Creative Coding II

03 openFrameworks

FILMUNIVERSITÄT
BABELSBERG
KONRAD WOLF

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

openFrameworks

- openFrameworks is free, open source C++ framework that has been strongly influenced by the Processing environment

- openFrameworks targets easy development of real-time applications & is primarily oriented for use in creative and experimental projects

- openFrameworks takes care of creating a graphical window, listening for mouse and keyboard events, etc., so developers can start with expressing their ideas fairly quickly

Overview

- openFrameworks is highly extensible using **addons**
  - **ofxAddon** are open source & generally built by members of the openFrameworks community
  - However, many addons are not maintained on a regular basis

- openFrameworks is **cross-platform** (supporting OS X, Windows, Linux, iOS, Android & Linux ARM devices such as Raspberry Pi)

- Finally, it has a very friendly & active community

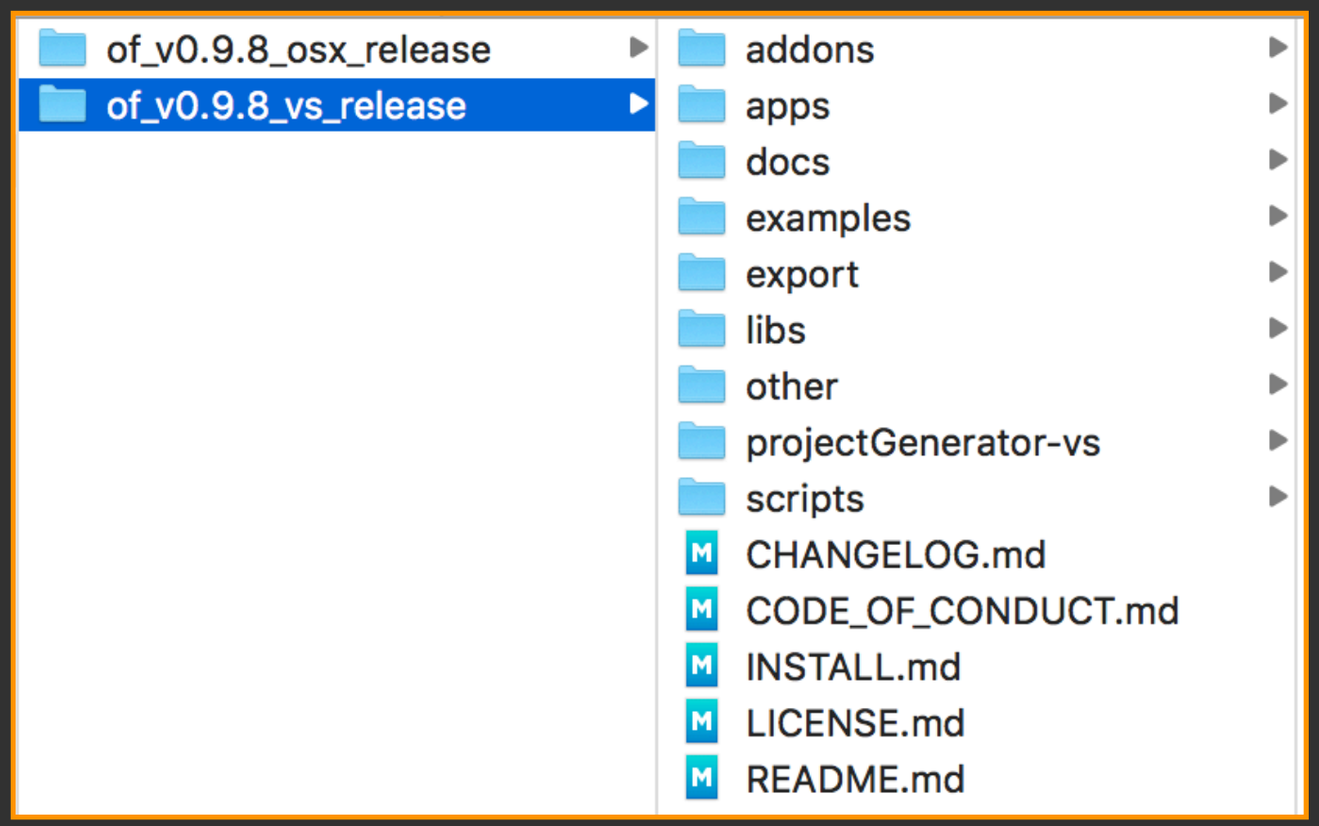
Installation

- Go to the http://openframeworks.cc/download/ page
- Download the latest public release (v0.9.8) for your target OS
- You will download a *.zip file
- Choose a folder on your OS where you want to develop stuff
- Unzip the folder
- **That’s it!**

  Everything else is what you will do now with your IDE

Folder Structure

- When you open the folder, you will find the following subfolders
- Note: ‘osx’ obviously refers to MacOS as target platform whereas ‘vs’ refers to ‘Visual Studio’ & thus Windows as target platform



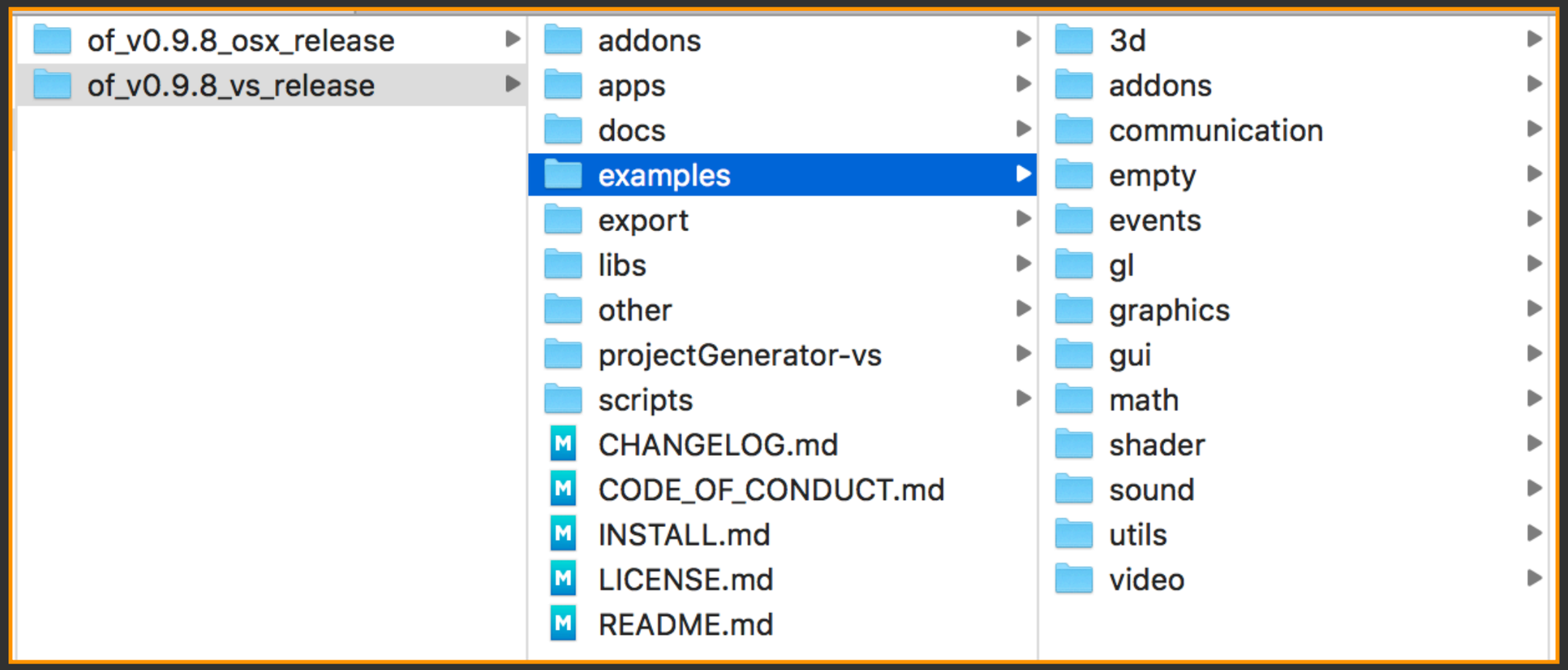
Folder Structure

- Checkout the README file for more info, this is an excerpt:

> `docs` has some documentation around OF usage, per platform things to consider, etc. You should definitely take a look in there; for example, if you are on OSX, read the osx.md. `apps` and `examples` are where projects go -- `examples` contains a variety of projects that show you how to use OF, and `apps` is where your own projects will go. `libs` contains the libraries that OF uses, including the openframeworks core itself. `addons` are for additional functionality that's not part of the core. `export` is for DLLs and dylibs that need to be put in each compiled project. The `scripts` folder has the templates and small scripts for automating OF per platform. `project generator` is a GUI based tool for making new projects - this folder is only there in packaged releases.

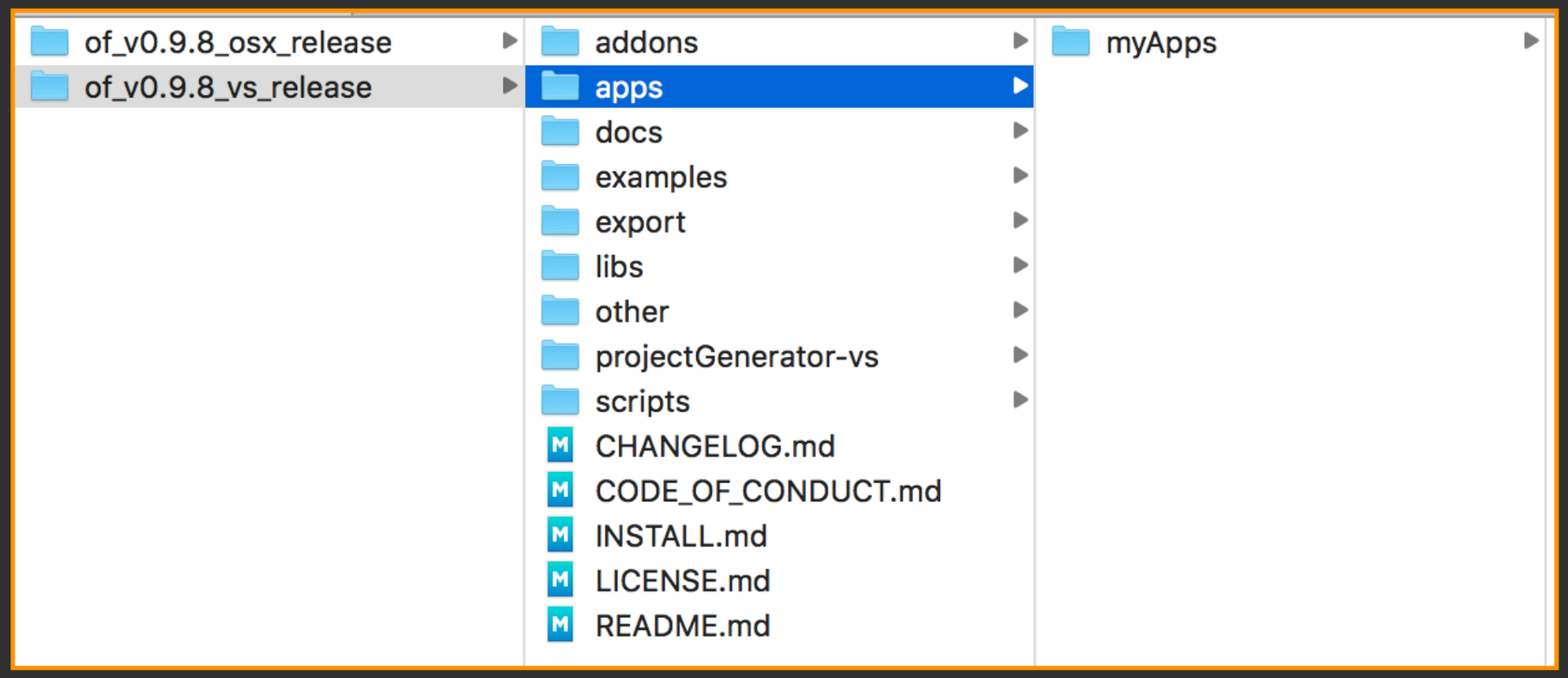
Folder Structure

- The '**examples**' folder contains oF examples to be built
- The list of subfolder gives an overview of the capabilities of oF



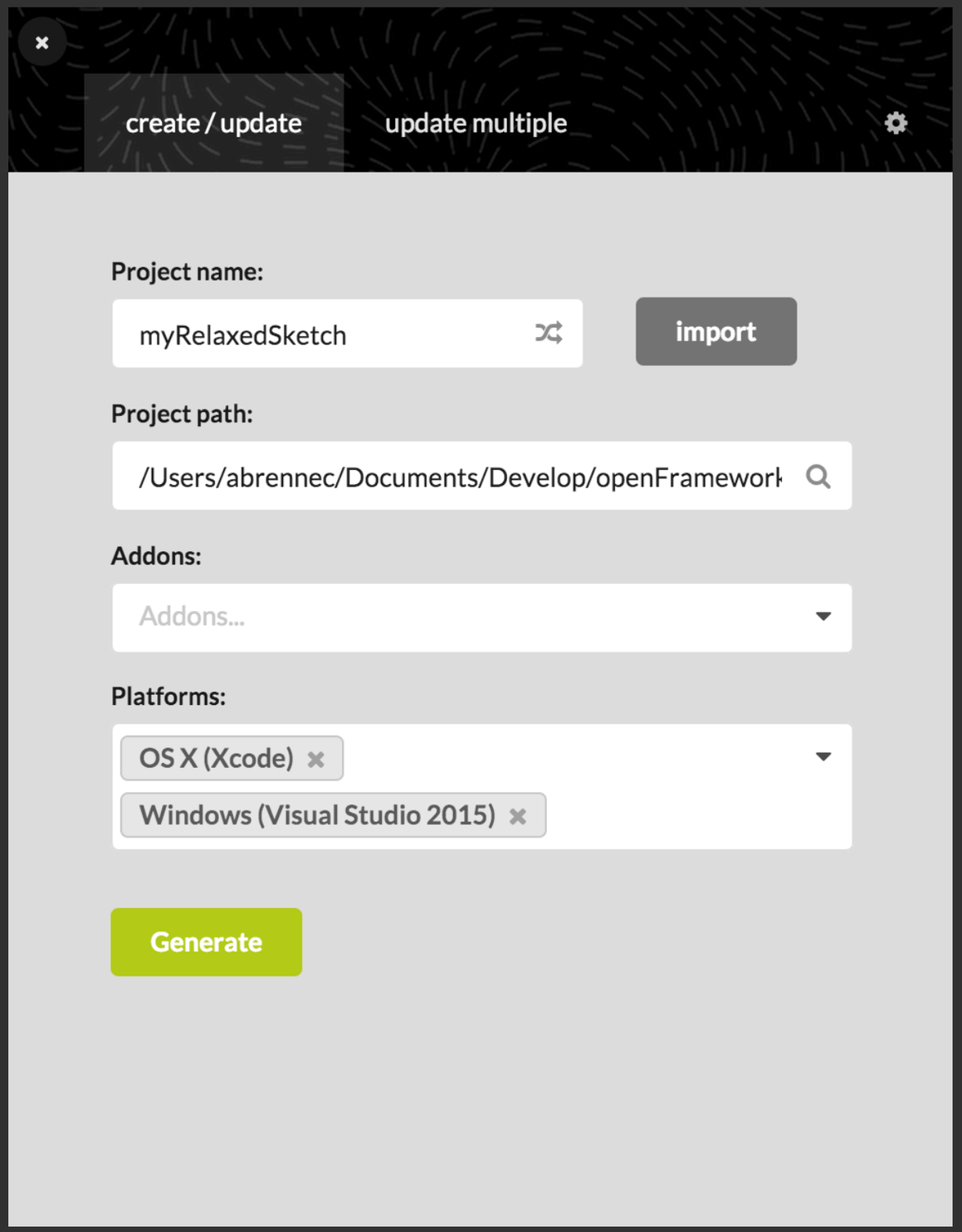
Folder Structure

- The '**apps**' folder is the folder where all of your example projects and applications will be stored
- If you use a different folder to store your apps, use the ProjectGenerator to create a specific project file that brings the correct paths to OF



Project Generator

- The Project Generator
  allows you to create your own empty
  - projectGenerator.app
  - projectGenerator.exe
  -
- Alternatively, use
  - openFrameworks plugin
    in Visual Studio
  - copy an apps/myApps folder



openFrameworks

- openFrameworks follows a certain pattern to create, update & draw applications that is reflected by three functions
  - setup ( )
  - update ( )
  - draw ( )

- The three functions are part of the ofApp class & thus appear in every (user-defined) ofApp.h & ofApp.cpp file

The setup( ) Function

ofApp.h

```
ofVideoPlayer player;
int counter;
```

ofApp.cpp

```
void ofApp::setup(){
    player.loadMovie("movie.mov");
    counter = 0;
}
```

Image credit: http://openframeworks.cc/ofBook/chapters/how_of_works.html

- The setup( ) function is called only **once** at program startup
- It is used to create the windowing context — the application window — and to initialize settings and variables

The update( ) & draw( ) Functions

- The update( ) & draw( ) functions are called regularly at a certain framerate and in that order during the lifetime of the application

- The update( ) function is used to update the application state, variables, perform or trigger calculations, etc.

- The draw( ) function is used to regularly draw the application and anything that we want to visualize onto the screen

The update( ) & draw( ) Functions

ofApp.h

```
float x;
```

ofApp.cpp

```
void ofApp::setup(){
    x = 0;
}

void ofApp::update(){
    x++;
}

void ofApp::draw(){
    ofDrawCircle(x,120,30);
}
```

- `setup()`
- `update()`
- `draw()`
- `update()`
- `draw()`
- ...

Image credit: http://openframeworks.cc/ofBook/chapters/animation.html

Image credit: http://openframeworks.cc/ofBook/chapters/how_of_works.html

Organization of the SDK

- The openFrameworks SDK is organized in classes to abstract & define the provided functionality & data types

- All openFrameworks classes follow a naming convention
  - “ofClassName” — core classes use “of” as prefix
  - “ofxAddonName” — Addon classes use “ofx” as prefix

- Classes are basically divided into utilities, data containers, and basic data types

Utility Classes

- Utility classes refer to classes that provide certain functionality to render, display & interact with audio visual data
- Examples of utility classes are
  - ofVideoGrabber
  - ofImage
  - ofSoundPlayer
  - …

ofApp.h

```
ofVideoPlayer player;
```

ofApp.cpp

```
void ofApp::update(){
    ofVideoPlayer player2 = player;
    player2.setFrame(100);
}
```

Image credit: http://openframeworks.cc/ofBook/chapters/how_of_works.html

Container Classes

- Container classes, on the other hand, contain data & provide functionality to manipulate these data

- They split into data containers & GL data containers

- The main difference between data containers & GL data containers is that the latter do not support data copying due to performance reasons

(GL) Data Container Classes

ofApp.h

```
ofPixels pixels1, pixels2;
ofTexture tex1, tex2;
```

ofApp.cpp

```
void ofApp::setup(){
    pixels1.allocate(640,480,OF_IMAGE_COLOR);
    pixels1.set(0);
    pixels2 = pixels1;
    pixels2.setColor(10,10,ofColor(255,255,255));

    tex1.allocate(640,480,GL_RGB);
    tex2.allocate(640,480,GL_RGB);
    tex1.loadData(pixels1);
    tex2.loadData(pixels2);
}

void ofApp::draw(){
    tex1.draw(0,0);
    tex2.draw(660,0);
}
```

Image credit: http://openframeworks.cc/ofBook/chapters/how_of_works.html

Basic Data Types

- Finally, openFrameworks also provides certain classes that specify basic data types required for 2D & 3D rendering like
  - ofRectangle
  - ofCircle,
  - ofVec3f
  - ofMatrix4x4
  - …

The Default main Function

```
#include "ofMain.h"
#include "ofApp.h"

//========================================================================
int main( )
{
    ofSetupOpenGL(1024,768,OF_WINDOW);          // <-------- setup the GL context

    // this kicks off the running of my app
    // can be OF_WINDOW or OF_FULLSCREEN
    // pass in width and height too:
    ofRunApp(new ofApp());

}
```

- The main.cpp file of a default openFrameworks app first sets up the drawing context, i.e., the OpenGL & window settings required for the graphics library and runs the application

The OpenGL Context Simplified

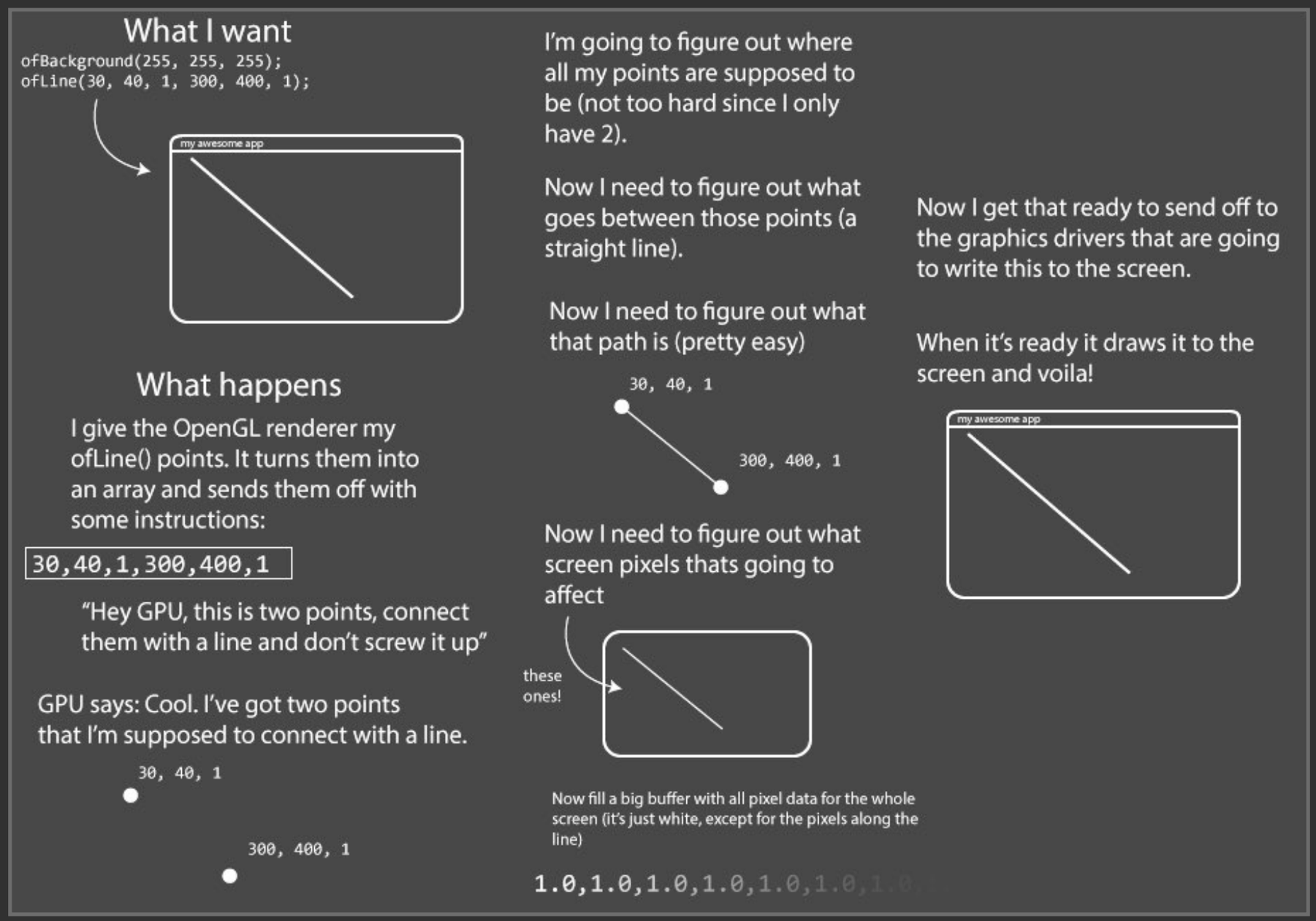


Image credit: http://openframeworks.cc/ofBook/chapters/openGL.html

OpenGL — An API for Graphics

- OpenGL is a cross-platform standardized **API** for rendering real-time computer graphics

- The actual **implementation** comes with the operating systems in the form of a library (opengl32.dll, libGL.so) & graphics card / hardware-specific device drivers

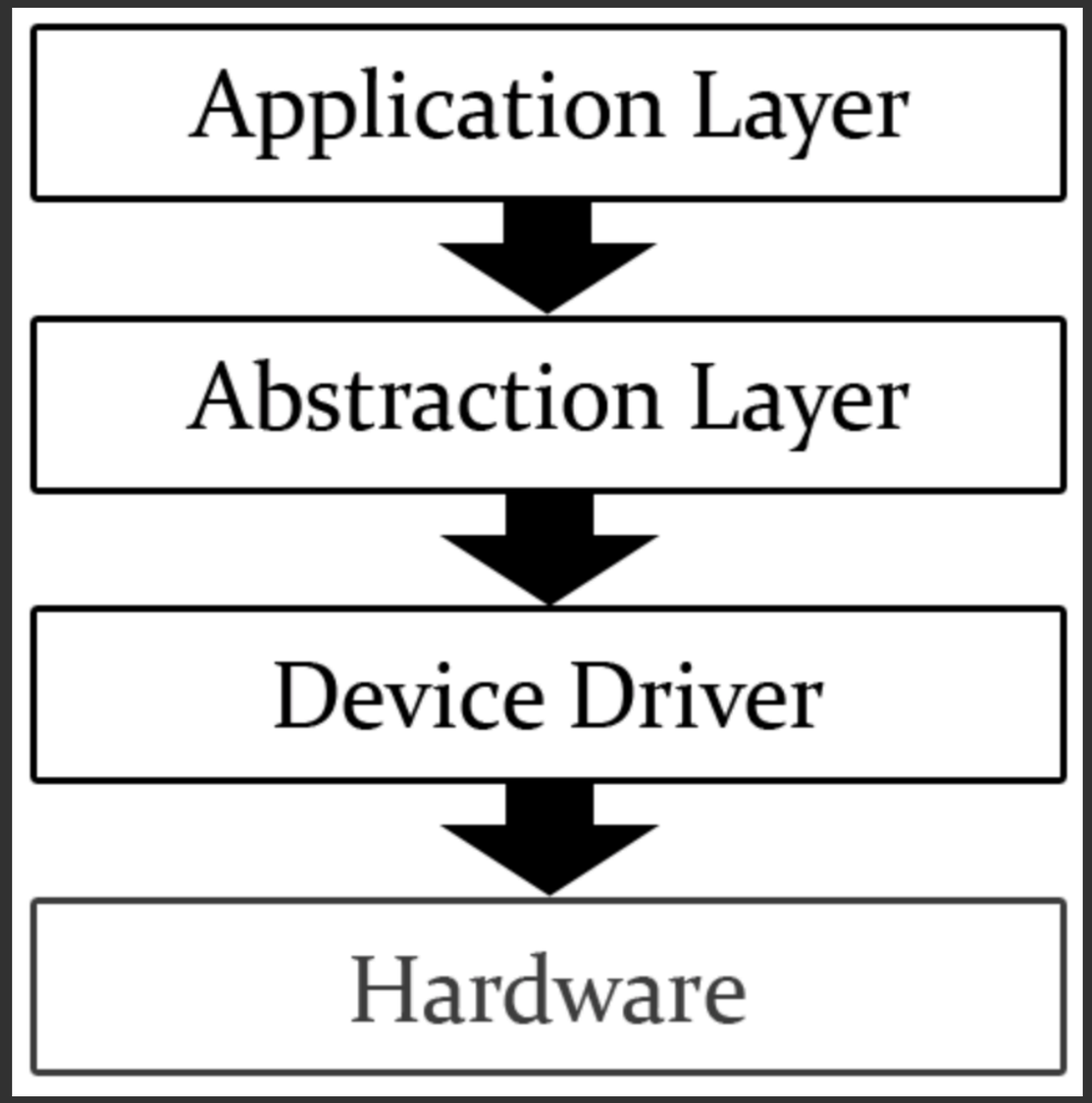


Image credit: http://openglbook.com/chapter-0-preface-what-is-opengl.html

OpenGL — An API for Graphics

- **The application layer** represents your program; here you can use the function declarations of the API

- **The abstraction layer** represents the OpenGL library on the operating system you are working on; a thin layer that passes the API calls on to the device driver layer

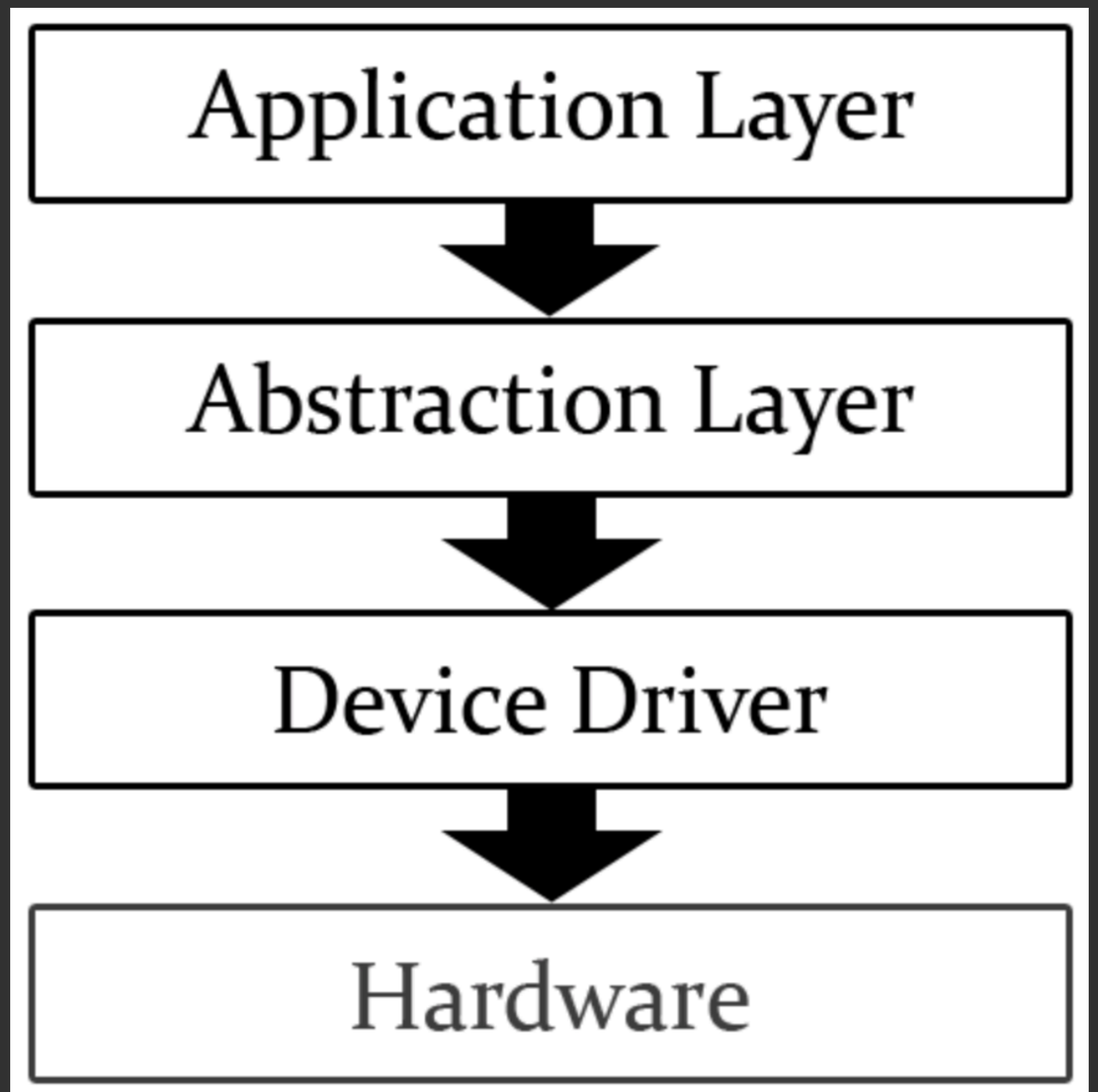


Image credit: http://openglbook.com/chapter-0-preface-what-is-opengl.html

OpenGL — An API for Graphics

- **The device driver** layer represents the hardware-specific implementation of the OpenGL API;
- The device driver directly talks to the graphics hardware

- **The hardware layer** represents the GPU (graphics processing unit), i.e., the graphics hardware that is “driven” / instructed by the driver

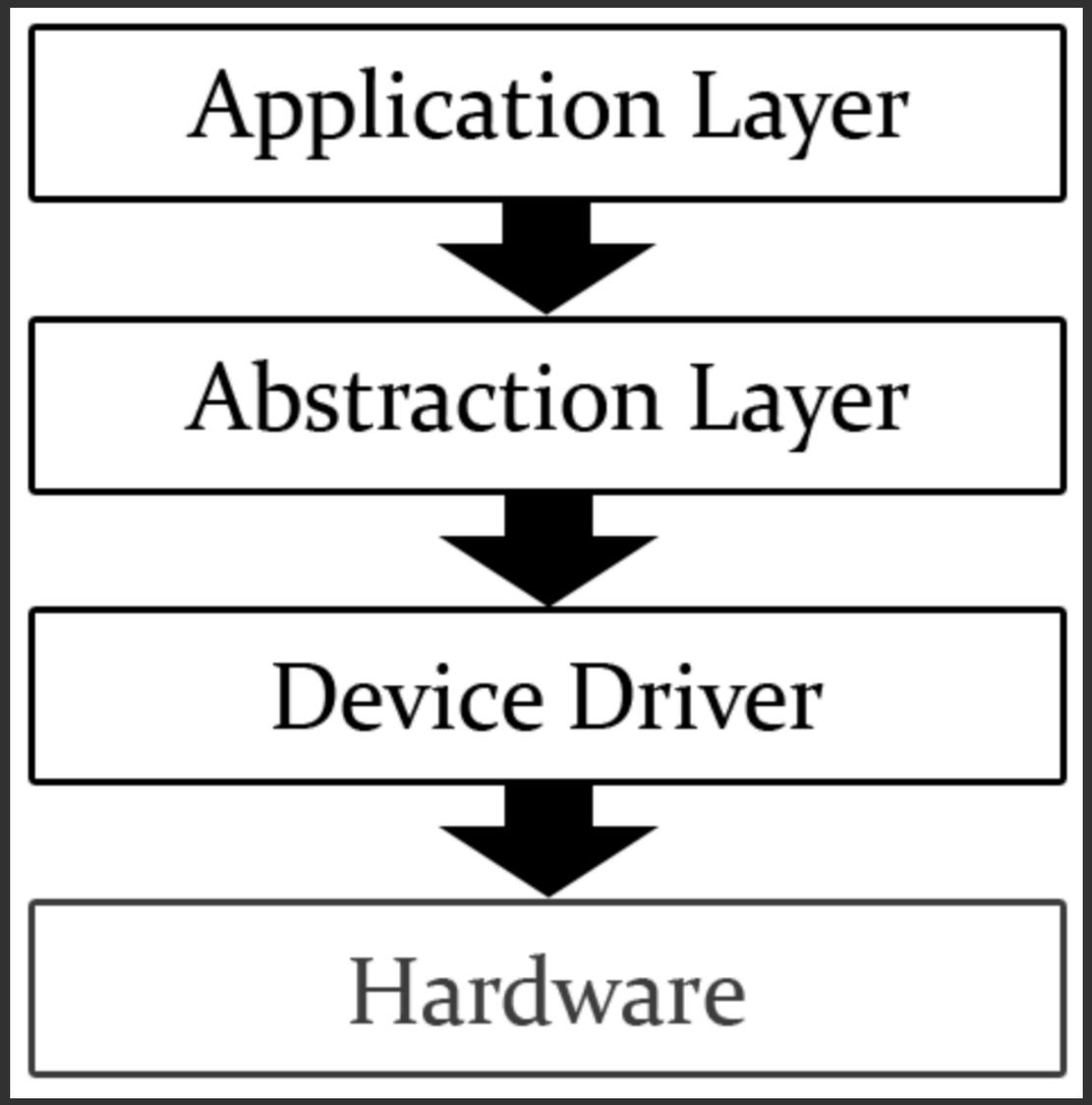


Image credit: http://openglbook.com/chapter-0-preface-what-is-opengl.html

OpenGL & the Window

- OpenGL is purely a graphics library that takes care of rendering graphics only

- To draw somewhere on the screen, a windowing context is required that tells OpenGL where to draw its graphics

- This is achieved with the help of with a windowing API

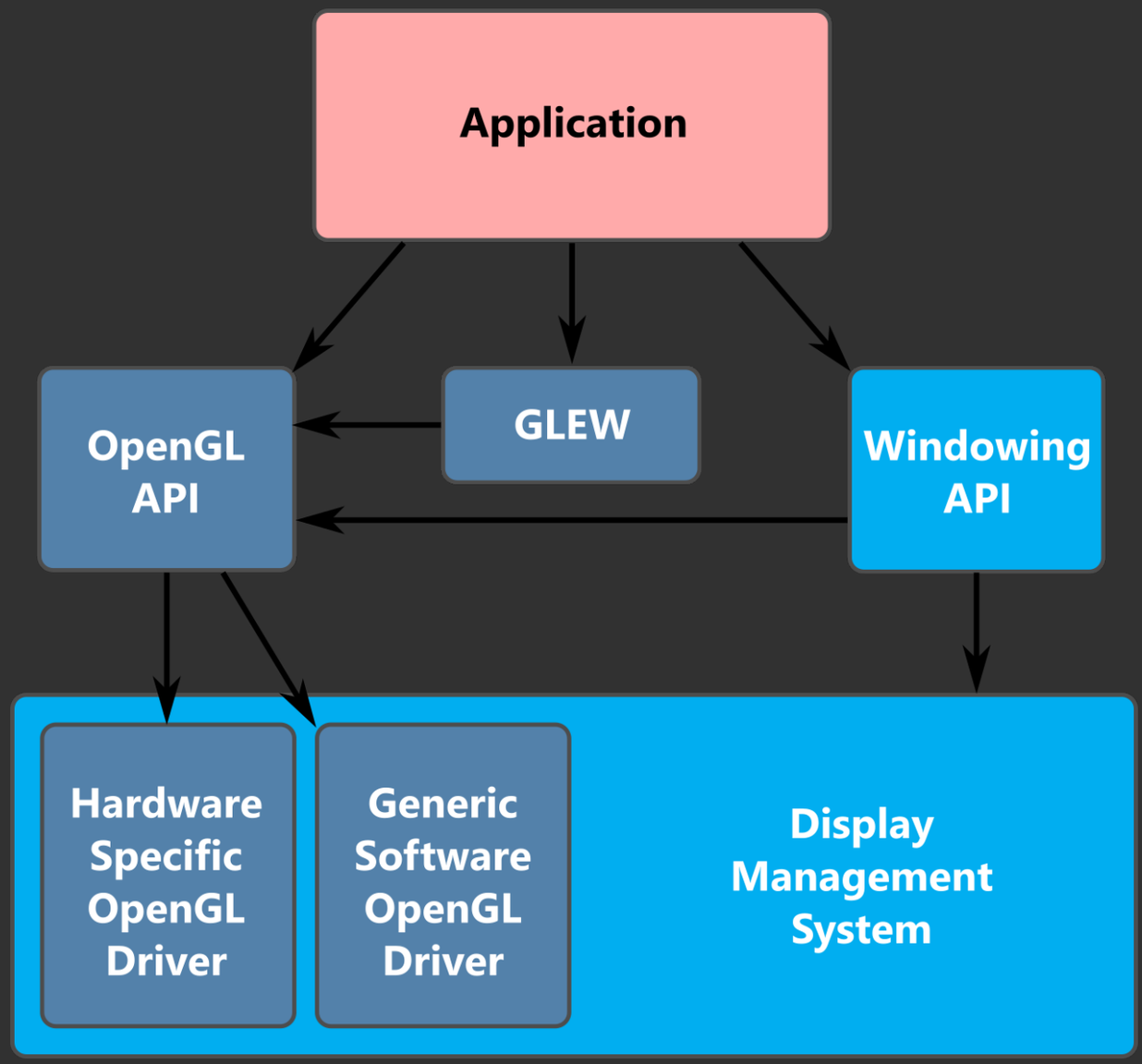


Image credit: http://www.cs.uregina.ca/Links/class-info/315/WebGL/Lab1/wip.html

OpenGL & the Window

- openFrameworks abstracts the difficult graphics & windowing function calls and provides and easy interface to setup the graphics & windowing context

- Nonetheless, all specific calls to the graphics hardware & windowing API can be used

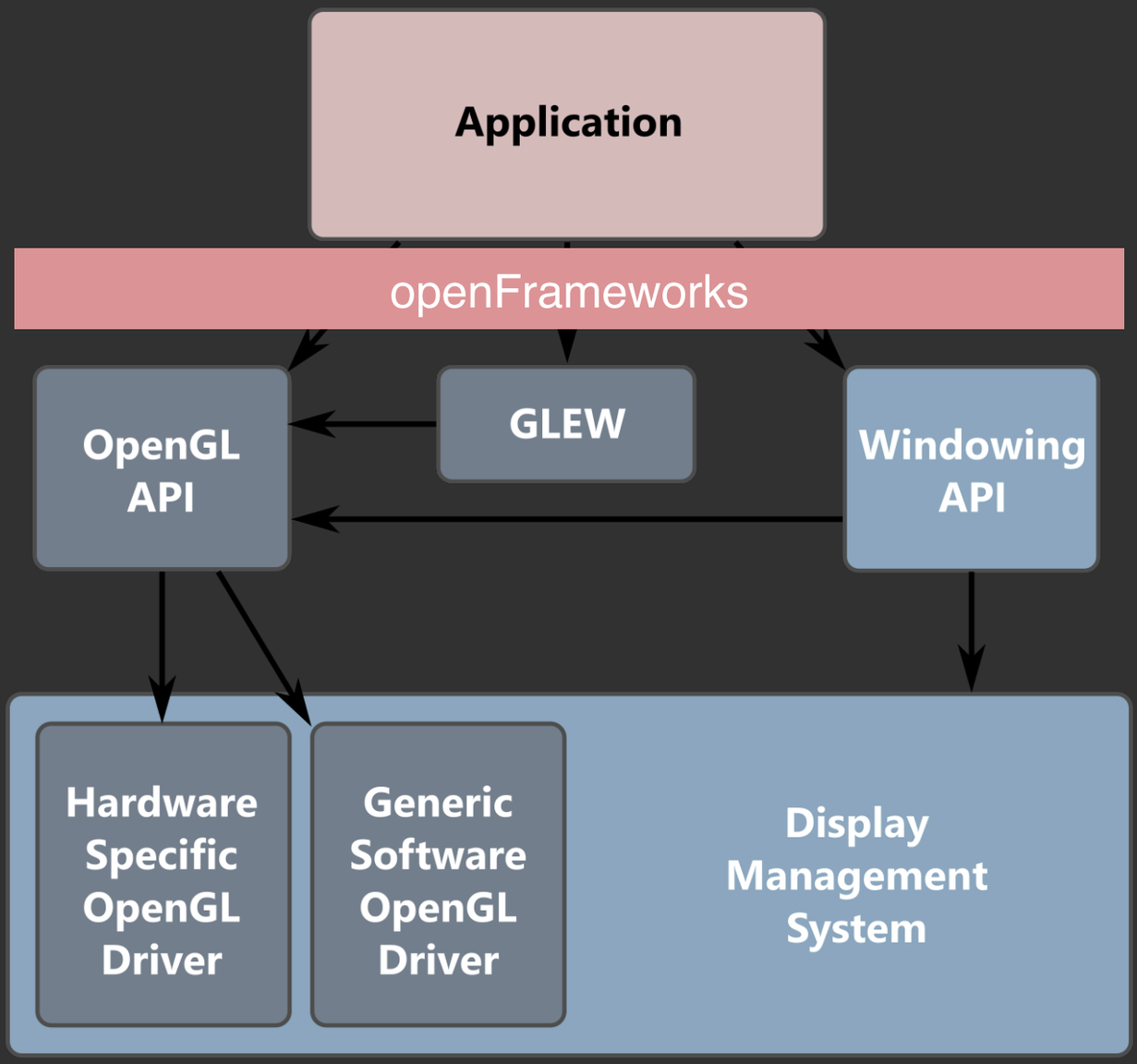


Image credit: http://www.cs.uregina.ca/Links/class-info/315/WebGL/Lab1/wip.html

Bibliography

- Last access on websites: 28 Mai 2024
- https://openframeworks.cc/