Raspberry Compote

Pseudo-random ramblings about programming and other geeky stuff

Tuesday, 2 April 2013

Low-level Graphics on Raspberry Pi (part seven)

In the previous example, we produced the same image in different display modes (color depths). Let's see if we can find some difference between the modes.

This example draws three color gradient circles partly overlapping:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <sys/mman.h>

// 'global' variables to store screen info
char *fbp = 0;
struct fb_var_screeninfo vinfo;
struct fb_fix_screeninfo finfo;

void put_pixel_RGB24(int x, int y, int r, int g, int b)
{
    // calculate the pixel's byte offset inside the buffer
    // note: x * 3 as every pixel is 3 consecutive bytes
    unsigned int pix_offset = x * 3 + y * finfo.line_length;

    // now this is about the same as 'fbp[pix_offset] = value'
    *((char*)(fbp + pix_offset)) = b;
    *((char*)(fbp + pix_offset + 1)) = g;
    *((char*)(fbp + pix_offset + 2)) = r;

}

void put_pixel_RGB565(int x, int y, int r, int g, int b)
{
    // calculate the pixel's byte offset inside the buffer
    // note: x * 2 as every pixel is 2 consecutive bytes
    unsigned int pix_offset = x * 2 + y * finfo.line_length;

    // now this is about the same as 'fbp[pix_offset] = value'
    // but a bit more complicated for RGB565
    //unsigned short c = ((r / 8) << 11) + ((g / 4) << 5) + (b / 8);
    unsigned short c = ((r / 8) * 2048) + ((g / 4) * 32) + (b / 8);
    // write 'two bytes at once'
    *((unsigned short*)(fbp + pix_offset)) = c;

}
```

Blog Archive

- ► 2016 (6)
- ► 2015 (3)
- ► 2014 (9)
- ▼ 2013 (9)
  - ▼ April (2)
    - Low-level Graphics on Raspberry Pi (part eight)
    - Low-level Graphics on Raspberry Pi (part seven)
  - ► March (4)
  - ► February (1)
  - ► January (2)
- ► 2012 (2)

Code Repository

- Low-level Graphics on RPi

Discussion

- Low-level Graphics on RPi
- Python Programming on RPi
- Java Programming on RPi

Links

- Raspberry Pi
- Python

```
// helper function for drawing - no more need to go mess with
// the main function when just want to change what to draw...
void draw() {

    int x, y;
    int r, g, b;
    int dr;
    int cr = vinfo.yres / 3;
    int cg = vinfo.yres / 3 + vinfo.yres / 4;
    int cb = vinfo.yres / 3 + vinfo.yres / 4 + vinfo.yres / 4;

    for (y = 0; y < (vinfo.yres); y++) {
        for (x = 0; x < vinfo.xres; x++) {
            dr = (int)sqrt((cr - x)*(cr - x)+(cr - y)*(cr - y));
            r = 255 - 256 * dr / cr;
            r = (r >= 0) ? r : 0;
            dr = (int)sqrt((cg - x)*(cg - x)+(cr - y)*(cr - y));
            g = 255 - 256 * dr / cr;
            g = (g >= 0) ? g : 0;
            dr = (int)sqrt((cb - x)*(cb - x)+(cr - y)*(cr - y));
            b = 255 - 256 * dr / cr;
            b = (b >= 0) ? b : 0;

            if (vinfo.bits_per_pixel == 16) {
                put_pixel_RGB565(x, y, r, g, b);
            }
            else {
                put_pixel_RGB24(x, y, r, g, b);
            }
        }
    }

}

// application entry point
int main(int argc, char* argv[])
{

    int fbfd = 0;
    struct fb_var_screeninfo orig_vinfo;
    long int screensize = 0;


    // Open the file for reading and writing
    fbfd = open("/dev/fb0", O_RDWR);
    if (!fbfd) {
      printf("Error: cannot open framebuffer device.\n");
      return(1);
    }
    printf("The framebuffer device was opened successfully.\n");

    // Get variable screen information
    if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)) {
      printf("Error reading variable information.\n");
    }
    printf("Original %dx%d, %dbpp\n", vinfo.xres, vinfo.yres,
       vinfo.bits_per_pixel );

    // Store for reset (copy vinfo to vinfo_orig)
    memcpy(&orig_vinfo, &vinfo, sizeof(struct fb_var_screeninfo));

    // Get fixed screen information
    if (ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo)) {
      printf("Error reading fixed information.\n");
    }

    // map fb to user mem
    screensize = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;
    fbp = (char*)mmap(0,
              screensize,
              PROT_READ | PROT_WRITE,
              MAP_SHARED,
              fbfd,
              0);

    if ((int)fbp == -1) {
        printf("Failed to mmap.\n");
    }
    else {
        // draw...
        draw();
        sleep(5);
    }

    // cleanup
    munmap(fbp, screensize);
    if (ioctl(fbfd, FBIOPUT_VSCREENINFO, &orig_vinfo)) {
        printf("Error re-setting variable information.\n");
    }
    close(fbfd);

    return 0;

}
```

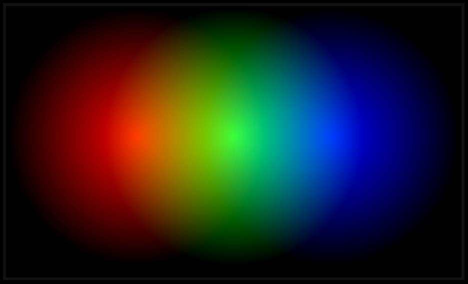
Save the new code (available also on GitHub) to fbtest7.c and compile using this command (as we now use the `sqrt()` function the math library, we need to tell the linker this with the '-lm' directive):

```
gcc -o fbtest7 -lm fbtest7.c
```

And then execute the following sequence:

```
fbset -depth 16
./fbtest7
fbset -depth 24
./fbtest7
```

...note how in the 16 bit mode there are noticeable bands in the color gradients - less so in the 24 bit mode. Note that the above code does not work in the 8 bit mode - it could be modified to produce similar enough image by setting the palette values suitably.



[Continued in part eight]

Posted by Unknown at 11:03

Labels: C, graphics, Linux, Raspberry Pi

No comments:

Post a Comment

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

Enter your comment...

Comment as: Lhunden (Googl ▾ Sign out

Publish Preview □ Notify me

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)