Β ΜΕΡΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΩΝ ΑΣΚΗΣΕΩΝ

ΟΜΑΔΑ:
ΧΡΙΣΤΙΝΑ ΟΛΥΜΠΙΑ ΣΟΛΔΑΤΟΥ Α.Μ. 4001
ΙΩΑΝΝΑ ΑΠΟΣΤΟΛΟΠΟΥΛΟΥ Α.Μ. 2935

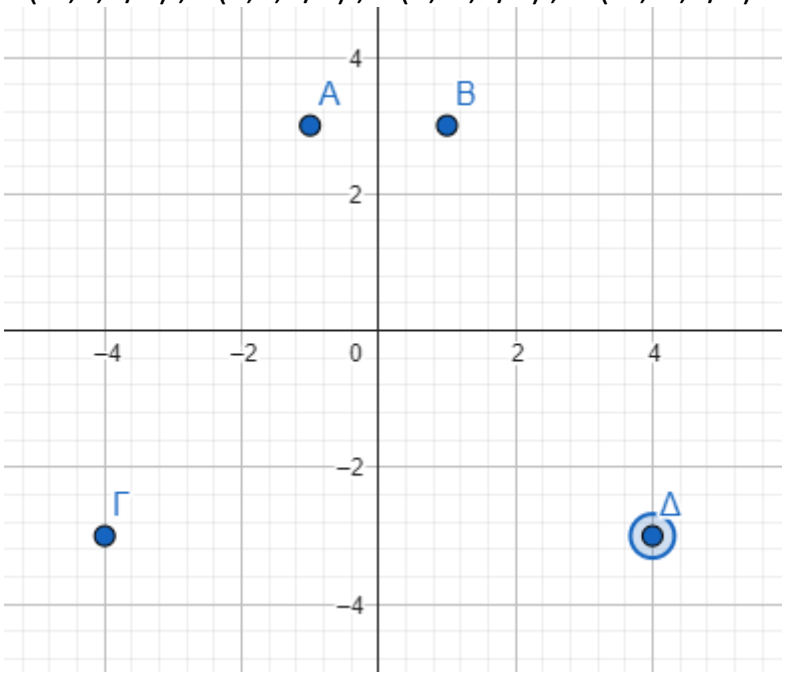
Ερώτημα i)

Τις διαστάσεις του παραθύρου αλλά και τον τίτλο τα ορίζουμε με την παρακάτω εντολή: window = glfwCreateWindow(1000, 1000, u8"Εργασία 1Β - Τραπεζοειδές Πρίσμα", NULL, NULL); .
Ο προσδιορισμός του χρώματος του background σε σκούρο μπλε γίνεται με την παρακάτω εντολή: glClearColor(0.0f, 0.0f, 0.2f, 0.0f); .
Για να τερματίζει η εφαρμογή με το space ελέχγετεται αν πατήθηκε το space:
while (glfwGetKey(window, GLFW_KEY_SPACE) != GLFW_PRESS && glfwWindowShouldClose(window) == 0); .

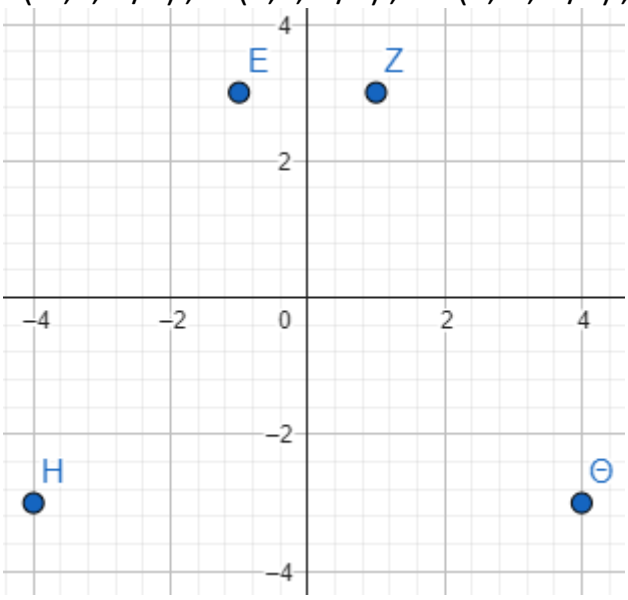
Ερώτημα ii)

Κάθε πλευρά του πρίσματος σχεδιάζεται από δύο τρίγωνα.
Οι κορυφές των τριγώνων είναι αποθηκευμένες στον πίνακα g_vertex_buffer_data[],κάθε τρεις γραμμές αναλογούν σε ένα τρίγωνο και κάθε 6 γραμμές αναλογούν σε κάθε πλευρά.

Η Βάση 1 βρίσκεται στο επίπεδο $z=h/2$ (επίπεδο παράλληλο στο επίπεδο xy) και δημιουργείται από τα τρίγωνα ΑΒΔ, ΔΓΑ .
Α(-1,3,h/2) , Β(1,3,h/2) , Γ(4,-3,h/2) , Δ(-4,-3,h/2)



Η Βάση 2 βρίσκεται στο επίπεδο z=-h/2 (επίπεδο παράλληλο στο επίπεδο xy) και δημιουργείται από τα τρίγωνα ΕΖΘ, ΘΗΕ .
Ε(-1,3,-h/2) , Ζ(1,3,-h/2) , Θ(4,-3,-h/2) , Η(-4,-3,-h/2)



Για τις άλλες τέσσερις πλευρές δηλαδή τα παραλληλόγραμμα :
Οι δύο πλευρές που είναι παράλληλες στον άξονα z σχεδιάζονται
Η πάνω (τα y>0) από τα τρίγωνα:ΕΖΒ , ΒΑΕ και η κάτω πλευρά (τα y<0) από τα τρίγωνα ΗΘΔ, ΔΓΗ.
Οι άλλες δύο πλευρές στα πλάγια, αυτή που έχει όλα τα x<0 από τα τρίγωνα : ΕΑΗ,ΗΑΓ
Και αυτή που έχει όλα τα x>0 από τα τρίγωνα ΖΒΘ,ΘΒΔ.

Για τον χρωματισμό των πλευρών δίνουμε το ίδιο χρώμα στις κορυφές των δύο τριγώνων που αποτελούν μία πλευρά και διαφορετικό από τις άλλες πλευρές. Ο πίνακας g_color_buffer_data[] έχει τις τιμές για το χρωματισμό κάθε κορυφής.

Ερώτημα iii)

Ορίσαμε ένα global διάνυσμα το camera_position=(10.0, 50.0, 0.0):

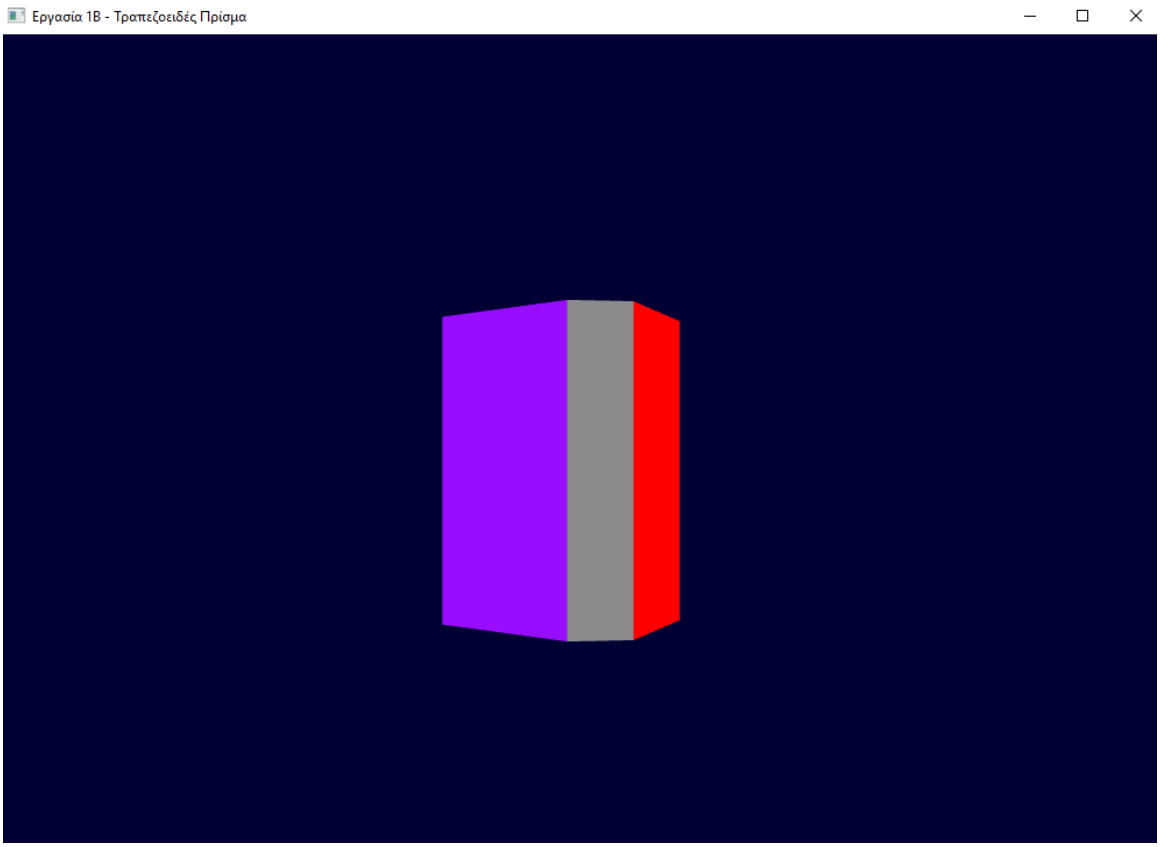
```
glm::vec3 camera_position = glm::vec3(10.0f, 50.0f, 0.0f);
```

Τοποθέτηση της κάμερας στο σημείο (10.0, 50.0, 0.0) ώστε να κοιτάει προς το σημείο P(0,0,0) του πρίσματος με ανιόν διάνυσμα (up vector) το (0.0, 0.0, 1.0) .

```
// Projection matrix : 30° Field of View, 4:3 ratio, display range : 0.1 unit <-> 100 units
glm::mat4 Projection = glm::perspective(glm::radians(30.0f), 4.0f / 3.0f, 0.1f, 100.0f);

//ARXIKOPOIHSH
// Camera matrix
View = glm::lookAt(
    camera_position,
    glm::vec3(0.0f, 0.0f, 0.0f),
    glm::vec3(0.0f, 0.0f, 1.0f)
);
// Model matrix
glm::mat4 Model = glm::mat4(1.0f);
// Our ModelViewProjection : multiplication of our 3 matrices
glm::mat4 MVP = Projection * View * Model; // Remember, matrix multiplication is the other way around
```

Screenshot της εφαρμογής μετά την τοποθέτηση της κάμερας στο (10.0, 50.0, 0.0):



Ερώτημα iv)

Για να μεγαλώνει/μικραίνει το ύψος $\boldsymbol{h}$ του πρίσματος, θα γίνεται κλιμάκωση ή σμίκρυνση του ύψους του πρίσματος . Με βάση τη θεωρία η κλμάκωση/σμίκρυνση στις 3Δ γίνεται πολλαπλασιάζωντας τις κορυφές του πρίσματός μας με έναν πίνακα 4*4 της παρακάτω μορφής.

$$\mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Στην περίπτωση μας θελουμε να αλλάζει μόνο η συνιστώσα z των κορυφών μας και να μεταβάλλεται από 1 εώς 5 στο θετικό z και από -5 ως -1 στο αρνητικό z , καθώς το $\boldsymbol{h}$ μεταβάλλεται στο διάστημα [2.0,10.0]. ΄Αρα θα είναι Sx=1 και Sy=1 .
Το Sz θα είναι από 1.0 εώς 5.0 . Στον πίνακα g_vertex_buffer_data[] που έχει τις κορυφές των τριγώνων από τα αποία σχηματίζεται κάθε πλευρά και τελικά το πρίσμα , βάλαμε τη z συνιστώσα 1 ή -1 ώστε μετά όταν πολλαπλασιαστεί με το Sz που αυξομειώνεται με βήμα 0.01 στο διάστημα [1.0, 5.0] να πάρει την τιμή του. Χρησιμοποιούμε τον πίνακα Model ως πίνακα μετασχηματισμού.
Η μεταβλητή που αντιπροσωπεύει το Sz είναι το scale_up_down η οποία αρχικοποιείται τυχαία στο διάστημα [1.0,5.0] .

Ο κώδικας (μέσα στο do-while loop) :

```
//αύξηση ύψους
if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS) {
    if (scale_up_down < 5.0f) {
        scale_up_down += 0.01f;
    }
}
//μείωση ύψους
if (glfwGetKey(window, GLFW_KEY_P) == GLFW_PRESS) {
    if (scale_up_down > 1.0f) {
        scale_up_down -= 0.01f;
    }
}
//transform matrix for scale up/scale down
Model = glm::scale(glm::mat4(1.0f), glm::vec3(1.0f, 1.0f, scale_up_down));
```

Εξήγηση κώδικα: Το πρώτο if είναι για την κλιμάκωση , αφού πατηθεί το πλήκτρο u γίνεται ένας ακόμη έλεγχος ώστε το Sz να μην ξεπεράσει το 5 καθώς θέλουμε το h = 2*Sz <=10.
Αντίστοιχα στη σμίκρυνση αντιστοιχεί το if που γίνεται έλεγχος αν πατήθηκε το πλήκτρο p, αν πατήθηκε γίνεται ακόμη ένας έλεγχος ώστε το Sz να μη γίνει μικρότερο από 1 καθώς θέλουμε h = 2*Sz >=2 .
Με την εντολή scale φτιάχνουμε τον scale Matrix .

Ερώτημα v)

Για τις κινήσεις της κάμερας και τον υπολογισμό του νέου πίνακα View καλείται η συνάρτηση camera_function() κάθε φορά πρίν τον υπολογισμό του MVP μέσα στο do –while loop.

```
//for camera movements
camera_function();
// new MVP
MVP = Projection * View * Model;
glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
```

Για διευκόλυνση ορίσαμε τα εξής global διανύσματα και πίνακες. Κάναμε τον πίνακα View global ώστε μετά τις αλλαγές στη θέση της κάμερας να τον υπολογίζουμε εκ νέου μέσα στη camera_function() . Το διάνυσμα help_position είναι οι ομογενείς συντεταγμένες του camera_position. Έχουμε τέσσερις πίνακες περιστροφής και το διάνυσμα helpv που είναι για τη λειτουργία zoom.

```
///*************************************************************
// GLOBAL μεταβλητές
//*************************************************************
glm::vec4 help_position = glm::vec4(10.0f, 50.0f, 0.0f, 1.0f);
glm::vec3 camera_position = glm::vec3(10.0f, 50.0f, 0.0f);
glm::vec3 helpv = glm::vec3(0.0f,0.0f,0.0f) - camera_position;
glm::mat4 rotate1_x = glm::rotate(glm::mat4(1.0f), 0.01f, glm::vec3(1.0f, 0.0f, 0.0f));
glm::mat4 rotate2_x = glm::rotate(glm::mat4(1.0f), -0.01f, glm::vec3(1.0f, 0.0f, 0.0f));
glm::mat4 rotate1_z = glm::rotate(glm::mat4(1.0f), 0.01f, glm::vec3(0.0f, 0.0f, 1.0f));
glm::mat4 rotate2_z = glm::rotate(glm::mat4(1.0f),-0.01f, glm::vec3(0.0f, 0.0f, 1.0f));
glm::mat4 View;
```

Για την κίνηση γύρω από τον άξονα x θα γίνεται περιστροφή της καμέρας. Σύμφωνα με τη θεωρία ο πίνακας περιστροφής στις 3Δ γύρω από τον άξονα x είναι:

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Με την εντολή rotate φτιάξαμε δύο πίνακες rotate1_x,rotate2_x για την περιστροφή της κάμερας γύρω από τον x άξονα με γωνία 0.01 και -0.01 για την αντίστροφη περιστροφή. Μέσα στη συνάρτηση camera_function γίνεται έλεγχος αν πατήθηκε το w ή το x και κάθε φορά υπολογίζεται η νέα θέση της κάμερας πολλαπλασιάζοντας το διάνυσμα help_position που αποτελεί τις ομογενείς συντεταγμένες του διανύσματος camera_position , με τον πίνακα περιστροφής ο οποίος είναι 4*4. Αν πατηθεί το w πολλαπλασιάζεται με τον rotate1_x ενώ αν πατηθεί το x πολλαπλασιάζεται με τον rotate2_x για αντίστροφη περιστροφή. Στη συνέχεια ενημερώνεται το 3*3 διάνυσμα camera_position καθώς και helpv ώστε να γίνεται zoom από τη νέα θέση.

Κώδικας:

```
// ΚΙΝΗΣΗ ΓΥΡΩ ΑΠΟ ΤΟΝ x'x
if (glfwGetKey(window, GLFW_KEY_W)) {
    help_position = rotate1_x * help_position;
    camera_position = glm::vec3(help_position[0], help_position[1], help_position[2]);
    helpv = glm::vec3(0.0f, 0.0f, 0.0f) - camera_position;
}
if (glfwGetKey(window, GLFW_KEY_X)) {
    help_position = rotate2_x * help_position;
    camera_position = glm::vec3(help_position[0], help_position[1], help_position[2]);
    helpv = glm::vec3(0.0f, 0.0f, 0.0f) - camera_position;
}
```

Αντίστοιχα για την περιστροφή γύρω από τον z άξονα ο πίνακας περιστροφής είναι:

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Με την εντολή rotate φτιάξαμε δύο πίνακες rotate1_z, rotate2_z για την περιστροφή της κάμερας γύρω από τον z άξονα με γωνία 0.01 και -0.01 για την αντίστροφη περιστροφή. Οπότε μέσα στη συνάρτηση camera_function γίνεται έλεγχος αν πατήθηκε το q ή το z και κάθε φορά υπολογίζεται η νέα θέση της κάμερας πολλαπλασιάζοντας το help_position με τον πίνακα περιστροφής rotate1_z αν πατηθεί το q, ή με τον rotate2_z για αντίστροφη περιστροφή αν πατηθεί το z. Στη συνέχεια ενημερώνεται το camera_position και το helpv .

Κώδικας:

```
// ΚΙΝΗΣΗ ΓΥΡΩ ΑΠΟ ΤΟΝ z'z
if (glfwGetKey(window, GLFW_KEY_Q)) {
    help_position = rotate1_z * help_position;
    camera_position = glm::vec3(help_position[0], help_position[1], help_position[2]);
    helpv = glm::vec3(0.0f, 0.0f, 0.0f) - camera_position;
}
if (glfwGetKey(window, GLFW_KEY_Z)) {
    help_position = rotate2_z * help_position;
    camera_position = glm::vec3(help_position[0], help_position[1],help_position[2]);
    helpv = glm::vec3(0.0f, 0.0f, 0.0f) - camera_position;
}
```

Για τη λειτουργία του zoom προσθέσαμε τον παρακάτω κώδικα μέσα στη camera_function():

```
//ZOOM
if (glfwGetKey(window, GLFW_KEY_KP_ADD) == GLFW_PRESS) {
    camera_position += 0.01f * helpv;
    help_position = glm::vec4(camera_position[0], camera_position[1], camera_position[2],1.0f);
}
if (glfwGetKey(window, GLFW_KEY_KP_SUBTRACT) == GLFW_PRESS) {
    camera_position -= 0.01f * helpv;
    help_position = glm::vec4(camera_position[0], camera_position[1], camera_position[2], 1.0f);
}
```

Θέλουμε να μετακινούμε την κάμερα προς το σημείο P(0,0,0) όταν πατηθεί το + και πίσω προς τη θέση της όταν πατηθεί το - . Χρησιμοποιούμε το βοηθητικό διάνυσμα
helpv = P(0,0,0) – camera_position το οποίο πολλαπλασιάζουμε με ένα συντελεστή το 0.01 και το προσθέτουμε στο camera_position αν θέλουμε να κάνουμε zoom in ή το αφαιρούμε αν θέλουμε να κάνουμε zoom out. Το helpv σε κάθε συνιστώσα του έχει την «απόσταση» ανάμεσα στις αντίστοιχες συνιστώσες του P(0,0,0) και του camera_position , οπότε προσθέτουμε ένα «βήμα» στο camera_position για να πλησιάσει η κάμερα στο σημείο P, αντίστοιχα αν το αφαιρούμε απομακρύνεται , το βήμα είναι 0.01*helpv .
Όταν γίνεται zoom ενημερώνουμε και help_position ώστε να γίνονται και οι άλλες λειτουργίες της κάμερας (κινήσεις γύρω από τους αξονες x,z) από τη θέση του zoom.
Τέλος στη camera_function υπολογίζεται εκ νέου ο πίνακας View:

```
View = glm::lookAt(
    camera_position,   // θέση της κάμερας
    glm::vec3(0.0f, 0.0f, 0.0f),    // η κάμερα κοιτάζει εδώ
    glm::vec3(0.0f, 0.0f, 1.0f)     //up vector
);
```

Πληροφορίες σχετικά με την υλοποίηση:

•Λειτουργικό Σύστημα : Windows 10 Home

• Περιβάλλον: Visual Studio x86

Πηγές και βοήθεια: συμμετοχή στις ώρες εργαστηρίου,γενικά google , www.opengl-tutorial.org, learnopengl.com, www.songho.ca/opengl

Στα αρχεία ProjBFragmentShader και ProjBVertexShader δεν κάναμε κάποια αλλαγή.