Γραφική με Υπολογιστές

Εργασία #2

Μετασχηματισμοί και Προβολές

ΕΠΙΜΕΛΕΙΑ: ΠΙΤΤΗΣ ΓΕΩΡΓΙΟΣ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1. Στόχος εργασίας

Η εργασία ασχολείται με τον μετασχηματισμό και την απεικόνιση 3D σκηνικών. Αυτό περιλαμβάνει την εφαρμογή affine μετασχηματισμών, όπως περιστροφή και μετατόπιση, για την προσαρμογή των 3D σκηνικών στη γεωμετρία της κάμερας. Επίσης, περιλαμβάνει τη μετατροπή των 3D σημείων από το Παγκόσμιο Σύστημα Συντεταγμένων (WCS) στο σύστημα συντεταγμένων της κάμερας. Ακόμη, παράγει τις προοπτικές προβολές των 3D σημείων λαμβάνοντας υπόψη το βάθος τους(δηλαδή την τρίτη συντεταγμένη τους) και τις επιστρέφει στο πέτασμα της κάμερας. Με λίγα λόγια, ο στόχος της εργασίας είναι η προβολή των 3D σκηνικών στο 2D πέτασμα της κάμερας.

2. Περιγραφή λειτουργίας και τρόπου κλήσης συναρτήσεων

Στο αρχείο functions.py έχουν υλοποιηθεί όλες οι ζητούμενες συναρτήσεις.

- <u>Κλάση Transform :</u>

  Οι συναρτήσεις της κλάσης Transform υπολογίζουν τους affine μετασχηματισμούς περιστροφής και μετατόπισης. Πιο συγκεκριμένα, η κλάση περιέχει ως attribute τον 4x4 πίνακα mat , ο οποίος αντιπροσωπεύει κάποιον affine μετασχηματισμό.

  - **def __init__(self):** Είναι ο constructor της κλάσης. Αρχικοποιεί τον πίνακα mat σε μοναδιαίο πίνακα I_4.

  - **rotate(self, theta, u):** Η συνάρτηση rotate υπολογίζει τον 3x3 πίνακα περιστροφής που αντιστοιχεί σε μια δεξιόστροφη περιστροφή κατά γωνία θ γύρω από άξονα που ορίζεται από το μοναδιαίο διάνυσμα u και στη συνέχεια ενημερώνει τον πίνακα mat. Τότε, ο πίνακας mat αντιπροσωπεύει τον affine μετασχηματισμό περιστροφής. Ένα αντικείμενο της κλάσης Transform καλεί τη συνάρτηση rotate με ορίσματα εισόδου την γωνία περιστροφής theta και το μοναδιαίο διάνυσμα u. Η συνάρτηση υπολογίζει τα στοιχεία του μη ομογενή πίνακα περιστροφής R κάνοντας χρήση της εξίσωσης 5.45:

    $$\boldsymbol{R} = \begin{bmatrix} (1-cosa){u_x}^2 + (cosa) & (1-cosa)u_x u_y - (sina)u_z & (1-cosa)u_x u_z + (sina)u_y \\ (1-cosa)u_y u_x + (sina)u_z & (1-cosa){u_y}^2 + (cosa) & (1-cosa)u_y u_z - (sina)u_x \\ (1-cosa)u_z u_x - (sina)u_y & (1-cosa)u_z u_y + (sina)u_x & (1-cosa){u_z}^2 + (cosa) \end{bmatrix}$$

    Τη πρώτη γραμμή του πίνακα R την αναθέτω στο διάνυσμα r1, τη δεύτερη στο διάνυσμα r2 και τη τρίτη στο διάνυσμα r3. Στη συνέχεια, με βάση την εξίσωση 5.49 ενημερώνω τον πίνακα mat με τα διανύσματα r1,r2,r3 (δηλαδή με τον πίνακα R).

- **translate(self, t):** Η συνάρτηση translate ενημερώνει τον πίνακα mat με το διάνυσμα μετατόπισης t. Τότε, ο πίνακας mat αντιπροσωπεύει τον affine μετασχηματισμό μετατόπισης. Ένα αντικείμενο της κλάσης Transform καλεί τη συνάρτηση translate με όρισμα εισόδου το διάνυσμα μετατόπισης t σε μη ομογενείς συντεταγμένες. Στη συνέχεια, με βάση την εξίσωση 5.37 ενημερώνω κατάλληλα τον πίνακα mat με το διάνυσμα μετατόπισης t.

- **transform_pts(self, pts) -> np.ndarray:** Η συνάρτηση transform_pts εφαρμόζει τον πίνακα μετασχηματισμού mat στα 3D σημεία του πίνακα pts που δέχεται ως είσοδο. Επιστρέφει τα μετασχηματισμένα σημεία. Ένα αντικείμενο της κλάσης Transform καλεί τη συνάρτηση transform_pts με όρισμα εισόδου τον 3xN πίνακα pts. Τα σημεία εισόδου έχουν μη ομογενείς συντεταγμένες. Για τον λόγο αυτό, πρέπει να προσθέσω το 1 σε κάθε στήλη του πίνακα pts ώστε να μετατρέψω τις συντεταγμένες των σημείων εισόδου σε ομογενείς. Δημιουργώ την λίστα modified_cols και αποθηκεύω σε αυτή τις ομογενείς συντεταγμένες των σημείων. Στη συνέχεια, αναθέτω στον πίνακα pts τις συντεταγμένες αυτές, με αποτέλεσμα να αλλάξουν οι διαστάσεις του σε 4xN. Επομένως, τώρα μπορώ να εφαρμόσω στον πίνακα pts τον πίνακα μετασχηματισμού mat και συνακόλουθα να παράξω τα μετασχηματισμένα σημεία. Τα μετασχηματισμένα σημεία αποθηκεύονται στον 4xN πίνακα pts_transform. Δημιουργώ τον πίνακα new_pts_transform και αποθηκεύω σε αυτόν τις μη ομογενείς συντεταγμένες των μετασχηματισμένων σημείων. Ενημερώνω τον πίνακα pts_transform με τις συντεταγμένες αυτές και στην συνέχεια τον επιστρέφω.

- **world2view(pts, R, c0) -> np.ndarray:** Η συνάρτηση μετασχηματίζει τα σημεία εισόδου του πίνακα pts στο σύστημα συντεταγμένων της κάμερας. Επιστρέφει τον Nx3 πίνακα pts_transform με τα σημεία εισόδου μετασχηματισμένα ως προς το σύστημα συντεταγμένων της κάμερας. Την καλώ με ορίσματα εισόδου τον 3xN πίνακα pts, τον 3x3 πίνακα περιστροφής του νέου συστήματος ως προς το αρχικό (R) και το 1x3 διάνυσμα αναφοράς του νέου συστήματος ως προς το αρχικό (c0). Μετατοπίζω τα σημεία εισόδου αφαιρώντας από αυτά το διάνυσμα αναφοράς και τα αποθηκεύω στον Nx3 πίνακα pts_new. Για να είναι η αφαίρεση εφικτή θα πρέπει τα σημεία εισόδου να βρίσκονται στις γραμμές και όχι στις στήλες του πίνακα pts. Για τον λόγο αυτό, στην αφαίρεση χρησιμοποιώ τον pts.T. Στη συνέχεια, εφαρμόζω τον πίνακα περιστροφής R στον pts_new.T και παράγω τον πίνακα pts_transform.

- **lookat(eye, up, target) -> Tuple[np.ndarray, np.ndarray]:** Η συνάρτηση παράγει και επιστρέφει τον 3x3 πίνακα περιστροφής R και το 1x3 διάνυσμα μετατόπισης t που χρειάζονται για το μετασχηματισμό των σημείων εισόδου από το WCS στο σύστημα συντεταγμένων της κάμερας. Την καλώ με ορίσματα εισόδου το 3x1 διάνυσμα eye που ταυτίζεται με το κέντρο της κάμερας , το 3x1 μοναδιαίο διάνυσμα up της κάμερας και το 3x1 διάνυσμα target που αντιπροσωπεύει το σημείο στο οποίο στοχεύει η κάμερα. Αρχικά, μετατρέπω τα διανύσματα εισόδου σε 1x3 διανύσματα. Στη συνέχεια, υπολογίζω το διάνυσμα ck , το οποίο ταυτίζεται με τον άξονα του φακού της κάμερας. Το σημείο c ταυτίζεται με το διάνυσμα eye και το σημείο k ταυτίζεται με το διάνυσμα target. Έπειτα, υπολογίζω τις συντεταγμένες των μοναδιαίων διανυσμάτων $\widehat{x_c}$,$\widehat{y_c}$, $\widehat{z_c}$ της κάμερας ως προς WCS. Χρησιμοποιώ την εξίσωση 6.6 $\left(\widehat{z_c} = \frac{\overrightarrow{ck}}{|\overrightarrow{ck}|}\right)$ και υπολογίζω το $\widehat{z_c}$. Στη συνέχεια, υπολογίζω το εσωτερικό γινόμενο της εξίσωσης 6.7 ($< \overrightarrow{up}, \widehat{z_c} >$) και το αποθηκεύω στην μεταβλητή dot_product. Εφαρμόζω την δεύτερη σχέση της εξίσωσης 6.7 ($\vec{v} = \overrightarrow{up} - < \overrightarrow{up}, \widehat{z_c} > \cdot \widehat{z_c}$) και υπολογίζω το διάνυσμα v το οποίο είναι παράλληλο προς το $\widehat{y_c}$. Βρίσκω το $\widehat{y_c}$ κάνοντας χρήση της πρώτης σχέσης στην εξίσωση 6.7 $\left(\widehat{y_c} = \frac{\vec{v}}{|\vec{v}|}\right)$. Ακόμη, χρησιμοποιώ την εξίσωση 6.8 και υπολογίζω το $\widehat{x_c}$ ($\widehat{x_c} = \widehat{y_c} \times \widehat{z_c}$). Βρίσκω τον πίνακα περιστροφής R από την εξίσωση 6.11 ($\boldsymbol{R} = [\widehat{x_c}\ \widehat{y_c}\ \widehat{z_c}]$) και το διάνυσμα μετατόπισης από την εξίσωση 6.12 ($\vec{t} = \vec{c}$, δηλαδή $\vec{t} = \overrightarrow{eye}$).

- **perspective_project(pts, focal, R, t) -> Tuple[np.ndarray, np.ndarray]:** Η συνάρτηση perspective_project υπολογίζει τις προοπτικές προβολές(pts_2d) και το βάθος των 3D σημείων εισόδου(depths) και στη συνέχεια τα επιστρέφει. Ουσιαστικά, χρησιμοποιώ τις συντεταγμένες x και y των 3D σημείων εισόδου για να δημιουργήσω τις προοπτικές τους προβολές. Επιπλέον, η συντεταγμένη z των σημείων αντιπροσωπεύει το βάθος τους. Θέλω να απεικονίσω στο επίπεδο του πετάσματος της κάμερας τις προβολές των 3D σημείων. Προκειμένου λοιπόν να μην "χαθεί" η συντεταγμένη z , την μεταφράζω σε βάθος. Την καλώ με ορίσματα εισόδου τον 3xN πίνακα pts που περιέχει τα 3D σημεία εισόδου, τη μεταβλητή focal που είναι η απόσταση του πετάσματος της κάμερας από το κέντρο, τον 3x3 πίνακα περιστροφής R προς το σύστημα συντεταγμένων της κάμερας και το 1x3 διάνυσμα μετατόπισης t προς το σύστημα συντεταγμένων της κάμερας. Αρχικά, μετασχηματίζω τα σημεία εισόδου στις συντεταγμένες της κάμερας χρησιμοποιώντας την συνάρτηση world2view με ορίσματα τον πίνακα pts, τον πίνακα R και το διάνυσμα t. Τα μετασχηματισμένα σημεία τα αποθηκεύω στον Nx3 πίνακα pts_transform. Δημιουργώ την λίστα depths και αποθηκεύω σε αυτή τα βάθη των μετασχηματισμένων σημείων. Δημιουργώ τις λίστες xq , yq για να αποθηκεύσω τις προβολές των συντεταγμένων x και y αντίστοιχα. Υπολογίζω τις προοπτικές προβολές των μετασχηματισμένων σημείων σύμφωνα με τις σχέσεις : $x_q = \frac{w \cdot x_p}{z_p}$ και

$y_q = \frac{w \cdot y_p}{z_p}$ στην σελίδα 72 του gr-notes.pdf. Ειδικότερα, το z_p ταυτίζεται με το depths[i] , το x_p ταυτίζεται με το pts_transform[i][0], το y_p ταυτίζεται με pts_transform[i][1] και το w ταυτίζεται με το focal. Αξίζει να σημειωθεί ότι για να βρω εδώ τις προοπτικές προβολές εργάζομαι με μη ομογενείς συντεταγμένες και όχι με ομογενείς όπως στη σελίδα 72. Δημιουργώ τον 2xN πίνακα pts_2d και αποθηκεύω στην πρώτη γραμμή του τις προβολές όλων των x συντεταγμένων των σημείων και στη δεύτερη γραμμή του τις προβολές όλων των y συντεταγμένων των σημείων. Όμως, επιστρέφω τον Nx2 πίνακα pts_2d.T , ο οποίος σε κάθε γραμμή του περιέχει το ζευγάρι των προοπτικών προβολών ενός μετασχηματισμένου σημείου.

- **rasterize(pts_2d, plane_w, plane_h, res_w, res_h) -> np.ndarray:** Η συνάρτηση απεικονίζει τις συντεταγμένες 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. Την καλώ με ορίσματα εισόδου τον πίνακα pts_2d που περιέχει τις προοπτικές προβολές των 3D σημείων, τις μεταβλητές plane_w, plane_h (διαστάσεις πετάσματος κάμερας) και τις μεταβλητές res_w, res_h (διαστάσεις καμβά εικόνας). Πρώτα, υπολογίζω τις κλίμακες **scale_x** και **scale_y** με βάση τις διαστάσεις του πετάσματος της κάμερας και του καμβά της εικόνας. Έτσι, βρίσκω πόσα pixels αντιστοιχούν σε μία μονάδα μήκους της κάμερας. Στη συνέχεια, μετακινώ τα σημεία του πετάσματος της κάμερας κατά plane_w/2 στην οριζόντια διάσταση και κατά plane_h/2 στην κάθετη διάσταση προκειμένου να ευθυγραμμιστούν οι δύο επιφάνειες(δηλαδή θέλω να ευθυγραμμίσω το πέτασμα της κάμερας με τον καμβά της εικόνας). Το αποτέλεσμα στρογγυλοποιείται για να λάβουμε ακέραιες τιμές pixel.

- **render_object(v_pos, v_clr, t_pos_idx, plane_h, plane_w, res_h, res_w, focal, eye , up, target) -> np.ndarray:** Η συνάρτηση render_object φωτογραφίζει μία 3D σκηνή ενός αντικειμένου από μία κάμερα. Επιστρέφει τον πίνακα image_array μεγέθους res_h × res_w × 3 (τη φωτογραφία του αντικειμένου). Επίσης, υλοποιεί όλο το pipeline της απεικόνισης του αντικειμένου. Επιπλέον, χρωματίζει το αντικείμενο χρησιμοποιώντας τη μέθοδο Gouraud. Την καλώ με ορίσματα εισόδου τον 3xN πίνακα v_pos που αναπαριστά τις τρισδιάστατες συντεταγμένες των σημείων του αντικειμένου, τον Nx3 πίνακα v_clr που περιέχει τα χρώματα των κορυφών, τον Fx3 πίνακα t_pos_idx που περιέχει δείκτες που δείχνουν σε σημεία στο v_pos που αποτελούν τις κορυφές των τριγώνων, τις διαστάσεις του πετάσματος της κάμερας (plane_h, plane_w), τις διαστάσεις του καμβά της εικόνας(res_h, res_w) , τη μεταβλητή focal και τα διανύσματα eye, up, target της κάμερας. Καλεί διαδοχικά τις συναρτήσεις lookat(eye, up, target), perspective_project(v_pos, focal, R, t), rasterize(pts_2d, plane_w, plane_h, res_w, res_h) και render_img(t_pos_idx_list, pts_rast_list, v_clr_list, depths_list, "g").

3. Αρχείο demo.py

Στο αρχείο demo.py φορτώνω τα δεδομένα του αρχείου hw2.npy και χρησιμοποιώ την συνάρτηση render_object καθώς και συναρτήσεις της κλάσης Transform για να υπολογίσω τις ζητούμενες εικόνες.

Για κάθε βήμα (α' έως γ') δημιουργώ ένα νέο αντικείμενο τύπου Transform. Αυτό το κάνω διότι όλοι οι διαδοχικοί affine μετασχηματισμοί συσσωρεύονται στον πίνακα mat. Επομένως, αν χρησιμοποιούσα το ίδιο αντικείμενο της κλάσης Transform σε κάθε βήμα, τότε στα βήματα 2 και 3 αντί για απλή μετατόπιση t_1 και t_2 αντίστοιχα, θα είχαμε επίσης μια επιπλέον περιστροφή (με πίνακα περιστροφής R από το βήμα 1).

Εναλλακτικά, θα μπορούσαμε να χρησιμοποιήσουμε το ίδιο αντικείμενο της κλάσης Transform σε κάθε βήμα , με την προϋπόθεση ότι θα ξαναρχικοποιούσαμε τον πίνακα mat σε I_4 πριν από κάθε βήμα.

4. Τα ενδεικτικά αποτελέσματα που παράγονται από το demo

Εικόνα 1 (0.jpg) : Αρχική εικόνα

Εικόνα 2 (1.jpg) : Περιστροφή κατά γωνία theta = 0.5235987755982988 rad προς άξονα rot_axis = [0 1 0]

Εικόνα 3 (2.jpg) : Μετατόπιση κατά t_1 = [1 0 0] (μετατόπιση κατά x)

Εικόνα 4 (3.jpg) : Μετατόπιση κατά t_2 = [0 0 -1] (Η μετατόπιση αυτή επηρεάζει μόνο το βάθος του αντικειμένου. Έχω μεγέθυνση του αντικειμένου)

5. Παρατηρήσεις

- Στις συναρτήσεις f_shading και g_shading αφαίρεσα το κομμάτι κώδικα :

  ```
  vertices = np.array(vertices)
  vertices[0][0] = -vertices[0][0]
  vertices[1][0] = -vertices[1][0]
  vertices[2][0] = -vertices[2][0]
  ```

  Το συγκεκριμένο κομμάτι κώδικα το είχα χρησιμοποιήσει στην 1η εργασία για να πραγματοποιήσω μια τεχνητή περιστροφή της φωτογραφίας ώστε να φαίνεται “ίσιο” το αντικείμενο.

- Η render_img χρησιμοποιεί τις f_shading και g_shading. Η f_shading χρησιμοποιεί την line_drawing. Η g_shading χρησιμοποιεί την line_drawing και την vector_interp. Για τον λόγο αυτό στην συγκεκριμένη εργασία χρησιμοποιώ όλες τις συναρτήσεις της 1ης εργασίας.