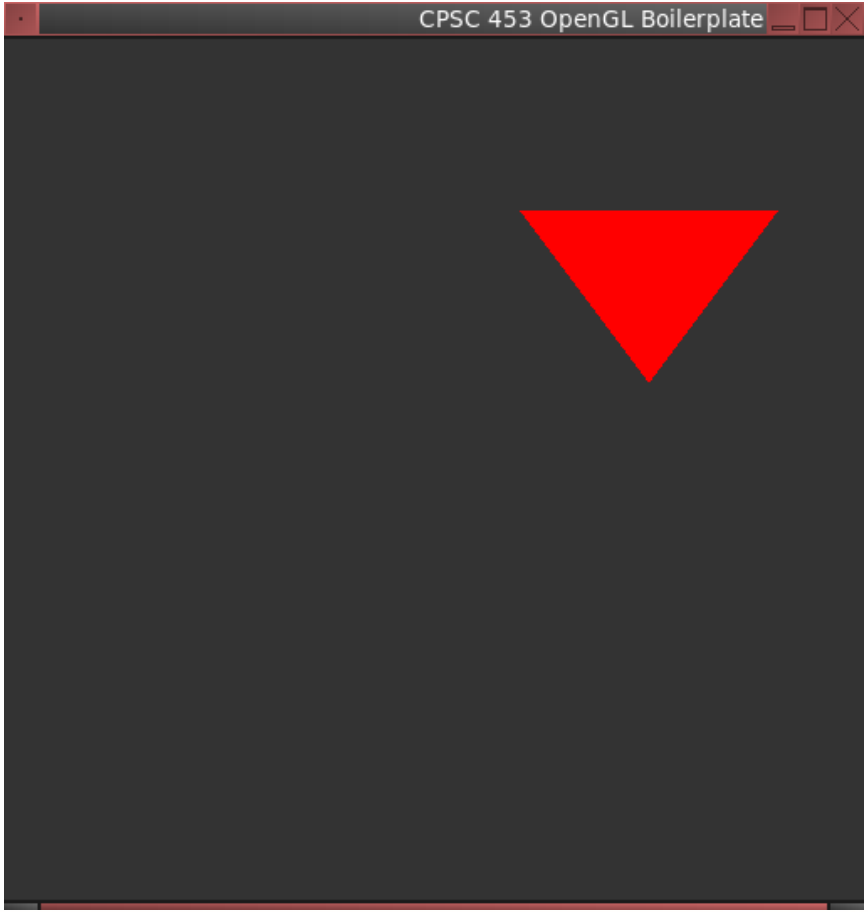
CPSC 453

GLFW and Code Walkthrough

UNIVERSITY OF CALGARY

Demo 2



UNIVERSITY OF CALGARY

Demo 2

- Includes
- Initialization
- Callbacks
- Contexts/Windows
- Shader Programs
- Geometry
- Display Loop
- Cleanup

UNIVERSITY OF CALGARY

Includes

```
#define GLFW_INCLUDE_GLCOREARB          // modern header

#define GL_GLEXT_PROTOTYPES             // add extension
                                        //  prototypes

#include <GLFW/glfw3.h>                 // GLFW
```

UNIVERSITY OF CALGARY

GLFW

UNIVERSITY OF CALGARY

GLFW

GLFW is an Open Source, multi-platform library for OpenGL, OpenGL ES and Vulkan development on the desktop. It provides a simple API for creating windows, contexts and surfaces, receiving input and events.

GLFW is written in C and has native support for Windows, macOS and many Unix-like systems using the X Window System, such as Linux and FreeBSD.

GLFW is licensed under the zlib/libpng license.

Gives you a window and OpenGL context with just two function calls

Support for OpenGL, OpenGL ES, Vulkan and related options, flags and extensions

Support for multiple windows, multiple monitors, high-DPI and gamma ramps

Support for keyboard, mouse, gamepad, time and window event input, via polling or callbacks

http://www.glfw.org/

Initialization

```
glfwInit();             // must be invoked before GL calls
```

UNIVERSITY OF CALGARY

Initialization

```
glfwInit();             // must be invoked before GL calls
```

- OpenGL library doesn’t provide functions, but **pointers to functions**
- Tremendous flexibility
- Nightmare to handle manually
- Ideally, convert back to functions

https://www.khronos.org/opengl/wiki/Load_OpenGL_Functions

UNIVERSITY OF CALGARY

Callbacks

```
glfwSetErrorCallback([](int error, const char*
description){
    cout << "GLFW ERROR " << error << ":" << endl;
    cout << description << endl;
  });      // WTF??
```

Callbacks

```
glfwSetErrorCallback([](int error, const char*
description){
    cout << "GLFW ERROR " << error << ":" << endl;
    cout << description << endl;
  });       // WTF??
```

```
#include <stdio.h>

void myError(int i) {
        printf("ERROR: %d\n", i);
        }

void doSomething( void(*callOnError)(int) ) {
        int error = 1;
        if ( error > 0 )
                callOnError( error );
        }

int main( int argc, char** argv ) {

        doSomething( myError );
        return 0;
        }
```

Callbacks

```
glfwSetErrorCallback([](int error, const char*
description){
    cout << "GLFW ERROR " << error << ":" << endl;
    cout << description << endl;
  });       // WTF??
```

```
Point a, b;                 // C++ you write

C = a.distance( b );
```

```
struct __Point a, b;        // approximate generated C code

__Point_distance_CXX324( b, &a );
```

Callbacks

```
glfwSetErrorCallback([](int error, const char*
description){
    cout << "GLFW ERROR " << error << ":" << endl;
    cout << description << endl;
  });       // WTF??
```

```
// Solution 1: Static functions

static void Point::myError(int i) {
     cout << “ERROR: “ << i << endl;
     }

doSomething( Point::myError );
```

Callbacks

```
glfwSetErrorCallback([](int error, const char*
description){
    cout << "GLFW ERROR " << error << ":" << endl;
    cout << description << endl;
  });       // WTF??
```

```
// Solution 2: std::function, std::placeholders, and std::bind
void Class::doSomething( std::function<void(int)> callOnError ) {

     // ...
     if ( error > 0 )
          callOnError( error );
     // ...
     }

void Point::myError( int i ) { cout << “ERROR: “ << i << endl; }

Point a; Class c;
c.doSomething( std::bind( &Point::myError, &a, std::placeholders::_1 ) );
```

Callbacks

```
glfwSetErrorCallback([](int error, const char*
description){
    cout << "GLFW ERROR " << error << ":" << endl;
    cout << description << endl;
  });      // WTF??
```

```
// Solution 3: lambda functions

doSomething( [](int i){

     cout << “ERROR: “ << i << endl;

     } );
```

Lambda Functions

Lambda expressions (since C++11)

Constructs a closure: an unnamed function object capable of capturing variables in scope.

Syntax

[*captures*] <*tparams*>(optional)(c++20) (*params*) *specifiers*(optional) *exception attr* -> *ret* { *body* }	(1)
[*captures*] (*params*) -> *ret* { *body* }	(2)
[*captures*] (*params*) { *body* }	(3)
[*captures*] { *body* }	(4)

1) Full declaration.

2) Declaration of a const lambda: the objects captured by copy cannot be modified.

3) Omitted trailing-return-type: the return type of the closure's `operator()` is determined according to the following rules:

- if the *body* consists of nothing but a single `return` statement with an expression, the return type is the type of the returned expression (after lvalue-to-rvalue, array-to-pointer, or function-to-pointer implicit conversion); (until C++14)
- otherwise, the return type is `void`.

The return type is deduced from `return` statements as if for a function whose return type is declared auto. (since C++14)

4) Omitted parameter list: function takes no arguments, as if the parameter list was `()`. This form can only be used if none of constexpr, mutable, exception specification, attributes, or trailing return type is used.

Lambda Functions

```
#include <iostream>

auto make_function(int& x) {
  return [&]{ std::cout << x << '\n'; };
}

int main() {
  int i = 3;
  auto f = make_function(i); // the use of x in f binds directly to i
  i = 5;
  f(); // OK; prints 5
}
```

http://en.cppreference.com/w/cpp/language/lambda

UNIVERSITY OF CALGARY

Callbacks

```
glfwSetErrorCallback(

    []  (int error, const char* description)  {

        cout << "GLFW ERROR " << error << ":" << endl;
        cout << description << endl;
        }

);
```

Key input

If you wish to be notified when a physical key is pressed or released or when it repeats, set a key callback.

```
glfwSetKeyCallback(window, key_callback);
```

The callback function receives the **keyboard key**, platform-specific scancode, key action and **modifier bits**.

```
void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_E && action == GLFW_PRESS)
        activate_airship();
}
```

The action is one of `GLFW_PRESS`, `GLFW_REPEAT` or `GLFW_RELEASE`. The key will be `GLFW_KEY_UNKNOWN` if GLFW lacks a key token for it, for example *E-mail* and *Play* keys.

The scancode is unique for every key, regardless of whether it has a key token. Scancodes are platform-specific but consistent over time, so keys will have different scancodes depending on the platform but they are safe to save to disk.

http://www.glfw.org/docs/latest/input_guide.html#input_keyboard

Windowing

```
gLFWwindow *window = 0;

glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

window = glfwCreateWindow(512, 512,
    "CPSC 453 OpenGL Boilerplate", 0, 0);
```

http://www.glfw.org/docs/latest/window_guide.html

UNIVERSITY OF CALGARY

Context ~~Windowing~~

```
gLFWwindow *window = 0;

glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

window = glfwCreateWindow(512, 512,
    "CPSC 453 OpenGL Boilerplate", 0, 0);
```

http://www.glfw.org/docs/latest/window_guide.html

UNIVERSITY OF CALGARY

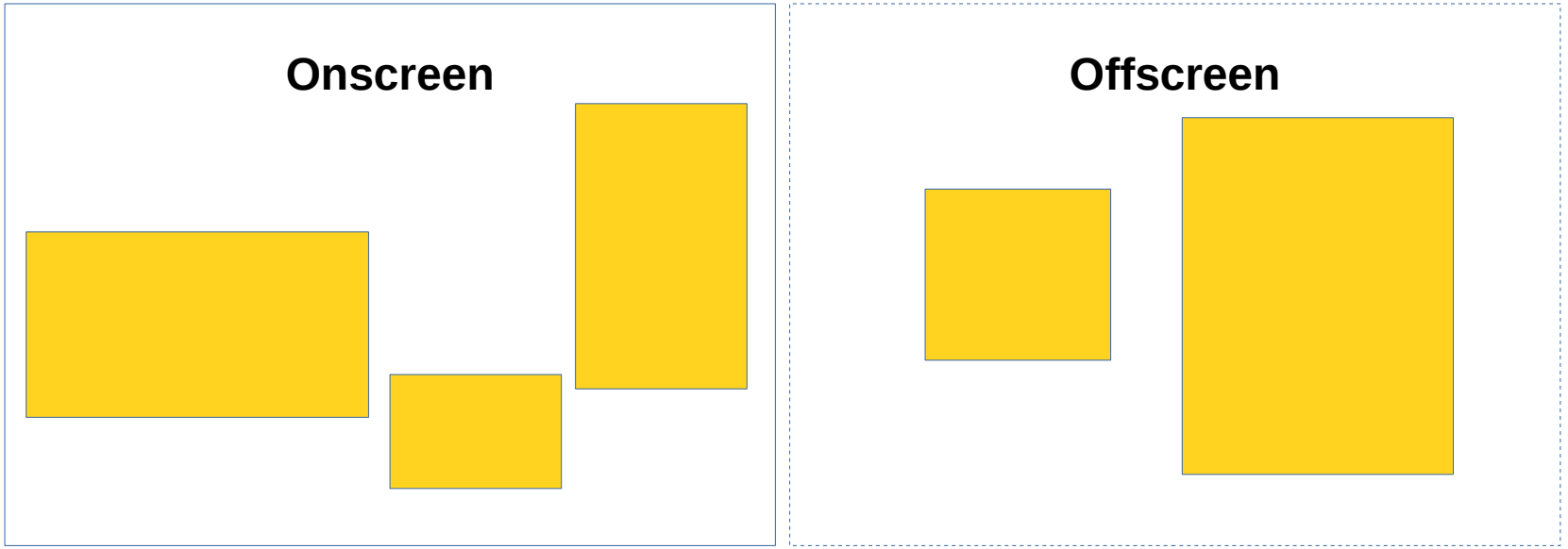
Context

```
glfwMakeContextCurrent(window);
```

http://www.glfw.org/docs/latest/context_guide.html

Context

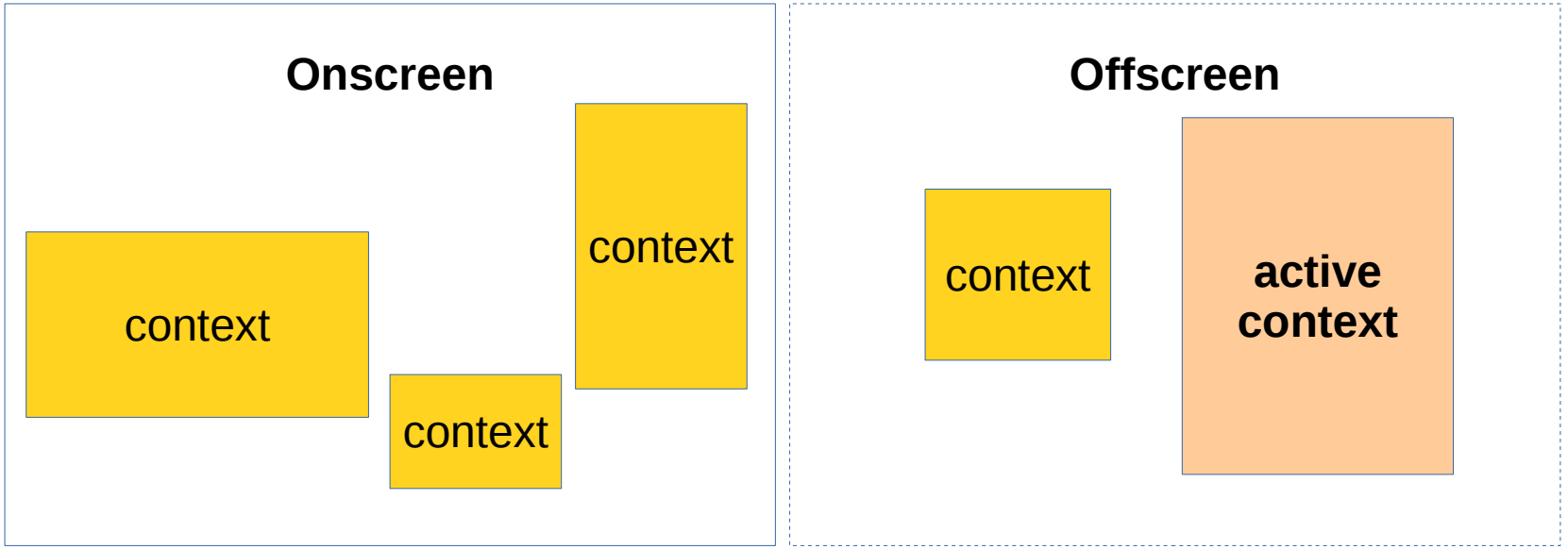
```
glfwMakeContextCurrent(window);
```



http://www.glfw.org/docs/latest/context_guide.html

Context

```
glfwMakeContextCurrent(window);
```



http://www.glfw.org/docs/latest/context_guide.html

Shader Programs

```
void init(string vertex_path, string fragment_path) {

     // Programs = grouping of shaders
     id=glCreateProgram();

     // Give the shader programming code to the graphics driver
     vertex_shader       = addShader(vertex_path,GL_VERTEX_SHADER);
     fragment_shader     = addShader(fragment_path,GL_FRAGMENT_SHADER);

     // If the shader code compiles, associate it with a program
     if(vertex_shader) glAttachShader(id,vertex_shader);
     if(fragment_shader) glAttachShader(id,fragment_shader);

     // Finally, glue the program code together
     glLinkProgram(id);
     }
```

Shader Programs

```
// Give the shader programming code to the graphics driver
GLuint addShader(string path, GLuint type) {

    /* ... read a text file into “buffer_array” ... */

    // Create a new shader of the given type
    GLuint shader = glCreateShader(type);

    // Compile the shader program
    glShaderSource(shader, 1, buffer_array, 0);
    glCompileShader(shader);

    // Check the results
    GLint status;
    glGetShaderiv(shader, GL_COMPILE_STATUS, &status);
    if (status == GL_FALSE) { /* ... ERROR!!! ... */ }

    // Give back the ID associated with the shader
    return shader;
    }
```

Shader Programs

```
// Vertex shader
#version 410

layout(location = 0) in vec2 position; // Define the incoming vertex position

void main() {                          // ... then just copy it over :'(
  gl_Position = vec4(position, 0.0, 1.0);
}
```

Shader Programs

```
// Fragment shader
#version 410

out vec4 FragmentColour;      // Define our outgoing colour

void main() {                 // ... then just write red
  FragmentColour = vec4(1,0,0,1);
}
```

Geometry

```
// Create a vertex buffer object for OpenGL
VertexArray va(3);
va.addBuffer("v", 0,
    vector<float> { 0.5, 0.2, 0.2, 0.6, 0.8, 0.6 });
```

Geometry

```
// Create a vertex buffer object for OpenGL
VertexArray va(3);
va.addBuffer("v", 0,
    vector<float> { 0.5, 0.2, 0.2, 0.6, 0.8, 0.6 });

/*
class VertexArray {
    std::map<string, GLuint> buffers;
    std::map<string, int> indices;
public:
    GLuint id;
    unsigned int count;

    VertexArray(int c) {
        glGenVertexArrays(1, &id);    // generate the array
        count = c;
    }

    … */
```

Geometry

```
// ACTUALLY create a vertex buffer object
void addBuffer(string name, int index, vector<float> buffer) {
      GLuint buffer_id;
      glBindVertexArray(id);        // tell OpenGL which VA we're using

      glGenBuffers(1, &buffer_id);  // and which buffer associated with the VA
      glBindBuffer(GL_ARRAY_BUFFER, buffer_id);
      glBufferData(GL_ARRAY_BUFFER, buffer.size() * sizeof(float),
                  buffer.data(), GL_STATIC_DRAW);          // transfer over static data

      buffers[name] = buffer_id;          // store the Ids locally
      indices[name] = index;

      int components = buffer.size() / count;   // tell OGL how it's formatted
      glVertexAttribPointer(index, components, GL_FLOAT, GL_FALSE, 0, 0);
      glEnableVertexAttribArray(index);

      // unset states
      glBindBuffer(GL_ARRAY_BUFFER, 0);
      glBindVertexArray(0);
      }
```

Display Loop

```
while (!glfwWindowShouldClose(window)) {

        render(p, va);          // the only non-trivial routine

        glfwSwapBuffers(window);

        glfwPollEvents();
    }
```

Display Loop

```
void render(Program &program, VertexArray &va) {

    glClearColor(0.2f, 0.2f, 0.2f, 1.0f);    // clear buffer
    glClear(GL_COLOR_BUFFER_BIT);

    glUseProgram(program.id);                // bind program
    glBindVertexArray(va.id);                // and V.A.
    glDrawArrays(GL_TRIANGLES, 0, va.count);

    glBindVertexArray(0);                    // unbind, in reverse
order
    glUseProgram(0);

}
```

UNIVERSITY OF CALGARY

Cleanup

```
glfwDestroyWindow(window);
glfwTerminate();
```

Cleanup

```
// void glDeleteBuffers( GLsizei n,  const GLuint * buffers );

glfwDestroyWindow(window);
glfwTerminate();
```

UNIVERSITY OF CALGARY

GLM

UNIVERSITY OF CALGARY

OpenGL Mathematics (GLM) is a C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specification.

GLM provides classes and functions designed and implemented with the same naming conventions and functionalities than GLSL so that when a programmer knows GLSL, he knows GLM as well which makes it really easy to use.

https://glm.g-truc.net/0.9.2/api/index.html