**CHAPTER 1**

**INTRODUCTION**

**1.1 Computer Graphics**

Computer graphics are [graphics](http://en.wikipedia.org/wiki/Graphics) created using [computers](http://en.wikipedia.org/wiki/Computer) and, more generally, the [representation](http://en.wikipedia.org/wiki/Representation) and [manipulation](http://en.wikipedia.org/wiki/Manipulation) of [pictorial](http://en.wikipedia.org/wiki/Pictorial) [data](http://en.wikipedia.org/wiki/Data) by a [computer](http://en.wikipedia.org/wiki/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 graphics have had a profound impact on many types of media and have revolutionized the [animation](http://en.wikipedia.org/wiki/Animation) and [video game](http://en.wikipedia.org/wiki/Video_game) industry.

**1.2 OpenGL(Open Graphics Library)**

OpenGL has become a widely accepted standard for developing graphics application. OpenGL is easy to learn, and it possesses most of the characteristics of other popular graphics system. It is top-down approach. OpenGL is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives.

OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms.

The interface between the application program and the graphics system can be specified through that set of function that resides in graphics library. The specification is called the APPLICATION PROGRAM INTERFACE (API). The application program sees only the API and is thus shielded from the details both the hardware and software implementation of graphics library. The software driver is responsible for interpreting the output of an API and converting these data to a form that is understood by the particular hardware.

Most of our applications will be designed to access openGL directly through functions in three libraries. Function in the main GL library have name that begin with the letter gl and stored in the library. The second is the openGL utility Library (GLU). This library uses only GL function but contains codes for creating common object and viewing. Rather then using an different library for each system we used available library called openGL utility toolkit(GLUT).

**1.3 Aim of the project**

This project is a demonstration of 3D graphical implementation of ‘ROBOT ROTATION’. This program implements the rotation,translation of 3D objects in openGL. The program is written in C++ language using GL, GLU and GLUT library functions.

An attempt has been made to develop an OpenGL package which meets necessary requirements of the user successfully. Since it is user friendly, it enables the user to interact efficiently and easily.

The development of the mini project has given us a good exposure to OpenGL by which we have learnt some of the techniques such as transformation, lighting and animation etc which help in development of the object on the computer screen. Hence it is helpful for us even to take up this field as our career too and develop some other features in OpenGL and provide as a token of contribution to the graphics world.

**CHAPTER 2**

**REQUIREMENTS & SPECIFICATION**

**2.1 Hardware requirements**

* Processor : Pentium IV 2.4 GHz or above
* Hard Disk : 20 GB, 80 GB, 160 GB or above
* Monitor : 15 VGA colour
* RAM : 256 MB or above
* QWERTY Keyboard
* Mouse

**2.2 Software Requirements**

* Operating System : Windows XP or Higher version.
* Tool used : Microsoft Visual Studio C++ and OpenGL
* Coding Language : C++
* Library used : GLUT

**CHAPTER 3**

**DESIGN**

This project is a demonstration of 3D graphical implementation of “ROBOT ROTATION” .This project implements the translation,rotation of 3D objects.

**3.1 ALGORITHM**

**STEP 1:**

In the declaration section we define variables to store the objects declaration, keyboard status,clearing the buffers and declare double buffer which is used as swap buffers.

**STEP 2:**

First,in the main function,the GLUT library is initialized and a session is negotiated with the window system.In addition the callbacks are set for the display,keyboard and reshape.

**STEP 3:**

In the display function is used to display the objects in the initial stage. This function is also used to display the objects on one at a time. It is also used to display the menu from which user can select the option.

**STEP 4:**

If we right click the button, we get the options like rotation of robot,hand rotation,leg rotation.

**STEP 5:**

If we select the rotation use arrow keys from the keyboard.

**STEP 6:**

If we click on hand rotation ,will get sub menu hand1,hand2 from hand1 we get option like at the shoulder S,s and at the elbow E,e hand2 we getat shoulder D,d and at the elbow F,f.

**STEP 7:**

If we click on the leg rotation we get options like leg1,leg2.If we select leg1 we get at the hip L,l and at the knees M,m similarly we get options for leg2 at the hip N,n and at the knees O,o.

**STEP 8:**

If we click to move the camera we select move the left button of the mouse horizontally.

**STEP 9:**

If we click on quit option we can come out of window.

**3.2 Built in functions:**

* **glVertex\*()**

It is the vertex function in graphics where \* can be interpreted as 2 or 3 characters of the form nt or ntv, where ‘n’ signifies no of dimensions and ‘t’ denotes data type.

* **glClearColor():** Specify clear values for the color buffers.
* **glColor3f():** Set the color.
* **glTranslatef(GLfloatx,GLfloaty,GLfloat z):**It produces a translation by x y z..
* **glPushMatrix( ):** To execute an instance transformation that applies to a particular object and does not apply to following objects in the code we use glPushMatrix( ) to push it on to stack before we multiply by instance transformation.
* **glPopMatrix():**We recover the above pushed code using glPopMatrix( ).
* **glLoadIdentity():** Replace the current matrix with the identity matrix.
* **glMatrixMode():** Sets the current matrix mode.
* **glProjection:** Projects global coordinates into clip space.
* **gluOrtho2D():** which defines a two dimensional viewing rectangle in the plane z=0.
* **Void glutInit():** Initializes GLUT, the arguments from main are passed in and can be used by the application.
* **Void glutMainLoop():** Cause the program to enter an event-processing loop. It should be the last statement in main function.
* **Void glutBitmap charcter(void \*font,int character):** Renders a bitmap charcter using openGL usage.bitmap font to use charcter to render(not confined to 8bits)

Glutbitmap character(GLUT\_BITMAP\_HELVETICA\_10,s[i]);

Glutbitmap character(GLUT\_BITMAP\_HELVETICA\_18,s[i]);

Glutbitmap character(GLUT\_BITMAP\_HELVETICA\_12,s[i]);

* **glScalef(x-axis,y-axis,z-axis)**

**A** scaling matrix with a fixed point of the origin allows for independent scaling along

the coordinate axis.

* **GlutSwapbuffers:** performs a buffer swap on the layer in use for the cuurent window
* **Void reshape(int w,int h):** it is called each time the window is moved or resized it sets the display window to the width ‘w’and height ’h’ specified.
* **glutpostRedisplay:** marks the normal plan of current window as needing to be redisplayed.
* **glutKeyboardFunc:**sets the keyboard callback.

**CHAPTER 4**

**TESTING**

Testing is a dynamic technique of verification and validation. It involves executing an implementation of the software with test data and examining the outputs of the software and its operational behaviour to check that it is performing as required.

The following statements serve as the objectives for testing:

1. Testing is a process of executing a program with the intent of finding error

2. A good test case is one that has a high probability of finding an as-yet undiscovered

Error.

3. A successful test is one that uncovers as-yet undiscovered error.

Verification and validation is intended to show that a system confirms to its specification and that the system meets the expectations of the customer. Verification involves checking that the software confirms to its specification. We should check that the system meets its specified functional and non-functional requirements. Validation ensures that the software meets the expectations of the customer. It goes beyond checking conformance of the system to its specification to showing that the software does what the customer expects as distinct from what has been specified.

The testing process should proceed in stages where testing is carried out incrementally in conjunction with system implementation. Figure 4.1 shows a five stage testing process where system components are tested, the integrated system is tested and, finally, the system is tested with the customer’s data.

The stages in the testing process are:

**4.1 Unit testing**

Individual components are tested to ensure that they operate correctly.

**4.2 Module testing**

A module is a collection of dependent components such as an object class, an abstract data type or some looser collection of procedures and functions. A module encapsulates related components, so can be tested without other system modules.

**4.3 Sub-system testing**

This phase involves testing collections of modules which have been integrated into sub-systems. The sub-system test process should concentrate on the detection of module interface errors by rigorously exercising these interfaces

**4.4 System testing**

The sub-systems are integrated to make up the system. This process is concerned with finding errors that result from unanticipated interactions between sub-systems and sub-system interface problems. It is also concerned with validating that the system meets its functional and non-functional requirements and testing the emergent system properties.

**4.5** **Acceptance testing:**

This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than simulated test data.

Table 4.1 Test cases for mouse options

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SLNO | TEST CASE  DESCRIPTION | EXPECTED  RESULT | ACTUAL  RESULT | REMARK |
| 1 | Click the RIGHT button on the display screen | Menu with robot rotation, hand rotation, leg rotation, to move the camera and quit | Menu with robot rotation is displayed | Pass |
| 2 | Click on the robot rotation | The user should be able to view the rotating robot | The user is able to view the rotating robot | Pass |
| 3  a  b | Click on the hand rotation  Click on the hand1  Click on the hand2 | The user should be able to view the option hand1,hand2  The user should be able to view the shoulder and elbow rotation  The user should be able to view the shoulder and elbow rotation | The user should be able to view the hand rotation of robot  The user is able to view the rotation of hand1  The user is able to view the rotation of hand2 | Pass  Pass  Pass |
| 4  a  b | Click on the leg rotation  Click on the leg1  Click on the leg2 | The user should be able to view the option leg1,leg2  The user should be able to view the rotation of hip and knees  The user should be able to view the rotationon of hip and knees | The user should be able to view the leg rotation of robot  The user should be able to view the leg1 rotation of robot  The user is able to view leg2 rotation of robot | Pass  pass  pass |
| 5 | Click on the Quit option | The user should be able to come out off the outputscreen | The user is able to come out off the output screen | Pass |

**CHAPTER 5**

**RESULT AND SNAPSHOTS**

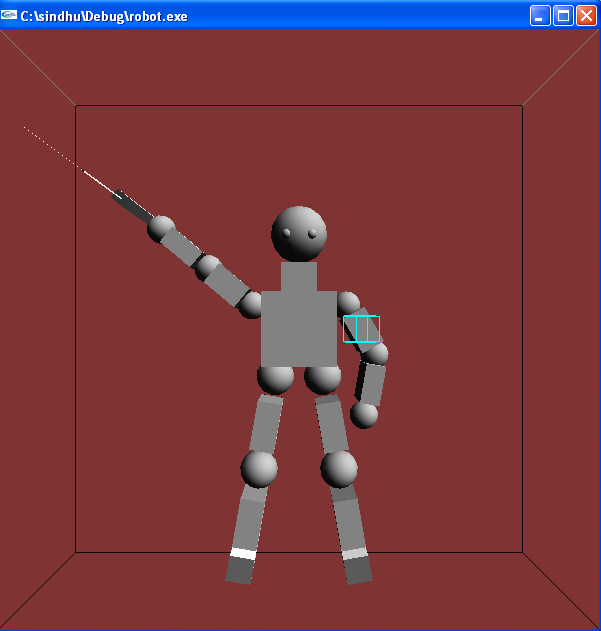
****

Fig. 5.1:Sample output

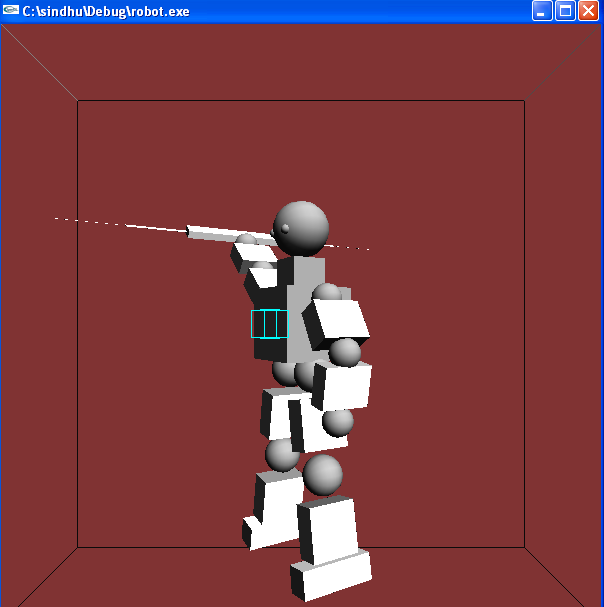


Fig 5.1:left rotation of robot

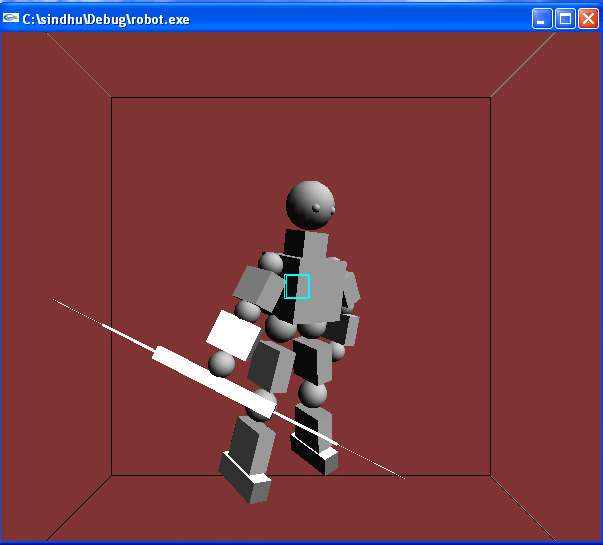


Fig 5.1:right rotation of robot

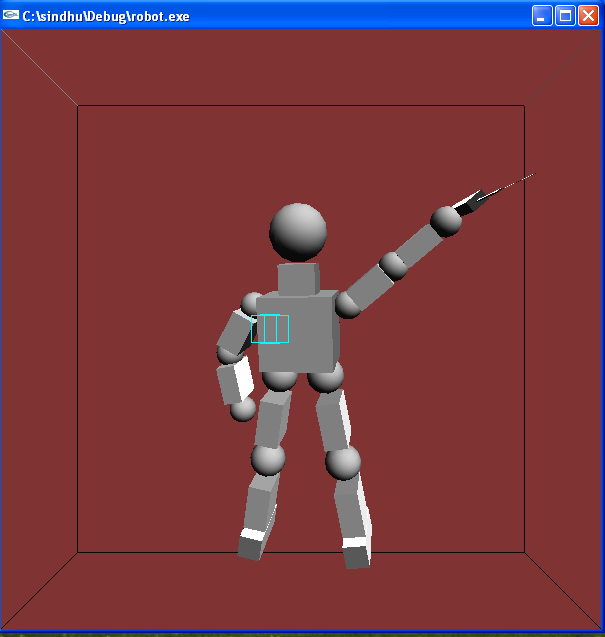


Fig 5.1:back view of robot

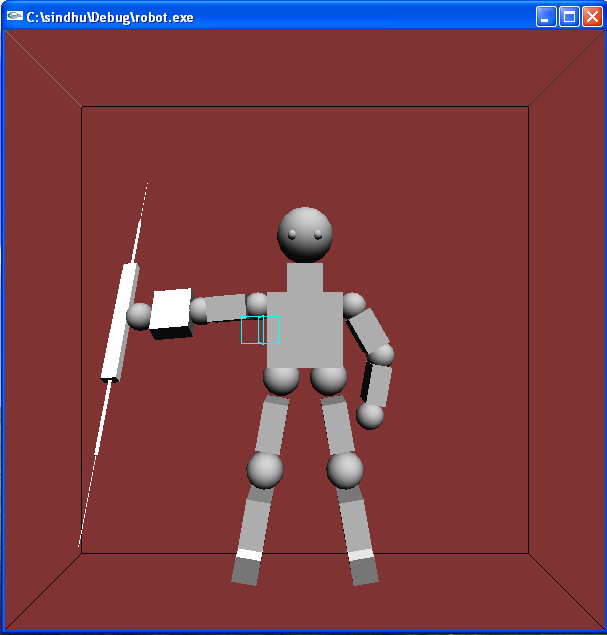
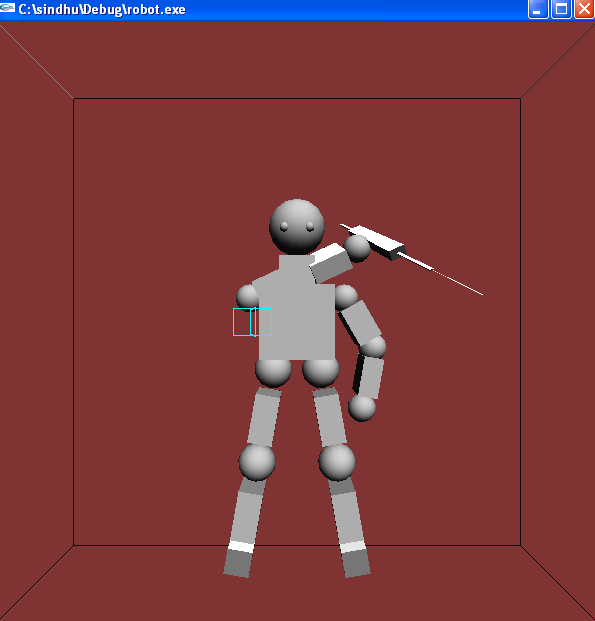
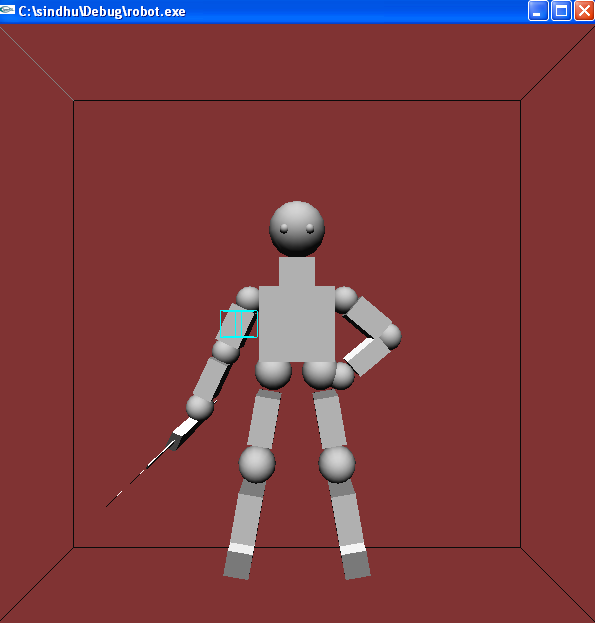
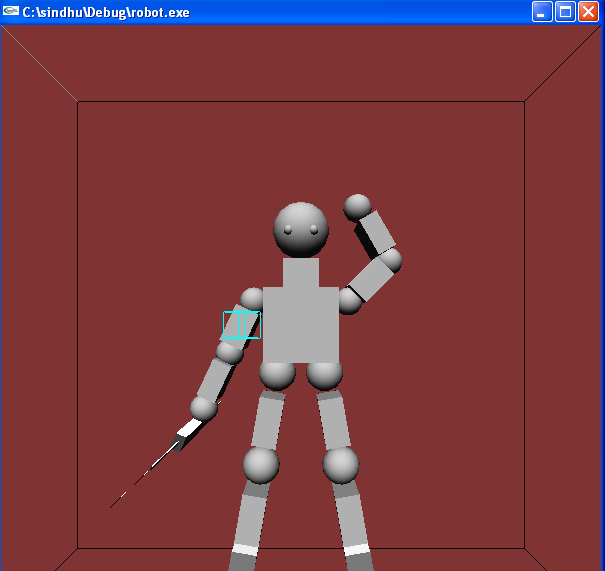
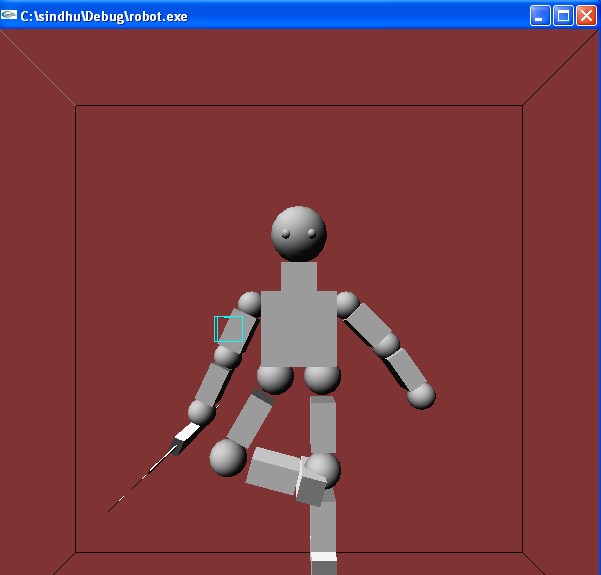
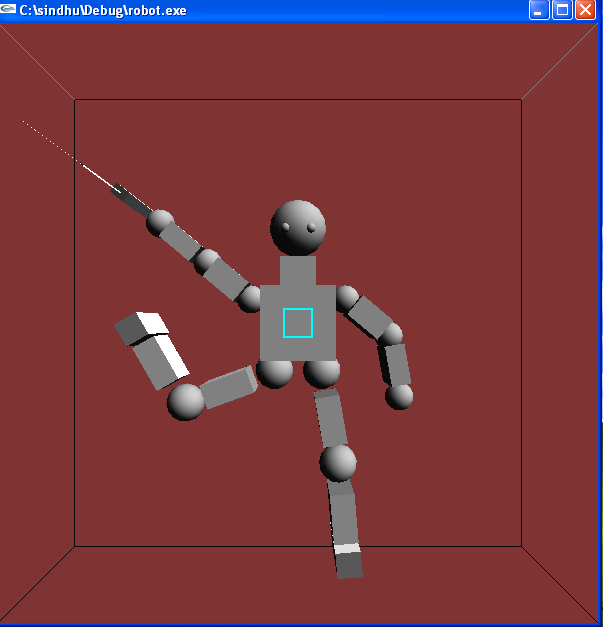
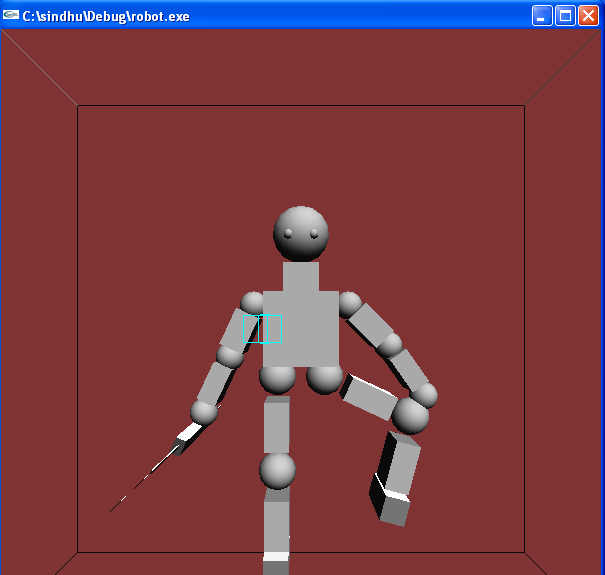
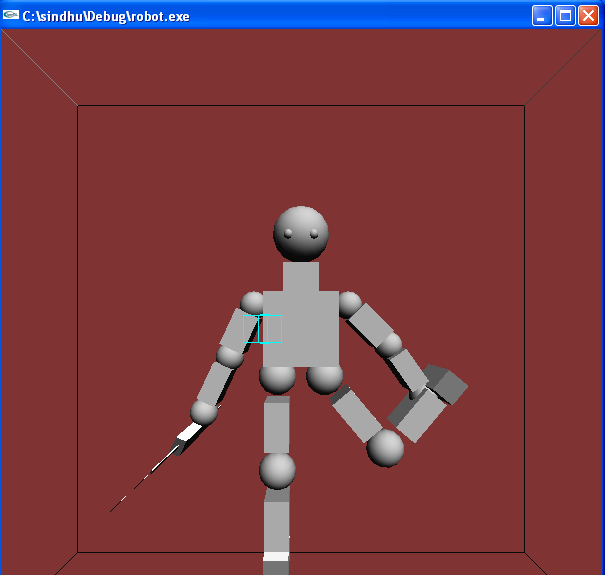
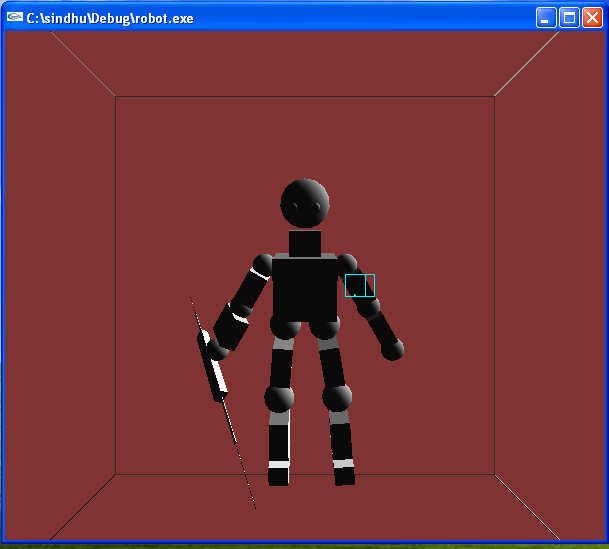
****

Fig 5.2:hand1 rotation in clockwise

Fig 5.2:hand1 in anticlockwise

Fig 5.3:hand2 in clockwise

Fig 5.3:hand2 in anti clockwiseFig 5.4:leg1 in anti clockwiseFig 5.4:leg1 in clockwiseFig 5.5:leg2 in clockwiseFig 5.5:leg2 in anti clockwiseFig 5.6:camera position

**CHAPTER 6**

**CONCLUSION AND FUTURE ENHANCEMENT**

**6.1 CONCLUSION**

An attempt has been made to develop an OpenGL package which meets necessary requirements of the user successfully. Since it is user friendly, it enables the user to interact efficiently and easily.

The development of the mini project has given us a good exposure to OpenGL by which we have learnt some of the technique which help in development of 3D objects and gaming.

Hence it is helpful for us even to take up this field as our career too and develop some other features in OpenGL and provide as a token of contribution to the graphics world.

**6.2 FUTURE ENHANCEMENT**

Enhancement indicates a multiple of ways, supported by a variety of technologies, in which robot rotation are enhanced in their abilities, features or functions.

* GUI enhancement can be done to make the geographical user interface more friendly and easier to use.
* A toolbar should be created to inform users to use the functions efficiently . This can be done by updating current windows application to .NET application.
* Providing high quality graphics.

**BIBLIOGRAPHY**

[1] Donald D Hearn and M. Pauline Baker,3rd edition,”*Computer Graphics with OpenGL*”.

[2] Edward Angel,5th edition, Pearson Education,2005,“*Interactive Computer Graphics”.*

[3] F.S.Hill and Stephen M.Kelly ,”Computer Graphics using OpenGl “,3rd edition .

**APPENDIX**

SOURCE CODE

#define VIEW\_TURN\_RATE 10

#include<stdio.h>

#include <GL/glut.h>

#include <stdlib.h>

static int shoulder1 = 0, elbow1 = 0, shoulder2 = 0, elbow2 = 0,leg11=0,leg12=0,leg21=0,leg22=0;

static int spin = 0, spin1=0;

static int begin;

static int turn,turn1=0;

// Mouse Function

Void movelight(int button, int state, int x, int y)

{

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN)

{

begin = x;

}

}

//Motion Function

Void motion(int x, int y)

{

spin = (spin + (x - begin)) % 360;

begin = x;

glutPostRedisplay();

}

//Initializations

void init(void)

{

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_DEPTH\_TEST);

glClearColor (0.5, 0.2, 0.2, 0.8);

// glShadeModel (GL\_FLAT);

}

// Drilling Machine

void drilling()

{

glPushMatrix();

glScalef (0.3, 0.2, 3.0);

glutSolidCube (1.0);

glScalef (0.2, 0.1, 2.0);

glutSolidCube (1.0);

glScalef (0.2, 0.1, 1.5);

glutSolidCube (1.0);

glPopMatrix();

}

//Body of the robot

void robo()

{

glPushMatrix();

//neck

glPushMatrix();

glTranslatef (0.0, 1.0, 0.0);

glutSolidCube(0.75);

glPopMatrix();

//head

glPushMatrix();

glTranslatef (0.0, 2.0, 0.0);

glutSolidSphere (0.6, 15, 15);

glPushMatrix(); //left eye

glTranslatef(-0.25,-0.08,0.5);

glutSolidSphere(0.1,15,15);

glPopMatrix();

glPushMatrix(); // right eye

glTranslatef(0.25,-0.08,0.5);

glutSolidSphere(0.1,15,15);

glPopMatrix();

glPopMatrix();

//chest

glPushMatrix();

glTranslatef (0.0, 0.0, 0.0);

glutSolidCube (1.5);

glPopMatrix();

//left hand

glPushMatrix();

glTranslatef (-1.0, 0.5, 0.0);

glutSolidSphere (0.3, 15, 15);

glRotatef ((GLfloat) shoulder1, 0.0, 0.0, 1.0);

glRotated(-40,0.0,0.0,1.0);

glTranslatef (-0.6, 0.0, 0.0);

glPushMatrix();

glScalef (0.8, 0.5, 1.0);

glutSolidCube (1.0); //shoulder1

glPopMatrix();

glTranslatef (-0.6, 0.0, 0.0);

glutSolidSphere (0.3, 15, 15);

glRotatef ((GLfloat) elbow1, 1.0, 0.0, 0.0);

glTranslatef (-0.6, 0.0, 0.0);

glPushMatrix();

glScalef (0.8, 0.4, 1.0);

glutSolidCube (1.0); //elbow1

glPopMatrix();

glTranslatef (-0.7, 0.0, 0.0);

glutSolidSphere (0.3, 15, 15);

glTranslatef (-0.4, 0.0, 0.0);

drilling(); //drilling machine

glPopMatrix();

// right hand

glPushMatrix();

glTranslatef (1.0, 0.5, 0.0);

glutSolidSphere (0.3, 15, 15);

glRotatef ((GLfloat) shoulder2, 0.0, 0.0, 1.0);

glRotated(-60,0.0,0.0,1.0);

glTranslatef (0.6, 0.0, 0.0);

glPushMatrix();

glScalef (0.8, 0.5, 1.0);

glutSolidCube (1.0); //shoulder2

glPopMatrix();

glTranslatef (0.6, 0.0, 0.0);

glutSolidSphere (0.3, 15, 15);

glRotatef ((GLfloat) elbow2, 0.0, 0.0, 1.0);

glRotated(-40,0.0,0.0,1.0);

glTranslatef (0.6, 0.0, 0.0);

glPushMatrix();

glScalef (0.8, 0.4, 1.0);

glutSolidCube (1.0); //elbow2

glPopMatrix();

glTranslatef (0.7, 0.0, 0.0);

glutSolidSphere (0.3, 15, 15);

glPopMatrix();

// leg1

glPushMatrix();

glTranslatef (-0.5, -1.0, 0.0);

glutSolidSphere (0.4, 15, 15);

glRotatef ((GLfloat) leg11, 0.0, 0.0, 1.0);

glRotated(80,0.0,0.0,1.0);

glTranslatef (-1.0, 0.0, 0.0);

glScalef (1.0, 0.5, 1.0);

glPushMatrix();

glPopMatrix();

glutSolidCube (1.0); //leg11

glTranslatef (-1.0, 0.0, 0.0);

glutSolidSphere (0.4, 15, 15);

glRotatef ((GLfloat) leg12, 0.0, 0.0, 1.0);

glTranslatef (-1.0, 0.0, 0.0);

glPushMatrix();

glScalef (1.0, 0.5, 1.0);]

glutSolidCube (1.0); //leg12

glPopMatrix();

glPushMatrix();

glTranslatef (-0.8, 0.0, 0.0); //foot

glScalef (0.5, 0.5, 1.5);

glutSolidCube (1.0);

glPopMatrix();

glPopMatrix();

// right leg

glPushMatrix();

glTranslatef (0.5, -1.0, 0.0);

glutSolidSphere (0.4, 15, 15);

glRotatef ((GLfloat) leg21, 0.0, 0.0, 1.0);

glRotated(-80,0.0,0.0,1.0);

glTranslatef (1.0, 0.0, 0.0);

glPushMatrix();

glScalef (1.0, 0.5, 1.0);

glutSolidCube (1.0); //leg21

glPopMatrix();

glTranslatef (1.0, 0.0, 0.0);

glutSolidSphere (0.4, 15, 15);

glRotatef ((GLfloat) leg22, 0.0, 0.0, 1.0);

glTranslatef (1.0, 0.0, 0.0);

glPushMatrix();

glScalef (1.0, 0.5, 1.0);

glutSolidCube (1.0); //leg22

glPopMatrix();

glPushMatrix();

glTranslatef (0.80, 0.0, 0.0);

glScalef (0.5, 0.5, 1.5);

glutSolidCube (1.0); //foot

glPopMatrix();

glPopMatrix();

glPopMatrix();

}

/\* start of rotation of robot functions \*/

Void TurnRight(void)

{

turn = (turn - VIEW\_TURN\_RATE) % 360;

}

Void TurnLeft(void)

{

turn = (turn + VIEW\_TURN\_RATE) % 360;

}

Void TurnForwards(void)

{

turn1 = (turn1 - VIEW\_TURN\_RATE) % 360;

}

void

TurnBackwards(void)

{

turn1 = (turn1 + VIEW\_TURN\_RATE) % 360;

}

//Display Function

void display(void)

{

glClear (GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT );

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

glMatrixMode(GL\_MODELVIEW);

glPushMatrix();

glTranslatef(0.0, 0.0, -5.0);

glPushMatrix();

glScalef(1.0,1.0,1.0);

glPushMatrix();

glColor3f(0.0,1.0,0.0);

glutWireCube(18.0); //wire frame

glPopMatrix();

glPushMatrix(); //robo function

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

glRotatef((GLfloat) turn1, 1.0, 0.0, 0.0);

robo();

glPopMatrix();

glPushMatrix(); // Light properties

glRotated((GLdouble) spin, 0.0, 1.0, 0.0);

glRotated(0.0, 1.0, 0.0, 0.0);

glLightfv(GL\_LIGHT0, GL\_POSITION, position);

glTranslated(0.0, 0.0, 1.5);

glDisable(GL\_LIGHTING);

glColor3f(0.0, 1.0, 1.0);

glutWireCube(0.5); // light source

glEnable(GL\_LIGHTING);

glPopMatrix();

glPopMatrix();

glPopMatrix();

glFlush();

glutSwapBuffers();

}

//Reshape function

void reshape (int w, int h)

{

glViewport (0, 0, (GLsizei) w, (GLsizei) h);

glMatrixMode (GL\_PROJECTION);

glLoadIdentity ();

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

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef (0.0, 0.0, -5.0);

}

// keyboard function

void keyboard (unsigned char key, int x, int y)

{

switch (key) {

case 's':

shoulder1 = (shoulder1 + 5) % 360;

glutPostRedisplay();

break;

case 'S':

shoulder1 = (shoulder1 - 5) % 360;

glutPostRedisplay();

break;

case 'e':

elbow1 = (elbow1 + 5) % 360;

glutPostRedisplay();

break;

case 'E':

elbow1 = (elbow1 - 5) % 360;

glutPostRedisplay();

break;

case 'd':

shoulder2 = (shoulder2 + 5) % 360;

glutPostRedisplay();

break;

case 'D':

shoulder2 = (shoulder2 - 5) % 360;

glutPostRedisplay();

break;

case 'f':

elbow2 = (elbow2 + 5) % 360;

glutPostRedisplay();

break;

case 'F':

elbow2 = (elbow2 - 5) % 360;

glutPostRedisplay();

break;

case 'l':

leg11 = (leg11 + 5) % 360;

glutPostRedisplay();

break;

case 'L':

leg11 = (leg11 - 5) % 360;

glutPostRedisplay();

break;

case 'm':

leg12 = (leg12 + 5) % 360;

glutPostRedisplay();

break;

case 'M':

leg12 = (leg12 - 5) % 360;

glutPostRedisplay();

break;

case 'n':

leg21 = (leg21 + 5) % 360;

glutPostRedisplay();

break;

case 'N':

leg21 = (leg21 - 5) % 360;

glutPostRedisplay();

break;

case 'o':

leg22 = (leg22 + 5) % 360;

glutPostRedisplay();

break;

case 'O':

leg22 = (leg22 - 5) % 360;

glutPostRedisplay();

break;

case 27:

exit(0);

break;

default:

break;

}

}

// Function for using arrow keys for robot rotation

void

special(int key, int x, int y)

{

int i = 0;

switch (key) {

/\* start of view position functions \*/

case GLUT\_KEY\_RIGHT:{

TurnRight();

i++;

}

break;

case GLUT\_KEY\_LEFT:{

TurnLeft();

i++;

}

break;

case GLUT\_KEY\_DOWN:{

TurnForwards();

i++;

}

break;

case GLUT\_KEY\_UP:{

TurnBackwards();

i++;

}

break;

}

if (i)

glutPostRedisplay();

}

// menu selection function

void

menu\_select(int mode)

{

switch (mode) {

case 4:

exit(EXIT\_SUCCESS);

}

}

void

null\_select(int mode)

{

}

//menu function

void glutMenu(void)

{

int glut\_menu[10];

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

glutAddMenuEntry("AT THE SHOULDERS : s,S", 0);

glutAddMenuEntry("AT THE ELBOW : e,E", 0);

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

glutAddMenuEntry("AT THE SHOULDERS : d,D", 0);

glutAddMenuEntry("AT THE ELBOW : f,F", 0);

glut\_menu[4] = glutCreateMenu(null\_select);

glutAddMenuEntry("AT THE HIP : l,L", 0);

glutAddMenuEntry("AT THE KNEES : m,M", 0);

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

glutAddMenuEntry("AT THE HIP : n,N", 0);

glutAddMenuEntry("AT THE KNEES : o,O", 0);

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

glutAddMenuEntry("MOVE THE LEFT BUTTON OF THE MOUSE HORIZONTALLY", 0);

glut\_menu[3] = glutCreateMenu(NULL);

glutAddSubMenu("LEG 1", glut\_menu[4]);

glutAddSubMenu("LEG 2", glut\_menu[5]);

glut\_menu[0] = glutCreateMenu(null\_select);

glutAddSubMenu("HAND 1", glut\_menu[1]);

glutAddSubMenu("HAND 2", glut\_menu[2]);

glut\_menu[7] = glutCreateMenu(null\_select);

glutAddMenuEntry("USE ARROW KEYS", 0);

glutCreateMenu(menu\_select);

glutAddMenuEntry("WHAT CAN I DO??? ", 0);

glutAddSubMenu("ROBOT ROTATION", glut\_menu[7]);

glutAddSubMenu("HAND ROTATION", glut\_menu[0]);

glutAddSubMenu("LEG ROTATION", glut\_menu[3]);

glutAddSubMenu("TO MOVE THE CAMERA",glut\_menu[6] );

glutAddMenuEntry("Quit", 4);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

// main function

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode (GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize (600, 600);

glutInitWindowPosition (100, 100);

glutCreateWindow (argv[0]);

init ();

glutMouseFunc(movelight);

glutMotionFunc(motion);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutKeyboardFunc(keyboard);

glutSpecialFunc(special);

glutMenu();

glutMainLoop();

return 0;

}