**CHAPTER 1**

**INTRODUCTION**

**1.1 Introduction to Computer Graphics**

Graphics are 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 images 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 objects, animation, and many more. In most areas, computer graphics is an abbreviation of CG. There are several tools used for the implementation of Computer Graphics. The basic is the <graphics.h> header file in Turbo-C, Unity for advanced and even OpenGL can be used for its Implementation. It was invented in 1960 by great researchers Verne Hudson and William Fetter from Boeing.

**Computer Graphics refers to several things:**

The manipulation and the representation of the image or the data in a graphical manner.

Various technology is required for the creation and manipulation.

Digital synthesis and its manipulation.

**Types of Computer Graphics**

Raster Graphics: In raster, graphics pixels are used for an image to be drawn. It is also known as a bitmap image in which a sequence of images is into smaller pixels. Basically, a bitmap indicates a large number of pixels together.

Vector Graphics: In vector graphics, mathematical formulae are used to draw different types of shapes, lines, objects, and so on.

**Applications**

* Computer Graphics are used for an aided design for engineering and architectural system- These are used in electrical automobiles, electro-mechanical, mechanical, electronic devices. For example gears and bolts.
* Computer Art – MS Paint.
* 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 pictures, music videos, television gaming.
* Education and training- It is used to understand the operations of complex systems. 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**

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer.

These objects are described as sequences of vertices or pixels.

OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

## **OpenGL Fundamentals**

This 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. Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls.

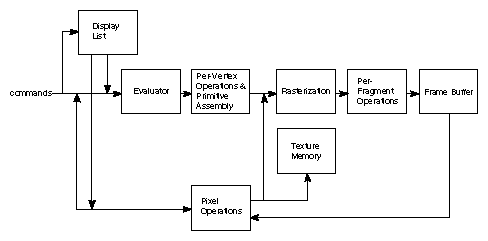
Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The type of 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.

## **Basic OpenGL Operation**

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.

**OpenGL Block Diagram**



**Fig 1.1: Block diagram**

As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing at a later time.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon.

Each fragment so produced is fed into the last stage, per-fragment operations, which performs the final operations on the data before it's stored as pixels in the frame buffer. These operations include conditional updates to the frame buffer based on incoming and previously stored z-value s (for z-buffering) and blending of incoming pixel colours with stored colours, as well as masking and other logical operations on pixel values.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

**CHAPTER 2**

**REQUIREMENTS AND SPECIFICATION**.

**2.1 Hardware Constraints**

• Processor: Intel i5 or above/AMD Ryzen

• RAM: 4GB

• Hard disk: 20GB(approx)

• Display: VGA Colour Monitor

**2.2 Software Constraints**

• Operating System: Windows 10/11/UBUNTU

• Language: C/Open GL

• Compiler: Ubuntu

**CHAPTER 3**

**DESIGN**

**3.1 Flowchart**

START

MAIN( )

INIT( )

**Fig 3.1: Flowchart**

MENU

CONTROLS

ROTATE,MOVE ROBOT

PROCESS NORMAL KEYS

RIGHT BUTTON()

LEFT BUTTON

KEYBOARD( )

MOUSE( )

DISPLAY( )

**3.2 Analysis of Flowchart**

The execution of the program starts with the main function. The main function takes the control to the init function. In init function then it encounters display function and the control is redirected to the display function. In display function the pointer goes to keyboard or mouse function based on the user interaction. This means the pointer of the function checks whether the user is interacting with the program using the keyboard or using the mouse, if the user is using the keyboard to interact then the program then the program checks for the keys that is pressed. If the normal keys are pressed they are processed and no actions are performed, however if keys specified are pressed then they are processed and specified actions are performed. If the pointer goes to the mouse function then mouse button is read and the specified action is performed and when the user is done performing all the functions the pointer goes to end of the program and the execution is completed and the execution ends. This above flowchart (Fig. 3.1) explains the same.

**CHAPTER 4**

**IMPLEMENTATION**

This program is implemented using various openGL functions:

* glutInit() : interaction between the windowing system and OPENGL is initiated.
* glutInitDisplayMode() : used when double buffering is required and depth information is required.
* glutCreateWindow() : this opens the OPENGL window and displays the title at top of the window.
* glutInitWindowSize() : specifies the size of the window.
* glutInitWindowPosition() : specifies the position of the window in screen co-ordinates.
* glutKeyboardFunc() : handles normal ASCII symbols.
* glutSpecialFunc() : handles special keyboard keys.
* glutReshapeFunc() : sets up the call back function for reshaping the window.
* 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 colour to faces.
* glFlush() : used to flush the pipeline.
* glutPostRedisplay() : used to trigger an automatic redraw of the object.
* glMatrixMode() : used to set up the required mode of the matrix.
* glLoadIdentity() : used to load or initialize to the identity matrix.
* glTranslatef() : used to translate or move the rotation centre from one point to another in three dimensions.
* glRotatef() : used to rotate an object through a specified rotation angle

**CODE SNIPPETS**

**CREATING GEOMETRIC BODY SHAPES**

void Octagon(float side, float height, char solid)

{

char j;

float x = sin(0.7) \* side, y = side / 2.0, z = height / 2.0, c;

c = x + y;

for (j = 0; j < 8; j++) {

glTranslatef(-c, 0.0, 0.0);

if (!solid)

glBegin(GL\_LINE\_LOOP);

else

glBegin(GL\_QUADS);

glNormal3f(-1.0, 0.0, 0.0);

glVertex3f(0.0, -y, z);

glVertex3f(0.0, y, z);

glVertex3f(0.0, y, -z);

glVertex3f(0.0, -y, -z);

glEnd();

glTranslatef(c, 0.0, 0.0);

if (solid)

{

glBegin(GL\_TRIANGLES);

glNormal3f(0.0, 0.0, 1.0);

glVertex3f(0.0, 0.0, z);

glVertex3f(-c, -y, z);

glVertex3f(-c, y, z);

glNormal3f(0.0, 0.0, -1.0);

glVertex3f(0.0, 0.0, -z);

glVertex3f(-c, y, -z);

glVertex3f(-c, -y, -z);

glEnd();

}

glRotatef(45.0, 0.0, 0.0, 1.0);

}

}

**CREATING AN ARM**

void ForeArm(char solid)

{

char i;

glNewList(SOLID\_MECH\_FOREARM, GL\_COMPILE);

#ifdef LIGHT

SetMaterial(mat\_specular, mat\_ambient, mat\_diffuse, mat\_shininess);

#endif

glColor3f(0.0, 1.0, 1.0);//fore arm light green

for (i = 0; i < 5; i++) {

glTranslatef(0.0, -0.1, -0.15);

Box(0.6, 0.8, 0.2, solid);

glTranslatef(0.0, 0.1, -0.15);

Box(0.4, 0.6, 0.1, solid);

}

glTranslatef(0.0, 0.0, 2.45);

Box(1.0, 1.0, 2.0, solid);

glTranslatef(0.0, 0.0, -1.0);

glEndList();

}

**ADDING LIGHTING EFFECTS**

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);

}

**CREATING MENU FUNCTIONS**

void glutMenu(void)

{

int glut\_menu[13];

glut\_menu[5] = glutCreateMenu(null\_select);

glutAddMenuEntry("forward : q,w", 0);

glutAddMenuEntry("backwards : a,s", 0);

glutAddMenuEntry("outwards : z,x", 0);

glutAddMenuEntry("inwards : Z,X", 0);

glut\_menu[6] = glutCreateMenu(null\_select);

glutAddMenuEntry("upwards : Q,W", 0);

glutAddMenuEntry("downwards : A,S", 0);

glutAddMenuEntry("outwards : 1,2", 0);

glutAddMenuEntry("inwards : 3,4", 0);

glut\_menu[1] = glutCreateMenu(null\_select);

glutAddMenuEntry(" : Page\_up", 0);

glut\_menu[8] = glutCreateMenu(null\_select);

glutAddMenuEntry("forward : y,u", 0);

glutAddMenuEntry("backwards : h.j", 0);

glutAddMenuEntry("outwards : Y,U", 0);

glutAddMenuEntry("inwards : H,J", 0);

glut\_menu[9] = glutCreateMenu(null\_select);

glutAddMenuEntry("forward : n,m", 0);

glutAddMenuEntry("backwards : N,M", 0);

glut\_menu[10] = glutCreateMenu(null\_select);

glutAddMenuEntry("toes up : K,L", 0);

glutAddMenuEntry("toes down : k,l", 0);

glut\_menu[11] = glutCreateMenu(null\_select);

glutAddMenuEntry("right : right arrow", 0);

glutAddMenuEntry("left : left arrow", 0);

glutAddMenuEntry("down : up arrow", 0);

glutAddMenuEntry("up : down arrow", 0);

glut\_menu[12] = glutCreateMenu(null\_select);

glutAddMenuEntry("right : p", 0);

glutAddMenuEntry("left : i", 0);

glutAddMenuEntry("up : 9", 0);

glutAddMenuEntry("down : o", 0);

glut\_menu[4] = glutCreateMenu(NULL);

glutAddSubMenu("at the shoulders? ", glut\_menu[5]);

glutAddSubMenu("at the elbows?", glut\_menu[6]);

glut\_menu[7] = glutCreateMenu(NULL);

glutAddSubMenu("at the bottompart? ", glut\_menu[8]);

glutAddSubMenu("at the knees?", glut\_menu[9]);

glutAddSubMenu("at the ankles? ", glut\_menu[10]);

glut\_menu[2] = glutCreateMenu(null\_select);

glutAddMenuEntry("turn left : d", 0);

glutAddMenuEntry("turn right : g", 0);

glutAddMenuEntry("Rocketpod : v", 0);

glut\_menu[3] = glutCreateMenu(null\_select);

glutAddMenuEntry("tilt backwards : f", 0);

glutAddMenuEntry("tilt forwards : r", 0);

glut\_menu[0] = glutCreateMenu(NULL);

glutAddSubMenu("move the arms.. ", glut\_menu[4]);

glutAddSubMenu("fire the vulcan guns?", glut\_menu[1]);

glutAddSubMenu("move the legs.. ", glut\_menu[7]);

glutAddSubMenu("move the torso?", glut\_menu[2]);

glutAddSubMenu("move the upper portion?", glut\_menu[3]);

glutAddSubMenu("rotate the scene..", glut\_menu[11]);

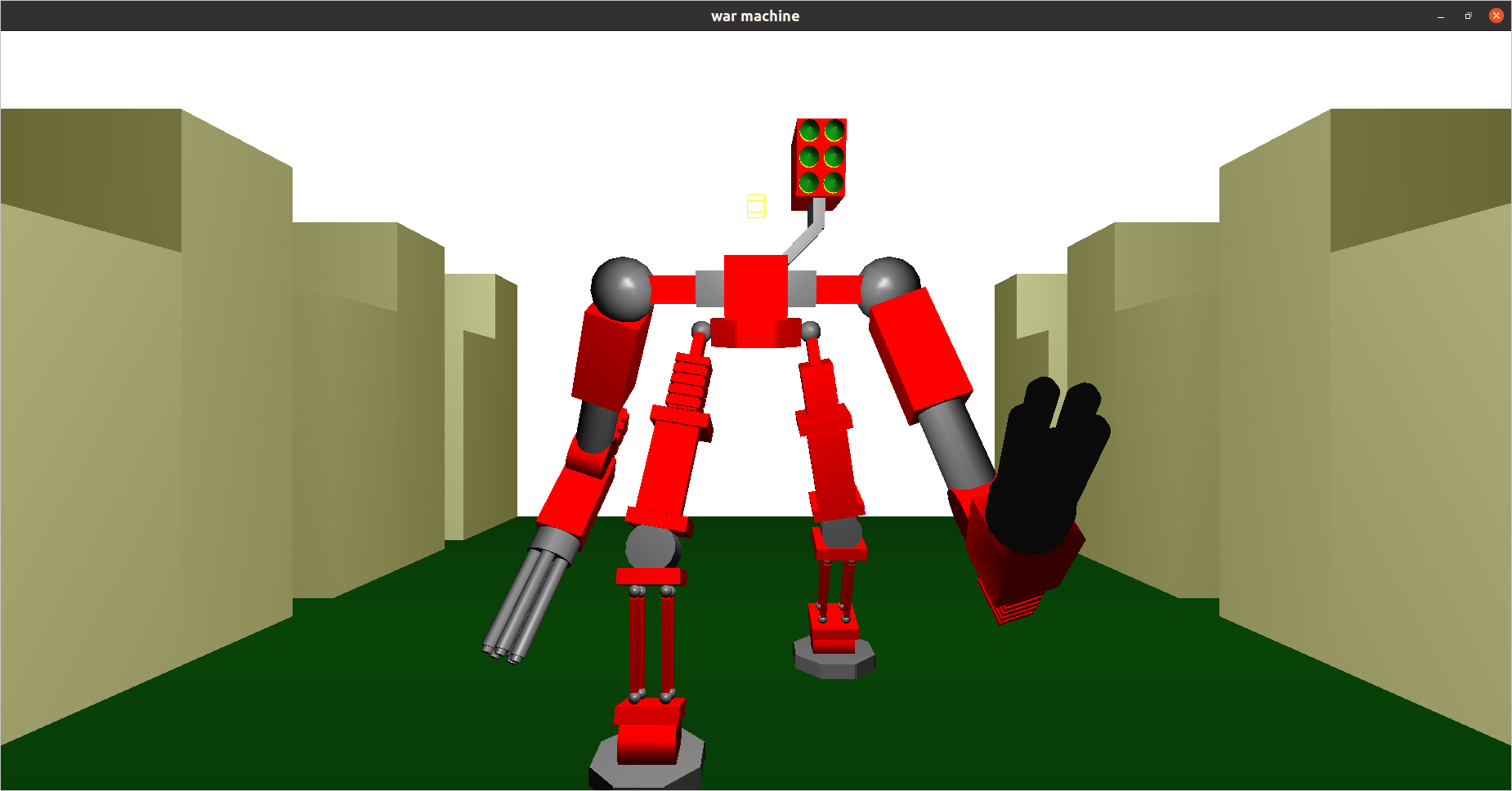
#ifdef MOVE\_LIGHT

glutAddSubMenu("rotate the light source..", glut\_menu[12]);

#endif

**CHAPTER 5**

**SNAPSHOTS**



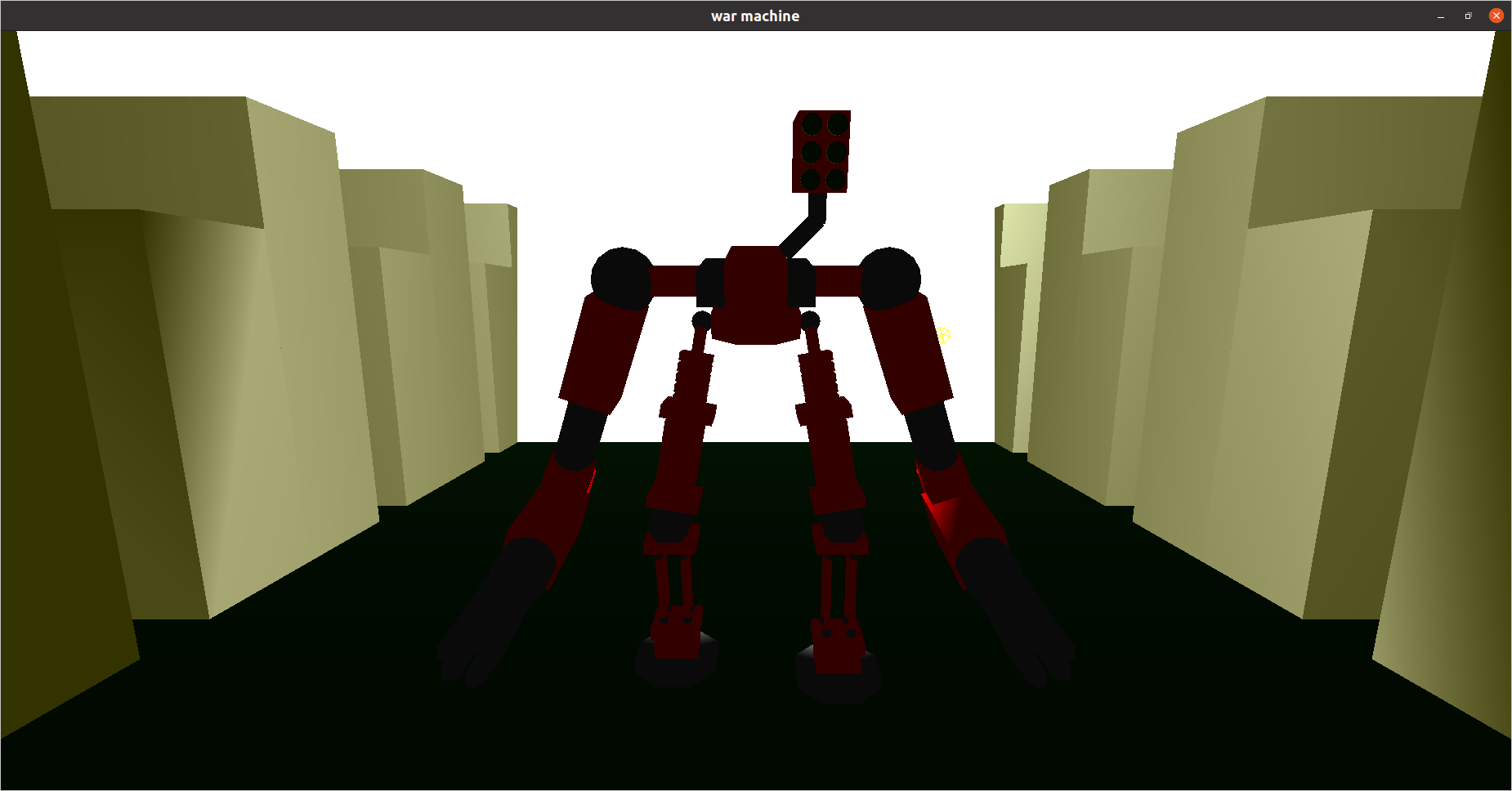
**Fig 5.1: Front View of the robot**

This is the front view of the robot while walking. In this view we will see the robot facing towards the camera. The robot’s arms, legs, torso everything and every object is visible to the viewer. This is also a daylight view of the robot. In this view each and every object that is created is visible to the viewer

****

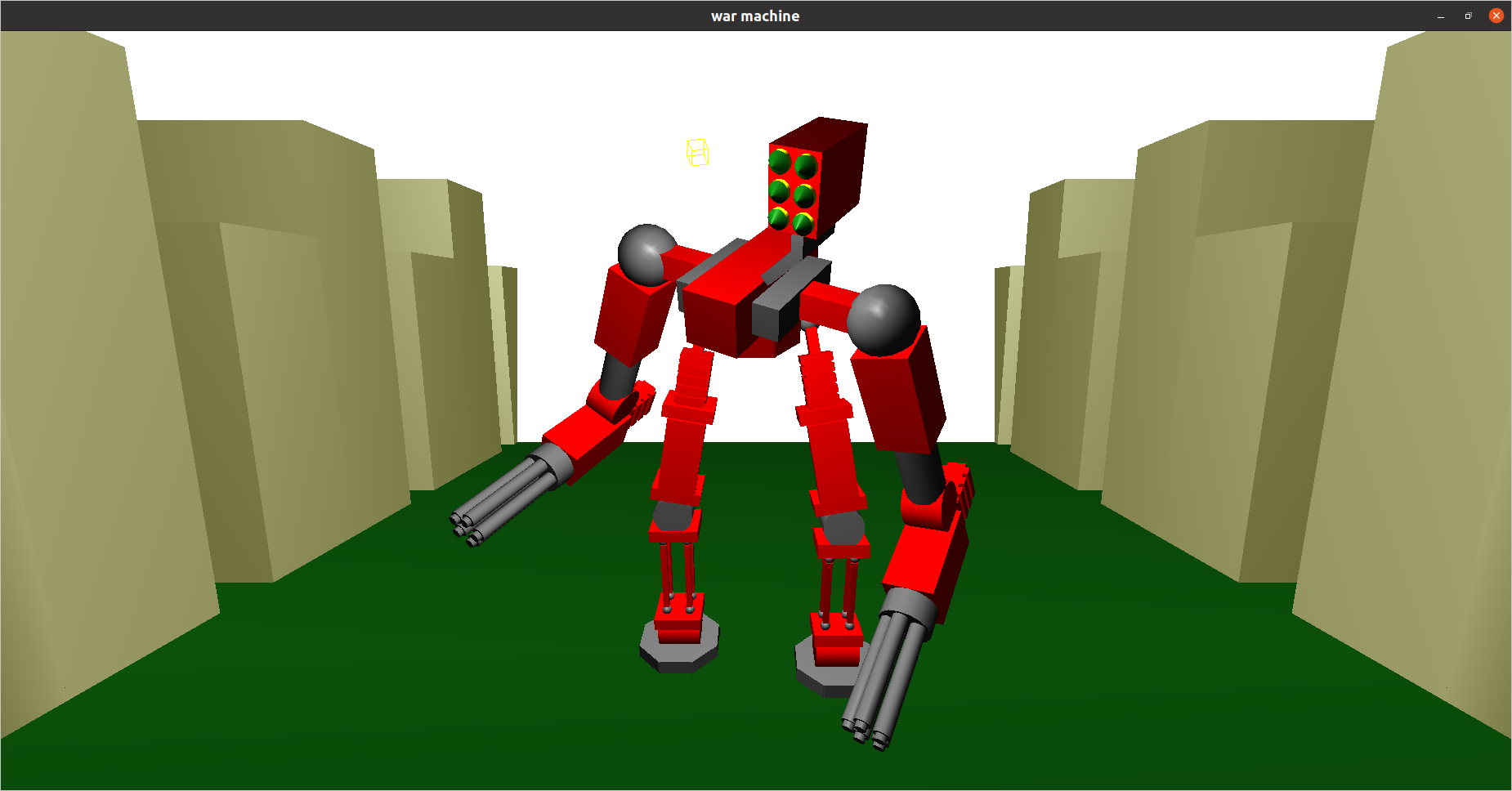
**Fig 5.2: Side View of the robot**

This is the side view of the robot. The side is achieved by rotating the camera using the arrow keys. Here not all the objects created for the robot is displayed but only certain of them are displayed such as only one side of the buildings are visible and the other leg and arms are partially visible. This is also a daylight view with background having daylight

****

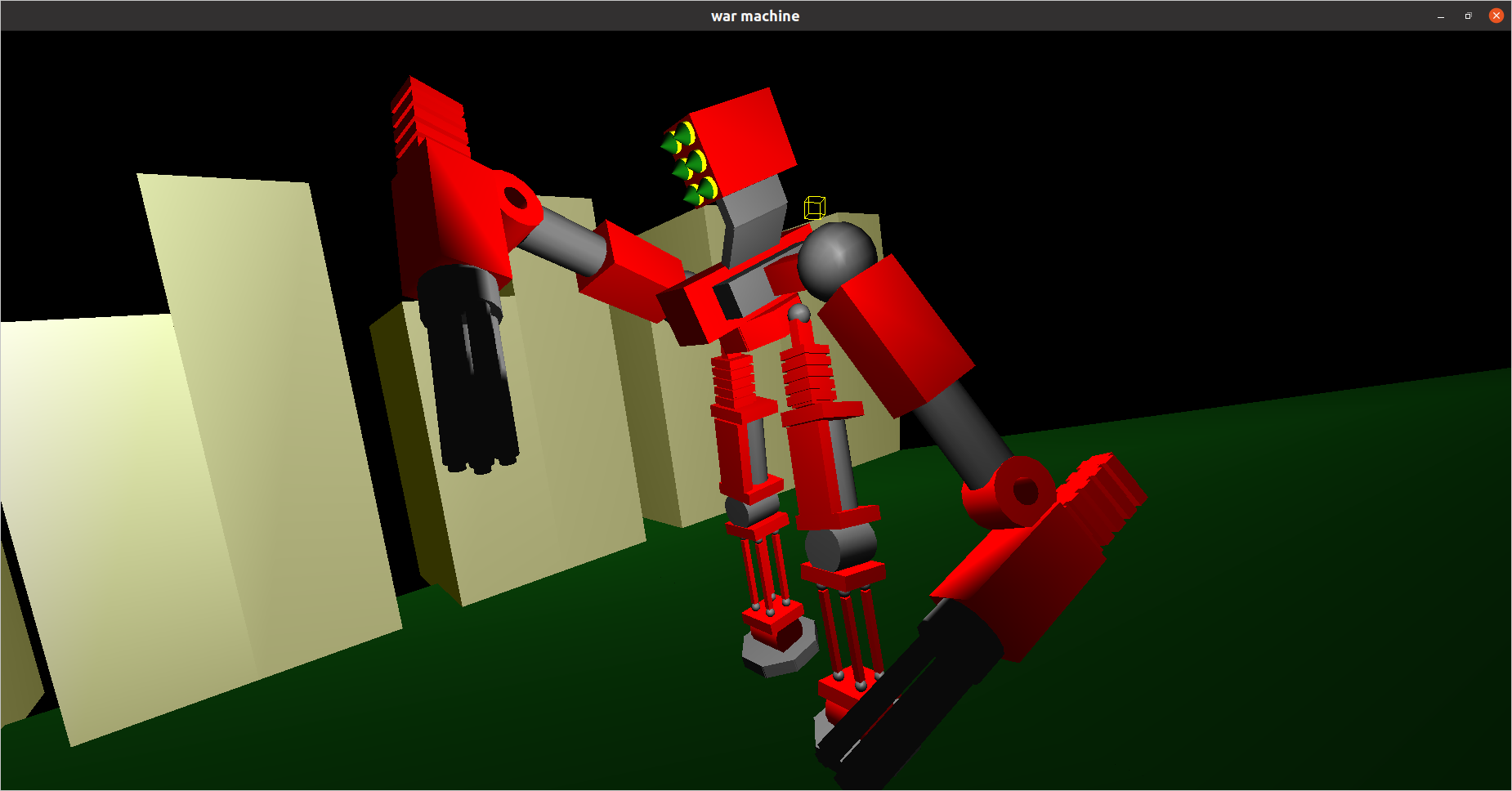
**Fig 5.3: Front view of robot with change in lighting position**

This is another type of front view where the lighting effects have come into picture. Even though it has a day light background the lighting effects are such that it is diffused with the light position here the rear side of the robot is receiving greater lightning than the front view hence it seems to the viewer that the front side of the robot is darker. Since this is also the front view all the objects created are mostly visible.

****

**Fig 5.4: Front View of robot with bent torso**

This is view of a robot with translation of its body parts rather than the whole robot itself. The torso and arms can be bent by first right clicking on the screen then selecting the corresponding option the we use arrow keys for translation (bending) of the torso and the view presented in the above figure is a front view with daylight background

****

**Fig 5.5:** **Side and night view of robot with bent torso and hands**

This is a side of the robot with its hands and torso bent and having a nightlight background, meaning this the night view. As depicted in the picture the hands and torso are bent by right clicking on the screen and selecting the appropriate option in the menu popped when the user right clicks then we can use the arrow keys to move the arms and torso objects in the direction we want. This is also the side view hence all the objects in the scene are not exactly visible

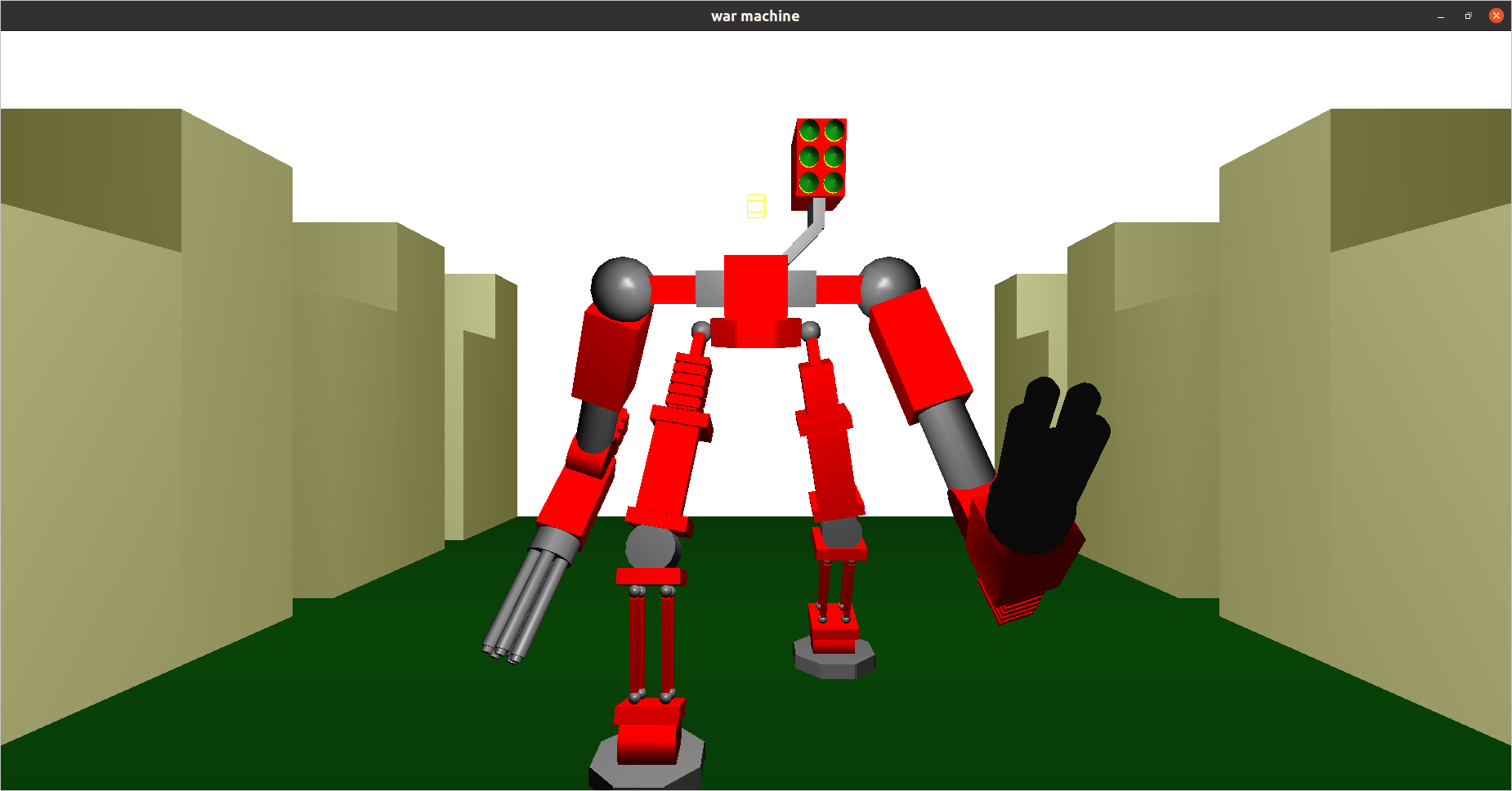
**CONCLUSION & FUTURE ENHANCEMENTS**

**CONCLUSION**

The project “War Robot Simulation” demonstrates the simulation and working of a robot. We first created the body frame, mechanical arms and legs. Then we created the functions for the user to rotate the robot in both clockwise and anti-clockwise direction, then we added functions for robot to walk. We then added texture and material properties to the robot. Then mouse and keyboard functions were made for making the robot interactive. This was done by adding specific key binds. Then we gave functions to the mouse buttons and keyboard keys then we designed a menu to show users about the actions that can be performed and actions were specified in this menu. Then all the codes are joined and Finally, we conclude that this program clearly illustrated the simulation of a robot using OpenGL and has been completed successfully and is ready to be demonstrated.

**FUTURE ENHANCEMENTS**

Since this project is a simulation, it can be further enhanced by the application of the additional parts and features for the robots using OpenGL for study purposes. This makes the project more useful for study purposes and easier to understand and analyse the working of robots for future designing.

**REFERENCES**

* [www.google.com](http://www.google.com)
* <https://learnopengl.com>
* https://en.wikipedia.org
* **TEXT BOOKS:**

INTERACTIVE COMPUTER GRAPHICS A TOP-DOWN APPROACH

* Edward Angel.

COMPUTER GRAPHICS, PRINCIPLES & PRACTICES

- Foley van dam

- Feiner Hughes