Dian测试学习文档

第一次接触这样的代码和工程测试，对于我来说肯定是难度不小，但同时它也是我从零到一的学习的第一步，让我度过了忙碌又充实的一周。我的学习过程基本按照题目中教程的方向。

配置环境

此前基本没有了解过操作系统，也从来没有使用过linux系统。为了使用提供的解析库，需在linux系统下工作，我了解到多种可行办法，更换整个电脑操作系统、配置双系统、安装虚拟机、使用windows的子系统wsl2等，最终选择了易于操作的虚拟机。在VMware安装了ubuntu系统的虚拟机。linux系统中有所不同的的是它的命令行窗口，很多操作都通过命令行中输入指令来实现。我在几天的使用中也逐步了解熟悉一些常用指令.

```
clear                       清除屏幕
cd ~                        当前用户目录
cd /                        根目录
cd -                        上一次访问的目录
cd ..                       上一级目录
mkdir aaa                   在当前目录下创建aaa目录，相对路径;
mkdir ./bbb                 在当前目录下创建bbb目录，相对路径;
mkdir /ccc                  在根目录下创建ccc目录，绝对路径;
ls
realpath
touch
gcc main.c -o main
./main
```

在虚拟机上安装了编译器后，就可以使用gcc等指令对所写c语言文件进行编译，并直接在命令行窗口运行.又在虚拟机上安装的vscode来编写代码。

其中，教程里给出的下面这条指令用于将所写的c语言文件main.c与需要的静态库链接到一起并生成电脑可运行的文件main。我每次调试程序均需重新将其编译一次，再运行编译后文件。（感觉有点麻烦，是应该这样吗？）

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

读题与理解

查阅资料，了解videodecoder库，了解到宽度和高度指的是帧横向与纵向的像素数，了解到rgb像素数据，每个像素由3个字节存储，分别对应红，绿，蓝的数值，再根据ANSI转义字符即可输出对应的颜色。存储每一帧所有像素信息时是行优先，每帧存储空间大小也可通过读入数据计算出来。转换为灰度值同样也可打印出灰度值，我将255的空间均匀分配给7个字符，以实现灰度的区分。

```
"\x1b[38;2;%d;%d;%dm",red,green,blue     //转义字符输出color像素块
target_height*target_width*3*sizeof(unsigned char)  //每一帧需要的存储空间
double gray = 0.2126 * red + 0.7152 * green + 0.0722 * blue;  //转换为灰度值
' ' '.' ':' '=' '#' '&' '@'
```

打印图片

将一帧图片的每个像素块全部打印出来，发现图片过大，终端根本显示不下，很难看出图片内容。进一步跟随教程，设计resize函数，将周围几个像素块用一个像素块来表示，降低图片分辨率。教程提供了两种思路，其中我实现了averagepooling,用多层循环有效地缩小了图片，使之能在终端完整展示。

```
for (int y = 0;y < height;y += y2){
        for (int x = 0;x < width;x += x2){
            double red_total = 0;
            double green_total = 0;
            double blue_total = 0;
            for (int i = 0;i < y2;i++){
                for (int j = 0;j < x2;j++){
                    data = frame.data + linesize*(y + i) + 3*(x + j);
                    red_total += *data;
                    green_total += *(data + 1);
                    blue_total += *(data + 2);
                }
            }
            unsigned char red = (unsigned char)(red_total / num);
            unsigned char green = (unsigned char)(green_total / num);
            unsigned char blue = (unsigned char)(blue_total / num);
            frame_resize.data[(x/x2)*3 + (y/y2)*target_width*3] = red;
            frame_resize.data[(x/x2)*3 + (y/y2)*target_width*3 + 1] = green;
            frame_resize.data[(x/x2)*3 + (y/y2)*target_width*3 + 2] = blue;
```

但问题也随之出现了，我发现整个画面非常的黑，只能依稀分辨出中间的无牙仔。当时想到是算均值的时侯一定把所有的值都普遍算小了。果然，第一遍写的时候将red_total等值认为与red一样全定义成unsigned char型（如下），导致容量不够，损失大量精度。后来全部改成double型，并在最后赋值给red时强行改为（unsigned char）。最终图片打印的效果良好。

```
unsigned char red_total = 0;
unsigned char green_total = 0;
unsigned char blue_total = 0;
```

待改进之处：编写的resize函数只能将帧的width和height缩小整数倍。

改进思路：可能需要在边界处进行更完善的讨论与分类。

播放动画

编写一个循环，将动画中每帧依次打印，确实得到了每一帧的画面，但它们是依次向下打印出来的，不能形成动画的效果。我们希望能在相同的位置打印完每一帧。查阅资料，可以利用转义字符，需要分两步实行：1.清除上一帧画面 2.将光标移动到初始位置。

```
        printf("\033[H\033[2J");     //清除上一帧
         printf("\033[1;1H");       //光标移动到初始位置
```

在每一帧的打印后都执行这两个操作，成功的将图片“定”在了原地。为避免帧播放过快，可设计一些延时指令。最初的想法是如下的傻瓜式指令。但它无法精确控制延时，在后续调节帧率的任务中并不实用。

```
         print_grey(resize_frame);
         for (int j = 0;j < 1000000;j++){}
         printf("\033[H\033[2J");
         printf("\033[1;1H");
```

在以上一些操作过后，动画基本可以播放，但画面卡顿，下半部分画面呈“闪现”状。

问题分析：打印时按行从上到下依次打印，删除时从后往前删除，导致下半部分画面存在时间短于上半部分，因此出现“闪现”等不协调的情况。且仅从上往下打印速度较慢，导致画面不流畅。

改进思路：1.前一帧从上往下打印，则后一帧从下往上打印。可以使画面较为协调，但打印速度依然较慢。需要再编写一套从下向上打印的函数（运用ANSI转义字符移动光标沿每行向上）。

由于经验缺乏与能力暂且不足，我此次测试只做到了这里，但这次测试让我从零开始，丰富了自身知识。我非常认同dian团队“干中学”的理念，期待能加入dian进一步提升自己的能力，结识更多优秀的同学。