ECE 598 – Advanced Operating Systems
Lecture 19

Vince Weaver

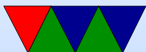
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

9 April 2015

Announcements

- HW#5 returned
- HW#6 and HW#7 posted soon



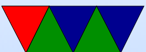
Linux graphic interface

- originally, none. VGA Text only
  X11 drove software directly.

- Attempt at GGI/KGI, Linus nixed it

- Framebuffer devices got in. Why? Well some machines had no textmode without it

- Gradually the DRI interface (Direct Rendering Interface) started providing abstractions needed for modern video



cards.

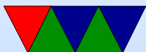
DRI1/DRI2/DRI3

DRM – event queueing?

KMS – kernel mode setting

GEM/TTM – memory allocation

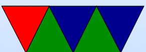
MESA3D – handles OpenGL translation



Higher Level

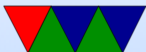
- X11 – client/server, network transparent
  MIT, 1984

- Wayland – Compositing Manager is mandatory
  Draw to an offscreen buffer, window manager copy to screen
  Can have 3d compositor, fancy effects



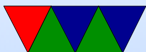
Even Higher

- Libraries like Qt, Gtk, (historically Motif)
- Desktops like KDE, GNOME, XFCE



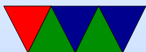
Raspberry Pi Framebuffer

- Pi *can* do advanced 3D GPU graphics.
  Not documented well (but getting better)
  But it is complex, more than we need for a simple OS
- The GPU firmware does provide for a simple flat framebuffer mode if you ask it nicely



Raspberry Pi Mailbox Interface

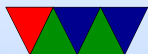
- How the ARM CPU communicates with the GPU that really run things
- Mailbox channels: MAILBOX_POWER 0
  MAILBOX_FRAMEBUFFER 1
  MAILBOX_VIRT_UART 2
  MAILBOX_VCHIQ 3
  MAILBOX_LED 4
  MAILBOX_BUTTONS 5
  MAILBOX_TOUCHSCREEN 6



- Mailbox

Address	Size	Name	Description	R/ W
0x2000b880	4	Read	Receive mail	R
0x2000b890	4	Poll	Check mail	R
0x2000b894	4	Sender	Sender info	R
0x2000b898	4	Status	Infor	R
0x2000b89c	4	Config	Settings	RW
0x2000b8a0	4	Write	Send mail	W

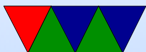
- to send to a mailbox:
  – sender waits until the Status field has a 0 in the



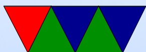
MAIL_FULL bit

- sender writes to Write such that the lowest 4 bits are the mailbox to write to, and the upper 28 bits are the message to write.

- To read a mailbox:
  - receiver waits until the Status field has a 0 in the MAIL_EMPTY
  - receiver reads from Read.
  - receiver confirms the message is for the correct mailbox, and tries again if not.



- Talk to GPU through this mailbox interface. Lots of things set in it (the GPU is in control on Pi). Things like power, clock enables, etc.

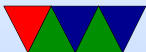


Raspberry Pi Framebuffer Interface

- You can send it an address to a piece of memory to use as a framebuffer and it will draw it to the screen over HDMI.

- ```
  struct frame_buffer_info_type {
          int phys_x,phys_y;      /* IN: Physical Width / Height*/
          int virt_x,virt_y;      /* IN: Virtual Width / Height */
          int pitch;              /* OUT: bytes per row */
          int depth;              /* IN: bits per pixel */
          int x,y;                /* IN: offset to skip when copying fb */
          int pointer;            /* OUT: pointer to the framebuffer */
          int size;               /* OUT: size of the framebuffer */
  };
  ```

- Write the address of FrameBufferInfo + 0x40000000 to

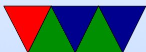

  ```

mailbox 1 (40000000 means don't cache)

Read the result from mailbox 1. If it is not zero, we didn't ask for a proper frame buffer.

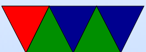
GPU firmware returns a framebuffer you can write to.

Copy our images to the pointer, and they will appear on screen!

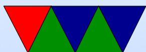


Using a Framebuffer

- How big is it?
- Why might it not just be X*Y*(bpp/8) bytes big? Alignment issues? Powers of two? Weird hardware reasons?
- Things like R/G/B order, padding bits, bits grouped together (on Apple II groups of 7 bytes), etc
- Otherwise it's just an exercise is calculating start address and then copying values

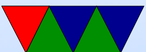


- How do you calculate colors?



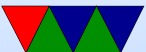
Putting a Pixel

- Depends a bit on the graphics mode you request
- For simplicity, request 800x600x24-bit
- Get back pointer, size, pitch
- Each X row has R,G,B bytes repeated for each pixel
- To get to next row increment by pitch value (bytes per row)
  fb[(x*3)+(y*pitch)]=r



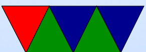
fb[(x*3)+(y*pitch)+1]=g
fb[(x*3)+(y*pitch)+2]=b

- pitch returned by the GPU. Normally it would just be (maxy*bpp)/8, but it can vary depending on how the hardware arranges the bits.



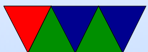
Drawing a Gradient

- Just draw a horizontal line, incrementing the color for each line



Console Display

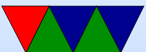
- Font / VGA Fonts
- console framebuffer. Color?
- scrolling
- backspace
- ANSI emulation



Bitmapped Font

- Each character an 8x8 (or 8x16, or similar) pattern
- ```
  unsigned char smiley[8]={
      0x7e,   /*     ******     */
      0x81,   /*    *       *   */
      0xa5,   /*    * *   * *   */
      0x81,   /*    *       *   */
      0xa5,   /*    * *   * *   */
      0x99,   /*    *   **  *   */
      0x81,   /*    *       *   */
      0x7e,   /*     ******     */
  };

  void put_smiley(int xoff, int yoff, int color) {
      for(y=0;y<8;y++) {
          for(x=0;x<8;x++) {
              if (simley[y]&(1<<(7-x))) {
                  putpixel(color,x+xoff,y+yoff);
  ```


  ```

```
            }
        }
    }
}
```

- Can find source of fonts online, VGA fonts. Just a binary set of bitmapped characters indexed by ASCII code.
- Usually 8x16 though; the custom font used in the homework is a hand-made 8x8 one