

视频技术入手指南

目录

设备选购.....	2
采集卡+高清头设备安装指南.....	3
网络摄像头安装指南.....	4
基于 cuvid 的 Cuda 视频支持详解.....	5
在视频处理中使用光栅 IO 分流	6

设备选购

- 1, 如果我们对实时性有很高要求 (0.1s), 选购环节, 应该定位于多路采集卡+hdmi 高清线+摄像头路线。这条路线, 我推荐的设备搭配为: 美乐威采集卡+高清线+小蚁或则狗头, 美乐威的采集卡相对 blackmagic 是更便宜的高端, 同样提供了接口到 c++ 的 sdk, hdmi 高清线需要购买摄像头*2 的数量, 因为我在实际使用过程发现高清线非常娇嫩, 放地上稍微不注意踩一脚, 或则接口稍微弯曲一下, 就会断掉, 至于摄制端小蚁和狗头可以通过 3d 打印自己做摄制机架, 现在一般都是 4k+30fps 起, 1080p+60fps 这类指标, 弄不来硬件就去多问网店。这类搭配, 美乐威 4 路的 4k 卡会卖 10000 上下, 因为 4k 视频的编码芯片还是非常高端的。一般来说, 1 张 4 路卡+10 跟高清线+4 个头+3d 打印的摄制架, 大概花销在 2 万出头, 到 3 万也很正常。
- 2, 如果对实时性要求不是那么高, 可以接受 0.5-1.0s 的延迟, 这时候可以选择网络摄像头, 一般来说, 网络摄像头分为红外支持+独立电源+wifi, 红外支持+poe 网线形式电源两种, 这些摄像头都可以支持 rtsp 协议 (rtsp 就是只有一路码流的视频)。购买摄像头以前, 一定要问清楚网店: 摄像头怎么供电, rtsp 视频支持度怎么样, 24h 监控稳定性, 有畸变怎么处理, 这些基本的东西一定问清楚。网络摄像头的优点是便宜, 缺点是帧率都不高, 一般是 30fps 以下, 根据自己的目标场景而定, 如果目标场景是运动, 比如行驶中的车辆, 人群, 我们就选择更高的帧率, 1000 元也可以买到 60fps 帧率的红外网络摄像头, 普通 2k 的红外摄像头, 大概在 500 元上下。

采集卡+高清头设备安装指南

如果是采集卡+高清头的硬件组合，应该是拿到就可以用，安装采集卡时，应该注意散热问题，因为多路的 4k 视频的光栅计算是非常容易发热，注意散热风道，风扇，液冷这类搭配。

接下来，到官方下载对应的 sdk 和工具，这类东西，都是 c++ 的。sdk 接口过来不是视频，都是一帧一帧的光栅，光栅数据可以通过 delphi -> c++ -> SDK，来获取

MemoryRaster.pas 库提供了强大的光栅支持系统，AI，CUDA，FFMPEG 等等支持，都是基于 MemoryRaster.pas 库提供的光栅地基在计算，接口到 Delphi 只需要接口 MemoryRaster 的数据结构就可以了

在光栅上画内容，是 zDrawEngine 最擅长干的事情。

把光栅编码成视频来处理，是 ffmpeg+cuda 最擅长干的事情。

识别光栅的内容，比如人脸，车牌，行人，那是 ZAI 最擅长干的事情。

下图是前几年我用的 8 路狗头+2 张 blackmagic 卡，总共花费了 32000，安装采集卡一定要注意散热风道，长期工作发热量非常大。



采集卡+高清头的优点是同步，实时性低于 0.1s

应用级开发可以做 VR，实时会议，实时交互，各种识别，比如卡口，车牌，等等

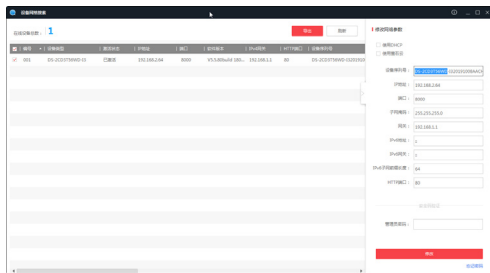
网络摄像头安装指南

网络摄像头，必须购买支持 RTSP/RTMP 协议的设备。

摄像头如果用 Wifi 网络，会频发断线问题，我们做应用软件时需要实现自动断线重连功能。RTSP/RTMP 自动化断线重连，我已经在 FFMPEG 库内置了自动化心跳检测，重连推流协议。无需操心。

电源系统，网络摄像头一般都是 poe 独立供电模块，也有 dc 供电模块，购买时候根据说明书安装即可。记住一点，没经验时，问清楚再下单。

我用的网络摄像头是海康威视红外头(型号：DS-2CD3T56WD)，使用前，需要用官方提供的工具先初始化，设置 ip 地址，激活，设置密码之类。如下图



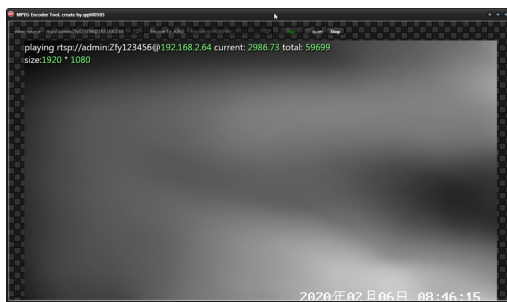
，我想，别的网路摄像头都会类似吧

这台设备是 ZAI 客户友情赞助的，我从拿到设备到开始动手解决视频方案，总共躺坑接近一个月。下面是我的应用经验：

首先，下单收货以后，别忙完成交易，先搭建好环境，跑几个小时来验证稳定性。

VLC 长期播放不一定会稳定，不确定 vlc 断线重连的支持度，建议使用内置的 ffmpeg Encoder Tool 进行稳定测试，如果发现问题，即时退单，换一下厂商。

在 video source 填写好 rtsp 的地址，encode to .h264 是编码格式输出，如果不填写，encoder tool 只播放不做输出，可以用于长时间稳定性测试。如果协议是 RTMP 一般是带有音频流，主视频流，子视频流，自定义数据流等等数据，FFMPEG Encoder Tool 会先过滤掉所有的非视频流，然后选择优先级最高的一个码流，既主视频流。该工具不会有声音，只有视频。该工具使用 ZAI 内置的视频支持系统，长期播放，不会有内存泄漏。



因为我经过实际测试，每隔 3-5 小时，摄制端系统会宕机一下，然后自动重启，卡顿十几秒，具体原因不详。或许，你在购买摄制端时，可以选择海康威视之外的品牌，或则别的型号试试，当然，你也可以入我的 DS-2CD3T56WD 坑，感同身受我的工作氛围。

FFMPEG Encoder Tool 在我的开源项目中，都会是内置的。

基于 cuvid 的 Cuda 视频支持详解

个人视频播放支持，一般是 CPU，因为单独视频的负载都没那么大，顶多跑到 20%的 cpu 开销。如果是视频推流，就会需要播放+编码，这时候，cpu 的开销就会达到 40%。多推两个，资源用光了。如果做类似，识别，应用的项目，这样肯定是不行的。这时候，我们就需要 cuvid 加速。

很多东西，由于我们不了解，往往都是看上去很美，实际接触才是坑多多。cuvid 也是这样。

Cuvid 是一个独立的 sdk 模块，安装完成 cuda sdk 以后，它会自动生效。

FFMPEG 是一个对各种编码器，解码器操作的集合框架，cuvid 在 ffmpeg 中就是一堆编码器和解码器。FFMPEG 在构建分发版本时处于兼容性考虑，不会默认来使用 cuvid，都是 cpu 编码解码，对于个人用户播放一点视频，那是没问题的，对于项目型 FFMPEG 需求，用 cpu 是不够的，得上 gpu，而我们上 gpu，就是非常规用法（ffmpeg 官方 demo 方案之外，更接近 nvidia 给出的 cuda demo）。

Nvidia 的 cuvid 是用 nvcc 构建而出的异构 gpu 机器码，再链接到应用中，既 ffmpeg+cuvid。我们在使用 ffmpeg 的 gpu 加速时，本质上是受 cuda 限制的，包括多路视频的编码指标，解码指标，等等，这些指标，只能从 cuda sdk 提供的文档去查找。在多数情况，每一代 GPU 都有各自的尖端架构代号，在 nvcc 中，以 sm2,sm3,sm4 这种命名开关来构建，要取得最佳的视频编码解码效率，一般是最新的显卡，或则学习卡。假如你的显卡架构是 sm5.0，而 cuda 的指定构建架构是 sm7.5（titan RTX），这样是不行的，除非 cuda 构建时指定 sm5.0。

ffmpeg+cuvid 支持的最重要指标是：HEVC(h265), H264，只有当视频解码器，编码器是这两个东西才能用到 cuvid，因为 cuvid 目前只支持这两个编码解码器。如果视频内置的编码器不是这两个东西，cuda 就不能干活，这时候就是用 cpu 来老牛拉车。

FFMPEG Encoder Tool 以.h264 标准作为交换格式，其实就是因为以上原因。

当我们安装完成 cuda sdk+FFMPEG SDK+MemoryRaster.pas 以后，所有的 Decoder+Encoder，都会自动切换成 cuvid 支持状态，编码 4k 视频，cpu 消耗一般都是 0。而 gpu 的消耗一般都是拉满的。另一方面，如果本地硬件不支持 cuda，视频支持系统会自动切换成 cpu 模式。

当我们从摄制端捕获 rtsp 到本地时，中间会解码，而解码这一步，都是 cuvid。cuda 对于多路视频解码的硬件性能支持提升会感觉非常明显（10 倍以上）。

当我们 Encoder to .H264 时，默认也是用 cuvid 内置的编码器，我们从 rtsp 解码一帧到光栅，然后再用 zDrawEngine 来画标注，弄弄图形+视觉，然后再用 cuda 编码，这一整个过程，用 cpu 会非常慢：我做到的单路视频重构极限是 50fps 左右，用 cuda 支持系统，我做到的单独视频重构极限可以到 700fps。

由于 cuda 硬件架构的关系，cuvid 的 decoder+encoder 是有限制的，不能无穷开。具体参数需要参考 nvidia 提出的 gpu 架构手册。

目前，好消息是 ffmpeg+cuvid 的支持我已经做到了傻瓜化+兼容化。拿着直接用，底层硬件最优化调整会自动解决。

总结：现代化视频流的处理就是用 cuda。

在视频处理中使用光栅 IO 分流

ffmpeg 在处理 rtsp/rtmp 协议时都会有一个积压缓冲区,假如我们的 ffmpeg 如果没有即时 readFrame,那么视频流数据会一直存在于积压缓冲区,缓冲区大了以后就会发生崩溃。这时候,我们需要保证 readFrame 一直在实时的进行着。我们用的方法就是使用光栅 IO;

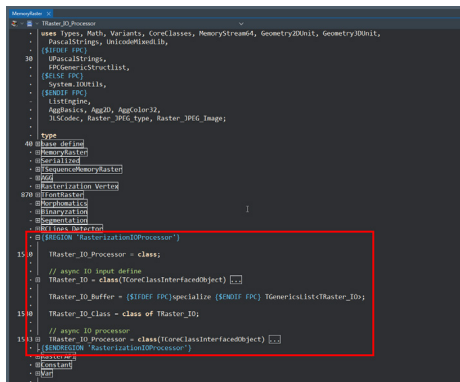
光栅 IO=Rasterization Input Output

光栅 IO 分流是对视频的像素光栅处理专用一个线程池,并且以队列输出输入。核心原理如下

- 输入视频光栅: input->1,2,3,4,5
- 1,2,3,4,5,会依次进入光栅 IO 分流的线程池,绝不卡顿 input
- 开始在后台处理光栅,处理先后顺序不会 1,2,3,4,5
- 处理完成,输出队列: output->1,2,3,4,5

以上处理机制,是无卡顿的流程,这在视频处理中,是非常重要的机制,因为视频一旦发现处理速度,跟不上流速度,会造成数据流积压,这时候如果我们不做跳帧处理,轻则发生大规模内存使用,重则发生崩溃。

具体支持细节我们参考 memoryRaster.pas 库中的 RasterizationIOProcessor,这是一个通用框架,并不局限于视频,甚至可以用于图片批处理。如下图



```
unit RasterIOProcessor;

uses Types, Math, Variants, CoreClasses, MemoryStream4, Geometry2DUnit, Geometry3DUnit,
    PascalStrings, UnicodeUtilsLib,
    {$IFDEF FPC}
    PASCALStrings,
    FPGenericStructList,
    {$ELSE}
    System.IDUnit,
    {$ENDIF FPC},
    ListingIO,
    AutoMatrix, AutoD, AutoColor32,
    NSColor, Raster_3PEG_type, Raster_3PEG_Image;

type
  TRasterIO = class(TThread)
  private
    FSerialized: Boolean;
  public
    FMemoryStream: TMemoryStream;
    FRasterizationWork: TRasterizationWork;
  end;
  TSerialized = Boolean;
  TMemoryStream = TMemoryStream;
  TRasterizationWork = TRasterizationWork;
  TSerialized = Boolean;
  TMemoryStream = TMemoryStream;
  TRasterizationWork = TRasterizationWork;
  TSerialized = Boolean;
  TMemoryStream = TMemoryStream;
  TRasterizationWork = TRasterizationWork;

interface

type
  TRasterIOProcessor = class;
  TRasterIO = class(TThread);
  TSerialized = Boolean;
  TMemoryStream = TMemoryStream;
  TRasterizationWork = TRasterizationWork;

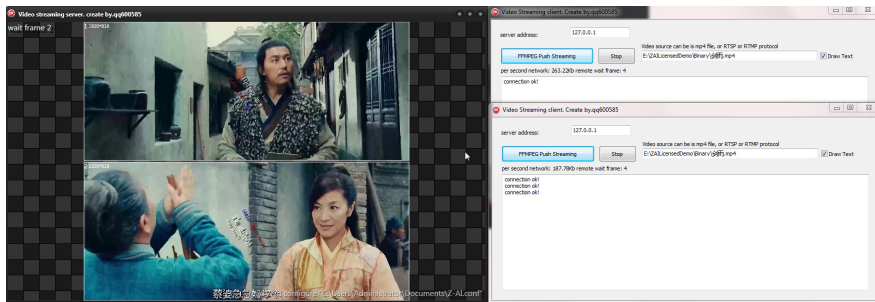
implementation

{$R *.res}

end.
```

另一个是 Demo,这是由服务端+客户端组成,客户端会解码 mp4 文件视频,重构视频,推流视频,服务器端会一次重新解码推流过来的视频。使用的硬件加速技术是 cuda。

特别说明:我工作用的开发构建系统都是 windows7,分发时可以切换到 windows10,简单来说,我在分发程序时,都会用 win10 重新构建一次。因为微软停止了 win7 支持



By.qq600585