实验报告

实验题目: h.261运动矢量搜索算法

姓名: 尹浚宇 学号: 161130118

【实验目的及要求】

探讨运动搜索算法的改进

提供的代码：

Main.cpp，CodeInter.cpp, CodeOneIntra.cpp, CountBit.cpp, Dct.cpp, IO.cpp, Not\_est.cpp, OutputVlc.cpp, Pred.cpp, RateControl.cpp, YUVto261.cpp, config.h, Global.h, Indices.h, OutputVlc.h。

代码的编译：与系统无关，可在linux或windows下编译，建议使用VisualStudio系统编译，也可以在其它编译器里编译。

在VS里编译过程：打开VS，创建新的空项目，将所有源文件添加，编译即可。

运行命令：程序名 输入YUV文件名 输出的文件名 待编码的帧数 帧频 P帧数 帧内帧的量化阶 帧间帧的量化阶

例如：程序名 foreman.yuv foreman.263 300 25 5 8 4

实验要求：将给定代码Mot\_est.cpp中MotionEstimation函数的全搜索算法替换成对数搜索或层次搜索，比较替换前后的视频质量和编码效率。

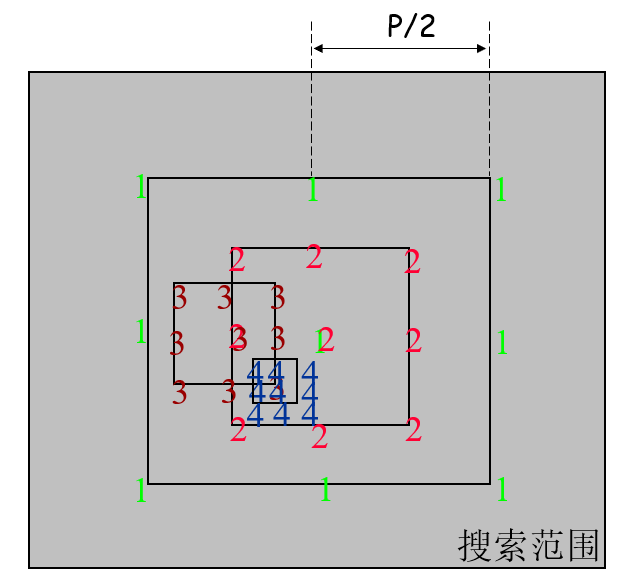
实验提交：Mot\_est.cpp文件和本实验报告

【实验原理】（简要说明本实验项目所涉及的理论知识）

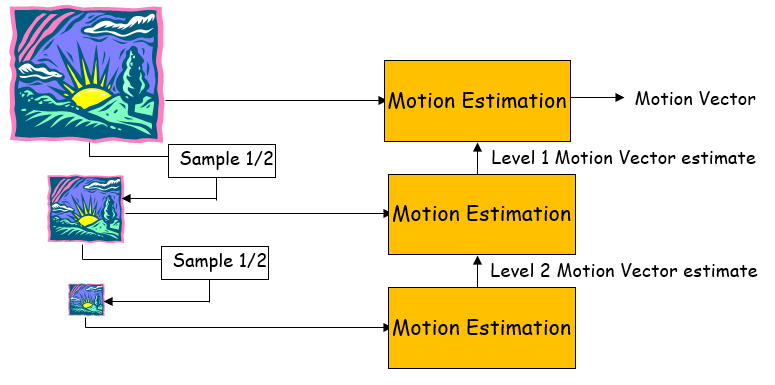
运动补偿算法的核心思想是在时间域上消除画面之间的冗余信息来压缩视频. 由于画面内容有连贯性, 因此当前画面的内容可以看作是前面画面部分内容的位移. 同时, 画面各处的运动矢量各不相同, 因此, 该算法将一帧画面细分成各个小块(宏块), 然后再以宏块为单位, 找出两帧画面中相应宏块之间的位移关系, 也就是运动矢量.

于是, 这里的核心就是找到两帧画面中最匹配的宏块. 对于一个待预测的宏块而言, 由于在整幅图像内搜索匹配耗时会非常大, 且一般而言, 宏块和它的最佳匹配在相邻两帧内的距离不会太远, 于是在搜索时, 只需以待预测的宏块作为中心, 确定一个搜索范围即可. 同时, 两个宏块之间的匹配度可由方差和或者绝对误差和计算.

这里在搜索宏块时, 有很多策略, 该项目涉及了全搜索, 对数搜索和层次搜索. 全搜索即用搜索范围的所有宏块与目标宏块计算匹配度, 这部分代码由实验框架给出. 这里我选择完成的是对数搜索, 该算法的思想是每一步搜索以目标宏块为中心, 搜索步长为半宽的正方形上的八个点, 然后以这八个点中匹配度最大的点为中心, 搜索步长减半重复上述过程, 直到搜索步长小于1为止, 图示如下(图中数字表示搜索的轮数):

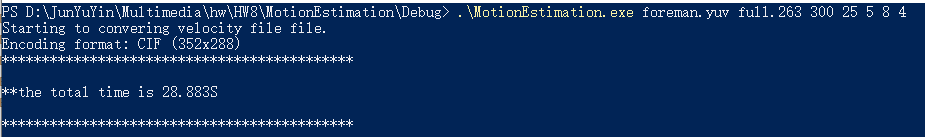


层次搜索即将图像连续两次下采样为原图的一半, 然后每一层宏块的大小也相应减半, 然后对最底层(图像最小的那层)进行全搜索, 然后将搜索到的最佳匹配的宏块逐层向上进行运动向量搜索, 在上层搜索时和对数搜索类似, 只搜索最佳匹配的宏块在该层对应的宏块周围的9个点(包括自身), 最后在原图上找到最佳匹配宏块, 图示如下:

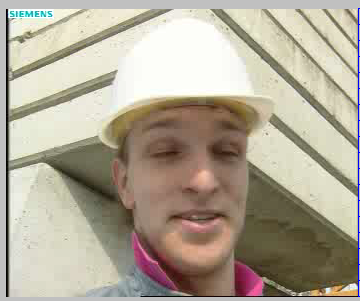


【实验结果】（比较转换结果，原有转换值与自己方法的比较，包括数值比较和显示输出比较）

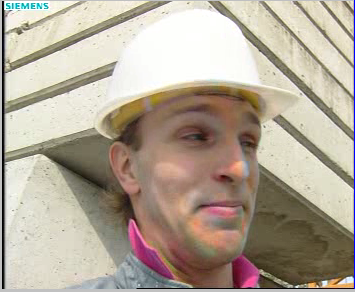
全搜索算法的运行时间, 输出文件大小和显示效果(前五帧):



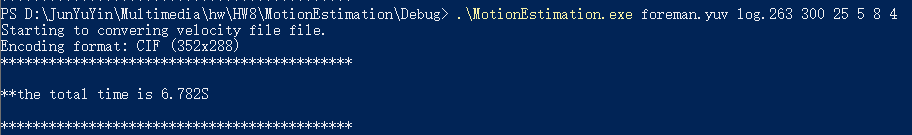




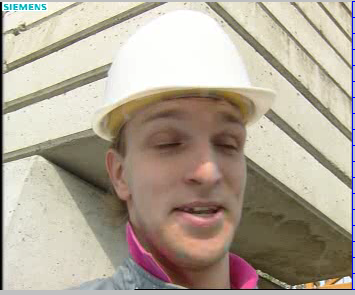


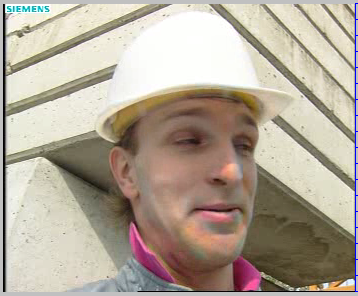


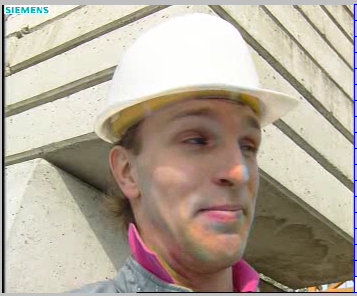
对数搜索算法的运行时间, 输出文件大小和显示效果(前五帧):











根据上面的对比, 我们可以看出全搜索的编码时间大约对数搜索编码时间的4-5倍, 但因为全搜索能为每一帧画面找到全局最优的运动向量, 所以全搜索编码文件的大小比对数搜索编码文件稍小一些. 最后就显示效果而言, 两种编码的文件在播放时肉眼基本看不出有差异.

【实验小结】（对本次实验的心得体会、思考和建议）

在实际应用中, 如果对编码的时间不敏感, 而期望所占空间尽量小, 应该选用全搜索算法; 如果要求编码的时间尽可能短, 则应该选用对数搜索或者层次搜索算法. 但现代计算机的存储空间一般而言相对充足, 而且从上面看出各种搜索方法对最后的视频质量似乎没有太大影响, 所以我觉得在大多数情况下, 使用对数搜索或者层次搜索是更好的选择.

这个链接<http://www.angelfire.com/in/H261/h263player.html>是一个H263文件的播放器, 我觉得可以在之后的教学中提供该链接.