```
// $Id: spincolors.cpp,v 1.43 2019-03-22 17:43:36-07 - - $

// Draw a spinning triangle in a circle, cycling colors
// red -> yellow -> green -> cyan -> blue -> magenta -> ...

#include <algorithm>
#include <cmath>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
#include <unordered_map>
using namespace std;

#include <GL/freeglut.h>
#include <libgen.h>
#include <sys/time.h>
#include <time.h>

enum class justify {LL, LR, UL, UR};
struct rgbcolor { GLubyte rgb[3] {}; };
struct {
   string name;
   int width {512};
   int height {384};
   rgbcolor pointer {};
   rgbcolor circle {};
   int margin = 5;
   GLfloat radius() { return min (width, height) / 2.0 - margin; };
} window;

const rgbcolor BLACK   {0x00, 0x00, 0x00};
const rgbcolor WHITE   {0xFF, 0xFF, 0xFF};
const rgbcolor RED     {0xFF, 0x00, 0x00};
const rgbcolor YELLOW  {0xFF, 0xFF, 0x00};
const rgbcolor GREEN   {0x00, 0xFF, 0x00};
const rgbcolor CYAN    {0x00, 0xFF, 0xFF};
const rgbcolor BLUE    {0x00, 0x00, 0xFF};
const rgbcolor MAGENTA {0xFF, 0x00, 0xFF};

```

```

string to_string (const rgbcolor& color) {
   ostringstream out;
   out << "0x" << hex << setiosflags (ios::uppercase) << setfill ('0')
       << setw(2) << static_cast<unsigned> (color.rgb[0])
       << setw(2) << static_cast<unsigned> (color.rgb[1])
       << setw(2) << static_cast<unsigned> (color.rgb[2]);
   return out.str();
}

string time_string() {
   struct timeval tv;
   gettimeofday (&tv, nullptr);
   time_t now = tv.tv_sec;
   struct tm tm;
   localtime_r (&now, &tm);
   char timebuf[64];
   strftime (timebuf, sizeof timebuf, "%T", &tm);
   char fracbuf[10];
   snprintf (fracbuf, sizeof fracbuf, ".%02ld", tv.tv_usec / 10'000);
   return string (timebuf) + string(fracbuf);
}

double time_seconds() {
   struct timeval tv;
   gettimeofday (&tv, nullptr);
   constexpr long million = 1'000'000;
   constexpr long fraction = million / 10;
   double microseconds = tv.tv_usec / fraction * fraction;
   return double (tv.tv_sec % 60) + microseconds / million;
}

```

```

void draw_text (justify where, const string& text, int ystep = 0) {
   static void* font = GLUT_BITMAP_9_BY_15;
   auto ustring = reinterpret_cast<const GLubyte*> (text.c_str());
   GLfloat length = glutBitmapLength (font, ustring);
   GLfloat height = glutBitmapHeight (font);
   GLfloat xpos = 0, ypos = 0;
   switch (where) {
      case justify::LL:
         xpos = - window.width / 2.0 + window.margin;
         ypos = - window.height / 2.0 + window.margin;;
         break;
      case justify::LR:
         xpos = window.width / 2.0 - length - window.margin;
         ypos = - window.height / 2.0 + window.margin;;
         break;
      case justify::UL:
         xpos = - window.width / 2.0 + window.margin;
         ypos = window.height / 2.0 - height;
         break;
      case justify::UR:
         xpos = window.width / 2.0 - length - window.margin;
         ypos = window.height / 2.0 - height;
         break;
   }
   ypos += height * ystep;
   glColor3ubv (BLACK.rgb);
   glRasterPos2f (xpos, ypos);
   glutBitmapString (font, ustring);
}

void draw_color (justify where, int index, rgbcolor color) {
   ostringstream buffer;;
   const char name[3] {'R', 'G', 'B'};
   buffer << name[index] << ' ' << fixed << setprecision(3)
          << color.rgb[index] / 255.0;
   draw_text (where, buffer.str(), 3 - index);
}

void draw_point (GLfloat radius, GLfloat degrees) {
   GLfloat xpos = radius * cos (degrees * M_PI / 180.0);
   GLfloat ypos = radius * sin (degrees * M_PI / 180.0);
   glVertex2f (xpos, ypos);
}

```

```

void draw_circle() {
   glBegin (GL_POLYGON);
   glColor3ubv (window.circle.rgb);
   for (GLfloat angle = 0; angle < 360.0; angle += 360.0 / 128.0) {
      draw_point (window.radius(), angle);
   }
   glEnd();
}

void draw_pointer() {
   glBegin (GL_POLYGON);
   glColor3ubv (window.pointer.rgb);
   draw_point (window.radius(), 90.0);
   draw_point (window.radius(), 225.0);
   draw_point (window.radius() * 0.5, 270.0);
   draw_point (window.radius(), 315.0);
   glEnd();
}

void draw_dots() {
   for (size_t step = 1; step < 10; step *= 5) {
      glEnable (GL_POINT_SMOOTH);
      glPointSize (window.radius() / 50.0 * (step == 1 ? 1 : 2));
      glBegin(GL_POINTS);
      glColor3ubv (BLACK.rgb);
      for (size_t dotpos = 0; dotpos < 60; dotpos += step) {
         draw_point (window.radius(), dotpos * 360.0 / 60.0);
      }
      glEnd();
   }
}

```

```

void set_colors (double seconds) {
   if (seconds < 10) {
      window.pointer = RED;
      window.pointer.rgb[1] = round ((seconds / 10.0) * 255.0);
   }else if (seconds < 20) {
      window.pointer = YELLOW;
      window.pointer.rgb[0] = round ((2.0 - seconds / 10.0) * 255.0);
   }else if (seconds < 30) {
      window.pointer = GREEN;
      window.pointer.rgb[2] = round ((seconds / 10.0 - 2.0) * 255.0);
   }else if (seconds < 40) {
      window.pointer = BLUE;
      window.pointer.rgb[1] = round ((4.0 - seconds / 10.0) * 255.0);
   }else if (seconds < 50) {
      window.pointer = BLUE;
      window.pointer.rgb[0] = round ((seconds / 10.0 - 4.0) * 255.0);
   }else {
      window.pointer = MAGENTA;
      window.pointer.rgb[2] = round ((6.0 - seconds / 10.0) * 255.0);
   }
   for (size_t pos = 0; pos < 3; ++pos) {
      window.circle.rgb[pos] = 255 - window.pointer.rgb[pos];
   }
}

void clear_color (const rgbcolor& color) {
   glClearColor (color.rgb[0] / 255.0,
                 color.rgb[1] / 255.0,
                 color.rgb[2] / 255.0, 1.0);
}

```

```

void display() {
   double seconds = time_seconds();
   set_colors (seconds);
   glClear (GL_COLOR_BUFFER_BIT);
   clear_color (WHITE);
   glPushMatrix();
   glRotatef (-seconds * 6.0, 0, 0, 1);
   draw_circle();
   draw_pointer();
   glPopMatrix();
   draw_dots();
   draw_text (justify::LL, to_string (window.pointer));
   draw_text (justify::LR, to_string (window.circle));
   draw_text (justify::UR, time_string());
   draw_text (justify::UL, to_string (window.width) + "x"
                         + to_string (window.height));
   for (int index = 0; index < 3; ++index) {
      draw_color (justify::LL, index, window.pointer);
   }
   for (int index = 0; index < 3; ++index) {
      draw_color (justify::LR, index, window.circle);
   }
   glutSwapBuffers();
}

void reshape (int width, int height) {
   window.width = width;
   window.height = height;
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D (-window.width / 2.0, +window.width / 2.0,
               -window.height / 2.0, +window.height / 2.0);
   glMatrixMode (GL_MODELVIEW);
   glViewport (0, 0, window.width, window.height);
   glutPostRedisplay();
}

constexpr GLfloat frequency_msec = 50;
void timer (int) {
   glutTimerFunc (frequency_msec, timer, 0);
   glutPostRedisplay();
}

```

```

int main (int argc, char** argv) {
   window.name = basename (argv[0]);
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
   glutInitWindowSize (window.width, window.height);
   glutInitWindowPosition (0, 0);
   glutCreateWindow (window.name.c_str());
   glutDisplayFunc (display);
   glutReshapeFunc (reshape);
   glutTimerFunc (frequency_msec, timer, 0);
   glutMainLoop();
   return 0;
}

```