SDL + FFmpeg 播放 mp4

PB20061372 朱云沁 Dec. 8, 2023

1 实验目的

1. 学习 SDL 编程的流程.
2. 了解 FFmpeg 工具包作用.
3. 学习 FFmpeg 工具包中命令行的使用.
4. 学习基于 FFmpeg 库编程的流程.

2 实验原理

2.1 SDL

SDL (Simple DirectMedia Layer) 是一套开放源代码的跨平台多媒体开发库, 使用 C 语言写成, 提供了数种控制图像、声音、输入的函数.

SDL2 提供了 Basics、Video、Input Events、Force Feedback、Audio、Threads、Timers、File Abstraction 、 Shared Object Support 、 Platform and CPU Information 、 Power Management、Additional Functionality 等不同类别的函数.

2.2 FFmpeg

FFmpeg 是一个开源的跨平台的音视频处理工具, 可以用来录制、转换、剪辑、合成、流媒体等多种操作. FFmpeg 的核心是 libavcodec 库, 它包含了众多的音视频编解码器, 支持多种格式和协议. FFmpeg 还提供了一些命令行工具, 如 ffmpeg、ffplay、ffprobe 等; 以及一些开发库, 如 libavformat、libavfilter、libswscale、libswresample 等, 供其他程序使用.

3 实验内容及结果

3.1 环境配置

实验在 M1 芯片 macOS 下进行, 使用 vcpkg 作为包管理器, CMake 作为构建工具, 采用 C++20 标准. Apple Clang 版本为 15.0.0, SDL2 版本为 2.28.5, FFmpeg 版本为 6.1.

具体配置步骤如下:

1. 安装 vcpkg: git clone https://github.com/microsoft/vcpkg && ./vcpkg/bootstrap-vcpkg.sh
2. 安装 SDL2 和 FFmpeg: ./vcpkg/vcpkg install sdl2 ffmpeg
3. 编写 CMakeLists.txt: 详见代码. 其中, SDL2 和 FFmpeg 的链接方式分别为
   - find_package(SDL2 CONFIG REQUIRED)
     target_link_libraries(main PRIVATE SDL2::SDL2)
   - find_package(FFMPEG REQUIRED)
     target_include_directories(main PRIVATE $FFMPEG_INCLUDE_DIRS) < br/ >
     $target_link_directories(main PRIVATE {FFMPEG_LIBRARY_DIRS})$
     target_link_libraries(main PRIVATE ${FFMPEG_LIBRARIES})

   其中 main 为可执行文件名. 构建时, 指定参数 CMAKE_TOOLCHAIN_FILE 为 [path to vcpkg]/scripts/buildsystems/vcpkg.cmake, 其中 [path to vcpkg] 为 vcpkg 的路径.

3.2 测试代码

编译运行 SDL 和 FFmpeg 测试代码 test_sdl.cpp 和 test_sdl.cpp, 结果如下:

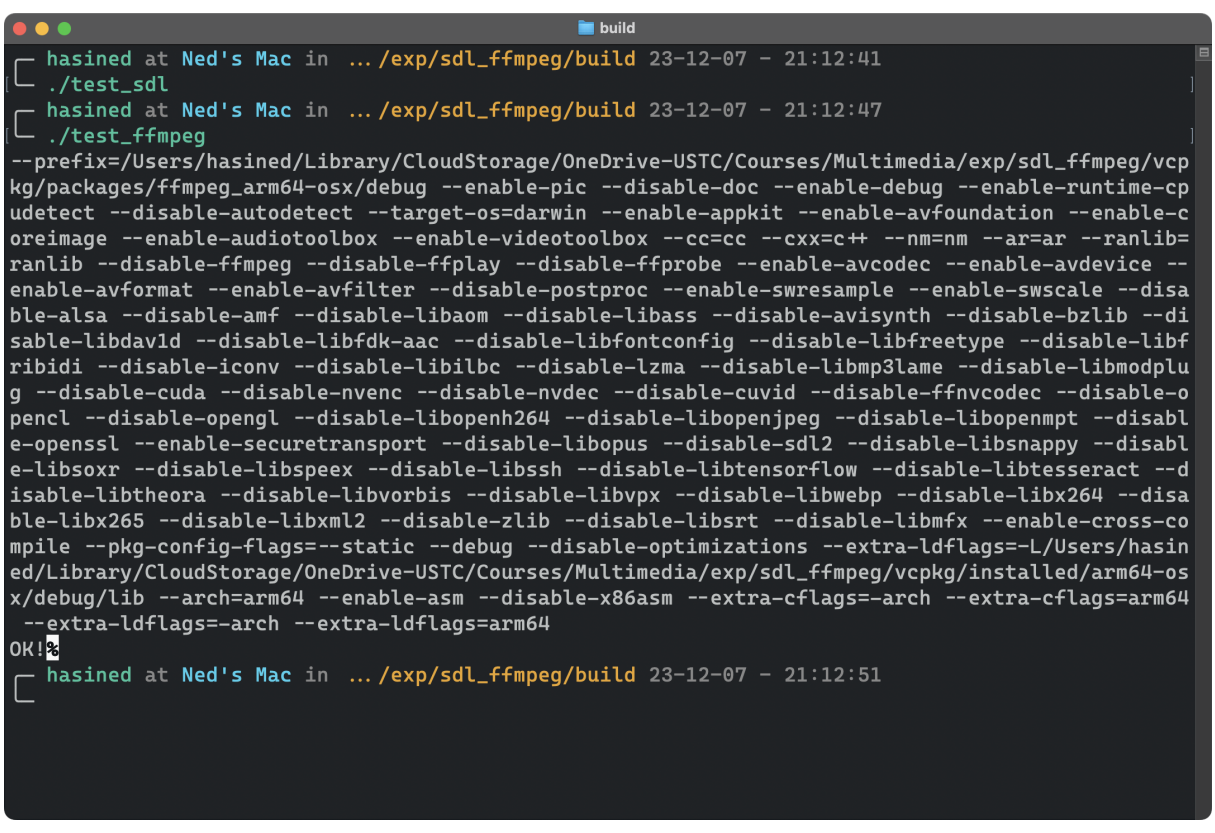

Figure 1. SDL 和 FFmpeg 测试代码运行结果.

输出正常, 说明环境配置成功.

3.3 调试示例代码

调试发现代码内存释放存在错误步骤, 将 avcodec_close、avcodec_parameters_free 替换为 avcodec_free_context 后正常运行. 对窗口标题、尺寸做了额外修改, 并集成了思考题提及的扩展功能, 最终运行结果如下:

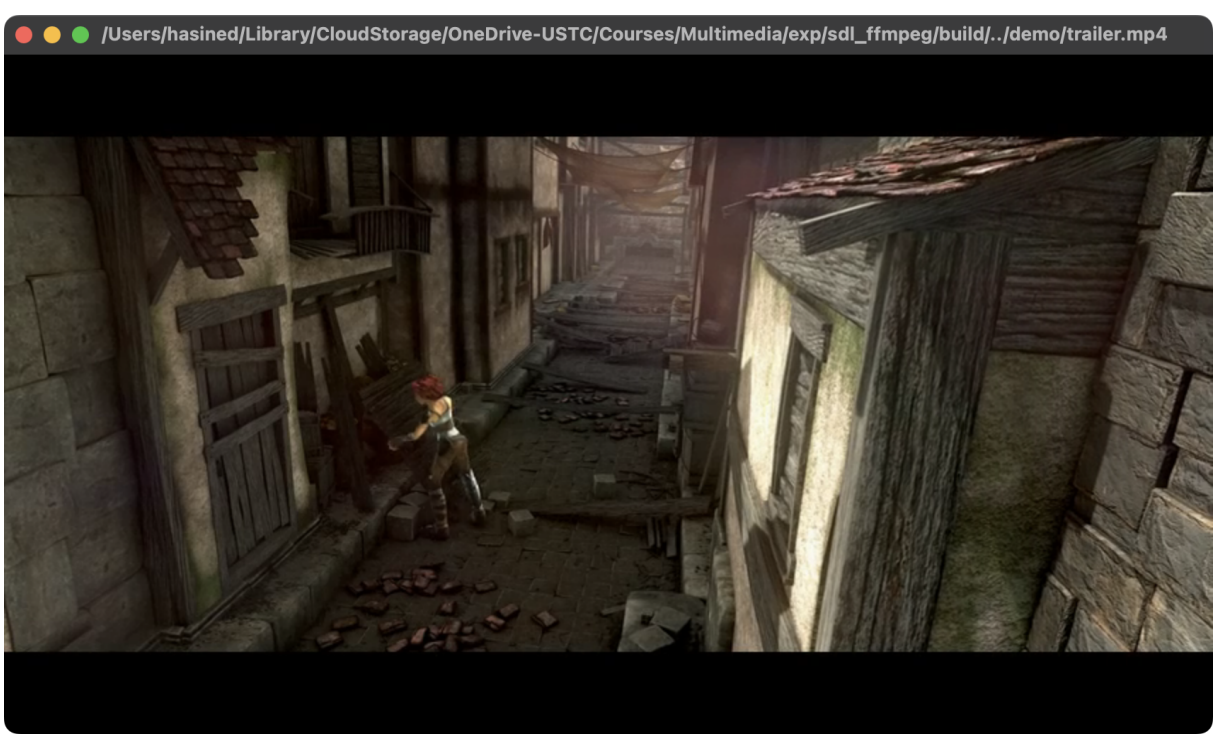

Figure 2. 示例代码运行结果.

程序能正确解码视频流, 并将每一帧渲染到窗口上.

3.4 完善示例代码

添加空格键暂停/播放功能, 具体步骤为:

1. 初始化 SDL 时, 增加 SDL_INIT_EVENTS 标志, 以支持事件处理.
2. 主循环中, 对 SDL_PollEvent 的返回值进行判断, 如果是 SDL_KEYDOWN 事件, 则判断按下的键是否为 SDLK_SPACE. 如果是, 则切换播放状态.
3. 主循环中, 若当前为暂停状态, 则跳过解码和渲染步骤.

实现较简单, 代码略.

4 思考题

1. 查阅 SDL 文档, 说明 SDL_CreateWindow 的作用是什么?

   用于创建特定标题、位置、尺寸和标志的窗口, 返回其 SDL_Window 类型的指针. 其原型为 SDL_Window *SDL_CreateWindow(const char *title, int x, int y, int w, int h, Uint32 flags), 其中, title 为窗口标题, x 和 y 为窗口左上角的坐标 (也可以是 SDL_WINDOWPOS_CENTERED 或 SDL_WINDOWPOS_UNDEFINED, 分别表示居中和未定义), w 和 h 为窗口的宽度和高度, flags 标志窗口的属性, 如是否全屏 (SDL_WINDOW_FULLSCREEN)、是否隐藏 (SDL_WINDOW_HIDDEN)、是否最大化 (SDL_WINDOW_MAXIMIZED)、是否可调整大小 (SDL_WINDOW_RESIZABLE) 等.

   该函数通常用于创建游戏或者 GUI 应用程序的主窗口.

2. 示例程序只能播放连续的图像得到, 不能播放声音. 自行查询资料得到, 如果在播放连续图像的同时还播放声音得到, 需要做哪些扩展?

   (1) 基于视频帧率控制播放速度: 根据视频流的 time_base 计算帧间间隔. 在每帧渲染完后, 计算该帧剩余持续时间, 通过 SDL_Delay 添加延迟, 以调整帧率.

   (2) 读取音频流信息, 配置音频解码器: 类比视频流的处理, 对 AVFormatContext 调用 av_find_best_stream 以获取音频类型的流索引, 然后通过 avcodec_find_decoder、avcodec_alloc_context3、avcodec_open2 等函数初始化音频解码器.

   (3) 配置 SDL 音频设备: 使用 SDL_AudioSpec 设置音频参数, 尤其是采样率、声道数、采样格式等, 然后调用 SDL_OpenAudioDevice 函数打开音频设备.

   (4) 同步地解码和播放音频: 在主循环中, 判断读取到的数据包是否为音频流. 如果是, 则类似于视频帧的处理, 调用 avcodec_send_packet、avcodec_receive_frame 以解码音频帧. 然后, 使用 swr_convert 将解码后的音频帧转换为所需格式, 使用 SDL_QueueAudio 推送到 SDL 音频缓冲区.

   (5) 释放资源: 在退出主循环后, 调用 avcodec_free_context、avformat_close_input、swr_free、SDL_CloseAudioDevice 等函数释放资源.

   详见代码.