三、使用HM进行简单的视频隐写demo

原创

杨酬勤 于 2021-04-22 18:41:38 发布

三、使用HM进行简单的视频隐写

前言

在前两篇博客中介绍了基于HEVC/H.265视频编解码器的视频隐写的基本思路，本文将详细记录我在使用HM做视频隐写的简单实验中遇到的各种问题和解决的思路等。

本次实验参考的博客是dy学长的两篇博客：第一篇博客、第二篇博客

感谢dy学长、另外一位网友和徐老师对我实验过程中的帮助和解答。

一、实验环境

在Windows10 x64的笔记本上，用VS2010对HM12.0进行实验。

HM12.0网盘分享（提取码：zdeu）

二、实验思路

本次实验是我对视频隐写的第一个实验，意在熟悉视频隐写的整个流程，以便将来进行其它的视频隐写实验。所以，这次实验选择了比较简单的对4x4块使用LSB算法，对帧内预测过程中的IPM进行隐写，且不论最终实验评价指标（即PSNR、SSIM等）的优劣。

本次实验是在上一篇博客的基础上进行的，按照上一篇博客的思路，将使用三个HM编解码器：
1、在第一个编码器HM_1中提取出所有的原始载体original_cover涉及到的相关信息——深度depth、划分模式partitionsize和original_IPM值——并保存下来；
2、在外部编写LSB隐写算法，对4x4块的original_IPM进行隐写。这里需要注意的是：并不是所有的4x4块的IPM都适合隐写，对IPM=0、1和34的块，不进行隐写——IPM为0或1时，块是没有预测方向的角度的，此时如若进行隐写修改，则将导致从无角度变成有角度，产生较大失真。IPM为34时，使用LSB进行隐写，若secret_msg==“1”，会使得隐写后的stego_IPM变成35，而在HM中，IPM的取值范围是[0,34]，导致越界。为避免这些影响，简单地将这三种情况排除，不进行隐写。最终，得到经过LSB算法隐写好后的stego_IPM；
3、在第二个编码器HM_2中将stego_IPM放回，并编码视频，得到嵌密视频。这里需要注意的是：HM_1和HM_2的视频参数应该是一模一样的；
4、在第三个解码器HM_3中对嵌密视频进行提取，提取所有隐写后的载体以及相关信息——深度depth、划分模式partitionsize和extract_IPM值；
5、在外部编写LSB提取算法，对4x4块的IPM进行提取，得到extract_msg，并与original_msg进行比对；

三、实验过程

3.1 提取原始载体

使用码流分析器我们可以得知：8x8的块与4x4的块它们的深度都为3，而其它更大的块的深度为2、1、0等，8x8的块与4x4的块之间的差别在于，8x8的块的划分模式是2Nx2N的，而4x4的块的划分模式是NxN的，而其它更大的块的划分模式都是2Nx2N。所以，对于所有深度depth=3、划分模式partitionsize=NxN的块，就是4x4的块，即就是我们要找的原始载体original_cover。

接下来查看TComDataCU.h中的TComDataCU类，这个类中定义了一个CU的各种属性，如该CU在Slice中的位置、Z-order、横坐标、纵坐标、深度、PU类型、编码模式和帧内预测模式等等。（TComDataCU类解释）

```
class TComDataCU
{
private:
  // 绝大部分已经省略
  UInt          m_uiCUAddr;           ///< CU address in a slice                                  在 slice 中的位置，用的是 raster 扫描（从左到右，从上到下）
  UInt          m_uiAbsIdxInLCU;      ///< absolute address in a CU. It's Z scan order            当前 CU 在 LCU 中的位置，位置用 Z 扫描顺序
  UInt          m_uiCUPelX;           ///< CU position in a pixel (X)                             CU 的横坐标
  UInt          m_uiCUPelY;           ///< CU position in a pixel (Y)                             CU 的纵坐标
  UChar*        getDepth              ()                        { return m_puhDepth;        } //返回深度信息
  UChar         getDepth              ( UInt uiIdx )            { return m_puhDepth[uiIdx]; } //返回深度信息
  Char*         getPartitionSize      ()                        { return m_pePartSize;        } //返回划分模式信息
  PartSize      getPartitionSize      ( UInt uiIdx )            { return static_cast<PartSize>( m_pePartSize[uiIdx] ); } //返回划分模式信息
  UChar*        getLumaIntraDir       ()                        { return m_puhLumaIntraDir;           }//返回帧内预测模式
  UChar         getLumaIntraDir       ( UInt uiIdx )            { return m_puhLumaIntraDir[uiIdx];    }//返回帧内预测模式
};
// TComDataCU类定义结束
```

再看一下PartSize类，这是一个枚举类型的类，其中2Nx2N的值为0，NxN的值为3。

```
/// supported partition shape
enum PartSize
{
  SIZE_2Nx2N,           ///< symmetric motion partition,  2Nx2N                    0
  SIZE_2NxN,            ///< symmetric motion partition,  2Nx N                    1
  SIZE_Nx2N,            ///< symmetric motion partition,   Nx2N                    2
  SIZE_NxN,             ///< symmetric motion partition,   Nx N                    3
  SIZE_2NxnU,           ///< asymmetric motion partition, 2Nx( N/2) + 2Nx(3N/2)    4
  SIZE_2NxnD,           ///< asymmetric motion partition, 2Nx(3N/2) + 2Nx( N/2)    5
  SIZE_nLx2N,           ///< asymmetric motion partition, ( N/2)x2N + (3N/2)x2N    6
  SIZE_nRx2N,           ///< asymmetric motion partition, (3N/2)x2N + ( N/2)x2N    7
  SIZE_NONE = 15        //                                                       15，指的是没有了
};
```

所以，把所有块的深度depth、划分模式partitionsize和预测模式IPM保存下来，那些depth=3&&partitionsize=3的IPM就是可供隐写的载体。

```
for(int i = 0 ; i < pcCU->getTotalNumPart() ; i ++){
 // printf("i = %d, 预测模式 = %d, 深度 = %d, 模式 = %d\n", i, pcCU->getLumaIntraDir(i), pcCU->getDepth(i), pcCU->getPartitionSize(i));
 fout_IPM << int(pcCU->getLumaIntraDir(i)) << ",";
 fout_depth << int(pcCU->getDepth(i)) << ",";
 fout_partition << int(pcCU->getPartitionSize(i)) << ",";
}
```

3.2 使用LSB隐写算法进行隐写

首先需要对所有块进行统计，计算出4x4块的个数，以便确定隐秘信息的长度，来生成隐秘信息original_msg。

```
# 统计一下4x4的个数
count_4x4_IPM = 0
for i in range(len(depth_num_list)):
    for j in range(len(depth_num_list[i])):
        if (depth_num_list[i])[j] == 3 and (partitionsize_num_list[i])[j] == 3:
            count_4x4_IPM += 1
print("count_4x4_IPM = ", count_4x4_IPM)
```

根据计算得到的个数count_4x4_IPM，生成隐秘信息original_msg。遍历所有块，对4x4块的合适IPM进行隐写。

```
# 对所有4x4块隐写
if depth_num_list[i][j] == 3 and partitionsize_num_list[i][j] == 3:
 msg = original_msg_list[0][idx_msg]
 # 如果IPM==34或者IPM==1或者IPM==0，则不隐写
 if IPM == 34 or IPM == 0 or IPM == 1:
  temp_IPM_list.append(IPM)
 else:
  IPM_binary_str = Integer_To_BinaryStr(IPM)
  for k in range(len(IPM_binary_str)):
   # msg==1时，LSB=1
   if msg == "1":
    IPM_binary_str = IPM_binary_str[:7] + "1"
   # msg==0时，LSB=0
   else:
    IPM_binary_str = IPM_binary_str[:7] + "0"
  IPM_num_stego = BinaryStr_To_Integer(IPM_binary_str)
  temp_IPM_list.append(IPM_num_stego)
  idx_msg += 1
# 不是4x4的块
else:
 temp_IPM_list.append(IPM)
```

3.3 放回含密载体

在HM_2中，把stego_IPM读取进来，并赋值给uiBestPUMode，替代它原来的IPM值。

这里需要注意的是：HM_1和HM_2的视频参数信息——例如QP、编码帧数等——应是一模一样的。

```
//=== update PU data ====
// 20210415周四 对所有块赋值stego_IPM
stego_num = atoi(stego_vectorString[order_num].c_str());// stego_IPM的值
uiBestPUMode = stego_num;
pcCU->setLumaIntraDirSubParts     ( uiBestPUMode, uiPartOffset, uiDepth + uiInitTrDepth ); // 这行代码是它本来就有的
pcCU->copyToPic                   ( uiDepth, uiPU, uiInitTrDepth );
```

3.4 提取含密载体

配置HM12.0解码端相关信息，按照3.1的方法，把深度depth、划分模式partitionsize和预测模式IPM保存下来。（HM解码端配置教程）

3.5 使用LSB提取算法进行比对

使用LSB的提取算法，把4x4块的合适IPM的LSB提取出来，并与原始的隐秘信息进行比对。

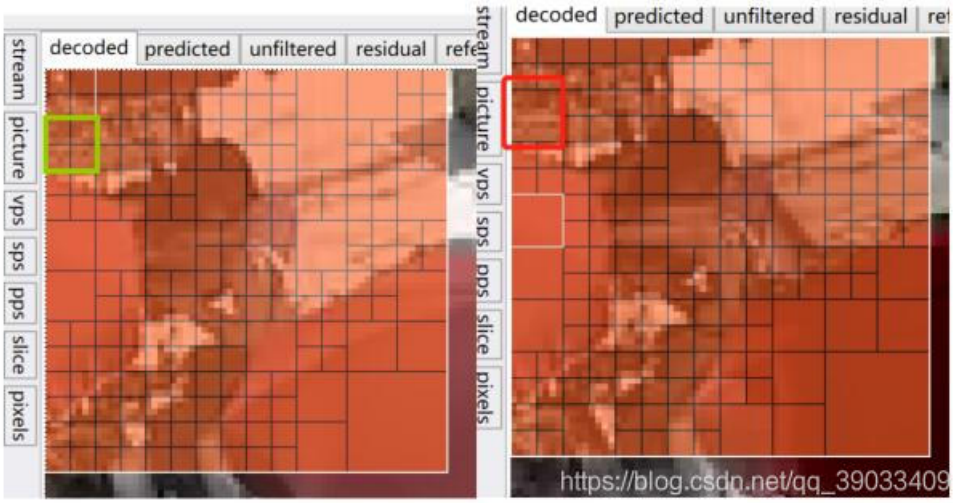
```
# 5、提取所有4x4的LSB
extract_msg = ""
for i in range(len(extract_IPM_list)):
    for j in range(len(extract_IPM_list[i])):
        # 判断是否是4x4块
        if depth_num_list[i][j] == 3 and partitionsize_num_list[i][j] == 3: # 是4x4块，提取LSB
            if extract_IPM_list[i][j] == 34 or extract_IPM_list[i][j] == 0 or extract_IPM_list[i][j] == 1: # IPM
=34、0、1则不提取
                continue
            else:
                temp_str = Integer_To_BinaryStr(extract_IPM_list[i][j])
                if temp_str[-1] == "1":
                    extract_msg += "1"
                else:
                    extract_msg += "0"
        else: # 不是4x4块，则继续
            continue
print("extract_msg = ", extract_msg)

# 6、判断extract_msg和original_msg是否一致
count = 0
for i in range(len(extract_msg)):
    if extract_msg[i] == original_msg_list[0][i]:
        continue
    else:
        count += 1
        print(i, extract_msg[i], original_msg_list[0][i])
print("提取正确率 = ", format((1 - count / len(extract_msg)) * 100, '.4f'), "%")
```

四、碰到的问题以及解决思路

4.1 块的结构体发生变化

在第二个编码器HM_2中，把stego_IPM放回后，编码视频，结果发现隐写后的视频的块结构发生了很大变化——例如4个4x4的块合并成了1个8x8的块——使得4x4块的数量发生了变化，4x4块的位置也发生了变化。这也就意味着，利用LSB算法提取出来的隐秘信息必定是错误的。



4.1.1 产生的原因

由于块的划分选择是一个递归的过程，一个块递归至8x8时，会计算当前划分模式的cost值，并继续向下递归至4个4x4，计算4个4x4划分模式的cost值，最终选择cost值小的那种划分模式。

例如(1016, 752)这个块，它应是4个4x4的块（IPM值分别为13、13、2、11），但是却合并成了1个8x8的块（IPM为0），说明是由于8x8块的cost值小于4个4x4块的cost值之和导致的。

```
x = 1016, y = 752, IPM = 0
yl_order_num = 49140, yl_stego_num = 13
x = 1016, y = 752, IPM = 13
yl_order_num = 49141, yl_stego_num = 13
x = 1016, y = 752, IPM = 13
yl_order_num = 49142, yl_stego_num = 2
x = 1016, y = 752, IPM = 2
yl_order_num = 49143, yl_stego_num = 11
x = 1016, y = 752, IPM = 11
```

4.1.2 解决方法

参考dy学长的博客，在递归过程中，通过修改cost来控制它的划分模式：判断当前递归层次的深度和划分模式，如果和HM_1中保存下来的该CU的原始深度、原始划分模式一致，则修改cost为-9999.9999。

```
// 20210415周四 在这里对递归中的depth和partitionsize进行判断，如果符合判断条件，则修改它的cost，使其结构体不发生变化
depth_num = atoi(depth_vectorString[j].c_str());
partitionsize_num = atoi(partitionsize_vectorString[j].c_str());
if(pcCU->getDepth(uiPartOffset) == depth_num && pcCU->getPartitionSize(uiPartOffset) == partitionsize_num){
 uiPUDistY = 0;
 uiPUDistC = 0;
 dPUCost   = -999999.9999;
}
```

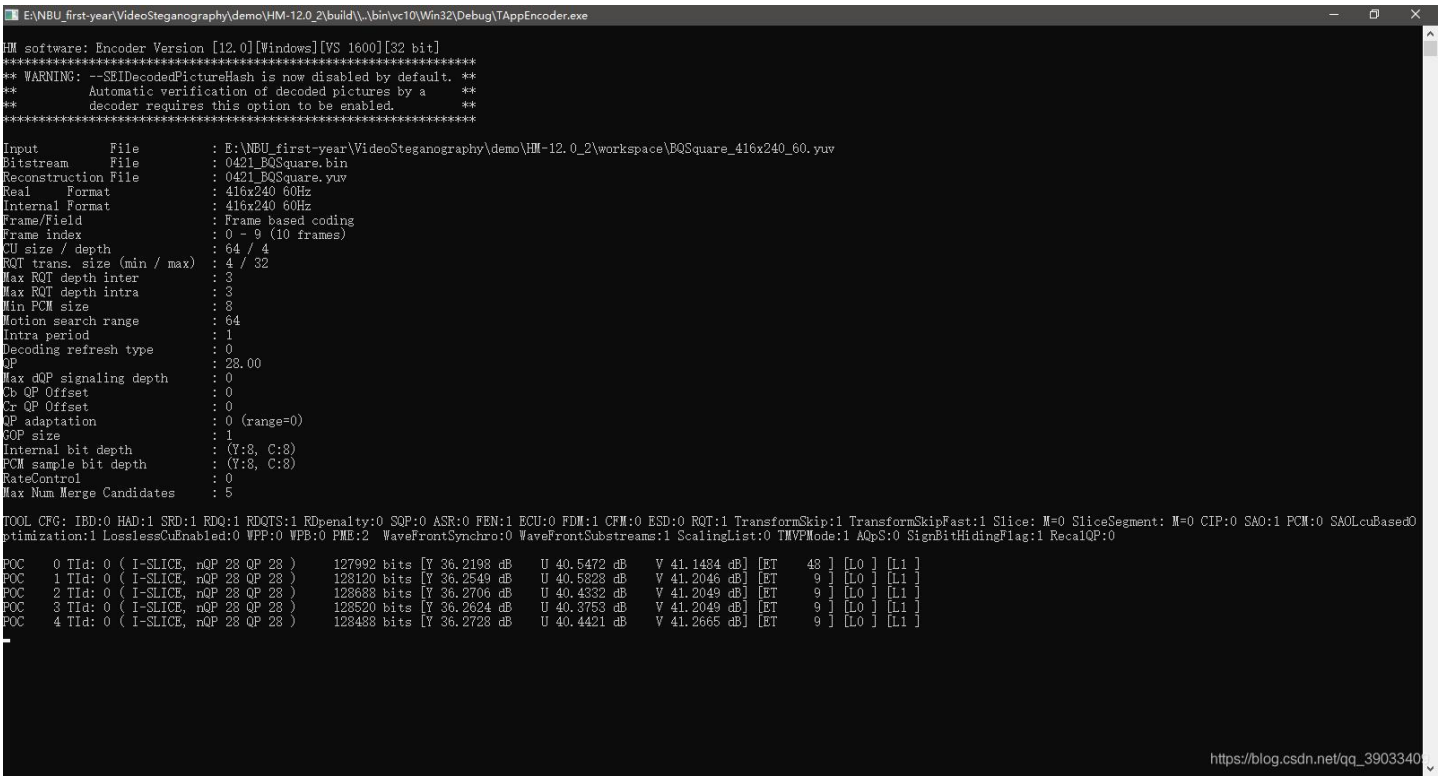
在递归返回过程中，通过设置flag来控制它的划分模式：判断当前递归返回时的深度和划分模式，如果当前层次的信息与原始信息一致，则设置flag=1，并在TEncCU::xCheckBestMode()的判断中新增一个刷新条件“ || flag == 1”。

```
 // 20210419 判断深度和划分模式是否一致
 //printf("x = %d, y = %d, depth = %d, partitionsize = %d, z-order = %d\n", rpcTempCU->getCUPelX(), rpcTempCU->getCUPelY(), uiDepth, eSize, rpcTempCU->getZorderIdxInCU());
 z_order_num = rpcTempCU->getZorderIdxInCU();// z-order
 o_num = rpcTempCU->getZorderIdxInCU() + rpcTempCU->getAddr() * 256 + rpcTempCU->getPic()->getPOC() * rpcTempCU->getPic()->getNumCUsInFrame() * 256;// 块的索引序号
 p_num = atoi(p_vectorString[o_num].c_str());// 原始的partitionsize的值
 d_num = atoi(d_vectorString[o_num].c_str());// 原始的depth的值
 if(eSize == p_num && uiDepth == d_num)
  flag = 1;
 else
  flag = 0;
```

4.2 一个有趣的现象

每次试验，在编码视频的过程中，都是第一帧（第0个POC）编码时间较长，接下来几帧的编码时间与原始的HM12.0压缩视频时间相同。

例如在编码BQSquare.yuv时，第1帧（第0个POC）编码的时间为48秒，剩余几帧编码的时间为9秒——这与使用HM12.0在QP=28时压缩BQSquare.yuv所用时间一样。



4.2.1 产生的原因

猜测：由于在HM_2中需要读取所有保存在硬盘上的载体相关信息，所以在第一帧中，需要将这些保存在硬盘上的信息读取进来保存到缓存中，导致了花费时间较长，而在接下来的递归中，由于是直接从缓存中读取数据，所以花费的时间较短。

用代码测试了一下，发现只有第一次的runtime是一个较大的数，接下来的runtime都为0，即证明了我的猜测。

```
clock_t begin_time = clock();// 开始时间
// 20210412 周一 把stego_IPM从.txt文件中读取进来，保存在一个栈中
while(getline(stego_in, stego_str, ',')){
 // 1、把file_str按照空格分成一个个字符串型的数字
 stringstream stego_input(stego_str);
 while(stego_input >> stego_result)
  stego_vectorString.push_back(stego_result); // vectorString中已经有了一个个按空格分开来的字符串型的数字了
}
clock_t end_time = clock();// 结束时间
printf("runtime = %lf\n", float(end_time - begin_time) / 1000); // 判断了一下，发现只有第一次读的时候是花费时间的，后面的runtime都是0了
system("pause");
```

在我看来，这是很重要的一点，因为之前写代码的时候一直担心在递归函数中读取信息，会不会每递归一次，就要花很长时间去读信息。现在看来，除了第一次递归读取信息时花费较多时间以外，接下来都是瞬时完成的。

也就是说，视频编码时间只是在第一次递归读取时增加了，接下来都是按正常的速率进行编码的。对于那些分辨率大的视频，第一帧编码时间较长，只是单纯的因为它们的数据量大而已，而不是由于递归重复读取导致的时间过长。

4.3 视频产生较大失真

最终，利用软件分析隐写后的视频质量，发现最终的PSNR值非常低，也就是说视频质量很差，有肉眼可见的较大失真。

4.3.1 产生的原因

经分析，可能导致PSNR值过低的原因有两点：

一、在LSB隐写时，没有考虑IPM=0以及IPM=1的情况，对这两种无角度的IPM都进行了隐写，导致IPM从无角度预测变成了有角度预测，产生较大失真；
二、在HM_2中，对uiBestPUMode强行赋值的行为，可以视作在写码流时修改IPM，这种修改方式由于没有对修改后的IPM进行再编码，使得接下来的块产生巨大失真；
三、在HM_2中，虽然保证了所有4x4的位置不发生变化，但是没有保证更大块的结构不变，那么就有可能4个8x8的块合并成了1个16x16的块，4个无角度的IPM合并成了1个有角度的IPM，导致这个大块（平坦区域）产生较大失真；

4.3.2 解决方法

针对第一种可能性，在LSB隐写算法中，对IPM=0和IPM=1进行了特殊处理——即对IPM=0和IPM=1的块不进行隐写——但是效果不理想，视频仍有较大失真。

~~*针对第三种可能性，应对重新赋值后的CU进行再编码，但是不知道该如何去实现。~~

~~*针对第三种可能性，不知道该如何解决。这两点算作是留着优化的点吧。~~

20210423更新：
由于直接覆盖stego_IPM的方式，是在块计算DCT/DST系数等操作之后进行的，所以会产生巨大的失真。

所以，在HM_2中采用新的策略——在遍历候选列表的时候，判断递归的深度和递归的划分模式，若depth和partitionsize都和HM_1中保存下来的值相同时，则把候选列表长度设置成1，并把候选列表中的值设置成stego_IPM。这样一来，编码器就会自动地根据stego_IPM进行编码，从而直接省去了修改cost的操作。

```
UInt    uiBestPUMode  = 0;
  UInt    uiBestPUDistY = 0;
  UInt    uiBestPUDistC = 0;
  Double  dBestPUCost   = MAX_DOUBLE;


  // 20210423周五 在这里对递归中的depth和partitionsize进行判断，如果符合判断条件，使他的预测模式只有这个值
  order_num = pcCU->getPic()->getPOC() * pcCU->getPic()->getNumCUsInFrame() * 256 + pcCU->getAddr() * 256 + pcCU
->getZorderIdxInCU() + uiPartOffset;// 块的索引值
  depth_num = atoi(depth_vectorString[order_num].c_str());
  partitionsize_num = atoi(partitionsize_vectorString[order_num].c_str());
  stego_num = atoi(stego_vectorString[order_num].c_str());// stego_IPM的值
  if(pcCU->getDepth(uiPartOffset) == depth_num && pcCU->getPartitionSize(uiPartOffset) == partitionsize_num){
   numModesForFullRD = 1;
   uiRdModeList[0] = stego_num;
  }

  // 遍历候选列表
  for( UInt uiMode = 0; uiMode < numModesForFullRD; uiMode++ )
  {
   // set luma prediction mode
   UInt uiOrgMode = uiRdModeList[uiMode];

   pcCU->setLumaIntraDirSubParts ( uiOrgMode, uiPartOffset, uiDepth + uiInitTrDepth );

   // set context models
   if( m_bUseSBACRD )
   {
    m_pcRDGoOnSbacCoder->load( m_pppcRDSbacCoder[uiDepth][CI_CURR_BEST] );
   }

   // determine residual for partition
   UInt   uiPUDistY = 0;
   UInt   uiPUDistC = 0;
   Double dPUCost   = 0.0;
   // 等等等，已略
  }
```

通过这种代码策略，最终隐写后的视频质量非常的高，隐写效果非常好，解决了失真严重的问题。

五、总结

通过这次demo的练习，对HM的代码有了初步的理解，对HEVC的一些知识有了更深的理解。

对于实验过程中碰到的种种问题，只有自己碰到了、解决了，才是真正的搞懂了。仅仅是参考别人的博客，纸上谈兵，看着好像是理解了，其实还是不然的。

虽然我写博客的初衷是希望能够给以后的人一些经验，但是有些东西还是得自己去试过了，尝试过了，错过了，才是最好的办法。