ΓΡΑΦΙΚΑ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΣΥΣΤΗΜΑΤΑ ΑΛΛΗΛΕΠΙΔΡΑΣΗΣ

Β ΜΕΡΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΩΝ ΑΣΚΗΣΕΩΝ

ΟΜΑΔΑ:

ΙΩΑΝΝΑ ΑΠΟΣΤΟΛΟΠΟΥΛΟΥ Α.Μ. 2935

ΧΡΙΣΤΙΝΑ ΟΛΥΜΠΙΑ ΣΟΛΔΑΤΟΥ Α.Μ. 4001

Αρχικά να αναφέρουμε ότι το έδαφος και τον κρατήρα δεν τα φορτώνουμε ως αρχεία .obj όπως τη μπάλα αλλά φτιάχνουμε τους πίνακες με τα vertices και τα uv.

Για το έδαφος:

Για το πλέγμα χρειάζονται 20*20 *2 τρίγωνα.

Σε μία for loop γεμίζουμε τον πίνακα ground_vertices[] με τις συντεταγμένες των κορυφών των τριγώνων του πλέγματος, κάθε κορυφή την βάζουμε μία φορά καθώς στη συνέχεια φτιάχνουμε index buffer. Οι κορυφές είναι 21*21=441. Ξεκινάμε από την κορυφή (0,100,0) και μειώνοντας το y κατά 5 αποθηκεύουμε τις πρώτες 21 κορυφές που βρίσκονται στο x=0, στη συνέχεια αυξάνουμε το x κατά 5 και το y κατά 100 για να αποθηκεύσουμε τις επόμενες 21 κορυφές που βρίσκονται στο x=5.Τελευταία αποθηκεύεται η κορυφή (100,0,0) .

Κώδικας:

```
float x = 0.0f;
float y = 100.0f;
float z = 0.0f;
int v = 0;
//-----GROUND XYZ coords
GLfloat ground_vertices[21 * 21 * 3] = { 0 };
for (int i = 0; i < 21 * 21 * 3; i += 3) {
    ground_vertices[i] = x;
    ground_vertices[i + 1] = y;
    ground_vertices[i + 2] = z;
    y = y - 5.0f;
    v = v + 1;
    if ((v) % 21 == 0) {
        x = x + 5.0f;
        y = 100.0f;
    }
}
GLuint ground_vertexbuffer;
glGenBuffers(1, &ground_vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, ground_vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(ground_vertices), ground_vertices, GL_STATIC_DRAW);
```

Στη συνέχεια φτιάχνουμε τον πίνακα με τα uv, για κάθε κορυφή (x,y,z) έχουμε και ένα (u,v). Οι uv συντεταγμένες είναι από 0 εώς 1 οπότε διαιρούμε με το 100 τις x,y συνιστώσες που είναι από 0 εώς 100 για να τις αντιστοιχίσουμε στο (0,1).

Κώδικας:

```
//-----GROUND  UV coords
y = 100.0f;
x = 0.0f;
v = 0;
GLfloat ground_uvs[21 * 21 * 2] = { 0 };
for (int i = 0; i < 21 * 21 * 2; i += 2) {
    ground_uvs[i] = float(x) / 100.0f;
    ground_uvs[i + 1] = float(y) / 100.0f;
    y = y - 5.0f;
    v = v + 1;
    if ((v) % 21 == 0) {
        x = x + 5.0f;
        y = 100.0f;
    }
}
GLuint ground_uvbuffer;
glGenBuffers(1, &ground_uvbuffer);
glBindBuffer(GL_ARRAY_BUFFER, ground_uvbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(ground_uvs), ground_uvs, GL_STATIC_DRAW);
```

φτιάχνουμε έναν βοηθητικό πίνακα τον indices[] που περιγράφει κάθε τρίγωνο με τρεις ακεραίους, κάθε ακέραιος αντιστοιχεί σε μία κορυφή.

Το πρώτο τετράγωνο θα σχηματιστεί από το τρίγωνο με κορυφές κ0,κ21,κ22 και το τρίγωνο κ22,κ1,κ0 οπότε στον indices για να περιγράψουμε το πρώτο τρίγωνο βάζουμε τους ακεραίους 0,21,22 και για το δεύτερο τρίγωνο 22,1,0.

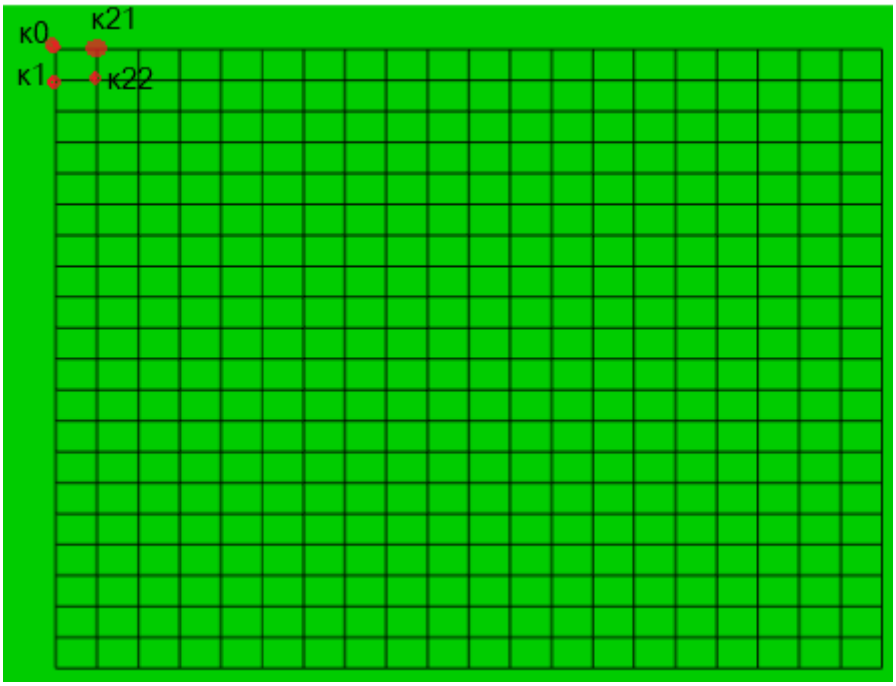
Είναι:

κ0 =(0,100,0)

κ1=(0,95,0)

κ21=(5,100,0)

κ22=(5,95,0)



Η λογική είναι η ίδια για όλα τα τετράγωνα του πλέγματος .Οι ακέραιοι που πρέπει να αποθηκεύσουμε στον indices είναι (400 τετράγωνα * 2 τρίγωνα * 3 ακέραιοι κάθε τρίγωνο).
Κώδικας:

```
//kathe trigwno perigrafetai apo treis akeraious, enan akeraio gia kathe korifi
//o pinakas indices exei tis perigrafes twn 800 trigwnwn tou plegmatos
GLuint indices[400 * 2 * 3] = { 0 };
int vertice = 0;
for (int i = 0; i < 400 * 2 * 3; i += 6) {
    if (vertice % 21 == 20) {
        vertice += 1;
    }
    indices[i] = vertice;
    indices[i + 1] = vertice + 21;
    indices[i + 2] = vertice + 22;
    indices[i + 3] = vertice + 22;
    indices[i + 4] = vertice + 1;
    indices[i + 5] = vertice;
    vertice += 1;
}
GLuint element_buffer;
glGenBuffers(1, &element_buffer);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, element_buffer);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
```

Στη συνέχεια φορτώνουμε το texture για το έδαφος:

```
// -----GROUND texture load
int width1, height1, nrChannels1;
unsigned char* ground_data = stbi_load("ground1.jpg", &width1, &height1, &nrChannels1, 0);

if (!ground_data)
{
    std::cout << "Failed to load texture" << std::endl;
}

GLuint ground_textureID;
glGenTextures(1, &ground_textureID);
glBindTexture(GL_TEXTURE_2D, ground_textureID);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width1, height1, 0, GL_RGB, GL_UNSIGNED_BYTE, ground_data);
glGenerateMipmap(GL_TEXTURE_2D);
GLuint Ground_TextureID = glGetUniformLocation(programID, "ground_texture");
```

Για τον σχεδιασμό του εδάφους μέσα στο do_while loop αντί για την εντολή glDrawArrays χρησιμοποιούμε την εντολή:

```
    );
    glDrawElements(GL_TRIANGLES, 2400,GL_UNSIGNED_INT,(void*)0);
```

<u>ΜΠΑΛΕΣ ΦΩΤΙΑΣ:</u>

Φορτώνουμε το texture της μπάλας και το αρχείο ball.obj για να πάρουμε τα vertices και τα uv:

```
// ------BALL texture load
int width3, height3, nrChannels3;
unsigned char* ball_data = stbi_load("ball.jpg", &width3, &height3, &nrChannels3, 0);

if (!ball_data)
{
    std::cout << "Failed to load texture" << std::endl;
}

GLuint ball_textureID;
glGenTextures(1, &ball_textureID);
glBindTexture(GL_TEXTURE_2D, ball_textureID);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width3, height3, 0, GL_RGB, GL_UNSIGNED_BYTE, ball_data);
glGenerateMipmap(GL_TEXTURE_2D);
GLuint Ball_TextureID = glGetUniformLocation(programID, "ball_texture");
```

```
// Read ball.obj file
std::vector<glm::vec3> ball_vertices;
std::vector<glm::vec3> ball_normals;
std::vector<glm::vec2> ball_uvs;
bool res = loadOBJ("ball.obj", ball_vertices, ball_uvs, ball_normals);

GLuint ball_vertexbuffer;
glGenBuffers(1, &ball_vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, ball_vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, ball_vertices.size() * sizeof(glm::vec3), &ball_vertices[0], GL_STATIC_DRAW);

GLuint ball_uvbuffer;
glGenBuffers(1, &ball_uvbuffer);
glBindBuffer(GL_ARRAY_BUFFER, ball_uvbuffer);
glBufferData(GL_ARRAY_BUFFER, ball_uvs.size() * sizeof(glm::vec2), &ball_uvs[0], GL_STATIC_DRAW);
```

Έχουμε ξεχωριστό model matrix για την μπάλα. Αυξάνουμε το μέγεθος της μπάλας(scale) και την μετατοπίζουμε(translate) σε ένα τυχαίο x,y και στο z=20.

```
//arxiki thesi tis mpalas, to x kai to y einai tyxaia, to z einai 20.
xb = (static_cast <float> (rand()) / static_cast <float>(RAND_MAX)) * 100;
yb = (static_cast <float> (rand()) / static_cast <float>(RAND_MAX)) * 100;
//gia na megalosei kai na metatopistei h mpala
ModelMatrix_ball = glm::translate(glm::mat4(1.0), glm::vec3(xb, yb, 20.0f)) * glm::scale(glm::mat4(1.0f), glm::vec3(5.0f, 5.0f, 5.0f));
```

Η μπάλα θα εμφανιστεί όταν πατηθεί το πλήκτρο B και θα αρχίσει να πέφτει. Αυτό γίνεται μετατοπίζοντας τη θέση της μπάλας μόνο στο z με βήμα -0.05 μέχρι να γίνει z=1.25, τα x,y δεν αλλάζουν . Όταν η μπάλα ακουμπίσει στο έδαφος(το z γίνει 1.25, η μπάλα έχει διάμετρο 2.5), σταματά ο σχεδιασμός της και την μετατοπίζουμε σε νέο τυχαίο x,y και z=20 ,όταν ξαναπατηθεί το B επαναλμβάνεται η διαδικασία. Στη διάρκεια που η μπάλα πέφτει αν πατηθεί το u μειώνουμε το z κατά 0.10 σε κάθε σχεδιάση για να φαίνεται πιο γρήγορη η κίνηση της μπάλας ενώ αν πατηθεί το p το μειώνουμε κατά 0.01, για να φαίνεται πιο αργή η κίνηση.
Για τις κινήσεις της μπάλας φτιάξαμε την ball_function() που καλείται μέσα do-while loop πριν τον κώδικα για τον σχεδιασμό της.
Κώδικας στο do-while loop:

```
// ----BALL------
ball_function();
if (ball_falling) {
    MVP = Projection * View * ModelMatrix_ball;
    glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, ball_textureID);
    glUniform1i(Ball_TextureID, 0);

    // 1rst attribute buffer : vertices
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, ball_vertexbuffer);
    glVertexAttribPointer(
        0,
        3,
        GL_FLOAT,
        GL_FALSE,
        0,
        (void*)0
    );
```

```
    // 2nd attribute buffer : UVs
    glEnableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER, ball_uvbuffer);
    glVertexAttribPointer(
        1,
        2,
        GL_FLOAT,
        GL_FALSE,
        0,
        (void*)0
    );
    //gia ton sxediasmo tis mpalas
    glDrawArrays(GL_TRIANGLES, 0, ball_vertices.size());

}
```

Η boolean μεταβλητη ball_falling γίνεται true από τη συνάρτηση ball_function αν πατηθεί το B. Κώδικας ball_function():

```
//sinartisi gia tis kiniseis ts mpalas
//gia na peftei i mpala ginetai translation - meiwnetai to z
void ball_function() {
    if (glfwGetKey(window, GLFW_KEY_B) == GLFW_PRESS) {
        if (!ball_falling) {
            zb_step = -0.05f;
            ball_falling = true;
        }
    }
    if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS) {
        if(ball_falling){
            //afksisi taxititas
            zb_step = -0.10f;
        }
    }
    if (glfwGetKey(window, GLFW_KEY_P) == GLFW_PRESS) {
        if (ball_falling) {
            //meiwsi taxititas
            zb_step = -0.01f;
        }
    }
    zb += zb_step;
    ModelMatrix_ball = glm::translate(glm::mat4(1.0), glm::vec3(xb, yb, zb)) * glm::scale(glm::mat4(1.0f), glm::vec3(5.0f, 5.0f, 5.0f));
```

```
    if (zb <= 1.25) {
        //an i mpala akoumpisei to edafos
        // kaleitai i sinartisi gia tous kratires, gia na prostethei mia akomi thesi metatopisis(emfanisis) tou kratira
        crater_function(xb,yb);
        //nees tyxaies sintetagmenes tis mpalas
        xb = (static_cast <float> (rand()) / static_cast <float>(RAND_MAX)) * 100;
        yb = (static_cast <float> (rand()) / static_cast <float>(RAND_MAX)) * 100;
        //to z ginetai pali 20
        zb = 20.0f;
        //neos pinakas metasximatismou
        ModelMatrix_ball = glm::translate(glm::mat4(1.0), glm::vec3(xb, yb, 20.0f)) * glm::scale(glm::mat4(1.0f), glm::vec3(5.0f, 5.0f, 5.0f));
        //to vima 0 gia na menei statheri i mpala
        zb_step = 0.0f;
        //ball_falling = false gia na min sxediazetai mexri na ksanapatithei to B
        ball_falling = false;

    }
}
```

Αν πατήθηκε το B (όχι κατά τη διάρκεια που πέφτει η μπάλα) κάνουμε το zb_step =-0.05 για το translate και το ball_falling =true. Επίσης αν πατηθεί το u ή το p αναλόγως ενημερώνουμε το zb_step ώστε να γίνει η αλλαγή στην “ταχύτητα”. Το zb_step προστίθεται στο zb που είναι η συνιστώσα z της θέσης της μπάλας. Στη συνέχεια ενημερώνεται το Model Matrix της μπάλας. Αν η μπάλα ακουμπίσει στο έδαφος ενημερωνουμε τα xb,yb με νέες τυχαίες τιμές ώστε όταν πατηθεί το B ξανά να εμφανιστεί η μπάλα σε άλλη θέση, κάνουμε πάλι το zb =20, ενημερώνεται το model matrix της μπάλας και κάνουμε το ball_falling=false για να μην σχεδιάζεται η μπάλα μέχρι να ξαναπατηθεί το B. Η crater_function() που καλέιται όταν η μπάλα ακουμπίσει το έδαφος θα εξηγηθεί παρακάτω καθώς αφορά τον σχεδιασμό του κρατήρα στη θέση που επεσε η μπάλα.

ΚΡΑΤΗΡΕΣ:

Ο κρατήρας είναι ένα τετράγωνο (δύο τρίγωνα) με πλευρά 5. Οι κορυφές του κρατήρα είναι οι (0, 0, 0.02) , (0, 5, 0.02) , (5, 5, 0.02), (5, 0, 0.02), (0, 0, 0.02).

Το z του κρατήρα είναι ελάχιστα μεγαλύτερο από το z του εδάφους.

Πίνακας :

```
//-----CRATER  XYZ coords
GLfloat crater_vertices[] = {
    0.0f , 5.0f, 0.02f,
    5.0f , 5.0f, 0.02f,
    5.0f , 0.0f, 0.02f,
    5.0f , 0.0f, 0.02f,
    0.0f , 0.0f, 0.02f,
    0.0f , 5.0f, 0.02f
};
GLuint crater_vertexbuffer;
glGenBuffers(1, &crater_vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, crater_vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(crater_vertices), crater_vertices, GL_STATIC_DRAW);
```

Ο πίνακας με τις uv συντεταγμένες του κρατήρα είναι:

```
//----- CRATER UV coords
GLfloat crater_uvs[] = {
    0.0f , 1.0f,
    1.0f , 1.0f,
    1.0f , 0.0f,
    1.0f , 0.0f,
    0.0f , 0.0f,
    0.0f , 1.0f
};

GLuint crater_uvbuffer;
glGenBuffers(1, &crater_uvbuffer);
glBindBuffer(GL_ARRAY_BUFFER, crater_uvbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(crater_uvs), &crater_uvs, GL_STATIC_DRAW);
```

Φορτώνουμε το texture του κρατήρα:

```
//------ CRATER  texture load
int width2, height2, nrChannels2;
unsigned char* crater_data = stbi_load("crater.jpg", &width2, &height2, &nrChannels2, 0);

if (!crater_data)
{
    std::cout << "Failed to load texture" << std::endl;
}

GLuint crater_textureID;
glGenTextures(1, &crater_textureID);
glBindTexture(GL_TEXTURE_2D, crater_textureID);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width2, height2, 0, GL_RGB, GL_UNSIGNED_BYTE, crater_data);
glGenerateMipmap(GL_TEXTURE_2D);
GLuint Crater_TextureID = glGetUniformLocation(programID, "crater_texture");
```

Έχουμε έναν κρατήρα τον οποίο κάνουμε μετατόπιση στις θέσεις που πέφτει η μπάλα.
Σ’ έναν πίνακα κρατάμε τις θέσεις στις οποίες πρέπει να εμφανίζεται ένας κρατήρας δηλαδή σε κάθε θέση της μπάλας (μόνο τις x,y συνιστώσες της μπάλας, το z του κρατήρα δεν αλλάζει) και σε ένα for loop μέσα στο do-while loop σχεδιάζουμε τον κρατήρα σε όλες τις θέσεις φτιάχνωντας τον πίνακα μετατόπισης του κρατήρα για κάθε μία θέση.
Κώδικας στο do-while loop:
Το craters_num είναι το πλήθος των κρατήρων (αυξάνεται όταν η μπάλα πέσει πρώτη φορά σε μία θέση).
Το ModelMatrix_crater είναι ο πίνακας μετατόπισης του κρατήρα στην i-οστή θέση της μπάλας.
Ο πίνακας crater_trans[] ενημερώνεται στη συνάρτηση crater_function(float x_b, float y_b).

```
// ----CRATERS------
for (int i = 0; i < craters_num; i++) {
    ModelMatrix_crater = glm::translate(glm::mat4(1.0), crater_trans[i]);
    MVP = Projection * View * ModelMatrix_crater;
    glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, crater_textureID);
    glUniform1i(Crater_TextureID, 0);
    // 1rst attribute buffer : vertices
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, crater_vertexbuffer);
    glVertexAttribPointer(
        0,
        3,
        GL_FLOAT,
        GL_FALSE,
        0,
        (void*)0
    );
```

```
    // 2nd attribute buffer : UVs
    glEnableVertexAttribArray(1);
    glBindBuffer(GL_ARRAY_BUFFER, crater_uvbuffer);
    glVertexAttribPointer(
        1,
        2,
        GL_FLOAT,
        GL_FALSE,
        0,
        (void*)0
    );
    //gia ton sxediasmo twn kratirwn
    glDrawArrays(GL_TRIANGLES, 0, 2*3);
}
```

Κώδικας συνάρτηση crater_function(float x_b, float y_b):

```
//gia tous kratires
//o pinakas crater_trans[] periexei tis theseis stis opoies prepei na metatopisoume ton kratira diladi ekei pou peftei i mpala
//craters_num einai to plithos twn kratirwn
void crater_function(float x_b, float y_b) {
    float xmod = fmod(x_b, 5.0f);
    float ymod = fmod(y_b, 5.0f);
    float x_new, y_new;
    //vriskoume to pio kontino pollaplasio tou 5 twn x,y tis mpalas
    //epeidi ta x,y twn korifwn tou plegmatos einai pollaplasia tou 5
    if (xmod < 2.5f) {
        x_new = x_b - xmod;
    }
    else {
        x_new = x_b + (5.0f - xmod);
    }
    if (ymod < 2.5f) {
        y_new = y_b - ymod;
    }
    else {
        y_new = y_b + (5.0f - ymod);
    }
```

```
//epeidi o kratiras tha metatopistei kata x_new sto x kai kata y_new sto y, gia na min vgei ektos plegmatos
if (x_new == 100.0f) {
    x_new = 95.0f;
}
if (y_new == 100.0f) {
    y_new = 95.0f;
}
//an exei ksanapesei sto ido simeio i mpala na min apothikeftei pali i thesi
for (int i = 0; i < craters_num; i++) {
    if (x_new==crater_trans[i].x && y_new == crater_trans[i].y) {
        break;
        return;
    }
}
crater_trans.push_back(glm::vec3(x_new, y_new, 0.0f));
craters_num += 1;
}
```

Η συνάρτηση παίρνει ως όρισμα τις συντεταγμένες x,y της μπάλας και καλείται όταν η μπάλα ακουμπίσει το έδαφος. Βρίσκουμε το πιο κοντινό πολλαπλάσιο του 5 στα x,y και το τετράγωνο στο οποίο εφαρμόζουμε τον κρατήρα είναι αυτό με κορυφές (x_new,y_new,0.02) , (x_new,y_new+5,0.02), (x_new+5,y_new+5,0.02), (x_new+5,y_new,0.02) .Αν η μπάλα βρίσκεται πρώτη φορά σ' αυτή τη θέση την κρατάμε στο πίνακα crater_trans και αυξάνουμε τον μετρητή των κρατήρων.

ΚΑΜΕΡΑ:

Αυξήσαμε το far από το 100 στο 500.

```
// Projection matrix : 30° Field of View, 4:3 ratio, display range : 0.1 unit <-> 500 units
glm::mat4 Projection = glm::perspective(glm::radians(30.0f), 4.0f / 3.0f, 0.1f, 500.0f);
```

Τοποθετήσαμε την κάμερα μετά από πειράματα στη θέση:

```
glm::vec3 camera_position = glm::vec3(21.5f, -109.4f, 134.8f);
```

```
    View = glm::lookAt(
        camera_position,   // θέση της κάμερας
        glm::vec3(50, 50, 0),    // η κάμερα κοιτάζει εδώ
        glm::vec3(0, 0, 1)     //up vector
    );
}
```

Ο κώδικας που αφορά τις κινήσεις της κάμερας είναι ίδιος με το προηγούμενο μέρος 1Β.

SHADERS:

Ο Vertex shader παίρνει ως input τις uv συντεταγμένες και τις περνάει στον fragment shader.
Ο fragment shader καλεί τη συνάρτηση texture και έχει σαν output το τελικό χρώμα.

Προβληματισμοί:Ο κρατήρας φαίνεται να μην εφαρμόζει όπως θα έπρεπε στα τετράγωνα.

Πληροφορίες σχετικά με την υλοποίηση:

- Λειτουργικό Σύστημα : Windows 10 Home
- Περιβάλλον: Visual Studio x86

Πηγές και βοήθεια: συμμετοχή στις ώρες εργαστηρίου,γενικά google , www.opengl-tutorial.org, learnopengl.com, www.songho.ca/opengl