

图像序列的视频流开发及网络发送的实现

刘怀广, 陈幼平, 谢经明, 彭向前

(华中科技大学 机械科学与工程学院, 湖北 武汉 430074)

摘 要: Raw 文件是一种原始图像, 对其序列文件视频流的开发是获取图像信息的重要手段. 利用 DirectShow 技术开发出源过滤器和发送过滤器, 实现了 Raw 序列文件的读取和发送. 文件映射和中间缓存技术的应用提高了文件读取和视频流的传输效率.

关键词: 序列文件; DirectShow; 视频流

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2011)01-0001-06

Video Stream Developed and Net Send Realized of Image Sequences

LIU Huai-guang, CHEN You-ping, XIE Jing-ming, PENG Xiang-qian

(College of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: The Raw files are raw data images. Video stream developed of Raw file sequences is an important method to get image information. With technology of directshow, the source filter and net send filter are developed to read and send the Raw file sequences. File mapping and middle memory improve the efficient of file reading and transmission of file sequences.

Key words: file sequences; DirectShow; videostream

1 引言

机载成像设备已广泛应用于军事国防、资源探测、灾害预报等领域. 对有些成像设备产生的原始图像 Raw 序列文件的处理, 是获取图像信息的重要手段. 目前, 对于 Raw 图像序列, 缺乏专门的信息处理机提供相关支持, 为适应图像信息处理的自动化与智能化的发展需要, 开发一套面向对象的 Raw 图像序列信息处理机, 对于挖掘图像信息, 充分发挥成像系统的功能具有重要的意义.

本文主要针对 Raw 图像序列文件, 利用 Microsoft DirectShow 技术, 开发一种信息处理机的服务器, 通过控制媒体流, 用户可以以指定的帧速率进行视频预览, 并以帧为基准, 实现图像数据和控制指令的同步发送, 同时接收下位机反馈指令, 还可以对关键帧进行回放和播放窗口的任意缩放.

2 整体方案

原始图像 Raw 文件是仅仅具有图像数据而不包含任何图像信息的文件, 要实现其序列文件的定帧预览和无失真缩放, 需要开发一种高效的视频流. DirectShow 框架是微软提供的在 Windows 平台上处理各种格式媒体文件的回放、音视频采集等高性能要求的多媒体应用接口技术^[1]. 其主要基于 COM 技术, 框架灵活、易于扩展、效率高, 是目前视频流开发的首选.

DirectShow 依据的是一种叫做过滤器(Filter)的 COM 单元, 每个过滤器都在视频流链路中具有简单而又明确的任务(如读取文件, 编码、解码等). 按功能来分, 过滤器主要分为源过滤器(Source Filter)、转换过滤器(Transform Filter)和渲染过滤器(Rendering Filter). 过滤器通过输入输出引脚(Pin)相互连接在一起组成一个过滤器图表(Filter

Graph),从而共同完成特定的视频处理功能.应用程序通过过滤器图表管理器来管理过滤器图表,利用相应的 API 接口,来实现对视频流的控制.视频流播放过程中,Filter 之间传递的是一个称为 Sample 的内存块,每个内存块包含时间信息和数据内容,并且由内存分配器统一管理.

本文采用 DirectShow 技术,开发了两个过滤器:RawsetSource 和 NetSender,用于完成序列文件的读取及数据的发送.

服务器连接后的过滤器图表如图 1 所示.图中共有四个过滤器,第一个过滤器(RawsetSource)和最后一个过滤器(NetSender)是要实现的过滤器.第一个过滤器负责读取图像序列文件并传递给下一级,中间的 Infinite Pin Tee Filter 是 windows 自带的一个中间过滤器,用于把输入端口带有时间戳的视频流拷贝任意份(本文是 2 份)分别发给每个输出引脚.输出引脚 1 用于视频的预览,其后连接的也是个 windows 自带的 Video Render 实现视频流的渲染.引脚 2 连的是发送过滤器,通过 TCP/IP 实现视频流的网络发送.



图 1 处理机服务器的过滤器图表

3 源过滤器的开发

信息处理机服务器所要实现的功能主要是预览和发送.要实现这两个功能,首先是数据源的获取.由于 Windows 没有过滤器支持 Raw 序列文件,因此需要开发相应的源过滤器(RawsetSource).

3.1 框架分析

DirectShow 中存在两种视频流模式:推模式和拉模式.推模式的源过滤器主要用来实时源的获取,比如网络数据,这种方式对于实时源不能进行回放,但是对于文件产生的视频流可以方便地进行控制(比如加时间戳、实现寻帧等);拉模式主要用来特定格式文件的回放,这种方式的视频流由下一级索取,因此不需要关心时间戳的问题^[2],但是对帧控制存在着暗箱操作,不方便进行主动寻帧.

本文的源过滤器负责读取 Raw 序列文件,考虑到寻帧的方便,视频流采用推模式,基类采用 CSource 和 IFileSourceFilter.其中 CSource 主要实现了推模式源过滤器的框架,而 IFileSourceFilter

引入了文件路径以及信息的接口.

实际上过滤器只是一种流的外壳,其包含的连接引脚 Pin 才是完成各种功能的组件.通常,源过滤器只含有一个输出引脚(OutputPin),用于向下传递数据.推模式的 Pin 继承于 CSourceStream,其输出 Pin 上的工作线程已经完成了大部分数据传递的工作(图 2),但是针对不同的开发要求还需要对一些虚函数进行重载和实现.

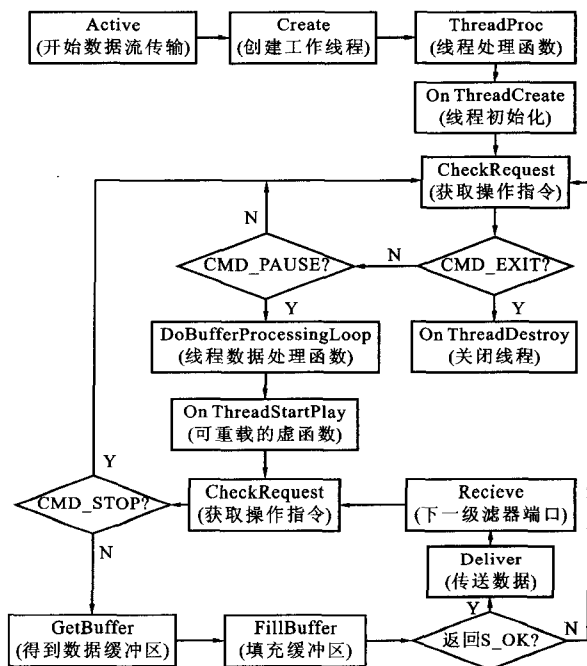


图 2 CSourceStream 的数据传输过程

图 2 所示为 CSourceStream 传输线程被激活的过程,每当过滤器图表由停止状态转换为暂停或运行(播放)状态时,过滤器图表管理器就会调用源过滤器的 Active()函数准备进行视频流的传输.图中的 CMD_EXIT、CMD_STOP、CMD_PAUSE 分别对应着线程的退出、暂停和运行.对于 CSourceStream,CMD_PAUSE 和 CMD_RUN 并没区别,都会引起线程的启动.图中的 OnThreadStartPlay 是个可以重载的虚函数,用于完成一些数据的初始化,将在寻帧和变速预览时起到作用.

需要说明的是,CSourceStream 传输线程的 Stop()并不表示线程的退出,只是设置 CMD_STOP 标志,让线程暂停传输数据.而真正让线程退出的是由 Inactive()调用的 Exit()函数.当应用程序通过 Filter Graph Manager 调用 IMediaControl 接口的 Stop()时,会通知每个过滤器 Inactive(),并退出传输线程.

RawsetSource 的输出引脚 Pin 需要完成的任

务主要为:(1)过滤器连接时媒体类型的协商和内存分配器的协商;(2)完成数据的读取和传递;(3)实现帧的定位;(4)实现播放帧速率动态改变。

针对(1)和(2),CSourceStream 已经有相应的虚函数供重载,为了完成(3)和(4)还需要引入 CSourceSeeking 基类.CSourceSeeking 继承于 IMediaSeeking 接口,用于媒体的随机定位和回放速率的设置.其含有三个比较重要的参数:m_rtStart,m_rtStop 和 m_dRateSeeking,分别用来记录定位时的开始时间、结束时间和播放速率。

源过滤器输出流(本文为 CRawStream)重载的主要函数如表 1 所示.Raw 序列文件由 8 位灰度图的原始数据组成,为了兼容新旧渲染过滤器,本文提供了非压缩 8 位和 32 位两种媒体格式,8 位的提供调色板,32 位的不提供调色板.实现代码如下:

```
VIDEOINFO* pvi = (VIDEOINFO*) pmt->Alloc-
FormatBuffer(sizeof(VIDEOINFO));
switch (iPosition)
{
case 0://8 bits
    pvi->bmiHeader.biClrUsed = 256; //调色板数
    pvi->bmiHeader.biBitCount = 8; //像素位数
    for(int i = 0; i < 256; i++) //填充调色板
    {
        pvi->TrueColorInfo.bmiColors[i].rgbRed = i;
        pvi->TrueColorInfo.bmiColors[i].rgbBlue = i;
        pvi->TrueColorInfo.bmiColors[i].rgbGreen = i;
        pvi->TrueColorInfo.bmiColors[i].rgbReserved
=0;
    }
    break;
case 1://32 bits
    pvi->bmiHeader.biBitCount = 32;
}
//填充媒体位图信息头
pvi->bmiHeader.biCompression = BI_RGB;
pvi->bmiHeader.biSize = sizeof(BITMAPINFO-
HEADER);
//位图宽度(初始化时传入)
pvi->bmiHeader.biWidth = m_iImageWidth;
//位图高度(初始化时传入)
pvi->bmiHeader.biHeight = m_iImageHeight;
pvi->bmiHeader.biPlanes = 1;
pvi->bmiHeader.biSizeImage = GetBitmapSize
(&pvi->bmiHeader);
pvi->bmiHeader.biClrImportant = 0;
//默认播放间隔(以百纳秒为单位)
```

```
pvi->AvgTimePerFrame = 400 000;
//设置主体类型
pmt->SetType(&MEDIATYPE_Video);
//设置次要类型
pmt->SetFormatType(&FORMAT_VideoInfo);
//非压缩方式
pmt->SetTemporalCompression(FALSE);
```

表 1 源过滤器输出流 CRawStream 重载函数

重载函数	函数说明
父类 CSourceStream	
GetMediaType	提供输出 Pin 支持的媒体类型
CheckMediaType	对媒体类型进行检查
SetMediaType	连接后,重新设定媒体类型
DecideBufferSize	设置内存分配器所含 Sample 的数量和大小
FillBuffer	填充 Sample 的内存区
OnThreadCreate	初始化函数,线程创建时调用
OnThreadStartPlay	循环初始化,Pause()时被调用
父类 CSourceSeeking	
ChangeRate	设置回放速率
ChangeStart	Seek 开始后被调用
ChangeStop	Seek 结束后被调用

CheckMediaType 是用来检查是否支持下游过滤器(Infinite Pin Tee Filter)输入 Pin 提供的媒体类型.为了保证在视频流传输中,传递的数据块与单帧位图大小一致,还需要在检查 Sample 是否为固定大小:

```
if((*(pMediaType->Type()) != MEDIATYPE_
Video) || !(pMediaType->IsFixedSize()))
    return E_INVALIDARG;
const GUID* SubType = pMediaType->Subtype();
if((*(SubType != MEDIASUBTYPE_RGB8)&&(*
SubType != MEDIASUBTYPE_RGB32))
    return E_INVALIDARG;
```

内存分配器的协商,一般由下游的过滤器提供,源过滤器只需要重载 DecideBufferSize 来确定具体参数(大小和数量).一般一个 Sample 大小是按照整帧图像的大小进行分配,为了尽可能地保证渲染的视频图像的实时性,内存分配器中只设定一个 Sample。

视频流播放的过程就是传输线程不断地调用 FillBuffer 来填充 Sample 的过程.因此,在 Fill-Buffer 中,需要不断地读取序列文件的内容,填充到

Sample 的内存块中,同时写入时间戳以供渲染使用.当 FillBuffer 返回 E_FALSE 的时候,源过滤器有理由认为文件播放完毕.因此,为了保证序列文件指针偏移的准确性,本文定义了一个 64 位的变量 m_llFramePos 用来记录当前帧的序号,每调用 FillBuffer 一次 m_llFramePos 就会加一,当 m_llFramePos 达到最大帧时,返回 E_FALSE.填充开始前,检查 Sample 内存大小(32 位的媒体类型需要除以 4 换算成 8 位)与单帧图像大小($m_iImageWidth \times m_iImageHeight$)是否一致,如果小于单帧图像(尽管几率很小),则以 Sample 的大小为准进行读取,否则以单帧图像的大小进行读取.当协商的媒体类型为 8 位,就直接把读取的内容复制到 Sample 中;如果是 32 位的,需要把读取数据的 1 位分别赋给 Sample 的 3 位,并把第 4 位的 alpha 通道置 0.每帧填充后,用 SetTime()设置 Sample 的时间戳,用 SetMediaTime()设置帧序号供 NetSender 使用.

3.2 帧操作的实现

服务器要实现的帧操作主要是寻帧和定帧预览.寻帧就是定位到任意指定的某一帧,定帧预览就是按照指定的帧速率进行回放,本文实现的帧速率范围为 5~30 f/s.

对于 DirectShow 支持的标准媒体格式,可以通过查询过滤器图表管理器的 IMediaSeeking 接口并设定 SetTimeFormat (&TIME _ FORMAT _ FRAME)来实现帧定位,但对于非标准媒体类型,需要手动实现.一般当应用程序向过滤器图表管理器发出帧操作后,过滤器图表管理器从 Renderer Filter 开始自下向上寻找实现 IMediaSeeking 接口的过滤器.对于拉模式,IMediaSeeking 接口主要实现在 Parser 或 Splitter Filter 上,而推模式主要实现在源过滤器上.Raw 图像序列文件是非标准媒体类型,为了完成帧操作,需要实现 IMediaSeeking 接口,因此 RawsetSource 成了 CSourceSeeking 类.

在过滤器进行帧操作前,线程有可能正在等待填充好的 Sample,为了防止帧定位时产生线程死锁,过滤器的输出 Pin 需要进行以下过程^[3]:

- (1) 调用 IPin::BeginFlush,开始线程数据的清理;
- (2) 挂起数据传输线程;
- (3) 调用 IPin::EndFlush 结束清理;
- (4) 重新启动传输线程;
- (5) 调用 IPin::NewSegment 把规定时间段内的数据帧传输下去;

(6) 在起始帧上设置 m_bDiscontinuity 标志.

IPin::BeginFlush 是告诉过滤器丢掉未处理的数据,并退出阻塞,这时下游的过滤器将拒绝接收上游的任何 Sample.IPin::EndFlush 是通知下游过滤器要进行新的 Sample 传递,结束拒绝,这样重启线程后就可以利用 IPin::NewSegment 传递指定时间范围的数据帧,产生新的视频流.

3.2.1 寻帧的实现

寻帧时,过滤器图表管理器将调用 SetPositions 传递一个起始时间 m_rtStart 和终止时间 m_rtStop 给源过滤器,源过滤器首先进行数据清理工作,然后根据起始时间换算成帧,重新把 Raw 序列文件的指针移动到相应起始帧的位置,从当前位置继续传输视频流.

源过滤器通过重载 ChangeStart 来完成上述工作,基本过程如图 3 所示.

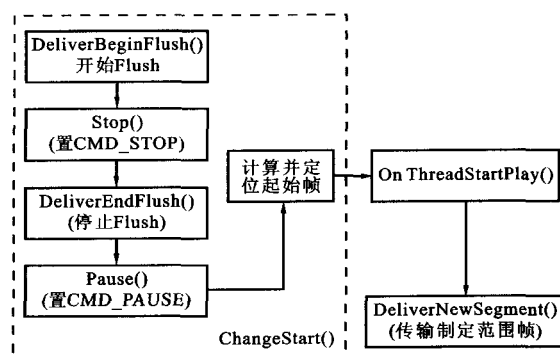


图 3 CReadStream 寻帧过程

结合图 2 的数据传输过程,从图 3 中可以看出,Stop()函数会令传输线程退出数据传输循环,Flush 完成后,重新启动循环(设置 CMD_PAUSE 或 CMD_RUN),然后计算并定位到起始帧.最后由传输线程调用 OnThreadStartPlay()设置一些初始化操作(比如单帧时间戳的归零、设置 m_bDiscontinuity 标志等),并实现 NewSegment()传输.

新的视频流开始以后,当过滤器每传递一个 Sample,就会把时间戳与结束时间 m_rtStop 进行比较,如果到达结束时间,就调用 ChangeStop()完成新视频流的清理工作.本文寻帧时结束帧并不知道,为了防止新视频流过早结束,应用程序在启动寻帧时不应设置结束时间:

```

SetPositions (&-position,
              AM _ SEEKING _ AbsolutePositioning,
              0, AM _ SEEKING _ NoPositioning);
  
```

起始帧的计算根据 SetPositions 传递的起始时间 position 来计算,其以百纳秒为单位。比如,应用程序要定位到第 5 帧,position 的值为 $5 \times iDefaultRepeatTime$, $iDefaultRepeatTime$ 是帧间隔时间,要和源过滤器在媒体协商时的设定的间隔时间保持一致,这里取 400 000。position 传递到源过滤器以后,会赋值给 $m_rtStart$,然后还要解析成序列文件的偏移字节数:

$$dwFileOffset = m_rtStart / iDefaultRepeatTime \times m_iImageWidth \times m_iImageHeight$$

3.2.2 定帧预览的实现

定帧预览,相当于从当前位置进行快进或慢放。应用程序通过 `IMediaSeeking::SetRate()` 来设定回放速率。过滤器图表管理器在执行 SetRate 后,会调用源过滤器的 SetRate 设置 $m_dRateSeeking$,然后调用 ChangeRate()通知过滤器速率已变,进行数据清理工作,并令视频流的起始时间戳归零。

因为 Filter Graph 末端的 Video Renderer 会依据 Sample 的时间戳来设定单帧图像的显示时间,因此速率设定后,只需要在 FillBuffer 中,把 Sample 的时间戳间隔时间变为相应的 $1/m_dRateSeeking$,就可以达到改变回放速率的目的。

3.3 播放效率的改进

最快的预览速率,很大程度上取决于 Raw 序列文件的单帧读取速度。实践证明,对于用 Win32 API 的 CreateFile 创建的文件内核对象,单帧图像大小为 512×512 时,最高预览速度可以达到 40 f/s(Intel P4 1 GB 内存,双核 CPU 1.6 GHz 下同)。但是,当单帧图像大小为 1024×1024 时,最高预览速度只能达到 8 f/s,并且存在着积累延时。为了提高大图像的预览速度,本文采用了文件映射+中间缓存的方式。工作原理如图 4 所示。

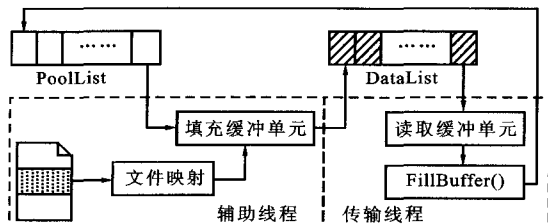


图 4 文件读取过程

文件映射就是把文件内容与进程的虚拟地址空间关联起来,然后通过文件视图映射来读写文件内容。对于大文件是一种比较有效的操作方式,并可以在一定程度上加快读取速度^[4]。中间缓存是定数量的内存块,用于存储读取的文件内容,这些内存块

由两个链表进行管理:PoolList 和 DataList。PoolList 管理未填充的内存块,DataList 管理已填充的内存块。

对于大文件(>500 MB),本文采用部分文件视图映射的方式。首先创建一个只读形式的文件映射对象,并关联整个文件,然后用文件视图映射的部分区域(每次映射 64 MB,当不够 64 MB 时以文件大小为准)。用 64 位的变量 $m_llFileOffset$ 来记录文件视图起始位置对应的偏移量。当要读取的文件内容超过当前视图映射的部分,需要重新建立一个文件视图,并更新 $m_llFileOffset$ 。

程序开始后,所有的内存块都在 PoolList 中等待填充,DataList 为空。当辅助线程启动以后,不断取得 PoolList 的头部内存块,通过文件视图映射得到内存指针,来填充每个内存块。填满后,放在 DataList 的尾部,等待读取。PoolList 为空时,线程就会等待。传输线程启动以后,会不断的调用 FillBuffer 函数取得 DataList 的内存块数据,依次从头部开始读取。每读完一个内存块,把其从 DataList 中去掉,并置于 PoolList 的尾部。这样就形成一个图像数据缓冲区,避免了文件 I/O 操作造成的延时影响,从而提高了播放效率。实践表明,这种方式读取大图像很有效,对于大小为 1024×1024 单帧大图像,可以达到 30f/s,而无延时现象。

4 发送过滤器的开发

发送过滤器(NetSender)在过滤器图表中是另一个终端过滤器,负责把 Infinite Pin Tee Filter 传递的图像数据发送给下位机进行处理(比如定位、识别),并接收下位机的反馈信息(如目标位置)。由于不需要向下传递视频流,NetSender 只含有一个输入 Pin(CNetInputPin)。

CNetInputPin 继承于 CBaseInputPin,其实现了推模式下的 IMemInputPin 接口。上一级过滤器的输出 Pin 可以通过调用 `IMemInputPin::Receive()` 来传递数据^[5]。因此,CNetInputPin 需要重载的关键的函数就是 Receive(),另外还有连接时的函数 GetMediaType()、CheckMediaType()等。

在 CNetInputPin::Receive 中,首先要解析传递来的 Sample,得到帧序号和图像数据,然后通过 TCP/IP 把数据发送出去。从图 1 中可以看出,Infinite Pin Tee Filter 把 RawsetSource 传递来的 Sample 拷贝了两份给输出 Pin,每一份都会增加 Sample 一次引用计数,当两个输出 Pin 的 Receive()都返回

S_OK 时, Sample 引用计数变为 0, Sample 才会被释放并返回 Allocator, 视频流才会继续运行. 因此, 如果 CNetInputPin::Receive() 不能及时的处理数据并返回 S_OK, 可能造成 Video Renderer 端预览的延时. 为了不影响预览的效果, 本文采用与 RawsetSource 类似的中间缓存的方式(图 5).

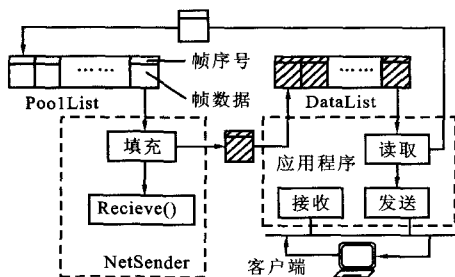


图 5 图像帧发送过程

中间缓存的每个内存单元以单帧图像+帧序号的结构体形式进行分配, 其操作过程与源过滤器类似. 应用程序启动后, 首先利用 WinSock 技术建立一个服务器, 等待客户端连接. 当有客户端连接后, 就会启动两个线程分别对应于发送和接收. 发送线程不断的从 DataList 的头部读取内存块, 并根据帧序号读取指令文件的相关操作, 连同图像数据一并发送, 然后清空内存单元并放回 PoolList 尾部. 而 CNetInputPin::Receive() 负责从 PoolList 头部取出空的内存块进行帧序号和图像数据的填充, 然后放入 DataList 尾部. 如果 PoolList 为空, 此时可能发送线程阻塞或未启动, 为了保证重新启动后发送的是实时帧, 取 DataList 尾部的内存单元进行重新填充.

接收线程用于接收客户端反馈的信息. 为了把反馈内容和帧对应起来, 反馈信息采用帧序号+内容的形式, 内容一般为目标的坐标、大小、方向角等. 接收线程每接收一条内容就会存入反馈文件, 以备后续图像调整和融合使用.

5 开发实例

图 6 所示为信息处理机服务器的一个开发界面. 由于 Raw 序列文件是原始图像文件, 用户选择文件后还要输入单帧图像信息(长度和宽度). 应用程序会把文件路径和图像信息传递给源过滤器(RawsetSource), 然后根据图 1 搭建的 Filter Graph 进行播放和发送.

图中播放的是一个单帧图像大小为 $1\,004 \times 1\,004$, 共 600 帧的序列图像. 如果用 ReadFile() 直

接读取单帧数据, 由于不是 512 的整数倍, 效率很低, 只能达到 6 f/s 的播放速率, 而采用文件映射+中间缓存后, 很容易达到 25 f/s 的速度.

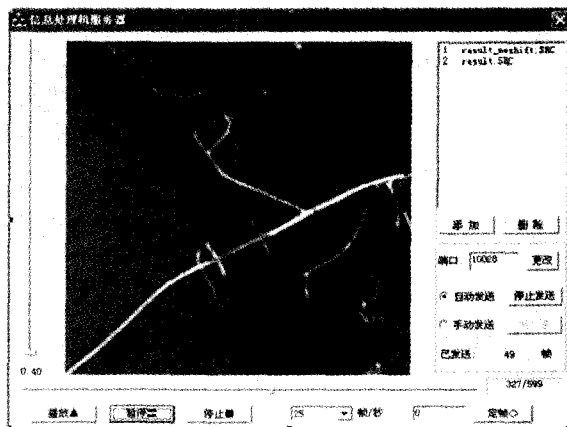


图 6 Raw 序列文件视频流实现界面

6 结束语

原始图像 Raw 文件是目前很多成像系统的文件格式, 对其序列文件信息处理机的开发是图像处理中一个比较有意义的应用. 用 DirectShow 技术开发序列文件的视频流, 可以很方便地实现图像帧的变速预览和网络发送. 中间缓存是协调多线程之间大块数据处理速度的有效方法, 利用双链表进行管理, 可以很大程度上提高视频流的传输效率. 与传统的 Win32 API 相比, 文件映射+中间缓存的方式提高了大文件的数据读写和处理速度.

参考文献:

- [1] Dasu A, Panchanathan S. A survey of media processing approaches[J]. IEEE Transactions on Circuit and System for Video Technology, 2002, 12(8): 1-13.
- [2] Jin Li, Yi Cui, Bin Chang. Peer Streaming: design and implementation of an on-demand distributed streaming system with digital rights management capabilities[J]. Multimedia Systems, 2007, 13(3): 173-190.
- [3] Microsoft Corporation. Microsoft DirectX 9.0 SDK document[M]. Seattle: Microsoft Corporation, 2002.
- [4] 周金萍, 徐丙立, 姜小光. Windows 系统编程[M]. 北京: 人民邮电出版社, 2002.
- [5] 张帆, 李波. 医学影像视频流网络传输[J]. 微电子学与计算机, 2008, 25(8): 25-28.

作者简介:

刘怀广 男, 博士研究生. 研究方向为机器视觉、模式识别和智能控制.