Πρώτο Μέρος (1Β) του Συνόλου Προγραμματιστικών Ασκήσεων
Γραφικά Υπολογιστών και Συστήματα Αλληλεπίδρασης

Ελευθέριος Ιωσηφίδης, 5233
Δανάη Χανλαρίδου, 5386

14/11/2025

1. Περιγραφή της εργασίας

Στόχος ήταν η δημιουργία μιας απλής 3D σκηνής σε OpenGL (GLFW/GLEW/GLM) με:

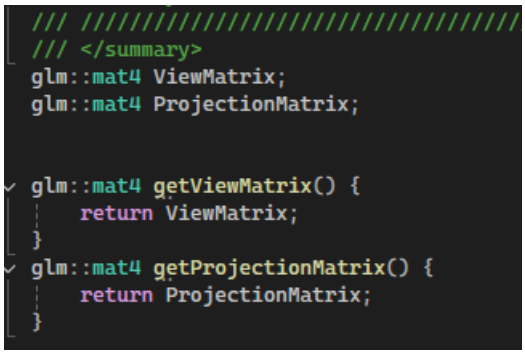
- παράθυρο 850×850, σκούρο γκρι φόντο, ελληνικό τίτλο, τερματισμό με πλήκτρο 1,
- σχεδίαση 3D χαρακτήρα A (ορθογώνιο παραλληλεπίπεδο + πυραμίδα),
- διαφορετικό χρώμα σε κάθε πλευρά του A,
- αρχική κάμερα στο (0, −5, 20), lookAt(0,0,0), up=(0,1,0), FOV=60° σταθερό,
- κίνηση του A στον άξονα x με βήμα a/2 σε πλήκτρα L/J (press-only),
- πέντε κύβοι πάνω από τον A, με δεδομένη αρχική κορυφή K του πρώτου κύβου στο (−9, 10, 0), m=2, απόσταση n=2,
- κάμερα ελεγχόμενη μόνο με πληκτρολόγιο (press-only):
  W/X γύρω από x, Q/Z γύρω από y, Numpad +/− για zoom (μεταβολή ακτίνας, όχι FOV).

Εξήγηση του κώδικα που μας δίνεται:

Χρησιμοποιούνται οι βιβλιοθήκες:
GLFW: δημιουργεί παράθυρο/Context OpenGL και χειρίζεται input.
GLEW: φορτώνει τις OpenGL συναρτήσεις (extensions).

GLM: μαθηματική βιβλιοθήκη (διανύσματα/μήτρες) τύπου C++ για OpenGL.

Global μεταβλητές & βοηθητικές συναρτήσεις



Η συνάρτηση void camera_function() παρατήρουμε πως είναι κενή αλλά με αυτήν θα χειριστούμε το πληκτρολόγιο και το ποντίκι στην συνέχεια της άσκησης.

Η συνάρτηση LoadShaders(…) ουσιαστικά:

- Διαβάζει τα αρχεία shader από δίσκο (vertex & fragment).
- Δημιουργεί δύο shader objects (glCreateShader).
- glShaderSource + glCompileShader για καθένα.
- Έλεγχο λαθών: glGetShaderiv(..., GL_INFO_LOG_LENGTH) και εκτύπωση μηνυμάτων σφαλμάτων/προειδοποιήσεων.

- Δημιουργεί πρόγραμμα (glCreateProgram), κάνει attach τους δύο shaders, link.
- Έλεγχος link και εκτύπωση log αν υπάρχει.
- Καθαρίζει (detach & delete shader objects) και επιστρέφει το ProgramID.

Η συνάρτηση main(void):

glfwInit() και ρυθμίσεις για OpenGL 3.3 Core.

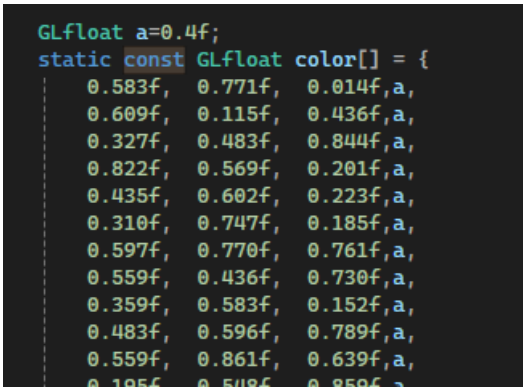
- Δημιουργεί παράθυρο 800×800 και κάνει το context current.
- glewInit() για να φορτωθούν οι pointers των OpenGL συναρτήσεων.
- glfwSetInputMode(..., GLFW_STICKY_KEYS, GL_TRUE) για να «πιάνει» το ESC.
- Καθορίζει χρώμα φόντου (γκρι) και ενεργοποιεί βάθος: glEnable(GL_DEPTH_TEST).
- Δημιουργεί και κάνει bind ένα **VAO** (υποχρεωτικό στο Core profile).
- Φορτώνει/κάνει link τους shaders:
  programID = LoadShaders("P1BVertexShader.vertexshader",
  "P1BFragmentShader.fragmentshader");
- Παίρνει το location του uniform **MVP**:
  MatrixID = glGetUniformLocation(programID, "MVP");

Επίσης έχουμε τους εξής πίνακες προβολής/κάμερας/μοντέλου:

```
glm::mat4 Projection = glm::perspective(glm::radians(45.0f), 4.0f / 4.0f, 0.1f, 100.0f
// Camera matrix
glm::mat4 View = glm::lookAt(
    glm::vec3(5.0f, 5.0f, 5.0f),
    glm::vec3(0.0f, 0.0f, 0.0f),
    glm::vec3(0.0f, 0.0f, 1.0f)
);

glm::mat4 Model = glm::mat4(1.0f);

glm::mat4 MVP = Projection * View * Model;
```

- Projection = Προοπτική με FOV 45°, τετράγωνο aspect 1:1 (4/4).
- View = Κάμερα στη (5,5,5), κοιτάει την αρχή, με άξονα z προς τα πάνω.
- Model = ταυτότητα (το αντικείμενο στην αρχή των αξόνων).
- MVP = P·V·M (τάξη πολλαπλασιασμού της GLM).

Στη συνέχεια, η γραμμή:
static const GLfloat cube[] περιέχει 36 κορυφές = 12 τριγωνα = 6 πλευρές κύβου (2 τρίγωνα ανά πλευρά). Για κάθε κορυφή δίνεται vec4 (συμπεριλ. alpha=0.4 για ημιδιαφάνεια). Δεν έχει ενεργοποιηθεί blending, άρα το alpha δεν θα έχει ορατό αποτέλεσμα (μόνο αν το shader το χρησιμοποιεί ειδικά).



```
GLfloat a=0.4f;
static const GLfloat color[] = {
    0.583f,  0.771f,  0.014f,a,
    0.609f,  0.115f,  0.436f,a,
    0.327f,  0.483f,  0.844f,a,
    0.822f,  0.569f,  0.201f,a,
    0.435f,  0.602f,  0.223f,a,
    0.310f,  0.747f,  0.185f,a,
    0.597f,  0.770f,  0.761f,a,
    0.559f,  0.436f,  0.730f,a,
    0.359f,  0.583f,  0.152f,a,
    0.483f,  0.596f,  0.789f,a,
    0.559f,  0.861f,  0.639f,a,
```

Δημιουργία Buffer Objects

glGenBuffers δύο VBOs:

- vertexbuffer για θέσεις (δένεται ως GL_ARRAY_BUFFER, περνάει τα cube).
- colorbuffer για χρώματα (περνάει τα color).

Ο βρόχος σχεδίασης έχει ως εξής:

Μέχρι να πατηθεί ESC ή κλείσει το παράθυρο:

1. glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT).
2. glUseProgram(programID).
3. Ξαναϋπολογίζει Projection/View/Model (ίδιες τιμές κάθε frame) και στέλνει MVP με glUniformMatrix4fv.
4. Δέσμευση & περιγραφή attributes:
    - Location 0 (θέση): 3 floats, stride 0, offset 0, από vertexbuffer.
    - Location 1 (χρώμα): 4 floats, stride 0, offset 0, από colorbuffer.
5. glDrawArrays(GL_TRIANGLES, 0, 36) για να σχεδιάσει τα 12 τρίγωνα.
6. glfwSwapBuffers & glfwPollEvents.

Τέλος, έχουμε τον καθαρισμό των πόρων

- glDeleteBuffers(1, &vertexbuffer);
- **(Λείπει)** glDeleteBuffers(1, &colorbuffer); ← πρέπει να προστεθεί.
- glDeleteVertexArrays(1, &VertexArrayID);
- glDeleteProgram(programID);
- glfwTerminate();

Τρέχουμε τον αρχικό κώδικα και στην οθόνη εμφανίζεται:

(i) Για την επίλυση αυτού του ερωτήματος έχουμε το παρακάτω screenshot όπου φαίνεται η δημιουργία ενός 850*850 παραθύρου, με γκρι φόντο και ελληνικό τίτλο. Για την σωστή χρήση των u nicodes, χρησιμοποιήσαμε το chat αλλιώς στο παράθυρο εμφανιζόντουσαν “??%?%...”

```
//(i) Window 850x850, dark gray bg, Greek title
if (!glfwInit()) { fprintf(stderr, "Failed to initialize GLFW\n"); return -1; }
glfwWindowHint(GLFW_SAMPLES, 4);
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

window = glfwCreateWindow(850, 850, u8"\u0395\u03C1\u03B3\u03B1\u03C3\u03AF\u03B1 1\u0392 \u2013 2025 \u2013 \u03A3\u03BA\u03B7\u03BD\u03B

if (!window) { fprintf(stderr, "Failed to open GLFW window\n"); glfwTerminate(); return -1; }
glfwMakeContextCurrent(window);
glewExperimental = true;
if (glewInit() != GLEW_OK) { fprintf(stderr, "Failed to initialize GLEW\n"); glfwTerminate(); return -1; }

glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);
glClearColor(0.15f, 0.15f, 0.15f, 1.0f);        // dark grey
glEnable(GL_DEPTH_TEST);
```

Τερματισμός με «1» υλοποιείται εδώ:

```
// quit with '1'
if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS) glfwSetWindowShouldClose(window, GLFW_TRUE);

// move J/L one step per press
bool L = glfwGetKey(window, GLFW_KEY_L) == GLFW_PRESS;
```

(ii) Σχεδιασμός χαρακτήρα A (παραλληλεπίπεδο + πυραμίδα) & αποθήκευση τριγώνων
Διαστάσεις:

- a = 3, b = a/2 = 1.5, c = 2, h = a/4 = 0.75.
- Δίνεται apex: v9 = (0, −7.75, 1.0).
- Πάνω επιφάνεια βάσης: yTop = v9.y − h = −7.75 − 0.75 = −8.5
- Κάτω επιφάνεια βάσης: yBot = yTop − b = −8.5 − 1.5 = −10.0
- x-όρια: xL = −a/2 = −1.5, xR = +a/2 = +1.5

- z-όρια: zF = 0 (μπροστά), zB = c = 2 (πίσω)

Κορυφές παραλληλεπιπέδου (v1…v8), με όρο «v1..v4 πάνω στο z=0»:

- v1(xL, yBot, zF) = (−1.5, −10.0, 0)
- v2(xR, yBot, zF) = (+1.5, −10.0, 0)
- v3(xR, yTop, zF) = (+1.5, −8.5, 0)
- v4(xL, yTop, zF) = (−1.5, −8.5, 0)
- v5(xL, yBot, zB) = (−1.5, −10.0, 2)
- v6(xR, yBot, zB) = (+1.5, −10.0, 2)
- v7(xR, yTop, zB) = (+1.5, −8.5, 2)
- v8(xL, yTop, zB) = (−1.5, −8.5, 2)
- v9 = (0, −7.75, 1)

Στο παρακάτω screenshot φαίνεται πως γεμίζονται vector<glm::vec3> Averts με κορυφές (triplets) και vector<glm::vec4> Acols με χρώματα ανά τρίγωνο.

```
//(ii)+ (iii) Character A geometry + per-face colors
vector<glm::vec3> Averts; vector<glm::vec4> Acols;
build_A(Averts, Acols);

GLuint vboA, cboA;
glGenBuffers(1, &vboA); glBindBuffer(GL_ARRAY_BUFFER, vboA);
glBufferData(GL_ARRAY_BUFFER, Averts.size() * sizeof(glm::vec3), Averts.data(), GL_STATIC_DRAW);
glGenBuffers(1, &cboA); glBindBuffer(GL_ARRAY_BUFFER, cboA);
glBufferData(GL_ARRAY_BUFFER, Acols.size() * sizeof(glm::vec4), Acols.data(), GL_STATIC_DRAW);
```

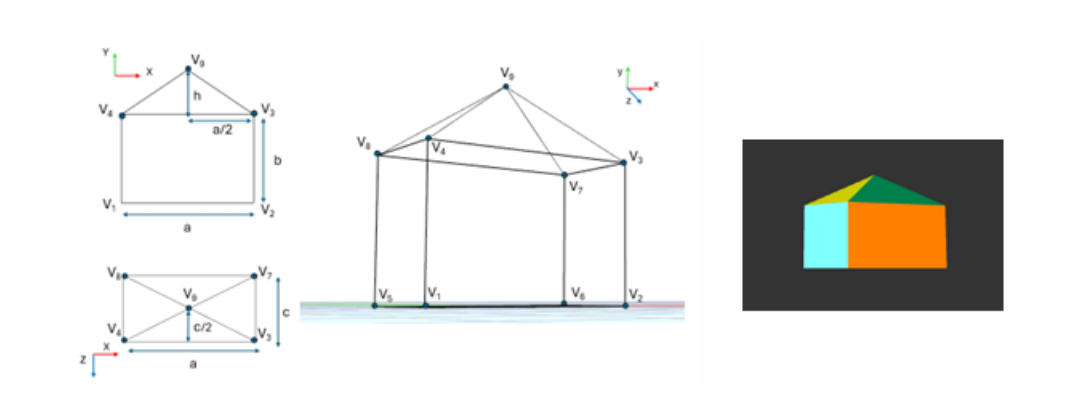
Η αποστολή στη GPU γίνεται ως εξής:

```
GLuint vboA, cboA;
glGenBuffers(1, &vboA); glBindBuffer(GL_ARRAY_BUFFER, vboA);
glBufferData(GL_ARRAY_BUFFER, Averts.size() * sizeof(glm::vec3), Averts.data(), GL_STATIC_DRAW);
```

Εδώ φαίνεται το κομμάτι του κώδικα που έχει την σήμανση των v1..v9 και τις τιμές τους.

```
// Base (12 tris)
// front z=0
tri(verts, v1, v2, v3); tint(cols, cFront);
tri(verts, v1, v3, v4); tint(cols, cFront);
// back z=c
tri(verts, v5, v7, v6); tint(cols, cBack);
tri(verts, v5, v8, v7); tint(cols, cBack);
// left x=xL
tri(verts, v1, v4, v8); tint(cols, cLeft);
tri(verts, v1, v8, v5); tint(cols, cLeft);
// right x=xR
tri(verts, v2, v6, v7); tint(cols, cRight);
tri(verts, v2, v7, v3); tint(cols, cRight);
// top y=yTop
tri(verts, v4, v3, v7); tint(cols, cTop);
tri(verts, v4, v7, v8); tint(cols, cTop);
// bottom y=yBot
tri(verts, v1, v5, v6); tint(cols, cBot);
tri(verts, v1, v6, v2); tint(cols, cBot);

// Roof (4 tris): (v4,v3),(v3,v7),(v7,v8),(v8,v4) with apex v9
tri(verts, v4, v3, v9); tint(cols, r1);
tri(verts, v3, v7, v9); tint(cols, r2);
tri(verts, v7, v8, v9); tint(cols, r3);
tri(verts, v8, v4, v9); tint(cols, r4);
```

Το σχεδιαστικό κομμάτι όπως δίνεται στην εκφώνηση είναι:



Εικόνα 2 – (πάνω αριστερά) Η μπροστινή όψη του Α (front view), (κάτω αριστερά) η όψη του Α από πάνω (top view), (μέση) η προβολή του Α υπό γωνία, (δεξιά) ο χρωματισμός του Α.

(iii) Διαφορετικά χρώματα σε κάθε πλευρά του Α

- Σημεία κώδικα: πάλι στο build_A(...) ορίζονται 10 διαφορετικά χρώματα (6 επιφάνειες βάσης + 4 της πυραμίδας) και με tint(cols, ...) αντιστοιχίζονται ανά τρίγωνο.
- Χρήση στον shader: layout(location=1) vec4 vCol (vertex shader περνάει fCol → fragment).
- Δέσιμο attribute:

```
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, cboA);
glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, (void*)0);
```

Εδώ, φαίνεται ο χρωματισμός του Α με διαφορετικές αποχρώσεις ανά πλευρά. Τα cFront, cBack, cLeft, cRight, cTop, cBot είναι οι 6 αποχρώσεις για τις πλευρές του βασικού παραλληλεπιπέδου (“βάσης”). Τα r1, r2, r3, r4 είναι οι 4 διαφορετικές αποχρώσεις για τα τρίγωνα της πυραμίδας (“στέγης”) του Α. Κάθε μία είναι glm::vec4(R, G, B, A) με τιμές από 0–1.

```
// Colors: 6 faces + 4 roof tris (all different)
glm::vec4 cFront(0.95, 0.50, 0.10, 1), cBack(0.10, 0.70, 0.95, 1),
    cLeft(0.20, 0.85, 0.35, 1), cRight(0.85, 0.20, 0.60, 1),
    cTop(0.95, 0.90, 0.20, 1), cBot(0.30, 0.30, 0.30, 1),
    r1(0.10, 0.60, 0.20, 1), r2(0.20, 0.25, 0.85, 1),
    r3(0.80, 0.25, 0.15, 1), r4(0.15, 0.75, 0.70, 1);
```

Ακριβώς παρακάτω, σε κάθε επιφάνεια του A, καλείται:

```
// Base (12 tris)
// front z=0
tri(verts, v1, v2, v3); tint(cols, cFront);
tri(verts, v1, v3, v4); tint(cols, cFront);
// back z=c
tri(verts, v5, v7, v6); tint(cols, cBack);
tri(verts, v5, v8, v7); tint(cols, cBack);
// left x=xL
tri(verts, v1, v4, v8); tint(cols, cLeft);
tri(verts, v1, v8, v5); tint(cols, cLeft);
// right x=xR
tri(verts, v2, v6, v7); tint(cols, cRight);
tri(verts, v2, v7, v3); tint(cols, cRight);
// top y=yTop
tri(verts, v4, v3, v7); tint(cols, cTop);
tri(verts, v4, v7, v8); tint(cols, cTop);
// bottom y=yBot
tri(verts, v1, v5, v6); tint(cols, cBot);
tri(verts, v1, v6, v2); tint(cols, cBot);

// Roof (4 tris): (v4,v3),(v3,v7),(v7,v8),(v8,v4) with apex v9
tri(verts, v4, v3, v9); tint(cols, r1);
tri(verts, v3, v7, v9); tint(cols, r2);
tri(verts, v7, v8, v9); tint(cols, r3);
tri(verts, v8, v4, v9); tint(cols, r4);
```

Η tint(cols, cFront) σημαίνει = “Βάλε το ίδιο χρώμα cFront και στα τρία τρίγωνα αυτά”. Η συνάρτηση αυτή προσθέτει το ίδιο χρώμα τρεις φορές, ένα για κάθε κορυφή του τριγώνου. Έτσι, κάθε τρίγωνο του αντικειμένου παίρνει τη δική του ενιαία απόχρωση.

```
static void tri(vector<glm::vec3>& v, glm::vec3 A, glm::vec3 B, glm::vec3 C) {
    v.push_back(A); v.push_back(B); v.push_back(C);
}
static void tint(vector<glm::vec4>& c, glm::vec4 col) {
    c.push_back(col); c.push_back(col); c.push_back(col);
}
```

Η tri() παίρνει τρεις κορυφές ενός τριγώνου (A, B, C) και τις προσθέτει (append) στο vector v, το οποίο περιέχει όλες τις κορυφές του αντικειμένου. Η tri(...) προσθέτει τις 3 κορυφές (γεωμετρία), ενώ η tint(...) προσθέτει το ίδιο χρώμα για τις 3 κορυφές. Για tri(verts, v1, v2, v3) σημαίνει = Πρόσθεσε στο vector verts τα σημεία v1, v2, v3 με αυτή τη σειρά.

(iv)Αρχική κάμερα (0, −5, 20), στόχος (0,0,0), up (0,1,0), FOV=60° (σταθερό)

- Projection μία φορά (σταθερό FOV=60°):

```
//(iv) Camera initial pose (0,-5,20) -> lookAt origin; FOV fixed 60
const float FOV = 60.0f;
glm::mat4 Projection = glm::perspective(glm::radians(FOV), 1.0f, 0.1f, 200.0f);
float camYaw = 0.0f, camPitch = 0.0f, camR = 20.0f;             // keep FOV fixed; zoom by radius
```

- View από «σφαιρικές» παραμέτρους (ακτίνα camR=20, γωνίες camYaw/pitch):

```
glm::vec3 eye = camPos();
glm::mat4 View = glm::lookAt(eye, glm::vec3(0, 0, 0), glm::vec3(0, 1, 0));
```

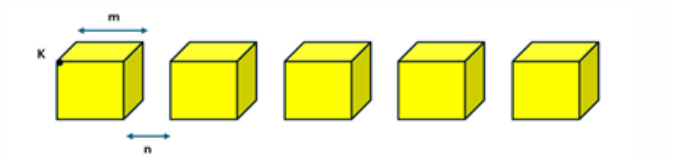
(v) Κίνηση A στον άξονα x, βήμα a/2 με L/J (press-only)

```
// (v) A moves by a/2 on X with J/L (edge-trigger)
float AoffsetX = 0.0f; const float step = A_len * 0.5f;     // 1.5
bool pL = false, pJ = false, pW = false, pX = false, pQ = false, pZ = false, pPLUS = false, pMINUS = false;

while (!glfwWindowShouldClose(window))
{
    // quit with '1'
    if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS) glfwSetWindowShouldClose(window, GLFW_TRUE);

    // move J/L one step per press
    bool L = glfwGetKey(window, GLFW_KEY_L) == GLFW_PRESS;
    bool J = glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS;
    if (L && !pL) AoffsetX += step;
    if (J && !pJ) AoffsetX -= step;
    pL = L; pJ = J;
```

Η εφαρμογή στο Model γίνεται με:

```
camera_function(); //update View matrix and Projection matrix
glm::mat4 ModelA = glm::translate(glm::mat4(1.0f), glm::vec3(AoffsetX, 0, 0));
glm::mat4 MVP_A = Projection * View * ModelA;
```

(vi) Σειρά από 5 κύβους επάνω από τον A – K = (−9, 10, 0), m=2, n=2



Εικόνα 3 - Η τοποθέτηση των κύβων

- Δημιουργία γεωμετρίας κύβου: build_cube(...) με 12 τρίγωνα ανά κύβο.
- Τοποθέτηση 5 κύβων:

```
//(vi) 5 cubes above A
vector<glm::vec3> cubes;
const float m = 2.0f, n = 2.0f, stride = m + n;                    // m=2, n=2
glm::vec3 K0(-9.0f, 10.0f, 0.0f);                                 // first cube K
for (int i = 0; i < 5; i++) build_cube(cubes, K0 + glm::vec3(stride * i, 0, 0), m);
```

Το παραπάνω, φτιάχνει έναν πίνακα `cubes` με όλες τις κορυφές και τρίγωνα των 5 κύβων.

- Ο πρώτος κύβος ξεκινά στο σημείο `K = (-9.0, 10.0, 0.0)` (δηλαδή πάνω και αριστερά από τον A).
- Κάθε επόμενος τοποθετείται **2 μονάδες μακριά + 2 μονάδες μέγεθος = 4 μονάδες κενό** (`stride`).

Άρα οι θέσεις τους (πάνω στον άξονα x) είναι:

(-9, 10, 0), (-5, 10, 0), (-1, 10, 0), (3, 10, 0), (7, 10, 0)

Ο σχεδιασμός των κύβων γίνεται παρακάτω. Συνδυάζεται ο πίνακας cubes με την πανοραμική κάμερα (Projection * View). Χρησιμοποιεί το View που καθορίζεται λίγο πιο πάνω.

```
// draw cubes (single solid color via constant attrib)
glm::mat4 MVP_I = Projection * View * glm::mat4(1.0f);
glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP_I[0][0]);

glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vboCubes);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
glDisableVertexAttribArray(1);
glVertexAttrib4f(1, 1.0f, 1.0f, 0.0f, 1.0f);


glDrawArrays(GL_TRIANGLES, 0, (GLsizei)cubes.size());
glDisableVertexAttribArray(0);
```

Χρώμα κύβων: έστω σταθερό κίτρινο:

```
glDisableVertexAttribArray(1);
glVertexAttrib4f(1, 1.0f, 1.0f, 0.0f, 1.0f);
```

Παρακάτω φαίνεται η κάμερα να βλέπει προς το κέντρο (0,0,0) όπου βρίσκεται ο Α. Κάθε κύβος προβάλλεται έτσι ώστε να φαίνεται από πάνω και ελαφρώς λοξά, δίνοντας το αποτέλεσμα της πανοραμικής λήψης. Μια κάμερα που βλέπει τη σκηνή από μακριά.

```
glm::vec3 eye = camPos();
glm::mat4 View = glm::lookAt(eye, glm::vec3(0, 0, 0), glm::vec3(0, 1, 0));
```

(vii) Κάμερα με πληκτρολόγιο (press-only): W/X (άξονας x), Q/Z (άξονας y), Numpad +/− (zoom)

Η συνάρτηση camera_function() λειτουργεί ως εξής:

- αρχικά η κάμερα είναι πρακτικά στο (0,-5,20) κοιτώντας το (0,0,0) με up (0,1,0),
- W/X περιστρέφει γύρω από x, Q/Z γύρω από y,
- Numpad + / − κάνει zoom χωρίς να αλλάζει το FOV (μένει 60°).

Οι παράμετροι/βήματα που χρειάστηκαν για την υλοποίηση αυτού του ερωτήματος είναι:

float camYaw=0.0f, camPitch=0.0f, camR=20.0f; // FOV σταθερό

const float dAng = glm::radians(5.0f);

const float dR   = 1.0f;

bool pW=false,pX=false,pQ=false,pZ=false,pPLUS=false,pMINUS=false;

Η ενημέρωση press-only γίνεται ως:

```
if (W && !pW) camPitch += dAng;  if (Xk && !pX) camPitch -= dAng;
if (Q && !pQ) camYaw += dAng;  if (Z && !pZ) camYaw -= dAng;
if (KP_PLUS && !pPLUS)  camR = std::max(2.0f, camR - dR);      // zoom in
if (KP_MINUS && !pMINUS) camR = std::min(80.0f, camR + dR);    // zoom out
```

Υπολογισμός eye & View (ή μέσω camera_function()):

```
glm::vec3 eye = camPos();
glm::mat4 View = glm::lookAt(eye, glm::vec3(0, 0, 0), glm::vec3(0, 1, 0));
```

2. Πληροφορίες σχετικά με την υλοποίηση

Η εφαρμογή γράφτηκε σε C++ και χρησιμοποιήσαμε Windows 10 (64-bit). Η εφαρμογή τρέχει ως εκτελέσιμο console application σε περιβάλλον Windows, με χρήση GLFW, GLEW και OpenGL. Η ανάπτυξη του κώδικα έγινε σε Visual Studio. Ενεργοποιήσαμε το σωστό working directory ώστε τα shader αρχεία ProjectVertexShader.vertexshader ProjectFragmentShader.fragmentshader να είναι προσβάσιμα στο runtime από τον φάκελο που τρέχει το .cpp. Σε περιβάλλον Windows, προτείνεται # define NOMINMAX πριν από τυχόν #include <windows.h>.

3. Αξιολόγηση

Χωρίσαμε την εργασία σε διακριτά μέρη:

<u>ΑΜ 5233:</u>

Υλοποίησε τη γεωμετρία του χαρακτήρα Α, με αναλυτικό υπολογισμό όλων των κορυφών v1–v9 και δημιουργία των αντίστοιχων τριγώνων της βάσης και της πυραμίδας. Δημιούργησε τη συνάρτηση build_A() με την οποία παράγονται οι πίνακες κορυφών (Averts) και χρωμάτων (Acols), καθώς και τις βοηθητικές συναρτήσεις tri() και tint(). Ανέλαβε τον χρωματισμό των πλευρών του χαρακτήρα Α. Ανέπτυξε τη συνάρτηση build_cube() για τη δημιουργία των κύβων της σκηνής και φρόντισε για την τοποθέτησή τους στη σωστή διάταξη (αρχική κορυφή K = (−9,10,0), m = 2, n = 2).

<u>ΑΜ 5386:</u>

Δημιούργησε την αρχικοποίηση του παραθύρου, του GLFW/GLEW περιβάλλοντος και τη διαμόρφωση του OpenGL context. Διαμόρφωσε το παράθυρο σύμφωνα με την εκφώνηση. Ανέπτυξε τη λογική του τερματισμού με πλήκτρο “1” και του κινητού χαρακτήρα Α πάνω στον άξονα x με πλήκτρα L/J, εισάγοντας edge-triggered λογική για ομαλή μετακίνηση. Υλοποίησε τη συνάρτηση camera_function(), η οποία διαχειρίζεται την περιστροφή και το zoom της κάμερας με τα πλήκτρα W, X, Q, Z, +, −, υπολογίζοντας σε κάθε καρέ τη νέα θέση του παρατηρητή σε σφαιρικές συντεταγμένες (yaw, pitch, radius). Ενσωμάτωσε τον έλεγχο πληκτρολογίου (keyboard handling) με glfwGetKey() ώ στε να καλύπτονται όλες οι απαιτούμενες κινήσεις. Εξασφάλισε ότι το FOV παραμένει σταθερό στα 60°, ενώ το zoom γίνεται μέσω αλλαγής της ακτίνας παρατήρησης camR.

<u>Δυσκολίες:</u>

1) Χειρισμός press-only input και σταθερό FOV
2) Η κατανόηση περιστροφής της κάμερας

Η συνεργασία χαρακτηρίστηκε από καλή επικοινωνία, διαχωρισμό αρμοδιοτήτων και συνεχείς δοκιμές. Συνεργατήκαμε τόσο εξ αποστάσεως (μέσω Git) όσο και δια ζώσης για δοκιμές και οπτική επαλήθευση του αποτελέσματος.

4. Αναφορές – Πηγές που χρησιμοποιήθηκαν κατά την εκπόνηση της εργασίας.

- OpenGL Tutorials (Matrices, MVP, VBO/VAO, Shaders) - https://learnopengl.com/Advanced-OpenGL/Geometry-Shader
- Για glm::perspective, glm::lookAt - https://learnopengl.com/Getting-started/Camera
- Για input handling και key codes - https://www.glfw.org/docs/3.0/group__input.html
- Youtube tutorials για κατανόηση της έννοιας “κάμερας” -

  https://www.youtube.com/watch?v=lsOoo3Q4_ag

  https://www.youtube.com/watch?v=U0_ONQQ5ZNM
- Σημειώσεις και διαφάνειες του μαθήματος