Visvesvaraya Technological University
Belagavi-590 018, Karnataka

A Mini Project Report on

"Atom Simulation"

Mini Project Report submitted in partial fulfillment of the requirement for the Computer Graphics Laboratory with Mini Project [18CSL67]

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by
Manjesh M [1JT19CS049]
Mohammed maaz [1JT19CS054]

Department of Computer Science and Engineering
Accredit by NBA, New Delhi
Jyothy Institute of Technology Tataguni,
Bengaluru-560082

Jyothy Institute of Technology
Tataguni, Bengaluru-560082
Department of Computer Science and Engineering

CERTIFICATE

Certified that the mini project work entitled **"Atom Simulation"** carried out by **Manjesh M [1JT19CS049]** and **Mohammed maaz [1JT19CS054],** bonafide students of Jyothy Institute of Technology, in partial fulfilment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** department of the **Visvesvaraya Technological University, Belagavi** during academic the year **2021-2022**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Mrs. Roopa Omkar Deshpande
Guide, Asst. Professor
Dept. of CSE

Dr. Prabhanjan S
Professor & HOD
Dept. of CSE

External Viva Examiner

Signature with Date:

1.

2.

ACKNOWLEDGEMENT

Firstly, we are very grateful to the esteemed institution **"Jyothy Institute Of Technology"** for providing us an opportunity to complete our project.

We express our sincere thanks to our **Principal Dr. Gopalakrishna K** for providing us with adequate facilities to undertake this mini project.

We would like to thank **Dr. Prabhanjan Soukar, Professor and Head of Computer Science and Engineering Department** for providing his valuable support.

We would like to thank our guide **Mrs. Roopa Onkar Deshpande, Assistant Professor** for her keen interest and guidance in preparing this work.

Finally, we would thank all our friends who have helped directly or indirectly in this mini project.

Manjesh M [1JT19CS049]

Mohammed maaz [1JT19CS054]

ABSTRACT

Everything you see around you is made up of atoms, and all atoms consist of subatomic particles. In the Atom simulation, you will learn the names of the basic subatomic particles and understand.

As a part of the project, you'll see how the electrons are revolving around the nucleus in their respective orbits. One can see and spot the nucleus, atoms and electrons and can understand how an electron revolves around the nucleus. The project has made in such a way that one can easily understand the simulation of atoms.

This project has been developed in Ubuntu OS with interfacing keyboard and mouse with menu driven interface. And plans to include lighting, shading and other features in future enhancement.

This project is written in C and used OpenGL (Open Graphics Library). Open Graphics Library is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit, to achieve hardware-accelerated rendering.

The main 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.

.

TABLE OF CONTENTS

CHAPTER 1
INTODUCTION

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO CG

Graphics is defined as any sketch or a drawing or a special network that pictorially represents some meaningful information. Computer Graphics is used where a set of image needs to be manipulated or the creation of the image in the form of pixels and is drawn on the computer. Computer Graphics can be used in digital photography, film, entertainment, electronic gadgets and all other core technologies which are required. It is a vast subject and area in the field of computer science. Computer Graphics can be used in UI design, rendering, geometric object, animation and many more. In most area, computer graphics is an abbreviation of CG. There are several tools used for implementation of Computer Graphics.

Computer Graphics refers to several things:

- The manipulation and the representation of the image or the data in a graphicalmanner.
- Various technologies required for the creation and manipulation.
- Digital synthesis and its manipulation.

Applications

- **Computer Graphics are used for aided design for engineering and architectural system-** These are used in electrical automobile, electro-mechanical, mechanical, electronic devices. For example: gears and bolts.
- **Computer Art –** MS Paint, Adobe Photoshop.
- **Presentation Graphics –** It is used to summarize financial statistical scientific or economic data. For example- Bar chart, Line chart.
- **Entertainment-** It is used in motion picture, music video, television gaming.
- **Education and training-** It is used to understand operations of complex system. It is also used for specialized system such for framing for captains, pilots and so on.
- **Visualization-** To study trends and patterns. For example- Analyzing satellite photo of earth.

1.2 INTRODUCTION TO OpenGL

Interface

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offer functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects. Functions in the main GL library have names that begin with gl and are stored in a library usually referred to as GL. The secondis the OpenGL Utility Library (GLU). The library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with the letter glu. Rather than using a different library for each system we use a readily available library called OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

Overview

- OpenGL (Open Graphics Library) is the interface between a graphics program and graphics hardware. It is streamlined. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.
- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- It is system independent. It does not assume anything about hardware or operating• system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a state machine. At any moment during the execution of a program there is a current model transformation.
- It is a rendering pipeline. The rendering pipeline consists of the following steps:

  *Defines objects mathematically.
  *Arranges objects in space relative to a viewpoint.
  *Calculates the color of the objects

1.3 INTRODUCTION TO ATOM SIMULATION

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.

CHAPTER 2
OpenGL Architecture

CHAPTER 2

2.1 OpenGL Architecture

1) Pipeline Architectures

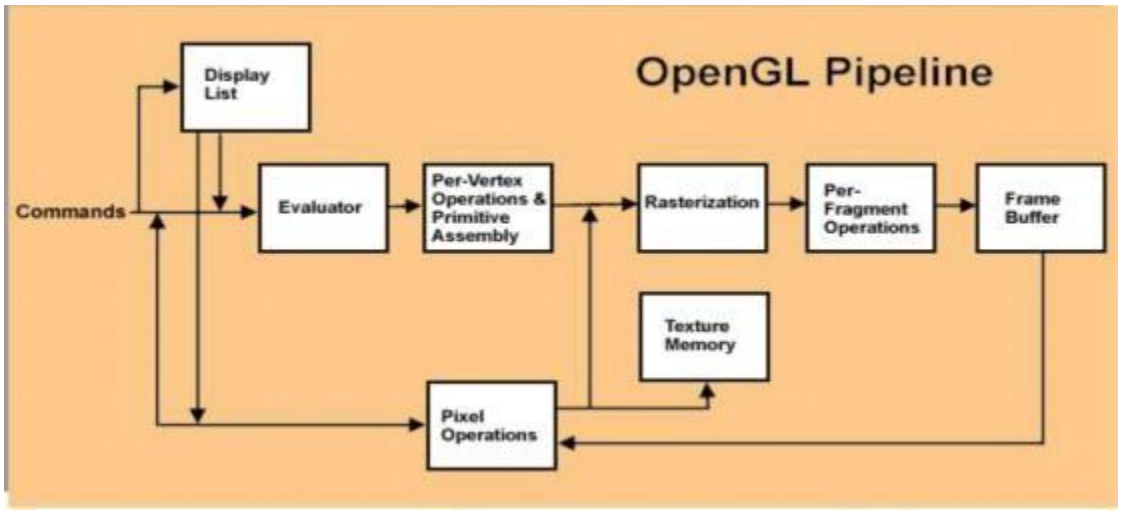

FIG:2.1 OPENGL PIPELINE ARCHITECTURE

- **Display Lists:** All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. (1 alternative to retaining data in a display list is processing the data immediately - also known as immediate mode.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.
- **Evaluators:** All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive thevertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colours,and spatial coordinate values from the control points.
- **Per-Vertex Operations:** For vertex data, next is the "per-vertex operations" stage, which converts. The vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen. If advanced features are enabled, this stage is even busier. If texturing is used, texturecoordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light sourceposition, material properties, and other lighting information to produce a colour value
- **Primitive Assembly:** Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane.Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped. In some cases,this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then view port and depth (z coordinate) operations are applied. If culling is enabled and the

primitive is a polygon, it then may be rejected bya culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines. The results or this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture- coordinate values and guidelines for the rasterization step.

- **Pixel Operations:** While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number ofcomponents. Next the data is scaled, biased, and processed by a pixel map. The resultsare clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias,mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixelcopy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.
- **Texture Assembly:** An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images areused, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to acceleratetexture performance. There may be specialized, high-performance texture memory. Ifthis memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.
- **Rasterization:** Rasterization is the conversion of both geometric and pixel data intofragments. Each fragment square corresponds to a pixel in the frame buffer. Line andpolygon stipples, line width, point size, shading model, and coverage calculations to support ant aliasing are taken into consideration as vertices are connected into lines orthe interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

2.2 OpenGL Engine and Drivers

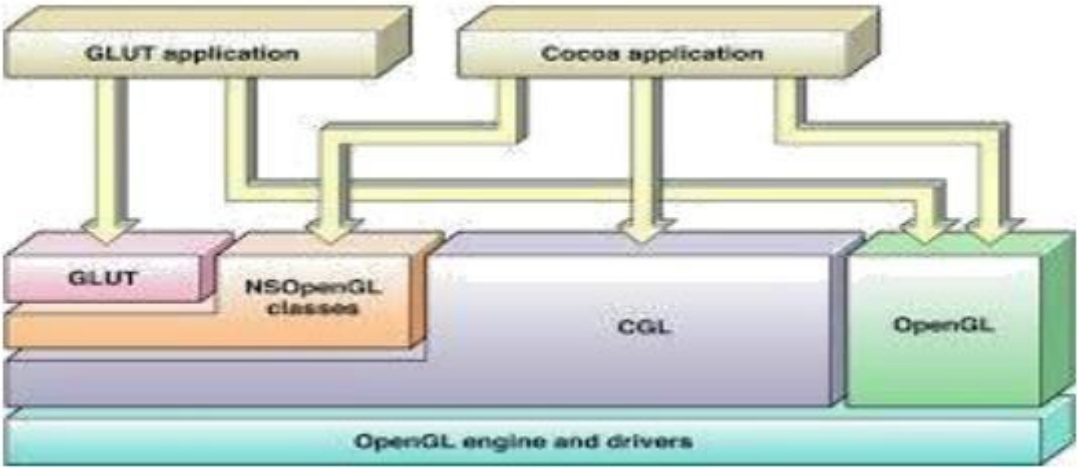

FIG:2.2 OPENGL ENGINE AND DRIVERS

CPU-GPU Cooperation

The architecture of OpenGL is based on a client-server model. An application program written to use the OpenGL API is the "client" and runs on the CPU. The implementation of the OpenGL graphics engine (including the GLSL shader programs we write) is the "server" and runs on the GPU. Geometry and many other types of attributes are stored in buffers called Vertex Buffer Objects (or VBOs). These buffers are allocated on the GPU and filled by your CPU program. We will get our first glimpse into this process (including how these buffers are allocated, used, and deleted) in the first sample program we will study.

Window Manager Interfaces

OpenGL is a pure output-oriented modeling and rendering API. It has no facilities for creating and managing windows, obtaining runtime events, or any other such window system dependent operation. OpenGL implicitly assumes a window-system interface that fills these needs and invokes user-written event handlers as appropriate.

It is beyond the scope of these notes to list, let alone compare and contrast, all window manager interfaces that can be used with OpenGL. Two very common window system interfaces are:

GLFW: Runs on Linux, Macintosh, and Windows.
GLUT: (freeglut): A window interface that used to be the most common one used when teaching OpenGL. It, too, runs on Linux, Macintosh, and Windows.

2.3 OpenGL Application Development-API's

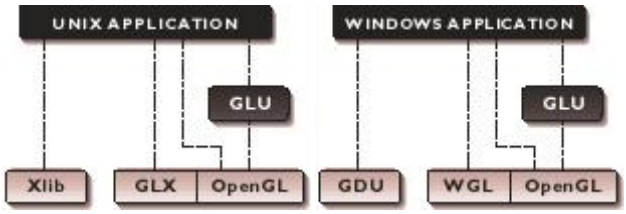

FIG:2.3 APPLICATIONS DEVELOPMENT(API'S)

This diagram demonstrates the relationship between OpenGL GLU and windowing APIs.

Leading software developers use OpenGL, with its robust rendering libraries, as the 2D/3D graphics foundation for higher-level APIs. Developers leverage the capabilities of OpenGL to deliver highly differentiated, yet widely supported vertical market solutions. For example,Open Inventor provides a cross-platform user interface and flexible scene graph that makes it easy to create OpenGL applications. IRIS Performer < leverages OpenGL functionality and delivers additional features tailored for the demanding high frame rate markets such as visualsimulation and virtual sets OpenGL Optimizer is a toolkit for real-time interaction, modification, and rendering of complex surface-based models such as those found in CAD/CAM and special effects creation. OpenGL Volumizer is a high-level immediate mode volume rendering API for the energy, medical and sciences markets. OpenGL Shader provides a common interface to support realistic visual effects, bump mapping, multiple textures, environment maps, volume shading and an unlimited array of new effects using hardware acceleration on standard OpenGL graphics cards

CHAPTER 3
IMPLEMENTATION

CHAPTER 3

CODE SECTION:

```
#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 eledl(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[] = "JYOTHY INSTITUTE OF TECHNOLOGY";
        drawhead(-490, 900, 0, cn);
        char pn[] = "Tataguni, Bangalore- 560082";
        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[] = "Manjesh M";
        drawhead(-690, -250, 0, p1);
        char p1u[] = "1JT19CS049";
        drawsubhead(-690, -300, 0, p1u);

        char p2[] = "Mohammed Maaz";
        drawhead(-690, -400, 0, p2);
        char p2u[] = "1JT19CS054";
        drawsubhead(-690, -450, 0, p2u);
```

```
    char gb[] = "GUIDED BY: ";
    drawhead(290, -150, 0, gb);

    char g1[] = "Mrs. Roopa Onkar Deshpande";
    drawhead(290, -250, 0, g1);
    char d1[] = "Asst. Professor, Dept. Of CSE, JIT";
    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 + 6.0;
   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 if(option==14){
        value=-1;
    }
    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("START SIMULATION", 11);
    glutAddMenuEntry("STOP SIMULATION", 12);
    glutAddMenuEntry("GOTO HOME SCREEN",14);
    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;
}
```

Commonly used Functions of OpenGL includes:

glBegin, glEnd	The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives.
glClear	The glClear function clearsbuffers to preset values.
glColor	These functions set the current color
glFlush	The glFlush function forces execution of OpenGL functions in finite time.
glLoadIdentity	The glLoadIdentity function replaces the current matrix with the identity matrix.
glMatrixMode	The glMatrixMode functionspecifies which matrix is the current matrix.
glVertex	These functions specify a vertex.
gluOrtho2D	The gluOrtho2D function defines a 2-D orthographicprojectionmatrix.

CHAPTER 4
RESULTS AND SNAPSHOTS

CHAPTER 4

RESULTS AND SNAPSHOTS

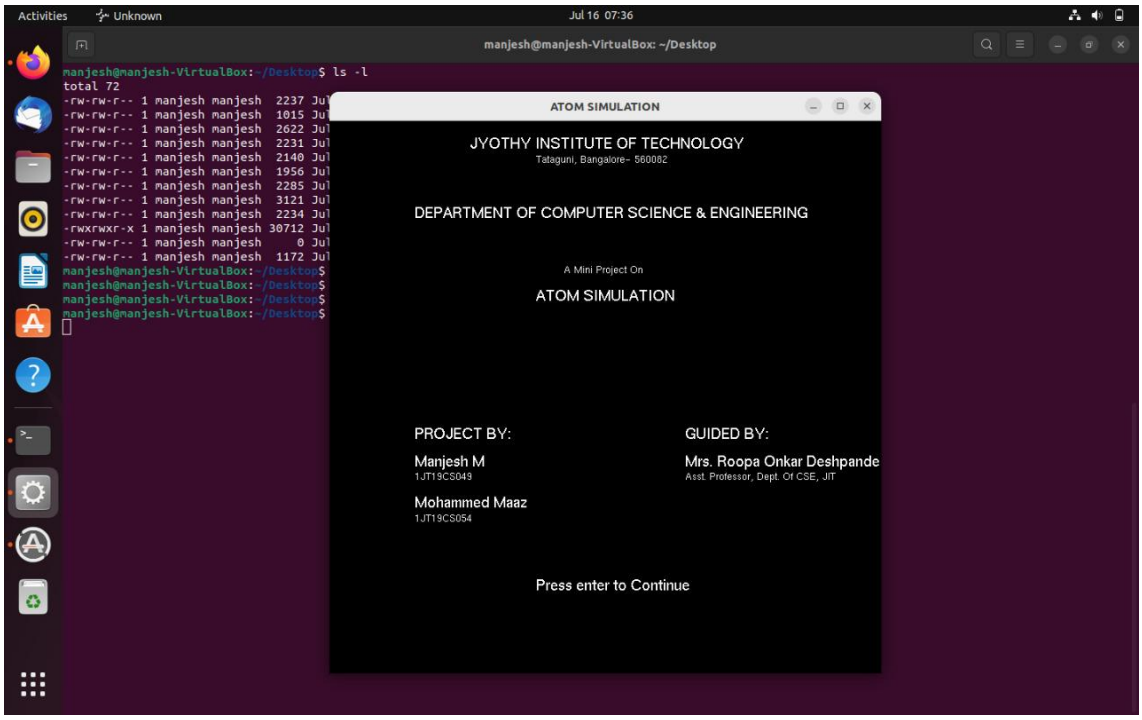

Figure 4.1: Snapshot of Home Screen

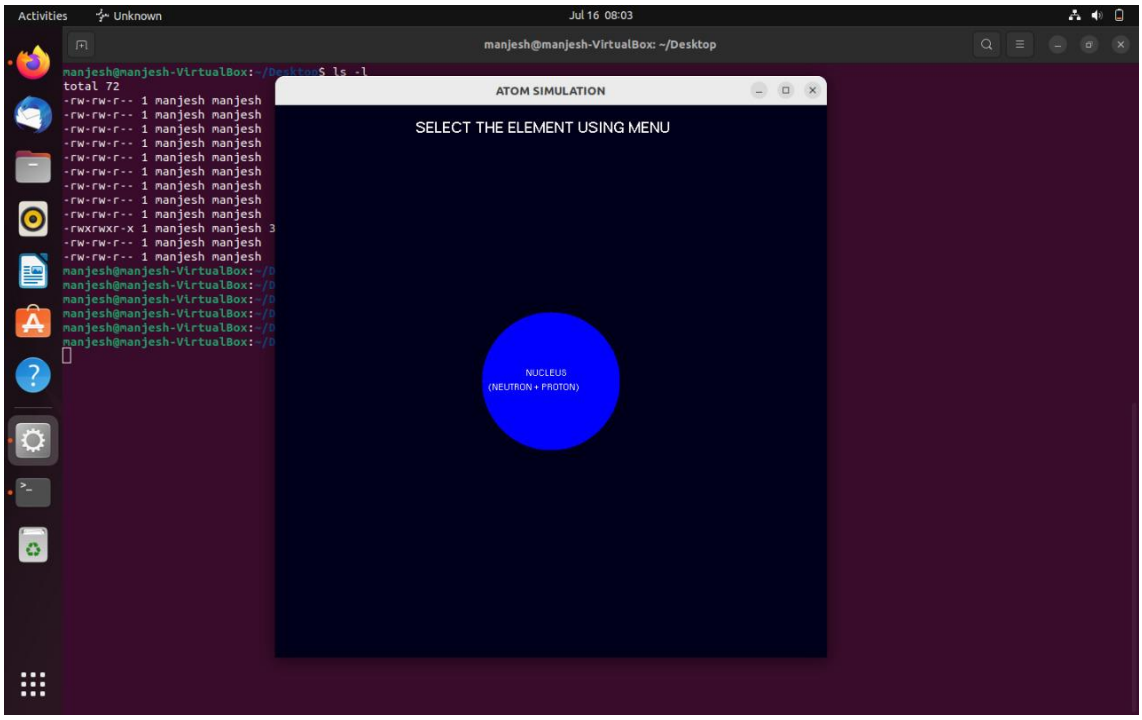

Figure 4.2: Snapshot of Starting Screen

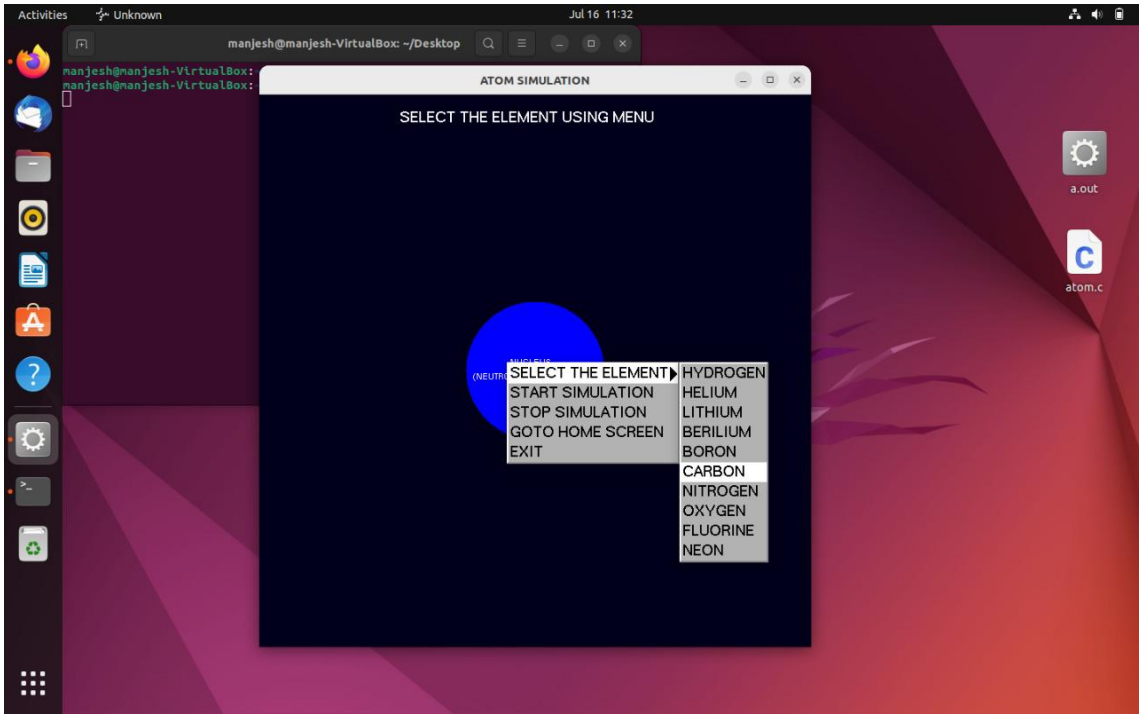

Figure 4.3: Snapshot of Menu Interface

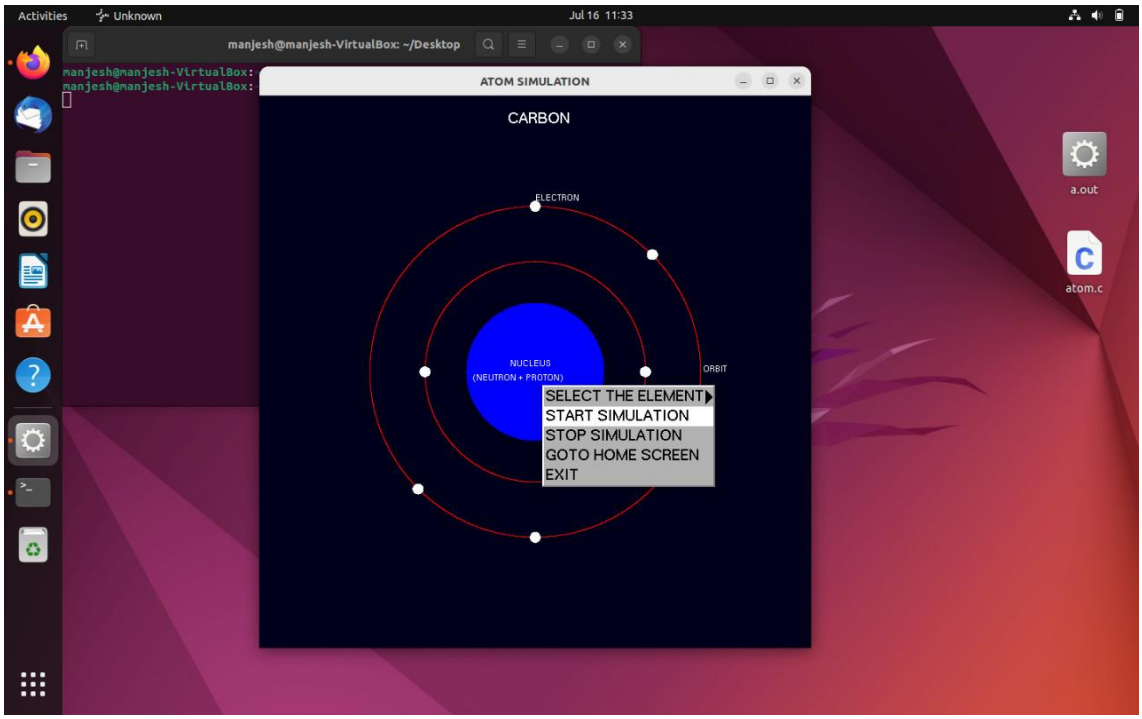

Figure 4.4: Snapshot of Atom Interface

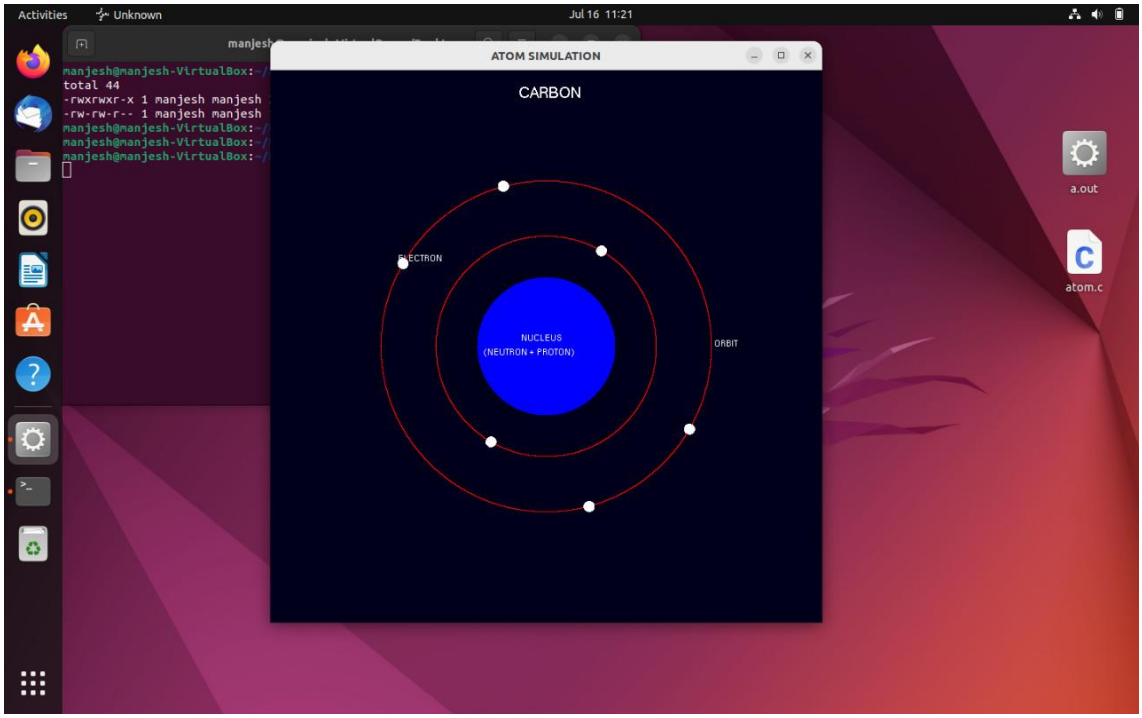

Figure 4.5: Snapshot of Atomic Simulation

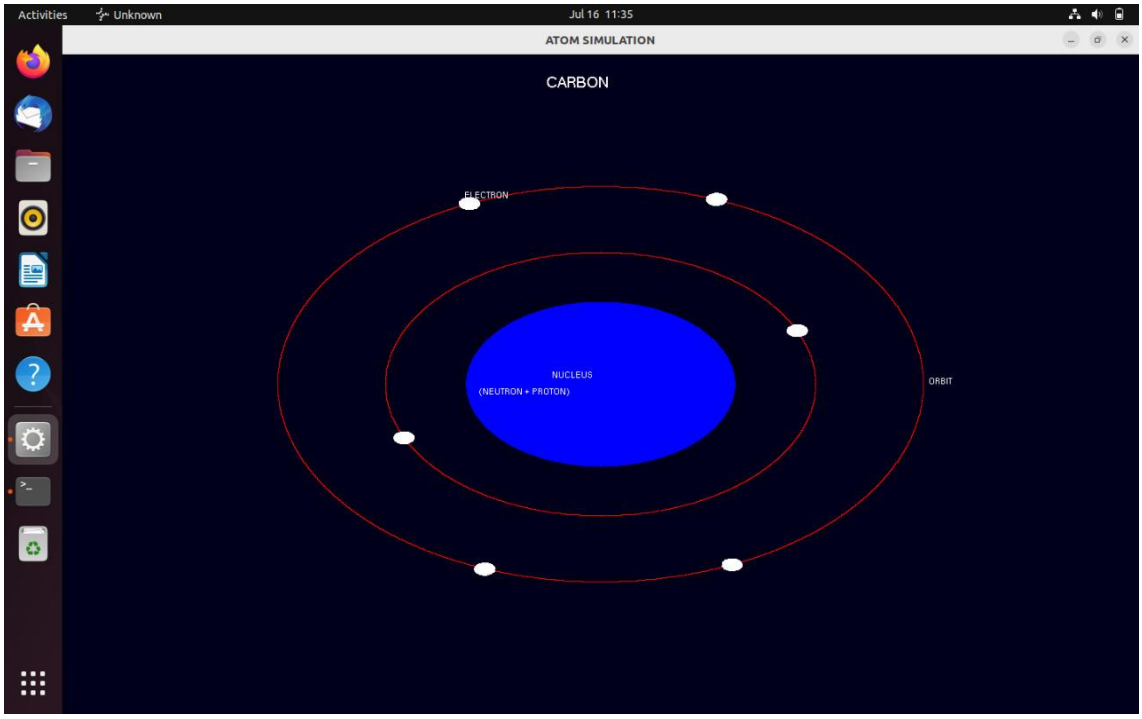

Figure 4.6: Snapshot of Full Atomic Simulation

CHAPTER 5
CONCLUSION

CHAPTER 5

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.

CHAPTER 6

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 Hearnand
[5] Pauline Baker, 2nd Edition, Pearson Education, 2003.
[6] www.google.com
[7] www.openglforum.org