Junk Modes

this was published in Vandalism News #43 /gpz

instead of an essay….

I was long thinking about writing a tutorial about coding, and the (obviously) real bad demos at ms99 finally kicked me to write something about all the little things that were discovered over the last decade of the scene. this beginning series of tutorials wont help you with learning 6502 assembly, and i will assume that you know certain basics. it'll bore the majority i guess, if i started with 'hello world' type stuff and explain the differences about 'OR' and 'AND'. You should know about handling your favourite assembler or monitor and should have a memory-map handy for reference, when it comes to advanced shit in later issues. Beeing able to knock up a well done intro should be a good level - If you wanna learn how to display a sprite in ml, or how to play a tune in irq, i'd advice you to get some decent book or other docs, or better, ask some other lazy scener

Instead it'll tell you lots of cheap tricks to make your lame code look like 31337 demos and let you win all major compos the next decade of the scene. (rotfl)

honestly, i had this wild dream of 2nd generation 14 years old c64 enthusiasts which squeze their first intro out of turbo-asm (you lucky bastards, can you imagine that most of the old legendary code was made in a ml-monitor?) and which dream of making a nice demo. I have seen a lot of 'unknown' ppl releasing at ms99 lately, and if i can just make one of the upcoming demos better, all the effort will be worth it.

so here it is…

Demo Factory 1 - grafix junk (graphic modes part 1)

This time i will just quickly tell you about what i consider known as 'junkmodes', that is basically the 8×8 and 4×4 based stuff that doesnt need any fld/fli based routine to be displayed. Now don't cry NOOO! we dont want this crap, but also if you dont like it you should get the basic idea since a lot of the more advanced stuff later will need a certain understanding of this shit.

These modes seem to be liked by the coders, since they are basically much easier to handle and thus also perform much better than for eg hires. Even if these modes have the obvious drawback of very low resolution and (if not handled clever) bad flicker in interlaced variations, they can be used for quite a bit of nice effects. (But please do me a favour and dont do yet another plasma!)

please notice that incase i speak of 'pixels' i may mostly refer to the 8×8, 4×4 or whatever pixel wide 'area' that is used to represent the smallest 'independent' square in the choosen mode.

basic junkmodes

these modes basically use the videoram for pixel data and do not use the color-ram at all. I've silently 'forgotten' to mention the 8×8 char mode (16 cols) you may use by filling the videoram by some char and using color-ram for your fx. This crap rarely looks even 'good enough' so just forget about it

1) 8×8 - Characters (40×25, 256 'patterns')

this is maybe the easiest, and most speedy gfx mode that is possible. it is done by using a charset that either consists of an 'animation' that is 256 steps long (for eg a jumping dot, a spinning quadrangle or sth, whatever... use your stoned brain to create sth that looks freaky or you have sort of 'colorfade' with semi-randomly setted pixels.

advantages:

- you can linearly adress your 'pixels'
- you can linearly adress your 'colors'/patterns

tips:

- its always a good idea to make that 'animation' loopable, that is, if you simply increment some value in the videoram, there shouldnt be glitches when it wraps from $ff to $00 (always nice to find out you dont have to care about upper/downer bounds in your code 8=))
- when using a colorfade, try different versions of 'patterns'. some effects may look better with 'random' style charset, other look just great with 'checkerboard' style things.
- usage of the extended color mode will let you use 4 colors and still hires-pixels, that just needs a clever designed characterset (i still dont see why ppl do monochrome hires effects)

2) 8×8 - Bitmap (40×25, 128 'colors')

this mode uses a bitmap with a 'checkerboard' pattern, so beeing able to display one of 128 'mixed' colors in a 8×8 area by writing 2 colors to high and lowbyte of the matching videoram position.

advantages:

- you can linearly adress your pixels

disadvantages:

- mode with most clumsy appearence of them all

3) 4×4 - Characters (80×50, 4 'colors')

This is a good compromise between speed and resolution, and quite useable for routines that would be to slow in 4×4-fli mode. It is achieved by using a character-set that represents all 256 combinations of 4×4 wide blocks possible in a cursor with 4 colors. The adressing can be compared with hires a little, if you arrange your charset that for eg. bit0-1 represent color 0, bit2-3 represent color 1, etc... you get the idea

tips:

- instead if filled blocks in 4 different colors, you may try using 'checkerboard' style 4×4 blocks or

any other 4 steps animation.
- take care of how your colors mix in resolution-interlaced modes.

4) 8×4 - Bitmap (40×50, 16 'colors') / 4×8 - Bitmap (80×25, 16 'colors')

Since in this modes 'pixels' are not squared, they usually dont look to well imho, but still it may be worth a try. These work with a bitmap that consists of 8×4 pixel wide 'blocks' (either vertical or horizontal) made of alternating multi-color bit-patterns (eg fill bitmap with $5a for vertical version). with this you will double resolution on one axis with the drawback of a bit more complicated pixel and color adressing and a slightly strange appearance to the eye.

tips:

- try (faked or real) resolution interlace to get back a squared pixel-appearance

interlace

1) faked resolution interlace

this is what most coders seem to add to their junkmode routines, that is simply alternating the softscroll register(s) each frame half a 'pixel' width to smoothen the ugly fat pixels. this habit is mostly the reason for those effects looking so bad and flickering like hell, so *please* think twice before doing it - and IF you use it, use clever colors that do mix very well so you get minimum flicker. (colors with matching luminance are said to work well in that case *G*)

2) color interlace

this uses two alternating videorams in order to 'mix' those. with clever colortables you will be able to use quite a lot of colors that dont flicker to much (luminances do the trick again). notice this works also with the charset-modes to get 'mixed' patterns. itll need a bit of additional code maybe to update the second videoram, but not that much.

3) 'real' resolution interlace

this uses two videorams, which are displayed alternating but with an offset of half a 'pixel' width (in the direction at which you wanna gain resolution). This will look much better than the faked thing, but it will also require to calculate the double amount of 'pixels' and so prolly take twice as much time to calculate a full frame. nevertheless if your fx is really fast, this is what you really should try.

'color-ram'

why not using the color-ram you may think, and you are right! BUT: think first if it will really make sence in your situation. Using the coloram will maybe give you some more flexibility, let you use some more colors or even let you overlay a totally different second effect that just works in colorram, but the additional code needed to handle it will also slow down everything, and in the worst case make your effect look worse than before.

tips:

- with some effects (eg tunnel style things) a static color-ram pattern looks even better than a 'real',moving color-ram effect.

other cheap tricks

- making every second line of the screen the same color as the background (in your bitmap or character set, whatever) gives your effect a 'scanline' type of appearance and it also fools the eye of the viewer a bit so the clumsy pixels arent that obvious, and you could additionaly use resolution interlace in x-direction.
- overlaying a checkerboard in background-color over your fx (same method as with the scanline-stuff) achieves a simelar, but still different looking thing. this trick doesnt really make sence combined with resolution interlace. (notice how your checkerboard-patterns would overlay in the interlace!)
- in modes with a checkerboard-type bitmap or character set, its a good idea to have a second 'inverse' version of it that you can display alternating each frame. This will smooth the mixed colors in bitmap modes quite a bit, and also doesnt flicker too much. this doesnt really make sence when combined with any other interlace-fx.
- if it fits to the style of your demo, let the background (or what ever you think looks good) flash to your basedrum or sth. (fools the eye a bit again and also let the whole picture 'pump' a bit on most 'bulk' display hardware)
- add little 'overlayed' style elements (sprites) whenever possible, these act as so called 'eye-catchers' (even more when they move a bit, flash, whatever) and thus again help to fool the viewer.
- in bitmap modes, dont forget that it is still a bitmap capable of more than an alternating pattern. If made cleverly it may be possible to overlay an entire multicolor-picture above your effect, and automagically speed up your effect as well, since less pixel need to be calculated (in unrolled fx loops, leave out any processing for videoram positions that match with your gfx)
- putting a 'border' (made of sprite-gfx for eg) around your effect has several advantages. a) simply looks much better than 'just' the fx alone b) reduces the amount of pixels to be calculated a little without the effect appearing 'smaller' to the viewer c) possibly reducing the flicker in resolution-interlaced modes due to 'smoother' borders.

appendix - luminance values

luminance is the 'brightness' portion of a color that is beeing displayed by your monitor. Also, if you connect a color-source to a black/white monitor or tv-set, the picture is made of the luminance values.

notice:

- colors with the same luminance look the same on a black/white monitor and can not be distinguished
- lumincance is NOT the average of the color's respective RGB-values (luminance/chrominance kind of color-system is 'weighted' on the subjective appearance of the respective color-component to a 'normed human eye') - it equals the physical 'energy' that hits the viewers eye. (thus another common expression for luminance is 'intensity'). The conversion from RGB to chroma/luma color system is (if good accuracy is desired) rather complex (at least one matrix-multiply needed), and shouldnt be too useable for realtime processing.
- luminance levels differ on different VIC-revisions, basically there are 2 different cases: either the luma-levels are breaken up into 5 steps (1-5 in the table) only or there are 9 different steps (1-5 in the table, also a-b-c are different levels). In other words, on some VIC's for eg 'orange' has the same luminance level as 'green' (= both level 3), on other revisions they have quite different levels (orange=level 3a,green=level 3c) - keep that in your mind when designing

color-tables etc. (simply always assume the 9-level version to avoid any hazzle)

```
1)  0    black
2a) 6,9    blue, brown
2b) 2,11    red, dark grey
3a) 4,8    violet, orange
3b) 12,14    middle grey, light blue
3c) 5,10    green, light red
4a) 3,15    cyan, light grey
4b) 7,13    yellow, light green
5)  1    white
```

last not least

so far so good... hope you liked this little start a bit, and you found a little also that isnt just old news for you. however, i had to start *somewhere* ... so, you ask what will maybe come next, and i'll tell you... before i'll go on with “grafix modes 2” (all c64 'standard' modes, hires and multicolor bitmap/sprites/charsets etc) i will give you some insight to “gaining speed - learn how screen-buffering becomes your best friend” (all about single, doubble and tripple buffering) which will be just the consequence to follow-up the previous tutorial. After all that we will come to “what was fld again?” (cookbook for fld/fli routines)“ which will lead us to “grafix modes 3” (last not least telling you about all those with an fld/fli-type of display-routine). At this point, we'll see what comes up next, i am thinking of both some more technical crap (inside irq/nmi, hacking $d011) as well as coding-style (self-generating/-modifying code, unrolled loops vs zeropage code) and maybe also some shit dedicated to specific demo-fx (wanna know how plasma worx? hehe).

so long... code your ass off! 😃

groepaz/hitmen, april, 26th, 1999 groepaz@gmx.net (feedback welcomed!)

... the C=ommunity loves you!