Εργασία 2: Μετασχηματισμοί και Προβολές

Πλευρίδη Βασιλική Βαρβάρα (ΑΕΜ:10454)

Μάϊος 2024

1 Εισαγωγή

Η παρούσα εργασία αναφέρεται σε μια σειρά από λειτουργίες που σχετίζονται με τη γεωμετρική αναπαράσταση και την απεικόνιση αντικειμένων σε έναν εικονικό χώρο. Αυτό περιλαμβάνει την προβολή 3D σημείων σε ένα επίπεδο εικόνας με βάση ένα μοντέλο προοπτικής προβολής, την αναγωγή των 2D σημείων σε pixel εικόνας, και την απεικόνιση τριγώνων στην εικόνα, λαμβάνοντας υπόψη το βάθος απόκρυψης. Επιπλέον, η εργασία περιλαμβάνει τις απαραίτητες μεθόδους για τη μετατροπή, την περιστροφή και τη μετάφραση των σημείων στον τρισδιάστατο χώρο, καθώς και την διαχείριση της απόκρυψης σε περιπτώσεις όπου τα τρίγωνα υπερβαίνουν τα όρια της εικόνας. Οι συναρτήσεις υλοποίησης όλων των παραπάνω, εμπεριέχονται στο αρχείο functions.py, ενώ στο αρχείο demo.py, γίνεται η παρουσίαση του ζητούμενου αντικειμένου, του οποίου τα δεδομένα βρίσκονται στο δοσμένο αρχείο hw2.py. Το αρχείο g_shading.py , από την πρώτη εργασία, συμπεριλαμβάνεται κι αυτό στο παραδοτεό, καθώς χρησιμοποιείται για τον χρωματισμό των επιμέρους τριγώνων.

2 Συναρτήσεις και σχολιασμός υλοποίησης

Παρακάτω σχολιάζονται αναλυτικά οι ζητούμενες συναρτήσεις και κλάσεις καθώς και πως αυτές υλοποιήθηκαν σε γλώσσα προγραματισμού python.

2.1 Κλάση Μετασχηματισμών Transform

Τα αντικείμενα αυτής της κλάσης είναι ουσιαστικά affine μετασχηματισμοί. Πέρα από τον constructor στον οποίο αρχικοποείται ο πίνακας μετασχηματισμού, υλοποιούνται οι εξής 3 μέθοδοι:

- **rotate (self, theta: float, u: np.ndarray)**: Υπολογίζεται ο πίνακας περιστροφής κατα γωνία theta περί άξονα με κατεύθυνση που δίνεται από το μοναδιαίο διάνυσμα u. Ο υπολογισμός του γίνεται με την βοήθεια του τύπου του Rodrigues, στον οποίο ο πίνακας R υπολογίζεται από την παρακάτω μαθηματική εξίσωση:

$$R = \begin{bmatrix} (1-\cos a)u_x^2 + \cos a & (1-\cos a)u_x u_y - (\sin a)u_z & (1-\cos a)u_x u_z + (\sin a)u_y \\ (1-\cos a)u_y u_x + (\sin a)u_z & (1-\cos a)u_y^2 + \cos a & (1-\cos a)u_y u_z - (\sin a)u_x \\ (1-\cos a)u_z u_x - (\sin a)u_y & (1-\cos a)u_z u_y + (\sin a)u_x & (1-\cos a)u_z^2 + \cos a \end{bmatrix}$$

  Ο επιστραφόμενος πίνακας είναι σε ομογενείς συντεταγμένες.

- **translate (self, t: np.ndarray)**:Υπολογίζεται ο πίνακας μετατόπισης Th σε ομογενείς συντεταγμένες σύμφωνα με τον παρακάτω τύπο:

$$Th = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ όπου } t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

- **transform_pts (self, pts:np.ndarray)**: Σε αυτήν τη μέθοδο, τα σημεία του πίνακα pts $\in \mathbb{R}^{3\times N}$, αφού γίνει η αναπαράστασή τους σε ομογενείς συντεταγμένες, μετασχηματίζονται με την βοήθεια του πίνακα μετασχηματισμού mat και τέλος επιστρέφονται σε καρτεσιανές συντεταγμένες.

2.2 Συνάρτηση αλλαγής συστήματος συντεταγμένων world2view

Η συνάρτηση **world2view(pts: np.ndarray, R: np.ndarray, c0: np.ndarray)** επιστρέφει τα σημεία pts μετασχηματισμένα ως προς το σύστημα συντεταγμένων της κάμερας. Για να το πετύχει αυτό, χρησιμοποιεί την εξής μαθηματική σχέση:

$$homogenous\_coordinates\_c = \left[\begin{array}{c|c} R^T & -R^T c0 \\ \hline 0_{1\times 3} & 1 \end{array}\right] pts\_homogenous$$

όπου pts_homogenous είναι τα σημεία εισόδου σε ομογενείς συντεταγμένες και homogenous_coordinates_c τα τελικά σημεία που αφού αναπαρασταθούν ξανά σε καρτεσιανές συντεταγμένες θα αποτελέσουν την έξοδο της συνάρτησης.

2.3 Συνάρτηση προσανατολισμού κάμερας lookat

Η συνάρτηση **lookat(eye: np.ndarray, up: np.ndarray, target: np.ndarray)** υπολογίζει τον πίνακα περιστροφής R και το διάνυσμα μετατόπισης d με βάση των εξής μαθηματικών εκφράσεων:

$$R = \begin{bmatrix} x_c & y_c & z_c \end{bmatrix} \text{ όπου } x_c = y_c \times z_c \quad y_c = \frac{t}{|t|} \quad z_c = \frac{ck}{|ck|} \quad \text{και } d = eye$$

Τα διανύσματα t και ck υπολογίζονται από τις σχέσεις:

$$t = up - \langle up, z_c \rangle z_c \quad ck = target - eye$$

2.4 Συνάρτηση προοπτικής προβολής με pinhole κάμερα perspective_project

Η συνάρτηση **perspective_project(pts: np.ndarray, focal: float, R: np.ndarray, t: np.ndarray)** ουσιαστικά υπολογίζει και επιστρέφει τις 2D συντεταγμένες των σημείων εισόδου στο πέτασμα της κάμερας. Για να το πετύχει αυτό χρησιμοποείται ο εξής πολλαπλασιασμός πινάκων:

$$\begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_p w \\ y_p w \\ z_p \\ z_p w \end{bmatrix} \text{ όπου } w = focal$$

Ο πίνακας εισόδου pts αφού μετασχηματιστεί κατάλληλα στο σύστημα συντεταγμένων της κάμερας με την βοήθεια της υλοποιημένης συνάρτησης world2view , μέσω ενός for loop, οδηγείται στην τελική του μορφή με την χρήση του παραπάνω τύπου. Στο τελικό tuple βέβαια, επιστρέφονται μόνο οι συντεταγμένες που μας ενδιαφέρουν (x_q, y_q) αλλά και το βάθος των σημείων (z_p).

2.5 Συνάρτηση απεικόνισης rasterize

Η συνάρτηση **rasterize(pts_2d: np.ndarray, plan_w: int, plane_h: int, res_w: int, res_h: int)** έχει ως στόχο την απεικόνιση των συντεταγμένων των σημείων από το σύστημα συντεταγμένων του πετάσματος της κάμερας,με πέτασμα plane_h × plane_w, σε ακέραιες θέσεις (pixels) της εικόνας διάστασης res_h × res_w. Σημαντικό είναι να σχολιαστεί ότι το αρχικό κέντρο (0,0) είναι στο κέντρο του ορθογώνιου πλαισίου της κάμερας και με την πρόσθεση +plane_w/2,+plane_w/2 αντίστοιχα μεταφέρεται στο τέρμα κάτω και αριστερά σημείο του πλαισίου. Για την αναγωγή σε pixels, αρκεί να γίνει χρησιμοποιηθεί η εξής μαθηματική σχέση για κάθε συντεταγμένη: $pixel\_coordinate = int(coordinate \times res/plane)$

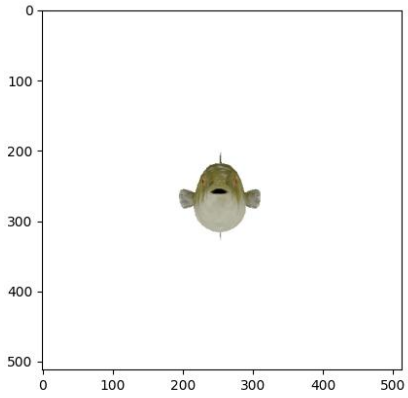
2.6 Συνάρτηση φωτογράφισης render_object

Η συνάρτηση **render_object(v_pos, v_clr, t_pos_idx, plane_h, plane_w**,επιστρέφει ουσιαστικά την φωτογραφία του αντικειμένου που περιγράφεται από τα ορίσματα εισόδου. Σε αυτήν λοιπόν, χρησιμοποιούνται όλες οι συναρτήσεις που υλοποιήθηκαν παραπάνω ώστε να παραχθούν εν τέλει τα τελικά pixels. Όπως και στην προηγούμενη εργασία , αρχικοποιείται η εικόνα και στην συνέχεια,γίνεται ο υπολογισμός του βάθους για

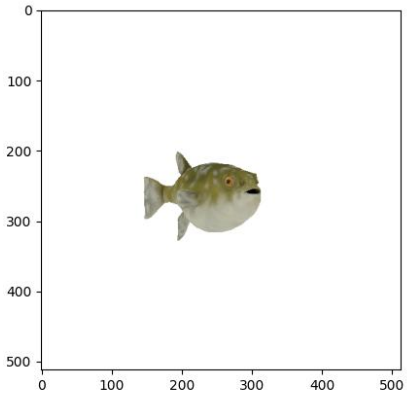
κάθε τρίγωνο με βάση το κέντρου βάρους των κορυφών του κάθε τριγώνου και στην συνέχεια η κατά φθίνουσα ταξινόμηση τους με κριτήριο το παραπάνω. Το τελικό στάδιο ειναι η σχεδίαση και ο χρωματισμός των τριγώνων με την βοήθεια της συνάρτησης g_shading από την προηγούμενη εργασία. Να σημειωθεί ότι ο συνθηκολογικός έλεγχος που υπάρχει στην υλοποίηση της python, γίνεται για να καλυφθεί η περιπτώση που το αντικείμενο δεν είναι ολόκληρο μέσα στο πλαίσιο της κάμερας. Το αντικείμενο που χρησιμοποιείται στην συγκεκριμένη εργασία, μετά και πριν τους προκαθορισμένους μετασχηματισμούς, δεν απαιτεί την υλοποίηση του **clipping**, παρόλα αυτά, ενδεικτικά υλοποιήθηκε και θα παρουσιαστεί και αναλυτικά παρακάτω, η περίπτωση που μέρος του αντικειμένου βρίσκεται πιο πάνω ή αριστερά από το πλαίσιο της κάμερας.

3 Scripts και αποτελέσματα

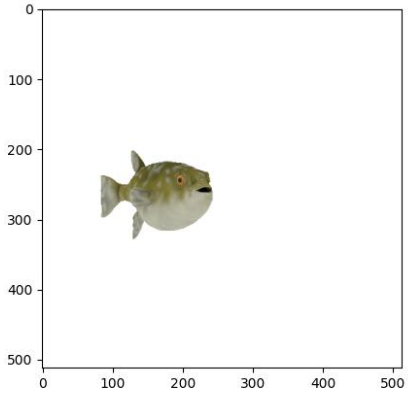
Για την επαλήθευση της σωστής λειτουργίας των κωδίκων, δημιουργήθηκε το αρχείο demo.py, στο οποίο αφού γίνει η φόρτωση των απαιτούμενων δεδομένων από το αρχείο hw2.npy, δημιουργεί αντικείμενα της κλάσης Transform για να γίνουν οι κατάλληλοι μετασχηματισμοί και με το κάλεσμα της συνάρτησης render_object να επιστραφεί και να αποθηκευτούν οι ζητούμενες εικόνες. Τα τελικά αποτελέσματα παρουσιάζονται παρακάτω:



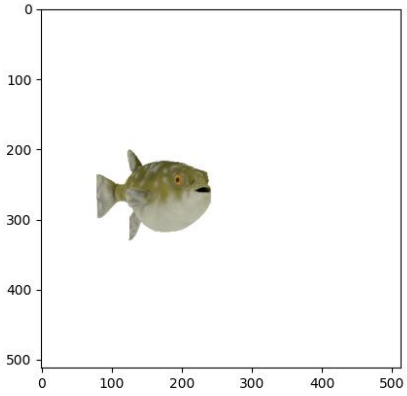
Αρχική θέση



Μετά την περιστροφή



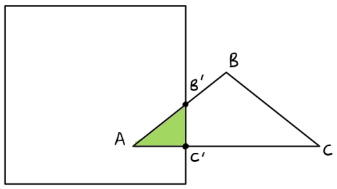
Μετά την πρώτη μετατόπιση



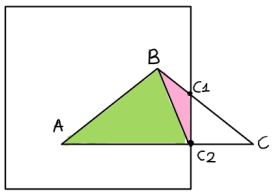
Μετά την δεύτερη μετατόπιση

4 Αποκοπή - Clipping

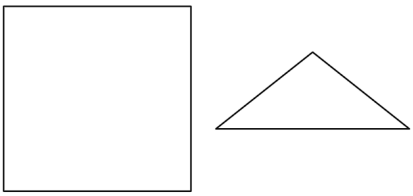
Σε αυτήν την ενότητα θα σχολιαστεί η διαχείριση της απόκρυψης σε περιπτώσεις όπου τα τρίγωνα υπερβαίνουν τα όρια της εικόνας. Υπάρχουν πολλές υποπεριπτώσεις, ανάλογα με το αν το τρίγωνο αποκόπτεται από πλευρά ή γωνία του πλαισίου της κάμερας αλλά και με το πλήθος των γωνιών που αποκόπτονται. Στην παρούσα εργασία, μελετήθηκε η αποκοπή τριγώνου από το μεγαλύτερο όριο του πλαισίου και στις δύο συντεταγμένες και για όλα τα πλήθη των γωνιών που βρίσκονται εκτός πλαισίου. Παρακάτω παρουσιάζονται οι 3 περιπτώσεις όπου το τρίγωνο θα πρέπει να μπει σε διαδικασία clipping:



1.Δυο κορυφές εκτός πλαισίου



2.Μία κορυφή εκτός πλαισίου

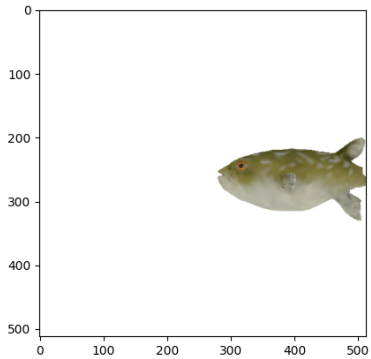


3.Όλες οι κορυφές εκτός πλαισίου

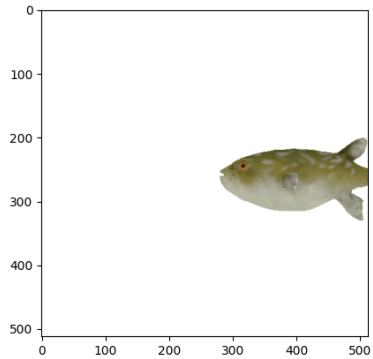
Ξεκινώντας από την περίπτωση 1, όπου οι γωνιές B και C βρίσκονται εκτός τριγώνου, αρκεί να βρεθούν τα σημεία B' και C' αντίστοιχα με την βοήθεια των εξισώσεων των ευθειών που τα διέπουν (AB και AC αντίστοιχα). Η δεύτερη περίπτωση είναι λίγο πιο πολύπλοκη. Παρατηρείται ότι το κομμάτι του τριγώνου εντός του πλαισίου είναι τετράπλευρο και έτσι διασπάται σε 2 επιμέρους τρίγωνα (ABC_2, BC_1C_2). Τέλος, η 3η περίπτωση είναι η πιο απλή καθώς το τρίγωνο αποκόπτεται ολόκληρο και δεν χρειάζεται κάποια περαιτέρω επεξεργασία.

Η υλοποίηση της συνάρτησης **clipping(lim,vertices_triangle,option,vcolors_triangle,image)** γίνεται με την εξής λογική: Για τις περιπτώσεις 1 και 2, αρχικά, αναζητώνται οι κορύφες που βρίσκονται εσωτερικά του πλαισίου και μεταφέρονται σε έναν νέο πίνακα new_vertices_triangle, ο οποίος θα περιέχει εν τέλει τις κορυφές του νέου τριγώνου. Στην συνέχεια, με την βοήθεια της συνάρτησης slope και calculate_coordinates υπολογίζονται οι συντεταγμένες των νέων κορύφων και τοποθετούνται και αυτές στον ίδιο πίνακα. Σημαντικό είναι να σχολιαστεί, ότι ταυτόχρονα δημιουργείται και αντίστοιχος πίνακας για τα χρώματα των κορυφών. Τα νέα χρώματα προκύπτουν με την βοήθεια της συνάρτησης vector_interp.Τέλος, καλείται η g_shading για την απεικόνιση των τριγώνων στην εικόνα.

Στο αρχείο demo.py, έχει γραφεί ένα παράδειγμα με ακολουθία μετασχηματισμών που απαιτούν την χρήση της παραπάνω συνάρτησης. Παρακάτω παρουσιάζεται το αποτέλεσμα αυτού καθώς και τι θα γίνοταν χωρίς την χρήση αυτής:



Χωρίς την χρήση clipping



Με την χρήση clipping