



# 2η Εργαστηριακή Άσκηση

Μάθημα : Γραφικά Υπολογιστών και συστήματα αλληλεπίδρασης

Στοιχεία Ομάδας : 1.Σπηλιόπουλος Σπήλιος 4495

2.Γκόνι Σίμων 4342

Ημερομηνία : 17/11/2022

## 1.Περιγραφή της αναφοράς

- Αναλυτική περιγραφή υλοποίησης κάθε ερωτήματος της εκφώνησης:

**Ερώτημα 1 :**

Όπως και στην άσκηση 1Α ο κώδικας παραμένει ο ίδιος, παρακάτω βρίσκονται τα στιγμιότυπα που υποδεικνύουν την λύση του ερωτήματος (i):

```
// Open a window and create its OpenGL context
window = glfwCreateWindow(1000, 1000, u8"Εργασία 1B - Τραπεζοειδές Πρίσμα", NULL, NULL); // Set Window Title with any character context casting u8 format
```

```
// Dark Blue background
glClearColor(0.0f, 0.0f, 0.2f, 0.0f);
glEnable(GL_DEPTH_TEST);
```

```
} // Check if the Space key was pressed or the window was closed
while (glfwGetKey(window, GLFW_KEY_SPACE) != GLFW_PRESS &&
      glfwWindowShouldClose(window) == 0); // GLFW_KEY_SPACE FOR TERMINATING THE PROGRAM INSTEAD OF ESCAPE
```

## Ερώτημα 2 :

### Συλλογιστική Πορεία :

- Σύμφωνα με τα ζητούμενα της άσκησης έπρεπε να υπολογίσουμε τις συντεταγμένες του πρίσματος.
- Το πρίσμα αποτελείται από δύο βάσεις (Βάση 1, Βάση 2) :  
Για να καταφέρουμε να δημιουργήσουμε τις βάσεις χρειαζόμαστε δύο σχήματα :
  1. Ένα παραλληλόγραμμο (αποτελείται από δύο τρίγωνα)
  2. Δύο επιπλέον Τρίγωνα (εκατέρωθεν του παραλληλογράμμου)

Σύμφωνα με τα χαρακτηριστικά που ζητάει η άσκηση ( $a = 2$ ,  $b = 8$ ,  $c = 6$ )

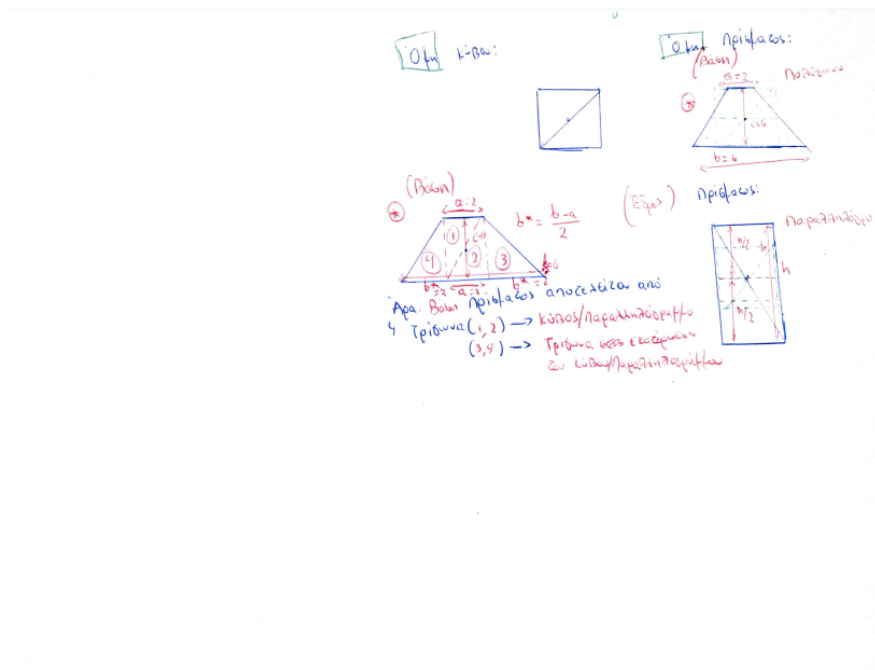
θα προκύψουν οι συγκεκριμένες συντεταγμένες που φαίνονται και στις παρακάτω εικόνες.

Στο σημείο που υπολογίζουμε τις συντεταγμένες, κάναμε λάθος από απροσεξία και υπολογίσαμε για  $b = 6$ . Έπειτα το διορθώσαμε και στο πρόγραμμα είναι υπολογισμένες για  $b = 8$ , όπως ζητάει η εκφώνηση. (Αντί για 3 και -3 στον άξονα X και X' αντίστοιχα, έχουμε 4 και -4).

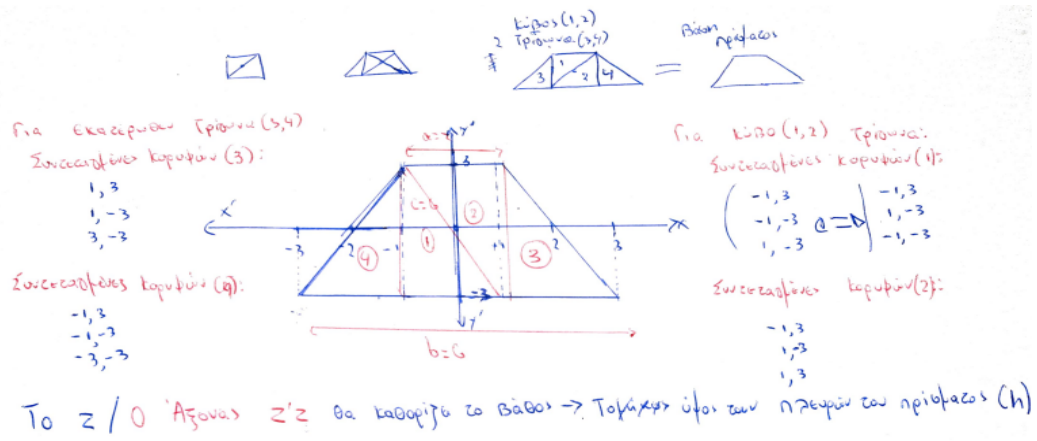
Αρχικά είχαμε ένα σταθερό  $h = 4$ , δηλαδή στον άξονα  $Z : z = + h/2 = 2$  και στον άξονα  $Z' : z' = - h/2 = -2$ . Αυτό για να ελέγξουμε το σχήμα μέχρι να υλοποιήσουμε τον κώδικα για το μεταβλητό  $h$ .

Έτσι θα έχουμε τις παρακάτω εικόνες :

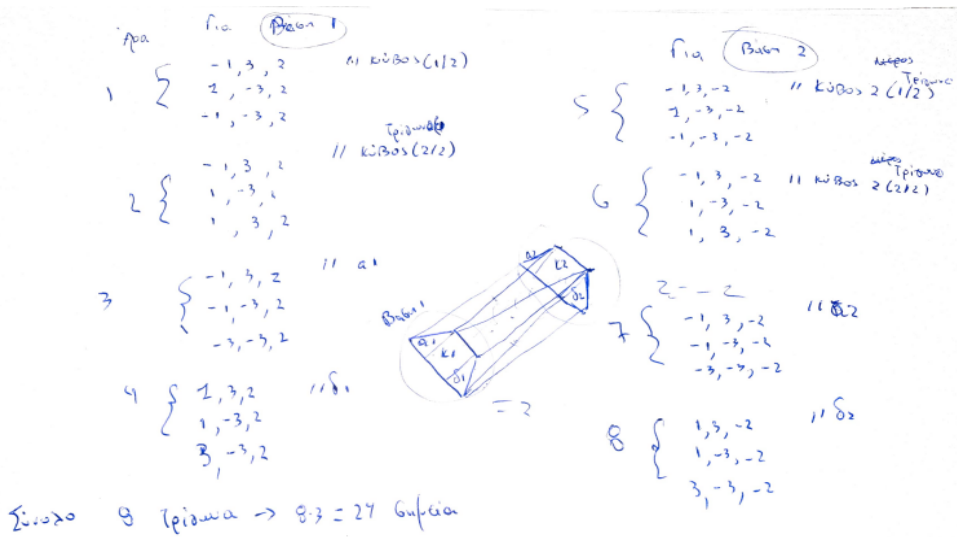
1.



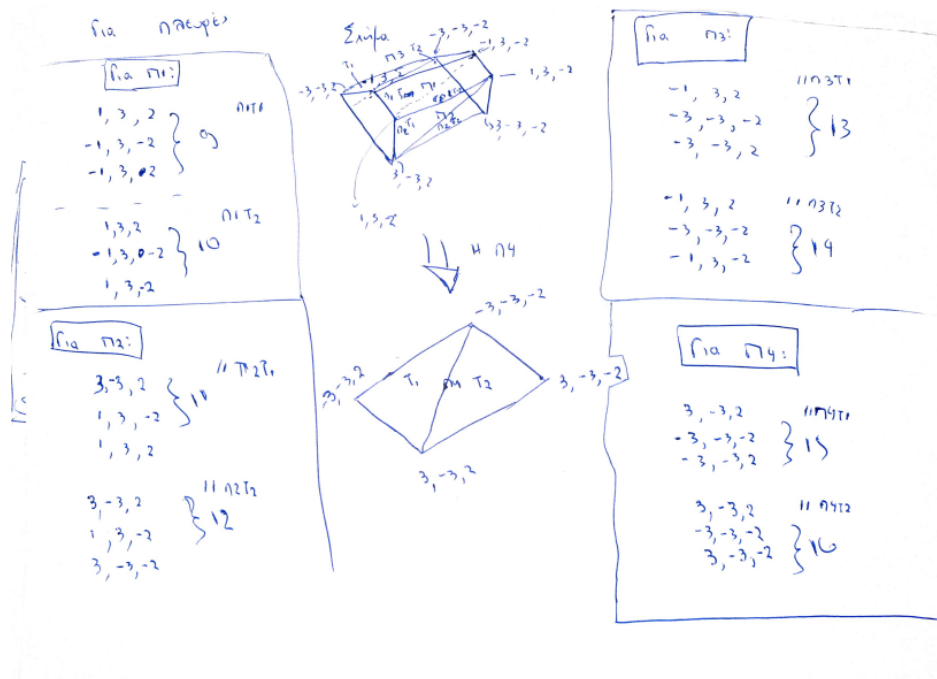
2.



3.



4.



- Μετά τον υπολογισμό των σημείων υλοποιήσαμε τη συνάρτηση για το μεταβλητό h.
- Τέλος, χρωμάτισαμε όλες τις πλευρές και τις βάσεις με διαφορετικό χρώμα

Οι γραμμές κώδικα που γίνονται όλες αυτές οι αλλαγές είναι οι εξής:

A. Συντεταγμένες Πρίσματος ( 457 - 550) :

```

457 static const GLfloat g_vertex_buffer_data[] = {
458
459
460 // For Front Base ( Base 1 ) ( z = h/2 )
461
462
463 -1.0f, 3.0f, h/2, // cube 1 Triangle : 1 / 2 (κ1) 1
464 1.0f, -3.0f, h/2,
465 -1.0f, -3.0f, h/2,
466
467 -1.0f, 3.0f, h/2, // cube 1 Triangle : 2 / 2 (κ1) 2
468 1.0f, 3.0f, h/2,
469 1.0f, -3.0f, h/2,
470
471
472 -1.0f, 3.0f, h/2, // Left Triangle (α1) 3
473 -1.0f, -3.0f, h/2,
474 -4.0f, -3.0f, h/2,
475
476 1.0f, 3.0f, h/2, // Right Triangle (δ1) 4
477 1.0f, -3.0f, h/2,
478 4.0f, -3.0f, h/2,
479
480
481 // For Back Base ( Base 2 ) ( z = - h/2 )
482
483
484 -1.0f, 3.0f, -h/2, // cube 2 Triangle : 1 / 2 (κ2) 5
485 1.0f, -3.0f, -h/2,
486 -1.0f, -3.0f, -h/2,
487

```

B. Μεταβλητό μήκος πρίσματος (h) (446 - 452) :

```

446 // For h variable //
447
448
449 float h;
450 srand(time(NULL)); // srand(time(NULL)) to initialize
451 h = (rand() % 10 + 2); // rand() % Max + Min, Max = 10, M
452 srand(0); // srand(0) in order to not save t
453

```

C. Χρωματισμός Κορυφών Τριγώνων (556- 649) :

```

556 // One color for each vertex.
557 static const GLfloat g_color_buffer_data[] = {
558
559
560 // For Base 1 ( z = h/2 ) // Red Color
561
562
563 0.5f, 0.0f, 0.0f, // cube 1 Triangle : 1 / 2 (κ1) 1
564 0.5f, 0.0f, 0.0f,
565 0.5f, 0.0f, 0.0f,
566
567 0.5f, 0.0f, 0.0f, // cube 1 Triangle : 2 / 2 (κ1) 2
568 0.5f, 0.0f, 0.0f,
569 0.5f, 0.0f, 0.0f,
570
571
572 0.5f, 0.0f, 0.0f, // Left Triangle (α1) 3

```

### Ερώτημα 3 :

Για το ερώτημα 3 το μόνο που χρειάστηκε να ρυθμίσουμε ήταν οι μεταβλητές της συνάρτησης `lookat()` και να τις θέσουμε σύμφωνα με αυτές που ζητούνται :

Το πρώτο όρισμα της `lookat()`, είναι η τοποθέτηση της κάμερας στο σημείο που ζητείται, το δεύτερο όρισμα είναι το σημείο το οποίο κοιτάζει η κάμερα και το τρίτο

είναι το ανιόν διάνυσμα. Παρακάτω φαίνονται οι γραμμές κώδικα μέσα από τα στιγμιότυπα.

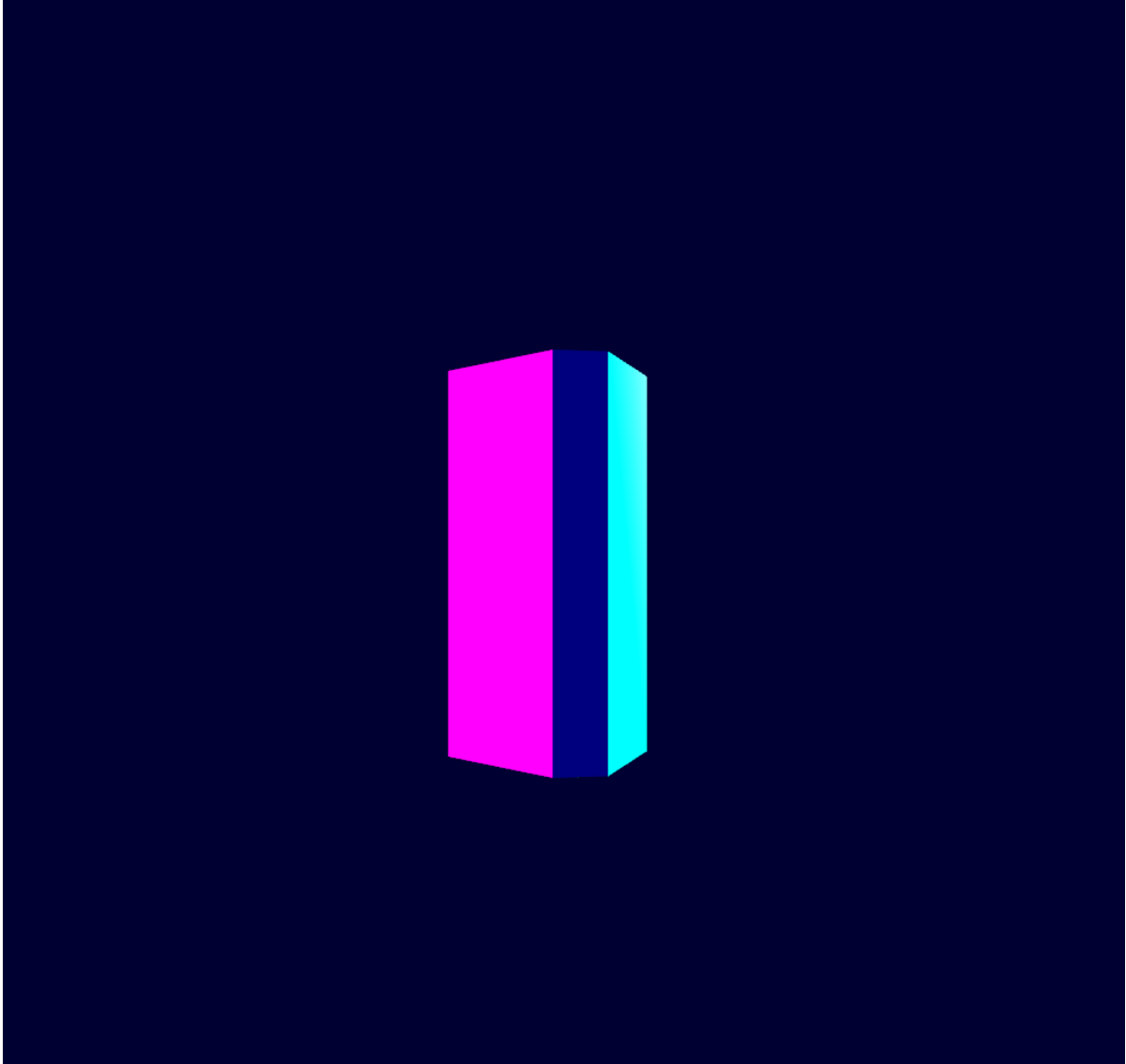
```
// Initial camera's Position (10, 50, 0)
glm::vec3 position = glm::vec3(cctvX, cctvY, cctvZ);

// Initial camera's Target : P(0, 0, 0)
glm::vec3 target = glm::vec3(0.0f, 0.0f, 0.0f);

// Up Vector
glm::vec3 up = glm::vec3(0.0f, 0.0f, 1.0f);
```

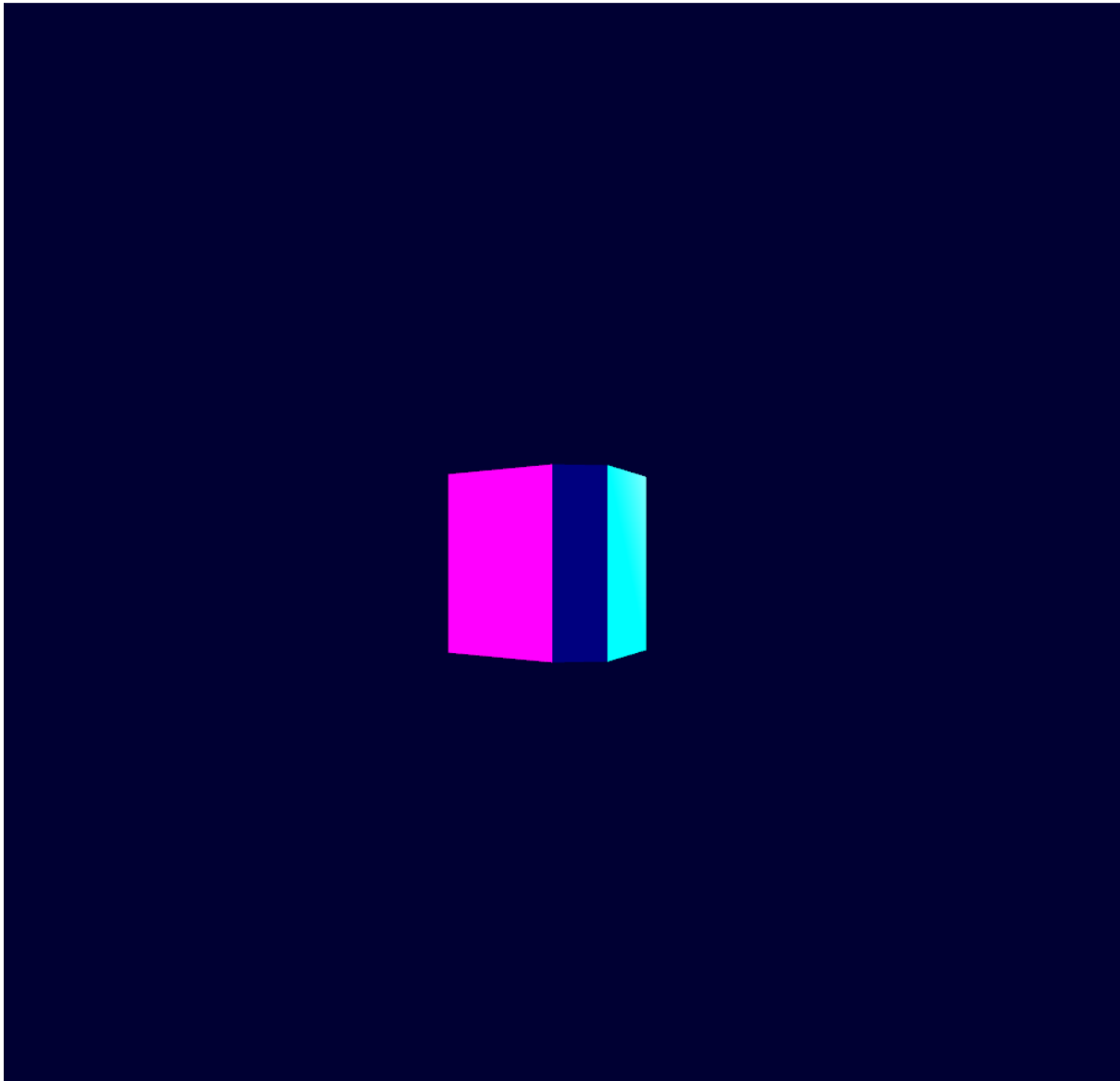
```
// Camera matrix
ViewMatrix = glm::lookAt(
    position,
    target,
    up
);
```

Αποτέλεσμα όλων των παραπάνω αλλαγών είναι το παρακάτω:



Και για διαφορετικό  $\theta$  έχουμε αντίστοιχα:



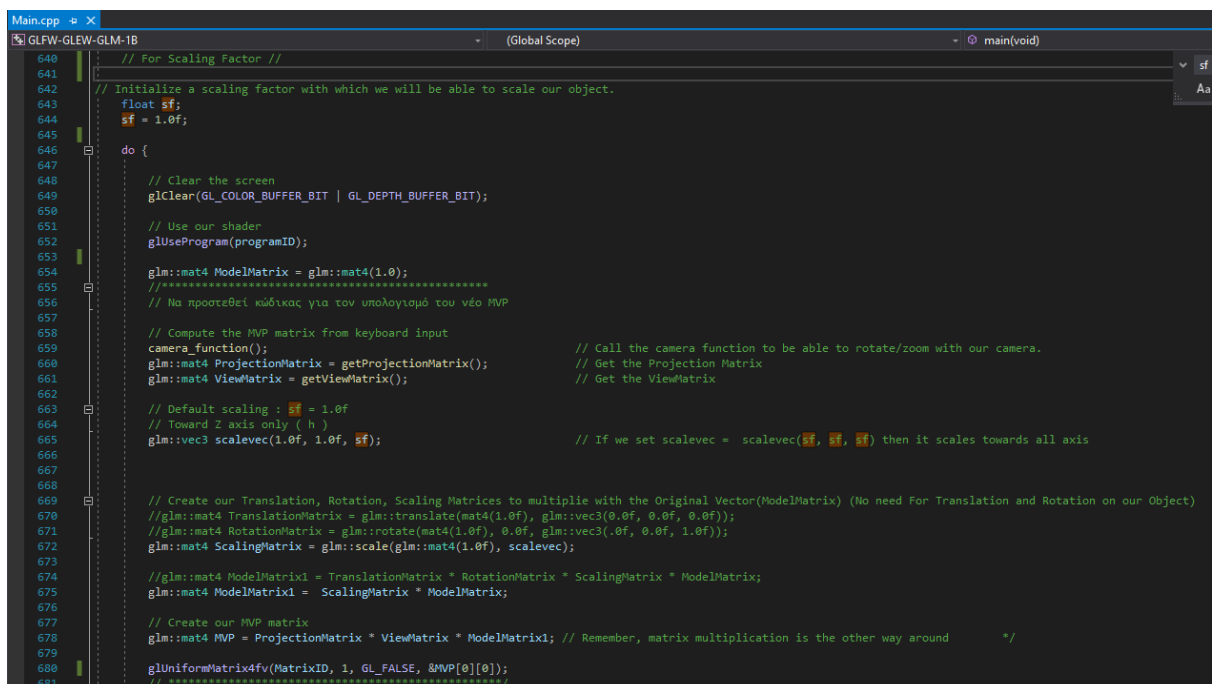


#### ***Ερώτημα 4 :***

- Αρχικά για το ύψος  $h$  προσδιορίσαμε μια τυχαία μεταβλητή όπως ζητείται από την άσκηση όπως προαναφέραμε και παραπάνω.
- Ύστερα προσδιορίσαμε μια μεταβλητή ώστε να μπορούμε να κάνουμε κλιμάκωση ή σμίκρυνση το αντικείμενο μας (scaling factor : sf) και την αρχικοποιούμε στο 1 έτσι ώστε το αντικείμενο μας αρχικά να έχει τις συντεταγμένες που ορίζουμε .
- Μέσα στην `camera_function()` υπάρχουν 3 πίνακες 4x4 (ProjectionMatrix & ViewMatrix καθώς και ο View τον οποίο θα χρειαστούμε στο επόμενο ερώτημα). Οι 2 πρώτοι πίνακες θα χρειαστούν για την δημιουργία του MVP πίνακα.

- Το διάνυσμα `scalevec` είναι υπεύθυνο για το `scale` μας και ορίζεται μέσα στην `do` Loop.
- Κάθε φορά που θέλουμε να κάνουμε `scale Up/Down` αυξομειώνουμε το `scaling factor`.
- Για να καταφέρουμε να κάνουμε `scale` ένα αντικείμενο πρέπει να φτιάξουμε τον `ScalingMatrix`. Οι πίνακες `TranslationMatrix` και `RotationMatrix` δεν χρησιμοποιούνται καθώς δεν θέλουμε να περιστρέψουμε ή να μετατοπίσουμε το πρίσμα μας.
- Με τον πολλαπλασιασμό του `ScalingMatrix` με τον `ModelMatrix` (ο μοναδιαίος πίνακας) δημιουργείται ένας καινούργιος πίνακας ονόματι `ModelMatrix1`, υπεύθυνος για `scale up/scale down`.
- Ο MVP πίνακας χρειάζεται απλά για τα `shaders` (ο οποίος προκύπτει με τον πολλαπλασιασμό των `ProjectionMatrix`, `ViewMatrix` και του `ModelMatrix1` τον οποίο μόλις ορίσαμε).
- Με το πλήκτρο <P> κάνουμε **scale down** και με το πλήκτρο <U> **scale up**, επίσης έχουμε θέσει κάποια όρια για το `scale` για να είναι ομοιόμορφο και να μην ξεφεύγει από το μέγεθος του παραθύρου.
- Καθώς ο χρήστης πατήσει ένα από τα δύο πλήκτρα, ο `scaling factor (sf)` ρυθμίζεται ανάλογα.

Εικόνες - Γραμμές Κώδικα ( 640 - 680, 718 - 736):



```

640 // For Scaling Factor //
641
642 // Initialize a scaling factor with which we will be able to scale our object.
643 float sf;
644 sf = 1.0f;
645
646 do {
647
648     // Clear the screen
649     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
650
651     // Use our shader
652     glUseProgram(programID);
653
654     glm::mat4 ModelMatrix = glm::mat4(1.0);
655     //*****
656     // No προστεθεί κώδικας για τον υπολογισμό του νέο MVP
657
658     // Compute the MVP matrix from keyboard input
659     camera_function(); // Call the camera function to be able to rotate/zoom with our camera.
660     glm::mat4 ProjectionMatrix = getProjectionMatrix(); // Get the Projection Matrix
661     glm::mat4 ViewMatrix = getViewMatrix(); // Get the ViewMatrix
662
663     // Default scaling : sf = 1.0f
664     // Toward Z axis only ( h )
665     glm::vec3 scalevec(1.0f, 1.0f, sf); // If we set scalevec = scalevec(sf, sf, sf) then it scales towards all axis
666
667
668
669     // Create our Translation, Rotation, Scaling Matrices to multiplie with the Original Vector(ModelMatrix) (No need For Translation and Rotation on our Object)
670     glm::mat4 TranslationMatrix = glm::translate(mat4(1.0f), glm::vec3(0.0f, 0.0f, 0.0f));
671     glm::mat4 RotationMatrix = glm::rotate(mat4(1.0f), 0.0f, glm::vec3(0.0f, 0.0f, 1.0f));
672     glm::mat4 ScalingMatrix = glm::scale(glm::mat4(1.0f), scalevec);
673
674     glm::mat4 ModelMatrix1 = TranslationMatrix * RotationMatrix * ScalingMatrix * ModelMatrix;
675     glm::mat4 ModelMatrix1 = ScalingMatrix * ModelMatrix;
676
677     // Create our MVP matrix
678     glm::mat4 MVP = ProjectionMatrix * ViewMatrix * ModelMatrix1; // Remember, matrix multiplication is the other way around */
679
680     glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
681     // *****

```

```

718 // If <U> is pressed then Scale Down the Prism to Z axis.
719 if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS) {
720
721
722     if (sf < 3.0f) { // Scale Up to a certain point (3.0f) so the object can be visible.
723         sf = sf + 0.01f; // Scale Up by adding a small float to original scaling factor.
724     }
725 }
726
727 // If <P> is pressed then Scale Up the Prism to Z axis.
728 else if (glfwGetKey(window, GLFW_KEY_P) == GLFW_PRESS) {
729
730     if (sf > 0.5f) { // Scale Down to a certain point ( 0.5f ) so the object can be visible.
731         sf = sf - 0.01f; // Scale Down by subtracting a small float to original scaling factor.
732     }
733 }
734
735 }
736

```

## Ερώτημα 5 :

### Συλλογιστική Πορεία :

- Αρχικά, για να καταφέρουμε να έχουμε κίνηση της κάμερας σύμφωνα με πλήκτρα πρέπει να δουλέψουμε στην συναρτηση της κάμερας ( camera\_function() ). Έπειτα, η ίδια συνάρτηση θα χρειαστεί να καλείται μέσα στην do loop έτσι ώστε να λειτουργεί για κάθε frame που δημιουργείται.
- Για την συνάρτηση της κάμερας:
  1. Θα χρειαστούμε τις διανυσματικές μεταβλητές position, target, up. Η μεταβλητή position θα αποθηκεύει τις νέες τιμές των συντεταγμένων X, Y και Z ανάλογα με τις ενέργειες που θέλουμε να εφαρμόσουμε(Zoom In/Out ). Η μεταβλητές target και up περιέχουν τα σημεία που θα κοιτάζει η κάμερα καθώς (target) και ως προς ποιόν άξονα θα είναι κάθετη η κάμερα ( up vector ). Οι δύο τελευταίες μεταβλητές δεν χρειάζεται να μεταβληθούν για τα συγκεκριμένα ερωτήματα ( βλέπε extra ).

2. Επίσης, θα χρειαστεί να αποθηκεύουμε κάθε φορά τις συντεταγμένες X,Y,Z στις μεταβλητές `cctvX`, `cctvY`, `cctvZ` αντίστοιχα (Zoom In/ Out).

3. Ακόμα, έχουμε τις μεταβλητές `angleX`, `angleY`, `angleZ` για να πραγματοποιούνται όλες τις περιστροφές (Rotation around all Axis)

4. Τέλος, έχουμε άλλη μια μεταβλητή, την `zoomFactor` την οποία θα χρησιμοποιήσουμε για τις λειτουργίες Zoom In/ Zoom Out.

- Για την περιστροφή γύρω από τους άξονες X και Z χρησιμοποιούμε πολλαπλασιασμό πινάκων όπως και για το scaling στο προηγούμενο ερώτημα. Κύρια διαφορά είναι ότι όλες οι αλλαγές εφαρμόζονται στον πίνακα View ο οποίος περιέχει την συνάρτηση `lookat()`. Αυτό σημαίνει ότι η περιστροφή που πραγματοποιείται γίνεται μόνο στην κάμερα. Αυτό μπορεί κανείς να το διαπιστώσει αν προσθέσει έναν επιπλέον πίνακα περιστροφής του αντικειμένου ή ένα παραπάνω αντικείμενο ή ένα background με texture.
- Το σκεπτικό είναι ότι αφού έχουμε τη δυνατότητα να εφαρμόσουμε τη συνάρτηση `rotate (glm::rotate)` στο αντικείμενο, τότε μπορούμε να το κάνουμε και στην κάμερα. Δεν χρειάζεται να εφαρμόσουμε πράξεις ημιτόνων και συνημιτόνων. Έτσι, ορίζουμε 3 νέους πίνακες (`RotationMatrixX`, `RotationMatrixY`, `RotationMatrixZ`, 1 πίνακα για κάθε άξονα).
- Όταν ο χρήστης πατήσει τα ανάλογα κουμπιά το μόνο που θα αλλάζει είναι η γωνία περιστροφής που αντιστοιχεί στο ανάλογο input( `angleX` για άξονα X, `angleY` για άξονα Y το οποίο δεν ζητείται, `angleZ` για άξονα Z).

- Τα angleX, angleY και angleZ αποτελούν ορίσματα των πινάκων RotateMatrixX, RotateMatrixY, RotateMatrixZ αντίστοιχα ( στην γωνία περιστροφής της συνάρτησης glm::rotate()).
- Σχετικά με την λειτουργία του Zoom In/Out έχουμε ορίσει την μεταβλητή zoomFactor η οποία θα αυξομειώνεται αντίστοιχα ανάλογα με τα inputs του χρήστη και έπειτα θα πολλαπλασιάζεται με τις 3 μεταβλητές του διανύσματος position ( cctvX, cctvY, cctvZ ).

Αφού η position είναι όρισμα στην lookat(), δηλαδή στον πίνακα View αρκεί να εφαρμόσουμε τους πολλαπλασιασμούς για περιστροφή σε έναν νέο πίνακα με όνομα ViewMatrix.

Εικόνες - Γραμμές Κώδικα ( 54 - 208 ):

```

54 // Initial Field of View
55 float initialFoV = 45.0f;
56
57
58
59 // For Camera Movement
60 // Initialize camera's Coordinates
61 float cctvX = 10.0f;
62 float cctvY = 50.0f;
63 float cctvZ = 0.0f;
64
65
66 // Initial camera's Position (10, 50, 0)
67 glm::vec3 position = glm::vec3(cctvX, cctvY, cctvZ);
68
69 // Initial camera's Target : P(0, 0, 0)
70 glm::vec3 target = glm::vec3(0.0f, 0.0f, 0.0f);
71
72 // Up Vector
73 glm::vec3 up = glm::vec3(0.0f, 0.0f, 1.0f);
74
75
76 // For Rotation
77
78 float angleZ = 0.0f;
79 float angleX = 0.0f;
80 float angleY = 0.0f;
81

```

```

84 void camera_function() {
85
86
87 ///////////////////////////////////////////////////
88
89 // For Zoom Factor (Multiplied with camera X, Y, Z coordinates)
90
91 float zoomFactor = 1.0f;
92
93
94
95
96 // X Rotation : when pressing W button
97 if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
98
99     angleZ = angleZ + 0.5f;
100 }
101
102 // X' Rotation : when pressing X button
103 if (glfwGetKey(window, GLFW_KEY_X) == GLFW_PRESS) {
104
105     angleZ = angleZ - 0.5f;
106 }
107
108
109
110
111 // Z Rotation : when pressing Q button
112 if (glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS) {
113
114     angleX = angleX + 0.5f;
115 }
116
117 // Z' Roation : when pressing Z button
118 if (glfwGetKey(window, GLFW_KEY_Z) == GLFW_PRESS) {
119
120     angleX = angleX - 0.5f;
121 }

```

```

124
125 // Extra code for rotation around Y axis, You can use Rotation around Y axis with Up Row and Down Row
126
127 // Y Rotation : when pressing Up row button
128 if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS) {
129
130     angleY = angleY + 0.5f;
131 }
132
133 // Y' Rotation : when pressing Down Row button
134 if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS) {
135
136     angleY = angleY - 0.5f;
137 }
138
139
140
141
142 // You can change initialFov in order to see better the inside of our object while Zooming In
143
144
145 // Zoom In
146 if (glfwGetKey(window, GLFW_KEY_KP_ADD) == GLFW_PRESS) {
147
148
149 // Slowly decrease the zoom factor and multiply it with our current X, Y, Z coordinates in order to move our camera closer to the
150 // The distance between our camera and the object becomes smaller.
151
152     zoomFactor = zoomFactor - 0.01f;
153
154     cctvX = cctvX * zoomFactor;
155     cctvY = cctvY * zoomFactor;
156     cctvZ = cctvZ * zoomFactor;
157
158     position = glm::vec3(cctvX, cctvY, cctvZ);
159 }
160

```

```

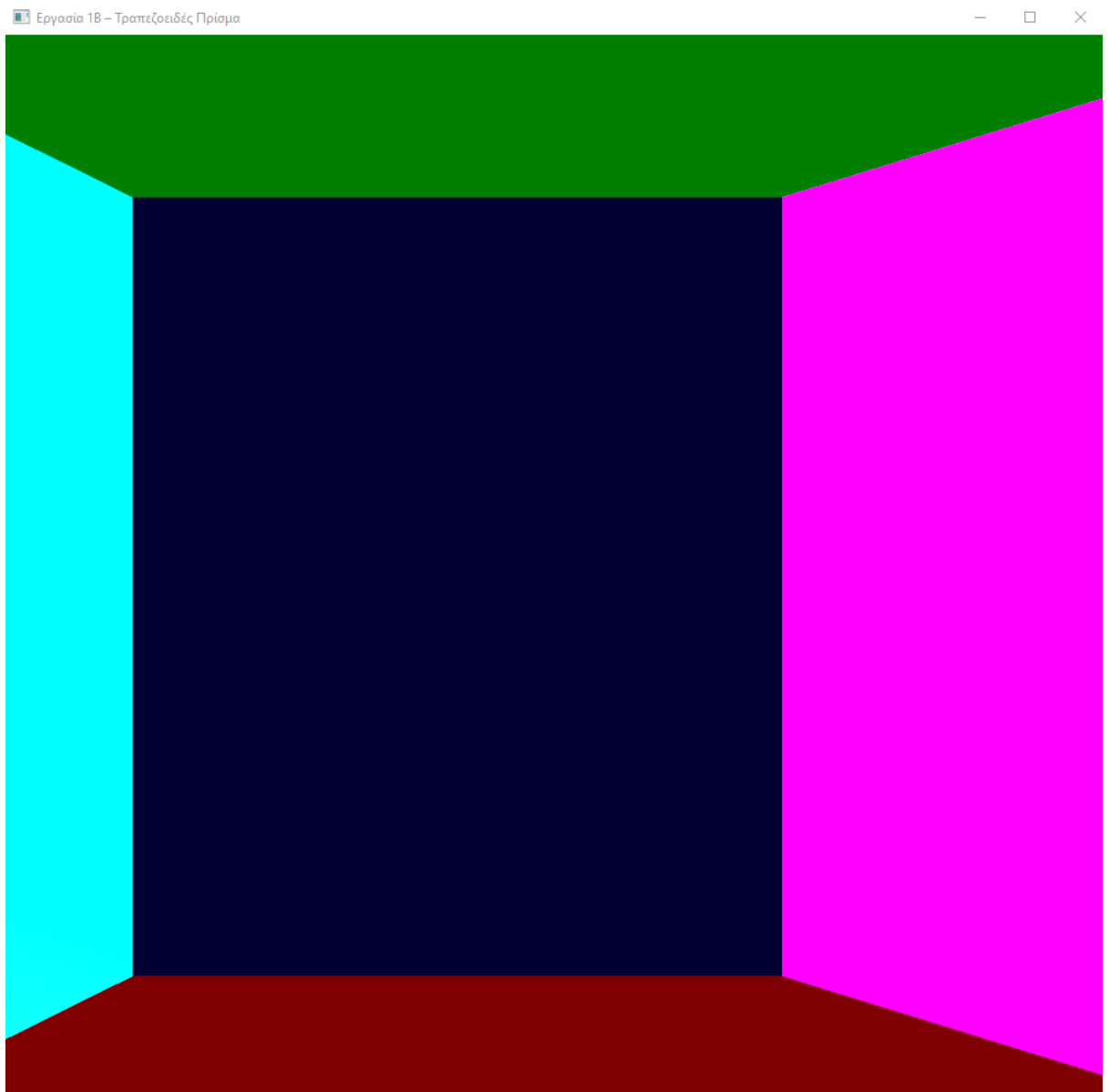
161 // Zoom Out
162 if (glfwGetKey(window, GLFW_KEY_KP_SUBTRACT) == GLFW_PRESS) {
163
164     // Slowly increase the zoom factor and multiply it with our current X, Y, Z coordinates in order to move our camera closer to the object.
165     // The distance between our camera and the object becomes larger
166
167     zoomFactor = zoomFactor + 0.01f;
168
169     cctvX = cctvX * zoomFactor;
170     cctvY = cctvY * zoomFactor;
171     cctvZ = cctvZ * zoomFactor;
172
173     position = glm::vec3(cctvX, cctvY, cctvZ);
174 }
175
176
177 float FoV = initialFoV; // - 5 * glfwGetMouseWheel(); // Now GLFW 3 requires setting up a callback for this. It's a bit too complicated for this beginn
178
179
180
181 glm::mat4 RotationMatrixX = glm::rotate(mat4(1.0f), glm::radians(angleX), glm::vec3(1.0f, 0.0f, 0.0f));
182 glm::mat4 RotationMatrixY = glm::rotate(mat4(1.0f), glm::radians(angleY), glm::vec3(0.0f, 1.0f, 0.0f));
183 glm::mat4 RotationMatrixZ = glm::rotate(mat4(1.0f), glm::radians(angleZ), glm::vec3(0.0f, 0.0f, 1.0f));
184 //glm::mat4 zoomMatrixCamera = glm::scale(mat4(1.0f), glm::vec3(zoomFactor, zoomFactor, zoomFactor));
185
186
187
188
189 // Projection matrix : 45° Field of View, 4:3 ratio, display range : 0.1 unit <-> 100 units
190 ProjectionMatrix = glm::perspective(glm::radians(FoV), 4.0f / 3.0f, 0.1f, 100.0f);
191
192
193
194
195 // Camera matrix
196 View = glm::lookAt(
197     position,
198     target,
199     up
200 );
201
202
203
204 //ViewMatrix = View * RotationMatrixZ * RotationMatrixY * RotationMatrixX * zoomMatrixCamera ; // We had an Error Scaling our Camera based on Depth Test...
205 // One side was not transparent(check README)
206 ViewMatrix = View * RotationMatrixZ * RotationMatrixY * RotationMatrixX;
207
208

```

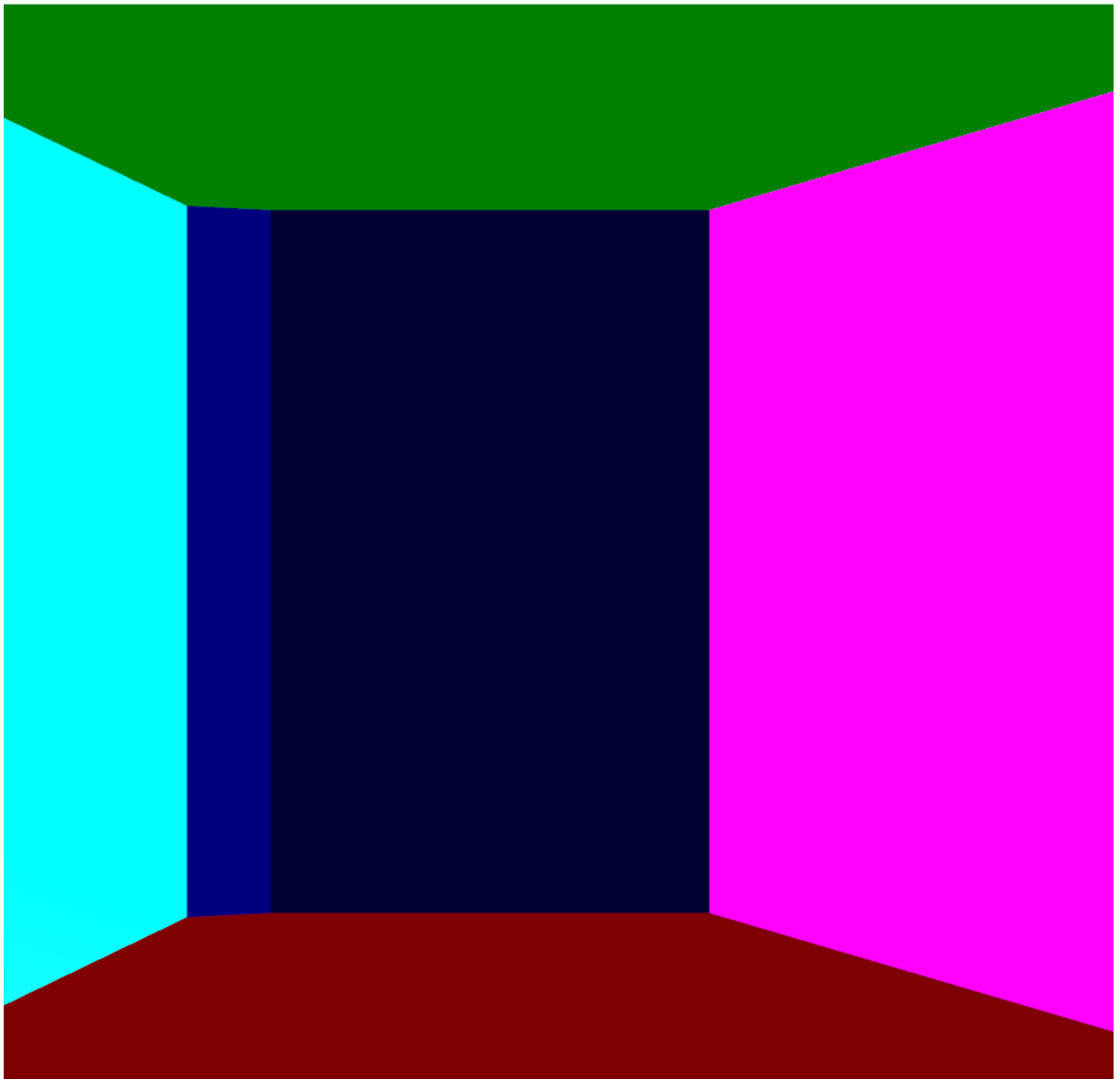
## Δυσκολίες:

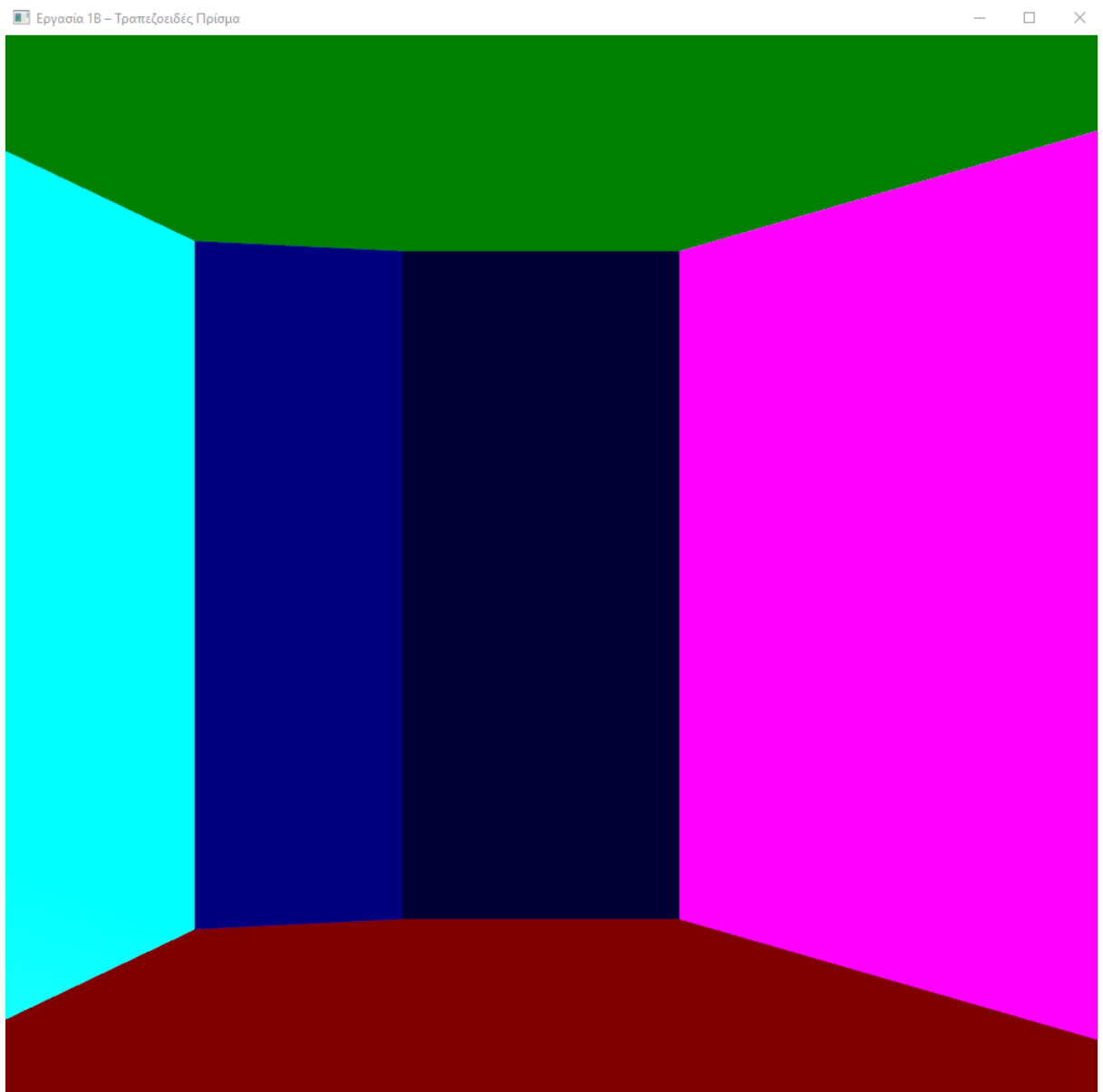
- Προσπαθήσαμε να κάνουμε το Zoom In/Out με την συνάρτηση (glm::scale) στην κάμερα. Αυτό, ενώ λειτουργεί, όταν η κάμερα μπαίνει μέσα στο αντικείμενο παρουσιάζει πρόβλημα και δεν εμφανίζει την απέναντι πλευρά του αντικειμένου ( παίρνει το χρώμα του background ). Απ' ότι καταλάβαμε, ίσως να υπάρχει σύγχυση με το DepthTest καθώς δεν ζωγραφίζεται αυτή η πλευρά επειδή η κάμερα δεν έχει οπτική επαφή μαζί της. Αν δεν ισχύει αυτό, τότε ο λόγος ίσως είναι ότι με αυτόν τον τρόπο η κάμερα δεν μπαίνει ποτέ πραγματικά στο αντικείμενο και η πλευρά η οποία κοιτάζει εξ αρχής είναι πάντα μπροστά από την οπτική μας γωνία. Παρακάτω παρουσιάζονται οι εικόνες με το σφάλμα που προκύπτουν.

## Εικόνες σφάλματος :









## 2.Πληροφορίες σχετικά με την υλοποίηση

Λειτουργικό Σύστημα : Windows 10

Περιβάλλον : Visual Studio 2019 x86

### 3. Σύντομη αξιολόγηση λειτουργίας ομάδας και της συνεργασίας

Δουλέψαμε από κοινού κατά την υλοποίηση της εργασίας. Δεν ανατέθηκαν ξεχωριστά κομμάτια ανά άτομο και η συνεργασία ήταν πολύ καλή.

### 4. Αναφορές-Πηγές

<https://learnopengl.com/Getting-started/Camera>

<http://glm.g-truc.net/0.9.4/api/a00151.html>

<http://www.c-jump.com/bcc/common/Talk3/Math/GLM>

[https://github.com/g-truc/glm/blob/master/manual.md#section3\\_8](https://github.com/g-truc/glm/blob/master/manual.md#section3_8)

[https://cs.brynmawr.edu/Courses/cs312/fall2010/lectures/gl\\_02.pdf](https://cs.brynmawr.edu/Courses/cs312/fall2010/lectures/gl_02.pdf)

<https://open.gl/drawing>

