数字视频音频实时同步传输探索与实现

郑阿奇

(南京师范大学数学与计算机科学学院, 210097, 南京)

[摘要]　在 Visualc++ 6.0 平台上开发视频和音频实时传输和回放, 对视频音频数据实时同步传输策略和算法进行探索与实现.

[关键词]　视频和音频, 实时, 同步, 传输

[中图分类号] TN911.72;　[文献标识码] B;　[文章编号] 1672-1292(2002)03-0016-05

0　引言

现代信息技术的发展, 人们对视频音频多媒体通信的需求越来越大, 视频和音频实时同步传输具有实用意义. 视频和音频实时同步传输主要解决视频音频信号同步采集、压缩、同步传输和信息接收以及同步播放等问题, 其最终目的是实现客户端的图像和声音与采集现场同步播放　视频和音频同步就是看到的图象和听到的声音时间差小, 实时就是要使播放端与采集的延时小, 使人不致于有明显的感觉. 下面以笔者在 Visual C++ 6.0 下开发的视频和音频实时同步播放系统的实例进行这方面探讨.

1　视频和音频实时同步传输

1.1　音频视频信号的采集

视频信号的采集可用摄像头连视频采集卡实现, 在要求不高的场合也可用 USB 摄像头直接连计算机的 USB 口; 声音信号用麦克风连声卡采集; 在 Visual C++ 6.0 借助于 MicroSoft VFW 类库进行开发.

采集端首先创建捕获窗口, 设置视频流和音频流的回调函数. 然后与设备相连, 再设置浏览速度和 preview 浏览方式. 部分设置程序如下:

```
capCreateCaptureWindow( 捕获窗口 ,WS_CHILD|WS_VISIBLE,0,
                        0,160,120,pParentHwnd,int(0))
capSetCallbackOnVideoStream(m_caphwnd,procVideo);
capSetCallbackOnWave Stream(m_caphwnd,procWave);
capDriverConnect(m_caphwnd,0);
capPreviewRate(m_caphwnd,33);
capPreview(m_caphwnd,TRUE);
```

1.2　视频音频同步方案的选择

由于系统将视频音频采集的数据流分别放入不同的缓冲区, 任一缓冲区采满, 系统调用回调函数, 在回调函数中读取视频数据或音频数据, 视频数据太大需要先进行压缩处理. 在这种方式下, 为了保证客户端的视频和音频的同步可有两个基本办法.

一是将视频数据与音频数据合成一个包一起发送. 这种方案表面上容易同步, 但实际存在一些问题. 在数据采集时如以音频为基点先将视频数据放入一个事先准备好的缓冲区中, 在一帧音频数据采集

收稿日期: 2002-01-21

作者简介: 郑阿奇, 南京师范大学数学与计算机科学学院副教授, 研究领域为数据库、网络与多媒体.

好后得到存放该缓冲区中的所有数据一起发送,到达客户端后同时进行视频的显示与音频的回放.但由于视频的数据较大,假如采集的视频格式为 352 288 24,那么每 1 帧的数据有 300k,经 JPEG 压缩约有 8k.而音频的数据要小得多,如果采集的音频格式为 8 位单声道,那么 1s 数据量是 11k,如果每 s 音频采集为 10 帧,那么每 1 帧为 1.1k.如果视频的采集的速度为每 s 20 帧,这样每一个包就有 1 帧音频和 2 帧视频,一次传输数据量太大,可能还会发送失败.还有它并不能十分准确的在每 1 帧音频数据采集好后,2 帧视频刚好就采集好了,很可能有的包中 1 帧,有的包中 2 帧或 3 帧的视频数据的情况,客户端很难编程.而每 s 10 帧音频的速度好像慢了一点,不能保证数据的实时性.如以视频为基点,在采集 1 帧视频数据的过程中将音频数据依次追加到一个事先准备好的缓冲区中,1 帧视频数据采集好后取得该缓冲区的中的所有数据一起发送,到达客户端后同时进行视频的显示与音频的回放.这种方式可提高音频采集的速度,从而更好保证数据的实时性,但是实际中由于音频采集的速度过快,每 1 帧之间的间隔太小,到缓冲区中取音频数据的过程中可能另 1 帧数据又产生,有可能会覆盖原来的数据,而得到错误的数据,实际调试中这种情况发生的机率比较高,得到的音频数据在回放中会有很刺耳的声音.

二是音频和视频数据分别传送.看起来这种方案没有同步,其实不然.可以将音频的采集的速度提高,这样每 1 帧音频的数据就很小.实际我们使用每 s 25 帧,如果音频的格式为 8 位单声道,那么每 1 帧的数据只有 441 个字节,这样就既可以保证数据的实时性,又可以便于网络传输.音频数据流缓冲区中数据变成音频数据包直接交发送线程发送.视频数据流缓冲区采集填满后系统就会自动调用视频数据回调函数,在该函数中将视频数据流缓冲区的首地址作为压缩数据源地址参数,同时填入数据长度和数据压缩所需其它参数后调用数据压缩线程.在数据压缩线程中,将当前 BMP 格式视频数据帧压缩 JPEG 格式视频数据帧,压缩结束后,将 JPEG 格式视频数据帧数据变成视频数据包直接交发送线程发送.音频和视频回调和压缩线程压缩函数部分代码如下:

```
  视频回调函数
LRESULT CALLBACK Streamvideocallback(HWND hwnd3,LPVIDEOHDR lpdata3)
{   pinfo= new SHOWINFO;
    pinfo- > bitmapinfo= showinfo.bitmapinfo;
    pinfo- > pbuf= lpdata3- > lpData;
      启动压缩线程
    AfxBeginThread(compress,(LPVOID)pinfo);
    return TRUE;
}
  音频回调函数
LRESULT CALLBACK wavestreamcallback(HWND hwnd,LPWAVEHDR lpdata)
{   m _socket.send(lpdata- > lpData,lpdata- > dwBufferLength);}
  压缩线程压缩函数
int Bmpbuftojpegbuf(unsigned char * pbmpbuf,unsigned char * pbuf)
{
     初始化...填入参数...
     数据压缩
  if(ijlWrite(&jcprops,IJL _JBUFF _WRITEWHOLEIMAGE)! = IJL _OK
      {ijlFree(&jcprops);return  0;}
  int l= jcprops.JPGSizeBytes;
     释放资源
  ijlFree(&jcprops);return l;
}
```

音频数据包和视频数据包可在包头进行标识以示区别, 但实际上, 由于两个数据包的长度差别很大, 所以以包的大小也就能够区分. 由于系统采集视频音频是相对同步的, 采满一帧就立即发送, 客户端接收到音频数据包不必与视频数据包协调而立即播放, 客户端接收到视频数据包不必与音频数据包协调而立即进行解压显示, 客户端音视频的回放就能同步, 即使由于网络传输出错等原因而丢一些帧, 仅可能会影响一点实时性.

1.3 数据的实时性

笔者在系统中采用了帧发送, 由于视频的数据量过大, 因此数据首先必须进行压缩, 压缩后发送大大减少了延迟, 网络协议考虑到在广播或多播的情况采用了 UDP 协议. 音频流的回放采用了低级的音频回放函数 waveOut. 如果每一帧数据到达都利用 waveOut 打开设备再播放, 然后释放设备, 由于每一帧音频数据很小, 加上打开设备、关闭设备都要花时间, 结果发出的声音失真. 如果将每一帧的数据加大, 这样发出的声音是好了但延迟又太长, 实时性降低. 考虑到 waveOut 函数播放一段缓冲区时在一开始向声卡写入数据就返回的特点, 可以事先开辟一块缓冲区, 缓冲的大小为 11k(8 位单声道 1s 的数据). 当第一帧音频数据到达后, 将其拷贝到缓冲区的最前面, 并启动音频设备开始播放这段缓冲区(大小为 11k), 在这过程中, 第二, 第三 帧数据陆续到达, 依次拷贝到这段播放缓冲区, 这样在开始播放的时候并没有 1s 的数据, 而是在播放的过程中不断地填入了数据. 考虑到实际中网络的不确定因数, 可以在第二或第三帧启动音频设备开始播放, 而人是感觉不出来的. 音频回放部分代码如下:

```
woid play()
{
    int i= waveOutPrepareHeader(hwaveout, &head, sizeof(WAVEHDR));
    if(i!= 0)
        return;
    i= waveOutWrite(hwaveout, &head, sizeof(WAVEHDR));
    if(i!= 0)
    {
        waveOutUnprepareHeader(hwaveout, &head, sizeof(WAVEHDR));
        return
    }
}
```

添加一个接收类, 基类为 CasyncSocket, 重载 onReceive(), 代码如下:

```
onReceive()
{
    int Count= Receive(paudioBuf, sizeof(paudioBuf));
    if(Count= = SOCKET_ERROR)
        return;
    memcpy(&pwavebuf[b* Count], pbuff, Count); b--;
    if(b= = 0)
    {
        head.dwBufferLength= 11025;
        head.lpData= (LPSTR)pwavebuf;
        head.dwFlags= 0L;
        head.dwLoops= 0L;
        play(); b= WAVE_TIMES;
    }
```

```
}
  在程序的开始部分加入下列代码:
WAVEFORMATEX waveformat;
  设置成 8 位单声道的设备
......
  打开声音设备
ASSERT(! waveOutOpen(& hwaveout, WAVE _ MAPPER, & waveformat, unsigned long
  (waveOutProc), NULL, CALLBACK _ FUNCTION));
  开辟 1t 音频数据缓冲
unsigned char * pwavebuf= new unsigned char[ 11025];
  设置无声缓冲
memset(pwavebuf, 128, 11025);
```

1.4 多线程的应用

用好多线程对于提高系统的实时性与同步有重要的意义.例如,在视频回调函数中启动压缩线程即可返回,在压缩线程压缩视频数据时系统又有可能调用音频回调函数,而音频回调中直接启动发送线程,发送音频数据.这样,不仅简化了编程,提高了程序的运行效率,同时,还提高了网络的传输效率.在客户端,视频数据的解压与显示,音频的回放等也使用不同的线程.采用多线程,其最终目的是更容易实现音频视频的同步和实时.下面列出压缩数据线程与发送数据线程的部分代码:

```
  发送线程
UINT sendthread(LPVOID parm)
{
    向多播地址发送数据(可以换上广播地址来广播)
  SENDINFO m _ psendinfo= (SENDINFO * ) parm;
  if(psendinfo- > semap= = 1)
    pSocket- > SendTo(pbuff1, psendinfo- > bufsize, 3000 广播 多播地址 );
  else
    pSocket- > SendTo(pbuff2, psendinfo- > bufsize, 3000 广播 多播地址 );
    return 0;
}
CJpeg1 m _ jpg;
CSemaphore semap(1, 1);
UINT compress(LPVOID parm)
{
  SHOWINFO * pinfo= (SHOWINFO * )parm;
    使压缩资源独占
  if(! semap.Lock(0))
  {
    delete pinfo;
    return 0;
  }
  if(! a)
  {
    sendbufsize= m _ jpg.Bmpbuftojpegbuf(pinfo- > bitmapinfo, pinfo- > pbuf, pbuff1);
```

```
            a++;
            psendinfo=new SENDINFO;
            psendinfo->semap=1;
            psendinfo->bufsize=sendbufsize;
            AfxBeginThread(sendthread,(LPVOID)&psendinfo);
        }
        else
        {
            sendbufsize=m_jbp.Bmpbuftojpegbuf(pinfo->bitmapinfo,pinfo->pbuf,pbuff2);
            psendinfo=new SENDINFO;
            psendinfo->semap=2;
            psendinfo->bufsize=sendbufsize;
            AfxBeginThread(sendthread,(LPVOID)&psendinfo);
            a--;
        }
        delete pinfo;
        semap.Unlock();
        return
    }
```

2 结束语

随着网络的发展,动态压缩技术的提高,视频和音频实时同步传输将有更大的发展和更多的应用领域.在这方面的应用研究具有广阔的应用前景.

[参考文献]

[1] D J Kruglinski. Visual C++ 6.0技术内幕[M]. 北京:希望电脑电子出版社,1999.
[2] Anthony Jones. 网络编程技术[M]. 北京:机械工业出版社,2000.
[3] 精英科技. 视频压缩与音频编码技术[M]. 北京:中国电力出版社,2001.

The Research and Realization about A Real-time Synchronism Transmission of Audio and Video

Zheng Aqi

(College of Mathematics Science and Computer Science, Nanjing Normal Univ, 210042, Nanjing, PRC)

Abstract: Real-time transmission and playback of Audio and Video are developed base on Visual C++ 6.0. Strategy and algorithm of real-time synchronism transmission of Audio and Video data are studied and realized.

Key words: Audio and Video, Real-time, Synchronism, Transmission

[责任编辑:刘健]