Γραφικά Υπολογιστών & Συστήματα Αλληλεπίδρασης
Αναφορά Προγραμματιστικής Άσκησης 1-Β:

Εξάσκηση στη χρήση βασικών βιβλιοθηκών γραφικών της OpenGL 3.3

Μέλη ομάδας:

Αντωνίου Χριστόδουλος AM: 2641
Τσιούρη Αγγελική AM: 3354

Εισαγωγική σημείωση

Όπως και στην πρώτη άσκηση, αρχικά, μελετήσαμε τα εργαστήρια του μαθήματος μαζί με τον ενδεικτικό κώδικα που μας δόθηκε σε αυτά για να κατανοήσουμε τις έννοιες που απαιτεί η άσκηση. Στη συνέχεια με τη βοήθεια του ενδεικτικού κώδικα των εργαστηρίων, κάναμε προσθήκες και τροποποιήσεις στον κώδικα μας από την πρώτη άσκηση για να πετύχουμε τα ζητούμενα του κάθε ερωτήματος. Η άσκηση έχει αναπτυχθεί σε Windows 10 με Visual Studio 2019. Το παραδοτέο .zip αρχείο περιλαμβάνει τα αρχεία "Main.cpp", "controls.cpp" και "controls.hpp" με τον κώδικα της άσκησης και τα συμπληρωματικά αρχεία "ColorFragmentShader.fragmentshader" και "TransformVertexShader.vertexshader" με τους shaders. Δεν συμπεριελαμβάνονται άλλα αρχεία των πακέτων ή του Visual Studio. Για να τρέξει το πρόγραμμα θα χρειαστεί να γίνει import σε ένα Visual Studio Project το οποίο περιέχει τις βιβλιοθήκες GLM, GLEW και GLFW. Ακολουθεί η αναλυτική επεξήγηση για την λειτουργία του προγράμματος χωρισμένη ανά ερώτημα.

Ερώτημα (i) – Δημιουργία παραθύρου

Στο ερώτημα αυτό χρησιμοποιούμε την βιβλιοθήκη **GLFW** στο αρχείο "Main.cpp" για την δημιουργία ενός παραθύρου. Στην αρχή του προγράμματος, κάνουμε include τον header της βιβλιοθήκης και δηλώνουμε μία μεταβλητή τύπου **GLFWwindow*** για το παράθυρο μας. Έπειτα, στην αρχή της main, αρχικοποιούμε την GLFW καλώντας την συνάρτηση **glfwInit()** και μετά καλούμε την **glfwCreateWindow(...)** για να δημιουργήσουμε το παράθυρο. Οι πρώτες δύο παράμετροι της συνάρτησης καθορίζουν το πλάτος και το ύψος του παραθύρου σε pixels αντίστοιχα. Επομένως, εδώ καθορίζουμε το επιθυμητό 800x800. Η τρίτη παράμετρος είναι ένα string με το όνομα του παραθύρου, "Πυραμίδα". Σημειώνεται πως λόγω encoding υπάρχει πιθανότητα οι ελληνικοί χαρακτήρες να μην εμφανίζονται σωστά. Στη περίπτωση αυτή μπορούμε να πάμε στο ακόλουθο menu του Visual Studio: Property Pages -> Configuration Properties -> C/C++ -> Command Line και να προσθέσουμε το string "/utf-8" στο πεδίο Additional Options.

Για το background του παραθύρου καλούμε την συνάρτηση **glClearColor(...)**, με παραμέτρους: red=0, green=0, blue=0, alpha=1, οι οποίες αντιστοιχούν στο χρώμα black χωρίς διαφάνεια. Για τον τερματισμό του προγράμματος με το πλήκτρο **Q (κεφαλαίο)**, έχουμε υλοποιήσει τις συνάρτησεις **isCapitalKeyPressed(...)** και **isCapsLockOn(...)** στο αρχείο "controls.cpp", όπου γίνεται η διαχείριση όλων των inputs από τον χρήστη. Η συνάρτηση **isCapsLockOn(...)** χρησιμοποιεί την **GetKeyState(...)** της βιβλιοθήκης **windows.h** για να ελέγξει εάν το CAPS LOCK είναι ενεργοποιημένο ή όχι. Η **isCapitalKeyPressed(...)** δέχεται ως όρισμα μία σταθερά της βιβλιοθήκης GLFW που αντιστοιχεί σε ένα key και ελέγχει εάν το key αυτό πατήθηκε ως κεφαλαίο ή μικρό. Πιο συγκεκριμένα, χρησιμοποιεί την **isCapsLockOn(...)** αλλά και την **glfwGetKey(...)** για να ελέγξει εάν ήταν πατημένο και κάποιο από τα left και right shifts. Τέλος,

καλούμε την **isCapitalKeyPressed(...)** με όρισμα την σταθερά GLFW_KEY_Q που αντιστοιχεί στο πλήκτρο q έπειτα, στην συνθήκη του do-while loop όπου γίνεται το render στο αρχείο "Main.cpp". Ως αποτέλεσμα, όταν ο χρήστης πατήσει το κεφαλαίο Q (και μόνο για το κεφαλαίο) η επανάληψη σταματά και το πρόγραμμα τερματίζει.

Ερώτημα (ii) – Σχεδίαση πυραμίδας

Όπως και στην προηγούμενη άσκηση, το πρώτο πράγμα που χρειαζόμαστε για να ζωγραφίσουμε το ζητούμενο σχήμα είναι να δημιουργήσουμε ένα **Vertex Array Object**. Έπειτα, καθορίζουμε τις συντεταγμένες στην οθόνη. Το μοντέλο της πυραμίδας που θέλουμε να ζωγραφίσουμε αποτελείται από 6 τρίγωνα, 2 για την τετράγωνη βάση και ένα για κάθε μία από τις 4 τριγωνικές πλευρές. Θέλουμε η βάση να βρίσκεται πάνω στο επίπεδο **xy**, επομένως τα δύο τρίγωνα από τα οποία αποτελείται θα έχουν μηδενικό **z**. Επιπλέον, θέλουμε η κορυφή της πυραμίδας να βρίσκεται πάνω στον θετικό άξονα **z**. Επομένως, τα τρίγωνα των τεσσάρων τριγωνικών πλευρών θα έχουν ένα κοινό σημείο της μορφής (0,0,H). Όπου **H** είναι το ύψος. Δηλαδή, το ύψος της πυραμίδας αντιστοιχίζεται στο άξονα **z**. Έπειτα, σύμφωνα με την εκφώνηση θέλουμε η βάση της πυραμίδας να βρίσκεται πάνω στον επίπεδο **xy** και να είναι κεντραρισμένη γύρω από την αρχή των αξόνων, (0, 0, 0), με κάθε πλευρά της να έχει μήκος 5.0. Έτσι μπορούμε να συμπαιράνουμε πως τα x και y των σημείων της βάσης θα είναι 2.5 ή -2.5 έτσω ώστε να προκύπτει 5.0 ως μήκος της κάθε πλευράς. Με βάση τα παραπάνω, καθορίζουμε τις συντεταγμένες του μοντέλου μας σε ένα πίνακα **g_vertex_buffer_data** 54 θέσεων. (βλ. Ερώτημα (vi) για εικόνες υπολογισμού συντεταγμένων)

Για τον υπολογισμό του τυχαίου ύψους της πυραμίδας **H** έχουν υλοποιηθεί η συναρτήσεις **initRandomGenerator(...)** και **generateRandomFloat(...)** στο αρχείο "Main.cpp". Η **initRandomGenerator(...)** καλείται μία φορα στην αρχή της εκτέλεσης του προγράμματος και σκοπός της είναι να αρχικοποιήσει την γεννήτρια τυχαίων αριθμών. Αυτό το επιτυγχάνει, καλώντας την συνάρτηση **srand(...)** με όρισμα (seed) την χρονική στιγμή που τρέχει το πρόγραμμα. Ο λόγος που αυτό είναι απαραίτητο είναι επείδη η συνάρτηση **rand(...)**, η οποία καλείται για την παραγωγή τυχαίων αριθμών, παράγει ψευδοτυχαίους αριθμούς. Δηλαδή, δίχως την κλήση της **srand(...)** η **rand(...)** παράγει τους ίδιους "τυχαίους" αριθμούς κάθε φορά που εκτελείται το πρόγραμμα από την αρχή. Αντίθετα, με την κλήση της **srand(...)** ως αρχικοποίηση φαίνεται να παίρνουμε ικανοποιητικά τυχαίους αριθμούς για τα πλαίσια της άσκησης. Η **generateRandomFloat(...)** χρησιμοποιεί την **rand(...)** για να βρει και να επιστρέψει ένα τυχαίο αριθμό εντός των ορίων που έχουν δοθεί ως ορίσματα.

Για να αντιστοιχίσουμε ένα χρώμα σε κάθε κορυφή της πυραμίδας δημιουργούμε ένα ακόμα attribute για το μοντέλο μας. Συγκεκριμένα, χρησιμοποιούμε την **generateRandomFloat(...)** για να παράγουμε τυχαίους αριθμούς που αντιστοιχούν σε RGB τιμές χρωμάτων και αποθηκεύουμε τα χρώματα αυτά σε ένα πίνακα **g_color_buffer_data** 54 θέσεων. Στη συνέχεια,

πρέπει να "περάσουμε" τα attributes που καθορίσαμε στην OpenGL. Για τον λόγο αυτό δημιουργούμε έναν buffer τύπου **GLuint** για κάθε attribute, **Vertex Buffer** για τις συντεταγμένες και **Color Buffer** για τα χρώματα και καλούμε τις συναρτήσεις **glGenBuffers(…)**, **glBindBuffer(…)** και **glBufferData(…)**. Σημειώνεται πως φορτώνουμε και τους shaders "**ColorFragmentShader.fragmentshader**" και "**TransformVertexShader.vertexshader**" στην αρχή του προγράμματος μέσω της συνάρτησης **LoadShaders(…)**. Επίσης, εφόσον πλέον βρισκόμαστε στον τριδιάστατο χώρο, καλούμε τις συναρτήσεις **glEnable(…)** και **glDepthFunc(…)** στην αρχή του προγράμματος με τα κατάλληλα ορίσματα για να αποτυπώνεται σωστά το βάθος των τριγώνων του μοντέλου μας. Στη συνέχεια, ζωγραφίζουμε τα τρίγωνα μέσα στο do-while loop ώς εξής:

1) Κάνουμε clear το παράθυρο από τυχόν προηγούμενα σχέδια καλώντας την **glClear(…)**
2) Χρησιμοποιούμε τον shader καλώντας την **glUseProgram(…)**
3) Υπολογίζουμε τον πίνακα MVP (Model * View * Projection) με τη βοήθεια συναρτήσεων του αρχείο "controls.cpp" (Το βήμα αυτό εξηγείται στα παρακάτω ερωτήματα καθώς στο παρόν ερώτημα σχεδιάζουμε μόνο μια φαινομενικά στατική εικόνα)
4) Χρησιμοποιούμε τους buffers των attributes του μοντέλου που δημιουργήσαμε παραπάνω καλώντας τις συναρτήσεις **glEnableVertexAttribArray(…)**, **glBindBuffer(…)** και **glVertexAttribPointer(…)** για κάθε attribute αντίστοιχα καθορίζοντας πληροφορίες όπως είναι το μέγεθος ενός τριγώνου.
5) Ζωγραφίζουμε τα τρίγωνα με την συνάρτηση **glDrawArrays(…)** όπου καθορίζουμε και τον αριθμό των vertices στον πίνακα μας που θέλουμε να ζωγραφιστούν. Στην περίπτωση μας έχουμε 6 τρίγωνα με το καθένα να αποτελείται απο 3 σημεία.
6) Τέλος, κάνουμε swap τους buffers με την συνάρτηση **glfwSwapBuffers(…)** έτσι ώστε να εμφανιστεί το σχέδιο στο παράθυρο μας.

Ερώτημα (iii) – Στατική κάμερα

Οι λειτουργίες της κάμερας υλοποιούνται εξ ολοκλήρου στο αρχείο "controls.cpp". Στο τέλος οι πίνακες Model, View και Projection περνάνε στην main του αρχείου "Main.cpp" έτσι ώστε να δημιουργηθεί ο πίνακας MVP και να αποδωθεί η σκήνη στο παράθυρο μας. Για να καθορίσουμε το αρχικό σημείο που θα είναι η κάμερα χρησιμοποιούμε μία μεταβλητή τύπου vec3 με το επιθυμητό σημείο (10, 10, 30) Για να υπολογίσουμε το κέντρο της πυραμίδας υλοποιήσαμε την συνάρτηση **setModelCenter(…)** στο αρχείο "controls.cpp" η οποία καλείται στη main για να καθορίσουμε το σημείο του κέντρου της πυραμίδας. Γνωρίζουμε πως η πυραμίδα δεν μετακινείται στους άξονες x και y. Επομένως, με δεδομένο ότι η κορυφή της πυραμίδας βρισκεται στον άξονα z και ότι το ύψος της πυραμίδας παράγεται τυχαία, το κέντρο της πυραμίδας θα βρίσκεται κατά προσέγγιση στο σημείο (0, 0, ύψος / 2). Το κέντρο της πυραμίδας αποθηκέυεται σε ακόμη μία μεταβλητή τύπου vec3. Ομοίως, καθορίζουμε και το ανιόν διάνυσμα της κάμερας. Έπειτα, στο τέλος της συνάρτησης **computeMatricesFromInputs()** η

οποία καλείται επαναληπτικά μέσα στο loop του render, καλούμε την συνάρτηση **lookAt(...)** με ορίσματα τα παραπάνω διανύσματα, έτσι ώστε να ενημερωθεί η θέση και το σημείο που κοιτάει η κάμερα. Η **lookAt(...)** μας επιστρέφει τον πίνακα View. Επιπλέον, στην συνάρτηση **computeMatricesFromInputs()** του αρχείου "controls.cpp" δημιουργούμε και τον πίνακα του Projection, όπου καθορίζονται το οπτικό πεδίο, η αναλογία απεικόνισης (aspect ratio) και το πόσο κοντά και μακριά θα βλέπει η κάμερα. Στη συνέχεια, οι δύο αυτοί πίνακες (μαζί με τον πίνακα Model) περνάνε στο render loop για να υπολογιστει ο MVP πίνακας.

Ερώτημα (iv) – Κλιμάκωση και σμίκρυνση πυραμίδας

Η κλιμάκωση/σμίκρυνση της πυραμίδας έχει υλοποιηθεί στη συνάρτηση **computeMatricesFromInputs()** στο αρχείο "controls.cpp". Όπως με τους πίνακες Projection και View, διατηρούμε και τον πίνακα Model στο αρχείο "controls.cpp" και το περνάμε στο render loop της main για να υπολογιστεί ο πίνακας MVP. Εσωτερικά της συνάρτησης **computeMatricesFromInputs()**, με τις κατάλληλες κλήσεις των συναρτήσεων **glfwGetKey(...)** και **isCapitalKeyPressed(...)** ελέγχουμε εάν κάποιο από τα κουμπία **<q>** ή **<z> (μικρά)** έχουν πατηθεί. Σε περίπτωση που κάποια από τα κουμπία αυτά έχει πατηθεί, καλούμε την συνάρτηση scale της βιβλιοθήκες glm, η οποία πολλαπλασιάζει τον πίνακα Model που δίνεται ως όρισμα με το διάνυσμα που δίνεται ως όρισμα για να πετύχουμε το επιθυμητό scaling. Στην περίπτωση μας επιθυμούμε ομοιόμορφο scaling και στους 3 άξονες, επομένως δίνουμε την τιμή 1.03 στα x y και z για το scale up και 0.97 αντίστοιχα για το scale down. Σημειώνεται πως παρά το scaling, το κέντρο της πυραμίδας στο οποίο κοιτά η κάμερα παραμένει ίδιο με αυτό του αρχικού μοντέλου.

Στο σημείο αυτό υλοποιήσαμε και ένα επιπλέον μετασχηματισμό, πέρα από τα απαιτούμενα της άσκησης. Με το πλήκτρο **R** (κεφαλαίο ή μικρό) το μοντέλο της πυραμίδας πραγματοποιεί περιστροφή γύρω από τον άξονα z, δηλαδή γύρω από τον εαυτό του εφόσον είναι κεντραρισμένο στην αρχή των αξόνων. Η περιστροφή γίνεται με την συνάρτηση **rotate(...)** της βιβλιοθήκης glm.

Ερώτημα (v) – Δυναμική κάμερα

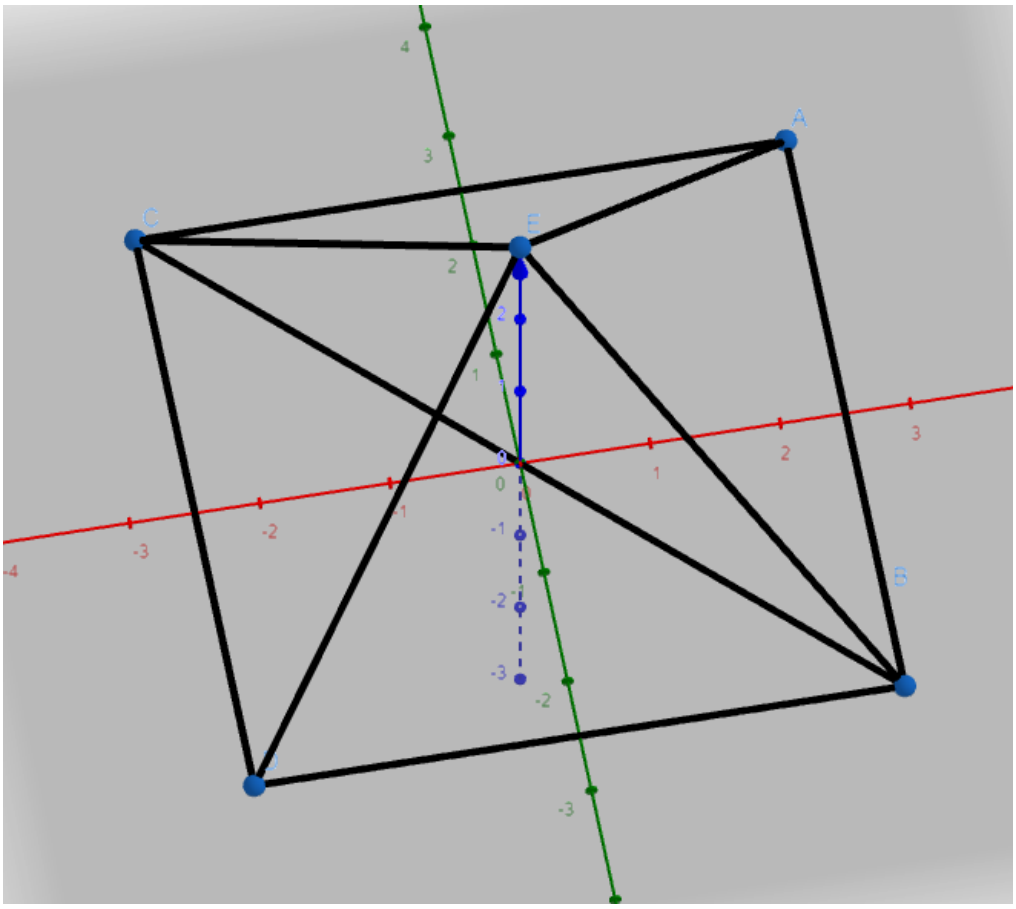
Όπως και στο προηγούμενο ερώτημα, οι κινήσεις της κάμερας έχουν υλοποιηθεί στη συνάρτηση **computeMatricesFromInputs()** στο αρχείο "controls.cpp". Με τα πλήκτρα **<W>** και **<X>** (κεφαλαία ή μικρά) η κάμερα γυρίζει γύρω από τον άξονα x. Η κίνηση έχει υλοποιηθεί περιστρέφοντας την κάμερα χωρίς να την μετατοπίσουμε. Η περιστροφή έχει υλοποιηθεί με την συνάρτηση **rotateX(...)** της υποβιβλιοθήκης **gtx** της **glm**, η οποία περιστρέφει ενα vec3 (το σημείο της κάμερας) γύρω από τον άξονα x με βάση μία γωνία που δίνεται ως όρισμα. Συγκεκριμένα, χρειάστηκε να κάνουμε include το header <**glm/gtx/rotate_vector.hpp**>. Με τα

πλήκτρα **<A>** και **<D>** (κεφαλαία ή μικρά) η κάμερα γυρίζει γύρω από τον άξονα y. Η κίνηση έχει υλοποιηθεί όπως και η κίνηση γύρω από τον άξονα x. Με τα πλήκτρα **<–>** και **<=>** (μικρά) ή **<_>** και **<+>** κεφαλαία πραγματοποιείται zoom in και zoom out αντίστοιχα προς/από το κέντρο της πυραμίδας. Η λειτουργία έχει υλοποιηθεί βρίσκοντας το κανονικοποιημένο διάνυσμα της διαφοράς της θέσης της κάμερας από το κέντρο της πυραμίδας. Και έπειτα μετατοπίζοντας την θέση της κάμερας κατά το διάνυσμα αυτό πολλαπλασιασμένο με τον καθορισμένο συντελεστή της ταχύτητας της μετατόπισης. Σημειώνεται πως σε όλους τους μετασχηματισμούς η κάμερα εξακολουθεί να κοιτά στο κέντρο της πυραμίδας.

Επιπλέον, πέρα από τα απαιτούμενα της άσκησης, με τα πλήκτρα **<O>** και **<P>** (κεφαλαία ή μικρά) πραγματοποιείται zoom in και zoom out αντίστοιχα. Σε αντίθεση με την παραπάνω λειτουργία το zoom σε αυτή την περίπτωση επιτυγχάνεται, μεταβάλλωντας το Field of View (οπτικό πεδίο) του Projection Matrix. Με τα πλήκτρα **<K>** και **<L>** (κεφαλαία ή μικρά) η κάμερα κινείται παράλληλα στον άξονα z. Η κίνηση έχει υλοποιηθεί με μία απλή πρόσθεση/αφαίρεση στο z της θέσης της κάμερας. Τέλος, με το πλήκτρο **<1>** (μικρό) ή **<!>** (κεφαλαίο) η κάμερα επανέρχεται στην αρχική της θέση.

Ερώτημα (vi)

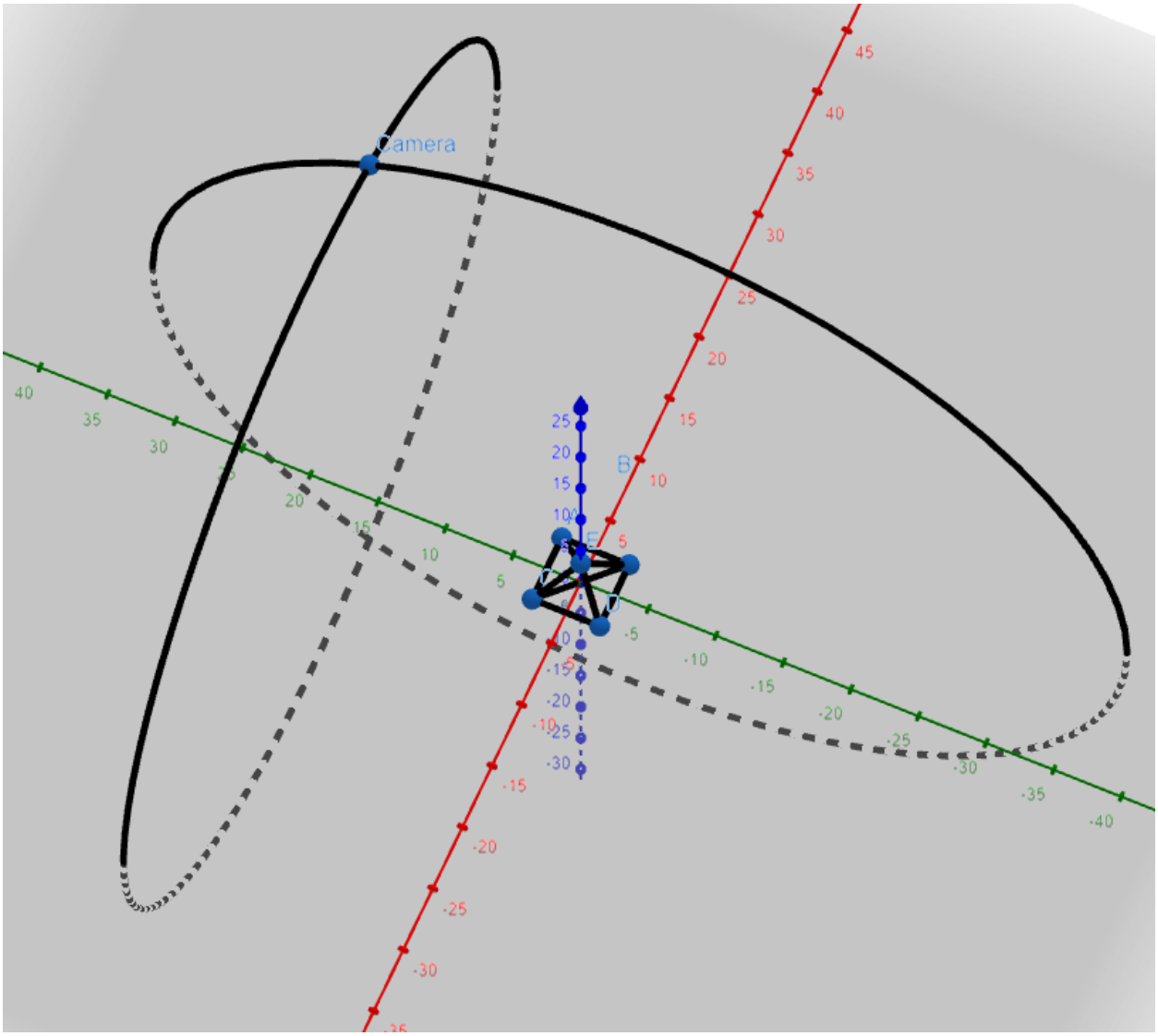
Το ερώτημα αυτό αναφέρεται στην δημιουργία της παρούσας αναφοράς καθώς και στον υπολογισμό των συντεταγμένων του μοντέλου της πυραμίδας. Ο υπολογισμός έχει γίνει με τη βοήθεια ενός 3D calculator. Αρχικά, σχεδιάστηκαν τα σημεία των τριγώνων της πυραμίδας ως εξής:



Τρίγωνα πυραμίδας

Θεωρούμε πως ο κόκκινος άξονας είναι ο άξονας x, ο πράσινος είναι ο άξονας y και ο μπλε είναι ο άξονας z. Τα σημεία A (2.5, 2.5, 0), B(2.5, -2.5, 0), C(-2.5, 2.5, 0) και B(2.5, -2.5, 0), C(-2.5, 2.5, 0), D(-2.5, -2.5, 0) σχηματίζουν τα δύο τρίγωνα της βάσης του μοντέλου. Το σημείο E(0, 0, H) είναι η κορυφή της πυραμίδας, όπου H είναι το ύψος της πυραμίδας και ισχύει 2.0<=H<=10.0. Οι τριγωνικές πλευρές της πυραμίδας σχηματίζονται όπως φαίνεται στο σχήμα ως εξής: ABE, AEC, ECD, EBD.

Επιπλέον, σχεδιάστηκαν ενδεικτικά και οι κινήσεις της κάμερας γύρω από τους άξονες x και y από την αρχική θέση της κάμερας (10, 10, 30). Φυσικά, οι κινήσεις της κάμερας γύρω από τους άξονες μπορεί να εντελώς διαφορετικές από αυτές που φαίνονται στο στιγμιότυπο.



Ενδεικτική κίνηση κάμερας