PBRTv4-小专题-PBRT 的 CMakeLists.txt 解读

Dezeming Family

2023 年 8 月 31 日

DezemingFamily 系列文章和电子文档**全部都有免费公开的电子版**，可以很方便地进行修改和重新发布。如果您获得了 DezemingFamily 的系列文章，可以从我们的网站 [https://dezeming.top/] 找到最新的版本。对文章的内容建议和出现的错误也欢迎在网站留言。

2023 年 9 月 6 日：完成本文第一版。

目录

一　引文

我本人也对 CMakeLists.txt 的编写并没有那么熟悉，还好 PBRT 的 CMakeLists.txt 也没那么长，所以就提供一个详细的解读。一方面是为了更熟悉 PBRT 工程，另一方面也是为了自己以后能够更好地编写 CMakeLists.txt。

二　代码第一部分

```
# 用于设定需要的最低版本的CMake，如果CMake运行的版本低于要求的版本，它将停止处理project并报告错误
cmake_minimum_required (VERSION 3.12)
# 工程定义
project (PBRT-V4 LANGUAGES CXX C)
# 将整个项目的C++标准设为C++17
set (CMAKE_CXX_STANDARD 17)
set (CMAKE_CXX_STANDARD_REQUIRED ON)
# For sanitizers
# cmake find_package时使用的路径
list (INSERT CMAKE_MODULE_PATH 0 "${CMAKE_SOURCE_DIR}/cmake")
```

CMAKE_MODULE_PATH 是一个列表，是在检查 CMake 附带的默认模块之前，指定要由 include() 或 find_package() 命令加载的 CMake 模块的搜索路径的目录列表。因为如果工程复杂，可能需要编写一些 cmake 模块。默认情况下，它为空。INSERT 和 0 表示把后面表示的路径插入到列表索引 0 的位置。如果把该值打印出来，就是 PBRT 源码目录下的 cmake 目录：

```
# 用message打印调试信息
message("CMAKE_MODULE_PATH = ${CMAKE_MODULE_PATH}$")
```

下面的内容提供了一些选项：

```
option (PBRT_FLOAT_AS_DOUBLE "Use 64-bit floats" OFF)
option (PBRT_BUILD_NATIVE_EXECUTABLE "Build executable optimized for CPU architecture of system pbrt was built on" ON)
option (PBRT_DBG_LOGGING "Enable (very verbose!) debug logging" OFF)
option (PBRT_NVTX "Insert NVTX annotations for NVIDIA Profiling and Debugging Tools" OFF)
option (PBRT_NVML "Use NVML for GPU performance measurement" OFF)
option (PBRT_USE_PREGENERATED_RGB_TO_SPECTRUM_TABLES "Use pregenerated rgbspectrum_*.cpp files rather than running rgb2spec_opt to generate them at build time" OFF)
```

在 CMake 中可以看到这些选项，并选择是否打开（鼠标放在条目上会显示信息）：

NVTX(Nvidia Tools Extension Library): 提供了 annotation 接口，方便可视化查看各个函数各个部分的耗时情况。

$ENV 用于获取环境变量，如果不存在该环境变量就会设置为空，此时就需要我们自己手动去设置。

```
set (PBRT_OPTIX7_PATH $ENV{PBRT_OPTIX7_PATH} CACHE PATH "Path to OptiX 7 SDK")
set (PBRT_GPU_SHADER_MODEL "" CACHE STRING "")
```

下面如何没有设置 CMake 类型或者 CMake 配置类型，就会打印信息并设置为 Release 类型。

```
if (NOT CMAKE_BUILD_TYPE AND NOT CMAKE_CONFIGURATION_TYPES)
  message (STATUS "Setting build type to 'Release' as none was specified.")
  set (CMAKE_BUILD_TYPE Release CACHE STRING "Choose the type of build."
    FORCE)
  set_property (CACHE CMAKE_BUILD_TYPE PROPERTY STRINGS "Debug" "Release" "
    MinSizeRel" "RelWithDebInfo")
endif ()
```

不过 CMake 配置类型默认是有的：

```
CMAKE_CONFIGURATION_TYPES = Debug;Release;MinSizeRel;RelWithDebInfo
```

然后定义了一个函数：

```
function (CHECK_EXT NAME DIR HASH)
```

CHECK_EXT 是函数名，后面的都是其参数。该函数用来检查 DIR 所示目录是否存在，并检查对应的 Hash 是否正确。该函数在下面被多次调用，检查了 OpenEXR/OpenVDB 等很多第三方库。

将预处理器定义添加到源文件的编译中：

```
add_compile_definitions ("$<$<CONFIG:DEBUG>:PBRT_DEBUG_BUILD>")
```

其中，参数是一个逻辑表达式 (logical expression)，用于创建条件输出。基本表达式是 0 和 1 表达式。由于其他逻辑表达式的计算结果为 0 或 1，因此可以组合它们来创建条件输出。因此，如果当前编译是 Debug 模式，就把 PBRT_DEBUG_BUILD 作为预编译宏。

启用测试（一定要在工程根目录下的 CMakeLists.txt 中设置，不然执行 make test 时会报错）。

```
enable_testing ()
```

通过 find_package 引入库（该方式只对支持 cmake 编译安装的库或者 camke 官方库有效）：

```
find_package (Sanitizers)
find_package (Threads)

find_package(OpenGL REQUIRED)
```

显式添加文件夹，在 visual studio 目录中生成文件夹：

```
set_property (GLOBAL PROPERTY USE_FOLDERS ON)
```

set_property 作用域可以是 GLOBAL，DIRECTORY，TARGET。当启动可以生成文件夹时，下面的代码（出现在本文的 CMakeLists.txt 后面）就把目标工程 soac 放在 visual studio 目录下的 cmd 目录里：

```
set_property (TARGET soac PROPERTY FOLDER "cmd")
```

如果是使用微软的 C++ 编译器，就增加下面三个宏：PBRT_IS_MSVC：

```
if (MSVC)
  list (APPEND PBRT_DEFINITIONS "PBRT_IS_MSVC" "_CRT_SECURE_NO_WARNINGS")
```

```
3   list (APPEND PBRT_DEFINITIONS "PBRT_IS_MSVC" "
        _ENABLE_EXTENDED_ALIGNED_STORAGE")
4 endif ()
```

我也不知道为什么 PBRT_IS_MSVC 被加进去两次，可能是作者的疏忽吧。

然后根据之前设置的 Options 参数 PBRT_FLOAT_AS_DOUBLE 是否被选择，决定是否加入下面的宏，来决定 Float 是 float 还是 double：

```
1 if (PBRT_FLOAT_AS_DOUBLE)
2   list (APPEND PBRT_DEFINITIONS "PBRT_FLOAT_AS_DOUBLE")
3 endif ()
4 if (PBRT_DBG_LOGGING)
5   list (APPEND PBRT_DEFINITIONS "PBRT_DBG_LOGGING")
6 endif ()
```

三　代码第二部分：第三方库

设置编译为静态库，然后把第三方库的构建目录添加过来，该目录下也有一个 CMakeLists.txt 文件：

```
1 set (BUILD_SHARED_LIBS OFF)
2 add_subdirectory (${CMAKE_CURRENT_SOURCE_DIR}/src/ext)
```

新的子目录下的 CMakeLists.txt 的解释如下。

有很多行代码都包含下面的句子：

```
1 set (STB_INCLUDE ${CMAKE_CURRENT_SOURCE_DIR}/stb PARENT_SCOPE)
```

意味着将变量 STB_INCLUDE 提供给父 CMakeLists.txt 环境（PARENT_SCOPE）所使用。有的库还需要进一步使用它自己的 CMakeLists.txt，因此有可能还会调用 add_subdirectory 函数。在源码根目录下的 CMakeLists.txt 中会把这些库都加进来：

```
1 set (ALL\_PBRT\_LIBS ...)
```

回到根目录下的 CMakeLists.txt 中。如果是 Windows 系统就用如下指令（其他系统也有类似处理，这里忽略）：

```
1 if (CMAKE_SYSTEM_NAME STREQUAL Windows)
2   list (APPEND PBRT_DEFINITIONS "PBRT_IS_WINDOWS" "NOMINMAX")
```

find_library 用来寻找库，可以给出库的路径，也可以用系统路径。

```
1 find_library (PROFILE_LIB profiler)
2 if (NOT PROFILE_LIB)
3     message (STATUS "Unable␣to␣find␣-lprofiler")
4 else ()
5     message (STATUS "Found␣-lprofiler:␣${PROFILE_LIB}")
6 endif ()
```

如果找到，后面会将该库添加到 PBRT 使用的库列表中，否则就不添加。如果没有可添加的执行文件，只能用 INTERFACE 来修饰，INTERFACE 库目标不编译源代码，也不在磁盘上生成库文件。但是它可能设置了属性，并且可以安装和导出，比如用 target_link_libraries 库来链接。

```
add_library (pbrt_warnings INTERFACE)
target_compile_options (
    pbrt_warnings
    INTERFACE
        ......
)
add_library (pbrt_opt INTERFACE)
```

四　代码第三部分：CUDA 和 Optix

```
# 定义一个不显示在项目里的库工程，用于设置一些属性并链接给其他工程
add_library (cuda_build_configuration INTERFACE)
# 检查 CUDA 是否可用
include (CheckLanguage)
check_language(CUDA)
```

整个判断 cuda 是否可用以及执行后续内容的代码如下：

```
# 判断是否可以用GPU编译以及执行编译
if (CMAKE_CUDA_COMPILER)
  if (CUDA_VERSION_MAJOR LESS 11)
    # 警告，不使用GPU编译
    ......
  else ()
    find_package (CUDA REQUIRED)
    # 执行GPU编译设置
    ......
```

执行 GPU 编译设置的代码基本结构如下：

```
# 从github的VIAME上copy的代码，我也不知道这是干什么的
if (NOT CUDA_VERSION_PATCH)
  if (CUDA_NVCC_EXECUTABLE AND
      CUDA_NVCC_EXECUTABLE STREQUAL CMAKE_CUDA_COMPILER AND
      CMAKE_CUDA_COMPILER_VERSION MATCHES [=[([0-9]+)\.([0-9]+)\.([0-9]+)
          ]=])
    set (CUDA_VERSION_PATCH "${CMAKE_MATCH_3}")
  elseif (CUDA_NVCC_EXECUTABLE)
    execute_process (COMMAND ${CUDA_NVCC_EXECUTABLE} "--version"
       OUTPUT_VARIABLE NOUT)
    if (NOUT MATCHES [=[ V([0-9]+)\.([0-9]+)\.([0-9]+)]=])
      set (CUDA_VERSION_PATCH "${CMAKE_MATCH_3}")
    endif ()
  endif ()
endif ()

if (CUDA_VERSION_MAJOR EQUAL 11 AND CUDA_VERSION_MINOR EQUAL 3 AND
   CUDA_VERSION_PATCH LESS 109)
```

```
  # 报错，因为CUDA 11.3编译会失败
  ......
endif ()

if ("${PBRT_OPTIX7_PATH}" STREQUAL "")
  # 警告
else ()
  # 存在Optix才允许使用CUDA编译器
  enable_language (CUDA)
  # 添加一些宏定义
  list (APPEND PBRT_DEFINITIONS "PBRT_BUILD_GPU_RENDERER")
  if (PBRT_NVTX)
      list (APPEND PBRT_DEFINITIONS "NVTX")
  endif ()
  if (PBRT_NVML)
      list (APPEND PBRT_DEFINITIONS "PBRT_USE_NVML")
  endif ()
  set (PBRT_CUDA_ENABLED ON)
  # 配置Optix环境
  ......
```

配置 Optix 环境部分，

```
# 包含cuda头文件
include_directories (${CMAKE_CUDA_TOOLKIT_INCLUDE_DIRECTORIES})
# 配置nvcc编译选项
target_compile_options(
  pbrt_warnings
  INTERFACE
  ......
)
# Willie hears ych..
string (APPEND CMAKE_CUDA_FLAGS " -Xnvlink -suppress-stack-size-warning")
target_compile_options (
  cuda_build_configuration
  INTERFACE
  ......
)
```

注意，pbrt_warnings 和 cuda_build_configuration 都是前面声明过的定义为 INTERFACE 类型的库。CMAKE_CUDA_FLAGS 用以设置 nvcc 编译选项。

剩下的关于 CUDA 和 Optix 的内容主要是设置其生成格式，包括 optix.cu.ptx_embedded.c 文件等，以后有时间再分析。

五　代码第四部分：编译指定

检查当前编译器是否支持 C++ 编译，"-march=native" 是需要检查的编译选项，COMPILER_SUPPORTS_MARCH_NATIVE 是存储检查结果的变量名。

```
# 必须包含
include (CheckCXXCompilerFlag)
# 检查编译选项本地编译器
check_cxx_compiler_flag ("-march=native" COMPILER_SUPPORTS_MARCH_NATIVE)
if (COMPILER_SUPPORTS_MARCH_NATIVE AND PBRT_BUILD_NATIVE_EXECUTABLE)
    target_compile_options (pbrt_opt INTERFACE
          "$<$<COMPILE_LANGUAGE:CUDA>:SHELL:-Xcompiler␣>-march=native")
endif ()
```

下面的内容我也不是很理解，谷歌上也查不到什么资料：

```
if (CMAKE_CXX_COMPILER_ID STREQUAL "Intel")
  list (APPEND PBRT_CXX_FLAGS "-std=c++17")

  find_program (XIAR xiar)
  if (XIAR)
    set (CMAKE_AR "${XIAR}")
  endif (XIAR)
  mark_as_advanced (XIAR)

  find_program(XILD xild)
  if (XILD)
    set (CMAKE_LINKER "${XILD}")
  endif (XILD)
  mark_as_advanced (XILD)

  # ICC will default to -fp-model fast=1, which performs value-unsafe
      optimizations which will
  # cause pbrt_test to fail. For safety, -fp-model precise is explicitly set
      here by default.
  set (FP_MODEL "precise" CACHE STRING "The␣floating␣point␣model␣to␣compile␣
      with.")
  set_property (CACHE FP_MODEL PROPERTY STRINGS "precise" "fast=1" "fast=2")

  list (APPEND PBRT_CXX_FLAGS "-fp-model" "${FP_MODEL}")
endif ()
```

之后有大量代码都是做检查，检查 CXX 编译器是否支持给定的 flag：

```
include (CheckCXXSourceCompiles)
check_cxx_source_compiles (
"
#␣一堆代码
...
" HAS_INTRIN_H)

if (HAS_INTRIN_H)
  list (APPEND PBRT_DEFINITIONS "PBRT_HAS_INTRIN_H")
endif ()
```

六　代码第五部分：代码构建工程

代码构建一共构建了下面的这些项目：

```
soac
pbrt_lib
rgb2spec_opt
pbrt_exe
imgtool
......
```

首先是源码的目录设置，将源码文件打包到不同的组：

```
source_group ("Source Files" FILES ${PBRT_SOURCE})
source_group ("Header Files" FILES ${PBRT_SOURCE_HEADERS})
source_group ("Source Files/cpu" FILES ${PBRT_CPU_SOURCE})
source_group ("Header Files/cpu" FILES ${PBRT_CPU_SOURCE_HEADERS})
source_group ("Source Files/util" FILES ${PBRT_UTIL_SOURCE})
source_group ("Header Files/util" FILES ${PBRT_UTIL_SOURCE_HEADERS})
source_group ("Source Files/wavefront" FILES ${PBRT_WAVEFRONT_SOURCE})
source_group ("Header Files/wavefront" FILES ${PBRT_WAVEFRONT_SOURCE_HEADERS
    })
if (PBRT_CUDA_ENABLED)
  source_group ("Source Files/gpu" FILES ${PBRT_GPU_SOURCE})
  source_group ("Header Files/gpu" FILES ${PBRT_GPU_SOURCE_HEADERS})
endif ()
```

注意其中出现了下面的代码，用于对 MSVC 编译器增加调试工具 natvis：

```
if (MSVC)
    set (PBRT_SOURCE ${PBRT_SOURCE} src/pbrt/visualstudio.natvis)
endif ()
```

如果支持 GPU，就把相关的 cpp 文件设置编译语言：

```
set_source_files_properties (
  src/pbrt/bsdf.cpp
  ......

  ${PBRT_WAVEFRONT_SOURCE}
  ${PBRT_GPU_SOURCE}

  src/pbrt/cmd/pspec_gpu.cpp

  PROPERTIES LANGUAGE CUDA
)
# 必须要将optix源码生成.ptx文件来调用执行
cuda_compile_and_embed (PBRT_EMBEDDED_PTX src/pbrt/gpu/optix.cu optix.cu)
```

有些工程，比如 soac，使用 add_custom_command(...) 将构建得到的 soac 可执行代码作用于 pbrt.soa，以生成新的头文件 pbrt_soa.h。该头文件后面会加入到 PBRT 的其他项目中。

```
add_custom_command (OUTPUT ${CMAKE_CURRENT_BINARY_DIR}/pbrt_soa.h
    COMMAND soac ${CMAKE_SOURCE_DIR}/src/pbrt/pbrt.soa > ${
        CMAKE_CURRENT_BINARY_DIR}/pbrt_soa.h
    DEPENDS soac ${CMAKE_SOURCE_DIR}/src/pbrt/pbrt.soa)
set (PBRT_SOA_GENERATED ${CMAKE_CURRENT_BINARY_DIR}/pbrt_soa.h)
```

至此，整个 PBRT 的 CMakeLists.txt 就分析完了，而里面有不少设置本人也确实难以理解，有些内容也是 PBRT 的作者从其他项目中 copy 过来的，也缺乏一些具体的说明。但这些代码我们可以当成一个固定的代码模板来使用。

如果以后对于 CMakeLists.txt 有新的认识，还会对本文进行补充。

参考文献

[1] https://github.com/mmp/pbrt-v4

[2] https://pbrt.org/

[3] https://developer.nvidia.com/designworks/optix/downloads/legacy

[4] https://pbrt.org/resources

[5] https://www.cnblogs.com/Heskey0/category/2166679.html

[6] https://github.com/ebruneton/clear-sky-models/tree/master/atmosphere/model/hosek

[7] https://www.cnblogs.com/Heskey0/p/15973546.html