Flagged Bucket Sort

By Christopher Jam.

The following was a run at a fast worst case perfect sort, as may be useful for a bullet hell shooter or other application with fast moving sprites.

The general idea is to maintain a list of 220 buckets, one for each y-position at which a sprite may be visible, and to skip over unused buckets by having a small (27 byte) table of flags.

For any non-zero flag byte, a lookup table used to quickly determine the index of the least significant one bit, so that that that bucket can then be emptied by pushing each sprite index from the list starting with that bucket entry onto the stack. The used buckets are cleared as they are emptied, so no post-cleanup is required, and the untouched buckets take no CPU time at all.

By packing eight flags into each flag byte, and unrolling the code to one block per flag byte, the worst case overhead per bucket is only 1.25 cycles. Thus, the CPU time for a perfect sort is only marginally worse than what it would be for a sort routine that only used half as many buckets.

Code is formatted for ACME, but should be fairly straightforward to translate for other assemblers.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;
;;
;; flagged bucketsort, a fast worst case perfect sort by Christopher Jam,
2017
;;
;;
;; Takes at most 38.5 rasterlines per frame for 32 sprites, regardless of
movement.
;;
;; Intended for use in fast random movement sprite multiplexors, but be
aware
;; that Doynax's radix sort is about 12% faster, and probably takes less RAM
;; This was a failed attempt to do better, but I wanted to document it in
;; case anyone else has ideas for improvement.
;;
;;
;;
;; Usage:
;;
;;     jsr sort_init  ;  once to prepare memory
;;
;;   then each time a sorted list is needed,
;;
;;     jsr sort               ; to create a sorted list of indices of all
entries containing y values in the range [ys..ye)
;;     jsr sort_get_next_in_y ; repeatedly to get index of next entry. $ff
```

```
is returned once once list is empty.
;;
;;   alternately, just read sorted list from $017f down, $ff terminated
;;
;; Required defines:
;;
;;     SORT_YS        ; start of range of non-added values (eg first visible
sprite position at top of screen, 30)
;;     SORT_YE        ; one past end of range of non-added (eg nonvisible y
position at bottom of screen, 250)
;;     SORT_BY        ; address table of values to sort (eg actor Y
position)
;;     SORT_LENGTH    ; length of sort_by table. Typically 32.
;;
;; Memory usage:
;;
;;   values output from $0160 to $017f
;;   internal tables at $0080 to $0080+SORT_LENGTH+NFLAGS (typically $80-$ba
inclusive)
;;
;;   1280 bytes of tables defined at the end of this file
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;


YBASE  =  SORT_YS  ; earlier versions masked this with $f8 to align the
group boundaries.
NFLAGS = (SORT_YE-YBASE)>>3
sort_next    = $80               ; SORT_LENGTH bytes.  For each actor, the
index of the next actor in that bucket, or 255
sort_flagset = $80+SORT_LENGTH    ; NFLAGS bytes.  for each line in ys..ye,
a bitflag to say whether there are any actors in that bucket


sort_init
    ldx#SORT_LENGTH-1
    lda#$ff
-   sta sort_next,x
    dex
    bpl -
    ldx#NFLAGS-1
    lda#$00
-   sta sort_flagset,x
    dex
    bpl -
    rts

sort_get_next_in_y
    dec sort_get_next_in_y+4
    ldy $180
```

```
    rts


sort
!zn sort {
    !for .sid, 0, SORT_LENGTH-1 {

    ldy SORT_BY+.sid          ;
    lda flagset_bit,y         ;   8
    ;beq +                    ;  leave this out, it slows down the worst
case.
    ldx flagset_index,y       ;
    ora sort_flagset,x        ;
    sta sort_flagset,x        ;  12

    lda#.sid                  ;
    ldx bucket,y              ;
    sta bucket,y              ;
    stx sort_next+.sid        ;  14   34
+
    }


    tsx
    stx .restore_stack+1
    ldx#127
    txs
    inx
    stx sort_get_next_in_y+4

    !for flagindex, 0, NFLAGS-1 {
    !zn {

; worst case timing is where every flagbyte has at least one flag set, and
each bucket has at most one actor.

    ldx sort_flagset+flagindex                ;
    beq .noflags                              ;          5     *27
.nextflag

    stx .get_remaining_flags+1                ;
    ldy log_least_bit,x                       ;  8
    lda bucket+YBASE+flagindex*8,y            ;
.pn
    pha                                       ;  7
    tax                                       ;
    lda sort_next,x                           ;
    bpl .pn                                   ;  8

    sta bucket+YBASE+flagindex*8,y            ;  5
.get_remaining_flags
```

```
    ldx least_one_cleared                       ;
    bne .nextflag                               ;  6      36      *32

    stx sort_flagset+flagindex                  ;          3      *27
.noflags



    }
    }       ; total 27*8+32*34+(32-27)=1309=32*40.91


    lda#255
    pha    ; push end marker

.restore_stack
    ldx#0
    txs
    rts
}




!align 255,0

bucket !fill 256,$ff  ; buckets.  For each bucket, the index of the first
actor in that bucket.

least_one_cleared
!for x, 0,255 {
    !byte x&(x-1)
}
log_least_bit
    !byte 255
!for x, 1,255 {
    !byte ((((x&-x)*0x55)&128)>>7)+ ((((x&-x)*0x33)&128)>>6)+ ((((x&-
x)*0x0f)&128)>>5)
}
flagset_bit
!for y, 0,255 {
    !if y>=SORT_YS and y<SORT_YE {
        !byte 1<<((y-YBASE)&7)
    } else {
        !byte 0
    }
}
flagset_index
!for y, 0,255 {
    !if y>=SORT_YS and y<SORT_YE {
        !byte ((y-YBASE)>>3)
    } else {
```

```
        !byte 0
    }
}
```

A minimal testbed:

```
    *= $0801
    !byte $0b,$08,$0a,$00,$9e,$32,$30,$36,$31,0,0,0
    sei
    jsr sort_init
    jsr sort
    ldx#0
-   jsr sort_get_next_in_y
    bmi done
    lda values,y
    sta $0400,x
    inx
    bne -
done
    jmp done

values
    !byte $30,$31,$32,$33,$34,$35,$36,$37,$38,$39
    !byte $30,$31,$32,$33,$34,$35,$36,$37,$38,$39
    !byte $39,$38,$37,$36,$35,$34,$33,$32,$31,$30
    !byte $39,$38,$37,$36,$35,$34,$33,$32,$31,$30

;; sort configuration

SORT_YS = 30          ; first visible y position at top
SORT_YE = SORT_YS+220 ; first nonvisible y posiion at bottom (ie, positions
30 to 249 inclusive can be seen)
SORT_BY = values      ; key to sort by
SORT_LENGTH  = 40
```