第七集：实例应用：如何使用 CMake 集成 OpenCV 和自定义模块

---

1. 引言

欢迎来到 CMake 教学的第七集！这是我们的最后一集，在这一集中，我们将回顾之前学到的所有知识，并通过一个完整的示例来展示如何使用 CMake 集成 OpenCV 和自定义模块。通过这个实例，您将更好地掌握 CMake 的核心概念以及如何使用 CMake 构建和管理实际项目。

我们会重点强调 CMake 的配置与使用，而 OpenCV 和自定义模块会作为项目的功能展示来辅助教学。最后，您将了解如何在复杂的项目中使用 CMake 来高效管理依赖库和项目结构。

---

2. 项目结构和准备

首先，我们来看一下项目的基本结构：

```
CMake_OpenCV_Project/
|
├── CMakeLists.txt    // 顶层 CMake 配置文件
├── build/            // 构建目录（自动生成）
├── config/           // 配置文件目录
├── kun.mp4           // 视频文件资源
├── main.cpp          // 主程序文件
├── modules/          // 自定义模块目录
|   └── cv/           // 自定义模块源码
├── readme.md         // 项目文档
```

这个项目会包含一个主程序 `main.cpp`，用于演示如何集成 OpenCV 库来处理图像或视频。我们将通过 CMake 进行编译和链接，展示如何在项目中集成第三方库（如 OpenCV）以及如何使用自定义模块（如 `modules/cv`）。

---

3. CMake 项目配置文件解析

顶层 CMakeLists.txt 文件

在项目根目录下的 `CMakeLists.txt` 文件用于配置项目的总体构建流程，它的核心目标是找到外部库（如 OpenCV）、配置自定义模块，并生成最终的可执行文件。

```
cmake_minimum_required(VERSION 3.15)

# 定义项目名称
project(CMAKE_TUTORIAL)

# 定义一个名为 PROJECT_DIR 的编译宏，方便在代码中使用项目的根目录
add_compile_definitions(PROJECT_DIR="${PROJECT_SOURCE_DIR}")

# 打印项目根目录，便于调试
message("${PROJECT_DIR}")

# 指定可执行文件的生成目标，将 main.cpp 编译为可执行文件
add_executable(${PROJECT_NAME} main.cpp)

# 设置 C++ 标准为 C++17
set(CMAKE_CXX_STANDARD 17)

# 开启编译命令导出（用于生成 compile_commands.json，便于编辑器自动补全）
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)

# 设置 CMake 前缀路径，CMake 将在这里查找依赖库
set(CMAKE_PREFIX_PATH "/")

# 添加子目录 modules，进行模块构建
add_subdirectory(modules)

# 链接 modules 库到主程序 CMAKE_TUTORIAL
target_link_libraries(CMAKE_TUTORIAL modules)
```

CMakeLists.txt 文件讲解：

1. **find_package(OpenCV REQUIRED)**：查找并引入 OpenCV 库，确保在编译时链接 OpenCV。
2. **add_subdirectory(modules)**：引入 `modules` 目录中的自定义库，并链接到主程序。
3. **add_compile_definitions(PROJECT_DIR="${PROJECT_SOURCE_DIR}")**：定义一个宏，将项目根目录的路径传递给代码，方便程序中使用。

---

4. 自定义模块的配置

自定义模块 `modules` 是项目中的重要组成部分，它展示了如何将自定义代码模块化，并通过 CMake 管理多个源文件。模块目录中的 `CMakeLists.txt` 负责配置 `cv` 子目录的源代码并生成共享库。

modules/CMakeLists.txt 文件

```
# 设置 C++ 标准
set(CMAKE_CXX_STANDARD 17)

# 查找 OpenCV库
find_package(OpenCV REQUIRED)

# 包含 OpenCV 的头文件
include_directories(${OpenCV_INCLUDE_DIRS})

# 输出 OpenCV 版本信息，便于调试
message("OpenCV version is ${OpenCV_VERSION}")

# 自动查找 cv 目录下的源文件并将其添加到变量 cv_src 中
aux_source_directory(cv cv_src)

# 创建共享库 modules，将 cv 目录中的源文件编译为共享库
add_library(modules SHARED ${cv_src})
```

```
# 链接 OpenCV库到 modules 共享库
target_link_libraries(modules
    ${OpenCV_LIBS}
)
```

CMakeLists.txt 文件讲解：

1. **aux_source_directory(cv cv_src)**：自动查找 `cv` 目录中的源文件，并将其包含进来构建为共享库。
2. **add_library(modules SHARED ${cv_src})**：将 `cv_src` 中的源文件编译为一个共享库 `modules`，供主程序使用。

---

5. 主程序代码实现

主程序 `main.cpp` 通过 OpenCV 捕获视频或摄像头图像，并通过自定义模块处理图像。

```
#include <opencv2/opencv.hpp>
#include <iostream>

// 从自定义模块中包含头文件
#include "modules/cv/cv.hpp"

int main()
{
    // 打开视频文件或摄像头
    cv::VideoCapture cap("/path/to/kun.mp4");
    if (!cap.isOpened())
    {
        std::cerr << "无法打开视频文件" << std::endl;
        return -1;
    }

    cv::Mat frame;
    while (true)
```

```
    {
        cap >> frame;  // 读取视频帧
        if (frame.empty())
        {
            std::cerr << "视频结束或无法读取帧" << std::endl;
            break;
        }

        // 在此调用自定义模块中的图像处理函数
        // 如: CMAKE_TUTORIAL::CV::processFrame(frame);

        // 显示图像
        cv::imshow("视频帧", frame);
        if (cv::waitKey(30) == 'q')
        {
            break;
        }
    }

    return 0;
}
```

---

6. 构建与运行

1. **构建项目**
   在项目根目录中，创建 `build` 目录并运行 CMake：

   ```
   mkdir build
   cd build
   cmake ..
   ```

2. **编译项目**
   运行 `make` 命令进行编译：

   ```
   make
   ```

3. **运行程序**
   编译完成后运行可执行文件：

   ```
   ./CMAKE_TUTORIAL
   ```

---

7. 总结与反思

通过这集的学习，您已经了解了如何使用 CMake 集成 OpenCV 和自定义模块。本集的重点在于复习之前学过的内容，并展示如何将多个模块和第三方库结合起来进行编译和链接。

CMake 是一个非常强大且灵活的构建系统，未来在大型项目中您可以使用它来处理复杂的依赖关系和跨平台构建任务。

作业

1. 将项目扩展，尝试添加更多的自定义模块，例如增加一个滤镜模块来处理视频帧。
2. 探索如何通过 CMake 集成其他外部库，如 Boost 或 Qt，来扩展项目的功能。