University of Thessaly

Department of Electrical and Computer Engineering

ECE433 - Computer Graphics with OpenGL

Project 3: Visiting a house or a 3D space

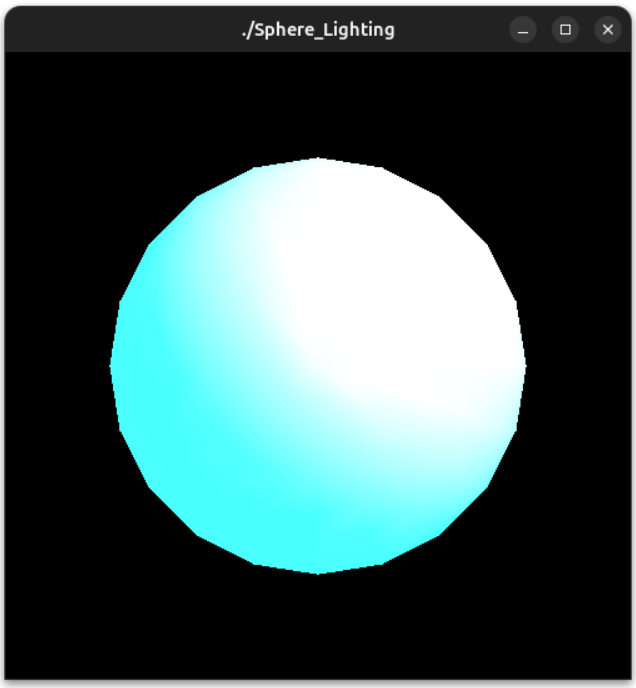
Group members
Ιωάννης Ιάσων Νικας
Αναστάσιος Καλοτσης
Παναγιώτης Νικόλαος Τσογκας

Group number: 01

January 30, 2026 (Fall 2025)

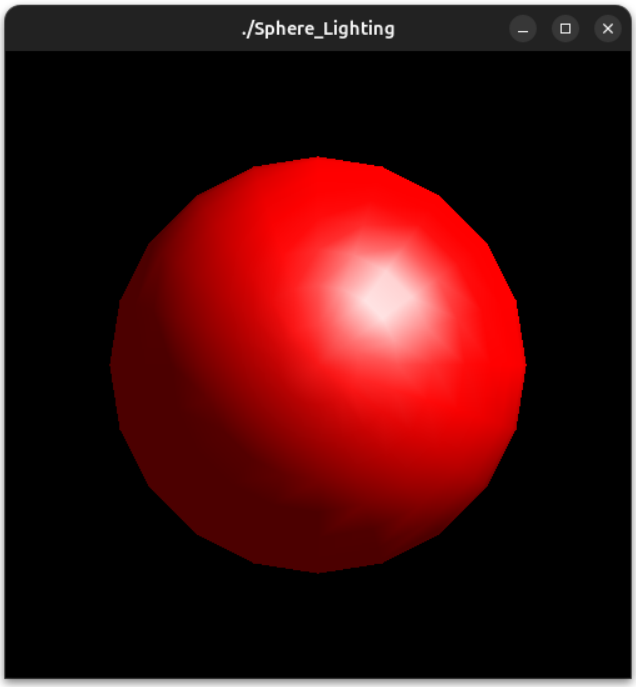
1 Exercise 1

Η αρχική σφαίρα του παραδείγματος είναι η παρακάτω:



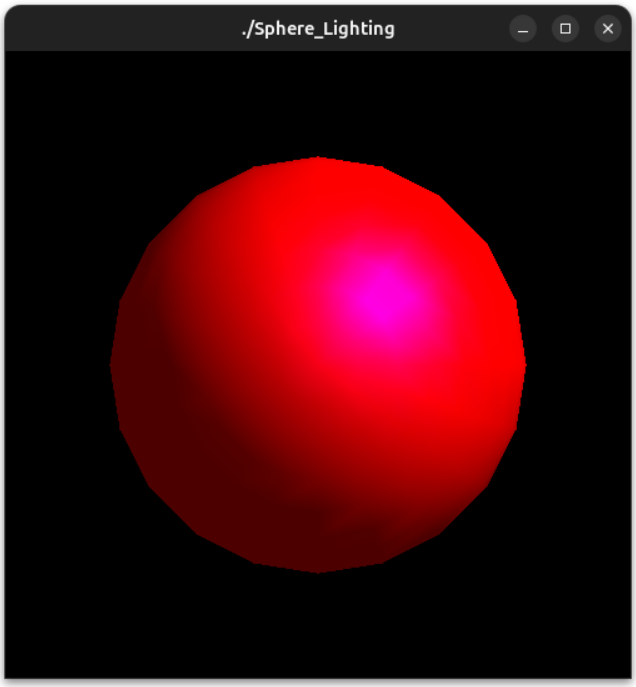
Changing sphere color:

Για να αλλάξουμε το χρώμα της σφαίρας, αλλάζουμε τις τιμές των χρωμάτων στην μεταβλητή mat_amb_diff σε $1.0, 0.0, 0.0, 1.0$, οι οποίες αντιστοιχούν στο χρώμα κόκκινο. Επιπλέον το χρώμα του φωτός είναι άσπρο. Επομένως θα αλλάξει το ambient και το diffuse lighting σε κόκκινο γιατί το υλικό απορρόφα το μπλε και το πράσινο και ανακλά μόνο το κόκκινο. Το specular παραμένει άσπρο, επείδη και το ίδιο είναι άσπρο αλλά και το χρώμα του φωτός.



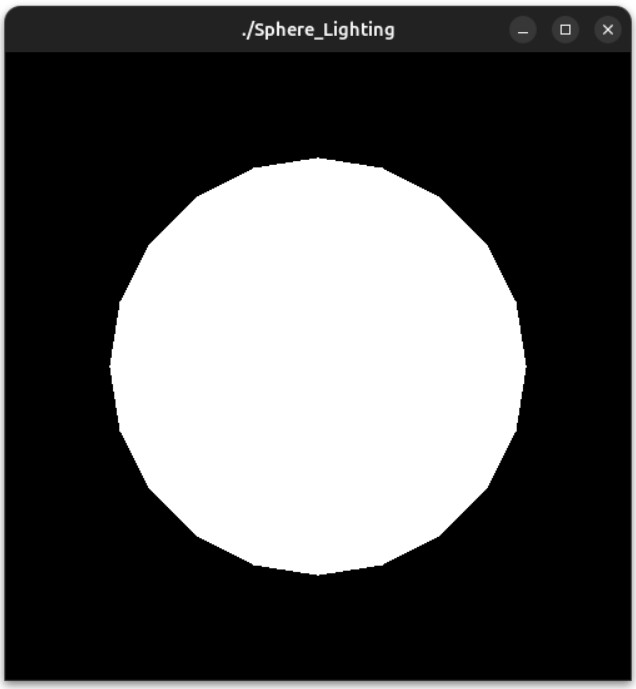
Changing light color:

Κρατάμε τα ίδια χρώματα που αναφέραμε παραπάνω και τώρα αλλάζουμε το χρώμα του φωτός(LIGHT0_light) σε $1.0, 0.0, 1.0, 1.0$, το οποίο αντιστοιχεί στο χρώμα magenta. Το χρώμα του diffuse και του ambient lighting παραμένει κόκκινο γιατί το μπλε μέρος του magenta απορροφάται από το υλικό, ενώ tospecular, αλλάζει σε magenta, επειδή το χρώμα του είναι ασπρό, οπότε το ανακλά.



Toggling lighting:

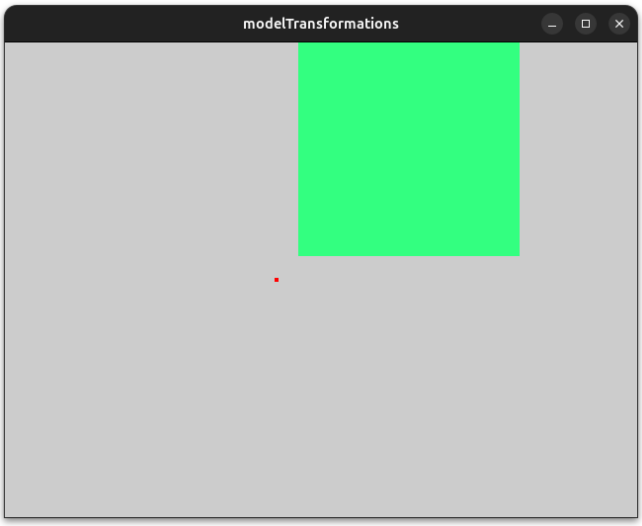
Απενεργοποιώντας τις εντολές για το φως στην OpenGL (glDisable(GL_LIGHTING)), η OpenGL αγνοεί όλες τις εντολές, που έχουν σχέση με το φως. Επομένως το χρώμα της σφαίρας θα εξαρτάται από την εντολή glColor. Το συγκεκριμένο παράδειγμα δεν έχει αυτήν την εντόλη, όποτε το χρώμα της σφαίρας θα είναι το default χρώμα που ορίζει η OpenGL δηλαδή το άσπρο.



2 Exercise 2

2.1 Using different types of culling:

Το αρχικό σχήμα του παραδείγματος είναι το παρακάτω:



Όταν αλλάζουμε το glCullFace από GL_BACK σε GL_FRONT, δεν ζωγραφίζεται τίποτα. Αυτό γινέται επειδή έχουμε ορίσει το πολύγωνο με φορά αντίθετη της κίνησης των δεικτών του ρολογιού, επομένως η OpenGL, το θεωρεί ως Front Face. Η glCullFace, με την παράμετρο που της δώσαμε αγνοεί ότι είναι Front Face και ζωγραφίζει ότι είναι Back Face. Στο Back Face όμως δεν υπάρχει τίποτα και για αυτόν τον λόγο δεν ζωγραφίζεται το πολύγωνο στην οθόνη. Τώρα όταν απενεργοποιούμε το culling το πολύγωνο εμφανίζεται κανονικά.



Figure 1: Βάζοντας GL_FRONT αντί για GL_BACK

3 Exercise 3

3.1 Applying textures to surfaces:

Αντικαταστήσαμε το τετράγωνο και το τρίγωνο με τον κύβο. Στην συνέχεια βάλαμε το texture σε κάθε πλευρά του κύβου, συνολικά 6 φορές. Τελός κάναμε λίγο rotate τον κύβο, για να φαίνονται παραπάνω από μια πλευρές.

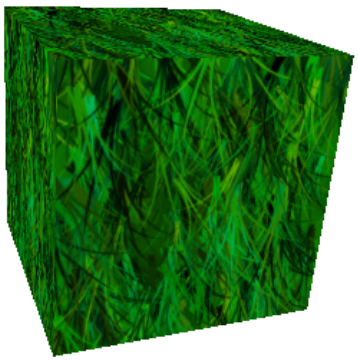


Figure 2: Παραπάνω μπορούμε να δούμε τον κύβο, με το αντίστοιχο texture.

4 Assignment

4.1 Δομή εργασίας και χειρισμός

4.1.1 Δομή φακέλων

Η υλοποίηση της εργασίας εφαρμόζει μια modular αρχιτεκτονική στην οποία ο κώδικας έχει διαχωριστεί σύμφωνα με την λειτουργία που εκτελεί. Επιπλέον τα textures και τα περισσότερα αντικείμενα βρίσκονται σε ξεχωριστούς φακέλους. Η δομή των φακέλων έχει ως εξής:

- **Πηγαίος κώδικας** (`src/`): Περιέχει το κύριο πρόγραμμα `visiting3Dhouse.cpp`, βασικές κλάσεις του παιχνιδιού (π.χ. `Object.cpp`) και τους «διαχειριστές» (π.χ. `GameManager.cpp`) που είναι υπεύθυνοι για την σωστή λειτουργία του προγράμματος.
  - **Κλάσεις για την κατασκευή του σπιτιού** (`src/house/`): Οι κλάσεις που προσφέρουν τα βασικά δομικά στοιχεία για την δημιουργία του σπιτιού. Λόγω της πολυπλοκότητας και την επαναληπτικότητα των κλάσεων αυτών τοποθετήθηκαν σε ξεχωριστό φάκελο ώστε να είναι πιο έυκολη η διαχείριση τους.
- **Ορισμοί Αντικειμένων** (`assets/`): Ο φάκελος ως η «αποθήκη» για όλα τα αντικείμενα που θέλουμε να εμφανίσουμε στον κόσμο. Κάποια αντικείμενα αντί να είναι ενσωματωμένα στον πηγαίο κώδικα του προγράμματος αποθηκεύονται σε ένα αρχείο κειμένου το οποίο φορτώνεται δυναμικά από το πρόγραμμα την ώρα της εκτέλεσης. Αυτό επιτρέπει για την άμεση επεξεργασία των αντικειμένων χωρίς να απαιτείται μεταγλώττιση.
- **Δεδομένα σκηνής** (`textures/`): Εδώ αποθηκεύονται τα αρχεία εικόνων που χρησιμοποιούνται από το πρόγραμμα για την εφαρμογή υφών στα αντικείμενα.

4.1.2 Μεταγλώττιση

Για την μεταγλώττιση χρησιμοποιήθηκε ένα `Makefile` με το οποίο πραγματοποιείται η μεταγλώττιση όλων των κλάσεων και η τελική σύνδεση και παραγωγή των εκτελέσιμων `views` και `visiting3Dhouse`. Το `Makefile` δίνει την δυνατότητα να χρησιμοποιηθουν μεταβλητές για την επιπλέον ρύθμιση των τελικών προγραμμάτων. Η μεταβλητή `USE_MOUSE` διαχειρίζεται την χρήση του ποντικιού για την περιστροφή της κάμερας και είναι ενεργοποιημένη απο προεπιλογή. Για την μεταγλώττιση σε περιβάλλον Linux και macOS προτείνεται να χρησιμοποιηθούν οι παρακάτω εντολές:

```
make all -j $(nproc)
```

Listing 1: Μεταγλώττιση σε περιβάλλον Linux με `USE_MOUSE` ενεργοποιημένο

```
make all -j $(sysctl -n hw.ncpu)
```

Listing 2: Μεταγλώττιση σε περιβάλλον macOS με `USE_MOUSE` ενεργοποιημένο

Λόγω κάποιων περιορισμών σε κάποιους διαχειριστές παραθύρων που δεν επιτρέπουν την μετακίνηση του δείκτη με την χρήση της συνάρτησης `glutWarpPointer` υπάρχει περίπτωση η χρήση του ποντικιού για την περιστροφή της κάμερας να μην λειτουργεί. Σε αυτή την περίπτωση μπορούν να χρησιμοποιηθούν οι ακόλουθες εντολές για την ενεργοποιήση χρήσης του πληκτρολογίου για την περιστροφή της κάμερας:

```
make all -j $(nproc) USE_MOUSE=0
```

Listing 3: Μεταγλώττιση σε περιβάλλον Linux με `USE_MOUSE` απενεργοποιημένο

```
make all -j $(sysctl -n hw.ncpu) USE_MOUSE=0
```

Listing 4: Μεταγλώττιση σε περιβάλλον macOS με `USE_MOUSE` απενεργοποιημένο

Επιπλέον μπορούν να προσδιοριστούν και οι υπόλοιπες μεταβλητές των οποίων η λειτουργία περιγράφεται παρακάτω:

Πίνακας 1: Makefile μεταβλητές

Μεταβλητή	**Προεπιλεγμένη τιμή**	**Λειτουργία**
`USE_MOUSE`	1	Ενεργοποιεί την περιστροφή της κάμερας με το ποντίκι
`VSYNC`	0	Ελέγχει την χρήση του VSYNC (μόνο για X window)
`MODEL`	0	Φορτώνει συνεχόμενα τα asset files

4.1.3 Χειρισμός του προγράμματος

Η πλοήγηση στον τρισδιάστατο χώρο υλοποιήθηκε με γνώμονα την ευχρηστία, ακολουθώντας τα πρότυπα των σύγχρονων εφαρμογών προσομοίωσης και παιχνιδιών (WASD controls). Παρακάτω φαίνονται οι αντιστοιχίσεις των πλήκτρων με τις λειτουργίες τους:

Πίνακας 2: Εντολές πλοήγησης και περιστροφής

Ενέργεια	**Πλήκτρο** (USE_MOUSE=1)	**Πλήκτρο** (USE_MOUSE=0)
Μετακίνηση (Κοινές Εντολές)		
Κίνηση Μπροστά	W	W
Κίνηση Πίσω	S	S
Κίνηση Αριστερά	A	A
Κίνηση Δεξιά	D	D
Ανύψωση (Fly Up)	SPACE	SPACE
Κάθοδος (Fly Down)	Z	Z
Περιστροφή Κάμερας		
Περιστροφή Πάνω	*Κίνηση Ποντικιού* ↑	I
Περιστροφή Κάτω	*Κίνηση Ποντικιού* ↓	K
Περιστροφή Αριστερά	*Κίνηση Ποντικιού* ←	J
Περιστροφή Δεξιά	*Κίνηση Ποντικιού* →	L
Παίκτης		
Εναλλαγή flight mode	F	F
Εναλλαγή φακού (φως στην camera)	E	E
Σύστημα		
Επαναφόρτωση των asset	R	R
Εναλλαγή Fullscreen	F11	F11
Έξοδος	ESC / Q	ESC / Q

4.2 Η κλάση `Object`

Η κλάση `Object` αποτελεί τη θεμελιώδη δομική μονάδα του προγράμματος, αφαιρώντας την πολυπλοκότητα της διαχείρισης της κατάστασης OpenGL από τα επιμέρους αντικείμενα. Λειτουργεί ως μια γενική αναπαράσταση μιας οντότητας στον τρισδιάστατο χώρο, προσφέροντας ενιαίο τρόπο διαχείρισης μετασχηματισμών και ιδιοτήτων υλικού.

4.2.1 Βασική ιδιότητες και μετασχηματισμοί

Κάθε αντικείμενο έχει την δική του κατάσταση η οποία αποτελείται από τις παρακάτω ιδιότητες:

- **Μετασχηματισμοί:** θέση, περιστροφή και κλιμάκωση (scale) στον τρισδιάστατο χώρο.
- **Οπτικές ιδιότητες:** Χρώμα, υλικό (material) και υφή (texture)
- **Φυσικές ιδιότητες και κατάσταση:** Ρύθμιση βαρύτητας και ρύθμιση απόκρυψης

Με αυτό τον τρόπο η διαχείριση ενός πιο περίπλοκου αντικειμένου γίνεται ευκολότερη καθώς μπορούμε να έχουμε πλήρως απομονωμένα αντικείμενα.

4.2.2 Διαχείριση αντικειμένων (`ObjectHandler`)

Η διαχείριση των στιγμιοτύπων (instances) γίνεται από ένα κεντρικό στατικό class (`ObjectHandler`) το οποίο αποθηκεύει κάθε νέο instance από `Object` μέσω του constructor του. Διατηρεί μια λίστα με όλα τα δημιουργημένα αντικείμενα (και μια για τα διαφανή αντικείμενα) επιτρέποντας μαζικές λειτουργίες.

4.2.3 Ιεραρχική Δομή (Tree-like structure)

Η κλάση υποστηρίζει μια δενδρική δομή αντικειμένων. Κάθε αντικείμενο μπορεί να λειτουργήσει σαν «γονέας» είτε και σαν «παιδί». Ο κάθε γονέας αποθηκεύει την λίστα των παιδιών του και είναι υπεύθυνος να εκτελέσει και για τα children του, οποιαδήποτε λειτουργία κληθεί στον ίδιο. Το `ObjectHandler` αφαιρεί τα αντικείμενα που έχουν parent από την λίστα του και πλέον η ανανέωση τους συμβαίνει αναδρομικά από τον parent. Τα αντικείμενα που έχουν γονέα κληρονομούν κάποιες ιδιότητες των γονέων όπως για παράδειγμα τους μετασχηματισμούς του. Αυτή η σχεδιαστική επιλογή αποδείχθηκε πολύ βοηθητική στην δημιουργία σύνθετων

αντικειμένων (π.χ. `House` που περιέχει το `Bedroom` το οποίο μπορεί να περιέχει ένα `Bed` ή ένα `Lantern` που απαρτίζεται από πολλαπλά cuboids).

4.2.4 Αφαίρεση και Πολυμορφισμός

Η κλάση περιέχει μια εικονική (virtual) μέθοδο (`drawInternal`) η οποία μπορεί να οριστεί από την κάθε υπόκλαση της. Η μέθοδος `draw` της κλάσης `Object` προετοιμάζει το περιβάλλον για την απεικόνιση (μετασχηματισμοί, υλικά, υφές) και έπειτα καλεί την εικόνική μέθοδο η οποία δίνει την δυνατότητα στις υποκλάσεις να παρέμβουν στην διαδικασία της απεικόνισης. Αυτό μας επιτρέπει να δημιουργούμε πολλά διαφορετικά σχήματα αλλα και ακόμα πιο περίπλοκα αντικείμενα:

- `Cube, Cuboid, Sphere`: Διαφορετικά σχήματα
- `RidgedWall, Lantern`: Πιο περίπλοκα σχήματα που απεικονίζουν παραπάνω από ένα πολύγωνα

4.2.5 Βελτιστοποίηση και Διαφάνεια

Για τη βελτιστοποίηση της απόδοσης, η κλάση αναγνωρίζει τα στατικά αντικείμενα (static objects) και δημιουργεί αυτόματα Display Lists κατά την πρώτη εκτέλεση, αποθηκεύοντας τις εντολές OpenGL στη μνήμη της κάρτας γραφικών. Η διαχείριση του validation/invalidation των display lists γίνεται αυτόματα μέσω των κλήσεων της `draw()` και των `setter` συναρτήσεων ιδιοτήτων, οι οποίες επηρεάζουν το τελικό αποτέλεσμα στην οθόνη. Με αυτόν τον τρόπο, ο χρήστης μπορεί να ορίσει ένα «βαρύ» αντικείμενο ως staticObject, ώστε να γίνεται re-render μόνο όταν υπάρχουν αλλαγές στη γεωμετρία του, αλλιώς καλούμε το display list. Επιπλέον, υποστηρίζεται η διαχείριση διάφανων αντικειμένων. Ο `ObjectHandler` αποθηκεύει ξεχωριστά τα αντικείμενα με διαφάνεια δίνοντας μας την δυνατότητα να τα ταξινομήσουμε σε δεύτερο χρόνο σύμφωνα με την απόσταση από την κάμερα, ώστε να απεικονίζονται με την σωστή σειρά.

4.3 Δομή των διαχειριστών (Managers)

Η αρχιτεκτονική του συστήματος βασίζεται στη χρήση στατικών κλάσεων (Managers), οι οποίες διαχωρίζουν τις ευθύνες του προγράμματος και αποτρέπουν τη χρήση καθολικών μεταβλητών στη main.

4.3.1 Διαχειριστής παιχνιδιού και κυρίως πρόγραμμα (`GameManager`, `visiting3Dhouse.cpp`)

Ο `GameManager` αποτελεί την «καρδιά» του προγράμματος. Υλοποιήθηκε ως μια στατική κλάση (όλα τα μέλη και οι μέθοδοι είναι static), επιτρέποντας την εύκολη πρόσβαση σε κρίσιμα δεδομένα (όπως η κάμερα και το περιβάλλον) από οποιοδήποτε σημείο του κώδικα, χωρίς την ανάγκη περάσματος δεικτών.

Η λειτουργία του χωρίζεται σε τέσσερις βασικούς άξονες:

- **Αρχικοποίηση σκηνής:** Μέσω της μεθόδου `init()`, ο διαχειριστής δεσμεύει μνήμη και δημιουργεί τα τρία βασικά συστατικά του κόσμου:
  1. Την `Camera`, τοποθετώντας την στην αρχική της θέση.
  2. Το `Environment`, ορίζοντας το χρώμα του ουρανού.
  3. Το `House`, το οποίο λειτουργεί ως το ριζικό αντικείμενο για το κτίριο.

  Στη συνέχεια καλείται η `loadLevelAssets()`, η οποία γεμίζει το σπίτι με έπιπλα (κρεβάτια, καναπέδες, πόρτες, κλπ.) χρησιμοποιώντας τον `AssetLoader` για να φορτώσει τα δεδομένα από αρχεία κειμένου στον φάκελο `assets/`. Επιπλέον ορίζει κάποιες global μεταβλητές οι οποίες μπορούν να χρησιμοποιηθούν απο οποιοδήποτε μέρος στον κώδικα
- **Κύριος Βρόχος Ενημέρωσης (Game Loop):** Η μέθοδος `runGameLoop()` καλείται σε κάθε καρέ (frame) μέσω της συνάρτησης `glutIdleFunc` που παρέχει η OpenGL και εκτελεί τα εξής:
  - **Υπολογισμός Delta Time (dt):** Υπολογίζει τον χρόνο που μεσολάβησε από το προηγούμενο καρέ. Η τιμή dt χρησιμοποιείται πολλαπλασιαστικά σε όλες τις κινήσεις (κάμερας και φυσικής), εξασφαλίζοντας ότι η ταχύτητα κίνησης είναι ανεξάρτητη από τον ρυθμό ανανέωσης (framerate independent movement).

- **Ενημέρωση Αντικειμένων:** Διατρέχει τη λίστα όλων των ενεργών αντικειμένων (`ObjectHandler`) και καλεί τη μέθοδο `update()` τους ώστε να εφαρμόσει οποιεσδήποτε αλλαγές χρειάζονται, όπως π.χ. η βαρύτητα.
- **Διαχείριση Εισόδου:** Καλεί τον `InputManager` για να μετακινήσει την κάμερα βάσει των πλήκτρων που πατήθηκαν.
- **Μέτρηση FPS:** Υπολογίζει τα καρέ ανά δευτερόλεπτο για την εμφάνιση στατιστικών.

- **Δυναμική διαχείριση των assets:** Ο `GameManager` παρέχει τη δυνατότητα επαναφόρτωσης, την ώρα της εκτέλεσης, (hot-reloading) της σκηνής μέσω της μεθόδου `reloadAssets()`. Αυτή η λειτουργία επιτρέπει στον χρήστη να τροποποιήσει τα αρχεία των assets (π.χ. να αλλάξει τη θέση ενός επίπλου στο .txt) και να δει τις αλλαγές πατώντας το πλήκτρο 'P', χωρίς να χρειαστεί επανεκκίνηση του προγράμματος.

- **Διαχείριση των OpenGL context:** Η μέθοδος `onWindowUpdate()` διαχειρίζεται την προβολή. Ρυθμίζει τον πίνακα προβολής (GL_PROJECTION) χρησιμοποιώντας την εντολή `gluPerspective` με οπτικό πεδίο (FOV) 80 μοιρών. Επιπλέον, σε περίπτωση αλλαγής του παραθύρου (π.χ. μετάβαση σε πλήρη οθόνη), αναλαμβάνει την επαν-αρχικοποίηση των states της OpenGL (φωτισμός, textures, depth test), καθώς αυτά χάνονται όταν αλλάζει το context.

4.3.2 Διαχειριστής χειρισμού (`InputManager`)

Ο `InputManager` αποτελεί ένα class το οποίο χρησιμοποιείται για τη διαχείριση της εισόδου από τον χρήστη, αξιοποιώντας κυρίως το πληκτρολόγιο και το ποντίκι. Περιέχει στατικές δομές, όπως τον πίνακα `pressedKeys`, ο οποίος χρησιμοποιείται για την αποθήκευση της τρέχουσας κατάστασης των πλήκτρων (πατημένο ή όχι), επιτρέποντας έτσι την ομαλή και συνεχή κίνηση της κάμερας χωρίς τις καθυστερήσεις που προκαλεί η τυπική επανάληψη χαρακτήρων του λειτουργικού συστήματος.

Ο διαχειριστής περιλαμβάνει τη μέθοδο `init()`, η οποία χρησιμοποιείται για την αρχικοποίηση των callbacks της βιβλιοθήκης GLUT. Συγκεκριμένα:

- `glutKeyboardFunc(keyboardDown)` – Ορίζει τη συνάρτηση που καλείται όταν πατηθεί ένα πλήκτρο, ενημερώνοντας τον πίνακα κατάστασης σε true ή εκτελώντας άμεσες ενέργειες (π.χ. toggle flight, flashlight).

- `glutKeyboardUpFunc(keyboardUp)` – Ορίζει τη συνάρτηση που καλείται όταν απελευθερωθεί ένα πλήκτρο, επαναφέροντας την κατάσταση στο false.

- `glutSpecialFunc(specialKeyboardDown)` – Διαχειρίζεται ειδικά πλήκτρα, όπως το F11 για την εναλλαγή σε πλήρη οθόνη.

- `glutPassiveMotionFunc(mouseMovePassive)` – (Εφόσον οριστεί το MOUSE_ROTATION) Καταγράφει την κίνηση του ποντικιού για τον υπολογισμό της περιστροφής της κάμερας.

Σημαντικό ρόλο παίζει η συνάρτηση `applyInputToCamera()`, η οποία καλείται σε κάθε frame για να εφαρμόσει τις εντολές κίνησης. Η συνάρτηση ελέγχει τον πίνακα `pressedKeys` και, ανάλογα με το ποια πλήκτρα είναι ενεργά (W, A, S, D, Space, κ.λπ.), καλεί τις αντίστοιχες μεθόδους της κλάσης `Camera` για μετακίνηση στον χώρο (Move Front, Back, Left, Right).

Επιπλέον, ο `InputManager` διαχειρίζεται την περιστροφή της κάμερας με δύο πιθανούς τρόπους, ανάλογα με τα preprocessor definitions. Στην περίπτωση του MOUSE_ROTATION, υπολογίζει τη διαφορά θέσης του δείκτη από το κέντρο της οθόνης (`pending_yaw`, `pending_pitch`) και επαναφέρει τον δείκτη στο κέντρο (`glutWarpPointer`). Εναλλακτικά, εάν δεν χρησιμοποιείται ποντίκι, η περιστροφή ελέγχεται μέσω πλήκτρων (I, J, K, L) ενημερώνοντας τον πίνακα `pressedRotationKeys` και προσαρμόζοντας τη γωνία θέασης μέσω της `updateCameraRotation()`.

Τέλος, αξίζει να τονιστεί η χρήση του `GameManager::dt` (Delta Time) κατά τον υπολογισμό της μετατόπισης εντός της `applyInputToCamera()`. Η ταχύτητα κίνησης δεν είναι σταθερή ανά frame, αλλά πολλαπλασιάζεται με τον χρόνο που μεσολάβησε από το προηγούμενο καρέ. Αυτό εξασφαλίζει ότι η ταχύτητα του χρήστη παραμένει σταθερή (Frame-Rate Independent Movement) ανεξάρτητα από την απόδοση του υπολογιστή και τον ρυθμό ανανέωσης της οθόνης. Παρόμοια λογική ακολουθείται και στην περιστροφή με

το ποντίκι, όπου εφαρμόζεται ο συντελεστής `GameManager::sensitivity` για τη ρύθμιση της ευαισθησίας απόκρισης.

4.3.3 Διαχειριστής παραθύρου (`WindowManager`)

Ο `WindowManager` υλοποιήθηκε ως μια στατική κλάση που λειτουργεί ως το επίπεδο διασύνδεσης μεταξύ της εφαρμογής και της βιβλιοθήκης GLUT. Σε αντίθεση με τον `GameManager`, δεν περιέχει τη λογική του κύριου βρόχου, αλλά είναι αποκλειστικά υπεύθυνος για τη δημιουργία, τη διαμόρφωση και τη διαχείριση του παραθύρου προβολής.

Η μέθοδος `init()` δέχεται ως ορίσματα τους δείκτες συναρτήσεων (function pointers) για τα `display` και `idle` callbacks, αποθηκεύοντάς τα για μελλοντική χρήση. Καθορίζει το display mode σε `GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH`, ενεργοποιώντας το διπλό buffering και το depth buffer.

Ιδιαίτερη έμφαση δόθηκε στη δυνατότητα εναλλαγής μεταξύ παραθυρικής λειτουργίας και πλήρους οθόνης (Game Mode). Η συνάρτηση `switchMode()` διαχειρίζεται αυτή τη μετάβαση καταστρέφοντας το τρέχον παράθυρο (`glutDestroyWindow`) και δημιουργώντας νέο (`glutEnterGameMode` ή `glutCreateWindow`), επαναφέροντας κάθε φορά τις απαραίτητες ρυθμίσεις εισόδου. Επιπλέον ενημερώνει τον `GameManager` για την αλλαγή του context ώστε να εκτελέσει τις κατάλληλες ενέργειες.

Επιπλέον λειτουργίες:

- **Οπτικοποίηση FPS:** Η μέθοδος `drawFPS()` υλοποιεί την προβολή των καρέ ανά δευτερόλεπτο. Για να επιτευχθεί αυτό, αλλάζει προσωρινά τον πίνακα προβολής σε δισδιάστατη ορθογραφική (`gluOrtho2D`) και απενεργοποιεί τον φωτισμό και το depth test, ώστε το κείμενο να σχεδιαστεί καθαρά «πάνω» από την τρισδιάστατη σκηνή.
- **Προσαρμογή μεγέθους παραθύρου:** Η συνάρτηση `reshape()` διαχειρίζεται την αλλαγή διαστάσεων του παραθύρου. Υπολογίζει τον νέο λόγο θέασης (aspect ratio) και, αντί να εφαρμόσει άμεσα τις αλλαγές στην κάμερα, μεταβιβάζει τις νέες διαστάσεις στον `GameManager::onWindowUpdate`, διατηρώντας έτσι τον διαχωρισμό ευθυνών (separation of concerns).

4.3.4 Διαχειριστής φωτισμού (`LightingManager`)

Ο `LightingManager` αποτελεί ένα class το οποίο χρησιμοποιείται για τη διαχείριση των πηγών φωτός στο πρόγραμμα. Περιέχει τη δομή `LightConfig`, η οποία χρησιμοποιείται για τον ορισμό των τιμών και των χαρακτηριστικών μιας πηγής φωτός, όπως η θέση, το χρώμα, ο τύπος φωτός, καθώς και οι παράμετροι ambient, diffuse και specular.

Επιπλέον, δίνεται η δυνατότητα ένα light source να γίνεται register σε κάποιο owner object, μέσω της (`registerLight`). Σε αυτή την περίπτωση, το φως κληρονομεί τις πληροφορίες μετασχηματισμού του αντικειμένου (translation, rotation, scale), προσθέτοντας τα αντίστοιχα τοπικά offsets, καθώς και πληροφορία ορατότητας (isHidden).

Ο διαχειριστής περιλαμβάνει επίσης τη μέθοδο `init()`, η οποία χρησιμοποιείται για την αρχικοποίηση του φωτισμού στην OpenGL. Συγκεκριμένα:

- `glEnable(GL_COLOR_MATERIAL)` – Επιτρέπει στην OpenGL να χρησιμοποιεί το χρώμα των αντικειμένων ως material, επηρεάζοντας το ambient και το diffuse lighting.
- `glEnable(GL_NORMALIZE)` – Εξασφαλίζει ότι τα normal vectors κανονικοποιούνται σωστά, ακόμα και μετά από αλλαγές κλίμακας στα αντικείμενα.
- `glEnable(GL_LIGHTING)` – Ενεργοποιεί τον μηχανισμό φωτισμού της OpenGL.
- `GLfloat globalAmbient[] = {0.01f, 0.01f, 0.01f, 0.0f}` – Ορίζει ένα χαμηλό παγκόσμιο ambient light, ώστε το πρόγραμμα να μην είναι πλήρως σκοτεινό.
- `glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globalAmbient)` – Εφαρμόζει το παγκόσμιο ambient lighting σε ολόκληρη το πρόγραμμα.
- `glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE)` – Ορίζει τον παρατηρητή ως τοπικό, βελτιώνοντας τον υπολογισμό του specular lighting.

Σημαντικό είναι να αναφερθούν και οι συναρτήσεις `updateAllLights()` και `updateLight()`, οι οποίες χρησιμοποιούνται για την ενημέρωση των πηγών φωτισμού κατά τη διάρκεια του προγράμματος.

Η `updateAllLights()` είναι υπεύθυνη για την ενεργοποίηση ή απενεργοποίηση των πηγών φωτισμού που είναι hidden, ανάλογα με την κατάσταση του αντίστοιχου owner object. Στη συνέχεια, καλεί τη `updateLight()` για όλα τα καταχωρημένα φώτα.

Η `updateLight()` εντοπίζει το mapped `LightConfig` μιας πηγής φωτισμού με βάση το αριθμητικό ID της και εφαρμόζει δυναμικά τις αντίστοιχες ρυθμίσεις στην OpenGL. Με αυτόν τον τρόπο δίνεται η δυνατότητα δυναμικής αλλαγής των φωτιστικών παραμέτρων κατά την εκτέλεση του προγράμματος.

Ένα χαρακτηριστικό παράδειγμα χρήσης αυτής της λειτουργίας είναι τα celestial bodies, τα οποία αξιοποιούν τη δυναμική ενημέρωση ώστε να αλλάζουν το χρώμα του φωτός που εκπέμπουν, προσφέροντας πιο ρεαλιστική ατμόσφαιρα στη σκηνή.

4.3.5 Διαχειριστής υφών (`TextureManager`)

Ο `TextureManager` διαχειρίζεται, αντίστοιχα με τους προηγούμενους διαχειριστές, τις διαδικασίες φόρτωσης, εφαρμογής και επεξεργασίας των textures. Επιπλέον, δίνει τη δυνατότητα δημιουργίας/σχεδίασης ενός flat τετραπλεύρου texture μέσω της συνάρτησης `drawQuadTex()`.

Σε συνεργασία με το header `TextureEnums.h`, επιτρέπει τον ορισμό textures μέσω ονομαστικών IDs (enum TextureID) και TextureMode για την επεξεργασία του u,v mapping του quadTex. Το quadTex μπορεί να χρησιμοποιηθεί για τη δομή πιο πολύπλοκων γεωμετριών, ένα παράδειγμα αποτελεί το CUBOID, το οποίο χρησιμοποιείται σαν βασικό building block για την κατασκευή των περισσότερων textured assets.

Αξίζει να σημειωθεί ότι θα μπορούσε να προστεθεί υποστήριξη για texture σχήματος triagTex, το οποίο θα ήταν αρκετά χρήσιμο για την υλοποίηση του Modeller που θα αναφερθεί παρακάτω.

- `init()` – Ενεργοποιεί την υποστήριξη 2D textures και blending, και φορτώνει προκαθορισμένα textures από αρχεία.
- `init(TextureID, std::string, GLint, int, int)` – Φορτώνει ένα texture από ένα αρχείο BMP ή RAW, δημιουργεί OpenGL texture object και παράγει mipmaps. Μπορεί να οριστεί και η μέθοδος (Filtering `magMethod`) και διαστάσεις για το texture.
- `init(TextureID, std::string, std::string)` – Φορτώνει ένα texture με συνοδευτικό mask για διαφάνεια, δημιουργεί RGBA texture και το καταχωρεί ως transparent.
- `bind(TextureID)` – Ενεργοποιεί το συγκεκριμένο texture για χρήση στην σκηνή. Αν δοθεί TextureID::NONE, απενεργοποιεί την χρήση texture.
- `drawQuadTex()` – Σχεδιάζει ένα quad με texture πάνω του (`quadTex`). Μπορεί να υποδιαιρεθεί σε πολλαπλά μικρότερα quads για καλύτερη ανάλυση mapping.
- `clear()` – Απελευθερώνει όλα τα OpenGL textures που έχουν φορτωθεί και καθαρίζει τις εσωτερικές δομές.
- `isTransparent(TextureID)` – Επιστρέφει αν το συγκεκριμένο texture έχει οριστεί ως transparent.

Είναι σημαντικό να αναφερθούμε στο κομμάτι κώδικα στην init() που δημιουργεί ένα OpenGL texture object. Η glGenTextures() παράγει ένα νέο texture ID και η glBindTexture() το ενεργοποιεί για τις επόμενες εντολές. Οι glTexParameteri() καθορίζουν τον τρόπο φιλτραρίσματος, την επανάληψη του texture (wrap S/T) και το συνδυασμό με το χρώμα του υλικού (modulate). Τέλος, η glPixelStorei(GL_UNPACK_ALIGNMENT, 4) διασφαλίζει σωστή ευθυγράμμιση των δεδομένων των pixels στη μνήμη.

4.3.6 Διαχειριστής υλικών (`MaterialManager`)

Η λογική του `MaterialManager` ακολουθεί τη φιλοσοφία του `TextureManager`, χρησιμοποιώντας binding για την αντιστοίχιση των materials σε αντικείμενα, μέσω της συνάρτησης `bind()` και ενός map. Η αρχικοποίηση των διαφορετικών τύπων Material γίνεται στη `init()`.

- `init()` – Αρχικοποιεί όλα τα προκαθορισμένα Material objects (MATTE, SHINY, COLD_LIGHT, κ.ά.) και τα αποθηκεύει στο map.

- `bind(MaterialID id)` – Ενεργοποιεί το συγκεκριμένο Material στην OpenGL σκηνή, εφαρμόζοντας τις παραμέτρους ambient, diffuse, specular, emission και shininess για το front face του αντικειμένου.

4.4 Η κλάση `Camera`

Η κλάση `Camera` υλοποιεί την έννοια του παρατηρητή στον τρισδιάστατο χώρο, προσφέροντας λειτουργικότητα First Person View. Διατηρεί την κατάσταση θέσης και προσανατολισμού και είναι υπεύθυνη για τον υπολογισμό του πίνακα προβολής (View Matrix) της OpenGL.

4.4.1 Εσωτερική κατάσταση και πεδία

Η κάμερα ορίζεται από ένα σύνολο διανυσμάτων και μεταβλητών που καθορίζουν τη θέση και τη γωνία θέασης:

- **Διανύσματα Θέσης:** Τα `Vec3 position` και `velocity` αποθηκεύουν την τρέχουσα θέση και την ταχύτητα της κάμερας αντίστοιχα.
- **Προσανατολισμός:** Τα διανύσματα `direction` (κατεύθυνση όρασης) και `up` (κάθετο διάνυσμα) ορίζουν το σύστημα συντεταγμένων της κάμερας.
- **Γωνίες Euler:** Η περιστροφή ελέγχεται μέσω των μεταβλητών `yaw` (περιστροφή γύρω από τον άξονα Υ) και `pitch` (περιστροφή γύρω από τον τοπικό άξονα X).
- **Κατάσταση:** Μεταβλητές όπως `gravity` και `flashlight` καθορίζουν αν εφαρμόζεται βαρύτητα και αν είναι ενεργός ο φακός.

4.4.2 Χειρισμός και Κίνηση

Ο έλεγχος της κάμερας γίνεται μέσω των μεθόδων `move()` και `rotate()`, οι οποίες καλούνται από τον `InputManager`.

- **Περιστροφή:** Η μέθοδος `updateDirection()` μετατρέπει τις γωνίες yaw και pitch σε καρτεσιανό διάνυσμα κατεύθυνσης χρησιμοποιώντας σφαιρικές συντεταγμένες:

$$x = \cos(yaw) \cdot \cos(pitch), \quad y = \sin(pitch), \quad z = \sin(yaw) \cdot \cos(pitch)$$

- **Μετακίνηση:** Η μέθοδος `move()` δέχεται μια κατεύθυνση (`FRONT, BACK, LEFT, RIGHT`) και μια τιμή μετατόπισης. Για την πλάγια κίνηση (strafing), υπολογίζεται το εξωτερικό γινόμενο (cross product) μεταξύ του `direction` και του `up`, εξασφαλίζοντας ορθή κίνηση στον τοπικό χώρο της κάμερας. Σημαντικό είναι ότι κατά την κίνηση στο οριζόντιο επίπεδο, αγνοείται η συνιστώσα y, ώστε ο χρήστης να μην «πετάει» όταν κοιτάζει ψηλά ή χαμηλά.

4.4.3 Φυσική και Ανανέωση (`update`)

Η μέθοδος `update()` εκτελείται σε κάθε frame και εφαρμόζει τους νόμους της φυσικής. Εφόσον η μεταβλητή `gravity` είναι ενεργή, η ταχύτητα στον άξονα y μειώνεται βάσει της επιτάχυνσης της βαρύτητας ($g \approx 9.81 m/s^2$) πολλαπλασιασμένη με το `GameManager::dt`. Επιπλέον, υλοποιείται ένας απλός έλεγχος σύγκρουσης με το έδαφος (ground collision), ο οποίος αποτρέπει την κάμερα από το να πέσει κάτω από το ύψος $y = 1.7$ μονάδες (προσομοιώνοντας το ύψος των ματιών ενός ανθρώπου). Χρησιμοποιώντας το πλήκτρο F ο χρήστης μπορεί να ενεργοποιήσει και να απενεργοποιήσει την μεταβλητή της βαρύτητας (flight mode).

4.4.4 Μετασχηματισμός OpenGL και Φωτισμός

Η μέθοδος `set()` είναι υπεύθυνη για την εφαρμογή της κάμερας στη σκηνή:

1. Καλεί τη συνάρτηση `gluLookAt`, η οποία δημιουργεί τον πίνακα μετασχηματισμού από τις συντεταγμένες κόσμου στις συντεταγμένες παρατηρητή (Eye Coordinates). Ως σημείο στόχου (center) ορίζεται το άθροισμα `position + direction`.

2. Ενημερώνει τη θέση του φακού (Flashlight). Η κάμερα περιέχει ένα αντικείμενο `LightConfig` το οποίο έχει γίνει register στον `LightingManager`. Σε κάθε καρέ, η θέση του φωτός ενημερώνεται ώστε να ταυτίζεται με τη θέση της κάμερας, δημιουργώντας το εφέ του φακού που κρατάει ο παίκτης. Με το πλήκτρο E μπορεί να ενεργοποιηθεί και να απενεργοποιηθεί αυτός ο φακός.

4.5 Οι φορτωτές `AssetLoader` και `Model`

Ο `AssetLoader` αναλαμβάνει την ανάγνωση αρχείων περιγραφής σκηνών και τη δημιουργία των αντίστοιχων Object στη μνήμη και λειτουργεί ως ένας parser. Η λειτουργία του διαχωρίζεται σε τρία διακριτά στάδια: τη λεκτική ανάλυση εντολών (command tokenization), την παραμετροποίηση ιδιοτήτων και την ιεραρχική σύνδεση (linking). Η ανάγνωση πραγματοποιείται γραμμή προς γραμμή, όπου κάθε εντολή αναλύεται μέσω string streams για την άμεση δημιουργία γεωμετρίας, λειτουργώντας ουσιαστικά ως ένα Factory παραγωγής αντικειμένων τύπου `Object`, αλλα και οτιδήποτε κάνει extend το `Object Class`. Η διαδικασία αυτή εκτελείται σε πραγματικό χρόνο με τη χρήση του flag MODEL=1 κατά τη compilation, αλλά και μέσω του κουμπιού R/r. Στη διαδικασία αυτή συμμετέχουν τα assets που έχουν οριστεί μέσα στη συνάρτηση `GameManager::loadLevelAssets()` με χρήση της `addAsset()`. Η τρέχουσα υλοποίηση κρίθηκε επαρκής για τις ανάγκες της εργασίας, ωστόσο μια βελτιωμένη προσέγγιση θα ήταν όλα τα assets που αποτελούν μέρος της Modelling Simulation και γίνονται (refresh) να καταχωρούνται σε έναν global C++ vector, ώστε να επιτυγχάνεται πιο οργανωμένη διαχείριση όπου και να ορίζονται.

Η μόνη πιο σύνθετη λογική αφορά την υποστήριξη ιεραρχίας μέσω των εντολών `BEGIN/END`. Κάθε `BEGIN` προσθέτει το τρέχον αντικείμενο σε μια στοίβα (stack), ώστε τα επόμενα αντικείμενα να συνδεθούν ως παιδιά του γονέα, ενώ κάθε `END` επαναφέρει το προηγούμενο πλαίσιο. Με αυτόν τον τρόπο δημιουργείται η σχέση γονέα-παιδιού για φωλιασμένα αντικείμενα, όπως ένα τραπέζι με πόδια ή ένα κτίριο με δωμάτια, διατηρώντας παράλληλα τις μετασχηματιστικές παραμέτρους του γονέα.

Αξίζει να σημειωθεί ότι η συγκεκριμένη λογική για Modelling Simulation είναι σχετικά σπαταλητή και χρησιμοποιήθηκε κυρίως για την αρχική τοποθέτηση των αντικειμένων στη σκηνή. Σε ένα σενάριο όπου τα αντικείμενα θα άλλαζαν συνεχώς σε πραγματικό χρόνο, θα ήταν προτιμότερο να εισαχθεί ένας μηχανισμός που αναδημιουργεί τα assets μόνο όταν αλλάζει η περιγραφή τους, ώστε να αποφευχθεί η συνεχής σπατάλη πόρων.

- `load(filename, offset)` – Η λογική του parsing.
- `addAsset(path, pos)` – Φορτώνει ένα asset και το προσθέτει στη λίστα levelAssets ως στατικό αντικείμενο και καλει την `load(filename, offset)`.
- `unloadLevelAssets()` – Διαγράφει όλα τα αντικείμενα που φορτώθηκαν μέσω addAsset και καθαρίζει τη λίστα levelAssets, απελευθερώνοντας τη μνήμη.
- `resolveTexture(name)` / `resolveMaterial(name)` – Αντιστοιχίζει συμβολοσειρές σε TextureID ή MaterialID, επιτρέποντας στα αρχεία κειμένου να αναφέρονται εύκολα σε τεξτυρες και υλικά.

Για την δομή των Asset File βλέπε A.

Η κλάση `Model` επεκτείνει τη βασική κλάση `Object` και χρησιμοποιείται για τη φόρτωση και την απεικόνιση εξωτερικών 3D models από αρχεία τύπου OBJ. Η φιλοσοφία της είναι παρόμοια με αυτή του `AssetLoader`, καθώς λειτουργεί ως ένας ελαφρύς parser γεωμετρίας, ωστόσο επικεντρώνεται αποκλειστικά στην ανάγνωση κορυφών και επιφανειών και όχι στη δημιουργία σύνθετης ιεραρχίας αντικειμένων. Ο στόχος της κλάσης `Model` είναι τα φορτωμένα models να παραμένουν εξ ολοκλήρου στατικά στο πλαίσιο της παρούσας υλοποίησης, λειτουργώντας ως σταθερά γεωμετρικά στοιχεία της σκηνής.

Η φόρτωση του model πραγματοποιείται δυναμικά κατά την πρώτη κλήση της `draw()` μέσω της `drawInternal()`, η οποία καλεί τη `loadAndCompile()`. Με αυτόν τον τρόπο, το model ενσωματώνεται στο υπάρχον σύστημα σχεδίασης χωρίς να απαιτείται ξεχωριστός μηχανισμός διαχείρισης.

Η `loadAndCompile()` διαβάζει το αρχείο OBJ γραμμή προς γραμμή, αγνοώντας κενές γραμμές και σχόλια. Τα δεδομένα αποθηκεύονται προσωρινά σε vectors και διαχωρίζονται σε τρεις βασικές κατηγορίες: θέσεις κορυφών (v), συντεταγμένες texturing (vt) και κανονικά διανύσματα (vn). Η σύνδεση αυτών των δεδομένων γίνεται μέσω των εντολών f, όπου κάθε όψη αναλύεται σε δείκτες κορυφών, υφών και κανονικών.

Για τη σχεδίαση των επιφανειών χρησιμοποιείται `GL_TRIANGLE_FAN`. Κατά τη διάρκεια της σχεδίασης εφαρμόζονται δυναμικά τα normals και τα texture coordinates, ενώ το χρώμα, το texture και το material κληρονομούνται από την κλάση `Object`.

Αξίζει να σημειωθεί ότι τα αντικείμανα σχεδιάζονται μία φορά στην αρχή του προγράμματος καθώς είναι αρχικοποιημένα ως staticObjects απο τον constructor τους.

4.6 Η υλοποίηση του σπιτιού (`House`)

Η κλάση `House` αποτελεί το αρχικό αντικείμενο της ιεραρχίας του κτιρίου. Κληρονομεί από τη βασική κλάση `Object` και λειτουργεί ως container για όλα τα επιμέρους δωμάτια και τα αντικείμενα που απαρτίζουν την κατοικία. Η κατασκευή βασίστηκε σε μια modular προσέγγιση, όπου επαναχρησιμοποιήσιμα δομικά στοιχεία συνδυάζονται για να σχηματίσουν τους τοίχους και τα δωμάτια.

4.6.1 Δομικά Στοιχεία (`BuildingBlocks`)

Για την αποφυγή επανάληψης κώδικα και τη διευκόλυνση της γεωμετρικής σχεδίασης, υλοποιήθηκε το αρχείο `BuildingBlocks.h/cpp`. Αυτό παρέχει εξειδικευμένες συναρτήσεις και κλάσεις για τα βασικά αρχιτεκτονικά στοιχεία:

- `RidgedWall:` Μια παραμετροποιήσιμη κλάση για τη διαδικαστική (procedural) δημιουργία τοίχων με ανάγλυφες ραβδώσεις. Η κλάση προσφέρει ευελιξία στην κατασκευή, επιτρέποντας στον χρήστη να ορίσει την κατεύθυνση των ραβδώσεων (Horizontal/Vertical) και την πυκνότητά τους με δύο τρόπους: είτε καθορίζοντας το συνολικό πλήθος (count) είτε την απόσταση μεταξύ τους (spacing).
- `FramedWindow:` Ένα σύνθετο αντικείμενο που αποτελείται από το πλαίσιο (μεταλλική υφή) και το τζάμι. Το τζάμι χρησιμοποιεί ειδικό texture (`glass.bmp`) και γίνεται register ως διαφανές αντικείμενο στον `ObjectHandler` για σωστή απεικόνιση (blending). Χρησιμοποιήθηκε για τα παράθυρα και τις πόρτες του σπιτιού.
- `Lantern:` Διακοσμητικά φανάρια που τοποθετούνται στην πόρτα του garage, που κάνουν register ένα light source και είναι κατασκευασμένα από συνδυασμό μικρότερων cuboids μερικά εκ των οποίων είναι και transparent.
- `RidgedPrism:` Χρησιμοποιείται για την κατασκευή της στέγης (roof) και άλλων πρισματικών δομών.

4.6.2 Διάρθρωση Δωματίων

Το σπίτι διαιρείται σε διακριτές κλάσεις, καθεμία από τις οποίες αντιπροσωπεύει έναν χώρο και ορίζει τους τοίχους, το πάτωμα και τα ανοίγματά του (πόρτες/παράθυρα). Οι κλάσεις αυτές βρίσκονται στον φάκελο `src/house/` και είναι οι εξής:

1. `Garage:` Ο χώρος στάθμευσης, ο οποίος περιέχει την γκαραζόπορτα και το μοντέλο του αυτοκινήτου. Για το αυτοκίνητο χρησιμοποιήθηκε ο Model loader και ένα .obj αρχείο.
2. `FamilyRoom :` Ο κεντρικός χώρος του σπιτιού μπροστά στην είσοδο με τραπέζι και χώρο καθιστικού.
3. `DiningRoom & Kitchen:` Χώροι εστίασης και προετοιμασίας φαγητού, σχεδιασμένοι ως ενιαίος χώρος με το σαλόνι για ανοιχτή διαρρύθμιση.
4. `Bedroom1 & Bedroom2:` Τα δύο υπνοδωμάτια του σπιτιού. Διαθέτουν διαφορετικές διαστάσεις και προσανατολισμό.
5. `Bathroom:` Ο χώρος του μπάνιου, με εξειδικευμένα assets (μπανιέρα, νιπτήρας).
6. `Hallway:` Ο διάδρομος που συνδέει τα υπνοδωμάτια και το μπάνιο με τον κυρίως χώρο.
7. `Porch:` Η βεράντα στην είσοδο του σπιτιού, η οποία περιλαμβάνει σκαλοπάτια, στύλους και διακοσμητικά φυτά.

4.6.3 Έπιπλα και Διακόσμηση

Η επίπλωση του σπιτιού δεν είναι (hardcoded) ως γεωμετρία, αλλά φορτώνεται δυναμικά μέσω του `AssetLoader` από αρχεία περιγραφής (.txt) και μοντέλα (.obj).

Συγκεκριμένα, έχουν τοποθετηθεί τα εξής αντικείμενα:

- **Υπνοδωμάτια:** Κρεβάτια (`bed.txt, bed2.txt`), ντουλάπες (`wardrobe.txt`), γραφεία, βιβλιοθήκες (`bookshelf.txt`) και τηλεόραση (`tv.obj` - φορτωμένο μέσω της κλάσης `Model`).
- **Σαλόνι:** Καναπέδες (`sofa.txt`), τραπεζάκι (`coffee.txt`), καρέκλες και poster στους τοίχους.
- **Κουζίνα:** Ψυγείο (`fridge.txt`), φούρνος (`stove.txt`), νεροχύτης (`kitchensink.txt`) και τραπεζαρία (`table.txt, chair.txt`).
- **Μπάνιο:** Μπανιέρα (`bathtub.txt`), λεκάνη (`toilet.txt`), πλυντήριο (`washing_machine.txt`) και στεγνωτήριο.
- **Γκαράζ:** Ένα λεπτομερές μοντέλο αυτοκινήτου (`Car.obj`).
- **Πίσω Αυλή:** Περίφραξη (`fence.txt`), καναπές (`sofa.txt`), τραπεζάκι (`coffee.txt`) και στέγαστρο (`teda.txt`).

Όλα τα παραπάνω αντικείμενα προστίθενται ως children στο αντικείμενο `House` (ή στα επιμέρους δωμάτια) μέσω της μεθόδου `addChildren`, επιτρέποντας τη μαζική μετακίνηση ή περιστροφή ολόκληρου του σπιτιού μαζί με τα έπιπλά του. Επιπλέον το κάθε αντικείμενο στο κάθε δωμάτιο είναι children στο object container εκείνου του δωματίου, διατηρώντας έτσι την ιεραρχία.

4.7 Η υλοποίηση του περιβάλλοντος (`Environment`)

Η κλάση `Environment` είναι υπεύθυνη για τη δημιουργία και τη διαχείριση του εξωτερικού κόσμου που περιβάλλει το σπίτι. Συνδυάζει διαδικαστική γεωμετρία (procedural geometry) με στατικά μοντέλα για να δημιουργήσει μια πλούσια σκηνή.

4.7.1 Ουράνια Σώματα και Ατμόσφαιρα

Η ατμόσφαιρα της σκηνής είναι δυναμική. Κατά την εκκίνηση (`spawn`), δημιουργούνται δύο ουράνια σώματα, ο Ήλιος (`Sun`) και η Σελήνη (`Moon`), τα οποία κληρονομούν από τη βασική κλάση `Celestial`. Η μέθοδος `updateSky()` λαμβάνει ως παράμετρο το ύψος του ήλιου (height factor) και προσαρμόζει το χρώμα του ουρανού (`glClearColor`) σε πραγματικό χρόνο. Επιπλέον αλλάζει το χρώμα του light source τον celestial σωμάτων προσδίδοντας μια ομαλή μετάβαση μεταξύ ημέρας και νύχτας. Υλοποιεί γραμμική παρεμβολή (Linear Interpolation) μεταξύ τριών καταστάσεων:

- **Νύχτα:** Σκούρο μπλε (0.02, 0.02, 0.1).
- **Ηλιοβασίλεμα:** Πορτοκαλί (0.8, 0.4, 0.2).
- **Ημέρα:** Γαλάζιο (0.4, 0.6, 0.9).

4.7.2 Έδαφος και Δάσος

Το έδαφος αναπαρίσταται από ένα εκτεταμένο επίπεδο αντικείμενο (`Cuboid`) διαστάσεων 100×100 μέτρων. Για την υφή του χρησιμοποιείται η εικόνα γρασιδιού (`TextureID::GRASS`), η οποία ρυθμίζεται σε λειτουργία επανάληψης (`TextureMode::REPEAT_FIT`) 100 φορές, ώστε να διατηρείται η υψηλή ανάλυση χωρίς παραμόρφωση. Για να το επιτύχουμε αυτό εφαρμόσαμε υφές οι οποίες έχουν επαναληπτικότητα και δεν έχουν απότομες αλλαγές. Πάνω στο έδαφος τοποθετείται το δάσος (`forest.txt`), το οποίο φορτώνεται ως εξωτερικό asset και περιβάλλει το σπίτι.

4.7.3 Οροσειρά με χρήση Splines/NURBS

Για τη δημιουργία του ορίζοντα, υλοποιήθηκε ένας δακτύλιος από βουνά που περικυκλώνει τη σκηνή. Αντί για στατικά μοντέλα, χρησιμοποιήθηκε η κλάση `NurbsSurface` για την παραγωγή γεωμετρίας βασισμένης σε μαθηματικές συναρτήσεις.

4.7.4 Ειδικά Αντικείμενα

Τέλος, στο περιβάλλον έχει τοποθετηθεί μια πύλη (Nether Portal) από το παιχνίδι (Minecraft), η οποία συνοδεύεται από μια δυναμική πηγή φωτός (`LightingManager`). Το φως ορίζεται ως μωβ χρώματος (RGB: 0.76, 0.07, 0.85) με τετραγωνική εξασθένηση (quadratic attenuation), προσθέτοντας ατμοσφαιρικό φωτισμό στη σκηνή.

4.7.5 Spline Objects

Το σύστημα `SplineObject` εισάγει υποστήριξη για καμπύλες και επιφάνειες τύπου NURBS (Non-Uniform Rational B-Splines), αξιοποιώντας τον GLU NURBS renderer. Η υλοποίηση βασίζεται σε κληρονομικότητα από την κλάση `Object`, επιτρέποντας στα spline αντικείμενα να ενσωματώνονται απρόσκοπτα στην λογική του προγράμματος, και αποτελεί έναν Wrapper για την χρήση των συναρτήσεων OpenGL.

Η κλάση `SplineObject` λειτουργεί ως κοινή βάση για όλες τα spline curve/surfaces. Ο renderer υλοποιείται ως static resource, με χρήση reference counting ώστε να αρχικοποιείται μόνο μία φορά μέσω της `gluNewNurbsRenderer()` και να αποδεσμεύεται αυτόματα όταν δεν υπάρχουν πλέον ενεργά σπλινε αντικείμενα.

Επιπλέον, η `SplineObject` παρέχει βοηθητική λογική για τη δημιουργία clamped knot vectors, απαραίτητων για τον ορισμό B-spline καμπυλών και επιφανειών.

Η κλάση `NurbsCurve` υλοποιεί NURBS curves μίας διάστασης. Τα ςοντρολ ποιντς μπορούν να δοθούν είτε άμεσα ως λίστα σημείων, είτε έμμεσα μέσω παραμετρικής lamda/anonymous συνάρτησης $f(t)$, και υπολογίζεται το grid απο τον constructor. Η σχεδίαση πραγματοποιείται μέσω GLU, με χρήση `gluBeginCurve()` και `gluNurbsCurve()`.

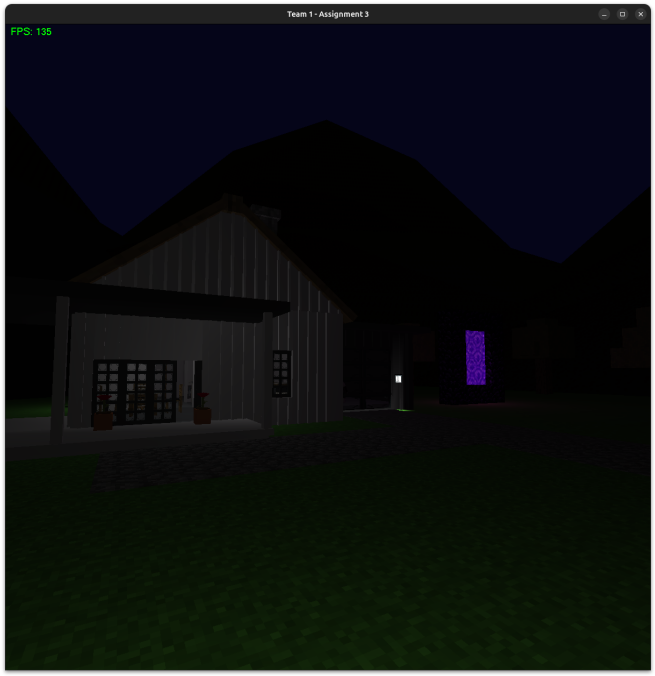
Η κλάση `NurbsSurface` επεκτείνει τη λογική αυτή σε NURBS surfaces δύο παραμέτρων (u,v). Οι επιφάνειες μπορούν να οριστούν είτε από πλέγμα control points είτε από παραμετρική lamda/anonymous συνάρτηση $f(u, v)$. Υποστηρίζεται πλήρως η χρήση material και texture, καθώς και προσαρμοζόμενο texture mapping μέσω `TextureConfig`, επιτρέποντας επαναλαμβανόμενα ή προσαρμοσμένα UV coordinates. Η επιφάνεια σχεδιάζεται μέσω `gluBeginSurface()` και `gluNurbsSurface()`, με ξεχωριστό NURBS map για τη γεωμετρία και για τα texture coordinates.

- `SplineObject::initRenderer()` – Αρχικοποιεί τον κοινό GLU NURBS renderer και ορίζει βασικές ιδιότητες όπως sampling tolerance και display mode.
- `generateClampedKnots()` – Δημιουργεί clamped knot vectors για καμπύλες και επιφάνειες.
- `NurbsCurve::updateControlPoints()` – Ενημερώνει τα control points και αναδημιουργεί τα αντίστοιχα knots.

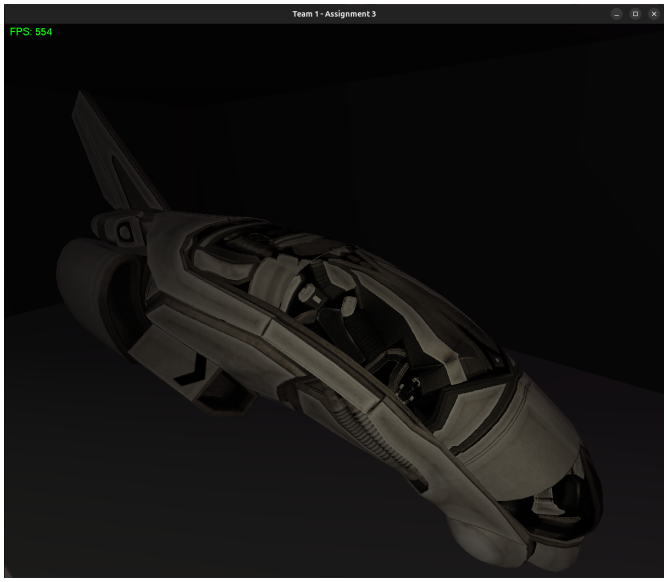
4.8 Τελικό αποτέλεσμα



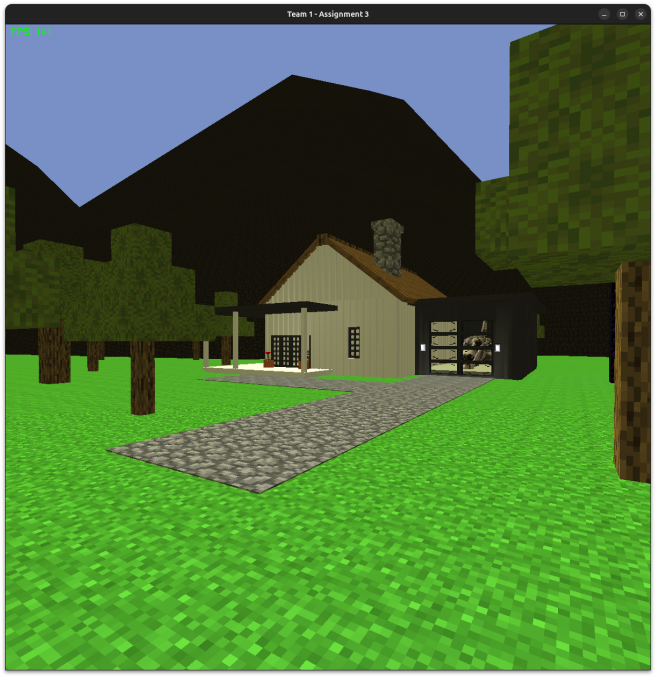
Σχήμα 3: Το αποτέλεσμα του προγράμματος views



Σχήμα 4: Το αποτέλεσμα του προγράμματος visiting3DHouse (Night)



Σχήμα 6: Futuristic Car Model (Night)



Σχήμα 5: Το αποτέλεσμα του προγράμματος visiting3DHouse (Day)



Σχήμα 7: Futuristic Car Model (Day)



Σχήμα 8: Bedroom

A Appendix

A.1 File Format

A′.1.1 Format αρχείων `AssetLoader`

Με βάση τον κώδικα του `AssetLoader.h` και τα αρχεία παραδειγμάτων (table.txt, bed.txt), τα αρχεία assets χρησιμοποιούν μια κειμενική δομή (text-based) που διαβάζεται γραμμή προς γραμμή.

Η λογική σύνταξης των αρχείων συνοψίζεται ως εξής:

- **Βασική Δομή & Σχόλια:** Κάθε γραμμή ξεκινά με μια λέξη-κλειδί (command) ακολουθούμενη από παραμέτρους. Γραμμές που ξεκινούν με # ή είναι κενές αγνοούνται. Συνήθως ξεκινάμε με ένα OBJECT που λειτουργεί ως

  άγκυρα (anchor) για τη μετακίνηση ολόκληρου του asset.

- **Δημιουργία Αντικειμένων (Primitives):** Η γεωμετρία ορίζεται με τη θέση (x y z) και τις διαστάσεις (width, height, length):

  - OBJECT x y z – Δημιουργεί ένα αόρατο σημείο/container, χρήσιμο για γονέα.
  - CUBOID x y z w h l – Δημιουργεί ορθογώνιο παραλληλεπίπεδο.
  - SPHERE x y z – Δημιουργεί σφαίρα.

- **Ιεραρχία (Parenting):** Η ιεραρχία των αντικειμένων υποστηρίζεται μέσω μιας στοίβας (stack):

  - BEGIN – Το τελευταίο αντικείμενο γίνεται γονέας και τα επόμενα αντικείμενα που δηλώνονται ανήκουν σε αυτό (θα κινούνται μαζί του).
  - END – Κλείνει την ομάδα και επαναφέρει το προηγούμενο πλαίσιο ιεραρχίας.

- **Ιδιότητες (Properties):** Μετά τον ορισμό ενός αντικειμένου, μπορούν να οριστούν ιδιότητες:

  - TEXTURE [name] – Ορίζει την υφή (π.χ. TEXTURE SPRUCE_PLANKS).
  - TEX_CONFIG [MODE] – Ρυθμίζει το πώς εφαρμόζεται η υφή (π.χ. REPEAT_FIT ή CUSTOM u v).
  - ROTATION angle x y z – Περιστροφή γύρω από τον άξονα (x,y,z) σε μοίρες.
  - SCALE x y z – Αλλαγή κλίμακας.
  - GRAVITY true/false – Ορίζει αν επηρεάζεται από βαρύτητα.
  - MATERIAL [name] – Ορίζει το υλικό του αντικειμένου, που επηρεάζει την αλληλεπίδραση με το φωτισμό (π.χ. MATERIAL MATTE, MATERIAL SHINNY, MATERIAL COLD_LIGHT).
  - SET_SUBDIVS [int] – Ορίζει τον αριθμό υποδιαιρέσεων (subdivisions) για αντικείμενα τύπου CUBOID.

  Περιορισμός: Η εντολή SET_SUBDIVS υποστηρίζεται μόνο για αντικείμενα τύπου CUBOID, διαφορετικά προκαλεί σφάλμα. Η χρήση της βοηθά στον καλύτερο υπολογισμό φωτισμού και στην εφαρμογή επαναλαμβανόμενων υφών σε μεγάλες επιφάνειες.