VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Belagavi-590018, Karnataka

A

MINI PROJECT REPORT

ON

"WALKING ROBOT"

Submitted in partial fulfillment for the award of the degree in **Bachelor of Engineering in Computer Science & Engineering**

Submitted by

MOHAMMED ANEES AHMED	**[1BH20CS020]**
MANJOT SINGH	**[1BH20CS018]**
AMRITESH SINGH	**[1BH20CS005]**

<u>Under the Guidance of</u>

Mrs. MONICA C
Assistant Professor
Department of Computer Science and Engineering

BANGALORE TECHNOLOGICAL INSTITUTE
(ISO 9001:2015 Certified Institute)
Sarjapur Road, Kodathi, Bangalore-560035
Department of Computer Science & Engineering
2022-2023

	BANGALORE TECHNOLOGICAL INSTITUTE **Bengaluru-35** **(An ISO 9001:2015 Certified Institute)** **Department of Computer Science and Engineering**	

CERTIFICATE

This is to certify that the Mini project report on "**WALKING ROBOT**" carried out by **MOHAMMED ANEES AHMED [1BH19CS0020**], **MANJOT SINGH [1BH20CS017], AMRITESH SINGH [1BH20CS005]** the Bonafide students of **Bangalore Technological Institute**, **Bangalore** in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of **Visvesvaraya Technological University, Belagavi** during the year **2022-2023**. Thus, it is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report. The seminar report has been approved, as it satisfies the academic requirement in the respect of the seminar report prescribed for the said degree.

Mrs. Monica C Assistant Professor Department of CSE	**Dr. Sohan Kumar Gupta** H.O.D, Department of CSE

External Viva

Name of the Examiners	**Signature with date**
1……………………….	……………………….
2……………………….	……………………

BANGALORE TECHNOLOGICAL INSTITUTE
Bengaluru-35
(An ISO 9001:2015 Certified Institute)
Department of Computer Science and Engineering

DECLARATION

We, students of sixth semester **B.E, COMPUTER SCIENCE AND ENGINEERING, BANGALORE TECHNOLOGICAL INSTITUTE, BANGALORE,** hereby declare that the Mini project on "**WALKING ROBOT**" has been independently carried out by me at **Bangalore Technological Institute, Bengaluru** and submitted in partial fulfillment of the requirements for the award of the degree in **Bachelor of Engineering in Computer Science & Engineering** of the **Visvesvaraya Technological University, Belagavi** during the academic year **2022-2023**. We also declare that, to the best of my knowledge and believe the work reported here does not form or part of any other dissertation on the basis of which a degree or award was conferred on an early occasion of this by any other students.

PLACE: BENGALURU

DATE:

MOHAMMED ANEES AHMED
[1BH19CS020]
MANJOT SINGH
[1BH20CS018]
AMRITESH SINGH
[1BH20CS005]

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals and elders. I would like to take this opportunity to thank them all.

We heartily extend my words of gratitude to our Mini project Coordinator , **Mrs. Monica C**, for her valuable advice, encouragement and suggestion given to me in the course of my Mini project work. We convey gratitude to her for having constantly monitored the development of the seminar report and setting up precise deadlines.

We would like to express my immense gratitude to the Head of Department **Dr. Sohan Kumar Gupta**, for his unfailing encouragement and suggestions given to me in the course of the work.

We would like to take this opportunity to express my gratitude to the Principal,**Dr. H S Nanda**, for giving me this opportunity to enrich knowledge. We are grateful to the President **Dr. A Prabhakara Reddy** and Secretary, **Sri. C L Gowda** for having provided us with a great infrastructure and well-furnished labs.

Finally, a note of thanks to the Department of Computer Science and Engineering, bothteaching and non-teaching staff for their cooperation extended.

Last but not the least, We acknowledge the support and feedback of my parents, guardians and friends, for their indispensable help always.

MOHAMMED ANEES AHMED

[1BH19CS020]

MANJOT SINGH

[1BH20CS018]

AMRITESH SINGH

[1BH20CS005]

ABSTRACT

This abstract presents the design and implementation of a walking robot utilizing the OpenGL graphics library. The goal of this project is to develop a visually appealing and realistic simulation of a walking robot, demonstrating its locomotion capabilities through a dynamic and interactive graphical environment. The walking robot is designed to mimic the movements and behavior of a human-like bipedal creature. The implementation involves the integration of OpenGL, a powerful and versatile graphics library, to render a three-dimensional environment and animate the robot's locomotion. The robot's model includes articulated limbs, joints, and sensors, allowing for complex and lifelike movements. To achieve realistic walking behavior, advanced techniques such as inverse kinematics and motion planning are employed.

CONTENTS

Chapter No.	Title	Page No.
	Abstract	**I**
	List of figures	**iv**
1	**Introduction**	**1-3**
	1.1 Computer Graphics	
2	**Literature Survey**	**4-7**
	2.1 Open GL Functions Technology	
	2.2 Open GL Fundamentals	
	2.3 Project Definition	
3	**System Requirement Specification**	**8**
	3.1 Minimum Hardware Requirements	
	3.2 Minimum Software Requirements	
4	**Design**	**9-10**
	4.1 System Design	
5	**Implementation**	**11-63**
	5.1 Introduction to visual studio	
	5.2 Algorithm	
	5.3 Open GL Functions	

	5.4 Source Code	
6	**Output**	**64-67**
	6.1 Snap Shots	
7	**Conclusion and Future Enhancement**	**68**
8	**Bibliography**	**69**

LIST OF FIGURES

Fig. No.	Title	Page No.
1.1	Graphics System	2
2.1	Graphics Pipeline	5
4.1	Data flow chart	10
6.1	Home page	64
6.2	Walking screen	65
6.3	Initial Screen	66
6.4	Frame	67
6.5	Instruction Menu	67

CHAPTER 1 INTRODUCTION

Computer graphics is concerned with all aspects of producing picture or an image using computers and, more generally, the representation and manipulation of pictorial data by a computer. The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Developments in computer graphic had a profound impact on many types of media and have revolutionized the animation and video game industry. Today computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. Such graphs are used to illustrate papers, reports, theses, and other presentation material. A range of tools and facilities are available to enable users to visualize their data, and computer graphics are used in many disciplines. We implement computer graphics using the OpenGL API. OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics.

OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn). Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), the OpenGL Utility Toolkit (GLUT) has been created to aid in the development of more complicated three-dimensional objects such as a sphere, a torus, and even a teapot. GLUT may not be satisfactory for full- featured OpenGL applications, but it is a useful starting point for learning OpenGL.GLUT is designed to fill the need for a window system independent programming

Interface for OpenGL programs. The interface is designed to be simple yet still meet the needs of useful OpenGL programs. Removing window system operations from OpenGL is a sound decision because it allows the OpenGL graphics system to be retargeted to various Systems including powerful but expensive graphics workstations as well as mass-production graphics systems like video games, set-top boxes for interactive television, and PCs.

The GLUT application-programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT routines also take relatively few parameters.

1.1COMPUTER GRAPHICS

The term computer graphics includes almost everything on computers that is not text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. The term Computer Graphics has several meanings The representation

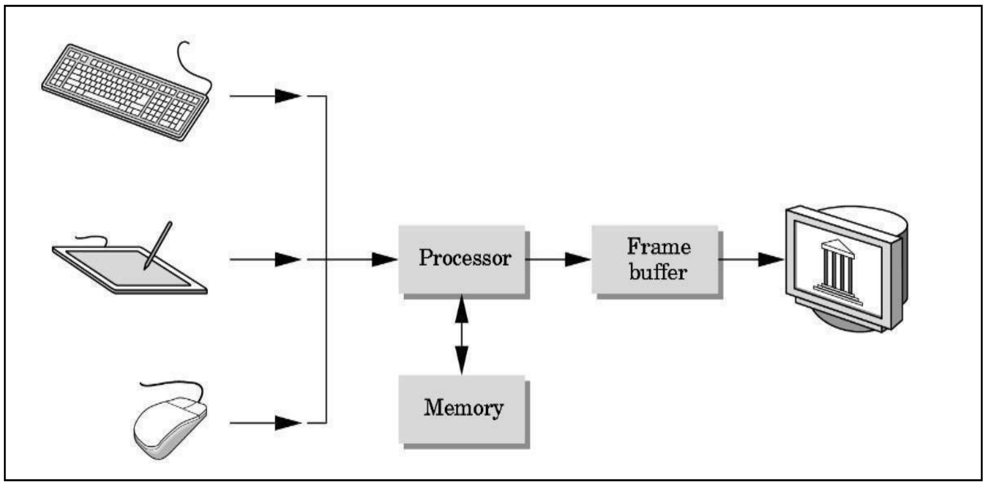

Fig 1.1 A Graphics System

and manipulation of pictorial data by a computer The various technologies used to create and manipulate such pictorial data The images so produced, and The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content Today computers and computer-generated images touch many aspects of our daily life.

Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpretation fig 1.1 inputs go to processor from there to frame buffer and then it displays the pixel. It illustrates the graphics system. A graphics system is just the graphics part of the software. that sits in between the hardware and application programs.

CHAPTER 2

LITERATURE SURVEY

2.1 OPENGL TECHNOLOGY

OpenGL is strictly defined as “a software interface to graphics hardware.” In essence, it is a 3D graphics and modelling library that is highly portable and very fast. Using OpenGL, you can create elegant and beautiful 3D graphics with exceptional visual quality. The greatest advantage to using OpenGL is that it is orders of magnitude faster than a ray tracer or software- rendering engine. Initially, it used algorithms carefully developed and optimized by Silicon Graphics, Inc. (SGI), an acknowledged world leader in computer graphics and animation.

1. Vertex processing: Much of the work in the pipeline is in converting object representations from one co-ordinate system to another. Every vertex is processed independently.

The co-ordinate system include the following:

- Object co-ordinates
- Camera co-ordinates
- Screen co-ordinates.

2. Clipping and Primitive assembly: clipping is necessary, because of the limitation that no imagining system can see the whole object at once. Cameras have film of limited size. Image outside the clipping volume is clipped out. Clipping is done by vertex basis.
3. Rasterization: generation of pixels in the frame buffer. Rasterizer determines which pixels in the frame buffer are inside the polygon. The output of the Rasterizer is a set of fragments for each primitive. Fragments carry color, location, depth and other information.
4. Fragment processing: Updates the pixel in the frame buffer. Color of the pixels that correspond to fragment can be read from the frame buffer. Color fragment may be altered by texture mapping.

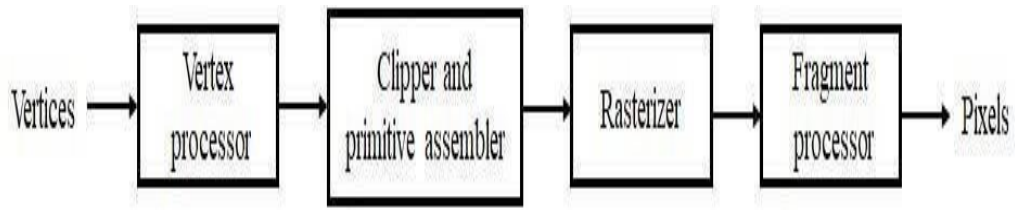

Fig 2.1 Graphics pipeline.

A computer graphics pipeline, rendering pipeline, is a conceptual model that describes what steps a graphics system needs to perform to render a 3D scene to a 2D screen. The graphics pipeline as four stages. These stages include vertex processing, clipping, Rasterization .

2.2 OPENGL FUNDAMENTALS

The section explains some of the concepts inherent in OpenGL. Primitives and Commands OpenGL draws primitives-points, line segments, or polygons-subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer).Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normal, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be clipped so that a particular primitive fits within a specified region; in this case, vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents. Commands are always processed in the order, in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state- querying commands return data that's consistent with complete execution of all previously issued OpenGL commands. Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects. (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives.) Arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene. Calculate the color of all the objects. The color might be explicitly assigned by the application, determined from specified lighting conditions, obtained by pastinga texture onto the objects, or some combination of these three actions. Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called rasterization. OpenGL is a standard specification that defines an API that is multi-language and multi-platform and that enables the codification of applications that output computerized graphics in 2D and 3 The interface consists in more than 250 different functions, which can be used to draw complex tridimensional scenes with simple primitives. It consists of many functions that help to create a real world object and a particular existence for an object can be given.

2.3 PROJECT DEFINITION

The developed walking robot simulation serves as a valuable tool for various applications, including robotics research, education, and entertainment. Researchers can utilize the simulation to study locomotion control algorithms, test different gait patterns, and explore new strategies for robot navigation and balance. Educators can incorporate the simulation into robotics courses to enhance students' understanding of bipedal locomotion and kinematics. Furthermore, the visually appealing nature of the simulation can captivate and entertain a broader audience, fostering interest and curiosity in the field of robotics.

Keys used

Use ← ↑→ ↓ keys to rotate the angle of your screen to see the robot walking from different direction. And to exit the walking screen you need to just press the escape button on your keyboard.

Use the right click of the mouse to start and also stop the walking robot, you can also change the structure of the robot.

2.4 ADVANTAGES

1. User-friendly.
2. Simple and interactive.
3. Multiple actions can be performed at once.
4. OpenGL is more functional than any other API
5. Reliability and Portability.

CHAPTER 3

SYSTEM REQUIREMENTS SPECIFICATION

3.1 Minimum hardware requirements

- **Microprocessor:** 1.0 GHz and above CPU based on either AMD or INTELMicroprocessor Architecture.
- **Main memory:** 512 MB RAM.
- **Hard Disk :** 100MB.
- **Hard disk speed in RPM:** 5400 RPM.
- **Keyboard:** QWERTY Keyboard.
- **Mouse:** 2 or 3 Button mouse.
- **Monitor:** 1024 x 768 display resolution.

3.2 Minimum software requirements

- Operating system: Windows XP and later.
- Visual C++ 2005 and later.
- OPENGL Library.
- X86.
- X64(WOW).
- Mouse Driver.
- Graphics Driver.
- C Libraries.

CHAPTER 4

DESIGN

4.1 SYSTEM DESIGN

The system design for a walking robot using OpenGL involves several key components and considerations to ensure a robust and efficient implementation. Here's a brief overview of the key aspects of the system design:

- **Robot Model:** Designing the 3D model of the walking robot is an essential step. It includes creating the geometry and structure of the robot's body, limbs, joints, and sensors. The model should accurately represent the physical characteristics and movements of a bipedal creature.
- **Animation and Kinematics:** Implementing animation and kinematics involves defining the joint angles and trajectories required for the robot's walking motion. Techniques such as inverse kinematics can be employed to calculate the joint angles based on desired foot positions and other constraints. Smooth and natural animations can be achieved through careful interpolation and blending techniques.
- **Physics Simulation:** Integrating a physics simulation enables the walking robot to interact with the virtual environment realistically. Physics-based algorithms can simulate gravity, friction, ground contact forces, and collisions, ensuring the robot's stability and adherence to physical constraints. This enhances the believability of the walking motion.
- **OpenGL Rendering:** OpenGL serves as the graphics rendering framework for creating a visually appealing and interactive environment for the walking robot. It handles rendering tasks such as drawing 3D objects, applying textures, handling lighting and shading effects, and managing the overall visual presentation.
- **User Interaction:** Consider incorporating user interaction mechanisms to allow users to control and observe the walking robot. This may involve implementing user interfaces, input devices (e.g., keyboard, mouse), and camera controls to provide an immersive and interactive experience. Users should have the ability to manipulate the robot's motion, viewpoint, and other parameters.

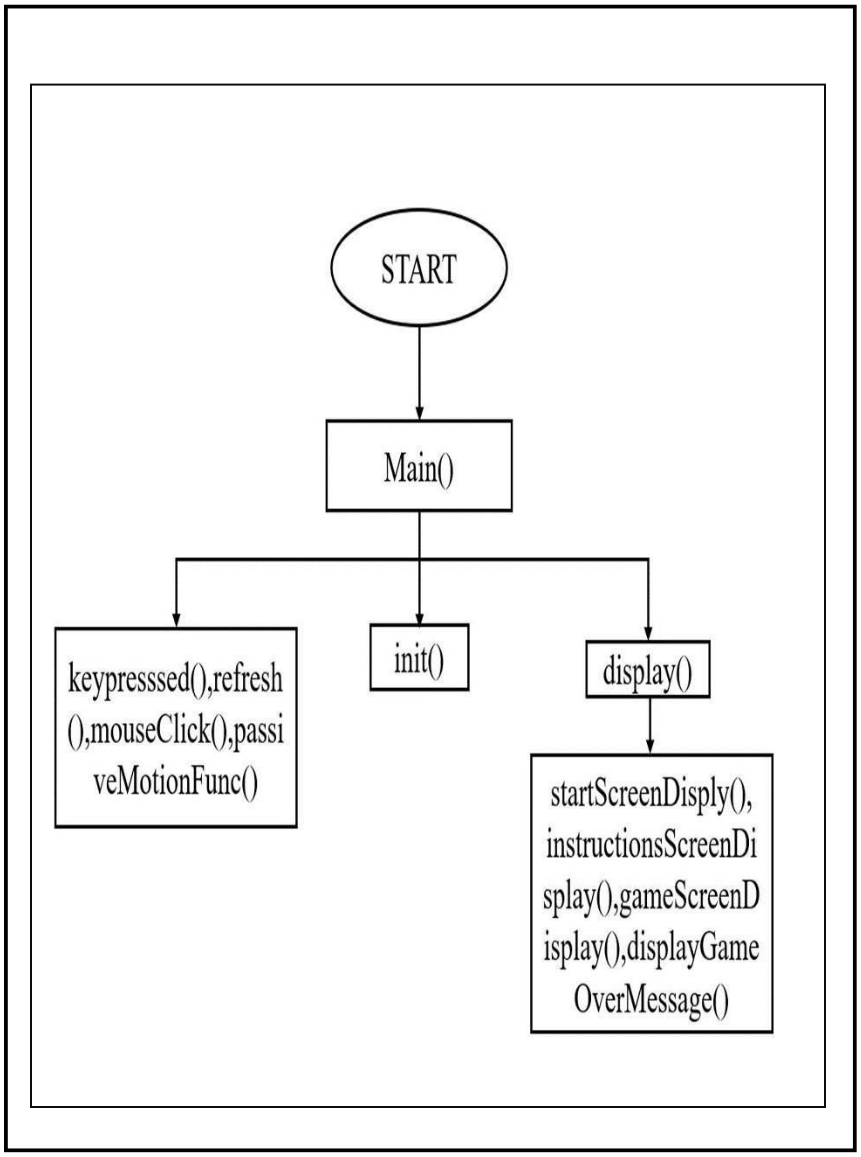
START
Main()
keypresssed(),refresh(),mouseClick(),passiveMotionFunc()
init()
display()
startScreenDisply(), instructionsScreenDisplay(),gameScreenDisplay(),displayGameOverMessage()

Fig 4.1 DataFlow Diagram.

A data flow diagram (DFD) is a graphical representation that illustrates how data flows through a system or process. It visually depicts the movement of data from one component to another, showing the inputs, outputs, processes, and storage involved in a system.

CHAPTER 5

IMPLEMENTATION

5.1 INTRODUCTION TO VISUAL STUDIO

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows super family of operating systems, as well as websites, web applications and web services. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source- level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for source-control systems (like Subversion) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Team Foundation Server client: Team Explorer).

Visual Studio supports different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++ and C++/CLI (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), and F# (as of Visual Studio 2010).Support for other languages such as M, Python, and Ruby among others is available via language services installed separately.

5.2 ALGORITHM

1. Initialize the display mode ;
2. Initialize the window position and size;
3. Create display window;
4. Initialize the keyboard function;
5. Initialize mouse function;
6. Call display function ;

5.3 OPENGL FUNCTIONS

The functions used in the programs are as follows:

- **main( ):** The control will first enter into this function and in this function it will create a window and it will initialize the window and then this function calls display function.
- **glutInit() :** interaction between the windowing system and OPENGL isinitiated
- **glutInitDisplayMode() :** used when double buffering is required anddepthinformation isrequired
- **glutCreateWindow() :** this opens the OPENGL window and displays thetitle at topof thewindow
- **glutInitWindowSize() :** specifies the size of the window
- **glutInitWindowPosition() :** specifies the position of the window in screenco-ordinates
- **glutKeyboardFunc() :** handles normal ASCII symbols
- **glutSpecialFunc() :** handles special keyboard keys
- **glutReshapeFunc() :** sets up the callback function for reshaping thewindow
- **glutIdleFunc() :** this handles the processing of the background
- **glutDisplayFunc() :** this handles redrawing of the window
- **glutMainLoop() :** this starts the main loop, it never returns

- **glViewport() :** used to set up the viewport
- **glVertex3fv() :** used to set up the points or vertices in three dimensions
- **glColor3fv() :** used to render color to faces
- **glFlush() :** used to flush the pipeline
- **glutPostRedisplay() :** used to trigger an automatic redrawal of the object
- **glMatrixMode() :** used to set up the required mode of the matrix
- **glLoadIdentity() :** used to load or initialize to the identity matrix

5.4 SOURCE CODE

```
#define SPHERE
#define COLOR
#define LIGHT
#define TORSO
#define HIP
#define SHOULDER
#define UPPER_ARM
#define LOWER_ARM
#define ROCKET_POD
#define UPPER_LEG
#define LOWER_LEG
#define NO_NORM
#define ANIMATION
#define DRAW_MECH
#define DRAW_ENVIRO
#define MOVE_LIGHT
#include <stdlib.h>
#include <math.h>
#define GLUT
#define GLUT_KEY
#define GLUT_SPEC
#include <GL/glut.h>
/* end of header files */
#define TEXTID
void DrawTextXY(double,double,double,double,char *);
#define SOLID_MECH_TORSO
```

```
#define SOLID_MECH_HIP

#define SOLID_MECH_SHOULDER

#define SOLID_MECH_UPPER_ARM

#define SOLID_MECH_FOREARM

#define SOLID_MECH_UPPER_LEG

#define SOLID_MECH_FOOT

#define SOLID_MECH_ROCKET

#define SOLID_MECH_VULCAN

#define SOLID_ENVIRO

#ifndef M_PI

#define M_PI 3.14

#endif

GLUquadricObj *qobj;

char leg = 0;

int shoulder1 = 0, shoulder2 = 0, shoulder3 = 0, shoulder4 = 0, lat1 = 20, lat2 = 20,

  elbow1 = 0, elbow2 = 0, pivot = 0, tilt = 10, ankle1 = 0, ankle2 = 0, heel1 = 0,

  heel2 = 0, hip11 = 0, hip12 = 10, hip21 = 0, hip22 = 10, fire = 0, solid_part = 0,

  anim = 0, turn = 0, turn1 = 0, lightturn = 0, lightturn1 = 0;

float elevation = 0.0, distance = 0.0, frame = 3.0

#ifdef LIGHT        // to change the color of robot box

GLfloat mat_specular[] = {0.0, 0.0, 1.0, 1.0};

GLfloat mat_ambient[] ={0.0, 0.0, 1.0, 1.0};

GLfloat mat_diffuse[] ={0.0, 0.0, 1.0, 1.0};

GLfloat mat_shininess[] ={128.0 * 0.4};

GLfloat mat_specular2[] ={0.508273, 0.508273, 0.508373};

GLfloat mat_ambient2[] ={0.19225, 0.19225, 0.19225};

GLfloat mat_diffuse2[] ={0.50754, 0.50754, 0.50754};

GLfloat mat_shininess2[] ={128.0 * 0.6};
```

```
//to change the wall colorfffffffff

GLfloat mat_specular3[] = {1.0, 1.0, 0.0};

GLfloat mat_ambient3[] ={1.0, 1.0, 0.0};

GLfloat mat_diffuse3[] ={1.0, 1.0, 0.0};

GLfloat mat_shininess3[] ={0.0 * 0.0};

//to change the plateform color

GLfloat mat_specular4[] = {0.633, 0.727811, 0.633};

GLfloat mat_ambient4[] = {0.0215, 0.1745, 0.0215};

GLfloat mat_diffuse4[] = {0.07568, 0.61424, 0.07568};

GLfloat mat_shininess4[] = {128 * 0.6};

GLfloat mat_specular5[] =

{0.60, 0.60, 0.50};

GLfloat mat_ambient5[] =

{0.0, 0.0, 0.0};

GLfloat mat_diffuse5[] =

{0.5, 0.5, 0.0};

GLfloat mat_shininess5[] =

{128.0 * 0.25};

#endif

/* end of material definitions */

/* start of the body motion functions */

void Heel1Add(void){

  heel1 = (heel1 + 3) % 360;}

void Heel1Subtract(void){

  heel1 = (heel1 - 3) % 360;

}

void DrawTextXY(double x,double y,double z,double scale,char *s){

   int i;
```

```
    glPushMatrix();
    glTranslatef(x,y,z);
    glScalef(scale,scale,scale);
    for (i=0;i<strlen(s);i++)
   glutStrokeCharacter(GLUT_STROKE_ROMAN,s[i]);
 glPopMatrix();
}
void Box(float width, float height, float depth, char solid){
  char i, j = 0;
  float x = width / 2.0, y = height / 2.0, z = depth / 2.0;
  for (i = 0; i < 4; i++) {
    glRotatef(90.0, 0.0, 0.0, 1.0);
    if (j) {
      if (!solid)
        glBegin(GL_LINE_LOOP);
      else
        glBegin(GL_QUADS);
      glNormal3f(-1.0, 0.0, 0.0);
      glVertex3f(-x, y, z);
      glVertex3f(-x, -y, z);
      glVertex3f(-x, -y, -z);
      glVertex3f(-x, y, -z);
      glEnd();
      if (solid) {
        glBegin(GL_TRIANGLES);
        glNormal3f(0.0, 0.0, 1.0);
        glVertex3f(0.0, 0.0, z);
        glVertex3f(-x, y, z);
        glVertex3f(-x, -y, z);
        glNormal3f(0.0, 0.0, -1.0);
        glVertex3f(0.0, 0.0, -z);
        glVertex3f(-x, -y, -z);
        glVertex3f(-x, y, -z);
        glEnd();
      }
      j = 0;
    } else {
      if (!solid)
        glBegin(GL_LINE_LOOP);
      else
        glBegin(GL_QUADS);
      glNormal3f(-1.0, 0.0, 0.0);
      glVertex3f(-y, x, z);
      glVertex3f(-y, -x, z);
      glVertex3f(-y, -x, -z);
      glVertex3f(-y, x, -z);
      glEnd();
      if (solid) {
        glBegin(GL_TRIANGLES);
        glNormal3f(0.0, 0.0, 1.0);
        glVertex3f(0.0, 0.0, z);
        glVertex3f(-y, x, z);
        glVertex3f(-y, -x, z);
        glNormal3f(0.0, 0.0, -1.0);
        glVertex3f(0.0, 0.0, -z);
        glVertex3f(-y, -x, -z);
```

```
        glVertex3f(-y, x, -z);
        glEnd();}
      j = 1;
    }}}
    glRotatef(45.0, 0.0, 0.0, 1.0);
  }
}
#ifdef NORM
void Normalize(float v[3]){
  GLfloat d = sqrt(v[1] * v[1] + v[2] * v[2] + v[3] * v[3]);
  if (d == 0.0) {
    printf("zero length vector");
    return;
  }
  v[1] /= d;
  v[2] /= d;
  v[3] /= d;
}
void NormXprod(float v1[3], float v2[3], float v[3], float out[3]){
  GLint i, j;
  GLfloat length;
  out[0] = v1[1] * v2[2] - v1[2] * v2[1];
  out[1] = v1[2] * v2[0] - v1[0] * v2[2];
  out[2] = v1[0] * v2[1] - v1[1] * v2[0];
  Normalize(out);
}
#endif
void SetMaterial(GLfloat spec[], GLfloat amb[], GLfloat diff[], GLfloat shin[]){
  glMaterialfv(GL_FRONT, GL_SPECULAR, spec);
  glMaterialfv(GL_FRONT, GL_SHININESS, shin);
  glMaterialfv(GL_FRONT, GL_AMBIENT, amb);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, diff);
 }
void MechTorso(char solid){
  glNewList(SOLID_MECH_TORSO, GL_COMPILE);
#ifdef LIGHT
  SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);
void MechHip(char solid){
  int i;
  glNewList(SOLID_MECH_HIP, GL_COMPILE);
#ifdef LIGHT
  SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);
#endif
  glColor3f(0.0, 1.0, 0.0);//hip lines form green
  Octagon(0.7, 0.5, solid);
#ifdef SPHERE
  for (i = 0; i < 2; i++) {
    if (i)
      glScalef(-1.0, 1.0, 1.0);
    glTranslatef(1.0, 0.0, 0.0);
#ifdef LIGHT
    SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
#endif
    glColor3f(0.0, 1.0, 0.0);//hip line form green
    if (!solid)
      gluQuadricDrawStyle(qobj, GLU_LINE);
    gluSphere(qobj, 0.2, 16, 16);

    glTranslatef(-1.0, 0.0, 0.0);

  }

  glScalef(-1.0, 1.0, 1.0);

#endif

  glEndList();

}

void Shoulder(char solid){

  glNewList(SOLID_MECH_SHOULDER, GL_COMPILE);

#ifdef LIGHT

  SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);

#endif

  glColor3f(0.0, 1.0, 0.0);//sholder color green

  Box(1.0, 0.5, 0.5, solid);

  glTranslatef(0.9, 0.0, 0.0);

#ifdef LIGHT

  SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);

#endif

  glColor3f(0.0, 1.0, 0.0);// sholder color green

#ifdef SPHERE

  if (!solid)

    gluQuadricDrawStyle(qobj, GLU_LINE);

  gluSphere(qobj, 0.6, 16, 16);

#endif

  glTranslatef(-0.9, 0.0, 0.0);

  glEndList();

}

void UpperArm(char solid){
  int i;

  glNewList(SOLID_MECH_UPPER_ARM, GL_COMPILE);

#ifdef LIGHT

  SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);

#endif

  glColor3f(1.0, 0.0, 0.0);//arm red

  Box(1.0, 2.0, 1.0, solid);

  glTranslatef(0.0, -0.95, 0.0);

  glRotatef(90.0, 1.0, 0.0, 0.0);

  glTranslatef(0.0, -0.15, -0.4);

  Box(2.0, 0.5, 2.0, solid);

  glTranslatef(0.0, -0.3, -0.2);

  glRotatef(90.0, 1.0, 0.0, 0.0);

#ifdef LIGHT

  SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);

#endif

  glColor3f(0.5, 0.5, 0.5);//leg joints grey

  gluCylinder(qobj, 0.6, 0.6, 3.0, 16, 10);

#ifdef LIGHT

  SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);

#endif

  glColor3f(0.0, 1.0, 0.0);//above the leg joint n below the fore leg

  glRotatef(-90.0, 1.0, 0.0, 0.0);

  glTranslatef(0.0, -1.5, 1.0);

  Box(1.5, 3.0, 0.5, solid);

  glTranslatef(0.0, -1.75, -0.8);

  Box(2.0, 0.5, 2.0, solid);

  glTranslatef(0.0, -0.9, -0.85);
```

```
#ifdef LIGHT
  SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
#endif
  glColor3f(1.0, 0.0, 0.0);//leg joints between fore leg and leg
  gluCylinder(qobj, 0.8, 0.8, 1.8, 16, 10);
  for (i = 0; i < 2; i++) {
    if (i)
      glScalef(-1.0, 1.0, 1.0);
    if (!solid)
      gluQuadricDrawStyle(qobj, GLU_LINE);
    if (i)
      glTranslatef(0.0, 0.0, 1.8);
    gluDisk(qobj, 0.0, 0.8, 16, 10);
    if (i)
      glTranslatef(0.0, 0.0, -1.8);
  }
  glScalef(-1.0, 1.0, 1.0);
  glEndList();
}
void Foot(char solid){
  glNewList(SOLID_MECH_FOOT, GL_COMPILE);
#ifdef LIGHT
  SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
#endif
  glColor3f(1.0, 0.0, 0.0);// color foot
  glRotatef(90.0, 1.0, 0.0, 0.0);
  Octagon(1.5, 0.6, solid);
  glRotatef(-90.0, 1.0, 0.0, 0.0);
  glEndList();}
void LowerLeg(char solid){
  float k, l;
#ifdef LIGHT
  SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);
#endif
  glColor3f(1.0, 0.0, 0.0);//leg joint
  for (k = 0.0; k < 2.0; k++) {
    for (l = 0.0; l < 2.0; l++) {
      glPushMatrix();
      glTranslatef(k, 0.0, l);
    glScalef(-1.0, 1.0, 1.0);
    glTranslatef(0.0, 0.0, 5.0);
  }
  glEndList();
}
void Toggle(void){
  if (solid_part)
    solid_part = 0;
  else
    solid_part = 1;
}
void disable(void){
  glDisable(GL_LIGHTING);
  glDisable(GL_DEPTH_TEST);
  glDisable(GL_NORMALIZE);
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
}
```

```
void lighting(void){

  GLfloat position[] =

  {0.0, 0.0, 2.0, 1.0};

#ifdef MOVE_LIGHT

  glRotatef((GLfloat) lightturn1, 1.0, 0.0, 0.0);

  glRotatef((GLfloat) lightturn, 0.0, 1.0, 0.0);

  glRotatef(0.0, 1.0, 0.0, 0.0);

#endif

  glEnable(GL_LIGHTING);

  glEnable(GL_LIGHT0);

  glEnable(GL_NORMALIZE);

  glDepthFunc(GL_LESS);

  glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

  glLightfv(GL_LIGHT0, GL_POSITION, position);

  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 80.0);

  glTranslatef(0.0, 0.0, 2.0);

  glDisable(GL_LIGHTING);

  Box(0.1, 0.1, 0.1, 0);

  glEnable(GL_LIGHTING);

}

void DrawMech(void){  int i, j;

  glScalef(0.5, 0.5, 0.5);

  glPushMatrix();

  glTranslatef(0.0, -0.75, 0.0);

  glRotatef((GLfloat) tilt, 1.0, 0.0, 0.0);

  glRotatef(90.0, 1.0, 0.0, 0.0);

#ifdef HIP

  glCallList(SOLID_MECH_HIP);
```

```
#endif

  glRotatef(-90.0, 1.0, 0.0, 0.0);

  glTranslatef(0.0, 0.75, 0.0);

  glPushMatrix();

  glRotatef((GLfloat) pivot, 0.0, 1.0, 0.0);

  glPushMatrix();

#ifdef TORSO

  glCallList(SOLID_MECH_TORSO);

#endif

  glPopMatrix();

  glPushMatrix();

  glTranslatef(0.5, 0.5, 0.0);

#ifdef ROCKET_POD

  glCallList(SOLID_MECH_ROCKET);

#endif

  glPopMatrix();

  for (i = 0; i < 2; i++) {

    glPushMatrix();

    if (i)

      glScalef(-1.0, 1.0, 1.0);

    glTranslatef(1.5, 0.0, 0.0);

#ifdef SHOULDER

    glCallList(SOLID_MECH_SHOULDER);

#endif

    glTranslatef(0.9, 0.0, 0.0);

    if (i) {

      glRotatef((GLfloat) lat1, 0.0, 0.0, 1.0);

      glRotatef((GLfloat) shoulder1, 1.0, 0.0, 0.0);
      glRotatef((GLfloat) shoulder3, 0.0, 1.0, 0.0);
    } else {
      glRotatef((GLfloat) lat2, 0.0, 0.0, 1.0);
      glRotatef((GLfloat) shoulder2, 1.0, 0.0, 0.0);
      glRotatef((GLfloat) shoulder4, 0.0, 1.0, 0.0);
    }
    glTranslatef(0.0, -1.4, 0.0);
#ifdef UPPER_ARM
    glCallList(SOLID_MECH_UPPER_ARM);
#endif
    glTranslatef(0.0, -2.9, 0.0);
    if (i)
      glRotatef((GLfloat) elbow1, 1.0, 0.0, 0.0);
    else
      glRotatef((GLfloat) elbow2, 1.0, 0.0, 0.0);
    glTranslatef(0.0, -0.9, -0.2);
#ifdef LOWER_ARM
    glCallList(SOLID_MECH_FOREARM);
    glPushMatrix();
    glTranslatef(0.0, 0.0, 2.0);
    glRotatef((GLfloat) fire, 0.0, 0.0, 1.0);
    glCallList(SOLID_MECH_VULCAN);
    glPopMatrix();
#endif
    glPopMatrix();
  }
  glPopMatrix();
  glPopMatrix();
```

```
   for (j = 0; j < 2; j++) {

 void myReshape(int w, int h){

   glViewport(0, 0, w, h);

   glMatrixMode(GL_PROJECTION);

   glLoadIdentity();

   gluPerspective(65.0, (GLfloat) w / (GLfloat) h, 1.0, 20.0);

   glMatrixMode(GL_MODELVIEW);

   if (step == 1 || step == 3) {

     if (frame <= 21.0 && frame >= 0.0) {

       angle = (180 / M_PI) * (acos(((cos((M_PI / 180) * frame) * 2.043) + 1.1625) / 3.2029));

       if (frame > 0)

         elevation = -(3.2055 - (cos((M_PI / 180) * angle) * 3.2055));

       else

         elevation = 0.0;

       if (step == 1) {

         elbow2 = hip11 = -frame;

         elbow1 = heel1 = frame;

         heel2 = 15;

         ankle1 = frame;

         if (frame > 0)

           hip21 = angle;

         else

           hip21 = 0;

         ankle2 = -hip21;

         shoulder1 = 1.5 * frame;

         shoulder2 = -frame * 1.5;

       } else {

         elbow1 = hip21 = -frame;
```

```
        elbow2 = heel2 = frame;
        heel1 = 15;
        ankle2 = frame;
        if (frame > 0)
          hip11 = angle;
        else
          hip11 = 0;
        ankle1 = -hip11;
        shoulder1 = -frame * 1.5;
        shoulder2 = frame * 1.5;
      }
      if (frame == 0.0)
        step++;
      if (frame > 0)
        frame = frame - 3.0;
    }
  }
  if (step == 4)
    step = 0;
  distance += 0.1678;
  glutPostRedisplay();
}
void animation(void)
{  animation_walk();
}#endif
#ifdef GLUT
#ifdef GLUT_KEY
/* ARGSUSED1 */
```

```
void keyboard(unsigned char key, int x, int y){
  int i = 0;
if (key == 27) exit (0);
#ifdef MOVE_LIGHT
  glutAddSubMenu("rotate the light source..", glut_menu[12]);
#endif
 glutCreateMenu(menu_select);
#ifdef ANIMATION
  glutAddMenuEntry("Start Walk", 1);
  glutAddMenuEntry("Stop Walk", 2);
#endif
  glutAddMenuEntry("Toggle Wireframe", 3);
  glutAddSubMenu("How do I ..", glut_menu[0]);
  glutAddMenuEntry("Quit", 4);
  glutAttachMenu(GLUT_LEFT_BUTTON);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
}
int main(int argc, char **argv){
#ifdef GLUT
  /* start of glut windowing and control functions */
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
  glutInitWindowSize(1000, 1000);
  glutCreateWindow("glutmech: Vulcan Gunner");
  myinit();
  glutDisplayFunc(display);
  glutReshapeFunc(myReshape);
#ifdef GLUT_KEY
```

```
  glutKeyboardFunc(keyboard);
#endif
#ifdef GLUT_SPEC
  glutSpecialFunc(special);
#endif
 glutMenu();
  glPointSize(2.0);
  glutMainLoop();
#endif
  return 0;}
```

CHAPTER 6

SAMPLE OUTPUT

6.1 SNAPSHOOTS

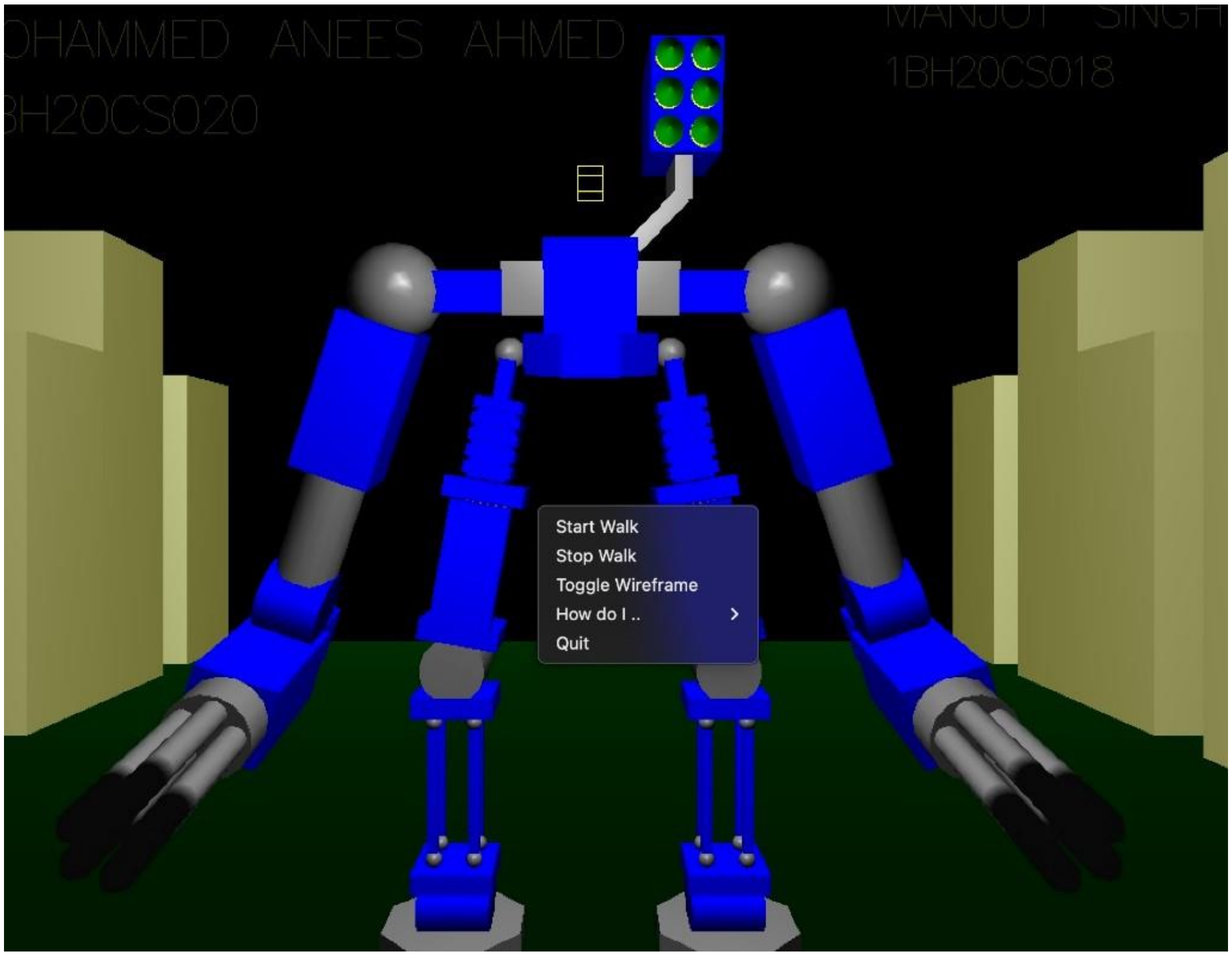

Fig 6.1 Home Page

Home screen of the game, where the user can start the game by clicking on the S/s button.

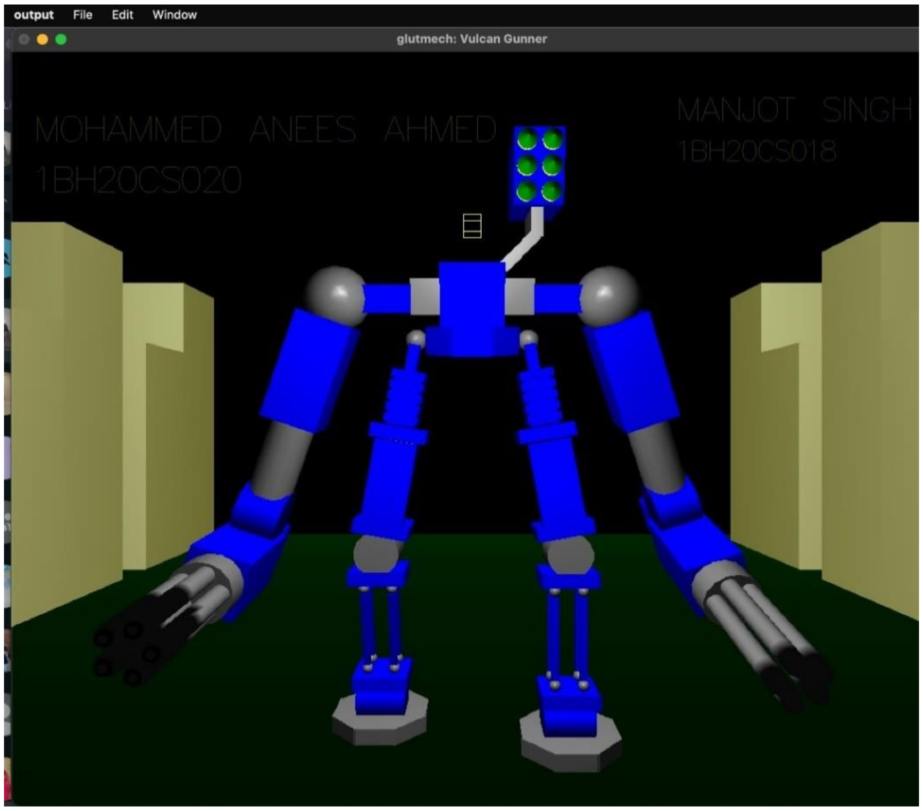

Fig 6.2 Walking Screen

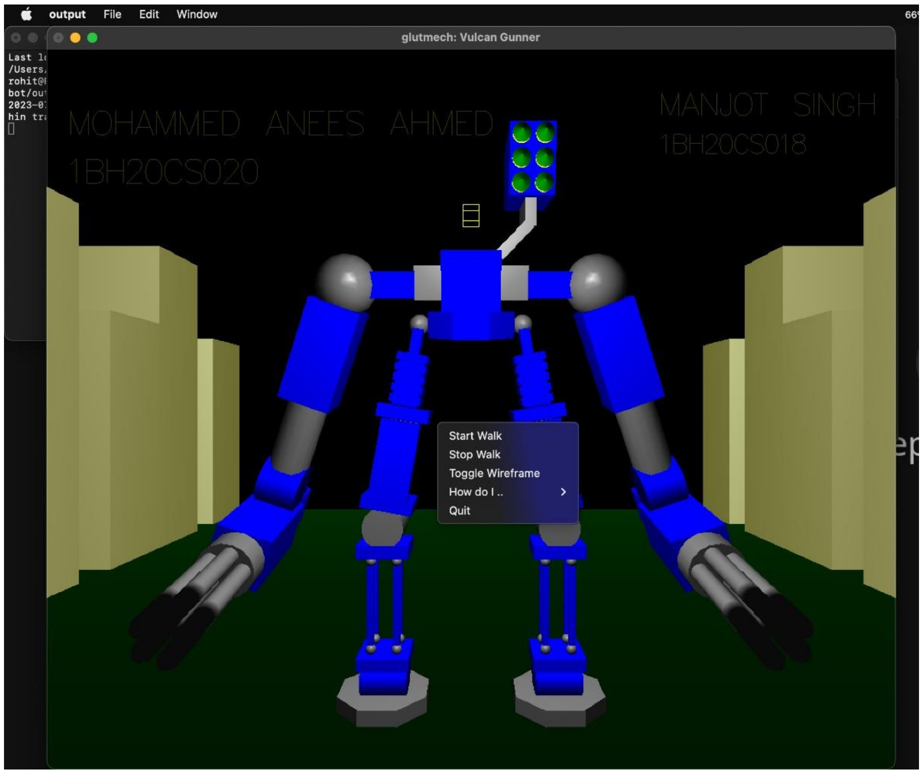

Fig 6.3 Initial screen

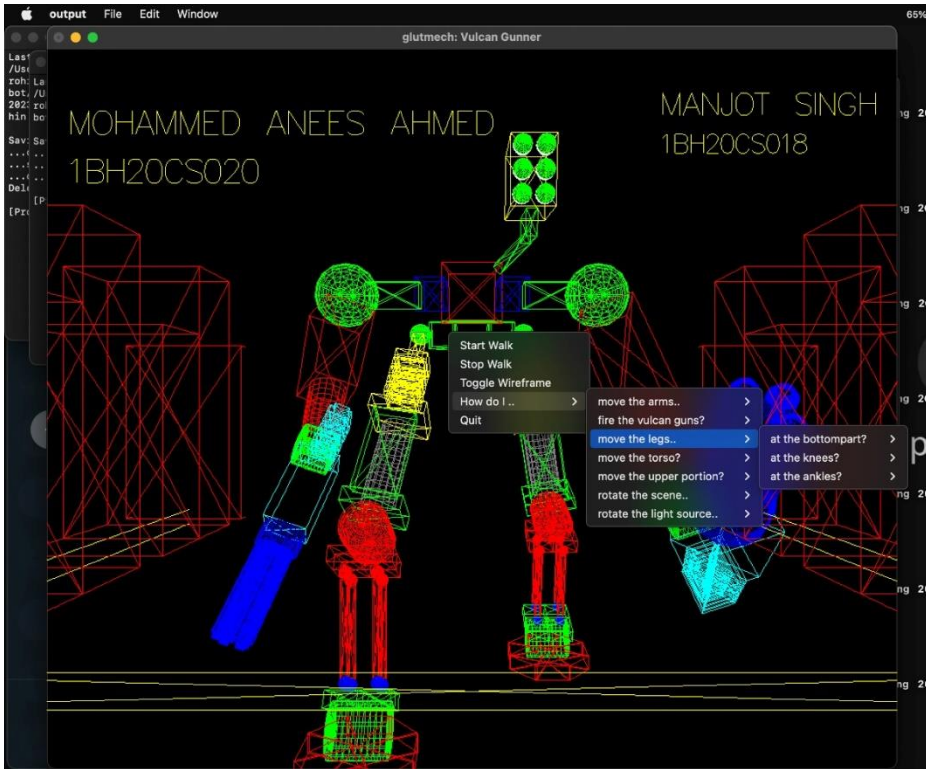

Fig 6.4 frame

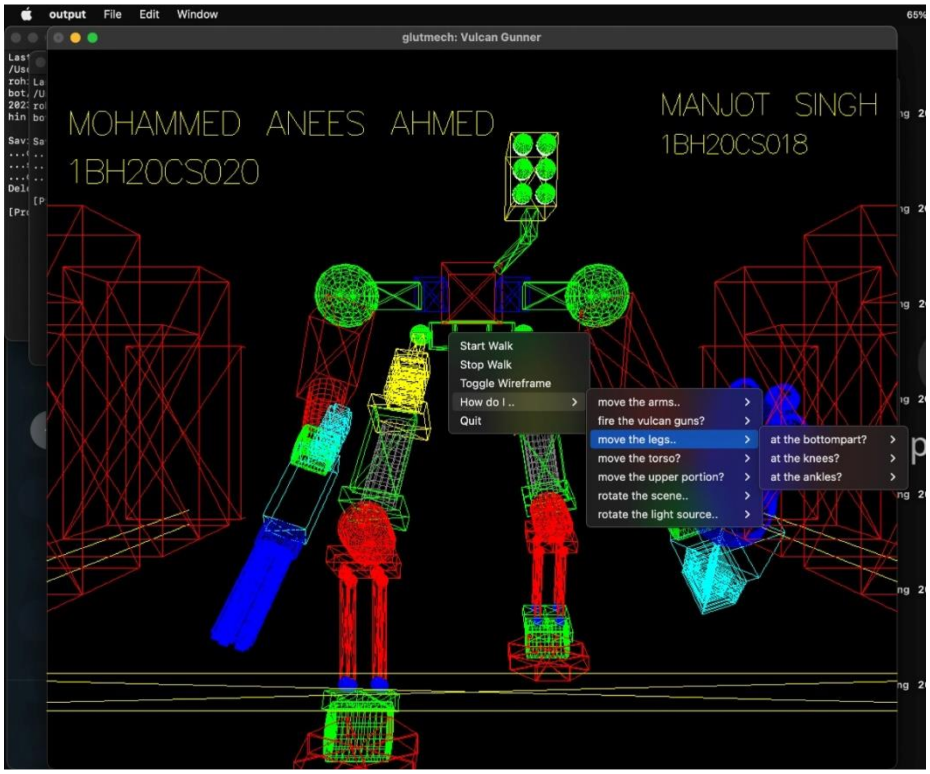

Fig 6.6 Instruction menu

Chapter 7: CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

In conclusion, the use of OpenGL in the development of a walking robot simulation offers numerous benefits and opportunities. By leveraging the power and versatility of the OpenGL graphics library, a visually appealing and realistic representation of a walking robot can be achieved. The system design for a walking robot using OpenGL involves various essential components. These include the creation of a detailed robot model, implementation of animation and kinematics algorithms to produce lifelike movements, integration of physics simulations to ensure stability and realistic interactions, and utilization of OpenGL's rendering capabilities to create immersive visual environments.

7.2 FUTURE ENHANCEMENT

Enhance the realism of the walking robot simulation by incorporating more advanced physics-based algorithms. This could involve simulating more realistic joint movements, dynamic interactions with the environment, and realistic physical properties like inertia and friction.

BIBLIOGRAPHY

BOOKS

- Edward Angel’s Interactive Computer Graphics Pearson Education 5th Edition
- Interactive computer Graphics --A top down approach using open GL— by EdwardAngel.
- Jackie.L.Neider,MarkWarhol,Tom.R.Davis,"OpenGLRed Book",Second. Revised Edition, 2005
- Donald D Hearn and M.PaulineBaker,"Computer Graphics with OpenGL", 3rd Edition.

WEBSITES

- www.OpenGL Redbook.com
- www.OpenGL simple examples.
- www.OpenGL programming guide.
- http://www.wikipedia.com
- http://basic4gl.wikispaces.com
- http://www.opengl.org
- http://pyopengl.sourceforge.net/documentation/manual/glut.3GLUT.htm