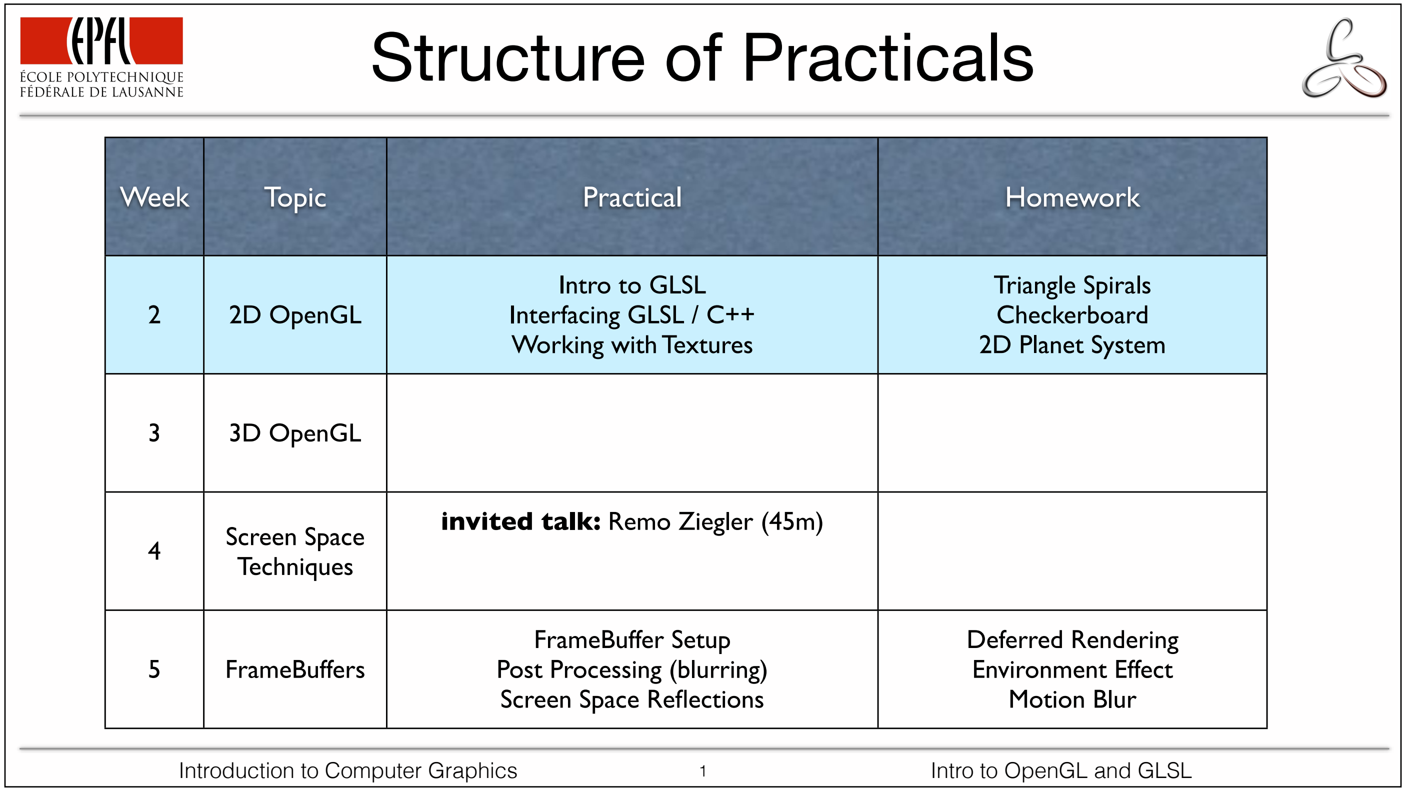
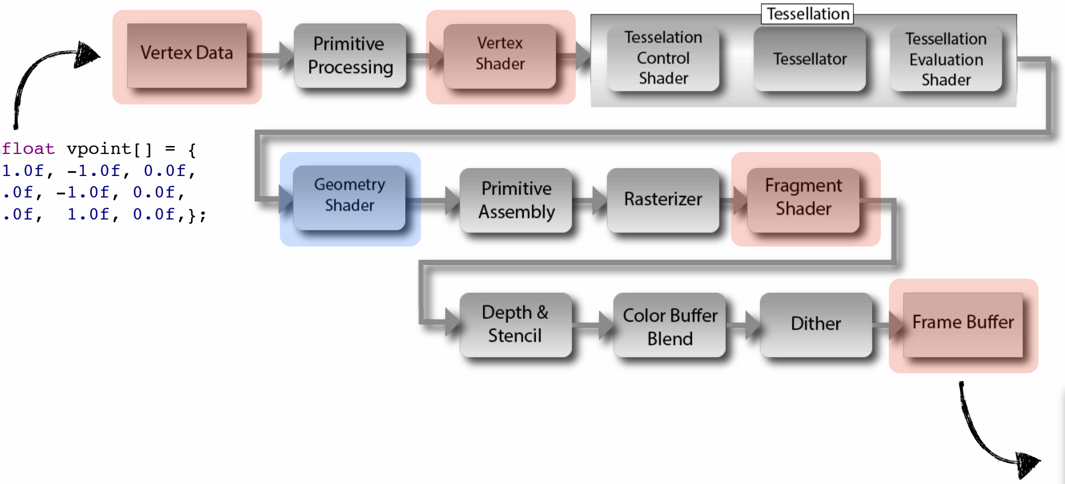


Structure of Practicals

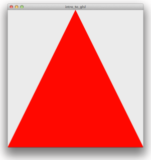
Week	Topic	Practical	Homework
2	2D OpenGL	Intro to GLSL Interfacing GLSL / C++ Working with Textures	Triangle Spirals Checkerboard 2D Planet System
3	3D OpenGL		
4	Screen Space Techniques	**invited talk:** Remo Ziegler (45m)	
5	FrameBuffers	FrameBuffer Setup Post Processing (blurring) Screen Space Reflections	Deferred Rendering Environment Effect Motion Blur

Mention how you cannot get started on HW without finishing practical

OpenGL Pipeline



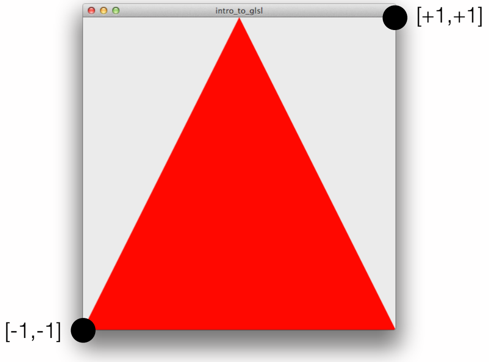
```
const GLfloat vpoint[] = {
       -1.0f, -1.0f, 0.0f,
       1.0f, -1.0f, 0.0f,
       0.0f,  1.0f, 0.0f,};
```



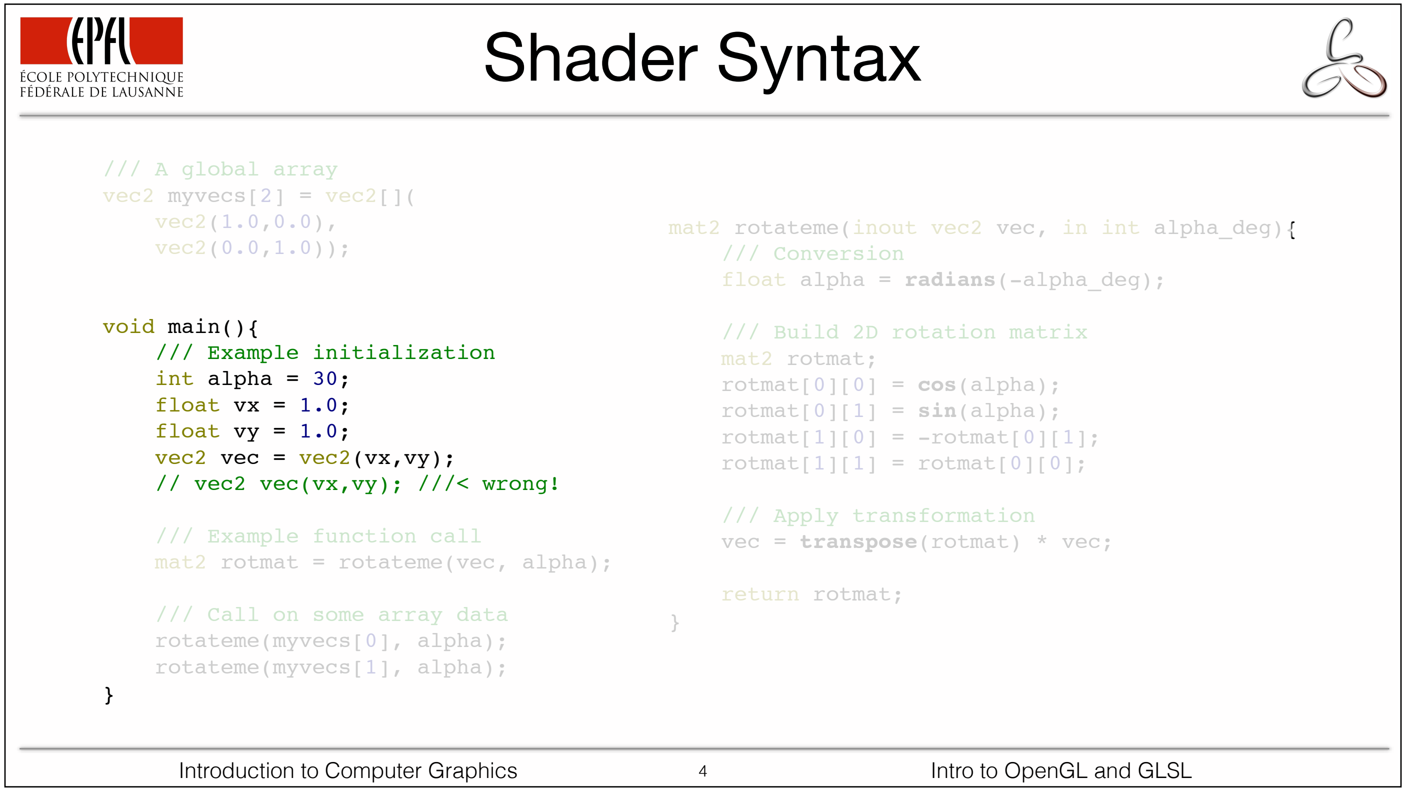
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Vertex / Fragment Shaders

```
const GLfloat vpoint_buffer[] = {
 /*V1*/ -1.0f, -1.0f, 0.0f,
 /*V2*/  1.0f, -1.0f, 0.0f,
 /*V3*/  0.0f,  1.0f, 0.0f};


// intro_to_glsl/triangle_vshader.glsl
#version 330 core
in vec3 vpoint;
void main(){
    gl_Position = vec4(vpoint, 1.0);
}

// intro_to_glsl/triangle_fshader.glsl
#version 330 core
out vec3 color;
void main(){
    color = vec3(1.0, 0.0, 0.0);
}
```



- the output window represents the [-1,+1]^2 domain (please do not resize the window)
- gl_Position output is implicitly defined in vshaders: http://www.opengl.org/wiki/Built-in_Variable_(GLSL)
- note that our positions must be converted to vec4 before assigning to gl_Position
- gl_FragColor is obsolete in core OpenGL >= 3.1



Shader Syntax

```
/// A global array
vec2 myvecs[2] = vec2[](
    vec2(1.0,0.0),
    vec2(0.0,1.0));


void main(){
    /// Example initialization
    int alpha = 30;
    float vx = 1.0;
    float vy = 1.0;
    vec2 vec = vec2(vx,vy);
    // vec2 vec(vx,vy); ///< wrong!

    /// Example function call
    mat2 rotmat = rotateme(vec, alpha);

    /// Call on some array data
    rotateme(myvecs[0], alpha);
    rotateme(myvecs[1], alpha);
}
```

```
mat2 rotateme(inout vec2 vec, in int alpha_deg){
    /// Conversion
    float alpha = radians(-alpha_deg);

    /// Build 2D rotation matrix
    mat2 rotmat;
    rotmat[0][0] = cos(alpha);
    rotmat[0][1] = sin(alpha);
    rotmat[1][0] = -rotmat[0][1];
    rotmat[1][1] = rotmat[0][0];

    /// Apply transformation
    vec = transpose(rotmat) * vec;

    return rotmat;
}
```

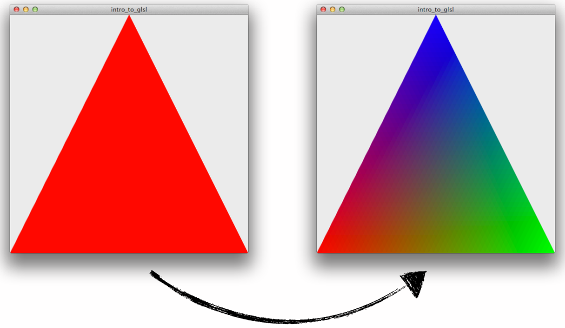
alternative ways of accessing a vec3:
- gl_Position.xyz = position;
- gl_Position.w = 1.0;

Task: color interpolation

linear interpolation

```
#version 330 core
in vec3 vpoint;
out vec3 fcolor;
void main() {
    gl_Position = vec4(vpoint, 1.0);
    fcolor = vec3(1,0,0);
}
```

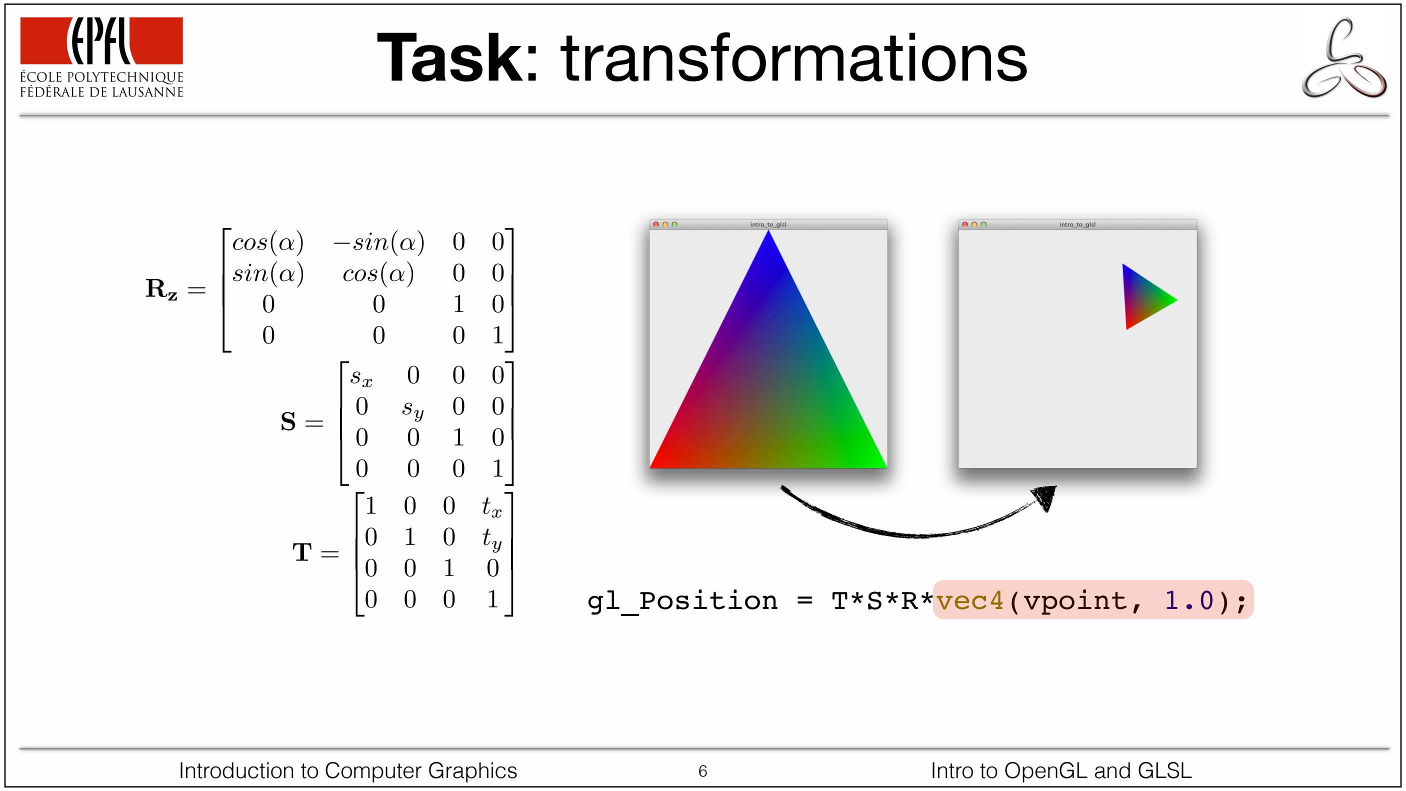
```
#version 330 core
in vec3 fcolor;
out vec3 color;
void main() {
    color = fcolor;
}
```



What to use?

- `const vec3 COLORS[3] = ...`
- `gl_VertexID (vshader)`

note that the out of vshader must have the same name of in fshader



Task: transformations

$$\mathbf{R_z} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 & 0 \\ sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

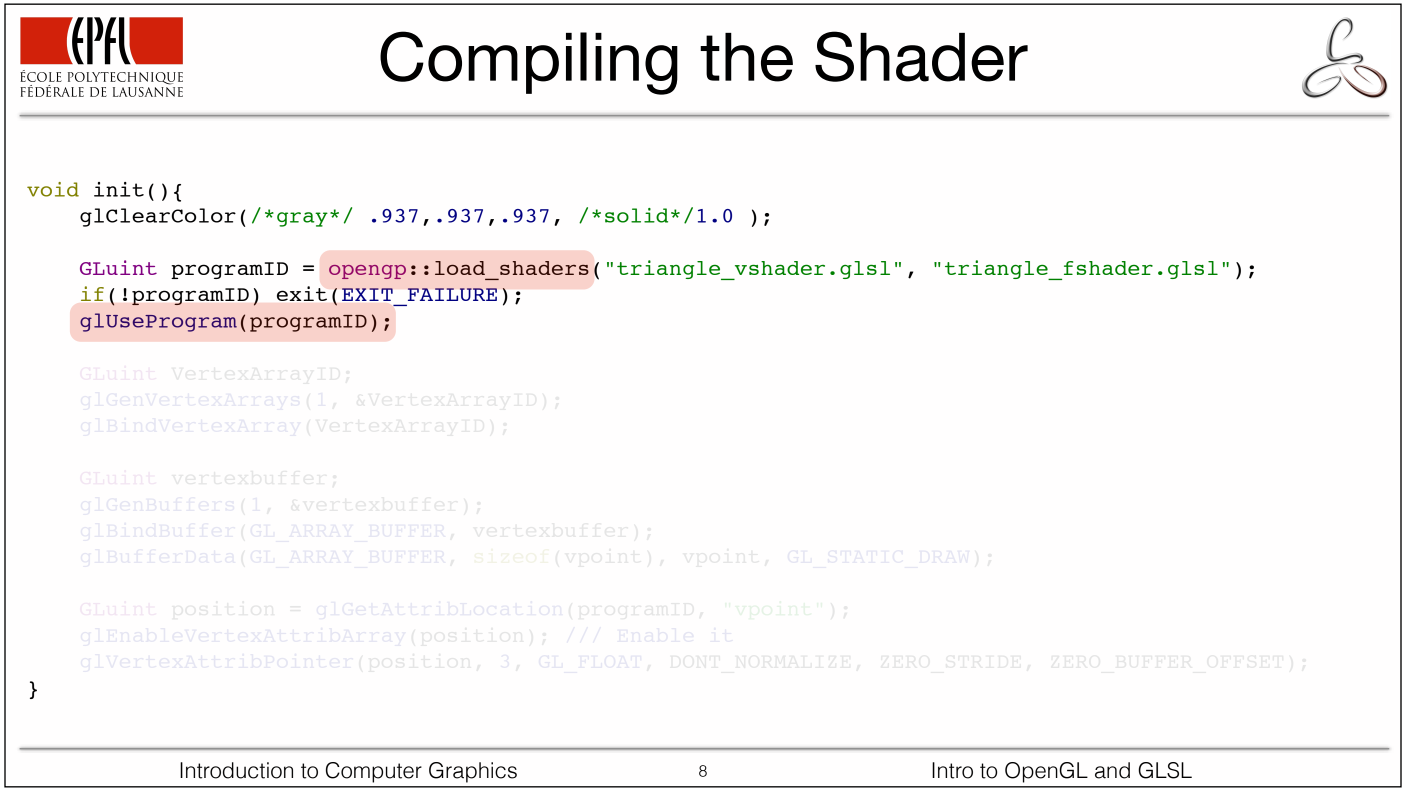
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
gl_Position = T*S*R*vec4(vpoint, 1.0);
```

- note before proceeding we must convert vertex position into homogeneous (vec4)
- note that above we apply rotation first, then scale, then translation
- note that T,S,R are mat4.
- mat4 M = mat4(1); creates an identity matrix
- M[col][row]: note the first index is COLUMN!!!!

C++ / GLSL interface

```
int main(int, char**){
    glfwInitWindowSize(512, 512);
    glfwCreateWindow("intro_to_glsl");
    glfwDisplayFunc(display);
    init();
    glfwMainLoop();
    return EXIT_SUCCESS;
}
```



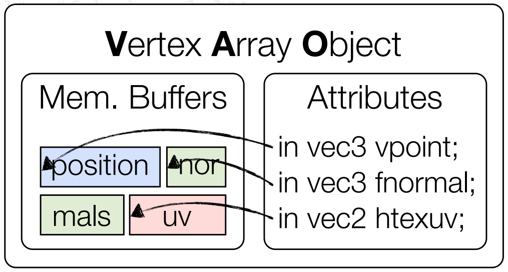
Compiling the Shader

```
void init(){
    glClearColor(/*gray*/ .937,.937,.937, /*solid*/1.0 );

    GLuint programID = opengp::load_shaders("triangle_vshader.glsl", "triangle_fshader.glsl");
    if(!programID) exit(EXIT_FAILURE);
    glUseProgram(programID);

    GLuint VertexArrayID;
    glGenVertexArrays(1, &VertexArrayID);
    glBindVertexArray(VertexArrayID);

    GLuint vertexbuffer;
    glGenBuffers(1, &vertexbuffer);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vpoint), vpoint, GL_STATIC_DRAW);

    GLuint position = glGetAttribLocation(programID, "vpoint");
    glEnableVertexAttribArray(position); /// Enable it
    glVertexAttribPointer(position, 3, GL_FLOAT, DONT_NORMALIZE, ZERO_STRIDE, ZERO_BUFFER_OFFSET);
}
```

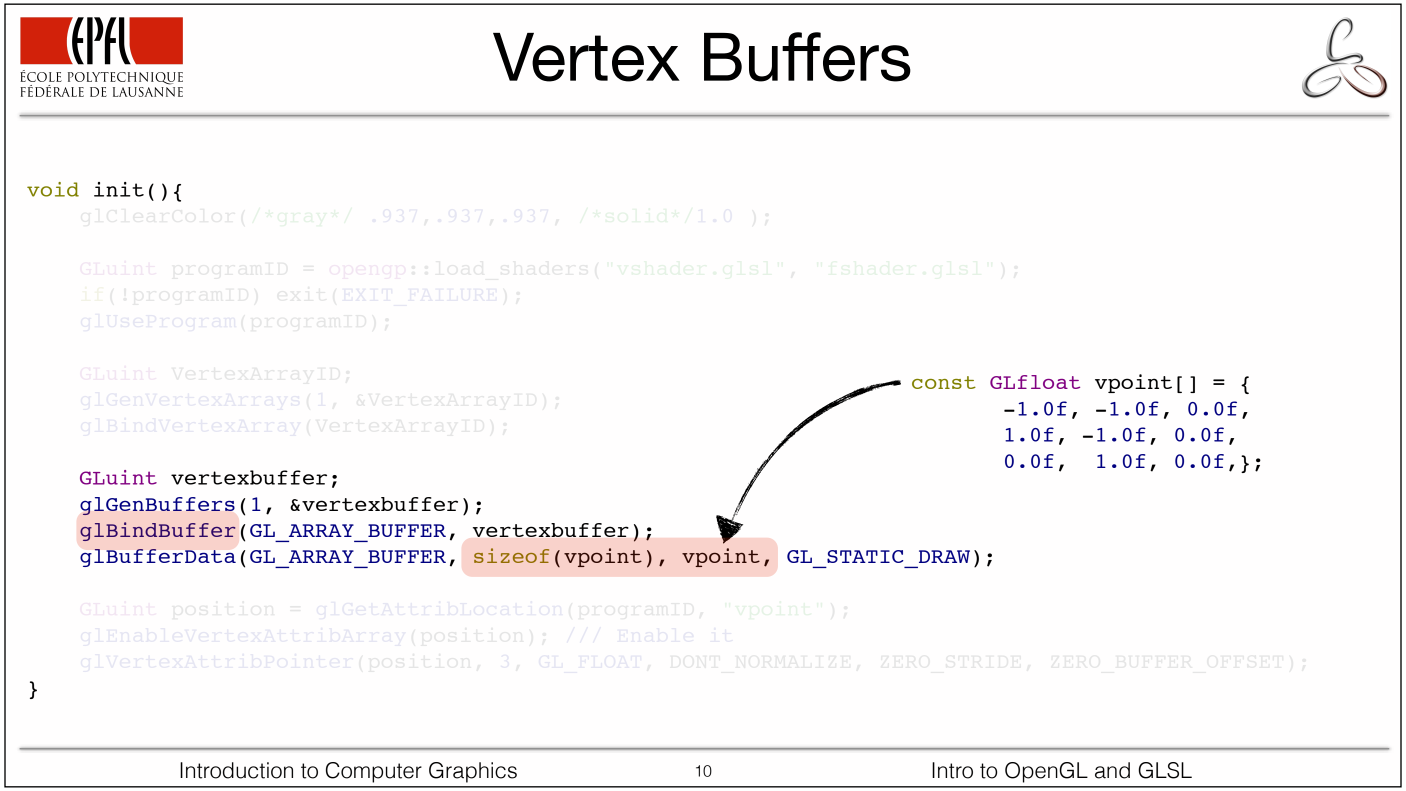
glUseProgram: before setting uniforms and before drawing!

Vertex Array Object

```
void init(){
    glClearColor(/*gray*/ .937,.937,.937, /*solid*/1.0 );

    GLuint programID = opengp::load_shaders("vshader.gls
    if(!programID) exit(EXIT_FAILURE);
    glUseProgram(programID);

    GLuint VertexArrayID;
    glGenVertexArrays(1, &VertexArrayID);
    glBindVertexArray(VertexArrayID);

    GLuint vertexbuffer;
    glGenBuffers(1, &vertexbuffer);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vpoint), vpoint, GL_STATIC_DRAW);

    GLuint position = glGetAttribLocation(programID, "vpoint");
    glEnableVertexAttribArray(position); /// Enable it
    glVertexAttribPointer(position, 3, GL_FLOAT, DONT_NORMALIZE, ZERO_STRIDE, ZERO_BUFFER_OFFSET);
}
```



VertexArrayObjects: is a container that wraps data (buffers) and its specification (attributes)
In initialization we fill-in the container, when we draw we just have to “bind” it
Creating it before anything else is **mandatory** in modern OpenGL/GPUs

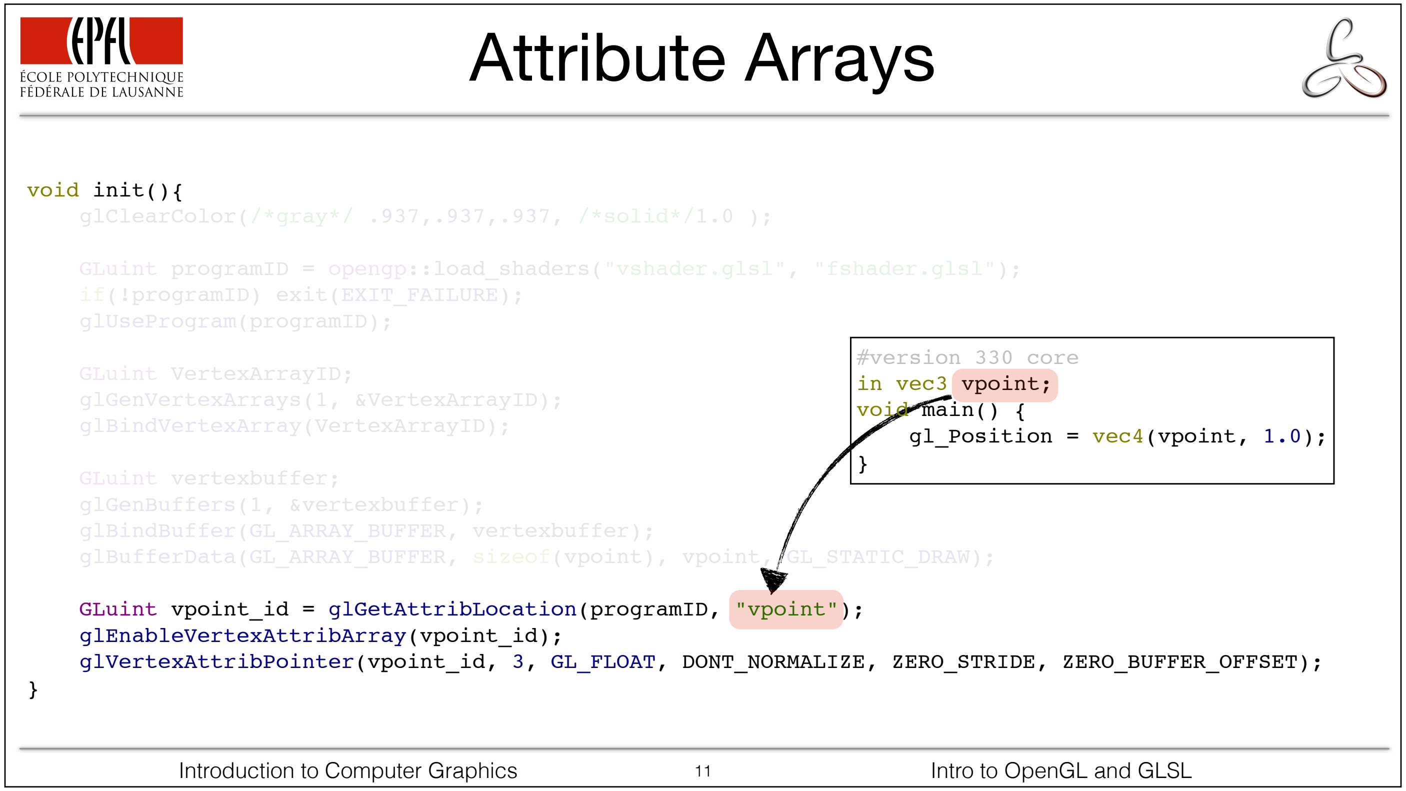


Vertex Buffers

```
void init(){
    glClearColor(/*gray*/ .937,.937,.937, /*solid*/1.0 );

    GLuint programID = opengp::load_shaders("vshader.glsl", "fshader.glsl");
    if(!programID) exit(EXIT_FAILURE);
    glUseProgram(programID);

    GLuint VertexArrayID;
    glGenVertexArrays(1, &VertexArrayID);
    glBindVertexArray(VertexArrayID);

    GLuint vertexbuffer;
    glGenBuffers(1, &vertexbuffer);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vpoint), vpoint, GL_STATIC_DRAW);

    GLuint position = glGetAttribLocation(programID, "vpoint");
    glEnableVertexAttribArray(position); /// Enable it
    glVertexAttribPointer(position, 3, GL_FLOAT, DONT_NORMALIZE, ZERO_STRIDE, ZERO_BUFFER_OFFSET);
}
```

```
const GLfloat vpoint[] = {
       -1.0f, -1.0f, 0.0f,
       1.0f, -1.0f, 0.0f,
       0.0f,  1.0f, 0.0f,};
```

We need to specify its location, its size and a “hint” of how we will be using it (optimization).

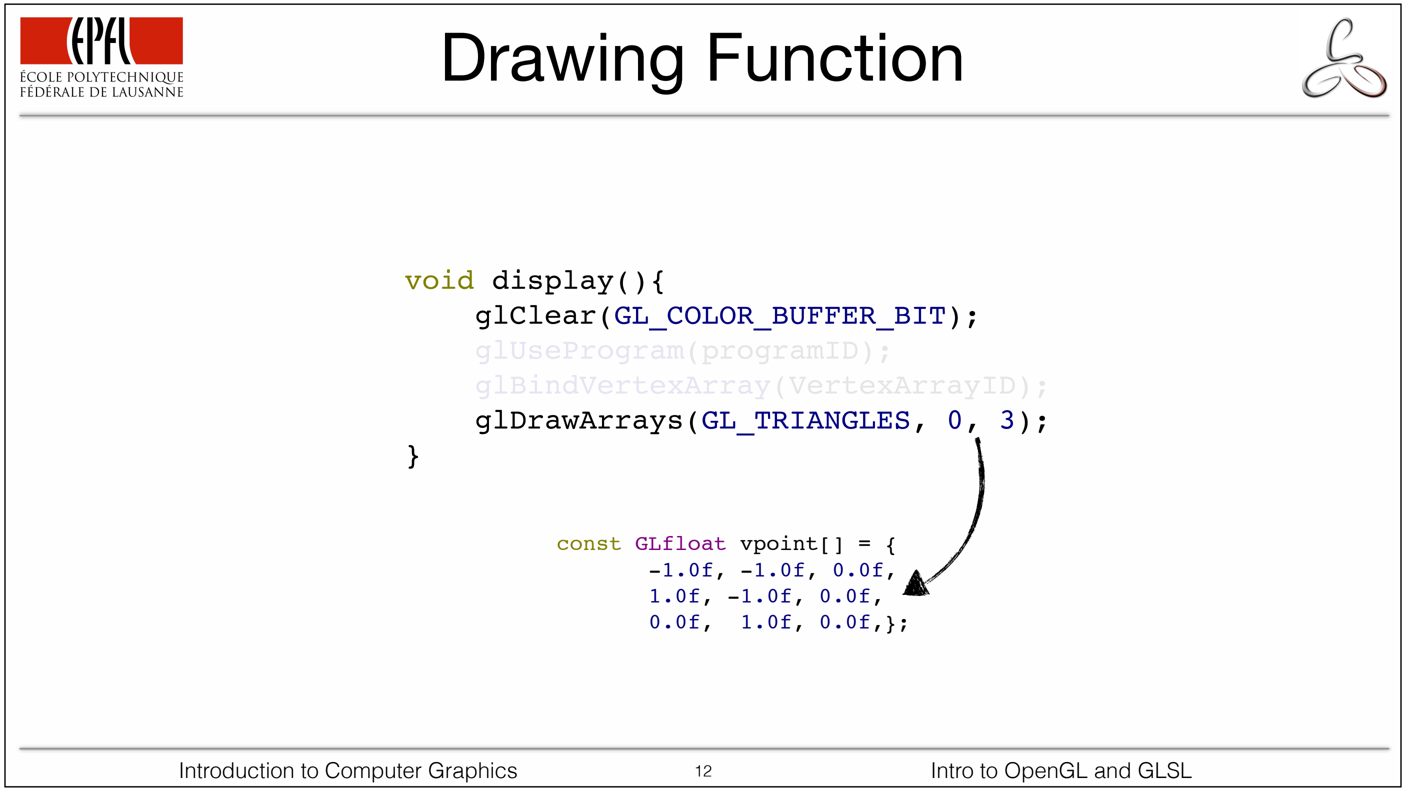


Attribute Arrays

```
void init(){
    glClearColor(/*gray*/ .937,.937,.937, /*solid*/1.0 );

    GLuint programID = opengp::load_shaders("vshader.glsl", "fshader.glsl");
    if(!programID) exit(EXIT_FAILURE);
    glUseProgram(programID);

    GLuint VertexArrayID;
    glGenVertexArrays(1, &VertexArrayID);
    glBindVertexArray(VertexArrayID);

    GLuint vertexbuffer;
    glGenBuffers(1, &vertexbuffer);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vpoint), vpoint, GL_STATIC_DRAW);

    GLuint vpoint_id = glGetAttribLocation(programID, "vpoint");
    glEnableVertexAttribArray(vpoint_id);
    glVertexAttribPointer(vpoint_id, 3, GL_FLOAT, DONT_NORMALIZE, ZERO_STRIDE, ZERO_BUFFER_OFFSET);
}
```

```
#version 330 core
in vec3 vpoint;
void main() {
    gl_Position = vec4(vpoint, 1.0);
}
```

Note that the description in glVertexAttribPointer applies to the vertexbuffer that was bound by glBindBuffer!
The bound buffer contains GL_FLOATs, each attribute is composed by 3 elements (thus vec3) and... nothing else special



Drawing Function

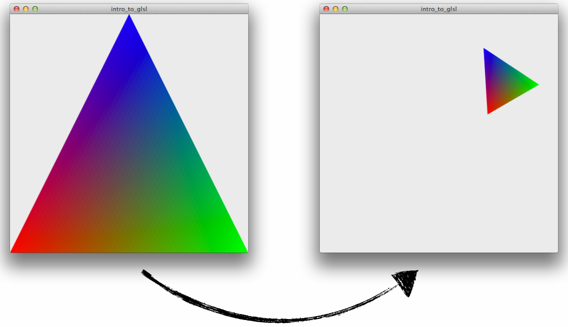
```
void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    glUseProgram(programID);
    glBindVertexArray(VertexArrayID);
    glDrawArrays(GL_TRIANGLES, 0, 3);
}
```

```
const GLfloat vpoint[] = {
     -1.0f, -1.0f, 0.0f,
     1.0f, -1.0f, 0.0f,
     0.0f,  1.0f, 0.0f,};
```

we are drawing “3” points
in the buffer we start with offset “0”

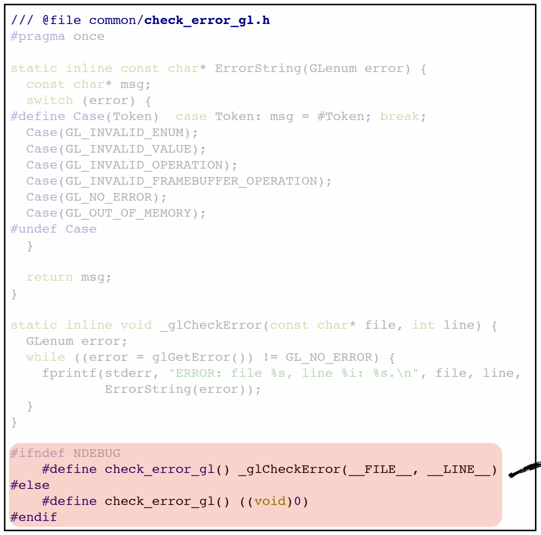
TASK: Shader Uniforms

```
uniform mat4 M;
in vec3 vpoint;
out vec3 fcolor;

void main() {
    gl_Position = M*vec4(vpoint, 1.0);
    fcolor = ...
}


void init(){
    /// compile and bind shader

    mat4 M; ///< typedef Eigen:Matrix4f mat4;
    M(0,0) = ...
    GLuint M_id = glGetUniformLocation(_pid, "M");
    glUniformMatrix4fv(M_id, 1, GL_FALSE, M.data());
}
```



http://eigen.tuxfamily.org/dox/AsciiQuickReference.txt

OpenGL Error Checking



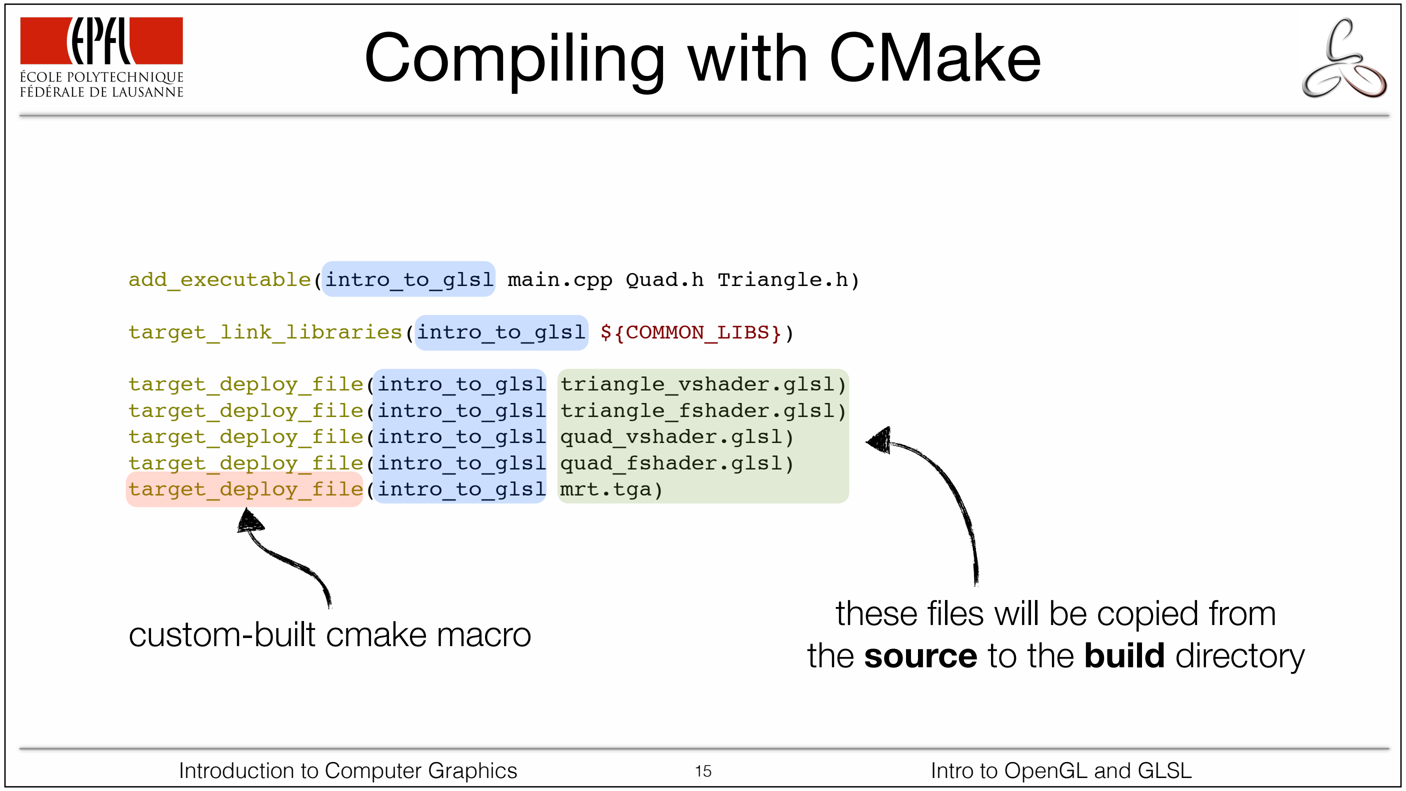
```
/// @file common/check_error_gl.h
#pragma once

static inline const char* ErrorString(GLenum error) {
  const char* msg;
  switch (error) {
#define Case(Token)  case Token: msg = #Token; break;
  Case(GL_INVALID_ENUM);
  Case(GL_INVALID_VALUE);
  Case(GL_INVALID_OPERATION);
  Case(GL_INVALID_FRAMEBUFFER_OPERATION);
  Case(GL_NO_ERROR);
  Case(GL_OUT_OF_MEMORY);
#undef Case
  }

  return msg;
}

static inline void _glCheckError(const char* file, int line) {
  GLenum error;
  while ((error = glGetError()) != GL_NO_ERROR) {
    fprintf(stderr, "ERROR: file %s, line %i: %s.\n", file, line,
            ErrorString(error));
  }
}

#ifndef NDEBUG
    #define check_error_gl() _glCheckError(__FILE__, __LINE__)
#else
    #define check_error_gl() ((void)0)
#endif
```

```
2   #include "check_error_gl.h"

3   void init(){
        ///...

27      GLuint VertexArrayID;
28      glGenVertexArrays(1, &VertexArrayID);
29      check_error_gl();
30      glBindVertexArray(1234567890);
31      check_error_gl();

        ///...
    }
```

```
ERROR: file /Users/andrea/Developer/icg15/src/intro_to_glsl/main.cpp, line 31: GL_INVALID_OPERATION.
```

!!!ONLY ACTIVE IN DEBUG MODE

```
set(CMAKE_BUILD_TYPE "Release")
set(CMAKE_BUILD_TYPE "Debug")
```



Compiling with CMake

```
add_executable(intro_to_glsl main.cpp Quad.h Triangle.h)

target_link_libraries(intro_to_glsl ${COMMON_LIBS})

target_deploy_file(intro_to_glsl triangle_vshader.glsl)
target_deploy_file(intro_to_glsl triangle_fshader.glsl)
target_deploy_file(intro_to_glsl quad_vshader.glsl)
target_deploy_file(intro_to_glsl quad_fshader.glsl)
target_deploy_file(intro_to_glsl mrt.tga)
```

custom-built cmake macro

these files will be copied from
the **source** to the **build** directory

shaders and textures need to be found at runtime by the executable!
target_deploy_file is a custom macro we defined in **icg_settings.cmake**

Task: refactoring C++

```
#include "icg_common.h"

#include "Triangle.h"
Triangle triangle;

void init(){
    glClearColor(.9,.9,.9,1);
    triangle.init();
}
void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    triangle.draw();
}
int main(int, char**){
    glfwInitWindowSize(512, 512);
    glfwCreateWindow("intro_to_glsl");
    glfwDisplayFunc(display);
    init();
    glfwMainLoop();
    triangle.cleanup();
    return EXIT_SUCCESS;
}
```

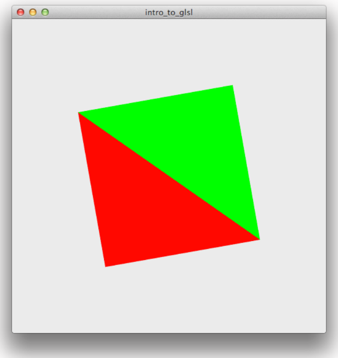
Triangle.h

```
#pragma once
#include "icg_common.h"

class Triangle{
private:
    GLuint _vao; ///< Vertex array objects
    GLuint _pid; ///< GLSL program ID
    GLuint _tex; ///< Texture IDs

public:
    void init(){
        // ...
    }

    void cleanup(){
        // ...
    }

    void draw(){
        // ...
    }
};
```

Task: create Quad.h

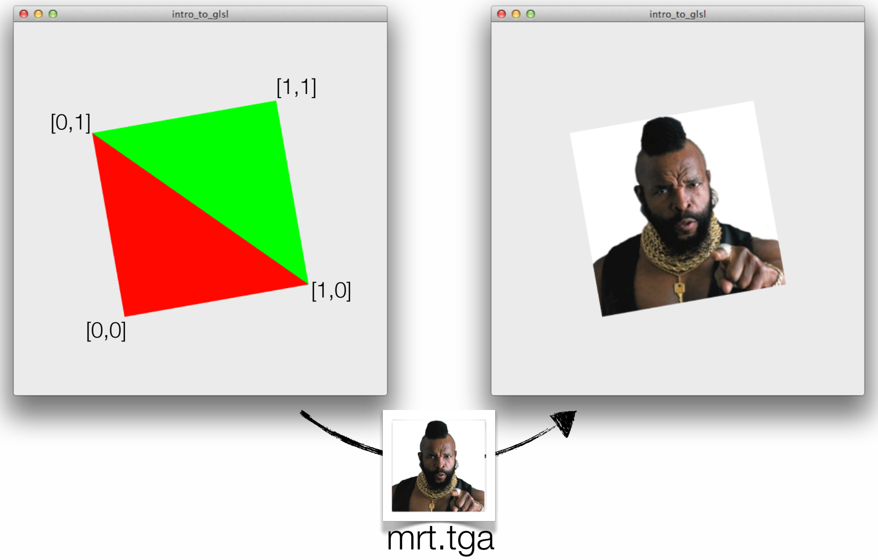
```
const GLfloat vpoint[] = {
    /*V1*/ -1.0f, -1.0f, 0.0f,
    /*V2*/ +1.0f, -1.0f, 0.0f,
    /*V3*/ -1.0f, +1.0f, 0.0f,
    /*V4*/ +1.0f, +1.0f, 0.0f };


glDrawArrays(GL_TRIANGLE_STRIP, ...);
```

```
#version 330 core
out vec3 color;
const vec3 COLORS[2] = vec3[](
    vec3(1.0,0.0,0.0),
    vec3(0.0,1.0,0.0));
void main() {
    color = COLORS[gl_PrimitiveID];
}
```



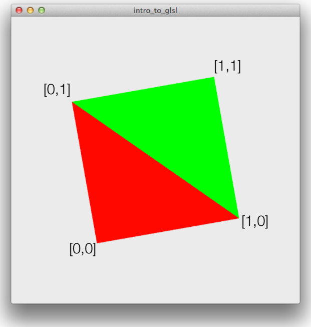
gl_PrimitiveID is a pre-defined variable for fragment shaders

Texturing the quad

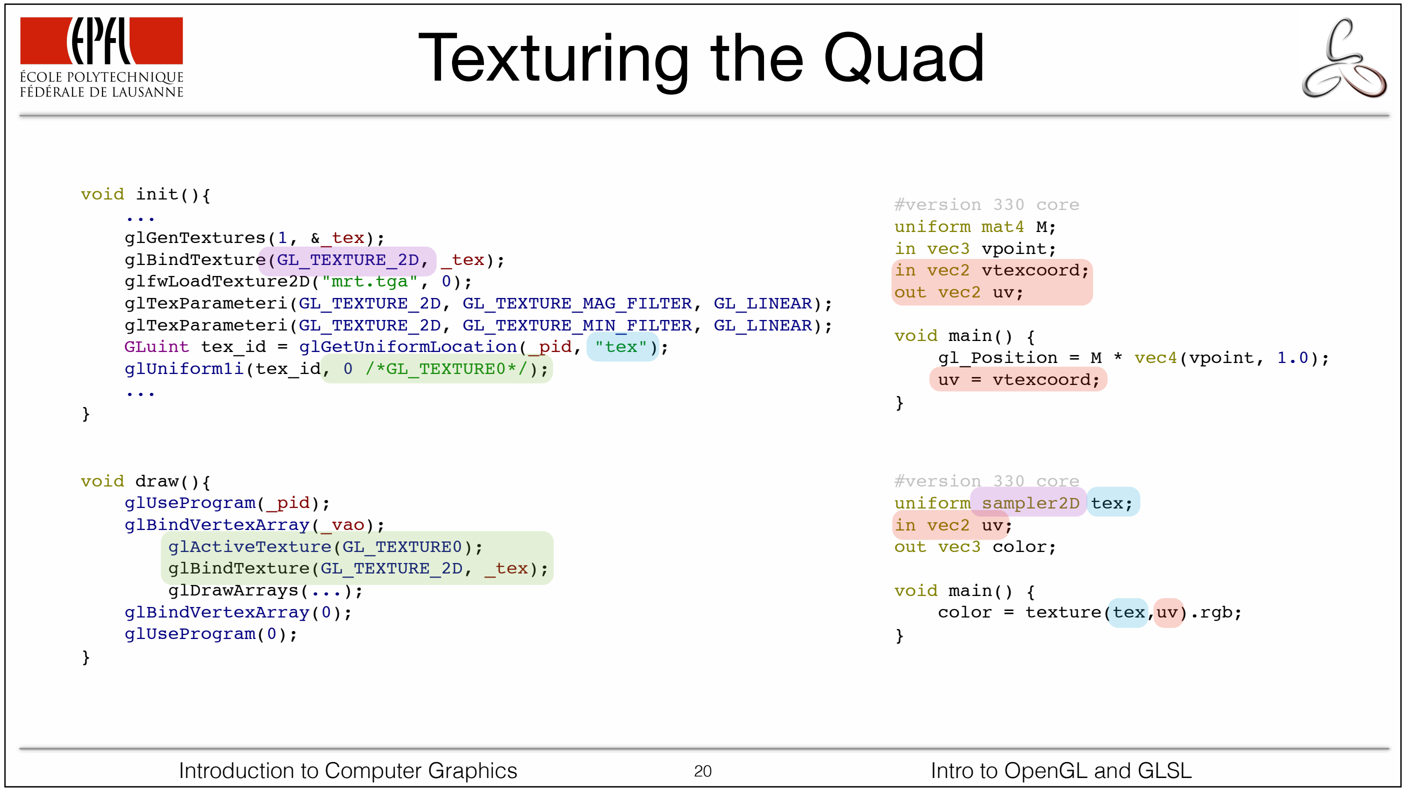


Vertex Texture Coordinates

```
void init(){
    ...
    {
        const GLfloat vtexcoord[] = { /*V1*/ 0.0f, 0.0f,
                                      /*V2*/ 1.0f, 0.0f,
                                      /*V3*/ 0.0f, 1.0f,
                                      /*V4*/ 1.0f, 1.0f};
        ///--- Buffer
        glGenBuffers(1, &_vbo);
        glBindBuffer(GL_ARRAY_BUFFER, _vbo);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vtexcoord), vtexcoord, GL_STATIC_DRAW);

        ///--- Attribute
        GLuint vtexcoord_id = glGetAttribLocation(_pid, "vtexcoord");
        glEnableVertexAttribArray(vtexcoord_id);
        glVertexAttribPointer(vtexcoord_id, 2, GL_FLOAT,
                              DONT_NORMALIZE, ZERO_STRIDE, ZERO_BUFFER_OFFSET);
    }
    ...
}
```



```
in vec2 vtexcoord; ///< for vshader only!!
```



Texturing the Quad

```
void init(){
    ...
    glGenTextures(1, &_tex);
    glBindTexture(GL_TEXTURE_2D, _tex);
    glfwLoadTexture2D("mrt.tga", 0);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    GLuint tex_id = glGetUniformLocation(_pid, "tex");
    glUniform1i(tex_id, 0 /*GL_TEXTURE0*/);
    ...
}
```

```
#version 330 core
uniform mat4 M;
in vec3 vpoint;
in vec2 vtexcoord;
out vec2 uv;

void main() {
    gl_Position = M * vec4(vpoint, 1.0);
    uv = vtexcoord;
}
```

```
void draw(){
    glUseProgram(_pid);
    glBindVertexArray(_vao);
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, _tex);
        glDrawArrays(...);
    glBindVertexArray(0);
    glUseProgram(0);
}
```

```
#version 330 core
uniform sampler2D tex;
in vec2 uv;
out vec3 color;

void main() {
    color = texture(tex,uv).rgb;
}
```

GL_TEXTURE_MIN/MAG_FILTER determines how texture will be interpolated upon rescaling

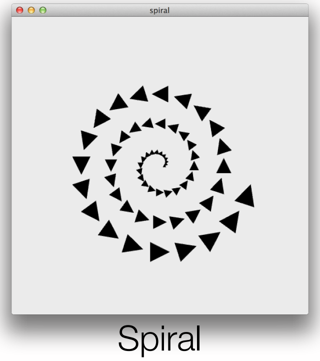
Homework 2

Submit **Hw1-Lastname-Lastname-Lastname.zip** with:

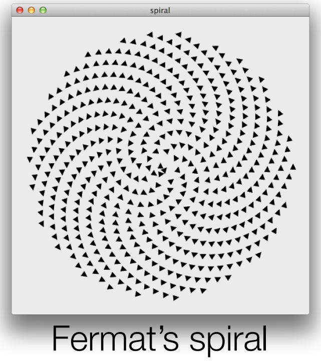
- source code - **do not** submit build folder!!!
  - **lose .5 points** if you include CMakeFiles, etc...
- executable - **must run** on the lab machines!!
  - **lose 1.5 points (each)** if the executable of an exercise doesn't run in the lab
- readme.txt - with any extra information

Each of the following exercises is **2 points** (**6 points total**)

Homework 2 - Part 1



Spiral



Fermat’s spiral

```
/// http://eigen.tuxfamily.org/dox/group__TutorialGeometry.html
typedef Eigen::Transform<float,3,Eigen::Affine> Transform;
Transform _M = Transform::Identity();
_M *= Eigen::Translation3f(Tx,Ty,0);
_M *= Eigen::AngleAxisf(alpha, Eigen::Vector3f::UnitZ());
_M *= Eigen::AlignedScaling3f(scale, scale, scale);
mat4 M = _M.matrix(); ///< v_new = TRA*ROT*SCA*v
```

$$r = c * \sqrt{\theta}$$

$$\theta = n * 137.508$$

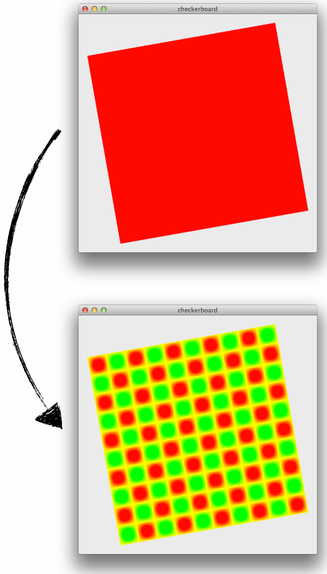
$$n = \{1, 2, \dots, N\}$$

(optional) use the Eigen tutorial on transformations:
Draw multiple triangles by calling glDrawArrays multiple times
Use a combination of rotation, scale and translation to obtain the necessary transformation
For the sunflower pattern, also see http://en.wikipedia.org/wiki/Fermat%27s_spiral

Homework 2 - Part 2

Create colormap 1D texture

```
///--- Create 1D texture (colormap)
{
    const int sz=3;
    GLfloat tex[3*sz] = {/*red*/    1.0, 0.0, 0.0,
                         /*yellow*/ 1.0, 1.0, 0.0,
                         /*green*/  0.0, 1.0, 0.0};
    glGenTextures(1, &_tex);
    glBindTexture(GL_TEXTURE_1D, _tex);
    glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB, sz, 0, GL_RGB, GL_FLOAT, tex);
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    GLuint tex_id = glGetUniformLocation(_pid, "colormap");
    glUniform1i(tex_id, 0 /*GL_TEXTURE0*/);
}
```

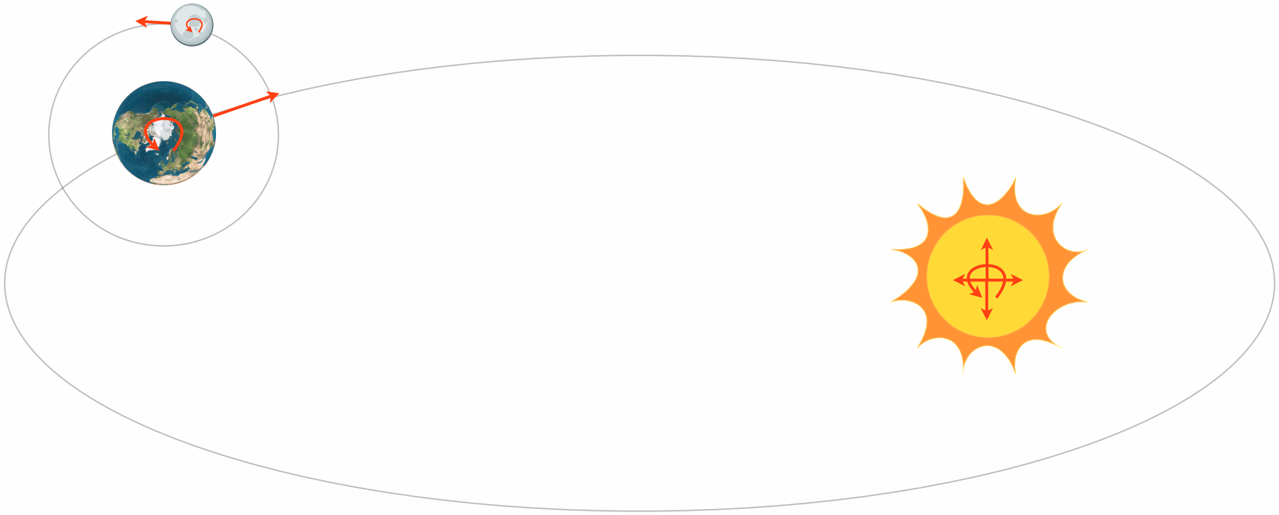


Access it in the fshader

```
uniform sampler1D colormap;
void main(){
    float value = ...
    color = texture(colormap, value).rgb;
}
```

you only need to modify the fragment shader
use sine function (argument is quad texture coordinate) to generate a value in the range [0,1]
use this value to access the colormap texture

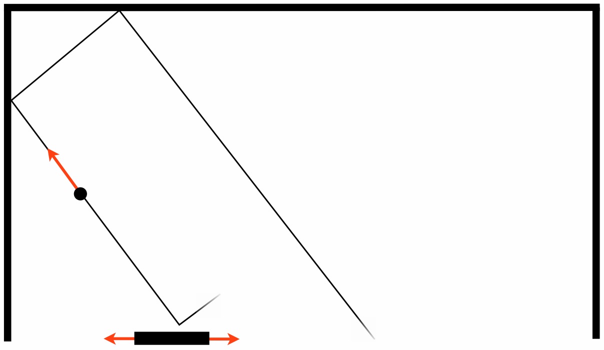
Homework 2 - Part 3



```
float time_secs = glfwGetTime(); ///< C++
```

Homework 2 - Part 4*

Simple Arkanoid ® game!!



*For the passionate!
2 point (but assignment points saturate at 6/6)