工程春招题学习文档

LEVEL 0-1 git

git建库与远程仓库设置

Git简易教程

基础设置：

```
git config --global user.name "你的名字"
git config --global user.email "你的邮箱"
```

注释：

1. git config是一个git命令，用来配置git的各种设置。
2. --globe代表该配置将被应用到全局范围内而不是当前仓库
3. 这条命令的含义是设置全局git配置，将我的名字和我的邮箱与我在git上注册的用户联系起来，方便在提交代码是标识作者身份。

初始化仓库：

要开始使用Git追踪某个项目，你需要初始化一个仓库(repository)。在项目的根目录下运行：

```
git init
```

添加文件：

将文件添加到仓库：

```
git add <文件名>
```

或者添加当前目录的所有文件：

```
git add .
```

提交更改：

提交更改，并写上提交信息：

```
git commit -m "提交信息"
```

查看状态：

查看仓库的状态：

```
git status
```

查看更改：

查看具体更改内容：

```
git diff
```

切换分支：

创建新分支：

```
git branch <分支名>
```

切换到某个分支：

```
git checkout <分支名>
```

合并分支：

将分支的更改合并到当前分支：

```
git merge <分支名>
```

拉取 & 推送：

从远程仓库拉取最新更改：

```
git pull <远程仓库地址>
```

将本地更改推送到远程仓库：

```
git push <远程仓库地址> <分支名>
```

安装Git步骤

1. 打开终端。
2. 更新包索引：

```
sudo apt update
```

3. 安装Git：

```
sudo apt install git
```

4. 验证安装：

```
git --version
```

远程仓库创建

1. **注册/登录账号**如果没有GitHub账号，需要先去GitHub官网注册一个。如果已经有账号了，直接登录即可。
2. **创建新仓库**登录后，在右上角头像旁边会有一个"+"号，点击它，然后选择"New repository"。
3. **填写仓库信息**在新页面中，填写你的仓库名称，可以选择公开或私有仓库，还可以添加一个README文件、.gitignore文件以及选择许可证。

本次任务较为简单不涉及商业等用途故未设置

4. **创建仓库**填写完信息后，点击页面底部的"Create repository"按钮，你的远程仓库就创建成功了。
5. **将本地仓库与远程仓库关联**如果你已经有了本地Git仓库，你可以通过以下命令将其与远程仓库关联：

```
git remote add 【远程仓库名字】
https://github.com/your_username/your_repository.git
```

这里的 `https://github.com/your_username/your_repository.git` 需要替换成你的GitHub仓库的实际URL。

6. **推送本地仓库到远程**如果你的本地仓库中已经有了代码，并且你想推送它到新的远程仓库，你可以使用以下命令：

```
git push -u origin master
```

这里的 `master` 可能需要替换为你的默认分支名，例如 `main` 或者你所使用的其他名字。

LEVEL 0-2 Parsing parameters

题目分析

对这个任务分析后可知作为整体任务的前置任务，我们需要设计一个类似目录的代码文件，其中目前需要的功能有两个：

1. 显示版本号
2. 展示帮助文档

解决思路与注意事项

面对这个问题我选择使用C语言进行处理（为了和后面的板块进行对接）

为了完成基本的管理功能，可以选择编写一个简单的命令行解析器，目前在这个解析器中需要考虑以下参数：

"-h" 即为--help：用来显示帮助文档

“-v” 即为--version：用来输出版本信息

代码文件

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

void show_help() {
    // 显示帮助信息
    printf("Usage: ./player [OPTIONS]\n");
    printf("Options:\n");
    printf("  -h        Display this help message and open document\n");
    printf("  -v        Output version information\n");

    // 使用系统调用打开指定位置的文档
    if (access("/home/tom/example.txt", F_OK) != -1) {
        system("xdg-open /home/tom/example.txt");
    }
    else {
        printf("无法打开文档。\n");
    }
}

void show_version() {
    // 显示版本信息
    printf("版本信息：Dian团队工程开发测试\n");
}

int main(int argc, char *argv[]) {
    if (argc == 1) {
        // 如果未提供任何命令行参数，则提示使用方法
        printf("Missing options. Use -h for usage information.\n");
    } else {
        // 解析命令行参数
        if (strcmp(argv[1], "-h") == 0) {
            show_help();
        } else if (strcmp(argv[1], "-v") == 0) {
            show_version();
        } else {
            // 如果提供了未知选项，则提示使用方法
            printf("Unknown option. Use -h for usage information.\n");
        }
    }

    return 0;
}
```

代码每一部分的作用请见代码块,注解都是为了解释代码加入的，在实际运行使用的vedio.py文件中我未标记注解

打开方式：

```
gcc player.c -o player
./player -h
./player -v
```

LEVEL 0-3 Video decoder

任务难点解析

1. 理解清楚什么是静态库，静态库是怎么制作的，里面都包含了什么东西
2. 学习如何使用静态库，如何正确使用头文件
3. 理解什么是动态库
4. 了解怎么将主文件和库联系在一起进行编译
5. **正确编写主函数**

解决思路

1. 首先认真学习了解库的相关知识(非常重要，切不可急躁)
2. 阅读指导文件，了解学习每个函数的作用，知道每个函数需要输入什么，需要输出什么
3. 编写C代码并且反复不断的修正错误

代码与解析

```
#include "video_decoder.h"  // 包含视频解码器的头文件

#include <stdio.h>  // 包含标准输入输出库的头文件
#include <stdbool.h>  // 包含标准布尔类型的头文件

int main(int argc, char *argv[]) {  // 主函数开始

    // 检查命令行参数个数是否足够
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <video_file>\n", argv[0]);  // 打印用法信息到标准错误流
        return -1;  // 返回错误码
    }
```

```
13
14      const char *filename = argv[1];  // 从命令行参数获取视频文件名
15
16      // 尝试初始化视频解码器，若失败则输出错误信息并返回
17      if (decoder_init(filename) != 0) {
18          fprintf(stderr, "Failed to initialize decoder with file %s\n",
    filename);
19          return -1;
20      }
21
22      double fps = get_fps();  // 获取视频的帧率
23      int delay = (int)(1000.0 / fps); // 计算每一帧的播放延迟，以毫秒为单位
24
25      bool running = true;  // 定义一个布尔变量，表示程序是否继续运行
26
27      // 循环读取视频帧，直到没有更多帧可读取或者读取失败
28      while (running) {
29          Frame frame = decoder_get_frame();  // 从视频解码器中获取一帧视频数据
30
31          // 假设帧的data为NULL代表没有更多帧可读取或者读取失败
32          if (frame.data == NULL) {
33              running = false;  // 将运行标志设置为false，表示不再继续运行循环
34              break;  // 退出循环
35          }
36          // 在这里可以对帧数据进行处理，比如显示到屏幕上等
37
38      }
39
40      decoder_close(); // 关闭视频解码器，释放资源
41      return 0;  // 返回正常退出码
42  }
```

LEVEL 1-1 Image print

题目分析与简单思路总结

任务要求如下：

1. **读取视频文件逐帧进内存**：要实现这个功能，需要使用静态库提供的接口，包括一个能够初始化视频文件的函数、一个读取下一帧的函数、以及一个关闭解码器的函数。我认为可以使用以下函数

   - `decoder_init` 初始化视频文件。
   - `decoder_get_frame` 读取下一帧。

- `decoder_close` 关闭解码器和释放资源。

2. **使用ANSI转义代码，将逐帧转换的字符画显示出来**：为了在终端中显示字符画，需要将获取的帧数据转换成字符，并使用ANSI转义代码来控制字符的颜色和位置。计算灰度值经查询得到RGB图像有3个通道，也就是一个3维的矩阵，灰度图只有一个通道。转换公式简单来说，就是把RGB3个通道的分量按照一定的比例计算到灰度图像中。公式如图
   Gray = R*0.2989* + *G*0.587 + B*0.114*

3. **显示彩色字符**：要显示彩色字符，你需要将帧的RGB值映射到终端能够显示的颜色上，并使用ANSI转义代码来设置字符颜色。

实现目标的C语言代码

成功打印一帧灰度图的代码

```c
#include "video_decoder.h"
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>

// ANSI 转义字符，用于设置终端颜色
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_GRAY    "\x1b[90m"

int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <video_file>\n", argv[0]);
        return -1;
    }

    const char *filename = argv[1]; // 从命令行参数获取文件名
    if (decoder_init(filename) != 0) {
        fprintf(stderr, "Failed to initialize decoder with file %s\n",
filename);
        return -1;
    }

    double fps = get_fps();
    int delay = (int)(1000000.0 / fps); // 计算每帧之间的延迟，单位微秒

    bool running = true;
    while (running) {
```

```
        Frame frame = decoder_get_frame();
        if (frame.data == NULL) { // 假设data为NULL代表没有更多帧或获取失败
            running = false;
            break;
        }

        // 打印灰度图
        printf(ANSI_COLOR_GRAY); // 设置终端颜色为灰色
        for (int y = 0; y < frame.height; ++y) {
            for (int x = 0; x < frame.width; ++x) {
                // 计算灰度值
                int index = y * frame.linesize + x * 3;
                unsigned char r = frame.data[index];
                unsigned char g = frame.data[index + 1];
                unsigned char b = frame.data[index + 2];
                double gray = 0.299 * r + 0.587 * g + 0.114 * b;
                printf("%c", gray < 128 ? '.' : '*'); // 根据灰度值打印不同字符
            }
            printf("\n");
        }
        printf(ANSI_COLOR_RESET); // 重置终端颜色
        running = false;
    }

    decoder_close(); // 清理解码器资源
    return 0;
}

```

为了更好的体现出灰度差异，我选择使用一组不同的字符来表示不同的灰度级别。这组字符从“最暗”到“最亮”排列，用以模拟灰度的变化。

```
" .:-=+*#%@"
```

这里 `' '` (空格) 是最暗的（代表黑色），`'@'` 是最亮的（代表白色）

调整打印效率的代码

```
#include "video_decoder.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
```

```c
#include <unistd.h>

#define GRAYSCALE_CHARS " .:-=+*#%@"

void print_frame_as_grayscale(Frame frame) {
    int char_len = strlen(GRAYSCALE_CHARS) - 1;
    char *output = malloc(frame.width * frame.height + frame.height + 1);
    char *p = output;

    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];
            double gray = 0.299 * r + 0.587 * g + 0.114 * b;
            int val = (int)(char_len * gray / 255);
            *p++ = GRAYSCALE_CHARS[val];
        }
        *p++ = '\n';
    }
    *p = '\0';

    printf("%s", output);
    free(output);
}

int main(int argc, char *argv[]) {
    // 省略其他代码部分...

    bool running = true;
    while (running) {
        Frame frame = decoder_get_frame();
        if (frame.data == NULL) { // 假设data为NULL代表没有更多帧或获取失败
            running = false;
            break;
        }

        print_frame_as_grayscale(frame);
        running = false; // 只打印第一帧
    }

    decoder_close(); // 清理解码器资源
    return 0;
}
```

在这个代码中，我创建了一个 `print_frame_as_grayscale` 函数来处理灰度图的打印。这个函数首先为整个帧分配一个足够大的字符数组。然后，它遍历帧中的每个像素，根据灰度值选取相应的字符，并将这些字符拼接到字符数组中。在遍历完成后，使用单个 `printf` 调用将整个字符数组输出到终端，然后释放该数组所占用的内存。

能打印RGB图的代码

```c
#include "video_decoder.h"
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>

#define GRAYSCALE_CHARS " .:-=+*#%@"

void print_frame_as_grayscale(Frame frame) {
    int char_len = strlen(GRAYSCALE_CHARS) - 1;
    char *output = malloc(frame.width * frame.height + frame.height + 1);
    char *p = output;

    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];
            double gray = 0.299 * r + 0.587 * g + 0.114 * b;
            int val = (int)(char_len * gray / 255);
            *p++ = GRAYSCALE_CHARS[val];
        }
        *p++ = '\n';
    }
    *p = '\0';

    printf("%s", output);
    free(output);
}

void print_frame_as_rgb(Frame frame) {
    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];

```

```
39              // 使用 ANSI 转义字符来设置前景色
40              printf("\x1b[38;2;%d;%d;%dm", r, g, b);
41          }
42          printf("\x1b[0m\n"); // 重置颜色并换行
43      }
44  }
45
46  // ANSI 转义字符，用于设置终端颜色
47  #define ANSI_COLOR_RESET   "\x1b[0m"
48  #define ANSI_COLOR_GRAY    "\x1b[90m"
49
50  int main(int argc, char *argv[]) {
51      if (argc < 2) {
52          fprintf(stderr, "Usage: %s <video_file>\n", argv[0]);
53          return -1;
54      }
55
56      const char *filename = argv[1]; // 从命令行参数获取文件名
57      if (decoder_init(filename) != 0) {
58          fprintf(stderr, "Failed to initialize decoder with file %s\n",
    filename);
59          return -1;
60      }
61
62      double fps = get_fps();
63      int delay = (int)(1000000.0 / fps); // 计算每帧之间的延迟，单位微秒
64
65      bool running = true;
66      while (running) {
67          Frame frame = decoder_get_frame();
68          if (frame.data == NULL) { // 假设data为NULL代表没有更多帧或获取失败
69              running = false;
70              break;
71          }
72
73          print_frame_as_grayscale(frame);
74          print_frame_as_rgb(frame);
75          running = false;
76      }
77
78      decoder_close(); // 清理解码器资源
79      return 0;
80  }
```

添加选择的代码

```c
int main(int argc, char *argv[]) {
    bool color = false; // 增加一个布尔变量来控制是否以彩色打印

    if (argc < 2) {
        fprintf(stderr, "Usage: %s <video_file> [--color]\n", argv[0]);
        return -1;
    }

    const char *filename = argv[1]; // 从命令行参数获取文件名
    if (argc == 3 && strcmp(argv[2], "--color") == 0) {
        color = true; // 如果有 --color 参数，则设置 color 为 true
    }

    if (decoder_init(filename) != 0) {
        fprintf(stderr, "Failed to initialize decoder with file %s\n",
filename);
        return -1;
    }

    // 不再需要计算每帧之间的延迟
    // double fps = get_fps();
    // int delay = (int)(1000000.0 / fps);

    Frame frame = decoder_get_frame();
    if (frame.data != NULL) { // 如果获取到帧数据
        if (color) {
            print_frame_as_rgb(frame); // 以彩色方式打印
        } else {
            print_frame_as_grayscale(frame); // 以灰度方式打印
        }
    } else {
        fprintf(stderr, "Failed to get frame from decoder.\n");
    }

    decoder_close(); // 清理解码器资源
    return 0;
}
//请注意这是只打印一帧的代码
```

```c
#include "video_decoder.h"
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h> // 为 strlen 和 strcmp 函数
#include <stdlib.h> // 为 malloc 和 free 函数
```

```
7
8  #define GRAYSCALE_CHARS " .:-=+*#%@"
9
10 void print_frame_as_grayscale(Frame frame) {
11     int char_len = strlen(GRAYSCALE_CHARS) - 1;
12     char *output = malloc(frame.width * frame.height + frame.height + 1);
13     char *p = output;
14
15     for (int y = 0; y < frame.height; ++y) {
16         for (int x = 0; x < frame.width; ++x) {
17             int index = y * frame.linesize + x * 3;
18             unsigned char r = frame.data[index];
19             unsigned char g = frame.data[index + 1];
20             unsigned char b = frame.data[index + 2];
21             double gray = 0.299 * r + 0.587 * g + 0.114 * b;
22             int val = (int)(char_len * gray / 255);
23             *p++ = GRAYSCALE_CHARS[val];
24         }
25         *p++ = '\n';
26     }
27     *p = '\0';
28
29     printf("%s", output);
30     free(output);
31 }
32
33 void print_frame_as_rgb(Frame frame) {
34     for (int y = 0; y < frame.height; ++y) {
35         for (int x = 0; x < frame.width; ++x) {
36             int index = y * frame.linesize + x * 3;
37             unsigned char r = frame.data[index];
38             unsigned char g = frame.data[index + 1];
39             unsigned char b = frame.data[index + 2];
40
41             // 使用 ANSI 转义字符来设置前景色
42             printf("\x1b[38;2;%d;%d;%dm", r, g, b);
43         }
44         printf("\x1b[0m\n"); // 重置颜色并换行
45     }
46 }
47
48 // ANSI 转义字符，用于设置终端颜色
49 #define ANSI_COLOR_RESET   "\x1b[0m"
50 #define ANSI_COLOR_GRAY    "\x1b[90m"
51
52 int main(int argc, char *argv[]) {
53     bool color = false; // 增加一个布尔变量来控制是否以彩色打印
```

```
54
55      if (argc < 2) {
56          fprintf(stderr, "Usage: %s <video_file> [-c]\n", argv[0]);
57          return -1;
58      }
59
60      const char *filename = argv[1]; // 从命令行参数获取文件名
61      if (argc == 3 && strcmp(argv[2], "-c") == 0) {
62          color = true; // 如果有 -c 参数，则设置 color 为 true
63      }
64
65      if (decoder_init(filename) != 0) {
66          fprintf(stderr, "Failed to initialize decoder with file %s\n",
    filename);
67          return -1;
68      }
69
70      double fps = get_fps();
71      int delay = (int)(1000000.0 / fps); // 计算每帧之间的延迟，单位微秒
72
73      bool running = true;
74      while (running) {
75          Frame frame = decoder_get_frame();
76          if (frame.data == NULL) { // 假设data为NULL代表没有更多帧或获取失败
77              running = false;
78              break;
79          }
80
81          if (color) {
82              print_frame_as_rgb(frame);
83          } else {
84              print_frame_as_grayscale(frame);
85          }
86
87          usleep(delay); // 根据视频的 FPS 等待一定的时间
88      }
89
90      decoder_close(); // 清理解码器资源
91      return 0;
92  }
93  //请注意这是连续打印帧的代码
```

编译程序格式

```
1  gcc main.c -o main -L. -lvideodecoder -lavformat -lavcodec -lavutil -lswscale
```

```
./main /home/tom/Reference_video/dragon.mp4 //打印灰色图
./main /home/tom/Reference_video/dragon.mp4 -c //打印彩色图
```

LEVEL 1-2 Downsample

任务分析

本任务需要完成的内容不多，主要强调了学习并使用新的东西的方法

这一项主要的需求就是要理解并学着编写average pooling &max pooling的原理。即为平均池和最大池。

代码展示

resize函数代码

```
// 平均池化 resize 函数
Frame resize_with_average_pooling(Frame frame, int target_width, int target_height) {
    Frame resized;
    resized.width = target_width;
    resized.height = target_height;
    resized.linesize = target_width * 3; // 假设RGB格式
    resized.data = malloc(resized.linesize * resized.height);

    int horizontal_step = frame.width / target_width;
    int vertical_step = frame.height / target_height;

    for (int y = 0; y < target_height; y++) {
        for (int x = 0; x < target_width; x++) {
            long sum_r = 0, sum_g = 0, sum_b = 0;
            for (int dy = 0; dy < vertical_step; dy++) {
                for (int dx = 0; dx < horizontal_step; dx++) {
                    int index = ((y * vertical_step + dy) * frame.linesize + (x * horizontal_step + dx) * 3);
                    sum_r += frame.data[index];
                    sum_g += frame.data[index + 1];
                    sum_b += frame.data[index + 2];
                }
            }
            int count = vertical_step * horizontal_step;
            resized.data[(y * resized.linesize) + x * 3]     = sum_r / count;
```

```
            resized.data[(y * resized.linesize) + x * 3 + 1] = sum_g / count;
            resized.data[(y * resized.linesize) + x * 3 + 2] = sum_b / count;
        }
    }

    return resized;
}
```

```
// 最大池化 resize 函数
Frame resize_with_max_pooling(Frame frame, int target_width, int target_height)
 {
    Frame resized;
    resized.width = target_width;
    resized.height = target_height;
    resized.linesize = target_width * 3; // 假设RGB格式
    resized.data = malloc(resized.linesize * resized.height);

    int horizontal_step = frame.width / target_width;
    int vertical_step = frame.height / target_height;

    for (int y = 0; y < target_height; y++) {
        for (int x = 0; x < target_width; x++) {
            unsigned char max_r = 0, max_g = 0, max_b = 0;
            for (int dy = 0; dy < vertical_step; dy++) {
                for (int dx = 0; dx < horizontal_step; dx++) {
                    int index = ((y * vertical_step + dy) * frame.linesize +
(x * horizontal_step + dx) * 3);
                    if (frame.data[index] > max_r) max_r = frame.data[index];
                    if (frame.data[index + 1] > max_g) max_g =
frame.data[index + 1];
                    if (frame.data[index + 2] > max_b) max_b =
frame.data[index + 2];
                }
            }
            resized.data[(y * resized.linesize) + x * 3]     = max_r;
            resized.data[(y * resized.linesize) + x * 3 + 1] = max_g;
            resized.data[(y * resized.linesize) + x * 3 + 2] = max_b;
        }
    }

    return resized;
}

```

总代码：

```c
#include "video_decoder.h"
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define GRAYSCALE_CHARS " .:-=+*#%@"

void print_frame_as_grayscale(Frame frame) {
    int char_len = strlen(GRAYSCALE_CHARS) - 1;
    char *output = malloc(frame.width * frame.height + frame.height + 1);
    char *p = output;

    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];
            double gray = 0.299 * r + 0.587 * g + 0.114 * b;
            int val = (int)(char_len * gray / 255);
            *p++ = GRAYSCALE_CHARS[val];
        }
        *p++ = '\n';
    }
    *p = '\0';

    printf("%s", output);
    free(output);
}

void print_frame_as_rgb(Frame frame) {
    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];

            // 使用 ANSI 转义字符来设置前景色
            printf("\x1b[38;2;%d;%d;%dm", r, g, b);
        }
        printf("\x1b[0m\n"); // 重置颜色并换行
```

```
    }
}

// ANSI 转义字符，用于设置终端颜色
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_GRAY    "\x1b[90m"

// 平均池化 resize 函数的原型声明
Frame resize_with_average_pooling(Frame frame, int target_width, int target_height);

// print_frame_as_grayscale 和 print_frame_as_rgb 函数定义

int main(int argc, char *argv[]) {
    bool color = false;
    int target_width = 0;
    int target_height = 0;

    if (argc < 4 || argc > 5) {
        fprintf(stderr, "Usage: %s <video_file> [-c] <target_width> <target_height>\n", argv[0]);
        return -1;
    }

    const char *filename = argv[1];
    if (argc >= 4 && strcmp(argv[2], "-c") == 0) {
        color = true;
        target_width = atoi(argv[3]);
        target_height = atoi(argv[4]);
    } else {
        target_width = atoi(argv[2]);
        target_height = atoi(argv[3]);
    }

    if (decoder_init(filename) != 0) {
        fprintf(stderr, "Failed to initialize decoder with file %s\n", filename);
        return -1;
    }

    double fps = get_fps();
    int delay = (int)(1000000.0 / fps);

    bool running = true;
    while (running) {
        Frame frame = decoder_get_frame();
        if (frame.data == NULL) {
```

```
            running = false;
            break;
        }

        // 调整大小
        Frame resized_frame = resize_with_average_pooling(frame, target_width, target_height);

        if (color) {
            print_frame_as_rgb(resized_frame);
        } else {
            print_frame_as_grayscale(resized_frame);
        }

        usleep(delay);
        free(resized_frame.data); // 释放调整大小后的帧数据
        running = false; // 只打印一帧
    }

    decoder_close();
    return 0;
}

// 平均池化 resize 函数的实现
Frame resize_with_average_pooling(Frame frame, int target_width, int target_height) {
    Frame resized;
    resized.width = target_width;
    resized.height = target_height;
    resized.linesize = target_width * 3; // 假设RGB格式
    resized.data = malloc(resized.linesize * resized.height);

    int horizontal_step = frame.width / target_width;
    int vertical_step = frame.height / target_height;

    for (int y = 0; y < target_height; y++) {
        for (int x = 0; x < target_width; x++) {
            long sum_r = 0, sum_g = 0, sum_b = 0;
            for (int dy = 0; dy < vertical_step; dy++) {
                for (int dx = 0; dx < horizontal_step; dx++) {
                    int index = ((y * vertical_step + dy) * frame.linesize + (x * horizontal_step + dx) * 3);
                    sum_r += frame.data[index];
                    sum_g += frame.data[index + 1];
                    sum_b += frame.data[index + 2];
                }
            }
```

```
            int count = vertical_step * horizontal_step;
            resized.data[(y * resized.linesize) + x * 3]     = sum_r / count;
            resized.data[(y * resized.linesize) + x * 3 + 1] = sum_g / count;
            resized.data[(y * resized.linesize) + x * 3 + 2] = sum_b / count;
        }
    }

    return resized;
}
```

编译程序格式：

```
gcc main.c -o main -L. -lvideodecoder -lavformat -lavcodec -lavutil -lswscale
./main /home/tom/Reference_video/dragon.mp4 80 40//打印灰色图并选择输入视频帧的大小
./main /home/tom/Reference_video/dragon.mp4 -c 80 40//打印彩色图并选择输入视频帧的大小，具体为宽高比
```

拓展任务：

对于此部分的代码我产生了一些自己的理解并按照自己的理解编写修改了代码，接下来首先还是贴出来代码：

代码块

```
#include "video_decoder.h"
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define GRAYSCALE_CHARS " .:-=+*#%@"

void print_frame_as_grayscale(Frame frame) {
    int char_len = strlen(GRAYSCALE_CHARS) - 1;
    char *output = malloc(frame.width * frame.height + frame.height + 1);
    char *p = output;

    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
```

```
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];
            double gray = 0.299 * r + 0.587 * g + 0.114 * b;
            int val = (int)(char_len * gray / 255);
            *p++ = GRAYSCALE_CHARS[val];
        }
        *p++ = '\n';
    }
    *p = '\0';

    printf("%s", output);
    free(output);
}

void print_frame_as_rgb(Frame frame) {
    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];

            // 使用 ANSI 转义字符来设置前景色
            printf("\x1b[38;2;%d;%d;%dm", r, g, b);
        }
        printf("\x1b[0m\n"); // 重置颜色并换行
    }
}

// ANSI 转义字符，用于设置终端颜色
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_GRAY    "\x1b[90m"
Frame resize_with_average_pooling(Frame frame, int target_width, int
target_height) {
    Frame resized;
    resized.width = target_width;
    resized.height = target_height;
    resized.linesize = target_width * 3; // 假设RGB格式
    resized.data = malloc(resized.linesize * resized.height);

    int horizontal_step = frame.width / target_width;
    int vertical_step = frame.height / target_height;

    for (int y = 0; y < target_height; y++) {
        for (int x = 0; x < target_width; x++) {
            long sum_r = 0, sum_g = 0, sum_b = 0;
```

```
64              for (int dy = 0; dy < vertical_step; dy++) {
65                  for (int dx = 0; dx < horizontal_step; dx++) {
66                      int index = ((y * vertical_step + dy) * frame.linesize +
    (x * horizontal_step + dx) * 3);
67                      sum_r += frame.data[index];
68                      sum_g += frame.data[index + 1];
69                      sum_b += frame.data[index + 2];
70                  }
71              }
72              int count = vertical_step * horizontal_step;
73              resized.data[(y * resized.linesize) + x * 3]     = sum_r / count;
74              resized.data[(y * resized.linesize) + x * 3 + 1] = sum_g / count;
75              resized.data[(y * resized.linesize) + x * 3 + 2] = sum_b / count;
76          }
77      }
78
79      return resized;
80  }
81
82  void show_help() {
83      // 显示帮助信息
84      printf("Usage: ./player [OPTIONS]\n");
85      printf("Options:\n");
86      printf("  -h        Display this help message and open document\n");
87      printf("  -v        Output version information\n");
88
89      // 使用系统调用打开指定位置的文档
90      if (access("/home/tom/example.txt", F_OK) != -1) {
91          system("xdg-open /home/tom/example.txt");
92      } else {
93          printf("无法打开文档。\n");
94      }
95  }
96
97  void show_version() {
98      // 显示版本信息
99      printf("版本信息: Dian团队工程开发测试\n");
100 }
101
102 int main(int argc, char *argv[]) {
103     bool color = false;
104     int target_width = 0;
105     int target_height = 0;
106     const char *filename = NULL; // 声明filename变量
107
108     if (argc == 1) {
109         // 如果未提供任何命令行参数，则提示使用方法
```

```
110         printf("Missing options. Use -h for usage information.\n");
111         return -1;
112     } else {
113         // 解析命令行参数
114         if (strcmp(argv[1], "-h") == 0) {
115             show_help();
116             return 0;
117         } else if (strcmp(argv[1], "-v") == 0) {
118             show_version();
119             return 0;
120         } else {
121             // 解析视频文件路径
122             filename = argv[1]; // 初始化filename
123             int arg_index = 2; // 声明并初始化arg_index
124 
125             if (argc >= 4 && strcmp(argv[2], "-c") == 0) {
126                 color = true;
127                 arg_index++;
128             }
129 
130             if (argc >= arg_index + 3 && strcmp(argv[arg_index], "-r") == 0) {
131                 // 提取resize参数
132                 target_width = atoi(argv[arg_index + 1]);
133                 target_height = atoi(argv[arg_index + 2]);
134             } else {
135                 // 如果提供了-r参数但未提供两个参数，则提示使用方法
136                 printf("Format error. Please enter ./player -h for help.\n");
137                 return -1;
138             }
139 
140             if (target_width <= 0 || target_height <= 0) {
141                 printf("Invalid resize parameters. Please enter positive
    integers for width and height.\n");
142                 return -1;
143             }
144 
145             if (decoder_init(filename) != 0) {
146                 fprintf(stderr, "Failed to initialize decoder with file %s\n",
    filename);
147                 return -1;
148             }
149 
150             double fps = get_fps();
151             int delay = (int)(1000000.0 / fps);
152 
153             bool running = true;
154             while (running) {
```

```
                Frame frame = decoder_get_frame();
                if (frame.data == NULL) {
                    running = false;
                    break;
                }

                // 调整大小
                Frame resized_frame = resize_with_average_pooling(frame,
    target_width, target_height);

                if (color) {
                    print_frame_as_rgb(resized_frame);
                } else {
                    print_frame_as_grayscale(resized_frame);
                }

                usleep(delay);
                free(resized_frame.data); // 释放调整大小后的帧数据
                running = false; // 只打印一帧
            }

            decoder_close();
            return 0;
        }
    }
}
```

编译程序格式：

```
gcc main.c -o main -L. -lvideodecoder -lavformat -lavcodec -lavutil -lswscale
./main /home/tom/Reference_video/dragon.mp4 -r 80 40 //打印灰色图
./main /home/tom/Reference_video/dragon.mp4 -c -r 80 40 //打印彩色图
./player -h //打开帮助文档
./player -v //查看版本号
```

代码功能解释

在这个代码中，我尝试着将命令输入行解析代码和打印图像的代码结合在一起，可以通过./player代码实现查看版本号和查看使用文档。同时也可以利用：“目标视频文件路径” -c -r num num这种代码选

择打印出灰度图还是RGB图片（有-c就是RGB，没有就是灰度图），当未输入-r或者-r后面的数字不合法时（只有一个数字或者是有负数），会提示输入./player -h查看帮助文档

LEVEL 1-3 Video player

任务分析

1. 修改循环使其连续读取并打印
2. 增加一个命令-d用来调整帧与帧之间的时间间隔
3. 使用 `system("clear");` 在打印每一帧前清除终端提高播放质量
4. 设计加入-f用来指明文件路径
5. 我额外增加了一些小东西使得参数传递可以不用考虑顺序了

代码展示

```c
#include "video_decoder.h"
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define GRAYSCALE_CHARS " .:-=+*#%@"

void print_frame_as_grayscale(Frame frame) {
    int char_len = strlen(GRAYSCALE_CHARS) - 1;
    char *output = malloc(frame.width * frame.height + frame.height + 1);
    char *p = output;

    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];
            double gray = 0.299 * r + 0.587 * g + 0.114 * b;
            int val = (int)(char_len * gray / 255);
            *p++ = GRAYSCALE_CHARS[val];
        }
        *p++ = '\n';
    }
    *p = '\0';

```

```
    printf("%s", output);
    free(output);
}

void print_frame_as_rgb(Frame frame) {
    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];

            // 使用 ANSI 转义字符来设置前景色
            printf("\x1b[38;2;%d;%d;%dm", r, g, b);
        }
        printf("\x1b[0m\n"); // 重置颜色并换行
    }
}

// ANSI 转义字符，用于设置终端颜色
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_GRAY    "\x1b[90m"
Frame resize_with_average_pooling(Frame frame, int target_width, int
target_height) {
    Frame resized;
    resized.width = target_width;
    resized.height = target_height;
    resized.linesize = target_width * 3; // 假设RGB格式
    resized.data = malloc(resized.linesize * resized.height);

    int horizontal_step = frame.width / target_width;
    int vertical_step = frame.height / target_height;

    for (int y = 0; y < target_height; y++) {
        for (int x = 0; x < target_width; x++) {
            long sum_r = 0, sum_g = 0, sum_b = 0;
            for (int dy = 0; dy < vertical_step; dy++) {
                for (int dx = 0; dx < horizontal_step; dx++) {
                    int index = ((y * vertical_step + dy) * frame.linesize +
(x * horizontal_step + dx) * 3);
                    sum_r += frame.data[index];
                    sum_g += frame.data[index + 1];
                    sum_b += frame.data[index + 2];
                }
            }
            int count = vertical_step * horizontal_step;
            resized.data[(y * resized.linesize) + x * 3]     = sum_r / count;
```

```
            resized.data[(y * resized.linesize) + x * 3 + 1] = sum_g / count;
            resized.data[(y * resized.linesize) + x * 3 + 2] = sum_b / count;
        }
    }

    return resized;
}

void show_help() {
    // 显示帮助信息
    printf("Usage: ./player [OPTIONS]\n");
    printf("Options:\n");
    printf("  -h        Display this help message and open document\n");
    printf("  -v        Output version information\n");

    // 使用系统调用打开指定位置的文档
    if (access("/home/tom/example.txt", F_OK) != -1) {
        system("xdg-open /home/tom/example.txt");
    } else {
        printf("无法打开文档。\n");
    }
}

void show_version() {
    // 显示版本信息
    printf("版本信息: Dian团队工程开发测试\n");
}

int main(int argc, char *argv[]) {
    bool color = false;
    int target_width = 0;
    int target_height = 0;
    int custom_delay = -1; // 用户可以通过命令行参数指定自定义延迟
    const char *filename = NULL; // 通过 -f 参数指定的文件名

    if (argc == 1) {
        printf("Missing options. Use -h for usage information.\n");
        return -1;
    } else {
        for (int i = 1; i < argc; i++) {
            if (strcmp(argv[i], "-h") == 0) {
                show_help();
                return 0;
            } else if (strcmp(argv[i], "-v") == 0) {
                show_version();
                return 0;
            } else if (strcmp(argv[i], "-c") == 0) {
```

```
            color = true;
        } else if (strcmp(argv[i], "-r") == 0 && i + 2 < argc) {
            target_width = atoi(argv[i + 1]);
            target_height = atoi(argv[i + 2]);
            i += 2; // 跳过接下来的两个参数
        } else if (strcmp(argv[i], "-d") == 0 && i + 1 < argc) {
            custom_delay = atoi(argv[i + 1]);
            i++; // 跳过接下来的一个参数
        } else if (strcmp(argv[i], "-f") == 0 && i + 1 < argc) {
            filename = argv[i + 1];
            i++; // 跳过文件路径参数
        }
    }

    if (!filename || target_width <= 0 || target_height <= 0) {
        printf("Invalid or missing parameters. Please check your input and use -h for help.\n");
        return -1;
    }

    if (decoder_init(filename) != 0) {
        fprintf(stderr, "Failed to initialize decoder with file %s\n", filename);
        return -1;
    }

    double fps = get_fps();
    int delay = custom_delay > 0 ? custom_delay * 1000 : (int)(1000000.0 / fps);

    bool running = true;
    while (running) {
        system("clear"); // 或者system("cls"); 用于Windows
        Frame frame = decoder_get_frame();
        if (frame.data == NULL) {
            running = false;
            break;
        }

        Frame resized_frame = resize_with_average_pooling(frame, target_width, target_height);

        if (color) {
            print_frame_as_rgb(resized_frame);
        } else {
            print_frame_as_grayscale(resized_frame);
        }
```

```
164
165             usleep(delay); // 根据FPS或自定义延迟等待
166             free(resized_frame.data); // 释放调整大小后的帧数据
167         }
168
169         decoder_close();
170         return 0;
171     }
172 }
173
```

编译指令

```
1 gcc main.c -o main -L. -lvideodecoder -lavformat -lavcodec -lavutil -lswscale
2 ./main  -f /home/tom/Reference_video/dragon.mp4 -r 80 40 //打印灰色图
3 ./main -f /home/tom/Reference_video/dragon.mp4 -c -r 80 40 //打印彩色图
4 ./player -h //打开帮助文档
5 ./player -v //查看版本号
```

LEVEL 2-1 Multithreading

思路分析

什么是多线程：

1. 线程是程序中完成一个独立任务的完整执行序列，即一个可调度的实体。
2. 根据运行环境和调度者的身份，线程可分为内核线程和用户线程。
3. 线程库负责管理所有执行线程，比如线程的优先级、时间片等。线程库利用longjmp来切换线程的执行，使他们看起来像”**并发**“**执行**，但实际内核仍然是把整个进程作为最小单位来调度。

- 进程：一个正在运行的程序，程序执行完，系统进行回收
- 线程：进程内部的一条执行路径（序列）
- 不同语言不同平台上线程的实现机制有所不同

优点：

提高效率。因为各个不同的进程必须紧密合作才能满足加锁和数据一致性方面的要求，而用多线程来完成就比多进程容易得多。

优化性能。一个混杂着输入、计算和输出的应用程序，可以将这几个部分分离为3个线程来完成，从而优化程序执行的性能。

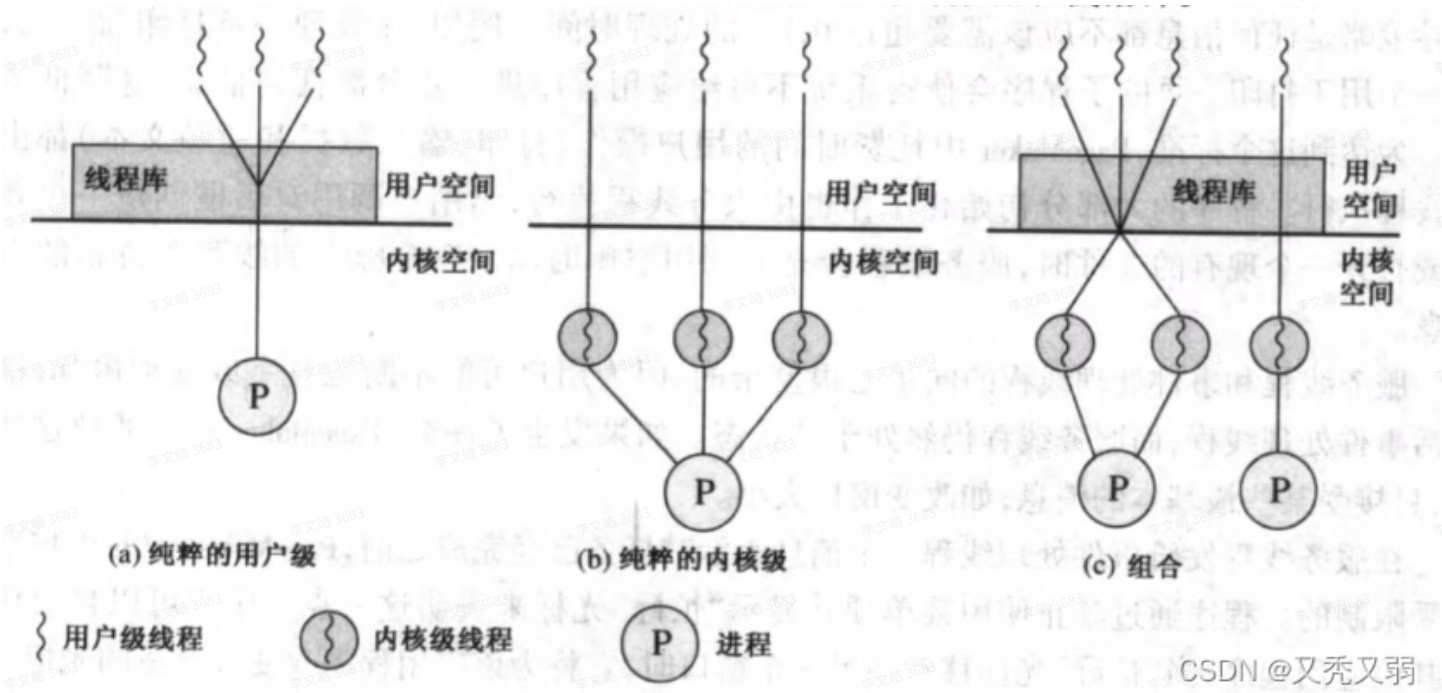
资源需求少。一般而言，线程之间的切换需要操作系统做的工作要比进程之间的切换少得多，因此多个线程多资源的需求要远小于多个进程。

缺点：

编写易出错。编写多线程程序需要非常仔细的设计。在多线程程序中，因时序上细微的偏差或无意造成的变量共享而引发错误的可能性是很大的。

调试难度大。对多线程程序的调试要比单个线程程序的调试困难得多，因为线程之间的交互难以控制。

实际情况复杂，要求高。将大量计算分为两个部分，并把这个两个部分作为不同的线程来运行的程序在一台单处理器机器上并不一定运行得更快，除非计算确实允许他的不同部分可以被同时计算，而且运行它的机器拥有多个处理器核来支持真正的多处理。



并发与并行

并发：一个处理器，在一段时间内交替运行

并行：任意时刻，两条程序都在执行（多个处理器）

并行是一种特殊的并发执行。

思考题

考虑到一个线程写缓存，一个线程读取缓存的情况下，我认为有几个重要的事项需要注意：

1. 要注意同步机制问题。在读取和写入缓存时，需要使用适当的同步机制，如互斥锁、信号量等，以防止多个线程同时访问和修改缓存数据导致的竞争条件和数据不一致性问题。
2. 必须要明确读写的先后顺序，要先进行写缓存之后再读取缓存，顺序出现问题会产生错误

3. 要确保读取到的缓存内容是最新的缓存内容而不是过期的内容或者是出现漏读取的情况
4. 要考虑到异常情况的处理机制问题

经查询我查到了一些常见的数据结构和解决方案

有队列，环形缓冲区，并发哈希表，信号量。其中我认为环形缓冲区可能是一个比较合适的选择，因为它可以提供较好的性能和空间利用率，并且相对容易实现。环形缓冲区是一个循环的、固定大小的缓冲区，可以通过读写指针来实现数据的存储和读取。它可以提供较好的性能和空间利用率。在多线程环境下，需要使用互斥锁或者其他同步机制来保护读写指针的访问。

代码

由于能力的问题，我并没有成功的解决出来给出正确的代码

LEVEL 2-2 Optimization

思路分析：

为了优化重复打印相似帧的问题和减少打印操作的时间复杂度，我们可以考虑以下两种优化方法：

1. **只打印变化的部分**：由于视频中连续帧之间的差异通常较小，我们可以只打印变化的部分。我们需要对比当前帧和前一帧，只更新发生变化的像素。这将大幅减少每帧需要打印的字符数量。
2. **缓冲区优化**：在当前的实现中，每个字符都是逐个打印的。这可以通过使用缓冲区进行优化。我们可以将整个帧的输出存储在一个字符串中，然后一次性打印出来，而不是逐个字符地打印。这样可以减少系统调用的次数，从而提高效率。

而为了实现这些优化，我通过查询和修改代码编写出了以下代码：

代码

```
// 假设 previous_frame_data 是一个全局变量，用于存储前一帧的数据
unsigned char *previous_frame_data = NULL;

void update_frame_as_grayscale(Frame frame) {
    int char_len = strlen(GRAYSCALE_CHARS) - 1;
    char *output = malloc(frame.width * frame.height + frame.height + 1);
    char *p = output;

    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            if (!previous_frame_data || memcmp(&frame.data[index],
&previous_frame_data[index], 3) != 0) {
```

```
                unsigned char r = frame.data[index];
                unsigned char g = frame.data[index + 1];
                unsigned char b = frame.data[index + 2];
                double gray = 0.299 * r + 0.587 * g + 0.114 * b;
                int val = (int)(char_len * gray / 255);
                *p++ = GRAYSCALE_CHARS[val];
            } else {
                *p++ = ' '; // 没有变化的像素用空格代替
            }
        }
        *p++ = '\n';
    }
    *p = '\0';

    printf("%s", output);
    free(output);

    // 更新前一帧数据
    if (previous_frame_data) {
        free(previous_frame_data);
    }
    previous_frame_data = malloc(frame.width * frame.height * 3);
    memcpy(previous_frame_data, frame.data, frame.width * frame.height * 3);
}
```

问题

然而在我尝试将这个代码放入文件后，打印出来的只有变化的部分，不难猜测，这和我在编辑函数时规定使用“ ”代替未变化的部分有关，我认为这种优化是不成功的，俗称“反向优化”，因此我没有将这部分代码编写到我的文件里面，单纯的使用某种符号代表未变化的部分显然是不合理的，但是很可惜的是我在进行了尝试，试图保留上一帧中不变的像素块时没有成功，出现了栈的错误，原因来源于我保留上一帧的指针指向堆栈被清空读取下一帧后仍然尝试从中读取上一帧的元素，发生了冲突。对于这个问题我还没有一个很好的方案进行处理。

LEVEL 3-1 Pause

代码展示

```
#include "video_decoder.h"
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
```

```c
#include <stdlib.h>
#include <termios.h>
#include <fcntl.h>

#define GRAYSCALE_CHARS " .:-=+*#%@"

void set_nonblocking_input() {
    struct termios oldt, newt;
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    fcntl(STDIN_FILENO, F_SETFL, O_NONBLOCK);
}

int kbhit() {
    struct timeval tv;
    fd_set fds;
    tv.tv_sec = 0;
    tv.tv_usec = 0;
    FD_ZERO(&fds);
    FD_SET(STDIN_FILENO, &fds);
    select(STDIN_FILENO + 1, &fds, NULL, NULL, &tv);
    return FD_ISSET(STDIN_FILENO, &fds);
}

void print_frame_as_grayscale(Frame frame) {
    int char_len = strlen(GRAYSCALE_CHARS) - 1;
    char *output = malloc(frame.width * frame.height + frame.height + 1);
    char *p = output;

    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];
            double gray = 0.299 * r + 0.587 * g + 0.114 * b;
            int val = (int)(char_len * gray / 255);
            *p++ = GRAYSCALE_CHARS[val];
        }
        *p++ = '\n';
    }
    *p = '\0';

    printf("%s", output);
    free(output);
```

```
}

void print_frame_as_rgb(Frame frame) {
    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];

            // 使用 ANSI 转义字符来设置前景色
            printf("\x1b[38;2;%d;%d;%dm", r, g, b);
        }
        printf("\x1b[0m\n"); // 重置颜色并换行
    }
}

// ANSI 转义字符，用于设置终端颜色
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_GRAY    "\x1b[90m"
Frame resize_with_average_pooling(Frame frame, int target_width, int
target_height) {
    Frame resized;
    resized.width = target_width;
    resized.height = target_height;
    resized.linesize = target_width * 3; // 假设RGB格式
    resized.data = malloc(resized.linesize * resized.height);

    int horizontal_step = frame.width / target_width;
    int vertical_step = frame.height / target_height;

    for (int y = 0; y < target_height; y++) {
        for (int x = 0; x < target_width; x++) {
            long sum_r = 0, sum_g = 0, sum_b = 0;
            for (int dy = 0; dy < vertical_step; dy++) {
                for (int dx = 0; dx < horizontal_step; dx++) {
                    int index = ((y * vertical_step + dy) * frame.linesize +
(x * horizontal_step + dx) * 3);
                    sum_r += frame.data[index];
                    sum_g += frame.data[index + 1];
                    sum_b += frame.data[index + 2];
                }
            }
            int count = vertical_step * horizontal_step;
            resized.data[(y * resized.linesize) + x * 3]     = sum_r / count;
            resized.data[(y * resized.linesize) + x * 3 + 1] = sum_g / count;
            resized.data[(y * resized.linesize) + x * 3 + 2] = sum_b / count;
```

```
        }
    }

    return resized;
}

void show_help() {
    // 显示帮助信息
    printf("Usage: ./player [OPTIONS]\n");
    printf("Options:\n");
    printf("  -h        Display this help message and open document\n");
    printf("  -v        Output version information\n");

    // 使用系统调用打开指定位置的文档
    if (access("/home/tom/example.txt", F_OK) != -1) {
        system("xdg-open /home/tom/example.txt");
    } else {
        printf("无法打开文档。\n");
    }
}

void show_version() {
    // 显示版本信息
    printf("版本信息: Dian团队工程开发测试\n");
}

int main(int argc, char *argv[]) {
    bool color = false;
    int target_width = 0;
    int target_height = 0;
    int custom_delay = -1; // 用户可以通过命令行参数指定自定义延迟
    const char *filename = NULL; // 通过 -f 参数指定的文件名

    if (argc == 1) {
        printf("Missing options. Use -h for usage information.\n");
        return -1;
    } else {
        for (int i = 1; i < argc; i++) {
            if (strcmp(argv[i], "-h") == 0) {
                show_help();
                return 0;
            } else if (strcmp(argv[i], "-v") == 0) {
                show_version();
                return 0;
            } else if (strcmp(argv[i], "-c") == 0) {
                color = true;
            } else if (strcmp(argv[i], "-r") == 0 && i + 2 < argc) {
```

```
145                 target_width = atoi(argv[i + 1]);
146                 target_height = atoi(argv[i + 2]);
147                 i += 2; // 跳过接下来的两个参数
148             } else if (strcmp(argv[i], "-d") == 0 && i + 1 < argc) {
149                 custom_delay = atoi(argv[i + 1]);
150                 i++; // 跳过接下来的一个参数
151             } else if (strcmp(argv[i], "-f") == 0 && i + 1 < argc) {
152                 filename = argv[i + 1];
153                 i++; // 跳过文件路径参数
154             }
155         }
156 
157         if (!filename || target_width <= 0 || target_height <= 0) {
158             printf("Invalid or missing parameters. Please check your input and use -h for help.\n");
159             return -1;
160         }
161 
162         if (decoder_init(filename) != 0) {
163             fprintf(stderr, "Failed to initialize decoder with file %s\n", filename);
164             return -1;
165         }
166 
167         double fps = get_fps();
168         int delay = custom_delay > 0 ? custom_delay * 1000 : (int)(1000000.0 / fps);
169 
170         bool running = true;
171         // 设置非阻塞输入
172         set_nonblocking_input();
173 
174         bool paused = false; // 标记是否暂停
175 
176         while (running) {
177             // 检测键盘输入
178             if (kbhit()) {
179                 char ch = getchar();
180                 if (ch == ' ') {
181                     paused = !paused; // 暂停/继续切换
182                 }
183             }
184 
185             if (!paused) {
186                 system("clear"); // 或者system("cls"); 用于Windows
187                 Frame frame = decoder_get_frame();
188                 if (frame.data == NULL) {
```

```
                    running = false;
                    break;
                }

                Frame resized_frame = resize_with_average_pooling(frame, target_width, target_height);

                if (color) {
                    print_frame_as_rgb(resized_frame);
                } else {
                    print_frame_as_grayscale(resized_frame);
                }

                usleep(delay); // 根据FPS或自定义延迟等待
                free(resized_frame.data); // 释放调整大小后的帧数据
            }
        }

        decoder_close();
        return 0;
    }
}

```

存在的问题

添加了暂停代码后，不知道具体是什么原因，导致视频的播放似乎更加模糊了一些

LEVEL 3-2 Accelerate

任务分析：

我选择在视频播放过程中，按下a键，视频播放速度加快2倍速

代码展示

```c
#include "video_decoder.h"
#include <stdio.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <termios.h>
#include <fcntl.h>
```

```
9
10 #define GRAYSCALE_CHARS " .:-=+*#%@"
11
12 void set_nonblocking_input() {
13     struct termios oldt, newt;
14     tcgetattr(STDIN_FILENO, &oldt);
15     newt = oldt;
16     newt.c_lflag &= ~(ICANON | ECHO);
17     tcsetattr(STDIN_FILENO, TCSANOW, &newt);
18     fcntl(STDIN_FILENO, F_SETFL, O_NONBLOCK);
19 }
20
21 int kbhit() {
22     struct timeval tv;
23     fd_set fds;
24     tv.tv_sec = 0;
25     tv.tv_usec = 0;
26     FD_ZERO(&fds);
27     FD_SET(STDIN_FILENO, &fds);
28     select(STDIN_FILENO + 1, &fds, NULL, NULL, &tv);
29     return FD_ISSET(STDIN_FILENO, &fds);
30 }
31
32 void print_frame_as_grayscale(Frame frame) {
33     int char_len = strlen(GRAYSCALE_CHARS) - 1;
34     char *output = malloc(frame.width * frame.height + frame.height + 1);
35     char *p = output;
36
37     for (int y = 0; y < frame.height; ++y) {
38         for (int x = 0; x < frame.width; ++x) {
39             int index = y * frame.linesize + x * 3;
40             unsigned char r = frame.data[index];
41             unsigned char g = frame.data[index + 1];
42             unsigned char b = frame.data[index + 2];
43             double gray = 0.299 * r + 0.587 * g + 0.114 * b;
44             int val = (int)(char_len * gray / 255);
45             *p++ = GRAYSCALE_CHARS[val];
46         }
47         *p++ = '\n';
48     }
49     *p = '\0';
50
51     printf("%s", output);
52     free(output);
53 }
54
55 void print_frame_as_rgb(Frame frame) {
```

```
    for (int y = 0; y < frame.height; ++y) {
        for (int x = 0; x < frame.width; ++x) {
            int index = y * frame.linesize + x * 3;
            unsigned char r = frame.data[index];
            unsigned char g = frame.data[index + 1];
            unsigned char b = frame.data[index + 2];

            // 使用 ANSI 转义字符来设置前景色
            printf("\x1b[38;2;%d;%d;%dm", r, g, b);
        }
        printf("\x1b[0m\n"); // 重置颜色并换行
    }
}

// ANSI 转义字符，用于设置终端颜色
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_GRAY    "\x1b[90m"
Frame resize_with_average_pooling(Frame frame, int target_width, int target_height) {
    Frame resized;
    resized.width = target_width;
    resized.height = target_height;
    resized.linesize = target_width * 3; // 假设RGB格式
    resized.data = malloc(resized.linesize * resized.height);

    int horizontal_step = frame.width / target_width;
    int vertical_step = frame.height / target_height;

    for (int y = 0; y < target_height; y++) {
        for (int x = 0; x < target_width; x++) {
            long sum_r = 0, sum_g = 0, sum_b = 0;
            for (int dy = 0; dy < vertical_step; dy++) {
                for (int dx = 0; dx < horizontal_step; dx++) {
                    int index = ((y * vertical_step + dy) * frame.linesize + (x * horizontal_step + dx) * 3);
                    sum_r += frame.data[index];
                    sum_g += frame.data[index + 1];
                    sum_b += frame.data[index + 2];
                }
            }
            int count = vertical_step * horizontal_step;
            resized.data[(y * resized.linesize) + x * 3]     = sum_r / count;
            resized.data[(y * resized.linesize) + x * 3 + 1] = sum_g / count;
            resized.data[(y * resized.linesize) + x * 3 + 2] = sum_b / count;
        }
    }

```

```
    return resized;
}

void show_help() {
    // 显示帮助信息
    printf("Usage: ./player [OPTIONS]\n");
    printf("Options:\n");
    printf("  -h        Display this help message and open document\n");
    printf("  -v        Output version information\n");

    // 使用系统调用打开指定位置的文档
    if (access("/home/tom/example.txt", F_OK) != -1) {
        system("xdg-open /home/tom/example.txt");
    } else {
        printf("无法打开文档。\n");
    }
}

void show_version() {
    // 显示版本信息
    printf("版本信息: Dian团队工程开发测试\n");
}

int main(int argc, char *argv[]) {
    bool color = false;
    int target_width = 0;
    int target_height = 0;
    int custom_delay = -1; // 用户可以通过命令行参数指定自定义延迟
    const char *filename = NULL; // 通过 -f 参数指定的文件名

    if (argc == 1) {
        printf("Missing options. Use -h for usage information.\n");
        return -1;
    } else {
        for (int i = 1; i < argc; i++) {
            if (strcmp(argv[i], "-h") == 0) {
                show_help();
                return 0;
            } else if (strcmp(argv[i], "-v") == 0) {
                show_version();
                return 0;
            } else if (strcmp(argv[i], "-c") == 0) {
                color = true;
            } else if (strcmp(argv[i], "-r") == 0 && i + 2 < argc) {
                target_width = atoi(argv[i + 1]);
                target_height = atoi(argv[i + 2]);
                i += 2; // 跳过接下来的两个参数
```

```
148             } else if (strcmp(argv[i], "-d") == 0 && i + 1 < argc) {
149                 custom_delay = atoi(argv[i + 1]);
150                 i++; // 跳过接下来的一个参数
151             } else if (strcmp(argv[i], "-f") == 0 && i + 1 < argc) {
152                 filename = argv[i + 1];
153                 i++; // 跳过文件路径参数
154             }
155         }
156 
157         if (!filename || target_width <= 0 || target_height <= 0) {
158             printf("Invalid or missing parameters. Please check your input and use -h for help.\n");
159             return -1;
160         }
161 
162         if (decoder_init(filename) != 0) {
163             fprintf(stderr, "Failed to initialize decoder with file %s\n", filename);
164             return -1;
165         }
166 
167         double fps = get_fps();
168         int delay = custom_delay > 0 ? custom_delay * 1000 : (int)(1000000.0 / fps);
169         int playback_speed = 1; // 播放速度，默认为1
170 
171         bool running = true;
172         // 设置非阻塞输入
173         set_nonblocking_input();
174 
175         bool paused = false; // 标记是否暂停
176 
177         while (running) {
178             // 检测键盘输入
179             if (kbhit()) {
180                 char ch = getchar();
181                 if (ch == ' ') {
182                     paused = !paused; // 暂停/继续切换
183                 } else if (ch == 'a') {
184                     playback_speed *= 2; // 播放速度加倍
185                 }
186             }
187 
188             if (!paused) {
189                 system("clear"); // 或者system("cls"); 用于Windows
190                 Frame frame = decoder_get_frame();
191                 if (frame.data == NULL) {
```

```
                    running = false;
                    break;
                }

                Frame resized_frame = resize_with_average_pooling(frame, target_width, target_height);

                if (color) {
                    print_frame_as_rgb(resized_frame);
                } else {
                    print_frame_as_grayscale(resized_frame);
                }

                usleep(delay / playback_speed); // 根据播放速度调整等待时间
                free(resized_frame.data); // 释放调整大小后的帧数据
            }
        }

        decoder_close();
        return 0;
    }
}

```

存在的问题

只能加速，连续按多次后会彻底抽象，不可阻挡。

最后注意：

我上传了两个main.c文件，其中main.c是存放的不包含暂停和加速功能的代码，而testmain.c是包含暂停和加速功能的代码，上传的example.txt代码是所编写的帮助文件