

# Build Systems: Combining CUDA and Modern CMake

GTC, San Jose, CA  
May, 2017

Robert Maynard

# Kitware, Inc.

- Founded in 1998 by five former GE Research employees
- 136 current employees; 47 with PhDs
- Privately held, profitable from creation
- Offices
  - Clifton Park, NY
  - Carrboro, NC
  - Santa Fe, NM
  - Lyon, France



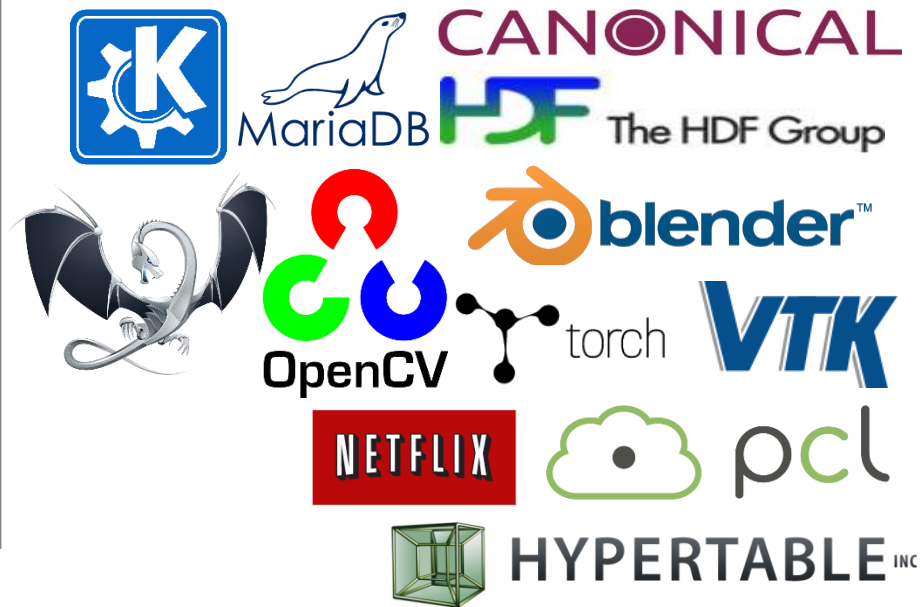
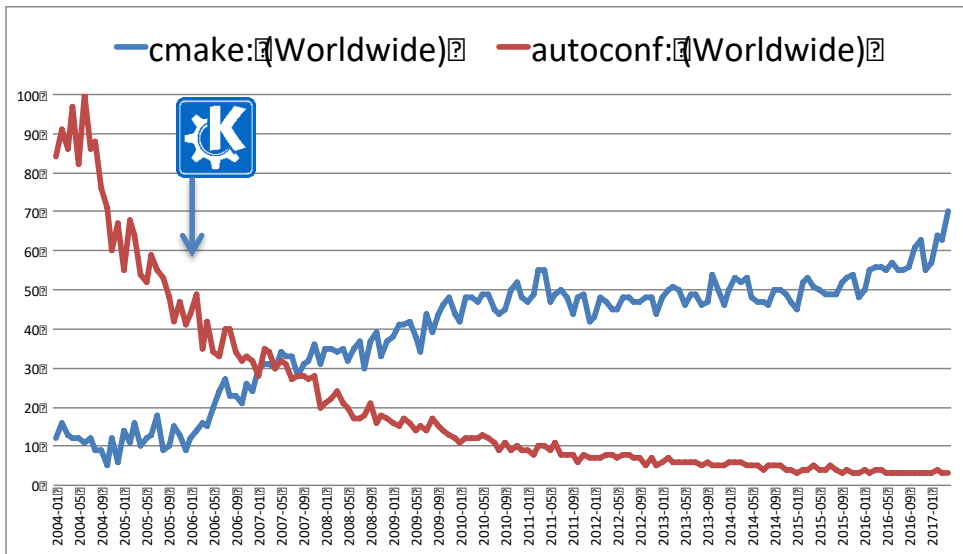
# Business Model: Open Source

- Open-source Software
  - Normally BSD-licensed
- Collaborative Research and Development
- Technology Integration
- Services, Support, Training, and Consulting



# Why CMake? Everyone is using it

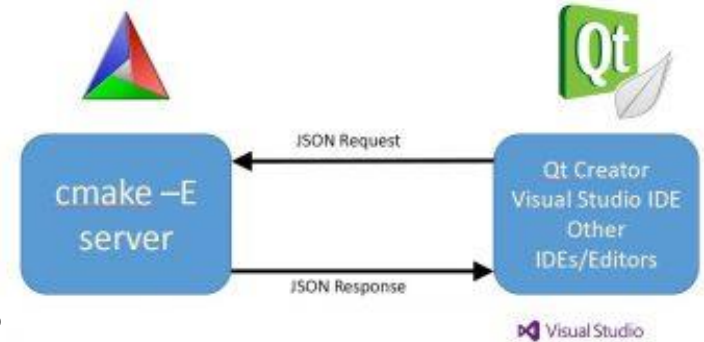
## KDE 2006 – First Tipping Point!



# Why CMake? Everyone is using it

## 2016-17 – Second Tipping Point!

- Introduction of CMake Server
  - QtCreator
  - VisualStudio 2017
- C++ Package Managers
  - Conan.io – provides helper scripts
  - Microsoft.vckpg
- Native CUDA language support
- CMake 3.5, 3.6, 3.7, and 3.8 in the last 14 months



# Classic CMake

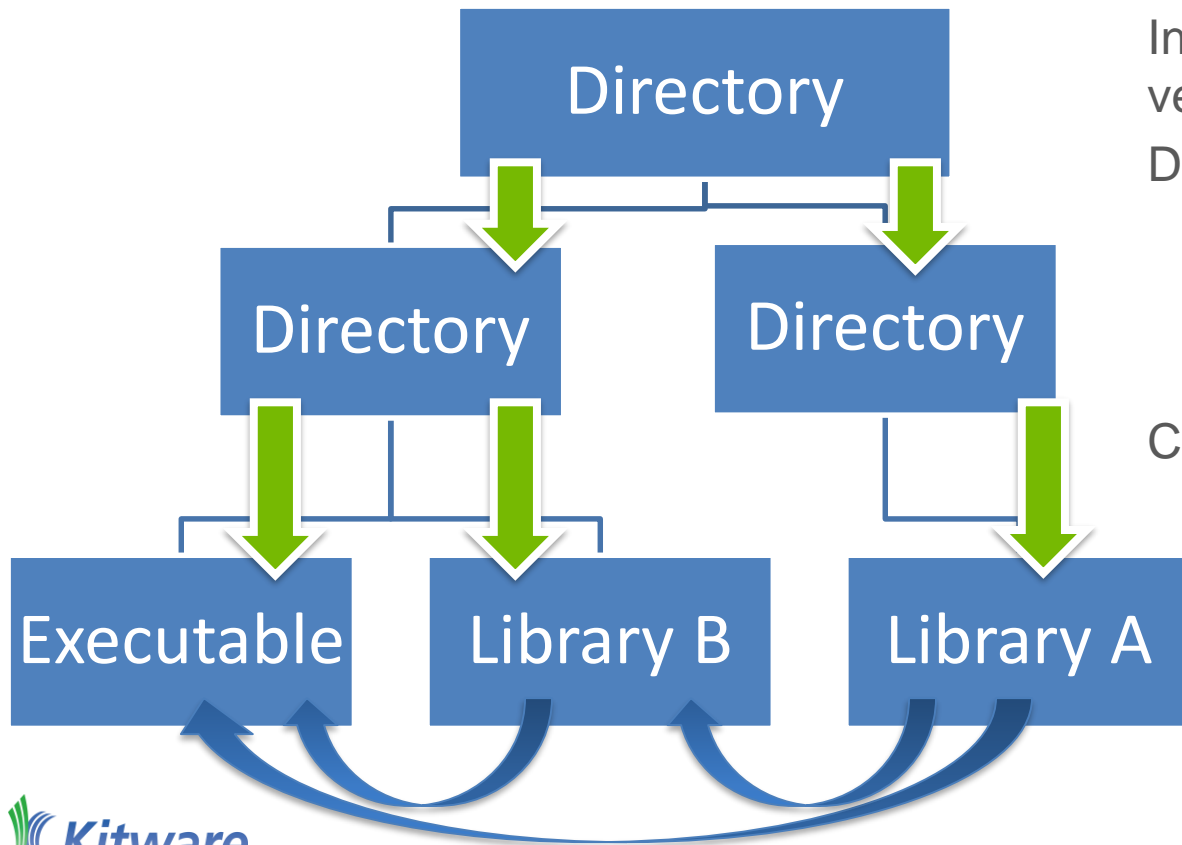
In reality most projects have a very directory centric model

Downward propagation

- Include Directories
- Compile Definitions
- Find Packages

Consumers have to know:

- Does the dependency generate build tree files
- Does the dependency use any new external package



# Modern CMake

- Modern CMake uses new(er) APIs
- Modern CMake is target focused
  - Include Directories
  - Compile Options
  - Compile Definitions
- Modern CMake introduces the concept of usage requirements
  - PUBLIC
  - PRIVATE
  - INTERFACE
- Modern CMake is more declarative

# Let's write CMake code!

```
cmake_minimum_required(VERSION 3.8)
project(Example CUDA CXX)

if(NOT DEFINED CMAKE_CXX_STANDARD)
    set(CMAKE_CXX_STANDARD 11)
    set(CMAKE_CXX_STANDARD_REQUIRED True)
endif()

if(NOT DEFINED CMAKE_CUDA_STANDARD)
    set(CMAKE_CUDA_STANDARD 11)
    set(CMAKE_CUDA_STANDARD_REQUIRED True)
endif()
```

```
-- The CUDA compiler identification is NVIDIA 8.0.61
-- The CXX compiler identification is AppleClang 7.3.0.7030029
-- Check for working CUDA compiler: /usr/local/cuda/bin/nvcc
-- Check for working CUDA compiler: /usr/local/cuda/bin/nvcc -- works
-- Detecting CUDA compiler ABI info
-- Detecting CUDA compiler ABI info - done
```



# Let's write CMake code!

Search:

☐ Grouped ☒ Advanced

Name	Value
CMAKE_AR	/usr/bin/ar
CMAKE_BUILD_TYPE	Debug
CMAKE_COLOR_MAKEFILE	<input checked="" type="checkbox"/>
CMAKE_CUDA_COMPILER	/usr/local/cuda/bin/nvcc
CMAKE_CUDA_FLAGS	-arch=sm_30
CMAKE_CUDA_FLAGS_DEBUG	-g
CMAKE_CUDA_FLAGS_MINSIZEREL	-O1 -DNDEBUG
CMAKE_CUDA_FLAGS_RELEASE	-O3 -DNDEBUG
CMAKE_CUDA_FLAGS_RELWITHDEBINFO	-O2 -g -DNDEBUG
CMAKE_CUDA_HOST_COMPILER	
CMAKE_CXX_COMPILER	/Applications/Xcode.app/Contents/Developer/Toolchains/Xco...
CMAKE_CXX_FLAGS	
CMAKE_CXX_FLAGS_DEBUG	-g

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Unix Makefiles

# Shared Library Mixed Languages

```
#build the black_scholes library  
add_library(black_scholes SHARED  
    black_scholes/Serial.cpp  
    black_scholes/Parallel.cu  
)  
target_include_directories(black_scholes  
    PRIVATE ${CMAKE_CURRENT_SOURCE_DIR}/black_scholes/  
    INTERFACE  
        $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/black_scholes/>  
        $<INSTALL_INTERFACE:include/example/black_scholes>  
)
```

**PRIVATE:** only the given target will use it

**INTERFACE:** only consuming targets use it

**BUILD\_INTERFACE:** used by consumers from this project or use the build directory

**INSTALL\_INTERFACE:** used by consumers after this target has been installed 10

# Shared Library Mixed Languages

```
target_compile_definitions(black_scholes
    PRIVATE KW_EXPORTS
    INTERFACE KW_IMPORTS
)
target_link_libraries(black_scholes
    PUBLIC compiler_info
)
```

**PUBLIC:** given target and consuming targets will use it

# Interface Library

```
#Compiler information is a header only library
add_library(compiler_info INTERFACE)
target_include_directories(compiler_info INTERFACE
    ${BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/compiler_info}
    ${INSTALL_INTERFACE:include/example/compiler_info}
)

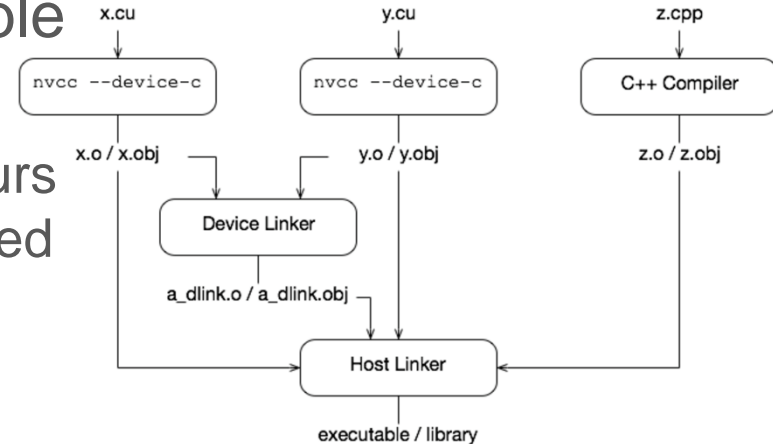
set(cxx_lang "${COMPILE_LANGUAGE:CXX}")
set(cuda_lang "${COMPILE_LANGUAGE:CUDA}")
set(debug_cxx_lang "${AND:${CONFIG:DEBUG},${cxx_lang}}")
set(debug_cuda_lang "${AND:${CONFIG:DEBUG},${cuda_lang}}")
target_compile_options(compiler_info INTERFACE
    #Build flags we want for all CXX builds
    ${${cxx_lang}:${BUILD_INTERFACE:-Wall}}
    #Build flags we want for all CUDA builds
    ${${cuda_lang}:${BUILD_INTERFACE:-Xcompiler=-Wall}}
    #Build flags we want for all CXX debug builds
    ${${debug_cxx_lang}:${BUILD_INTERFACE:-Wshadow;-Wunused-parameter}}
    #Build flags we want for all CUDA debug builds
    ${${debug_cuda_lang}:${BUILD_INTERFACE:-Xcompiler=-Wshadow;-Wunused-parameter}}
)
```

# Lets Run CMake

```
[ 25%] Building CXX object CMakeFiles/black_scholes.dir/black_scholes/Serial.cpp.o
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++ -DKW_EXPORTS -Dblack_scholes_EXPORTS -I/Users/robert/Work/cmake_tutorial/cuda_src/producer/black_scholes -I/Users/robert/Work/cmake_tutorial/cuda_src/producer/compiler_info -g -fPIC -Wall -Wshadow -Wunused-parameter -std=gnu++11 -o CMakeFiles/black_scholes.dir/black_scholes/Serial.cpp.o -c /Users/robert/Work/cmake_tutorial/cuda_src/producer/black_scholes/Serial.cpp
[ 50%] Building CUDA object CMakeFiles/black_scholes.dir/black_scholes/Parallel.cu.o
/usr/local/cuda/bin/nvcc -DKW_EXPORTS -Dblack_scholes_EXPORTS -I/Users/robert/Work/cmake_tutorial/cuda_src/producer/black_scholes -I/Users/robert/Work/cmake_tutorial/cuda_src/producer/compiler_info -arch=sm_30 -g -Xcompiler=-fPIC -Xcompiler=-Wall -Xcompiler=-Wshadow,-Wunused-parameter -std=c++11 -x cu -c /Users/robert/Work/cmake_tutorial/cuda_src/producer/black_scholes/Parallel.cu -o CMakeFiles/black_scholes.dir/black_scholes/Parallel.cu.o
[ 75%] Linking CXX device code/Users/robert/Work/cmake_tutorial/cuda_build/CMakeFiles/black_scholes.dir/cmake_device_link.o
/Applications/Xcode.app/Contents/bin/cmake -E cmake_link_script CMakeFiles/black_scholes.dir/dlink.txt --verbose=1
/usr/local/cuda/bin/nvcc -arch=sm_30 -g -Xcompiler=-fPIC -shared -dlink CMakeFiles/black_scholes.dir/black_scholes/Serial.cpp.o CMakeFiles/black_scholes.dir/black_scholes/Parallel.cu.o -o CMakeFiles/black_scholes.dir/cmake_device_link.o -L/usr/local/cuda/lib
[100%] Linking CXX shared library libblack_scholes.dylib
```

# Separable Compilation

- Separable compilation allows CUDA code to call device functions implemented in other translation units
- CMake 3.8 is capable of separable compilation and device linking
  - device linking of static libraries occurs when they are consumed by a shared library or executable




# Separable Compilation

```
add_library(support STATIC support_functions.cu)
set_target_properties(support PROPERTIES
    CUDA_SEPARABLE_COMPILATION ON
    PROPERTIES POSITION_INDEPENDENT_CODE ON)
target_link_libraries(support PRIVATE compiler_info)

add_library(black_scholes
    black_scholes/Serial.cpp
    black_scholes/Parallel.cu
)
...
target_link_libraries(black_scholes PUBLIC compiler_info support)
```

# Separable Compilation

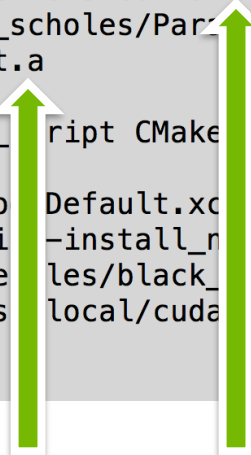
```
[ 20%] Building CUDA object CMakeFiles/support.dir/support_functions.cu.o
/usr/local/cuda/bin/nvcc -I/Users/robert/Work/cmake_tutorial/cuda_src/producer/compiler_inf
o -arch=sm_30 -g -Xcompiler=-fPIC -Xcompiler=-Wall -Xcompiler=-Wshadow,-Wunused-parameter
-std=c++11 -x cu -dc /Users/robert/Work/cmake_tutorial/cuda_src/producer/support_functions.cu
-o CMakeFiles/support.dir/support_functions.cu.o
[ 40%] Linking CUDA static library libsupport.a
```





# Separable Compilation

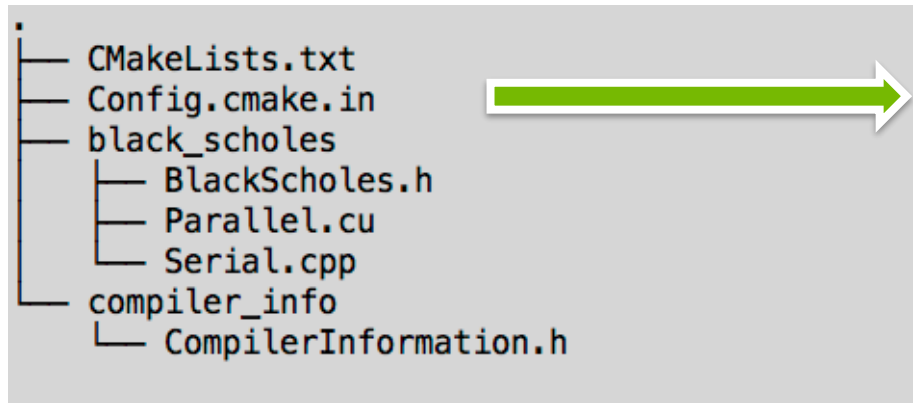
```
[ 83%] Linking CUDA device code/Users/robert/Work/cmake_tutorial/cuda_build/CMakeFiles/black_scholes.dir/cmake_device_link.o
/Applications/CMake.app/Contents/bin/cmake -E cmake_link_script CMakeFiles/black_scholes.dir/dlink.txt --verbose=1
/usr/local/cuda/bin/nvcc -arch=sm_30 -g -Xcompiler=-fPIC -shared -dlink CMakeFiles/black_scholes.dir/black_scholes/Serial.cpp.o CMakeFiles/black_scholes.dir/black_scholes/Parallel.cu.o -o CMakeFiles/black_scholes.dir/cmake_device_link.o -L/usr/local/cuda/lib libsupport.a
[100%] Linking CXX shared library libblack_scholes.dylib
/Applications/CMake.app/Contents/bin/cmake -E cmake_link_script CMakeFiles/black_scholes.dir/link.txt --verbose=1
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++ -g -dynamiclib -Wl,-headerpad_max_install_names -o libblack_scholes.dylib -install_name @rpath/libblack_scholes.dylib CMakeFiles/black_scholes.dir/black_scholes/Serial.cpp.o CMakeFiles/black_scholes.dir/black_scholes/Parallel.cu.o CMakeFiles/black_scholes.dir/cmake_device_link.o -L/usr/local/cuda/lib libsupport.a /usr/local/cuda/lib/libcudart_static.a -lcudadevrt
[100%] Built target black_scholes
```



# Export Configuration

CMake is able to produce two types of config files

1. You have config files that are part of the build tree of a project. These contain build paths that can only be used on the current machine
2. You have installed config files that are meant to be machine relocatable. Large projects generally ship these as part of the SDK




Export Configuration File

# Export Configuration

```
# generate the config file that includes the exports
include(CMakePackageConfigHelpers)
configure_package_config_file(${CMAKE_CURRENT_SOURCE_DIR}/Config.cmake.in
    "${CMAKE_CURRENT_BINARY_DIR}/Example-Config.cmake"
    INSTALL_DESTINATION "lib/cmake/example"
    NO_SET_AND_CHECK_MACRO
    NO_CHECK_REQUIRED_COMPONENTS_MACRO
)
```

@PACKAGE\_INIT@

```
include ( "${CMAKE_CURRENT_LIST_DIR}/example-targets.cmake" )
```



```
##### Expanded from @PACKAGE_INIT@ by configure_package_config_file() #####
##### Any changes to this file will be overwritten by the next CMake run #####
##### The input was Config.cmake.in #####

get_filename_component(PACKAGE_PREFIX_DIR "${CMAKE_CURRENT_LIST_DIR}/../../.." ABSOLUTE)

#####

include ( "${CMAKE_CURRENT_LIST_DIR}/example-targets.cmake" )
```

# Install Export Configuration

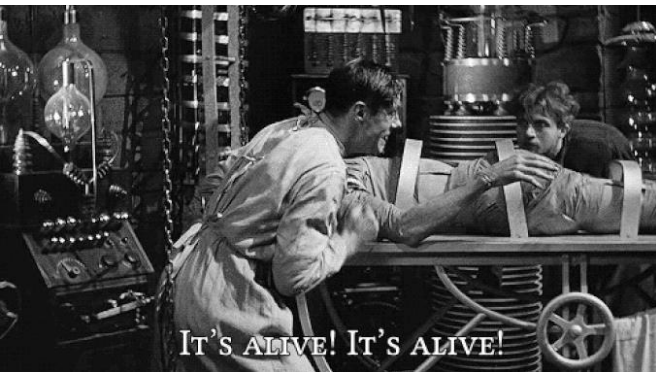
```
install(TARGETS
  compiler_info
  black_scholes
  EXPORT example
  LIBRARY DESTINATION lib
)
```

```
#Generate the export targets for the build tree
#Needs to be after the install(TARGETS ) command
export(EXPORT example FILE example-targets.cmake)
```

```
# install export file which is used by other cmake projects to import these
# targets
install(EXPORT example DESTINATION lib/cmake/example)
```

```
install(FILES
  ${CMAKE_CURRENT_BINARY_DIR}/Example-Config.cmake
  DESTINATION lib/cmake/example
)
```

# The Proof



```
# Create imported target black_scholes
add_library(black_scholes SHARED IMPORTED)

set_target_properties(black_scholes PROPERTIES
  INTERFACE_COMPILE_DEFINITIONS "KW_IMPORTS"
  INTERFACE_INCLUDE_DIRECTORIES "/Users/robert/Work/cmake_tutorial/cuda_src/prod
  INTERFACE_LINK_LIBRARIES "support;compiler_info"
)

# Create imported target support
add_library(support STATIC IMPORTED)

set_target_properties(support PROPERTIES
  INTERFACE_LINK_LIBRARIES "\\$<LINK_ONLY:compiler_info>"
)

# Import target "black_scholes" for configuration "Debug"
set_property(TARGET black_scholes APPEND PROPERTY IMPORTED_CONFIGURATIONS DEBUG)
set_target_properties(black_scholes PROPERTIES
  IMPORTED_LOCATION_DEBUG "/Users/robert/Work/cmake_tutorial/cuda_build/libblack
  IMPORTED_SONAME_DEBUG "@rpath/libblack_scholes.dylib"
)

# Import target "support" for configuration "Debug"
set_property(TARGET support APPEND PROPERTY IMPORTED_CONFIGURATIONS DEBUG)
set_target_properties(support PROPERTIES
  IMPORTED_LINK_INTERFACE_LANGUAGES_DEBUG "CUDA"
  IMPORTED_LOCATION_DEBUG "/Users/robert/Work/cmake_tutorial/cuda_build/libsuppo
)

# This file does not depend on other imported targets which have
```

IT'S ALIVE! IT'S ALIVE!

0 miss

# CMake 3.9: MSVC

- CMake 3.9 adds CUDA support to MSVC
- Will require the CUDA MSBuild extensions

# CMake 3.9: OBJECT targets

- CMake 3.9 is expanding OBJECT support
- Will be
  - Installable
  - Exportable
  - Importable
  - Usable in Generator Expressions



# CMake 3.9: PTX

- CMake 3.9 will add support for PTX files
- Will be
  - Installable
  - Exportable
  - Importable
  - Usable in Generator Expressions

# CMake 3.9: PTX

```
add_library(CudaPTXObjects OBJECT kernelA.cu kernelB.cu)
set_property(TARGET CudaPTXObjects PROPERTY CUDA_PTX_COMPILATION ON)
target_compile_definitions(CudaPTXObjects PUBLIC "PTX_FILES")

add_custom_command(
  OUTPUT "${CMAKE_CURRENT_BINARY_DIR}/embedded_objs.h"
  COMMAND ${CMAKE_COMMAND}
  "-DOBJECTS:STRING=\"${TARGET_OBJECTS:CudaPTXObjects}\""
  "-DOUTPUT:FILEPATH=\"${output_file}\""
  -P ${CMAKE_CURRENT_SOURCE_DIR}/bin2c_wrapper.cmake
  DEPENDS CudaPTXObjects
  COMMENT "Converting Object files to a C header"
)
```

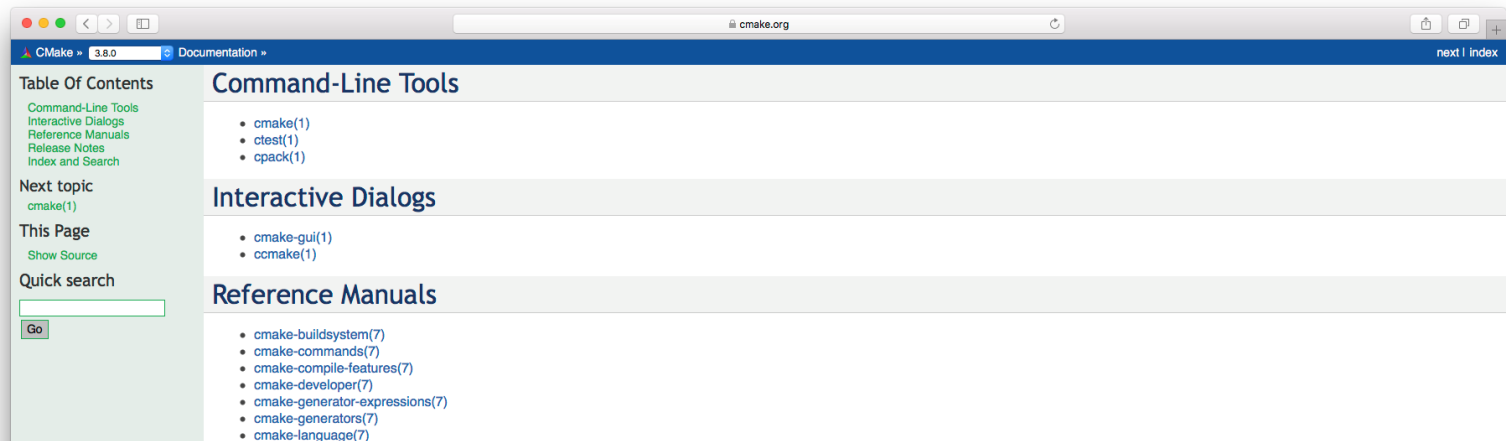
# Now that you are inspired

- Explore more CUDA+CMake snippets
  - [https://gitlab.kitware.com/robertmaynard/cmake\\_cuda\\_tests](https://gitlab.kitware.com/robertmaynard/cmake_cuda_tests)

add_definitions	Enable even more examples.	8 months ago
as_cu	Enable even more examples.	8 months ago
cmake	Update compiler_info to use the FindCUDALibs code.	2 months ago
compile_flags	Cleanup the compile flag example.	7 months ago
compiler_info	Update compiler_info to use the FindCUDALibs code.	2 months ago
consume_compile_features	Implement a consume compiler feature test.	7 months ago
cpp_consuming	Updates now that CMake CUDA has been taught implicit link dependencies	8 months ago
dynamic	Complete refactor of the test cases to be split into multiple use-cases	8 months ago
enable_cpp11	Updates now that CMake CUDA has been taught implicit link dependencies	8 months ago
generate_ptx	remove unneeded files.	4 days ago
import_ptx	Show how to use imported ptx OBJECT libraries	a month ago
mixed_dynamic	Updates now that CMake CUDA has been taught implicit link dependencies	8 months ago
mixed_static	Updates now that CMake CUDA has been taught implicit link dependencies	8 months ago

# Now that you are inspired

- Read “how to write a CMake buildsystem”
  - <https://cmake.org/cmake/help/v3.8/manual/cmake-buildsystem.7.html> Explore the CMake documentation
- Explore the CMake documentation
  - <https://www.cmake.org/cmake/help/v3.8/>



# Thank You!

Robert Maynard

[robert.maynard@kitware.com](mailto:robert.maynard@kitware.com)

@robertjmaynard

Thanks to NVIDIA for all the technical support when developing this work

Checkout out:

Kitware @ [www.kitware.com](http://www.kitware.com)

CMake @ [www.cmake.org](http://www.cmake.org)

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!