Pseudo-random ramblings about programming and other geeky stuff

Thursday, 10 April 2014

Low-level Graphics on Raspberry Pi (shapes)

By now (this was started a year ago already) I would assume most of the most interested readers having developed their own drawing functions for creating more useful output. But just in case there are newcomers or someone needs a push to the right direction... Let's take a look at drawing some basic shapes.

In part eight we drew some vertical and horizontal lines by repeating calls to `put_pixel` for adjacent pixels. How about sloping lines? Not so trivial (except exactly 1:1 slope) - as one pixel step in one direction may need multiple pixels step in the other direction and normally gaps would not be wanted. Luckily this has been studied decades ago and there exists a well known algorithm for drawing sloping lines called the Bresenham's line algorithm which in our case can be implemented as:

```
void draw_line(int x0, int y0, int x1, int y1, int c) {
  int dx = x1 - x0;
  dx = (dx >= 0) ? dx : -dx; // abs()
  int dy = y1 - y0;
  dy = (dy >= 0) ? dy : -dy; // abs()
  int sx;
  int sy;
  if (x0 < x1)
    sx = 1;
  else
    sx = -1;
  if (y0 < y1)
    sy = 1;
  else
    sy = -1;
  int err = dx - dy;
  int e2;
  int done = 0;
  while (!done) {
    put_pixel(x0, y0, c);
    if ((x0 == x1) && (y0 == y1))
      done = 1;
    else {
      e2 = 2 * err;
      if (e2 > -dy) {
        err = err - dy;
        x0 = x0 + sx;
      }
      if (e2 < dx) {
        err = err + dx;
        y0 = y0 + sy;
      }
    }
  }
}
```

...which should work for any line orientations.

Drawing other shapes becomes a lot easier using the line function - like rectangle outline:

```
// (x0, y0) = left top corner coordinates
// w = width and h = height
void draw_rect(int x0, int y0, int w, int h, int c) {
  draw_line(x0, y0, x0 + w, y0, c); // top
  draw_line(x0, y0, x0, y0 + h, c); // left
  draw_line(x0, y0 + h, x0 + w, y0 + h, c); // bottom
  draw_line(x0 + w, y0, x0 + w, y0 + h, c); // right
}
```

Even filling a rectangle can be implemented using the same:

```
void fill_rect(int x0, int y0, int w, int h, int c) {
  int y;
  for (y = 0; y < h; y++) {
    draw_line(x0, y0 + y, x0 + w, y0 + y, c);
  }
}
```

...for each line/y we draw a horizontal line... Of course it might be a good idea to optimize this using memset()...

The Bresenham's line algorithm has been extended to draw circles as well - this is very clever optimisation only calculating the points for one eight of the circle and just mirroring the rest:

```
void draw_circle(int x0, int y0, int r, int c)
{
  int x = r;
  int y = 0;
  int radiusError = 1 - x;

  while(x >= y)
  {
    // top left
    put_pixel(-y + x0, -x + y0, c);
    // top right
    put_pixel(y + x0, -x + y0, c);
    // upper middle left
    put_pixel(-x + x0, -y + y0, c);
    // upper middle right
    put_pixel(x + x0, -y + y0, c);
    // lower middle left
    put_pixel(-x + x0, y + y0, c);
    // lower middle right
    put_pixel(x + x0, y + y0, c);
    // bottom left
    put_pixel(-y + x0, x + y0, c);
    // bottom right
    put_pixel(y + x0, x + y0, c);

    y++;
    if (radiusError < 0)
 {
   radiusError += 2 * y + 1;
 } else {
      x--;
   radiusError+= 2 * (y - x + 1);
    }
  }
}
```

...and it is trivial to modify this to draw filled circles (in four horizontal slices):

```
void fill_circle(int x0, int y0, int r, int c) {
  int x = r;
  int y = 0;
  int radiusError = 1 - x;

  while(x >= y)
  {
```

```
    // top
    draw_line(-y + x0, -x + y0, y + x0, -x + y0, c);
    // upper middle
    draw_line(-x + x0, -y + y0, x + x0, -y + y0, c);
    // lower middle
    draw_line(-x + x0, y + y0, x + x0, y + y0, c);
    // bottom
    draw_line(-y + x0, x + y0, y + x0, x + y0, c);

    y++;
    if (radiusError < 0)
    {
      radiusError += 2 * y + 1;
    } else {
      x--;
      radiusError+= 2 * (y - x + 1);
    }
  }
}
```

Now use these to draw something - a quick (admittedly pretty much mindless) example:

```
void draw() {

  int x;

  // some pixels
  for (x = 0; x < vinfo.xres; x+=5) {
    put_pixel(x, vinfo.yres / 2, WHITE);
  }

  // some lines (note the quite likely 'Moire pattern')
  for (x = 0; x < vinfo.xres; x+=20) {
    draw_line(0, 0, x, vinfo.yres, GREEN);
  }

  // some rectangles
  draw_rect(vinfo.xres / 4, vinfo.yres / 2 + 10,
            vinfo.xres / 4, vinfo.yres / 4, PURPLE);
  draw_rect(vinfo.xres / 4 + 10, vinfo.yres / 2 + 20,
            vinfo.xres / 4 - 20, vinfo.yres / 4 - 20, PURPLE);
  fill_rect(vinfo.xres / 4 + 20, vinfo.yres / 2 + 30,
            vinfo.xres / 4 - 40, vinfo.yres / 4 - 40, YELLOW);

  // some circles
  int d;
  for(d = 10; d < vinfo.yres / 6; d+=10) {
    draw_circle(3 * vinfo.xres / 4, vinfo.yres / 4, d, RED);
  }

  fill_circle(3 * vinfo.xres / 4, 3 * vinfo.yres / 4,
              vinfo.yres / 6, ORANGE);
  fill_circle(3 * vinfo.xres / 4, 3 * vinfo.yres / 4,
              vinfo.yres / 8, RED);

}
```

...might be a good idea to make sure we are in 8 bit mode for the color constants to work...
Full code in GitHub.

[Continued in next part Text]

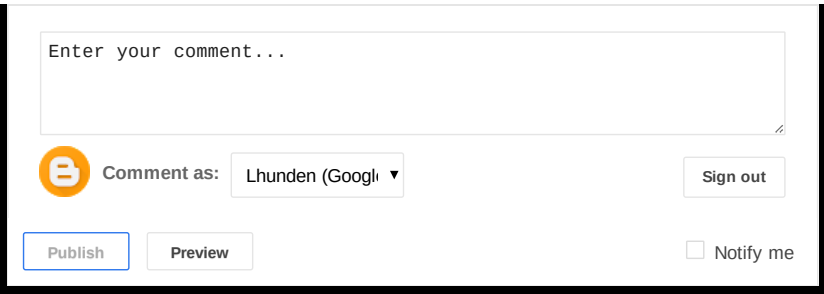
Posted by Unknown at 15:34

Labels: C, graphics, Linux, Raspberry Pi

No comments:

Post a Comment

Note: only a member of this blog may post a comment.



Newer Post Home Older Post

Subscribe to: Post Comments (Atom)