GY

中华人民共和国广播电视和网络视听行业标准

GY/T 368—2023

先进高效视频编码

Advanced and efficient video coding

2023－05－11 发布　　2023－05－11 实施

国家广播电视总局　发布

目　　次

前　言

本文件按照GB/T 1.1—2020《标准化工作导则 第1部分：标准化文件的结构和起草规则》的规定起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由全国广播电影电视标准化技术委员会（SAC/TC 239）归口。

本文件起草单位：国家广播电视总局广播电视科学研究院、中央广播电视总台、国家广播电视总局广播电视规划院、北京大学、鹏城实验室、华为技术有限公司、深圳市大疆创新科技有限公司、浙江大学、清华大学、三星电子株式会社、上海海思技术有限公司、北京大学深圳研究生院、中山大学、北京字节跳动科技有限公司、上海大学、联发博动科技（北京）有限公司、上海国茂数字技术有限公司、哈尔滨工业大学、中国科学院计算技术研究所、武汉大学、中国科学技术大学、电子科技大学、广东博华超高清创新中心有限公司、腾讯科技（深圳）有限公司、杭州海康威视数字技术股份有限公司、北京达佳互联信息技术有限公司、阿里云计算有限公司、OPPO广东移动通信有限公司、浙江大华技术股份有限公司、同济大学、绍兴文理学院、绍兴市北大信息技术科创中心、北京博雅睿视科技有限公司、杭州当虹科技股份有限公司、咪咕文化科技有限公司。

本文件主要起草人：高文、盛志凡、黄铁军、马思伟、郭晓强、潘晓菲、宁金辉、郑萧桢、郑建铧、梁凡、王苫社、周芸、朱易、王惠明、杨海涛、虞露、何芸、王荣刚、赵海武、王稷、李忠良、赵日洋、范晓鹏、林和源、张伟民、胡潇、刘博、张乾、黎政、魏娜、李革、李琳、徐嵩、张嘉琪、李俊儒、傅天亮、张玉槐、赵寅、张莉、朴银姬、王振宇、范奎、马祥、郑建成、欧阳晓、王凡、王力强、牛犇犇、高小丁、谢熙、许桂森、徐巍炜、贾川民、雷萌、简云瑞、王琦、陈焕浜、鲁晓牧、何鸣、马海川、李一鸣、王梦杨、刘杉、许晓中、王英彬、胡晔、王莉、孙煜程、陈方栋、曹小强、潘冬萍、修晓宇、陈伟、郭哲伟、陈杰、李新伟、廖如伶、王东、谢志煌、林聚财、江东、方诚、张雪、方瑞东、林涛、周开伦、王淑慧、赵利平、于化龙、林翔宇、王迪、魏亮、赵海英、龙仕强、韦裕京、韩巍、陈勇、姜波。

引　言

本文件的发布机构提请注意，声明符合本文件时，可能涉及到6.1、6.2、7.1、7.2、8.3、9.1、9.2、9.5～9.11、9.17相关的专利的使用。专利列表如下：

序号	章条编号	专利号	专利名称	专利权人
1	6.1	200710126108.5	实现随机访问的方法及解码器	华为技术有限公司
2	6.1	201910679070.7	图像显示顺序的确定方法、装置和视频编解码设备	华为技术有限公司
3	6.2	201410253649.4	视频编码方法和解码方法和相关装置	华为技术有限公司
4	7.1	201010135828.X	视频数据的打包、编解码方法及装置及系统	华为技术有限公司/清华大学
5	7.1	201110007657.7	多图像块划分的编解码方法和装置	华为技术有限公司/清华大学
6	7.1.3	03157076.3	一种定位编码图像标识的方法	北京大学
7	7.1.3.1，7.2.3.1	202010446922.0	按图像内容特征进行帧级划分和快速编解码的方法和装置	绍兴文理学院
8	7.1.5	201680091245.7	用于对画面轮廓线的编码单元进行编码或解码的方法和装置	三星电子株式会社
9	7.1.6	201210546675.7	一种头信息编解码、解码方法及装置	北京大学
10	7.1.6	201811198705.3	一种编解码的方法和装置	华为技术有限公司/清华大学
11	7.1.6	201910254106.7	视频编码方法、视频解码方法及相关设备	华为技术有限公司/清华大学
12	7.1.6,7.2.6,8.3,8.3.3，8.3.4.11，9.5.8.6.4	201580072015.1	使用高精度跳过编码的视频编码设备和视频解码设备及其方法	三星电子株式会社
13	7.1.6,7.2.6,8.3,9.5.8.2,9.5.8.4,9.5.8.5	PCT/KR2018/003800	Apparatus and Method for Encoding Motion Vector Determined using Adaptive Motion Vector Resolution, and Apparatus and Method for Decoding Motion Vector	三星电子株式会社
14	7.1.6,7.2.6,8.3,9.5.8.4	201910011266.9	对运动矢量信息进行编/解码的方法及装置	三星电子株式会社
15	7.1.6，7.2.11，8.3.3.2，8.3.4	202010446922.0	按图像内容特征进行帧级划分的快速编解码方法	北京博雅睿视科技有限公司
16	7.1.7，8.3.3	201910144295.2	一种量化块的解码方法、装置及电子设备	北京大学深圳研究生院
17	7.1.8	201110347691.9	一种视频图像滤波处理方法和装置	北京大学
18	7.1.9，7.2.9，9.11.2.3	202010509307.X	图像滤波方法、装置、设备及可读存储介质	咪咕文化科技有限公司/北京大学/中国移动通信集团有限公司

19	7.1.9，7.2.9，9.11.2.3	202010509322.4	图像滤波方法、装置、设备及存储介质	咪咕文化科技有限公司/北京大学/中国移动通信集团有限公司
20	7.1.11，7.2.11	202110377059.2	对分量下梁样格式数据进行点预测的编码解码方法及装置	同济大学
21	7.1.11，7.2.11，8.3.3.1，8.3.3.2，8.3.4.1，8.3.4.24	202110628553.1	一种串长度参数编解码方法和装置	绍兴市北大信息技术科创中心
22	7.1.11，7.2.11，8.3.4.1，8.3.4.24	202110630826.6	不同类型串采用不同长度二值化方案的编解码方法及装置	同济大学
23	7.1.11，7.2.11，8.3.4.1，8.3.4.24	202011428699.3	一种串长度参数混合编码、解码方法及装置	绍兴文理学院
24	7.1.11，7.2.11，8.3.4.24	202011324094.X	一种串长度参数编码、解码方法和装置	绍兴文理学院
25	7.1.11，7.2.11，8.3.4.24	202011490545.7	融合帧内块复制和帧内串复制的编码参数的编码、解码方法和装置	绍兴文理学院
26	7.1.11，7.2.11，9.7.3.2	202010493939.1	对分量下采样格式数据进行串预测的数据编码和解码方法	同济大学
27	7.1.11，7.2.11，9.7.3.3	202010488925.0	使用点预测和常现位置数组的数据编码方法和解码方法	同济大学
28	7.2	201910888383.3	视频解码方法、视频编码方法、装置、设备及存储介质	华为技术有限公司
29	7.2	201510150090.7	图像编解码方法和相关装置	浙江大学/华为技术有限公司
30	7.2	201611228200.8	图像编解码方法及装置	浙江大学/华为技术有限公司
31	7.2.5	201910372891.6	视频译码器及相应方法	华为技术有限公司
32	7.2.5	201910246994.8	块划分方法、视频编解码方法、视频编解码器	华为技术有限公司
33	7.2.5	201910017097.X	一种图像块划分方法及装置	华为技术有限公司
34	7.2.5	201810581662.0	一种编解码方法及装置	华为技术有限公司/清华大学
35	7.2.5	201811198624.3	图像块的划分方法和装置	华为技术有限公司/清华大学
36	7.2.7，8.3.3	201910145192.8	一种量化系数结束标志位的上下文模型选取方法及装置	北京大学深圳研究生院
37	8.3	201910335981.8	视频编码器、视频解码器及相应方法	华为技术有限公司
38	9.1，9.2.6，7.2.2	201310695685.1	基于加权量化的视频压缩编解码方法及编解码器	北京大学深圳研究生院
39	9.2.2	201910704770.7	视频编码、解码方 法、装置及计算机存储介质	华为技术有限公司
40	9.2.6	201010155175.1	频带加权量化编解码方法和装置	华为技术有限公司

41	9.2.6	200780000403.4	在编解码中的实现量化的方法和装置	华为技术有限公司/清华大学
42	9.2.6	200710193851.2	量化模式、图像编码、解码方法、编码器、解码器及系统	华为技术有限公司
43	9.2.6，9.6.2	201910141265.6	一种反量化方法、系统、设备及计算机可读介质	北京大学深圳研究生院
44	9.5.4	201680091331.8	通过块映射来对图像进行编码或解码的方法和装置	三星电子株式会社
45	9.5.6.3.2	201980005257.7	视频编解码的方法和装置	深圳市大疆创新科技有限公司
46	9.5.6.5，9.7.3.2，9.7.3.3	202011465573.3	一种对数据进行有损或无损压缩的编码、解码的方法或装置	绍兴文理学院
47	9.5.6.5.6	202110632191.3	限制点预测常现位置及其点矢量数目的编解码方法及装置	同济大学
48	9.5.7.8.9	202010504292.8	帧间预测方法、编码器、解码器以及计算机存储介质	OPPO广东移动通信有限公司
49	9.5.8	03143429.0	一种视频编解码中的运动矢量预测方法和装置	华为技术有限公司
50	9.5.8	200810102353.7	运动矢量的缩放方法和装置、编解码方法和系统	华为技术有限公司
51	9.5.8.4	201310008682.6	一种运动矢量预测的方法	北京大学
52	9.5.8.4.2	201310034524.8	基于方向和距离判别的运动矢量预测方法	北京大学深圳研究生院
53	9.6	201210019843.7	一种获得变换块尺寸的方法和模块	华为技术有限公司/清华大学
54	9.6	201811198621.X	图像块的变换、反变换方法和装置	华为技术有限公司/清华大学
55	9.6	201810278488.2	图像块编码中的变换方法、解码中的 反变换方法及装置	华为技术有限公司/清华大学
56	9.6.5.3，9.7.1.5	201910796327.7	参数获取方法、像素点对选择方法及相关设备	咪咕文化科技有限公司/北京大学
57	9.6.5.3，9.7.1.5	202110855539.5	像素点对选择方法、设备及计算机可读存储介质	咪咕文化科技有限公司/北京大学
58	9.7.1.5	202010510423.3	图像编码方法、图像解码方法及相关装置	OPPO广东移动通信有限公司
59	9.7.4.2，9.7.5	201210546537.9	一种多方向的帧内预测编解码方法及装置	北京大学
60	9.8	201910108004.4	一种帧间预测的方法和装置	华为技术有限公司
61	9.8.3.2	201410127457.9	一种针对色度的插值方法及滤波器	北京大学深圳研究生院
62	9.8.5.1，9.8.5.2	PCT/CN2020/077491	图像预测方法、编码器、解码器以及存储介质	OPPO广东移动通信有限公司
63	9.9	201810992362.1	视频编码器、视频解码器及相应方法	华为技术有限公司
64	9.9	201711494258.1	图像的预测方法、装置及编解码器	华为技术有限公司
65	9.9	201810118179.9	一种双向帧间预测方法及装置	华为技术有限公司/浙江大学

66	9.10	201310128415.2	一种视频编解码环路滤波的实现方法	北京大学
67	9.11	202110220816.5	一种图像处理方法、装置、设备及可读存储介质	咪咕文化科技有限公司/北京大学/中国移动通信集团有限公司
68	9.17	201711025757.6	确定仿射编码块的运动矢量的方法和装置	华为技术有限公司

本文件的发布机构对于该专利的真实性、有效性和范围无任何立场。

该专利持有人已向本文件的发布机构保证，他愿意同任何申请人在合理且无歧视的条款和条件下，就专利授权许可进行谈判。该专利持有人的声明已在本文件的发布机构备案，相关信息可以通过以下联系方式获得：

联 系 人：黄铁军
地　　址：北京大学理科2号楼2641室
邮政编码：100871
电子邮件：tjhuang@pku.edu.cn
电　　话：+8610-62756172
传　　真：+8610-62751638
网　　址：http://www.avs.org.cn

请注意除上述专利外，本文件的某些内容仍可能涉及专利。本文件的发布机构不承担识别专利的责任。

先进高效视频编码

1 范围

本文件规定了适应多种比特率、分辨率和质量要求的高效视频压缩方法编码位流的结构、语法、语义和解析、解码过程。

本文件适用于广播电视、网络电视、网络视音频、数字存储媒体、静止图像序列等视频编码。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 41808—2022 高动态范围电视节目制作和交换图像参数值

GB/T 41809—2022 超高清晰度电视系统节目制作和交换参数值

GY/T 155—2000 高清晰度电视节目制作及交换用视频参数值

GY/T 257.1—2012 广播电视先进音视频编解码 第1部分：视频

GY/T 299.1—2016 高效音视频编码 第1部分：视频

GY/T 358—2022 高动态范围电视系统显示适配元数据技术要求

ISO/CIE 11664-1:2019 色度 第1部分：标准比色观测器（Colorimetry — Part 1: Standard Colorimetric Observers）

ISO/CIE 11664-3:2019 色度 第3部分：CIE三色值（Colorimetry - Part 3: CIE Tristimulus Values）

ITU-R BT.470 传统模拟电视系统（Conventional analogue television systems）

ITU-R BT.709 用于高清晰度电视节目制作和国际节目交换的参数值（Parameter values for the HDTV standards for production and international programme exchange）

SMPTE 170M 电视 用于演播室应用的复合模拟视频信号NTSC（Television - Composite Analog Video Signal - NTSC for Studio Applications）

SMPTE 240M 电视 1125行高清制作系统信号参数（Television - 1125-Line High-Definition Production Systems - Signal Parameters）

3 术语和定义

下列术语和定义适用于本文件。

3.1

B图像 B picture

帧间预测中可使用知识图像、显示顺序上过去的和将来的图像作为参考进行解码的图像。

3.2

变换系数 transform coefficient

变换域上的一个标量。

3.3

最大编码单元 largest coding unit; LCU

由图像划分得到一个L×L的亮度样值块和对应的色度样值块。

3.4

编码单元 coding unit

由亮度编码块和色度编码块组成的单元，可包含一个亮度编码块和对应的两个色度编码块，也可只包含一个亮度编码块，或只包含色度编码块。

3.5

编码树 coding tree

规定最大编码单元划分为一个或多个编码单元的方式。

3.6

编码块 coding block

一个M×M的样值块。

注：最大编码块由图像的三个样值矩阵（亮度和两个色度）中的一个矩阵划分得到一个K×K的样值块。

3.7

变换块 transform block

包含 M×N 的系数块。

3.8

补偿后样本 compensated sample

经预测补偿得到的样本。

3.9

不完全匹配串 incompletely matched sample string

至少包含一个未匹配样本的串。

3.10

残差 residual

样本或数据元素的重建值与其预测值之差。

3.11

参考索引 reference index

参考图像在参考图像队列中的编号。

注：在参考图像队列0中的编号称为L0参考索引，在参考图像队列1中的编号称为L1参考索引。

3.12

参考图像 reference picture

解码过程中用于后续图像帧间预测的图像。

3.13

参考图像队列 reference picture list

当前图像的参考图像所组成的图像队列。

注：参考图像队列包括参考图像队列0和参考图像队列1。

3.14

参考图像队列0 reference picture list 0

用于P图像或B图像解码的参考图像队列。

注：参考图像队列0包括显示顺序上过去的和将来的图像。

3.15

参考图像队列1 reference picture list 1

用于B图像解码的参考图像队列。

注：参考图像队列1包括显示顺序上过去的和将来的图像。

3.16

参考图像缓冲区 reference picture buffer

保存解码图像并用于预测的缓冲区。

3.17

层 layer

位流中的分级结构。

注：高层包含低层，编码层由高到低依次为：序列、图像、片、最大编码单元、编码单元和编码块。

3.18

场 field

一帧由相间的行构成两场，由Y，Cb，Cr的三个样本矩阵构成。

3.19

常现位置 frequently occurring position

图像内的一个样本的位置。这个位置的样本的值在同一图像内经常重复出现，可当作其他样本的预测样本。

3.20

重建样本 reconstructed sample

由解码器根据位流解码得到并构成解码图像的样本。

3.21

串复制帧内预测 intra string copy prediction

在同一解码图像中使用已解码的样本复制至当前样本区域，作为当前样本预测值的模式。

注：复制样本的区域的二维形状和样本数目与当前样本区域的二维形状和样本数量相同。

3.22

串矢量 string vector

用于串复制帧内预测模式的二维矢量。

注：串矢量值为当前串和参考串之间的坐标偏移量，其中，当前串与参考串均在当前图像中。

3.23

单位基矢量串 unit basis vector string

串矢量等于（0,-1）的串。

3.24

等值串 equal value string

所有样本的值都相等的串。

3.25

点矢量 point vector

表示常现位置的二维坐标的矢量。

3.26

点预测 point prediction

在串复制帧内预测中，一个串的所有参考样本是同一个常现位置上的样本，该串上所有像素的预测样本都是该常现位置的样本的值。

3.27

点预测信息表 point prediction information list

用于存放一个使用非普通串子模式的编码单元中所使用的点矢量。

3.28

二元符号 bin

组成二元符号串的符号，包括‘0’和‘1’。

3.29

二元符号串 bin string

有限位二元符号组成的有序序列，最左边符号是最高有效位，最右边符号是最低有效位。

3.30

反变换 inverse transform

将变换系数矩阵转换成空域样值矩阵的过程。

3.31

反量化 dequantization

对量化系数缩放后得到变换系数的过程。

3.32

仿射控制点运动矢量组 affine control point motion vectors

用于计算仿射预测子块运动矢量的参考运动矢量，包含两个或三个运动矢量。

3.33

仿射样本改善信息 affine sample refinement information

用于改善仿射预测单元预测样本的信息，包括仿射样本改善标志、仿射样本水平改善标志、仿射样本垂直改善标志、第一偏差矩阵、第二偏差矩阵、第三偏差矩阵和第四偏差矩阵。

3.34

仿射预测单元 affine prediction unit

由多个子块组成，每个子块由一个4×4或8×8的亮度预测块和对应的色度预测块组成。各子块的运动矢量可不同。

3.35

仿射运动信息 affine motion information

用于仿射预测单元的帧间预测的六元组，由仿射类型（四参数或六参数）、参考预测模式、L0运动矢量集合、L1运动矢量集合、L0参考索引和L1参考索引构成。

3.36

非普通串子模式 non-ordinary string submode

使用等值串、单位基矢量串和未匹配样本串进行预测的模式。

3.37

分量 component

图像的三个样值矩阵（亮度和两个色度）中的一个矩阵或矩阵中的单个样值。

3.38

光栅扫描 raster scan

将二维矩形光栅映射到一维光栅，一维光栅的入口从二维光栅的第一行开始，从左到右依次扫描光栅中的行。

3.39

划分 partition

将一个集合分为子集的过程。

注：集合中的每个元素属于且只属于某一个子集。

3.40

划分方式 partition type

划分获得的子集的组织方式。

3.41

I图像 I picture

只使用帧内预测解码的图像。

3.42

级 level

在某一类下对语法元素和语法元素参数值的限定集合。

3.43

解码顺序 decoding order

解码过程根据图像之间的预测关系，对每幅图像解码的顺序。

3.44

解码图像 decoded picture

解码器根据位流重建的图像。

3.45

解码图像缓冲区 decoded picture buffer

保存解码图像并用于输出重排序和输出定时的缓冲区。

注：解码图像缓冲区包括参考图像缓冲区。

3.46

解析过程 parse

由位流获得语法元素的过程。

3.47

块 block

一个M×N（M列N行）的样值矩阵或者变换系数矩阵。

3.48

块复制帧内预测 intra block copy prediction

在同一解码图像中使用已解码的样值复制至当前样本区域生成当前样本预测值的过程。

注：复制样值的区域与当前样本区域大小相同。

3.49

块矢量 block vector

用于块复制帧内预测模式的二维矢量。

注：块矢量为当前块和参考块之间的坐标偏移量，其中，当前块与参考块均在当前图像中。

3.50

块扫描 block scan

量化系数的特定串行排序方式。

3.51

类 profile

语法、语义及算法的子集。

3.52

历史点预测信息表 historical point prediction information list

用于非普通串子模式的由预测单元的点预测信息构成的表。

3.53

历史运动信息表 historical motion information list

用于帧间预测的由预测单元的运动信息构成的表。

3.54

历史帧内复制信息表 historical intra copy information list

用于块复制帧内预测或串复制帧内预测的由帧内复制信息构成的表。

3.55

亮度 luma

表示图像的明暗程度。

注：符号为Y，表示亮度信号的样值矩阵或单个样值。

3.56

量化参数 quantization parameter

在解码过程对量化系数进行反量化的参数。

3.57

量化系数 quantization coefficient

反量化前变换系数的值。

3.58

六参数仿射预测单元 affine prediction unit with six parameters

具有六个参数的仿射预测单元，其中仿射控制点运动矢量组中的运动矢量个数为三个。

3.59

滤波后样本 filtered sample

经去块效应滤波得到的样本。

3.60

P图像 P picture

帧间预测中使用知识图像和显示顺序上过去的图像作为参考进行解码的图像。

3.61

匹配串 ordinary string

使用任意合法串矢量进行预测的串。

3.62

偏移后样本 offseted sample

经样值偏移自适应补偿得到的样本。

3.63

片 patch

按光栅扫描顺序排列的若干相邻的最大编码单元。

3.64

普通串子模式 ordinary string submode

使用匹配串和不完全匹配串进行预测的模式。

3.65

起始码 start code

长度为32位、形式在整个位流中是唯一的二进制码。

注：起始码有多种用途，其中之一是用来标识位流语法结构的开始。

3.66

RL图像　Reference Layer picture

使用知识图像作为参考进行帧间预测解码的P图像或B图像。

3.67

色度　chroma

用于表示亮度和红色、蓝色的差值。

注：色度符号为Cr和Cb，表示两种色差信号的样值矩阵或单个样值。

3.68

四参数仿射预测单元　affine prediction unit with four parameters

具有四个参数的仿射预测单元，其中仿射控制点运动矢量组中的运动矢量个数为两个。

3.69

随机访问　random access

从某一点而非位流起始点开始对位流解码并恢复出解码图像。

3.70

填充位　stuffing bits

编码时插入位流中的位串，在解码时被丢弃。

3.71

图像重排序　picture reordering

解码顺序和输出顺序不同时对解码图像进行重新排序的过程。

注：输出顺序为输出解码图像的顺序，与显示顺序相同。

3.72

往返扫描　traverse scan

将二维矩形映射到一维串，从二维矩形的起始行开始逐行扫描整个二维矩形的过程。

注：起始行的行扫描方向是从左到右，起始行的相邻行的行扫描方向是从右到左，任意两个相邻行具有相反的行扫描方向。

3.73

位串　bit string

有限个二进制位的有序序列。

注：位串最左边位是最高有效位（MSB），最右边位是最低有效位（LSB）。

3.74

位流　bitstream

编码图像所形成的二进制数据流。

3.75

位流缓冲区　bitstream buffer

存储位流的缓冲区。

3.76

位流顺序　bitstream order

编码图像在位流中的排列顺序，与图像解码的顺序相同。

3.77

未匹配样本　unmatched sample

在串复制帧内预测中，没有参考样本的样本。

3.78

未匹配样本串 unmatched sample string

只包含未匹配样本的串。

3.79

显示顺序 display order

显示解码图像的顺序。

注：不应被输出的图像是不具有显示顺序的图像。

3.80

样本 sample

构成图像的基本元素。

3.81

样本宽高比 width height ratio

一幅图像中亮度样本列间的水平距离与行间的垂直距离之比，表示为$h \div v$。

注：h为水平方向样本个数，v为垂直方向样本个数。

3.82

样值 sample value

样本的幅值。

3.83

游程 run

在解码过程中若干连续的相同数据元素个数。在块扫描中一个非0系数前（沿块扫描顺序）值为0的系数的个数。

3.84

预测补偿 prediction compensation

由语法元素解码得到的样本残差与其对应的预测值之和。

3.85

预测单元 prediction unit

由编码单元划分得到的一个或多个预测块。

注：预测单元可包含一个亮度预测块和对应的色度预测块，也可只包含一个亮度预测块，或只包括色度预测块。

3.86

预测过程 prediction process

使用预测器对当前解码样值或者数据元素进行估计的过程。

3.87

预测划分方式 prediction partition type

将编码单元分为帧内预测块或帧间预测单元的划分方式。

3.88

预测块 prediction block

由编码单元划分得到的使用相同预测过程的M×N的样值块。

3.89

预测值 prediction value

在样值或数据元素的解码过程中，用到的先前已解码的样值或数据元素的组合。

3.90

语法元素 syntax element

位流中的数据单元解析后的结果。

3.91

运动矢量 motion vector

用于帧间预测的二维矢量，由当前图像指向参考图像，其值为当前块和参考块之间的坐标偏移量。

注：指向参考图像队列0和参考图像队列1中的参考图像的运动矢量分别称为L0运动矢量和L1运动矢量。

3.92

运动信息 motion information

用于帧间预测的五元组，由预测参考模式、L0运动矢量、L1运动矢量、L0参考索引和L1参考索引构成。

3.93

帧 frame

视频信号空间信息的表示，由一个亮度样本矩阵（Y）和两个色度样本矩阵（Cb和Cr）构成。

3.94

帧间编码 inter coding

使用帧间预测对编码单元或图像进行编码。

3.95

帧间预测 inter prediction

使用先前解码图像生成当前图像样本预测值的过程。

3.96

帧内编码 intra coding

使用帧内预测对编码单元或图像进行编码。

注：使用普通帧内预测的编码单元称为普通帧内编码单元；使用块复制帧内预测的编码单元称为块复制帧内编码单元；使用串复制帧内预测的编码单元称为串复制帧内编码单元。

3.97

帧内复制信息 intra copy information

用于块复制帧内预测或串复制帧内预测的信息，包括位移矢量、位置、尺寸信息和重复次数。

3.98

帧内模式频数表 intra mode frequency list

用于帧内预测，由预测单元的帧内预测模式的频数构成的表。

3.99

帧内预测 intra prediction

在相同解码图像中使用先前解码的样值生成当前样本预测值的过程。

注：帧内预测分为普通帧内预测、块复制帧内预测和串复制帧内预测。

3.100

知识图像 library picture

解码当前位流时使用的非当前位流的参考图像。

3.101

知识位流 library stream

包含知识图像的位流。

3.102

主位流 sequence stream

由该位流以外的信息提供的知识图像进行解码的位流。

3.103

字节对齐 byte alignment

从位流的第一个二进制位开始，某二进制位的位置是8的整数倍。

4 缩略语

下列缩略语适用于本文件。

ALF 自适应修正滤波（Adaptive Leveling Filter）
AMVR 自适应运动矢量精度（Adaptive Motion Vector Resolution）
ASR 仿射预测样本改善（Affine Sample Refinement）
AWP 角度加权预测模式（Angular Weighted Prediction）
BBS 位流缓冲区最小尺寸（Bitstream Buffer Minimum Size）
BBV 位流参考缓冲区管理（Bitstream Buffer Verifier）
BGC 双向梯度修正（Bi-directional Gradient Correction）
BIO 双向光流（BI-directional Optical flow）
CB 编码块（Coding Block）
CBR 恒定比特率（Constant Bit Rate）
CU 编码单元（Coding Unit）
DAR 显示宽高比（Display Aspect Ratio）
DMVR 解码端运动矢量改良（Decoder-side Motion Vector Refinement）
DOI 解码顺序索引（Decode Order Index）
DPB 解码图像缓冲区（Decoded Picture Buffer）
EALF 增强自适应修正滤波（Enhanced Adaptive Leveling Filter）
EMVR 扩展运动矢量精度（Extended Motion Vector Resolution）
ETMVP 增强时域运动矢量预测（Enhanced Temporal Motion Vector Prediction）
FIMC 基于频数信息的帧内编码（Frequency-based Intra Mode Coding）
HLG 混合对数伽马（Hybrid Log-Gamma）
HMVP 基于历史信息的运动矢量预测（History-based Motion Vector Prediction）
IBC 帧内块复制（Intra Block Copy）
IIP 改进型帧内预测（Improved Intra Prediction）
ISC 帧内串复制（Intra String Copy）
IST 隐择变换（Implicit Selected Transform）
MSB 最高有效位（Most Significant Bit）
NTSC 国家电视标准委员会（National Televison System Committee）
OBMC 重叠块运动补偿（Overlapped Block Motion Compensation）
OS 普通串（Ordinary String）
PAL 逐行倒相（Phase Alternative Line）
PB 预测块（Prediction Block）
PBT 基于位置的变换（Position Based Transform）
PCM 脉冲编码模式（Pulse Coding Mode）
PMC 跨多分量预测（Prediction from Multiple Cross-component）
POI 图像顺序索引（Picture Order Index）
PQ 感知量化（Perception Quantization）

ROI 感兴趣区域（Region Of Interesting）
SAR 样本宽高比（Sample Aspect Ratio）
SAWP 空域角度加权预测（Spatial Angular Weighted Prediction）
SBT 子块变换（Sub-Block Transform）
SECAM 按顺序传送彩色与存储（Séquentiel couleur à mémoire）
SMVD 对称运动矢量差（Symmetric Motion Vector Difference）
ST 二次变换（Secondary Transform）
TSCPM 跨分量两步预测模式（Two-Step Cross-component Prediction Mode）
UMVE 高级运动矢量表达（Ultimate Motion Vector Expression）

5 约定

5.1 概述

本文件中使用的数学运算符和优先级参照C语言。但对整型除法和算术移位操作进行了特定定义。除特别说明外，约定编号和计数从0开始。

5.2 算术运算符

算术运算符定义见表1。

表1 算术运算符定义

算术运算符	定义
+	加法运算
-	减法运算（二元运算符）或取反（一元前缀运算符）
*	乘法运算
a^b	幂运算，表示a的b次幂。也可表示上标
/	整除运算，沿向0的取值方向截断。例如，7/4和-7/-4截断至1，-7/4和7/-4截断至-1
÷	除法运算，不做截断或四舍五入
$\frac{a}{b}$	除法运算，不做截断或四舍五入
$\sum_{i=a}^{b} f(i)$	自变量i取由a到b（含b）的所有整数值时，函数 $f(i)$ 的累加和
$a \ \% \ b$	模运算，a除以b的余数，其中a与b都是正整数
$\lceil . \rceil$	上取整
$\lfloor \cdot \rfloor$	下取整
$a \sim b$	a到b（含a和b）的所有整数值

5.3 逻辑运算符

逻辑运算符定义见表2。

表2　逻辑运算符定义

逻辑运算符	定义
a && *b*	*a*和*b*之间的与逻辑运算
a || *b*	*a*和*b*之间的或逻辑运算
!	逻辑非运算
express ? *a* : *b*	如果表达式*express*的结果为真或不为0，则使用*a*进行赋值；否则，使用*b*进行赋值

5.4　关系运算符

关系运算符定义见表3。

表3　关系运算符定义

关系运算符	定义
>	大于
>=	大于或等于
<	小于
<=	小于或等于
==	等于
!=	不等于

5.5　位运算符

位运算符定义见表4。

表4　位运算符定义

位运算符	定义
&	与运算
|	或运算
~	取反运算
a >> *b*	将*a*以2的补码整数表示的形式向右移*b*位。仅当*b*取正数时定义此运算
a << *b*	将*a*以2的补码整数表示的形式向左移*b*位。仅当*b*取正数时定义此运算

5.6　赋值

赋值运算定义见表5。

表5　赋值运算定义

赋值运算	定义
=	赋值运算符
++	递增，*x*++相当于*x* = *x* + 1。当用于数组下标时，在自加运算前先求变量值
--	递减，*x*--相当于*x* = *x* - 1。当用于数组下标时，在自减运算前先求变量值
+=	自加指定值，例如*x* += 3相当于*x* = *x* + 3，*x* += (-3)相当于*x* = *x* + (-3)
-=	自减指定值，例如*x* -= 3相当于*x* = *x* - 3，*x* -= (-3)相当于*x* = *x* - (-3)

5.7 数学函数

数学函数定义见公式（1）～公式（12）。

$$\mathrm{Abs}(x)=\begin{cases}x & ; \quad x>=0\\ -x & ; \quad x<0\end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

式中：

x——自变量x。

$$\mathrm{Ceil}(x)=\lceil x \rceil \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

式中：

x——自变量x。

$$\mathrm{Floor}(x)=\lfloor x \rfloor \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3)$$

式中：

x——自变量x。

$$\mathrm{Clip1}(x)=\mathrm{Clip3}(0,2^{BitDepth}-1,x) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4)$$

式中：

x ——自变量x；

$BitDepth$ ——编码样本精度。

$$\mathrm{Clip3}(i,j,x)=\begin{cases}i & ; \quad x<i\\ j & ; \quad x>j\\ x & ; \quad \text{其他}\end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (5)$$

式中：

x——自变量x；

i——下界；

j——上界。

$$\mathrm{Median}(x,y,z)=x+y+z-\mathrm{Min}(x,\mathrm{Min}(y,z))-\mathrm{Max}(x,\mathrm{Max}(y,z)) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (6)$$

式中：

x——自变量x；

y——自变量y；

z——自变量z。

$$\mathrm{Min}(x,y)=\begin{cases}x & ; \quad x<=y\\ y & ; \quad x>y\end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (7)$$

式中：

x——自变量x；

y——自变量y。

$$\mathrm{Max}(x,y)=\begin{cases}x & ; \quad x>=y\\ y & ; \quad x<y\end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (8)$$

式中：

x——自变量x；

y——自变量y。

$$\mathrm{Sign}(x)=\begin{cases}1 & ; \quad x>=0\\ -1 & ; \quad x<0\end{cases} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (9)$$

式中：

x——自变量x。

$$\mathrm{Log}(x)=\log_2 x \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (10)$$

式中：

x——自变量x。

$$\mathrm{Rounding}(x,s)=\begin{cases}\mathrm{Sign}(x)\times((\mathrm{Abs}(x)+(1<<(s-1)))>>s) & s\geq 1\\ x & s=0\end{cases} \quad \cdots\cdots\cdots\cdots\cdots \quad (11)$$

式中：

x——自变量x；

s——自变量s。

$$\mathrm{Exchange}(x,y)=(y,x) \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (12)$$

式中：

x——自变量x；

y——自变量y。

5.8 结构关系符

结构关系符定义见表6。

表6 结构关系符

结构关系符	定义
->	例如：*a* ->*b*表示*a*是一个结构，*b*是*a*的一个成员变量

5.9 位流语法、解析过程和解码过程的描述方法

5.9.1 描述方法

位流语法描述方法类似C语言。位流的语法元素使用粗体字表示，每个语法元素通过名字（用下划线分割的英文字母组，所有字母都是小写）、语法和语义来描述。语法表和正文中语法元素的值用常规字体表示。

某些情况下，可在语法表中应用从语法元素导出的其他变量值，这样的变量在语法表或正文中用不带下划线的小写字母和大写字母混合命名。大写字母开头的变量用于解码当前以及相关的语法结构，也可用于解码后续的语法结构。小写字母开头的变量只在它们所在的章条内使用。

语法元素值的助记符和变量值的助记符与它们的值之间的关系在正文中说明。在某些情况下，二者等同使用。助记符由一个或多个使用下划线分隔的字母组表示，每个字母组以大写字母开始，也可包括多个大写字母。

位串的长度是4的整数倍时，可使用十六进制符号表示。十六进制的前缀是“0x”，例如“0x1a”

表示位串“0001 1010”。

条件语句中0表示FALSE，非0表示TRUE。

语法表描述了所有符合本文件的位流语法的超集，附加的语法限制在相关条中说明。

表7给出了描述语法的伪代码例子。当语法元素出现时，表示从位流中读一个数据单元。

表7　语法描述的伪代码

伪代码	描述符
/*语句是一个语法元素的描述符，或者说明语法元素的存在、类型和数值，下面给出两个例子。*/	
syntax_element	ue(v)
conditioning statement	
/*花括号括起来的语句组是复合语句，在功能上视作单个语句。*/	
{	
statement	
…	
}	
/*“while”语句测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。*/	
while (condition)	
statement	
/*“do … while”语句先执行循环体一次，然后测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。*/	
do	
statement	
while (condition)	
/*“if … else”语句首先测试condition，如果为TRUE，则执行primary语句，否则执行alternative语句。如果alternative语句不需要执行，结构的“else”部分和相关的alternative语句可忽略。*/	
if (condition)	
primary statement	
else	
alternative statement	
/*“for”语句首先执行initial语句，然后测试condition，如果condition为TRUE，则重复执行primary语句和subsequent语句直到condition不为TRUE。*/	
for (initial statement; condition; subsequent statement)	
primary statement	
/*“break”语句用于do-while、while和for循环体中，可使当前循环体立即终止循环。*/	
break	

解析过程和解码过程用文字和类似C语言的伪代码描述。

5.9.2 函数

5.9.2.1 概述

以下函数用于语法描述。假定解码器中存在一个位流指针，这个指针指向位流中要读取的下一个二进制位的位置。函数由函数名及左右圆括号内的参数构成。函数也可没有参数。

5.9.2.2 byte_aligned( )

如果位流的当前位置是字节对齐的，返回TRUE，否则返回FALSE。

5.9.2.3 next_bits(n)

返回位流的随后*n*个二进制位，MSB在前，不改变位流指针。如果剩余的二进制位少于*n*，则返回0。

5.9.2.4 byte_aligned_next_bits(n)

如果位流当前位置不是字节对齐的,返回位流当前位置的下一个字节开始的*n*个二进制位,MSB在前，不改变位流指针；如果位流当前位置是字节对齐的，返回位流随后的*n*个二进制位，MSB在前，不改变位流指针。如果剩余的二进制位少于*n*，则返回0。

5.9.2.5 next_start_code( )

在位流中寻找下一个起始码，将位流指针指向起始码前缀的第一个二进制位。函数定义应符合表8的规定。

表8 next_start_code 函数的定义

函数定义	描述符
next_start_code( ) {	
stuffing_bit	‘1’
while (! byte_aligned( ))	
stuffing_bit	‘0’
while (next_bits(24) != ’0000 0000 0000 0000 0000 0001’)	
stuffing_byte	‘00000000’
}	

stuffing_byte应出现图像头之后和第一个片起始码之前。

5.9.2.6 is_end_of_patch( )

在位流中检测是否已达到片的结尾，如果已到达片的结尾，返回TRUE，否则返回FALSE。此函数不修改位流指针。函数定义应符合表9的规定。

表9 is_end_of_patch 函数的定义

函数定义	描述符
is_end_of_patch( ) {	
if (byte_aligned( )) {	

表9（续）

函数定义	描述符
if (next_bits(32) == 0x80000001)	
return TRUE /* 片结束 */	
}	
else {	
if ((byte_aligned_next_bits(24) == 0x000001) && is_stuffing_pattern( ))	
return TRUE /* 片结束 */	
}	
return FALSE;	
}	

5.9.2.7 is_stuffing_pattern( )

在位流中检测当前字节中剩下的位或在字节对齐时下一个字节是否是片结尾填充的二进制位，如果是，则返回TRUE，否则返回FALSE。此函数不修改位流指针。函数定义应符合表10的规定。

表10 is_stuffing_pattern 函数的定义

函数定义	描述符
is_stuffing_pattern ( ) {	
if (next_bits(8-n) == (1<<(7-n))) /* n=0～7，为位流指针在当前字节的位置偏移，n为0时位流指针指向当前字节最高位 */	
return TRUE	
else	
return FALSE;	
}	

5.9.2.8 read_bits(n)

返回位流的随后n个二进制位，MSB在前，同时位流指针前移n个二进制位。如果n等于0，则返回0，位流指针不前移。

函数也用于解析过程和解码过程的描述。

5.9.3 描述符

描述符表示不同语法元素的解析过程，应符合表11的规定。

表11 描述符

描述符	说明
ae(v)	高级熵编码的语法元素。解析过程在8.3中定义
b(8)	一个任意取值的字节。解析过程由函数read_bits(8)的返回值规定
f(n)	取特定值的连续n个二进制位。解析过程由函数read_bits(n)的返回值规定

表 11（续）

描述符	说明
i(n)	n位整数。在语法表中，如果n是“v”，其位数由其他语法元素值确定。解析过程由函数read_bits(n)的返回值规定，该返回值用高位在前的2的补码表示
r(n)	连续n个‘0’。解析过程由函数read_bits(n)的返回值规定
se(v)	有符号整数语法元素，用指数哥伦布码编码。解析过程在8.2中定义
u(n)	n位无符号整数。在语法表中，如果n是“v”，其位数由其他语法元素值确定。解析过程由函数read_bits(n)的返回值规定，该返回值用高位在前的二进制表示
ue(v)	无符号整数语法元素，用指数哥伦布码编码。解析过程在8.2中定义

5.9.4 保留、禁止和标记位

本文件定义的位流语法中，某些语法元素的值被标注为“保留”(reserved)或“禁止”(forbidden)。

“保留”定义了一些特定语法元素值用于将来对本文件的扩展。这些值不应出现在符合本文件的位流中。

“禁止”定义了一些特定语法元素值，这些值不应出现在符合本文件的位流中。

注：禁止某些值的目的通常是为了避免在位流中出现伪起始码。

“标记位”（marker_bit）指该位的值应为‘1’。

位流中的“保留位”（reserved_bits）表明保留了一些语法单元用于将来对本文件的扩展，解码处理应忽略这些位。“保留位”不应出现从任意字节对齐位置开始的21个以上连续的‘0’。

6 编码位流的结构

6.1 视频序列

6.1.1 通则

视频序列是位流的最高层语法结构。视频序列由第一个序列头开始，序列结束码或视频编辑码表明了一个视频序列的结束。视频序列的第一个序列头到第一个出现的序列结束码或视频编辑码之间的序列头为重复序列头。每个序列头后面跟着一个或多个编码图像，每幅图像之前应有图像头。编码图像在位流中按位流顺序排列，位流顺序应与解码顺序相同。解码顺序可与显示顺序不相同。

6.1.2 逐行和隔行视频序列

本文件支持两种序列：逐行视频序列和隔行视频序列。

帧由三个样本矩阵构成，包括一个亮度样本矩阵（Y）和两个色度样本矩阵（Cb和Cr）。样本矩阵元素的值为整数。Y、Cb和Cr三个分量与原始的（模拟）红、绿和蓝色信号之间的关系，包括原始信号的色度和转移特性等可在位流中定义，这些信息不影响解码过程。

场由构成帧的三个样本矩阵中相间的行构成，即帧样本矩阵的第一行、第三行、第五行等奇数行构成一个场，称为顶场；第二行、第四行、第六行等偶数行构成另一个场，称为底场。

解码器的输出是一系列图像。两帧之间存在着一个帧时间间隔。对隔行视频序列而言，每帧的两场之间存在着一个场时间间隔。对逐行视频序列而言，每帧的两场之间时间间隔为0。

6.1.3 序列头

视频序列头由视频序列起始码开始，后面跟着一串编码图像数据。

使用重复序列头的主要目的是支持对视频序列的随机访问。

序列头后的第一幅解码图像应是I图像或RL图像。

当前图像对应的序列头为解码顺序在当前图像之前的最近的序列头。

如果当前图像对应的序列头后的第一幅解码图像为I图像，且当前图像的显示顺序在该I图像之后，则当前图像的参考图像应在该I图像和显示顺序在该I图像之后的图像范围内。

如果当前图像对应的序列头后的第一幅解码图像为RL图像，并且当前图像的显示顺序在该RL图像之后，则当前图像的参考图像应在该RL图像、显示顺序在该RL图像之后的图像和该RL图像所参考的知识图像范围内。

在对位流进行编辑或随机访问的情况下，重复序列头之前的全部数据可被丢弃，这样得到的一个新的位流仍应符合本文件。

6.2 图像

6.2.1 通则

一幅图像可是一帧或一场，其编码数据由图像起始码开始，到序列起始码、序列结束码或下一个图像起始码结束。

在位流中，隔行扫描图像的两场的编码数据可依次出现，也可交融出现。两场数据的解码和显示顺序在图像头中规定。

图像的解码处理包括解析过程和解码过程。

6.2.2 图像格式

6.2.2.1 4:0:0 格式

对于4:0:0格式，只包括Y矩阵。

亮度样本位置示例见图1。

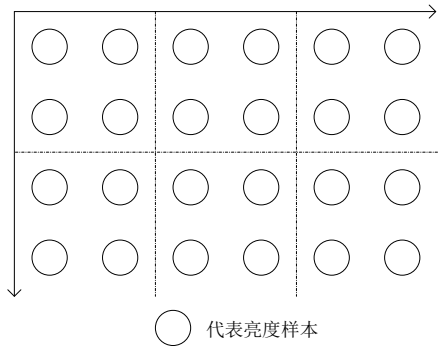


图1 4:0:0 格式下亮度样本位置示例

6.2.2.2 4:2:0 格式

对于4:2:0格式，Cb和Cr矩阵水平和垂直方向的尺寸都只有Y矩阵的一半。Y矩阵的行数和每行样本数都应是偶数。如果图像两场的编码数据依次出现，则Y矩阵的行数还应能被4整除。

亮度和色度样本位置示例见图2。

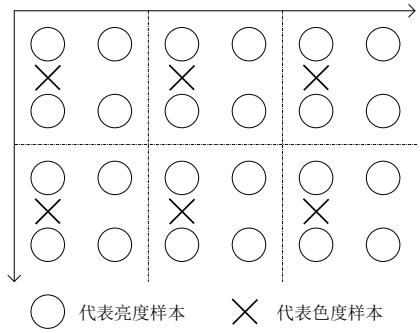


图2　4:2:0 格式下亮度和色度样本位置示例

6.2.2.3　4:2:2 格式

对于4:2:2格式，Cb和Cr矩阵在水平方向的尺寸只有Y矩阵的一半，在垂直方向的尺寸和Y相同。Y矩阵的每行样本数应是偶数。如果图像两场的编码数据依次出现，则Y矩阵的行数也应是偶数。

亮度和色度样本位置示例见图3。

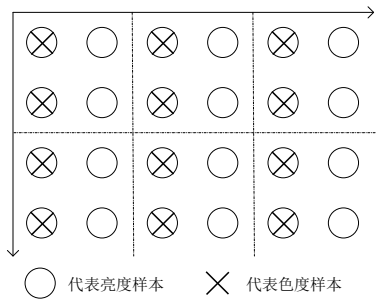


图3　4:2:2 格式下亮度和色度样本位置示例

6.2.2.4　4:4:4 格式

对于4:4:4格式，Cb和Cr矩阵在水平和垂直方向的尺寸都和Y矩阵一样。如果图像两场的编码数据依次出现，则Y矩阵的行数应是偶数。

亮度和色度样本位置示例见图4。

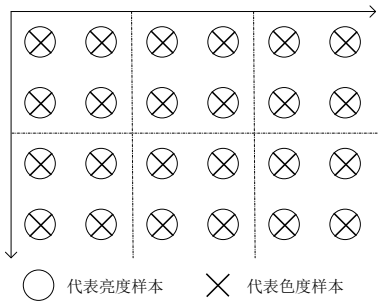


图4　4:4:4 格式下亮度和色度样本位置示例

6.2.3　图像类型

本文件定义了四种解码图像：

a) I 图像；

b) P 图像；

c) B 图像；

d) RL 图像。

6.2.4 图像间的顺序

如果视频序列中没有B图像，则解码顺序与显示顺序相同。如果视频序列中包含B图像，则解码顺序与显示顺序不同，解码图像输出显示前应进行图像重排序。

序列头后的第一幅解码图像应是I图像或RL图像。码流中显示顺序在该I图像或RL图像之后的图像的解码顺序应在该I图像或RL图像之后。

示例1：I/RL 图像和 P 图像之间有 3 幅 B 图像，两幅连续的 P 图像之间也有 3 幅 B 图像。按照解码顺序用图像 0I/RL 预测图像 4P，用图像 4P 和 0I/RL 预测图像 2B，用图像 2B 和 0I/RL 预测图像 1B，用图像 4P 和 2B 预测图像 3B。解码顺序是 0I/RL、4P、2B、1B、3B；显示顺序是 0I/RL、1B、2B、3B、4P。

按照解码顺序排序：

解码顺序	0	1	2	3	4	5	6	7	8	9	10	11	12
类型	I/RL	P	B	B	B	P	B	B	B	P	B	B	B
显示顺序	0	4	2	1	3	8	6	5	7	12	10	9	11

按照显示顺序排序：

显示顺序	0	1	2	3	4	5	6	7	8	9	10	11	12
类型	I/RL	B	B	B	P	B	B	B	P	B	B	B	P
解码顺序	0	3	2	4	1	7	6	8	5	11	10	12	9

示例2：I/RL 图像和 P 图像之间有 7 幅 B 图像，两幅连续的 P 图像之间也有 7 幅 B 图像。按照解码顺序用图像 0I/RL 预测图像 8P，用图像 8P 和 0I/RL 预测图像 4B，用图像 4B 和 0I/RL 预测图像 2B，用图像 2B 和 0I/RL 预测图像 1B，用图像 4B 和 2B 预测图像 3B，用图像 8P 和 4B 预测图像 6B，用图像 6B 和 4B 预测图像 5B，用图像 8P 和 6B 预测图像 7B。解码顺序是 0I/RL、8P、4B、2B、1B、3B、6B、5B、7B；显示顺序是 0I/RL、1B、2B、3B、4B、5B、6B、7B、8P。

按照解码顺序排序：

解码顺序	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
类型	I/RL	P	B	B	B	B	B	B	B	P	B	B	B	B	B	B	B
显示顺序	0	8	4	2	1	3	6	5	7	16	12	10	9	11	14	13	15

按照显示顺序排序：

显示顺序	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
类型	I/RL	B	B	B	B	B	B	B	P	B	B	B	B	B	B	B	P
解码顺序	0	4	3	5	2	7	6	8	1	12	11	13	10	15	14	16	9

示例3：I/RL 图像和 P 图像之间有 2 幅 B 图像，两幅连续的 P 图像之间也有 2 幅 B 图像。按照解码顺序用图像 0I/RL 预测图像 3P，用图像 3P 和 0I/RL 预测图像 1B 和 2B。解码顺序是 0I/RL、3P、1B、2B；显示顺序是 0I/RL、1B、2B、3P。

按照解码顺序排序：

解码顺序	0	1	2	3	4	5	6	7	8	9	10	11	12
类型	I/RL	P	B	B	P	B	B	I	B	B	P	B	B
显示顺序	0	3	1	2	6	4	5	9	7	8	12	10	11

按照显示顺序排序：

显示顺序	0	1	2	3	4	5	6	7	8	9	10	11	12
类型	I/RL	B	B	P	B	B	I	B	B	P	B	B	P

解码顺序 0 2 3 1 5 6 4 8 9 7 11 12 10

示例4：I/RL 图像后有 16 幅 B 图像

按照解码顺序排序：

解码顺序	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
类型	I/RL	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
显示顺序	0	16	8	4	2	1	3	6	5	7	12	10	9	11	14	13	15

按照显示顺序排序：

显示顺序	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
类型	I/RL	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
解码顺序	0	5	4	6	3	8	7	9	2	12	11	13	10	15	14	16	1

6.2.5 参考图像和参考图像队列

P图像可有知识图像和显示顺序上位于当前图像之前的（过去的）多幅参考图像，这些参考图像构成P图像的参考图像队列0。

B图像可有知识图像、多幅显示顺序位于当前图像之前的（过去的）参考图像和多幅显示顺序位于当前图像之后的（将来的）参考图像，这些参考图像构成B图像的参考图像队列0和参考图像队列1。

注：参考图像队列0和参考图像队列1包含的参考图像可能相同，解码时根据预测方向标识使用的参考图像队列类别及参考图像。

运动矢量所指的参考像素可超出参考图像的边界，在这种情况下对超出参考图像边界的整数样本应使用距离该整数参考样本所指位置最近的图像内的整数样本进行边界扩展。

场边界扩展方法和参考图像边界扩展方法相同。

6.3 片

片是图像中的矩形区域，包含若干最大编码单元在图像内的部分，片之间不应重叠。片结构示意图见图5，其中，“A”～“F”代表不同的片，黑色粗体实线为不同片的边界。

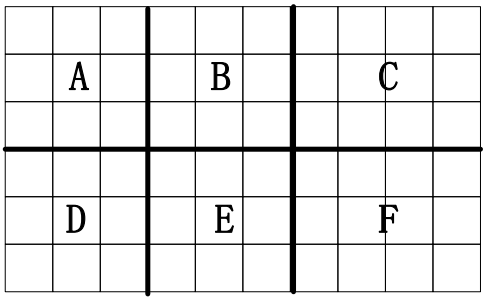


图5 片结构示意图

6.4 最大编码单元、编码树、编码单元和变换块

图像划分为最大编码单元，最大编码单元之间不应重叠，最大编码单元左上角的样本不应超出图像边界，最大编码单元右下角的样本可超出图像边界。

最大编码单元划分为一个或多个编码单元，由编码树决定。编码单元划分为一个或多个变换块。

7 位流的语法和语义

7.1 语法描述

7.1.1 起始码

起始码是一组特定的位串。在符合本文件的位流中，除起始码外的任何情况下都不应出现这些位串。

起始码由起始码前缀和起始码值构成。起始码前缀是位串‘0000 0000 0000 0000 0000 0001’。所有的起始码都应字节对齐。

起始码值是一个8位整数，用来表示起始码的类型，应符合表12的规定。

表12 起始码值

起始码类型	起始码值（十六进制）
片起始码（patch_start_code）	00～7F
保留	80～8E
片结束码（patch_end_code）	8F
保留	90～AF
视频序列起始码（video_sequence_start_code）	B0
视频序列结束码（video_sequence_end_code）	B1
用户数据起始码（user_data_start_code）	B2
帧内预测图像起始码（intra_picture_start_code）	B3
保留	B4
视频扩展起始码（extension_start_code）	B5
帧间预测图像起始码（inter_picture_start_code）	B6
视频编辑码（video_edit_code）	B7
保留	B8
系统起始码	B9～FF

部分语法元素取特定值时可得到与起始码前缀相同的位串，称为伪起始码。符合本文件的编码器和解码器应使用附录A定义的方法处理伪起始码问题。

7.1.2 视频序列语法

7.1.2.1 视频序列定义

视频序列定义应符合表13的规定。

表13 视频序列定义

视频序列定义	描述符
video_sequence( ) {	
do {	
sequence_header( )	
extension_and_user_data(0)	
do {	
if (next_bits(32) == intra_picture_start_code)	
intra_picture_header( )	

表 13（续）

视频序列定义	描述符
else	
inter_picture_header( )	
extension_and_user_data(1)	
picture_data( )	
} while ((next_bits(32) == inter_picture_start_code) || (next_bits(32) == intra_picture_start_code))	
} while ((next_bits(32) != video_sequence_end_code) && (next_bits(32) != video_edit_code))	
if (next_bits(32) == video_sequence_end_code)	
video_sequence_end_code	f(32)
if (next_bits(32) == video_edit_code)	
video_edit_code	f(32)
}	

7.1.2.2 序列头定义

序列头定义应符合表14的规定。

表14 序列头定义

序列头定义	描述符
sequence_header( ) {	
video_sequence_start_code	f(32)
profile_id	u(8)
level_id	u(8)
progressive_sequence	u(1)
field_coded_sequence	u(1)
library_stream_flag	u(1)
if (! LibraryStreamFlag) {	
library_picture_enable_flag	u(1)
if (LibraryPictureEnableFlag) {	
duplicate_sequence_header_flag	u(1)
}	
}	
marker_bit	f(1)
horizontal_size	u(14)
marker_bit	f(1)
vertical_size	u(14)
chroma_format	u(2)
sample_precision	u(3)
if (profile_id == 0x22 || profile_id == 0x32) {	
encoding_precision	u(3)

表 14（续）

序列头定义	描述符
}	
marker_bit	f(1)
aspect_ratio	u(4)
frame_rate_code	u(4)
marker_bit	f(1)
bit_rate_lower	u(18)
marker_bit	f(1)
bit_rate_upper	u(12)
low_delay	u(1)
temporal_id_enable_flag	u(1)
marker_bit	f(1)
bbv_buffer_size	u(18)
marker_bit	f(1)
max_dpb_size_minus1	u(4)
rpl1_index_exist_flag	u(1)
rpl1_same_as_rpl0_flag	u(1)
marker_bit	f(1)
num_ref_pic_list_set[0]	ue(v)
for (j = 0; j < NumRefPicListSet[0]; j++) {	
reference_picture_list_set(0, j)	
}	
if (! Rpl1SameAsRpl0Flag) {	
num_ref_pic_list_set[1]	ue(v)
for (j = 0; j < NumRefPicListSet[1]; j++) {	
reference_picture_list_set(1, j)	
}	
}	
num_ref_default_active_minus1[0]	ue(v)
num_ref_default_active_minus1[1]	ue(v)
log2_lcu_size_minus2	u(3)
log2_min_cu_size_minus2	u(2)
log2_max_part_ratio_minus2	u(2)
max_split_times_minus6	u(3)
log2_min_qt_size_minus2	u(3)
log2_max_bt_size_minus2	u(3)
log2_max_eqt_size_minus3	u(2)
marker_bit	f(1)
weight_quant_enable_flag	u(1)
if (WeightQuantEnableFlag)	

表 14（续）

序列头定义	描述符
{	
load_seq_weight_quant_data_flag	u(1)
if (load_seq_weight_quant_data_flag == '1') {	
weight_quant_matrix( )	
}	
}	
st_enable_flag	u(1)
sao_enable_flag	u(1)
alf_enable_flag	u(1)
affine_enable_flag	u(1)
smvd_enable_flag	u(1)
ipcm_enable_flag	u(1)
amvr_enable_flag	u(1)
num_of_hmvp_cand	u(4)
umve_enable_flag	u(1)
if ((NumOfHmvpCand != 0) && AmvrEnableFlag) {	
emvr_enable_flag	u(1)
}	
intra_pf_enable_flag	u(1)
tscpm_enable_flag	u(1)
marker_bit	f(1)
dt_enable_flag	u(1)
if (DtEnableFlag) {	
log2_max_dt_size_minus4	u(2)
}	
pbt_enable_flag	u(1)
EipmEnableFlag = 0	
MipfEnableFlag = 0	
IntraPfChromaEnableFlag = 0	
UmveEnhancementEnableFlag = 0	
AffineUmveEnableFlag = 0	
SbTmvpEnableFlag = 0	
SrccEnableFlag = 0	
EnhancedStEnableFlag = 0	
EnhancedTscpmEnableFlag = 0	
MaecEnableFlag = 0	
if (profile_id == 0x30 || profile_id == 0x32) {	
EipmEnableFlag = 1	
pmc_enable_flag	u(1)

表 14（续）

序列头定义	描述符
if (TscpmEnableFlag) {	
EnhancedTscpmEnableFlag = 1	
}	
iip_enable_flag	u(1)
sawp_enable_flag	u(1)
MipfEnableFlag = 1	
if (IntraPfEnableFlag) {	
IntraPfChromaEnableFlag = 1	
}	
if (UmveEnableFlag) {	
UmveEnhancementEnableFlag = 1	
}	
if (AffineEnableFlag) {	
AffineUmveEnableFlag = 1	
}	
if (AffineEnableFlag) {	
asr_enable_flag	u(1)
}	
awp_enable_flag	u(1)
SbTmvpEnableFlag = 1	
etmvp_mvap_enable_flag	u(1)
dmvr_enable_flag	u(1)
bio_enable_flag	u(1)
bgc_enable_flag	u(1)
inter_pf_enable_flag	u(1)
inter_pc_enable_flag	u(1)
obmc_enable_flag	u(1)
if (StEnableFlag) {	
EnhancedStEnableFlag = 1	
}	
sbt_enable_flag	u(1)
ist_enable_flag	u(1)
SrccEnableFlag = 1	
MaecEnableFlag = 1	
esao_enable_flag	u(1)
ccsao_enable_flag	u(1)
if (EsaoEnableFlag) {	
SaoEnableFlag = 0	
}	

表 14（续）

序列头定义	描述符
if (AlfEnableFlag) {	
ealf_enable_flag	u(1)
}	
ibc_enable_flag	u(1)
marker_bit	u(1)
isc_enable_flag	u(1)
if (IbcEnableFlag || IscEnableFlag) {	
num_of_intra_hmvp_cand	u(4)
}	
fimc_enable_flag	u(1)
nn_tools_set_hook	u(8)
if (NnFilterEnableFlag) {	
num_of_nn_filter_minus1	ue(v)
}	
marker_bit	u(1)
}	
if (low_delay == '0')	
output_reorder_delay	u(5)
cross_patch_loop_filter_enable_flag	u(1)
ref_colocated_patch_flag	u(1)
stable_patch_flag	u(1)
if (stable_patch_flag == '1') {	
uniform_patch_flag	u(1)
if (uniform_patch_flag == '1') {	
marker_bit	f(1)
patch_width_minus1	ue(v)
patch_height_minus1	ue(v)
}	
}	
reserved_bits	r(2)
next_start_code( )	
}	

7.1.2.3 参考图像队列配置集定义

参考图像队列配置集定义应符合表15的规定。

表15　参考图像队列配置集定义

参考图像队列配置集定义	描述符
reference_picture_list_set(list, rpls) {	
if (LibraryPictureEnableFlag) {	
reference_to_library_enable_flag	u(1)
}	
num_of_ref_pic[list][rpls]	ue(v)
for (i = 0; i < NumOfRefPic[list][rpls]; i++) {	
if (ReferenceToLibraryEnableFlag) {	
library_index_flag[list][rpls][i]	u(1)
}	
if (LibraryIndexFlag[list][rpls][i]) {	
referenced_library_picture_index[list][rpls][i]	ue(v)
}	
else {	
abs_delta_doi[list][rpls][i]	ue(v)
if (abs_delta_doi[list][rpls][i] > 0) {	
sign_delta_doi[list][rpls][i]	u(1)
}	
}	
}	
}	

7.1.2.4　加权量化数据定义

加权量化数据定义应符合表16的规定。

表16　加权量化数据定义

自定义加权量化矩阵定义	描述符
weight_quant_matrix( ) {	
for (sizeId = 0; sizeId < 2; sizeId++) {	
WQMSize = 1 << (sizeId+2)	
for (i=0; i<WQMSize; i++) {	
for (j=0; j<WQMSize; j++) {	
weight_quant_coeff	ue(v)
if (sizeId == 0)	
$WeightQuantMatrix_{4x4}$[i][j] = WeightQuantCoeff	
else	
$WeightQuantMatrix_{8x8}$[i][j] = WeightQuantCoeff	
}	
}	
}	
}	

7.1.2.5 扩展和用户数据定义

扩展和用户数据定义应符合表17的规定。

表17 扩展和用户数据定义

扩展和用户数据定义	描述符
extension_and_user_data(i) {	
while ((next_bits(32) == extension_start_code) || (next_bits(32) == user_data_start_code)) {	
if (next_bits(32) == extension_start_code)	
extension_data(i)	
if (next_bits(32) == user_data_start_code)	
user_data( )	
}	
}	

扩展数据定义应符合表18的规定。

表18 扩展数据定义

扩展数据定义	描述符
extension_data(i) {	
while (next_bits(32) == extension_start_code) {	
extension_start_code	f(32)
if (i == 0) { /* 序列头之后 */	
if (next_bits(4) == '0010') /* 序列显示扩展 */	
sequence_display_extension( )	
else if (next_bits(4) == '0011') /* 时域可伸缩扩展 */	
temporal_scalability_extension( )	
else if (next_bits(4) == '0100') /* 版权扩展 */	
copyright_extension()	
else if (next_bits(4) == '1010') /* 目标设备显示和内容元数据扩展 */	
mastering_display_and_content_metadata_extension( )	
else if (next_bits(4) == '1011') /* 摄像机参数扩展 */	
camera_parameters_extension( )	
else if (next_bits(4) == '1101') /* 参考知识图像扩展 */	
cross_random_access_point_reference_extension( )	
else	
while (next_bits(24) != '0000 0000 0000 0000 0000 0001') {	
reserved_extension_data_byte	u(8)
}	
}	
else { /* 图像头之后 */	
if (next_bits(4) == '0100') /* 版权扩展 */	

表 18（续）

扩展数据定义	描述符
copyright_extension( )	
else if (next_bits(4) == ’0101’) /* 高动态范围图像元数据扩展 */	
hdr_dynamic_metadata_extension( )	
else if (next_bits(4) == ’0111’) /* 图像显示扩展 */	
picture_display_extension( )	
else if (next_bits(4) == ’1011’) /* 摄像机参数扩展 */	
camera_parameters_extension( )	
else if (next_bits(4) == ’1100’) /* 感兴趣区域参数扩展 */	
roi_parameters_extension( )	
else if (next_bits(4) == ’1110’) /* 自适应帧内刷新参数扩展 */	
air_parameters_extension( )	
else {	
while (next_bits(24) != ’0000 0000 0000 0000 0000 0001’)	
reserved_extension_data_byte	u(8)
}	
}	
}	
}	

用户数据定义应符合表19的规定。

表19 用户数据定义

用户数据定义	描述符
user_data( ) {	
user_data_start_code	f(32)
while (next_bits(24) ! ’0000 0000 0000 0000 0000 0001’) {	
user_data	b(8)
}	
}	

7.1.2.6 序列显示扩展定义

序列显示扩展定义应符合表20的规定。

表20 序列显示扩展定义

序列显示扩展定义	描述符
sequence_display_extension( ) {	
extension_id	f(4)
video_format	u(3)

表 20（续）

序列显示扩展定义	描述符
sample_range	u(1)
colour_description	u(1)
if (colour_description) {	
colour_primaries	u(8)
transfer_characteristics	u(8)
matrix_coefficients	u(8)
}	
display_horizontal_size	u(14)
marker_bit	f(1)
display_vertical_size	u(14)
td_mode_flag	u(1)
if (td_mode_flag == '1') {	
td_packing_mode	u(8)
view_reverse_flag	u(1)
}	
next_start_code( )	
}	

7.1.2.7 时域可伸缩扩展定义

时域可伸缩扩展定义应符合表21的规定。

表21 时域可伸缩扩展定义

时域可伸缩扩展定义	描述符
temporal_scalability_extension( ) {	
extension_id	f(4)
num_of_temporal_level_minus1	u(3)
for (i=0; i<num_of_temporal_level_minus1; i++) {	
temporal_frame_rate_code[i]	u(4)
temporal_bit_rate_lower[i]	u(18)
marker_bit	f(1)
temporal_bit_rate_upper[i]	u(12)
}	
next_start_code( )	
}	

7.1.2.8 版权扩展定义

版权扩展定义应符合表22的规定。

表22　版权扩展定义

版权扩展定义	描述符
copyright_extension( ) {	
extension_id	f(4)
copyright_flag	u(1)
copyright_id	u(8)
original_or_copy	u(1)
reserved_bits	r(7)
marker_bit	f(1)
copyright_number_1	u(20)
marker_bit	f(1)
copyright_number_2	u(22)
marker_bit	f(1)
copyright_number_3	u(22)
next_start_code( )	
}	

7.1.2.9　高动态范围图像元数据扩展定义

高动态范围图像元数据扩展定义应符合表23的规定。

表23　高动态范围图像元数据扩展定义

高动态范围图像元数据扩展定义	描述符
hdr_dynamic_metadata_extension( ) {	
extension_id	f(4)
hdr_dynamic_metadata_type	u(4)
while (next_bits(24) != '0000 0000 0000 0000 0000 0001') {	
extension_data_byte	u(8)
}	
}	

7.1.2.10　目标设备显示和内容元数据扩展定义

目标设备显示和内容元数据扩展定义应符合表24的规定。

表24　目标设备显示和内容元数据扩展定义

目标设备显示和内容元数据扩展定义	描述符
mastering_display_and_content_metadata_extension( ) {	
extension_id	f(4)
for (c=0; c<3; c++) {	
display_primaries_x[c]	u(16)
marker_bit	f(1)

表 24（续）

目标设备显示和内容元数据扩展定义	描述符
display_primaries_y[c]	u(16)
marker_bit	f(1)
}	
white_point_x	u(16)
marker_bit	f(1)
white_point_y	u(16)
marker_bit	f(1)
max_display_mastering_luminance	u(16)
marker_bit	f(1)
min_display_mastering_luminance	u(16)
marker_bit	f(1)
max_content_light_level	u(16)
marker_bit	f(1)
max_picture_average_light_level	u(16)
marker_bit	f(1)
reserved_bits	r(16)
next_start_code( )	
}	

7.1.2.11 摄像机参数扩展定义

摄像机参数扩展定义应符合表25的规定。

表25 摄像机参数扩展定义

摄像机参数扩展定义	描述符
camera_parameters_extension( ) {	
extension_id	f(4)
reserved_bits	r(1)
camera_id	u(7)
marker_bit	f(1)
height_of_image_device	u(22)
marker_bit	f(1)
focal_length	u(22)
marker_bit	f(1)
f_number	u(22)
marker_bit	f(1)
vertical_angle_of_view	u(22)
marker_bit	f(1)
camera_position_x_upper	i(16)

表 25（续）

摄像机参数扩展定义	描述符
marker_bit	f(1)
camera_position_x_lower	i(16)
marker_bit	f(1)
camera_position_y_upper	i(16)
marker_bit	f(1)
camera_position_y_lower	i(16)
marker_bit	f(1)
camera_position_z_upper	i(16)
marker_bit	f(1)
camera_position_z_lower	i(16)
marker_bit	f(1)
camera_direction_x	i(22)
marker_bit	f(1)
camera_direction_y	i(22)
marker_bit	f(1)
camera_direction_z	i(22)
marker_bit	f(1)
image_plane_vertical_x	i(22)
marker_bit	f(1)
image_plane_vertical_y	i(22)
marker_bit	f(1)
image_plane_vertical_z	i(22)
marker_bit	f(1)
reserved_bits	r(16)
next_start_code( )	
}	

7.1.2.12 感兴趣区域参数扩展定义

感兴趣区域参数扩展定义应符合表26的规定。

表26 感兴趣区域参数扩展定义

感兴趣区域参数扩展定义	描述符
roi_parameters_extension( ) {	
extension_id	f(4)
current_picture_roi_num	u(8)
roiIndex = 0	
if (PictureType != 0) {	
prev_picture_roi_num	u(8)

表 26（续）

感兴趣区域参数扩展定义	描述符
for (i=0; i<PrevPictureROINum; i++) {	
roi_skip_run	ue(v)
if (roi_skip_run != '0') {	
for (j=0; j<roi_skip_run; j++) {	
skip_roi_mode[i+j]	u(1)
if (j % 22 == 0) {	
marker_bit	f(1)
}	
if (skip_roi_mode == '1') {	
ROIInfo[roiIndex]->asisx = PrevROIInfo[i+j] ->asisx	
ROIInfo[roiIndex]->asisy = PrevROIInfo[i+j] ->asisy	
ROIInfo[roiIndex]->width = PrevROIInfo[i+j] -> width	
ROIInfo[roiIndex]->height = PrevROIInfo[i+j] -> height	
roiIndex++	
}	
}	
i += j	
marker_bit	f(1)
}	
else {	
roi_axisx_delta	se(v)
marker_bit	f(1)
roi_axisy_delta	se(v)
marker_bit	f(1)
roi_width_delta	se(v)
marker_bit	f(1)
roi_height_delta	se(v)
marker_bit	f(1)
ROIInfo[roiIndex]->asisx = PrevROIInfo[i+j] ->asisx + ROIAxisxDelta	
ROIInfo[roiIndex]->asisy = PrevROIInfo[i+j]->asisy + ROIAxisyDelta	
ROIInfo[roiIndex]->width = PrevROIInfo[i+j] -> width + ROIWidthDelta	
ROIInfo[roiIndex]->height = PrevROIInfo[i+j]-> height + ROIHeightDelta	
roiIndex++	
}	
}	
}	
for (i=roiIndex; i<current_picture_roi_num; i++) {	
roi_axisx	u(6)
marker_bit	f(1)

表 26（续）

感兴趣区域参数扩展定义	描述符
roi_axisy	u(6)
marker_bit	f(1)
roi_width	u(6)
marker_bit	f(1)
roi_height	u(6)
marker_bit	f(1)
ROIInfo[roiIndex]->asisx = roi_axisx	
ROIInfo[roiIndex]->asisy = roi_axisy	
ROIInfo[roiIndex]->width = roi_width	
ROIInfo[roiIndex]->height = roi_height	
roiIndex++	
}	
for (i=0; i<roiIndex; i++) {	
PrevROIInfo[i]->asisx = ROIInfo[i]->asisx	
PrevROIinfo[i]->asisy = ROIInfo[i]->asisy	
PrevROIinfo[i]->width = ROIInfo[i]->width	
PrevROIinfo[i]->height = ROIInfo[i]->height	
}	
next_start_code( )	
}	

7.1.2.13 参考知识图像扩展定义

参考知识图像扩展定义应符合表27的规定。

表27 参考知识图像扩展定义

参考知识图像扩展定义	描述符
cross_random_access_point_reference_extension( ) {	
extension_id	f(4)
crr_lib_number	u(3)
marker_bit	f(1)
i=0	
while (i<crr_lib_number) {	
crr_lib_pid[i]	u(9)
i++	
if (i%2 == 0) {	
marker_bit	f(1)
}	
}	
next_start_code( )	
}	

7.1.2.14 自适应帧内刷新参数扩展定义

自适应帧内刷新参数扩展定义应符合表28的规定。

表28 自适应帧内刷新参数扩展定义

自适应帧内刷新参数扩展定义	描述符
air_parameters_extension( ) {	
extension_id	f(4)
air_bound_x	u(10)
air_bound_y	u(10)
next_start_code( )	
}	

7.1.3 图像定义

7.1.3.1 帧内预测图像头定义

帧内预测图像头定义应符合表29的规定。

表29 帧内预测图像头定义

帧内预测图像头定义	描述符
intra_picture_header( ) {	
intra_picture_start_code	f(32)
bbv_delay	u(32)
time_code_flag	u(1)
if (time_code_flag == '1')	
time_code	u(24)
decode_order_index	u(8)
if (LibraryStreamFlag)	
library_picture_index	ue(v)
if (temporal_id_enable_flag == '1')	
temporal_id	u(3)
if (low_delay == '0')	
picture_output_delay	ue(v)
if (low_delay == '1')	
bbv_check_times	ue(v)
progressive_frame	u(1)
if (progressive_frame == '0')	
picture_structure	u(1)
top_field_first	u(1)
repeat_first_field	u(1)
if (field_coded_sequence == '1') {	
top_field_picture_flag	u(1)

表 29（续）

帧内预测图像头定义	描述符
reserved_bits	r(1)
}	
ref_pic_list_set_flag[0]	u(1)
if (RefPicListSetFlag[0]) {	
if (NumRefPicListSet[0] > 1) {	
ref_pic_list_set_index[0]	ue(v)
}	
}	
else {	
reference_picture_list_set(0, NumRefPicListSet[0])	
}	
if (Rpl1IndexExistFlag)	
ref_pic_list_set_flag[1]	u(1)
if (RefPicListSetFlag[1]) {	
if (Rpl1IndexExistFlag && NumRefPicListSet[1] > 1) {	
ref_pic_list_set_index[1]	ue(v)
}	
}	
else {	
reference_picture_list_set(1, NumRefPicListSet[1])	
}	
fixed_picture_qp_flag	u(1)
picture_qp	u(7)
if ((! FixedPictureQpFlag) && (profile_id == 0x32 || profile_id == 0x30)) {	
cu_delta_qp_flag	u(1)
if (CuDeltaQpFlag) {	
cu_qp_group_size_log2_minus3	u(2)
}	
}	
deblocking_filter_disable_flag	u(1)
if (! DeblckingFilterDisableFlag) {	
if (profile_id == 0x32 || profile_id == 0x30) {	
deblocking_filter_type	u(1)
}	
deblocking_filter_parameter_flag	u(1)
if (deblocking_filter_parameter_flag == '1') {	
alpha_c_offset	se(v)
beta_offset	se(v)
}	

表 29（续）

帧内预测图像头定义	描述符
if (profile_id == 0x32 || profile_id == 0x30) {	
picture_dbr_v_enable_flag	u(1)
if (PictureDbrVEnableFlag) {	
dbr_v_offset0_minus1	u(2)
dbr_v_ offset1_minus1	u(2)
}	
picture_alt_dbr_v_enable_flag	u(1)
if (PictureAltDbrVEnableFlag) {	
dbr_v_alt_offset0_minus1	u(2)
dbr_v_alt_offset1_minus1	u(2)
}	
if (PictureDbrVEnableFlag || PictureAltDbrVEnableFlag) {	
dbr_v_threshold_minus1	u(1)
}	
picture_dbr_h_enable_flag	u(1)
if (PictureDbrHEnableFlag) {	
dbr_h_offset0_minus1	u(2)
dbr_h_ offset1_minus1	u(2)
}	
picture_alt_dbr_h_enable_flag	u(1)
if (PictureAltDbrHEnableFlag) {	
dbr_h_alt_offset0_minus1	u(2)
dbr_h_alt_offset1_minus1	u(2)
}	
if (PictureDbrHEnableFlag || PictureAltDbrHEnableFlag) {	
dbr_h_threshold_minus1	u(1)
}	
}	
}	
chroma_quant_param_disable_flag	u(1)
if (chroma_quant_param_disable_flag == '0') {	
chroma_quant_param_delta_cb	se(v)
chroma_quant_param_delta_cr	se(v)
}	
if (WeightQuantEnableFlag) {	
picture_weight_quant_enable_flag	u(1)
if (PictureWeightQuantEnableFlag) {	
picture_weight_quant_data_index	u(2)
if (picture_weight_quant_data_index == '01') {	

表 29（续）

帧内预测图像头定义	描述符
reserved_bits	r(1)
weight_quant_param_index	u(2)
weight_quant_model	u(2)
if (weight_quant_param_index == '01') {	
for (i=0; i<6; i++) {	
weight_quant_param_delta1[i]	se(v)
}	
}	
if (weight_quant_param_index == '10') {	
for (i=0; i<6; i++) {	
weight_quant_param_delta2[i]	se(v)
}	
}	
}	
else if (picture_weight_quant_data_index == '10') {	
weight_quant_matrix( )	
}	
}	
}	
if (EsaoEnableFlag) {	
esao_parameter_picture_header_set( )	
}	
if (CcsaoEnableFlag) {	
ccsao_parameter_picture_header_set( )	
}	
if (AlfEnableFlag) {	
for (compIndex=0; compIndex<3; compIndex++) {	
picture_alf_enable_flag[compIndex]	u(1)
}	
if (PictureAlfEnableFlag[0] == 1 || PictureAlfEnableFlag[1] == 1 || PictureAlfEnableFlag[2] == 1) {	
alf_parameter_set( )	
}	
}	
if (PmcEnableFlag){	
picture_pmc_param_index	u(1)
}	
if (SawpEnableFlag) {	
picture_awp_refine_index	u(1)

表 29（续）

帧内预测图像头定义	描述符
}	
if (IbcEnableFlag) {	
picture_ibc_enable_flag	u(1)
}	
if (IscEnableFlag) {	
picture_isc_os_enable_flag	u(1)
picture_isc_evs_ubvs_enable_flag	u(1)
}	
if (FimcEnableFlag) {	
picture_fimc_enable_flag	u(1)
}	
if (IstEnableFlag) {	
picture_ts_enable_flag	u(1)
}	
if (NnFilterEnableFlag) {	
for (compIdx=0; compIdx<3; compIdx++) {	
picture_nn_filter_enable_flag[comIdx]	u(1)
if (PictureNnFilterEnableFlag[comIdx]) {	
picture_nn_filter_adaptive_flag[comIdx]	u(1)
if (PictureNnFilterAdaptiveFlag[comIdx] == 0) {	
if (NumOfNnFilter > 1) {	
picture_nn_filter_set_index[comIdx]	ue(v)
}	
}	
}	
}	
}	
next_start_code( )	
}	

7.1.3.2 帧间预测图像头定义

帧间预测图像头定义应符合表30的规定。

表30 帧间预测图像头定义

帧间预测图像头定义	描述符
inter_picture_header( ) {	
inter_picture_start_code	f(32)
random_access_decodable_flag	u(1)

表 30（续）

帧间预测图像头定义	描述符
bbv_delay	u(32)
picture_coding_type	u(2)
decode_order_index	u(8)
if (temporal_id_enable_flag == '1')	
temporal_id	u(3)
if (low_delay == '0')	
picture_output_delay	ue(v)
if (low_delay == '1')	
bbv_check_times	ue(v)
progressive_frame	u(1)
if (progressive_frame == '0') {	
picture_structure	u(1)
}	
top_field_first	u(1)
repeat_first_field	u(1)
if (field_coded_sequence == '1') {	
top_field_picture_flag	u(1)
reserved_bits	r(1)
}	
ref_pic_list_set_flag[0]	u(1)
if (RefPicListSetFlag[0]) {	
if (NumRefPicListSet[0] > 1) {	
ref_pic_list_set_index[0]	ue(v)
}	
}	
else {	
reference_picture_list_set(0, NumRefPicListSet[0])	
}	
if (Rpl1IndexExistFlag)	
ref_pic_list_set_flag[1]	u(1)
if (RefPicListSetFlag[1]) {	
if (Rpl1IndexExistFlag && NumRefPicListSet[1] > 1) {	
ref_pic_list_set_index[1]	ue(v)
}	
}	
else {	
reference_picture_list_set(1, NumRefPicListSet[1])	
}	
num_ref_active_override_flag	u(1)

表 30（续）

帧间预测图像头定义	描述符
if (num_ref_active_override_flag == '1')	
num_ref_active_minus1[0]	ue(v)
if (picture_coding_type == '10')	
num_ref_active_minus1[1]	ue(v)
}	
fixed_picture_qp_flag	u(1)
picture_qp	u(7)
if ((! FixedPictureQpFlag) && (profile_id == 0x32 || profile_id == 0x30)) {	
cu_delta_qp_flag	u(1)
if (CuDeltaQpFlag) {	
cu_qp_group_size_log2_minus3	u(2)
}	
}	
if (! ((picture_coding_type == '10') && (PictureStructure == 1)))	
reserved_bits	r(1)
deblocking_filter_disable_flag	u(1)
if (! DeblockingFilterDisableFlag) {	
if (profile_id == 0x32 || profile_id == 0x30) {	
deblocking_filter_type	u(1)
}	
deblocking_filter_parameter_flag	u(1)
if (deblocking_filter_parameter_flag == '1') {	
alpha_c_offset	se(v)
beta_offset	se(v)
}	
if (profile_id == 0x32 || profile_id == 0x30) {	
picture_dbr_v_enable_flag	u(1)
if (PictureDbrVEnableFlag) {	
dbr_v_offset0_minus1	u(2)
dbr_v_ offset1_minus1	u(2)
}	
picture_alt_dbr_v_enable_flag	u(1)
if (PictureAltDbrVEnableFlag) {	
dbr_v_alt_offset0_minus1	u(2)
dbr_v_alt_offset1_minus1	u(2)
}	
if (PictureDbrVEnableFlag || PictureAltDbrVEnableFlag) {	
dbr_v_threshold_minus1	u(1)
}	

表 30（续）

帧间预测图像头定义	描述符
picture_dbr_h_enable_flag	u(1)
if (PictureDbrHEnableFlag) {	
dbr_h_offset0_minus1	u(2)
dbr_h_ offset1_minus1	u(2)
}	
picture_alt_dbr_h_enable_flag	u(1)
if (PictureAltDbrHEnableFlag) {	
dbr_h_alt_offset0_minus1	u(2)
dbr_h_alt_offset1_minus1	u(2)
}	
if (PictureDbrHEnableFlag || PictureAltDbrHEnableFlag) {	
dbr_h_threshold_minus1	u(1)
}	
}	
}	
chroma_quant_param_disable_flag	u(1)
if (chroma_quant_param_disable_flag == '0') {	
chroma_quant_param_delta_cb	se(v)
chroma_quant_param_delta_cr	se(v)
}	
if (WeightQuantEnableFlag) {	
picture_weight_quant_enable_flag	u(1)
if (PictureWeightQuantEnableFlag) {	
picture_weight_quant_data_index	u(2)
if (picture_weight_quant_data_index == '01') {	
reserved_bits	r(1)
weight_quant_param_index	u(2)
weight_quant_model	u(2)
if (weight_quant_param_index == '01') {	
for (i=0; i<6; i++) {	
weight_quant_param_delta1[i]	se(v)
}	
}	
if (weight_quant_param_index == '10') {	
for (i=0; i<6; i++) {	
weight_quant_param_delta2[i]	se(v)
}	
}	
}	

表 30（续）

帧间预测图像头定义	描述符
else if (picture_weight_quant_data_index == '10') {	
weight_quant_matrix( )	
}	
}	
}	
if (EsaoEnableFlag) {	
esao_parameter_picture_header_set( )	
}	
if (CcsaoEnableFlag) {	
ccsao_parameter_picture_header_set( )	
}	
if (AlfEnableFlag) {	
for (compIndex=0; compIndex<3; compIndex++) {	
picture_alf_enable_flag[compIndex]	u(1)
}	
if (PictureAlfEnableFlag[0] == 1 || PictureAlfEnableFlag[1] == 1 || PictureAlfEnableFlag[2] == 1) {	
alf_parameter_set( )	
}	
}	
if (AffineEnableFlag) {	
affine_subblock_size_flag	u(1)
}	
if (PmcEnableFlag){	
picture_pmc_param_index	u(1)
}	
if ((AwpEnableFlag && PictureType == 2) || SawpEnableFlag) {	
picture_awp_refine_index	u(1)
}	
if (UmveEnhancementEnableFlag) {	
picture_umve_step_set_index	u(1)
}	
if (ObmcEnableFlag && PictureType == 2) {	
picture_obmc_blending_set_index	u(1)
}	
if (InterPcEnableFlag){	
picture_inter_pc_skip_flag	u(1)
}	
if (IbcEnableFlag) {	

表 30（续）

帧间预测图像头定义	描述符
picture_ibc_enable_flag	u(1)
}	
if (IscEnableFlag) {	
picture_isc_os_enable_flag	u(1)
picture_isc_evs_ubvs_enable_flag	u(1)
}	
if (FimcEnableFlag) {	
picture_fimc_enable_flag	u(1)
}	
if (IstEnableFlag || SbtEnableFlag) {	
picture_ts_enable_flag	u(1)
}	
if (NnFilterEnableFlag) {	
for (compIdx=0; compIdx<3; compIdx++) {	
picture_nn_filter_enable_flag[comIdx]	u(1)
if (PictureNnFilterEnableFlag[comIdx]) {	
picture_nn_filter_adaptive_flag[comIdx]	u(1)
if (PictureNnFilterAdaptiveFlag[comIdx] == 0) {	
if (NumOfNnFilter > 1) {	
picture_nn_filter_set_index[comIdx]	ue(v)
}	
}	
}	
}	
}	
next_start_code( )	
}	

7.1.3.3 图像显示扩展定义

图像显示扩展定义应符合表31的规定。

表31 图像显示扩展定义

图像显示扩展定义	描述符
picture_display_extension( ) {	
extension_id	f(4)
for (i=0; i<NumberOfFrameCentreOffsets; i++) {	
picture_centre_horizontal_offset	i(16)
marker_bit	f(1)

表 31（续）

图像显示扩展定义	描述符
picture_centre_vertical_offset	i(16)
marker_bit	f(1)
}	
next_start_code( )	
}	

7.1.3.4 图像数据定义

图像数据定义应符合表32的规定。

表32 图像数据定义

图像数据定义	描述符
picture_data( ) {	
do {	
patch( )	
} while (next_bits(32) == patch_start_code)	
}	

7.1.4 片定义

片定义应符合表33的规定。

表33 片定义

片定义	描述符
patch( ) {	
patch_start_code	f(32)
if (! FixedPictureQpFlag) {	
fixed_patch_qp_flag	u(1)
patch_qp	u(7)
}	
if (SaoEnableFlag) {	
for (compIndex=0; compIndex<3; compIndex++) {	
patch_sao_enable_flag[compIndex]	u(1)
}	
}	
while (! byte_aligned( ))	
aec_byte_alignment_bit	f(1)
do {	
if (! FixedQP) {	
if (! CuDeltaQpFlag) {	

表 33（续）

片定义	描述符
lcu_qp_delta	ae(v)
}	
PreviousDeltaQP = lcu_qp_delta	
}	
if (SaoEnableFlag) {	
if (PatchSaoEnableFlag[0] || PatchSaoEnableFlag[1] || PatchSaoEnableFlag[2]) {	
if (MergeFlagExist)	
sao_merge_type_index	ae(v)
if (SaoMergeMode == 'SAO_NON_MERGE') {	
for (compIndex=0; compIndex<3; compIndex++) {	
if (PatchSaoEnableFlag[compIndex]) {	
sao_mode[compIndex]	ae(v)
if (SaoMode[compIndex] == 'SAO_Interval') {	
for (j=0; j<MaxOffsetNumber; j++) {	
sao_interval_offset_abs[compIndex][j]	ae(v)
if (SaoIntervalOffsetAbs[compIndex][j])	
sao_interval_offset_sign[compIndex][j]	ae(v)
}	
sao_interval_start_pos[compIndex]	ae(v)
sao_interval_delta_pos_minus2[compIndex]	ae(v)
}	
if (SaoMode[compIndex] == 'SAO_Edge') {	
for (j=0; j<MaxOffsetNumber; j++)	
sao_edge_offset[compIndex][j]	ae(v)
sao_edge_type[compIndex]	ae(v)
}	
}	
}	
}	
}	
}	
if (EsaoEnableFlag) {	
for(compIndex=0; compIndex<3; compIndex++) {	
if (PictureEsaoEnableFlag[compIndex] && PictureEsaoLcuControlFlag[compIndex]) {	
esao_lcu_enable_flag[compIndex][LcuIndex]	ae(v)
if (EsaoLcuEnableFlag[compIndex][LcuIndex] && PictureEsaoSetNum[compIndex]>1) {	
esao_lcu_set_index[compIndex][LcuIndex]	ae(v)
}	

表 33（续）

片定义	描述符

}
}
}
if (CcsaoEnableFlag) {
for(compIndex=1; compIndex<3; compIndex++) {
if (PictureCcsaoEnableFlag[compIndex] && PictureCcsaoLcuControlFlag[compIndex]) {
ccsao_lcu_enable_flag[compIndex][LcuIndex]
if (CcsaoLcuEnableFlag[compIndex][LcuIndex] && PictureCcsaoSetNum[compIndex]>1){
ccsao_lcu_set_index[compIndex][LcuIndex]
}
}
}
}
for (compIndex=0; compIndex<3; compIndex++) {
if (PictureAlfEnableFlag[compIndex] == 1)
alf_lcu_enable_flag[compIndex][LcuIndex]
}
for (compIdx=0; compIdx<3; compIdx++) {
if (pictureNnFilterAdaptiveFlag[compIdx]) {
nn_filter_lcu_enable_flag[compIdx][LcuIdx]
if (NnFilterLcuEnableFlag[compIdx][LcuIdx]) {
if (NumOfNnFilter > 1) {
nn_filter_lcu_set_index[compIdx][LcuIdx]
}
}
}
}
x0 = (LcuIndex % PictureWidthInLcu) * LcuSize
y0 = (LcuIndex / PictureWidthInLcu) * LcuSize
Component = 0
coding_unit_tree(x0, y0, 0, 1<<LcuSizeInBit, 1<<LcuSizeInBit, 1, 'PRED_No_Constraint')
aec_lcu_stuffing_bit
} while (! is_end_of_patch( ))
next_start_code( )
patch_end_code
}
	ae(v)
	ae(v)
	ae(v)
	ae(v)
	ae(v)
	f(32)

7.1.5 编码树定义

编码树定义应符合表34的规定。

表34　编码树定义

编码树定义	描述符
coding_unit_tree(x0, y0, split, width, height, qt, mode) {	
isBoundary = ((x0+width) > PicWidthInLuma) || ((y0+height) > PicHeightInLuma)	
rightBoundary = ((x0+width) > PicWidthInLuma) && ((y0+height) <= PicHeightInLuma)	
bottomBoundary = ((x0 + width) <= PicWidthInLuma) && ((y0 + height) > PicHeightInLuma)	
allowNoSplit = 0	
allowSplitQt = 0	
allowSplitBtVer = 0	
allowSplitBtHor = 0	
allowSplitEqtVer = 0	
allowSplitEqtHor = 0	
if (isBoundary) {	
allowNoSplit = 0	
if ((PictureType == 0) && (width > 64) && (height > 64)) {	
allowSplitQt = 1	
allowNoSplit = 1	
}	
else if ((width == 64 && height > 64) || (height == 64 && width > 64)) {	
allowSplitBtHor = 1	
allowSplitBtVer = 1	
}	
else if (! rightBoundary && ! bottomBoundary) {	
allowSplitQt = 1	
}	
else if (rightBoundary) {	
allowSplitBtVer = 1	
}	
else if (bottomBoundary) {	
allowSplitBtHor = 1	
}	
}	
else {	
if (((width == 64) && (height > 64)) || ((height == 64) && (width > 64))) {	
allowSplitBtHor = 1	
allowSplitBtVer = 1	
allowNoSplit = 1	
}	
else if (split >= MaxSplitTimes) {	

表 34（续）

编码树定义	描述符
allowNoSplit = 1	
}	
else if ((PictureType == 0) && (width == 128) && (height == 128)) {	
allowSplitQt = 1	
allowNoSplit = 1	
}	
else {	
if ((width <= height * MaxPartRatio) && (height <= width * MaxPartRatio))	
allowNoSplit = 1	
if ((width > MinQtSize) && qt)	
allowSplitQt = 1	
if ((width <= MaxBtSize) && (height <= MaxBtSize) && (width > MinBtSize) && (height < MaxPartRatio*width))	
allowSplitBtVer = 1	
if ((width <= MaxBtSize) && (height <= MaxBtSize) && (height > MinBtSize) && (width < MaxPartRatio*height))	
allowSplitBtHor = 1	
if ((width <= MaxEqtSize) && (height <= MaxEqtSize) && (height >= MinEqtSize*2) && (width >= MinEqtSize*4) && (height*4 <= MaxPartRatio*width))	
allowSplitEqtVer = 1	
if ((width <= MaxEqtSize) && (height <= MaxEqtSize) && (width >= MinEqtSize*2) && (height >= MinEqtSize*4) && (width*4 <= MaxPartRatio*height))	
allowSplitEqtHor = 1	
}	
}	
allowSplitBt = allowSplitBtVer || allowSplitBtHor	
allowSplitEqt = allowSplitEqtVer || allowSplitEqtHor	
if (allowSplitQt && (allowNoSplit || allowSplitBt || allowSplitEqt)) {	
qt_split_flag	ae(v)
}	
if (! QtSplitFlag) {	
if (allowNoSplit && (allowSplitBt || allowSplitEqt)) {	
bet_split_flag	ae(v)
}	
if (BetSplitFlag) {	
if (allowSplitBt && allowSplitEqt)	
bet_split_type_flag	ae(v)
if ((! BetSplitTypeFlag && allowSplitBtHor && allowSplitBtVer) || (BetSplitTypeFlag && allowSplitEqtHor && allowSplitEqtVer))	
bet_split_dir_flag	ae(v)
}	

表 34（续）

编码树定义	描述符
}	
if ((mode == 'PRED_No_Constraint') && (PictureType != 0) && ((((BetSplitFlag && !BetSplitTypeFlag) || QtSplitFlag) && (width * height == 64)) || (BetSplitTypeFlag && (width * height == 128)))) {	
root_cu_constraint	ae(v)
modeChild = root_cu_constraint ? 'PRED_Intra_Only' : 'PRED_Inter_Only'	
}	
else {	
modeChild = mode	
}	
if (ChildSizeOccur4) {	
if (Component == 0) {	
LumaWidth = width	
LumaHeight = height	
Component = 1	
}	
}	
if ((! FixedQP) && CuDeltaQpFlag && (width * height >= CuQpGroupAreaSize)) {	
NumDeltaQp = 0	
CuQpGroupX = x0	
CuQpGroupY = y0	
}	
if (BlockSplitMode == 'SPLIT_QT') {	
QtWidth = width / 2	
QtHeight = height / 2	
x1 = x0 + QtWidth	
y1 = y0 + QtHeight	
coding_unit_tree(x0, y0, split+1, QtWidth, QtHeight, 1, modeChild)	
if (x1 < PicWidthInLuma)	
coding_unit_tree(x1, y0, split+1, QtWidth, QtHeight, 1, modeChild)	
if (y1 < PicHeightInLuma)	
coding_unit_tree(x0, y1, split+1, QtWidth, QtHeight, 1, modeChild)	
if ((x1 < PicWidthInLuma) && (y1 < PicHeightInLuma))	
coding_unit_tree(x1, y1, split+1, QtWidth, QtHeight, 1, modeChild)	
if ((LumaWidth == width) && (LumaHeight == height) && ChildSizeOccur4) {	
coding_unit(x0, y0, width, height, 'PRED_No_Constraint', 'COMPONENT_Chroma')	
Component = 0	
}	
}	
else if (BlockSplitMode == 'SPLIT_BT_VER') {	

表 34（续）

编码树定义	描述符
x1 = x0 + width / 2	
coding_unit_tree(x0, y0, split+1, width/2, height, 0, modeChild)	
if (x1 < PicWidthInLuma)	
coding_unit_tree(x1, y0, split+1, width/2, height, 0, modeChild)	
if ((LumaWidth == width) && (LumaHeight == height) && ChildSizeOccur4) {	
coding_unit (x0, y0, width, height, 'PRED_No_Constraint', 'COMPONENT_Chroma')	
Component = 0	
}	
}	
else if (BlockSplitMode == 'SPLIT_BT_HOR') {	
y1 = y0 + height / 2	
coding_unit_tree(x0, y0, split+1, width, height/2, 0, modeChild)	
if (y1 < PicHeightInLuma)	
coding_unit_tree(x0, y1, split+1, width, height/2, 0, modeChild)	
if ((LumaWidth == width) && (LumaHeight == height) && ChildSizeOccur4) {	
coding_unit(x0, y0, width, height, 'PRED_No_Constraint', 'COMPONENT_Chroma')	
Component = 0	
}	
}	
else if (BlockSplitMode == 'SPLIT_EQT_VER') {	
x1 = x0 + width / 4	
x2 = x0 + (3 * width / 4)	
y1 = y0 + height / 2	
coding_unit_tree(x0, y0, split+1, width/4, height, 0, modeChild)	
coding_unit_tree(x1, y0, split+1, width/2, height/2, 0, modeChild)	
coding_unit_tree(x1, y1, split+1, width/2, height/2, 0, modeChild)	
coding_unit_tree(x2, y0, split+ 1, width/4, height, 0, modeChild)	
if ((LumaWidth == width) && (LumaHeight == height) && ChildSizeOccur4) {	
coding_unit(x0, y0, width, height, 'PRED_No_Constraint', 'COMPONENT_Chroma')	
Component = 0	
}	
}	
else if (BlockSplitMode == 'SPLIT_EQT_HOR') {	
x1 = x0 + width / 2	
y1 = y0 + height / 4	
y2 = y0 + (3 * height / 4)	
coding_unit_tree(x0, y0, split+1, width, height/4, 0, modeChild)	
coding_unit_tree(x0, y1, split+1, width/2, height/2, 0, modeChild)	
coding_unit_tree(x1, y1, split+1, width/2, height/2, 0, modeChild)	

表 34（续）

编码树定义	描述符
coding_unit_tree(x0, y2, split+1, width, height/4, 0, modeChild)	
if ((LumaWidth == width) && (LumaHeight == height) && ChildSizeOccur4) {	
coding_unit(x0, y0, width, height, 'PRED_No_Constraint', 'COMPONENT_Chroma')	
Component = 0	
}	
}	
else {	
if (Component == 0) {	
coding_unit(x0, y0, width, height, mode, 'COMPONENT_LUMACHROMA')	
}	
else if (Component == 1) {	
coding_unit(x0, y0, width, height, mode, 'COMPONENT_LUMA')	
}	
}	
}	

7.1.6 编码单元定义

编码单元定义应符合表35的规定。

表35 编码单元定义

编码单元定义	描述符
coding_unit(x0, y0, width, height, mode, component) {	
CodingTreeComponent = component	
if (component == 'COMPONENT_Chroma') {	
if ((PriorCuMode != 0) && (chroma_format != '00'))	
intra_chroma_pred_mode_index	ae(v)
if ((TscpmEnableFlag || PmcEnableFlag) && (IntraChromaPredModeIndex == 1)) {	
if (EnhancedTscpmEnableFlag || PmcEnableFlag) {	
chroma_eipm_index	ae(v)
}	
if (PmcEnableFlag && TscpmEnableFlag && (EnhancedTscpmEnableFlag || (ChromaEipmIndex == 0))) {	
chroma_pmc_index	ae(v)
}	
if (ChromaPmcIndex) {	
chroma_pmc_param_index	ae(v)
}	
}	

表 35（续）

编码单元定义	描述符
NumOfTransBlocks = 3	
ctp_y[0] = 0	
CuCtp += ctp_y[0]	
if (IntraChromaPredMode != 'Intra_Chroma_PCM') {	
if (IntraChromaPredMode == 'Intra_Chroma_PMC' || IntraChromaPredMode == 'Intra_Chroma_PMC_LT' || IntraChromaPredMode == 'Intra_Chroma_PMC_T' || IntraChromaPredMode == 'Intra_Chroma_PMC_L' || IntraChromaPredMode == 'Intra_Chroma_EPMC' || IntraChromaPredMode == 'Intra_Chroma_EPMC_LT' || IntraChromaPredMode == 'Intra_Chroma_EPMC_L' || IntraChromaPredMode == 'Intra_Chroma_EPMC_T' || IntraChromaPredMode == 'Intra_Chroma_EPMC2' || IntraChromaPredMode == 'Intra_Chroma_EPMC2_LT' || IntraChromaPredMode == 'Intra_Chroma_EPMC2_L' || IntraChromaPredMode == 'Intra_Chroma_EPMC2_T') {	
CuCtp += (1 << 1)	
}	
else {	
ctp_u	ae(v)
CuCtp += (ctp_u << 1)	
}	
ctp_v	ae(v)
CuCtp += (ctp_v << 2)	
}	
if (EnhancedStEnableFlag && IntraCuFlag && (IntraChromaPredMode != 'Intra_Chroma_PCM') && (! IntraPfFlag) && (! IipFlag)) {	
if (ctp_u || ctp_v) {	
chroma_st_flag	ae(v)
}	
}	
IstTuFlag = 0	
isIstApply = 0	
for (i=NumOfTransBlocks－2; i<NumOfTransBlocks; i++) {	
IsPcmMode[i] = (IntraChromaPredMode == 'Intra_Chroma_PCM')	
IsChroma = 0	
if (i == NumOfTransBlocks -1 || i == NumOfTransBlocks -2) {	
IsChroma = 1	
}	
block(i, width, height, CuCtp, IsChroma, IsPcmMode[i], isIstApply, IntraCuFlag, component)	
}	
}	
else {	

表 35（续）

编码单元定义	描述符
if (PictureType != 0) {	
if (mode != 'PRED_Intra_Only') {	
skip_flag	ae(v)
}	
AllowAwp = AwpEnableFlag && (width >= 8) && (height >= 8) && (width <= 64) && (height <= 64)	
AllowEtmvp = EtmvpMvapEnableFlag && (width >= 8) && (height >= 8)	
if (SkipFlag) {	
if (UmveEnableFlag || AllowAwp || AllowEtmvp)	
advanced_pred_flag	ae(v)
if (AdvancedPredFlag && AllowEtmvp && (UmveEnableFlag || AllowAwp))	
etmvp_flag	ae(v)
if (AdvancedPredFlag && (! EtmvpFlag) && UmveEnableFlag && AllowAwp)	
awp_flag	ae(v)
if (AffineEnableFlag && (! AdvancedPredFlag) && (width >= 16) && (height >= 16))	
affine_flag	ae(v)
if (AffineUmveEnableFlag && AffineFlag)	
affine_umve_flag	ae(v)
if (InterPcEnableFlag && PictureInterPcSkipFlag && width * height >= 64 && width <= 128 && height <= 128 && !AffineFlag && !AwpFlag &&!EtmvpFlag) {	
inter_pc_flag	ae(v)
if (InterPcFlag)	
inter_pc_index	ae(v)
}	
}	
if (! SkipFlag) {	
if (mode != 'PRED_Intra_Only') {	
direct_flag	ae(v)
}	
if (DirectFlag) {	
if (UmveEnableFlag || AllowAwp || AllowEtmvp)	
advanced_pred_flag	ae(v)
if (AdvancedPredFlag && AllowEtmvp && (UmveEnableFlag || AllowAwp))	
etmvp_flag	ae(v)
if (AdvancedPredFlag && (! EtmvpFlag) && UmveEnableFlag && AllowAwp)	
awp_flag	ae(v)
if (AffineEnableFlag && (! AdvancedPredFlag) && (width >= 16) && (height >= 16))	
affine_flag	ae(v)

表 35（续）

编码单元定义	描述符
if (AffineUmveEnableFlag && AffineFlag)	
affine_umve_flag	ae(v)
if (InterPfEnableFlag && (! AwpFlag) && (! AffineFlag) && (! EtmvpFlag) && (width * height >= 64) && (width <= 64) && (height <= 64)) {	
inter_pf_flag	ae(v)
if (InterPfFlag)	
inter_pf_index	ae(v)
}	
if (InterPcEnableFlag && width * height >= 64 && width <= 128 && height <= 128 && !AffineFlag && !AwpFlag &&!EtmvpFlag && !InterPfFlag) {	
inter_pc_flag	ae(v)
if (InterPcFlag)	
inter_pc_index	ae(v)
}	
}	
if ((! DirectFlag) && (mode == 'PRED_No_Constraint'))	
intra_cu_flag	ae(v)
}	
}	
PartSize = 'SIZE_2Mx2N'	
if (! SkipFlag && ! DirectFlag) {	
ibcAllowedFlag = PictureIbcEnableFlag && (mode == 'PRED_No_Constraint' || mode == 'PRED_Intra_Only') && IntraCuFlag && width <= 64 && height <= 64	
iscAllowedFlag = PictureIscEnableFlag && (mode == 'PRED_No_Constraint' || mode == 'PRED_Intra_Only') && IntraCuFlag && width <=32 && height <= 32	
if (ibcAllowedFlag && iscAllowedFlag) {	
isc_ibc_cu_flag	ae(v)
if (IscIbcCuFlag) {	
ibc_cu_flag	ae(v)
}	
}	
else if ((! ibcAllowedFlag) && iscAllowedFlag) {	
isc_ibc_cu_flag	ae(v)
}	
else if (ibcAllowedFlag && (! iscAllowedFlag)) {	
ibc_cu_flag	ae(v)
}	

表 35（续）

编码单元定义	描述符
if (IbcCuFlag) {	
if (HbvpEmptyFlag && NumOfIntraHmvpCand > 0)	
cbvp_index	ae(v)
if (PictureIbcEnableFlag)	
abvr_index	ae(v)
mv_diff_x_abs_bv	ae(v)
if (MvDiffXAbsBv)	
mv_diff_x_sign_bv	ae(v)
mv_diff_y_abs_bv	ae(v)
if (MvDiffYAbsBv)	
mv_diff_y_sign_bv	ae(v)
}	
if (IscCuFlag) {	
if (PictureIscEvsUbvsEnableFlag && PictureIsc0sEnableFlag)	
isc_subtype_flag	ae(v)
if (! IscSubtypeFlag)	
isc_ordinary_sp_coding_unit(x0, y0, width, height, component)	
else	
isc_evs_ubvs_coding_unit(x0, y0, width, height, component)	
}	
}	
if (SawpEnableFlag && IntraCuFlag && (width >= 8) && (height >= 8) && (width <= 32) && (height <= 32) && (component != 'COMPONENT_CHROMA') && ! IbcCuFlag && ! IscCuFlag) {	
sawp_flag	ae(v)
if (SawpFlag) {	
sawp_index	ae(v)
sawp_pred_mode0_index	ae(v)
sawp_pred_mode1_index	ae(v)
}	
}	
if (DtEnableFlag && IntraCuFlag && ! SawpFlag && ! IbcCuFlag && ! IscCuFlag) {	
allowDtHorSplit = (height >= DtMinSize) && (height <= DtMaxSize) && (width / height < 4) && (width <= DtMaxSize)	
allowDtVerSplit = (width >= DtMinSize) && (width <= DtMaxSize) && (height / width < 4) && (height <= DtMaxSize)	
if (allowDtHorSplit || allowDtVerSplit) {	
dt_split_flag	ae(v)
if (DtSplitFlag) {	
if (allowDtHorSplit && allowDtVerSplit) {	

表 35（续）

编码单元定义	描述符
dt_split_dir	ae(v)
}	
else if (allowDtHorSplit) {	
DtSplitDir = 1	
}	
else {	
DtSplitDir = 0	
}	
if (DtSplitDir) {	
dt_split_hqt_flag	ae(v)
if (! DtSplitHqtFlag) {	
dt_split_hadt_flag	ae(v)
}	
}	
else {	
dt_split_vqt_flag	ae(v)
if (! DtSplitVqtFlag) {	
dt_split_vadt_flag	ae(v)
}	
}	
}	
}	
}	
if (UmveFlag) {	
umve_mv_index	ae(v)
umve_step_index	ae(v)
umve_dir_index	ae(v)
}	
else if ((SkipFlag || DirectFlag) && EtmvpFlag) {	
cu_etmvp_cand_index	ae(v)
}	
else if ((SkipFlag || DirectFlag) && AffineFlag) {	
cu_affine_cand_index	ae(v)
if (AffineUmveFlag){	
affine_umve_step_index0	ae(v)
affine_umve_dir_index0	ae(v)
affine_umve_step_index1	ae(v)
affine_umve_dir_index1	ae(v)
}	

表 35（续）

编码单元定义	描述符
}	
else if ((SkipFlag || DirectFlag) && AwpFlag) {	
if (PictureType == 2) {	
awp_mvr_cand_flag0	ae(v)
if (AwpMvrCandFlag0) {	
awp_mvr_cand_step0	ae(v)
awp_mvr_cand_dir0	ae(v)
}	
awp_mvr_cand_flag1	ae(v)
if (AwpMvrCandFlag1) {	
awp_mvr_cand_step1	ae(v)
awp_mvr_cand_dir1	ae(v)
}	
}	
awp_index	ae(v)
awp_cand_index0	ae(v)
awp_cand_index1	ae(v)
}	
else if (SkipFlag || DirectFlag) {	
cu_subtype_index	ae(v)
}	
if (! SkipFlag && ! DirectFlag && ! IbcCuFlag && ! IscCuFlag) {	
if (! IntraCuFlag) {	
if (AffineEnableFlag && (width >= 16) && (height >= 16))	
affine_flag	ae(v)
if (AmvrEnableFlag) {	
if (EmvrEnableFlag && ! AffineFlag) {	
extend_mvr_flag	ae(v)
}	
if (AffineFlag)	
affine_amvr_index	ae(v)
else	
amvr_index	ae(v)
}	
if (PictureType == 2) {	
inter_pred_ref_mode_index	ae(v)
}	
if (SmvdEnableFlag && SmvdApplyFlag && ! AffineFlag && (InterPredRefModeIndex == 2) && ! ExtendMvrFlag) {	

表 35（续）

编码单元定义	描述符
smvd_flag	ae(v)
}	
if (MvExistL0) {	
if ((! SmvdFlag) && NumRefActive[0] > 1)	
pu_reference_index_l0	ae(v)
mv_diff_x_abs_l0	ae(v)
if (MvDiffXAbsL0)	
mv_diff_x_sign_l0	ae(v)
mv_diff_y_abs_l0	ae(v)
if (MvDiffYAbsL0)	
mv_diff_y_sign_l0	ae(v)
if (AffineFlag) {	
mv_diff_x_abs_l0_affine	ae(v)
if (MvDiffXAbsL0Affine)	
mv_diff_x_sign_l0_affine	ae(v)
mv_diff_y_abs_l0_affine	ae(v)
if (MvDiffYAbsL0Affine)	
mv_diff_y_sign_l0_affine	ae(v)
}	
}	
if (MvExistL1 && (! SmvdFlag)) {	
if (NumRefActive[1] > 1)	
pu_reference_index_l1	ae(v)
mv_diff_x_abs_l1	ae(v)
if (MvDiffXAbsL1)	
mv_diff_x_sign_l1	ae(v)
mv_diff_y_abs_l1	ae(v)
if (MvDiffYAbsL1)	
mv_diff_y_sign_l1	ae(v)
if (AffineFlag) {	
mv_diff_x_abs_l1_affine	ae(v)
if (MvDiffXAbsL1Affine)	
mv_diff_x_sign_l1_affine	ae(v)
mv_diff_y_abs_l1_affine	ae(v)
if (MvDiffYAbsL1Affine)	
mv_diff_y_sign_l1_affine	ae(v)
}	
}	
if (BgcEnableFlag && MvExistL0 && MvExistL1 && (width * height >= 256)) {	

表 35（续）

编码单元定义	描述符
bgc_flag	ae(v)
if (BgcFlag)	
bgc_index	ae(v)
}	
}	
else {	
TuOrder = 0	
for (i=0; i<NumOfIntraPredBlock; i++) {	
intra_luma_pred_mode_index	ae(v)
if (EipmEnableFlag && (IntraLumaPredModeIndex > 1)) {	
luma_eipm_index	ae(v)
}	
}	
if (PartSize == 'SIZE_2Mx2N') {	
IsPcmMode[TuOrder] = (IntraLumaPredMode == 'Intra_Luma_PCM')	
TuOrder++	
}	
else {	
IsPcmMode[0] = 0	
IsPcmMode[1] = 0	
IsPcmMode[2] = 0	
IsPcmMode[3] = 0	
TuOrder=3	
}	
if (IntraCuFlag && (chroma_format != '00') && (component == 'COMPONENT_LUMACHROMA')) {	
intra_chroma_pred_mode_index	ae(v)
if ((TscpmEnableFlag || PmcEnableFlag) && (IntraChromaPredModeIndex == 1)) {	
if (EnhancedTscpmEnableFlag || PmcEnableFlag) {	
chroma_eipm_index	ae(v)
}	
if (PmcEnableFlag && TscpmEnagleFlag && (EnhancedTscpmEnableFlag || (ChromaEipmIndex == 0))) {	
chroma_pmc_index	ae(v)
}	
if (ChromaPmcIndex) {	
chroma_pmc_param_index	ae(v)
}	

表 35（续）

编码单元定义	描述符
}	
IsPcmMode[TuOrder+1] = (IntraChromaPredMode == 'Intra_Chroma_PCM')	
IsPcmMode[TuOrder+2] = (IntraChromaPredMode == 'Intra_Chroma_PCM')	
}	
if (IntraPfEnableFlag && (PartSize == 'SIZE_2Mx2N') && (! IsPcmMode[0]) && (! SawpFlag)) {	
intra_pf_flag	ae(v)
}	
if (IipEnableFlag && (PartSize == 'SIZE_2Mx2N') && (!IsPcmMode[0]) && (! IntraPfFlag) && (width * height >= 64) && (width <=64) && (height <=64) && (! SawpFlag)) {	
iip_flag	ae(v)
}	
}	
}	
CuCtp = 0	
if ((! IntraCuFlag && ! SkipFlag) || IbcCuFlag) {	
if (! DirectFlag && component == 'COMPONENT_LUMACHROMA')	
ctp_zero_flag	ae(v)
if (! CtpZeroFlag) {	
if (PbtEnableFlag && (width / height < 4) && (height / width < 4) && (width >= 8) && (width <= 32) && (height >= 8) && (height <= 32) && (! InterPfFlag) && (! InterPcFlag)) {	
pbt_cu_flag	ae(v)
}	
if (! PbtCuFlag) {	
if (component == 'COMPONENT_LUMACHROMA') {	
ctp_u	ae(v)
ctp_v	ae(v)
}	
CuCtp = ctp_u << (NumOfTransBlocks - 2)	
CuCtp += (ctp_v << (NumOfTransBlocks - 1))	
if (((ctp_u != 0) || (ctp_v != 0)) || (component !='COMPONENT_LUMACHROMA')) {	
ctp_y[0]	ae(v)
CuCtp += ctp_y[0]	
}	
else {	
CuCtp += ctp_y[0]	
}	

表 35（续）

编码单元定义	描述符
if (SbtEnableFlag && (width <= 64) && (height <= 64) && (width > 4 || height > 4) && (! InterPfFlag) && (! InterPcFlag) && ctp_y[0] && (! IbcCuFlag)) {	
sbt_cu_flag	ae(v)
if (SbtCuFlag) {	
SbtVerHalf = width >= 8 && (height * 2 <= width * MaxPartRatio)	
SbtVerQuad = width >= 16 && (height * 4 <= width * MaxPartRatio)	
SbtHorHalf = height >= 8 && (width * 2 <= height * MaxPartRatio)	
SbtHorQuad = height >= 16 && (width * 4 <= height * MaxPartRatio)	
if ((SbtVerHalf || SbtHorHalf) && (SbtVerQuad || SbtHorQuad)) {	
sbt_quad_flag	ae(v)
}	
if ((SbtQuadFlag && SbtVerQuad && SbtHorQuad) || (!SbtQuadFlag && SbtVerHalf && SbtHorHalf)) {	
sbt_dir_flag	ae(v)
}	
}	
}	
}	
else {	
if (component == 'COMPONENT_LUMACHROMA') {	
ctp_u	ae(v)
ctp_v	ae(v)
}	
CuCtp = ctp_u << (NumOfTransBlocks - 2)	
CuCtp += (ctp_v << (NumOfTransBlocks - 1))	
for (i=0; i<NumOfTransBlocks - 2; i++) {	
ctp_y[i]	ae(v)
CuCtp += (ctp_y[i] << i)	
}	
}	
}	
}	
else if (! SkipFlag && ! IscCuFlag) {	
CuCtp = 0	
if (! IsPcmMode[0]) {	
for (i=0; i<NumOfTransBlocks - 2; i++) {	
ctp_y[i]	ae(v)
CuCtp += (ctp_y[i] << i)	
}	

表 35（续）

编码单元定义	描述符
if (PartSize == 'SIZE_2Mx2N' && ctp_y[0] && (width < 64) && (height < 64) && IntraCuFlag && PictureTsEnableFlag) {	
enhanced_ts_flag	ae(v)
}	
}	
if ((component == 'COMPONENT_LUMACHROMA') && (IntraChromaPredMode != 'Intra_Chroma_PCM')) {	
if (IntraChromaPredMode == 'Intra_Chroma_PMC' || IntraChromaPredMode == 'Intra_Chroma_PMC_LT' || IntraChromaPredMode == 'Intra_Chroma_PMC_T' || IntraChromaPredMode == 'Intra_Chroma_PMC_L' || IntraChromaPredMode == 'Intra_Chroma_EPMC' || IntraChromaPredMode == 'Intra_Chroma_EPMC_LT' || IntraChromaPredMode == 'Intra_Chroma_EPMC_L' || IntraChromaPredMode == 'Intra_Chroma_EPMC_T' || IntraChromaPredMode == 'Intra_Chroma_EPMC2' || IntraChromaPredMode == 'Intra_Chroma_EPMC2_LT' || IntraChromaPredMode == 'Intra_Chroma_EPMC2_L' || IntraChromaPredMode == 'Intra_Chroma_EPMC2_T') {	
ctp_u = 1	
}	
else {	
ctp_u	ae(v)
}	
ctp_v	ae(v)
}	
CuCtp += (ctp_u << (NumOfTransBlocks - 2))	
CuCtp += (ctp_v << (NumOfTransBlocks - 1))	
}	
if ((! FixedQP) && CuDeltaQpFlag && CuCtp && (component != 'COMPONENT_CHROMA')) {	
cu_qp_delta_abs	ae(v)
if (cu_qp_delta_abs)	
cu_qp_delta_sign	ae(v)
NumDeltaQp += 1	
}	
if (EnhancedStEnableFlag && (component == 'COMPONENT_LUMACHROMA) && IntraCuFlag && (!IbcCuFlag) && (IntraChromaPredMode != 'Intra_Chroma_PCM') && (! IntraPFlag) && (! IipFlag) && (! SawpFlag)) {	
if (ctp_u || ctp_v) {	
chroma_st_flag	ae(v)
}	
}	
IstTuFlag=0	

表 35（续）

编码单元定义	描述符
if (PartSize == 'SIZE_2Mx2N' && ctp_y[0] && (width < 64) && (height < 64) && IbcCuFlag && PictureTsEnableFlag &&IstEnableFlag) {	
ts_cu_flag	ae(v)
IstTuFlag = TsCuFlag	
}	
for (i=0; i<NumOfTransBlocks; i++) {	
if (i < NumOfTransBlocks - 2) {	
blockWidth = ((TransformSplitDirection == 0) || (TransformSplitDirection == 2)) ? width : (TransformSplitDirection == 1 ? width >> 1 : width >> 2)	
blockHeight = ((TransformSplitDirection == 0) || (TransformSplitDirection == 3)) ? height : (TransformSplitDirection == 1 ? height >> 1 : height >> 2)	
blockX = x0 + (((TransformSplitDirection == 0) || (TransformSplitDirection == 2)) ? 0 : (TransformSplitDirection == 1 ? ((blockWidth >> 1) * (i % 2)) : ((blockWidth >> 2) * i)))	
blockY = y0 + (((TransformSplitDirection == 0) || (TransformSplitDirection == 3)) ? 0 : (TransformSplitDirection == 1 ? ((blockHeight >> 1) * (i / 2)) : ((blockHeight >> 2) * i)))	
if (SbtCuFlag) {	
blockWidth = SbtDirFlag ? blockWidth : (SbtQuadFlag ? blockWidth / 4 : blockWidth / 2)	
blockHeight = ! SbtDirFlag ? blockHeight : (SbtQuadFlag ? blockHeight / 4 : blockHeight / 2)	
}	
}	
IsChroma = 0	
if (i == NumOfTransBlocks -1 || i == NumOfTransBlocks -2) {	
IsChroma = 1	
}	
isIstApply = ! DtSplitFlag && ! SbtCuFlag && ! IbcCuFlag && !PbtCuFlag && (blockWidth<64 && blockHeight<64) && IstEnableFlag	
block(i, blockWidth, blockHeight, CuCtp, IsChroma, IsPcmMode[i], isIstApply, IntraCuFlag, component)	
if ((i < NumOfTransBlocks - 2) && SbtCuFlag) {	
blockX = (SbtDirFlag || (! SbtPosFlag)) ? 0 : (SbtQuadFlag ? blockWidth * 3 / 4: blockWidth / 2)	
blockY = (!SbtDirFlag || (! SbtPosFlag)) ? 0 : (SbtQuadFlag ? blockHeight * 3 / 4: blockHeight / 2)	
}	
}	
}	
}	

7.1.7 变换块定义

变换块定义应符合表36的规定。

表36 变换块定义

变换块定义	描述符
block(i, blockWidth, blockHeight, CuCtp, isChroma, isPcm, isIstApply, isIntra, component) {	
M_1 = blockWidth	
M_2 = blockHeight	
for (x=0; x<M_1; x++) {	
for (y=0; y<M_2; y++)	
QuantCoeffMatrix[x][y] = 0	
}	
if (! isPcm) {	
if (CuCtp & (1 << i)) {	
if (SrccEnableFlag) {	
scan_region_x	ae(v)
scan_region_y	ae(v)
initScanOrder(ScanOrder, ScanRegionX + 1, ScanRegionY + 1, blockWidth)	
lastScanPos = ((ScanRegionX + 1) * (ScanRegionY + 1)) - 1	
lastSet = lastScanPos >> 4	
is_last_x = 0	
is_last_y = 0	
EscapeDataPresent = 0	
NumEvenCoeff = 0	
Pre5Gt0[5] = {0, 0, 0, 0, 0}	
Pre5GtX[5] = {0, 0, 0, 0, 0}	
for(i = lastset; i >= 0; i - -) {	
setPos = i << 4	
set_nz[i] = 0	
for(n = (i == lastSet ? lastScanPos - setPos : 15); n >= 0; n—) {	
blkpos = ScanOrder[setPos + n];	
PosX = blkpos & (blockWidth - 1)	
PosY = blkpos >> Log(blockWidth)	
if ((PosX == 0 && PosY == ScanRegionY && is_last_y == 0) || (PosY == 0 && PosX == ScanRegionX && is_last_x == 0)) {	
sig_flag[blkpos] = 1	
}	
else {	
significant_flag	ae(v)
sig_flag[blkpos] = SignificantFlag	
}	

表 36（续）

变换块定义	描述符
if (sig_flag[blkpos]) {	
if (PosX == ScanRegionX)	
is_last_x = 1	
if (PosY == ScanRegionY)	
is_last_y = 1	
set_nz[i]++	
}	
for (j=4; j > 0; j--) {	
Pre5Gt0[j] = Pre5Gt0[j-1]	
}	
Pre5Gt0[0] = sig_flag[blkpos]	
}	
if (set_nz[i]) {	
EscapeDataPresent = 0	
if (PictureTsEnableFlag) {	
for (n=(i == lastSet ? lastScanPos - setPos : 15); n>=0; n—) {	
blkpos = ScanOrder[setPos + n]	
if (sig_flag[blkpos]) {	
coeff_abs_level_greater1_flag	ae(v)
if (CoeffAbsLevelGt1Flag) {	
coeff_abs_level_greater2_flag	ae(v)
if (CoeffAbsLevelGt2Flag) {	
coeff_abs_level_greater4_flag	ae(v)
if (CoeffAbsLevelGt4Flag) {	
coeff_abs_level_greater8_flag	ae(v)
if (CoeffAbsLevelGt8Flag) {	
EscapeDataPresent = 1	
abs_coef[blkpos] = 9	
}	
else {	
base_level = 5	
coeff_abs_level_remaining	ae(v)
abs_coef[blkpos] = base_level + CoeffAbsLevelRem	
}	
}	
else {	
base_level = 3;	

表 36（续）

变换块定义	描述符
coeff_abs_level_remaining	ae(v)
abs_coef[blkpos] = base_level + CoeffAbsLevelRem	
}	
}	
else {	
abs_coef[blkpos] = 2	
}	
}	
else {	
abs_coef[blkpos] = 1	
}	
for (j=4; j > 0; j++) {	
Pre5GtX[j] = Pre5GtX[j-1]	
}	
Pre5GtX[0] = abs_coef [blkpos]	
}	
}	
}	
else {	
for (n=(i == lastSet ? lastScanPos - setPos : 15); n >= 0; n--) {	
blkpos = ScanOrder[setPos + n]	
if (sig_flag[blkpos]) {	
coeff_abs_level_greater1_flag	ae(v)
if (coeffAbsLevelGt1Flag) {	
coeff_abs_level_greater2_flag	ae(v)
if (CoeffAbsLevelGt2Flag) {	
EscapeDataPresent = 1	
abs_coef[blkpos] = 3	
}	
else {	
abs_coef[blkpos] = 2	
}	
}	
else {	
abs_coef[blkpos] = 1	
}	
for (j=4; j > 0; j++) {	

表 36（续）

变换块定义	描述符
Pre5GtX[j] = Pre5GtX[j-1]	
}	
Pre5GtX[0] = abs_coef [blkpos]	
}	
}	
}	
}	
if (EscapeDataPresent) {	
if (PictureTsEnableFlag) {	
base_level = 9	
}	
else {	
base_level = 3	
}	
for(n = (i == lastSet ? lastScanPos - setPos: 15); n >= 0; n—) {	
blkpos = ScanOrder[setPos + n]	
if (sig_flag[blkpos]) {	
if (abs_coef[blkpos] == base_level) {	
coeff_abs_level_remaining	ae(v)
abs_coef[blkpos] += CoeffAbsLevelRem	
}	
}	
}	
}	
for (n = (i == lastSet ? lastScanPos - setPos : 15); n >= 0; n—) {	
blkpos = ScanOrder[setPos + n]	
if (sig_flag[blkpos]) {	
PosX = blkpos & (blockwidth - 1)	
PosY = blkpos >> Log(blockWidth)	
coeff_sign	ae(v)
QuantCoeffMatrix[PosX][PosY] = CoeffSign ? -abs_coef[blkpos] : abs_coef[blkpos]	
}	
}	
}	
for (x=0; x<M_1; x++) {	
for (y=0; y<M_2; y++){	
if (QuantCoeffMatrix[x][y] % 2 == 0) {	

表 36（续）

变换块定义	描述符
NumEvenCoeff++	
}	
}	
}	
if (SbtCuFlag) {	
SbtPosFlag = NumEvenCoeff % 2 ? 0 : 1	
}	
if (isIstApply) {	
IstTuFlag = (! PictureTsEnableFlag && (ScanRegionX >=16 || ScanRegionY >=16)) ? 0 : NumEvenCoeff % 2 ? 0 : 1	
if (PictureTsEnableFlag && IntraCuFlag) {	
IstTuFlag = EnhancedTsFlag ? (IstTuFlag ? 2 : 1) : 0	
}	
}	
}	
else {	
blockWidth = isChroma ? blockWidth / 2 : blockWidth	
blockHeight = isChroma ? blockHeight / 2 : blockHeight	
indexW = Log(blockWidth) - 1	
indexH = Log(blockHeight) - 1	
NumOf Coeff = blockWidth * blockHeight	
NumEvenCoeff = 0	
ScanPosOffset = 0	
CoeffRestrict = 0	
do {	
coeff_run	ae(v)
coeff_level_minus1	ae(v)
coeff_sign	ae(v)
AbsLevel = coeff_level_minus1 + 1	
ScanPosOffset = ScanPosOffset + coeff_run	
PosxInBlk = InvScanCoeffInBlk[indexW][indexH][ScanPosOffset][0]	
PosyInBlk = InvScanCoeffInBlk[indexW][indexH][ScanPosOffset][1]	
if (PosxInBlk >= 16 || PosyInBlk >= 16) {	
CoeffRestrict = 1	
}	
QuantCoeffMatrix[PosxInBlk][PosyInBlk] = coeff_sign ? -AbsLevel : AbsLevel	
if (ScanPosOffset >= NumOf Coeff - 1) {	
break	

表 36（续）

变换块定义	描述符
}	
coeff_last	ae(v)
ScanPosOffset = ScanPosOffset + 1	
} while (! coeff_last)	
for(x=0; x<M_1; x++) {	
for(y=0; y<M_2; y++){	
if (QuantCoeffMatrix[x][y] % 2 == 0) {	
NumEvenCoeff++	
}	
}	
}	
if (SbtCuFlag) {	
SbtPosFlag = NumEvenCoeff % 2 ? 0 : 1	
}	
if (isIstApply) {	
IstTuFlag = NumEventCoeff % 2 ? 0 : 1	
}	
if (CoeffRestrict && ! PictureTsEnableFlag) {	
IstTuFlag = 0	
}	
if (isIstApply && PictureTsEnableFlag && IntraCuFlag) {	
IstTuFlag = EnhancedTsFlag ? (IstTuFlag ? 2 : 1) : 0	
}	
}	
if (EnhancedStEnableFlag && (! isChroma)) {	
if (isIntra && (! DtSplitFlag) && (! IstTuFlag) && (! IbcCuFlag)) {	
est_tu_flag	ae(v)
}	
}	
}	
}	
else {	
if ((component != 'COMPONENT_CHROMA' && i == 0) || (component == 'COMPONENT_CHROMA' && i == 1)) {	
aec_ipcm_stuffing_bit	ae(v)
while (! byte_aligned()) {	
aec_byte_alignment_bit0	f(1)
}	
}	

表 36（续）

变换块定义	描述符
M_1 = isChroma ? blockWidth / 2 : blockWidth	
M_2= isChroma ? blockHeight / 2 : blockHeight	
xMin = Min(32, M_1)	
yMin = Min(32, M_2)	
for (yStep=0; yStep<M_2/yMin; yStep++) {	
for (xStep=0; xStep<M_1/xMin; xStep++) {	
for (y=0; y<yMin; y++) {	
for (x=0; x<xMin; x++) {	
pcm_coeff	f(n)
QuantCoeffMatrix[x+xStep*xMin][y + yStep*yMin] = pcm_coeff	
}	
}	
}	
}	
}	
}	

7.1.8 自适应修正滤波参数定义

自适应修正滤波参数定义应符合表37的规定。

表37 自适应修正滤波参数定义

自适应修正滤波参数定义	描述符
alf_parameter_set( ) {	
if (EalfEnableFlag) {	
coeffNum = 15	
FilterNum = 64	
}	
else {	
coeffNum = 9	
FilterNum = 16	
}	
if (PictureAlfEnableFlag[0] == 1) {	
alf_filter_num_minus1	ue(v)
if (EalfEnableFlag) {	
alf_region_order_index	u(2)
}	
for (i=0; i<alf_filter_num_minus1+1; i++) {	
if ((i > 0) && (alf_filter_num_minus1 != (FilterNum-1)))	

表 37（续）

自适应修正滤波参数定义	描述符
alf_region_distance[i]	ue(v)
for (j=0; j<coeffNum; j++)	
alf_coeff_luma[i][j]	se(v)
if (EalfEnableFlag)	
alf_luma_shift_num_minus6[i]	se(v)
}	
}	
if (PictureAlfEnableFlag[1] == 1) {	
for (j=0; j<coeffNum; j++)	
alf_coeff_chroma[0][j]	se(v)
if (EalfEnableFlag)	
alf_chroma_shift_num_minus6[0]	se(v)
}	
if (PictureAlfEnableFlag[2] == 1) {	
for (j=0; j<coeffNum; j++)	
alf_coeff_chroma[1][j]	se(v)
if (EalfEnableFlag)	
alf_chroma_shift_num_minus6[1]	se(v)
}	
}	

7.1.9 增强样值偏移自适应补偿参数定义

增强样值偏移自适应补偿参数定义应符合表38的规定。

表38 增强样值偏移自适应补偿参数定义

增强样值偏移自适应补偿参数定义	描述符
esao_parameter_picture_header_set( ) {	
for(compIndex=0;compIndex<3;compIndex++){	
picture_esao_enable_flag[compIndex]	u(1)
if (PictureEsaoEnableFlag[compIndex]){	
picture_esao_lcu_control_flag[compIndex]	u(1)
if (PictureEsaoLcuControlFlag[compIndex]){	
picture_esao_set_num_minus1[compIndex]	u(1)
}	
for (setIndex=0; setIndex<PictureEsaoSetNum[compIndex]; setIndex++) {	
if (compIndex == 0) {	
picture_esao_adaptive_param_minus1[setIndex]	u(4)
esao_luma_param_type[setIndex]	u(1)

表 38（续）

增强样值偏移自适应补偿参数定义	描述符
}	
else {	
picture_esao_adaptive_param_index[compIndex][setIndex]	u(7)
esao_chroma_band_mode_flag[compIndex][setIndex]	u(1)
if (EsaoChromaBandModeFlag[compIndex][**setIndex**]) {	
esao_chroma_start_band[compIndex][setIndex]	ue(v)
esao_chroma_band_num[compIndex][setIndex]	ue(v)
}	
}	
numCoeff = (compIndex == 0) ? ((EsaoLumaParamType[setIndex] ? 9 : 17) * PictureEsaoAdaptiveParam[setIndex]) : EsaoTableUV[PictureEsaoAdaptiveParamIndex[compIndex][setIndex]]	
startIndex = (compIndex == 0) ? 0 : (EsaoChromaBandModeFlag[compIndex][setIndex] ? EsaoChromaStartBand[compIndex][setIndex] : 0)	
numCoeff = (compIndex != 0 && EsaoChromaBandModeFlag[compIndex][setIndex]) ? EsaoChromaBandNum[compIndex][setIndex] : numCoeff	
for(i=startIndex; i<startIndex+numCoeff; i++) {	
esao_filter_coeff[compIndex][setIndex][i]	se(v)
}	
}	
}	
}	
}	

7.1.10 跨分量样值偏移自适应补偿参数定义

跨分量样值偏移自适应补偿参数定义应符合表39的规定。

表39 跨分量样值偏移自适应补偿参数定义

跨分量样值偏移自适应补偿参数定义	描述符
ccsao_parameter_picture_header_set( ) {	
for (compIndex=1; compIndex<3;compIndex++) {	
picture_ccsao_enable_flag[compIndex]	u(1)
if (PictureCcsaoEnableFlag[compIndex]) {	
picture_ccsao_lcu_control_flag[compIndex]	u(1)
if (PictureCcsaoLcuControlFlag[compIndex]) {	
picture_ccsao_set_num_minus1[compIndex]	u(2)
}	
for (setIndex=0; setIndex<PictureCcsaoSetNum[compIndex]; setIndex++) {	

表 39（续）

跨分量样值偏移自适应补偿参数定义	描述符
picture_ccsao_mode[compIndex][setIndex]	u(1)
if (PictureCcsaoMode[compIndex][setIndex]) {	
picture_ccsao_edge_type[compIndex][setIndex]	u(2)
picture_ccsao_edge_band_num_index[compIndex][setIndex]	u(2)
picture_ccsao_edge_thr_index[compIndex][setIndex]	u(3)
}	
else {	
picture_ccsao_cand_index[compIndex][setIndex]	u(4)
picture_ccsao_luma_band_num_minus1[compIndex][setIndex]	u(4)
picture_ccsao_chroma_band_num_minus1[compIndex][setIndex]	u(1)
}	
for (i=0; i<PictureCcsaoClassNum[compIndex][setIndex]; i++) {	
ccsao_offset[compIndex][setIndex][i]	se(v)
}	
}	
}	
}	

7.1.11 串复制帧内预测定义

串复制帧内预测的普通串子模式编码单元定义应符合表40的规定。

表40 串复制帧内预测的普通串子模式编码单元定义

串复制帧内预测的普通串子模式编码单元定义	描述符
isc_ordinary_sp_coding_unit(x0,y0,width,height,component) {	
NumCodedPixel = 0	
NumTotalPixel = width * height	
i = 0	
while (NumCodedPixel < NumTotalPixel) {	
StrXInCu = TravScan[Log(width) - 2][Log(height) - 2] [NumCodedPixel][0]	
StrYInCu = TravScan[Log(width) - 2][Log(height) - 2] [NumCodedPixel][1]	
NumMatchedPixel[i] = 0	
isc_match_type[i]	ae(v)
if (IscMatchType[i] == 1) {	
CurRemainingPixelQuarter = (NumTotalPixel - NumCodedPixel) >> 2	
if (CurRemainingPixelQuarter ==1) {	
StrLen[i] = NumTotalPixel - NumCodedPixel	
}	

表 40（续）

串复制帧内预测的普通串子模式编码单元定义	描述符
else {	
isc_next_remaining_pixel_in_cu[i]	ae(v)
StrLen[i] = NumTotalPixel - NumCodedPixel - (IscNextRemainingPixelInCu[i] << 2)	
}	
}	
else {	
StrLen[i] = 4	
for (j=0; j<StrLen[i]; j++) {	
isc_pixel_match_type[i][j]	ae(v)
if (IscPixelMatchType[i][j]) {	
NumMatchedPixel += 1	
}	
}	
}	
if (IscMatchType[i] || NumMatchedPixel) {	
isc_sv_above_flag[i]	ae(v)
if (! IscSvAboveFlag) {	
isc_sv_recent_flag[i]	ae(v)
if (IscSvRecentFlag) {	
isc_sv_recent_index[i]	ae(v)
}	
else {	
isc_sv_y_abs[i]	ae(v)
if (IscSvYAbs[i])	
isc_sv_y_sign[i]	ae(v)
if (IscSvYAbs[i] == 0) {	
isc_sv_x_sign[i]	ae(v)
isc_sv_x_abs_minus1[i]	ae(v)
}	
else {	
if (! IscSvYSign[i]) {	
isc_sv_x_abs_minus1[i]	ae(v)
}	
else {	
isc_sv_x_non_zero_flag[i]	ae(v)
if (IscSvXNonZeroFlag[i]) {	
isc_sv_x_sign[i]	ae(v)
isc_sv_x_abs_minus1[i]	ae(v)

表 40（续）

串复制帧内预测的普通串子模式编码单元定义	描述符
} }	
} }	
} }	
} }	
NumCodedPixel += StrLen[i]	
i++	
}	
IscPartNum = i	
NumCodedPixel = 0	
i = 0	
while (NumCodedPixel < NumTotalPixel) {	
if (IscMatchType[i] == 0) {	
for (j=0; j<StrLen[i]; j++) {	
if (! IscPixelMatchType[i][j]) {	
StrXInCu = TravScan[Log(width) - 2][Log(height) - 2] [NumCodedPixel + j][0]	
StrYInCu = TravScan[Log(width) - 2][Log(height) - 2] [NumCodedPixel + j][1]	
isc_unmatched_pixel_y[i][j]	ae(v)
if (component == 'COMPONENT_LUMACHROMA' && ! (StrXInCu & 0x1 || StrYInCu & 0x1)) {	
isc_unmatched_pixel_cb[i][j]	ae(v)
isc_unmatched_pixel_cr[i][j]	ae(v)
}	
}	
}	
}	
NumCodedPixel += StrLen[i]	
i++	
}	
}	

串复制帧内预测非普通串子模式编码单元定义应符合表41的规定。

表41　串复制帧内预测非普通串子模式编码单元定义

串复制帧内预测非普通串子模式编码单元定义	描述符
isc_evs_ubvs_coding_unit(x0,y0,width,height,component) {	
i = 0	
NumTotalPixel = width * height	

表 41（续）

串复制帧内预测非普通串子模式编码单元定义	描述符
NumCodedPixel = 0	
isc_ups_present_flag	ae(v)
PvNum = 0	
isc_num_of_new_pv	ae(v)
if (PrevPvBufSize > 0 && IscNumOfNewPv < 15)	
isc_num_of_reused_pv	ae(v)
for (k = 0; PvNum < IscNumOfReusedPv; k++) {	
isc_prev_pv_not_reused_run	ae(v)
k += IscPrevPvNotReusedRun	
PpInfoList[PvNum][0] = PrevPpInfoList[k][0]	
PpInfoList[PvNum][1] = PrevPpInfoList[k][1]	
FopYonly[PvNum] = PrevFopYonly[k]	
EvsDpbIndex[PvNum] = PrevEvsDpbIndex[k]	
EvsDpbReactivatedYonly[PvNum] = PrevEvsDpbReactivatedYonly[k]	
CompLumaFreqOccurPos[PvNum] = PrevCompLumaFreqOccurPos[k]	
PvNum ++	
PrevPpInfoList[k][0] = -1	
PrevPpInfoList[k][1] = -1	
}	
MaxValPvAddress = IscNumOfReusedPv - 1 + IscUpsPresentFlag	
MaxValPvAddressUpperBound = MaxValPvAddress + IscNumOfNewPv	
if (IscUpsPresentFlag)	
isc_ups_max_length_minus1	ae(v)
if (IscNumOfNewPv > 0)	
MaxValPvAddress++	
while (NumCodedPixel < width)	
isc_evs_ups_coding(x0,y0,width,height,component)	
FirstStringNumFlag = 1	
while (NumCodedPixel < NumTotalPixel) {	
isc_evs_ups_num_minus1	ae(v)
if (FirstStringNumFlag) {	
StringNum = IscEvsUPSNumMinus1	
FirstStringNumFlag = 0	
}	
else	
StringNum = IscEvsUPSNumMinus1 + 1	
for (k = 0; k < StringNum; k++) {	
isc_evs_ups_coding(x0,y0,width,height,component)	
}	

表 41（续）

串复制帧内预测非普通串子模式编码单元定义	描述符
if (NumCodedPixel < NumTotalPixel) {	
maxStrLength = NumTotalPixel - NumCodedPixel	
maxValPrefix = Ceil(Log(maxStrLength)) - 1	
k = 0	
if (maxStrLength>2) {	
str_length_minus1_prefix	ae(v)
}	
if (StrLengthMinus1Prefix==0){	
maxValInfix = maxStrLength==1? 0:1	
}	
else {	
StrLengthMinus1Prefix += 1	
maxValInfix = min(maxStrLength - 1 - (1 << (StrLengthMinus1Prefix - 1)), (1 << (StrLengthMinus1Prefix - 1)) - 1)	
}	
n = Ceil(Log(maxValInfix+1))	
if (n >1)	
str_length_minus1_infix	ae(v)
if (! (StrLengthMinus1Infix < ((1<<n) - maxValInfix - 1) || maxValInfix == 0)) {	
k = 1	
str_length_minus1_suffix	ae(v)
}	
if (StrLengthMinus1Prefix ==0) {	
StrLengthMinus1 = StrLengthMinus1Suffix	
}	
else {	
StrLengthMinus1 =(1 << (StrLengthMinus1Prefix -1)) + (StrLengthMinus1Infix << k) + StrLengthMinus1Suffix - ((1<<n) - maxValInfix-1) × k	
}	
StrLen[i] = StrLengthMinus1 + 1	
StringType[i] = 0	
for (k=0; k<StrLen[i]; ++k) {	
sxInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][0]	
syInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][1]	
offsetAbove = (syInCu & 1) == 0 ? 2*sxInCu + 1 : 2*(width - sxInCu) - 1	
PvFlag[sxInCu][syInCu] = 0	
if (PixelType[NumCodedPixel - offsetAbove] == 2) {	
PixelType[NumCodedPixel] = 2	

表 41（续）

串复制帧内预测非普通串子模式编码单元定义	描述符
UnmatchedPixelIndex[NumCodedPixel] = UnmatchedPixelIndex[NumCodedPixel - offsetAbove]	
}	
else {	
PixelType[NumCodedPixel] = 0	
PvAddr[NumCodedPixel] = PvAddr[NumCodedPixel - offsetAbove]	
if (component == 'COMPONENT_LUMACHROMA' && ((sxInCu | syInCu) & 0x1) == 0 && EvsDpbReactivatedYonly[PvAddr[NumCodedPixel]] == 1) {	
xInLcuR = x0 + sxInCu - LcuRx0	
yInLcuR = y0 + syInCu - LcuRy0	
LcuRowBufY[xInLcuR][yInLcuR] = LcuRowBufY[PPInfoList[PvAddr[NumCodedPixel]][0]][PPInfoList[PvAddr[NumCodedPixel]][1]]	
LcuRowBufU[xInLcuR>>1][yInLcuR>>1] = EvsDpb[1][EvsDpbIndex[PvAddr[NumCodedPixel]]]	
LcuRowBufV[xInLcuR>>1][yInLcuR>>1] = EvsDpb[2][EvsDpbIndex[PvAddr[NumCodedPixel]]]	
PpInfoList[PvAddr[NumCodedPixel]][0] = xInLcuR	
PpInfoList[PvAddr[NumCodedPixel]][1] = yInLcuR	
EvsDpbIndex[PvAddr[NumCodedPixel]] = 28	
EvsDpbReactivatedYonly[PvAddr[NumCodedPixel]] = 0	
PvFlag[sxInCu][syInCu] = 1	
}	
}	
NumCodedPixel++	
}	
i ++	
}	
}	
IscPartNum = i	
i = 0	
NumCodedPixel = 0	
while (NumCodedPixel < NumTotalPixel) {	
if (StringType[i] == 2) {	
for (k=0; k<StrLen[i]; k++) {	
isc_nos_up_y[i+k]	ae(v)
if (! upYonly[i+k]) {	
isc_nos_up_cb[i+k]	ae(v)
isc_nos_up_cr[i+k]	ae(v)
}	

表 41（续）

串复制帧内预测非普通串子模式编码单元定义	描述符
}	
NumCodedPixel += StrLen[i]	
i += k	
}	
else if (StringType[i] == 1) {	
if (PvType[i] == 1 || PvType[i] == 3) {	
isc_fopixel_y	ae(v)
LcuRowBufY[PpInfoList[PvAddr[NumCodedPixel]][0]][PpInfoList[PvAddr[NumCodedPixel]][1]] = IscFopixelY	
}	
if (PvType[i] == 2 || PvType[i] == 3) {	
isc_fopixel_cb	ae(v)
isc_fopixel_cr	ae(v)
LcuRowBufU[PpInfoList[PvAddr[NumCodedPixel]][1]>>1][PpInfoList[PvAddr[NumCodedPixel]][1]>>1] = IscFopixelCb	
LcuRowBufV[PpInfoList[PvAddr[NumCodedPixel]][1]>>1][PpInfoList[PvAddr[NumCodedPixel]][1]>>1] = IscFopixelCr	
}	
NumCodedPixel += StrLen[i]	
i++	
}	
else {	
NumCodedPixel += StrLen[i]	
i++	
}	
}	
}	

串复制帧内预测等值串与未匹配样本串编码过程定义应符合表42的规定。

表42　串复制帧内预测等值串与未匹配样本串编码过程定义

串复制帧内预测等值串与未匹配样本串编码过程定义	描述符
isc_evs_ups_coding(x0, y0, width, height, component) {	
if (IscNumOfNewPv + IscNumOfReusedPv == 0) {	
maxStrLength = IscUpsMaxLengthMinus1 + 1	
maxValPrefix = Ceil(Log(maxStrLength))	
k = 0	
if (maxStrLength > 1) {	
str_length_minus1_prefix	ae(v)

表 42（续）

串复制帧内预测等值串与未匹配样本串编码过程定义	描述符
}	
if (StrLengthMinus1Prefix == 0) {	
maxValInfix = 0	
}	
else {	
maxValInfix = min(maxStrLength - 1 - (1 << (StrLengthMinus1Prefix - 1)), (1 << (StrLengthMinus1Prefix - 1)) - 1)	
}	
n = Ceil(Log(maxValInfix +1))	
if (n >1) {	
str_length_minus1_infix	ae(v)
}	
if (! (StrLengthMinus1Infix < ((1<<n) - maxValInfix - 1) || maxValInfix == 0)) {	
k=1	
str_length_minus1_suffix	ae(v)
}	
if (StrLengthMinus1Prefix == 0) {	
StrLengthMinus1 = 0	
}	
else {	
StrLengthMinus1 =(1 << (StrLengthMinus1Prefix -1)) + (StrLengthMinus1Infix << k) + StrLengthMinus1Suffix - ((1<<n) - maxValInfix-1) × k	
}	
StrLen[i] = StrLengthMinus1 + 1	
for (k=0; k< StrLengthMinus1 + 1; k++) {	
sxInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][0]	
syInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][1]	
StringType[i] = 2	
PixelType[NumCodedPixel] = 2	
UnmatchedPixelIndex[NumCodedPixel] = i	
upYonly[i] = 1	
if (component == 'COMPONENT_LUMACHROMA' && (sxInCu | syInCu]) & 0x1) == 0) {	
upYonly[i] = 0	
}	
PvAddr[NumCodedPixel] = 0	
PvFlag[sxInCu][syInCu] = 0	
PvType[i] = 0	
i++	
NumCodedPixel++	

表 42（续）

串复制帧内预测等值串与未匹配样本串编码过程定义	描述符
}	
}	
else {	
MaxVal=MaxValPvAddress-1	
n = Ceil(Log(MaxValPvAddress + 1))	
k = 0	
if (n > 1)	
pv_address_prefix	ae(v)
if (! (PvAddressPrefix < ((1<<n) - MaxVal-1) || MaxVal == 0)) {	
k = 1	
pv_address_infix	ae(v)
}	
offsetAbove = (TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][1] & 1) == 0 ? 2×TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][0] + 1 : 2×(width - TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][0]) - 1	
if((NumCodedPixel == 0 || NumCodedPixel > 0 && StringType[i - 1] == 0 && PixelType[NumCodedPixel-offsetAbove]==2) && PvAddressPrefix==0 && PvAddressInfix==0 && MaxValPvAddress!=0)	
pv_address_suffix	ae(v)
PvAddress = (PvAddressPrefix << k) + PvAddressInfix - ((1<<n)-MaxVal-1) × k - PvAddressSuffix	
if (NumCodedPixel > 0) {	
if (StringType[i - 1] != 0) {	
if ((PvAddress += (PvAddr[NumCodedPixel-1]+1)) >= (MaxVal +2))	
PvAddress -= (MaxVal + 2)	
}	
else if (PixelType[NumCodedPixel - offsetAbove] != 2) {	
if ((PvAddress += (PvAddr[NumCodedPixel-offsetAbove]+1)) >= (MaxVal +2))	
PvAddress -= (MaxVal + 2)	
}	
else {	
if ((PvAddress += (MaxVal + 1 +1)) >= (MaxVal +2))	
PvAddress -= (MaxVal + 2)	
}	
}	
else {	
if ((PvAddress += (MaxVal + 1 + 1)) >= (MaxVal +2))	
PvAddress -= (MaxVal + 2)	
}	
PvType[i] = 0	

表 42（续）

串复制帧内预测等值串与未匹配样本串编码过程定义	描述符
if ((PvNum < IscNumOfReusedPv + IscNumOfNewPv) && (PvAddress == (MaxValPvAddress-IscUpsPresentFlag))) {	
maxStrLength = NumTotalPixel - NumCodedPixel	
maxValPrefix = Ceil(Log(maxStrLength))	
k = 0	
if (maxStrLength > 1) {	
str_length_minus1_prefix	ae(v)
}	
if (StrLengthMinus1Prefix == 0) {	
maxValInfix = 0	
}	
else {	
maxValInfix = min(maxStrLength - 1 - (1 << (StrLengthMinus1Prefix - 1)), (1 << (StrLengthMinus1Prefix - 1)) - 1)	
}	
n = Ceil(Log(maxValInfix +1))	
if (n >1) {	
str_length_minus1_infix	ae(v)
}	
if (! (StrLengthMinus1Infix < ((1<<n) - maxValInfix - 1) || maxValInfix == 0)) {	
k = 1	
str_length_minus1_suffix	ae(v)
}	
if (StrLengthMinus1Prefix == 0) {	
StrLengthMinus1 = 0	
}	
else {	
StrLengthMinus1 = (1 << (StrLengthMinus1Prefix -1)) + (StrLengthMinus1Infix << k) + StrLengthMinus1Suffix - ((1<<n) - maxValInfix-1) × k	
}	
StrLen[i] = StrLengthMinus1 + 1	
xInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][0]	
yInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][1]	
xInLcuR = x0 + xInCu - LcuRx0	
yInLcuR = y0 + yInCu - LcuRy0	
PpInfoList[PvNum][0] = xInLcuR	
PpInfoList[PvNum][1] = yInLcuR	
PvType[i] = 1	
FopYonly[PvNum] = component == 'COMPONENT_LUMA' ? 0 : 1	
EvsDpbIndex[PvNum] = 28	

表 42（续）

串复制帧内预测等值串与未匹配样本串编码过程定义	描述符
EvsDpbReactivatedYonly[PvNum] = 0	
CompLumaFreqOccurPos[PvNum] = component == 'COMPONENT_LUMA' ? 1 : 0	
for (k=0; k<StrLen[i]; k++) {	
xInCu2 = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel+k][0]	
yInCu2 = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel+k][1]	
PvFlag[xInCu2][yInCu2] = 0	
}	
PvNum++	
NumOfNewPvReceived++	
if (MaxValPvAddress < MaxValPvAddressUpperBound)	
MaxValPvAddress++	
}	
else {	
if (PvAddress < PvNum) {	
maxStrLength = NumTotalPixel - NumCodedPixel	
maxValPrefix = Ceil(Log(maxStrLength))	
k = 0	
if(maxStrLength > 1) {	
str_length_minus1_prefix	ae(v)
}	
if(StrLengthMinus1Prefix==0) {	
maxValInfix = 0	
}	
else {	
maxValInfix = min(maxStrLength - 1 - (1 << (StrLengthMinus1Prefix - 1)), (1 << (StrLengthMinus1Prefix - 1)) - 1)	
}	
n = Ceil(Log(maxValInfix +1))	
if(n >1) {	
str_length_minus1_infix	ae(v)
}	
if(!(StrLengthMinus1Infix <((1<<n) - maxValInfix - 1) || maxValInfix==0)) {	
k=1	
str_length_minus1_suffix	ae(v)
}	
if(StrLengthMinus1Prefix ==0) {	
StrLengthMinus1 = 0	
}	
else {	

表 42（续）

串复制帧内预测等值串与未匹配样本串编码过程定义	描述符
StrLengthMinus1 =(1 << (StrLengthMinus1Prefix -1)) + (StrLengthMinus1Infix << k) + StrLengthMinus1Suffix - ((1<<n) - maxValInfix-1) × k	
}	
StrLen[i] = StrLengthMinus1 + 1	
}	
}	
if (PvAddress < PvNum) {	
if (component == 'COMPONENT_LUMACHROMA' && FopYonly[PvAddress]) {	
log2Size = log(min(LcuSize,64))	
PvX = PpInfoList[PvAddress][0]	
PvY = PpInfoList[PvAddress][1]	
if (PvX >> log2Size != (x0-LcuRx0) >> log2Size || PvY >> log2Size != (y0-LcuRy0) >> log2Size) {	
xInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][0]	
yInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][1]	
xInLcuR = x0 + xInCu - LcuRx0	
yInLcuR = y0 + yInCu - LcuRy0	
if (EvsDpbIndex[PvAddress] < 28) {	
LcuRowBufY[xInLcuR][yInLcuR] = EvsDpb[0][EvsDpbIndex[PvAddress]]	
EvsDpbIndex[PvAddress] = 28	
}	
else {	
LcuRowBufY[xInLcuR][yInLcuR] = LcuRowBufY[PpInfoList[PvAddress][0]][PpInfoList[PvAddress][1]]	
}	
PpInfoList[PvAddress][0] = xInLcuR	
PpInfoList[PvAddress][1] = yInLcuR	
}	
for (k=0; k<StrLen[i]; k++) {	
xInCu = TravScan[Log(width)-2][Log(height)-2][NumCodedPixel+k][0]	
yInCu = TravScan[Log(width)-2][Log(height)-2][NumCodedPixel+k][1]	
if (((xInCu | yInCu) & 0x1) == 0) {	
xInLcuR = x0 + xInCu - LcuRx0	
yInLcuR = y0 + yInCu - LcuRy0	
LcuRowBufY[xInLcuR][yInLcuR] = LcuRowBufY[PpInfoList[PvAddress]][0]][PpInfoList[PvAddress]][1]]	
PpInfoList[PvAddress][0] = xInLcuR	
PpInfoList[PvAddress][1] = yInLcuR	
PvFlag[xInCu][yInCu] = 1	
if (PvType[i] == 1)	

表 42（续）

串复制帧内预测等值串与未匹配样本串编码过程定义	描述符
PvType[i] = 3	
else	
PvType[i] = 2	
FopYonly[PvAddress] = 0	
break	
}	
}	
}	
else if (component == 'COMPONENT_LUMACHROMA' && PvAddress < IscNumOfReusedPv) {	
if (EvsDpbIndex[PvAddress] < 28) {	
xInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][0]	
yInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][1]	
xInLcuR = x0 + xInCu - LcuRx0	
yInLcuR = y0 + yInCu - LcuRy0	
PpInfoList[PvAddress][0] = xInLcuR	
PpInfoList[PvAddress][1] = yInLcuR	
LcuRowBufY[xInLcuR][yInLcuR] = EvsDpb[0][EvsDpbIndex[PvAddress]]	
EvsDpbReactivatedYonly[PvAddress] = 1	
}	
for (k=0; k<StrLen[i]; k++) {	
xInCu = TravScan[Log(width)-2][Log(height)-2][NumCodedPixel+k][0]	
yInCu = TravScan[Log(width)-2][Log(height)-2][NumCodedPixel+k][1]	
log2Size = log(min(LcuSize,64))	
PvX = PpInfoList[PvAddress][0]	
PvY = PpInfoList[PvAddress][1]	
if (((xInCu | yInCu) & 0x1) == 0 && ((PvX >> log2Size != (x0-LcuRx0) >> log2Size || PvY >> log2Size != (y0-LcuRy0) >> log2Size) || (EvsDpbIndex[PvAddress] < 28))) {	
xInLcuR = x0 + xInCu - LcuRx0	
yInLcuR = y0 + yInCu - LcuRy0	
PvFlag[xInCu][yInCu] = 1	
LcuRowBufY[xInLcuR][yInLcuR] = LcuRowBufY[PPInfoList[PvAddress][0]][PPInfoList[PvAddress][1]]	
if (EvsDpbIndex[PvAddress] < 28){ //a sleeping pv	
LcuRowBufU[xInLcuR>>1][yInLcuR>>1] = EvsDpb[1][EvsDpbIndex[PvAddress]]	
LcuRowBufV[xInLcuR>>1][yInLcuR>>1] = EvsDpb[2][EvsDpbIndex[PvAddress]]	
EvsDpbIndex[PvAddress] = 28	
EvsDpbReactivatedYonly[PvAddress] = 0	
}	

表 42（续）

串复制帧内预测等值串与未匹配样本串编码过程定义	描述符
else {	
LcuRowBufU[xInLcuR>>1][yInLcuR>>1] = LcuRowBufU[PpInfoList[PvAddress][0]>>1][PpInfoList[PvAddress][1]>>1]	
LcuRowBufV[xInLcuR>>1][yInLcuR>>1] = LcuRowBufV[PpInfoList[PvAddress][0]>>1][PpInfoList[PvAddress][1]>>1]	
}	
PpInfoList[PvAddress][0] = xInLcuR	
PpInfoList[PvAddress][1] = yInLcuR	
break	
}	
}	
}	
StringType[i] = 1	
i++	
for (k=0; k<StrLen[i]; k++) {	
PixelType[NumCodedPixel] = 1	
PvAddr[NumCodedPixel] = PvAddress	
NumCodedPixel++	
}	
}	
else {	
maxStrLength = IscUpsMaxLengthMinus1 + 1	
maxValPrefix = Ceil(Log(maxStrLength))	
k=0	
if(maxStrLength>1) {	
str_length_minus1_prefix	ae(v)
}	
if(StrLengthMinus1Prefix==0) {	
maxValInfix = 0	
}	
else {	
maxValInfix = min(maxStrLength - 1 - (1 << (StrLengthMinus1Prefix - 1)), (1 << (StrLengthMinus1Prefix - 1)) - 1)	
}	
n = Ceil(Log(maxValInfix +1))	
if (n >1) {	
str_length_minus1_infix	ae(v)
}	
if (! (StrLengthMinus1Infix < ((1<<n) - maxValInfix - 1) || maxValInfix==0)) {	
k=1	

表 42（续）

串复制帧内预测等值串与未匹配样本串编码过程定义	描述符
str_length_minus1_suffix	ae(v)
}	
if (StrLengthMinus1Prefix ==0) {	
StrLengthMinus1 = 0	
}	
else {	
StrLengthMinus1 =(1 << (StrLengthMinus1Prefix -1)) + (StrLengthMinus1Infix << k) + StrLengthMinus1Suffix - ((1<<n) - maxValInfix-1) × k	
}	
StrLen[i] = StrLengthMinus1 + 1	
for (k=0; k< StrLengthMinus1 + 1; k++) {	
sxInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][0]	
syInCu = TravScan[Log(width) - 2][Log(height) - 2][NumCodedPixel][1]	
StringType[i] = 2	
PixelType[NumCodedPixel] = 2	
UnmatchedPixelIndex[NumCodedPixel] = i	
upYonly[i] = 1	
if (component == 'COMPONENT_LUMACHROMA' && ((sxInCu | syInCu) & 0x1) == 0) {	
upYonly[i] = 0	
}	
PvFlag[sxInCu][syInCu] = 0	
PvAddr[NumCodedPixel] = PvAddress	
i++	
NumCodedPixel++	
}	
}	
}	
}	

7.2 语义描述

7.2.1 视频扩展语义

本文件定义了若干视频扩展，在语法的不同位置，可出现的视频扩展是不同的。每一种视频扩展都有一个唯一的视频扩展标号，应符合表43的规定。

表43 视频扩展标号

视频扩展标号	含义
0000	保留
0001	保留

表 43（续）

视频扩展标号	含义
0010	序列显示扩展
0011	时域可伸缩扩展
0100	版权扩展
0101	高动态范围图像元数据扩展
0110	禁止
0111	图像显示扩展
1000	保留
1001	保留
1010	目标设备显示和内容元数据扩展
1011	摄像机参数扩展
1100	感兴趣区域参数扩展
1101	参考知识图像扩展
1110	自适应帧内刷新参数扩展
1111	保留

7.2.2 视频序列语义

7.2.2.1 视频序列

视频编辑码 video_edit_code

位串‘0x000001B7’。说明紧跟video_edit_code的第一幅I图像后续的B图像可能因缺少参考图像而不能正确解码。

视频序列结束码 video_sequence_end_code

位串‘0x000001B1’。标识视频序列的结束。如果POI的值大于$(2^{32}-1)$，位流中应插入1个视频序列结束码。

7.2.2.2 序列头

视频序列起始码 video_sequence_start_code

位串‘0x000001B0’。标识视频序列的开始。

类标号 profile_id

8位无符号整数。表示位流符合的类。

级标号 level_id

8位无符号整数。表示位流符合的级。

类和级应符合附录B的规定。

知识位流标志 library_stream_flag

二值变量。值为‘1’表示当前位流是知识位流，值为‘0’表示当前位流是主位流。LibraryStreamFlag的值等于library_stream_flag的值。

知识图像允许标志 library_picture_enable_flag

二值变量。值为‘1’表示视频序列中可存在使用知识图像作为参考图像的帧间预测图像，值为‘0’表示视频序列中不应存在使用知识图像作为参考图像的帧间预测图像。LibraryPictureEnableFlag的值

等于library_picture_enable_flag的值。如果位流中不存在library_picture_enable_flag，LibraryPicutreEnableFlag的值等于0。

知识位流重复序列头标志 duplicate_sequence_header_flag

二值变量。值为‘1’表示当前主位流的序列头中除library_stream_flag、library_picture_enable_flag和duplicate_sequence_header_flag外的所有语法元素的值均应与当前主位流所参考的知识位流的序列头中对应语法元素的值相同，值为‘0’表示当前主位流的序列头中除library_stream_flag、library_picture_enable_flag和duplicate_sequence_header_flag外的其它语法元素的值可与当前主位流所参考的知识位流的序列头中对应语法元素的值不同。

逐行视频序列标志 progressive_sequence

二值变量。规定视频序列的扫描格式。值为‘1’表示编码视频序列只包含逐行扫描的帧图像；值为‘0’表示编码视频序列只包含逐行扫描图像，或表示编码视频序列只包含隔行扫描图像。

如果progressive_sequence的值为‘1’，相邻两个显示时刻间隔为帧周期。如果progressive_sequence的值为‘0’，相邻两个显示时刻间隔为场周期。

场视频序列标志 field_coded_sequence

二值变量。值为‘1’表示编码视频序列中的图像是场图像，值为‘0’表示编码视频序列中的图像是帧图像。如果progressive_sequence的值为‘1’，则field_coded_sequence的值应为‘0’。

水平尺寸 horizontal_size

14位无符号整数。规定图像亮度分量可显示区域（该区域与图像的左侧边缘对齐）的宽度，即水平方向样本数。

PictureWidthInMinBu和PictureWidthInMinCu的计算如下：

```
PictureWidthInMinBu = (horizontal_size + MiniSize - 1) / MiniSize
PictureWidthInMinCu = (PictureWidthInMinBu * MiniSize + MiniSize - 1) / MinCuSize
```

horizontal_size不应为‘0’。horizontal_size的单位应是图像每行样本数。可显示区域的左上角样本应与解码图像左上角样本对齐。

垂直尺寸 vertical_size

14位无符号整数。规定图像亮度分量可显示区域（该区域与图像的顶部边缘对齐）的高度，即垂直方向扫描行数。

在视频序列位流中，当progressive_sequence和field_coded_sequence的值均为‘0’时，PictureHeightInMinBu和PictureHeightInMinCu的计算如下：

```
PictureHeightInMinBu = 2 * ((vertical_size + 2 * MiniSize - 1) / (2 * MiniSize))
PictureHeightInMinCu = 2 * ((PictureHeightInMinBu * MiniSize + 2 * MiniSize - 1) / (2 * MinCuSize))
```

在其他情况下，PictureHeightInMinBu和PictureHeightInMinCu的计算如下：

```
PictureHeightInMinBu = (vertical_size + MiniSize - 1) / MiniSize
PictureHeightInMinCu = (vertical_size + MinCuSize - 1) / MinCuSize
```

vertical_size不应为0。vertical_size的单位应是图像样本的行数。

MiniSize的值由类规定，应符合附录B的规定。

horizontal_size、vertical_size与图像边界的关系应符合图6的规定。图6中，实线表示图像可显示区域边界，其宽度和高度分别由horizontal_size和vertical_size决定；虚线表示图像边界，其宽度

和高度分别由PictureWidthInMinBu和PictureHeightInMinBu决定。PicWidthInLuma的值等于PictureWidthInMinBu乘以MiniSize的积，PicHeightInLuma的值等于PictureHeightInMinBu乘以MiniSize的积。例如horizontal_size的值为1920，vertical_size的值为1080，则当progressive_sequence和field_coded_sequence的值均为‘0’时，PictureWidthInMinBu * MiniSize等于1920，PictureHeightInMinBu * MiniSize等于1088；否则PictureWidthInMinBu * MiniSize等于1920，PictureHeightInMinBu * MiniSize等于1080。

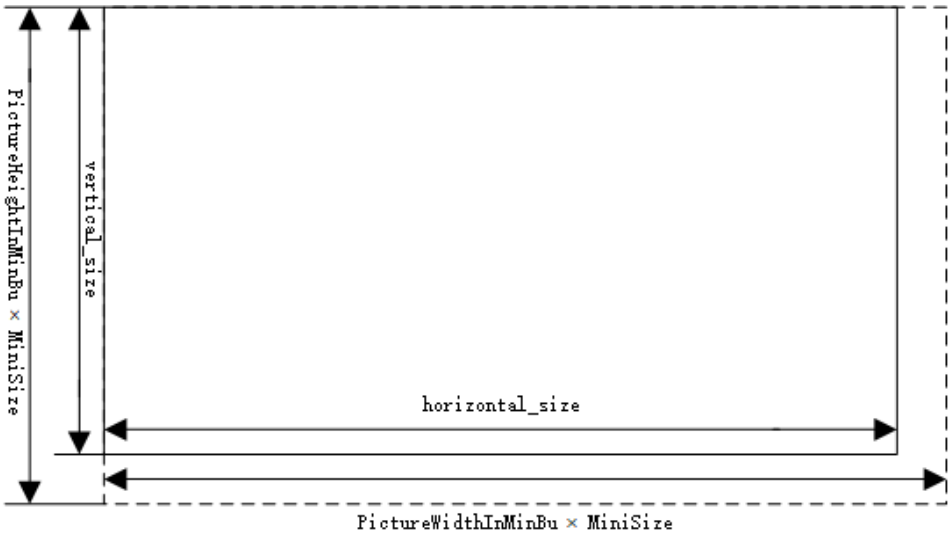


图6　图像边界示意图

色度格式　chroma_format

2位无符号整数。规定色度分量的格式，应符合表44的规定。

表44　色度格式

chroma_format的值	含义
00	保留
01	4:2:0
10	保留
11	保留

样本精度　sample_precision

3位无符号整数。规定亮度和色度样本的精度，应符合表45的规定。如果sample_precision的值为‘001’，则SamplePrecision的值为8；如果sample_precision的值为‘010’，则SamplePrecision的值为10。

表45　样本精度

sample_precision的值	含义
000	禁止
001	亮度和色度均为8bit精度
010	亮度和色度均为10bit精度
011～111	保留

编码样本精度　encoding_precision

3位无符号整数。规定亮度和色度样本的编码精度，应符合表46的规定。如果encoding_precision的值为‘001’，则BitDepth的值为8；如果encoding_precision的值为‘010’，则BitDepth的值为10。如果位流中不存在encoding_precision，则BitDepth的值为8。BitDepth的值不应小于SamplePrecision的值。

表46　编码样本精度

encoding_precision的值	含义
000	禁止
001	亮度和色度均为8bit精度
010	亮度和色度均为10bit精度
011～111	保留

宽高比　aspect_ratio

4位无符号整数，规定重建图像的SAR或DAR，应符合表47的规定。

表47　宽高比

aspect_ratio的值	SAR	DAR
0000	禁止	禁止
0001	1.0	—
0010	—	4:3
0011	—	16:9
0100	—	2.21:1
0101～1111	—	保留

如果位流中没有序列显示扩展，那么整个重建图像将要映射到整个活动显示区域。样本宽高比为：

```
SAR = DAR * vertical_size ÷ horizontal_size
```

注：在这种情况下，horizontal_size和vertical_size受源图像的样本宽高比和选定的显示宽高比限制。

如果位流中有序列显示扩展出现，样本宽高比为：

```
SAR = DAR * display_vertical_size ÷ display_horizontal_size
```

帧率代码　frame_rate_code

4位无符号整数。规定帧率，应符合表48的规定。

表48　帧率代码

frame_rate_code的值	帧率
0000	禁止
0001	24000÷1001（23.976...）
0010	24
0011	25
0100	30000÷1001（29.97...）

表 48（续）

frame_rate_code的值	帧率
0101	30
0110	50
0111	60000÷1001（59.94...）
1000	60
1001	100
1010	120
1011	200
1100	240
1101	300
1110	120000÷1001（119.88...）
1111	保留

连续两帧之间的时间间隔是帧率的倒数。隔行扫描帧中两场之间的时间间隔是帧率的倒数的1/2。

如果progressive_sequence的值为‘1’，帧周期等于帧率的倒数。

如果progressive_sequence的值为‘0’，场周期等于帧率的倒数的1/2。

比特率低位 bit_rate_lower

BitRate的低18位。

比特率高位 bit_rate_upper

BitRate的高12位。BitRate为：

```
BitRate = (bit_rate_upper << 18) + bit_rate_lower
```

BitRate以400bit/s为单位计算视频位流的比特率，并向上取整。BitRate不应为0。对LibraryPictureEnableFlag的值为1的主位流，视频位流的比特率包含了该主位流以及所参考的知识位流的总比特率。

低延迟 low_delay

二值变量。值为‘1’表示参考图像队列0和参考图像队列1中均不包含显示顺序上将来的图像，不存在图像重排序延时，位流中可能包含所谓“大图像”（应符合附录C的规定）；值为‘0’表示参考图像队列0或参考图像队列1中可包含显示顺序上将来的图像,存在图像重排序延时,位流中不包含所谓“大图像”（应符合附录C的规定）。

时间层标识允许标志 temporal_id_enable_flag

二值变量。值为‘1’表示视频序列允许使用时间层标识，值为‘0’表示视频序列不使用时间层标识。

位流缓冲区尺寸 bbv_buffer_size

18位无符号整数。规定了位流参考解码器对视频序列解码的位流缓冲区尺寸(应符合附录C的规定)。BBS是位流参考解码器对视频序列解码所需的位流缓冲区最小尺寸（按位计算）：

```
BBS = 16 * 1024 * bbv_buffer_size
```

最大解码图像缓冲区大小　max_dpb_size_minus1

4位无符号整数。表示解码当前位流所需要的最大的解码图像缓冲区的大小（以单幅图像存储缓冲区大小为单位）。max_dpb_size_minus1的值应大于0，小于16且小于当前位流的level_id对应的最大DPB值（具体由级规定，应符合附录B的规定）。MaxDpbSize的值等于max_dpb_size_minus1加1。

参考图像队列1索引存在标志　rpl1_index_exist_flag

二值变量。值为‘0’表示位流中不应出现ref_pic_list_set_flag[1]和ref_pic_list_set_index[1]；值为‘1’表示可出现。Rpl1IndexExistFlag的值等于rpl1_index_exist_flag的值。

参考图像队列相同标志　rpl1_same_as_rpl0_flag

二值变量。值为‘0’表示位流中应出现num_ref_pic_list_set[1]和reference_picture_list_set(1,rplsIndex)，值为‘1’表示不应出现。Rpl1SameAsRpl0Flag值等于rpl1_same_as_rpl0_flag的值。

如果rpl1_same_as_rpl0_flag的值为‘1’，则num_ref_pic_list_set[1]值等于num_ref_pic_list_set[0]值，且参考图像队列配置集reference_picture_list_set(1,rplsIndex)中每个语法元素的值与参考图像队列配置集reference_picture_list_set(0,rplsIndex)中对应语法元素的值相同，其中rplsIndex的取值范围为0～(num_ref_pic_list_set[0]-1)。

参考图像队列配置集数　num_ref_pic_list_set[0]、num_ref_pic_list_set[1]

表示参考图像队列配置集的数量。解析过程见8.3。取值范围为0～64。NumRefPicListSet[0]的值等于num_ref_pic_list_set[0]值。如果Rpl1SameAsRpl0Flag的值为1，则NumRefPicListSet[1]的值等于num_ref_pic_list_set[0]的值；否则NumRefPicListSet[1]的值等于num_ref_pic_list_set[1]值。

默认活跃参考图像数　num_ref_default_active_minus1[0]、num_ref_default_active_minus1[1]

表示解码图像时，参考图像队列中参考索引（pu_reference_index_l0、pu_reference_index_l1）默认的最大值。解析过程见8.3。取值范围为0～14。

最大编码单元尺寸　log2_lcu_size_minus2

3位无符号整数。表示最大编码单元的大小，取值范围为3～5。LcuSizeInBit的值等于log2_lcu_size_minus2的值加2。MaxQtSize的值等于$2^{LcuSizeInBit}$。

最小编码单元尺寸　log2_min_cu_size_minus2

2位无符号整数。表示最小编码单元的大小，取值范围为0～2。MinCuSize、MinBtSize和MinEqtSize的值均等于$2^{log2_min_cu_size_minus2+2}$。

划分单元最大比　log2_max_part_ratio_minus2

2位无符号整数。表示最大编码单元宽比高或者高比宽的比值，MaxPartRatio的值等于$2^{log2_max_part_ratio_minus2+2}$。

编码树最大划分次数　max_split_times_minus6

3位无符号整数。表示所允许的编码单元最大划分次数。四叉树、二叉树、扩展四叉树每划分一次，编码单元划分深度加1。MaxSplitTimes的值等于max_split_times的值加6。

最小四叉树尺寸　log2_min_qt_size_minus2

3位无符号整数。表示所允许的四叉树划分最小编码单元尺寸，取值范围为0～5。MinQtSize的值等于$2^{log2_min_qt_size_minus2+2}$。

最大二叉树尺寸　log2_max_bt_size_minus2

3位无符号整数。表示所允许的二叉树划分最大编码单元尺寸，取值范围为0～5。MaxBtSize的值等于$2^{log2_max_bt_size_minus2+2}$。

最大扩展四叉树尺寸　log2_max_eqt_size_minus3

2位无符号整数。表示所允许的扩展四叉树最大编码单元尺寸，取值范围为0～3。MaxEqtSize的值等于$2^{log2_max_eqt_size_minus3+3}$。

加权量化允许标志　weight_quant_enable_flag

二值变量。值为‘1’表示视频序列允许使用加权量化，值为‘0’表示视频序列不使用加权量化。WeightQuantEnableFlag的值等于weight_quant_enable_flag的值。

加权量化矩阵加载标志　load_seq_weight_quant_data_flag

二值变量。值为‘1’表示4×4和8×8变换块的加权量化矩阵按表16（见7.1.2.4）从序列头中加载；值为‘0’表示4×4和8×8变换块的加权量化矩阵应根据附录D确定。LoadSeqWeightQuantDataFlag的值等于load_seq_weight_quant_data_flag的值。如果位流中不存在load_seq_weight_quant_data_flag，则LoadSeqWeightQuantDataFlag的值等于0。

二次变换允许标志　st_enable_flag

二值变量。值为‘1’表示可使用ST，值为‘0’表示不应使用ST。StEnableFlag的值等于st_enable_flag的值。

样值偏移自适应补偿允许标志　sao_enable_flag

二值变量。值为‘1’表示可使用SAO，值为‘0’表示不应使用SAO。SaoEnableFlag的值等于sao_enable_flag的值。

跨分量样值偏移自适应补偿允许标志位　ccsao_enable_flag

二值变量。值为‘1’表示可使用跨分量样值偏移自适应补偿，值为‘0’表示不应使用跨分量样值偏移自适应补偿。CcsaoEnableFlag的值等于ccsao_enable_flag的值。如果位流中不存在ccsao_enable_flag，则CcsaoEnableFlag的值等于0。

自适应修正滤波允许标志　alf_enable_flag

二值变量。值为‘1’表示可使用ALF，值为‘0’表示不应使用ALF。AlfEnableFlag的值等于alf_enable_flag的值。

仿射运动补偿允许标志　affine_enable_flag

二值变量。值为‘1’表示可使用仿射运动补偿，值为‘0’表示不应使用仿射运动补偿。AffineEnableFlag的值等于affine_enable_flag的值。

仿射预测样本改善允许标志　asr_enable_flag

二值变量。值为‘1’表示可使用ASR，值为‘0’表示不应使用ASR。AsrEnableFlag的值等于asr_enable_flag的值。如果位流中不存在asr_enable_flag，则AsrEnableFlag的值为0。

对称运动矢量差模式允许标志　smvd_enable_flag

二值变量。值为‘1’表示可使用SMVD，值为‘0’表示不应使用SMVD。SmvdEnableFlag的值等于smvd_enable_flag的值。

脉冲编码调制模式允许标志　ipcm_enable_flag

二值变量。值为‘1’表示可使用脉冲编码调制模式，值为‘0’表示不应使用脉冲编码调制模式。IpcmEnableFlag的值等于ipcm_enable_flag的值。

自适应运动矢量精度允许标志　amvr_enable_flag

二值变量。值为‘1’表示可使用自AMVR，值为‘0’表示不应使用AMVR。AmvrEnableFlag的值等于amvr_enable_flag的值。

帧间预测候选历史运动信息数　num_of_hmvp_cand

4位无符号整数。NumOfHmvpCand的值等于num_of_hmvp_cand的值，取值范围为0～8。NumOfHmvpCand的值为0，表示进行帧间预测时不应使用历史运动信息。

帧内预测滤波允许标志　intra_pf_enable_flag

二值变量。值为‘1’表示可使用帧内预测滤波，值为‘0’表示不应使用帧内预测滤波。IntraPfEnableFlag的值等于intra_pf_enable_flag。

高级运动矢量表达模式允许标志　umve_enable_flag

二值变量。值为‘1’表示可使用UMVE，值为‘0’表示不应使用UMVE。UmveEnableFlag的值等于umve_enable_flag的值。

运动矢量精度扩展模式允许标志　emvr_enable_flag

二值变量。值为‘1’表示可使用EMVR，值为‘0’表示不应使用EMVR。EmvrEnableFlag的值等于emvr_enable_flag的值。

跨分量两步预测模式允许标志　tscpm_enable_flag

二值变量。值为‘1’表示可使用TSCPM，值为‘0’表示不应使用TSCPM。TscpmEnableFlag的值等于tscpm_enable_flag的值。

帧内衍生模式允许标志　dt_enable_flag

二值变量。值为‘1’表示可使用帧内衍生模式，值为‘0’表示不应使用帧内衍生模式。DtEnableFlag的值等于dt_enable_flag的值。

衍生模式待划分边长最大尺寸　log2_max_dt_size_minus4

2位无符号整数。表示所允许的衍生模式待划分边长的最大值。取值范围为0～2。DtMaxSize的值等于$2^{log2_max_dt_size_minus4+4}$。DtMinSize的值等于16。

基于位置的变换允许标志　pbt_enable_flag

二值变量。值为‘1’表示可使用PBT，值为‘0’表示不应使用PBT。PbtEnableFlag的值等于pbt_enable_flag的值。

重叠块运动补偿允许标志　obmc_enable_flag

二值变量。值为‘1’表示可使用OBMC，值为‘0’表示不应使用OBMC。ObmcEnableFlag的值等于obmc_enable_flag的值。如果位流中不存在obmc_enable_flag，则ObmcEnableFlag的值为0。

帧间预测滤波允许标志　inter_pf_enable_flag

二值变量。值为‘1’表示可使用帧间预测滤波；值为‘0’表示不应使用帧间预测滤波。InterPfEnableFlag的值等于inter_pf_enable_flag的值。如果位流中不存在inter_pf_enable_flag，则InterPfEnableFlag的值为0。

帧间预测修正允许标志　inter_pc_enable_flag

二值变量。值为‘1’表示可使用帧间预测修正，值为‘0’表示不应使用帧间预测修正。InterPcEnableFlag的值等于inter_pc_enable_flag的值。如果位流中不存在inter_pc_enable_flag，则InterPcEnableFlag的值为0。

隐择变换允许标志　ist_enable_flag

二值变量。值为‘1’表示可使用IST，值为‘0’表示不应使用IST。IstEnableFlag的值等于ist_enable_flag。如果位流中不存在ist_enable_flag，则IstEnableFlag的值为0。

块复制帧内预测允许标志　ibc_enable_flag

二值变量。值为‘1’表示可使用IBC；值为‘0’表示不应使用IBC。IbcEnableFlag的值等于ibc_enable_flag的值。如果位流中不存在ibc_enable_flag，则IbcEnableFlag的值为0。

串复制帧内预测允许标志　isc_enable_flag

二值变量。值为‘1’表示可使用ISC，值为‘0’表示不应使用ISC。IscEnableFlag的值等于isc_enable_flag的值。如果位流中不存在isc_enable_flag，则IscEnableFlag的值为0。

帧内预测候选历史运动信息数　num_of_intra_hmvp_cand

4位无符号整数。NumOfIntraHmvpCand的值等于num_of_intra_hmvp_cand的值，取值范围为0～12。NumOfIntraHmvpCand的值为0，表示进行帧内预测时不应使用历史运动信息。

基于频数信息的帧内编码允许标志　fimc_enable_flag

二值变量。值为‘1’表示可使用FIMC，值为‘0’表示不应使用FIMC。FimcEnableFlag的值等于fimc_enable_flag的值。如果位流中不存在fimc_enable_flag，则FimcEnableFlag的值等于0。

子块变换允许标志　sbt_enable_flag

二值变量。值为‘1’表示可使用SBT，值为‘0’表示不应使用SBT。SbtEnableFlag的值等于sbt_enable_flag的值。如果位流中不存在sbt_enable_flag，则SbtEnableFlag的值为0。

双向光流允许标志　bio_enable_flag

二值变量。值为‘1’表示可使用BIO，值为‘0’表示不应使用BIO。BioEnableFlag的值等于bio_enable_flag的值。如果位流中不存在bio_enable_flag，则BioEnableFlag的值为0。

解码端运动矢量改良允许标志　dmvr_enable_flag

二值变量。值为‘1’表示可使用DMVR，值为‘0’表示不应使用DMVR。DmvrEnableFlag的值等于dmvr_enable_flag的值。如果位流中不存在dmvr_enable_flag，则DmvrEnableFlag的值为0。

双向梯度修正允许标志　bgc_enable_flag

二值变量。值为‘1’表示可使用BGC，值为‘0’表示不应使用BGC。BgcEnableFlag的值等于bgc_enable_flag的值。如果位流中不存在bgc_enable_flag，则BgcEnableFlag的值为0。

增强的跨分量两步预测模式允许标志　enhanced_tscpm_enable_flag

二值变量。值为‘1’表示可使用增强的跨分量两步预测模式，值为‘0’表示不应使用增强的跨分量两步预测模式。EnhancedTscpmEnableFlag的值等于enhanced_tscpm_enable_flag的值。如果位流中不存在enhanced_tscpm_enable_flag，则EnhancedTscpmEnableFlag的值为0。

跨多分量预测允许标志　pmc_enable_flag

二值变量。值为‘1’表示可使用PMC，值为‘0’表示不应使用PMC。PmcEnableFlag的值等于pmc_enable_flag的值。如果位流中不存在pmc_enable_flag，则PmcEnableFlag的值为0。

改进型帧内预测允许标志　iip_enable_flag

二值变量。值为‘1’表示可使用IIP，值为‘0’表示不应使用IIP。IipEnableFlag的值等于iip_enable_flag。如果位流中不存在iip_enable_flag，则IipEnableFlag的值为0。

空域角度加权预测模式允许标志　sawp_enable_flag

二值变量。值为‘1’表示可使用SAWP，值为‘0’表示不应使用SAWP。SawpEnableFlag的值等于sawp_enable_flag的值。如果位流中不存在sawp_enable_flag，则SawpEnableFlag的值为0。

角度加权预测模式允许标志　awp_enable_flag

二值变量。值为‘1’表示可使用AWP，值为‘0’表示不应使用AWP。AwpEnableFlag的值等于awp_enable_flag的值。如果位流中不存在awp_enable_flag，则AwpEnableFlag的值为0。

增强样值偏移自适应补偿允许标志　esao_enable_flag

二值变量。值为‘1’表示可使用增强样值偏移自适应补偿，值为‘0’表示不应使用增强样值偏移自适应补偿。EsaoEnableFlag的值等于esao_enable_flag的值。如果位流中不存在esao_enable_flag，则EsaoEnableFlag的值为0。如果EsaoEnableFlag的值为1，则SaoEnableFlag的值应为0。

增强时域运动矢量预测和运动矢量角度预测允许标志　etmvp_mvap_enable_flag

二值变量。值为‘1’表示可使用增强时域运动矢量预测和运动矢量角度预测，值为‘0’表示不应使用增强时域运动矢量预测和运动矢量角度预测。EtmvpMvapEnableFlag的值等于etmvp_mvap_enable_flag的值。如果位流中不存在etmvp_mvap_enable_flag，则EtmvpMvapEnableFlag的值为0。如果EtmvpMvapEnableFlag的值为1，则NumOfMvapCand的值等于5；否则NumOfMvapCand的值等于0。

增强自适应修正滤波允许标志　ealf_enable_flag

二值变量。值为‘1’表示应使用EALF，值为‘0’表示不应使用EALF。EalfEnableFlag的值等于ealf_enable_flag的值。如果位流中不存在ealf_enable_flag，则EalfEnableFlag的值为0。

神经网络工具集　nn_tools_set_hook

8位无符号整数。标识是否使用神经网络的工具。如果位流中不存在nn_tools_set_hook，则nn_tools_set_hook的值应为‘00000000’。解码处理应该忽略这些位。

nn_tools_set_hook最低位的值为‘1’表示可使用神经网络滤波，nn_tools_set_hook最低位的值为‘0’表示不应使用神经网络滤波。NnFilterEnableFlag的值等于nn_tools_set_hook & 0x01的值。如果位流中不存在nn_tools_set_hook，则NnFilterEnableFlag的值等于0。其他位保留。

神经网络滤波模型数 num_of_nn_filter_minus1

表示神经网络滤波可使用的滤波模型数量。解析过程见8.2。NumOfNnFilter的值等于num_of_nn_filter_minus1的值加1。如果位流中不存在num_of_nn_filter_minus1，则NumOfNnFilter的值等于0。解码处理应该忽略这些位。

图像重排序延迟 output_reorder_delay

5位无符号整数。由于图像编解码顺序与显示顺序不一致带来的重排序延迟，以解码图像为单位。由于一幅解码图像的显示时间和progressive_sequence、progressive_frame、repeat_first_field、picture_structure等语法元素的值有关，所以这段时间的绝对长度是不固定的，但是在这段时间内显示的解码图像数是固定的。low_delay值为‘0’时，OutputReorderDelay的值等于output_reorder_delay的值；low_delay值为‘1’时，OutputReorderDelay的值为0。

跨片环路滤波允许标志 cross_patch_loop_filter_enable_flag

二值变量。值为‘1’时表示可跨越片边界进行去块效应滤波、样本偏移补偿及自适应修正滤波，值为‘0’表示不应跨越片边界进行去块效应滤波、样本偏移补偿及自适应修正滤波。CplfEnableFlag的值等于cross_patch_loop_filter_enable_flag的值。

片划分一致性标志 stable_patch_flag

二值变量。值为‘1’时表示当前视频序列中所有图像划分为片的方式均应相同，值为‘0’表示当前视频序列中图像划分为片的方式可不相同。

参考同位置片标志 ref_colocated_patch_flag

二值变量。值为‘1’时表示进行帧间预测时只使用参考图像同位置片边界内的采样值进行参考，值为‘0’表示进行帧间预测时可使用参考图像同位置片边界外的采样值进行参考。

统一片大小标志 uniform_patch_flag

二值变量。值为‘1’时表示除最右边和最下边的片外，图像中其他片的大小应相同；值为‘0’表示片的大小可不同。

片宽度 patch_width_minus1

片高度 patch_height_minus1

片的宽度和高度，以LCU为单位。patch_width_minus1的值应小于256，patch_height_minus1的值应小于144。图像中各个片以LCU为单位的宽度、高度和位置按以下方法获得：

```
tempW = patch_width_minus1 + 1
tempH = patch_height_minus1 + 1
LcuSize = 1 << LcuSizeInBit
PictureWidthInLcu = (horizontal_size + LcuSize -1) / LcuSize
PictureHeightInLcu = (vertical_size + LcuSize -1) / LcuSize
if (tempW> PictureWidthInLcu) {
    tempW = PictureWidthInLcu
}
if (tempH> PictureHeightInLcu){
    tempH = PictureHeightInLcu
```

```
}
if (tempW < Min(2, PictureWidthInLcu)) {
    tempW= Min(2, PictureWidthInLcu)
}
numPatchColumns = PictureWidthInLcu / tempW
numPatchRows = PictureHeightInLcu / tempH
for (i = 0; i < numPatchRows; i++) {
    for (j = 0; j < numPatchColumns; j++) {
        PatchSizeInLcu[i][j]->Width = patch_width_minus1 + 1
        PatchSizeInLcu[i][j]->Height= patch_height_minus1 + 1
        PatchSizeInLcu[i][j]->PosX = (patch_width_minus1 + 1) * j
        PatchSizeInLcu[i][j]->PosY= (patch_height_minus1 + 1) * i
    }
}
temp = PictureHeightInLcu % tempH
if (temp != 0) {
    for (j = 0; j < numPatchColumns; j++) {
        PatchSizeInLcu[numPatchRows][j]->Width += tempW
        PatchSizeInLcu[numPatchRows][j]->Height += temp
        PatchSizeInLcu[numPatchRows][j]->PosX += tempW * j
        PatchSizeInLcu[numPatchRows][j]->PosY += tempH * numPatchRows
    }
    numPatchRows++
}
temp = PictureWidthInLcu % tempW
if (temp != 0) {
    for (j = 0; j < numPatchRows; j++) {
        PatchSizeInLcu[j][numPatchColums-1]->Width += temp
    }
}
```

MinPatchWidthInPixel的值等于Min($2^{LcuSizeInBit+1}$, horizontal_size)。任何片以像素为单位的宽度不应小于MinPatchWidthInPixel的值。

EipmEnableFlag的值为0表示不应使用亮度帧内预测扩展模式,值为1表示应使用亮度帧内预测扩展模式。

MipfEnableFlag的值为0表示不应使用多组合帧内预测滤波,值为1表示应使用多组合帧内预测滤波。

IntraPfChromaEnableFlag的值为0表示不应使用帧内色度预测滤波，值为1表示应使用帧内色度预测滤波。

UmveEnhancementEnableFlag的值为0表示不应使用高级运动矢量表达增强模式，值为1表示可使用高级运动矢量表达增强模式。

AffineUmveEnableFlag的值为0表示不应使用仿射高级运动矢量表达模式,值为1表示可使用仿射高级运动矢量表达模式。

SbTmvpEnableFlag的值为0表示不应使用子块时域运动信息预测，值为1表示应使用子块时域运动信息预测。

SrccEnableFlag的值为0表示不应使用基于扫描区域的系数编码，值为1表示应使用基于扫描区域的系数编码。

MaecEnableFlag的值为0表示不应使用多假设概率模型高级熵编码，值为1表示应使用多假设概率模型高级熵编码。

EnhancedStEnableFlag的值为0表示不应使用增强二次变换，值为1表示应使用增强二次变换。

EnhancedTscpmEnableFlag的值为0表示不应使用增强的跨分量两步预测模式，值为1表示可使用增强的跨分量两步预测模式。

7.2.2.3 参考图像队列配置集

参考知识图像标志 reference_to_library_enable_flag

二值变量。值为‘1’表示当前参考图像队列配置集中可有参考图像是知识图像，值为‘0’表示当前参考图像队列配置集的所有参考图像均不应是知识图像。ReferenceToLibraryEnableFlag 的值等于 reference_to_library_enable_flag 的值。如果位流中不存在 reference_to_library_enable_flag，则 ReferenceToLibraryEnableFlag 的值等于 0。

知识图像索引标志 library_index_flag[list][rpls][i]

二值变量。值为‘1’表示索引为 list 的参考图像队列配置集中的第 rpls 个参考图像队列中第 i 幅参考图像是知识图像，值为‘0’表示索引为 list 的参考图像队列配置集中的第 rpls 个参考图像队列中第 i 幅参考图像不是知识图像。LibraryIndexFlag[list][rpls][i] 的值等于 library_index_flag[list][rpls][i]的值。如果位流中不存在 library_index_flag[list][rpls][i]，则 LibraryIndexFlag[list][rpls][i]的值为 0。

被参考的知识图像索引 referenced_library_picture_index[list][rpls][i]

表示参考图像队列 list 的第 rpls 个参考图像配置集中第 i 幅参考图像的知识图像索引。取值范围为 0～511。ReferencedLibraryPictureIndex[list][rpls][i] 的值等于 referenced_library_picture_index[list][rpls][i]的值。

参考图像数 num_of_ref_pic[list][rpls]

表示参考图像配置集中参考图像的数量。解析过程见8.3。NumOfRefPic[list][rpls]的值等于 num_of_ref_pic[list][rpls]的值。num_of_ref_pic的值应小于MaxDpbSize的值。

参考图像DOI差值绝对值 abs_delta_doi[list][rpls][i]

如果第i幅参考图像是参考图像队列配置集中出现的第一幅非知识图像的参考图像，表示当前图像的DOI与该参考图像的DOI的差值的绝对值且abs_delta_doi[list][rpls][i]的值不应为0；否则，表示在参考图像队列list中，离该参考图像最近的前一幅非知识图像的参考图像的DOI与该参考图像的DOI的差值的绝对值。解析过程见8.3。取值范围为0～2^8-1。

参考图像DOI差值符号 sign_delta_doi[list][rpls][i]

二值变量。值为‘0’表示DeltaDoi[list][rpls][i]的值等于abs_delta_doi[list][rpls][i]，值为‘1’表示DeltaDoi[list][rpls][i]的值等于-abs_delta_doi[list][rpls][i]。

如果RpllSameAsRpl0Flag的值为1，进行以下操作：

——将 NumOfRefPic[1][rpls]的值设置为 NumOfRefPic[0][rpls]的值，rpls 的取值范围为 0～(NumRefPicListSet[0]-1)；

——将 DeltaDoi[1][rpls][i]的值设置为 DeltaDoi[0][rpls][i]的值，rpls 的取值范围为 0～(NumRefPicListSet[0]-1)，i 的取值范围为 0～NumOfRefPic[1][rpls]。

7.2.2.4 自定义加权量化矩阵

加权量化矩阵系数 weight_quant_coeff

加权量化矩阵的系数,取值范围应为1～255。WeightQuantCoeff的值等于weight_quant_coeff的值。

7.2.2.5 扩展和用户数据

7.2.2.5.1 扩展数据

视频扩展起始码 extension_start_code

位串‘0x000001B5’。标识视频扩展数据的开始。

视频扩展数据保留字节 reserved_extension_data_byte

8位无符号整数。保留位。解码器应丢弃这些数据。视频扩展数据保留字节中不应出现从任意字节对齐位置开始的21个以上连续的‘0’。

7.2.2.5.2 用户数据

用户数据起始码 user_data_start_code

位串‘0x000001B2’。标识用户数据的开始。用户数据连续存放，直到下一个起始码。

用户数据字节 user_data

8位整数。用户数据的含义由用户自行定义。用户数据中不应出现从任意字节对齐位置开始的21个以上连续的‘0’。

7.2.2.6 序列显示扩展

本文件不定义显示过程。这一扩展中的信息对解码过程没有影响，解码器可忽略这些信息。

视频扩展标号 extension_id

位串‘0010’。标识序列显示扩展。

视频格式 video_format

3位无符号整数。说明视频在按本文件进行编码之前的格式，应符合表49的规定。如果位流中没有出现序列显示扩展，可假设视频格式为“未作规定的视频格式”。

表49 视频格式

video_format的值	含义
000	分量信号
001	PAL
010	NTSC
011	SECAM
100	禁止
101	未作规定的视频格式
110	保留
111	保留

样值范围 sample_range

二值变量。说明亮度和色度信号样值的范围。如果位流中没有出现序列显示扩展，设sample_range为‘0’。

彩色信息描述 colour_description

二值变量。值为‘1’表示位流中有colour_primaries、transfer_characteristics和matrix_coefficients，值为‘0’表示位流中没有colour_primaries、transfer_characteristics和matrix_coefficients。

彩色三基色 colour_primaries

8位无符号整数。说明源图像三基色的色度坐标，应符合表50的规定。

表50 彩色三基色

colour_primaries的值	彩色三基色		
0	禁止		
1	基色	*x*	*y*
	绿	0.300	0.600
	蓝	0.150	0.060
	红	0.640	0.330
	白D65	0.3127	0.3290 [a]
2	未作规定的视频 图像特性未知		
3	保留		
4	基色	*x*	*y*
	绿	0.21	0.71
	蓝	0.14	0.08
	红	0.67	0.33
	白 C	0.310	0.316 [b]
5	基色	*x*	*y*
	绿	0.29	0.60
	蓝	0.15	0.06
	红	0.64	0.33
	白 D65	0.313	0.329 [c]
6	基色	*x*	*y*
	绿	0.310	0.595
	蓝	0.155	0.070
	红	0.630	0.340
	白D65	0.3127	0.3290 [d]
7	基色	*x*	*y*
	绿	0.310	0.595
	蓝	0.155	0.070
	红	0.630	0.340
	白D65	0.3127	0.3290 [e]
8	普通胶片（彩色滤光镜，C光源）		
	基色	*x*	*y*
	绿	0.243	0.692(Wratten 58)
	蓝	0.145	0.049(Wratten 47)
	红	0.681	0.319(Wratten 25)
	白C	0.310	0.316

表 50（续）

colour_primaries的值	彩色三基色		
9	基色	x	y
	绿	0.170	0.797
	蓝	0.131	0.046
	红	0.708	0.292
	白D65	0.3127	0.3290 [f]
10～255	保留		

[a] GY/T 155—2000 中规定的三基色。

[b] ITU-R BT.470 System M 中规定的三基色。

[c] ITU-R BT.470 System B，G 中规定的三基色。

[d] SMPTE 170M 中规定的三基色。

[e] SMPTE 240M 中规定的三基色。

[f] GB/T 41809—2022 中规定的三基色。

如果位流中没有序列显示扩展，或者colour_description的值是‘0’，假设色度已由应用本身隐含定义。

光电转移特性 transfer_characteristics

8位无符号整数。说明源图像的光电转移特性，应符合表51的规定。表51中Lc是线性输入信号，与光的强度成正比；E'是输出的电信号。

表51 光电转移特性

transfer_characteristics的值	光电转移特性
0	禁止
1	$E' = 1.099\ Lc^{0.45} - 0.099,\ 0.018 \leqslant Lc \leqslant 1.33$ $E' = 4.500\ Lc,\ -0.0045 \leqslant Lc < 0.018$ $E' = -\{1.099(-4Lc)^{0.45}-0.099\}/4,\ -0.25 \leqslant Lc < -0.0045$ [a]
2	未作规定的视频 图像特性未知
3	保留
4	假设显示伽玛值为2.2 [b]
5	假设显示伽玛值为2.8 [c]
6	$E' = 1.099\ Lc^{0.45} - 0.099,\ 0.018 \leqslant Lc \leqslant 1$ $E' = 4.500\ Lc,\ 0 \leqslant Lc < 0.018$ [d]
7	$E' = 1.1115\ Lc^{0.45} - 0.1115,\ Lc \geqslant 0.0228$ $E' = 4.0\ Lc,\ Lc < 0.0228$ [e]
8	线性转移特性，即E′ = Lc
9	对数转移特性（范围为100:1）： $E' = 1.0-(\lg(Lc))/2,\ 0.01 \leqslant Lc \leqslant 1$ $E' = 0.0,\ Lc < 0.01$

表 51（续）

transfer_characteristics的值	光电转移特性
10	对数转移特性（范围为316.22777:1）： E' = 1.0-(lg (Lc))/2.5，0.0031622777≤ Lc ≤ 1 E' = 0.0，Lc <0.0031622777
11	E' = 1.0993 $Lc^{0.45}$ - 0.0993，0.0181 ≤ Lc ≤ 1 E' = 4.500 Lc，0≤ Lc < 0.0181 [f]
12	E' = ((c_1 + c_2 Lc^n) ÷ (1 + c_3 Lc^n))^m c_1 = c_3 − c_2 + 1 = 3424 ÷ 4096 = 0.8359375 c_2 = 32 * 2413 ÷ 4096 = 18.8515625 c_3 = 32 * 2392 ÷ 4096 = 18.6875 m = 128 * 2523 ÷ 4096 = 78.84375 n = 0.25 * 2610 ÷ 4096 = 0.1593017578125 Lc等于1对应于显示亮度为10000cd/m^2 [g]
13	保留
14	E' = 3 * $Lc^{0.5}$，0≤ Lc ≤1/12 E' = a * ln (12 * Lc - b) + c，1/12< Lc ≤ 1 a = 0.17883277 b = 0.28466892 c = 0.55991073 [h]
15～255	保留
[a] GY/T 155—2000 中规定的光电转移特性。 [b] ITU-R BT.470 System M 中规定的光电转移特性。 [c] ITU-R BT.470 System B，G 中规定的光电转移特性。 [d] ITU-R BT.709，SMPTE 170M 和 GB/T 41809—2022 的 10bit 系统中规定的光电转移特性。 [e] SMPTE 240M 中规定的光电转移特性。 [f] GB/T 41809—2022 中 12bit 系统规定的光电转移特性。 [g] GB/T 41808—2022 中规定的 PQ 光电转移特性。 [h] GB/T 41808—2022 中规定的 HLG 光电转移特性。	

如果位流中没有序列显示扩展，或者colour_description的值是‘0’，则假设光电转移特性已由应用本身隐含定义。

彩色信号转换矩阵系数　matrix_coefficients

8位无符号整数。说明从红绿蓝三基色转换为亮度和色度信号时采用的转换矩阵，应符合表52的规定。

表52　彩色信号转换矩阵系数

matrix_coefficients的值	彩色信号转换矩阵
0	禁止
1	E'_Y = 0.2126 E'_R + 0.7152 E'_G + 0.0722 E'_B E'_{CB} = (E'_B− E'_Y)/1.8556 E'_{CR} = (E'_R− E'_Y)/1.5748 [a]

表 52（续）

matrix_coefficients的值	彩色信号转换矩阵
2	未作规定的视频 图像特性未知
3	保留
4	$E'_Y = 0.59\ E'_G + 0.11\ E'_B + 0.30\ E'_R$ $E'_{CB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$ $E'_{CR} = -0.421 E'_G - 0.079\ E'_B + 0.500\ E'_R$ [b]
5	$E'_Y = 0.587\ E'_G + 0.114\ E'_B + 0.299\ E'_R$ $E'_{CB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$ $E'_{CR} = -0.419\ E'_G - 0.081\ E'_B + 0.500\ E'_R$ [c]
6	$E'_Y = 0.587\ E'_G + 0.114\ E'_B + 0.299\ E'_R$ $E'_{CB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$ $E'_{CR} = -0.419\ E'_G - 0.081\ E'_B + 0.500\ E'_R$ [d]
7	$E'_Y = 0.701\ E'_G + 0.087\ E'_B + 0.212\ E'_R$ $E'_{CB} = -0.384\ E'_G + 0.500\ E'_B - 0.116\ E'_R$ $E'_{CR} = -0.445\ E'_G - 0.055\ E'_B + 0.500\ E'_R$ [e]
8	$E'_Y = 0.2627\ E'_R + 0.6780\ E'_G + 0.0593\ E'_B$ $E'_{CB} = (E'_B - E'_Y)/1.8814$ $E'_{CR} = (E'_R - E'_Y)/1.4746$ [f]
9	$E'_Y = (0.2627\ E_R + 0.6780\ E_G + 0.0593\ E_B)'$ $E'_{CB} = \begin{cases} \dfrac{E'_B - E'_Y}{-2N_B}, & N_B \le E'_B - E'_Y \le 0 \\ \dfrac{E'_B - E'_Y}{2P_B}, & 0 \le E'_B - E'_Y \le P_B \end{cases}$ $E'_{CR} = \begin{cases} \dfrac{E'_R - E'_Y}{-2N_R}, & N_R \le E'_R - E'_Y \le 0 \\ \dfrac{E'_R - E'_Y}{2P_R}, & 0 \le E'_R - E'_Y \le P_R \end{cases}$ 其中， $P_B = 0.7910$，$N_B = -0.9702$ $P_R = 0.4969$，$N_R = -0.8591$ [g]
10～255	保留
表中E'_Y是值在0和1之间的模拟量，E'_{CB}和E'_{CR}是值在-0.5和0.5之间的模拟量，E'_R、E'_G和E'_B是值在0和1之间的模拟量。	

[a] GY/T 155—2000 中规定的彩色信号转换矩阵。

[b] FCC 中规定的彩色信号转换矩阵。

[c] ITU-R BT.470 System B，G 中规定的彩色信号转换矩阵。

[d] SMPTE 170M 中规定的彩色信号转换矩阵。

[e] SMPTE 240M 中规定的彩色信号转换矩阵。

[f] GB/T 41809—2022 中规定的非恒定亮度系统。

[g] GB/T 41809—2022 中规定的恒定亮度系统。

如果sample_range的值为‘0’，Y、Cb和Cr与E'_Y、E'_{CB}和E'_{CR}的关系如下：

$$Y = (219 * 2^{BitDepth-8} * E'_Y) + 2^{BitDepth-4}$$
$$Cb = (224 * 2^{BitDepth-8} * E'_{CB}) + 2^{BitDepth-1}$$
$$Cr = (224 * 2^{BitDepth-8} * E'_{CR}) + 2^{BitDepth-1}$$

如果sample_range的值为‘1’，Y、Cb和Cr与E'_Y、E'_{CB}和E'_{CR}的关系如下：

$$Y = ((2^{BitDepth} - 1) * E'_Y)$$
$$Cb = ((2^{BitDepth} - 1) * E'_{CB}) + 2^{BitDepth-1}$$
$$Cr = ((2^{BitDepth} - 1) * E'_{CR}) + 2^{BitDepth-1}$$

其中BitDetph是编码样本精度。例如：BitDepth = 8，sample_range的值为‘0’时：

$$Y = (219 * E'_Y) + 16$$
$$Cb = (224 * E'_{CB}) + 128$$
$$Cr = (224 * E'_{CR}) + 128$$

Y的取值范围为16～235，Cb和Cr的取值范围为16～240。

BitDepth = 8，sample_range的值为‘1’时：

$$Y = (255 * E'_Y)$$
$$Cb = (255 * E'_{CB}) + 128$$
$$Cr = (255 * E'_{CR}) + 128$$

Y、Cb和Cr的取值范围都为0～255。

注1：本文件规定的解码过程将输出的 Y、Cb 和 Cr 的样值范围限制在 0～$2^{BitDepth}$-1 之间。如果位流中没有出现序列显示扩展，或者 colour_description 的值是‘0’，假设转换矩阵已由应用本身隐含定义。

注2：某些应用可能有多个不同的视频信号，而不同的视频信号又可能具有不同的彩色三基色、转移特性和/或转换矩阵。在这种情况下应用首先将这些不同的参数集转换到一个统一的参数集。

水平显示尺寸　display_horizontal_size

垂直显示尺寸　display_vertical_size

display_horizontal_size和display_vertical_size都是14位无符号整数。它们共同定义了一个矩形，如果该矩形的尺寸比编码图像的尺寸小，宜只显示编码图像的一部分；如果该矩形的尺寸比编码图像的尺寸大，宜只在显示设备的一部分上显示重建图像。

display_horizontal_size的单位应是编码图像每行样本数。display_vertical_size的单位应是编码图像的行数。

display_horizontal_size和display_vertical_size对解码过程没有影响。它们可被显示过程使用。本文件不定义显示过程。

立体视频模式标志　td_mode_flag

二值变量。值为‘0’表示当前视频序列是单目视频，值为‘1’表示当前视频序列包含多个视点或深度信息。

立体视频拼接模式　td_packing_mode

8位无符号整数。规定立体视频的拼接模式，应符合表53的规定。

表53　立体视频拼接模式

td_packing_mode的值	含义
0	左右拼接
1	上下拼接
2	四视点拼接
3～255	保留

表53中左右拼接表示当前视频序列为左右拼接双目立体视频，在解码图像中，左视点的图像的像素均在右视点的图像的像素的左侧；上下拼接表示当前视频序列为上下拼接双目立体视频，在解码图像中，左视点的图像的像素均在右视点的图像的像素的上侧；四视点拼接表示当前视频序列包含4个视点的图像，在解码图像中，从右到左4个视点图像的像素依次位于左上、右上、左下、右下的位置。

视点反转标志　view_reverse_flag

二值变量。值为‘0’表示视点顺序不变，值为‘1’表示视点顺序反转。

7.2.2.7　时域可伸缩扩展

视频扩展标号　extension_id

位串‘0011’。标识时域可伸缩扩展。

时间层数量　num_of_temporal_level_minus1

3位无符号整数。表示视频序列包含的时间层数量。时间层数量等于视频序列中所有图像的最高时间层标识的值与最低时间层标识的值之差加1。时间层数量的值不应该超过MAX_TEMPORAL_ID的值。

时间层帧率代码　temporal_frame_rate_code[i]

4位无符号整数。表示截取到时间层标识值等于i时的帧率，见表48。i表示视频序列中所有图像的最高时间层标识。

时间层比特率低位　temporal_bit_rate_lower[i]

18位无符号整数。表示截取到时间层标识值等于i时的BitRate的低18位。i表示视频序列中所有图像的最高时间层标识。

时间层比特率高位　temporal_bit_rate_upper[i]

12位无符号整数。表示截取到时间层标识值等于i时的BitRate的高12位。i表示视频序列中所有图像的最高时间层标识。

7.2.2.8　版权扩展

视频扩展标号　extension_id

位串‘0100’。标识版权扩展。

版权标志　copyright_flag

二值变量。值为‘1’表示该版权扩展定义的版权信息有效期直到下一个版权扩展或视频序列结束码，值为‘0’表示该版权扩展没有定义版权信息。

版权信息由copyright_id和CopyrightNumber进一步说明。

版权标号　copyright_id

8位无符号整数。版权所有者的代码，由版权注册机构统一分配。如果为‘0’，表示没有相关版权信息。

如果copyright_id的值为‘0’，CopyrightNumber应为0。

如果copyright_flag的值为‘0’，copyright_id应为‘0’。

原创或拷贝　original_or_copy

二值变量。值为‘1’表示源视频的内容是原创的，值为‘0’表示源视频的内容是拷贝的。

版权号1　copyright_number_1

20位无符号整数。CopyrightNumber的第44到第63位。

版权号2　copyright_number_2

22位无符号整数。CopyrightNumber的第22到第43位。

版权号3　copyright_number_3

22位无符号整数。CopyrightNumber的第0到第21位。

CopyrightNumber是64位无符号整数：

```
CopyrightNumber = (copyright_number_1 << 44) + (copyright_number_2 << 22) + copyright_number_3
```

如果copyright_flag的值是‘1’，CopyrightNumber和该版权扩展说明的源视频内容一一对应，CopyrightNumber为0说明没有相关信息；如果copyright_flag的值是‘0’，CopyrightNumber也应为0。

7.2.2.9　高动态范围图像元数据扩展

视频扩展标号　extension_id

位串‘0101’。标识高动态范围图像元数据扩展。

高动态范围图像元数据类型　hdr_dynamic_metadata_type

4位无符号整数。标识动态元数据类型。hdr_dynamic_metadata_type的值为5时元数据类型应符合GY/T 358—2022的规定，其他值保留。

视频扩展数据字节　extension_data_byte

8位无符号整数。视频扩展数据字节中不应出现从任意字节对齐位置开始的21个以上连续的‘0’。

7.2.2.10　目标显示设备和内容元数据扩展

视频扩展标号　extension_id

位串‘1010’。标识目标显示设备和内容元数据扩展。

显示设备三基色X坐标，显示设备三基色Y坐标　display_primaries_x[c], display_primaries_y[c]

16位无符号整数。分别表示归一化后的显示设备三基色的色度x坐标和y坐标。该坐标符合ISO 11664-1:2019中规定的CIE 1931，以0.00002为单位，范围为0～50000。c的值为0、1、2分别对应于绿、蓝、红三色。

显示设备标准白光X坐标，显示设备标准白光Y坐标　white_point_x, white_point_y

16位无符号整数。分别表示归一化后的显示设备标准白光的色度x坐标和y坐标。该坐标符合ISO/CIE 11664-1:2019、ISO/CIE 11664-3:2019中规定的CIE 1931，以0.00002为单位，范围为0～50000。

显示设备最大显示亮度　max_display_mastering_luminance

16位无符号整数。表示显示设备的最大显示亮度。以1cd/m^2为单位，范围为1cd/m^2～65535cd/m^2。

显示设备最小显示亮度　min_display_mastering_luminance

16位无符号整数。表示显示设备的最小显示亮度。以0.0001cd/m^2为单位，范围为0.0001cd/m^2～6.5535cd/m^2。max_display_mastering_luminance的值应大于min_display_mastering_luminance的值。

显示内容最大亮度　max_content_light_level

16位无符号整数。表示显示内容的最大亮度。以1cd/m^2为单位，范围为1cd/m^2～65535cd/m^2。max_content_light_level的值为某一显示内容的所有显示图像的最大亮度PictureMaxLightLevel的最大值。显示图像最大亮度PictureMaxLightLevel计算如下。

——对显示图像有效显示区域内的所有像素依次计算像素的 R、G、B 分量的最大值 maxRGB。有效显示区域是由 display_horizontal_size 和 display_vertical_size 共同定义的矩形区域：

1） 将像素的非线性(R',G',B')值转换为线性(R,G,B)值，并校准为以 1cd/m^2为单位的值；

2） 由像素校准后的(R,G,B)值，计算得到像素 R、G、B 分量的最大值 maxRGB。

——显示图像的 PictureMaxLightLevel 等于有效显示区域内的所有像素的 maxRGB 中的最大值。

显示内容最大图像平均亮度 max_picture_average_light_level

16位无符号整数。表示显示内容的最大图像平均亮度。以1cd/m^2为单位，范围为1cd/m^2～65535cd/m^2。max_picture_average_light_level 的值为某一显示内容的所有显示图像的图像平均亮度PictureAverageLightLevel的最大值。显示图像平均亮度PictureAverageLightLevel计算如下。

——对显示图像有效显示区域内的所有像素依次计算像素的 R、G、B 分量的最大值 maxRGB。有效显示区域是由 display_horizontal_size 和 display_vertical_size 共同定义的矩形区域：

1） 将像素的非线性(R',G',B')值转换为线性(R,G,B)值，并校准为以 1cd/m^2为单位的值；

2） 由像素校准后的(R,G,B)值，计算得到像素 R、G、B 分量的最大值 maxRGB。

——显示图像的 PictureAverageLightLevel 等于有效显示区域内的所有像素的 maxRGB 的平均值。

7.2.2.11 摄像机参数扩展

摄像机垂直视角和坐标系统示意图见图7和图8。

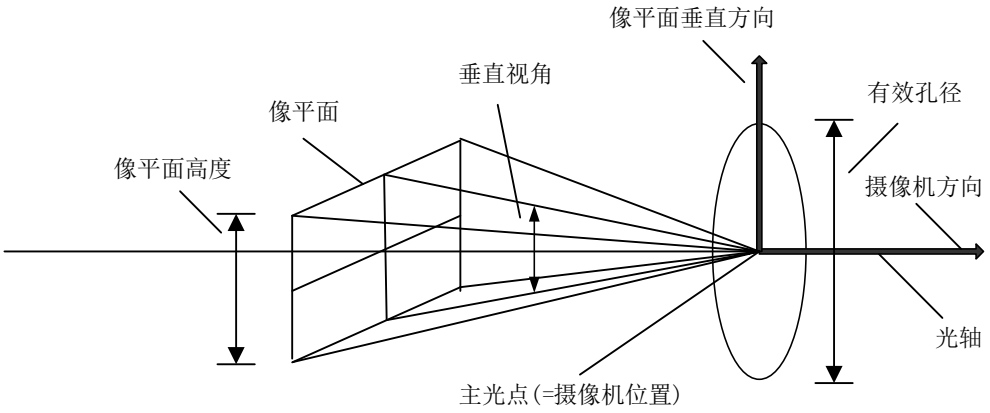


图7 垂直视角示意图

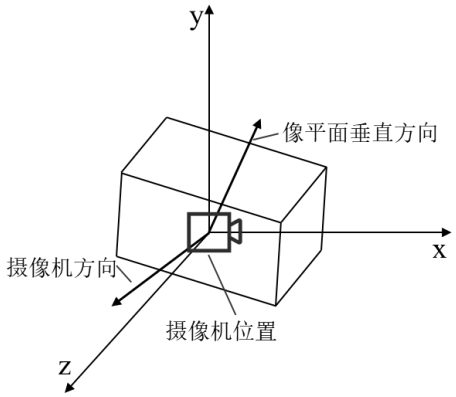


图8　摄像机坐标系统示意图

视频扩展标号　extension_id

位串‘1011’。标识摄像机参数扩展。

摄像机标号　camera_id

7位无符号整数。摄像机标识符。

图像设备高度　height_of_image_device

22位无符号整数。给出图像设备的高度，以0.001mm为单位，范围为0mm～4194.303mm。

焦距　focal_length

22位无符号整数。给出摄像机的焦距，以0.001mm为单位，范围为0mm～4194.303mm。

光圈　f_number

22位无符号整数。给出摄像机的光圈（光圈 = 焦距 ÷ 镜头的有效孔径），以0.001为单位，范围为0～4194.303。

垂直视角　vertical_angle_of_view

22位无符号整数。给出由图像设备顶端和底端决定的垂直视角，以0.0001°为单位，范围为0°～180°。

摄像机坐标X高位，摄像机坐标Y高位，摄像机坐标Z高位　camera_position_x_upper, camera_position_y_upper, camera_position_z_upper

分别表示CameraPositionX，CameraPositionY和CameraPositionZ的高16位。

摄像机坐标X低位，摄像机坐标Y低位，摄像机坐标Z低位　camera_position_x_lower, camera_position_y_lower, camera_position_z_lower

分别表示CameraPositionX，CameraPositionY和CameraPositionZ的低16位。

CameraPositionX，CameraPositionY和CameraPositionZ是一组32位整数，用2的补码表示。说明摄像机光学原点在由用户定义的全局坐标系中的坐标值，每个坐标值都以0.001mm为单位，范围为-2,147,483.648mm～2,147,483.647mm。

摄像机方向矢量X，摄像机方向矢量Y，摄像机方向矢量Z　camera_direction_x, camera_direction_y, camera_direction_z

22位整数，用2的补码表示。说明摄像机的方向，每个值范围为-2,097,152～2,097,151。摄像机的方向用从摄像机光学原点到摄像机前面位于摄像机光轴上的某点的矢量表示。

图像平面垂直矢量X，图像平面垂直矢量Y，图像平面垂直矢量Z　image_plane_vertical_x, image_plane_vertical_y, image_plane_vertical_z

22位整数，用2的补码表示。说明摄像机向上的方向，每个值的范围为-2,097,152～2,097,151。摄像机向上的方向用平行于设备的边缘，方向从底到顶的矢量表示。

7.2.2.12 ROI参数扩展

感兴趣区域参数扩展标号 extension_id

位串‘1100’。标识ROI参数扩展。

ROIInfo数组存储了当前图像的ROI信息,其中ROIInfo[i]表示当前图像第i个ROI的信息。ROIInfo[i]为一结构体，其中每个变量包括asisx、asisy、width和height四个参数。ROIInfo[i]->asisx表示第i个ROI的左上角区域在图像中的横坐标位置,ROIInfo[i]->asisy表示第i个ROI的左上角区域在图像中的纵坐标位置，ROIInfo[i]->width表示第i个ROI在图像中的宽度，ROIInfo[i]->width的单位为图像的列数，ROIInfo[i]->height表示第i个ROI在图像中的高度，ROIInfo[i]->height的单位为图像的行数。

PrevROIInfo数组存储了当前图像解码顺序前一幅图像的ROI信息,其中PrevROIInfo[i]表示当前图像解码顺序前一幅图像第i个ROI的信息。PrevROIInfo[i]为一结构体,其中每个变量包括asisx、asisy、width和height四个参数。PrevROIInfo[i]->asisx表示第i个ROI的左上角区域在图像中的横坐标位置，PrevROIInfo[i]->asisy表示第i个ROI的左上角区域在图像中的纵坐标位置，PrevROIInfo[i]->width表示第i个ROI在图像中的宽度,PrevROIInfo[i]->width的单位为图像的列数,PrevROIInfo[i]->height表示第i个ROI在图像中的高度，PrevROIInfo[i]->height的单位为图像的行数。

当前图像感兴趣区域数 current_picture_roi_num

8位无符号整数。表示当前图像的ROI的个数。

前一幅图像感兴趣区域数 prev_picture_roi_num

8位无符号整数。表示当前图像按解码顺序前一幅图像的ROI的个数。PrevPictureROINum的值等于prev_picture_roi_num的值。如果位流中不存在prev_picture_roi_num,则PrevPictureROINum的值为0。

跳过模式感兴趣区域数 roi_skip_run

表示ROI被标记为跳过模式的个数。当ROI被标记为跳过模式时，ROI的信息由skip_roi_mode导出。roi_skip_run的值应小于256。

感兴趣区域跳过模式 skip_roi_mode[i+j]

1位无符号整数。值为0表示当前图像按解码顺序前一幅图像中第i+j个ROI在当前图像中已经不存在。值为1表示当前图像按解码顺序前一幅图像中ROI参数为ROIInfo[i+j]的ROI与当前图像标号为roiIndex的ROI的参数相同。

感兴趣区域横坐标增量 roi_axisx_delta

表示一个ROI左上角在图像中的横坐标的增量。

感兴趣区域纵坐标增量 roi_axisy_delta

表示一个ROI左上角在图像中的纵坐标的增量。

感兴趣区域宽度增量 roi_width_delta

表示一个ROI在图像中的宽度的增量。

感兴趣区域高度增量 roi_height_delta

表示一个ROI在图像中的高度的增量。

roi_axisx_delta、roi_axisy_delta、roi_width_delta和roi_height_delta的单位是像素，用于获得ROIInfo[roiIndex]->asisx、ROIInfo[roiIndex]->asisy、ROIInfo[roiIndex]->width和ROIInfo[roiIndex]->height。ROIInfo[roiIndex]->width的值不应大于horizontal_size，ROIInfo[roiIndex]->height的值不应大于vertical_size，ROIInfo[roiIndex]->asisx+ROIInfo[roiIndex]->width的值不应大于horizontal_size，ROIInfo[roiIndex]->asisy+ROIInfo[roiIndex]->height的值不应大于vertical_size。

感兴趣区域横坐标 roi_axisx

表示一个ROI左上角在图像中的横坐标。

感兴趣区域纵坐标 roi_axisy

表示一个ROI左上角在图像中的纵坐标。

感兴趣区域宽度 roi_width

表示一个ROI在图像中的宽度。

感兴趣区域高度 roi_height

表示一个ROI在图像中的高度。

roi_axisx、roi_axisy、roi_width和roi_height的单位是像素，用于获得ROIInfo[roiIndex]->asisx、ROIInfo[roiIndex]->asisy、ROIInfo[roiIndex]->width和ROIInfo[roiIndex]->height。ROIInfo[roiIndex]->width的值不应大于horizontal_size，ROIInfo[roiIndex]->height的值不应大于vertical_size，ROIInfo[roiIndex]->asisx+ROIInfo[roiIndex]->width的值不应大于horizontal_size，ROIInfo[roiIndex]->asisy+ROIInfo[roiIndex]->height的值不应大于vertical_size。

7.2.2.13 参考知识图像扩展

视频扩展标号 extension_id

位串‘1101’。标识参考知识图像扩展。

后续主位流图像参考的外部知识图像数 crr_lib_number

3位无符号整数。表示位流中从当前位置到下一个序列头之间的主位流图像参考的外部知识图像数。值为‘001’表示后续主位流图像参考的外部知识图像数为1；值‘000’和‘010’~‘111’保留。

外部知识图像的索引编号 crr_lib_pid[i]

9位无符号整数，取值范围为0～511。用于描述位流中从当前位置到下一个序列头之间的主位流图像所参考的外部知识图像的索引编号。

7.2.2.14 自适应帧内刷新参数扩展

自适应帧内刷新参数扩展标号 extension_id

位串‘1110’。标识自适应帧内刷新参数扩展。

当前图像可划分为已刷新区域和未刷新区域（见图9），已刷新区域仅利用当前图像或其他图像已刷新区域进行解码，未刷新区域可利用当前图像或其他图像进行解码。

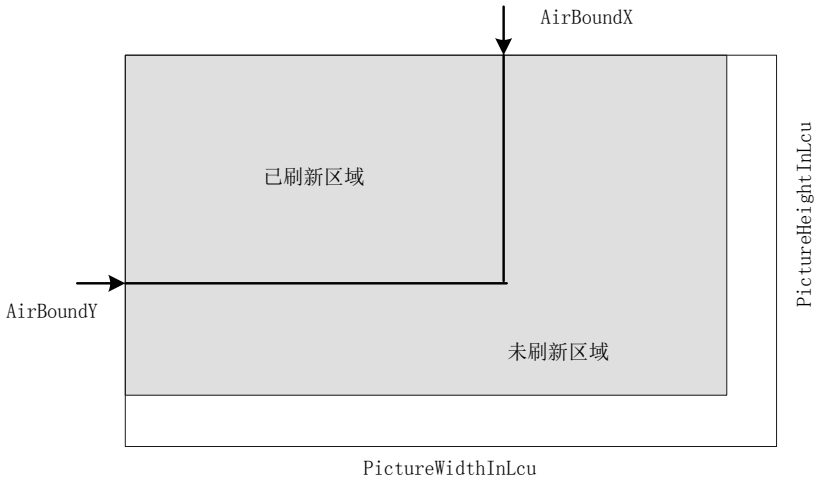


图9　自适应帧内刷新区域划分

自适应帧内刷新边界横坐标　air_bound_x

10位无符号整数。表示已刷新区域和未刷新区域的边界在图像中的横坐标，以LCU的宽度为单位。AirBoundX的值等于air_bound_x的值。如果位流中不存在air_bound_x，则AirBoundX的值为0。AirBoundX的值应小于或等于PictureWidthInLcu。

自适应帧内刷新边界纵坐标　air_bound_y

10位无符号整数。表示已刷新区域和未刷新区域的边界在图像中的纵坐标，以LCU的高度为单位。AirBoundY的值等于air_bound_y的值。如果位流中不存在air_bound_y，则AirBoundY的值为0。AirBoundY的值应小于或等于PictureHeightInLcu。

7.2.3　图像语义

7.2.3.1　帧内预测图像头

帧内预测图像头中compIndex等于0表示亮度分量，等于1表示Cb分量，等于2表示Cr分量。

帧内预测图像起始码　intra_picture_start_code

位串‘0x000001B3’。标识I图像的开始。

BBV延时　bbv_delay

32位无符号整数。

BbvDelay的值等于bbv_delay的值。如果temporal_id_enable_flag的值为‘1’，则bbv_delay的值应为BbvDelayMax。BbvDelayMax的值等于0xFFFFFFFF。

如果BbvDelay不等于BbvDelayMax，它规定了BBV从收到图像起始码的最后一个字节到开始解码图像之间要等待的时间。这个时间用从27MHz系统时钟导出的90kHz时钟周期数来表示。如果视频序列中某一幅图像的BbvDelay等于BbvDelayMax，那么整个视频序列中所有图像的BbvDelay均应等于BbvDelayMax。应符合附录C的规定。

时间编码标志　time_code_flag

二值变量。值为‘1’表示位流中包含time_code，值为‘0’表示位流中没有time_code。

时间编码　time_code

24位位串，包括以下字段：TimeCodeHours、TimeCodeMinutes、TimeCodeSeconds和TimeCodePictures，应符合表54的规定。表中TimeCodeHours，TimeCodeMinutes，TimeCodeSeconds和TimeCodePictures字

段是无符号整数。time_code描述从当前图像开始（含当前图像）的位流中第一幅图像（显示顺序）的显示时间。

表54　时间编码（time_code）

time_code的字段	取值	单位	描述符
保留	0	—	u(1)
TimeCodeHours	0～23	小时（h）	u(5)
TimeCodeMinutes	0～59	分（min）	u(6)
TimeCodeSeconds	0～59	秒（s）	u(6)
TimeCodePictures	0～63	如果帧率小于64，单位是图像；否则，单位是1/64s	u(6)

解码顺序索引　decode_order_index

8位无符号整数。说明当前图像的解码顺序索引值。当前图像的解码顺序索引DOI的值等于decode_order_index的值。

知识图像索引　library_picture_index

说明知识位流中当前图像的知识图像索引，解析过程见8.2。取值范围为0～511。LibraryPictureIndex的值等于library_picture_index的值。同一主位流所参考的所有知识图像中任意两幅不同的知识图像所对应的知识图像索引不应相同。

时间层标识　temporal_id

3位无符号整数。说明当前图像的时间层标识。时间层标识的取值范围为0～MAX_TEMPORAL_ID。如果位流中不存在temporal_id，则当前图像的时间层标识应为0。时间层标识为0说明是最低层。

图像输出延迟　picture_output_delay

从图像完成解码到输出需要等待的时间,以解码图像为单位,解析过程见8.2。low_delay的值为‘0’时，PictureOutputDelay的值等于picture_output_delay的值。low_delay的值为‘1’时，PictureOutputDelay的值为0。picture_output_delay的值应小于64。

引用参考图像队列配置集标志　ref_pic_list_set_flag[0]、ref_pic_list_set_flag[1]

二值变量。值为‘1’表示通过序列头获得当前图像的参考图像队列0（或参考图像队列1）的参考图像队列配置集，值为‘0’表示通过图像头获得当前图像的参考图像队列0（或参考图像队列1）的参考图像队列配置集。

如果NumRefPicListSet[0]（或NumRefPicListSet[1]）的值为0，则ref_pic_list_set_flag[0]（或ref_pic_list_set_flag[1]）的值应为‘0’。RefPicListSetFlag[0]的值等于ref_pic_list_set_flag[0]的值。如果Rpl1IndexExistFlag的值为0,则RefPicListSetFlag[1]的值等于ref_pic_list_set_flag[0]的值；否则RefPicListSetFlag[1]的值等于ref_pic_list_set_flag[1]的值。

引用参考图像队列配置集索引　ref_pic_list_set_index[0]、ref_pic_list_set_index[1]

表示当前图像的参考图像队列0（或参考图像队列1）的参考图像队列配置集索引值。该值由Ceil(Log(NumRefPicListSet[i]))位表示，取值范围为0～(NumRefPicListSet[i]-1)（i等于0或1）。

如果位流中不存在ref_pic_list_set_index[0]，则ref_pic_list_set_index[0]的值应为‘0’。如果位流中不存在ref_pic_list_set_index[1]且Rpl1IndexExistFlag的值为0，则ref_pic_list_set_index[1]的值等于ref_pic_list_set_index[0]的值。如果位流中不存在ref_pic_list_set_index[1]且Rpl1IndexExistFlag的值为1，则ref_pic_list_set_index[1]的值应为‘0’。

如果ref_pic_list_set_flag[i]的值为‘1’,则RplsIndex[i]的值等于ref_pic_list_set_index[i]的值；否则，RplsIndex[i]的值等于NumRefPicListSet[i]的值（i等于0或1）。

BBV检测次数 bbv_check_times

如果low_delay的值为‘0’，位流中不应出现bbv_check_times，此时BbvCheckTimes等于0。如果位流中出现bbv_check_times，则由bbv_check_times解析得到BbvCheckTimes，解析过程见8.2。bbv_check_times的值应小于2^{16}-1。

BbvCheckTimes大于0表示当前图像是一个大图像（应符合附录C的规定）。

逐行帧标志 progressive_frame

二值变量。值为‘0’表示当前图像所在的帧的两场是隔行场，这两场之间存在一个场时间间隔；值为‘1’表示当前图像所在的帧的两场实际上来自同一时刻。

如果progressive_sequence的值为‘1’，则progressive_frame的值应为‘1’。如果field_coded_sequence的值为‘1’时，则progressive_frame的值应为‘0’。

图像编码结构标志 picture_structure

二值变量。picture_structure的值为‘0’表示当前帧的两场的编码数据依次出现，值为‘1’表示当前帧的两场的编码数据交融出现。

如果progressive_sequence的值为‘1’，则picture_structure的值也应为‘1’。如果field_coded_sequence的值为‘0’，则picture_structure的值应为‘1’；如果field_coded_sequence的值为‘1’，则picture_structure的值应为‘0’。PictureStructure的值等于picture_structure的值。如果位流中不存在picture_structure，则PictureStructure的值应为1。

顶场在先 top_field_first

二值变量。其含义由progressive_sequence、progressive_frame、picture_structure和repeat_first_field决定。如果field_coded_sequence的值为‘1’,则top_field_first的值应为‘1’。

具体如下：

——如果progressive_sequence和field_coded_sequence的值均是‘0’，top_field_first说明解码场的输出显示顺序：

1) 如果PictureStructure的值是‘1’，则解码处理首先解码整帧；
2) 如果top_field_first的值是‘1’，则顶场在底场之前输出；
3) 如果top_field_first的值是‘0’，则底场在顶场之前输出。

——如果progressive_sequence的值是‘1’，top_field_first和repeat_first_field一起说明显示当前帧的次数（1次、2次或3次）：

1) 如果repeat_first_field的值是‘0’，则top_field_first的值也应是‘0’，显示当前帧一次；
2) 如果top_field_first的值是‘0’，repeat_first_field的值是‘1’，则显示当前帧两次；
3) 如果top_field_first和repeat_first_field的值都是‘1’，显示当前帧三次。

如果progressive_sequence、field_coded_sequence和progressive_frame的值都是‘0’，则视频序列中所有图像的top_field_first的值应相同。

重复首场 repeat_first_field

二值变量。如果progressive_frame的值是‘0’，则repeat_first_field的值应为‘0’。

具体如下：

——如果progressive_sequence和progressive_frame的值都是‘0’，则repeat_first_field的值也应是‘0’，显示两个场，第一场后面跟着第二场。如果field_coded_sequence的值是‘0’，则由top_field_first决定是第一场是顶场还是底场。

——如果 progressive_sequence 的值是‘0’，progressive_frame 的值是‘1’，那么：

1) 如果 repeat_first_field 的值是‘0’，显示两个场，第一场（由 top_field_first 决定是顶场还是底场），后面跟着第二场；
2) 如果 repeat_first_field 的值是‘1’，显示三个场，第一场（由 top_field_first 决定是顶场还是底场），后面跟着第二场，最后重复输出第一场。

顶场场图像标志　top_field_picture_flag

二值变量。值为‘1’表示当前图像是顶场图像，值为‘0’表示当前图像是底场图像。

固定图像量化因子标志　fixed_picture_qp_flag

二值变量。值为‘1’表示在该幅图像内量化因子不变，值为‘0’表示在该帧图像内量化因子可变。FixedPictureQpFlag的值等于fixed_picture_qp_flag的值。

图像量化因子　picture_qp

7位无符号整数。给出图像的量化因子。量化因子取值范围为0～(63 + 8 × (BitDepth − 8))。

编码单元量化参数可变标志　cu_delta_qp_flag

二值变量。值为‘1’表示图像中一棵编码树内的编码单元的量化参数可能不同，值为‘0’表示图像中任意一棵编码树内的所有编码单元的量化参数相同。CuDeltaQpFlag的值等于cu_delta_qp_flag。如果位流中不存在cu_delta_qp_flag，则CuDeltaQpFlag的值为0。

图像级编码单元量化组尺寸　cu_qp_group_size_log2_minus3

2位无符号整数。给出编码单元量化组的尺寸，取值范围为0～3。cu_qp_group_size_log2_minus3 的值应小于或等于(log2_lcu_size_minus2 − 1)。CuQpGroupAreaSize 的值等于 1 << ((cu_qp_group_size_log2_minus3 + 3) * 2)。

去块滤波禁用标志　deblocking_filter_disable_flag

二值变量。值为‘1’表示不应使用去块效应滤波，值为‘0’表示应使用去块效应滤波。DeblockingFilterDisableFlag的值等于deblocking_filter_disable_flag的值。

去块滤波类型　deblocking_filter_type

1位无符号整数。值为‘1’表示当前图像应使用改进型去块滤波，值为‘0’表示当前图像不应使用改进型去块滤波。DeblockingFilterType的值等于deblocking_filter_type。如果位流中不存在deblocking_filter_type，则DeblockingFilterType的值为0。

去块滤波参数标志　deblocking_filter_parameter_flag

二值变量。值为‘1’表示位流中包含alpha_c_offset和beta_offset，值为‘0’表示位流中不存在alpha_c_offset和beta_offset。

α和C索引的偏移　alpha_c_offset

当前图像环路滤波α和C索引的偏移,alpha_c_offset取值范围为-8～8,环路滤波参数AlphaCOffset等于alpha_c_offset。如果位流中不存在alpha_c_offset，则AlphaCOffset的值为0。

β索引的偏移　beta_offset

当前图像环路滤波β索引的偏移，beta_offset取值范围为-8～8，环路滤波参数BetaOffset等于beta_offset。如果位流中不存在beta_offset，则BetaOffset的值为0。

图像级去块滤波垂直调整允许标志　picture_dbr_v_enable_flag

二值变量。值为‘1’表示当前图像可使用去块滤波垂直调整，值为‘0’表示当前图像不应使用去块滤波垂直调整。PictureDbrVEnableFlag的值等于picture_dbr_v_enable_flag的值。如果位流中不存在picture_dbr_v_enable_flag，则PhDbrVEnableFlag的值为0。

图像级增强去块滤波垂直调整允许标志　picture_alt_dbr_v_enable_flag

二值变量。值为‘1’表示当前图像应使用增强去块滤波垂直调整，值为‘0’表示当前图像不应使用增强去块滤波垂直调整。PictureAltDbrVEnableFlag的值等于picture_alt_dbr_v_enable_flag的值。如果位流中不存在picture_alt_dbr_v_enable_flag，则PictureAltDbrVEnableFlag的值为0。

去块滤波垂直调整阈值　dbr_v_threshold_minus1

用于确定当前图像去块滤波垂直调整的阈值，取值范围为0～1。DbrVThreshold的值等于dbr_v_threshold_minus1的值加1。如果位流中不存在dbr_v_threshold_minus1，则DbrVThreshold的值为0。

去块滤波垂直调整偏移值0　dbr_v_offset0_minus1

用于确定当前图像去块滤波垂直调整的偏移值0，取值范围为0～3。DbrVOffset0的值等于dbr_v_offset0_minus1的值加1后的负值。如果位流中不存在dbr_v_offset0_minus1，则DbrVOffset0的值为0。

去块滤波垂直调整偏移值1　dbr_v_offset1_minus1

用于确定当前图像去块滤波垂直调整的偏移值1，取值范围为0～3。DbrVOffset1的值等于dbr_v_offset1_minus1的值加1。如果位流中不存在dbr_v_offset1_minus1，则DbrVOffset1的值为0。

增强去块滤波垂直调整偏移值0　dbr_v_alt_offset0_minus1

用于确定当前图像增强去块滤波垂直调整的偏移值0，取值范围为0～3。DbrVAltOffset0的值等于dbr_v_alt_offset0_minus1的值加1后的负值。如果位流中不存在dbr_v_alt_offset0_minus1，则DbrVAltOffset0的值为0。

增强去块滤波垂直调整偏移值1　dbr_v_alt_offset1_minus1

用于确定当前图像增强去块滤波垂直调整的偏移值1，取值范围为0～3。DbrVAltOffset1的值等于dbr_v_alt_offset1_minus1的值加1。如果位流中不存在dbr_v_alt_offset1_minus1，则DbrVAltOffset1的值为0。

图像级去块滤波水平调整允许标志　picture_dbr_h_enable_flag

二值变量。值为‘1’表示当前图像可使用去块滤波水平调整，值为‘0’表示当前图像不应使用去块滤波水平调整。PictureDbrHEnableFlag的值等于picture_dbr_h_enable_flag的值。如果位流中不存在picture_dbr_h_enable_flag，则PictureDbrHEnableFlag的值为0。

图像级增强去块滤波水平调整允许标志　picture_alt_dbr_h_enable_flag

二值变量。值为‘1’表示当前图像应使用增强去块滤波水平调整，值为‘0’表示当前图像不应使用增强去块滤波水平调整。PictureAltDbrHEnableFlag的值等于picture_alt_dbr_h_enable_flag的值。如果位流中不存在picture_alt_dbr_h_enable_flag，则PictureAltDbrHEnableFlag的值为0。

去块滤波水平调整阈值　dbr_h_threshold_minus1

用于确定当前图像去块滤波水平调整的阈值，取值范围为0～1。DbrHThreshold的值等于dbr_h_threshold_minus1的值加1。如果位流中不存在dbr_h_threshold_minus1，则DbrHThreshold的值为0。

去块滤波水平调整偏移值0　dbr_h_offset0_minus1

用于确定当前图像去块滤波水平调整的偏移值0，取值范围为0～3。DbrHOffset0的值等于dbr_h_offset0_minus1的值加1后的负值。如果位流中不存在dbr_h_offset0_minus1，则DbrHOffset0的值为0。

去块滤波水平调整偏移值1　dbr_h_offset1_minus1

用于确定当前图像去块滤波水平调整的偏移值1，取值范围为0～3。DbrHOffset1的值等于dbr_h_offset1_minus1的值加1。如果位流中不存在dbr_h_offset1_minus1，则DbrHOffset1的值为0。

增强去块滤波水平调整偏移值0　dbr_h_alt_offset0_minus1

用于确定当前图像增强去块滤波水平调整的偏移值0，取值范围为0～3。DbrHAltOffset0的值等于dbr_h_alt_offset0_minus1的值加1后的负值。如果位流中不存在dbr_h_alt_offset0_minus1，则DbrHAltOffset0的值为0。

增强去块滤波水平调整偏移值1　dbr_h_alt_offset1_minus1

用于确定当前图像增强去块滤波水平调整的偏移值1，取值范围为0～3。DbrHAltOffset1的值等于dbr_h_alt_offset1_minus1的值加1。如果位流中不存在dbr_h_alt_offset1_minus1，则DbrHAltOffset1的值为0。

色度量化参数禁用标志　chroma_quant_param_disable_flag

二值变量。值为‘1’表示当前图像的图像头中不存在chroma_quant_param_delta_cb和chroma_quant_param_delta_cr，值为‘0’表示当前图像的图像头中存在chroma_quant_param_delta_cb和chroma_quant_param_delta_cr。

色度量化参数增量Cb　chroma_quant_param_delta_cb

色度量化参数增量Cr　chroma_quant_param_delta_cr

色度编码块量化参数相对于CurrentQp的增量，取值范围为-16～16。解析过程见8.2，解码过程见9.5.2。如果当前图像的图像头中不存在chroma_quant_param_delta_cb和chroma_quant_param_delta_cr，则chroma_quant_param_cb和chroma_quant_param_cr的值均为‘0’。

图像加权量化允许标志　picture_weight_quant_enable_flag

二值变量。值为‘1’表示当前图像可使用加权量化，值为‘0’表示当前图像不应使用加权量化。PictureWeightQuantEnableFlag的值等于picture_weight_quant_enable_flag。如果当前图像的图像头位流中不存在picture_weight_quant_enable_flag，则PictureWeightQuantEnableFlag的值应为0。

图像加权量化数据加载索引　picture_weight_quant_data_index

2位无符号整数。值为‘00’表示当前图像的4×4和8×8变换块的加权量化矩阵根据序列头确定，值为‘01’表示当前图像的4×4和8×8变换块的加权量化矩阵根据当前图像头中加载的加权量化参数导出，值为‘10’表示当前图像的4×4和8×8变换块的加权量化矩阵从当前图像头中直接加载，值为‘11’保留。

加权量化参数索引　weight_quant_param_index

2位无符号整数。当前图像的加权量化参数索引。值为‘11’保留。

加权量化矩阵模型　weight_quant_model

2位无符号整数，规定加权量化参数的分布模型。值为‘11’保留。如果当前图像头的位流中存在weight_quant_model，则WeightQuantModel的值等于weight_quant_model的值。

加权量化参数增量1　weight_quant_param_delta1[i]

加权量化参数增量2　weight_quant_param_delta2[i]

当前图像的加权量化参数的增量，取值范围为-128～127。解析过程见8.2。解码过程见9.2.7。如果当前图像头的位流中不存在weight_quant_param_delta1[i]或weight_quant_param_delta2[i]，则weight_quant_param_delta1[i]或weight_quant_param_delta2[i]的值为0。

图像级自适应修正滤波允许标志　picture_alf_enable_flag[compIndex]

二值变量。值为‘1’表示当前图像的第compIndex个分量可使用自适应修正滤波，值为‘0’则表示当前图像的第compIndex个分量不应使用自适应修正滤波。PictureAlfEnableFlag[compIndex]的值等于picture_alf_enable_flag[compIndex]的值。

图像级块复制帧内预测允许标志　picture_ibc_enable_flag

二值变量。值为‘1’表示当前图像可使用块复制帧内预测，值为‘0’表示当前图像不应使用块复制帧内预测。PictureIbcEnableFlag的值等于picture_ibc_enable_flag的值。如果位流中不存在picture_ibc_enable_flag，则PictureIbcEnableFlag的值为0。

图像级串复制帧内预测普通串子模式允许标志　picture_isc_os_enable_flag

二值变量。值为‘1’表示当前图像可使用串复制帧内预测中的普通串子模式，值为‘0’表示当前图像不应使用串复制帧内预测中的普通串子模式。PictureIscOsEnableFlag的值等于picture_isc_os_enable_flag的值。如果位流中不存在picture_isc_os_enable_flag，则PictureIscOsEnableFlag的值为0。

图像级串复制帧内预测非普通串子模式允许标志　picture_isc_evs_ubvs_enable_flag

二值变量。值为‘1’表示当前图像可使用串复制帧内预测中的非普通串子模式，值为‘0’表示当前图像不应使用串复制帧内预测中的非普通串子模式。PictureIscEvsUbvsEnableFlag的值等于picture_isc_evs_ubvs_enable_flag的值。如果位流中不存在picture_isc_evs_ubvs_enable_flag，则PictureIscEvsUbvsEnableFlag的值为0。PictureIscEnableFlag的值等于PictureIscOsEnableFlag || PictureIscEvsUbvsEnableFlag。

图像级基于频数信息的帧内编码允许标志　picture_fimc_enable_flag

二值变量。值为‘1’表示当前图像可使用基于频数信息的帧内编码，值为‘0’表示当前图像不应使用基于频数信息的帧内编码。PictureFimcEnableFlag的值等于picture_fimc_enable_flag的值。如果位流中不存在picture_fimc_enable_flag，则PictureFimcEnableFlag的值为0。

图像级变换跳过允许标志　picture_ts_enable_flag

二值变量。值为‘1’表示当前图像可使用变换跳过方法，值为‘0’表示当前图像不应使用变换跳过方法。PictureTsEnableFlag的值等于picture_ts_enable_flag的值。如果位流中不存在picture_ts_enable_flag，则PictureTsEnableFlag的值为0。

图像级跨多分量预测参数索引　picture_pmc_param_index

1位无符号整数。用于确定当前图像的跨多分量预测参数。PicturePmcParamIndex的值等于picture_pmc_param_index。如果位流中不存在picture_pmc_param_index，则PicturePmcParamIndex的值为0。

图像级神经网络滤波允许标志　picture_nn_filter_enable_flag[compIdx]

二值变量。值为‘1’表示当前图像的第compIdx个分量可使用神经网络滤波，值为‘0’则表示当前图像的第compIdx个分量不应使用神经网络滤波。PictureNnFilterEnableFlag[compIdx]的值等于picture_nn_filter_enable_flag[compIdx]的值。如果位流中不存在picture_nn_filter_enable_flag[compIdx]，则PictureNnFilterEnableFlag[compIdx]的值为0。解码处理应该忽略这些位。

图像级选择性滤波自适应标志　picture_nn_filter_adaptive_flag[compIdx]

二值变量。值为‘1’表示当前图像的第compIdx个分量可使用自适应的神经网络滤波，值为‘0’则表示当前图像的第compIdx个分量不应使用自适应的神经网络滤波。PictureNnFilterAdaptiveFlag[compIdx]的值等于picture_nn_filter_adaptive_flag[compIdx]的值。如果位流中不存在picture_nn_filter_adaptive_flag[compIdx]，则PictureNnFilterAdaptiveFlag[compIdx]的值为0。解码处理应该忽略这些位。

图像级神经网络滤波模型索引　picture_nn_filter_set_index[compIdx]

表示当前图像的第compIdx个分量使用神经网络滤波的模型索引。解析过程见8.2，取值范围为0～NumOfNnFilter-1。PictureNnFilterSetIndex[compIdx]的值等于picture_nn_filter_set_index[compIdx]的值。如果位流中不存在picture_nn_filter_set_index[compIdx]，则PictureNnFilterSetIndex[compIdx]的值为-1。解码处理应该忽略这些位。

7.2.3.2　帧间预测图像头

帧间预测图像起始码　inter_picture_start_code

位串‘0x000001B6’。标识P图像或B图像的开始。

随机访问正确解码标志　random_access_decodable_flag

二值变量。值为‘1’表示当前图像只参考解码顺序在当前图像对应的序列头之后且random_access_decodable_flag的值为‘1’的图像，值为‘0’表示当前图像不一定只参考解码顺序在当前图像对应的序列头之后且random_access_decodable_flag的值为‘1’的图像。其中，对应的序列头指的是码流中在当前图像之前最近的一个序列头。RandomAccessDecodableFlag的值等于random_access_decodable_flag的值。如果当前图像的图像头中不存在random_access_decodable_flag，则当前图像的RandomAccessDecodableFlag的值应为1。

如果当前图像的RandomAccessDecodableFlag的值为0，则在其对应的序列头发生随机访问时，当前图像可能无法正确解码。

序列头后的第一幅解码图像应是I图像或RL图像。解码顺序在该I图像或RL图像之后，显示顺序在该I图像或RL图像之前的图像称为该序列头对应的前置图像，只有前置图像的RandomAccessDecodableFlag的值可以为0，其余图像的RandomAccessDecodableFlag的值应为1。并且对于同一个序列头对应的前置图像，RandomAccessDecodableFlag的值为0的图像的显示顺序应在RandomAccessDecodableFlag的值为1的图像的显示顺序之前。

图像编码方式　picture_coding_type

2位无符号整数。规定帧间预测图像的类型，应符合表55的规定。

表55　帧间预测图像的类型

picture_coding_type的值	帧间预测图像的类型
00	禁止
01	P图像
10	B图像
11	保留

活跃参考图像数重载标志　num_ref_active_override_flag

二值变量。值为‘1’表示位流中存在num_ref_active_minus1[0]或num_ref_active_minus1[1]，值为‘0’表示位流中不存在num_ref_active_minus1[0]和num_ref_active_minus1[1]。

活跃参考图像数　num_ref_active_minus1[0]、num_ref_active_minus1[1]

表示解码当前图像时，参考图像队列0（或参考图像队列1）的参考索引的最大值。解析过程见8.3。取值范围为0～14。NumRefActive[i]的值等于num_ref_active_minus1[i]加1（i等于0或1）。如果位流中不存在num_ref_active_minus1[0]，则NumRefActive[0]的值等于0；如果位流中不存在num_ref_active_minus1[1]，则NumRefActive[1]的值等于0。

当前图像解码的参考图像队列i的参考索引的最大值等于num_ref_active_minus1[i]的值（i等于0或1）。如果当前图像是I图像，则NumRefActive[0]和NumRefActive[1]的值均为0。如果当前图像是P图像且NumRefActiveOverrideFlag的值为0，则NumRefActive[0]的值等于num_ref_default_active_minus1[0] +1，NumRefActive[1]的值为0。如果当前图像是B图像且NumRefActiveOverrideFlag的值为0，则NumRefActive[i]的值等于num_ref_default_active_minus1[i]+1（i等于0或1）。

如果num_ref_active_override_flag的值为‘0’，则NumRefActive[i]的值等于num_ref_default_active_minus1[i]的值加1（i等于0或1）。NumRefActive[i]的值应小于或等于NumOfRefPic[i][RplsIndex[i]]的值（i等于0或1）。

仿射预测子块尺寸标志　affine_subblock_size_flag

二值变量。值为‘0’表示当前图像仿射预测子块最小尺寸为4×4，值为‘1’表示最小尺寸为8×8。AffineSubblockSizeFlag的值等于affine_subblock_size_flag的值。如果位流中不存在affine_subblock_size_flag，则AffineSubblockSizeFlag的值为0。

图像级高级运动矢量表达模式运动矢量偏移量集合索引　picture_umve_step_set_index

1位无符号整数。用于确定高级运动矢量表达模式可使用的运动矢量偏移量。PictureUmveStepSetIndex的值等于picture_umve_step_set_index。如果位流中不存在picture_umve_step_set_index，则PictureUmveStepSetIndex的值为0。

图像级重叠块运动补偿加权系数集合索引　picture_obmc_blending_set_index

1位无符号整数。用于确定重叠块运动补偿加权系数。PictureObmcBlendingSetIndex的值等于picture_obmc_blending_set_index。如果位流中不存在picture_obmc_blending_set_index，则PictureObmcBlendingSetIndex的值为0。

图像级角度加权预测权重索引　picture_awp_refine_index

1位无符号整数。用于确定角度加权预测模式的参考权重。PictureAwpRefineIndex的值等于picture_awp_refine_index的值。如果位流中不存在picture_awp_refine_index，则PictureAwpRefineIndex的值为0。

图像级跳过模式帧间预测修正允许标志　picture_inter_pc_skip_flag

二值变量。值为‘1’表示当前图像使用跳过模式的编码单元可使用帧间预测修正，值为‘0’表示当前图像使用跳过模式的编码单元不应使用帧间预测修正。PictureInterPcSkipFlag的值等于picture_inter_pc_skip_flag的值。如果位流中不存在picture_inter_pc_skip_flag，则PictureInterPcSkipFlag的值为0。

7.2.3.3　图像显示扩展

图像显示扩展不定义显示过程。图像显示扩展中的信息对解码处理没有影响，解码器可忽略这些信息。

图像显示扩展允许显示矩形（其尺寸由序列显示扩展定义）按图像移动。其中一项应用是实现全景扫描。

图像显示扩展标号　extension_id

位串‘0111’。标识图像显示扩展。

图像中心水平偏移　picture_centre_horizontal_offset

16位整数。以1/16样本为单位给出水平偏移。正值表示重建图像的中心位置在显示矩形中心的右侧。

图像中心垂直偏移　picture_centre_vertical_offset

16位整数。以1/16样本为单位给出垂直偏移。正值表示重建图像的中心位置在显示矩形中心的下方。

显示矩形区域的尺寸在序列显示扩展中定义。编码图像内区域的坐标由图像显示扩展定义。

重建图像的中心指由horizontal_size和vertical_size定义的矩形的中心。

在隔行视频序列中，一幅编码图像可能与一个、两个或三个解码场有关，因此图像显示扩展最多可以定义三组偏移量。

7.1.3.3中NumberOfFrameCentreOffsets的值按以下方式定义：

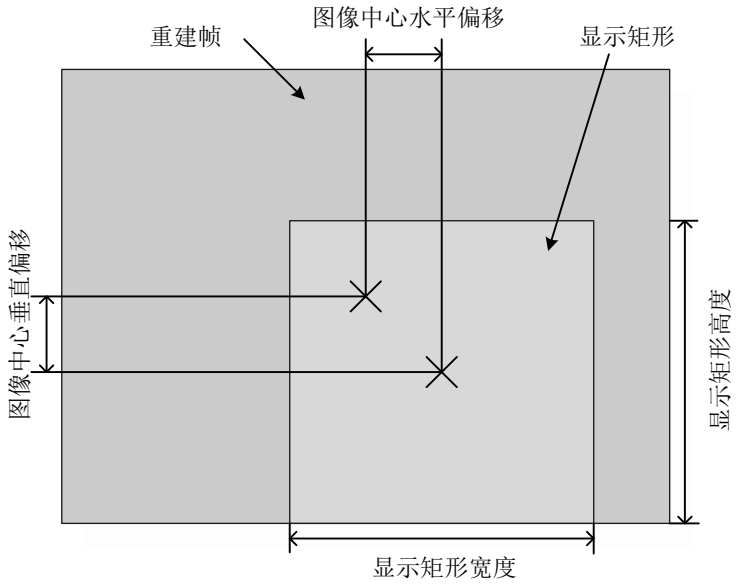
```
if ( progressive_sequence == '1' ) {
    if ( repeat_first_field == '1' ) {
        if ( top_field_first == '1' )
            NumberOfFrameCentreOffsets = 3
        else
            NumberOfFrameCentreOffsets = 2
    }
    else {
        NumberOfFrameCentreOffsets = 1
    }
}
else {
    if ( picture_structure == '0' ) {
        NumberOfFrameCentreOffsets = 1
    }
    else {
        if ( repeat_first_field == '1' )
            NumberOfFrameCentreOffsets = 3
        else
            NumberOfFrameCentreOffsets = 2
    }
}
```

如果前面的序列头之后没有序列显示扩展，那么位流中不应出现图像显示扩展。

如果一幅图像没有图像显示扩展，使用最近的解码图像的中心偏移量。

注：丢失图像的中心偏移量的值与前一非丢失图像的值相同。在序列头后，所有图像的中心偏移量置为0，直到出现图像显示扩展。

利用图像中心偏移量可定义一个矩形区域，这个区域在整个重建图像范围内移动来实现全景扫描。图像中心偏移量参数见图10。



注1：显示矩形的尺寸可能大于重建图像。

注2：在场图像中，图像中心垂直偏移表示的中心偏移量以 1/16 帧行为单位。

注3：图中，图像中心垂直偏移和图像中心水平偏移均为负值。

图10　图像中心偏移量参数

7.2.4　片语义

片中compIndex等于0表示亮度分量，等于1表示Cb分量，等于2表示Cr分量。

片起始码　patch_start_code

32位位串，前24位是‘0000 0000 0000 0000 0000 0001’，后8位是patch_index。patch_index是8位无符号整数，取值范围为0x00～0x7F。

按以下过程导出片的行索引和列索引：

```
PatchIndexX = patch_index % NumPatchColumns
PatchIndexY = patch_index / NumPatchColumns
LcuRow = PatchSizeInLCU[PatchIndexY][PatchIndexX]->PosY
LcuColumn = PatchSizeInLCU[PatchIndexY][PatchIndexX]->PosX
```

固定片量化因子标志　fixed_patch_qp_flag

二值变量。值为‘1’表示在该片内量化因子不变，值为‘0’表示在该片内量化因子可变。FixedPatchQpFlag的值等于fixed_patch_qp_flag的值。

片量化因子　patch_qp

7位无符号整数。给出片的量化因子，取值范围为0～(63 + 8 × (BitDepth − 8))。PatchQp的值等于patch_qp的值，如果patch_qp不存在，则PatchQp的值等于picture_qp的值。

片样值偏移自适应补偿允许标志　patch_sao_enable_flag[compIndex]

二值变量。PatchSaoEnableFlag[compIndex]的值等于patch_sao_enable_flag[compIndex]的值。PatachSaoEnableFlag[compIndex]的值为1表示在该片内第compIndex个分量可使用样值偏移自适应补偿，值为0则表示在该片内第compIndex个分量不应使用样值偏移自适应补偿。如果位流中不存在patch_sao_enable_flag[compIndex]，则PatchSaoEnableFlag[compIndex]的值为0。

高级熵编码字节对齐填充位　aec_byte_alignment_bit

填充位。值应为‘1’。

最大编码单元量化参数增量　lcu_qp_delta

给出当前最大编码单元的量化参数相对预测量化参数的增量。解析过程见8.3。LcuQpDelta的值等于lcu_qp_delta的值。LcuQpDelta的取值范围应为(-32－4*(BitDepth-8))～(32＋4×(BitDepth-8))。

样值偏移自适应补偿合并方式索引　sao_merge_type_index

样值偏移自适应补偿合并方式的索引值。解析过程见8.3。SaoMergeTypeIndex的值等于sao_merge_type_index的值。如果当前样本的样值偏移自适应补偿单元E的左边样值偏移自适应补偿单元L存在,且与E对应的最大编码单元和L对应的最大编码单元处于同一片,则SaoMergeLeftAvai的值为1;否则SaoMergeLeftAvai的值为0。如果当前样本的样值偏移自适应补偿单元E的上边样值偏移自适应补偿单元U存在,且与E对应的最大编码单元和U对应的最大编码单元处于同一片,则SaoMergeUpAvai的值为1;否则SaoMergeUpAvai的值为0。

样值偏移自适应补偿模式　sao_mode[compIndex]

样值偏移自适应补偿模式的索引值。解析过程见8.3。由sao_mode的值查表56得到SaoMode和MaxOffsetNumber的值。

表56　样值偏移自适应补偿模式和最大偏移数

sao_mode	SaoMode	MaxOffsetNumber	偏移补偿模式
0	SAO_Off	0	关闭模式
1	SAO_Interval	4	区间模式
2	SAO_Edge	4	边缘模式

样值偏移自适应补偿区间模式偏移值绝对值　sao_interval_offset_abs[compIndex][j]

区间模式偏移值的绝对值,取值范围为0～7。解析过程见8.3。SaoIntervalOffsetAbs[compIndex][j]的值等于sao_interval_offset_abs[compIndex][j]的值。

样值偏移自适应补偿区间模式偏移值符号值　sao_interval_offset_sign[compIndex][j]

区间模式偏移值的符号位。解析过程见8.3。如果sao_interval_offset_sign[compIndex][j]的值为‘0’，则SaoIntervalOffsetSign[compIndex][j]的值等于1；否则SaoIntervalOffsetSign[compIndex][j]的值等于-1。如果位流中不存在sao_interval_offset_sign[compIndex][j]，则SaoIntervalOffsetSign[compIndex][j]的值为0。

样值偏移自适应补偿区间模式起始偏移子区间位置　sao_interval_start_pos[compIndex]

区间模式起始偏移子区间的位置，取值范围为0～31。解析过程见8.3。SaoIntervalStartPos[compIndex]的值等于sao_interval_start_pos[compIndex]的值。

样值偏移自适应补偿区间模式起始偏移子区间位置差　sao_interval_delta_pos_minus2[compIndex]

区间模式起始偏移子区间的位置差，取值范围为0～14。解析过程见8.3。SaoIntervalDeltaPosMinus2[compIndex]的值等于sao_interval_delta_pos_minus2[compIndex]的值。

样值偏移自适应补偿边缘模式偏移值　sao_edge_offset[compIndex][j]

边缘模式的偏移值。解析过程见8.3。SaoEdgeOffset[compIndex][j]的值等于sao_edge_offset[compIndex][j]的值。sao_edge_offset[compIndex][0]的取值范围为-1～6，sao_edge_offset[compIndex][1]的取值范围为0～1，sao_edge_offset[compIndex][2]的取值范围为-1～0，sao_edge_offset[compIndex][3]的取值范围为-6～1。

样值偏移自适应补偿边缘模式类型 sao_edge_type[compIndex]

边缘模式的类型。解析过程见8.3。SaoEdgeType[compIndex]的值等于sao_edge_type[compIndex]的值。

最大编码单元增强样值偏移自适应补偿允许标志 esao_lcu_enable_flag[compIndex][LcuIndex]

二值变量。值为‘1’表示第LcuIndex个最大编码单元compIndex分量的样本应使用增强样值偏移自适应补偿，值为‘0’表示第LcuIndex个最大编码单元compIndex分量的样本不应使用增强样值偏移自适应补偿。解析过程见8.3。EsaoLcuEnableFlag[compIndex][LcuIndex]的值等于esao_lcu_enable_flag[compIndex][LcuIndex]的值。如果位流中不存在esao_lcu_enable_flag[compIndex][LcuIndex]，则EsaoLcuEnableFlag[compIndex][LcuIndex]的值等于PictureEsaoEnableFlag[compIndex]的值。

最大编码单元增强样值偏移自适应补偿组索引 esao_lcu_set_index[compIndex][LcuIndex]

表示第LcuIndex个最大编码单元compIndex分量的样本应使用的增强样值偏移自适应补偿系数组，解析过程见8.3。esao_lcu_set_index[compIndex][LcuIndex]的取值范围为0～1。EsaoLcuSetIndex[compIndex][LcuIndex]的值等于esao_lcu_set_index[compIndex][LcuIndex]的值。如果位流中不存在esao_lcu_set_index[compIndex][LcuIndex]，则EsaoLcuSetIndex[compIndex][LcuIndex]的值为0。

最大编码单元跨分量样值偏移自适应补偿允许标志 ccsao_lcu_enable_flag[compIndex][LcuIndex]

二值变量。值为‘1’表示第LcuIndex个最大编码单元compIndex分量的样本应使用跨分量样值偏移自适应补偿，值为‘0’表示第LcuIndex个最大编码单元compIndex分量的样本不应使用跨分量样值偏移自适应补偿。解析过程见8.3。CcsaoLcuEnableFlag[compIndex][LcuIndex]的值等于ccsao_lcu_enable_flag[compIndex][LcuIndex]的值。如果位流中不存在ccsao_lcu_enable_flag[compIndex][LcuIndex]，则CcsaoLcuEnableFlag[compIndex][LcuIndex]的值等于PictureCcsaoEnableFlag[compIndex]的值。

最大编码单元跨分量样值偏移自适应补偿组索引 ccsao_lcu_set_index[compIndex][LcuIndex]

表示第LcuIndex个最大编码单元compIndex分量的样本应使用的跨分量样值偏移自适应补偿组，解析过程见8.3。CcsaoLcuSetIndex[compIndex][LcuIndex]的值等于ccaso_lcu_set_index[compIndex][LcuIndex]的值。ccsao_lcu_set_index[compIndex][LcuIndex]的取值范围为0～3。如果位流中不存在ccsao_lcu_set_index[compIndex][LcuIndex]，则CcsaoLcuSetIndex[compIndex][LcuIndex]的值为0。

最大编码单元自适应修正滤波允许标志 alf_lcu_enable_flag[compIndex][LcuIndex]

二值变量。值为‘1’表示第LcuIndex个最大编码单元compIndex分量的样本应使用自适应修正滤波，值为‘0’表示第LcuIndex个最大编码单元compIndex分量的样本不应使用自适应修正滤波。AlfLcuEnableFlag[compIndex][LcuIndex]的值等于alf_lcu_enable_flag[compIndex][LcuIndex]的值。

最大编码单元神经网络滤波允许标志 nn_filter_lcu_enable_flag[compIdx][LcuIdx]

二值变量。值为‘1’表示第LcuIdx个最大编码单元compIdx分量的样本应使用神经网络滤波，值为‘0’表示第LcuIdx个最大编码单元compIdx分量的样本不应使用神经网络滤波。解析过程见8.3。NnFilterLcuEnableFlag[compIdx][LcuIdx]的值等于nn_filter_lcu_enable_flag[compIdx][LcuIdx]的值。如果位流中不存在nn_filter_lcu_enable_flag[compIdx][LcuIdx]，则NnFilterLcuEnableFlag[compIdx][LcuIdx]值为0。解码处理应该忽略这些位。

最大编码单元神经网络滤波模型索引 nn_filter_lcu_set_index[compIdx][LcuIdx]

表示第LcuIdx个最大编码单元compIdx分量的样本使用的神经网络滤波的模型索引，解析过程见8.3，取值范围为0～NumOfNnFilter-1。NnFilterLCUSetIndex[compIdx][LcuIdx]的值等于nn_filter_lcu_set_index[compIdx][LcuIdx]的值。如果位流中不存在nn_filter_lcu_set_index[compIdx][LcuIdx]，则NnFilterLCUSetIndex[compIdx][LcuIdx]值为-1。解码处理应该忽略这些位。

高级熵编码最大编码单元填充位　aec_lcu_stuffing_bit

填充位。片的最后一个最大编码单元的单元aec_lcu_stuffing_bit的值应为‘1’，解析过程见8.3。

片结束码　patch_end_code

位串‘0x0000018F’。标识片的结束。

7.2.5　编码树语义

四叉树划分标志　qt_split_flag

二值变量。值为‘1’表示应使用划分过程进行四叉树划分，值为‘0’表示不应进行四叉树划分。解析过程见8.3。QtSplitFlag的值等于qt_split_flag。如果位流中不存在qt_split_flag，则QtSplitFlag的值等于allowSplitQt的值。

编码单元预测模式限制　root_cu_constraint

二值变量。值为‘1’表示当前编码树节点覆盖的编码单元的预测模式限制是‘PRED_Intra_Only’，值为‘0’表示当前编码树节点覆盖的编码单元的预测模式限制是‘PRED_Inter_Only’。如果位流中不存在root_cu_constraint，则当前编码单元的预测模式限制应与父节点一致。

二叉树扩展四叉树划分标志　bet_split_flag

二值变量。值为‘1’表示应使用划分过程进行二叉树扩展四叉树划分，值为‘0’表示不应进行二叉树扩展四叉树行划分。解析过程见8.3。BetSplitFlag的值等于bet_split_flag。如果位流中不存在bet_split_flag，则BetSplitFlag等于allowNoSplit的值取反。

二叉树扩展四叉树划分类型标志　bet_split_type_flag

二值变量。值为‘0’表示进行二叉树扩展四叉树划分时应使用二叉树划分，值为‘1’表示进行二叉树扩展四叉树划分时应使用扩展四叉树。解析过程见8.3。BetSplitTypeFlag的值等于bet_split_type_flag。如果位流中不存在bet_split_type_flag，则BetSplitTypeFlag的值按以下方式确定：

——如果BetSplitFlag的值为0，则BetSplitTypeFlag等于0；

——否则，如果allowSplitBtVer的值为1或allowSplitBtHor的值为1，则BetSplitTypeFlag等于0；

——否则，如果allowSplitBtVer和allowSplitBtHor的值均为0，则BetSplitTypeFlag等于1。

二叉树扩展四叉树划分方向标志　bet_split_dir_flag

二值变量。值为‘1’表示进行二叉树扩展四叉树划分时应使用垂直划分，值为‘0’表示进行二叉树扩展四叉树划分时应使用水平划分。解析过程见8.3。BetSplitDirFlag的值等于bet_split_dir_flag。如果位流中不存在bet_split_dir_flag，则BetSplitDirFlag的值按以下方式确定：

——如果BetSplitFlag的值为0，则BetSplitDirFlag等于0；

——否则，如果BetSplitTypeFlag的值为0，则BetSplitDirFlag等于allowSplitBtVer；

——否则，如果BetSplitTypeFlag的值为1，则BetSplitDirFlag等于allowSplitEqtVer。

由QtSplitFlag、BetSplitFlag、BetSplitTypeFlag和BetSplitDirFlag查表57得到BlockSplitMode。

表57 BlockSplitMode

QtSplitFlag	BetSplitFlag	BetSplitTypeFlag	BetSplitDirFlag	BlockSplitMode
0	0	0	0	NO_SPLIT
1	0	0	0	SPLIT_QT
0	1	0	1	SPLIT_BT_VER
0	1	0	0	SPLIT_BT_HOR
0	1	1	1	SPLIT_EQT_VER
0	1	1	0	SPLIT_EQT_HOR

如果至少满足以下条件之一，则ChildSizeOccur4的值为1；否则，ChildSizeOccur4的值为0：

——BlockSplitMode 的值为‘SPLIT_QT’且当前节点的宽度或高度等于 8；

——BlockSplitMode 的值为‘SPLIT_BT_VER’且当前节点的宽度等于 8；

——BlockSplitMode 的值为‘SPLIT_BT_HOR’且当前节点的高度等于 8；

——BlockSplitMode 的值为‘SPLIT_EQT_VER’且当前节点的宽度等于 16 或高度等于 8；

——BlockSplitMode 的值为‘SPLIT_EQT_HOR’且当前节点的高度等于 16 或宽度等于 8。

7.2.6 编码单元语义

跳过模式标志 skip_flag

二值变量。值为‘1’表示当前编码单元使用跳过模式，值为‘0’表示不是跳过模式。解析过程见8.3。SkipFlag的值等于skip_flag的值。如果位流不存在skip_flag，则SkipFlag的值为0。

高级预测模式标志 advanced_pred_flag

二值变量。值为‘1’表示当前编码单元使用高级运动矢量表达模式或角加权预测模式或增强运动矢量预测模式，值为‘0’表示不是高级运动矢量表达模式或角加权预测模式或增强运动矢量预测模式。解析过程见8.3。AdvancedPredFlag的值等于advanced_pred_flag的值。如果位流中不存在advanced_pred_flag，则AdvancedPredFlag的值为0。

增强时域运动矢量预测模式标志 etmvp_flag

二值变量。值为‘1’表示当前编码单元使用增强时域运动矢量预测模式，值为‘0’表示当前编码单元不是增强时域运动矢量预测模式。解析过程见8.3。EtmvpFlag的值等于etmvp_flag的值。如果位流中不存在etmvp_flag，则EtmvpFlag的值如下：

```
EtmvpFlag = AdvancedPredFlag && AllowEtmvp
```

角度加权预测模式标志 awp_flag

二值变量。值为‘1’表示当前编码单元是角度加权预测模式，值为‘0’表示当前编码单元不是角度加权预测模式。解析过程见8.3。AwpFlag的值等于awp_flag的值。如果位流中不存在awp_flag，则AwpFlag的值如下：

```
AwpFlag = AdvancedPredFlag && (! EtmvpFlag) && AllowAwp
```

UmveFlag的值如下：

```
UmveFlag = AdvancedPredFlag && (! EtmvpFlag) && (! AwpFlag)
```

仿射模式标志 affine_flag

二值变量。值为‘1’表示当前编码单元是仿射模式，值为‘0’表示不是仿射模式。解析过程见8.3。AffineFlag的值等于affine_flag的值。如果位流中不存在affine_flag，则AffineFlag的值为0。

仿射高级运动矢量表达模式标志　affine_umve_flag

二值变量。值为‘1’表示当前编码单元是仿射高级运动矢量表达模式，值为‘0’表示不是仿射高级运动矢量表达模式。解析过程见8.3。AffineUmveFlag的值等于affine_umve_flag的值。如果位流中不存在affine_umve_flag，则AffineUmveFlag的值为0。

直接模式标志　direct_flag

二值变量。值为‘1’表示当前编码单元是直接模式，值为‘0’表示不是直接模式。解析过程见8.3。directFlag的值等于direct_flag的值。如果位流中不存在direct_flag，则directFlag的值为0。

帧间预测滤波标志　inter_pf_flag

二值变量。值为‘1’表示当前编码单元应使用帧间预测滤波，值为‘0’表示不应使用帧间预测滤波。解析过程见8.3。InterPfFlag的值等于inter_pf_flag的值。如果位流中不存在inter_pf_flag，则InterPfFlag的值为0。

帧间预测滤波索引　inter_pf_index

帧间预测滤波的索引值。解析过程见8.3。InterPfIndex的值等于inter_pf_index的值。InterPfIndex的值应为0或1。

帧间预测修正标志　inter_pc_flag

二值变量。值为‘1’表示当前编码单元应使用帧间预测修正，值为‘0’表示不应使用帧间预测修正。解析过程见8.3。InterPcFlag的值等于inter_pc_flag的值。如果位流中不存在inter_pc_flag，则InterPcFlag的值为0。

帧间预测修正索引　inter_pc_index

帧间预测修正索引值。解析过程见8.3。InterPcIndex的值等于inter_pc_index的值。InterPcIndex的值应为0、1或2。

帧内编码单元标志　intra_cu_flag

二值变量。值为‘1’表示当前编码单元的预测类型是普通帧内预测或块复制帧内预测或串复制帧内预测，值为‘0’表示当前编码单元的预测类型是帧间预测。解析过程见8.3。IntraCuFlag的值等于intra_cu_flag。如果位流中不存在intra_cu_flag，则：

——如果当前图像是 I 图像，或当前图像不是 I 图像且当前编码单元的预测模式限制是‘PRED_Intra_Only’，则 IntraCuFlag 的值为 1；

——如果当前编码单元只包含色度编码块且 PriorCuMode 的值等于 1，则 IntraCuFlag 的值为 1；

——否则，IntraCuFlag 的值为 0。

基础运动矢量索引　umve_mv_index

高级运动矢量表达模式中使用的基础运动矢量的索引值。解析过程见8.3。解码过程见9.5.7.8.5。UmveMvIndex的值等于umve_mv_index的值。UmveMvIndex的值应为0或1。

运动矢量偏移量索引　umve_step_index

高级运动矢量表达模式中使用的运动矢量偏移量的索引值。解析过程见8.3。解码过程见9.5.7.8.5。UmveStepIndex的值等于umve_step_index的值。

运动矢量方向索引　umve_dir_index

高级运动矢量表达模式中使用的运动矢量方向的索引值。解析过程见8.3。解码过程见9.5.7.8.5。UmveDirIndex的值等于umve_dir_index的值。

仿射运动矢量索引　cu_affine_cand_index

跳过模式、直接模式或仿射高级运动矢量表达模式下的仿射模式索引值。解析过程见8.3。解码过程见9.5.7。AffineCandIndex的值等于cu_affine_cand_index的值。如果位流中不存在cu_affine_cand_index，则AffineCandIndex的值为0。

增强时域运动矢量预测模式索引　cu_etmvp_cand_index

增强时域运动矢量预测模式索引值。解析过程见8.3。解码过程见9.5.7。EtmvpCandIndex的值等于cu_etmvp_cand_idex的值。

角度加权预测模式索引　awp_index

角度加权预测模式索引值。解析过程见8.3。如果当前图像为B图像，则AwpIndex的值等于awp_index的值；否则，根据awp_index的值以及当前编码单元的宽度W和高度H得到AwpIndex的值。

```
AwpIndex = awpWxH[awp_index] << 1
```

其中，W和H为8、16、32、64，awpWxH的定义如下：

```
awp8x8 = {8,13,14,15,16,22}
awp8x16 = {4,5,6,12,13,14,15,18,20,21,22,24}
awp8x32 = {0,1,2,4,5,6,8,9,10,12,13,14,15,16,17,18,20,21,22,24,25,26}
awp8x64 = {0,1,2,4,5,6,8,9,10,12,13,14,15,16,17,18,20,21,22,24,25,26}
awp16x8 = {3,4,6,12,13,14,15,18,20,24,26,27}
awp16x16 = {2,3,4,5,6,8,12,13,14,15,16,18,20,21,22,24,26,27}
awp16x32 = {0,1,2,3,4,5,6,8,9,10,12,13,14,15,16,17,18,20,21,22,24,25,26,27}
awp16x64 = {0,1,2,3,4,5,6,8,9,10,12,13,14,15,16,17,18,20,21,22,24,25,26,27}
awp32x8 = {0,2,3,4,6,7,8,10,11,12,13,14,15,16,18,19,20,22,24,26,27}
awp32x16 = {0,1,2,3,4,6,7,8,10,11,12,13,14,15,16,17,18,19,20,21,22,24,25,26,27}
awp32x32 = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,24,25,26,27}
awp32x64 = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,24,25,26,27}
awp64x8 = {0,2,3,4,6,7,8,10,11,12,13,14,15,16,18,19,20,22,23,24,26,27}
awp64x16 = {0,1,2,3,4,6,7,8,10,11,12,13,14,15,16,18,19,20,22,23,24,25,26,27}
awp64x32 = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,26,27}
awp64x64 = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27}
```

如果位流中不存在awp_index，则AwpIndex的值为0。

角度加权预测第一运动矢量修正标志　awp_mvr_cand_flag0

二值变量。值为‘1’表示应修正角度加权预测第一运动矢量，值为‘0’表示不应修正角度加权预测第一运动矢量。解析过程见8.3。AwpMvrCandFlag0的值等于awp_mvr_cand_flag0的值。如果位流中不存在awp_mvr_cand_flag0，则AwpMvrCandFlag0的值为0。

角度加权预测第一运动矢量修正步长索引值　awp_mvr_cand_step0

角度加权预测第一运动矢量修正步长索引值。解析过程见8.3。解码过程见9.5.7。AwpMvrCandStep0的值等于awp_mvr_cand_step0的值。如果位流中不存在awp_mvr_cand_step0，则AwpMvrCandStep0的值为0。

角度加权预测第一运动矢量修正方向索引值　awp_mvr_cand_dir0

角度加权预测第一运动矢量修正方向索引值。解析过程见8.3。解码过程见9.5.7。AwpMvrCandDir0的值等于awp_mvr_cand_dir0的值。如果位流中不存在awp_mvr_cand_dir0，则AwpMvrCandDir0的值为0。

角度加权预测第二运动矢量修正标志　awp_mvr_cand_flag1

二值变量。值为‘1’表示应修正角度加权预测第二运动矢量，值为‘0’表示不应修正角度加权预测第二运动矢量。解析过程见8.3。AwpMvrCandFlag1的值等于awp_mvr_cand_flag1的值。如果位流中不存在awp_mvr_cand_flag1，则AwpMvrCandFlag1的值为0。

角度加权预测第二运动矢量修正步长索引值　awp_mvr_cand_step1

角度加权预测第二运动矢量修正步长索引值。解析过程见8.3。解码过程见9.5.7。AwpMvrCandStep1的值等于awp_mvr_cand_step1的值。如果位流中不存在awp_mvr_cand_step1，则AwpMvrCandStep1的值为0。

角度加权预测第二运动矢量修正方向索引值　awp_mvr_cand_dir1

角度加权预测第二运动矢量修正方向索引值。解析过程见8.3。解码过程见9.5.7。AwpMvrCandDir1的值等于awp_mvr_cand_dir1的值。如果位流中不存在awp_mvr_cand_dir1，则AwpMvrCandDir1的值为0。

角度加权预测模式第一运动信息索引　awp_cand_index0

角度加权预测模式第一运动信息索引值。解析过程见8.3。解码过程见9.5.7。AwpCandIndex0的值等于awp_cand_index0的值。如果位流中不存在awp_cand_index0，则AwpCandIndex0的值为0。

角度加权预测模式第二运动信息索引　awp_cand_index1

角度加权预测模式第二运动信息索引值。解析过程见8.3。解码过程见9.5.7。AwpCandIndex1的值等于awp_cand_index1的值。如果(AwpMvrCandFlag0 == 0 && AwpMvrCandFlag1 == 0) || (AwpMvrCandFlag0 == 1 && AwpMvrCandFlag1 == 1 && AwpMvrCandStep0 == AwpMvrCandStep1 && AwpMvrCandDir0 == AwpMvrCandDir1) && (AwpCandIndex1 >= AwpCandIndex0), AwpCandIndex1等于AwpCandIndex1+1。如果位流中不存在awp_cand_indxe1，则AwpCandIndex1的值为0。

仿射运动矢量偏移量索引　affine_umve_step_index0，affine_umve_step_index1

仿射高级运动矢量表达模式中控制点0和控制点1使用的运动矢量偏移量的索引值。解析过程见8.3。解码过程见9.5.7。AffineUmveStepIndex0和AffineUmveStepIndex1的值分别等于affine_umve_step_index0和affine_umve_step_index1的值。

仿射运动矢量方向索引　affine_umve_dir_index0，affine_umve_dir_index1

仿射高级运动矢量表达模式中控制点0和控制点1使用的运动矢量方向的索引值。解析过程见8.3。解码过程见9.5.7。AffineUmveDirIndex0和AffineUmveDirIndex1的值分别等于affine_umve_dir_index0和affine_umve_dir_index1的值。

空域角度加权预测模式标志　sawp_flag

二值变量。值为‘1’表示当前编码单元是空域角度加权预测模式，值为‘0’表示当前编码单元不是空域角度加权预测模式。解析过程见8.3。SawpFlag的值等于sawp_flag的值。如果位流中不存在sawp_flag，则SawpFlag的值为0。

空域角度加权预测模式索引　sawp_index

空域角度加权预测模式索引值。解析过程见8.3。SawpIndex的值等于sawp_index的值。如果位流中不存在sawp_index，则SawpIndex的值为0。

空域角度加权预测模式第一预测模式索引　sawp_pred_mode0_index

空域角度加权预测模式第一预测模式的索引值。解析过程见8.3。解码过程见9.5.6。sawp_pred_mode0_index的取值范围为0～25。SawpPredMode0Index的值等于sawp_pred_mode0_index的值。

空域角度加权预测模式第二预测模式索引　sawp_pred_mode1_index

空域角度加权预测模式第二预测模式的索引值。解析过程见8.3。解码过程见9.5.6。sawp_pred_mode1_index的取值范围为0～25。SawpPredMode1Index的值等于sawp_pred_mode1_index的值。

衍生模式划分标志　dt_split_flag

二值变量。值为‘1’表示应进行衍生模式划分，值为‘0’表示不应进行衍生模式划分。解析过程见8.3。如果位流中不存在dt_split_flag，则DtSplitFlag的值为0。

衍生模式划分方向　dt_split_dir

二值变量。值为‘1’表示进行水平衍生模式划分，值为‘0’表示进行垂直衍生模式划分。解析过程见8.3。DtSplitDir的值等于dt_split_dir的值。

水平四叉衍生模式划分标志　dt_split_hqt_flag

二值变量。值为‘1’表示应进行水平四叉衍生模式划分，值为‘0’表示不应进行水平四叉衍生模式划分。解析过程见8.3。DtSplitHqtFlag的值等于dt_split_hqt_flag的值。

垂直四叉衍生模式划分标志　dt_split_vqt_flag

二值变量。值为‘1’表示应进行垂直四叉衍生模式划分，值为‘0’表示不应进行垂直四叉衍生模式划分。解析过程见8.3。DtSplitVqtFlag的值等于dt_split_vqt_flag的值。

水平非对称衍生模式标志　dt_split_hadt_flag

二值变量。值为‘0’表示应进行‘SIZE_2MxnU’模式划分，值为‘1’表示应进行‘SIZE_2MxnD’模式划分。解析过程见8.3。DtSplitHadtFlag的值等于dt_split_hadt_flag的值。

垂直非对称衍生模式标志　dt_split_vadt_flag

二值变量。值为‘0’表示应进行‘SIZE_nLx2N’模式划分，值为‘1’表示应进行‘SIZE_nRx2N’模式划分。解析过程见8.3。DtSplitVadtFlag的值等于dt_split_vadt_flag的值。

仿射自适应运动矢量精度索引　affine_amvr_index

编码单元的仿射运动矢量精度。解析过程见8.3，解码过程见9.5.7。AffineAmvrIndex的值等于affine_amvr_index的值。如果位流中不存在affine_amvr_index，则AffineAmvrIndex的值为0。

自适应运动矢量精度索引　amvr_index

编码单元的运动矢量精度。解析过程见8.3，解码过程见9.5.7。AmvrIndex的值等于amvr_index的值。如果位流中不存在amvr_index，则AmvrIndex的值为0。

编码单元子类型索引　cu_subtype_index

当前编码单元的跳过模式或直接模式的运动信息索引值。解析过程见8.3。CuSubtypeIndex的值等于cu_subtype_index的值。如果位流中不存在cu_subtype_index，则CuSubtypeIndex的值为0。如果当前图像是P图像，则cu_subtype_index值的取值范围为0～1+ Max(NumOfMvapCand,NumOfHmvpCand)。如果当前图像是B图像，则cu_subtype_index值的取值范围为0～3+ Max(NumOfMvapCand,NumOfHmvpCand)。

串复制或块复制帧内预测标志　isc_ibc_cu_flag

二值变量。值为‘1’表示当前编码单元是串复制帧内预测模式或块复制帧内预测模式，值为‘0’表示当前编码单元既不是串复制帧内预测模式也不是块复制帧内预测模式。解析过程见8.3。IscIbcCuFlag的值等于isc_ibc_cu_flag的值。如果位流中不存在isc_ibc_cu_flag，则IscIbcCuFlag的值为0。

块复制帧内模式标志　ibc_cu_flag

二值变量。值为‘1’表示当前编码单元是块复制帧内预测模式，值为‘0’表示当前编码单元不是块复制帧内预测模式。解析过程见8.3。IbcCuFlag的值等于ibc_cu_flag。如果位流中不存在ibc_cu_flag，则IbcCuFlag的值为0。

IscCuFlag的值为1表示当前编码单元是串复制帧内预测模式，值为0表示当前编码单元不是串复制帧内预测模式。IscCuFlag的值如下：

```
IscCuFlag = IscIbcCuFlag && (! IbcCuFlag)
```

串复制帧内预测子模式标志　isc_subtype_flag

二值变量。解析过程见 8.3。值为‘1’表示当前编码单元是串复制帧内预测非普通串子模式，值为‘0’表示当前编码单元是串复制帧内预测普通串子模式。IscSubtypeFlag 的值等于 isc_subtype_flag 的值。如果位流中不存在 isc_subtype_flag，则 IscSubtypeFlag 的值按以下方法确定：

```
IscSubtypeFlag = PictureIscEvsUbvsEnableFlag ? 1 : 0
```

预测块矢量类别索引　cbvp_index

当前编码单元预测块矢量索引值。解析过程见8.3。解码过程见9.5.6。CbvpIndex的值等于cbvp_index的值。如果位流中不存在cbvp_index，则CbvpIndex的值为0。

自适应块矢量精度索引　abvr_index

用于确定编码单元的块矢量精度。解析过程见8.3，解码过程见9.5.6。如果abvr_index值为‘0’，则AbvrIndex的值等于0。如果abvr_index值为‘1’，则AbvrIndex的值等于2。如果位流中不存在abvr_index，则AbvrIndex的值为0。

块矢量水平分量差绝对值　mv_diff_x_abs_bv

块矢量垂直分量差绝对值　mv_diff_y_abs_bv

块矢量差值的绝对值。MvDiffXAbsBv 等于 mv_diff_x_abs_bv 的值，MvDiffYAbsBv 等于 mv_diff_y_abs_bv的值。

块矢量水平分量差符号值　mv_diff_x_sign_bv

块矢量垂直分量差符号值　mv_diff_y_sign_bv

块矢量差值的符号位。MvDiffXSignbV的值等于mv_diff_x_sign_bv的值，MvDiffYSignBv的值等于mv_diff_y_sign_bv。如果位流中不存在mv_diff_x_sign_bv或mv_diff_y_sign_bv，则MvDiffXSignBv或MvDiffYSignBv的值为0。

MvDiffXBv和MvDiffYBv的值如下，取值范围为-32768～32767。

$$MvDiffXBv = (-1)^{MvDiffXSignBv} * MvDiffXAbsBv$$

$$MvDiffYBv = (-1)^{MvDiffYSignBv} * MvDiffYAbsBv$$

预测参考模式索引　inter_pred_ref_mode_index

用于确定当前预测单元使用的预测参考模式。解析过程见8.3。解码过程见9.5.3。InterPredRefModeIndex 等于 inter_pred_ref_mode_index 的值。如果位流中不存在inter_pred_ref_mode_index，或当前图像是P图像，则InterPredRefModeIndex的值为0。

对称运动矢量差标志　smvd_flag

二值变量。值为‘1’表示当前预测单元应使用对称运动矢量差模式，值为‘0’表示当前编码单元不应使用对称运动矢量差模式。解析过程见8.3。SmvdFlag的值等于smvd_flag的值。如果位流中不存在smvd_flag，则SmvdFlag的值为0。

如果DistanceIndex-DistanceIndexL0等于DistanceIndexL1-DistanceIndex，则SmvdApplyFlag的值为1；否则，SmvdApplyFlag的值为0。DistanceIndexL0是当前图像的参考图像队列0中第0幅图像的距离索引，DistanceIndexL1是当前图像的参考图像队列1中第0幅图像的距离索引。

运动矢量精度扩展模式标识　extend_mvr_flag

二值变量。值为‘1’表示当前预测单元应使用运动矢量扩展精度模式，值为‘0’表示当前预测单元不应使用运动矢量扩展精度模式。解析过程见8.3。解码过程见9.5.7.6。ExtendMvrFlag的值等于extend_mvr_flag的值。

帧内亮度预测模式索引　intra_luma_pred_mode_index

亮度编码块的帧内预测模式。解析过程见8.3。解码过程见9.5.6。intra_luma_pred_mode_index的取值范围为0～33。IntraLumaPredModeIndex的值等于intra_luma_pred_mode_index的值。

如果NumOfIntraPredBlock的值大于1，从符合本文件的位流中解码得到的IntraLumaPredMode的值不应为33。

亮度帧内预测扩展模式索引　luma_eipm_index

用于确定当前编码单元的亮度帧内预测扩展模式。解析过程见8.3。解码过程见9.5.6.3。LumaEipmIndex的值等于luma_eipm_index的值。如果位流中不存在luma_eipm_index，则LumaEipmIndex的值为0。

帧内色度预测模式索引　intra_chroma_pred_mode_index

用于确定4:2:0格式下当前编码单元中两个色度编码块的帧内预测模式。解析过程见8.3。解码过程见9.5.6.3。IntraChromaPredModeIndex的值等于intra_chroma_pred_mode_index的值。如果位流中不存在intra_chroma_pred_mode_index，则IntraChromaPredModeIndex的值等于-1。如果位流中不存在intra_chroma_pred_mode_index则，IntraChromaPredMode的值为-1。

如果isRedundant（见9.5.6.3）的值为1，则intra_chroma_pred_mode_index的值不应为((TscpmEnableFlag || PmcEnableFlag) ? 5 : 4))。

如果当前编码单元只包含色度编码块，则PriorCuMode的值由当前色度编码块右下角色度样本位置上的亮度样本所在的编码单元的预测类型决定。如果该编码单元的预测类型是普通帧内预测，则PriorCuMode的值为1；否则，如果编码单元的预测类型是块复制帧内预测或串复制帧内预测，PriorCuMode的值为2；否则，如果该编码单元的预测类型是帧间预测，PriorCuMode的值为0。

如果当前编码单元只包含色度编码块且PriorCuMode为1，当前编码单元的预测类型是普通帧内预测，则令其色度编码块对应的普通亮度帧内预测模式IntraLumaPredMode等于当前色度编码块右下角色度样本位置上的亮度样本所在的空域帧内预测信息存储单元的帧内亮度预测模式。

如果当前编码单元只包含色度编码块且PriorCuMode为2，当前编码单元的预测类型是普通帧内预测，则令其色度编码块对应的普通亮度帧内预测模式IntraLumaPredMode等于‘Intra_Luma_DC’。

如果当前编码单元包含亮度编码块和色度编码块，则当前色度编码块对应的普通亮度帧内预测模式IntraLumaPredMode等于编码单元中PredBlockOrder的值为0的预测块的普通亮度预测模式。

如果当前编码单元只包含色度编码块且PriorCuMode为0，当前编码单元的预测类型是帧间预测，则令当前色度编码块的运动信息等于当前色度编码块右下角色度样本位置上的亮度样本所在的空域运动信息存储单元的运动信息。

色度帧内预测扩展模式索引　chroma_eipm_index

用于确定当前编码单元的色度帧内预测扩展模式。解析过程见8.3。解码过程见9.5.6.3。ChromaEipmIndex的值等于chroma_eipm_index。如果位流中不存在chroma_eipm_index，则ChromaEipmIndex的值为0。

色度帧内跨多分量预测模式索引　chroma_pmc_index

用于确定当前编码单元的色度帧内跨多分量预测模式。解析过程见8.3。解码过程见9.5.6.3。ChromaPmcIndex的值等于chroma_pmc_index。如果位流中不存在chroma_pmc_index，则ChromaPmcIndex的值等于PmcEnableFlag的值。

帧内色度跨多分量预测参数索引　chroma_pmc_param_index

用于确定当前编码单元的色度帧内跨多分量预测参数。解析过程见8.3。解码过程见9.5.6.3。ChromaPmcParamIndex的值等于chroma_pmc_param_index。如果位流中不存在chroma_pmc_param_index，则ChromaPmcParamIndex的值为0。

帧内预测滤波标志　intra_pf_flag

二值变量。值为‘1’表示当前编码单元应使用帧内预测滤波，值为‘0’表示当前编码单元不应使用帧内预测滤波。解析过程见8.3。IntraPfFlag的值等于intra_pf_flag的值。如果位流中不存在intra_pf_flag，则IntraPfFlag的值为0。

改进型帧内预测标志　iip_flag

二值变量。值为‘1’表示当前编码单元应使用改进型帧内预测，值为‘0’表示当前编码单元不应使用改进型帧内预测。解析过程见8.3。IipFlag的值等于iip_flag的值。如果位流中不存在iip_flag，则IipFlag的值为0。

L0预测单元参考索引　pu_reference_index_l0

预测单元的L0参考索引。解析过程见8.3。RefIndexL0的值等于pu_reference_index_l0的值。如果位流中不存在pu_reference_index_l0，则RefIndexL0的值为0。pu_reference_index_l0的值应小于NumRefActive[0]的值。

L0运动矢量水平分量差绝对值　mv_diff_x_abs_l0

L0运动矢量垂直分量差绝对值　mv_diff_y_abs_l0

来自参考图像队列0的运动矢量差值的绝对值。解析过程见8.3。MvDiffXAbsL0的值等于mv_diff_x_abs_l0的值，MvDiffYAbsL0的值等于mv_diff_y_abs_l0的值。

L0运动矢量水平分量差符号值　mv_diff_x_sign_l0

L0运动矢量垂直分量差符号值　mv_diff_y_sign_l0

来自参考图像队列0的运动矢量差值的符号位。解析过程见8.3。MvDiffXSignL0的值等于mv_diff_x_sign_l0的值，MvDiffYSignL0的值等于mv_diff_y_sign_l0。如果位流中不存在mv_diff_x_sign_l0或mv_diff_y_sign_l0，MvDiffXSignL0或MvDiffYSignL0的值为0。

MvDiffXL0和MvDiffYL0的计算如下，取值范围为-32768～32767。

$$MvDiffXL0 = (-1)^{MvDiffXSignL0} * MvDiffXAbsL0$$

$$MvDiffYL0 = (-1)^{MvDiffYSignL0} * MvDiffYAbsL0$$

仿射帧间模式L0运动矢量水平分量差绝对值　mv_diff_x_abs_l0_affine

仿射帧间模式L0运动矢量垂直分量差绝对值　mv_diff_y_abs_l0_affine

来自参考图像队列0的运动矢量差值的绝对值。解析过程见8.3。MvDiffXAbsL0Affine的值等于mv_diff_x_abs_l0_affine的值，MvDiffYAbsAffineAffine的值等于mv_diff_y_abs_l0_affine的值。

仿射帧间模式L0运动矢量水平分量差符号值　mv_diff_x_sign_l0_affine

仿射帧间模式L0运动矢量垂直分量差符号值　mv_diff_y_sign_l0_affine

来自参考图像队列0的运动矢量差值的符号位。解析过程见8.3。MvDiffXSignL0Affine的值等于mv_diff_x_sign_l0_affine的值，MvDiffYSignL0Affine的值等于mv_diff_y_sign_l0_affine。如果位流中不存在mv_diff_x_sign_l0_affine或mv_diff_y_sign_l0_affine，则MvDiffXSignL0Affine或MvDiffYSignL0Affine的值为0。

MvDiffXL0Affine和MvDiffYL0Affine的计算如下，取值范围为-32768～32767。

$$MvDiffXL0Affine = (-1)^{MvDiffXSignL0Affine} * MvDiffXAbsL0Affine$$

$$MvDiffYL0Affine = (-1)^{MvDiffYSignL0Affine} * MvDiffYAbsL0Affine$$

L1预测单元参考索引　pu_reference_index_l1

预测单元的L1参考索引。解析过程见8.3。RefIndexL1的值等于pu_reference_index_l1的值。如果位流中不存在pu_reference_index_l1，则RefIndexL1的值为0。pu_reference_index_l1的值应小于NumRefActive[1]的值。

L1运动矢量水平分量差绝对值　mv_diff_x_abs_l1

L1运动矢量垂直分量差绝对值　mv_diff_y_abs_l1

来自参考图像队列1的运动矢量差值的绝对值。解析过程见8.3。MvDiffXAbsL1的值等于mv_diff_x_abs_l1的值，MvDiffYAbsL1的值等于mv_diff_y_abs_l1的值。

L1运动矢量水平分量差符号值　mv_diff_x_sign_l1

L1运动矢量垂直分量差符号值　mv_diff_y_sign_l1

来自参考图像队列1的运动矢量差值的符号位。解析过程见8.3。MvDiffXSignL1的值等于mv_diff_x_sign_l1的值，MvDiffYSignL1的值等于mv_diff_y_sign_l1。如果位流中不存在mv_diff_x_sign_l1或mv_diff_y_sign_l1，则MvDiffXSignL1或MvDiffYSignL1的值为0。

如果SmvdFlag的值为0，MvDiffXL1和MvDiffYL1的计算如下，取值范围为-32768～32767。

$$MvDiffXL1 = (-1)^{MvDiffXSignL1} * MvDiffXAbsL1$$
$$MvDiffYL1 = (-1)^{MvDiffYSignL1} * MvDiffYAbsL1$$

如果SmvdFlag的值为1，MvDiffXL1和MvDiffYL1的计算如下：

$$MvDiffXL1 = Clip3(-32768, 32767, -MvDiffXL0)$$
$$MvDiffYL1 = Clip3(-32768, 32767, -MvDiffYL0)$$

仿射帧间模式L1运动矢量水平分量差绝对值　mv_diff_x_abs_l1_affine

仿射帧间模式L1运动矢量垂直分量差绝对值　mv_diff_y_abs_l1_affine

来自参考图像队列1的运动矢量差值的绝对值。解析过程见8.3。MvDiffXAbsL1Affine的值等于mv_diff_x_abs_l1_affine的值，MvDiffYAbsAffineAffine的值等于mv_diff_y_abs_l1_affine的值。

仿射帧间模式L1运动矢量水平分量差符号值　mv_diff_x_sign_l1_affine

仿射帧间模式L1运动矢量垂直分量差符号值　mv_diff_y_sign_l1_affine

来自参考图像队列1的运动矢量差值的符号位。解析过程见8.3。MvDiffXSignL1Affine的值等于mv_diff_x_sign_l1_affine的值，MvDiffYSignL1Affine的值等于mv_diff_y_sign_l1_affine。如果位流中不存在mv_diff_x_sign_l1_affine或mv_diff_y_sign_l1_affine，则MvDiffXSignL1Affine或MvDiffYSignL1Affine的值为0。

MvDiffXL1Affine和MvDiffYL1Affine的计算如下，取值范围为-32768～32767。

$$MvDiffXL1Affine = (-1)^{MvDiffXSignL1Affine} * MvDiffXAbsL1Affine$$
$$MvDiffYL1Affine = (-1)^{MvDiffYSignL1Affine} * MvDiffYAbsL1Affine$$

双向梯度修正标志　bgc_flag

二值变量。解析过程见8.3。值为‘1’表示当前编码单元应使用双向梯度修正，值为‘0’表示不应使用双向梯度修正。BgcFlag的值等于bgc_flag的值。如果位流中不存在bgc_flag，则BgcFlag的值为0。

双向梯度修正索引　bgc_index

双向梯度修正的索引值。解析过程见8.3。BgcIndex的值等于bgc_index的值。如果位流中不存在bgc_index，则BgcIndex的值为0。BgcIndex的值应为0或1。

变换块系数标志　ctp_zero_flag

二值变量。解析过程见8.3。CtpZeroFlag的值等于ctp_zero_flag的值。如果位流中不存在ctp_zero_flag，则CtpZeroFlag的值为0。

基于位置的变换块标志　pbt_cu_flag

二值变量。解析过程见8.3。PbtCuFlag的值等于pbt_cu_flag的值。如果位流中不存在pbt_cu_flag，则PbtCuFlag的值为0。

编码单元量化参数增量绝对值　cu_qp_delta_abs

当前编码单元量化参数相对于预测量化参数的增量的绝对值。解析过程见8.3。CuQpDeltaAbs的值等于cu_qp_delta_abs的值。

编码单元量化参数增量符号值　cu_qp_delta_sign

当前编码单元量化参数相对于预测量化参数的增量的符号位。解析过程见8.3。CuQpDeltaSign的值等于cu_qp_delta_sign的值。CuQpDeltaAbs的取值范围应为0～(32 + 4*(BitDepth-8))。

CuDeltaQp的计算如下：

$$CuDeltaQp = (-1)^{CuQpDeltaSign} * CuQpDeltaAbs$$

如果位流中不存在cu_qp_delta_abs，CuDeltaQp的值为0。

Cb变换块编码模板　ctp_u

Cr变换块编码模板　ctp_v

ctp_u和ctp_v分别确定Cb变换块和Cr变换块中是否包含编码数据。解析过程见8.3。如果位流中不存在ctp_u，则ctp_u的值为0；如果位流中不存在ctp_v，则ctp_v的值为0。

亮度变换块编码模板　ctp_y[i]

ctp_y[i]确定亮度变换块i（亮度变换块的编号i见9.5.5）是否包含编码数据。解析过程见8.3。如果位流中不存在ctp_y[0]且同时满足以下条件，ctp_y[0]的值为1；否则，ctp_y[0]的值为0。

——CtpZeroFlag 的值为 0；

——PbtCuFlag 的值为 0；

——IntraCuFlag 的值为 0；

——SkipFlag 的值为 0；

——IscCuFlag 的值为 0；

——ctp_u 和 ctp_v 的值均为 0；

——当前编码单元包含亮度分量。

扩展变换跳过标志　enhanced_ts_flag

解析过程见8.3。值为‘1’表示应使用扩展变换跳过，值为‘0’表示不应使用扩展变换跳过。EnhancedTsFlag 的值等于 enhanced_ts_flag 的值。如果位流中不存在 enhanced_ts_flag，则EnhancedTsFlag的值为0。

子块变换标志　sbt_cu_flag

二值变量。值为‘1’表示当前编码单元应使用子块变换，值为‘0’表示当前编码单元不应使用子块变换。解析过程见8.3。SbtCuFlag的值等于sbt_cu_flag的值。如果位流中不存在sbt_cu_flag，则SbtCuFlag的值为0。

子块变换大小标志　sbt_quad_flag

二值变量。值为‘1’表示当前亮度变换块是亮度编码块的1/4大小，值为‘0’表示当前亮度变换块是亮度编码块的1/2大小。解析过程见8.3。SbtQuadFlag的值等于sbt_quad_flag的值。如果位流中不存在sbt_quad_flag，则SbtQuadFlag的值为0。

子块变换方向标志　sbt_dir_flag

二值变量。值为‘1’表示当前亮度编码块的宽度等于亮度变换块的宽度，值为‘0’表示当前亮度编码块的高度等于亮度变换块的高度。解析过程见8.3。SbtDirFlag的值等于sbt_dir_flag的值。如果

位流中不存在sbt_dir_flag，则当SbtQuadFlag为1时，SbtDirFlag的值等于SbtHorQuad的值；当SbtQuadFlag为0时，SbtDirFlag的值等于SbtHorHalf的值。

色度二次变换标志　chroma_st_flag

二值变量。值为‘1’表示当前色度变换块应使用二次变换，值为‘0’表示当前色度变换块不应使用二次变换。解析过程见8.3。ChromaStFlag的值等于chroma_st_flag的值。如果位流中不存在chroma_st_flag，则ChromaStFlag的值为0。

变换跳过标志　ts_cu_flag

二值变量。解析过程见8.3。TsCuFlag的值等于ts_cu_flag的值。如果位流中不存在ts_cu_flag，则TsCuFlag的值为0。

7.2.7　变换块语义

变换块中的扫描表ScanCGInBlk、ScanCoeffInCG、InvScanCoeffInBlk和ScanOrder的定义应符合附录E的规定。

扫描区域右端横坐标　scan_region_x

扫描区域下端纵坐标　scan_region_y

用于确定系数扫描区域，解析过程见8.3。scan_region_x和scan_region_y的值应小于或等于31。ScanRegionX的值等于scan_region_x的值，ScanRegionY的值等于scan_region_y的值。系数扫描区域是以(0,0)为左上角，以(ScanRegionX,ScanRegionY)为右下角的矩形区域。

系数非零标志　significant_flag

用于确定当前系数是否为非零系数，解析过程见8.3。significant_flag值为‘0’表示当前系数为零系数，值为‘1’表示当前系数为非零系数。SignificantFlag的值等于significant_flag的值。

系数绝对值大于1标志　coeff_abs_level_greater1_flag

用于确定当前系数的绝对值是否大于1，解析过程见8.3。coeff_abs_level_greater1_flag的值为‘0’表示当前系数的绝对值为1，值为‘1’表示当前系数的绝对值大于1。CoeffAbsLevelGt1Flag的值等于coeff_abs_level_greater1_flag的值。如果位流中不存在coeff_abs_level_greater1_flag，则CoeffAbsLevelGt1Flag的值等于0。

系数绝对值大于2标志　coeff_abs_level_greater2_flag

用于确定当前系数的绝对值是否大于2，解析过程见8.3。coeff_abs_level_greater2_flag的值为‘0’表示当前系数的绝对值为2，值为‘1’表示当前系数的绝对值大于2。CoeffAbsLevelGt2Flag的值等于coeff_abs_level_greater2_flag的值。如果位流中不存在coeff_abs_level_greater2_flag，则CoeffAbsLevelGt2Flag的值等于0。

系数绝对值大于4标志　coeff_abs_level_greater4_flag

用于确定当前系数的绝对值是否大于4，解析过程见8.3。coeff_abs_level_greater4_flag的值为‘0’表示当前系数的绝对值小于等于4，值为‘1’表示当前系数的绝对值大于4。CoeffAbsLevelGt4Flag的值等于coeff_abs_level_greater4_flag的值。如果位流中不存在coeff_abs_level_greater4_flag，则CoeffAbsLevelGt4Flag的值等于0。

系数绝对值大于8标志　coeff_abs_level_greater8_flag

用于确定当前系数的绝对值是否大于8，解析过程见8.3。coeff_abs_level_greater8_flag的值为‘0’表示当前系数的绝对值小于等于8，值为‘1’表示当前系数的绝对值大于8。CoeffAbsLevelGt8Flag的值等于coeff_abs_level_greater8_flag的值。如果位流中不存在coeff_abs_level_greater8_flag，则CoeffAbsLevelGt8Flag的值等于0。

系数绝对值剩余值　coeff_abs_level_remaining

用于确定当前系数的绝对值的剩余值。解析过程见8.3。CoeffAbsLevelRem的值等于coeff_abs_level_remaining的值。

系数游程　coeff_run

用于确定游程的长度,解析过程见8.3。coeff_run的值不应大于(NumOfCoeff－ScanPosOffset－1),其中NumOfCoeff是当前变换块系数的个数，ScanPosOffset是上一个非零系数的位置。

系数幅值　coeff_level_minus1

用于确定非零量化系数的幅值，解析过程见8.3。

系数符号　coeff_sign

用于确定非零量化系数的正负性，解析过程见8.3。coeff_sign值为‘0’表示当前系数是正数，值为‘1’表示当前系数是负数。CoeffSign的值等于coeff_sign的值。

系数结束符　coeff_last

用于确定是否为非零量化系数块中最后一个非零系数，解析过程见8.3。

增强二次变换标志　est_tu_flag

解析过程见8.3。est_tu_flag的值为‘1’表示当前变换块应使用增强的二次变换，值为‘0’表示当前变换块不应使用增强的二次变换。EstTuFlag的值等于est_tu_flag的值。如果位流中不存在est_tu_flag，则EstTuFlag的值为0。

熵编码脉冲编码调制模式填充位　aec_ipcm_stuffing_bit

填充位。值应为‘1’，解析过程见8.3。

高级熵编码字节对齐填充位　aec_byte_alignment_bit0

填充位。值应为‘0’。

脉冲编码调制模式系数　pcm_coeff

n位无符号整数，n的值等于SamplePrecision的值。用于确定脉冲调制编码模式的系数。

7.2.8　自适应修正滤波参数语义

图像亮度分量自适应修正滤波器个数　alf_filter_num_minus1

alf_filter_num_minus1的值加1表示当前图像亮度分量自适应修正滤波器的个数。alf_filter_num_minus1的取值范围应为0～FilterNum-1。AlfFilterNumMinus1的值等于alf_filter_num_minus1的值。

图像亮度分量自适应修正滤波区域顺序索引　alf_region_order_index

alf_region_order_index表示当前图像亮度分量自适应修正滤波区域顺序的索引值。AlfRegionOrderIndex的值等于alf_region_order_scan_index的值。

图像亮度分量相邻自适应修正滤波区域个数　alf_region_distance[i]

alf_region_distance[i]表示亮度分量第i个自适应修正滤波区域基本单元起始标号与第i-1个自适应修正滤波区域基本单元起始标号间的差值。alf_region_distance[i]的取值范围应为1～FilterNum-1。当位流中不存在alf_region_distance[i]时，如果i等于0，alf_region_distance[i]的值为0;如果i不等于0且AlfFilterNumMinus1的值为FilterNum-1,则alf_region_distance[i]的值为1。

alf_region_distance[i]（i=0～AlfFilterNumMinus1）之和应小于或等于FilterNum-1。

图像亮度分量样本自适应修正滤波器系数　alf_coeff_luma[i][j]

alf_coeff_luma[i][j]表示亮度分量第i个自适应修正滤波滤波器的第j个系数。AlfCoeffLuma[i][j]的值等于alf_coeff_luma[i][j]的值。

如果EalfEnableFlag的值等于0，AlfCoeffLuma[i][j]（j=0～7）的取值范围应为-64～63，AlfCoeffLuma[i][8]的取值范围应为-1088～1071。

如果EalfEnableFlag的值等于1，AlfCoeffLuma[i][j]（j=0～13）的取值范围应为-64～63，且：

——如果AlfLumaShift[i]的值等于5，AlfCoeffLuma[i][14]的取值范围应为-1824～1859；

——否则，如果AlfLumaShift[i]的值等于6，则AlfCoeffLuma[i][14]的取值范围应为-1856～1827；

——否则，如果AlfLumaShift[i]的值等于7，则AlfCoeffLuma[i][14]的取值范围应为-1920～1763。

图像亮度分量样本自适应修正滤波器移位 alf_luma_shift_num_minus6[i]

alf_luma_shift_num_minus6[i]的值加6表示亮度分量第i个自适应滤波器的移位值。AlfLumaShift[i]的值等于alf_luma_shift_num_minus6[i]的值加6。

alf_luma_shift_num_minus6[i]的取值范围应为-1～1。

图像色度分量自适应修正滤波器系数 alf_coeff_chroma[0][j]，alf_coeff_chroma[1][j]

alf_coeff_chroma[0][j]表示Cb分量第j个自适应修正滤波器的系数，alf_coeff_chroma[1][j]表示Cr分量第j个自适应修正滤波器的系数。AlfCoeffChroma[0][j]的值等于alf_coeff_chroma[0][j]的值，AlfCoeffChroma[1][j]的值等于alf_coeff_chroma[1][j]的值。

如果EalfEnableFlag的值等于0，AlfCoeffChroma[i][j]（i=0～1，j=0～7）的取值范围应为-64～63，AlfCoeffChroma[i][8]（i=0～1）的取值范围应为-1088～1071。

如果EalfEnableFlag的值等于1，AlfCoeffChroma[i][j]（i=0～1，j=0～13）的取值范围应为-64～63，且：

——如果AlfChromaShift[i]的值等于5，AlfCoeffChroma[i][14]（i=0～1）的取值范围应为-1824～1859；

——如果AlfChromaShift[i]的值等于6，AlfCoeffChroma[i][14]（i=0～1）的取值范围应为-1856～1827；

——如果AlfChromaShift[i]的值等于7，AlfCoeffChroma[i][14]（i=0～1）的取值范围应为-1920～1763。

图像色度分量样本自适应修正滤波器移位 alf_chroma_shift_num_minus6[i]

alf_chroma_shift_num_minus6[0]的值加6表示Cb分量自适应滤波器的移位值，AlfChromaShift[0]的值等于alf_chroma_shift_num_minus6[0]的值加6。alf_chroma_shift_num_minus6[1]的值加6表示Cr分量自适应滤波器的移位值，AlfChromaShift[1]的值等于alf_chroma_shift_num_minus6[1]的值加6。alf_chroma_shift_num_minus6[0]和alf_chroma_shift_num_minus6[1]的取值范围应为-1～1。

7.2.9 增强样值偏移自适应补偿参数语义

增强样值偏移自适应补偿参数中compIndex等于0表示亮度分量，等于1表示Cb分量，等于2表示Cr分量。

图像增强样值偏移自适应补偿允许标志 picture_esao_enable_flag[compIndex]

二值变量。值为‘1’表示当前图像的compIndex分量可使用增强样值偏移自适应补偿，值为‘0’表示当前图像的compIndex分量不应使用增强样值偏移自适应补偿。PictureEsaoEnableFlag[compIndex]的值等于picture_esao_enable_flag[compIndex]的值。

图像增强样值偏移自适应补偿最大编码单元控制标志 picture_esao_lcu_control_flag[compIndex]

二值变量。值为‘0’表示当前图像的compIndex分量的所有最大编码单元均根据PictureEsaoEnableFlag[compIndex]的值决定是否使用增强样值偏移自适应补偿，值为‘1’表示当前图像的compIndex分量的每一个最大编码单元单独决定是否使用增强样值偏移自适应补偿。PictureEsaoLcuControlFlag[compIndex]的值等于picture_esao_lcu_control_flag[compIndex]的值。如果位流中不存在picture_esao_lcu_control_flag[compIndex]，则PictureEsaoLcuControlFlag[compIndex]的值位0。

图像增强样值偏移自适应补偿偏移值系数 picture_esao_set_minus1[compIndex]

1位无符号整数。表示当前图像的第compIndex个分量的偏移值组数，取值范围为0～1。PictureEsaoSetNum[compIndex]的值等于picture_esao_set_num_minus1[compIndex]的值加1。如果位流中不存在picture_esao_set_num_minus1[compIndex]，则PictureEsaoSetNum[compIndex]的值为1。

图像增强样值偏移自适应补偿自适应参数 picture_esao_adaptive_param_minus1[setIndex]

4位无符号整数。用于确定当前图像的compIndex分量的增强样值偏移自适应补偿自适应参数值。取值范围为0～15。PictureEsaoAdaptiveParam[setIndex]的值等于picture_esao_adaptive_param_minus1[setIndex]的值加1。

增强样值偏移自适应补偿亮度参数类型 esao_luma_param_type[setIndex]

1位无符号整数。确定当前图像亮度分量的增强样值偏移自适应补偿参数类型。EsaoLumaParamType[setIndex]的值等于esao_luma_param_type[setIndex]的值。

图像增强样值偏移自适应补偿自适应参数索引值 picture_esao_adaptive_param_index[compIndex][setIndex]

7位无符号整数。当前图像compIndex分量的增强样值偏移自适应补偿自适应参数索引值。取值范围为0～95。PictureEsaoAdaptiveParamIndex[compIndex][setIndex]的值等于picture_esao_adaptive_param_index[compIndex][setIndex]的值。

增强样值偏移自适应补偿色度子带模式标志 esao_chroma_band_mode_flag[compIndex][setIndex]

1位无符号整数。值为‘1‘表示当前图像的compIndex分量应使用子带模式，值为‘0’表示当前图像的compIndex分量不应使用子带模式。EsaoChromaBandModeFlag[compIndex][setIndex]的值等于esao_chroma_band_mode_flag[compIndex][setIndex]的值。

增强样值偏移自适应补偿色度起始子带 esao_chroma_start_band[compIndex][setIndex]

表示当前图像的compIndex分量的起始子带。解析过程见8.2。EsaoChromaStartBand[compIndex][setIndex]的值等于esao_chroma_start_band[compIndex][setIndex]的值。

增强样值偏移自适应补偿色度子带数 esao_chroma_band_num[compIndex][setIndex]

表示当前图像的compIndex分量的子带个数。解析过程见8.2。EsaoChromaBandNum[compIndex][setIndex]的值等于esao_chroma_band_num[compIndex][setIndex]的值。

增强样值偏移自适应补偿系数 esao_filter_coeff[compIndex][setIndex][i]

表示当前图像的compIndex分量的增强样值偏移自适应补偿系数。解析过程见8.2。取值范围为-15～15。EsaoFilterCoeff[compIndex][setIndex][i]的值等于esao_filter_coeff[compIndex][setIndex][i]的值。如果码流中不存在esao_filter_coeff[compIndex][setIndex][i],则EsaoFilterCoeff[compIndex][setIndex][i]的值为0。

7.2.10 跨分量样值偏移自适应补偿参数语义

跨分量样值偏移自适应补偿参数中compIndex等于1表示Cb分量，等于2表示Cr分量。

图像跨分量样值偏移自适应补偿允许标志 picture_ccsao_enable_flag[compIndex]

二值变量。值为‘1’表示当前图像的compIndex分量可使用跨分量样值偏移自适应补偿，值为‘0’表示当前图像的compIndex分量不应使用跨分量样值偏移自适应补偿。PictureCcsaoEnableFlag[compIndex]的值等于picture_ccsao_enable_flag[compIndex]的值。如果位流中不存在picture_ccsao_enable_flag[compIndex],则PictureCcsaoEnableFlag[compIndex]的值为0。

图像跨分量样值偏移自适应补偿最大编码单元控制标志 picture_ccsao_lcu_control_flag[compIndex]

二值变量。值为‘0’表示当前图像的compIndex分量的所有最大编码单元均根据PictureCcsaoEnableFlag[compIndex]的值决定是否使用跨分量样值偏移自适应补偿，值为‘1’表示当前图像的compIndex分量的每一个最大编码单元单独决定是否使用跨分量样值偏移自适应补偿。PictureCcsaoLcuControlFlag[compIndex]的值等于picture_ccsao_lcu_control_flag[compIndex]的值。如果位流中不存在picture_ccsao_lcu_control_flag[compIndex]，则PictureCcsaoLcuControlFlag[compIndex]的值为0。

图像跨分量样值偏移自适应补偿偏移值组数 picture_ccsao_set_num_minus1[compIndex]

2位无符号整数。表示当前图像的第compIndex个分量的偏移值组数，取值范围为0～3。PictureCcsaoSetNum[compIndex]的值等于picture_ccsao_set_num_minus1[compIndex]的值加1。如果位流中不存在picture_ccsao_set_num_minus1[compIndex]，则PictureCcsaoSetNum[compIndex]的值为1。

图像跨分量样值偏移自适应补偿候选索引 picture_ccsao_cand_index[compIndex][setIndex]

4位无符号整数。用于确定当前图像跨分量样值偏移自适应补偿所使用的亮度样本的位置，取值范围为0～8。PictureCcsaoCandIndex[compIndex][setIndex]的值等于picture_ccsao_cand_index[compIndex][setIndex]的值。

图像跨分量样值偏移自适应补偿亮度条带个数 picture_ccsao_luma_band_num_minus1[compIndex][setIndex]

4位无符号整数。当前图像的compIndex分量选择的补偿亮度条带个数，取值范围为0～15。PictureCcsaoLumaBandNum[compIndex][setIndex]的值等于picture_ccsao_luma_band_num_minus1[compIndex][setIndex]的值加1。

图像跨分量样值偏移自适应补偿色度条带个数 picture_ccsao_chroma_band_num_minus1[compIndex][setIndex]

1位无符号整数。当前图像的compIndex分量选择的补偿色度条带个数，取值范围为0～1。PictureCcsaoChromaBandNum[compIndex][setIndex]的值等于picture_ccsao_chroma_band_num_minus1[compIndex][setIndex]的值加1。PictureCcsaoClassNum[compIndex][setIndex]的值等于PictureCcsaoLumaBandNum[compIndex][setIndex]的值乘PictureCcsaoChromaBandNum [compIndex][setIndex]的值。

跨分量样值偏移自适应补偿偏移值 ccsao_offset[compIndex][setIndex][i]

表示当前图像的compIndex分量的跨分量样值偏移自适应补偿偏移值，取值范围为-15～15。解析过程见8.2。CcsaoOffset[compIndex][setIndex][i]的值等于ccsao_offset[compIndex][setIndex][i]的值。

图像跨分量样值偏移自适应补偿模式 picture_ccsao_mode[compIndex][setIndex]

1位无符号整数。表示当前图像的compIndex分量的跨分量样值偏移自适应补偿模式。值为0表示当前图像的跨分量样值偏移自适应补偿模式为区间模式，值为1表示当前图像的跨分量样值偏移自适应补偿模式为边缘模式。PictureCcsaoMode[compIndex][setIndex]的值等于picture_ccsao_mode[compIndex][setIndex]的值。

图像跨分量样值偏移自适应补偿边缘模式类型 picture_ccsao_edge_type[compIndex][setIndex]

2位无符号整数。表示当前图像的compIndex分量的跨分量样值偏移自适应补偿边缘模式的类型，取值范围为0～3。PictureCcsaoEdgeType[compIndex][setIndex]的值等于picture_ccsao_edge_type[compIndex][setIndex]的值。

图像跨分量样值偏移自适应补偿边缘模式条带索引 picture_ccsao_edge_band_num_index[compIndex][setIndex]

2位无符号整数。表示当前图像的compIndex分量的跨分量样值偏移自适应补偿边缘模式选择的补偿像素类型以及条带个数，取值范围为0～3。PictureCcsaoEdgeBandNumIndex[compIndex][setIndex]的值等于picture_ccsao_edge_band_num_index[compIndex][setIndex]的值。

按以下方式导出PictureCcsaoClassNum[compIndex][setIndex]：

```
if (PictureCcsaoEdgeBandNumIndex[compIndex][setIndex] == 0 ||
PictureCcsaoEdgeBandNumIndex[compIndex][setIndex] == 1) {
    PictureCcsaoEdgeBandNum[compIndex][setIndex] = PictureCcsaoEdgeBandNumIndex[compIndex][setIndex] + 1
}
else {
    PictureCcsaoEdgeBandNum[compIndex][setIndex] = PictureCcsaoEdgeBandNumIndex[compIndex][setIndex] - 1
}
PictureCcsaoClassNum[compIndex][setIndex] = PictureCcsaoEdgeBandNum[compIndex][setIndex] * 16
```

图像跨分量样值偏移自适应补偿边缘模式阈值索引 picture_ccsao_edge_thr_index[compIndex][setIndex]

3位无符号整数。表示当前图像的compIndex分量的跨分量样值偏移自适应补偿边缘模式的阈值索引，取值范围为0～7。PictureCcsaoEdgeThr[compIndex][setIndex]的值等于(picture_ccsao_edge_thr_index[compIndex][setIndex]+1)*2。

7.2.11 串复制帧内预测语义

串复制帧内预测中扫描表TravScan[4][4][1024][2]按附录E生成。

串复制帧内预测的匹配类型 isc_match_type[i]

解析过程见8.3。值为‘1’表示当前编码单元的第i部分是一个匹配串，值为‘0’表示第i部分是一个不完全匹配串。IscMatchType[i]等于isc_match_type[i]的值。如果位流中不存在isc_match_type[i]，则IscMatchType[i]的值为0。

剩余样本数量 isc_next_remaining_pixel_in_cu[i]

完成当前编码单元第i部分的解码之后当前编码单元中剩余的尚未完成解码的像素的数目。解析过程见8.3。IscNextRemainingPixelInCu[i]的值等于isc_next_remaining_pixel_in_cu[i]的值。

样本匹配类型标志 isc_pixel_match_type[i][j]

解析过程见8.3。值为‘1’表示当前编码单元的第i部分的第j个像素是一个匹配样本，值为‘0’表示是一个未匹配样本。IscPixelMatchType[i][j]等于isc_pixel_match_type[i][j]的值。如果位流中不存在isc_pixel_match_type[i][j]，则IscPixelMatchType[i][j]的值为1。

串复制帧内预测未匹配样本亮度分量值 isc_unmatched_pixel_y[i][j]

串复制帧内预测未匹配样本色度Cb分量值 isc_unmatched_pixel_cb[i][j]

串复制帧内预测未匹配样本色度Cr分量值 isc_unmatched_pixel_cr[i][j]

当前编码单元第i部分的第j个未匹配像素的亮度分量、色度Cb分量和色度Cr分量的值。解析过程见8.3。IscUnmatchedPixelY[i][j]、IscUnmatchedPixelCb[i][j]和IscUnmatchedPixelCr[i][j]分别等于isc_unmatched_pixel_y[i][j]、isc_unmatched_pixel_cb[i][j]和isc_unmatched_pixel_cr[i][j]的值。

串矢量上方标志 isc_sv_above_flag[i]

解析过程见8.3。值为‘1’表示当前编码单元第i部分的串矢量是(0,-1)，值为‘0’表示串矢量不是(0,-1)。IscSvAboveFlag[i] 的值等于isc_sv_above_flag[i]的值。当isc_sv_above_flag[i]的值等于0时，不允许isc_sv_y_abs的值等于1、isc_sv_y_sign的值等于1且isc_sv_x_non_zero_flag的值等于0。

串矢量历史标志 isc_sv_recent_flag[i]

解析过程见8.3。值为‘1’表示应从历史帧内复制信息表中导出当前编码单元第i部分的串矢量，值为‘0’表示不应从历史帧内复制信息表中导出串矢量。IscSvRecentFlag[i] 的值等于isc_sv_recent_flag[i]的值。如果位流中不存在 isc_sv_recent_flag [i]，则 IscSvRecentFlag[i]的值为0。

如果IscSvAboveFlag[i]和IscSvRecentFlag[i]的值均为0，则当前编码单元第i部分的串矢量的X分量的值由IscSvXNonZeroFlag[i]、IscSvXSign[i]和IscSvXAbs[i]确定，Y分量的值由IscSvYSign[i]和IscSvYAbs[i]确定。

串矢量历史索引 isc_sv_recent_index[i]

当前编码单元第i部分的串矢量在历史帧内复制信息表中的索引值。解析过程见8.3。解码过程见9.5.6.5.2。IscSvRecentIndex[i]的值等于isc_sv_recent_index[i]的值。

串矢量Y分量绝对值 isc_sv_y_abs[i]

当前编码单元第 i 部分的串矢量的 Y 分量的绝对值。解析过程见 8.3。IscSvYAbs[i]的值等于isc_sv_y_abs[i]的值。

串矢量Y分量符号位 isc_sv_y_sign[i]

当前编码单元第i部分的串矢量的Y分量的符号位。值为‘1’表示Y分量的值为负数，值为‘0’表示Y分量的值为正数。解析过程见8.3。IscSvYSign[i]的值等于isc_sv_y_sign[i]的值。如果位流中不存在isc_sv_y_sign[i]，则IscSvYSign[i]的值为1。

串矢量X分量符号位 isc_sv_x_sign[i]

当前编码单元第i部分的串矢量的X分量的符号位。值为‘1’表示X分量的值为负数，值为‘0’表示X分量的值为正数。解析过程见8.3。IscSvXSign[i]的值等于isc_sv_x_sign[i]的值。如果位流中不存在isc_sv_x_sign[i]，则IscSvXSign[i]的值为1。

串矢量X分量绝对值 isc_sv_x_abs_minus1[i]

当前编码单元第i部分的串矢量的X分量的绝对值减1。解析过程见8.3。IscSvXAbs[i] 的值等于isc_sv_x_abs_minus1[i]的值加1。如果位流中不存在isc_sv_x_abs_minus1[i]，则IscSvXAbs[i]的值等于0。

串矢量X分量标志 isc_sv_x_non_zero_flag[i]

解析过程见8.3。值为‘1’表示当前编码单元第i部分的串矢量的X分量的值不等于0，值为‘0’表示X分量的值等于0。IscSvXNonZeroFlag[i] 的值等于isc_sv_x_non_zero_flag[i]的值。如果位流中不存在isc_sv_x_non_zero_flag[i]，则IscSvXNonZeroFlag[i]的值为1。

串复制帧内预测未匹配样本串存在标志 isc_ups_present_flag

二值变量。解析过程见8.3。值为‘1’表示当前编码单元中存在未匹配样本串，值为‘0’表示当前编码单元中不存在未匹配样本串。IscUpsPresentFlag 的值等于 isc_ups_present_flag 的值。如果位流中不存在isc_ups_present_flag，则IscUpsPresentFlag的值为0。

新增点矢量数 isc_num_of_new_pv

当前编码单元中新增点矢量的个数。解析过程见8.3。IscNumOfNewPv 的值等于isc_num_of_new_pv的值。如果位流中不存在isc_num_of_new_pv，则IscNumOfNewPv 的值为0。

复用点矢量数 isc_num_of_reused_pv

表示当前编码单元中复用历史点预测历史运动信息表PrevPpInfoList中的点矢量数。解析过程见

8.3。IscNumOfReusedPv 的值等于 isc_num_of_reused_pv 的值。如果位流中不存在 isc_num_of_reused_pv，则 IscNumOfReusedPv 的值为 0。

历史点预测信息表中连续排列的不复用点矢量数　isc_prev_pv_not_reused_run

表示在从历史点预测信息表 PrevPpInfoList 中选择复用的点矢量时，连续排列的不复用的点矢量数。解析过程见 8.3。IscPrevPvNotReusedRun 的值等于 isc_prev_pv_not_reused_run 的值。如果位流中不存在 isc_prev_pv_not_reused_run，则 IscPrevPvNotReusedRun 的值为 0。

未匹配样本串长度最大值　isc_ups_max_length_minus1

isc_ups_max_length_minus1 的值加 1 表示当前编码单元中所有未匹配样本串长度的最大值。解析过程见 8.3。IscUpsMaxLengthMinus1 的值等于 isc_ups_max_length_minus1 的值。如果位流中不存在 isc_ups_max_length_minus1，则 IscUpsMaxLengthMinus1 的值为 0。

等值串或未匹配样本串数　isc_evs_ups_num_minus1

用于确定连续的等值串或未匹配样本串的个数。解析过程见 8.3。IscEvsUPSNumMinus1 的值等于 isc_evs_ups_num_minus1 的值。如果位流中不存在 isc_evs_ups_num_minus1，则 IscEvsUPSNumMinus1 的值为 0。

串长度第一部分　str_length_minus1_prefix

串长度的第一部分。解析过程见 8.3。StrLengthMinus1Prefix 等于 str_length_minus1_prefix 的值。如果位流中不存在 str_length_minus1_prefix，则 StrLengthMinus1Prefix 的值为 0。

串长度第二部分　str_length_minus1_infix

串长度的第二部分。解析过程见 8.3。StrLengthMinus1Infix 等于 str_length_minus1_infix 的值。如果位流中不存在 str_length_minus1_infix，则 StrLengthMinus1Infix 的值为 0。

串长度第三部分　str_length_minus1_suffix

串长度的第三部分。解析过程见 8.3。StrLengthMinus1Suffix 等于 str_length_minus1_suffix 的值。如果位流中不存在 str_length_minus1_suffix，则 StrLengthMinus1Suffix 的值为 0。

点矢量在点预测信息表中索引值的第一部分　pv_address_prefix

点矢量索引值的第一部分，解析过程见 8.3。PvAdressPrefix 等于 pv_address_prefix 的值。如果位流中不存在 pv_address_prefix，则 PvAdressPrefix 的值为 0。

点矢量在点预测信息表中索引值的第二部分　pv_address_infix

点矢量索引值的第二部分，解析过程见 8.3。PvAdressInfix 等于 pv_address_infix 的值。如果位流中不存在 pv_address_infix，则 PvAdressInfix 的值为 0。

点矢量在点预测信息表中索引值的第三部分　pv_address_suffix

点矢量索引值的第三部分，解析过程见 8.3。PvAdressSuffix 等于 pv_address_suffix 的值。如果位流中不存在 pv_address_suffix，则 PvAdressSuffix 的值为 0。

新增点矢量对应常现位置像素亮度分量值　isc_fopixel_y

新增点矢量对应常现位置像素Cb分量值　isc_fopixel_cb

新增点矢量对应常现位置像素Cr分量值　isc_fopixel_cr

新增点矢量对应的常现位置上像素的亮度分量、色度Cb分量和色度Cr分量的值。解析过程见8.3。

串复制帧内预测未匹配样本亮度分量值　isc_nos_up_y[i]

串复制帧内预测未匹配样本色度Cb分量值　isc_nos_up_cb[i]

串复制帧内预测未匹配样本色度Cr分量值　isc_nos_up_cr[i]

非普通串子模式中的当前编码单元第i部分的未匹配像素的亮度分量、色度Cb分量和色度Cr分量的值。解析过程见8.3。IscNosUpY[i]、IscNosUpCb[i]和IscNosUpCr[i]的值分别等于isc_nos_up_y[i]、isc_nos_up_cb[i]和isc_nos_up_cr[i]的值。

8 解析过程

8.1 k 阶指数哥伦布码

k阶指数哥伦布码的结构应符合GY/T 257.1—2012和GY/T 299.1—2016的规定。解析k阶指数哥伦布码时，首先从位流的当前位置开始寻找第一个非零位，并将找到的零位个数记为leadingZeroBits，然后根据leadingZeroBits计算CodeNum。用伪代码描述如下：

```
leadingZeroBits = -1
for (b=0; ! b; leadingZeroBits++) {
    b = read_bits(1)
}
CodeNum = 2^(leadingZeroBits+k) - 2^k + read_bits(leadingZeroBits + k)
```

0阶、1阶、2阶和3阶指数哥伦布码的结构应符合表58的规定。指数哥伦布码的位串分为“前缀”和“后缀”两部分。前缀由leadingZeroBits个连续的‘0’和一个‘1’构成。后缀由leadingZeroBits + k个位构成，即表中的x_i串，x_i的值为‘0’或‘1’。

表58 k 阶指数哥伦布码

阶数	码字结构	CodeNum取值范围
k=0	1	0
	0 1 x_0	1～2
	0 0 1 x_1 x_0	3～6
	0 0 0 1 x_2 x_1 x_0	7～14
	.....	.....
k=1	1 x_0	0～1
	0 1 x_1 x_0	2～5
	0 0 1 x_2 x_1 x_0	6～13
	0 0 0 1 x_3 x_2 x_1 x_0	14～29
	.....	.....
k=2	1 x_1 x_0	0～3
	0 1 x_2 x_1 x_0	4～11
	0 0 1 x_3 x_2 x_1 x_0	12～27
	0 0 0 1 x_4 x_3 x_2 x_1 x_0	28～59
	.....	.....
k=3	1 x_2 x_1 x_0	0～7
	0 1 x_3 x_2 x_1 x_0	8～23
	0 0 1 x_4 x_3 x_2 x_1 x_0	24～55
	0 0 0 1 x_5 x_4 x_3 x_2 x_1 x_0	56～119
	.....	.....

8.2 ue(v)和 se(v)的解析过程

ue(v)和se(v)描述的语法元素使用0阶指数哥伦布码，它们的解析过程为：

——ue(v)：语法元素的值等于 CodeNum；

——se(v)：根据表 59 给出的有符号指数哥伦布码的映射关系求语法元素的值。

表59　se(v)与 CodeNum 的映射关系

CodeNum值	语法元素值
0	0
1	1
2	-1
3	2
4	-2
5	3
6	-3
k	$(-1)^{k+1}$*Ceil(k÷2)

8.3　ae(v)的解析过程

8.3.1　概述

对片进行解析前，首先初始化所有二元符号模型和熵编码解码器，见8.3.2。然后从位流中依次解析得到二元符号串，见8.3.3。最后根据二元符号串得到ae(v)描述的语法元素的值，见8.3.4。

如果存在以下任意一种情况，则初始化熵编码解码器，见8.3.2.2：

——当前编码单元只包含亮度分量且 IntraLumaPredMode 等于‘Intra_Luma_PCM’，或当前编码单元包含亮度分量和色度分量且 IntraLumaPredMode 等于‘Intra_Luma_PCM’且 IntraChromaPredMode 不等于‘Intra_Chroma_PCM’，在解析完亮度编码块的 pcm_coeff 后，初始化熵编码解码器；

——当前编码单元只包含色度分量且 IntraChromaPredMode 等于‘Intra_Chroma_PCM’，或当前编码单元包含亮度分量和色度分量且 IntraLumaPredMode 等于‘Intra_Luma_PCM’且 IntraChromaPredMode 等于‘Intra_Chroma_PCM’，在解析完色度编码块的 pcm_coeff 后，初始化熵编码解码器。

8.3.2　初始化

8.3.2.1　初始化二元符号模型

解码器应初始化ctxArray中的每个二元符号模型。

如果MaecEnableFlag的值为0，则一个二元符号模型包括变量mps、cycno和lgPmps。mps的位宽为1位，cycno的位宽为2位，lgPmps的位宽为11位。mps和cycno的值应初始化为0，lgPmps的值应初始化为1023。

否则，一个二元符号模型包括变量mps、cycno、counterThr1、counterThr2、lgPmps0、lgPmps1、diffWin0和diffWin1。mps的位宽为1位，cycno的位宽为5位，lgPmps0和lgPms1的位宽为10位。mps和cycno的值应初始化为0，lgPmps0和lgPms1的值应初始化为1023。

二元符号模型中，diffWin0和diffWin1的值均应为1，baseWin的值应为5，counterThr1和counterThr2的值由二元符号所在当前图像的图像类型得到：

——如果当前图像是 I 图像，counterThr1 的值为 0，counterThr2 的值为 8；

——如果当前图像是P图像或B图像，counterThr1的值为3，counterThr2的值为16。

8.3.2.2 初始化熵编码解码器

rS1、rT1、bFlag、cFlag、valueS、boundS、valueT和valueD是用于熵编码解码器的变量。boundS是一个大于0的整数。valueD的值为0或1，bFlag的值为0或1，cFlag的值为0或1。valueS和boundS的位宽是大于或等于Log(boundS+1)的最小整数，rS1的位宽是大于或等于Log(boundS+2)的最小整数，rT1的位宽是8位，valueT的位宽是9位。rS1的值应初始化为0，rT1的值应初始化为0xFF。如果boundS的值为254，则valueS和rS1的位宽是8位。

注：valueS记录了预先连续读进多少位才会使valueT的最高位为1。这一功能可用于快速读取位流及并行解码。在极端的状况下，valueS的值有可能超过16位可表示的范围。boundS限制了连续读进‘0’的个数，从而对valueS的位宽进行合理的控制。

valueS、valueT和valueD的初始化过程用伪代码描述如下：

```
valueS=0
valueT=read_bits(9)
valueD=1
```

8.3.3 二元符号串解析

8.3.3.1 概述

解析二元符号串的步骤如下。

a) 设二元符号的索引号binIndex的值为-1，二元符号串为空。

b) binIndex的值加1，然后进行以下操作。

1) 如果当前二元符号是以下二元符号之一，置BypassFlag的值为1：

- sao_mode中binIndex不为0的二元符号；
- sao_interval_offset_abs中binIndex不为0的二元符号；
- sao_interval_offset_sign的二元符号；
- sao_edge_offset的二元符号；
- sao_interval_start_pos的二元符号；
- sao_interval_delta_pos_minus2的二元符号；
- sao_edge_type的二元符号；
- ccsao_lcu_set_index的二元符号；
- mv_diff_x_sign_l0、mv_diff_y_sign_l0、mv_diff_x_sign_l0_affine、mv_diff_y_sign_l0_affine、mv_diff_x_sign_l1、mv_diff_y_sign_l1、mv_diff_x_sign_l1_affine、mv_diff_y_sign_l1_affine、mv_diff_x_sign_bv或mv_diff_y_sign_bv的二元符号；
- mv_diff_x_abs_l0、mv_diff_y_abs_l0、mv_diff_x_abs_l0_affine、mv_diff_y_abs_l0_affine、mv_diff_x_abs_l1、mv_diff_y_abs_l1、mv_diff_x_abs_l1_affine或mv_diff_y_abs_l1_affine中binIndex大于或等于3的二元符号；
- mv_diff_x_abs_bv或mv_diff_y_abs_bv或isc_sv_y_abs[i]中binIndex等于6且前一位二元符号等于0的二元符号；
- mv_diff_x_abs_bv或mv_diff_y_abs_bv或isc_sv_y_abs[i]中binIndex大于或等于7的二元符号；

- coeff_sign 的二元符号；
- coeff_run 中 binIndex 大于或等于 16 的二元符号；
- umve_mv_index 中 binIndex 不为 0 的二元符号；
- umve_step_index 中 binIndex 不为 0 的二元符号；
- sawp_index 的二元符号；
- sawp_pred_mode0_index 中 binIndex 不为 0 的二元符号；
- sawp_pred_mode1_index 中 binIndex 不为 0 的二元符号；
- awp_index 的二元符号；
- awp_mvr_cand_step0 中 binIndex 不为 0 的二元符号；
- awp_mvr_cand_step1 中 binIndex 不为 0 的二元符号；
- awp_cand_index0 中 binIndex 不为 0 的二元符号；
- awp_cand_index1 中 binIndex 不为 0 的二元符号；
- chroma_eipm_index 中 binIndex 不为 0 的二元符号；
- affine_umve_step_index0 中 binIndex 不为 0 的二元符号；
- affine_umve_step_index1 中 binIndex 不为 0 的二元符号；
- coeff_level_minus1 中 binIndex 大于或等于 8 的二元符号；
- scan_region_x 中第二部分的二元符号（第一、二部分的定义见 8.3.4.20）；
- scan_region_y 中第二部分的二元符号（第一、二部分的定义见 8.3.4.20）；
- coeff_abs_level_remaining 中的二元符号；
- isc_next_remaining_pixel_in_cu[i]的第二和第三部分的二元符号（第一、二、三部分的定义见 8.3.4.24）；
- cu_qp_delta_sign 的二元符号；
- esao_lcu_set_index 的二元符号；
- isc_sv_recent_index[i]中 binIndex 大于或等于 3 的二元符号；
- isc_sv_x_sign[i]或 isc_sv_y_sign[i]的二元符号；
- isc_sv_x_abs_minus1[i]的二元符号；
- isc_unmatched_pixel_y[i][j] 、 isc_unmatched_pixel_cb[i][j] 或 isc_unmatched_pixel_cr[i][j]的二元符号；
- isc_nos_up_y[i]、isc_nos_up_cb[i]或 isc_nos_up_cr[i]的二元符号；
- isc_num_of_reused_pv 的二元符号；
- isc_num_of_new_pv 的二元符号；
- isc_prev_pv_not_reused_run 的二元符号；
- str_length_minus1_prefix 中 binIndex 大于或等于 5 的二元符号；
- str_length_minus1_infix 或 str_length_minus1_suffix 的二元符号；
- pv_address_prefix、pv_address_infix 或 pv_address_suffix 的二元符号；
- isc_fopixel_y、isc_fopixel_cb、isc_fopixel_cr 的二元符号。

2) 否则，如果当前二元符号是以下二元符号之一，置 BypassFlag 的值为 0 且 StuffingBitFlag 的值为 1：
   - aec_lcu_stuffing_bit 的二元符号；
   - aec_ipcm_stuffing_bit 的二元符号。
3) 否则，置 BypassFlag 和 StuffingBitFlag 的值为 0，根据 binIndex 得到每个二元符号对应的唯一的 ctxIndex，并根据 ctxIndex 导出二元符号模型 ctx（见 8.3.3.2）。

c) 如果当前二元符号是 coeff_last，置 CtxWeight 的值为 1；否则，置 CtxWeight 的值为 0。

d) 解析当前二元符号（见 8.3.3.3）。
e) 将由步骤 c)得到的二元符号加入二元符号串的尾部，得到更新的二元符号串。
f) 将由步骤 d)得到的二元符号串与 8.3.4 中对应的表格进行比较。如果该二元符号串与表格中某个二元符号串相匹配，则完成二元符号串的解析；否则回到步骤 b)，继续解析下一个二元符号。

8.3.3.2 导出二元符号模型

8.3.3.2.1 导出二元符号模型

如果ctxIndexIncW不存在，二元符号模型ctx等于ctxArray[ctxIndex]，其中ctxArray是保存二元符号模型的数组,ctxIndex是数组的索引值;否则,二元符号模型ctx和ctxW分别等于ctxArray[ctxIndex]和ctxArray[ctxIndexW]，其中ctxArray是保存二元符号模型的数组，ctxIndex和ctxIndexW是数组的索引值。语法元素的每个二元符号的ctxIndex等于ctxIndexInc加ctxIndexStart，ctxIndexW等于ctxIndexIncW加ctxIndexStart。各语法元素对应的ctxIndexStart及每个二元符号对应的ctxIndexInc见表60，ctxIndexIncW见8.3.3.2.27。

表60 语法元素对应的 ctxIndexStart 和 ctxIndexInc

语法元素	ctxIndexInc	ctxIndexStart	ctx的数量
lcu_qp_delta	见8.3.3.2.2	0	4
sao_merge_type_index	binIndex+SaoMergeLeftAvai+SaoMergeUpAvai-1	4	3
sao_mode	0	7	1
sao_interval_offset_abs	0	8	1
alf_lcu_enable_flag	0	9	1
qt_split_flag	见8.3.3.2.3	10	4
bet_split_flag	见8.3.3.2.4	14	9
bet_split_type_flag	见8.3.3.2.5	23	3
bet_split_dir_flag	见8.3.3.2.6	26	5
root_cu_constraint	0	31	1
intra_chroma_pred_mode_index	见8.3.3.2.7	32	3
chroma_eipm_index	0	35	1
chroma_pmc_index	0	36	1
chroma_pmc_param_index	0	37	1
ctp_u	0	38	1
ctp_v	0	39	1
chroma_st_flag	0	40	1
ts_cu_flag	0	41	1
skip_flag	见8.3.3.2.8	42	4
advanced_prediction_flag	0	46	1
affine_flag	0	47	1
affine_umve_flag	0	48	1
direct_flag	见8.3.3.2.9	49	2
inter_pf_flag	0	51	1

表 60（续）

语法元素	ctxIndexInc	ctxIndexStart	ctx的数量
inter_pf_index	0	52	1
inter_pc_flag	0	53	1
inter_pc_index	binIndex	54	2
intra_cu_flag	见8.3.3.2.10	56	6
sawp_flag	0	62	1
dt_split_flag	0	63	1
dt_split_dir	0	64	1
dt_split_hqt_flag	0	65	1
dt_split_vqt_flag	0	66	1
dt_split_hadt_flag	0	67	1
dt_split_vadt_flag	0	68	1
umve_mv_index	0	69	1
umve_step_index	0	70	1
umve_dir_index	binIndex	71	2
cu_affine_cand_index	binIndex	73	4
etmvp_flag	0	77	1
cu_etmvp_cand_index	binIndex	78	4
awp_flag	0	82	1
awp_mvr_cand_flag0 awp_mvr_cand_flag1	0	83	1
awp_mvr_cand_step0 awp_mvr_cand_step1	0	84	1
awp_mvr_cand_dir0 awp_mvr_cand_dir1	binIndex	85	2
awp_cand_index0	0	87	1
awp_cand_index1	见8.3.3.2.11	88	2
affine_umve_step_index0 affine_umve_step_index1	0	90	1
affine_umve_dir_index0 affine_umve_dir_index1	binIndex	91	2
cu_subtype_index	binIndex	93	11
isc_ibc_cu_flag	0	104	1
ibc_cu_flag	见8.3.3.2.12	105	3
cbvp_index	binIndex	108	6
abvr_index	binIndex	114	1
extend_mvr_flag	0	115	1
affine_amvr_index	binIndex	116	2
amvr_index	binIndex	118	4
inter_pred_ref_mode_index	见8.3.3.2.13	122	3

表 60（续）

语法元素	ctxIndexInc	ctxIndexStart	ctx的数量
smvd_flag	0	125	1
pu_reference_index_l0 pu_reference_index_l1	见8.3.3.2.14	126	3
mv_diff_x_abs_l0 mv_diff_x_abs_l0_affine mv_diff_x_abs_l1 mv_diff_x_abs_l1_affine	见8.3.3.2.15	129	3
mv_diff_y_abs_l0 mv_diff_y_abs_l0_affine mv_diff_y_abs_l1 mv_diff_y_abs_l1_affine	见8.3.3.2.15	132	3
mv_diff_x_abs_bv	见8.3.3.2.16	135	7
mv_diff_y_abs_bv	见8.3.3.2.17	142	8
bgc_flag	0	150	1
bgc_index	0	151	1
intra_luma_pred_mode_index sawp_pred_mode0_index sawp_pred_mode1_index	见8.3.3.2.18	152	7
luma_eipm_index	0	159	1
intra_pf_flag	0	160	1
iip_flag	0	161	1
ctp_zero_flag	见8.3.3.2.19	162	2
pbt_cu_flag	0	164	1
ctp_y[i]	0	165	1
cu_qp_delta_abs	见8.3.3.2.20	166	4
enhanced_ts_flag	0	170	1
sbt_cu_flag	见8.3.3.2.21	171	2
sbt_quad_flag	0	173	1
sbt_dir_flag	见8.3.3.2.22	174	3
scan_region_x	见8.3.3.2.23	177	28
scan_region_y	见8.3.3.2.23	205	28
significant_flag	见8.3.3.2.24	233	60
coeff_abs_level_greater1_flag coeff_abs_level_greater2_flag coeff_abs_level_greater4_flag coeff_abs_level_greater8_flag	见8.3.3.2.25	293	22
coeff_run	见8.3.3.2.26	315	24
coeff_level_minus1	见8.3.3.2.26	339	24
coeff_last	见8.3.3.2.27	363	34

表 60（续）

语法元素	ctxIndexInc	ctxIndexStart	ctx的数量
est_tu_flag	0	397	1
esao_lcu_enable_flag	0	398	1
ccsao_lcu_enable_flag	0	399	1
isc_match_type[i]	0	400	1
isc_pixel_match_type[i][j]	0	401	1
isc_next_remaining_pixel_in_cu[i]	binIndex	402	3
isc_sv_above_flag[i]	0	405	1
isc_sv_recent_flag[i]	0	406	1
isc_sv_recent_index[i]	binIndex	407	3
isc_sv_y_abs[i]	见8.3.3.2.16	410	7
isc_sv_x_non_zero_flag[i]	0	417	1
isc_subtype_flag	0	418	1
isc_ups_present_flag	0	419	1
isc_evs_ups_num_minus1	0	420	1
isc_ups_max_length_minus1	Min(binIndex,3)	421	4
str_length_minus1_prefix	(binIndex+1)>>1	425	3
nn_filter_lcu_enable_flag	compIdx	428	3
nn_filter_lcu_set_index	compIdx*3+Min(binIdx,2)	431	9

8.3.3.2.2 确定 lcu_qp_delta 的 ctxIndexInc

根据以下方法确定lcu_qp_delta的ctxIndexInc：

——如果 binIndex 和 PreviousDeltaQP 均等于 0，则 ctxIndexInc 等于 0；

——否则，如果 binIndex 等于 0 且 PreviousDeltaQP 不等于 0，则 ctxIndexInc 等于 1；

——否则，如果 binIndex 等于 1，则 ctxIndexInc 等于 2；

——否则，ctxIndexInc 等于 3。

8.3.3.2.3 确定 qt_split_flag 的 ctxIndexInc

根据以下方法确定qt_split_flag的ctxIndexInc（当前亮度编码块E与其左边亮度编码块A、上边亮度编码块B的关系见9.5.4）：

——如果当前图像为帧内预测图像且 E 的宽度为 128，则 ctxIndexInc 等于 3；

——否则，如果 A“存在”且 A 的高度小于 E 的高度、B“存在”且 B 的宽度小于 E 的宽度，则 ctxIndexInc 等于 2；

——否则，如果 A “存在” 且 A 的高度小于 E 的高度，或 B “存在” 且 B 的宽度小于 E 的宽度，则 ctxIndexInc 等于 1；

——否则，ctxIndexInc 等于 0。

8.3.3.2.4 确定 bet_split_flag 的 ctxIndexInc

根据以下方法确定bet_split_flag的ctxIndexInc（当前亮度编码块E与其左边亮度编码块A、上边亮度编码块B的关系见9.5.4）：

——如果A“存在”且A的高度小于E的高度、B“存在”且B的宽度小于E的宽度，则ctxIndexInc等于2；

——否则，如果A“存在”且A的高度小于E的高度，或B“存在”且B的宽度小于E的宽度，则ctxIndexInc等于1；

——否则，ctxIndexInc等于0。

根据以下方法进一步确定bet_split_flag的ctxIndexInc：

——如果E的宽度乘以E的高度的积大于1024，则ctxIndexInc不变；

——否则，如果E的宽度乘以E的高度的积大于256，则ctxIndexInc增加3；

——否则，ctxIndexInc增加6。

8.3.3.2.5 确定bet_split_type_flag的ctxIndexInc

根据以下方法确定bet_split_type_flag的ctxIndexInc（当前亮度编码块E与其左边亮度编码块A、上边亮度编码块B的关系见9.5.4）：

——如果A“存在”且A的高度小于E的高度、B“存在”且B的宽度小于E的宽度，则ctxIndexInc等于2；

——否则，如果A“存在”且A的高度小于E的高度，或B“存在”且B的宽度小于E的宽度，则ctxIndexInc等于1；

——否则，ctxIndexInc等于0。

8.3.3.2.6 确定bet_split_dir_flag的ctxIndexInc

根据以下方法确定bet_split_dir_flag的ctxIndexInc（E是当前亮度编码块）：

——如果E的宽度为128、高度为64，则ctxIndexInc等于4；

——否则，如果E的宽度为64、高度为128，则ctxIndexInc等于3；

——否则，如果E的高度大于E的宽度，则ctxIndexInc等于2；

——否则，如果E的宽度大于E的高度，则ctxIndexInc等于1；

——否则，ctxIndexInc等于0。

8.3.3.2.7 确定intra_chroma_pred_mode_index的ctxIndexInc

根据以下方法确定intra_chroma_pred_mode_index的ctxIndexInc：

——如果binIndex等于0，则ctxIndexInc等于0；

——否则，如果TscpmEnableFlag的值等于1且binIndex等于1，则ctxIndexInc等于2；

——否则，如果PmcEnableFlag的值等于1且binIndex等于1，则ctxIndexInc等于2

——否则，ctxIndexInc等于1。

8.3.3.2.8 确定skip_flag的ctxIndexInc

根据以下方法确定skip_flag的ctxIndexInc（当前预测块E与其左边预测块A、上边预测块B的关系见9.5.4）：

——如果当前预测块E的宽度乘以当前预测块E的高度的积小于64，则ctxIndexInc等于3；

——否则，如果当前预测块E的左边预测块A“存在”且A是跳过模式、当前预测块E的上边预测块B“存在”且B是跳过模式，则ctxIndexInc等于2；

——否则，如果当前预测块E的左边预测块A“存在”且A是跳过模式，或当前预测块E的上边预测块B“存在”且B是跳过模式，则ctxIndexInc等于1。

——否则，ctxIndexInc等于0。

8.3.3.2.9 确定 direct_flag 的 ctxIndexInc

根据以下方法确定direct_flag的ctxIndexInc：

——如果当前预测块 E 的宽度乘以当前预测块 E 的高度的积小于 64，或当前预测块 E 的宽度大于 64，或当前预测块 E 的高度大于 64，则 ctxIndexInc 等于 1；

——否则，ctxIndexInc 等于 0。

8.3.3.2.10 确定 intra_cu_flag 的 ctxIndexInc

根据以下方法确定intra_cu_flag的ctxIndexInc（当前预测块E与其左边预测块A、上边预测块B的关系见9.5.4）：

——如果当前预测块 E 的宽度大于 64，或当前预测块 E 的高度大于 64，则 ctxIndexInc 等于 5；

——否则，如果当前预测块 E 的左边预测块 A“存在”且预测块 A 是帧内预测模式、当前预测块 E 的上边预测块 B“存在”且预测块 B 是帧内预测模式，则 ctxIndexInc 等于 2；

——否则，如果当前预测块 E 的左边预测块 A“存在”且预测块 A 是帧内预测模式，或当前预测块 E 的上边预测块 B“存在”且预测块 B 是帧内预测模式，则 ctxIndexInc 等于 1。

——否则，ctxIndexInc 等于 0。

如果 ctxIndexInc 等于 0，根据以下方法进一步确定 intra_cu_flag 的 ctxIndexInc：

——如果当前预测块 E 的宽度乘以当前预测块 E 的高度的积大于 256，则 ctxIndexInc 等于 0；

——否则,如果当前预测块 E 的宽度乘以当前预测块 E 的高度的积大于 64,则 ctxIndexInc 等于 3；

——否则，ctxIndexInc 等于 4。

8.3.3.2.11 确定 awp_cand_index1 的 ctxIndexInc

根据以下方法确定awp_cand_index1的ctxIndexInc：

——如果 AwpMvrCandFlag0 和 AwpMvrCandFlag1 均为 0，则 ctxIndexInc 等于 1；

——否则，如果 AwpMvrCandFlag0 和 AwpMvrCandFlag1 均为 1，且 AwpMvrCandStep0 和 AwpMvrCandDir0 分别等于 AwpMvrCandStep1 和 AwpMvrCandDir1，则 ctxIndexInc 等于 1；

——否则，ctxIndexInc 等于 0。

8.3.3.2.12 确定 ibc_cu_flag 的 ctxIndexInc

根据以下方法确定ibc_cu_flag的ctxIndexInc（当前预测块E与其左边预测块A、上边预测块B的关系见9.5.4）：

——如果当前预测块 E 的左边预测块 A“存在”且 A 是块复制帧内预测模式、当前预测块 E 的上边预测块 B“存在”且 B 是块复制帧内预测模式，则 ctxIndexInc 等于 2；

——否则，如果当前预测块 E 的左边预测块 A“存在”且 A 是块复制帧内预测模式，或当前预测块 E 的上边预测块 B“存在”且 B 是块复制帧内预测模式，则 ctxIndexInc 等于 1；

——否则，ctxIndexInc 等于 0。

8.3.3.2.13 确定 inter_pred_ref_mode_index 的 ctxIndexInc

根据以下方法确定inter_pred_ref_mode_index的ctxIndexInc：

——如果该语法元素的 binIndex 为 0，且当前预测块 E 的宽度乘以当前预测块 E 的高度的积小于 64，则 ctxIndexInc 等于 2；

——否则，如果该语法元素的 binIndex 为 0，且当前预测块 E 的宽度乘以当前预测块 E 的高度的积大于等于 64，则 ctxIndexInc 等于 0；

——否则（该语法元素的 binIndex 为 1），ctxIndexInc 等于 1。

8.3.3.2.14 确定 pu_reference_index_l0 或 pu_reference_index_l1 的 ctxIndexInc

根据以下方法确定pu_reference_index的ctxIndexInc：
——如果 binIndex 等于 0，则 ctxIndexInc 等于 0；
——否则，如果 binIndex 等于 1，则 ctxIndexInc 等于 1；
——否则，ctxIndexInc 等于 2。

8.3.3.2.15 确定 mv_diff_x_abs_l0、mv_diff_x_abs_l0_affine、mv_diff_x_abs_l1、mv_diff_x_abs_l1_affine、mv_diff_y_abs_l0、mv_diff_y_abs_l0_affine、mv_diff_y_abs_l1 或 mv_diff_y_abs_l1_affine 的 ctxIndexInc

根据以下方法确定 mv_diff_x_abs_l0、mv_diff_x_abs_l0_affine、mv_diff_x_abs_l1、mv_diff_x_abs_l1_affine、mv_diff_y_abs_l0、mv_diff_y_abs_l0_affine、mv_diff_y_abs_l1 或 mv_diff_y_abs_l1_affine的ctxIndexInc：
——如果 binIndex 等于 0，则 ctxIndexInc 等于 0；
——否则，如果 binIndex 等于 1，则 ctxIndexInc 等于 1；
——否则，如果 binIndex 等于 2，则 ctxIndexInc 等于 2；
——否则，BypassFlag 的值为 1。

8.3.3.2.16 确定 mv_diff_x_abs_bv 或 isc_sv_y_abs[i]的 ctxIndexInc

根据以下方法确定mv_diff_x_abs_bv或isc_sv_y_abs[i]的ctxIndexInc：
——如果 binIndex 小于等于 5，则 ctxIndexInc 等于 binIndex；
——否则，如果 binIndex 等于 6，且前一个 bin 为 1，则 ctxIndexInc 等于 6；
——否则，BypassFlag 的值为 1。

8.3.3.2.17 确定 mv_diff_y_abs_bv 的 ctxIndexInc

根据以下方法确定mv_diff_y_abs_bv的ctxIndexInc：
——如果 mv_diff_x_abs_bv 等于 0 且 binIndex 等于 0，则 ctxIndexInc 等于 7；
——否则，如果 binIndex 小于等于 5，则 ctxIndexInc 等于 binIndex；
——否则，如果 binIndex 等于 6，且前一个 bin 为 1，则 ctxIndexInc 等于 6；
——否则，BypassFlag 的值为 1。

8.3.3.2.18 确定 intra_luma_pred_mode_index、sawp_pred_mode0_index 或 sawp_pred_mode1_index 的 ctxIndexInc

根据语法元素 intra_luma_pred_mode_index 的 binIndex 和当前已解析的二元符号串，intra_luma_pred_mode_index的ctxIndexInc应符合表61的规定。表61中，x_i(i=1～4)的值为‘0’或‘1’。

表61 前缀二元符号串与 ctxIndexInc 的关系

binIndex	当前已解析的二元符号串	ctxIndexInc的值
0	空	0
1	0	1
2	0 x_1	2

表 61（续）

binIndex	当前已解析的二元符号串	ctxIndexInc的值
3	0 x_1 x_2	3
4	0 x_1 x_2 x_3	4
5	0 x_1 x_2 x_3 x_4	5
1	1	6

根据以下方法确定sawp_pred_mode0_index或sawp_pred_mode1_index的ctxIndexInc：
——如果 binIndex 等于 0，则 ctxIndexInc 等于 0；
——否则，BypassFlag 的值为 1。

8.3.3.2.19 确定 ctp_zero_flag 的 ctxIndexInc

根据以下方法确定ctp_zero_flag的ctxIndexInc：
——如果当前预测块 E 的宽度大于 64，或当前预测块 E 的高度大于 64，则 ctxIndexInc 等于 1；
——否则，ctxIndexInc 等于 0。

8.3.3.2.20 确定 cu_qp_delta_abs 的 ctxIndexInc

根据以下方法确定cu_qp_delta_abs的ctxIndexInc：
——如果 binIndex 为 0，则 ctxIndexInc = min(NumDeltaQp, 2)；
——否则，ctxIndexInc 等于 3。

8.3.3.2.21 确定 sbt_cu_flag 的 ctxIndexInc

根据以下方法确定sbt_cu_flag的ctxIndexInc：
——如果当前亮度编码块的宽度和高度的乘积大于或等于 256，则 ctxIndexInc 等于 0；
——否则，ctxIndexInc 等于 1。

8.3.3.2.22 确定 sbt_dir_flag 的 ctxIndexInc

根据以下方法确定sbt_dir_flag的ctxIndexInc：
——如果当前亮度编码块的宽度等于高度，则 ctxIndexInc 等于 0；
——否则，如果当前亮度编码块的宽度小于高度，则 ctxIndexInc 等于 1；
——否则，ctxIndexInc 等于 2。

8.3.3.2.23 确定 scan_region_x 或 scan_region_y 的 ctxIndexInc

根据以下步骤确定scan_region_x或scan_region_y的ctxIndexInc。
——令 ctxIndexInc 的值为 0。
——当前块的宽度和高度是 blockWidth 和 blockHeight。如果需要确定的是 scan_region_x 的 ctxIndexInc，则 M 等于 blockWidth；如果需要确定的是 scan_region_y 的 ctxIndexInc，则 M 等于 blockHeight。
——如果当前块是亮度编码块，则：

```
offset = prefix_ctx[logM]
shift = (logM+1)>>2
```

其中 prefix_ctx[8]={0,0,0,3,6,10,15,21}。

——如果当前块是色度编码块，则：

```
offset = 25
shift = Clip3(0, 2, (M >> 3))
```

——执行以下操作：

1) 如果 binIndex 等于 0，则 ctxIndexInc = offset。
2) 否则，如果 binIndex 大于 0，且前一个 bin 为 1、binIndex 小于 group_index[Min(M,32)-1]，则 ctxIndexInc = offset + (binIndex>>shift)。其中 group_index[32]={0,1,2,3,4,4,5,5,6,6,6,6,7,7,7,7,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9}。

8.3.3.2.24 确定 significant_flag 的 ctxIndexInc

根据以下步骤依次执行，确定significant_flag的ctxIndexInc。

——令 ctxIndexInc 的值为 0。

——如果当前块是亮度编码块，则：

```
scanArea = (ScanRegionX + 1) * (ScanRegionY + 1)
ctxIndexInc = ctxIndexInc + scanArea <= 4 ? 0 : 17 << (scanArea <= 16 ? 0 : 1)
```

——如果当前块是色度编码块，则：

```
ctxIndexInc = ctxIndexInc + 51
```

——扫描由左上角坐标（0,0）和右下角坐标（ScanRegionX,ScanRegionY）确定的矩形区域。

——如果 PosX 不等于 ScanRegionX 或 PosY 不等于 ScanRegionY：

1) 如果当前块是亮度编码块，且 PosX 不为 0 或 PosY 不为 0：
   - 如果当前块预测类型是普通帧内预测：

```
i = (PosX<= ScanRegionX/2) ? (PosX <= ScanRegionX/4 ? 0 : 1) : ( PosX <= ScanRegionX * 3/4 ? 2 : 3 )
j = (PosY <= ScanRegionY/2) ? (PosY <= ScanRegionY/4 ? 0 : 1) : ( PosY <= ScanRegionY * 3/4 ? 2 : 3 )
s = i + j
if (s <= 1) {
    ctxIndexInc = ctxIndexInc + 4
}
else if (s > 2) {
    ctxIndexInc = ctxIndexInc + 8
}
else {
    ctxIndexInc = ctxIndexInc + 12
}
```

   - 否则：

```
if ((PosX <= ScanRegionX / 2) && (PosY <= ScanRegionY / 2)) {
    ctxIndexInc = ctxIndexInc + 4
}
```

```
else {
    ctxIndexInc = ctxIndexInc + 8
}
```

- 如果当前块是色度编码块，且 PosX 不为 0 或 PosY 不为 0，则 ctxIndexInc = ctxIndexInc + 4。

2) 令 Pre5Gt0[j]（j=0～4）中 Pre5Gt0[j]的值不为零的个数为 num_gt0，ctxIndexInc 为：

```
ctxIndexInc = ctxIndexInc + Min(num_gt0, 3) + 1
```

8.3.3.2.25 确定 coeff_abs_level_greater1_flag，coeff_abs_level_greater2_flag，coeff_abs_level_greater4_flag 或 coeff_abs_level_greater8_flag 的 ctxIndexInc

根据以下步骤依次执行，确定coeff_abs_level_greater1_flag、coeff_abs_level_greater2_flag、coeff_abs_level_greater4_flag或coeff_abs_level_greater8_flag的ctxIndexInc。

——令 ctxIndexInc 的值为 0。

——如果当前块是色度块，则 ctxIndexInc = ctxIndexInc + 17。

——扫描由左上角坐标（0,0）和右下角坐标（ScanRegionX,ScanRegionY）确定的矩形区域中的非零系数位置。

——如果（PosX,PosY）不是扫描到的第一个非零系数位置：

1) 如果当前块是亮度块，且 PosX 不等于 0 或 PosY 不等于 0：

- 如果当前块预测类型是普通帧内预测，则：

```
i =(PosX<= ScanRegionX/2) ? (PosX <= ScanRegionX/4 ? 0 : 1) : ( PosX <= ScanRegionX * 3/4 ? 2 : 3)
j= (PosY <= ScanRegionY/2) ? (PosY <= ScanRegionY/4 ? 0 : 1) : ( PosY <= ScanRegionY * 3/4 ? 2 : 3)
s = i + j
if (s <= 1) {
    ctxIndexInc = ctxIndexInc + 4
}
else if (s > 2) {
    ctxIndexInc = ctxIndexInc + 8
}
else {
    ctxIndexInc = ctxIndexInc + 12
}
```

- 否则：

```
if ((PosX <= ScanRegionX / 2) && (PosY <= ScanRegionY / 2)) {
    ctxIndexInc = ctxIndexInc + 4
}
else {
    ctxIndexInc = ctxIndexInc + 8
}
```

2) 令 Pre5GtX[j]（j=0～4）中 Pre5GtX[j]的绝对值大于 k 的个数为 num_gtX（k=1,2,4,8）：

- 如果需要确定的是 coeff_abs_level_greater1_flag 的 ctxIndexInc，k 等于 1；
- 如果需要确定的是 coeff_abs_level_greater2_flag 的 ctxIndexInc，k 等于 2；
- 如果需要确定的是 coeff_abs_level_greater4_flag 的 ctxIndexInc，k 等于 4；
- 如果需要确定的是 coeff_abs_level_greater8_flag 的 ctxIndexInc，k 等于 8；
- ctxIndexInc 为：

```
ctxIndexInc = ctxIndexInc + Min(num_gtX, 3) + 1
```

8.3.3.2.26 确定 coeff_run 或 coeff_level_minus1 的 ctxIndexInc

根据以下方法确定coeff_run或coeff_level_minus1的ctxIndexInc：

——令 ctxIndexInc 的值为 0；

——当前系数块上一个解码出的 level 值记为 prev_level，如果没有上一个解码出的 level 值，则 prev_level = 6，ctxIndexInc = ctxIndexInc + min(prev_level-1, 5) * 2；

——如果当前块是色度编码块，则 ctxIndexInc = ctxIndexInc + 12；

——如果 binIndex 大于或等于 1，则 ctxIndexInc = ctxIndexInc + 1。

8.3.3.2.27 确定 coeff_last 的 ctxIndexInc 和 ctxIndexIncW

当前系数块上一个解码出的level值记为prev_level，如果没有上一个解码出的level值，则prev_level的值等于6。

根据以下方法确定coeff_last第一个上下文的ctxIndexInc：

——令 ctxIndexInc 的值为 min(prev_level-1,5)；

——如果当前块是色度编码块，则 ctxIndexInc = ctxIndexInc + 6。

根据以下方法确定coeff_last第二个上下文的ctxIndexIncW：

——令 ctxIndexIncW 的值为 12 + Floor(Log(ScanPosOffset +1))；

——如果当前块是色度编码块，则 ctxIndexIncW = ctxIndexIncW + 12。

8.3.3.3 二元符号解析

8.3.3.3.1 解析过程

二元符号的解析过程如下。

a) 解析二元符号值 binVal：
   1) 如果 BypassFlag 为 1，执行 decode_bypass 过程（见 8.3.3.3.4）；
   2) 否则,如果 StuffingBitFlag 为 1,执行 decode_aec_stuffing_bit 过程(见 8.3.3.3.3)；
   3) 否则，令 cFlag 等于 1，执行 decode_decision 过程（见 8.3.3.3.2）。
b) 如果 binVal 的值为 0，则二元符号为‘0’；如果 binVal 的值为 1，则二元符号为‘1’。

8.3.3.3.2 decode_decision 解析

decode_decision解析的输入是bFlag、cFlag、rS1、rT1、valueS、valueT、valueD以及上下文模型。如果CtxWeight的值为0，输入的上下文模型为ctx；否则输入的上下文模型为ctx和ctxW。decode_decision解析的输出是二元符号值binVal。decode_decision解析如下：

```
decode_decision( ) {
    if ( CtxWeight == 0) {
        predMps = ctx->mps
```

```
        if (MaecEnableFlag) {
            lgPmps = ((ctx->lgPmps0 + ctx->lgPmps1 + 1) >> 3
        }
        else {
            lgPmps = ctx->lgPmps >> 2
        }
    }
    else {
        if ( ctx->mps == ctxW->mps ) {
            predMps = ctx->mps
            lgPmps = ( ctx->lgPmps + ctxW->lgPmps ) >> 1
        }
        else {
            if ( ctx->lgPmps < ctxW->lgPmps ) {
                predMps = ctx->mps
                lgPmps = 1023 - ( ( ctxW->lgPmps - ctx->lgPmps ) >> 1 )
            }
            else {
                predMps = ctxW->mps
                lgPmps = 1023 - ( ( ctx->lgPmps - ctxW->lgPmps ) >> 1 )
            }
        }
        lgPmps = lgPmps >> 2
    }
    if (valueD || (bFlag == 1 && rS1 == boundS) ) {
        rS1 = 0
        valueS = 0
        while ( valueT < 0x100 && valueS < boundS ) {
            valueS++
            valueT = (valueT << 1) | read_bits(1)
        }
        if ( valueT < 0x100 )
            bFlag = 1
        else
            bFlag = 0
        valueT = valueT & 0xFF
    }
    if ( rT1 >= lgPmps ) {
        rS2 = rS1
        rT2 = rT1 - lgPmps
        sFlag = 0
    }
    else {
```

```
        rS2 = rS1 + 1
        rT2 = 256 + rT1 - lgPmps
        sFlag = 1
    }
    if ( (rS2 > valueS || (rS2 == valueS && valueT >= rT2)) && bFlag == 0 ) {
        binVal = ! predMps
        if ( sFlag == 0 )
            tRlps = lgPmps
        else
            tRlps = rT1 + lgPmps
        if ( rS2 == valueS )
            valueT = valueT - rT2
        else
            valueT = 256 + ((valueT << 1 ) | read_bits(1)) - rT2
        while ( tRlps < 0x100 ) {
            tRlps = tRlps << 1
            valueT = (valueT << 1 ) | read_bits(1)
        }
        rT1 = tRlps & 0xFF
        valueD = 1
    }
    else {
        binVal = predMps
        rS1 = rS2
        rT1 = rT2
        valueD = 0
    }
    if ( cFlag ) {
        if ( CtxWeight == 0 ) {
            ctx = update_ctx( binVal, ctx )
        }
        else {
            ctx = update_ctx( binVal, ctx )
            ctxW = update_ctx( binVal, ctxW )
        }
    }
    return (binVal)
}
```

8.3.3.3.3 decode_aec_stuffing_bit 解析

decode_aec_stuffing_bit解析的输入是bFlag、cFlag、rS1、rS2、valueS、valueT和valueD。decode_aec_stuffing_bit解析的输出是二元符号值binVal。ctx0是二元符号模型，令ctx0->lgPmps等

于4，ctx0->mps等于0。令cFlag等于0，ctx等于ctx0，代入decode_decision解析实现decode_aec_stuffing_bit解析。

注：decode_aec_stuffing_bit解析也可参考附录F的描述方式实现。

8.3.3.3.4 decode_bypass 解析

decode_bypass解析的输入是bFlag、cFlag、rS1、rS2、valueS、valueT和valueD。decode_bypass解析的输出是二元符号值binVal。ctx1是二元符号模型，令ctx1->lgPmps等于1024，ctx1->mps等于0。令cFlag等于0，ctx等于ctx1，代入decode_decision解析实现decode_bypass解析。

注：decode_bypass解析也可参考附录F的描述方式实现。

8.3.3.3.5 update_ctx 解析

update_ctx解析的输入是binVal和ctx。update_ctx解析的输出是更新后的ctx。update_ctx解析如下：

```
update_ctx( ) {
    if (MaecEnableFlag) {
        a = baseWin  -  diffWin0
        b = baseWin + diffWin1
        cwr = a
        if ( ctx->cycno < counterThr1 )
            cwr = a  -  1
        else if ( ctx->cycno < counterThr2 )
            cwr = a
        longUpdate = (ctx->cycno < counterThr2) ? 0 : 1
        if ( binVal != ctx->mps ) {
            if ( ctx->cycno < counterThr2 )
                ctx->cycno = ctx->cycno + 1
            else
                ctx->cycno = counterThr2
        }
        else if ( ctx->cycno == 0) {
            ctx->cycno = 1
        }
        if ( binVal == ctx->mps ) {
            ctx->lgPmps0 = ctx->lgPmps0  -  (ctx->lgPmps0 >> cwr) - (ctx->lgPmps0 >> (cwr+2))
            ctx->lgPmps1 = longUpdate ? (ctx->lgPmps1  -  (ctx->lgPmps1 >> b) - (ctx->lgPmps1 >> (b+2))) :
ctx->lgPmps0
        }
        else {
            ctx->lgPmps0 = ctx->lgPmps0 + cwr2LGS[cwr]
            ctx->lgPmps1 = longUpdate ? (ctx->lgPmps1 + cwr2LGS[b]) : ctx->lgPmps0
            if ( ctx->lgPmps0 > 1023 ) {
                ctx->lgPmps0 = 2047 - ctx->lgPmps0
```

```
            }
            if ( ctx->lgPmps1 > 1023 ) {
                ctx->lgPmps1 = 2047 - ctx->lgPmps1
            }
        }
    }
    else {
        if (ctx->cycno <= 1)
            cwr = 3
        else if ( ctx->cycno ==2)
            cwr = 4
        else
            cwr = 5

        if (binVal != ctx->mps) {
            if (ctx->cycno < 2)
                ctx->cycno = ctx->cycno + 1
            else
                ctx->cycno = 3
        }
        else if (ctx->cycno == 0) {
            ctx->cycno = 1
        }
        if (binVal == ctx->mps) {
            ctx->lgPmps = ctx->lgPmps  -  (ctx->lgPmps >> cwr) - (ctx->lgPmps >> (cwr+2))
        }
        else {
            ctx->lgPmps = ctx->lgPmps + cwr2LGS[cwr]
            if (ctx->lgPmps > 1023) {
                ctx->lgPmps = 2047 - ctx->lgPmps
            }
        }
    }
    return (ctx)
}
```

其中diffWin0、diffWin1和baseWin见8.3.2.1，cwr2LGS[10]的值是{427, 427, 427, 197, 95, 46, 23, 12, 6, 3}。

8.3.4 反二值化方法

8.3.4.1 概述

语法元素的反二值化方法应符合表62的规定。

表62　语法元素的反二值化方法

语法元素	反二值化方法
lcu_qp_delta	见 8.3.4.5
sao_merge_type_index	见 8.3.4.6
sao_mode	见 8.3.4.2，maxVal=2，sao_mode 的值等于 synElVal
sao_interval_offset_abs	见 8.3.4.2，maxVal=7，sao_interval_offset_abs 的值等于 synElVal
sao_interval_offset_sign	见 8.3.4.4，sao_interval_offset_sign 的值等于 synElVal
sao_interval_start_pos	见 8.3.4.7
sao_interval_delta_pos_minus2	见 8.3.4.8
sao_edge_offset[compIndex][j]（j = 0～3）	见 8.3.4.9
sao_edge_type	见 8.3.4.10
alf_lcu_enable_flag	见 8.3.4.4，alf_lcu_enable_flag 的值等于 synElVal
aec_lcu_stuffing_bit	见 8.3.4.4，aec_lcu_stuffing_bit 的值等于 synElVal
qt_split_flag	见 8.3.4.4，qt_split_flag 的值等于 synElVal
bet_split_flag	见 8.3.4.4，bet_split_flag 的值等于 synElVal
bet_split_type_flag	见 8.3.4.4，bet_split_type_flag 的值等于 synElVal
bet_split_dir_flag	见 8.3.4.4，bet_split_dir_flag 的值等于 synElVal
root_cu_constraint	见 8.3.4.4，root_cu_constraint 的值等于 synElVal
intra_chroma_pred_mode_index	见 8.3.4.2，maxVal=((TscpmEnableFlag || PmcEnableFlag) ? 5 : 4))，intra_chroma_pred_mode_index 的值等于 synElVal
chroma_eipm_index	见 8.3.4.2，maxVal=3，chroma_eipm_index 的值等于 synElVal
chroma_pmc_index	见 8.3.4.4，chroma_pmc_index 的值等于 synElVal
chroma_pmc_param_index	见 8.3.4.4，chroma_pmc_param_index 的值等于 synElVal
ctp_u	见 8.3.4.4，ctp_u 的值等于 synElVal
ctp_v	见 8.3.4.4，ctp_v 的值等于 synElVal
skip_flag	见 8.3.4.4，skip_flag 的值等于 synElVal
advanced_pred_flag	见 8.3.4.4，advanced_pred_flag 的值等于 synElVal
awp_flag	见 8.3.4.4，awp_flag 的值等于 synElVal
etmvp_flag	见 8.3.4.4，etmvp_flag 的值等于 synElVal
cu_etmvp_cand_index	见 8.3.4.2，maxVal=4，cu_etmvp_cand_index 的值等于 synElVal
affine_flag	见 8.3.4.4，affine_flag 的值等于 synElVal
affine_umve_flag	见 8.3.4.4，affine_umve_flag 的值等于 synElVal
direct_flag	见 8.3.4.4，direct_flag 的值等于 synElVal
inter_pf_flag	见 8.3.4.4，inter_pf_flag 的值等于 synElVal
inter_pf_index	见 8.3.4.4，inter_pf_index 的值等于 synElVal
inter_pc_flag	见 8.3.4.4，inter_pc_flag 的值等于 synElVal
inter_pc_index	见 8.3.4.2，maxVal=2，inter_pc_index 的值等于 synElVal
intra_cu_flag	见 8.3.4.4，intra_cu_flag 的值等于 synElVal
sawp_flag	见 8.3.4.4，sawp_flag 的值等于 synElVal
sawp_index	见 8.3.4.15
sawp_pred_mode0_index	见 8.3.4.12

表 62（续）

语法元素	反二值化方法
sawp_pred_mode1_index	如果 sawp_pred_mode0_index 的值大于 1，见 8.3.4.12；否则，见 8.3.4.13
dt_split_flag	见 8.3.4.4，dt_split_flag 的值等于 synElVal
dt_split_dir	见 8.3.4.4，dt_split_dir 的值等于 synElVal
dt_split_hqt_flag	见 8.3.4.4，dt_split_hqt_flag 的值等于 synElVal
dt_split_vqt_flag	见 8.3.4.4，dt_split_vqt_flag 的值等于 synElVal
dt_split_hadt_flag	见 8.3.4.4，dt_split_hadt_flag 的值等于 synElVal
dt_split_vadt_flag	见 8.3.4.4，dt_split_vadt_flag 的值等于 synElVal
umve_mv_index	见 8.3.4.2，maxVal=1，umve_mv_index 的值等于 synElVal
umve_step_index	如果 PictureUmveStepSetIndex 的值为 0，见 8.3.4.2，maxVal=4，umve_step_index 的值等于 synElVal；否则，见 8.3.4.11
umve_dir_index	见 8.3.4.14
cu_affine_cand_index	见 8.3.4.2，maxVal=4，cu_affine_cand_index 的值等于 synElVal
awp_index	见 8.3.4.15
awp_mvr_cand_flag0	见 8.3.4.4，awp_mvr_cand_flag0 的值等于 synElVal
awp_mvr_cand_flag1	见 8.3.4.4，awp_mvr_cand_flag1 的值等于 synElVal
awp_mvr_cand_step0	见 8.3.4.2，maxVal=4，awp_mvr_cand_step0 的值等于 synElVal
awp_mvr_cand_step1	见 8.3.4.2，maxVal=4，awp_mvr_cand_step1 的值等于 synElVal
awp_mvr_cand_dir0	见 8.3.4.14
awp_mvr_cand_dir1	见 8.3.4.14
awp_cand_index0	见 8.3.4.2，maxVal=4，awp_cand_index0 的值等于 synElVal
awp_cand_index1	见 8.3.4.2，maxVal=(AwpMvrCandFlag0 == 0 && AwpMvrCandFlag1 == 0) || (AwpMvrCandFlag0 == 1 && AwpMvrCandFlag1 == 1 && AwpMvrCandStep0 == AwpMvrCandStep1 && AwpMvrCandDir0 == AwpMvrCandDir1)? 3 : 4，awp_cand_index1 的值等于 synElVal
affine_umve_step_index0	见 8.3.4.2，maxVal=4，affine_umve_step_index0 的值等于 synElVal
affine_umve_step_index1	见 8.3.4.2，maxVal=4，affine_umve_step_index1 的值等于 synElVal
affine_umve_dir_index0	见 8.3.4.14
affine_umve_dir_index1	见 8.3.4.14
cu_subtype_index	见 8.3.4.2，maxVal=((PictureType == 1) ? (1: 3) + Max(NumOfMvapCand,NumOfHmvpCand))，cu_subtype_index 的值等于 synElVal
isc_ibc_cu_flag	见 8.3.4.4，isc_ibc_cu_flag 的值等于 synElVal
ibc_cu_flag	见 8.3.4.4，ibc_cu_flag 的值等于 synElVal
isc_subtype_flag	见 8.3.4.4，isc_subtype_flag 的值等于 synElVal
cbvp_index	见 8.3.4.2，maxVal=6，cbvp_index 的值等于 synElVal
abvr_index	见 8.3.4.2，maxVal=1，abvr_index 的值等于 synElVal
extend_mvr_flag	见 8.3.4.4，extend_mvr_flag 的值等于 synElVal
affine_amvr_index	见 8.3.4.2，maxVal=2，affine_amvr_index 的值等于 synElVal

表 62（续）

语法元素	反二值化方法
amvr_index	见 8.3.4.2，maxVal=4，amvr_index 的值等于 synElVal
inter_pred_ref_mode_index	见 8.3.4.16
smvd_flag	见 8.3.4.4，smvd_flag 的值等于 synElVal
pu_reference_index_l0	见 8.3.4.2，maxVal=NumRefActive[0]-1，pu_reference_index_l0 的值等于 synElVal
pu_reference_index_l1	见 8.3.4.2，maxVal=NumRefActive[1]-1，pu_reference_index_l1 的值等于 synElVal
mv_diff_x_abs_l0 mv_diff_x_abs_l0_affine mv_diff_x_abs_l1 mv_diff_x_abs_l1_affine	见 8.3.4.17
mv_diff_x_sign_l0 mv_diff_x_sign_l0_affine mv_diff_x_sign_l1 mv_diff_x_sign_l1_affine mv_diff_x_sign_bv	见 8.3.4.4 ， mv_diff_x_sign_l0 、 mv_diff_x_sign_l0_affine 、 mv_diff_x_sign_l1、mv_diff_x_sign_l1_affine 或 mv_diff_x_sign_bv 的值等于 synElVal
mv_diff_y_abs_l0 mv_diff_y_abs_l0_affine mv_diff_y_abs_l1 mv_diff_y_abs_l1_affine	见 8.3.4.17
mv_diff_y_sign_l0 mv_diff_y_sign_l0_affine mv_diff_y_sign_l1 mv_diff_y_sign_l1_affine mv_diff_y_sign_bv	见 8.3.4.4 ， mv_diff_y_sign_l0 、 mv_diff_y_sign_l0_affine 、 mv_diff_y_sign_l1、mv_diff_y_sign_l1_affine 或 mv_diff_y_sign_bv 的值等于 synElVal
mv_diff_x_abs_bv、mv_diff_y_abs_bv	见 8.3.4.18
bgc_flag	见 8.3.4.4，bgc_flag 的值等于 synElVal
bgc_index	见 8.3.4.4，bgc_index 的值等于 synElVal
intra_luma_pred_mode_index	见 8.3.4.19
luma_eipm_index	见 8.3.4.4，luma_eipm_index 的值等于 synElVal
intra_pf_flag	见 8.3.4.4，intra_pf_flag 的值等于 synElVal
iip_flag	见 8.3.4.4，iip_flag 的值等于 synElVal
ctp_zero_flag	见 8.3.4.4，ctp_zero_flag 的值等于 synElVal
pbt_cu_flag	见 8.3.4.4，pbt_cu_flag 的值等于 synElVal
ctp_y[i]	见 8.3.4.4，cty_y[i]的值等于 synElVal
cu_qp_delta_abs	见 8.3.4.3，cu_qp_delta_abs 的值等于 synElVal
cu_qp_delta_sign	见 8.3.4.4，cu_qp_delta_sign 的值等于 synElVal
enhanced_ts_flag	见 8.3.4.4，enhanced_ts_flag 的值等于 synElVal
sbt_cu_flag	见 8.3.4.4，sbt_cu_flag 的值等于 synElVal

表 62（续）

语法元素	反二值化方法
sbt_quad_flag	见 8.3.4.4，sbt_quad_flag 的值等于 synElVal
sbt_dir_flag	见 8.3.4.4，sbt_dir_flag 的值等于 synElVal
chroma_st_flag	见 8.3.4.4，chroma_st_flag 的值等于 synElVal
ts_cu_flag	见 8.3.4.4，ts_cu_flag 的值等于 synElVal
scan_region_x	见 8.3.4.20，scan_region_x 的值等于 synElVal
scan_region_y	见 8.3.4.20，scan_region_y 的值等于 synElVal
significant_flag	见 8.3.4.4，significant_flag 的值等于 synElVal
coeff_abs_level_greater1_flag	见 8.3.4.4，coeff_abs_level_greater1_flag 的值等于 synElVal
coeff_abs_level_greater2_flag	见 8.3.4.4，coeff_abs_level_greater2_flag 的值等于 synElVal
coeff_abs_level_greater4_flag	见 8.3.4.4，coeff_abs_level_greater4_flag 的值等于 synElVal
coeff_abs_level_greater8_flag	见 8.3.4.4，coeff_abs_level_greater8_flag 的值等于 synElVal
coeff_abs_level_remaining	见 8.3.4.21，coeff_abs_level_remaining 的值等于 synElVal
coeff_run	见 8.3.4.22，TH=16，coeff_run 的值等于 synElVal
coeff_level_minus1	见 8.3.4.22，TH=8，coeff_level_minus1 的值等于 synElVal
coeff_sign	见 8.3.4.4，coeff_sign 的值等于 synElVal
coeff_last	见 8.3.4.4，coeff_last 的值等于 synElVal
est_tu_flag	见 8.3.4.4，est_tu_flag 的值等于 synElVal
aec_ipcm_stuffing_bit	见 8.3.4.4，aec_ipcm_stuffing_bit 的值等于 synElVal
esao_lcu_enable_flag	见 8.3.4.4，esao_lcu_enable_flag 的值等于 synElVal
esao_lcu_set_index	见 8.3.4.2，maxVal=1，esao_lcu_set_index 的值等于 synElVal
ccsao_lcu_enable_flag	见 8.3.4.4，ccsao_lcu_enable_flag 的值等于 synElVal
ccsao_lcu_set_index	见 8.3.4.2，maxVal=PictureCcsaoSetNum-1，ccsao_lcu_set_index 的值等于 synElVal
isc_match_type[i]	见 8.3.4.4，isc_match_type[i]的值等于 synElVal
isc_sv_above_flag[i]	见 8.3.4.4，isc_sv_above_flag[i]的值等于 synElVal
isc_sv_recent_flag[i]	见 8.3.4.4，isc_sv_recent_flag[i]的值等于 synElVal
isc_sv_x_non_zero_flag[i]	见 8.3.4.4，isc_sv_x_non_zero_flag[i]的值等于 synElVal
isc_sv_x_sign[i]	见 8.3.4.4，isc_sv_x_sign[i]的值等于 synElVal
isc_sv_y_sign[i]	见 8.3.4.4，isc_sv_y_sign[i]的值等于 synElVal
isc_sv_x_abs_minus1[i]	见 8.3.4.23，isc_sv_x_abs_minus1[i]的值等于 synElVal
isc_sv_y_abs[i]	见 8.3.4.18
isc_unmatched_pixel_y[i][j]	见表 85，len=BitDepth，isc_unmatched_pixel_y[i][j]的值等于 synElVal
isc_unmatched_pixel_cb[i][j]	见表 85，len=BitDepth，isc_unmatched_pixel_cb[i][j]的值等于 synElVal
isc_unmatched_pixel_cr[i][j]	见表 85，len=BitDepth，isc_unmatched_pixel_cr[i][j]的值等于 synElVal
isc_nos_up_y[i]	见表 85，len=BitDepth，isc_nos_up_y[i]的值等于 synElVal
isc_nos_up_cb[i]	见表 85，len=BitDepth，isc_nos_up_cb[i]的值等于 synElVal

表 62（续）

语法元素	反二值化方法
isc_nos_up_cr[i]	见表 85，len=BitDepth, isc_nos_up_cr[i]的值等于 synElVal
isc_sv_recent_index[i]	见 8.3.4.2，maxVal=11，isc_sv_recent_index[i]的值等于 synElVal
isc_next_remaining_pixel_in_cu[i]	见 8.3.4.24，isc_next_remaining_pixel_in_cu[i]的值等于 synElVal
isc_pixel_match_type[i][j]	见 8.3.4.4，isc_pixel_match_type[i][j]的值等于 synElVal
isc_ups_present_flag	见 8.3.4.4，isc_ups_present_flag 的值等于 synElVal
isc_num_of_new_pv	见表 58，k=0，isc_num_of_new_pv 的值等于 CodeNum
isc_num_of_reused_pv	见表 58，k=0，isc_num_of_reused_pv 的值等于 CodeNum
isc_prev_pv_not_reused_run	见表 58，k=0，isc_prev_pv_not_reused_run 的值等于 CodeNum
isc_ups_max_length_minus1	见 8.3.4.2，maxVal = NumTotalPixel，isc_ups_max_length_minus1 的值等于 synElVal
isc_evs_ups_num_minus1	见 8.3.4.2，maxVal = NumTotalPixel - NumCodedPixel + FirstStringNumFlag－1，isc_evs_ups_num_minus1 的值等于 synElVal
str_length_minus1_prefix	见 8.3.4.2，maxVal = maxValPrefix，str_length_minus1_prefix 的值等于 synElVal
str_length_minus1_infix	见表 85，len= Ceil(Log(maxValInfix+1))-1, str_length_minus1_infix 的值等于 synElVal
str_length_minus1_suffix	见表 85，len=1, str_length_minus1_suffix 的值等于 synElVal
pv_address_prefix	见表 85，len= Ceil(Log(MaxValPvAddress+1))-1, pv_address_prefix 的值等于 synElVal
pv_address_infix	见表 85，len=1, pv_address_infix 的值等于 synElVal
pv_address_suffix	见表 85，len=1, pv_address_suffix 的值等于 synElVal
isc_fopixel_y	见表 85，len=BitDepth, isc_fopixel_y 的值等于 synElVal
isc_fopixel_cb	见表 85，len=BitDepth, isc_fopixel_cb 的值等于 synElVal
isc_fopixel_cr	见表 85，len=BitDepth, isc_fopixel_cr 的值等于 synElVal
nn_filter_lcu_enable_flag	见 8.3.4.4，nn_filter_lcu_enable_flag 的值等于 synElVal
nn_filter_lcu_set_index	见 8.3.4.2，maxVal = NumOfNnFilter-1，nn_filter_lcu_set_index 的值等于 synElVal

8.3.4.2 采用截断一元码的反二值化方法

由二元符号串根据表63得到synElVal的值。

表63 synElVal 与二元符号串的关系（截断一元码）

synElVal	二元符号串							
0	1							
1	0	1						
2	0	0	1					
3	0	0	0	1				
4	0	0	0	0	1			

表 63（续）

synElVal	二元符号串							
5	0	0	0	0	0	1		
…	0	0	0	0	0	0	…	
maxVal-1	0	0	0	0	0	0	…	1
maxVal	0	0	0	0	0	0	…	0
binIndex	0	1	2	3	4	5	…	maxVal-1

8.3.4.3 采用一元码的反二值化方法

由二元符号串根据表64得到synElVal的值。

表64 synElVal 的值与二元符号串的关系（一元码）

synElVal	二元符号串					
0	1					
1	0	1				
2	0	0	1			
3	0	0	0	1		
4	0	0	0	0	1	
5	0	0	0	0	0	1
…						
binIndex	0	1	2	3	4	5

8.3.4.4 采用标记位的反二值化方法

由二元符号串根据表65得到synElVal的值。

表65 synElVal 的值与二元符号串的关系（标记位）

synElVal	二元符号串
0	0
1	1
binIndex	0

8.3.4.5 lcu_qp_delta 的反二值化方法

首先由二元符号串根据 8.3.4.3 得到 synElVal 的值，然后根据以下方法得到 lcu_qp_delta 的值：

——如果 synElVal 的值能被 2 整除，则 cu_qp_delta = -(synElVal / 2)；

——否则，cu_qp_delta = (synElVal+1) / 2。

8.3.4.6 sao_merge_type_index 的反二值化方法

由SaoMergeLeftAvai的值、SaoMergeUpAvai的值和二元符号串查表66得到sao_merge_type_index的值。

表66 sao_merge_type_index 的值与二元符号串的关系

SaoMergeLeftAvai的值	SaoMergeUpAvai的值	sao_merge_type_index的值	二元符号串	
1	0	0	0	
		1	1	
0	1	0	0	
		1	1	
1	1	0	0	0
		1	1	
		2	0	1
binIndex			0	1

8.3.4.7 sao_interval_start_pos 的反二值化方法

由二元符号串查表67得到sao_interval_start_pos的值。

表67 sao_interval_start_pos 的值与二元符号串的关系

sao_interval_start_pos的值	二元符号串				
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	1	1	0	0	0
4	0	0	1	0	0
…	…	…	…	…	…
28	0	0	1	1	1
29	1	0	1	1	1
30	0	1	1	1	1
31	1	1	1	1	1
binIndex	0	1	2	3	4

8.3.4.8 sao_interval_delta_pos_minus2 的反二值化方法

由二元符号串查表68得到sao_interval_delta_pos_minus2的值。

表68 sao_interval_delta_pos_minus2 的值与二元符号串的关系

sao_interval_delta_pos_minus2的值	二元符号串							
0	1	0						
1	1	1						
2	0	1	0	0				
3	0	1	0	1				
4	0	1	1	0				
5	0	1	1	1				
6	0	0	1	0	0	0		

表 68（续）

sao_interval_delta_pos_minus2的值	二元符号串							
7	0	0	1	0	0	1		
8	0	0	1	0	1	0		
9	0	0	1	0	1	1		
10	0	0	1	1	0	0		
11	0	0	1	1	0	1		
12	0	0	1	1	1	0		
13	0	0	1	1	1	1		
14	0	0	0					
binIndex	0	1	2	3	4	5	6	7

8.3.4.9 sao_edge_offset[compIndex][j]的反二值化方法

由二元符号串查表69得到sao_edge_offset[compIndex][0]或sao_edge_offset[compIndex][3]的值;由二元符号串查表70得到sao_edge_offset[compIndex][1]或sao_edge_offset[compIndex][2]的值。

表69 sao_edge_offset[compIndex][0]和 sao_edge_offset[compIndex][3]的值与二元符号串的关系

sao_edge_offset[compIndex][0]	sao_edge_offset[compIndex][3]	二元符号串						
1	-1	1						
0	0	0	1					
2	-2	0	0	1				
-1	1	0	0	0	1			
3	-3	0	0	0	0	1		
4	-4	0	0	0	0	0	1	
5	-5	0	0	0	0	0	0	1
6	-6	0	0	0	0	0	0	0
binIndex		0	1	2	3	4	5	6

表70 sao_edge_offset[compIndex][1]和 sao_edge_offset[compIndex][2]的值与二元符号串的关系

sao_edge_offset[compIndex][1]	sao_edge_offset[compIndex][2]	二元符号串
0	0	1
1	-1	0
binIndex		0

8.3.4.10 sao_edge_type 的反二值化方法

由二元符号串查表 71 得到 sao_edge_type 的值。

表71　sao_edge_type 的值与二元符号串的关系

sao_edge_type的值	二元符号串	
0	0	0
1	1	0
2	0	1
3	1	1
binIndex	0	1

8.3.4.11　umve_step_index 的反二值化方法

如果 PictureUmveStepSetIndex 的值为 1，由二元符号串查表 72 得到 umve_step_index 的值。

表72　umve_step_index 的值与二元符号串的关系

umve_step_index的值	二元符号串			
0	0	0	0	
1	0	0	1	
2	0	1	1	
3	0	1	0	
4	1	0		
5	1	1	0	
6	1	1	1	0
7	1	1	1	1
binIndex	0	1	2	3

8.3.4.12　sawp_pred_mode0_index 和 sawp_pred_mode1_index 的反二值化方法

由二元符号串查表 73 得到 sawp_pred_mode0_index 或 sawp_pred_mode1_index 的值。

表73　sawp_pred_mode0_index 或 sawp_pred_mode0_index 的值与二元符号串的关系

sawp_pred_mode0_index或sawp_pred_mode1_index的值	二元符号串					
0	1	0				
1	1	1				
2	0	0	0	0	0	0
3	0	0	0	0	0	1
4	0	1	0	0	0	
5	0	0	0	0	1	0
6	0	0	0	0	1	1
7	0	1	0	0	1	
8	0	0	0	1	0	0
9	0	0	0	1	0	1
10	0	1	0	1	0	
11	0	0	0	1	1	0

表 73（续）

sawp_pred_mode0_index或sawp_pred_mode1_index的值	二元符号串					
12	0	0	0	1	1	1
13	0	1	0	1	1	
14	0	0	1	0	0	0
15	0	0	1	0	0	1
16	0	1	1	0	0	
17	0	0	1	0	1	0
18	0	0	1	0	1	1
19	0	1	1	0	1	
20	0	0	1	1	0	0
21	0	0	1	1	0	1
22	0	1	1	1	0	
23	0	0	1	1	1	0
24	0	0	1	1	1	1
25	0	1	1	1	1	
binIndex	0	1	2	3	4	5

8.3.4.13 sawp_pred_mode1_index 的反二值化方法

由二元符号串查表 74 得到 sawp_pred_mode1_index 的值。

表74 sawp_pred_mode1_index 的值与二元符号串的关系

sawp_pred_mode1_index的值	二元符号串					
0	1					
1	0	0	0	0	0	0
2	0	0	0	0	0	1
3	0	1	0	0	0	
4	0	0	0	0	1	0
5	0	0	0	0	1	1
6	0	1	0	0	1	
7	0	0	0	1	0	0
8	0	0	0	1	0	1
9	0	1	0	1	0	
10	0	0	0	1	1	0
11	0	0	0	1	1	1
12	0	1	0	1	1	
13	0	0	1	0	0	0
14	0	0	1	0	0	1
15	0	1	1	0	0	
16	0	0	1	0	1	0

表 74（续）

sawp_pred_mode1_index的值	二元符号串					
17	0	0	1	0	1	1
18	0	1	1	0	1	
19	0	0	1	1	0	0
20	0	0	1	1	0	1
21	0	1	1	1	0	
22	0	0	1	1	1	0
23	0	0	1	1	1	1
24	0	1	1	1	1	
binIndex	0	1	2	3	4	5

8.3.4.14 umve_dir_index、awp_mvr_cand_dir0 、awp_mvr_cand_dir1、affine_umve_dir_index0 和 affine_umve_dir_index1 的反二值化方法

由二元符号串查表75得到synElVal的值。umve_dir_index、awp_mvr_cand_dir0、awp_mvr_cand_dir1、awp_mvr_cand_dir1、affine_umve_dir_index0和affine_umve_dir_index1的值等于synElVal的值。

表75 synElVal 与二元符号串的关系

synElVal的值	二元符号串	
0	0	0
1	0	1
2	1	0
3	1	1
binIndex	0	1

8.3.4.15 awp_index 和 sawp_index 的反二值化方法

第 1 步，如果当前图像是 B 图像或 SawpFlag 的值等于 1，则 maxValue 的值为 56；否则，根据当前编码单元的宽度 W 和高度 H 查表 76 确定 maxValue 的值。

表76 W 和 H 的值与 maxValue 的关系

W	H	maxValue
8	8	6
	16	12
	32	22
	64	22
16	8	12
	16	18
	32	24
	64	24

表 76（续）

W	H	maxValue
32	8	21
	16	25
	32	27
	64	27
64	8	22
	16	24
	32	27
	64	28

第 2 步，计算 stage、num0、num1 和 thd 的值。

```
stage = Ceil(Log(maxValue)) - 1
num0 = 1 << stage
num1 = 1 << (stage + 1)
thd = (num0 << 1) - maxValue
```

第 3 步，由二元符号串查表 77 得到 synElVal 的值。其中，0～(thd-1)的二元符号串为值为 0～thd-1 对应的长度为 stage 的定长码，thd～(maxValue-1)的二元符号串为值为(thd << 1)～num1 对应的长度为(stage+1)的定长码。

表77　synElVal 的值与二元符号串的关系

synElVal的值	二元符号串			
0	0	...	0	
...	...	...	...	
thd - 1	...	...	...	
thd	...	...	...	0
thd + 1	...	...	...	1
...	...	...	...	...
maxVlaue - 2	1	...	1	0
maxVlaue - 1	1	...	1	1
binIndex	0	...	stage - 1	stage

第 4 步，awp_index 和 sawp_index 的值等于 synElVal。

8.3.4.16　inter_pred_ref_mode_index 的反二值化方法

由二元符号串查表 78 得到 inter_pred_ref_mode_index 的值。

表78 inter_pred_ref_mode_index 的值与二元符号串的关系

inter_pred_ref_mode_index的值	二元符号串	
0	0	0
1	0	1
2	1	
binIndex	0	1

8.3.4.17 mv_diff_x_abs_l0、mv_diff_x_abs_l0_affine、mv_diff_x_abs_l1、mv_diff_x_abs_l1_affine、mv_diff_y_abs_l0、mv_diff_y_abs_l0_affine、mv_diff_y_abs_l1 和 mv_diff_y_abs_l1_affine 的反二值化方法

由二元符号串查表 79 得到 synElVal 的值。表 79 中，如果 synElVal 的值大于或等于 3 并且 synElVal 的值为奇数，二元符号串的前四位为‘1110’，后续位为(synElVal-3)/2 对应的 0 阶指数哥伦布码（见表 58）；如果 synElVal 的值大于 3 并且 synElVal 的值为偶数，二元符号串的前四位为‘1111’，后续位为(synElVal-3)/2 对应的 0 阶指数哥伦布码。

mv_diff_x_abs_l0、mv_diff_x_abs_l0_affine、mv_diff_x_abs_l1、mv_diff_x_abs_l1_affine、mv_diff_y_abs_l0、mv_diff_y_abs_l0_affine、mv_diff_y_abs_l1 或 mv_diff_y_abs_l1_affine 的值等于 synElVal。

表79 synElVal 与二元符号串的关系

synElVal的值	二元符号串										
0	0										
1	1	0									
2	1	1	0								
3	1	1	1	0	1						
4	1	1	1	1	1						
5	1	1	1	0	0	1	0				
6	1	1	1	1	0	1	0				
7	1	1	1	0	0	1	1				
8	1	1	1	1	0	1	1				
9	1	1	1	0	0	0	1	0	0		
10	1	1	1	1	0	0	1	0	0		
11	1	1	1	0	0	0	1	0	1		
12	1	1	1	1	0	0	1	0	1		
13	1	1	1	0	0	0	1	1	0		
14	1	1	1	1	0	0	1	1	0		
…											
binIndex	0	1	2	3	4	5	6	7	8	9	10

8.3.4.18 mv_diff_x_abs_bv、mv_diff_y_abs_bv 和 isc_sv_y_abs[i]的反二值化方法

由二元符号串查表 80 得到 synElVal 的值。表 80 中，如果 synElVal 的值大于或等于 5 且小于 9，

二元符号的前 6 位是‘111110’，后续位是(synElVal-5)对应的长度为 2 的定长码（见表 85)。如果 synElVal 的值大于或等于 9 且小于 17，二元符号的前 7 位是‘1111110’，后续位是(synElVal-9)对应的长度为 3 的定长码（见表 85)。如果 synElVal 的值大于或等于 17 且 synElVal 的值为奇数，二元符号串的前 8 位是‘11111110’，后续位是(synElVal-17)/2 对应的 2 阶指数哥伦布码（见表 58)；如果 synElVal 的值大于 17 且 synElVal 的值为偶数，二元符号串的前 8 位是‘11111111’，后续位是(synElVal-17)/2 对应的 2 阶指数哥伦布码。

mv_diff_x_abs_bv、mv_diff_y_abs_bv 或 isc_sv_y_abs[i]的值等于 synElVal。

表80　synElVal 与二元符号串的关系

synElVal的值	二元符号串										
0	0										
1	1	0									
2	1	1	0								
3	1	1	1	0							
4	1	1	1	1	0						
5	1	1	1	1	1	0	0	0			
6	1	1	1	1	1	0	0	1			
7	1	1	1	1	1	0	1	0			
8	1	1	1	1	1	0	1	1			
9	1	1	1	1	1	1	0	0	0	0	
10	1	1	1	1	1	1	0	0	0	1	
11	1	1	1	1	1	1	0	0	1	0	
12	1	1	1	1	1	1	0	0	1	1	
13	1	1	1	1	1	1	0	1	0	0	
14	1	1	1	1	1	1	0	1	0	1	
15	1	1	1	1	1	1	0	1	1	0	
16	1	1	1	1	1	1	0	1	1	1	
17	1	1	1	1	1	1	1	0	1	0	0
18	1	1	1	1	1	1	1	1	1	0	0
19	1	1	1	1	1	1	1	0	1	0	1
20	1	1	1	1	1	1	1	1	1	0	1
…											
binIndex	0	1	2	3	4	5	6	7	8	9	10

8.3.4.19　intra_luma_pred_mode_index 的反二值化方法

由二元符号串查表 81 得到 intra_luma_pred_mode_index 的值。

表81　intra_luma_pred_mode_index 的值与二元符号串的关系

intra_luma_pred_mode_index的值	二元符号串					
0	1	0				
1	1	1				

表 81（续）

intra_luma_pred_mode_index的值	二元符号串					
2	0	0	0	0	0	0
3	0	0	0	0	0	1
4	0	0	0	0	1	0
…						
32	0	1	1	1	1	0
33	0	1	1	1	1	1
binIndex	0	1	2	3	4	5

8.3.4.20 scan_region_x 或 scan_region_y 的反二值化方法

令当前块的宽度和高度是 blockWidth 和 blockHeight。如果要确定的语法元素是 scan_region_x，则 M 等于 blockWidth；如果要确定的语法元素是 scan_region_y，则 M 等于 blockHeight。

按以下方法得到 scan_region_x 或 scan_region_y 的值 synElVal（包括第一部分的值 synElVal1 和第二部分的值 synElVal2）。其中 val1[binIndex]和 val2[binIndex]分别是第一部分和第二部分的 binIndex 对应的二元符号值。

```
count = 0
min_in_group[10]={0,1,2,3,4,6,8,12,16,24}
group_index[32]={0,1,2,3,4,4,5,5,6,6,6,6,7,7,7,7,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9}
for (binIndex=0; binIndex<group_index[Min(M, 32)-1]; binIndex++) {
    if (val1[binIndex] == 0) {
        break
    }
    count++
}
if (count <= 3) {
    synElVal1 = count
    synElVal = synElVal1
}
else {
    synElVal1 = min_in_group[count]
    synElVal2 = 0
    binIndex = 0
    for (i=((count-2)>>1)-1; i>=0; i--) {
        synElVal_2 += val2[binIndex] << i
        binIndex++
    }
    synElVal = synElVal1 + synElVal2
}
```

8.3.4.21 coeff_abs_level_remaining 的反二值化方法

coeff_abs_level_remaining 的反二值化方法如下：

——如果 EscapeDataPresent 等于 1，则使用 0 阶指数哥伦布码由表 82 得到 synElVal 的值；

——否则，如果 CoeffAbsLevelGt2Flag 的值为 1 且 CoeffAbsLevelGt4Flag 的值为 0，则使用长度为 len=1 的定长码根据表 82 得到 synElVal 的值；

——否则，如果 CoeffAbsLevelGt4Flag 的值为 1 且 CoeffAbsLevelGt8Flag 的值为 0，则使用长度为 len=2 的定长码根据表 82 得到 synElVal 的值。

表82　synElVal 与二元符号串的关系

synElVal 的值	二元符号串							
0	1							
1	0	1	0					
2	0	1	1					
3	0	0	1	0	0			
4	0	0	1	0	1			
5	0	0	1	1	0			
6	0	0	1	1	1			
7	0	0	0	1	0	0	0	
8	0	0	0	1	0	0	1	
9	0	0	0	1	0	1	0	
binIndex	0	1	2	3	4	5	6	…

8.3.4.22　coeff_run 和 coeff_level_minus1 的反二值化方法

由二元符号串查表 83 得到 synElVal 的值。如果 synElVal 的值小于 TH，则使用截断一元码且 maxVal 等于 TH；如果 synElVal 的值大于或等于 TH，则二元符号串的前 TH 位均为 0，后续位为(synElVal-TH)对应的 0 阶指数哥伦布码。

表83　coeff_run 的值与二元符号串的关系

synElVal的值	二元符号串																				
0	1																				
1	0	1																			
2	0	0	1																		
3	0	0	0	1																	
4	0	0	0	0	1																
5	0	0	0	0	0	1															
6	0	0	0	0	0	0	1														
7	0	0	0	0	0	0	0	1													
8	0	0	0	0	0	0	0	0	1												
9	0	0	0	0	0	0	0	0	0	1											
10	0	0	0	0	0	0	0	0	0	0	1										
11	0	0	0	0	0	0	0	0	0	0	0	1									
12	0	0	0	0	0	0	0	0	0	0	0	0	1								

表 83（续）

synElVal的值	二元符号串																					
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1								
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1							
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1						
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1					
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0			
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1			
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
…																						
binIndex	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

8.3.4.23 isc_sv_x_abs_minus1[i]的反二值化方法

由二元符号串查表 84 得到 synElVal 的值。表 84 中，二元符号串是 synElVal 的值对应的 5 阶指数哥伦布码（见表 58）。

表84 synElVal 与二元符号串的关系

synElVal 的值	二元符号串										
0	1	0	0	0	0	0					
1	1	0	0	0	0	1					
2	1	0	0	0	1	0					
3	1	0	0	0	1	1					
4	1	0	0	1	0	0					
5	1	0	0	1	0	1					
6	1	0	0	1	1	0					
7	1	0	0	1	1	1					
8	1	0	1	0	0	0					
9	1	0	1	0	0	1					
10	1	0	1	0	1	0					
11	1	0	1	0	1	1					
12	1	0	1	1	0	0					
13	1	0	1	1	0	1					
14	1	0	1	1	1	0					
15	1	0	1	1	1	1					
16	1	1	0	0	0	0					
…											
31	1	1	1	1	1	1					

表 84（续）

synElVal 的值	二元符号串										
32	0	1	0	0	0	0	0	0			
…											
binIndex	0	1	2	3	4	5	6	7	8	9	…

8.3.4.24 isc_next_remaining_pixel_in_cu[i]的反二值化方法

isc_next_remaining_pixel_in_cu[i]二元符号串由三部分组成。

第 1 步，进行第一部分的反二值化。

a) 计算 maxValPrefix：

```
if (CurRemainingPixelQuarter <= 5) {
    maxValPrefix = 1
}
else if (CurRemainingPixelQuarter <= 21) {
    maxValPrefix = 2
}
else {
    maxValPrefix = 3
}
```

b) 由第一部分和 maxVal=maxValPrefix 查表 63 得到 synElVal 的值 a。如果 a 等于 0，位流中不存在第二部分和第三部分，isc_next_remaining_pixel_in_cu[i]的值为 0，结束反二值化；否则（即 a 大于 0），继续执行后续步骤。

第 2 步，进行第二部分的反二值化。

a) 计算 maxValInfix：

```
if (a == 1) {
    d = 1
    maxValInfix = Min(CurRemainingPixelQuarter - 2, 3)
}
else if (a == 2) {
    d = 5
    maxValInfix = Min(CurRemainingPixelQuarter - 6, 15)
}
else {
    d = 21
    maxValInfix = Min(CurRemainingPixelQuarter - 22, 255)
}
```

b) 令 n=Ceil(Log(maxValInfix + 1))，由第二部分和 len=n-1 查表 85 得到 synElVal 的值 b。如果 len 的值小于 1，二元符号串为空，b 的值为 0。

表85 synElVal 与二元符号串的关系（长度为 len 的定长码）

synElVal 的值	二元符号串				
0	0	0	…	0	0
1	0	0	…	0	1
2	0	0	…	1	0
3	0	0	…	1	1
…			…		
$2^{len}-4$	1	1	…	0	0
$2^{len}-3$	1	1	…	1	0
$2^{len}-2$	1	1	…	1	0
$2^{len}-1$	1	1	…	1	1
binIndex	0	1	…	len-2	len-1

第 3 步，进行第三部分的反二值化。

a) 如果 b 小于 2^{n}-maxValInfix-1，或 maxValInfix 为 0，则 k 等于 0；否则，k 等于 1。

b) 由第三部分和 len=k 查表 85 得到 synElVal 的值 c。如果 len 的值小于 1，二元符号串为空，c 的值为 0。

第 4 步，根据 d、b、c、k、n 和 maxValInfix 计算 isc_next_remaining_pixel_in_cu[i]的值。

$$isc\_next\_remaining\_pixel\_in\_cu[i] = d + (b << k) + c - (2^{n} - maxValInfix - 1) * k$$

9 解码过程

9.1 序列解码

序列解码过程如下：

第1步，将InitialFlag的值初始化为0。

第2步，解码序列头：

a) 如果 InitialFlag 的值为 1，直接执行步骤 b)，否则：
   1) 将 DOIPrev 和 DOICycleCnt 的值均初始化为 0；
   2) 将 InitialFlag 的值置为 1；
   3) 将 LibraryBufferEmpty 的值初始化为 1；
   4) 清空解码图像缓冲区。

b) 根据序列头中得到的 WeightQuantEnableFlag 和 LoadSeqWeightQuantDataFlag 的值，按附录 D 或表 16（见 7.1.2.4）确定 4×4 加权量化矩阵 $WeightQuantMatrix_{4x4}$ 和 8×8 加权量化矩阵 $WeightQuantMatrix_{8x8}$。

第3步，依次解码图像，直到遇到序列起始码或序列结束码或视频编辑码。

第4步，执行以下操作：

a) 如果遇到序列起始码，继续执行第 2 步；

b) 否则，如果遇到视频序列结束码或者视频编辑码，则按照 POI 从小到大的顺序依次输出解码图像缓冲区中每幅“未输出”的非知识图像，直到所有非知识图像均已输出。

9.2 图像解码

9.2.1 概述

图像的解码过程如下。

a) 解码图像头（见 9.2.2）。
b) 导出参考图像队列信息（见 9.2.3）。
c) 更新解码图像缓冲区（见 9.2.4）。
d) 构建参考图像队列（见 9.2.5）。
e) 解码当前图像的各个片（见 9.3），得到补偿后样本。
f) 如果DeblockingFilterDisableFlag的值为0,对补偿后样本进行去块效应滤波操作(见9.10)，得到滤波后样本；如果 DeblockingFilterDisableFlag 的值为 1，直接将补偿后样本作为滤波后样本。
g) 如果 SaoEnableFlag 的值为 1，对滤波后样本进行样值偏移自适应补偿操作（见 9.11.1），得到偏移后样本；如果 SaoEnableFlag 的值为 0 且 EsaoEnableFlag 的值为 1，对滤波后样本进行增强样值偏移自适应补偿（见 9.11.2），得到偏移后样本；如果 SaoEnableFlag 和 EsaoEnableFlag 的值均为 0，直接将滤波后样本作为偏移后样本。
h) 如果 CcsaoEnableFlag 的值为 1，对偏移后样本进行跨分量样值偏移自适应补偿操作（见 9.11.3），得到跨分量偏移后样本；如果 CcsaoEnableFlag 的值为 0，直接将偏移后样本作为跨分量偏移后样本。
i) 如果 AlfEnableFlag 的值为 1，对跨分量偏移后样本进行自适应修正滤波操作（见 9.12），得到重建样本；如果 AlfEnableFlag 的值为 0，直接将跨分量偏移后样本作为重建样本。
j) 重建样本构成当前图像的解码图像。
k) 如果 LibraryStreamFlag 的值等于 0，将当前图像的解码图像移入解码图像缓冲区，置该图像为“未输出”和“被参考”。
l) 输出解码图像（见 9.2.6）。
m) 存储当前解码图像的运动信息（见 9.5.7.10）和图像级量化参数。

9.2.2 图像头解码

图像头解码过程如下。

a) 如果当前图像起始码是 0x000001B3，则 PictureType 等于 0，当前图像是 I 图像。
b) 如果当前图像起始码是 0x000001B6：
   1) 如果 picture_coding_type 等于‘01’，则 PictureType 等于 1，当前图像是 P 图像；
   2) 如果 picture_coding_type 等于‘10’，则 PictureType 等于 2，当前图像是 B 图像。
c) 预测量化参数 PreviousQp 初始化为 picture_qp，预测量化参数增量 PreviousDeltaQP 初始化为 0，固定量化因子标志 FixedQP 等于 FixedPictureQpFlag。
d) 如果 LibraryStreamFlag 的值等于 0 且 DOI 小于 DOIPrev，则解码图像缓冲区中的所有非知识图像的解码顺序索引都减去 256，DOICycleCnt 加 1。
e) 如果 LibraryStreamFlag 的值等于 0，则 DOIPrev 等于 DOI。
f) POI 等于 DOI+PictureOutputDelay-OutputReorderDelay+256×DOICycleCnt。解码图像缓冲区中的非知识图像的显示顺序索引值与当前图像的显示顺序索引值的差值的绝对值应小于 128。
g) 距离索引 DistanceIndex 等于 POI 乘 2。
h) 如果 PictureWeightQuantEnableFlag 的值为 1,从当前图像头的位流中导出的加权量化矩阵：
   1) 按 9.2.7.2 确定 4×4 和 8×8 变换块的加权量化矩阵；

2) 根据上一步确定的 8×8 加权量化矩阵，按 9.2.7.3 确定 M_1×M_2变换块（M_1或 M_2大于 8）的加权量化矩阵。

符合本文件的位流应满足：如果当前图像是非知识图像，则当前图像的 DOI 不应与解码图像缓冲区中任意非知识图像的 DOI 相同，且当前图像的 POI 不应与解码图像缓冲区中任意非知识图像的 POI 相同。

9.2.3 导出参考图像队列信息

P 图像解码时只应使用参考图像队列 0。B 图像解码时可同时使用参考图像队列 0 和参考图像队列 1。导出参考图像队列 i（i 等于 0 或 1）的信息的方法如下。

a) 初始化 LibraryPictureExistFlag 和 j 为 0，doiBase 为当前图像的 DOI。

b) 进行 NumOfRefPic[i][RplsIndex[i]]次以下操作：

1) 如果 LibraryIndexFlag[i][RplsIndex[i]][j]的值等于 1，则令 RefPicDoiList[i][j]等于 ReferencedLibraryPictureIndex[i][RplsIndex[i]][j]，LibraryPictureExistFlag 等于 1，距离索引 DistanceIndex 等于当前图像的 POI 减 1 并乘 2，j 等于 j+1；

2) 如果 LibraryIndexFlag[i][RplsIndex[i]][j]的值等于 0，则令 RefPicDoiList[i][j]等于 doiBase-DeltaDoi[i][RplsIndex[i]][j]，并令 doiBase 等于 RefPicDoiList[i][j]，距离索引 DistanceIndex 等于 DOI 为 RefPicDoiList[i][j]的图像的 POI 乘 2，j 等于 j+1。

符合本文件的位流应满足以下要求。

a) 对于参考图像队列 i（i 等于 0 或 1），NumRefActive[i]的值应小于或等于 NumOfRefPic[i][RplsIndex[i]]的值。

b) 在参考图像队列 i（i 等于 0 或 1）中，对于任意 k 和 m（k 不等于 m），如果 LibraryIndexFlag[i][RplsIndex[i]][k]等于 LibraryIndexFlag[i][RplsIndex[i]][m]，则 RefPicDoiList[i][k]不应等于 RefPicDoiList[i][m]。

c) 如果当前图像是 RL 图像（记为 RL0）。

1) 对于 j 小于 NumRefActive[i]的所有 LibraryIndexFlag[i][RplsIndex[i]][j]均应为 1。

2) 如果位流中在 RL0 对应的序列头之前存在另一幅 RL 图像（记为 RL1），且 RL1 是位流中紧跟另一个序列头之后的 RL 图像中与 RL0 距离最近的 RL 图像，则应满足以下任一条件：

- RL0 所参考的每一幅知识图像均为 RL1 所参考的知识图像之一；
- RL0 所参考的知识图像最多只有 NumOfUpdatedLibrary 幅不同于 RL1 所参考的知识图像，且 RL0 与 RL1 的解码时间间隔至少为 MinLibraryUpdatePeriod 秒。

9.2.4 更新解码图像缓冲区

完成图像头解码及参考图像队列构建之后，在解码当前图像之前应先更新解码图像缓冲区。如果当前图像是知识位流中的图像，则不应更新解码图像缓冲区。

对解码图像缓冲区中所有非知识图像，如果该图像既不在参考图像队列0中又不在参考图像队列1中，则将该图像标记为“不被参考”；否则（该图像在参考图像队列0或参考图像队列1中），将该图像标记为“被参考”。如果一幅图像被标记为“不被参考”，则后续该图像不应再被标记为“被参考”。

对解码图像缓冲区依次进行以下操作：

a) 移出图像：

1) 移出解码图像缓冲区中所有被标记为“不被参考”且“已输出”的非知识图像；

2) 如果解码图像缓冲区中存在知识图像，且当前图像的 LibraryPictureExistFlag 等于 1，且当前图像所参考的知识图像的知识图像索引与解码图像缓冲区中的知识图像的知识图像索引不相同，则移出解码图像缓冲区中的知识图像并置 LibraryBufferEmpty 为 1。

b) 移入图像：如果 LibraryBufferEmpty 为 1 且当前图像的 LibraryPictureExistFlag 等于 1，

则将该知识图像索引对应的知识图像从外部移入解码图像缓冲区并置 LibraryBufferEmpty 为0。

符合本文件的位流应满足以下要求：

——在参考图像队列 i（i 等于 0 或 1）中最前面的 NumRefActive[i]幅参考图像的 temporal_id 应小于或等于当前图像的 temporal_id；

——解码过程中，当前解码图像及“未输出”图像（包括知识图像）的总数不应超过 MaxDpbSize 的值。

9.2.5 构建参考图像队列

构建参考图像队列 i（i 等于 0 或 1）的方法如下。

a) 初始化 j 为 0。

b) 进行 NumOfRefPic[i][RplsIndex[i]]次以下操作。

1) 如果 LibraryIndexFlag[i][RplsIndex[i]][j]的值等于 1，将知识图像索引等于 RefPicDoiList[i][j]的知识图像从解码图像缓冲区移入参考图像队列 i 中第 j 个位置。

2) 否则（如果 LibraryIndexFlag[i][RplsIndex[i]][j]的值等于 0），将解码图像缓冲区中解码顺序索引等于 RefPicDoiList[i][j]的非知识图像移入参考图像队列 i 中第 j 个位置。符合本文件的位流应满足，该非知识图像存在于解码图像缓冲区中。

3) j=j+1。

9.2.6 输出解码图像

执行以下步骤输出解码图像：

a) 如果当前序列的 LibraryStreamFlag 的值等于 1，则输出当前解码图像和对应的 LibraryPictureIndex。

b) 否则（LibraryStreamFlag 的值等于 0）：

1) 标记“可输出”图像：

- 在解码图像缓冲区中查找被标记为“未输出”且解码顺序索引与图像输出延迟之和小于或等于 DOI 的非知识图像，如果存在则标记该图像为“可输出”；
- 如果 PictureOutputDelay 的值为 0，则标记当前解码图像为“可输出”。

2) 如果存在“可输出”的非知识图像，则输出“可输出”图像中显示顺序索引最小的图像，并把该图像标记为“已输出”。

9.2.7 确定加权量化矩阵

9.2.7.1 概述

如果WeightQuantEnableFlag的值为0或PictureWeightQuantEnableFlag的值为0，则加权量化矩阵$WeightQuantMatrix_{8x8}$的元素$WeightQuantMatrix_{8x8}[i][j]$（i,j=0～7）的值均初始化为64；加权量化矩阵$WeightQuantMatrix_{4x4}$的元素$WeightQuantMatrix_{4x4}[i][j]$（i,j =0～3）的值均初始化为64；加权量化矩阵$WeightQuantMatrix_{M1xM2}$的元素$WeightQuantMatrix_{M1xM2}[i][j]$（i = 0～$M_1$-1，j = 0～$M_2$-1）的值均初始化为64，其中$M_1$或者$M_2$大于8；WqmShift初始化为2。

否则，如果WeightQuantEnableFlag的值为1且PictureWeightQuantEnableFlag的值为1，先根据9.2.7.2确定4×4和8×8加权量化矩阵，再根据9.2.7.3映射导出M_1×M_2加权量化矩阵，其中M_1或者M_2大于8；WqmShift初始化为2。

完成上述操作后，根据9.2.7.4修正加权量化矩阵$WeightQuantMatrix_{M1xM2}$。

9.2.7.2 确定 4×4 和 8×8 加权量化矩阵

如果picture_weight_quant_data_index的值为‘00’，则根据LoadSeqWeightQuantDataFlag的值，按附录D或表16（见7.1.2.4）确定4×4加权量化矩阵WeightQuantMatrix$_{4x4}$和8×8加权量化矩阵WeightQuantMatrix$_{8x8}$。

否则，如果picture_weight_quant_data_index的值为‘10’，则根据表16（见7.1.2.4）确定4×4加权量化矩阵WeightQuantMatrix$_{4x4}$和8×8加权量化矩阵WeightQuantMatrix$_{8x8}$。

否则，如果picture_weight_quant_data_index的值为‘01’，则按照以下步骤确定4×4和8×8加权量化矩阵：

第1步，确定当前图像的wqP（wqP的元素的取值范围应为1～255）。

a) 如果weight_quant_param_index的值为‘00’,则wqP[i]=WeightQuantParamDefault[i]（i=0～5），其中，加权量化参数WeightQuantParamDefault[i]＝{64，49，53，58，58，64}（i=0～5）。

b) 如果weight_quant_param_index的值为‘01’，从weight_quant_param_delta1[i]解析得到wqPDelta1[i]。wqP[i]=wqPDelta1[i]＋WeightQuantParamBase1[i]（i=0～5），其中，加权量化参数WeightQuantParamBase1[i]＝{67，71，71，80，80，106}（i=0～5）。

c) 如果weight_quant_param_index的值为‘10’，从weight_quant_param_delta2[i]解析得到wqPDelta2[i]。wqP[i]=wqPDelta2[i]＋WeightQuantParamBase2[i]（i=0～5），其中，加权量化参数WeightQuantParamBase2[i]＝{64，49，53，58，58，64}（i=0～5）。

第2步，根据WeightQuantModel确定8×8加权量化矩阵wqM8x8。

a) 如果WeightQuantModel的值为0，则wqM8x8的定义为：

```
wqM8x8 = {
{wqP[0], wqP[0], wqP[0], wqP[4], wqP[4], wqP[4], wqP[5], wqP[5]}
{wqP[0], wqP[0], wqP[3], wqP[3], wqP[3], wqP[3], wqP[5], wqP[5]}
{wqP[0], wqP[3], wqP[2], wqP[2], wqP[1], wqP[1], wqP[5], wqP[5]}
{wqP[4], wqP[3], wqP[2], wqP[2], wqP[1], wqP[5], wqP[5], wqP[5]}
{wqP[4], wqP[3], wqP[1], wqP[1], wqP[5], wqP[5], wqP[5], wqP[5]}
{wqP[4], wqP[3], wqP[1], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5]}
{wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5]}
{wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5]}}
```

b) 如果WeightQuantModel的值为1，则wqM8x8的定义为：

```
wqM8x8 = {
{wqP[0], wqP[0], wqP[0], wqP[4], wqP[4], wqP[4], wqP[5], wqP[5]}
{wqP[0], wqP[0], wqP[4], wqP[4], wqP[4], wqP[4], wqP[5], wqP[5]}
{wqP[0], wqP[3], wqP[2], wqP[2], wqP[2], wqP[1], wqP[5], wqP[5]}
{wqP[3], wqP[3], wqP[2], wqP[2], wqP[1], wqP[5], wqP[5], wqP[5]}
{wqP[3], wqP[3], wqP[2], wqP[1], wqP[5], wqP[5], wqP[5], wqP[5]}
{wqP[3], wqP[3], wqP[1], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5]}
{wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5]}
{wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5]}}
```

c) 如果WeightQuantModel的值为2，则wqM8x8的定义为：

```
wqM8x8 = {
{wqP[0], wqP[0], wqP[0], wqP[4], wqP[4], wqP[3], wqP[5], wqP[5]}
{wqP[0], wqP[0], wqP[4], wqP[4], wqP[3], wqP[2], wqP[5], wqP[5]}
{wqP[0], wqP[4], wqP[4], wqP[3], wqP[2], wqP[1], wqP[5], wqP[5]}
{wqP[4], wqP[4], wqP[3], wqP[2], wqP[1], wqP[5], wqP[5], wqP[5]}
{wqP[4], wqP[3], wqP[2], wqP[1], wqP[5], wqP[5], wqP[5], wqP[5]}
{wqP[3], wqP[2], wqP[1], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5]}
{wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5]}
{wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5], wqP[5]}}
```

第3步，根据WeightQuantModel确定4×4加权量化矩阵wqM4x4。

a) 如果 WeightQuantModel 的值为 0，则 wqM4x4 的定义为：

```
wqM4x4 = {
{wqP[0], wqP[4], wqP[3], wqP[5]}
{wqP[4], wqP[2], wqP[1], wqP[5]}
{wqP[3], wqP[1], wqP[1], wqP[5]}
{wqP[5], wqP[5], wqP[5], wqP[5]}}
```

b) 如果 WeightQuantModel 的值为 1，则 wqM4x4 的定义为：

```
wqM4x4 = {
{wqP[0], wqP[4], wqP[4], wqP[5]}
{wqP[3], wqP[2], wqP[2], wqP[5]}
{wqP[3], wqP[2], wqP[1], wqP[5]}
{wqP[5], wqP[5], wqP[5], wqP[5]}}
```

c) 如果 WeightQuantModel 的值为 2，则 wqM4x4 的定义为：

```
wqM4x4 = {
{wqP[0], wqP[4], wqP[3], wqP[5]}
{wqP[4], wqP[3], wqP[2], wqP[5]}
{wqP[3], wqP[2], wqP[1], wqP[5]}
{wqP[5], wqP[5], wqP[5], wqP[5]}}
```

注：8×8变换块的加权量化矩阵$WeightQuantMatrix_{8x8}$等于wqM8x8；4×4变换块的加权量化矩阵$WeightQuantMatrix_{4x4}$等于wqM4x4。

9.2.7.3 $M_1 \times M_2$加权量化矩阵的导出方法

如果M_1或者M_2中至少有一个大于或等于8，按以下方法确定$M_1 \times M_2$变换块所使用的加权量化矩阵$WeightQuantMatrix_{M1xM2}$：

a) 根据 9.2.7.2 确定 8×8 变换块所使用的加权量化矩阵 $WeightQuantMatrix_{8x8}$；

b) 确定 $M_1 \times M_2$ 变换块所使用的加权量化矩阵 $WeightQuantMatrix_{M1xM2}$ 为 $WeightQuantMatrix_{M1xM2}[x][y] = WeightQuantMatrix_{8x8}[x>>shift][y>>shift]$，$x=0 \sim M_1-1$，$y=0 \sim M_2-1$，其中 shift 等于 $Max(Log(M_1),Log(M_2))-3$。

9.2.7.4 $M_1 \times M_2$加权量化矩阵的修正方法

对加权量化矩阵$WeightQuantMatrix_{M1xM2}$的元素$WeightQuantMatrix_{M1xM2}[i][j]$（i=0～$M_1$-1，j=0～$M_2$-1），如果i的值大于或等于32，或j的值大于或等于32，则将$WeightQuantMatrix_{M1xM2}[i][j]$置为0；否则，$WeightQuantMatrix_{M1xM2}[i][j]$的值保持不变。

9.3 片解码

片解码过程如下。

a) 计算 LcuIndex。

```
LcuIndex = LcuRow * PictureWidthInLcu + LcuColumn
```

b) 如果 FixedPictureQpFlag 等于 0，预测量化参数 PreviousQp 等于 PatchQp，预测量化参数增量 PreviousDeltaQP 初始化为 0，固定量化因子标志 FixedQP 等于 FixedPatchQpFlag。

c) 以 LCU 为单位计算当前片的边界：

```
PatchLeftInLcu = LcuIndex % PictureWidthInLcu
PatchRightInLcu = PatchLeftInLcu + PatchSizeInLCU[PatchIndexY][PatchIndexX]->Width
PatchAboveInLcu = LcuIndex / PictureWidthInLcu
PatchBelowInLcu = PatchAboveInLcu + PatchSizeInLCU[PatchIndexY][PatchIndexX]->Height
```

d) 依次解码各个最大解码单元（见 9.4）。

9.4 最大编码单元解码

按片内的光栅扫描顺序依次解码最大解码单元，过程如下。

a) 如果当前最大编码单元是片中当前行的第一个最大编码单元，将历史运动信息表的候选项数 CntHmvp、历史帧内复制信息表的候选项数 CntIntraHmvp 和历史帧内复制信息表是否为空的标志 HbvpEmptyFlag 的值均初始化为 0；将串复制帧内预测模式的非普通串子模式的历史点预测信息表 PrevPpInfoList[28][2]中所有元素的值初始化为-1，将 PrevPvBufSize 的值初始化为 0，将 LcuRx0 和 LcuRy0 分别设为 x0 和 y0。

b) 初始化帧内预测模式频数表 FimcFrequencyList，其中 cntFimc 是频数表中高频模式候选项数，cntFimc 的值应为 2，ModeFimc[0]和 ModeFimc[1]是频数表中的高频模式。

```
if ((x0 % (1 << LcuSizeInBit) == 0) && (y0 % (1 << (LcuSizeInBit - 1)) == 0)) {
    cntFimc = 2
    ModeFimc[0] = 'Intra_Luma_Vertical'
    ModeFimc[1] = 'Intra_Luma_Horizontal'
    for (i=0; i<4; i++) {
        FimcFrequencyList[i] = 0
    }
}
```

c) 解码当前最大编码单元的编码树，依次解码编码树的各编码单元（见 9.5）。完成当前最大编码单元的解码后，按以下方法更新 LcuIndex。更新后，如果 LcuIndex / PictureWidthInLcu 的值大于或等于 PatchBelowInLcu，则结束当前片内所有最大解码单元的解码。

```
if ((LcuIndex + 1) % PictureWidthInLcu >= PatchRightInLcu) {
    LcuIndex += (PictureWidthInLcu - PatchSizeInLCU[PatchIndexY][PatchIndexX]->Width +1)
}
else {
    LcuIndex++
}
```

9.5 编码单元解码

9.5.1 概述

编码单元解码分为预测样本解码和残差样本解码。

预测样本解码包括：

a) 确定编码单元类型和相关信息（见 9.5.3）；

b) 对预测类型为普通帧内预测的编码单元分别导出其所包含的所有帧内预测块的普通帧内预测模式（见 9.5.6.3）并进行普通帧内预测（见 9.7.1），对预测类型为块复制帧内预测的编码单元分别导出其所包含的块矢量信息（见 9.5.6.4）并进行块复制帧内预测（见 9.7.2），对预测类型为串复制帧内预测的编码单元分别导出其所包含的串复制帧内预测信息（见 9.5.6.5）并进行串复制帧内预测（见 9.7.3），对预测类型为帧间预测的编码单元分别导出其所包含的所有帧间预测单元的运动信息（见 9.5.7）并进行帧间预测（见 9.8）；

c) 存储编码单元的运动信息(见 9.5.7.10.2),存储编码单元的空域帧内预测信息(见 9.5.6.8)。

残差样本解码包括：

a) 确定量化参数（见 9.5.2）；

b) 确定编码单元划分为变换块的方式（见 9.5.5）；

c) 依次解码各个变换块（见 9.6），如果当前编码单元的亮度编码块包含多个变换块，则首先将编码单元的残差样值矩阵 ResidueMatrix 中的值置为 0，然后将解码得到的 $M_1 \times M_2$ 大小的亮度变换块放置在编码单元的残差样值矩阵 ResidueMatrix 中以(blockX,blockY)为左上角，宽度为 M_1，高度为 M_2 的区域内。

完成预测样本解码和残差样本解码后，进行预测补偿得到补偿后样本（见9.9），然后更新等值串参考图像缓冲区（见9.5.6.5.4）。

9.5.2 确定量化参数

确定量化参数QP_X（X为Y、Cb或Cr）过程如下：

第1步，确定当前编码单元的量化参数CurrentQp，其取值范围应为0～(63 + 8 ×(BitDepth－8))：

——如果 FixedQP 为 1 或 CuDeltaQpFlag 为 0，则 CurrentQp 等于 PreviousQp 加上当前编码单元所在的最大编码单元的 LcuQpDelta；

——否则,如果 FixedQP 为 0 且 CuDeltaQpFlag 为 1 且当前编码单元仅包含色度分量,则 CurrentQp 等于当前编码单元右下角 4×4 子块对应的亮度编码单元的量化参数；

——否则,如果FixedQP为0且CuDeltaQpFlag为1且CuCtp为0,则CurrentQp等于PreviousCuQP；

——否则，CurrentQp 等于 PreviousCuQp 加上 CuDeltaQp。

PreviousQp的值等于上一个解码的最大编码单元的量化参数QP_Y。如果上一个解码的最大编码单元“不可用”或FixedQP等于1，则PreviousQP的值等于PatchQp。如果上一个解码的最大编码单元与当前编码单元不属于同一个片，则上一个解码的最大编码单元“不可用”。

如果当前编码单元左上角的坐标等于(CuQpGroupX,CuQpGroupY)，PreviousCuQq的值等于当前编码单元左边包含亮度分量的编码单元A的亮度量化参数QP_Y；否则，PreviousCuQP的值等于上一个解码的包含亮度分量的编码单元的亮度量化参数QP_Y。如果编码单元A“不可用”,则PreviousCuQp的值等于PatchQp。

第2步，亮度的量化参数QP_Y等于其所在编码单元的CurrentQp。QP_Y的取值范围应是0～[63 + 8 × (BitDepth − 8)]。

第3步，确定色度的量化参数QP_{Cb}和QP_{Cr}。

a) 计算 xCb 和 xCr，其取值范围应为-16～63。其中 IntraChromaPredMode 见 9.5.6.3。

```
xCb = Clip3(-16, 63, CurrentQp - 8 * (BitDepth - 8) + chroma_quant_param_delta_cb)
xCr = Clip3(-16, 63, CurrentQp - 8 * (BitDepth - 8) + chroma_quant_param_delta_cr)
if (IntraChromaPredMode == 'Intra_Chroma_PMC' || IntraChromaPredMode == 'Intra_Chroma_PMC_LT' ||
IntraChromaPredMode == 'Intra_Chroma_PMC_T' || IntraChromaPredMode == 'Intra_Chroma_PCM_L' ||
IntraChromaPredMode == 'Intra_Chroma_EPMC' || IntraChromaPredMode == 'Intra_Chroma_EPMC_LT' ||
IntraChromaPredMode == 'Intra_Chroma_EPMC_T' || IntraChromaPredMode == 'Intra_Chroma_EPMC_L' ||
IntraChromaPredMode == 'Intra_Chroma_EPMC2' || IntraChromaPredMode == 'Intra_Chroma_EPMC2_LT' ||
IntraChromaPredMode == 'Intra_Chroma_EPMC2_T' || IntraChromaPredMode == 'Intra_Chroma_EPMC2_L') {
    xCr = Clip3(-16, 63, CurrentQp - 8 * (BitDepth - 8) + chroma_quant_param_delta_cr + 1)
}
```

b) 分别以 xCb 和 xCr 为索引查表 86 得到 QP'_{cb} 和 QP'_{cr}，并计算 Cb、Cr 色度编码块的量化参数 QP_{cb} 和 QP_{cr}。

```
QPcb = Clip3(0, 63 + 8 * (BitDepth - 8), QP'cb + 8 * (BitDepth - 8))
QPcr = Clip3(0, 63 + 8 * (BitDepth - 8), QP'cr + 8 * (BitDepth - 8))
```

表86 色度量化参数与 xCb、xCr 的映射关系

xCb、xCr的值	QP'_{cb}和QP'_{cr}的值
＜43	xCb、xCr
43	42
44	43
45	43
46	44
47	44
48	45
49	45
50	46
51	46
52	47
53	47
54	48
55	48
56	48
57	49

表 86（续）

xCb、xCr的值	QP'_{Cb}和QP'_{Cr}的值
58	49
59	49
60	50
61	50
62	50
63	51

9.5.3 编码单元类型和相关信息

编码单元的分量模式包括‘COMPONENT_Luma’（仅包含亮度编码块）、‘COMPONENT_Chroma’（仅包含色度编码块）或‘COMPONENT_Luma_Chroma’（可包含亮度编码块和色度编码块）。

编码单元的预测模式限制包括‘PRED_Intra_Only’（可使用普通帧内预测、块复制帧内预测或串复制帧内预测）、‘PRED_Inter_Only’（仅使用帧间预测）或‘PRED_No_Constraint’（可使用普通帧内预测、块复制帧内预测、串复制帧内预测和帧间预测）。

如果IntraCuFlag的值为1，预测划分尺寸（PartSize）、预测划分方式（SplitMode）、预测参考模式（PredRefMode）和编码单元类型（CuType）由DtSplitFlag、DtSplitDir、DtSplitHqtFlag、DtSplitHadtFlag、DtSplitVqtFlag、DtSplitVadtFlag、块复制帧内模式标志（IbcCuFlag）和串复制帧内模式标志（IscCuFlag）查表87得到。

如果IntraCuFlag的值为1且当前编码单元的分量模式不是‘COMPONENT_Chroma’，帧内亮度预测块数（NumOfIntraPredBlock）由DtSplitFlag、DtSplitDir、DtSplitHqtFlag、DtSplitHadtFlag、DtSplitVqtFlag、DtSplitVadtFlag、块复制帧内模式标志（IbcCuFlag）和串复制帧内模式标志（IscCuFlag）查表87得到。

表87 编码单元类型和相关信息（IntraCuFlag 的值为 1）

DtSplitFlag	DtSplitDir	DtSplitHqtFlag	DtSplitHadtFlag	DtSplitVqtFlag	DtSplitVadtFlag	块复制帧内模式标志	串复制帧内预测标志	预测划分尺寸	预测划分方式	帧内亮度预测块数	预测参考模式	编码单元类型
0	—	—	—	—	—	0	0	SIZE_2Mx2N	I_NO_SPLIT	1	PRED_I	I_2M_2N
1	1	1	—	—	—	—	—	SIZE_2MxhN	HOR_tN	4	PRED_I	I_2M_hN
1	1	0	0	—	—	—	—	SIZE_2MxnU	HOR_UP	2	PRED_I	I_2M_nU
1	1	0	1	—	—	—	—	SIZE_2MxnD	HOR_DOWN	2	PRED_I	I_2M_nD
1	0	—	—	1	—	—	—	SIZE_hMx2N	VER_tN	4	PRED_I	I_hM_2N
1	0	—	—	0	0	—	—	SIZE_nLx2N	VER_LEFT	2	PRED_I	I_nL_2N

表87（续）

DtSplitFlag	DtSplitDir	DtSplitHqtFlag	DtSplitHadtFlag	DtSplitVqtFlag	DtSplitVadtFlag	块复制帧内模式标志	串复制帧内预测标志	预测划分尺寸	预测划分方式	帧内亮度预测块数	预测参考模式	编码单元类型
1	0	—	—	0	1	—	—	SIZE_nRx2N	VER_RIGHT	2	PRED_I	I_nR_2N
0	—	—	—	—	—	1	0	SIZE_2Mx2N	I_NO_SPLIT	1	PRED_I	IBC_2M_2N
0	—	—	—	—	—	0	1	SIZE_2Mx2N	I_NO_SPLIT	1	PRED_I	ISC_2M_2N

如果IntraCuFlag的值为0，预测划分尺寸（PartSize）、预测划分方式（SplitMode）和编码单元类型（CuType）由图像类型（PictureType）、跳过模式标志（SkipFlag）和直接模式标志（DirectFlag）查表88得到。

表88　编码单元类型和相关信息（IntraCuFlag 的值为 0）

图像类型	跳过模式标志	直接模式标志	预测划分尺寸	预测划分方式	编码单元类型
1	1	0	SIZE_2Mx2N	NO_SPLIT	P_Skip
	0	1	SIZE_2Mx2N	NO_SPLIT	P_Direct
	0	0	SIZE_2Mx2N	NO_SPLIT	P_2M_2N
2	1	0	SIZE_2Mx2N	NO_SPLIT	B_Skip
	0	1	SIZE_2Mx2N	NO_SPLIT	B_Direct
	0	0	SIZE_2Mx2N	NO_SPLIT	B_2M_2N

如果编码单元类型是‘P_Skip’‘P_Direct’‘B_Skip’或‘B_Direct’，先按9.5.7.8.2导出有效角度预测模式数（ValidMvapModeNum），再根据编码单元类型（CuType）、编码单元子类型索引（CuSubTypeIndex）、UMVE模式标志（UmveFlag）、ETMVP模式标志（EtmvpFlag）、仿射模式标志（AffineFlag）、AWP模式标志（AwpFlag）、子块时域运动信息标志（SbTmvpFlag）和预测参考模式索引（InterPredRefModeIndex）查表89得到编码单元子类型（CuSubType）和预测参考模式（predRefMode）。

表89中，如果SbTmvpEnableFlag的值为1且当前编码单元的宽度和高度均大于或等于16，则SbTmvpFlag的值为1；否则，SbTmvpFlag的值为0。符合本文件的位流应满足UmveFlag、EtmvpFlag、AffineFlag和AwpFlag中最多可有一个的值为1。

表89中，如果当前图像是P图像：

——如果当前编码单元类型是‘P_Skip’且CuSubTypeIndex大于或等于2：

1) 如果ValidMvapModeNum大于0，且CuSubTypeIndex小于2+ValidMvapModeNum，则编码单元子类型是‘P_Skip_Mvap’；

2) 否则，编码单元子类型是‘P_Skip_Hmvp’。

——如果当前编码单元类型是‘P_Direct’且CuSubTypeIndex大于或等于2：

1) 如果ValidMvapModeNum大于0，且CuSubTypeIndex小于2+ValidMvapModeNum，则编码单元子类型是‘P_Direct_Mvap’；

2) 否则，编码单元子类型是‘P_Direct_Hmvp’。

表89中，如果当前图像是B图像：

——如果当前编码单元类型是‘B_Skip’且CuSubTypeIndex大于或等于4：

1) 如果ValidMvapModeNum大于0，且CuSubTypeIndex小于4+ValidMvapModeNum，则编码单元子类型是‘B_Skip_Mvap’；

2) 否则，编码单元子类型是‘B_Skip_Hmvp’。

——如果当前编码单元类型是‘B_Direct’且CuSubTypeIndex大于或等于4：

1) 如果ValidMvapModeNum大于0，且CuSubTypeIndex小于4+ValidMvapModeNum，则编码单元子类型是‘B_Direct_Mvap’；

2) 否则，编码单元子类型是‘B_Direct_Hmvp’。

表89 编码单元子类型和预测参考模式

编码单元类型	编码单元子类型索引	UMVE模式标志	ETMVP模式标志	仿射模式标志	AWP模式标志	子块时域运动信息标志	预测参考模式索引	编码单元子类型	预测参考模式
P_Skip	0	0	0	0	0	0	—	P_Skip_Temporal	PRED_List0
	0	0	0	0	0	1	—	P_Skip_SbTemporal	PRED_List0
	1	0	0	0	0	—	—	P_Skip_Spatial_list0	PRED_List0
	2 ～ 1 + Max(NumOfMvapCand, NumOfHmvpCand)	0	0	0	0	—	—	P_Skip_Mvap或P_Skip_Hmvp	PRED_List0
	—	1	0	0	0	—	—	P_Skip_Umve	PRED_List0
	—	0	1	0	0	—	—	P_Skip_Etmvp	PRED_List0
	—	0	0	1	0	—	—	P_Skip_Affine	PRED_List0
	—	0	0	0	1	—	—	P_Skip_Awp	由9.5.7.8.9得到
P_Direct	0	0	0	0	0	0	—	P_Direct_Temporal	PRED_List0
	0	0	0	0	0	1	—	P_Direct_SbTemporal	PRED_List0
	1	0	0	0	0	—	—	P_Direct_Spatial_list0	PRED_List0
	2 ～ 1 + Max(NumOfMvapCand, NumOfHmvpCand)	0	0	0	0	—	—	P_Direct_Mvap或P_Direct_Hmvp	PRED_List0
	-	1	0	0	0	—	—	P_Direct_Umve	PRED_List0
	—	0	1	0	0	—	—	P_Direct_Etmvp	PRED_List0
	—	0	0	1	0	—	—	P_Direct_Affine	PRED_List0
	—	0	0	0	1	—	—	P_Direct_Awp	由9.5.7.8.9得到
P_2M_2N	—	—	—	0	—	—	—	P_Inter	PRED_List0
	—	—	—	1	—	—	—	P_Inter_Affine	PRED_List0

表 89（续）

编码单元类型	编码单元子类型索引	UMVE模式标志	ETMVP模式标志	仿射模式标志	AWP模式标志	子块时域运动信息标志	预测参考模式索引	编码单元子类型	预测参考模式
B_Skip	0	0	0	0	0	0	—	B_Skip_Temporal	PRED_List01
	0	0	0	0	0	1	—	B_Skip_SbTemporal	PRED_List01
	1	0	0	0	0	—	—	B_Skip_Spatial_list01	PRED_List01
	2	0	0	0	0	—	—	B_Skip_Spatial_list1	PRED_List1
	3	0	0	0	0	—	—	B_Skip_Spatial_list0	PRED_List0
	4 ～ 3 + Max(NumOfMvapCand, NumOfHmvpCand)	0	0	0	0	—	—	B_Skip_Mvap或B_Skip_Hmvp	由9.5.7.8.4和9.5.7.8.7.4得到
	—	1	0	0	0	—	—	B_Skip_Umve	由9.5.7.8.5得到
	—	0	1	0	0	—	—	B_Skip_Etmvp	由9.5.7.8.10得到
	—	0	0	1	0	—	—	B_Skip_Affine	由9.5.7.8.7得到
	—	0	0	0	1	—	—	B_Skip_Awp	由9.5.7.8.9得到
B_Direct	0	0	0	0	0	0	—	B_Direct_Temporal	PRED_List01
	0	0	0	0	0	1	—	B_Direct_SbTemporal	PRED_List01
	1	0	0	0	0	—	—	B_Direct_Spatial_list01	PRED_List01
	2	0	0	0	0	—	—	B_Direct_Spatial_list1	PRED_List1
	3	0	0	0	0	—	—	B_Direct_Spatial_list0	PRED_List0
	4 ～ 3 + Max(NumOfMvapCand, NumOfHmvpCand)	0	0	0	0	—	—	B_Direct_Mvap 或 B_Direct_Hmvp	由9.5.7.8.4和9.5.7.8.7.4得到
B_Direct	—	1	0	0	0	—	—	B_Direct_Umve	由9.5.7.8.5得到
	—	0	1	0	0	—	—	B_Direct_Etmvp	由9.5.7.8.10得到
	—	0	0	1	0	—	—	B_Direct_Affine	由9.5.7.8.7得到
	—	0	0	0	1	—	—	B_Skip_Awp	由9.5.7.8.9得到

表 89（续）

编码单元类型	编码单元子类型索引	UMVE模式标志	ETMVP模式标志	仿射模式标志	AWP模式标志	子块时域运动信息标志	预测参考模式索引	编码单元子类型	预测参考模式
B_2M_2N	—	—	—	0	—	—	0	B_Inter	PRED_List0
	—	—	—	0	—	—	1	B_Inter	PRED_List1
	—	—	—	0	—	—	2	B_Inter	PRED_List01
	—	—	—	1	—	—	0	B_Inter_Affine	PRED_List0
	—	—	—	1	—	—	1	B_Inter_Affine	PRED_List1
	—	—	—	1	—	—	2	B_Inter_Affine	PRED_List01

表87和表88中的预测划分方式（SplitMode）表示编码单元划分为帧内预测块或帧间预测单元的方式：

——如果帧内预测(包括普通帧内预测、块复制帧内预测和串复制帧内预测)划分方式是‘NO_SPLIT’，1 个 2M×2N 的编码单元划分为 1 个 2M×2N 的亮度预测块、1 个 M×N 的 Cb 预测块和 1 个 M×N 的 Cr 预测块（见图 11）；

——如果普通帧内预测划分方式是‘HOR_tN’，1 个 2M×2N 的编码单元划分为 4 个 2M×0.5N 的亮度预测块、1 个 M×N 的 Cb 预测块和 1 个 M×N 的 Cr 预测块（见图 12）；

——如果普通帧内预测划分方式是‘VER_tN’，1 个 2M×2N 的编码单元划分为 4 个 0.5M×2N 的亮度预测块、1 个 M×N 的 Cb 预测块和 1 个 M×N 的 Cr 预测块（见图 13）；

——如果普通帧内预测划分方式是‘HOR_UP’，1 个 2M×2N 的编码单元划分为 1 个 2M×0.5N 的亮度预测块、1 个 2M×1.5N 的亮度预测块、1 个 M×N 的 Cb 预测块和 1 个 M×N 的 Cr 预测块(见图 14)；

——如果普通帧内预测划分方式是‘HOR_DOWN’，1 个 2M×2N 的编码单元划分为 1 个 2M×1.5N 的亮度预测块、1 个 2M×0.5N 的亮度预测块、1 个 M×N 的 Cb 预测块和 1 个 M×N 的 Cr 预测块（见图 15）；

——如果普通帧内预测划分方式是‘VER_LEFT’，1 个 2M×2N 的编码单元划分为 1 个 0.5M×2N 的亮度预测块、1 个 1.5M×2N 的亮度预测块、1 个 M×N 的 Cb 预测块和 1 个 M×N 的 Cr 预测块（见图 16）；

——如果普通帧内预测划分方式是‘VER_RIGHT’，1 个 2M×2N 的编码单元划分为 1 个 1.5M×2N 的亮度预测块、1 个 0.5M×2N 的亮度预测块、1 个 M×N 的 Cb 预测块和 1 个 M×N 的 Cr 预测块（见图 17）；

——如果帧间预测划分方式是‘NO_SPLIT’，1 个 2M×2N 的编码单元不进行划分（见图 18）。

图11～图17中矩形里的数字表示各预测块的PredBlockOrder。图18中矩形里的数字表示各帧间编码单元的PredUnitOrder。

如果编码单元的编码单元类型为‘I_2M_2N’‘I_2M_hN’‘I_2M_nU’‘I_2M_nD’‘I_hM_2N’‘I_nL_2N’或‘I_nR_2N’，则其预测类型为普通帧内预测，其所包含的预测块的预测类型为普通帧内预测；否则，如果编码单元类型为‘IBC_2M_2N’，则其预测类型为块复制帧内预测，其所包含的预测块的预测类型为块复制帧内预测；否则，如果编码单元类型为‘ISC_2M_2N’，则其预测类型为串复制帧内预测，其所包含的预测块的预测类型为串复制帧内预测；否则该编码单元的预测类型为帧间预测，其所包含的预测单元和预测块的预测类型为帧间预测。

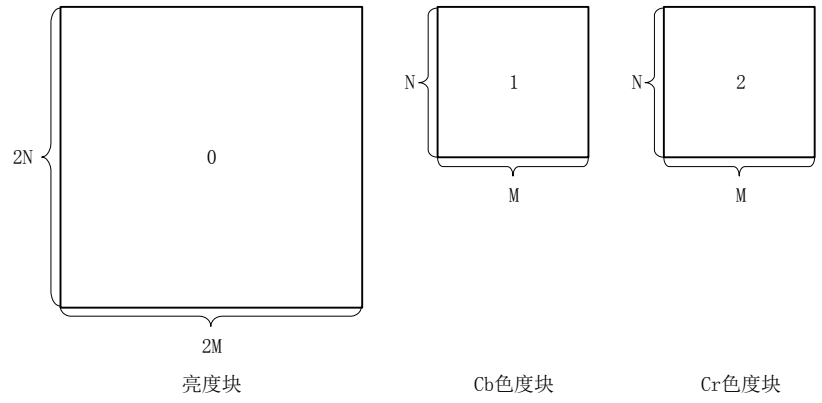


图11　帧内编码单元划分为预测块（预测划分方式‘NO_SPLIT’）

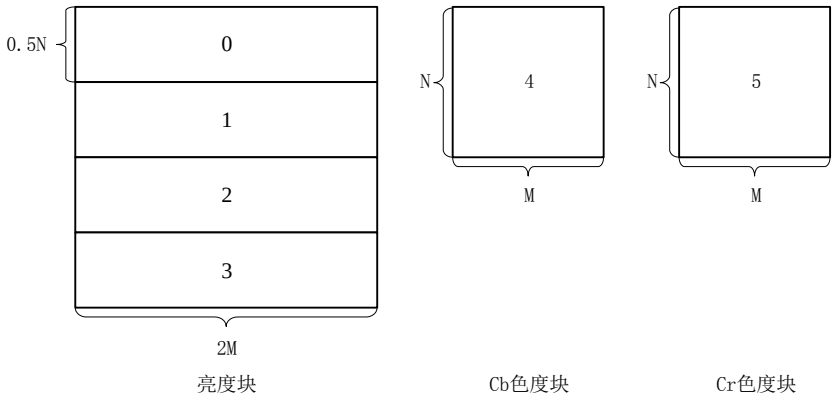


图12　普通帧内编码单元划分为预测块（预测划分方式‘HOR_tN’）

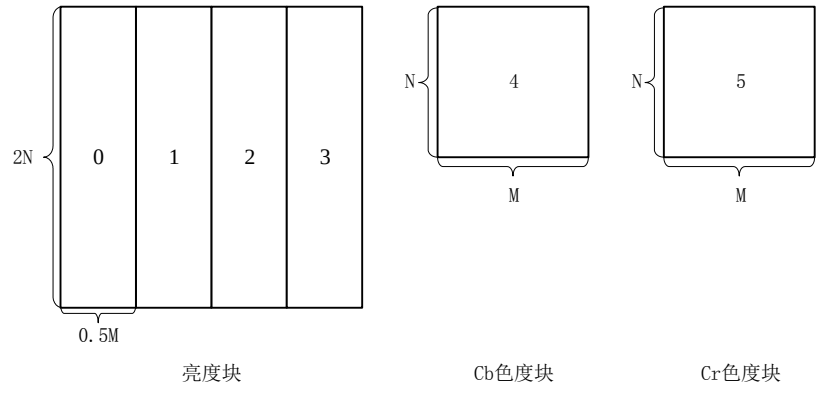


图13　普通帧内编码单元划分为预测块（预测划分方式‘VER_tN’）

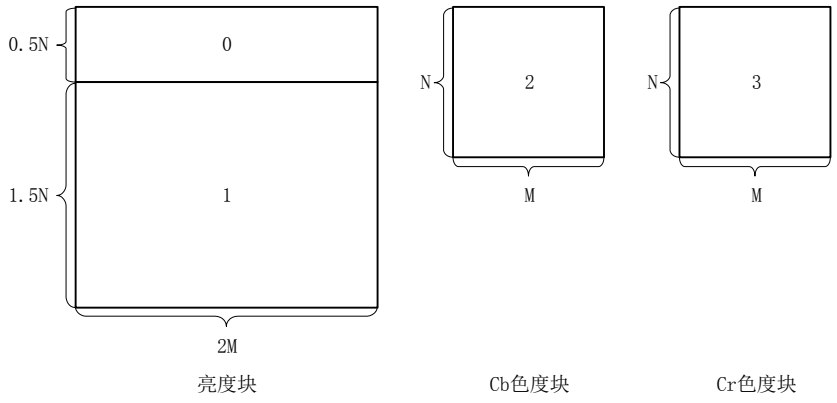


图14　普通帧内编码单元划分为预测块（预测划分方式‘HOR_UP’）

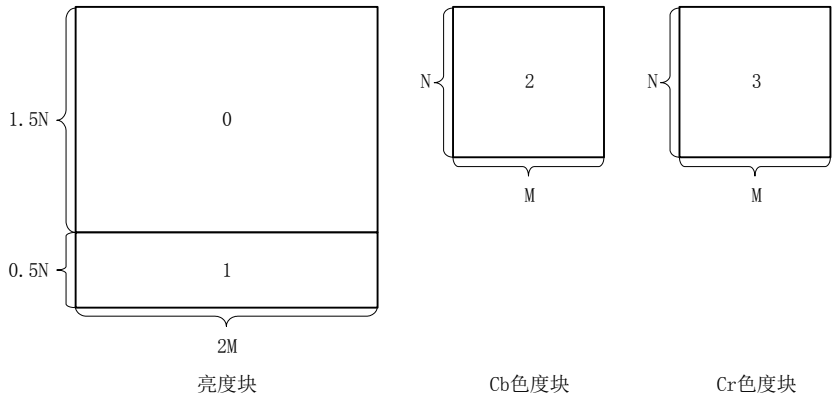


图15　普通帧内编码单元划分为预测块（预测划分方式‘HOR_DOWN’）

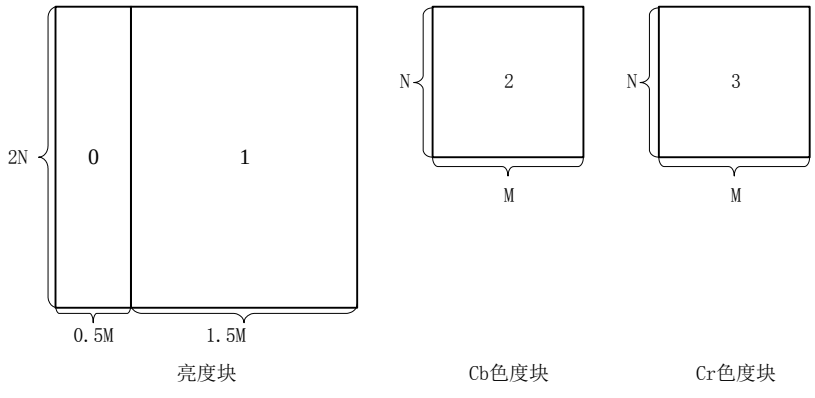


图16　普通帧内编码单元划分为预测块（预测划分方式‘VER_LEFT’）

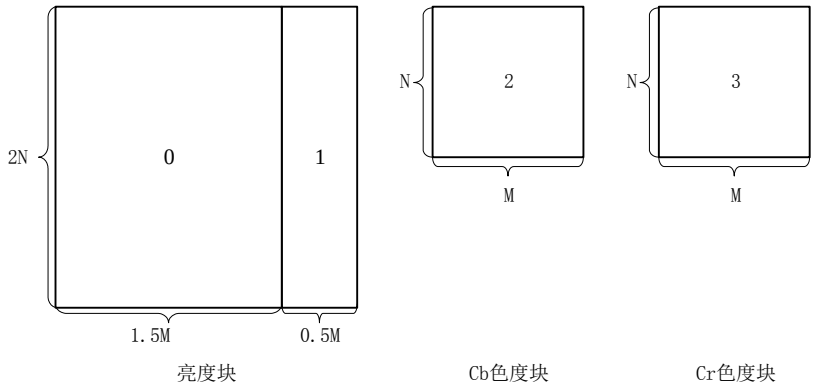


图17　普通帧内编码单元划分为预测块（预测划分方式‘VER_RIGHT’）

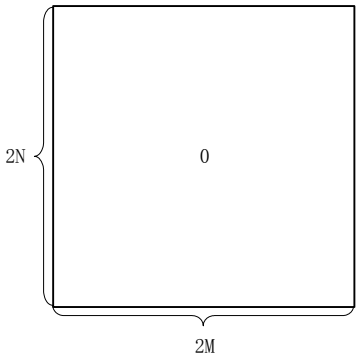


图18　帧间编码单元划分为预测单元（预测划分方式‘NO_SPLIT’）

9.5.4　相邻块

块E的相邻块A是样本(x_0-1, y_0)所在的块，块E的相邻块B是样本(x_0, y_0-1)所在的块，块E的相邻块C是样本(x_1+1, y_0-1)所在的块，块E的相邻块D是样本(x_0-1, y_0-1)所在的块，块E的相邻块F是样本（x_0-1，y_1）所在的块，块E的相邻块G是样本（x_1, y_0-1）所在的块。其中(x_0, y_0)是块E左上角样本在图像中的坐标，(x_1, y_0)是块E右上角样本在图像中的坐标，(x_0, y_1)是块E左下角样本在图像中的坐标。块E和它的相邻块A、B、C和D的空间位置关系见图19。

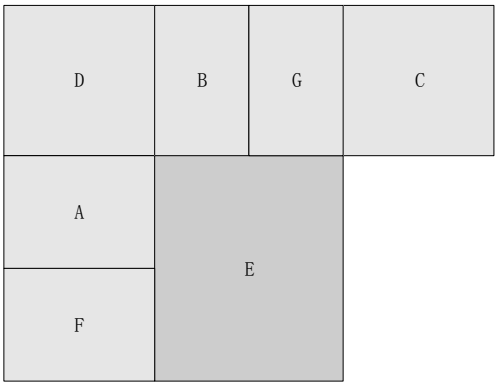


图19　块 E 和相邻块的空间位置关系

相邻块X（X为A、B、C、D、F或G）“存在”指该块应在图像内并且该块应与块E属于同一片；否则相邻块“不存在”。

如果块“不存在”或者尚未解码，则此块“不可用”；否则此块“可用”。如果图像样本所在的块“不存在”或者此样本尚未解码，则此样本“不可用”；否则此样本“可用”。

9.5.5 确定编码单元划分为变换块的方式

确定TransformSplitDirection的方法如下：

a) 如果 PbtCuFlag 的值为 1，则 TransformSplitDirection 的值为 1；
b) 否则，如果当前编码单元是普通帧内编码单元且预测划分方式是‘HOR_tN’‘HOR_UP’或‘HOR_DOWN’，则 TransformSplitDirection 的值为 2；
c) 否则，如果当前编码单元是普通帧内编码单元且预测划分方式是‘VER_tN’‘VER_LEFT’或‘VER_RIGHT’，则 TransformSplitDirection 的值为 3；
d) 否则，TransformSplitDirection 的值为 0。

当前编码单元划分为变换块的方式如下：

a) 如果 TransformSplitDirection 的值为 0，则当前编码单元划分为 3 个变换块（1 个矩形亮度变换块和 2 个矩形色度变换块，见图 20），NumOfTransBlocks 的值为 3；
b) 否则，如果 TransformSplitDirection 的值为 1，则当前编码单元划分为 6 个变换块（4 个矩形亮度变换块和 2 个矩形色度变换块，见图 21），NumOfTransBlocks 的值为 6；
c) 否则，如果 TransformSplitDirection 的值为 2，当前编码单元划分为 6 个变换块（4 个矩形亮度变换块和 2 个矩形色度变换块，见图 22），NumOfTransBlocks 的值为 6；
d) 否则，如果 TransformSplitDirection 的值为 3，则当前编码单元划分为 6 个变换块（4 个矩形亮度变换块和 2 个矩形色度变换块，见图 23），NumOfTransBlocks 的值为 6。

图 20 ～ 图 23 中数字为编码单元中变换块的顺序号。CuCtp 表示变换块顺序号为 0 到 NumOfTransBlocks-1的块是否包含非零变换系数。CuCtp的第n位（n等于0的位是最低有效位）等于‘0’表示顺序号为n的变换块没有非零系数，等于‘1’表示该变换块至少有一个非零系数。如果CuCtp的第n位等于‘1’，则当前$M_1 \times M_2$变换块应按9.6.3定义的方法进行反变换。

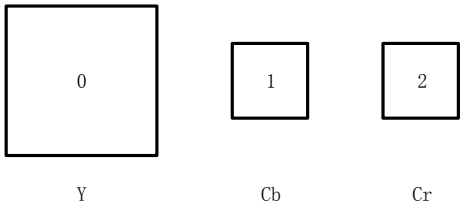


图20 编码单元划分为 1 个矩形亮度变换块和 2 个矩形色度变换块（4:2:0 格式）

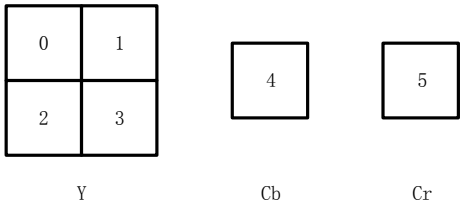


图21 编码单元划分为 4 个矩形亮度变换块和 2 个矩形色度变换块（4:2:0 格式）

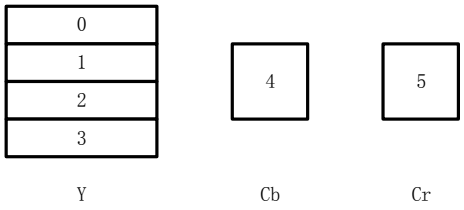


图22　编码单元划分为 4 个矩形亮度变换块和 2 个矩形色度变换块（4:2:0 格式）

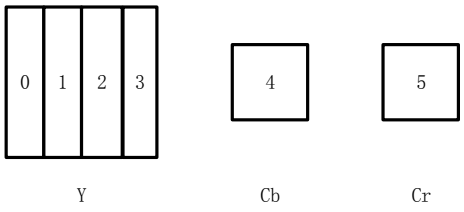


图23　编码单元划分为 4 个矩形亮度变换块和 2 个矩形色度变换块（4:2:0 格式）

9.5.6　帧内预测模式

9.5.6.1　概述

如果编码单元的帧内预测类型是普通帧内预测模式，按9.5.6.3导出当前编码单元的每一个预测块的亮度预测模式和色度预测模式。

如果编码单元的帧内预测类型是块复制帧内预测，按9.5.6.4导出当前编码单元的块矢量。

如果编码单元的帧内预测类型是串复制帧内预测，按9.5.6.5导出当前编码单元的全部串矢量和未匹配像素。

块复制帧内预测和串复制帧内预测可使用历史帧内复制信息表IntraHmvpCandList中的历史运动信息，包括位移矢量信息、位置信息、尺寸信息和重复次数。

9.5.6.2　获得相邻块空域帧内预测信息存储单元

当前预测单元的亮度预测块 E 的相邻亮度预测块 X（X 为 A、B、C、D、F 或 G）的空域帧内预测信息存储单元如下（见图 19）：

——相邻块 A 的空域帧内预测信息存储单元是样本(x_0-1, y_0)所对应的空域帧内预测信息存储单元；

——相邻块 B 的空域帧内预测信息存储单元是样本(x_0, y_0-1)所对应的空域帧内预测信息存储单元；

——相邻块 C 的空域帧内预测信息存储单元是样本(x_1+1, y_0-1)所对应的空域帧内预测信息存储单元；

——相邻块 D 的空域帧内预测信息存储单元是样本(x_0-1, y_0-1)所对应的空域帧内预测信息存储单元；

——相邻块 F 的空域帧内预测信息存储单元是样本(x_0-1, y_1)所对应的空域帧内预测信息存储单元；

——相邻块 G 的空域帧内预测信息存储单元是样本(x_1, y_0-1)所对应的空域帧内预测信息存储单元。

9.5.6.3　普通帧内预测模式

9.5.6.3.1　概述

如果当前分量是亮度分量且SawpFlag的值为0，按9.5.6.3.2导出当前预测块的亮度预测模式。如果当前分量是亮度分量且SawpFlag的值为1，按9.5.6.3.3导出当前预测块的亮度预测模式。按9.5.6.8存储空域帧内亮度预测模式。

如果当前分量是色度分量，按9.5.6.3.4导出当前预测块的色度预测模式。

9.5.6.3.2 普通亮度帧内预测模式

如果当前预测块E是亮度块，第1步计算当前预测块预测模式的预测值predIntraPredMode0和predIntraPredMode1的值。

——如果 PictureFimcEnableFlag 的值为 1，计算如下：

```
predIntraPredMode0 = Min(ModeFimc[0], ModeFimc[1])
predIntraPredMode1 = Max(ModeFimc[0], ModeFimc[1])
```

——如果 PictureFimcEnableFlag 的值为 0，按以下方法计算。

1) 如果左边预测块 A“存在”且为普通帧内预测，则将 A 的 IntraLumaPredMode 赋值给 intraPredModeA；否则 intraPredModeA 等于 0。
2) 如果上边预测块 B“存在”且为普通帧内预测，则将 B 的 IntraLumaPredMode 赋值给 intraPredModeB；否则 intraPredModeB 等于 0。
3) 如果 intraPredModeA 不等于 intraPredModeB，则 predIntraPredMode0 等于 Min(intraPredModeA, intraPredModeB)，predIntraPredMode1 等于 Max(intraPredModeA, intraPredModeB)。
4) 否则，如果 intraPredModeA 等于 0，则 predIntraPredMode0 等于 0，predIntraPredMode1 等于 2。
5) 否则，则 predIntraPredMode0 等于 0，predIntraPredMode1 等于 intraPredModeA。

第2步，计算IntraLumaPredMode的值。

——如果 IntraLumaPredModeIndex 的值为 0，则 IntraLumaPredMode 等于 predIntraPredMode0。

——否则，如果 IntraLumaPredModeIndex 的值为 1，则 IntraLumaPredMode 等于 predIntraPredMode1。

——否则，IntraPredMode 的值等于(IntraLumaPredModeIndex + LumaEipmIndex * 32)。

第3步，检查是否需要修改IntraLumaPredMode的值：

——如果 IntraPredMode 减 2 的值小于 predIntraPredMode0，则 IntraLumaPredMode 等于 IntraPredMode 减 2；

——否则，如果 IntraPredMode 减 1 的值大于 predIntraPredMode0 并且小于 predIntraPredMode1，则 IntraLumaPredMode 等于 IntraPredMode 减 1；

——否则，IntraLumaPredMode 的值不变。

第 4 步，如果 PictureFimcEnableFlag 的值为 1，根据当前预测单元的亮度帧内预测模式 IntraLumaPredMode按9.5.6.6更新帧内预测模式频数表。

第5步，根据IntraLumaPredMode的值，查表90得到亮度预测块的帧内预测模式。

表90 亮度预测块帧内预测模式

IntraLumaPredMode	帧内预测模式
0	Intra_Luma_DC
1	Intra_Luma_Plane

表 90（续）

IntraLumaPredMode	帧内预测模式
2	Intra_Luma_Bilinear
3～11	Intra_Luma_Angular
12	Intra_Luma_Vertical
13～23	Intra_Luma_Angular
24	Intra_Luma_Horizontal
25～32	Intra_Luma_Angular
33	Intra_Luma_PCM
34～65	Intra_Luma_Angular

亮度预测块帧内预测模式见图24。

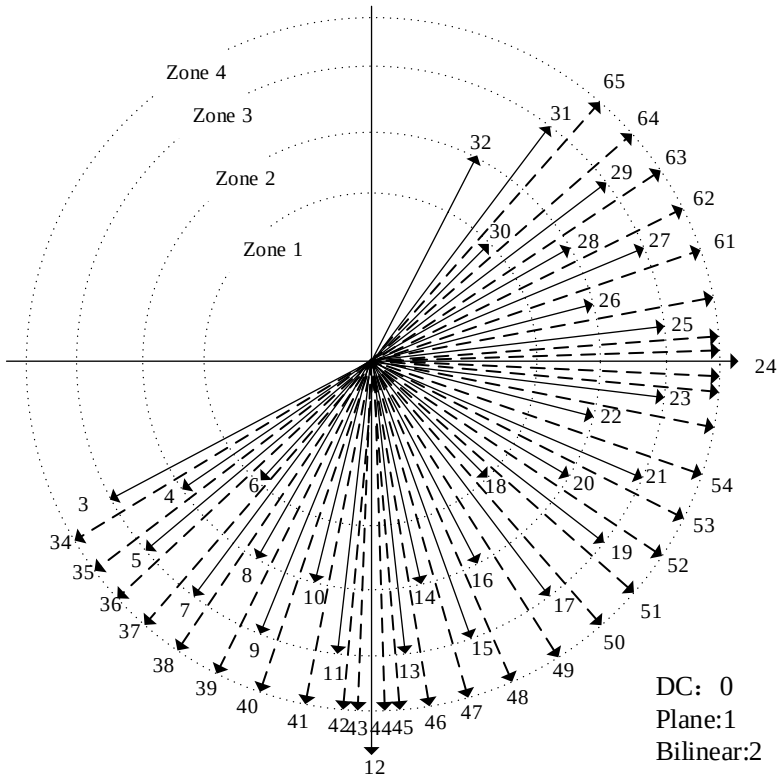


图24　亮度预测块帧内预测模式

9.5.6.3.3　空域角度加权预测亮度帧内预测模式

如果当前预测块E是亮度块，令modAngNum等于SawpIndex%8。

第1步，计算当前预测块预测模式的预测值predIntraPredMode[i]的值，i的取值范围为0～1。

a)　对 intraPredModeA 执行以下操作。

1)　如果左边预测块 A“存在”且为普通帧内预测，则将 A 的 IntraLumaPredMode 赋值给 intraPredModeA；否则，intraPredModeA 的值等于-1。

2)　对 intraPredModeA 进行模式有效性处理。

3) 如果 intraPredModeA 等于-1，则 intraPredModeA 等于表 91 中 modAngNum 对应的候选模式 0。

b) 对 intraPredModeB 执行以下操作。

1) 如果上边预测块 B“存在”且为普通帧内预测，则将 B 的 IntraLumaPredMode 赋值给 intraPredModeB；否则，intraPredModeB 的值等于-1。

2) 对 intraPredModeB 进行模式有效性处理。

3) 如果 intraPredModeB 等于-1，则 intraPredModeB 等于表 91 中 modAngNum 对应的候选模式 0。

c) 如果 intraPredModeA 等于 intraPredModeB，执行以下操作：

1) 如果 intraPredModeA 不等于表 91 中 modAngNum 对应的候选模式 0，则 intraPredModeA 等于表 91 中 modAngNum 对应的候选模式 0；

2) 否则，intraPredModeA 等于表 91 中 modAngNum 对应的候选模式 1。

d) predIntraPredMode[0]等于 Min(intraPredModeA,intraPredModeB)，predIntraPredMode[1]等于 Max(intraPredModeA,intraPredModeB)。

第2步，计算SawpPredMode0的值。

```
if (SawpPredMode0Index < 2) {
    SawpPredMode0 = predIntraPredMode[SawpPredMode0Index]
}
else {
    SawpPredMode0Index += 3
    SawpPredMode0 = (SawpPredMode0Index + (SawpPredMode0Index >= predIntraPredMode[0] ? 1 : 0) +
((SawpPredMode0Index + 1) >= predIntraPredMode[1] ? 1 : 0))
}
```

第3步，计算SawpPredMode1的值。

```
if (SawpPredMode0Index < 2) {
    if (SawpPredMode1Index >= SawpPredMode0Index) {
        SawpPredMode1Index = SawpPredMode1Index + 1
    }
}
if (SawpPredMode1Index < 2) {
    SawpPredMode1 = predIntraPredMode[SawpPredMode1Index]
}
else {
    SawpPredMode1Index += 3
    SawpPredMode1 = (SawpPredMode1Index + (SawpPredMode1Index >= predIntraPredMode[0] ? 1 : 0) +
((SawpPredMode1Index + 1) >= predIntraPredMode[1] ? 1 : 0))
}
```

9.5.6.3.3中对intraPredModeX进行以下模式有效性处理，其中X为A或B：

```
if (intraPredModeX <=4 || (intraPredModeX >= 31 && intraPredModeX <= 34 ) || intraPredModeX == 65) {
    intraPredModeX = -1
```

```
}
else if (intraPredModeX > 34) {
    if (intraPredModeX < 44) {
        intraPredModeX = intraPredModeX - 30
    }
    else if (intraPredModeX < 58) {
        intraPredModeX = intraPredModeX - 33
    }
    else {
        intraPredModeX = intraPredModeX - 34
    }
}
```

表91 modAngNum 对应的候选模式

modAngNum	候选模式0	候选模式1
0	30	6
1	27	24
2	24	12
3	21	24
4	18	24
5	15	12
6	12	24
7	9	12

9.5.6.3.4 普通色度帧内预测模式

如果当前预测块E是色度编码块，第1步计算isRedundant的值：

——如果 SawpFlag 等于 1，则 isRedundant 等于 0；

——否则，如果 E 对应的普通亮度预测模式 IntraLumaPredMode 等于 0、2、12 或 24，则 isRedundant 等于 1；

——否则，isRedundant 等于 0。

第2步，计算IntraChromaPredMode的值。

```
if ((TscpmEnableFlag || PmcEnableFlag) && (IntraChromaPredModeIndex == 1)) {
    IntraChromaPredMode = 5 + ChromaEipmIndex + 4 * ChromaPmcIndex + 4 * ChromaPmcParamIndex * (1 +
PicturePmcParamIndex)
}
else {
    if ((TscpmEnableFlag || PmcEnableFlag) && (IntraChromaPredModeIndex != 0)) {
        IntraChromaPredModeIndex = IntraChromaPredModeIndex - 1
    }
    if (! isRedundant) {
        IntraChromaPredMode = IntraChromaPredModeIndex
```

```
    }
    else {
        if (IntraLumaPredMode == 0) {
            predIntraChromaPredMode = 1
        }
        else if (IntraLumaPredMode == 2) {
            predIntraChromaPredMode = 4
        }
        else if (IntraLumaPredMode == 12) {
            predIntraChromaPredMode = 3
        }
        else if (IntraLumaPredMode == 24) {
            predIntraChromaPredMode = 2
        }
        if (IntraChromaPredModeIndex == 0) {
            IntraChromaPredMode = 0
        }
        else if (IntraChromaPredModeIndex < predIntraChromaPredMode) {
            IntraChromaPredMode = IntraChromaPredModeIndex
        }
        else {
            IntraChromaPredMode = IntraChromaPredModeIndex + 1
        }
    }
}
```

第3步，根据IntraChromaPredMode的值，查表92得到色度预测块的帧内预测模式。

表92　色度预测块普通色度帧内预测模式

IntraChromaPredMode	帧内预测模式
0	Intra_Chroma_DM（IntraLumaPredMode的值不等于33）
0	Intra_Chroma_PCM（IntraLumaPredMode的值等于33）
1	Intra_Chroma_DC
2	Intra_Chroma_Horizontal
3	Intra_Chroma_Vertical
4	Intra_Chroma_Bilinear
5	Intra_Chroma_TSCPM
6	Intra_Chroma_TSCPM_LT
7	Intra_Chroma_TSCPM_L
8	Intra_Chroma_TSCPM_T
9	Intra_Chroma_PMC
10	Intra_Chroma_PMC_LT

表 92（续）

IntraChromaPredMode	帧内预测模式
11	Intra_Chroma_PMC_L
12	Intra_Chroma_PMC_T
13	Intra_Chroma_EPMC
14	Intra_Chroma_EPMC_LT
15	Intra_Chroma_EPMC_L
16	Intra_Chroma_EPMC_T
17	Intra_Chroma_EPMC2
18	Intra_Chroma_EPMC2_LT
19	Intra_Chroma_EPMC2_L
20	Intra_Chroma_EPMC2_T

9.5.6.4 块复制帧内预测模式

9.5.6.4.1 概述

先按9.5.6.4.2得到块矢量的预测值MvPredXBv和MvPredYBv，再按9.5.6.4.3解码得到块矢量bvE。完成当前预测单元的解码后，令HbvpEmptyFlag的值为1，按9.5.6.4.4更新历史帧内复制信息表。

符合本文件的位流应满足9.5.6.4.5规定的限制。

当前块复制帧内预测单元的宽度和高度分别为widthCur和heightCur，当前预测单元左上角位置的坐标是(xCur,yCur)。历史运动信息IntraHmvpCandList[X]（X=0～NumOfIntraHmvpCand-1）的位移矢量是intraMvCandX，位置是(xCandX,yCandX)，尺寸是sizeCandX，重复次数是cntCandX。

如果AbvrIndex的值为0，块矢量的基本单位是1个整数样本；如果AbvrIndex的值为1，块矢量的基本单位是4个整数样本。

9.5.6.4.2 块矢量预测

9.5.6.4.2.1 导出块矢量预测值

本条定义导出块矢量预测值MvPredXBv和MvPredYBv的方法。

第1步，构建历史帧内复制信息类别列表blockMotionClass[Y]（Y=0～6）。

a) 历史帧内复制信息候选项数 NumAllowedHbvpCand = Min(CntIntraHmvp, NumOfIntraHmvpCand)。如果 NumAllowedHbvpCand 的值小于 2，不进行块矢量预测，立即结束本条的预测过程；否则继续执行下面的步骤。

b) 将 cbvpValidClass[Y]（Y = 0～6）初始化为 0，根据历史帧内复制信息分类方法对 IntraHmvpCandList[X]（X = NumAllowedHbvpCand-1～0）进行分类。

   1) 如果 CbvpIndex 等于 0，执行 9.5.6.4.2.2 的方法 a)。如果 cbvpValidClass[0]等于 1，结束分类过程。

   2) 如果 CbvpIndex 等于 1，依次执行 9.5.6.4.2.2 的方法 a)和 b)。如果 cbvpValidClass[1]等于 1，结束分类过程。

   3) 如果CbvpIndex等于2,依次执行9.5.6.4.2.2的方法a)、b)和g)。如果cbvpValidClass[2]等于 1，结束分类过程。

4) 如果CbvpIndex等于3,依次执行9.5.6.4.2.2的方法a)、b)和f)。如果cbvpValidClass[3]等于1，结束分类过程。
5) 如果CbvpIndex等于4,依次执行9.5.6.4.2.2的方法a)、b)和c)。如果cbvpValidClass[4]等于1，结束分类过程。
6) 如果CbvpIndex等于5,依次执行9.5.6.4.2.2的方法a)、b)和d)。如果cbvpValidClass[5]等于1，结束分类过程。
7) 如果CbvpIndex等于6,依次执行9.5.6.4.2.2的方法a)、b)和e)。如果cbvpValidClass[6]等于1，结束分类过程。

第2步，令Y等于CbvpIndex，按以下方法导出候选类列表CbvpCandidateList。

a) 如果 cbvpValidClass[Y]等于 1，CbvpCandidate 等于 blockMotionClass[Y]。
b) 否则，基于空域临近块，导出空域临近块块矢量候选 BlockMotionSpatialCandidate。
   1) 如果图像类型为 I 图像，且左上角坐标为(xCur-8,yCur+heightCur/2)的 4×4 子块所在的编码单元已重建，且与当前编码单元属于同一个最大编码单元，且是串复制帧内预测模式或块复制帧内预测模式，则 BlockMotionSpatialCandidateA 等于该 4×4 子块对应的空域帧内预测信息存储单元中的串矢量或块矢量。
   2) 否则，如果左上角坐标为(xCur-4,yCur+heightCur/2)的 4×4 子块所在的编码单元已重建且是串复制帧内预测模式，以该 4×4 子块中左下角样本的串矢量对应参考子串的第一个样本和最后一个样本在编码单元中的坐标分别为(x1,y1)和(x2,y2)，该参考子串的长度为 StrLen，该参考子串所在编码单元的宽度为 widthRef。如果该参考子串不满足条件(((y1 == y2) || (x1 == x2) || (StrLen==(y2 - y1 + 1) * widthRef)))，则 BlockMotionSpatialCandidateA 等于该 4×4 子块对应的空域帧内预测信息存储单元中的串矢量。
   3) 如果图像类型为 I 图像，且左上角坐标为(xCur+widthCur/2,yCur-8)的 4×4 子块所在的编码单元已重建，且与当前编码单元属于同一个最大编码单元，且是串复制帧内预测模式或块复制帧内预测模式，则 BlockMotionSpatialCandidateB 等于以该 4×4 子块对应的空域帧内预测信息存储单元中的串矢量或块矢量。
   4) 否则，如果左上角坐标为(xCur+widthCur/2,yCur-4)的 4×4 子块所在的编码单元已重建且是串复制帧内预测模式，以该 4×4 子块中左下角样本的串矢量对应参考子串的第一个样本和最后一个样本在编码单元中的坐标分别为(x1,y1)和(x2,y2)，该参考子串的长度为 StrLen，该参考子串所在编码单元的宽度为 widthRef。若该参考子串不满足条件(((y1 == y2) || (x1 == x2) || (StrLen == (y2 - y1 + 1) * widthRef)))，则 BlockMotionSpatialCandidateB 等于该 4×4 子块对应的空域帧内预测信息存储单元中的串矢量。
c) 按以下步骤基于空域临近块块矢量候选 BlockMotionSpatialCandidate 导出候选类 CbvpCandidate：
   1) 如果 CbvpIndex 为偶数，且 BlockMotionSpatialCandidateA 存在，则 CbvpCandidate 的块矢量等于 BlockMotionSpatialCandidateA；
   2) 如果 CbvpIndex 为奇数，且 BlockMotionSpatialCandidateB 存在，则 CbvpCandidate 的块矢量等于 BlockMotionSpatialCandidateB。
d) 如果对应的空域临近块块矢量候选 BlockMotionSpatialCandidate 也不存在,按以下步骤基于历史帧内复制信息表 IntraHmvpCandList 导出候选类 CbvpCandidate：
   1) 如果 CbvpIndex 小于 NumAllowedHbvpCand，则 CbvpCandidate 的块矢量等于 IntraHmvpCandList[NumAllowedHbvpCand%(CbvpIndex+1)]的块矢量；

2) 否则，CbvpCandidate 的块矢量等于 IntraHmvpCandList[0]的块矢量。

第3步，MvPredXBv和MvPredYBv分别等于CbvpCandidate的X分量和Y分量。

9.5.6.4.2.2 历史帧内复制信息分类方法

块矢量预测过程中对历史帧内复制信息IntraHmvpCandList[X]分类的方法如下：

a) 如果 sizeCandX 大于 32，且 cbvpValidClass[0]等于 0，将 IntraHmvpCandList[X]加入 blockMotionClass[0]，并令 cbvpValidClass[0]等于 1；

b) 否则，如果 cntCandX 大于或等于 3，且 cbvpValidClass[1]等于 0，将 IntraHmvpCandList[X]加入 blockMotionClass[1]，并令 cbvpValidClass[1]等于 1；

c) 否则，如果 xCandX 小于 xCur，且 yCandX 小于 yCur，且 cbvpValidClass[4]等于 0，将 IntraHmvpCandList[X]加入 blockMotionClass[4]，并令 cbvpValidClass[4]等于 1；

d) 否则，如果 xCandX 大于或等于 xCur + widthCur，且 yCandX 小于 yCur，且 cbvpValidClass[5]等于 0，将 IntraHmvpCandList[X]加入 blockMotionClass[5]，并令 cbvpValidClass[5]等于 1；

e) 否则，如果 yCandX 大于或等于 yCur + heightCur，且 xCandX 小于 xCur，且 cbvpValidClass[6]等于 0，将 IntraHmvpCandList[X]加入 blockMotionClass[6]，并令 cbvpValidClass[6]等于 1；

f) 否则，如果 yCandX 小于 yCur，且 xCandX 大于或等于 xCur，且 xCandX 小于 xCur + widthCur，且 cbvpValidClass[3]等于 0，将 IntraHmvpCandList[X]加入 blockMotionClass[3]，并令 cbvpValidClass[3]等于 1；

g) 否则，如果 xCandX 小于 xCur，且 yCandX 大于或等于 yCur，且 yCandX 小于 yCur + heightCur，且 cbvpValidClass[2]等于 0，将 IntraHmvpCandList[X]加入 blockMotionClass[2]，并令 cbvpValidClass[2]等于 1。

9.5.6.4.3 块矢量解码

块矢量解码过程如下：

```
if (NumAllowedHbvpCand < 2) {
    MvDiffXBv = MvDiffXBv << (AbvrIndex + 2)
    MvDiffYBv = MvDiffYBv << (AbvrIndex + 2)
    bvE->x = Clip3(-32768, 32767, MvDiffXBv)
    bvE->y = Clip3(-32768, 32767, MvDiffYBv)
}
else {
    MvDiffXBv = MvDiffXBv << (AbvrIndex + 2)
    MvDiffYBv = MvDiffYBv << (AbvrIndex + 2)
    MvPredXBv = Rounding(MvPredXBv, AbvrIndex + 2) << (AbvrIndex + 2)
    MvPredYBv = Rounding(MvPredYBv, AbvrIndex + 2) << (AbvrIndex + 2)
    bvE->x = Clip3(-32768, 32767, MvDiffXBv + MvPredXBv)
    bvE->y = Clip3(-32768, 32767, MvDiffYBv + MvPredYBv)
}
```

9.5.6.4.4 块复制预测模式下更新历史帧内复制信息表

位移矢量是块矢量bvE；位置是当前预测块左上角位置的坐标(xCur,yCur)；尺寸是当前预测块宽度与高度的积，即widthCur*heightCur。当前预测块的重复次数应初始化为0。

如果NumOfIntraHmvpCand的值大于0，根据当前块复制帧内预测块的运动信息，按9.5.6.7更新历史帧内复制信息表IntraHmvpCandList；否则，不执行更新操作。

9.5.6.4.5 块复制帧内预测的限制

从符合本文件的位流中解码得到的块矢量指向的参考块应限制在当前最大编码单元或左边N个最大编码单元的范围内。参考块的所有参考样本均应与当前块的样本属于同一个片，且已完全重建。N的值如下：

```
N = (1 << ((7 - LcuSizeInBit) << 1)) - (LcuSizeInBit < 7 ? 1 : 0)
```

令当前块左上角位置的坐标为(xCur,yCur)，参考块亮度分量左上角位置的坐标为(xRefTL,yRefTL)，参考块亮度分量右下角位置的坐标为(xRefBR,yRefBR)。坐标(xRefTL,yRefTL)和(xRefBR,yRefBR)应满足以下条件：

```
vSize = (LcuSize >= 64) ? 64 : LcuSize
xRefTL / vSize == xRefBR / vSize
yRefTL / vSize == yRefBR / vSize
```

当块矢量指向的参考块的左上角落在左边相邻最大编码单元，且最大编码单元的尺寸为128×128时，参考块左上角右移128像素后的位置所在的64×64区域的左上角应尚未重建，且坐标(xCur,yCur)应满足以下条件：

```
xCur != ((xRefTL + 128) / 64) * 64
yCur != (yRefTL / 64) * 64
```

9.5.6.5 串复制帧内预测模式

9.5.6.5.1 通则

串复制帧内预测模式有两种子模式：普通串子模式、非普通串子模式。

非普通串子模式的编码单元由等值串、单位基矢量串和未匹配样本组成。一个等值串的所有样本的值均等于同一个常现位置上的样本的值。点矢量(pvX,pvY)=(x,y) - (LcuRx0,LcuRy0)指向常现位置，其中(x,y)和(LcuRx0,LcuRy0)分别是常现位置和当前最大编码单元行左上角样本在图像中的坐标。单位基矢量串的串矢量(svX,svY)为(0,-1)。

串复制帧内预测模式的编码单元按往返扫描顺序划分为IscPartNum各部分，依次对各部分进行处理。

在普通串子模式中，如果IscMatchType[i](i=0～IscPartNum-1)的值为1，或IscMatchType[i]的值为0且NumMatchedPixel[i]大于0，则按9.5.6.5.2解码得到串矢量；否则，直接从位流中得到未匹配样本。完成当前预测单元的解码后，如果IscMatchType[i]为1且IscSvAboveFlag[i]的值不为1，则令HbvpEmptyFlag的值为1，按9.5.6.5.3更新历史帧内复制信息表。

在非普通串子模式中，如果StringType[i]的值等于0(即第i部分是单位基矢量串)，则按9.5.6.5.2解码得到串矢量；否则，如果StringType[i]的值等于1（即第i部分是等值串），则从点预测信息表中得到点矢量；否则，直接从位流中得到未匹配像素。完成当前预测单元的解码后，分别按9.5.6.5.4和9.5.6.5.5更新等值串参考图像缓冲区和历史点预测信息表。

符合本文件的位流应满足9.5.6.5.6规定的限制。

9.5.6.5.2 串矢量解码

如果IscSubtypeFlag的值等于1，则单位基矢量串的串矢量Sv[i]为：

```
Sv[i]->x = 0
Sv[i]->y = -1
```

否则，按以下方法解码得到第i部分的串矢量Sv[i]。

——如果 IscSvAboveFlag[i]等于 1，Sv[i]等于(0,-1)。

——否则，如果 IscSvRecentFlag[i] 等于 1，则 Sv[i] 等于 IntraHmvpCandList[CntIntraHmvp-1-IscSvRecentIndex[i]]的位移矢量。

——否则，Sv[i]为：

$$Sv[i]->x = (-1)^{IscSvXSign[i]} * IscSvXAbs[i]$$

$$Sv[i]->y = (-1)^{IscSvYSign[i]} * IscSvYAbs[i]$$

9.5.6.5.3 串复制预测模式下更新历史帧内复制信息表

位移矢量是当前预测块第i部分的串矢量Sv[i]；位置是当前预测块第i部分的第一个样本的位置坐标(xi,yi)；尺寸是当前预测块第i部分的串长度StrLen[i]。当前预测块第i部分的重复次数应初始化为0。

如果NumOfIntraHmvpCand的值大于0，根据当前串复制预测块的运动信息，按以下方法更新历史帧内复制信息表IntraHmvpCandList；否则，不执行更新操作。

a) 初始化 strPos 为 0。

b) i=0～IscPartNum-1，执行步骤 c)和步骤 d)。

c) 如果 IscMatchType[i]的值为 1。

1) 计算该部分的第一个样本的坐标：

```
xi = TravScan[Log(CuWidth)-2][Log(CuHeight)-2][strPos][0]
yi = TravScan[Log(CuWidth)-2][Log(CuHeight)-2][strPos][1]
```

2) 如果 Sv[i]不等于(0,-1)，按 9.5.6.7 更新历史帧内复制信息表。

3) strPos 加 StrLen[i]。

d) 否则，跳过历史帧内复制信息表的更新，strPos 加 4。

9.5.6.5.4 串复制预测模式下更新等值串参考样本缓冲区

令 vSize 的值等于 Min(LcuSize,64)，如果 PictureIscEvsUbvsEnableFlag 的值等于 1，且(x0+width)%vSize的值等于0，且(y0+height)%vSize的值等于0，按以下方法更新等值串参考样本缓冲区EvsDpb[3][28]。

第1步，将cnt的值初始化为0，令tmpEvsDpb[3][28]中的值全部等于0。

第2步，令i= 0～PrevPvBufSize-1，执行以下操作。

a) 检查历史点预测信息表 PrevPpInfoList[28]的第 i 个点矢量(pvX[i],pvY[i])是否满足以下条件。

1) 点矢量指向的参考像素限制在当前最大编码单元或左边 N 个最大编码单元的范围内，N 的大小由最大编码单元的大小决定。

```
N = (1 << ((7 - log2_lcu_size_minus2 + 2) << 1)) - (((log2_lcu_size_minus2 + 2) < 7) ? 1 : 0)
```

2) 令参考像素亮度分量 A 的位置坐标为(xRefA,yRefA)。当点矢量指向的参考像素落在相邻最大编码单元，且最大编码单元的尺寸为 128×128 时，该亮度分量 A 右移 128 像素后的位置所在的 64×64 区域的左上角应尚未重建，且上述坐标应满足以下条件。

```
xCur != ((xRefA + 128) / 64) * 64
yCur != (yRefA / 64) * 64
```

3) 点矢量指向的参考像素位于已完成重建的区域,且应与当前编码单元中的样本处于同一个片内。

b) 如果 PrevPPInfoList[i] 满足条件，PrevEvsDpbIndex[i] 的值等于 28，PrevEvsDpbReactivatedYonly[i]的值等于 0。

c) 否则，按以下方法将 PrevPPInfoList[i]对应的参考像素存入 tmpEvsDpb。

```
if (PrevEvsDpbIndex[i] < 28 ) {
    tmpEvsDpb[0][cnt] = EvsDpb[0][PrevEvsDpbIndex[i]]
    if (PrevFopYonly[k] == 0) {
        tmpEvsDpb[1][cnt] = EvsDpb[1][PrevEvsDpbIndex[i]]
        tmpEvsDpb[2][cnt] = EvsDpb[2][PrevEvsDpbIndex[i]]
    }
}
else {
    tmpEvsDpb[0][cnt] = LcuRowBufY[pvX[i]][pvY[i]]
    if (PrevFopYonly[k] == 0) {
        tmpEvsDpb[1][cnt] = LcuRowBufU[pvX[0]>>1][pvY[1]>>1]
        tmpEvsDpb[2][cnt] = LcuRowBufV[pvX[0]>>1][pvY[1]>>1]
    }
}
PrevEvsDpbIndex[i] = cnt
cnt = cnt + 1
```

第3步，令EvsDpb等于tmpEvsDpb。

9.5.6.5.5 串复制预测模式下更新历史点预测信息表

完成当前编码单元的解码后，如果当前编码单元采用串复制帧内预测非普通串子模式且(IscNumOfNewPv+IscNumofReusedPv)不等于0，则根据当前编码单元的点预测信息表PpInfoList更新历史点预测信息表PrevPpInfoList，并更新列表PrevFopYonly、PrevEvsDpbReactivatedYonly、PrevCompLumaFreqOccurPos和PrevEvsDpbIndex；否则，不执行本条定义的操作。

令PrevPvBufSize等于历史点预测信息表中点矢量的总数,PvBufSize等于当前编码单元的点预测信息表中点矢量的总数，tmpPvBuf[i]、tmpFlag[i]、tmpEvsDpbIndex[i]、tmpEvsDpbReactivatedYonly[i]和tmpCompLumaFreqOccurPos[i]（i=0～27）是点预测信息临时缓冲区，按以下方式执行更新操作：

```
tmpIndex = 0;
for (k=0; k<PrevPvBufSize; k++) {
```

```
    if (PrevPpInfoList[k][0] != -1 && PrevPpInfoList[k][1] != -1) {
        tmpPvBuf[tmpIndex][0] = PrevPpInfoList[k][0]
        tmpPvBuf[tmpIndex][1] = PrevPpInfoList[k][1]
        tmpFlag[tmpIndex]= PrevFopYonly[k]
        tmpEvsDpbIndex[tmpIndex]= PrevEvsDpbIndex[k]
        tmpEvsDpbReactivatedYonly[tmpIndex] = PrevEvsDpbReactivatedYonly[k]
        tmpCompLumaFreqOccurPos[tmpIndex] = PrevCompLumaFreqOccurPos[k]
        tmpIndex++
    }
}
PrevPvBufSize = Min(28, PvNum + tmpIndex)
for (k=0; k<PrevPvBufSize; k++) {
    if (k <PvBufSize) {
        PrevPpInfoList[k][0] = PpInfoList[k][0]
        PrevPpInfoList[k][1] = PpInfoList[k][1]
        PrevFopYonly[k] = FopYonly[k]
        PrevEvsDpbIndex[k] = EvsDpbIndex[k]
        PrevCompLumaFreqOccurPos[k] = CompLumaFreqOccurPos[k]
        PrevEvsDpbReactivatedYonly[k] = EvsDpbReactivatedYonly[k]
    }
    else {
         PrevPpInfoList[k][0] = tmpPvBuf[k-PvNum][0]
         PrevPpInfoList[k][1] = tmpPvBuf[k-PvNum][1]
         PrevFopYonly[k] = tmpFlag[k-PvNum]
         PrevEvsDpbIndex[k] = tmpEvsDpbIndex[k-PvNum]
         PrevCompLumaFreqOccurPos[k] = tmpCompLumaFreqOccurPos[k-PvNum]
         PrevEvsDpbReactivatedYonly[k] = tmpEvsDpbReactivatedYonly[k-PvNum]
    }
}
```

9.5.6.5.6 串复制帧内预测的限制

从符合本文件的位流中解码得到的串矢量指向的参考串或点矢量指向的常现位置中任意参考像素应限制在当前最大编码单元或左边N个最大编码单元的范围内。N的值如下：

```
N = (1 << ((7 - LcuSizeInBit) << 1)) - (LcuSizeInBit < 7 ? 1 : 0)
```

如果IscSubtypeFlag的值为0，令当前块左上角位置的坐标为(xCur,yCur)，参考串任意亮度分量A和B的位置坐标分别为(xRefA,yRefA)和(xRefB,yRefB)。坐标(xRefA,yRefA)和(xRefB,yRefB)应满足以下条件：

```
vSize = (LcuSize >= 64 ? 64 : LcuSize)
xRefA / vSize == xRefB / vSize
yRefA / vSize == yRefB / vSize
```

如果IscSubtypeFlag的值为0，且当参考串任意亮度分量A落在左边相邻最大编码单元，且最大编码单元的尺寸为128×128时，该亮度分量A右移128像素后的位置所在的64×64区域的左上角应尚未重建，且坐标(xCur,yCur)应满足以下条件：

```
xCur != ((xRefA + 128) / 64) * 64
yCur != (yRefA / 64) * 64
```

如果IscSubtypeFlag的值为0，当前串的参考串中的所有参考样本均应与当前串中的样本处于同一个片内，且已完全重建。如果串矢量Y分量的值大于或等于0，则串矢量指向的参考串中任意参考像素不应位于当前串内。

如果IscSubtypeFlag的值为1，令当前块左上角位置的坐标为(xCur,yCur)，常现位置亮度分量A的位置坐标为(xRef,yRef)。坐标(xCur,yCur)和(xRef,yRef)应满足以下条件：

```
xCur != ((xRef + 128) / 64) * 64
yCur != (yRef / 64) * 64
```

令numOfP等于IscNumOfNewPv+IscNumofReusedPv，如果IscSubtypeFlag的值为1：

——如果当前编码单元的宽度乘以高度的积大于 64，则 numOfP 的值不应大于 15；

——否则，如果当前编码单元的宽度乘以高度的积大于 32（即为 8×8、4×16 或 16×4），则 numOfP 的值不应大于 10；

——否则，如果当前编码单元的宽度乘以高度的积大于 16（即为 8×4 或 4×8），则 numOfP 的值不应大于 5；

——否则（即为 4×4），则 numOfP 的值不应大于 2。

当前编码单元使用的点矢量均存放在点预测信息表PpInfoList[15][2]中。PpInfoList[15][2]中的点矢量由两部分组成，一部分来自历史点预测信息表PrevPpInfoList[28][2]，历史点预测信息表中的点矢量的数量不应大于28；另一部分是当前编码单元中新出现的点矢量。

如果当前编码单元使用普通串子模式，则匹配串的个数、至少包含一个匹配样本的不完全匹配串的个数、未匹配样本的个数，以及IscPartNumSplit四者之和应小于或等于当前编码单元样本数的四分之一；否则，如果当前编码单元使用非普通串子模式，则等值串的个数、单位基矢量串的个数、未匹配样本的个数，以及IscPartNumSplit四者之和应小于或等于当前编码单元样本数的四分之一。其中普通串子模式的IscPartNumSplit按9.7.3.2得到，等值串及单位基矢量串模式的IscPartNumSplit按9.7.3.3得到。

9.5.6.6 更新帧内预测模式频数表

如果IntraLumaPredMode的值为‘Intra_Luma_DC’‘Intra_Luma_Bilinear’‘Intra_Luma_Vertical’或‘Intra_Luma_Horizontal’，则按以下方法更新帧内模式频数表FimcFrequencyList以及高频模式ModeFimc[0]和ModeFimc[1]。

帧内模式频数表FimcFrequencyList中，FimcFrequencyList[0]、FimcFrequencyList[1]、FimcFrequencyList[2]和FimcFrequencyList[3]分别对应‘Intra_Luma_DC’‘Intra_Luma_Bilinear’‘Intra_Luma_Vertical’和‘Intra_Luma_Horizontal’模式的频数。IntraLumaPredMode、ModeFimc[0]和ModeFimc[1]分别对应帧内预测模式频数表的索引为IntraLumaPredModeIndex、modeIndex0和modeIndex1。

```
FimcFrequencyList[IntraLumaPredModeIndex] = FimcFrequencyList[IntraLumaPredModeIndex] + 1
```

```
freqCurr = FimcFrequencyList[IntraLumaPredModeIndex]
mode0 = ModeFimc[0]
freq0 = FimcFrequencyList[modeIndex0]
mode1 = ModeFimc[1]
freq1 = FimcFrequencyList[modeIndex1]
if (freqCurr >= freq0) {
    ModeFimc[0] = IntraLumaPredMode
    if (mode0 != IntraLumaPredMode) {
        ModeFimc[1] = mode0
    }
}
else if (freqCurr >= freq1) {
    ModeFimc[1] = IntraLumaPredMode
}
```

9.5.6.7 更新历史帧内复制信息表

历史帧内复制信息表中的历史帧内复制信息记为IntraHmvpCandList[X]，包括位移矢量intraMvCandX，位置(xCandX,yCandX)，尺寸sizeCandX和重复次数cntCandX。

令intraCur是当前块的帧内复制信息，包括位移矢量intraMvCur，位置(xCur,yCur)，尺寸sizeCur和重复次数cntCur。更新历史帧内复制信息表的方法如下。

a) 将 X 和 cntCur 均初始化为 0。

b) 如果 CntIntraHmvp 等于 0,则 IntraHmvpCandList[CntIntraHmvp]为当前预测单元的帧内预测运动信息，CntIntraHmvp 加 1，结束本条的更新过程。

c) 否则，根据 intraMvCur 和 intraMvCandX 是否相等判断当前预测块的帧内预测运动信息和 IntraHmvpCandList[X]是否相同。

   1) 如果 intraMvCur 和 intraMvCandX 相同，执行步骤 d)；否则，X 加 1。

   2) 如果 X 小于 CntIntraHmvp，执行步骤 c)；否则，执行步骤 e)。

d) cntCur 等于 cntCandX 的值加 1。如果 sizeCur 小于 sizeCandX，则 xCur，yCur 和 sizeCur 分别等于 xCandx，yCandx 和 sizeCandX。

e) 如果 X 小于 CntIntraHmvp，则：

   1) i 从 X 到 CntIntraHmvp-1，令 IntraHmvpCandList[i]等于 IntraHmvpCandList[i+1]；

   2) IntraHmvpCandList[CntIntraHmvp-1]等于当前预测单元的帧内预测运动信息。

   否则，如果 X 等于 CntIntraHmvp 且 CntIntraHmvp 等于 NumOfIntraHmvpCand，则：

   1) i 从 0 到 CntIntraHmvp-1，令 IntraHmvpCandList[i]等于 IntraHmvpCandList[i+1]；

   2) IntraHmvpCandList[CntIntraHmvp-1]等于当前预测单元的帧内预测运动信息。

   否则，如果 X 等于 CntIntraHmvp 且 CntIntraHmvp 小于 NumOfIntraHmvpCand，则 IntraHmvpCandList[CntIntraHmvp]等于当前预测单元的帧内预测运动信息，CntIntraHmvp 加 1。

9.5.6.8 帧内预测信息存储

9.5.6.8.1 导出空域角度加权预测帧内亮度预测模式存储方式

令W和H是当前编码单元的宽度和高度，(x,y)是当前编码单元中的每个4×4块的左上角像素位置，pos_x等于((x >> 2) << 2) + 2，pos_y等于((y >> 2) << 2) + 2。按以下方法导出每个4×4块的帧内亮度预测模式存储方式SawpModeType。

a) 计算 stepIndex、angleIndex 和 angelAreaIndex 的值。

```
stepIndex = (SawpIndex >> 3) - 3
modAngNum = SawpIndex % 8
if (modAngNum == 2) {
    angleIndex = 7
}
else if (modAngNum == 6) {
    angleIndex = 8
}
else {
    angleIndex = modAngNum % 2
}
angleAreaIndex = modAngNum >> 1
```

b) 根据 angleAreaIndex 查表 93 得到 tP 和 fP，并计算当前编码单元中每个 4×4 块的帧内亮度预测模式存储方式 SawpModeType。

```
if (tP >= fP) {
    SawpModeType = 0
}
else {
    SawpModeType = 1
}
```

表93 angleArealndex 和 fP、tP 的对应关系

angleAreaIndex	fP	tP
0	H + (W >> angleIndex) - 2 + stepIndex * (((H + (W >> angleIndex)) >>2) -1)	(pos_y << 1) + ((pos_x << 1) >> angleIndex)
1	H + (W >> angleIndex) + stepIndex * (((H + (W >> angleIndex)) >> 2) -1) - ((W << 1) >> angleIndex)	(pos_y << 1) - ((pos_x << 1) >> angleIndex)
2	W + (H >> angleIndex) + stepIndex * (((W + (H >> angleIndex)) >> 2) -1) - ((H << 1) >> angleIndex)	(pos_x << 1) - ((pos_y << 1) >> angleIndex)
3	W + (H >> angleIndex) - 2 + stepIndex * (((W + (H >> angleIndex)) >> 2) -1)	(pos_x << 1) + ((pos_y << 1) >> angleIndex)

9.5.6.8.2 空域帧内预测信息存储

首先，确定样本(x,y)对应的空域帧内预测信息存储单元的左上角像素点位置(x_0,y_0)：

$x_0 = (x >> 2) << 2$

$y_0 = (y >> 2) << 2$

其次，设X是(x_0, y_0)亮度样本所在的预测单元，按以下方法将帧内亮度预测模式存储到样本(x,y)对应的空域帧内亮度预测模式存储单元：

a) 该空域帧内预测信息存储单元的预测类型为当前编码单元的预测类型。

b) 如果(x0,y0)亮度样本所在的预测单元是普通帧内预测且SawpFlag为0，则该空域帧内亮度预测模式存储单元的帧内亮度预测模式等于IntraLumaPredMode。

c) 否则，如果(x0,y0)亮度样本所在的预测单元是普通帧内预测且SawpFlag为1，按9.5.6.8.1导出空域角度加权预测帧内亮度预测模式存储方式SawpModeType：

1) 如果SawpModeType等于0，则该空域帧内亮度预测模式存储单元的帧内亮度预测模式等于SawpPredMode0；

2) 否则（SawpModeType等于1），则该空域帧内亮度预测模式存储单元的帧内亮度预测模式等于SawpPredMode1。

d) 否则，如果当前编码单元的预测类型为块复制帧内预测，该空域帧内预测信息存储单元的块矢量等于当前编码单元的块矢量。

e) 否则，如果当前编码单元的预测类型为串复制帧内预测，该空域帧内预测信息存储单元的串矢量等于坐标为(x_0, y_0+3)样本对应的串矢量。

9.5.7 运动信息

9.5.7.1 概述

本条定义了导出预测单元运动信息的方法。

如果当前预测单元包含亮度分量，执行以下处理。

第1步，确定MvExistL0、MvExistL1的值。

——如果同时满足以下条件，则当前预测单元的MvExistL0的值为1，否则MvExistL0的值为0：

1) 当前预测单元所在编码单元的编码单元类型是‘B_2M_2N’或‘P_2M_2N’；

2) 当前预测单元的预测参考模式是‘PRED_List0’或‘PRED_List01’。

——如果同时满足以下条件，则当前预测单元的MvExistL1的值为1，否则MvExistL1的值为0：

1) 当前预测单元所在编码单元的编码单元类型是‘B_2M_2N’或‘P_2M_2N’；

2) 当前预测单元的预测参考模式是‘PRED_List1’或‘PRED_List01’。

第2步，导出运动信息。

——如果当前预测单元所在编码单元的编码单元类型是‘P_Skip’或‘P_Direct’或‘B_Skip’或‘B_Direct’，按9.5.7.8导出运动信息。

——否则：

1) 如果当前预测单元的MvExistL0为1，先按9.5.7.6得到运动矢量预测值，再按9.5.7.7解码得到L0运动矢量；

2) 如果当前预测单元的MvExistL1为1，先按9.5.7.6得到运动矢量预测值，再按9.5.7.7解码得到L1运动矢量。

——按9.5.7.11导出相邻块单向运动信息矩阵。

第3步，按9.5.7.9用当前预测单元的运动信息更新历史运动信息表。

如果当前预测单元的预测参考模式是‘PRED_List01’，或AffineSubblockSizeFlag的值为1，则该预测单元的各个8×8子块运动矢量可不同；否则该预测单元的各个4×4子块运动矢量可不同。

运动矢量的基本单位如下。

——如果当前编码单元不是仿射模式，根据 AmvrIndex 的值决定运动矢量的基本单位：

1) 如果 AmvrIndex 的值为 0，运动矢量的基本单位是 1/4 个整数样本；
2) 如果 AmvrIndex 的值为 1，运动矢量的基本单位是 1/2 个整数样本；
3) 如果 AmvrIndex 的值为 2，运动矢量的基本单位是 1 个整数样本；
4) 如果 AmvrIndex 的值为 3，运动矢量的基本单位是 2 个整数样本；
5) 如果 AmvrIndex 的值为 4，运动矢量的基本单位是 4 个整数样本。

——否则（编码单元是仿射模式），根据 AffineAmvrIndex 的值决定运动矢量的基本单位：

1) 如果 AffineAmvrIndex 的值为 0，运动矢量的基本单位是 1/4 个整数样本；
2) 如果 AffineAmvrIndex 的值为 1，运动矢量的基本单位是 1 个整数样本；
3) 如果 AffineAmvrIndex 的值为 2，运动矢量的基本单位是 1/16 个整数样本。

9.5.7.2 判断运动信息的异同

如果两个运动信息满足下面一个或多个条件，则两个运动信息不同；否则两个运动信息相同：

——InterPredRefMode 不同；

——InterPredRefMode 均为 0，L0 运动矢量不相等或 L0 参考索引不相等；

——InterPredRefMode 均为 1，L1 运动矢量不相等或 L1 参考索引不相等；

——InterPredRefMode 均为 2，L0 运动矢量、L1 运动矢量、L0 参考索引或 L1 参考索引中有任意一个不相等。

9.5.7.3 判断参考帧的异同

如果两个参考帧满足下面一个或多个条件，则两个参考帧不同；否则两个参考帧相同：

——其中一个参考帧是非知识图像，另一个参考帧是知识图像；

——两个参考帧都是非知识图像，对应的参考图像 DOI 不相等；

——两个参考帧都是知识图像，对应的知识图像索引不相等。

9.5.7.4 导出仿射预测单元的子块信息和控制参数

9.5.7.4.1 概述

先按9.5.7.4.2导出子块仿射运动矢量组，如果AsrEnableFlag的值为1，再按9.5.7.4.3导出仿射预测单元的子块偏差矩阵和仿射样本改善信息。

9.5.7.4.2 导出仿射预测单元的子块运动矢量组

如果仿射控制点运动矢量组中有3个运动矢量，运动矢量组表示为mvsAffine(mv0,mv1,mv2)；否则如果仿射控制点运动矢量组中有2个运动矢量，运动矢量组表示为mvsAffine(mv0,mv1)。

第1步，计算dHorX、dVerX、dHorY 和dVerY。

——dHorX 和 dHorY 的计算如下：

```
dHorX = (mv1->x - mv0->x) << (7 - Log(width))
dHorY = (mv1->y - mv0->y) << (7 - Log(width))
```

——如果 mvsAffine 中有 3 个运动矢量，dVerX 和 dVerY 的计算如下：

```
dVerX = (mv2->x - mv0->x) << (7 - Log(height))
```

```
dVerY = (mv2->y - mv0->y) << (7 - Log(height))
```

——否则（mvsAffine 中有 2 个运动矢量），dVerX 和 dVerY 的计算如下：

```
dVerX = - dHorY
dVerY = dHorX
```

令(xE,yE)是当前预测单元亮度预测块左上角样本在当前图像的亮度样本矩阵中的位置，当前预测单元的宽度和高度分别是width和height，(x,y)是亮度子块左上角位置的坐标，子块的宽度和高度分别是subWidth和subHeight，当前预测单元亮度预测块的左上角样本所在的子块是A子块，右上角样本所在的子块是B子块，左下角样本所在的子块是C子块，见图25。

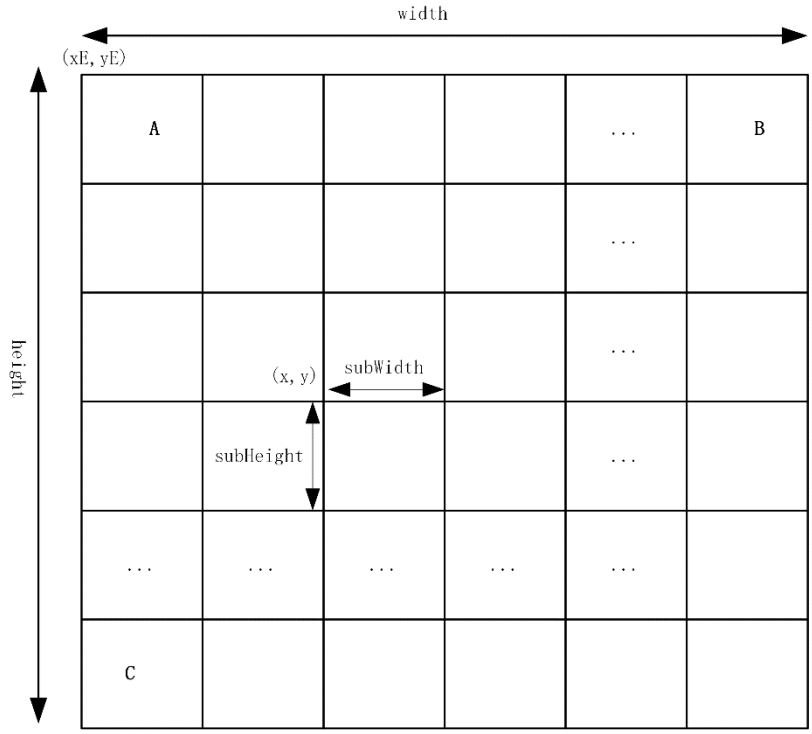


图25　样本的位置

第2步，计算亮度子块的运动矢量mvE。

——如果当前预测单元的预测参考模式是‘Pred_List01’，或 AffineSubblockSizeFlag 等于 1，则 subWidth 和 subHeight 均等于 8，(x,y)是 8×8 子块左上角位置的坐标，计算每个 8×8 亮度子块的运动矢量：

1) 如果当前子块是 A，则 xPos 和 yPos 均等于 0；
2) 否则，如果当前子块是 B，则 xPos 等于 width，yPos 等于 0；
3) 否则,如果当前子块为C且mvsAffine中有3个运动矢量,则xPos等于0,yPos等于height；
4) 否则，xPos 等于(x-xE)+4，yPos 等于(y-yE)+4；
5) 当前 8×8 子块的运动矢量 mvE：

```
mvE->x = Clip3(-131072, 131071, Rounding((mv0->x << 7) + dHorX * xPos + dVerX * yPos, 7))
mvE->y = Clip3(-131072, 131071, Rounding((mv0->y << 7) + dHorY * xPos + dVerY * yPos, 7))
```

——如果当前预测单元的预测参考模式是‘PRED_List0’或‘PRED_List1’且AffineSubblockSizeFlag等于0，则subWidth和subHeight均等于4，(x,y)是4×4子块左上角位置的坐标，计算每个4×4亮度子块的运动矢量：

1) 如果当前子块是A，则xPos和yPos均等于0；
2) 否则，如果当前子块是B，则xPos等于width，yPos等于0；
3) 否则,如果当前子块是C且mvAffine中有3个运动矢量,则xPos等于0,yPos等height；
4) 否则，xPos等于(x-xE)+2，yPos等于(y-yE)+2；
5) 当前4×4子块的运动矢量mvE：

```
mvE->x = Clip3(-131072, 131071, Rounding((mv0->x << 7) + dHorX * xPos + dVerX * yPos, 7))
mvE->y = Clip3(-131072, 131071, Rounding((mv0->y << 7) + dHorY * xPos + dVerY * yPos, 7))
```

符合本文件的位流中，按以下方法计算MemoryAccess的值：

——如果mvAffine中有2个运动矢量，计算mv2：

```
mv2->x = Clip3(-131072, 131071, Rounding((mv0->x << 7) + dVerX*height, 7))
mv2->y = Clip3(-131072, 131071, Rounding((mv0->y << 7) + dVerY*height, 7))
```

——计算mv3：

```
mv3->x = Clip3(-131072, 131071, Rounding((mv0->x << 7) + dHorX*width + dVerX*height, 7))
mv3->y = Clip3(-131072, 131071, Rounding((mv0->y << 7) + dHorY*width + dVerY*height, 7))
```

——计算maxX、minX、maxY和minY：

```
maxX = Max((mv0->x >> 4), Max((mv1->x >> 4) + width, Max((mv2->x >> 4), (mv3->x >> 4) + width)))
minX = Min((mv0->x >> 4), Min((mv1->x >> 4) + width, Min((mv2->x >> 4), (mv3->x >> 4) + width)))
maxY = Max((mv0->y >> 4), Max((mv1->y >> 4), Max((mv2->y >> 4) + height, (mv3->y >> 4) + height)))
minY = Min((mv0->y >> 4), Min((mv1->y >> 4), Min((mv2->y >> 4) + height, (mv3->y >> 4) + height)))
```

——计算MemoryAccess：

```
MemoryAccess = (maxX - minX + 7) * (maxY - minY + 7)
```

符合本文件的位流中MemoryAccess的值应满足以下条件：

```
MemoryAccess <= ((width + 7 + width / 4) * (height + 7 + height / 4))
```

9.5.7.4.3 导出仿射预测单元的子块偏差矩阵和控制参数

本条导出仿射样本改善信息asrInfo(ASRFlag,ASRHorFlag,ASRVerFlag,dMvA,dMvB,dMvC,dMvN)。

如果仿射控制点运动矢量组中有3个运动矢量,运动矢量组表示为mvsAffine(mv0,mv1,mv2);否则，如果仿射控制点运动矢量组中有2个运动矢量，运动矢量组表示为mvsAffine(mv0,mv1)。

令当前预测单元的宽度和高度分别是width和height，(x,y)是亮度子块左上角位置的坐标，子块的宽度和高度分别是subWidth和subHeight，当前预测单元亮度预测块的左上角样本所在的子块是A子块，右上角样本所在的子块是B子块，左下角样本所在的子块是C子块，其他样本所在的子块是其它子块，见图25。

第1步，计算dHorX、dVerX、dHorY和dVerY。

——dHorX和dHorY的计算如下：

```
dHorX = (mv1->x - mv0->x) << (7 - Log(width))
dHorY = (mv1->y - mv0->y) << (7 - Log(width))
```

——如果 mvsAffine 中有 3 个运动矢量，dVerX 和 dVerY 的计算如下：

```
dVerX = (mv2->x - mv0->x) << (7 - Log(height))
dVerY = (mv2->y - mv0->y) << (7 - Log(height))
```

——否则（mvsAffine 中有 2 个运动矢量），dVerX 和 dVerY 的计算如下：

```
dVerX = - dHorY
dVerY = dHorX
```

第2步，导出子块偏差矩阵dMvX（X为A、B、C或N，其中N为任意位置）。dMvX[i][j]是子块偏差矩阵的元素，其中(i,j)是每个亮度子块内部像素的坐标，i的取值范围为0～(subWidth-1)，j的取值范围为0～(subHeight-1)。

——如果 mvAffine 中有 2 个运动矢量且 dHorX 和 dHorY 均等于 0，令 ASRFlag 等于 0。

——否则,如果 mvAffine 中有 3 个运动矢量,且 dHorX、dHorY、dVerX 和 dVerY 均等于 0,令 ASRFlag 等于 0。

——否则，

1) 令 ASRFlag 等于 1。
2) 如果当前预测单元的预测参考模式是‘PRED_List01’或 AffineSubblockSizeFlag 等于 1，则 subWidth 和 subHeight 均等于 8。
3) 如果当前预测单元的预测参考模式是‘PRED_List0’或‘PRED_List1’，且 AffineSubblockSizeFlag 等于 0，则 subWidth 和 subHeight 均等于 4。
4) 如果当前子块是 A，则当前子块的偏差矩阵是 AffineDmvA：

```
AffineDmvA[i][j][0] = dHorX * 4 * i + dVerX * 4 * j
AffineDmvA[i][j][1] = dHorY * 4 * i + dVerY * 4 * j
```

5) 否则，如果当前子块是 B，则当前子块偏差矩阵是 AffineDmvB：

```
AffneDmvB[i][j][0] = dHorX * 4 * (i + 1 - subWidth) + dVerX * 4 * j
AffineDmvB[i][j][1] = dHorY * 4 * (i + 1 - subWidth) + dVerY * 4 * j
```

6) 否则，如果当前子块为 C 且运动矢量组中有 3 个运动矢量，则当前子块偏差矩阵是 AffineDmvC：

```
AffineDmvC[i][j][0] = dHorX * 4 * i + dVerX * 4 * (j + 1 - subHeight)
AffineDmvC[i][j][1] = dHorY * 4 * i + dVerY * 4 * (j + 1 - subHeight)
```

7) 否则，如果当前子块为 C 且运动矢量组中有 2 个运动矢量，则当前子块偏差矩阵是 AffineDmvC：

```
AffineDmvC[i][j][0] = dHorX * (2 + 4 * i - 2 * subWidth) + dVerX * (2 + 4 * j - 2 * subHeight)
AffineDmvC[i][j][1] = dHorY * (2 + 4 * i - 2 * subWidth) + dVerY * (2 + 4 * j - 2 * subHeight)
```

8) 否则，当前子块偏差矩阵是 AffineDmvN：

```
AffineDmvN[i][j][0] = dHorX * (2 + 4 * i - 2 * subWidth) + dVerX * (2 + 4 * j - 2 * subHeight)
```

```
AffineDmvN[i][j][1] = dHorY * (2 + 4 * i  -  2 * subWidth) + dVerY * (2 + 4 * j  -  2 * subHeight)
```

9) 计算 dMvHorAbsMax、dMvVerAbsMax 和 dMvThresh。

```
dMvHorAbsMax = Max(Max(Abs(dMvN[0][0][0]), Abs(dMvN[subWidth-1][0][0])),
Max(Abs(dMvN[0][subHeight-1][0]), Abs(dMvN[subWidth-1][subHeight-1][0])))
dMvVerAbsMax = Max(Max(Abs(dMvN[0][0][1]), Abs(dMvN[subWidth-1][0][1])),
Max(Abs(dMvN[0][subHeight-1][1]), Abs(dMvN[subWidth-1][subHeight-1][1])))
dMvThresh = ((1<<5) * 10)
```

10) 将 ASRHorFlag 和 ASRVerFlag 初始化为 1：

- 如果 dMvHorAbsMax 小于 dMvThresh 且 dMvVerAbsMax 小于 dMvThresh，则 ASRFlag、ASRHorFlag 和 ASRVerFlag 均等于 0；
- 否则，如果 dMvHorAbsMax 小于 dMvThresh，则 ASRHorFlag 为 0；
- 否则，如果 dMvVerAbsMax 小于 dMvThresh，则 ASRVerFlag 为 0。

11) 如果 ASRHorFlag 等于 1，导出亮度子块 X 的 dMvX[i][j]（X 为 A，B，C，N），i=0～(subWidth-1),j=0～(subHeight-1)：

```
dMvX[i][j][0] = Clip3(-31, 31, Rounding(dMvX[i][j][0], 8))
```

12) 如果 ASRVerFlag 等于 1，导出亮度子块 X 的 dMvX[i][j]（X 为 A，B，C，N），i=0～(subWidth-1),j=0～(subHeight-1)：

```
dMvX[i][j][1] = Clip3(-31, 31, Rounding(dMvX[i][j][1], 8))
```

9.5.7.5 获得相邻块空域运动信息存储单元

当前预测单元的亮度预测块 E 的相邻亮度预测块 X（X 为 A、B、C、D、F 或 G）的空域运动信息存储单元如下（见图 19）：

——相邻块 A 的空域运动信息存储单元是样本（x_0-1,y_0）所对应的空域运动信息存储单元；
——相邻块 B 的空域运动信息存储单元是样本（x_0,y_0-1）所对应的空域运动信息存储单元；
——相邻块 C 的空域运动信息存储单元是样本（x_1+1,y_0-1）所对应的空域运动信息存储单元；
——相邻块 D 的空域运动信息存储单元是样本（x_0-1,y_0-1）所对应的空域运动信息存储单元；
——相邻块 F 的空域运动信息存储单元是样本（x_0-1，y_1）所对应的空域运动信息存储单元；
——相邻块 G 的空域运动信息存储单元是样本（x_1,y_0-1）所对应的空域运动信息存储单元。

9.5.7.6 运动矢量预测

9.5.7.6.1 概述

如果AffineFlag和ExtendMvrFlag的值均为0，运动矢量预测过程见9.5.7.6.2；如果AffineFlag的值为0且ExtendMvrFlag的值为1，运动矢量预测过程见9.5.7.6.4。如果AffineFlag的值为1，运动矢量预测过程见9.5.7.6.3。

9.5.7.6.2 普通运动矢量预测

本条定义导出mvX，referenceIndexX，BlockDistanceX（X为A、B、C、D，A、B、C或D）的方法。X与当前预测单元的亮度预测块E的相邻关系见图19。

第1步，执行以下操作。

a) 如果相邻亮度预测块 X（X 为 A、B、C 或 D）和其所在的预测单元满足以下条件之一，则 mvX 等于零矢量，BlockDistanceX 等于 1，referenceIndexX 等于-1；否则 mvX 等于相邻亮度预测块 X 所在的预测单元中与当前待预测运动矢量对应的参考图像队列相同的运动矢量，BlockDistanceX 等于 mvX 对应的 BlockDistance，referenceIndexX 等于 mvX 对应的 referenceIndex。

1) 相邻亮度预测块 X“不可用”。
2) 相邻亮度预测块 X 采用帧内预测模式。
3) 相邻亮度预测块 X 所在的预测单元不存在与当前待预测运动矢量对应的参考图像队列相同的运动矢量。

b) 如果 referenceIndexC 等于-1，则 mvC 等于 mvD，BlockDistanceC 等于 BlockDistanceD，referenceIndexC 等于 referenceIndexD。

第2步，计算待预测运动矢量预测值MVEPred。

a) 如果 referenceIndexX（X 为 A、B 或 C）不等于-1，则根据 BlockDistanceX 和 BlockDistanceE 对 mvX(mvX_x,mvX_y) 进行缩放得到 MVX(MVX_x,MVX_y)；否则 MVX 为零矢量。其中 BlockDistanceE 是当前预测单元对应的 BlockDistance。

```
MVA->x = Clip3(-32768, 32767, Sign(mvA->x * BlockDistanceE * BlockDistanceA) * ((Abs(mvA->x * BlockDistanceE
* (16384 / BlockDistanceA)) + 8192) >> 14))
MVA->y = Clip3(-32768, 32767, Sign(mvA->y * BlockDistanceE * BlockDistanceA) * ((Abs(mvA->y * BlockDistanceE
* (16384 / BlockDistanceA)) + 8192) >> 14))

MVB->x = Clip3(-32768, 32767, Sign(mvB->x * BlockDistanceE * BlockDistanceB) * ((Abs(mvB->x * BlockDistanceE
* (16384 / BlockDistanceB)) + 8192) >> 14))
MVB->y = Clip3(-32768, 32767, Sign(mvB->y * BlockDistanceE * BlockDistanceB) * ((Abs(mvB->y * BlockDistanceE
* (16384 / BlockDistanceB)) + 8192) >> 14))

MVC->x = Clip3(-32768, 32767, Sign(mvC->x * BlockDistanceE * BlockDistanceC) * ((Abs(mvC->x * BlockDistanceE
* (16384 / BlockDistanceC)) + 8192) >> 14))
MVC->y = Clip3(-32768, 32767, Sign(mvC->y * BlockDistanceE * BlockDistanceC) * ((Abs(mvC->y * BlockDistanceE
* (16384 / BlockDistanceC)) + 8192) >> 14))
```

b) 如果referenceIndexA、referenceIndexB、referenceIndexC三者中只有一个referenceIndexX不为-1，那么MVEPred等于MVX（X为A、B或C），结束第2步；否则执行步骤c)。

c) 计算MVEPred的x和y运动矢量分量。

```
if (((MVA->x < 0) && (MVB->x > 0) && (MVC->x > 0)) || ((MVA->x > 0) && (MVB->x < 0) && (MVC->x < 0))) {
    MVEPred->x = (MVB->x + MVC->x) / 2
}
else if (((MVB->x < 0) && (MVA->x > 0) && (MVC->x > 0)) || ((MVB->x > 0) && (MVA->x < 0) && (MVC->x < 0)))
{
    MVEPred->x = (MVA->x + MVC->x) / 2
}
```

```
else if (((MVC->x < 0) && (MVA->x > 0) && (MVB->x > 0)) || ((MVC->x > 0) && (MVA->x < 0) && (MVB->x < 0)))
{
    MVEPred->x = (MVA->x + MVB->x) / 2
}
else if ((Abs(MVA->x - MVB->x) <= Abs(MVB->x - MVC->x)) && (Abs(MVA->x - MVB->x) <= Abs(MVC->x - MVA->x)))
{
    MVEPred->x = (MVA->x + MVB->x) / 2
}
else if ((Abs(MVB->x - MVC->x) <= Abs(MVA->x - MVB->x)) && (Abs(MVB->x - MVC->x) <= Abs(MVC->x - MVA->x)))
{
    MVEPred->x = (MVB->x + MVC->x) / 2
}
else {
    MVEPred->x = (MVA->x + MVC->x) / 2
}

if (((MVA->y < 0) && (MVB->y > 0) && (MVC->y > 0)) || ((MVA->y > 0) && (MVB->y < 0) && (MVC->y < 0))) {
    MVEPred->y = (MVB->y + MVC->y) / 2
}
else if (((MVB->y < 0) && (MVA->y > 0) && (MVC->y > 0)) || ((MVB->y > 0) && (MVA->y < 0) && (MVC->y < 0)))
{
    MVEPred->y = (MVA->y + MVC->y) / 2
}
else if (((MVC->y < 0) && (MVA->y > 0) && (MVB->y > 0)) || ((MVC->y > 0) && (MVA->y < 0) && (MVB->y < 0)))
{
    MVEPred->y = (MVA->y + MVB->y) / 2
}
else if ((Abs(MVA->y - MVB->y) <= Abs(MVB->y - MVC->y)) && (Abs(MVA->y - MVB->y) <= Abs(MVC->y - MVA->y)))
{
    MVEPred->y = (MVA->y + MVB->y) / 2
}
else if ((Abs(MVB->y - MVC->y) <= Abs(MVA->y - MVB->y)) && (Abs(MVB->y - MVC->y) <= Abs(MVC->y - MVA->y)))
{
    MVEPred->y = (MVB->y + MVC->y) / 2
}
else }
    MVEPred->y = (MVA->y + MVC->y) / 2
}
```

第3步，执行以下操作：

```
MvEPred->x = Clip3(-32768, 32767, Rounding( MvEPred->x, AmvrIndex ) << AmvrIndex)
MvEPred->y = Clip3(-32768, 32767, Rounding( MvEPred->y, AmvrIndex ) << AmvrIndex)
```

如果当前待预测运动矢量预测值MVEPred对应参考图像队列0中的图像，则MVEPredL0等于MVEPred；否则，如果当前待预测运动矢量预测值MVEPred对应参考图像队列1中的图像，则MVEPredL1等于MVEPred。

9.5.7.6.3 仿射运动矢量预测

本条定义导出MvX，referenceIndexX，BlockDistanceX（X为A、B、C、D、G，A、B、C、D或G）的方法。X与当前预测单元的亮度预测块E的相邻关系见图19。

第1步，执行以下操作：

a) 如果相邻亮度预测块 X（X 为 A、B、C、D 或 G）满足以下条件之一，则 mvX 等于零矢量，BlockDistanceX 等于 1，referenceIndexX 等于-1：
   1) 相邻亮度预测块 X“不可用”；
   2) 相邻亮度预测块 X 采用帧内预测模式；
   3) 相邻亮度预测块 X 不存在与当前待预测运动矢量对应的参考图像队列相同的运动矢量。
b) 否则，mvX 等于相邻亮度预测块 X 与当前待预测运动矢量对应的参考图像队列相同的运动矢量，BlockDistanceX 等于 X 所在的预测单元对应的 BlockDistance，referenceIndexX 等于 X 所在的预测单元对应的 referenceIndex。

第2步，计算当前待预测运动矢量预测值MvEPred。

a) 如果相邻预测块 A 所在的预测单元的 referenceIndexA 不等于-1，根据 BlockDistanceA 和 BlockDistanceE 对 mvA 进行缩放得到 MvEPred。

```
MvEPred->x = Clip3(-32768, 32767, Sign(mvA->x * BlockDistanceE * BlockDistanceA) * ((Abs(mvA->x * BlockDistanceE * (16384 / BlockDistanceA)) + 8192) >> 14))
MvEPred->y = Clip3(-32768, 32767, Sign(mvA->y * BlockDistanceE * BlockDistanceA) * ((Abs(mvA->y * BlockDistanceE * (16384 / BlockDistanceA)) + 8192) >> 14))
```

b) 否则，如果相邻预测块 B 所在的预测单元的 referenceIndexB 不等于-1，根据 BlockDistanceB 和 BlockDistanceE 对 mvB 进行缩放得到 MvEPred。

```
MvEPred->x = Clip3(-32768, 32767, Sign(mvB->x * BlockDistanceE * BlockDistanceB) * ((Abs(mvB->x * BlockDistanceE * (16384 / BlockDistanceB)) + 8192) >> 14))
MvEPred->y = Clip3(-32768, 32767, Sign(mvB->y * BlockDistanceE * BlockDistanceB) * ((Abs(mvB_y * BlockDistanceE * (16384 / BlockDistanceB)) + 8192) >> 14))
```

c) 否则，如果相邻预测块 D 所在的预测单元的 referenceIndexD 不等于-1，根据 BlockDistanceD 和 BlockDistanceE 对 mvD 进行缩放得到 MvEPred。

```
MvEPred->x = Clip3(-32768, 32767, Sign(mvD->x * BlockDistanceE * BlockDistanceD) * ((Abs(mvD->x * BlockDistanceE * (16384 / BlockDistanceD)) + 8192) >> 14))
MvEPred->y = Clip3(-32768, 32767, Sign(mvD->y * BlockDistanceE * BlockDistanceD) * ((Abs(mvD->y * BlockDistanceE * (16384 / BlockDistanceD)) + 8192) >> 14))
```

d) 否则，MvEPred 为零矢量。

第3步，计算当前待预测运动矢量预测值MvEPredAffine。

a) 如果相邻预测块 G 所在的预测单元的 referenceIndexG 不等于-1，根据 BlockDistanceG 和 BlockDistanceE 对 mvG 进行缩放得到 MvEPredAffine。

```
MvEPredAffine->x = Clip3(-32768, 32767, Sign(mvG->x * BlockDistanceE * BlockDistanceG) * ((Abs(mvG->x * BlockDistanceE * (16384 / BlockDistanceG)) + 8192) >> 14))
MvEPredAffine->y = Clip3(-32768, 32767, Sign(mvG->y * BlockDistanceE * BlockDistanceG) * ((Abs(mvG->y * BlockDistanceE * (16384 / BlockDistanceG)) + 8192) >> 14))
```

b) 否则，如果相邻预测块 C 所在的预测单元的 referenceIndexC 不等于-1，根据 BlockDistanceC 和 BlockDistanceE 对 mvC 进行缩放得到 MvEPredAffine。

```
MvEPredAffine->x = Clip3(-32768, 32767, Sign(mvC->x * BlockDistanceE * BlockDistanceC) * ((Abs(mvC->x * BlockDistanceE * (16384 / BlockDistanceC)) + 8192) >> 14))
MvEPredAffine->y = Clip3(-32768, 32767, Sign(mvC->y * BlockDistanceE * BlockDistanceC) * ((Abs(mvC->y * BlockDistanceE * (16384 / BlockDistanceC)) + 8192) >> 14))
```

c) 否则，MvEPredAffine 为零矢量。

第4步，执行以下操作：

```
MvEPred->x = MvEPred->x << 2
MvEPred->y = MvEPred->y << 2
MvEPredAffine->x = MvEPredAffine->x << 2
MvEPredAffine->y = MvEPredAffine->y << 2
mvrIndex = (AffineAmvrIndex == 0) ? 2 : ((AffineAmvrIndex == 1) ? 4 : 0)
MvEPred->x = Clip3(-131072, 131071, Rounding( MvEPred->x, mvrIndex ) << mvrIndex)
MvEPred->y = Clip3(-131072, 131071, Rounding( MvEPred->y, mvrIndex ) << mvrIndex)
MvEPredAffine->x = Clip3(-131072, 131071, Rounding( MvEPredAffine->x, mvrIndex ) << mvrIndex)
MvEPredAffine->y = Clip3(-131072, 131071, Rounding( MvEPredAffine->y, mvrIndex ) << mvrIndex)
```

如果当前待预测运动矢量预测值MVEPred、MVEPredAffine对应参考图像队列0中的图像，则MVEPredL0等于MVEPred，MVEPredL0Affine等于MVEPredAffine；否则，如果当前待预测运动矢量预测值MVEPred、MVEPredAffine对应参考图像队列1中的图像，则MVEPredL1等于MVEPred，MVEPredL1Affine等于MVEPredAffine。

9.5.7.6.4 扩展精度运动矢量预测

第1步，得到运动信息。

a) 令 hmvpIndex 的值等于 AmvrIndex；
b) 如果 CntHmvp 的值大于 hmvpIndex 的值，则取历史运动信息表中第(CntHmvp – hmvpIndex)个运动信息，记为 motionInfo；
c) 否则，如果 CntHmvp 的值大于 0 且小于或等于 hmvpIndex，则取历史运动信息表中第 CntHmvp 个运动信息，记为 motionInfo；
d) 否则，如果当前图像是 P 图像，令 motionInfo 的 L0 运动矢量为零矢量，L0 参考索引值为 0，预测参考模式为‘PRED_List0’；
e) 否则，如果当前图像是 B 图像，令 motionInfo 的 L0 运动矢量为零矢量，L1 运动矢量为零矢量，L0 参考索引值为 0，L1 参考索引值为 0，预测参考模式为‘PRED_List01’；
f) 具有运动信息 motionInfo 的预测单元的 BlockDistance 分别记为 BlockDistanceL0 和 BlockDistanceL1，当前预测单元所在的图像与参考图像队列 0 和参考图像队列 1 中的参考图

像对应的 BlockDistance 分别记为 BlockDistanceE0 和 BlockDistanceE1，motionInfo 的 L0 运动矢量和 L1 运动矢量分别记为 mvHmvp0 和 mvHmvp1，motionInfo 的 L0 参考索引和 L1 参考索引分别记为 hmvpRef0 和 hmvpRef1，motionInfo 的预测参考模式记为 hmvpInterMode。

第2步，计算当前待预测运动矢量预测值MvEPred。

a) 由预测参考模式 hmvpInterMode、当前预测单元的预测参考模式查表 94 得到运动矢量 mv、距离索引 D1 和距离索引 D2。如果当前待预测运动矢量预测值 MVEPred 对应参考图像队列 0 中的图像，则 MVEPredL0 等于 MVEPred；否则，如果当前待预测运动矢量预测值 MVEPred 对应参考图像队列 1 中的图像，则 MVEPredL1 等于 MVEPred。

```
MvEPred->x = Clip3(-32768, 32767, Sign(mv->x * D1 * D2) * ((Abs(mv->x * D1 * (16384 / D2)) + 8192) >> 14))
MvEPred->y = Clip3(-32768, 32767, Sign(mv->y * D1 * D2) * ((Abs(mv->y * D1 * (16384 / D2)) + 8192) >> 14))
```

b) 执行以下操作：

```
MVEPred->x = Clip3(-32768, 32767, Rounding( MVEPred->x, AmvrIndex ) << AmvrIndex)
MVEPred->y = Clip3(-32768, 32767, Rounding( MVEPred->y, AmvrIndex ) << AmvrIndex)
```

表94 运动矢量、距离索引和预测参考模式

预测参考模式	当前预测单元的预测参考模式	运动矢量	距离索引D1	距离索引D2	运动矢量预测值
PRED_List0	PRED_List0	mvHmvp0	BlockDistanceE0	BlockDistanceL0	mvePredL0
	PRED_List1	mvHmvp0	BlockDistanceE1	BlockDistanceL0	mvePredL1
	PRED_List01	mvHmvp0	BlockDistanceE0	BlockDistanceL0	mvePredL0
		mvHmvp0	BlockDistanceE1	BlockDistanceL0	mvePredL1
PRED_List1	PRED_List0	mvHmvp1	BlockDistanceE0	BlockDistanceL1	mvePredL0
	PRED_List1	mvHmvp1	BlockDistanceE1	BlockDistanceL1	mvePredL1
	PRED_List01	mvHmvp1	BlockDistanceE0	BlockDistanceL1	mvePredL0
		mvHmvp1	BlockDistanceE1	BlockDistanceL1	mvePredL1
PRED_List01	PRED_List0	mvHmvp0	BlockDistanceE0	BlockDistanceL0	mvePredL0
	PRED_List1	mvHmvp1	BlockDistanceE1	BlockDistanceL1	mvePredL1
	PRED_List01	mvHmvp0	BlockDistanceE0	BlockDistanceL0	mvePredL0
		mvHmvp1	BlockDistanceE1	BlockDistanceL1	mvePredL1

如果当前待预测运动矢量预测值MVEPred对应参考图像队列0中的图像，则MVEPredL0等于MVEPred；否则，如果当前待预测运动矢量预测值MVEPred对应参考图像队列1中的图像，则MVEPredL1等于MVEPred。

9.5.7.7 运动矢量解码

9.5.7.7.1 概述

如果AffineFlag的值为0，运动矢量解码过程见9.5.7.7.2；否则，运动矢量解码过程见9.5.7.7.3。

9.5.7.7.2 普通运动矢量解码

按以下方法解码得到运动矢量：

a) 如果当前预测单元的预测参考模式是‘Pred_List0’：

```
MvDiffXL0 = MvDiffXL0 << AmvrIndex
MvDiffYL0 = MvDiffYL0 << AmvrIndex
mvE->x = Clip3(-32768, 32767, MvDiffXL0 + MVEPredL0->x)
mvE->y = Clip3(-32768, 32767, MvDiffYL0 + MVEPredL0->y)
```

mvE 是当前预测单元的 L0 运动矢量，MvE 等于 mvE。

b) 如果当前预测单元的预测参考模式是‘Pred_List1’：

```
MvDiffXL1 = MvDiffXL1 << AmvrIndex
MvDiffYL1 = MvDiffYL1 << AmvrIndex
mvE->x = Clip3(-32768, 32767, MvDiffXL1 + MVEPredL1->x)
mvE->y = Clip3(-32768, 32767, MvDiffYL1 + MVEPredL1->y)
```

mvE 是当前预测单元的 L1 运动矢量，MvE 等于 mvE。

c) 如果当前预测单元的预测参考模式是‘Pred_List01’：

```
MvDiffXL0 = MvDiffXL0 << AmvrIndex
MvDiffYL0 = MvDiffYL0 << AmvrIndex
MvDiffXL1 = MvDiffXL1 << AmvrIndex
MvDiffYL1 = MvDiffYL1 << AmvrIndex
mvE0->x = Clip3(-32768, 32767, MvDiffXL0 + MVEPredL0->x)
mvE0->y = Clip3(-32768, 32767, MvDiffYL0 + MVEPredL0->y)
mvE1->x = Clip3(-32768, 32767, MvDiffXL1 + MVEPredL1->x)
mvE1->y = Clip3(-32768, 32767, MvDiffYL1 + MVEPredL1->y)
```

mvE0、mvE1 分别为当前预测单元的 L0 运动矢量和 L1 运动矢量，MvE0 等于 mvE0，MvE1 等于 mvE1。

9.5.7.7.3 仿射运动矢量解码

按以下方法解码得到仿射运动矢量。

a) 如果当前预测单元的预测参考模式是‘Pred_List0’：

```
mvrIndex = (AffineAmvrIndex == 0) ? 2 : ((AffineAmvrIndex == 1) ? 4 : 0)
MvDiffXL0 = MvDiffXL0 << mvrIndex
MvDiffYL0 = MvDiffYL0 << mvrIndex
MvDiffXL0Affine = MvDiffXL0Affine << mvrIndex
MvDiffYL0Affine = MvDiffYL0Affine << mvrIndex
mv1->x = Clip3(-131072, 131071, MvDiffXL0 + MVEPredL0->x)
mv1->y = Clip3(-131072, 131071, MvDiffYL0 + MVEPredL0->y)
mv2->x = Clip3(-131072, 131071, MvDiffXL0Affine + MVEPredL0Affine->x)
mv2->y = Clip3(-131072, 131071, MvDiffYL0Affine + MVEPredL0Affine->y)
```

mvsAffine(mv1,mv2)是当前预测单元的仿射控制点运动矢量组，当前预测单元是四参数仿射模式，根据 9.5.7.4.2 定义的过程由 mvsAffine 导出当前预测单元的 L0 运动矢量集合（记作 MvArrayL0）。MvArrayL0 由所有亮度预测子块的 L0 运动矢量组成。

b) 否则，如果当前预测单元的预测参考模式是‘Pred_List1’：

```
mvrIndex = (AffineAmvrIndex == 0) ? 2 : ((AffineAmvrIndex == 1) ? 4 : 0)
MvDiffXL1 = MvDiffXL1 << mvrIndex
MvDiffYL1 = MvDiffYL1 << mvrIndex
MvDiffXL1Affine = MvDiffXL1Affine << mvrIndex
MvDiffYL1Affine = MvDiffYL1Affine << mvrIndex
mv1->x = Clip3(-131072, 131071, MvDiffXL1 + MVEPredL1->x)
mv1->y = Clip3(-131072, 131071, MvDiffYL1 + MVEPredL1->y)
mv2->x = Clip3(-131072, 131071, MvDiffXL1Affine + MVEPredL1Affine->x)
mv2->y = Clip3(-131072, 131071, MvDiffYL1Affine + MVEPredL1Affine->y)
```

mvsAffine(mv1,mv2)是当前预测单元的仿射控制点运动矢量组,当前预测单元是四参数仿射模式，根据 9.5.7.4.2 定义的过程由 mvsAffine 导出当前预测单元中的 L1 运动矢量集合（记作 MvArrayL1）。MvArrayL1 由所有亮度预测子块的 L1 运动矢量组成。

c) 否则，如果当前预测单元的预测参考模式是‘Pred_List01’：

```
mvrIndex = (AffineAmvrIndex == 0) ? 2 : ((AffineAmvrIndex == 1) ? 4 : 0)
MvDiffXL0 = MvDiffXL0 << mvrIndex
MvDiffYL0 = MvDiffYL0 << mvrIndex
MvDiffXL1 = MvDiffXL1 << mvrIndex
MvDiffYL1 = MvDiffYL1 << mvrIndex
MvDiffXL0Affine = MvDiffXL0Affine << mvrIndex
MvDiffYL0Affine = MvDiffYL0Affine << mvrIndex
MvDiffXL1Affine = MvDiffXL1Affine << mvrIndex
MvDiffYL1Affine = MvDiffYL1Affine << mvrIndex
mv01->x = Clip3(-131072, 131071, MvDiffXL0 + MVEPredL0->x)
mv01->y = Clip3(-131072, 131071, MvDiffYL0 + MVEPredL0->y)
mv02->x = Clip3(-131072, 131071, MvDiffXL0Affine + MVEPredL0Affine->x)
mv02->y = Clip3(-131072, 131071, MvDiffYL0Affine + MVEPredL0Affine->y)
mv11->x = Clip3(-131072, 131071, MvDiffXL1 + MVEPredL1->x)
mv11->y = Clip3(-131072, 131071, MvDiffYL1 + MVEPredL1->y)
mv12->x = Clip3(-131072, 131071, MvDiffXL1Affine + MVEPredL1Affine->x)
mv12->y = Clip3(-131072, 131071, MvDiffYL1Affine + MVEPredL1Affine->y)
```

mvsAffine0(mv01,mv02)是当前预测单元的 L0 仿射控制点运动矢量组，mvsAffine1(mv11,mv12)是当前预测单元的 L1 仿射控制点运动矢量组，当前预测单元是四参数仿射模式，根据 9.5.7.4.2 定义的过程由 mvsAffine0 和 mvsAffine1 分别导出当前预测单元中的 L0 运动矢量集合（记作 MvArrayL0）和 L1 运动矢量集合（记作 MvArrayL1）。MvArrayL0 和 MvArrayL1 分别由所有亮度预测子块的 L0 运动矢量和 L1 运动矢量组成。

9.5.7.8 导出跳过模式和直接模式的运动信息

9.5.7.8.1 概述

如果当前预测单元所在的编码单元的编码单元子类型是‘P_Skip_Temporal’或‘P_Direct_Temporal’，按9.5.7.8.4导出运动信息motionInfo、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘P_Skip_SbTemporal’或‘P_Direct_SbTemporal’，按9.5.7.8.4导出运动信息集合MotionArray、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘B_Skip_Temporal’或‘B_Direct_Temporal’，按9.5.7.8.4导出运动信息motionInfo、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘B_Skip_SbTemporal’或‘B_Direct_SbTemporal’，按9.5.7.8.4导出运动信息集合MotionArray、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘P_Skip_Spatial_List0’或‘P_Direct_Spatial_List0’，按9.5.7.8.3导出运动信息motionInfo、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘B_Skip_Spatial_list0’‘B_Skip_Spatial_list1’‘B_Skip_Spatial_list01’‘B_Direct_Spatial_list0’‘B_Direct_Spatial_list1’或‘B_Direct_Spatial_list01’，按9.5.7.8.3导出运动信息motionInfo、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘P_Skip_Mvap’‘P_Direct_Mvap’、B_Skip_Mvap’或‘B_Direct_Mvap’，按9.5.7.8.8导出当前预测单元的运动信息阵列MotionArray、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘P_Skip_Hmvp’或‘P_Direct_Hmvp’，先分别按9.5.7.8.4和9.5.7.8.3导出motionInfo0和motionInfo1，再按9.5.7.8.6导出运动信息motionInfo、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘B_Skip_Hmvp’或‘B_Direct_Hmvp’，先分别按9.5.7.8.4和9.5.7.8.3导出motionInfo0、motionInfo1、motionInfo2和motionInfo3，再按9.5.7.8.6导出运动信息motionInfo、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘P_Skip_Umve’‘P_Direct_Umve’‘B_Skip_Umve’或‘B_Direct_Umve’，按9.5.7.8.5导出运动信息motionInfo、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘P_Skip_Etmvp’‘P_Direct_Etmvp’‘B_Skip_Etmvp’或‘B_Direct_Etmvp’，按9.5.7.8.10导出运动矢量阵列MotionArray、BgcFlag和BgcIndex。

如果当前预测单元所在的编码单元的编码单元子类型是‘P_Skip_Affine’‘P_Direct_Affine’‘B_Skip_Affine’或‘B_Direct_Affine’，按9.5.7.8.7导出仿射运动信息motionInfoAffine、BgcFlag和BgcIndex，以及仿射样本改善信息asrInfo0和asrInfo1。

如果当前预测单元所在的编码单元的编码单元子类型是‘B_Skip_Awp’‘B_Direct_Awp’‘P_Skip_Awp’或‘P_Direct_Awp’，按9.5.7.8.9导出运动信息motionInfoAwp0、motionInfoAwp1、BgcFlag和BgcIndex。

9.5.7.8.3、9.5.7.8.4和9.5.7.8.7中使用的运动信息存储单元的定义见9.5.7.10。9.5.7.8.5中判断运动信息是否相同的方法见9.5.7.8.3。

9.5.7.8.2 导出有效运动矢量角度预测模式数

令W和H是当前解码单元的宽度和高度，m和n的分别是W/4和H/4，(x,y)是当前解码单元左上角的像素坐标。A_0，A_1，…，A_{m-1}是当前块左下角位置的4×4块，A_m，A_{m+1}，…，A_{m+n-1}是当前块左侧位置所在的4×4块，A_{m+n}是当前块左上角位置所在的4×4块，A_{m+n+1}，A_{m+n+2}，…，A_{2m+n}是当前块上边位置所在的4×4块，A_{2m+n+1}，A_{2m+n+1}，…，A_{2m+2n}是当前块右上角位置所在的4×4块，见图26。如果i的取值范围是0～m+n，则A_i的坐标分别是(x-1,y+H+W-4*i-1)；如果i的取值范围是m+n+1～2m+2n，则A_i的坐标分别为(x+4*i-W-H-1,y-1)。

A_i位置空域存储单元的运动信息“可用”指A_i位置在图像内且A_i位置对应的编码单元与当前编码单元属于同一片且A_i位置对应的编码单元已解码且采用帧间预测模式；否则A_i位置空域存储单元的运动信息“不可用”。

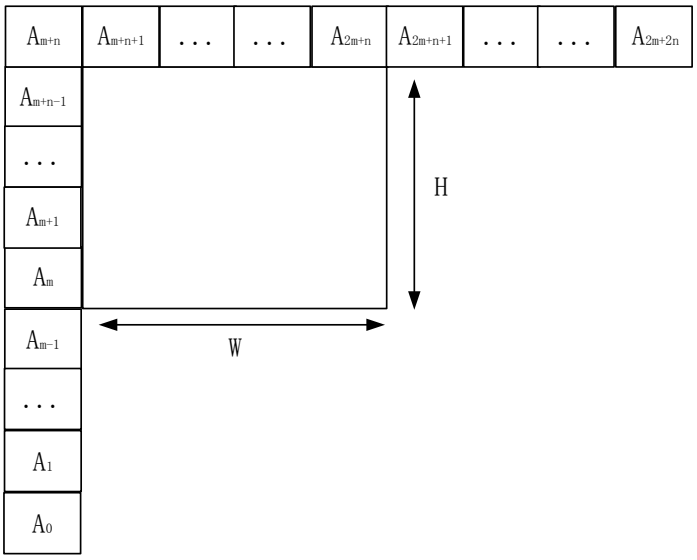


图26　运动矢量角度预测样本的位置

按以下步骤导出ValidMvapModeNum和ValidMvapModeFlag[s]（s=0～4）：

a) 将ValidMvapModeNum初始化为0，ValidMvapModeFlag[s]初始化为0（s=0～4）。

b) 如果满足以下条件之一,则ValidMvapModeNum等于0,ValidMvapModeFlag[s]等于0(s=0～4)：

1) EtmvpMvapEnableFlag 的值为 0；

2) W 小于 8，或 H 小于 8，或 W 和 H 均等于 8。

c) 否则，按下列步骤更新 ValidMvapModeNum 和 ValidMvapModeFlag[s]的值，其中判断运动信息是否相同的方法见 9.5.7.2：

1) 如果 $A_{m-1+H/8}$ 与 A_{m+n-1} 位置空域运动信息存储单元的运动信息均“可用”且运动信息不同，则 ValidMvapModeFlag[0]等于 1，并将 ValidMvapModeNum 的值加 1；

2) 如果 $A_{m+n+1+W/8}$ 与 A_{m+n+1} 位置空域运动信息存储单元的运动信息均“可用”且运动信息不同，则 ValidMvapModeFlag[1]等于 1，并将 ValidMvapModeNum 的值加 1；

3) 如果 A_{m+n-1} 与 A_{m+n} 位置空域运动信息存储单元的运动信息均“可用”且运动信息不同，或 A_{m+n} 和 A_{m+n+1} 位置空域运动信息存储单元的运动信息均“可用”且运动信息不同，则 ValidMvapModeFlag[2]等于 1，并将 ValidMvapModeNum 的值加 1；

4) 如果 $A_{W/8-1}$ 与 A_{m-1} 位置空域运动信息存储单元的运动信息均“可用”且运动信息不同，或 A_{m-1} 与 $A_{m-1+H/8}$ 位置空域运动信息存储单元的运动信息均“可用”且运动信息不同，则 ValidMvapModeFlag[3]等于 1，并将 ValidMvapModeNum 的值加 1；

5) 如果 $A_{m+n+1+W/8}$ 与 A_{2m+n+1} 位置空域运动信息存储单元的运动信息均“可用”且运动信息不同，或 A_{2m+n+1} 与 $A_{2m+n+1+H/8}$ 位置空域运动信息存储单元的运动信息均“可用”且运动信息不同，则 ValidMvapModeFlag[4]等于 1，并将 ValidMvapModeNum 的值加 1。

9.5.7.8.3　空域运动信息导出方法

9.5.7.8.3.1　概述

如果编码单元子类型是‘P_Skip_Spatial_list0’或‘P_Direct_Spatial_list0’，按9.5.7.8.3.2导出motionInfo(interPredRefMode, mvE0, mvE1, refIndexL0, refIndexL1)、BgcFlag和BgcIndex。

如果编码单元子类型是‘B_Skip_Spatial_list01’或‘B_Direct_Spatial_list01’,按9.5.7.8.3.3导出motionInfo(interPredRefMode, mvE0, mvE1, refIndexL0, refIndexL1)、BgcFlag和BgcIndex。

如果编码单元子类型是‘B_Skip_Spatial_list1’或‘B_Direct_Spatial_list1’，按9.5.7.8.3.4导出motionInfo(interPredRefMode, mvE0, mvE1, refIndexL0, refIndexL1)、BgcFlag和BgcIndex。

如果编码单元子类型是‘B_Skip_Spatial_list0’或‘B_Direct_Spatial_list0’，按9.5.7.8.3.5导出motionInfo(interPredRefMode, mvE0, mvE1, refIndexL0, refIndexL1)、BgcFlag和BgcIndex。

如果编码单子类型是‘P_Skip_Hmvp’或‘P_Direct_Hmvp’，按9.5.7.8.3.2导出motionInfo1(interPredRefMode, mvE0, mvE1, refIndexL0, refIndexL1)、BgcFlag和BgcIndex。

如果编码单子类型是‘B_Skip_Hmvp’或‘B_Direct_Hmvp’，则分别按9.5.7.8.3.3、9.5.7.8.3.4和9.5.7.8.3.5导出motionInfo1，motionInfo2，motionInfo3。

9.5.7.8.3.2 P图像空域运动信息导出方法

运动信息、BgcFlag和BgcIndex导出方法如下：

a) 如果当前预测单元的亮度预测块的相邻亮度预测块F、G、C、A、B、D六者中预测参考模式为‘PRED_List0’的预测块的个数大于或等于1，则按F、G、C、A、B、D的顺序依次扫描相邻亮度预测块得到第一个扫描到的预测参考模式为‘PRED_List0’的预测块，将该预测块的空域运动信息存储单元的L0运动矢量和L0参考索引分别作为当前预测单元的L0运动矢量mvE0和L0参考索引refIndexL0；
b) 否则，当前预测单元的L0运动矢量mvE0为零矢量，且当前预测单元的L0参考索引refIndexL0的值等于0；
c) interPredRefMode的值等于‘PRED_List0’，refIndexL1的值等于-1，mvE1是零矢量，BgcFlag和BgcIndex的值均置为0。

9.5.7.8.3.3 B图像空域运动信息导出方法1

运动信息、BgcFlag和BgcIndex导出方法1如下。

a) 如果当前预测单元的亮度预测块的相邻亮度预测块F、G、C、A、B、D六者中预测参考模式为‘PRED_List01’的预测块的个数大于或等于1，则按F、G、C、A、B、D的顺序依次扫描相邻亮度预测块得到第一个扫描到的预测参考模式为‘PRED_List01’的预测块，将该预测块的空域运动信息存储单元的L0运动矢量和L1运动矢量分别作为当前预测单元的L0运动矢量mvE0和L1运动矢量mvE1，并将该空域运动信息存储单元的L0参考索引和L1参考索引分别作为当前预测单元的L0参考索引refIndexL0和L1参考索引refIndexL1，同时将该空域运动信息存储单元的BgcFlag和BgcIndex记为BgcFlagX和BgcIndexX，当前预测单元的BgcFlag和BgcIndex的值分别为(BgcFlagX && !InterPcFlag)和(BgcIndexX && !InterPcFlag)。
b) 否则，如果当前预测单元的亮度预测块的相邻亮度预测块F、G、C、A、B、D六者中预测参考模式为‘PRED_List0’的预测块的个数大于或等于1，且预测参考模式为‘PRED_List1’的预测块的个数大于或等于1，则按F、G、C、A、B、D的顺序依次扫描相邻亮度预测块得到第一个扫描到的预测参考模式为‘PRED_List0’的预测块和第一个扫描到的预测参考模式为‘PRED_List1’的预测块，将预测参考模式为‘PRED_List0’的预测块的空域运动信息存储单元的L0运动矢量和L0运动索引作为当前预测单元的L0运动矢量mvE0和L0运动索引refIndexL0；将预测参考模式为‘PRED_List1’的预测块的空域运动信息存储单元的L1运动

矢量和 L1 参考索引作为当前预测单元的 L1 运动矢量 mvE1 和 L1 参考索引 refIndexL1，并将当前预测单元的 BgcFlag 和 BgcIndex 均置为 0。

c) 否则，当前预测单元的 L0 运动矢量 mvE0 和 L1 运动矢量 mvE1 均为零矢量，且当前预测单元的 L0 参考索引 refIndexL0 和 L1 参考索引 refIndexL1 的值均等于 0，并将当前预测单元的 BgcFlag 和 BgcIndex 均置为 0。

d) interPredRefMode 的值等于‘PRED_List01’。

9.5.7.8.3.4 B 图像空域运动信息导出方法 2

运动信息、BgcFlag和BgcIndex导出方法如下：

a) 如果当前预测单元的亮度预测块的相邻亮度预测块 F、G、C、A、B、D 六者中预测参考模式为‘PRED_List1’的预测块的个数大于或等于 1，则按 F、G、C、A、B、D 的顺序依次扫描相邻亮度预测块得到第一个扫描到的预测参考模式为‘PRED_List1’的预测块，将该预测块的空域运动信息存储单元的 L1 运动矢量和 L1 参考索引作为当前预测单元的 L1 运动矢量 mvE1 和 L1 参考索引 refIndexL1；

b) 否则，如果当前预测单元的亮度预测块的相邻亮度预测块 F、G、C、A、B、D 六者中预测参考模式为‘PRED_List01’的预测块的个数大于或等于 1，则按 D、B、A、C、G、F 的顺序依次扫描相邻亮度预测块得到第一个扫描到的预测参考模式为‘PRED_List01’的预测块，将该预测块的空域运动信息存储单元的 L1 运动矢量和 L1 参考索引作为当前预测单元的 L1 运动矢量 mvE1 和 L1 参考索引 refIndexL1；

c) 否则，当前预测单元的 L1 运动矢量 mvE1 为零矢量，且当前预测单元的 L1 参考索引 refIndexL1 的值等于 0；

d) interPredRefMode 的值等于‘PRED_List1’，refIndexL0 的值等于-1，mvE0 是零矢量，BgcFlag 和 BgcIndex 的值均置为 0。

9.5.7.8.3.5 B 图像空域运动信息导出方法 3

运动信息、BgcFlag和BgcIndex导出方法如下：

a) 如果当前预测单元的亮度预测块的相邻亮度预测块 F、G、C、A、B、D 六者中预测参考模式为‘PRED_List0’的预测块的个数大于或等于 1，则按 F、G、C、A、B、D 的顺序依次扫描相邻亮度预测块得到第一个扫描到的预测参考模式为‘PRED_List0’的预测块，将该预测块的空域运动信息存储单元的 L0 运动矢量和 L0 参考索引作为当前预测单元的 L0 运动矢量 mvE0 和 L0 参考索引 refIndexL0；

b) 否则，如果当前预测单元的亮度预测块的相邻亮度预测块 F、G、C、A、B、D 六者中预测参考模式为‘PRED_List01’的预测块的个数大于或等于 1，则按 D、B、A、C、G、F 的顺序依次扫描相邻亮度预测块得到第一个扫描到的预测参考模式为‘PRED_List01’的预测块，将该预测块的空域运动信息存储单元的 L0 运动矢量和 L0 参考索引作为当前预测单元的 L0 运动矢量 mvE0 和 L0 参考索引 refIndexL0；

c) 否则，当前预测单元的 L0 运动矢量 mvE0 为零矢量，且当前预测单元的 L0 参考索引 refIndexL0 的值等于 0；

d) interPredRefMode 的值等于‘PRED_List0’，refIndexL1 的值等于-1，mvE1 是零矢量，BgcFlag 和 BgcIndex 的值均置为 0。

9.5.7.8.4 时域运动信息导出方法

9.5.7.8.4.1 概述

如果当前编码单元子类型是‘P_Skip_Temporal’‘P_Direct_Temporal’‘B_Skip_Temporal’或‘B_Direct_Temporal’，按9.5.7.8.4.2导出BgcFlag、BgcIndex和motionInfo0，并分别对mvE、refIndexL0和interPredRefMode赋值；否则，按9.5.7.8.4.3导出BgcFlag、BgcIndex和motionInfo0，并分别对mvE、refIndexL0和interPredRefMode赋值。

如果编码单子类型是‘P_Skip_Hmvp’‘P_Direct_Hmvp’‘B_Skip_Hmvp’或‘B_Direct_Hmvp’且SbTmvpEnableFlag的值为0，按9.5.7.8.4.2导出BgcFlag、BgcIndex和motionInfo0。

9.5.7.8.4.2 时域运动信息导出方法1

如果当前图像是P图像，导出运动信息、BgcFlag和BgcIndex的方法如下。

a) 如果参考图像队列0中参考索引为0的图像中与当前预测单元的左上角亮度样本位置对应的亮度样本所在的时域运动信息存储单元存储的预测参考模式为‘PRED_List1’或‘PRED_I’，当前预测单元的 L0 运动矢量 MvE0 为零矢量，并令当前预测单元的 L0 参考索引值 RefIndexL0 等于 0。

b) 否则，

1) 当前预测单元的L0参考索引refIndex0等于0；

2) 将当前图像的距离索引记为DistanceIndexCur，当前预测单元的L0参考索引对应的参考图像的距离索引记为 DistanceIndexL0，计算 BlockDistanceL0=DistanceCur – DistanceIndexL0；

3) 在参考图像队列0中参考索引为0的图像中与当前预测单元的左上角亮度样本位置对应的亮度样本所在的时域运动信息存储单元的L0运动矢量记为mvRef(mvRef_x,mvRef_y)，该运动信息存储单元所在的图像的距离索引记为 DistanceIndexCol，该运动矢量指向的参考单元所在的图像的距离索引记为 DistanceIndexRef，计算 BlockDistanceRef = DistanceIndexCol – DistanceIndexRef；

4) 计算当前预测单元的L0运动矢量mvE0。

```
mvE0->x = Clip3(-32768, 32767, Sign(mvRef->x * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->x * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE0->y = Clip3(-32768, 32767, Sign(mvRef->y * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->y * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

c) interPredRefMode 的值等于‘PRED_List0’，BgcFlag 和 BgcIndex 的值均置为 0。

如果当前图像是B图像，导出运动信息、BgcFlag和BgcIndex的方法如下。

a) 如果参考图像队列1中参考索引值为0的图像中与当前预测单元的左上角亮度样本位置对应的亮度样本所在的时域运动信息存储单元存储的预测参考模式为‘PRED_List1’或‘PRED_I’，则当前预测单元的L0参考索引和L1参考索引均等于0。以当前预测单元所在编码单元的尺寸和位置作为当前预测单元的尺寸和位置，然后将按9.5.7.6.2得到的L0运动矢量预测值和L1运动矢量预测值分别作为当前预测单元的L0运动矢量MvE0和L1运动矢量MvE1，并令当前预测单元的L0参考索引RefIndexL0和L1参考索引RefIndexL1均等于0。

b) 否则，

1) 当前预测单元的L0参考索引refIndex0和L1参考索引refIndex1均等于0。

2) 将当前图像的距离索引记为DistanceIdexCur，当前预测单元的L0参考索引和L1参考索引对应的图像的距离索引分别记为 DistanceIndexL0 和 DistanceIndexL1，计算 BlockDistanceL0=DistanceIndexCur – DistanceIndexL0，BlockDistanceL1=DistanceIndexCur – DistanceIndexL1 。

3) 在参考图像队列1中参考索引为0的图像中与当前预测单元的左上角亮度样本位置对应的亮度样本所在的时域运动信息存储单元的L0运动矢量记为mvRef(mvRef_x,mvRef_y)，该运动信息存储单元所在的图像的距离索引记为 DistanceIndexCol，该运动矢量指向的参考单元所在的图像的距离索引记为 DistanceIndexRef，计算 BlockDistanceRef = DistanceIndexCol – DistanceIndexRef。

4) 当前预测单元的L0参考索引RefIndexL0等于0,计算当前预测单元的L0运动矢量mvE0。

```
mvE0->x = Clip3(-32768, 32767, Sign(mvRef->x * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->x * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE0->y = Clip3(-32768, 32767, Sign(mvRef->y * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->y * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

5) 当前预测单元的L1参考索引RefIndexL1等于0,计算当前预测单元的L1运动矢量mvE1。

```
mvE1->x = Clip3(-32768, 32767, Sign(mvRef->x * BlockDistanceL1 * BlockDistanceRef) * (((Abs(mvRef->x * BlockDistanceL1 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE1->y = Clip3(-32768, 32767, Sign(mvRef->y * BlockDistanceL1 * BlockDistanceRef) * (((Abs(mvRef->y * BlockDistanceL1 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

c) interPredRefMode 等于‘PRED_List01’，BgcFlag 和 BgcIndex 的值均置为0。

9.5.7.8.4.3 时域运动信息导出方法2

第1种情况，当前图像是P图像，导出运动信息、BgcFlag和BgcIndex的方法如下。

a) 以当前编码单元作为预测单元按9.5.7.8.4.2导出的预测单元的L0参考索引RefIndexL0，记为refIndexT0，以及预测单元的L0运动矢量mvE0，记为mvT0。

b) 将当前编码单元划分为四个子块,每个子块的长和宽为当前编码单元的1/2,位置关系见图27。图27中数字是每个子块的index。对于index为0的子块，（col_x,col_y）是其左上角亮度样本的位置坐标；对于index为1的子块，(col_x,col_y)是其右上角亮度样本的位置坐标；对于index为2的子块，（col_x,col_y）是其左下角亮度样本的位置坐标；对于index为3的子块，（col_x,col_y）是其右下角亮度样本的位置坐标。对每个子块执行以下步骤，导出每个子块的运动信息 MotionInfo[i](interPredRefMode,mvE0,refIndex0,mvE1,refIndex1)，MotionArray 由 MotionInfo0[i]构成。

1) 如果参考图像队列0中参考索引为0的图像中位置坐标为(col_x,col_y)的亮度样本所在的时域运动信息存储单元存储的预测参考模式为‘PRED_I’，当前预测单元的L0运动矢量MvE0为mvT0，并令当前预测单元的L0参考索引值RefIndexL0等于refIndexT0。

2) 否则，

- 当前预测单元的L0参考索引refIndex0等于0。
- 将当前图像的距离索引记为DistanceIndexCur，当前预测单元L0参考索引对应的参考图像的距离索引记为 DistanceIndexL0，计算 BlockDistanceL0=DistanceCur – DistanceIndexL0。
- 如果参考图像队列0中参考索引为0的图像中位置坐标为(col_x,col_y)的亮度样本所在的时域运动信息存储单元预测参考模式为“PRED_List0”或“PRED_List01”，则将该时域运动信息存储单元L0运动矢量记为mvRef(mvRef_x,mvRef_y)，该运动信息存储单元所在的图像的距离索引记为 DistanceIndexCol，该运动矢量指向的参考

单元所在的图像的距离索引记为 DistanceIndexRef，计算 BlockDistanceRef = DistanceIndexCol - DistanceIndexRef。

- 否则，将该时域运动信息存储单元 L1 运动矢量记为 mvRef(mvRef_x,mvRef_y)，该运动信息存储单元所在的图像的距离索引记为 DistanceIndexCol，该运动矢量指向的参考单元所在的图像的距离索引记为 DistanceIndexRef，计算 BlockDistanceRef = DistanceIndexCol – DistanceIndexRef。
- 计算当前预测单元的 L0 运动矢量 mvE0。

```
mvE0->x = Clip3(-32768, 32767, Sign(mvRef->x * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->x * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE0->y = Clip3(-32768, 32767, Sign(mvRef->y * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->y * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

c) interPredRefMode 的值等于‘PRED_List0’，BgcFlag 和 BgcIndex 的值均置为 0。

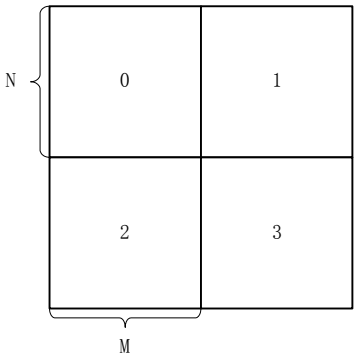


图27 预测单元划分为子块

第2种情况，当前图像是B图像，导出运动信息、BgcFlag和BgcIndex的方法如下。

a) 以当前编码单元作为预测单元按 9.5.7.8.4.2 导出预测单元的 L0 参考索引 RefIndexL0 和 L1 参考索引 RefIndexL1,分别记为 refIndexT0 和 refIndexT1,以及预测单元的 L0 运动矢量 mvE0 和 L1 运动矢量 mvE1，分别记为 mvT0 和 mvT1。

b) 将当前编码单元划分为四个子块,每个子块的长和宽为当前编码单元的 1/2,位置关系见图 27。图 27 中数字是每个子块的 index。对于 index 为 0 的子块，（col_x,col_y）是其左上角亮度样本的位置坐标；对于 index 为 1 的子块，(col_x,col_y)是其右上角亮度样本的位置坐标；对于 index 为 2 的子块，（col_x,col_y）是其左下角亮度样本的位置坐标；对于 index 为 3 的子块，（col_x,col_y）是其右下角亮度样本的位置坐标。对每个子块执行以下步骤，导出每个子块的运动信息 MotionInfo[i](interPredRefMode,mvE0,refIndex0,mvE1, refIndex1)，MotionArray 由 MotionInfo0[i]构成：

1) 如果参考图像队列 1 中参考索引值为 0 的图像中位置坐标为（col_x,col_y）的亮度样本所在的时域运动信息存储单元存储的预测参考模式为‘PRED_I’，则当前预测单元的 L0 运动矢量 MvE0 为 mvT0，L1 运动矢量 MvE1 为 mvT1，并令当前预测单元的 L0 参考索引值 RefIndexL0 等于 refIndexT0，L1 参考索引值 RefIndexL1 等于 refIndexT1。

2) 否则，

- 当前预测单元的 L0 参考索引 refIndex0 和 L1 参考索引 refIndex1 均等于 0。

- 将当前图像的距离索引记为DistanceIdexCur，当前预测单元的L0参考索引和L1参考索引对应的图像的距离索引分别记为DistanceIndexL0和DistanceIndexL1，计算BlockDistanceL0=DistanceIndexCur − DistanceIndexL0，BlockDistanceL1=DistanceIndexCur − DistanceIndexL1。
- 如果参考图像队列1中参考索引为0的图像中位置坐标为（col_x,col_y）的亮度样本所在的时域运动信息存储单元的预测参考模式为“PRED_List0”或“PRED_List01”，则将该时域运动信息存储单元L0运动矢量记为mvRef(mvRef_x,mvRef_y)，该运动信息存储单元所在的图像的距离索引记为 DistanceIndexCol，该运动矢量指向的参考单元所在的图像的距离索引记为 DistanceIndexRef，计算 BlockDistanceRef = DistanceIndexCol - DistanceIndexRef。
- 否则，将该时域运动信息存储单元L1运动矢量记为mvRef(mvRef_x,mvRef_y)，该运动信息存储单元所在的图像的距离索引记为 DistanceIndexCol，该运动矢量指向的参考单元所在的图像的距离索引记为DistanceIndexRef，计算BlockDistanceRef = DistanceIndexCol − DistanceIndexRef。
- 计算当前预测单元的L0运动矢量mvE0。

```
mvE0->x = Clip3(-32768, 32767, Sign(mvRef->x * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->x * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE0->y = Clip3(-32768, 32767, Sign(mvRef->y * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->y * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

- 计算当前预测单元的L1运动矢量mvE1。

```
mvE1->x = Clip3(-32768, 32767, Sign(mvRef->x * BlockDistanceL1 * BlockDistanceRef) * (((Abs(mvRef->x * BlockDistanceL1 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE1->y = Clip3(-32768, 32767, Sign(mvRef->y * BlockDistanceL1 * BlockDistanceRef) * (((Abs(mvRef->y * BlockDistanceL1 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

c) interPredRefMode的值等于‘PRED_List01’，BgcFlag和BgcIndex的值均置为0。

9.5.7.8.5 高级运动矢量表达运动信息导出方法

本条定义导出运动信息motionInfo、BgcFlag和BgcIndex的方法。

第1步,F、G、C、A和D是当前预测单元E的相邻预测块(见图19),确定mvBaseE0、mvBaseE1、RefIndexL0、RefIndexL1、InterPredRefMode、BgcFlag和BgcIndex。

a) 如果F存在且采用帧间预测模式，则F“可用”；否则，F“不可用”。
b) 如果G存在且采用帧间预测模式且G和F的运动信息不相同，则G“可用”；否则，G“不可用”。
c) 如果C存在且采用帧间预测模式且C和G的运动信息不相同，则C“可用”；否则，C“不可用”。
d) 如果A存在且采用帧间预测模式且A和F的运动信息不相同，则A“可用”；否则，A“不可用”。
e) 如果D存在且采用帧间预测模式且D和A的运动信息不相同且D和G的运动信息也不相同,则D“可用”；否则，D“不可用”。
f) 如果F、G、C、A、D均“不可用”且UmveMvIndex等于1，则MvBaseE0是零矢量，RefIndexL0等于0，InterPredRefMode是‘PRED_List0’，BgcFlag和BgcIndex的值均置为0。

g) 否则，如果F、G、C、A、D中有UmveMvIndex+1个“可用”，记第UmveMvIndex+1个可用的预测块所在的预测单元为X，mvUmveBaseInterPredRefMode为X的预测参考模式：
   1) 如果X的预测参考模式为‘PRED_List0’，则mvUmveBaseMv0为X的L0运动矢量，mvUmveBaseRef0为X的L0参考索引，mvBaseE0等于mvUmveBaseMv0，mvBaseE1不存在，RefIndexL0等于mvUmveBaseRef0,InterPredRefMode等于mvUmveBaseInterPredRefMode，BgcFlag和BgcIndex的值均置为0；
   2) 否则，如果X的预测参考模式为‘PRED_List1’，则mvUmveBaseMv1为X的L1运动矢量，mvUmveBaseRef1为X的L1参考索引，mvBaseE1等于mvUmveBaseMv1，mvBaseE0不存在，RefIndexL1等于mvUmveBaseRef1,InterPredRefMode等于mvUmveBaseInterPredRefMode，BgcFlag和BgcIndex的值均置为0；
   3) 否则，如果X的预测参考模式为‘PRED_List01’，则mvUmveBaseMv0为X的L0运动矢量，mvUmveBaseRef0为X的L0参考索引,mvUmveBaseMv1为X的L1运动矢量,mvUmveBaseRef1为X的L1参考索引，mvBaseE0等于mvUmveBaseMv0，mvBaseE1等于mvUmveBaseMv1，RefIndexL0等于mvUmveBaseRef0，RefIndexL1等于mvUmveBaseRef1，InterPredRefMode等于mvUmveBaseInterPredRefMode，BgcFlag和BgcIndex的值分别等于（BgcFlagX && !InterPcFlag）和（BgcIndexX && !InterPcFlag），其中BgcFlagX和BgcIndexX分别是X的BgcFlag和BgcIndex。

h) 否则，判断当前预测单元所在的编码单元的编码单元类型：
   1) 如果编码单元子类型是‘P_Skip_Umve’或‘P_Direct_Umve’，按9.5.7.8.4.2定义的方法导出运动矢量mvUmveBaseMv0、参考帧索引mvUmveBaseRef0，并令mvUmveBaseInterPredRefMode等于‘PRED_List0’，mvBaseE0等于mvUmveBaseMv0，mvBaseE1不存在，RefIndexL0等于mvUmveBaseRef0，RefIndexL1等于-1，InterPredRefMode等于mvUmveBaseInterPredRefMode,BgcFlag和BgcIndex的值置为0；
   2) 如果编码单元子类型是‘B_Skip_Umve’或‘B_Direct_Umve’，按9.5.7.8.4.2定义的方法导出L0参考索引mvUmveBaseRef0、L0运动矢量UmveMv0、L1参考索引mvUmveBaseRef1和L1运动矢量UmveMv1,并令mvUmveBaseInterPredRefMode等于‘PRED_List01’,mvBaseE0等于UmveMv0，mvBaseE1等于UmveMv1，RefIndexL0等于mvUmveBaseRef0，RefIndexL1等于mvUmveBaseRef1，InterPredRefMode等于mvUmveBaseInterPredRefMode，BgcFlag和BgcIndex的值置为0。

第2步，确定运动矢量偏移量mvOffset0和mvOffset1。

a) 根据UmveStepIndex和PictureUmveStepSetIndex的值查表95得到运动矢量偏移量UmveOffset。
b) 如果InterPredRefMode的值为‘PRED_List0’,则mvOffset0的值等于UmveOffset,mvOffset1不存在。
c) 否则，如果InterPredRefMode的值为‘PRED_List1’，则mvOffset1的值等于UmveOffset，mvOffset0不存在。
d) 否则，如果InterPredRefMode的值为‘PRED_List01’，则参考图像队列0中参考索引为RefIndexL0的图像和参考图像队列1中参考索引为RefIndexL1的图像的距离索引分别等于DistanceIndexL0和DistanceIndexL1，当前预测单元所在的图像的距离索引记为DistanceIndexCur,当前预测单元的参考单元所在的参考图像队列0和参考图像队列1中图像的BlockDistance分别记为BlockDistanceL0和BlockDistanceL1。
   1) 按9.5.7.1得到BlockDistanceL0和BlockDistanceL1。
   2) 计算运动矢量偏移量mvOffset0和mvOffset1：

```
if (Abs(BlockDistanceL1) >= Abs(BlockDistanceL0)) {
    mvOffset0 = Clip3(-32768, 32767, Sign(BlockDistanceL1 * BlockDistanceL0) * ( (Abs((16384 / BlockDistanceL1) * BlockDistanceL0) * UmveOffset + 8192) >> 14) )
    mvOffset1 = Clip3(-32768, 32767, (16384 * UmveOffset + 8192) >> 14)
}
else {
    mvOffset0 = Clip3(-32768, 32767, (16384 * UmveOffset + 8192) >> 14)
    mvOffset1 = Clip3(-32768, 32767, Sign(BlockDistanceL1 * BlockDistanceL0) * ( (Abs((16384 / BlockDistanceL0) * BlockDistanceL1) * UmveOffset + 8192) >> 14) )
}
```

表95 UmveOffset 与 UmveStepIndex 的对应关系

UmveStepIndex	UmveOffset	
	PictureUmveStepSetIndex的值为0	PictureUmveStepSetIndex的值为1
0	1	1
1	2	2
2	4	4
3	8	8
4	16	16
5	—	32
6	—	64
7	—	128

第3步，导出运动矢量：

a) 如果 InterPredRefMode 的值为‘PRED_List0’，按照下述过程导出当前预测单元的 mvE0，令 MvE 等于 mvE0。

```
if ((UmveDirIndex == 0) || (UmveDirIndex == 1)) {
    mvE0->x = mvBaseE0->x + (-1)^UmveDirIndex * mvOffset0
    mvE0->y = mvBaseE0->y
}
else if ((UmveDirIndex == 2) || (UmveDirIndex == 3)) {
    mvE0->x = mvBaseE0->x
    mvE0->y = mvBaseE0->y + (-1)^UmveDirIndex * mvOffset0
}
mvE0->x = Clip3(-32768, 32767, mvE0->x)
mvE0->y = Clip3(-32768, 32767, mvE0->y)
```

b) 如果 InterPredRefMode 的值为‘PRED_List1’，按照下述过程导出当前预测单元的 mvE1，令 MvE 等于 mvE1。

```
if ((UmveDirIndex == 0) || (UmveDirIndex == 1)) {
    mvE1->x = mvBaseE1->x + (-1)^UmveDirIndex * mvOffset1
```

```
    mvE1->y = mvBaseE1->y
}
else if ((UmveDirIndex == 2) || (UmveDirIndex == 3)) {
    mvE1->x = mvBaseE1->x
    mvE1->y = mvBaseE1->y + (-1)^UmveDirIndex * mvOffset1
}
mvE1->x = Clip3(-32768, 32767, mvE1->x)
mvE1->y = Clip3(-32768, 32767, mvE1->y)
```

c) 如果 InterPredRefMode 的值为‘PRED_List01’，按照下述过程导出当前预测单元的 mvE0 和 mvE1，令 MvE0 等于 mvE0，MvE1 等于 mvE1。

```
if ((UmveDirIndex == 0) || (UmveDirIndex == 1)) {
    mvE0->x = mvBaseE0->x + (-1)^UmveDirIndex * mvOffset0
    mvE0->y = mvBaseE0->y
    mvE1->x = mvBaseE1->x + (-1)^UmveDirIndex * mvOffset1
    mvE1->y = mvBaseE1->y
}
else if ((UmveDirIndex == 2) || (UmveDirIndex == 3)) {
    mvE0->x = mvBaseE0->x
    mvE0->y = mvBaseE0->y + (-1)^UmveDirIndex * mvOffset0
    mvE1->x = mvBaseE1->x
    mvE1->y = mvBaseE1->y + (-1)^UmveDirIndex * mvOffset1
}
mvE0->x = Clip3(-32768, 32767, mvE0->x)
mvE0->y = Clip3(-32768, 32767, mvE0->y)
mvE1->x = Clip3(-32768, 32767, mvE1->x)
mvE1->y = Clip3(-32768, 32767, mvE1->y)
```

RefIndexL0 等于 mvUmveBaseRef0， RefIndexL1 等于 mvUmveBaseRef1， InterPredRefMode 等于 mvUmveBaseInterPredRefMode。

9.5.7.8.6 历史运动信息候选项导出方法

根据历史运动信息表HmvpCandList和motionInfoX，按以下方法导出运动信息motionInfo、BgcFlag和BgcIndex。

a) 将可选的历史运动信息候选项数量 NumAllowedCand 初始化为 Min(CntHmvp,NumOfHmvpCand)，历史运动信息表索引 hmvpIndex 初始化为 1。如果当前图像是 P 图像，则 candIndex 等于 1+ValidMvapModeNum；如果当前图像是 B 图像，则 candIndex 等于 3+ValidMvapModeNum。

b) 如果 numAllowedCand 等于 0，结束导出过程。如果当前图像是 P 图像，则 motionInfo 等于 motionInfo1；如果当前图像是 B 图像，则 motionInfo 等于 motionInfo3，BgcFlag 和 BgcIndex 的值均置为 0。

c) 否则，执行以下操作，直到 candIndex 等于 CuSubTypeIndex，或 hmvpIndex 大于 NumAllowedCand。

1) 令 tmpHmvp 等于 HmvpCandList[CntHmvp-hmvpIndex]。

2) 按9.5.7.2依次判断tmpHmvp中的运动信息与motionInfoX是否相同。如果均不相同，则candIndex加1，hmvpIndex加1；否则，hmvpIndex加1。

d) 如果candIndex等于CuSubTypeIndex，则motionInfo等于tmpHMVP中的运动信息，BgcFlag和BgcIndex分别等于tmpHMVP中的BgcFlag和BgcIndex;如果candIndex小于CuSubTypeIndex，则motionInfo等于HmvpCandList[cntHmvp-1]中的运动信息，BgcFlag和BgcIndex分别等于HmvpCandList[cntHmvp-1]中的BgcFlag和BgcIndex。

e) 如果InterPcFlag的值为1，则将BgcFlag和BgcIndex均置为0。

motionInfoX中X的值如下。

——当SbTmvpEnableFlag的值为0时，如果当前图像是P图像，X为0或1；如果当前图像是B图像，X为0、1、2或3。

——当SbTmvpEnableFlag的值为1时，如果当前图像是P图像，X为1；如果当前图像是B图像，X为1、2或3。

9.5.7.8.7 仿射运动信息导出方法

9.5.7.8.7.1 概述

本条用于导出仿射运动信息、BgcFlag和BgcIndex，以及仿射样本改善信息asrInfo0和asrInfo1。

首先，根据9.5.7.8.7.3导出仿射控制点运动矢量组、预测参考模式、参考索引、BgcFlag和BgcIndex。

如果AffineUmveFlag的值等于1，则根据9.5.7.8.7.4对仿射控制点运动矢量组进行调整。

如果当前预测单元的预测参考模式是‘PRED_List0’或‘PRED_List01’，按9.5.7.4根据仿射控制点运动矢量组得到MvArrayL0和asrInfo0；

如果当前预测单元的预测参考模式是‘PRED_List1’或‘PRED_List01’，按9.5.7.4根据仿射控制点运动矢量组得到MvArrayL1和asrInfo1。

9.5.7.8.7.2 仿射运动矢量继承

令(xN,yN)是相邻块X所在预测单元的左上角样本在图像中的坐标，(xC,yC)是当前预测单元的左上角样本在图像中的坐标，widthX和heightX是相邻块X所在的预测单元的宽度和高度，width和height是当前预测单元的宽度和高度。

仿射运动继承方法如下。

a) 如果yC % MaxQtSize等于0且yN + heightX等于yC，则将isBoundy初始化为1，否则，将isBoundy初始化为0。

b) 如果isBoundy等于1，则mvN0是X所在的预测单元左下角样本所在的4×4子块的运动矢量，mvN1是X所在的预测单元右下角样本所在的4×4子块的运动矢量，widthOffset等于xC-xN，heightOffset等于0；否则，mvN0是X所在的预测单元左上角样本所在的4×4子块的运动矢量，mvN1是X所在的预测单元右上角样本所在的4×4子块的运动矢量，mvN2是X所在的预测单元左下角样本所在的4×4子块的运动矢量，widthOffset等于xC-xN，heightOffset等于yC-yN。

c) 执行以下操作。

1) 计算变量mvScaleHor、mvScaleVer、dHorX、dVerX、dHorY和dVerY。

```
mvScaleHor = mvN0_x << 7
mvScaleVer = mvN0_y<< 7
dHorX = (mvN1->x - mvN0->x) << (7 - Log(widthX))
dHorY = (mvN1->y - mvN0->y) << (7 - Log(widthX))
```

2) 如果 X 所在的预测单元是六参数仿射预测单元且 isBoundy 为 0，则：

```
dVerX = (mvN2->x - mvN0->x) << (7 - Log(heightX))
dVerY = (mvN2->y - mvN0->y) << (7 - Log(heightX))
```

3) 否则：

```
dVerX = -dHorY
dVerY = dHorX
```

d) 如果 X 所在的预测单元是六参数仿射预测单元且 isBoundy 为 0，则当前预测单元是六参数仿射预测单元，仿射控制点运动矢量组是 mvsAffine(mv0,mv1,mv2)，计算运动矢量 mv0、mv1 和 mv2。

```
mv0->x = Clip3(-131072, 131071, Rounding(mvScaleHor + dHorX * widthOffset + dVerX * heightOffset, 7) << 2)
mv0->y = Clip3(-131072, 131071, Rounding(mvScaleVer + dHorY * widthOffset + dVerY * heightOffset, 7) << 2)
mv1->x = Clip3(-131072, 131071, Rounding(mvScaleHor + dHorX * (widthOffset + width) + dVerX * heightOffset,
7) << 2)
mv1->y = Clip3(-131072, 131071, Rounding(mvScaleVer + dHorY * (widthOffset + width) + dVerY * heightOffset,
7) << 2)
mv2->x= Clip3(-131072, 131071, Rounding(mvScaleHor + dHorX * widthOffset + dVerX * (heightOffset + height),
  7) << 2)
mv2->y= Clip3(-131072, 131071, Rounding(mvScaleVer + dHorY * widthOffset + dVerY * (heightOffset + height),
  7) << 2)
```

e) 否则，当前预测单元是四参数仿射预测单元，仿射控制点运动矢量组是 mvsAffine(mv0,mv1)，计算运动矢量 mv0 和 mv1。

```
mv0->x = Clip3(-131072, 131071, Rounding(mvScaleHor + dHorX * widthOffset + dVerX * heightOffset, 7) << 2)
mv0->y = Clip3(-131072, 131071, Rounding(mvScaleVer + dHorY * widthOffset + dVerY * heightOffset, 7) << 2)
mv1->x = Clip3(-131072, 131071, Rounding(mvScaleHor + dHorX * (widthOffset + width) + dVerX * heightOffset,
7) << 2)
mv1->y = Clip3(-131072, 131071, Rounding(mvScaleVer + dHorY * (widthOffset + width) + dVerY * heightOffset,
7) << 2)
```

9.5.7.8.7.3 导出仿射运动信息

本条定义了导出当前预测单元E的仿射运动信息的方法。

第1步，确定E的相邻亮度样本H。

a) 如果当前图像是 P 图像，令 H 为参考图像队列 0 中参考索引为 0 的图像中与当前预测单元 E 的右下角亮度样本位置对应的亮度样本（见图 28）。如果亮度样本 H 所在的时域运动信息存储单元存储的预测参考模式为‘PRED_I’或者‘PRED_List1’，则 H“不存在”；否则，H“存在”。

b) 如果当前图像是 B 图像，令 H 为参考图像队列 1 中参考索引为 0 的图像中与当前预测单元 E 的右下角亮度样本位置对应的亮度样本（见图 29）。如果亮度样本 H 所在的时域运动信息存

储单元存储的预测参考模式为‘PRED_I’或者‘PRED_List1’，则H“不存在”；否则，H“存在”。

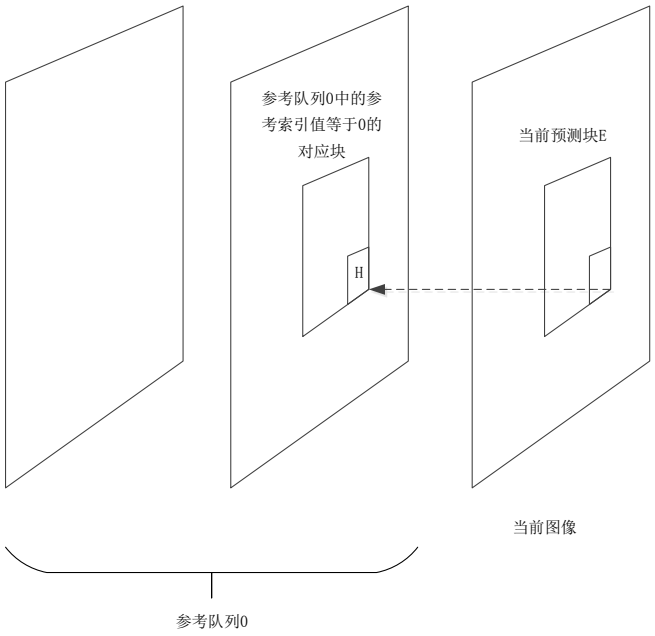


图28　预测单元E和相邻亮度样本H的空间位置关系（当前图像是P图像）

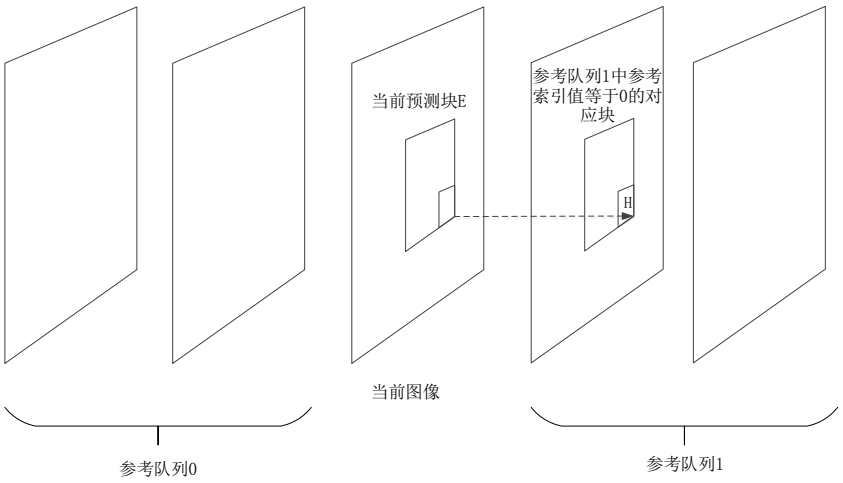


图29　预测单元E和相邻亮度样本H的空间位置关系（当前图像是B图像）

第2步，确定E的预测参考模式、参考索引和仿射控制点运动矢量组。

a) 当AffineCandIndex的值为0或1时，如果F、G、C、A或D所在的预测单元中至少有AffineCandIndex+1个“可用”且不同的仿射预测单元，记第AffineCandIndex+1个可用的仿射预测单元包括的预测块为X；否则，执行第3步。

b) E的预测参考模式和参考索引分别等于X的空域运动信息存储单元的预测参考模式和参考索引。

c) 如果X所在的预测单元的预测参考模式为‘Pred_List0’，由X的空域运动信息存储单元的运动信息按9.5.7.8.7.2定义的方法得到E的L0仿射控制点运动矢量组mvs_L0。E的BgcFlag和BgcIndex均为0。

d) 如果X所在的预测单元的预测参考模式为‘Pred_List1’，由X的空域运动信息存储单元的运动信息按9.5.7.8.7.2定义的方法得到E的L1仿射控制点运动矢量组mvs_L1。E的BgcFlag和BgcIndex均为0。

e) 否则，E的预测参考模式为‘Pred_List01’，由X的空域运动信息存储单元的运动信息按9.5.7.8.7.2定义的方法得到E的L0仿射控制点运动矢量组mvs_L0和仿射控制点运动矢量组mvs_L1。E的BgcFlag和BgcIndex分别等于X的BgcFlag和BgcIndex。

第3步，如果F、G、C、A和D所在的预测单元都不是仿射预测单元，AffineStartOffset等于0；如果F、G、C、A或D所在的预测单元中存在仿射预测单元，且均是同一个仿射预测单元，则AffineStartOffset等于1；否则，AffineStartOffset等于2。

第4步，确定相邻块。

a) 如果A、B和D中至少有一个“可用”且为帧间预测模式，则按A、B、D的顺序扫描得到第一个“可用”的相邻块X0；否则，X0“不存在”。

b) 如果G和C中至少有一个“可用”且为帧间预测模式，则按G、C的顺序扫描得到第一个“可用”且为帧间预测模式的相邻块X1；否则，X1“不存在”。

c) 如果F“可用”且为帧间预测模式，则记相邻块F为X2；否则，X2“不存在”。

d) 如果H“存在”，则记H为X3；否则，X3“不存在”。

第5步，导出运动矢量。

a) 如果X3“存在”且当前图像是B图像，则按9.5.7.8.4.2定义的方法导出L0运动矢量MVX3_L0和L1运动矢量MVX3_L1，X3的L0和L1的参考索引均等于0；否则，如果X3“存在”且当前图像是P图像，则按9.5.7.8.4.2定义的方法导出L0运动矢量MVX3_L0，X3的L0参考索引等于0，L1参考索引等于-1；否则MVX3_L0和MVX3_L1“不存在”。

b) 如果X(X为X0、X1或X2)“存在”且X的L0参考索引不等于-1，则X的L0运动矢量为MVX_L0；否则MVX_L0“不存在”。

c) 如果X(X为X0、X1或X2)“存在”且X的L1参考索引不等于-1，则X的L1运动矢量为MVX_L1；否则MVX_L1“不存在”。

d) 如果MVX0_X和MVX2_X均“存在”（X为L0或L1）且X的参考索引相同，则：

```
MVX2_1_X->x = Rounding(((MVX2_X->y - MVX0_X->y)<<7) * M / N + (MVX0_X->x << 7), 7)
MVX2_1_X->y = Rounding(-((MVX2_X->x - MVX0_X->x) << 7) * M / N + (MVX0_X->y << 7), 7)
```

其中M和N分别是当前编码单元的宽度和高度。

e) 否则，MVX2_1_X（X为L0或L1）“不存在”。

第6步，导出仿射控制点运动矢量组。

a) 分别判断以下X（X为L0或L1）运动矢量组合是否“可用”（如果组合中所有元素包含的运动矢量都“存在”且对应的参考索引相同，则组合“可用”；否则，组合“不可用”）。

```
mvs1_X: { (MVX0_X->x, MVX0_X->y),
          (MVX1_X->x, MVX1_X->y),
          (MVX2_X->x, MVX2_X->y) }
```

```
mvs2_X: { (MVX0_X->x, MVX0_X->y),
          (MVX1_X->x, MVX1_X->y),
          (MVX0_X->x + MVX3_X->x - MVX1_X->x, MVX0_X->y + MVX3_X->y - MVX1_X->y) }
mvs3_X: { (MVX0_X->x, MVX0_X->y),
          (MVX0_X->x + MVX3_X->x - MVX2_X->x, MVX0_X->y + MVX3_X->y - MVX2_X->y),
          (MVX2_X->x, MVX2_X->y) }
mvs4_X: { (MVX1_X->x + MVX2_X->x - MVX3_X->x, MVX1_X->y + MVX2_X->y - MVX3_X->y),
          (MVX1_X->x, MVX1_X->y),
          (MVX2_X->x, MVX2_X->y) }
mvs5_X: { (MVX0_X->x, MVX0_X->y),
          (MVX1_X->x, MVX1_X->y) }
mvs6_X: { (MVX0_X->x, MVX0_X->y),
          (MVX2_1_X->x, MVX2_1_X->y) }
```

b) 判断运动矢量候选项 mvsX（X 为 1、2、3、4、5、6）是否“可用”：
   1) 如果 mvsX_L0“可用”且 mvsX_L1“不可用”，则 mvsX 可用且 mvsX 只包含 mvsX_L0；
   2) 否则，如果 mvsX_L1“可用”且 mvsX_L0“不可用”，则 mvsX“可用”且 mvsX 只包含 mvsX_L1；
   3) 否则，如果 mvsX_L0 和 mvsX_L1 均“可用”，则 mvsX“可用”且 mvsX 包含 mvsX_L0 和 mvsX_L1；
   4) 否则，mvsX“不可用”。

c) 如果“可用”候选项的数量大于或等于 AffineCandIndex-AffineStartOffset+1，则按 mvs1、mvs2、mvs3、mvs4、mvs5、mvs6 的顺序取第 AffineCandIndex-AffineStartOffset+1 个“可用”候选项并记为 mvs。将 mvs 中所有运动矢量的 x 分量和 y 分量均左移两位，再将 mvs 中所有运动矢量的 x 分量和 y 分量的值均限制在[-131072,131071]区间内；否则，候选项“不存在”。

d) 如果 mvs 中只包含 mvs_L0，当前预测单元的预测参考模式是‘Pred_List0’，当前预测单元的 L0 参考索引 RefIndex0 等于 mvs_L0 对应的参考索引 refIndex0。如果 mvs_L0 中有三个运动矢量，当前预测单元是六参数仿射预测单元；否则当前预测单元是四参数仿射预测单元。当前预测单元的 L0 仿射控制点运动矢量组是 mvs_L0。当前预测单元 E 的 BgcFlag 和 BgcIndex 均为 0。

e) 如果 mvs 中只包含 mvs_L1，当前预测单元的预测参考模式是‘Pred_List1’，当前预测单元的 L1 参考索引 RefIndex1 等于 mvs_L1 对应的参考索引 refIndex1。如果 mvs_L1 中有三个运动矢量，当前预测单元是六参数仿射预测单元；否则当前预测单元是四参数仿射预测单元。当前预测单元的 L1 仿射控制点运动矢量组是 mvs_L1。当前预测单元 E 的 BgcFlag 和 BgcIndex 均为 0。

f) 如果 mvs 中包含 mvs_L0 和 mvs_L1，当前预测单元的预测参考模式是‘Pred_List01’，当前预测单元的 L0 参考索引 RefIndex0 和 L1 参考索引 RefIndex1 分别等于 mvs_L0 和 mvs_L1 对应的参考索引 refIndex0 和 refIndex1。如果 mvs_L0 或 mvs_L1 中有三个运动矢量，当前预测单元是六参数仿射预测单元；否则当前预测单元是四参数仿射预测单元。当前预测单元的 L0 仿射控制点运动矢量组是 mvs_L0，L1 仿射控制点运动矢量组是 mvs_L1。如果所取候选项为 mvs4，当前预测单元 E 的 BgcFlag 和 BgcIndex 分别等于 X1 的 BgcFlag 和 BgcIndex；否则，当前预测单元 E 的 BgcFlag 和 BgcIndex 分别等于 X0 的 BgcFlag 和 BgcIndex。

g) 如果 mvs“不存在”，当前预测单元的预测参考模式是‘Pred_List0’，当前预测单元的 L0 参考索引 RefIndexL0 为 0，当前预测单元是四参数仿射预测单元，当前预测单元的仿射控制

点运动矢量组 mvsAffine 中有且仅有两个零矢量。当前预测单元 E 的 BgcFlag 和 BgcIndex 均为 0。

9.5.7.8.7.4 调整仿射控制点运动矢量组

本条定义了调整仿射控制点运动矢量组。

令mvsAffineLX为预测单元的LX仿射控制点运动矢量组，mv0和mv1分别是mvsAffineLX中的第一个和第二个运动矢量。其中，如果预测参考模式为‘PRED_List0’，则X为0；如果预测参考模式为‘PRED_List1’，则X为1；如果预测参考模式为‘PRED_List01’，则X为0或1。

对mvY（Y等于0或1）按以下方法调整。

a) 当 Y 等于 0 时，affineUmveStepIndex 的值等于 AffineUmveStepIndex0，affineUmveDirIndex 的值等于 AffineUmveDirIndex0；否则，affineUmveStepIndex 的值等于 AffineUmveStepIndex1，affineUmveDirIndex 的值等于 AffineUmveDirIndex1。

b) 根据 affineUmveStepIndex 查表 96 得到 affineMvOffset。

c) 对 mvY 进行以下调整：

```
if ((affineUmveDirIndex == 0) || (affineUmveDirIndex == 1)) {
    mvY->x = mvY->x + (-1)^affineUmveDirIndex * affineMvOffset
    mvY->y = mvY->y
}
else if ((affineUmveDirIndex == 2) || (affineUmveDirIndex == 3)) {
    mvY->x = mvY->x
    mvY->y = mvY->y+ (-1)^affineUmveDirIndex * affineMvOffset
}
mvY->x = Clip3(-131072, 131071, mvY->x)
mvY->y = Clip3(-131072, 131071, mvY->y)
```

表96 affineMvOffset 与 affineUmveStepIndex 的对应关系

affineUmveStepIndex	affineMvOffset
0	1
1	2
2	4
3	8
4	16

9.5.7.8.8 运动矢量角度预测运动信息导出方法

本条导出运动矢量角度预测单元子块的运动信息阵列MotionArray、BgcFlag和BgcIndex。

第1步，得到运动矢量角度预测模式号。

a) 令 curMvapMode 是当前编码单元的运动矢量角度预测模式号，将 curMvapMode 初始化为 0。

b) 如果当前图像是 P 图像，candIndex 为 1；如果当前图像是 B 图像，candIndex 为 3。执行以下操作，直到 candIndex 等于 CuSubTypeIndex，结束导出过程：

1) 如果 ValidMvapModeFlag[curMvapMode]等于 1，则 candIndex 加 1，curMvapMode 加 1；

2) 否则，curMvapMode 加 1。

第2步，填充参考运动信息列表neighborMotions[i]（i=0～2m+2n）。

令W和H是当前解码单元的宽度和高度，m和n的分别是W/4和H/4，(x,y)是当前解码单元左上角的像素坐标。A_0，A_1，…，A_{m-1}是当前块左下角位置的4×4块；A_m，A_{m+1}，…，A_{m+n-1}是当前块左侧位置所在的4×4块；A_{m+n}是当前块左上角位置所在的4×4块；A_{m+n+1}，A_{m+n+2}，…，A_{2m+n}是当前块上边位置所在的4×4块；A_{2m+n+1}，A_{2m+n+1}，…，A_{2m+2n}是当前块右上角位置所在的4×4块，见图26。如果i的取值范围是0～m+n，则A_i的坐标分别是(x-1,y+H+W-4*i-1)；如果i的取值范围是m+n+1～2m+2n，则A_i的坐标分别为(x+4*i-W-H-1,y-1)。

如果以下条件均满足，则A_i位置空域存储单元的运动信息“可用”，否则“不可用”：

a) A_i位置在图像内；

b) A_i位置对应的编码单元与当前编码单元属于同一片；

c) A_i位置对应的编码单元已解码且采用帧间预测模式。

令neighborMotions[i]是参考运动信息列表中的第i个运动信息（interPredRefMode, mvE0, mvE1, refIndexL0, refIndexL1），其中i的取值范围是0～2m+2n。

如果A_i位置空域存储单元的运动信息“可用”，则A_i位置存储单元的运动信息记为$motionInfoA_i$（i=0～2m+2n）。

a) 填充 neighborMotions[0]：

1) 如果 A_0位置空域存储单元的运动信息“不可用”，则 neighborMotions[0]的预测参考模式 interPredRefMode 设为“PRED_List0”，neighborMotions[0]的 mvE0 设为零矢量，neighborMotions[0]的 refIndexL0 设为 0；

2) 否则，neighborMotions[0]设为 $motionInfoA_0$。

b) 按以下步骤从小到大依次遍历 A_i，填充 neighborMotions[i]：

1) 如果 A_i位置空域存储单元的运动信息“可用”，则 neighborMotions[i]为 $motionInfoA_i$；

2) 否则，neighborMotions[i]等于 neighborMotions[i-1]。

第3步，导出运动信息阵列MotionArray，BgcFlag和BgcIndex。

根据 curMvapMode 和 neighborMotions 确定当前编码单元内部子块的运动信息阵列 MotionArray[i][j]（interPredRefMode, MvE0,MvE1,RefIndexL0,RefIndexL1），其中i=0～（W >> 3）-1,j=0～（H >> 3）-1,(i,j)为当前编码单元内的8×8子块的索引，i为子块的水平索引值，j为子块的垂直索引值。对每个子块进行遍历，导出运动矢量角度预测的子块运动信息阵列MotionArray，并将BgcFlag和BgcIndex置零。

```
m=W >> 2
n= H>> 2

if (curMvapMode == 0) {
    MotionArray[i][j] = neighborMotions[m + n - 1 - (j << 1)]
}
else if (curMvapMode == 1) {
    MotionArray[i][j] = neighborMotions[m + n + 1 + (i << 1)]
}
else if (curMvapMode == 2) {
    MotionArray[i][j] = neighborMotions[m + n - (j << 1) + (i << 1)]
}
else if (curMvapMode == 3) {
```

```
    MotionArray[i][j] = neighborMotions[m + n - (j << 1) - (i << 1) - 3]
}
else if (curMvapMode == 4) {
    MotionArray[i][j] = neighborMotions[m + n + (j << 1) + (i << 1) + 3]
}
```

9.5.7.8.9 角度加权预测模式运动信息导出方法

9.5.7.8.9.1 概述

导出角度加权预测模式第一运动信息MotionInfoAwp0(mvAwp0L0, mvAwp0L1, RefIndexAwp0L0, RefIndexAwp0L1, interAwpPredRefMode0)、第二运动信息MotionInfoAwp1(mvAwp1L0, mvAwp1L1, RefIndexAwp1L0, RefIndexAwp1L1, interAwpPredRefMode1)、BgcFlag和BgcIndex。

9.5.7.8.9.2 角度加权预测模式运动信息查重

如果运动信息A和运动信息B满足下面一个或多个条件，则运动信息A和运动信息B不同；否则，运动信息A和运动信息B相同：

a) 按9.5.7.3判定由运动信息A中的预测参考模式、参考索引所确定的参考图像和由运动信息B中的预测参考模式、参考索引所确定的参考图像不同；
b) 运动信息A中的预测参考模式对应的运动矢量和运动信息B中的预测参考模式对应的运动矢量不同。

注：预测参考模式‘PRED_List0’对应的运动矢量为L0运动矢量，预测参考模式‘PRED_List1’对应的运动矢量为L1运动矢量。

9.5.7.8.9.3 导出角度加权预测的时域运动信息

令当前预测单元的左上角亮度样本位置为(x,y)，亮度预测块的宽度和高度分别为W和H。

第1步，计算(tx,ty)：

a) 初始化tx=x+W，ty=y+H；
b) 如果tx大于当前预测单元所在的LCU的最右边的亮度样本位置的横坐标，或tx大于当前预测单元所在的图像或patch的最右边的亮度样本位置的横坐标，则tx=x+W－1；
c) 如果ty大于当前预测单元所在的LCU的最下边的亮度样本位置的纵坐标，或ty大于当前预测单元所在的图像或patch的最下边的亮度样本位置的纵坐标，则ty=y+H－1。

第2步，导出运动信息。

a) 如果当前图像为P图像：
   1) 如果参考图像队列0中参考索引值为0的图像中坐标位置是(tx,ty)的亮度样本所在的时域运动信息存储单元存储的预测参考模式为“PRED_I”或“PRED_L1”，当前预测单元的L0运动矢量MvE0为零矢量，并令当前预测单元的L0参考索引值RefIndexL0等于0，结束第2步；否则，继续执行以下操作。
   2) 当前预测单元的L0参考索引等于0，当前预测单元的L0参考索引对应的图像的距离索引记为DistanceIndexL0，当前预测单元的L0参考索引对应的图像的BlockDistance记为BlockDistanceL0。
   3) 计算BlockDistanceRef。

```
BlockDistanceRef = DistanceIndexCol - DistanceIndexRef
```

4) 计算当前预测单元的L0运动矢量mvE0。

```
mvE0_x = Clip3(-32768, 32767, Sign(mvRef_x * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef_x * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE0_y = Clip3(-32768, 32767, Sign(mvRef_y * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef_y * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

5) interPredRefMode的值等于‘PRED_List0’。

b) 否则，当前图像为B图像时：

1) 如果参考图像队列1中参考索引值为0的图像中坐标位置是(tx,ty)的亮度样本所在的时域运动信息存储单元存储的预测参考模式为‘PRED_I’，以当前预测单元所在编码单元的尺寸和位置作为当前预测单元的尺寸和位置，然后将按9.5.7.6.2得到的L0运动矢量预测值和L1运动矢量预测值分别作为当前预测单元的L0运动矢量MvE0和L1运动矢量MvE1，并令当前预测单元的L0参考索引RefIndexL0和L1参考索引RefIndexL1均等于0，结束第2步；否则，继续执行以下操作。

2) 当前预测单元的L0参考索引和L1参考索引均等于0，当前预测单元的L0参考索引和L1参考索引对应的图像的距离索引分别记为DistanceIndexL0和DistanceIndexL1，当前预测单元的L0参考索引和L1参考索引对应的图像的BlockDistance分别记为BlockDistanceL0和BlockDistanceL1。

3) 如果参考图像队列1中参考索引为0的图像中坐标位置是(tx,ty)的亮度样本所在的时域运动信息存储单元的预测参考模式为‘PRED_L0’或‘PRED_List01’，则将该时域运动信息存储单元L0运动矢量记为mvRef，该时域运动信息存储单元所在的图像的距离索引记为DistanceIndexCol，该运动矢量指向的参考单元所在的图像的距离索引记为DistanceIndexRef;否则,如果参考图像队列1中参考索引为0的图像中坐标位置是(tx,ty)的亮度样本所在的时域运动信息存储单元的预测参考模式为‘PRED_List1’，将该时域运动信息存储单元L1运动矢量记为mvRef，该时域运动信息存储单元所在的图像的距离索引记为DistanceIndexCol，该运动矢量指向的参考单元所在的图像的距离索引记为DistanceIndexRef。

4) 计算BlockDistanceRef。

```
BlockDistanceRef = DistanceIndexCol - DistanceIndexRef
```

5) 令当前预测单元的L0参考索引RefIndexL0等于0,计算当前预测单元的L0运动矢量mvE0。

```
mvE0_x = Clip3(-32768, 32767, Sign(mvRef_x * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef_x * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE0_y = Clip3(-32768, 32767, Sign(mvRef_y * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef_y * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

6) 令当前预测单元的L1参考索引RefIndexL1等于0,计算当前预测单元的L1运动矢量mvE1。

```
mvE1_x = Clip3(-32768, 32767, Sign(mvRef_x * BlockDistanceL1 * BlockDistanceRef) * (((Abs(mvRef_x * BlockDistanceL1 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE1_y = Clip3(-32768, 32767, Sign(mvRef_y * BlockDistanceL1 * BlockDistanceRef) * (((Abs(mvRef_y * BlockDistanceL1 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

7） interPredRefMode 的值等于‘PRED_List01’。

9.5.7.8.9.4 角度加权预测模式运动信息

本条定义了导出mvAwp0L0，mvAwp0L1，RefIndexAwp0L0，RefIndexAwp0L1，mvAwp1L0，mvAwp1L1，RefIndexAwp1L0、RefIndexAwp1L1、interAwpPredRefMode0和interAwpPredRefMode1的方法。

第1步，F、G、C、A、B和D是当前预测单元E的相邻预测块（见图19），确定F、G、C、A、B和D的“可用”性。

a） 如果 F 存在且采用帧间预测模式，则 F“可用”；否则，F“不可用”。

b） 如果 G 存在且采用帧间预测模式，则 G“可用”；否则，G“不可用”。

c） 如果 C 存在且采用帧间预测模式，则 C“可用”；否则，C“不可用”。

d） 如果 A 存在且采用帧间预测模式，则 A“可用”；否则，A“不可用”。

e） 如果 B 存在且采用帧间预测模式，则 B“可用”；否则，B“不可用”。

f） 如果 D 存在且采用帧间预测模式，则 D“可用”；否则，D“不可用”。

第2步，初始化index等于0，按9.5.7.8.9.3导出运动信息T，然后将T和可用块的运动信息（依照F、G、C、A、B、D的顺序）逐个按照如下步骤，放入运动信息候选列表AwpCandArray。

a） 如果当前运动信息的预测参考模式是‘PRED_List0’或‘PRED_List1’，将其与 AwpCandArray 中已经存在的运动信息按 9.5.7.8.9.2 进行查重操作，若均不重复则放入 AwpCandArray[index]，index 加 1。如果 index 等于 5，则转到第 4 步。

b） 否则，根据当前可用运动信息的遍历顺序编号（从 0 开始），按照如下规则修改运动信息的预测参考模式，并将修改后的运动信息与 AwpCandArray 中已经存在的运动信息按 9.5.7.8.9.2 进行查重操作，若均不重复则放入 AwpCandArray[index]，index 加 1。如果 index 等于 5，则转到第 4 步。

1） 若遍历顺序编号为偶数，将当前运动信息的预测参考模式修改为‘PRED_List0’。

2） 若遍历顺序编号为奇数，将当前运动信息的预测参考模式修改为‘PRED_List1’。

第3步，如果index小于5，按以下方法生成运动信息motionInfo[i]（i=0～3）。motionInfo[i]的预测参考模式等于AwpCandArray[0]的预测参考模式,AwpCandArray[0]的运动信息中与预测参考模式对应的运动矢量记为mv，motionInfo[i]与预测参考模式对应的运动矢量记为mvInfo[i]。将motionInfo[i]按i从0到3的顺序依次放入AwpCandArray[index]，index加1。如果index等于5，则转到第4步。

```
amv[0] = mv->x
amv[1] = mv->y
axis = i % 2
sf = i > 1 ? -1 : 1
tmp = Abs(amv[axis])
sign = amv[axis] < 0 ? 1 : 0
amv[axis] = tmp < 8 ? 8 : (tmp <= 64 ? ((tmp * (4 + sf) + 2) >> 2) : (tmp <= 128 ? ((tmp * (8 + sf) + 4) >>
3) : ((tmp * (32 + sf) + 16) >> 5)))
amv[axis] = amv[axis] * (-1)^sign
if (i > 1 && tmp < 8) {
    amv[axis] = -amv[axis]
}
amv[axis] = Clip3(-32768, 32767, amv[axis])
mvInfo[i]->x = amv[0]
mvInfo[i]->y = amv[1]
```

第4步，将运动信息AwpCandArray[AwpCandIndex1]赋值给motionInfoAwp0。

第5步，将运动信息AwpCandArray[AwpCandIndex0]赋值给motionInfoAwp1。

第6步，将BgcFlag和BgcIndex的值均置为0。如果AwpMvrCandFlag0或者AwpMvrCandFlag1等于1，对mvAwp0L0，mvAwp0L1以及mvAwp1L0，mvAwp1L1进行修正。

a) 根据 AwpMvrCandStep0 和 AwpMvrCandStep1 的值查表 97 分别得到运动矢量偏移量 AwpMvrOffset0 和 AwpMvrOffset1。

b) 如果 AwpMvrCandFlagX 的值等于 1，修正 mvAwpXL0，mvAwpXL1，X 为 0 或 1。

1) 如果 RefIndexAwpXL0 的值大于等于 0，则 mvAwpXL0 (mvAwpXL0_x, mvAwpXL0_y)修正为：

```
if ((AwpMvrCandDirX == 0) || (AwpMvrCandDirX == 1)) {
    mvAwpXL0->x = mvAwpXL0->x + (-1) ^AwpMvrCandDirX * AwpMvrOffsetX
    mvAwpXL0->y = mvAwpXL0->y
}
else if ((AwpMvrCandDirX == 2) || (AwpMvrCandDirX == 3)) {
    mvAwpXL0->x = mvAwpXL0->x
    mvAwpXL0->y = mvAwpXL0->y + (-1) ^AwpMvrCandDirX * AwpMvrOffsetX
}
mvAwpXL0->x = Clip3(-32768, 32767, mvAwpXL0->x)
mvAwpXL0->y = Clip3(-32768, 32767, mvAwpXL0->y)
```

2) 否则(RefIndexAwpXL1 的值大于等于 0)，mvAwpXL1 (mvAwpXL1_x, mvAwpXL1_y)修正为：

```
if ((AwpMvrCandDirX == 0) || (AwpMvrCandDirX == 1)) {
    mvAwpXL1->x = mvAwpXL1->x + (-1) ^AwpMvrCandDirX * AwpMvrOffsetX
    mvAwpXL1->y = mvAwpXL1->y
}
else if ((AwpMvrCandDir0 == 2) || (AwpMvrCandDir0 == 3)) {
    mvAwpXL1->x = mvAwpXL1->x
    mvAwpXL1->y = mvAwpXL1->y + (-1) ^AwpMvrCandDirX * AwpMvrOffsetX
}
mvAwpXL1->x = Clip3(-32768, 32767, mvAwpXL1->x)
mvAwpXL1->y = Clip3(-32768, 32767, mvAwpXL1->y)
```

其中，在 AwpMvrCandDirX、AwpMvrOffset 和 AwpMvrCanDirX 中，X 的值为 0 或 1。

表97 AwpMvrCandStep0、AwpMvrCandStep1 与 AwpMvrOffset0、AwpMvrOffset1 的对应关系

AwpMvrCandStep0，AwpMvrCandStep1	AwpMvrOffset0，AwpMvrOffset1
0	1
1	2
2	4
3	8
4	16

9.5.7.8.10 增强时域运动信息导出方法

9.5.7.8.10.1 概述

按9.5.7.8.10.2导出增强时域运动信息阵列MotionArray、BgcFlag和BgcIndex。

9.5.7.8.10.2 增强时域运动信息导出方法

本条导出增强时域运动矢量预测模式的运动信息阵列MotionArray、BgcFlag和BgcIndex。

第1步，导出缺省运动信息motionInfoDefault。

F是当前预测单元E的相邻预测块（见图19）。将F的运动信息按9.5.7.8.10.3导出当前预测单元的缺省运动信息motionInfoDefault(mvDefaultL0,mvDefaultL1,refDefaultL0,refDefaultL1，predModeDefault)，其中参考预测单元为F所在的空域存储单元，mvDefaultLX等于mvEX，refDefaultLX等于refIndexLX,预测参考模式predModeDefault等于interPredRefMode。如果当前图像是P图像,X为0；如果当前图像是B图像，X为0或1。

第2步，导出坐标(Mx,My)。

令(Xpos,Ypos)是当前预测单元在图像中的左上角坐标(x_ctb_pos,y_ctb_pos)是当前最大编码单元在图像中的左上角坐标，cu_width和cu_height分别是当前预测单元的宽度和高度，PicWidthInLuma和PicHeightInLuma分别是当前图像的宽度和高度，lcu_size是当前最大编码单元的大小。如果当前图像是P图像，时域参考图像是参考图像队列0中参考帧索引值为0的参考图像，如果当前图像是B图像，时域参考图像是参考图像队列1中参考帧索引值为0的参考图像。按以下方法导出坐标(Mx,My)：

```
Mx = Clip3(x_ctb_pos, min(x_ctb_pos + lcu_size - cu_width, PicWidthInLuma - cu_width), ((Xpos+4)>>3)<<3)
My = Clip3 (y_ctb_pos, min(y_ctb_pos + lcu_size - cu_height, PicHeightInLuma - cu_height), ((Ypos+4)>>3)<<3)
```

第3步，导出(Px,Py)。

令时域参考图像中的左上角坐标为(Mx,My)，所在的时域参考块为当前图像的原始匹配块，原始匹配块的宽度和高度与当前预测单元的宽度和高度一致。A1～A4，B1～B4，C1～C4是时域参考图像中位于原始匹配块周围的相邻4×4块（见图30），按以下方法导出坐标(Px,Py)。

a) 构建增强时域候选列表 EtmvpCandArray[i]（i=0～4），其中判断运动信息是否相同的方法见9.5.7.2。
   1) 将 EtmvpCandArray[i]初始化为 0，i=0～4；length 初始化为 0。
   2) 令 EtmvpCandArray[0]等于 0，length 加 1。
   3) 如果 My+8 小于或等于 Min(y_ctb_pos + lcu_size - cu_height,PicHeightInLuma - cu_height)，且 A1 与 C3 位置的运动信息不同或 A2 与 C4 位置的运动信息不同，则 EtmvpCandArray[length]等于 1，length 加 1。
   4) 如果 Mx+8 小于或等于 Min(x_ctb_pos + lcu_size - cu_width,PicWidthInLuma - cu_width)，且 A1 与 B2 位置的运动信息不同或 A3 与 B4 位置的运动信息不同，则 EtmvpCandArray[length]等于 2，length 加 1。
   5) 如果 My-8 大于或等于 y_ctb_pos，且 A3 与 C1 位置的运动信息不同或 A4 与 C2 位置的运动信息不同，则 EtmvpCandArray[length]等于 3，length 加 1。
   6) 如果 Mx-8 大于或等于 x_ctb_pos，且 A2 与 B1 位置的运动信息不同或 A4 与 B3 位置的运动信息不同，则 EtmvpCandArray[length]等于 4，length 加 1。
   7) 如果 length 小于 5，执行以下操作：

```
while (length < 5) {
    EtmvpCandArray[length] = EtmvpCandArray[length-1]
```

```
    length++
}
```

b) 根据 EtmvpCandIndex 和 EtmvpCandArray 查表 98 得到(Px,Py)。

表98 EtmvpCandArray[EtmvpCandIndex]与 Px、Py 的对应关系

EtmvpCandArray[EtmvpCandIndex]的值	Px 的值	Py 的值
0	Mx	My
1	Mx	My+8
2	Mx+8	My
3	Mx	My－8
4	Mx-8	My

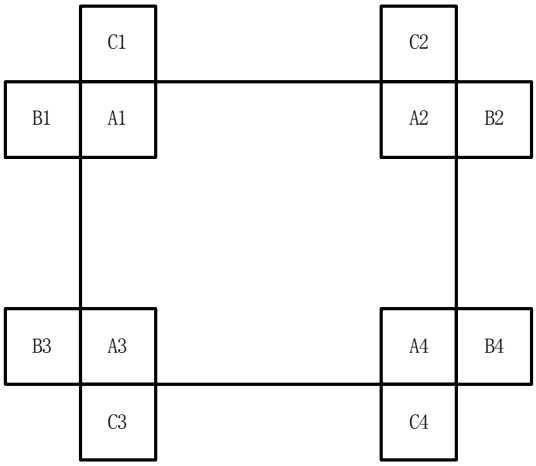


图30 原始匹配块与空间相邻块的关系

第4步，导出运动信息阵列MotionArray和BgcFlag和BgcIndex。

令MotionArray[i][j]为当前预测单元内部8×8子块的运动信息(interPredRefMode, MvE0, MvE1, RefIndexL0, RefIndexL1)，(i,j)是当前预测单元内8×8子块的索引，i是子块水平索引值，j是子块垂直索引值，i=0～(cu_width >> 3)-1，j=0～(cu_height >> 3)-1)。

令子块的时域运动信息是时域参考图像中左上角(Px+(i<<3),Py+(j<<3))位置所在的时域存储单元的运动信息。子块的时域运动信息“可用”指时域参考图像中左上角(Px+(i<<3)，Py+(j<<3))位置所在的时域存储单元采用帧间预测；否则子块的时域运动信息“不可用”。

按以下步骤导出运动信息阵列MotionArray。

a) 根据(Px,Py)确定当前预测单元的每个 8×8 子块的运动信息阵列 MotionArray，并将 BgcFlag 和 BgcIndex 的值均置为 0。

b) 依次遍历时域参考图像左上角(Px+(i<<3)，Py+(j<<3))位置的每个尺寸大小为 8×8 的子块，导出运动信息阵列 MotionArray[i][j]。

1) 如果当前子块的时域运动信息“可用”，将子块的时域运动信息按 9.5.7.8.10.3 导出 MotionArray[i][j]，其中参考预测单元是时域参考图像中(Px+(i<<3)，Py+(j<<3))位置所在时域存储单元，MotionArray[i][j]中的 MvEX 等于 mvEX，MotionArray[i][j]中的 RefIndexLX 等于 refIndexLX，MotionArray[i][j]中的 interPredRefMode 等于

interPredRefMode。如果当前块所在图像是 P 图像，则 X 为 0；如果当前块所在图像是 B 图像，则 X 为 0 或 1。

2) 否则，MotionArray[i][j]等于 motionInfoDefault。

9.5.7.8.10.3 增强时域运动信息辅助导出方法

如果当前图像是 P 图像，按以下方法导出时域运动信息。

a) 如果参考预测单元的预测参考模式是‘PRED_List0’，将参考预测单元所在图像的距离索引记为 DistanceIndexCol，参考预测单元的参考图像队列 0 中对应的图像的距离索引记为 DistanceIndexRef，参考预测单元所在参考图像队列 0 中图像的 BlockDistance 分别记为 BlockDistanceRef，参考预测单元的 L0 运动矢量记为 mvRef。将当前预测单元所在的图像的距离索引记为 DistanceIndexCur，当前预测单元的 L0 参考索引对应的图像的距离索引分别记为 DistanceIndexL0，当前预测单元的 L0 参考索引对应的图像的 BlockDistance 记为 BlockDistanceL0。

1) 计算 BlockDistanceRef：

```
BlockDistanceRef = DistanceIndexCol - DistanceIndexRef
```

2) 当前预测单元的预测参考模式 interPredRefMode 是‘PRED_List0’，当前预测单元的 L0 参考索引 refIndexL0 等于 0，计算当前预测单元的 L0 运动矢量 mvE0：

```
mvE0->x = Clip3(-32768, 32767, Sign(mvRef->x * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->x * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE0->y = Clip3(-32768, 32767, Sign(mvRef->y * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->y * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

b) 否则，当前预测单元的预测参考模式 interPredRefMode 是‘PRED_List0’；当前预测单元的 L0 参考索引 refIndexL0 等于 0，当前预测单元的 L0 运动矢量 mvE0 是零矢量。

如果当前图像是 B 图像，按以下方法导出时域运动信息。

a) 如果参考预测单元的预测参考模式是‘PRED_List0’，将参考预测单元所在图像的距离索引记为 DistanceIndexCol，参考预测单元的参考图像队列 1 中对应的图像的距离索引记为 DistanceIndexRef，参考预测单元所在参考图像队列 0 中图像的 BlockDistance 分别记为 BlockDistanceRef，参考预测单元的 L0 运动矢量记为 mvRef。将当前预测单元所在的图像的距离索引记为 DistanceIndexCur，当前预测单元的 L0 参考索引对应的图像的距离索引分别记为 DistanceIndexL0，当前预测单元的 L0 参考索引对应的图像的 BlockDistance 记为 BlockDistanceL0。

1) 计算 BlockDistanceRef：

```
BlockDistanceRef = DistanceIndexCol - DistanceIndexRef
```

2) 当前预测单元的预测参考模式 interPredRefMode 是‘PRED_List0’，当前预测单元的 L0 参考索引 refIndexL0 等于 0，计算当前预测单元的 L0 运动矢量 mvE0：

```
mvE->x = Clip3(-32768, 32767, Sign(mvRef->x * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->x * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE->y = Clip3(-32768, 32767, Sign(mvRef->y * BlockDistanceL0 * BlockDistanceRef) * (((Abs(mvRef->y * BlockDistanceL0 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

b) 否则，如果参考预测单元的预测参考模式是‘PRED_List1’，将参考预测单元所在图像的距离索引记为 DistanceIndexCol，参考预测单元的参考图像队列 1 中对应的图像的距离索引记为 DistanceIndexRef，参考预测单元所在参考图像队列 1 中图像的 BlockDistance 分别记为 BlockDistanceRef，参考预测单元的 L1 运动矢量记为 mvRef。将当前预测单元所在的图像的距离索引记为 DistanceIndexCur，当前预测单元的 L1 参考索引对应的图像的距离索引分别记为 DistanceIndexL1，当前预测单元的 L1 参考索引对应的图像的 BlockDistance 记为 BlockDistanceL1。

1) 计算 BlockDistanceRef：

```
BlockDistanceRef = DistanceIndexCol - DistanceIndexRef
```

2) 当前预测单元的预测参考模式 interPredRefMode 是‘PRED_List1’，当前预测单元的 L1 参考索引 refIndexL1 等于 0，计算当前预测单元的 L1 运动矢量 mvE1：

```
mvE->x = Clip3(-32768, 32767, Sign(mvRef->x * BlockDistanceL1 * BlockDistanceRef) * (((Abs(mvRef->x * BlockDistanceL1 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
mvE->y = Clip3(-32768, 32767, Sign(mvRef->y * BlockDistanceL1 * BlockDistanceRef) * (((Abs(mvRef->y * BlockDistanceL1 * (16384 / BlockDistanceRef))) + 8192 ) >> 14))
```

c) 否则，如果参考预测单元的预测参考模式是‘PRED_List01’，将参考预测单元所在图像的距离索引记为 DistanceIndexCol，参考预测单元的参考图像队列 0 和参考图像队列 1 中对应的图像的距离索引记为 DistanceIndexRef0 和 DistanceIndexRef1，参考预测单元所在参考图像队列 0 和参考图像队列 1 中图像的 BlockDistance 分别记为 BlockDistanceRef0 和 BlockDistanceRef1，参考预测单元的 L0 和 L1 运动矢量分别记为 mvRef0 和 mvRef1。将当前预测单元所在的图像的距离索引记为 DistanceIndexCur，当前预测单元的 L0 和 L1 参考索引对应的图像的距离索引分别记为 DistanceIndexL0 和 DistanceIndexL1；当前预测单元的 L0 和 L1 参考索引对应的图像的 BlockDistance 记为 BlockDistanceL0 和 BlockDistanceL1。

1) 计算 BlockDistanceRef0 和 BlockDistanceRef1：

```
BlockDistanceRef0 = DistanceIndexCol - DistanceIndexRef0
BlockDistanceRef1 = DistanceIndexCol - DistanceIndexRef1
```

2) 当前预测单元的预测参考模式 interPredRefMode 是‘PRED_List01’，当前预测单元的 L0 参考索引 refIndexL0 和 L1 参考索引 refIndexL1 均等于 0，分别计算当前预测单元的 L0 运动矢量 mvE0 和 L1 运动矢量 mvE1：

```
mvE0->x = Clip3(-32768, 32767, Sign(mvRef0->x * BlockDistanceL0 * BlockDistanceRef0) * (((Abs(mvRef0->x * BlockDistanceL0 * (16384 / BlockDistanceRef0))) + 8192 ) >> 14))
mvE0->y = Clip3(-32768, 32767, Sign(mvRef0->y * BlockDistanceL0 * BlockDistanceRef0) * (((Abs(mvRef0->y * BlockDistanceL0 * (16384 / BlockDistanceRef0))) + 8192 ) >> 14))

mvE1->x = Clip3(-32768, 32767, Sign(mvRef1->x * BlockDistanceL1 * BlockDistanceRef1) * (((Abs(mvRef1->x * BlockDistanceL1 * (16384 / BlockDistanceRef1))) + 8192 ) >> 14))
mvE1->y = Clip3(-32768, 32767, Sign(mvRef1->y * BlockDistanceL1 * BlockDistanceRef1) * (((Abs(mvRef1->y * BlockDistanceL1 * (16384 / BlockDistanceRef1))) + 8192 ) >> 14))
```

d) 否则，将当前预测单元的预测参考模式 interPredRefMode 设为‘PRED_List01’，当前预测单元的 L0 参考索引 refIndexL0 和 L1 参考索引均为 0，当前预测单元的 L0 运动矢量 mvE0 和 L1 运动矢量 mvE1 均为零矢量。

9.5.7.9 更新历史运动信息表

完成当前预测单元的解码后，如果满足以下条件之一，则不执行本条定义的操作：

a) 当前预测单元是仿射预测单元或角度加权预测单元；
b) 当前编码单元的编码单元子类型为‘P_Skip_Mvap’或‘P_Direct_Mvap’或‘B_Skip_Mvap’或‘B_Direct_Mvap’；
c) 当前编码单元的编码单元子类型为‘P_Skip_Etmvp’或‘P_Direct_Etmvp’或‘B_Skip_Etmvp’或‘B_Direct_Etmvp’；
d) 当前编码单元的编码单元子类型为‘P_Skip_SbTemporal’或‘P_Direct_SbTemporal’或‘B_Skip_SbTemporal’或‘B_Direct_SbTemporal’。

当NumOfHmvpCand大于0时，根据当前预测块的运动信息、BgcFlag和BgcIndex更新历史运动信息表HmvpCandList；否则，不执行本条定义的操作。

更新历史运动信息表的方法如下。

a) 将 hmvpIndex 初始化为 0。
b) 如果 CntHmvp 等于 0，则 HmvpCandList[CntHmvp]中的运动信息、BgcFlag 和 BgcIndex 分别等于当前预测单元的运动信息、BgcFlag 和 BgcIndex，CntHmvp 加 1。
c) 否则，按 9.5.7.2 判断当前预测块的运动信息和 HmvpCandList[hmvpIndex]是否相同：
   1) 如果运动信息相同，执行步骤 d)，否则，hmvpIndex 加 1；
   2) 如果 hmvpIndex 小于 CntHmvp，执行步骤 c)，否则，执行步骤 d)。
d) 如果 hmvpIndex 小于 CntHmvp，则：
   1) i 从 hmvpIndex 到 CntHmvp-1，令 HmvpCandList[i]等于 HmvpCandList[i+1]；
   2) HmvpCandList[CntHmvp-1]中的运动信息、BgcFlag 和 BgcIndex 分别等于当前预测单元的运动信息、BgcFlag 和 BgcIndex。

   否则，如果 hmvpIndex 等于 CntHmvp 且 CntHmvp 等于 NumOfHmvpCand，则：
   1) i 从 0 到 CntHmvp-1，令 HmvpCandList[i]等于 HmvpCandList[i+1]；
   2) HmvpCandList[CntHmvp-1]中的运动信息、BgcFlag 和 BgcIndex 分别等于当前预测单元的运动信息、BgcFlag 和 BgcIndex。

   否则，如果 hmvpIndex 等于 CntHmvp 且 CntHmvp 小于 NumOfHmvpCand，则 HmvpCandList[CntHmvp]中的运动信息、BgcFlag 和 BgcIndex 分别等于当前预测单元的运动信息、BgcFlag 和 BgcIndex，CntHmvp 加 1。

9.5.7.10 运动信息的存储

9.5.7.10.1 导出角度加权预测模式运动信息存储方式

令W和H是当前编码单元的宽度和高度，(x,y)是当前编码单元中的每个4×4块的左上角像素位置，posX等于((x >> 2) << 2) + 2，posY等于((y >> 2) << 2) + 2。按以下方法导出每个4×4块的运动信息存储方式AwpMvType：

a) 计算 stepIndex、angleIndex 和 angelAreaIndex 的值。

```
stepIndex = (AwpIndex >> 3) - 3
modAngNum = AwpIndex % 8
```

```
if (modAngNum == 2) {
    angleIndex = 7
}
else if (modAngNum == 6) {
    angleIndex = 8
}
else {
    angleIndex = modAngNum % 2
}
angleAreaIndex = modAngNum >> 1
```

b) 根据 angleAreaIndex 查表 99 得到 tP 和 fP，并计算当前编码单元中每个 4×4 块的运动信息存储方式 AwpMvType。

```
if (tP >= fP) {
    AwpMvType = 0
}
else {
    AwpMvType = 1
}
```

表99 angleAreaIndex 和 fP、tP 的对应关系

angleAreaIndex	fP	tP
0	H + (W >> angleIndex) - 2 + stepIndex * (((H + (W >> angleIndex)) >>2) -1)	(posY << 1) + ((posX << 1) >> angleIndex)
1	H + (W >> angleIndex) + stepIndex * (((H + (W >> angleIndex)) >> 2) -1) - ((W << 1) >> angleIndex)	(posY << 1) - ((posX << 1) >> angleIndex)
2	W + (H >> angleIndex) + stepIndex * (((W + (H >> angleIndex)) >> 2) -1) - ((H << 1) >> angleIndex)	(posX << 1) - ((posY << 1) >> angleIndex)
3	W + (H >> angleIndex) - 2 + stepIndex * (((W + (H >> angleIndex)) >> 2) -1)	(posX << 1) + ((posY << 1) >> angleIndex)

9.5.7.10.2 空域运动信息的存储

首先，确定样本(x,y)对应的空域运动信息存储单元的左上角像素点位置(x_0,y_0)：

```
x0 = (x >> 2) << 2
y0 = (y >> 2) << 2
```

其次，设X是(x_0,y_0)亮度样本所在的预测单元，RefIndexL0和RefIndexL1分别是X的L0参考索引和L1参考索引，interPredRefMode是X的预测参考模式，按以下方法将运动信息存储到样本(x,y)对应的空域运动信息存储单元。

a) 如果(x_0, y_0)亮度样本所在的预测单元是普通帧内预测或块复制帧内预测或串复制帧内预测，则该运动信息存储单元的 L0 运动矢量和 L1 运动矢量均为零矢量，L0 参考索引和 L1 参考索引均为-1，该运动信息存储单元的 BgcFlag 和 BgcIndex 均为 0，该运动信息存储单元的预测参考模式设为‘PRED_I’。

b) 否则，如果 X 的 AffineFlag 值为 0，且 X 的 AwpFlag 值为 0，该运动信息存储单元的预测参考模式是 interPredRefMode：

1) 如果 interPredRefMode 等于 0，则该运动信息存储单元的 L0 运动矢量和 L0 参考索引分别等于 X 的 L0 运动矢量和 L0 参考索引，该运动信息存储单元的 L1 运动矢量是零矢量，L1 参考索引等于-1，该运动信息存储单元的 BgcFlag 和 BgcIndex 均为 0；

2) 否则，如果 interPredRefMode 等于 1，则该运动信息存储单元的 L1 运动矢量和 L1 参考索引分别等于 X 的 L1 运动矢量和 L1 参考索引，该运动信息存储单元的 L0 运动矢量是零矢量，L0 参考索引等于-1，该运动信息存储单元的 BgcFlag 和 BgcIndex 均为 0；

3) 否则，如果 interPredRefMode 等于 2，则该运动信息存储单元的 L0 运动矢量和 L0 参考索引分别等于 X 的 L0 运动矢量和 L0 参考索引，该运动信息存储单元的 L1 运动矢量和 L1 参考索引分别等于 X 的 L1 运动矢量和 L1 参考索引，该运动信息存储单元的 BgcFlag 和 BgcIndex 分别等于 X 的 BgcFlag 和 BgcIndex。

c) 否则，如果 X 的 AffineFlag 值为 0，且 X 的 AwpFlag 值为 1，按 9.5.6.6 导出角度加权预测模式的存储方式 AwpMvType，BgcFlag 和 BgcIndex 均为 0：

1) 如果 AwpMvType 等于 0，则该运动信息存储单元的 L0 运动矢量和 L0 参考索引分别等于 mvAwp0L0 和 RefIndexAwp0L0，该运动信息存储单元的 L1 运动矢量和 L1 参考索引分别等于 mvAwp0L1 和 RefIndexAwp0L1；

2) 否则，如果 AwpMvType 等于 1，则该运动信息存储单元的 L0 运动矢量和 L0 参考索引分别等于 mvAwp1L0 和 RefIndexAwp1L0，该运动信息存储单元的 L1 运动矢量和 L1 参考索引分别等于 mvAwp1L1 和 RefIndexAwp1L1。

d) 否则，该运动信息存储单元的预测参考模式是 interPredRefMode，该运动信息存储单元的仿射类型等于该预测单元的仿射类型：

1) 如果 interPredRefMode 等于 0，则该运动信息存储单元的 L0 运动矢量和 L0 参考索引分别等于 mvAffineL0 和 RefIndexL0，该运动信息存储单元的 L1 运动矢量是零矢量，L1 参考索引等于-1，该运动信息存储单元的 BgcFlag 和 BgcIndex 均为 0；

2) 否则，如果 interPredRefMode 等于 1，则该运动信息存储单元的 L1 运动矢量和 L1 参考索引分别等于 mvAffineL1 和 RefIndexL1，该运动信息存储单元的 L0 运动矢量是零矢量，L0 参考索引等于-1，该运动信息存储单元的 BgcFlag 和 BgcIndex 均为 0；

3) 否则，如果 interPredRefMode 等于 2，则该运动信息存储单元的 L0 运动矢量和 L0 参考索引分别等于 mvAffineL0 和 RefIndexL0，该运动信息存储单元的 L1 运动矢量和 L1 参考索引分别等于 mvAffineL1 和 RefIndexL1，该运动信息存储单元的 BgcFlag 和 BgcIndex 分别等于 X 的 BgcFlag 和 BgcIndex。

其中，mvE0是MvArrayL0在(x_0, y_0)位置的4×4单元的L0运动矢量，mvE1是MvArrayL1在(x_0, y_0)位置的4×4单元的L1运动矢量。

mvAffineL0和mvAffineL1为：

```
mvAffineL0->x = Clip3(-32768, 32767, Rounding(mvE0->x,2))
mvAffineL0->y = Clip3(-32768, 32767, Rounding(mvE0->y,2))
mvAffineL1->x = Clip3(-32768, 32767, Rounding(mvE1->x,2)
mvAffineL1->y = Clip3(-32768, 32767, Rounding(mvE1->y,2))
```

9.5.7.10.3 时域运动信息的存储

首先，确定样本(x,y)所对应的运动信息存储单元的左上角像素点位置(x_0,y_0)、宽度nWidth和高度nHeight：

```
x0 = (x >> 4) << 4
y0 = (y >> 4) << 4
if (x0 + 16 >= PictureWidthInMinBu * MiniSize){
  nWidth = PictureWidthInMinBu * MiniSize  -  x0
}
else {
  nWidth = 16
}
if (y0 + 16 >= PictureHeightInMinBu * MiniSize) {
  nHeight = PictureHeightInMinBu * MiniSize  -  y0
}
else {
  nHeight = 16
}
```

其次，确定将运动信息存储到样本(x,y)对应的运动信息存储单元的方法如下：

a) 如果 EtmvpMvapEnableFlag、SbTmvpEnableFlag 和 AwpEnableFlag 的值均为 0：
   1) 如果(x0+nWidth/2,y0+nHeight/2)亮度样本所在编码单元的预测类型为普通帧内预测或块复制帧内预测或串复制帧内预测，或(x0+nWidth/2,y0+nHeight/2)亮度样本所在预测单元的预测参考模式是‘PRED_List1’，则将该运动信息存储单元的 L0 参考索引值和 L1 参考索引值均设为-1，将该运动信息存储单元的预测参考模式设为‘PRED_I’；
   2) 否则，将该运动信息存储单元的运动矢量和参考索引设为(x0+nWidth/2,y0+nHeight/2)亮度样本所在 4×4 预测块的 L0 运动矢量和 L0 参考索引，并将该运动信息存储单元的预测参考模式设为‘PRED_List0’；
   3) 该运动信息存储单元的 BgcFlag 和 BgcIndex 均为 0。
b) 如果 EtmvpMvapEnableFlag 的值为 1 或 SbTmvpEnableFlag 的值为 1 或 AwpEnableFlag 的值为 1。
   1) 如果(x0+nWidth/2,y0+nHeight/2)亮度样本所在编码单元的预测类型为普通帧内预测或块复制帧内预测或串复制帧内预测，则将该运动信息存储单元的预测参考模式设为‘PRED_I’。
   2) 否则，如果(x0+nWidth/2,y0+nHeight/2)亮度样本所在预测单元的预测参考模式是‘PRED_List0’将该运动信息存储单元的 L0 的运动矢量和参考索引设为(x0+nWidth/2,y0+nHeight/2)亮度样本所在 4×4 预测块的 L0 的运动矢量和参考索引，将该运动信息存储单元的预测参考模式设为‘PRED_List0’；如果(x0+nWidth/2,y0+nHeight/2)亮度样本所在预测单元的预测参考模式是‘PRED_List1’将该运动信息存储单元的 L1 的运动矢量和参考索引设为(x0+nWidth/2,y0+nHeight/2)亮度样本所在 4×4 预测块的 L1 的运动矢量和参考索引，将该运动信息存储单元的预测参考模式设为‘PRED_List1’；如果(x0+nWidth/2,y0+nHeight/2)亮度样本所在预测单元的预测参考模式是‘PRED_List01’，将该运动信息存储单元的 L0 的运动矢量和参考索引设为

(x0+nWidth/2,y0+nHeight/2)亮度样本所在4×4预测块的L0运动矢量和参考索引，将该运动信息存储单元的L1的运动矢量和参考索引设为(x0+nWidth/2,y0+nHeight/2)亮度样本所在4×4预测块的L1的运动矢量和参考索引，并将该运动信息存储单元的预测参考模式设为‘PRED_List01’。

3) 该运动信息存储单元的BgcFlag和BgcIndex均为0。

9.5.7.11 导出相邻块单向运动信息矩阵

本条定义相邻块单向运动信息矩阵导出过程。当前编码单元的宽度和高度为W和H，当前编码单元左上角样本位置为(xCb,yCb)。

当前编码单元上侧相邻块单向运动信息矩阵为MotionInfoTop[xSbIndex]，左侧相邻块单向运动信息矩阵为MotionInfoLeft[ySbIndex]，其中xSbIndex的取值范围为0～m-1，ySbIndex的取值范围为0～n-1,m和n的分别为W/4和H/4,其中MotionInfoTop和MotionInfoLeft中每个运动信息单元包含mvE0,mvE1，refIndexL0，refIndexL1，interPredRefMode。

第1步，按以下步骤导出上侧相邻块单向运动信息矩阵MotionInfoTop。

对于xSbIndex从0～m-1,上侧相邻子块位置(xAbove,yAbove)为((xCb>>2)+xSbIndex,(yCb>>2)-1)。记(xAbove,yAbove)对应的亮度样本所在的空域运动信息存储单元为aboveMotion。

a) 如果上侧相邻子块不存在，设置上侧相邻块单向运动信息MotionInfoTop[xSbIndex]中的运动矢量mvE0和mvE1为零矢量，L0参考索引refIndexL0和L1参考索引refIndexL1等于-1，interPredRefMode设置为‘PRED_I’。

b) 否则，如果aboveMotion中的interPredRefMode不等于‘PRED_List01’，则MotionInfoTop[xSbIndex]中的运动信息等于aboveMotion中的运动信息。

c) 否则（即aboveMotion中的interPredRefMode等于‘PRED_List01’），将当前图像的距离索引记为DistanceIdexCur，上侧相邻块L0参考索引和L1参考索引对应的图像的距离索引分别记为DistanceIndexL0和DistanceIndexL1，计算BlockDistanceL0=Abs(DistanceIndexCur – DistanceIndexL0)，BlockDistanceL1=Abs(DistanceIndexCur – DistanceIndexL1)，将上侧相邻子块L0参考索引和L1参考索引对应的图像的图像级量化参数分别记为QP0和QP1。

1) 如果BlockDistanceL0小于BlockDistanceL1，设置上侧相邻块单向运动信息MotionInfoTop[xSbIndex]为：将运动矢量mvE0和参考索引refIndexL0分别设置为aboveMotion中L0运动矢量和L0参考索引，运动矢量mvE1和参考索引refIndexL1分别设置为零矢量和-1，interPredRefMode设置为‘PRED_List0’。

2) 否则，如果BlockDistanceL1小于BlockDistanceL0，设置上侧相邻块单向运动信息MotionInfoTop[xSbIndex]为：将运动矢量mvE1和参考索引refIndexL1分别设置为aboveMotion中L1运动矢量和L1参考索引，运动矢量mvE0和参考索引refIndexL0分别设置为零矢量和-1，interPredRefMode设置为‘PRED_List1’。

3) 否则，如果上侧相邻块L0参考索引对应的图像的CuDeltaQpFlag和L1参考索引对应的图像的CuDeltaQpFlag均等于0且QP1小于QP0，设置上侧相邻块单向运动信息MotionInfoTop[xSbIndex]为：将运动矢量mvE1和参考索引refIndexL1分别设置为aboveMotion中L1运动矢量和L1参考索引，运动矢量mvE0和参考索引refIndexL0分别设置为零矢量和-1，interPredRefMode设置为‘PRED_List1’。

4) 否则，设置上侧相邻块单向运动信息MotionInfoTop[xSbIndex]为：将运动矢量mvE0和参考索引refIndexL0分别设置为aboveMotion中L0运动矢量和L0参考索引，运动矢量mvE1和参考索引refIndexL1分别设置为零矢量和-1，interPredRefMode设置为‘PRED_List0’。

第2步，按以下步骤导出左侧相邻块单向运动信息矩阵MotionInfoLeft。

对于ySbIndex从0到n-1每个取值，左侧相邻子块位置(xLeft,yLeft)为((xCb>>2)-1,(yCb>>2)+ySbIndex)。记(xLeft,yLeft)对应的亮度样本所在的空域运动信息存储单元为leftMotion。

a) 如果左侧相邻子块不存在,设置左侧相邻块单向运动信息MotionInfoLeft[ySbIndex]中的运动矢量mvE0和mvE1为零矢量；L0参考索引refIndexL0和L1参考索引refIndexL1等于-1。

b) 否则，leftMotion的interPredRefMode不等于‘PRED_List01’，则MotionInfoLeft[xSbIndex]中的运动信息等于leftMotion中的运动信息。

c) 否则(即leftMotion的interPredRefMode等于‘PRED_List01’)，将当前图像的距离索引记为DistanceIdexCur，左侧相邻块L0参考索引和L1参考索引对应的图像的距离索引分别记为DistanceIndexL0和DistanceIndexL1，计算BlockDistanceL0=Abs(DistanceIndexCur – DistanceIndexL0)，BlockDistanceL1=Abs(DistanceIndexCur – DistanceIndexL1)，将左侧相邻块L0参考索引和L1参考索引对应的图像的图像级量化参数分别记为QP0和QP1。

1) 如果BlockDistanceL0小于BlockDistanceL1,设置左侧相邻块单向运动信息MotionInfoLeft[ySbIndex]为：将运动矢量mvE0和参考索引refIndexL0分别设置为leftMotion中L0运动矢量和L0参考索引,运动矢量mvE1和参考索引refIndexL1分别设置为零矢量和-1,interPredRefMode设置为‘PRED_List0’。

2) 否则，如果BlockDistanceL1小于BlockDistanceL0，设置左侧相邻块单向运动信息MotionInfoLeft[ySbIndex]为：将运动矢量mvE1和参考索引refIndexL1分别设置为leftMotion中L1运动矢量和L1参考索引,运动矢量mvE0和参考索引refIndexL0分别设置为零矢量和-1，interPredRefMode设置为‘PRED_List1’。

3) 否则，如果上侧相邻块L0参考索引对应的图像的CuDeltaQpFlag和L1参考索引对应的图像的CuDeltaQpFlag均等于0且QP1小于QP0,设置左侧相邻块单向运动信息MotionInfoLeft[ySbIndex]为：将运动矢量mvE1和参考索引refIndexL1分别设置为leftMotion中L1运动矢量和L1参考索引,运动矢量mvE0和参考索引refIndexL0分别设置为零矢量和-1，interPredRefMode设置为‘PRED_List1’。

4) 否则，设置左侧相邻块单向运动信息MotionInfoLeft[ySbIndex]为：将运动矢量mvE0和参考索引refIndexL0分别设置为leftMotion中L0运动矢量和L0参考索引,运动矢量mvE1和参考索引refIndexL1分别设置为零矢量和-1，interPredRefMode设置为‘PRED_List0’。

9.6 变换块解码

9.6.1 概述

本条定义$M_1 \times M_2$变换块的解码过程。变换块的量化系数经过反量化（见9.6.2）和反变换（见9.6.3），得到残差样本矩阵。

9.6.2 反量化

本条定义将$M_1 \times M_2$二维量化系数矩阵QuantCoeffMatrix转换为$M_1 \times M_2$二维变换系数矩阵CoeffMatrix的过程。从符合本文件的位流中解码得到的QuantCoeffMatrix的元素的取值范围应为$-2^{15} \sim 2^{15}-1$。

如果帧内预测模式既不是‘Intra_Luma_PCM’也不是‘Intra_Chroma_PCM’，二维变换系数矩阵CoeffMatrix的计算如下：

```
m = Log(M1 * M2) / 2
shift1 = m + BitDepth - 14
shift2 = 1 << 7
```

```
for (i=0; i<M1; i++) {
    for (j=0; j<M2; j++) {
        CoeffMatrix[i][j] = Clip3(-32768, 32767, (((((QuantCoeffMatrix[i][j] * WeightQuantMatrixM1xM2[i][j]) >> WqmShift) * (DequantTable(QPX)) >> 4) + 2^((ShiftTable(QPX)+shift1)-1)) >> (ShiftTable(QPX) + shift1)))
    }
}
if (((M1 == 2 * M2) || (M2 == 2 * M1)) || ((M1 == 8 * M2) || (M2 == 8 * M1))) {
    for (i=0; i<M1; i++) {
        for (j=0; j<M2; j++) {
            CoeffMatrix[i][j] = (CoeffMatrix[i][j] * 181 + shift2) >> 8
        }
    }
}
```

如果帧内预测模式是‘Intra_Luma_PCM’或‘Intra_Chroma_PCM’，二维变换系数矩阵CoeffMatrix的计算如下：

```
for (i=0; i<M1; i++) {
    for (j=0; j<M2; j++) {
        CoeffMatrix[i][j] = QuantCoeffMatrix[i][j]
    }
}
```

量化参数QP_X（X为Y、Cb或Cr）与DequantTable和ShiftTable的关系见表100。

表100 QP_X与DequantTable和ShiftTable的关系

QP_X的值	DequantTable(QP_X)的值	ShiftTable(QP_X)的值
0	32768	14
1	36061	14
2	38968	14
3	42495	14
4	46341	14
5	50535	14
6	55437	14
7	60424	14
8	32932	13
9	35734	13
10	38968	13
11	42495	13
12	46177	13
13	50535	13
14	55109	13

表 100（续）

QP_X的值	DequantTable(QP_X)的值	ShiftTable(QP_X)的值
15	59933	13
16	65535	13
17	35734	12
18	38968	12
19	42577	12
20	46341	12
21	50617	12
22	55027	12
23	60097	12
24	32809	11
25	35734	11
26	38968	11
27	42454	11
28	46382	11
29	50576	11
30	55109	11
31	60056	11
32	65535	11
33	35734	10
34	38968	10
35	42495	10
36	46320	10
37	50515	10
38	55109	10
39	60076	10
40	65535	10
41	35744	9
42	38968	9
43	42495	9
44	46341	9
45	50535	9
46	55099	9
47	60087	9
48	65535	9
49	35734	8
50	38973	8
51	42500	8
52	46341	8
53	50535	8

表 100（续）

QP_X的值	DequantTable(QP_X)的值	ShiftTable(QP_X)的值
54	55109	8
55	60097	8
56	32771	7
57	35734	7
58	38965	7
59	42497	7
60	46341	7
61	50535	7
62	55109	7
63	60099	7
64	32768	6
65	36061	6
66	38968	6
67	42495	6
68	46341	6
69	50535	6
70	55437	6
71	60424	6
72	32932	5
73	35734	5
74	38968	5
75	42495	5
76	46177	5
77	50535	5
78	55109	5
79	59933	5

9.6.3 反变换

9.6.3.1 概述

本条定义将$M_1 \times M_2$变换系数矩阵CoeffMatrix转换为残差样本矩阵ResidueMatrix的过程。

第1种情况，当前变换块的帧内预测模式是‘Intra_Luma_PCM’或‘Intra_Chroma_PCM’，按9.6.3.4导出残差样本矩阵ResidueMatrix。

第2种情况，当前变换块的帧内预测模式是普通帧内预测模式（不包括‘Intra_Luma_PCM’和‘Intra_Chroma_PCM’模式），或当前块的预测类型是块复制帧内预测，或当前块的预测类型是帧间预测。

——如果 PictureTsEnableFlag 为 0 且 IstTuFlag 为 1 且满足以下条件之一，按 9.6.3.3 导出残差样本矩阵 ResidueMatrix：

- 当前变换块是亮度普通帧内预测残差块，且 M_1 和 M_2 的值均小于 64；

- 当前变换块是亮度帧间预测残差块，且 M_1 和 M_2 的值均小于 32。

——否则，如果 PictureTsEnableFlag 为 1 且 IstTuFlag 不为 0 且 M_1 和 M_2 的值均小于 64 且满足以下条件之一，按 9.6.3.5 导出残差样本矩阵 ResidueMatrix：

- 当前变换块是亮度帧内预测残差块；
- 当前变换块是亮度块复制帧内预测残差块；
- 当前变换块是亮度帧间预测残差块且当前变换块所在编码单元的 DirectFlag 为 0。

——否则，如果 PictureTsEnableFlag 为 1 且 M_1 和 M_2 的值均小于 64 且当前变换块是亮度帧间预测残差块且当前变换块所在编码单元的 SbtCuFlag 的值大于 0，按 9.6.3.5 导出残差样本矩阵 ResidueMatrix。

如果以上两种情况都不满足，按9.6.3.2导出残差样本矩阵ResidueMatrix。

9.6.3.2 传统反变换方法

本条定义导出残差样值矩阵ResidueMatrix的方法。

第1步，如果当前变换块是帧内预测残差块，M_1或M_2的值大于4且StEnableFlag的值等于1，对CoeffMatrix进行以下操作。

a) 由变换系数矩阵得到 4×4 矩阵 C。

```
for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
        C[i][j] = CoeffMatrix[i][j]
    }
}
```

b) 如果当前变换块满足以下 4 个条件之一，先得到矩阵 P，再得到 C，其中 S_4 是 4×4 反变换矩阵。

1) 当前变换块是亮度预测残差块，EnhancedStEnableFlag 为 0 且 IntraLumaPredMode 为 0～2 或 13～32 或 44～65，且 9.7.1.2 中坐标为(x0-1, y0+j-1)（j=1～M_2）的参考样本“可用”。
2) 当前变换块是亮度预测残差块，DtSplitFlag 为 1，且 IntraLumaPredMode 为 0～2 或 13～32 或 44～65，且 9.7.1.2 中坐标为(x0-1, y0+j-1)（j=1～M_2）的参考样本“可用”。
3) 当前变换块是亮度预测残差块，EnhancedStEnableFlag 为 1 且 EstTuFlag 为 1。
4) 当前变换块是色度预测残差块，EnhancedStEnableFlag 为 1 且 ChromaStFlag 为 1。

```
P = C * S4

for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
        C[i][j] = Clip3(-32768, 32767, (P[i][j] + 2^6) >> 7)
    }
}
```

c) 如果当前变换块满足以下 4 个条件之一，先得到矩阵 Q，再得到 C，其中 S_4^T 是 S_4 的转置。

1) 当前变换块是亮度预测残差块，EnhancedStEnableFlag 为 0 且 IntraLumaPredMode 为 0～23 或 34～57，且 9.7.1.2 中坐标为(x0+i-1, y0-1)（i=1～M_1）的参考样本“可用”。
2) 当前变换块是亮度预测残差块，DtSplitFlag 为 1，且 IntraLumaPredMode 为 0～23 或 34～57，且 9.7.1.2 中坐标为(x0+i-1, y0-1)（i=1～M_1）的参考样本“可用”。

3) 当前变换块是亮度预测残差块，EnhancedStEnableFlag 为 1 且 EstTuFlag 为 1。

4) 当前变换块是色度预测残差块，EnhancedStEnableFlag 为 1 且 ChromaStFlag 为 1。

```
Q = S4^T * C

for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
        C[i][j] = Clip3(-32768, 32767, (Q[i][j] + 2^6) >> 7)
    }
}
```

d) 根据矩阵 C 修改变换系数矩阵。

```
for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
        CoeffMatrixf[i][j] = C[i][j]
    }
}
```

第2步，对变换系数矩阵进行垂直反变换，得到矩阵K。

a) 由变换系数矩阵和反变换矩阵得到矩阵 V。

1) 如果 M_1 和 M_2 均等于 4 且 StEnableFlag 等于 1 且满足以下条件之一，则矩阵 V 如下，其中 D_4^T 是 4×4 反变换矩阵 D4 的转置：

- 如果当前变换块是亮度帧内预测残差块且 EnhancedStEnableFlag 为 0；
- 如果当前变换块是亮度帧内预测残差块且 DtSplitFlag 为 1；
- 如果当前变换块是亮度帧内预测残差块且 EnhancedStEnableFlag 为 1 且 EstTuFlag 为 1；
- 如果当前变换块是色度帧内预测残差块且 EnhancedStEnableFlag 为 1 且 ChromaStFlag 为 1。

```
V = D4^T * CoeffMatrix
```

2) 否则，如果 PbtCuFlag 的值为 1 且变换块的顺序号为 0 或 1，则矩阵 V 如下，其中 $DCT8_{M2}^T$ 是 $M_2 \times M_2$ 反变换矩阵 $DCT8_{M2}$ 的转置。

```
V = DCT8_M2^T * CoeffMatrix
```

3) 否则，如果 PbtCuFlag 的值为 1 且变换块的顺序号为 2 或 3，则矩阵 V 如下，其中 $DST7_{M2}^T$ 是 $M_2 \times M_2$ 反变换矩阵 $DST7_{M2}$ 的转置。

```
V = DST7_M2^T * CoeffMatrix
```

4) 否则，如果 SbtCuFlag 的值为 1 且变换块的顺序号为 0 且变换块的宽度小于 64 且变换块的高度小于 32，且 SbtDirFlag 为 1 且 SbtPosFlag 为 0，则矩阵 V 如下。

```
V = DCT8_M2^T * CoeffMatrix
```

5) 否则，如果 SbtCuFlag 的值为 1 且变换块的顺序号为 0 且变换块的宽度小于 64 且变换块的高度小于 32，且 SbtDirFlag 为 0 或 SbtPosFlag 为 1，则矩阵 V 如下。

$V = DST7_{M2}^{T} * CoeffMatrix$

6) 否则，矩阵 V 如下，其中 $DCT2_{M2}^{T}$ 是 $M_2 \times M_2$ 反变换矩阵 $DCT2_{M2}$ 的转置。

$V = DCT2_{M2}^{T} * CoeffMatrix$

b) 由矩阵 V 得到矩阵 K。

```
for (i=0; i<M1; i++) {
    for (j=0; j<M2; j++) {
        K[i][j] = Clip3(-32768, 32767, (V[i][j] + 2^4) >> 5)
    }
}
```

第3步，对矩阵K进行水平反变换，得到矩阵H。

a) 由变换系数矩阵和反变换矩阵得到矩阵 W，并得到 MaxValue 和 shift1。

1) 如果 M_1 和 M_2 的值均等于 4 且 StEnableFlag 的值等于 1 且满足以下条件之一，则 MaxValue 和 shift1 以及矩阵 W 如下，其中 D_4 是 4×4 反变换矩阵：

- 如果当前变换块是亮度帧内预测残差块且 EnhancedStEnableFlag 为 0；
- 如果当前变换块是亮度帧内预测残差块且 DtSplitFlag 为 1；
- 如果当前变换块是亮度帧内预测残差块且 EnhancedStEnableFlag 为 1 且 EstTuFlag 为 1；
- 如果当前变换块是色度帧内预测残差块且 EnhancedStEnableFlag 为 1 且 ChromaStFlag 为 1。

$W = K * D_4$

$MaxValue = (1 << BitDepth) - 1$

$shift1 = 22 - BitDepth$

2) 否则，如果 PbtCuFlag 的值为 1 且变换块的顺序号为 0 或 2，则 MaxValue 和 shift1 以及矩阵 W 如下，其中 $DCT8_{M1}$ 是 $M_1 \times M_1$ 反变换矩阵。

$W = K * DCT8_{M1}$

$MaxValue = (1 << BitDepth) - 1$

$shift1 = 20 - BitDepth$

3) 否则，如果 PbtCuFlag 的值为 1 且变换块的顺序号为 1 或 3，则 MaxValue 和 shift1 以及矩阵 W 如下，其中 $DST7_{M1}$ 是 $M_1 \times M_1$ 反变换矩阵。

$W = K * DST7_{M1}$

$MaxValue = (1 << BitDepth) - 1$

$shift1 = 20 - BitDepth$

4) 否则，如果 SbtCuFlag 的值为 1 且变换块的顺序号为 0 且变换块的宽度小于 32 且变换块的高度小于 64，且 SbtDirFlag 为 0 且 SbtPosFlag 为 0，则 MaxValue 和 shift1 以及矩阵 W 如下。

$W = K * DCT8_{M1}$

```
MaxValue = (1 << BitDepth) - 1
shift1 = 20 - BitDepth
```

5) 否则，如果 SbtCuFlag 的值为 1 且变换块的顺序号为 0 且变换块的宽度小于 32 且变换块的高度小于 64，且 SbtDirFlag 为 1 或 SbtPosFlag 为 1，则 MaxValue 和 shift1 以及矩阵 W 如下。

```
W = K * DST7M1
MaxValue = (1 << BitDepth) - 1
shift1 = 20 - BitDepth
```

6) 否则，MaxValue 和 shift1 以及矩阵 W 如下，其中 $DCT2_{M1}$ 是 $M_1 \times M_1$ 反变换矩阵。

```
W = K * DCT2M1
MaxValue = (1 << BitDepth) - 1
shift1 = 20 - BitDepth
```

c) 由矩阵 W 得到矩阵 H。

```
for (i=0; i<M1; i++) {
    for (j=0; j<M2; j++) {
        H[i][j] = Clip3(-MaxValue-1, MaxValue, (W[i][j] + 2^(shift1-1)) >> shift1)
    }
}
```

第4步，把矩阵H直接作为残差样值矩阵ResidueMatrix。

二次变换反变换矩阵S_4和D_4如下：

```
S4 = {
{123, -35, -8, -3}
{-32, -120, 30, 10}
{14, 25, 123, -22}
{8, 13, 19, 126}}

D4 = {
{34, 58, 72, 81}
{77, 69, -7, -75}
{79, -33 -75, 58}
{55, -84, 73, -28}}
```

DCT2型变换的变换核尺寸范围为4×4～64×64，DCT8型变换的变换核尺寸范围为4×4～16×16，DST7型变换的变换核尺寸范围为4×4～16×16。反变换矩阵应符合附录G的规定。

9.6.3.3 隐择反变换方法

本条定义导出残差样值矩阵ResidueMatrix的方法。

第1步，对变换系数矩阵进行垂直反变换，得到矩阵K。

a) 由变换系数矩阵和反变换矩阵得到矩阵 V，其中 $DST7_{M2}^{T}$ 是 $M_2 \times M_2$ 反变换矩阵 $DST7_{M2}$ 的转置。

```
V = DST7_M2^T * CoeffMatrix
```

b) 由矩阵 V 得到矩阵 K。

```
for (i=0; i<M1; i++) {
    for (j=0; j<M2; j++) {
        K[i][j] = Clip3(-32768, 32767, (V[i][j] + 2^4) >> 5)
    }
}
```

第 2 步，对矩阵 K 进行水平反变换，得到矩阵 H。

a) 由变换系数矩阵和反变换矩阵得到矩阵 W，并得到 MaxValue 和 shift1，其中 $DST7_{M2}$ 是 $M_2 \times M_2$ 反变换矩阵。

```
W = K * DST7_M2
MaxValue = (1 << BitDepth) - 1
shift1 = 20 - BitDepth
```

b) 由矩阵 W 得到矩阵 H。

```
for (i=0; i<M1; i++) {
    for (j=0; j<M2; j++) {
        H[i][j] = Clip3(-MaxValue-1, MaxValue, (W[i][j] + 2^(shift1-1)) >> shift1)
    }
}
```

第3步，把矩阵H直接作为残差样值矩阵ResidueMatrix。

DST7型变换的变换核尺寸范围为4×4～32×32。反变换矩阵应符合附录G的规定。

9.6.3.4 PCM 模式

本条定义导出残差样值矩阵ResidueMatrix的方法。

如果帧内预测模式是‘Intra_Luma_PCM’或‘Intra_Chroma_PCM’，按以下方法由变换系数矩阵CoeffMatrix得到残差样值矩阵ResidueMatrix。

```
for (i=0; i<M1; i++) {
    for (j=0; j<M2; j++) {
        ResidueMatrix[i][j] = CoeffMatrix[i][j] << (BitDepth - SamplePrecision)
    }
}
```

9.6.3.5 隐择反变换跳过方法

本条定义导出残差样值矩阵ResidueMatrix的方法。

第1步，由变换系数矩阵得到矩阵W。

a) 计算移位 shift。

```
shift = 15 - BitDepth - ((Log(M1) + Log(M2)) >> 1)
```

b) 计算矩阵 W。

```
rndFactor = 1 << (shift - 1)
for (i=0; i<M1; i++) {
    for (j=0; j<M2; j++) {
        W[i][j] = (CoeffMatrix[m][n] + rndFactor) >> shift
    }
}
```

第2步，把矩阵W直接作为残差样值矩阵ResidueMatrix。

第1步的步骤b)中位置索引m和n的计算如下：

a) 如果当前变换块是亮度帧内预测残差块，且 IstTuFlag 的值等于 2，则：

```
if (((i == (M1 >> 1) - 1) && (j == (M2 >> 1))) || ((i == (M1 >> 1)) && (j == ((M2 >> 1) - 1)))) {
    m = i
    n = j
}
else {
    m = M1 - 1 - i
    n = M2 - 1 - j
}
```

b) 否则：

```
m = i
n = j
```

9.7 帧内预测

9.7.1 普通帧内预测

9.7.1.1 概述

本条定义M×N亮度编码块和K×L色度编码块的帧内预测过程。

如果当前编码单元的类型是‘I_2M_nU’，将PredBlockOrder为1的帧内亮度预测块水平拆分成与PredBlockOrder为0的亮度预测块尺寸相同的3个子预测块，3个子预测块共用相同的帧内预测模式；否则，如果当前编码单元的类型是‘I_2M_nD’，将PredBlockOrder为0的帧内亮度预测块水平拆分成与PredBlockOrder为1的亮度预测块尺寸相同的3个子预测块，3个子预测块共用相同的帧内预测模式；否则，如果当前编码单元的类型是‘I_nL_2N’，将PredBlockOrder为1的帧内亮度预测块垂直拆分成与PredBlockOrder为0的亮度预测块尺寸相同的3个子预测块，3个子预测块共用相同的帧内预测模式；否则，如果当前编码单元的类型是‘I_nR_2N’，将PredBlockOrder为0的帧内亮度预测块垂直拆分成与PredBlockOrder为1的亮度预测块尺寸相同的3个子预测块，3个子预测块共用相同的帧内预测模式。

分别按9.7.1.2和9.7.1.3得到当前块的亮度和色度参考样本。

对于亮度编码块，如果SawpFlag等于0，根据IntraLumaPredMode按9.7.1.4进行亮度帧内预测，得到M×N亮度编码块的预测样本矩阵predMatrix。如果SawpFlag等于1，首先分别将SawpPredMode0和SawpPredMode1作为IntraLumaPredMode按9.7.1.4进行亮度帧内预测，得到M×N亮度编码块的预测样本矩阵predLumaMatrix0和predLumaMatrix1；然后根据predLumaMatrix0和predLumaMatrix1按9.7.1.6得到M×N亮度编码块的预测样本矩阵predMatrix。

对于色度编码块，如果SawpFlag等于1且IntraChromaPredMode等于0，首先分别将SawpPredMode0和SawpPredMode1作为当前编码单元中PredBlockOrder的值为0的预测块的IntraLumaPredMode按9.7.1.5进行色度帧内预测，得到K×L色度编码块的预测样本矩阵predChromaMatrix0和predChromaMatrix1；然后根据predChromaMatrix0和predChromaMatrix1按9.7.1.6得到K×L色度编码块的预测样本矩阵predMatrix。否则，按9.7.1.5进行色度帧内预测，得到K×L色度编码块的预测样本矩阵predMatrix。

对于亮度编码块,当前块上边的参考样本记为r[i],左边的参考样本记为c[j],其中r[0]等于c[0]，如果i大于2M，则r[i]等于r[2M]，如果j大于2N，则c[j]等于c[2N]。

对于色度编码块，当前块上边的参考样本记为row[i]，左边的参考样本记为col[j]，其中row[0]等于col[0]，如果i大于2K，则row[i]等于row[2K]，如果j大于2L，则col[i]等于col[2L]。

9.7.1.2 亮度参考样本的获得

用I表示当前块所在的图像的补偿后样本的亮度样值矩阵。

设当前块左上角样本在样本矩阵I中的坐标是(x_0, y_0)，其参考样本按以下规则获得。

a) 初始化 r[i]、c[j]为 $2^{BitDepth-1}$（i=0～2M，j=0～2N；BitDepth 是编码样本精度）。

b) 如果坐标为(x_0+i-1, y_0-1)（i=1～M）的样本均“可用”，则 r[i]等于 $I[x_0+i-1][y_0-1]$，r[i]“可用”；否则 r[i]“不可用”。

c) 如果坐标为(x_0+i-1, y_0-1)（i=M+1～2M）的样本“可用”，则 r[i]等于 $I[x_0+i-1][y_0-1]$；否则 r[i]等于 r[i-1]。

d) 如果坐标为(x_0-1, y_0+j-1)（j=1～N）的样本均“可用”，则 c[j]等于 $I[x_0-1][y_0+j-1]$，c[j]“可用”；否则 c[j]“不可用”。

e) 如果坐标为(x_0-1, y_0+j-1)（j=N+1～2N）的样本“可用”，则 c[j]等于 $I[x_0-1][y_0+j-1]$；否则 c[j]等于 c[j-1]。

f) 如果坐标为(x_0-1, y_0-1)的样本“可用”，则 r[0]等于 $I[x_0-1][y_0-1]$，r[0]“可用”。否则：

   1) 如果 r[1]“可用”并且 c[1]“不可用”，则 r[0]等于 r[1]，r[0]“可用”；

   2) 否则，如果 c[1]“可用”并且 r[1]“不可用”，则 r[0]等于 c[1]，r[0]“可用”；

   3) 否则，如果 r[1]“可用”并且 c[1]“可用”，则 r[0]等于 r[1]，r[0]“可用”，否则 r[0]“不可用”。

9.7.1.3 色度参考样本的获得

用I表示当前块所在的图像的补偿后样本的色度样值矩阵。

设当前块左上角样本在样本矩阵I中的坐标是(x_0, y_0)，其参考样本按以下规则获得：

a) 初始化 row[i]、col[i]为 $2^{BitDepth-1}$（i=0～2K，j=0～2L；BitDepth 是编码样本精度）。

b) 如果坐标为(x_0+i-1, y_0-1)(i=1～K)的样本均“可用”，则 row[i]等于 $I[x_0+i-1][y_0-1]$，row[i]“可用”；否则 row[i]“不可用”。

c) 如果坐标为(x_0+i-1, y_0-1)（i=K+1～2K）的样本“可用”，则 row[i]等于 $I[x_0+i-1][y_0-1]$；否则 row[i]等于 row[i-1]。

d) 如果坐标为(x_0-1,y_0+j-1)（j=1～L)的样本均“可用”，则 col[j]等于 I[x_0-1][y_0+j-1]，col[j]“可用”；否则 col[j]“不可用”。

e) 如果坐标为(x_0-1,y_0+j-1)（j=L+1～2L）的样本“可用”，则 col[j]等于 I[x_0-1][y_0+j-1]；否则 col[j]等于 col[j-1]。

f) 如果坐标为(x_0-1,y_0-1)的样本“可用”，则 row[0]等于 I[x_0-1][y_0-1]，row[0]“可用”。否则：

1) 如果 row[1]“可用”并且 col[1]“不可用”，则 row[0]等于 row[1]，row[0]“可用”；

2) 否则，如果 col[1]“可用”并且 row[1]“不可用”，则 row[0]等于 col[1]，row[0]“可用”；

3) 否则，如果 row[1]“可用”并且 col[1]“可用”，则 row[0]等于 row[1]，row[0]“可用”，否则 row[0]“不可用”。

9.7.1.4 亮度预测块普通帧内预测

9.7.1.4.1 概述

如果IntraPfFlag和IipFlag的值均为0，按9.7.1.4.2得到predMatrix；否则，如果IntraPfFlag的值为1，先按9.7.1.4.2得到orgPredMatrix，再按9.7.1.4.4得到predMatrix；否则，如果IipFlag的值为1，按9.7.1.4.3得到predMatrix。

9.7.1.4.2 预测方法

根据IntraLumaPredMode决定亮度编码块普通帧内预测方法如下：

a) IntraLumaPredMode 等于 12（Intra_Luma_Vertical 预测）。

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        orgPredMatrix[x][y] = r[x+1]
        predMatrix[x][y] = orgPredMatrix[x][y]
    }
}
```

b) IntraLumaPredMode 等于 24（Intra_Luma_Horizontal 预测）。

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        orgPredMatrix[x][y] = c[y+1]
        predMatrix[x][y] = orgPredMatrix[x][y]
    }
}
```

c) IntraLumaPredMode 等于 0（Intra_Luma_DC 预测）。

1) 如果 r[i]、c[j]（i=1～M,j=1～N）均“可用”，则：

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        ri = 0
```

```
        for (i=1; i<=M; i++) {
            ri += r[i]
        }
        cj = 0
        for (j=1; j<=N; j++) {
            cj += c[j]
        }
        orgPredMatrix[x][y] = ((ri + cj + ((M + N) >> 1)) * (4096 / (M + N))) >> 12
        predMatrix[x][y] = orgPredMatrix[x][y]
    }
}
```

2) 否则，如果 r[i]（i=1～M）均“可用”，且 c[j]（j=1～N）均“不可用”，则：

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        ri = 0
        for (i=1; i<=M; i++) {
            ri += r[i]
        }
        orgPredMatrix[x][y] = ((ri + (M >> 1)) >> Log(M)
        predMatrix[x][y] = orgPredMatrix[x][y]
    }
}
```

3) 否则，如果 c[j]（j=1～N）均“可用”，且 r[i]（i=1～M）均“不可用”，则：

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        cj = 0
        for (j=1; j<=N; j++) {
            cj += c[j]
        }
        orgPredMatrix[x][y] = ((cj + (N >> 1)) >> Log(N)
        predMatrix[x][y] = orgPredMatrix[x][y]
    }
}
```

4) 否则，

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        orgPredMatrix[x][y] = 2^BitDepth-1
        predMatrix[x][y] = orgPredMatrix[x][y]
    }
}
```

其中 BitDepth 是编码样本精度。

d) IntraLumaPredMode 等于 1（Intra_Luma_Plane 预测）。

```
ibMult[5] = {13, 17, 5, 11, 23}
ibShift[5] = {7, 10, 11, 15, 19}
imh = ibMult[Log(M) - 2]
ish = ibShift[Log(M) - 2]
imv = ibMult[Log(N) - 2]
isv = ibShift[Log(N) - 2]

ih = 0
for (i=0; i<(M>>1); i++) {
    ih += (i + 1) * (r[(M >> 1) + 1 + i] - r[(M >> 1) - 1 - i])
}
iv = 0
for (i=0; i<(N>>1); i++) {
    iv += (i + 1) * (c[(N >> 1) + 1 + i] - c[(N >> 1) - 1 - i])
}

ia = (r[M] + c[N]) << 4
ib = ((ih << 5) * imh + (1 << (ish - 1))) >> ish
ic = ((iv << 5) * imv + (1 << (isv - 1))) >> isv
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        orgPredMatrix[x][y] = (ia + (x - ((M >> 1) - 1)) * ib + (y - ((N >> 1) - 1)) * ic + 16) >> 5
        predMatrix[x][y] = Clip1(orgPredMatrix[x][y])
    }
}
```

e) IntraLumaPredMode 等于 2（Intra_Luma_Bilinear 预测）。

```
bilinearWeight[3] = {21, 13, 7}
weight = bilinearWeight[Log(Max(M, N) / Min(M, N)) - 1]
ishift = Log(Min(M, N))

ia = r[M]
ib = c[N]
if (M == N) {
    ic = (ia + ib + 1) >> 1
}
else {
    ic = (((ia << Log(M)) + (ib << Log(N))) * weight + (1 << (ishift + 5))) >> (ishift + 6)
}
for (x=0; x<M; x++) {
```

```
    for (y=0; y<N; y++) {
        orgPredMatrix[x][y] = ((((ia - c[y+1]) * (x + 1)) << Log(N)) + (((ib - r[x+1]) * (y + 1)) <<  Log(M))
+ ((r[x+1] + c[y+1]) << (Log(M) + Log(N))) + ((ic << 1) - ia - ib) * x * y + (1 << (Log(M) + Log(N)))) >>
(Log(M) + Log(N) + 1)
        predMatrix[x][y] = Clip1(orgPredMatrix[x][y])
    }
}
```

f) IntraLumaPredMode 不等于 0、1、2、12、24（Intra_Luma_Angular 预测）。

1) 初始化，首先执行以下操作，然后令x从0到M，y从0到N，依次执行步骤2）到步骤5）。

```
r[-1] = c[1]
r[-2] = c[2]
c[-1] = r[1]
c[-2] = r[2]
filterBit[4] = {7, 7, 7, 6}
filterOffset[4] = {64, 64, 64, 32}
xyAxis = dirXYFlag[IntraLumaPredMode]
xySign = dirXYSign[IntraLumaPredMode]
imx = divDxy[IntraLumaPredMode][0]
isx = divDxy[IntraLumaPredMode][1]
imy = divDyx[IntraLumaPredMode][0]
isy = divDyx[IntraLumaPredMode][1]
```

其中xyAxis、xySign、imx、isx、imy和isy由IntraLumaPredMode查表101得到。

2) 如果xySign小于0，则：

```
if (xyAxis == 0) { /* 参考上边像素 */
    offset = (((y + 1) * imx * 32) >> isx)  -  (((y + 1) * imx) >> isx) * 32
    iX = x + (((y + 1) * imx) >> isx)
    iY = -1
    if (y == 0) {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (M * N <= 64  ? 2  : 1)  : 0
    }
}
else { /* xyAxis == 1, 参考左边像素 */
    offset = (((x + 1) * imy * 32) >> isy)  -  (((x + 1) * imy) >> isy) * 32
    iX = -1
    iY = y + (((x + 1) * imy) >> isy)
    if (x == 0) {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 3 : 2) : 0
```

```
    }
    else {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 2 : 1) : 0
    }
}
```

3) 如果xySign大于0，则：

```
offsetx = (((y + 1) * imx * 32) >> isx) - (((y + 1) * imx) >> isx) * 32
iXx = x - (((y + 1) * imx) >> isx)
offsety = (((x + 1) * imy * 32) >> isy) - (((x + 1) * imy) >> isy) * 32
iYy = y - (((x + 1) * imy) >> isy)
if (iYy <= -1) { /* 参考上边像素 */
    offset = offsetx
    iX = iXx
    iY = -1
}
else { /* 参考左边像素 */
    offset = offsety
    iX = -1
    iY = iYy
}
if (xyAxis == 0) {
    if (y == 0) {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 2 : 1) : 0
    }
}
else { /* xyAxis == 1 */
    if (x == 0) {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 2 : 1) : 0
    }
}
```

4) 如果iY等于-1（参考上边像素）：

```
if (xySign < 0) {
    iXn = iX + 1
    iXnP2 = iX + 2
```

```
    iXnN1 = iX  -  1
}
else {
    iXn = iX  -  1
    iXnP2 = iX  -  2
    iXnN1 = iX + 1
}
orgPredMatrix[x][y] = (r[iXnN1+1] * filter[0] + r[iX+1] * filter[1] + r[iXn+1] * filter[2] + r[iXnP2+1] *
filter[3] + filterOffset[filterIndex]) >> filterBit[filterIndex]
if (MipfEnableFlag == 0)
    predMatrix[x][y] = orgPredMatrix[x][y]
else
    predMatrix[x][y] = Clip1( orgPredMatrix[x][y] )
```

其中filter[0]到filter[3]由filterIndex和offset查表102得到。

5) 否则，如果iX等于-1（参考左边像素）：

```
if (xySign < 0) {
    iYn = iY + 1
    iYnP2 = iY + 2
    iYnN1 = iY  -  1
}
else {
    iYn = iY  -  1
    iYnP2 = iY  -  2
    iYnN1 = iY + 1
}
orgPredMatrix[x][y] = (c[iYnN1+1] * filter[0] + c[iY+1] * filter[1] + c[iYn+1] * filter[2] + c[iYnP2+1] *
filter[3] + filterOffset[filterIndex]) >> filterBit[filterIndex]
if (MipfEnableFlag == 0)
    predMatrix[x][y] = orgPredMatrix[x][y]
else
    predMatrix[x][y] = Clip1( orgPredMatrix[x][y] )
```

其中filter[0]到filter[3]由filterIndex和offset查表102得到。

g) IntraLumaPredMode 等于‘Intra_Luma_PCM’

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        predMatrix[x][y] = 0
    }
}
```

表101　角度预测模式的方向

IntraLumaPredMode	dirXYFlag[IntraLumaPredMode]	dirXYSign[IntraLumaPredMode]	divDxy[IntraLumaPredMode]	divDyx[IntraLumaPredMode]
0	—	—	—	—
1	—	—	—	—
2	—	—	—	—
3	0	-1	{11,2}	{93,8}
4	0	-1	{2,0}	{1,1}
5	0	-1	{11,3}	{93,7}
6	0	-1	{1,0}	{1,0}
7	0	-1	{93,7}	{11,3}
8	0	-1	{1,1}	{2,0}
9	0	-1	{93,8}	{11,2}
10	0	-1	{1,2}	{4,0}
11	0	-1	{1,3}	{8,0}
12	—	—	—	—
13	0	1	{1,3}	{8,0}
14	0	1	{1,2}	{4,0}
15	0	1	{93,8}	{11,2}
16	0	1	{1,1}	{2,0}
17	0	1	{93,7}	{11,3}
18	1	1	{1,0}	{1,0}
19	1	1	{11,3}	{93,7}
20	1	1	{2,0}	{1,1}
21	1	1	{11,2}	{93,8}
22	1	1	{4,0}	{1,2}
23	1	1	{8,0}	{1,3}
24	—	—	—	—
25	1	-1	{8,0}	{1,3}
26	1	-1	{4,0}	{1,2}
27	1	-1	{11,2}	{93,8}
28	1	-1	{2,0}	{1,1}
29	1	-1	{11,3}	{93,7}
30	1	-1	{1,0}	{1,0}
31	1	-1	{93,7}	{11,3}
32	1	-1	{1,1}	{2,0}
33	—	—	—	—
34	0	-1	{585,8}	{7,4}
35	0	-1	{205,7}	{5,3}
36	0	-1	{73,6}	{449,9}
37	0	-1	{449,9}	{73,6}

表 101（续）

IntraLumaPredMode	dirXYFlag[IntraLumaPredMode]	dirXYSign[IntraLumaPredMode]	divDxy[IntraLumaPredMode]	divDyx[IntraLumaPredMode]
38	0	-1	{5,3}	{205,7}
39	0	-1	{7,4}	{585,8}
40	0	-1	{5,4}	{819,8}
41	0	-1	{3,4}	{1365,8}
42	0	-1	{1,4}	{16,0}
43	0	-1	{1,5}	{32,0}
44	0	1	{1,5}	{32,0}
45	0	1	{1,4}	{16,0}
46	0	1	{3,4}	{1365,8}
47	0	1	{5,4}	{819,8}
48	0	1	{7,4}	{585,8}
49	0	1	{5,3}	{205,7}
50	0	1	{449,9}	{73,6}
51	1	1	{73,6}	{449,9}
52	1	1	{205,7}	{5,3}
53	1	1	{585,8}	{7,4}
54	1	1	{819,8}	{5,4}
55	1	1	{1365,8}	{3,4}
56	1	1	{16,0}	{1,4}
57	1	1	{32,0}	{1,5}
58	1	-1	{32,0}	{1,5}
59	1	-1	{16,0}	{1,4}
60	1	-1	{1365,8}	{3,4}
61	1	-1	{819,8}	{5,4}
62	1	-1	{585,8}	{7,4}
63	1	-1	{205,7}	{5,3}
64	1	-1	{73,6}	{449,9}
65	1	-1	{449,9}	{73,6}

表102　参考像素滤波器系数

filterIndex	0				1				2				3			
offset	filter[0]	filter[1]	filter[2]	filter[3]	filter[0]	filter[1]	filter[2]	filter[3]	filter[0]	filter[1]	filter[2]	filter[3]	filter[0]	filter[1]	filter[2]	filter[3]
0	32	64	32	0	23	82	21	2	0	11	106	12	-1	0	64	0
1	31	63	33	1	21	82	23	2	1	9	105	15	-1	-1	63	2
2	30	62	34	2	19	82	25	2	2	7	105	18	-2	-2	62	4
3	29	61	35	3	18	81	27	2	3	6	103	21	-2	-2	60	7
4	28	60	36	4	16	80	30	2	4	4	102	25	-3	-2	58	10
5	27	59	37	5	15	79	32	2	5	2	100	29	-3	-3	57	12
6	26	58	38	6	13	78	35	2	6	1	99	32	-4	-4	56	14
7	25	57	39	7	12	77	37	2	7	0	96	36	-4	-4	55	15
8	24	56	40	8	11	75	40	2	8	-1	94	39	-4	-4	54	16
9	23	55	41	9	10	74	42	2	9	-2	92	43	-5	-5	53	18
10	22	54	42	10	9	72	45	2	10	-3	89	47	-5	-6	52	20
11	21	53	43	11	8	70	47	3	11	-4	86	51	-5	-6	49	24
12	20	52	44	12	7	68	50	3	12	-4	83	55	-6	-6	46	28
13	19	51	45	13	6	67	52	3	13	-5	80	59	-6	-5	44	29
14	18	50	46	14	6	64	54	4	14	-5	77	62	-6	-4	42	30
15	17	49	47	15	5	62	57	4	15	-6	74	66	-6	-4	39	33
16	16	48	48	16	4	60	60	4	16	-6	70	70	-6	-4	36	36
17	15	47	49	17	4	57	62	5	17	-6	66	74	-6	-4	33	39
18	14	46	50	18	4	54	64	6	18	-6	62	77	-5	-4	30	42
19	13	45	51	19	3	52	67	6	19	-6	59	80	-5	-4	29	44
20	12	44	52	20	3	50	68	7	20	-6	55	83	-4	-4	28	46
21	11	43	53	21	3	47	70	8	21	-5	51	86	-4	-3	24	49

表 102（续）

filterIndex	0				1				2				3			
offset	filter[0]	filter[1]	filter[2]	filter[3]	filter[0]	filter[1]	filter[2]	filter[3]	filter[0]	filter[1]	filter[2]	filter[3]	filter[0]	filter[1]	filter[2]	filter[3]
22	10	42	54	22	2	45	72	9	22	-5	47	89	-3	-2	20	52
23	9	41	55	23	2	42	74	10	23	-5	43	92	-2	-2	18	53
24	8	40	56	24	2	40	75	11	24	-4	39	94	-1	-2	16	54
25	7	39	57	25	2	37	77	12	25	-4	36	96	0	-2	15	55
26	6	38	58	26	2	35	78	13	26	-4	32	99	1	-2	14	56
27	5	37	59	27	2	32	79	15	27	-3	29	100	2	-2	12	57
28	4	36	60	28	2	30	80	16	28	-3	25	102	4	-2	10	58
29	3	35	61	29	2	27	81	18	29	-2	21	103	6	-1	7	60
30	2	34	62	30	2	25	82	19	30	-2	18	105	7	0	4	62
31	1	33	63	31	2	23	82	21	31	-1	15	105	9	0	2	63

9.7.1.4.3 改进型预测方法

第1步，初始化。

```
r[-1] = c[1]
r[-2] = c[2]
r[-3] = c[3]
r[-4] = c[4]
c[-1] = r[1]
c[-2] = r[2]
c[-3] = r[3]
c[-4] = r[4]
```

第2步，根据IntraLumaPredMode决定亮度编码块改进型普通帧内预测方法如下：

a) IntraLumaPredMode 等于 12（Intra_Luma_Vertical 预测）。

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        orgPredMatrix[x][y] = (21 * r[x-1] + 63 * r[x] + 88 * r[x+1] + 63 * r[x+2] + 21 * r[x+3]) >> 8
        predMatrix[x][y] = Clip1(((r[x+d+1] + r[x-d+1] + c[y+d+1] + c[y-d+1]) * w0 + orgPredMatrix[x][y]
* w1 + 128) >> 8)
    }
}
```

b) IntraLumaPredMode 等于 24（Intra_Luma_Horizontal 预测）。

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        orgPredMatrix[x][y] = (21 * c[y-1] + 63 * c[y] + 88 * c[y+1] + 63 * c[y+2] + 21 * c[y+3]) >> 8
        predMatrix[x][y] = Clip1(((r[x+d+1] + r[x-d+1] + c[y+d+1] + c[y-d+1]) * w0 + orgPredMatrix[x][y]
* w1 + 128) >> 8)
    }
}
```

c) IntraLumaPredMode 等于 0（Intra_Luma_DC 预测）。

1) 如果 r[i]、c[j]（i=1～M, j=1～N）均“可用”，则 base 的值如下：

```
ri = 0
for (i=1; i<=M; i++) {
    ri += r[i]
}
cj = 0
for (j=1; j<=N; j++) {
    cj += c[j]
}
base = ((ri + cj + ((M + N) >> 1)) * (4096 / (M + N))) >> 12
```

2) 否则，如果 r[i]（i=1～M）均“可用”，且 c[j]（j=1～N）均“不可用”，则 base 的值如下：

```
ri = 0
for (i=1; i<=M; i++) {
    ri += r[i]
}
base = (ri + (M >> 1)) >> Log(M)
```

3) 否则，如果 c[j]（j=1～N）均“可用”，且 r[i]（i=1～M）均“不可用”，则 base 的值如下：

```
cj = 0
for (j=1; j<=N; i++) {
    cj += c[j]
}
base = (cj + (N >> 1)) >> Log(N)
```

4) 否则，base 的值如下：

$$base = 2^{BitDepth-1}$$

其中 BitDepth 是编码样本精度。

5) 导出 orgPredMatrix。

```
orgPredMatrix[0][0] = (base * 126 + (c[0] + c[2] + r[2]) * 18 + (c[1]+r[1]) * 38 + 128) >> 8
orgPredMatrix[1][0] = (base * 169 + (r[1] + r[3]) * 18 + r[2] * 38 + c[1] * 13 + 128) >> 8
orgPredMatrix[0][1] = (base * 169 + (c[1] + c[3]) * 18 + c[2] * 38 + r[1] * 13 + 128) >> 8
orgPredMatrix[1][1] = (base * 230 + r[2] * 13 + c[2] * 13 + 128) >> 8
for (x=2; x<M; x++) {
    for (y=2; y<N; y++) {
        orgPredMatrix[x][0] = (base * 182 + (r[x] + r[x+2]) * 18 + r[x+1] * 38 + 128) >> 8
        orgPredMatrix[0][y] = (base * 182 + (c[y] + c[y+2]) * 18 + c[y+1] * 38 + 128) >> 8
        orgPredMatrix[x][1] = (base * 243 + r[x+1] * 13 + 128) >> 8
        orgPredMatrix[1][y] = (base * 243 + c[y+1] * 13 + 128) >> 8
        orgPredMatrix[x][y] = base
    }
}
```

6) 导出 predMatrix。

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        predMatrix[x][y] = Clip1(((r[x+d+1] + r[x-d+1] + c[y+d+1] + c[y-d+1]) * w0 + orgPredMatrix[x][y]
* w1 + 128) >> 8)
    }
}
```

d) IntraLumaPredMode 等于 1（Intra_Luma_Plane 预测）。

1) 导出 ia、ib、ic 和 base。

```
ibMult[5] = {13, 17, 5, 11, 23}
ibShift[5] = {7, 10, 11, 15, 19}
imh = ibMult[Log(M) - 2]
ish = ibShift[Log(M) - 2]
imv = ibMult[Log(N) - 2]
isv = ibShift[Log(N) - 2]

ih = 0
for (i=0; i<(M>>1); i++) {
    ih += (i + 1) * (r[(M >> 1) + 1 + i] - r[(M >> 1) - 1 - i])
}
iv = 0
for (i=0; i<(N>>1); i++) {
    iv += (i + 1) * (c[(N >> 1) + 1 + i] - c[(N >> 1) - 1 - i])
}

ia = (r[M] + c[N]) << 4
ib = ((ih << 5) * imh + (1 << (ish - 1))) >> ish
ic = ((iv << 5) * imv + (1 << (isv - 1))) >> isv
base= ia - ((M >> 1) - 1) * ib + ((N >> 1) - 1) * ic + 16
```

2) 导出 orgPredMatrix。

```
orgPredMatrix[0][0] = (((base * 126 + 69 * ib + 69 * ic) >> 5) + (c[0] + c[2] + r[2]) * 18 + (c[1]+r[1])
* 38 + 128) >> 8
orgPredMatrix[1][0] = (((base * 169 + 195 * ib + 87 * ic)  >> 5) + (r[1] + r[3]) * 18 + r[2] * 38 + c[1]
* 13 + 128) >> 8
orgPredMatrix[0][1] = (((base * 169 + 195 * ic + 87 * ib) >> 5) + (c[1] + c[3]) * 18 + c[2] * 38 + r[1] *
13 + 128) >> 8
orgPredMatrix[1][1] = (((base * 230 + 256 * ib + 256 * ic) >> 5) + r[2] * 13 + c[2] * 13 + 128) >> 8
for (x=2; x<M; x++) {
    for (y=2; y<N; y++) {
        orgPredMatrix[x][0] = (((base * 182 + x * ib * 182 + 87 * ic) >> 5) + (r[x] + r[x+2]) * 18 + r[x+1]
* 38 + 128) >> 8
        orgPredMatrix[0][y] = (((base * 182 + y * ic * 182 + 87 * ib) >> 5) + (c[x] + c[y+2]) * 18 + c[y+1]
* 38 + 128) >> 8
        orgPredMatrix[x][1] = (((base * 243 + x * ib * 243 + 269 * ic) >> 5) + r[x+1] * 13 + 128) >> 8
        orgPredMatrix[1][y] = (((base * 243 + y * ic * 243 + 269 * ib) >> 5) + c[y+1] * 13 + 128) >> 8
        orgPredMatrix[x][y] = (base + x * ib + y * ic) >> 5
    }
}
```

3) 导出 predMatrix[x][y]。

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        predMatrix[x][y] = Clip1((r[x+d+1] + r[x-d+1] + c[y+d+1] + c[y-d+1]) * w0 + orgPredMatrix[x][y] *
w1 + 128) >> 8)
    }
}
```

e) IntraLumaPredMode 等于 2（Intra_Luma_Bilinear 预测）。

```
bilinearWeight[3] = {21, 13, 7}
weight = bilinearWeight[Log(Max(M, N) / Min(M, N)) - 1]
ishift = Log(Min(M, N))

ia = r[M]
ib = c[N]
if (M == N) {
    ic = (ia + ib + 1) >> 1
}
else {
    ic = (((ia << Log(M)) + (ib << Log(N))) * weight + (1 << (ishift + 5))) >> (ishift + 6)
}
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        orgPredMatrix[x][y] = (((((ia<<10) - (82 * c[y-1] + 252 * c[y] + 356 * c[y+1] + 252 * c[y+2] + 82
* c[y+3])) * (x+1)) << Log(N)) + ((((ib<<10) - (82 * r[x-1] + 252 * r[x] + 356 * r[x+1] + 252 * r[x+2] +
82 * r[x+3])) * (y+1)) << Log(M)) + ((82 * r[x-1] + 252 * r[x] + 356 * r[x+1] + 252 * r[x+2] + 82 * r[x+3]
+ 82 * c[y-1] + 252 * c[y] + 356 * c[y+1] + 252 * c[y+2] + 82 * c[y+3]) << (Log(M) + Log(N))) + (((ic<<1)
 - ia - ib) * x * y<<10) + (1 << (Log(M) + Log(N) + 10))) >> (Log(M) + Log(N) + 11)
        predMatrix[x][y] = Clip1((orgPredMatrix[x][y] * w1 + (r[x+d+1] + r[x-d+1] + c[y+d+1] + c[y-d+1])
* w0 + 128) >> 8)
    }
}
```

f) IntraLumaPredMode 不等于 0、1、2、12、24（Intra_Luma_Angular 预测）。

1) 初始化，首先执行以下操作，然后令x从0到M，y从0到N，依次执行步骤2）到步骤5）。

```
filterBitshiftLong[4] = {10, 10, 10, 9}
filterOffsetLong[4] = {512, 512, 512, 256}
xyAxis = dirXYFlag[IntraLumaPredMode]
xySign = dirXYSign[IntraLumaPredMode]
imx = divDxy[IntraLumaPredMode][0]
isx = divDxy[IntraLumaPredMode][1]
imy = divDyx[IntraLumaPredMode][0]
isy = divDyx[IntraLumaPredMode][1]
```

其中xyAxis、xySign、imx、isx、imy和isy由IntraLumaPredMode查表101得到。

2) 如果xySign小于0，则：

```
if (xyAxis == 0) { /* 参考上边像素 */
    offset = (((y + 1) * imx * 32) >> isx) - (((y + 1) * imx) >> isx) * 32
    iX = x + (((y + 1) * imx) >> isx)
    iY = -1
    if (y == 0) {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (M * N <= 64  ? 2  : 1)  : 0
    }
}
else { /* xyAxis == 1, 参考左边像素 */
    offset = (((x + 1) * imy * 32) >> isy) - (((x + 1) * imy) >> isy) * 32
    iX = -1
    iY = y + (((x + 1) * imy) >> isy)
    if (x == 0) {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 2 : 1) : 0
    }
}
```

3) 如果xySign大于0，则：

```
offsetx = (((y + 1) * imx * 32) >> isx) - (((y + 1) * imx) >> isx) * 32
iXx = x - (((y + 1) * imx) >> isx)
offsety = (((x + 1) * imy * 32) >> isy) - (((x + 1) * imy) >> isy) * 32
iYy = y - (((x + 1) * imy) >> isy)
if (iYy <= -1) { /* 参考上边像素 */
    offset = offsetx
    iX = iXx
    iY = -1
}
else { /* 参考左边像素 */
    offset = offsety
    iX = -1
    iY = iYy
}
if (xyAxis == 0) {
    if (y == 0) {
```

```
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 2 : 1) : 0
    }
}
else { /* xyAxis == 1 */
    if (x == 0) {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (M * N <= 64 ? 2 : 1) : 0
    }
}
```

4) 如果iY等于-1（参考上边像素）：

```
if (xySign < 0) {
    iXn = iX + 1
    iXnP2 = iX + 2
    iXnP3 = iX + 3
    iXnP4 = iX + 4
    iXnN1 = iX - 1
    iXnN2 = iX - 2
    iXnN2 = iX - 2
}
else {
    iXn = iX - 1
    iXnP2 = iX - 2
    iXnP3 = iX - 3
    iXnP4 = iX - 4
    iXnN1 = iX + 1
    iXnN2 = iX + 2
    iXnN3 = iX + 3
}
orgPredMatrix[x][y] = (r[iXnN3+1] * filter[0] + r[iXnN2+1] * filter[1] + r[iXnN1+1] * filter[2] + r[iX+1]
* filter[3] + r[iXn+1] * filter[4] + r[iXnP2+1] * filter[5] + r[iXnP3+1] * filter[6] + r[iXnP4+1] * filter[7]
+ filterOffsetLong[filterIndex]) >> filterBitshiftLong[filterIndex]
predMatrix[x][y] = Clip1(((r[x+d+1] + r[x-d+1] + c[y+d+1] + c[y-d+1]) * w0 + orgPredMatrix[x][y] * w1 + 128) >>
8)
```

其中filter[0]到filter[7]由filterIndex和offset查表103～表106得到。

5) 否则，如果 iX等于-1（参考左边像素）：

```
if (xySign < 0) {
    iYn = iY + 1
    iYnP2 = iY + 2
    iYnP3 = iY + 3
    iYnP4 = iY + 4
    iYnN1 = iY - 1
    iYnN2 = iY - 2
    iYnN3 = iY - 3
}
else {
    iYn = iY - 1
    iYnP2 = iY - 2
    iYnP3 = iY - 3
    iYnP4 = iY - 4
    iYnN1 = iY + 1
    iYnN2 = iY + 2
    iYnN3 = iY + 3
}
orgPredMatrix[x][y] = (c[iYnN3+1] * filter[0] + c[iYnN2+1] * filter[1] + c[iYnN1+1] * filter[2] + c[iY+1]
* filter[3] + c[iYn+1] * filter[4] + c[iYnP2+1] * filter[5] + c[iYnP3+1] * filter[6] + c[iYnP4+1] * filter[7]
+ filterOffsetLong[filterIndex]) >> filterBitLong[filterIndex]
predMatrix[x][y] = Clip1(((r[x+d+1] + r[x-d+1] + c[y+d+1] + c[y-d+1]) * w0 + orgPredMatrix[x][y] * w1 + 128) >>
8)
```

其中filter[0]到filter[7]由filterIndex和offset查表103～表106得到。

g) IntraLumaPredMode 等于‘Intra_Luma_PCM’

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        predMatrix[x][y] = 0
    }
}
```

上述改进型预测过程中，如果x或y等于0，则d等于1；否则d等于2。如果x或y小于2，则w0等于10，w1等于216；否则w0等于13，w1等于204。其中x的取值范围为0～M-1，y的取值范围为0～N-1。

表103 参考像素滤波器系数（filterIndex 等于 0）

offset	filter[0]	filter[1]	filter[2]	filter[3]	filter[4]	filter[5]	filter[6]	filter[7]
0	46	145	210	222	210	145	46	0
1	44	142	208	221	211	147	49	2
2	43	139	206	221	211	149	52	3
3	41	136	204	221	212	151	55	4
4	40	133	202	220	212	153	58	6

表 103（续）

offset	filter[0]	filter[1]	filter[2]	filter[3]	filter[4]	filter[5]	filter[6]	filter[7]
5	39	130	200	220	212	155	61	7
6	37	126	198	220	213	157	64	9
7	36	123	196	219	213	160	67	10
8	34	120	194	219	213	162	71	11
9	33	117	192	218	214	163	74	13
10	31	114	190	218	214	166	77	14
11	30	111	188	218	214	167	80	16
12	28	108	186	217	215	170	83	17
13	27	105	184	217	215	172	86	18
14	26	101	182	217	215	174	89	20
15	24	99	180	216	216	176	92	21
16	23	95	178	216	216	178	95	23
17	21	92	176	216	216	180	99	24
18	20	89	174	215	217	182	101	26
19	18	86	172	215	217	184	105	27
20	17	83	170	215	217	186	108	28
21	16	80	167	214	218	188	111	30
22	14	77	166	214	218	190	114	31
23	13	74	163	214	218	192	117	33
24	11	71	162	213	219	194	120	34
25	10	67	160	213	219	196	123	36
26	9	64	157	213	220	198	126	37
27	7	61	155	212	220	200	130	39
28	6	58	153	212	220	202	133	40
29	4	55	151	212	221	204	136	41
30	3	52	149	211	221	206	139	43
31	2	49	147	211	221	208	142	44

表104　参考像素滤波器系数（filterIndex 等于 1）

offset	filter[0]	filter[1]	Filter[2]	filter[3]	filter[4]	filter[5]	filter[6]	filter[7]
0	33	156	209	223	211	156	33	3
1	30	153	208	223	212	159	36	3
2	27	149	208	223	213	162	39	3
3	26	146	207	223	213	164	42	3
4	23	141	206	223	214	168	46	3
5	21	138	205	223	215	170	49	3
6	18	133	204	223	216	174	53	3
7	17	130	204	222	216	176	56	3

表 104（续）

offset	filter[0]	filter[1]	Filter[2]	filter[3]	filter[4]	filter[5]	filter[6]	filter[7]
8	16	125	203	222	217	178	60	3
9	14	123	202	222	217	180	63	3
10	13	118	201	222	218	182	67	3
11	12	113	199	222	218	184	72	4
12	10	109	198	221	219	187	76	4
13	9	106	198	221	219	188	79	4
14	9	101	195	221	219	189	84	6
15	7	97	195	220	220	191	88	6
16	6	92	194	220	220	194	92	6
17	6	88	191	220	220	195	97	7
18	6	84	189	219	221	195	101	9
19	4	79	188	219	221	198	106	9
20	4	76	187	219	221	198	109	10
21	4	72	184	218	222	199	113	12
22	3	67	182	218	222	201	118	13
23	3	63	180	217	222	202	123	14
24	3	60	178	217	222	203	125	16
25	3	56	176	216	222	204	130	17
26	3	53	174	216	223	204	133	18
27	3	49	170	215	223	205	138	21
28	3	46	168	214	223	206	141	23
29	3	42	164	213	223	207	146	26
30	3	39	162	213	223	208	149	27
31	3	36	159	212	223	208	153	30

表105　参考像素滤波器系数（filterIndex 等于 2）

offset	filter[0]	filter[1]	filter[2]	filter[3]	filter[4]	filter[5]	filter[6]	filter[7]
0	16	170	215	226	214	169	15	-1
1	13	165	214	226	215	173	20	-2
2	10	162	215	226	216	176	22	-3
3	8	157	214	226	216	179	27	-3
4	6	152	214	226	217	182	31	-4
5	3	146	213	226	218	186	36	-4
6	2	143	214	226	218	188	39	-6
7	0	137	213	226	219	190	45	-6
8	-1	132	212	226	220	192	49	-6
9	-3	128	213	226	220	194	53	-7
10	-4	122	211	225	221	197	59	-7

表 105（续）

offset	filter[0]	filter[1]	filter[2]	filter[3]	filter[4]	filter[5]	filter[6]	filter[7]
11	-6	116	210	225	222	200	64	-7
12	-6	112	211	225	222	200	68	-8
13	-7	106	210	225	222	203	74	-9
14	-7	101	209	225	223	204	78	-9
15	-9	96	208	224	224	206	84	-9
16	-9	90	207	224	224	207	90	-9
17	-9	84	206	224	224	208	96	-9
18	-9	78	204	223	225	209	101	-7
19	-9	74	203	222	225	210	106	-7
20	-8	68	200	222	225	211	112	-6
21	-7	64	200	222	225	210	116	-6
22	-7	59	197	221	225	211	122	-4
23	-7	53	194	220	226	213	128	-3
24	-6	49	192	220	226	212	132	-1
25	-6	45	190	219	226	213	137	0
26	-6	39	188	218	226	214	143	2
27	-4	36	186	218	226	213	146	3
28	-4	31	182	217	226	214	152	6
29	-3	27	179	216	226	214	157	8
30	-3	22	176	216	226	215	162	10
31	-2	20	173	215	226	214	165	13

表106　参考像素滤波器系数（filterIndex 等于 3）

offset	filter[0]	filter[1]	filter[2]	filter[3]	filter[4]	filter[5]	filter[6]	filter[7]
0	0	91	108	114	108	91	0	0
1	-1	88	107	114	108	93	3	0
2	-3	85	106	114	109	95	6	0
3	-3	82	107	114	109	96	8	-1
4	-3	79	109	114	109	96	11	-3
5	-4	76	108	114	109	98	14	-3
6	-6	73	107	114	110	100	17	-3
7	-6	72	107	114	110	100	18	-3
8	-6	70	107	114	110	100	20	-3
9	-7	67	106	114	111	102	22	-3
10	-8	64	106	113	111	104	25	-3
11	-9	60	106	113	112	105	29	-4
12	-9	56	107	113	112	106	33	-6
13	-7	54	107	113	112	104	35	-6

表 106（续）

offset	filter[0]	filter[1]	filter[2]	filter[3]	filter[4]	filter[5]	filter[6]	filter[7]
14	-6	53	107	113	112	103	36	-6
15	-6	49	106	113	112	104	40	-6
16	-6	45	105	112	112	105	45	-6
17	-6	40	104	112	113	106	49	-6
18	-6	36	103	112	113	107	53	-6
19	-6	35	104	112	113	107	54	-7
20	-6	33	106	112	113	107	56	-9
21	-4	29	105	112	113	106	60	-9
22	-3	25	104	111	113	106	64	-8
23	-3	22	102	111	114	106	67	-7
24	-3	20	100	110	114	107	70	-6
25	-3	18	100	110	114	107	72	-6
26	-3	17	100	110	114	107	73	-6
27	-3	14	98	109	114	108	76	-4
28	-3	11	96	109	114	109	79	-3
29	-1	8	96	109	114	107	82	-3
30	0	6	95	109	114	106	85	-3
31	0	3	93	108	114	107	88	-1

9.7.1.4.4 亮度帧内预测滤波

亮度帧内预测滤波过程如下：

```
predMatrixIPF[-1][-1] = 0
for (x=0; x<M; x++) {
    predMatrixIPF[x][-1] = r[x+1]
}
for (y=0; y<N; y++) {
    predMatrixIPF[-1][y] = c[y+1]
}
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        predMatrixIPF[x][y] = orgPredMatrix[x][y]
    }
}
if ((IntraLumaPredMode == 0) || (IntraLumaPredMode == 1) || (IntraLumaPredMode == 2)) {
    for (x=0; x<M; x++) {
        for (y=0; y<N; y++) {
            predMatrix[x][y] = Clip1((f[x] * predMatrixIPF[-1][y] + f[y] * predMatrixIPF[x][-1] + (64 -
f[x] - f[y]) * predMatrixIPF[x][y] + 32 ) >> 6)
```

```
        }
    }
}
else if (((IntraLumaPredMode >= 3) && (IntraLumaPredMode <= 18)) || ((IntraLumaPredMode >= 34) &&
(IntraLumaPredMode <= 50))) {
    for (x=0; x<M; x++) {
        for (y=0; y<N; y++) {
            predMatrix[x][y] = Clip1((f[x] * predMatrixIPF[-1][y] + (64- f[x]) * predMatrixIPF[x][y] + 32) >>
6)
        }
    }
}
else if (((IntraLumaPredMode >= 19) && (IntraLumaPredMode <= 32)) || (IntraLumaPredMode >= 51) &&
(IntraLumaPredMode <= 65)) {
    for (x=0; x<M; x++) {
        for (y=0; y<N; y++) {
            predMatrix[x][y] = Clip1((f[y] * predMatrixIPF[x][-1] + (64 - f[y]) * predMatrixIPF[x][y] +
32) >> 6)
        }
    }
}
```

其中，predMatrixIPF是(M+1)×(N+1)大小的样本矩阵，M×N块内坐标为(x,y)的像素对应的滤波系数f[x]和f[y]分别由M、x和N、y查表107得到。

表107　帧内预测边界滤波系数表

x 或 y	预测块的宽度或高度				
	4	8	16	32	64
0	24	44	40	36	52
1	6	25	27	27	44
2	2	14	19	21	37
3	0	8	13	16	31
4	0	4	9	12	26
5	0	2	6	9	22
6	0	1	4	7	18
7	0	1	3	5	15
8	0	0	2	4	13
9	0	0	1	3	11
10～63	0	0	0	0	0

9.7.1.5　色度预测块普通帧内预测

9.7.1.5.1　概述

如果IntraPfChromaEnableFlag的值为1且CodingTreeComponent为‘COMPONENT_LUMACHROMA’且IntraPfFlag的值为1，先根据9.7.1.5.2得到orgPredMatrix，再根据9.7.1.5.3得到predMatrix；否则，根据9.7.1.5.2得到predMatrix。

9.7.1.5.2 预测方法

根据IntraChromaPredMode决定色度编码块普通帧内预测方法如下：

a) IntraChromaPredMode 等于 0（Intra_Chroma_DM 和 Intra_Chroma_PCM 预测）。
   1) 如果 IntraLumaPredMode 等于 12，则 IntraChromaPredMode 的值为 3，转到步骤 d)。
   2) 如果 IntraLumaPredMode 等于 24，则 IntraChromaPredMode 的值为 2，转到步骤 c)。
   3) 如果 IntraLumaPredMode 等于 0，则 IntraChromaPredMode 的值为 1，转到步骤 b)。
   4) 如果 IntraLumaPredMode 等于 2，则 IntraChromaPredMode 的值为 4，转到步骤 e)。
   5) 如果 IntraLumaPredMode 等于 1，则：

```
ibMult[5] = {13, 17, 5, 11, 23}
ibShift[5] = {7, 10, 11, 15, 19}
imh = ibMult[Log(K) - 2]
ish = ibShift[Log(K) - 2]
imv = ibMult[Log(L) - 2]
isv = ibShift[Log(L) - 2]

ih = 0
for (i=0; i<(K>>1); i++) {
    ih += (i + 1) * (row[(K >> 1) + 1 + i] - row[(K >> 1) - 1 - i])
}
iv = 0
for (i=0; i<(L>>1); i++) {
    iv += (i + 1) * (col[(L >> 1) + 1 + i] - col[(L >> 1) - 1 - i])
}

ia = (row[K] + col[L]) << 4
ib = ((ih << 5) * imh + (1 << (ish - 1))) >> ish
ic = ((iv << 5) * imv + (1 << (isv - 1))) >> isv
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        predMatrix[x][y] = Clip1((ia + (x-((K>>1)-1)) * ib + (y - ((L>>1) - 1)) * ic + 16) >> 5)
    }
}
```

   6) 如果 IntraLumaPredMode 等于 33，则：

```
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        predMatrix[x][y] = 0
    }
}
```

7） 如果 IntraLumaPredMode 不等于 0、1、2、12、24、33。

- 初始化：

```
row[-1] = col[1]
row[-2] = col[2]
col[-1] = row[1]
col[-2] = row[2]
filterBit[4] = {7, 7, 7, 6}
filterOffset[4] = {64, 64, 64, 32}
xyAxis = dirXYFlag[IntraLumaPredMode]
xySign = dirXYSign[IntraLumaPredMode]
imx = divDxy[IntraLumaPredMode][0]
isx = divDxy[IntraLumaPredMode][1]
imy = divDyx[IntraLumaPredMode][0]
isy = divDyx[IntraLumaPredMode][1]
```

- 如果 xySign 小于 0，则：

```
if (xyAxis == 0) { /* 参考上边像素 */
    offset = (((y + 1) * imx * 32) >> isx)  -  (((y + 1) * imx) >> isx) * 32
    iX = x + (((y + 1) * imx) >> isx)
    iY = -1
    if (y <=1) {
        filterIndex = MipfEnableFlag ? (K * L <= 32 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (K * L <= 32  ? 2  : 1)  : 0
    }
}
else { /* xyAxis == 1, 参考左边像素 */
    offset = (((x + 1) * imy * 32) >> isy)  -  (((x + 1) * imy) >> isy) * 32
    iX = -1
    iY = y + (((x + 1) * imy) >> isy)
    if (x <=1) {
        filterIndex = MipfEnableFlag ? (K * L <= 32 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (K * L <= 32 ? 2 : 1) : 0
    }
}
```

- 如果 xySign 大于 0，则：

```
offsetx = (((y + 1) * imx * 32) >> isx)  -  (((y + 1) * imx) >> isx) * 32
iXx = x  -  (((y + 1) * imx) >> isx)
```

```
offsety = (((x + 1) * imy * 32) >> isy)  -  (((x + 1) * imy) >> isy) * 32
iYy = y  -  (((x + 1) * imy) >> isy)
if (iYy <= -1) { /* 参考上边像素 */
    offset = offsetx
    iX = iXx
    iY = -1
}
else { /* 参考左边像素 */
    offset = offsety
    iX = -1
    iY = iYy
}
if (xyAxis == 0) {
    if (y <= 1) {
        filterIndex = MipfEnableFlag ? (K * L <= 32 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (K * L <= 32 ? 2 : 1) : 0
    }
}
else { /* xyAxis == 1 */
    if (x <= 1) {
        filterIndex = MipfEnableFlag ? (K * L <= 32 ? 3 : 2) : 0
    }
    else {
        filterIndex = MipfEnableFlag ? (K * L <= 32 ? 2 : 1) : 0
    }
}
```

- 如果 iY 等于-1（参考上边像素）：

```
if (dx * dy < 0) {
    iXn = iX + 1
    iXnP2 = iX + 2
    iXnN1 = iX  -  1
}
else {
    iXn = iX  -  1
    iXnP2 = iX  -  2
    iXnN1 = iX + 1
}
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
```

```
        orgPredMatrix[x][y] = (row[iXnN1+1] * filter[0] + row[iX+1] * filter[1] + row[iXn+1] * filter[2]
+ row[iXnP2+1] * filter[3] + filterOffset[filterIndex]) >> filterBit[filterIndex]
        if (MipfEnableFlag == 0)
            predMatrix[x][y] = orgPredMatrix[x][y]
        else
            predMatrix[x][y] = Clip1( orgPredMatrix[x][y] )
    }
}
```

其中 filter[0]到 filter[3]由 filterIndex 和 offset 查表 102 得到。

- 否则，如果 iX 等于-1（参考左边像素）：

```
if (dx * dy < 0) {
    iYn = iY + 1
    iYnP2 = iY + 2
    iYnN1 = iY  - 1
}
else {
    iYn = iY  - 1
    iYnP2 = iY  - 2
    iYnN1 = iY + 1
}
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        orgPredMatrix[x][y] = (col[iYnN1+1] * filter[0] + col[iY+1] * filter[1] + col[iYn+1] * filter[2]
+ col[iYnP2+1] * filter[3] + filterOffset[filterIndex]) >> filterBit[filterIndex]
        if (MipfEnableFlag == 0)
            predMatrix[x][y] = orgPredMatrix[x][y]
        else
            predMatrix[x][y] = Clip1( orgPredMatrix[x][y] )
    }
}
```

其中 filter[0]到 filter[3]由 filterIndex 和 offset 查表 102 得到。

b) IntraChromaPredMode 等于 1（Intra_Chroma_DC 预测）。

1) 如果 row[i]、col[j]（i=1～K, j=1～L）均“可用”，则：

```
rowi = 0
for (i=1; i<=K; i++) {
    rowi += row[i]
}
colj = 0
for (j=1; j<=L; j++) {
    colj += col[j]
```

```
}
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        predMatrix[x][y] = ((rowi + colj + ((K + L) >> 1)) * (4096 / (K + L))) >> 12
    }
}
```

2) 否则，如果 row[i]（i=1～K）均“可用”，且 col[j]（j=1～L）均“不可用”，则：

```
rowi = 0
for (i=1; i<=K; i++) {
    rowi += row[i]
}
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        predMatrix[x][y] = (rowi + (K >> 1)) >> Log(K)
    }
}
```

3) 否则，如果 col[j]（i=1～L）均“可用”，且 row[i]（i=1～K）均“不可用”，则：

```
colj = 0
for (j=1; j<=L; j++) {
    colj += col[j]
}
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        predMatrix[x][y] = (colj + (L >> 1)) >> Log(L)
    }
}
```

4) 否则，

```
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        predMatrix[x][y] = 2^(BitDepth-1)
    }
}
```

其中 BitDepth 是编码样本精度。

c) IntraChromaPredMode 等于 2（Intra_Chroma_Horizontal 预测）。

```
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        orgPredMatrix[x][y] = col[y+1]
        predMatrix[x][y] = orgPredMatrix[x][y]
    }
}
```

d) IntraChromaPredMode 等于 3（Intra_Chroma_Vertical 预测）。

```
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        orgPredMatrix[x][y] = row[x+1]
        predMatrix[x][y] = orgPredMatrix[x][y]
    }
}
```

e) IntraChromaPredMode 等于 4（Intra_Chroma_Bilinear 预测）。

```
bilinearWeight[3] = {21, 13, 7}
weight = bilinearWeight[Log(Max(K, L) / Min(K, L)) - 1]
ishift = Log(Min(K, L))

ia = row[K]
ib = col[L]
if (K == L) {
    ic = (ia + ib + 1) >> 1
}
else {
    ic = (((ia << Log(K)) + (ib << Log(L))) * weight + (1 << (ishift + 5))) >> (ishift + 6)
}
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        predMatrix[x][y] = Clip1(((((ia - col[y+1]) * (x + 1)) << Log(L)) + (((ib - row[x+1]) * (y + 1))
<< Log(K)) + ((row[x+1] + col[y+1]) << (Log(K) + Log(L))) + ((ic << 1) - ia - ib) * x * y + (1 << (Log(K)
+ Log(L)))) >> (Log(K) + Log(L) + 1))
    }
}
```

f) IntraChromaPredMode 等于 5～20，令当前亮度预测块的尺寸为 M×N，色度预测块的尺寸为 K×L，I 表示当前块所在的图像的补偿后样本的亮度样值矩阵，Cb 表示当前块所在的图像的补偿后样本的色度 Cb 样值矩阵，r[i]、c[j]（i=0～2M，j=0～2N）是亮度参考样本，row[i]、col[i]（i=0～2K，j=0～2L）是色度参考样本。

1) 如果 IntraChromaPredMode 等于 5、6、7 或 8，或如果 IntraChromaPredMode 等于 9、10、11、12、13、14、15、16、17、18、19 或 20 且当前为 Cb 分量，则：

```
for (x=0; x<2K; x++) {
    for (y=0; y<2L; y++) {
        predChroma[x][y] = Clip1(((α * I[x][y]) >> iShift) + β)
    }
}
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
```

```
        if (x == 0) {
            predMatrix[0][y] = (predChroma[0][2*y] + predChroma[0][2*y+1] + 1) >> 1
        }
        else {
            predMatrix[x][y] = (predChroma[2*x-1][2*y] + 2 * predChroma[2*x][2*y] + predChroma[2*x+1][2*y]
+ predChroma[2*x-1][2*y+1] + 2 * predChroma[2*x][2*y+1] + predChroma[2*x+1][2*y+1] + 4) >> 3
        }
    }
}
```

2) 否则：

```
if (IntraChromaPredMode == 9 || IntraChromaPredMode == 10 || IntraChromaPredMode == 11 || IntraChromaPredMode
== 12) {
    k1 = 1
    k2 = 1
    p = 0
}
else if (IntraChromaPredMode == 13 || IntraChromaPredMode == 14 || IntraChromaPredMode == 15 ||
IntraChromaPredMode == 16) {
    k1 = 1
    k2 = 2
    p = 1
}
else if (IntraChromaPredMode == 17 || IntraChromaPredMode == 18 || IntraChromaPredMode == 19 ||
IntraChromaPredMode == 20) {
    k1 = 2
    k2 = 1
    p = 0
}
for (x=0; x<2K; x++) {
    for (y=0; y<2L; y++) {
        predChroma[x][y] = Clip3(0, (1 << (BitDepth + 2)) - 1, ((((k1*αCb + p)/k2 + αCr) * I[x][y]) >> iShift)
+ (k1*βCb + p)/k2 + βCr)
    }
}
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        if (x == 0) {
            predMatrixTemp[0][y] = (predChroma[0][2*y] + predChroma[0][2*y+1] + 1) >> 1
        }
        else {
```

```
            predMatrixTemp[x][y]    =    (predChroma[2*x-1][2*y]    +    2    *    predChroma[2*x][2*y]    +
predChroma[2*x+1][2*y] + predChroma[2*x-1][2*y+1] + 2 * predChroma[2*x][2*y+1] + predChroma[2*x+1][2*y+1]
+ 4) >> 3
        }
        predMatrix[x][y] = clip1(predMatrixTemp[x][y]  -  (k1*Cb[x][y] + p)/k2)
    }
}
```

上述色度帧内预测过程中的iShift以及线性模型系数α_X和β_X（X是Cb或Cr）按以下方法导出：

a) 如果满足以下条件之一，按如下过程导出模型系数和 iShift：

1) 如果 row[i]（i=1～K）和 col[i]（i=1～L）均“不可用”；
2) IntraChromaPredMode 等于 8、12、16 或 20 且 row[i]（i=1～K）均“不可用”；
3) IntraChromaPredMode 等于 7、11、15 或 19 且 col[i]（i=1～L）均“不可用”；

```
αX = 0
βX = 1 << (BitDepth  -  1)
iShift = 0
```

b) 否则，$iShift_X$等于 16，并按以下方法导出α_X和β_X（X 是 Cb 或 Cr）：

1) 如果 IntraChromaPredMode 等于 5、9、13 或 17：

- 如果 row[i]（i=1～K）、col[j]（j=1～L）均“可用”：

```
posA0 = 0
posL0 = 0
if (K >= L) {
    posA1 = K  -  (K / L)
    posL1 = L  -  1
}
else {
    posA1 = K  -  1
    posL1 = L  -  (L / K)
}
x[0] = (r[2*posA0] + 2 * r[2*posA0+1] + r[2*posA0+2] + 2) >> 2
y[0] = row[posA0+1]
x[1] = (r[2*posA1] + 2 * r[2*posA1+1] + r[2*posA1+2] + 2) >> 2
y[1] = row[posA1+1]
x[2] = (c[2*posL0+1] + c[2*posL0+2] + 1) >> 1
y[2] = col[posL0+1]
x[3] = (c[2*posL1+1] + c[2*posL1+2] + 1) >> 1
y[3] = col[posL1+1]
```

- 如果 row[i]“可用”且 col[j]均“不可用”（i=1～K，j=1～L）：

```
posA0 = 0
posA1 = K / 4
```

```
posA2 = 2 * K / 4
posA3 = 3 * K / 4
x[0] = (3 * r[2*posA0+1] + r[2*posA0+2] + 2) >> 2
y[0] = row[posA0+1]
x[1] = (r[2*posA1] + 2 * r[2*posA1+1] + r[2*posA1+2] + 2) >> 2
y[1] = row[posA1+1]
x[2] = (r[2*posA2] + 2 * r[2*posA2+1] + r[2*posA2+2] + 2) >> 2
y[2] = row[posA2+1]
x[3] = (r[2*posA3] + 2 * r[2*posA3+1] + r[2*posA3+2] + 2) >> 2
y[3] = row[posA3+1]
```

- 如果 row[i]均“不可用”且 col[j]均“可用”（i=1～K，j=1～L）：

```
posL0 = 0
posL1 = L / 4
posL2 = 2 * L / 4
posL3 = 3 * L / 4
x[0] = (c[2*posL0+1] + c[2*posL0+2] + 1) >> 1
y[0] = col[posL0+1]
x[1] = (c[2*posL1+1] + c[2*posL1+2] + 1) >> 1
y[1] = col[posL1+1]
x[2] = (c[2*posL2+1] + c[2*posL2+2] + 1) >> 1
y[2] = col[posL2+1]
x[3] = (c[2*posL3+1] + c[2*posL3+2] + 1) >> 1
y[3] = col[posL3+1]
```

2) 如果 IntraChromaPredMode 等于 6、10、14 或 18：

- 如果 row[i]（i=1～K）、col[j]（j=1～L）均“可用”：

```
posA0 = 3*K / 8
posL0 = 3*L / 8
posA1 = 5*K / 8
posL1 = 5*L / 8
x[0] = (r[2*posA0] + 2 * r[2*posA0+1] + r[2*posA0+2] + 2) >> 2
y[0] = row[posA0+1]
x[1] = (r[2*posA1] + 2 * r[2*posA1+1] + r[2*posA1+2] + 2) >> 2
y[1] = row[posA1+1]
x[2] = (c[2*posL0+1] + c[2*posL0+2] + 1) >> 1
y[2] = col[posL0+1]
x[3] = (c[2*posL1+1] + c[2*posL1+2] + 1) >> 1
y[3] = col[posL1+1]
```

- 如果 row[i]均“可用”且 col[j]均“不可用”（i=1～K，j=1～L）：

```
posA0 = K / 8
```

```
posA1 = 3 * K / 8
posA2 = 5 * K / 8
posA3 = 7 * K / 8
x[0] = （posA0 == 0） ? ( (3 * r[2*posA0+1] + r[2*posA0+2] + 2) >> 2 ) : ((r[2*posA1] + 2 * r[2*posA1+1]
+ r[2*posA1+2] + 2) >> 2)
y[0] = row[posA0+1]
x[1] = (r[2*posA1] + 2 * r[2*posA1+1] + r[2*posA1+2] + 2) >> 2
y[1] = row[posA1+1]
x[2] = (r[2*posA2] + 2 * r[2*posA2+1] + r[2*posA2+2] + 2) >> 2
y[2] = row[posA2+1]
x[3] = (r[2*posA3] + 2 * r[2*posA3+1] + r[2*posA3+2] + 2) >> 2
y[3] = row[posA3+1]
```

- 如果 row[i]均“不可用”且 col[j]均“可用”（i=1～K，j=1～L）：

```
posL0 = L / 8
posL1 = 3 * L / 8
posL2 = 5 * L / 8
posL3 = 7 * L / 8
x[0] = (c[2*posL0+1] + c[2*posL0+2] + 1) >> 1
y[0] = col[posL0+1]
x[1] = (c[2*posL1+1] + c[2*posL1+2] + 1) >> 1
y[1] = col[posL1+1]
x[2] = (c[2*posL2+1] + c[2*posL2+2] + 1) >> 1
y[2] = col[posL2+1]
x[3] = (c[2*posL3+1] + c[2*posL3+2] + 1) >> 1
y[3] = col[posL3+1]
```

3) 如果 IntraChromaPredMode 等于 8、12、16 或 20，如果 row[i]均“可用”，令 len 等于 K：

```
posA0 = 0
posA1 = len / 4
posA2 = 2 * len / 4
posA3 = 3 * len / 4
x[0] = (3 * r[2*posA0+1] + r[2*posA0+2] + 2) >> 2
y[0] = row[posA0+1]
x[1] = (r[2*posA1] + 2 * r[2*posA1+1] + r[2*posA1+2] + 2) >> 2
y[1] = row[posA1+1]
x[2] = (r[2*posA2] + 2 * r[2*posA2+1] + r[2*posA2+2] + 2) >> 2
y[2] = row[posA2+1]
x[3] = (r[2*posA3] + 2 * r[2*posA3+1] + r[2*posA3+2] + 2) >> 2
y[3] = row[posA3+1]
```

4) 如果 IntraChromaPredMode 等于 7、11、15 或 19，如果 col[j]均“可用”，令 len 等于 L：

```
posL0 = 0
posL1 = len / 4
posL2 = 2 * len / 4
posL3 = 3 * len / 4
x[0] = (c[2*posL0+1] + c[2*posL0+2] + 1) >> 1
y[0] = col[posL0+1]
x[1] = (c[2*posL1+1] + c[2*posL1+2] + 1) >> 1
y[1] = col[posL1+1]
x[2] = (c[2*posL2+1] + c[2*posL2+2] + 1) >> 1
y[2] = col[posL2+1]
x[3] = (c[2*posL3+1] + c[2*posL3+2] + 1) >> 1
y[3] = col[posL3+1]
```

5) 线性模型系数α_x和β_x：

```
minIndex[2] = {0, 2}
maxIndex[2] = {1, 3}
if (x[minIndex[0]] > x[minIndex[1]]) {
    Exchange(minIndex[0], minIndex[1])
}
if (x[maxIndex[0]] > x[maxIndex[1]]) {
    Exchange(maxIndex[0], maxIndex[1])
}
if (x[minIndex[0]] > x[maxIndex[1]]) {
    Exchange(minIndex[0], maxIndex[0])
    Exchange(minIndex[1], maxIndex[1])
}
if (x[minIndex[1]] > x[maxIndex[0]]) {
    Exchange(minIndex[1], maxIndex[0])
}
xMin = (x[minIndex[0]] + x[minIndex[1]] + 1) >> 1
yMin = (y[minIndex[0]] + y[minIndex[1]] + 1) >> 1
xMax = (x[maxIndex[0]] + x[maxIndex[1]] + 1) >> 1
yMax = (y[maxIndex[0]] + y[maxIndex[1]] + 1) >> 1
diffX = xMax - xMin
diffY = yMax - yMin
if (diffX > 64) {
    shift = (BitDepth > 8) ? BitDepth - 6 : 2
    add = 1 << (shift - 1)
    αx = (diffY * TscpmTable[((diffX+add)>>shift)-1] + add) >> shift
    βx = yMin - ((αx * xMin) >> iShiftx)
```

```
}
else if (diffX > 0) {
    αx = diffY * TscpmTable[diffX-1]
    βx = yMin - ((αx * xMin) >> iShiftx)
}
else {
    αx = 0
    βx = yMin
}
```

其中 TscpmTable 见表 108。

表108 TscpmTable 查找

索引	TscpmTable	索引	TscpmTable	索引	TscpmTable	索引	TscpmTable
0	65536	16	3855	32	1985	48	1337
1	32768	17	3640	33	1927	49	1310
2	21845	18	3449	34	1872	50	1285
3	16384	19	3276	35	1820	51	1260
4	13107	20	3120	36	1771	52	1236
5	10922	21	2978	37	1724	53	1213
6	9362	22	2849	38	1680	54	1191
7	8192	23	2730	39	1638	55	1170
8	7281	24	2621	40	1598	56	1149
9	6553	25	2520	41	1560	57	1129
10	5957	26	2427	42	1524	58	1110
11	5461	27	2340	43	1489	59	1092
12	5041	28	2259	44	1456	60	1074
13	4681	29	2184	45	1424	61	1057
14	4369	30	2114	46	1394	62	1040
15	4096	31	2048	47	1365	63	1024

9.7.1.5.3 色度帧内预测滤波

色度帧内预测滤波过程如下：

```
predMatrixIPF[-1][-1] = 0
for (x=0; x<K; x++) {
    predMatrixIPF[x][-1] = row[x+1]
}
for (y=0; y<L; y++) {
    predMatrixIPF[-1][y] = col[y+1]
}
for (x=0; x<K; x++) {
```

```
    for (y=0; y<L; y++) {
        predMatrixIPF[x][y] = orgPredMatrix[x][y]
    }
}
if (IntraChromaPredMode == 3 || IntraChromaPredMode == 8) {
    for (x=0; x<K; x++) {
        for (y=0; y<L; y++) {
            predMatrix[x][y] = Clip1((f[x] * predMatrixIPF[-1][y] + (64- f[x]) * predMatrixIPF[x][y] + 32) >>
6)
        }
    }
}
else if (IntraChromaPredMode == 2 || IntraChromaPredMode == 7) {
    for (x=0; x<K; x++) {
        for (y=0; y<L; y++) {
            predMatrix[x][y] = Clip1((f[y] * predMatrixIPF[x][-1] + (64 - f[y]) * predMatrixIPF[x][y] +
32) >> 6)
        }
    }
}
```

其中，predMatrixIPF是(K+1)×(L+1)大小的样本矩阵，K×L块内坐标为(x,y)的像素对应的滤波系数f[x]和f[y]分别由K、x和L、y查表107得到。

9.7.1.6 导出空域角度加权预测样本

9.7.1.6.1 概述

输入：空域角度加权预测模式索引值SawpIndex、predMatrixSawp0、predMatrixSawp1以及当前编码单元的宽度W和高度H。

输出：空域角度加权预测样本矩阵predMatrixSawp。

首先按9.7.1.6.2得到空域角度加权预测权重矩阵，再按9.7.1.6.3得到空域角度加权预测样本矩阵。

9.7.1.6.2 导出空域角度加权预测权重矩阵

输入：当前预测单元的宽度W和高度H。

输出：空域角度加权预测的亮度权重矩阵SawpWeightMatrixY和色度权重矩阵SawpWeightMatrixUV。

第1步，计算变量stepIndex、angleIndex和angleAreaIndex。

```
stepIndex = (SawpIndex >> 3) - 3
modAngNum = SawpIndex % 8
if (modAngNum == 2) {
    angleIndex = 7
}
else if (modAngNum == 6) {
    angleIndex = 8
```

```
}
else {
    angleIndex = modAngNum % 2
}
angleAreaIndex = modAngNum >> 1
```

第2步，先根据angleAreaIndex查表109得到vL，再根据angleAreaIndex和PictureAwpRefineIndex查表110得到shift和fP，最后得到参考权重列表ReferenceWeights。

```
for (x=0; x<vL; x++) {
    ReferenceWeights[x] = Clip3(0, 8, (x - fP) << shift)
}
```

表109 angleArealndex 和 vL 的对应关系

angleAreaIndex	vL
0	(H + (W >> angleIndex)) << 1
1	
2	(W + (H >> angleIndex)) << 1
3	

表110 angleArealndex，PictureAwpRefineIndex 和 shift，fP 的对应关系

angleAreaIndex	PictureAwpRefineIndex	shift	fP
0	0	0	(vL >> 1) - 6 + stepIndex * ((vL >> 3) - 1)
	1	2	(vL >> 1) - 3 + stepIndex * ((vL >> 3) - 1)
1	0	0	(vL >> 1) - 4 + stepIndex * ((vL >> 3) -1) – ((W << 1) >> angleIndex)
	1	2	(vL >> 1) - 1 + stepIndex * ((vL >> 3) -1) – ((W << 1) >> angleIndex)
2	0	0	(vL >> 1) - 4 + stepIndex * ((vL >> 3) -1) – ((H << 1) >> angleIndex)
	1	2	(vL >> 1) - 1 + stepIndex * ((vL >> 3) -1) – ((H << 1) >> angleIndex)
3	0	0	(vL >> 1) - 6 + stepIndex *((vL >> 3) -1)
	1	2	(vL >> 1) - 3 + stepIndex * ((vL >> 3) -1)

第3步，先根据angleAreaIndex查表111得到tP，再得到亮度权重矩阵SawpWeightMatrixY。

```
for (y=0; y<H; x++) {
  for (x=0; x<W; x++) {
    SawpWeightMatrixY[x][y] = ReferenceWeights[tP]
  }
}
```

表111 angleArealndex 和 tP 的对应关系

angleAreaIndex	tP
0	(y << 1) + ((x << 1) >> angleIndex)
1	(y << 1) - ((x << 1) >> angleIndex)
2	(x << 1) - ((y << 1) >> angleIndex)
3	(x << 1) + ((y << 1) >> angleIndex)

第4步，得到色度权重矩阵SawpWeightMatrixUV。

```
for (y=0; y<H/2; y++) {
    for (x=0; x<W/2; x++) {
        SawpWeightMatrixUV[x][y] = SawpWeightMatrixY[x<<1][y<<1]
    }
}
```

9.7.1.6.3 导出空域角度加权预测样本

输入：空域角度加权预测亮度权重矩阵SawpWeightMatrixY和色度权重矩阵SawpWeightMatrixUV，当前预测块左上角样本位置(xE,yE)，当前预测块的宽度W和高度H。

输出：加权预测样本矩阵predMatrix。

如果当前预测块是亮度预测块：

```
for (x=0; x<W; x++) {
    for (y=0; y<H; y++) {
        predMatrix[x][y] = (predLumaMatrix0[x][y] * SawpWeightMatrixY[x][y] + predLumaMatrix1 [x][y] * (8
- SawpWeightMatrixY[x][y] ) + 4) >> 3
    }
}
```

如果当前预测块是色度预测块：

```
for (x=0; x<W/2; x++) {
    for (y=0; y<H/2; y++) {
        predMatrix[x][y] = (predChromaMatrix0[x][y] * SawpWeightMatrixUV[x][y] + predChromaMatrix1 [x][y]
* (8 - SawpWeightMatrixUV[x][y] ) + 4) >> 3
    }
}
```

9.7.2 块复制帧内预测

9.7.2.1 概述

本条定义块复制帧内预测样本矩阵的导出过程。所导出的预测样本矩阵的水平尺寸和垂直尺寸与当前预测单元中对应分量预测块的水平尺寸和垂直尺寸一致。

9.7.2.2 导出预测样本

如果当前预测块是亮度预测块，令当前预测块左上角样本在当前图像亮度样本矩阵中的位置是(xE,yE)，亮度预测样本矩阵predMatrixIbc的宽度和高度分别是W和H，当前预测单元块矢量是BvE。predMatrixIbc的元素predMatrixIbc[x][y]的值是当前图像未滤波重建的整像素精度亮度样本矩阵中(xPos,yPos)位置的样本值。其中(xPos,yPos)的值如下。

```
xPos = xE + x + (BvE->x >> 2)
yPos = yE + y + (BvE->y >> 2)
```

如果当前预测块是色度预测块,令包含当前预测块左上角样本的亮度预测块的左上角样本在当前图像的亮度样本矩阵中的位置是(xE,yE)，色度预测样本矩阵predMatrixIbc的宽度和高度分别是W和H，包含当前预测块所在预测单元的亮度编码块的块矢量是BvE。predMatrixIbc的元素predMatrixIbc[x][y]的值是当前图像未滤波重建的1/2精度色度样本矩阵中(xPos,yPos)位置的样本值。当前图像未滤波重建的色度1/2精度样本矩阵中各个位置的元素值按9.7.2.3插值得到。其中(xPos,yPos)的值如下。

```
xPos = (xE + 2 * x) + (BvE->x >> 2)
yPos = (yE + 2 * y) + (BvE->y >> 2)
```

9.7.2.3 块复制帧内预测色度样本的插值

参考图像色度样本矩阵中的样本位置见图31，A，B，C，D是相邻整像素样本，dx与dy是整像素样本A周边分像素样本a(dx,dy)与A的水平和垂直距离，dx等于fx&1，dy等于fy&1，其中(fx,fy)是该分像素样本在1/2精度的色度样本矩阵中的坐标。整像素Ax,y和周边的3个分像素样本ax,y(dx,dy)的具体位置见图32。

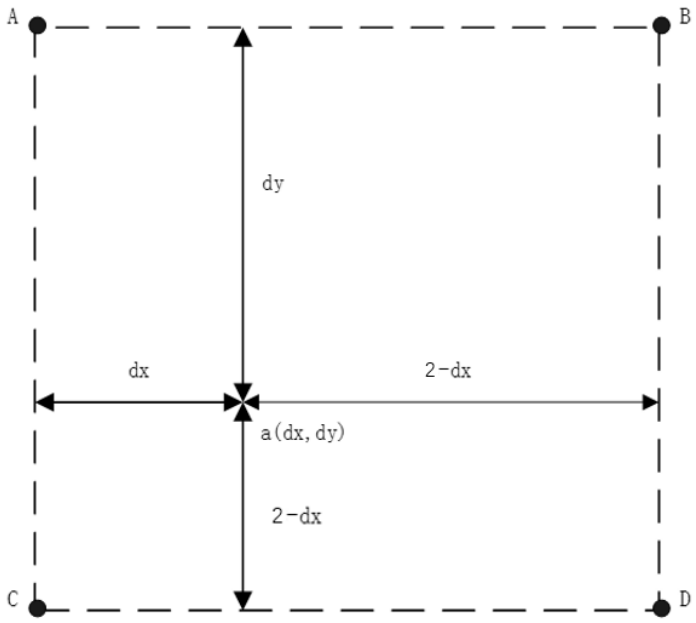


图31 块复制帧内预测色度样本位置

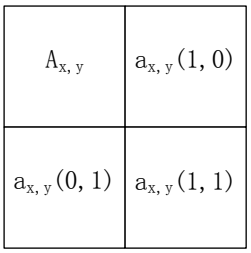


图32　块复制帧内预测整像素样本和分像素样本的位置

色度滤波系数见表112。

表112　块复制帧内预测色度滤波系数

数组标识	滤波器系数
C[0]	{64,0}
C[1]	{32,32}

对于dx等于0或dy等于0的分像素点，可直接用色度整像素插值得到，对于dx不等于0且dy不等于0的点，使用整像素行（dy等于0）上的分像素进行计算：

```
if (dx == 0) {
    a_{x,y}(0,dy) = Clip3(0, (1 << BitDepth) - 1, (C[dy][0] * A_{x,y} + C[dy][1] * A_{x,y+1} + 32) >> 6)
}
else if (dy == 0) {
    a_{x,y}(dx,0) = Clip3(0, (1 << BitDepth) - 1, (C[dx][0] * A_{x,y} + C[dx][1] * A_{x+1,y} + 32) >> 6)
}
else {
    a_{x,y}(dx,dy) = Clip3(0, (1 << BitDepth) - 1, (C[dy][0] * a'_{x,y}(dx,0) + C[dy][1] * a'_{x,y+1}(dx,0) + (1 << (19-BitDepth))) >> (20-BitDepth))
}
```

其中，$a'_{x,y}(dx,0)$是整像素行上的分像素的临时值，定义为：

```
a'_{x,y}(dx,0) = (C[dx][0] * A_{x,y} + C[dx][1] * A_{x+1,y} + ((1 << (BitDepth - 8)) >> 1)) >> (BitDepth - 8)
```

9.7.3　串复制帧内预测

9.7.3.1　概述

本条定义用于串复制帧内预测的预测样本矩阵的导出过程。所导出的预测样本矩阵的水平尺寸和垂直尺寸与当前预测单元中对应分量预测块的水平尺寸和垂直尺寸一致。

如果IscSubtypeFlag的值为0，则按9.7.3.2导出预测样本矩阵predMatrixIsc；否则，如果IscSubtypeFlag的值为1，则按9.7.3.3导出预测样本矩阵predMatrixIsc。

9.7.3.2　导出普通串子模式的预测样本

当前编码单元按照往返扫描顺序划分为 IscPartNum 个部分。

第 1 步，令 cuWidth 和 cuHeight 分别是当前编码单元的宽度和高度，将 strPos 和 IscPartNumSplit

初始化为0。

第2步，令i从0到IscPartNum-1，如果IscMatchType[i]等于1，第i部分的长度是StrLen[i]，执行以下操作a)～c)；否则，如果IscMatchType[i]等于0，仅执行操作c)：

a) 初始化。

```
NoOverlapStrNum = 1
RemainingStrLen = StrLen[i]
```

b) 计算NoOverlapStrLen。

```
NoOverlapStrLen[1] = StrLen[i]
if (Sv[i]->y < 0 && Abs(Sv[i]->x) < cuWidth) {
    xStartPos = TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos][0]
    yStartPos = TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos][1]
    NoOverlapStrLen[1] = Min((Abs(Sv[i]->y) - 1) * cuWidth + ((yStartPos & 0x1) ? xStartPos + 1 : cuWidth
- xStartPos), RemainingStrLen)
}
while (RemainingStrLen > NoOverlapStrLen[NoOverlapStrNum]) {
    RemainingStrlen -= NoOverlapStrLen[NoOverlapStrNum]
    IscPartNumSplit++
    NoOverlapStrNum++
    NoOverlapStrLen[NoOverlapStrNum] = Min(Abs(Sv[i]->y) * cuWidth, RemainingStrLen)
}
```

c) 如果IscMatchType[i]等于1，令k从1～NoOverlapStrNum。对每个k，令j从0～NoOverlapStrLen[k]-1，执行以下操作；否则，如果IscMatchType[i]等于0，令j从0～3，执行以下操作。

1) 计算当前亮度样本的坐标(xj,yj)。

```
xj = TravScan[Log(CuWidth)-2][Log(CuHeight)-2][strPos][0]
yj = TravScan[Log(CuWidth)-2][Log(CuHeight)-2][strPos][1]
```

2) 如果当前预测块是亮度预测块，令当前预测块左上角样本在当前图像的亮度样本矩阵中的位置是(xE,yE)，当前预测单元第i部分串矢量是Sv[i]。执行以下操作：

- 如果IscMatchType[i]等于1，或IscMatchType[i]等于0且PixelMatchTypeFlag[i][j]等于1，如果(xPos,yPos)不在当前编码单元内，则亮度预测样本矩阵predMatrixIsc元素predMatrixIsc[xj][yj]的值是当前图像未滤波重建的整像素精度亮度样本矩阵中(xPos,yPos)位置的样本值；否则，亮度预测样本矩阵predMatrixIsc元素predMatrixIsc[xj][yj]的值是predMatrixIsc[xj + Sv[i]->x][yj + Sv[i]->y]的值。

```
xPos = xE + xj + Sv[i]->x
yPos = yE + yj + Sv[i]->y
```

- 否则，如果IscMatchType[i]等于0且PixelMatchTypeFlag[i][j]等于0，predMatrixIsc[xj][yj]等于当前预测单元的IscUnmatchedPixelY[i][j]的值。

3) 如果当前预测块是色度预测块，且 xj & 0x1 的值和 yj & 0x1 的值均等于 0，令包含当前预测块左上角样本的亮度预测块的左上角样本在当前图像的亮度样本矩阵中的位置是(xE,yE)，当前预测单元第 i 部分串矢量是 Sv[i]。执行以下操作：

- 如果 IscPixelMatchType[i][j]等于 1 且 IscMatchType[i]等于 0,或 IscMatchType[i]等于 1，色度预测样本矩阵 predMatrixIsc 的元素 predMatrixIsc[x][y]的值是当前图像未滤波重建的整像素精度色度样本矩阵中(xPos,yPos)位置的样本值。

```
xPos = (xE + xj + Sv[i]->x) >> 1
yPos = (yE + yj + Sv[i]->y) >> 1
```

- 如果 IscMatchType[i]等于 0 且 IscPixelMatchType[i][j]等于 0，Cb 分量色度预测样本矩阵 predMatrixIsc 的元素 predMatrixIsc[x][y]的值等于当前预测单元的 IscUnmatchedPixelCb[i][j]的值；Cr 分量色度预测样本矩阵 predMatrixIsc 的元素 predMatrixIsc[x][y]的值等于当前预测单元的 IscUnmatchedPixelCr[i][j]的值。

4) strPos 加 1。

9.7.3.3 导出非普通串子模式的预测样本

令 cuWidth 和 cuHeight 分别是当前编码单元的宽度和高度，当前编码单元按照往返扫描顺序划分为 IscPartNum 个部分，令 strPos 和 IscPartNumSplit 为 0，令 i 从 0～IscPartNum-1，按以下方法依次导出各部分的预测样本。

第 1 情况，当前预测块第 i 部分是单位基矢量串或等值串。

a) 该部分的长度等于 StrLen[i]。

b) 如果当前串是单位基矢量串，执行以下操作。

1) 初始化。

```
NoOverlapStrNum = 1
RemainingStrLen = StrLen[i]
```

2) 计算 xStartPos、yStartPos 和 yEndPos。

```
xStartPos = TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos][0]
yStartPos = TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos][1]
yEndPos = TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos+StrLen[i]][1]
```

3) 计算 NoOverlapStrLen。

```
if (yEndPos  -  yStartPos < 1){
    NoOverlapStrLen[1] = StrLen[i]
}
else {
    NoOverlapStrLen[1] = ((yStartPos & 0x1) ? xStartPos + 1 : cuWidth  -  xStartPos)
}
while (RemainingStrLen > NoOverlapStrLen[NoOverlapStrNum]) {
    NoOverlapStrNum++
    IscPartNumSplit++
    RemainingStrlen -= NoOverlapStrLen[NoOverlapStrNum-1]
```

```
    NoOverlapStrLen[NoOverlapStrNum] = Min(cuWidth, RemainingStrLen)
}
```

c) 如果当前串是等值串，令 j 从 0～StrLen[i]-1，执行以下操作；否则，如果当前串是单位基矢量串，令 k 从 1～NoOverlapStrNum，对每个 k，令 j 从 0～NoOverlapStrLen[k]-1，执行以下操作。

1) 计算当前亮度样本的坐标，记为(xj,yj)：

```
xj = TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos][0]
yj = TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos][1]
```

2) 如果当前预测块是亮度预测块,记当前预测块左上角样本在当前图像的亮度样本矩阵中的位置为(xE,yE)，亮度预测样本矩阵 predMatrixIsc 的元素 predMatrixIsc[xj][yj]的值如下：

```
if (StringType[i] == 1) {
    predMatrixIsc[xj][yj] = LcuRowBufY[PpInfoList[PvAddr[strPos]][0]][PpInfoList[PvAddr[strPos]][1]]
}
else {
    predMatrixIsc[xj][yj] = predMatrixIsc[xj][yj-1]
}
```

3) 如果当前预测块是色度预测块，且 xj & 0x1 的值和 yj & 0x1 的值均等于 0，导出 C00、C10、C01、C11 和 uvCnt 的值，色度预测样本矩阵 predMatrixIsc 中的元素 predMatrixIsc[xj>>1][yj>>1]的值如下：

```
predMatrixIsc[xj>>1][yj>>1] = PvFlag[xj][yj]  ? C00  : (C00 + C10 + C01 + C11 + (uvCnt >> 1)) / Max(uvCnt, 1)
```

4) strPos 加 1。

第 2 种情况，当前预测块第 i 部分的类型是未匹配像素。

a) 计算当前亮度样本的坐标，记为(xj,yj)：

```
xj = TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos][0]
yj = TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos][1]
```

b) 如果当前预测块是亮度预测块，则亮度预测样本矩阵 predMatrixIsc 的元素 predMatrixIsc[xj][yj]等于 IscNosUpY[i]。其中 IscNosUpY[i]是当前预测单元第 i 部分的未匹配像素的 Y 分量值。

c) 如果当前预测块是色度预测块，且 xj & 0x1 的值和 yj & 0x1 的值均等于 0，导出 C00、C10、C01、C11 和 uvCnt 的值，色度预测样本矩阵 predMatrixIsc 的元素 predMatrixIsc[xj>>1][yj>>1]的值等于：

```
predMatrixIsc[xj>>1][yj>>1] = (C00 + C10 + C01 + C11 + (uvCnt>>1)) / Max(uvCnt,1)
```

d) strPos 加 1。

IscNosUpC[i]是当前预测单元第i部分的未匹配像素的色度分量值，C00、C10、C01、C11、LcuRowBufC和IscNosUpC中的C为Cb或Cr。导出C00、C10、C01、C11和uvCnt的过程如下：

```
offsetAbove = 2 * (cuWidth  -  TravScan[Log(cuWidth)-2][Log(cuHeight)-2][strPos][0])  -  1

x00InLcuR = PPInfoList[PvAddr[strPos]][0]
y00InLcuR = PPInfoList[PvAddr[strPos]][1]
x10InLcuR = PPInfoList[PvAddr[strPos+1]][0]
y10InLcuR = PPInfoList[PvAddr[strPos+1]][1]
x01InLcuR = PPInfoList[PvAddr[strPos+offsetAbove]][0]
y01InLcuR = PPInfoList[PvAddr[strPos+offsetAbove]][1]
x11InLcuR = PPInfoList[PvAddr[strPos+offsetAbove-1]][0]
y11InLcuR = PPInfoList[PvAddr[strPos+offsetAbove-1]][1]

uvCnt = 4 - (PixelType[strPos] != 2  ? ((x00InLcuR | y00InLcuR) & 1) : upYonly[UnmatchedPixelIndex[strPos]])
 - (PixelType[strPos + 1] != 2  ? ((x10InLcuR|y10InLcuR)&1) : upYonly[UnmatchedPixelIndex[strPos + 1]]) -
(PixelType[strPos + offsetAbove] != 2  ? ((x01InLcuR|y01InLcuR)&1) : upYonly[UnmatchedPixelIndex[strPos +
offsetAbove]])  -  (PixelType[strPos  +  offsetAbove  -  1]  !=  2  ?  ((x11InLcuR|y11InLcuR)&1)  :
upYonly[UnmatchedPixelIndex[strPos + offsetAbove - 1]])

eP00 = ((x00InLcuR | y00InLcuR) & 1) == 0 ? LcuRowBufC[x00InLcuR>>1][y00InLcuR>>1] : 0
uP00 = upYonly[UnmatchedPixelIndex[strPos]] == 0 ? IscNosUpC[UnmatchedPixelIndex[strPos]] : 0
eP10 = ((x10InLcuR | y10InLcuR) & 1) == 0 ? LcuRowBufC[x10InLcuR>>1][y10InLcuR>>1] : 0
uP10 = upYonly[UnmatchedPixelIndex[strPos + 1]] == 0 ? IscNosUpC[UnmatchedPixelIndex[strPos + 1]] : 0
eP01 = ((x01InLcuR | y01InLcuR) & 1) == 0 ? LcuRowBufC[x01InLcuR>>1][y01InLcuR>>1] : 0
uP01 = upYonly[UnmatchedPixelIndex[strPos + offsetAbove]] == 0 ? IscNosUpC[UnmatchedPixelIndex[strPos +
offsetAbove]] : 0
eP11 = ((x11InLcuR | y11InLcuR) & 1) == 0 ? LcuRowBufC[x11InLcuR>>1][y11InLcuR>>1] : 0
uP11 = upYonly[UnmatchedPixelIndex[strPos + offsetAbove] - 1] == 0 ? IscNosUpC[UnmatchedPixelIndex[strPos
+ offsetAbove - 1]] : 0

C00 = PixelType[strPos] != 2  ? eP00 : uP00
C10 = PixelType[strPos+1] != 2  ? eP10 : uP10
C01 = PixelType[strPos+offsetAbove] != 2  ? eP01 : uP01
C11 = PixelType[strPos+offsetAbove-1] != 2  ? eP11 : uP11
```

9.8 帧间预测

9.8.1 概述

本条定义帧间预测过程。

先按9.8.2得到PredMatrixL0、PredMatrixL1、GradXL0、GradYL0、GradXL1和GradYL1。

再由PredMatrixL0、PredMatrixL1、GradXL0、GradYL0、GradXL1和GradYL1按9.8.3得到PredMatrixComb。

最后由PredMatrixL0、PredMatrixL1、PredMatrixComb按9.8.4得到PredMatrixInter。

9.8.2 导出预测样本

9.8.2.1 概述

输入：当前编码单元的宽度W和高度H，当前编码单元左上角样本在当前图像的亮度样本矩阵中的位置(xE,yE)。

输出：预测样本矩阵PredMatrixL0和PredMatrixL1,梯度矩阵GradXL0、GradYL0、GradXL1和GradYL1。

如果当前编码单元只包含色度编码块，由运动信息MotionInfo、(xE,yE)、W和H按9.8.2.2得到predMatrixL0和predMatrixL1。

否则，如果当前编码单元的编码单元子类型是‘P_Skip_Affine’‘P_Direct_Affine’‘P_Inter_Affine’‘B_Skip_Affine’‘B_Direct_Affine’或‘B_Inter_Affine’，则DMVRFlag的值等于0，按以下方法导出当前预测单元的预测样本。

a) 如果满足以下条件之一，则子块的宽度 sWa 和高度 sHa 均等于 8，iShift 等于 3；否则子块的宽度 sWa 和高度 sHa 均等于 4，iShift 等于 2。
   1) AffineSubblockSizeFlag的值为1。
   2) 当前预测单元的预测参考模式为‘PRED_List01’。
b) 令 i 和 j 是当前 sWa×sHa 子块的水平和垂直索引值，i=0～((W>>iShift)-1)，j=0～((H>>iShift)-1)，当前子块 X 左上角的坐标是(x,y)。如果 i 等于 0 且 j 等于 0，则 X 是 A 子块；否则，如果 i 等于 0 且 j 等于(W/sWa)-1，则 X 是 C 子块；否则，如果 i 等于(H/sHa)-1 且 j 等于 0，则 X 是 B 子块；否则，X 是 N 子块，见图 25。按以下方法导出 mvC0 和 mvC1。
   1) 如果sWa等于8，则mvC0等于MvArrayL0[i][j]，mvC1等于MvArrayL1[i][j]。
   2) 否则：

```
x = xE + sWa * i
y = yE + sHa * j
ti = (i >> 1) << 1
tj = (j >> 1) << 1
mvC0->x    =    (MvArrayL0[ti][tj]->x    +    MvArrayL0[ti+1][tj]->x    +    MvArrayL0[ti][tj+1]->x    +
MvArrayL0[ti+1][tj+1]->x + 2)  >> 2
mvC0->y    =    (MvArrayL0[ti][tj]->y    +    MvArrayL0[ti+1][tj]->y    +    MvArrayL0[ti][tj+1]->y    +
MvArrayL0[ti+1][tj+1]->y + 2) >> 2
mvC1->x    =    (MvArrayL1[ti][tj]->x    +    MvArrayL1[ti+1][tj]->x    +    MvArrayL1[ti][tj+1]->x    +
MvArrayL1[ti+1][tj+1]->x + 2)  >> 2
mvC1->y    =    (MvArrayL1[ti][tj]->y    +    MvArrayL1[ti+1][tj]->y    +    MvArrayL1[ti][tj+1]->y    +
MvArrayL1[ti+1][tj+1]->y + 2) >> 2
```

c) 当前子块的运动信息MotionInfo由当前预测单元的仿射运动信息motionInfoAffine中的预测参考模式、L0 运动矢量集合中的运动矢量 MvArrayL0[i][j]、L1 运动矢量集合中的运动矢量 MvArrayL1[i][j]、当前预测单元的 L0 参考索引值 refIndexL0 和当前预测单元的 L0 参考索引值 refIndexL1 组成。由运动信息 MotionInfo、(x,y)、mvC0、mvC1、sWa、sHa、asrInfo0、asrInfo1 和 X（X 是 A、B、C 或 N）按 9.8.2.3 得到 predMatrixL0 和 predMatrixL1。
d) PredMatrixL0 由各个子块的 predMatrixL0 组成，PredMatrixL1 由各个子块的 predMatrixL1 组成，GradXL0 由各个子块的 gradXL0 组成，GradYL0 由各个子块的 gradYL0 组成，GradXL1 由各个子块的 gradXL1 组成，GradYL1 由各个子块的 gradYL1 组成。

否则，如果当前编码单元的编码单元子类型是‘P_Skip_Mvap’‘P_Direct_Mvap’‘B_Skip_Mvap’或‘B_Direct_Mvap’，令i和j是当前8×8子块的水平和垂直索引值，i=0～((W>>3)-1),j=0～((H>>3)-1)，按以下方法导出当前预测单元的预测样本。

a) 当前子块的运动信息 MotionInfo 是 MotionArray[i][j]，当前子块左上角的坐标(x,y)等于(xE+8*i,yE+8*j)，子块的宽度 sWa 和高度 sHa 均等于 8。如果满足以下条件之一，则 DmvrFlag 的值为 0；否则，DmvrFlag 的值为 1。
   1) DmvrEnableFlag的值为0。
   2) 当前子块的预测参考模式不是‘PRED_List01’。
   3) 当前预测单元的InterPcFlag的值为1。
   4) 当前子块的参考图像队列0中参考索引是RefIndexL0的图像和参考图像队列1中参考索引是RefIndexL1的图像均位于当前图像显示顺序之前或之后。
   5) 当前子块的参考图像队列0中参考索引是RefIndexL0的图像的图像距离索引是DistanceIndexRef0，当前子块的参考图像队列1中参考索引是RefIndexL1的图像的图像距离索引是DistanceIndexRef1，当前图像的距离索引是DistanceIndexCur，Abs(DistanceIndexRef0-DistanceIndexCur) 和 Abs(DistanceIndexRef1-DistanceIndexCur)不相等。
   6) 当前子块的参考图像队列 0 中参考索引是 refIndexL0 的图像和参考图像队列 1 中参考索引是 refIndexL1 的图像是同一参考图像，且 L0 运动矢量和 L1 运动矢量相同。
b) 如果 DmvrFlag 的值为 1，由运动信息 MotionInfo、(x,y)、sWa 和 sHa 按 9.8.2.4 得到 predMatrixL0、predMatrixL1、gradXL0、gradYL0、gradXL1 和 gradYL1。
c) 否则，由运动信息MotionInfo、(x,y)、sWa和sHa按9.8.2.2得到predMatrixL0、predMatrixL1、gradXL0、gradYL0、gradXL1 和 gradYL1。
d) PredMatrixL0 由各个子块的 predMatrixL0 组成，PredMatrixL1 由各个子块的 predMatrixL1 组成，GradXL0 由各个子块的 gradXL0 组成，GradYL0 由各个子块的 gradYL0 组成，GradXL1 由各个子块的 gradXL1 组成，GradYL1 由各个子块的 gradYL1 组成。

否则，如果当前编码单元的编码单元子类型是‘P_Skip_Etmvp’‘P_Direct_Etmvp’‘B_Skip_Etmvp’或‘B_Direct_Etmvp’，则DmvrFlag等于0，令i和j是当前8×8子块的水平和垂直索引值，i=0～((W>>3)-1),j=0～((H>>3)-1)，按以下方法导出当前预测单元的预测样本。

a) 当前子块的运动信息 MotionInfo 是 MotionArray[i][j]，当前子块左上角的坐标(x,y)等于(xE+8*i,yE+8*j)，子块的宽度 sWa 和高度 sHa 均等于 8。
b) 由运动信息 MotionInfo、(x,y)、sWa 和 sHa 按 9.8.2.2 得到 predMatrixL0、predMatrixL1、gradXL0、gradYL0、gradXL1 和 gradYL1。
c) PredMatrixL0 由各个子块的 predMatrixL0 组成，PredMatrixL1 由各个子块的 predMatrixL1 组成，GradXL0 由各个子块的 gradXL0 组成，GradYL0 由各个子块的 gradYL0 组成，GradXL1 由各个子块的 gradXL1 组成，GradYL1 由各个子块的 gradYL1 组成。

否则，如果当前编码单元子类型是‘P_Skip_SbTemporal’‘P_Direct_SbTemporal’‘B_Skip_SbTemporal’或‘B_Direct_SbTemporal’，则将当前预测单元划分为4个子块（见图27，图中的数字是子块的索引值），按以下方法导出当前预测单元的预测样本。

a) 对于索引值是 i 的子块，运动信息 MotionInfo 是 MotionArray[i]，左上角的坐标(x,y)等于(xE+(W/2*(i%2)), yE+(H/2*(i/2)))，宽度 sWa 和高度 sHa 分别等于 W/2 和 H/2。如果满足以下条件之一，则 DmvrFlag 的值为 0；否则，DmvrFlag 的值为 1。
   1) DmvrEnableFlag的值为0。
   2) 当前子块的预测参考模式不是‘PRED_List01’。

3) 当前预测单元的InterPcFlag的值为1。
4) 当前子块的参考图像队列0中参考索引是RefIndexL0的图像和参考图像队列1中参考索引是RefIndexL1的图像均位于当前图像显示顺序之前或之后。
5) 当前子块的参考图像队列0中参考索引是RefIndexL0的图像的图像距离索引是DistanceIndexRef0，当前子块的参考图像队列1中参考索引是RefIndexL1的图像的图像距离索引是DistanceIndexRef1，当前图像的距离索引是DistanceIndexCur，Abs(DistanceIndexRef0-DistanceIndexCur)和Abs(DistanceIndexRef1-DistanceIndexCur)不相等。
6) 当前子块的参考图像队列0中参考索引是refIndexL0的图像和参考图像队列1中参考索引是refIndexL1的图像是同一参考图像，且L0运动矢量和L1运动矢量相同。

b) 如果 DmvrFlag 的值为 1，由运动信息 MotionInfo、(x,y)、sWa 和 sHa 按 9.8.2.4 得到 predMatrixL0、predMatrixL1、gradXL0、gradYL0、gradXL1 和 gradYL1。
c) 否则，由运动信息MotionInfo、(x,y)、sWa和sHa按9.8.2.2得到predMatrixL0、predMatrixL1、gradXL0、gradYL0、gradXL1 和 gradYL1。
d) PredMatrixL0 由各个子块的 predMatrixL0 组成，PredMatrixL1 由各个子块的 predMatrixL1 组成，GradXL0 由各个子块的 gradXL0 组成，GradYL0 由各个子块的 gradYL0 组成，GradXL1 由各个子块的 gradXL1 组成，GradYL1 由各个子块的 gradYL1 组成。

否则，如果当前编码单元子类型是‘B_Skip_Awp’‘B_Direct_Awp’‘P_Skip_Awp’或‘P_Direct_Awp’，按以下方法导出PredMatrixAwp0以及PredMatrixAwp1。

a) 由运动信息 MotionInfoAwp0、(xE,yE)、W 和 H 按 9.8.2.2 得到 PredMatrixAwp0。如果预测参考模式 interAwpPredRefMode0 是‘PRED_List0’，令 predMatrixAwp0 等于 predMatrixL0；否则，令 PredMatrixAwp0 等于 predMatrixL1。
b) 由运动信息 MotionInfoAwp1、(xE,yE)、W 和 H 按 9.8.2.2 得到 PredMatrixAwp1。如果预测参考模式 interAwpPredRefMode1 是‘PRED_List0’，令 predMatrixAwp1 等于 predMatrixL0；否则，令 PredMatrixAwp1 等于 predMatrixL1。

否则，按以下步骤导出predMatrixL0和predMatrixL1。

a) 如果满足以下条件之一，则 DmvrFlag 的值为 0；否则，DmvrFlag 的值为 1。
1) DmvrEnableFlag的值为0。
2) 当前预测单元的DirectFlag的值为0且SkipFlag的值为0。
3) 当前预测单元的InterPcFlag、UmveFlag、AffineFlag、AwpFlag或EtmvpFlag中有一个的值为1。
4) 当前预测单元的预测参考模式不是“PRED_List01”。
5) 当前预测单元的高度或宽度小于8。
6) 当前预测单元的高度等于4且宽度等于8。
7) 当前预测单元的高度等于8且宽度等于4。
8) 当前预测单元的参考图像队列0中参考索引是RefIndexL0的图像和参考图像队列1中参考索引是RefIndexL1的图像均位于当前图像显示顺序之前或之后。
9) 当前预测单元的参考图像队列0中参考索引是RefIndexL0的图像的图像距离索引是DistanceIndexRef0，当前预测单元的参考图像队列1中参考索引是RefIndexL1的图像的图像距离索引是DistanceIndexRef1，当前图像的距离索引是DistanceIndexCur，Abs(DistanceIndexRef0-DistanceIndexCur)和Abs(DistanceIndexRef1-DistanceIndexCur)不相等。

10) 当前预测单元的参考图像队列0中参考索引是refIndexL0的图像和参考图像队列1中参考索引是refIndexL1的图像是同一参考图像，且L0运动矢量和L1运动矢量相同。

b) 如果 DmvrFlag 的值为 1，由当前编码单元的运动信息 MotionInfo、(xE,yE)、W 和 H 按 9.8.2.4 得到 predMatrixL0、predMatrixL1、gradXL0、gradYL0、gradXL1 和 gradYL1。

c) 否则，由当前编码单元的运动信息MotionInfo、(xE,yE)、W和H按9.8.2.2得到predMatrixL0、predMatrixL1、gradXL0 gradYL0、gradXL1 和 gradYL1。

d) PredMatrixL0 由各个子块的 predMatrixL0 组成，PredMatrixL1 由各个子块的 predMatrixL1 组成，GradXL0 由各个子块的 gradXL0 组成，GradYL0 由各个子块的 gradYL0 组成，GradXL1 由各个子块的 gradXL1 组成，GradYL1 由各个子块的 gradYL1 组成。

9.8.2.2 导出普通预测样本

输入：运动信息motionInfo(mvE0，mvE1，refIndexL0，refIndexL1，interPredRefMode)，左上角样本位置(xE,yE)，以及当前编码单元的宽度W和高度H。

输出：预测样本矩阵predMatrixL0和predMatrixL1。

对于亮度块，执行以下操作：

——如果当前块的预测参考模式 interPredRefMode 等于‘PRED_List0’，则亮度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考索引为 RefIndexL0 的参考图像的 1/4 精度亮度样本矩阵中(xPos,yPos)位置的样本值。其中 MvE 是当前预测单元的 L0 运动矢量。

```
xPos = ((xE + x) << 2) + MvE->x
yPos = ((yE + y) << 2) + MvE->y
```

——如果 interPredRefMode 等于‘PRED_List1’，则亮度预测样本矩阵 predMatrixL1 的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIndexL1 的 1/4 精度亮度样本矩阵中(xPos,yPos)位置的样本值。其中 MvE 是当前预测单元的 L1 运动矢量。

```
xPos = ((xE + x) << 2) + MvE->x
yPos = ((yE + y) << 2) + MvE->y
```

——如果 interPredRefMode 等于‘PRED_List01’，则亮度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考帧索引为 RefIndexL0 的 1/4 精度亮度样本矩阵中(x0Pos,y0Pos)位置的样本值;predMatrixL1 的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIndexL1 的 1/4 精度亮度样本矩阵中(x1Pos,y1Pos)位置的样本值。其中 MvE0 和 MvE1 分别是当前预测单元的 L0 运动矢量和 L1 运动矢量。

```
x0Pos = ((xE + x) << 2) + MvE0->x
y0Pos = ((yE + y) << 2) + MvE0->y
x1Pos = ((xE + x) << 2) + MvE1->x
y1Pos = ((yE + y) << 2) + MvE1->y
```

对于色度块，执行以下操作：

——如果 interPredRefMode 等于‘PRED_List0’，则色度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考索引为 RefIndexL0 的 1/8 精度色度样本矩阵中(xPos,yPos)位置的样本值。其中 MvE 是当前预测单元的 L0 运动矢量。

```
xPos = ((xE + 2 * x) << 2) + MvE->x
yPos = ((yE + 2 * y) << 2) + MvE->y
```

——如果 interPredRefMode 等于‘PRED_List1’，则色度预测样本矩阵 predMatrixL1 的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIndexL1 的 1/8 精度色度样本矩阵中(xPos,yPos)位置的样本值。其中 MvE 是当前预测单元的 L1 运动矢量。

```
xPos = ((xE + 2 * x) << 2) + MvE->x
yPos = ((yE + 2 * y) << 2) + MvE->y
```

——如果 interPredRefMode 等于‘PRED_List01’，则色度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考索引为 RefIndexL0 的 1/8 精度色度样本矩阵中(x0Pos,y0Pos)位置的样本值;predMatrixL1 的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIndexL1 的 1/8 精度色度样本矩阵中(x1Pos,y1Pos)位置的样本值。其中 MvE0 和 MvE1 分别是当前预测单元的 L0 运动矢量和 L1 运动矢量。

```
x0Pos = ((xE + 2 * x) << 2) + MvE0->x
y0Pos = ((yE + 2 * y) << 2) + MvE0->y
x1Pos = ((xE + 2 * x) << 2) + MvE1->x
y1Pos = ((yE + 2 * y) << 2) + MvE1->y
```

参考图像的亮度1/4精度样本矩阵和色度1/8精度样本矩阵中各个位置的值分别按9.8.2.5.2和9.8.2.6.2插值得到。参考图像外的整数样本应使用该图像内距离该样本最近的整数样本（边缘或角样本）代替，即运动矢量能指向参考图像外的样本。x的取值范围为0～W-1，y的取值范围为0～H-1。

9.8.2.3 导出仿射预测样本

9.8.2.3.1 概述

输入：运动信息motionInfo(mvE0,mvE1,refIndexL0,refIndexL1,interPredRefMode)，当前编码单元的宽度W和高度H，左上角样本位置(xE,yE)，色度L0运动矢量mvC0，色度L1运动矢量mvC1，仿射改善信息 asrInfo0(ASRFlag0,ASRHorFlag0,ASRVerFlag0,dMvA0,dMvB0,dMvC0,dMvN0) 和 asrInfo1(ASRFlag1,ASRHorFlag1,ASRVerFlag1,dMvA1,dMvB1,dMvC1,dMvN1)，以及参数X（X是A、B、C或N，见图25）。

输出：预测样本矩阵predMatrixL0和predMatrixL1。

由motionInfo、(xE,yE)、mvC0、mvC1、W和H按9.8.2.3.2得到预测样本矩阵predMatrixL0和predMatrixL1。

如果ASRFlag0的值为1，由参考队列0、mvE0、refIndexL0、predMatrixL0、ASRHorFlag0、ASRVerFlag0、dMvX0、(xE,yE)、W和H按9.8.2.3.3得到改善后的预测样本矩阵predMatrixL0。

如果ASRFlag1的值为1，由参考队列1、mvE1、refIndexL1、predMatrixL1、ASRHorFlag1、ASRVerFlag1、dMvX1、(xE,yE)、W和H按9.8.2.3.3得到改善后的预测样本矩阵predMatrixL1。

9.8.2.3.2 导出仿射预测样本的方法

输入：运动信息motionInfo(mvE0,mvE1,refIndexL0,refIndexL1,interPredRefMode)，左上角样本位置(xE,yE)，色度L0运动矢量mvC0，色度L1运动矢量mvC1，宽度W和高度H。

对于亮度编码块，执行以下操作：

——如果当前块的预测参考模式 interPredRefMode 等于‘PRED_List0’，则亮度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考索引为 refIndexL0 的参考图像的 1/16 精度亮度样本矩阵中(xPos,yPos)位置的样本值。其中 MvE 是当前预测单元的 L0 运动矢量。

```
xPos = ((xE + x) << 4)+MvE->x
yPos = ((yE + y) << 4) + MvE->y
```

——如果 interPredRefMode 等于‘PRED_List1’，则亮度预测样本矩阵 predMatrixL1 的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIndexL1 的 1/16 精度亮度样本矩阵中(xPos,yPos)位置的样本值。其中 MvE 是当前预测单元的 L1 运动矢量。

```
xPos = ((xE + x) << 4)+MvE->x
yPos = ((yE + y) << 4) + MvE->y
```

——如果 interPredRefMode 等于‘PRED_List01’，则亮度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考帧索引为 RefIndexL0 的 1/16 精度亮度样本矩阵中(x0Pos,y0Pos)位置的样本值；predMatrixL1 的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIndexL1 的 1/16 精度亮度样本矩阵中(x1Pos,y1Pos)位置的样本值。其中 MvE0 和 MvE1 分别是当前预测单元的 L0 运动矢量和 L1 运动矢量。

```
x0Pos = ((xE + x) << 4) + MvE0->x
y0Pos = ((yE + y) << 4) + MvE0->y
x1Pos = ((xE + x) << 4) + MvE1->x
y1Pos = ((yE + y) << 4) + MvE1->y
```

对于色度编码块，执行以下操作：

——如果 interPredRefMode 等于‘PRED_List0’，则色度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考索引为 RefIndexL0 的 1/32 精度色度样本矩阵中(xPos,yPos)位置的样本值。

```
xPos = ((xE + 2 * x) << 4) + mvC0->x
yPos = ((yE + 2 * y) << 4) + mvC0->y
```

——如果 interPredRefMode 等于‘PRED_List1’，则色度预测样本矩阵 predMatrixL1 的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIndexL1 的 1/32 精度色度样本矩阵中(xPos,yPos)位置的样本值。

```
xPos = ((xE + 2 * x) << 4) + mvC1->x
yPos = ((yE + 2 * y) << 4) + mvC1->y
```

——如果 interPredRefMode 等于‘PRED_List01’，则色度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考索引为 RefIndexL0 的 1/32 精度色度样本矩阵中 x0Pos,y0Pos)位置的样本值；predMatrixL1 的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIndexL1 的 1/32 精度色度样本矩阵中 x1Pos,y1Pos)位置的样本值。

```
x0Pos = ((xE + 2 * x) << 4) + nvC0->x
```

```
y0Pos = ((yE + 2 * y) << 4) + mvC0->y
x1Pos = ((xE + 2 * x) << 4) + mvC1->x
y1Pos = ((yE + 2 * y) << 4) + mvC1->y
```

参考图像的亮度1/16精度样本矩阵和色度1/32精度样本矩阵中各个位置的值分别按9.8.2.5.4和9.8.2.6.3插值得到。参考图像外的整数样本应使用该图像内距离该样本最近的整数样本（边缘或角样本）代替，即运动矢量能指向参考图像外的样本，x的取值范围是0～W-1，y的取值范围是0～H-1。

9.8.2.3.3 改善仿射预测样本

输入：参考队列K(K为0或1),运动矢量mv,参考索引refIndex,预测样本矩阵predMatrix,ASRHorFlag，ASRVerFlag，子块偏差矩阵dMv，左上角样本位置(xE,yE)，宽度W和高度H。

输出：改善后的预测样本矩阵predMatrixLK（K为0或1）。

第1步，设predMatrixTemp是(W+2)×(H+2)的临时预测样本矩阵，refLumaMatrix是参考图像队列K中参考帧索引为refIndex的参考图像亮度样本矩阵。按以下方法给predMatrixTmp赋值。

```
for (i=0; i<=W+1; i++) {
    for (j=0; j<=H+1; j++) {
        xPos = xE  -  1 + i + ((mv->x + 7) >> 4)
        yPos = yE  -  1 + j + ((mv->y + 7) >> 4)
        predMatrixTemp[i][j] = refLumaMatrix[xPos][yPos]
    }
}

for (i=0; i<W; i++) {
    for (j=0; j<H; j++) {
        predMatrixTemp[i+1][j+1] = predMatrix[i][j]
    }
}
```

第2步，计算predMatrix[i][j]。

a) 如果 ASRHorFlag 等于 1 且 ASRVerFlag 等于 1：

```
for (i=0; i<W; i++) {
    for (j=0; j<H; j++) {
        predMatrix[i][j] = (PredMatrixTemp[i][j] * (-dMv[i][j][0]  -  dMv[i][j][1]) +
                        PredMatrixTemp[i+1][j] * ((-dMv[i][j][1]) << 3) +
                        PredMatrixTemp[i+2][j] * (dMv[i][j][0]  -  dMv[i][j][1]) +
                        PredMatrixTemp[i][j+1] * ((-dMv[i][j][0]) << 3) +
                        PredMatrixTemp[i+1][j+1] * (1 << 9) +
                        PredMatrixTemp[i+2][j+1] * (dMv[i][j][0] << 3) +
                        PredMatrixTemp[i][j+2] * (-dMv[i][j][0] + dMv[i][j][1]) +
                        PredMatrixTemp[i+1][j+2] * (dMv[i][j][1] << 3) +
                        PredMatrixTemp[i+2][j+2] * (dMv[i][j][0] + dMv[i][j][1]) + (1 << 8)) >> 9
    }
```

```
}
```

b) 否则，如果 ASRHorFlag 等于 1：

```
for (i=0; i<W; i++) {
    for (j=0; j<H; j++) {
        predMatrix[i][j] = (PredMatrixTemp[i][j+1] * (-dMv[i][j][0]) +
                        PredMatrixTemp[i+1][j+1] * (1 << 6) +
                        PredMatrixTemp[i+2][j+1] * dMv[i][j][0] + (1 << 5)) >> 6
    }
}
```

c) 否则：

```
for (i=0; i<W; i++) {
    for (j=0; j<H; j++) {
        predMatrix[i][j] = (PredMatrixTemp[i+1][j] * (-dMv[i][j][1]) +
                        PredMatrixTemp[i+1][j+1] * (1 << 6) +
                        PredMatrixTemp[i+1][j+2] * dMv[i][j][1] + (1 << 5)) >> 6
    }
}
```

第3步，限制predMatrix[i][j]的取值范围。

```
for (i=0; i<W; i++) {
    for (j=0; j<H; j++) {
        predMatrix[i][j] = Clip3(0, (1 << BitDepth) - 1, predMatrix [i][j])
    }
}
```

9.8.2.4 导出解码端运动改良的样本

9.8.2.4.1 概述

输入：运动信息motionInfo(interPredRefMode,mvE0,mvE1,RefIndexL0,RefIndexL1)，左上角样本位置(xCb,yCb)，宽度nNbW和高度nNbH。

输出：改良后的预测样本矩阵predMatrixL0和predMatrixL1。

按9.8.2.4.2得到改良后的预测样本矩阵predMatrixL0和predMatrixL1。

9.8.2.4.2 解码端运动矢量的改良方法

输入：运动信息motionInfo(interPredRefMode,mvE0,mvE1,RefIndexL0,RefIndexL1)，左上角样本位置(xCb,yCb)，宽度nNbW和高度nNbH。

输出：改良后的预测样本矩阵predMatrixL0和predMatrixL1。

第1步，首先，将运动矢量mvE0和mvE1水平分量范围限制在((-LcuSize-4 - xCb)<<2)到((PicWidthInLuma+LcuSize+4 - xCb - nNbW)<<2)之间，垂直分量范围限制在((-LcuSize- 4 - yCb)<<2)

和((PicHeightInLuma+LcuSize+4 - yCb - nNbH)<<2)之间。然后，将运动矢量mvE0和mvE1水平分量和垂直分量限制在-32768～32767内。

由mvE0和mvE1得到起始运动矢量sMV0和sMV1。

```
signMV0->x = mvE0->x >= 0 ? 1 : (-1)
signMV0->y = mvE0->y >= 0 ? 1 : (-1)
signMV1->x = mvE1->x >= 0 ? 1 : (-1)
signMV1->y = mvE1->y >= 0 ? 1 : (-1)

absMV0->x = Clip3(0, 32763, Abs(mvE0->x))
absMV0->y = Clip3(0, 32763, Abs(mvE0->y))
absMV1->x = Clip3(0, 32763, Abs(mvE1->x))
absMV1->y = Clip3(0, 32763, Abs(mvE1->y))

xFracMV0->x = absMV0->x & 3
yFracMV0->y = absMV0->y & 3
xFracMV1->x = absMV1->x & 3
yFracMV1->y = absMV1->y & 3

sMV0->x = (((absMV0->x >> 2) << 2) + (((xFracMV0->x > 2) ? 1 : 0) << 2)) * signMV0->x
sMV0->y = (((absMV0->y >> 2) << 2) + (((yFracMV0->y > 2) ? 1 : 0) << 2)) * signMV0->y
sMV1->x = (((absMV1->x >> 2) << 2) + (((xFracMV1->x > 2) ? 1 : 0) << 2)) * signMV1->x
sMV1->y = (((absMV1->y >> 2) << 2) + (((yFracMV1->y > 2) ? 1 : 0) << 2)) * signMV1->y
sMV0->x = Clip3(-32760, 32756, sMV0->x)
sMV0->y = Clip3(-32760, 32756, sMV0->y)
sMV1->x = Clip3(-32760, 32756, sMV1->x)
sMV1->y = Clip3(-32760, 32756, sMV1->y)
```

第2步，设subW和subH分别是子块的宽度和高度，将当前块的亮度图像块划分为subWNum×subWNum个不重叠且位置相邻的子块。计算第i个子块的左上角样本位置(xCbSub,yCbSub)，子块的运动矢量ssubMV0和ssubMV1。其中i的取值范围为0～subWNum×subWNum-1。

```
subW = nNbW < 16 ? nNbW : 16
subH = nNbH < 16 ? nNbH : 16
subWNum= nNbW / subW
subHNum= nNbH / subH
xCbSub= subW * (subn % subWNum)
yCbSub = subH * (subn / subWNum)
ssubMV0->x= sMV0->x
ssubMV0->y= sMV0->y
ssubMV1->x= sMV1->x
ssubMV1->y= sMV1->y
```

第 3 步，由(xCb,yCb)、(xCbSub,yCbSub)和(subW,subH)，对 ssubMV0 和 ssubMV1 进行限制。

```
if ((ssubMV0->x < ((-128 - xCb) << 2)) || (ssubMV0->x > ((PicWidthInLuma + 128 - nNbW - xCb) << 2)) {
    ssubMV0->x = Clip3(((-128 - xCb) << 2), ((PicWidthInLuma + 128 -nNbW - xCb) << 2), ssubMV0->x)
}
if ((ssubMV0->y < ((-128 - yCb) << 2)) || (ssubMV0->y > ((PicHeightInLuma + 128 - nNbH - yCb) << 2)) {
    ssubMV0->y = Clip3(((-128 - yCb) << 2), ((PicHeightInLuma + 128 -nNbH - yCb) <<2 ), ssubMV0->y)
if ((ssubMV1->x < ((-128 - xCb) << 2)) || (ssubMV1->x > ((PicWidthInLuma + 128 -nNbW - xCb) << 2)) {
    ssubMV1->x = Clip3(((-128 - xCb) << 2), ((PicWidthInLuma + 128 -nNbW - xCb) << 2), ssubMV1->x)
}
if ((ssubMV1->y < ((-128 - yCb) << 2)) || (ssubMV1->y > ((PicHeightInLuma + 128 -nNbH - yCb) << 2)) {
    ssubMV1->y = Clip3(((-128 - yCb) << 2), ((PicHeightInLuma + 128 -nNbH - yCb) <<2 ), ssubMV1->y)
}
```

第4步，由(xCb,yCb)、(xCbSub,yCbSub)、ssubMV0和ssubMV1按9.8.2.4.3得到子块的亮度搜索窗口样本矩阵predSearchMatrixL0和predSearchMatrixL1，亮度搜索窗口样本矩阵的宽度和高度分别是SearchW和SearchH；按9.8.2.4.4得到子块的色度搜索窗口样本矩阵predSearchMatrixC0和predSearchMatrixC1，色度搜索窗口样本矩阵的宽度和高度分别是SearchWC和SearchHC。

```
SearchW = subW+FilterSize+2*IterCount
SearchH = subH+FilterSize+2*IterCount
SearchWC = (subW>>1)+FilterSizeC+2*IterCount
SearchHC = (subH>>1)+FilterSizeC+2*IterCount
```

其中IterCount等于2，FilterSize等于8，FilterSizeC等于4，见图33。

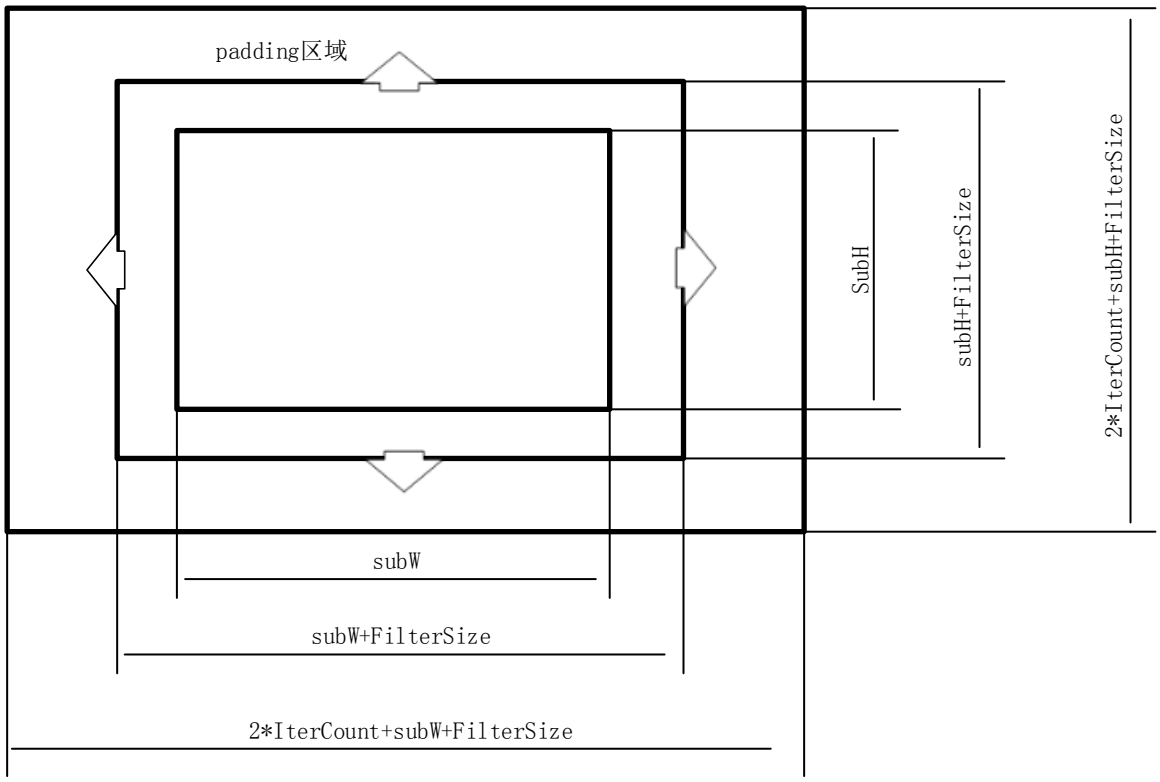


图33　亮度/色度搜索窗口样本矩阵生成方式示意图

第5步，遍历index=0～20，计算5×5的子块绝对值误差和矩阵sadMatrix。

```
LeftFilter = (FilterSize >> 1) - 1
searchStart = IterCount + LeftFilter
xPos = arrayPosX[index]
yPos = arrayPosY[index]
sadTmp = 0
for (x=0; x<subW; x++) {
    for (y=0; y<subH; y++) {
        sadTmp += Abs(predSearchMatrixL0[x+SearchStart+xPos][y+searchStart+yPos] -
predSearchMatrixL1[x+searchStart-xPos][y+SearchStart-yPos])
    }
}
sadMatrix[xPos+2][yPos+2] = sadTmp
```

其中arrayPosX[index]和arrayPosY[index]的值见表113。

表113 arrayPosX 和 arrayPosY 的值

index的值	arrayPosX的值	arrayPosY的值
0	0	0
1	0	1
2	0	-1
3	1	0
4	-1	0
5	0	2
6	1	1
7	2	0
8	1	-1
9	0	-2
10	-1	-1
11	-2	0
12	-1	1
13	1	2
14	2	1
15	-1	2
16	-2	1
17	1	-2
18	2	-1
19	-1	-2
20	-2	-1

第6步，在sadMatrix中查找目标位置arrayPosBestX和arrayPosBestY。

```
startX = 2
startY = 2
arrayPosBestX = startX
arrayPosBestY = startY
for (j=0; j<3; j++){
    sadEMin = sadMatrix[startX][startY]
    best0 = 0
    for (index=1; index<5; index++) {
        if (startX+arrayPosX[idx]>= 0 && startX+arrayPosX[idx] < 5 && startY+arrayPosY[idx]>= 0 &&
startY+arrayPosY[idx] < 5) {
            if (sadEMin > sadMatrix[startX+arrayPosX[index]][startY+arrayPosY[index]]) {
                sadEMin = sadMatrix[startX+arrayPosX[index]][startY+arrayPosY[index]]
                best0 = index
            }
        }
    }
    arrayPosBestX = startX + arrayPosX[best0]
    arrayPosBestY = startY + arrayPosY[best0]
    if (sadEMin == 0 || best0 == 0 || j == 2) {
        break
    }
    startX += arrayPosX[best0]
    startY += arrayPosY[best0]
}
```

第7步，从arrayPosBestX和arrayPosBestY得到子块整数像素位置偏移subnMV0和subnMV1。

```
subnMV0->x-= arrayPosBestX - 2
subnMV0->y = arrayPosBestY - 2
subnMV1->x-= 2 - arrayPosBestX
subnMV1->y = 2 - arrayPosBestY
```

第8步，计算子块亚像素位置偏移subqpelX和subqpelY。将subqpelX和subqpelY初始化为0，如果以下条件均满足，由sadH0、sadH1和sadH2按9.8.2.4.5得到subqpleX；由sadV0、sadV1和sadV2按9.8.2.4.5得到subqpleY。其中，sadH0、sadH1、sadH2、sadV0、sadV1和sadV2的计算如下：

a) sadH1/sadV1 不等于 0；
b) array_pos_best_x 大于或等于-1 且小于或等于 1，同时 array_pos_best_y 大于或等于-1 且小于或等于 1。

```
sadH0 = sadMatrix[arrayPosBestX-1][ arrayPosBestY]
sadH1 = sadMatrix[arrayPosBestX][arrayPosBestY]
sadH2 = sadMatrix[arrayPosBestX+1][arrayPosBestY]
sadV0 = sadMatrix[arrayPosBestX][arrayPosBestY-1]
sadV1 = sadMatrix[arrayPosBestX][arrayPosBestY]
sadV2 = sadMatrix[arrayPosBestX][arrayPosBestY+1]
```

第9步，根据子块整数像素位置偏移subnMV0和subnMV1，子块亚像素位置偏移subqpelX和subqpelY，分别从亮度和色度搜索窗口样本矩阵predSearchMatrixL0、predSearchMatrixL1、predSearchMatrixC0、和predSearchMatrixC1得到亮度预测样本矩阵和色度预测样本矩阵。

对predMatrixL0的操作如下：

a) 限制 subnMV0、subqpelX 和 subqpelY 的取值范围。

```
subMVTempL0->x = ((subnMV0->x) << 2) + subqpelX + sMV0->x
if ((subMVTempL0->x < ((-128 - xCb - xCbSub) << 2)) || (subMVTempL0->x > ((PicWidthInLuma + 128 - subW - xCb - xCbSub) << 2)) {
    subMVTempL0->x = Clip3(((-128 - xCb - xCbSub) << 2), ((PicWidthInLuma + 128 - subW - xCb - xCbSub) << 2), subMVTempL0->x)
    subqpelX = subMVTempL0->x & 0x3
    subnMV0->x = (subMVTempL0->x >> 2) - (sMV0->x >> 2)
    subnMV0->x = Clip3(-IterCount, IterCount, subnMV0->x)
}
subMVTemp->L0->y = ((subnMV0->y) << 2) + subqpelY + sMV0->y
if ((subMVTempL0->y < ((-128 - yCb - yCbSub) << 2)) || (subMVTempL0->y > ((PicHeightInLuma + 128 - subH - yCb - yCbSub) << 2)) {
    subMVTempL0->y = Clip3(((-128 - yCb - yCbSub) << 2), ((PicHeightInLuma + 128 - subH - yCb - yCbSub) <<2 ), subMVTempL0->y)
    subqpelY = subMVTempL0->y & 0x3
    subnMV0->y = (subMVTempL0->y >> 2) - (sMV0->y >> 2)
    subnMV0->y = Clip3(-IterCount, IterCount, subnMV0->y)
}
```

b) 亮度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x+xCbSub][y+yCbSub]的值等于 predSearchMatrixL0 的 1/4 精度亮度样本矩阵中位置为(x0,y0)的样本值。

```
x0 = ((x + IterCount + LeftFilter + subnMV0->x) <<2) + subqpelX
y0 = ((y + IterCount + LeftFilter + subnMV0->y) <<2) + subqpelY
```

c) 色度预测样本矩阵 predMatrixL0 的元素 predMatrixL0[x+(xCbSub>>1)][y+(yCbSub>>1)]的值等于 predSearchMatrixC0 的 1/8 精度色度样本矩阵中位置为(x0,y0)的样本值

```
x0 = ((2 * x + 2 * IterCount + 2 * LeftFilterC + 2 * subnMV0->x) << 2) + (subqpelX << 1)
y0 = ((2 * y + 2 * IterCount + 2 * LeftFilterC + 2 * subnMV0->y) << 2) + (subqpelY << 1)
```

对predMatrixL1的操作如下：

a) 限制 subnMV1、subqpelX 和 subqpelY 的取值范围。

```
subMVTempL1->x = ((subnMV1->x) << 2) + subqpelX + sMV1->x
if ((subMVTempL1->x < ((-128 - xCb - xCbSub) << 2)) || (subMVTempL1->x > ((PicWidthInLuma + 128 - subW - xCb - xCbSub) << 2)) {
    subMVTempL1->x = Clip3(((-128 - xCb - xCbSub) << 2), ((PicWidthInLuma + 128 - subW - xCb - xCbSub) << 2), subMVTempL1->x)
```

```
    subqpelX = subMVTempL1->x & 0x3
    subnMV1->x = (subMVTempL1->x >> 2) - (sMV1->x >> 2)
    subnMV1->x = Clip3(-IterCount, IterCount, subnMV1->x)
}
subMVTemp->L1->y = ((subnMV1->y) << 2) + subqpelY + sMV1->y
if ((subMVTempL1->y < ((-128 - yCb - yCbSub) << 2)) || (subMVTempL1->y > ((PicHeightInLuma + 128 - subH
- yCb - yCbSub) << 2)) {
    subMVTempL1->y = Clip3(((-128 - yCb - yCbSub) << 2), ((PicHeightInLuma + 128 - subH - yCb - yCbSub)
<<2 ), subMVTempL1->y)
    subqpelY = subMVTempL1->y & 0x3
    subnMV1->y = (subMVTempL1->y >> 2) - (sMV1->y >> 2)
    subnMV1->y = Clip3(-IterCount, IterCount, subnMV1->y)
}
```

b) 亮度预测样本矩阵 predMatrixL1 的元素 predMatrixL1[x+xCbSub][y+yCbSub]的值等于 predSearchMatrixL1 的 1/4 精度亮度样本矩阵中位置为(x1,y1)的样本值。

```
x0 = -(((x + IterCount + LeftFilter + subnMV1->x) <<2) - subqpelX)
y0 = -(((y + IterCount + LeftFilter + subnMV1->y) <<2) - subqpelY)
```

c) 色度预测样本矩阵 predMatrixL1 的元素 predMatrixL1[x+(xCbSub>>1)][y+(yCbSub>>1)]的值等于 predSearchMatrixC1 的 1/8 精度色度样本矩阵中位置为(x1,y1)的样本值。

```
x0 = ((2 * x + 2 * IterCount + 2 * LeftFilterC + 2 * subnMV1->x) << 2) - (subqpelX << 1)
y0 = ((2 * y + 2 * IterCount + 2 * LeftFilterC + 2 * subnMV1->y) << 2) - (subqpelY << 1)
```

predSearchMatrixL0、predSearchMatrixL1、predSearchMatrixC0、predSearchMatrixC1的亮度1/4精度样本矩阵和色度1/8精度样本矩阵中各个位置的值分别按9.8.2.5.2和9.8.2.6.2插值得到，paddingBuf外的整数样本应使用该paddingBuf内距离该样本最近的整数样本（边缘或角样本）代替。

9.8.2.4.3 生成亮度搜索窗口样本矩阵

输入：当前块左上角样本的位置(xCb,yCb)，当前子块左上角样本的位置(xCbSub,yCbSub)，当前子块的运动矢量ssubMV0和ssubMV1。

输出：亮度搜索窗口样本矩阵predSearchMatrixL0和predSearchMatrixL1。

第1步，设ref0LumaMatrix是参考图像队列0中参考帧索引为RefIndexL0的整数精度亮度样本矩阵，ref1LumaMatrix是参考图像队列1中参考帧索引为RefIndexL1的整数精度亮度样本矩阵。计算predSearchMatrixL0[x+IterCount][y+IterCount]和predSearchMatrixL1[x+IterCount][y+IterCount]。

```
LeftFilter = (FilterSize >> 1) - 1
for (x=0; x<subW+FilterSize; x++) {
    for (y=0; y<subH+FilterSize; y++) {
        x0Pos = (xCb + xCbSub + x - LeftFilter) + (ssubMV0->x >> 2)
        y0Pos = (yCb + yCbSub + y - LeftFilter) + (ssubMV0->y >> 2)
        predSearchMatrixL0[x+IterCount][y+IterCount] = ref0LumaMatrix[x0Pos][y0Pos]
        x1Pos = (xCb + xCbSub + x - LeftFilter) + (ssubMV1->x >> 2)
```

```
        y1Pos = (yCb + yCbSub + y  -  LeftFilter) + (ssubMV1->y >> 2)
        predSearchMatrixL1[x+IterCount][y+IterCount] = ref1LumaMatrix[x1Pos][y1Pos]
    }
}
```

第2步，计算predSearchMatrixL0[x1][y1]和predSearchMatrixL1[x1][y1]。

```
for (x1=0; x1<IterCount; x1++) {
    for (y1=IterCount; y1<subH+FilterSize+IterCount; y1++) {
        predSearchMatrixL0[x1][y1] = predSearchMatrixL0[IterCount][y1]
        predSearchMatrixL0[x1][y1] = predSearchMatrixL1[IterCount][y1]
    }
}
```

第3步，计算predSearchMatrixL0[x2][y2]和predSearchMatrixL1[x2][y2]。

```
for (x2= IterCount+subW+FilterSize; x2<2*IterCount+subW+FilterSize; x2++) {
    for (y2=InterCount; y2<subH+FilterSize+IterCount; y2++) {
        predSearchMatrixL0[x2][y2] = predSearchMatrixL0[IterCount+subW+FilterSize-1][y2]
        predSearchMatrixL0[x2][y2] = predSearchMatrixL1[IterCount+subW+FilterSize-1][y2]
    }
}
```

第4步，计算predSearchMatrixL0[x3][y3]和predSearchMatrixL1[x3][y3]。

```
for (x3= 0; x3<2*IterCount+subW+FilterSize; x3++) {
    for (y3=0; y3<IterCount; y3++) {
        predSearchMatrixL0[x3][y3] = predSearchMatrixL0[x3][IterCount]
        predSearchMatrixL0[x3][y3] = predSearchMatrixL1[x3][IterCount]
    }
}
```

第5步，计算predSearchMatrixL0[x4][y4]和predSearchMatrixL1[x4][y4]。

```
for (x4= 0; x4<2*IterCount+subW+FilterSize; x4++) {
    for (y4= IterCount+subH+FilterSize; y4<2*IterCount+subH+FilterSize; y4++) {
        predSearchMatrixL0[x4][y4] = predSearchMatrixL0[x4][IterCount+subH+FilterSize-1]
        predSearchMatrixL0[x4][y4] = predSearchMatrixL0[x4][IterCount+subH+FilterSize-1]
    }
}
```

9.8.2.4.4 生成色度搜索窗口样本矩阵

输入：当前块左上角样本的位置(xCb,yCb)，当前子块左上角样本的位置(xCbSub,yCbSub)，当前子块的运动矢量ssubMV0和ssubMV1。

输出：亮度搜索窗口样本矩阵predSearchMatrixC0和predSearchMatrixC1。

第1步，设refOChromaMatrix是参考图像队列0中参考帧索引为RefIndexL0的整数精度色度样本矩阵，ref1ChromaMatrix是参考图像队列1中参考帧索引为RefIndexL1的整数精度色度样本矩阵。计算predSearchMatrixL0[x+IterCount][y+IterCount]和predSearchMatrixL1[x+IterCount][y+IterCount]。

```
LeftFilterC = (FilterSizeC >> 1) - 1
for (x=0; x<(subW>>1)+FilterSizeC; x++) {
    for (y=0; y<(subH>>1)+FilterSizeC; y++) {
        x0Pos = ((xCb + xCbSub) >> 1) + x - LeftFilterC + (ssubMV0->x >> 3)
        y0Pos = ((yCb + yCbSub) >> 1) + y - LeftFilterC + (ssubMV0->y >> 3)
        predSearchMatrixC0[x+IterCount][y+IterCount] = ref0ChromaMatrix[x0Pos][y0Pos]
        x1Pos = ((xCb + xCbSub) >> 1) + x - LeftFilterC + (ssubMV1->x >> 3)
        y1Pos = ((yCb + yCbSub) >> 1) + y - LeftFilterC + (ssubMV1->y >> 3)
        predSearchMatrixC1[x+IterCount][y+IterCount] = ref1ChromaMatrix[x1Pos][y1Pos]
    }
}
```

第2步，计算predSearchMatrixC0[x1][y1]和predSearchMatrixC1[x1][y1]。

```
for (x1=0; x1<IterCount; x1++) {
    for (y1=IterCount; y1<(subH>>1)+FilterSizeC+IterCount; y1++) {
        predSearchMatrixC0[x1][y1] = predSearchMatrixC0[IterCount][y1]
        predSearchMatrixC1[x1][y1] = predSearchMatrixC1[IterCount][y1]
    }
}
```

第3步，计算predSearchMatrixC0[x2][y2]和predSearchMatrixC1[x2[y2]。

```
for (x2=IterCount+(subW>>1)+FilterSizeC; x2<2*IterCount+(subW>>1)+FilterSizeC; x2++) {
    for (y2=IterCount; y2<(subH>>1)+FilterSizeC+IterCount; y2++) {
        predSearchMatrixC0[x2][y2] = predSearchMatrixC0[IterCount+(subW>>1)+FilterSizeC-1][y2]
        predSearchMatrixC1[x2][y2] = predSearchMatrixC1[IterCount+(subW>>1)+FilterSizeC-1][y2]
    }
}
```

第4步，计算predSearchMatrixC0[x3][y3]和predSearchMatrixC1[x3[y3]。

```
for (x3=0; x3<2*IterCount+(subW>>1)+FilterSizeC; x3++) {
    for (y3=0; y3<IterCount; y3++) {
        predSearchMatrixC0[x3][y3] = predSearchMatrixC0[x3][IterCount]
        predSearchMatrixC1[x3][y3] = predSearchMatrixC1[x3][IterCount]
    }
}
```

第5步，计算predSearchMatrixC0[x4][y4]和predSearchMatrixC1[x4[y4]。

```
for (x4=0; x4<2*IterCount+(subW>>1)+FilterSizeC; x4++) {
    for (y4=IterCount+(subH>>1)+FilterSizeC; y4<2*IterCount+(subH>>1)+FilterSizeC; y4++) {
```

```
        predSearchMatrixC0[x4][y4] = predSearchMatrixC0[x4][IterCount+(subH>>1)+FilterSizeC-1]
        predSearchMatrixC1[x4][y4] = predSearchMatrixC0[x4][IterCount+(subH>>1)+FilterSizeC-1]
    }
}
```

9.8.2.4.5 计算亚像素位置偏移

输入：平均绝对误差和sad0、sad1和sad2。

输出：计算亚像素位置偏移deltaMv。

第1步，计算iNum和iDenom。

```
iNum = (sad0 - sad2) << 4)
iDenom = ((sad0 + sad2 - (sad1 << 1)))
```

第2步，如果iDenom不等于0，计算deltaMv。

```
if ((sad0 != sad1) && (sad2 != sad1)) {
    sign = 0
    if (iNum < 0) {
        sign = 1
        iNum = -iNum
    }
    q = 0
    iDenom = (iDenom << 3)
    if (iNum >= iDenom) {
        iNum -= iDenom
        q++
    }
    q = (q << 1)
    iDenom = (iDenom >> 1)
    if (iNum >= iDenom) {
        iNum -= iDenom
        q++
    }
    q = (q << 1)
    if (iNum >= (iDenom >> 1)) {
        q++
    }
    if (sign != 0) {
        deltaMv = -q
    }
    else {
        deltaMv = q
    }
```

```
}
else if (sad0 == sad1) {
    deltaMv = -8
}
else {
    deltaMv = 8
}
deltaMv >> 2
```

9.8.2.5 亮度样本的插值过程

9.8.2.5.1 概述

如果当前预测单元的AffineFlag的值为0，先9.8.2.5.2进行亮度样本插值，再按9.8.2.5.3进行梯度计算；否则，按9.8.2.5.4进行亮度样本插值。

9.8.2.5.2 普通亮度样本的插值过程

图34给出了亮度样本矩阵中整数样本、1/2样本和1/4样本的位置，其中用大写字母标记的是整数样本位置，用小写字母标记的是1/2和1/4样本位置。这些样本位置与其在1/4精度的亮度样本矩阵中的坐标(f_x,f_y)的对应关系见表114，其中xFracL等于f_x & 3,yFracl等于f_y & 3。

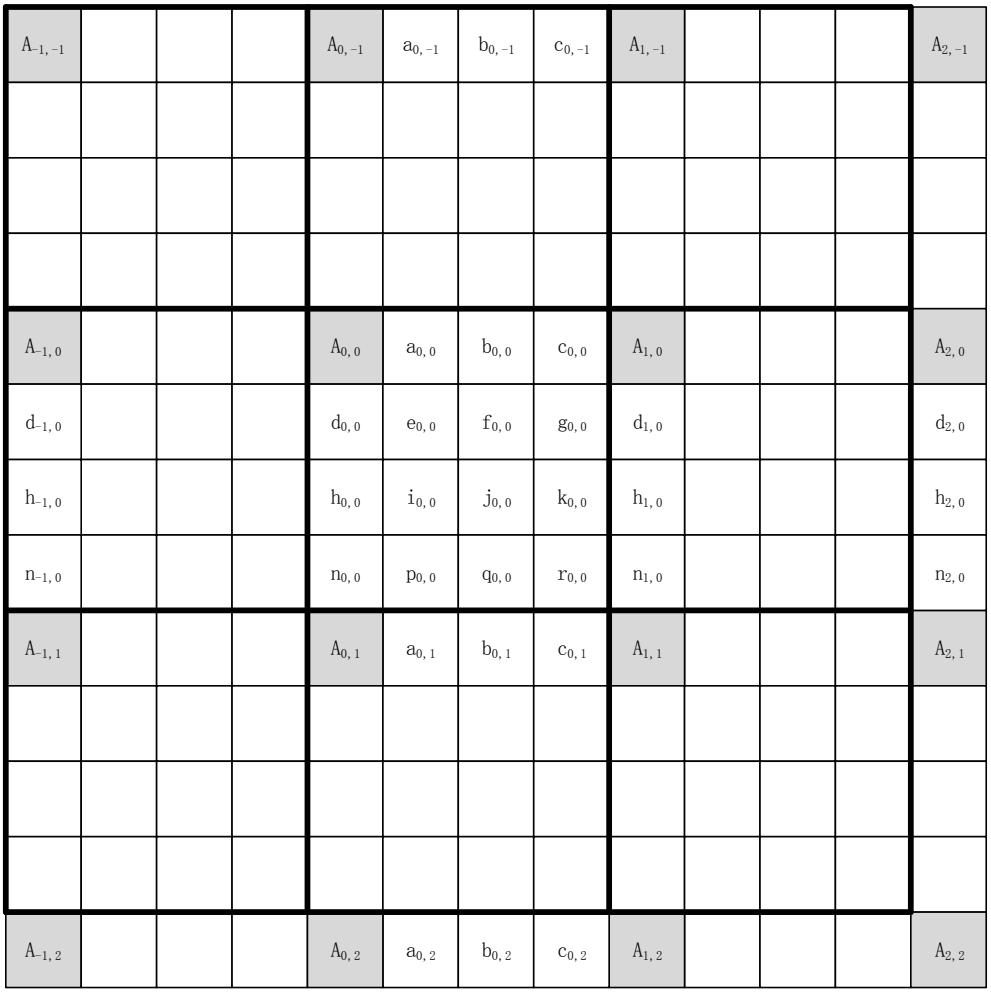


图34　整数样本、1/2 样本和 1/4 样本的位置

表114　预测样本矩阵元素

xFracL的值	yFracL的值	图34中的样本位置
0	0	$A_{0,0}$
0	1	$d_{0,0}$
0	2	$h_{0,0}$
0	3	$n_{0,0}$
1	0	$a_{0,0}$
1	1	$e_{0,0}$
1	2	$i_{0,0}$
1	3	$p_{0,0}$
2	0	$b_{0,0}$
2	1	$f_{0,0}$
2	2	$j_{0,0}$
2	3	$q_{0,0}$
3	0	$c_{0,0}$
3	1	$g_{0,0}$

表 114（续）

xFracL的值	yFracL的值	图34中的样本位置
3	2	$k_{0,0}$
3	3	$r_{0,0}$

样本位置 $a_{0,0}$、$b_{0,0}$及 $c_{0,0}$的预测值由水平方向距离插值点最近的 8 个整数值滤波得到，预测值获取方式如下：

$$a0,0 = Clip1((-A_{-3,0} + 4*A_{-2,0} - 10*A_{-1,0} + 57*A_{0,0} + 19*A_{1,0} - 7*A_{2,0} + 3*A_{3,0} - A_{4,0} + 32) >> 6)$$

$$b_{0,0} = Clip1((-A_{-3,0} + 4*A_{-2,0} - 11*A_{-1,0} + 40*A_{0,0} + 40*A_{1,0} - 11*A_{2,0} + 4*A_{3,0} - A_{4,0} + 32) >> 6)$$

$$c_{0,0} = Clip1((-A_{-3,0} + 3*A_{-2,0} - 7*A_{-1,0} + 19*A_{0,0} + 57*A_{1,0} - 10*A_{2,0} + 4*A_{3,0} - A_{4,0} + 32) >> 6$$

样本位置$d_{0,0}$、$h_{0,0}$及$n_{0,0}$的预测值由垂直方向距离插值点最近的8个整数值滤波得到，预测值获取方式如下：

$$d_{0,0} = Clip1((-A_{0,-3} + 4*A_{0,-2} - 10*A_{0,-1} + 57*A_{0,0} + 19*A_{0,1} - 7*A_{0,2} + 3*A_{0,3} - A_{0,4} + 32) >> 6)$$

$$h_{0,0} = Clip1((-A_{0,-3} + 4*A_{0,-2} - 11*A_{0,-1} + 40*A_{0,0} + 40*A_{0,1} - 11*A_{0,2} + 4*A_{0,3} - A_{0,4} + 32) >> 6)$$

$$n_{0,0} = Clip1((-A_{0,-3} + 3*A_{0,-2} - 7*A_{0,-1} + 19*A_{0,0} + 57*A_{0,1} - 10*A_{0,2} + 4*A_{0,3} - A_{0,4} + 32) >> 6)$$

样本位置 $e_{0,0}$、$i_{0,0}$、$p_{0,0}$、$f_{0,0}$、$j_{0,0}$、$q_{0,0}$、$g_{0,0}$、$k_{0,0}$及 $r_{0,0}$的预测值获取方式如下：

$$e_{0,0} = Clip1((-a'_{0,-3} + 4*a'_{0,-2} - 10*a'_{0,-1} + 57*a'_{0,0} + 19*a'_{0,1} - 7*a'_{0,2} + 3*a'_{0,3} - a'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$i_{0,0} = Clip1((-a'_{0,-3} + 4*a'_{0,-2} - 11*a'_{0,-1} + 40*a'_{0,0} + 40*a'_{0,1} - 11*a'_{0,2} + 4*a'_{0,3} - a'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$p_{0,0} = Clip1((-a'_{0,-3} + 3*a'_{0,-2} - 7*a'_{0,-1} + 19*a'_{0,0} + 57*a'_{0,1} - 10*a'_{0,2} + 4*a'_{0,3} - a'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$f_{0,0} = Clip1((-b'_{0,-3} + 4*b'_{0,-2} - 10*b'_{0,-1} + 57*b'_{0,0} + 19*b'_{0,1} - 7*b'_{0,2} + 3*b'_{0,3} - b'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$j_{0,0} = Clip1((-b'_{0,-3} + 4*b'_{0,-2} - 11*b'_{0,-1} + 40*b'_{0,0} + 40*b'_{0,1} - 11*b'_{0,2} + 4*b'_{0,3} - b'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$q_{0,0} = Clip1((-b'_{0,-3} \quad b'_{0,-2} - 7*b'_{0,-1} + 19*b'_{0,0} + 57*b'_{0,1} - 10*b'_{0,2} + 4*b'_{0,3} - b'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$g_{0,0} = Clip1((-c'_{0,-3} + 4*c'_{0,-2} - 10*c'_{0,-1} + 57*c'_{0,0} + 19*c'_{0,1} - 7*c'_{0,2} + 3*c'_{0,3} \quad c'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$k_{0,0} = Clip1((-c'_{0,-3} + 4*c'_{0,-2} - 11*c'_{0,-1} + 40*c'_{0,0} + 40*c'_{0,1} - 11*c'_{0,2} + 4*c'_{0,3} - c'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$r_{0,0} = Clip1((-c'_{0,-3} \quad c'_{0,-2} - 7*c'_{0,-1} + 19*c'_{0,0} + 57*c'_{0,1} - 10*c'_{0,2} + 4*c'_{0,3} - c'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

其中：

$$a'_{0,i} = (-A_{-3,I} + 4*A_{-2,i} - 10*A_{-1,i} + 57*A_{0,i} + 19*A_{1,i} - 7*A_{2,i} + 3*A_{3,i} - A_{4,i} + ((1 << BitDepth - 8) >> 1)) >> (BitDepth - 8)$$

$b'_{0,i} = (-A_{-3,i} + 4*A_{-2,i} - 11*A_{-1,i} + 40*A_{0,i} + 40*A_{1,i} - 11*A_{2,i} + 4*A_{3,i} - A_{4,i} + ((1 << BitDepth - 8) >> 1)) >> (BitDepth - 8)$

$c'_{0,i} = (-A_{-3,i} + 3*A_{-2,i} - 7*A_{-1,i} + 19*A_{0,i} + 57*A_{1,i} - 10*A_{2,i} + 4*A_{3,i} - A_{4,i} + ((1 << BitDepth - 8) >> 1)) >> (BitDepth - 8)$

9.8.2.5.3 双向光流亮度样本梯度的计算过程

梯度矩阵 gradXL0 的元素 gradXL0[x][y]的值是参考图像队列 0 中参考索引为 RefIndexL0 的参考图像的 1/4 精度亮度样本矩阵中(f_x, f_y)位置的水平梯度值，梯度矩阵 gradYL0 的元素 gradYL0[x][y]的值是其竖直梯度值。梯度矩阵 gradXL1 的元素 gradXL1[x][y]的值是参考图像队列 1 中参考索引为 RefIndexL1 的参考图像的 1/4 精度亮度样本矩阵中(f_x, f_y)位置的水平梯度值，梯度矩阵 gradYL1 的元素 gradYL1[x][y]的值是其竖直梯度值。使用 8 抽头滤波器按以下方法得到。

xFracL 的值为 0，1，2，3 的 gradXL[x][y]（包括 gradXL0[x][y]和 gradXL1[x][y]）中间值为：

$gradXL'[0][y] = (-4*A_{-3,y} + 11*A_{-2,y} - 39*A_{-1,y} - 1*A_{0,y} + 41*A_{1,y} - 14*A_{2,y} + 8*A_{3,y} - 2*A_{4,y} + 2) >> 2$

$gradXL'[1][y] = (-2*A_{-3,y} + 6*A_{-2,y} - 19*A_{-1,y} - 31*A_{0,y} + 53*A_{1,y} - 12*A_{2,y} + 7*A_{3,y} - 2*A_{4,y} + 2) >> 2$

$gradXL'[2][y] = (-A_{-2,y} - 50*A_{0,y} + 50*A_{1,y} + A_{3,y} + 2) >> 2$

$gradXL'[3][y] = (2*A_{-3,y} - 7*A_{-2,y} + 12*A_{-1,y} - 53*A_{0,y} + 31*A_{1,y} + 19*A_{2,y} - 6*A_{3,y} + 2*A_{4,y} + 2) >> 2$

yFracL 的值为 0，1，2，3 的 gradXL[x][y]（包括 gradXL0[x][y]和 gradXL1[x][y]）值为：

$gradXL[x][0] = (64*gradXL'[x][0] + 8192) >> 14$

$gradXL[x][1] = (-1*gradXL'[x][-3] + 4*gradXL'[x][-2] - 10*gradXL'[x][-1] + 57*gradXL'[x][0] + 19*gradXL'[x][1] - 7*gradXL'[x][2] + 3*gradXL'[x][3] - 1*gradXL'[x][4] + 8192) >> 14$

$gradXL[x][2] = (-1*gradXL'[x][-3] + 4*gradXL'[x][-2] - 11*gradXL'[x][-1] + 40*gradXL'[x][0] + 40*gradXL'[x][1] - 11*gradXL'[x][2] + 4*gradXL'[x][3] - 1*gradXL'[x][4] + 8192) >> 14$

$gradXL[x][3] = (-1*gradXL'[x][-3] + 3*gradXL'[x][-2] - 7*gradXL'[x][-1] + 19*gradXL'[x][0] + 57*gradXL'[x][1] - 10*gradXL'[x][2] + 4*gradXL'[x][3] - 1*gradXL'[x][4] + 8192) >> 14$

xFracL 的值为 0，1，2，3 的 gradYL[x][y]（包括 gradYL0[x][y]和 gradYL1[x][y]）中间值为：

$gradYL'[0][y] = (64*A_{0,y} + 2) >> 2$

$gradYL'[1][y] = (-1*A_{-3,y} + 4*A_{-2,y} - 10*A_{-1,y} + 57*A_{0,y} + 19*A_{1,y} - 7*A_{2,y} + 3*A_{3,y} - 1*A_{4,y} + 2) >> 2$

$gradYL'[2][y] = (-1*A_{-3,y} + 4*A_{-2,y} - 11*A_{-1,y} + 40*A_{0,y} + 40*A_{1,y} - 11*A_{2,y} + 4*A_{3,y} - 1*A_{4,y} + 2) >> 2$

$gradYL'[3][y] = (-1*A_{-3,y} + 3*A_{-2,y} - 7*A_{-1,y} + 19*A_{0,y} + 57*A_{1,y} - 10*A_{2,y} + 4*A_{3,y} - 1*A_{4,y} + 2) >> 2$

yFracL 的值为 0，1，2，3 的 gradYL[x][y]（包括 gradYL0[x][y]和 gradYL1[x][y]）值为：

$gradYL[x][0] = (-4*gradYL'[x][-3] + 11*gradYL'[x][-2] - 39*gradYL'[x][-1] - 1*gradYL'[x][0] + 41*gradYL'[x][1] - 14*gradYL'[x][2] + 8*gradYL'[x][3] - 2*gradYL'[x][4] + 8192) >> 14$

$gradYL[x][1] = (-2*gradYL'[x][-3] + 6*gradYL'[x][-2] - 19*gradYL'[x][-1] - 31*gradYL'[x][0] + 53*gradYL'[x][1] - 12*gradYL'[x][2] + 7*gradYL'[x][3] - 2*gradYL'[x][4] + 8192) >> 14$

$gradYL[x][2] = (-1*gradYL'[x][-2] - 50*gradYL'[x][0] + 50*gradYL'[x][1] + gradYL'[x][3] + 8192) >> 14$

$gradYL[x][3] = (2*gradYL'[x][-3] - 7*gradYL'[x][-2] + 12*gradYL'[x][-1] - 53*gradYL'[x][0] + 31*gradYL'[x][1] + 19*gradYL'[x][2] - 6*gradYL'[x][3] + 2*gradYL'[x][4] + 8192) >> 14$

9.8.2.5.4 仿射亮度样本的插值过程

参考图像亮度样本矩阵中的样本位置见图35，A、B、C、D是相邻整像素样本，dx与dy是整像素样本A周边分像素样本a(dx,dy)与A的水平和垂直距离，dx等于fx&15，dy等于fy&15，其中(fx,fy)是该分像素样本在1/16精度的亮度样本矩阵中的坐标。整像素$A_{x,y}$和周边的255个分像素样本$a_{x,y}$(dx,dy)的具体位置见图36。

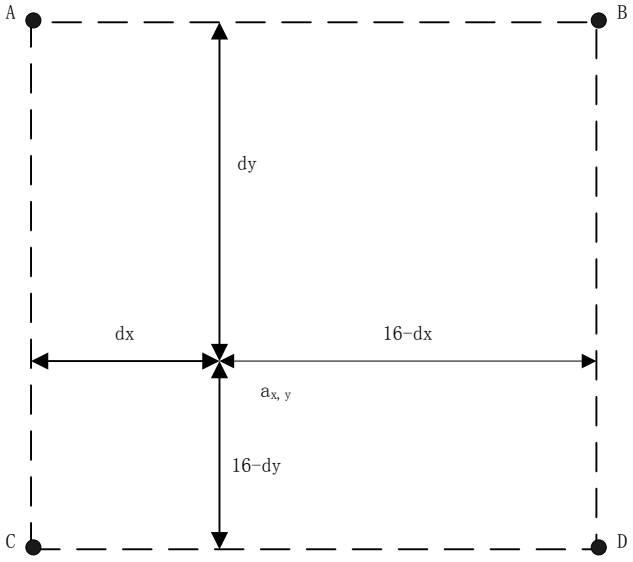


图35 仿射预测亮度样本的位置

$A_{x,y}$	$a_{x,y}(1,0)$	$a_{x,y}(2,0)$	$a_{x,y}(3,0)$	...	$a_{x,y}(15,0)$
$a_{x,y}(0,1)$	$a_{x,y}(1,1)$	$a_{x,y}(2,1)$	$a_{x,y}(3,1)$	...	$a_{x,y}(15,1)$
$a_{x,y}(0,2)$	$a_{x,y}(1,2)$	$a_{x,y}(2,2)$	$a_{x,y}(3,2)$	...	$a_{x,y}(15,2)$
$a_{x,y}(0,3)$	$a_{x,y}(1,3)$	$a_{x,y}(2,3)$	$a_{x,y}(3,3)$	...	$a_{x,y}(15,3)$
...	...	...	...	...	...
$a_{x,y}(0,15)$	$a_{x,y}(1,15)$	$a_{x,y}(2,15)$	$a_{x,y}(3,15)$	...	$a_{x,y}(15,15)$

图36　仿射预测整像素样本和分像素样本的位置

样本位置$a_{x,0}$（x=1～15）由水平方向上距离插值点最近的8个整数值滤波得到，预测值的获取方式如下：

$$a_{x,0} = Clip1((f_L[x][0]*A_{-3,0} + f_L[x][1]*A_{-2,0} + f_L[x][2]*A_{-1,0} + f_L[x][3]*A_{0,0} + f_L[x][4]*A_{1,0} + f_L[x][5]*A_{2,0} + f_L[x][6]*A_{3,0} + f_L[x][7]*A_{4,0} + 32) >> 6)$$

样本位置$a_{0,y}$（y=1～15）由垂直方向上距离插值点最近的8个整数值滤波得到，预测值的获取方式如下：

$$a_{0,y} = Clip1((f_L[y][0]*A_{0,-3} + f_L[y][1]*A_{0,-2} + f_L[y][2]*A_{0,-1} + f_L[y][3]*A_{0,0} + f_L[y][4]*A_{0,1} + f_L[y][5]*A_{0,2} + f_L[y][6]*A_{0,3} + f_L[y][7]*A_{0,4} + 32) >> 6)$$

样本位置$a_{x,y}$（x=1～15，y=1～15）的预测值获取方式如下：

$$a_{x,y} = Clip1((f_L[y][0]*a'_{x,y-3} + f_L[y][1]*a'_{x,y-2} + f_L[y][2]*a'_{x,y-1} + f_L[y][3]*a'_{x,y} + f_L[y][4]*a'_{x,y+1} + f_L[y][5]*a'_{x,y+2} + f_L[y][6]*a'_{x,y+3} + f_L[y][7]*a'_{x,y+4} + (1 << (19 - BitDepth))) >> (20 - BitDepth))$$

其中：

$$a'_{x,y} = (f_L[x][0]*A_{-3,y} + f_L[x][1]*A_{-2,y} + f_L[x][2]*A_{-1,y} + f_L[x][3]*A_{0,y} + f_L[x][4]*A_{1,y} + f_L[x][5]*A_{2,y} + f_L[x][6]*A_{3,y} + f_L[x][7]*A_{4,y} + ((1 << (BitDepth - 8)) >> 1)) >> (BitDepth - 8)$$

仿射预测亮度插值滤波器系数见表115。

表115　仿射预测亮度插值滤波器系数

分像素点位置	亮度插值滤波器系数							
	$f_L[p][0]$	$f_L[p][1]$	$f_L[p][2]$	$f_L[p][3]$	$f_L[p][4]$	$f_L[p][5]$	$f_L[p][6]$	$f_L[p][7]$
1	0	1	−3	63	4	−2	1	0
2	−1	2	−5	62	8	−3	1	0
3	−1	3	−8	60	13	−4	1	0
4	−1	4	−10	58	17	−5	1	0
5	−1	4	−11	52	26	−8	3	−1
6	−1	3	−9	47	31	−10	4	−1
7	−1	4	−11	45	34	−10	4	−1
8	−1	4	−11	40	40	−11	4	−1
9	−1	4	−10	34	45	−11	4	−1
10	−1	4	−10	31	47	−9	3	−1
11	−1	3	−8	26	52	−11	4	−1
12	0	1	−5	17	58	−10	4	−1
13	0	1	−4	13	60	−8	3	−1
14	0	1	−3	8	62	−5	2	−1
15	0	1	−2	4	63	−3	1	0

9.8.2.6　色度样本的插值过程

9.8.2.6.1　概述

如果当前预测单元的AffineFlag的值为0，色度样本的插值过程见9.8.2.6.2；否则，色度样本的插值过程见9.8.2.6.3。

9.8.2.6.2　普通色度样本的插值过程

参考图像色度样本矩阵中的样本位置见图37，A、B、C、D是相邻整像素样本，dx与dy是整像素样本A周边分像素样本a(dx,dy)与A的水平和垂直距离，dx等于fx&7，dy等于fy&7，其中(fx,fy)是该分像素样本在1/8精度的色度样本矩阵中的坐标。整像素$A_{x,y}$和周边的63个分像素样本$a_{x,y}$(dx,dy)的具体位置见图38。

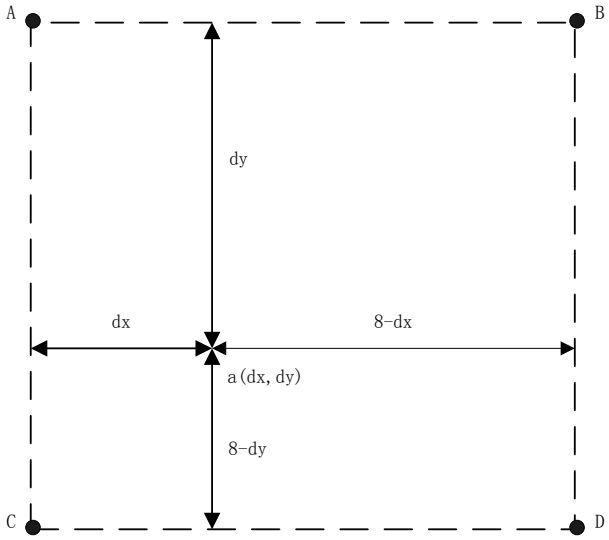


图37　普通预测色度样本位置

$A_{x,y}$	$a_{x,y}(1,0)$	$a_{x,y}(2,0)$	$a_{x,y}(3,0)$	$a_{x,y}(4,0)$	$a_{x,y}(5,0)$	$a_{x,y}(6,0)$	$a_{x,y}(7,0)$
$a_{x,y}(0,1)$	$a_{x,y}(1,1)$	$a_{x,y}(2,1)$	$a_{x,y}(3,1)$	$a_{x,y}(4,1)$	$a_{x,y}(5,1)$	$a_{x,y}(6,1)$	$a_{x,y}(7,1)$
$a_{x,y}(0,2)$	$a_{x,y}(1,2)$	$a_{x,y}(2,2)$	$a_{x,y}(3,2)$	$a_{x,y}(4,2)$	$a_{x,y}(5,2)$	$a_{x,y}(6,2)$	$a_{x,y}(7,2)$
$a_{x,y}(0,3)$	$a_{x,y}(1,3)$	$a_{x,y}(2,3)$	$a_{x,y}(3,3)$	$a_{x,y}(4,3)$	$a_{x,y}(5,3)$	$a_{x,y}(6,3)$	$a_{x,y}(7,3)$
$a_{x,y}(0,4)$	$a_{x,y}(1,4)$	$a_{x,y}(2,4)$	$a_{x,y}(3,4)$	$a_{x,y}(4,4)$	$a_{x,y}(5,4)$	$a_{x,y}(6,4)$	$a_{x,y}(7,4)$
$a_{x,y}(0,5)$	$a_{x,y}(1,5)$	$a_{x,y}(2,5)$	$a_{x,y}(3,5)$	$a_{x,y}(4,5)$	$a_{x,y}(5,5)$	$a_{x,y}(6,5)$	$a_{x,y}(7,5)$
$a_{x,y}(0,6)$	$a_{x,y}(1,6)$	$a_{x,y}(2,6)$	$a_{x,y}(3,6)$	$a_{x,y}(4,6)$	$a_{x,y}(5,6)$	$a_{x,y}(6,6)$	$a_{x,y}(7,6)$
$a_{x,y}(0,7)$	$a_{x,y}(1,7)$	$a_{x,y}(2,7)$	$a_{x,y}(3,7)$	$a_{x,y}(4,7)$	$a_{x,y}(5,7)$	$a_{x,y}(6,7)$	$a_{x,y}(7,7)$

图38　普通预测整像素样本和分像素样本的位置

普通预测色度滤波系数见表116。

表116　普通预测色度滤波系数

数组标识	滤波器系数
C[0]	{ 0, 64, 0, 0 }
C[1]	{ -4, 62, 6, 0 }
C[2]	{ -6, 56, 15, -1 }
C[3]	{ -5, 47, 25, -3 }
C[4]	{ -4, 36, 36, -4 }
C[5]	{ -3, 25, 47, -5 }
C[6]	{ -1, 15, 56, -6 }
C[7]	{ 0, 6, 62, -4 }

对于dx等于0或dy等于0的分像素点，可直接用色度整像素插值得到，对于dx不等于0且dy不等于0的点，使用整像素行（dy等于0）上的分像素进行计算：

```
if (dx == 0) {
    a_{x,y}(0,dy) = Clip3(0, (1<<BitDepth)-1, (C[dy][0]*A_{x,y-1}+C[dy][1]*A_{x,y}+C[dy][2]*A_{x,y+1}+C[dy][3]*A_{x,y+2}+32) >> 6)
}
else if (dy == 0) {
    a_{x,y}(dx,0) = Clip3(0, (1<<BitDepth)-1, (C[dx][0]*A_{x-1,y}+C[dx][1]*A_{x,y}+C[dx][2]*A_{x+1,y}+C[dx][3]*A_{x+2,y}+32) >> 6)
}
else {
    a_{x,y}(dx,dy) = Clip3(0, (1<<BitDepth)-1, (C[dy][0]*a'_{x,y-1}(dx,0) + C[dy][1]*a'_{x,y}(dx,0) + C[dy][2]*a'_{x,y+1}(dx,0)
+ C[dy][3]*a'_{x,y+2}(dx,0) + (1 << (19-BitDepth))) >> (20-BitDepth))
}
```

其中，$a'_{x,y}(dx,0)$是整像素行上的分像素的临时值，定义为：

```
a'_{x,y}(dx,0) = (C[dx][0]*A_{x-1,y} + C[dx][1]*A_{x,y} + C[dx][2]*A_{x+1,y} + C[dx][3]*A_{x+2,y} + ((1 << (BitDepth-8)) >> 1)) >>
(BitDepth - 8)
```

9.8.2.6.3　仿射色度样本的插值过程

参考图像色度样本矩阵中的样本位置见图39，A、B、C、D是相邻整像素样本，dx与dy是整像素样本A周边分像素样本a(dx,dy)与A的水平和垂直距离，dx等于fx&31，dy等于fy&31，其中(fx,fy)是该分像素样本在1/32精度的色度样本矩阵中的坐标。整像素$A_{x,y}$和周边的1023个分像素样本$a_{x,y}(dx,dy)$的具体位置见图40。

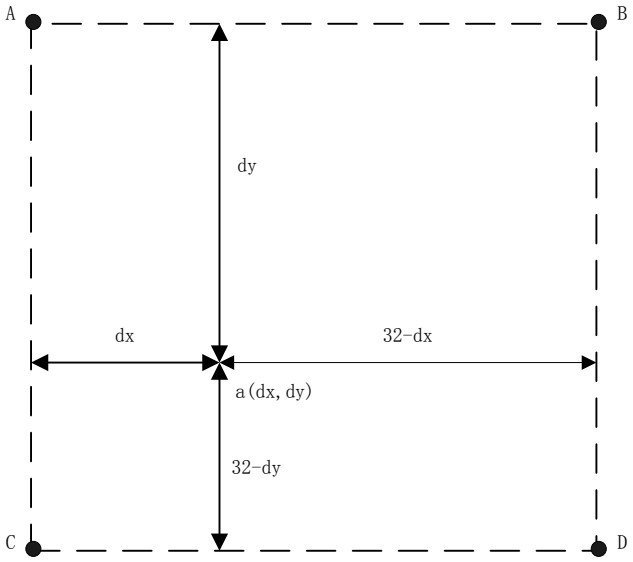


图39　仿射预测色度样本的位置

$A_{x,y}$	$a_{x,y}(1,0)$	$a_{x,y}(2,0)$	$a_{x,y}(3,0)$	...	$a_{x,y}(31,0)$
$a_{x,y}(0,1)$	$a_{x,y}(1,1)$	$a_{x,y}(2,1)$	$a_{x,y}(3,1)$	...	$a_{x,y}(31,1)$
$a_{x,y}(0,2)$	$a_{x,y}(1,2)$	$a_{x,y}(2,2)$	$a_{x,y}(3,2)$	...	$a_{x,y}(31,2)$
$a_{x,y}(0,3)$	$a_{x,y}(1,3)$	$a_{x,y}(2,3)$	$a_{x,y}(3,3)$	...	$a_{x,y}(31,3)$
...	...	...	...	...	...
$a_{x,y}(0,31)$	$a_{x,y}(1,31)$	$a_{x,y}(2,31)$	$a_{x,y}(3,31)$	...	$a_{x,y}(31,31)$

图40　仿射预测整像素样本和分像素样本的位置

对于dx等于0或dy等于0的分像素点，可直接用色度整像素插值得到，对于dx不等于0且dy不等于0的点，使用整像素行（dy等于0）上的分像素进行计算：

```
if (dx == 0) {
    a_{x,y}(0,dy) = Clip3(0, (1<<BitDepth)-1, (f_C[dy][0]*A_{x,y-1}+f_C[dy][1]*A_{x,y}+f_C[dy][2]*A_{x,y+1}+f_C[dy][3]*A_{x,y+2}+32) >> 6)
}
else if (dy == 0) {
    a_{x,y}(dx,0) = Clip3(0, (1<<BitDepth)-1, (f_C[dx][0]*A_{x-1,y}+f_C[dx][1]*A_{x,y}+f_C[dx][2]*A_{x+1,y}+f_C[dx][3]*A_{x+2,y}+32) >> 6)
}
else {
    a_{x,y}(dx,dy) = Clip3(0, (1<<BitDepth)-1, (C[dy][0]*a'_{x,y-1}(dx,0) + C[dy][1]*a'_{x,y}(dx,0) + C[dy][2]*a'_{x,y+1}(dx,0)
+ C[dy][3]*a'_{x,y+2}(dx,0) + (1 << (19-BitDepth))) >> (20-BitDepth))
}
```

其中，$a'_{x,y}(dx,0)$是整像素行上的分像素的临时值，定义为：

$$a'_{x,y}(dx,0) = (f_C[dx][0]*A_{x-1,y} + f_C[dx][1]*A_{x,y} + f_C[dx][2]*A_{x+1,y} + f_C[dx][3]*A_{x+2,y} + ((1 << (BitDepth-8)) >> 1)) >> (BitDepth-8)$$

仿射预测色度插值滤波器系数见表117。

表117　仿射预测色度插值滤波器系数

分像素位置	插值滤波器系数			
	$f_C[p][0]$	$f_C[p][1]$	$f_C[p][2]$	$f_C[p][3]$
1	-1	63	2	0
2	-2	62	4	0
3	-2	60	7	-1
4	-2	58	10	-2
5	-3	57	12	-2
6	-4	56	14	-2
7	-4	55	15	-2
8	-4	54	16	-2
9	-5	53	18	-2
10	-6	52	20	-2
11	-6	49	24	-3
12	-6	46	28	-4
13	-5	44	29	-4
14	-4	42	30	-4
15	-4	39	33	-4
16	-4	36	36	-4
17	-4	33	39	-4
18	-4	30	42	-4
19	-4	29	44	-5

表 117（续）

分像素位置	插值滤波器系数			
	$f_C[p][0]$	$f_C[p][1]$	$f_C[p][2]$	$f_C[p][3]$
20	-4	28	46	-6
21	-3	24	49	-6
22	-2	20	52	-6
23	-2	18	53	-5
24	-2	16	54	-4
25	-2	15	55	-4
26	-2	14	56	-4
27	-2	12	57	-3
28	-2	10	58	-2
29	-1	7	60	-2
30	0	4	62	-2
31	0	2	63	-1

9.8.3 预测样本合成

9.8.3.1 概述

输入：运动信息motionInfo(mvE0,mvE1,refIndexL0,refIndexL1,interPredRefMode,interPredRefMode)，当前编码单元的宽度W和高度H，预测样本矩阵predMatrixL0和predMatrixL1。

输出：合成后的预测样本矩阵PredMatrixComb。导出PredMatrixComb的方法如下。

第1种情况，当前编码单元只包含色度编码块，由运动信息中的预测参考模式interPredRefMode、W和H按9.8.3.3得到合成后的预测样本矩阵PredMatrixComb。

第2种情况，当前编码单元的编码单元子类型是‘P_Skip_Affine’‘P_Direct_ Affine’‘P_Inter_ Affine’‘B_Skip_Affine’‘B_Direct_ Affine’或‘B_Inter_ Affine’，由interPredRefMode、predMatrixL0、predMatrixL1、W和H按9.8.3.3得到PredMatrixComb。

第3种情况，当前编码单元的编码单元子类型是‘P_Skip_Mvap’‘P_Direct_Mvap’‘B_Skip_Mvap’‘B_Direct_Mvap’‘P_Skip_Etmvp’‘P_Direct_Etmvp’‘B_Skip_Etmvp’或‘B_Direct_Etmvp’，按以下方法得到PredMatrixComb。

a) 将当前预测单元划分为宽度 subW 和高度 subH 均等于 8 的子块。令 i 和 j 分别是子块在当前预测单元内的水平索引值和垂直索引值，i 的取值范围为 0～(W>>3)-1，j 的取值范围为 0～(H>>3)-1，MotionArray[i][j]是当前子块的运动信息，子块左上角坐标(x,y)等于(xE+8*i,yE+8*j)。

b) 如果当前子块的 interPredRefMode 等于‘PRED_List01’：

1) 如果 BioEnableFlag 的值等于 1，且当前子块的参考图像队列 0 中参考帧索引值为 refIndexL0 的参考帧和参考图像队列 1 中参考索引值为 refIndexL1 的参考帧显示顺序分别位于当前图像的两侧，且当前预测块的 AmvrIndex 的值等于 0，且当前预测块的 InterPcFlag 的值为 0，则 BioFlag 的值为 1，根据 predMatrixL0、predMatrixL1、gradXL0、gradXL1、gradYL0、gradYL1、subW 和 subH 按 9.8.3.2 得到 predMatrixCombSub；

2) 否则，BioFlag 的值为 0，并由 interPredRefMode、predMatrixL0、predMatrixL1、subW 和 subH 按 9.8.3.3 得到 predMatrixCombSub。

c) 否则，如果当前子块的 interPredRefMode 等于‘PRED_List0’或‘PRED_List1’，由 interPredRefMode、predMatrixL0、predMatrixL1、subW 和 subH 按 9.8.3.3 得到 predMatrixSingleSub：

1) 如果 ObmcEnableFlag 等于 1，且当前图像类型不为 P 图像，且当前编码单元包含不小于 64 个亮度样本，且 i 或 j 等于 0，则令 wFlag 等于(W>=16?1:0)，hFlag 等于(H>=16?1:0)，由当前子块左上角位置(xE+8*i,yE+8*j)，当前子块宽度 subW 和高度 subH，当前子块水平索引 i 和竖直索引 j，标志位 wFlag 和 hFlag，以及当前子块 predMatrixSingleSub 按 9.8.3.5 得到 predMatrixCombSub；

2) 否则，令 predMatrxiCombSub 等于 predMatrixSignleSub。

d) PredMatrixComb 由各个子块的 predMatrixCombSub 组成。

第4种情况，当前编码单元的编码单元子类型是‘P_Skip_SbTemporal’‘P_Direct_SbTemporal’‘B_Skip_SbTemporal’或‘B_Direct_SbTemporal’，则将当前预测单元划分为4个子块（见图27，图中的数字是子块的索引值），按以下方法得到PredMatrixComb。

a) 令 MotionArray[i]是索引值为 i 的子块的运动信息，子块的宽度 subW 和高度 subH 分别等于 W/2 和 H/2，子块左上角坐标(x,y)等于(xE+(W/2*(i%2)),yE+((H/2)*(i/2)))。

b) 如果当前子块的 interPredRefMode 等于‘PRED_List01’，且 BioEnableFlag 的值等于 1，且当前子块的参考图像队列 0 中参考帧索引值为 refIndexL0 的参考帧和参考图像队列 1 中参考索引值为 refIndexL1 的参考帧显示顺序分别位于当前图像的两侧，且当前预测块的 AmvrIndex 的值等于 0，且当前预测块的 InterPcFlag 的值为 0，则 BioFlag 的值为 1；否则，BioFlag 的值为 0。

c) 如果 BioFlag 的值等于 1，由 predMatrixL0、predMatrixL1、gradXL0、gradXL1、gradYL0、gradYL1、subW 和 subH 按 9.8.3.2 得到 predMatrixComb。

d) 否则，由 interPredRefMode、predMatrixL0、predMatrixL1、subW 和 subH 按 9.8.3.3 得到 predMatrixCombSub。

e) PredMatrixComb 由各个子块的 predMatrixCombSub 组成。

第5种情况，当前编码单元的编码单元子类型是‘B_Skip_Awp’‘B_Direct_Awp’‘P_Skip_Awp’或‘P_Direct_Awp’，按9.8.3.4得到predMatrixAwp，作为PredMatrixComb。

如果以上5种情况都不满足，按以下方法得到PredMatrixComb。

a) 如果当前预测单元的 interPredRefMode 等于‘PRED_List01’：

1) 如果 BioEnableFlag 的值等于 1，且当前预测单元的参考图像队列 0 中参考帧索引值为 refIndexL0 的参考帧和参考图像队列 1 中参考索引值为 refIndexL1 的参考帧显示顺序分别位于当前图像的两侧，且当前预测块的 AmvrIndex 的值等于 0，且当前预测块的 InterPcFlag 的值为 0，则 BioFlag 的值为 1，由 predMatrixL0、predMatrixL1、gradXL0、gradXL1、gradYL0、gradYL1、W 和 H 按 9.8.3.2 得到 predMatrixCombSub；

2) 否则，BioFlag 的值为 0，由 interPredRefMode、predMatrixL0、predMatrixL1、W 和 H 按 9.8.3.3 得到 predMatrixCombSub。

b) 否则，如果当前预测单元的 InterPredRefMode 等于‘PRED_List0’或‘PRED_List1’，由 interPredRefMode、predMatrixL0、predMatrixL1、W 和 H 按 9.8.3.3 得到 predMatrixSingleSub：

1) 如果 ObmcEnableFlag 等于 1，且当前图像类型不为 P 图像，且当前编码单元包含不小于 64 个亮度样本，则令 wFlag 等于(W>=16?1:0)，hFlag 等于(H>=16?1:0)，由当前编码单元左上角位置(xE,yE)，编码单元宽度 W 和高度 H，水平方向索引 0 和垂直方向索引 0，标志

位 wFlag 和 hFlag，以及当前预测单元 predMatrixSingleSub 按 9.8.3.5 得到 predMatrixCombSub；

2) 否则，令 predMatrxiCombSub 等于 predMatrixSignleSub。

c) PredMatrixComb 等于 predMatrixCombSub。

9.8.3.2 导出双向光流预测样本改善值

输入：predMatrixL0、predMatrixL1、gradXL0、gradXL1、gradYL0和gradYL1，以及高度W和宽度H。

输出：合成后的预测样本矩阵predMatrixComb。

第1步，初始化W×H矩阵s1、s2、s3、s4和s5。

```
for (x=0; x<W; x++) {
    for (y=0; y<H; y++) {
        t = predMatrixL0[x][y] - predMatrixL1[x][y]
        tx = gradXL0[x][y] + gradXL1[x][y]
        ty = gradYL0[x][y] + gradYL1[x][y]
        s1[x][y] = tx * tx
        s2[x][y] = tx * ty
        s3[x][y] = -tx * t
        s4[x][y] = ty * ty
        s5[x][y] = -ty * t
    }
}
```

第2步，令(tx,ty)是4×4子块的左上角坐标，(i,j)是4×4子块中样本点位置坐标，tx的取值范围为0～(W>>2)，ty的取值范围为0～(H>>2)。按以下方法依次处理当前块的各个4×4子块，得到子块的predMatrixComb。

a) 计算 t1、t2、t3、t4 和 t5。

```
t1 = 0
t2 = 0
t3 = 0
t4 = 0
t5 = 0
for (i=tx*4; i<tx*4+4; i++) {
    for (j=ty*4; j<ty*4+4; j++) {
        t1 += s1[i][j]
        t2 += s2[i][j]
        t3 += s3[i][j]
        t4 += s4[i][j]
        t5 += s5[i][j]
    }
}
```

b) 令 LIMITBIO 等于 768，DENOMBIO 等于 2，计算 vx 和 vy。

```
if (t1 > DENOMBIO) {
    vx = Clip3(-LIMITBIO, LIMITBIO, (t3 << 5) >> Floor(Log(t1 + DENOMBIO))
}
else {
    vx = 0
}
if (t4 > DENOMBIO) {
    vy = Clip3(-LIMITBIO, LIMITBIO, ((t5 << 6) - vx * t2) >> Floor(Log((t4 + DENOMBIO ) << 1))
}
else {
    vy = 0
}
```

c) 计算子块样本位置的预测样本改善值b。

```
for (i=tx*4; i<tx*4+4; i++) {
    for (j=ty*4; j<ty*4+4; j++) {
        b[i][j] = vx * (gradXL0[i][j] - gradXL1[i][j]) + vy * (gradYL0[i][j] - gradYL1[i][j])
        b[i][j] = (b[i][j] > 0) ? ((b[i][j] + 32) >> 6) : (-((-b[i][j] + 32) >> 6))
    }
}
```

d) 得到predMatrixComb。

```
for (i=tx*4; i<tx*4+4; i++) {
    for (j=ty*4; j<ty*4+4; j++) {
        predMatrixComb[i][j] = Clip1((predMatrixL0[i][j] + predMatrixL1[i][j] + b[i][j]+1) >> 1)
    }
}
```

第3步，当前块的predMatrixComb由4×4子块的predMatrixComb组成。

9.8.3.3 合成普通预测样本

输入：预测参考模式interPredRefMode，predMatrixL0和predMatrixL1，以及宽度W和高度H。

输出：合成后的预测样本矩阵predMatrixComb。

如果interPredRefMode等于‘PRED_List0’：

```
for (x=0; x<W; x++) {
    for (y=0; y<H; y++) {
        predMatrixComb[x][y] = predMatrixL0[x][y]
    }
}
```

如果interPredRefMode等于‘PRED_List1’：

```
for (x=0; x<W; x++) {
```

```
    for (y=0; y<H; y++) {
        predMatrixComb[x][y] = predMatrixL1[x][y]
    }
}
```

如果interPredRefMode等于‘PRED_List01’：

```
for (x=0; x<W; x++) {
    for (y=0; y<H; y++) {
        predMatrixComb[x][y] = (predMatrixL0[x][y] + predMatrixL1[x][y] + 1) >> 1
    }
}
```

9.8.3.4 角度加权预测样本导出过程

9.8.3.4.1 概述

输入：角度加权预测模式索引值AwpIndex、PredMatrixAwp0、PredMatrixAwp1，以及宽度W和高度H。

输出：角度加权预测样本矩阵predMatrixAwp。

先按9.8.3.4.2得到角度加权预测权重矩阵，再按9.8.3.4.3得到角度加权预测样本矩阵。

9.8.3.4.2 导出角度加权预测权重矩阵

输入：角度加权预测模式索引值AwpIndex，以及宽度W和高度H。

输出：角度加权预测的亮度权重矩阵AwpWeightMatrixY和色度权重矩阵AwpWeightMatrixUV。

第1步，计算变量stepIndex、angleIndex和angleAreaIndex。

```
stepIndex = (AwpIndex >> 3) - 3
modAngNum = AwpIndex % 8
if (modAngNum == 2) {
    angleIndex = 7
}
else if (modAngNum == 6) {
    angleIndex = 8
}
else {
    angleIndex = modAngNum % 2
}
angleAreaIndex = modAngNum >> 1
```

第2步，如果当前图像是P图像，令PictureAwpRefineIndex等于1。先根据angleAreaIndex查表118得到vL，再根据angleAreaIndex和PictureAwpRefineIndex查表119得到shift和fP，如果当前图像是P图像，令shift等于3。最后得到参考权重列表ReferenceWeights。

```
for (x=0; x<vL; x++) {
    ReferenceWeights[x] = Clip3(0, 8, (x - fP) << shift)
}
```

表118 angleArealndex 和 vL 的对应关系

angleAreaIndex	vL
0	(H + (W >> angleIndex)) << 1
1	
2	(W + (H >> angleIndex)) << 1
3	

表119 angleArealndex，PictureAwpRefineIndex 和 shift，fP 的对应关系

angleAreaIndex	PictureAwpRefineIndex	shift	fP
0	0	0	(vL >> 1) - 6 + stepIndex * ((vL >> 3) - 1)
	1	2	(vL >> 1) - 3 + stepIndex * ((vL >> 3) - 1)
1	0	0	(vL >> 1) - 4 + stepIndex * ((vL >> 3) -1) – ((W << 1) >> angleIndex)
	1	2	(vL >> 1) - 1 + stepIndex * ((vL >> 3) -1) – ((W << 1) >> angleIndex)
2	0	0	(vL >> 1) - 4 + stepIndex * ((vL >> 3) -1) – ((H << 1) >> angleIndex)
	1	2	(vL >> 1) - 1 + stepIndex * ((vL >> 3) -1) – ((H << 1) >> angleIndex)
3	0	0	(vL >> 1) - 6 + stepIndex * ((vL >> 3) - 1)
	1	2	(vL >> 1) - 3 + stepIndex * ((vL >> 3) - 1)

第3步，先根据当前图像类型和angleAreaIndex查表120得到tP，再得到亮度权重矩阵AwpWeightMatrixY。

```
for (x=0; x<W; x++) {
  for (y=0; y<H; y++) {
    AwpWeightMatrixY[x][y] = ReferenceWeights[tP]
  }
}
```

表120 angleArealndex 和 tP 的对应关系

图像类型	angleAreaIndex	tP
B 图像	0	(y << 1) + ((x << 1) >> angleIndex)
	1	(y << 1) - ((x << 1) >> angleIndex)
	2	(x << 1) - ((y << 1) >> angleIndex)
	3	(x << 1) + ((y << 1) >> angleIndex)
P 图像	0	((y >> 2) << 3) + 4 + ((((x >> 2) << 3) + 4) >> angleIndex)
	1	((y >> 2) << 3) + 4 – ((((x >> 2) << 3) + 4) >> angleIndex)
	2	((x >> 2) << 3) + 4 – ((((y >> 2) << 3) + 4) >> angleIndex)
	3	((x >> 2) << 3) + 4 + ((((y >> 2) << 3) + 4) >> angleIndex)

第4步，得到色度权重矩阵AwpWeightMatrixUV。

a) 如果当前图像是 P 图像，则：

```
for (x=0; x<W/2; x++) {
    for (y=0; y<H/2; y++) {
        AwpWeightMatrixUV[x][y] = AwpWeightMatrixY[(x>>2)<<3][(y>>2)<<3]
    }
}
```

b) 否则：

```
for (x=0; x<W/2; y++) {
    for (y=0; y<H/2; x++) {
        AwpWeightMatrixUV[x][y] = AwpWeightMatrixY[x<<1][y<<1]
    }
}
```

9.8.3.4.3 导出角度加权预测样本

输入：角度加权预测亮度权重矩阵AwpWeightMatrixY和色度权重矩阵AwpWeightMatrixUV，当前预测块的宽度W和高度H。

输出：加权预测样本矩阵predMatrixAwp。

如果当前预测块是亮度预测块：

```
for (x=0; x<W; x++)
    for (y=0; y<H; y++) {
        predMatrixAwp[x][y] = (PredMatrixAwp0[x][y] * AwpWeightMatrixY[x][y] + PredMatrixAwp1[x][y] * (8
- AwpWeightMatrixY[x][y] ) + 4) >> 3
    }
}
```

如果当前预测块是色度预测块：

```
for (x=0; x<W/2; x++)
    for (y=0; y<H/2; y++) {
        predMatrixAwp[x][y] = (PredMatrixAwp0[x][y] * AwpWeightMatrixUV[x][y] + PredMatrixAwp1[x][y] * (8
- AwpWeightMatrixUV[x][y] ) + 4) >> 3
    }
}
```

9.8.3.5 合成重叠块预测样本

9.8.3.5.1 概述

输入：当前编码单元或子块左上角样本的位置(xSb,ySb)，宽度subW和高度subH，水平索引i和竖直索引j，标志位wFlag和hFlag，预测样本矩阵PredMatrixSingleSub，当前编码单元上侧相邻块单向运动信息矩阵MotionInfoTop和左侧相邻块单向运动信息矩阵MotionInfoLeft。

输出：调整后的当前子块的预测样本矩阵 PredMatrixCombSub。

9.8.3.5.2 重叠块运动补偿过程

输入：当前编码单元左上角样本的位置(xCb,yCb)，当前子块左上角样本位置（xSb，ySb），当前编码单元的宽度W和高度H，当前子块宽度subW和高度subH，当前子块水平索引i和竖直索引j，当前子块的预测样本矩阵PredMatrixSingleSub，当前编码单元上侧相邻块单向运动信息矩阵MotionInfoTop和左侧相邻块单向运动信息矩阵MotionInfoLeft。

输出：调整后的当前子块的预测样本矩阵PredMatrixCombSub。

第1步，更新当前编码单元上边缘像素。

a) 如果hFlag等于1，令色度加权标识chromaBlending为1；否则，令chromaBlending为0。将当前子块划分为宽度subWidth为4，高度subHeight为(subH>>1)的加权块。令水平索引xSbIndex从0～(subW>>2)-1，竖直索引ySbIndex从0～1，设当前加权块位置(xCurr,yCurr)为(xSb+(xSbIndex<<2),ySb+ySbIndex*subHeight)，其相邻运动信息topMotionInfo为MotionInfoTop[(subW>>2)*i+xSbIndex]中的运动信息，加权方向标识blendingDir为0，令xSbIndex从0～(subW>>2)-1，循环执行步骤1.2。

b) 如果j等于0，对于竖直索引ySbIndex等于0的加权块依次执行以下操作，导出加权块预测样本矩阵PredMatrixobmcSub；否则（j不等于0，或竖直索引ySbIndex等于1的加权块），令预测样本矩阵PredMatrixobmcSub等于PredMatrixSingleSub中与当前加权块对应的部分。

   1) 如果 topMotionInfo 中的预测参考模式为‘Pred_List0’或‘Pred_List1’，由运动信息 topMotionInfo、(xCurr,yCurr)、subWidth 和 subHeight 按 9.8.2.2 得到预测样本 predAboveOBMCL0 和 predAboveOBMCL1。

   2) 如果 topMotionInfo 中的预测参考模式为‘Pred_List0’，由 subWidth、subHeight、predAboveOBMCL0、blendingDir、chromaBlending 和预测样本矩阵 PredMatrixSingleSub 中与当前加权块对应的部分按 9.8.3.5.3 导出预测样本矩阵 PredMatrixCombSub。

   3) 如果 topMotionInfo 中的预测参考模式为‘Pred_List1’，由 subWidth、subHeight、predAboveOBMCL1、blendingDir、chromaBlending 和预测样本矩阵 PredMatrixSingleSub 中与当前加权块对应的部分按 9.8.3.5.3 导出预测样本矩阵 PredMatrixCombSub。

c) 当前块的预测样本矩阵 PredMatrixObmcSub 由其中各个加权块的预测样本矩阵 PredMatrixobmcSub组成。

第2步，更新当前编码单元左边缘像素。

a) 如果wFlag等于1，令色度加权标识chromaBlending为1；否则，令chromaBlending置为0。将当前子块划分为宽度subSWidth为(subW>>1)，高度subHeight为4的加权块。令加权块水平索引xSbIndex从0～1，ySbIndex从0～(subH>>2)-1，设当前子块位置(xCurr,yCurr)为(xCb+subH/2,yCb+(ySbIndex<<2))，运动信息leftMotoinInfo为MotionInfoLeft[(subH>>2]*j+ySbIndex]中的运动信息，加权方向标识blendingDir为1，令ySbIndex从0～(subH>>2)-1，循环执行步骤2.2。

b) 如果i等于0，对于水平索引xSbIndex等于0的加权块依次执行以下操作，导出加权块预测样本矩阵PredMatrixobmcSub；否则（i不等于0，或水平索引xSbIndex等于1的加权块），令预测样本矩阵PredMatrixCombSub等于PredMatrixObmcSub中与当加权块对应的部分。

   1) 如果 leftMotoinInfo 中的预测参考模式为‘Pred_List0’或‘Pred_List1’，由运动信息 leftMotionInfo、(xCurr,yCurr)、subWidth 和 subHeight 按 9.8.2.2 得到预测样本 predLeftOBMCL0 和 predLeftOBMCL1。

2) 如果 leftMotionInfo 中的预测参考模式为‘Pred_List0’，由 subWidth、subHeight、predLeftOBMCL0、blendingDir、chromaBlending 和预测样本矩阵 PredMatrixObmcSub 中与当前加权块对应的部分按 9.8.3.5.3 导出预测样本矩阵 PredMatrixCombSub。

3) 如果 leftMotionInfo 中的预测参考模式为‘Pred_List1’，由 subWidth、subHeight、predLeftOBMCL1、blendingDir、chromaBlending 和预测样本矩阵 PredMatrixObmcSub 中与当前加权块对应的部分按 9.8.3.5.3 导出预测样本矩阵 PredMatrixCombSub。

c) 步骤2.3，当前块的预测样本矩阵PredMatrixComSub由其中各个加权块的预测样本矩阵PredMatrixcomSub组成。

9.8.3.5.3 重叠块运动补偿值加权过程

输入：加权块宽度subWidth和高度subHeight，当前加权块加权预测样本矩阵predObmc，加权方向标识blendingDir和色度加权标识chromaBlending,以及当前加权块合成前预测样本矩阵predMatrixIn。

输出：合成后预测样本矩阵 PredMatrixOut。

如果CodingTreeComponent不等于‘COMPONENT_CHROMA’，predMatrixInLuma是predMatrixIn中的亮度预测样本矩阵，predObmcLuma是predObmc中的亮度预测样本矩阵，PredMatrixOut是PredMatrixOut中的亮度预测样本矩阵。

如果CodingTreeComponent不等于‘COMPONENT_LUMA’，predMatrixInCb和predMatrixInCr是predMatrixIn中的两个色度样本矩阵，predObmcCb和predObmcCr是predObmc中的两个色度样本矩阵，PredMatrixOutCb和PredMatrixOutCr是PredMatrixOut中的两个色度样本矩阵。

令wL和wC分别是亮度和色度样本使用的加权系数向量。

如果blendingDir等于0，令curSize等于subHeight；否则（blendingDir等于1），令curSize等于subWidth。按以下步骤更新当前子块预测样本。

a) 如果PictureObmcBlendingSetIndex等于0，设wL和wC分别为表121中W[curSize]和W[curSize>>1]向量中加权系数；否则，设wL和wC分别为表122中W[curSize]和W[curSize>>1]向量中加权系数。

b) 如果CodingTreeComponent不等于‘COMPONENT_CHROMA’，按以下方法对当前子块亮度预测样本进行加权调整。

```
for (x=0; x<subWidth; x++) {
    for (y=0; y<subHeight; y++) {
        PredMatrixOutLuma[x][ y] = Clip1((wL[y] * PredMatrixInLuma[x][y] + (64 - wL[y]) * predObmcLuma[x][y] + 32 ) >> 6)
    }
}
```

c) 如果CodingTreeComponent不等于‘COMPONENT_LUMA’且chromaBlending等于1，按以下方法对当前子块色度预测样本进行加权调整。

```
for (x=0; x<subWidth>>1; x++) {
    for (y=0; y<subHeight>>1; y++) {
        PredMatrixOutCb[x][ y] = Clip1((wC[y] * PredMatrixInCb[x][y] + (64 - wC[y]) * predObmcCb[x][y] + 32 ) >> 6)
        PredMatrixOutCr[(xSb>>1)+x][ (ySb>>1)+y] = Clip1((wC[y] * PredMatrixInCr[x][ y] + (64 - wC[y]) * predObmcCr[x][y] + 32 ) >> 6)
```

```
    }
}
```

表121　重叠块运动补偿加权系数（一）

Index	W[64]	W[32]	W[16]	W[8]	W[4]	W[2]
0	56	56	56	57	58	59
1	56	57	57	58	60	63
2	56	57	58	60	63	—
3	57	57	59	61	64	—
4	57	58	59	62	—	—
5	57	58	60	63	—	—
6	57	59	61	64	—	—
7	57	59	61	64	—	—
8	58	59	62	—	—	—
9	58	60	62	—	—	—
10	58	60	63	—	—	—
11	58	60	63	—	—	—
12	58	61	64	—	—	—
13	59	61	64	—	—	—
14	59	61	64	—	—	—
15	59	62	64	—	—	—
16	59	62	—	—	—	—
17	59	62	—	—	—	—
18	60	62	—	—	—	—
19	60	63	—	—	—	—
20	60	63	—	—	—	—
21	60	63	—	—	—	—
22	60	63	—	—	—	—
23	60	63	—	—	—	—
24	61	63	—	—	—	—
25	61	64	—	—	—	—
26	61	64	—	—	—	—
27	61	64	—	—	—	—
28	61	64	—	—	—	—
29	61	64	—	—	—	—
30	61	64	—	—	—	—
31	62	64	—	—	—	—
32	62	—	—	—	—	—
33	62	—	—	—	—	—
34	62	—	—	—	—	—
35	62	—	—	—	—	—

表 121（续）

Index	W[64]	W[32]	W[16]	W[8]	W[4]	W[2]
36	62	—	—	—	—	—
37	62	—	—	—	—	—
38	62	—	—	—	—	—
39	63	—	—	—	—	—
40	63	—	—	—	—	—
41	63	—	—	—	—	—
42	63	—	—	—	—	—
43	63	—	—	—	—	—
44	63	—	—	—	—	—
45	63	—	—	—	—	—
46	63	—	—	—	—	—
47	63	—	—	—	—	—
48	63	—	—	—	—	—
49	63	—	—	—	—	—
50	64	—	—	—	—	—
51	64	—	—	—	—	—
52	64	—	—	—	—	—
53	64	—	—	—	—	—
54	64	—	—	—	—	—
55	64	—	—	—	—	—
56	64	—	—	—	—	—
57	64	—	—	—	—	—
58	64	—	—	—	—	—
59	64	—	—	—	—	—
60	64	—	—	—	—	—
61	64	—	—	—	—	—
62	64	—	—	—	—	—
63	64	—	—	—	—	—

表122　重叠块运动补偿加权系数（二）

Index	W[64]	W[32]	W[16]	W[8]	W[4]	W[2]
0	40	41	41	42	45	49
1	41	42	44	47	53	62
2	41	43	46	51	60	—
3	42	44	48	55	64	—
4	43	45	50	59	—	—
5	43	46	52	61	—	—
6	44	48	54	63	—	—

表 122（续）

Index	W[64]	W[32]	W[16]	W[8]	W[4]	W[2]
7	44	49	56	64	—	—
8	45	50	58	—	—	—
9	46	51	59	—	—	—
10	46	52	61	—	—	—
11	47	53	62	—	—	—
12	47	54	63	—	—	—
13	48	55	63	—	—	—
14	48	56	64	—	—	—
15	49	57	64	—	—	—
16	49	57	—	—	—	—
17	50	58	—	—	—	—
18	51	59	—	—	—	—
19	51	60	—	—	—	—
20	52	60	—	—	—	—
21	52	61	—	—	—	—
22	53	61	—	—	—	—
23	53	62	—	—	—	—
24	54	62	—	—	—	—
25	54	63	—	—	—	—
26	55	63	—	—	—	—
27	55	63	—	—	—	—
28	55	64	—	—	—	—
29	56	64	—	—	—	—
30	56	64	—	—	—	—
31	57	64	—	—	—	—
32	57	—	—	—	—	—
33	58	—	—	—	—	—
34	58	—	—	—	—	—
35	58	—	—	—	—	—
36	59	—	—	—	—	—
37	59	—	—	—	—	—
38	59	—	—	—	—	—
39	60	—	—	—	—	—
40	60	—	—	—	—	—
41	60	—	—	—	—	—
42	61	—	—	—	—	—
43	61	—	—	—	—	—
44	61	—	—	—	—	—
45	62	—	—	—	—	—

表 122（续）

Index	W[64]	W[32]	W[16]	W[8]	W[4]	W[2]
46	62	—	—	—	—	—
47	62	—	—	—	—	—
48	62	—	—	—	—	—
49	62	—	—	—	—	—
50	63	—	—	—	—	—
51	63	—	—	—	—	—
52	63	—	—	—	—	—
53	63	—	—	—	—	—
54	63	—	—	—	—	—
55	63	—	—	—	—	—
56	64	—	—	—	—	—
57	64	—	—	—	—	—
58	64	—	—	—	—	—
59	64	—	—	—	—	—
60	64	—	—	—	—	—
61	64	—	—	—	—	—
62	64	—	—	—	—	—
63	64	—	—	—	—	—

9.8.4 预测样本修正

9.8.4.1 概述

输入：当前编码单元的宽度W和高度H，当前编码单元左上角样本的坐标(xE,yE)，预测样本矩阵predMatrixL0、predMatrixL1和合成后的预测样本矩阵predMatrixComb。

输出：帧间预测的预测样本矩阵predMatrixInter。

如果BgcFlag的值等于1，由predMatrixL0、predMatrixL1、predMatrixComb和BgcIndex按9.8.4.2得到predMatrixBgc；否则，predMatrixBgc等于predMatrixComb。

如果 InterPfFlag 的值等于 1，由 predMatrixBgc 和 InterPfIndex 按 9.8.4.3 得到 predMatrixInterPf；否则，predMatrixInterPf 等于 predMatrixBgc。

如果 InterPcFlag 的值等于 1，由 predMatrixInterPf 和 InterPcIndex 按 9.8.4.4 得到 predMatrixInter；否则，predMatrixInter 等于 predMatrixInterPf。

9.8.4.2 导出双向梯度修正后的预测样本

输入：预测样本矩阵predMatrixL0、predMatrixL1和predMatrixComb，以及宽度M和高度N。

输出：修正后的预测样本矩阵predMatrixBgc。

第1步，导出修正样本改善值矩阵g。

a) 如果BgcIndex值为1：

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
```

```
        g[x][y] = (predMatrixL0[x][y] - predMatrixL1[x][y]) >> 3
    }
}
```

b) 如果BgcIndex值为0：

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        g[x][y] = (predMatrixL1[x][y] - predMatrixL0[x][y]) >> 3
    }
}
```

第2步，导出修正后的预测样本矩阵predMatrixBgc。

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        predMatrixBgc[x][y] = Clip1(PredMatrixComb[x][y] + g[x][y])
    }
}
```

9.8.4.3 导出帧间预测滤波后的预测样本

输入：修正后的预测样本矩阵predMatrixBgc，以及亮度块的宽度M和高度N，色度块的宽度K和高度L。

输出：帧间预测的预测样本矩阵predMatrixInterPf。

第1步，分别按9.7.1.2和9.7.1.3得到当前块的亮度和色度参考样本。

a) 对于亮度块,当前块上边的参考样本记为r[i],左边的参考样本记为c[j],其中r[0]等于c[0]。对于色度块，当前块上边的参考样本记为row[i]，左边的参考样本记为col[j]，其中row[0]等于col[0]。

b) 如果当前块是亮度块：

```
for (i=0; i<2M; i++) {
    topPel[i] = r[i]
}
for (j=0; i<2N; i++) {
    leftPel[j] = c[j]
}
```

c) 如果当前块是色度块：

```
for (i=0; i<2K; i++) {
    topPel[i] = row[i]
}
for (j=0; i<2L; i++) {
    leftPel[j] = col[j]
}
```

第2步，如果InterPfIndex为0，按以下方法得到预测样本矩阵predMatrixTmp。其中，如果当前预测块是亮度块，x的取值范围为0～M-1，y的取值范围为0～N-1；如果当前预测块是色度块，x的取值范围为0～K-1，y的取值范围为0～L-1。

a) 得到预测样本矩阵predPlane。

```
predV = ((N - 1 - y) * topPel[x+1] + (y + 1) * leftPel[N+1] + (N >> 1)) >> Log(N)
predH = ((M - 1 - x) * leftPel[y+1] + (x + 1) * topPel[M+1] + (M >> 1)) >> Log(M)
predPlane[x][y] = (predV + predH + 1) >> 1
```

b) 得到预测样本矩阵predMatrixTmp。

```
predMatrixTmp[x][y] = ((predMatrixBgc [x][y] * 5 + predPlane[x][y] * 3 + 4) >> 3)
```

第3步，否则，如果InterPfIndex为1，按以下方法得到预测样本矩阵predMatrixTmp。其中，如果当前预测块是亮度块，x的取值范围为0～M-1，y的取值范围为0～N-1；如果当前预测块是色度块，x的取值范围为0～K-1，y的取值范围为0～L-1。

```
predMatrixTmp[x][y] = Clip1((f[x] * leftPel[y+1] + f[y] * topPel[x+1] + (64 - f[x] - f[y]) * predMatrixBgc [x][y] + 32 ) >> 6)
```

第4步，得到帧间预测的预测样本矩阵predMatrixInterPf。其中，如果当前预测块是亮度块，x的取值范围为0～M-1，y的取值范围为0～N-1；如果当前预测块是色度块，x的取值范围为0～K-1，y的取值范围为0～L-1。

```
predMatrixInterPf [x][y] = predMatrixTmp[x][y]
```

9.8.4.3中，M×N亮度块内坐标为(x,y)的像素对应的滤波系数f[x]和f[y]分别由M、x和N、y查表107得到；K×L色度块内坐标为(x,y)的像素对应的滤波系数f[x]和f[y]分别由K、x和L、y查表107得到。

9.8.4.4 导出帧间预测修正后的预测样本

输入：预测样本矩阵predMatrixInterPf，以及亮度块的宽度M和高度N，色度块的宽度K和高度L。

输出：帧间预测的预测样本矩阵predMatrixInter。

分别按9.7.1.2和9.7.1.3得到当前块的亮度和色度参考样本。

对于亮度块，当前块上边的参考样本记为r[i]，左边的参考样本记为c[j]，其中i=0～M，j=0～N。令rp[i+1]等于亮度predMatrixInterPf[i][0]（i=0～M-1），cp[j+1]等于亮度predMatrixInterPf[0][j](j=0～N-1)。

对于色度块，当前块上边的参考样本记为row[i]，左边的参考样本记为col[j]，其中i=0～K，j=0～L。令rowp[i+1]等于色度predMatrixInterPf[i][0]（i=0～K-1），colp[j+1]等于色度predMatrixInterPf[0][j](j=0～L-1)。

亮度块的处理如下：

a) 如果满足以下条件之一，则：

1) r[i]（i=1～M）、c[j]（j=1～N）均“不可用”；

2) InterPcIndex等于1且r[i]（i=1～M）均“不可用”；

3) InterPcIndex等于2且c[j]（j=1～N）均“不可用”。

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        predMatrixInter[x][y] = 1 << (Bitdepth - 1)
    }
}
```

b) 否则，执行以下操作。

1) 如果当前编码单元子类型为‘P_Skip_SbTemporal’‘P_Direct_SbTemporal’‘P_Skip_Mvap’‘P_Direct_Mvap’‘B_Skip_SbTemporal’‘B_Direct_SbTemporal’‘B_Skip_Mvap’或‘B_Direct_Mvap’，则P等于8；否则，P等于16。

2) 如果M大于P，则ML等于P；否则ML等于M。如果N大于P，则NL等于P；否则NL等于N。

3) 如果InterPcIndex等于0且r[i]（i=1～M）、c[j]（j=1～N）均“可用”，则：

```
posA0 = 0
posL0 = 0
if (ML >= NL) {
    posA1 = ML - ML / NL
    posL1 = NL - 1
}
else {
    posA1 = ML - 1
    posL1 = NL - NL / ML
}
x[0] = rp[posA0+2]
y[0] = r[posA0+2]
x[1] = rp[Min(posA1+2, ML)]
y[1] = r[Min(posA1+2, ML)]
x[2] = cp[posL0+2]
y[2] = c[posL0+2]
x[3] = cp[Min(posL1+2, NL)]
y[3] = c[Min(posL1+2, NL)]
```

4) 否则，如果InterPcIndex等于0且r[i]均“可用”且c[j]均“不可用”（i=1～M，j=1～N），或InterPcIndex等于1且r[i]均“可用”（i=1～M）：

```
posA0 = 0
posA1 = ML / 4
posA2 = 2 * (ML / 4)
posA3 = 3 * (ML / 4)
x[0] = rp[posA0+1]
y[0] = r[posA0+1]
x[1] = rp[posA1+1]
y[1] = r[posA1+1]
```

```
x[2] = rp[posA2+1]
y[2] = r[posA2+1]
x[3] = rp[posA3+1]
y[3] = r[posA3+1]
```

5) 否则，如果 InterPcIndex 等于 0 且 r[i]均“不可用”且 c[j]均“可用”（i=1～M，j=1～N），或 InterPcIndex 等于 2 且 c[j]均“可用”j=（1～N）：

```
posL0 = 0
posL1 = NL / 4
posL2 = 2 * (NL / 4)
posL3 = 3 * (NL / 4)
x[0] = cp[posL0+1]
y[0] = c[posL0+1]
x[1] = cp[posL1+1]
y[1] = c[posL1+1]
x[2] = cp[posL2+1]
y[2] = c[posL2+1]
x[3] = cp[posL3+1]
y[3] = c[posL3+1]
```

6) 令 minIndex[2]={0,2}，maxIndex[2]={1,3}，执行以下操作：

```
if (x[minIndex[0]] > x[minIndex[1]]) {
    Exchange(minIndex[0], minIndex[1])
}
if (x[maxIndex[0]] > x[maxIndex[1]]) {
    Exchange(maxIndex[0], maxIndex[1])
}
if (x[minIndex[0]] > x[maxIndex[1]]) {
    Exchange(minIndex[0], maxIndex[0])
    Exchange(minIndex[1], maxIndex[1])
}
if (x[minIndex[1]] > x[maxIndex[0]]) {
    Exchange(minIndex[1], maxIndex[0])
}
xMin = (x[minIndex[0]]+x[minIndex[1]]+1)>>1
yMin = (y[minIndex[0]]+y[minIndex[1]]+1)>>1
xMax = (x[maxIndex[0]]+x[maxIndex[1]]+1)>>1
yMax = (y[maxIndex[0]]+y[maxIndex[1]]+1)>>1
diffX = xMax  -  xMin
diffY = yMax  -  yMin
iShift = 16
```

7) 执行以下操作：

```
if (diffX > 64) {
    shift = (BitDepth > 8) ? BitDepth - 6 : 2
    add = 1 << (shift - 1)
    α = (diffY * TscpmTable[((diffX + add) >> shift) - 1] + add) >> shift
    β = yMin - ((α * xMin) >> iShift)
}
else if (diffX > 0) {
    α = (diffY * TscpmTable[diffX-1])
    β = yMin - ((α * xMin) >> iShift)
}
else {
    α = 0
    β = yMin
}
```

8) 导出修正后的预测值：

```
for (x=0; x<M; x++) {
    for (y=0; y<N; y++) {
        predMatrixInter[x][y] = Clip1(((α * predMatrixPf[x][y]) >> iShift) + β)
    }
}
```

色度块的处理如下：

a) 如果满足以下条件之一，则：

1) row[i]（i=1～K）、col[j]（j=1～L）均“不可用”；
2) InterPcIndex 等于 1 且 row[i]（i=1～K）均“不可用”；
3) InterPcIndex 等于 2 且 col[j]（j=1～L）均“不可用”。

```
for (x=0; x<K; x++) {
    for (y=0; y<L; y++) {
        predMatrixInter[x][y] = 1 << (Bitdepth - 1)
    }
}
```

b) 否则，执行以下操作：

1) 如果当前编码单元子类型为‘P_Skip_SbTemporal’‘P_Direct_SbTemporal’‘P_Skip_Mvap’‘P_Direct_Mvap’‘B_Skip_SbTemporal’‘B_Direct_SbTemporal’‘B_Skip_Mvap’或‘B_Direct_Mvap’，则 P 等于 4；否则，P 等于 8。
2) 如果 K 大于 P，则 KL 等于 P；否则 KL 等于 K。如果 L 大于 P，则 LL 等于 P；否则 LL 等于 L。
3) 如果 InterPcIndex 等于 0 且 row[i]（i=1～K）、col[j]（j=1～L）均“可用”：

```
posA0 = 0
posL0 = 0
```

```
if (LL >= LL) {
    posA1 = KL  -  KL / LL
    posL1 = LL  -  1
}
else {
    posA1 = KL  -  1
    posL1 = LL  -  LL / KL
}
x[0] = rowp[posA0+2]
y[0] = row[posA0+2]
x[1] = rowp[Min(posA1+2, KL)]
y[1] = row[Min(posA1+2, KL)]
x[2] = colp[posL0+2]
y[2] = col[posL0+2]
x[3] = colp[Min(posL1+2, LL)]
y[3] = col[Min(posL1+2, LL)]
```

4) 否则，如果 InterPcIndex 等于 0 且 row[i]均“可用”且 col[j]均“不可用”（i=1～K，j=1～L），或 InterPcIndex 等于 1 且 row[i]均“可用”（i=1～K）：

```
posA0 = 0
posA1 = KL / 4
posA2 = 2 * (KL / 4)
posA3 = 3 * (KL / 4)
x[0] = rowp[posA0+1]
y[0] = row[posA0+1]
x[1] = rowp[posA1+1]
y[1] = row[posA1+1]
x[2] = rowp[posA2+1]
y[2] = row[posA2+1]
x[3] = rowp[posA3+1]
y[3] = row[posA3+1]
```

5) 否则，如果 InterPcIndex 等于 0 且 row[i]均“不可用”且 col[j]均“可用”（i=1～K，j=1～L），或 InterPcIndex 等于 2 且 col[j]均“可用”j=（1～L）：

```
posL0 = 0
posL1 = LL / 4
posL2 = 2 * (LL / 4)
posL3 = 3 * (LL / 4)
x[0] = colp[posL0+1]
y[0] = col[posL0+1]
x[1] = colp[posL1+1]
y[1] = col[posL1+1]
```

```
x[2] = colp[posL2+1]
y[2] = col[posL2+1]
x[3] = colp[posL3+1]
y[3] = col[posL3+1]
```

6) 令 minIndex[2]={0,2}，maxIndex[2]={1,3}，执行以下操作：

```
if (x[minIndex[0]] > x[minIndex[1]]) {
    Exchange(minIndex[0], minIndex[1])
}
if (x[maxIndex[0]] > x[maxIndex[1]]) {
    Exchange(maxIndex[0], maxIndex[1])
}
if (x[minIndex[0]] > x[maxIndex[1]]) {
    Exchange(minIndex[0], maxIndex[0])
    Exchange(minIndex[1], maxIndex[1])
}
if (x[minIndex[1]] > x[maxIndex[0]]) {
    Exchange(minIndex[1], maxIndex[0])
}
xMin = (x[minIndex[0]]+x[minIndex[1]]+1)>>1
yMin = (y[minIndex[0]]+y[minIndex[1]]+1)>>1
xMax = (x[maxIndex[0]]+x[maxIndex[1]]+1)>>1
yMax = (y[maxIndex[0]]+y[maxIndex[1]]+1)>>1
diffX = Abs(xMax - xMin)
diffY = Abs(yMax - yMin)
if (diffX > 64) {
    shift = (BitDepth > 8) ? BitDepth - 6 : 2
    add = 1 << (shift - 1)
    diffX = TscpmTable[((diffX + add) >> shift) - 1]
}
```

7) 如果 diffX 小于等于 1<<(Bitdepth-8)或 diffY 小于等于 1<<(Bitdepth-8)，则 predMatrixInter[x][y]等于 Clip1((yMin+yMax)/2)，(x=0～K-1,y=0～L-1)；否则，predMatrixInter 等于 predMatrixInterPf。

9.9 预测补偿

9.9.1 概述

本条定义预测补偿的过程。

先按9.9.2导出预测样本矩阵predMatrixTmp，再按9.9.3由predMatrixTmp导出补偿后样本矩阵CompMatrix。

如果当前编码单元的分量模式是‘COMPONENT_CHROMA’且PictureIscEvsUbvsEnableFlag的值等于1，按9.9.4导出常现位置色度样本值。

9.9.2 导出预测样本矩阵

本条导出预测样本矩阵predMatrixTmp。

令当前预测单元的宽度和高度分别是W和H，x的取值范围为0～W-1，y的取值范围为0～H-1。

如果当前预测单元的预测类型是普通帧内预测：

```
for (x=0; x<W; x++) {
    for (y=0; y<H; y++) {
        predMatrixTmp[x][y] = predMatrix[x][y]
    }
}
```

如果当前预测单元的预测类型是块复制帧内预测：

```
for (x=0; x<W; x++) {
    for (y=0; y<H; y++) {
        predMatrixTmp[x][y] = predMatrixIbc[x][y]
    }
}
```

如果当前预测单元的预测类型是串复制帧内预测：

```
for (x=0; x<W; x++) {
    for (y=0; y<H; y++) {
        predMatrixTmp[x][y] = predMatrixIsc[x][y]
    }
}
```

如果当前预测单元的预测类型是帧间预测：

```
for (x=0; x<W; x++) {
    for (y=0; y<H; y++) {
        predMatrixTmp[x][y] = predMatrixInter[x][y]
    }
}
```

predMatrix是帧内预测的预测样本矩阵，predMatrixIbc是当前编码帧未滤波重建图像的预测样本矩阵，predMatrixInter是帧间预测的预测样本矩阵。

9.9.3 导出补偿后样本矩阵

本条导出补偿后样本矩阵CompMatrix。

令当前预测单元的宽度和高度分别是W和H，x的取值范围为0～W-1，y的取值范围为0～H-1，补偿后样本矩阵如下：

```
for (x=0; x<W; x++) {
    for (y=0; y<H; y++) {
        CompMatrix[x][y] = Clip1(predMatrixTmp[x][y] + ResidueMatrix[x][y])
    }
}
```

9.9.4 常现位置色度样本坐标

令k从0～PrevPvBufSize-1，如果PrevCompLumaFreqOccurPos[k]等于1，执行以下操作。

a) 记当前编码单元的左上角样本在当前图像的色度样本矩阵中的位置为（xC,yC），当前最大编码单元行左上角样本在当前图像中的坐标为（LcuRx0,LcuRy0），历史点预测信息表PrevPpInfoList中第k个点矢量(PrevPpInfoList[k][0],PrevPpInfoList[k][1])为(P_vX_k,P_vY_k)。

b) 计算（x_k,y_k）。

$$x_k = (P_vX_k >> 1) + (LcuRx0 >> 1) - xC$$
$$y_k = (P_vY_k >> 1) + (LcuRy0 >> 1) - yC$$

c) 将CompMatrix[x_k][y_k]作为LcuRowBufC[P_vX_k>>1][P_vY_k>>1]用于后续在分量模式为‘COMPONENT_LUMACHROMA’的非普通串子模式中从LcuRowBufC中取出常现位置的色度分量完成色度预测样本导出，见9.7.3.3，其中C为U或V。

d) 令PrevCompLumaFreqOccurPos[k]等于0,即该常现位置以及点矢量从此等同于来自分量模式为‘COMPONENT_LUMACHROMA’的编码单元。

9.10 去块效应滤波

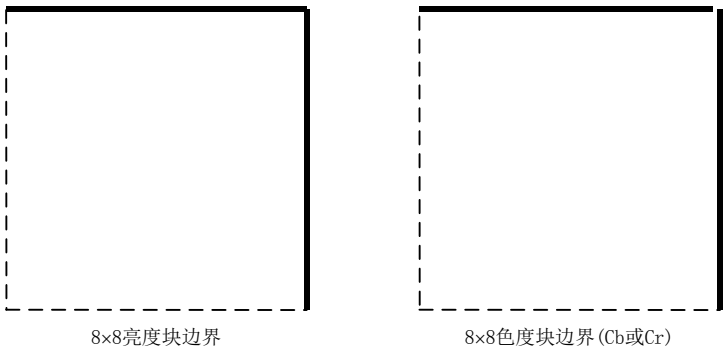
9.10.1 概述

对亮度和色度分别做去块效应滤波。去块效应滤波的单位是滤波块，按照光栅扫描顺序依次处理每个滤波块。亮度滤波块的尺寸是8×8，色度滤波块的尺寸是8×8。每个滤波块包括一条垂直边界和一条水平边界，见图41中粗线。亮度滤波块的每条边界的长度为8个亮度样本，平均分为8段。色度滤波块的每条边界的长度是8个色度样本，平均分为8段。

对每个滤波块，首先滤波垂直边界，然后滤波水平边界。

对每条边界，首先按9.10.2判断该边界是否需要滤波。如果需要滤波，则按9.10.3确定去块滤波调整参数，并按9.10.4计算该条边界的每段边界的滤波强度，然后根据该段边界的滤波强度进行去块效应滤波（见9.10.5～9.10.10）；否则，直接将补偿后样本的值作为滤波后样本的值。

当前边界两侧的样值可能在以前的去块效应滤波过程中已经被修改,当前边界的滤波的输入为这些可能被修改的样值。当前滤波块的垂直边界的滤波过程中修改的样值作为水平边界滤波过程的输入。



注：实线为滤波块的垂直边界和水平边界，虚线为其他滤波块待滤波的边界。

图41 滤波单元中需要处理的边界

9.10.2 是否跳过去块效应滤波的判断

满足以下条件之一的边界不需要滤波：

——待滤波边界是图像边界；

——待滤波边界是片边界且 CplfEnableFlag 为 0；

——待滤波边界是亮度滤波边界，且该亮度滤波边界不是亮度编码块或亮度变换块的边界；

——待滤波边界是色度滤波边界，且该色度滤波边界不是色度编码块或色度变换块的边界，且该色度滤波边界对应的亮度滤波边界不是亮度编码块的边界；

——待滤波边界是亮度滤波边界，且该亮度滤波边界所在的编码单元的 SbtCuFlag 为 1，且该亮度滤波边界不是亮度编码块的边界；

——待滤波边界所在的编码单元的 IscCuFlag 为 1。

9.10.3 确定去块滤波调整参数

本条确定去块滤波调整参数 DbrThresold、DbrOffset0、DbrOffset1、DbrAltOffset0 和 DbrAltOffset1。

如果当前为垂直边界且PictureDbrVEnableFlag为1,或当前为水平边界且PictureDbrHEnableFlag为 1，则 PictureDbrEnableFlag 的值为 1；否则，PictureDbrEnableFlag 的值为 0。

如果当前为垂直边界且 PictureAltDbrVEnableFlag 为 1，或当前为水平边界且 PictureAltDbrHEnableFlag 为 1，则 PictureAltDbrEnableFlag 的值为 1；否则，PictureAltDbrEnableFlag 的值为 0。

对于垂直边界：

```
DbrThreshold = DbrVThreshold
DbrOffset0 = DbrVOffset0
DbrOffset1 = DbrVOffset1
DbrAltOffset0 = DbrVAltOffset0
DbrAltOffset1 = DbrVAltOffset1
```

对于水平边界：

```
DbrThreshold = DbrHThreshold
DbrOffset0 = DbrHOffset0
DbrOffset1 = DbrHOffset1
DbrAltOffset0 = DbrHAltOffset0
DbrAltOffset1 = DbrHAltOffset1
```

9.10.4 边界滤波强度的推导过程

图42表示某一段滤波块边界（用黑色粗线表示），其两侧的8个样本分别记为p_0、p_1、p_2、p_3和q_0、q_1、q_2、q_3。

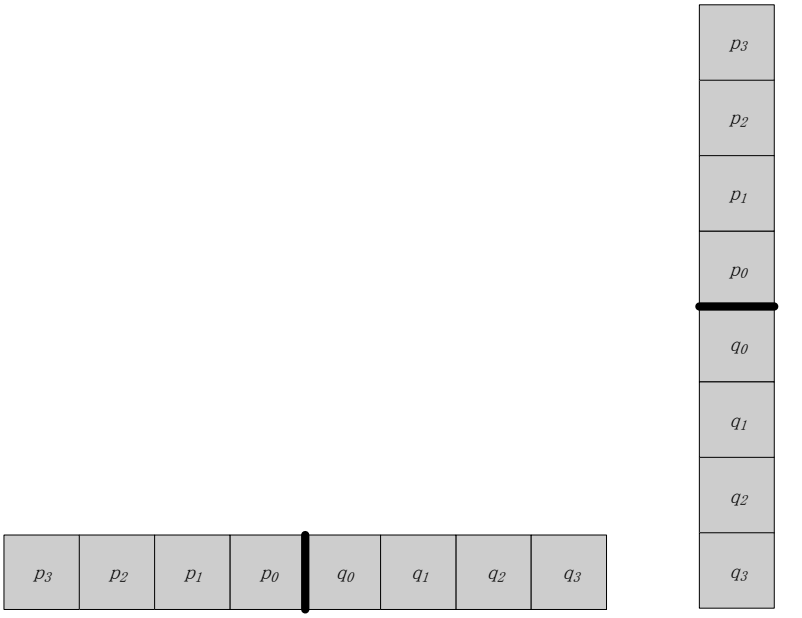


图42　滤波块边界样本

如果满足以下所有条件，则边界滤波强度B_S等于0。

a) p_0和 q_0所在的编码单元所有变换块的量化系数均为 0。其中，如果 p_0（或 q_0）是亮度样本且 p_0（或 q_0）所在的编码单元只包含亮度样本，则 p_0（或 q_0）所在的编码单元指包含 p_0（或 q_0）的亮度编码单元；如果 p_0（或 q_0）是色度样本且 p_0（或 q_0）所在的编码单元只包含色度样本，则 p_0（或 q_0）所在的编码单元指包含 p_0（或 q_0）对应亮度样本的编码单元；否则（即 p_0或 q_0所在的编码单元同时包含亮度样本和色度样本），则 p_0（或 q_0）所在的编码单元指包含 p_0（或 q_0）的编码单元。

b) p_0和 q_0所在的编码单元的预测类型不是帧内。

c) 记 B_P和 B_Q分别是 p_0和 q_0对应的 4×4 亮度编码块，B_P和 B_Q的运动信息同时满足下列条件 1 和 2 或同时满足条件 3、4 和 5。

   1) B_P和 B_Q分别对应的空域运动信息存储单元的 L0 参考索引均等于-1，或 B_P和 B_Q分别对应的空域运动信息存储单元的 L0 参考索引对应的参考帧为同一帧且空域运动信息存储单元的 L0 运动矢量的所有分量的差均小于一个整像素点。

   2) B_P和 B_Q分别对应的空域运动信息存储单元的 L1 参考索引均等于-1，或 B_P和 B_Q分别对应的空域运动信息存储单元的 L1 参考索引对应的参考帧为同一帧且空域运动信息存储单元的 L1 运动矢量的所有分量的差均小于一个整像素点。

   3) 满足以下条件之一：

      - B_P对应的空域运动信息存储单元的 L0 参考索引等于-1 且 B_Q对应的空域运动信息存储单元的 L1 参考索引等于-1；
      - B_P对应的空域运动信息存储单元的 L0 参考索引对应的参考帧与 B_Q对应的空域运动信息存储单元的 L1 参考索引对应的参考帧为同一帧且 B_P对应的空域运动信息存储单元的 L0 运动矢量和 B_Q对应的空域运动信息存储单元的 L1 运动矢量的所有分量的差均小于一个整像素点。

   4) 满足以下条件之一：

- B_Q对应的空域运动信息存储单元的L0参考索引等于-1且B_P对应的空域运动信息存储单元的L1参考索引等于-1；
- B_Q对应的空域运动信息存储单元的L0参考索引对应的参考帧与B_P对应的空域运动信息存储单元的L1参考索引对应的参考帧为同一帧且B_Q对应的空域运动信息存储单元的L0运动矢量和B_P对应的空域运动信息存储单元的L1运动矢量的所有分量的差均小于一个整像素点。

5) B_P对应的空域运动信息存储单元的L0参考索引对应的参考帧和B_Q对应的空域运动信息存储单元的L0参考索引对应的参考帧不为同一帧；B_P对应的空域运动信息存储单元的L1参考索引对应的参考帧和B_Q对应的空域运动信息存储单元的L1参考索引对应的参考帧不为同一帧。

否则，按以下方法计算边界滤波强度Bs。

a) 计算p_0和q_0所在的编码单元的平均量化参数QP_{av}。如果是亮度样本，应使用亮度编码块的量化参数；如果是色度样本，应使用色度编码块的量化参数。令p_0所在编码单元的量化参数为QP_p，q_0所在编码单元的量化参数为QP_q，平均量化参数为：

$$QP_{av} = (QP_p + QP_q + 1) >> 1$$

b) 计算索引IndexA和IndexB。

```
IndexA = Clip3(0, 63, QPav - 8 * (BitDepth - 8) + AlphaCOffset)
IndexB = Clip3(0, 63, QPav - 8 * (BitDepth - 8) + BetaOffset)
```

c) 分别根据IndexA和IndexB查表123得到α'和β'的值，再根据BitDepth得到α、β的值。

```
α = α' << (BitDepth - 8)
β = β' << (BitDepth - 8)
```

d) 如果DeblockingFilterType为1，且Abs(p0-q0)大于或等于4*α，则Bs等于0；否则，按以下方法计算Bs。

1) 将fL和fR的值均置为0，计算fS。

```
if (Abs(p0 - p1) < β) {
    fL += 2
}
if (Abs(p0 - p2) < β) {
    fL++
}
if (Abs(q0 - q1) < β) {
    fR += 2
}
if (Abs(q0 - q2) < β) {
    fR++
}
fS = fL + fR
```

2) 根据fS确定Bs。

- 当 fS 等于 6 时，如果 Abs(p_0-p_1)小于或等于β/4 且 Abs(q_0-q_1)小于或等于β/4 且 Abs(p_0-q_0)小于α，则 Bs 等于 4；否则 Bs 等于 3。
- 当 fS 等于 6 且 DeblockingFilterType 等于 1 时，如果 Abs(p0-p1)小于或等于β/4 且 Abs(q0-q1)小于或等于β/4 且 Abs(p0 - p3)小于或等于β/2 且 Abs(q0 - q3)小于或等于β/2 且 Abs(p0-q0)小于α，则 Bs 等于 4；否则 Bs 等于 3。
- 当 fS 等于 5 时，如果 p_0等于 p_1且 q_0等于 q_1，则 Bs 等于 3；否则 Bs 等于 2。
- 当 fS 等于 5 且 DeblockingFilterType 等于 1 时，如果 p0 等于 p1 且 q0 等于 q1 且 Abs(p2 - q2)小于α，则 Bs 等于 3；否则 Bs 等于 2。
- 当 fS 等于 4 时，如果 fL 等于 2，则 Bs 等于 2；否则 Bs 等于 1。
- 当 fS 等于 3 时，如果 Abs(p_1-q_1)小于β，则 Bs 等于 1；否则 Bs 等于 0。
- 当 fS 为其它值时，Bs 等于 0。

3) 如果步骤 2)得到的 Bs 不等于 0 且滤波的边界是色度编码块边界，则 Bs 减 1。

表123 IndexA 和 IndexB 与块边界阈值α'和β'与的关系

IndexA/IndexB	α′	β′	IndexA/IndexB	α′	β′	IndexA/IndexB	α′	β′	IndexA/IndexB	α′	β′
0	0	0	16	1	1	32	4	4	48	16	15
1	0	0	17	1	1	33	4	4	49	17	16
2	0	0	18	1	1	34	5	5	50	19	17
3	0	0	19	1	1	35	5	5	51	21	18
4	0	0	20	1	1	36	6	5	52	23	19
5	0	0	21	2	1	37	6	6	53	25	20
6	0	0	22	2	2	38	7	6	54	27	21
7	0	0	23	2	2	39	7	7	55	29	22
8	1	0	24	2	2	40	8	8	56	32	23
9	1	1	25	2	2	41	9	8	57	35	23
10	1	1	26	2	2	42	10	9	58	38	24
11	1	1	27	3	2	43	10	10	59	41	24
12	1	1	28	3	3	44	11	11	60	45	25
13	1	1	29	3	3	45	12	12	61	49	25
14	1	1	30	3	3	46	13	13	62	54	26
15	1	1	31	4	3	47	15	14	63	59	27

9.10.5 亮度分量 Bs 等于 4 时的边界滤波过程

边界滤波强度Bs的值为4时，对p_0、p_1、p_2和q_0、q_1、q_2滤波的计算过程如下（P_0、P_1、P_2和Q_0、Q_1、Q_2是滤波后的值）：

$$P_0 = (p_2 * 3 + p_1 * 8 + p_0 * 10 + q_0 * 8 + q_1 * 3 + 16) >> 5$$

$$P_1 = (p_2 * 4 + p_1 * 5 + p_0 * 4 + q_0 * 3 + 8) >> 4$$

$$P_2 = (p_3 * 2 + p_2 * 2 + p_1 * 2 + p_0 * 1 + q_0 * 1 + 4) >> 3$$

$$Q_0 = (p_1 * 3 + p_0 * 8 + q_0 * 10 + q_1 * 8 + q_2 \times 3 + 16) >> 5$$

$$Q_1 = (p_0 * 3 + q_0 * 4 + q_1 * 5 + q_2 * 4 + 8) >> 4$$

$$Q_2 = (p_0 * 1 + q_0 * 1 + q_1 * 2 + q_2 * 2 + q_3 * 2 + 4) >> 3$$

如果PictureDbrEnableFlag的值为1，按9.10.11调整P_0、P_1、P_2、Q_0、Q_1和Q_2的值。

9.10.6 亮度分量 Bs 等于 3 时的边界滤波过程

边界滤波强度Bs的值为3时，对p_0、p_1和q_0、q_1滤波的计算过程如下（P_0、P_1和Q_0、Q_1是滤波后的值）：

```
P0 = (p2 + (p1 << 2) + (p0 << 2) + (p0 << 1) + (q0 << 2) + q1 + 8) >> 4
P1 = ((p2 << 1) + p2 + (p1 << 3) + (p0 << 2) + q0 + 8) >> 4
Q0 = (p1 + (p0 << 2) + (q0 << 2) + (q0 << 1) + (q1 << 2) + q2 + 8) >> 4
Q1 = ((q2 << 1) + q2 + (q1 << 3) + (q0 << 2) + p0 + 8) >> 4
```

如果PictureDbrEnableFlag的值为1，按9.10.11调整P_0、P_1、Q_0和Q_1的值。

9.10.7 亮度分量 Bs 等于 2 时的边界滤波过程

边界滤波强度Bs的值为2时，对p_0和q_0滤波的计算过程如下（P_0和Q_0是滤波后的值）：

```
P0 = ((p1 << 1) + p1 + (p0 << 3) + (p0 << 1) + (q0 << 1) + q0 + 8) >> 4
Q0 = ((p0 << 1) + p0 + (q0 << 3) + (q0 << 1) + (q1 << 1) + q1 + 8) >> 4
```

如果PictureDbrEnableFlag的值为1，按9.10.11调整P_0和Q_0的值。

9.10.8 亮度分量 Bs 等于 1 时的边界滤波过程

边界滤波强度Bs的值为1时，对p_0和q_0滤波的计算过程如下（P_0和Q_0是滤波后的值）：

```
P0 = ((p0 << 1) + p0 + q0 + 2) >> 2
Q0 = ((q0 << 1) + q0 + p0 + 2) >> 2
```

如果PictureDbrEnableFlag的值为1，按9.10.11调整P_0和Q_0的值。

9.10.9 亮度分量 Bs 等于 0 时的边界滤波过程

边界滤波强度Bs的值为0且PictureAltDbrEnableFlag的值为1时，对p_0和q_0滤波调整的计算过程如下（P_0和Q_0是滤波调整后的值）：

```
DP0 = (q0 - p0+2) >> 2
if (DP0< -DbrThreshold) {
    P0 = Clip1(p0 + DbrAltOffset0)
}
else if (DP0 >DbrThreshold) {
    P0 = Clip1(p0 + DbrAltOffset1)
}
DQ0 = (p0 -q0+2) >> 2
if (DQ0 < -DbrThreshold) {
    Q0 = Clip1(q0 + DbrAltOffset0)
}
```

```
else if (DQ0> DbrThreshold) {
    Q0 = Clip1(q0 + DbrAltOffset1)
}
```

9.10.10 色度分量 Bs 大于 0 时的边界滤波过程

边界滤波强度Bs的值大于0时，对p_0和q_0滤波的计算过程如下（P_0和Q_0是滤波后的值）：

```
P0 = ((p1 << 1) + p1 + (p0 << 3) + (p0 << 1) + (q0 << 1) + q0 + 8) >> 4
Q0 = ((p0 << 1) + p0 + (q0 << 3) + (q0 << 1) + (q1 << 1) + q1 + 8) >> 4
```

边界滤波强度Bs的值等于3时，对p_1和q_1滤波的计算过程如下（P_1和Q_1是滤波后的值）：

```
P1 = ((p2 << 1) + p2 + (p1 << 3) + (p0 << 1) + p0+(q0 << 1) + 8) >> 4
Q1 = ((q2 << 1) + q2 + (q1 << 3) + (q0 << 1) + q0+(p0 << 1) + 8) >> 4
```

9.10.11 去块效应滤波调整过程

本条调整P_i和Q_i（i可为0、1或2）的值：

```
if (pi > Pi + DbrThreshold) {
    Pi = Clip1((Pi + pi + 1) >> 1 + DbrOffset0)
}
else if (pi < Pi - DbrThreshold) {
    Pi = Clip1((Pi + pi + 1) >> 1 + DbrOffset1)
}
if (qi > Qi + DbrThreshold) {
    Qi = Clip1((Qi + qi + 1) >> 1 + DbrOffset0)
}
else if (qi < Qi - DbrThreshold) {
    Qi = Clip1((Qi + qi + 1) >> 1 + DbrOffset1)
}
```

9.11 样值偏移自适应补偿

9.11.1 基本样值偏移自适应补偿

9.11.1.1 概述

如果PatchSaoEnableFlag[compIndex]的值为1，先按9.11.1.2导出基本样值偏移自适应补偿单元，再按9.11.1.3导出与当前基本样值偏移自适应补偿单元对应的基本样值偏移自适应补偿信息，最后按9.11.1.4对当前基本样值偏移自适应补偿单元内的各个样本的各分量进行操作，得到偏移后样本值。

否则，直接将滤波后样本对应分量的值作为偏移后该样本分量的值。

9.11.1.2 导出基本样值偏移自适应补偿单元

先确定当前最大编码单元，再根据当前最大编码单元按下列步骤得到与其对应的当前样值偏移自适应补偿单元：

a) 将当前最大编码单元所在亮度和色度样本区域各向左移四个样本单位后再各向上移四个样本单位，得到区域E1；
b) 如果区域E1超出当前图像边界，则将超出部分去除，得到区域E2，否则令E2等于E1；
c) 如果当前最大编码单元包含图像最右列的样本且不包含图像最下行的样本，则将区域E2的右边界向右扩展至图像的右边界，得到区域E3；
d) 否则，如果当前最大编码单元包含图像最下行的样本且不包含图像最右列的样本，则将区域E2的下边界向下扩展至图像的边界，得到区域E3；
e) 否则，如果当前最大编码单元同时包含图像最右列的样本和最下行的样本，则将区域E2的右边界向右扩展至图像的右边界后再将新区域的下边界向下扩展至图像的下边界，得到区域E3；
f) 否则，令E3等于E2；
g) 将区域E3作为当前样值偏移自适应补偿单元。

9.11.1.3 导出基本样值偏移自适应补偿信息

本条定义了样值偏移自适应补偿单元中各分量的样值偏移自适应补偿操作所需信息的导出方法。这些信息包括：样值偏移自适应补偿模式SaoMode[compIndex]、样值偏移自适应补偿的偏移值SaoOffset[compIndex][j]、样值偏移自适应补偿边缘模式类型SaoEdgeType[compIndex]、样值偏移自适应补偿区间模式的起始偏移子区间位置SaoIntervalStartPos[compIndex]和样值偏移自适应补偿区间模式的起始偏移子区间的位置差值SaoIntervalDeltaPosMinus2[compIndex]。

第1步，如果SaoMergeLeftAvai为1或SaoMergeUpAvai为1，则MergeFlagExist的值为1；否则MergeFlagExist的值为0。根据SaoMergeLeftAvai、SaoMergeUpAvai和SaoMergeTypeIndex查表124得到样值偏移自适应补偿合并模式SaoMergeMode。

表124 样值偏移自适应补偿模式

SaoMergeLeftAvai的值	SaoMergeUpAvai的值	SaoMergeTypeIndex的值	样值偏移自适应补偿合并模式
0	0	—	SAO_NON_MERGE
1	0	0	SAO_NON_MERGE
		1	SAO_MERGE_LEFT
0	1	0	SAO_NON_MERGE
		1	SAO_MERGE_UP
1	1	0	SAO_NON_MERGE
		1	SAO_MERGE_LEFT
		2	SAO_MERGE_UP

第2步，导出SaoMode[compIndex]：
a) 如果SaoMergeMode的值是‘SAO_NON_MERGE’，从位流中解析得到SaoMode[compIndex]的值；
b) 如果SaoMergeMode的值是‘SAO_MERGE_LEFT’，SaoMode[compIndex]的值等于左边样值偏移自适应补偿单元的SaoMode[compIndex]的值；
c) 如果SaoMergeMode值是‘SAO_MERGE_UP’，SaoMode[compIndex]的值等于上边样值偏移自适应补偿单元的SaoMode[compIndex]的值。

第3步，导出SaoOffset[compIndex]、SaoIntervalStartPos[compIndex]、SaoIntervalDeltaPosMinus2[compIndex]和SaoEdegType[compIndex]。

a) 如果SaoMode[compIndex]的值是‘SAO_Interval’，先按以下方法导出SaoOffset[compIndex][j]（j=0～3）、SaoIntervalStartPos[compIndex]和SaoIntervalDeltaPosMinus2[compIndex],然后按9.11.1.4.1进行样值偏移自适应补偿操作：
   1) 如果SaoMergeMode的值为‘SAO_NON_MERGE’，从位流中解析得到SaoIntervalOffsetAbs[compIndex][j](j=0～3)、SaoIntervalOffsetSign[compIndex][j]（j=0～3）、SaoIntervalStartPos[compIndex]和SaoIntervalDeltaPosMinus2[compIndex]，并计算SaoOffset[compIndex][j]。

```
for (j=0; j<4; j++) {
    SaoOffset[compIndex][j] = SaoIntervalOffsetAbs[compIndex][j] * SaoIntervalOffsetSign[compIndex][j]
}
```

   2) 否则，如果SaoMergeMode的值是‘SAO_MERGE_LEFT’，SaoOffset[compIndex][j]（j=0～3）的值等于左边样值偏移自适应补偿单元的SaoOffset[compIndex][j]（j=0～3）的值，SaoIntervalStartPos[compIndex]和SaoIntervalDeltaPosMinus2[compIndex]的值等于左边样值偏移自适应补偿单元的SaoIntervalStartPos[compIndex]和SaoIntervalDeltaPosMinus2[compIndex]的值。
   3) 否则，SaoOffset[compIndex][j]（j=0～3）的值等于上边样值偏移自适应补偿单元的SaoOffset[compIndex][j]（j=0～3）的值，SaoIntervalStartPos[compIndex]和SaoIntervalDeltaPosMinus2[compIndex]的值等于上边样值偏移自适应补偿单元的SaoIntervalStartPos[compIndex]和SaoIntervalDeltaPosMinus2[compIndex]的值。

b) 否则，如果SaoMode[compIndex]的值是‘SAO_Edge’，先按以下方法导出SaoOffset[compIndex][j]（j=0～3）和SaoEdgeType[compIndex]，再按9.11.1.4.2进行样值偏移自适应补偿操作：
   1) 如果SaoMergeMode的值是‘SAO_NON_MERGE’，从位流中解析得到SaoEdgeOffset[compIndex][j]（j=0～3）和SaoEdgeType[compIndex]的值，并计算SaoOffset[compIndex][j]。

```
for (j=0; j<4; j++) {
    SaoOffset[compIndex][j] = SaoEdgeOffset[compIndex][j]
}
```

   2) 否则，如果SaoMergeMode的值是‘SAO_MERGE_LEFT’，SaoOffset[compIndex][j]（j=0～3）和SaoEdgeType[compIndex]的值等于左边样值偏移自适应补偿单元的SaoOffset[compIndex][j]（j=0～3）和SaoEdgeType[compIndex]的值。
   3) 否则，SaoOffset[compIndex][j]（j=0～3）和SaoEdgeType[compIndex]的值等于上边样值偏移自适应补偿单元的SaoOffset[compIndex][j](j=0～3)和SaoEdgeType[compIndex]的值。

9.11.1.4 基本样值偏移自适应补偿操作

9.11.1.4.1 SAO_Interval模式的操作

如果样值偏移自适应补偿单元的SaoMode[compIndex]是‘SAO_Interval’，进行以下操作：
第1步，根据滤波后样本的compIndex分量值查表125得到该分量对应的saoOffset[compIndex]。

表125　区间模式的偏移值

样值	偏移值 （saoOffset[compIndex]）
SaoIntervalOffsetPos[compIndex][0] << shift1 ～ (SaoIntervalOffsetPos[compIndex][0] << shift1) + ((1 << shift1) - 1)	SaoOffset[compIndex][0]
SaoIntervalOffsetPos[compIndex][1] << shift1 ～ (SaoIntervalOffsetPos[compIndex][1] << shift1) + ((1 << shift1) - 1)	SaoOffset[compIndex][1]
SaoIntervalOffsetPos[compIndex][2] << shift1 ～ (SaoIntervalOffsetPos[compIndex][2] << shift1) + ((1 << shift1) - 1)	SaoOffset[compIndex][2]
SaoIntervalOffsetPos[compIndex][3] << shift1 ～ (SaoIntervalOffsetPos[compIndex][3] << shift1) + ((1 << shift1) - 1)	SaoOffset[compIndex][3]
其他	0

表125中：

```
shift1 = BitDepth - 5
SaoIntervalOffsetPos[compIndex][0] = SaoIntervalStartPos[compIndex]
SaoIntervalOffsetPos[compIndex][1] = (SaoIntervalStartPos[compIndex] + 1) % 32
SaoIntervalOffsetPos[compIndex][2]=(SaoIntervalStartPos[compIndex]+SaoIntervalDeltaPosMinus2[compIndex]
+ 2) % 32
SaoIntervalOffsetPos[compIndex][3]=(SaoIntervalStartPos[compIndex]+SaoIntervalDeltaPosMinus2[compIndex]
+ 3) % 32
```

第2步，根据当前样本的滤波后样本的compIndex分量x[compIndex]计算偏移后样本的compIndex分量y[compIndex]。

```
y[compIndex] = Clip1(x[compIndex] + saoOffset[compIndex])
```

9.11.1.4.2　SAO_Edge 模式的操作

如果样值偏移自适应补偿单元的SaoMode[compIndex]是‘SAO_Edge’，进行以下操作：

第1步，根据SaoEdgeType[compIndex]的值确定当前滤波后样本c的相邻滤波后样本a和b（见图43）。如果满足以下条件之一，则根据当前样本的滤波后样本的compIndex分量x[compIndex]计算偏移后样本的compIndex分量y[compIndex]，y[compIndex] = x[compIndex]，并结束补偿过程；否则，继续执行第2步：

——a 或 b 不与 c 处在同一片内，且 CplfEnableFlag 的值为 0；

——a 或 b 不在当前图像内。

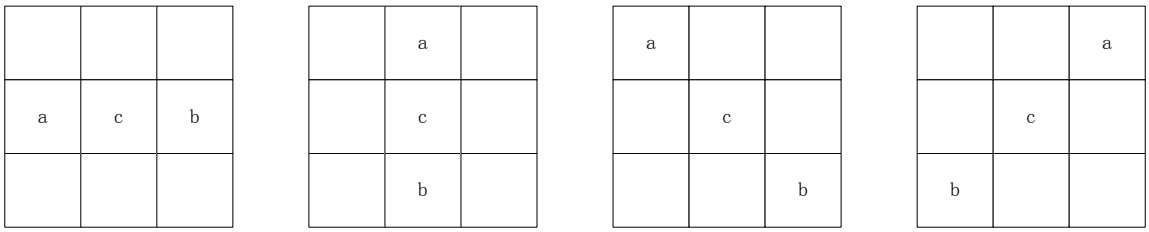


图43　当前滤波后样本和相邻滤波后样本的关系

第2步，根据表126利用当前样本c的滤波后compIndex分量值x_c与相邻样本a和b的滤波后样本compIndex分量值x_a和x_b的关系，确定当前样本compIndex分量的saoOffset[compIndex]。

表126　边缘模式的偏移值

样本分量值的关系	偏移值（saoOffset[compIndex]）
$x_c < x_a \ \&\& \ x_c < x_b$	SaoOffset[compIndex][0]
$(x_c < x_a \ \&\& \ x_c == x_b) \ \|\| \ (x_c == x_a \ \&\& \ x_c < x_b)$	SaoOffset[compIndex][1]
$(x_c > x_a \ \&\& \ x_c == x_b) \ \|\| \ (x_c == x_a \ \&\& \ x_c > x_b)$	SaoOffset[compIndex][2]
$x_c > x_a \ \&\& \ x_c > x_b$	SaoOffset[compIndex][3]
其他	0

第3步，根据当前样本的滤波后样本的compIndex分量x[compIndex]计算偏移后样本的compIndex分量y[compIndex]。

```
y[compIndex] = Clip1(x[compIndex] + saoOffset[compIndex])
```

9.11.1.4.3　SAO_Off 模式的操作

如果样值偏移自适应补偿单元的SaoMode[compIndex]是‘SAO_Off’，直接将当前样本滤波后样本compIndex分量的值作为该样本偏移后样本compIndex分量值。

9.11.2　增强样值偏移自适应补偿

9.11.2.1　概述

如果PictureEsaoEnableFlag[compIndex]为1，先按9.11.2.2导出增强样值偏移自适应补偿单元，再按9.11.2.3对增强样值偏移自适应补偿单元内的各个样本的各个分量进行操作，得到偏移后样本值；否则，直接将滤波后样本对应分量的值作为偏移后该样本分量的值。

9.11.2.2　导出增强样值偏移自适应补偿单元

先确定当前最大编码单元，如果当前分量为色度分量，则直接将当前最大编码单元所对应的色度样本区域作为色度增强样值偏移自适应补偿单元；否则，将当前最大编码单元按下列步骤得到与其对应的增强样值偏移自适应补偿单元。

第1种情况，CplfEnableFlag的值等于0。

a)　当前最大编码单元所在亮度样本区域为 E1。

b)　如果区域 E1 超出当前片边界，则将超出部分去除，得到区域 E2；否则令 E2 等于 E1。

c) 如果当前最大编码单元包含当前片最左列的样本或最右列的样本或最上行的样本或最下行的样本，则：
   1) 如果当前最大编码单元包含当前片最左列的样本，则将区域 E2 的左边界向右缩进一列，得到区域 E3；
   2) 如果当前最大编码单元包含当前片最右列的样本，则将区域 E2 的右边界向左缩进一列，得到区域 E3；
   3) 如果当前最大编码单元包含当前片最上行的样本，则将区域 E2 的上边界向下缩进一行，得到区域 E3；
   4) 如果当前最大编码单元包含当前片最下行的样本，则将区域 E2 的下边界向上缩进一行，得到区域 E3。
d) 否则，令 E3 等于 E2。
e) 将区域 E3 作为当前增强样值偏移自适应补偿单元。

第2种情况，CplfEnableFlag的值等于1。

a) 当前最大编码单元所在亮度样本区域为 E1。
b) 如果区域 E1 超出当前图像边界，则将超出部分去除，得到区域 E2；否则令 E2 等于 E1。
c) 如果当前最大编码单元包含图像最左列的样本或最右列的样本或最上行的样本或最下行的样本，则：
   1) 如果当前最大编码单元包含图像最左列的样本，则将区域 E2 的左边界向右缩进一列，得到区域 E3；
   2) 如果当前最大编码单元包含图像最右列的样本，则将区域 E2 的右边界向左缩进一列，得到区域 E3；
   3) 如果当前最大编码单元包含图像最上行的样本，则将区域 E2 的上边界向下缩进一行，得到区域 E3；
   4) 如果当前最大编码单元包含图像最下行的样本，则将区域 E2 的下边界向上缩进一行，得到区域 E3。
d) 否则，令 E3 等于 E2。
e) 将区域 E3 作为当前增强样值偏移自适应补偿单元。

9.11.2.3 增强样值偏移自适应补偿操作

如果 EsaoLcuEnableFlag[compIndex][LcuIndex] 等于 1，则对 compIndex 分量使用第 EsaoLcuSetIndex[compIndex][LcuIndex]组偏移值进行增强样值偏移自适应补偿；否则不进行增强样值偏移自适应补偿。亮度分量的增强样值偏移自适应补偿操作如下：

```
setIndex = EsaoLcuSetIndex[compIndex][LcuIndex]
C = 0
C1 = 0
for (a=-1; a<=1; a++) {
    for (b=-1; b<=1; b++) {
        if (a != 0 && b != 0) {
            if (EsaoLumaParamType[setIndex] == 0) {
                C1+=( p(x, y)< p(x+a, y+b) ? 1 : ( p(x, y)== p(x+a, y+b)?0:-1))
            }
            else {
```

```
                C1 += (p(x, y) < p(x+a, y+b) ? 1 : 0)
            }
        }
    }
}
C1 = (EsaoLumaParamType[setIndex] == 0) ? C1+8 : C1
C = ((p(x,y) * PictureEsaoAdaptiveParam[setIndex]) >> BitDepth) * (EsaoLumaParamType[setIndex] ? 9 : 17)
+ C1
p'(x, y) = Clip1(p(x, y) + EsaoFilterCoeff[0][setIndex][C])
```

其中，p(x,y)是滤波后样本，p'(x,y)是增强样值偏移自适应补偿后样本。

色度分量的增强样值偏移自适应补偿操作如下：

```
setIndex = EsaoLcuSetIndex[compIndex][LcuIndex]
C = ((p(x,y) * EsaoTableUV[PictureEsaoAdaptiveParamIndex[compIndex][setIndex]])) >> BitDepth
p'(x, y) = Clip1(p(x, y) + EsaoFilterCoeff[compIndex][setIndex][C])
```

其中,p(x,y)是滤波后样本,p'(x,y)是增强样值偏移自适应补偿后样本。EsaoTableUV的值见表127。

表127 EsaoTableUV 的值

k	EsaoTableUV的值
0~59	(k+1)*2
60	122
61	124
62	128
63	132
64	136
65	140
66	144
67	148
68	152
69	156
70	160
71	164
72	168
73	172
74	176
75	180
76	184
77	188
78	192
79	196
80	200

表 127（续）

k	EsaoTableUV的值
81	204
82	208
83	212
84	216
85	220
86	224
87	228
88	232
89	236
90	240
91	244
92	248
93	256
94	264
95	272

9.11.3 跨分量样值偏移自适应补偿

9.11.3.1 概述

如果PictureCcsaoEnableFlag[compIndex]的值为1，先按9.11.3.2导出跨分量样值偏移自适应补偿单元，然后按9.11.3.3对当前跨分量样值偏移自适应补偿单元内的各分量样本进行操作，得到跨分量偏移后样本值。

否则，直接将偏移后样本对应分量的值作为跨分量偏移后该样本分量的值。

9.11.3.2 导出跨分量样值偏移自适应补偿单元

先确定当前最大编码单元，再根据当前最大编码单元按下列步骤得到与其对应的当前跨分量样值偏移自适应补偿单元。

第1种情况，CplfEnableFlag的值等于0：

a) 当前最大编码单元所在色度样本区域为 E1；
b) 如果区域 E1 超出当前片边界，则将超出部分去除，得到区域 E2，否则令 E2 等于 E1；
c) 如果当前最大编码单元包含当前片最左列的样本且不包含当前片最上行的样本，则将区域 E2 的左边界向右缩进一列，得到区域 E3；
d) 否则，如果当前最大编码单元包含当前片最上行的样本且不包含当前片最左列的样本，则将区域 E2 的上边界向下缩进一行，得到区域 E3；
e) 否则，如果当前最大编码单元同时包含当前片最左列的样本和最上行的样本，则将区域 E2 的左边界向右缩进一列后再将新区域的上边界向下缩进一行，得到区域 E3；
f) 否则，令 E3 等于 E2；
g) 将区域 E3 作为当前跨分量样值偏移自适应补偿单元。

第2种情况，CplfEnableFlag的值等于1：

a) 当前最大编码单元所在色度样本区域为 E1；

b) 如果区域E1超出当前图像边界，则将超出部分去除，得到区域E2，否则令E2等于E1；
c) 如果当前最大编码单元包含图像最左列的样本且不包含图像最上行的样本，则将区域E2的左边界向右缩进一列，得到区域E3；
d) 否则，如果当前最大编码单元包含图像最上行的样本且不包含图像最左列的样本，则将区域E2的上边界向下缩进一行，得到区域E3；
e) 否则，如果当前最大编码单元同时包含图像最左列的样本和最上行的样本，则将区域E2的左边界向右缩进一列后再将新区域的上边界向下缩进一行，得到区域E3；
f) 否则，令E3等于E2；
g) 将区域E3作为当前跨分量样值偏移自适应补偿单元。

9.11.3.3 跨分量样值偏移自适应补偿操作

9.11.3.3.1 概述

如果CcSaoLCUEnableFlag[compIndex][LcuIndex]等于1，则对compIndex分量进行跨分量样值偏移自适应补偿；否则不进行跨分量样值偏移自适应补偿。

如果PictureCcsaoMode[compIndex][CcsaoLcuSetIndex[compIndex][LcuIndex]]的值等于0，则按9.11.3.3.2对compIndex分量根据第CcsaoLcuSetIndex[compIndex][LcuIndex]组偏移值进行区间模式跨分量样值偏移自适应补偿。否则，如果PictureCcsaoMode[compIndex][CcsaoLcuSetIndex[compIndex][LcuIndex]]的值等于1，则按9.11.3.3.3对compIndex分量根据第CcsaoLcuSetIndex[compIndex][LcuIndex]组偏移值进行边缘模式跨分量样值偏移自适应补偿。

9.11.3.3.2 区间模式跨分量样值偏移自适应补偿操作

令当前色度分量样本位置为(xC,yC)，当前色度分量滤波后样本值为p(xC,yC)，前色度分量偏移后样本值为p'(xC,yC)，对应的亮度分量样本位置为(xL,yL)，对应的亮度分量滤波后样本值为p(xL,yL)，跨分量偏移后样本值为p''(xC,yC)，区间模式的跨分量样值偏移自适应补偿操作如下。

```
setIndex = CcsaoLcuSetIndex[compIndex][LcuIndex]
xL = (xC << 1) + ccsaoCandPosX
yL = (yC << 1) + ccsaoCandPosY[lineIndex]
bandL = (p(xL, yL) * PictureCcsaoLumaBandNum[compIndex][setIndex]) >> BitDepth
bandC = (p(xC, yC) * PictureCcsaoChromaBandNum[compIndex][setIndex]) >> BitDepth
classIndex = bandL * PictureCcsaoChromaBandNum[compIndex][setIndex] + bandC
p'' (xC, yC) = Clip1(p' (xC, yC) + CcsaoOffset[compIndex][setIndex][classIndex])
```

其中ccsaoCandPosX和ccsaoCandPosY[lineIndex]的值由PictureCcsaoCandIndex[compIndex][setIndex]查表128得到。如果CplfEnableFlag的值为0且当前最大编码单元包含当前片最下行的样本，或CplfEnableFlag的值为1且当前最大编码单元包含图像最下行的样本，则lineIndex的值为0，否则lineIndex的值如下：

```
lcuHeightC = 1<< (LcuSizeInBit-1)
lineIndex = (yC % lcuHeightC) >= lcuHeightC - 4 ? (lcuHeightC - yC - 1) : 0
```

表128　跨分量样值偏移自适应补偿坐标偏移值

PictureCcsaoCandIndex[compIndex][setIndex]	ccsaoCandPosX	ccsaoCandPosY[0]	ccsaoCandPosY[1]	ccsaoCandPosY[2]	ccsaoCandPosY[3]
0	-1	-1	0	2	4
1	0	-1	0	2	4
2	1	-1	0	2	4
3	-1	0	0	2	4
4	0	0	0	2	4
5	1	0	0	2	4
6	-1	1	1	2	4
7	0	1	1	2	4
8	1	1	1	2	4

9.11.3.3.3　边缘模式跨分量样值偏移自适应补偿操作

令当前色度分量样本位置为(xC,yC)，当前色度分量滤波后样本值为 p(xC,yC)，当前色度分量偏移后样本值为 p'(xC,yC)，对应的亮度分量样本位置为(xL,yL)，对应的亮度分量滤波后样本值为 p(xL,yL)，相邻的亮度分量滤波后样本值分别为 p(xL+a1,yL+a2)和 p(xL+b1,yL+b2)。跨分量偏移后样本值为 p''(xC,yC)，边缘模式的跨分量样值偏移自适应补偿操作如下。

```
setIndex = CcsaoLcuSetIndex[compIndex][LcuIndex]
xL = (xC << 1)
yL = (yC << 1) + (lineIndex == 3 ? 4 : (lineIndex == 2 ? 2 : 0))
a1 = ccsaoEdgeTypeX[0]
a2 = ccsaoEdgeTypeY[0][lineIndex]
b1 = ccsaoEdgeTypeX[1]
b2 = ccsaoEdgeTypeY[1][lineIndex]
diff1 = p(xL,yL) - p(xL+a1, (yC<<1)+a2)
diff2 = p(xL,yL) - p(xL+b1, (yC<<1)+b2)
if (PictureCcsaoEdgeBandNumIndex[compIndex][setIndex] < 2) {
    band = (p(xL,yL) * PictureCcsaoEdgeBandNum[compIndex][setIndex]) >> BitDepth
}
else {
    band = (p(xC,yC) * PictureCcsaoEdgeBandNum[compIndex][setIndex]) >> BitDepth
}
classIndex = band *16 + C1 * 4 + C2
p' ' (xC, yC) = Clip1(p' (xC, yC) + CcsaoOffset[compIndex][setIndex][classIndex])
```

其中CX（X为1或2）的计算如下：

```
thr = PictureCcsaoEdgeThr[compIndex][setIndex]
CX = diffX < 0 ? ( diffX < -thr ? 0 : 1 ) : ( diffX < thr ? 2 : 3 )
```

ccsaoEdgeTypeX[0] 、 ccsaoEdgeTypeX[1] 、 ccsaoEdgeTypeY[0][lineIndex] 和 ccsaoEdgeTypeY[1][lineIndex]的值由PictureCcsaoEdgeType[compIndex][setIndex]查表129得到。如果CplfEnableFlag的值为0且当前最大编码单元包含当前片最下行的样本，或CplfEnableFlag的值为1且当前最大编码单元包含图像最下行的样本，则lineIndex的值为0，否则lineIndex的值如下：

```
lcuHeightC = 1<< (LcuSizeInBit-1)
lineIndex = (yC % lcuHeightC) >= lcuHeightC  -  4 ? (lcuHeightC - yC - 1) : 0
```

表129　跨分量样值偏移自适应补偿边缘模式坐标偏移值

PictureCcsaoEdgeType[compIndex][setIndex]	ccsaoEdgeTypeX[0]	ccsaoEdgeTypeX[1]	ccsaoEdgeTypeY[0][0]	ccsaoEdgeTypeY[0][1]	ccsaoEdgeTypeY[0][2]	ccsaoEdgeTypeY[0][3]	ccsaoEdgeTypeY[1][0]	ccsaoEdgeTypeY[1][1]	ccsaoEdgeTypeY[1][2]	ccsaoEdgeTypeY[1][3]
0	-1	1	0	0	2	4	0	0	2	4
1	0	0	-1	0	2	4	1	1	2	4
2	-1	1	-1	0	2	4	1	1	2	4
3	1	-1	-1	0	2	4	1	1	2	4

9.12　自适应修正滤波

9.12.1　概述

如果PictureAlfEnableFlag[compIndex]为0，直接将偏移后样本分量的值作为对应重建样本分量的值；否则，对相应的偏移后样本分量进行自适应修正滤波。

自适应修正滤波的单位是由最大编码单元导出的自适应修正滤波单元，按照光栅扫描顺序依次处理。首先按9.12.2解码各分量的自适应修正滤波系数，再按9.12.3导出自适应修正滤波单元，并按9.12.4确定当前自适应修正滤波单元亮度分量的自适应修正滤波系数索引，最后按9.12.5对自适应修正滤波单元的亮度和色度分量进行自适应修正滤波，得到重建样本。

9.12.2　自适应修正滤波系数解码

自适应修正滤波系数的解码过程如下：

第1步，对系数AlfCoeffLuma进行处理。

a)　如果EalfEnableFlag为0，从位流中解析得到亮度样本的第i组滤波系数AlfCoeffLuma[i][j]（i=0～alf_filter_num_minus1，j=0～7）。对系数AlfCoeffLuma[i][8]（即图44的系数C8）进行以下处理。

```
for (i=0; i<=alf_filter_num_minus1; i++) {
    coeffLuma = 0
    for(j=0; j<8; j++) {
        coeffLuma += 2 * AlfCoeffLuma[i][j]
    }
    AlfCoeffLuma[i][8] += 64  -  coeffLuma
}
```

其中 AlfCoeffLuma[i][j]（j=0～7）的位宽是 7 位，取值范围为-64～63。经上述处理后AlfCoeffLuma[i][8]的取值范围为 0～127。

b) 如果 EalfEnableFlag 为 1,从位流中解析得到亮度样本的第 i 组滤波系数 AlfCoeffLuma[i][j]（i=0～alf_filter_num_minus1，j=0～13）。对系数 AlfCoeffLuma[i][14]（即图 45 的系数 C14）进行以下处理。

```
for (i=0; i<=alf_filter_num_minus1; i++) {
    coeffLuma = 0
    sum = 1 << AlfLumaShift[i]
    for(j=0; j<14; j++) {
        coeffLuma += 2 * AlfCoeffLuma[i][j]
    }
    AlfCoeffLuma[i][14] += sum  -  coeffLuma
}
```

其中 AlfCoeffLuma[i][j]（j=0～13）的位宽是 7 位，取值范围为-64～63。经上述处理后AlfCoeffLuma[i][14]的取值范围为 0～127。

第2步，根据alf_region_distance[i]（i>1）得到亮度分量自适应修正滤波系数索引数组（记作alfCoeffIndexTab[FilterNum]）。

```
count = 0
alfCoeffIndexTab[0] = 0
for(i=1; i<alf_filter_num_minus1+1; i++) {
    for(j=0; j<alf_region_distance[i]-1; j++) {
        alfCoeffIndexTab[count+1] = alfCoeffIndexTab[count]
        count = count + 1
    }
    alfCoeffIndexTab[count+1] = alfCoeffIndexTab[count] + 1
    count = count + 1
}
for(i=count; i<FilterNum; i++) {
    alfCoeffIndexTab[i] = alfCoeffIndexTab[count]
}
```

第3步，对系数AlfCoeffChroma进行处理。

a) 如果 EalfEnableFlag 为 0，从位流中解析得到色度样本的滤波系数 AlfCoeffChroma[0][j]和AlfCoeffChroma[1][j]（j=0～7）。分别对系数 AlfCoeffChroma[0][8]和 AlfCoeffChroma[1][8]（即图 44 的系数 C8）进行以下处理。

```
coeffChroma0 = 0
coeffChroma1 = 0
for(j=0; j<8; j++) {
    coeffChroma0 += 2 * AlfCoeffChroma[0][j]
    coeffChroma1 += 2 * AlfCoeffChroma[1][j]
```

```
}
AlfCoeffChroma[0][8] += 64 - coeffChroma0
AlfCoeffChroma[1][8] += 64 - coeffChroma1
```

其中 AlfCoeffChroma[i][j]（j=0～7）的位宽是 7 位，取值范围为-64～63。经上述处理后 AlfCoeffChroma[i][8]的取值范围为 0～127。

b) 如果 EalfEnableFlag 为 1，从位流中解析得到色度样本的滤波系数 AlfCoeffChroma[0][j]和 AlfCoeffChroma[1][j]（j=0～13）。分别对系数 AlfCoeffChroma[0][14]和 AlfCoeffChroma[1][14]（即图 45 的系数 C14）进行以下处理。

```
coeffChroma0 = 0
coeffChroma1 = 0
sumChroma0 = 1 << AlfChromaShift[0]
sumChroma1 = 1 << AlfChromaShift[1]
for(j=0; j<14; j++) {
    coeffChroma0 += 2 * AlfCoeffChroma[0][j]
    coeffChroma1 += 2 * AlfCoeffChroma[1][j]
}
AlfCoeffChroma[0][14] += sumChroma0 - coeffChroma0
AlfCoeffChroma[1][14] += sumChroma1 - coeffChroma1
```

其中 AlfCoeffChroma[i][j]（j=0～13）的位宽是 7 位，取值范围为-64～63。经上述处理后 AlfCoeffChroma[i][14]的取值范围为 0～127。

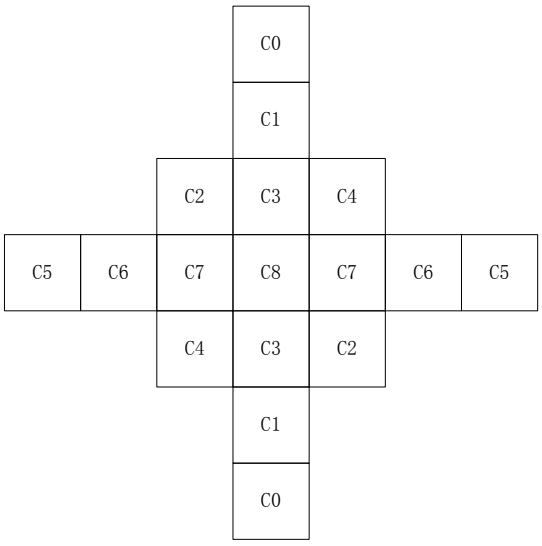


图44 自适应修正滤波系数（7×7 十字形加 3×3 方形）

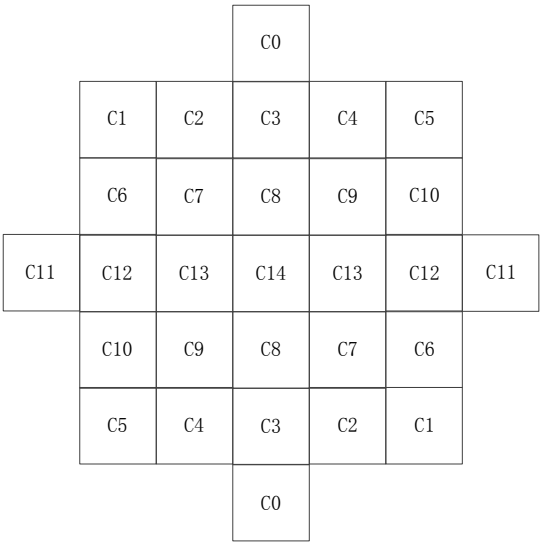


图45　自适应修正滤波系数（7×7 十字形加 5×5 方形）

9.12.3　导出自适应修正滤波单元

先确定当前最大编码单元，根据当前最大编码单元按下列步骤导出自适应修正滤波单元（见图46）。

a)　将当前最大编码单元所在样本区域超出图像边界的部分删除，得到样本区域 D。

b)　如果区域 D 的下边界所在的样本不属于图像的下边界，将亮度分量和色度分量样本区域 D 的下边界向上收缩四行，得到区域 E1；否则，令 E1 等于 D。区域 D 的最后一行样本为区域的下边界。

c)　如果区域 E1 的上边界所在的样本属于图像的上边界，或者属于片边界且 CplfEnableFlag 的值为 0，令 E2 等于 E1；否则，将亮度分量和色度分量样本区域 E1 的上边界向上扩展四行，得到区域 E2。区域 E1 的第一行样本为区域的上边界。

d)　将区域 E2 作为当前自适应修正滤波单元。图像的第一行样本为图像的上边界，最后一行样本为图像的下边界。

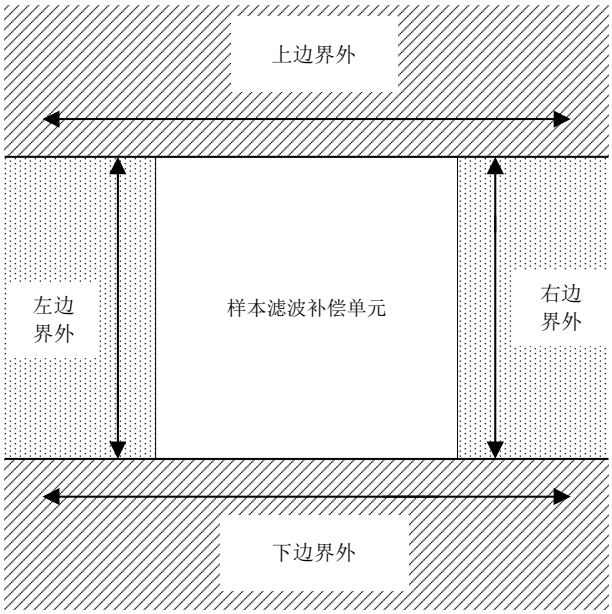


图46　自适应修正滤波单元

9.12.4　确定亮度分量自适应修正滤波单元自适应修正滤波系数索引

如果EalfEnableFlag为0，根据以下方法计算当前亮度分量自适应修正滤波单元的自适应修正滤波系数索引（记作filterIndex）。

```
xInterval = ((((horizontal_size + (1 << LcuSizeInBit) - 1) >> LcuSizeInBit) + 1) >> 2) << LcuSizeInBit
yInterval = ((((vertical_size + (1 << LcuSizeInBit) - 1) >> LcuSizeInBit) + 1) >> 2) << LcuSizeInBit
if (xInterval == 0 && yInterval == 0) {
    index = 15
}
else if (xInterval == 0) {
    index = Min(3, y / yInterval) * 4 + 3
}
else if (yInterval == 0) {
    index = Min(3, x / xInterval) + 12
}
else {
    index = Min(3, y / yInterval) * 4 + Min(3, x / xInterval)
}
filterIndex = alfCoeffIndexTab[regionTable[index]]
```

其中(x,y)是导出当前自适应修正滤波单元的最大编码单元左上角样本在图像中的坐标，regionTable定义如下。

```
regionTable[16] = {0, 1, 4, 5, 15, 2, 3, 6, 14, 11, 10, 7, 13, 12, 9, 8}
```

如果EalfEnableFlag为1，根据以下方法计算当前亮度分量自适应修正滤波单元的自适应修正滤波系数索引（记作filterIndex）。

```
lcu_width = 1 << LcuSizeInBit
lcu_height = 1 << LcuSizeInBit
y_interval = ((((PicHeightInLuma + lcu_height - 1) / lcu_height) + 4) / 8 * lcu_height)
x_interval = ((((PicWidthInLuma + lcu_width - 1) / lcu_width) + 4) / 8 * lcu_width)
if (yInterval == 0) {
    y_st_offset=0
}
else {
    y_cnt = Clip3(0, 8, (PicHeightInLuma + y_interval - 1) / y_interval)
    y_st_offset = PicHeightInLuma - y_interval * (y_cnt - 1)
    y_st_offset = (y_st_offset + lcu_height / 2) / lcu_height * lcu_height
}
if (xInterval == 0) {
    x_st_offset=0
}
else {
    x_cnt = Clip3(0, 8, (PicWidthInLuma + x_interval - 1) / x_interval)
    x_st_offset = PicWidthInLuma - x_interval * (x_cnt - 1)
    x_st_offset = (x_st_offset + lcu_width / 2) / lcu_width * lcu_width
}
y_index = (y_interval == 0) ? 7 : Clip3(0, 7, y / y_interval)
y_index_offset = y_index <<3
y_index2 = (y_interval == 0 || y < y_st_offset) ? 0 : Clip3(-1, 6, (y - y_st_offset) / y_interval) + 1
y_index_offset2 = y_index2<<3
x_index = (x_interval == 0) ? 7: Clip3(0, 7, x / x_interval)
x_index2 = (x_interval == 0 || x < x_st_offset) ? 0 : Clip3(-1, 6, (x - x_st_offset) / x_interval) + 1
if (AlfRegionOrderIndex == 0) {
    filterIndex = alfCoeffIndexTab[regionTable[0][y_index_offset + x_index]]
}
else if (AlfRegionOrderIndex == 1)
    filterIndex = alfCoeffIndexTab[regionTable[1][ y_index_offset + x_index2]]
}
else if (AlfRegionOrderIndex == 2)
    filterIndex=alfCoeffIndexTab[regionTable[2][ y_index_offset2 + x_index2]]
}
else if (AlfRegionOrderIndex == 3)
    filterIndex= alfCoeffIndexTab[regionTable[3][ y_index_offset2 + x_index]]
}
```

其中(x,y)是导出当前自适应修正滤波单元的最大编码单元左上角样本在图像中的坐标，regionTable定义如下。

```
regionTable[4][64]={
```

{63,60,59,58,5,4,3,0,62,61,56,57,6,7,2,1,49,50,55,54,9,8,13,14,48,51,52,53,10,11,12,15,47,46,33,32,31,30,17,16,44,45,34,35,28,29,18,19,43,40,39,36,27,24,23,20,42,41,38,37,26,25,22,21}
{42,43,44,47,48,49,62,63,41,40,45,46,51,50,61,60,38,39,34,33,52,55,56,59,37,36,35,32,53,54,57,58,26,27,28,31,10,9,6,5,25,24,29,30,11,8,7,4,22,23,18,17,12,13,2,3,21,20,19,16,15,14,1,0}
{21,22,25,26,37,38,41,42,20,23,24,27,36,39,40,43,19,18,29,28,35,34,45,44,16,17,30,31,32,33,46,47,15,12,11,10,53,52,51,48,14,13,8,9,54,55,50,49,1,2,7,6,57,56,61,62,0,3,4,5,58,59,60,63}
{0,1,14,15,16,19,20,21,3,2,13,12,17,18,23,22,4,7,8,11,30,29,24,25,5,6,9,10,31,28,27,26,58,57,54,53,32,35,36,37,59,56,55,52,33,34,39,38,60,61,50,51,46,45,40,41,63,62,49,48,47,44,43,42}}

9.12.5 自适应修正滤波操作

如果AlfLcuEnableFlag[compIndex][LcuIndex]等于1，则对compIndex分量进行自适应修正滤波；否则不进行自适应修正滤波。

当自适应修正滤波过程中用到的样本是自适应修正滤波单元内的样本时，直接使用该样本进行滤波；当自适应修正滤波过程中用到的样本不是自适应修正滤波单元内的样本时，按以下方式进行滤波：

a) 如果该样本在图像边界外，或在片边界外且 CplfEnableFlag 为 0，则使用自适应修正滤波单元内距离该样本最近的样本代替该样本进行滤波；
b) 否则，如果该样本在自适应修正滤波单元上边界或下边界外，则使用自适应修正滤波单元内距离该样本最近的样本代替该样本进行滤波；
c) 否则，直接使用该样本进行滤波。

如果EalfEnableFlag等于0，自适应修正滤波单元亮度分量的自适应修正滤波操作如下：

```
ptmp = AlfCoeffLuma[filterIndex][8] * p(x, y)
for(j=0; j<8; j++) {
    ptmp += AlfCoeffLuma[filterIndex][j] * (p(x-Hor[j], y-Ver[j]) + p(x+Hor[j], y+Ver[j])
}
ptmp = (ptmp + 32) >> 6
p’(x, y) = Clip3(0, (1 << BitDepth) - 1, ptmp)
```

其中，p(x,y)为偏移后样本，p’(x,y)为重建样本，Hor[j]和 Ver[j]（j=0～7）见表 130。

自适应修正滤波单元色度分量的自适应修正滤波操作如下：

```
ptmp = AlfCoeffChroma[i][8] * p(x, y)
for(j=0; j<8; j++) {
    ptmp += AlfCoeffChroma[i][j] * (p(x-Hor[j], y-Ver[j]) + p(x+Hor[j], y+Ver[j])
}
ptmp = (ptmp + 32) >> 6
p’(x, y) = Clip3(0, (1 << BitDepth) - 1, ptmp)
```

其中，p(x,y)为偏移后样本，p’(x,y)为重建样本，Hor[j]和 Ver[j]（j=0～7）见表 130。

表130 样本补偿滤波坐标偏移值

j的值	Hor[j]的值	Ver[j]的值
0	0	3
1	0	2

表 130（续）

j的值	Hor[j]的值	Ver[j]的值
2	1	1
3	0	1
4	1	-1
5	3	0
6	2	0
7	1	0

如果EalfEnableFlag等于1，自适应修正滤波单元亮度分量的自适应修正滤波操作如下：

```
ptmp = AlfCoeffLuma[filterIndex][14] * p(x, y)
offset = 1 << (AlfLumaShift[filterIndex] - 1)
for(j=0; j<14; j++) {
    ptmp += AlfCoeffLuma[filterIndex][j] * (p(x-Hor[j], y-Ver[j]) + p(x+Hor[j], y+Ver[j])
}
ptmp = (ptmp + offset) >> AlfLumaShift[filterIndex]
p’(x, y) = Clip3(0, (1 << BitDepth) - 1, ptmp)
```

其中，p(x,y)为偏移后样本，p’(x,y)为重建样本，Hor[j]和 Ver[j]（j=0～13）见表 131。

自适应修正滤波单元色度分量的自适应修正滤波操作如下：

```
ptmp = AlfCoeffChroma[i][14] * p(x, y)
offset = 1 << (AlfChromaShift[i] - 1)
for(j=0; j<14; j++) {
    ptmp += AlfCoeffChroma[i][j] * (p(x-Hor[j], y-Ver[j]) + p(x+Hor[j], y+Ver[j])
}
ptmp = (ptmp + offset) >> AlfChromaShift[i]
p’(x, y) = Clip3(0, (1 << BitDepth) - 1, ptmp)
```

其中，p(x,y)为偏移后样本，p’(x,y)为重建样本，Hor[j]和Ver[j]（j=0～13）见表131。

表131　样本补偿滤波坐标偏移值

j的值	Hor[j]的值	Ver[j]的值
0	0	3
1	2	2
2	1	2
3	0	2
4	1	-2
5	2	-2
6	2	1
7	1	1
8	0	1

表 131（续）

j的值	Hor[j]的值	Ver[j]的值
9	1	-1
10	2	-1
11	3	0
12	2	0
13	1	0

附 录 A
（规范性）
防止伪起始码方法

本附录定义防止在位流中出现伪起始码的方法。起始码的形式、含义，以及为了使起始码字节对齐而进行填充的方法见7.1.1和5.9.2。

为了防止出现伪起始码，编码时应按照以下方法处理：写入一位时，如果该位是一个字节的第二最低有效位，检查该位之前写入的22位，如果这22位都是‘0’，在该位之前插入‘10’，该位成为下一个字节的最高有效位。

解码时应按以下方法处理：每读入一个字节时，检查前面读入的两个字节和当前字节，如果这三个字节构成位串‘0000 0000 0000 0000 0000 0010’，丢弃当前字节的最低两个有效位。丢弃一个字节最低两个有效位可采用任意等效的方式，本文件不做规定。

在编码和解码时对于序列头、序列显示扩展、时域可伸缩扩展、版权扩展、内容加密扩展、目标设备显示和内容元数据扩展、参考知识图像扩展、用户数据、摄像机参数扩展、视频扩展数据保留字节中的数据不采用本附录规定的伪起始码方法。

附 录 B
（规范性）
类和级

B.1 通则

类和级提供了一种定义本文件的语法和语义的子集的手段。类和级对位流进行了各种限制，同时也规定了对某一特定位流解码所需要的解码器能力。类是本文件规定的语法、语义及算法的子集。符合某个类规定的解码器应完全支持该类定义的子集。级是在某一类下对语法元素和语法元素参数值的限定集合。在给定类的情况下，不同级往往意味着对解码器能力和存储器容量的不同要求。

本附录描述了不同类和级所对应的各种限制。所有未被限定的语法元素和参数可以取任何本文件所允许的值。如果一个解码器能对某个类和级所规定的语法元素的所有允许值正确解码，则称此解码器在这个类和级上符合本文件。如果一个位流中不存在某个类和级所不允许的语法元素，并且其所含有的语法元素的值不超过此类和级所允许的范围，则认为此位流在这个类和级上符合本文件。

profile_id和level_id定义了位流的类和级。

B.2 类

本文件定义的类应符合表B.1的规定。

表B.1 类

profile_id的值	类
0x00	禁止
0x20	基准8位类（Main 8bit profile）
0x22	基准10位类（Main 10bit profile）
0x30	加强8位类（High 8bit profile）
0x32	加强10位类（High 10bit profile）
其他	保留

对于一个给定的类，不同的级支持相同的语法子集。

基准8位类的位流满足以下条件。

——profile_id 的值应为 0x20。

——progressive_sequence 的值应为‘1’ 或“0”。如果 progressive_sequence 的值为‘0’，整个视频序列中所有图像的 top_field_first 的值均应相同。

——chroma_format 的值应为‘01’。

——sample_precision 的值应为‘001’。

——视频序列起始码与随后第一个视频序列结束码之间，或视频序列起始码与随后第一个视频编辑码之间所有编码图像的 PictureStructure 的值均应相同。

——视频序列起始码与随后第一个视频序列结束码之间，或视频序列起始码与随后第一个视频编辑码之间所有编码图像的 progressive_frame 的值均应相同。

——MiniSize 的值应为 8。

注：编码时图像宽度和图像高度处理为 MiniSize 的整数倍。

——log2_lcu_size_minus2 的取值范围应为 3～5。
——log2_min_cu_size_minus2 的值应为 0。
——patch_width_minus1 的值应为 PictureWidthInLCU-1。
——stable_patch_flag 的值应为‘1’。
——ref_colocated_patch_flag 的值应为‘0’。
——uniform_patch_flag 的值应为‘1’。
——属于同一图像的片应按 patch_index 的值从小到大依次出现。
——如果编码块的任一边长大于 64，该编码块只应选择跳过模式或帧间预测模式。如果选择帧间预测模式，ctp_zero_flag 的值应为‘1’。
——如果当前编码单元的亮度样本的数量小于 64，当前编码单元不应使用跳过模式和直接模式，且当前编码单元的预测单元的预测参考模式不应为‘PRED_List01’。
——MaxSplitTimes 的值应为 6。
——MaxPartRatio 的值应为 8。
——log2_max_bt_size_minus2 的值应为 4 或 5。
——如果 I 图像的最大编码单元的宽度和高度均等于 128，则该最大编码单元应使用四叉树划分且 qt_split_flag 的值应为‘1’。
——如果编码树节点的宽度等于 128 且高度等于 64 且该编码树节点被划分，则该编码树节点应使用垂直二叉树划分且 bet_split_dir_flag 的值应为‘1’。
——如果编码树节点的宽度等于 64 且高度等于 128 且该编码树节点被划分，则该编码树节点应使用水平二叉树划分且 bet_split_dir_flag 的值应为‘0’。
——如果当前序列的 library_stream_flag 为‘1’，则该序列的所有图像均应为 I 图像。
——如果位流中存在语法元素 duplicate_sequence_header_flag，则该语法元素的值应为‘1’。
——解码图像缓冲区中最多只应存在 1 幅知识图像。
——NumOfUpdatedLibrary 的值应为 1。
——MinLibraryUpdatePeriod 的值应为 1。
——MAX_TEMPORAL_ID 的值应为 7。
——应符合 B.3 规定的级限制。
——支持的级应包括：2.0.15、2.0.30、2.0.60、4.0.30、4.0.60、6.0.30、6.2.30、6.0.60、6.2.60、6.0.120、6.2.120、8.0.30、8.2.30、8.0.60、8.2.60、8.0.120、8.2.120、10.0.30、10.2.30、10.0.60、10.2.60、10.0.120 和 10.2.120。

基准10位类的位流应满足以下条件：

——profile_id 的值应为 0x22。
——progressive_sequence 的值应为‘1’或“0”。如果 progressive_sequence 的值为‘0’，整个视频序列中所有图像的 top_field_first 的值均应相同。
——chroma_format 的值应为‘01’。
——sample_precision 的值应为‘001’或‘010’。
——encoding_precision 的值应为‘001’或‘010’。
——视频序列起始码与随后第一个视频序列结束码之间，或视频序列起始码与随后第一个视频编辑码之间所有编码图像的 PictureStructure 的值均应相同。
——视频序列起始码与随后第一个视频序列结束码之间，或视频序列起始码与随后第一个视频编辑码之间所有编码图像的 progressive_frame 的值均应相同。
——MiniSize 的值应为 8。

注：编码时图像宽度和图像高度处理为 MiniSize 的整数倍

——log2_lcu_size_minus2 的取值范围应为 3～5。
——log2_min_cu_size_minus2 的值应为 0。
——patch_width_minus1 的值应为 PictureWidthInLCU-1。
——stable_patch_flag 的值应为‘1’。
——ref_colocated_patch_flag 的值应为‘0’。
——uniform_patch_flag 的值应为‘1’。
——属于同一图像的片应按 patch_index 的值从小到大依次出现。
——如果编码块的任一边长大于 64，该编码块只应选择跳过模式或帧间预测模式。如果选择帧间预测模式，ctp_zero_flag 的值应为‘1’。
——如果当前编码单元的亮度样本的数量小于 64，当前编码单元不应使用跳过模式和直接模式，且当前编码单元的预测单元的预测参考模式不应为‘PRED_List01’。
——MaxSplitTimes 的值应为 6。
——MaxPartRatio 的值应为 8。
——log2_max_bt_size_minus2 的值应为 4 或 5。
——如果 I 图像的最大编码单元的宽度和高度均等于 128，则该最大编码单元应使用四叉树划分且 qt_split_flag 的值应为‘1’。
——如果编码树节点的宽度等于 128 且高度等于 64 且该编码树节点被划分，则该编码树节点应使用垂直二叉树划分且 bet_split_dir_flag 的值应为‘1’。
——如果编码树节点的宽度等于 64 且高度等于 128 且该编码树节点被划分，则该编码树节点应使用水平二叉树划分且 bet_split_dir_flag 的值应为‘0’。
——如果当前序列的 library_stream_flag 为‘1’，则该序列的所有图像均应为 I 图像。
——如果位流中存在语法元素 duplicate_sequence_header_flag，则该语法元素的值应为‘1’。
——解码图像缓冲区中最多只应存在 1 幅知识图像。
——NumOfUpdatedLibrary 的值应为 1。
——MinLibraryUpdatePeriod 的值应为 1。
——MAX_TEMPORAL_ID 的值应为 7。
——应符合 B.3 规定的级限制。
——支持的级应包括：2.0.15、2.0.30、2.0.60、4.0.30、4.0.60、6.0.30、6.2.30、6.0.60、6.2.60、6.0.120、6.2.120、8.0.30、8.2.30、8.0.60、8.2.60、8.0.120、8.2.120、10.0.30、10.2.30、10.0.60、10.2.60、10.0.120 和 10.2.120。

加强8位类的位流满足以下条件：

——profile_id 的值应为 0x30。
——progressive_sequence 的值应为‘1’ 或“0”。如果 progressive_sequence 的值为‘0’，整个视频序列中所有图像的 top_field_first 的值均应相同。
——chroma_format 的值应为‘01’。
——sample_precision 的值应为‘001’。
——视频序列起始码与随后第一个视频序列结束码之间，或视频序列起始码与随后第一个视频编辑码之间所有编码图像的 PictureStructure 的值均应相同。
——视频序列起始码与随后第一个视频序列结束码之间，或视频序列起始码与随后第一个视频编辑码之间所有编码图像的 progressive_frame 的值均应相同。
——MiniSize 的值应为 8。

注：编码时图像宽度和图像高度处理为 MiniSize 的整数倍

——log2_lcu_size_minus2 的取值范围应为 3～5。

——log2_min_cu_size_minus2 的值应为 0。
——stable_patch_flag 的值应为‘1’。
——ref_colocated_patch_flag 的值应为‘0’。
——uniform_patch_flag 的值应为‘1’。
——属于同一图像的片应按 patch_index 的值从小到大依次出现。
——如果编码块的任一边长大于 64，该编码块只应选择跳过模式或帧间预测模式。如果选择帧间预测模式，ctp_zero_flag 的值应为‘1’。
——如果当前编码单元的亮度样本的数量小于 64，当前编码单元不应使用跳过模式和直接模式，且当前编码单元的预测单元的预测参考模式不应为‘PRED_List01’。
——MaxSplitTimes 的值应为 6。
——MaxPartRatio 的值应为 8。
——log2_max_bt_size_minus2 的值应为 4 或 5。
——如果 I 图像的最大编码单元的宽度和高度均等于 128，则该最大编码单元应使用四叉树划分且 qt_split_flag 的值应为‘1’。
——如果编码树节点的宽度等于 128 且高度等于 64 且该编码树节点被划分，则该编码树节点应使用垂直二叉树划分且 bet_split_dir_flag 的值应为‘1’。
——如果编码树节点的宽度等于 64 且高度等于 128 且该编码树节点被划分，则该编码树节点应使用水平二叉树划分且 bet_split_dir_flag 的值应为‘0’。
——如果当前序列的 library_stream_flag 为‘1’，则该序列的所有图像均应为 I 图像。
——如果位流中存在语法元素 duplicate_sequence_header_flag，则该语法元素的值应为‘1’。
——解码图像缓冲区中最多只应存在 1 幅知识图像。
——NumOfUpdatedLibrary 的值应为 1。
——MinLibraryUpdatePeriod 的值应为 1。
——num_of_nn_filter_minus1 的取值应为 0～3。
——MAX_TEMPORAL_ID 的值应为 7。
——应符合 B.3 规定的级限制。
——支持的级应包括：2.0.15、2.0.30、2.0.60、4.0.30、4.0.60、6.0.30、6.2.30、6.4.30、6.6.30、6.0.60、6.2.60、6.4.60、6.6.60、6.0.120、6.2.120、6.4.120、6.6.120、8.0.30、8.2.30、8.4.30、8.6.30、8.0.60、8.2.60、8.4.60、8.6.60、8.0.120、8.2.120、8.4.120、8.6.120、10.0.30、10.2.30、10.4.30、10.6.30、10.0.60、10.2.60、10.4.60、10.6.60、10.0.120、10.2.120、10.4.120 和 10.6.120。

加强10位类的位流满足以下条件：

——profile_id 的值应为 0x32。
——progressive_sequence 的值应为‘1’ 或“0”。如果 progressive_sequence 的值为‘0’，整个视频序列中所有图像的 top_field_first 的值均应相同。
——chroma_format 的值应为‘01’。
——sample_precision 的值应为‘001’或‘010’。
——encoding_precision 的值应为‘001’或‘010’。
——视频序列起始码与随后第一个视频序列结束码之间，或视频序列起始码与随后第一个视频编辑码之间所有编码图像的 PictureStructure 的值均应相同。
——视频序列起始码与随后第一个视频序列结束码之间，或视频序列起始码与随后第一个视频编辑码之间所有编码图像的 progressive_frame 的值均应相同。
——MiniSize 的值应为 8。

注：编码时图像宽度和图像高度处理为MiniSize的整数倍

——log2_lcu_size_minus2的取值范围应为3～5。

——log2_min_cu_size_minus2的值应为0。

——stable_patch_flag的值应为‘1’。

——ref_colocated_patch_flag的值应为‘0’。

——uniform_patch_flag的值应为‘1’。

——属于同一图像的片应按patch_index的值从小到大依次出现。

——如果编码块的任一边长大于 64，该编码块只应选择跳过模式或帧间预测模式。如果选择帧间预测模式，ctp_zero_flag的值应为‘1’。

——如果当前编码单元的亮度样本的数量小于 64，当前编码单元不应使用跳过模式和直接模式，且当前编码单元的预测单元的预测参考模式不应为‘PRED_List01’。

——MaxSplitTimes的值应为6。

——MaxPartRatio的值应为8。

——log2_max_bt_size_minus2的值应为4或5。

——如果I图像的最大编码单元的宽度和高度均等于128，则该最大编码单元应使用四叉树划分且qt_split_flag的值应为‘1’。

——如果编码树节点的宽度等于128且高度等于64且该编码树节点被划分，则该编码树节点应使用垂直二叉树划分且bet_split_dir_flag的值应为‘1’。

——如果编码树节点的宽度等于64且高度等于128且该编码树节点被划分，则该编码树节点应使用水平二叉树划分且bet_split_dir_flag的值应为‘0’。

——如果当前序列的library_stream_flag为‘1’，则该序列的所有图像均应为I图像。

——如果位流中存在语法元素duplicate_sequence_header_flag，则该语法元素的值应为‘1’。

——解码图像缓冲区中最多只应存在1幅知识图像。

——NumOfUpdatedLibrary的值应为1。

——MinLibraryUpdatePeriod的值应为1。

——num_of_nn_filter_minus1的取值应为0～3。

——MAX_TEMPORAL_ID的值应为7。

——应符合B.3规定的级限制。

——支持的级应包括：2.0.15、2.0.30、2.0.60、4.0.30、4.0.60、6.0.30、6.2.30、6.4.30、6.6.30、6.0.60、6.2.60、6.4.60、6.6.60、6.0.120、6.2.120、6.4.120、6.6.120、8.0.30、8.2.30、8.4.30、8.6.30、8.0.60、8.2.60、8.4.60、8.6.60、8.0.120、8.2.120、8.4.120、8.6.120、10.0.30、10.2.30、10.4.30、10.6.30、10.0.60、10.2.60、10.4.60、10.6.60、10.0.120、10.2.120、10.4.120和10.6.120。

B.3 级

级应符合表B.2的规定。

表B.2 级

level_id的值	级
0x00	禁止
0x10	2.0.15
0x12	2.0.30

表 B.2（续）

level_id的值	级
0x14	2.0.60
0x20	4.0.30
0x22	4.0.60
0x40	6.0.30
0x41	6.4.30
0x42	6.2.30
0x43	6.6.30
0x44	6.0.60
0x45	6.4.60
0x46	6.2.60
0x47	6.6.60
0x48	6.0.120
0x49	6.4.120
0x4A	6.2.120
0x4B	6.6.120
0x50	8.0.30
0x51	8.4.30
0x52	8.2.30
0x53	8.6.30
0x54	8.0.60
0x55	8.4.60
0x56	8.2.60
0x57	8.6.60
0x58	8.0.120
0x59	8.4.120
0x5A	8.2.120
0x5B	8.6.120
0x60	10.0.30
0x61	10.4.30
0x62	10.2.30
0x63	10.6.30
0x64	10.0.60
0x65	10.4.60
0x66	10.2.60
0x67	10.6.60
0x68	10.0.120
0x69	10.4.120
0x6A	10.2.120
0x6B	10.6.120
其他	保留

对于所有类，每个最大编码单元编码后最大二进制位数的限制应符合表B.3的规定。

表B.3　最大编码单元编码后最大二进制位数

图像格式	最大编码单元编码后最大二进制位数
4:2:0	$2^{(LcuSizeInBit-1)} + 2^{(LcuSizeInBit*2)}$*BitDepth*1.5

级2.0.15、2.0.30和2.0.60的参数限制应符合表B.4的规定。

表B.4　级 2.0.15、2.0.30 和 2.0.60 的参数限制

参数	级		
	2.0.15	2.0.30	2.0.60
每行最大样本数	352	352	352
每帧最大行数	288	288	288
每秒最大帧数	15	30	60
每帧最大片数	16	16	16
最大亮度样本速率（样本每秒）	1 520 640	3 041 280	6 082 560
最大位速率（位每秒）	1 500 000	2 000 000	2 500 000
每帧最大位数（位）	1 500 000	2 000 000	2 500 000
BBV缓冲区大小（位）	1 507 328	2 015 232	2 506 752
解码图像缓冲区大小（图像）	16	16	16

级4.0.30和4.0.60的参数限制应符合表B.5的规定。

表B.5　级 4.0.30 和 4.0.60 的参数限制

参数	级	
	4.0.30	4.0.60
每行最大样本数	720	720
每帧最大行数	576	576
每秒最大帧数	30	60
每帧最大片数	32	32
最大亮度样本速率（样本每秒）	12 441 600	24 883 200
最大位速率（位每秒）	6 000 000	10 000 000
每帧最大位数（位）	2 500 000	2 500 000
BBV缓冲区大小（位）	6 012 928	10 010 624
解码图像缓冲区大小（图像）	16	16

级6.0.30和6.2.30的参数限制应符合表B.6的规定。

表B.6　级 6.0.30 和 6.2.30 的参数限制

参数	级	
	6.0.30	6.2.30
每行最大样本数	2 048	2 048
每帧最大行数	1 152	1 152
每秒最大帧数	30	30
每帧最大片数	64	64
最大亮度样本速率（样本每秒）	66 846 720	66 846 720
最大位速率（位每秒）	12 000 000	30 000 000
每帧最大位数（位）	2 500 000	2 500 000
BBV缓冲区大小（位）	12 009 472	30 015 488
解码图像缓冲区大小（图像）	Min(14622720/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级6.4.30和6.6.30的参数限制应符合表B.7的规定。

表B.7　级 6.4.30 和 6.6.30 的参数限制

参数	级	
	6.4.30	6.6.30
每行最大样本数	2 048	2 048
每帧最大行数	1 152	1 152
每秒最大帧数	30	30
每帧最大片数	64	64
最大亮度样本速率（样本每秒）	66 846 720	66 846 720
最大位速率（位每秒）	12 000 000	30 000 000
每帧最大位数（位）	2 500 000	2 500 000
BBV缓冲区大小（位）	12 009 472	30 015 488
解码图像缓冲区大小（图像）	Min(16711680/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级6.0.60和6.2.60的参数限制应符合表B.8的规定。

表B.8　级 6.0.60 和 6.2.60 的参数限制

参数	级	
	6.0.60	6.2.60
每行最大样本数	2 048	2 048
每帧最大行数	1 152	1 152
每秒最大帧数	60	60
每帧最大片数	64	64
最大亮度样本速率（样本每秒）	133 693 440	133 693 440
最大位速率（位每秒）	20 000 000	50 000 000

表 B.8（续）

参数	级	
	6.0.60	6.2.60
每帧最大位数（位）	2 500 000	2 500 000
BBV缓冲区大小（位）	20 004 864	50 003 968
解码图像缓冲区大小（图像）	Min(14622720/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级6.4.60和6.6.60的参数限制应符合表B.9的规定。

表B.9　级 6.4.60 和 6.6.60 的参数限制

参数	级	
	6.4.60	6.6.60
每行最大样本数	2 048	2 048
每帧最大行数	1 152	1 152
每秒最大帧数	60	60
每帧最大片数	64	64
最大亮度样本速率（样本每秒）	133 693 440	133 693 440
最大位速率（位每秒）	20 000 000	50 000 000
每帧最大位数（位）	2 500 000	2 500 000
BBV缓冲区大小（位）	20 004 864	50 003 968
解码图像缓冲区大小（图像）	Min(16711680/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级6.0.120和6.2.120的参数限制应符合表B.10的规定。

表B.10　级 6.0.120 和 6.2.120 的参数限制

参数	级	
	6.0.120	6.2.120
每行最大样本数	2 048	2 048
每帧最大行数	1 152	1 152
每秒最大帧数	120	120
每帧最大片数	64	64
最大亮度样本速率（样本每秒）	267 386 880	267 386 880
最大位速率（位每秒）	25 000 000	100 000 000
每帧最大位数（位）	2 500 000	2 500 000
BBV缓冲区大小（位）	25 001 984	100 007 936
解码图像缓冲区大小（图像）	Min(14622720/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级6.4.120和6.6.120的参数限制应符合表B.11的规定。

表B.11　级 6.4.120 和 6.6.120 的参数限制

参数	级	
	6.4.120	6.6.120
每行最大样本数	2 048	2 048
每帧最大行数	1 152	1 152
每秒最大帧数	120	120
每帧最大片数	64	64
最大亮度样本速率（样本每秒）	267 386 880	267 386 880
最大位速率（位每秒）	25 000 000	100 000 000
每帧最大位数（位）	2 500 000	2 500 000
BBV缓冲区大小（位）	25 001 984	100 007 936
解码图像缓冲区大小（图像）	Min(16711680/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级8.0.30和8.2.30的参数限制应符合表B.12的规定。

表B.12　级 8.0.30 和 8.2.30 的参数限制

参数	级	
	8.0.30	8.2.30
每行最大样本数	4 096	4 096
每帧最大行数	2 304	2 304
每秒最大帧数	30	30
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	283 115 520	283 115 520
最大位速率（位每秒）	25 000 000	100 000 000
每帧最大位数（位）	10 000 000	10 000 000
BBV缓冲区大小（位）	25 001 984	100 007 936
解码图像缓冲区大小（图像）	Min(58490880/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级8.4.30和8.6.30的参数限制应符合表B.13的规定。

表B.13　级 8.4.30 和 8.6.30 的参数限制

参数	级	
	8.4.30	8.6.30
每行最大样本数	4 096	4 096
每帧最大行数	2 304	2 304
每秒最大帧数	30	30
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	283 115 520	283 115 520
最大位速率（位每秒）	25 000 000	100 000 000

表 B.13（续）

参数	级	
	8.4.30	8.6.30
每帧最大位数（位）	10 000 000	10 000 000
BBV缓冲区大小（位）	25 001 984	100 007 936
解码图像缓冲区大小（图像）	Min(66846720/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级8.0.60和8.2.60的参数限制应符合表B.14的规定。

表B.14　级 8.0.60 和 8.2.60 的参数限制

参数	级	
	8.0.60	8.2.60
每行最大样本数	4 096	4 096
每帧最大行数	2 304	2 304
每秒最大帧数	60	60
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	566 231 040	566 231 040
最大位速率（位每秒）	40 000 000	160 000 000
每帧最大位数（位）	10 000 000	10 000 000
BBV缓冲区大小（位）	40 009 728	160 006 144
解码图像缓冲区大小（图像）	Min(58490880/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级8.4.60和8.6.60的参数限制应符合表B.15的规定。

表B.15　级 8.4.60 和 8.6.60 的参数限制

参数	级	
	8.4.60	8.6.60
每行最大样本数	4 096	4 096
每帧最大行数	2 304	2 304
每秒最大帧数	60	60
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	566 231 040	566 231 040
最大位速率（位每秒）	40 000 000	160 000 000
每帧最大位数（位）	10 000 000	10 000 000
BBV缓冲区大小（位）	40 009 728	160 006 144
解码图像缓冲区大小（图像）	Min(66846720/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级8.0.120和8.2.120的参数限制应符合表B.16的规定。

表B.16　级 8.0.120 和 8.2.120 的参数限制

参数	级	
	8.0.120	8.2.120
每行最大样本数	4 096	4 096
每帧最大行数	2 304	2 304
每秒最大帧数	120	120
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	1 132 462 080	1 132 462 080
最大位速率（位每秒）	60 000 000	240 000 000
每帧最大位数（位）	10 000 000	10 000 000
BBV缓冲区大小（位）	60 014 592	240 009 216
解码图像缓冲区大小（图像）	Min(58490880/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级8.4.120和8.6.120的参数限制应符合表B.17的规定。

表B.17　级 8.4.120 和 8.6.120 的参数限制

参数	级	
	8.4.120	8.6.120
每行最大样本数	4 096	4 096
每帧最大行数	2 304	2 304
每秒最大帧数	120	120
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	1 132 462 080	1 132 462 080
最大位速率（位每秒）	60 000 000	240 000 000
每帧最大位数（位）	10 000 000	10 000 000
BBV缓冲区大小（位）	60 014 592	240 009 216
解码图像缓冲区大小（图像）	Min(66846720/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级10.0.30和10.2.30的参数限制应符合表B.18的规定。

表B.18　级 10.0.30 和 10.2.30 的参数限制

参数	级	
	10.0.30	10.2.30
每行最大样本数	8 192	8 192
每帧最大行数	4 608	4 608
每秒最大帧数	30	30
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	1 069 547 520	1 069 547 520
每片最大亮度样本速率（样本每秒）	267 386 880	267 386 880

表 B.18（续）

参数	级	
	10.0.30	10.2.30
最大位速率（位每秒）	60 000 000	240 000 000
每帧最大位数（位）	30 000 000	30 000 000
BBV缓冲区大小（位）	60 014 592	240 009 216
解码图像缓冲区大小（图像）	Min(226492416/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级10.4.30和10.6.30的参数限制应符合表B.19的规定。

表B.19　级 10.4.30 和 10.6.30 的参数限制

参数	级	
	10.4.30	10.6.30
每行最大样本数	8 192	8 192
每帧最大行数	4 608	4 608
每秒最大帧数	30	30
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	1 069 547 520	1 069 547 520
每片最大亮度样本速率（样本每秒）	267 386 880	267 386 880
最大位速率（位每秒）	60 000 000	240 000 000
每帧最大位数（位）	30 000 000	30 000 000
BBV缓冲区大小（位）	60 014 592	240 009 216
解码图像缓冲区大小（图像）	Min(301989888/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级10.0.60和10.2.60的参数限制应符合表B.20的规定。

表B.20　级 10.0.60 和 10.2.60 的参数限制

参数	级	
	10.0.60	10.2.60
每行最大样本数	8 192	8 192
每帧最大行数	4 608	4 608
每秒最大帧数	60	60
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	2 139 095 040	2 139 095 040
每片最大亮度样本速率（样本每秒）	534 773 760	534 773 760
最大位速率（位每秒）	120 000 000	480 000 000
每帧最大位数（位）	30 000 000	30 000 000
BBV缓冲区大小（位）	120 012 800	480 002 048
解码图像缓冲区大小（图像）	Min(226492416/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级10.4.60和10.6.60的参数限制应符合表B.21的规定。

表B.21　级 10.4.60 和 10.6.60 的参数限制

参数	级	
	10.4.60	10.6.60
每行最大样本数	8 192	8 192
每帧最大行数	4 608	4 608
每秒最大帧数	60	60
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	2 139 095 040	2 139 095 040
每片最大亮度样本速率（样本每秒）	534 773 760	534 773 760
最大位速率（位每秒）	120 000 000	480 000 000
每帧最大位数（位）	30 000 000	30 000 000
BBV缓冲区大小（位）	120 012 800	480 002 048
解码图像缓冲区大小（图像）	Min(301989888/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级10.0.120和10.2.120的参数限制应符合表B.22的规定。

表B.22　级 10.0.120 和 10.2.120 的参数限制

参数	级	
	10.0.120	10.2.120
每行最大样本数	8 192	8 192
每帧最大行数	4 608	4 608
每秒最大帧数	120	120
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	4 278 190 080	4 278 190 080
每片最大亮度样本速率（样本每秒）	1 069 547 520	1 069 547 520
最大位速率（位每秒）	240 000 000	800 000 000
每帧最大位数（位）	30 000 000	30 000 000
BBV缓冲区大小（位）	240 009 216	800 014 336
解码图像缓冲区大小（图像）	Min(226492416/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

级10.4.120和10.6.120的参数限制应符合表B.23的规定。

表B.23　级 10.4.120 和 10.6.120 的参数限制

参数	级	
	10.4.120	10.6.120
每行最大样本数	8 192	8 192
每帧最大行数	4 608	4 608

表 B.23（续）

参数	级	
	10.4.120	10.6.120
每秒最大帧数	120	120
每帧最大片数	128	128
最大亮度样本速率（样本每秒）	4 278 190 080	4 278 190 080
每片最大亮度样本速率（样本每秒）	1 069 547 520	1 069 547 520
最大位速率（位每秒）	240 000 000	800 000 000
每帧最大位数（位）	30 000 000	30 000 000
BBV缓冲区大小（位）	240 009 216	800 014 336
解码图像缓冲区大小（图像）	Min(301989888/(PictureWidthInMinBu * MiniSize * PictureHeightInMinBu * MiniSize),16)	

某一级下 BBV 缓冲区大小是 16384 位的倍数。

与表 B.4 到表 B.23 有关的语法元素包括：horizontal_size、vertical_size、frame_rate_code、bbv_buffer_size。

一幅图像中二元符号的最大个数应小于或等于：

当前图像编码后位流的二进制位数*1.8+((horizontal_size * vertical_size * (BitDepth + (BitDepth>>1)))>>5)

式中，一幅图像的二元符号包括 decode_decision、decode_aec_stuffing_bit、decode_bypass 过程解码的二元符号。

附 录 C
（规范性）
位流参考缓冲区管理

C.1 通则

本附录定义了位流参考缓冲区管理（BBV）。

BBV有一个输入缓冲区，称为BBV缓冲区。编码数据按照C.3.1定义的方式进入BBV缓冲区，按照C.3.2定义的方式移出BBV缓冲区。符合本文件的位流不应导致BBV缓冲区上溢。如果low_delay的值为‘0’，编码数据应按C.3.2.2定义的方式移出BBV缓冲区；如果low_delay的值为‘1’，编码数据应按C.3.2.3定义的方式移出BBV缓冲区。符合本文件的位流在每幅图像的解码时刻均不应导致BBV缓冲区下溢。

本附录中所有的运算都是实数运算，不存在舍入误差，例如BBV缓冲区中的位数不必是整数值。

C.2 约定

C.2.1 约定一

BBV和视频编码器的时钟频率和帧率相同，并且同步操作。

C.2.2 约定二

BBV缓冲区的大小为BBS，单位为二进制位。

C.2.3 约定三

编码数据输入BBV缓冲区的最大速率R_{max}（单位为位每秒）按公式（C.1）计算。

$$R_{max} = BitRate \times 400 \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \text{(C.1)}$$

式中：

R_{max} ——编码数据输入BBV缓冲区的最大速率；

BitRate——以400bit/s为单位计算视频位流的比特率。

C.3 基本操作

C.3.1 数据输入

C.3.1.1 通则

本附录定义了两种方法来计算编码数据进入BBV缓冲区的速率。这两种方法不应同时使用。

C.3.1.2 方法一

C.3.1.2.1 操作过程

在BBV中，第n幅图像的编码数据f(n)包括以下数据（如果存在）。

——序列头、跟在序列头之后的扩展和用户数据、视频编辑码。这些数据定义为 b(n)，b(n)与本幅图像起始码之间不应有其他图像的数据。

——本幅图像的所有编码数据。

——跟在本幅图像头后的图像显示扩展。

——跟在本幅图像后的填充数据、视频序列结束码。

如果BbvDelay的值不等于BbvDelayMax，第n幅图像进入BBV缓冲区的速率$R(n)$按公式（C.2）计算。

$$R(n) = d^{*}_{n} \div (\tau(n) - \tau(n+1) + t(n+1) - t(n)) \quad \cdots\cdots\cdots\cdots\cdots\cdots (C.2)$$

式中：

$R(n)$ ——第n幅图像进入BBV缓冲区的速率；

d^{*}_{n} ——从第n幅图像的起始码后第1个位到第n+1幅图像的起始码后第1个位之间所有的位数；

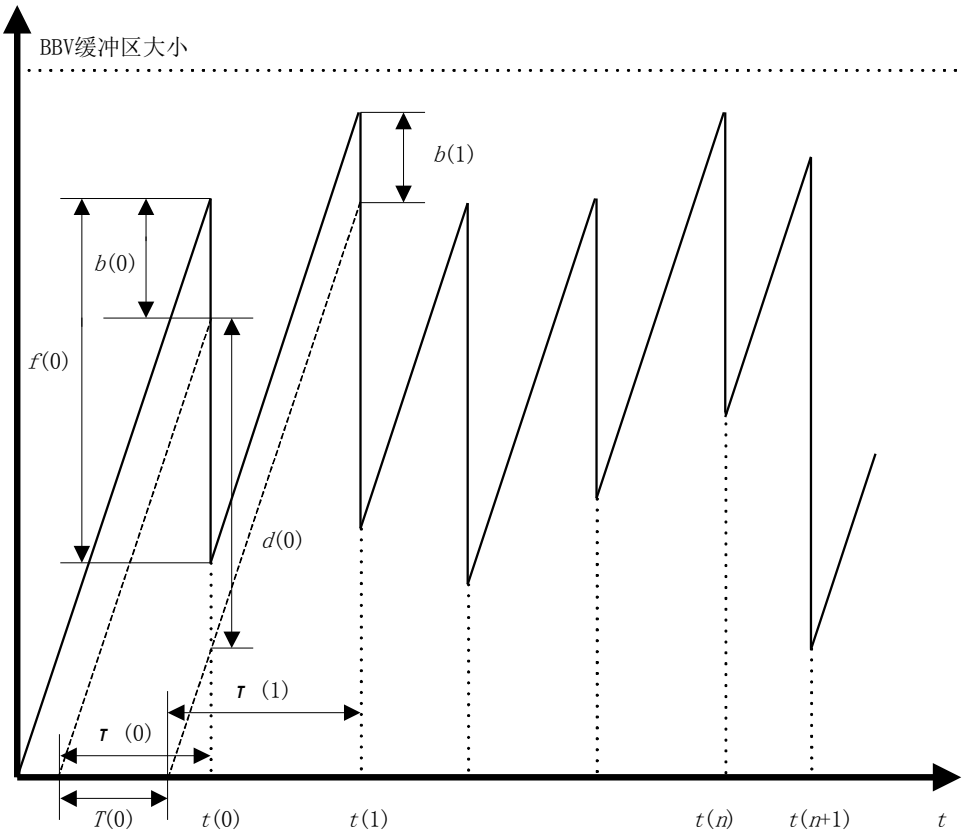
$\tau(n)$ ——第n幅图像的BbvDelay的值，单位为秒（s）；

$t(n)$ ——第n幅图像的编码数据从BBV缓冲区移出的时间，单位为秒（s）；

$t(n+1)-t(n)$——第n+1幅和第n幅图像的解码时间间隔，单位为秒（s）。

在视频序列开始和结束时，可能缺少参数来无歧义地确定这个时间间隔，此时可采用下面的方法来处理。

图C.1为第n幅图像的编码数据按方法一进入BBV缓冲区的示意图。



图C.1 BBV 缓冲区占用情况一

C.3.1.2.2 视频序列开始时的歧义性

对视频序列进行随机访问时，序列头后开始的若干幅图像缺少对应的输出图像。在这种情况下，如果位流是系统流的一部分，可从系统流来确定解码时间间隔。

如果还不能无歧义地确定解码时间间隔，就无法确定$R(n)$。此时在一段有限的时间内（这个时间总是小于BbvDelay的最大值）BBV不能准确地确定充满度的变化轨迹，从而无法在整个位流中严格地验证BBV缓冲区。编码器总是知道在每个重复序列头后$t(n+1)-t(n)$的值，因此也知道如何生成不违反BBV限制的位流。

C.3.1.2.3 视频序列结束时的歧义性

如果一幅图像的编码数据后第1个起始码是视频序列结束码，此时无法确定该幅图像编码数据的位数。在这种情况下，应存在一个输入速率，这个速率不应导致BBV缓冲区上溢；在low_delay的值为‘1’时，这个速率也不应导致缓冲区下溢。该速率应小于在视频序列头中定义的最大速率。

视频序列的第1幅图像的起始码前的所有数据和这个起始码输入BBV缓冲区后，每一幅图像的编码数据应在由位流中的BbvDelay规定的时间内输入BBV缓冲区，并在这个时间开始解码处理。输入速率由公式（C.2）确定。

所有位流的$R(n)$均不应大于R_{max}。

在CBR情况下，视频序列中的R(n)在BbvDelay允许的精度范围内是一个常数。

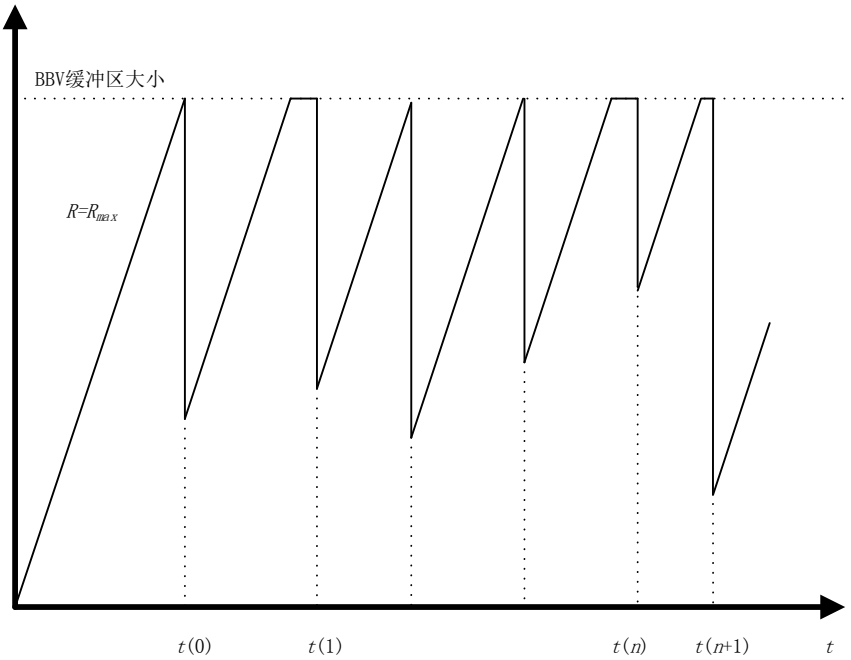
C.3.1.3 方法二

如果BbvDelay的值为BbvDelayMax，编码数据按以下方式输入BBV缓冲区：

——如果BBV缓冲区没有充满，数据以速率R_{max}输入缓冲区；

——如果BBV缓冲区充满，则数据不能进入该缓冲区直到缓冲区中的部分数据被移出。

视频序列的第1幅图像的起始码前的所有数据和这个起始码输入BBV缓冲区后，数据继续输入BBV缓冲区直到缓冲区充满，并在这个时间开始解码处理，见图C.2。



图C.2 BBV 缓冲区占用情况二

C.3.2 数据移出

C.3.2.1 概述

位流中第n幅图像的编码数据移出BBV缓冲区的时间为第n幅图像的解码时刻。视频序列的第1幅图像的解码时刻t(0)等于序列第1幅图像的图像起始码进入BBV缓冲区的时刻加上τ(0)。第n幅图像的解码时刻t(n)和编码数据的移出BBV缓冲区的方式根据C.3.2.2和C.3.2.3分别确定。

C.3.2.2 非低延迟

本条定义low_delay的值为‘0’时第n幅图像的解码时刻t(n)和编码数据的移出BBV缓冲区的方式。

对于不包含知识位流的解码位流，第n幅图像的解码时刻t(n)是前一幅图像的解码时刻t(n-1)加上一个检测时间间隔delta(n)。

对于包含知识位流和主位流的解码位流，如果第n幅图像是知识图像，则该图像的参考解码时刻t'(n)等于t'(n－1)，其中t'(0)＝t(0)；该图像的解码时刻t(n)是前一幅图像的解码时刻t(n-1)加上一个检测时间间隔delta(n)。

对于包含知识位流和主位流的解码位流，如果第n幅图像不是知识图像，则该图像的参考解码时刻t'(n)等于t'(n－1)加delta'(n)，其中t'(0)＝t(0)；该图像的解码时刻t(n)如下：

——如果第 n-1 幅图像不是知识图像且 t(n-1) ＋ delta(n) 小于参考解码时刻 t'(n)-delta'(n)-delta'(n-1)，则 t(n)是前一幅图像的解码时刻 t(n-1)加上两倍的检测时间间隔 delta(n)；

——如果第 n-1 幅图像是知识图像且 t(n-1) ＋ delta(n-1) 小于参考解码时刻 t'(n)-delta'(n-1)-delta'(n-2)，则 t(n)是前一幅图像的解码时刻 t(n-1)加上两倍的检测时间间隔 delta(n-1)；

——否则，则 t(n)是前一幅图像的解码时刻 t(n-1)加上一个检测时间间隔 delta(n)。

检测时间间隔delta(n)和假设检测时间间隔delta'(n)在C.4定义。

在每幅图像的解码时刻t(n)，BBV缓冲区充满度应小于BBS。并且 BBV缓冲区充满度B(n)应大于等于f(n)，否则发生下溢，符合本文件的码流不应发生下溢。

在每幅图像的解码时刻t(n) 瞬时移出该幅图像的编码数据并解码。

C.3.2.3 低延迟

本条定义low_delay的值为‘1’时第n幅图像的解码时刻t(n)和编码数据的移出BBV缓冲区的方式。

对于不包含知识位流的解码位流，第n幅图像的解码时刻t(n)是前一幅图像的解码时刻t(n-1)加上一个检测时间间隔delta(n)，再加上BbvCheckTimes(n)乘以图像周期的积。

对于包含知识位流和主位流的解码位流，如果第n幅图像是知识图像，则该图像的参考解码时刻t'(n)＝t'(n－1)，其中t'(0)＝t(0)；该图像的解码时刻t(n)是前一幅图像的解码时刻t(n-1)加上一个检测时间间隔delta(n)。

对于包含知识位流和主位流的解码位流，如果第n幅图像不是知识图像，则该图像的参考解码时刻t'(n)＝t'(n－1)+delta'(n)+(BbvCheckTimes(n)*图像周期)，其中t'(0)＝t(0)；该图像的解码时刻t(n)如下：

——如果第 n-1 幅图像不是知识图像且 t(n-1)＋delta(n)+(BbvCheckTimes(n)*图像周期)小于参考解码时刻 t'(n)-delta'(n)-(BbvCheckTimes(n)* 图像周期)-delta'(n-1) －(BbvCheckTimes(n-1)*图像周期)，则 t(n)是前一幅图像的解码时刻 t(n-1)加上两倍的检测时间间隔 delta(n)，再加上 BbvCheckTimes 乘以图像周期的积；

——如果第 n-1 幅图像是知识图像且 t(n-1)＋delta(n-1)+(BbvCheckTimes(n)*图像周期)小于参考解码时刻 t'(n)-delta'(n-1)-(BbvCheckTimes(n)* 图像周期)-delta'(n-2)-(BbvCheckTimes(n-2)*图像周期)，则 t(n)是前一幅图像的解码时刻 t(n-1)加上两倍的检测时间间隔 delta(n-1)，再加上 BbvCheckTimes(n)乘以图像周期的积；

——否则，t(n)是前一幅图像的解码时刻 t(n-1)加上一个检测时间间隔 delta(n)，再加上BbvCheckTimes(n)乘以图像周期的积。

检测时间间隔delta(n)和假设检测时间间隔delta'(n)在C.4定义。如果field_coded_sequence的值为‘1’，则图像周期为帧率倒数的0.5倍，如果field_coded_sequence的值为‘0’，则图像周期为帧率的倒数。

在每幅图像的解码时刻$t(n)$,BBV缓冲区充满度应小于BBS,该图像的所有编码数据均应在缓冲区中，然后该图像被瞬时移出。

如果BbvCheckTimes(n)大于0，当前解码图像定义为大图像。一个视频序列的最后一幅图像不应是大图像。

C.4 缓冲区检测时间间隔

记帧率的倒数为T，记帧率加1的和的倒数为D。如果解码位流中不包含知识位流，则P等于T；如果解码位流中包含知识位流和主位流，则P等于D。

BBV缓冲区检测时间间隔delta(n)和假设检测时间间隔delta'(n)由解码完第n-1幅图像后输出的非知识图像（见9.2.4）确定。

当progressive_sequence的值为‘1’且field_coded_sequence的值为‘0’时：

——如果该输出图像的repeat_first_field的值为‘0’，则delta(n)等于P，delta'(n)等于T；

——如果该输出图像的repeat_first_field的值为‘1’且top_field_first的值为‘0’,则delta(n)等于2P，delta'(n)等于2T；

——如果该输出图像的repeat_first_field的值为‘1’且top_field_first的值为‘1’,则delta(n)等于3P，delta'(n)等于3T。

当progressive_sequence的值为‘0’且field_coded_sequence的值为‘0’时：

——如果该输出图像的repeat_first_field的值为‘0’，则delta(n)等于P，delta'(n)等于T；

——如果该输出图像的repeat_first_field的值为‘1’，则delta(n)等于1.5P，delta'(n)等于1.5T。

当progressive_sequence的值为‘0’且field_coded_sequence的值为‘1’时,delta(n)等于0.5P，delta'(n)等于0.5T。

附 录 D
（规范性）
默认加权量化矩阵

本附录定义序列头位流中加载的4×4和8×8变换块的默认加权量化矩阵$WeightQuantMatrix_{4x4}$和$WeightQuantMatrix_{8x8}$。

$$WeightQuantMatrix_{4x4}=\begin{bmatrix}64 & 64 & 64 & 68\\ 64 & 64 & 68 & 72\\ 64 & 68 & 76 & 80\\ 72 & 76 & 84 & 96\end{bmatrix}$$

$$WeightQuantMatrix_{8x8}=\begin{bmatrix}64 & 64 & 64 & 64 & 68 & 68 & 72 & 76\\ 64 & 64 & 64 & 68 & 72 & 76 & 84 & 92\\ 64 & 64 & 68 & 72 & 76 & 80 & 88 & 100\\ 64 & 68 & 72 & 80 & 84 & 92 & 100 & 112\\ 68 & 72 & 80 & 84 & 92 & 104 & 112 & 128\\ 76 & 80 & 84 & 92 & 104 & 116 & 132 & 152\\ 96 & 100 & 104 & 116 & 124 & 140 & 164 & 188\\ 104 & 108 & 116 & 128 & 152 & 172 & 192 & 216\end{bmatrix}$$

附 录 E
（规范性）
扫描表生成方法

扫描表InvScanCoeffInBlk[7][7][pos][2]通过以下过程生成：

```
for (yIndex = 0; yIndex < 7; yIndex++) {
    ySize = 1 << (yIndex + 1)
    for (xIndex = 0; xIndex < 7; xIndex++) {
        xSize = 1 << (xIndex + 1)
        pos = 0
        numLine = xSize + ySize - 1
        InvScanCoeffInBlk[yIndex][xIndex][0][0] = 0
        InvScanCoeffInBlk[yIndex][xIndex][0][1] = 0
        pos++
        for (l = 1; l < numLine; l++) {
            if (l % 2) {
                x = Min(l, xSize - 1)
                y = Max(0, l - (xSize - 1))
                while (x ≥ 0 && y < ySize) {
                    InvScanCoeffInBlk[yIndex][xIndex][pos][0] = x
                    InvScanCoeffInBlk[yIndex][xIndex][pos][1] = y
                    pos++
                    x--
                    y++
                }
            }
            else {
                y = Min(l, ySize - 1)
                x = Max(0, l - (ySize - 1))
                while (y ≥ 0 && x < xSize) {
                    InvScanCoeffInBlk[yIndex][xIndex][pos][0] = x
                    InvScanCoeffInBlk[yIndex][xIndex][pos][1] = y
                    pos++
                    x++
                    y--
                }
            }
        }
    }
}
```

基于扫描区域的系数扫描方法的扫描表 ScanOrder 的初始化方法如下，其中输入为扫描区域宽度 xSize，扫描区域高度 ySize，系数矩阵宽度 width：

```
initScanOrder(scanOrder, xSize, ySize, width) {
    pos = 0
    numLine = xSize + ySize - 1
    /* starting point */
    scan[pos] = 0
    pos++

    /* loop */
    for (l = 1; l < numLine; l++) {
        if (l % 2) { /* decreasing loop */
            x = Min(l, xSize - 1)
            y = Max(0, l - (xSize - 1))
            while (x ≥ 0 && y < ySize) {
                scan[pos] = y * width + x
                pos++
                x--
                y++
            }
        }
        else { /* increasing loop */
            y = Min(l, ySize - 1)
            x = Max(0, l - (ySize - 1))
            while (y ≥ 0 && x < xSize) {
                scan[pos] = y * width + x
                pos++
                x++
                y--
            }
        }
    }
}
```

扫描表TravScan[4][4][1024][2]通过以下过程生成：

```
for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        sPos = 0;
        for (y = 0; y < (4 << j); y+=2) {
            for (x = 0; x < (4 << i); x++) {
                TravScan[i][j][sPos + x][0] = x
                TravScan[i][j][sPos + x][1] = y
```

```
                TravScan[i][j][sPos + 2 * (4 << i) - 1 - x][0] = x
                TravScan[i][j][sPos + 2 * (4 << i) - 1 - x][1] = y + 1
            }
            sPos += (2 * (4 << i))
        }
    }
}
```

附 录 F
（资料性）
高级熵编码的解码参考描述方法

本附录描述了高级熵编码的解码参考描述方法，包括decode_aec_stuffing_bit和decode_bypass过程的参考描述方法。

decode_aec_stuffing_bit过程的输入是valueD、bFlag、rS1、rT1、valueS和valueT。decode_aec_stuffing_bit过程的输出是二元符号值binVal。decode_aec_stuffing_bit过程如下：

```
decode_aec_stuffing_bit( ) {
    predMps = 0
    if (valueD || (bFlag == 1 && rS1 == boundS) ) {
        rS1 = 0
        valueS = 0
        while ( valueT < 0x100 && valueS < boundS ) {
            valueS++
            valueT = (valueT << 1) | read_bits(1)
        }
        if ( valueT < 0x100 )
            bFlag = 1
        else
            bFlag = 0
        valueT = valueT & 0xFF
    }
    if ( rT1 ) {
        rS2 = rS1
        rT2 = rT1 - 1
    }
    else {
        rS2 = rS1 + 1
        rT2 = 255
    }
    if ( rS2 > valueS || (rS2 == valueS && valueT ≥ rT2) && bFlag == 0 ) {
        binVal = ! predMps
        if ( rS2 == valueS )
            valueT = valueT - rT2
        else
            valueT = 256 + ((valueT << 1 ) | read_bits(1)) - rT2
        valueT = (valueT << 8 ) | read_bits(8)
        rT1 = 0
        valueD = 1
```

```
    }
    else {
        binVal = predMps
        rS1 = rS2
        rT1 = rT2
        valueD = 0
    }
    return (binVal)
}
```

decode_bypass过程的输入是valueD、bFlag、rS1、rT1、valueS和valueT。decode_bypass过程的输出是二元符号值binVal。decode_bypass过程如下：

```
decode_bypass( ) {
    predMps = 0
    if (valueD || (bFlag == 1 && rS1 == boundS) ) {
        rS1 = 0
        valueD = 1
        valueT = (valueT << 1 ) | read_bits(1)
        if ( valueT ≥ (256 + rT1) ) {
            binVal = ! predMps
            valueT -=(256 + rT1)
        }
        else {
            binVal = predMps
        }
    }
    else {
        rS2 = rS1 + 1
        if ( rS2 > valueS || (rS2 == valueS && valueT ≥ rT1) && bFlag == 0 ) {
            binVal = ! predMps
            if ( rS2 == valueS )
                valueT = valueT  -  rT1
            else
                valueT = 256 + ((valueT << 1 ) | read_bits(1))  -  rT1
            rS1 = 0
            valueD = 1
        }
        else {
            binVal = predMps
            rS1= rS2
            valueD = 0
        }
```

```
    }
    return (binVal)
}
```

附 录 G
（规范性）
反变换矩阵

G.1 DCT2 型反变换矩阵

G.1.1 4×4 DCT2反变换矩阵

4×4 DCT2反变换矩阵定义如下：

$DCT2_4$ = {
{ 32 32 32 32 }
{ 42 17 -17 -42 }
{ 32 -32 -32 32 }
{ 17 -42 42 -17 }}

G.1.2 8×8 DCT2反变换矩阵

8×8 DCT2反变换矩阵定义如下：

$DCT2_8$ = {
{ 32 32 32 32 32 32 32 32 }
{ 44 38 25 9 -9 -25 -38 -44 }
{ 42 17 -17 -42 -42 -17 17 42 }
{ 38 -9 -44 -25 25 44 9 -38 }
{ 32 -32 -32 32 32 -32 -32 32 }
{ 25 -44 9 38 -38 -9 44 -25 }
{ 17 -42 42 -17 -17 42 -42 17 }
{ 9 -25 38 -44 44 -38 25 -9 }}

G.1.3 16×16 DCT2反变换矩阵

16×16 DCT2反变换矩阵定义如下：

$DCT2_{16}$ = {
{ 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 }
{ 45 43 40 35 29 21 13 4 -4 -13 -21 -29 -35 -40 -43 -45 }
{ 44 38 25 9 -9 -25 -38 -44 -44 -38 -25 -9 9 25 38 44 }
{ 43 29 4 -21 -40 -45 -35 -13 13 35 45 40 21 -4 -29 -43 }
{ 42 17 -17 -42 -42 -17 17 42 42 17 -17 -42 -42 -17 17 42 }
{ 40 4 -35 -43 -13 29 45 21 -21 -45 -29 13 43 35 -4 -40 }
{ 38 -9 -44 -25 25 44 9 -38 -38 9 44 25 -25 -44 -9 38 }
{ 35 -21 -43 4 45 13 -40 -29 29 40 -13 -45 -4 43 21 -35 }
{ 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 }
{ 29 -40 -13 45 -4 -43 21 35 -35 -21 43 4 -45 13 40 -29 }
{ 25 -44 9 38 -38 -9 44 -25 -25 44 -9 -38 38 9 -44 25 }
{ 21 -45 29 13 -43 35 4 -40 40 -4 -35 43 -13 -29 45 -21 }

{ 17 -42 42 -17 -17 42 -42 17 17 -42 42 -17 -17 42 -42 17 }
{ 13 -35 45 -40 21   4 -29 43 -43 29 -4 -21 40 -45 35 -13 }
{ 9 -25 38 -44 44 -38 25 -9 -9 25 -38 44 -44 38 -25   9 }
{ 4 -13 21 -29 35 -40 43 -45 45 -43 40 -35 29 -21 13 -4 }}

G.1.4 32×32 DCT2反变换矩阵

32×32 DCT2反变换矩阵定义如下：

$DCT2_{32}$ = {
{ T_{11} T_{12} }
{ T_{21} T_{22} }}
其中，T_{11} = {
{ 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 }
{ 45 45 44 43 41 39 36 34 30 27 23 19 15 11   7   2 }
{ 45 43 40 35 29 21 13   4 -4 -13 -21 -29 -35 -40 -43 -45 }
{ 45 41 34 23 11 -2 -15 -27 -36 -43 -45 -44 -39 -30 -19 -7 }
{ 44 38 25   9 -9 -25 -38 -44 -44 -38 -25 -9   9 25 38 44 }
{ 44 34 15 -7 -27 -41 -45 -39 -23 -2 19 36 45 43 30 11 }
{ 43 29   4 -21 -40 -45 -35 -13 13 35 45 40 21 -4 -29 -43 }
{ 43 23 -7 -34 -45 -36 -11 19 41 44 27 -2 -30 -45 -39 -15 }
{ 42 17 -17 -42 -42 -17 17 42 42 17 -17 -42 -42 -17 17 42 }
{ 41 11 -27 -45 -30   7 39 43 15 -23 -45 -34   2 36 44 19 }
{ 40   4 -35 -43 -13 29 45 21 -21 -45 -29 13 43 35 -4 -40 }
{ 39 -2 -41 -36   7 43 34 -11 -44 -30 15 45 27 -19 -45 -23 }
{ 38 -9 -44 -25 25 44   9 -38 -38   9 44 25 -25 -44 -9 38 }
{ 36 -15 -45 -11 39 34 -19 -45 -7 41 30 -23 -44 -2 43 27 }
{ 35 -21 -43   4 45 13 -40 -29 29 40 -13 -45 -4 43 21 -35 }
{ 34 -27 -39 19 43 -11 -45   2 45   7 -44 -15 41 23 -36 -30 }}
T_{12} = {
{ 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 }
{ -2 -7 -11 -15 -19 -23 -27 -30 -34 -36 -39 -41 -43 -44 -45 -45 }
{-45 -43 -40 -35 -29 -21 -13 -4   4 13 21 29 35 40 43 45 }
{ 7 19 30 39 44 45 43 36 27 15   2 -11 -23 -34 -41 -45 }
{ 44 38 25   9 -9 -25 -38 -44 -44 -38 -25 -9   9 25 38 44 }
{-11 -30 -43 -45 -36 -19   2 23 39 45 41 27   7 -15 -34 -44 }
{-43 -29 -4 21 40 45 35 13 -13 -35 -45 -40 -21   4 29 43 }
{ 15 39 45 30   2 -27 -44 -41 -19 11 36 45 34   7 -23 -43 }
{ 42 17 -17 -42 -42 -17 17 42 42 17 -17 -42 -42 -17 17 42 }
{-19 -44 -36 -2 34 45 23 -15 -43 -39 -7 30 45 27 -11 -41 }
{-40 -4 35 43 13 -29 -45 -21 21 45 29 -13 -43 -35   4 40 }
{ 23 45 19 -27 -45 -15 30 44 11 -34 -43 -7 36 41   2 -39 }
{ 38 -9 -44 -25 25 44   9 -38 -38   9 44 25 -25 -44 -9 38 }
{-27 -43   2 44 23 -30 -41   7 45 19 -34 -39 11 45 15 -36 }
{-35 21 43 -4 -45 -13 40 29 -29 -40 13 45   4 -43 -21 35 }

{ 30 36 -23 -41 15 44 -7 -45 -2 45 11 -43 -19 39 27 -34 }}

T_{21} = {

{ 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 }

{ 30 -36 -23 41 15 -44 -7 45 -2 -45 11 43 -19 -39 27 34 }

{ 29 -40 -13 45 -4 -43 21 35 -35 -21 43 4 -45 13 40 -29 }

{ 27 -43 -2 44 -23 -30 41 7 -45 19 34 -39 -11 45 -15 -36 }

{ 25 -44 9 38 -38 -9 44 -25 -25 44 -9 -38 38 9 -44 25 }

{ 23 -45 19 27 -45 15 30 -44 11 34 -43 7 36 -41 2 39 }

{ 21 -45 29 13 -43 35 4 -40 40 -4 -35 43 -13 -29 45 -21 }

{ 19 -44 36 -2 -34 45 -23 -15 43 -39 7 30 -45 27 11 -41 }

{ 17 -42 42 -17 -17 42 -42 17 17 -42 42 -17 -17 42 -42 17 }

{ 15 -39 45 -30 2 27 -44 41 -19 -11 36 -45 34 -7 -23 43 }

{ 13 -35 45 -40 21 4 -29 43 -43 29 -4 -21 40 -45 35 -13 }

{ 11 -30 43 -45 36 -19 -2 23 -39 45 -41 27 -7 -15 34 -44 }

{ 9 -25 38 -44 44 -38 25 -9 -9 25 -38 44 -44 38 -25 9 }

{ 7 -19 30 -39 44 -45 43 -36 27 -15 2 11 -23 34 -41 45 }

{ 4 -13 21 -29 35 -40 43 -45 45 -43 40 -35 29 -21 13 -4 }

{ 2 -7 11 -15 19 -23 27 -30 34 -36 39 -41 43 -44 45 -45 }}

T_{22} = {

{ 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 }

{-34 -27 39 19 -43 -11 45 2 -45 7 44 -15 -41 23 36 -30 }

{-29 40 13 -45 4 43 -21 -35 35 21 -43 -4 45 -13 -40 29 }

{ 36 15 -45 11 39 -34 -19 45 -7 -41 30 23 -44 2 43 -27 }

{ 25 -44 9 38 -38 -9 44 -25 -25 44 -9 -38 38 9 -44 25 }

{-39 -2 41 -36 -7 43 -34 -11 44 -30 -15 45 -27 -19 45 -23 }

{-21 45 -29 -13 43 -35 -4 40 -40 4 35 -43 13 29 -45 21 }

{ 41 -11 -27 45 -30 -7 39 -43 15 23 -45 34 2 -36 44 -19 }

{ 17 -42 42 -17 -17 42 -42 17 17 -42 42 -17 -17 42 -42 17 }

{-43 23 7 -34 45 -36 11 19 -41 44 -27 -2 30 -45 39 -15 }

{-13 35 -45 40 -21 -4 29 -43 43 -29 4 21 -40 45 -35 13 }

{ 44 -34 15 7 -27 41 -45 39 -23 2 19 -36 45 -43 30 -11 }

{ 9 -25 38 -44 44 -38 25 -9 -9 25 -38 44 -44 38 -25 9 }

{-45 41 -34 23 -11 -2 15 -27 36 -43 45 -44 39 -30 19 -7 }

{ -4 13 -21 29 -35 40 -43 45 -45 43 -40 35 -29 21 -13 4 }

{ 45 -45 44 -43 41 -39 36 -34 30 -27 23 -19 15 -11 7 -2 }}

G.1.5 64×64 DCT2反变换矩阵

64×64 DCT2反变换矩阵定义如下：

$DCT2_{64}$ = {

{ T_{11} T_{12} T_{13} T_{14} }

{ T_{21} T_{22} T_{23} T_{24} }

{ T_{31} T_{32} T_{33} T_{34} }

{ T_{41} T_{42} T_{43} T_{44} }}

其中，T_{11} = {

{ 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 }
{ 45 45 45 45 44 44 43 42 41 40 39 38 37 36 34 33 }
{ 45 45 44 43 41 39 36 34 30 27 23 19 15 11 7 2 }
{ 45 44 42 39 36 31 26 20 14 8 1 -6 -12 -18 -24 -30 }
{ 45 43 40 35 29 21 13 4 -4 -13 -21 -29 -35 -40 -43 -45 }
{ 45 42 37 30 20 10 -1 -12 -22 -31 -38 -43 -45 -45 -41 -36 }
{ 45 41 34 23 11 -2 -15 -27 -36 -43 -45 -44 -39 -30 -19 -7 }
{ 45 39 30 16 1 -14 -28 -38 -44 -45 -40 -31 -18 -3 12 26 }
{ 44 38 25 9 -9 -25 -38 -44 -44 -38 -25 -9 9 25 38 44 }
{ 44 36 20 1 -18 -34 -44 -45 -37 -22 -3 16 33 43 45 38 }
{ 44 34 15 -7 -27 -41 -45 -39 -23 -2 19 36 45 43 30 11 }
{ 44 31 10 -14 -34 -45 -42 -28 -6 18 37 45 40 24 1 -22 }
{ 43 29 4 -21 -40 -45 -35 -13 13 35 45 40 21 -4 -29 -43 }
{ 43 26 -1 -28 -44 -42 -24 3 30 44 41 22 -6 -31 -45 -40 }
{ 43 23 -7 -34 -45 -36 -11 19 41 44 27 -2 -30 -45 -39 -15 }
{ 42 20 -12 -38 -45 -28 3 33 45 34 6 -26 -44 -39 -14 18 }}

T_{12} = {

{ 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 }
{ 31 30 28 26 24 22 20 18 16 14 12 10 8 6 3 1 }
{ -2 -7 -11 -15 -19 -23 -27 -30 -34 -36 -39 -41 -43 -44 -45 -45 }
{-34 -38 -41 -44 -45 -45 -45 -43 -40 -37 -33 -28 -22 -16 -10 -3 }
{-45 -43 -40 -35 -29 -21 -13 -4 4 13 21 29 35 40 43 45 }
{-28 -18 -8 3 14 24 33 39 44 45 44 40 34 26 16 6 }
{ 7 19 30 39 44 45 43 36 27 15 2 -11 -23 -34 -41 -45 }
{ 37 44 45 41 33 20 6 -10 -24 -36 -43 -45 -42 -34 -22 -8 }
{ 44 38 25 9 -9 -25 -38 -44 -44 -38 -25 -9 9 25 38 44 }
{ 24 6 -14 -31 -42 -45 -39 -26 -8 12 30 41 45 40 28 10 }
{-11 -30 -43 -45 -36 -19 2 23 39 45 41 27 7 -15 -34 -44 }
{-39 -45 -38 -20 3 26 41 45 36 16 -8 -30 -43 -44 -33 -12 }
{-43 -29 -4 21 40 45 35 13 -13 -35 -45 -40 -21 4 29 43 }
{-20 8 33 45 39 18 -10 -34 -45 -38 -16 12 36 45 37 14 }
{ 15 39 45 30 2 -27 -44 -41 -19 11 36 45 34 7 -23 -43 }
{ 41 43 22 -10 -37 -45 -30 1 31 45 36 8 -24 -44 -40 -16 }}

T_{13} = {

{ 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 }
{ -1 -3 -6 -8 -10 -12 -14 -16 -18 -20 -22 -24 -26 -28 -30 -31 }
{-45 -45 -44 -43 -41 -39 -36 -34 -30 -27 -23 -19 -15 -11 -7 -2 }
{ 3 10 16 22 28 33 37 40 43 45 45 45 44 41 38 34 }
{ 45 43 40 35 29 21 13 4 -4 -13 -21 -29 -35 -40 -43 -45 }
{ -6 -16 -26 -34 -40 -44 -45 -44 -39 -33 -24 -14 -3 8 18 28 }
{-45 -41 -34 -23 -11 2 15 27 36 43 45 44 39 30 19 7 }
{ 8 22 34 42 45 43 36 24 10 -6 -20 -33 -41 -45 -44 -37 }
{ 44 38 25 9 -9 -25 -38 -44 -44 -38 -25 -9 9 25 38 44 }

{-10 -28 -40 -45 -41 -30 -12 8 26 39 45 42 31 14 -6 -24 }
{-44 -34 -15 7 27 41 45 39 23 2 -19 -36 -45 -43 -30 -11 }
{ 12 33 44 43 30 8 -16 -36 -45 -41 -26 -3 20 38 45 39 }
{ 43 29 4 -21 -40 -45 -35 -13 13 35 45 40 21 -4 -29 -43 }
{-14 -37 -45 -36 -12 16 38 45 34 10 -18 -39 -45 -33 -8 20 }
{-43 -23 7 34 45 36 11 -19 -41 -44 -27 2 30 45 39 15 }
{ 16 40 44 24 -8 -36 -45 -31 -1 30 45 37 10 -22 -43 -41 }}

T_{14} = {

{ 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 }
{-33 -34 -36 -37 -38 -39 -40 -41 -42 -43 -44 -44 -45 -45 -45 -45 }
{ 2 7 11 15 19 23 27 30 34 36 39 41 43 44 45 45 }
{ 30 24 18 12 6 -1 -8 -14 -20 -26 -31 -36 -39 -42 -44 -45 }
{-45 -43 -40 -35 -29 -21 -13 -4 4 13 21 29 35 40 43 45 }
{ 36 41 45 45 43 38 31 22 12 1 -10 -20 -30 -37 -42 -45 }
{ -7 -19 -30 -39 -44 -45 -43 -36 -27 -15 -2 11 23 34 41 45 }
{-26 -12 3 18 31 40 45 44 38 28 14 -1 -16 -30 -39 -45 }
{ 44 38 25 9 -9 -25 -38 -44 -44 -38 -25 -9 9 25 38 44 }
{-38 -45 -43 -33 -16 3 22 37 45 44 34 18 -1 -20 -36 -44 }
{ 11 30 43 45 36 19 -2 -23 -39 -45 -41 -27 -7 15 34 44 }
{ 22 -1 -24 -40 -45 -37 -18 6 28 42 45 34 14 -10 -31 -44 }
{-43 -29 -4 21 40 45 35 13 -13 -35 -45 -40 -21 4 29 43 }
{ 40 45 31 6 -22 -41 -44 -30 -3 24 42 44 28 1 -26 -43 }
{-15 -39 -45 -30 -2 27 44 41 19 -11 -36 -45 -34 -7 23 43 }
{-18 14 39 44 26 -6 -34 -45 -33 -3 28 45 38 12 -20 -42 }}

T_{21} = {

{ 42 17 -17 -42 -42 -17 17 42 42 17 -17 -42 -42 -17 17 42 }
{ 41 14 -22 -44 -37 -6 30 45 31 -3 -36 -45 -24 12 40 42 }
{ 41 11 -27 -45 -30 7 39 43 15 -23 -45 -34 2 36 44 19 }
{ 40 8 -31 -45 -22 18 44 34 -3 -38 -42 -12 28 45 26 -14 }
{ 40 4 -35 -43 -13 29 45 21 -21 -45 -29 13 43 35 -4 -40 }
{ 39 1 -38 -40 -3 37 41 6 -36 -42 -8 34 43 10 -33 -44 }
{ 39 -2 -41 -36 7 43 34 -11 -44 -30 15 45 27 -19 -45 -23 }
{ 38 -6 -43 -31 16 45 22 -26 -45 -12 34 41 1 -40 -36 10 }
{ 38 -9 -44 -25 25 44 9 -38 -38 9 44 25 -25 -44 -9 38 }
{ 37 -12 -45 -18 33 40 -6 -44 -24 28 43 1 -42 -30 22 45 }
{ 36 -15 -45 -11 39 34 -19 -45 -7 41 30 -23 -44 -2 43 27 }
{ 36 -18 -45 -3 43 24 -31 -39 12 45 10 -40 -30 26 42 -6 }
{ 35 -21 -43 4 45 13 -40 -29 29 40 -13 -45 -4 43 21 -35 }
{ 34 -24 -41 12 45 1 -45 -14 40 26 -33 -36 22 42 -10 -45 }
{ 34 -27 -39 19 43 -11 -45 2 45 7 -44 -15 41 23 -36 -30 }
{ 33 -30 -36 26 38 -22 -40 18 42 -14 -44 10 45 -6 -45 1 }}

T_{22} = {

{ 42 17 -17 -42 -42 -17 17 42 42 17 -17 -42 -42 -17 17 42 }
{ 16 -20 -44 -38 -8 28 45 33 -1 -34 -45 -26 10 39 43 18 }

{-19 -44 -36 -2 34 45 23 -15 -43 -39 -7 30 45 27 -11 -41 }
{-43 -37 -1 36 44 16 -24 -45 -30 10 41 39   6 -33 -45 -20 }
{-40 -4 35 43 13 -29 -45 -21 21 45 29 -13 -43 -35   4 40 }
{-12 31 44 14 -30 -45 -16 28 45 18 -26 -45 -20 24 45 22 }
{ 23 45 19 -27 -45 -15 30 44 11 -34 -43 -7 36 41   2 -39 }
{ 44 28 -20 -45 -18 30 44   8 -37 -39   3 42 33 -14 -45 -24 }
{ 38 -9 -44 -25 25 44   9 -38 -38   9 44 25 -25 -44 -9 38 }
{ 8 -39 -34 16 45 14 -36 -38 10 45 20 -31 -41   3 44 26 }
{-27 -43   2 44 23 -30 -41   7 45 19 -34 -39 11 45 15 -36 }
{-45 -16 37 34 -20 -44 -1 44 22 -33 -38 14 45   8 -41 -28 }
{-35 21 43 -4 -45 -13 40 29 -29 -40 13 45   4 -43 -21 35 }
{ -3 44 16 -39 -28 31 37 -20 -43   8 45   6 -44 -18 38 30 }
{ 30 36 -23 -41 15 44 -7 -45 -2 45 11 -43 -19 39 27 -34 }
{ 45   3 -45 -8 44 12 -43 -16 41 20 -39 -24 37 28 -34 -31 }}

T_{23} = {
{ 42 17 -17 -42 -42 -17 17 42 42 17 -17 -42 -42 -17 17 42 }
{-18 -43 -39 -10 26 45 34   1 -33 -45 -28   8 38 44 20 -16 }
{-41 -11 27 45 30 -7 -39 -43 -15 23 45 34 -2 -36 -44 -19 }
{ 20 45 33 -6 -39 -41 -10 30 45 24 -16 -44 -36   1 37 43 }
{ 40   4 -35 -43 -13 29 45 21 -21 -45 -29 13 43 35 -4 -40 }
{-22 -45 -24 20 45 26 -18 -45 -28 16 45 30 -14 -44 -31 12 }
{-39   2 41 36 -7 -43 -34 11 44 30 -15 -45 -27 19 45 23 }
{ 24 45 14 -33 -42 -3 39 37 -8 -44 -30 18 45 20 -28 -44 }
{ 38 -9 -44 -25 25 44   9 -38 -38   9 44 25 -25 -44 -9 38 }
{-26 -44 -3 41 31 -20 -45 -10 38 36 -14 -45 -16 34 39 -8 }
{-36 15 45 11 -39 -34 19 45   7 -41 -30 23 44   2 -43 -27 }
{ 28 41 -8 -45 -14 38 33 -22 -44   1 44 20 -34 -37 16 45 }
{ 35 -21 -43   4 45 13 -40 -29 29 40 -13 -45 -4 43 21 -35 }
{-30 -38 18 44 -6 -45 -8 43 20 -37 -31 28 39 -16 -44   3 }
{-34 27 39 -19 -43 11 45 -2 -45 -7 44 15 -41 -23 36 30 }
{ 31 34 -28 -37 24 39 -20 -41 16 43 -12 -44   8 45 -3 -45 }}

T_{24} = {
{ 42 17 -17 -42 -42 -17 17 42 42 17 -17 -42 -42 -17 17 42 }
{-42 -40 -12 24 45 36   3 -31 -45 -30   6 37 44 22 -14 -41 }
{ 19 44 36   2 -34 -45 -23 15 43 39   7 -30 -45 -27 11 41 }
{ 14 -26 -45 -28 12 42 38   3 -34 -44 -18 22 45 31 -8 -40 }
{-40 -4 35 43 13 -29 -45 -21 21 45 29 -13 -43 -35   4 40 }
{ 44 33 -10 -43 -34   8 42 36 -6 -41 -37   3 40 38 -1 -39 }
{-23 -45 -19 27 45 15 -30 -44 -11 34 43   7 -36 -41 -2 39 }
{-10 36 40 -1 -41 -34 12 45 26 -22 -45 -16 31 43   6 -38 }
{ 38 -9 -44 -25 25 44   9 -38 -38   9 44 25 -25 -44 -9 38 }
{-45 -22 30 42 -1 -43 -28 24 44   6 -40 -33 18 45 12 -37 }
{ 27 43 -2 -44 -23 30 41 -7 -45 -19 34 39 -11 -45 -15 36 }
{ 6 -42 -26 30 40 -10 -45 -12 39 31 -24 -43   3 45 18 -36 }

{-35 21 43 -4 -45 -13 40 29 -29 -40 13 45 4 -43 -21 35 }
{ 45 10 -42 -22 36 33 -26 -40 14 45 -1 -45 -12 41 24 -34 }
{-30 -36 23 41 -15 -44 7 45 2 -45 -11 43 19 -39 -27 34 }
{ -1 45 6 -45 -10 44 14 -42 -18 40 22 -38 -26 36 30 -33 }}

T_{31} = {
{ 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 }
{ 31 -34 -28 37 24 -39 -20 41 16 -43 -12 44 8 -45 -3 45 }
{ 30 -36 -23 41 15 -44 -7 45 -2 -45 11 43 -19 -39 27 34 }
{ 30 -38 -18 44 6 -45 8 43 -20 -37 31 28 -39 -16 44 3 }
{ 29 -40 -13 45 -4 -43 21 35 -35 -21 43 4 -45 13 40 -29 }
{ 28 -41 -8 45 -14 -38 33 22 -44 -1 44 -20 -34 37 16 -45 }
{ 27 -43 -2 44 -23 -30 41 7 -45 19 34 -39 -11 45 -15 -36 }
{ 26 -44 3 41 -31 -20 45 -10 -38 36 14 -45 16 34 -39 -8 }
{ 25 -44 9 38 -38 -9 44 -25 -25 44 -9 -38 38 9 -44 25 }
{ 24 -45 14 33 -42 3 39 -37 -8 44 -30 -18 45 -20 -28 44 }
{ 23 -45 19 27 -45 15 30 -44 11 34 -43 7 36 -41 2 39 }
{ 22 -45 24 20 -45 26 18 -45 28 16 -45 30 14 -44 31 12 }
{ 21 -45 29 13 -43 35 4 -40 40 -4 -35 43 -13 -29 45 -21 }
{ 20 -45 33 6 -39 41 -10 -30 45 -24 -16 44 -36 -1 37 -43 }
{ 19 -44 36 -2 -34 45 -23 -15 43 -39 7 30 -45 27 11 -41 }
{ 18 -43 39 -10 -26 45 -34 1 33 -45 28 8 -38 44 -20 -16 }}

T_{32} = {
{ 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 }
{ -1 -45 6 45 -10 -44 14 42 -18 -40 22 38 -26 -36 30 33 }
{-34 -27 39 19 -43 -11 45 2 -45 7 44 -15 -41 23 36 -30 }
{-45 10 42 -22 -36 33 26 -40 -14 45 1 -45 12 41 -24 -34 }
{-29 40 13 -45 4 43 -21 -35 35 21 -43 -4 45 -13 -40 29 }
{ 6 42 -26 -30 40 10 -45 12 39 -31 -24 43 3 -45 18 36 }
{ 36 15 -45 11 39 -34 -19 45 -7 -41 30 23 -44 2 43 -27 }
{ 45 -22 -30 42 1 -43 28 24 -44 6 40 -33 -18 45 -12 -37 }
{ 25 -44 9 38 -38 -9 44 -25 -25 44 -9 -38 38 9 -44 25 }
{-10 -36 40 1 -41 34 12 -45 26 22 -45 16 31 -43 6 38 }
{-39 -2 41 -36 -7 43 -34 -11 44 -30 -15 45 -27 -19 45 -23 }
{-44 33 10 -43 34 8 -42 36 6 -41 37 3 -40 38 1 -39 }
{-21 45 -29 -13 43 -35 -4 40 -40 4 35 -43 13 29 -45 21 }
{ 14 26 -45 28 12 -42 38 -3 -34 44 -18 -22 45 -31 -8 40 }
{ 41 -11 -27 45 -30 -7 39 -43 15 23 -45 34 2 -36 44 -19 }
{ 42 -40 12 24 -45 36 -3 -31 45 -30 -6 37 -44 22 14 -41 }}

T_{33} = {
{ 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 }
{-33 -30 36 26 -38 -22 40 18 -42 -14 44 10 -45 -6 45 1 }
{-30 36 23 -41 -15 44 7 -45 2 45 -11 -43 19 39 -27 -34 }
{ 34 24 -41 -12 45 -1 -45 14 40 -26 -33 36 22 -42 -10 45 }
{ 29 -40 -13 45 -4 -43 21 35 -35 -21 43 4 -45 13 40 -29 }

{-36 -18 45 -3 -43 24 31 -39 -12 45 -10 -40 30 26 -42 -6 }
{-27 43   2 -44 23 30 -41 -7 45 -19 -34 39 11 -45 15 36 }
{ 37 12 -45 18 33 -40 -6 44 -24 -28 43 -1 -42 30 22 -45 }
{ 25 -44   9 38 -38 -9 44 -25 -25 44 -9 -38 38   9 -44 25 }
{-38 -6 43 -31 -16 45 -22 -26 45 -12 -34 41 -1 -40 36 10 }
{-23 45 -19 -27 45 -15 -30 44 -11 -34 43 -7 -36 41 -2 -39 }
{ 39 -1 -38 40 -3 -37 41 -6 -36 42 -8 -34 43 -10 -33 44 }
{ 21 -45 29 13 -43 35   4 -40 40 -4 -35 43 -13 -29 45 -21 }
{-40   8 31 -45 22 18 -44 34   3 -38 42 -12 -28 45 -26 -14 }
{-19 44 -36   2 34 -45 23 15 -43 39 -7 -30 45 -27 -11 41 }
{ 41 -14 -22 44 -37   6 30 -45 31   3 -36 45 -24 -12 40 -42 }}

T_{34} = {

{ 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 32 -32 -32 32 }
{-45   3 45 -8 -44 12 43 -16 -41 20 39 -24 -37 28 34 -31 }
{ 34 27 -39 -19 43 11 -45 -2 45 -7 -44 15 41 -23 -36 30 }
{ -3 -44 16 39 -28 -31 37 20 -43 -8 45 -6 -44 18 38 -30 }
{-29 40 13 -45   4 43 -21 -35 35 21 -43 -4 45 -13 -40 29 }
{ 45 -16 -37 34 20 -44   1 44 -22 -33 38 14 -45   8 41 -28 }
{-36 -15 45 -11 -39 34 19 -45   7 41 -30 -23 44 -2 -43 27 }
{ 8 39 -34 -16 45 -14 -36 38 10 -45 20 31 -41 -3 44 -26 }
{ 25 -44   9 38 -38 -9 44 -25 -25 44 -9 -38 38   9 -44 25 }
{-44 28 20 -45 18 30 -44   8 37 -39 -3 42 -33 -14 45 -24 }
{ 39   2 -41 36   7 -43 34 11 -44 30 15 -45 27 19 -45 23 }
{-12 -31 44 -14 -30 45 -16 -28 45 -18 -26 45 -20 -24 45 -22 }
{-21 45 -29 -13 43 -35 -4 40 -40   4 35 -43 13 29 -45 21 }
{ 43 -37   1 36 -44 16 24 -45 30 10 -41 39 -6 -33 45 -20 }
{-41 11 27 -45 30   7 -39 43 -15 -23 45 -34 -2 36 -44 19 }
{ 16 20 -44 38 -8 -28 45 -33 -1 34 -45 26 10 -39 43 -18 }}

T_{41} = {

{ 17 -42 42 -17 -17 42 -42 17 17 -42 42 -17 -17 42 -42 17 }
{ 16 -40 44 -24 -8 36 -45 31 -1 -30 45 -37 10 22 -43 41 }
{ 15 -39 45 -30   2 27 -44 41 -19 -11 36 -45 34 -7 -23 43 }
{ 14 -37 45 -36 12 16 -38 45 -34 10 18 -39 45 -33   8 20 }
{ 13 -35 45 -40 21   4 -29 43 -43 29 -4 -21 40 -45 35 -13 }
{ 12 -33 44 -43 30 -8 -16 36 -45 41 -26   3 20 -38 45 -39 }
{ 11 -30 43 -45 36 -19 -2 23 -39 45 -41 27 -7 -15 34 -44 }
{ 10 -28 40 -45 41 -30 12   8 -26 39 -45 42 -31 14   6 -24 }
{ 9 -25 38 -44 44 -38 25 -9 -9 25 -38 44 -44 38 -25   9 }
{ 8 -22 34 -42 45 -43 36 -24 10   6 -20 33 -41 45 -44 37 }
{ 7 -19 30 -39 44 -45 43 -36 27 -15   2 11 -23 34 -41 45 }
{ 6 -16 26 -34 40 -44 45 -44 39 -33 24 -14   3   8 -18 28 }
{ 4 -13 21 -29 35 -40 43 -45 45 -43 40 -35 29 -21 13 -4 }
{ 3 -10 16 -22 28 -33 37 -40 43 -45 45 -45 44 -41 38 -34 }
{ 2 -7 11 -15 19 -23 27 -30 34 -36 39 -41 43 -44 45 -45 }

{ 1 -3 6 -8 10 -12 14 -16 18 -20 22 -24 26 -28 30 -31 }}

T_{42} = {

{ 17 -42 42 -17 -17 42 -42 17 17 -42 42 -17 -17 42 -42 17 }
{-18 -14 39 -44 26 6 -34 45 -33 3 28 -45 38 -12 -20 42 }
{-43 23 7 -34 45 -36 11 19 -41 44 -27 -2 30 -45 39 -15 }
{-40 45 -31 6 22 -41 44 -30 3 24 -42 44 -28 1 26 -43 }
{-13 35 -45 40 -21 -4 29 -43 43 -29 4 21 -40 45 -35 13 }
{ 22 1 -24 40 -45 37 -18 -6 28 -42 45 -34 14 10 -31 44 }
{ 44 -34 15 7 -27 41 -45 39 -23 2 19 -36 45 -43 30 -11 }
{ 38 -45 43 -33 16 3 -22 37 -45 44 -34 18 1 -20 36 -44 }
{ 9 -25 38 -44 44 -38 25 -9 -9 25 -38 44 -44 38 -25 9 }
{-26 12 3 -18 31 -40 45 -44 38 -28 14 1 -16 30 -39 45 }
{-45 41 -34 23 -11 -2 15 -27 36 -43 45 -44 39 -30 19 -7 }
{-36 41 -45 45 -43 38 -31 22 -12 1 10 -20 30 -37 42 -45 }
{ -4 13 -21 29 -35 40 -43 45 -45 43 -40 35 -29 21 -13 4 }
{ 30 -24 18 -12 6 1 -8 14 -20 26 -31 36 -39 42 -44 45 }
{ 45 -45 44 -43 41 -39 36 -34 30 -27 23 -19 15 -11 7 -2 }
{ 33 -34 36 -37 38 -39 40 -41 42 -43 44 -44 45 -45 45 -45 }}

T_{43} = {

{ 17 -42 42 -17 -17 42 -42 17 17 -42 42 -17 -17 42 -42 17 }
{-42 20 12 -38 45 -28 -3 33 -45 34 -6 -26 44 -39 14 18 }
{-15 39 -45 30 -2 -27 44 -41 19 11 -36 45 -34 7 23 -43 }
{ 43 -26 -1 28 -44 42 -24 -3 30 -44 41 -22 -6 31 -45 40 }
{ 13 -35 45 -40 21 4 -29 43 -43 29 -4 -21 40 -45 35 -13 }
{-44 31 -10 -14 34 -45 42 -28 6 18 -37 45 -40 24 -1 -22 }
{-11 30 -43 45 -36 19 2 -23 39 -45 41 -27 7 15 -34 44 }
{ 44 -36 20 -1 -18 34 -44 45 -37 22 -3 -16 33 -43 45 -38 }
{ 9 -25 38 -44 44 -38 25 -9 -9 25 -38 44 -44 38 -25 9 }
{-45 39 -30 16 -1 -14 28 -38 44 -45 40 -31 18 -3 -12 26 }
{ -7 19 -30 39 -44 45 -43 36 -27 15 -2 -11 23 -34 41 -45 }
{ 45 -42 37 -30 20 -10 -1 12 -22 31 -38 43 -45 45 -41 36 }
{ 4 -13 21 -29 35 -40 43 -45 45 -43 40 -35 29 -21 13 -4 }
{-45 44 -42 39 -36 31 -26 20 -14 8 -1 -6 12 -18 24 -30 }
{ -2 7 -11 15 -19 23 -27 30 -34 36 -39 41 -43 44 -45 45 }
{ 45 -45 45 -45 44 -44 43 -42 41 -40 39 -38 37 -36 34 -33 }}

T_{44} = {

{ 17 -42 42 -17 -17 42 -42 17 17 -42 42 -17 -17 42 -42 17 }
{-41 43 -22 -10 37 -45 30 1 -31 45 -36 8 24 -44 40 -16 }
{ 43 -23 -7 34 -45 36 -11 -19 41 -44 27 2 -30 45 -39 15 }
{-20 -8 33 -45 39 -18 -10 34 -45 38 -16 -12 36 -45 37 -14 }
{-13 35 -45 40 -21 -4 29 -43 43 -29 4 21 -40 45 -35 13 }
{ 39 -45 38 -20 -3 26 -41 45 -36 16 8 -30 43 -44 33 -12 }
{-44 34 -15 -7 27 -41 45 -39 23 -2 -19 36 -45 43 -30 11 }
{ 24 -6 -14 31 -42 45 -39 26 -8 -12 30 -41 45 -40 28 -10 }

{ 9 -25 38 -44 44 -38 25 -9 -9 25 -38 44 -44 38 -25   9 }
{-37 44 -45 41 -33 20 -6 -10 24 -36 43 -45 42 -34 22 -8 }
{ 45 -41 34 -23 11   2 -15 27 -36 43 -45 44 -39 30 -19   7 }
{-28 18 -8 -3 14 -24 33 -39 44 -45 44 -40 34 -26 16 -6 }
{ -4 13 -21 29 -35 40 -43 45 -45 43 -40 35 -29 21 -13   4 }
{ 34 -38 41 -44 45 -45 45 -43 40 -37 33 -28 22 -16 10 -3 }
{-45 45 -44 43 -41 39 -36 34 -30 27 -23 19 -15 11 -7   2 }
{ 31 -30 28 -26 24 -22 20 -18 16 -14 12 -10   8 -6   3 -1 }}

G.2 DCT8 型反变换矩阵

G.2.1 4×4 DCT8反变换矩阵

4×4 DCT8反变换矩阵定义如下：

$DCT8_4$ = {
{ 42 37 27 15 }
{ 37   0 -37 -37 }
{ 27 -37 -15 42 }
{ 15 -37 42 -27 }}

G.2.2 8×8 DCT8反变换矩阵

8×8 DCT8反变换矩阵定义如下：

$DCT8_8$ = {
{ 44 42 39 35 30 23 16   8 }
{ 42 30   8 -16 -35 -44 -39 -23 }
{ 39   8 -30 -44 -23 16 42 35 }
{ 35 -16 -44 -8 39 30 -23 -42 }
{ 30 -35 -23 39 16 -42 -8 44 }
{ 23 -44 16 30 -42   8 35 -39 }
{ 16 -39 42 -23 -8 35 -44 30 }
{ 8 -23 35 -42 44 -39 30 -16 }}

G.2.3 16×16 DCT8反变换矩阵

16×16 DCT8反变换矩阵定义如下：

$DCT8_{16}$ = {
{ 45 44 43 42 41 39 36 34 31 28 24 20 17 13   8   4 }
{ 44 41 34 24 13   0 -13 -24 -34 -41 -44 -44 -41 -34 -24 -13 }
{ 43 34 17 -4 -24 -39 -45 -41 -28 -8 13 31 42 44 36 20 }
{ 42 24 -4 -31 -44 -39 -17 13 36 45 34   8 -20 -41 -43 -28 }
{ 41 13 -24 -44 -34   0 34 44 24 -13 -41 -41 -13 24 44 34 }
{ 39   0 -39 -39   0 39 39   0 -39 -39   0 39 39   0 -39 -39 }
{ 36 -13 -45 -17 34 39 -8 -44 -20 31 41 -4 -43 -24 28 42 }
{ 34 -24 -41 13 44   0 -44 -13 41 24 -34 -34 24 41 -13 -44 }
{ 31 -34 -28 36 24 -39 -20 41 17 -42 -13 43   8 -44 -4 45 }
{ 28 -41 -8 45 -13 -39 31 24 -42 -4 44 -17 -36 34 20 -43 }

{ 24 -44 13 34 -41   0 41 -34 -13 44 -24 -24 44 -13 -34 41 }
{ 20 -44 31   8 -41 39 -4 -34 43 -17 -24 45 -28 -13 42 -36 }
{ 17 -41 42 -20 -13 39 -43 24   8 -36 44 -28 -4 34 -45 31 }
{ 13 -34 44 -41 24   0 -24 41 -44 34 -13 -13 34 -44 41 -24 }
{ 8 -24 36 -43 44 -39 28 -13 -4 20 -34 42 -45 41 -31 17 }
{ 4 -13 20 -28 34 -39 42 -44 45 -43 41 -36 31 -24 17 -8 }}

G.3 DST7 型反变换矩阵

G.3.1 4×4 DST7反变换矩阵

4×4 DST7反变换矩阵定义如下：

$DST7_4$ = {
{ 15 27 37 42 }
{ 37 37   0 -37 }
{ 42 -15 -37 27 }
{ 27 -42 37 -15 }}

G.3.2 8×8 DST7反变换矩阵

8×8 DST7反变换矩阵定义如下：

$DST7_8$ = {
{ 8 16 23 30 35 39 42 44 }
{ 23 39 44 35 16 -8 -30 -42 }
{ 35 42 16 -23 -44 -30   8 39 }
{ 42 23 -30 -39   8 44 16 -35 }
{ 44 -8 -42 16 39 -23 -35 30 }
{ 39 -35 -8 42 -30 -16 44 -23 }
{ 30 -44 35 -8 -23 42 -39 16 }
{ 16 -30 39 -44 42 -35 23 -8 }}

G.3.3 16×16 DST7反变换矩阵

16×16 DST7反变换矩阵定义如下：

$DST7_{16}$ = {
{ 4   8 13 17 20 24 28 31 34 36 39 41 42 43 44 45 }
{ 13 24 34 41 44 44 41 34 24 13   0 -13 -24 -34 -41 -44 }
{ 20 36 44 42 31 13 -8 -28 -41 -45 -39 -24 -4 17 34 43 }
{ 28 43 41 20 -8 -34 -45 -36 -13 17 39 44 31   4 -24 -42 }
{ 34 44 24 -13 -41 -41 -13 24 44 34   0 -34 -44 -24 13 41 }
{ 39 39   0 -39 -39   0 39 39   0 -39 -39   0 39 39   0 -39 }
{ 42 28 -24 -43 -4 41 31 -20 -44 -8 39 34 -17 -45 -13 36 }
{ 44 13 -41 -24 34 34 -24 -41 13 44   0 -44 -13 41 24 -34 }
{ 45 -4 -44   8 43 -13 -42 17 41 -20 -39 24 36 -28 -34 31 }
{ 43 -20 -34 36 17 -44   4 42 -24 -31 39 13 -45   8 41 -28 }
{ 41 -34 -13 44 -24 -24 44 -13 -34 41   0 -41 34 13 -44 24 }
{ 36 -42 13 28 -45 24 17 -43 34   4 -39 41 -8 -31 44 -20 }

{ 31 -45 34 -4 -28 44 -36   8 24 -43 39 -13 -20 42 -41 17 }
{ 24 -41 44 -34 13 13 -34 44 -41 24   0 -24 41 -44 34 -13 }
{ 17 -31 41 -45 42 -34 20 -4 -13 28 -39 44 -43 36 -24   8 }
{ 8 -17 24 -31 36 -41 43 -45 44 -42 39 -34 28 -20 13 -4 }}

G.3.4 32×32 DST7反变换矩阵

32×32 DST7反变换矩阵定义如下：

$DST7_{32}$ = {
{T_{11} T_{12}}
{T_{21} T_{22}}}
其中，T_{11} = {
{ 2   4   6   9 11 13 15 17 19 21 23 25 26 28 30 31}
{ 6 13 19 25 30 34 38 41 43 45 45 44 43 40 37 33}
{ 11 21 30 37 42 45 45 42 37 30 21 11   0 -11 -21 -30}
{ 15 28 38 44 45 40 31 19   4 -11 -25 -36 -43 -45 -42 -34}
{ 19 34 43 44 37 23   4 -15 -31 -42 -45 -39 -26 -9 11 28}
{ 23 39 45 38 21 -2 -25 -40 -45 -37 -19   4 26 41 45 36}
{ 26 43 43 26   0 -26 -43 -43 -26   0 26 43 43 26   0 -26}
{ 30 45 37 11 -21 -42 -42 -21 11 37 45 30   0 -30 -45 -37}
{ 33 45 28 -6 -37 -44 -23 13 40 42 17 -19 -43 -39 -11 25}
{ 36 43 17 -23 -45 -31   6 39 41 11 -28 -45 -26 13 42 38}
{ 38 40   4 -36 -42 -9 33 43 13 -30 -44 -17 26 45 21 -23}
{ 40 36 -9 -43 -30 17 45 23 -25 -45 -15 31 43   6 -37 -39}
{ 42 30 -21 -45 -11 37 37 -11 -45 -21 30 42   0 -42 -30 21}
{ 43 23 -31 -39 11 45 13 -38 -33 21 44   2 -43 -25 30 40}
{ 44 15 -39 -28 30 38 -17 -44   2 45 13 -40 -26 31 37 -19}
{ 45   6 -44 -13 42 19 -39 -25 36 30 -31 -34 26 38 -21 -41}}
T_{12} = {
{ 33 34 36 37 38 39 40 41 42 43 43 44 44 45 45 45}
{ 28 23 17 11   4 -2 -9 -15 -21 -26 -31 -36 -39 -42 -44 -45}
{-37 -42 -45 -45 -42 -37 -30 -21 -11   0 11 21 30 37 42 45}
{-23 -9   6 21 33 41 45 43 37 26 13 -2 -17 -30 -39 -44}
{ 40 45 41 30 13 -6 -25 -38 -45 -43 -33 -17   2 21 36 44}
{ 17 -6 -28 -42 -44 -34 -15   9 30 43 44 33 13 -11 -31 -43}
{-43 -43 -26   0 26 43 43 26   0 -26 -43 -43 -26   0 26 43}
{-11 21 42 42 21 -11 -37 -45 -30   0 30 45 37 11 -21 -42}
{ 44 36   4 -30 -45 -31   2 34 45 26 -9 -38 -43 -21 15 41}
{ 4 -33 -44 -21 19 44 34 -2 -37 -43 -15 25 45 30 -9 -40}
{-45 -25 19 45 28 -15 -44 -31 11 43 34 -6 -41 -37   2 39}
{ 2 41 34 -11 -44 -28 19 45 21 -26 -44 -13 33 42   4 -38}
{ 45 11 -37 -37 11 45 21 -30 -42   0 42 30 -21 -45 -11 37}
{ -9 -45 -15 37 34 -19 -44 -4 42 26 -28 -41   6 45 17 -36}
{-43   4 45 11 -41 -25 33 36 -21 -43   6 45   9 -42 -23 34}
{ 15 43 -9 -45   2 45   4 -44 -11 43 17 -40 -23 37 28 -33}}

T_{21} = {
{ 45 -2 -45 4 45 -6 -44 9 44 -11 -43 13 43 -15 -42 17}
{ 45 -11 -42 21 37 -30 -30 37 21 -42 -11 45 0 -45 11 42}
{ 44 -19 -36 34 21 -43 -2 44 -17 -37 33 23 -43 -4 45 -15}
{ 43 -26 -26 43 0 -43 26 26 -43 0 43 -26 -26 43 0 -43}
{ 41 -33 -15 45 -21 -28 43 -6 -38 37 9 -44 26 23 -45 13}
{ 39 -38 -2 40 -37 -4 41 -36 -6 42 -34 -9 43 -33 -11 43}
{ 37 -42 11 30 -45 21 21 -45 30 11 -42 37 0 -37 42 -11}
{ 34 -44 23 15 -42 39 -9 -28 45 -30 -6 38 -43 17 21 -44}
{ 31 -45 33 -2 -30 45 -34 4 28 -45 36 -6 -26 44 -37 9}
{ 28 -44 40 -19 -11 36 -45 34 -9 -21 41 -43 26 2 -30 44}
{ 25 -41 44 -33 11 15 -36 45 -39 21 4 -28 43 -43 30 -6}
{ 21 -37 45 -42 30 -11 -11 30 -42 45 -37 21 0 -21 37 -45}
{ 17 -31 41 -45 42 -33 19 -2 -15 30 -40 45 -43 34 -21 4}
{ 13 -25 34 -41 45 -44 40 -33 23 -11 -2 15 -26 36 -42 45}
{ 9 -17 25 -31 37 -41 44 -45 44 -42 38 -33 26 -19 11 -2}
{ 4 -9 13 -17 21 -25 28 -31 34 -37 39 -41 43 -44 45 -45}}
T_{22} = {
{ 41 -19 -40 21 39 -23 -38 25 37 -26 -36 28 34 -30 -33 31}
{-21 -37 30 30 -37 -21 42 11 -45 0 45 -11 -42 21 37 -30}
{-38 31 25 -42 -6 45 -13 -39 30 26 -41 -9 45 -11 -40 28}
{ 26 26 -43 0 43 -26 -26 43 0 -43 26 26 -43 0 43 -26}
{ 34 -40 -2 42 -31 -17 45 -19 -30 43 -4 -39 36 11 -44 25}
{-31 -13 44 -30 -15 44 -28 -17 45 -26 -19 45 -25 -21 45 -23}
{-30 45 -21 -21 45 -30 -11 42 -37 0 37 -42 11 30 -45 21}
{ 36 -2 -33 45 -25 -13 41 -40 11 26 -45 31 4 -37 43 -19}
{ 25 -44 38 -11 -23 43 -39 13 21 -43 40 -15 -19 42 -41 17}
{-39 17 13 -37 45 -33 6 23 -42 43 -25 -4 31 -45 38 -15}
{-19 38 -45 37 -17 -9 31 -44 42 -26 2 23 -40 45 -34 13}
{ 42 -30 11 11 -30 42 -45 37 -21 0 21 -37 45 -42 30 -11}
{ 13 -28 39 -45 43 -36 23 -6 -11 26 -38 44 -44 37 -25 9}
{-44 39 -31 21 -9 -4 17 -28 37 -43 45 -43 38 -30 19 -6}
{ -6 15 -23 30 -36 40 -43 45 -45 43 -39 34 -28 21 -13 4}
{ 45 -44 43 -42 40 -38 36 -33 30 -26 23 -19 15 -11 6 -2}}