6.3 Linux 视频输出

6.3.1 基本概念

系统中要对这一节要介绍的设备是液晶显示屏，即 LCD（ Liquid Crystal Display 的简称），有必要先对该设备做一个简单的了解。

图 6-8 液晶屏 LCD

LCD 的构造主要是在玻璃基板当中放置液晶膜，基板玻璃上设置 TFT（薄膜晶体管），在其之上还有彩色滤光片，通过 TFT 玻璃上的信号与电压改变来控制液晶分子的转动方向，从而达到控制每个像素点偏振光出射与否而达到显示目的。

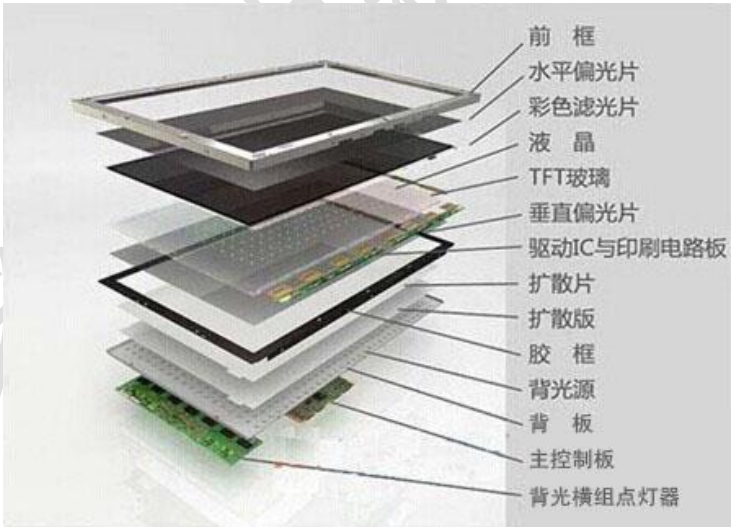

图 6-9 液晶屏的内部结构

下面解释几个名词（部分内容选自维基百科 http://zh.wikipedia.org）：

1，像素（pixel）

像素又称画素，为图像显示的基本单位。这个单词最初的来源指的是“图像元素（picture element）”，在英文中将那两个单词合并，创造了一个新的单词 pixel，这就是所谓的像素。每个这样的信息元素是一个抽象的采样。

每个像素可有各自的颜色值，可采三原色显示，因而又分成红、绿、蓝三种子像素（RGB 色域），或者青、品红、黄和黑（CMYK 色域，印刷行业以及打印机中常见）。照片是一个个采样点的集合，在图像没有经过不正确的/有损的压缩或相机镜头合适的前提下，单位面积内的像素越多代表分辨率越高，所显示的图像就会接近于真实物体。

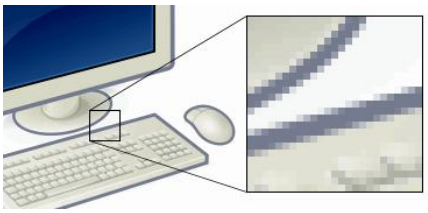

图 6-10 像素点

右图中这个例子显示一组计算机的配件被放大的一部分。不同的灰度混合在一起产生了光滑图像的假相。

2，分辨率（image resolution）

图像效果最重要的指标系数之一是分辨率，分辨率是指单位面积显示像素的数量，在日常用语中之分辨率多用于图像的清晰度。分辨率越高代表图像质量越好，越能表现出更多的细节；但相对的，因为纪录的信息越多，文件也就会越大。个人计算机里的图像，可以使用图像处理软件（例如 Adobe Photoshop、PhotoImpact）调整大小、编修照片等。

下图显示了一系列不同分辨率的图片的差别：

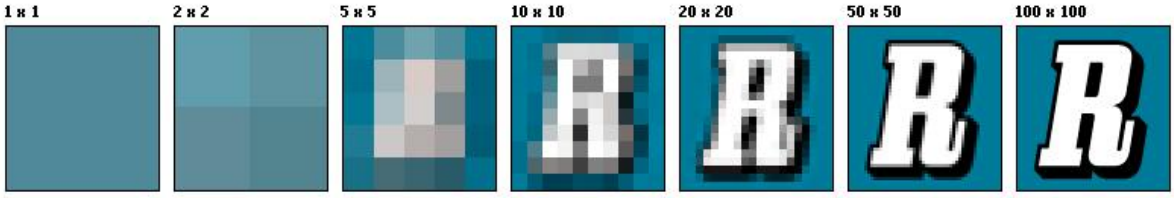

图 6-11 不同分辨率的差异

描述分辨率的单位有：dpi（dots per inch 点每英寸）、lpi（line per inch 线每英寸）和 ppi（pixel per inch 每英寸像素）。但只有 lpi 是描述光学分辨率的尺度的。虽然 dpi 和 ppi 也属于分辨率范畴内的单位，但是他们的含义与 lpi 不同。而且 lpi 与 dpi 无法换算，只能凭经验估算。在实践中，也经常用 X*Y 的方式来表达分辨率的大小，比如上图所示的几张图片。

另外，ppi 和 dpi 经常都会出现混用现象。但是他们所用的领域也存在区别。从技术角度说，"像素"只存在于电脑显示领域，而"点"只出现于打印或印刷领域。

3，色彩深度（BPP, bits per pixel）

一个像素所能表达的不同颜色数取决于比特每像素（BPP,bit per pixel）。这个最大数可以通过取 2 的色彩深度次幂来得到。例如，常见的取值有：

8 bpp：256 色，亦称为"8 位色"。

16 bpp：2^{16}=65536 色，称为高彩色，亦称为"16 位色"。

24 bpp：2^{24}=16777216 色，称为真彩色，通常的记法为"1670 万色"，亦称为"24 位色"。

32 bpp：$2^{24}+2^{8}$，计算机领域较常见的 32 位色并不是表示 2^{32} 种颜色，而是在 24 位色基础上增加了 8 位（2^{8}=256 级）的灰度（亦称"灰阶"，有时亦被实现为 alpha 透明度），因此 32 位色的色彩总数和 24 位色是相同的，32 位色也称为真彩色、或全彩色。

48 bpp：2^{48}=281,474,976,710,656 色，用于很多专业的扫描仪。

256 色或者更少的色彩的图形经常以块或平面格式存储于显存中，其中显存中的每个像素是到一个称为调色板的颜色数组的索引值。这些模式因而有时被称为索引模式。虽然每次只有 256 色，但是这 256 种可以选自一个通常是 16 兆色的调色板，所以可以有多种组合。

对于超过 8 位的深度，这些数位就是三个分量（红绿蓝）的各自的数位的总和。一个 16 位的深度通常分为 5 位红色和 5 位蓝色，6 位绿色（眼睛对于绿色更为敏感）。24 位的深度一般是每个分量 8 位。而 32 位的颜色深度也是常见的：这意味着 24 位的像素有 8 位额外的数位来描述透明度。

简单地讲。一个像素点所对应的字节数目越多，其色彩深度越深，表现力就越细腻。

6.3.2 FRAME BUFFER

首先明确，framebuffer 是一种很底层的机制，他是在 Linux 系统中，为了能够屏蔽各种不同的显示设备的具体细节，Linux 内核提供的一个覆盖于显示芯片之上的虚拟层，将显卡或者显存设备抽象掉，提供给一个统一干净又抽象的编程接口，使得内核可以很方便地将显卡硬件抽象成一块可直接操作的内存，而且还提供了封装好的各种操作和设置，大大提高内核开发的效率。因此 framebuffer 的存在是为了方便显卡驱动的编写，而有时我们会将这个术语用在诸多涉及 Linux 视频输出的场合。

在用户层层面，我们更加不用关心具体的显存位置、显卡型号、换页机制等等细节，而是直接基于 frame-buffer 来映射显存，frame-buffer 就是所谓的帧缓冲机制。

LCD 显示器一般对应的设备节点文件是/dev/fb0，当然如果系统有多个显示设备的话，还可能有/dev/fb1、/deb/fb2 等，这些文件是读写显示设备的入口。回忆 4.2.1 中关于内存映射的概念，我们可以将一个文件的内容映射到一块内存上，按照这个推理，我们可以将 frame-buffer 所抽象的内核物理显存（如果机器没有显卡，那么就是系统分配的一段充当显存的物理内存）映射到用户空间的虚拟内存上，这样一来，我们就可以在应用程序直接写屏了。

要使用 frame-buffer，需要先理解以下的结构体，他们在/usr/include/Linux/fb.h 中被定义：

1，struct fb_fix_screeninfo{ }

这个结构体保存显示设备不能被修改的信息，比如显存（或起到显存作用的内存）的起始物理地址、扫描线尺寸、显卡加速器类别等。具体代码如下：

```
vincent@ubuntu:/usr/include/Linux$ cat fb.h -n
 ......
150  struct fb_fix_screeninfo {
151          char id[16];
152          unsigned long smem_start;  // 显存起始地址（实际物理地址）
153
154          __u32 smem_len;            /* 显存大小 */
155          __u32 type;                /* 像素构成 */
```

```
156     __u32 type_aux;           /* 交叉扫描方案 */
157     __u32 visual;             /* 色彩构成 */
158     __u16 xpanstep;           /* x 轴平移步长（若支持）*/
159     __u16 ypanstep;           /* y 轴平移步长（若支持）*/
160     __u16 ywrapstep;          /* y 轴循环步长（若支持）*/
161     __u32 line_length;        /* 扫描线大小（字节）*/
162     unsigned long mmio_start; /* 缺省映射内存地址 */
163                               /* （物理地址） */
164     __u32 mmio_len;           /* 缺省映射内存大小 */
165     __u32 accel;              /* 当前显示加速器芯片 */
166     __u16 reserved[3];        /* 保留 */
167 };
168
 ……
```

以上信息是由驱动程序根据硬件配置决定的，应用程序无法修改，应用程序应该根据该结构体提供的具体信息来构建和操作 frame-buffer 映射内存，比如扫描线的大小，即一行的字节数。这个大小决定了映射内存的宽度。

2，struct fb_bitfield { }

该结构体保存了色彩构成具体方案。具体代码如下：

vincent@ubuntu:/usr/include/Linux$ **cat fb.h -n**

```
 ……
180 struct fb_bitfield {
181     __u32 offset;             /* 色彩位域偏移量 */
182     __u32 length;             /* 色彩位域长度 */
183     __u32 msb_right;
184 };
185
 ……
```

3，struct fb_var_screeninfo{ }

这个结构体保存显示设备可以被调整的信息，比如可见显示区 X/Y 轴分辨率、虚拟显示区 X/Y 轴分辨率、色彩深度、色彩构成等等。具体代码如下：

vincent@ubuntu:/usr/include/linux$ **cat fb.h -n**

```
 ……
232
233 struct fb_var_screeninfo {
234     __u32 xres;               /* 可见区的宽度分辨率 */
235     __u32 yres;               /* 可见区的高度分辨率 */
236     __u32 xres_virtual;       /* 虚拟区的宽度分辨率 */
237     __u32 yres_virtual;       /* 虚拟区的高度分辨率 */
238     __u32 xoffset;            /* 虚拟区到可见区的宽度偏移量 */
239     __u32 yoffset;            /* 虚拟区到可见区的高度偏移量 */
240
```

```
241     __u32 bits_per_pixel;  /* 色彩深度 */
242     __u32 grayscale;       /* 灰阶（若为非 0） */
243
244     struct fb_bitfield red;       /* 红色色彩位域构成 */
245     struct fb_bitfield green;     /* 绿色色彩位域构成 */
246     struct fb_bitfield blue;      /* 蓝色色彩位域构成 */
247     struct fb_bitfield transp;    /* 透明属性 */
248
249     __u32 nonstd;          /* 非标准像素格式（若为非 0） */
250
251     __u32 activate;        /* 设置参数合适生效 */
252
253     __u32 height;          /* 图片高度（单位毫米） */
254     __u32 width;           /* 图片宽度（单位毫米） */
255
256     __u32 accel_flags;     /* 显示卡选项 */
257
 ......
```

注意到上述代码中的各种分辨率和 X 轴和 Y 轴偏移量，他们的关系决定了 LCD 显示器上显示的效果，可见区和虚拟区的关系如下：

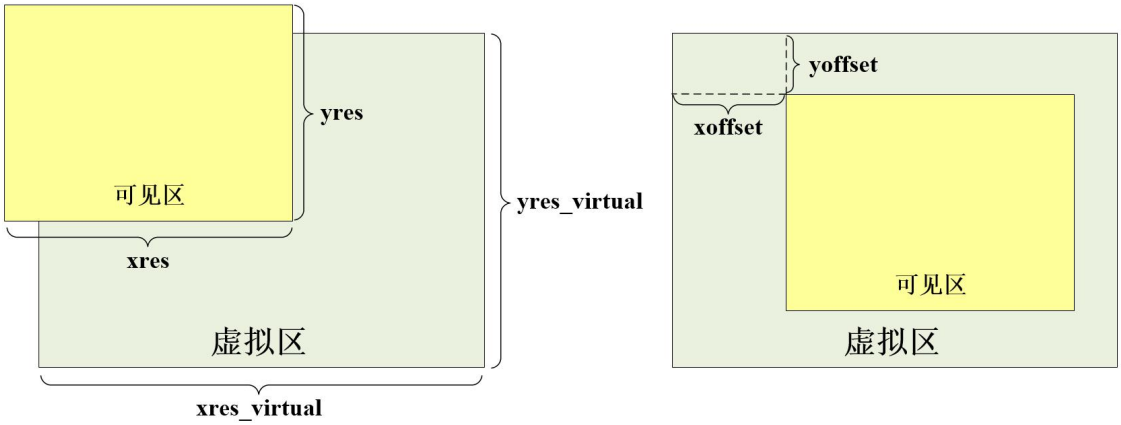

图 6-12 可见区和虚拟区

xres_virtual 和 yres_virtual 决定了虚拟区的大小，而 xres 和 yres 决定了屏幕上可见区域的大小，如果虚拟区比可见区大，我们还可以调整 xoffset 和 yoffset 来显示不同的部分，比如，让 yoffset 逐渐变大，从显示效果上看来，就好像一张图片平滑地向上移动。

图片、映射内存和显示屏的关系示意图如下，他们的像素点一一对应：

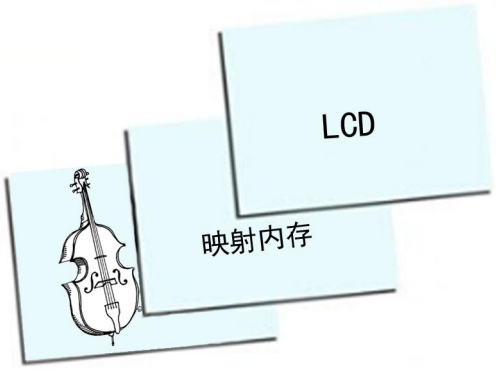

图 6-13 映射内存和 LCD

从上述代码运行结果来看，群创 AT070TN92-7 英寸液晶显示屏缺省的可见分辨率为 800 * 480，缺省的虚拟分辨率为 800*960。

可见分辨率反映了 LCD 屏幕上的真实情况：其在横向长度上有 800 个像素点，在纵向宽度上有 480 个像素点，而且他的色彩深度 BPP 是 32 位真彩色，我们只要在这四个字节上填充相应的数据，就可以控制该像素点显现不同的颜色，如图：

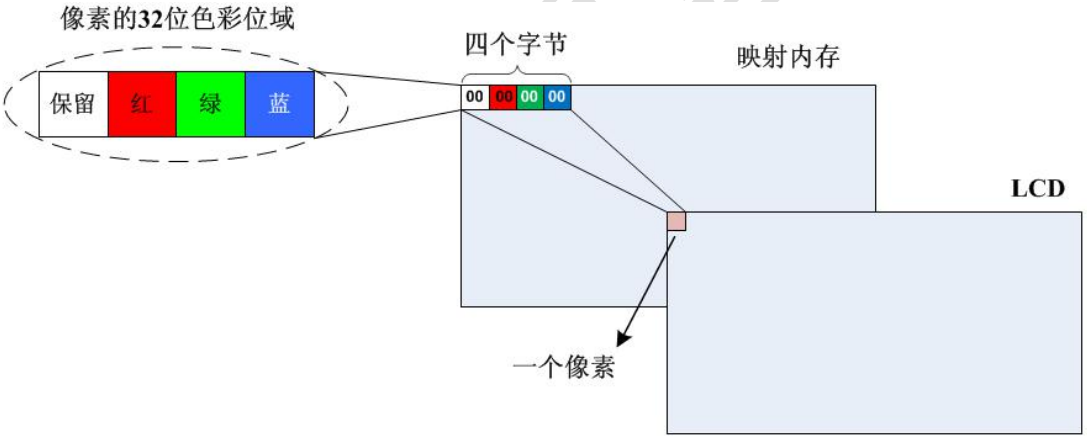

图 6-14 色彩深度