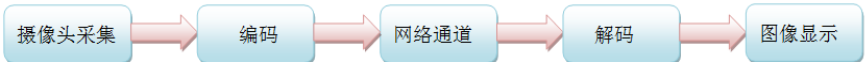
视频会议分析

视频会议的特点：低延迟，音频，视频 ，多人聊天．关于低延迟，视频会议与直播的项目是不同的．直播项目是一个 1 对多的过程，并且，是允许 5 秒左右的网络延迟的，而视频会议，一般延迟要求 1 秒以内，因为会议可以有不到 1 秒的思考时间，所以技术要求延迟小于 1 秒还是可以接受的，当然，如果延迟能达到 100ms 的话就更好了．对于音频视频方面，项目中是需要采集摄像头，采集声音，以及利用白板功能采集桌面，

视频处理流程如下:



对于视频方面的考虑，首先获取到视频数据后，是比较大的，网络传输很占用带宽，所以需要编码(压缩)，来减小数据大小．那么要使用什么编码合适呢？前面已经学到了使用 ffmpeg 编码．视频方面可以使用它将视频编码为 h264 格式数据再进行传输．

高级功能:

（1）动态调整视频的清晰度

在 Internet 上，网络速度是实时动态变化的，所以，在视频会议中，为了优先保证语音的通话质量，需要实时调整视频的相关参数，其最主要的就是调整编码的清晰度，因为清晰度越高，对带宽要求越高，反之亦然。

比如，当检测网络繁忙时，就自动降低编码的清晰度，以降低对带宽的占用。

（2）自动丢弃视频帧

同样网络繁忙时，还有一个方法，就是发送方是主动丢弃要发送的视频帧，这样在接收方看来，就是帧频 fps 降低了

对于音频方面，和上面的描述类似，也是需要编码后再传输，而音频方面的编码，可选的方式有很多,可以使用 ffmpeg 或者 sdl，这两种一般优先选择 ffmpeg 更好．而音频的处理还没有这么简单．在编码之前，采集音频数据，一般还需要考虑一些问题：回音消除，降噪，抑制声音断断续续，静音检测，混音．

文中简单描述如下:

音频的主要流程是这样的:

语音采集指的是从麦克风采集音频数据，即声音样本转换成数字信号。其涉及到几个重要的参数：采样频率、采样位数、声道数。

假设我们将采集到的音频帧不经过编码，而直接发送，那么我们可以计算其所需要的带宽要求，仍以上例：320*100 =32KBytes/s，如果换算为 bits/s，则为 256kb/s。这是个很大的带宽占用。而通过网络流量监控工具，我们可以发现采用类似 QQ 等 IM 软件进行语音通话时，流量为 3-5KB/s，这比原始流量小了一个数量级。而这主要得益于音频编码技术。 所以，在实际的语音通话应用中，编码这个环节是不可缺少的。目前有很多常用的语音编码技术，像 G.729、G.711、iLBC、AAC、SPEEX 等等。

当一个音频帧完成编码后，即可通过网络发送给通话的对方。对于语音对话这样 Realtime 应用，低延迟和平稳是非常重要的，这就要求我们的网络传送非常顺畅。

当对方接收到编码帧后，会对其进行解码，以恢复成为可供声卡直接播放的数据。

完成解码后，即可将得到的音频帧提交给声卡进行播放。

高级功能介绍:

如果仅仅依靠上述的技术就能实现一个效果良好的应用于广域网上的语音对话系统，那就太 easy 了。正是由于很多现实的因素为上述的概念模型引入了众多挑战，使得网络语音系统的实现不是那么简单，其涉及到很多专业技术。一个“效果良好”的语音对话系统应该达到如下几点：低延迟，背景噪音小，声音流畅、没有卡、停顿的感觉，没有回音。

对于低延迟，只有在低延迟的情况下，才能让通话的双方有很强的 Realtime 的感觉。当然，这个主要取决于网络的速度和通话双方的物理位置的距离，就单纯软件的角度，优化的可能性很小。

（1）回音消除

现在大家几乎都已经都习惯了在语音聊天时，直接用 PC 或笔记本的声音外放功能。当使用外放功能时，扬声器播放的声音会被麦克风再次采集，传回给对方，这样对方就听到了自己的回音。

回音消除的原理简单地来说就是，回音消除模块依据刚播放的音频帧，在采集的音频帧中做一些类似抵消的运算，从而将回声从采集帧中清除掉。这个过程是相当复杂的，因为它还与你聊天时所处的房间的大小、以及你在房

间中的位置有关，因为这些信息决定了声波反射的时长。 智能的回音消除模块，能动态调整内部参数，以最佳适应当前的环境。现在做的比较好的回音消除有，webrtc. webrtc 顾名思义是引用在网页端的技术，是谷歌最早提出的算法.

（2）噪声抑制

噪声抑制又称为降噪处理，是根据语音数据的特点，将属于背景噪音的部分识别出来，并从音频帧中过滤掉。有很多编码器都内置了该功能。

（3）抖动缓冲区

抖动缓冲区（JitterBuffer）用于解决网络抖动的问题。所谓网络抖动，就是网络延迟一会大一会小，在这种情况下，即使发送方是定时发送数据包的（比如每 100ms 发送一个包），而接收方的接收就无法同样定时了，有时一个周期内一个包都接收不到，有时一个周期内接收到好几个包。如此，导致接收方听到的声音就是一卡一卡的。

JitterBuffer 工作于解码器之后，语音播放之前的环节。即语音解码完成后，将解码帧放入 JitterBuffer，声卡的播放回调到来时，从 JitterBuffer 中取出最老的一帧进行播放。

JitterBuffer 的缓冲深度取决于网络抖动的程度，网络抖动越大，缓冲深度越大，播放音频的延迟就越大。所以，JitterBuffer 是利用了较高的延迟来换取声音的流畅播放的，因为相比声音一卡一卡来说，稍大一点的延迟但更流畅的效果，其主观体验要更好。

当然，JitterBuffer 的缓冲深度不是一直不变的，而是根据网络抖动程度的变化而动态调整的。当网络恢复到非常平稳通畅时，缓冲深度会非常小，这样因为 JitterBuffer 而增加的播放延迟就可以忽略不计了。

（4）静音检测

在语音对话中，要是当一方没有说话时，就不会产生流量就好了。静音检测就是用于这个目的的。静音检测通常也集成在编码模块中。静音检测算法结合前面的噪声抑制算法，可以识别出当前是否有语音输入，如果没有语音输入，就可以编码输出一个特殊的的编码帧（比如长度为 0）。特别是在多人视频会议中，通常只有一个人在发言，这种情况下，利用静音检测技术而节省带宽还是非常可观的。

（5）混音

在视频会议中，多人同时发言时，我们需要同时播放来自于多个人的语音数据，而声卡播放的缓冲区只有一个，

所以，需要将多路语音混合成一路，这就是混音算法要做的事情。

视频相关知识

图像采集方案

本实例使用 opencv 采集摄像头图片
为什么使用 opencv？采集摄像头的图片，方法有很多，比如 Qt 自带的 QCamera，以及开源如 ffmpeg，opencv 等. 使用 opencv 可以方便未来扩展，比如添加人脸识别的功能. 使用 ffmpeg 会涉及视频的编码和解码，这个过程较复杂，不适合初学者.

配置环境

该实例的环境采用 Qt5.12.11 + opencv4.2.0，因为编译器使用 Mingw 的类型，所以 opencv 需要单独的编译生成. 最终生成的结果包含头文件，引入库，以及动态库.
使用 cmake 编译 opencv 的过程可以参考一下如下文章:
https://baijiahao.baidu.com/s?id=1662883232064729369&wfr=spider&for=pc
https://blog.csdn.net/huihut/article/details/78701814
下面直接使用编译好的版本.
首先添加头文件路径和引入库

```
INCLUDEPATH+=C:/Qt/opencv-release/include/opencv2\
             C:/Qt/opencv-release/include

LIBS+=C:\Qt\opencv-release\lib\libopencv_calib3d420.dll.a\
      C:\Qt\opencv-release\lib\libopencv_core420.dll.a\
      C:\Qt\opencv-release\lib\libopencv_features2d420.dll.a\
      C:\Qt\opencv-release\lib\libopencv_flann420.dll.a\
      C:\Qt\opencv-release\lib\libopencv_highgui420.dll.a\
      C:\Qt\opencv-release\lib\libopencv_imgproc420.dll.a\
      C:\Qt\opencv-release\lib\libopencv_ml420.dll.a\
      C:\Qt\opencv-release\lib\libopencv_objdetect420.dll.a\
      C:\Qt\opencv-release\lib\libopencv_video420.dll.a\
      C:\Qt\opencv-release\lib\libopencv_videoio420.dll.a
```

以上的路径 C:/Qt/opencv-release 需要改为自己电脑中的目录

添加头文件

```
#include"highgui/highgui.hpp"
#include"imgproc/imgproc.hpp"
#include"core/core.hpp"
```

```
#include"opencv2\imgproc\types_c.h"
using namespace cv;
#define FRAME_RATE (15)
```

图像采集

```
#define IMAGE_WIDTH (320)
#define IMAGE_HEIGHT (240)
1.定义对象
cv::VideoCapture cap;
2.使用定时器,定时采集
timer->start(1000/FRAME_RATE - 10);
3.开启采集
cap.open(0);//打开默认摄像头
if(!cap.isOpened()){
	QMessageBox::information(NULL,tr("提示"),tr("视频没有打开"));
	return;
}
4.关闭采集
m_timer->stop();
if(cap.isOpened())
		cap.release();
5.采集流程
Mat frame;
if( !cap.read(frame) )
{
	return;
}
//将 opencv 采集的 BGR 的图片类型转化为 RGB24 的类型
cvtColor(frame,frame,CV_BGR2RGB);
//定义 QImage 对象，用于发送数据以及图片显示
QImage image ((unsigned const
char*)frame.data,frame.cols,frame.rows,QImage::Format_RGB888);
//转化为大小更小的图片
image = image.scaled( IMAGE_WIDTH,IMAGE_HEIGHT, Qt::KeepAspectRatio );
//发送图片
Q_EMIT SIG_sendVideoFrame( image );
```

图片压缩

```
//控制类发送视频帧函数
void CKernel::slot_sendVideoFrame(QImage img);
```

```
//压缩图片从 RGB24 格式压缩到 JPEG 格式，发送出去
QByteArray ba;
QBuffer qbuf(&ba); // QBuffer 与 QByteArray 字节数组联立联系
img.get()->save( &qbuf , "JPEG" , 50 );   //将图片的数据写入 ba
//使用 ba 对象，可以获取图片对应的缓冲区
可以使用 ba.data() , ba.size()将缓冲区发送出去
```

加载图片

```
QByteArray ba( buf, nlen);
QImage img;
img.loadFromData( ba );
//可以使用 img 来显示图片
```

显示图片的绘图事件函数

```
void VideoItem::paintEvent(QPaintEvent *event)
{
    //画黑背景
    QPainter painter(this);
    painter.setBrush( Qt::black );
    painter.drawRect( 0,0, this->width() , this->height());

    //画图片
    if( m_image.size().height()<= 0   ) return;

    // 加载图片用 QImage，画图使用 QPixmap
    // 图片缩放   scaled
    QPixmap   pixmap =   QPixmap::fromImage( m_image.scaled( QSize( this->width() , this->height() - ui->lb_name->height()), Qt::KeepAspectRatio   ));

    //画的位置
    int x = this->width() - pixmap.width();
    int y = this->height() - pixmap.height() - ui->lb_name->height();
    x = x /2;
    y = ui->lb_name->height() + y/2;

    painter.drawPixmap( QPoint(x,y) , pixmap );
    painter.end();
}
```