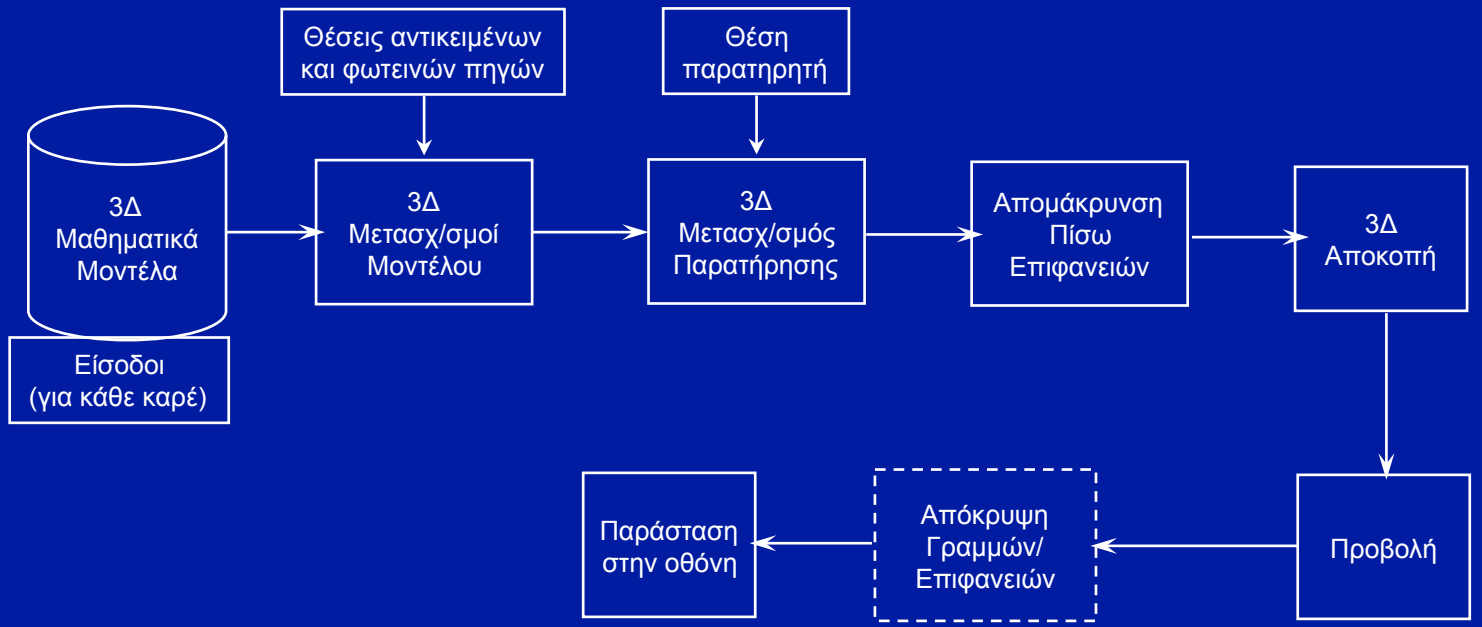
Αλγόριθμοι Παράστασης Βασικών Σχημάτων

Προσέγγιση μαθηματικών σχημάτων από διακριτά pixels:

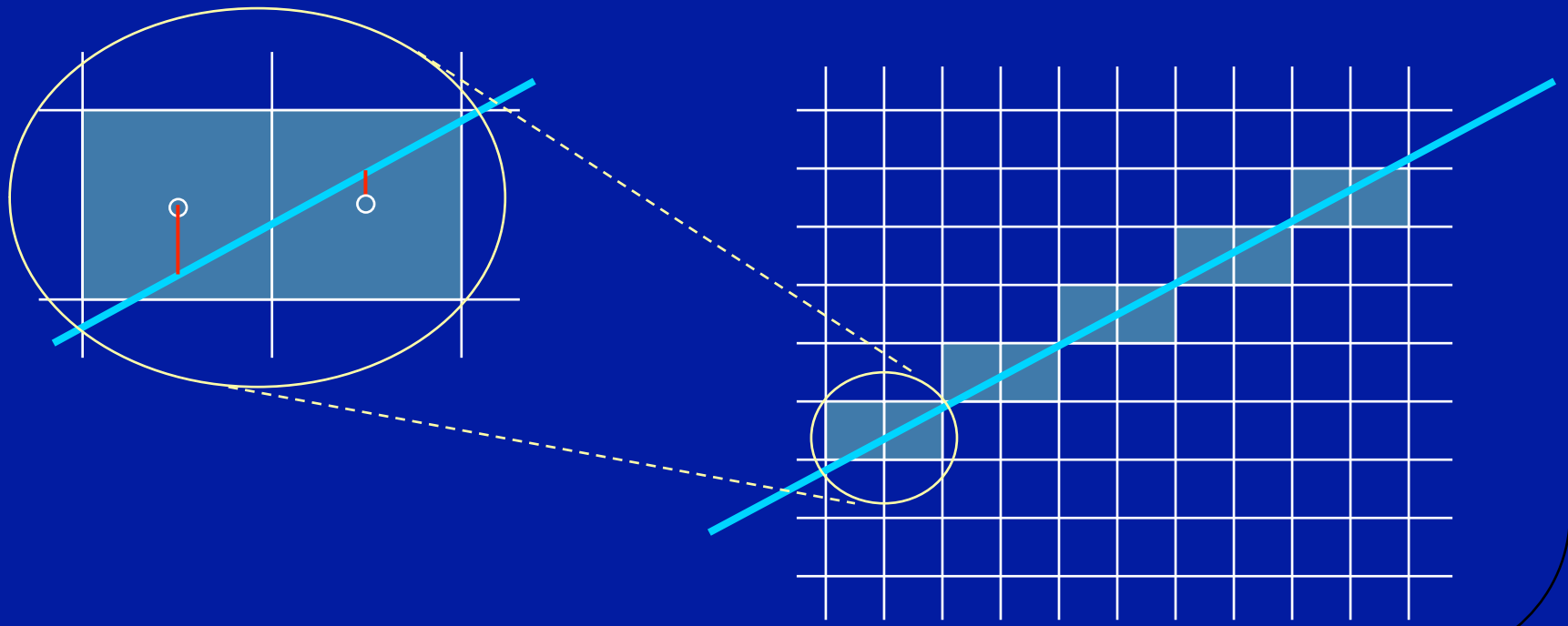
- Ευθύγραμμο τμήμα, κύκλος, κωνικές τομές, πολύγωνο.
- S/W ή H/W.



Ευθύγραμμο Τμήμα: Βασικός Αλγόριθμος

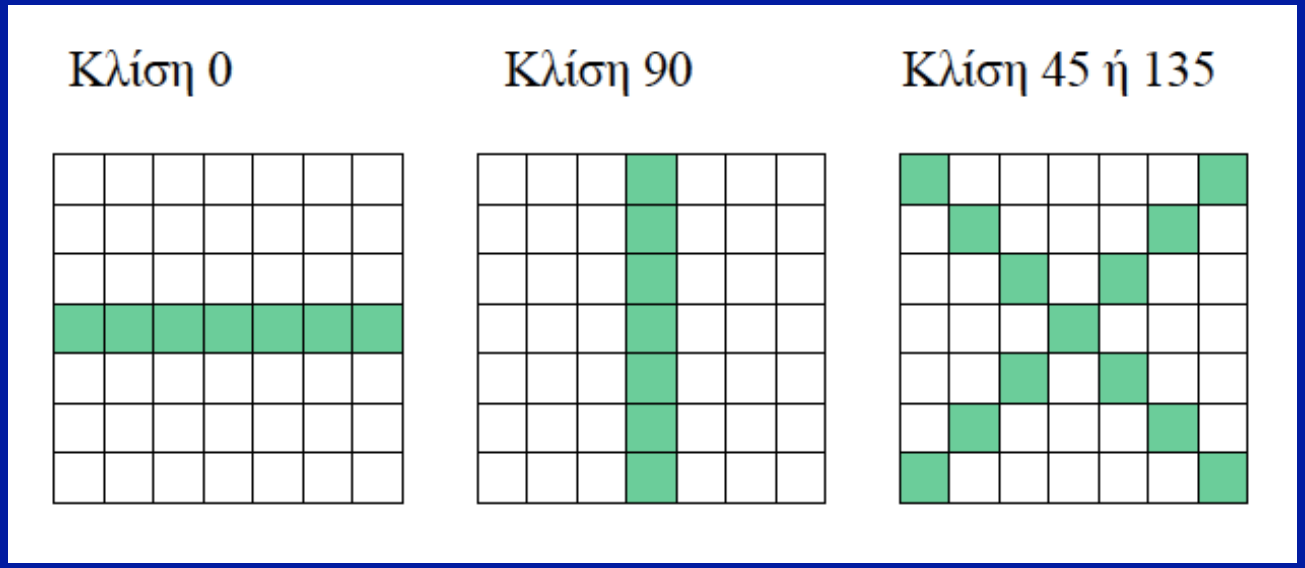
Κριτήρια καλού αλγόριθμου ευθύγραμμου τμήματος:

- Σταθερό πάχος ανεξάρτητο κλίσης, όχι κενά (συνεκτική)
- Pixels όσο το δυνατόν πλησιέστερα στη μαθηματική πορεία της
- Ταχύτητα



Ευθύγραμμο Τμήμα: Βασικός Αλγόριθμος

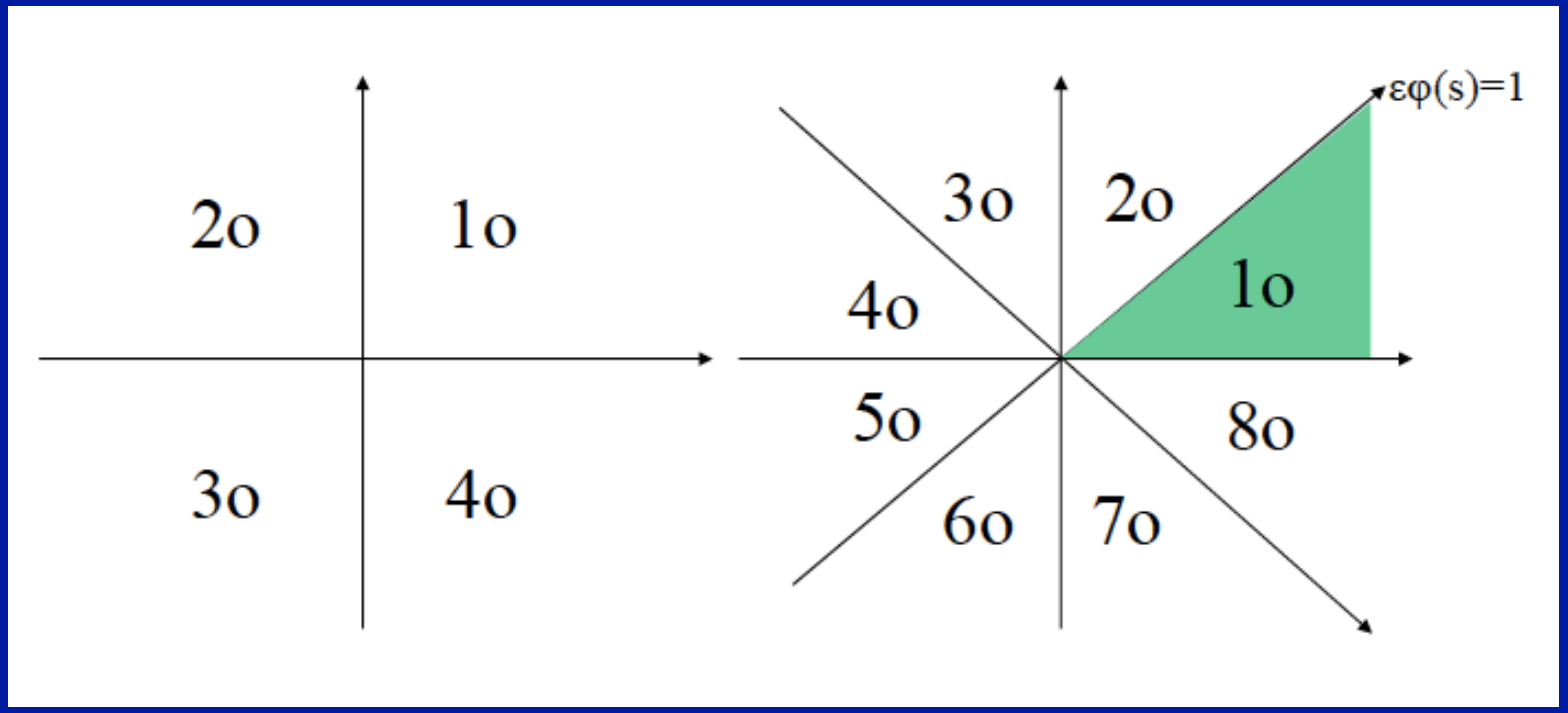
Πάχος 1 pixel



Ευθύγραμμο Τμήμα: Βασικός Αλγόριθμος

Τεταρτημόρια

Οκταμόρια



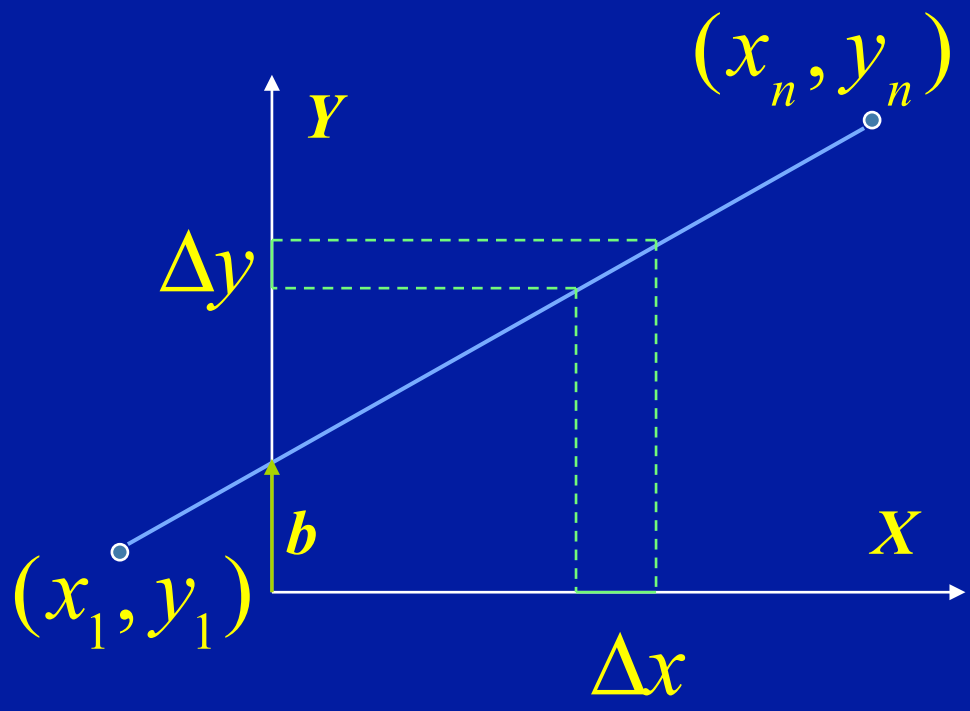
Ευθύγραμμο Τμήμα: Βασικός Αλγόριθμος

Έστω ευθύγραμμο τμήμα μεταξύ P1 (x1, y1) και Pn (xn, yn) 1ου οκταμορίου. Για κάθε σημείο P(x, y) του ευθύγραμμου τμήματος ισχύει::

$$y = s \cdot x + b$$

$$s = \frac{y_n - y_1}{x_n - x_1} = \frac{\Delta y}{\Delta x}$$

$$b = \frac{y_1 x_n - y_n x_1}{x_n - x_1}$$



Ευθύγραμμο Τμήμα: Βασικός Αλγόριθμος

Τα βήματα είναι:

- Υπολόγισε την κλίση s και την μετατόπιση b.
- Αρχικοποίηση των τιμών (πεδία διακύμανσης του x, βήμα του x).
- Υπολόγισε το πρώτο y για το πρώτο x.
- Υπολόγισε το δεύτερο y για το δεύτερο x.
- .....
- Υπολόγισε το τελευταίο y για το τελευταίο x.
- Εκτύπωσε τα αποτελέσματα.

Επαναληπτικός βρόγχος

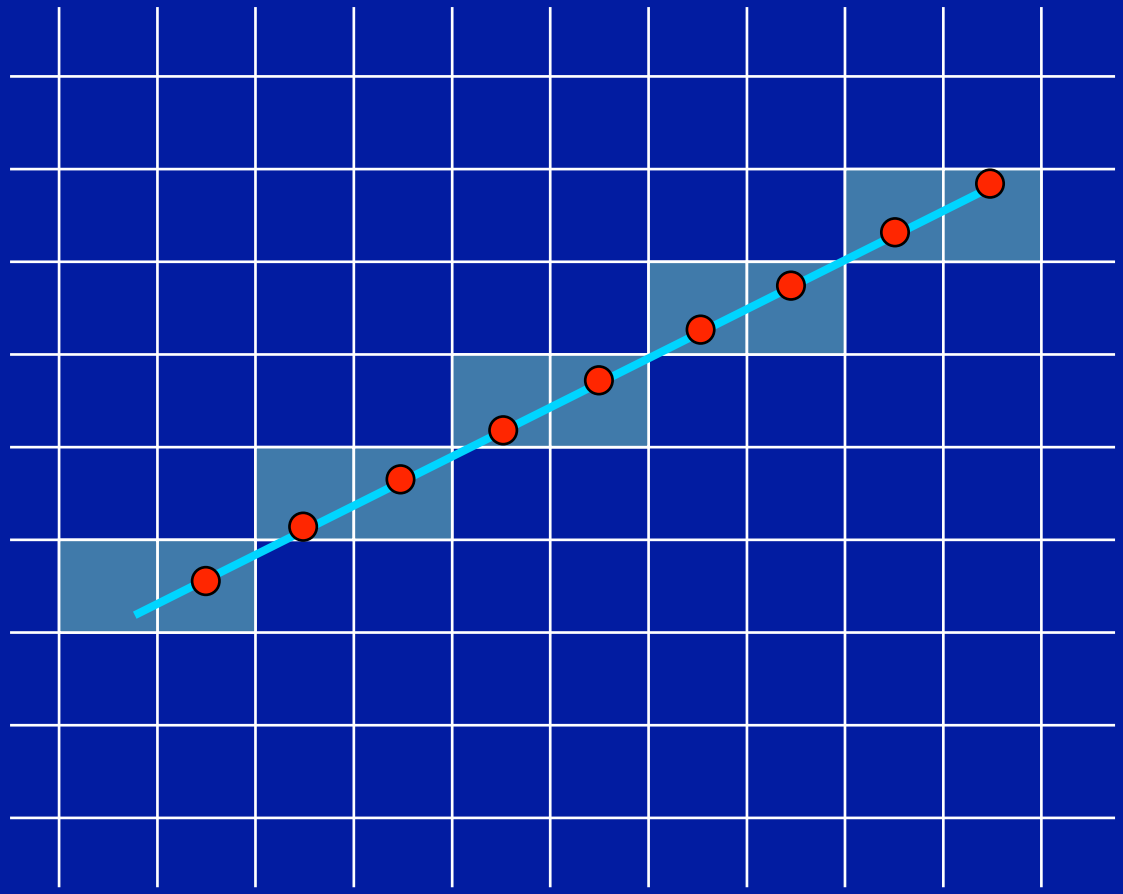
Ευθύγραμμο Τμήμα: Βασικός Αλγόριθμος

Ο αντίστοιχος βρόχος σχεδίασης των pixels της ευθείας είναι::

```
void line1(int x1, int y1, int xn,int yn)
{
    float s,b,y;
    int x;
    s=(yn-y1) / (xn-x1);
    b=(y1*xn - yn*x1) / (xn-x1);
    for (x=x1; x<=xn; x++)
    {
        y=s*x+b;
        setPixel(x, round(y));
    }
}
```

Ευθύγραμμο Τμήμα: Βασικός Αλγόριθμος

Οι τιμές του Y άξονα στρογγυλεύονται στο πλησιέστερο pixel



Ευθύγραμμο Τμήμα: Βελτιωμένος Αλγόριθμος

Ο πολλαπλασιασμός μέσα στο βρόγχο μπορεί να απλοποιηθεί αφού ισχύει:

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = sx_{i+1} + b = sx_i + b + s = y_i + s$$

Οι παραπάνω σχέσεις ισχύουν για το 1ο οκταμόριο

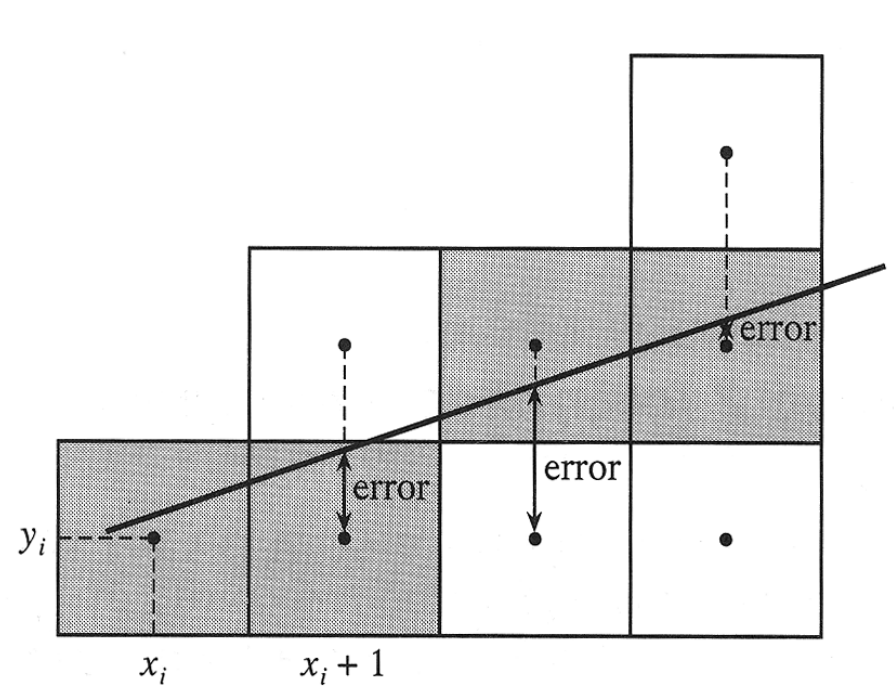
Ευθύγραμμο Τμήμα: Βελτιωμένος Αλγόριθμος

Ο αντίστοιχος βρόγχος σχεδίασης των pixels της ευθείας είναι::

```
void line2(int x1, int y1, int xn,int yn)
{
    float s,y;
    int x;
    s=(yn-y1) / (xn-x1);
    y=y1;
    for (x=x1; x<=xn; x++)
    {
        setPixel(x, round(y));
        y=y+s;
    }
}
```

Ευθύγραμμο Τμήμα: Κατάργηση στρογγύλευσης (round)

- Error είναι η απόσταση του pixel (xi+1,yi) από ιδεατή ευθεία
- Η στρογγύλευση μπορεί να αντικατασταθεί από σταδιακή επαύξηση του σφάλματος και σύγκριση:
- Αν η νέα, επαυξημένη τιμή του Y ξεπερνά το εύρος του pixel, τότε η σχεδίαση γίνεται στο επόμενο κατά Y pixel



Ευθύγραμμο Τμήμα: Κατάργηση στρογγύλευσης (round)

```
void line3(int x1, int y1, int xn, int yn)
{   float s,error;
    int x,y;
    s=(yn-y1) / (xn-x1);
    y=y1;
    error=0;
    for (x=x1; x<=xn; x++)
    {   setPixel(x, y);
        error=error+s;
        if (error>=0.5)
        { y++; error-- }
    }
}
```

Ευθύγραμμο Τμήμα – Χρήση Μόνο Ακεραίων (Αλγόριθμος Bresenham)

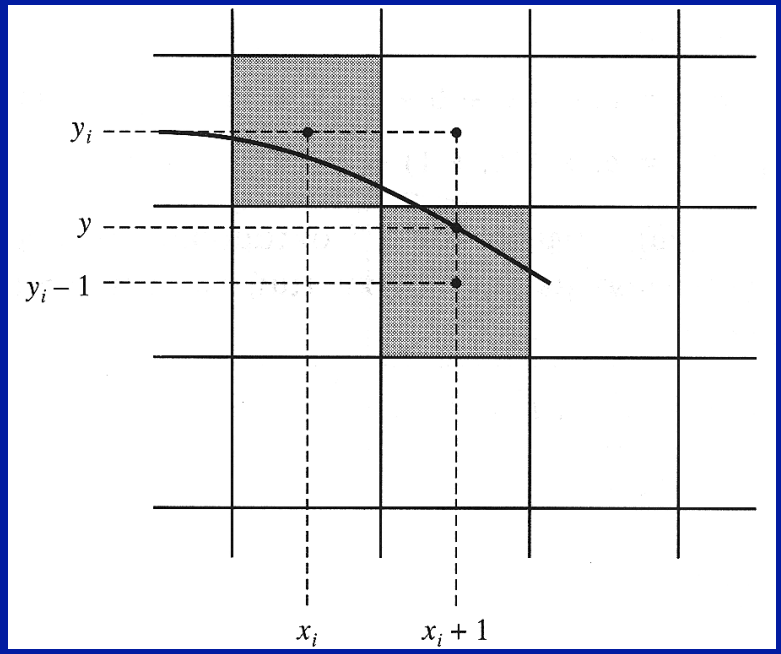
- Αντικατάσταση πραγματικών μεταβλητών από ακέραιες με κατάλληλη κλιμάκωση s, error & συνθήκης επιλογής
- Πολλαπλασιάζουμε με dx = xn-x1 $\rightarrow$ s = dy
- Πολλαπλασιάζουμε με dx $\rightarrow$ Συνθήκη error $\geq$ dx/2 $\Leftrightarrow$ error $\geq \lfloor dx/2 \rfloor \Leftrightarrow$ error ≥ 0 & αρχική αφαίρεση $\lfloor dx/2 \rfloor$ από error
- Πολλαπλασιάζουμε με dx $\rightarrow$ error=error-dx

Ευθύγραμμο Τμήμα – Χρήση Μόνο Ακεραίων (Αλγόριθμος Bresenham)

```
void bresenham(int x1, int y1, int xn, int yn)
{
    int error,x,y,dx,dy;
    dx=xn-x1;
    dy=yn-y1;
    error=-dx/2; y=y1;
    for (x=x1; x<=xn; x++)
    {
        setPixel(x,y);
        error=error+dy;
        if (error>=0)
        { y++; error=error-dx }
    }
}
```

Αλγόριθμος Bresenham για Κύκλο (2ο οκταμόριο)

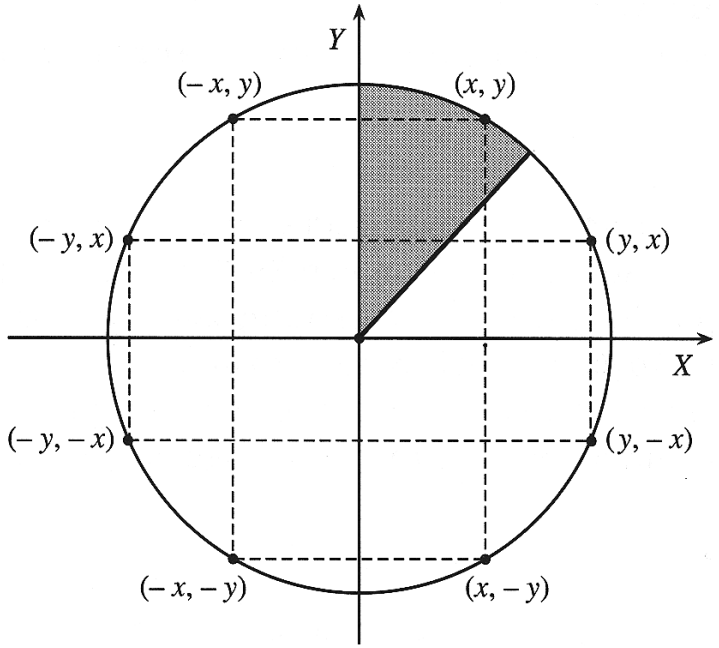
- Έστω ότι βρισκόμαστε στο pixel (xi,yi)
- Το επόμενο pixel θα είναι είτε το (xi+1,yi) είτε το (xi+1,yi-1) για το 2ο οκταμόριο (δεξιόστροφη πορεία)
- Για το 1ο οκταμόριο αντιστοίχως το επόμενο pixel θα είναι είτε το (xi,yi+1) είτε το (xi-1,yi +1) (αριστερόστροφη πορεία)
- Άρα αν setpixel(x,y) για τιο 2ο οκταμόριο, τότε setpixel(y,x) για το 1ο οκταμόριο
- Η απόφαση στηρίζεται στην απόσταση της αναμενόμενης τιμής του Y επί της τροχιάς του κύκλου από τα δυο υποψήφια pixels
- Προσπαθούμε να υπολογίσουμε την επόμενη θέση με μοναδικά στοιχεία την τρέχουσα θέση και την ακτίνα του κύκλου



Το επόμενο pixel για το 2ο οκταμόριο

Κύκλος

- 8-πλή συμμετρία, δημιουργούμε ένα οκταμόριο (έστω 2^o)



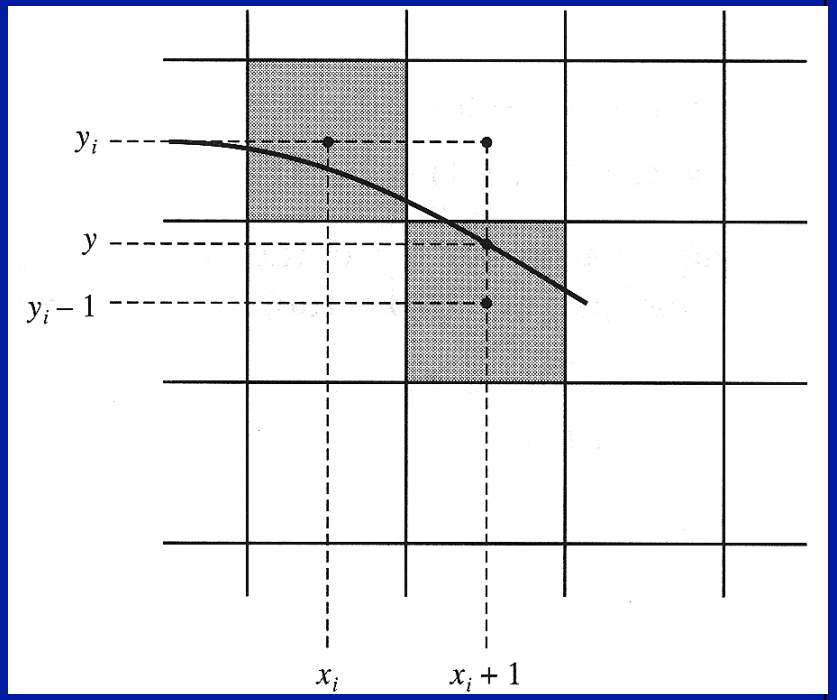

```
circle_symmetry (x,y,colour)
int x,y,colour;
   {setpixel(x,y,colour);
    setpixel(y,x,colour);
    setpixel(y,-x,colour);
    setpixel(x,-y,colour);
    setpixel(-x,-y,colour);
    setpixel(-y,-x,colour);
    setpixel(-y,x,colour);
    setpixel(-x,y,colour);
   }
```

Αλγόριθμος Bresenham – συνέχεια

- Μεταβλητές απόφασης: (πάντα >0)

$$d_1 = y_i^2 - y^2$$

$$d_2 = y^2 - (y_i - 1)^2$$



- Αν d1 < d2, επιλέγεται το πάνω pixel, αλλιώς το κάτω
- Διαφορετικά: Αν $e_i = d_1 - d_2 > 0$ σχεδίασε το κάτω pixel
- Για $x = x_i + 1$, ισχύει $y^2 = r^2 - (x_i + 1)^2$ από Πυθαγόρειο θεώρημα
- Έτσι:

$$e_i = y_i^2 - r^2 + (x_i + 1)^2 + (y_i - 1)^2 - r^2 + (x_i + 1)^2$$

$$= 2(x_i + 1)^2 + y_i^2 + (y_i - 1)^2 - 2r^2$$

Αλγόριθμος Bresenham – συνέχεια

Η τιμή e_{i+1} υπολογίζεται επαναληπτικά ως εξής:

$$
\begin{aligned}
e_{i+1} &= 2(x_{i+1}+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2r^2 \\
&= 2(x_i+2)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2r^2 \\
&= 2x_i^2 + 8x_i + 8 + y_{i+1}^2 + y_{i+1}^2 - 2y_{i+1} + 1 - 2r^2 \\
&= 2(x_i+1)^2 + 4x_i + 6 + 2y_{i+1}^2 - 2y_{i+1} + 1 - 2r^2 \\
&= e_i - y_i^2 - y_i^2 + 2y_i - 1 + 4x_i + 6 + 2y_{i+1}^2 - 2y_{i+1} + 1 \\
&= e_i + 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i)
\end{aligned}
$$

Για τον υπολογισμό του e_{i+1} χρησιμοποιείται το εξής τέχνασμα:

$$
\begin{aligned}
&\text{Αν } e_i < 0 \Rightarrow y_{i+1} = y_i \Rightarrow e_{i+1} = e_i + 4(x_i+1) + 2 \\
&\text{Αν } e_i \geq 0 \Rightarrow y_{i+1} = y_i - 1 \\
&\qquad \Rightarrow e_{i+1} = e_i + 4x_i + 6 + 2\left((y_i-1)^2 - y_i^2\right) - 2(y_i - 1 - y_i) \\
&\qquad \text{‹} \quad e_{i+1} = e_i + 4(x_i+1) + 2 - 4(y_i - 1)
\end{aligned}
$$

Θεωρώντας σαν πρώτο σημείο του 2ου οκταμορίου το σημείο $(x,y)=(0,r)$

$$e_1 = 2 + r^2 + (r-1)^2 - 2r^2 = 3 - 2r$$

Αλγόριθμος Bresenham – συνέχεια

Χρησιμοποιώντας για σημείο εκκίνησης το (0,r), παίρνουμε

$$e_1 = 2 + r^2 + (r-1)^2 - 2r^2 = 3 - 2r$$

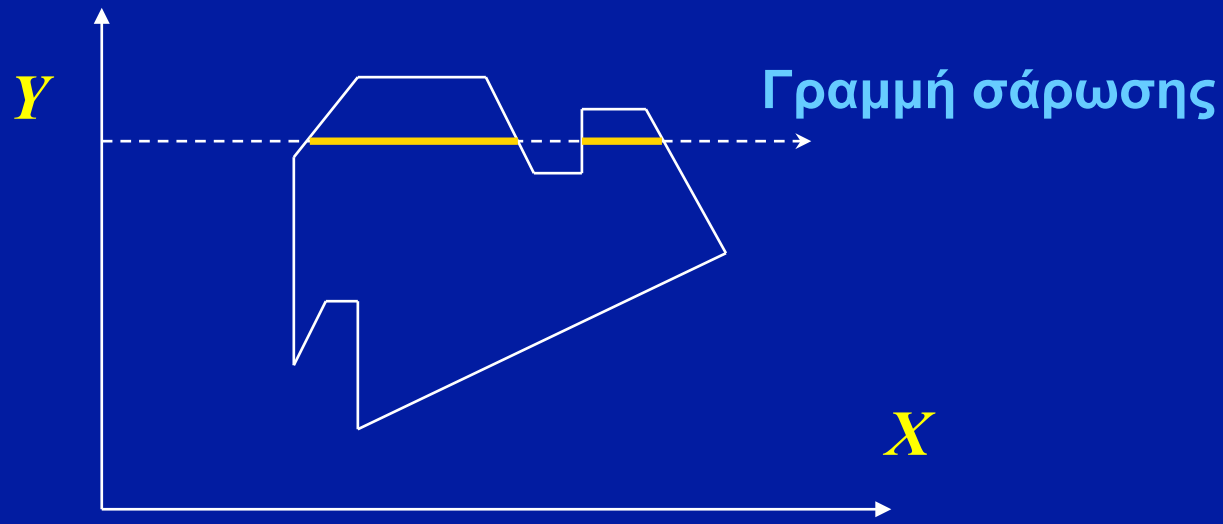
```
void circle(int r)
{
    int x,y,e;
    x=0; y=r;
    e=3-2*r;
    while (x<=y)
    {
        setPixel(x,y);
        x++;
        if (e>=0)
        {
            y--;
            e=e-4*y;
        }
        e=e+4*x+2;
    }
}
```

Αλγόριθμος Bresenham – Κύκλος με Κέντρο (x0,y0)

```
void circle(int r, int x0, int y0)
{
    int x,y,e;
    x=0; y=r;
    e=3-2*r;
    while (x<=y)
    {
        setPixel(x+x0,y+y0);
        x++;
        if (e>=0)
        {
            y--;
            e=e-4*y;
        }
        e=e+4*x+2;
    }
}
```

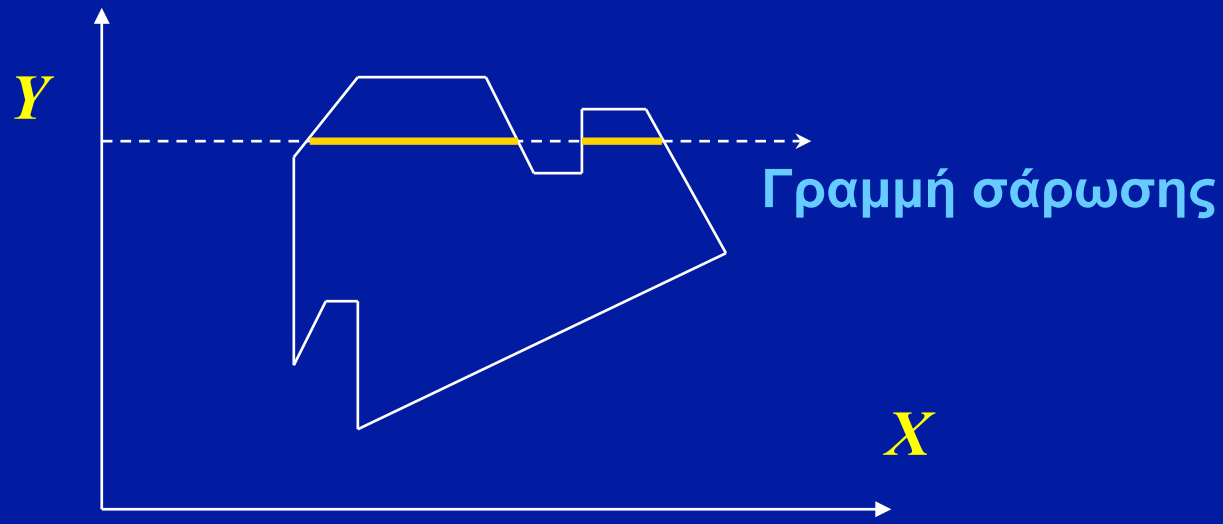
Χρωματισμός - Γέμισμα Πολυγώνων

- Ζητείται ο χρωματισμός του εσωτερικού ενός πολυγώνου που ορίζεται από τυχαία ευθύγραμμα τμήματα
- Αλγόριθμος (YX)



Χρωματισμός - Αλγόριθμος YX

- Βρίσκουμε αναλυτικά τις τομές της κάθε πλευράς με τις γραμμές σάρωσης και τις αποθηκεύουμε σε μια λίστα
- Ταξινομούμε τη λίστα πρώτα κατά Y και μετά κατά X
- Απομακρύνουμε διαδοχικά από τη λίστα ζεύγη σημείων και σχεδιάζουμε τα pixels ανάμεσά τους (τα δύο σημεία έχουν κοινό Y και συνεχόμενα X)
- Οι οριζόντιες γραμμές αγνοούνται

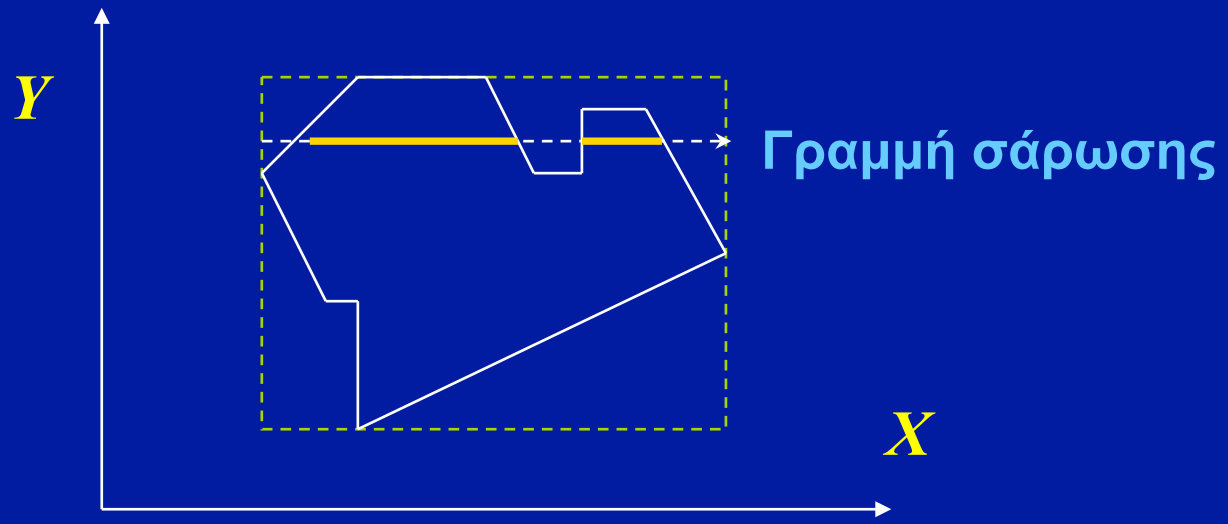


Χρωματισμός - Πρόβλημα των Αλγορίθμων με Ταξινόμηση

- Η πολυπλοκότητά τους αυξάνει μη γραμμικά με το μέγεθος των πολυγώνων!
- Χρειάζεται να δεσμευτεί δυναμικά αρκετή μνήμη για τη σχεδίαση, γεγονός που φορτώνει τη μνήμη και εισάγει καθυστερήσεις

Χρωματισμός - Αλγόριθμος Τυφλής Σάρωσης

- Ξεκινάει από εξωτερικό σημείο κάθε scan-line και σαρώνει προς την ίδια κατεύθυνση X τις γραμμές
- Σε κάθε σημείο, ελέγχει αν βρίσκεται αριστερά ή δεξιά των πλευρών
- Αν έχει συναντήσει περιττό αριθμό πλευρών, τότε σχεδιάζει το pixel, αλλιώς προχωράει στο επόμενο

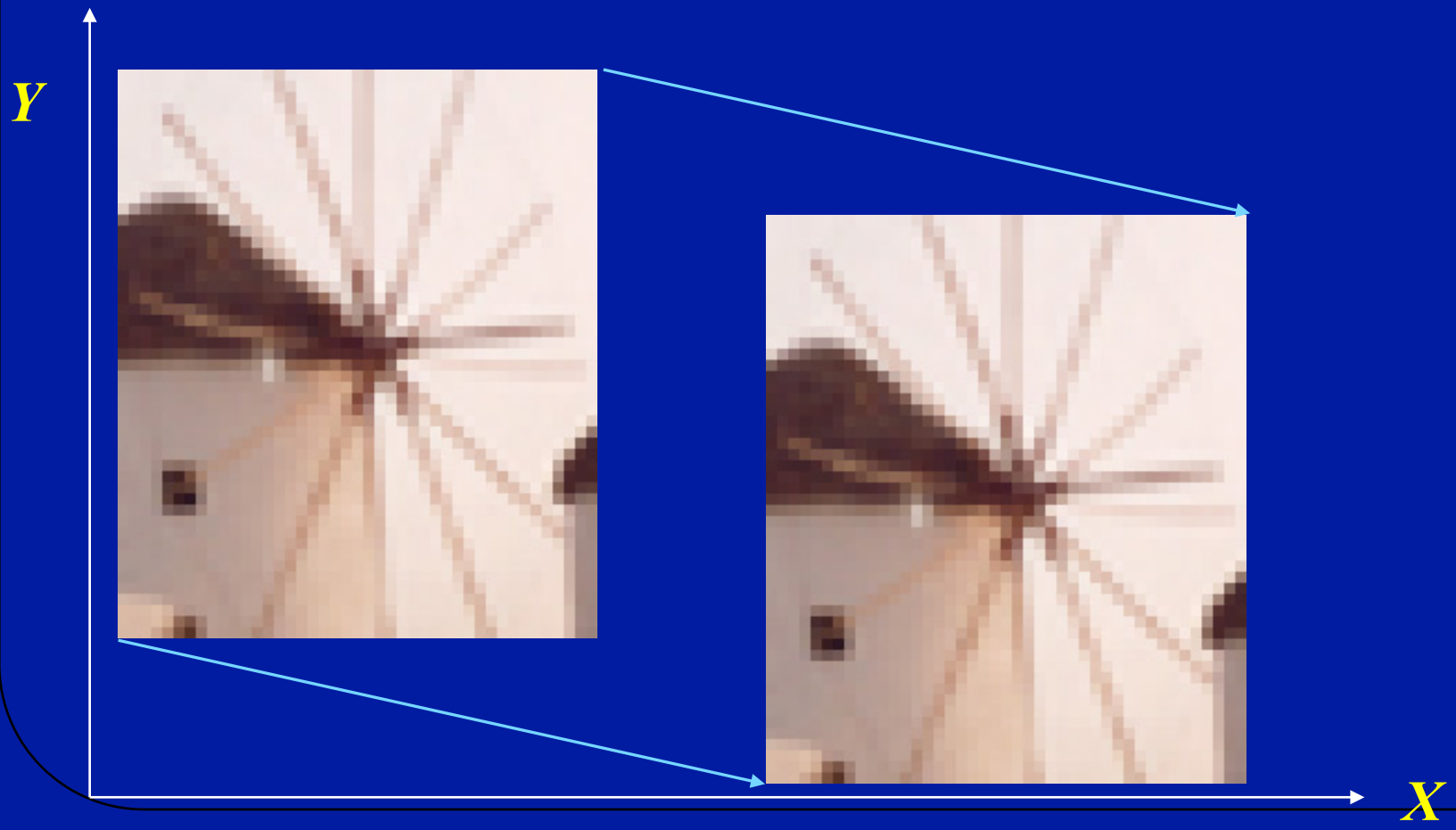


Χρωματισμός - Προβλήματα Αλγόριθμου Τυφλής Σάρωσης

- Απαιτεί συνεχείς ελέγχους
- Οι γραμμές των πολυγώνων πρέπει να έχουν σταθερή φορά (πχ δεξιόστροφη)

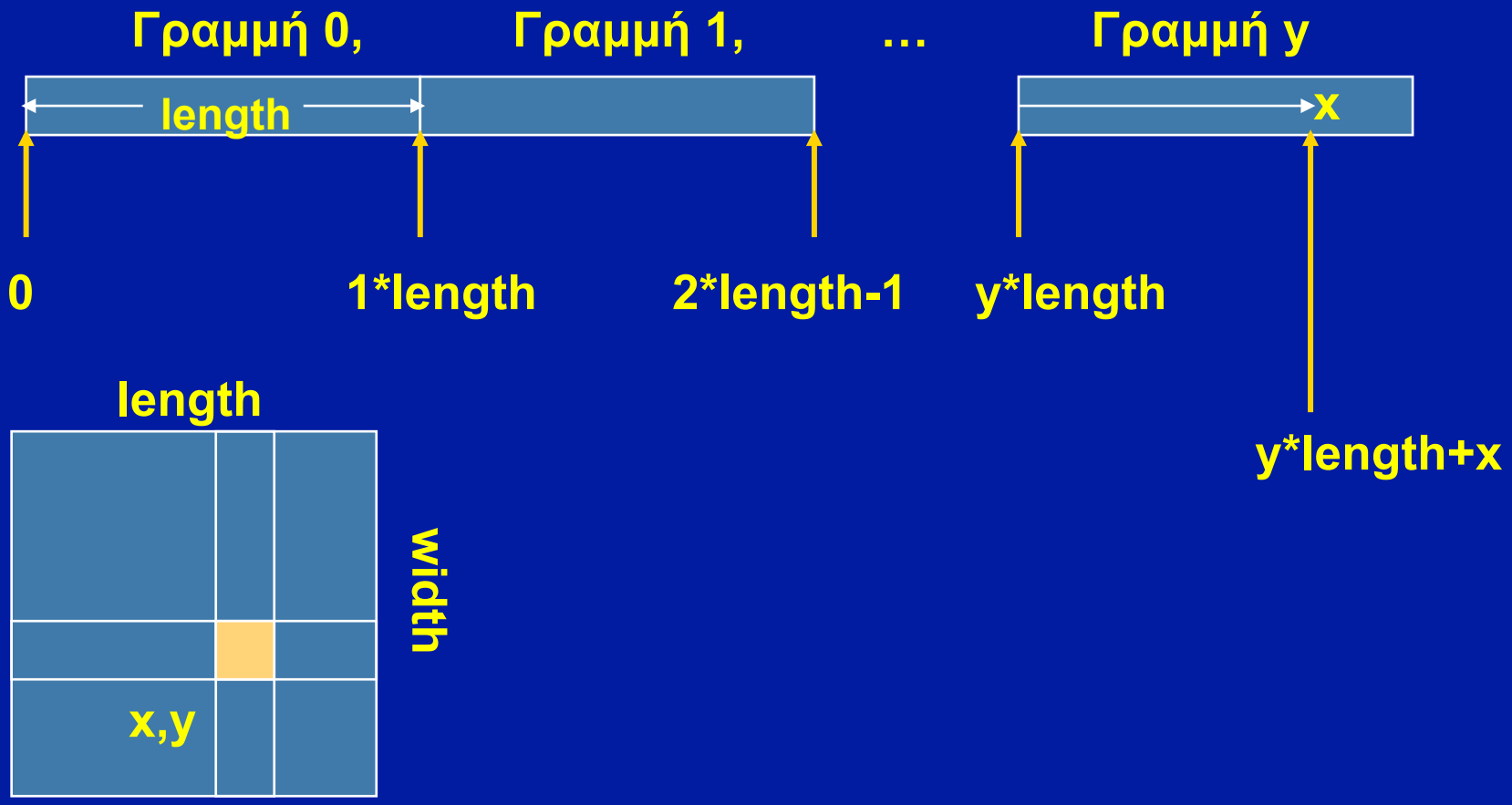
Γρήγορη αντιγραφή περιοχής Pixels (Bit-Blitting)

Κρίσιμης σημασίας σε γραφικά περιβάλλοντα με επικαλυπτόμενα παράθυρα ή scrolling (πχ. Windows)



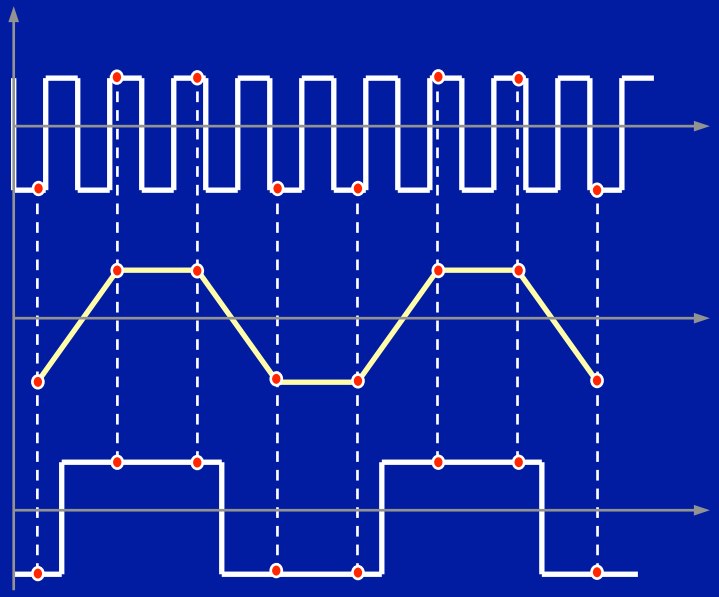
Εσωτερική δομή του Frame Buffer

Οι εικόνες στην κύρια μνήμη ή στην μνήμη της κάρτας γραφικών γράφονται σαν μονοδιάστατοι πίνακες:



Aliasing (Ταύτιση)

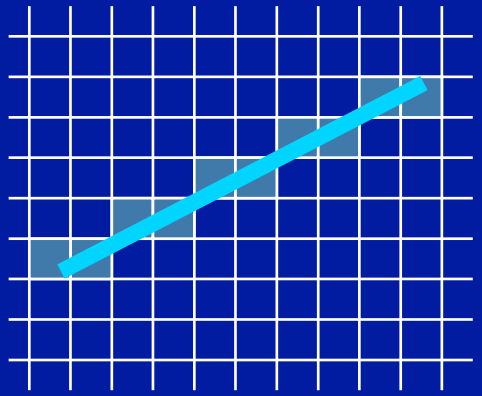
- Aliasing: Παραγωγή ίδιου ψηφιακού σήματος από δύο αναλογικά σήματα διαφορετικής συχνότητας
- Στα γραφικά προκύπτει το φαινόμενο όταν αναλυτικές μορφές (όπως ευθύγραμμα τμήματα κλπ) μετατρέπονται σε διακριτές, δηλαδή pixels

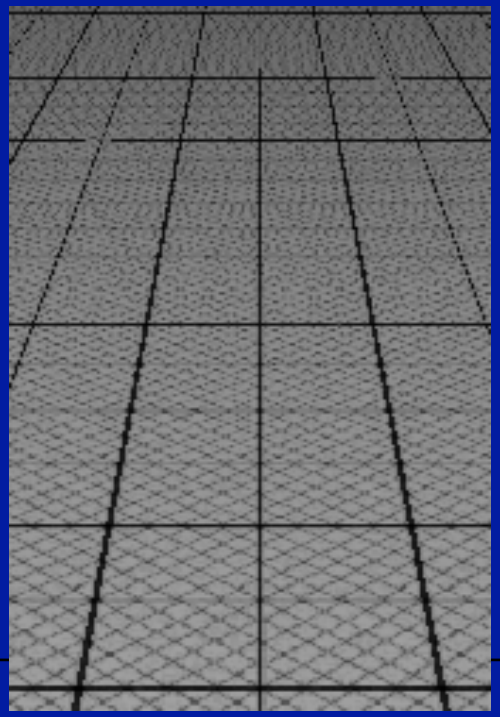


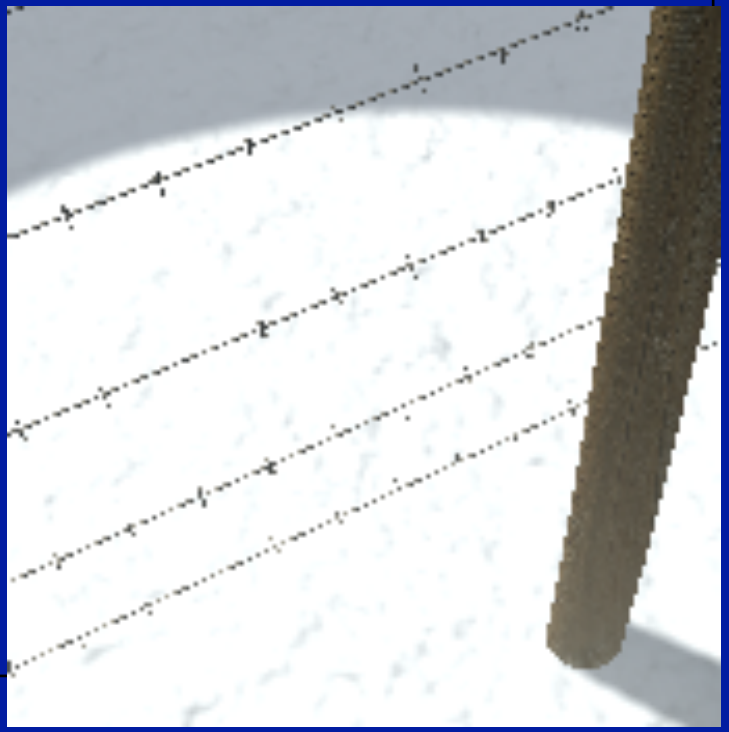
Aliasing και Γραφικά

Κύριες περιπτώσεις εμφάνισης aliasing:

- Η τεθλασμένη εμφάνιση ακμών πολυγώνων και γραμμών
- Η λανθασμένη εμφάνιση επαναλαμβανόμενων λεπτομερειών (πχ υφή)

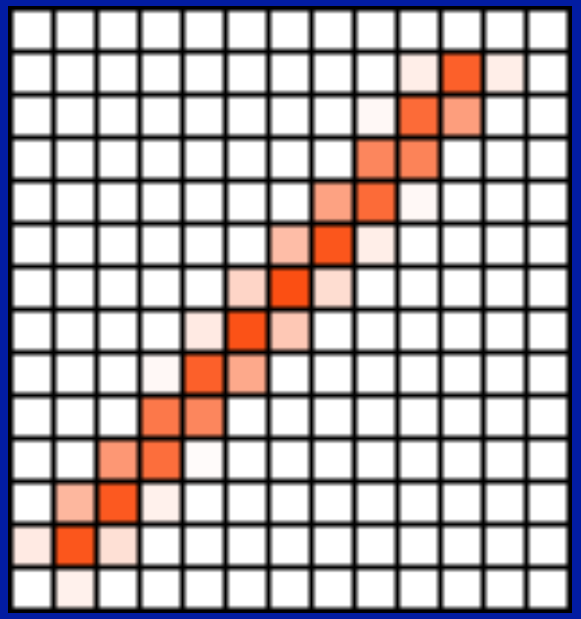






Αντιμετώπιση Aliasing

- Κατασκευαστικοί αλγόριθμοι για γραμμές: χρωματίζουν και γειτονικά pixels με λιγότερο χρώμα – πρόβλημα με επικαλυπτόμενα σχήματα
- Low-pass post-filtering
- Supersampling
- Pre-filtering πολυγώνων
- Low-pass Pre-filtering για textures (mip-mapping)



MxN Supersampling

- Παράγουμε MxN δείγματα (subpixels) για κάθε pixel τα οποία αποθηκεύονται στον Frame Buffer
- Στη συνέχεια η υπερδειγματοληπτημένη εικόνα φιλτράρεται αποτυπώνοντας σε κάθε pixel το μέσο όρο των MXN σημείων
- Η μέθοδος απαιτεί αρκετή μνήμη (MxN) για real-time απεικόνιση και MxN μεγαλύτερη ισχύ
- Δημοφιλής μέθοδος για σύγχρονες κάρτες γραφικών (πχ Nvidia GeForce, ATI Radeon chipsets)

Supersampling