ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

Γραφική με Υπολογιστές Αναφορά

Εργασία 2

Διακολουκάς Δημήτριος
ΑΕΜ 10642

Email: ddiakolou@ece.auth.gr

Περιεχόμενα

Κεφάλαιο 1

Περιγραφή Προβλήματος και Ζητούμενα

1.1 Εισαγωγή

Η δεύτερη εργασία της σειράς μαθημάτων Γραφική με Υπολογιστές εστιάζει στη δημιουργία ενός πλήρους pipeline προβολής 3D αντικειμένων μέσω μετασχηματισμών, προοπτικής προβολής και τελικής απόδοσης με χρήση υφής (texture mapping). Στόχος είναι η κατανόηση της λειτουργίας μίας εικονικής κάμερας, καθώς και η προσομοίωση της κίνησης σε δυναμικά περιβάλλοντα.

Το προς απόδοση αντικείμενο είναι μια 3D πυραμίδα, η οποία κινείται πάνω σε κυκλική τροχιά. Η κάμερα μπορεί είτε να είναι σταθερή είτε να ακολουθεί δυναμικά το όχημα (camera-follow mode). Το τελικό αποτέλεσμα είναι δύο ανιματιον διάρκειας 5 δευτερολέπτων το καθένα, που αποθηκεύονται σε μορφή ιδεο.

1.2 Δοσμένα Δεδομένα – hw2.npy και εικόνα

Το αρχείο `hw2.npy` περιέχει όλα τα απαραίτητα δεδομένα της σκηνής:

- **v_pos:** $3 \times N$ πίνακας με τις τρισδιάστατες συντεταγμένες των κορυφών του αντικειμένου.
- **v_uvs:** $N \times 2$ πίνακας με τις συντεταγμένες υφής (UV) για κάθε κορυφή.
- **t_pos_idx:** $F \times 3$ πίνακας που καθορίζει τα τρίγωνα μέσω δεικτών σε **v_pos**.
- **stone-72_diffuse.jpg:** Η εικόνα υφής που προβάλλεται στην επιφάνεια του αντικειμένου μέσω UV mappingπου αναπαριστά έναν βράχο.
- **k_road_center:** Τρισδιάστατο διάνυσμα που δηλώνει το κέντρο της κυκλικής τροχιάς του αυτοκινήτου.
- **k_road_radius:** Ακτίνα της κυκλικής τροχιάς (τύπου `float`).
- **car_velocitdy:** Ταχύτητα του οχήματος σε μονάδες χώρου ανά δευτερόλεπτο.
- **k_cam_car_rel_pos:** Τρισδιάστατο διάνυσμα που καθορίζει τη σχετική θέση της κάμερας ως προς το όχημα.
- **k_cam_up:** Κανονικοποιημένο διάνυσμα 3×1 που δηλώνει τον κάθετο προσανατολισμό της κάμερας (up vector).

- **k_cam_target:** Τρισδιάστατο σημείο-στόχος προς το οποίο κοιτάει η κάμερα (μόνο για demo 2).
- **k_sensor_height, k_sensor_width:** Διαστάσεις του αισθητήρα της κάμερας σε μονάδες χώρου.
- **k_f:** Εστιακή απόσταση (focal length) του μοντέλου κάμερας (τύπου `int`).
- **k_duration:** Συνολική διάρκεια προσομοίωσης (σε δευτερόλεπτα).
- **k_fps:** Ρυθμός (frames per second) της προσομοίωσης.

1.3 Ζητούμενα

Η εργασία ζητά την υλοποίηση των εξής συναρτήσεων:

1. `translate(t_vec)`: Δημιουργεί affine πίνακα μετατόπισης 4×4.
2. `rotate(axis, angle, center)`: Δημιουργεί affine πίνακα περιστροφής 4×4 γύρω από τυχαίο άξονα.
3. `compose(mat1, mat2)`: Σύνθεση δύο affine πινάκων μετασχηματισμού.
4. `world2view(pts, R, c0)`: Μετασχηματίζει σημεία από τον παγκόσμιο χώρο (WCS) στο σύστημα συντεταγμένων της κάμερας.
5. `lookat(eye, up, target)`: Υπολογίζει τον πίνακα περιστροφής και μετατόπισης της κάμερας ώστε να κοιτά το στόχο.
6. `perspective_project(pts, focal, R, t)`: Προοπτική προβολή σημείων από 3D σε 2D επίπεδο, με χρήση pinhole μοντέλου κάμερας.
7. `rasterize(pts_2d, plane_w, plane_h, res_w, res_h)`: Χαρτογραφεί τις συντεταγμένες από το επίπεδο της κάμερας σε pixel συντεταγμένες εικόνας.
8. `render_object(...)`: Συνδυάζει όλες τις παραπάνω λειτουργίες και επιστρέφει εικόνα της σκηνής με χρήση texture shading.

1.4 Προσομοιώσεις και Απαιτήσεις

Θα πρέπει να παραχθούν δύο προσομοιώσεις:

- **Demo 1:** Κάμερα στατική ως προς το όχημα, προσανατολισμένη πάντα προς την κατεύθυνση κίνησης. Το διάνυσμα $\hat{z}_c$ είναι πάντα παράλληλο με το διάνυσμα ταχύτητας του αυτοκινήτου.
- **Demo 2:** Κάμερα που περιστρέφεται για να κοιτά πάντα ένα σταθερό σημείο (`k_cam_target`).

Κάθε προσομοίωση πρέπει να διαρκεί 5 δευτερόλεπτα, με 25 frames ανά δευτερόλεπτο, και να αποθηκεύεται ως αρχείο βίντεο τύπου `.mp4`.

1.5 Συναρτήσεις Υλοποίησης Χρωματισμού και Υφής

Στην παρούσα εργασία, γίνεται χρήση και επαναχρησιμοποίηση βοηθητικών συναρτήσεων από την προηγούμενη εργασία για την υλοποίηση χαρτογράφησης υφής (texture mapping) στο νέο pipeline της `render_object`. Οι συναρτήσεις αυτές είναι οι `vector_interp`, `t_shading` και `render_img`.

Συνάρτηση γραμμικής παρεμβολής μεταξύ διανυσμάτων

Η `vector_interp(p1, p2, V1, V2, coord, dim)` συνάρτηση αυτή υλοποιεί γραμμική παρεμβολή μεταξύ δύο διανυσμάτων V_1, V_2 που αντιστοιχούν σε δύο σημεία p_1 και p_2. Δεχόμενη μια ενδιάμεση τιμή `coord` ως προς τη διάσταση `dim` (x ή y), επιστρέφει την παρεμβολημένη τιμή V στο ενδιάμεσο σημείο.

- Αν $p_1[idx] = p_2[idx]$, επιστρέφει το V_1.
- Διαφορετικά, υπολογίζει τον παράγοντα παρεμβολής $t = \frac{coord - p_1}{p_2 - p_1}$.
- Επιστρέφει $V = (1 - t) \cdot V_1 + t \cdot V_2$.

Συνάρτηση σκίασης υφής

Η `t_shading(img, vertices, uv, textImg)` είναι η βασική συνάρτηση απόδοσης υφής (texture shading) για κάθε τρίγωνο:

- Ταξινομεί τις κορυφές ως προς το y (scanline order).
- Για κάθε γραμμή y (scanline), υπολογίζει τα σημεία τομής A, B των πλευρών του τριγώνου με την γραμμή.
- Παρεμβάλει τα αντίστοιχα UV των A, B χρησιμοποιώντας τη `vector_interp`.
- Για κάθε x στο διάστημα $[A_x, B_x]$, παρεμβάλει UV και δειγματοληπτεί το χρώμα από την εικόνα υφής.
- Αποθηκεύει το τελικό χρώμα στην εικόνα εξόδου.

Η συνάρτηση αυτή είναι απαραίτητη για τον χρωματισμό με βάση την εικόνα υφής και εξασφαλίζει ομαλή παρεμβολή κατά μήκος των ακμών και scanlines.

Συνάρτηση απόδοσης εικόνας

Η `render_img(faces, vertices, vcolors, uvs, depth, shading, texImg)` είναι η κεντρική ρουτίνα που αναλαμβάνει να ταξινομήσει τα τρίγωνα και να καλέσει τη σωστή τεχνική χρωματισμού για κάθε ένα:

- Δέχεται ως είσοδο όλα τα δεδομένα: κορυφές, χρώματα, UV, βάθη, συνδεσιμότητα και τύπο shading.

- Υπολογίζει το μέσο βάθος για κάθε τρίγωνο και τα ταξινομεί κατά φθίνουσα σειρά βάθους.
- Για κάθε τρίγωνο:
  - Αν `shading = "t"`, καλεί τη `t_shading`, όπως άλλωστε ενδείκνυται να χρησιμοποιηθεί από την εκφώνηση και στην βασική συνάρτηση render_object(...).
  - (σε άλλο σενάριο, θα μπορούσε να καλέσει `f_shading`).
- Επιστρέφει την τελική εικόνα με όλα τα τρίγωνα αποδοσμένα.

Η δομή αυτών των συναρτήσεων διατηρεί την αρχιτεκτονική που υλοποιήθηκε στην προηγούμενη εργασία και επιτρέπει την ευκολία σύνδεσης της `render_img()` με νέα pipelines απόδοσης όπως η `render_object()` της παρούσας εργασίας.

Κεφάλαιο 2

Ανάλυση Υλοποίησης Συναρτήσεων Μετασχηματισμών

2.1 Περίληψη Σκοπού

Οι συναρτήσεις μετασχηματισμών της παρούσας εργασίας στοχεύουν στην υλοποίηση βασικών γραμμικών μετασχηματισμών στο ομογενές σύστημα συντεταγμένων. Ειδικότερα, παρέχονται υλοποιήσεις για μεταφορά (translation), περιστροφή (rotation) και σύνθεση μετασχηματισμών (composition). Όλες επιστρέφουν 4×4 πίνακες που χρησιμοποιούνται για την αναπαράσταση και συνδυασμό μετασχηματισμών σε 3D γραφικά.

2.2 Συνάρτηση `translate()`

Η συνάρτηση `translate()` δέχεται ένα διάνυσμα μετατόπισης $t \in \mathbb{R}^3$ και επιστρέφει τον αντίστοιχο affine μετασχηματισμό σε μορφή πίνακα 4×4.

Pseudo-code για `translate`

```
function translate(t_vec):
    xform = identity_matrix(4)
    xform[:3, 3] = t_vec
    return xform
```

Ο πίνακας που επιστρέφεται έχει τη μορφή:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3 Συνάρτηση `rotate()`

Η συνάρτηση `rotate()` παράγει τον affine μετασχηματισμό περιστροφής ως προς άξονα $a \in \mathbb{R}^3$, γωνία θ και σημείο $c \in \mathbb{R}^3$.

Pseudo-code για `rotate`

```
function rotate(axis, angle, center):
    normalize(axis)
    R = Rodrigues(axis, angle)
    t = center - R @ center
    xform = identity_matrix(4)
    xform[:3, :3] = R
    xform[:3, 3] = t
    return xform
```

Η περιστροφή βασίζεται στον αλγόριθμο του Rodrigues και μετατοπίζει το κέντρο ώστε να διατηρηθεί η περιστροφή ως προς το σημείο c.

2.4 Συνάρτηση `compose()`

Η `compose()` υλοποιεί τον συνδυασμό δύο μετασχηματισμών μέσω πολλαπλασιασμού πινάκων:

$$T_{\text{total}} = T_2 \cdot T_1$$

Pseudo-code για `compose`

```
function compose(mat1, mat2):
    return mat2 @ mat1
```

Αυτή η συνάρτηση είναι ιδιαίτερα χρήσιμη για την εφαρμογή ακολουθιών μετασχηματισμών σε αντικείμενα 3D.

2.5 Παρατηρήσεις και Συμπεράσματα

Οι παραπάνω συναρτήσεις συγκροτούν τη βάση για τον καθορισμό της θέσης και του προσανατολισμού των αντικειμένων στον 3D χώρο. Ιδιαίτερη προσοχή δίνεται στην κανονικοποίηση των διανυσμάτων και τη διατήρηση των affine ιδιοτήτων. Η υλοποίηση είναι πλήρως συμβατή με το pipeline του `render_object` της εργασίας.

Κεφάλαιο 3

Υλοποίηση βασικών συναρτήσεων

Σε αυτό το κεφάλαιο αναλύονται οι βασικές συναρτήσεις που χρησιμοποιούνται για τον υπολογισμό των μετασχηματισμών προβολής και την απόδοση του τελικού αντικειμένου στην εικόνα. Οι συναρτήσεις αυτές περιλαμβάνουν μετατροπή συντεταγμένων, ορισμό συστήματος κάμερας, προβολή και rasterization.

3.1 Μετατροπή Συντεταγμένων: `world2view()`

Η συνάρτηση `world2view` μετατρέπει τις συντεταγμένες ενός σημείου από το παγκόσμιο σύστημα αναφοράς στο σύστημα της κάμερας. Ο τύπος που χρησιμοποιείται είναι:

$$p_{\text{view}} = R \cdot (p_{\text{world}} - c_0)$$

όπου R είναι ο πίνακας περιστροφής της κάμερας και c_0 είναι η θέση της.

Pseudo-code `world2view`

```
function world2view(pts, R, c0):
    for each point p in pts:
        p_view = R @ (p - c0)
    return all transformed points as Nx3
```

3.2 Ορισμός Συστήματος Κάμερας: `lookat()`

Η `lookat` δημιουργεί το σύστημα συντεταγμένων της κάμερας με βάση τη θέση της κάμερας `eye`, το σημείο στόχο `target` και το διάνυσμα προς τα πάνω `up`. Η περιστροφή R δημιουργείται από τρεις ορθογώνιους άξονες:

$$\hat{z}_c = \frac{target - eye}{\|target - eye\|}, \quad \hat{x}_c = \frac{\hat{z}_c \times up}{\|\hat{z}_c \times up\|}, \quad \hat{y}_c = \hat{x}_c \times \hat{z}_c$$

Pseudo-code `lookat`

```
function lookat(eye, up, target):
    z = normalize(target - eye)
    x = normalize(cross(z, up))
    y = cross(x, z)
    return [x, y, -z] as rotation matrix R, and eye as translation
```

3.3 Προοπτική Προβολής: `perspective_project()`

Η συνάρτηση `perspective_project` μετασχηματίζει τα σημεία από 3D σε 2D με χρήση του εστιακού μήκους f, εφαρμόζοντας την προοπτική προβολή:

$$x' = \frac{f}{z} \cdot x, \quad y' = \frac{f}{z} \cdot y$$

και διατηρεί το βάθος κάθε σημείου.

Pseudo-code `perspective_project`

```
function perspective_project(pts, focal, R, t):
    pts_view = world2view(pts, R, t)
    for each point p:
        x' = (focal / p.z) * p.x
        y' = (focal / p.z) * p.y
    return [x', y'] and depth
```

3.4 Μετατροπή σε Εικονοστοιχεία: `rasterize()`

Η `rasterize` μετατρέπει τις κανονικοποιημένες 2D συντεταγμένες σε θέσεις εικονοστοιχείων, με βάση τις διαστάσεις του καμβά και του αισθητήρα. Οι τελικές τιμές περιορίζονται στα όρια της εικόνας.

Pseudo-code `rasterize`

```
function rasterize(pts_2d, plane_w, plane_h, res_w, res_h):
    for each point (x, y):
        pixel_x = x * scale_x + center_x
        pixel_y = -y * scale_y + center_y
        clamp to [0, res-1]
    return pixel coordinates
```

3.5 Απόδοση Αντικειμένου: render_object()

Η βασική συνάρτηση render_object συνδυάζει όλα τα παραπάνω βήματα για την τελική απόδοση του 3D μοντέλου στην εικόνα:

- Υπολογίζει τον μετασχηματισμό κάμερας (lookat)
- Εφαρμόζει την προβολή (perspective_project)
- Μετατρέπει σε pixels (rasterize)
- Επιστρέφει τελική εικόνα με χρήση render_img

Pseudo-code render_object

```
function render_object(...):
    R, t = lookat(eye, up, target)
    pts_2d, depth = perspective_project(v_pos, focal, R, t)
    pixel_coords = rasterize(pts_2d, plane_w, plane_h, res_w, res_h)
    vertices = concat(pixel_coords, depth)
    image = render_img(faces,vertices,vcolors,uvs,depth,"t",texImg)
    return image
```

Όλες οι παραπάνω συναρτήσεις έχουν υλοποιηθεί εντός του αρχείου all_functs.py

Κεφάλαιο 4

Ανάλυση αποτελεσμάτων, συμπεράσματα και οπτικοποίηση

Σε αυτό το κεφάλαιο παρουσιάζεται η ανάλυση των εκτελέσιμων αρχείων `demo1.py` και `demo2.py`, τα οποία παράγουν δυναμικές προσομοιώσεις βασισμένες στις συναρτήσεις που υλοποιήθηκαν στα προηγούμενα κεφάλαια. Η ανάλυση επικεντρώνεται στη ροή των συναρτήσεων, στη δομή των αρχείων και στην τελική παραγωγή καρέ και βίντεο.

4.1 Αρχείο `demo1.py` – Προσομοίωση με Forward Facing Camera

Το αρχείο `demo1.py` (συγκεκριμένα η `render_forward_demo(...)`) υλοποιεί την περίπτωση όπου η κάμερα είναι σταθερά στραμμένη προς την κατεύθυνση κίνησης του οχήματος.

- Η συνάρτηση `load_data()` φορτώνει τα απαραίτητα δεδομένα (`hw2.npy`) και την εικόνα υφής.
- Κάθε καρέ αντιστοιχεί σε χρονική στιγμή $t = \frac{\text{frame}}{\text{fps}}$.
- Η γωνιακή θέση του οχήματος υπολογίζεται ως $\theta = \omega t$, όπου $\omega = \frac{v}{r}$ είναι η γωνιακή ταχύτητα.
- Η θέση του αυτοκινήτου και η θέση της κάμερας προκύπτουν με χρήση ημιτονοειδών συναρτήσεων πάνω σε κυκλική τροχιά.
- Το διάνυσμα `tangent` υπολογίζεται ως εφαπτόμενο της τροχιάς και χρησιμοποιείται για τον προσδιορισμό του σημείου `target`, δηλαδή του σημείου προς το οποίο κοιτά η κάμερα.
- Η `render_object()` λαμβάνει όλα τα παραπάνω και δημιουργεί το καρέ μέσω των εξής συναρτήσεων:
  - `lookat()` – Υπολογισμός μετασχηματισμού από παγκόσμιο σε σύστημα κάμερας.
  - `perspective_project()` – Προβολή σημείων στον 2D καμβά.
  - `rasterize()` – Αντιστοίχιση προβολών σε pixels.
  - `render_img()` – Απόδοση χρωμάτων με texture shading.
- Η εικόνα αποθηκεύεται με `plt.imsave()` και προστίθεται στο βίντεο τύπου .mp4 μέσω `imageio.mimsave()`και το βίντεο διαρκεί 5 sec στα 25 fps.

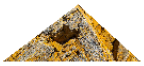
4.1.1 Αποτελέσματα Farward Facing Camera

Παρακάτω παρατίθενται ορισμένα αποτελέσματα ανά χρονική στιγμή του βίντεο (αριθμός frame) στην περίπτωση της εκτέλεσης στο demo1.py .

Καρέ 0 από `demo1.py`

Καρέ 30 από `demo1.py`

Καρέ 60 από `demo1.py`



Καρέ 90 από `demo1.py`

Σχήμα 4.1: Χαρακτηριστικά καρέ από την εκτέλεση του `demo1.py` καθώς το αντικείμενο κινείται και η κάμερα ακολουθεί τη φορά κίνησης.

Εκτελώντας το αρχείο `demo1.py` δημιουργείται ένα βίντεο 5s στα 25 fps με συνολικά 125 καρέ, αποθηκευμένο ως `demo_1_video.mp4`, ενώ οι εικόνες συλλέγονται σε folder που θα δημιουργηθεί ονόματι demo_1.

4.2 Αρχείο demo2.py – Προσομοίωση με Target Tracking Camera

Το αρχείο `demo2.py` (και ειδικότερα η `render_target_tracking_demo(...)`) υλοποιεί το σενάριο κατά το οποίο η κάμερα είναι μονίμως προσανατολισμένη σε ένα σταθερό σημείο `target` του χώρου, ανεξάρτητα από την κατεύθυνση της κίνησης του οχήματος.

- Όπως και στην προηγούμενη περίπτωση, η `load_data()` διαβάζει τα δεδομένα γεωμετρίας και την υφή.
- Η θέση του οχήματος υπολογίζεται μέσω του τύπου:

$$\theta = \omega t, \quad \omega = \frac{v}{r}$$

$$\texttt{car\_pos} = \texttt{center} + r \cdot [\cos(\theta), 0, \sin(\theta)]$$

- Η θέση της κάμερας `cam_pos` παραμένει ίδια με την περίπτωση του `demo1.py`, όμως ο στόχος `target` είναι προκαθορισμένος από το αρχείο `hw2.npy` και παραμένει σταθερός.
- Η συνάρτηση `render_object()` καλείται με διαφορετικό `target` για κάθε καρέ, ο οποίος δεν σχετίζεται με την κατεύθυνση κίνησης.
- Η απεικόνιση γίνεται και εδώ με τη χρήση texture mapping, ενώ η διαδικασία αποθήκευσης και δημιουργίας του τελικού βίντεο είναι πανομοιότυπη με την περίπτωση του demo1.py.

4.2.1 Αποτελέσματα Target Tracking Camera

Στην παρακάτω εικόνα παρουσιάζονται τέσσερα καρέ που αντιστοιχούν σε διαφορετικές χρονικές στιγμές της εκτέλεσης του `demo2.py`. Σε κάθε καρέ παρατηρείται ότι η κάμερα ακολουθεί σταθερά το σημείο στόχο, ανεξαρτήτως της κατεύθυνσης κίνησης.

Καρέ 0 από `demo2.py`

Καρέ 30 από `demo2.py`

Καρέ 80 από `demo2.py`

Καρέ 124 από `demo2.py`

Σχήμα 4.2: Χαρακτηριστικά καρέ από την εκτέλεση του `demo2.py` με την κάμερα προσηλωμένη σε σταθερό target point.

Εκτελώντας το αρχείο `demo2.py` δημιουργείται ένα βίντεο 5s στα 25 fps με συνολικά 125 καρέ, αποθηκευμένο ως `demo_2_video.mp4`, ενώ οι εικόνες συλλέγονται σε folder που θα δημιουργηθεί ονόματι demo_2.

4.3 Συμπεράσματα και Παρατηρήσεις

Η παρούσα εργασία περιελάμβανε την υλοποίηση δύο διαφορετικών σεναρίων προβολής τρισδιάστατου αντικειμένου σε 2D καμβά, μέσα από την προσομοίωση κάμερας που κινείται ή παραμένει σταθερά προσηλωμένη σε ένα σημείο στόχο. Τα αποτελέσματα από τα δύο σενάρια παρουσιάστηκαν παραπάνω.

- Στην περίπτωση του `demo1.py` , η κάμερα είναι **ευθυγραμμισμένη με τη φορά κίνησης του οχήματος**, καθώς το διάνυσμα στόχευσης προκύπτει προσθέτοντας

το `tangent` διάνυσμα στη θέση της κάμερας. Αυτό έχει ως αποτέλεσμα το αντικείμενο να αλλάζει θέση και μέγεθος μέσα στο κάδρο. Η κάμερα ακολουθεί την πορεία του οχήματος και παρατηρούμε μια συνεχή αλλαγή οπτικής γωνίας, η οποία προσφέρει μια πιο δυναμική εμπειρία παρατήρησης.

- Αντίθετα, στο `demo2.py` , η κάμερα είναι **σταθερά στραμμένη σε συγκεκριμένο στόχο** (target point) καθ' όλη τη διάρκεια της προσομοίωσης. Το σημείο αυτό παραμένει σταθερό στον χώρο, και η κάμερα περιστρέφεται γύρω του καθώς το όχημα κινείται κυκλικά. Ως αποτέλεσμα, η προοπτική παραμένει σχεδόν σταθερή και το αντικείμενο φαίνεται σχεδόν ακίνητο στο κέντρο της εικόνας, παρότι το όχημα αλλάζει θέση στο χώρο.

- Και στις δύο περιπτώσεις, η τεχνική `texture shading` λειτουργεί σωστά και αποδίδει με επιτυχία την υφή πάνω στην επιφάνεια του αντικειμένου.

- Η χρήση της συνάρτησης `lookat()` αποδείχθηκε κομβική για τον ορισμό του συστήματος συντεταγμένων της κάμερας, καθώς και για την ορθή μετατροπή των σημείων από το WCS στο σύστημα της κάμερας.

- Η χρήση διαφορετικών μεθόδων υπολογισμού του `target` οδήγησε σε δύο ενδιαφέρουσες, οπτικά διακριτές προσομοιώσεις, οι οποίες ενισχύουν την κατανόηση της λειτουργίας του πλαισίου απόδοσης εικόνας από τρισδιάστατα μοντέλα.

Συνολικά, η εργασία ανέδειξε τη σημασία του σωστού ορισμού της θέσης και κατεύθυνσης της κάμερας για την παραγωγή δυναμικής ή σταθερής οπτικής προσομοίωσης. Η επιτυχής ενσωμάτωση των συναρτήσεων `lookat, perspective_project, rasterize, render_img` και η δημιουργία των βίντεο αποτελούν απόδειξη της σωστής υλοποίησης του συστήματος προβολής και απόδοσης τρισδιάστατων δεδομένων.

Βιβλιογραφία

[1] https://docs.opencv.org/4.x/d9/df8/tutorial_root.html

[2] https://imageio.readthedocs.io/en/stable/