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

Εξάσκηση στη χρήση βασικών βιβλιοθηκών γραφικών της OpenGL 3.3

Μέλη ομάδας:

Αντωνίου Χριστόδουλος AM: 2641
Τσιούρη Αγγελική AM: 3354

Εισαγωγική σημείωση

Όπως και στις δύο προηγούμενες ασκήσεις, αρχικά, μελετήσαμε τα εργαστήρια του μαθήματος μαζί με τον ενδεικτικό κώδικα που μας δόθηκε για να κατανοήσουμε τις έννοιες που απαιτεί η άσκηση. Στη συνέχεια με τη βοήθεια του ενδεικτικού κώδικα, κάναμε προσθήκες και τροποποιήσεις στον κώδικα μας από την δεύτερη άσκηση για να πετύχουμε τα ζητούμενα του κάθε ερωτήματος. Σημειώνεται πως εφόσον έχουμε να κάνουμε με φόρτωση πολλαπλών objects και textures, έχουμε κάνει refactor τον κώδικα με σκόπο τον περιορισμό του duplicate code και την διευκόλυνση ανάγνωσης του. Επαναλαμβανόμενες λειτουργίες όπως η φόρτωση ενός object ή texture ή η εφαρμογή ενός attribute κατά το rendering έχουν γίνει extract σε αντίστοιχα functions. Επιπλέον, έχουν εξαληφθεί τα scope resolution operators glm και std (εφόσον χρησιμοποιούμε τα αντίστοιχα namespaces) έτσι ώστε ο κώδικας να είναι πιο καθαρός.

Η άσκηση έχει αναπτυχθεί σε Windows 10 με Visual Studio 2019. Το παραδοτέο .zip αρχείο περιλαμβάνει τα εξής αρχεία:

- "Main.cpp", "controls.cpp", "controls.hpp", "stb_image.h" και "objloader.hpp" με τον κώδικα της άσκησης.
- "TextureFragmentShader.fragmentshader" και "TransformVertexShader.vertexshader" με τους shaders.
- "sun.obj", "planet.obj", "meteor.obj" όπου βρίσκονται τα objects προς φόρτωση.
- "sun.jpg", "planet.jpg". neptune.jpg", "meteor.jpg", "mars.jpg" και "earth.jpg" με τις εικόνες των textures.

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, "Ηλιακό Σύστημα". 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.

Όπως και στις προηγούμες δύο ασκήσεις, για το background του παραθύρου καλούμε την συνάρτηση **glClearColor(...)**, με παραμέτρους: red=0.04, green=0.02, blue=0.08, alpha=1, οι οποίες αντιστοιχούν σε μία απόχρωση μαύρου 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, 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) – Φόρτωση σφαίρας και texture ηλίου και υλοποίηση κάμερας

Για να φορτώσουμε την σφαίρα του ηλίου (ή ένα από τα άλλα objects της άσκησης) χρειαζόμαστε εναν **buffer** για τα **vertices** (sunVertexBuffer) και έναν ακόμη για τις **UV συντεταγμένες** (sunUVBuffer). Επιπλέον, χρειαζόμαστε **λίστες** για τα **vertices** του object (sunVertices), για τα **normals** (sunNormals) και για τις **UV συντεταγμένες** (sunUVs). Τα παραπάνω περνάνε ως ορίσματα στη συνάρτηση **loadObject(...)** μάζι με το όνομα του αρχείου .**obj** που περιέχει το object. Η συνάρτηση **loadObject(...)** είναι υπέυθυνη για την φόρτωση ενός object στους παραπάνω buffers και λίστες. Με την σειρά της καλεί την **loadOBJ(...)** η οποία διαβάζει το αρχείο .obj και περνάει τα vertices, τα normal και τις UV συντεταγμένες του object στις αντίστοιχες λίστες. Στη συνέχεια, η **loadObject(...)** δημιουργεί τους δύο buffers ως εξής: Καλεί την **glGenBuffers(...)** και έπειτα την **glBindBuffer(...)** για να επιλέξει τον νέο buffer και τέλος την **glBufferData(...)** για να αρχικοποιήσει τα δεδομένα του buffer. Αυτό γίνεται ξεχωριστά για κάθε έναν από τους buffers.

Για να φορτώσουμε το texture του ηλίου (ή ένα από τα άλλα textures της άσκησης) αρχικά δηλώνουμε έναν πίνακα τύπου **GLuint** με κάθε θέση του πίνακα να αντιστοιχεί σε ένα διαφορετικό texture. Καλούμε την **glGenTextures(...)** με ορίσματα τον παραπάνω πίνακα και το μέγεθος του. Φυσικά, το παραπάνω γίνεται μία φορα και όχι ξεχωριστά για κάθε texture. Στη συνέχεια καλούμε την **createTexture(...)** για κάθε texture που θέλουμε να παράγουμε ξεχωριστά. Η **createTexture(...)** δέχεται ως όρισματα το **όνομα του αρχείου .jpg** που περιέχει την εικόνα του texture, μία **σταθερά τύπου GL_TEXTUREX** (όπου Χ είναι ο αύξοντας αριθμός του texture) καθώς και την **θέση του πίνακα textureID** που θέλουμε να έχουμε το νέο texture. Με την σειρά της καλεί την **stbi_load(...)** για να διαβάσει το αρχείο **.jpg** και στη συνέχεια ενεργοποιεί το texture με την **glActiveTexture(...)** με όρισμα την παραπάνω σταθερά. Επιλέγει το texture καλώντας την **glBindTexture(...)**. Καθορίζει της παραμέτρους του texture καλώντας την **glTexParameteri(...)**. Σημειώνεται πως εδώ δίνουμε ως τελευταίο όρισμα την σταθερά **GL_LINEAR** το οποίο έχει ως αποτέλεσμα να γίνεται ελαφρώς smoothing του texture. Τέλος, περνάμε την εικόνα του texture στην OpenGL καλώντας την συνάρτηση **glTexImage2D(...)**.

Οι λειτουργίες της κάμερας υλοποιούνται εξ' ολοκλήρου στο αρχείο "controls.cpp" όπου διατηρούνται και οι πίνακες Model, View και Perspective. Ο πίνακας **Perspective** δημιουργείται μία φορά στην αρχή του αρχείου, ενώ για κάθε μοντέλο διατηρείται ένας ξεχωριστος πίνακας **Model** (π.χ. SunModel για τον ήλιο). Ο πίνακας **View** δημιουργείται επαναληπτικά μέσα στη συνάρτηση **computeMatrices(...)** η οποία είναι υπέυθυνη για οποιοδήποτε μετασχηματισμό πίνακα κατά την εκτέλεση του προγράμματος και καλείται μέσα στο **loop του render** στη "Main.cpp". Συγκεκριμένα, ο πίνακας View δημιουργείται καλώντας την συνάρτηση **lookAt(...)** με ορίσματα, ένα διάνυσμα που καθορίζει την **θέση της κάμερας**, ένα διάνυσμα που καθορίζει το **σημείο που θα κοιτάει** η κάμερα και ένα ακόμη διάνυσμα που καθορίζει το **ανιόν διάνυσμα**. Η θέση της κάμερας ενημερώνεται και αυτή μέσα στη **computeMatrices(...)** και πως η κάμερα κοιτάει πάντα στο κέντρο του ηλίου, δηλαδή την αρχή των αξόνων. Σημειώνεται,

πως έχουμε αλλάξει την αρχική θέση της κάμερας αλλά και το μέγιστο display range κατά τη δημιουργία του πίνακα Projection (από 100 σε 400) έτσι ώστα να φαίνονται όλα τα μοντέλα στο παράθυρο μας. Επίσης το ανιόν διάνυσμα, έχει αλλάξει από (0, 0, 1) στην προηγούμενη άσκηση σε (0, 1, 0).

Όσον αφορά τις κινήσεις της κάμερας, με τα πλήκτρα **<W>** και **<X>** (κεφαλαία ή μικρά) η κάμερα γυρίζει γύρω από τον άξονα x. Η κίνηση έχει υλοποιηθεί περιστρέφοντας την θέση της κάμερας χωρίς να την μετατοπίσουμε. Η περιστροφή έχει υλοποιηθεί με την συνάρτηση **rotateX(...)** της υποβιβλιοθήκης **gtx** της **glm**, η οποία περιστρέφει ενα vec3 (θέση της κάμερας) γύρω από τον άξονα x με βάση μία γωνία που δίνεται ως όρισμα. 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 του ηλίου. Και έπειτα μετατοπίζοντας την θέση της κάμερας κατά το διάνυσμα αυτό πολλαπλασιασμένο με τον καθορισμένο συντελεστή της ταχύτητας της μετατόπισης.

Στο σημείο αυτό, έχοντας φορτώσει με επιτυχία το object και το texture του ηλίου δεν μένει παρά να τα ζωγραφίσουμε στην οθόνη. Σύμφωνα με την εκφώνηση, θέλουμε το κέντρο του ηλίου να βρίσκεται στην αρχή των αξόνων (0, 0, 0), επομένως δεν χρειάζεται κάποιος μετασχηματισμός. Το πρώτο που χρειαζόμαστε είναι ένα **Vertex Array Object** το οποίο και δημιουργούμε στην αρχή της main στο αρχείο “Main.cpp”. Έπειτα φορτώνουμε τα αρχεία των shaders με την βοήθεια της συνάρτησης **LoadShaders(...)** και καλούμε την συνάρτηση **glGetUniformLocation(...)** για να παρούμε την θέση του uniform variable του πίνακα MVP στον vertex shader.

Έπειτα περνάμε στο **render loop**, όπου αρχικά καλείται η **computeMatrices(...)** η οποία υπολογίζει του πίνακες Models των μοντέλων και View για την κάμερα. Για να σχεδιάσουμε το μοντέλο του ηλίου (ή ένα από τα άλλα μοντέλα της άσκησης) παίρνουμε τους πίνακες **Model**, **View** και **Projection** από το αρχείο “controls.cpp” με τους κατάλληλους getters. Σημειώνεται πως αρκεί να πάρουμε τους πίνακες View και Projection μία φορά σε κάθε επανάληψη καθώς αυτοί μένουν σταθεροί σε μία επανάληψη του render. Ωστόσο, για κάθε μοντέλο πρέπει να πάρουμε τον πίνακα Model που του αντιστοιχεί. Στη συνέχεια, υπολογίζουμε τον πίνακα MVP και καλούμε την **glUniformMatrix4fv(...)** για να στείλουμε τον μετασχηματισμό μας στον vertex shader. Η επιλογή του επιθυμητού texture γίνεται με την βοήθεια της συνάρτησης **selectTexture(...**) η οποία εσωτερικά επιλέγει το texture που δέχθηκε ως όρισμα με την συνάρτηση **glBindTexture(...)** και ειδοποιεί τον fragment shader να το χρησιμοποιήσει με την συνάρτηση **glUniform1(...)**. Για να εφαρμόσουμε ένα attribute έχει δημιουργηθεί η βοηθητική συνάρτηση **applyAttribute(...)** η οποία δέχεται ως ορίσματα τον αύξοντα αριθμό, τον buffer

και το μέγεθος του attribute. Εσωτερικά, ενεργοποιέι το attribute με την **glEnableVertexAttribArray(...)**, επιλέγει τον buffer με την **glBindBuffer(...)** και το εφαρμόζει με την **glVertexAttribPointer(...)**. Επομένως, καλούμε την **applyAttribute(...)**, μία φορά για τον buffer των vertices και ακόμη μία για τον buffer των UV συντεταγμένων. Τέλος, ζωγραφίζουμε τα τρίγωνα με την **glDrawArrays(...)**. Και αφού έχουμε ζωγραφίσει το μόντελο (και άλλα πιθανά μοντέλα που θα προστεθούν παρακάτω) καλούμε την **glfwSwapBuffers(...)** για να εμφανιστούν οι αλλαγές στο παράθυρο μας.

Ερώτημα (iii) – Σφαίρα, texture και μετασχηματισμοί πλανήτη

Η φόρτωση της σφαίρας και του texture του πλανήτη έχουν υλοποιηθεί όπως και η φόρτωση της σφαίρας και του texture του ηλίου αντίστοιχα. Πιο συγκεκριμένα, για να φορτώσουμε το object της σφαίρας του πλανήτη, **δημιουργούμε καινούργιους buffers για τα vertices και για τις UV συντεταγμένες** και καλούμε την **loadObject(...)** όπου δίνουμε και το όνομα του αντίστοιχου .obj αρχείου. Η διαδικασία που ακολουθεί είναι ίδια με αυτή που περιγράφεται παραπάνω για την φόρτωση της σφαίρας του ηλίου. Για να φορτώσουμε το texture του πλανήτη **αυξάνουμε το μέγεθος του πίνακα που περιέχει τα texture IDs κατά ένα**. Ομοίως στην κλήση της **glGenTextures(...)** αυξάνουμε το μέγεθος που δίνουμε ως όρισμα κατά ένα. Τέλος, καλούμε την **createTexture(...)** άλλη μία φορά δίνοντας ως ορίσματα το **νέο όνομα του αρχείου .jpg** που περιέχει την εικόνα του texture, την **σταθερά τύπου GL_TEXTUREX** (όπου το X είναι αυξημένο κατά ένα) καθώς και την **επόμενη θέση του πίνακα textureID.**

Αυτό που διαφέρει από το μοντέλο του ηλίου είναι πως θέλουμε το κέντρο του πλανήτη αρχικά να βρίσκεται στο σημείο (25, 0 ,0) στο σύστημα παγκόσμιων συντεταγμένων. Αυτό επιτυγχάνεται στη συνάρτηση **setInitialMatrices()** στο αρχείο "controls.cpp" όπου μετατοπίζουμε τον πλανήτη στο επιθυμητό αρχικό σημείο, πολλαπλασιάζοντας τον πίνακα Model του πλανήτη (PlanetModel) με το σημείο αυτό. Με δεδομένο ότι ο πλανήτης αρχικά βρίσκεται στην αρχή των αξόνων, έχουμε μετατόπιση στο επιθυμητό σημείο. Η συνάρτηση **setInitialMatrices()** καλείται ακριβώς πριν από το render loop στο αρχείο "Main.cpp".

Επιπλέον, θέλουμε ο πλανήτης να κινείται σε κυκλική τροχιά και με σταθερή ταχύτητα γύρω από το κέντρο του ήλιου, δηλαδή το (0, 0, 0). Αυτό επιτυγχάνεται στην συνάρτηση **transformPlanet()** στο αρχείο "controls.cpp" όπου ο πίνακας Model του πλανήτη πολλαπλασιάζεται με το δίανυσμα (-25, 0, 0) έτσι ώστε να μετατοπιστεί στην αρχή των αξόνων, έπειτα περιστρέφεται κάθε φορά με την ίδια σταθερή γωνία (planetMoveSpeed) γύρω από τον άξονα y και τέλος πολλαπλασιάζεται πάλι με το διάνυσμα (25, 0, 0) έτσι ώστε να μετατοπιστεί κοντά στο σημείο που βρισκόταν. Η συνάρτηση **transformPlanet()** καλείται στην **computeMatrices()** η οποία με την σειρά της καλείται μέσα στο render loop. Το αποτέλεμσα είναι η περιστροφή του πλανήτη γύρω από το κέντρο των αξόνων. Ο ίδιος μετασχηματισμός περιστροφής εφαρμόζεται και στο διάνυσμα της θέσης του κέντρου του πλανήτη. Η ενημέρωση του κέντρου του πλανήτη μας χρειάζεται για την ανίχνευση σύγκρουσης στο επόμενο ερώτημα.

Ερώτημα (iv) – Σύγκρουση μετεωρίτη με τον πλανήτη ή τον ήλιο

Η φόρτωση της σφαίρας και του texture του μετεωρίτη έχουν υλοποιηθεί όπως και τα δύο προηγούμενα μοντέλα και άρα θα ήταν περιττό να τα περιγράψουμε ξανά. Αυτό που διαφέρει είναι η "εκτόξευση του μετεωρίτη" από την θέση του παρατηρητή καθώς και η πιθανή σύγκρουση με κάποιο από τα άλλα μοντέλα στο χώρο.

Για την εκτόξευση (ή αλλιως δημιουργία) του μετεωρίτη από την θέση του παρατηρητή χρειαζόμαστε την **θέση της κάμερας**, η οποία ενημερώνεται μέσα στη συνάρτηση **computeMatrices()** στο αρχείο "controls.cpp". Επιπλέον, χρειαζόμαστε μεταβλητές τύπου boolean για να κρατάμε πληροφορία για το **εάν υπάρχει ενεργός μετεωρίτης** στη σκηνή (isMeteorActive), πληροφορία για το **εάν ο μετεωρίτης συγκρούστηκε με τον πλανήτη** (collidedWithPlanet) και πληροφορία για το **εάν ο μετεωρίτης συγκρούστηκε με τον ήλιο** (collidedWithSun). Οι μεταβλητές αυτές αρχικοποιούνται στην τιμή false. Ακόμη, χρειαζόμαστε μεταβλητές με τα **διανύσματα των θέσεων των κέντρων** των μοντέλων αλλά και μεταβλητές με τις **ακτίνες των σφαιρών** των μοντέλων.

Στη συνάρτηση **computeMatrices()** ελέγχεται αρχικά εάν ο χρήστης έχει πατήσει το κουμπί **<spacebar>** και στην περίπτωση αυτή, εφόσον δεν υπάρχει ήδη ενεργός μετεωρίτης στην σκηνή καλείται η συνάρτηση **spawnMeteor()** που είναι υπεύθυνη για τη δημιουργία ενός μετεωρίτη. Δηλαδή, κάθε χρονική στιγμή μπορούμε να έχουμε το πολύ ένα μετεωρίτη στη σκηνή. Όταν ο υπάρχον μετεωρίτης εξαφανιστεί έπειτα από κάποια σύγκρουση, ο χρήστης μπορεί να πατήσει ξανά το **<spacebar>** για να δημιουργήσει νέο μετεωρίτη. Εσωτερικά της συνάρτησης **spawnMeteor()** κάνουμε: 1) reset τον πίνακα Model του μετεωρίτη (MeteorModel) σε περίπτωση που δεν πρόκειται για τον πρώτο μετεωρίτη που δημιουργήθηκε, 2) δίνουμε την τιμή true στη boolean μεταβλητή που κρατάει πληροφορία για ενεργό μετεωρίτη, 3) φτιάχνουμε ένα πίνακα μετατόπισης με βάση την τρέχουσα θέση της κάμερας και πολλαπλασιάζουμε με αυτόν τον πίνακα Model του πλανήτη αλλά και το διάνυσμα της θέσης του πλανήτη. Αυτό έχει ως αποτέλεσμα ο μετεωρίτης να μετατοπιστεί στην τρέχουσα θέση της κάμερας.

Στη συνέχεια της **computeMatrices()** έλεγχεται εάν υπάρχει ενεργός μετεωρίτης στη σκηνή και στην περίπτωση που υπάρχει καλούντε η συνάρτηση **transformMeteor()** που είναι υπεύθυνη για την κίνηση του μετεωρίτη προς το κέντρο του ήλιου και η συνάρτηση **checkForCollisions()** που είναι υπεύθυνη για τον έλεγχο πιθανών συγκρούσεων. Η κίνηση του μετεωρίτη στην **transformMeteor()** έχει υλοποιηθεί με παρόμοια λογική με το zoom in/zoom out της κάμερας. Συγκεκριμένα, βρίσκουμε το κανονικοποιημένο διάνυσμα της διαφοράς της θέσης του μετεωρίτη από το κέντρο του ηλίου. Έπειτα μετατοπίζουμε την θέση του μετεωρίτη κατά το διάνυσμα αυτό πολλαπλασιασμένο με τον καθορισμένο συντελεστή της ταχύτητας της μετατόπισης. Έτσι, έχουμε βρει την νέα θέση του μετεωρίτη. Για να υπολογίσουμε και τον νέο πίνακα Model του μετεωρίτη, υπολογίζουμε την διαφορά της προηγούμενης θέσης του από την επόμενη και με βάση αυτή την διαφορά φτιάχνουμε έναν πίνακα μετατόπισης, ο οποίος

πολλαπλασιάζεται με τον πίνακα Model. Το αποτέλεμσα είναι η μετατόπιση του μετεωρίτη προς το κέντρο του ηλίου. Η συνάρτηση **checkForCollisions()** ελέγχει για πιθανές συγκρούσεις και ενημερώνει τις αντίστοιχες Boolean μεταβλητές. Για τον έλεγχο των συγκρούσεων έχει υλοποιηθεί η βοηθητική συνάρτηση **detectSphereCollision()** η οποία δέχεται ως ορίσματα τα σημεία των κέντρων δύο σφαιρών καθώς και τις ακτίνες τους και ελέγχει εάν η απόσταση των κέντρων είναι μικρότερη από το άθροισμα των ακτίνων για να ανιχνέυση εάν έχουμε σύγκρουση. Σημειώνεται πως στην περίπτωση του ηλίου έχει δοθεί εσκεμμένα μικρότερη ακτίνα από την κανονική ακτίνα του μοντέλου έτσι ώστε να καθυστερείται η ανίχνευση της σύγκρουσης με τον μετεωρίτη και να δίνεται η ψευδαίσθηση πως ο μετεωρίτης χάνεται μέσα στον ήλιο. Τέλος, μετά την κλήση της **computeMatrices()** οι boolean μεταβλητές περνάνε στο render loop με τους κατάλληλους getters για να ελέγξουμε ποια μοντέλα θα πρέπει να σχεδιαστούν στο τρέχον frame.

Bonus Ερωτήματα

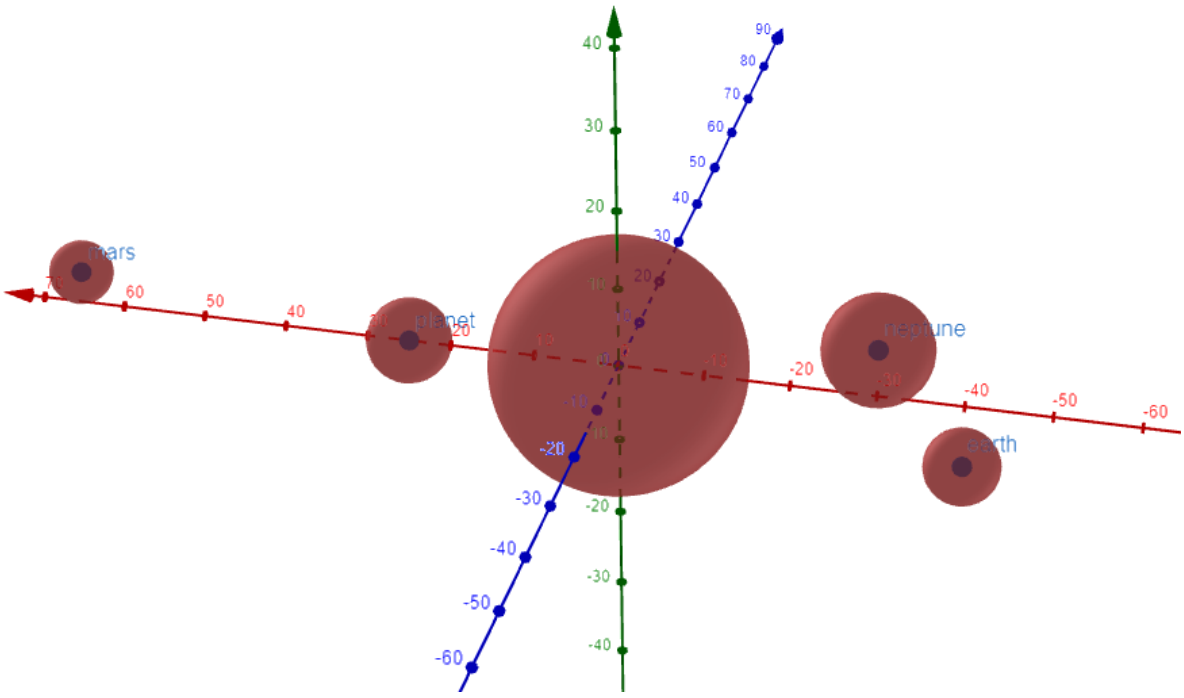
Όσον αφορά τα bonus ερωτήματα έχουν υλοποιηθεί τα εξής:

- Προσθήκη των πλανητών Neptune, Earth και Mars με την αντίστοιχη φόρτωση των textures. Τα μοντέλα βασίζονται στην υπάρχουσα σφαίρα του πρώτου πλανήτη στην οποία έχει εφαρμοστεί scaling. Κάθε πλανήτης έχει διαφορετικό αρχικό σημείο και διαφορετική απόσταση από τον ήλιο, έτσι ώστε οι πλανήτες να μην συναντιούνται μεταξύ τους. Η σύγκρουση με τον μετεωρίτη έχει υλοποιηθεί και για τους νέους πλανήτες.
- Με τα πλήκτρα <u> και <p> (κεφαλαία ή μικρα) αυξομειώνονται οι ταχύτητες περιστροφής των πλανητών. Η αυξομείωση είναι +/- 5% της τρέχουσας ταχύτητας σε κάθε frame που ένα από τα παραπάνω πλήκτρα είναι πατημένο.

Επιπλέον, πέρα από τα ζητούμενα της άσκησης έχει υλοποιηθεί και κίνηση του ηλίου γύρω από τον εαυτό του. Με το πλήκτρο <1> η κάμερα επιστρέφει στην αρχική της θέσης και με το πλήκτρο <2> επαναφέρονται οι πλανήτες που έχουν εξφανιστεί (δηλαδή, που δεν ζωγραφίζονται).

Ερώτημα (v)

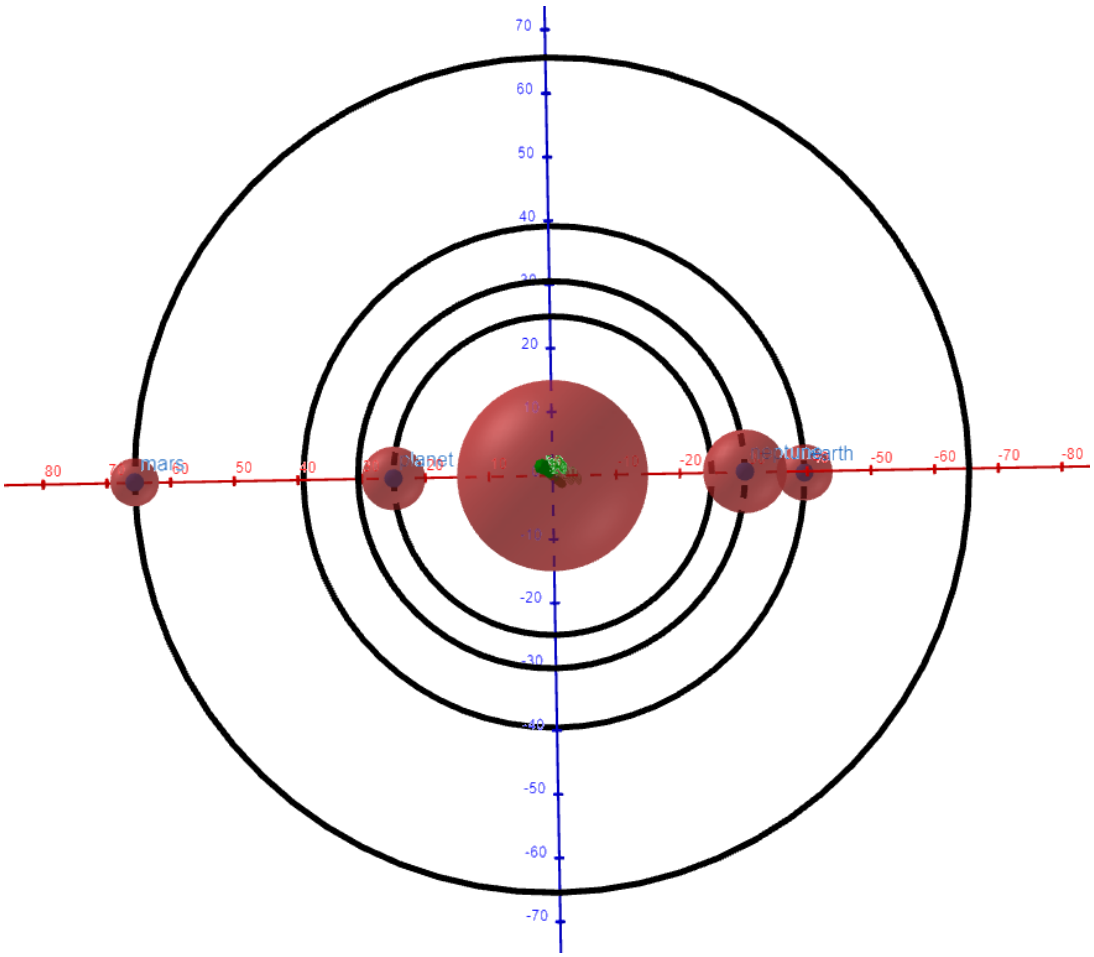
Το ερώτημα αυτό αναφέρεται στην δημιουργία της παρούσας αναφοράς καθώς και στον υπολογισμό των συντεταγμένων των μοντέλων της σκηνής. Ο υπολογισμός έχει γίνει με τη βοήθεια ενός 3D calculator. Αρχικά, σχεδιάστηκαν οι σφαίρες και τα κέντρα του ηλίου και των πλανητών στις αρχικές τους θέσεις.



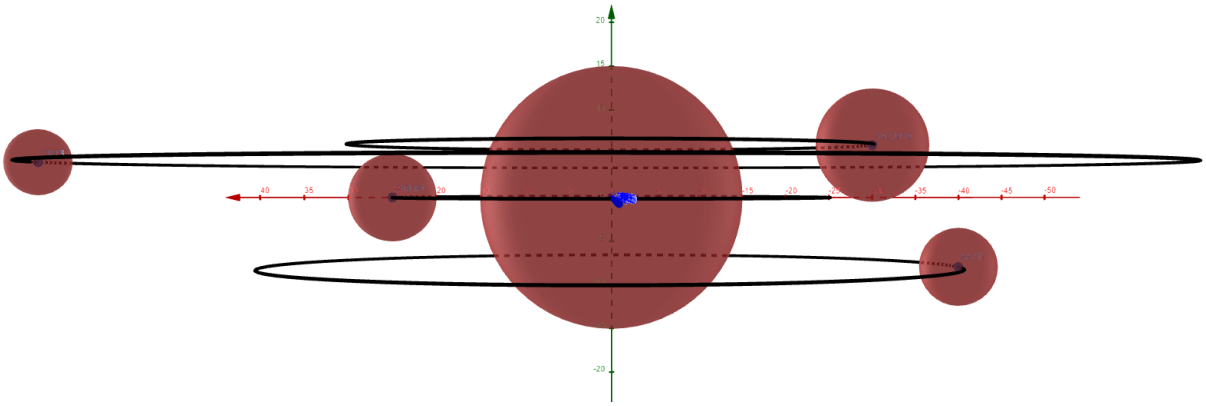
Ήλιος και πλανήτες στις αρχικές τους θέσεις

Θεωρούμε πως ο κόκκινος άξονας είναι ο άξοβας x, ο πράσινος είναι ο y και ο μπλε είναι ο z. Ο ήλιος έχει ακτίνα 15 και το κέντρο του βρίσκεται στο σημείο A(0, 0, 0) όπου και παραμένει. Ο πλανήτης Planet έχει ακτίνα 5 και το κέντρο του βρίσκεται στο σημείο B(25, 0, 0). Ο πλανήτης Neptune βασίζεται στο ίδιο μοντέλο με τον πλανήτη Planet αλλά έχει γίνει μεγένθυση κατά 30%. Επομένως, έχει ακτίνα στο σημείο 6.5 ενώ το κέντρο του βρίσκεται στο σημείο C(-30, 6, 0). Ο Πλανήτης Earth βασίζεται επίσης στο μοντέλο του Planet με μία σμίκρυνση κατά 10%. Έχει ακτίνα 4.5 και το κέντρο του βρίσκεται στο σημείο D(-40, -8, 0). Ομοίως, ο πλανήτης Mars έχει υποστεί σμίκρυνση 25%, έχει ακτίνα 3.75 και το κέντρο του βρίσκεται στο σημείο E(65, 4, 0).

Οι πλανήτες κινούνται με κυκλική τροχιά γύρω από τον άξονα y. Έχουν διαφορετική απόσταση από τον άξονα y και είναι ελαφρώς μετατοπισμένοι και στον άξονα y, έτσι ώστε να μην συναντιούνται μεταξύ τους καθώς κινούνται.

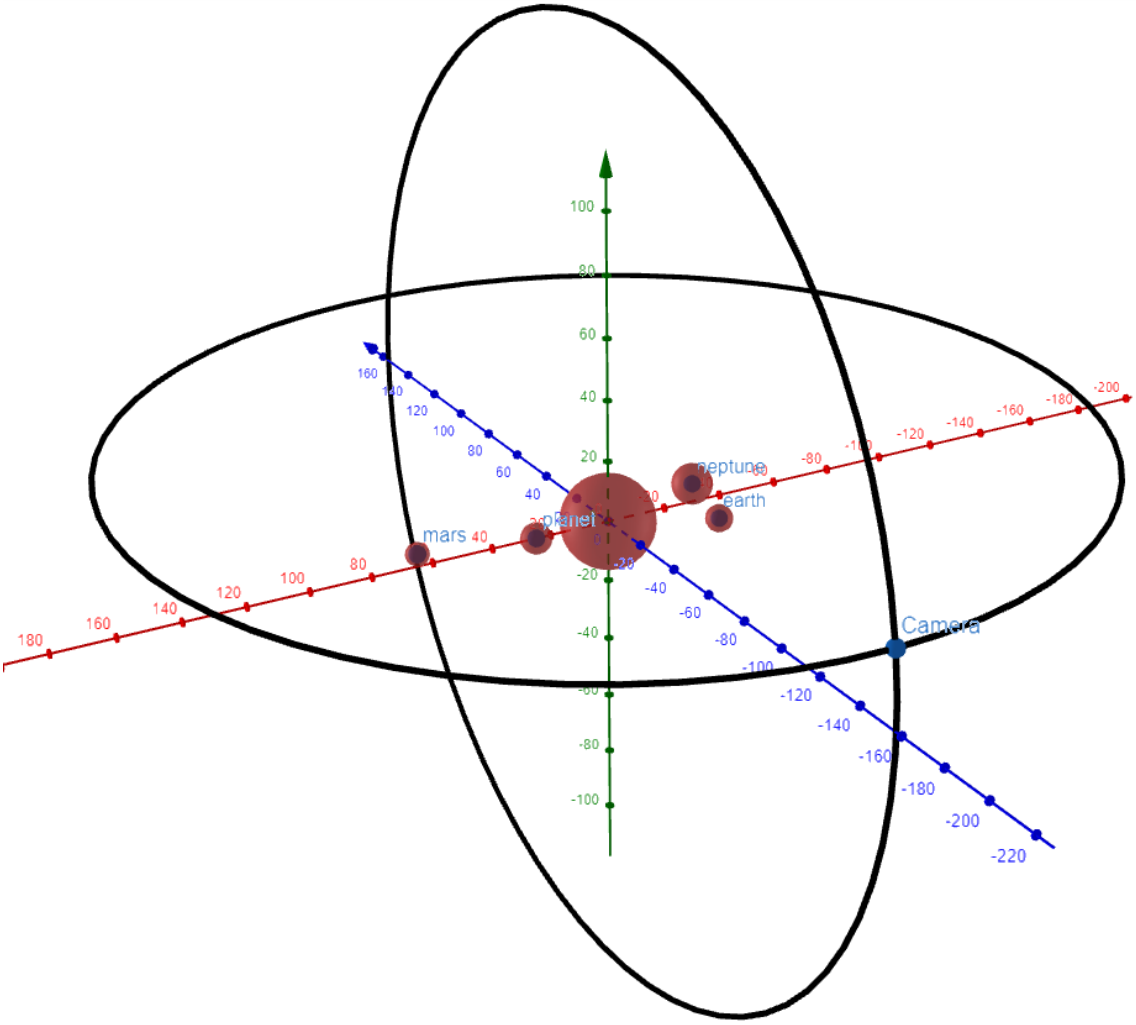


Οι τροχιές τον πλανητών κάθετα στο άξονα y



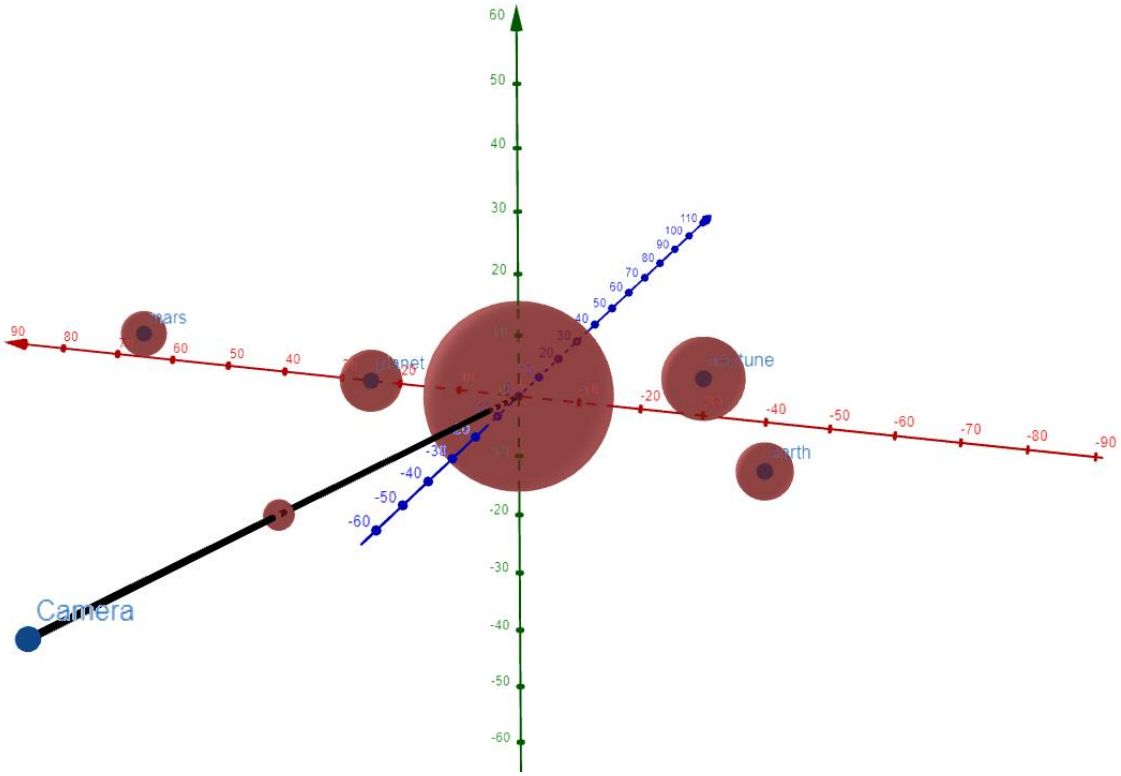
Οι τροχιές τον πλανητών κάθετα στο άξονα z

Η κάμερα βρίσκεται αρχικά στο σημείο CAM(-0.5, 24, -155). Σχεδιάστηκαν ενδεικτικά και οι κινήσεις της κάμερας γύρω από τους άξονες x και y από την αρχική της θέση. Φυσικά, οι κινήσεις της κάμερας γύρω από τους άξονες μπορεί να εντελώς διαφορετικές από αυτές που φαίνονται στο στιγμιότυπο.



Σημείο και ενδεικτικές κινήσεις κάμερας

Τέλος, ενδεικτικά έχει σχεδιαστεί και η κίνηση του μετεωρίτη πάνω στο ευθύγραμμο τμήμα που συνδέει την θέση της κάμερας με το κέντρο του ηλίου.



Ενδεικτική κίνηση μετεωρίτη