Πρώτο Μέρος (1Α) του Συνόλου Προγραμματιστικών Ασκήσεων

Γραφικά Υπολογιστών και Συστήματα Αλληλεπίδρασης

Ελευθέριος Ιωσηφίδης, 5233

Δανάη Χανλαρίδου, 5386

28/10/2025

1. Περιγραφή της εργασίας

Σκοπός της εργασίας 1Α ήταν η υλοποίηση μιας απλής εφαρμογής OpenGL που:

1. Ανοίγει ένα παράθυρο 900x900 με συγκεκριμένο τίτλο στα ελληνικά, έχει σκούρο γκρι background, και κλείνει με το πλήκτρο 1.

2. Σχεδιάζει το γράμμα «Α» χρησιμοποιώντας μόνο τρίγωνα, με διαστάσεις που δίνονται από την εκφώνηση (πλάτος a, ύψος b, τρίγωνο κορυφής ύψους c).

3. Επιτρέπει στο γράμμα Α να μετακινείται οριζόντια μέσα στο παράθυρο με τα πλήκτρα J (αριστερά) και L (δεξιά), σε βήματα σταθερά (a/2), χωρίς να βγαίνει εκτός ορατής σκηνής.

4. Δημιουργεί και σχεδιάζει ένα «αστέρι» πάνω από το γράμμα Α. Το αστέρι εμφανίζεται σε τυχαία θέση εντός συγκεκριμένου range, μένει στην οθόνη για λίγο και μετά «ξαναγεννιέται» αλλού.

(i) Για το ερώτημα αυτό αλλάξαμε αρχικά την εντολή

```
window = glfwCreateWindow(750, 750, "Project 1A template", NULL, NULL);
```

και την κανάμε:

```
window = glfwCreateWindow(900, 900, u8"\u0388\u03C1\u03B3\u03B1\u03C3\
        u03AF\u03B1 1\u0391 \u2013 5233 \u2013 5386", NULL, NULL);
```

Τα πρώτα δύο ορίσματα 900, 900 ρυθμίζουν το μέγεθος του παραθύρου σε pixels. Το τρίτο όρισμα είναι ο τίτλος του παραθύρου. Χρησιμοποιήσαμε UTF-8 escape sequences (\u0388 κλπ.) για να περάσουν σωστά ελληνικοί χαρακτήρες και παύλα τύπου en-dash (–). Αυτό λύθηκε γιατί αν βάζαμε απευθείας ελληνικά, το GLFW σε μερικά build settings τα εμφάνιζε ως "????".

Έπειτα, ρυθμίζουμε το χρώμα του background από:

```
glClearColor(0.0f, 0.0f, 1.02f, 0.0f); // μπλε φόντο
```

σε → `glClearColor(0.5f, 0.5f, 0.5f, 1.0f);  // γκρι φόντο`

Η glClearColor(r,g,b,a) ορίζει το χρώμα καθαρισμού του color buffer. (0.5,0.5,0.5) = μεσαίο γκρι.

Η παρακάτω εντολή μας λέει πως το loop έβγαινε μόνο με Q

```
while (glfwGetKey(window, GLFW_KEY_Q) != GLFW_PRESS && glfwWindowShouldClose(window) == 0);
```

Εμείς βάλαμε τις εξής εντολές μέσα στα loop, ώστε αν Αν ο χρήστης πατήσει 1, τότε καλούμε glfwSetWindowShouldClose(window, GLFW_TRUE). Το glfwWindowShouldClose(window) στη συνθήκη του while θα γίνει true και το loop θα τερματίσει.

```
    glfwPollEvents();
    // Check if the "1" key is pressed to close the app
    if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, GLFW_TRUE);
}
while (glfwGetKey(window, GLFW_KEY_Q) != GLFW_PRESS &&  glfwWindowShouldClose(window) == 0);
```

Με αυτό ικανοποιήθηκε η απαίτηση "Με το πλήκτρο 1 η εφαρμογή τερματίζει".

(ii) Η σχεδιαστική σκέψη που ακολουθήσαμε σε αυτό το ερώτημα είναι:

- Θεωρούμε ότι το γράμμα Α ακουμπά “χαμηλά” στο παράθυρο, γύρω από y ≈ -10, ώστε να είναι ορατό από την κάμερα.

- Πλάτος παραλληλογράμμου = a = 3 ⇒ μισό πλάτος = 1.5.
  Άρα οι οριζόντιες άκρες είναι x = -1.5 και x = +1.5.

- Ύψος παραλληλογράμμου = b = 1.5.
  Αν πάρουμε κάτω μέρος στο y = -10.0, τότε πάνω μέρος = -10.0 + 1.5 = -8.5.

- Τρίγωνο κορυφής:

  - Βάση = η πάνω πλευρά του παραλληλογράμμου, από (-1.5, -8.5) έως (1.5, -8.5)

  - Ύψος = c = 0.75 ⇒ κορυφή στο y = -8.5 + 0.75 = -7.75

  - Η κορυφή είναι στο κέντρο x=0.

Άρα, τελικά σημεία:

- Κάτω παραλληλόγραμμο:
  BL = (-1.5, -10.0), BR = (1.5, -10.0), TL = (-1.5, -8.5), TR = (1.5, -8.5)

- Πάνω τρίγωνο:
  Left = (-1.5, -8.5), Right = (1.5, -8.5), Top = (0.0, -7.75)

Το ‘σπάμε’ σε τρίγωνα (αφού η OpenGL Core θέλει τρίγωνα μόνο):

- Παραλληλόγραμμο → 2 τρίγωνα

  1. (-1.5, -10.0), (1.5, -10.0), (1.5, -8.5)

  2. (-1.5, -10.0), (1.5, -8.5), (-1.5, -8.5)

- Τρίγωνο κορυφής → 1 τρίγωνο
  3. (-1.5, -8.5), (1.5, -8.5), (0.0, -7.75)

Το επόμενο screenshot δείχνει πως ο πίνακας shape[] αντικαθιστά τον αρχικό shape[] του κώδικα που δίνεται που απλά είχε δύο τυχαία τρίγωνα.

```
//example shape
GLfloat shape[] = {
    //first triangle
    -1.5f, -10.0f, 0.0f,
     1.5f, -10.0f, 0.0f,
     1.5f, -8.5f, 0.0f,

     //second triangle
    -1.5f, -10.0f, 0.0f,
     1.5f, -8.5f, 0.0f,
    -1.5f, -8.5f, 0.0f,

    //third triangle
    -1.5f, -8.5f, 0.0f,
     1.5f, -8.5f, 0.0f,
     0.0f, -7.75f, 0.0f

};
```

Παρακάτω φαίνεται το Vertex buffer object (VBO) για το γράμμα Α. Χρησιμοποιούμε GL_DYNAMIC_DRAW γιατί τα vertices αυτά θα τα αλλάζουμε αργότερα οριζόντια (για την κίνηση).

```
GLuint vbuffer;
glGenBuffers(1, &vbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(shape), shape, GL_DYNAMIC_DRAW);
/*****************************************************************/
```

Η σχεδίαση μέσα στο render loop γίνεται ως εξής:

```
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vbuffer);

glVertexAttribPointer(
    0,                  // attribute 0, must match the layout in the shader.
    3,                  // size
    GL_FLOAT,           // type
    GL_FALSE,           // normalized?
    0,                  // stride
    (void*)0            // array buffer offset
);

// Draw the triangle !
glDrawArrays(GL_TRIANGLES, 0, 3*3); // 3 vertices for each triangle -> 9 ver
glDisableVertexAttribArray(0);
```

(iii) Η σκέψη που ακολουθήσαμε σε αυτό το ερώτημα είναι:

- Κρατάμε μια μεταβλητή moveX που είναι η οριζόντια μετατόπιση του γράμματος A.
- Σε κάθε πάτημα του πλήκτρου L ή J αλλάζουμε το moveX κατά +1.5 ή -1.5.
- Αντί να αλλάζουμε κάποιο uniform matrix, παίρνουμε τις αρχικές κορυφές στο shape[] και φτιάχνουμε ένα νέο προσωρινό array movedShape[] όπου σε κάθε κορυφή προσθέτουμε το moveX μόνο στη συντεταγμένη x.
- Μετά γράφουμε ξανά αυτά τα δεδομένα στο VBO με glBufferSubData, άρα το αντικείμενο "πηγαίνει" δεξιά/αριστερά όπως ζητάει η εκφώνηση (CPU-side update των vertices).

Στο κώδικα, υλοποιούμε όλες τις απαραίτητες μεταβλητές:

```
////////////////////////////////
float moveX = 0.0f;       // current horizontal offset
const float step = 1.5f;        // movement step = a/2
const float limit = 11.0f - 1.5f; // window limit minus half the A width

// Edge-trigger flags (one step per press)
static bool prevL = false, prevJ = false;
////////////////////////////////
```

και μετά διαβάζουμε το input στο loop:

```
////////////////////////////////////////////////////////////////
// --- Κίνηση: 1 βήμα ανά πάτημα ---
bool L = glfwGetKey(window, GLFW_KEY_L) == GLFW_PRESS;
bool J = glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS;

bool moved = false;
if (L && !prevL) { moveX += step; moved = true; }
if (J && !prevJ) { moveX -= step; moved = true; }
```

με prevL / prevJ σημαίνει ότι όταν πατήσουμε L και μετά το αφήσουμε, μετράει μία φορά. Δεν κάνει "συνεχή" κίνηση αν κρατάς το κουμπί πατημένο. Έπειτα, θέτουμε τα όρια:

```
// clamp για να μη βγει εκτός
if (moved) {
    if (moveX > limit) moveX = limit;
    if (moveX < -limit) moveX = -limit;
```

Το limit εδώ είναι 9.5 με βάση το 11.0 – 1.5. Η ιδέα είναι ότι δεν αφήνουμε την αριστερή άκρη του γράμματος A (x=-1.5+moveX) ή τη δεξιά άκρη (x=+1.5+moveX) να φύγει έξω από το ορατό εύρος [-11, 11]. Μετά, όταν υπάρχει κίνηση, ξανα-υπολογίζουμε τα νέα (μετατοπισμένα) vertices και τα ξαναστέλνουμε στο GPU:

```
// γράψε τις μετατοπισμένες κορυφές στο VBO
GLfloat movedShape[sizeof(shape) / sizeof(GLfloat)];
for (int i = 0; i < (int)(sizeof(shape) / sizeof(GLfloat)); i += 3) {
    movedShape[i] = shape[i] + moveX; // x + offset
    movedShape[i + 1] = shape[i + 1];       // y
    movedShape[i + 2] = shape[i + 2];       // z
}
glBindBuffer(GL_ARRAY_BUFFER, vbuffer);
glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(movedShape), movedShape);
```

Δεν χρησιμοποιούμε μετασχηματισμούς. Αντί γι' αυτό ξαναγράφουμε τα vertices μέσα στο ίδιο VBO.

(iv) Η γεωμετρία αστεριού μπορεί να δοθεί ως: Αν το κέντρο του αστεριού είναι (X, Y):

- Το κεντρικό τετράγωνο έχει πλευρά m = 0.5 ⇒ μισή πλευρά h = m/2 = 0.25.
  Άρα κορυφές τετραγώνου:
  (X - 0.25, Y - 0.25)
  (X + 0.25, Y - 0.25)
  (X + 0.25, Y + 0.25)
  (X - 0.25, Y + 0.25)

  Αυτό το τετράγωνο σχεδιάζεται με δύο τρίγωνα.

- Ακτίνες:

  - Πάνω τρίγωνο:
    βάση = η πάνω πλευρά του τετραγώνου, δηλαδή από (X-0.25, Y+0.25) μέχρι (X+0.25, Y+0.25)
    apex = (X, Y+0.25+n) = (X, Y+1.75)

  - Κάτω τρίγωνο:
    βάση στο κάτω μέρος,
    apex = (X, Y-0.25-n) = (X, Y-1.75)

  - Δεξί τρίγωνο:
    βάση στη δεξιά πλευρά του τετραγώνου,
    apex = (X+0.25+n, Y) = (X+1.75, Y)

  - Αριστερό τρίγωνο:
    apex = (X-1.75, Y)

Στο κώδικα αυτό σχηματίζεται μέσα σε ένα μικρό helper-lambda makeStar. Φτιάχνουμε 18 κορυφές (6 τρίγωνα × 3 κορυφές το καθένα). Όλες αυτές αποθηκεύονται στο array star[].

```
// 6 triangles * 3 vertices * (x,y,z)
float star[18 * 3];

// inline "builder" for a star centered at (X,Y)
auto makeStar = [&](float X, float Y) {
    const float m = 0.5f, n = 1.5f, h = m * 0.5f;
    int k = 0;
    auto V = [&](float x, float y) { star[k++] = x; star[k++] = y; star[k++] = 

    // center square (2 tris)
    V(X - h, Y - h); V(X + h, Y - h); V(X + h, Y + h);
    V(X - h, Y - h); V(X + h, Y + h); V(X - h, Y + h);
    // spikes: up, down, right, left
    V(X - h, Y + h); V(X + h, Y + h); V(X, Y + h + n);
    V(X - h, Y - h); V(X + h, Y - h); V(X, Y - h - n);
    V(X + h, Y - h); V(X + h, Y + h); V(X + h + n, Y);
    V(X - h, Y - h); V(X - h, Y + h); V(X - h - n, Y);
    };
```

Φτιάχνουμε VBO για το αστέρι:

```
// ===== STAR  =====
GLuint vboStar;
glGenBuffers(1, &vboStar);
```

```
glBindBuffer(GL_ARRAY_BUFFER, vboStar);
glBufferData(GL_ARRAY_BUFFER, sizeof(star), star, GL_DYNAMIC_DRAW);
```

Την τυχαία θέση που θα έχει του αστεριού την υλοποιήσαμε παρακάτω:

```
std::srand((unsigned)std::time(nullptr));
auto rnd = [](float lo, float hi) { return lo + (hi - lo) * (std::rand() / (float)RAND_MAX); };

float sx = rnd(-11.0f, 11.0f);
float sy = rnd(5.0f, 11.0f);
makeStar(sx, sy);
```

Η εμφάνιση θα γίνεται κάθε 1s:

```
do {
    // ===== STAR  =====
    // respawn star every ~1 second
    double now = glfwGetTime();
    if (now - star_t0 > 1.0) {
        star_t0 = now;
        sx = rnd(-11.0f, 11.0f);
        sy = rnd(-5.0f, 11.0f);
        makeStar(sx, sy);
        glBindBuffer(GL_ARRAY_BUFFER, vboStar);
        glBufferData(GL_ARRAY_BUFFER, sizeof(star), star, GL_DYNAMIC_DRAW);
    }
```

Σχεδίαση του αστεριού στο loop:

```
    // draw STAR (18 vertices)
    glBindBuffer(GL_ARRAY_BUFFER, vboStar);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
    glDrawArrays(GL_TRIANGLES, 0, 18);
    glDisableVertexAttribArray(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.

3. Αξιολόγηση

Χωρίσαμε την εργασία σε διακριτά μέρη:

- ΑΜ 5233:
  Υλοποίησε το βασικό πλαίσιο του προγράμματος (αρχικοποίηση GLFW/GLEW, δημιουργία παραθύρου και shader program), καθώς και τον κώδικα για το γράμμα Α. Υπολόγισε τις συντεταγμένες όλων των κορυφών του γράμματος Α (παραλληλόγραμμο + τρίγωνο), τις μετέτρεψε σε τρίγωνα και τις πέρασε σε buffers για OpenGL.

- ΑΜ 5386:
  Υλοποίησε τον χειρισμό εισόδου από το πληκτρολόγιο, τα όρια κίνησης στον άξονα x, και τον μηχανισμό μετατόπισης. Επιπλέον, υλοποίησε τη λογική για τα "αστέρια": παραγωγή τυχαίας θέσης στο εύρος που δίνεται από την εκφώνηση, δημιουργία του σχήματος του αστεριού από 6 τρίγωνα και εναλλαγή θέσης ανά τακτά χρονικά διαστήματα.

<u>Δυσκολίες:</u>

1) Η βασική δυσκολία ήταν τόσο η κατανόηση της γεωμετρίας μόνο σε 2Δ ενώ η κάμερα είναι 3Δ. Όσο και ο υπολογισμός και ο περιορισμός της κίνησης του γράμματος Α χωρίς μετασχηματισμούς.

2) Η γεωμετρική κατασκευή του αστεριού αποκλειστικά με τρίγωνα.

3) Ο σωστός χειρισμός UTF-8 για τον ελληνικό τίτλο.

Γενικά, η συνεργασία λειτούργησε καλά: κάναμε συχνές μικρές δοκιμές (build-compile–run) μετά από κάθε αλλαγή και έτσι διορθώσαμε bugs σχετικά γρήγορα. Η κατανομή δουλειάς ήταν δίκαιη και οι δύο είχαμε εικόνα όλου του κώδικα στο τέλος, όχι μόνο "του κομματιού μας".

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

- GLFW documentation και examples (χρήση για δημιουργία παραθύρου, Events/ Keyboard Input) - https://www.glfw.org/docs/latest/input_guide.html

- GLEW documentation (χρήση για φόρτωση OpenGL extensions πριν από οποιαδήποτε OpenGL κλήση)

- OpenGL Samples - https://github.com/g-truc/ogl-samples

- Διαφάνειες/Σημειώσεις μαθήματος για αναπαράσταση 2Δ σχημάτων