一、HM使用方法

在HM-16.20目录下打开build文件夹（工程存放在build目录下）， 打开VS2013(.sln)，可以看到HM包含的工程。HM解决方案包含了9个工程：

(1) TAppCommon 编码器和解码器共用的应用函数

(2) TAppDecoder 解码器的应用函数

(3) TAppEncoder 编码器的应用函数

(4) TLibCommon 编码器和解码器共用的库函数

(5) TLibDecoder 解码器的库函数

(6) TLibEncoder 编码器的库函数

(7) TLibVideoIO 视频输入、输出库函数

(8)TAppDecoderAnalyser

(9)TLibDecoderAnalyser

其中，'T'代表'Test'（这一个的理解可能有误），'App'代表'Application'，表明该工程主要包含一些应用函数，'Lib'代表'Library'，表明该工程主要包含一些库函数，这里顺便提一下，应用函数与库函数的主要区别是：前者是面向用户的，主要是通过调用若干库函数实现更为丰富和复杂的功能，而后者是面向程序设计者的，或者说对用户是不可见的，它由程序设计者来实现，主要是对一些基本的功能进行底层设计与实现，对于用户来说，只关心这些库函数的接口以及如何调用，不需也不应该关心它的实现。'Common'表明该工程包含的一些函数是编码器和解码器共用的，'Decoder'表明该工程包含的函数是解码器使用的，而'Encoder'表明该工程包含的函数是编码器使用的。'VideoIO'工程主要是实现对YUV文件的读写操作。

在属性中配置设为debug x64，右击“Solution 'HM\_VC2013'(9 projects)”->“build solution”。正常的话，等待片刻所有工程都能得到正确的编译，并最终在HM文件夹下生成bin目录，在目录xxx\bin\vc2013\x64\Debug下，最终会生成两个可执行文件，TAppEncoder.exe和TAppDecoder.exe，分别为编码和解码的可执行文件。HEVC标准规定的是解码端，编码端可以随意改动，因此一般的工作都是集中于编码器方面。

# 编码器的使用

右击“TAppEncoder”->“Set as Startup Project”, 这个项目就会加黑，接着，再次右击“TAppEncoder”->“Properties”->“Configuration Properties”->“Debugging”，在弹出的右窗口中有这么两行需要关注：“Command Arguments“和“Working Directory”。前者用于输入运行时的命令参数，后者用于指定工程的工作目录。先说工作目录，将其设置为xxx\bin\vc2013\x64\Debug，即上一步中编译生成的可执行文件的绝对路径。与JM有所不同的是，我们需要至少使用两个配置文件，即在HM文件夹的cfg（存储了公共测试的配置文件，分为4种）目录中的某一个配置文件，以及per-sequence目录下的某一个配置文件。工程目录中的cfg文件夹中有多个预先设定好的配置文件，分别代表了不同的编码模式。该目录中的pre-sequence文件夹中指定了多个输入测试序列的配置信息。前者主要用于配置编码器的各种编码参数，后者主要用于指定输入yuv测试文件，分辨率，待编码帧数等等。何谓配置文件呢，一般编码一个视频需要两个配置文件，第一个就是你如果要编码一个50帧的视频，一个配置文件会对这个50帧的视频本身做说明，例如它的文件名，它编码的码率，输出的二进制文件名和重建的视频。另一个是你编码中CU大小的设置等等具体到编码的东西。接下来，为了方便起见，我们可以把需要使用到的配置文件、yuv测试序列复制到该目录下（由于yuv测试序列可能很大，我们可以将其地址复制到配置文件中，下面对相关配置文件进行更改，用记事本打开per-sequence文件，可看到以下参数

InputFile 将测试序列路径（可以是绝对路径，也可以是相对路径）写出来

SourceWidth 分辨率（宽度）

SourceHeight 分辨率（高度）

FramesToBeEncoded 要编码的帧数，为了节省时间我们把FramesToBeEncoded的500改为10，即只编码10帧。）

在完成了上述步骤之后，我们就可以在“Command Arguments”一栏填入

-c encoder\_intra\_main.cfg -c BasketballDrill.cfg（-c是指定使用该配置文件的参数进行编码），表明这个编码器使用这两个配置文件所指定的参数进行编码。

# 解码器的使用

在到填入“Command Arguments”之前，跟编码器是完全一样的，这里就不再重复了，输入的命令如下-b xxx.bin -o deco.yuv，-b指定输入的二进制码流，即编码输出码流文件，-o指定输出的yuv文件。设置完毕之后，运行解码器，应该能够解码出deco.yuv文件来了。

二、HM代码中关于变量、函数的命名规则

（1）类的命名：

一般来说，一个头文件只包含一个类的定义，文件名即为类名，且该类是属于哪个工程的，它的名字就以该工程的前几个字母开头，如类TAppEncTop，它就是以工程TAppEncoder的前7个字母开头，因此，从该类的名字，就能够看出该类是属于哪个工程的。

（2）变量的命名：

对于类的数据成员来说，一般以'm\_'开头，即'member'；对于全局变量来说，一般以'g\_'开头，即'global'。

对于一般的变量（包含上述两种变量）来说，有如下命名规则：'p'，该变量是指针类型，即'pointer'，n个'p'则表明该指针为n级指针；'c'，该变量是某个类的对象，即'class'；'i'，该变量是整型，即'int'；'u'，该变量是无符号型，即'unsigned'；'h'，该变量是字符型，这里不用'c'来代表'char’应该是为了避免跟前面的'class'冲突了；'b'，该变量是布尔类型，即'bool’；'d'，该变量是双精度浮点数，即'double'；'f'，该变量是单精度浮点数，即'float'；'a'，该变量是类组，即'array'；'e'，该变量是枚举类型，即'enum'；'r'，该变量是引用类型，等等。值得一提的是，不是每个变量的命名都满足这些规则，具体情况还是要具体分析的。但是，按照这些规则，80%以上的变量都能一眼看出它的特性来。

（3）函数的命名：

一般来说，对于一个类的成员函数来说，如果该函数的访问权限是'protected'，则在其函数名前加上'x'；但是，在我看代码过程中，有些'protected'的成员函数并没按照这个规则来命名，所以，这一条规则仅供参考。能够肯定的是，只要某个函数名字前有个'x'，则该函数一定是某个类的protected成员函数。

说了这么多，举个具体实例来说明下吧：

在工程TLibEncoder的头文件TEncCu.h中，定义了一个类TEncCu，有个数据成员m\_ppcBestCU，根据前面的命名规则，它首先是个数据成员，是个二级指针，且是指向一个类的，实际上，它是这么声明的，TComDataCU\*\*，TComDataCU它就是一个类，且该变量被声明为二级指针，据此，符合上述命名规则。

还有一个数据成员这么声明UChar m\_uhTotalDepth; 无符号字符型，同样也是符合命名规则的。

在该类中，有这么一个成员函数xCompressCU，以'x'开头命名，在类中被声明为'protected'，符合命名规则。

三、HEVC参考软件代码总结

1.编码器程序从"TAppEncoder"工程中的encmain.cpp文件开始的，此文件中包含程序运行的入口函数"main"，在main函数中主要做了编码器对象的创建、分析配置文件，初始化配置参数，和编码器最重要的功能"encode"。

2.在"encode"函数中，主要实现了读取YUV文件的数据、初始化工具对象，例如：GOPEncoder、SliceEncoder、CUEncder……。在此函数里，还包括一个encode函数，调用CompressGOP函数来具体执行编码任务。

3.在CompressGOP函数中，完成了以下的功能：

一，InitGOP将文件的码流分成若干GOP以便后续程序能够顺利执行。

二，InitEncSlice创建编码的Slice。

三，在此函数中，还包括preCompressSlice和CompressSlice两个函数，前者的作用是选择不同的lamuda进行编码（编码是调用了CompressCU函数，后续介绍），后者是在最好的lamuda下进行编码。

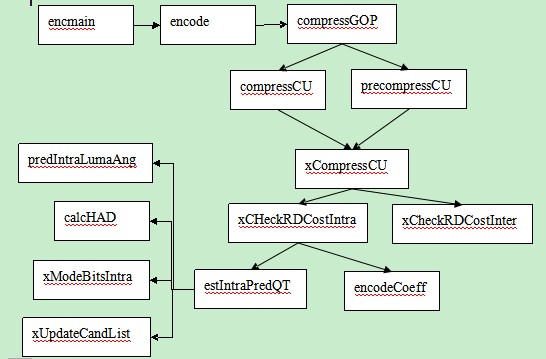
四，循环滤波

五，……（熵编码等，还没看）。

4.在xCompressCU函数中（CompressCU函数的主体也是调用xComprssCU函数），先进行帧间预测xCheckRDCostMerge2Nx2N，xCheckRDCostInter等。在做完帧间预测后进行阵内预测，这是调用的函数是xCheckRDCostIntra，在xCompressCU函数的后续部分，还递归调用自身以实现对每个CU的编码。变换编码在encodeCoeff中实现，量化在xCheckIntraPCM完成。

5.xCheckRDCostIntra函数，主要完成帧内预测的任务，对亮度的预测使用estIntraPredQT，对色度使用estIntraPredChromaQT。

6.estIntraPredQT函数，在思想对亮度的处理和色度的处理是一样的，所以只介绍亮度的处理函数。在estIntraPredQT函数中，主要完成了RDCost的选择，在其中predIntraLumaAng函数实现了方向的预测；calcHAD函数计算了SATD；xModeBitsIntra函数计算编码的码率；xUpdateCandList更新了最好的RDCost所使用的模式。



编码的简要流程

TAppEncTop的encode函数调用了compressGOP这个函数，这是一个非常重要的函数，它对一个图像组进行编码。

compressGOP函数（这个函数将近2000行，我就不贴上来了。。。）：

1、调用xInitGOP设置图像组中帧的数量

2、设置一些图像增强信息

3、进入一个大的for循环，对图像组的每一帧图像进行处理

3.1、说明一下GOPEntry并不表示一个图像组，而是表示某一帧对应的GOP（即图像组）的信息，它包含了这个帧的参考帧等重要的信息。

3.2、调用selectReferencePictureSet、createExplicitReferencePictureSetFromReference、applyReferencePictureSet、arrangeLongtermPicturesInRPS、setRefPicList、setRefPOCList、setList1IdxToList0Idx等几个函数，设置当前帧的参考图像集、参考帧、并对这些参考帧进行排序等等，比较复杂的功能。

3.3、进入一个while循环，对帧的每一片（slice）进行编码，主要是调用compressSlice来选出最优的参数，以及预测、变换、量化的编码工作，然后进行环路滤波，最后调用encodeSlice对其进行熵编码

4、一些收尾工作！

**摘自CSDN----NB\_vol\_1--- HM编码器代码阅读(7)——整个编码流程以及相关的函数**

整个流程可以从compressGOP函数开始着手：

1、compressGOP对一整个图像组（GOP）进行编码，主要是遍历GOP中每一帧，对每一帧进行单独编码

2、每一帧又会被划分成若干slice（HM15中，每一帧对应一个slice），因此对每一帧的处理就转换成对每一片的处理。

3、每一个slice都会调用compressSlice来对其进行预测、变换、量化等操作，然后选出最优的参数，最后调用encodeSlice对其进行熵编码！

4、slice的划分。编码slice的时候并不是对整个slice进行处理，而是会继续把slice划分成LCU（最大的CU），然后对每一个CU递归调用compressCU和encodeCU进行编码。其中encodeCU会被compressSlice和encodeSlice调用，在compressSlice中调用encodeCU的目的是为了选择最优的参数，而在encodeSlice中调用encodeCU的目的是对CU进行熵编码。

5、LCU的大小是64x64，一个slice可以划分为若干LCU，而每一个CU（或者LCU）又可以递归地被划分成4个子CU。如何判断某一个CU是否会继续向下划分，可以根据RD（拉格朗日率失真）来判断！

6、compressCU的实质就是对每一个CU递归的调用xCompressCU。在xCompressCU的内部，会根据帧的类型（I帧、P帧、B帧）进行不同的处理。对于I帧，进行的是帧内预测；对于P帧和B帧则进行帧间预测。

7、帧内预测的详解（入口函数：xCheckRDCostIntra）：

7.1、对于亮度分量：

7.1.1、调用estIntraPredQT。主要做模式选择的工作，负责选出对于当前PU的最优模式，如DC、planar、角度等模式。

（1）首先对N个候选模式进行粗粒度筛选。代价函数是SATD+λ\*ModeBits。选出若干个可能的候选模式。下面是相关的函数。

（2）predIntraLumaAng。计算当前PU的预测值。

（3）calcHAD。计算SATD代价。

（4）xModeBitsIntra。计算当前模式所消耗的比特数。

（5）xUpdateCandList。更新模式的代价，保持前N个模式的代价最小。

（6）选出N个模式之后，这N个模式会进入xRecurIntraCodingQT函数继续处理。

7.1.2、调用xRecurIntraCodingQT。根据选出的模式进行PU的分割，然后进行变换量化等工作。

（1）这个函数会被调用两次，第一次调用直接把PU当作TU，只为算出N个模式的RD代价，从而选出一个最优的，在这个最优的模式被选出后，再次调用这个函数，对这个最优的模式进行PU的分割。下面的是相关的函数。

（2）xIntraCodingLumaBlk。对当前TU进行求残差，对残差进行变换、量化、反量化、反变换、重建等一系列工作，并求出失真。

（3）xGetIntraBitsQT。求出当前模式的所有信息进行熵编码会产生的比特数。

（4）calcRdCos。根据 xIntraCodingLumaBlk得到的失真和 xGetIntraBitsQT产生的比特数进行RD代价的计算，从而比较各个模式的优劣。

（5）xSetIntraResultQT。保存最优模式的数据。

7.2、对于色度分量，过程和亮度大部分一样。

8、帧间预测的详解：

8.1、帧间分为两种方式，一种是默认的inter模式（入口函数xCheckRDCostInter），另一种是merge模式（入口函数xCheckRDCostMerge2Nx2N）。

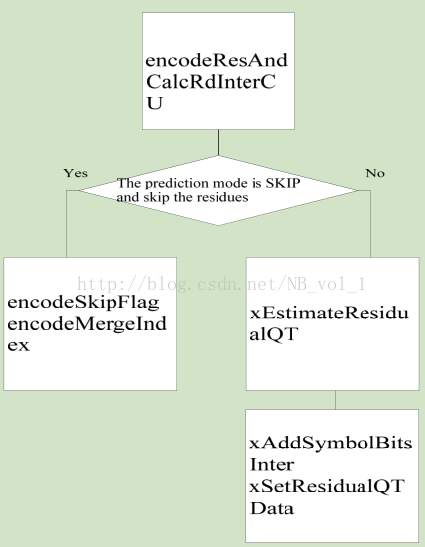
8.2、inter模式的调用流程：xCheckRDCostInter——>predInterSearch——>encodeResAndCalcRdInterCU

8.3、merge模式的调用流程：xCheckRDCostMerge2Nx2N——>motionCompensation——>encodeResAndCalcRdInterCU

8.4、predInterSearch进行的是ME（运动估计）和MC（运动补偿）的过程。

8.5、motionCompensation进行的是MC的工作。因为merge进行的是MV预测，因此没有ME（运动估计）的过程。

8.6、encodeResAndCalcRdInterCU。根据预测值，求出残差，然后进行TU的划分，然后进行变换、量化等操作以及RD代价的计算。流程图如下：



8.6.1、encodeSkipFlag。对skip模式的标志进行编码。

8.6.2、encodeMergeIndex。对MVP的索引进行编码。

8.6.3、xEstimateResidualQT。在非skip模式的时候，进行RQT的决定（即应该把PU分割成什么样的TU）。

8.6.4、xAddSymbolBitsInter。计算当前模式在进行熵编码时产生的比特数。

8.6.5、xSetResidualQTData。保存当前最优的残差信息。

9、其他函数相关函数：

9.1、帧内

initPattern ：判断周围块的存在性

initAdiPattern：获取周围像素的值当做生成预测值的像素，并开辟出一片缓存 区存储经过多种滤波类型的预测值

getPredictorPtr: 根据不同模式选择经过不同类型滤波的预测集

predIntraLumaAng： 对亮度信号进行预测，里面会调用 xPredIntraPlanar，xPredIntraAng 以及 xDCPredFiltering

predIntraChromaAng ： 对色度信号进行预测，里面会调用 xPredIntraPlanar 和xPredIntraAng

xPredIntraPlanar： planar 模式的预测

xPredIntraAng： 角度的方向性预测

xDCPredFiltering： 对 DC 的预测值进行滤波

getLumaRecPixels： 获取亮度的重建值，为进行 LM 模式的预测做准备

predLMIntraChroma：对 LM 模式进行预测，即利用亮度的相关性，对色度进行预测

9.2、帧间

getInterMergeCandidates： 获取 merge 的候选运动参数集

motionCompensation：进行运动补偿

xMotionEstimation：进行运动估计

xEstimateMvPredAMVP：选出代价最小的 MVP

xCheckBestMVP：在知道 MV 的情况下比较各个 MVP 的优劣，并保存最优的

xMergeEstimation：在 inter 模式时也可以使用 merge 模式的运动估计方法，这个函数用于计算这种情况时的代价

9.3、变换

transformNxN：会调用 xT 和 xQuant 函数

invtransformNxN：会调用 xDeQuant 和 xIT 函数

xT： 对残差信号进行变换

xQuant：对变换系数进行量化

xDeQuant ：反量化

xIT：反变换

9.4、熵编码（在这节中主要介绍编码端为算 RD 代价而设计的熵编码函数，实际的熵编码函数在后面进行介绍）

9.4.1、帧内熵编码

xEncIntraHeader： 编码 intra 的一些头部信息， 主要包括： 模式号， PU 的分割类型，PCM 标志，如果是 B 或 P slice，还包括 skip 的标志位和编码模式的类型

xEncSubdivCbfQT：会编码 Cbf 和 TU 分割的标志位

xEncCoeffQT：编码每个 TU 的系数

encodeCoeffNxN：调用 codeCoeffNxN 来编码每个 TU 的残差系数

encodeTransformSubdivFlag： 调用codeTransformSubdivFlag来编码TU分割的标志，是否继续分割

encodeQtCbf：编码 cbf 标志位，检查是否有非零的系数

encodePredMode：编码所采用的编码模式

encodePartSize：编码 PU 的分割类型

encodeIntraDirModeLuma：编码 PU 的亮度模式号

encodeIntraDirModeChroma：编码 PU 的色度模式号

9.4.2、帧间熵编码

encodePredMode：编码所采用的编码模式

encodePredMode：编码所采用的编码模式

encodePartSize：编码 PU 的分割类型

encodePredInfo: 编码运动参数

（1） merge 的标志位来区别是否采用 merge 模式，具体函数：encodeMergeFlag然后分（2）和（3）两种情况。

（2） merge 模式：只需传输运动候选集的索引，具体函数：encodeMergeIndex

（3） 正常的 inter 模式

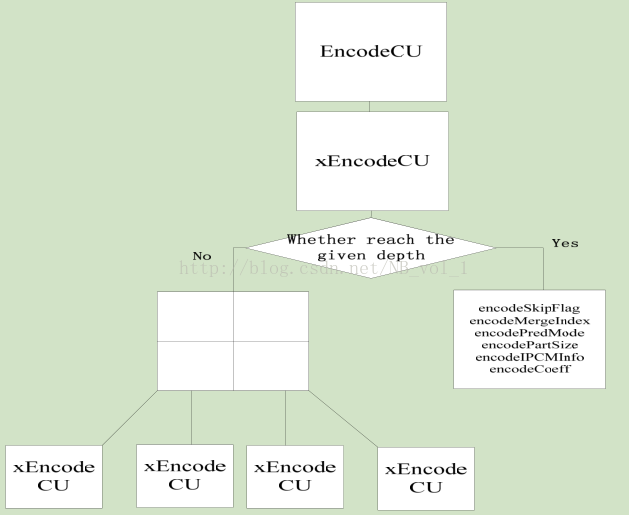
A. encodeInterDirPU：编码帧间的预测方向，前向，后向，或多方向

B. encodeRefFrmIdxPU： 编码参考帧索引

C. encodeMvdPU：编码 MV 的残差 MVD

D. encodeMVPIdxPU： 编码 MVP 的索引

10、熵编码（encodeCU）。encodeCU以LCU为单位进行熵编码。它的内部实际是针对每一个CU递归调用xEncodeCU。流程如下：



10.1、encodeSkipFlag 编码是否是 skip 模式

10.2、encodeMergeIndex 如果是 skip 模式会编码选用哪套 MVP 的参数

10.3、encodePredMode 编码 CU 的模式，是 intra 还是 inter

10.4、encodePartSize 编码 CU 中的 PU 的类型

10.5、encodeIPCMInfo 如果选用了 PCM 模式会编码 PCM 模式的信息

10.6、encodePredInfo 编码预测的信息，如果是帧内，编码模式号，如果是帧间，则编码运动信息

10.7、encodeCoeff 编码残差系数

10.8、encodeCoeff 中会编码 TU 的分割标志位，cbf 和残差系数的信息

**一些容易混淆的类和结构：**

TcomPic 是图片类，它包含TComPicSym（图像符号类）和TComPicYuv（yuv数据类：包括原始数据，预测数据、残差数据）

TvideoIOYuv 是用来读取yuv文件的类，读取出来之后把数据放到TComPicYuv中

TComPicSym中则存放了指向片（TComSlice）的二级指针，和指向CU（TComDataCU）的二维指针，其实片和CU的实际数据仍然存放在TComPicYuv中

TEncEntropyIf是熵编码算法的纯虚类

TEncSbac（TEncEntropyIf的子类）是SBAC熵编码算法的实现，SBAC是CABAC算法的一个改良

TencCavlc（TEncEntropyIf的子类） 是CAVLC熵编码算法的实现

TEncEntropy是熵编码器类，它和TEncEntropyIf的区别是，TEncEntropy是一个算法的管理和控制类，而TEncEntropyIf则是具体算法的实现类

TEncBinIf是二进制编码的纯虚类（二进制编码和熵编码是有区别的，熵编码需要用到二进制编码，还有一些其他没有熵编码的元素也可能用到二进制编码类）

TencBinCABAC（TEncBinIf的实现） 二进制CABAC类

TcomBitIf是比特流虚类

TComOutputBitstream（TcomBitIf的子类）输出的比特流类

TComBitCounter（TcomBitIf的子类）比特流计数器类

TEncGOP是图像组处理类，也就是图像组编码器类（它的m\_pcListPic存放图像组数据）

GOPEntry是图像组对象类，它包含了图像组必要的一些信息，TEncGOP处理的对象就是他

TCom和TEnc的区别在于，TEnc一般是管理器或者编码器，而TCom则是数据存储类或者算法实现类，例如TEncCu是CU编码器类，而TComDataCU则是CU的数据类（或者说CU数据的操作类，因为实际的数据仍然存放在TComPicYuv中）

TComPicYuv和TComYuv的区别，TComYuv可以看作是一个方便操作YUV的类，主要用于预测阶段，从 TComPicYuv产生而来，TComYuv的作用没有TComPicYuv那么大

**一些其他数据类型的定义：**

（1）Pxl 是8比特的像素类型

（2）Pel 是16比特的像素类型

（3）TCoeff 是变换系数的数据类型（int）

（4）TComPicSym图像符号

（5）枚举类型SliceConstraint，定义了和片相关的一些常量

（6）枚举类型SAOComponentIdx，定义了SAO组件的索引

（7）枚举类型SAOMode，定义了SAO的模式

（8）枚举类型SAOModeMergeTypes，定义了SAO模式合并的类型

（9）枚举类型SAOModeNewTypes定义了SAO模式的新类型

（10）枚举类型SAOEOClasses定义了SAO的EO类型

（11）结构体SAOOffset定义了SAO的偏移

（12）结构体SAOBlkParam定义了SAO块的参数

（13）枚举DFunc定义了失真的计算函数

（14）枚举RefPicList定义了参考列表

（15）枚举TextType定义图像纹理的类型（即亮度，色度等）

（16）枚举MVP\_DIR定义了运动矢量预测的方向

（17）枚举CI\_IDX定义了SBAC率失真方面的优化

（18）枚举COEFF\_SCAN\_TYPE定义了系数的扫描方式

以TCom开头的数据结构是存放数据和信息的类；以TEnc开头的数据结构则是编码功能实现的类；例如TComPic是存放图像数据信息的类，而TEncPic则是实现帧编码功能的类；以TApp开头的的是应用程序的入口。

以TApp开头的类：

（1）TAppEncCfg 编码器配置信息类，定义了一大堆的配置信息

（2）TAppEncTop HEVC编码器应用对象类，继承自TAppEncCfg，实现了整个编码框架的入口，读取文件，初始化，编码、写数据以及结束处理等操作。

以TEnc开头的类：

（1）TEncCfg 编码过程中的配置信息，TAppEncCfg的配置信息直接从文件中读取，而TEncCfg得信心则是通过计算或者判定得到

（2）TEncTop 编码主类，实现编码的主要入口。

TApp和TEnc的关系：TAppEncTop调用TEncTop，TEncTop再调用其他的各个部分编码功能，而TAppEncTop继承自TAppEncCfg，TEncTop继承自TEncCfg

数据和编码的类：

（1）TVideoIOYuv 用于打开、关闭、读写YUV文件。

（2）TComYuv 实现了内存YUV数据，指向YUV三个内存分量的指针，以及它们的宽高信息

（3）TComPicYuv 对TComYuv来说是更加仔细的实现（包括了边缘填充）

（3）GOPEntry 存放了图像组的信息，例如poc，参考图像列表等等

（4）TEncGOP 实现了对图像组的编码功能

（5）TComPic 图像数据类，包含TComPicYuv和TComPicSym的对象

（6）TComPicSym是图像符号类，里面存放了更加详细的图像信息

（7）TComSlice 存放了条带（片）的数据信息

（8）TEncSlice 实现片的编码功能

（9）TComDataCU 存放CU（编码单元）的数据信息

（10）TEncCu 实现了CU的编码功能

（11）TEncSearch 实现了运动搜索的功能

（12）TComPattern 提供了相邻像素的访问方法

（13）TComMv 定义了MV（运动向量）

（14）TComPrediction 实现了预测的功能

（15）TComRdCost 实现了率失真的计算功能

（16）TComRom.h 文件里定义全局的函数

（17）TComInterpolationFilter 实现了内插值滤波功能

（18）TComLoopFilter 实现了环路滤波

（19）TComSampleAdaptiveOffset 实现了SAO方面的计算

（20）TComTrQuant 实现了变换和量化的功能

（21）TEncAnalyze 一个统计信息分析类

（22）TEncPreanalyzer 源文件分析类，主要用来计算图像的特征

（23）TEncRateCtrl 码率控制类

（24）TEncSampleAdaptiveOffset SAO编码实现类

熵编码的类：

（1）TComBitIf 是比特编码的虚基类，是比特编码的基础接口（If就是Interface）。比特编码包括输出比特流，比特计数器、SBAC编码、CABAC编码等等

（2）TComOutputBitstream 是比特流输出类，主要用于比特流的输出，写到文件或者buffer中

（3）TComInputBitstream 比特流输入类（它不是TComBitIf的子类）

（4）TComBitCounter 比特计数器，用于统计比特的数量

（5）TEncBinIf 二进制编码接口类

（6）TEncBinCABAC CABAC编码实现类

（7）TEncBinCABACCounter CABAC比特计数实现类

（8）TEncEntropyIf 熵编码器的接口类

（9）TEncCavlc CAVLC熵编码实现类

（10）TEncSbac SABAC熵编码器实现类

由此可见熵编码部分主要分为三个部分：二进制化编码器，熵编码器，以及比特流统计以及写出类

**文件的功能以及作用：**

AccessUnit.h 定义了存取单元（或者说访问单元），实质是一个类型为NALUnitEBSP的list

NAL.h 定义了NALUnit（NAL单元）和NALUnitEBSP（EBSP类型的NAL单元）

AnnexBwrite.h 实现了把存取单元写入外部数据队列的功能

CommonDef.h 定义各种宏

ContextModel.h/ContextModel.cpp 定义了熵编码需要的上下文模型

ContextModel3DBuffer.h/ContextModel3DBuffer.cpp 定义了熵编码上下文模型需要的3维缓存区

ContextTables.h 定义各种上下文所需的表格，例如二进制化、熵编码等等

encmain.cpp 编码器的主函数

libmd5.h/libmd5.cpp/MD5.h md5的实现

NALwrite.h/NALwrite.cpp 定义了输出的NAL单元（OutputNALUnit）、实现把NAL单元写到外部输出流中（std::ostream）

program\_options\_lite.h/program\_options\_lite.cpp 编码器的选项配置

SEI.h/SEI.cpp 图像增强信息方面

SEIwrite.h/SEIwrite.cpp 写SEI信息到比特流中

SyntaxElementWriter.h/SyntaxElementWriter.cpp 语法元素写入器

TAppEncCfg.h/TAppEncCfg.cpp 编码器应用程序配置类

TAppEncTop.h/TAppEncTop.cpp 继承自TAppEncCfg，编码器应用程序顶层类（或者说编码器的主类）

TComBitCounter.h/TComBitCounter.cpp 比特计数器

TComBitStream.h/TComBitStream.cpp 比特流

TComCABACTables.h/TComCABACTables.cpp定义了CABAC所需要的各种表格

TComDataCU.h/TComDataCU.cpp CU（编码单元的定义）

TComInterpolationFilter.h/TComInterpolationFilter.cpp 插值过滤器

TComList.h 公用的列表的定义，继承自std的list

TComLoopFilter.h/TComLoopFilter.cpp 环路滤波器的实现

TComMotionInfo.h/TComMotionInfo.cpp 运动信息的实现

TComMv.h mv（运动向量）的定义

TComPattern.h/TComPattern.cpp 公共的模式类，定义了YUV三个颜色分量的方位方法和相邻像素的访问方法

TcomPic.h/TcomPic.cpp 是图片类，它包含TComPicSym（图像符号类）和TComPicYuv（yuv数据类：包括原始数据，预测数据、惨差数据）

TComPicSym.h/TComPicSym.cpp 定义了图像符号类，定义了从图像到片和cu的访问方式

TComPicYuv.h/TComPicYuv.cpp 图像的yuv数据类：包括原始数据，预测数据、残差数据

TComPicYuvMD5.cpp 图像的yuv的md5的实现

TComPrediction.h/TComPrediction.cpp 预测的实现（帧内预测/帧间预测）

TComRdCost.h/TComRdCost.cpp率失真代价

TComRdCostWeightPrediction.h/TComRdCostWeightPrediction.cpp 带率失真权重的预测

TComRom.h/TComRom.cpp 全局的变量和函数

TComSampleAdaptiveOffset.h/TComSampleAdaptiveOffset.cpp SAO的定义和实现

TComSlice.h/TComSlice.cpp 片的定义和实现

TComTrQuant.h/TComTrQuant.cpp变换和量化的实

TComWeightPrediction.h/TComWeightPrediction.cpp带权重的预测

TComYuv.h/TComYuv.cpp TComPicYuv和TComYuv的区别，TComYuv可以看作是一个方便操作YUV的类，主要用于预测阶段，从TComPicYuv产生而来，TComYuv的作用没有TComPicYuv那么大

TEncAnalyze.h/TEncAnalyze.cpp 编码器分析类（性能分析类）

TEncBinCoder.h 二进制编码器的定义（定义了二进制化和熵编码的一些类）

TEncBinCoderCABAC.h/TEncBinCoderCABAC.cpp CABAC二进制编码器

TEncBinCoderCABACCounter.h/TEncBinCoderCABACCounter.cpp CABAC比特计数器

TEncCavlc.h/TEncCavlc.cpp CAVLC熵编码器

TEncCfg.h 编码器的基础配置类

TEncCu.h/TEncCu.cpp CU编码器

TEncEntropy.h/TEncEntropy.cpp 熵编码器

TEncGOP.h/TEncGOP.cpp 图像组编码器

TEncPic.h/TEncPic.cpp 图像编码器

TEncPreanalyzer.h/TEncPreanalyzer.cpp 预测分析器编码器

TEncRateCtrl.h/TEncRateCtrl.cpp 速率控制器（帧速率、比特速率等等）

TEncSampleAdaptiveOffset.h/TEncSampleAdaptiveOffset.cpp SAO编码器

TEncSbac.h/TEncSbac.cpp SBAC编码器（SBAC是CABAC的改进,是并行的CABAC算法）

TEncSearch.h/TEncSearch.cpp 运动搜索的实现

TEncSlice.h/TEncSlice.cpp 片编码器

TEncTop.h/TEncTop.cpp 公共的编码器类（与编码器应用类不一样，编码器应用类是一个包装的接口类）

TvideoIOYuv.h/TvideoIOYuv.cpp YUV的I/O类，是用来读取yuv文件的类，读取出来之后把数据放到TComPicYuv中

TypeDef.h 基础类型的定义

WeightPredAnalysis.h/WeightPredAnalysis.cpp 带权预测分析

TCom和TEnc的区别在于，TEnc一般是管理器或者编码器，而TCom则是数据存储类或者算法实现类，例如TEncCu是CU编码器类，而TComDataCU则是CU的数据类（或者说CU数据的操作类，因为实际的数据仍然存放在TComPicYuv中）

四、HM代码学习

# 1. 帧内预测

由于帧内预测涉及到的函数的数量相对于编解码器复杂部分来说少，但事实上大大小小也牵涉到了十几二十个函数，下面针对重点函数进行讨论。

## 1.1 fillReferenceSamples函数

它主要功能是在真正进行帧内预测之前，使用重建后的Yuv图像对当前PU的相邻样点进行赋值，为接下来进行的角度预测提供参考样点值。位置：TLibCommon->Soures Flies->TCompattern.cpp->Void fillReferenceSamples().

这个函数实际上实现的是官方当前标准（JCTVC-J1003）draft 8.4.4.2.2（Reference sample substitution process for intra sample prediction），具体内容我这里就不重复了，有兴趣的朋友可以自己下下来去看看，我先简单把该过程复述一遍：

（1）如果所有相邻点均不可用，则参考样点值均被赋值为DC值；

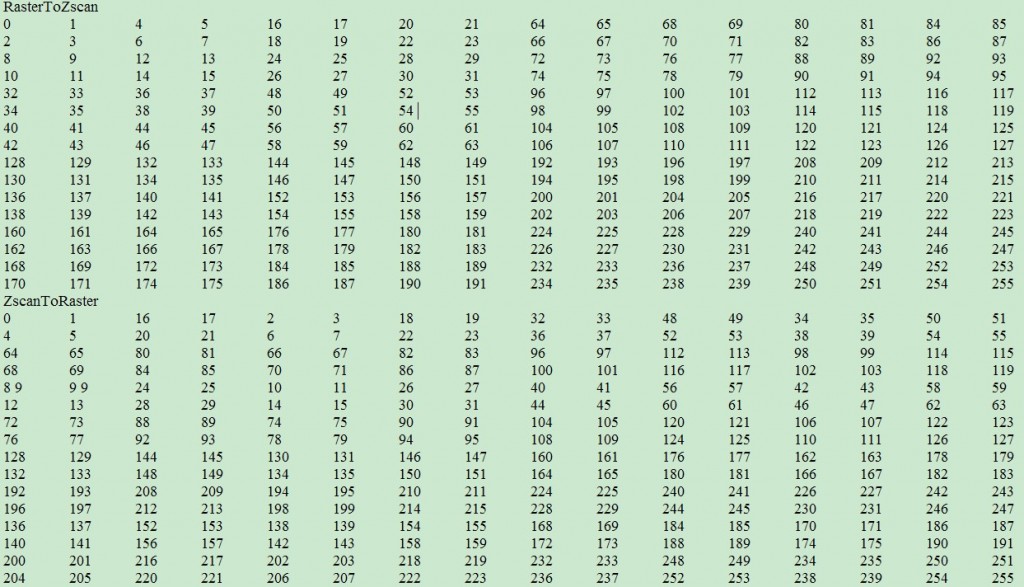
（2）如果所有相邻点均可用，则参考样点值都会被赋值为重建Yuv图像中与其位置相同的样点值；

（3）如果不满足上述两个条件，则按照从左下往左上，从左上往右上的扫描顺序进行遍历，

如果第一个点不可用，则使用下一个可用点对应的重建Yuv样点值对其进行赋值；对于除第一个点外的其它邻点，如果该点不可用，则使用它的前一个样点值进行赋值（前一个步骤保证了前一个样点值一定是存在的），直到遍历完毕。

注：

HEVC定义了Z扫描顺序, 具体到代码中，为了处理的方便，并没有使用上图这种定义方式，而是以4x4块为最小单位，对CU进行分割，同时，为了简化计算，在初始化时定义了几个地址映射的数组，g\_auiRasterToZscan, g\_auiZscanToRaster, g\_auiRasterToX, g\_auiRasterToY



## 1.2 initAdiPattern函数

它的主要功能有三个，

（1）检测当前PU的相邻样点包括左上、上、右上、左、左下邻域样点值的可用性，或者说检查这些点是否存在；

（2）参考样点的替换过程，主要实现的是JCTVC-J1003即draft 8.4.4.2.2的内容，主要由函数fillReferenceSamples来完成，这个在之前的文章已经讨论过了；

（3）相邻样点即参考样点的平滑滤波，主要实现draft 8.4.4.2.3的内容。