "Atomic Number: 05",
                      "Atomic Number: 06",
                      "Atomic Number: 07",
                      "Atomic Number: 08",
                      "Atomic Number: 09",
                      "Atomic Number: 10"};


const char* atMass[] = {"", "Atomic Mass: 1.0080 u",
                      "Atomic Mass: 4.0026 u",
                      "Atomic Mass: 7.00 u",
                      "Atomic Mass: 9.0122 u",
                      "Atomic Mass: 10.81 u",
                      "Atomic Mass: 12.011 u",
                      "Atomic Mass: 14.007",
                      "Atomic Mass: 15.999 u",
                      "Atomic Mass: 18.9984 u",
                      "Atomic Mass: 20.180 u"};


const char* oxState[] = {"", "Oxidation State: +1, -1",
                      "Oxidation State: 0",
                      "Oxidation State: +1",
                      "Oxidation State: +2",
                      "Oxidation State: +3",
                      "Oxidation State: +4, -2 ,-4",
                      "Oxidation State: +5,+4,+3,+2,+1,-1,-2,-3",
                      "Oxidation State: -2",
                      "Oxidation State: -1",
                      "Oxidation State: 0"};
```

```
const char* CGB[] = {"", "Chemical Group Block: Non-metal",
                     "Chemical Group Block: Noble Gas",
                     "Chemical Group Block: Alkali Metal",
                     "Chemical Group Block: Alkaline Earth Metal",
                     "Chemical Group Block: Metalloid",
                     "Chemical Group Block: Non-metal",
                     "Chemical Group Block: Non-metal",
                     "Chemical Group Block: Non-metal",
                     "Chemical Group Block: Halogen",
                     "Chemical Group Block: Noble Gas"};


const char* atRad[] = {"", "Atomic Radius: 120 pm",
                       "Atomic Radius: 140 pm",
                       "Atomic Radius: 182 pm",
                       "Atomic Radius: 153 pm",
                       "Atomic Radius: 192 pm",
                       "Atomic Radius: 170 pm",
                       "Atomic Radius: 155 pm",
                       "Atomic Radius: 152 pm",
                       "Atomic Radius: 135 pm",
                       "Atomic Radius: 154 pm"};


const char* ionEg[] = {"", "Ionization Energy: 13.598 eV",
                       "Ionization Energy: 24.587 eV",
                       "Ionization Energy: 5.392 eV",
                       "Ionization Energy: 9.323 eV",
                       "Ionization Energy: 8.298 eV",
                       "Ionization Energy: 11.260 eV",
                       "Ionization Energy: 14.534 eV",
                       "Ionization Energy: 13.618 eV",
                       "Ionization Energy: 17.423 eV",
                       "Ionization Energy: 21.565 eV"};


const char* mp[] = {"", "Melting Point: 13.81 K",
                    "Melting Point: 0.97 K",
                    "Melting Point: 453.65 K",
                    "Melting Point: 1560 K",
                    "Melting Point: 2348 K",
                    "Melting Point: 3823 K",
                    "Melting Point: 63.15 K",
                    "Melting Point: 54.36 K",
                    "Melting Point: 53.53 K",
                    "Melting Point: 24.06 K"};
```

```
const char* bp[] = {"", "Boiling Point: 20.28 K",
                    "Boiling Point: 4.22 K",
                    "Boiling Point: 1615 K",
                    "Boiling Point: 2744 K",
                    "Boiling Point: 4273 K",
                    "Boiling Point: 4098 K",
                    "Boiling Point: 77.36 K",
                    "Boiling Point: 90.20 K",
                    "Boiling Point: 85.03 K",
                    "Boiling Point: 27.07 K"};


const char* den[] = {"", "Density: 0.08988 mg/cm^3",
                     "Density: 0.1785 mg/cm^3",
                     "Density: 0.534 g/cm^3",
                     "Density: 1.85 g/cm^3",
                     "Density: 2.37 g/cm^3",
                     "Density: 2.267 g/cm^3",
                     "Density: 1.2506 mg/cm^3",
                     "Density: 1.429 mg/cm^3",
                     "Density: 1.696 mg/cm^3",
                     "Density: 0.899 mg/cm^3"};



void normalText(int posX, int posY, const char* text)
{
   glColor3f(0.0f, 0.0f, 0.0f);  // Black

   glRasterPos2i(posX, posY);
   for (int i = 0; text[i] != '\0'; i++)
   {
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);
   }
}


void headText(int posX, int posY, const char* text)
{
   glColor3f(0.0f, 0.0f, 0.0f);  // Black

   glRasterPos2i(posX, posY);
   for (int i = 0; text[i] != '\0'; i++)
   {
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, text[i]);
   }
}
```

```
void identity()
{
   glColor3f(1.0f, 0.0f, 0.0f);    //red
   glBegin(GL_TRIANGLE_FAN);
      for (int i = 0; i <= 360; i++)
      {
         float theta = 2.0f * 3.14159f * float(i) / 360;
         float x = 750 + 8 * cos(theta);
         float y = 200 + 8 * sin(theta);

         glVertex2f(x, y);
      }
   glEnd();

   normalText(770, 200, "- Nucleus(Protons + Neutrons)");


   glColor3f(0.0f, 1.0f, 0.0f);    //green
   glBegin(GL_TRIANGLE_FAN);
      for (int i = 0; i <= 360; i++)
      {
         float theta = 2.0f * 3.14159f * float(i) / 360;
         float x = 750 + 8 * cos(theta);
         float y = 160 + 8 * sin(theta);

         glVertex2f(x, y);
      }
   glEnd();

   normalText(770, 160, "- Electron");


   glColor3f(0.0f, 0.0f, 1.0f);    //green
   glBegin(GL_LINE_LOOP);
      for (int i = 0; i <= 360; i++)
      {
         float theta = 2.0f * 3.14159f * float(i) / 360;
         float x = 750 + 8 * cos(theta);
         float y = 120 + 8 * sin(theta);

         glVertex2f(x, y);
      }
   glEnd();

   normalText(770, 120, "- Orbit");
}
```

```
void arrowKey()
{
   glColor3f(1,0.5,0);    //Orange

   glBegin(GL_POLYGON);
      glVertex2f(150, 670);
      glVertex2f(180, 680);
      glVertex2f(180, 660);
   glEnd();

   glBegin(GL_POLYGON);
      glVertex2f(1000, 670);
      glVertex2f(970, 680);
      glVertex2f(970, 660);
   glEnd();
}


void detailText(int i)
{
   normalText(750.0f, 550.0f, atNo[i]);
   normalText(750.0f, 515.0f, atMass[i]);
   normalText(750.0f, 480.0f, oxState[i]);
   normalText(750.0f, 445.0f, CGB[i]);
   normalText(750.0f, 410.0f, atRad[i]);
   normalText(750.0f, 375.0f, ionEg[i]);
   normalText(750.0f, 340.0f, mp[i]);
   normalText(750.0f, 305.0f, bp[i]);
   normalText(750.0f, 270.0f, den[i]);

   identity();
   arrowKey();

}
```

Models.h

```
#include <windows.h>
#include <GL/glut.h>
#include "Functions.h"
#include "Details.h"

float pi = 3.1428;

void home()
{
   headText(380, 600, "Maharaja Institute of Technology, Mysore");
   headText(330, 550, "Department of Computer Science and Engineering");

   normalText(450, 450, "Computer Graphics Mini Project on ");
   headText(520, 410, "Atom Simulator");

   headText(530,300, "Project By: ");
   headText(180, 250, "Skanda S Koushik - 4MH20CS092");
   headText(650, 250, "Sriram Kashyap - 4MH20CS094");

   headText(470, 150, "Press Enter to Continue");
   headText(435, 100, "Press Space to Start Simulation");
}

void hydrogen()
{
   headText(500, 660, "HYDROGEN : H");

   FilledCircle(350.0f, 325.0f, 50.0f);

   Circle(350.0f, 325.0f, 175.0f);

   rotatingCircle(350.0f, 325.0f, 175.0f, 0);

   detailText(1);
}


void helium()
{
   headText(500, 660, "HELIUM : He");

   FilledCircle(350.0f, 325.0f, 50.0f);

   Circle(350.0f, 325.0f, 175.0f);
```

```
    rotatingCircle(350.0f, 325.0f, 175.0f, 0);
    rotatingCircle(350.0f, 325.0f, 175.0f, pi);

    detailText(2);
}

void lithium()
{
    headText(500, 660, "LITHIUM : Li");

    FilledCircle(350.0f, 325.0f, 50.0f);

    Circle(350.0f, 325.0f, 125.0f);
    Circle(350.0f, 325.0f, 250.0f);

    rotatingCircle(350.0f, 325.0f, 125.0f, 0);
    rotatingCircle(350.0f, 325.0f, 125.0f, pi);

    rotatingCircle(350.0f, 325.0f, 250.0f, pi/2);

    detailText(3);


}


void beryllium()
{
    headText(500, 660, "BERYLLIUM : Be");

    FilledCircle(350.0f, 325.0f, 50.0f);

    Circle(350.0f, 325.0f, 125.0f);
    Circle(350.0f, 325.0f, 250.0f);

    rotatingCircle(350.0f, 325.0f, 125.0f, 0);
    rotatingCircle(350.0f, 325.0f, 125.0f, pi);

    rotatingCircle(350.0f, 325.0f, 250.0f, pi/2);
    rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/2);

    detailText(4);
}
```

```
void boron()
{
   headText(500, 660, "BORON : B");

   FilledCircle(350.0f, 325.0f, 50.0f);

   Circle(350.0f, 325.0f, 125.0f);
   Circle(350.0f, 325.0f, 250.0f);

   rotatingCircle(350.0f, 325.0f, 125.0f, 0);
   rotatingCircle(350.0f, 325.0f, 125.0f, pi);

   rotatingCircle(350.0f, 325.0f, 250.0f, pi/4);
   rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/4);
   rotatingCircle(350.0f, 325.0f, 250.0f, 7*pi/4);

   detailText(5);
}


void carbon()
{
   headText(500, 660, "CARBON : C");

   FilledCircle(350.0f, 325.0f, 50.0f);

   Circle(350.0f, 325.0f, 125.0f);
   Circle(350.0f, 325.0f, 250.0f);

   rotatingCircle(350.0f, 325.0f, 125.0f, 0);
   rotatingCircle(350.0f, 325.0f, 125.0f, pi);

   rotatingCircle(350.0f, 325.0f, 250.0f, pi/4);
   rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/4);
   rotatingCircle(350.0f, 325.0f, 250.0f, 5*pi/4);
   rotatingCircle(350.0f, 325.0f, 250.0f, 7*pi/4);

   detailText(6);
}


void nitrogen()
{
   headText(500, 660, "NITROGEN : N");
   FilledCircle(350.0f, 325.0f, 50.0f);

   Circle(350.0f, 325.0f, 125.0f);
```

```
    Circle(350.0f, 325.0f, 250.0f);

    rotatingCircle(350.0f, 325.0f, 125.0f, 0);
    rotatingCircle(350.0f, 325.0f, 125.0f, pi);

    rotatingCircle(350.0f, 325.0f, 250.0f, pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, pi/2);
    rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, 5*pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, 7*pi/4);

    detailText(7);
}


void oxygen()
{
    headText(500, 660, "OXYGEN : O");

    FilledCircle(350.0f, 325.0f, 50.0f);

    Circle(350.0f, 325.0f, 125.0f);
    Circle(350.0f, 325.0f, 250.0f);

    rotatingCircle(350.0f, 325.0f, 125.0f, 0);
    rotatingCircle(350.0f, 325.0f, 125.0f, pi);

    rotatingCircle(350.0f, 325.0f, 250.0f, pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, pi/2);
    rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, 5*pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/2);
    rotatingCircle(350.0f, 325.0f, 250.0f, 7*pi/4);

    detailText(8);
}


void fluorine()
{
    headText(500, 660, "FLUORINE : F");

    FilledCircle(350.0f, 325.0f, 50.0f);

    Circle(350.0f, 325.0f, 125.0f);
    Circle(350.0f, 325.0f, 250.0f);

    rotatingCircle(350.0f, 325.0f, 125.0f, 0);
```

```
    rotatingCircle(350.0f, 325.0f, 125.0f, pi);

    rotatingCircle(350.0f, 325.0f, 250.0f, pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, pi/2);
    rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, pi);
    rotatingCircle(350.0f, 325.0f, 250.0f, 5*pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/2);
    rotatingCircle(350.0f, 325.0f, 250.0f, 7*pi/4);

    detailText(9);
}


void neon()
{
    headText(500, 660, "NEON : Ne");

    FilledCircle(350.0f, 325.0f, 50.0f);

    Circle(350.0f, 325.0f, 125.0f);
    Circle(350.0f, 325.0f, 250.0f);

    rotatingCircle(350.0f, 325.0f, 125.0f, 0);
    rotatingCircle(350.0f, 325.0f, 125.0f, pi);

    rotatingCircle(350.0f, 325.0f, 250.0f, pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, pi/2);
    rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, pi);
    rotatingCircle(350.0f, 325.0f, 250.0f, 5*pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, 3*pi/2);
    rotatingCircle(350.0f, 325.0f, 250.0f, 7*pi/4);
    rotatingCircle(350.0f, 325.0f, 250.0f, 0);

    detailText(10);
}
```

Main.cpp

```
#include <windows.h>
#include <GL/glut.h>
#include "Models.h"


int val = 0;
int mainmenu;
int motion = 0;

void Draw()
{
   glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
   glClear(GL_COLOR_BUFFER_BIT);

   switch(val)
   {
      case 0: home(); break;
      case 1: hydrogen(); break;
      case 2: helium(); break;
      case 3: lithium(); break;
      case 4: beryllium(); break;
      case 5: boron(); break;
      case 6: carbon(); break;
      case 7: nitrogen(); break;
      case 8: oxygen(); break;
      case 9: fluorine(); break;
      case 10: neon(); break;
      case 11: exit(0);
   }

   glFlush();
}


void keyboard(unsigned char key, int x, int y)
{
   if(key == 'q' || key == 'Q' || key == 27)   // Escape
      exit(0);

   if(key == 13)   // Enter
   {
      if(val == 0)
         val = 1;
   }
```

```
    if(key == 'h' || key == 'H')
        val = 0;

    if(key == 32)   // Space
    {
        if(motion == 0)
        {
            glutIdleFunc(Spin);
            motion = 1;
        }
        else
        {
            glutIdleFunc(NULL);
            motion = 0;
        }
    }

    if(key == 'r' || key == 'R')
        glutReshapeWindow(1150, 700);

    glutPostRedisplay();
}


void specialKey(int key, int x, int y)
{
    switch(key)
    {
    case GLUT_KEY_RIGHT:
        val++;
        if(val > 10)
            val = 10;
        break;

    case GLUT_KEY_LEFT:
        val--;
        if(val < 1)
            val = 1;
        break;
    }
    glutPostRedisplay();
}
```

```
void menu(int i)
{
   val = i;
   glutPostRedisplay();
}


void createMenu(void)
{
   mainmenu = glutCreateMenu(menu);
   glutAddMenuEntry("Go Home",0);
   glutAddMenuEntry("Hydrogen", 1);
   glutAddMenuEntry("Helium", 2);
   glutAddMenuEntry("Lithium", 3);
   glutAddMenuEntry("Beryllium", 4);
   glutAddMenuEntry("Boron", 5);
   glutAddMenuEntry("Carbon", 6);
   glutAddMenuEntry("Nitrogen", 7);
   glutAddMenuEntry("Oxygen", 8);
   glutAddMenuEntry("Fluorine", 9);
   glutAddMenuEntry("Neon", 10);
   glutAddMenuEntry("Exit",11);

   glutAttachMenu(GLUT_RIGHT_BUTTON);
}


// Main function
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(1150, 700);
   glutInitWindowPosition(50, 50);

   glutCreateWindow("2D Atom with Details");
   gluOrtho2D(0, 1150, 0, 700);

   glutDisplayFunc(Draw);
   glutKeyboardFunc(keyboard);
   glutSpecialFunc(specialKey);
   createMenu();

   glutMainLoop();
   return 0;
}
```

CHAPTER – 3

RESULT ANALYSIS

3.1 Snap Shots

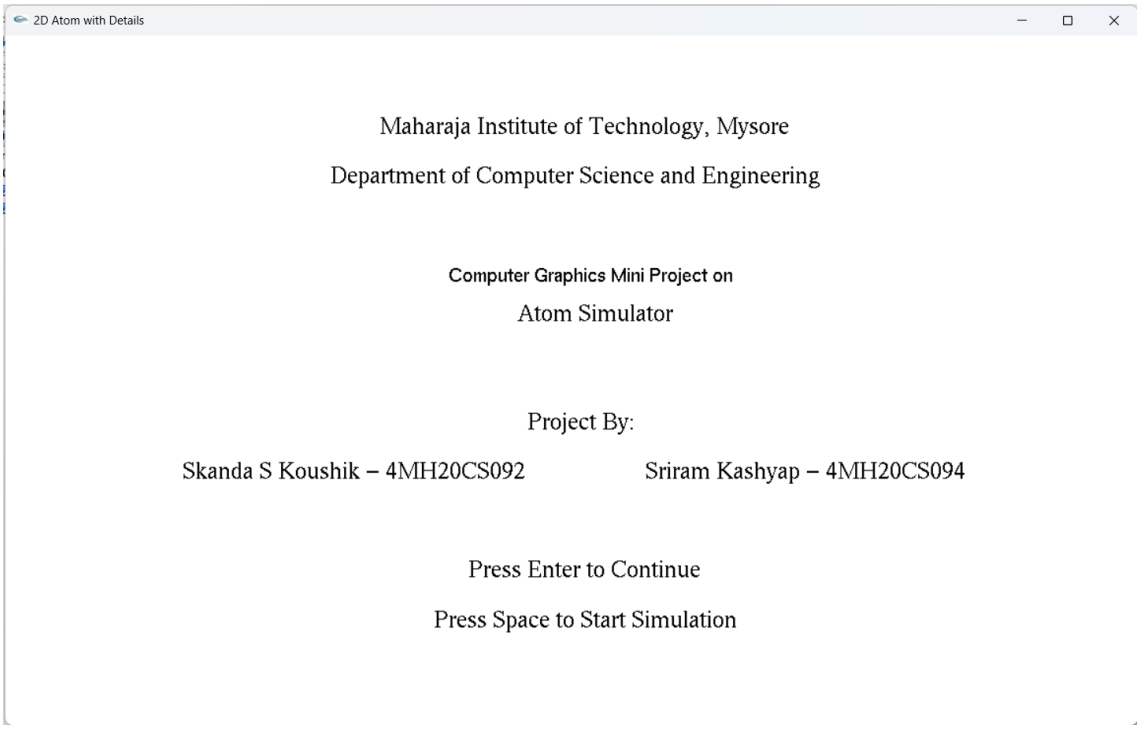


Fig 01: Home View

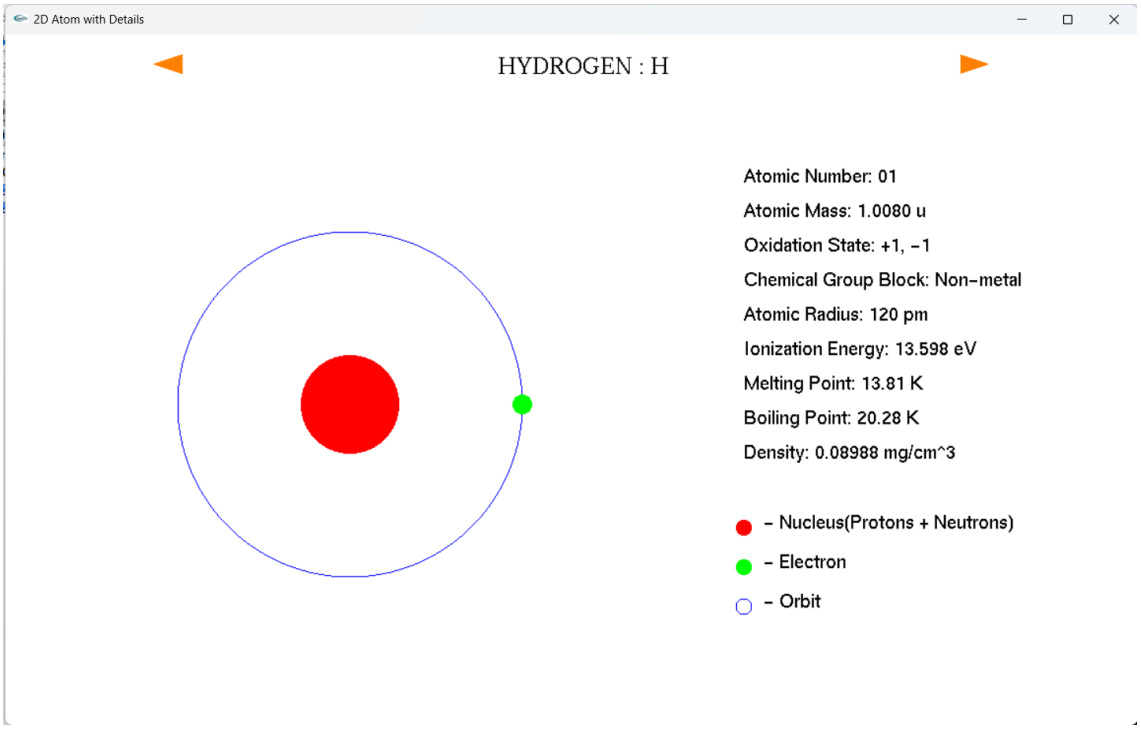


Fig 02: Hydrogen

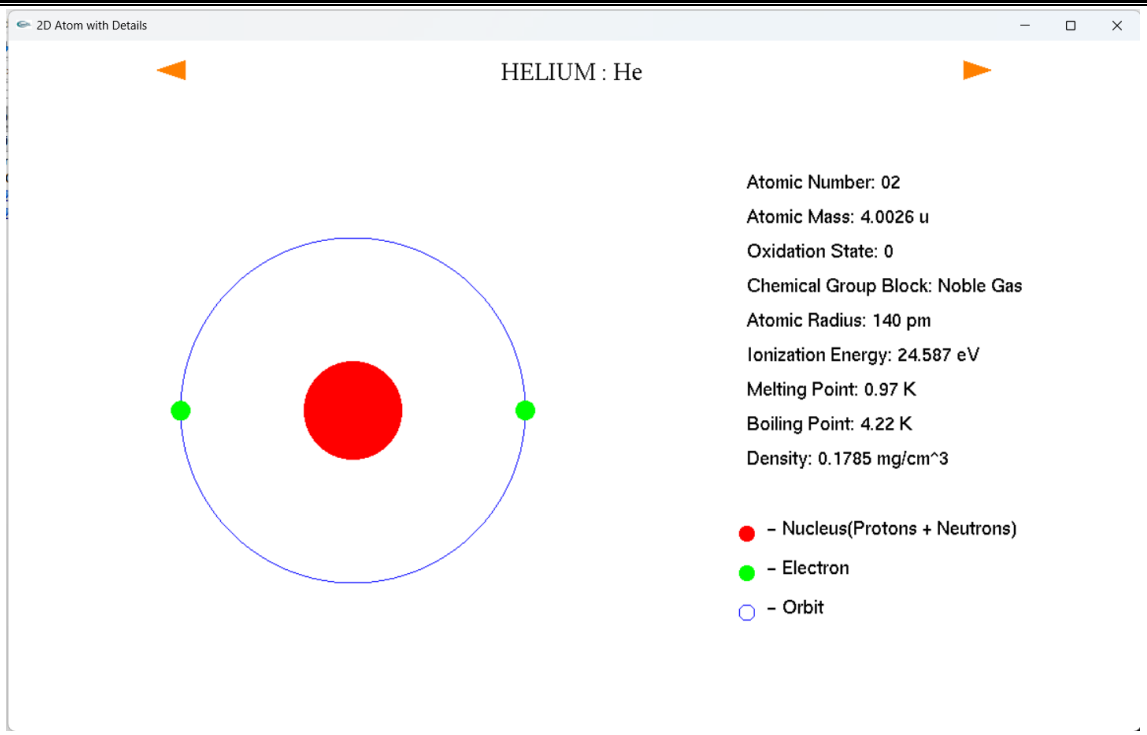


Fig 03: Helium

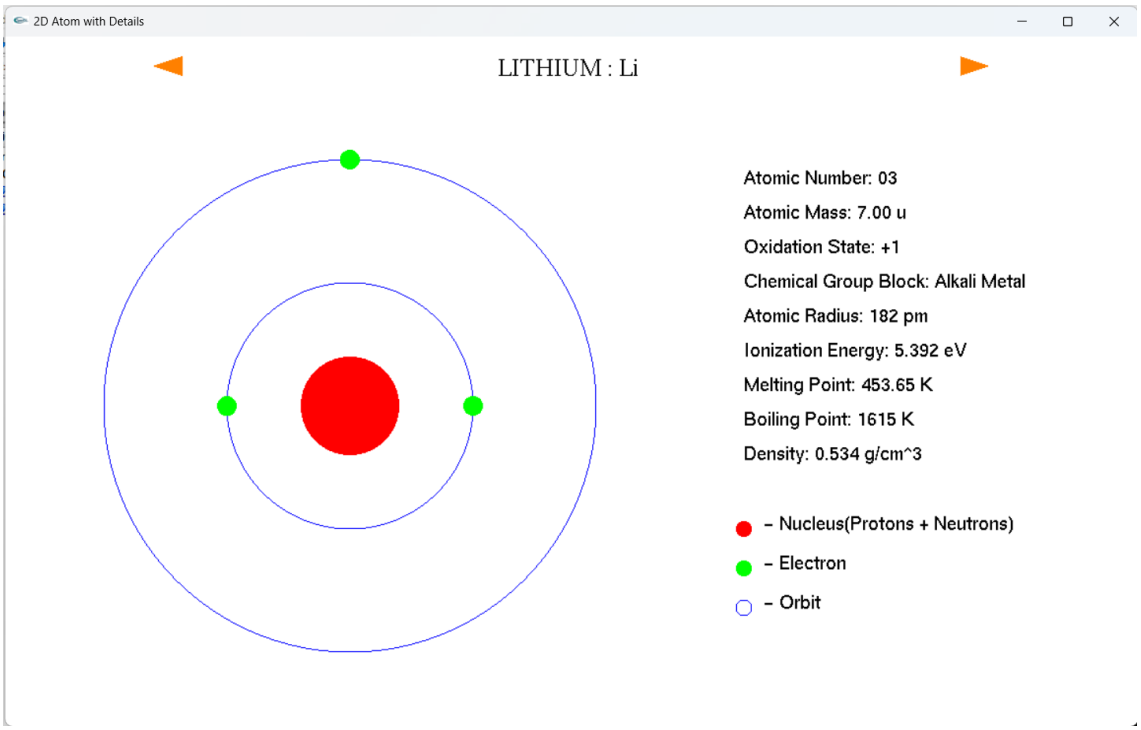


Fig 04: Lithium

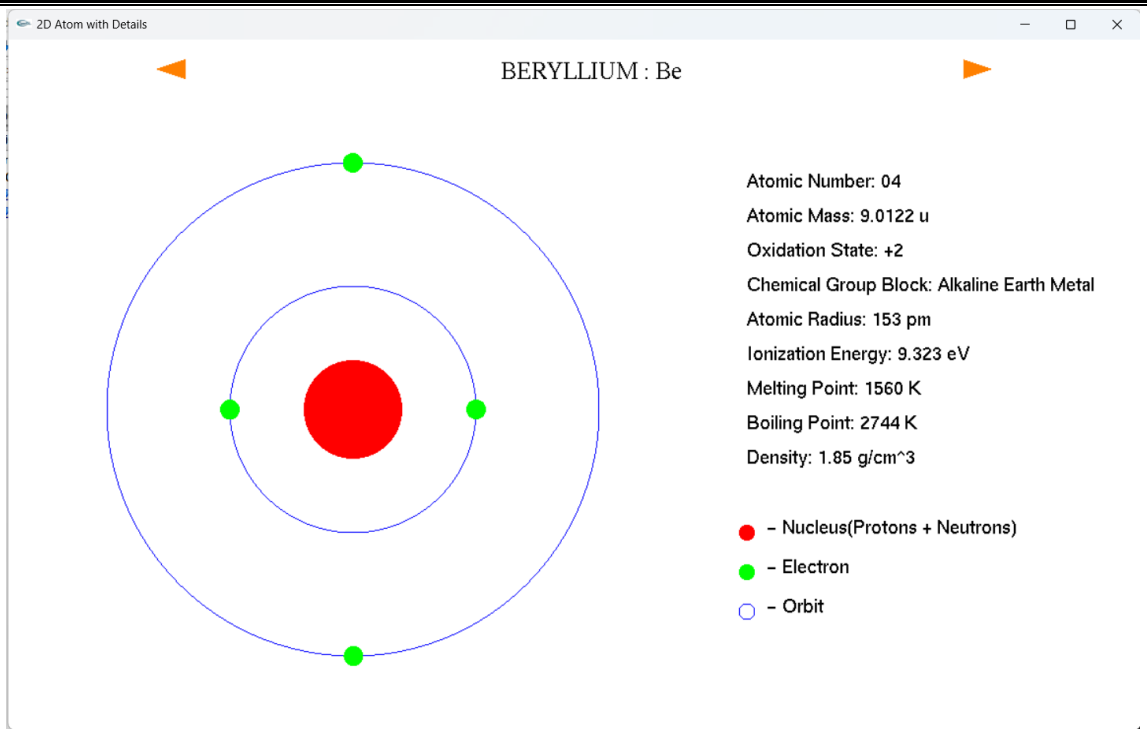


Fig 05: Beryllium

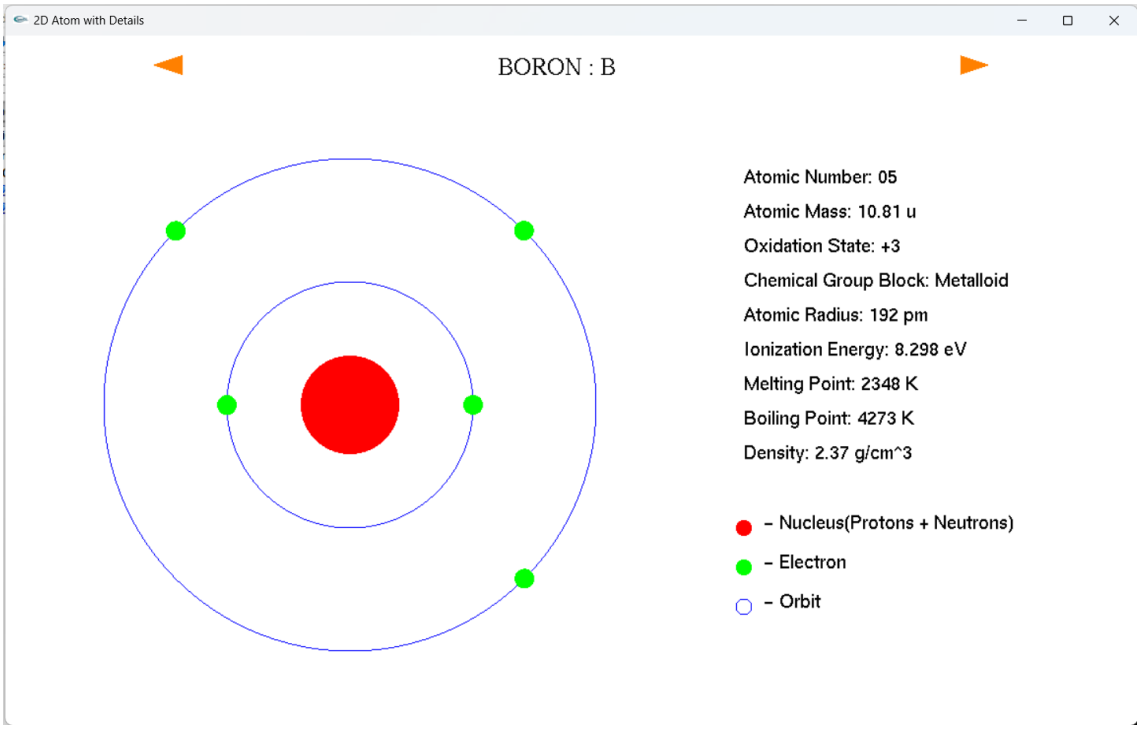


Fig 06: Boron

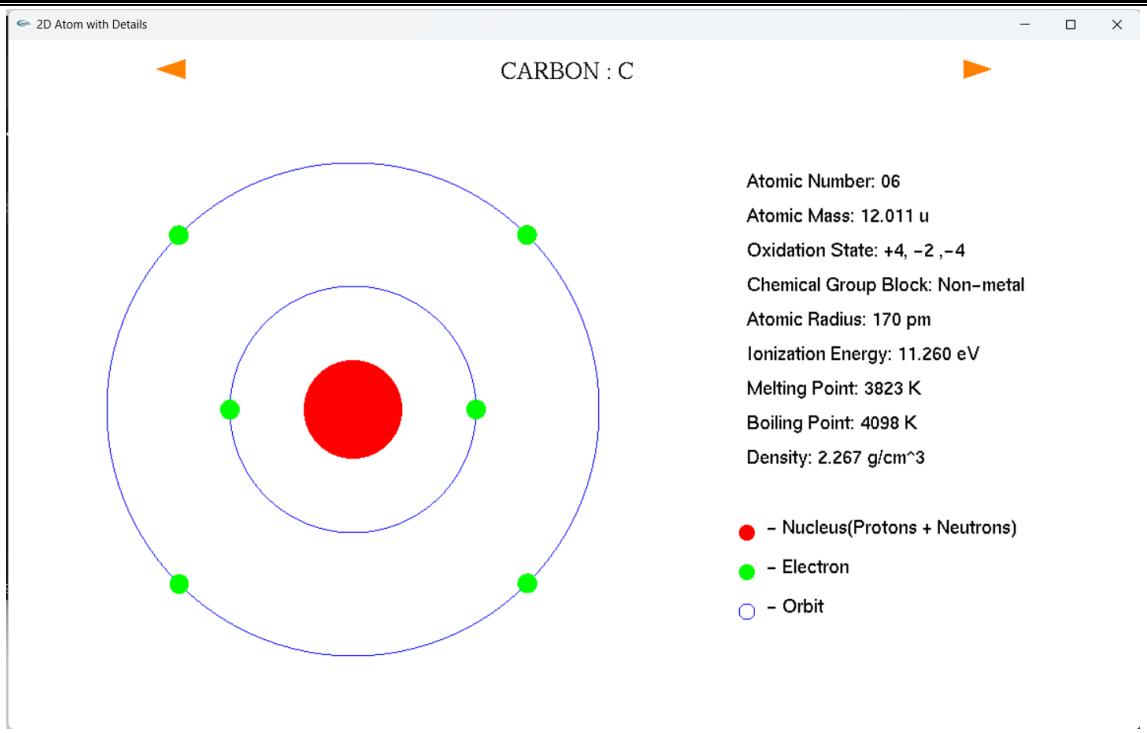


Fig 07: Carbon

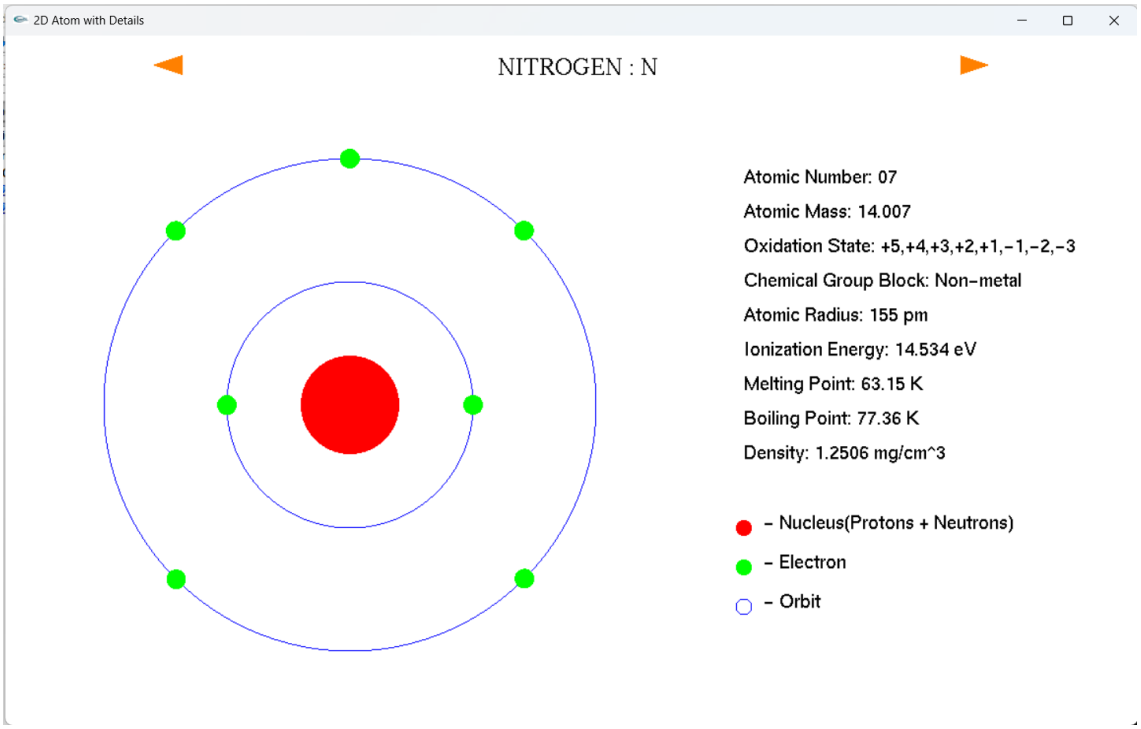


Fig 08: Nitrogen

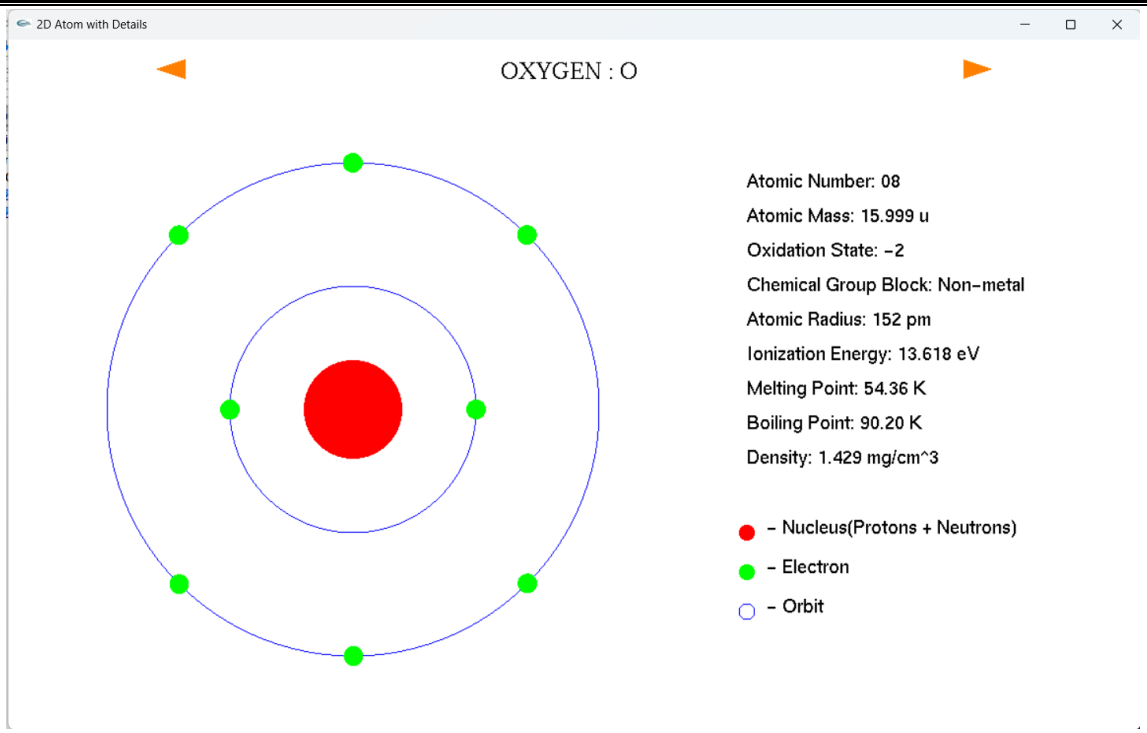


Fig 09: Oxygen

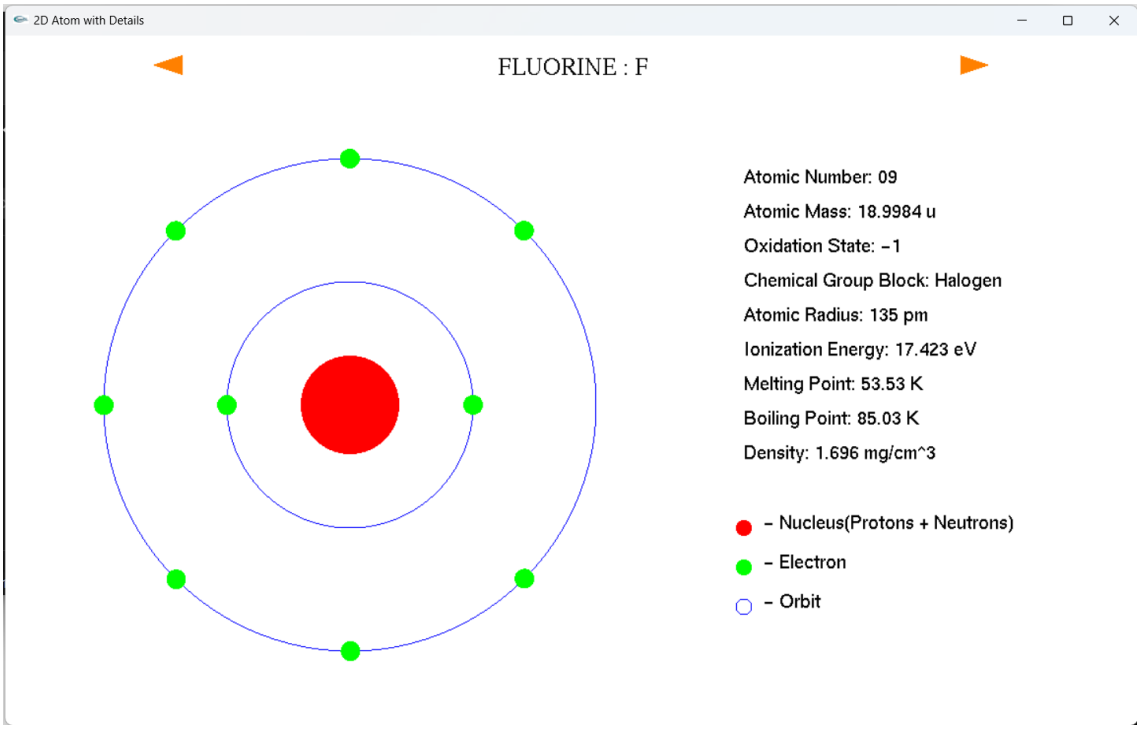


Fig 09: Fluorine

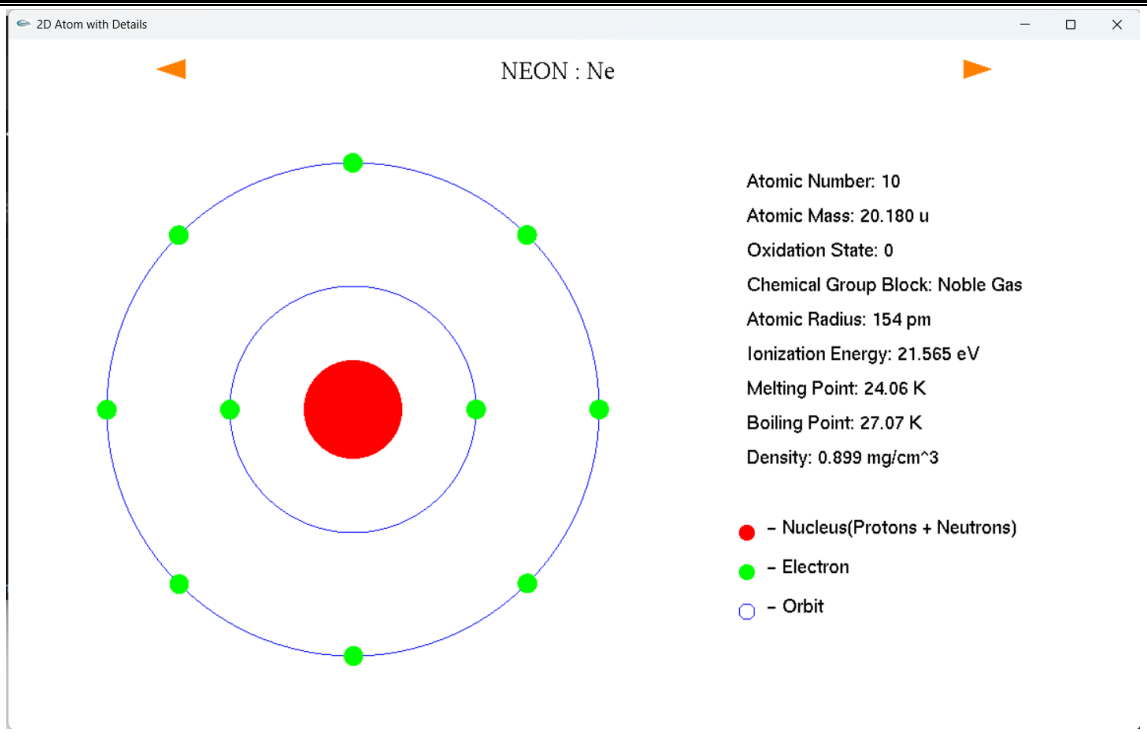

2D Atom with Details
NEON : Ne
Atomic Number: 10
Atomic Mass: 20.180 u
Oxidation State: 0
Chemical Group Block: Noble Gas
Atomic Radius: 154 pm
Ionization Energy: 21.565 eV
Melting Point: 24.06 K
Boiling Point: 27.07 K
Density: 0.899 mg/cm^3
- Nucleus(Protons + Neutrons)
- Electron
- Orbit

Fig 11: Neon

3.2 Discussions

When the Atom Simulator program is executed, the display window shows a 2D representation of atoms with various details. The program provides the following views and interactions:

- **Home View**: (Fig 01) The initial view displays information about the project and the developers. Users can press Enter to continue and Space to start the simulation.

- **Atom Views**: (Fig 02-11) The program allows users to view different atoms by selecting them using the left and right arrow keys. Each atom is represented by a filled circle with rotating circles around it to simulate electrons.

- **Details View**: (Fig 02-11) When an atom is selected, detailed information about the atom is displayed, including atomic number, atomic mass, oxidation state, chemical group block, atomic radius, ionization energy, melting point, boiling point, and density.

- **Interaction**: Users can interact with the program using the keyboard and mouse. The following interactions are supported:

    1. **Keyboard**: Users can press 'q', 'Q', or Esc to exit the program. Pressing Enter advances to the next atom. Pressing 'h' or 'H' goes back to the home view. Pressing Space toggles, the rotation of the atom.

    2. **Mouse**: Users can move the mouse or touchpad to navigate around the room and explore the atoms in a 3D-like environment. The mouse function of OpenGL is used to enable this functionality.

CHAPTER – 4

CONCLUTION AND FUTURE WORK

4.1 Conclusion

The Atom Simulator program offers an interactive and educational experience for users to explore different atoms in a 2D representation. The program allows users to view atoms, navigate through the room, and access detailed information about each atom. By providing a visual representation of atoms and their properties, the program enhances understanding and engagement in the field of chemistry. It serves as a valuable tool for students, researchers, and anyone interested in learning about atomic structures and properties. With its user-friendly interface and interactive features, the Atom Simulator program offers an enjoyable and informative way to explore the world of atoms.

4.2 Future Enhancements

3D Visualization: Upgrade the program to support 3D rendering, allowing users to explore atoms in a more immersive and realistic environment. This can provide a better sense of depth and spatial arrangement of atoms.

Additional Atom Models: Expand the program to include a wider range of atom models, covering more elements from the periodic table. This would offer users the opportunity to explore and learn about a greater variety of atoms.

CHAPTER – 5

REFERENCES

https://github.com/Adarsh232001/Atom-Simulation

https://learnopengl.com/In-Practice/Text-Rendering

https://stackoverflow.com/

https://www.opengl.org/resources/libraries/