开发日志

唐德伟

2.12

上午11点接到题目，但由于星期二上下午满课，所以翘了下午一门电路理论课开始做工程题。（不算严格翘，只是偶尔听课，大部分时间在课堂上折腾电脑）。

工程图乍一看不算太难，我注意到“我们提供的视频解析库只能在 Linux 平台下使⽤，建议使⽤ Linux 系统作为编程平台，如果你使⽤的不是 Linux 系统，请考虑使⽤其他⽅法进⾏视频解析”这行说明，发现我还得配置linux环境（悲）。正式开启了我备受折磨的一天。

我对Linux系统几乎0基础，为了配一台Linux虚拟机，我从网上找了很多资料，下载了很多软件Ubuntu,VMware,Debian,后来发现只有Ubuntu最好用，VMwareVMware Workstation太慢了。

我还安装了Windows的Linux子系统，以及将wsl1给更新到wls2（不然vscode使用remote-wls时会出现很多问题），以及学习windows主机和ubuntu互传文件，和Ubuntu查看本机IP地址和设置传输权限。

一天基本都在配置环境。

另外，我开始做LEVEL 0-2 Parsing parameters任务（起初我用c++写工程），即命令行参数。命令行里的指令可能包含可选参数，用一般的读取指令操作比较麻烦，后来我思考用list链表存储，每次遇“-”字符就保存一条指令，与之后的指令一同存储为一个整体，遇到新的“-”则处理已保存的指令，然后清空链表。但是后来发现链表读取数据比较麻烦，所以用了c++的可变长数组vector容器来实现，效果很好。（但是后来因为c++链接.a失败问题，改用了c，用c重新实现了这种方法，最后优化成不需要复制内存，只要改变指针指向的实现方式）

完成LEVEL 0-2。

参考文献：

Ubuntu下载:<https://releases.ubuntu.com/20.04/>

Windows的Linux子系统（WSL）安装：<https://www.jianshu.com/p/dabcbb9dd5e1>

VScode的WSL使用：<https://blog.csdn.net/yao00037/article/details/119858692>

比较wsl版本：<https://learn.microsoft.com/zh-cn/windows/wsl/compare-versions>

# VSCODE更改文件时，提示：EACCES: permission denied的解决办法（ubuntu16.04虚拟机）：<https://blog.csdn.net/Dontla/article/details/120415587>

# wsl使用vscode连接，远程安装C/C++ 拓展时，报错：

<https://blog.csdn.net/wj617906617/article/details/133875034>

更新wsl1到wsl2：<https://download.csdn.net/download/weixin_43180746/12871470?spm=1001.2101.3001.6661.1&utm_medium=distribute.pc_relevant_t0.none-task-download-2%7Edefault%7ECTRLIST%7EAntiPaid-1-12871470-blog-118978891.235%5Ev43%5Econtrol&depth_1-utm_source=distribute.pc_re>

<https://huaweicloud.csdn.net/63561c0ed3efff3090b5a9b2.html?spm=1001.2101.3001.6650.15&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Eactivity-15-123813570-blog-118978891.235%5Ev43%5Econtrol&depth_1-utm_source=distribu>

# vscode检测到#include错误，请更新includePath。解决方法：

<https://www.cnblogs.com/lihuigang/p/17186051.html>

VScode的IntelliSense for cross-compiling：

[https://code.visualstudio.com/docs/cpp/configure-intellisense-crosscompilation#\_intellisense-mode](https://code.visualstudio.com/docs/cpp/configure-intellisense-crosscompilation" \l "_intellisense-mode)

vscode在Linux环境下运行“权限不够”的解决方法：

<https://blog.csdn.net/qq_37435462/article/details/111602904>

### VMware Workstation Pro：

<https://vmware.vmecum.com/>

# 如何在Debian 10 Linux安装g++/gcc开发工具：

<https://www.myfreax.com/how-to-install-gcc-compiler-on-debian-10/>

# Linux / Ubuntu上使用vscode编译运行和调试C/C++：

<https://zhuanlan.zhihu.com/p/80659895>

# windows主机和ubuntu互传文件的4种方法：

[https://blog.csdn.net/luobeihai/article/details/124003483#%E6%A6%82%E8%A7%88](https://blog.csdn.net/luobeihai/article/details/124003483" \l "%E6%A6%82%E8%A7%88)

FILEZILLA中文网

<https://www.filezilla.cn/download/client>

# Ubuntu 18.04 查看本机IP地址的两种方法 ip和ifconfig:

<https://blog.csdn.net/qq_34626094/article/details/113113380>

2.13

上午满课，下午开始工程进程，寻求windows主机和ubuntu互传文件解决方案，于是找到了Filezilla，但是发现Filezilla有很多问题，传输总是失败，查阅后发现是linux对应文件权限不够，为only read状态，于是展开了漫长的改错环节，后来对vsftpd.conf进行配置后才使问题得以解决，终于实现将.a文件与视频文件传输到linux系统里。

由于没接触过.a库，所以总是链接不上.a文件，上网搜寻了很久才得以解决，但是.cpp链接出了问题，只能改用.c文件写工程了。顺便学会.a文件的创建与vscode上lib库，src库，include库的命令行链接操作：

g++ -c \*.cpp

ar rcs libtest.a \*.o

g++ src/main.cpp -I include/ -L lib/ -l test -o math.out

./math.out

gcc src/main.c -I include/ -L lib/ -lvideodecoder -lavformat -lavcodec -lavutil -lswscale -o main.out

学习了一点ANSI转义码。

参考文献：

# 如何查看Ubuntu的IP地址以及端口号:

<https://blog.csdn.net/qq_39346534/article/details/107459325>

# Filezilla无法启动传输的问题:

<https://blog.csdn.net/your_flavor/article/details/118611412>

# 解决利用FileZilla无法向Ubuntu服务器上传文件问题:

<https://blog.csdn.net/sollinzhang/article/details/103532353>

# ftp上传文件时出错：550 Permission denied:

<https://blog.csdn.net/u014691333/article/details/115227957>

# Linux搭建FTP的vsftpd.conf文件配置详解:

<https://blog.csdn.net/ever_peng/article/details/80161924>

# [Linux系统中vsftpd配置及如何添加vsFTPd用户和设置权限](https://www.cnblogs.com/liangjiayan/p/10917883.html):

<https://www.cnblogs.com/liangjiayan/p/10917883.html>

# FileZilla报错严重文件传输错误 550permission denied:

<https://blog.csdn.net/weixin_44070109/article/details/119452830>

# 静态库和动态库基于Windows和VScode:

<https://blog.csdn.net/qq_53744721/article/details/122902857>

# linux下vscode编译和调试时链接库:

<https://blog.csdn.net/qq_29642797/article/details/110141435>

# GCC创建和使用静态链接库（.a文件）:

<https://c.biancheng.net/view/7168.html>

# 如何判定一段内存地址是不可访问的？:

<https://zhuanlan.zhihu.com/p/147988056>

# C语言中经常遇到的 segmentation fault 错误:

<https://blog.csdn.net/lds_lsj/article/details/48199101>

# ANSI转义代码(ansi escape code):

<https://zhuanlan.zhihu.com/p/570148970>

2.14

首先深度学习了ANSI码，用它实现了命令行字体的颜色变化，于是开始着手利用libvideodecoder库写视频帧打印项目。Frame.data的数据我花了一些时间才搞懂原理：3个一组存放RGB值，每行数据（linesize）存放视频帧一行像素的色彩对，linesize始终是width的3倍，且data数组貌似足够大，基本不会被越界。

同时经测验得到用get\_frame\_index()获取帧一般不能达到最大帧get\_total\_frames()，如bad apple中仅能获取到6570帧，而总帧为6574，所以判断视频是否播放完全时判定如下即可：

if (get\_frame\_index() >= get\_total\_frames() - 10)

利用这些性质，我很快就写出了根据二维像素格位置计算data对应下标的函数：

int frame\_cal(int row,int colmn){

    return frame\_linesize \* row + colmn \* 3;

}

利用这个函数，我很容易就能打印出一帧所有像素的分布情况，于是我先将图片帧的每个像素都打印到命令行上，但效果并不理想，看起来有很多画面割裂，我推断是一行的打印值过多，需要缩减。

于是我开始尝试resize画面帧，下一个难点是像素的RGB获取，简单学习并写了下average pooling & max pooling原理的resize函数，于是就可以实现对一帧进行彩色打印了，灰度图打印同理。

会做一帧后播放动画就很简单了，不过需要注意控制帧率。我采用了计时器的方式来控制帧率。

current\_time = clock();

if (current\_time > last\_time + 1 second / video\_play\_fps){

       printf("\x1b[H\x1b[2J"); //清屏

last\_time = current\_time;

...............

       frame\_show(&f); // 显示一帧

}

对于同⼀个视频⽽⾔，前后两帧的差异似乎并不⼤，反复打印每⼀帧太⿇烦了，且容易造成画面非常闪（没开双缓冲导致的）。所以我决定人为忽略掉一些帧，来让画面更流程（因为更新帧太多会导致画面非常闪）。

int skip\_frame = 5;

// 在上面的if语句中插入以下代码

for (int i = 0;i < skip\_frame;i ++) f = decoder\_get\_frame();

这样就让我们的画面既能稳定播放，又不至于非常闪了。

前面提到我没开双缓冲，我去查了下Linux下双缓冲绘图非常麻烦，所以忽略。

基本实现LEVEL 0-3，LEVEL 1-1，LEVEL 1-2，LEVEL 1-3，LEVEL 2-2。

参考文献：

## 免费更改视频视频分辨率网站:

[https://online-video-cutter.com/cn/resize-video#google\_vignette](https://online-video-cutter.com/cn/resize-video" \l "google_vignette)

# 解决FileZilla连接虚拟机尝试连接“ECONNREFUSED - 连接被服务器拒绝”失败：

<https://blog.csdn.net/qq_41228218/article/details/104022889>

# 关于 linux c 清屏命令：

<https://blog.csdn.net/silno/article/details/109678293>

# 双缓冲(Double Buffer)原理和使用:

<https://blog.csdn.net/xiaohui_hubei/article/details/16319249?spm=1001.2101.3001.6650.2&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-2-16319249-blog-90136682.235%5Ev43%5Econtrol&depth_1-utm_source=distribute.pc_relevant.none>

# Linux-FrameBuffer双缓冲机制显示图像:

<https://blog.csdn.net/vincent040/article/details/116301614>

# Linux中多线程时编译报错：undefined reference to `thrd\_create‘:

<https://blog.csdn.net/weixin_53057335/article/details/127178761>

2.15

休息一天。

2.16

开始处理多线程工作。多线程这边确实很麻烦，需要考虑全局变量的读取与线程分配问题。我花了很多时间才搞好分配线程的工作。

我查了Linux最大线程数量，其非常之多，我的linux虚拟机就有接近5万，所以我完全可以多利用一些线程来辅助工作。

关于多线程处理问题，一开始我对buffer缓存的存储方式是利用一个RGB数组存放resize之后的所有像素格（相当于一个resize之后的Frame中的data数据），并将其对应帧索引等信息包装成结构体组成一个缓存存储单元Buffer\_saving，利用该单元构建一个含若干元素的数组buffer\_save（100~200）。

利用多线程在每次打印前调用若干个线程对未构建好的buffer\_save进行帧的填充，同时应注意避免一个帧数据采集线程重复被调用，以及避免同一个Buffer\_saving同时被多个线程处理，所以要对线程设置繁忙状态处理，和其他一些细节的边界判断处理。

最后把以及采集好数据的Buffer\_saving交给打印线程进行打印，需要控制数据采集线程不能到达打印线程的位置。这时我用的打印线程是基于printf进行打印的，打印效率不算太高，播放视频帧率一块也很容易闪。

多线程使用的最大好处就是使我的视频的最大帧率变得非常高，打印帧率可以提高到600帧/s（但实际综合帧率应该在400帧/s左右），使10s内速通bad apple成为可能。

由于我的代码播放视频画面看着还是很闪，但是从工程题目给的bad apple演示视频来看画面非常流畅，让我比较好奇，于是细究了下才发现演示视频用的是字符集打印画面。于是我去搜了搜字符集打印图原理，于是到自己的计算机上跑了一下，发现效果非常好，画面流畅至极，即便将帧率设置到100帧/s也不会出现闪烁的情况，very good。

同时基础的可选命令行参数并不能满足一些特定需求，于是我对命令行参数做了些调整与更新：

-c / --color 后可接4种参数 m/a/g/p 分别对应堆积池采用Max pooling/Average pooling算法,打印灰度图/打印字符图

-fps / --fps 设置视频初始播放帧率

-s / --skip 设置每次打印帧时忽略的帧数

做了这些补充主要是为了便于调试代码。

完成LEVEL 2-1。

参考文献：

《C How To Program》-Pual Deitel Harvey Deitel版P694-P699

# 【Linux】Linux可以运行多少进程,一个进程可以开多少线程:

<https://blog.csdn.net/bandaoyu/article/details/90578446>

2.17

多线程完工之后就是获取终端输入来实现对视频加速与暂停了，到网上找了个轮子之后直接利用一个线程来异步获取终端输入就ok了，加速与暂停功能实现是非常容易的。

实现暂停和加速还是不太符合我的预期，所以我照样对终端输入的处理多加点功能：帮助h/H，减速a/A，加速d/D，暂停p/P/空格，字符图打印切换g/G，强制退出程序q/Q。

做完这些补充后程序的可操纵性就很强了。

完成LEVEL 3-1，LEVEL 3-2。

至此，项目基本全部完成。剩下是做些优化。

对于level2-2的思考：前一天我做了多线程，但其中打印线程是利用printf进行对像素块一一打印出了，效率较慢。网络冲浪后发现由于printf需要格式化，puts打印效率大约是printf的3/2，所以我对线程数据收集与打印函数进行了优化。新的Buffer\_saving存储数据的方式是将带颜色信息的像素方块统一存放到一个字符串里，也就是将整张帧图连同换行符一同存储在一个字符串，打印时直接用puts()函数直接把整个字符串打印下来就可以了。

这样可以让打印的效率达到几乎极致（至少没开双缓冲绘图时应该是这样），对打印函数改良之后在较高帧率（60帧/s ~ 100帧/s）下抗闪烁质量明显提高，低帧下闪烁程度至少能在我的可接受范围内了。

另外我还对字符图打印做了个额外的punct\_buffer存储，以便无论是彩图打印还是灰度图打印都可以在播放时按g键就可以随时切换字符图打印了，观赏效果很好。

至此我的工程优化完成。

如下是线程优化代码，之后无其他内容，可忽略。

**线程优化代码：**

#define THRD\_NUM 200        // 线程最大数量

#define BUFFER\_MAX 1000      // 帧缓存最大上限

#define THRD\_PRINT 0        // 专门负责打印的线程

#define THRD\_GETKEY 90      // 专门负责获取终端输入的线程

thrd\_t threads[THRD\_NUM];               // 线程数组

int threads\_busy[THRD\_NUM] = {0};       // 对应线程是否繁忙

typedef struct{         // 帧缓冲保存容器

    char \*buffer;   // 帧缓存像素数据

    char \*punct\_buffer; // 字符图帧缓存像素数据

    int is\_done;    // 对应帧是否构建完全

    int index;      // 帧的序列

    //uint8 \*buffer;  // 帧像素数据(已废用)

}Buffer\_saving;

Buffer\_saving buffer\_save[BUFFER\_MAX + 10] = {0};       // 帧缓冲保存（打印帧时使用该容器中缓冲进行打印）

int buffer\_cur\_collect[100] = {0};      // 对应线程当前处理的帧序列

int buffer\_cur\_print = 0;               // 当前打印的帧

int buffer\_update\_velocity = 3;         // 一次进行帧缓冲储存处理的线程数量

typedef struct{     // 执行多线程保存帧缓冲时的传参容器

    int cur\_thrd\_index;         // 当前处理帧缓冲保存的线程索引

    Frame frame;            // 帧数据

    int handle\_frame\_index; // 当前处理的帧的序列

}Collect\_data;

int frame\_buffer\_collect(void \*ptr);        // 帧缓存收集

int frame\_buffer\_print(void \*ptr);          // 帧缓存打印

Main函数while（1）循环体内部：

if (is\_play\_paused == false) printf("\x1b[H\x1b[2J");                        //清屏

// 多线程储存帧缓冲（存储速率远远大于打印速率）

for (int i = 1;i <= buffer\_update\_velocity && get\_frame\_index() <= get\_total\_frames() - 5 ;i ++){

if (threads\_busy[i] == false){  // 繁忙中的线程跳过

f = decoder\_get\_frame();                    // 解码一帧，获取对应数据

threads\_busy[i] = true;     // 对应线程置繁忙状态，防止重复调用相同线程

Collect\_data collect\_data = {i,f,get\_frame\_index()};    // 传参

thrd\_create(&threads[i],frame\_buffer\_collect,&collect\_data);   // 创建线程工作

for (int i = 0;i < skip\_frame;i ++) f = decoder\_get\_frame();

// 略读，忽略一些帧

}

}

if (is\_play\_paused == false) { // 打印帧缓冲

printf("current index: %d / %d\t",buffer\_save[buffer\_cur\_print - 1].index,total\_frame);

printf("time: %d  (%.2fs)\t",(int)current\_time,1.0 \* current\_time / (1 second));

printf("Video play speed: %.2f\tSource fps equal:%.2f\n",video\_play\_fps,video\_play\_fps \* (1 + skip\_frame));

thrd\_create(&threads[THRD\_PRINT],frame\_buffer\_print,NULL);  // 创 建打印线程

thrd\_join(threads[THRD\_PRINT],NULL); // 打印线程结束后才继续执行之后 语句

printf("current index: %d / %d\t",buffer\_save[buffer\_cur\_print - 1].index,total\_frame);

  printf("time: %d  (%.2fs)\t",(int)current\_time,1.0 \* current\_time / (1 second));

printf("Video play speed: %.2f\tSource fps equal:%.2f\n",video\_play\_fps,video\_play\_fps \* (1 + skip\_frame));

}

函数部分:

int frame\_buffer\_collect(void \*ptr){        // 帧缓存收集

    Collect\_data cur\_data = \*(Collect\_data \*)ptr;

    if (cur\_data.frame.data == NULL || (buffer\_cur\_collect[cur\_data.cur\_thrd\_index] + 10) % BUFFER\_MAX == buffer\_cur\_print) {

        threads\_busy[cur\_data.cur\_thrd\_index] = false;

        return thrd\_error;      // 若传参为空或所有缓存区（非严格）均满， 退出线程

    }

    While(buffer\_save[buffer\_cur\_collect[cur\_data.cur\_thrd\_index]].i s\_done != false){

            // 使当前线程欲收集的缓存区指向尚 未收集过缓存的缓存区

        if ((buffer\_cur\_collect[cur\_data.cur\_thrd\_index] + 10) % BUFFER\_MAX == buffer\_cur\_print) {

            threads\_busy[cur\_data.cur\_thrd\_index] = false;

            return thrd\_error;      // 所有缓存区（非严格）均满，退出线程

        }

        buffer\_cur\_collect[cur\_data.cur\_thrd\_index] = (buffer\_cur\_collect[cur\_data.cur\_thrd\_index] + 1) % BUFFER\_MAX;

    }

    char \*new\_buffer = (char \*)malloc(25 \* new\_frame\_height \* new\_frame\_width \* 3); // 13 + 9 + 2

                // 帧存储开辟空间，由于一个"\x1b[38;2;%d;%d;%dm█"元素存储 16~22字节（依据3个%d数字的位数增加而增加），所以统一设 字节量为25

    char \*new\_punct\_buffer = (char \*)malloc(new\_frame\_height \* (new\_frame\_width + 1) + 100);

                // 字符图帧开辟空间

    int pos = 0,punct\_pos = 0;  // 位置索引

    char p[25];     // 辅助单元

    color color2;

    for (int i = 0;i < new\_frame\_height;i ++){      // 保存缓存信息

        for (int j = 0;j < new\_frame\_width;j ++){

            if (pooling == MAX\_POOLING)

                color2=resize\_max\_pooling(&cur\_data.frame,pixel\_width, pixel\_height,i,j);

            else if (pooling == AVERAGE\_POOLING)

                color2=resize\_average\_pooling(&cur\_data.frame,pixel\_w idth,pixel\_height,i,j);

            new\_punct\_buffer[punct\_pos++]=gray\_to\_punct(gray\_cal(&col or2));

            if (is\_gray) color2.R=color2.G=color2.B=gray\_cal(&color2);

            sprintf(p,"\x1b[38;2;%d;%d;%dm█",color2.R,color2.G,color2. B);

            int m = 3;     // 颜色RGB3个数总位数

            while (color2.R /= 10) m += 1;

            while (color2.G /= 10) m += 1;

            while (color2.B /= 10) m += 1;

            strncpy(new\_buffer + pos,p,13 + m); // 保存颜色信息

            pos += 13 + m;  // 索引更新

        }

        new\_buffer[pos ++] = '\n';

        new\_punct\_buffer[punct\_pos ++] = '\n';

    }

    new\_punct\_buffer[punct\_pos] = '\0';

    new\_buffer[pos] = '\0';

    new\_buffer = realloc(new\_buffer,pos + 1);   // 重新分配空间，避免内存浪费

    // 更新缓冲相关信息

    buffer\_save[buffer\_cur\_collect[cur\_data.cur\_thrd\_index]].index = cur\_data.handle\_frame\_index;

    buffer\_save[buffer\_cur\_collect[cur\_data.cur\_thrd\_index]].buffer = new\_buffer;

    buffer\_save[buffer\_cur\_collect[cur\_data.cur\_thrd\_index]].punct\_b uffer = new\_punct\_buffer;

    buffer\_save[buffer\_cur\_collect[cur\_data.cur\_thrd\_index]].is\_done = true;

    threads\_busy[cur\_data.cur\_thrd\_index] = false;  // 更新线程状态

    buffer\_cur\_collect[cur\_data.cur\_thrd\_index] = (buffer\_cur\_collect[cur\_data.cur\_thrd\_index] + 1) % BUFFER\_MAX;

    return thrd\_success;

}