Steam Engine

CS352: Computer Graphics & Visualization Lab

Project Code

Course Instructor:

Dr. Somnath Dey

Submitted By:

Koneti Anuhya – 200001037

Kothuru Sharvani – 200001038

Nelavalli Sri Nikhitha –200001052

```
#include <GL/glut.h>
#include <GL/glu.h>
#include <bits/stdc++.h>
#include <math.h>
#include <time.h>
#include <SOIL/SOIL.h>

using namespace std;

#define TRUE  1
#define FALSE 0

/* Dimensions of texture image. */
#define IMAGE_WIDTH  128
#define IMAGE_HEIGHT 128

/* Step to be taken for each rotation. */
#define ANGLE_STEP 20

/* 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
/*Reference for code:-https://github.com/itmesneha/Steam-Engine-using-
OpenGL/blob/master/main.cpp*/
/* Rotation angles */
GLdouble view_h = 270, view_v = 0, view_w = 0, head_angle = 0;
GLint crank_angle = 0;

/* Crank rotation step. */
GLdouble crank_step = 5;
GLshort light1 = TRUE, light2 = TRUE;

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

GLfloat light_ambient[4];

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

/* Scaling Factor */
GLdouble sx = 1;
GLdouble sy = 1;
GLdouble sz = 1;

int flag = 0;

/* Variable used in the creaton of glu objects */
GLUquadricObj *obj;
```

```
/* Function to load image as an openGL texture*/
GLint LoadGLTexture(const char *filename)
{
    GLuint textureID = SOIL_load_OGL_texture(
        filename,
        SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID,
        SOIL_FLAG_INVERT_Y
    );

    return textureID;
}

/* 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);
      //glScalef(sx, sy, sz);
    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);
    //glScalef(sx, sy, sz);
    glPushMatrix();
      glRotatef(180, 0.0, 1.0, 0.0);
        //glScalef(sx, sy, sz);
      gluDisk(object, innerRadius, outerRadius, 20, 1);
    glPopMatrix();
    glTranslatef(0.0, 0.0, lenght);
    gluDisk(object, innerRadius, outerRadius, 20, 1);
  glPopMatrix();

}

/* Function to draw a piston.  */
void draw_piston(void)

{
  glPushMatrix();
    glColor4f(1.3, 0.6, 0.9, 1.0);
    glPushMatrix();
      glRotatef(90, 0.0, 1.0, 0.0);
      glTranslatef(0.0, 1.0, -0.07);
    //glScalef(sx, sy, sz);
      myCylinder(obj, 0.125, 0.06, 0.12);
    glPopMatrix();
    glRotatef(-90, 1.0, 0.0, 0.0);
```

```
    glTranslatef(0.0, 0.0, 0.05);
    //glScalef(sx, sy, sz);
    myCylinder(obj, 0.06, 0.0, 0.6);
    glTranslatef(0.0, 0.0, 0.6);
    //glScalef(sx, sy, sz);
    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);
    //glScalef(sx, sy, sz);
    myBox(0.5, 3.0, 0.5);
    glColor3f(1.5, 1.1, 0.5);
    glRotatef(90, 0.0, 1.0, 0.0);
    glTranslatef(0.0, 0.9, -0.4);
    //glScalef(sx, sy, sz);
    myCylinder(obj, 0.1, 0.0, 2);
  glPopMatrix();
}

/* Draws the cylinder head at the appropreate angle, doing the necesary
   translations for the rotation. */
void draw_cylinder_head(void)
{
  glPushMatrix();
    glColor4f(0.5, 1.0, 1.5, 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);
    //glScalef(sx, sy, sz);
    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(1.6, 0.3, 0.0, 1.0);
    glRotatef(90, 0.0, 1.0, 0.0);
    //glScalef(sx, sy, sz);
    myCylinder(obj, 0.625, 0.08, 0.5);
  glPopMatrix();
}

/* Draws the crank bell, and the pivot pin for the piston. Also calls the
   appropreate display list of a piston doing the nesacary rotations before
   hand.  */
void draw_crankbell(void)
{
  glPushMatrix();
    glColor4f(1.0, 1.5, 0.5, 1.0);
    glRotatef(90, 0.0, 1.0, 0.0);
    //glScalef(sx, sy, sz);
```

```
    myCylinder(obj, 0.3, 0.08, 0.12);
    glColor4f(0.5, 0.1, 1.5, 1.0);
    //glScalef(sx, sy, sz);
    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);
    //drawing the textured and shaded piston
    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);
    //glScalef(sx, sy, sz);
        glTranslatef(0.0, 0.0, -1.0);
        myCylinder(obj, 0.08, 0.0, 1.4);
      glPopMatrix();

      glPushMatrix();
    //glScalef(sx, sy, sz);
        glTranslatef(0.28, 0.0, 0.0);
        draw_crankbell();
      glPopMatrix();

      glPushMatrix();
    //glScalef(sx, sy, sz);
        glTranslatef(-0.77, 0.0, 0.0);
    //drawing the textured and shaded flywheel
    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;
    glScalef(sx, sy, sz);
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // glColor3f(1, 1, 1);
    glPushMatrix();
    /* The alpha test discards fragments depending on the outcome of a
comparison between an incoming fragment's alpha value and a constant
reference value */
    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);
    glRotatef(view_w, 0, 0, 1);
    /* glAlphaFunc specifies the reference value and the comparison
function. The comparison is performed only if alpha testing is enabled */
    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();
        //glScalef(sx, sy, sz);
        glTranslatef(0.5, 1.4, 0.0);
        draw_cylinder_head();
        glPopMatrix();
        glPushMatrix();
        //glScalef(sx, sy, sz);
        glTranslatef(0.0, -0.8, 0.0);
        draw_crank();
        glPopMatrix();
    }while (pass > 0);
    /* Image rendering and labelling all the parts*/
    glColor4f(0.9,0.9,0.9,1.0);
    glEnable(GL_TEXTURE_2D);
    GLuint texture = LoadGLTexture("IITIlogo.png");
    glBindTexture( GL_TEXTURE_2D, texture );
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f);glVertex3f(-0.2, -0.2, 0.25);
        glTexCoord2f(0.0f, 1.0f);glVertex3f(-0.2, 0.3, 0.25);
        glTexCoord2f(1.0f, 1.0f);glVertex3f(0.2, 0.3, 0.25);
        glTexCoord2f(1.0f, 0.0f);glVertex3f(0.2, -0.2, 0.25);
    glEnd();

    glColor3f(1.5,1.1,0.5);
    GLuint texture1 = LoadGLTexture("enginepole.png");
    glBindTexture( GL_TEXTURE_2D, texture1 );
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 0.0f);glVertex3f(0.8, 0.85, 0.1);
    glTexCoord2f(0.0f, 1.0f);glVertex3f(0.8, 0.95, 0.1);
    glTexCoord2f(1.0f, 1.0f);glVertex3f(1.2,0.95, 0.1);
    glTexCoord2f(1.0f, 0.0f);glVertex3f(1.2, 0.85, 0.1);
glEnd();

glColor4f(1.6,0.3,0.0,1.0);
GLuint texture2 = LoadGLTexture("flywheel.png");
glBindTexture( GL_TEXTURE_2D, texture2 );
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 0.0f);glVertex3f(-0.7, -0.8, 0.65);
    glTexCoord2f(0.0f, 1.0f);glVertex3f(-0.7, -0.6, 0.65);
    glTexCoord2f(1.0f, 1.0f);glVertex3f(-0.3, -0.6, 0.65);
    glTexCoord2f(1.0f, 0.0f);glVertex3f(-0.3, -0.8, 0.65);
glEnd();

glColor4f(0.5,1.0,1.5,0.1);
GLuint texture3 = LoadGLTexture("cylinderhead.png");
glBindTexture( GL_TEXTURE_2D, texture3 );
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 0.0f);glVertex3f(0.4, 0.0, 0.23);
    glTexCoord2f(0.0f, 1.0f);glVertex3f(0.4, 0.8, 0.23);
    glTexCoord2f(1.0f, 1.0f);glVertex3f(0.6, 0.8, 0.23);
    glTexCoord2f(1.0f, 0.0f);glVertex3f(0.6, 0.0, 0.23);
glEnd();

glColor4f(1.3,0.6,0.9,1.0);
GLuint texture4 = LoadGLTexture("piston.png");
glBindTexture( GL_TEXTURE_2D, texture4 );
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 0.0f);glVertex3f(0.45, -0.45, 0.07);
    glTexCoord2f(0.0f, 1.0f);glVertex3f(0.45, -0.25, 0.07);
    glTexCoord2f(1.0f, 1.0f);glVertex3f(0.55, -0.25, 0.07);
    glTexCoord2f(1.0f, 0.0f);glVertex3f(0.55, -0.45, 0.07);
glEnd();

glColor4f(0.9,0.9,0.9,1.0);
GLuint texture5 = LoadGLTexture("support.png");
glBindTexture( GL_TEXTURE_2D, texture5 );
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 0.0f);glVertex3f(-0.2, -0.6, 0.25);
```

```
        glTexCoord2f(0.0f, 1.0f);glVertex3f(-0.2, -0.4, 0.25);
        glTexCoord2f(1.0f, 1.0f);glVertex3f(0.2, -0.4, 0.25);
        glTexCoord2f(1.0f, 0.0f);glVertex3f(0.2, -0.6, 0.25);
    glEnd();
    glDisable(GL_TEXTURE_2D);
    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)
{
    //creates different coloured lights
    for(int i=0; i<3; i++){
        light_ambient[i] = ((float)rand()) / RAND_MAX;
    }
    light_ambient[3] = 1.0;
    glLightfv(GL_LIGHT1, GL_AMBIENT, light_ambient);
  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 reconises the key and if so
   acts on it, updateing the screen. */
void keyboard(unsigned char key, int x, int y)
{
  switch (key) {
      //scale up by a factor of (2,2,2)
      case '9':
          sx = 2;
          sy = 2;
          sz = 2;
          break;
      //scale down by a factor of (0.5,0.5,0.5)
      case '8':
          sx = 0.5;
          sy = 0.5;
          sz = 0.5;
          flag = 1;
          break;
      //displaying the shaded model and wired model
      case 's':
          sx = 1;
          sy = 1;
          sz = 1;
        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;
  //displaying transparent model
  case 'o':
        sx = 1;
        sy = 1;
        sz = 1;
  if (transparent == FALSE) {
    transparent = TRUE;
  } else {
    transparent = FALSE;
  }
  break;
//Rotate crank in Anticlockwise direction
case 'a':
        sx = 1;
        sy = 1;
        sz = 1;
  if ((crank_angle += crank_step) >= 360)
    crank_angle = 0;
  head_angle = head_look_up_table[crank_angle];
  break;
//Rotate crank in Clockwise direction
case 'z':
        sx = 1;
        sy = 1;
        sz = 1;
  if ((crank_angle -= crank_step) <= 0)
    crank_angle = 360;
  head_angle = head_look_up_table[crank_angle];
  break;
//Rotate along y-axis
case '1':
        sx = 1;
        sy = 1;
        sz = 1;
  if ((view_h -= ANGLE_STEP) <= 0)
    view_h = 360;
  break;
case '6':
        sx = 1;
        sy = 1;
        sz = 1;
  if ((view_h += ANGLE_STEP) >= 360)
    view_h = 0;
  break;
//Rotate along x-axis
case '2':
        sx = 1;
        sy = 1;
        sz = 1;
  if ((view_v += ANGLE_STEP) >= 360)
    view_v = 0;
```

```
    break;
  case '5':
          sx = 1;
          sy = 1;
          sz = 1;
    if ((view_v -= ANGLE_STEP) <= 0)
      view_v = 360;
    break;
  //Rotate along z-axis
  case '3':
          sx = 1;
          sy = 1;
          sz = 1;
    if ((view_w += ANGLE_STEP) >= 360)
      view_w = 0;
    break;
  case '4':
          sx = 1;
          sy = 1;
          sz = 1;
    if ((view_w -= ANGLE_STEP) <= 0)
      view_w = 360;
    break;
 //Enabling and disabling Animation
  case ' ':
          sx = 1;
          sy = 1;
          sz = 1;
    if (anim) {
      glutIdleFunc(0);
      anim = FALSE;
    } else {
      glutIdleFunc(animation);
      anim = TRUE;
    }
    break;
  //Switch on and off for light1
  case 'n':
          sx = 1;
          sy = 1;
          sz = 1;
    if (light1) {
      glDisable(GL_LIGHT1);
      light1 = FALSE;
    } else {
      glEnable(GL_LIGHT1);
      light1 = TRUE;
    }
    break;
  //Switch on and off for light2
  case 'x':
          sx = 1;
          sy = 1;
          sz = 1;
    if (light2) {
      glDisable(GL_LIGHT2);
      light2 = FALSE;
    } else {
      glEnable(GL_LIGHT2);
      light2 = TRUE;
    }
```

```
    break;
  //increase speed
  case '+':
    if ((++crank_step) > 45)
      crank_step = 45;
    break;
  //decrease spped
  case '-':
    if ((--crank_step) <= 0)
      crank_step = 0;
    break;
  default:
    return;
  }
  glutPostRedisplay();
}

/* Function to add key values to the menu options */
void menu(int val)
{
  unsigned char key;
  switch (val) {
      case 1:
    key = 's';
    break;
  case 2:
    key = ' ';
    break;
  case 3:
    key = 'o';
    break;
  case 4:
    key = 'n';
    break;
  case 5:
    key = 'x';
    break;
  case 6:
    key = '+';
    break;
  case 7:
    key = '-';
    break;
  case 8:
    key = '9';
    break;
  case 9:
    key = '8';
    break;
   case 10:
     key = 'a';
     break;
   case 11:
     key = 'z';
      break;
  default:
    return;
  }
  //calling keyboard function for specific action
  keyboard(key, 0, 0);
}
```

```
/* Function to display menu on the screen */
void create_menu(void)
{
  glutCreateMenu(menu);
  glutAttachMenu(GLUT_LEFT_BUTTON);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
  glutAddMenuEntry("Shaded", 1);
  glutAddMenuEntry("Animation", 2);
  glutAddMenuEntry("Transparency", 3);
  glutAddMenuEntry("Light1 on/off", 4);
  glutAddMenuEntry("Light2 on/off", 5);
  glutAddMenuEntry("Speed UP", 6);
  glutAddMenuEntry("Slow Down", 7);
  glutAddMenuEntry("Scale UP", 8);
  glutAddMenuEntry("Scale Down", 9);
  glutAddMenuEntry("Anticlockwise", 10);
  glutAddMenuEntry("Clockwise", 11);
}

/* Makes a simple check pattern image. (Copied from the redbook 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))));
  }
}

/* Initialises texturing, lighting, display lists, and everything else
   associated with the model. */
void myinit(void)
{
    //specifying light positions
    GLfloat light_position2[] = {1.0, 1.0, 1.0, 0.0};
    GLfloat light_position3[] = {-1.0, 1.0, 1.0, 0.0};
    GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};

    glLightfv(GL_LIGHT1, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT1, GL_SPECULAR, light_specular);
```

```
    glLightfv(GL_LIGHT2, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT2, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT2, GL_SPECULAR, light_specular);

    glClearColor(0.0, 0.0, 0.0, 0.0);
    obj = gluNewQuadric();
    make_table();
    make_image();

    /* Setting up Lighting */
    glLightfv(GL_LIGHT1, GL_POSITION, light_position2);
    glLightfv(GL_LIGHT2, GL_POSITION, light_position3);

    /* Initialise display lists */
    glNewList(list_piston_shaded, GL_COMPILE);
    //glScalef(sx, sy, sz);
      draw_piston();
    glEndList();

    glNewList(list_flywheel_shaded, GL_COMPILE);
    //glScalef(sx, sy, sz);
      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);

    /* Initial render mode is with full shading and LIGHT 1, 2
       enabled. */
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT1);
    glEnable(GL_LIGHT2);
    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);
}
/* 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);
}

/* Main program. An interactive model of a miniture steam engine.
```

```
   Sets system in Double Buffered mode and initialises all the call-back
   functions. */
int main(int argc, char **argv)
{
  glutInitWindowSize(750, 750);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH |
GLUT_MULTISAMPLE);
  glutCreateWindow("Steam Engine");
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  create_menu();
  myinit();
  glutReshapeFunc(myReshape);
  glutMainLoop();
  return 0;
}
```