Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

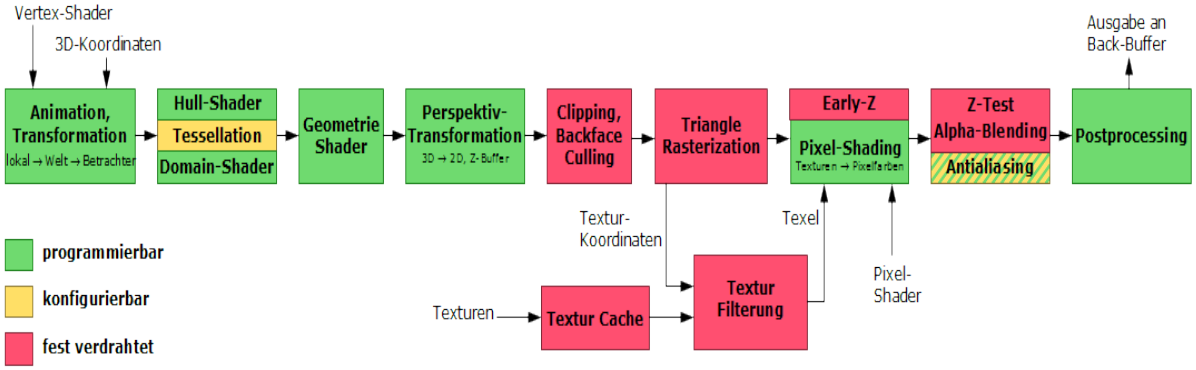
Προπτυχιακό Μάθημα: «Γραφικά Υπολογιστών και Συστήματα Αλληλεπίδρασης»

Πρώτη Προγραμματιστική Άσκηση

Όνομα Φοιτητών – Α.Μ.:

Παναγιώτης Ντουνέτας - 2781

Γεώργιος Κρομμύδας – 3260



ΙΩΑΝΝΙΝΑ,

2020

Μέρος - 1ο: Περιγραφή Εργασίας:

Σε αυτή την προγραμματιστική άσκηση είχαμε να υλοποιήσουμε με χρήση της γλώσσας C++ και της OpenGL μία σκηνή η οποία περιέχει στερεά αντικείμενα. Ως βασικό κομμάτι της σκηνής έχουμε έναν διάφανο κύβο, ο οποίος στο εσωτερικό του περιέχει μία σφαίρα στο κέντρο του και πολλά άλλα μικρά αντικείμενα. Τα αντικείμενα αυτά μπορεί να είναι είτε μικρές σφαίρες με ακτίνα **d**, η οποία δεν πρέπει να ξεπερνάει την τιμή 10, είτε μικρούς κυλίνδρους με ύψος αντίστοιχα **d** και διάμετρο βάσης **d**, είτε μικροί κύβοι με πλευρά **d**.

Μέρος - 2ο: Υλοποίηση Εργασίας:

Το πρώτο κομμάτι που υλοποιήσαμε ήταν το παράθυρο για την σκηνή μας. Χρησιμοποιώντας την βιβλιοθήκη της <**GLFW/glfw3.h**>, σχεδιάσαμε ένα παράθυρο με το όνομα 'Συγκρουόμενα'. Αρχικά το παράθυρο μας έχει διαστάσεις 600x600, τις οποίες τις αποθηκεύσαμε σε δύο μεταβλητές **win_h** και **win_l**. Για να δημιουργήσουμε το παράθυρο χρησιμοποιούμε την εντολή **glfwCreateWindow()** με τις παραπάνω διαστάσεις και για να δώσουμε τον τίτλο χρησιμοποιούμε την συνάρτηση **glfwSetWindowTitle()** με την ιδιαιτερότητα πως χρησιμοποιούμε το **u8** αριστερά του τίτλου για να εμφανιστούν οι ελληνικοί χαρακτήρες. Επιπλέον, γίνεται αρχικοποίηση της GLEW. Το χρώμα του παραθύρου θα πρέπει να είναι μαύρο και αυτό το επιτυγχάνουμε με την εντολή **glClearColor()**, με τα ορίσματά της να έχουν την τιμή **0.0f**.

<<glEnable, glBlendFunc: Η `glBlendFunc(GL_SRC_COLOR, GL_DST_COLOR)` χρησιμοποιεί τα χρώματα εισόδου και προορισμού. Στο Internet πολλοί χρησιμοποιούσαν τα ορίσματα GL_ONE_MINUS_SRC_ALPHA και GL_ONE_MINUS_DST_ALPHA. Στην δική μας περίπτωση αυτά δεν εμφάνιζαν τίποτα παρά μόνο το μαύρο background. >>

Κατά το άνοιγμα του παραθύρου καλούμε και τους shaders **TransformVertexShader.vertexshader**, **ColorFragmentShader.fragmentshader** μέσω της συνάρτησης **LoadShaders()**. Παίρνουν τον πίνακα μετασχηματισμού **MVP** και χρωματίζουν τα αντικείμενα. Σε περίπτωση που θέλουμε να κλείσουμε το

παράθυρο, τότε χρησιμοποιούμε την συνάρτηση **glfwGetKey()** μαζί με το κουμπί **ESC**, το οποίο καλείται από το όρισμα **GLFW_KEY_ESCAPE**.

Το επόμενο κομμάτι που υλοποιήσαμε είναι τα αντικείμενα της σκηνής. Το πρώτο αντικείμενο που υλοποιήθηκε ήταν ο κύβος. Το πρόγραμμα **Cube.cpp** υλοποιεί τον κύβο με την συνάρτηση **makeCube()**, η οποία δέχεται ως όρισμα το αρχικό σημείο κατασκευής του κύβου μέσα στην σκηνή **(posX, posY, posZ)** και το όρισμα **size**, το οποίο σχεδιάζει τις κορυφές του και τις αντίστοιχες ακμές του. Επιπλέον, η κατασκευή του κύβου γίνεται με χρήση τριγώνων. Ο πίνακας **cube_verticles[]** αποθηκεύει τις κορυφές του κύβου που επρόκειτο να δημιουργηθούν. Επίσης, ο χρωματισμός του κύβου γίνεται μέσω του πίνακα **cube_color[]** ο οποίος χρωματίζει τον κύβο με τα **rgb** και με το **a** καθορίζει την διαφάνειά του. Για να γίνουν οι παραπάνω διαδικασίες χρησιμοποιούμε τον **vertexbuffer** όπου σχεδιάζει τις ακμές των τριγώνων του κύβου και τον **colorbuffer** όπου χρωματίζει τον κύβο. Μετά την δημιουργία του κύβου, διαγράφουμε τους **buffers**, έτσι ώστε να τους χρησιμοποιήσουμε για την δημιουργία του επόμενου αντικειμένου.

Το δεύτερο αντικείμενο της σκηνής μας είναι η σφαίρα. Το πρόγραμμα **Sphere.cpp** υλοποιεί την σφαίρα με χρήση της συνάρτησης **makeSphere()** όπου δέχεται ως ορίσματα την ακτίνα που θα έχει και τα άλλα τρία είναι για την κίνηση της σφαίρας μέσα στην σκηνή. Αρχικοποιούμε κάποιες σταθερές που αφορούν το μέγεθος του grid της σφαίρας και τους πίνακες **sphere_pos[nn]**, **sphere_nor[nn]**, **spere_col[nn]** και **sphere_ix[na*(nb - 1)*6]**. Ο πρώτος πίνακας κρατάει την πληροφορία για τις κορυφές της σφαίρας, ο δεύτερος πίνακας κρατάει την πληροφορία της σφαίρας όταν κανονικοποιηθεί, δηλαδή η επιφάνεια της σφαίρας δεν εμφανίζει είναι λεία και δεν εμφανίζονται τα τρίγωνα, ο τρίτος πίνακας κρατάει το χρώμα της σφαίρας και ο τελευταίος πίνακας κρατάει την πληροφορία των ακμών των τριγώνων. (*Παρατήρηση: Επειδή ο κώδικας της σφαίρας ήταν αρκετά περίπλοκος και η μέθοδος δημιουργίας της "περίεργη" , ενώ προσπαθήσαμε διάφορους τρόπους να επεξεργαστούμε το χρώμα ανά "τρίγωνο" όπως και στον κύβο δεν είχαμε σωστό αποτέλεσμα διότι υπήρχε παραμόρφωση της τελικής σφαίρας.*) Αρχικά, σχεδιάζουμε τα σημεία της σφαίρας και στην συνέχεια με χρήση τριγώνων να τα ενώνουμε. Υπολογίζουμε τα σημεία με χρήση των παραμετρικών εξισώσεων και τα

αποθηκεύουμε στον πίνακα **sphere_pos[nn]**. Για να υλοποιηθεί η σφαίρα, την χωρίσαμε σε ημισφαίρια, τα οποία τα φτιάξαμε με χρήση τριγώνων. Πρώτα από όλα, ενώνουμε τα σημεία με τις αντίστοιχες ακμές, έτσι ώστε να σχηματιστούν. Το κάνουμε και κάθε τέταρτο του ημισφαιρίου, ανανεώνοντας τον πίνακα **sphere_ix[ix]**. Αυτή η διαδικασία εκτελείται για όλες τις ακμές της σφαίρας στο εκάστοτε ημισφαίριο μέχρι να ολοκληρωθεί η κατασκευή της.

Τέλος, για την δημιουργία του αντικειμένου χρησιμοποιούμε τους **buffers** και τους πίνακες **VAO/VBO** οι οποίοι θα καλούνται για την εμφάνιση της σφαίρας στην σκηνή μας. Αρχικοποιήσαμε μία μεταβλητή **GLuint i**, η οποία μας δείχνει τα βήματα της υλοποίησης. Όταν πάρει την τιμή 0, τότε θα ξεκινήσει η ένωση των κορυφών της σφαίρας. Όταν πάρει την τιμή 1, τότε θα ξεκινήσει να σχεδιάζει τις ακμές που συνδέει τις κορυφές. Όταν πάρει την τιμή 2, τότε θα ξεκινήσει να εξομαλύνει το σχήμα και να μην εμφανίζονται τα τρίγωνα. Τέλος, όταν πάρει την τιμή 3, τότε θα χρωματιστεί η σφαίρα.

Τα αντικείμενα εμφανίζονται μέσω του πίνακα **MVP** ο οποίος προβάλει στην σκηνή μας τα αντικείμενα που μπορεί να δει η κάμερα. Για να εμφανιστεί ο διαφανής κύβος **SC** στη σκηνή μας, καλούμε την συνάρτηση **makeCube(0, 0, 0, 100, k, p, mp)** στην **mymain.cpp** μέσα στο while loop που κρατάει το παράθυρο ανοικτό. Ο κύβος αποτελεί το βασικό κομμάτι της σκηνής μας και μέσα σε αυτό θα υπάρχουν τα υπόλοιπα αντικείμενα. Για να εμφανιστεί η σφαίρα **SPH** καλούμε την συνάρτηση **makeSphere(15, mv_updown, mv_lr, mv_io)** η οποία δέχεται την ακτίνα ως πρώτο όρισμα και τα υπόλοιπα καθορίζουν την κίνηση της σφαίρας μέσα στην σκηνή.

Επιπλέον:

-Αρχείο object loading μαζί με το Header αρχείο του: Δημιουργήσαμε τον Loader με βάση το tutorial, σε παραλλαγή με ένα βίντεο στο youtube(αποτελείται απο 3 parts 45,46,47) επειδή οι εντολές του tutorial ήταν της C και στο δικό μας μηχάνημα υπήρχε το ERROR στις fscanf, fprintf "Unsafe commands". Διαβάζει μία προς μία τις γραμμές του .obj αρχείου και ανάλογα με το πρόθεμα κάθε γραμμής αποθηκεύει στους κατάλληλους πίνακες τα νούμερα των vertices, normals και UV (για σφαίρα ή

κύλινδρο). Όμως το Blender αντικείμενο δεν φορτώνεται με το error “invalid preprocessor command 'Blender' ”. Το παραθέτουμε μήπως έχετε κάποια εύκολη λύση από εμπειρία στο παρελθόν. Η συνολική ιδέα ήταν να έχουμε ένα αντικείμενο κύβο/σφαίρα/κύλινδρο τα οποία θα λειτουργούν μόνο για τα μικρά νέα τυχαία αντικείμενα. Είναι ανεξάρτητα από τον κυρίως κύβο και σφαίρα διότι εκείνα περιέχουν έξτρα ορίσματα για την αρχικοποίηση τους.

-Αρχείο shaderHandler και το Header του: Δημιουργούν τη διαδικασία για τη φόρτωση των shaders και ελέγχεται αν γίνεται σωστά, αλλιώς εμφανίζει μήνυμα λάθους.

Τέλος, μετά την main συνάρτηση βρίσκεται ο handler των πλήκτρων για τα ερωτήματα 3 και 4.

Ερώτημα 3:

Βέλη LEFT, RIGHT: Επιτρέπουν την κίνηση της σφαίρας στον άξονα x.

Βέλη UP, DOWN: Επιτρέπουν την κίνηση της σφαίρας στον άξονα y.

Πλήκτρα =,- (Το + και – στην εκφώνηση): Επιτρέπουν την κίνηση της σφαίρας στον άξονα z.

Ερώτημα 4:

Πλήκτρα a,d: Κίνηση της κάμερας στον άξονα x.

Πλήκτρα s,x: Κίνηση της κάμερας στον άξονα y.

Πλήκτρα w,e: Κίνηση της κάμερας στον άξονα z.

Τα παραπάνω δίνονται σαν όρισμα (τα πρώτα στην glm::LookAt και τα επόμενα στην ίδια τη σφαίρα) ανα frame για να επιτευχθεί η κίνηση.

Μέρος - 3ο: Αποτελέσματα Εργασίας:

Με λιγότερα λόγια, έχουν υλοποιηθεί μερικώς τα ερωτήματα 1,3 και ολόκληρο το 4.

Προβλήματα/Παρατηρήσεις/Ειδικές συνθήκες:

-Το πρόγραμμα έχει θέμα με υπερχείλιση μνήμης. Τρέχει για περίπου 10 δευτερόλεπτα και ύστερα σταματάει. Ξέρουμε ότι αυτό ίσως οφείλεται στην έλλειψη disable εντολών ή στο μη – άδειασμα των Buffers και σε κάθε Loop της while υπερφορτώνεται από νέους “βασικούς” κύβους και σφαίρες.

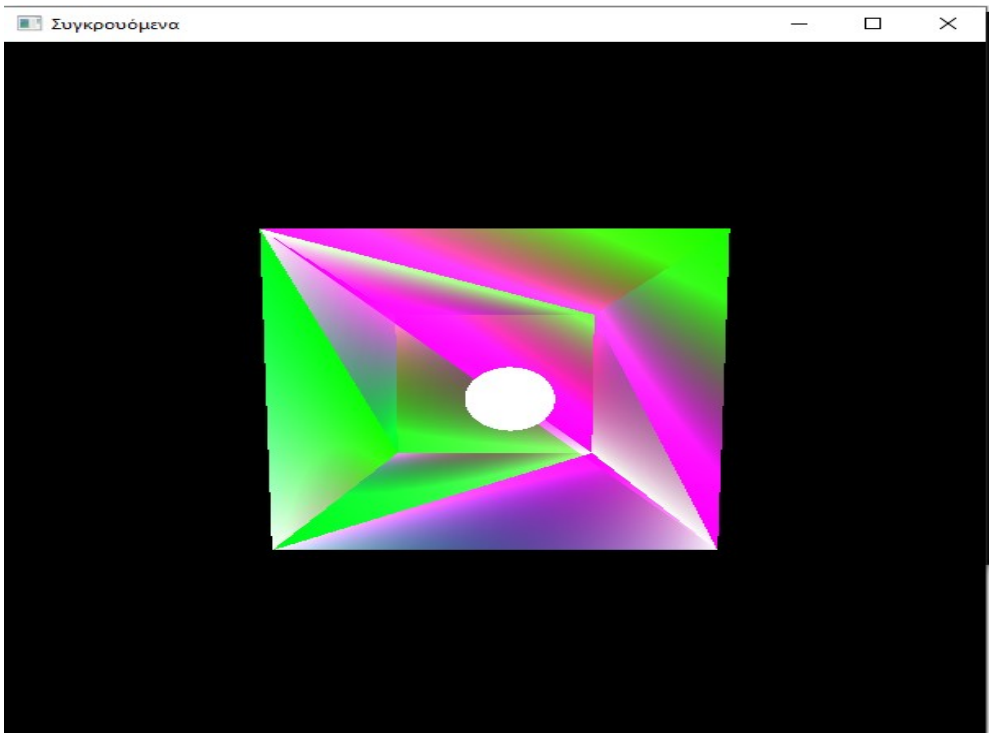
-Το χρώμα του κύβου δεν είναι πλήρως transparent. Αυτό οφείλεται είτε σε λάθος στους shaders ή (όπως ανέφερα παραπάνω) σε λάθος χρήση των ορισμάτων της glBlendFunc.

-Το χρώμα της σφαίρας επηρεάζεται από τον shader Και παραμένει μονίμως λευκό ακόμα και όταν πήγαμε χειροκίνητα να δώσουμε το κόκκινο.

-Στην αρχή του προγράμματος υπάρχουν οι μεταβλητές υπεύθυνες για την κίνηση της κάμερας και της σφαίρας όπως και τα random χρώματα με επιλογές 0 ή 1.

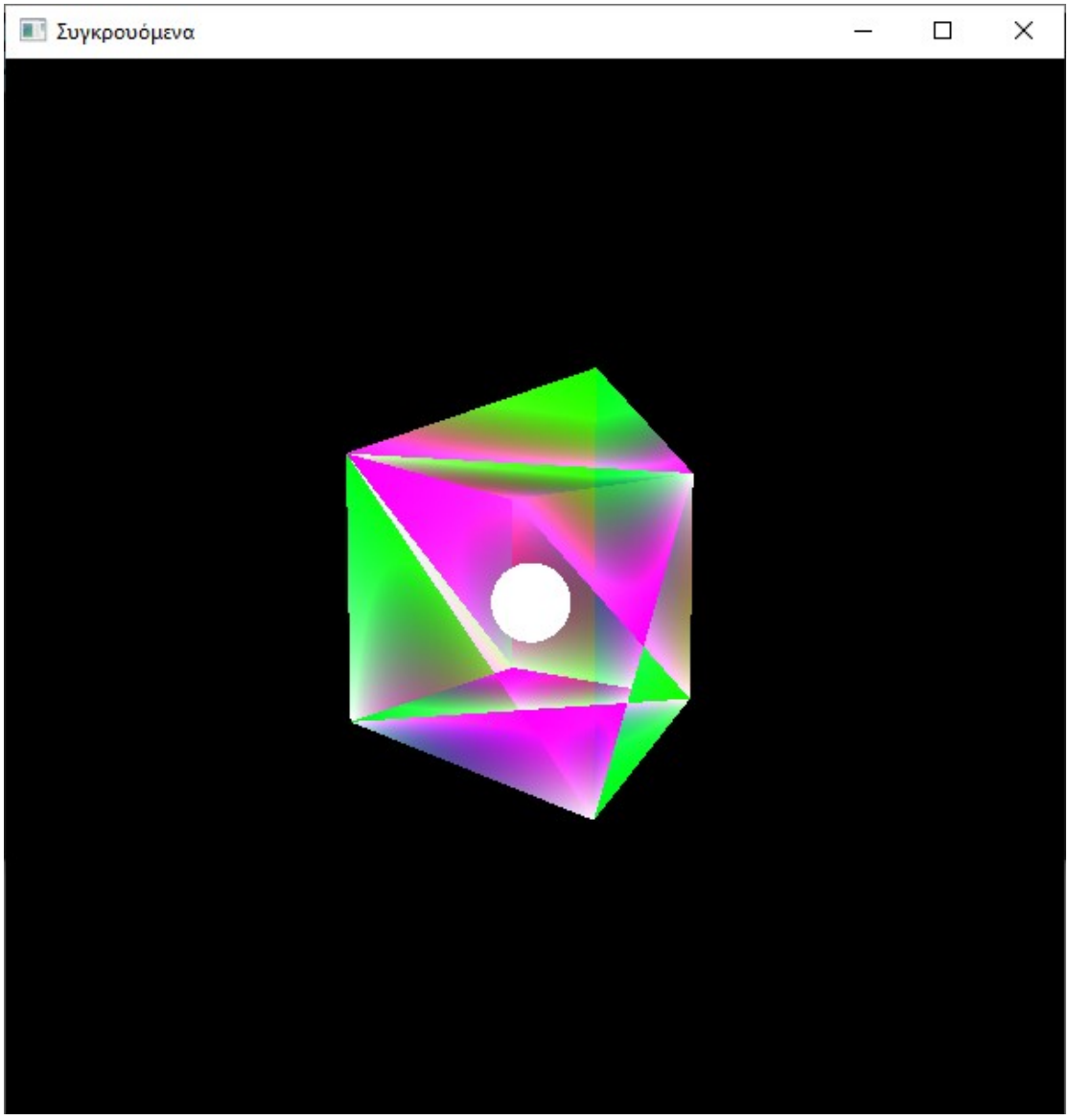
Παραδείγματα λειτουργίας:

1. Απλή έναρξη:

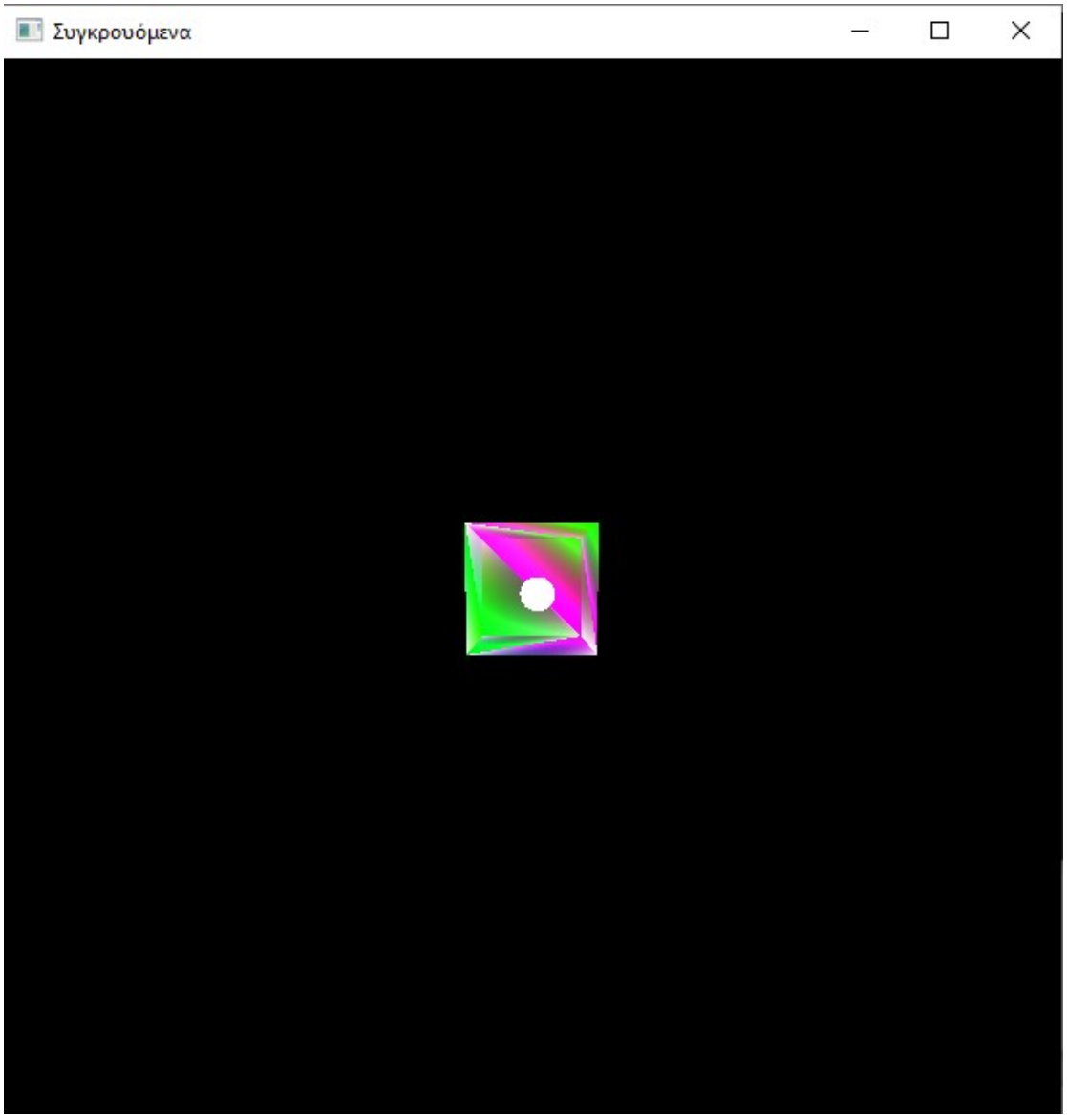


2. Μετακίνηση κάμερας σε όλους τους άξονες:

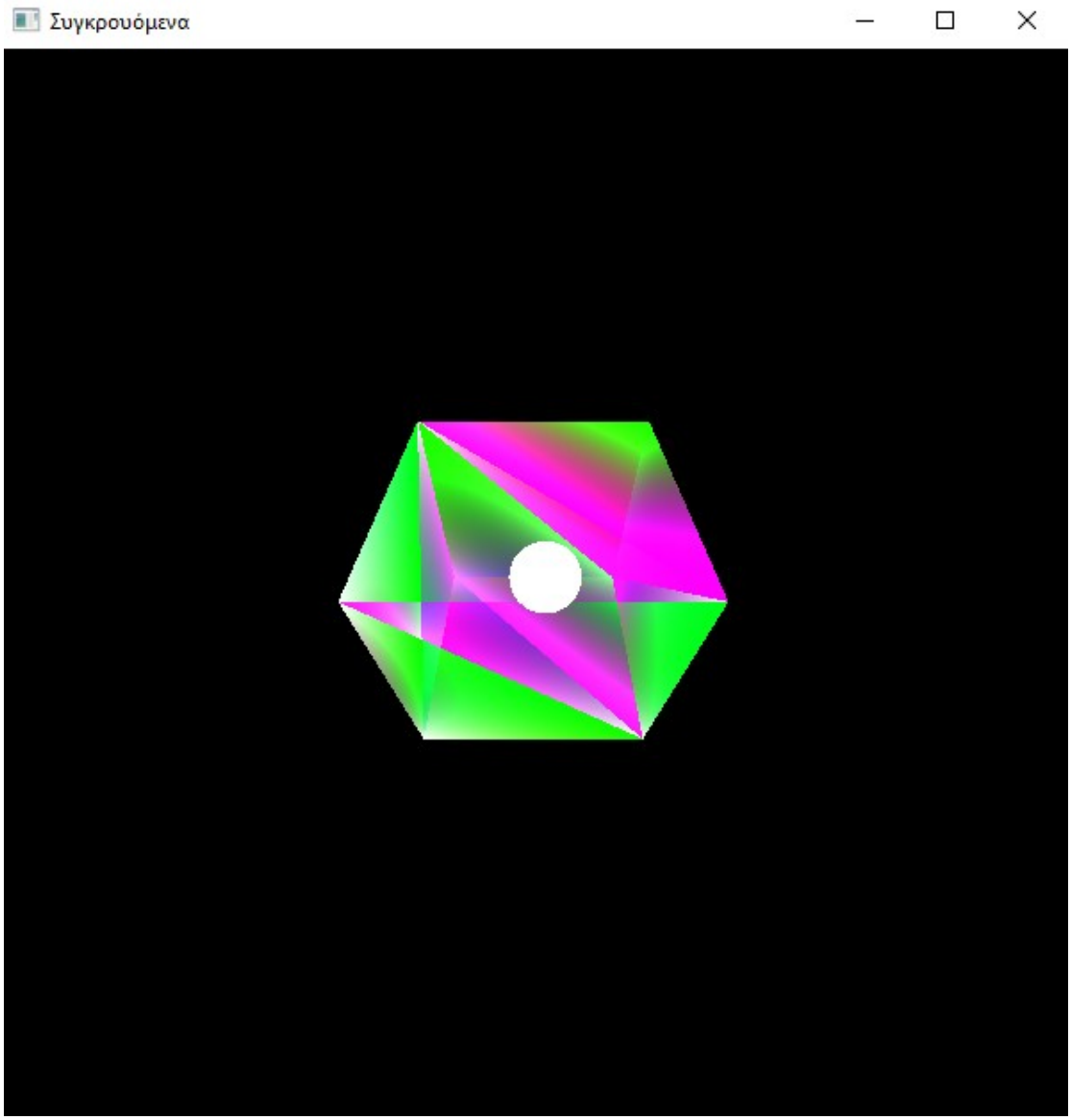
Μετκίνηση στον Z:



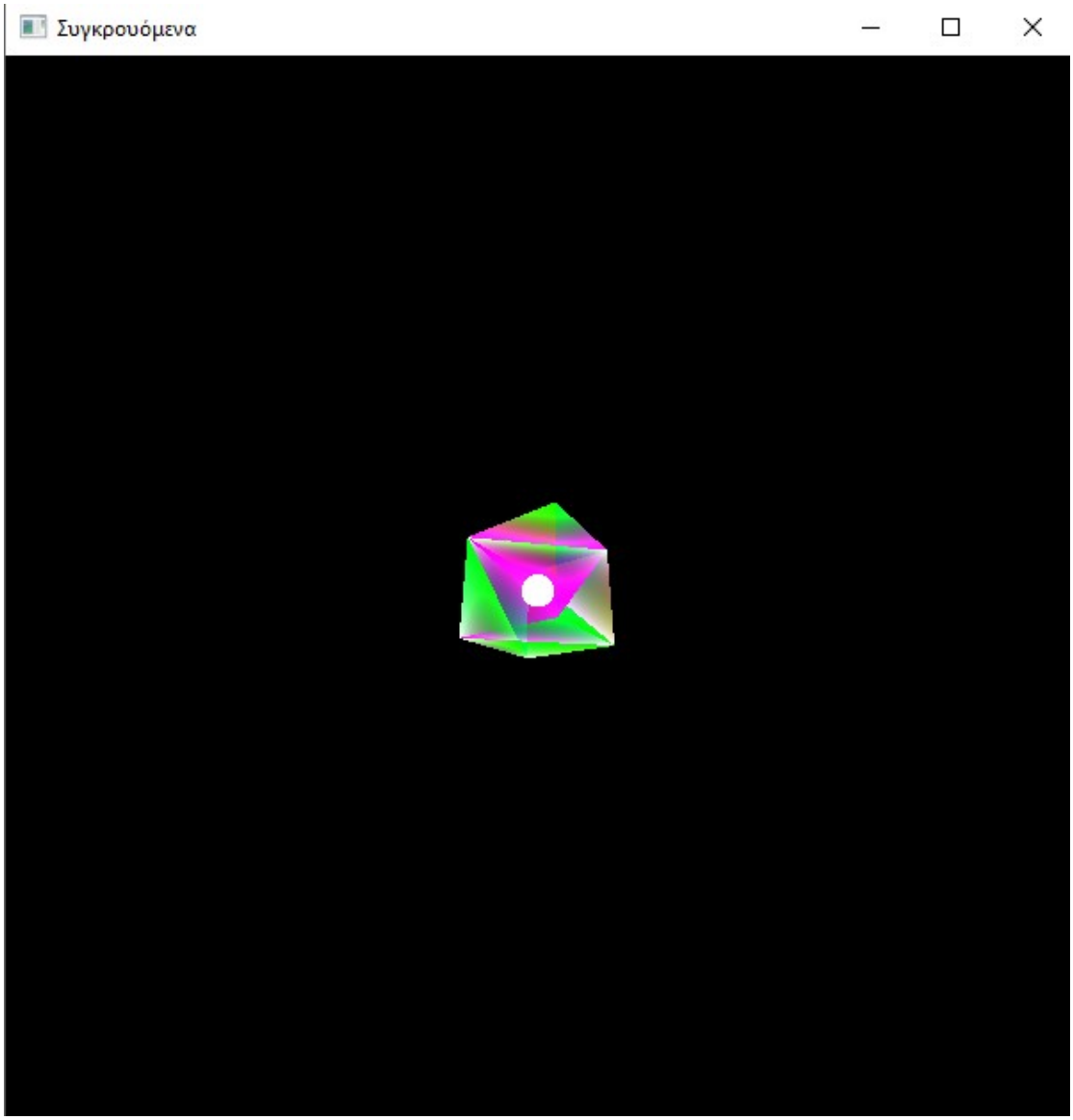
Μετακίνηση στον X:



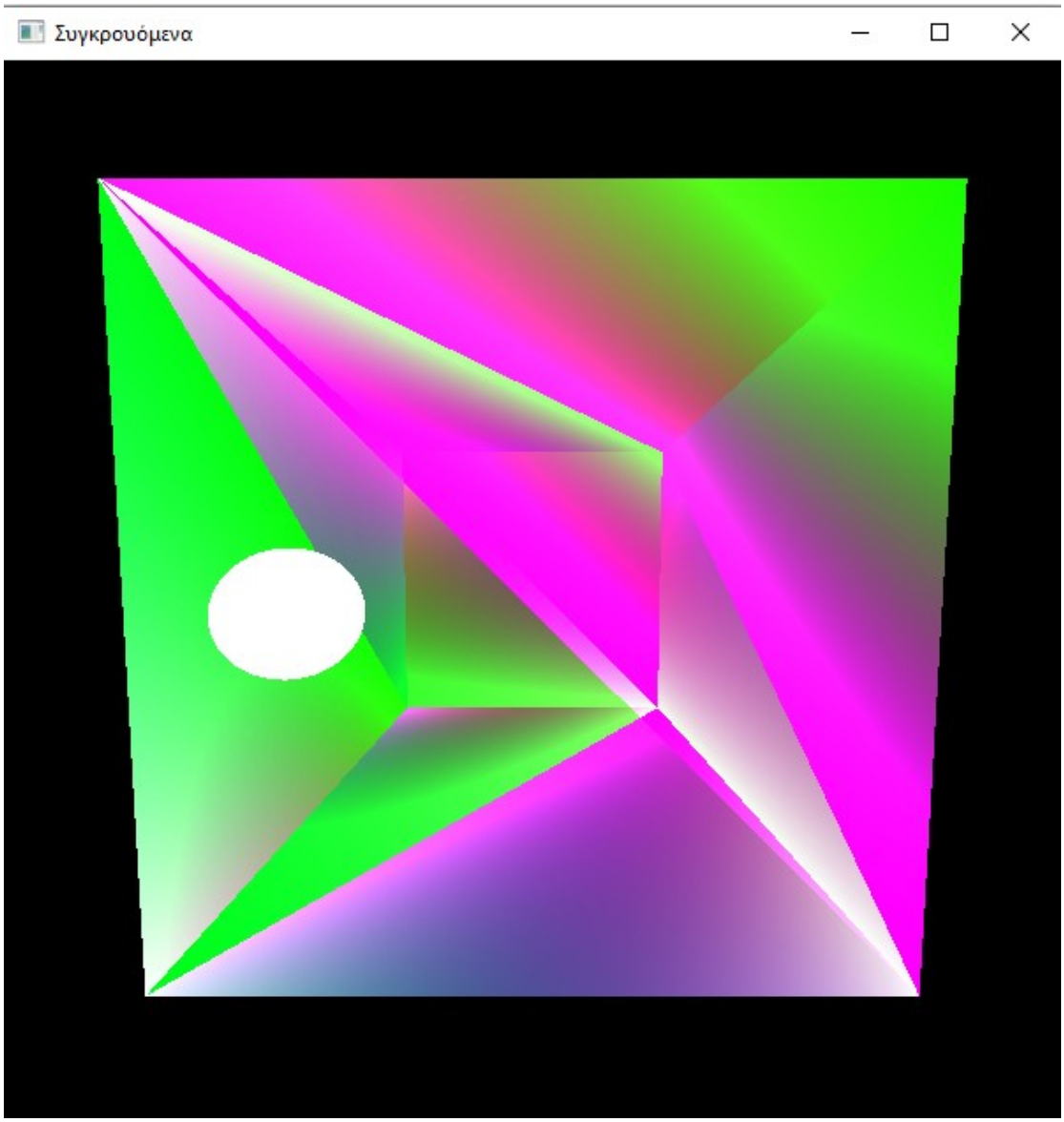
Μετακίνηση στον Y:

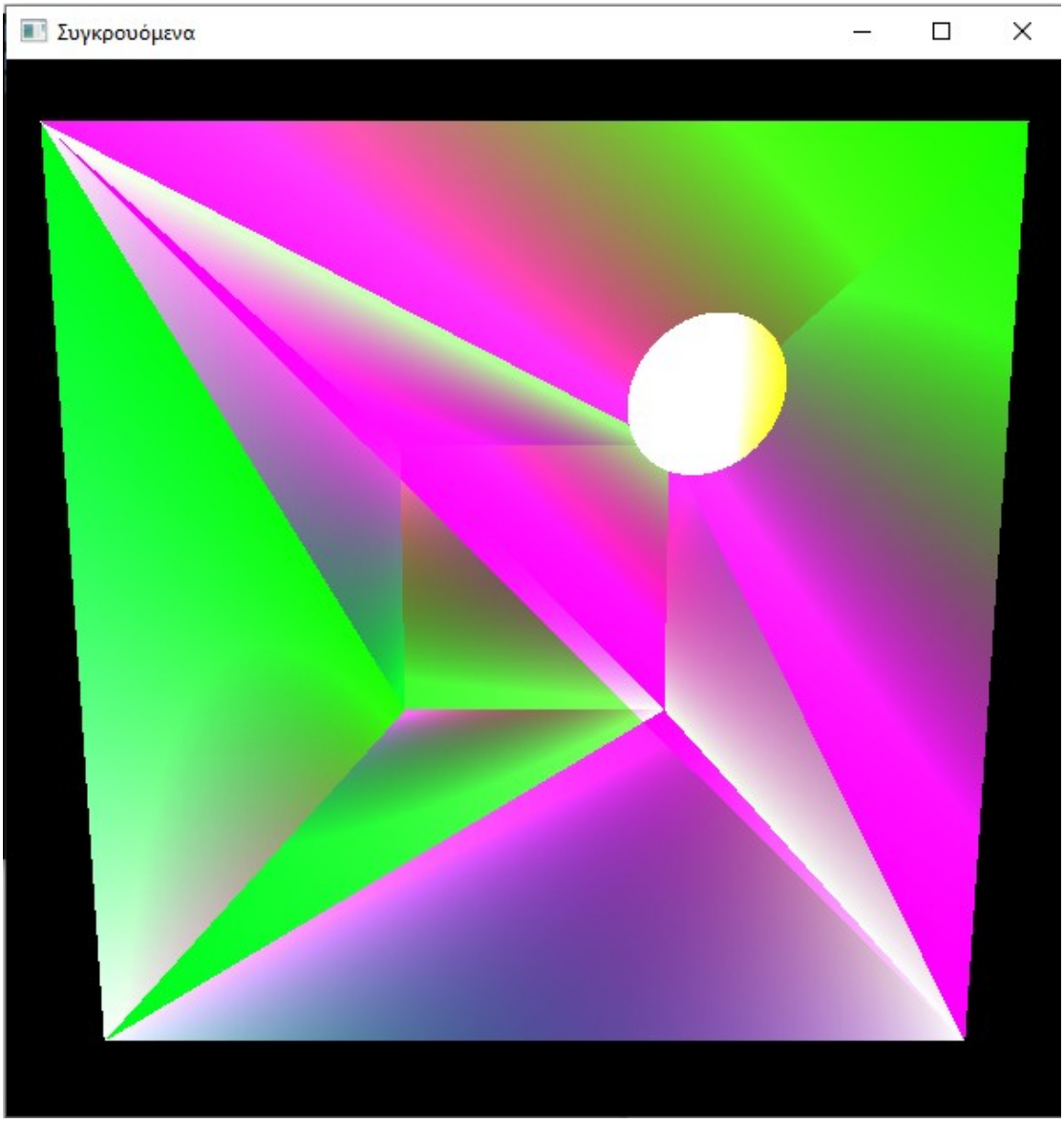


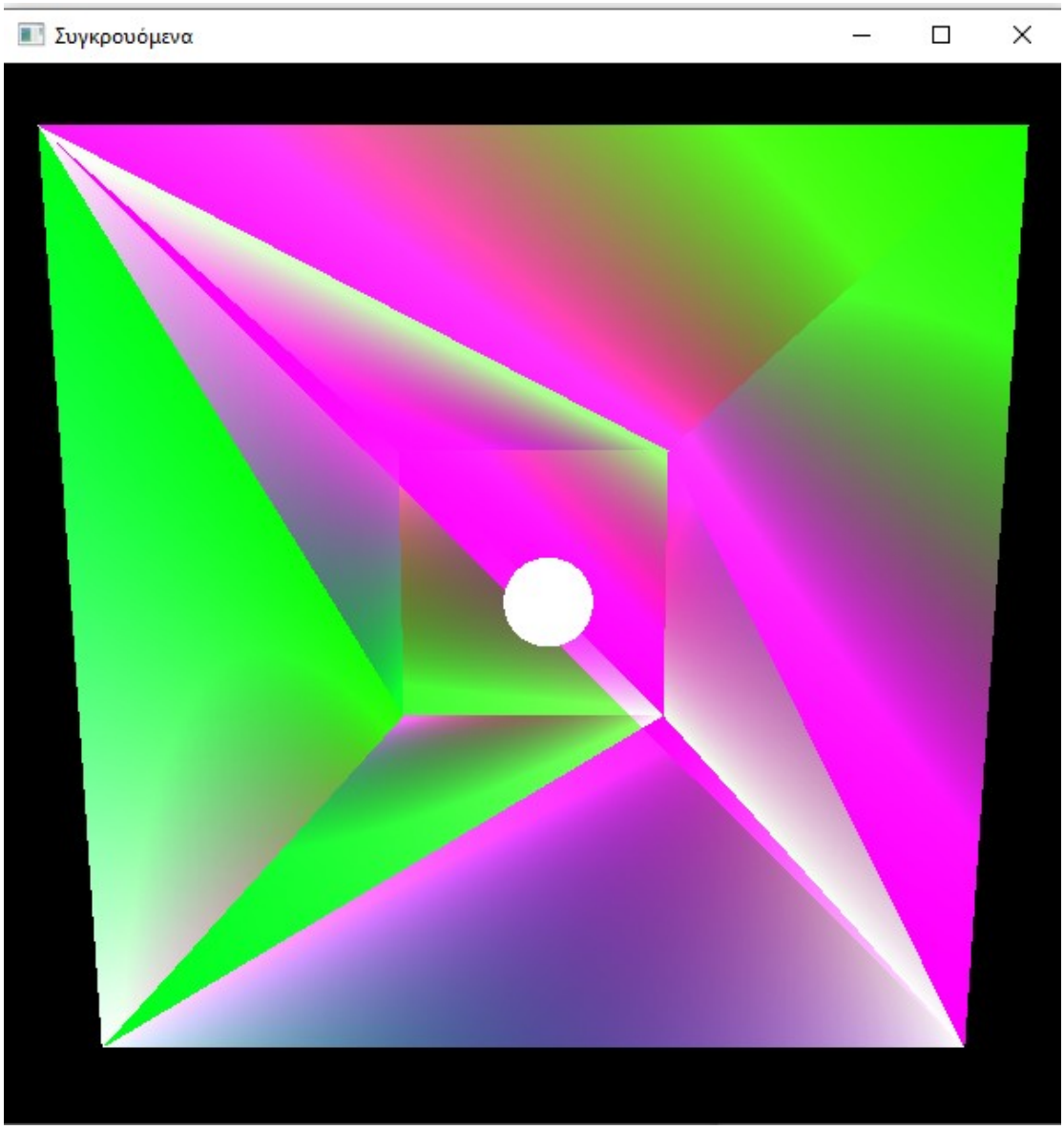
Τυχαία:



Και μερικά παραδείγματα κίνησης της σφαίρας:




Συγκρουόμενα


Συγκρουόμενα

Μέρος - 4ο: Βιβλιογραφία Εργασίας – Πηγές στο Διαδίκτυο:

Δημιουργία Σφαίρας: https://stackoverflow.com/questions/43087620/draw-sphere-in-opengl-4-0 , τελευταία απάντηση.

Βοήθεια για τις συναρτήσεις της OpenGL3.3: http://docs.gl/

Ιδέες για υλοποίηση γενικότερα:

https://github.com/giawa/opengl4tutorials – Tutorials άλλης μορφής

https://www.youtube.com/watch?v=chbjxTmbQes – Blending και χρήση της glBlendFunc

https://stackoverflow.com/questions/39377679/scaling-and-moving-3d-object-with-opengl – Βοήθεια στον τρόπο σκέψης για την κίνηση της κάμερας/ τροποποίηση μεγέθους σχημάτων.

Επιπλέον όλα τα δικά σας tutorial που είχατε ήδη δώσει θα παρατηρήσετε πως το 90% του κώδικα είναι βασισμένο εκεί.

Ευχαριστούμε για τον χρόνο σας και μακάρι να μπορούσαμε να κάνουμε ακόμα περισσότερα.