ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΑΣΚΗΣΗ 1-Γ

Γραφικά Υπολογιστών και Συστήματα Αλληλεπίδρασης - ΜΥΥ702

Ακαδ. Έτος 2021-2022

ΚΙΚΙΔΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ (4387)
ΤΣΑΜΠΙΡΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ (4508)

Λειτουργία του προγράμματος

Στο πρόγραμμά μας, ανοίγουμε ένα παράθυρο 800x800 με τίτλο «Ηλιακό Σύστημα» με μαύρο background. Το πρόγραμμα τερματίζει πατώντας το **Q** (κεφαλαίο). Φορτώνουμε τον Ήλιο (μία σφαίρα S με κέντρο το A(0,0,0) με το αντίστοιχο texture), έναν πλανήτη (μία σφαίρα P με κέντρο το B(25,0,0) με το αντίστοιχο texture) και επιπρόσθετα έναν ακόμα πλανήτη (βασιζόμενοι στην σφαίρα P με κέντρο το C(-50,0,0) και ένα αντίστοιχο texture). Οι πλανήτες περιστρέφονται γύρω από τον Ήλιο σε κυκλική τροχιά με ακτίνα 25 και 50 αντίστοιχα με σταθερή ταχύτητα (η οποία μπορεί να αυξηθεί ή να μειωθεί για τον πλανήτη P πατώντας τα πλήκτρα **u** και **p** αντίστοιχα). Η κάμερα μετακινείται προς το κέντρο του Ήλιου και μακριά από αυτό με τα πλήκτρα **+** και **–** αντίστοιχα, έχουμε υλοποιήσει και μετακίνηση γύρω από τους άξονες x και y με τα πλήκτρα **w x** και **a d** αντίστοιχα, ωστόσο δεν λειτουργεί απόλυτα σωστά. Τέλος, ο χρήστης πατώντας το πλήκτρο **spacebar** ενεργοποιεί έναν μετεωρίτη M (με το αντίστοιχο texture), ο οποίος ξεκινά από την θέση της κάμερας και κατευθύνεται προς το κέντρο του Ήλιου. Αν ο μετεωρίτης χτυπήσει τον πλανήτη P (όχι τον πλανήτη Άρη, αν και μπορεί να επεκταθεί ο κώδικας ώστε να λειτουργεί και για την περίπτωση σύγκρουσης με αυτόν) τότε εξαφανίζονται και ο μετεωρίτης και ο πλανήτης P, ενώ αν δεν συγκρουστεί, θα χαθεί μέσα στον ήλιο.

Προβλήματα που υπάρχουν

Δεν έχουμε υλοποιήσει την έκρηξη στην περίπτωση σύγκρουσης του μετεωρίτη με τον πλανήτη. Θα μπορούσαμε να χρησιμοποιήσουμε Particles/Instancing αλλά ήταν αρκετά σύνθετο.

Ιδιαιτερότητες/Ειδικές συνθήκες

Όταν πατάμε τα πλήκτρα για να μετακινήσουμε την κάμερα γύρω από τους άξονες x και y, μπορεί να βρεθούμε σε μία εντελώς άκυρη θέση όπου μπορούμε να βγούμε με zoom-out πατώντας το **–** .

Bonus που υλοποιήθηκαν

Υλοποιήσαμε τα γ. (προσθήκη περισσότερων πλανητών, συγκεκριμένα του πλανήτη Άρη) και ε. (αυξομείωση της ταχύτητας περιστροφής της σφαίρας P).

Ανάλυση του κώδικα που χρησιμοποιήθηκε

Για το αρχείο Source.cpp

- Στην γραμμή 318, ορίζουμε το μέγεθος και τον τίτλο του παραθύρου και στην 346 το χρώμα του background.

```
window = glfwCreateWindow(800, 800, "Ηλιακό Σύστημα", NULL, NULL);
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
```

- Στις γραμμές 367-371 φορτώνουμε τα textures.

```
int width, height, nrChannels;
unsigned char* sunImage = stbi_load("sun.jpg", &width, &height, &nrChannels, 0);
unsigned char* planetImage = stbi_load("planet.jpg", &width, &height, &nrChannels, 0);
unsigned char* meteorImage = stbi_load("meteor.jpg", &width, &height, &nrChannels, 0);
unsigned char* marsImage = stbi_load("mars.jpg", &width, &height, &nrChannels, 0);
```

- Στις γραμμές 383-385 παράγουμε τα textures.

```
GLuint textureID[4];
glGenTextures(4, textureID);
GLuint TextureID = glGetUniformLocation(programID, "myTextureSampler");
```

- Στις γραμμές 388-406 φορτώνουμε τα .obj αρχεία.

```
std::vector<glm::vec3> verticesSun;
std::vector<glm::vec3> normalsSun;
std::vector<glm::vec2> uvsSun;
bool resSun = loadOBJ("sun.obj", verticesSun, uvsSun, normalsSun);
```

```
std::vector<glm::vec3> verticesPlanet;
std::vector<glm::vec3> normalsPlanet;
std::vector<glm::vec2> uvsPlanet;
bool resPlanet = loadOBJ("planet.obj", verticesPlanet, uvsPlanet, normalsPlanet);

std::vector<glm::vec3> verticesMeteor;
std::vector<glm::vec3> normalsMeteor;
std::vector<glm::vec2> uvsMeteor;
bool resMeteor = loadOBJ("meteor.obj", verticesMeteor, uvsMeteor, normalsMeteor);

std::vector<glm::vec3> verticesMars;
std::vector<glm::vec3> normalsMars;
std::vector<glm::vec2> uvsMars;
bool resMars = loadOBJ("mars.obj", verticesMars, uvsMars, normalsMars);
```

- Στις γραμμές 410-430 τα φορτώνουμε σε ένα VBO (σε έναν vertexbuffer[4] κρατάμε τα vertices όλων των αντικειμένων και αντίστοιχα για τα uvs σε έναν uvbuffer[4]).

```
GLuint vertexbuffer[4];
glGenBuffers(4, vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER,vertexbuffer[0]);
glBufferData(GL_ARRAY_BUFFER,verticesSun.size()*sizeof(glm::vec3),&verticesSun[0], GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER,vertexbuffer[1]);
glBufferData(GL_ARRAY_BUFFER,verticesPlanet.size()*sizeof(glm::vec3),&verticesPlanet[1],GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER,vertexbuffer[2]);
glBufferData(GL_ARRAY_BUFFER,verticesMeteor.size()*sizeof(glm::vec3),&verticesMeteor[2],GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER,vertexbuffer[3]);
glBufferData(GL_ARRAY_BUFFER,verticesMeteor.size()*sizeof(glm::vec3),&verticesMeteor[3],GL_STATIC_DRAW);

GLuint uvbuffer[4];
glGenBuffers(4, uvbuffer);
glBindBuffer(GL_ARRAY_BUFFER, uvbuffer[0]);
glBufferData(GL_ARRAY_BUFFER, uvsSun.size() * sizeof(glm::vec2), &uvsSun[0], GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, uvbuffer[1]);
glBufferData(GL_ARRAY_BUFFER, uvsPlanet.size() * sizeof(glm::vec2), &uvsPlanet[1], GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, uvbuffer[2]);
glBufferData(GL_ARRAY_BUFFER, uvsMeteor.size() * sizeof(glm::vec2), &uvsMeteor[2], GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, uvbuffer[3]);
glBufferData(GL_ARRAY_BUFFER, uvsMeteor.size() * sizeof(glm::vec2), &uvsMeteor[3], GL_STATIC_DRAW);
```

- Στις γραμμές 432-449 ορίζουμε κάποιες global μεταβλητές που θα πειράζουμε μέσα στην do while, αφορούν τις περιστροφές των πλανητών γύρω από τον ήλιο, κάποια flags για όταν ενεργοποιείται για πρώτη φορά και γενικά ο μετεωρίτης και για την περίπτωση που έχουμε σύγκρουση πλανήτη και μετεωρίτη. Ακόμη, ορίζουμε κάποιες μεταβλητές που αφορούν την θέση του μετεωρίτη και την μείωση που θα έχουμε σε κάθε άξονα κατά την μετακίνηση προς το κέντρο. Και επίσης δίνουμε μία αρχική ταχύτητα περιστροφής στον πλανήτη P.

```
double x = 25.0f;
double z = 0.0f;
double angle = 0.0f;

double x2 = -50.0f;
double z2 = 0.0f;
double angle2 = 0.0f;

bool meteorInit = true;
bool meteorActivate = false;
bool planetCollision = false;

float meteorXPos, meteorYPos, meteorZPos;
float stepX, stepY, stepZ;

double speed = 0.05f;
```

- Στις γραμμές 459-477, ορίζουμε τους MVP πίνακες για τους πλανήτες και τον ήλιο, για τους πλανήτες έχουμε και μετακίνησή τους στις κατάλληλες θέσεις που ορίζουν οι μεταβλητές x, z, x2, z2 (με translate) και αύξηση του μεγέθους του πλανήτη Άρη επί 3 (με scale).

```
computeMatricesFromInputs();
glm::mat4 ProjectionMatrix = getProjectionMatrix();
glm::mat4 ViewMatrix = getViewMatrix();

glm::mat4 ModelMatrixSun = glm::mat4(1.0);
glm::mat4 SunMVP = ProjectionMatrix * ViewMatrix * ModelMatrixSun;

glm::mat4 ModelMatrixPlanet = glm::mat4(1.0);
ModelMatrixPlanet = glm::translate(ModelMatrixPlanet, glm::vec3(x, 0.0f, z));
glm::mat4 PlanetMVP = ProjectionMatrix * ViewMatrix * ModelMatrixPlanet;

glm::mat4 ModelMatrixMars = glm::mat4(1.0);
ModelMatrixMars = glm::translate(ModelMatrixMars, glm::vec3(x2, 0.0f, z2));
ModelMatrixMars = glm::scale(ModelMatrixMars, glm::vec3(3.0f, 3.0f, 3.0f));
glm::mat4 MarsMVP = ProjectionMatrix * ViewMatrix * ModelMatrixMars;
```

- Στις γραμμές 480-484 και 493, μετακινούμε τον πλανήτη P γύρω από τον ήλιο σε κυκλική τροχιά αλλάζοντας τις συντεταγμένες x και z. Αντίστοιχα κάνουμε και για τον πλανήτη Άρη στις γραμμές 496-501.

```
double radius = 25.0f;
x = radius * sin(3.14 * 2 * angle / 360);
z = radius * cos(3.14 * 2 * angle / 360);
glm::vec3 planetPos = glm::vec3(x, 0.0f, z);
ModelMatrixPlanet = glm::translate(ModelMatrixPlanet, planetPos);

angle += speed;

double radiusMars = -50.0f;
x2 = radiusMars * sin(3.14 * 2 * angle2 / 360);
z2 = radiusMars * cos(3.14 * 2 * angle2 / 360);
glm::vec3 marsPos = glm::vec3(x2, 0.0f, z2);
ModelMatrixMars = glm::translate(ModelMatrixMars, marsPos);
angle2 += 0.05f;
```

- Στις γραμμές 487-492, μπορούμε να προσαρμόσουμε την ταχύτητα περιστροφής του πλανήτη P πατώντας είτε το **u** για να την αυξήσουμε είτε το **p** για να την μειώσουμε.

```
if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS) {
        speed += 0.001f;
}
if (glfwGetKey(window, GLFW_KEY_P) == GLFW_PRESS) {
        speed -= 0.001f;
}
```

- Στη γραμμή 504, ορίζουμε τον MVP πίνακα του μετεωρίτη.

```
glm::mat4 ModelMatrixMeteor = glm::mat4(1.0);
```

- Στις γραμμές 507-509 εντοπίζουμε αν έχει πατηθεί το πλήκτρο **spacebar**, όπου σε αυτή την περίπτωση, ενεργοποιούμε τον μετεωρίτη (θέτωντας το meteorActivate flag σε true).

```
if (glfwGetKey(window, GLFW_KEY_SPACE) == GLFW_PRESS) {
        meteorActivate = true;
}
```

- Μόλις ενεργοποιηθεί ο μετεωρίτης και για όσο είναι ενεργός, μπαίνουμε μέσα στο if block στις γραμμές 512-592, ο κώδικας στο εσωτερικό περιγράφεται παρακάτω.

```
if (meteorActivate) { … }
```

- Στις γραμμές 513-515, ελέγχουμε αν έχει ήδη γίνει σύγκρουση, όπου σε αυτή την περίπτωση, απενεργοποιούμε τον μετεωρίτη (θέτωντας το meteorActivate flag σε false).

```
if (planetCollision) {
	meteorActivate = false;
}
```

- Στις γραμμές 517-529, αρχικοποιούμε την 1η θέση του μετεωρίτη, που είναι επί της ουσίας η τρέχουσα θέση της κάμερας, απενεργοποιούμε το flag meteorInit (θέτοντάς το σε false), ώστε να μην ξαναρχικοποιηθεί και ορίζουμε και το βήμα με το οποίο ο μετεωρίτης θα κατευθύνεται προς το κέντρο του ήλιου (το βήμα θα μειώνει ομοιόμορφα και τις 3 συντεταγμένες του μετεωρίτη μέχρι να φτάσουν πολύ κοντά στο μηδέν).

```
if (meteorInit) {
	glm::vec3 camPos = getPosition();
	meteorXPos = camPos.x;
	meteorYPos = camPos.y;
	meteorZPos = camPos.z;
	std::cout << "DEBUG ::: Meteor Starting Position: " + glm::to_string(glm::vec3(meteorXPos, meteorYPos,
meteorZPos)) + "\n";
	ModelMatrixMeteor = glm::translate(ModelMatrixMeteor, glm::vec3(meteorXPos,meteorYPos,meteorZPos));
	meteorInit = false;
	int stepFactor = 2500;
	stepX = meteorXPos / stepFactor;
	stepY = meteorYPos / stepFactor;
	stepZ = meteorZPos / stepFactor;
}
```

- Στις γραμμές 530-536, μετακινούμε τον μετεωρίτη προς τον ήλιο, με το βήμα που έχουμε ορίσει παραπάνω.

```
meteorXPos -= stepX;
meteorYPos -= stepY;
meteorZPos -= stepZ;
glm::vec3 meteorPos = glm::vec3(meteorXPos, meteorYPos, meteorZPos);
ModelMatrixMeteor = glm::translate(ModelMatrixMeteor, meteorPos);
glm::mat4 MeteorMVP = ProjectionMatrix * ViewMatrix * ModelMatrixMeteor;
std::cout << "DEBUG ::: Meteor Position: " + glm::to_string(glm::vec3(meteorXPos, meteorYPos,
meteorZPos)) +"\n";
```

- Στις γραμμές 539-544, ελέγχουμε αν ο μετεωρίτης είναι πολύ κοντά στο κέντρο του ήλιου, όπου τότε τον απενεργοποιούμε (με meteorActivate = false) και ορίζουμε το βήμα του σε 0 ώστε να μην βγει έξω από αυτόν αν ξαναπατήσουμε το **spacebar**.

```
if (meteorXPos < 1.0f && meteorYPos < 1.0f && meteorZPos < 1.0f) {
	meteorActivate = false;
	stepX = 0;
	stepY = 0;
	stepZ = 0;
}
```

- Στις γραμμές 547-555, υπολογίζουμε την απόσταση μεταξύ του πλανήτη P και του μετεωρίτη με βάση τον τύπο $d(meteorCoord_2 - planetCoord_1) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$ και αν η απόσταση αυτή είναι μικρότερη από το άθροισμα των ακτινών του πλανήτη P (=5) και του μετεωρίτη (=2), τότε έχουμε σύγκρουση και εξαφανίζονται και τα δύο. Απενεργοποιούμε τον μετεωρίτη (με meteorActivate = false), ενεργοποιούμε το flag της συγκρουσης (με planetCollision = true) και μηδενίζουμε το βήμα.

```
double distanceMeteorPlanetCenters = sqrt((pow((meteorPos.x - planetPos.x), 2.0f)) +
(pow((meteorPos.y - planetPos.y), 2.0f)) + (pow((meteorPos.z - planetPos.z), 2.0f)));
if (distanceMeteorPlanetCenters < 7) {
	meteorActivate = false;
	planetCollision = true;
	stepX = 0;
	stepY = 0;
	stepZ = 0;
	printf("\nDEBUG ::: Meteor and Planet Collided\n\n");
}
```

- Αν ο μετεωρίτης είναι ενεργός, και δεν έχει απενεργοποιηθεί από τις παραπάνω συνθήκες, τότε θα ζωγραφιστεί (γραμμές 558-591).

```
if (meteorActivate) {
	glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MeteorMVP[0][0]);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, meteorImage);
	glGenerateMipmap(GL_TEXTURE_2D);
	glActiveTexture(GL_TEXTURE2);
	glBindTexture(GL_TEXTURE_2D, textureID[1]);
	glUniform1i(TextureID, 2);

	glEnableVertexAttribArray(0);
	glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer[2]);
	glVertexAttribPointer(0,3,GL_FLOAT,GL_FALSE,0,(void*)0);

	glEnableVertexAttribArray(1);
	glBindBuffer(GL_ARRAY_BUFFER, uvbuffer[2]);
	glVertexAttribPointer(1,2,GL_FLOAT,GL_FALSE,0,(void*)0);

	glDrawArrays(GL_TRIANGLES, 0, verticesMeteor.size());

}
```

- Αντίστοιχα, όπως ζωγραφίσαμε τον μετεωρίτη παραπάνω, ζωγραφίζουμε και τον ήλιο και τον πλανήτη P (και τον πλανήτη Άρη) στις γραμμές 597-628, 636-667 και 672-703 αντίστοιχα.
- Για τον πλανήτη P μόνο, αν έχει γίνει σύγκρουση με τον μετεωρίτη, έχουμε μια if συνθήκη στις γραμμές 633-635, όπου ουσιαστικά μηδενίζουμε τον MVP πίνακά του, ώστε να μην εμφανίζεται.

```
if (planetCollision) {
	PlanetMVP = glm::mat4(0.0f);
}
```

- Και στις γραμμές 715-716, ανιχνεύουμε αν έχει πατηθεί το **Q** κεφαλαίο ή το X του παραθύρου, ώστε να τερματιστεί το πρόγραμμα, όσο δεν έχει γίνει κάτι από τα παραπάνω, το πρόγραμμα τρέχει κανονικά.

```
while ((!(GetKeyState('Q') & 0x8000) || (!(GetKeyState(VK_SHIFT) & 0x8000) ^
(GetKeyState(VK_CAPITAL) & 1))) && glfwWindowShouldClose(window) == 0);
```

Για το αρχείο controls.hpp

- Στις γραμμές 7, 8 και 12, ορίζουμε την GLM_FORCE_SWIZZLE για να δουλευεί ο μετεωρίτης σωστά, την GLM_ENABLE_EXPERIMENTAL ώστε να μπορούμε να χρησιμοποιήσουμε την string_cast και να εκτυπώνουμε συναρτήσεις της GLM και δηλώνουμε την συνάρτηση getPosition() που θα ορίσουμε στο αρχείο controls.cpp

```
#define GLM_FORCE_SWIZZLE
#define GLM_ENABLE_EXPERIMENTAL
glm::vec3 getPosition();
```

Για το αρχείο controls.cpp

- Στη γραμμή 12, εισάγουμε την string_cast.hpp προκειμένου να εκτυπώνουμε συναρτήσεις της GLM

```
#include <glm/gtx/string_cast.hpp>
```

- Στις γραμμές 20-33, ορίζουμε κάποιες global μεταβλητές που θα χρησιμοποιήσουμε παρακάτω

```
glm::mat4 ViewMatrix;
glm::mat4 ProjectionMatrix;
double x = 100.0;
double y = 1.0;
double z = 100.0;
glm::vec3 position = glm::vec3(x, y, z);
double angleA = 0.0f;
double angleB = 0.0f;
float horizontalAngle = 3.14f;
```

```
float verticalAngle = 0.0f;
float initialFoV = 45.0f;
float speed = 5.0f;
```

- Στις γραμμές 41-48 ορίζουμε τις συναρτήσεις getPosition() και setPos(), η πρώτη επιστρέφει την θέση της κάμερας σε ένα διάνυσμα ενώ η δεύτερη, ενημερώνει τα x,y,z της θέσης της κάμερας, αφού τα έχουμε μεταβάλει κατά το zoom-in/zoom-out της κάμερας και κατά την περιστροφή στους άξονες X και Y.

```
glm::vec3 getPosition() {
	return position;
}
void setPos() {
	x = position.x;
	y = position.y;
	z = position.z;
}
```

- Στη γραμμή 93, εκτυπώνουμε την τρέχουσα θέση της κάμερας.

```
std::cout << "DEBUG ::: Camera Position: " + glm::to_string(position) + "\n";
```

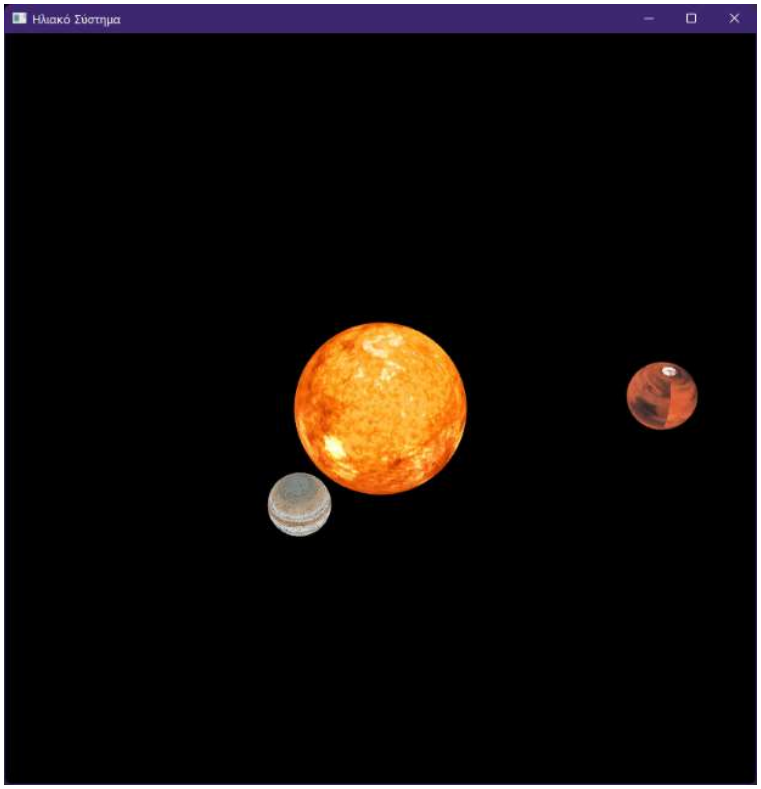
- Στις γραμμές 95-102 ελέγχουμε το zoom-in/zoom-out της κάμερας με τα πλήκτρα **+** και **–** (για το πλήκτρο +, αρκεί να πατηθεί το πλήκτρο που έχει πάνω το +, δηλαδή το =, γι'αυτό και χρησιμοποιούμε την GLFW_KEY_EQUAL).

```
if (glfwGetKey(window, GLFW_KEY_EQUAL) == GLFW_PRESS) {
	position += direction * deltaTime * speed;
	setPos();
}
if (glfwGetKey(window, GLFW_KEY_MINUS) == GLFW_PRESS) {
	position -= direction * deltaTime * speed;
	setPos();
}
```

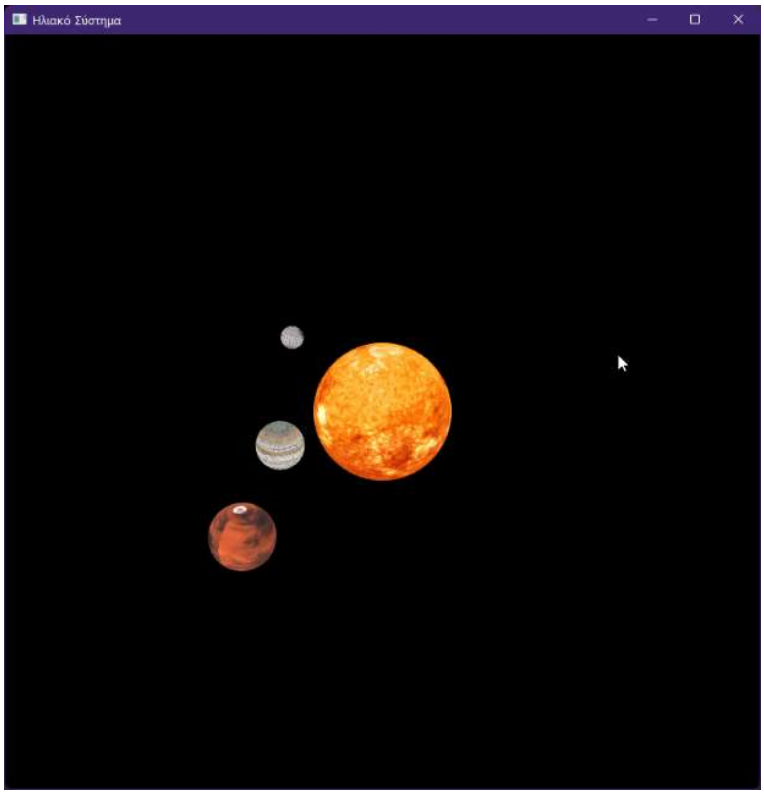
- Στις γραμμές 105-131, υλοποιούμε την κίνηση γύρω από τους άξονες X και Y, όπως κάναμε για τους πλανήτες.

```
// Y-axis rotation
if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS) {
	setPos();
	x = y * sin(3.14 * 2 * angleA / 360);
	z = y * cos(3.14 * 2 * angleA / 360);
	position = glm::vec3(x, y, z);
	angleA += 0.1f;
}
if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) {
	setPos();
	x = y * sin(3.14 * 2 * angleA / 360);
	z = y * cos(3.14 * 2 * angleA / 360);
	position = glm::vec3(x, y, z);
	angleA -= 0.1f;
}
```

```
// X-axis rotation
if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
	setPos();
	y = x * sin(3.14 * 2 * angleB / 360);
	z = x * cos(3.14 * 2 * angleB / 360);
	position = glm::vec3(x, y, z);
	angleB += 0.1f;
}
if (glfwGetKey(window, GLFW_KEY_X) == GLFW_PRESS) {
	setPos();
	y = x * sin(3.14 * 2 * angleB / 360);
	z = x * cos(3.14 * 2 * angleB / 360);
	position = glm::vec3(x, y, z);
	angleB -= 0.1f;
}
```



Εκτέλεση του προγράμματος



Με την ενεργοποίηση του μετεωρίτη (εικόνα από άλλη θέση)