INDEX:

CHAPTER 1:

INTRODUCTION

1.1: INTRODUCTION TO STEAM ENGINE.

The aim of this project is to create a STEAM ENGINE. The Engine is made up of a Piston, Engine Pole, Cylinder Head, Flywheel, Crank Bell and a Crank. The viewer is allowed to rotate the Engine's Crank either in clock wise or in anti-clock wise direction. The viewer can also slow up or slow down the Crank speed.

First a Piston, Engine Pole, Cylinder Head, Flywheel, Crank Bell and a Crank is created using myCylinder( ) function. The main primitives used inside the myCylinder( ) function to create a Cylinder is gluCylinder( ) and gluDisk( ) . So every time, myCylinder( ) function is called inside the functions used to create Piston, Engine Pole, Cylinder Head, Flywheel, Crank Bell and Crank. The parts mentioned above are combined to form a Steam Engine image. We can make Steam Engine transparent and display. In display function, at first it clears the drawing buffer and if transparency is set, displays the model twice, first time accepting those fragments with a ALPHA value of 1 only, then with DEPTH_BUFFER writing disabled for those with other values. Initially when the animation is not called, the crank angle will not change and the window is idle. When called increments the crank angle by ANGLE_STEP, updates the head angle and notifies the system that the screen needs to be updated. When a menu option has been selected, it translates the menu item identifier into a keystroke, then calls the keyboard function. A menu will be associated with the mouse too. The viewer can also see the shaded and textured steam engine.

The controls are:-

1. 'a' - > To rotate crank anti-clock wise.
2. 'z' - > To rotate crank clock wise.
3. '+' and '-' - > To speed up and speed down
4. 'o' -> Transparency.
5. '0' and 1' - > Right light and Left light respectively
6. 's' and 't' - > Shading and Texture respectively

1.2:INTRODUCTION TO OPENGL

Most of our application will be designed to access OpenGL directly through functions in three libraries. Functions in the main GL (or OpenGL in windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in windows). The second is the OpenGL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all

OpenGL implementations; functions in the GLU library begin with letters glu.
To interface with the window system and to get input from e xternal devices into our programs, we need at least one more system-specific library that provides the "glue" between the window system and OpenGL. For the X window system, this library is functionality that should be expected in any modern windowing system.

Fig 2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT li brary for other window systems.

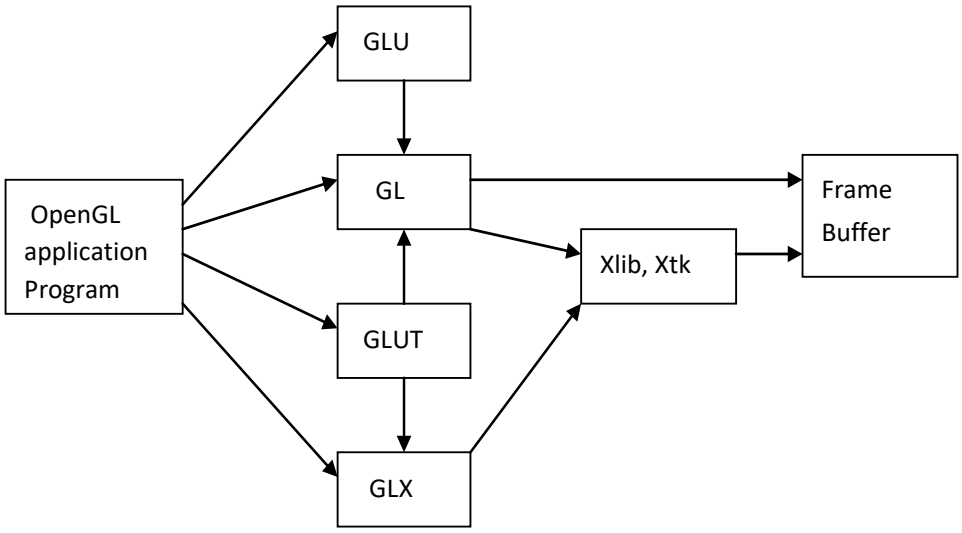


Fig 1.2 Library organization

1.2.1:OpenGL Command Syntax:

OpenGL commands use the prefix gl and initial capital letters for each word making up the command name. Similarly, OpenGL defined constants begin with GL_, use all capital letters and use underscores to separate words (like GL_COLOR_BUFFER_BIT)

CHAPTER 2:

LITERATURE SURVEY

Provides the description of the following functions.

void glScalef(TYPE sx, TYPE sy, TYPE sz)

alters the current matrix by a scaling of (sx, sy, sz). TYPE here is GLfloat.

Here in the above considered example we use scaling to minimize the length of the curve at each iteration. For this curve we use the scale factor to be 3 units because we substitute a line by 4 lines in each iteration.

void glRotatef(TYPE angle, TYPE dx, TYPE dy, TYPE dz)

alters the current matrix by a rotation of angle degrees about the axis(dx, dy, dz). TYPE here is GLfloat.

For a Koch curve we rotate by 60° about the z-axis.

void glTranslatef(TYPE x, TYPE y, TYPE z)

alters the current matrix by a displacement of (x, y, z). TYPE here is GLfloat.

We need to translate to display the new position of the line from the old position and also to go out to the beginning of the next side while drawing.

void glLoadIdentity()

sets the current transformation matrix to an identity matrix.

void glPushMatrix()

pushes to the matrix stack corresponding to the current matrix mode.

void glPopMatrix()

pops from the matrix stack corresponding to the current matrix mode.

void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)

defines a two-dimensional viewing rectangle in the plane z=0.

void glutMouseFunc(myMouse)

refers to the mouse callback function. The function to callback is defined as

```
void myMouse(int button, int state, int x
 {
 if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
if (glutGetModifiers() & GLUT_ACTIVE_SHIFT)
decrease a level of recursion else increase a level of recursion
}
```

Here mouse interface is given to increase a level of recursion by clicking mouse button and also to decrease a level of recursion by doing the same holding the shift on the keyboard.

void glutKeyboardFunc(myKey)

refers to the keyboard callback function. The function to callback is defined as

```
void myKey(unsigned char key, int x, int y)
{
 if (c == 't')
exit
if (c == 's')
//STATEMENTS and repeat when finished
}
```

Here keyboard interface is given to quit, the user can quit by pressing ‘q’ and to see next example of the implementation, the user should press ‘n’.

void glutSwapBuffers()

swaps the front and back buffers.

User defined functions are used to color the curves in a standard cycle rainbow manner which becomes very easy for the user to identify the levels of recursion for the curves.

void glutInit(int *argc, charargv)**

Initializes GLUT< the arguments from main are passed in and can by the application.

void glutCreateWindow(char *title)

Creates a window on the display. The string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

void glutInitDisplaymode(unsigned int mode)

Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model(GLUT_RGB<GLUT_INDEX) and buffering ((GLUT_SINGLE<GLUT_DOUBLE).

void glutInitWindowSize(int width, int heights)

Specifies the initial height and width of the window in pixels.

void glutInitWindowPosition(int x, int y)

Specifies the initial position of the top-left corner of the window in pixels.

void glViewport(int x, int y ,GLsize i width, GLsize i height)

Specifies a width * height viewport in pixels whose lower-left corner is at (x,y) measured from he origin of the window.

void glutMainLoop()

Cause the program to enter an event –processing loop.it should be the statement in main.

void glutPostRedisplay()

Requests that the display callback be executed after the current callback returns.

CHAPTER 3:

OBJECTIVES

The project simulates the working of a steam engine. It illustrates how the linear motion of the piston is converted into rotary motion. The engine is initially at rest. On right clicking the user is provided with a menu which provides five options-shaded, animate, increase speed, decrease speed, transparent.

Benefits

1.Simplicity: The project is built with the help of many standard library functions of the glut package. Hence the readability of the project is good.

2.Usability: Since the project is developed on a windows platform, is has good usability since system have implemented windows today.

3.Flexibility: It is easy to add new features to the project since all the projects are independent of their existence in the project. Hence the flexibility of the objects is beneficial

CHAPTER 4:

REQUIREMENTS AND SPECIFICATION

Hardware Constraints

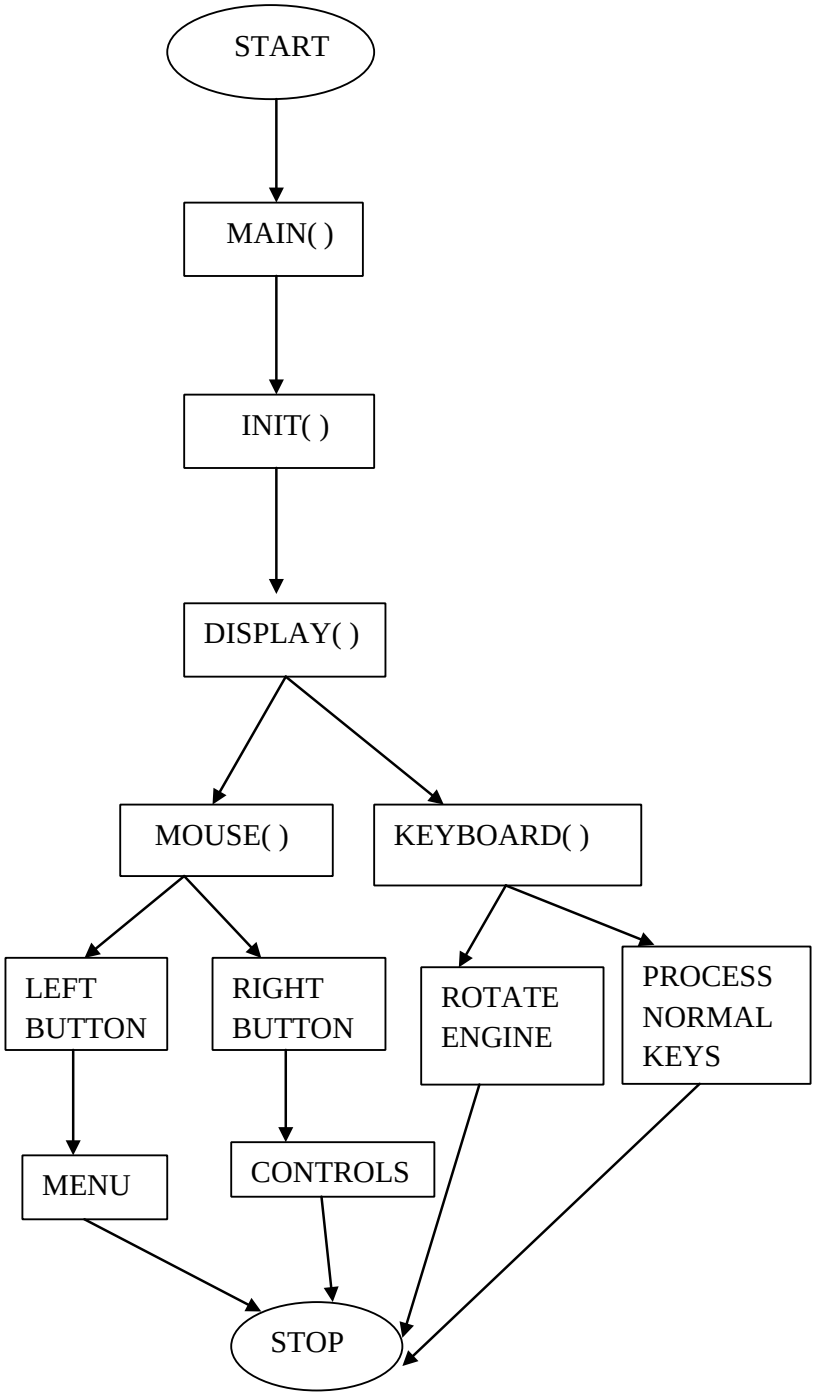
- Processor: Pentium PC
- RAM: 512MB
- Hard Disk: 20GB(approx)
- Display: VGA Color Monitor

Software Constraints

- Operating System: Windows 98SE/2000/XP/Vista/UBUNTU
- Language: Open GL
- Compiler/IDE: Code::Blocks IDE

CHAPTER 5:

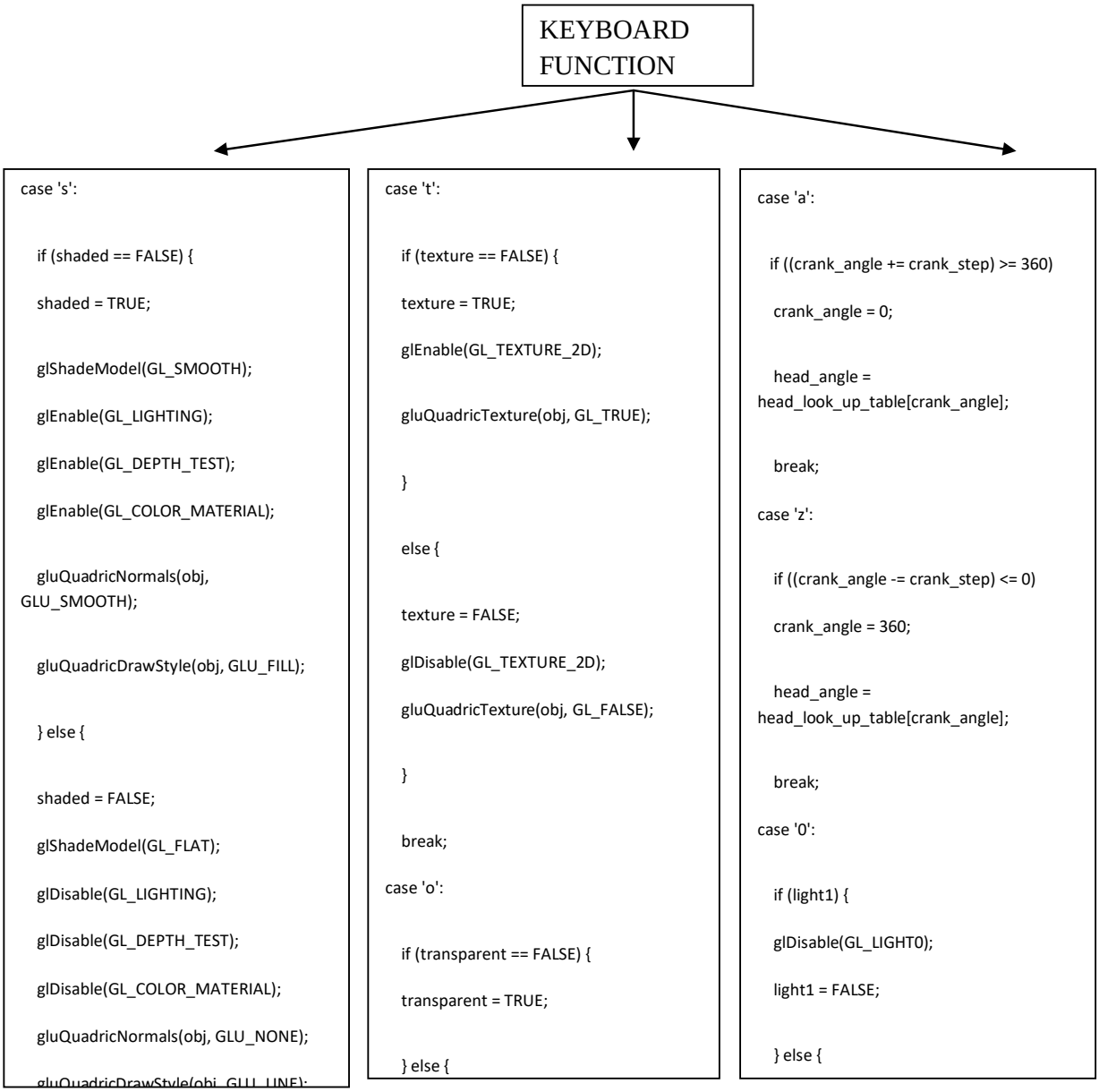
SOFTWARE DESIGN



CHAPTER 6:

IMPLEMENTATION

5.1:Display function:



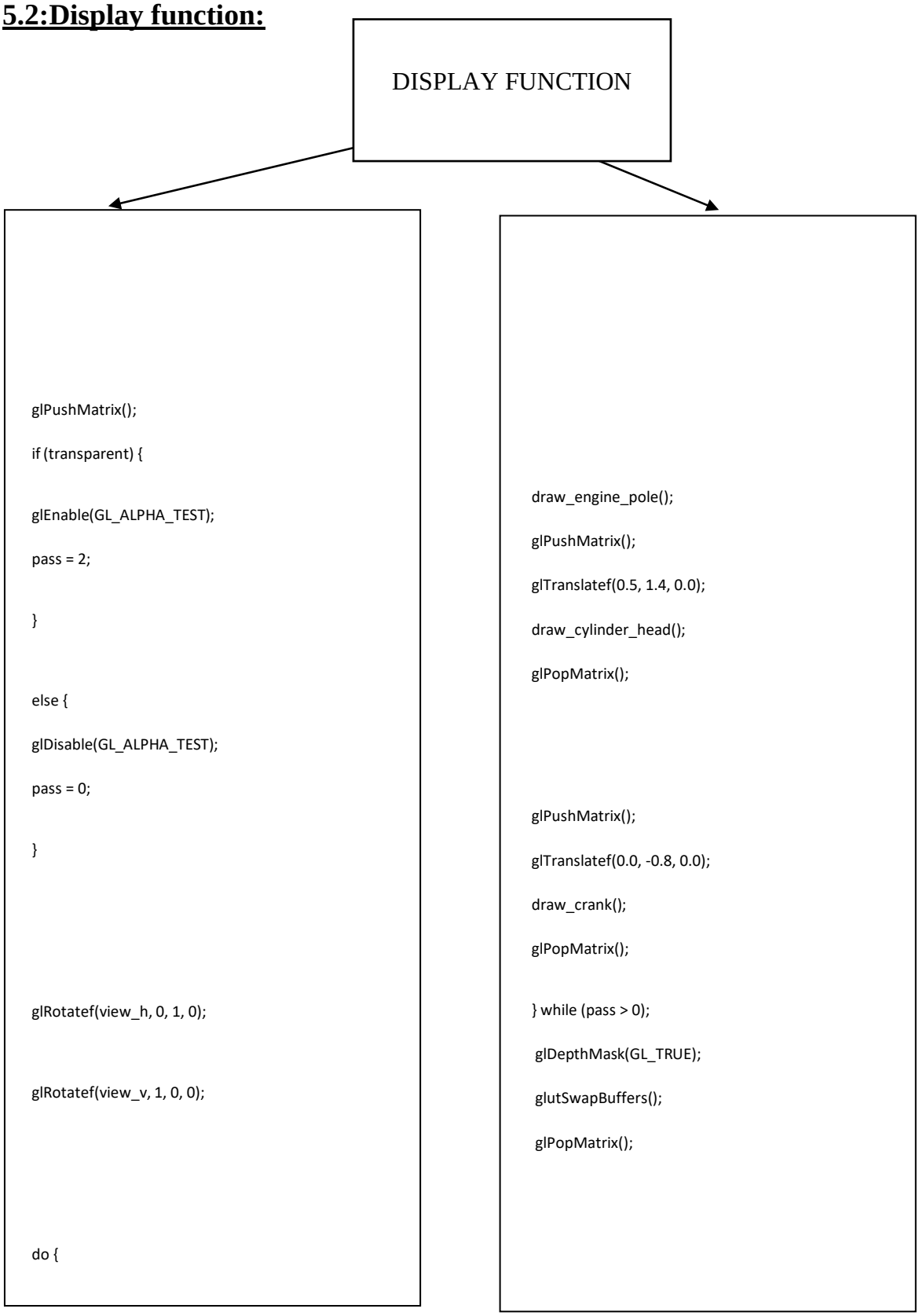
KEYBOARD FUNC

(Contd)

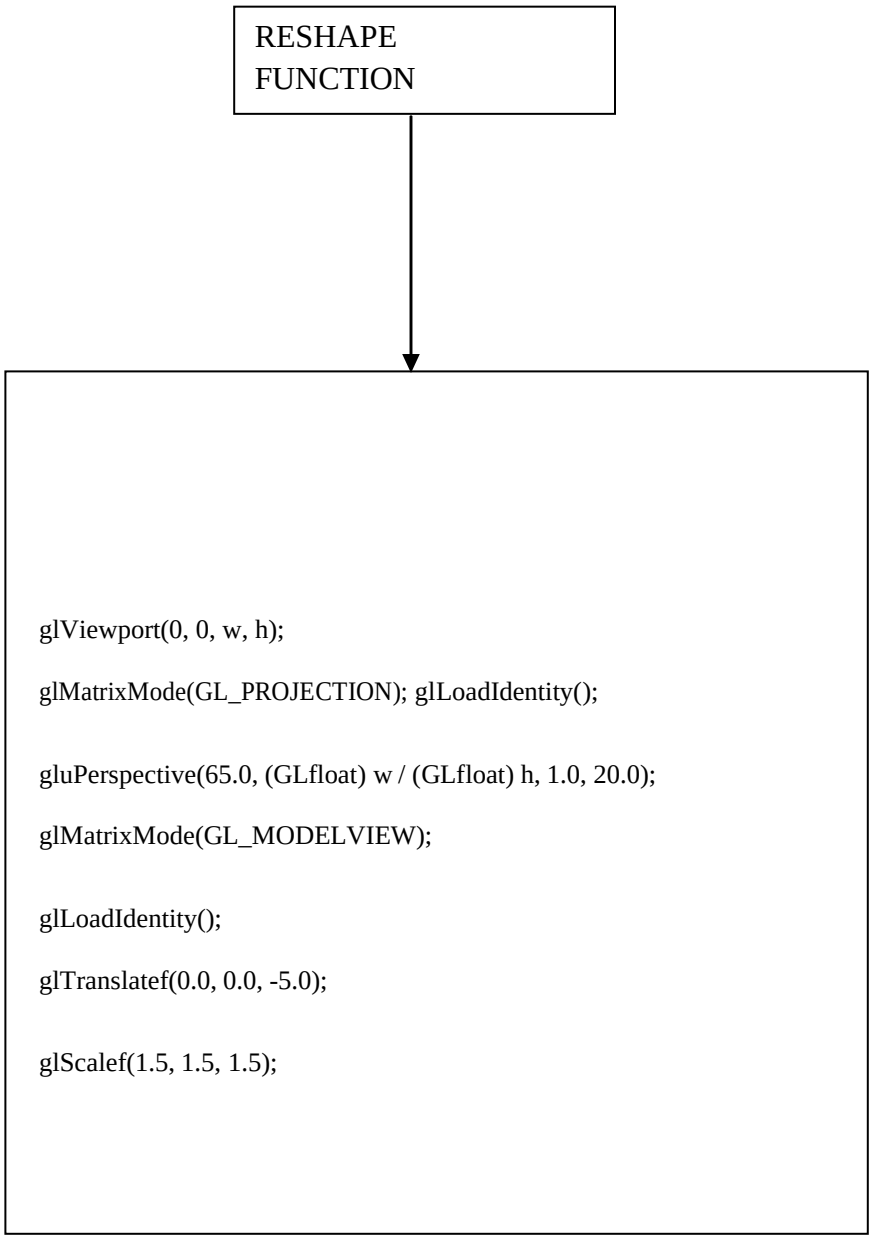
```
case '1':
   if (light2) {
   glDisable(GL_LIGHT1);
   light2 = FALSE;
   } else {
    glEnable(GL_LIGHT1);
    light2 = TRUE;
   }
   break;
case '4':
   if ((view_h -= ANGLE_STEP) <= 0)
    view_h = 360;
    break;
case '6':
    if ((view_h += ANGLE_STEP) >= 360)
     view_h = 0;
     break;
case '8':
     if ((view_v += ANGLE_STEP) >= 360)
     view_v = 0;
     break;
case '2':
     if ((view_v -= ANGLE_STEP) <= 0)
     view_v = 360;
     break;
```

```
case ' ':
   if (anim) {
   glutIdleFunc(0);
   anim = FALSE;
   }
   else {
   glutIdleFunc(animation);
   anim = TRUE;
   }
   break;
case '+':
   if ((++crank_step) > 45)
   crank_step = 45;
   break;
case '-':
   if ((--crank_step) <= 0)
   crank_step = 0;
   break;
default:
   return;
```

5.2:Display function:



5.3:RESHAPE function



CHAPTER 7:

Appendex Code:

```
#include<iostream>
#include<fstream>
#include<stdlib.h>
#include <stdio.h>
#include <GL/glut.h>
#include <math.h>

#define TRUE  1
#define FALSE 0

/* Dimensions of texture image. */
#define IMAGE_WIDTH  64
#define IMAGE_HEIGHT 64

/* Step to be taken for each rotation. */
#define ANGLE_STEP 10

/* Magic numbers for relationship b/w cylinder head and crankshaft. */
#define MAGNITUDE  120
#define PHASE      270.112
#define FREQ_DIV   58
#define ARC_LENGHT 2.7
#define ARC_RADIUS 0.15

/* Rotation angles */
```

```
GLdouble view_h = 270, view_v = 0, head_angle = 0;
GLint crank_angle = 0;

/* Crank rotation step. */
GLdouble crank_step = 5;

/* Toggles */
GLshort shaded = TRUE, anim = FALSE;
GLshort texture = FALSE, transparent = FALSE;
GLshort light1 = TRUE, light2 = FALSE;

/* Storage for the angle look up table and the texture map */
GLdouble head_look_up_table[361];
GLubyte image[IMAGE_WIDTH][IMAGE_HEIGHT][3];

/* Identifiers for each Display list */
GLint list_piston_shaded = 1;
GLint list_piston_texture = 2;
GLint list_flywheel_shaded = 4;
GLint list_flywheel_texture = 8;

/* Variable used in the creation of glut objects */
GLUquadricObj *obj;

/* Draws a box by scaling a glut cube of size 1. Also checks the shaded
   toggle to see which rendering style to use. NB Texture doesn't work
   correctly due to the cube being scaled. */
void
myBox(GLdouble x, GLdouble y, GLdouble z)
```

```
{
  glPushMatrix();
    glScalef(x, y, z);
    if (shaded)
      glutSolidCube(1);
    else
      glutWireCube(1);
  glPopMatrix();
}

/* Draws a cylinder using glu function, drawing flat disc's at each end,
   to give the appearence of it being solid. */
void
myCylinder(GLUquadricObj * object, GLdouble outerRadius,
  GLdouble innerRadius, GLdouble lenght)
{
  glPushMatrix();
    gluCylinder(object, outerRadius, outerRadius, lenght, 20, 1);
    glPushMatrix();
      glRotatef(180, 0.0, 1.0, 0.0);
      gluDisk(object, innerRadius, outerRadius, 20, 1);
    glPopMatrix();

    glTranslatef(0.0, 0.0, lenght);
    gluDisk(object, innerRadius, outerRadius, 20, 1);
  glPopMatrix();
}

/* Draws a piston.  */
```

```
void
draw_piston(void)
{
  glPushMatrix();
    glColor4f(0.9, 0.6, 0.9, 1.5);

    glPushMatrix();
      glRotatef(90, 0.0, 1.0, 0.0);
      glTranslatef(0.0, 0.0, -0.07);
      myCylinder(obj, 0.125, 0.06, 0.12);
    glPopMatrix();

    glRotatef(-90, 1.0, 0.0, 0.0);
    glTranslatef(0.0, 0.0, 0.05);
    myCylinder(obj, 0.06, 0.0, 0.6);
    glTranslatef(0.0, 0.0, 0.6);
    myCylinder(obj, 0.2, 0.0, 0.5);
  glPopMatrix();
}

/* Draws the engine pole and the pivot pole for the cylinder head. */
void
draw_engine_pole(void)
{
  glPushMatrix();
    glColor4f(0.9, 0.9, 0.9, 1.0);
    myBox(0.5, 3.0, 0.5);

    glColor3f(1.5, 2.1, 3.5);
```

```
   glRotatef(90, 0.0, 1.0, 0.0);
   glTranslatef(0.0, 0.9, -0.4);
   myCylinder(obj, 0.1, 0.0, 2);
  glPopMatrix();
}

/* Draws the cylinder head at the appropriate angle, doing the necessary
   translations for the rotation. */
void
draw_cylinder_head(void)
{
  glPushMatrix();
   glColor4f(3.5, 1.0, 0.5, 0.1);
   glRotatef(90, 1.0, 0.0, 0.0);
   glTranslatef(0, 0.0, 0.4);
   glRotatef(head_angle, 1, 0, 0);
   glTranslatef(0, 0.0, -0.4);
   myCylinder(obj, 0.23, 0.21, 1.6);
   glRotatef(180, 1.0, 0.0, 0.0);
   gluDisk(obj, 0, 0.23, 20, 1);
  glPopMatrix();
}

/* Draws the flywheel.  */
void
draw_flywheel(void)
{
  glPushMatrix();
   glColor4f(0.5, 0.5, 1.0, 1.0);
```

```
   glRotatef(90, 0.0, 1.0, 0.0);
   myCylinder(obj, 0.625, 0.08, 0.5);
  glPopMatrix();
}

/* Draws the crank bell, and the pivot pin for the piston. Also calls the
   appropriate display list of a piston doing the necessary rotations before
   hand.  */
void
draw_crankbell(void)
{
  glPushMatrix();
   glColor4f(1.0, 2.5, 3.5, 1.0);
   glRotatef(90, 0.0, 1.0, 0.0);
   myCylinder(obj, 0.3, 0.08, 0.12);

   glColor4f(0.5, 0.1, 0.5, 1.0);
   glTranslatef(0.0, 0.2, 0.0);
   myCylinder(obj, 0.06, 0.0, 0.34);

   glTranslatef(0.0, 0.0, 0.22);
   glRotatef(90, 0.0, 1.0, 0.0);
   glRotatef(crank_angle - head_angle, 1.0, 0.0, 0.0);
   if (shaded) {
     if (texture)
       glCallList(list_piston_texture);
     else
       glCallList(list_piston_shaded);
   } else
```

```
    draw_piston();
  glPopMatrix();
}

/* Draws the complete crank. Piston also gets drawn through the crank bell
   function. */
void
draw_crank(void)
{
  glPushMatrix();
    glRotatef(crank_angle, 1.0, 0.0, 0.0);

    glPushMatrix();
      glRotatef(90, 0.0, 1.0, 0.0);
      glTranslatef(0.0, 0.0, -1.0);
      myCylinder(obj, 0.08, 0.0, 1.4);
    glPopMatrix();

    glPushMatrix();
      glTranslatef(0.28, 0.0, 0.0);
      draw_crankbell();
    glPopMatrix();

    glPushMatrix();
      glTranslatef(-0.77, 0.0, 0.0);
      if (shaded) {
        if (texture)
          glCallList(list_flywheel_texture);
        else
```

```
        glCallList(list_flywheel_shaded);
     } else
       draw_flywheel();
   glPopMatrix();
 glPopMatrix();
}

/* Main display routine. Clears the drawing buffer and if transparency is
   set, displays the model twice, 1st time accepting those fragments with
   a ALPHA value of 1 only, then with DEPTH_BUFFER writing disabled for
   those with other values. */
void
display(void)
{
  int pass;

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glPushMatrix();
    if (transparent) {
      glEnable(GL_ALPHA_TEST);
      pass = 2;
    } else {
      glDisable(GL_ALPHA_TEST);
      pass = 0;
    }

    /* Rotate the whole model */
    glRotatef(view_h, 0, 1, 0);
```

```
  glRotatef(view_v, 1, 0, 0);

  do {
   if (pass == 2) {
    glAlphaFunc(GL_EQUAL, 1);
    glDepthMask(GL_TRUE);
    pass--;
   } else if (pass != 0) {
    glAlphaFunc(GL_NOTEQUAL, 1);
    glDepthMask(GL_FALSE);
    pass--;
   }
   draw_engine_pole();

   glPushMatrix();
    glTranslatef(0.5, 1.4, 0.0);
    draw_cylinder_head();
   glPopMatrix();

   glPushMatrix();
    glTranslatef(0.0, -0.8, 0.0);
    draw_crank();
   glPopMatrix();
  } while (pass > 0);
  glDepthMask(GL_TRUE);
  glutSwapBuffers();
 glPopMatrix();
}
```

```
/* Called when the window is idle. When called increments the crank angle
   by ANGLE_STEP, updates the head angle and notifies the system that
   the screen needs to be updated. */
void
animation(void)
{
  if ((crank_angle += crank_step) >= 360)
    crank_angle = 0;
  head_angle = head_look_up_table[crank_angle];
  glutPostRedisplay();
}

/* Called when a key is pressed. Checks if it recognizes the key and if so
   acts on it, updating the screen. */
/* ARGSUSED1 */
void
keyboard(unsigned char key, int x, int y)
{
  switch (key) {
  case 's':
    if (shaded == FALSE) {
      shaded = TRUE;
      glShadeModel(GL_SMOOTH);
      glEnable(GL_LIGHTING);
      glEnable(GL_DEPTH_TEST);
      glEnable(GL_COLOR_MATERIAL);
      gluQuadricNormals(obj, GLU_SMOOTH);
      gluQuadricDrawStyle(obj, GLU_FILL);
    } else {
```

```
    shaded = FALSE;
    glShadeModel(GL_FLAT);
    glDisable(GL_LIGHTING);
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_COLOR_MATERIAL);
    gluQuadricNormals(obj, GLU_NONE);
    gluQuadricDrawStyle(obj, GLU_LINE);
    gluQuadricTexture(obj, GL_FALSE);
  }
  if (texture && !shaded);
  else
    break;
 case 't':
  if (texture == FALSE) {
    texture = TRUE;
    glEnable(GL_TEXTURE_2D);
    gluQuadricTexture(obj, GL_TRUE);
  } else {
    texture = FALSE;
    glDisable(GL_TEXTURE_2D);
    gluQuadricTexture(obj, GL_FALSE);
  }
  break;
 case 'o':
  if (transparent == FALSE) {
    transparent = TRUE;
  } else {
    transparent = FALSE;
  }
```

```
  break;


case 'a':
  if ((crank_angle += crank_step) >= 360)
    crank_angle = 0;
  head_angle = head_look_up_table[crank_angle];
  break;
case 'z':
  if ((crank_angle -= crank_step) <= 0)
    crank_angle = 360;
  head_angle = head_look_up_table[crank_angle];
  break;
case '0':
  if (light1) {
    glDisable(GL_LIGHT0);
    light1 = FALSE;
  } else {
    glEnable(GL_LIGHT0);
    light1 = TRUE;
  }
  break;
case '1':
  if (light2) {
    glDisable(GL_LIGHT1);
    light2 = FALSE;
  } else {
    glEnable(GL_LIGHT1);
    light2 = TRUE;
  }
```

```
  break;
 case '4':
  if ((view_h -= ANGLE_STEP) <= 0)
    view_h = 360;
  break;

 case '8':
  if ((view_v += ANGLE_STEP) >= 360)
    view_v = 0;
  break;
 case '2':
  if ((view_v -= ANGLE_STEP) <= 0)
    view_v = 360;
  break;
 case ' ':
  if (anim) {
    glutIdleFunc(0);
    anim = FALSE;
  } else {
    glutIdleFunc(animation);
    anim = TRUE;
  }
  break;
 case '+':
  if ((++crank_step) > 45)
    crank_step = 45;
  break;
 case '-':
  if ((--crank_step) <= 0)
```

```
    crank_step = 0;
   break;
  default:
   return;
  }
  glutPostRedisplay();
}

/* ARGSUSED1 */
void
special(int key, int x, int y)
{
  switch (key) {
  case GLUT_KEY_LEFT:
   if ((view_h -= ANGLE_STEP) <= 0)
    view_h = 360;
   break;
  case GLUT_KEY_RIGHT:
   if ((view_h += ANGLE_STEP) >= 360)
    view_h = 0;
   break;
  case GLUT_KEY_UP:
   if ((view_v += ANGLE_STEP) >= 360)
    view_v = 0;
   break;
  case GLUT_KEY_DOWN:
   if ((view_v -= ANGLE_STEP) <= 0)
    view_v = 360;
   break;
```

```
  default:
    return;
  }
  glutPostRedisplay();
}

/* Called when a menu option has been selected. Translates the menu item
   identifier into a keystroke, then call's the keyboard function. */
void
menu(int val)
{
  unsigned char key;

  switch (val) {
  case 1:
    key = 's';
    break;
  case 2:
    key = ' ';
    break;
  case 3:
    key = 't';
    break;
  case 4:
    key = 'o';
    break;
  case 5:
    key = '0';
    break;
```

```
  case 7:
    key = '+';
    break;
  case 8:
    key = '-';
    break;
  default:
    return;
  }
  keyboard(key, 0, 0);
}

/* Initializes the menu of toggles. */
void
create_menu(void)
{
  glutCreateMenu(menu);
  glutAttachMenu(GLUT_LEFT_BUTTON);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
  glutAddMenuEntry("Shaded", 1);
  glutAddMenuEntry("Animation", 2);
  glutAddMenuEntry("Texture", 3);
  glutAddMenuEntry("Transparency", 4);
  glutAddMenuEntry("Light On/Off", 5);

  glutAddMenuEntry("Speed UP", 7);
  glutAddMenuEntry("Slow Down", 8);
}
```

```
/* Makes a simple check pattern image. (Copied from the red book example
   "checker.c".) */
void
make_image(void)
{
  int i, j, c;

  for (i = 0; i < IMAGE_WIDTH; i++) {
    for (j = 0; j < IMAGE_HEIGHT; j++) {
      c = ((((i & 0x8) == 0) ^ ((j & 0x8)) == 0)) * 255;
      image[i][j][0] = (GLubyte) c;
      image[i][j][1] = (GLubyte) c;
      image[i][j][2] = (GLubyte) c;
    }
  }
}

/* Makes the head look up table for all possible crank angles. */
void
make_table(void)
{
  GLint i;
  GLdouble k;

  for (i = 0, k = 0.0; i < 360; i++, k++) {
    head_look_up_table[i] =
      MAGNITUDE * atan(
      (ARC_RADIUS * sin(PHASE - k / FREQ_DIV)) /
```

```
    ((ARC_LENGHT - ARC_RADIUS * cos(PHASE - k / FREQ_DIV))));
  }
}

/* Initializes texturing, lighting, display lists, and everything else
   associated with the model. */
void
myinit(void)
{
  GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
  GLfloat mat_shininess[] = {50.0};
  GLfloat light_position1[] = {1.0, 1.0, 1.0, 0.0};
  GLfloat light_position2[] = {-1.0, 1.0, 1.0, 0.0};

  glClearColor(0.0, 0.0, 0.0, 0.0);

  obj = gluNewQuadric();
  make_table();
  make_image();

  /* Set up Texturing */
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
  glTexImage2D(GL_TEXTURE_2D, 0, 3, IMAGE_WIDTH,
    IMAGE_HEIGHT, 0, GL_RGB, GL_UNSIGNED_BYTE,
    image);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)
/* Set up Lighting */
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glLightfv(GL_LIGHT0, GL_POSITION, light_position1);
glLightfv(GL_LIGHT1, GL_POSITION, light_position2);

/* Initial render mode is with full shading and LIGHT 0
  enabled. */
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glDepthFunc(GL_LEQUAL);
glEnable(GL_DEPTH_TEST);
glDisable(GL_ALPHA_TEST);

glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
glShadeModel(GL_SMOOTH);

/* Initialize display lists */
glNewList(list_piston_shaded, GL_COMPILE);
 draw_piston();
glEndList();
glNewList(list_flywheel_shaded, GL_COMPILE);
 draw_flywheel();
glEndList();

gluQuadricTexture(obj, GL_TRUE);
glNewList(list_piston_texture, GL_COMPILE);
```

```
    draw_piston();
  glEndList();
  glNewList(list_flywheel_texture, GL_COMPILE);
    draw_flywheel();
  glEndList();
  gluQuadricTexture(obj, GL_FALSE);
}

/* Called when the model's window has been reshaped.  */
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, 0.0, -5.0);  /* viewing transform  */
  glScalef(1.5, 1.5, 1.5);
}

void DrawText(const char *text, int length, int x, int y)
{
    glMatrixMode(GL_PROJECTION);
    double *matrix = new double[16];
    glGetDoublev(GL_PROJECTION_MATRIX,matrix);
    glLoadIdentity();
    glOrtho(0,800,0,600,-5,5);
```

```
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glPushMatrix();
    glLoadIdentity();
    glRasterPos2i(x,y);
    for(int i = 0;i<length;i++)
    {
       glutBitmapCharacter(GLUT_BITMAP_9_BY_15,(int)text[i]);
    }
    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glLoadMatrixd(matrix);
    glMatrixMode(GL_MODELVIEW);

}
void display_other()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    gluLookAt(0,0,-10,0,0,3,0,1,0);

    std::string text1;
    std::string text2;
    std::string text3;
    std::string text4;
    std::string text5;
    std::string text6;
    std::string text7;
    std::string text8;
    std::string text9;
```

```
    text1 = "Miniature Reciprocating Steam Engine by Umesh Abbanna AND Sachin chavan;
    text2 = "Keypad Arrow keys rotates object.";
     text3 ="Rotate crank: 'a' = anti-clock wise 'z' = clock wise";
     text4 ="Crank Speed : '+' = Speed up by 1   '-' = Slow Down by 1";
     text5 = "Toggle      : 's' = Shading        't' = Texture";
     text6 = "            : 'y' = Animation      'o' = Transparency";
     text7 = "            : '0' = Light On/Off";
     text8 = " Alternatively a pop up menu with all toggles is attached";
     text9 = " to the left mouse button.";
    DrawText(text1.data(),text1.size(),200,500);
    DrawText(text2.data(),text2.size(),200,460);
    DrawText(text3.data(),text3.size(),200,420);
    DrawText(text4.data(),text4.size(),200,380);
    DrawText(text5.data(),text5.size(),200,340);
    DrawText(text6.data(),text6.size(),200,310);
    DrawText(text7.data(),text7.size(),200,270);
    DrawText(text8.data(),text8.size(),200,230);
    DrawText(text9.data(),text9.size(),200,190);

}

/* Main program. An interactive model of a miniature steam engine.
   Sets system in Double Buffered mode and initializes all the call-back
   functions. */
int
main(int argc, char **argv)
{
    /*
  puts("Miniature Reciprocating Steam Engine\n");
```

```
puts("Keypad Arrow keys (with NUM_LOCK on) rotates object.");
puts("Rotate crank: 'a' = anti-clock wise 'z' = clock wise");
puts("Crank Speed : '+' = Speed up by 1   '-' = Slow Down by 1");
puts("Toggle      : 's' = Shading        't' = Texture");
puts("            : 'y' = Animation      'o' = Transparency");
puts("            : '0' = Right Light    '1' = Left Light");
puts(" Alternatively a pop up menu with all toggles is attached");
puts("   to the left mouse button.\n");
*/

glutInit(&argc, argv);
glutInitWindowSize(800, 800);
 glutCreateWindow("Instructions");
glutDisplayFunc(display_other);
glutInitWindowSize(800, 800);

/* Transparency won't work properly without GLUT_ALPHA */
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH | GLUT_MULTISAMPLE);
glutCreateWindow("Miniature Steam Engine");
glutDisplayFunc(display);
glutKeyboardFunc(keyboard);
glutSpecialFunc(special);
create_menu();

myinit();

glutReshapeFunc(myReshape);
glutMainLoop();
return 0;           /* ANSI C requires main to return int. */
```

CHAPTER 9:

CONCLUSION and FUTURE SCOPE.

This project allows the user to rotate the piston in a Steam Engine. Its like a Miniature Steam Engine Simulation.

Future scope:

1.SIMULATOR

Not only the movement of piston, we can make the whole parts in the steam engine working so that it will be a simulator of steam engine. By modifying this project we can construct a fully fledged simulator. Students who are studying about the steam engine can work through this and it will be very helpful for them. Almost a complete picturization of a steam engine can be done through this.

2.DESIGN OF STEAM ENGINES

Engineers who build Steam Engines can design their model by looking this project. They get a good picturization by seeing this and it will be helpful for them in building steam engines. So this project will be benefited by Engineers

CHAPTER 10:

BIBLIOGRAPHY.

1.Interactive Computer Graphics A Top-Down Approach with OpenGL - Edward Angel, 5th Edition, Addison-Wesley, 2008

2. The Official Guide to Learning OpenGL, by Jackie Neider, Tom Davis, Mason Woo (THE RED BOOK)

3. OpenGL Super Bible by Richard S. Wright, Jr. and Michael Sweet

4. http://www.opengl.org/ online man pages.

5. And lastly GOOGLE.