Проект по Компјутерска Графика, летен семестар 2022/23

КВИДИЧ ТЕРЕН (QUIDDITCH FIELD) ВО OPENGL

Ментори:

Лошковска Сузана

Јоксимовски Бобан

Изработиле:

Теодора Ковачка 211172

Филип Пенчиќ 211260

Содржина

Краток Вовед 3
Текот на креирање на проектот 4
Цртање на теренот со помош на Blender 5
Готовиот 3Д модел за теренот 6
Објаснување за поважните функции и класи 7
Читање на 3Д објектите: класата Object 7
Вовед во класата Object 7
Функцијата Object::load_model 8
Функцијата Object::set_buffer_data 8
Интерактивно движење низ сцената: класа Camera 9
Функцијата Camera::ProcessKeyboard 9
Функцијата ToggleFlightMode 10
Финалниот продукт 11
Референци 12

Краток Вовед

Идејата за проектот потекнува од франчизата Хари Потер (анг. Harry Potter), поточно од измисленият спорт наречен Квидич (анг. Quidditch). Правилата на спортот се едноставни: играчот треба да даде гол во еден од трите обрачи на спротивниот тим, додека лета на метла. Кратко видео од спортот може да погледнете на следниот линк.

Целата околина за Квидич спортот е соствена од: терен каде што играчите летаат на метли, високи обрачи (голови), а околу целиот терен се поставени високи кули каде што седи публиката (Слика 1).

Во овој проект нашата цел беше да направиме програма што ќе овозможи движење низ една едноставна верзија од теренот, каде гледачот ќе може да го истражува теренот „пешки“ или „на метла“.

Слика 1: Изглед на теренот за Квидич спортот

Текот на креирање на проектот

Првичниот проект го започнавме со клонирање на проектот кој го користевме на аудиториските вежби, специфично на гранката „camera-4“. Во рамките на проектот во папката „vendor“ ја додадовме assimp библиотеката, а во датотеката CMakeLists ги додадовме потребните зависности за истата да биде точно интегрирана во проектот.

Откако успешно ја поврзавме библиотеката, започнавме со имплементирање на читањето на објектите. За таа цел креиравме нови помошни датотеки – Object.hpp и Object.cpp, во чии рамки се имплементира читањето на координатите за позиција на објектите како и нивните бои и текстури и нивно поврзување во OpenGL проектот.

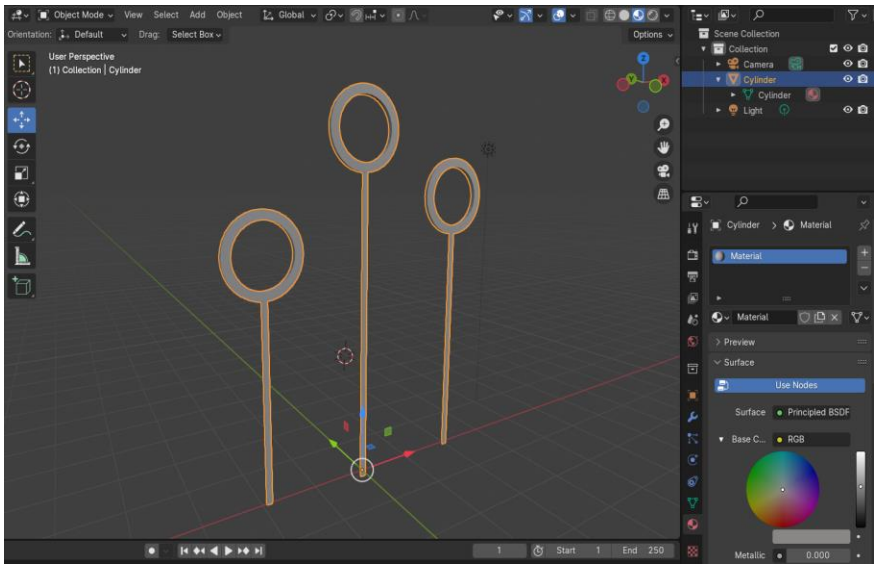
По успешно тестирање на вчитувањето со едноставни 3Д објекти, започнавме со цртање на самиот терен во програмата Blender. Користените слики за текстдодадовме и во рамките на самиот проект во папката res/textures и успешно ги поврзавме истите во датотеката за материјали („field.mtl“).

Откако целиот терен успешно се рендерираше во OpenGL проектот, последната промена што ја направивме беше во рамките на датотеката Camera.cpp, каде што ги променивме функциите за менување на камерата и погледот, за да го добиеме посакуваниот резултат на движење низ теренот „пешки“ и на „метла“. Логиката на движењето е следна:

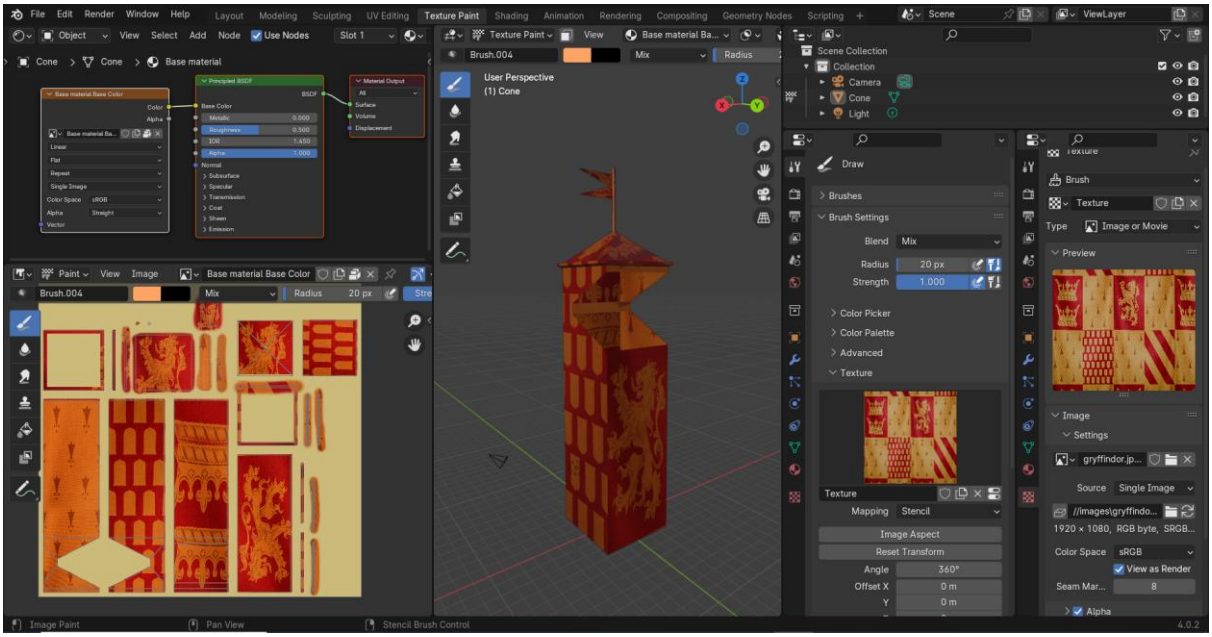
- → Играчот почнува со движење на земја по теренот. Движењето се одвива во прво лице. Може да се движи со копчињата „A“, „W“, „S“ и „D“ од тастатурата, а насоката на движење се менува со движење на глувчето.
- → Доколку го притисне копчето „M“ од тастатурата, на играчот му е овозможено „летање“ низ теренот на замислена метла.
- → Движењето на метлата се одвива на истиот начин како и движење на земја, само нема ограничување дека играчот мора да се движи по x-оската.

Цртање на теренот со помош на Blender

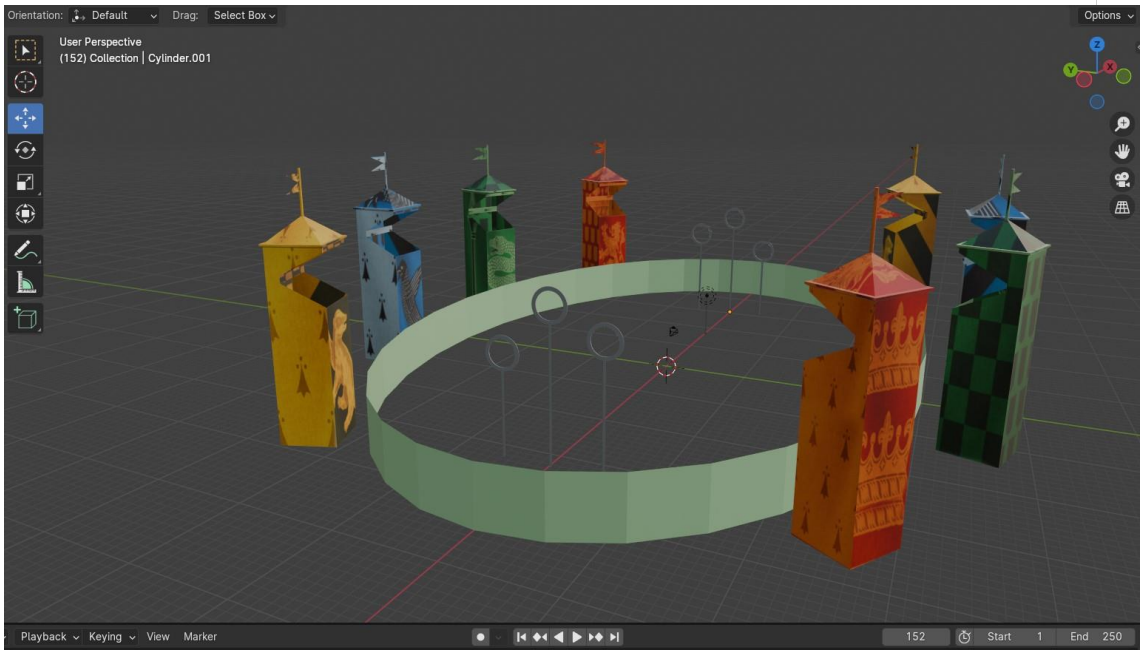
Со цел да можеме да креираме покомплексен терен, одлучивме дека истиот првично ќе го нацртаме со помош на програмата Блендер (анг. Blender). Започнавме со исцртување на кулите и головите, додавање на текстури на истите и на крај поврзување на сите објекти во една единствена 3Д сцена (Слики 2, 3, 4 и 5). Целиот готов терен го експортиравме во една „.obj“ датотека со име „field.obj“, каде што беа зачувани координатите за позиција како и текстури на теренот. Оваа датотека подоцна ја исчитавме во OpenGl проектот, со помош на класата Object.



Слика 2: Основно цртање на 3Д формите во Blender

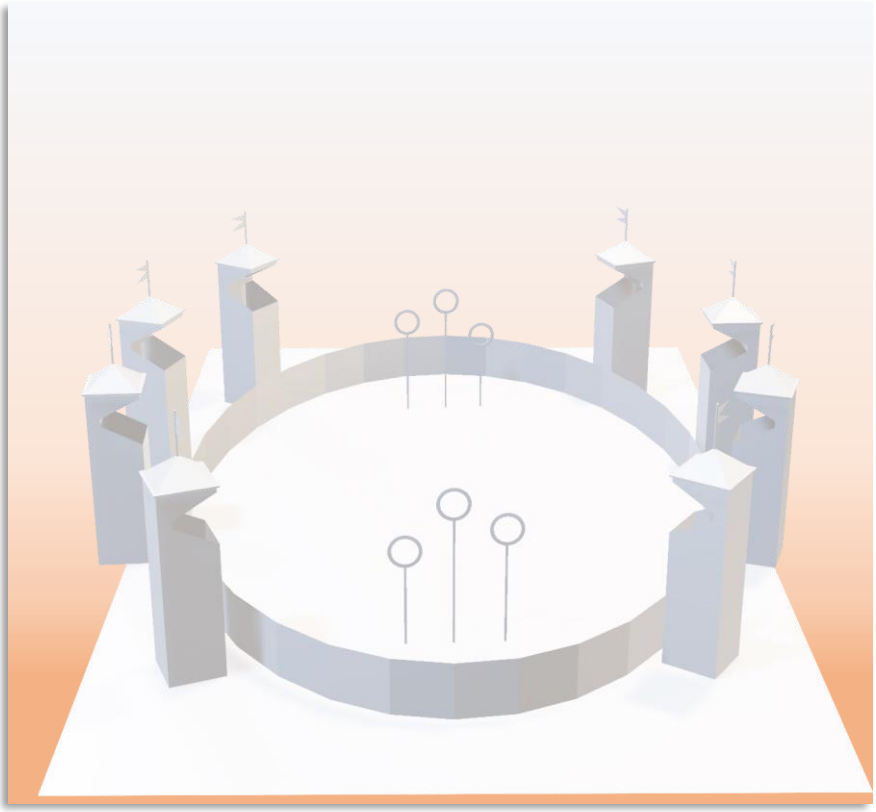


Слика 3: Додавање на текстури за секој 3Д објект во Blender



Слика 4: Склопување на целиот терен во Blender

Готовиот 3Д модел за теренот



Слика 5 field.obj датотеката прочитана како 3Д објект

Објаснување за поважните функции и класи

Читање на 3Д објектите: класата Object

За вчитување на целата „field.obj“ датотека во вектори со координати за позиција и текстури, во проектот одлучивме дека ќе ја користиме библиотеката *Open-Asset-Importer-Library - assimp.* За таа цел, креиравме нова класа – *Object* која ќе се справува со читањето од „field.obj“ датотеката и поврзувањето на 3Д објектите во проектот. Декларирањето и имплементацијата на функциите на оваа класа се наоѓаат во датотеките „Object.hpp“ и „Object.cpp“. Во продолжение ќе ја разгледаме оваа класа.

Вовед во класата Object

Во рамки на *Object* класата, чуваме инстанца од *aiScene* структурата (подоцна ќе ја викаме само *сцена*). *aiScene* е структура во *assimp* библиотеката која нуди олеснување на читањето на датотеки каде се чуваат информации за 3Д објекти, па затоа и ја користиме. Во рамки на конструкторот на *Object,* со помош на класата *Importer* (повторно класа од библиотеката *assimp*), ја читаме нашата датотека „field.obj“ и ја сместуваме во *aiScene* сцената.

```
Object::Object(const char *model_path) {
    scene = Object::importer.ReadFile(model_path,
                                      aiProcess_JoinIdenticalVertices |
aiProcess_Triangulate | aiProcess_FlipUVs);
    load_model();
}
```

Во рамки на *Object* класата чуваме и други помошни структури: *Mesh* и *Texture* за поорганизирано чување на податоците за објектите. Дополнително, чуваме и вектори од сите објекти и сите текстури на сцената, во класата тие се именувани *mesh_list u texture_list*.

```
struct Mesh {
    unsigned int VAO, VBO1, VBO2, VBO3, EBO;

    std::vector<glm::vec3> vert_positions;
    std::vector<glm::vec3> vert_normals;
    std::vector<glm::vec2> tex_coords;
    std::vector<unsigned int> vert_indices;
    unsigned int tex_handle;
};

struct Texture {
    unsigned int textureID;
    std::string image_name;
};
```

```
std::vector<Mesh> mesh_list;
std::vector<Texture> texture_list;
```

Функцијата Object::load_model

Функцијата *load_model* која ја повикуваме во конструкторот, итеративно изминува низ податоците од *aiScene* сцената и го сместува секој посебен 3Д објект во посебна структура *Mesh* а неговите текстури во структура *Texture*, а сите објекти со нивните текстури ги додава во векторите *mesh_list* и *tex_coords* (вектор за координати на текстури на секој објект посебно, деклариран во рамки на *Mesh* структурата). Во следниот дел од кодот се додаваат позициите и текстурите[1] на објектите во рамки на посебни *Mesh* структури, како и во векторите со сите објекти.

```
glm::vec3 position{};
position.x = mesh->mVertices[i2].x;
position.y = mesh->mVertices[i2].y;
position.z = mesh->mVertices[i2].z;
mesh_list[i].vert_positions.push_back(position);

if (mesh->HasNormals())
{
    glm::vec3 normal{};
    normal.x = mesh->mNormals[i2].x;
    normal.y = mesh->mNormals[i2].y;
    normal.z = mesh->mNormals[i2].z;
    mesh_list[i].vert_normals.push_back(normal);
}
else
    mesh_list[i].vert_normals.push_back(glm::vec3(0.0f, 0.0f, 0.0f));

if (mesh->HasTextureCoords(0)) // Only slot [0] is in question.
{
    glm::vec2 tex_coords{};
    tex_coords.x = mesh->mTextureCoords[0][i2].x;
    tex_coords.y = mesh->mTextureCoords[0][i2].y;
    mesh_list[i].tex_coords.push_back(tex_coords);
}
else
    mesh_list[i].tex_coords.push_back(glm::vec2(0.0f, 0.0f));
```

Функцијата Object::set_buffer_data

Во рамки на оваа функција ги поврзуваме сите позиции и координати со OpenGl проектот, користејќи ги функциите *glBindBuffer, glBufferData, glEnableVertexAttribArray и glVertexAttribPointer*. Пример за вчитување на позициите на објектите е во следниот код. Слично правиме и за останатите податоци.

```
glBindBuffer(GL_ARRAY_BUFFER, mesh_list[index].VBO1);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) *
mesh_list[index].vert_positions.size(), &mesh_list[index].vert_positions[0],
GL_STATIC_DRAW);

glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
```

[1] Забелешка: се додаваат и податоци за нормалите на објектите, но бидејќи ние во овој проект не работиме со светлина, во овој случај се игнорираат. Секако, доколку се импортира друга датотека што има нормали, истите би биле успешно прочитани.

Интерактивно движење низ сцената: класа Camera

Движењето низ сцената беше веќе овозможено во гранката „camera-4“ која ја клониравме од проектот од аудиториските вежби. Првата промена што ја направивме беше во готовата класа *Camera* беше додавање на информација дали играчот се движи пешки (*HUMAN_WALKING*) или лета (*FLYING*) за полесно справување со двете ситуации.

```
enum Camera_Mode {
    HUMAN_WALKING, FLYING
};

class Camera {
public:
    // Camera Attributes
    glm::vec3 Position;
    glm::vec3 Front;
    glm::vec3 Up;
    glm::vec3 Right;
    glm::vec3 WorldUp;
    // Euler Angles
    float Yaw;
    float Pitch;
    // Camera options
    float MovementSpeed;
    float MouseSensitivity;
    float Zoom;
    Camera_Mode mode;
```

Функцијата Camera::ProcessKeyboard

Во рамки на оваа функција, во зависност од вредноста на атрибутот *mode,* движењето преку тастатура или е ограничено на движење само по x-оската, или е целосно слободно за движење во сите правци (вака веќе и беше имплементирано). Во продолжение е кодот за ограничување на движењето во случајот *HUMAN_WALKING.*

```
    case HUMAN_WALKING:
        switch (direction) {
            case FORWARD:
                Position += glm::vec3(glm::cos(glm::radians(Yaw)), 0,
glm::sin(glm::radians(Yaw))) *
                            velocity; //Y is not affected, Y is looking up
                break;
            case BACKWARD:
                Position -= glm::vec3(glm::cos(glm::radians(Yaw)), 0,
glm::sin(glm::radians(Yaw))) *
                            velocity; //Y is not affected, Y is looking up
                break;
            case LEFT:
                Position -= Right * velocity;
                break;
            case RIGHT:
                Position += Right * velocity;
                break;
        }
        break;
```

Функцијата ToggleFlightMode

Во оваа функција е овозможено менување на начините на движење наизменично при секое повикување на функцијата, т.е. доколку начинот бил *HUMAN_WALKING* по повикот на функцијата ќе биде *FLYING* и обратно.

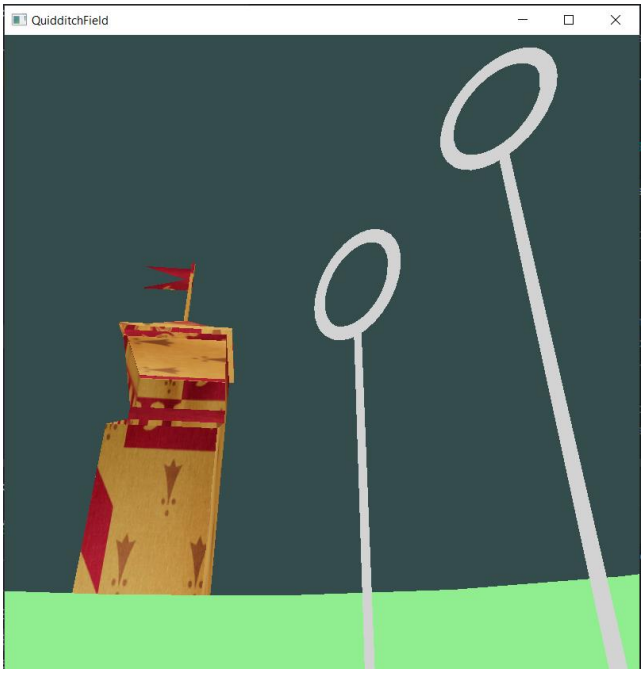
```
void Camera::ToggleFlightMode() {
    mode = (mode == HUMAN_WALKING) ? FLYING : HUMAN_WALKING;
}
```

Дополнително, оваа функција ја повикуваме на секое притискање на копчето „M“ од тастатурата, т.е. во функцијата *processInput()* во *main* функцијата.

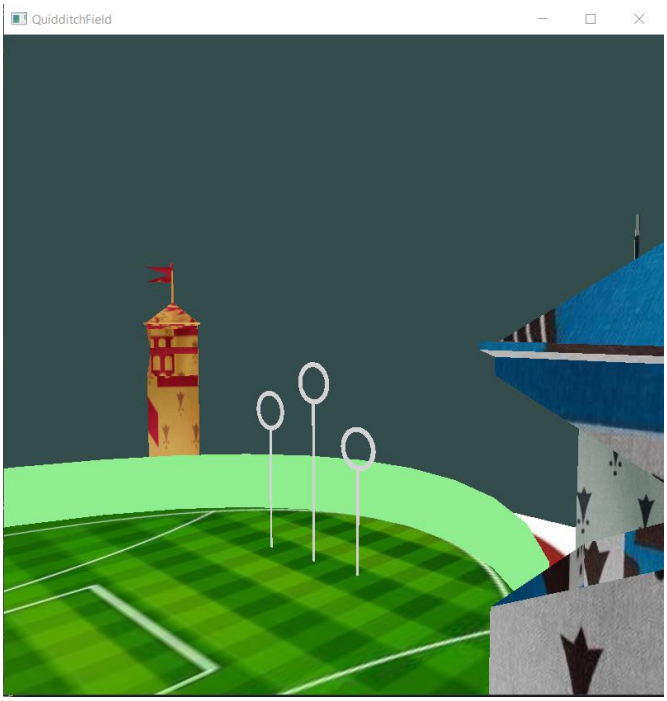
```
void processInput(GLFWwindow *window) {
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        camera.ProcessKeyboard(camera.mode, FORWARD, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        camera.ProcessKeyboard(camera.mode,BACKWARD, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyboard(camera.mode,LEFT, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyboard(camera.mode,RIGHT, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_M) == GLFW_PRESS)
        camera.ToggleFlightMode();
}
```

Финалниот продукт

Финалниот продукт што се добива откако ќе се изврши програмата е целиот Квидич терен, рендериран со помош на OpenGl, каде што играчот може да се движи низ истиот и да го истражува пешки (Слика 6) или со „летање“ (Слика 7).



Слика 6 Поглед нагоре, од „пешачење“



Слика 7 Поглед надолу, од „летање“

Референци

1. За почетниот проект
   https://github.com/joksim/OpenGLPrj

2. За поврзување на assimp библиотеката и менување на CMakeLists
   https://github.com/Polytonic/Glitter

3. За креирање на Object.hpp и Object.cpp класите
   https://www.programmingcreatively.com/opengl-tutorial-5-qs.php
   https://www.youtube.com/watch?v=GovbphOagoQ&t=604s&ab_channel=CodeImposter

4. За цртање на теренот во Blender
   https://www.youtube.com/watch?v=YoikJKRCq34&t=360s

5. За додавање текстури во Blender
   https://www.youtube.com/watch?v=lO8qDNZf5o0&t=804s

6. За менувањето на погледот и камерата
   https://stackoverflow.com/questions/71707566/opengl-first-person-realistic-keyboard-movement