

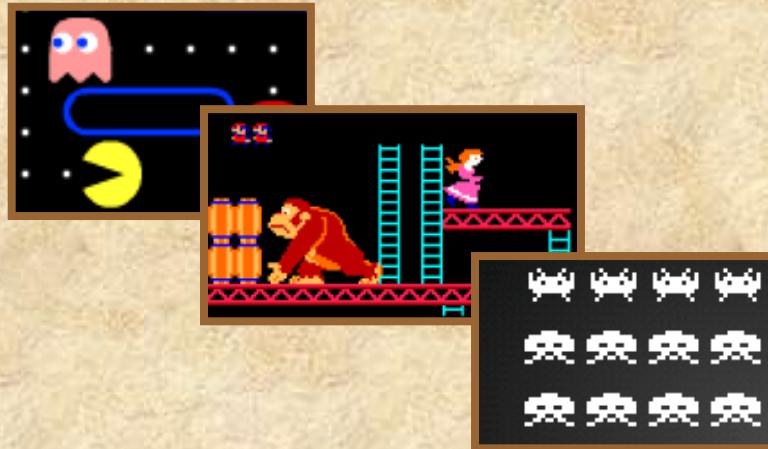


**HY454 : ΑΝΑΠΤΥΞΗ ΕΞΥΠΝΩΝ ΔΙΕΠΑΦΩΝ ΚΑΙ
ΠΑΙΧΝΙΔΙΩΝ**

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ**



**ΔΙΔΑΣΚΩΝ
Αντώνιος Σαββίδης**



ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΩΝ, Διάλεξη 3η



Περιεχόμενα

- **Screen depth and resolution**
- Bitmaps and bit block transfer
- Direct bitmap/ video access
- Color and key masking
- Bounding boxes and collision detection
- Display frequency and vertical retrace synchronization
- Double buffering, page flipping, triple buffering
- Pixel-level primitives and operations



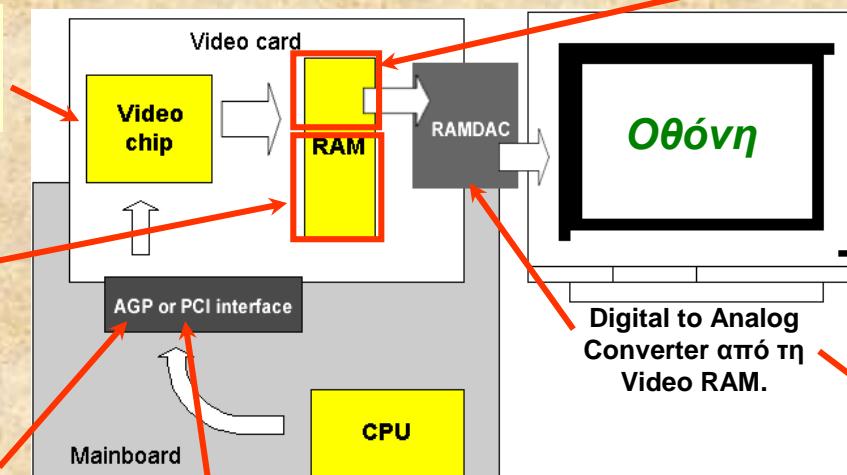
Screen depth and resolution (1/10)

- Η κάρτα γραφικών ουσιαστικά «οδηγεί» την οθόνη και υποστηρίζει τις εναλλακτικές αναλύσεις (resolutions)
- Ότι εμφανίζεται στην οθόνη βρίσκεται αποθηκευμένο σε συγκεκριμένο τμήμα της μνήμης (chip) της κάρτας γραφικών που ονομάζεται συνήθως frame buffer ή και video RAM

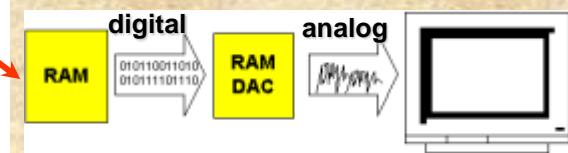
To video chip είναι ένας graphics processor που προσφέρει γραφικές λειτουργίες απευθείας σε h/w (δηλ. graphics accelerator)

Εκτός του frame buffer, αυτό το τμήμα της μνήμης της κάρτας γραφικών είναι στη διάθεση του προγραμματιστή (και ο frame buffer είναι αλλά...καλύτερα μην τον πειράζετε)

Accelerated Graphics Port (AGP) είναι η γέφυρα για την χρήση των GP functions μέσω της CPU



O frame buffer είναι το τμήμα μνήμης της κάρτας που ο RAMDAC, με τη συχνότητα της οθόνης (π.χ. 70 Hz), το αποτυπώνει στη οθόνη συνέχεια (π.χ. 70 φορές/sec).





Screen depth and resolution (2/10)

- Η ανάλυση της οθόνης σε pixels ρυθμίζεται από την κάρτα, η οποία υποστηρίζει ορισμένες εναλλακτικές αναλύσεις, όπως 640×480 , 800×600 και 1024×768 .
 - Αυτό σημαίνει ότι ο frame buffer είναι ουσιαστικά pixel array (ή αλλιώς pixel map) με μέγεθος που εξαρτάται από το εκάστοτε *resolution* (Width * Height)
- Το κάθε pixel έχει μία τιμή που ορίζει το χρώμα που θα ζωγραφιστεί στην αντίστοιχη θέση. Ο χώρος μνήμης **D** που καταλαμβάνει ένα pixel ονομάζεται και screen / pixel depth και είναι ο ίδιος για όλα τα pixels. Η κάρτα γραφικών μπορεί να υποστηρίζει εναλλακτικά **D**.
 - **D=1** σημαίνει *1 bit ανά pixel* που σημαίνει μόνο δύο χρώματα μπορούν να αποθηκευτούν (μαύρο = 0 και άσπρο = 1). Σε 1024×768 resolution απαιτούνται 658K για τον frame buffer στην video RAM. Το 1 bit mode είναι ξεπερασμένο, αλλά σε αυτό οφείλεται η ονομασία bitmap (Bit Map).
 - **D=8** σημαίνει *8 bits ανά pixel* που σημαίνει 256 χρώματα και σε 1024×768 5.6MB για τον frame buffer στην video RAM. Η χρήση screen depth 8 bits απαιτεί κάποιες επιπλέον επεξηγήσεις.



Screen depth and resolution (3/10)

- **8 bits ή palette mode.** Ο τρόπος που μία τιμή 8 bits αντιστοιχεί σε 256 χρώματα είναι ο εξής:
 - Υπάρχει ένας πίνακας 256 θέσεων, που ονομάζεται *palette*, με δομή παρόμοια με τον τύπο Palette που ορίζεται παρακάτω:

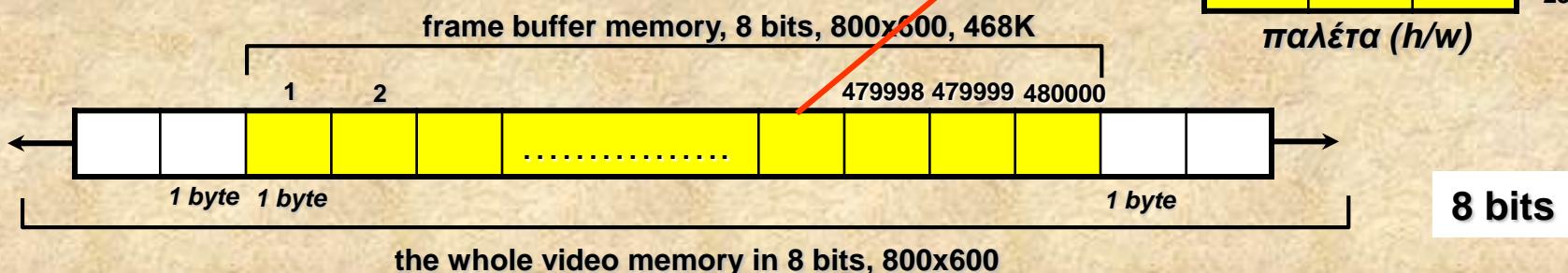
```
struct RGB { unsigned char r,g,b; };  
typedef RGB Palette[256];
```
 - Ένα instance αυτού του πίνακα έχει η κάρτα γραφικών και περιέχει τη λίστα των 256 χρωμάτων που μπορείτε να χρησιμοποιήσετε. Κάθε χρώμα ορίζεται με τη «μίξη» των RGB values, βάσει των οποίων το h/w ζωγραφίζει ένα pixel στην οθόνη.
 - Έτσι, ένα byte (8-bits pixel) είναι ουσιαστικά ένα index σε αυτή την παλέτα. Όταν η κάρτα ζωγραφίζει τον frame buffer στην οθόνη, κάνει τα εξής για κάθε pixel:
 - ◆ διαβάζει την τιμή του pixel byte *b*
 - ◆ το χρησιμοποιεί ως index στην παλέτα, *palette[b]*
 - ◆ λαμβάνει τα RGB values της θέσης αυτής: *palette[b].r*, *palette[b].g*, και *palette[b].b*
 - ◆ ζωγραφίζει στην αντίστοιχη θέση της οθόνης το pixel με χρώμα που προκύπτει από τη μίξη αυτών των τιμών στα βασικά χρώματα



Screen depth and resolution (4/10)

- Στα 8 bits μπορείτε να θέσετε εναλλακτικές παλέτες, ανάλογα με το ποια χρώματα (δηλ. ποιες αποχρώσεις) χρησιμοποιεί το παιχνίδι σας.
- Βάσει των κανόνων μίξης των χρωμάτων εύκολα βρίσκουμε ότι:
 - (R:0,G:0,B:0) είναι **μαύρο**, τιμής χρώματος 0
 - (R:255,G:255,B:255) είναι **άσπρο**, τιμής χρώματος 1
 - (R:255,G:0,B:0) είναι **κόκκινο**, τιμής χρώματος 252
 - (R:0,G:255,B:0) είναι **πράσινο**, τιμής χρώματος 253
 - (R:0,G:0,B:255) είναι **μπλε**, τιμής χρώματος 254
 - (R:255,G:255,B:0) είναι **κίτρινο**, τιμής χρώματος 255

R	G	B	
0	0	0	0
255	255	255	1
...	
255	0	0	252
0	255	0	253
0	0	255	254
255	255	0	255



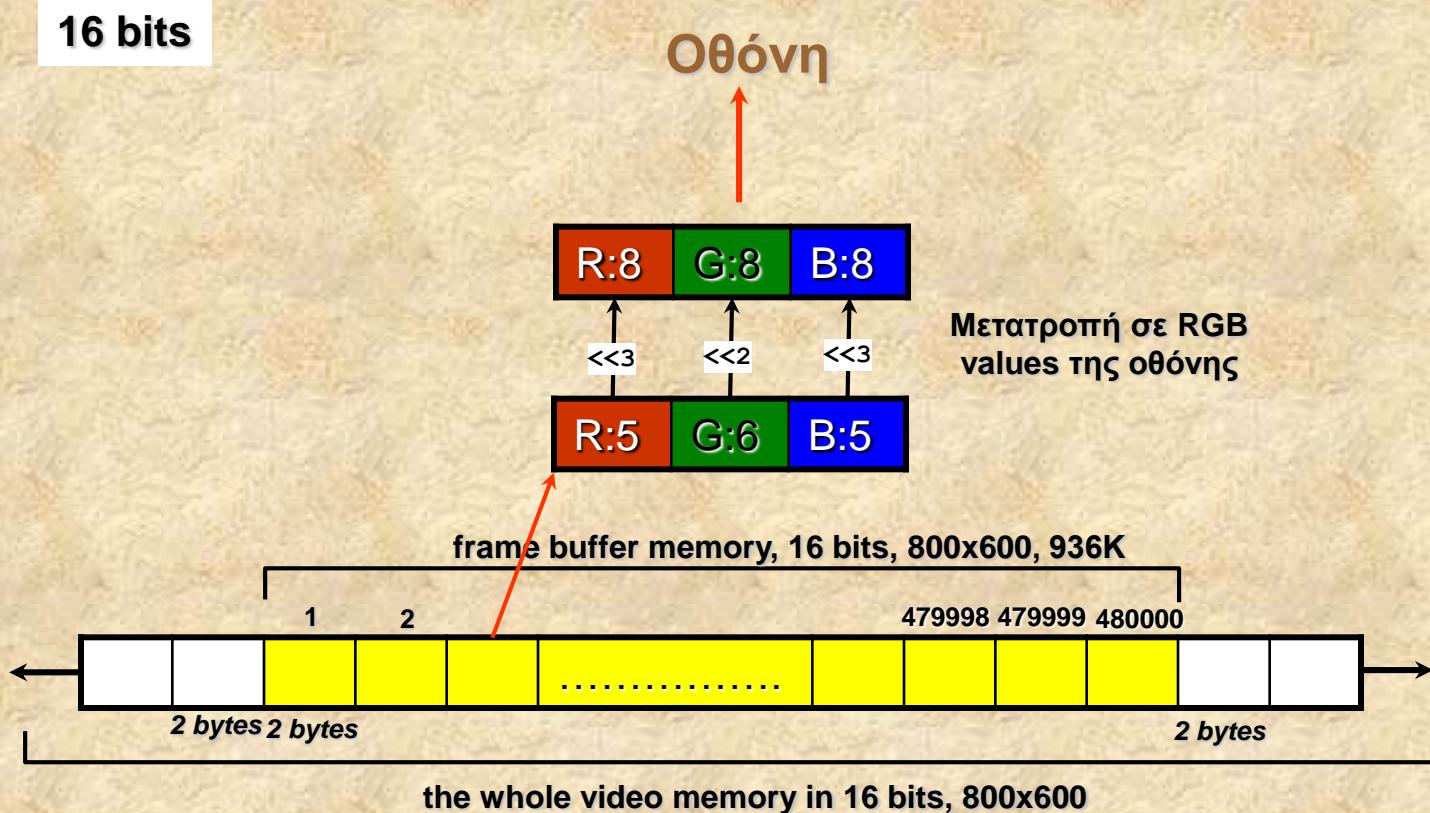


Screen depth and resolution (5/10)

- Πέραν των 8 bits ανά κάθε pixel, σε κανένα άλλο mode δεν υφίσταται παλέτα, αλλά τα ίδια τα RGB values είναι αποθηκευμένα στο χώρο μνήμης που λαμβάνει το κάθε pixel.
- Στα **16 bits** ή **high color mode**, τα χρώματα μοιράζονται συνήθως ως **R(5), G(6), B(5)**, ωστόσο η σειρά με την οποία βρίσκονται μέσα στα 16 bits του pixel ποικίλει (μερικές κάρτες σε αφήνουν να επιλέξεις τη διάταξη, ενώ σε άλλες είναι fixed).
 - Πρακτικά λοιπόν έχουμε 65536 χρώματα.
 - Καθώς οι τιμές των RGB σε 16 bits είναι μικρότερες του 255, το h/w κάνει τα εξής shifts πριν ζωγραφίσει ένα 16 bit pixel: **R<<3, G<<2, B<<3** διατηρώντας το LSb στα shift positions (π.χ. Το $1<<3$ γίνεται 1111 και όχι 1000)
 - Δηλ. δεν αντιπροσωπεύονται κάποια χρώματα χαμηλής έντασης. Έτσι, π.χ., στα 16 bits μερικά χρώματα είναι:
 - ◆ R:00000 G:000000 B:00000 **Μαύρο**, τιμή χρώματος 0
 - ◆ R:11111 G:111111 B:11111 **Ασπρό**, τιμή χρώματος σε **RGB** διάταξη **FFFF**
 - ◆ R:11111 G:000000 B:00000 **Κόκκινο**, τιμή χρώματος 0 σε **RBG** διάταξη **F800**
 - ◆ R:11111 G:111111 B:00000 **Κίτρινο**, τιμή χρώματος 0 σε **GBR** διάταξη **FC1F**



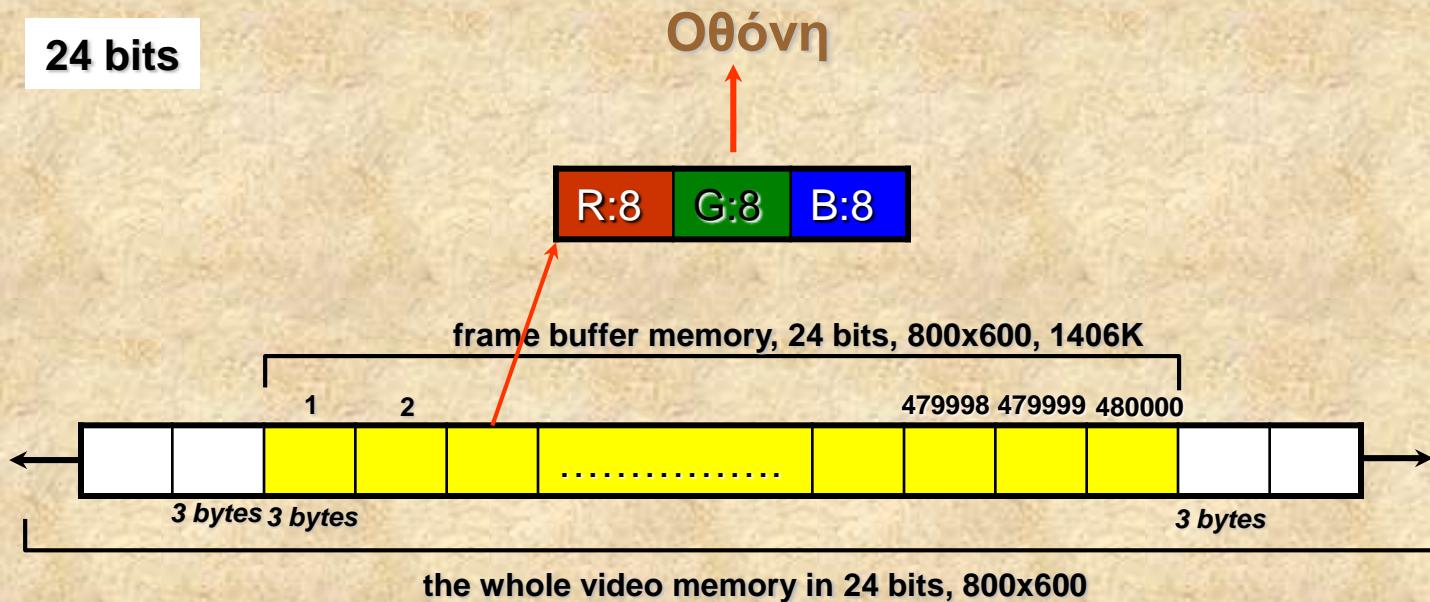
Screen depth and resolution (6/10)





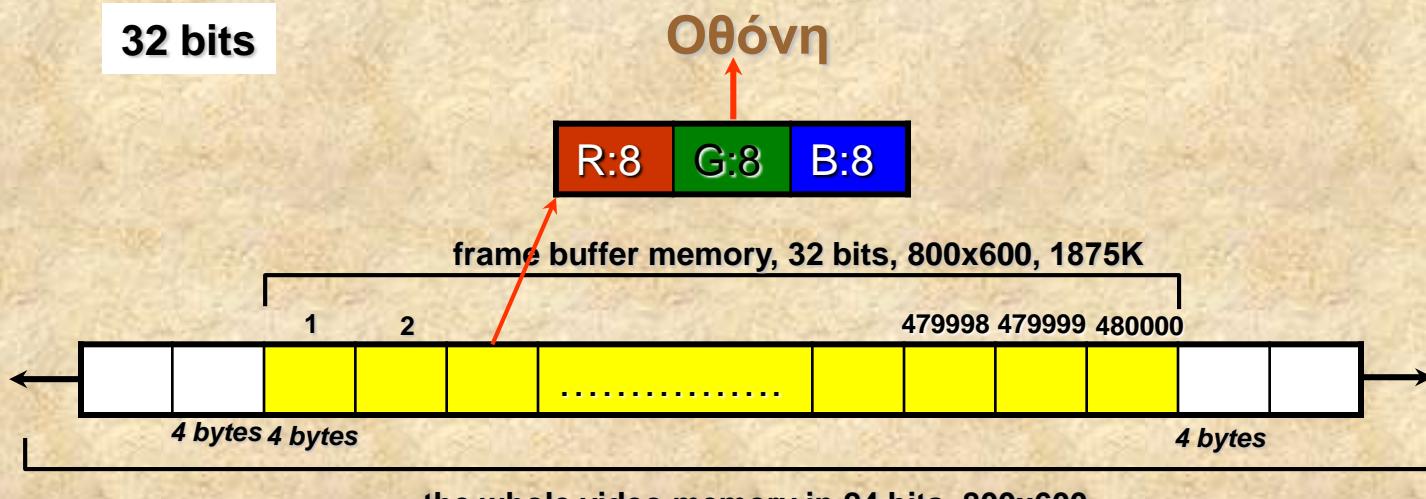
Screen depth and resolution (7/10)

- Στα **24 bits ή true color mode** κάθε pixel έχει 3 bytes, με ένα byte για κάθε βασικό χρώμα, δηλ. καλύπτει ακριβώς το εύρος του h/w, άρα ζωγραφίζεται ένα pixel με τις RGB τιμές που περιέχει όπως είναι.
- Οι τιμές των βασικών χρωμάτων RGB εμφανίζονται σε 3 bytes η σειρά των οποίων εξαρτάται από την κάρτα γραφικών.



Screen depth and resolution (8/10)

- Στα **32 bits** κάθε pixel έχει 4 bytes, εκ των οποίων τα τρία είναι reserved για τα βασικά χρώματα, με τρόπο ανάλογο των 24 bits, και το ένα επιπλέον byte έχει το όνομα alpha channel ή alpha blending value.
 - Επειδή ο ρόλος του alpha είναι πολύ ειδικός σε συγκεκριμένες γραφικές εντολές, θα εξηγηθεί αργότερα όταν πρώτα δούμε τις εντολές αυτές.
 - Το alpha value μπορεί να αγνοηθεί εάν ποτέ δεν χρειάζεστε τις ευκολίες που προσφέρει. Άλλα τότε γιατί κανείς να θέλει να δουλεύει σε 32 bits? Γιατί όπως θα δούμε αργότερα είναι ταχύτερη η επεξεργασία σε σύγκριση με 24 bits, αλλά καταναλώνει περισσότερη μνήμη.





Screen depth and resolution (9/10)

■ Συνηθισμένα APIs (1/2)

- Για την διαχείριση, pixel depth και του resolution συνήθως προσφέρονται κατάλληλες συναρτήσεις,
- ενώ μόνο σε higher-level libraries συναντώνται συναρτήσεις για διαχείριση των χρωμάτων με τη βοήθεια ενός γενικού τύπου χρώματος, συνήθως του μεγαλύτερου, π.χ, 32 bits

```
typedef unsigned short Dim;
struct Rect { int x, y, w, h; };
struct Point { int x, y; };
enum BitDepth { bits8 = 1, bits16, bits24, bits32 };

bool Open (Dim rw, Dim rh, BitDepth depth);
void Close (void);
Dim GetResWidth (void);
Dim GetResHeight (void);
BitDepth GetDepth (void);
```



Screen depth and resolution (10/10)

■ Συνηθισμένα APIs (2/2)

- Δεν είναι πολύ συνηθισμένο τα system libraries για άμεση χρήση της κάρτας γραφικών και της οθόνης (direct graphics media access) να σας δίνουν έτοιμες τις συναρτήσεις για colors.

```
typedef unsigned int Color;
typedef unsigned char RGBValue;
typedef unsigned char Alpha;
struct RGB { RGBValue r, g, b; };
struct RGBA : public RGB { RGBValue a; };
typedef RGB Palette[256];
void SetPalette (RGB* palette);

Color Make8  (RGBValue r, RGBValue g, RGBValue b);
Color Make16 (RGBValue r, RGBValue g, RGBValue b);
Color Make24 (RGBValue r, RGBValue g, RGBValue b);
Color Make32 (RGBValue r, RGBValue g, RGBValue b, Alpha alpha = 0);
```

Εντοπίστε τις αντίστοιχες συναρτήσεις και τύπους στη βιβλιοθήκη Allegro / SDL

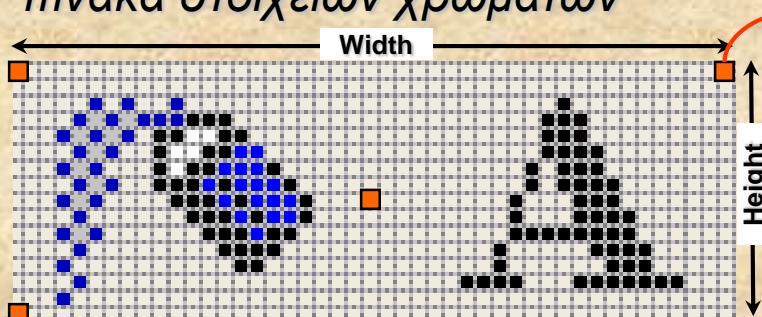


Περιεχόμενα

- Screen depth and resolution
- ***Bitmaps and bit block transfer***
- Direct bitmap / video access
- Color and key masking
- Bounding boxes and collision detection
- Display frequency and vertical retrace synchronization
- Double buffering, page flipping, triple buffering
- Pixel-level primitives and operations

Bitmaps and bit block transfer (1/6)

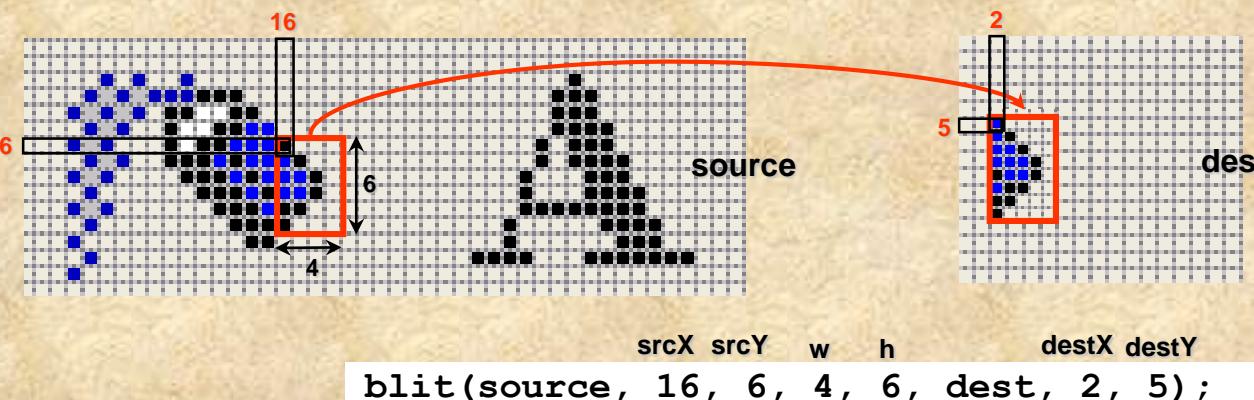
- Ένα bitmap είναι μία τετράδα $\langle W, H, B, D \rangle$ με τις εξής ιδιότητες:
 - $D \in \{1, 2, 3, 4\}$ και ονομάζεται *bitmap depth*
 - B είναι συνεχόμενη μνήμη από $W \times H \times D$ bytes και ονομάζεται *pixel buffer*
 - ο συμβολισμός $\text{Pixel}(x, y)$: $x \in [0, W]$, $y \in [0, H]$, αναφέρεται στα D bytes που αρχίζουν από την διεύθυνση $B[(y^*W+x)^*D]$ και λέγεται *pixel storage*
 - ➔ αυτή η δομή είναι ένας δισδιάστατος πίνακας από *pixels*, πρόσβαση στον οποίο γίνεται σωστά μόνο με γνώση των W, H, B, D
 - ➔ ένα *bitmap* χρησιμοποιείται για την αποθήκευση ζωγραφιών με τη μορφή *πίνακα στοιχείων χρωμάτων*



Pixel storage με συγκεκριμένο χρώμα στο format (bit depth) που αντιστοιχεί σε όλο το bitmap
(Π.χ. 4 bytes, δηλ. D είναι 4, δηλ. 32 bits)

Bitmaps and bit block transfer (2/6)

- Ο τύπος *bitmap* υποστηρίζεται με διάφορες παραλλαγές σε όλες τις βιβλιοθήκες για άμεση πρόσβαση στην οθόνη και στη μνήμη της κάρτας γραφικών
- Ένα bitmap μπορεί να αποτυπωθεί στην οθόνη. Αυτό στην πράξη σημαίνει μεταφορά (copy) bits από τη μνήμη του bitmap στην μνήμη της οθόνης (frame buffer). Αυτή είναι μία χαρακτηριστική λειτουργία που λέγεται bit block transfer – *blit*
- Η ίδια λειτουργία υφίσταται και μεταξύ bitmaps.
- Επιπλέον, υποστηρίζονται παράμετροι που ορίζουν το ορθογώνιο τμήμα του πηγαίου bitmap που πρόκειται να γίνει copy, και τη θέση στο bitmap προορισμού όπου πρόκειται να γίνει η επικόλληση.



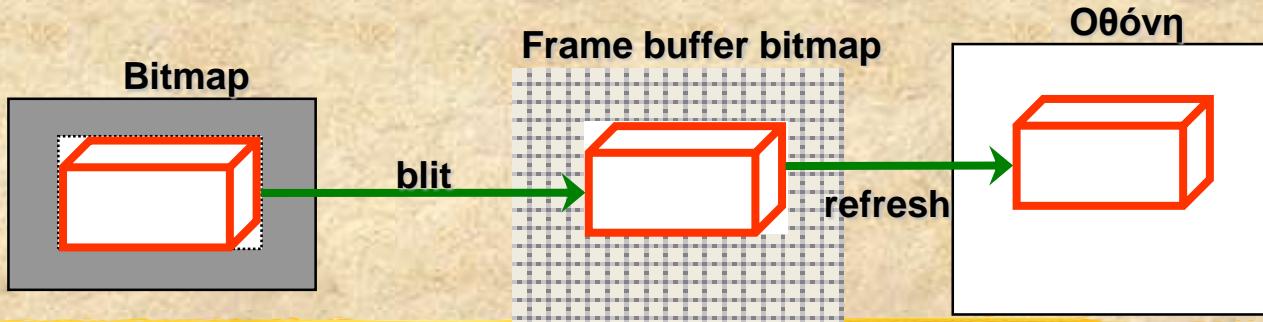


Bitmaps and bit block transfer (3/6)

- Για να πετύχει ένα blit πρέπει οπωσδήποτε τα δύο bitmaps να έχουν το **ίδιο bit depth**. Θα μπορούσατε όμως να υλοποιήσετε cross-depth blit με conversion (ακριβό και μάλλον άσκοπο)
- Πέρα από τα bitmaps που υπάρχουν στο program (main) memory, **μπορούν να δημιουργηθούν video bitmaps** που χρησιμοποιούν τη διαθέσιμη video memory (αυτή που περισσεύει εάν εξαιρέσουμε το frame buffer).
 - Επιτρέπονται blits μεταξύ bitmaps που βρίσκεται στην video memory και αυτών που βρίσκονται στη main memory. Γενικά στις περισσότερες κάρτες επιτρέπονται όλων των ειδών οι συνδυασμοί για blits.
 - ◆ Συνήθως τα blits μεταξύ των video bitmaps είναι τα ταχύτερα.
 - ◆ Άλλα σήμερα οι μνήμες είναι τόσο γρήγορες που καλά είναι να το ελέγξετε
 - ◆ Λόγω του περιορισμένου χώρου στην video RAM δεν μπορούμε να έχουμε όλα μας τα bitmaps στην video RAM
- Επίσης ο ίδιος **o frame buffer μπορεί να γίνει referenced** μέσω μίας built-in μεταβλητής τύπου bitmap, με width, height και depth αυτά της οθόνης.
 - Άρα για να εμφανίσουμε ένα bitmap στην οθόνη αρκεί ένα κατάλληλο blit με target bitmap το bitmap instance reference της οθόνης (του frame buffer)

Bitmaps and bit block transfer (4/6)

- Η ταχύτητα ενός blit είναι αντιστρόφως ανάλογη του συνολικού μεγέθους του bitmap σε bytes (δηλαδή του pixel buffer)
- Έτσι δύο ίδιου μεγέθους bitmaps σε διαφορετικό bit depth γίνονται blit ως εξής:
 - έστω N msec στα 8 bits,
 - τότε θεωρητικά $2xN$ στα 16 bits, $3xN$ στα 24 bits, $4xN$ στα 32 bits, καθώς θα έχουν διπλάσιο, τριπλάσιο και τετραπλάσιο μέγεθος αντίστοιχα
 - ◆ το θεωρητικό δεν είναι πάντα σωστό στην πράξη (εξαρτάται από τον επεξεργαστή και την κάρτα, το data bus και τα mov instructions)
- Γι' αυτό το λόγο δεν επιλέγουμε πάντα τα 32 bits ή ακόμη και τα 24 bits.
 - Επιπλέον, όσο μεγαλύτερο το bit depth (αλλά και το resolution), τόσο περισσότερη μνήμη καταλαμβάνει και ο frame buffer, ενώ μικρότερο αριθμό από video bitmaps μπορούμε να φτιάξουμε (υποπολλαπλάσιο!).





Bitmaps and bit block transfer (5/6)

- Τέτοιου είδους block transfer μεταξύ του main memory και video RAM συνήθως υποστηρίζεται από το h/w (είναι accelerated)
- Καθώς και η οθόνη είναι ένα bitmap, όσο μεγαλύτερο bit depth και resolution έχουμε, τόσο περισσότερο χρόνο θα χρειάζεται το παιχνίδι για να κάνει κάθε φορά refresh τη σκηνή του παιχνιδιού.
- Στα 32 bits, η τιμή για *alpha blending* παίζει τον εξής ρόλο στην περίπτωση που έχουμε blit από src σε dest:
 - for each source pixel A και αντίστοιχο destination pixel B
 - ◆ If **alpha channel value of A = 0** (αδιαφανές - opaque)
 - destination pixel is overwritten with value of A
 - ◆ If **alpha channel value A = 255** (διαφανές - transparent)
 - destination pixel retains its value B
 - ◆ **Otherwise**
 - destination pixel B gets $B + (A - B) * (1 - \text{alpha} / 255)$



Bitmaps and bit block transfer (6/6)

■ Συνηθισμένα APIs

- Για τη δημιουργία bitmaps προφέρονται software libraries τα οποία συνήθως παρέχουν τα παρακάτω:
 - ◆ Τύπο bitmap που συνήθως είναι opaque ως προς την εσωτερική δομή αποθήκευσης των pixels (pixel buffer), blit functions και κάποια accessors

```
typedef void* Bitmap;
Bitmap  BitmapLoad (const std::string& path);
Bitmap  BitmapCreate (Dim w, Dim h);
Bitmap  BitmapCopy (Bitmap bmp);
Bitmap  BitmapClear (Bitmap bmp, Color c);
void    BitmapDestroy (Bitmap bmp);
Bitmap  BitmapGetScreen (void);
Dim     BitmapGetWidth (Bitmap bmp);
Dim     BitmapGetHeight (Bitmap bmp);
void    BitmapBlit (
                Bitmap src, const Rect& from,
                Bitmap dest, const Point& to
            );
```

Εντοπίστε τις αντίστοιχες συναρτήσεις και τύπους στη βιβλιοθήκη Allegro / SDL



Περιεχόμενα

- Screen depth and resolution
- Bitmaps and bit block transfer
- ***Direct bitmap / video access***
- Color key and masking
- Bounding boxes and collision detection
- Display frequency and vertical retrace synchronization
- Double buffering, page flipping, triple buffering
- Pixel-level primitives and operations



Direct bitmap / video access (1/9)

- Προφανώς πέρα από blit operations πρέπει να προσφέρεται και ένας τρόπος πρόσβασης σε pixel storage επίπεδο, ώστε να μπορούμε να δημιουργούμε το περιεχόμενο των bitmaps
- Είναι επιθυμητό ο τρόπος αυτός να είναι κοινός για όλων των ειδών τα bitmaps, είτε είναι στη main memory είτε πρόκειται για video bitmaps
- Είναι απαραίτητο ο τρόπος αυτός να αποδίδει τη μεγαλύτερη δυνατή ταχύτητα πρόσβασης
- Θα δούμε ότι βασιζόμαστε στο μοντέλο που δώσαμε για τη δομή του bitmap, ειδικότερα τον pixel buffer, αλλά με κάποιες μικρές τροποποιήσεις



Direct bitmap / video access (2/9)

■ Το μοντέλο (1/2)

- **Κλείδωμα** (lock / request / grant) πριν και **απελευθέρωση** (unlock / release / free) μετά για την πρόσβαση
 - ◆ Συνεπώς επιβάλλεται η αποκλειστική χρήση ενός bitmap μεταξύ δύο τέτοιων calls για άμεση πρόσβαση στη μνήμη
- **Πρόσβαση ανά γραμμή** από pixels, όπου η απόσταση στη μνήμη μεταξύ δύο συνεχόμενων γραμμών να μην θεωρείται πάντα 0,
 - ◆ αλλά είναι μία τιμή που εν γένει διαφέρει ανά bitmap, και επιστρέφεται μετά το κλείδωμα
- **Πρόσβαση ανά pixel** βάσει του bit depth, αλλά κυρίως των κανόνων αποθήκευσης των χρωμάτων στα συνεχόμενα bytes που καταλαμβάνει ένα color value
 - ◆ επειδή η τοποθέτηση των RGB values στα bytes αυτά εξαρτάται από την κάρτα, το system library θα σας προσφέρει συναρτήσεις για την εξαγωγή της εκάστοτε δομής



Direct bitmap / video access (3/9)

Το μοντέλο (2/2)

```
using BitmapAccessFunctor = std::function<void(PixelMemory*)>;  
  
void BitmapAccessPixels (Bitmap bmp, const BitmapAccessFunctor & f) {  
    auto result = BitmapLock(bmp);  
    assert(result);  
    auto mem = BitmapGetMemory(bmp);  
    auto offset = BitmapGetLineOffset(bmp);  
  
    for (auto y = BitmapGetHeight(bmp); y--;) {  
        auto buff = mem;  
        for (auto x = BitmapGetWidth(bmp); x--;) {  
            f(buff);  
            buff += GetDepth();  
        }  
        mem += offset;  
    }  
  
    BitmapUnlock(bmp);  
}
```



Direct bitmap / video access (4/9)

■ Συνηθισμένα APIs (1/3)

```
typedef unsigned char* PixelMemory;
bool           BitmapLock (Bitmap);
void           BitmapUnlock (Bitmap);
PixelMemory   BitmapGetMemory (Bitmap);
int            BitmapGetLineOffset (Bitmap);

void           WritePixelColor8  (PixelMemory, RGBValue);
void           WritePixelColor16 (PixelMemory, const RGB&);
void           WritePixelColor24 (PixelMemory, const RGB&);
void           WritePixelColor32 (PixelMemory, const RGB&, Alpha a);

void           ReadPixelColor8  (PixelMemory, RGBValue* );
void           ReadPixelColor16 (PixelMemory, RGB* );
void           ReadPixelColor24 (PixelMemory, RGB* );
void           ReadPixelColor32 (PixelMemory, RGB*, Alpha* );
```

Εντοπίστε τις αντίστοιχες συναρτήσεις και τύπους στη βιβλιοθήκη Allegro / SDL



Direct bitmap / video access (5/9)

■ Example – inverting and tinting pixels

```
#define INVERTED_BYTE(b)      (255 - (b))
#define TINTED_BYTE(b, f)      ((b) * (f))

void InvertPixelColor32 (PixelMemory pixel) {
    // assume RGBA order, low to high byte, alpha retained!
    pixel[0] = INVERTED_BYTE(pixel[0]);
    pixel[1] = INVERTED_BYTE(pixel[1]);
    pixel[2] = INVERTED_BYTE(pixel[2]);
}

void TintPixelColor32 (PixelMemory pixel, float f) {
    pixel[0] = TINTED_BYTE(pixel[0], f);
    pixel[1] = TINTED_BYTE(pixel[1], f);
    pixel[2] = TINTED_BYTE(pixel[2], f);
}

void BitmapInvertPixels32 (Bitmap bmp)
    { BitmapAccessPixels(bmp, &InvertPixelColor32); }

void BitmapTintPixels32 (Bitmap bmp, float f) {
    BitmapAccessPixels(
        bmp,
        [f](PixelMemory pixel) { TintPixelColor32(pixel, f); }
    );
}
```



Direct bitmap / video access (6/9)

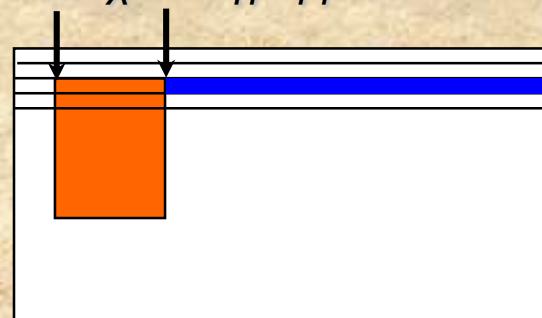
Sample Length:	8	8	8	8																												
Channel Membership:	Red Green Blue Alpha																															
Bit Number:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

```
unsigned GetRedShiftRGBA (void);
unsigned GetRedBitMaskRGBA (void);
unsigned GetGreenShiftRGBA (void);
unsigned GetGreenBitMaskRGBA (void);
unsigned GetBlueShiftRGBA (void);
unsigned GetBlueBitMaskRGBA (void);
unsigned GetAlphaShiftRGBA (void);
unsigned GetAlphaBitMaskRGBA (void);

// firstly mask to isolate the RGB component, then shift to get value
RGBValue GetRedRGBA (PixelMemory pixel) {
    Color c = *((Color*) pixel);
    return (c & GetRedBitMaskRGBA()) >> GetRedShiftRGBA();
}
```

Direct bitmap / video access (7/9)

- Γιατί εμφανίζεται αυτή η τιμή απόστασης μεταξύ δύο διαδοχικών γραμμών ενός bitmap και τι εξυπηρετεί?
 - Αυτή η απόσταση εμφανίζεται λόγω των ειδικών τρόπων αποθήκευσης των video bitmaps στην video RAM
 - ◆ μπορεί να χρησιμοποιούνται επιπλέον bytes (padding)
 - ◆ οι σειρές μπορεί να αποθηκεύονται ανάποδα (από κάτω προς τα πάνω)
 - Μερικά libraries προτιμούν να παρουσιάζουν όλη τη video RAM σαν ένα μεγάλο bitmap πάνω στο οποίο δημιουργείτε sub-bitmaps
 - ◆ *Τότε, στην περίπτωση των sub-bitmaps, είναι αναπόφευκτη η απόσταση στη μνήμη μεταξύ δύο διαδοχικών γραμμών*



A sub-bitmap
within a bitmap
(orange) and the
stride per sub-
bitmap line (blue)



Direct bitmap / video access (8/9)

■ Συνηθισμένα APIs (2/3)

- Παρακάτω δίνεται η *PutPixel* για συγκεκριμένο bit depth. Οι αντίστοιχες για διαφορετικά bit depth δεν διαφέρουν πολύ.
- Μπορείτε να βάλετε όλες αυτές σε έναν πίνακα και να καλείτε την κατάλληλη με το αντίστοιχο bit depth,
- αλλά υπάρχει και πιο γρήγορη λύση...

```
void PutPixel8 (Bitmap b, Dim x, Dim y, Color c) {  
    if (!BitmapLock(b))  
        assert(false);  
    WritePixelColor8(  
        BitmapGetMemory(b) +  
        y * (BitmapGetWidth(b) + BitmapGetLineOffset(b)) + x,  
        (RGBValue) c  
    );  
    BitmapUnlock(b);  
}
```



Direct bitmap / video access (9/9)

■ Συνηθισμένα APIs (3/3)

- Η παρακάτω τακτική πρέπει να ακολουθείται σε κάθε περίπτωση που έχουμε διαφορετικές συναρτήσεις ανά bit-depth - είναι γρηγορότερη από έναν κλασικό dispatch table, ουσιαστικά είναι υλοποίηση virtual table.

```
// function table approach
static void PutPixel16 (Bitmap b, Dim x, Dim y, Color c);
static void PutPixel24 (Bitmap b, Dim x, Dim y, Color c);
static void PutPixel32 (Bitmap b, Dim x, Dim y, Color c);
typedef void (*PutPixelFunc)(Bitmap b, Dim x, Dim y, Color c);
static PutPixelFunc putPixelFuncs[] = {
    PutPixel8, PutPixel16, PutPixel24, PutPixel32
};
static PutPixelFunc currPutPixel;
extern void PutPixel (Bitmap b, Dim x, Dim y, Color c)
{ (*currPutPixel)(b, x, y, c); }

extern void InstallPutPixel (void) // upon initialisation
{ currPutPixel = putPixelFuncs[GetDepth()-1]; }
```