#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include <math.h>

#include <GL/glut.h>

/\* Some <math.h> files do not define M\_PI... \*/

#ifndef M\_PI

#define M\_PI 3.141592654

#endif

#ifdef WIN32

#define drand48() (((float) rand())/((float) RAND\_MAX))

#define srand48(x) (srand((x)))

#else

extern double drand48(void);

extern void srand48(long seedval);

#endif

#define XSIZE 100

#define YSIZE 75

#define RINGS 5

#define BLUERING 0

#define BLACKRING 1

#define REDRING 2

#define YELLOWRING 3

#define GREENRING 4

#define BACKGROUND 8

enum

{

BLACK = 0,

RED,

GREEN,

YELLOW,

BLUE,

MAGENTA,

CYAN,

WHITE

};

typedef short Point[2];

GLenum rgb, doubleBuffer, directRender;

int SPEEDLIMIT=4000;

unsigned char rgb\_colors[RINGS][3];

int mapped\_colors[RINGS];

float dests[RINGS][3];

float offsets[RINGS][3];

float angs[RINGS];

float rotAxis[RINGS][3];

int iters[RINGS];

GLuint theTorus;

void FillTorus(float rc, int numc, float rt, int numt)

{

int i, j, k;

double s, t;

double x, y, z;

double pi, twopi;

pi = M\_PI;twopi = 2 \* pi;

for (i = 0; i < numc; i++)

{

glBegin(GL\_QUAD\_STRIP);

for (j = 0; j <= numt; j++)

{

for (k = 1; k >= 0; k--)

{

s = (i + k) % numc + 0.5;

t = j % numt;

x = cos(t \* twopi / numt) \* cos(s \* twopi / numc);

y = sin(t \* twopi / numt) \* cos(s \* twopi / numc);

z = sin(s \* twopi / numc);

glNormal3f(x, y, z);

x = (rt + rc \* cos(s \* twopi / numc)) \* cos(t \* twopi / numt);

y = (rt + rc \* cos(s \* twopi / numc)) \* sin(t \* twopi / numt);

z = rc \* sin(s \* twopi / numc);glVertex3f(x, y, z);

}

}

glEnd();

}

}

float Clamp(int iters\_left, float t)

{

if (iters\_left < 3)

return 0.0;

return (iters\_left - 2) \* t / iters\_left;

}

void Idle(void)

{

int i, j;

int more = GL\_FALSE;

for (i = 0; i < RINGS; i++)

{

if (iters[i])

{

for (j = 0; j < 3; j++)

offsets[i][j] = Clamp(iters[i], offsets[i][j]);

angs[i] = Clamp(iters[i], angs[i]);

iters[i]--;

more = GL\_TRUE;

}

}

if (more)

glutPostRedisplay();

else

glutIdleFunc(NULL);

}

void DrawScene(void)

{

int i;glPushMatrix();

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

{

if (rgb)

glColor3ubv(rgb\_colors[i]);

else

glIndexi(mapped\_colors[i]);

glPushMatrix();

glTranslatef(dests[i][0] + offsets[i][0], dests[i][1] + offsets[i][1],dests[i][2] + offsets[i][2]);

glRotatef(angs[i], rotAxis[i][0], rotAxis[i][1], rotAxis[i][2]);

glCallList(theTorus);

glPopMatrix();

}

glPopMatrix();

if (doubleBuffer)

glutSwapBuffers();

else

glFlush();

}

float MyRand(void)

{

return 10.0 \* (drand48() - 0.5);

}

void ReInit(void)

{

int i;

float deviation;

deviation = MyRand() / 2;

deviation = deviation \* deviation;

for (i = 0; i < RINGS; i++)

{

offsets[i][0] = MyRand();

offsets[i][1] = MyRand();

offsets[i][2] = MyRand();

angs[i] = 260.0 \* MyRand();

rotAxis[i][0] = MyRand();

rotAxis[i][1] = MyRand();

rotAxis[i][2] = MyRand();

iters[i] = (deviation \* MyRand() + 60.0);

}

}

void Init(void)

{

int i;

float top\_y = 1.0;

float bottom\_y = 0.0;

float top\_z = 0.15;

float bottom\_z = 0.69;

float spacing = 2.5;

static float lmodel\_ambient[] = {0.0, 0.0, 0.0, 0.0};

static float lmodel\_twoside[] = {GL\_FALSE};

static float lmodel\_local[] = {GL\_FALSE};

static float light0\_ambient[] = {0.1, 0.1, 0.1, 1.0};

static float light0\_diffuse[] = {1.0, 1.0, 1.0, 0.0};

static float light0\_position[] = {0.8660254, 0.5, 1, 0};

static float light0\_specular[] = {1.0, 1.0, 1.0, 0.0};

static float bevel\_mat\_ambient[] = {0.0, 0.0, 0.0, 1.0};

static float bevel\_mat\_shininess[] = {40.0};

static float bevel\_mat\_specular[] = {1.0, 1.0, 1.0, 0.0};

static float bevel\_mat\_diffuse[] = {1.0, 0.0, 0.0, 0.0};

srand48(0x102342);

ReInit();

for (i = 0; i < RINGS; i++)

rgb\_colors[i][0] = rgb\_colors[i][1] = rgb\_colors[i][2] = 0;

rgb\_colors[BLUERING][2] = 255;

rgb\_colors[REDRING][0] = 255

;rgb\_colors[GREENRING][1] = 255;

rgb\_colors[YELLOWRING][0] = 255;

rgb\_colors[YELLOWRING][1] = 255;

mapped\_colors[BLUERING] = BLUE;mapped\_colors[REDRING] = RED;

mapped\_colors[GREENRING] = GREEN;

mapped\_colors[YELLOWRING] = YELLOW;

mapped\_colors[BLACKRING] = BLACK;

dests[BLUERING][0] = -spacing;

dests[BLUERING][1] = top\_y;

dests[BLUERING][2] = top\_z;

dests[BLACKRING][0] = 0.0;

dests[BLACKRING][1] = top\_y;

dests[BLACKRING][2] = top\_z;

dests[REDRING][0] = spacing;

dests[REDRING][1] = top\_y;

dests[REDRING][2] = top\_z;

dests[YELLOWRING][0] = -spacing / 2.0;

dests[YELLOWRING][1] = bottom\_y;

dests[YELLOWRING][2] = bottom\_z;

dests[GREENRING][0] = spacing / 2.0;

dests[GREENRING][1] = bottom\_y;

dests[GREENRING][2] = bottom\_z;

theTorus = glGenLists(1);

glNewList(theTorus, GL\_COMPILE);

FillTorus(0.1, 8, 1.0, 25);

glEndList();

glEnable(GL\_CULL\_FACE);

glCullFace(GL\_BACK);

glEnable(GL\_DEPTH\_TEST);

glClearDepth(1.0);

if (rgb)

{

glClearColor(0.5, 0.5, 0.5, 0.0);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light0\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light0\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light0\_specular);

glLightfv(GL\_LIGHT0, GL\_POSITION, light0\_position);

glEnable(GL\_LIGHT0);

glLightModelfv(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, lmodel\_local);

glLightModelfv(GL\_LIGHT\_MODEL\_TWO\_SIDE, lmodel\_twoside);

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, lmodel\_ambient);

glEnable(GL\_LIGHTING);

glMaterialfv(GL\_FRONT, GL\_AMBIENT, bevel\_mat\_ambient);

glMaterialfv(GL\_FRONT, GL\_SHININESS, bevel\_mat\_shininess);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, bevel\_mat\_specular);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, bevel\_mat\_diffuse);

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE);

glEnable(GL\_COLOR\_MATERIAL);

glShadeModel(GL\_SMOOTH);

}

Else

{

glClearIndex(BACKGROUND);

glShadeModel(GL\_FLAT);

}

glMatrixMode(GL\_PROJECTION);

gluPerspective(45, 1.33, 0.1, 100.0);

glMatrixMode(GL\_MODELVIEW);

}

void Reshape(int width, int height)

{

glViewport(0, 0, width, height);

}

/\* ARGSUSED1 \*/

void Key(unsigned char key, int x, int y)

{

switch (key)

{

case 27 : exit(0);

break;

case ' ' : ReInit();

glutIdleFunc(Idle);

break;

}

}

GLenum Args(int argc, char \*\*argv)

{

GLint i;

rgb = GL\_TRUE;

doubleBuffer = GL\_TRUE;

for (i = 1; i < argc; i++)

{

if (strcmp(argv[i], "-ci") == 0)

{

rgb = GL\_FALSE;

}

else if (strcmp(argv[i], "-rgb") == 0)

{

rgb = GL\_TRUE;

}

else if (strcmp(argv[i], "-sb") == 0)

{

doubleBuffer = GL\_FALSE;

}

else if (strcmp(argv[i], "-db") == 0)

{

doubleBuffer = GL\_TRUE;

}

Else

{

printf ("%s (Bad option).\n", argv[i]);r

eturn GL\_FALSE;

}

}

return GL\_TRUE;

}

void visible(int vis)

{

if (vis == GLUT\_VISIBLE)

{

glutIdleFunc(Idle);

}

else{glutIdleFunc(NULL);

}

}

void speed\_menu(int id)

{

SPEEDLIMIT=4000;

switch(id)

{

case 1: SPEEDLIMIT =SPEEDLIMIT\*2;

ReInit();

glutIdleFunc(Idle);

break;

case 2: SPEEDLIMIT =SPEEDLIMIT\*1.5;

ReInit();

glutIdleFunc(Idle);

break;

case 3: SPEEDLIMIT =SPEEDLIMIT;

ReInit();

glutIdleFunc(Idle);

break;

case 4: SPEEDLIMIT =SPEEDLIMIT/2;

ReInit();

glutIdleFunc(Idle);

break;

case 5: SPEEDLIMIT =SPEEDLIMIT/1000;

ReInit();

glutIdleFunc(Idle);

break;

}

glutPostRedisplay();

}

void olympic\_menu(int id){switch(id)

{

case 1: ReInit();

break;

case 2: exit(0);

break;

}

}

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

{

GLenum type;glutInitWindowSize(400, 300);

glutInit(&argc, argv);

if (Args(argc, argv) == GL\_FALSE)

{

exit(1);

}

type = (rgb) ? GLUT\_RGB : GLUT\_INDEX;

type |= (doubleBuffer) ? GLUT\_DOUBLE : GLUT\_SINGLE;

glutInitDisplayMode(type);

glutCreateWindow("Olympic Logo");

Init();

glutReshapeFunc(Reshape);

glutKeyboardFunc(Key);

glutDisplayFunc(DrawScene);

sub=glutCreateMenu(speed\_menu);

glutAddMenuEntry("Very slow",1);

glutAddMenuEntry("Slow",2);

glutAddMenuEntry("Normal",3);

glutAddMenuEntry("Fast",4);

glutAddMenuEntry("Very Fast",5);

glutCreateMenu(olympic\_menu);

glutAddSubMenu("Re-initialise",sub);

glutAddMenuEntry("Random Position",1);

glutAddMenuEntry("Quit",2);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutVisibilityFunc(visible);

glutMainLoop();

return 0; }