TUMUSIIME FRANCIS 2020/BSE/066/PS

Manual for Video Decoding and Rendering Library (For Developers)

Introduction

This library provides functionality to decode and render video files using hardware acceleration. It leverages FFmpeg for decoding and SDL for rendering, ensuring efficient video playback. It's developed for developers seeking to incorporate high-performance video playback capabilities into their applications.

Prerequisites

- **FFmpeg:** Ensure FFmpeg is installed on your system.
- **SDL:** Install SDL2 development libraries.

Integration Process:

1. **Placing the Header:** Ensure the "VideoDecoder.h" header file is located in a directory accessible to your compiler, often within your project's include path.
2. **Adding the Directive:** At the beginning of your C++ file, insert the line `#include "VideoDecoder.h"` to incorporate the header's contents.
3. **Using the Library:** Once included, you can create `VideoDecoder` objects, call its methods, and access its public members to leverage the library's features effectively.

Note: Remember to link your project against the library's implementation files (e.g., "VideoDecoder.cpp") and any external libraries it depends on (FFmpeg and SDL2) during the compilation process.

The header file contains;

i. **Class Definitions:** The "VideoDecoder.h" header primarily defines the `VideoDecoder` class, which serves as the core interface for interacting with the library's decoding and rendering capabilities.

ii. **Public Members:** It outlines the public members of the `VideoDecoder` class, including its constructor, methods like `Initialize()` and `Play()`, and any relevant properties you can directly access.

API Documentation for VideoDecorder.h

API Documentation for VideoDecoder.h

Classes:

VideoDecoder

Public Members:

- **Constructor:**
  - ❖ `VideoDecoder(const std::string& path)`:
    - ▪ Creates a VideoDecoder object for decoding and rendering the video file at the specified path.
    - ▪ Throws a `std::runtime_error` exception for errors, such as file opening failures.

- **Methods:**
  - ❖ `void Initialize()`:
    - ▪ Initializes the decoder and prepares for playback.
    - ▪ Opens the video file, finds the video stream, opens the hardware decoder, and creates SDL window and renderer.
    - ▪ Throws `std::runtime_error` exceptions for errors during initialization.

  - ❖ `void Play()`:
    - ▪ Starts decoding and rendering the video.

    - Loops through video frames, decoding them and rendering them to the SDL window.
    - Handles frame rate calculation and delays for smooth playback.
    - Exits gracefully when an SDL_QUIT event is received.

- `int GetVideoStreamIndex() const`:
    - Returns the index of the video stream within the container.
    - Useful for accessing video stream metadata.

Error Handling:

- Throws `std::runtime_error` exceptions with informative error messages for various errors.
- Users should employ try-catch blocks to handle these exceptions gracefully.

Implementation of the Library

I'll now generate a comprehensive API documentation based on the provided code:

API Documentation for VideoDecoder Library

Classes

VideoDecoder

Public Members:

- **Constructor:**
    - `VideoDecoder(const std::string& path)`:
        - Creates a VideoDecoder object for decoding and rendering the video file at the specified path.

        - Throws a `std::runtime_error` exception for errors, such as file opening failures.

- **Methods:**
    - `void Initialize()`:
        - Initializes the decoder and prepares for playback.
        - Opens the video file, finds the video stream, opens the hardware decoder, and creates SDL window and renderer.
        - Throws `std::runtime_error` exceptions for errors during initialization.
    - `void Play()`:
        - Starts decoding and rendering the video.
        - Loops through video frames, decoding them and rendering them to the SDL window.
        - Handles frame rate calculation and delays for smooth playback.
        - Exits gracefully when an SDL_QUIT event is received.
    - `int GetVideoStreamIndex() const`:
        - Returns the index of the video stream within the container.
        - Useful for accessing video stream metadata.

Error Handling:

- Throws `std::runtime_error` exceptions with informative messages for various errors.
- Users should handle exceptions in their code using try-catch blocks.

Dependencies:

- FFmpeg library (for video decoding)
- SDL2 library (for rendering)

Header File (VideoDecoder.h):

- Contains the class declaration and public members.

Implementation File (VideoDecoder.cpp):

- Contains the implementation of the class methods and private members.

Private Members:

- **OpenInputFile():** Opens the video file using FFmpeg.
- **FindVideoStream():** Locates the video stream within the container.
- **OpenCodec():** Opens the hardware decoder for the video stream.
- **CreateSDLWindow():** Creates an SDL window for video display.
- **CreateSDLRenderer():** Creates an SDL renderer for rendering frames.
- **CreateSDLTexture():** Creates an SDL texture to hold decoded video frames.
- **PrintVideoMetadata() const:** Prints information about the video stream (codec, dimensions, bitrate).
- **DecodeAndRenderFrames():** Main loop for decoding and rendering frames.

Additional Notes:

- Ensure SDL is initialized before creating a VideoDecoder object.
- Handle exceptions gracefully in your code.
- See the provided example for usage guidelines.

The methods implementation for the library functionality are not included in the header file but instead put in the VideoDecorder.cpp because of the following;

Encapsulation and Information Hiding:

- The functions are private member functions of the `VideoDecoder` class, designed to be internal implementation details.
- Exposing them in the header would make them part of the public API, potentially complicating usage and hindering future changes.

Separation of Interface and Implementation:

- The header file serves as the interface, defining the public members that users interact with.

- The .cpp file contains the implementation details, keeping those private to maintain a clear separation.

Modularity and Compilation Efficiency:

- Separating interface and implementation allows for independent compilation of different parts of the code.
- Changes to the internal implementation in the .cpp file don't require recompilation of code that only uses the public API.

Preventing Unintended Usage:

- Keeping those functions private ensures they are not mistakenly used directly by users, which could lead to incorrect usage or unexpected behavior.
- It promotes a clear and controlled way of interacting with the library through its public interface.

Potential for Future Changes:

- Encapsulation allows the flexibility to modify internal implementation without affecting the public API or breaking user code.
- This enables the library to evolve and improve without disrupting existing integrations.

Key Features

- Hardware-accelerated decoding using the h264_qsv decoder
- SDL integration for video rendering
- Automatic aspect ratio adjustment for proper display
- Frame rate-based playback for smooth viewing

Example Code (C++)

```
#include "VideoDecoder.h"

#include <iostream>

int main() {

    try {

        // Initialize SDL with video capabilities

        if (SDL_Init(SDL_INIT_VIDEO) < 0) {

            throw std::runtime_error("SDL initialization failed: " +
std::string(SDL_GetError()));

        }

        // Create a VideoDecoder object for the desired video file

        VideoDecoder videoDecoder("path/to/your/video.mp4");

        // Initialize the decoder

        videoDecoder.Initialize();

        // Play the video

        videoDecoder.Play();

    } catch (const std::exception& e) {

        std::cerr << "Error: " << e.what() << std::endl;

    }

    // Clean up SDL resources

    SDL_Quit();

    return 0;

}
```

Key Points:

- **Clear SDL Initialization:** Explicitly initializes SDL with SDL_Init(SDL_INIT_VIDEO) for video features.
- **Error Handling:** Uses a try-catch block to handle potential exceptions during SDL initialization or video decoding/rendering.
- **Informative Error Messages:** Provides descriptive error messages in case of exceptions, aiding in troubleshooting.
- **Graceful SDL Cleanup:** Ensures SDL is properly cleaned up with SDL_Quit() even if exceptions occur.
- **Concise Code:** Maintains clarity and brevity for easy understanding.

Note:

- Replace "path/to/your/video.mp4" with the actual path to your video file.
- Link your project against the FFmpeg and SDL2 libraries.