#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];

/\* 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;

/\* Variable used in the creaton of glu 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(0.3, 0.6, 0.9, 1.0);

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(0.5, 0.1, 0.5); 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 appropreate angle, doing the necesary

translations for the rotation. \*/ void

draw\_cylinder\_head(void)

{

glPushMatrix();

glColor4f(0.5, 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

appropreate display list of a piston doing the nesacary rotations before hand. \*/ void

draw\_crankbell(void)

{

glPushMatrix();

glColor4f(1.0, 0.5, 0.5, 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 reconises the key and if so

acts on it, updateing 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 '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; }

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 6: key = '1'; break; case 7: key = '+'; break; case 8: key = '-'; break; default: return; }

keyboard(key, 0, 0);

}

/\* Initialises the menu of toggles. \*/ void

create\_menu(void)

{

glutCreateMenu(menu); glutAttachMenu(GLUT\_RIGHT\_BUTTON); glutAddMenuEntry("Shaded", 1); glutAddMenuEntry("Animation", 2); glutAddMenuEntry("Texture", 3);

glutAddMenuEntry("Transparency", 4);

glutAddMenuEntry("Right Light (0)", 5);

glutAddMenuEntry("Left Light (1)", 6); glutAddMenuEntry("Speed UP", 7); glutAddMenuEntry("Slow Down", 8);

}

/\* 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)

{

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);

/\* Initialise 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);

}

/\* Main program. An interactive model of a miniture steam engine. Sets system in Double Buffered mode and initialises all the callback functions. \*/ int

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

{

puts("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(" : ' ' = 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");

glutInitWindowSize(400, 400); glutInit(&argc, argv);

/\* Transperancy won't work properly without GLUT\_ALPHA \*/ glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH | GLUT\_MULTISAMPLE);

glutCreateWindow("Steam Engine");

glutDisplayFunc(display); glutKeyboardFunc(keyboard); glutSpecialFunc(special); create\_menu();

myinit();

glutReshapeFunc(myReshape); glutMainLoop();

return 0; /\* ANSI C requires main to return int. \*/

}