ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΗ ΑΣΚΗΣΗ 1-Γ

Κουφόπουλος Μύρων ΑΜ 4398

Τριανταφυλλόπουλος Ανδρέας ΑΜ 4504

Ο κώδικας που παραδίδεται είναι 2 αρχεία .cpp που βασίστηκαν στο παράδειγμα που μας δώσατε, μια tutorial07.cpp και μια controls.cpp. Το πρόγραμμα μας το φτιάξαμε σε Visual Studio στα windows και για να τρέξει σωστά έχουμε κάνει include την βιβλιοθήκη windows.h.

i) Οι αλλαγές του ερωτήματος βρίσκονται στο tutorial07.cpp

Για το πρώτο ερώτημα φτιάξαμε το βασικό παράθυρο να είναι στο 800x800 με τίτλο Ηλιακό Σύστημα .

```
258     // Open a window and create its OpenGL context
259     window = glfwCreateWindow(800, 800, u8"Ηλιακό Σύστημα", NULL, NULL);
260     if (window == NULL) {
261         fprintf(stderr, "Failed to open GLFW window. If you have an Intel GPU,
262         getchar();
263         glfwTerminate();
264         return -1;
265     }
266     glfwMakeContextCurrent(window);
```

Για το μαύρο background γράψαμε την παρακάτω εντολή με κάθε RGB όλα 0.

```
277     // Black background
278     glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
```

Για τον τερματισμό του προγράμματος κάναμε τα εξής:

Για τον έλεγχο του Caps Lock κάναμε include την βιβλιοθήκη windows.h για να μπορούμε να καταλαβαίνουμε ποτέ είναι ενεργοποιημένο το Caps Lock. Κάθε φορά ελέγχουμε την κατάσταση του Caps Lock με αυτή την εντολή:

```
GetKeyState(0x14) == 0
```

Όταν ο ακέραιος που επιστρέφει το GetKeyState είναι 0 το Caps Lock είναι απενεργοποιημένο ενώ όταν επιστρέφει τιμή ≠ 0 το Caps Lock είναι ενεργοποιημένο.

Για τον έλεγχο κλεισίματος παραθύρου με το ‘Q’ κατασκευάσαμε 3 διαφορετικές περιπτώσεις που μπορεί να έχουμε (δεν έχουν τοποθετηθεί στον έλεγχο της while αλλά εσωτερικά με if για να ξεχωρίζουν οι 3 περιπτώσεις και να είναι πιο ευκολά αναγνώσιμο)

Η πρώτη περίπτωση ελέγχει αν ο χρήστης πατήσει το Q ενώ το Caps Lock είναι ενεργοποιημένο και δεν πατιέται κάποιο από τα Shift (αριστερό ή δεξί).

```
if ((glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS) &&
    (GetKeyState(0x14) != 0) &&
    (glfwGetKey(window, GLFW_KEY_LEFT_SHIFT) != GLFW_PRESS) &&
    (glfwGetKey(window, GLFW_KEY_RIGHT_SHIFT) != GLFW_PRESS)) {
    break;
}
```

Η δεύτερη περίπτωση ελέγχει αν ο χρήστης πατήσει το Q ενώ το Caps Lock είναι απενεργοποιημένο και πιέζεται το αριστερό Shift.

```
else if ((glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS) &&
    (GetKeyState(0x14) == 0) &&
    (glfwGetKey(window, GLFW_KEY_LEFT_SHIFT) == GLFW_PRESS)) {
    break;
}
```

Η τρίτη περίπτωση ελέγχει αν ο χρήστης πατήσει το Q ενώ το Caps Lock είναι απενεργοποιημένο και πιέζεται το δεξί Shift.

```
else if ((glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS) &&
    (GetKeyState(0x14) == 0) &&
    (glfwGetKey(window, GLFW_KEY_RIGHT_SHIFT) == GLFW_PRESS)) {
    break;
}
```

ii) Οι αλλαγές του ερωτήματος βρίσκονται στο tutorial07.cpp

Αρχικά φορτώνουμε την σφαίρα S η οποία θα αντιπροσωπεύει τον ήλιο.

```
387     // Read our .obj file
388     std::vector<glm::vec3> vertices;
389     std::vector<glm::vec3> normals;
390     std::vector<glm::vec2> uvs;
391     bool res = loadOBJ("sun.obj", vertices, uvs, normals);
```

Παρατηρήσαμε από το αρχείο sun.obj πως η σφαίρα S είχε ακτίνα 15 οπότε θεωρήσαμε πως δεν χρειάζεται να κάνουμε κάποιες αλλαγές. Παρακάτω παρατίθενται οι συντεταγμένες των κορυφών μέσα από το αρχείο sun.obj . Από αυτό καταλαβαίνουμε πως εφόσον υπάρχουν κορυφές με ακτίνα 15 από το κέντρο της σφαίρας αυτές θα μας δίνουν την ακτίνα της σφαίρας μας.

```
69    v 15 0 0
```
```
133   v 0 15 0
```
```
24454 v 0 0 15
```

Όμως γνωρίζουμε πως αν χρειαζόταν να γίνουν αλλαγές στο μέγεθος της σφαίρας θα γινόντουσαν μέσα από την συνάρτηση scale την οποία έχουμε υλοποιήσει στη προηγούμενη άσκηση.

Γενικά για το άνοιγμα των textures αρχικοποιήσαμε έναν πίνακα για την αποθήκευση τους (δόθηκαν 3 textures 1 για τον ήλιο, 1 για τον πλανήτη και 1 για τον κομήτη). Μέσω της

συνάρτησης glGenTextures δημιουργήσαμε 5 textureID για να κάνουμε bind το κάθε texture με το αντίστοιχο ID (τα δυο έξτρα textures είναι για το bonus ερώτημα).

```
298     GLuint textureID[5];
299     glGenTextures(5, textureID);
```

Εδώ βάζουμε στη πρώτη θέση του πίνακα το πρώτο texture για τον ήλιο. Αρχικοποιούμε τις ακέραιες μεταβλητές width height και nrChannels οι οποίες θα χρησιμοποιηθούν για όλα τα textures (δεν φτιάξαμε καινούργιες διότι δεν ήταν απαραίτητο να χρησιμοποιούνται ταυτόχρονα). Γίνεται έλεγχος αν δεν φορτώθηκαν σωστά τα δεδομένα του texture παίρνουμε μήνυμα σφάλματος.

```
300     // "Bind" the newly created texture : all future texture functions will modify this texture
301     glBindTexture(GL_TEXTURE_2D, textureID[0]);
302     // Load the texture
303     int width, height, nrChannels;
304     unsigned char* data = stbi_load("sun.jpg", &width, &height, &nrChannels, 0);
305     if (data)
306     {
307
308     }
309     else
310     {
311         std::cout << "Failed to load texture" << std::endl;
312     }
313     // Give the image to OpenGL
314     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
315     glGenerateMipmap(GL_TEXTURE_2D);
```

Παρατίθεται ο κώδικας για τον ήλιο ο οποίος σχεδιάζει τον ήλιο (κώδικας μέσα στο while) στο σημείο A(0,0,0) για το οποίο δεν χρειάστηκε να γίνει κάποια αλλαγή στην αρχικοποίησή του. Επίσης οι buffers αρχικοποιήθηκαν διαφορετικοί για κάθε object.

```
410     // buffers gia ton hlio
411     GLuint vertexbuffer;
412     glGenBuffers(1, &vertexbuffer);
413     glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
414     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(glm::vec3), &vertices[0], GL_STATIC_DRAW);
415
416     GLuint uvbuffer;
417     glGenBuffers(1, &uvbuffer);
418     glBindBuffer(GL_ARRAY_BUFFER, uvbuffer);
419     glBufferData(GL_ARRAY_BUFFER, uvs.size() * sizeof(glm::vec2), &uvs[0], GL_STATIC_DRAW);
420
```

Οι γραμμές που σχηματίζουν τον ήλιο καθ όλη την διάρκεια του προγράμματος είναι στις γραμμές 479-532.

Επίσης να σημειωθεί πως για όλα τα object χρησιμοποιούμε ίδια ονόματα programID, projectionMatrix, ViewMatrix και MVP.

Ακόμα για την αντιστοίχιση του texture πάνω στις U,V συντεταγμένες του σχήματος κάνουμε active κάθε φορά το GL_TEXTURE0 και κάνουμε bind το σωστό textureID που έχουμε αποθηκεύσει στην λίστα.

```
498         // Bind our texture in Texture Unit 0
499         glActiveTexture(GL_TEXTURE0);
500         glBindTexture(GL_TEXTURE_2D, textureID[0]);
501         // Set our "myTextureSampler" sampler to use Texture Unit 0
502         glUniform1i(TextureID, 0);
```

Οι αλλαγές παρακάτω βρίσκονται στο controls.cpp.

Για την κίνηση της κάμερας μέσω των πλήκτρων έχουμε :

Γραμμές στο κώδικα 65-139 για την κίνηση αναλόγως τα πλήκτρα που πατιούνται, η λειτουργία των κεφαλαίων είναι αντίστοιχη όπως αναφέραμε παραπάνω.

Η αναλυτική επεξήγηση της κίνησης της κάμερας αναφέρεται στην προηγούμενη άσκηση (λειτουργεί όπως ζητείται).

iii) Οι αλλαγές του ερωτήματος βρίσκονται στο tutorial07.cpp

Αρχικά φορτώνουμε το planet.obj και κρατάμε τα στοιχεία του σε διαφορετικά διανύσματα απ ότι στον ήλιο.

```
392     std::vector<glm::vec3> vertices2;
393     std::vector<glm::vec3> normals2;
394     std::vector<glm::vec2> uvs2;
395     bool res2 = loadOBJ("planet.obj", vertices2, uvs2, normals2);
```

Εδώ δημιουργούμε δυο Buffers, έναν για τις κορυφές και έναν για τις UV συντεταγμένες.

```
421 // buffers gia ton planhth
422 GLuint vertexbuffer2;
423 glGenBuffers(1, &vertexbuffer2);
424 glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer2);
425 glBufferData(GL_ARRAY_BUFFER, vertices2.size() * sizeof(glm::vec3), &vertices2[0], GL_STATIC_DRAW);
426
427 GLuint uvbuffer2;
428 glGenBuffers(1, &uvbuffer2);
429 glBindBuffer(GL_ARRAY_BUFFER, uvbuffer2);
430 glBufferData(GL_ARRAY_BUFFER, uvs2.size() * sizeof(glm::vec2), &uvs2[0], GL_STATIC_DRAW);
```

Εδώ φορτώνουμε το texture του πλανήτη και κρατάμε τα δεδομένα του στη λίστα των textures.

```
317 // "Bind" the newly created texture : all future texture functions will modify this texture
318 glBindTexture(GL_TEXTURE_2D, textureID[1]);
319 // Load the texture
320 data = stbi_load("planet.jpg", &width, &height, &nrChannels, 0);
321 if (data)
322 {
323
324 }
325 else
326 {
327     std::cout << "Failed to load texture" << std::endl;
328 }
329 // Give the image to OpenGL
330 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
331 glGenerateMipmap(GL_TEXTURE_2D);
```

Στις γραμμές από 534-588 γίνεται ο κατ επανάληψη σχεδιασμός του πλανήτη.

Εδώ γίνονται οι δυο κινήσεις του πλανήτη, η περιστροφή γύρω από τον ήλιο και το bonus κομμάτι δηλαδή η περιστροφή ταυτόχρονα και γύρω από τον εαυτό του. Αρχικά δημιουργούμε τον ModelMatrix του πλανήτη και στη συνέχεια του ορίζουμε το κέντρο γύρω από το οποίο θα περιστρέφεται (τροχιά - orbit). Στην συνέχεια του λέμε ότι πρέπει να περιστρέφεται πάνω στον άξονα y'y, δηλαδή ότι θα έχει πάντα σταθερό y και ίσο με 0 στο κέντρο του πλανήτη. Έπειτα του ορίζουμε το μέγεθος της ακτίνας της τροχιάς πάνω στον άξονα y το οποίο είναι 25 (άρα έμμεσα 'αρχικοποιούμε' τον πλανήτη στο σημείο στο σημείο

B (25,0,0)). Για την περιστροφή γύρω από τον εαυτό του γραμμή 545-546 γίνεται η κίνηση γύρω από τον άξονα z του πλανήτη . Στη συνέχεια υπολογίζουμε τον MVP του πλανήτη.

```
541    glm::mat4 ModelMatrix1 = glm::mat4(1.0);
542    ModelMatrix1 = glm::translate(ModelMatrix1, glm::vec3(0.0f, 0.0f, 0.0f));   // gurw apo pou thelw na kanw orbit
543    ModelMatrix1 = glm::rotate(ModelMatrix1, m_scale, glm::vec3(0.0f, 1.0f, 0.0f)); //a3onas panw ston opoio 8elw na kanei orbit
544    ModelMatrix1 = glm::translate(ModelMatrix1, glm::vec3(25.0f, 0.0f, 0.0f));  // h aktina pou thelw na kanei orbit
545    ModelMatrix1 = glm::translate(ModelMatrix1, glm::vec3(0.0f, 0.0f, 0.0f));
546    ModelMatrix1 = glm::rotate(ModelMatrix1, m_scale_planet, glm::vec3(0.0f, 0.0f, 1.0f)); //rotate gyrw apo ton eauto tou
547    MVP = ProjectionMatrix * ViewMatrix * ModelMatrix1;
```

Στη γραμμή 560 και 572 βάζουμε πάλι σαν Attribute Array το 0 και το 1 όπως και στον ήλιο και αυτό δεν μας επηρεάζει διότι κάθε φορά που τελειώνει η ρουτίνα για το κάθε object κάνουμε disable το κάθε vertex Attribute Array. Αυτό θα το κάνουμε για όλα τα object που θα δημιουργήσουμε.

```
559    // 1rst attribute buffer : vertices
560    glEnableVertexAttribArray(0);
561    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer2);
562    glVertexAttribPointer(
563        0,                  // attribute
564        3,                  // size
565        GL_FLOAT,           // type
566        GL_FALSE,           // normalized?
567        0,                  // stride
568        (void*)0            // array buffer offset
569    );
570
571    // 2nd attribute buffer : UVs
572    glEnableVertexAttribArray(1);
573    glBindBuffer(GL_ARRAY_BUFFER, uvbuffer2);
574    glVertexAttribPointer(
575        1,                                // attribute
576        2,                                // size
577        GL_FLOAT,                         // type
578        GL_FALSE,                         // normalized?
579        0,                                // stride
580        (void*)0                          // array buffer offset
581    );
```

Το texture ακολουθεί την ίδια διαδικασία που υλοποιήσαμε και για τον ήλιο.

```
553    // Bind our texture in Texture Unit 0
554    glActiveTexture(GL_TEXTURE0);
555    glBindTexture(GL_TEXTURE_2D, textureID[1]);
556    // Set our "myTextureSampler" sampler to use Texture Unit 0
557    glUniform1i(TextureID, 0);
```

Επίσης δημιουργούμε δυο μεταβλητές την float m_scale η οποία είναι για την περιστροφή του πλανήτη γύρω από τον ήλιο και την float m_scale_planet η οποία είναι για την περιστροφή του πλανήτη γύρω από τον εαυτό του (bonus). Οι δυο μεταβλητές αρχικοποιούνται εκτός της while και στο τέλος των διαδικασιών της while γραμμές 813 και 816 αυξάνουμε σταθερά και τις δυο αυτές μεταβλητές που οδηγούν σε σταθερή

περιστροφή σε κάθε περίπτωση (λόγω του bonus με τα u και p αυξομειώνεται η περιστροφή γύρω από τον εαυτό του).

Παρακάτω γίνεται ένας έλεγχος πριν τον σχεδιασμό του πλανήτη. Όταν ένας κομήτης πετυχαίνει τον πλανήτη το flag planet γίνεται ίσο με 1 και δεν γίνεται απεικονίζεται ούτε εμποδίζει πλέον ο πλανήτης αφού έχουν διαγραφεί πλέον και οι buffer των κορυφών και των UV συντεταγμένων (γραμμή 746-747).

```
583     // Draw the triangle !
584     if (planet == 0)
585         glDrawArrays(GL_TRIANGLES, 0, vertices2.size());
```

iv) Οι αλλαγές του ερωτήματος βρίσκονται στο tutorial07.cpp

Αρχικά φορτώνουμε το object του μετεωρίτη.

```
396     std::vector<glm::vec3> vertices3;
397     std::vector<glm::vec3> normals3;
398     std::vector<glm::vec2> uvs3;
399     bool res3 = loadOBJ("meteor.obj", vertices3, uvs3, normals3);
```

Δημιουργούμε ξεχωριστούς buffers για τις κορυφές και τις U,V συντεταγμένες του αντικειμένου.

```
432     // buffers gia ton komhth
433     GLuint vertexbuffer3;
434     glGenBuffers(1, &vertexbuffer3);
435     glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer3);
436     glBufferData(GL_ARRAY_BUFFER, vertices3.size() * sizeof(glm::vec3), &vertices3[0], GL_STATIC_DRAW);
437
438     GLuint uvbuffer3;
439     glGenBuffers(1, &uvbuffer3);
440     glBindBuffer(GL_ARRAY_BUFFER, uvbuffer3);
441     glBufferData(GL_ARRAY_BUFFER, uvs3.size() * sizeof(glm::vec2), &uvs3[0], GL_STATIC_DRAW);
442
```

Περνάμε το texture για τον μετεωρίτη όπως και παραπάνω.

```
333     // "Bind" the newly created texture : all future texture functions will modify this texture
334     glBindTexture(GL_TEXTURE_2D, textureID[2]);
335     // Load the texture
336     data = stbi_load("meteor.jpg", &width, &height, &nrChannels, 0);
337     if (data)
338     {
339
340     }
341     else
342     {
343         std::cout << "Failed to load texture" << std::endl;
344     }
345     // Give the image to OpenGL
346     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
347     glGenerateMipmap(GL_TEXTURE_2D);
```

Η επόμενη υλοποίηση του κώδικα βρίσκεται εντός της while:

Για να περάσει το πρόγραμμα τον έλεγχο και να σχεδιάσει τον μετεωρίτη πρέπει είτε ο χρήστης να πατήσει SPACE (μία φορά) και να τον δημιουργήσει είτε αν υπάρχει ήδη ο μετεωρίτης (δηλαδή έχουμε ήδη πατήσει SPACE) να έχει αλλάξει το flag του count σε 1.

```
704         // meteoriths
705         if (glfwGetKey(window, GLFW_KEY_SPACE) == GLFW_PRESS || count != 0) { //elegxos an
706             count = 1;
```

Αρχικοποιούμε το speed το οποίο είναι ένας float αριθμός για την σταθερή κίνηση του μετεωρίτη προς το κέντρο του κύκλου. Ακόμα αρχικοποιούμε ένα διάνυσμα vec3 που το ονομάζουμε positNew το οποίο μετά δημιουργία του μετεωρίτη θα κρατάει την θέση της κάμερας από την οποία δημιουργήθηκε ο μετεωρίτης και θα επιτρέπει στην κάμερα να κινείται αυτόνομα.

```
707             float speed = 0.01f;
708             glm::mat4 ModelMatrix2 = glm::mat4(1.0);
709             //dhmiourgia meteorhth
710             glUseProgram(programID);
711             glm::vec3 positNew;
```

Η συνθήκη αυτή πληρείται όταν ο χρήστης δημιουργεί το μετεωρίτη (μπορεί να έχει δημιουργήσει πολλούς πιο πριν όμως στο σχήμα αυτή την στιγμή θα βρίσκεται ένας). Το choose flag είναι αρχικοποιημένο στο 0 και εφόσον περάσει το πρώτο έλεγχο γίνεται 1 για να μην δημιουργείται άλλος κομήτης αλλά να κινείται μόνο ο υπάρχων. Δημιουργήσαμε μια συνάρτηση getPosition() από την οποία κρατάμε κάθε φορά την θέση που βρίσκεται η κάμερά μας. Η συνάρτηση αυτή υλοποιείται στην controls.cpp και προστέθηκε και στο controls.hpp.

(screenshot από controls.cpp)

```
23      // Initial position : on +Z
24      glm::vec3 position = glm::vec3(10, 10, 30);
25
26      glm::vec3 getPosition() {
27          return position;
28      }
```

Στη συνέχεια περνάμε αυτό το position στην positNew για να το κρατάμε όπως εξηγήσαμε παραπάνω. Υπολογίζονται κανονικά οι προβολές από τις εισόδους του χρήστη για την κίνηση της κάμερας. Ορίζουμε ένα διάνυσμα direction το οποίο χρησιμοποιούμε για να υπολογίζουμε την κατεύθυνση του μετεωρίτη που είναι πάντα προς το (0,0,0) (το κέντρο του συστήματος αξόνων). Κανονικοποιούμε το διάνυσμα direction με τη βοήθεια της θέσης που βρίσκεται ο μετεωρίτης και υπολογίζουμε την νέα του θέση στην γραμμή 722. Ο μετεωρίτης εδώ κινείται με ταχύτητα speed. Τελικά υπολογίζεται ο ModelMatrix του μετεωρίτη και τον μετατοπίζουμε στη θέση που υπολογίσαμε παραπάνω.

```
712         if (choose == 0)
713         {
714             choose = 1;
715             glm::vec3 posit = getPosition(); //pairnoume thn 8esh ths kameras gia thn topo8esia dhmio
716             positNew = posit; //thn krataw gia 8esh anaforas
717             computeMatricesFromInputs();
718             ProjectionMatrix = getProjectionMatrix();
719             ViewMatrix = getViewMatrix();
720             glm::vec3 direction = glm::vec3(0.0f, 0.0f, 0.0f); //orizw thn kateu8ynsh pros to kentro
721             direction = glm::normalize(posit - direction); //kanonikopoish ths kateu8ynshs
722             posit -= direction * speed; //ypologizw thn epomenh 8esh
723             ModelMatrix2 = glm::translate(glm::mat4(1.0), posit); //thn pernaw ston ModelMatrix
724             MVP = ProjectionMatrix * ViewMatrix * ModelMatrix2;
725         }
```

Αν ο μετεωρίτης υπάρχει ήδη τότε το flag choose είναι ίσο με 1 και θα συνεχίζει σε αυτό το κομμάτι σου κώδικα. Εδώ υπολογίζονται όλα όπως παραπάνω αλλά η θέση της κάμερας δεν μας επηρεάζει πάρα μόνο η αρχική θέση από την οποία δημιουργήθηκε ο μετεωρίτης. Το positNew ενημερώνεται συνεχώς με την θέση που βρίσκεται ο μετεωρίτης.

```
726         else
727         {   // gia na mporoume na doume ton meteorith kai na mhn kanei thn kinhsh ths cameras
728             computeMatricesFromInputs();
729             ProjectionMatrix = getProjectionMatrix();
730             ViewMatrix = getViewMatrix();
731             glm::vec3 direction = glm::vec3(0.0f, 0.0f, 0.0f);
732             direction = glm::normalize(positNew - direction);
733             positNew -= direction * speed;
```

```
540         else
541         {   // gia na mporoume na doume ton meteorith kai na mhn kanei thn kinhsh ths cameras
542             computeMatricesFromInputs();
543             ProjectionMatrix = getProjectionMatrix();
544             ViewMatrix = getViewMatrix();
545             glm::vec3 direction = glm::vec3(0.0f, 0.0f, 0.0f);
546             direction = glm::normalize(positNew - direction);
547             positNew -= direction * speed;
```

Παρακάτω ελέγχουμε αρχικά αν ο μετεωρίτης έχει εισχωρήσει στον ήλιο για να τον εξαφανίσουμε. Αυτό ελέγχεται με τη βοήθεια του διανύσματος positNew όπου ελέγχουμε αν η απόσταση και των τριών συντεταγμένων του μετεωρίτη είναι μέσα στον ήλιο. Χρησιμοποιούμε την απόλυτη τιμή στις συντεταγμένες που παίρνουμε για να αποφύγουμε λάθη όταν ο μετεωρίτης βρίσκεται σε αρνητικές συντεταγμένες. Στην γραμμή 723 ενημερώνουμε τον ModelMatrix του μετεωρίτη. Στη συνέχεια ελέγχουμε αν ο μετεωρίτης έχει συγκρουστεί με τον πλανήτη. Για να το κάνουμε αυτό παίρνουμε το κέντρο του πλανήτη και το κέντρο του μετεωρίτη. Ο ένας έχει ακτίνα 5 και ο άλλος ακτίνα 2 οπότε για να έχουμε σύγκρουση πρέπει τα δυο κέντρα να απέχουν απόσταση < 7 (5+2). Τις συντεταγμένες του πλανήτη τις παίρνουμε από τον ModelMatrix1 του ο οποίος είναι ένας 4*4 πίνακας και εμείς παίρνουμε τα στοιχεία [3][0], [3][1], [3][2]. Αυτό το κάνουμε γιατί εκεί βρίσκονται οι συντεταγμένες του κέντρου του πλανήτη. Το αντίστοιχο κάνουμε και για τον μετεωρίτη για να βρούμε τις συντεταγμένες του παίρνοντας το ModelMatrix2. Κάνουμε

την αφαίρεση για κάθε για κάθε ζεύγος συντεταγμένων x,y,z χωρίς να μας επηρεάζει ποιο είναι πρώτο αφού χρησιμοποιούμε απόλυτη τιμή. Αν έχει γίνει σύγκρουση του μετεωρίτη με τον πλανήτη, σύμφωνα με την εκφώνηση, τους εξαφανίζουμε και τους δυο. Για τον μετεωρίτη κάνουμε το flag count ίσο με μηδέν για να μην ξαναμπεί στην while και συνεχίσει να υπάρχει ο μετεωρίτης και να μπορούμε να δημιουργήσουμε άλλον μετεωρίτη .Ακόμα κάνουμε το choose ίσο με μηδέν ώστε όταν ξαναφτιαχτεί ο επόμενος μετεωρίτης να ξαναπάρουμε το καινούργιο position της κάμερας. Για τον πλανήτη κάνουμε το flag planet = 0 για να μην συνεχίσει να υπάρχει ο πλανήτης και διαγράφουμε τα buffers του διότι δεν θέλουμε να ξαναδημιουργηθεί άλλος πλανήτης. Οι γραμμές του screenshot 734-748.

```
734             //elelgxos an exei xa8ei mesa ston hlio gia na e3afanistei
735             if (abs(positNew.x) < 1.0 && abs(positNew.y) < 1.0 && abs(positNew.z) < 1.0) {
736                 count = 0;
737                 choose = 0;
738             }
739             ModelMatrix2 = glm::translate(glm::mat4(1.0), positNew);
740             //elegxos an exei xtyphsei ton planhth gia na katastrafoun kai oi 2
741             if ((abs(ModelMatrix2[3][0] - ModelMatrix1[3][0]) < 7.0) && (abs(ModelMatrix2[3][1] - ModelMatrix1[3][1]) < 7.0) && (abs(ModelMatrix2[3][2] - ModelMatrix1[3][2]) < 7.0) && planet == 0) {
742                 bool play = PlaySound("sound.wav", NULL, SND_ASYNC);
743                 count = 0;
744                 choose = 0;
745                 planet = 1;
746                 glDeleteBuffers(1, &vertexbuffer2);
747                 glDeleteBuffers(1, &uvbuffer2);
748             }
```

Bonus Ερωτήματα

Α) Για το εφέ ήχου προσθέσαμε το αρχείο sound.wav που ακούγεται όταν έχουμε σύγκρουση πλανήτη και μετεωρίτη. Ένα παράδειγμα φαίνεται παρακάτω.

```
741             if ((abs(ModelMatrix2[3][0] - ModelMatrix1[3][0]) < 7.0)
742                 bool play = PlaySound("sound.wav", NULL, SND_ASYNC);
```

Για να μπορέσει να πάρει η συνάρτηση στατικούς χαρακτήρες (το sound.wav) προσθέσαμε στο project μια βιβλιοθήκη την Winmm.lib.

Β) Παραπάνω εξηγήθηκε το πως κάναμε την κίνηση των πλανητών γύρω από τον εαυτό τους.

Γ) Προσθέσαμε δυο ακόμα πλανήτες με τα textures της Γης (earth.jpg) και του ποσειδώνα (Neptune.jpg) με διαφορετικές τροχιές από αυτή του βασικού πλανήτη, αλλά πάνω στον ίδιο άξονα κίνησης. Επίσης τους δώσαμε την ίδια ταχύτητα κίνησης γύρω από τον ήλιο για να μπορεί να ελεγχθεί η σύγκρουση του μετεωρίτη με τους πλανήτες. Για την εύρεση των πλανητών μέσα στον κώδικα αναφέρονται σαν Bonus 1 και Bonus 2 πλανήτες.

Ε) Με τα πλήκτρα u και p γίνεται αυξομείωση της ταχύτητας περιστροφής γύρω από τον εαυτό του βασικού μας πλανήτη. Ο έλεγχος γίνεται στις σειρές από 819-834. Αυτό που κάνουμε είναι ότι αυξάνουμε την μεταβλητή delim ώστε να περιστρέφεται πλέον ο πλανήτης γρηγορότερα. Δεν ήταν ξεκάθαρο αν η εκφώνηση εννοούσε την περιστροφή γύρω από τον ήλιο ή από τον εαυτό του, για την υλοποίηση της αλλαγής ταχύτητας στην περιστροφή γύρω από τον ήλιο θα ήταν αντίστοιχος κώδικας απλά με άλλη μεταβλητή αυξομείωσης.

Σημείωση:

1) Πάρα πολλά σχόλια υπάρχουν και μέσα στον κώδικα για καλύτερη κατανόηση του κώδικα μας. Κυρίως για τα flags στις γραμμές 465-476.

2) Διαπιστώσαμε ότι σε διαφορετικούς υπολογιστές (με διαφορετικές κάρτες γραφικών) οι ταχύτητες όλων των κινήσεων διαφέρουν. Συνήθως το δυνατότερο σύστημα τα κάνει όλα γρηγορότερα και αν είναι πολύ γρήγορη η περιστροφή γύρω από τον ήλιο μπορεί να εξαφανιστούν κάποιοι πλανήτες. Το πρόγραμμα είναι πλήρως λειτουργικό. Αν υπάρξει

οποιοδήποτε πρόβλημα μπορείτε να αυξομειώσετε τις ταχύτητες ανάλογα με τις ανάγκες του συστήματος στο οποίο διορθώνετε στις γραμμές 813-817 και τη γραμμή 707 για την ταχύτητα του μετεωρίτη.