

# **8bit demo coding horror**

Μιχάλης Κάργας (Optimus)

0x375 – 0x08

18 Ιουλίου 2012

# Γιατί;

- Νοσταλγία
- Περιέργεια
- Πρόκληση
- Μαγικά Κόλπα!

# ZX Spectrum 48k/128k/+2/+3



- CPU: Z80 @ 3.5Mhz
- Memory: 48k / 128k
- Graphics
  - 256 \* 192 – 1bpp
  - 32 \* 24 color attributes
  - 16 color palette
- Sound
  - Beeper (48k)
  - Yamaha AY-3-8912 (rest)
- Special Weapons
  - None

# Amstrad CPC 464/664/6128



- CPU: Z80 @ 4Mhz
- Memory: 64k / 128k
- Graphics
  - 160 \* 200 – 4bpp
  - 320 \* 200 – 2bpp
  - 640 \* 200 – 1bpp
  - 27 color pallete
- Sound
  - Yamaha AY-3-8912
- Special Weapons
  - CRTC controller  
(overscan, hardware scrolling (sort of), screen splits, etc)

# Commodore 64



- CPU: 6502 @ 0.985Mhz
- Memory: 64k
- Graphics
  - 160 \* 200 – 2bpp
  - 320 \* 200 – 1bpp
  - Char/multicolor modes
  - 16 color palette
- Sound
  - SID 6581
- Special Weapons
  - VIC chip (hardware sprites, pixel perfect hardware scrolling, line interrupts, char modes)

# Ενότητες

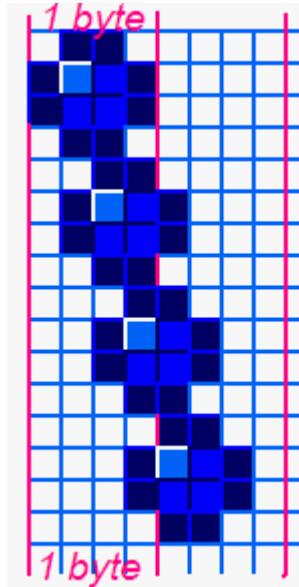
- Sprites/Simple blitting
- Line based effects
- Pixel based effects
- Unrolled code tricks
- 3D effects

# Ένα byte την ημέρα..

- Το πιο απλό που μπορείς να κάνεις: Copy Paste γραφικά
- Έξυπνος τρόπος για γρήγορο blitting
  - Χρησιμοποίηση της στόιβας στον Z80 για γρήγορο blitting
    - LD SP,end\_of\_line\_addr
    - PUSH HL: PUSH HL: PUSH HL: PUSH HL many times
  - Πολύ χρήσιμο ειδικά στον Spectrum

# Sprites

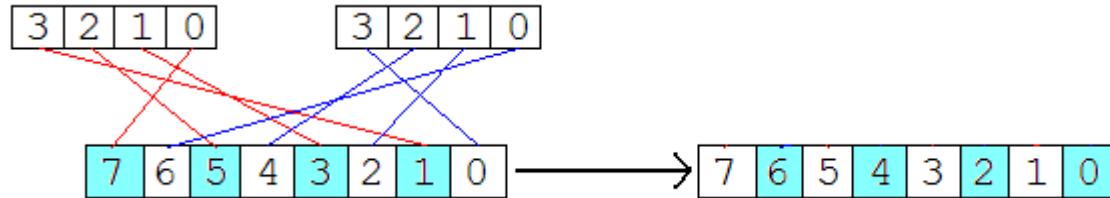
- Ας υποθέσουμε πως κάνουμε απλό copy paste (σε μαύρο background)
  - Πρόβλημα! Ένα byte = πολλά pixels. Σπαστή κίνηση στο X.
- Κλασική λύση; Αποθήκευση shifted sprite versions.



- Μειονεκτήματα: Τετραπλάσια μνήμη (και μην ξεχνάμε και τις μάσκες)

# Sprites

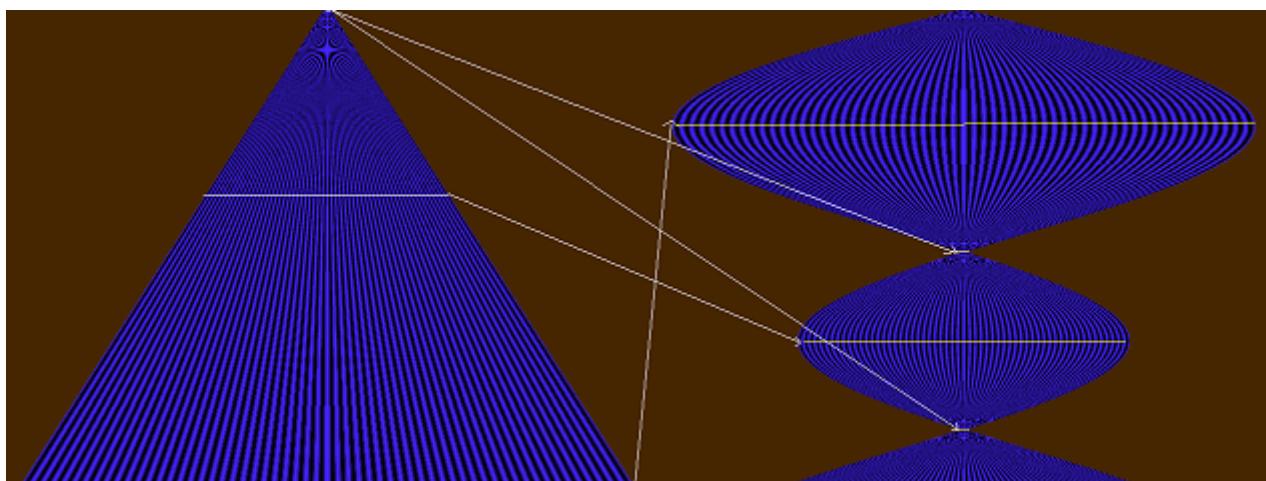
- Δεν προτιμούταν η μέθοδος με τα 4 sprite versions σε παιχνίδια (καλύτερα λιγότερη ταχύτητα παρά γεμάτη μνήμη)
- Καλύτερο για demos με ένα sprite, π.χ. Πόσες μπάλες μπορείς να βγάλεις σε 50hz.
- Σε παιχνίδια είχαμε τις εξής τεχνικές:
  - Bit shifting (το pixel configuration του amstrad ευνοούσε σχετικά)



- Mirroring.
- Σε sprite records έχουμε optimal unrolled codes για διάφορα shifted versions του sprite και στο X και στο Y.

# Γραμμή γραμμή..

- Τα πιο κλασικά εφφέ στα 8bits
- Με γρήγορο blitting διαφορετικών γραμμών γραφικών που εναλάσσονται με το χρόνο γίνεται μια μεγάλη γκάμα εφφέ (αυτό εξαρτάται από το set γραφικών που χρησιμοποιείς)
  - Στον Spectrum θα το κάνεις με software blits
  - Στους Amstrad/C64 με hardware tricks



# Πως γράφω pixel μαμά;

- Δυσκολία εύρεσης γρήγορης λύσης λόγω μη φιλικού screen configuration στα περισσότερα 8bit μηχανήματα
- Γενικά, θέλουμε να κάνουμε conversion από X,Y συντεταγμένες σε video memory address
- Έπειτα, ένα byte είναι πολλά pixels, οπότε η X συντεταγμένη θα μας πει πως να γράψουμε αυτό το pixel χωρίς να καταστρέψουμε την πληροφορία για τα υπόλοιπα.

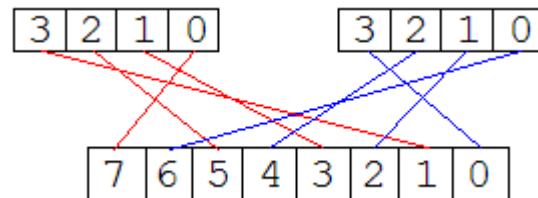
# Πως γράφω pixel μαμά;

- Παράδειγμα σε Amstrad CPC
  - Η videoram ξεκινάει στην &C000
  - Δεν είναι ακριβώς γραμική (ακολουθεί video)
    - 80 bytes ανά γραμμή
    - Add 2048 για να πας στην επόμενη γραμμή εκτός και αν  $Y \% 8 = 7$
    - Τότε αφαιρείς  $7 * 2048$  για να πας 7 γραμμές πίσω και προσθέτεις 80 για να πας στο επόμενο char line.
    - Προσθέτεις και το offset του X, π.χ. το pixel X=100 σε videoram 2bpp είναι  $100/4 = 25$ bytes δεξιά.

$$\text{address} = (Y \% 8) * 2048 + (Y / 8) * 80 + (X / 4);$$

# Πως γράφω pixel μαμά;

- Ποια bits πρέπει να κάνουμε set ώστε να γράψουμε αυτό το pixel χωρίς να αλλοιώσουμε τα γειτονικά pixels που ανοίκουν στο ίδιο byte;
- Θυμηθείτε και πόσο περίεργο είναι το configuration των pixels, π.χ. στο 4bpp mode.



- Προσωπικά δεν έχω κάνει το decoding σε κώδικα. Προτιμάται τρόπος με precalcs tables.

# Πως γράφω pixel μαμά;

- Πως (δεν) θα το έκανες στον Amstrad
  - $\text{address} = (\text{Y \% 8}) * 2048 + (\text{Y / 8}) * 80 + (\text{X / 2})$
  - Μερικά setbit ή κάποιο convert από πίνακα του στυλ  $\text{final\_byte} = \text{pretab}[\text{X \% 1}, \text{Color}]$
  - $\text{vram}[\text{address}] |= \text{final\_byte}$
- Πως το κάνουμε σε ανθρώπινα μηχανήματα
  - $\text{vram}[\text{Y} * \text{ScreenWidth} + \text{X}] = \text{color}$

# Πως γράφω pixel μαμά;

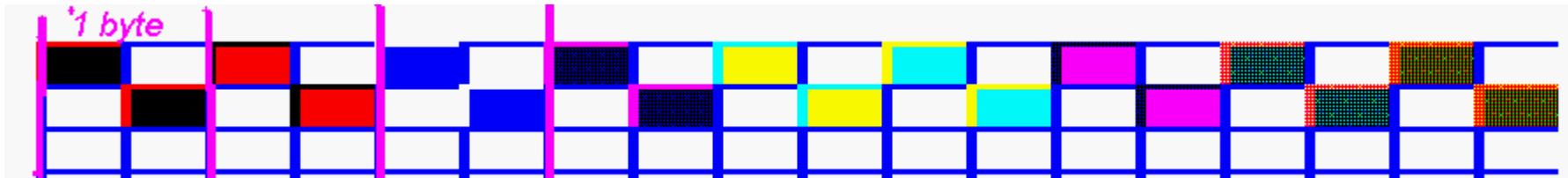
- Πως το κάνουμε στον Amstrad (σε 4bpp)
  - Init Precalc table για το Y
$$\text{vram}[Y] = (Y \% 8) * 2048 + (Y / 8) * 80$$
  - Final address =  $\text{vram}[Y] + (X \gg 1)$
  - Δεν γλιτώνουμε τον πίνακα  $\text{final\_byte} = \text{prefab}[X \% 1, \text{Color}]$
  - $\text{vram}[\text{address}] |= \text{final\_byte}$
  - Πιο απλή αντιστοίχηση αν χρησιμοποιούμε ας πούμε μόνο ένα χρώμα στην dot routine μας.

# Πως γράφω pixel μαμά;

- Για bitmap effects όπου θέλουμε individual pixel control δεν ακολουθείται η ίδια διαδικασία
- Μα φυσικά, αφού το επόμενο pixel είναι στο X+1 και στο ίδιο Y
- Σε 4bpp υπολογίζουμε αριστερό και δεξί pixel color byte, τα ενώνουμε και γράφουμε το byte στην address και μετά πάμε στο αμέσως επόμενο byte.
- Δεν ξεχνάμε στο τέλος της γραμμής με τον precalc table[Y] να κάνουμε reset την address στην επόμενη γραμμή από κάτω και πρώτο byte.

# Πως γράφω pixel μαμά;

- Σε 4bpp συνηθίζω να έχω table με τα byte για left pixels και right pixels από χρώμα 0-15.



- Πολλά μπορούν να βελτιωθούν. Αν έχεις texture mapping based effect, αποθήκευσε το texture με left shifted pixels και right shifted pixels διπλά. Γλυτώνεις την μετατροπή.
- Αλλιώς τα gradient based effects (π.χ. fire, plasma) μπορούν να γίνουν και με πιο χοντρά pixels με precalced γραφικά από dithered tiles. Οπότε μπορείς να γράφεις full bytes πιά και να αποφύγεις την παλιά διαδικασία.

