Πανεπιστήμιο Ιωαννίνων
Τμήμα Μηχ. Η/Υ και Πληροφορικής

ΜΥΥ702
Γραφικά Υπολογιστών & Συστήματα Αλληλεπίδρασης

Διδάσκοντες: Ιωάννης Φούντος, Βασιλική Σταμάτη

Αναφορά Προγραμματιστικής Άσκησης 1-Γ

Ομάδα:
Παναγιώτης Βουζαλής 2653

Χειμερινό Εξάμηνο 2022

Περιεχόμενα

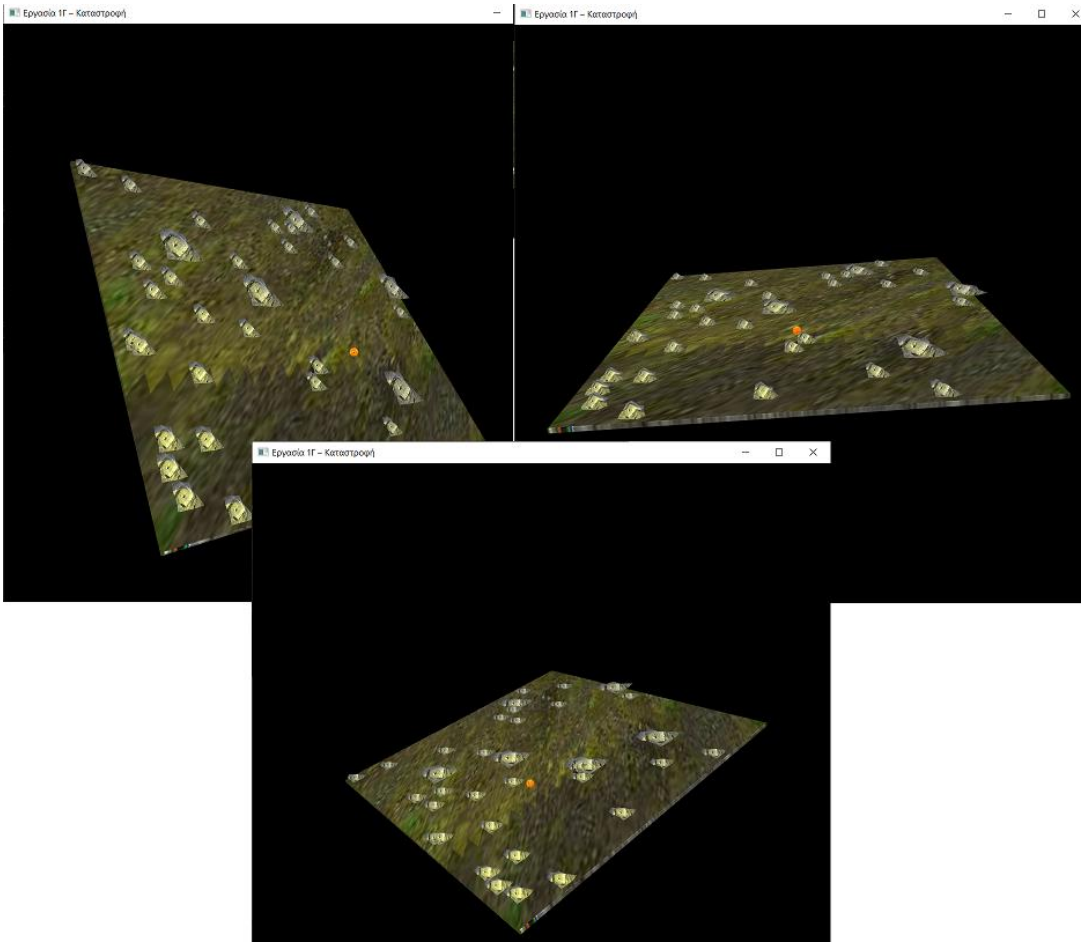
Χαρακτηριστικά Υλοποίησης

Λειτουργικό Σύστημα: Windows 10 Enterprise LTSC Version 21H2

Περιβάλλον Υλοποίησης:
Microsoft Visual Studio 2022 x86 Community Edition

Η υλοποίησή μας στηρίζεται στο υλικό που μας δόθηκε (αρχείο “Proj1C-files.zip”, αρχείο “obj_and_textures_example.zip”), καθώς και στο tutorial 08 της OpenGL (github.com/opengl-tutorials/ogl/tree/master/tutorial08_basic_shading) το οποίο κληθήκαμε να ακολουθήσουμε.

Συνοπτική παρουσίαση της εφαρμογής



Ερώτημα (i)

Καλούμαστε να δημιουργήσουμε το βασικό παράθυρο της εφαρμογής μας, μεγέθους 1000x1000 pixels, και με τίτλο “Εργασία 1Γ - Καταστροφή”.

```
#define windowWidth 1000

#define windowHeight 1000

.............
// Open a window and create its OpenGL context
    window = glfwCreateWindow(windowWidth, windowHeight, u8"Εργασία 1Γ – 
Καταστροφή", NULL, NULL);
```

Το background πρέπει να είναι σκούρο μπλε.

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

Χρησιμοποιώντας το πλήκτρο space οποιαδήποτε στιγμή έχοντας ως ενεργό παράθυρο αυτό της εφαρμογής μας, η εφαρμογή τερματίζει.

```
} // Check if the SPACE key was pressed or the window was closed
    while( glfwGetKey(window, GLFW_KEY_SPACE ) != GLFW_PRESS &&
glfwWindowShouldClose(window) == 0 );
```

Ερώτημα (ii)

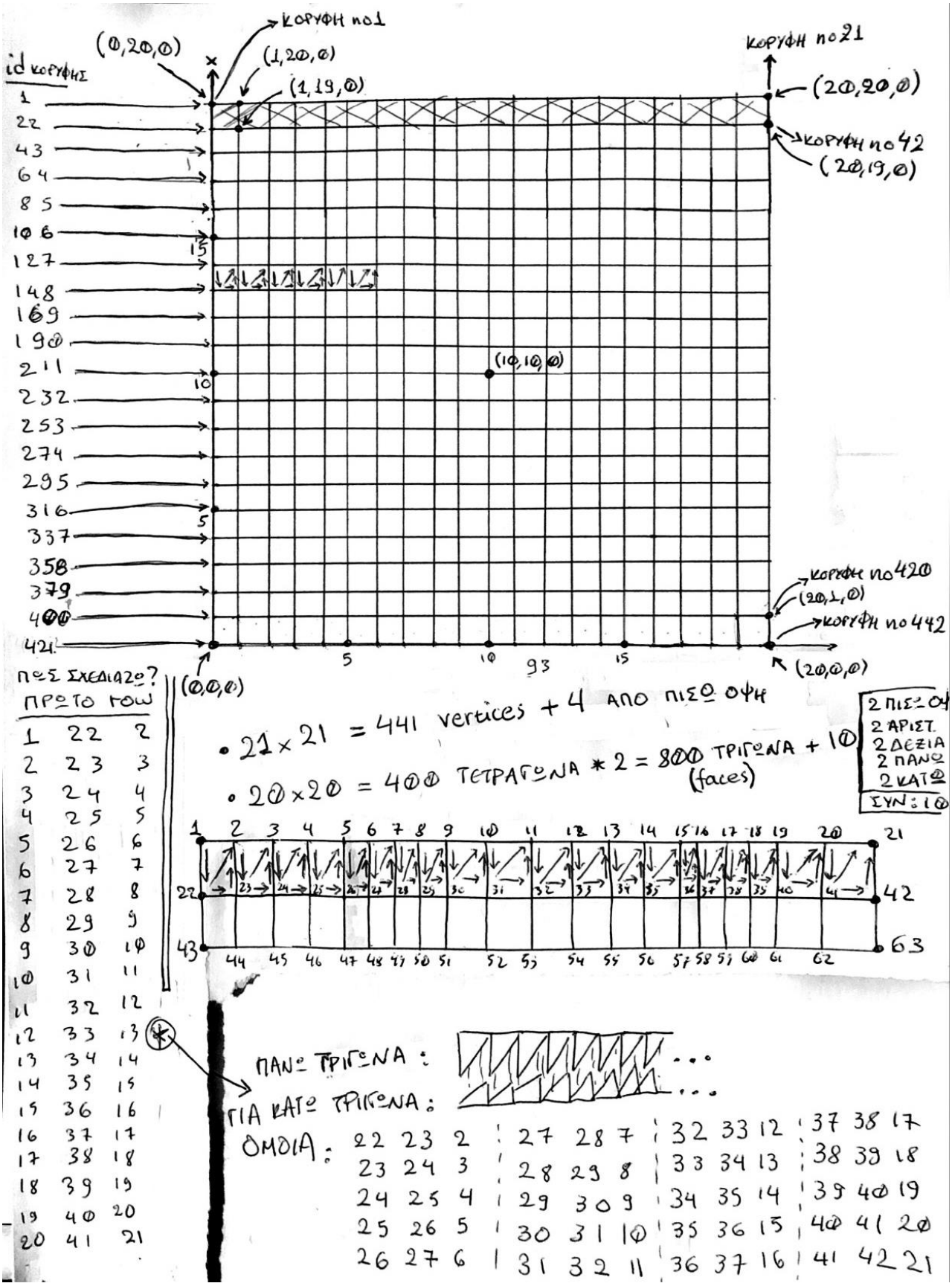
Σε αυτό το ερώτημα καλούμαστε να φορτώσουμε το έδαφος της εφαρμογής. Θα το αναπαραστήσουμε με ένα πλέγμα απο 400 τετράγωνα (20x20) και 800 τρίγωνα. Το πλέγμα θα έχει μία γωνία στην αρχή των αξόνων και θα επεκτείνεται προς τους θετικούς άξονες x και y. Επιπλέον, στο πλέγμα θα εφαρμόσουμε texture.

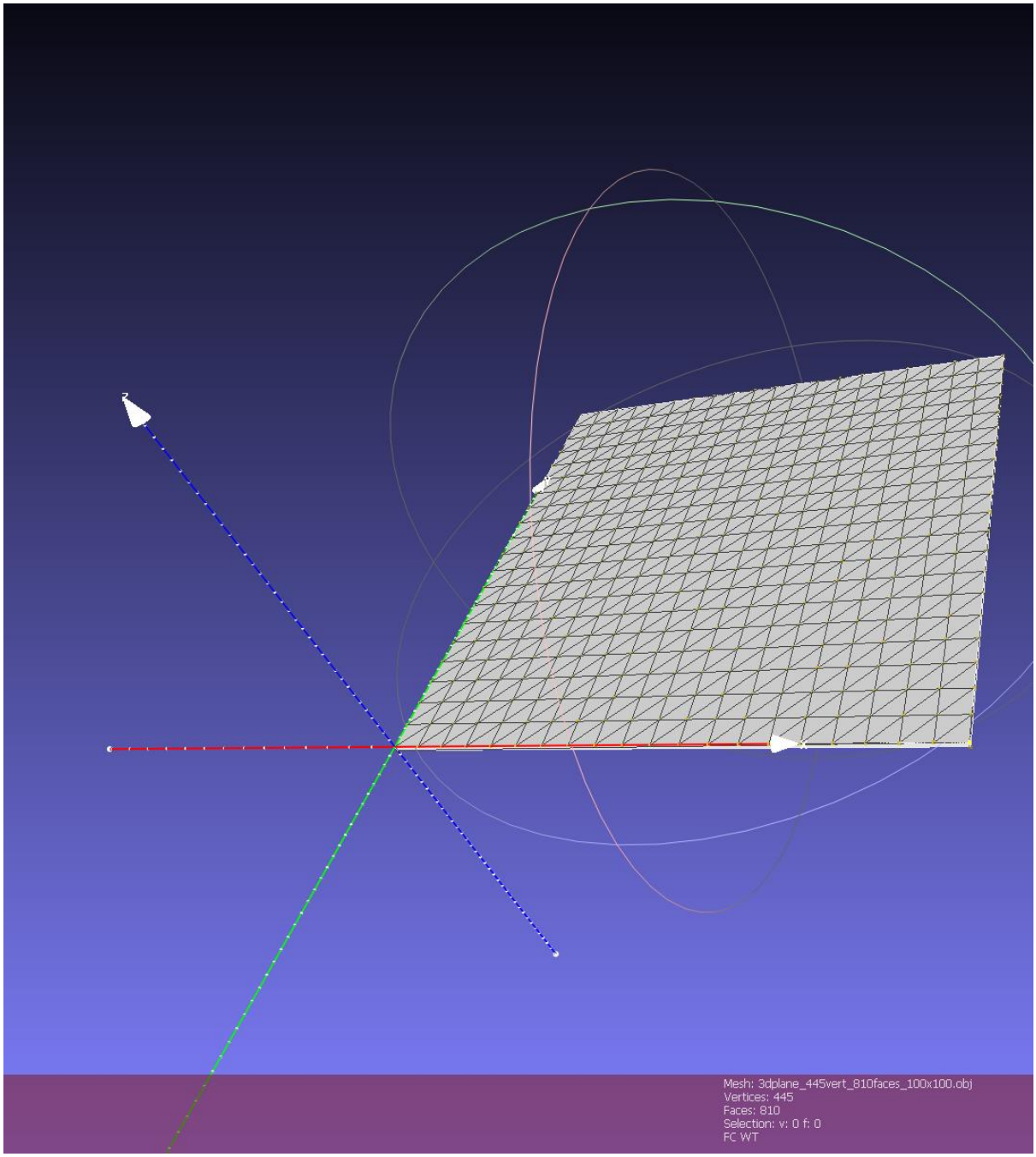
Όλα ξεκίνησαν με το σχεδιασμό της ιδέας μας στο χαρτί, και τον προβληματισμό του πώς θα σχεδιάσουμε τα τρίγωνα με σωστό τρόπο.

Έπειτα ήρθε ο σχεδιασμός του actual επιπέδου στο meshlab και το export του σε αρχείο .obj. Για τις ανάγκες της σωστής επίδειξης της λειτουργίας της εφαρμογής μας αλλά και για το λόγο ότι επιθυμούμε να αλλάζει η μορφολογία του εδάφους μετά από κάθε σύγκρουση με fireball, αποφασίσαμε το grid επιπέδου μας να μην είναι 2D αλλα 3D.

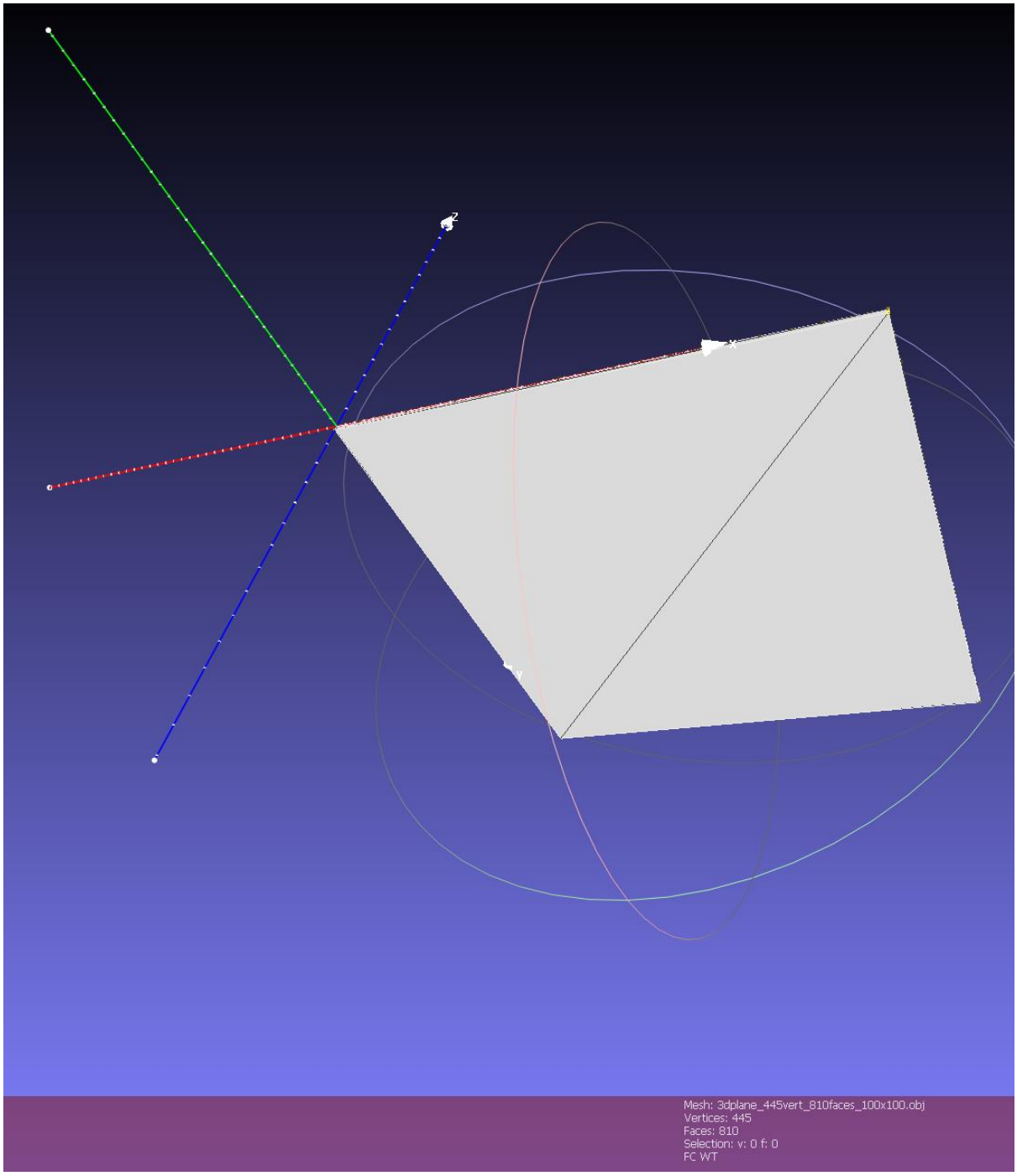
Παρακάτω παρουσιάζεται η δημιουργία του επιπέδου μας, το οποίο θα έχει τελικές διαστάσεις 20x20 τετράγωνα, θα αποτελείται από 445 ακμές (από τις αρχικές 441) και 810 τρίγωνα (από τα αρχικά 800), και θα έχει “πάχος” 1.

Στην επόμενη εικόνα διακρίνεται και το σκεπτικό μας σχετικά με τη σειρά σχεδίασης των τριγώνων.

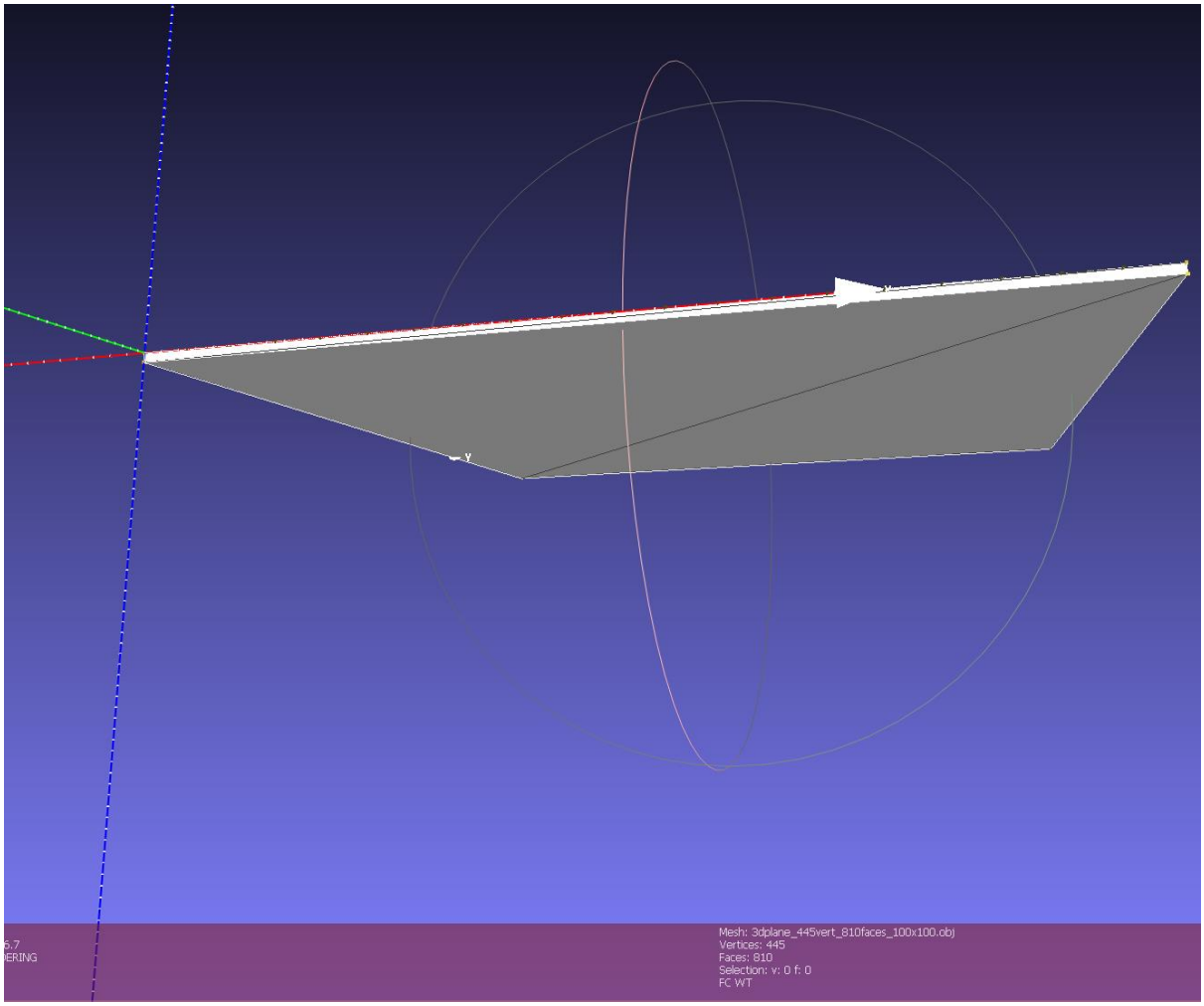

id κορυφής
(0,20,0)
κορυφή no1
(1,20,0)
(1,19,0)
κορυφή no21
(20,20,0)
κορυφή no42
(20,19,0)
(10,10,0)
κορυφή no420
(20,1,0)
κορυφή no442
(20,0,0)
(0,0,0)
• 21 × 21 = 441 vertices + 4 ΑΠΟ ΠΙΣΩ ΟΨΗ
• 20 × 20 = 400 ΤΕΤΡΑΓΩΝΑ * 2 = 800 ΤΡΙΓΩΝΑ + 10 (faces)
2 ΠΙΣΩ ΟΨΗ
2 ΑΡΙΣΤ.
2 ΔΕΞΙΑ
2 ΠΑΝΩ
2 ΚΑΤΩ
ΣΥΝ: 10
ΠΩΣ ΣΧΕΔΙΑΖΩ?
ΠΡΩΤΟ ΤΟΩ
ΠΑΝΩ ΤΡΙΓΩΝΑ :
ΓΙΑ ΚΑΤΩ ΤΡΙΓΩΝΑ :
ΟΜΟΙΑ :
22 23 2 | 27 28 7 | 32 33 12 | 37 38 17
23 24 3 | 28 29 8 | 33 34 13 | 38 39 18
24 25 4 | 29 30 9 | 34 35 14 | 39 40 19
25 26 5 | 30 31 10 | 35 36 15 | 40 41 20
26 27 6 | 31 32 11 | 36 37 16 | 41 42 21



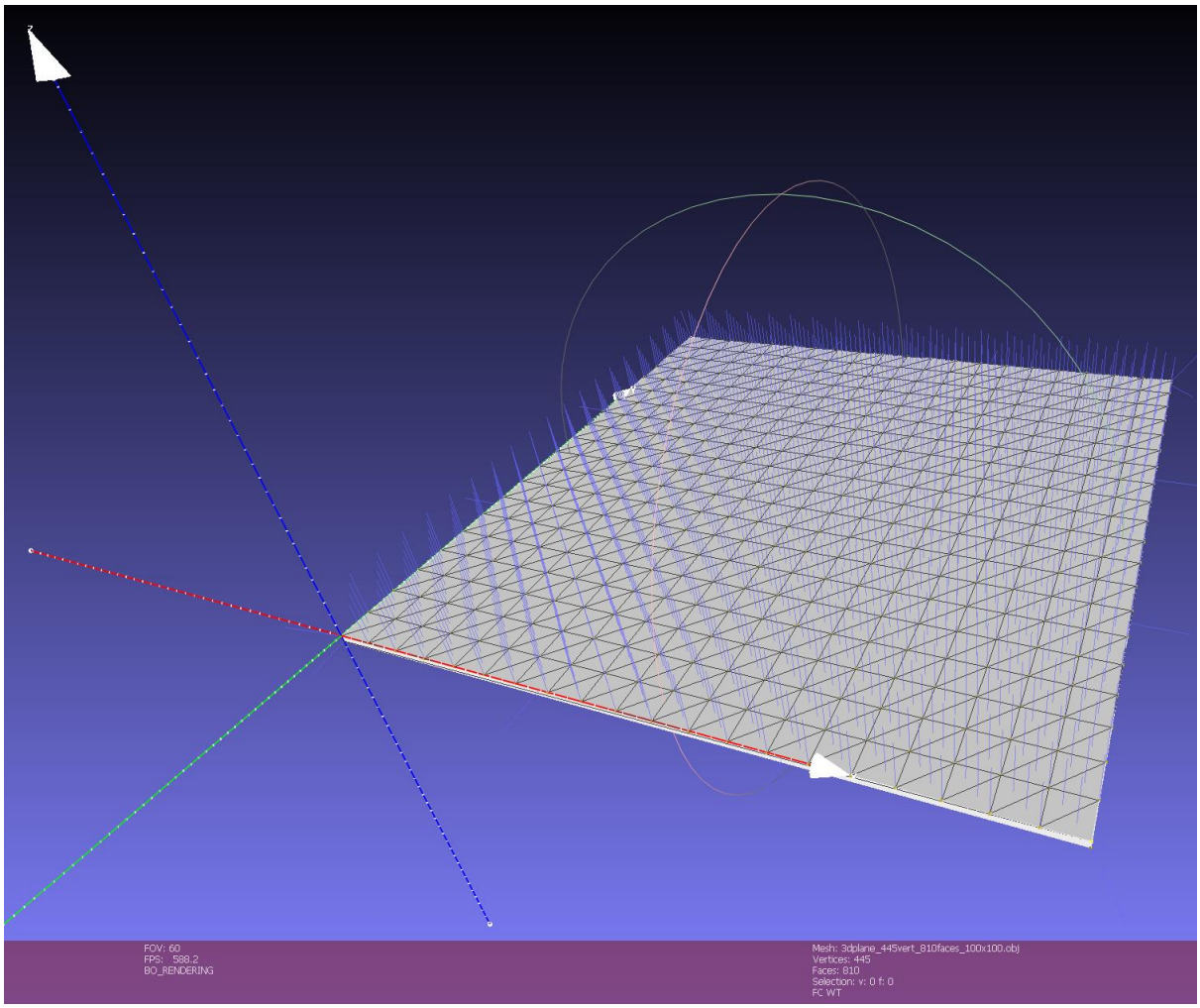
Επάνω face του grid(normal προς το +Z). Διακρίνονται 441 ακμές και 800 τρίγωνα.



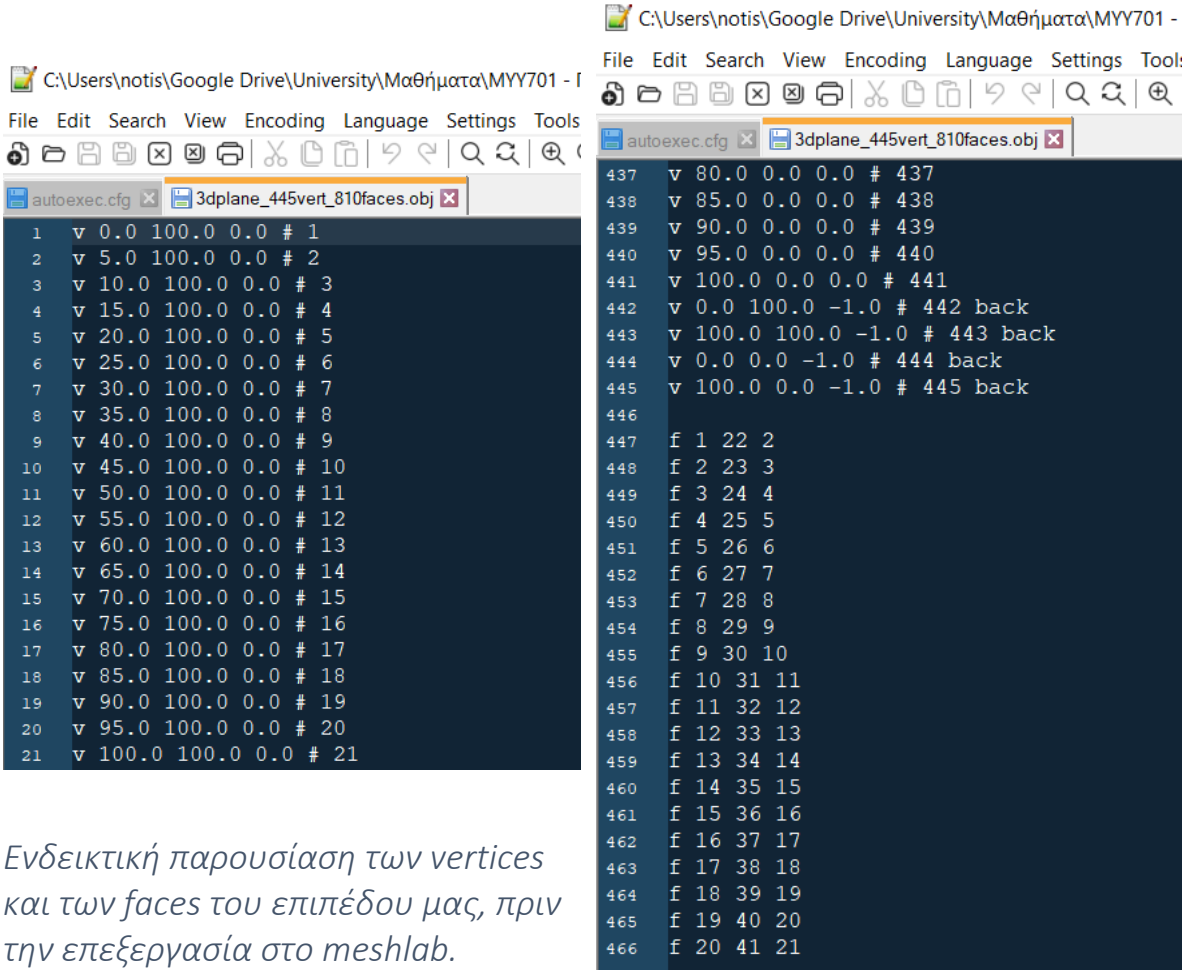
Κάτω face του grid(normal προς το -Z). Διακρίνονται 4 ακμές και 2 τρίγωνα.



Μπροστά face του grid(normal προς το -Y). Διακρίνονται 4 ακμές και 2 τρίγωνα. Αντίστοιχα σχεδιάζονται και οι 3 υπόλοιπες πλευρές.

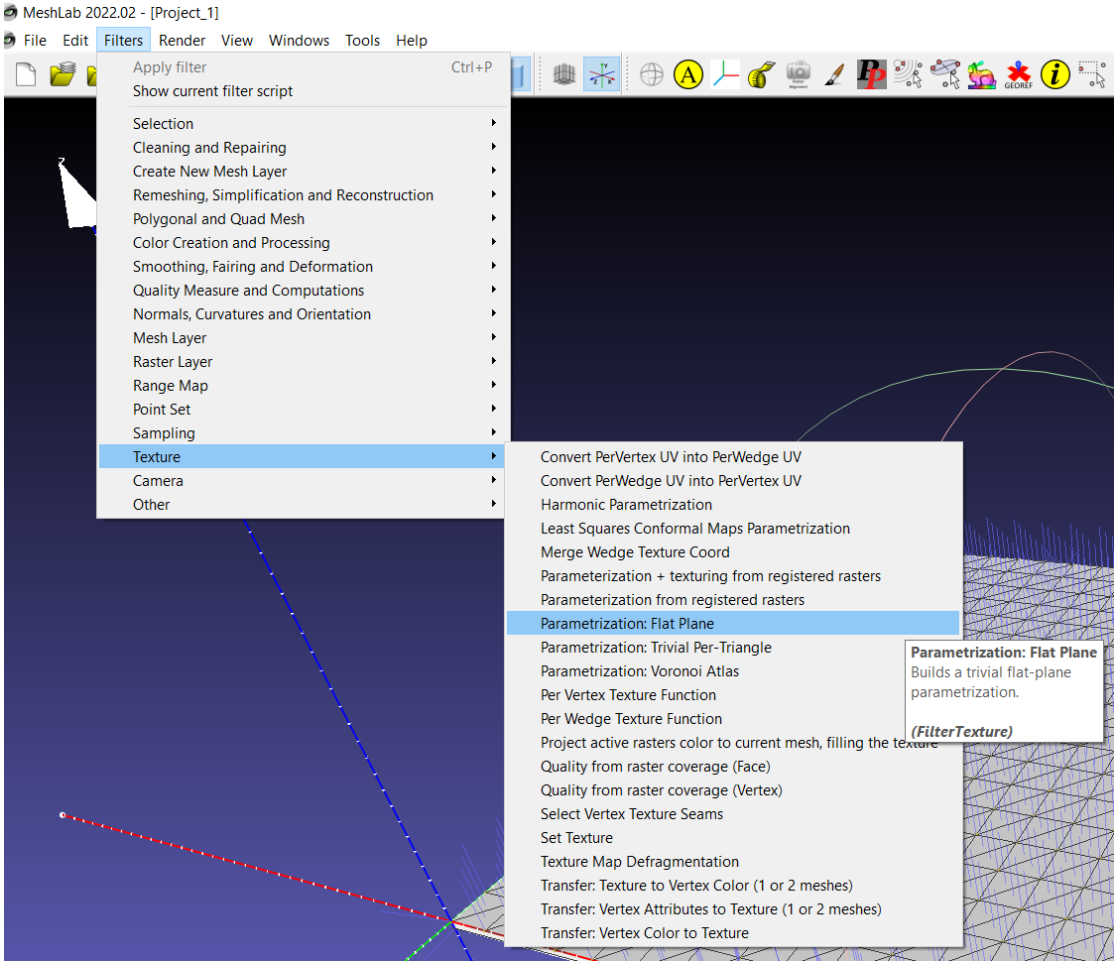


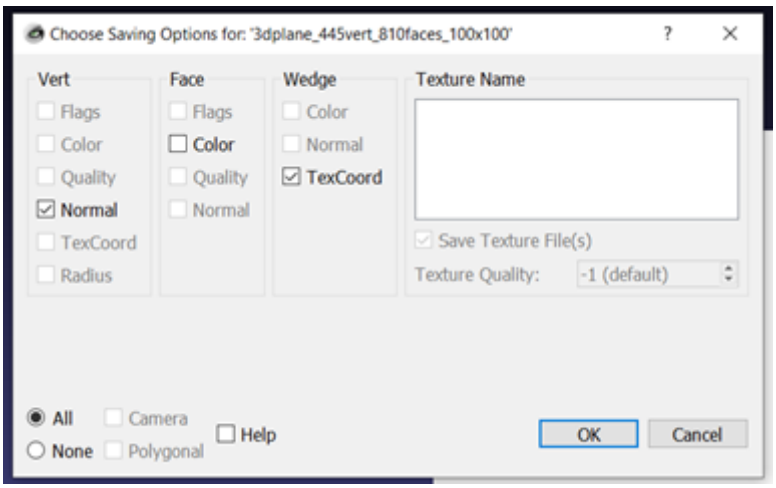
Ενδεικτική άποψη του grid με τα normals ορατά.



Ενδεικτική παρουσίαση των vertices και των faces του επιπέδου μας, πριν την επεξεργασία στο meshlab.
(αρχείο obj primitives/3dplane_445vert_810faces.obj)

Για να εφαρμόσουμε το texture που επιθυμούμε στο έδαφός μας, ακολουθούμε τον τρόπο που μας έδειξε η κ. Σταμάτη στο εργαστήριο:





Τελικά, πάμε στον κώδικά μας και του υπαγορεύουμε να κάνει load το αρχείο 3dplane_445vert_810faces.obj καθώς και το texture που επιλέγουμε:

```
// Load the textures
    int width[number_of_objects], height[number_of_objects],
nrChannels[number_of_objects];
    unsigned char* data[number_of_objects];
    // Load the ground obj
    data[0] = stbi_load("textures/ground1.jpg", &width[0], &height[0],
&nrChannels[0], 0);
```

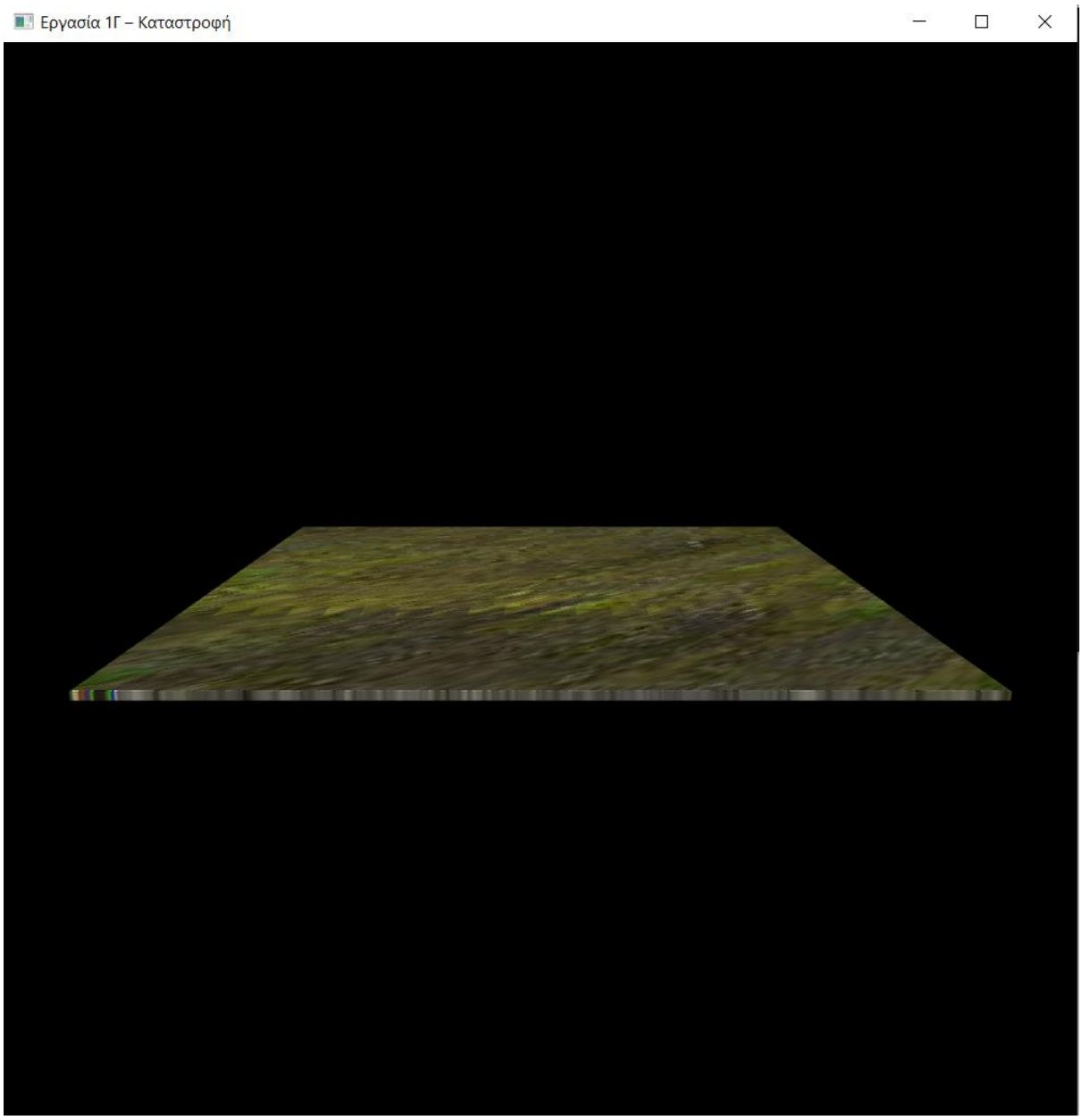
```
// Read our .obj files
    bool resOBJS[number_of_objects];
    // Load the ground obj texture
    resOBJS[0] = loadOBJ("objs/3dplane_445vert_810faces_100x100.obj",
vertices[0], uvs[0], normals[0]);
```

```
do{

        // Clear the screen
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // Use our shader
        glUseProgram(programID);

        // Compute the MVP matrix from keyboard and mouse input
        computeMatricesFromInputs();
        glm::mat4 ProjectionMatrix = getProjectionMatrix();
        glm::mat4 ViewMatrix = getViewMatrix();
        glm::mat4 ModelMatrix = glm::mat4(1.0);
        glm::mat4 MVP = ProjectionMatrix * ViewMatrix * ModelMatrix;

        // Send our transformation to the currently bound shader in the "MVP"
uniform
        glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
        glUniformMatrix4fv(ModelMatrixID, 1, GL_FALSE, &ModelMatrix[0][0]);
        glUniformMatrix4fv(ViewMatrixID, 1, GL_FALSE, &ViewMatrix[0][0]);
```

```
glUniform3f(LightID, lightPosition.x, lightPosition.y, lightPosition.z);

        // Bind our texture in Texture Unit 0
        glActiveTexture(GL_TEXTURE0);

        //glBindTexture(GL_TEXTURE_2D, textureID);
        glBindTexture(GL_TEXTURE_2D, textureID[0]);

        // Set our "myTextureSampler" sampler to use Texture Unit 0
        glUniform1i(uniformTextureID[0], 0);

        // 1rst attribute buffer : vertices
        glEnableVertexAttribArray(0);
        glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer[0]);
        glVertexAttribPointer(
            0,                  // attribute
            3,                  // size
            GL_FLOAT,           // type
            GL_FALSE,           // normalized?
            0,                  // stride
            (void*)0            // array buffer offset
        );

        // 2nd attribute buffer : UVs
        glEnableVertexAttribArray(1);
        glBindBuffer(GL_ARRAY_BUFFER, uvbuffer[0]);
        glVertexAttribPointer(
            1,                                // attribute uv
            2,                                // size
            GL_FLOAT,                         // type
            GL_FALSE,                         // normalized?
            0,                                // stride
            (void*)0                          // array buffer offset
        );

        // 3rd attribute buffer : normals
        glEnableVertexAttribArray(2);
        glBindBuffer(GL_ARRAY_BUFFER, normalbuffer[0]);
        glVertexAttribPointer(
            2,                              // attribute normals
            3,                              // size
            GL_FLOAT,                       // type
            GL_FALSE,                       // normalized?
            0,                              // stride
            (void*)0);                      // array buffer offset
```

```
        // Draw the plane
        glDrawArrays(GL_TRIANGLES, 0, vertices[0].size());
```

Τελικά, με το που ανοίξει η εφαρμογή μας θα αντικρίσουμε το grid εδάφους μας:



Ερώτημα (iii)

(αρχείο “controls.cpp”)

Καλούμαστε να τοποθετήσουμε την κάμερα ώστε να κοιτάει προς το σημείο - κέντρο του grid μας. Στην περίπτωσή μας αυτό είναι το σημείο (50, 50, 0.5). Ανιόν διάνυσμα θα είναι το (0.0, 0.0, 1.0). FoV επιλέγουμε 60 μοίρες.

```
unsigned int cameraMode = 1;

// Initial camera position : on +Z
// Camera coords in in world space
glm::vec3 cameraPosition = glm::vec3(50.0f, -75.0f, 30.0f);
// Camera looks at (10.0, 10.0, -0.5)
glm::vec3 cameraCenterPoint = glm::vec3(50.0f, 50.0f, 0.5f);
// Camera zooms in/out in relation to the center of the grid
glm::vec3 cameraZoomReferencePoint = glm::vec3(50.0f, 50.0f, 0.5f);

float zoomingStep = 0.1f;

// Camera is using an up vector of (0.0, 0.0, 1.0) - head looks at z axis
int differentCameraAngle = 0; // alternate camera view flag

glm::vec3 upVector = glm::vec3(0.0f, 0.0f, 1.0f);

// Initial Field of View - WAS 45
float initialFoV = 60.0f;

// 1 degree = 0.0174532925 radians
float degree = glm::radians(0.0174532925f);
// Rotation "speed" is 10 degrees
float rotationAngle = 10.0f * degree;
```

Καλούμαστε να υλοποιήσουμε λειτουργικότητα που θα περιστρέφει την κάμερα γύρω απο τους άξονες x με τα πλήκτρα <w> και <x> και z με τα πλήκτρα <a> και <d>.

Επίσης επιθυμούμε λειτουργία zoom in/out με τα πλήκτρα <+> και <-> του numpad, η οποία θα έχει ως σημείο αναφοράς το κέντρο του grid.

Επιπρόσθετα από τα ζητούμενα του ερωτήματος, υλοποιήσαμε και μία εναλλακτική άποψη της κάμερας, η οποία ενεργοποιείται με το πλήκτρο <2> του number row του πληκτρολογίου. Για επιστροφή στην αρχική άποψη της κάμερας, μπορούμε να πατήσουμε οποιαδήποτε στιγμή το πλήκτρο <1> του number row του πληκτρολογίου.

Η λειτουργία της κάμερας παρουσιάζεται εκτενώς στις επόμενες σελίδες.

```
unsigned int cameraMode = 1;

// Initial camera position : on +Z
// Camera coords in in world space
glm::vec3 cameraPosition = glm::vec3(50.0f, -75.0f, 30.0f);
// Camera looks at (10.0, 10.0, -0.5)
glm::vec3 cameraCenterPoint = glm::vec3(50.0f, 50.0f, 0.5f);
// Camera zooms in/out in relation to the center of the grid
glm::vec3 cameraZoomReferencePoint = glm::vec3(50.0f, 50.0f, 0.5f);
float zoomingStep = 0.1f;

int differentCameraAngle = 0; // alternate camera view flag

// Camera is using an up vector of (0.0, 0.0, 1.0) - head looks at z axis
glm::vec3 upVector = glm::vec3(0.0f, 0.0f, 1.0f);
// Initial Field of View - WAS 45
float initialFoV = 60.0f;

// 1 degree = 0.0174532925 radians
float degree = glm::radians(0.0174532925f);
// Rotation "speed" is 10 degrees
float rotationAngle = 10.0f * degree;

glm::mat4 getViewMatrix() { return ViewMatrix; }
glm::mat4 getProjectionMatrix() { return ProjectionMatrix; }
glm::vec3 getCameraPosition() { return cameraPosition; }

// Rotation on x axis means that the coords y,z of the yz plane change while x
coord stays the same
glm::mat3 rotationMatrixX = mat3(
    1, 0, 0,
    0, cos(rotationAngle), -sin(rotationAngle),
    0, sin(rotationAngle), cos(rotationAngle)
);

// Rotation on z axis means that the coords x,y of the xy plane change while z
coord stays the same
glm::mat3 rotationMatrixZ = mat3(
    cos(rotationAngle), -sin(rotationAngle), 0,
    sin(rotationAngle), cos(rotationAngle), 0,
    0, 0, 1
);
```

```
void computeMatricesFromInputs() {

    glm::vec3 normalizedVector = glm::normalize((cameraPosition -
cameraCenterPoint));
    // Zoom in
    if (glfwGetKey(window, GLFW_KEY_KP_ADD) == GLFW_PRESS) { cameraPosition -=
zoomingStep * normalizedVector; }
    // Zoom out
    if (glfwGetKey(window, GLFW_KEY_KP_SUBTRACT) == GLFW_PRESS) {
cameraPosition += zoomingStep * normalizedVector; }

    // Rotate counterclockwise in regards to x axis
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
        cameraPosition = rotationMatrixX * cameraPosition;
        upVector = rotationMatrixX * upVector;
    }
    // Rotate clockwise in regards to x axis
    if (glfwGetKey(window, GLFW_KEY_X) == GLFW_PRESS) {
        cameraPosition = cameraPosition * rotationMatrixX;
        upVector = upVector * rotationMatrixX;
    }

    // Rotate counterclockwise in regards to z axis
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS) {
        cameraPosition = rotationMatrixZ * cameraPosition;
        upVector = rotationMatrixZ * upVector;
    }
    // Rotate clockwise in regards to z axis
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) {
        cameraPosition = cameraPosition * rotationMatrixZ;
        upVector = upVector * rotationMatrixZ;
    }

    // Change camera angles depending on what number row key is pressed

    if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS) {
        cameraPosition = cameraPosition;
        cameraCenterPoint = glm::vec3(55.0f, 55.0f, 0.5f);
    }

    if (glfwGetKey(window, GLFW_KEY_2) == GLFW_PRESS) {
        cameraPosition = cameraPosition;
        cameraCenterPoint = glm::vec3(0.0f, 0.0f, 0.5f);
    }
```

```
float FoV = initialFoV;// - 5 * glfwGetMouseWheel();

    // Projection matrix : 45° Field of View, 1:1 ratio, display range : 0.1
unit <-> 100 units
    ProjectionMatrix = glm::perspective(glm::radians(FoV), 4.0f / 4.0f, 0.1f,
500.0f);

    // Camera matrix
    ViewMatrix = glm::lookAt(
        cameraPosition,     // Camera is here
        cameraCenterPoint,  // and looks here : at the same position
        upVector            // Camera is using the modified up vector
    );

    // Rotate on x axis means that the coords y,z of the yz plane change while
x coord stays the same
    // Rotate on z axis means that the coords x,y of the xy plane change while
z coord stays the same
    //printf("X: %f, Y: %f, Z: %f\n", cameraPosition.x, cameraPosition.y,
cameraPosition.z); // Debugging
```

Ερώτημα (iv)

Σε αυτό το ερώτημα καλούμαστε να δημιουργήσουμε fireballs που θα προσκρούουν στο επίπεδό μας, αλλάζοντας τη μορφολογία του (bonus ερώτημα γ).

Το object της σφαίρας που μας δόθηκε στα αρχεία της άσκησης παρουσιάζεται στο αρχείο “ball.obj”. Έχει διάμετρο 0.5. Επειδή η ανάλυση της οθόνης που χρησιμοποιήσαμε για την ανάπτυξη της εφαρμογής είναι αρκετά μεγάλη, για λόγους καθαρά quality-of-life έτσι ώστε να βλέπουμε καλύτερα τη σφαίρα, επιλέξαμε να χρησιμοποιήσουμε μια σφαίρα με διάμετρο 1 (αρχείο “BALLZ.obj”).

Φυσικά στον κώδικά μας μπορούμε να επιλέξουμε ποιο object σφαίρας θέλουμε να φορτώσουμε όπως φαίνεται παρακάτω:

```
// Load the fireball obj texture
// Slightly bigger ball to not make our eyes hurt
resOBJS[1] = loadOBJ("objs/BALLZ.obj", vertices[1], uvs[1], normals[1]);
// Original ball
// resOBJS[1] = loadOBJ("objs/ball.obj", vertices[1], uvs[1], normals[1]);
```

Η δημιουργία της ελαφρώς μεγαλύτερης σφαίρας έγινε στο meshlab.

Όταν ο χρήστης πατάει το πλήκτρο <b> εμφανίζεται μια σφαίρα σε τυχαίο σημείο με συντεταγμένες x (0 ως 100), y (0 ως 100), z σταθερό = 20:

```
int should_launch_fireball = 0;        // Flag to launch fireball

glm::vec3 calculateFireballSpawnPoint() {

    // Orismos lower kai upper bound gia ta spawn coords tou fireball
    float spawnLowerBound = 0.0f;
    float spawnUpperBound = 100.0f;

    // Choose a random value between [lowerBound, upperBound]
    // We choose the 500th and 1000th number that are provided by the seed
    float x;
    float y;
    for (int i = 0; i < 500; i++) { x = spawnLowerBound + (float)rand() *
(float)(spawnUpperBound - spawnLowerBound) / (float)RAND_MAX; }
    for (int i = 0; i < 1000; i++) { y = spawnLowerBound + (float)rand() *
(float)(spawnUpperBound - spawnLowerBound) / (float)RAND_MAX; }

    glm::vec3 spawnPoint = glm::vec3(x, y, 20.0f); // The fireball always
starts at a height of z = 20
    //glm::vec3 spawnPoint = glm::vec3(20.0, 20.0f, 20.0f); // Debugging
    //std::cout << "Fireball spawn point: " + glm::to_string(spawnPoint) <<
std::endl; // Debugging
    return spawnPoint;
}
```

```
// Launch the fireball
        if (glfwGetKey(window, GLFW_KEY_B) == GLFW_PRESS) {
            should_launch_fireball = 1;
            // Preload some coords for the fireball
            glm::vec3 fireballPos = calculateFireballSpawnPoint();
        }

if (should_launch_fireball == 1) {
      // Draw the fireball
```

Στη σφαίρα θα εφαρμόσουμε υφή φωτιάς “fire.jpg” όπως φαίνεται παρακάτω:

```
    // Load the fireball obj
    data[1] = stbi_load("textures/fire1.jpg", &width[1], &height[1],
&nrChannels[1], 0);
```

Η σφαίρα κινείται σε ευθεία τροχιά προς το έδαφος με σταθερή ταχύτητα και όταν φτάσει στο έδαφος γίνεται έκρηξη και η σφαίρα χάνεται.

Παρακάτω παρουσιάζεται η σχεδίαση της σφαίρας και η πορεία της προς το έδαφος. Η κίνηση προς το έδαφος γίνεται με translate του z coordinate της σφαίρας (μιας που η κίνηση είναι κάθετη όπως λέει η εκφώνηση).

Πρώτα σχεδιάζεται η σφαίρα, ξεκινάει την πορεία της προς το έδαφος, έπειτα γίνεται έλεγχος για πλήκτρα που τροποποιούν την ταχύτητά της (bonus ερώτημα β), και τέλος ελέγχουμε για collision μεταξύ της σφαίρας και του επιπέδου.

Εαν υπάρχει collision τότε δίνεται η εντολή στον κώδικά μας να παίξει το ηχητικό εφέ της έκρηξης (bonus ερώτημα α) και να αλλάξει τη μορφολογία του εδάφους (bonus ερώτημα δ) εφαρμόζοντας ανάλογη υφή “crater.jpg” στον κρατήρα που δημιουργείται.

Όλες αυτές οι λειτουργίες παρουσιάζονται εκτενώς παρακάτω:

```
if (should_launch_fireball == 1) {
            glBindTexture(GL_TEXTURE_2D, textureID[1]);
            glUniform1i(uniformTextureID[1], 0);

            glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer[1]);
            glVertexAttribPointer(
                0,                          // attribute
                3,                          // size
                GL_FLOAT,                   // type
                GL_FALSE,                   // normalized?
                0,                          // stride
                (void*)0);                  // array buffer offset

            glBindBuffer(GL_ARRAY_BUFFER, uvbuffer[1]);
            glVertexAttribPointer(
                1,                          // attribute uv
                2,                          // size
                GL_FLOAT,                   // type
                GL_FALSE,                   // normalized?
                0,                          // stride
                (void*)0);                  // array buffer offset

            glBindBuffer(GL_ARRAY_BUFFER, normalbuffer[1]);
            glVertexAttribPointer(
                2,                          // attribute normals
                3,                          // size
                GL_FLOAT,                   // type
                GL_FALSE,                   // normalized?
                0,                          // stride
                (void*)0);                  // array buffer offset

            glm::mat4 fireballModelMatrix = glm::translate(glm::mat4(1.0f),
fireballPos);

            // Move z fireball coord by zOffset
            glm::mat4 newFireballModelMatrix =
                translate(glm::mat4(1.0f), -glm::vec3(0, 0, fireballPos.z *
zOffset)) * fireballModelMatrix;

            // Calculate MVP
            MVP = ProjectionMatrix * ViewMatrix * newFireballModelMatrix;

            glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
```

```
glUniformMatrix4fv(ModelMatrixID, 1, GL_FALSE, &newFireballModelMatrix[0][0]);
            glUniformMatrix4fv(ViewMatrixID, 1, GL_FALSE, &ViewMatrix[0][0]);

            glUniform3f(LightID, lightPosition.x, lightPosition.y,
lightPosition.z);

            // Draw the fireball
            glDrawArrays(GL_TRIANGLES, 0, vertices[1].size());
            checkForSpeedPresses(); // Adjust the fireball's speed at will
            //printf("Fireball speed: %f\n", zOffset); // Debugging
            //printf("Fireball Z: %f\n", fireballPos.z); // Debugging
            fireballPos.z -= fireballPos.z * zOffset; // The fireball always
comes closer to the ground plane
        }
```

```
float zOffset_initial = 0.001f; // initial speed offset for fireball descent
0.0001f
float zOffset = zOffset_initial; // speed offset for fireball descent
void checkForSpeedPresses() {
    // Make the fireball descend faster or slower

    if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS) {
        zOffset = zOffset * 1.025f;
        //if (zOffset > 0.1f) { zOffset = 0.0001f; } // Reset offset if too
fast
    }
    else if (glfwGetKey(window, GLFW_KEY_P) == GLFW_PRESS) {
        zOffset = zOffset / 1.02f;
        if (zOffset < 0.00001f) { zOffset = 0.0001f; } // Reset offset if too
slow
    }
}
```

```
// If flag is NOT raised then draw the fireball
        if (dont_draw_fireball != 1) { glDrawArrays(GL_TRIANGLES, 0,
vertices[1].size()); }

        // Check if we have a collision
        if (dont_draw_fireball != 1 && checkForCollision(fireballPos,
ground_plane, fireball_radius)) {
            // Play explosion sound
            sndPlaySound(TEXT("C4_explosion.wav"), SND_ASYNC);

            // Halt the launching of fireballs
            should_launch_fireball = 0;

            // Erase the fireball that collided
            dont_draw_fireball = 0;

            // Initialize new coords for the fireball and reset the zOffset
            fireballPos = calculateFireballSpawnPoint();
            zOffset = zOffset_initial;

        }
        checkForCollision(fireballPos, ground_plane, fireball_radius); //
check collision between fireball and ground plane
```

```
bool checkForCollision(glm::vec3 fireballPos, glm::vec3 ground_plane, float
fireball_radius) {
    // Check if the fireball collides with the ground plane

    float distance = fireballPos.z - ground_plane.z; // simple distance of z
coords
    if (distance < fireball_radius) {

        //crater_impact_coords = glm::vec3(fireballPos.x, fireballPos.y, -
1.0f); // complex crater
        crater_impact_coords = glm::vec3(fireballPos.x, fireballPos.y, -1.5f);
// simple crater
        float offset = 5.0f;
        if ((100.0f - fireballPos.x) < offset) { crater_impact_coords =
glm::vec3(crater_impact_coords.x - offset, crater_impact_coords.y,
crater_impact_coords.z); }
        if ((100.0f - fireballPos.y) < offset) { crater_impact_coords =
glm::vec3(crater_impact_coords.x, crater_impact_coords.y - offset,
crater_impact_coords.z); }

        std::cout << "Fireball impact point:" +
glm::to_string(crater_impact_coords) << std::endl; // Debugging
        should_draw_craters_array[craterCounter] = 1;
        craters_coords_array[craterCounter] = crater_impact_coords;
        craterCounter += 1;

        return true;
    }
    return false;
}
```

```
float fireball_radius = 1.0; // Taken from meshlab - BALLZ.obj attributes
glm::vec3 ground_plane = glm::vec3(0.0f, 0.0f, 0.0f);

// Array with flag variables used to indicate whether to draw a crater or not
int should_draw_craters_array[number_of_objects - 2] = {};
// Array with the coords of each crater
glm::vec3 craters_coords_array[number_of_objects - 2] = {};
// Count the number of craters
int craterCounter = 0;
// Initialize a coords vector
glm::vec3 crater_impact_coords = glm::vec3(0.0f, 0.0f, 0.0f);
```

Εάν υπάρξει collision τότε πρέπει να δημιουργήσουμε κρατήρα. Αυτό γίνεται κρατώντας ένα πίνακα απο flags που υποδεικνύουν εάν πρέπει να σχεδιαστεί ένας κρατήρας ή όχι.

Επιπλέον κρατάμε τις συντεταγμένες του κρατήρα σε έναν ανάλογο πίνακα.

Με αυτόν τον τρόπο μπορούμε να επιλέξουμε εμείς μέχρι πόσοι κρατήρες μπορούν να σχεδιάζονται κάθε φορά στην οθόνη μας.

Για τις ανάγκες της άσκησης επιλέξαμε να μπορούμε να εφαρμόσουμε μέχρι 400 μικρούς κρατήρες όταν γίνεται πρόσκρουση fireball και επιπέδου, και 400 μεγάλοι κρατήρες όταν γίνεται πρόσκρουση fireball σε σημείο του επιπέδου στο οποίο ήδη υπάρχει κρατήρας (bonus ερώτημα γ).

```
// Number of objects to be drawn on screen.
// This number includes the crater textures.
// Depending on the number of polygons the crater texture has
// and the number of craters we wish to display
// the startup loading time of our app will increase.
// We create an array, so 0 refers to ground plane, 1 to fireball
// 2 to 802 refer to craters when a fireball hits an area for the 1st time
#define number_of_objects 802
int objCounter = number_of_objects;
#define number_of_big_craters 400
```

```
float fireball_radius = 1.0; // Taken from meshlab - BALLZ.obj attributes
glm::vec3 ground_plane = glm::vec3(0.0f, 0.0f, 0.0f); // Our ground plane sits
at the xy plane where z = 0

// Array with flag variables used to indicate whether to draw a crater or not
int should_draw_craters_array[number_of_objects - 2] = {};
// Array with the coords of each crater
glm::vec3 craters_coords_array[number_of_objects - 2] = {};
// Count the number of craters
int craterCounter = 0;

// Initialize a coords vector
glm::vec3 crater_impact_coords = glm::vec3(0.0f, 0.0f, 0.0f);

// Array with flag variables used to indicate whether to draw a bigger crater
or not
int should_draw_2ndcraters_array[number_of_big_craters] = {};
// Array with the coords of each bigger crater
glm::vec3 second_craters_coords_array[number_of_big_craters] = {};
// Count the number of 2nd craters
int secondCraterCounter = 0;
```

Εφόσον πια έχουμε τις συντεταγμένες του κρατήρα μας το μόνο που έχουμε να κάνουμε είναι να τον σχεδιάσουμε:

```
// Draw craters dynamically only if they should be drawn, i.e the flag
variable is raised
        for (int i = 2; i < number_of_objects; i++) {
            int craterID = i - 2;
            if (should_draw_craters_array[craterID] == 1) {

                // Draw the crater
                glBindTexture(GL_TEXTURE_2D, textureID[i]);
                glUniform1i(uniformTextureID[i], 0);

                glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer[i]);
                glVertexAttribPointer(
                    0,                          // attribute
                    3,                          // size
                    GL_FLOAT,                   // type
                    GL_FALSE,                   // normalized?
                    0,                          // stride
                    (void*)0);                  // array buffer offset

                glBindBuffer(GL_ARRAY_BUFFER, uvbuffer[i]);
                glVertexAttribPointer(
                    1,                          // attribute uv
                    2,                          // size
                    GL_FLOAT,                   // type
                    GL_FALSE,                   // normalized?
                    0,                          // stride
                    (void*)0);                  // array buffer offset

                glBindBuffer(GL_ARRAY_BUFFER, normalbuffer[i]);
                glVertexAttribPointer(
                    2,                          // attribute normals
                    3,                          // size
                    GL_FLOAT,                   // type
                    GL_FALSE,                   // normalized?
                    0,                          // stride
                    (void*)0);                  // array buffer offset

                glm::mat4 craterModelMatrix = glm::mat4(1.0f);

                // Move the crater to the correct position
                glm::mat4 newCraterModelMatrix =
                    translate(glm::mat4(1.0f),
glm::vec3(craters_coords_array[craterID].x, craters_coords_array[craterID].y,
craters_coords_array[craterID].z)) * craterModelMatrix;
```

```
                // Calculate MVP
                MVP = ProjectionMatrix * ViewMatrix * newCraterModelMatrix;

                glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
                glUniformMatrix4fv(ModelMatrixID, 1, GL_FALSE,
&newCraterModelMatrix[0][0]);
                glUniformMatrix4fv(ViewMatrixID, 1, GL_FALSE,
&ViewMatrix[0][0]);

                // Draw the crater
                glDrawArrays(GL_TRIANGLES, 0, vertices[i].size());
            }
        }
```

Με αντίστοιχο τρόπο σχεδιάζονται και οι μεγαλύτεροι κρατήρες όταν πέσει κάποιο fireball σε περιοχή που ήδη υπάρχει κρατήρας.

Κλείνοντας τα βασικά ζητούμενα της άσκησης, αναφέρουμε πως με το που ξεκινάει η main συνάρτηση του κώδικά μας γίνονται κάποιες αρχικοποιήσεις (παρουσιάζονται αυτές που δημιουργήσαμε εμείς):

```
// Initialize our arrays because they contain trash
    for (int i = 2; i < number_of_objects; i++) {
        should_draw_craters_array[i-2] = 0;
        craters_coords_array[i-2] = glm::vec3(0.0f, 0.0f, 0.0f);
    }
    for (int i = 0; i < number_of_big_craters; i++) {
        should_draw_2ndcraters_array[i] = 0;
        second_craters_coords_array[i] = glm::vec3(0.0f, 0.0f, 0.0f);
    }

    // Initialize a seed
    srand(time(NULL));
```

```
// Number of objects to be drawn on screen.
// This number includes the crater textures.
// Depending on the number of polygons the crater texture has
// and the number of craters we wish to display
// the startup loading time of our app will increase.
// We create an array, so 0 refers to ground plane, 1 to fireball
// 2 to 802 refer to craters when a fireball hits an area for the 1st time
#define number_of_objects 802
int objCounter = number_of_objects;
#define number_of_big_craters 400
```

```
    std::vector<glm::vec3> vertices[number_of_objects];
    std::vector<glm::vec3> normals[number_of_objects];
    std::vector<glm::vec2> uvs[number_of_objects];

    std::vector<glm::vec3> vertices_bigC[number_of_big_craters];
    std::vector<glm::vec3> normals_bigC[number_of_big_craters];
    std::vector<glm::vec2> uvs_bigC[number_of_big_craters];
```

Υπενθυμίζουμε πως στην εφαρμογή μας υποστηρίζονται 400 objects μικρού κρατήρα και 400 μεγάλου κρατήρα. Αυτά τα νούμερα φυσικα μπορούν να αλλάξουν προσέχοντας πάντα πως θα υπάρχει ανάλογη επιβάρυνση στο start up time της εφαρμογής.

Για τον σωστό σχεδιασμό των κρατήρων, τόσο των μικρών αλλά και τον μεγάλων, πρέπει αρχικά με την έναρξη της εφαρμογής να γίνουν load τα αντίστοιχα objects και textures, έτσι ώστε να είναι έτοιμα για να σχεδιαστούν όταν αυτό χρειαστεί:

```
// Load the crater objs texts
    for (int i = 2; i < number_of_objects; i++) {
        data[i] = stbi_load("textures/crater1.jpg", &width[i], &height[i],
&nrChannels[i], 0);
    }
    // Load the bigger crater objs texts
    int width_bigC[number_of_big_craters], height_bigC[number_of_big_craters],
nrChannels_bigC[number_of_big_craters];
    unsigned char* data_bigC[number_of_big_craters];
    for (int i = 0; i < number_of_big_craters; i++) {
        data_bigC[i] = stbi_load("textures/crater1.jpg", &width_bigC[i],
&height_bigC[i], &nrChannels_bigC[i], 0);
    }
```

```
// Load the crater objs
    for (int i = 2; i < number_of_objects; i++) {
        resOBJS[i] = loadOBJ("objs/simple_crater.obj", vertices[i], uvs[i],
normals[i]);
    }
    // Load the big crater objs
    bool resOBJS_bigC[number_of_big_craters];
    for (int i = 0; i < number_of_big_craters; i++) {
        resOBJS[i] = loadOBJ("objs/simple_crater_2nd_impact.obj",
vertices_bigC[i], uvs_bigC[i], normals_bigC[i]);
    }
```

Παρακάτω παρουσιάζεται η φόρτωση των object μας στους VBOs:

```
GLuint textureID[number_of_objects];
    GLuint uniformTextureID[number_of_objects]; // handle for uniforms
    GLuint vertexbuffer[number_of_objects];
    GLuint uvbuffer[number_of_objects];
    GLuint normalbuffer[number_of_objects];

    GLuint textureID_bigC[number_of_big_craters];
    GLuint uniformTextureID_bigC[number_of_big_craters]; // handle for
uniforms
    GLuint vertexbuffer_bigC[number_of_big_craters];
    GLuint uvbuffer_bigC[number_of_big_craters];
    GLuint normalbuffer_bigC[number_of_big_craters];
```

```
for (int i = 0; i < number_of_objects; i++) {

        glGenTextures(1, &textureID[i]);

        // "Bind" the newly created texture : all future texture functions
will modify this texture
        glBindTexture(GL_TEXTURE_2D, textureID[i]);

        // Give the image to OpenGL
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width[i], height[i], 0, GL_RGB,
GL_UNSIGNED_BYTE, data[i]);
        glGenerateMipmap(GL_TEXTURE_2D);

        // Get a handle for our "myTextureSampler" uniform
        uniformTextureID[i] = glGetUniformLocation(programID,
"myTextureSampler");

        // Load it into a VBO

        glGenBuffers(1, &vertexbuffer[i]);
        glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer[i]);
        glBufferData(GL_ARRAY_BUFFER, vertices[i].size() * sizeof(glm::vec3),
&vertices[i][0], GL_STATIC_DRAW);

        glGenBuffers(1, &uvbuffer[i]);
        glBindBuffer(GL_ARRAY_BUFFER, uvbuffer[i]);
        glBufferData(GL_ARRAY_BUFFER, uvs[i].size() * sizeof(glm::vec2),
&uvs[i][0], GL_STATIC_DRAW);

        glGenBuffers(1, &normalbuffer[i]);
        glBindBuffer(GL_ARRAY_BUFFER, normalbuffer[i]);
        glBufferData(GL_ARRAY_BUFFER, normals[i].size() * sizeof(glm::vec2),
&normals[i][0], GL_STATIC_DRAW);
    }
```

```
for (int i = 0; i < number_of_big_craters; i++) {

        glGenTextures(1, &textureID_bigC[i]);

        // "Bind" the newly created texture : all future texture functions
will modify this texture
        glBindTexture(GL_TEXTURE_2D, textureID_bigC[i]);

        // Give the image to OpenGL
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width_bigC[i], height_bigC[i],
0, GL_RGB, GL_UNSIGNED_BYTE, data_bigC[i]);
        glGenerateMipmap(GL_TEXTURE_2D);

        // Get a handle for our "myTextureSampler" uniform
        uniformTextureID_bigC[i] = glGetUniformLocation(programID,
"myTextureSampler");

        // Load it into a VBO

        glGenBuffers(1, &vertexbuffer_bigC[i]);
        glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer_bigC[i]);
        glBufferData(GL_ARRAY_BUFFER, vertices_bigC[i].size() *
sizeof(glm::vec3), &vertices_bigC[i][0], GL_STATIC_DRAW);

        glGenBuffers(1, &uvbuffer_bigC[i]);
        glBindBuffer(GL_ARRAY_BUFFER, uvbuffer_bigC[i]);
        glBufferData(GL_ARRAY_BUFFER, uvs_bigC[i].size() * sizeof(glm::vec2),
&uvs_bigC[i][0], GL_STATIC_DRAW);

        glGenBuffers(1, &normalbuffer_bigC[i]);
        glBindBuffer(GL_ARRAY_BUFFER, normalbuffer_bigC[i]);
        glBufferData(GL_ARRAY_BUFFER, normals_bigC[i].size() *
sizeof(glm::vec2), &normals_bigC[i][0], GL_STATIC_DRAW);
    }
```

Bonus Ερωτήματα

Προσθήκη εφέ ήχου στην έκρηξη του fireball

Για την προσθήκη εφέ ήχου κάνουμε include τις κατάλληλες βιβλιοθήκες και απλά δείχνουμε στον κώδικά μας το σημείο που θέλουμε να παίξει το εφέ:

```
// Include windows.h
#include <windows.h>
#include <mmsystem.h>

#pragma comment(lib, "Winmm.lib")
```

```
// Check if we have a collision
        if (checkForCollision(fireballPos, ground_plane, fireball_radius)) {
            // Play explosion sound
            sndPlaySound(TEXT("C4_explosion.wav"), SND_ASYNC);

            ............

        }
```

Προσθήκη φωτισμού

Για την υλοποίηση φωτισμού προστέθηκαν οι εξής γραμμές κώδικα και χρησιμοποιήθηκαν οι shaders “ StandardShading.fragmentshader” και “ StandardShading.vertexshader” από το tutorial 08 της openGL, όπως μας υποδείχθηκε στο εργαστήριο.

Έγιναν μικρές τροποποιήσεις στους shaders, κυρίως για το χρώμα του φωτός που θέλουμε να εκπέμπεται, την έντασή του, και το ambient color των object μας.

```
int main( void ){
    // Create and compile our GLSL program from the shaders
    //GLuint programID = LoadShaders( "TransformVertexShader.vertexshader",
"TextureFragmentShader.fragmentshader" );
    GLuint programID = LoadShaders("StandardShading.vertexshader",
"StandardShading.fragmentshader"); // From OpenGL tutorial 8
```

```
// Get a handle for our "LightPosition" uniform
    glUseProgram(programID);
    GLuint LightID = glGetUniformLocation(programID,
"LightPosition_worldspace");
    .........
    glm::vec3 lightPosition = glm::vec3(0, 0, 100); // Let there be light at
these coords
```

```
do{
      ......... .........
        glUniform3f(LightID, lightPosition.x, lightPosition.y,
lightPosition.z);
      ......... .........
}
```

(StandardShading.fragmentshader)

```
.............
void main(){

    // Light emission properties
    // You probably want to put them as uniforms
    vec3 LightColor = vec3(1,1,0);
    float LightPower = 5000.0f;

    // Material properties
    vec3 MaterialDiffuseColor = texture( myTextureSampler, UV ).rgb;

    vec3 MaterialAmbientColor = vec3(1.0,1.0,1.0) * MaterialDiffuseColor;
    //vec3 MaterialAmbientColor = vec3(0.1,0.1,0.1) * MaterialDiffuseColor;
      ..........................
}
```

Αυξομείωση της ταχύτητας πτώσης της σφαίρας

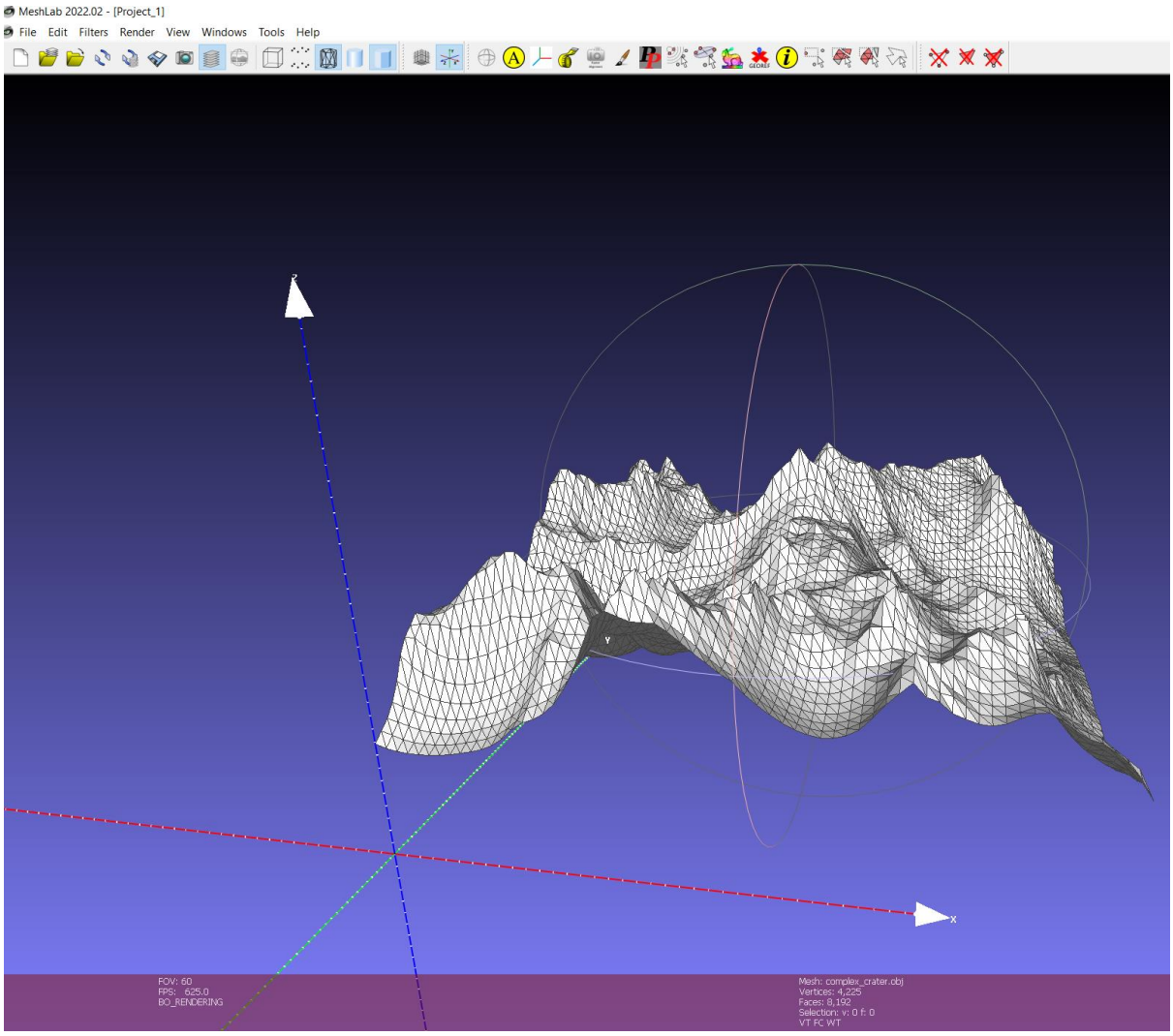
Με τα πλήκτρα <u> και <p> γίνεται αυξομείωση της ταχύτητας της σφαίρας:

```
if (should_launch_fireball == 1) {
            .........
             .........
            // Draw the fireball
            glDrawArrays(GL_TRIANGLES, 0, vertices[1].size());
            checkForSpeedPresses(); // Adjust the fireball's speed at will
            //printf("Fireball speed: %f\n", zOffset); // Debugging
            //printf("Fireball Z: %f\n", fireballPos.z); // Debugging
            fireballPos.z -= fireballPos.z * zOffset; // The fireball always
comes closer to the ground plane
        }
```

```
float zOffset_initial = 0.001f; // initial speed offset for fireball descent
0.0001f
float zOffset = zOffset_initial; // speed offset for fireball descent
void checkForSpeedPresses() {
    // Make the fireball descend faster or slower

    if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS) {
        zOffset = zOffset * 1.025f;
        //if (zOffset > 0.1f) { zOffset = 0.0001f; } // Reset offset if too
fast
    }
    else if (glfwGetKey(window, GLFW_KEY_P) == GLFW_PRESS) {
        zOffset = zOffset / 1.02f;
        if (zOffset < 0.00001f) { zOffset = 0.0001f; } // Reset offset if too
slow
    }
}
```

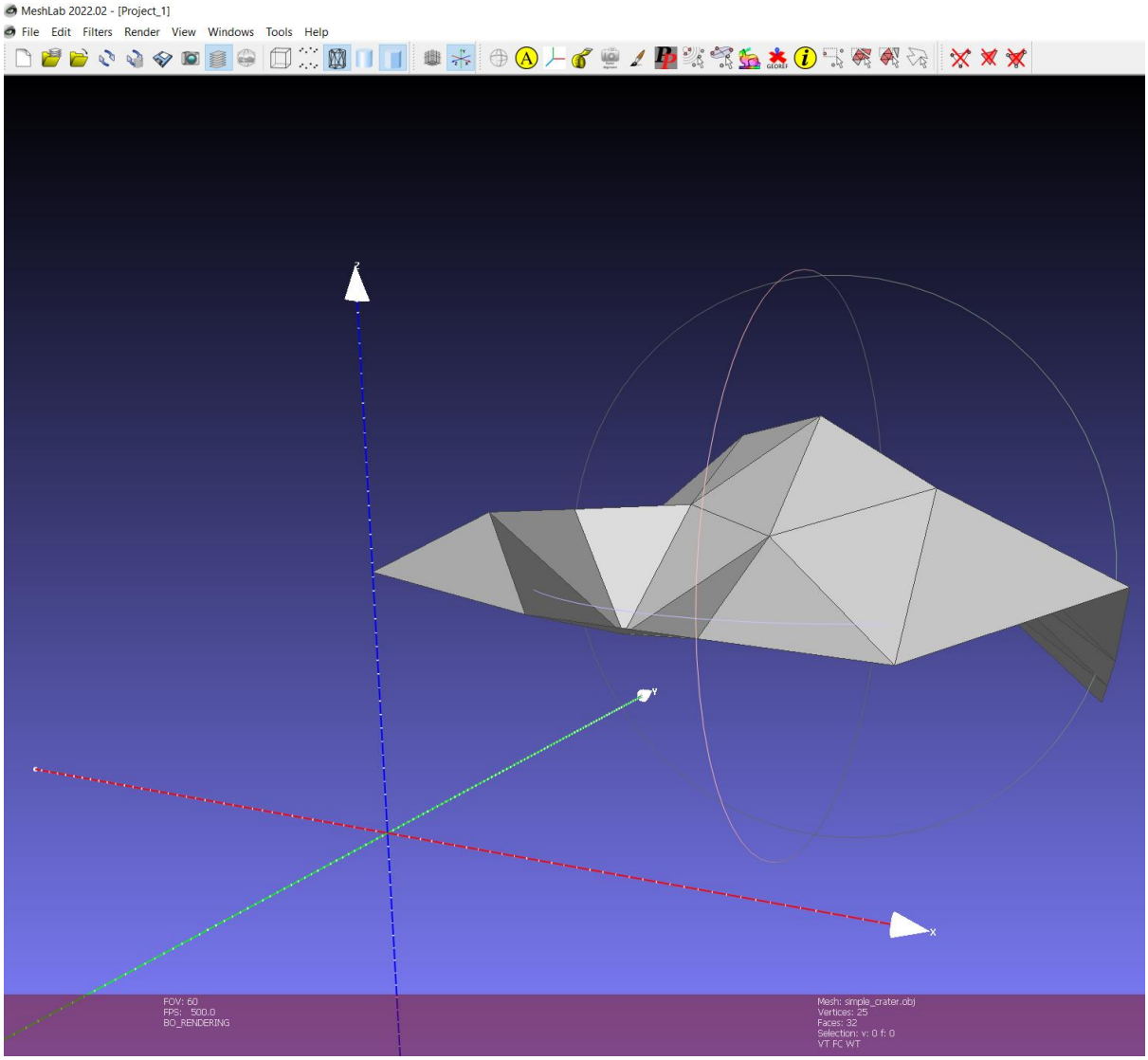
Αλλαγή της γεωμετρίας του εδάφους μετά την πρόσκρουση με fireball

Για την υλοποίηση αυτής της λειτουργίας δημιουργήσαμε στο meshlab object κρατήρα που προσομοιώνει έδαφος χτυπημένο απο μετεωρίτη.



Object εδάφους αποτελούμενο από περίπου 8000 τρίγωνα. (complex_crater.obj)

Επειδή η χρήση ενός τόσου πολύπλοκου σχήματος έκανε την εφαρμογή μας να αργεί εξαιρετικά στο startup (μιας που φορτώνονται 800 τέτοια objects) καταλήξαμε να δημιουργούμε ένα πιο απλό object κρατήρα με λιγότερα τρίγωνα:



Object εδάφους αποτελούμενο από περίπου 30 τρίγωνα. (simple_crater.obj)

Επιθυμούμε να αλλάζει η γεωμετρία του εδάφους μετά απο πρόσκρουση με fireball. Εφόσον πια έχουμε το object του κρατήρα μας, το μόνο που έχουμε να κάνουμε είναι να πούμε στον κώδικά μας πότε και πού να το σχεδιάσει.

Αυτό που κάνουμε είναι απλά να πούμε στον κώδικά μας να σχεδιάσει το object κρατήρα πάνω από το object εδάφους, δίνοντας έτσι την αίσθηση πως τα fireballs αλλάζουν τη μορφολογία του εδάφους.

```
// Load the crater objs
    for (int i = 2; i < number_of_objects; i++) {
        resOBJS[i] = loadOBJ("objs/simple_crater.obj", vertices[i], uvs[i],
normals[i]);
    }
```

```
// Draw craters dynamically only if they should be drawn, i.e the flag
ariable is raised
        for (int i = 2; i < number_of_objects; i++) {
            int craterID = i - 2;
            if (should_draw_craters_array[craterID] == 1) {

                // Load the crater
                .................
                glm::mat4 craterModelMatrix = glm::mat4(1.0f);
                // Move the crater to the correct position
                glm::mat4 newCraterModelMatrix =
                    translate(glm::mat4(1.0f),
glm::vec3(craters_coords_array[craterID].x, craters_coords_array[craterID].y,
craters_coords_array[craterID].z)) * craterModelMatrix;
                // Calculate MVP
                ...................
                // Draw the crater
                glDrawArrays(GL_TRIANGLES, 0, vertices[i].size());
            }
        }
```

Δημιουργία μεγαλύτερης ζημιάς στο έδαφος αν πέσει fireball σε σημείο που ήδη υπάρχει κρατήρας.

Τροποποιήσαμε το αρχείο κρατήρα μας (“simple_crater.obj”) μέσω scale στο meshlab και το μεγενθύναμε σε διπλάσιες διαστάσεις για να προσομοιώνει ένα μεγαλύτερο κρατήρα.

Με την έναρξη της εφαρμογής μας λέμε στον κώδικα να φορτώσει τα object μεγαλύτερου κρατήρα, και τα σχεδιάζουμε on demand όταν αυτά χρειαστούν:

```
// Load the big crater objs
    bool resOBJS_bigC[number_of_big_craters];
    for (int i = 0; i < number_of_big_craters; i++) {
        resOBJS[i] = loadOBJ("objs/simple_crater_2nd_impact.obj",
vertices_bigC[i], uvs_bigC[i], normals_bigC[i]);
    }

// Draw bigger craters dynamically only if they should be drawn, i.e the flag
variable is raised
        for (int i = 0; i < number_of_big_craters; i++) {
            if (should_draw_2ndcraters_array[i] == 1) {
                ……….
                // Move the crater to the correct position
                glm::mat4 newCraterModelMatrix =
                    translate(glm::mat4(1.0f),
glm::vec3(second_craters_coords_array[i].x, second_craters_coords_array[i].y,
second_craters_coords_array[i].z)) * craterModelMatrix;
                // Calculate MVP
                MVP = ProjectionMatrix * ViewMatrix * newCraterModelMatrix;
                glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
                glUniformMatrix4fv(ModelMatrixID, 1, GL_FALSE,
&newCraterModelMatrix[0][0]);
                glUniformMatrix4fv(ViewMatrixID, 1, GL_FALSE,
&ViewMatrix[0][0]);
                // Draw the crater
                glDrawArrays(GL_TRIANGLES, 0, vertices_bigC[i].size());
```

Η λογική που ακολουθούμε για τη σχεδίαση του μεγαλύτερου κρατήρα είναι η εξής:

Κάθε φορά που δημιουργούμε ένα fireball υπολογίζουμε τις συντεταγμένες του σημείου που θα προσκρούσει. Εάν αυτές οι συντεταγμένες είναι οι ίδιες με κάποια προηγούμενη πρόσκρουση, ή βρίσκονται εντός μιας ακτίνας από κάποια προηγούμενη πρόσκρουση, τότε φορτώνουμε τον μεγαλύτερο κρατήρα.

```
bool checkForCollision(glm::vec3 fireballPos, glm::vec3 ground_plane, float
fireball_radius) {
    // Check if the fireball collides with the ground plane
    ..................................
        float proximity = 10.0f;
        for (int i = 0; i < craterCounter; i++) {
            if ((crater_impact_coords.x - craters_coords_array[i].x == 0.0f)
&& (crater_impact_coords.y - craters_coords_array[i].y == 0.0f)) { // Fireball
impact point is the same as before
            //if (((crater_impact_coords.x - craters_coords_array[i].x >=
0.0f) && (crater_impact_coords.x - craters_coords_array[i].x < proximity)) &&
((crater_impact_coords.y - craters_coords_array[i].y >= 0.0f) &&
(crater_impact_coords.y - craters_coords_array[i].y < proximity))) {

                should_draw_2ndcraters_array[secondCraterCounter] = 1;
                second_craters_coords_array[secondCraterCounter] =
crater_impact_coords;
                secondCraterCounter += 1;
                return true;
            }
        }
        should_draw_craters_array[craterCounter] = 1;
        craters_coords_array[craterCounter] = crater_impact_coords;
        craterCounter += 1;

        return true;
    }
    return false;
}
```

Στον κώδικά μας είναι ενεργοποιημένη η επιλογή φόρτωσης μεγαλύτερου κρατήρα εάν οι συντεταγμένες της πρόσκρουσης είναι ακριβώς οι ίδες με κάποια προηγούμενη. Αυτο δουλέυει as intended και μπορεί να εξακριβωθεί αν στη γραμμή #291 του κώδικά μας βάλουμε το fireball να προσκρούει στις ίδιες συντεταγμένες κάθε φορά που πατάμε το <b>.

Αυτό που δε δουλεύει as intended είναι η φόρτωση μεγαλύτερου κρατήρα εάν οι συντεταγμένες της πρόσκρουσης είναι εντός μιας ακτίνας απο τις συντεταγμένες κάποιας προηγούμενης πρόσκρουσης, και για αυτό βρίσκεται σε σχόλια.

Εκεί υπάρχει ένα μικρό bug που δεν καταφέραμε να αντιμετωπίσουμε. Η φόρτωση του μεγαλύτερου κρατήρα λειτουργεί, αλλά όχι πάντα. Είμαστε σίγουροι πως αυτό έχει να κάνει με την συνθήκη την οποία έχουμε βάλει σε σχόλια.

Βιβλιογραφία

Για την εκπόνηση της παρούσας άσκησης χρησιμοποιήθηκαν οι εξής online αναφορές:

- https://robotics.stackexchange.com/questions/10702/rotation-matrix-sign-convention-confusion
- https://www.glfw.org/docs/latest/group__keys.html
- https://stackoverflow.com/questions/12040539/utf-8-compatibility-in-c
- https://www.color-hex.com/color/004225
- https://www.color-hex.com/color/ff2800
- https://web.cse.ohio-state.edu/~crawfis.3/cse581/Homework/TriangleRGB-Cpp.htm
- https://learnopengl.com/Getting-started/Hello-Triangle
- https://stackoverflow.com/questions/52195149/timestamp-of-time0-at-multiple-places-in-a-c-program
- https://www.digitalocean.com/community/tutorials/compare-strings-in-c-plus-plus
- https://stackoverflow.com/questions/58494179/how-to-create-a-grid-in-opengl-and-drawing-it-with-lines
- https://softwareengineering.stackexchange.com/questions/345052/opengl-generating-a-plane-with-only-one-triangle-strip

- https://stackoverflow.com/questions/5915753/generate-a-plane-with-triangle-strips
- https://www.youtube.com/watch?v=45MIykWJ-C4
- https://stackoverflow.com/questions/52008680/including-winmm-lib-does-not-work
- https://github.com/opengl-tutorials/ogl
- https://github.com/opengl-tutorials/ogl/tree/master/tutorial08_basic_shading
- https://stackoverflow.com/questions/21034935/playsound-in-c
- https://learnopengl.com/Advanced-OpenGL/Instancing