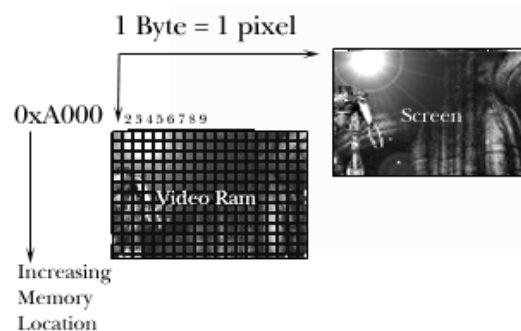


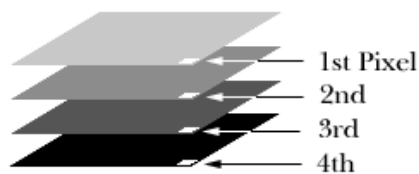
Using Unchained Video Modes

Intro

At first glance, Unchained video modes seem VERY confusing! Don't worry, this document should explain all there is to accessing video ram in this odd configuration. Normal Video Ram is Linear or Non-Planar or in Chained Mode. Each says the same thing, for each and every pixel to the right of our current position, go right one into Vram and that memory location corresponds to that pixel on the screen. Here's a little pic for ya.



As you see here, each pixel corresponds to exactly one byte in video ram. This is very nice to use, very easy to program and it just makes sense! Now imagine if each spot in video ram marks a spot, and a little flag tells us that we can access 1 of 4 pixels. In this scheme we would move right in video ram every 4 pixels. To plot our first 2 pixels we would say something like `plot(0xa000,1)` for the 1st pixel `plot(0xa000,2)` for the 2nd. To plot pixels 5 and 6 we would say `plot(0xa000+1,1)` `plot(0xa000+1,2)`. See how we are using the two markers to move through video ram? Here's how Video Ram is arranged under an Unchained Mode.



A Little Story

Imagine we are a master electrician and we just got done installing 64 panels of lights. Each panel can have 4 intensities, not just on or off! To control this huge array of lights we have a knob that selects which panel to control, and a switch that will control its brightness. If we wanted to test our wiring, we could turn our knob to position 0 and flip the switch to its 4 positions. Everything looks good, great! To go to the next panel we turn the knob to 1 and flip our switch, the light flickers a little, but its ok! To test panel 3 we turn the knob to 2 and repeat our switch test. This movement is how we are going to traverse through Video Ram. Just think that every time you turn the knob or move the switch, you will be going to the next pixel on the screen.

Unchaining Video Ram

```
void Unchain()
{ asm (“ movw $0x3c4,%%dx
      movw $0x604,%%ax
      out %%ax,%%dx
      movw $0xf02,%%ax
      out %%ax,%%dx

      movw $0x3d4,%%dx
      movw $0x0014,%%ax
      out %%ax,%%dx
      movw $0xE317,%%ax
      out %%ax,%%dx “
      :
      :
      :”%ax”,”dx”);
}
```

or Normal C++

```
void Unchain()
{ outpw(0x3c4,0x604);
  outpw(0x3c4,0xf02);
  outpw(0x3d4,0x0014);
  outpw(0x3d4,0xe317);
}
```

Here we are turning the CHAIN4 mode of the Sequencer off, turning off the CRTIC's Long mode and turning on the byte mode!

Selecting the Plane to Write

```
outp(0x3c4,0x02);
outp(0x3c5,(0x01<<plane));
```

With two outp commands we select the second index under the Sequence Controller (0x3c4) which is the Map Mask Register (0x2) , which controls which plane we are allowed to write to. If we set all bits or send a 0xf to port 0x3c5, then any 8 bit value we write will set all 4 pixels to that value. This is great for quick clearscreen functions! In our function, plane is the plane we want to set.

To move an offscreen buffer to Video Ram in Unchained Mode, it would look something like this.

```

void MoveBuffer()
{ long bufferposition;

  for(short position=0;position < 64; position++)
  { for(short planenumber=0; planenumber<4; planenumber++,bufferposition++)
    { outp(0x3c4,0x02);
      outp(0x3c5,(0x01<<planenumber));
      memset(0xa000+position,buffer[bufferposition],1);
    }
  }
}

```

As you can see this was made to be easy to look at. In reality your routines will take a performance dive since moving memory in this scheme is more difficult. As you progress and understand how Video Ram is organized you can get back the performance that was lost and final exploit the video card to its full potential!

Contact Information

I just wanted to mention that everything here is copyrighted, feel free to distribute this document to anyone you want, just don't modify it! You can get a hold of me through my website or direct email. Please feel free to email me about anything. I can't guarantee that I'll be of ANY help, but I'll sure give it a try :-)

Email : Justin Deltener <deltener@mindtremors.com>

Webpage : <http://www.inversereality.org>

