New C++ Project in Visual Studio

1. open Visual studio 2017
2. File>New>Project>
3. Templates/Visual C++/Empty Project
4. Set Name and directory and create
5. in Solution Explorer window (View>Solution Explorer) check “Show All Files” icon.
6. Create Main.cpp: (in Solution Explorer) Right Mouse click > Add > New Item > C++ file
7. Try first with basic “hallo world” script (inside Main.cpp):

#include <iostream>

int main()

{

std::cout << "Hello World" <<std::endl;

std::cin.get();

}

1. Press Local Window Debugger, and see if it works.

**If we want to Change the VS Folder Structure**

**From**

***To***

1. Open Property Page: in Solution Explorer right click on project and click Properties (or Alt+Enter)
2. **“Configuration:”** dropdown set to “**All Configurations”**.
3. **“Platforms:”** dropdown set to “**All Platforms”**.
4. Configuration Properties/General/Output Directory: $(SolutionDir)bin\$(Platform)\$(Configuration)\
5. Configuration Properties/General/Intermediate Directory: $(SolutionDir)bin\intermediates\$(Platform)\$(Configuration)\
6. Click OK
7. in Solution Explorer right click on Solution and “Clean Solution”
8. Delete all Debug directories.

**Adding GLFW Libtaty (for creating a window, that supports multiple platforms.)**

1. Copy code from <https://www.glfw.org/documentation.html> to Main.cpp
2. Download <https://www.glfw.org/download.html> “32-bit Windows binaries”
3. In Solution (main) Folder create “Dependencies” Folder, inside create “GLFW“ folder
4. Copy from downloaded zip (glfw-3.2.1.bin.WIN32.zip\glfw-3.2.1.bin.WIN32) “include” and “lib-vc2015” folders into “GLFW“
5. From Dependencies/GLFW/lib-vc2015 folder delete all, but glfw3.lib (we don’t need dynamic \*.dll libraries)
6. Open MainManu>Project>Properties/
7. **“Configuration:”** dropdown set to “**All Configurations”**.
8. **“Platforms:”** dropdown set to “**All Platforms”**.
9. C++>General/Additional Include Directories: $(SolutionDir)Dependencies\GLFW\include;
10. Linker/General/Additional Library Directories: $(SolutionDir)Dependencies\GLFW\lib-vc2015;
11. Linker/Input/Additional Dependencies: glfw3.lib;opengl32.lib;User32.lib;Gdi32.lib;Shell32.lib;
12. Build
13. If “1>glfw3.lib(win32\_init.c.obj) : error LNK2019: unresolved external symbol \_\_imp\_\_\*\*\*@12 referenced in function” kind of errors in Output window,

* google \*\*\*.
* Open <https://docs.microsoft.com> …
* Find Requirements>Library \*.lib and add this name to the Linker/Input/Additional Dependencies:

1. Draw “legacy openGL” triangle for the testing purpose: (after glClear() add)

glBegin(GL\_TRIANGLES);

glVertex2f(-0.5f, -0.5f);

glVertex2f(0.0f, 0.5f);

glVertex2f(0.5f, -0.5f);

glEnd();

**Modern OpenGL –** **GLEW (alternatives glad gl3w): Accessing OpenGL functions on multiplatform.**

1. Download binadies from <http://glew.sourceforge.net/> > Windows 32-bit and 64-bit
2. Copy from glew-2.1.0-win32.zip the glew-2.1.0 folder into Dependencies folder.
3. Rename glew-2.1.0 folder to GLEW
4. Read Dependencies\GLEW\doc\index.html info
5. Open MainManu>Project>Properties/
6. **“Configuration:”** dropdown set to “**All Configurations”**.
7. **“Platforms:”** dropdown set to “**All Platforms”**.
8. C++>General/Additional Include Directories: $(SolutionDir)Dependencies\GLEW\include;
9. (path from glew32s.lib file) Linker/General/Additional Library Directories: $(SolutionDir)Dependencies\GLEW\lib\Release\Win32
10. Add to Linker/Input/Additional Dependencies: glew32s.lib;
11. Ok
12. Place “#include <GL/glew.h>” before everything else
13. Add to: Properties>Configuration Properties>C/C++>Preprocessor> Preprocessor Definitions: GLEW\_STATIC;
14. Ok
15. To test if it works add after glfwMakeContextCurrent(window); line (donf forget #include <iostream>):

if (glewInit() != GLEW\_OK)

std::cout << "glew error" << std::endl;

else

std::cout << glGetString(GL\_VERSION) << std::endl;

**Draw Triangle with GLEW**

1. Keep open <http://docs.gl/> can help with OpenGL functions.
2. Add before loop:

unsigned int buffer;//id for buffer

glGenBuffers(1, &buffer);//assigns id

glBindBuffer(GL\_ARRAY\_BUFFER, buffer);//selects buffer

glBufferData(GL\_ARRAY\_BUFFER, 6 \* sizeof(float), positions, GL\_STATIC\_DRAW);//set size f buffer, static or dinamic, and give a data, that is oprional (can be NULL, and data can be assigned later)

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, 2 \* sizeof(float), 0);//aattribute, how many variables, data type, no normalize, size of one vertex, offset where attribute starts

1. Change old glBegin…glEnd code in loop to:

//since we dont have index buffer yet:

glDrawArrays(GL\_TRIANGLES,0,3);//type, start position, number of vertexes. //It draws the buffer blinded before with glBindBuffer()

**Shaders**

**Fragment Shader- called for every pixel**

**Vertex Shader – only several times (less expensive)**

1. Create shader compiler function:

static unsigned int CompileShader(unsigned int type, const std::string& source)

{

unsigned int id = glCreateShader(type);

const char\* src = source.c\_str();

glShaderSource(id, 1, &src, nullptr);

glCompileShader(id);

return id;

}

1. Add error tracer to the function (add lines before “return” ). In traces sintax error in shaders:

int result;

glGetShaderiv(id, GL\_COMPILE\_STATUS, &result);

if (result == GL\_FALSE)

{

int length;

glGetShaderiv(id, GL\_INFO\_LOG\_LENGTH, &length);

char\* message = (char\*)alloca(length \* sizeof(char));

glGetShaderInfoLog(id, length, &length, message);

std::cout << "failed to compile"<<

(type==GL\_VERTEX\_SHADER? "vertex": "fragment")

<<" Shader"<< std::endl;

std::cout << message << std::endl;

glDeleteShader(id);

return 0;

}

1. Add create shader function:

static unsigned int CreateShader(const std::string& vertexSchader, const std::string& fragmentSchader)

{

unsigned int program = glCreateProgram();

unsigned int vs = CompileShader(GL\_VERTEX\_SHADER, vertexSchader);

unsigned int fs = CompileShader(GL\_FRAGMENT\_SHADER, fragmentSchader);

glAttachShader(program,vs);

glAttachShader(program,fs);

glLinkProgram(program);

glValidateProgram(program);

glDeleteShader(vs);

glDeleteShader(fs);

return program;

}

1. Write shaders (inside main function):

std::string vertexShader =

"#version 330 core\n"

"\n"

"layout(location = 0) in vec4 position;"

"\n"

"void main()\n"

"{\n"

" gl\_Position = position;\n"

"}\n";

std::string fragmentShader =

"#version 330 core\n"

"\n"

"layout(location = 0) out vec4 color;"

"\n"

"void main()\n"

"{\n"

" color = vec4(1.0, 0.0, 0.0, 1.0);\n"

"}\n";

1. Add following lines (before the loop):

unsigned int shader = CreateShader(vertexShader, fragmentShader);

glUseProgram(shader);

1. Add following lines (after the loop):

glDeleteProgram(shader);

shaders in external files

1. Needs several includes:

#include <iostream>

#include <fstream>

#include <string>

1. Following function takes file link and returnes string:

std::string StringFromFile(const std::string& file\_link)

{

std::ifstream ifs(file\_link);

std::string s;

std::getline(ifs, s, (char)ifs.eof());

return s;

}

1. “shader” variable line is changed to:

unsigned int shader = CreateShader(StringFromFile("res/vertexShader.shader"), StringFromFile("res/fragmentShader.shader"));

Index Buffers

(prevents us from repeating vertex data)

1. Change “positions” array to:

float positions[] =

{

-0.5f, -0.5f,

0.5f, -0.5f,

0.5f, 0.5f,

-0.5f, 0.5f

};

1. Add index array:

unsigned int indices[] =

{

0,1,2,

2,3,0

};

1. Modify following line to grant it with dynamic size:

glBufferData(GL\_ARRAY\_BUFFER, (sizeof(positions) / sizeof(\*positions)) \* sizeof(float), positions, GL\_STATIC\_DRAW);

1. Add Buffer for Index as well:

unsigned int ibo;

glGenBuffers(1, &ibo);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, ibo);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, (sizeof(indices) / sizeof(\*indices)) \* sizeof(unsigned int), indices, GL\_STATIC\_DRAW);

1. Change line “glDrawArrays(GL\_TRIANGLES,0,3);” to:

glDrawElements(GL\_TRIANGLES,6,GL\_UNSIGNED\_INT, nullptr);//type, number of indexes, data type, pointer to index buffer

Dealing With OpenGL Errors

glGetError or glDebugMessageCallback (from4.3 version)

1. Add the following lines after the last “#define”:

//OpenGL Debuging

#define ASSERT(x) if (!(x)) \_\_debugbreak();

#define GLCall(x) GLClearError();\

x;\

ASSERT(GLLogCall())

static void GLClearError()

{

while (glGetError()!=GL\_NO\_ERROR);

}

static bool GLLogCall()

{

while (GLenum error = glGetError())

{

std::cout << "[OpenGL Error]"<< error << std::endl;

return false;

}

return true;

}

1. Wrap every gl line of code into GLCall(………………);

Uniforms in OpenGL(get data from CPU into a shader)

1. To send variable to fragmentShader.shader add to it before the main function the following line:

uniform vec4 u\_Color;

1. To use if change the line in main function:

color = u\_Color;

1. In the main program (Main.cpp) after the shader is bound (“GLCall(glUseProgram(shader));”) get the location of the uniform:

int location = glGetUniformLocation(shader, "u\_Color");

1. Send data to defined location (to animate a parameter move this line inside loop with some variable instead of float):

glUniform4f(location, 1.0f, 0.5f, 0.0f, 1.0f);

1. To animate variable (0.0-1.0):

Before loop:

float n = 0.0f;

Inside loop:

n =n + 0.01f;

float animNormalizedFloat = (float)((int)(n\*100.0f) % 100) / 100.0f;

(and e.g. glUniform4f(location, animNormalizedFloat, 0.5f, 0.0f, 1.0f); inside loop)

1. ggg