Starting a C++ project

using CMake, Catch and google/benchmark

Marc-Olivier Andrez
Senior Software Engineer at AMIA-Systems

Coding process (C++)

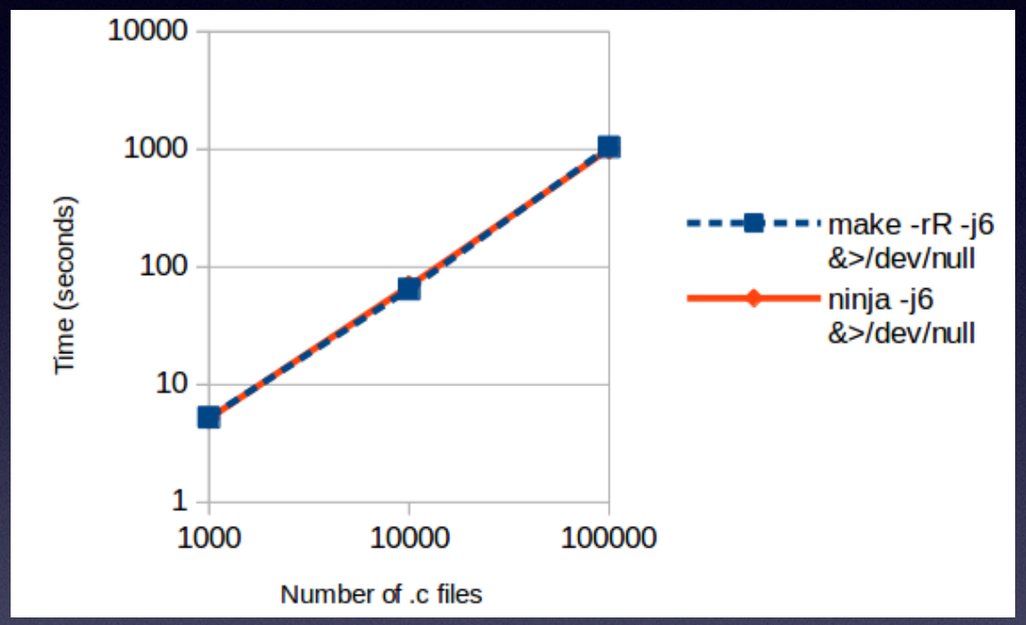
- Code
- Build
- Test
- Benchmark
- Refactor

Coding process (C++)

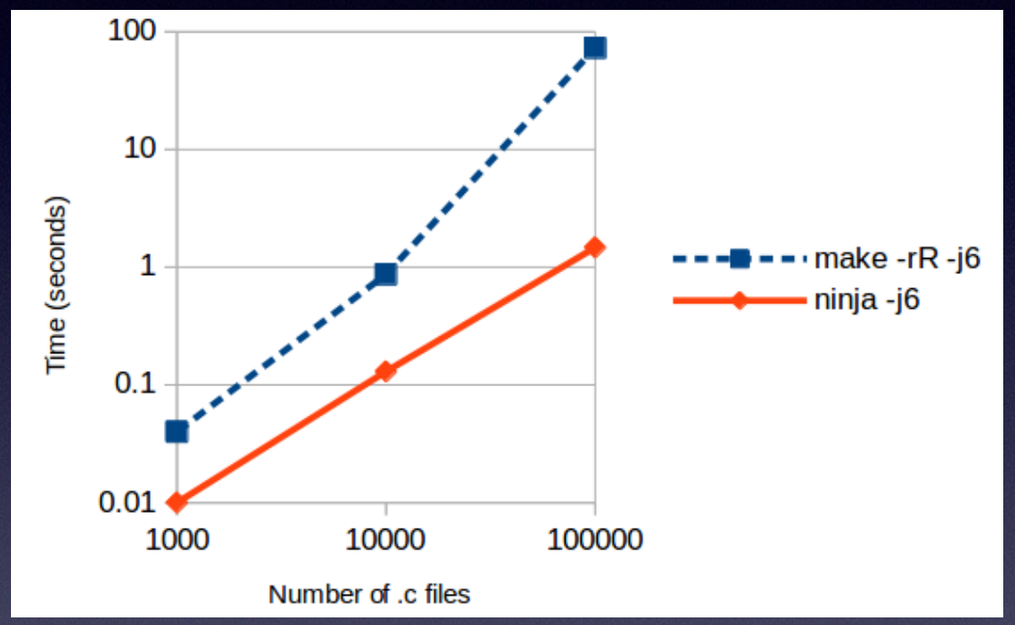
- Code
- Build (CMake + Ninja)
- Test (Catch)
- Benchmark (google/benchmark)
- Refactor

Generalities about build systems

Benchmark: Ninja vs Make



Fresh build



No op build

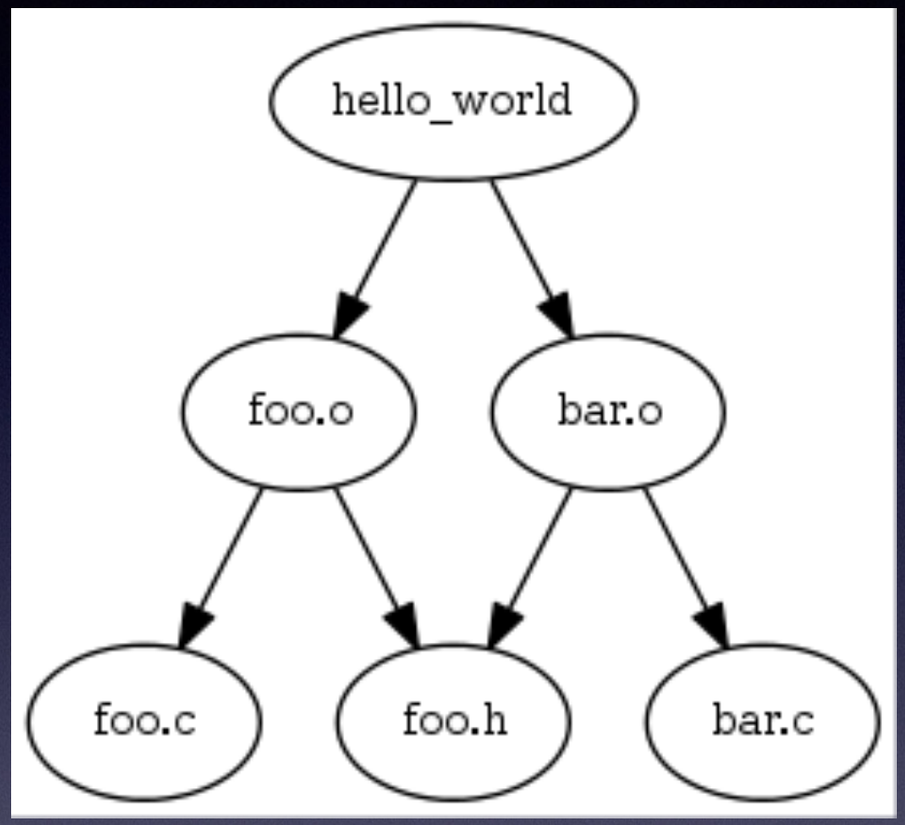
http://david.rothlis.net/ninja-benchmark/

Targets and dependencies

Targets

Dependency targets

Source files



Tup - http://gittup.org/tup/

- Incremental builds: rebuild targets only if dependencies have changed or must be rebuilt

How to collect header files?

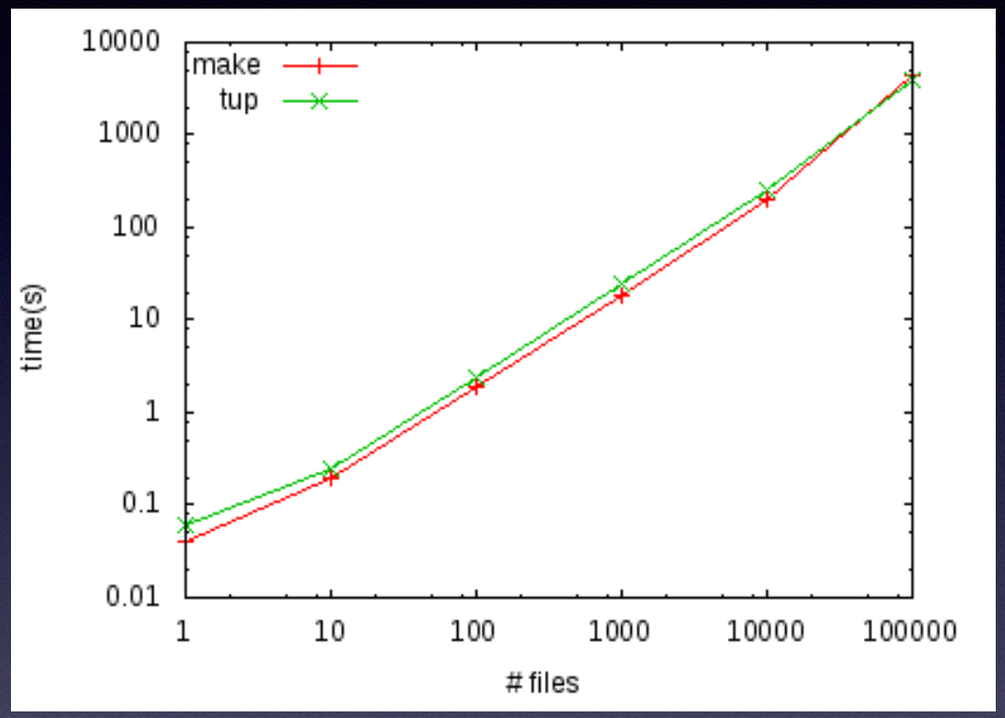
```
// foo.h

#include “bar.h”


#ifdef USE_42
  #include “42.h”
#else
  #include “49_3.h”
#endif
```

- Manually collect header files (fastidious and error prone)
- Use compiler to collect header files (takes time when compiling)

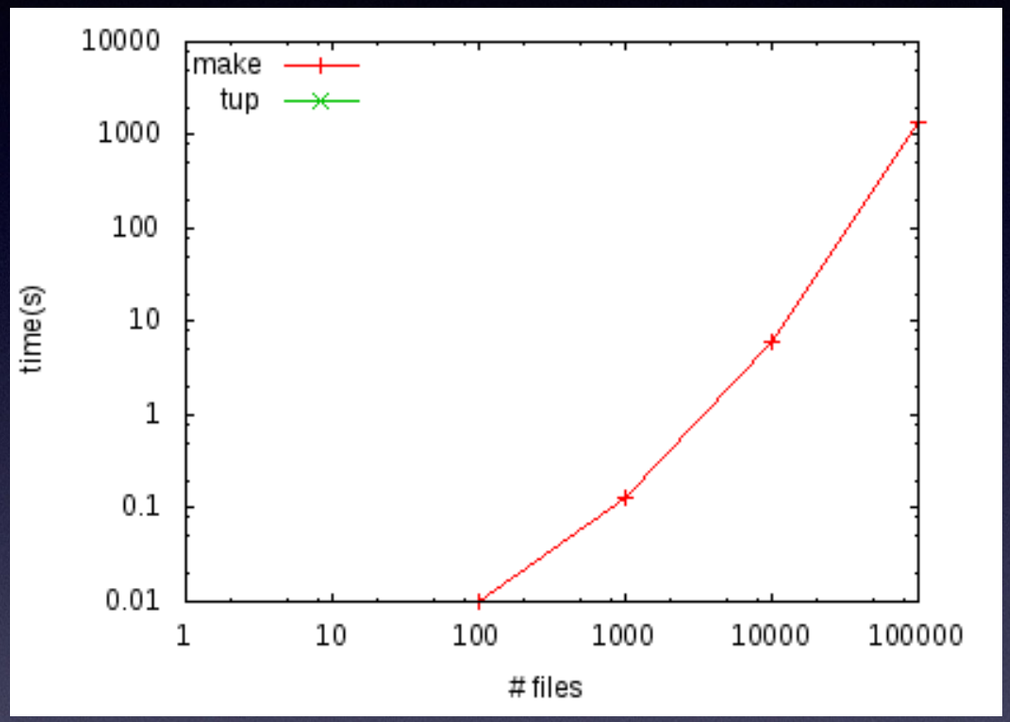
Incremental build systems

- MSBuild - default build system of Visual Studio projects
- Xcode - default build system of Xcode projects
- GNU Make - Makefiles have to define all dependencies
- Ninja (design) - similar to make but can collect header files
- Scons - write targets and dependencies in Python
- Bazel - higher level description of build targets
- tup - bottom-up calculation of targets that must be rebuilt
- Shake - similar to make, build scripts are written in Haskell

Benchmark: tup vs Make



Fresh build



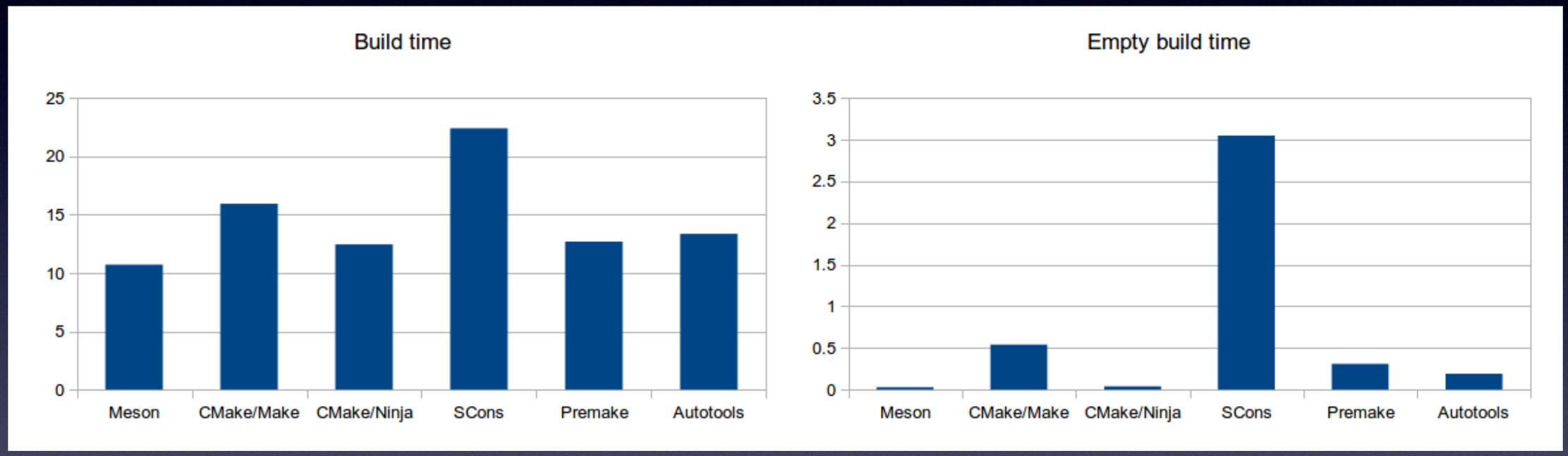
No op build

http://gittup.org/tup/make_vs_tup.html

Generators of build systems

- CMake - define high level targets and commands using its own syntax
- Meson - scripts written in Python
- Premake - scripts written in Lua
- QMake - convenient for Qt projects - no Ninja generator
  - Gradually replaced by Qbs (similar to Bazel)
- GN (Presentation) - replacing GYP (performances issues)

Benchmark: Meson vs CMake vs Premake vs SCons vs Autotools



Experiment with 1,000 simple c files

https://sourceforge.net/p/meson/wiki/Simple%20comparison/

Evaluating build systems

1. **Correctness** (Recompiles **if and only if** necessary)
2. **Speed** (for full build and incremental builds)
3. Build scripts: **concise** but **flexible**
4. **Multi-platforms** (requires only few changes)
5. **Play well with others** (IDEs, third-party libraries)

CMake tutorial

First CMakeLists.txt (1)

Index of source directory
e.g. 1 -> 01_...

```
cmake_minimum_required(VERSION 3.7)

project(hwProj LANGUAGES CXX VERSION 1.0)

add_executable(helloWorld
    main.cpp
)
```

Running cmake & ninja (1)

```
$ mkdir ../build_aux && cd ../build_aux

$ cmake -G"CodeBlocks - Ninja" \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_CXX_FLAGS="-std=c++1z" \
  ../01-helloWorld

$ ninja

$ ./helloWorld
```

Adding install target (2)

```
cmake_minimum_required(VERSION 3.7)

project(hwProj LANGUAGES CXX VERSION 1.0)

add_executable(helloWorld
    main.cpp
)

install(TARGETS helloWorld
    RUNTIME DESTINATION bin
)
```

Running cmake & ninja (2)

```
$ mkdir ../build_aux && cd ../build_aux

$ cmake -G"CodeBlocks - Ninja" \
   -DCMAKE_BUILD_TYPE=Release \
   -DCMAKE_CXX_FLAGS="-std=c++1z" \
   -DCMAKE_INSTALL_PREFIX=../install_aux \
   ../02-helloWorld-installTarget

$ ninja install

$ ../install_aux/bin/helloWorld
```

Creating library (3)

```
cmake_minimum_required(
        VERSION 3.7)
project(moplot
        LANGUAGES CXX
        VERSION 1.0)

add_library(moplot
        curve2D.cpp
        curve2D.hpp
)
add_executable(testMoplot
        testCurve2D.cpp
)
target_link_libraries(testMoplot
        PUBLIC moplot
)
```

```
#continuation
install(TARGETS moplot
        RUNTIME DESTINATION bin
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib)
install(FILES curve2D.hpp
        DESTINATION include/moplot
        COMPONENT Devel)
```

Running cmake & ninja (3)

```
$ mkdir ../build_aux && cd ../build_aux

$ cmake -G"CodeBlocks - Ninja" \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_CXX_FLAGS="-std=c++1z" \
  -DCMAKE_INSTALL_PREFIX=../install_aux \
  ../03-libMoplot

$ ninja install

$ ./testMoplot
```

Separate directories (4)

```
# ./CMakeLists.txt
cmake_minimum_required(
      VERSION 3.7)
project(moplot LANGUAGES CXX
      VERSION 1.0)
add_subdirectory(src)
add_subdirectory(tests)
```

```
# tests/CMakeLists.txt
add_executable(testMoplot
      testCurve2D.cpp
)
target_link_libraries(testMoplot
      PUBLIC moplot
)
```

```
# src/CMakeLists.txt
add_library(moplot
      curve2D.cpp
      curve2D.hpp
)
target_include_directories(moplot
      PUBLIC .
)
install(TARGETS moplot
      RUNTIME DESTINATION bin
      LIBRARY DESTINATION lib
      ARCHIVE DESTINATION lib)
install(FILES curve2D.hpp
      DESTINATION include/moplot
      COMPONENT Devel)
```

Running cmake & ninja (4)

```
$ mkdir ../build_aux && cd ../build_aux

$ cmake -G"CodeBlocks - Ninja" \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_CXX_FLAGS="-std=c++1z" \
  -DCMAKE_INSTALL_PREFIX=../install_aux \
  ../04-libMoplot-separateDirectories

$ ninja install

$ ./tests/testMoplot
```

Importing Catch Library (5)

```
# tests/CMakeLists.txt
find_file(CatchHeader catch/catch.hpp)
if(NOT CatchHeader)
    message(FATAL_ERROR "Could not find Catch header: ${CatchHeader}")
endif()
get_filename_component(CatchIncludeDir ${CatchHeader} DIRECTORY)
message(STATUS "Catch found in ${CatchIncludeDir}")
add_library(catch INTERFACE IMPORTED GLOBAL)
set_property(TARGET catch PROPERTY
    INTERFACE_INCLUDE_DIRECTORIES ${CatchIncludeDir})

add_executable(testMoplot
    testCurve2D.cpp
)
target_link_libraries(testMoplot
    PUBLIC moplot catch
)
add_test(NAME testMoplot COMMAND testMoplot ~[Skip])
```

Enabling ctest (5)

```
# ./CMakeLists.txt
cmake_minimum_required(VERSION 3.7)
project(moplot LANGUAGES CXX VERSION 1.0)
enable_testing()
add_subdirectory(src)
add_subdirectory(tests)
```

Running cmake & ninja (5)

```
$ mkdir ../build_aux && cd ../build_aux

$ cmake -G"CodeBlocks - Ninja" \
   -DCMAKE_BUILD_TYPE=Release \
   -DCMAKE_CXX_FLAGS="-std=c++1z" \
   -DCMAKE_INSTALL_PREFIX=../install_aux \
   -DCMAKE_PREFIX_PATH=$third_parties
   ../05-importedLibrary

$ ninja

$ CTEST_OUTPUT_ON_FAILURE=true   ninja test

$ ninja install
```

Using CatchMain Library (6)

```
# tests/CMakeLists.txt
find_package(CatchMain REQUIRED)
add_executable(testMoplot
    testCurve2D.cpp
)
target_link_libraries(testMoplot
    PUBLIC moplot CatchMain
)
add_test(NAME testMoplot
    COMMAND testMoplot ~[Skip])
```

Exporting cmake file (6)

```
# src/CMakeLists.txt
add_library(moplot
        curve2D.cpp
        curve2D.hpp
)
target_include_directories(moplot PUBLIC
        $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>
        $<INSTALL_INTERFACE:include/moplot>
)
install(TARGETS moplot EXPORT moplotConfig
        RUNTIME DESTINATION bin
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib
)
install(FILES curve2D.hpp
        DESTINATION include/moplot
        COMPONENT Devel)
include(CMakePackageConfigHelpers)
install(EXPORT moplotConfig
        FILE moplotConfig.cmake
        DESTINATION lib/cmake/moplot)
```

Exporting dependencies

```
# https://github.com/Marc--Olivier/third_parties/blob/master/src/google-benchmark-main/CMakeLists.txt
install(TARGETS GoogleBenchmarkMain EXPORT GoogleBenchmarkMainTarget
        RUNTIME DESTINATION bin
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib)
include(CMakePackageConfigHelpers)
set(ConfigPackageLocation lib/cmake/GoogleBenchmarkMain)
install(FILES GoogleBenchmarkMainConfig.cmake
        DESTINATION ${ConfigPackageLocation})
install(EXPORT GoogleBenchmarkMainTarget
        FILE GoogleBenchmarkMainTarget.cmake
        NAMESPACE "benchmark::"
        DESTINATION ${ConfigPackageLocation})
```

```
# https://github.com/Marc--Olivier/third_parties/blob/master/src/google-benchmark-main/GoogleBenchmarkMainConfig.cmake
if(NOT TARGET benchmark::benchmark)
        find_package(benchmark)
endif()
include("${CMAKE_CURRENT_LIST_DIR}/GoogleBenchmarkMainTarget.cmake")
```

Packaging binaries (7)

```
# ./CMakeLists.txt
cmake_minimum_required(VERSION 3.7)
project(pwd LANGUAGES CXX VERSION 1.0)

find_package(Boost REQUIRED COMPONENTS filesystem system)
add_executable(pwd
	main.cpp
)
# To run pwd, set PATH (Windows), LD_LIBRARY_PATH (Linux),
# or DYLD_LIBRARY_PATH (macOS) to $THIRD_PARTIES/lib
target_link_libraries(pwd PUBLIC
	Boost::boost ${Boost_FILESYSTEM_LIBRARY} $
{Boost_SYSTEM_LIBRARY})
install(TARGETS pwd
	RUNTIME DESTINATION bin
	LIBRARY DESTINATION lib)
install(FILES ${Boost_FILESYSTEM_LIBRARY} ${Boost_SYSTEM_LIBRARY}
	DESTINATION bin)
# BundleUtilities should be used to fix runtime dependencies
include(CPack)
```

Evaluating CMake

Advantages

- Reliable
- Fast (at least when used with Ninja)
- Not many differences between different platforms
- Plays well with IDEs (Visual Studio, Xcode, QtCreator, CLion, Eclipse, CodeBlocks)
- Plays well with third-party libraries (easy to switch between debug and release)
- Compiles in external build directories

Drawbacks

- Writing CMake extensions (functions, macros) is painful
- Export of libraries could be improved (aliases, package dependencies)
- Lack of documentation to create redistributable binaries (e.g. copying third-party dlls on Windows, fix rpath on macOS)

Catch (short) tutorial

First test using Catch (5)

```
// tests/testCurve2D.cpp
#define CATCH_CONFIG_MAIN
#include <catch.hpp>

#include "Curve2D.hpp"

namespace moplot {
namespace tests {
SCENARIO("Test create Curve2D", "[Curve2D]") {
    GIVEN("None") {
        WHEN("curve = Curve2D{{1.0, 2.0}, {}}") {
            auto curve = Curve2D{{1.0, 2.0}, {}};
        }
    }
}


}
}
```

Why using library CatchMain (6)

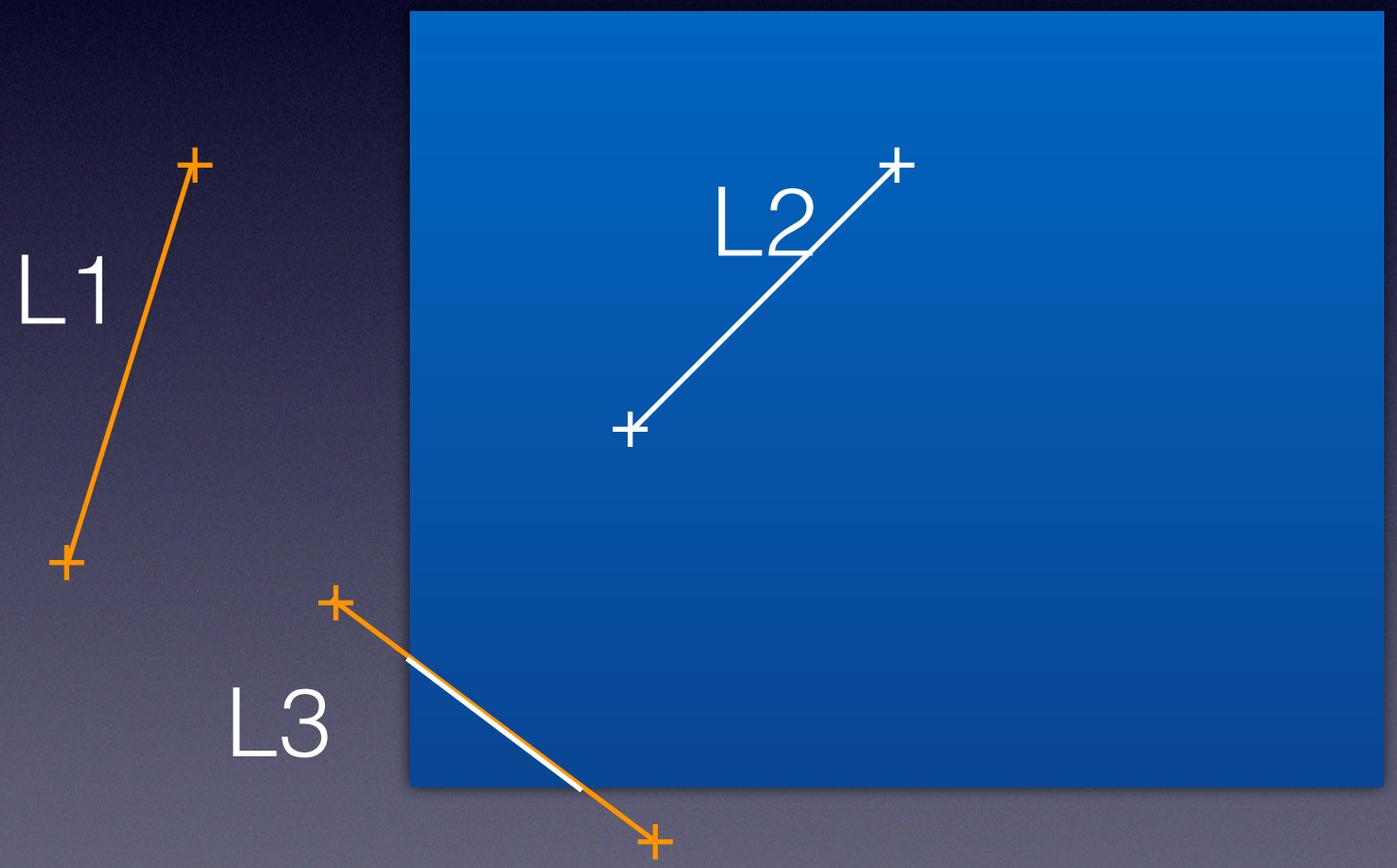
```
// ${third_parties}/CatchMain/catchMain.cpp
#define CATCH_CONFIG_MAIN
#include <catch.hpp>
```

```
// tests/testCurve2D.cpp
#include <catch.hpp>

#include "Curve2D.hpp"

namespace moplot {
namespace tests {
SCENARIO("Test create Curve2D", "[Curve2D]") {
    GIVEN("None") {
        WHEN("curve = Curve2D{{1.0, 2.0}, {}}") {
            auto curve = Curve2D{{1.0, 2.0}, {}};
        }
    }
}

}
}
```

Checking exception (8)

```
SCENARIO("Test create Curve2D", "[Curve2D]") {
    GIVEN("None") {
        WHEN("curve = Curve2D{{1.0, 2.0}, {}}") {
            THEN("Throws std::invalid_argument("
                 "\"Invalid sizes: 2 xvals vs. 0 yvals\")") {
                CHECK_THROWS_AS(Curve2D({1.0, 2.0}, {}),
                    std::invalid_argument);
                CHECK_THROWS_WITH(Curve2D({1.0, 2.0}, {}),
                    "Invalid sizes: 2 xvals vs. 0 yvals");
            }
        }
    }
}
```

Intersection line/rectangle



Checking intersection (8)

```
SCENARIO("Intersection line/rectangle", "[Curve2D]") {
  GIVEN("rect = Rect{xmin=100, xmax=500, ymin=10, ymax=60}") {
    auto rect = Rect{100, 500, 10, 60};
    WHEN("intersection = intersect("
         "rect, Line{x1=350, y1=20, x2=120, y2=40})") {
      auto intersection = intersect(rect, Line{350.0, 20.0, 120.0, 40.0});
      THEN("!intersection.empty and "
           "intersection.line == Line{x1=350, y1=20, x2=120, y2=40}") {
        REQUIRE(!intersection.empty);
        CHECK(intersection.line.x1 == Approx(350.0));
        CHECK(intersection.line.y1 == Approx(20.0));
        CHECK(intersection.line.x2 == Approx(120.0));
        CHECK(intersection.line.y2 == Approx(40.0));
      }
    }
  }
}
```

Rect, Line, Intersection (8)

```
struct Rect {
	double xmin;
	double xmax;
	double ymin;
	double ymax;
};

struct Line {
	double x1;
	double y1;
	double x2;
	double y2;
};

struct Intersection {
	Line line;
	bool empty;
};

Intersection intersect(const Rect& rect, const Line& line);
```

Using matchers (8)

```
SCENARIO("Intersection line/rectangle", "[Curve2D]") {
  GIVEN("rect = Rect{xmin=100, xmax=500, ymin=10, ymax=60}") {
    auto rect = Rect{100, 500, 10, 60};
    WHEN("intersection = intersect("
         "rect, Line{x1=350, y1=20, x2=120, y2=40})") {
      auto intersection = intersect(rect, Line{350.0, 20.0, 120.0, 40.0});
      THEN("!intersection.empty and "
           "intersection.line == Line{x1=350, y1=20, x2=120, y2=40}")
           {
        REQUIRE(!intersection.empty);
        CHECK_THAT(intersection.line,
          Equals(Line{350.0, 20.1, 120.0, 40.0}));
      }
    }
  }
}
```

Implementing matcher (8)

```
class MatcherLineEquals : public Catch::MatcherBase<Line> {
  Line& const line_;
public:
  MatcherLineEquals(Line const& line) : line_(line) {}
  virtual bool match(Line const& rhs) const override { return
    rhs.x1 == Approx(line_.x1) && rhs.y1 == Approx(line_.y1) &&
    rhs.x2 == Approx(line_.x2) && rhs.y2 == Approx(line_.y2);
  }
  virtual std::string describe() const override {
    return "!= " + Catch::toString(line_);
  }
};

inline auto Equals(Line const& line) {
  return MatcherLineEquals(line);
}
```

Catch::StringMaker (8)

```
namespace Catch {
template<> struct StringMaker<moplot::Line> {
    static std::string convert(moplot::Line const& line) {
        std::ostringstream ss;
        ss << "{x1=" << line.x1 << ", y1=" << line.y1
            << ", x2=" << line.x2 << ", y2=" << line.y2 << "}";
        return ss.str();
    }
};
}
```

“*Programming with GUTs*”

Kevlin Henney, Build Stuff 2015

video: https://www.youtube.com/watch?v=azoucC_fwzw

Google benchmark

in 4 slides

Running google benchmark

```
-----------------------------------------------------------------
Benchmark                       Time             CPU Iterations
-----------------------------------------------------------------
BM_Intersect/1024           19855 ns        19806 ns        40715
BM_Intersect/2048           68584 ns        68397 ns        12973
BM_Intersect/4096          163424 ns       162922 ns         4407
```

Function name

Input size

Minimum real time

Minimum CPU time

Number of iterations

“In all cases, the number of iterations for which the benchmark is run is governed by the amount of time the benchmark takes”

https://github.com/google/benchmark#controlling-number-of-iterations

CMake & google benchmark (9)

```
# tests/CMakeLists.txt
...
find_package(benchmark)
add_executable(benchmarkMoplot
    benchmarkCurve2D.cpp
)
target_link_libraries(benchmarkMoplot
    PUBLIC moplot benchmark::benchmark
)
```

Implementing benchmark (9)

```
#include <benchmark/benchmark.h>
#include "Curve2D.hpp"

BENCHMARK_MAIN();
namespace moplot { namespace bm {
void BM_Intersect(benchmark::State& state) {
    auto lines = generateLines(state.range(0), Rect{0.0, 1000.0, 0.0, 100.0});
    Rect rect{200.0, 800.0, 20.0, 80.0};
    while (state.KeepRunning()) {
        for(const auto& line : lines) {
            intersect(rect, line);
        }
    }
}
BENCHMARK(BM_Intersect)->RangeMultiplier(2)->Range(1 << 10, 1 << 12);
} }
```

"Tuning C++: Benchmarks, and CPUs, and Compilers! Oh My!"

Chandler Carruth, CppCon 2015

video: https://www.youtube.com/watch?v=nXaxk27zwlk

You can now

- Code
- Build
- Test
- Benchmark
- Refactor

in C++

Thank you!

References

- [1] CMake documentation - https://cmake.org/cmake/help/latest/index.html
- [2] CMake/Tutorials - https://cmake.org/Wiki/CMake/Tutorials
- [3] cmake-packages - https://cmake.org/cmake/help/latest/manual/cmake-packages.7.html
- [4] ctest(1) - https://cmake.org/cmake/help/latest/manual/ctest.1.html
- [5] Catch documentation - https://github.com/philsquared/Catch/tree/master/docs
- [6] Google/benchmark - https://github.com/google/benchmark