Γραφική με Υπολογιστές - Πλήρωση Τριγώνων

Γκούντρας Ιωάννης

AEM:10332

3 Απριλίου, 2024

Abstract

Σε αυτήν την εργασία υλοποιούνται δύο βασικοί αλγόριθμοι πλήρωσης τριγώνων, ο Flat Shading και ο Gouraud Shading, με τη χρήση Python και των βιβλιοθηκών OpenCV και Numpy.

1 Συνάρτηση γραμμικής παρεμβολής

Η βοηθητική συνάρτηση `vector_interp(p1, p2, V1, V2, coord, dim)` είναι η πρώτη που υλοποιήθηκε. Πραγματοποιεί γραμμική παρεμβολή στο σημείο $p = (x, y)$ μεταξύ δύο διανυσμάτων V_1 και V_2 δεδομένων των σημείων $p1 = (x_1, y_1)$ και $p2 = (x_2, y_2)$ στα οποία αντιστοιχούν, δεδομένου ότι το p ανήκει στην ευθεία που ορίζουν τα σημεία. Το διάνυσμα προκύπτει ως εξής:

$$V = (1 - a)V_1 + aV_2 \tag{1}$$

Για να προσδιοριστεί το ποσοστό a χρησιμοποιούνται:

- Οι τετμημένες των σημείων αν $dim = 1$
- Οι τεταγμένες των σημείων αν $dim = 2$

Στην πρώτη περίπτωση ορίζεται ως x_1 το πιο αριστερό σημείο εκ των p_1 και p_2 και x_2 το άλλο καθώς και V_{left} το διάνυσμα που αντιστοιχεί στο x_1 και V_{right} το άλλο. Στη συνέχεια υπολογίζεται η διαφορά $x_2 - x_1$ και προκύπτει το ποσοστό

$$a = \frac{\text{coord} - x_1}{x_2 - x_1} \tag{2}$$

το οποίο χρησιμοποιείται για να παραχθεί το τελικό διάνυσμα

$$V = (1 - a)V_{left} + aV_{right} \tag{3}$$

Για τον παραπάνω υπολογισμό χρησιμοποιούνται οι συναρτήσεις `numpy.add()` και `numpy.multiply()`.

Παρόμοια στην δεύτερη περίπτωση ορίζεται ως y_1 το χαμηλότερο σημείο και V_{down} το αντίστοιχο διάνυσμα. Η διαδικασία που ακολουθεί είναι ίδια με την πρώτη περίπτωση.

Τέλος, πραγματοποιείται και στις δύο περιπτώσεις ο έλεγχος για το αν τα p_1 και p_2 έχουν την ίδια τετμημένη ή τεταγμένη αντίστοιχα, όπου επιστρέφεται κατευθείαν το V_1 ή το V_2.

2 Συνάρτηση Flat Shading

Η επόμενη συνάρτηση που υλοποιήθηκε είναι η `f_shading(img, vertices, vcolors)`. Αρχικά υπολογίζεται το χρώμα όλου του τριγώνου ως ο διανυσματικός μέσος όρος των χρωμάτων των τριών κορυφών του, από τον πίνακα `vcolors`. Στη συνέχεια αρχικοποιείται ένα dictionary για κάθε edge του τριγώνου που περιέχει:

- το όνομα του edge (π.χ. *"Edge 1"*)

- τα x_{min}, x_{max}, y_{min}, y_{max} του edge
- το slope του edge

Αφού αποθηκευτούν αυτά τα dictionaries στο `edges` array, υπολογίζεται το $y_{minTotal}$ και $y_{maxTotal}$ του τριγώνου. Αρχικοποιούνται επίσης ένα άδειο array για τα active edges και ένα για τα active points. Έπειτα ξεκινάει η βασική for loop της συνάρτησης, που υλοποιεί όλα τα scanline, με εύρος από $y_{minTotal}$ έως $y_{maxTotal} + 1$.

Σε κάθε επανάληψη ανανεώνεται ο πίνακας των active edges, προσθέτοντες όσες πλευρές έχουν $y_{min} = y$ και αφαιρόντας όσες πλευρές ήταν ήδη στον πίνακα και έχουν $y_{max} = y$. Υπάρχει επίσης ειδική μέριμνα για τις οριζόντιες πλευρές οι οποίες δεν αποδίδονται στα active edges του τριγώνου.

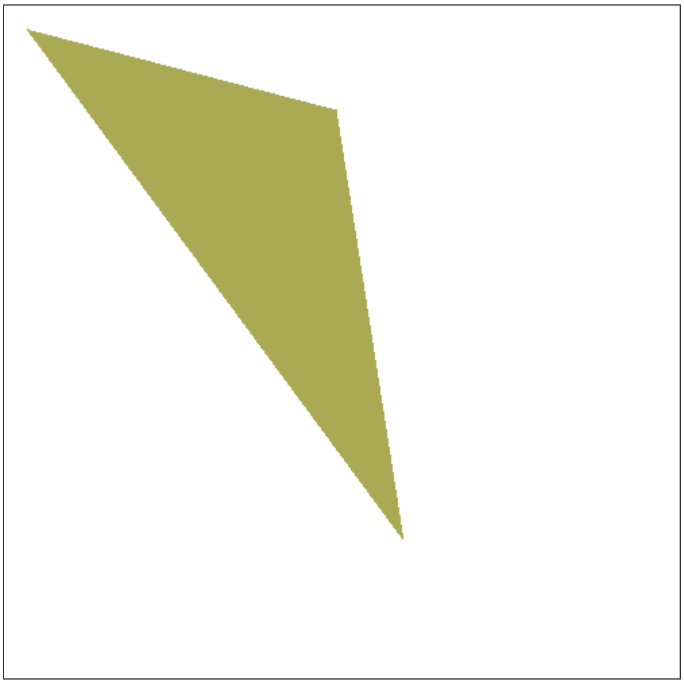

Figure 1: Απεικόνιση της `f_shading` σε λευκό καμβά

Έχοντας πλέον τα active edges, σε κάθε επανάληψη ο πίνακας των active points επαναφέρεται σε κενό για να υπολογιστούν εκ νέου. Γνωρίζοντας ότι τα σημεία της κάθε πλευράς υπακούουν στην εξίσωση

$$y = m_k x + b_k \tag{4}$$

ή ισοδύναμα

$$x = (y - b_k)/m_k \tag{5}$$

όπου m_k είναι το slope της πλευράς, οδηγούμαστε στο ότι αύξηση του y κατά μία μονάδα αντιστοιχεί σε μεταβολή του x κατά $1/m_k$. Έτσι τα νέα active points υπολογίζονται ανάλογα με το slope ως εξής:

- Για slope > 0 το νέο σημείο είναι το

$$(x_{min} + \frac{1}{\text{slope}} y - y_{min}, y)$$

- Για slope < 0 το νέο σημείο είναι το

$$(x_{max} + \frac{1}{\text{slope}} y - y_{min}, y)$$

- Για slope $= \infty$ το νέο σημείο είναι το

$$(x_{min}, y) = (x_{max}, y)$$

- Για slope $= 0$ δεν γίνεται κάποια ενέργεια διότι τα active points της οριζόντιας πλευράς ήδη έχουν υπολογιστεί από τις άλλες πλευρές που τα περιέχουν. *(Σημείωση: θεωρητικά κανένα από τα active edges δεν θα έχει slope $= 0$ λόγω του ελέγχου που γίνεται προηγουμένως)*

Τέλος, ο πίνακας των active points γίνεται sort με βάση τις τετμημένες τους και χρησιμοποιείται μια δεύτερη for loop με εύρος από την τετμημένη του πρώτου έως την τετμημένη του δεύτερου active point, όπου βάφονται όλα τα ενδιάμεσα σημεία με το χρώμα του τριγώνου.

Το ολοκληρωμένο αποτέλεσμα της συνάρτησης με ορίσματα `img` έναν λευκό καμβά 512×512, `vertices` τα $[20, 20], [250, 80], [300, 400]$ και `vcolors` τα $[1, 0, 0], [0, 1, 1], [1, 1, 0]$ φαίνεται στην εικόνα 1.

3 Συνάρτηση Gouraud Shading

Η επόμενη συνάρτηση που υλοποιήθηκε είναι η `g_shading(img, vertices, vcolors)`. Βασίζεται στην ίδια λογική που ακολουθεί η `f_shading`. Πλέον τα dictionaries των edges περιέχουν επιπλέον:

- `top_color` το χρώμα του vertex με το μεγαλύτερο y
- `bottom_color` το χρώμα του vertex με το μικρότερο y

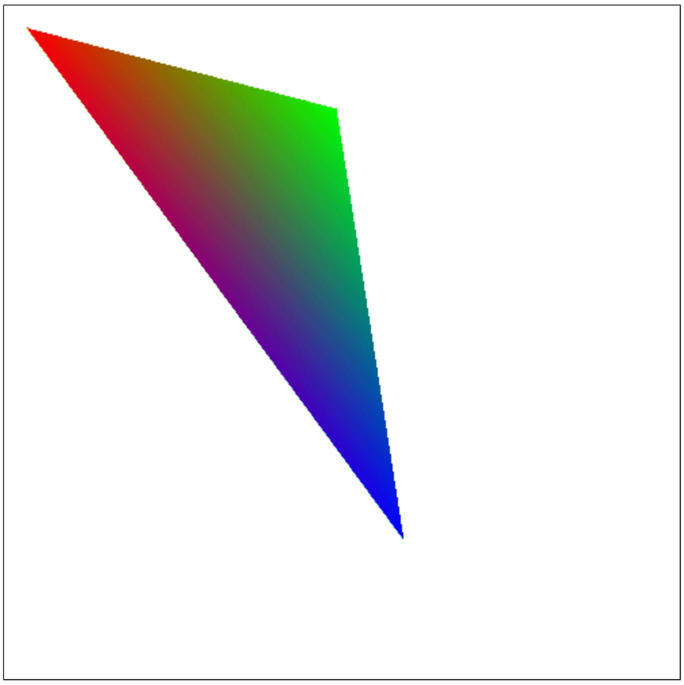

Figure 2: Απεικόνιση της `g_shading` σε λευκό καμβά

Αφού υπολογιστούν τα active edges και active points με τον ίδιο τρόπο όπως στην `f_shading` η διαφοροποίηση γίνεται στο πως θα αποδωθεί το χρώμα στο τρίγωνο. Σε κάθε επανάληψη των scanlines, ταυτόχρονα με τον υπολογισμό των active points, υπολογίζεται και μέσω της `vector_interp()` το χρώμα που τους αντιστοιχεί και αποθηκεύονται μαζί σε έναν πίνακα με όνομα `active_points_colors` που για δύο active points (x_1, y_1) και (x_2, y_2) είναι της μορφής:

$$\left[\left[\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} R_1 \\ G_1 \\ B_1 \end{bmatrix}\right], \left[\begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \begin{bmatrix} R_2 \\ G_2 \\ B_2 \end{bmatrix}\right]\right] \tag{6}$$

Αφού ο πίνακας αυτός ταξινομηθεί με βάση τις τιμές x_1 και x_2, στην επόμενη for loop που τρέχει για κάθε scanline χρησιμοποιείται ξανά η `vector_interp` ανάμεσα στα x_1 και x_2, για να υπολογιστεί το χρώμα σε κάθε σημείο του scanline και να βαφτεί με αυτό ο πίνακας `img` σε εκείνες τις συντεταγμένες.

Το ολοκληρωμένο αποτέλεσμα της συνάρτησης με ορίσματα `img` έναν λευκό καμβά 512×512, `vertices` τα $[20, 20], [250, 80], [300, 400]$ και `vcolors` τα $[1, 0, 0], [0, 1, 0], [0, 0, 1]$ φαίνεται στην εικόνα 2.

4 Συνάρτηση Render Image

Η συνάρτηση `render_img(faces, vertices, vcolors, depth, shading)` αξιοποιεί όλες τις προηγούμενες ώστε να παράξει την τελική εικόνα. Αρχικά αρχικοποιεί ένα `img` array διαστάσεων $512 \times 512 \times 3$ καθώς και ένα άδειο array `triangles`. Στη συνέχεια σε ένα loop για κάθε `face` του πίνακα `faces` της εισόδου υπολογίζεται το depth του ως το κέντρο βάρους του depth των ακμών του και κατασκευάζεται ένα dictionary `triangle` που περιέχει:

- `vertices` array με τις κορυφές του τριγώνου
- `color` array με τα αντίστοιχα χρώματα των κορυφών
- `depth` η τιμή του depth του τριγώνου

Τέλος σε κάθε επανάληψη το `triangle` γίνεται append σε έναν πίνακα `triangles`. Έπειτα ο πίνακας αυτός γίνεται reverse sort με βάση το βάθος έτσι ώστε τα μακρινά τρίγωνα να έρθουν πρώτα.

Τελικά για κάθε τρίγωνο του πίνακα `triangles`, καλείται η συνάρτηση `f_shading` αν το `shading = f` αλλιώς η `g_shading` αν το `shading = g` με όρισμα κάθε φορα την αρχική εικόνα img. Αφού κληθεί η συνάρτηση για όλα τα τρίγωνα, επιστρέφεται η τελική φωτογραφία.

5 Demos

Και τα δύο demos υλοποιούνται με την ίδια λογική. Αρχικά φορτώνονται τα δεδομένα από το αρχείο `hw1.py` με τη μορφή dictionary. Στη συνέχεια καλείται η συνάρτηση `render_img()`, στο `demo_f.py` με `shading = f` ενώ στο `demo_g.py` με `shading = g`. Η εικόνα που επιστρέφεται αποθηκεύεται στη μεταβλητή `img`.

Στη συνέχεια η `img` πολλαπλασιάζεται με 255: `img *= 255` έτσι ώστε οι RGB τιμές που είχαν εύρος $[0, 1]$ να αποκτήσουν εύρος $[0, 255]$. Έπειτα ο πίνακας img μετατρέπεται σε πίνακα ακεραίων (`numpy.uint8`) και αλλάζει το colorspace της φωτογραφίας από RGB σε BGR (που χρησιμοποιεί by default η OpenCV). Τέλος η φωτογραφία αποθηκέυεται σαν `jpg` αρχείο και εμφανίζεται χρησιμοποιώντας την `imshow()` της OpenCV. Τα τελικά αποτελέσματα φαίνονται στις εικόνες 3 και 4.

Figure 3: Η τελική φωτογραφία που παράγεται από το `demo_f.py`

Figure 4: Η τελική φωτογραφία που παράγεται από το `demo_g.py`