## 1.视频编码的介绍

视频压缩又称视频编码，所谓视频编码方式就是通过特定的压缩技术，将某个视频格式的文件转换成另一种视频格式的文件的方式。

一般通用的数据压缩方案如下图所示：

输入数据 输出数据

存储器或网络

解码器

编码器

目前几种视频编码的比较：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编码标准 | 特性 | | 优点 | 缺点 |
| JPEG  (15:1) | 将图像分为8x8的像素块，按照从左到右，从上到下的光栅扫描方式进行排序。DCT在8x8的像素块进行计算，再对64个DCT系数用均匀量化表进行标量量化 | JPEG的性能，用质量与比特率之比来衡量，是相当优越的 | | 压缩率低 |
| MPEG-1  (26:1) | 对动作不激烈的视频信号可获得较好的图像质量，但当动作激烈时，图像就会产生马赛克现象。它没有定义用于额外数据流进行编对码的格式，因此这种技术不能广泛推广。它主要用于家用VCD，它需要的存储空间比较大； | | 对动作不激烈的视频信号可获得较好的图像质量； | 当动作激烈时，图像就会产生马赛克现象； |
| MPEG-4  （50:1） | 支持对象型态编码及合成图像的压缩、适用于高阶交互功能与特殊视频制作、容错性编码技术及细微式可调性编码技术，可适用于频宽变化剧烈的网络，更适于交互AV服务以及远程监控。 | | 压缩率高，质量优，容错性好，视频质量分辨率比较高，而数据速率相对较低，采用面向对象的压缩方式。 | 专利收费不合理 |
| H.264  (102:1) | H.264在编码框架上还是沿用以往的MC-DCT结构，即运动补偿加变换编码的混合结构，因此它保留了一些先前标准的特点。 | | 压缩率高，压缩出来的视频质量高； | 较高的复杂度 |

## 2.H264编码原理

H264是新一代的编码标准，以高压缩高质量和支持多种网络的流媒体传输著称，在编码方面，我理解的他的理论依据是：参照一段时间内图像的统计结果表明，在相邻几幅图像画面中，一般有差别的像素只有10%以内的点,亮度差值变化不超过2%，而色度差值的变化只有1%以内。所以对于一段变化不大图像画面，我们可以先编码出一个完整的图像帧A，随后的B帧就不编码全部图像，只写入与A帧的差别，这样B帧的大小就只有完整帧的1/10或更小！B帧之后的C帧如果变化不大，我们可以继续以参考B的方式编码C帧，这样循环下去。这段图像我们称为一个序列（序列就是有相同特点的一段数据），当某个图像与之前的图像变化很大，无法参考前面的帧来生成，那我们就结束上一个序列，开始下一段序列，也就是对这个图像生成一个完整帧A1，随后的图像就参考A1生成，只写入与A1的差别内容。

在H264协议里定义了三种帧，完整编码的帧叫I帧，参考之前的I帧生成的只包含差异部分编码的帧叫P帧，还有一种参考前后的帧编码的帧叫B帧。

H264采用的核心算法是帧内压缩和帧间压缩，帧内压缩是生成I帧的算法，帧间压缩是生成B帧和P帧的算法。

**序列的说明：**

在H264中图像以序列为单位进行组织，一个序列是一段图像编码后的数据流，以I帧开始，到下一个I帧结束。

一个序列的第一个图像叫做 IDR 图像（立即刷新图像），IDR 图像都是 I 帧图像。H.264 引入 IDR 图像是为了解码的重同步，当解码器解码到 IDR 图像时，立即将参考帧队列清空，将已解码的数据全部输出或抛弃，重新查找参数集，开始一个新的序列。这样，如果前一个序列出现重大错误，在这里可以获得重新同步的机会。IDR图像之后的图像永远不会使用IDR之前的图像的数据来解码。

一个序列就是一段内容差异不太大的图像编码后生成的一串数据流。当运动变化比较少时，一个序列可以很长，因为运动变化少就代表图像画面的内容变动很小，所以就可以编一个I帧，然后一直P帧、B帧了。当运动变化多时，可能一个序列就比较短了，比如就包含一个I帧和3、4个P帧。

**三种帧的说明：**

1、I帧

I帧:帧内编码帧 ，I帧表示关键帧，你可以理解为这一帧画面的完整保留；解码时只需要本帧数据就可以完成（因为包含完整画面）

I帧特点:

1)它是一个全帧压缩编码帧。它将全帧图像信息进行JPEG压缩编码及传输;

2)解码时仅用I帧的数据就可重构完整图像;

3)I帧描述了图像背景和运动主体的详情;

4)I帧不需要参考其他画面而生成;

5)I帧是P帧和B帧的参考帧(其质量直接影响到同组中以后各帧的质量);

6)I帧是帧组GOP的基础帧(第一帧),在一组中只有一个I帧;

7)I帧不需要考虑运动矢量;

8)I帧所占数据的信息量比较大。

2、P帧

P帧:前向预测编码帧。P帧表示的是这一帧跟之前的一个关键帧（或P帧）的差别，解码时需要用之前缓存的画面叠加上本帧定义的差别，生成最终画面。（也就是差别帧，P帧没有完整画面数据，只有与前一帧的画面差别的数据）

P帧的预测与重构:P帧是以I帧为参考帧,在I帧中找出P帧“某点”的预测值和运动矢量,取预测差值和运动矢量一起传送。在接收端根据运动矢量从I帧中找出P帧“某点”的预测值并与差值相加以得到P帧“某点”样值,从而可得到完整的P帧。

P帧特点:

1)P帧是I帧后面相隔1~2帧的编码帧;

2)P帧采用运动补偿的方法传送它与前面的I或P帧的差值及运动矢量(预测误差);

3)解码时必须将I帧中的预测值与预测误差求和后才能重构完整的P帧图像;

4)P帧属于前向预测的帧间编码。它只参考前面最靠近它的I帧或P帧;

5)P帧可以是其后面P帧的参考帧,也可以是其前后的B帧的参考帧;

6)由于P帧是参考帧,它可能造成解码错误的扩散;

7)由于是差值传送,P帧的压缩比较高。

3、B帧

B帧:双向预测内插编码帧。B帧是双向差别帧，也就是B帧记录的是本帧与前后帧的差别（具体比较复杂，有4种情况，但我这样说简单些），换言之，要解码B帧，不仅要取得之前的缓存画面，还要解码之后的画面，通过前后画面的与本帧数据的叠加取得最终的画面。B帧压缩率高，但是解码时CPU会比较累。

B帧的预测与重构

B帧以前面的I或P帧和后面的P帧为参考帧,“找出”B帧“某点”的预测值和两个运动矢量,并取预测差值和运动矢量传送。接收端根据运动矢量在两个参考帧中“找出(算出)”预测值并与差值求和,得到B帧“某点”样值,从而可得到完整的B帧。

B帧特点

1）B帧是由前面的I或P帧和后面的P帧来进行预测的;

2）B帧传送的是它与前面的I或P帧和后面的P帧之间的预测误差及运动矢量;

3）B帧是双向预测编码帧;

4）B帧压缩比最高,因为它只反映丙参考帧间运动主体的变化情况,预测比较准确;

5）B帧不是参考帧,不会造成解码错误的扩散。

注:I、B、P各帧是根据压缩算法的需要，是人为定义的,它们都是实实在在的物理帧。一般来说，I帧的压缩率是7（跟JPG差不多），P帧是20，B帧可以达到50。可见使用B帧能节省大量空间，节省出来的空间可以用来保存多一些I帧，这样在相同码率下，可以提供更好的画质。

**压缩算法的说明：**

h264的压缩方法:

1.分组:把几帧图像分为一组(GOP，也就是一个序列),为防止运动变化,帧数不宜取多。

2.定义帧:将每组内各帧图像定义为三种类型,即I帧、B帧和P帧;

3.预测帧:以I帧做为基础帧,以I帧预测P帧,再由I帧和P帧预测B帧;

4.数据传输:最后将I帧数据与预测的差值信息进行存储和传输。

帧内（Intraframe）压缩也称为空间压缩（Spatial compression）。当压缩一帧图像时，仅考虑本帧的数据而不考虑相邻帧之间的冗余信息，这实际上与静态图像压缩类似。帧内一般采用有损压缩算法，由于帧内压缩是编码一个完整的图像，所以可以独立的解码、显示。帧内压缩一般达不到很高的压缩，跟编码jpeg差不多。

帧间（Interframe）压缩的原理是：相邻几帧的数据有很大的相关性，或者说前后两帧信息变化很小的特点。也即连续的视频其相邻帧之间具有冗余信息,根据这一特性，压缩相邻帧之间的冗余量就可以进一步提高压缩量，减小压缩比。帧间压缩也称为时间压缩（Temporal compression），它通过比较时间轴上不同帧之间的数据进行压缩。帧间压缩一般是无损的。帧差值（Frame differencing）算法是一种典型的时间压缩法，它通过比较本帧与相邻帧之间的差异，仅记录本帧与其相邻帧的差值，这样可以大大减少数据量。

顺便说下有损（Lossy ）压缩和无损（Lossy less）压缩。无损压缩也即压缩前和解压缩后的数据完全一致。多数的无损压缩都采用RLE行程编码算法。有损压缩意味着解压缩后的数据与压缩前的数据不一致。在压缩的过程中要丢失一些人眼和人耳所不敏感的图像或音频信息,而且丢失的信息不可恢复。几乎所有高压缩的算法都采用有损压缩,这样才能达到低数据率的目标。丢失的数据率与压缩比有关,压缩比越小，丢失的数据越多,解压缩后的效果一般越差。此外,某些有损压缩算法采用多次重复压缩的方式,这样还会引起额外的数据丢失。

## 3.利用FFmpge进行视频压缩(从图像到H264视频流)

首先设计一个视频压缩相关的类，定义如下：

1. class Ffmpeg\_Encoder
2. {
3. public:
5. AVFrame \*m\_pRGBFrame;   //帧对象
6. AVFrame \*m\_pYUVFrame;   //帧对象
7. AVCodec \*pCodecH264;    //编码器
8. AVCodecContext \*c;      //编码器数据结构对象
9. uint8\_t \*yuv\_buff;      //yuv图像数据区
10. uint8\_t \*rgb\_buff;      //rgb图像数据区
11. SwsContext \*scxt;       //图像格式转换对象
12. uint8\_t \*outbuf;        //编码出来视频数据缓存
13. int outbuf\_size;        //编码输出数据去大小
14. int nDataLen;           //rgb图像数据区长度
15. int width;              //输出视频宽度
16. int height;             //输出视频高度
17. avpacket pkt;            //数据包结构体
18. public:
19. void Ffmpeg\_Encoder\_Init();//初始化
20. void Ffmpeg\_Encoder\_Setpara(CodecID mycodeid,int vwidth,int vheight);//设置参数,第一个参数为编码器,第二个参数为压缩出来的视频的宽度，第三个视频则为其高度
21. void Ffmpeg\_Encoder\_Encode(FILE \*file, uint8\_t \*data);//编码并写入数据到文件
22. void Ffmpeg\_Encoder\_Close();//关闭
23. };

对类中声明的四个函数进行定义：

初始化函数：

1. **void Ffmpeg\_Encoder::Ffmpeg\_Encoder\_Init()**
2. **{**
3. **av\_register\_all();**
4. **avcodec\_register\_all();**
6. **m\_pRGBFrame = new AVFrame[1];//RGB**帧数据赋值
7. m\_pYUVFrame = new AVFrame[1];//YUV帧数据赋值
9. c = NULL;//解码器指针对象赋初值
10. **}**

设置参数：

1. void Ffmpeg\_Encoder::Ffmpeg\_Encoder\_Setpara(CodecID mycodeid, int vwidth, int vheight)
2. {
3. pCodecH264 = avcodec\_find\_encoder(mycodeid);//查找h264编码器
4. if (!pCodecH264)
5. {
6. fprintf(stderr, "h264 codec not found\n");
7. exit(1);
8. }
9. width = vwidth;
10. height = vheight;
12. c = avcodec\_alloc\_context3(pCodecH264);//函数用于分配一个AVCodecContext并设置默认值，如果失败返回NULL，并可用av\_free()进行释放
13. c->bit\_rate = 400000; //设置采样参数，即比特率
14. c->width = vwidth;//设置编码视频宽度
15. c->height = vheight; //设置编码视频高度
16. c->time\_base.den = 2;//设置帧率,num为分子和den为分母，如果是1/25则表示25帧/s
17. c->time\_base.num = 1;
18. c->gop\_size = 10; //设置GOP大小,该值表示每10帧会插入一个I帧
19. c->max\_b\_frames = 1;//设置B帧最大数,该值表示在两个非B帧之间，所允许插入的B帧的最大帧数
20. c->pix\_fmt = PIX\_FMT\_YUV420P;//设置像素格式
22. av\_opt\_set(c->priv\_data, "tune", "zerolatency", 0);//设置编码器的延时，解决前面的几十帧不出数据的情况
24. if (avcodec\_open2(c, pCodecH264, NULL) < 0)return;//打开编码器
26. nDataLen = vwidth\*vheight \* 3;//计算图像rgb数据区长度
28. yuv\_buff = new uint8\_t[nDataLen/2];//初始化数据区，为yuv图像帧准备填充缓存
29. rgb\_buff = new uint8\_t[nDataLen];//初始化数据区，为rgb图像帧准备填充缓存
30. outbuf\_size = 100000;////初始化编码输出数据区
31. outbuf = new uint8\_t[outbuf\_size];
33. scxt = sws\_getContext(c->width, c->height, PIX\_FMT\_BGR24, c->width, c->height, PIX\_FMT\_YUV420P, SWS\_POINT, NULL, NULL, NULL);//初始化格式转换函数
34. }

编码并写入数据文件：

1. void Ffmpeg\_Encoder::Ffmpeg\_Encoder\_Encode(FILE \*file, uint8\_t \*data)
2. {
3. memcpy(rgb\_buff, data, nDataLen);//拷贝图像数据到rgb图像帧缓存中准备处理
4. avpicture\_fill((AVPicture\*)m\_pRGBFrame, (uint8\_t\*)rgb\_buff, PIX\_FMT\_RGB24, width, height);//将rgb\_buff填充到m\_pRGBFrame
5. avpicture\_fill((AVPicture\*)m\_pYUVFrame, (uint8\_t\*)yuv\_buff, PIX\_FMT\_YUV420P, width, height);//将yuv\_buff填充到m\_pYUVFrame
6. sws\_scale(scxt, m\_pRGBFrame->data, m\_pRGBFrame->linesize, 0, c->height, m\_pYUVFrame->data, m\_pYUVFrame->linesize);// 将RGB转化为YUV
7. int myoutputlen = avcodec\_encode\_video(c, outbuf, outbuf\_size, m\_pYUVFrame);
8. fwrite(outbuf, 1, myoutputlen, file);
9. }
10. void Ffmpeg\_Encoder::Ffmpeg\_Encoder\_Close()
11. {
12. delete[]m\_pRGBFrame;
13. delete[]m\_pYUVFrame;
14. delete[]rgb\_buff;
15. delete[]yuv\_buff;
16. delete[]outbuf;
17. sws\_freeContext(scxt);
18. avcodec\_close(c);//关闭编码器
19. av\_free(c);
20. }

最后在主函数对这几个函数进行调用就可以了，直接用OpencCV来打开图像并取得图像的数据区。

1. void main()
2. {
3. Ffmpeg\_Encoder ffmpegobj;
4. ffmpegobj.Ffmpeg\_Encoder\_Init();//初始化编码器
5. ffmpegobj.Ffmpeg\_Encoder\_Setpara(CODEC\_ID\_H264,800,600);//设置编码器参数
7. //图象编码
8. FILE \*f = NULL;
9. char \* filename = "myData.h264";
10. fopen\_s(&f, filename, "wb");//打开文件存储编码完成数据
12. IplImage\* img = NULL;//OpenCV图像数据结构指针
13. IplImage\* resizeimg = NULL;//尺寸
14. int picturecount = 1;
15. while (picturecount != 9)
16. {
17. /\*\*此部分用的是OpenCV读入图像对象并取得图像的数据区，也可以用别的方法获得图像数据区\*\*/
18. char chpicname[100];
19. sprintf(chpicname, "testpicture\\Horse%d.jpg", picturecount);//获得图片路径
20. //sprintf(chpicname, "1.jpg", picturecount);//获得图片路径
21. img = cvLoadImage(chpicname, 1);//打开图像
22. //由于OpenCV图像数据区是以BGR排列的，所以要将其数据转换为正常的RGB排列才能做进一步的压缩，不然压出来的视频颜色会不正确
23. uchar\* data = (uchar\*)(img->imageData);
24. uchar mid = 0;
25. for (int row = 0; row<img->height; row++)
26. for (int cols = 0; cols < img->width; cols++)
27. {
28. mid = data[row\*img->widthStep / sizeof(uchar)+cols\*img->nChannels + 0];//G
29. data[row\*img->widthStep / sizeof(uchar)+cols\*img->nChannels + 0] = data[row\*img->widthStep / sizeof(uchar)+cols\*img->nChannels + 2];
30. data[row\*img->widthStep / sizeof(uchar)+cols\*img->nChannels + 2] = mid;
31. }
32. resizeimg = cvCreateImage(cvSize(800, 600), 8, 3);
33. cvResize(img, resizeimg, CV\_INTER\_LINEAR);//调整图像大小
34. /\*\*此部分用的是OpenCV读入图像对象并取得图像的数据区，也可以用别的方法获得图像数据区\*\*/
36. ffmpegobj.Ffmpeg\_Encoder\_Encode(f, (uchar\*)resizeimg->imageData);//编码
38. cvReleaseImage(&img);//释放图像数据结构指针对像所指内容
39. cvReleaseImage(&resizeimg);
40. picturecount++;
41. }
42. fclose(f);
43. ffmpegobj.Ffmpeg\_Encoder\_Close();
44. }