

UNIVERSITY OF THESSALY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE433 - COMPUTER GRAPHICS WITH OPENGL

---

## Project 2: Filling Polygons – Transformations - Clipping

---

*Group members*

Ιωάννης Ιάσων ΝΙΚΑΣ - 3771

Αναστάσιος ΚΑΛΟΥΣΗΣ - 3792

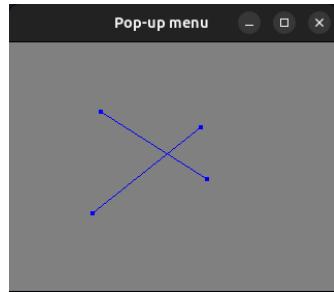
Παναγιώτης Νικόλαος ΤΣΟΓΚΑΣ -  
3672

*Group number:* 01

December 5, 2025 (Fall 2025)

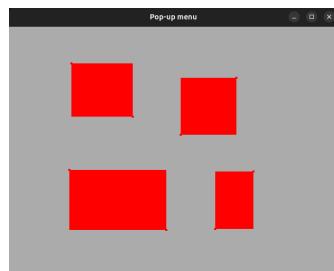
## Exercise 1

Changed Window Size:



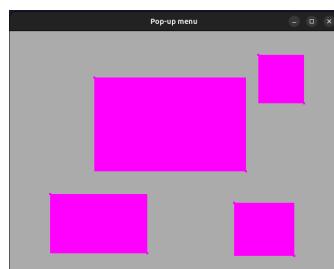
Σχήμα 1: Αλλάζαμε το window size, αλλάζοντας τις τιμές από την glutInitWindowSize.

Changed Background Color:



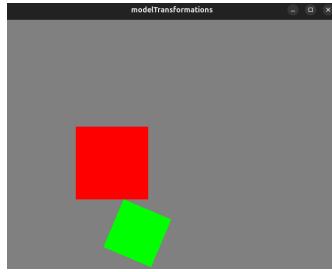
Σχήμα 2: Αλλάζαμε το background color, αλλάζοντας τις τιμές από την glClearColor.

Changed Drawing Colour:



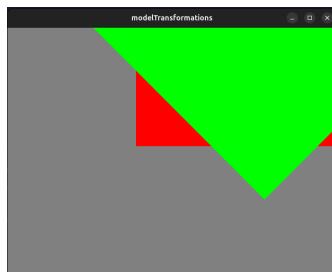
Σχήμα 3: Αλλάζαμε το drawing color, αλλάζοντας τις τιμές από την glColor3f. Πιο συγκεκριμένα για το χρώμα magenta, βάλαμε στην glColor3f τις τιμές 1.0, 0.0, 1.0 αντίστοιχα.

**Changed Coordinates of the points:**



Σχήμα 4: Αλλάζουμε τις συντεταγμένες των σημείων, δίνοντας διαφορετικά σημεία στην glRectf..

**Changed projection area:**



Σχήμα 5: Αλλάζουμε το projection area, αλλάζοντας τις τιμές από την gluOrtho2D.

## Exercise 2

Για τον πίνακα  $M$  θα ισχύει:

$$\begin{aligned}
 M &= T(4, -1) \times S(1.5, 0.5) \times R(90^\circ) \\
 &= \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} 1.5 & 0 & 4 \\ 0 & 0.5 & -1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1.5 & 4 \\ 0.5 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$

Επομένως για να βρούμε τα νέα σημεία του τριγώνου δημιουργούμε τον πίνακα:

$$\begin{bmatrix} x'_A & x'_B & x'_C \\ y'_A & y'_B & y'_C \\ 1 & 1 & 1 \end{bmatrix}$$

τον οποίο θα βρούμε εφαρμόζοντας τον πίνακα  $M$  στον ακόλουθο πίνακα:

$$\begin{bmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 1 \\ 0 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

Οπότε οι τελικές συντεταγμένες θα είναι:

$$\begin{bmatrix} x'_A & x'_B & x'_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{bmatrix} = M \times \begin{bmatrix} 0 & 2 & 1 \\ 0 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1.5 & 4 \\ 0.5 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 2 & 1 \\ 0 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 2.5 & -0.5 \\ -1 & 0 & -0.5 \\ 1 & 1 & 1 \end{bmatrix}$$

## Exercise 3

Θέλουμε να περιστρέψουμε το σημείο  $P(3, 2)$  γύρω από το σημείο  $Q(1, 1)$  κατά  $60^\circ$ . Για να το πραγματοποιήσουμε θα πρέπει αρχικά να μετατοπίσουμε το σημείο  $Q$  στην αρχή των αξόνων δηλαδή  $T(-1, -1)$ , έπειτα να πραγματοποιήσουμε την περιστροφή κατά  $60^\circ$ , δηλαδή  $R(60^\circ)$  και τέλος μετατοπίζουμε το σημείο στην αρχική του θέση δηλαδή  $T(1, 1)$ . Γράφοντας τις μετατροπές από τα δεξιά στα αριστερά ο τελικός πίνακας  $M$  θα είναι:

$$M = T(1, 1) \times R(60^\circ) \times T(-1, -1) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1/2 & -\sqrt{3}/2 & 0 \\ \sqrt{3}/2 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/2 & -\sqrt{3}/2 & 1 \\ \sqrt{3}/2 & 1/2 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/2 & -\sqrt{3}/2 & \frac{1+\sqrt{3}}{2} \\ \sqrt{3}/2 & 1/2 & \frac{1-\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Οπότε εφαρμόζοντας τον πίνακα  $M$  στο σημείο  $P$  θα πάρουμε:

$$M \times \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/2 & -\sqrt{3}/2 & \frac{1+\sqrt{3}}{2} \\ \sqrt{3}/2 & 1/2 & \frac{1-\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{4-\sqrt{3}}{2} \\ \frac{3-\sqrt{3}}{2} \\ 1 \end{bmatrix}$$

## Exercise 4

### 1. Shearing in x

Θα πολλαπλασιάσουμε τα σημεία με τον εξής πίνακα:

$$M = \begin{bmatrix} 1 & 1.2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Οπότε εφαρμόζοντας τον  $M$  στα σημεία έχουμε:

$$M \times \begin{bmatrix} 0 & 4 & 4 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1.2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 4 & 4 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 4 & 6.4 & 2.4 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

### 2. Shearing in y

Θα πολλαπλασιάσουμε τα σημεία με τον εξής πίνακα:

$$M = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Οπότε εφαρμόζοντας τον  $M$  στα σημεία έχουμε:

$$M \times \begin{bmatrix} 0 & 4 & 4 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 4 & 4 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 4 & 4 & 0 \\ 0 & -2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

## Exercise 5

Έχουμε τον πίνακα  $M = T(2, -3) \times S(2, 3) \times R(30^\circ)$ . Για να βρούμε τον αντίστροφό του θα εφαρμόσουμε τους αντίστροφους μετασχηματισμούς με ανάποδη σειρά:

$$\begin{aligned} M = T(2, -3) \times S(2, 3) \times R(30^\circ) &\implies M \times M^{-1} = T(2, -3) \times S(2, 3) \times R(30^\circ) \times M^{-1} \implies \\ I = T(2, -3) \times S(2, 3) \times R(30^\circ) \times M^{-1} &\implies T^{-1}(2, -3) = S(2, 3) \times R(30^\circ) \times M^{-1} \implies \\ T^{-1}(2, -3) \times S^{-1}(2, 3) &= R(30^\circ) \times M^{-1} \implies M^{-1} = R^{-1}(30^\circ) \times T^{-1}(2, -3) \times S^{-1}(2, 3) \\ &\implies M^{-1} = R(-30^\circ) \times T(-2, 3) \times S(1/2, 1/3) \end{aligned}$$

Επομένως ο  $M^{-1}$  είναι ο εξής:

$$\begin{aligned} M &= \begin{bmatrix} \cos(-30^\circ) & -\sin(-30^\circ) & 0 \\ \sin(-30^\circ) & \cos(-30^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & -\sqrt{3} + \frac{3}{2} \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 + \frac{3\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{4} & \frac{1}{6} & \frac{1}{2} - \frac{\sqrt{3}}{2} \\ -\frac{1}{4} & \frac{\sqrt{3}}{6} & \frac{1}{2} + \frac{\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Οπότε εφαρμόζοντας τον  $M^{-1}$  στο σημείο  $P$  έχουμε:

$$M^{-1} \times \begin{bmatrix} 5 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{4} & \frac{1}{6} & \frac{1}{2} - \frac{\sqrt{3}}{2} \\ -\frac{1}{4} & \frac{\sqrt{3}}{6} & \frac{1}{2} + \frac{\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \cdot \frac{\sqrt{3}}{4} + \frac{5}{6} + \frac{1}{2} - \frac{\sqrt{3}}{2} \\ -5 \cdot \frac{1}{4} + 5 \cdot \frac{\sqrt{3}}{6} + \frac{1}{2} + \frac{\sqrt{3}}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{3}{4} + \frac{4\sqrt{3}}{3} \\ \frac{3\sqrt{3}}{4} + \frac{4}{3} \\ 1 \end{bmatrix}$$

## Exercise 6

Για να δημιουργήσουμε το local σύστημα πρέπει να περιστρέψουμε τους άξονες κατά  $30^\circ$  και να μεταφέρουμε την αρχή των αξόνων στο  $(2, -1)$ . Επομένως για να μετατρέψουμε τα σημεία του global συστήματος θα πρέπει να εφαρμόσουμε περιστροφή κατά  $-30^\circ$  και έπειτα να τα μετατοπίσουμε κατά  $(-2, 1)$ . Δηλαδή:

$$M = T(-2, 1) \times R(-30^\circ) = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & -2 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Όμως θέλουμε τον πίνακα για την μετατροπή σημείων από το local στο global σύστημα. Επόμενως θέλουμε τον  $M^{-1}$ , δηλαδή:

$$M^{-1} = R(30^\circ) \times T(2, -1) = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & \sqrt{3} + \frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 1 - \frac{\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Για να μετατρέψουμε το σημείο  $P' = (1, 2)$  από το local στο global όχι το πολλαπλασιάσουμε με τον  $M^{-1}$ , δηλαδή:

$$M^{-1} \times \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} & \sqrt{3} + \frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} & 1 - \frac{\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{3\sqrt{3}}{2} - \frac{1}{2} \\ \frac{3}{2} + \frac{\sqrt{3}}{2} \\ 1 \end{bmatrix}$$

Για να μετατρέψουμε το σημείο  $P = (1, 1)$  από το global στο local όχι το πολλαπλασιάσουμε με τον  $M$ , δηλαδή:

$$M \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & -2 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}+1}{2} - 2 \\ \frac{\sqrt{3}-1}{2} + 1 \\ 1 \end{bmatrix}$$

## Exercise 7

Ο πίνακας  $M$  όχι είναι ο εξής:

$$\begin{aligned} M &= T(3, 0, 2) \times R_y(45^\circ) \times S(1, 2, 1) \\ &= \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 3 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 3 \\ 0 & 2 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Εφαρμόζοντας τον πίνακα  $M$  στο σημείο  $V = (1, 1, 1)$  παίρνουμε

$$M \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 3 \\ 0 & 2 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2} + 3 \\ 2 \\ 2 \\ 1 \end{bmatrix}$$

## Exercise 8

Για να δημιουργήσουμε το local σύστημα πρέπει να περιστρέψουμε το σύστημα αρχικά κατά  $30^\circ$  γύρω από τον άξονα  $z$  και έπειτα κατά  $45^\circ$  γύρω από τον άξονα  $x$ . Επίσης πρέπει να μεταφέρουμε την αρχή των αξόνων στο  $(1, 2, 3)$ , δηλαδή να μετατοπίσουμε όλα τα υπόλοιπα σημεία κατά  $(-1, -2, -3)$ . Δηλαδή ο πίνακας για την μετατροπή από το global στο local όχι είναι:

$$M = T(-1, -2, -3) \times R_x(45^\circ) \times R_z(30^\circ)$$

Επομένως ο πίνακας που πρέπει να εφαρμοστεί στα σημεία από το local για να τα μετατρέψουμε στο global σύστημα όχι είναι ο  $M^{-1}$ , για τον οποίο ισχύει:

$$\begin{aligned}
M^{-1} &= R_z(-30^\circ) \times R_x(-45^\circ) \times T(1, 2, 3) \\
&= \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{4} & \frac{1}{4} & 0 \\ -\frac{1}{2} & \frac{3}{4} & \frac{\sqrt{3}}{2} & 0 \\ 0 & -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{4} & \frac{1}{4} & \frac{\sqrt{3} + \frac{3}{4}}{4} \\ -\frac{1}{2} & \frac{3}{4} & \frac{\sqrt{3}}{2} & \frac{3\sqrt{3} + 1}{2} \\ 0 & -\frac{1}{2} & \frac{\sqrt{3}}{2} & \frac{3\sqrt{3} - 1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Για να μετατρέψουμε το σημείο  $P' = (2, 0, -1)$  σε global συντεταγμένες ως το πολλαπλασιάσουμε με τον  $M^{-1}$ , δηλαδή:

$$M \times \begin{bmatrix} 2 \\ 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{4} & \frac{1}{4} & \frac{\sqrt{3} + \frac{3}{4}}{4} \\ -\frac{1}{2} & \frac{3}{4} & \frac{\sqrt{3}}{2} & \frac{3\sqrt{3} + 1}{2} \\ 0 & -\frac{1}{2} & \frac{\sqrt{3}}{2} & \frac{3\sqrt{3} - 1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2\sqrt{3} + \frac{1}{2} \\ \sqrt{3} \\ \sqrt{3} - 1 \\ 1 \end{bmatrix}$$

## Exercise 9

### 1. Algorithm Development

Στην υλοποίηση του αλγορίθμου αξιοποιούνται οι δομές activeEdgeTable και activeEdgeList, οι οποίες είναι ορισμένες ως `vector<vector<Edge<int>>` και `vector<Edge<int>`, αντίστοιχα. Αρχικά, ο αλγόριθμος κατασκευάζει το activeEdgeTable τοποθετώντας κάθε edge στη θέση `edge.getMinY()` - `minYpoly`, έτσι ώστε κάθε εγγραφή του πίνακα να αντιστοιχεί στα edges που “ενεργοποιούνται” στη συγκεκριμένη τιμή y. Με τον τρόπο αυτό, η δομή λειτουργεί ως πίνακας οργάνωσης των σημείων έναρξης των edges κατά μήκος της κατακόρυφης σάρωσης των scanlines.

Στη συνέχεια, για κάθε τιμή y, κατασκευάζεται το activeEdgeList, το οποίο περιέχει όλα τα edges που είναι ενεργά για την τρέχουσα scanline. Αφού δημιουργηθεί το activeEdgeList, ο αλγόριθμος καλεί τη συνάρτηση `fillLine()`, η οποία αναλαμβάνει να γεμίσει όλα τα σημεία μεταξύ των διαδοχικών ζευγών edges. Εξαίρεση αποτελεί το τελευταίο ζευγάρι edges ανά scanline, το οποίο χρειάζεται ειδική μεταχείριση ώστε να αποφευχθεί η σχεδίαση σημείων που βρίσκονται ακριβώς πάνω στο δεύτερο edge, αποτρέποντας έτσι την εμφάνιση επικαλύψεων.

---

**Algorithm 1** Fill Algorithm

---

```
1: minYpoly ← min(poly.getVertices().Y)
2: maxYpoly ← max(poly.getVertices().Y)
3: numScanlines ← maxYpoly – minYpoly + 1
4:
5: activeEdgeTable ← array of vectors of size numScanlines
6: for (each edge  $e$  of the polygon) do
7:   if  $e.\text{maxY} = e.\text{minY}$  then
8:     continue
9:   end if
10:  activeEdgeTable[ $e.\text{minY} - \text{minYpoly}$ ].insert( $e$ )
11: end for
12:
13: activeEdgeList ←  $\emptyset$ 
14: for ( $i \leftarrow 0$  to numScanlines – 1) do
15:   for each edge  $e$  in activeEdgeList do
16:     if  $e.\text{maxY} = i + \text{minYpoly}$  then
17:       activeEdgeList.remove( $e$ )
18:     end if
19:   end for
20:
21: activeEdgeList.insertAll(activeEdgeTable[i])
22: fillCurrentLine( $i + \text{minYpoly}$ )
23:
24: for each edge  $e$  in activeEdgeList do
25:    $e.\text{incrementX}()$ 
26: end for
27: end for
28:
29: procedure FILLLINE( $y$ )
30:   edges ← activeEdgeList
31:   for (each successive pair of edges  $(e_1, e_2)$  except last) do
32:     draw horizontal line at  $y$  from  $x = \lceil e_1.\text{currentX} \rceil$  to  $x = \lfloor e_2.\text{currentX} \rfloor$ 
33:   end for
34:   draw horizontal line at  $y$  from  $x = \lceil e_1.\text{currentX} \rceil$  to  $x = \lfloor e_2.\text{currentX} \rfloor - 1$ 
35: end procedure
```

---

## 2. Color Interpolation Extension

---

**Algorithm 2** Fill Algorithm with color interpolation

---

```

1: minYpoly ← min(poly.getVertices().Y)
2: maxYpoly ← max(poly.getVertices().Y)
3: numScanlines ← maxYpoly – minYpoly + 1
4:
5: activeEdgeTable ← array of vectors of size numScanlines
6: for (each edge  $e$  of the polygon) do
7:   if  $e.\text{maxY} = e.\text{minY}$  then
8:     continue
9:   end if
10:  activeEdgeTable[ $e.\text{minY} - \text{minYpoly}$ ].insert( $e$ )
11: end for
12:
13: activeEdgeList ←  $\emptyset$ 
14: for ( $i \leftarrow 0$  to numScanlines – 1) do
15:   for each edge  $e$  in activeEdgeList do
16:     if  $e.\text{maxY} = i + \text{minYpoly}$  then
17:       activeEdgeList.remove( $e$ )
18:     end if
19:   end for
20:
21: activeEdgeList.insertAll(activeEdgeTable[i])
22: fillCurrentLine( $i + \text{minYpoly}$ , activeEdgeList)
23:
24: for each edge  $e$  in activeEdgeList do
25:    $e.\text{incrementX}()$ 
26:    $e.\text{incrementColor}()$ 
27: end for
28: end for
29:
30: procedure FILLCURRENTLINE( $y$ , activeEdgeList)
31:    $N \leftarrow \text{activeEdgeList.size}$ 
32:
33:   for  $i \leftarrow 0$  to  $N - 2$  step 2 do
34:      $e_1 \leftarrow \text{activeEdgeList}[i]$ 
35:      $e_2 \leftarrow \text{activeEdgeList}[i + 1]$ 
36:      $x_{\text{start}} \leftarrow \lceil e_1.\text{currentX} \rceil$ 
37:     if  $i = N - 2$  then                                 $\triangleright$  Use specific boundary logic for the last pair vs. others
38:        $x_{\text{end}} \leftarrow \lceil e_2.\text{currentX} \rceil - 1$ 
39:     else
40:        $x_{\text{end}} \leftarrow \lfloor e_2.\text{currentX} \rfloor$ 
41:     end if
42:     width ←  $x_{\text{end}} - x_{\text{start}}$ 
43:     if width > 0 then
44:        $C \leftarrow e_1.\text{color}$ 
45:        $\Delta C \leftarrow (e_2.\text{color} - e_1.\text{color})/\text{width}$ 
46:       for  $x \leftarrow x_{\text{start}}$  to  $x_{\text{end}}$  do
47:         DRAWPIXEL( $x, y, C$ )
48:          $C \leftarrow C + \Delta C$ 
49:       end for
50:     end if
51:   end for
52: end procedure

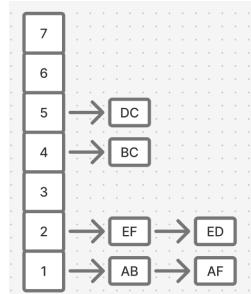
```

---

Στη μονοχρωματική υλοποίηση, κάθε pixel λάμβανε την ίδια τιμή RGB, η οποία αντιστοιχούσε κοινά σε όλα τα vertices. Για να υποστηριχθεί color interpolation, επεκτείναμε αυτή τη λογική ενσωματώνοντας στα edges γραμμική παρεμβολή χρώματος τόσο κατά μήκος του άξονα Y όσο και κατά μήκος του άξονα X. Πιο συγκεκριμένα, τροποποιήσαμε τον τρόπο με τον οποίο ενημερώνεται το χρώμα στις κλίσεις της glVertex, αξιοποιώντας βοηθητικές συναρτήσεις όπως getCurrentColor() και incrementColor(). Η μεταβολή του χρώματος υπολογίζεται γραμμικά με βάση τη σχέση  $\text{delta} = \frac{\text{colorEnd}-\text{colorStart}}{\#edgeHeight}$ , ώστε να αποφεύγονται περιττοί πολλαπλασιασμοί. Αρχικά πραγματοποιύμε παρεμβολή κατά μήκος του άξονα Y για κάθε active edge, δηλαδή κάθε edge στο ActiveEdgeList, ώστε για το εκάστοτε για το προσδιορίζεται το ενδιάμεσο χρώμα του σημείου που αντιστοιχεί στη θέση τομής της scanline με το edge, τα οποία θα αντιστοιχηθούν στα colorEnd, colorStart για χρήση στη παρεμβολή του άξονα X. Στη συνέχεια, κατά τη διάρκεια σχεδίασης της scanline, εφαρμόζουμε δεύτερη γραμμική παρεμβολή χρώματος κατά μήκος του άξονα X, χρησιμοποιώντας τα προαναφερθέντα χρώματα colorEnd, colorStart. Με αυτόν τον τρόπο υπολογίζεται το χρώμα κάθε σημείου του πολυγώνου.

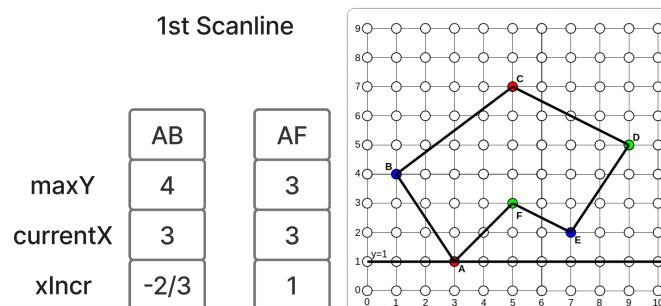
### 3. Example Test Case

Αρχικοποιούμε το active edge table, όπως φαίνεται και στην παρακάτω εικόνα.

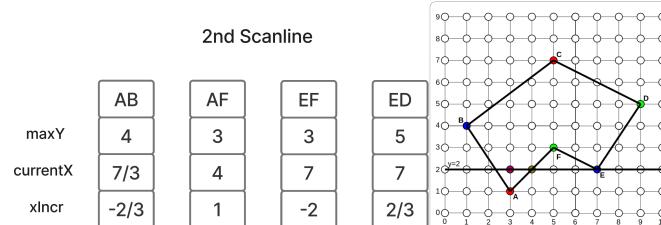


Αφού βρούμε το active edge table, προχωράμε και δημιουργούμε το active edge list.

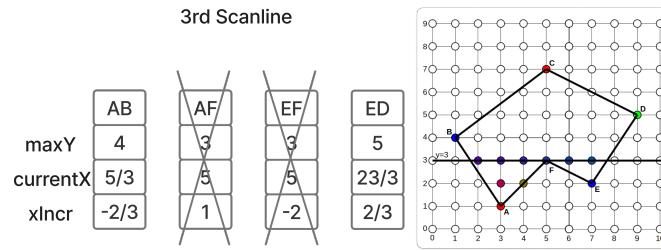
1<sup>η</sup> Scanline:



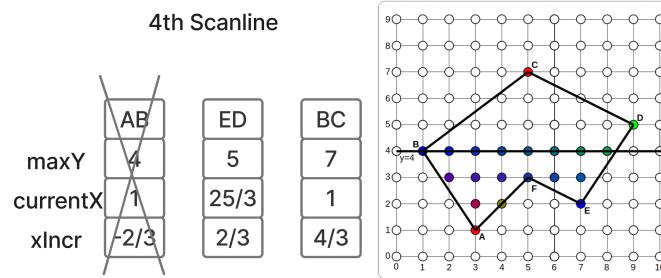
2<sup>η</sup> Scanline:



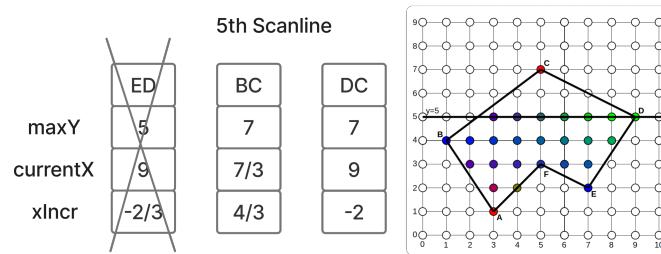
3<sup>η</sup> Scanline:



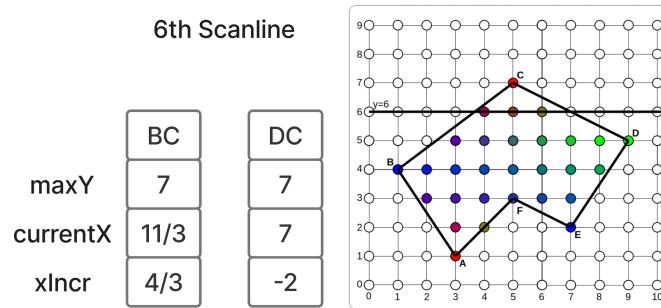
4<sup>n</sup> Scanline:



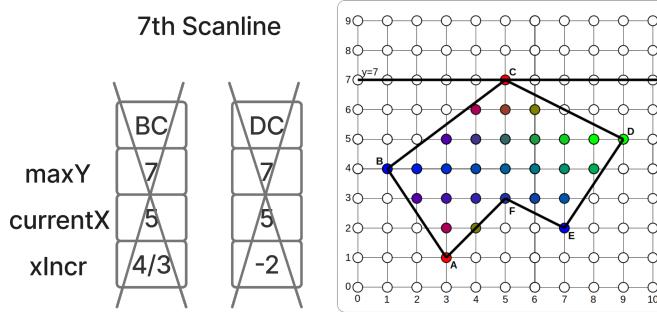
5<sup>n</sup> Scanline:



6<sup>n</sup> Scanline:



7<sup>η</sup> Scanline:



## Exercise 10

### 1. Algorithm Development

---

#### Algorithm 3 Sutherland-Hodgman Algorithm

---

```

1: oldPoints ← toFloat(poly.getVertices())
2: newPoints ← ∅
3: for (each boundary in {LEFT, RIGHT, BOTTOM, TOP}) do
4:   for (each curPoint in oldPoints) do
5:     curEdge ← (curPoint, curPoint+1)
6:     state ← ClippingWindow.getEdgeState(boundary, curEdge)
7:     Switch state do
8:       Case IN_IN :
9:         insertFront(newPoints, curPoint+1)
10:      Case IN_OUT :
11:        insertFront(newPoints, interceptPoint)
12:      Case OUT_IN :
13:        insertFront(newPoints, interceptPoint)
14:        insertFront(newPoints, curPoint+1)
15:      Case OUT_OUT :
16:        /* insert nothing */
17:    end for
18:    oldPoints ← newPoints
19:    newPoints ← ∅
20: end for
21: draw(toInt(oldPoints))

```

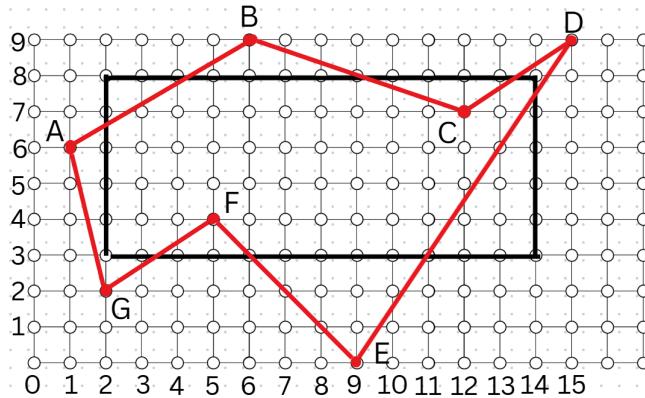
---

Η υλοποίηση του αλγορίθμου βασίζεται στη χρήση δύο δομών τύπου vector, οι οποίες λειτουργούν ως FIFO Queue. Συγκεκριμένα, το oldPoints περιέχει τα σημεία προς επεξεργασία, ενώ το newPoints αποθηκεύει τα τροποποιημένα σημεία που προκύπτουν από την επεξεργασία κάθε WindowEdge. Για κάθε edge του παραθύρου, ο αλγόριθμος λαμβάνει διαδοχικά τα σημεία του oldPoints και τα μετασχηματίζει σύμφωνα με τις τέσσερις πιθανές καταστάσεις που προκύπτουν από τη σχέση μεταξύ του edge (point, point->next) και του αντίστοιχου WindowEdge. Τα επεξεργασμένα σημεία εισάγονται στο newPoints, ενώ η διαδικασία ολοκληρώνεται όταν όλα τα σημεία έχουν εξεταστεί για κάθε WindowEdge.

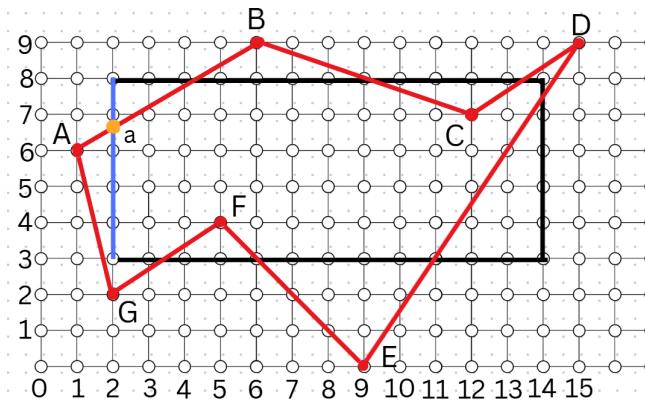
Επιπλέον, αξιοποιούνται οι βοηθητικές συναρτήσεις toFloat() και toInt(), οι οποίες συμβάλλουν στη μείωση του αριθμητικού σφάλματος κατά τον υπολογισμό των σημείων που τέμνονται με το Window. Το ζήτημα αυτό καθίσταται ιδιαίτερα σημαντικό σε περιπτώσεις όπου ένα σημείο υφίσταται επαναλαμβανόμενη επεξεργασία από διαδοχικά WindowEdges, με αποτέλεσμα την σταδιακή συσσώρευση σφάλματος στις τελικές του συντεταγμένες.

## 2. Example Test Case

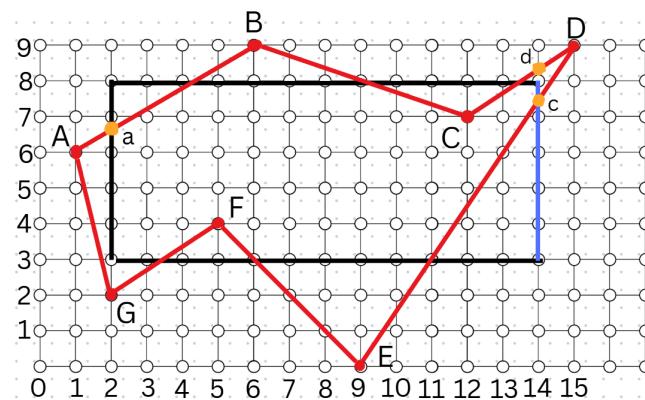
Το αρχικό πολύγωνο έχει την παρακάτω μορφή:



Η αρχική λίστα των κορυφών είναι  $GFEDCBA$ , πρωτού πραγματοποιηθεί ο αλγόριθμος Sutherland-Hodgman. Ο αλγόριθμος πρώτα ελέγχει σε σχέση με το αριστερό σύνορο του clipping window. Όλες οι ακμές είναι μέσα, εκτός από τις ακμές  $AB$ ,  $AG$ . Για την ακμή  $AB$  βρίσκουμε το intersection της ακμής με το αριστερό σύνορο και επειδή η ακμή πάει από έξω προς τα μέσα, αποθηκεύουμε και το intersection point αλλά και το σημείο  $B$ . Η ακμή  $AG$  είναι εκτός clipping window, οπότε δεν αποθηκεύουμε τίποτα. Η τροποποιημένη λίστα είναι  $GFEDCBa$ .

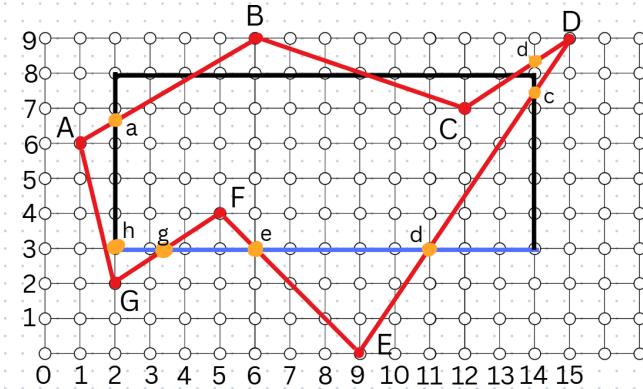


Στην συνέχεια ελέγχουμε σε σχέση με το δεξιό σύνορο του clipping window. Όλες οι ακμές είναι μέσα, εκτός από τις  $CD$  και  $DE$ . Για την ακμή  $CD$  επειδή πηγαίνει από μέσα προς τα έξω αποθηκεύουμε μόνο το intersection point. Για την ακμή  $DE$  επειδή πηγαίνει από έξω προς τα μέσα αποθηκεύουμε το intersection point και το σημείο  $E$ . Η τροποποιημένη λίστα είναι  $aGFEcbCB$ .

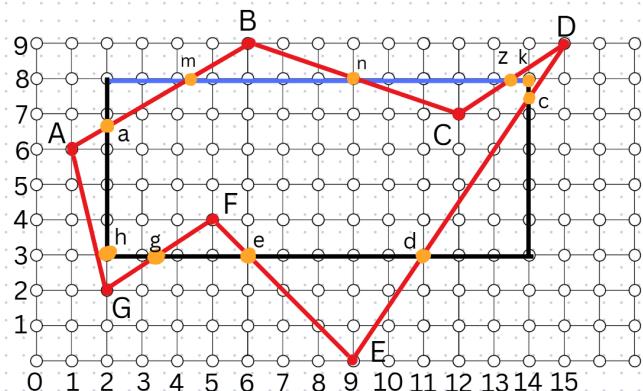


Έχοντας ελέγξει το αριστερό και δεξιό σύνορο, προχωράμε στο κάτω σύνορο του clipping window. Οι ακμές που δεν είναι μέσα είναι οι  $cE$ ,  $EF$ ,  $FG$ ,  $Ga$ . Οι ακμές  $cE$  και  $FG$  πηγαίνουν από μέσα προς τα έξω, συνεπώς αποθηκεύουμε μόνο τα intersection points. Οι ακμές  $EF$  και  $Ga$  πηγαίνουν από έξω προς τα μέσα,

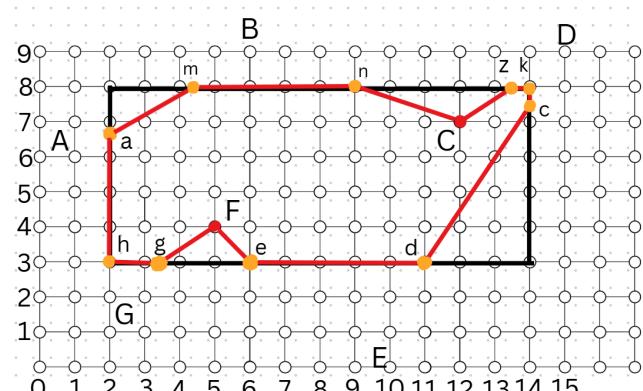
οπότε αποθηκέυουμε και τα intersection points αλλά και τα σημεία  $F$  και  $a$  αντίστοιχα. Η νέα λίστα είναι  $BahgFedcbC$ .



Τέλος, προχωράμε στο πάνω σύνορο του clipping window. Οι ακμές που δεν είναι μέσα είναι οι  $aB$ ,  $BC$ ,  $Cd$ ,  $dc$ . Οι ακμές  $BC$  και  $dc$  πηγαίνουν από έξω προς τα μέσα, επομένως αποθηκέυουμε τα intersection points αυτών των ακμών με το clipping window και τα σημεία  $C$ ,  $c$  αντίστοιχα. Οι ακμές  $aB$ ,  $Cd$  πηγαίνουν από μέσα προς τα έξω, άρα αποθηκέυουμε μόνο τα intersection points. Η τελική λίστα κορυφών αφού έχει πραγματοποιηθεί ο αλγόριθμος είναι  $CnmahgFedckz$ .



Παρακάτω μπορούμε να δούμε το τελικό πολύγωνο, μετά το πέρασμα του αλγορίθμου.



## Exercise 11

### Implementation structure and algorithm integration

Η υλοποίηση έχει σχεδιαστεί χρησιμοποιώντας αντικειμενοστραφή προγραμματισμό για να διασφαλιστεί ο διαχωρισμός των αρμοδιοτήτων. Ο κάθισκας οργανώνεται σε τέσσερα κύρια μέρη:

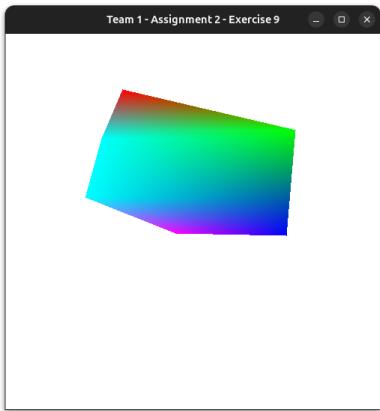
- **Κύρια Εφαρμογή (exercise11.cpp):** Λειτουργεί ως το σημείο εισόδου. Διαχειρίζεται την αρχικοποίηση του OpenGL/GLUT, χειρίζεται την είσοδο του χρήστη και ελέγχει τον κύριο βρόχο απεικόνισης (display). Αναθέτει τις γεωμετρικές λειτουργίες στην κλάση Polygon.
- **Κλάση Polygon (polygon.cpp, polygon.h):** Αυτή η κλάση ενσωματώνει όλη τη λογική που σχετίζεται με τη δομή δεδομένων του πολυγώνου και τον αλγόριθμο γεμίσματος. Διατηρεί μια global λίστα όλων των πολυγώνων (vector<Polygon> polys) και διαχειρίζεται την αποθήκευση των κορυφών. Είναι υπεύθυνη για την εκτέλεση του αλγορίθμου σάρωσης γραμμών μέσω της μεθόδου fill().
- **Κλάση Edge (edge.cpp, edge.h):** Είναι μια βοηθητική κλάση για τον αλγόριθμο scanline. Αναπαριστά μια ακμή του πολυγώνου και αποθηκεύει τα απαραίτητα δεδομένα για τους υπολογισμούς, όπως το ελάχιστο και μέγιστο Y, το τρέχον X, το τρέχον χρώμα RGB και τις αντίστοιχες τιμές αύξησης (xIncrement, rIncrement κ.λπ.) που απαιτούνται για την παρεμβολή.
- **Utilities (utilities.h):** Ορίζει κοινές δομές όπως Point και RGB που χρησιμοποιούνται σε όλη την εφαρμογή.

### User interaction and color handling

Η αλληλεπίδραση με τον χρήστη πραγματοποιείται μέσω των συναρτήσεων επανωλήσεως (callbacks) του GLUT που έχουν καταχωρηθεί στη main (glutMouseFunc και glutKeyboardFunc).

- **Δημιουργία Πολυγώνου (Ποντίκι):**
  - **Αριστερό Κλικ (LMB):** Καλεί την Polygon::getCurrentOrCreate()->addVertex(). Αυτό προσθέτει μια κορυφή στη θέση του κλικ με το τρέχον επιλεγμένο χρώμα.
  - **Δεξιό Κλικ (RMB):**
    - \* Εάν σχεδιάζεται ένα πολύγωνο, καλεί την poly->addLastVertex(), η οποία προσθέτει το τελικό σημείο, επισημάνει το πολύγωνο ως ολοκληρωμένο (complete) και ενεργοποιεί τον αλγόριθμο γεμίσματος.
    - \* Εάν δεν υπάρχει ενεργό πολύγωνο (δηλ. το προηγούμενο έχει ολοκληρωθεί), καλεί την Polygon::clear(), η οποία καθαρίζει τον πίνακα (αφαιρεί όλα τα πολύγωνα).
- **Επιλογή Χρώματος (Πληκτρολόγιο):** Η keyboardHandler περιμένει συγκεκριμένους χαρακτήρες και ενημερώνει την μεταβλητή curColor. Αυτό το χρώμα ανατίθεται στην επόμενη κορυφή που θα τοποθετήσει ο χρήστης:
  - **R / r:** Κόκκινο, **G / g:** Πράσινο, **B / b:** Μπλε
  - **C / c:** Κυανό, **M / m:** Ματζέντα, **Y / y:** Κίτρινο
  - **Q / q:** Τερματίζει την εφαρμογή.
- **Επανασχεδιασμός:** Κάθε φορά που συμβαίνει ένα event εισόδου (κλικ ή πάτημα πλήκτρου), καλείται η glutPostRedisplay(). Η συνάρτηση display() στη συνέχεια επαναλαμβάνει τη διαδικασία για όλα τα πολύγωνα και καλεί τη μέθοδό τους draw(). Εάν ένα πολύγωνο είναι ατελές, σχεδιάζει τις κορυφές, ενώ εάν είναι ολοκληρωμένο, εκτελεί το γέμισμα.

## Screenshots of the Implementation:



Σχήμα 6: Ο αλγόριθμος fill.

## Exercise 12

### Implementation structure and algorithm integration

Η υλοποίηση χρησιμοποιεί τη δομή της προηγούμενης άσκησης, εισάγοντας νέες κλάσεις για τη διαχείριση του παραθύρου αποκοπής. Ο κώδικας οργανώνεται στα εξής μέρη:

- **Κύρια Εφαρμογή (exercise12.cpp):** Διαχειρίζεται την κατάσταση της εφαρμογής (SelectingState), εναλλάσσοντας μεταξύ λειτουργίας σχεδίασης πολυγώνου και ορισμού παραθύρου αποκοπής. Ελέγχει την είσοδο του χρήστη και τον κώδικα OpenGL.
- **Κλάση ClippingWindow (clippingWindow.cpp, clippingWindow.h):** Μια νέα κλάση που αναπαριστά το ορθογώνιο παράθυρο αποκοπής. Περιέχει τη λογική για τον αλγόριθμο αποκοπής και διαχειρίζεται τον υπολογισμό των τομών με τις πλευρές του παραθύρου.
- **Κλάση Polygon (polygon.cpp, polygon.h):** Παρόμοια με την Άσκηση 11, αλλά προσαρμοσμένη ώστε να υποστηρίζει την αντικατάσταση των κορυφών της με τις νέες, αποκομμένες κορυφές που παράγονται από τη διαδικασία αποκοπής.
- **Κλάση Edge (edge.cpp, edge.h):** Η κλάση έχει μετατραπεί σε πρότυπο (template) ώστε να υποστηρίζει τόσο ακέραιες συντεταγμένες (για την οθόνη) όσο και αριθμούς κινητής υποδιαστολής (για ακριβείς υπολογισμούς τομών κατά την αποκοπή).

### User interaction handling

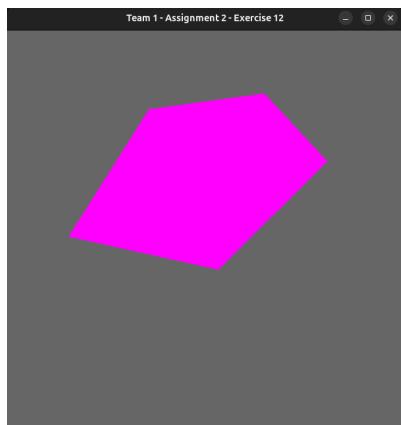
Η εφαρμογή διαθέτει δύο καταστάσεις λειτουργίας: **Polygon Drawing** και **Clipping Window**. Η αλληλεπίδραση ορίζεται ως εξής:

- **Εναλλαγή Λειτουργίας (F1):** Ο χρήστης πατά το πλήκτρο F1 για να μεταβεί από τη σχεδίαση πολυγώνων στον ορισμό του παραθύρου αποκοπής και αντίστροφα.
- **Λειτουργία Πολυγώνου:**
  - **Αριστερό Κλικ:** Προσθέτει κορυφές στο πολύγωνο.
  - **Δεξί Κλικ:** Ολοκληρώνει το πολύγωνο.
- **Λειτουργία Παραθύρου:**
  - **Αριστερό Κλικ & Σύρσιμο (Drag):** Ο χρήστης ορίζει το παράθυρο αποκοπής πατώντας και σύροντας το ποντίκι. Το παράθυρο ενημερώνεται δυναμικά στην οθόνη μέσω της glutMotionFunc.

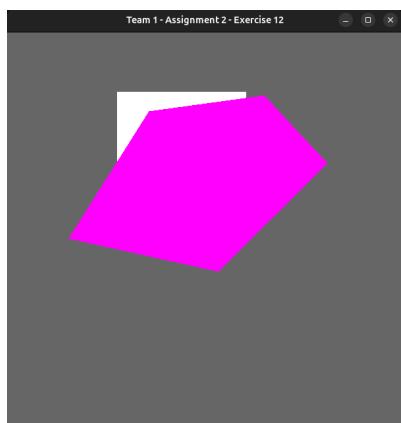
- **Πληκτρολόγιο:**

- **R / r (Run):** Εκτελεί την αποκοπή (clipSelection) στο υπάρχοντα πολύγωνα με βάση το ενεργό παράθυρο.
- **F / f (Fill):** Εναλλάσσει το γέμισμα του πολυγώνου (αυξομειώνοντας το glPointSize ή ενεργοποιώντας τον αλγόριθμο γεμίσματος της Άσκησης 11).
- **C / c (Clear):** Καθαρίζει την οθόνη, διαγράφοντας τόσο τα πολύγωνα όσο και το παράθυρο αποκοπής.
- **Q / q (Quit):** Τερματίζει την εφαρμογή.

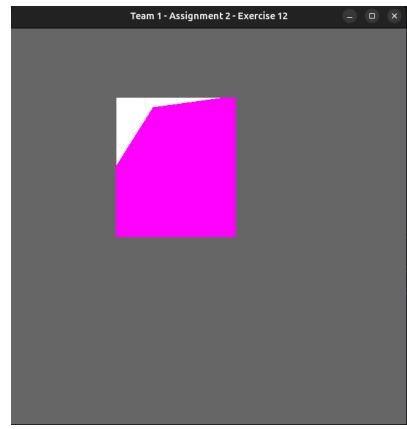
### Screenshots of the Implementation:



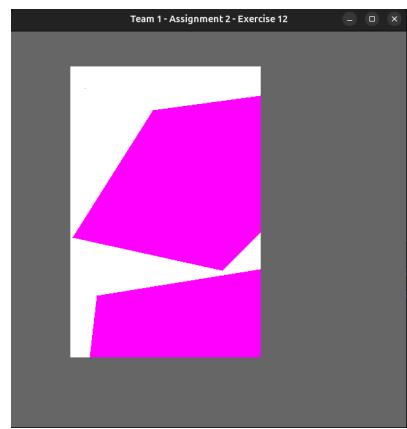
Σχήμα 7: Σχεδιασμός του πολυγώνου.



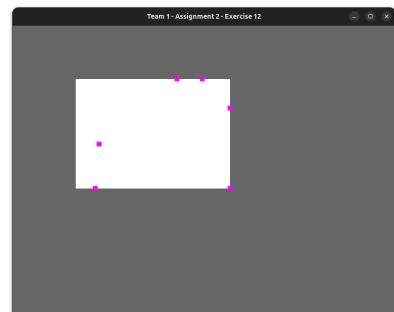
Σχήμα 8: Επιλογή του Clipping Window.



Σχήμα 9: Εφαρμόζοντας το clipping.



Σχήμα 10: Κάνοντας clip παραπάνω από ένα πολύγωνο.



Σχήμα 11: Πατώντας 'f' εμφανίζουμε τα σημεία αντί για το ολόκληρο γεμάτο πολύγωνο.

## Specifications

Για να εκτελέσουμε και να ελέγξουμε τις υλοποιήσεις μας χρησιμοποιήσαμε το ακόλουθο σύστημα:

- CPU: AMD Ryzen 7 5800X 8-Core Processor
- Caches (sum of all):
  - L1d: 256 KiB (8 instances)
  - L1i: 256 KiB (8 instances)
  - L2: 4 MiB (8 instances)
  - L3: 32 MiB (1 instance)
- Memory: 32GB DDR4
- GPU: GeForce RTX 3060 Lite Hash Rate
- Compiler version: gcc (Ubuntu 13.3.0-6ubuntu2 24.04) 13.3.0
- Operating System: Ubuntu LTS 24.04

# Appendix

## Code of Exercise 11

Here is our program for exercise 11. File name: polygon.cpp

```
1 void Polygon::fill() {
2     initActiveEdgeTable();
3
4     int yStart = getMinY();
5     int offset = 0;
6     for (const std::vector<Edge>& curEdgeList : activeEdgeTable) {
7         int scanlineY = yStart + offset;
8
9         std::vector<Edge>::iterator curEdgeIt = activeEdgeList.begin();
10        while (curEdgeIt != activeEdgeList.end()) {
11            if (curEdgeIt->getMaxY() == scanlineY)
12                curEdgeIt = activeEdgeList.erase(curEdgeIt);
13            else
14                ++curEdgeIt;
15        }
16
17        for (const Edge& newEdge : curEdgeList) {
18            activeEdgeList.push_back(newEdge);
19        }
20
21        std::sort(activeEdgeList.begin(), activeEdgeList.end(),
22                  [] (const Edge& a, const Edge& b) {
23                      return a.getCurrentX() < b.getCurrentX();
24                  });
25
26        fillLine(scanlineY);
27
28        for (Edge& newEdge : activeEdgeList) {
29            newEdge.incrementX();
30            newEdge.incrementColor();
31        }
32        offset++;
33    }
34 }
35 }
```

Listing 1: fill function of exercise 11.

```
1 void Polygon::fillLine(int y) {
2     int numEdges = (int) activeEdgeList.size();
3     if (numEdges < 2) return;
4
5     for (int i = 0; i < numEdges - 2; i += 2) {
6         Edge& startEdge = activeEdgeList[i];
7         Edge& endEdge = activeEdgeList[i + 1];
8         RGB startColor = startEdge.getCurrentColor();
9         RGB endColor = endEdge.getCurrentColor();
10        int x1 = (int) ceil(startEdge.getCurrentX());
11        int x2 = (int) floor(endEdge.getCurrentX());
12        if (x1 > x2) continue;
13
14        RGB currentColor = startColor;
15        float rIncr = (float) (endColor.red - startColor.red) / (x2 - x1);
16        float gIncr = (float) (endColor.green - startColor.green) / (x2 - x1);
17        float bIncr = (float) (endColor.blue - startColor.blue) / (x2 - x1);
18
19        glBegin(GL_POINTS);
```

```

20     for (int j = x1; j <= x2; j++) {
21         glColor3f(currentColor.red, currentColor.green, currentColor.blue)
22     ;
23         glVertex2i(j, y);
24
25         currentColor.red += rIncr;
26         currentColor.green += gIncr;
27         currentColor.blue += bIncr;
28     }
29     glEnd();
30 }
31
32 Edge& startEdge = activeEdgeList[numEdges - 2];
33 Edge& endEdge = activeEdgeList[numEdges - 1];
34 RGB startColor = startEdge.getCurrentColor();
35 RGB endColor = endEdge.getCurrentColor();
36 int x1 = (int) ceil(startEdge.getCurrentX());
37 int x2 = ((int) ceil(endEdge.getCurrentX())) - 1;
38 if (x1 > x2) return;
39
40 RGB currentColor = startColor;
41 float rIncr = (float) (endColor.red - startColor.red) / (x2 - x1);
42 float gIncr = (float) (endColor.green - startColor.green) / (x2 - x1);
43 float bIncr = (float) (endColor.blue - startColor.blue) / (x2 - x1);
44
45 glBegin(GL_POINTS);
46 for (int j = x1; j <= x2; j++) {
47     glColor3f(currentColor.red, currentColor.green, currentColor.blue);
48     glVertex2i(j, y);
49
50     currentColor.red += rIncr;
51     currentColor.green += gIncr;
52     currentColor.blue += bIncr;
53 }
54 glEnd();
}

```

Listing 2: fillLine function of exercise 11.

## Code of Exercise 12

Here is our program for exercise 12. File name: clippingWindow.cpp

```

1 void ClippingWindow::clipSelection() {
2     active = true;
3     vector<Polygon> polys = Polygon::getPolys();
4
5     for (const Polygon& curPoly : polys) {
6         vector<Point<float>> newPoints;
7         vector<Point<float>> oldPoints = toFloat(curPoly.getVertices());
8
9         for (WindowEdge boundary = LEFT; boundary <= TOP; ++boundary) {
10             if (oldPoints.empty()) break;
11
12             for (vector<Point<float>>::iterator curPoint = oldPoints.begin();
13 curPoint != oldPoints.end() - 1; ++curPoint) {
14                 switch (getState(boundary, Edge<float>(*curPoint, *(curPoint +
15 1)))) {
16                     case IN_IN: {
17                         newPoints.push_back(*(curPoint + 1));
18                         break;
19                     }
20                 }
21             }
22         }
23     }
24 }

```

```

18         case IN_OUT: {
19             newPoints.push_back(intersectEdge(boundary, Edge<float>(*curPoint, *(curPoint + 1))));
20             break;
21         }
22         case OUT_IN: {
23             newPoints.push_back(intersectEdge(boundary, Edge<float>(*curPoint, *(curPoint + 1))));
24             newPoints.push_back(*(curPoint + 1));
25             break;
26         }
27         default:
28             break;
29     }
30 }
31
32 Point<float> front = oldPoints.front();
33 Point<float> back = oldPoints.back();
34 switch (getState(boundary, Edge<float>(back, front))) {
35     case IN_IN: {
36         newPoints.push_back(front);
37         break;
38     }
39     case IN_OUT: {
40         newPoints.push_back(intersectEdge(boundary, Edge<float>(back, front)));
41         break;
42     }
43     case OUT_IN: {
44         newPoints.push_back(intersectEdge(boundary, Edge<float>(back, front)));
45         newPoints.push_back(front);
46         break;
47     }
48     default:
49         break;
50 }
51
52 std::reverse(newPoints.begin(), newPoints.end());
53 oldPoints = std::move(newPoints);
54 }
55
56 vector<Point<int>> finalPoints = toInteger(oldPoints);
57
58 Polygon *newPolygon = Polygon::getCurrentOrCreate(true);
59 for (const Point<int>& curPoint : finalPoints) {
60     newPolygon->addVertex(curPoint);
61 }
62 newPolygon->finish();
63 }
}

```

Listing 3: clipSelection function of exercise 12.

```

1 Point<float> ClippingWindow::intersectEdge(WindowEdge boundary, Edge<float>
edge) {
2     Point<float> intersectPoint;
3     Point<float> start = edge.getStart();
4     Point<float> end = edge.getEnd();
5     intersectPoint.rgb = start.rgb;
6
7     float slope = 0.0f;

```

```

8     bool vertical = (start.x == end.x);
9     if (!vertical) {
10         slope = (end.y - start.y) / (end.x - start.x);
11     }
12
13     switch (boundary) {
14         case LEFT: {
15             intersectPoint.x = static_cast<float>(getMinX());
16             intersectPoint.y = vertical
17                 ? start.y
18                 : start.y + slope * (intersectPoint.x - start.x);
19             break;
20         }
21         case RIGHT: {
22             intersectPoint.x = static_cast<float>(getMaxX());
23             intersectPoint.y = vertical
24                 ? start.y
25                 : start.y + slope * (intersectPoint.x - start.x);
26             break;
27         }
28         case BOT: {
29             intersectPoint.y = static_cast<float>(getMinY());
30             intersectPoint.x = (slope == 0.0f)
31                 ? start.x
32                 : start.x + ((intersectPoint.y - start.y) / slope);
33             break;
34         }
35         case TOP: {
36             intersectPoint.y = static_cast<float>(getMaxY());
37             intersectPoint.x = (slope == 0.0f)
38                 ? start.x
39                 : start.x + ((intersectPoint.y - start.y) / slope);
40             break;
41         }
42     }
43
44     return intersectPoint;
45 }
```

Listing 4: intersectEdge function of exercise 12.