Ορλάνδος Αλέξιος
sdi2300149
Γραμμένο σε Linux!
Η εργασία χωρίζεται στους παρακάτω φακέλους:

- */program* : Το κεντρικό source αρχείο
- */include* : Τα header files
- */source* : Υλοποιήσεις των header files
- */shaders* : Υλοποίηση του fragment και vertex shader.
- */glad* : Το glad.c και glad.h για το OpenGL 3.3 Core.
- */build* : Το εκτελέσιμο αρχείο.
- */obj* : Τα objective ενδιάμεσα αρχεία.
- */resources* : Τα δύο δοσμένα folders για τον πλανήτη και τους κύβους.
- *README.md* : Οδηγίες για την εκτέλεση του προγράμματος.
- *makefile* :
    - Για compilation, **make**.
    - Για να τρέξει το πρόγραμμα, **make run**.
    - *Είναι αναγκαίες οι βιβλιοθικες GLFW3 και CGLM :*
        - *sudo apt install libglfw3-dev*
        - *sudo apt install libcglm-dev*

Ως προς τα περιεχόμενα του /source, έχουμε:

1) **obj_loader.c :**
    - *capacityCheck (helper)* : Βοηθάει με το reallocation των arrays καθώς διαβάζονται δεδομένα.
    - *loadObj* : Ανάγνωση των περιεχόμενων ενός object file διαβάζοντας γεωμετρικά δεδομένα (position, uv, normal). Χρησιμοποιεί τις face γραμμές ωστέ να κάνει index το τελικό αποτέλεσμα, ένα struct LoadedObject (obj_loader.h). Αυτό περιέχει ένα array από struct Vertex (3 position, 2 uv, 3 normal) (vertex.h) με όλα τα faces του αντικειμένου. Εν πράξη, ένα face είναι τρία συνεχόμενα struct Vertex στο struct LoadedObject, ξεκινώντας από την αρχή (vertex[0]). Άμεσο αποτέλεσμα, ένα position για παράδειγμα πιθανό να είναι στο struct πολλές φορές, εάν ανήκει σε πάνω από ένα faces.
    - *freeObj* : Ελευθερώνει την δυναμική μνήμη (το Vertex array) μέσα σε ένα LoadedObject)

2) **mtl_loader.c :**
    - *loadMtl* : Ανάγνωση των περιεχόμενων ενός mtl file διαβάζοντας δεδομένα για το υλικό. Τα αποθηκεύει σε struct MaterialData (mtl_loader.h).

3) **shader.c :**
    - *readFile (helper)* : Διαβάζει τα περιεχόμενα του δοσμένου file, γυρνώντας τα ως string.
    - *compileShader (helper)* : Χρησιμοποιεί τις συναρτήσεις της OpenGL για να κάνει compile το shader (δοσμένο ως string).
    - *loadShader* : Χρησιμοποιεί τις δύο helper συναρτήσεις για το fragment και vertex shader, τα κάνει attach και τα δύο σε program, και επιστρέφει αυτό.

4) **texture.c :**
    - *loadTexture* : Αρχικά χρησιμοποιεί συναρτήσεις της OpenGL ώστε να δημιουργήσει ένα texture id και να ορίσει τα σωστά texture parameters για αυτό. Έπειτα, η συνάρτηση της βιβλιοθήκης stb_image.h (ξένο library) χρησιμοποιείται ώστε να δημιουργηθεί το input που θα δοθεί στην OpenGL ώστε να γίνει bind to texture. Επιστρέφετε το texture id.

5) **camera.c :**
    - *initCamera* : Δίνει τις αρχικές τιμές στο struct Camera (camera.h). Καλεί την updateCameraMatrix(βλέπε παρακάτω) μια φορά, για πρώτη φορά.
    - *processCameraInput* : Ανάλογα με τα πατημένα κουμπιά, αλλοιώνει τα στοιχεία τις struct Camera. Συγκεκριμένα, υπάρχει αυξομείωση του angleX και angleY με τα βελάκια στο πληκτρολόγιο.
    - *updateCameraMatrix* : Υπολογίζει τον front και target vector, και με συνδυασμό την (στατική) θέση της κάμερας, καλεί την glm_lookat ώστε να ενημερώσει το view matrix (περιέχεται στο struct Camera).

Το **vertex.h** δεν έχει υλοποίηση, δήλωνει μονάχα το struct Vertex ως 3 position values, 2 texture values, και 3 normal values.
Το **cube.h** δεν έχει υλοποίηση, δηλώνει μονάχα πίνακα με τα vertex των κύβων.
Το **stb_image.h** (ξένο library) βρίσκεται στο /include.

Ως προς τα περιεχόμενα του /shaders, έχουμε:

1) **vertex.dlsl :** Μονάχα τα view, model, projection transforms.
2) **fragment.dlsl :** Εδώ εφαρμόζεται ο φωτισμός. Υπάρχουν δύο περιπτώσεις.
    - Εάν έχουμε να κάνουμε με pixel του *πλανήτη* (που είναι φωτεινή πηγή) εφαρμόζουμε μονάχα ένα glow με ένταση ίση με (material.diffuse + material.ambient). Πολλαπλασιαζουμε με το rgb του texture σε αυτό το pixel, και χρησιμοποιουμε το transparency του texture σε αυτο το σημείο.

- Εάν έχουμε να κάνουμε με pixel *κύβου*, χρησιμοποιούμε κλασικό phong φωτισμό με ambient, diffuse, specular . Επιλέγω να χρησιμοποιώ για τους κύβους το mtl του πλανήτη, μιας και ως φωτεινή πηγή, ο πλανήτης εφαρμόζει glow αντί για phong, και άρα πολλές από της τιμές του mtl δεν θα χρησιμοποιούνταν. Δεν χρησιμοποιώ την half vector μέθοδο, αλλά εφαρμόζω μείωση της φωτεινότητας με απόσταση μέσω του f(d).

Ως προς την **main.c**, έχουμε:

- *processInput (helper)* : Ανάλογα με τα πατημένα κουμπιά, μπορεί να κάνει παύση στην ροή του χρόνου (P) (η κάμερα μπορεί όμως να γυρίσει κανονικά) ή να κλείσει το παράθυρο (Esc). Καλεί επίσης την processCameraInput ωστέ να επιτρέπεται περιστροφή της κάμερας μέσω των πλήκτρων με τα βελάκια.
- **main** :
    1) *Setup* :
       Αρχικοποίηση της βιβλιοθήκης GLFW, δημιουργία παραθύρου και φόρτωση των συναρτήσεων OpenGL μέσω του GLAD.
    2) *Loading* :
       Κάλεσμα προηγουμένως ορισμένων συναρτήσεων για φόρτωση του fragment και vertex shader, των texture του πλανήτη και κύβου, το obj και mtl του πλανήτη
    3) *VAO/VBO* :
       Για τον πλανήτη: Γεμίζει τον buffer με δεδομένα από το array planet.vertices. Ορίζονται attributes για position (loc 0), texture cords (loc 1), και normal (loc 2).
       Για τους κύβους: Γεμίζει τον buffer με τα δεδομένα από τον στατικό πίνακα cubeVertices. Ορίζονται παρόμοια attributes.
    4) *Uniforms* :
       Εύρεση των τοποθεσιών των uniforms (model matrix, view matrix, **projection matrix**, planet vertices, cube vertices, light position (πλανήτης), isPlanet, **view position (κάμερα)**, **material.ambient/.diffuse/ .specular/ .shininess**) στον shader ώστε να χρησιμοποιηθούν στην επανάληψη. Τα uniforms με έντονα γράμματα δεν αλλάζουν τιμή μετά τον ορισμό τους και άρα ορίζονται εκτός της κεντρικής επανάληψης.
    5) *Main Render Loop* :
        - *Διαχείριση χρόνου* : Υπολογισμός deltaTime, activeTime(περνάει ο ενεργός χρόνος όταν το σύστημα δεν είναι σε στάση παύσης χάρη στο ‘P’)
        - *Διαχείριση κάμερας (και λοιπών εισόδων πληκτρολογίου)* : Μέσο καλέσματος της processInput.

- *Planet Rendering* :
    1) Υπολογισμός Model Matrix πλανήτη (κυκλική τροχιά γύρω από το 0,0 με βάση το χρόνο).
    2) Ενημέρωση της θέσης φωτός (lightPos) να ταυτίζεται με τη θέση του πλανήτη.
    3) Ενεργοποίηση του isPlanet = 1 (για glow shader effect).
    4) Σχεδίαση (glDrawArrays).

- *Cube Rendering* :
    1) Απενεργοποίηση του isPlanet (χρήση phong φωτισμού).
    2) Βρόχος για 6 κύβους. Για κάθε κύβο υπολογίζεται ιεραρχικός μετασχηματισμός: planet position -> local orbit (γύρω από πλανήτη) -> self rotation.
    3) Σχεδίαση (glDrawArrays).

6) *Cleanup* :
    - Απελευθέρωση της μνήμης (Shader Program, Textures, LoadedObject) και τερματισμός του παραθύρου.