#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <glut.h>

#ifdef GL\_VERSION\_1\_1

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

#ifndef M\_PI

#define M\_PI 3.14159265

#endif

#define MAXSIZE 512 /\* Set this to your maximum texture size (square)

\*/

#define TEXT "OpenGL"

float ang = 2.0;

float scale = 1.05;

float tx = 0.0, ty = 0.0;

int oldx, oldy;

int lmb = 0;

int mmb = 0;

int autospin = 0;

float atime = 0.0;

int smooth = 1;

int seedmode = 0;

float seedsize = 0.1;

int primtype = GL\_LINES;

float primsize = 1.0;

int nprims = 10;

float r, g, b;

float dr, dg, db;

int randomcolours = 0;

/\* returns a random floating point number between 0.0 and 1.0 \*/

float frand(void)

{

return (float) (rand() / 32767.0);

}

void init\_colours(float speed)

{

r = frand(); g = frand(); b = frand();

dr = frand() / speed; dg = frand() / speed; db = frand() / speed;

}

void bounce(float \*n, float \*dn)

{

\*n += \*dn;

if (\*n > 1.0) { \*n = 1.0; \*dn = -\*dn; }

if (\*n < 0.0) { \*n = 0.0; \*dn = -\*dn; }

}

/\* generate pretty colours by bouncing rgb values up and down \*/

void set\_colour(void)

{

if (randomcolours) {

glColor3f(frand(), frand(), frand());

} else {

bounce(&r, &dr);

bounce(&g, &dg);

bounce(&b, &db);

glColor3f(r, g, b);

}

}

/\* seed pattern with some random primitives in centre of screen \*/

void seed(void)

{

int i;

glBegin(primtype);

for(i=0; i<nprims; i++) {

set\_colour();

glVertex2f((frand() - 0.5) \* seedsize, (frand() - 0.5) \* seedsize);

}

glEnd();

}

/\* seed pattern with a circular pattern \*/

void seed\_circle(void)

{

int i;

double a;

glBegin(primtype);

for(i=0; i<nprims; i++) {

a = ((double) i \* 2 \* M\_PI) / nprims;

glColor4f(0.0, 0.0, 0.0, 1.0);

glVertex2d(0.0, 0.0);

set\_colour();

glVertex2d(sin(a) \* (seedsize / 2.0), cos(a) \* (seedsize / 2.0));

}

glEnd();

}

/\* bit of a silly one, this \*/

void seed\_teapot(void)

{

glLoadIdentity();

glTranslatef((frand() - 0.5) \* seedsize, (frand() - 0.5) \* seedsize, 0.0);

glRotatef(frand() \* 360.0, frand(), frand(), frand());

set\_colour();

glutWireTeapot(seedsize);

}

/\* seed with text string \*/

void seed\_text(char \*string)

{

int i;

int width = 0;

for (i = 0; i < strlen(string); i++) {

width += glutStrokeWidth(GLUT\_STROKE\_ROMAN, string[i]);

}

glLoadIdentity();

glScalef(seedsize / 100.0, seedsize / 100.0, seedsize / 100.0);

glTranslatef(-width / 2.0, -50.0, 0.0);

for (i = 0; i < strlen(string); i++) {

set\_colour();

glutStrokeCharacter(GLUT\_STROKE\_ROMAN, string[i]);

}

}

/\* copy screen image to texture memory \*/

void grab\_screen(void)

{

glCopyTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, 0, 0, MAXSIZE, MAXSIZE,

0);

if (smooth) {

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

} else {

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\_DECAL);

glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_FASTEST);

}

void reset(void)

{

ang = 0.0;

scale = 1.0;

tx = ty = 0.0;

autospin = 0;

glClear(GL\_COLOR\_BUFFER\_BIT);

grab\_screen();

}

void redraw(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(tx, ty, 0.0);

glRotatef(ang, 0.0, 0.0, 1.0);

glScalef(scale, scale, scale);

if (autospin) {

ang = 3.0 \* cos(atime);

scale = 1.0 + ( sin(atime / 4.0) \* 0.1) ;

atime += 0.01;

}

/\* draw feedback square \*/

glEnable(GL\_TEXTURE\_2D);

glBegin(GL\_QUADS);

glTexCoord2f(0.0, 0.0); glVertex2f(-1.0, -1.0);

glTexCoord2f(1.0, 0.0); glVertex2f(1.0, -1.0);

glTexCoord2f(1.0, 1.0); glVertex2f(1.0, 1.0);

glTexCoord2f(0.0, 1.0); glVertex2f(-1.0, 1.0);

glEnd();

glDisable(GL\_TEXTURE\_2D);

/\* draw square outline \*/

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-1.0, -1.0);

glVertex2f(1.0, -1.0);

glVertex2f(1.0, 1.0);

glVertex2f(-1.0, 1.0);

glEnd();

/\* seed pattern \*/

glLoadIdentity();

switch(seedmode) {

case 0:

seed();

break;

case 1:

seed\_circle();

break;

case 2:

seed\_teapot();

break;

case 3:

seed\_text(TEXT);

break;

}

/\* grab screen as texture \*/

grab\_screen();

glutSwapBuffers();

}

void mouse(int button, int state, int x, int y)

{

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

lmb = 1;

oldx = x; oldy = y;

}

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_UP) {

lmb = 0;

}

if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_DOWN) {

mmb = 1;

oldx = x; oldy = y;

}

if (button == GLUT\_MIDDLE\_BUTTON && state == GLUT\_UP) {

mmb = 0;

}

}

void motion(int x, int y)

{

if (lmb) {

ang += ((oldx - x) / 4.0 );

scale += ((oldy - y) / 400.0);

oldx = x; oldy = y;

glutPostRedisplay();

}

if (mmb) {

tx += ((float) (x - oldx)) / 500.0;

ty += ((float) (oldy - y)) / 500.0;

oldx = x; oldy = y;

glutPostRedisplay();

}

}

void main\_menu(int i)

{

switch(i) {

case 1:

autospin = !autospin;

atime = 0.0;

break;

case 2:

reset();

break;

case 3:

exit(0);

}

}

void mode\_menu(int i)

{

smooth = i;

}

void seed\_menu(int i)

{

seedmode = i;

}

void prim\_menu(int i)

{

primtype = i;

}

void size\_menu(int i)

{

seedsize = 1.0 / i;

}

void psize\_menu(int i)

{

primsize = (float) i;

glPointSize(primsize);

glLineWidth(primsize);

}

void no\_menu(int i)

{

nprims = i;

}

void colour\_menu(int i)

{

switch(i) {

case 0:

init\_colours(500.0);

randomcolours = 0;

break;

case 1:

init\_colours(100.0);

randomcolours = 0;

break;

case 2:

init\_colours(10.0);

randomcolours = 0;

break;

case 3:

randomcolours = 1;

break;

}

}

int mainmenu;

int modemenu, seedmenu, primmenu, sizemenu, psizemenu, nomenu,

colourmenu;

void create\_menus(void)

{

modemenu = glutCreateMenu(mode\_menu);

glutAddMenuEntry("Chunky", 0);

glutAddMenuEntry("Smooth", 1);

seedmenu = glutCreateMenu(seed\_menu);

glutAddMenuEntry("Primitives", 0);

glutAddMenuEntry("Circle", 1);

glutAddMenuEntry("Teapot", 2);

glutAddMenuEntry("Text", 3);

colourmenu = glutCreateMenu(colour\_menu);

glutAddMenuEntry("Slow", 0);

glutAddMenuEntry("Medium", 1);

glutAddMenuEntry("Fast", 2);

glutAddMenuEntry("Random", 3);

primmenu = glutCreateMenu(prim\_menu);

glutAddMenuEntry("Dots", GL\_POINTS);

glutAddMenuEntry("Lines", GL\_LINES);

glutAddMenuEntry("Triangles", GL\_TRIANGLES);

sizemenu = glutCreateMenu(size\_menu);

glutAddMenuEntry("Tiny", 20);

glutAddMenuEntry("Small", 10);

glutAddMenuEntry("Medium", 5);

glutAddMenuEntry("Large", 2);

nomenu = glutCreateMenu(no\_menu);

glutAddMenuEntry("1", 1);

glutAddMenuEntry("2", 2);

glutAddMenuEntry("3", 3);

glutAddMenuEntry("5", 5);

glutAddMenuEntry("10", 10);

glutAddMenuEntry("20", 20);

glutAddMenuEntry("30", 30);

glutAddMenuEntry("50", 50);

psizemenu = glutCreateMenu(psize\_menu);

glutAddMenuEntry("1", 1);

glutAddMenuEntry("3", 3);

glutAddMenuEntry("5", 5);

mainmenu = glutCreateMenu(main\_menu);

glutAddSubMenu("Texture mode", modemenu);

glutAddSubMenu("Seed mode", seedmenu);

glutAddSubMenu("Seed size", sizemenu);

glutAddSubMenu("Colours", colourmenu);

glutAddMenuEntry("-----------", -1);

glutAddSubMenu("Primitive", primmenu);

glutAddSubMenu("Number", nomenu);

glutAddSubMenu("Size", psizemenu);

glutAddMenuEntry("-----------", 0);

glutAddMenuEntry("Autospin", 1);

glutAddMenuEntry("Reset", 2);

glutAddMenuEntry("Quit", 3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);

glutCreateWindow("Trippy Hallicunation");

glutReshapeWindow(MAXSIZE,MAXSIZE);

glutDisplayFunc(redraw);

glutIdleFunc(redraw);

glutMouseFunc(mouse);

glutMotionFunc(motion);

create\_menus();

init\_colours(100.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1.0, 1.0, -1.0, 1.0, 0.0, 1.0);

glViewport(0, 0, MAXSIZE, MAXSIZE);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glutMainLoop();

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

}

#else

int

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

{

fprintf (stderr, "This program demonstrates a feature which is not in OpenGL Version 1.0.\n");

fprintf (stderr, "If your implementation of OpenGL Version 1.0 has the right extensions,\n");

fprintf (stderr, "you may be able to modify this program to make it run.\n");

return 0;

}

#endif