МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Московский государственный технологический университет «СТАНКИН»

(ФГБОУ ВО «МГТУ «СТАНКИН»)

Институт информационных систем и технологий

Кафедра информационных систем

Программирование специализированных вычислительных устройств
Отчет по лабораторной работе

«3D моделлирование посредством OpenGL для Веба» Часть №1

Студент группы ИДБ-19-07: Касьян А.И.

Проверил доцент кафедры ИС: к.т.н. Волкова О.Р.

October 30, 2020

Оглавление

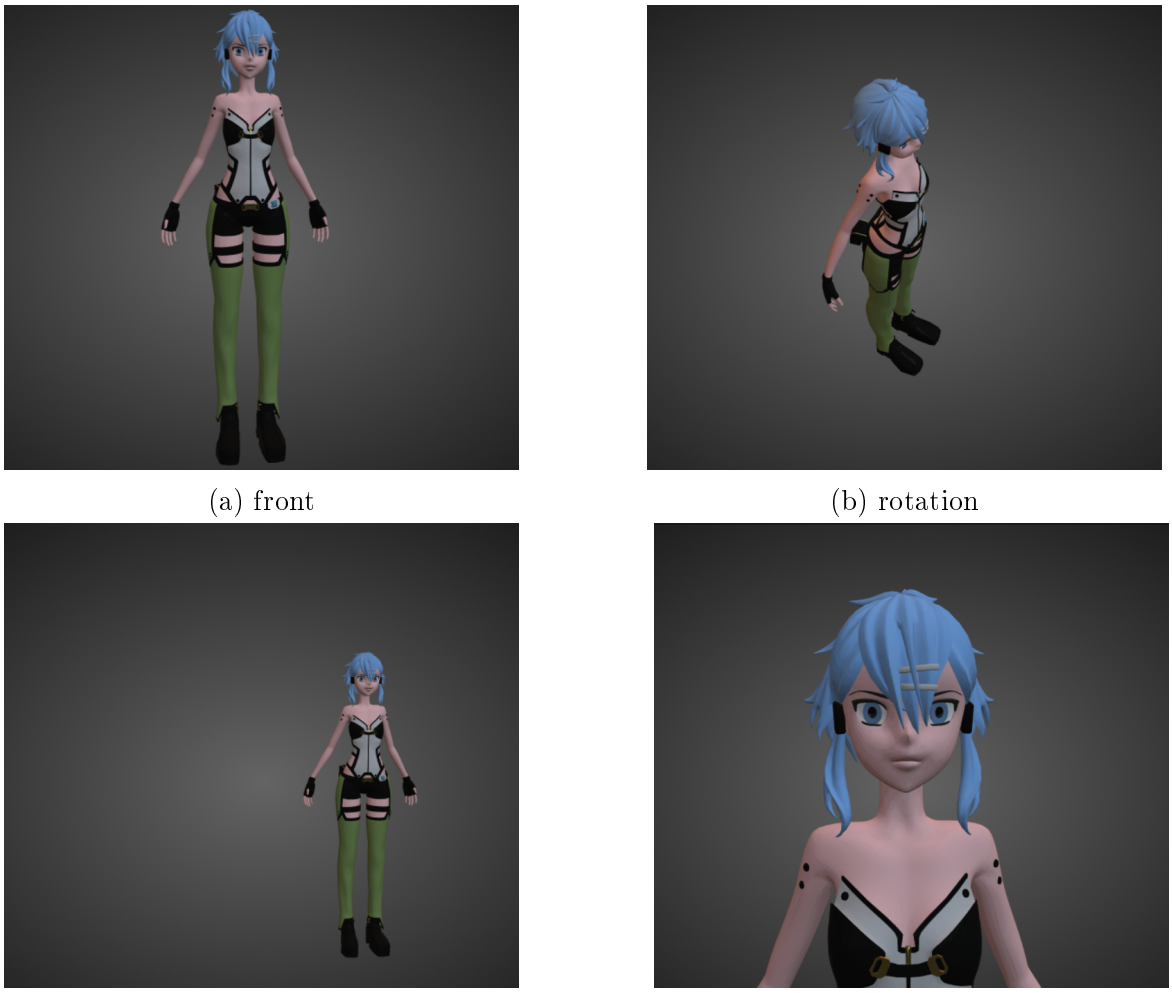
1 Вводное слово

В данной секции я бы хотел обозначить некоторые моменты, которые могут сбивать.

- Данный, как и все остальные документы я оформляю с помощью LaTex, в связи с этим ни doc ни docx файлы предоставить не могу (если таковые потребуются). Все source-файлы находятся на GitHub'e в соответствующей папке *doc*
- Из-за не маленького количества кода я не могу полностью сопроводить листингом. Весь код есть на моём GitHub'e. Однако, основные моменты я вставлю и прокоментирую.
- Всвязи с плотным графиком на работе, я не могу уделять время на лаб. работы. По этой же причине я взял задание у преподавателя ООП (Разумовского А.И.) и оно по своей тематике очень схоже с лаб. работами.
- Я сразу хочу подметить, что проект не создавался как кроссплатформенное решение, т.к. я считаю, что это глупо использовать для кроссплатформы OpenGL (так считаю не только я: ссылка на видео), т.к. у каждой платформы есть более подходящие спецификации, с более комплексным и гибким API. Поэтому проект работает на платформе Windows (а точнее, компилятор MSVC) и Windows/*nix с компилятором emcc (Emscripten compiler). За поведение при использовании других компиляторов я ответственности не несу.
- Если вы заглянете в код, коментарии там на английском. Т.к. я в большинстве случаев использую английский и мне на нём проще излагать свои мысли. (Если взгляните на мой GH, он весь на английском).

2 Постановка задания

Требуется реализовать отрисовку 3D-моделей, посредством OpenGL и портировать программу под web, с помощью Emscripten. Так же должна присутствовать возможность осмотра модели, я это сделал посредством камеры, сама модель статична. В идеале результат должен выглядеть как-то так:



(a) front (b) rotation

(c) movement (d) zoom

Figure 1: Transformations

3 Набор инструментов

1. Docker - для развёртки на любой машине
2. Appveyor - сборка и тестирование для MSVC компиляторов
3. Travis - сборка и тестирование для emscripten.
4. CMake - для генерации файлов сборки.
5. googletests - для тестов.
6. Python3 - для запуска вебсервера
7. YAML-CPP 0.6.3 - для парсинга YAML файлов
8. GLFW 3.3.2 - API для OpenGL.

9. GLEW 2.1.0 - для низкоуровневого взаимодействия с OpenGL.
10. SOIL 1.20 - библиотека для загрузки изображений.
11. GLM 0.9.9.8 - библиотека для удобной работы с линейной алгеброй и OpenGL.
12. EMSDK 2.0.7 - emscripten compiler.
13. Assimp 5.0.1 - для парсинга obj и mlt файлов.

4 Результат выполнения задания

В данной части я распишу принцип работы шейдоров и класс, который я создал (будем честными, нагло скопипастил и разобрал). А в следующей я разберу трансформации и текстуры.

4.1 CMake

Начну я с CMake'а, т.к. это одна из основополагающих проекта. Так как я пишу под разные компиляторы - я использую генерацию сборочных файлов. Поэтому я думаю стоит уделить пару слов CMakeLists.txt.

```
# Set the variables responsible for specific files.
set(${PROJECT_NAME}_SOURCES
                "application/sources/shader.cpp"
                "application/sources/camera.cpp"
                "application/sources/texture.cpp"
                "application/sources/application.cpp"
            )
set(${PROJECT_NAME}_INCLUDES
                "application/includes/shader.h"
                "application/includes/camera.h"
                "application/includes/texture.h"
                "application/includes/vertex.h"
                "application/includes/application.h"
            )
set(${PROJECT_NAME}_MAIN "application/main.cxx")

file(GLOB_RECURSE CFG_FILES ${CMAKE_SOURCE_DIR}/config/*.yaml)
file(GLOB_RECURSE ASSETS ${CMAKE_SOURCE_DIR}/assets/*)
```

Технически, в данном куске мы собираем все файлы, требуемые для проекта. Далее я подключаю созданные мною модули, которые лежат в *cmake/Modules*

```
include(setflags)
include(safeguard)
include(graphics)
include(configloader)
```

Я думаю стоит пробежаться только по двум модулям - graphics и configloader.

1. Дабы сделать интерфейс более удобным, я добавил конфигурацию проекта через yaml-файл. И написал небольшую библиотеку для чтения конфигурационного файла.

В configloader находится сборка данной библиотеки. По данному пути *application/-configloader/CMakeLists.txt* вы можете найти CMakeList, который собирает эту библиотеку из source-файлов.

2. graphics в свою очередь собирает и устанавливает флаги линковки для графических библиотек. В связи с тем, что у Emscripten есть свой GLFW, там стоят пару if-else, в зависимости от компиляторов.

Далее устанавливаются листы с библиотеками:

```
# Not in-build emscripten libraries,
set(DEP_EMSDK_LIBS
    soil2
)

# The libraries, required by application,
# however they are already built in emscripten
# But they are not built in the ordinar one.
if (NOT FOR_EMSDK)
    set(DEP_LIBS
        glfw
        libglew_shared
    )
endif()

# Other Libs, actually created by my-self.
set(LIBS
    ConfigLoader
)

set(INTERFACE_LIB
    glm
)
```

Библиотеки разделены на несколько групп, т.к. для Emscripten'а не требуется API для OpenGL, да и OpenGL как токовой (веб использует WebGL =/). А у glm интерфейссное подключения, а т.к. я преношу все библиотеки в одно место при генерации сборочных файлов - перенести glm не удасться.
Создаём исполняемый файл и линкуем все библиотеки к нему. Некоторые библиотеки линкуются, только если компилятор - это MSVC.

```
# adding executable
add_executable(${PROJECT_NAME}
                ${${PROJECT_NAME}_MAIN}
                ${${PROJECT_NAME}_SOURCES}
                ${${PROJECT_NAME}_INCLUDES}
                ${CFG_FILES}
                ${ASSETS}
            )
target_link_libraries(${PROJECT_NAME}
                        ${LIBS}
                        ${DEP_LIBS}
                        ${DEP_EMSDK_LIBS}
```

```
                        ${INTERFACE_LIB}
                    )
target_include_directories(
    ${PROJECT_NAME} PUBLIC
    $<BUILD_INTERFACE:${PROJECT_SOURCE_DIR}/application/includes>
    $<INSTALL_INTERFACE:${PROJECT_SOURCE_DIR}/application/includes>
)

if (NOT FOR_EMSDK)
    find_package(OpenGL REQUIRED)
    target_link_libraries(${PROJECT_NAME} OpenGL::GL)
endif()
```

4.2 Шейдеры

Как я и говорил в этой части я разберу только шейдеры. Т.к. я использую веб версию, которая не поддерживает GLSL пришлось шейдеры разделить на отдельные группы файлов (для десктопа и для веба). Но выглядят они очень схоже:

```
#version 330 core


in vec2 TexCoord;

out vec4 color;

uniform sampler2D ourTexture1;

void main()
{
    color = texture(ourTexture1, TexCoord);
}
```

(a) desktop fragment shader

```
#version 300 es
precision mediump float;

in vec2 TexCoord;

out vec4 color;

uniform sampler2D ourTexture1;

void main()
{
    color = texture(ourTexture1, TexCoord);
}
```

(b) web fragment shader

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 2) in vec2 texCoord;
out vec2 TexCoord;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    gl_Position = projection * view * model
         * vec4(position, 1.0f);

    TexCoord = vec2(texCoord.x,
                    1.0 - texCoord.y);
}
```

(c) vertex shader

```
#version 300 es
layout (location = 0) in vec3 position;
layout (location = 2) in vec2 texCoord;
out vec2 TexCoord;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    gl_Position = projection * view * model
                    * vec4(position, 1.0f);

    TexCoord = vec2(texCoord.x,
                1.0 - texCoord.y);
}
```

(d) web vertex shader

Figure 2: Shaders

Я разделил шейдеры на файлы с соответсвующими расширениями:

- **vs** - вертекс шейдеры
- **frag** - фрагмент шейдеры
- **wvs** - вертекс шейдеры для веба
- **wfrag** - фрагмент шейдеры для веба

Код шейдера я разбирать не хочу, он максимально тревиален и скопирован из первого попавшегося туториала. В вертекс шейдере на вход принимается координата текстуры, на выходе ловим цвет этой координты, согласно глобально- установленной текстуре. Во фрагмент шейдере на входе передаётся позиция и кордината текстуры, так же присутствуют матрицы, отвечающие за трансформацию. Устанавливаем позицию и координату текстуры.

4.3 Обработка шейдеров

Для обработки шейдеров я создал класс Shader, его интерфейс выглядит так:

```
1  class Shader
2  {
3    private:
4      // Member vars
5      unsigned int program;
6      std::string loadShaderFile(const char *path);
7      unsigned int compileShader(unsigned int type, const char *path);
8      void linkProgram(unsigned int vertex_shader, unsigned int fragment_shader);
9    public:
10     // Constructor generates the shader on the fly
11     Shader(const char *vertex_path, const char *fragment_path);
12     // Uses the current shader
13     void Use();
14     void Unuse();
15     void set1i(int value, const char *name);
16     void setMat4fv(glm::mat4 value, const char *name, unsigned char transpose = GL_FALSE);
17     ~Shader();
18 };
```

Figure 3: Shaders Interface

Я не хочу приводить листинг всей имплиментации - с ней можно ознакомиться на GitHub'e. Я приведу только те, которые мне кажется стоят пояснения. Дабы не загряхнять код в основном файле - я вынес установку матриц (на данный момент для трансформации) в отдельные методы.

```
1  void Shader::set1i(int value, const char *name)
2  {
3      this->Use();
4      glUniform1i(glGetUniformLocation(this->program, name), value);
5      this->Unuse();
6  }
7  void Shader::setMat4fv(glm::mat4 value, const char *name, unsigned char transpose)
8  {
9      this->Use();
10     glUniformMatrix4fv(glGetUniformLocation(this->program, name), 1, transpose, glm::value_ptr(value));
11     this->Unuse();
12 }
```

Все пояснения хорошо оформлены мной в коментариях.

```cpp
std::string Shader::loadShaderFile(const char *path)
{
    std::string shader_code;   // The code of shaders as a string
    std::ifstream shader_file; // The very file
    // ensures ifstream objects able to throw exceptions
    shader_file.exceptions(std::ifstream::badbit);
    try
    {
        // Open file
        shader_file.open(path);
        std::stringstream shader_stream;
        // Read file's buffer contents into stream
        shader_stream << shader_file.rdbuf();
        // close file handler
        shader_file.close();
        // Convert stream into string
        shader_code = shader_stream.str();
    }
    catch (std::ifstream::failure &error)
    {
        std::cerr << "An error has occured in the " << __FILE__
                  << " in line " << __LINE__ << "." << std::endl
                  << "Details: " << error.what() << std::endl;
    }
    return (shader_code);
}

unsigned int Shader::compileShader(unsigned int type, const char *path)
{
    char log_info[512]; // log information if something wrong
    int success;        // the variable indicates if a shader's compilation went wrong

    // shader of a specific type (vertex/fragment/geometry)
    unsigned int shader = glCreateShader(type);
    // load source code from a file
    std::string str_shader_code = loadShaderFile(path);
    const char *shader_code = str_shader_code.c_str();
    // compile shaders itself
    glShaderSource(shader, 1, &shader_code, NULL);
    glCompileShader(shader);

    // check if a compilation done well
    glGetShaderiv(shader, GL_COMPILE_STATUS, &success);
    if (!success)
    {
        glGetShaderInfoLog(shader, 512, NULL, log_info);
        std::cerr << "An error has occured in the " << __FILE__
                  << " in line " << __LINE__ << "." << std::endl
                  << "Error occured in the: " << path << std::endl
                  << "Details: " << log_info << std::endl;
    }
    return shader;
}
```

Figure 4: Shaders Interface

4.4 Использование шейдеров

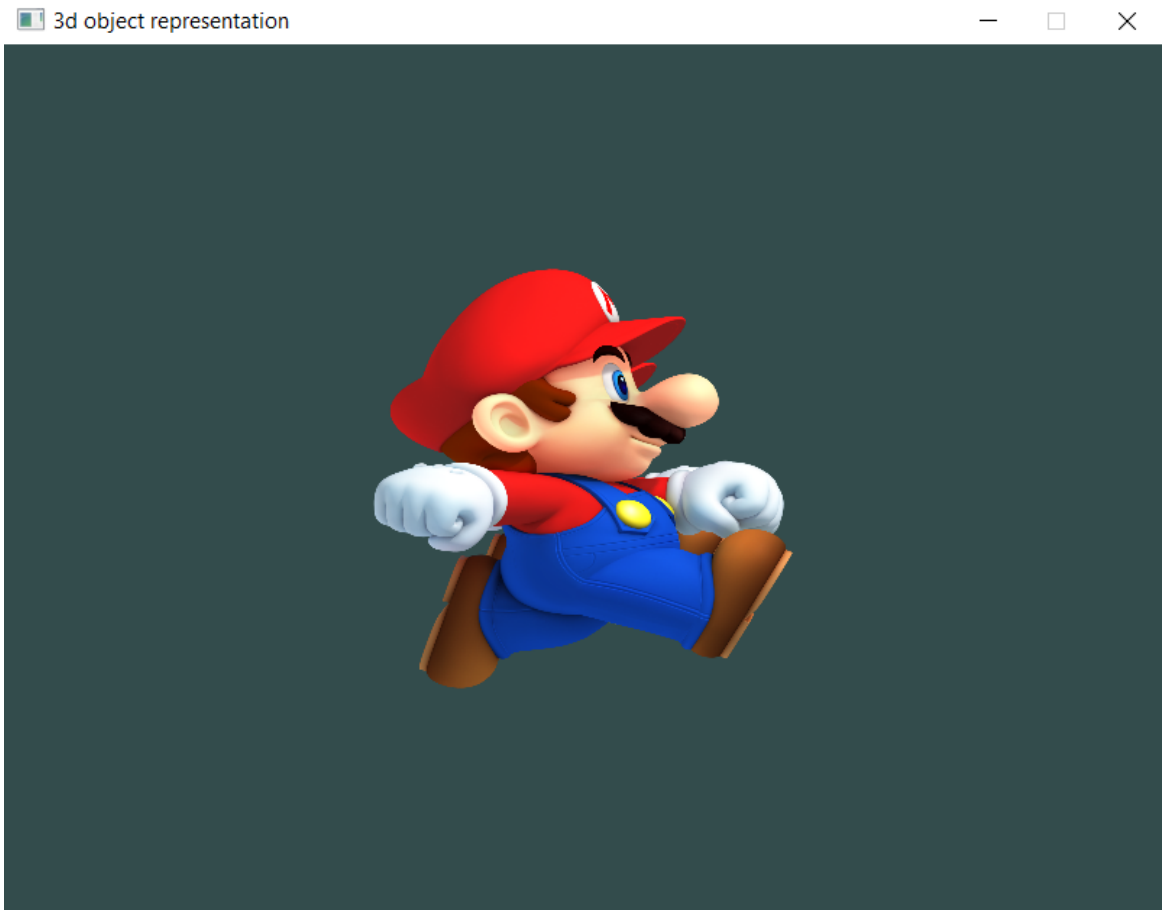
На данный момент я их использую в основном файле, потом это занесётся в отдельный класс. Я заменил код, который не относится к шейдерам на «...» дабы не громоздить здесь.

```cpp
int main()
{
    ...
#ifndef __EMSCRIPTEN__
    Shader ourShader("assets/shaders/core.vs", "assets/shaders/core.frag");
#else
    Shader ourShader("assets/shaders/core.wvs", "assets/shaders/core.wfrag");
#endif
    ...
#ifdef __EMSCRIPTEN__
    std::function<void()> mainLoop = [&]() {
#else
    while (!glfwWindowShouldClose(window))
    {
#endif
        // Set frame time
        ...
        // Check and call events
        ...
        // Clear the colorbuffer
        ...
        // Draw our first triangle
        ourShader.Use();

        // Bind Textures using texture units
        texture.bind(0);
        ourShader.set1i(0, "ourTexture1");

        glm::mat4 projection(1);
        projection = glm::perspective(camera.GetZoom(),
                     (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT,
                     0.1f, 1000.0f);

        // Create camera transformation
        glm::mat4 view(1);
        view = camera.GetViewMatrix();

        // Pass the matrices to the shader
        ourShader.setMat4fv(view, "view");
        ourShader.setMat4fv(projection, "projection");

        glBindVertexArray(VAO);

        for (GLuint i = 0; i < 10; i++)
        {
            // Calculate the model matrix for each object and pass it to shader before drawing
            ourShader.setMat4fv(model, "model");
            ourShader.Use();
            glDrawArrays(GL_TRIANGLES, 0, 36);
        }

        glBindVertexArray(0);

        // Swap the buffers
        glfwSwapBuffers(window);
#ifdef __EMSCRIPTEN__
    };
    emscripten_set_main_loop_arg(dispatch_main, &mainLoop, 0, 1);
#else
    }
#endif
...
}
```

Небольшой момент, который я пояснию - это компиляционные дерективы `ifdef-else-endif` Т.к. для emscripten'а требуется немного другой синтаксис - для этого я использую компиляционные дерективы `ifdef-else-endif`, которые в зависимоти от компилятора будут подгружать разные файлы с шейдерами.

5 Промежуточный результат

В данном документе я показываю, что шейдеры работают правильно. Т.е. то, что отображается и как - на данном этапе нас не волнует.



(a) Shader's proper result

6 Конец первой части

На этом я закончу небольшой разбор шейдеров и их обработку. Во второй части мы рассмотрим:

- Трансформацию и текстурирование.
- Так же я уделю немного места для emscripten'а.
- Посмотрим на результат, который у меня имеется на данный момент.
- Затронем что ещё необходимо сделать/улучшить.