VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"JNANA SANGAMA", BELGAUM-590 018

A Mini Project Report On

"WALKING ROBOT"

Submitted in the partial fulfillment of Sixth Semester
COMPUTER GRAPHICS AND VISUALIZATION LABORATORY

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted By

LAAVANYA VIJAYAKUMAR
1VJ20CS024

VIDYA YADAV R
1VJ20CS056

Under the Guidance of

Ms. Renuka N D

Assistance Professor, Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
VIJAYA VITTALA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belagavi | Recognized by Govt. of Karnataka | Approved by AICTE, New Delhi)

KOTHANUR, BENGALURU – 560 077

2022-23

VIJAYA VITTALA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belagavi | Recognized by Govt. of Karnataka | Approved by AICTE, New Delhi)

Kothanur, Bengaluru – 560077

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the mini project entitled ***"WALKING ROBOT"*** is a bonafide work carried out by **LAAVANYA VIJAYAKUMAR [1VJ20CS024], VIDYA YADAV R[1VJ20CS056]** in partial fulfillment for the degree of Bachelor of Engineering in Computer Science & Engineering of the **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**, during the year **2022-23**. It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for the said Degree.

Ms. Renuka, N D B. E, M. Tech Assitant Professor Department of CSE	Prof. Mangala Patil, B.E, M. Tech Head of the Department Department of CSE	Dr. S Rajendra B.E, M.E, Ph. D Principal

Name of the Examiner	**Signature with Date**
1.	1. .....
2.	2. .....

CONTENTS

ACKNOWLEDGEMENT

The satisfactory and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible,whose constant guidance and encouragement crowned the efforts with success.

We would like to express my deepest sense of gratitude to our Mini-Project guide **Miss. Renuka N D** Assistant Professor, Department of Computer Science & Engineering for her constant support and guidance throughout the mini project work.

It gives us immense pleasure to thank **Mrs. Mangala Patel S**, Professor and Headof Department for her constant support and encouragement.

We would like to express our thanks to the Principal **Dr . Rajendra S**, for his encouragement that motivated us for the successful completion of mini projectwork.

We would to profoundly thank **Management of Vijaya Vittala Institute of Technology** for providing such a healthy environment for the successful completion of mini project work.

Last , but not the least , I would hereby acknowledge and thank my parent who have been a source of inspiration and also instrumental in the successful completionof the mini project work.

ABSTRACT

This project is one of the sample projects on Computer Graphics. Though many difficulties were faced during the project as well as many errors occurred, we succeeded to compile and run the program. There may be some limitations on this project as well, so, in the near future we would like to be hopeful in further improvements. We are highly obliged to all helping hands and to all inspirations to make this project successful.

Well, this project will be applicable to most of all. As well as, we are more hopeful for more advice, new ideas and inspiration to make more other projects. To had tried our best to include each and every basic feature of graphics in our projects.

From this very project we were able to achieve various knowledge in computer graphics andalso in logical coding. We refresh our knowledge in C Programming. Moreover, we also gainedan experience of group work, team coordination. We learned how team work is very much important in engineering field.

The aim of the project is to give a demo of a moving object with different colored light sources reflecting on it.

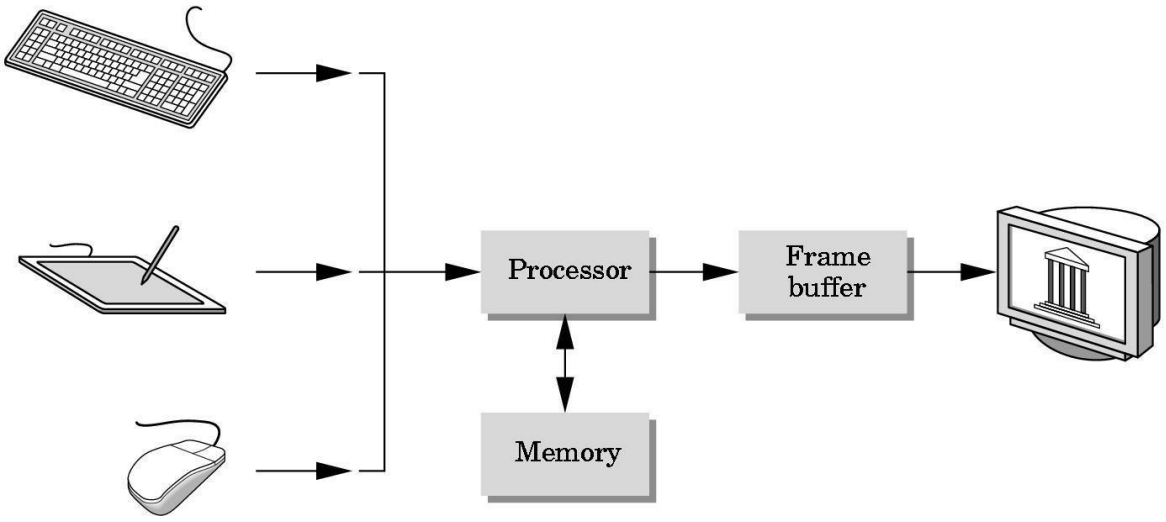
The software is developed for user convenience using the concept of GUI. Here the user is interactive with the simulation through menu and keyboard interface concepts containing multiple windows having the information of keyboard controls, object information and Simulation. This software is an excellent tool that can be used as a simulator to explain about the multilight.

Chapter 1

INTRODUCTION

1.1 Computer Graphics

- Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D Or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly.
- Computers have become a powerful medium for the rapid and economical production of pictures.
- Graphics provide a so natural means of communicating with the computer that they have become widespread.
- Interactive graphics is the most important means of producing pictures since the invention of photography and television .
- We can make pictures of not only the real world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry.
- A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.



1.2 OpenGL Technology

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms.

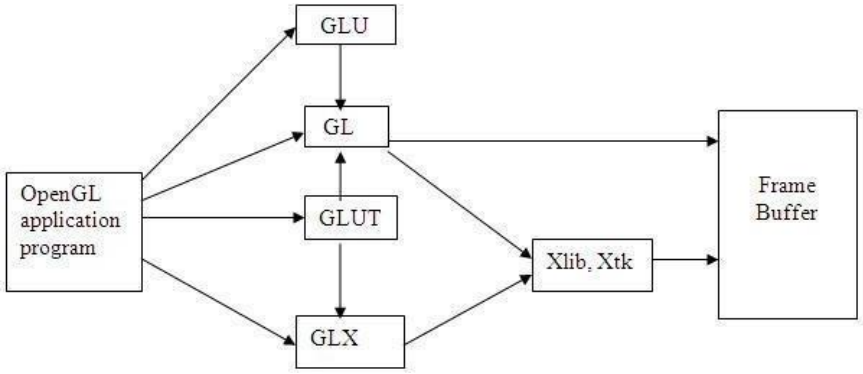
OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

OpenGL Available Everywhere: Supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments.

OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, Open Step, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies.

The OpenGL interface

Our application will be designed to access OpenGL directly through functions in three libraries namely: gl,glu,glut.



Chapter 2

LITERATURE SURVEY

The basic functions like glcolor3f(…); gltotatef(..),gltranslate(..) etc that are most commonly used in the code are taken from the prescribed VTU Text book "INTERACTIVE COMPUTER GRAPHICS" 5th edition by Edward Angel.[1].

The lab programs in the syllabus also serve as a basic template for creating a project. The usageof colors and specifications are taken from the various programs that were taught in the lab.[1]. The VTU prescribed text book serves as a huge database of functions and they are used in theproject. The C++ concepts which are used are being taken from "object oriented programming" by Sourav Sahay.

Some concepts like constructing bowl and fountain are taken from the search results in codecolony.com..

Chapter 3

REQUIREMENTS AND SPECIFICATIONS

3.1 Purpose of the requirements document

The software requirement specification is the official statement of what is required for development of particular project. It includes both user requirements and system requirements. This requirement document is utilized by variety of users starting from project manager who gives project to the engineer responsible for development of project.

It should give details of how to maintain, test, verify and what all the actions to becarried out through

life cycle of project.

Scope of the project

The scope is to use the basic primitives defined in openGL library creating complex objects. We make use of different concepts such as pushmatrix(),translate(), popmatrix(),timerfunction.

Definition

The project **WALKING ROBOT** is created todemonstrate OpenGL's concepts. It encompasses some of the skills learnt in our OpenGLclasses such as pushmatrix(),translate() ,popmatrix(),timer function

.

Acronyms & Abbreviations

OpenGL provides a powerful but primitive set of rendering command, and all higherlevel design must be done in terms of these commands.

OpenGL Utility Toolkit(GLUT):- windows-system-independent toolkit.

References

OpenGL tutorials
Interactive Computer Graphics

3.2 Specific requirements

User Requirement:

- Easy to understand and should be simple.
- The built-in functions should be utilized to maximum extent.
- OpenGL library facilities should be used.

Software Requirements:

- Ubuntu Os.
- Eclipse compiler.

Hardware Requirements:

- Processor- Intel or AMD(Advanced Micro Devices)
- RAM- 512MB(minimum)
- Hard Disk-1MB(minimum)
- Mouse
- Keyboard
- Monitor

Chapter 4

DESIGN

4.1 User Defined Functions

- **myinit():**
  This function initializes light source for ambient, diffuse and specular types.

- **display():**
  This function creates and translates all the objects in a specified location in aparticular order and also rotates the objects in different axes.

glClear(GL_COLOR_BUFFER_BIT);glFlush();

- **timerfunc():**
  This function starts a timer in the event loop that delays the event loop for delaymiiliseconds.

- **MainLoop():**
  This function whose execution will cause the program to begin an event processingloop.

- **PushMatrix():**
  Save the present values of attributes and matrices placing ,or pushing on the top of Stack

.

- **PopMatrix():**
  We can recover them by removing them from stack;

- **Translated();**
  In translate func the variables are components of the displacement vector.

- **main():**
  The execution of the program starts from this function. It initializes the graphics systemand includes many callback functions.

- **PostRedisplay():**
  It ensures that the display will be drawn only once each time the program goes throughthe event loop.

Chapter 5

IMPLEMENTATION

5.1 FUNCTIONS

GL_LINES -

Treats each pair of vertices as an independent line segment.Vertices 2n - 1 and 2n define line n. N/2 lines are drawn.

GL_LINE_LOOP -

Draws a connected group of line segments from the first vertex to the last, then back to the first.Vertices n and n + 1 define line n. The last line, however, is defined by vertices N and N lines aredrawn.

Basic Functions

glPushMatrix, glPopMatrix Function

The glPushMatrix and glPopMatrix functions push and pop the current matrix stack.

SYNTAX: void glPushMatrix ();

void glPopMatrix(void);

glBegin, glEnd Function

The glBegin and glEnd functions delimit the vertices of a primitive or a group of likeprimitives.

SYNTAX:

void glBegin, glEnd (GLenum mode);

PARAMETERS:

- mode -

The primitive or primitives that will be created from vertices presented between glBegin andthe subsequent glEnd. The following are accepted symbolic constants and their meanings:

Transformation Functions

glTranslate Function

The glTranslated and glTranslatef functions multiply the current matrix by a translationmatrix.

SYNTAX:

void glTranslate (x, y, z);

PARAMETERS:

- x, y, z - The x, y, and z coordinates of a translation vector.

Functions used to display glMatrixMode Function

The glMatrixMode function specifies which matrix is the current matrix.

SYNTAX: void glMatrixMode (GLenum mode);

PARAMETERS:

- mode - The matrix stack that is the target for subsequent matrix operations. The modeparameter can assume one of three values:

glLoadIdentity Function

The glLoadIdentity function replaces the current matrix with the identity matrix.

SYNTAX:

void glLoadIdentity(void);

5.2 FUNCTIONS USED TO SET THE VIEWING VOLUME

glOrtho

This function defines orthographic viewing volume with all parameters measured from thecenter of projection. multiply the current matrix by a perspective matrix.

SYNTAX:

void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

PARAMETERES:

- left, right -

Specify the coordinates for the left and right vertical clipping planes.

- bottom, top -

Specify the coordinates for the bottom and top horizontal clipping planes.

- nearVal, farVal -

Specify the distances to the nearer and farther depth clipping planes. These values are negative ifthe plane is to be behind the viewer.

5.3 CALL BACK FUNCTIONS

glutDisplayFunc Function

glutDisplayFunc sets the display callback for the current window.

SYNTAX:

void glutDisplayFunc (void (*func) (void));

glutReshapeFunc Function

glutReshapeFunc sets the reshape callback for the current window.

SYNTAX:

void glutReshapeFunc(void (*func)(int width, int height));

5.4 MAIN FUNCTION

glutInit Function

glutInit is used to initialize the GLUT library.

SYNTAX:

glutInit (int *argc, char **argv);

PARAMETERS:

- argcp - A pointer to the program's unmodified argc variable from main. Upon return, the valuepointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.
- Argv - The program's unmodified argv variable from main. Like argcp, the data for argv willbe updated because glutInit extracts any command line options understood by the GLUT library.
  - glutInit (&argc, argv);

glutInitDisplayMode Function

glutInitDisplayMode sets the initial display mode.

SYNTAX:

void glutInitDisplayMode (unsigned int mode);

PARAMETERS:

- mode - Display mode, normally the bitwise OR-ing of GLUT display mode bit masks. See valuesbelow:

GLUT_RGB: An alias for GLUT_RGBA.

GLUT_DOUBLE: Bit mask to select a double buffered window. This overridesGLUT_SINGLE.

GLUT_DEPTH: Bit mask to select a window with a depth buffer.

glutMainLoop Function

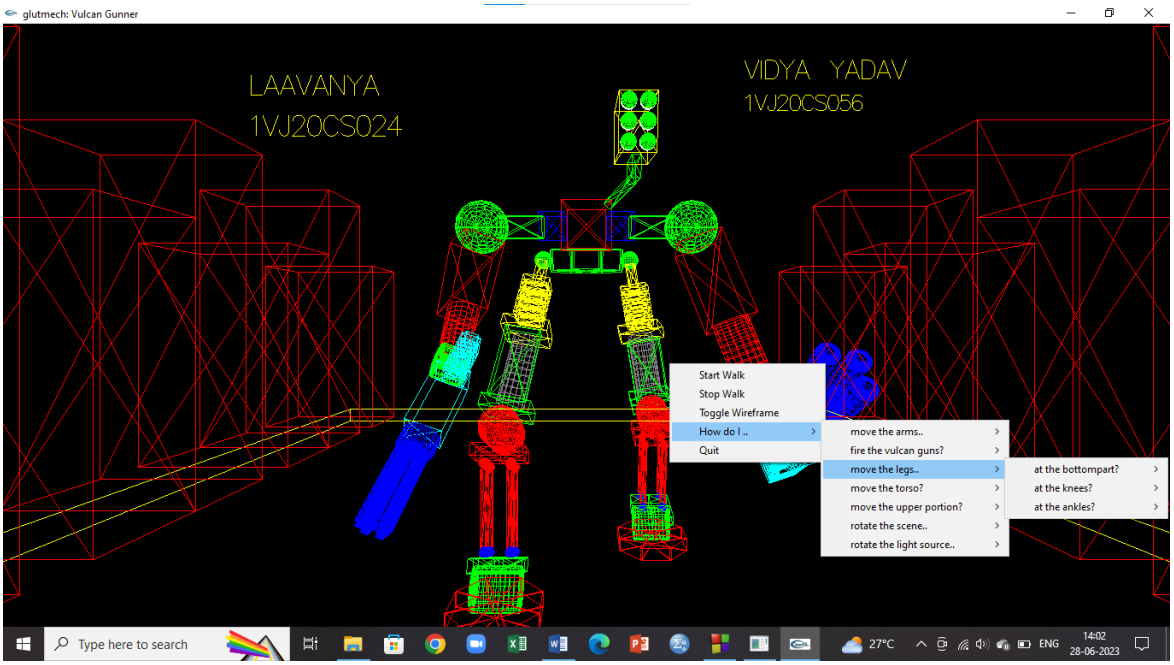
glutMainLoop enters the GLUT event processing loop.

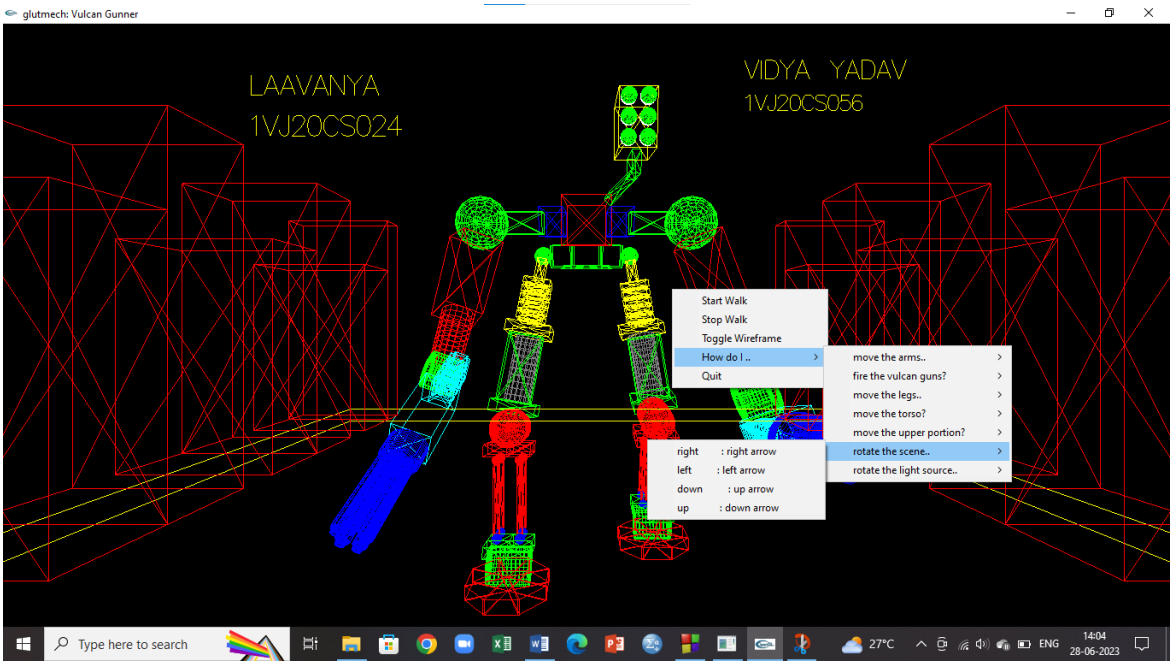
SYNTAX:

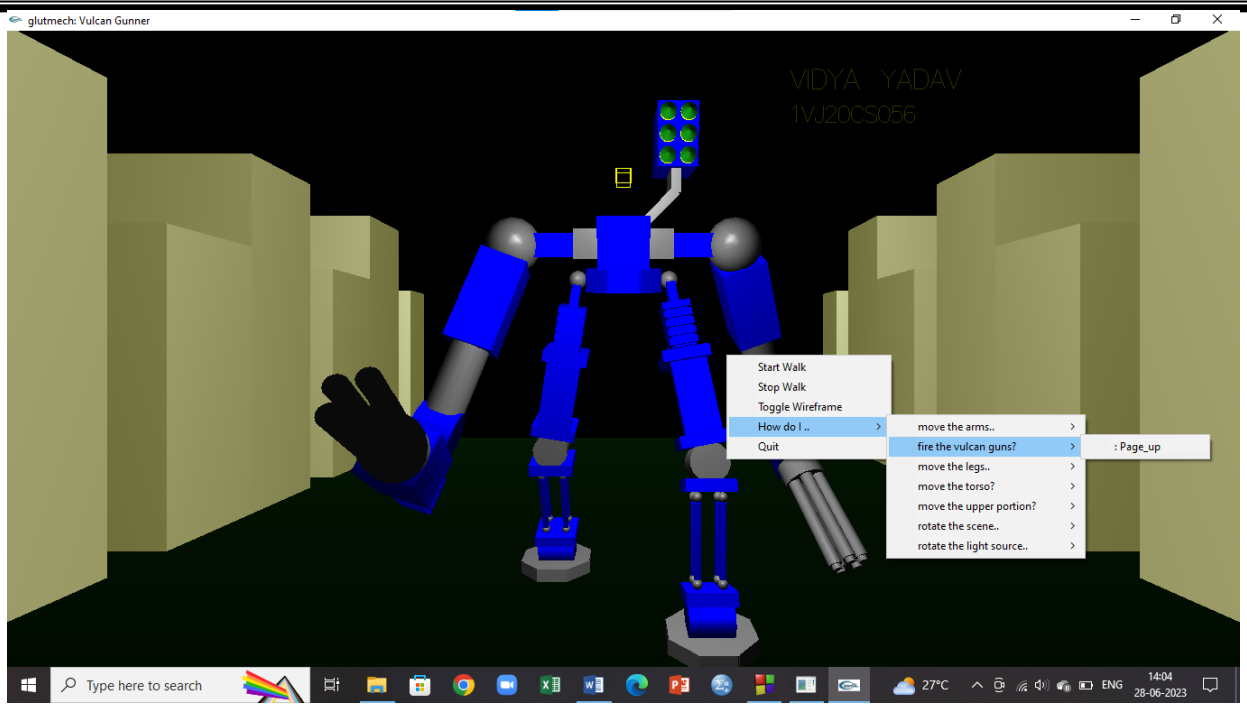
void glutMainLoop(void);

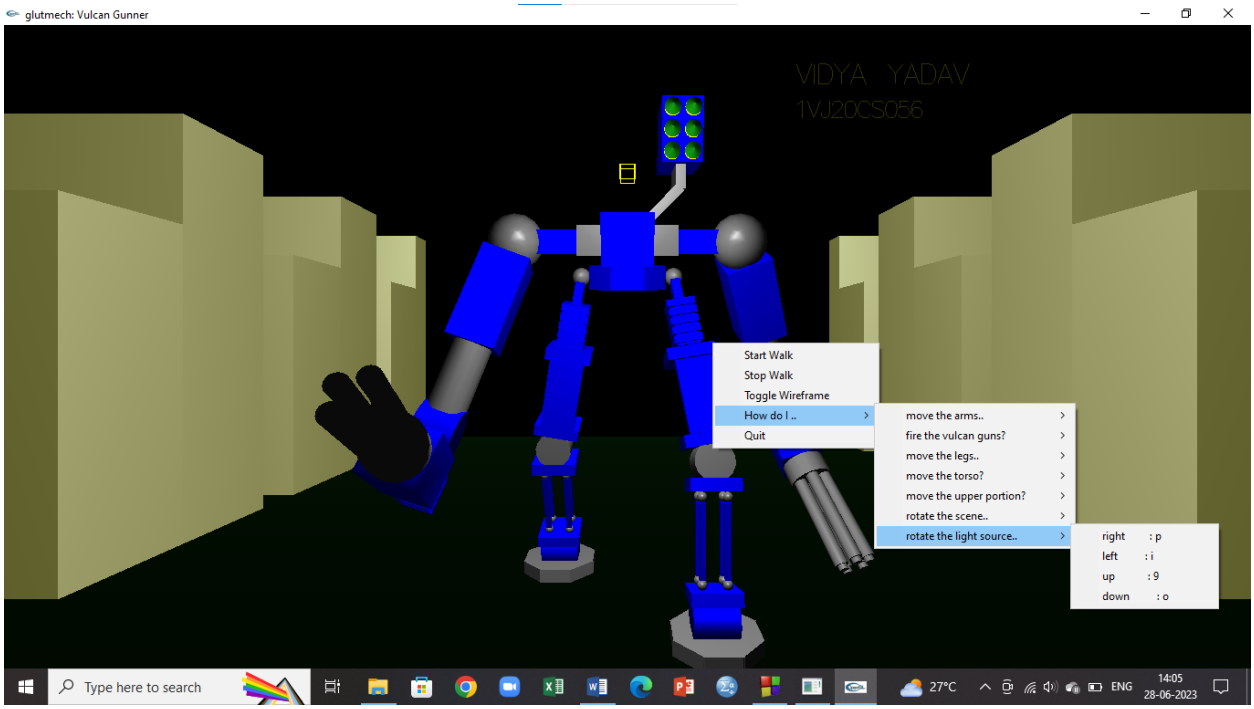
Chapter 6

SNAP SHOTS






glutmech: Vulcan Gunner
VIDYA YADAV
1VJ20CS056
Start Walk
Stop Walk
Toggle Wireframe
How do I ..
Quit
move the arms..
fire the vulcan guns?
move the legs..
move the torso?
move the upper portion?
rotate the scene..
rotate the light source..
: Page_up
Type here to search
27°C
ENG
14:04
28-06-2023


glutmech: Vulcan Gunner
VIDYA YADAV
1VJ20CS056
Start Walk
Stop Walk
Toggle Wireframe
How do I ..
Quit
move the arms..
fire the vulcan guns?
move the legs..
move the torso?
move the upper portion?
rotate the scene..
rotate the light source..
right : p
left : i
up : 9
down : o
Type here to search
27°C
ENG
14:05
28-06-2023

APPENDIX

```
#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
/* end of compilation conditions */

/* start various header files needed */
#include <windows.h>
#include <stdlib.h>
#include <math.h>
#define GLUT
#define GLUT_KEY
#define GLUT_SPEC
#include <GL/glut.h>
/* end of header files */
#define TEXTID      19
void DrawTextXY(double,double,double,double,char *);
/* start of display list definitions */
#define SOLID_MECH_TORSO            1
#define SOLID_MECH_HIP          2
#define SOLID_MECH_SHOULDER     3
#define SOLID_MECH_UPPER_ARM    4
#define SOLID_MECH_FOREARM          5
#define SOLID_MECH_UPPER_LEG        6
#define SOLID_MECH_FOOT             7
#define SOLID_MECH_ROCKET           8
#define SOLID_MECH_VULCAN           9
#define SOLID_ENVIRO                10
/* end of display list definitions */
/* start of motion  variables */
#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
 /* foot1v[] = {} foot2v[] = {} */ ;

/* end of motion variables */

/* start of material definitions */
#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 Heel2Add(void)
{
  heel2 = (heel2 + 3) % 360;
}
void Heel2Subtract(void)
```

```
{
  heel2 = (heel2 - 3) % 360;
}
void Ankle1Add(void)
{
  ankle1 = (ankle1 + 3) % 360;
}
void Ankle1Subtract(void)
{
  ankle1 = (ankle1 - 3) % 360;
}
void Ankle2Add(void)
{
  ankle2 = (ankle2 + 3) % 360;
}
void Ankle2Subtract(void)
{
  ankle2 = (ankle2 - 3) % 360;
}
void RotateAdd(void)
{
  pivot = (pivot + 3) % 360;
}
void RotateSubtract(void)
{
  pivot = (pivot - 10) % 360;
}
void MechTiltSubtract(void)
{
  tilt = (tilt - 10) % 360;
}
void MechTiltAdd(void)
{
  tilt = (tilt + 10) % 360;
}
void elbow1Add(void)
{
  elbow1 = (elbow1 + 2) % 360;
}
void elbow1Subtract(void)
{
  elbow1 = (elbow1 - 2) % 360;
}
void elbow2Add(void)
{
  elbow2 = (elbow2 + 2) % 360;
}
void elbow2Subtract(void)
{
  elbow2 = (elbow2 - 2) % 360;
}
void shoulder1Add(void)
{
  shoulder1 = (shoulder1 + 5) % 360;
}
void shoulder1Subtract(void)
```

```
{
  shoulder1 = (shoulder1 - 5) % 360;
}
void shoulder2Add(void)
{
  shoulder2 = (shoulder2 + 5) % 360;
}
void shoulder2Subtract(void)
{
  shoulder2 = (shoulder2 - 5) % 360;
}
void shoulder3Add(void)
{
  shoulder3 = (shoulder3 + 5) % 360;
}
void shoulder3Subtract(void)
{
  shoulder3 = (shoulder3 - 5) % 360;
}
void shoulder4Add(void)
{
  shoulder4 = (shoulder4 + 5) % 360;
}
void shoulder4Subtract(void)
{
  shoulder4 = (shoulder4 - 5) % 360;
}
void lat1Raise(void)
{
  lat1 = (lat1 + 5) % 360;
}
void lat1Lower(void)
{
  lat1 = (lat1 - 5) % 360;
}
void lat2Raise(void)
{
  lat2 = (lat2 + 5) % 360;
}
void lat2Lower(void)
{
  lat2 = (lat2 - 5) % 360;
}
void FireCannon(void)
{
  fire = (fire + 20) % 360;
}
void RaiseLeg1Forward(void)
{
  hip11 = (hip11 + 3) % 360;
}
void LowerLeg1Backwards(void)
{
  hip11 = (hip11 - 3) % 360;
}
void RaiseLeg1Outwards(void)
```

```
{
  hip12 = (hip12 + 10) % 360;
}
void LowerLeg1Inwards(void)
{
  hip12 = (hip12 - 10) % 360;
}
void RaiseLeg2Forward(void)
{
  hip21 = (hip21 + 3) % 360;
}
void LowerLeg2Backwards(void)
{
  hip21 = (hip21 - 3) % 360;
}
void RaiseLeg2Outwards(void)
{
  hip22 = (hip22 + 10) % 360;
}
void LowerLeg2Inwards(void)
{
  hip22 = (hip22 - 10) % 360;
}
/* end of body motion functions */

/* start of light source position functions */
void TurnRight(void)
{
  turn = (turn - 10) % 360;
}

void TurnLeft(void)
{
  turn = (turn + 10) % 360;
}

void TurnForwards(void)
{
  turn1 = (turn1 - 10) % 360;
}

void TurnBackwards(void)
{
  turn1 = (turn1 + 10) % 360;
}
void LightTurnRight(void)
{
  lightturn = (lightturn + 10) % 360;
}

void LightTurnLeft(void)
{
  lightturn = (lightturn - 10) % 360;
}

void LightForwards(void)
```

```
{
  lightturn1 = (lightturn1 + 10) % 360;
}

void LightBackwards(void)
{
  lightturn1 = (lightturn1 - 10) % 360;
}

/* end of light source position functions */
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();
}
/* start of geometric shape functions */
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;
  }
 }
}

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);
  }
}

/* end of geometric shape functions */
#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);
#endif
  glColor3f(1.0, 0.0, 0.0);//torso red color
  Box(1.0, 1.0, 3.0, solid);
  glTranslatef(0.75, 0.0, 0.0);
#ifdef LIGHT
  SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
#endif
  glColor3f(0.0, 0.0, 1.0);//torso blue color
  Box(0.5, 0.6, 2.0, solid);
  glTranslatef(-1.5, 0.0, 0.0);
```

```
  Box(0.5, 0.6, 2.0, solid);
  glTranslatef(0.75, 0.0, 0.0);
  glEndList();
}

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);
#ifdef LIGHT
  SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
#endif
  glColor3f(1.0, 0.0, 0.0);//arm red
  if (!solid)
    gluQuadricDrawStyle(qobj, GLU_LINE);
  gluCylinder(qobj, 0.4, 0.4, 1.5, 16, 10);
#ifdef LIGHT
  SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);
#endif
  glColor3f(0.0, 1.0, 0.0);// arm joint green
  glRotatef(-90.0, 1.0, 0.0, 0.0);
  glTranslatef(-0.4, -1.85, 0.0);
  glRotatef(90.0, 0.0, 1.0, 0.0);
  for (i = 0; i < 2; i++) {
    if (!solid)
      gluQuadricDrawStyle(qobj, GLU_LINE);
    if (i)
      gluCylinder(qobj, 0.5, 0.5, 0.8, 16, 10);
    else
      gluCylinder(qobj, 0.2, 0.2, 0.8, 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, 0.8);
    gluDisk(qobj, 0.2, 0.5, 16, 10);
    if (i)
      glTranslatef(0.0, 0.0, -0.8);
  }
  glScalef(-1.0, 1.0, 1.0);
  glRotatef(-90.0, 0.0, 1.0, 0.0);
  glTranslatef(0.4, 2.9, 0.0);
  glEndList();
}

void VulcanGun(char solid)
{
  int i;

  glNewList(SOLID_MECH_VULCAN, GL_COMPILE);
```

```
#ifdef LIGHT
  SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
#endif
  glColor3f(0.0, 0.0, 1.0);//gun color

  if (!solid) {
    gluQuadricDrawStyle(qobj, GLU_LINE);
  }
  gluCylinder(qobj, 0.5, 0.5, 0.5, 16, 10);

  glTranslatef(0.0, 0.0, 0.5);
 gluDisk(qobj, 0.0, 0.5, 16, 10);


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

    glRotatef(72.0, 0.0, 0.0, 1.0);
    glTranslatef(0.0, 0.3, 0.0);

    if (!solid) {
      gluQuadricDrawStyle(qobj, GLU_LINE);
    }
    gluCylinder(qobj, 0.15, 0.15, 2.0, 16, 10);
    gluCylinder(qobj, 0.06, 0.06, 2.0, 16, 10);
    glTranslatef(0.0, 0.0, 2.0);
    gluDisk(qobj, 0.1, 0.15, 16, 10);
    gluCylinder(qobj, 0.1, 0.1, 0.1, 16, 5);
    glTranslatef(0.0, 0.0, 0.1);
    gluDisk(qobj, 0.06, 0.1, 16, 5);
      glTranslatef(0.0, -0.3, -2.1);
    }
  glEndList();
}

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();
}
```

```
void UpperLeg(char solid)
{
  int i;

  glNewList(SOLID_MECH_UPPER_LEG, GL_COMPILE);
#ifdef LIGHT
  SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);
#endif
  glColor3f(1.0, 1.0, 0.0);//color yellow
  if (!solid) {
    gluQuadricDrawStyle(qobj, GLU_LINE);
  }
  glTranslatef(0.0, -1.0, 0.0);
  Box(0.4, 1.0, 0.7, solid);
  glTranslatef(0.0, -0.65, 0.0);
  for (i = 0; i < 5; i++) {
    Box(1.2, 0.3, 1.2, solid);
    glTranslatef(0.0, -0.2, 0.0);
    Box(1.0, 0.1, 1.0, solid);
    glTranslatef(0.0, -0.2, 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);
#ifdef LIGHT
      SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);
#endif
      glColor3f(1.0, 0.0, 0.0);//red
      Box(1.0, 0.5, 1.0, solid);
      glTranslatef(0.0, -0.45, 0.0);
#ifdef LIGHT
      SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
#endif
      glColor3f(1.0, 0.0, 0.0);
#ifdef SPHERE
      if (!solid)
        glutWireSphere(0.2, 16, 10);
      else
        glutSolidSphere(0.2, 16, 10);
#endif
      if (leg)
        glRotatef((GLfloat) heel1, 1.0, 0.0, 0.0);
      else
        glRotatef((GLfloat) heel2, 1.0, 0.0, 0.0);
      /* glTranslatef(0.0, -0.2, 0.0); */
      glTranslatef(0.0, -1.7, 0.0);
#ifdef LIGHT
      SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);
#endif
      glColor3f(1.0, 0.0, 0.0);//color of the below leg( pillers)
```

```
    Box(0.25, 3.0, 0.25, solid);
    glTranslatef(0.0, -1.7, 0.0);
#ifdef LIGHT
    SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
#endif
    glColor3f(0.0, 0.0, 1.0);//joints
#ifdef SPHERE
    if (!solid)
      glutWireSphere(0.2, 16, 10);
    else
      glutSolidSphere(0.2, 16, 10);
#endif
    if (leg)
      glRotatef((GLfloat) - heel1, 1.0, 0.0, 0.0);
    else
      glRotatef((GLfloat) - heel2, 1.0, 0.0, 0.0);
    glTranslatef(0.0, -0.45, 0.0);
#ifdef LIGHT
    SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);
#endif
    glColor3f(0.0, 1.0, 0.0);// leg n foot joints color
    Box(1.0, 0.5, 1.0, solid);
    if (!k && !l) {
      int j;

      glTranslatef(-0.4, -0.8, 0.5);
      if (leg)
        glRotatef((GLfloat) ankle1, 1.0, 0.0, 0.0);
      else
        glRotatef((GLfloat) ankle2, 1.0, 0.0, 0.0);
      glRotatef(90.0, 0.0, 1.0, 0.0);
      if (!solid)
        gluQuadricDrawStyle(qobj, GLU_LINE);
      gluCylinder(qobj, 0.8, 0.8, 1.8, 16, 10);
      for (j = 0; j < 2; j++) {
        if (!solid)
          gluQuadricDrawStyle(qobj, GLU_LINE);
        if (j) {
          glScalef(-1.0, 1.0, 1.0);
          glTranslatef(0.0, 0.0, 1.8);
        }
        gluDisk(qobj, 0.0, 0.8, 16, 10);
        if (j)
          glTranslatef(0.0, 0.0, -1.8);
      }
      glScalef(-1.0, 1.0, 1.0);
      glRotatef(-90.0, 0.0, 1.0, 0.0);
      glTranslatef(0.95, -0.8, 0.0);
      glCallList(SOLID_MECH_FOOT);
    }
    glPopMatrix();
  }
 }
}

void RocketPod(char solid)
```

```
{

  int i, j, k = 0;

  glNewList(SOLID_MECH_ROCKET, GL_COMPILE);
#ifdef LIGHT
  SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
#endif
  glColor3f(0.0, 1.0, 0.0);//rocket port color
  glScalef(0.4, 0.4, 0.4);
  glRotatef(45.0, 0.0, 0.0, 1.0);
  glTranslatef(1.0, 0.0, 0.0);
  Box(2.0, 0.5, 3.0, solid);
  glTranslatef(1.0, 0.0, 0.0);
  glRotatef(45.0, 0.0, 0.0, 1.0);
  glTranslatef(0.5, 0.0, 0.0);
  Box(1.2, 0.5, 3.0, solid);
  glTranslatef(2.1, 0.0, 0.0);
  glRotatef(-90.0, 0.0, 0.0, 1.0);
#ifdef LIGHT
  SetMaterial(mat_specular, mat_ambient, mat_diffuse, mat_shininess);
#endif
  glColor3f(1.0, 1.0, 0.0);
  Box(2.0, 3.0, 4.0, solid);
  glTranslatef(-0.5, -1.0, 1.3);
  for (i = 0; i < 2; i++) {
    for (j = 0; j < 3; j++) {
      if (!solid) {
        gluQuadricDrawStyle(qobj, GLU_LINE);
      }
      glTranslatef(i, j, 0.6);
#ifdef LIGHT
      SetMaterial(mat_specular3, mat_ambient3, mat_diffuse3, mat_shininess3);
#endif
      glColor3f(1.0, 1.0, 1.0);
      gluCylinder(qobj, 0.4, 0.4, 0.3, 16, 10);
      glTranslatef(0.0, 0.0, 0.3);
#ifdef LIGHT
      SetMaterial(mat_specular4, mat_ambient4, mat_diffuse4, mat_shininess4);
#endif
      glColor3f(0.0, 1.0, 0.0);
      gluCylinder(qobj, 0.4, 0.0, 0.5, 16, 10);
      k++;
      glTranslatef(-i, -j, -0.9);
    }
  }
  glEndList();
}

void Enviro(char solid)
{

  int i, j;

  glNewList(SOLID_ENVIRO, GL_COMPILE);
  SetMaterial(mat_specular4, mat_ambient4, mat_diffuse4, mat_shininess4);
```

```
 glColor3f(1.0, 1.0, 0.0);//out line of the walking path
 Box(20.0, 0.5, 30.0, solid);
 SetMaterial(mat_specular4, mat_ambient3, mat_diffuse2, mat_shininess);
 glColor3f(1.0, 0.0, 0.0);//the surrounding area color
 glTranslatef(0.0, 0.0, -10.0);
 for (j = 0; j < 6; j++) {
   for (i = 0; i < 2; i++) {
     if (i)
       glScalef(-1.0, 1.0, 1.0);
     glTranslatef(10.0, 4.0, 0.0);
     Box(4.0, 8.0, 2.0, solid);
     glTranslatef(0.0, -1.0, -3.0);
     Box(4.0, 6.0, 2.0, solid);
     glTranslatef(-10.0, -3.0, 3.0);
   }
   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++) {
    glPushMatrix();
    if (j) {
      glScalef(-0.5, 0.5, 0.5);
      leg = 1;
    } else {
      glScalef(0.5, 0.5, 0.5);
      leg = 0;
    }
    glTranslatef(2.0, -1.5, 0.0);
    if (j) {
      glRotatef((GLfloat) hip11, 1.0, 0.0, 0.0);
      glRotatef((GLfloat) hip12, 0.0, 0.0, 1.0);
    } else {
      glRotatef((GLfloat) hip21, 1.0, 0.0, 0.0);
      glRotatef((GLfloat) hip22, 0.0, 0.0, 1.0);
    }
    glTranslatef(0.0, 0.3, 0.0);
#ifdef UPPER_LEG
    glPushMatrix();
    glCallList(SOLID_MECH_UPPER_LEG);
    glPopMatrix();
#endif
    glTranslatef(0.0, -8.3, -0.4);
    if (j)
      glRotatef((GLfloat) - hip12, 0.0, 0.0, 1.0);
    else
      glRotatef((GLfloat) - hip22, 0.0, 0.0, 1.0);
    glTranslatef(-0.5, -0.85, -0.5);
#ifdef LOWER_LEG
    LowerLeg(1);
#endif
    glPopMatrix();
  }
```

```
}

void display(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
 //glClearColor(1.0, 1.0, 1.0, 1.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glEnable(GL_DEPTH_TEST);
  glPushMatrix();
  glRotatef((GLfloat) turn, 0.0, 1.0, 0.0);
  glRotatef((GLfloat) turn1, 1.0, 0.0, 0.0);
#ifdef LIGHT
  if (solid_part) {
    glPushMatrix();
    lighting();
    glPopMatrix();
  } else
    disable();
#endif
#ifdef DRAW_MECH
  glPushMatrix();
  glTranslatef(0.0, elevation, 0.0);
  DrawMech();
  glPopMatrix();
#endif
#ifdef DRAW_ENVIRO
  glPushMatrix();
  if (distance >= 20.136)
    distance = 0.0;
  glTranslatef(0.0, -5.0, -distance);
  glCallList(SOLID_ENVIRO);
  glTranslatef(0.0, 0.0, 10.0);
  glCallList(SOLID_ENVIRO);
  glPopMatrix();
#endif
  glPopMatrix();

 glLoadName(TEXTID);
 glColor3f(1,1,0);
 DrawTextXY(-2.5,0.2,2.0,0.0015,"1ce09cs009");
 DrawTextXY(-2.5,0.5,2.0,0.0015,"ARMUGAM(AARU)");
 DrawTextXY(2.5,2.2,-2.0,0.0025,"1ce09cs002");
 DrawTextXY(2.5,2.7,-2.0,0.0030,"ABDUL");
  glFlush();
  glutSwapBuffers();

}


void myinit(void)
{
  char i = 1;

  qobj = gluNewQuadric();
#ifdef LIGHT
  SetMaterial(mat_specular2, mat_ambient2, mat_diffuse2, mat_shininess2);
```

```
#endif
  glEnable(GL_DEPTH_TEST);
  MechTorso(i);
  MechHip(i);
  Shoulder(i);
  RocketPod(i);
  UpperArm(i);
  ForeArm(i);
  UpperLeg(i);
  Foot(i);
  VulcanGun(i);
  Enviro(i);
}

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);
  glLoadIdentity();
  glTranslatef(0.0, 1.2, -5.5);  /* viewing transform  */
}

#ifdef ANIMATION
 void animation_walk(void)

{
  float angle;
  static int step;

  if (step == 0 || step == 2) {

    if (frame >= 0.0 && frame <= 21.0) {
      if (frame == 0.0)
        frame = 3.0;
      angle = (180 / M_PI) * (acos(((cos((M_PI / 180) * frame) * 2.043) + 1.1625) / 3.2059));
      if (frame > 0) {
        elevation = -(3.2055 - (cos((M_PI / 180) * angle) * 3.2055));
      } else
        elevation = 0.0;
      if (step == 0) {
        hip11 = -(frame * 1.7);
        if (1.7 * frame > 15)
          heel1 = frame * 1.7;
        heel2 = 0;
        ankle1 = frame * 1.7;
        if (frame > 0)
          hip21 = angle;
        else
          hip21 = 0;
        ankle2 = -hip21;
        shoulder1 = frame * 1.5;
        shoulder2 = -frame * 1.5;
        elbow1 = frame;
```

```
      elbow2 = -frame;
    } else {
      hip21 = -(frame * 1.7);
      if (1.7 * frame > 15)
        heel2 = frame * 1.7;
      heel1 = 0;
      ankle2 = frame * 1.7;
      if (frame > 0)
        hip11 = angle;
      else
        hip11 = 0;
      ankle1 = -hip11;
      shoulder1 = -frame * 1.5;
      shoulder2 = frame * 1.5;
      elbow1 = -frame;
      elbow2 = frame;
    }
    if (frame == 21)
      step++;
    if (frame < 21)
      frame = frame + 3.0;
  }
}
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);
  switch (key) {
    /* start arm control functions */
  case 'q':{
     shoulder2Subtract();
     i++;
            i++;
   }
   break;
  case 'a':{
     shoulder2Add();
     i++;
   }
   break;
  case 'w':{
     shoulder1Subtract();
     i++;
   }
   break;
  case 's':{
     shoulder1Add();
     i++;
   }
   break;
  case '2':{
     shoulder3Add();
     i++;
   }
   break;
  case '1':{
```

```
    shoulder4Add();
    i++;
  }
  break;
case '4':{
    shoulder3Subtract();
    i++;
  }
  break;
case '3':{
    shoulder4Subtract();
    i++;
  }
  break;

case 'z':{
    lat2Raise();
    i++;
  }
  break;
case 'Z':{
    lat2Lower();
    i++;
  }
  break;
case 'x':{
    lat1Raise();
    i++;
  }
  break;
case 'X':{
    lat1Lower();
    i++;
  }
  break;

case 'A':{
    elbow2Add();
    i++;
  }
  break;
case 'Q':{
    elbow2Subtract();
    i++;
  }
  break;
case 'S':{
    elbow1Add();
    i++;
  }
  break;
case 'W':{
    elbow1Subtract();
    i++;
  }
  break;
```

```
  /* end of arm control functions */

  /* start of torso control functions */
case 'd':{
    RotateAdd();
    i++;
  }
  break;
case 'g':{
    RotateSubtract();
    i++;
  }
  break;
case 'r':{
    MechTiltAdd();
    i++;
  }
  break;
case 'f':{
    MechTiltSubtract();
    i++;
  }
  break;
  /* end of torso control functions */

  /* start of leg control functions */
case 'h':{
    RaiseLeg2Forward();
    i++;
  }
  break;
case 'y':{
    LowerLeg2Backwards();
    i++;
  }
  break;
case 'Y':{
    RaiseLeg2Outwards();
    i++;
  }
  break;
case 'H':{
    LowerLeg2Inwards();
    i++;
  }
  break;

case 'j':{
    RaiseLeg1Forward();
    i++;
  }
  break;
case 'u':{
    LowerLeg1Backwards();
    i++;
  }
```

```
  break;
case 'U':{
    RaiseLeg1Outwards();
    i++;
  }
  break;
case 'J':{
    LowerLeg1Inwards();
    i++;
  }
  break;

case 'N':{
    Heel2Add();
    i++;
  }
  break;
case 'n':{
    Heel2Subtract();
    i++;
  }
  break;
case 'M':{
    Heel1Add();
    i++;
  }
  break;
case 'm':{
    Heel1Subtract();
    i++;
  }
  break;

case 'k':{
    Ankle2Add();
    i++;
  }
  break;
case 'K':{
    Ankle2Subtract();
    i++;
  }
  break;
case 'l':{
    Ankle1Add();
    i++;
  }
  break;
case 'L':{
    Ankle1Subtract();
    i++;
  }
  break;
  /* end of leg control functions */

  /* start of light source position functions */
```

```
  case 'p':{
     LightTurnRight();
     i++;
    }
    break;
  case 'i':{
     LightTurnLeft();
     i++;
    }
    break;
  case 'o':{
     LightForwards();
     i++;
    }
    break;
  case '9':{
     LightBackwards();
     i++;
    }
    break;

    /* end of light source position functions */
  }
  if (i)
    glutPostRedisplay();
}

#endif

#ifdef GLUT_SPEC
/* ARGSUSED1 */
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;
    /* end of view postions functions */
    /* start of misecllleneous functions */
  case GLUT_KEY_PAGE_UP:{
      FireCannon();
      i++;
    }
    break;
    /* end of miscelleneous functions */

  }
  if (i)
    glutPostRedisplay();
}

#endif
#endif
void menu_select(int mode)
{
  switch (mode) {
#ifdef ANIMATION
  case 1:
   glutIdleFunc(animation);
    break;
#endif
  case 2:
    glutIdleFunc(NULL);
    break;
  case 3:
    Toggle();
    glutPostRedisplay();
    break;
  case 4:
    exit(EXIT_SUCCESS);

  }
}

/* ARGSUSED */
void null_select(int mode)
{
}

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

  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();
  /* end of glut windowing and control functions */
#endif
  return 0;             /* ANSI C requires main to return int. */
}
```

REFERENCES

1) OpenGL Programming Guide (Addison-Wesley Publishing Company)

2) The OpenGL Utility Toolkit (GLUT)Programming Interface

-API Version 3 BY MARK J. KILGARD

3) Interactive computer graphics

-A top-down approach by using Open GL by EDWARD ANGEL