The Atari ST M68000 tutorial part 15 – on fading to black

It has occurred to me that by striving ever forward, we've forgotten to speak about some basic things, so for this tutorial and the next one, we'll be taking a step back and reviewing some things. You may have guessed these techniques yourself, but it never hurts to have it spelled out. Also, I thought I'd share some new thoughts on development, we'll take that first.

Most of the source for the tutorials in the past I've actually written in Devpac on a real Atari, but it has now become clear to me that developing in Windows on an IBM compatible is easier and more efficient. I got the tip over at www.atari-forum.com, a discussion forum for all topics Atari (where I'm one of the moderators for the coding section, yay). Have one "launcher file" with only one line

```
include whateveryoursourcename.s
```

By doing this, you'll assemble any source files you want, and you can edit those source files outside of Devpac, and then assemble them in Devpac. When I wrote this tutorial, I had a file named _WRAP.S that had the line "include tut15.s" in it. Then I used Ultraedit (my editor of choice) to edit TUT15.S, I also had Devpac running under STEem. Whenever I felt like assembling my source, I just saved in Ultraedit, alt-tabbed into STEem and hit alt-a to assemble my source; smooth and easy.

Speaking of Ultraedit, there is a topic going on over at http://www.atari-forum.com/viewtopic.php?t=946 to try and work out good syntax high lightning for Atari assembly in Ultraedit (www.ultraedit.com). Wow, that's a lot of various things you wouldn't have seen pop up in a tutorial from say 1994. Now onto the serious stuff.

The palette is an extremely powerful thing when you want to change colors quick and easy. Unfortunately it has the obvious limitation of not changing the pixels. Using the palette you can black out the screen without erasing the contents (by setting all colors to black), make things pulse (by incrementing and decrementing color intensity) or wait with displaying a picture. Say you want to calculate a big fractal, just set the palette to all 0, calculate your fractal, then whap in the palette to display the result. The effect will be that no one will see you draw the fractal, only the final result will be shown.

As we've been through before, there are 16 colors in the palette, the first one being the background color, located at $ff8240. Each color is a word long, making the palette end at $ff825e. Each word is built up like this

```
00000RRR0GGG0BBB
```

The first three bits control blue intensity, then there's a zero bit, the next three bits control green intensity, a zero and the final (non-zero) three bits control red intensity. The maximum value you can get out of three bits is 8, and since the color intensities are at 4 bit boundaries, they are very easy to access in hex (since each character in hex mode is a 4 bit quantity). Thus $700 means max intensity of red and zero intensity for green and blue, $444 means medium intensity for all three colors.

When they built the STe, they thought that it would be nice to have more colors in the palette, and indeed, it's easy to just add an additional bit since that would still have the palette on a 4 bit boundary, making each color range from 0-15. However, there was a problem, they could not add a bit in the beginning and just shift the other bits to the left, since that would mean all old palette values would in effect be shifted left one bit creating an entirely different value than was originally intended.

The solution to this problem is cunning, but unfortunate. They added the least significant bit where the zero bit used to be. This maintains backwards compatibility, and adds 8 new possible color intensities. So the STe palette looks like this

```
0000rRRRgGGGbBBB
```

This means that $700 is still (almost) maximum intensity of red. What in the memory is perceived as the most significant bit, is in palette terms the least significant bit. This sounds very confusing perhaps, but just picture moving the uppermost bit of each color intensity first. Let's say then that we want the intensity between $100 and $200, this would be color $900, since that would be

```
0000rRRRgGGGbBBB
0000100100000000
```

Which we can interpret as

```
0000RRRrGGGgBBBb
0000001100000000
```

Thus, when using the STe palette, we must think about the fact that the most significant bit for each color, is in actuality the least significant bit. The number order for intensities, from lowest to highest is 0, 8, 1, 9, 2, A, 3, B, 4, C, 5, D, 6, E, 7, F. So if you use color $fff, the STe will interpret this as intensity 15 for all colors, and the ST will interpret it as color intensity 7, since the ST doesn't care about whether the fourth bit is set or not.

That should be all there is to the palette, making full utilization of it will be up to each one. In order to do something I thought we'd just do a simple fade in effect. Fading in a picture is so much nicer than just whipping it onto screen. Fading out is also much nicer than just zapping it away, you can also fade to white and make the screen sort of flash away.

What we want is to begin with a black palette and pixel data on the screen, then increment the color values of the palette until they reach the values intended for the picture. In order to keep things simple, I opted to skip the STe palette since there's lots of shifting involved whenever you want to use it. So the fade will only have a maximum of 7 intensities to work with, making it a pretty bad looking fade effect.

We'll need a copy of the original palette, and a current palette which we increment until it reaches the original. It would be tempting to compare the real palette to the current one and add $111 (one intensity of each color) if they don't match, but that won't work. Say one color is supposed to be $100, if we compare our current $000 with that, they don't match, so we

add $111 making the current color $111, which is more than $100. Instead, we must compare each red, green and blue value individually. This can easily be done by just masking off all bits except the three controlling the intensity for either red, green or blue.

```
            and.w       #%011100000000,d0     mask off all but red values
            and.w       #%011100000000,d1     mask off all but red values

            cmp.w       d1,d0                 see if red is correct intensity
            beq         red_fin               if not ...
            add.w       #%000100000000,d1     ... add one intensity of red
red_fin
```

Let's assume d0 holds the real color, and d1 holds the temporary. All bits except the ones controlling red are masked off, then values compared. If they do not match, add one to the value. The value to add will be different depending on which intensity we check for, since different intensities begin at different bit positions. That's pretty much it, here's the entire source

```
            section text

            jsr         initialise

            movem.l     picture+2,d0-d7       put picture palette in d0-d7
            movem.l     d0-d7,pal             copy palette to pal

            movem.l     temp_pal,d0-d7        put current palette in d0-d7
            movem.l     d0-d7,$ff8240         apply current palette (all 0)

            move.w      #2,-(a7)              get physbase
            trap        #14
            addq.l      #2,a7

            move.l      d0,a0                 a0 points to screen memory
            move.l      #picture+34,a1        a1 points to picture

            move.l      #7999,d0              8000 longwords to a screen
loop
            move.l      (a1)+,(a0)+           move one longword to screen
            dbf         d0,loop

            move.l      $70,old_70            backup $70
            move.l      #main,$70             start main routine

            move.w      #7,-(a7)              wait keypress
            trap        #1
            addq.l      #2,a7

            move.l      old_70,$70            restore $70

            jsr         restore

            clr.l       -(a7)
            trap        #1

main
            move.w      sr,-(a7)              backup status register
            or.w        #$0700,sr             disable interrupts
            movem.l     d0-d7/a0-a6,-(a7)     backup registers
```

```
            add.l       #1,counter              increment counter variable
            cmp.l       #15,counter             only execute main sometimes
            bne         do_nothing              skip instructions
            clr.l       counter                 reset counter

            move.l      #pal,a0                 a0 points to values to reach
            move.l      #temp_pal,a1            a1 points to current values

            rept        16                      do for each color
            jsr         check_red               see if red intensity should
increase
            jsr         check_green             see if green intensity should
increase
            jsr         check_blue              see if blue intensity should
increase
            add.l       #2,a0                   point to next color
            add.l       #2,a1                   point to next color
            endr

            movem.l     temp_pal,d0-d7          put current palette in d0-d7
            movem.l     d0-d7,$ff8240           apply current palette

do_nothing
            movem.l     (a7)+,d0-d7/a0-a6       restore registers
            move.w      (a7)+,sr                restore status register
            rte                                 finnished interrupt


check_red
            move.w      (a0),d0                 move one final color into d0
            move.w      (a1),d1                 move one temp color into d1

            and.w       #%011100000000,d0       mask off all but red values
            and.w       #%011100000000,d1       mask off all but red values

            cmp.w       d1,d0                   see if red is correct intensity
            beq         red_fin                 if not ...
            add.w       #%000100000000,(a1)     ... add one intensity of red
red_fin
            rts


check_green
            move.w      (a0),d0                 move one final color into d0
            move.w      (a1),d1                 move one temp color into d1

            and.w       #%000001110000,d0       mask off all but green values
            and.w       #%000001110000,d1       mask off all but green values

            cmp.w       d1,d0                   see if green at correct
intensity
            beq         green_fin               if not ...
            add.w       #%000000010000,(a1)     ... add one intensity of green
green_fin
            rts


check_blue
            move.w      (a0),d0                 move one final color into d0
            move.w      (a1),d1                 move one temp color into d1
```

```
            and.w       #%000000000111,d0     mask off all but blue values
            and.w       #%000000000111,d1     mask off all but blue values

            cmp.w       d1,d0                 see if blue at correct intensity
            beq         blue_fin              if not ...
            add.w       #%000000000001,(a1)   ... add one intensity of blue
blue_fin
            rts



            include     initlib.s

            section data
old_70                  dc.l        0
picture                 incbin      sleepsun.pi1
counter                 dc.l        0

            section bss
pal                     ds.w        16
temp_pal                ds.w        16
```

First I save the palette of the picture in a storage space, then I put the temporary palette in, since the temporary palette is initialized to all 0's, this has the effect of blacking out the screen. Next I load up the picture as described in tutorial 6 and set up the main routine.

The counter code is for delay purposes; otherwise the fade effect would hardly be visible. I make a0 point to the palette to reach, and point a1 to the temporary one. Then I check the individual intensities, and add 2 to each pointer in order to point to the next color, repeating this for the number of colors in the palette, namely 16.

You will notice that the check sub-routines are a bit different than the one described above, I add to the value pointed to by a1, which is the current palette. It may be considered slightly bad program habit to just assume that a1 points to the current palette like that, but coding demos and assembly in general depends on tight kept code that knows what it's doing. Besides, the tutorials aren't really for teaching you how to make good code; they are intended as basic introductions to various coding techniques.

That's that, one easy effect achieved by manipulating the palette. If you want to fade to white, just set the temporary palette to the real palette, and increment until you reach $777. If you want to experiment, I suggest trying to implement the effect with a STe palette instead, the included picture has an STe palette so it's ready to go. This should involve shifting the fourth bit of each color intensity down as the first when adding to the color intensity, and then shift it back. For the next tutorial, I think we'll handle full screen scrolling, without moving any picture data!

perihelion of poSTmortem, 2003-03-28

"I wish for this night-time
to last for a lifetime
The darkness around me
Shores of a solar sea

Oh how I wish to go down with the sun
Sleeping
Weeping
With you”
- Nightwish, Sleeping Sun