Smooth Linecrunching

(note: this article has a slight bug in that each crunched line scrolls the rest of the screen up by 7 pixels, not 8, so the example code actually scrolls the screen up by 7/8ths of the value read out of the sine table. Still works well enough for demonstration purposes, but something to keep in mind if you want to use it as the starting point for an AGSP routine or anything else that requires precise positioning).

Basically linecrunching is a VIC-trick that allows you to easily move graphics vertically on the screen. It is especially useful for bitmaps, where it would take far too much time to move it by copying.

On a more technical level, linecrunching is taking a full char-line of graphics and compressing it into a single raster-line. This is done by changing the YSCROLL value in $d011 on a badline, but before the VIC starts stealing CPU cycles (cycle 14 to be exact), to make the badline-condition no longer true. Simplified, this basically makes the VIC start reading from the next char-line.

If you continuously do this for several badlines, lines will continuously be crunched, and the VIC will eventually loop back to reading the start of the screen. It also can read the normally invisible last $18 bytes of the screen data. This causes the VIC to actually wrap to a little bit past the beginning of the screen data, causing a weird split-effect...

The problem with simply linecrunching is that it's pretty rough. If you tried scrolling it would look quite jumpy and just generally bad. We need to find some way to make the scrolling SMOOTH, but how??

Before rasterline $30, you must change the YSCROLL through $d011 to the soft-scroll value you want to use. But doing that will also change the line the first badline occurs on. We must account for this, and start linecrunching on the correct rasterline, via IRQ or some waitloop.

Now, getting the hardscroll value is easy, just divide by 8. This value is also how many char-lines to crunch.

Now there is one last problem- it looks smoother, but it is still quite jumpy... let's explain why. The greater the amount of crunched char-lines is, the more the screen scrolls up. BUT the greater the YSCROLL value is, the more the screen scrolls DOWN. Easy fix, all you have to do is flip the YSCROLL value.

All of the lines you crunched “build up” at the top of the screen, which can look quite bad... in the code example I used an illegal gfx mode to “cover” it up.

Finally, here is the code. 64tass syntax.

```
sin_index               = $fb
char_lines_to_crunch    = $fc
initial_yscroll         = $fd

            * = $0801
            .word $080b,0 ;basic stub
            .byte $9e,$32,$30,$36,$31
            .byte 0,0,0

            sei
            lda #$35
            sta 1
            lda #$1b
```

```
            sta $d011
            lda #<nmi
            sta $fffa
            lda #<irqa
            sta $fffe
            lda #>irqa
            sta $fffb
            sta $ffff
            lda #$7f
            sta $dc0d
            sta $dd0d
            lda $dc0d
            lda $dd0d
            lda #1
            sta $d01a
            ldx #$17 ;get rid of ugly post-screen chars
            lda #$20
-           sta $07e8,x
            dex
            bpl -
            inc $d019
            cli
mainloop    lda $d011 ;wait for new frame
            bpl *-3
            lda $d011
            bmi *-3
            inc sin_index
            ldy sin_index
            ldx sintbl,y
            txa
            lsr
            lsr
            lsr
            sta char_lines_to_crunch
            txa
            and #7
            eor #7 ;linecrunch scrolls up, but increasing the yscroll
scrolls down
                   ;therefore we must flip the yscroll to make it match
            sta initial_yscroll
            clc
            adc #$30-3 ;before linecrunching takes 3 rasterlines
            sta $d012
            lda initial_yscroll
            ora #$50
            sta $d011
            lda #$18 ;use the invalid textmode to "cover up" the linecrunch
bug area
            sta $d016
            jmp mainloop
```

```
            ;stable raster routine. not exactly necessary for linecrunch,
but might as well :)
irqa        tsx
            lda #<irqb
            sta $fffe
            inc $d012
            inc $d019
            cli
            .fill 40,$ea

irqb        txs
            ldx #8
            dex
            bne *-1
            .byte 4,$ea ;nop $ea
            lda $d012
            cmp $d012 ;last cycle of CMP reads data from $d012
            beq + ;add extra cycle if still the same line
+           ldx #9
            dex
            bne *-1
            ;this is the part that actually performs the linecrunch
            clc
            lda initial_yscroll
            ldx char_lines_to_crunch
crunchloop  adc #1
            and #7
            ora #$50
            sta $d011
            ldy #8 ;notice that there is lots of time left in the rasterline
for more effects!
            dey
            bne *-1
            nop
            nop
            .byte 4,$ea ;nop $ea
            dex
            bpl crunchloop
            and #%10111111 ;disable invalid gfx mode
            sta $d011
            lda #8
            sta $d016
            lda #<irqa
            sta $fffe
            inc $d019
nmi         rti

            ;precalculated sine-table
            .align $100
sintbl      .byte
100,102,105,107,110,112,115,117,120,122,124,127,129,131,134,136,138,141,143,
```

```
145,147,149,151,153,156,158,160,162,163,165,167,169,171,172,174,176,177,179,
180,182,183,184,186,187,188,189,190,191,192,193,194,195,196,196,197,198,198,
199,199,199,200,200,200,200,200,200,200,200,200,199,199,199,198,198,197,196,
196,195,194,193,192,191,190,189,188,187,186,184,183,182,180,179,177,176,174,
172,171,169,167,165,163,162,160,158,156,153,151,149,147,145,143,141,138,136,
134,131,129,127,124,122,120,117,115,112,110,107,105,102,100,98,95,93,90,88,8
5,83,80,78,76,73,71,69,66,64,62,59,57,55,53,51,49,47,44,42,40,38,37,35,33,31
,29,28,26,24,23,21,20,18,17,16,14,13,12,11,10,9,8,7,6,5,4,4,3,2,2,1,1,1,0,0,
0,0,0,0,0,0,0,1,1,1,2,2,3,4,4,5,6,7,8,9,10,11,12,13,14,16,17,18,20,21,23,24,
26,28,29,31,33,35,37,38,40,42,44,47,49,51,53,55,57,59,62,64,66,69,71,73,76,7
8,80,83,85,88,90,93,95,98
```