# Γραφικά Υπολογιστών

Ιόνιο Πανεπιστήμιο
Τμήμα Πληροφορικής

Στέργιος Παλαμάς, Επίκουρος Καθηγητής

Διαφάνειες βασισμένες στο υλικό του κ. Φοίβου Μυλωνά

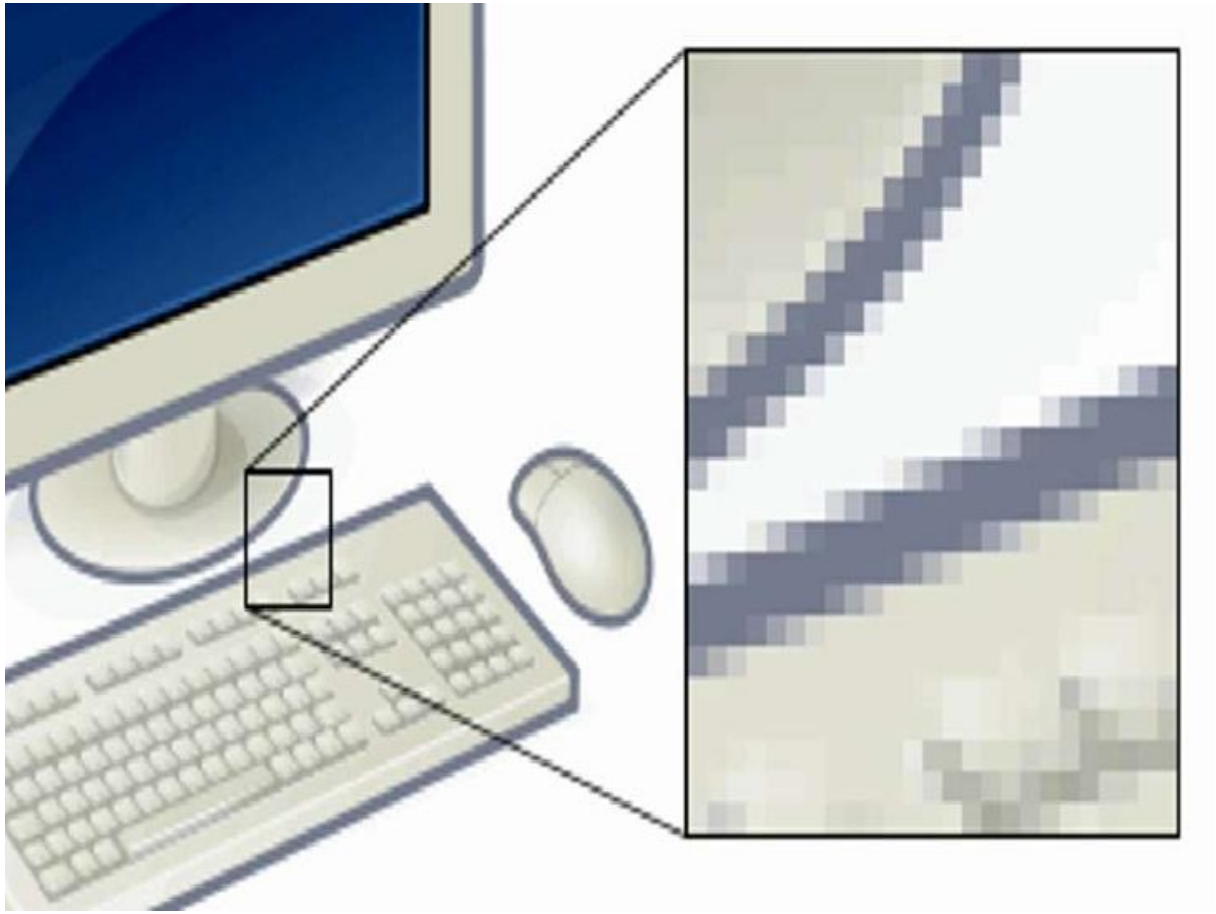
# Μάθημα 9:

## Σχεδίαση Ευθύγραμμων Τμημάτων

Διαφάνειες βασισμένες στο υλικό του κ. Φοίβου Μυλωνά

# Σχεδίαση

- Ποιο είναι το Νο 1 πρόβλημα των 2D οθονών ?



# Σχεδίαση

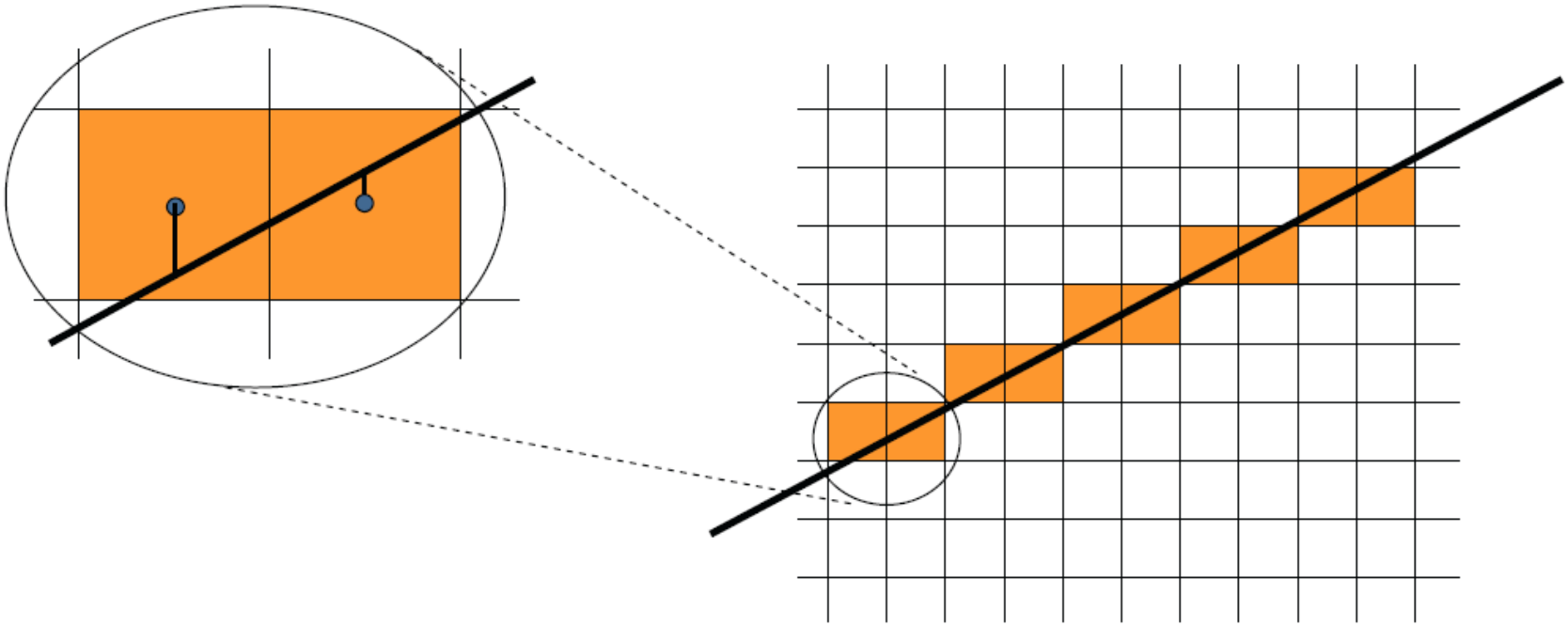
- 2Δ οθόνες αποτελούνται από διακριτά πλέγματα pixels!
- **Σχεδίαση:** μετατροπή 2Δ στοιχειωδών σχημάτων σε διακριτή παράσταση pixels
  - με όσο το δυνατόν καλύτερη προσέγγιση!
- Πολυπλοκότητα σχεδίασης: **O(P·p)**, όπου:
  - **P** είναι το πλήθος στοιχειωδών σχημάτων, και
  - **p** ο αριθμός των pixels

# Απεικόνιση Σχημάτων

- Οι αλγόριθμοι σχεδίασης υποστηρίζονται πλέον και από υλικό (hardware chips).
- **Κριτήρια «καλού» αλγορίθμου**:
  - Όσο γίνεται πιο κοντά στη μαθηματική πορεία.
  - Διατήρηση σταθερού πάχους ανεξάρτητα από το μήκος ή/και την κλίση.
    - minimum πάχος: 1 pixel, όταν / − |
  - Με όσο το δυνατόν μικρότερο υπολογιστικό κόστος (μεγαλύτερη ταχύτητα).

# Απεικόνιση Σχημάτων

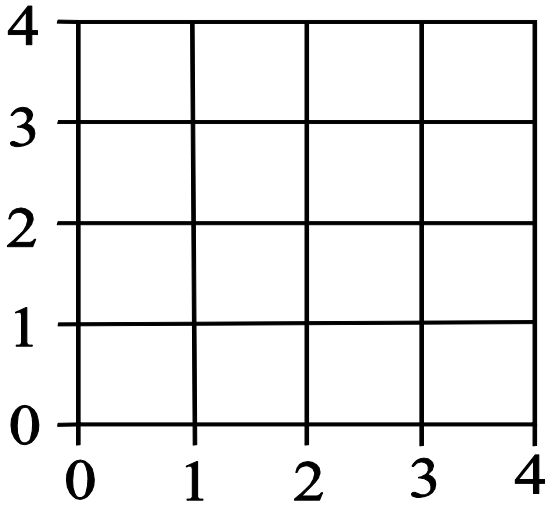
- Κριτήρια «καλού» αλγορίθμου:
  - Όσο γίνεται πιο κοντά στη μαθηματική πορεία, σταθερό πάχος, μεγάλη ταχύτητα.



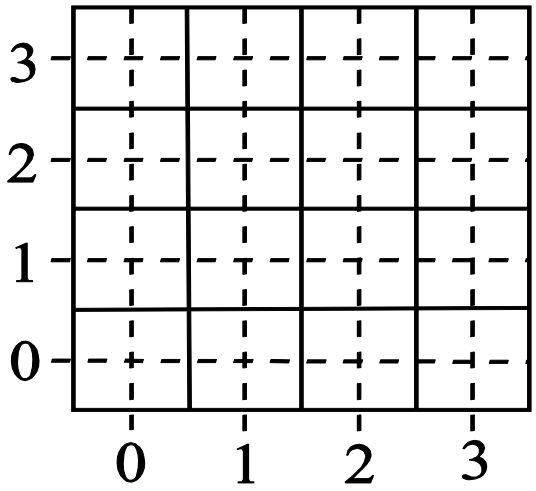
# Έννοιες Σχεδίασης

- 2 τρόποι για να παραστήσουμε το πλέγμα των pixels
  - Κέντρα σε **ημίσειες** συντεταγμένες
  - Κέντρα σε **ακέραιες** συντεταγμένες (**προτεινόμενο**)

- Κέντρα σε ημίσειες συντεταγμένες

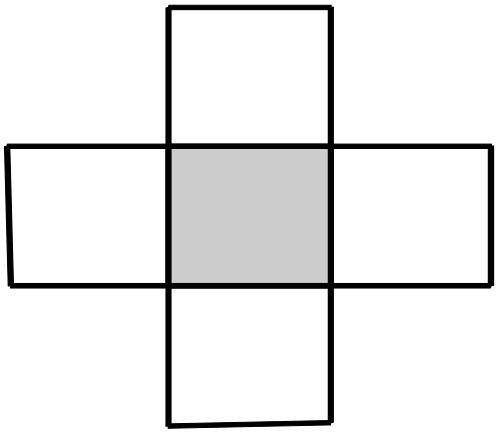



Κέντρα σε ακέραιες συντεταγμένες

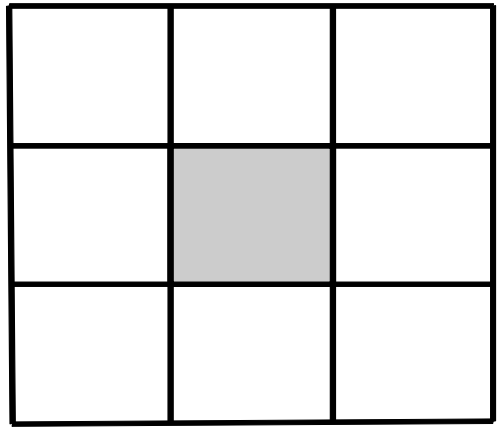


# Έννοιες Σχεδίασης

- **Σύνδεση:** αναπαράσταση των γειτόνων ενός pixel
  - **Τετραπλή** σύνδεση
  - **Οκταπλή** σύνδεση
- Τετραπλή Σύνδεση



Οκταπλή Σύνδεση



- Προκλήσεις της σχεδίασης
  - Καθορισμός των pixels που περιγράφουν το σχήμα ακριβώς
  - Αποδοτικότητα

# Απεικόνιση **Ευθύγραμμου** Τμήματος

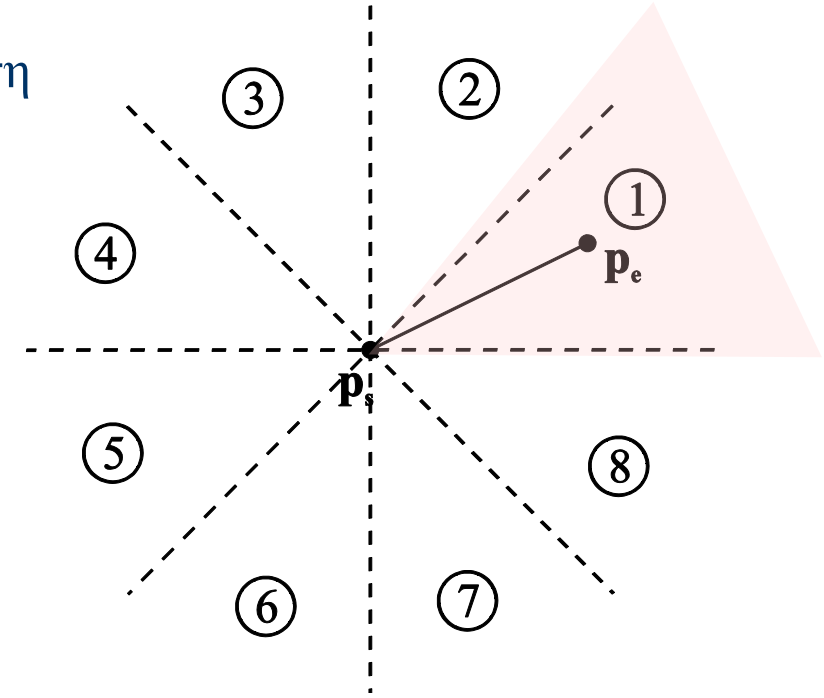
- Αλγόριθμοι υποστηρίζονται και από hardware.
- Κριτήρια «καλού» αλγορίθμου:
  - Όσο γίνεται πιο κοντά στη μαθηματική πορεία
  - Πάχος σταθερό και ανεξάρτητο από μήκος και κλίση.
  - Με όσο το δυνατόν μικρότερο υπολογιστικό κόστος (μεγαλύτερη ταχύτητα).

# Απεικόνιση Ευθύγραμμου Τμήματος

- Βηματική προσέγγιση:
  - 1ος αλγόριθμος
  - 2ος αλγόριθμος: επέκταση 1ου
  - 3ος αλγόριθμος: επέκταση 2ου
  - 4ος αλγόριθμος: επέκταση 3ου
    - αποδοτικός «αλγόριθμος του Bresenham»

# Αλγόριθμοι Σχεδίασης Ευθύγραμμου Τμήματος

- Επιθυμητές ιδιότητες ενός αλγορίθμου σχεδίασης ευθ. τμημάτων:
  - Τα pixels να είναι όσο πιο κοντά στη μαθηματική πορεία της ευθείας
  - Σταθερό πλάτος, ανεξάρτητο από την κλίση της ευθείας
  - Όχι κενά
  - Υψηλή απόδοση




**Τα 8 οκταμόρια και ένα ευθ. τμήμα στο πρώτο οκταμόριο**

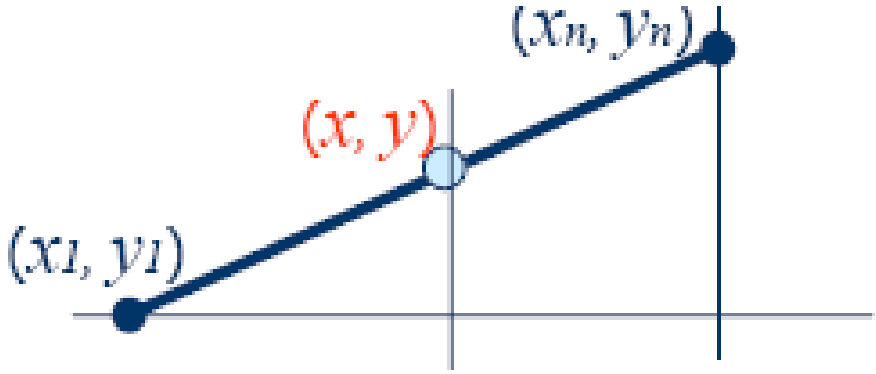
# Αλγόριθμος 1

- Έστω ευθύγραμμο τμήμα μεταξύ pixels $P_1(x_1,y_1)$, $P_n(x_n,y_n)$ στο πρώτο οκταμόριο.

εξίσωση ευθείας

Τότε: $y=s*x+b$

όπου




$$s = \frac{y_n - y_1}{x_n - x_1} = \frac{\Delta y}{\Delta x} \quad \text{και} \quad b = \frac{y_1 x_n - y_n x_1}{x_n - x_1}$$

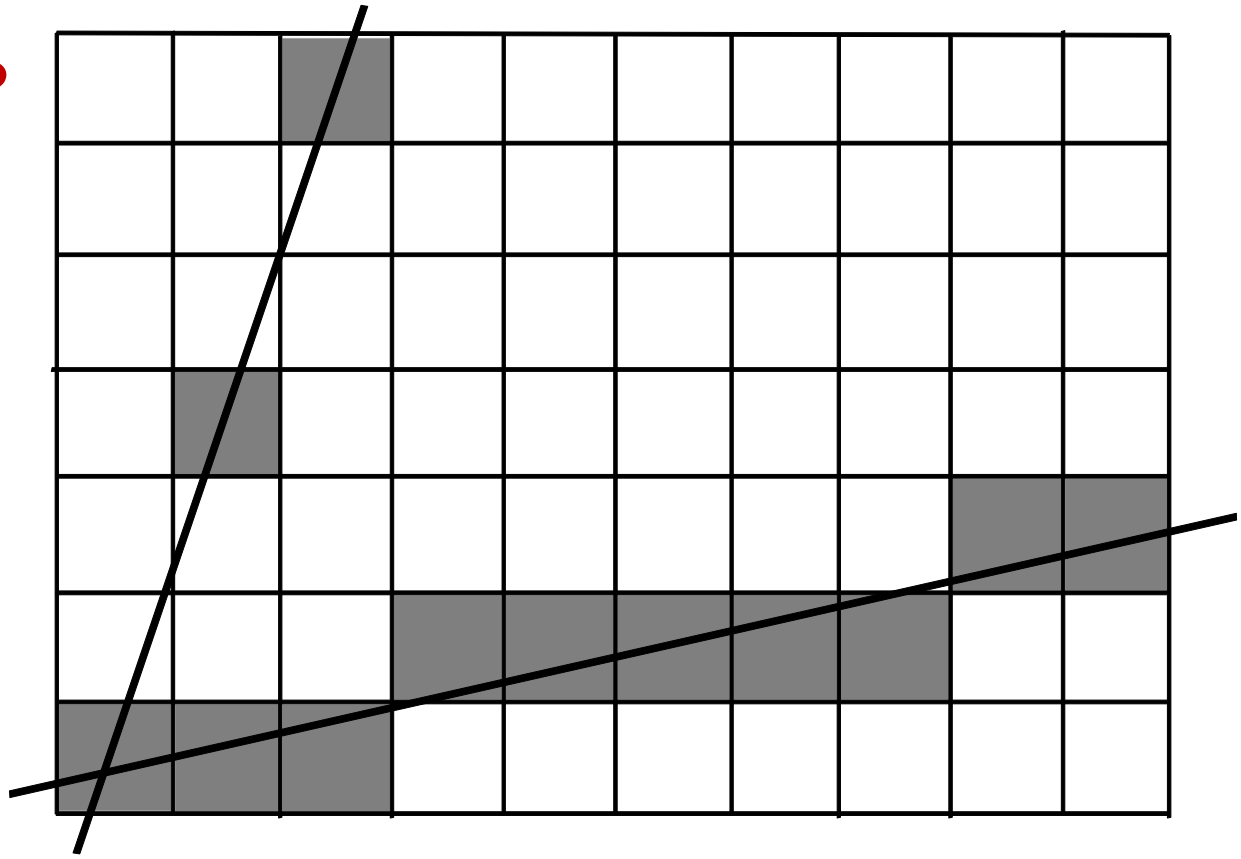
# Υλοποίηση

```
void line1(x1, y1, xn, yn, colour)
int x1, y1, xn, yn, colour;
/* colour: ακέραια αναπαράσταση (τιμή) του
   χρώματος του ευθύγραμμου τμήματος */
{
   float s, b, y;
   int x;
   s =( yn - y1)/(float)( xn - x1);
   b =( y1 * xn - yn * x1)/(float)( xn - x1);
   for (x = x1; x <= xn; x++){
       y = s * x + b;
       setpixel(x, round(y), colour);
   }
}
```

# Αλγόριθμος 1

- Παράδειγμα του **`line1`** στο πρώτο και δεύτερο οκταμόριο:

**Γιατί ?**

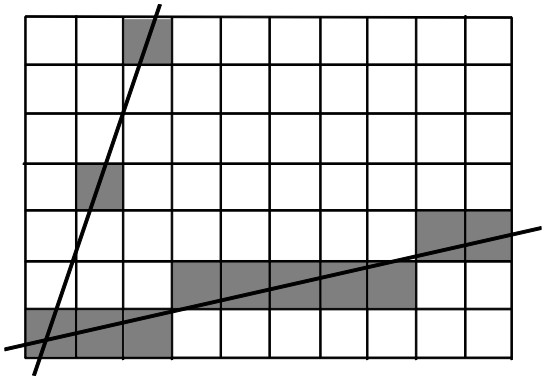


# Αλγόριθμος 1

- Παράδειγμα **`line1`** στο δεύτερο οκταμόριο:

$P_1(x_1,y_1)$, $P_n(x_n,y_n)$, y=s*x+b, $s = \frac{y_n - y_1}{x_n - x_1} = \frac{\Delta y}{\Delta x}$ και $b = \frac{y_1 x_n - y_n x_1}{x_n - x_1}$

$P_1(1,1)$
$P_2(3,7)$
s = 3
b = -2
=> y=3x-2



x=1 ⇨ y = 1 (1,1)

x=2 ⇨ y = 4 (2,4)

x=3 ⇨ y = 7 (3,7)

x=4 ⇨ y = 10 (4,10)

```
void line1(x1, y1, xn, yn, colour)
int x1, y1, xn, yn, colour;
/* colour: ακέραια αναπαράσταση (τιμή) του χρώματος
    του ευθύγραμμου τμήματος */
{
    float s, b, y;
    int x;
    s =( yn - y1)/(float)( xn - x1);
    b =( y1 * xn - yn * x1)/(float)( xn - x1);
    for (x = x1; x <= xn; x++){
          y = s * x + b;
          setpixel(x, round(y), colour);
    }
}
```

# Παραδείγματα “line 1”

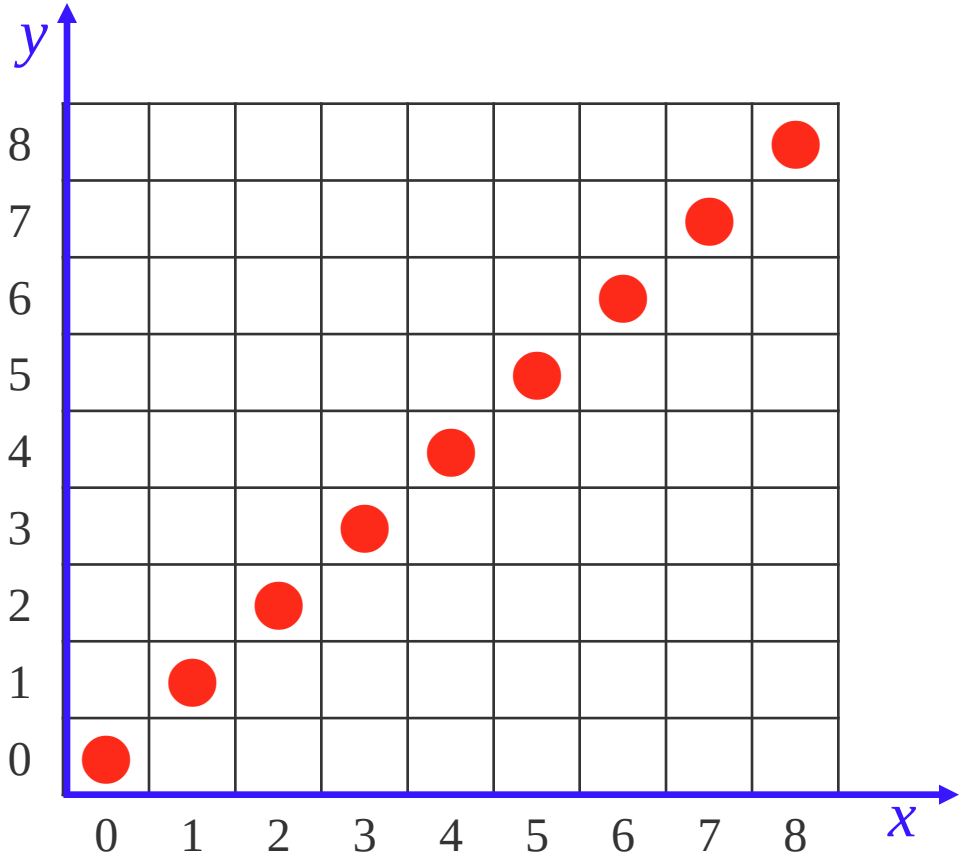
$y = x$

$s = 1$

$b = 0$

$$s = \frac{y_n - y_1}{x_n - x_1} = \frac{\Delta y}{\Delta x} \quad \text{και} \quad b = \frac{y_1 x_n - y_n x_1}{x_n - x_1}$$

| x | y |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |



# Παραδείγματα “line 1”

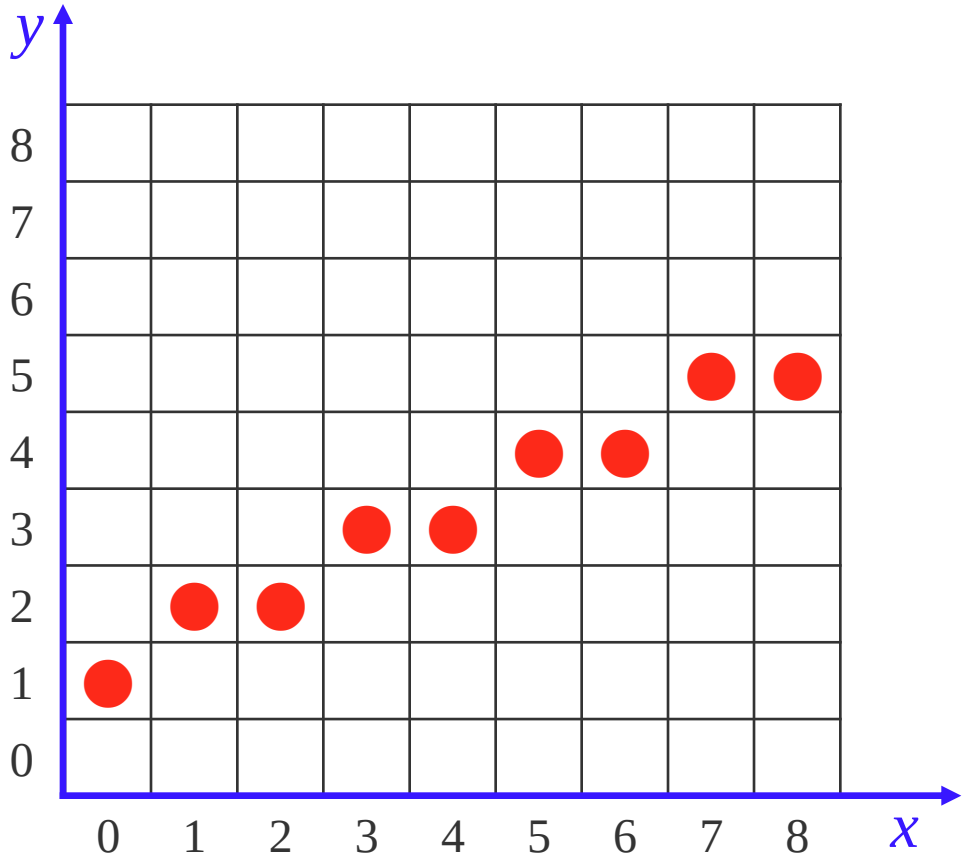
$y = ½\,x + 1$

$s = ½$

$b = 1$

$$s = \frac{y_n - y_1}{x_n - x_1} = \frac{\Delta y}{\Delta x} \quad \text{και} \quad b = \frac{y_1 x_n - y_n x_1}{x_n - x_1}$$

| x | y | round(y) |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1.5 | 2 |
| 2 | 2 | 2 |
| 3 | 2.5 | 3 |
| 4 | 3 | 3 |
| 5 | 3.5 | 4 |
| 6 | 4 | 4 |
| 7 | 4.5 | 5 |
| 8 | 5 | 5 |



# Παραδείγματα “line 1”

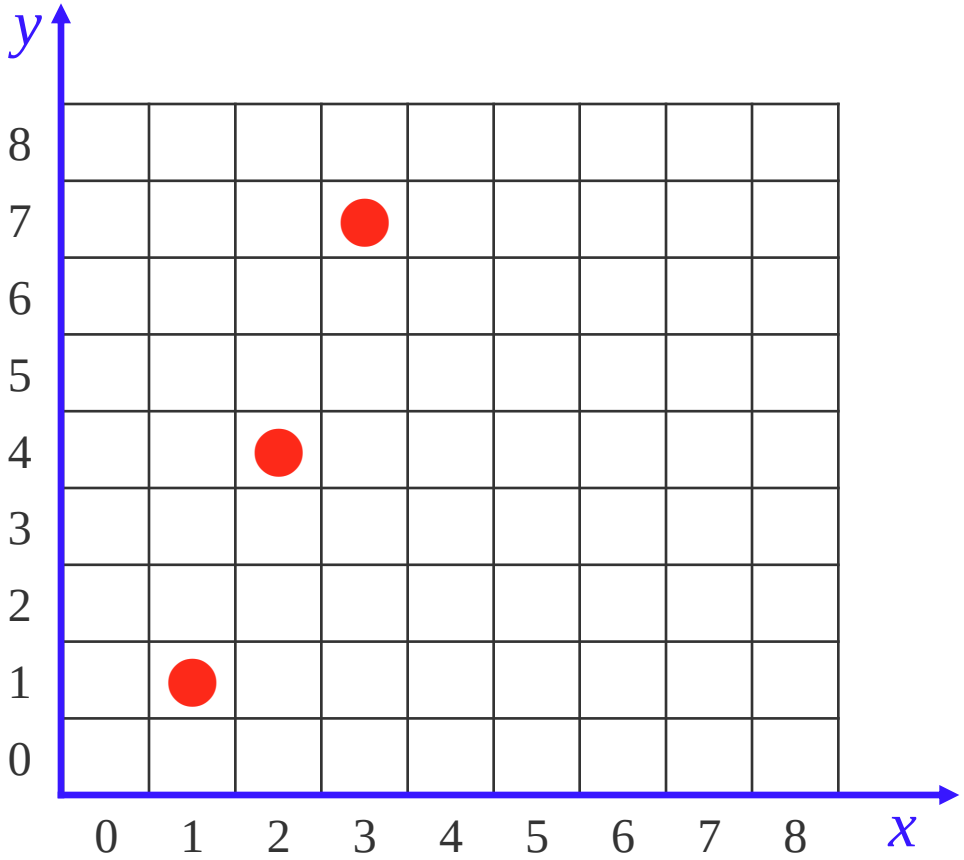
$y = 3x - 2$

$s = 3$

$b = -2$

$$s = \frac{y_n - y_1}{x_n - x_1} = \frac{\Delta y}{\Delta x} \quad \text{και} \quad b = \frac{y_1 x_n - y_n x_1}{x_n - x_1}$$

| x | y | round(y) |
|---|---|---|
| 0 | -2 | -2 |
| 1 | 1 | 1 |
| 2 | 4 | 4 |
| 3 | 7 | 7 |
| 4 | 10 | 10 |
| 5 | 13 | 13 |
| 6 | 16 | 16 |
| 7 | 19 | 19 |
| 8 | 22 | 22 |



# Αλγόριθμος 2 (αποδοτικότερος)

- Σε κάθε επανάληψη, το **x** αυξάνεται κατά 1:

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

  ⇨ ο πολλαπλασιασμός μέσα στο βρόγχο μπορεί να αποφευχθεί μετατρέποντας τον τύπο σε αναδρομικό:

$$y_{i+1} = sx_{i+1} + b =$$

$$s(x_i + 1) + b =$$

$$sx_i + b + s =$$

$$y_i + s$$

# Υλοποίηση

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

- Πλεονεκτήματα:
  - πρόσθεση αντί πολλαπλασιασμού
  - δεν υπολογίζεται πουθενά το b!

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

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

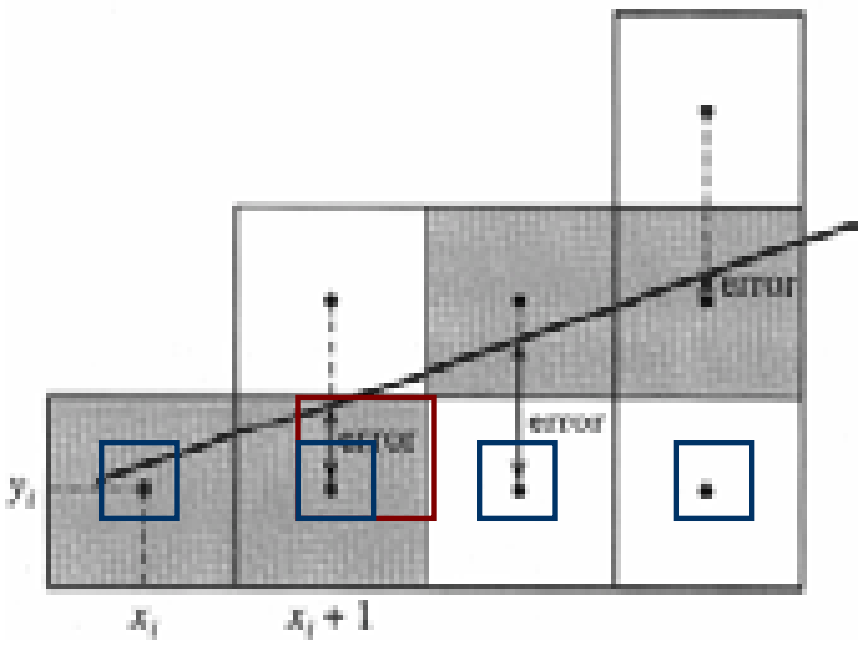
Το επόμενο y είναι το προηγούμενο αυξημένο κατά την κλίση (s) του ευθύγραμμου τμήματος.

Το x αυξάνει πάντα κατά 1 μέχρι να φτάσει το τελικό $x_n$

# Αλγόριθμος 3

- **Αποφυγή της στρογγυλοποίησης μέσα στο βρόγχο** με χωρισμό του **`y`** σε ακέραιο και δεκαδικό μέρος (**`error`**) και χρήση μόνο του ακεραίου για εύρεση του επομένου σημείου.

- Σφάλμα = κατακόρυφη απόσταση pixel από ιδεατή πορεία.



# Υλοποίηση

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

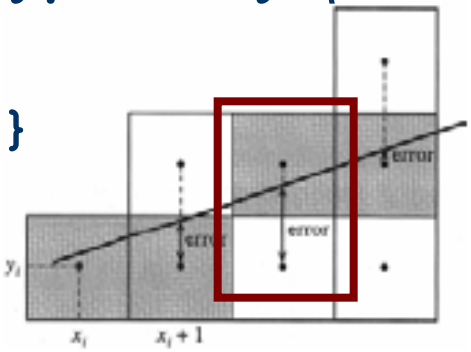
# Αλγόριθμος 3

- Ο **`line3`** μοιάζει με τον υπολογισμό του δίσεκτου έτους!
  - **Σε κάθε επανάληψη, η κλίση προστίθεται στο error**.

    ```
    error = error + s;
    ```

  - Όταν το error είναι **μεγαλύτερο της μισής μονάδας**, η ευθεία μεταβαίνει στο **επόμενο pixel**.

    ```
    if (error >= 0.5){ y++; error--}
    ```




  - Το y αυξάνει και το error μειώνεται αντίστοιχα, ώστε <u>το άθροισμά τους να παραμένει σταθερό</u>.

# Αλγόριθμος 3

- Ο **`line3`** μοιάζει με τον υπολογισμό του δίσεκτου έτους!
  - Κάθε έτος έχει 365.25 ημέρες, αλλά το ημερολογιακό έτος έχει ακέραιο πλήθος ημερών.
  - Κάθε 4 έτη δημιουργείται σφάλμα της τάξεως της μίας πλήρους ημέρας.
  - Γι' αυτό κάθε 4 χρόνια προστίθεται μια ημέρα στο έτος ώστε το σφάλμα να «απορροφάται».

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

Το επόμενο y είναι το προηγούμενο αυξημένο κατά την κλίση (s) του ευθύγραμμου τμήματος.

Το x αυξάνει πάντα κατά 1 μέχρι να φτάσει το τελικό $x_n$

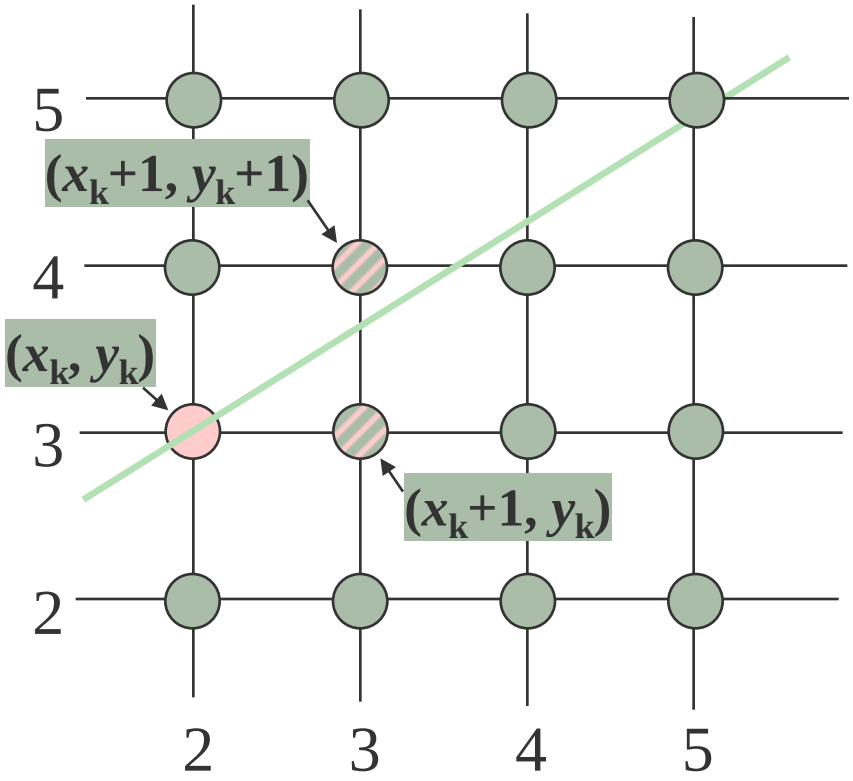
```
line3 (x1, y1, xn, yn, colour)
int x1, y1, xn, yn, colour;
{
float s, error;
int x, y;
    s =(yn - y1)/(xn - x1);
    y = y1;
    error = 0;
    for (x = x1; x <= xn; x++){
          setpixel(x, y, colour);
          error = error + s;
           if (error >= 0.5){
                    y++;
    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

Το y παραμένει σταθερό μέχρι το error (η απόκλιση από το επιθυμητό σημείο που αυξάνεται σε κάθε βήμα κατά τη κλίση της γραμμής) να ξεπεράσει την τιμή 0.5. Τότε αυξάνουμε το y κατά 1 και χαμηλώνουμε το error. Το error θα παίζει από -0.5 → 0.5

Το x αυξάνει πάντα κατά 1 μέχρι να φτάσει το τελικό $x_n$

# Αλγόριθμος 4 (Bresenham) -Η βασική ιδέα!

Μετακίνηση κατά μήκος του **άξονα x σε μοναδιαία διαστήματα** και σε κάθε βήμα **επιλογή μεταξύ δύο διαφορετικών y συντεταγμένων**




Για παράδειγμα, από τη θέση (2, 3) θα πρέπει να επιλέξουμε μεταξύ (3, 3) και (3, 4).

Θέλουμε το σημείο που βρίσκεται πιο κοντά στην ιδεατή γραμμή.

# Αλγόριθμος 4 (Bresenham)

- Jack E. Bresenham, IBM, 1962(!)
- Κατάλληλη κλιμάκωση (scaling)
  - `s, error`
  - συνθήκης επιλογής pixel

  ⇨ **αντικατάσταση πραγματικών (real)** μεταβλητών από **ακέραιες (integers)**
- Χρησιμοποιεί μόνο **πρόσθεση**, **αφαίρεση** και τεχνική **bit-shifting** ακεραίων ⇨ πολύ φθηνές πράξεις στην αρχιτεκτονική υπολογιστών!

# Αλγόριθμος 4 (Bresenham)

1. Πολλαπλασιάζοντας με $dx=x_n-x_1$
   - οι μεταβλητές **s** και **error** γίνονται ακέραιες.
   - **S * dx = (dy/dx) * dx = dy**
   - **error = 0.5 * dx = dx/2 (στο δυαδικό υλοποιείται με 1 shift δεξιά όλων των ψηφίων)**
   - **Πχ** $0\ 0\ 0\ 0\ 1\ 0\ 1\ 0_{<2>} = 10_{<10>}$

     Shift 1 θέση →

     $0\ 0\ 0\ 0\ 0\ 1\ 0\ 1_{<2>} = 5_{<10>}$

2. Η σύγκριση για την επιλογή του επόμενου pixel γίνεται με **floor(dx/2) αντί του 0.5** (δηλαδή ελέγχουμε αν **error ≥ dx/2**)
   - Αφαιρώντας από την αρχική τιμή του **error** το **dx/2**, η σύγκριση μπορεί να γίνει και με το 0.

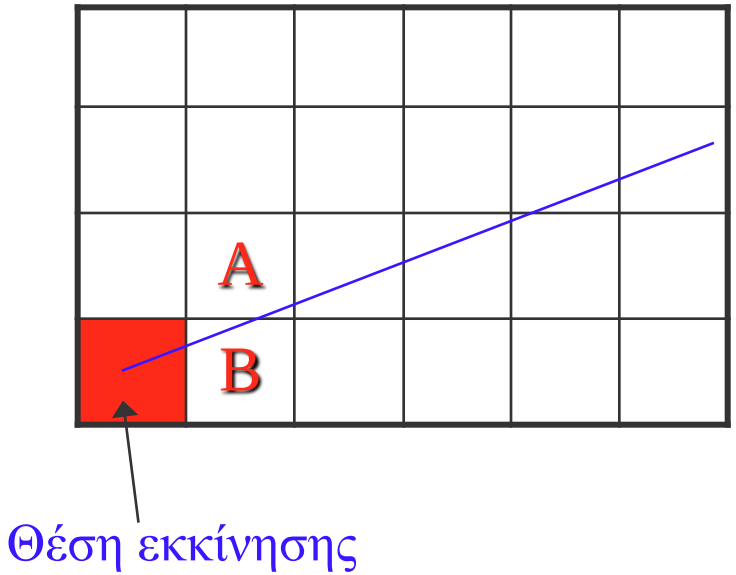
3. Αντί να αυξάνουμε το error κατά 1 , θα το αυξάνουμε κατά 1* dx = dx

- Λειτουργεί για <u>θετικές κλίσεις</u>:

  $x_1<x_n$ και κλίση **[0,1]**

# Αλγόριθμος 4 (Bresenham)

- Βάση του αλγορίθμου:




- Από τη θέση εκκίνησης **διάλεξε για επόμενο μόνο το A ή το B**

# Υλοποίηση

```
line4 (x1, y1, xn, yn, colour)
int x1, y1, xn, yn, colour;
{
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, colour);
    error = error + dy;
    if (error >= 0){ y++; error = error - dx}
   }
}
```

**Υλοποίηση σε 58 γλώσσες προγραμματισμού:**
**http://rosettacode.org/wiki/Bitmap/Bresenham%27s_line_algorithm**

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

```
line4 (x1, y1, xn, yn, colour)
int x1, y1, xn, yn, colour;
{
int error, x, y, dx, dy;
dx = xn-x1; dy = yn-y1;
y = y1;
error =-dx/2;
for (x = x1; x <= xn; x++) {
   setpixel(x, y, colour);
   error = error + dy;
   if (error >= 0){
      y++;
      error = error - dx
   }
}
}
```

Αντί του s = dy/dx χρησιμοποιούμε το dy

Αντί να συγκρίνουμε με 0.5 συγρίνουμε με dx*0.5 = dx/2. Βάζοντας αρχικά error = -dx/2 αντί το 0 , μπορούμε μετά να συγκρίνουμε με το 0

Αντί να μειώνουμε το error κατά 1 , το μειώνουμε κατά 1*dx = dx

Το y παραμένει σταθερό μέχρι το error (η απόκλιση από το επιθυμητό σημείο που αυξάνεται σε κάθε βήμα κατά τη κλίση της γραμμής) να ξεπεράσει την τιμή 0.5. Τότε αυξάνουμε το y κατά 1 και χαμηλώνουμε το error επίσης κατά 1.

Το x αυξάνει πάντα κατά 1 μέχρι να φτάσει το τελικό $x_n$

Το y παραμένει σταθερό μέχρι το error (που αυξάνεται σε κάθε βήμα κατά dy) να ξεπεράσει την τιμή 0. Τότε αυξάνουμε το y κατά 1 και χαμηλώνουμε το error κατά dx.

Το x αυξάνει πάντα κατά 1 μέχρι να φτάσει το τελικό $x_n$

# Αλγόριθμος Bresenham

- **Αντικατάσταση πραγματικών** μεταβλητών του **`line3`** από **ακέραιες.**
- Πολλαπλασιάζουμε με $\mathbf{dx = x_e - x_s}$
  → **`s`** και **`error`** γίνονται ακέραιοι.
- Η συνθήκη για επιλογή επόμενου pixel γίνεται:

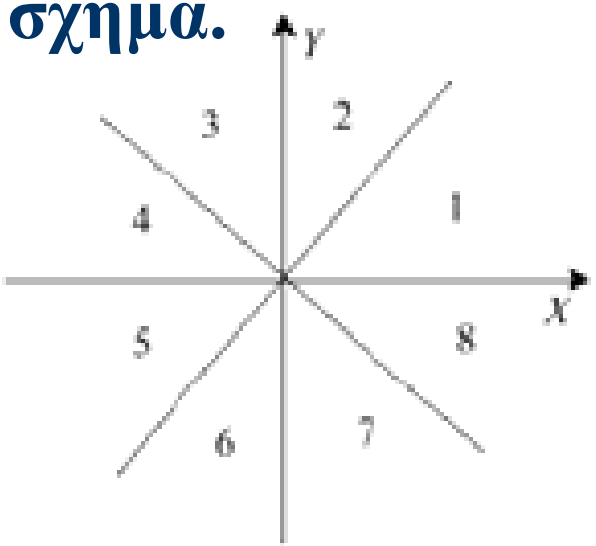
$$e \geq \frac{dx}{2}$$

- $\frac{dx}{2}$ : υπολογίζεται με ολίσθηση (bit-shift)

- Για καλύτερη απόδοση:
  - Αντικατάσταση της συνθήκης $e \geq \frac{dx}{2}$ με $e \geq 0$
  - Αρχική αφαίρεση $\frac{dx}{2}$ από $e$

# Περιορισμοί αλγορίθμου 4 1/3

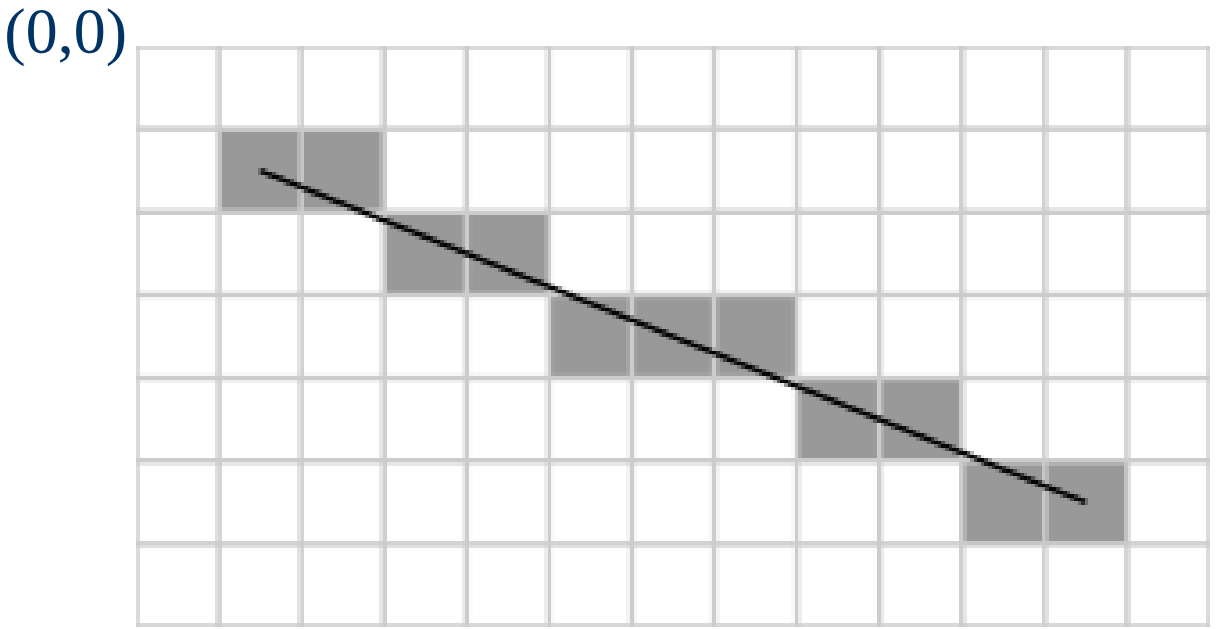
- Λειτουργεί μόνο στο 1ο οκταμόριο. Εύκολα μπορεί να επεκταθεί και στα άλλα με κατάλληλες εναλλαγές των ρόλων x, y, αρχικού και τελικού σημείου. **Άξονας ταχύτερης κίνησης και είδος μεταβολής του άλλου άξονα δίνονται στο σχήμα.**




| Οκταμόριο | Άξονας ταχυτ. Κίνησης | Άλλος άξονας |
|---|---|---|
| 1 | *x* | Αυξάνεται |
| 2 | *y* | » |
| 3 | *y* | Μειώνεται |
| 4 | *x* | » |
| 5 | *x* | Αυξάνεται |
| 6 | *y* | » |
| 7 | *y* | Μειώνεται |
| 8 | *x* | » |

# Περιορισμοί αλγορίθμου 4 2/3

- 8ο οκταμόριο:



# Περιορισμοί αλγορίθμου 4 3/3

- Ευθύγραμμα τμήματα διαφορετικών κλίσεων μπορεί να σχεδιαστούν με διαφορετική φωτεινότητα.
- Το φαινόμενο αυτό μπορεί να διορθωθεί με τεχνικές αντι-ταύτισης (anti-aliasing).

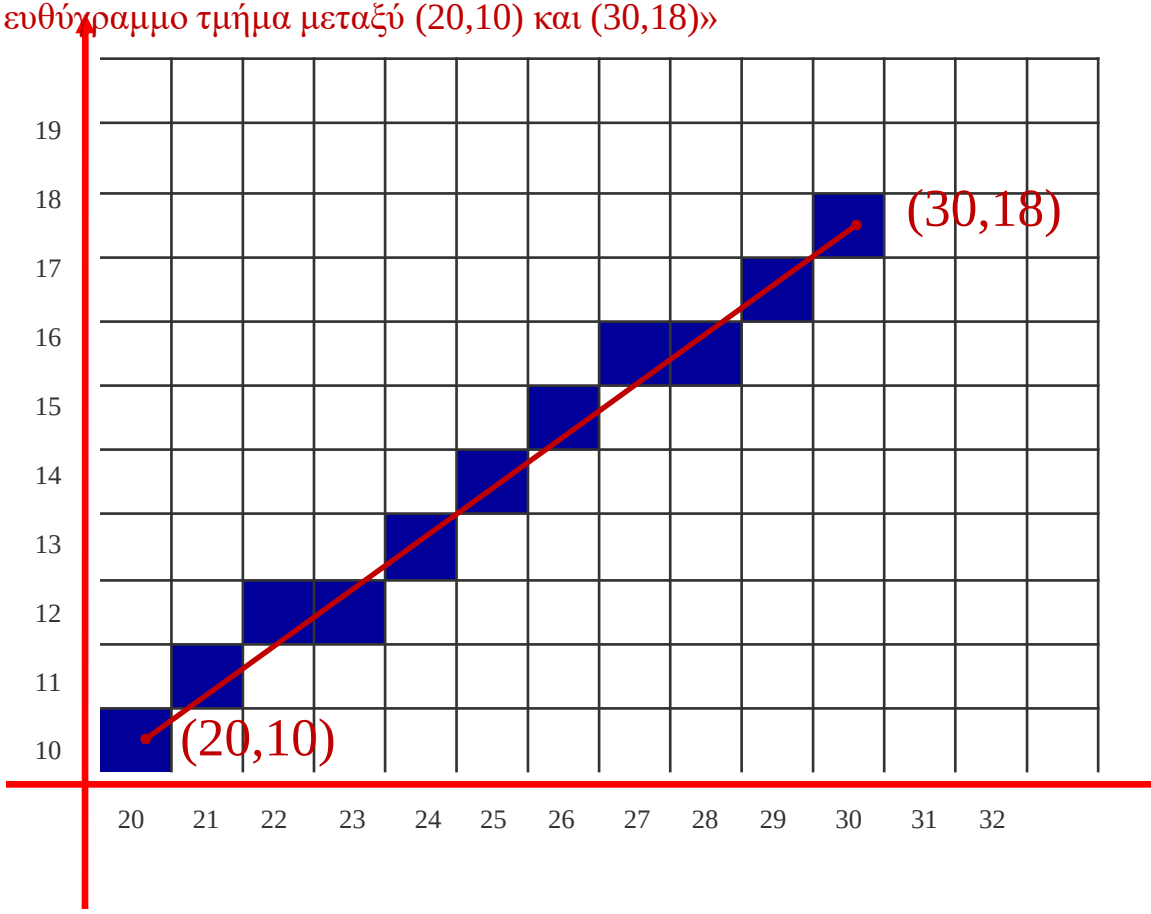
# Παράδειγμα Bresenham

*Αρχικές τιμές*

$dx = 10$
$dy = 8$
$error_0 = -5$
$x_0 = 20$
$y_0 = 10$

| $(x_{i+1}, y_{i+1})$ | error |
|---|---|
| (20,10) | 3 → -7 |
| (21,11) | 1 → -9 |
| (22,12) | -1 |
| (23,12) | 7 → -3 |
| (24,13) | 5 → -5 |
| (25,14) | 3 → -7 |
| (26,15) | 1 → -9 |
| (27,16) | -1 |
| (28,16) | 7 → -3 |
| (29,17) | 5 → -5 |
| (30,18) | 3 → -7 |

«Σχεδιάστε το ευθύγραμμο τμήμα μεταξύ (20,10) και (30,18)»




```
line4 (x1, y1, xn, yn, colour)
int x1, y1, xn, yn, colour;
{
     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, colour);
        error = error + dy;
        if (error >= 0){
             y++;
             error = error – dx
        }
     }
}
```

# Αλγόριθμος 4 (Bresenham)

- **Πλεονεκτήματα**:
  - ταχύτητα
  - απλότητα υλοποίησης
- Υλοποιημένος σε:
  - εκτυπωτές, plotters, …
  - κάρτες γραφικών (firmware ή hardware chip)
  - βιβλιοθήκες γραφικών (software)

# Αλγόριθμος 4 (Bresenham)

- **Ειδικές περιπτώσεις:**
  - Οριζόντιες γραμμές ($\Delta y = 0$)
  - Κάθετες γραμμές ($\Delta x = 0$)
  - Διαγώνιες γραμμές ($|\Delta x| = |\Delta y|$)

…οδηγούνται κατευθείαν στον καταχωρητή πλαισίου (frame-buffer) χωρίς να "περάσουν» από επεξεργασία μέσω των αλγορίθμων σχεδίασης.