

Γραφικά & Οπτικοποίηση

Κεφάλαιο 2

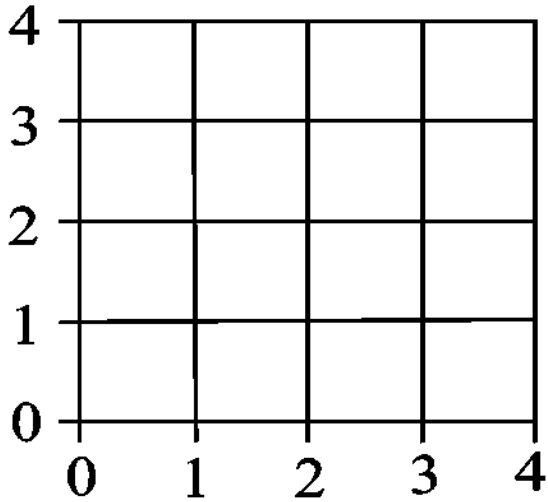
Αλγόριθμοι Σχεδίασης

Σχεδίαση

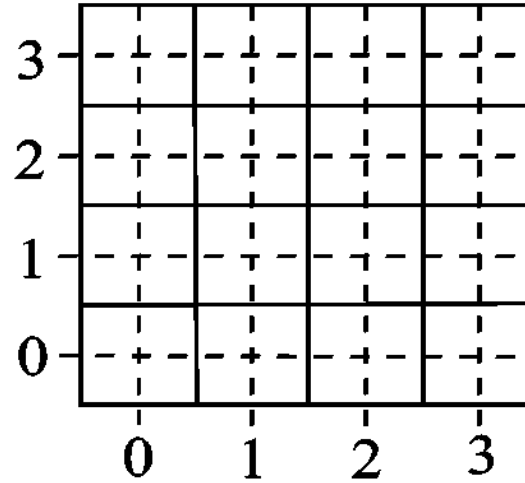
- 2Δ οθόνες αποτελούνται από διακριτά πλέγματα εικονοστοιχείων (pixels)
- **Σχεδίαση:** μετατροπή 2Δ στοιχειωδών σχημάτων σε διακριτή παράσταση εικονοστοιχείων
- Πολυπλοκότητα σχεδίασης: $O(Pp)$, όπου P είναι το πλήθος στοιχειωδών σχημάτων και p ο αριθμός των εικονοστοιχείων
- 2 τρόποι για να παραστήσουμε το πλέγμα των εικονοστοιχείων
 - Κέντρα σε ημίσειες συντεταγμένες
 - Κέντρα σε ακέραιες συντεταγμένες (προτεινόμενο)
- **Σύνδεση:** αναπαράσταση των γειτόνων ενός εικονοστοιχείου
 - Τετραπλή σύνδεση
 - Οκταπλή σύνδεση
- Προκλήσεις της σχεδίασης
 - Καθορισμός των εικονοστοιχείων που περιγράφουν το σχήμα ακριβώς
 - Αποδοτικότητα

Σχεδίαση (2)

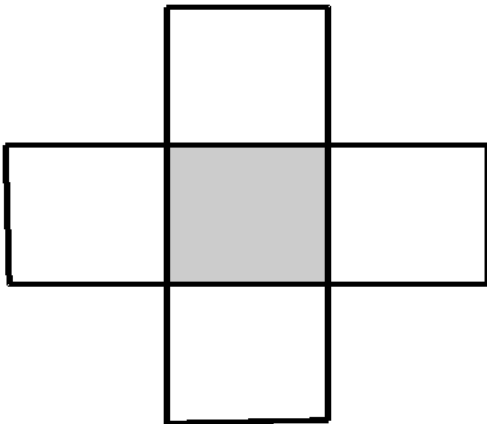
- Κέντρα σε ημίσειες συντεταγμένες



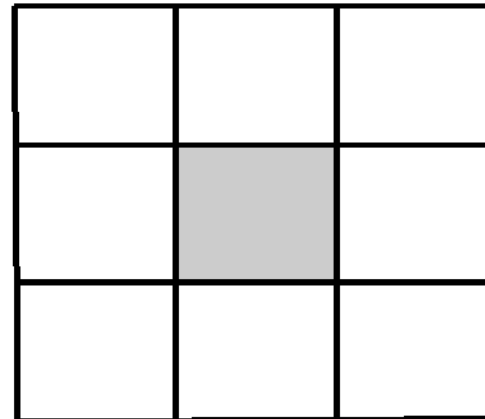
- Κέντρα σε ακέραιες συντεταγμένες



- Τετραπλή Σύνδεση



- Οκταπλή Σύνδεση



Μαθηματικές Καμπύλες

- Ορισμός

- **Πεπλεγμένη Μορφή:**

Π.χ. :

$$f(x, y) \begin{cases} < 0, & \text{το σημείο } (x,y) \text{ είναι εντός της καμπύλης} \\ = 0, & \text{το σημείο } (x,y) \text{ είναι πάνω στην καμπύλη} \\ > 0, & \text{το σημείο } (x,y) \text{ είναι εκτός της καμπύλης} \end{cases}$$

- **Παραμετρική Μορφή:**

- Συνάρτηση μιας παραμέτρου $t \in [0, 1]$
 - Το t αντιστοιχεί στο μήκος τόξου πάνω στην καμπύλη
 - Η καμπύλη ιχνηλατείται καθώς το t πηγαίνει από το 0 στο 1

Π.χ.: $l(t) = (x(t), y(t))$

Μαθηματικές Καμπύλες(2)

- Παραδείγματα
 - **Πεπλεγμένη μορφή:**
 - **Ευθ. τμήμα:** $l(x, y) \equiv ax + by + c = 0$
όπου a, b, c : συντελεστές ευθ. τμήματος
if $l(x, y) = 0$ τότε το (x, y) είναι πάνω στην καμπύλη
else if $l(x, y) < 0$ τότε το (x, y) είναι στο ένα ημι-επίπεδο
else if $l(x, y) > 0$ τότε το (x, y) είναι στο άλλο ημι-επίπεδο
 - **Κύκλος:** $c(x, y) \equiv (x - x_c)^2 + (y - y_c)^2 - r^2 = 0$
όπου (x_c, y_c) : το κέντρο & r : η ακτίνα
if $c(x, y) = 0$ τότε το (x, y) είναι πάνω στον κύκλο
else if $c(x, y) < 0$ τότε το (x, y) είναι εντός του κύκλου
else if $c(x, y) > 0$ τότε το (x, y) είναι εκτός του κύκλου

Μαθηματικές Καμπύλες(3)

- Παραδείγματα:
 - **Παραμετρική Μορφή:**
 - **Ευθ. τμήμα:** $\mathbf{l}(t) = (x(t), y(t))$

$$\text{όπου } x(t) = x_1 + t(x_2 - x_1) ,$$

$$y(t) = y_1 + t(y_2 - y_1) ,$$

$$t \in [0,1]$$

- **Κύκλος:** $\mathbf{c}(t) = (x(t), y(t))$

$$\text{όπου } x(t) = x_c + r \cos(2\pi t) ,$$

$$y(t) = y_c + r \sin(2\pi t),$$

$$t \in [0,1]$$

Πεπερασμένες Διαφορές

- Οι συναρτήσεις που ορίζουν στοιχειώδη σχήματα πρέπει να εκτιμούνται για κάθε εικονοστοιχείο \Rightarrow ακριβό
- Παίρνοντας υπόψη τις πεπερασμένες διαφορές γλιτώνουμε κόστος
- Εμπροσθεν διαφορές (fd):
 - ◆ Πρώτες(fd) : $\delta f_i = f_{i+1} - f_i$
 - ◆ Δεύτερες (fd): $\delta^2 f_i = \delta f_{i+1} - \delta f_i$
 - ◆ k^{th} (fd): $\delta^k f_i = \delta^{k-1} f_{i+1} - \delta^{k-1} f_i$
- Πεπλεγμένες συναρτήσεις χρησιμοποιούνται για τον έλεγχο αν ένα εικονοστοιχείο ανήκει ή όχι στο σχήμα
π.χ.: το εικονοστοιχείο (x, y) είναι εντός αν $|f(x, y)| < e$,
όπου e : σχετίζεται με το πάχος της ευθείας

Πεπερασμένες Διαφορές(2)

- Παραδείγματα:

- Εκτίμηση της συνάρτησης ευθ. τμήματος αυξητικά:

→ από το (x, y) στο $(x+1, y)$

Υπολογισμός των έμπροσθεν πεπερασμένων διαφορών της πεπλεγμένης συνάρτησης ευθ. τμήματος στην κατεύθυνση του x - άξονα από

το εικονοστοιχείο x στο $x+1$: $\delta_x l(x, y) = l(x+1, y) - l(x, y) = a$

$$l(x, y) + \delta_x l(x, y) = l(x, y) + a$$

→ από το (x, y) στο $(x, y+1)$

Υπολογισμός των έμπροσθεν πεπερασμένων διαφορών της πεπλεγμένης συνάρτησης ευθ. τμήματος στην κατεύθυνση του y - άξονα από

το εικονοστοιχείο y στο $y+1$: $\delta_y l(x, y) = l(x, y+1) - l(x, y) = b$

$$l(x, y) + \delta_y l(x, y) = l(x, y) + b$$

Πεπερασμένες Διαφορές(3)

- Παραδείγματα:
 - Εκτίμηση της συνάρτησης κύκλου αυξητικά:
→ από το (x, y) στο $(x+1, y)$

Υπολογισμός των έμπροσθεν πεπερασμένων διαφορών της πεπλεγμένης συνάρτησης κύκλου. Επειδή είναι δευτέρου βαθμού έχει 2 πεπερασμένες διαφορές στη x-κατεύθυνση από το x στο x+1:

$$\delta_x c(x, y) = c(x+1, y) - c(x, y) = 2(x - x_c) + 1$$

$$\delta_x^2 c(x, y) = \delta_x c(x+1, y) - \delta_x c(x, y) = 2$$

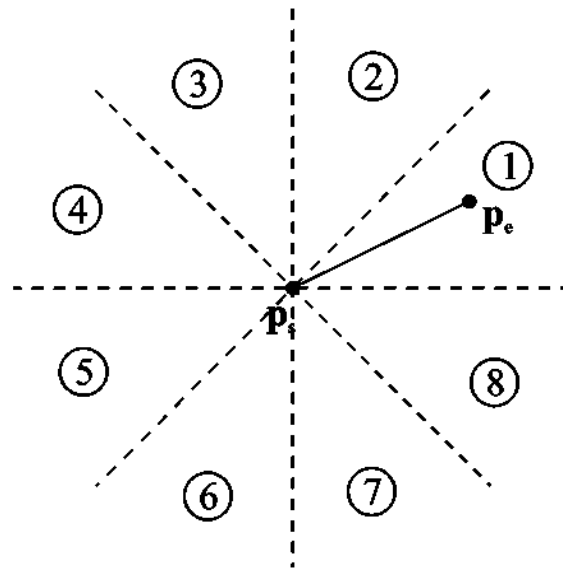
Υπολόγισε $\delta_x c(x, y) = \delta_x c(x-1, y) + \delta_x^2 c(x, y)$

$$c(x+1, y) = c(x, y) + \delta_x c(x, y)$$

→ από το (x, y) στο $(x, y+1)$ είναι παρόμοια

Αλγόριθμοι Σχεδίασης Ευθύγραμμου Τμήματος

- Επιθυμητές ιδιότητες ενός αλγορίθμου σχεδίασης ευθ. τμημάτων:
 - Τα εικονοστοιχεία 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 Σχεδίασης Ευθύγραμμου Τμήματος 1

- Έστω ευθύγραμμο τμήμα μεταξύ των εικονοστοιχείων $p_s = (x_s, y_s)$ και $p_e = (x_e, y_e)$ στο 1^ο οκταμόριο

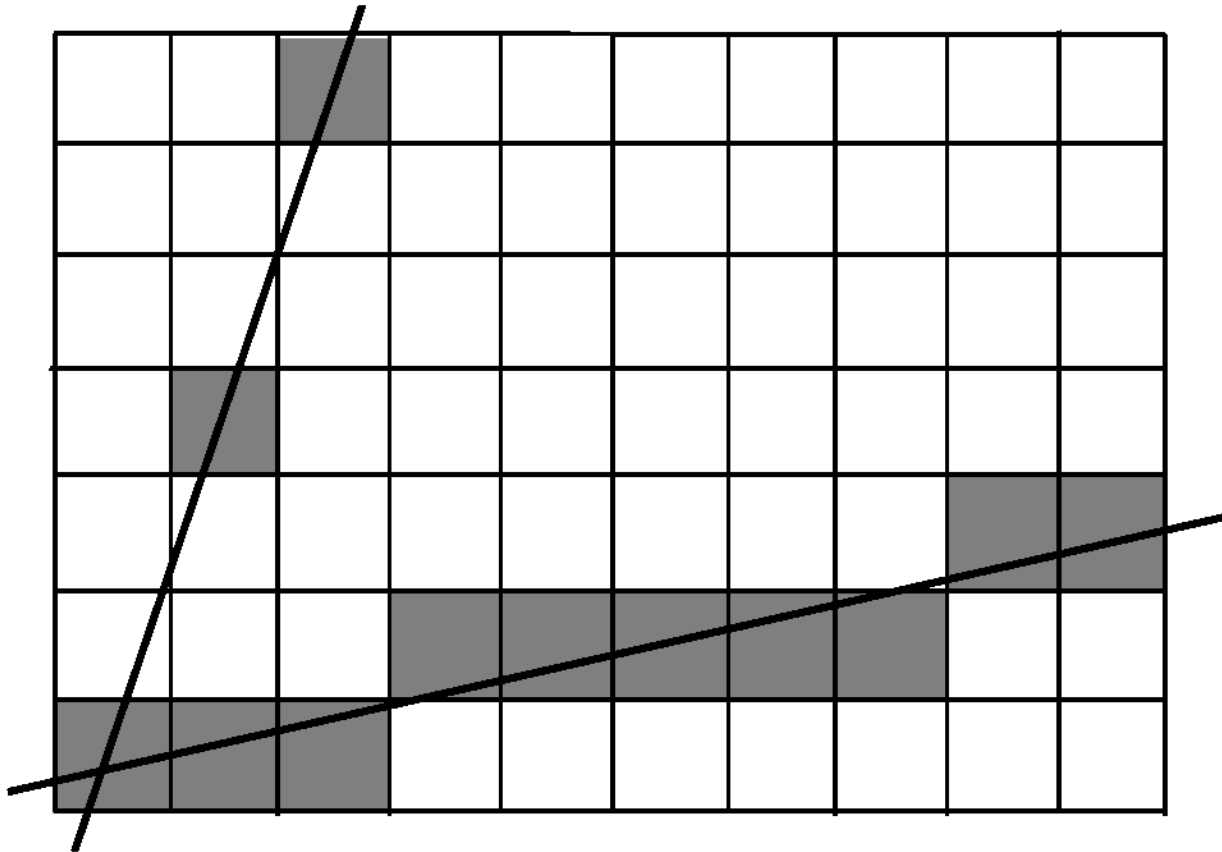
- Κλίση ευθείας: $s = \frac{y_e - y_s}{x_e - x_s}$, $y = y_s + \text{round}(s \cdot (x - x_s))$, $x = x_s, \dots, x_e$

Αλγόριθμος:

```
line1 ( int xs, int ys, int xe, int ye, colour c ) {  
    float s; int x, y;  
    s = (ye - ys) / (xe - xs); (x, y) = (xs, ys);  
    while (x <= xe) {  
        setpixel (x, y, c);  
        x = x + 1;  
        y = ys + round(s * (x - xs));  
    }  
}
```

Αλγόριθμος Σχεδίασης Ευθύγραμμου Τμήματος 1(2)

- Παράδειγμα του line1 στο πρώτο και δεύτερο οκταμόριο:



Αλγόριθμός Σχεδίασης Ευθύγραμμου Τμήματος 2

- Αποφυγή στρογγυλοποιήσεων: διαχωρισμός του y σε ακέραιο και δεκαδικό μέρος e
- Υπολογισμός κάθε τιμής αυξητικά

Αλγόριθμος:

```
line2 ( int xs, int ys, int xe, int ye, colour c ) {  
    float s, e; int x, y;  
    e = 0;  s = (ye - ys) / (xe - xs);  (x, y) = (xs, ys);  
    while (x <= xe) {  
        /* -1/2 <= e < 1/2 */  
        setpixel(x, y, c);  
        x = x + 1;  
        e = e + s;  
        if (e >= 1/2) {  
            y = y + 1;  
            e = e - 1;  
        }  
    }  
}
```

Αλγόριθμος Σχεδίασης Ευθύγραμμου Τμήματος 2(2)

- Ο line2 μοιάζει με τον υπολογισμό του δίσεκτου έτους
 - Σε κάθε επανάληψη, η κλίση προστίθεται στην e
 - Όταν η e είναι μεγαλύτερη της μισής μονάδας, η ευθεία μεταβαίνει στο επόμενο εικονοστοιχείο
 - Το y αυξάνει και το e μειώνεται αντίστοιχα, ώστε το άθροισμα τους να παραμένει σταθερό
- Κάθε έτος έχει 365,25 μέρες αλλά το ημερολογιακό έτος έχει ακέραιο πλήθος ημερών
- Κάθε τέσσερα έτη δημιουργείται σφάλμα της τάξεως της μίας πλήρους ημέρας
- Για αυτό κάθε 4 χρόνια προστίθεται μια ημέρα στο έτος ώστε το σφάλμα να 'απορροφάται'

Αλγόριθμος Bresenham

- Αντικατάσταση πραγματικών μεταβλητών του line2 από ακέραιες
- Πολλαπλασιάζουμε με $dx = x_e - x_s \rightarrow s$ και e γίνονται ακέραιοι
- Η συνθήκη για επιλογή επόμενου εικονοστοιχείου γίνεται

$$e \geq \left\lfloor \frac{dx}{2} \right\rfloor$$

- $\left\lfloor \frac{dx}{2} \right\rfloor$: υπολογίζεται με ολίσθηση

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

Αλγόριθμος Bresenham (2)

- Αντικατάσταση πραγματικών μεταβλητών από ακέραιες

Αλγόριθμος

```
line3 ( int xs, int ys, int xe, int ye, colour c ) {  
    int x, y, e, dx, dy;  
    dx = (xe - xs);  dy=(ye - ys);  e = - (dx >> 1);  (x, y)=(xs, ys);  
    while (x <= xe) {  
        /* -dx <= e < 0 */  
        setpixel(x, y, c);  
        x = x + 1;  
        e = e + dy;  
        if (e >= 0) {  
            y = y + 1;  
            e = e - dx;  
        }  
    }  
}
```


Αλγόριθμος Bresenham (3)

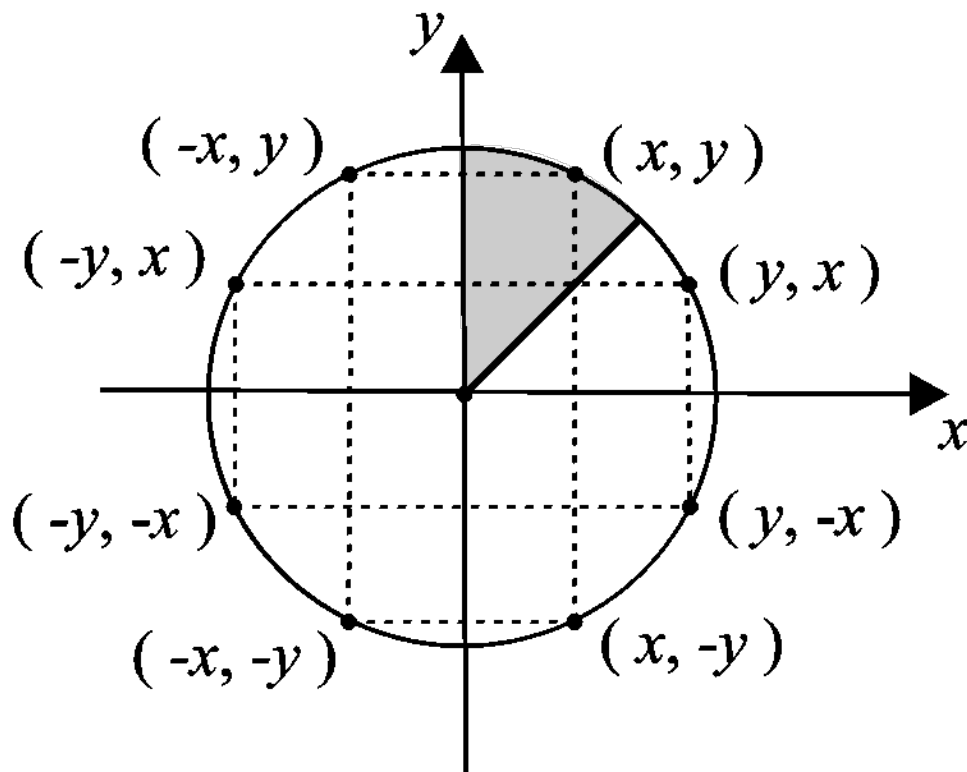
- Ο line3 λειτουργεί μόνο στο 1^ο οκταμόριο
- Για τα υπόλοιπα οκταμόρια χρειάζονται οι παρακάτω αλλαγές

Οκταμόριο	Άξονας ταχ. κίνησης	Άλλος άξονας
1	x	Αυξάνεται
2	y	Αυξάνεται
3	y	Μειώνεται
4	x	Αυξάνεται
5	x	Μειώνεται
6	y	Μειώνεται
7	y	Αυξάνεται
8	x	Μειώνεται

- Ο αλγόριθμος Bresenham καλύπτει τις απαιτήσεις ενός αποδοτικού αλγορίθμου σχεδίασης

Σχεδίαση Κύκλου

- 8-πλη συμμετρία
- Υπολογισμός ενός οκταμορίου
- Τα υπόλοιπα οκταμόρια υπολογίζονται με βάση τη συμμετρία



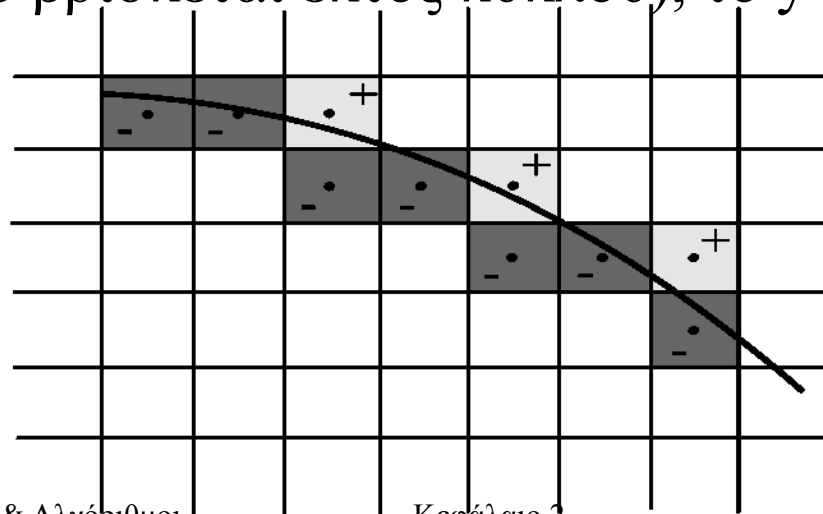
Αλγόριθμος Σχεδίασης Κύκλου

- Σχεδιασμός των 8 συμμετρικών σημείων του κύκλου

```
set8pixels ( int x, y, colour c ) {  
    setpixel(x, y, c);  
    setpixel(y, x, c);  
    setpixel(y, -x, c);  
    setpixel(x, -y, c);  
    setpixel(-x, -y, c);  
    setpixel(-y, -x, c);  
    setpixel(-y, x, c);  
    setpixel(-x, y, c);  
}
```

Αλγόριθμος Bresenham για Κύκλο

- r : η ακτίνα του κύκλου
- Το κέντρο του είναι το εικονοστοιχείο $(0, 0)$
- Ο αλγόριθμος ξεκινάει από το εικονοστοιχείο $(0, r)$ και σχεδιάζει *μόλις κάτω* από το τόξο του κύκλου
- Σχεδιάζει ένα κυκλικό τόξο στο 2° οκταμόριο
- Η μεταβλητή x αυξάνεται σε κάθε βήμα
- Όταν η τιμή της συνάρτησης του κύκλου γίνει θετική (το εικονοστοιχείο βρίσκεται εκτός κύκλου), το y μειώνεται



Αλγόριθμος Bresenham για Κύκλο (2)

- Αλλαγή κέντρου σε $(0, \frac{1}{2})$ για επιλογή εικονοστοιχείων πάνω στο τόξο του κύκλου

$$c(x, y) = x^2 + (y - \frac{1}{2})^2 - r^2 = 0$$

- Αρχικοποίηση της μεταβλητής λάθους σε:

$$c(0, r) = (r - \frac{1}{2})^2 - r^2 = \frac{1}{4} - r$$

- Το $\frac{1}{4}$ παραλείπεται αφού η μεταβλητή είναι ακέραια
- Για την αύξηση του e χρησιμοποιούμε τις πεπερασμένες διαφορές της συνάρτησης κύκλου για τα 2 πιθανά βήματα του αλγορίθμου:

$$c(x+1, y) - c(x, y) = (x+1)^2 - x^2 = 2x+1$$

$$c(x, y-1) - c(x, y) = (y - \frac{3}{2})^2 - (y - \frac{1}{2})^2 = -2y+2$$

Αλγόριθμος Bresenham για Κύκλο (3)

Αλγόριθμος:

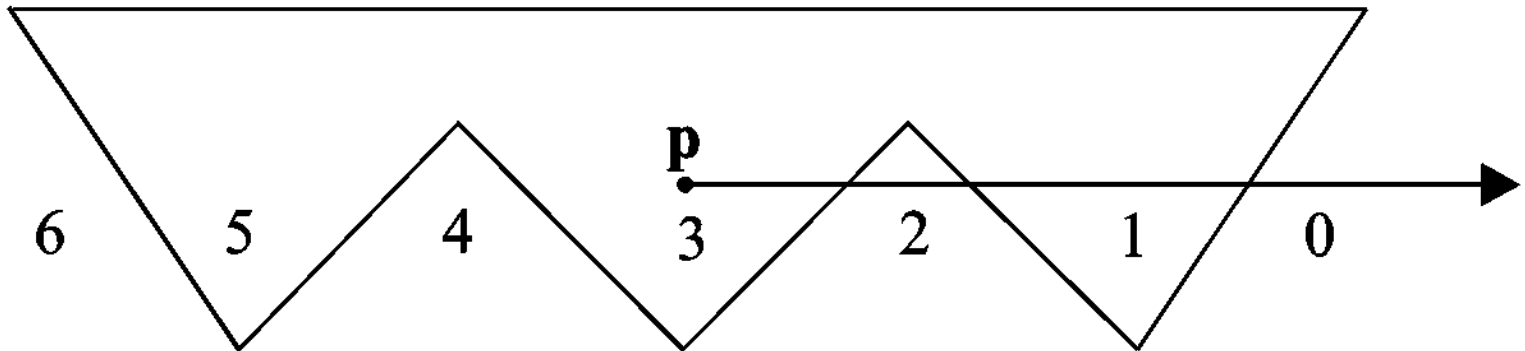
```
circle ( int r, colour c ) {  
    int x, y, e;  
    x = 0;  y = r;  e = - r;  
    while (x <= y) {  
        /* e == x^2 + (y - 1/2)^2 - r^2 */  
        set8pixels(x, y, c);  
        e = e + 2 * x + 1;  
        x = x + 1;  
        if (e >= 0) {  
            e = e - 2 * y + 2;  
            y = y - 1;  
        }  
    }  
}
```

Έλεγχοι Εσωτερικού Σημείου

- Πολύγωνο $\left. \begin{array}{l} n \text{ κορυφές } (v_0, \dots, v_{n-1}) \\ n \text{ ακμές} \end{array} \right\}$ σχηματίζουν μια κλειστή καμπύλη $v_0, v_1, \dots, v_{n-1}, v_0$
- **Θεώρημα Καμπύλης Jordan :** Μια συνεχής κλειστή καμπύλη στο επίπεδο, χωρίζει το επίπεδο σε 2 περιοχές, την ‘εσωτερική’ και την ‘εξωτερική’
- Για τη σχεδίαση χρειάζεται να γνωρίζουμε αν ένα εικονοστοιχείο p βρίσκεται εντός ενός πολυγώνου P . Θα δούμε 2 ελέγχους:
 - Έλεγχος Ισοτιμίας (parity test)
 - Έλεγχος Αριθμού Περιελίξεων (winding number test)

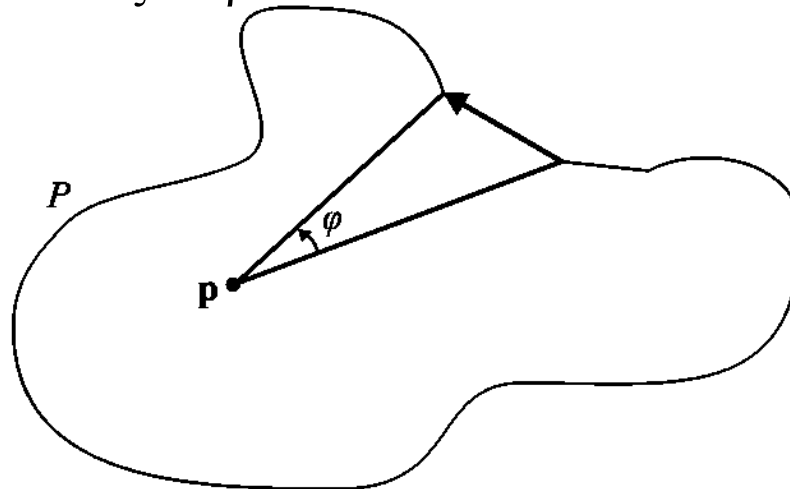
Έλεγχοι Εσωτερικού Σημείου (2)

- Έλεγχος ισοτιμίας (parity test):
 - Σχεδιάζουμε μια ημιευθεία από το εικονοστοιχείο p
 - Μετρούμε τον αριθμό των τομών της ημιευθείας με την περίμετρο του P
 - Αν $\#$ τομών $==$ περιττός, τότε το p είναι εσωτερικό του P
 - Αλλιώς το p είναι εξωτερικό του P



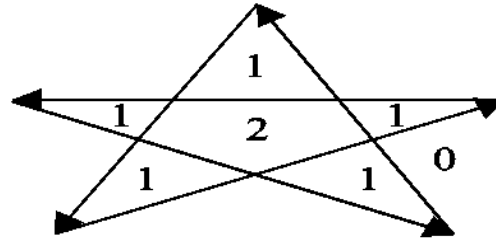
Έλεγχοι Εσωτερικού Σημείου (3)

- Έλεγχος Αριθμού Περιελίξεων (Winding Number Test):
 - $\omega(P, p)$ μετράει το # των πλήρων περιστροφών μιας ακτίνας από το p που ιχνηλατεί το P
$$\omega(P, p) = \frac{1}{2\pi} \int d\varphi$$
 - Για κάθε δεξιόστροφη περιστροφή: $\omega(P, p) ++$
 - Για κάθε αριστερόστροφη περιστροφή: $\omega(P, p) --$
 - Αν $\omega(P, p) ==$ περιττός τότε το p είναι εσωτερικό του P
 - Αλλιώς το p είναι εξωτερικό του P

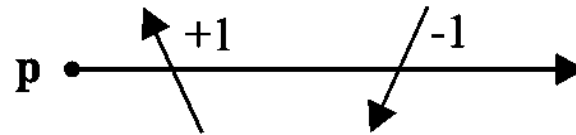


Έλεγχοι Εσωτερικού Σημείου (4)

- Π.χ. έλεγχου αριθμού περιελίξεων για εσωτερικό σημείο πολυγώνου:

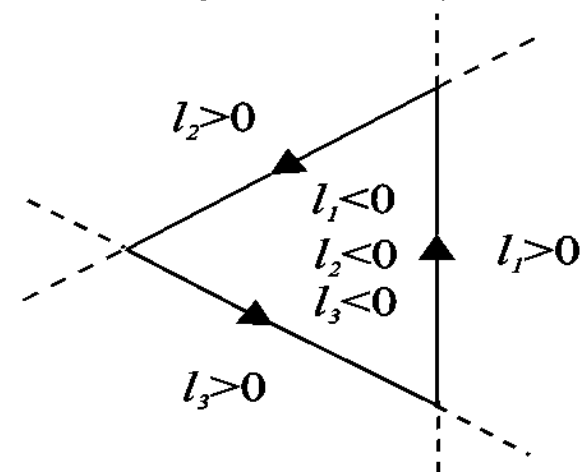


- Απλός υπολογισμός του αριθμού περιελίξεων:



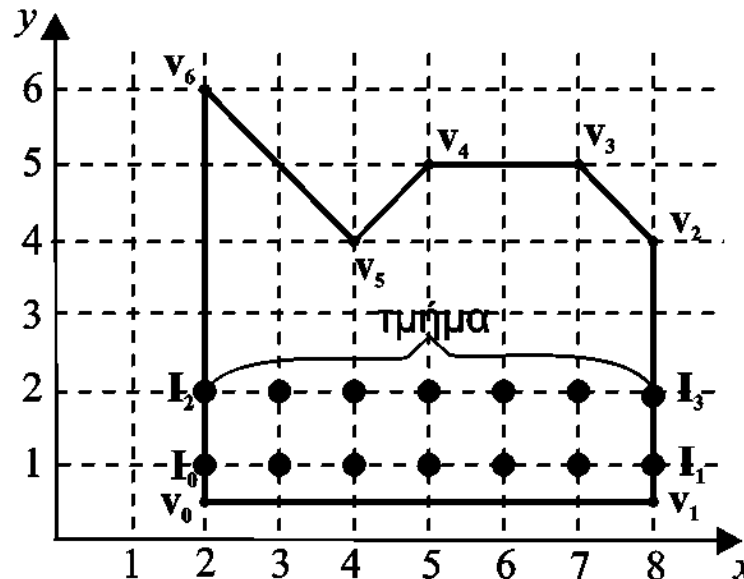
- Έλεγχος πρόσημου για ένα σημείο μέσα σε κυρτό πολύγωνο:

$$\text{sign}(l_0(\mathbf{p})) = \text{sign}(l_1(\mathbf{p})) = \dots = \text{sign}(l_{n-1}(\mathbf{p}))$$



Βασικός Αλγόριθμος Σχεδίασης Πολυγώνου

- Βασικός Αλγόριθμος Σχεδίασης Πολυγώνου:
 - βασίζεται στον έλεγχο ισοτιμίας
 - βήματα:
 1. Υπολόγισε τις τομές $I(x, y)$ κάθε ακμής με όλες τις γραμμές σάρωσης και αποθήκευσέ τις σε μια λίστα
 2. Ταξινόμησε τις τομές με βάση τα (y, x)
 3. Εξαγωγή ζευγών από τη λίστα & γέμισμα των εικονοστοιχείων μεταξύ τους

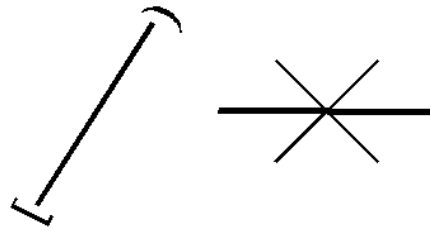


Ιδιόζουσες Περιπτώσεις

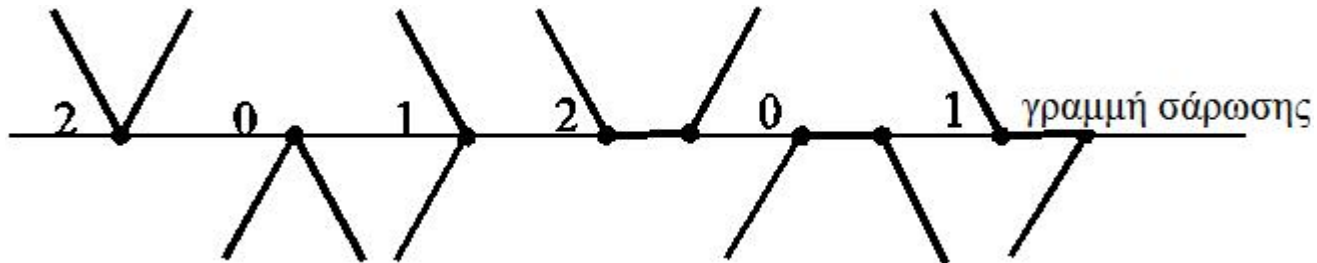
- Πρόβλημα:
 - αν μια κορυφή του πολυγώνου βρίσκεται ακριβώς πάνω σε μια γραμμή σάρωσης οι τομές θα είναι 2, 1 ή 0 ;
- Λύσεις:
 - Θεώρησε την ακμή κλειστή στην κορυφή με το ελάχιστο y και ανοιχτή στην κορυφή με το μέγιστο y
 - Αγνόησε τις οριζόντιες ακμές
- Βασικός Αλγόριθμος Σχεδίασης Πολυγώνου:
 - Αναποτελεσματικός λόγω του κόστους των υπολογισμών για την εύρεση των τομών

Ιδιάζουσες Περιπτώσεις (2)

- Κανόνας για τις ιδιάζουσες περιπτώσεις:

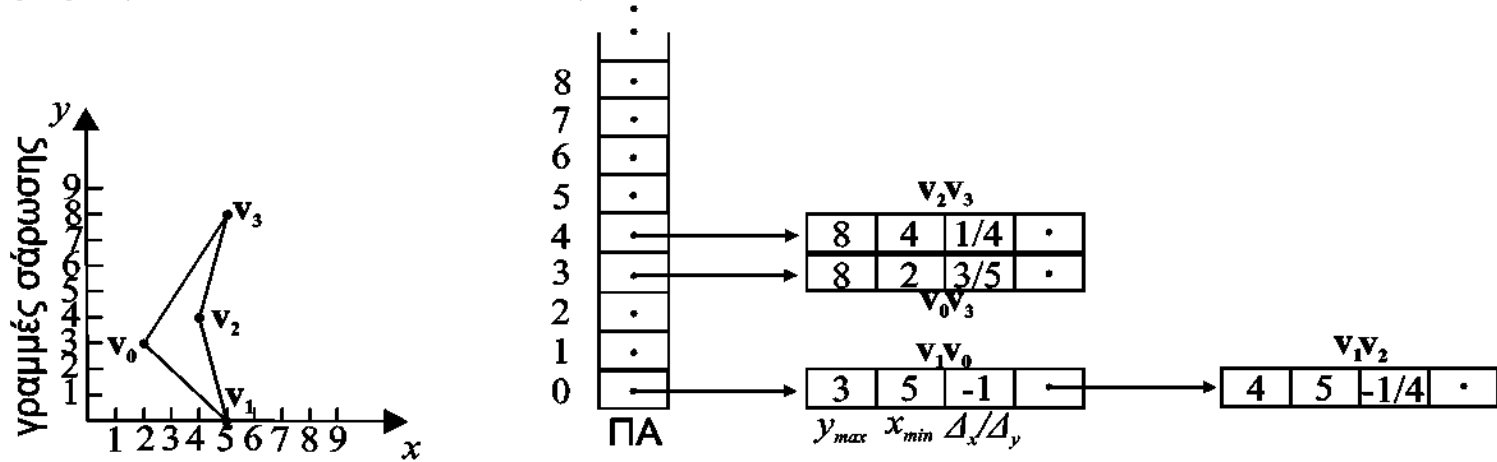


- Αποτέλεσμα του κανόνα για τις ιδιάζουσες περιπτώσεις:

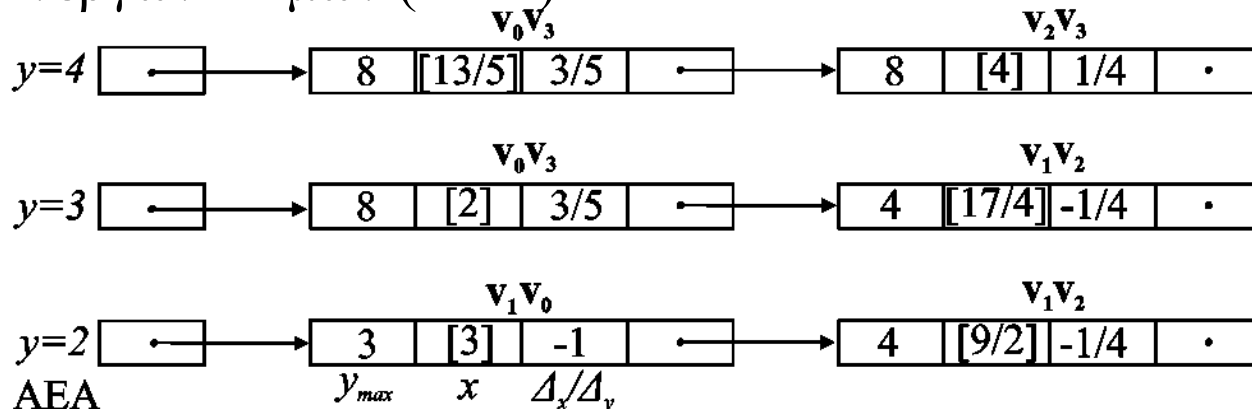


Αλγόριθμος Σχεδίασης Πολυγώνων με ET

- Εκμεταλλεύεται τη συνάφεια μεταξύ γραμμών σάρωσης & τη συνάφεια των ακμών
- Χρησιμοποιεί Πίνακα Ακμών (ΠΑ) και



- Λίστα Ενεργών Ακμών (ΛΕΑ)



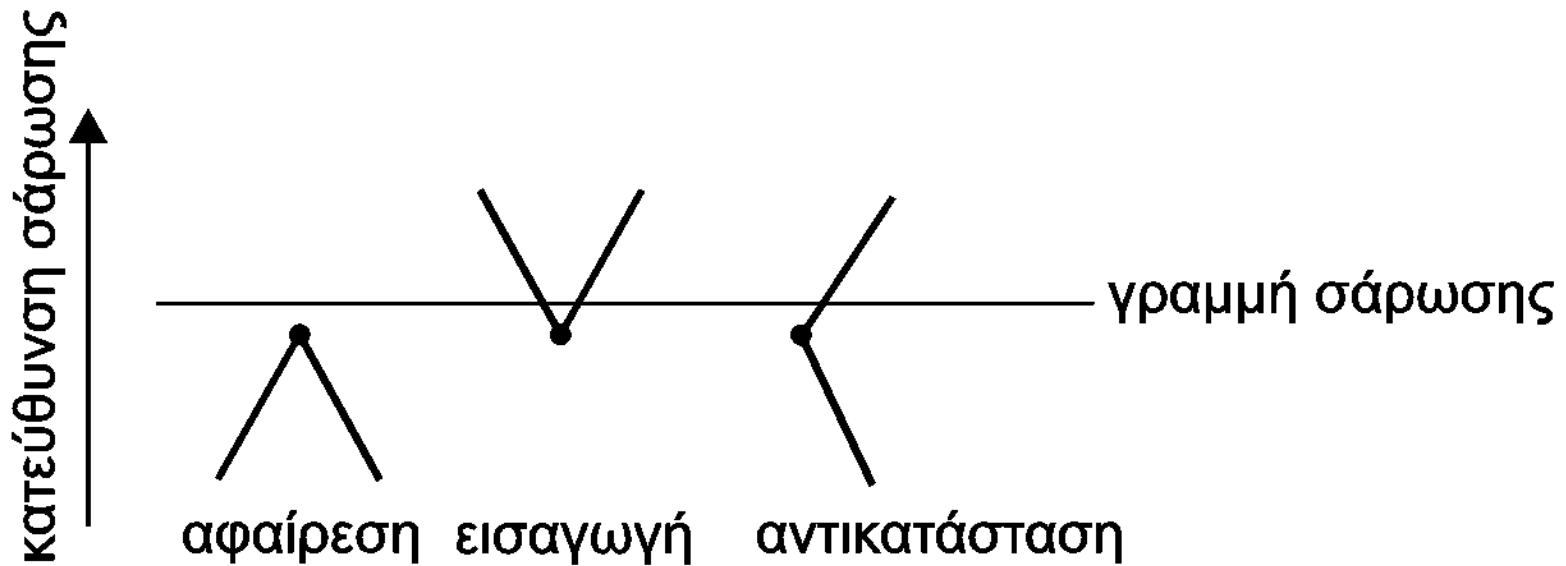
Αλγόριθμος Σχεδίασης Πολυγώνων με ΠΑ (2)

Αλγόριθμος:

1. Δημιούργησε τον ΠΑ για το πολύγωνο. Αυτός περιέχει για κάθε ακμή το μέγιστο y , το ελάχιστο x και την αντίστροφη κλίση $(y_{\max}, x_{\min}, 1/s)$. Η εγγραφή μιας πλευράς εισάγεται στη λίστα του ΠΑ που αντιστοιχεί στην ελάχιστη y τιμή της.
2. Για κάθε γραμμή σάρωσης y που τέμνει το πολύγωνο (από κάτω προς τα επάνω) επανέλαβε
 - (a) Ενημέρωσε τις τομές στην ΛΕΑ για την τρέχουσα γραμμή σάρωσης:
$$x = x + 1/s$$
 - (b) Εισήγαγε ακμές από το y -bucket του ΠΑ στην ΛΕΑ
 - (c) Αφαίρεσε από την ΛΕΑ τις ακμές με $y_{\max} \leq y$
 - (d) Ταξινόμησε πάλι την ΛΕΑ κατά x
 - (e) Για κάθε ζεύγος ακμών της ΛΕΑ, χρωμάτισε τα εικονοστοιχεία μεταξύ των x τιμών του

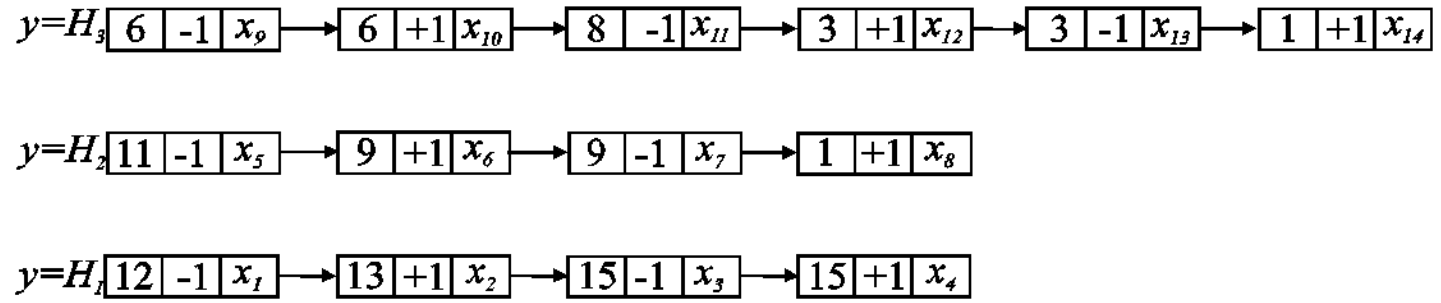
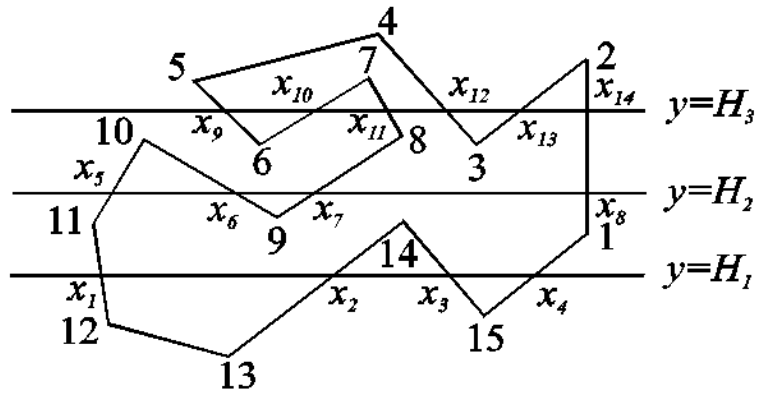
Αλγόριθμος Σχεδίασης Πολυγώνων με ΠΑ (3)

- Οι ακμές της ΛΕΑ αλλάζουν σε κορυφές σύμφωνα με την παρακάτω εικόνα:



Αλγόριθμος Κρίσιμων Σημείων για Σχεδίαση Πολυγώνου

- Χρησιμοποιεί τα τοπικά ελάχιστα (κρίσιμα σημεία) για αποφυγή του ΠΑ
- Παράδειγμα πολυγώνου και των περιεχομένων της ΛΕΑ για 3 γραμμές σάρωσης



Αλγόριθμος Κρίσιμων Σημείων για Σχεδίαση Πολυγώνου (2)

Αλγόριθμος:

1. Βρες και αποθήκευσε τα κρίσιμα σημεία του πολυγώνου
2. Για κάθε γραμμή σάρωσης y που τέμνει το πολύγωνο από κάτω προς τα πάνω
 - (a) Για κάθε κρίσιμο σημείο $c(c_x, c_y) \mid (y-1 < c_y \leq y)$ ιχνηλάτησε την περίμετρο του πολυγώνου και προς τις 2 κατευθύνσεις. Σταμάτησε όταν συναντήσεις τη γραμμή σάρωσης y ή ένα τοπικό μέγιστο. Για κάθε τομή με την y φύλαξε στην *ΛΕΑ* τις τιμές $(v, \pm 1, x)$: (αριθμός κορυφής εκκίνησης, δείκτης κατεύθυνσης: -1 ή $+1$ ανάλογα αν είναι αριστερόστροφα ή δεξιόστροφα, x -συντεταγμένη του σημείου τομής).
 - (b) Για κάθε εγγραφή της *ΛΕΑ* που προϋπήρχε του βήματος (a), ιχνηλατούμε την περίμετρο σύμφωνα με το δείκτη κατεύθυνσης της εγγραφής. Αν βρεθεί τομή με τη γραμμή σάρωσης y , ανανέωσε την εγγραφή με τον αριθμό κορυφής και τη x -συντεταγμένη της νέας τομής. Αν βρεθεί ένα τοπικό μέγιστο, διέγραψε την εγγραφή.
 - (c) Ταξιόμησε την *ΛΕΑ* κατά x -συντεταγμένη αν χρειάζεται.
 - (d) Για κάθε ζεύγος ακμών της *ΛΕΑ*, χρωμάτισε τα εικονοστοιχεία μεταξύ τους.

Αλγόριθμος Σχεδίασης Τριγώνου

- **Τρίγωνο:** το απλούστερο, επίπεδο, κυρτό πολύγωνο
- Καθορισμός των εικονοστοιχείων που καλύπτει ένα τρίγωνο → εκτέλεση ελέγχου εσωτερικού σημείου σε όλα τα εικονοστοιχεία που βρίσκονται μέσα στο περιβάλλον κιβώτιο (bounding box) του τριγώνου
- Ο έλεγχος εσωτερικού σημείου μπορεί να είναι η εκτίμηση των 3 γραμμικών συναρτήσεων που ορίζονται από τις κορυφές του
- Για κάθε εικονοστοιχείο p του περιβάλλοντος κιβωτίου, αν οι τιμές και των 3 συναρτήσεων έχουν το ίδιο πρόσημο, τότε το p είναι εντός του τριγώνου, αλλιώς είναι έκτος
- Για μεγαλύτερη απόδοση, οι συναρτήσεις υπολογίζονται με βάση τις πεπερασμένες διαφορές.

Αλγόριθμος Σχεδίασης Τριγώνου(2)

Αλγόριθμος:

```
triangle1 ( vertex v0, v1, v2, colour c ) {
```

```
  line l0, l1, l2;
```

```
  float e0, e1, e2, e0t, e1t, e2t;
```

```
  /* Υπολογισμός των συντελεστών ευθείας (a, b, c) από τις κορυφές */
```

```
  mkline(v0, v1, &l0); mkline(v1, v2, &l1); mkline(v2, v0, &l2);
```

```
  /* Υπολογισμός του περιβάλλοντος κιβωτίου του τριγώνου */
```

```
  bb_xmin = min(v0.x, v1.x, v2.x);
```

```
  bb_xmax = max(v0.x, v1.x, v2.x);
```

```
  bb_ymin = min(v0.y, v1.y, v2.y);
```

```
  bb_ymax = max(v0.y, v1.y, v2.y);
```

```
  /* Εκτίμηση των γραμμικών συναρτήσεων στο σημείο(bb_xmin, bb_ymin) */
```

```
  e0 = l0.a * bb_xmin + l0.b * bb_ymin + l0.c;
```

```
  e1 = l1.a * bb_xmin + l1.b * bb_ymin + l1.c;
```

```
  e2 = l2.a * bb_xmin + l2.b * bb_ymin + l2.c;
```

Αλγόριθμος Σχεδίασης Τριγώνου (3)

Αλγόριθμος (συνέχεια):

```
for (y=bb_ymin; y<=bb_ymax; y++) {  
    e0t = e0; e1t = e1; e2t = e2;  
    for (x=bb_xmin; x<=bb_xmax; x++) {  
        if (sign(e0)==sign(e1)==sign(e2))  
            setpixel(x,y,c);  
        e0 = e0 + l0.a;  
        e1 = e1 + l1.a;  
        e2 = e2 + l2.a;  
    }  
    e0 = e0t + l0.b;  
    e1 = e1t + l1.b;  
    e2 = e2t + l2.b;  
}
```

Αλγόριθμος Σχεδίασης Τριγώνου (4)

- Αν το περιβάλλον κιβώτιο είναι μεγάλο τότε ο αλγόριθμος `triangle1` είναι πολυδάπανος
- Εναλλακτική προσέγγιση: **Περίπατος Ακμών**
 - Χρήση 3 αλγόριθμων σχεδίασης ευθυγράμμων τμημάτων Bresenham για ιχνηλάτηση των ακμών του τριγώνου
 - Λειτουργία κατά γραμμή σάρωσης με συγχρονισμό των Bresenham
 - Για κάθε γραμμή σάρωσης που τέμνει το τρίγωνο υπολόγισε τις 2 τομές (Bresenham) και χρωμάτισε τα μεταξύ τους εικονοστοιχεία
 - Προσοχή στις ειδικές περιπτώσεις
- Οι παραπάνω αλγόριθμοι είναι κατάλληλοι για υλοποίηση σε υλικό λόγω απλότητας

Αλγόριθμοι Γεμίσματος

- Μια απλή προσέγγιση είναι το **γέμισμα (flood fill)**

Αλγόριθμος:

```
flood_fill ( polygon P, colour c ) {
```

```
  point s;
```

```
  draw_perimeter ( P, c );
```

```
  s = get_seed_point ( P );
```

```
  flood_fill_recur ( s, c );
```

```
}
```

```
flood_fill_recur ( point (x,y), colour fill_colour ); {
```

```
  colour c;
```

```
  c = getpixel(x,y); /* διάβασε το χρώμα του τρέχοντος εικονοστοιχείου */
```

```
  if (c != fill_colour) {
```

```
    setpixel(x,y,fill_colour);
```

```
    flood_fill_recur((x+1,y), fill_colour ); flood_fill_recur((x-1,y),fill_colour );
```

```
    flood_fill_recur((x,y+1), fill_colour ); flood_fill_recur((x,y-1), fill_colour );
```

```
  }
```

```
}
```

Αλγόριθμοι Γεμίσματος (2)

- Για περιοχές με τετραπλή σύνδεση οι παραπάνω 4 αναδρομικές κλήσεις είναι επαρκείς
- Για περιοχές με οκταπλή σύνδεση χρειάζονται ακόμα 4 αναδρομικές κλήσεις
 - `flood_fill_recur((x+1,y+1), fill_colour);`
 - `flood_fill_recur((x+1,y-1), fill_colour);`
 - `flood_fill_recur((x-1,y+1), fill_colour);`
 - `flood_fill_recur((x-1,y-1), fill_colour);`
- Το βασικό πρόβλημα είναι η μη αποδοτικότητα

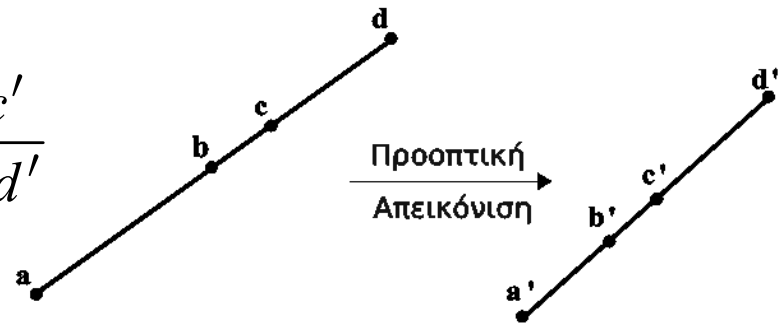
Προοπτική Διόρθωση

- Η σχεδίαση των αντικειμένων πραγματοποιείται στον 2Δ χώρο οθόνης, ενώ οι ιδιότητες τους σχετίζονται με τις 3Δ κορυφές τους
- Ο μετασχηματισμός προβολής γενικά δεν διατηρεί την αναλογία των αποστάσεων \rightarrow η γραμμική παρεμβολή των τιμών των ιδιοτήτων στο χώρο οθόνης είναι λανθασμένη
- **Η Προοπτική Διόρθωση** χρησιμοποιείται για να αποκτηθεί η σωστή τιμή χαρακτηριστικού στο προβαλλόμενο σημείο
- Βασίζεται στο γεγονός ότι οι προβολικοί μετασχηματισμοί διατηρούν τους χιαστούς λόγους (cross-ratios)

Προοπτική Διόρθωση (2)

- Παράδειγμα:
 - Έστω **ad** ευθύγραμμο τμήμα και **b** το μέσο του στον 3Δ χώρο
 - Έστω **a'**, **d'**, **b'** οι προοπτικές προβολές των σημείων **a**, **d**, **b**

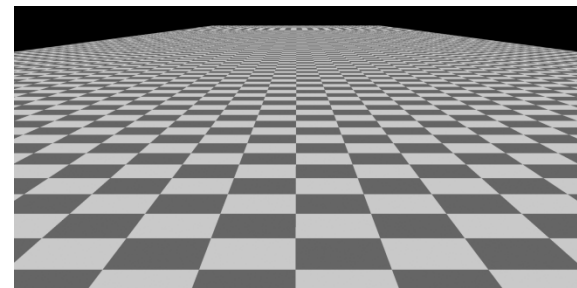
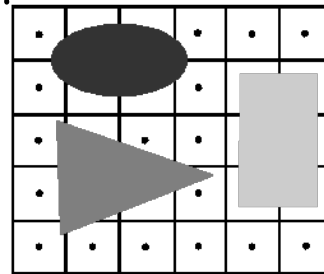
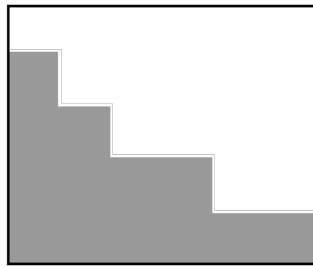
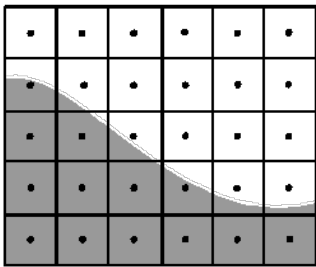
$$\left. \begin{array}{l} \frac{ac}{cd} = \frac{a'c'}{c'd'} \\ \frac{ab}{bd} = 1, \quad \frac{a'b'}{b'd'} = q \end{array} \right\} \frac{ac}{cd} = \frac{a'c'}{qc'd'}$$



- Ο Heckbert προτείνει μια αποδοτική λύση για την προοπτική διόρθωση:
 - Προοπτική διαίρεση ιδιότητας :
 - ◆ Έστω $[x, y, z, w, c]^T$ οι συντεταγμένες μιας κορυφής πριν την προοπτική προβολή, όπου c είναι η τιμή της ιδιότητας $\rightarrow [x/w, y/w, z/w, c/w, 1/w]^T$ είναι οι συντεταγμένες της μετά την προβολή

Αντιτάυτιση στον Χώρο

- Οι αλγόριθμοι σχεδίασης στοιχειωδών σχημάτων θεωρούν τα εικονοστοιχεία σαν σημεία
- Τα εικονοστοιχεία όμως έχουν μια μικρή επιφάνεια και **δεν** είναι μαθηματικά σημεία → φαινόμενα ταύτισης
- Φαινόμενα ταύτισης:
 - ακανόνιστη εμφάνιση περιγραμμάτων αντικειμένων
 - λανθασμένη σχεδίαση μικρών αντικειμένων
 - λανθασμένη σχεδίαση λεπτομερειών



Τεχνικές Αντιαύτισης

- Η αντιαύτιση επωφελείται της ανάλυσης φωτεινότητας για να κερδίσει χωρική ανάλυση

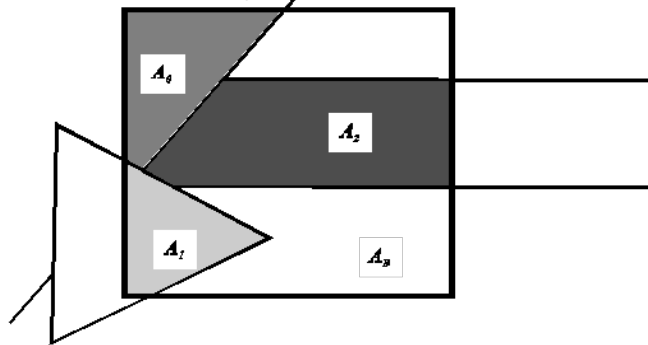
Υπάρχουν 2 κατηγορίες τεχνικών αντιαύτισης:

- **Προ-Φιλτράρισμα:**
 - αφαιρεί τις υψηλές συχνότητες πριν την δειγματοληψία
 - διαχειρίζεται το εικονοστοιχείο σαν πεπερασμένη περιοχή
 - υπολογίζει την % συνεισφορά κάθε στοιχειώδους σχήματος στην περιοχή του εικονοστοιχείου
- **Μετά-Φιλτράρισμα:**
 - αφαιρεί τις υψηλές συχνότητες μετά την δειγματοληψία
 - αυξάνει την συχνότητα δειγματοληψίας
 - επανέρχεται στην κανονική ανάλυση με μέσους όρους

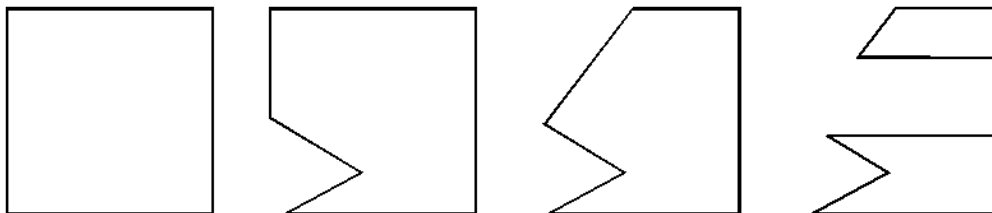
Μέθοδοι Αντιαύτισης με Προ-Φιλτράρισμα

Σχεδίαση Πολυγώνου με Αντιαύτιση : Αλγόριθμος Catmull

- Θεωρεί κάθε εικονοστοιχείο σαν ένα τετράγωνο παράθυρο
- Αποκόπτει όλα τα επικαλύπτοντα πολύγωνα
- Υπολογίζει την ορατή περιοχή κάθε πολυγώνου σαν % της επιφάνειας του εικονοστοιχείου



- Για την αποκοπή πολυγώνων απαιτείται γενικός αλγόριθμος, όπως ο Greiner-Horman



Αλγόριθμος Catmull

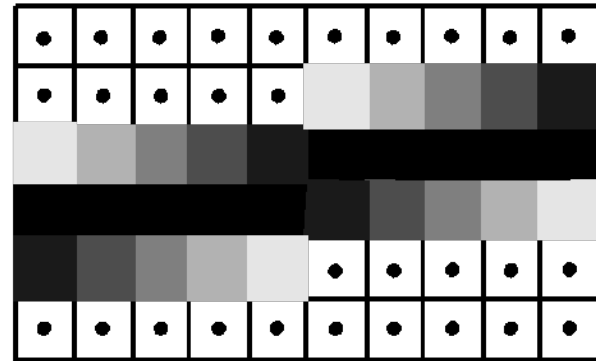
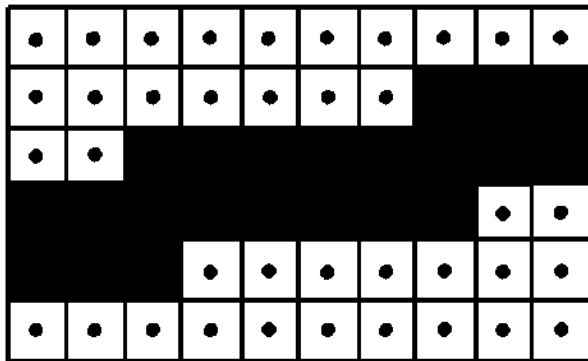
Αλγόριθμος:

1. Αποκοπή όλων των πολυγώνων ως προς το παράθυρο του εικονοστοιχείου →
 $P_0 \dots P_{n-1}$: τα τμήματα αποκομμένων πολυγώνων
 2. Απομάκρυνση κρυμμένων επιφανειών:
 - (a) Ταξινόμηση των πολυγώνων $P_0 \dots P_{n-1}$ ως προς το βάθος
 - (b) Αποκοπή ως προς την περιοχή που σχηματίζεται από την αφαίρεση των πολυγώνων από το εναπομείναν παράθυρο εικονοστοιχείου με σειρά βάθους →
 $P_0 \dots P_{m-1}$ ($m \leq n$) τα ορατά μέρη των πολυγώνων &
 $A_0 \dots A_{m-1}$ οι αντίστοιχες επιφάνειές τους
 3. Υπολογισμός του τελικού χρώματος του εικονοστοιχείου:
 $A_0 C_0 + A_1 C_1 + \dots + A_{m-1} C_{m-1} + A_B C_B$
όπου C_i : το χρώμα του πολυγώνου i & A_B, C_B : επιφάνεια και χρώμα φόντου
- Ο παραπάνω αλγόριθμος είναι πρακτικά μη βιώσιμος:
 - Εξωπραγματικοί υπολογισμοί
 - Κάθε πολύγωνο δεν έχει σταθερό χρώμα σε ένα εικονοστοιχείο (υφή)

Μέθοδοι Αντιαύτισης με Προ-Φιλτράρισμα (2)

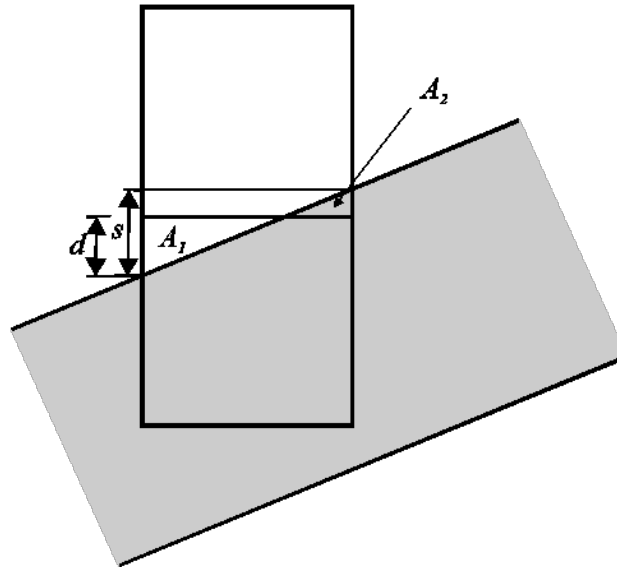
Σχεδίαση Ευθείας με Αντιαύτιση

- Αλγόριθμος Bresenham
 - Δυαδική απόφαση για την επιλογή του πλησιέστερου εικονοστοιχείου στην μαθηματική εξίσωση της ευθείας → ακανόνιστες ευθείες και ακμές πολυγώνων
- Οι ευθείες πρέπει να έχουν συγκεκριμένο πλάτος → μοντελοποίηση σαν λεπτά παραλληλόγραμμα
 - Η δυαδική απόφαση είναι λανθασμένη
 - Η τιμή χρώματος εξαρτάται από το % επικάλυψης από την ευθεία



Σχεδίαση Ευθείας με Αντιτάυτιση

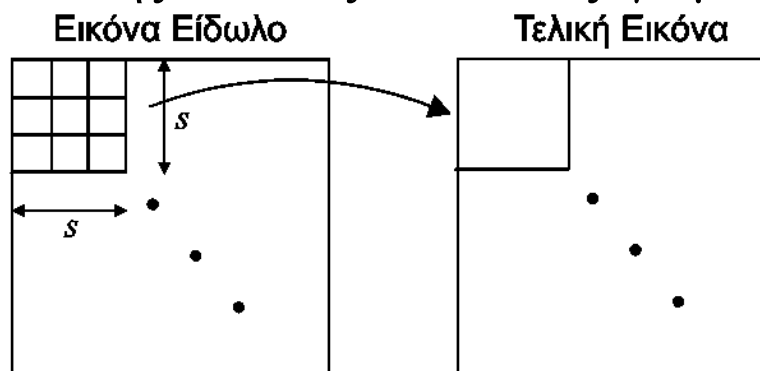
- Παράδειγμα:



- Η ευθεία βρίσκεται στο 1^ο οκταμόριο με κλίση $s = -\frac{a}{b}$
- 2 εικονοστοιχεία καλύπτονται μερικώς από την ευθεία
- Υπολογισμός του % των τριγώνων A_1 & A_2
- Χρώμα πάνω εικονοστοιχείου = χρώμα ευθείας σε % A_2
- Χρώμα κάτω εικονοστοιχείου = χρώμα ευθείας σε % $(1-A_1)$
- Περιοχές τριγώνων: $A_1 = \frac{d^2}{2s}$ $A_2 = \frac{(s-d)^2}{2s}$

Μέθοδοι Αντιαύτισης με Μετά-Φιλτράρισμα

- Χρησιμοποιούν περισσότερα του 1 δείγματος για κάθε εικονοστοιχείο → εικόνα σε μεγαλύτερη ανάλυση
- Το αποτέλεσμα επανέρχεται στην ανάλυση του πλέγματος εικονοστοιχείων με ζυγισμένους μέσους όρους
- Λόγω απλότητας είναι η πιο διαδεδομένη τεχνική
- Παράδειγμα:
 - Για εικόνα 1024×1024 λαμβάνουμε 3072×3072 δείγματα
 - 9 δείγματα για κάθε εικονοστοιχείο (3 οριζόντια \times 3 κάθετα)
 - 3×3 εικονικά εικονοστοιχεία αντιστοιχούν σε 1 της τελικής εικόνας
 - Το χρώμα της τελικής εικόνας είναι ο ζυγισμένος μέσος όρος των 9 δειγμάτων



Αλγόριθμος Μετά-Φιλτραρίσματος

Αλγόριθμος:

1. Η συνεχής εικόνα δειγματοληπτείται s φορές της τελικής ανάλυσης (s οριζόντια \times s κάθετα) δημιουργώντας την εικόνα I_u .
2. Η I_u περνά από φίλτρο χαμηλών συχνοτήτων ώστε να εξαλειφθούν οι υψηλές συχνότητες που προκαλούν ταύτιση.
3. Η φιλτραρισμένη εικόνα επαναδειγματοληπτείται στην τελική ανάλυση για να παραχθεί η τελική εικόνα I_f .

- Χρήση $s \times s$ φίλτρου συνέλιξης h αντί για μέσο όρο των $s \times s$ δειγμάτων

- Βήματα:

- Θέσε το φίλτρο πάνω στο εικονοστοιχείο της I_u
- Υπολόγισε την τιμή της τελικής εικόνας

$$I_f(i, j) = \sum_{p=0}^{s-1} \sum_{q=0}^{s-1} I_u(i * s + p, j * s + q) \cdot h(p, q)$$

- Μετακίνησε το φίλτρο

Αλγόριθμος Μετά-Φιλτραρίσματος (2)

- Παραδείγματα φίλτρων συνέλιξης:

					1	2	3	4	3	2	1				
					2	4	6	8	6	4	2				
			1	2	3	2	1	3	6	9	12	9	6	3	
			2	4	6	4	2	4	8	12	16	12	8	4	
		1	2	3	2	1	3	6	9	12	9	6	3		
	1	2	1	2	4	6	4	2	2	4	6	8	6	4	2
	2	4	2	2	4	6	4	2	1	2	3	4	3	2	1
	1	2	1	1	2	3	2	1							

- Για αποφυγή μετάθεσης χρωμάτων, κανονικοποιούμε το φίλτρο

$$\sum_{p=0}^{s-1} \sum_{q=0}^{s-1} h(p, q) = 1$$

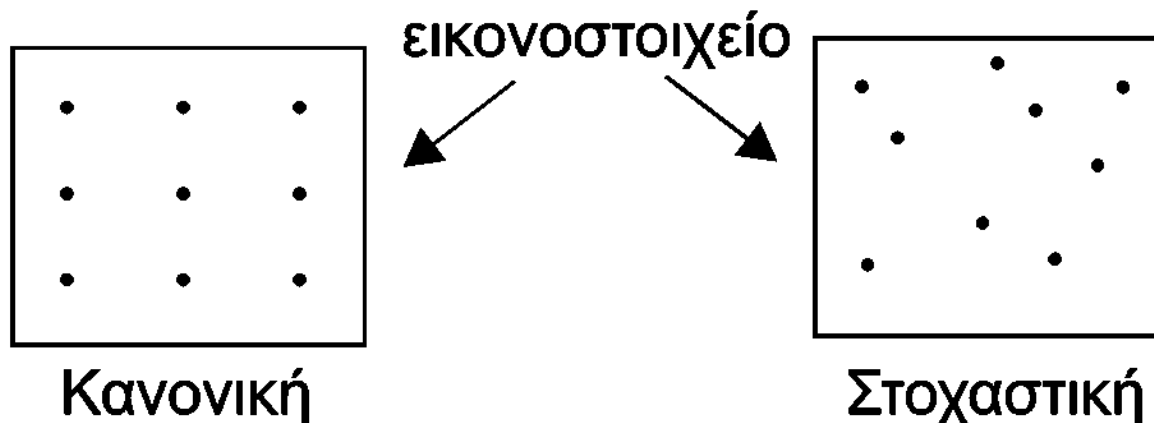
- Όσο πιο μεγάλο $s \rightarrow$ τόσο καλύτερα αποτελέσματα

- Μειονεκτήματα:

- $\uparrow s \rightarrow \uparrow$ χρόνος παραγωγής εικόνας & \uparrow απαίτηση μνήμης
- Όσο και να μεγαλώσει το s , θεωρητικά το πρόβλημα ταύτισης παραμένει
- Δεν προσαρμόζεται στην πολ/τητα της εικόνας \rightarrow άχρηστοι υπολογισμοί

Άλλοι Αλγόριθμοι Μετά-Φιλτραρίσματος

- Προσαρμοστικό Μετά-Φιλτράρισμα:
 - Αυξάνει τον ρυθμό δειγματοληψίας όπου υπάρχουν υψηλές συχνότητες
 - Πιο πολύπλοκος αλγόριθμος
- Στοχαστικό Μετά-Φιλτράρισμα:
 - Δειγματοληπτεί την εικόνα μη ομοιόμορφα
 - Τα φαινόμενα ταύτισης μετατρέπονται σε θόρυβο (το μάτι τα αγνοεί)



2Δ Αλγόριθμοι Αποκοπής

- Εμποδίζουν την ανάθεση τιμών εκτός ορίων στην συσκευή απεικόνισης
- **Αντικείμενο Αποκοπής (παράθυρο):** Η συσκευή απεικόνισης συνήθως μοντελοποιείται σαν ορθογώνιο παραλληλόγραμμο, το οποίο ορίζει τις τιμές εντός ορίων
- **Αποκοπτόμενο Αντικείμενο:** στοιχειώδες αντικείμενο σκηνης
- Άμεση γενίκευση από 2Δ σε 2Δ
- Σχέση αποκοπτόμενου αντικειμένου και αντικειμένου αποκοπής
 - Το αποκοπτόμενο βρίσκεται εντός αντικειμένου αποκοπής: σχεδίαση
 - Το αποκοπτόμενο βρίσκεται εκτός αντικειμένου αποκοπής : όχι σχεδίαση
 - Το αποκοπτόμενο τέμνει το αντικείμενο αποκοπής: υπολογισμός της τομής από έναν αλγόριθμο 2Δ αποκοπής και σχεδίαση του αποτελέσματος

Αποκοπή Σημείου

- Τετριμμένη:
 - Έλεγχος αν το σημείο (x, y) είναι εντός του αντικειμένου αποκοπής
- Αν το αντικείμενο αποκοπής είναι ορθογώνιο παραλληλόγραμμο:
 - Εκμετάλλευση των διαγώνιων κορυφών (x_{\min}, y_{\min}) , (x_{\max}, y_{\max})
- Έλεγχος:

Αν $x_{\min} \leq x \leq x_{\max}$ & $y_{\min} \leq y \leq y_{\max}$

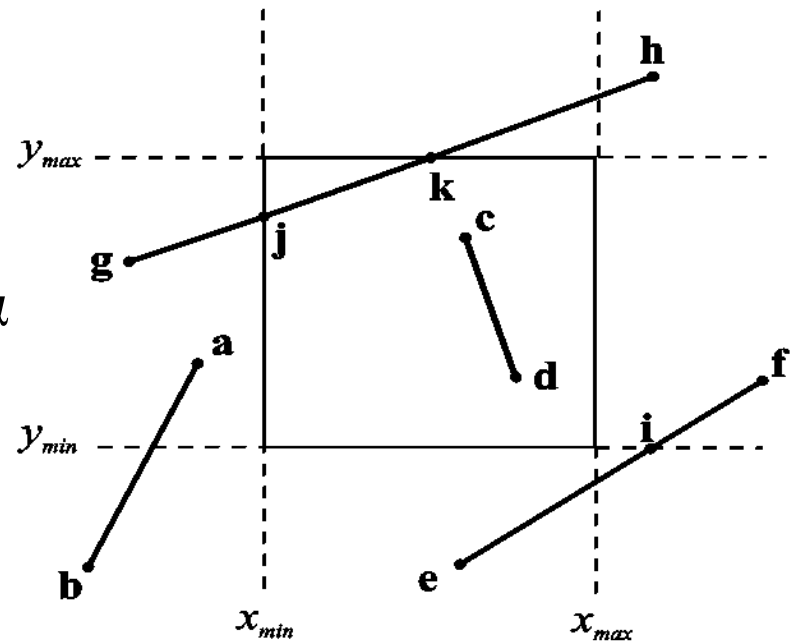
Τότε το σημείο είναι εξ' ολοκλήρου εντός αντικειμένου αποκοπής και σχεδιάζεται

Διαφορετικά το σημείο είναι εξ' ολοκλήρου εκτός αντικειμένου αποκοπής και ΔΕΝ σχεδιάζεται

Αποκοπή Ευθείας – Αλγόριθμος CS

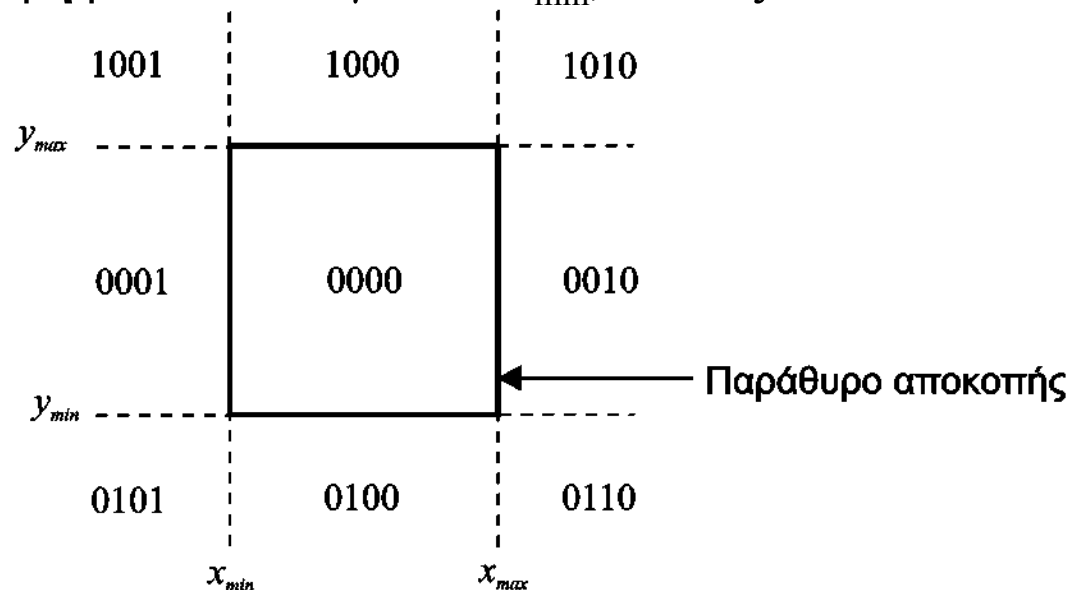
Αλγόριθμος Cohen – Sutherland (CS)

- Έλεγχος χαμηλού κόστους για απόφαση αν ένα ευθύγραμμο τμήμα βρίσκεται εξ'ολοκλήρου εντός ή εξ'ολοκλήρου εκτός του παραθύρου αποκοπής
- Για κάθε ευθύγραμμο τμήμα, μη τετριμμένης απόφασης, υπολογίζεται τομή με μια από τις ευθείες που ορίζουν τα όρια του παραθύρου
- Ο αλγόριθμος εφαρμόζεται αναδρομικά και στα δυο προκύπτοντα ευθύγραμμα τμήματα



Αποκοπή Ευθείας – Αλγόριθμος CS (2)

- Επίπεδο παραθύρου αποκοπής διαιρείται σε 9 περιοχές
- Σε κάθε περιοχή ανατίθεται ένας 4-ψήφιος δυαδικός κωδικός
- Κωδικός προκύπτει με βάση τους παρακάτω κανόνες:
 - **Πρώτο ψηφίο:** Θέσε 1 για $y > y_{max}$, αλλιώς θέσε 0
 - **Δεύτερο ψηφίο:** Θέσε 1 για $y < y_{min}$, αλλιώς θέσε 0
 - **Τρίτο ψηφίο:** Θέσε 1 για $x > x_{max}$, αλλιώς θέσε 0
 - **Τέταρτο ψηφίο:** Θέσε 1 για $x < x_{min}$, αλλιώς θέσε 0



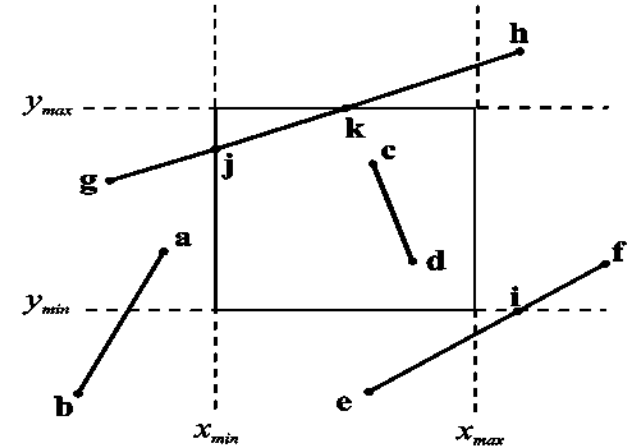
Αποκοπή Ευθείας – Αλγόριθμος CS (3)

- Έστω c_1, c_2 οι 4-ψήφιοι κωδικοί των άκρων ενός ευθύγραμμου τμήματος
- Σε κάθε ακραίο σημείο ανατίθεται ένας 4-ψήφιος κωδικός σύμφωνα με τους παραπάνω κανόνες
- Τότε οι έλεγχοι χαμηλού κόστους είναι:
 - $\text{Av } c1 \vee c2 = 0000$
Τότε το ευθύγραμμο τμήμα βρίσκεται εξ'ολοκλήρου εντός
 - $\text{Av } c1 \wedge c2 \neq 0000$
Τότε το ευθύγραμμο τμήμα βρίσκεται εξ'ολοκλήρου εκτός

Αποκοπή Ευθείας – Αλγόριθμος CS (4)

- Παράδειγμα :

Σημείο	Κωδικός	Σημείο	Κωδικός
a	0001	e	0100
b	0101	f	0010
c	0000	g	0001
d	0000	h	1010



- Το **ab** είναι εξολοκλήρου εκτός, αφού $0001 \wedge 0101 \neq 0000$
- Το **cd** είναι εξολοκλήρου εντός, αφού $0000 \vee 0000 = 0000$
- Για τα **ef** & **gh** οι παραπάνω έλεγχοι δεν μπορούν να αποφασίσουν \Rightarrow υπολογισμός των σημείων τομής
- Τομή **ef** με γραμμή $y = y_{min}$ αφού το 2^ο ψηφίο του κωδικού διαφέρει στα **e** & **f**
- Συνέχεια με το ευθύγραμμο τμήμα **if** καθότι το 2^ο ψηφίο του κώδικα της κορυφής **f** έχει τιμή 0 (εντός)
- Για το **gh** υπολόγισε ένα από τα σημεία τομής **k** & συνέχισε με το **gk** υπολογίζοντας την τομή **j** & αναδρομικά φτάσε σε μια τετριμμένη απόφαση για το **jk** που είναι εντός

Αποκοπή Ευθείας – Αλγόριθμος CS (5)

Αλγόριθμος:

```
CS_Clip ( vertex p1, p2, float xmin, xmax, ymin, ymax ) {  
  int c1, c2;  vertex i;  edge e;  
  c1 = mkcode (p1);  c2 = mkcode (p2);  
  if ((c1 | c2) == 0)  
    /* p1p2 είναι εντός */  
  else if ((c1 & c2) != 0)  
    /* p1p2 είναι εκτός */  
  else {  
    e= /* γραμμή παραθύρου με (c1 bit != c2 bit) */  
    i = intersect_lines (e, (p1,p2));  
    if outside (e, p1)  
      CS_Clip(i, p2, xmin, xmax, ymin, ymax);  
    else  
      CS_Clip(p1, i, xmin, xmax, ymin, ymax);  
  }  
}
```

Αποκοπή Ευθείας – Αλγόριθμος Skala

Αλγόριθμος Skala:

- Πιο αποδοτικός από τον αλγόριθμο CS μέσω κατηγοριοποίησης των κορυφών του παραθύρου αποκοπής σε σχέση με το αποκοπτόμενο ευθύγραμμο τμήμα
- Ένας δυαδικός κωδικός c_i ανατίθεται σε κάθε κορυφή του παραθύρου αποκοπής $v_i = (x_i, y_i)$ ως εξής:
- $$\diamond c_i = \begin{cases} 1, & \text{όταν } l(x_i, y_i) \geq 0 \\ 0, & \text{διαφορετικά} \end{cases}$$

Όπου $l(x, y)$ είναι η συνάρτηση που ορίζεται από το ευθύγραμμο τμήμα που πρόκειται να αποκοπεί
- Ο κωδικός c_i υποδηλώνει την πλευρά του ευθυγράμμου τμήματος στην οποία βρίσκεται η κορυφή v_i

Αποκοπή Ευθείας – Αλγόριθμος Skala (2)

- Οι κωδικοί υπολογίζονται θεωρώντας της κορυφές σε μια συνεπή διάταξη γύρω από το παράθυρο αποκοπής (π.χ. αριστερόστροφα)
- Μια ακμή του παραθύρου αποκοπής τέμνεται από το ευθύγραμμο τμήμα για κάθε αλλαγή στην κωδικοποίηση των κορυφών (από 0 σε 1 ή από 1 σε 0)
- Ένας προ-υπολογισμένος πίνακας δίνει άμεσα τις ακμές του παραθύρου αποκοπής που τέμνονται από το ευθύγραμμο τμήμα με διάνυσμα κωδικών $[c_0, c_1, c_2, c_3]$ \rightarrow αντικαθιστά την αναδρομική περίπτωση του αλγορίθμου CS

Αποκοπή Ευθείας – Αλγόριθμος LB

Αλγόριθμος Liang – Barsky (LB)

- Αποκοπή ευθείας χωρίς αναδρομή
- > 30% πιο αποδοτικός από τον CS
- Μπορεί εύκολα να επεκταθεί στις 3Δ
- Βασίζεται στην παραμετρική εξίσωση του αποκοπτόμενου ευθυγράμμου τμήματος από το $\mathbf{p}_1(x_1, y_1)$ στο $\mathbf{p}_2(x_2, y_2)$:

$$\mathbf{P} = \mathbf{p}_1 + t (\mathbf{p}_2 - \mathbf{p}_1), \quad t \in [0, 1]$$

ή

$$x = x_1 + t \Delta x, \quad y = y_1 + t \Delta y$$

όπου

$$\Delta x = x_2 - x_1, \quad \Delta y = y_2 - y_1$$

Αποκοπή Ευθείας – Αλγόριθμος LB (2)

- Για το τμήμα του ευθυγράμμου τμήματος που βρίσκεται εντός του παραθύρου αποκοπής ισχύει:

$$x_{\min} \leq x_1 + t \Delta x \leq x_{\max} ,$$

$$y_{\min} \leq y_1 + t \Delta y \leq y_{\max}$$

ή

$$-t \Delta x \leq x_1 - x_{\min} ,$$

$$t \Delta x \leq x_{\max} - x_1 ,$$

$$-t \Delta y \leq y_1 - y_{\min} ,$$

$$t \Delta y \leq y_{\max} - y_1$$

Αποκοπή Ευθείας – Αλγόριθμος LB (3)

- Οι παραπάνω ανισότητες έχουν την κοινή μορφή:

$$t p_i \leq q_i ,$$

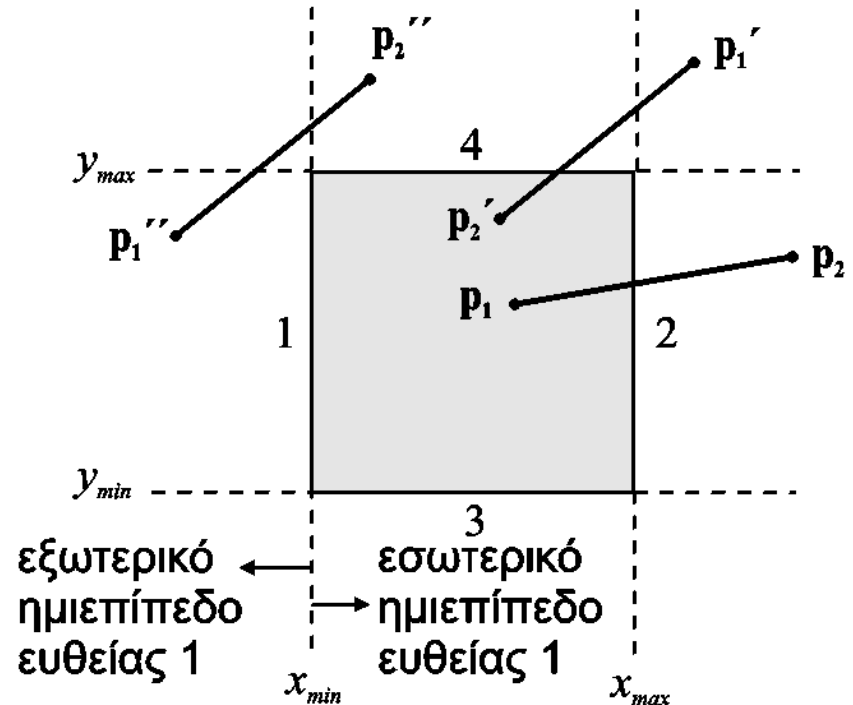
όπου

$$p_1 = -\Delta x , q_1 = x_1 - x_{\min}$$

$$p_2 = \Delta x , q_2 = x_{\max} - x_1$$

$$p_3 = -\Delta y , q_3 = y_1 - y_{\min}$$

$$p_4 = \Delta y , q_4 = y_{\max} - y_1$$



Αποκοπή Ευθείας – Αλγόριθμος LB (4)

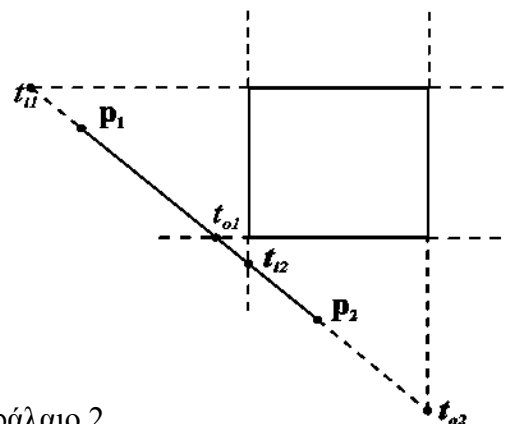
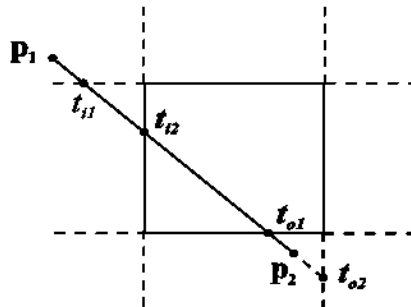
- Παρατηρήσεις:
 - Αν $\mathbf{p}_i = 0$, το ευθύγραμμο τμήμα είναι παράλληλο προς την ακμή i του παραθύρου και το πρόβλημα αποκοπής είναι τετριμμένο
 - Αν $\mathbf{p}_i \neq 0$, η παραμετρική τιμή του σημείου τομής του ευθυγράμμου τμήματος με τη ευθεία που ορίζεται από την ακμή i του παραθύρου είναι $t_i = \mathbf{q}_i / \mathbf{p}_i$
 - Αν $\mathbf{p}_i < 0$, το κατευθυνόμενο ευθύγραμμο τμήμα είναι «εισερχόμενο» ως προς την ακμή i του παραθύρου
 - Αν $\mathbf{p}_i > 0$, το κατευθυνόμενο ευθύγραμμο τμήμα είναι «εξερχόμενο» ως προς την ακμή i του παραθύρου

Αποκοπή Ευθείας – Αλγόριθμος LB (5)

- Τα t_{in} και t_{out} υπολογίζονται ως εξής:

$$t_{in} = \max(\{\frac{q_i}{p_i} \mid p_i < 0, i:1..4\} \cup \{0\}), \quad t_{out} = \min(\{\frac{q_i}{p_i} \mid p_i > 0, i:1..4\} \cup \{1\})$$

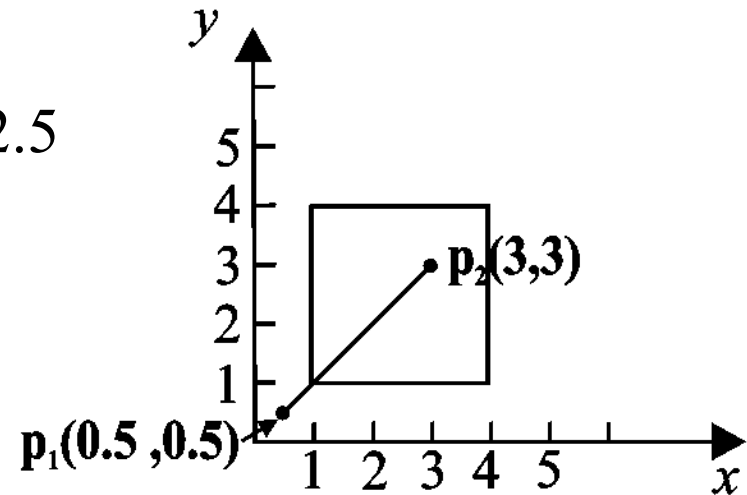
- Τα $\{0\}$, $\{1\}$ αποτελούν τις αρχικές και τελικές παραμετρικές τιμές στα ακραία σημεία του ευθύγραμμου τμήματος
- Αν $t_{in} \leq t_{out}$, οι τιμές t_{in} και t_{out} εισάγονται στην παραμετρική εξίσωση ευθείας για να υπολογιστούν τα ακραία σημεία του αποκομμένου τμήματος
- Διαφορετικά δεν υπάρχει τομή με το παράθυρο αποκοπής



Αποκοπή Ευθείας – Αλγόριθμος LB (6)

Παράδειγμα αλγορίθμου LB:

- Υπολόγισε: $\Delta x = 2.5$ and $\Delta y = 2.5$
- Υπολόγισε: $p_1 = -2.5, q_1 = -0.5$
 $p_2 = 2.5, q_2 = 3.5$
 $p_3 = -2.5, q_3 = -0.5$
 $p_4 = 2.5, q_4 = 3.5.$



- Υπολόγισε: $t_{in} = \max(\{\frac{q_1}{p_1}, \frac{q_3}{p_3}\} \cup \{0\}) = 0.2$, $t_{out} = \min(\{\frac{q_2}{p_2}, \frac{q_4}{p_4}\} \cup \{1\}) = 1$
- Αφού $t_{in} < t_{out}$ υπολόγισε τα ακραία σημεία $\mathbf{p}_1'(x_1', y_1')$, $\mathbf{p}_2'(x_2', y_2')$ του αποκομμένου ευθύγραμμου τμήματος χρησιμοποιώντας την παραμετρική εξίσωση:

$$x_1' = x_1 + t_{in} \Delta x = 0.5 + 0.2 \cdot 2.5 = 1$$

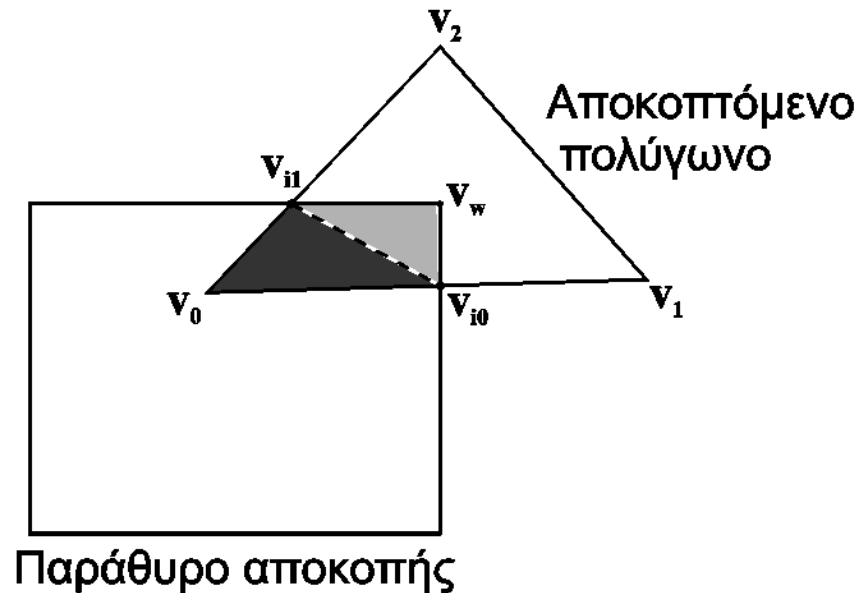
$$y_1' = y_1 + t_{in} \Delta y = 0.5 + 0.2 \cdot 2.5 = 1$$

$$x_2' = x_1 + t_{out} \Delta x = 0.5 + 1 \cdot 2.5 = 3$$

$$y_2' = y_1 + t_{out} \Delta y = 0.5 + 1 \cdot 2.5 = 3$$

Αποκοπή Πολυγώνου

- Στην 2Δ αποκοπή πολυγώνων το αποκοπτόμενο αντικείμενο και το αντικείμενο αποκοπής είναι πολύγωνα (**αποκοπτόμενο πολύγωνο, πολύγωνο αποκοπής**)
- Γιατί είναι σημαντική η αποκοπή πολυγώνου?

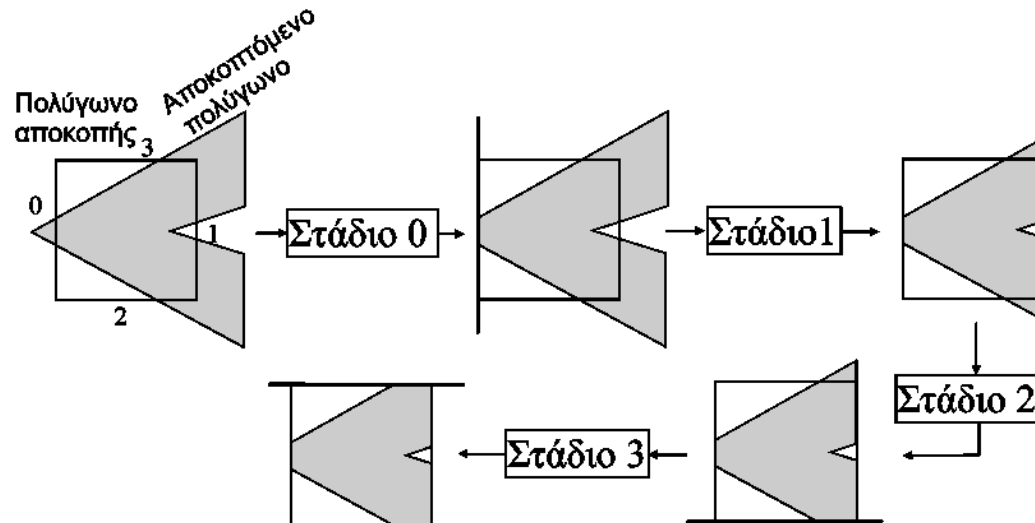


- Η αποκοπή πολυγώνου δεν θεωρείται πολλαπλή αποκοπή ευθείας

Αποκοπή Πολυγώνου – Αλγόριθμος SH

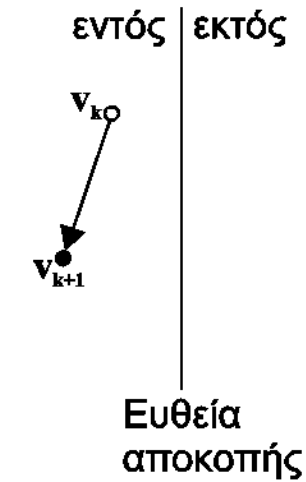
Αλγόριθμος Sutherland – Hodgman (SH):

- Αποκόπτει ένα τυχαίο πολύγωνο ως προς ένα **κυρτό** πολύγωνο αποκοπής
- Περιλαμβάνει m στάδια σωλήνωσης που αντιστοιχούν στις m ακμές του πολυγώνου αποκοπής
- Το στάδιο i | $i: 0 \dots m-1$ αποκόπτει το αποκοπτόμενο πολύγωνο ως προς το ευθύγραμμο τμήμα που ορίζεται από την ακμή i του πολυγώνου αποκοπής
- Η είσοδος του σταδίου i | $i: 1 \dots m-1$ είναι η έξοδος του σταδίου $i-1$
- Το πολύγωνο αποκοπής πρέπει να είναι κυρτό

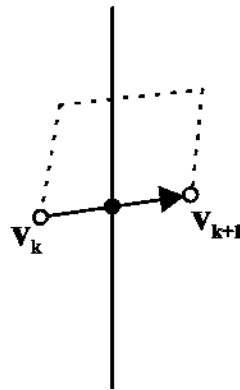


Αποκοπή Πολυγώνου – Αλγόριθμος SH (2)

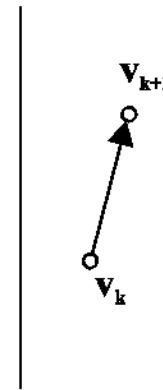
- Για κάθε στάδιο του αλγορίθμου SH υπάρχουν 4 πιθανές σχέσεις μεταξύ μιας ευθείας αποκοπής και μιας ακμής του πολυγώνου $v_k v_{k+1}$



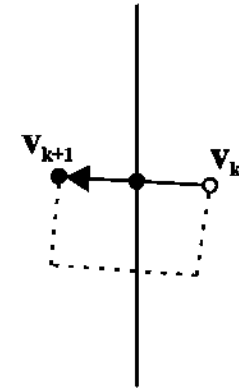
Περίπτωση 1:
1 έξοδος



Περίπτωση 2:
1 έξοδος



Περίπτωση 3:
0 έξοδοι



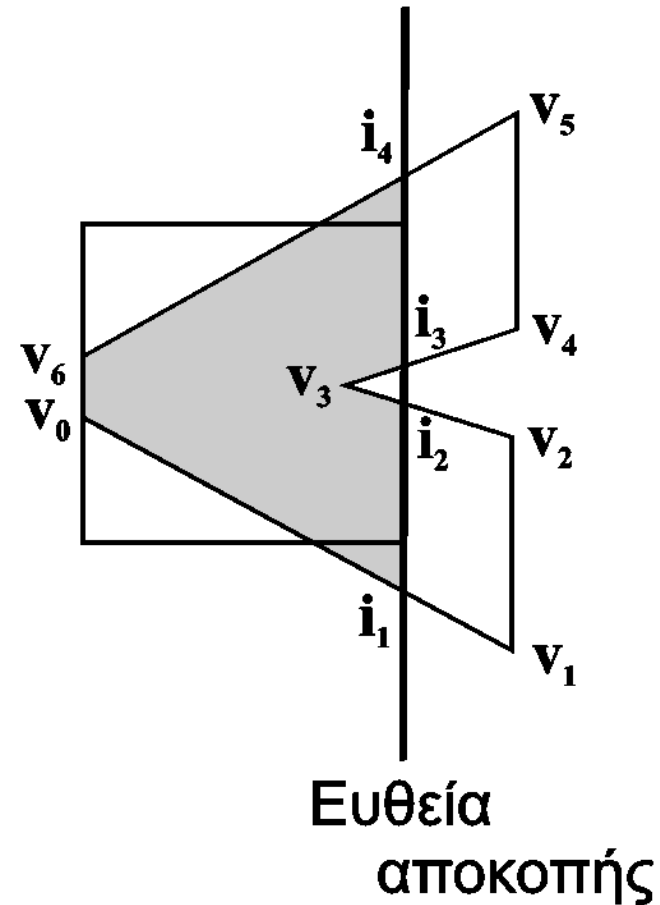
Περίπτωση 4:
2 έξοδοι

- κορυφή εξόδου

Αποκοπή Πολυγώνου – Αλγόριθμος SH (3)

- Παράδειγμα 1^ο σταδίου του αλγορίθμου SH:

V_k	V_{k+1}	Περίπτωση	Έξοδος
V_0	V_1	2	i_1
V_1	V_2	3	-
V_2	V_3	4	i_2, V_3
V_3	V_4	2	i_3
V_4	V_5	3	-
V_5	V_6	4	i_4, V_6
V_6	V_0	1	V_0



Αποκοπή Πολυγώνου – Αλγόριθμος SH (4)

- Αλγόριθμος:

```
polygon SH_Clip ( polygon C, S ) { /*C κυρτό */
    int i, m;
    edge e;
    polygon InPoly, OutPoly;
    m = getedgenumber(C);
    InPoly = S;
    for (i=0; i<m; i++) {
        e = getedge(C,i);
        SH_Clip_Edge(e,InPoly,OutPoly);
        InPoly = OutPoly
    }
    return OutPoly
}
```


Αποκοπή Πολυγώνου – Αλγόριθμος SH (5)

- Αλγόριθμος:

```
SH_Clip_Edge ( edge e, polygon InPoly, OutPoly ) {  
    int k, n; vertex vk, vkplus1, i;  
    n = getedgenumber(InPoly);  
    for (k=0; k<n; k++) {  
        vk = getvertex(InPoly,k); vkplus1=getvertex(InPoly,(k+1) mod n);  
        if (inside(e, vk) and inside(e, vkplus1))  
            /* Περίπτωση 1 */  
            putvertex(OutPoly,vkplus1)  
        else if (inside(e, vk) and !inside(e, vkplus1)) {  
            /* Περίπτωση 2 */  
            i = intersect_lines(e, (vk,vkplus1)); putvertex(OutPoly,i)  
        }  
        else if (!inside(e, vk) and !inside(e, vkplus1))  
            /* Περίπτωση 3 */  
        else {  
            /* Περίπτωση 4 */  
            i = intersect_lines(e, (vk,vkplus1)); putvertex(OutPoly,i);  
            putvertex(OutPoly,vkplus1)  
        }  
    }  
}
```

```
}
```

Αποκοπή Πολυγώνου – Αλγόριθμος SH (6)

- Η πολυπλοκότητα του αλγορίθμου SH είναι $O(mn)$ όπου m και n το πλήθος των κορυφών του πολυγώνου αποκοπής και του αποκοπτόμενου πολυγώνου αντίστοιχα
- Δεν χρειάζονται περίπλοκες διαδικασίες ή δομές δεδομένων → ο αλγόριθμος SH είναι πολύ αποδοτικός
- Ο αλγόριθμος SH είναι κατάλληλος για υλοποίηση σε υλικό καθότι, εν γένει, το πολύγωνο αποκοπής είναι σταθερό

Αποκοπή Πολυγώνου – Αλγόριθμος GH

Αλγόριθμος Greiner – Hormann

- Κατάλληλος για γενική περίπτωση πολυγώνου αποκοπής (C) και αποκοπτόμενου (S)
- Τα πολύγωνα S και C μπορεί να είναι τυχαία κλειστά πολύγωνα, ακόμα και να τέμνουν τους εαυτούς τους
- Οι πολυπλοκότητα των βημάτων 1 και 2 είναι $O(mn)$ όπου m και n είναι το πλήθος των κορυφών των πολυγώνων C και S αντίστοιχα
- Η συνολική πολυπλοκότητα του αλγορίθμου GH είναι $O(mn)$
- Στην πράξη, οι σύνθετες δομές δεδομένων που χρησιμοποιούνται στον αλγόριθμο GH τον κάνουν λιγότερο αποδοτικό από τον αλγόριθμο SH

Αποκοπή Πολυγώνου – Αλγόριθμος GH (2)

- Ο αλγόριθμος GH βασίζεται στον έλεγχο αριθμού περιελίξεων για σημείο p και πολύγωνο P , συμβολικά $\rightarrow \omega(P, p)$
- $\omega(P, p)$ δεν αλλάζει όσο η τοπολογική σχέση του σημείου p και του πολυγώνου P παραμένει σταθερή
- **Αν** το p τμήσει το P τότε το $\omega(P, p)$ αυξάνεται ή μειώνεται
- **Αν** το $\omega(P, p)$ είναι περιττό τότε το p βρίσκεται εντός του P , διαφορετικά βρίσκεται εκτός

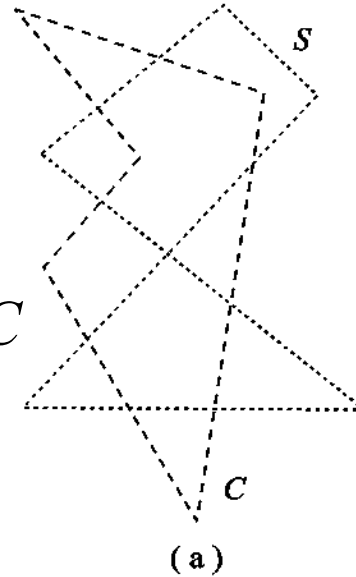
Αποκοπή Πολυγώνου – Αλγόριθμος GH (3)

- 3 βήματα αλγορίθμου GH:
 1. Ακολουθήσε την περίμετρο του S ξεκινώντας από την κορυφή v_{s0} . Μια νοητή ακίδα εναλλάσσεται μεταξύ των καταστάσεων *on* και *off* κάθε φορά που συναντάται η περίμετρος του C . Η αρχική κατάσταση της ακίδας είναι *on* αν η v_{s0} βρίσκεται εντός του C και *off* διαφορετικά. Έτσι υπολογίζεται το τμήμα της περιμέτρου του S που βρίσκεται εντός του C
 2. Όπως το βήμα 1, αλλά αντέστρεψε τους ρόλους των S και C . Με αυτόν τον τρόπο υπολογίζεται το τμήμα της περιμέτρου του C που βρίσκεται εντός του S
 3. Η ένωση των αποτελεσμάτων των βημάτων 1 και 2 είναι το αποτέλεσμα της αποκοπής του S από το C (ή αντίστοιχα της αποκοπής του C από το S)

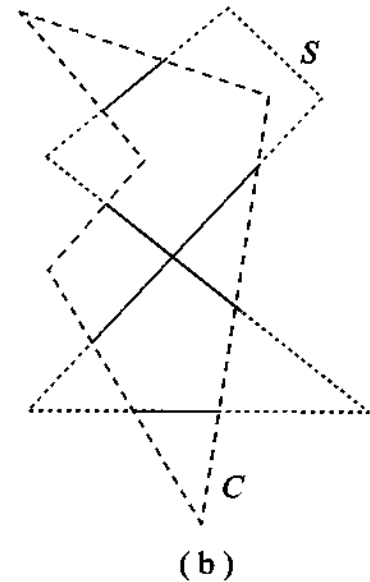
Αποκοπή Πολυγώνου – Αλγόριθμος GH (4)

- Παράδειγμα αλγορίθμου GH:

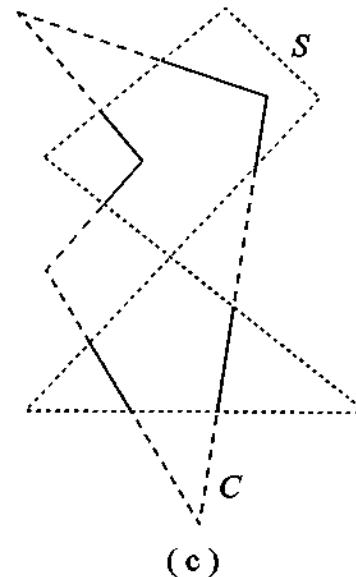
(a) Τα αρχικά πολύγωνα S , C



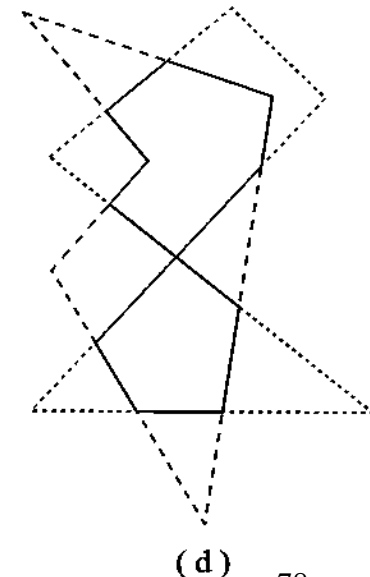
(b) Μετά το βήμα 1 του GH



(c) Μετά το βήμα 2 του GH



(d) Το τελικό αποτέλεσμα



Αποκοπή Πολυγώνου – Αλγόριθμος GH (5)

- Ο αλγόριθμος GH υπολογίζει την τομή των επιφανειών 2 πολυγώνων, $C \cap S$
- Εύκολα γενικεύεται για τον υπολογισμό $C \cup S$, $C - S$ και $S - C$ αλλάζοντας τις αρχικές καταστάσεις των ακίδων των S και C
- Υπάρχουν 4 πιθανοί συνδυασμοί της αρχικής κατάστασης
- Οι γενικεύσεις δεν χρησιμεύουν στο πρόβλημα αποκοπής