STEAM ENGINE SIMULATION

Steam Engine code:

```
#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
call
back
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. */
}
```