Προγραμματιστική Άσκηση 1

Γραφικά Υπολογιστών και Συστήματα Αλληλεπίδρασης

Τσανίδης Δημήτριος 1935

Δήμητρα Μαχαιρίδου 4108

Στοιχεία επικοινωνίας:

email: dimitrtsanidis@gmail.com

maxedimi@gmail.com

msTeams: DIMITRIOS TSANIDIS

DIMITRA MACHAIRIDOU

ΓΕΝΙΚΑ

Υλοποιήσαμε την εργασία στο περιβάλλον Visual Studio 2019.

Το εκτελέσιμο είναι σε 32bit configuration.

Χρησιμοποιούμε τις βιβλιοθήκες GLEW, GLFW, GLM. Και για τις τρεις βιβλιοθήκες συνδέσαμε τις 32bit εκδοχές.

Υλοποιήσαμε όλα τα μη-bonus υποερωτήματα, εκτός του (ii).

Συγκεκριμένα:

Το πρόγραμμα ξεκινάει με το άνοιγμα ενός παραθύρου 600x600 με τον τίτλο "Συγκρουόμενα". Με το πάτημα του πλήκτρου <Esc> το πρόγραμμα τερματίζει. Το background του παραθύρου είναι χρώμα μαύρο.

Υλοποιήσαμε έναν διάφανο κύβο στις ζητούμενες διαστάσεις και θέση και μια συμπαγή σφαίρα στον κέντρο του, με την ζητούμενη ακτίνα.

Το χρώμα του κύβου υπολογίζεται τυχαία, όμως είναι 70% διαφανές ωστέ να φαίνεται η σφαίρα εντός του.

Η σφαίρα αρχικά εμφανίζεται με υφή την εικόνα texture.jpg που βρίσκεται στον φάκελο

του εκτελέσιμου της εφαρμογής.

Με το πάτημα του πλήκτρου <T> η υφή σταματά να εμφανίζεται και η σφαίρα πέρνει το χρώμα κόκκινο.

Επόμενο πάτημα του παραπάνω πλήκτρου επαναφέρη την υφή και ούτω καθεξής.

Πατώντας τα πλήκτρα <αριστερά,δεξιά,πάνω,κάτω,+,-> η σφαίρα μετακινείται στους άξονες x,y και z, αντίστοιχα, παραμένοντας όμως εντός των ορίων του κύβου.

Η κίνηση της κάμερας δεν είναι σωστά υλοποιημένη. Η κάμερα περιστρέφεται αλλά με τα πλήκτρα <A,D,W,S> αλλά όχι γύρω του κεντρου του κύβου. Κρατήσαμε και το ποντίκι ενεργό για ευκολία σας στην παρατήρηση του κύβου.

Αναλυτικότερα:

(i) Δημιουργούμε το παράθυρο με την χρήση της GLFW. Πιο συγκεκριμένα καλούμε την συνάρτηση glfwCreateWindow(). Για λόγους αισθητικής απενεργοποιούμε τον κέρσορα του ποντικιού.

Κατά την εκτέλεση της mainloop του rendering φροντίσαμε μέσω της glfwGetKey να σταματούμε loop και αφότου διαγράψουμε τους buffer που χρειαστήκαμε για τα υπόλοιπα ερωτήματα κλείνουμε το παράθυρο μέσω της μεθόδου glfwTerminate().

Όσον αφορά τον κύβο:

Δημιουργούμε τον κύβο στατικά και hardcoded. Δηλαδή ορίσαμε έναν πίνακα και περάσαμε τα vertices manually, ως τριαδες x,y,z όπου κάθε μεταβλητή έχει την τιμή είτε 0 είτε 100.

Αντίστοιχα για το color υπολογίζουμε ένα τυχαίο χρώμα με την χρήση της rand() και τα περνάμε manually σε πίνακα για τα colors. Σημειώνουμε πως κατά το rendering ο κύβος μερικές φορές δεν φαίνεται γιατί ένα απ' τα τυχαία χρώματα είναι το μαύρο.

Δεσμεύουμε τους παραπάνω πίνακες σε VBOs.

Πριν ζωγραφίσουμε τον κύβο ενεργοποιούμε το blending με τις εντολές glEnable(GL_BLEND); glBlendColor(0.0f, 0.0f, 0.0f, 0.3f); glBlendFunc(GL_CONSTANT_ALPHA, GL_ONE_MINUS_CONSTANT_ALPHA); οπου 0.3f είναι το A του χρώματος.

Σημειώνουμε πως ζωγραφίζουμε τον κύβο τελευταίο καθώς τα αντικείμενα που είναι transparent πρέπει να ζωγραφίζονται μετά τα αντικείμα που δεν είναι. Απενεργοποιούμε το blending πριν ζωγραφίσουμε τα υπόλοιπα αντικείμενο στο επόμενο loop.

Στην συνέχεια θέτουμε το Model Matrix μας ώς μοναδιαίο, επειδή τα vertices μας δεν είναι

normalized οπότε δεν χρειάζεται scaling, υπολογίζουμε τον MVP και τον στέλνουμε στον vertex shader.

Ενεργοποιούμε τα VBOs για το color και vertex με τις εντολές glEnableVertexAttribArray(), glBindBuffer(), glVertexAttribPointer() για κάθε vbo.

Ζωγραφίζουμε τον κύβο με την εντολή glDrawArrays().

Τέλος απενεργοποιούμε τα VBOs.

Σημειώνουμε πως ζωγραφίζουμε και το περίγραμμα του κύβου (πριν ζωγραφίσουμε τον κύβο), για αισθητικούς λόγους, με χρήση των κατάλληλων indeces (που δηλώνουμε manually) μεταξύ των vertex του κύβου, και με την χρήση της glDrawElements().

(ii) Δεν υλοποιήθηκε.

(iii) Για την δημιουργία της κύριας σφαίρας χρησιμοποιήσαμε τον αλγόριθμο που παρατίθεται στο παρακάτω link: http://www.songho.ca/opengl/gl_sphere.html

Η υλοποίηση διαφέρει ελαφρώς.

Ακολουθεί μια σύντομη περιγραφή του αλγορίθμου όπως τον υλοποιήσαμε:

Χωρίζουμε την σφαίρα σε κάθετες και οριζόντιες λωρίδες, που ονομάζουμε stacks και sectors αντίστοιχα. Οι τομές μεταξύ των stacks και των sectors είναι τα vertices (κορυφές) του κύκλου.

Για κάθε stack υπολογίζονται όλες οι τομές με κάθε sector ως τριάδες (x,y,z) και αποθηκέυονται σε ένα πίνακα από floats που καλούμε vertices[].

Ακόμα για κάθε stack υπολογίζονται για κάθε sector και οι συντεταγμένες των texture, ως διάδες απο floats και αποθηκεύονται σε πίνακα που καλούμε texCoords[].

Οι αναλυτικοί τύποι για τον υπολογισμό τον vertices και texCoords αναγράφονται στο παραπάνω link.

Στην συνέχεια αποθηκεύουμε δύο πίνακες χρωμάτων στο ίδιο μέγεθος με τον πίνακα vertices. Περισσότερα για τους πίνακες χρωμάτων θα αναφέρουμε κατα την περιγραφή του ερωτήματος (v).

Τέλος υπολογίζουμε τα indices του κύκλου για να καταγράψουμε τις συνδέσεις μεταξύ των vertices, ώστε κατά το rendering να τα συνδέσουμε στα κατάλληλα τρίγωνα ώς εξής:

Κάθε sector σε ένα stack χρειάζεται 2 τρίγωνα. Εάν το πρώτο index στο παρόν stack είναι το k1 και το στο επόμενο stack το k2, τότε τα τελευταία indices αντίθετα με την φορά του

ρολογιού για 2 τρίγωνα είναι:

k1 -> k2 -> k1+1

k1+1 -> k2 -> k2+1

Σημειώνεται ότι το κορυφαίο και χαμηλότερο stack χρειάζονται μόνο ένα τρίγωνο.

Έπειτα δεσμεύουμε τους πίνακες indices, vertices και texCoords, καθώς και τους δύο πίνακες χρωμάτων σε VBOs, στον VAO που έχουμε ήδη δεσμεύσει.

Πριν ζωγραφίσουμε την σφαίρα, κάνουμε tranform τον Model Matrix μας στο σημείο που θέλουμε να βρίσκεται το κέντρο της σφαίρας, υπολογίζουμε τον νέο MVP μας και τον περνάμε στον vertex shader.

Κατά το rendering ενεργοποιούμε τα VBOs και το IBO με τις εντολές glEnableVertexAttribArray(), glBindBuffer(), glVertexAttribPointer() για κάθε vbo. Σημειώνουμε πως δεσμεύουμε τον IBO με την εντολή glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, iboCubeId).

Τέλος ζωγραφίζουμε την σφαίρα με την εντολή glDrawElements() και απενεργοποιούμε τους VBOs.

Όσον αφορά την κίνηση της σφαίρας δεσμεύουμε μια callback μέθοδο key_callback() μέσω της εντολής glfwSetKeyCallback(window, key_callback) η οποία καλείται κάθε φορά που πατείται ένα πλήκτρο.

Αν το πλήκτρο είναι <ΑΡΙΣΤΕΡΑ_ΒΕΛΑΚΙ,ΔΕΞΙΑ_ΒΕΛΑΚΙ,ΠΑΝΩ_ΒΕΛΑΚΙ,ΚΑΤΩ_ΒΕΛΑΚΙ,+,-> αλλάζει μια global μεταβλητή trasformSphereX,trasformSphereY και trasformSphereZ αντίστοιχα η οποία αυξομειώνει το translate της εντολής

ModelMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(trasformSphereX, trasformSphereY, trasformSphereZ)) η οποία εντολή καθορίζει που θέλουμε να γίνει render το κέντρο της σφαίρας.

(iv) Για το υποερώτημα αυτό χρησιμοποιήσαμε κώδικα του tutorial. Προσθέσαμε κώδικα για το zoom in, zoom out.

Επίσης στην κίνηση δεξια, αριστερά, πάνω, κάτω, προσθέσαμε rotate αυξομειώνοντας τις μεταβλητές horizontalAngle και verticalAngle αντίστοιχα.

Γνωρίζουμε πως η υλοποίηση του υποερωτήματος είναι λάθος. Προσπαθήσαμε να υλοποιήσουμε τον παρακάτω κώδικα που είναι σωστός αλλά πάλι είχε θέμα:

float rotCamera = 0.0f;

```
glm::vec3 origPos = *getPosition();

if (glfwGetKey( window, GLFW_KEY_D ) == GLFW_PRESS){

        rotCamera += 0.01;

        glm::vec3 rot(cos(rotCamera) * 10.0 - origPos.x,  0, sin(rotCamera) * 10.0 -
        origPos.z);

        setPosition(origPos + rot);

}
```

και έπειτα:

```
ViewMatrix = glm::lookAt(

        position,          // Camera is here

        glm::vec3( 50, 50, 50 ), //To look always to the cube

        up                 // Head is up (set to 0,-1,0 to look upside-down)

 );
```

(v) Για την εμφάνιση του texture χρησιμοποιούμε την stb_image.

Την κάνουμε include ως:

```
#define STB_IMAGE_IMPLEMENTATION

#include "stb_image.h"
```

Για την φόρτωση της εικόνας χρησιμοποιούμε τον κώδικα που μας δίνεται στο tutorial.

Προσθέσαμε την εντολή glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_R, GL_REPEAT) για να γίνεται επανάληψη της εικόνας στο αντικείμενο στον τρισδιάστατο παρά στον δισδιάστατο χώρο.

Επίσης χρησιμοποιήσαμε την εντολή glPixelStorei(GL_UNPACK_ALIGNMENT, 1), για να θέσουμε την pixel storage mode σε byte-alignment.

Επειδή θέλουμε το texture να ενεργοποιείται και να απενεργοποιείται, δηλώνουμε μια global boolean textureEnabled που αλλάζει τιμή όταν πατείται το πλήκτρο <T>.

Στην key_callback(), ελέχγουμε για το πάτημα του πλήκτρου <T> και αλλάζουμε την τιμή της textureEnabled.

Κατά το rendering ελέγχεται εάν η μεταβλητή textureEnabled είναι true ή false.

Εάν είναι true ενεργοποιείται το vbo texCoords και το vbo για τα χρώματα, που περιλαμβάνει το χρώμα άσπρο σε κάθε vertex.

Εάν είναι false ενεργοποιείται μόνο το vbo για τα χρώματα που περιλαμβάνει το χρώμα κόκκινο.

Αυτό συμβαίνει επειδή στον fragment shader μας, το τελικό χρώμα των pixels υπολογίζεται ως textColor = texture(ourTexture, texCoord)*vec4(fragmentColor,1);

Εάν το texCoords VBO δεν ενεργοποιηθεί το texture(ourTexture, texCoord) γίνεται το default μοναδιαίο διάνυσμα όποτε το τελικό χρώμα είναι μόνο το χρώμα που έχει υπολογιστεί στον vertex shader (στην προκειμένη το κόκκινο).

Εάν το texCoords ενεργοποιηθεί τότε παράλληλα ενεργοποιείται και το VBO με το λευκο χρώμα (1,1,1) οπότε το vec4(fragmentColor,1) γίνεται μοναδιαίο διάνυσμα και το textColor είναι ίσο με texture(ourTexture, texCoord).

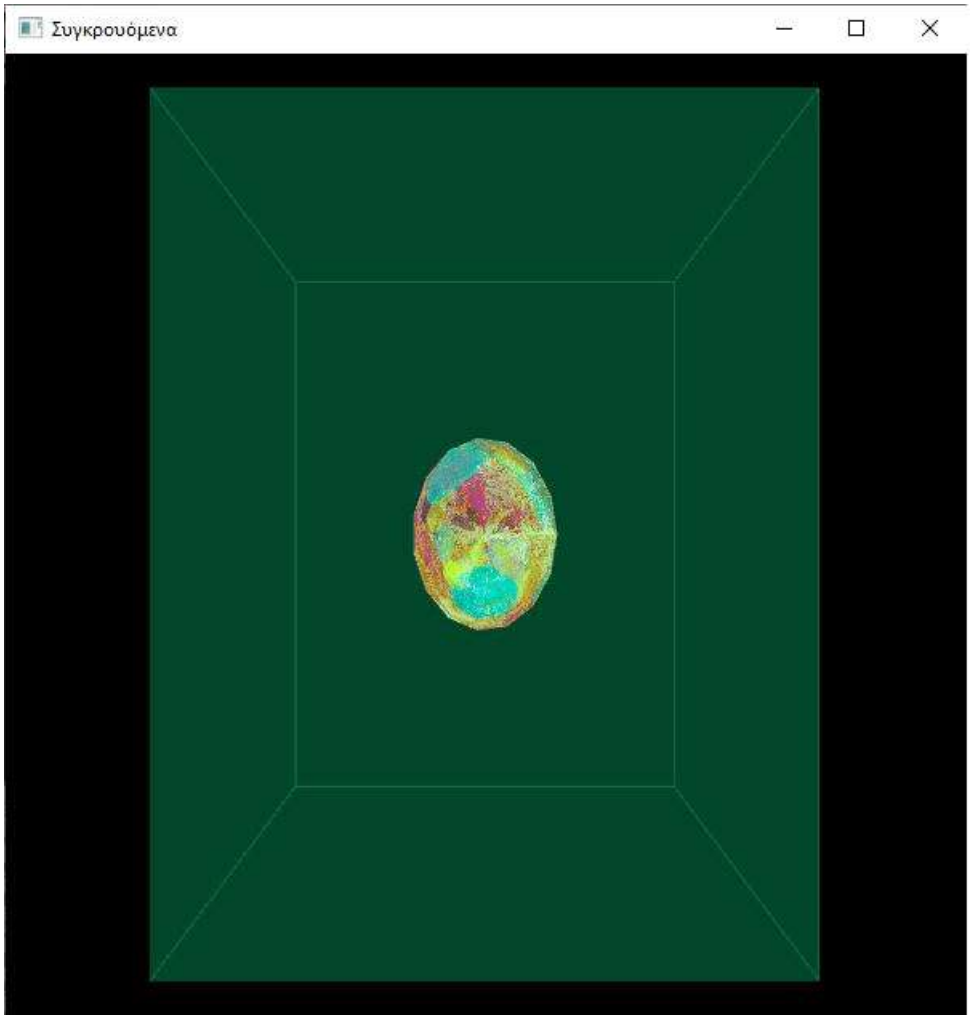
Τεχνικά:

Όλα τα .h και .hpp βρίσκονται στον φάκελο include ο οποίος βρίσκεται στον φάκελο του εκτελέσιμου.

Οι φάκελοι των βιβλιοθηκών γίνονται link από τον φάκελο External Resources τον οποίο είχαμε τοποθετημένο στο προηγούμενο φάκελο από αυτον του solution. Τον φάκελο External Resources δεν τον παρέχουμε.

Ο κώδικας δεν έχει ελεγχεί σε άλλους υπολογιστές.

Η εικόνα του τι φαίνεται κατα την εκκίνηση του προγράμματος.


Συγκρουόμενα