Atomic Structure Simulation: A 3D Visualization of Electrons in Various Atomic Orbits

Submitted by

**Jeevan E G-1RVU22CSE069**

**Pranav P Kulkarni-1RVU22CSE119**

School of Computer Science

RV University

Submitted to

**Prof. Evlin Vidyu Latha**

Designation

School of Computer Science

RV University

Submission date-12-04-2025

# Certificate

Certified that **Jeevan E G(1RVU22CSE069), Pranav P Kulkarni(1RVU22CSE119)** have carried out the Project work presented in this report entitled

" **Atomic Structure Simulation: A 3D Visualization of Electrons in Various Atomic Orbits**"

for the Computer Graphics course in partial fulfilment of B.Tech.(Hons) in School of Computer Science and Engineering of the RV University, Bengaluru during the year 2025-2026

under my supervision. The report embodies results of original work and studies carried out by students themselves and the contents of the Project do not form the basis for the award of any other degree to the candidate or to anybody else.

Name of Professor: **Evlin Vidyu Latha**

Designation: Assistant Professor

Date:12-04-2025

1. **Problem Statement:**

The aim of this project is to develop an educational visualization tool that simulates the atomic structure of the first ten elements of the periodic table (Hydrogen to Neon) using OpenGL and GLUT. The program should provide an interactive graphical representation of different atoms based on the Bohr model, allowing users to visualize the nucleus and orbiting electrons with proper animation. The program should enable users to select different elements, start/stop animations, and interact with the simulation through both mouse and keyboard controls.

### 2)Implementation Details :

### 1. Initialization and Window Setup

**GLUT Initialization Functions:**

* **glutInit(&argc, argv)**: Initializes the GLUT library and processes command line arguments
* **glutInitWindowPosition(100, 100)**: Sets the initial position of the window on the screen
* **glutInitWindowSize(700, 700)**: Sets the initial dimensions of the window (700x700 pixels)
* **glutCreateWindow("ATOM SIMULATION")**: Creates a window with the specified title

**Graphics Initialization:**

* **gluOrtho2D(-1000, 1000, -1000, 1000)**: Sets up a 2D orthographic projection with specified coordinate boundaries
  + This establishes a coordinate system with x and y both ranging from -1000 to 1000

**2. Drawing and Rendering Functions**

**Basic Drawing Commands:**

* **glBegin(GL\_POINTS)**: Starts defining a sequence of individual points
* **glBegin(GL\_POLYGON)**: Starts defining a filled polygon
* **glEnd()**: Ends the current drawing primitive sequence
* **glVertex2i(x, y)**: Specifies a vertex point with integer coordinates
* **glVertex2f(x, y)**: Specifies a vertex point with floating-point coordinates

**Color Setting:**

* **glClearColor(0, 0, 0.1, 0.9)**: Sets the background color (dark blue) with alpha value
* **glColor3f(r, g, b)**: Sets the current drawing color using RGB values (0.0-1.0)
  + Used with values like (1,1,1) for white, (0,0,1) for blue, and (2,0,0) for bright red

**Text Rendering:**

* **glRasterPos3f(x, y, z)**: Sets the raster position for text drawing
* **glutBitmapCharacter(font, character)**: Renders a character using a bitmap font
  + Used with fonts like GLUT\_BITMAP\_HELVETICA\_10, GLUT\_BITMAP\_HELVETICA\_12, and GLUT\_BITMAP\_HELVETICA\_18

**Screen Clearing and Buffer Management:**

* **glClear(GL\_COLOR\_BUFFER\_BIT)**: Clears the color buffer to the background color
* **glutSwapBuffers()**: Swaps the back and front buffers to display the rendered scene
  + This is essential for double-buffered rendering which prevents flickering

**3. Transformation Functions**

**Matrix Operations:**

* **glPushMatrix()**: Saves the current transformation matrix on a stack
* **glRotatef(angle, x, y, z)**: Applies a rotation transformation
  + Used as glRotatef(angle, 0, 0, 1) to rotate around the z-axis by 'angle' degrees
* **glPopMatrix()**: Restores the previously saved transformation matrix
  + These matrix operations are crucial for animating the orbiting electrons

**4. Event Handling and Callback Registration**

**Display and Animation:**

* **glutDisplayFunc(display)**: Registers the function to call when the window needs to be redrawn
* **glutIdleFunc(rotate)**: Sets the function to call during idle time (used for animation)
  + Setting this to NULL stops the animation

**User Input Handling:**

* **glutMouseFunc(mouseControl)**: Registers the function to handle mouse button events
* **glutKeyboardFunc(keyboard)**: Registers the function to handle regular keyboard input
* **glutSpecialFunc(fkey)**: Registers the function to handle special keys (e.g., function keys)

**Window Management:**

* **glutReshapeWindow(width, height)**: Changes the size of the window
* **glutGet(GLUT\_SCREEN\_WIDTH)**, **glutGet(GLUT\_SCREEN\_HEIGHT)**: Retrieves screen dimensions

**5. Menu System Functions**

**Menu Creation and Management:**

* **glutCreateMenu(menu)**: Creates a menu and returns its ID
* **glutAddMenuEntry(label, value)**: Adds a menu item with specified label and value
* **glutAddSubMenu(label, submenu\_id)**: Adds a submenu to the current menu
* **glutAttachMenu(button)**: Attaches the menu to a specific mouse button
  + Used with GLUT\_RIGHT\_BUTTON to attach to right mouse button
* **glutDetachMenu(button)**: Detaches the menu from a specified mouse button

**6. Application Control**

**Program Flow Control:**

* **glutPostRedisplay()**: Marks the current window for redisplay
  + Triggers a call to the display function on the next iteration of the main loop
* **glutMainLoop()**: Enters the GLUT event processing loop
  + This is where the program spends most of its execution time, processing events
* **exit(0)**: Terminates the program with a success status code

**7. Mathematical Functions**

The program uses several mathematical functions from the standard math library:

* **cos(angle)** and **sin(angle)**: Calculate coordinates on circular paths
  + These are used extensively to position orbits and electrons using parametric equations

**8. Core Drawing Implementation**

Several custom drawing functions were implemented using OpenGL primitives:

* **Circle Drawing**: Uses GL\_POINTS to draw orbit circles
* **Nucleus Drawing**: Uses GL\_POLYGON to draw a filled circle representing the nucleus
* **Electron Drawing**: Uses GL\_POLYGON to draw small filled circles representing electrons at various positions
* **Text Rendering**: Multiple functions for different text sizes and positions

**9. Element Visualization Strategy**

The program visualizes elements by:

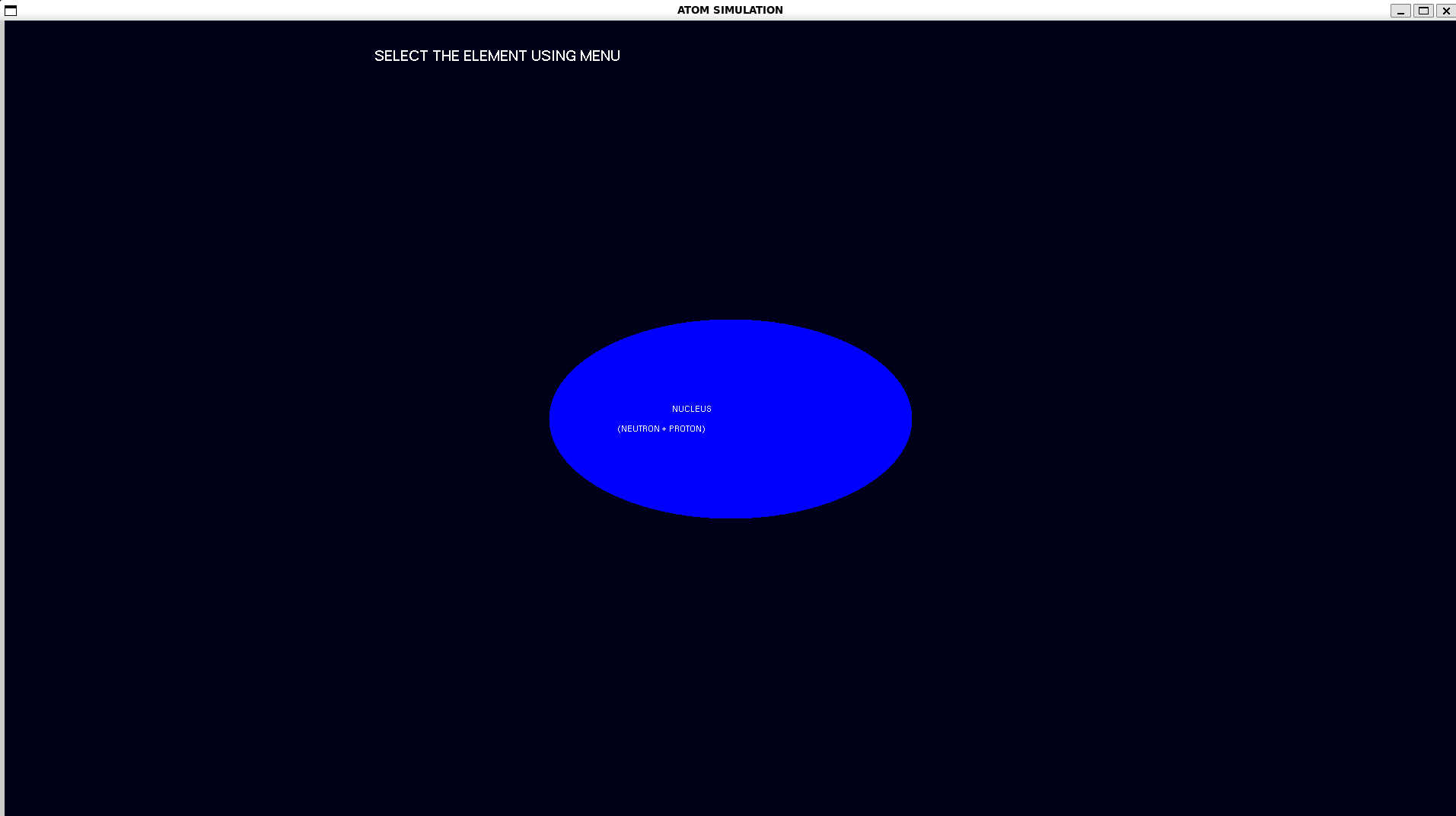
1. Drawing the nucleus as a central blue circle
2. Drawing one orbit (radius 400) for the first electron shell (K shell)
3. Drawing a second orbit (radius 600) for the second electron shell (L shell)
4. Placing electrons at specific positions:
   * First shell: Right and left positions (K shell can hold 2 electrons)
   * Second shell: Top, bottom, and diagonal positions (L shell can hold 8 electrons)
5. Using rotation transformation to animate electrons orbiting around the nucleus

Output:

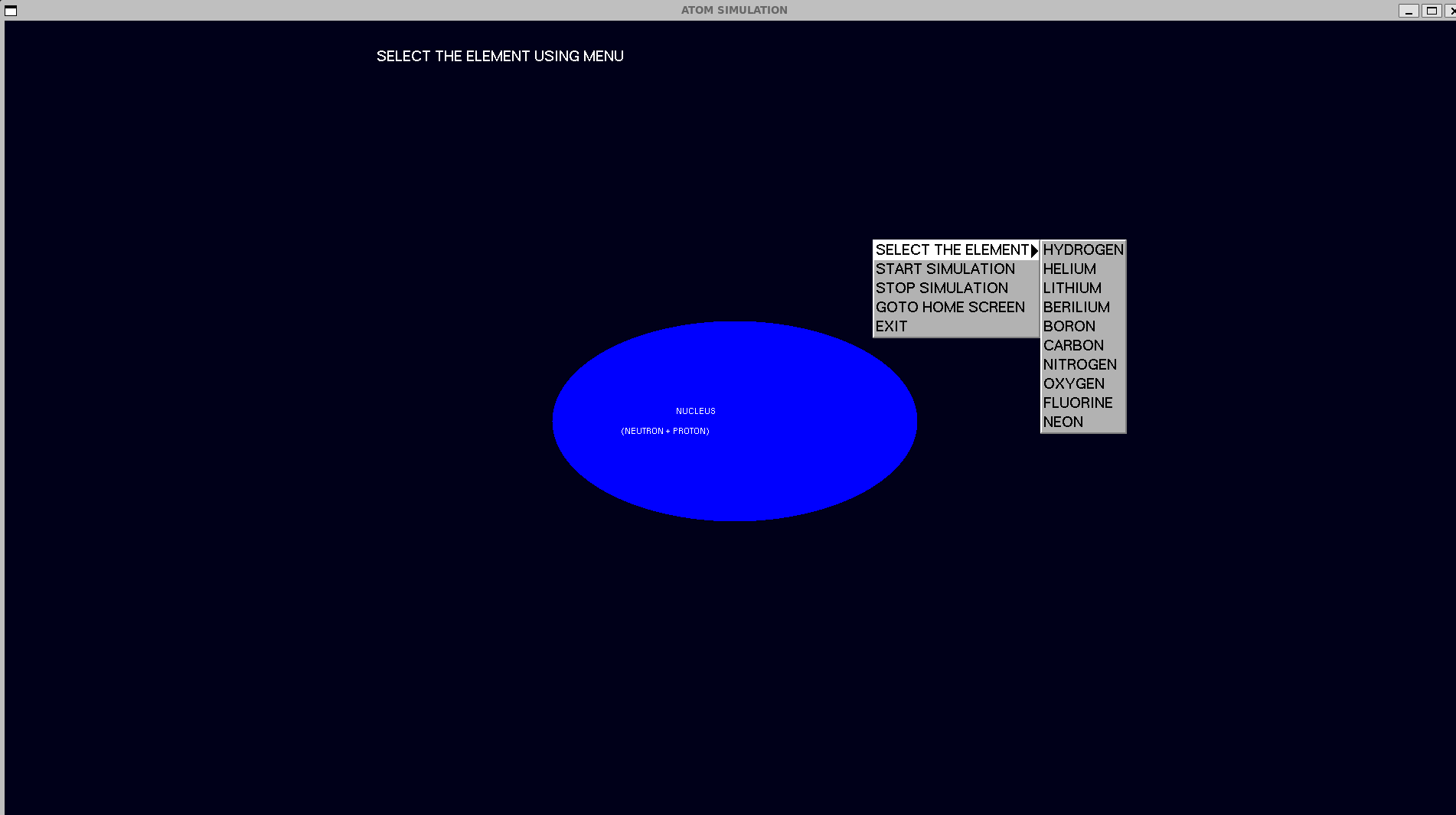
* Front Page



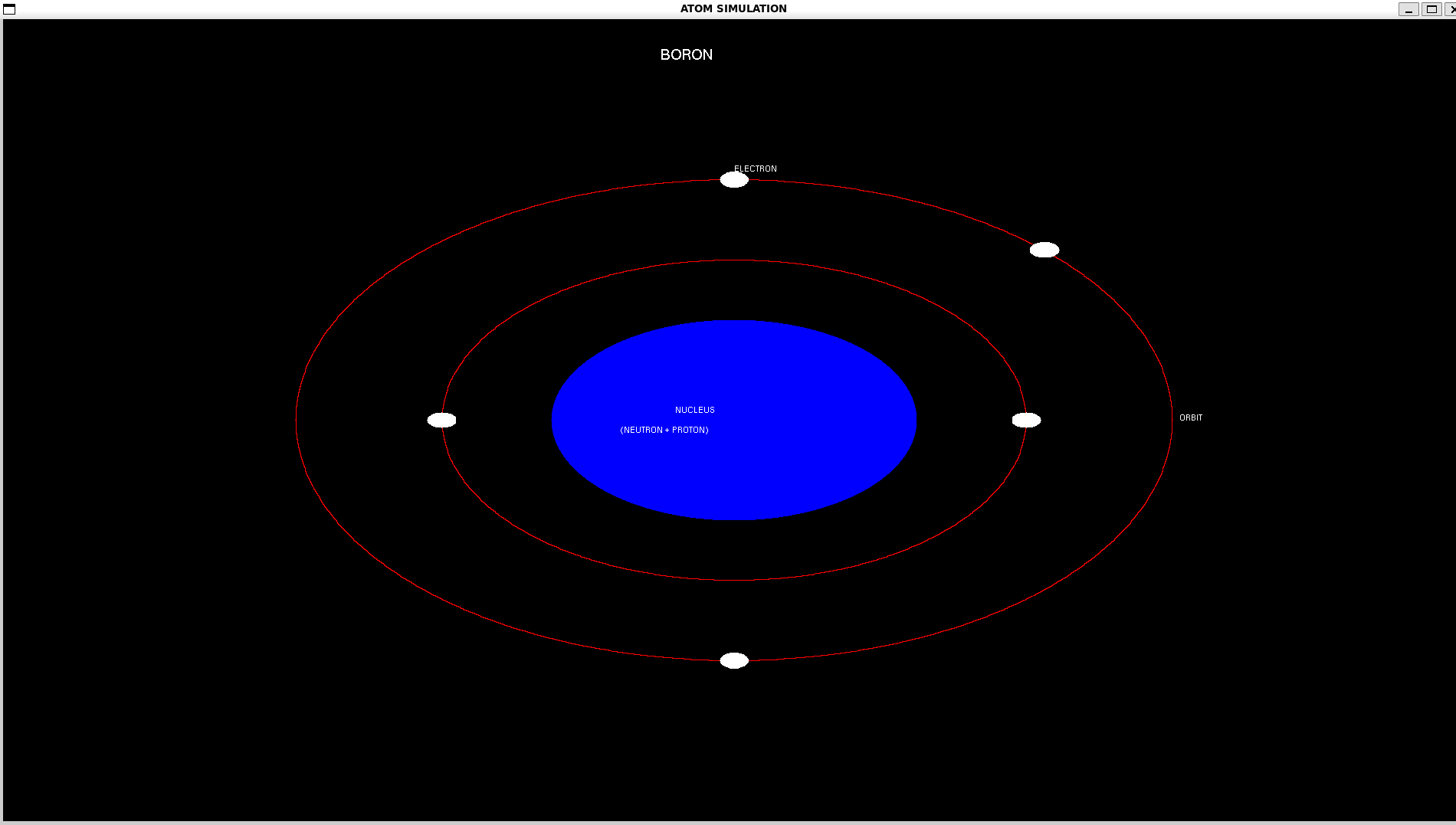
* Nucleus Drawn



* Showing Menu and Submenu to select Elements



* Sample Element (BORON)



GitHub Link: