ICS 35.040
L 71

团　　体　　标　　准

T/AI 109.2—2020

信息技术 智能媒体编码
第 2 部分：视频

Information Technology Intelligent Media Coding

Part 2: Video

2020－05－13 发布　　2020－05－13 实施

中关村视听产业技术创新联盟　发布

目　次

前言 II
引言 III
1　范围 1
2　规范性引用文件 1
3　术语和定义 1
4　缩略语 8
5　约定 9
6　编码位流的结构 16
7　位流的语法和语义 21
8　解析过程 96
9　解码过程 115
附录 A（规范性附录）　伪起始码方法 196
附录 B（规范性附录）　档次和级别 197
附录 C（规范性附录）　位流参考缓冲区管理 206
附录 D（规范性附录）　加权量化矩阵 211
附录 E（规范性附录）　扫描表 212
附录 F（资料性附录）　高级熵编码器解码器参考描述方法 213
附录 G（规范性附录）　反变换矩阵 216

前　言

本标准按照GB/T 1.1—2009给出的规则起草。

本标准由数字音视频编解码技术标准工作组提出。

本标准由中关村视听产业技术创新联盟归口。

本标准起草单位：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。

本标准起草人：高文、黄铁军、马思伟、郑萧桢、郑建铧、梁凡、王苫社、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、陈焕浜、鲁晓牧、何鸣、马海川、陈震中、李一鸣、王梦杨、罗法蕾、雷萌。

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、7.1.3、7.1.5、7.1.6、7.1.8、7.2、7.2.5、7.2.6、8.3、8.3.3、8.3.4.11、9.2.6、9.5.4、9.5.8.2、9.5.8.4、9.5.8.5、9.5.8.6.4、9.6、9.7.4.2、9.7.5、9.8、9.10中如下19项与数字视频编解码技术相关的专利的使用。专利名称如下：

ZL03157076.3，一种定位编码图像标识的方法；ZL201110347691.9，一种视频图像滤波处理方法和装置；ZL201210546537.9，一种多方向的帧内预测编解码方法及装置；L201210546675.7，一种头信息编解码、解码方法及装置；ZL201310128415.2，一种视频编解码环路滤波的实现方法；ZL201310008682.6，一种运动矢量预测的方法；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；201680091331.8，通过块映射来对图像进行编码或解码的方法和装置；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；PCT/KR2018/003800，Apparatus and Method for Encoding Motion Vector Determined using Adaptive Motion Vector Resolution, and Apparatus and Method for Decoding Motion Vector；201910011266.9，对运动矢量信息进行编／解码的方法及装置；200780000403.4，在编解码中的实现量化的方法和装置；201910679070.7，图像显示顺序的确定方法、装置和视频编解码设备；201910335981.8，视频编码器、视频解码器及相应方法；201910108004.4，一种帧间预测的方法和装置；201910888383.3，视频解码方法、视频编码方法、装 置、设备及存储介质；201810581662.0，一种编解码方法及装置；201811198624.3，图像块的划分方法和装置；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。

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（数字音视频编解码技术标准工作组秘书长）

通讯地址：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这些专利的责任。

信息技术 智能媒体编码 第2部分：视频

1 范围

本部分规定了适应多种位率、分辨率和质量要求的智能媒体高效视频压缩方法的解码过程。

本部分适用于电视广播、数字电影、网络电视、网络视频、视频监控、实时通信、即时通信、数字存储媒体、静止图像等应用。

2 规范性引用文件

下列文件对于本部分的应用是必不可少的。凡是注日期的引用文件，仅注日期的版本适用于本部分。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本部分。

GY/T 155-2000 高清晰度电视节目制作及交换用视频参数值

ISO 11664-1/CIE S 014-1 色度 第1部分：标准比色观测器（Colorimetry – Part 1: Standard Colorimetric Observers）

ISO 11664-3/CIE S 014-3 色度 第3部分：CIE三色值（Colorimetry – Part 3: CIE Tristimulus Values）

CIE S 015 室外工作场景照明（Lighting of Outdoor Workplaces）

GB/T 33475.1-2019 信息技术 高效多媒体编码 第1部分:系统

GY/T 307-2017 超高清晰度电视系统节目制作和交换参数值

GY/T 315-2018 高动态范围电视节目制作和交换图像参数值

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

帧间预测中可使用显示顺序上过去的和将来的参考图像进行解码的图像。

3.2

保留 reserved

定义了一些特定语法元素值，这些值用于将来对本部分的扩展。

注：这些值不应出现在符合本部分的位流中。

3.3

变换系数 transform coefficient

变换域上的一个标量。

3.4

编码单元 coding unit

包括一个亮度编码块和对应的色度编码块，由最大编码单元划分得到。

3.5

编码块 coding block

一个M×M的样值块。编码块由最大编码块划分得到。

3.6

编码图像　coded picture

一幅图像的编码表示。

3.7

补偿后样本　compensated sample

经预测补偿得到的样本。

3.8

残差　residual

样本或数据元素的重建值与其预测值之差。

3.9

参考索引　reference index

参考图像在参考图像队列中的编号。其中在参考图像队列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。

3.10

参考图像　reference picture

解码过程中用于后续图像帧间预测的图像。

3.11

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，可包括参考图像队列0和参考图像队列1。

3.12

参考图像队列0　reference picture list 0

当前图像的参考图像所组成的图像队列0，可包括显示顺序上过去的和将来的图像。

3.13

参考图像队列1　reference picture list 1

当前图像的参考图像所组成的图像队列1，可包括显示顺序上过去的和将来的图像。

3.14

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。

3.15

层　layer

位流中的分级结构，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。

3.16

场　field

由构成帧的三个样本矩阵中相间的行构成。

3.17

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。

3.18

二元符号　bin

组成二元符号串的符号，包括‘0’和‘1’。

3.19

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。

3.20

反变换 inverse transform

将变换系数矩阵转换成空域样值矩阵的过程。

3.21

反量化 dequantization

对量化系数缩放后得到变换系数的过程。

3.22

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。

3.23

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。

3.24

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、L1运动矢量、L0参考索引和L1参考索引构成。

3.25

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。

3.26

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，然后扫描第二行、第三行，依次类推。光栅中的行从左到右扫描。

3.27

划分 partition

将一个集合分为子集。集合中的每个元素属于且只属于某一个子集。

3.28

划分方式 partition type

划分获得的子集的组织方式。

3.29

I图像 I picture

只使用帧内预测解码的图像。

3.30

级别 level

在某一档次下对语法元素和语法元素参数值的限定集合。

3.31

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。

3.32

解码图像 decoded picture

解码器根据位流重建的图像。

3.33

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：包括参考图像缓冲区。

3.34

解析过程　parse

由位流获得语法元素的过程。

3.35

禁止　forbidden

定义了一些特定语法元素值。

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。

3.36

块　block

一个M×N（M列N行）的样值矩阵或者变换系数矩阵。

3.37

块扫描　block scan

量化系数的特定串行排序方式。

3.38

档次　profile

本部分规定的语法、语义及算法的子集。

3.39

历史运动信息表　historical motion information list

用于帧间预测，由预测单元的运动信息构成。

3.40

亮度　luma

Y表示亮度信号的样值矩阵或单个样值。

3.41

量化参数　quantization parameter

在解码过程对量化系数进行反量化的参数。

3.42

量化系数　quantization coefficient

反量化前变换系数的值。

3.43

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

仿射控制点运动矢量组中的运动矢量个数为三个的仿射预测单元。

3.44

滤波后样本　filtered sample

经去块效应滤波得到的样本。

3.45

P图像　P picture

帧间预测中只使用显示顺序上过去的参考图像进行解码的图像。

3.46

偏移后样本　offseted sample

经样值偏移补偿得到的样本。

3.47

片 patch

按光栅扫描顺序排列的相邻若干最大编码单元。

3.48

起始码 start code

长度为32位的二进制码字，其形式在整个位流中是唯一的。

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。

3.49

RL图像 RL picture

只使用知识图像作为参考图像进行帧间预测解码的图像。

3.50

色度 chroma

Cr和Cb两种色差信号中任一种的样值矩阵或单个样值。

3.51

视频序列 sequence

编码位流的最高层语法结构，包括一个或多个连续的编码图像。

3.52

输出顺序 output order

输出解码图像的顺序，与显示顺序相同。

3.53

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

仿射控制点运动矢量组中的运动矢量个数为两个的仿射预测单元。

3.54

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的能力。

3.55

填充位 stuffing bits

编码时插入位流中的位串，在解码时被丢弃。

3.56

图像 picture

一幅图像可是一帧或一场。

3.57

图像重排序 picture reordering

若解码顺序和输出顺序不同，对解码图像进行重排序的过程。

3.58

位串 bit string

有限个二进制位的有序序列，其最左边位是最高有效位（MSB），最右边位是最低有效位（LSB）。

3.59

位流 bitstream

编码图像所形成的二进制数据流。

3.60

位流缓冲区 bitstream buffer

存储位流的缓冲区。

3.61

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。

3.62

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。

3.63

样本 sample

构成图像的基本元素。

3.64

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:v，其中h为水平方向样本个数；v为垂直方向样本个数。

3.65

样值 sample value

样本的幅值。

3.66

游程 run

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

3.67

预测 prediction

预测过程的具体实现。

3.68

预测补偿 prediction compensation

求由语法元素解码得到的样本残差与其对应的预测值之和。

3.69

预测单元 prediction unit

包括一个亮度预测块和对应的色度预测块，由编码单元划分得到。

3.70

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。

3.71

预测划分方式 prediction partition type

编码单元划分为帧内预测块或帧间预测单元的方式。

3.72

预测块 prediction block

一个使用相同预测过程的M×N的样值块，由编码单元划分得到。

3.73

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。

3.74

语法元素 syntax element

位流中的数据单元解析后的结果。

3.75

源 source

编码前视频素材或其某些属性。

3.76

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。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。

3.77

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L1参考索引构成。

3.78

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。

3.79

帧间编码 inter coding

使用帧间预测对编码单元或图像进行编码。

3.80

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。

3.81

帧内编码 intra coding

使用帧内预测对编码单元或图像进行编码。

3.82

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。

3.83

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。

3.84

知识图像输入端口 library picture input port

主位流解码器的一个端口，用于从主位流解码器外部输入主位流解码所需的知识图像。

3.85

知识图像索引输出端口 library index output port

主位流解码器的一个端口，用于向主位流解码器外部输出主位流解码所需的知识图像的索引号。

3.86

知识位流 library stream

包含知识图像的位流。

3.87

知识位流解码器 library stream decoder

解码知识位流的解码器。

3.88

主位流　main stream

可参考由该位流以外的信息提供的知识图像进行解码的位流。

3.89

主位流解码器　main stream decoder

解码主位流的解码器。该解码器应包含知识图像索引输出端口和知识图像输入端口。

3.90

字节　byte

8位的位串。

3.91

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。

3.92

最大编码单元　largest coding unit

包括一个L×L的亮度样值块和对应的色度样值块，由图像划分得到。

3.93

最大编码块　largest coding block

一个K×K的样值块，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。

4　缩略语

下列缩略语适用于本文件。

AEC	高级熵编码（Advanced Entropy Code）
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）
BBV	位流参考缓冲区管理（Bitstream Buffer Verifier）
CBR	恒定位率（Constant Bit Rate）
CB	编码块（Coding Block）
CU	编码单元（Coding Unit）
CUT	编码树（Coding Unit Tree）
DPB	解码图像缓冲区（Decoded Picture Buffer）
DT	衍生模式树（Derived Tree）
HMVP	基于历史信息的运动矢量预测（History-based Motion Vector Prediction）
LCB	最大编码块（Largest Coding Block）
LCU	最大编码单元（Largest Coding Unit）
LSB	最低有效位（Least Significant Bit）
MSB	最高有效位（Most Significant Bit）
PB	预测块（Prediction Block）
PBT	基于位置的变换（Position Based Transform）
PU	预测单元（Prediction Unit）
ROI	感兴趣区域（Region of Interesting）
SAO	样值偏移补偿（Sample Adaptive Offset）
TB	变换块（Transform Block）

UMVE　　高级运动矢量表达（Ultimate Motion Vector Expression）

5　约定

5.1　概述

本部分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
$\lceil \cdot \rceil$	上取整

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

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

数学函数定义见式（1）～式（11）。

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

式中：

x ——自变量x。

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

式中：

x ——自变量x。

$$\mathrm{Clip1}(x)=\mathrm{Clip3}(0,2^{BitDepth}-1,x) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3)$$

式中：

x ——自变量x；

$BitDepth$ ——编码样本精度。

$$\mathrm{Clip3}(i,j,x)=\begin{cases} i & ; \quad x<i \\ j & ; \quad x>j \\ x & ; \quad \text{其他} \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4)$$

式中：

x ——自变量x；

i ——下界；

j ——上界。

$$\mathrm{Median}(x,y,z)=x+y+z-\mathrm{Min}(x,\mathrm{Min}(y,z))-\mathrm{Max}(x,\mathrm{Max}(y,z)) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (5)$$

式中：

x ——自变量x；

y ——自变量y；

z ——自变量z。

$$\mathrm{Min}(x,y)=\begin{cases} x & ; \quad x<=y \\ y & ; \quad x>y \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (6)$$

式中：

x ——自变量x；

y ——自变量y。

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

式中：

x ——自变量x；

y ——自变量y。

$$\mathrm{Sign}(x)=\begin{cases} 1 & ; \quad x>=0 \\ -1 & ; \quad x<0 \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (8)$$

式中：

x ——自变量x。

$$\mathrm{Log}(x)=\log_2 x \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (9)$$

式中：

x ——自变量x。

$$\mathrm{Ln}(x)=\log_e x \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \quad (10)$$

式中：

x ——自变量x；

e ——自然对数的底，其值为2.718281828…。

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

式中：

x ——自变量x；

s ——自变量s。

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 位流语法的描述方法

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小节内使用。

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部分的位流语法的超集，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conditon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	

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。函数定义见表8。

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。函数定义见表9。

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( )) {	
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。函数定义见表10。

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，见表11。

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。
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

本部分定义的位流语法中，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部分的扩展。这些值不应出现在符合本部分的位流中。

“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部分的位流中。

“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部分的扩展，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 概述

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

本部分支持两种序列：逐行序列和隔行序列。

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。对隔行序列而言，每帧的两场之间存在着一个场时间间隔。对逐行序列而言，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。

序列头可在位流中重复出现，称为重复序列头。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以下范围内：该I图像、显示顺序在这幅I图像之后的图像。

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以下范围内：该RL图像、显示顺序在该RL图像之后的图像、该RL图像所参考的知识图像。

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部分。

6.2 图像

6.2.1 概述

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。

亮度样本位置见图1。

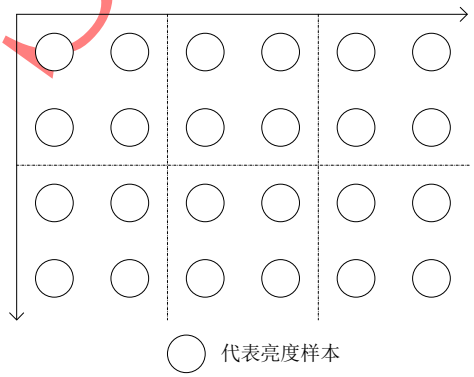


图1 4:0:0 格式下亮度样本位置

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。另外，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见图2。

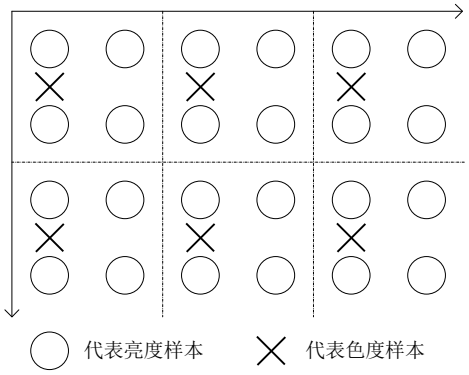


图2　4:2:0 格式下亮度和色度样本位置

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见图3。

图3　4:2:2 格式下亮度和色度样本位置

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见图4。

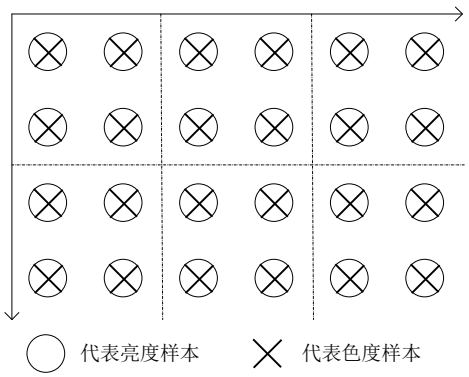


图4　4:4:4 格式下亮度和色度样本位置

6.2.3 图像类型

本部分定义了三种解码图像：

a) I 图像；

b) P 图像；

c) B 图像。

6.2.4 图像间的顺序

如果视频序列中没有B图像，解码顺序与显示顺序相同。如果视频序列中包含B图像，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。

下面举例说明图像重排序。

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。

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。片结构见图5。

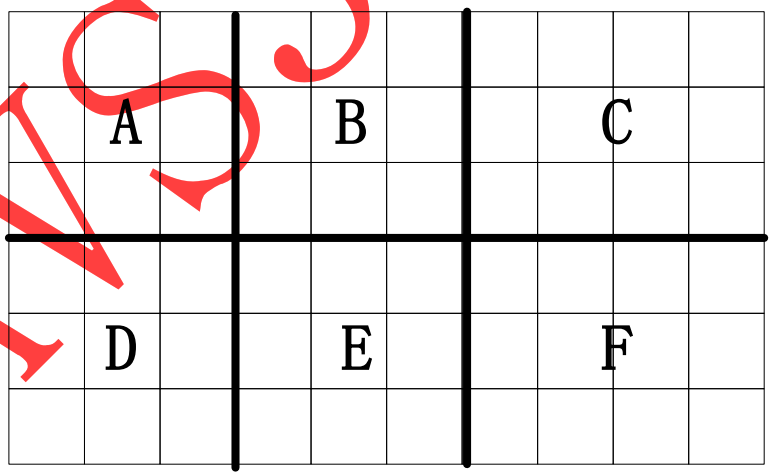


图5 片结构

6.4 最大编码单元、编码树和编码单元

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。在符合本部分的位流中，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，见表12。

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。符合本部分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

视频序列定义见表13。

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( )	
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

序列头定义见表14。

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) {	

表 14（续）

序列头定义	描述符
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) {	
encoding_precision	u(3)
}	
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_minus1	u(4)
rpl1_idx_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++) {	

表 14（续）

序列头定义	描述符
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)	
{	
load_seq_weight_quant_data_flag	u(1)
if (load_seq_weight_quant_data_flag == '1') {	
weight_quant_matrix( )	
}	
}	
secondary_transform_enable_flag	u(1)
sample_adaptive_offset_enable_flag	u(1)
adaptive_leveling_filter_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 (num_of_hmvp_cand && AmvrEnableFlag) {	
emvr_enable_flag	u(1)
}	
ipf_enable_flag	u(1)
tscpm_enable_flag	u(1)

表 14（续）

序列头定义	描述符
marker_bit	f(1)
dt_enable_flag	u(1)
if (DtEnableFlag) {	
log2_max_dt_size_minus4	u(2)
}	
pbt_enable_flag	u(1)
if (low_delay == '0')	
output_reorder_delay	u(5)
cross_patch_loopfilter_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见表15。

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) {	

表 15（续）

参考图像队列配置集定义	描述符
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

自定义加权量化矩阵定义见表16。

表16 自定义加权量化矩阵定义

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 扩展和用户数据语法

扩展和用户数据定义、扩展数据定义、用户数据定义分别见表17、表18、表19。

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( )	
}	
}	

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) == '0110')　/* 内容加密扩展 */	
cei_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 {	
while (next_bits(24) != '0000 0000 0000 0000 0000 0001')	
reserved_extension_data_byte	u(8)
}	
}	
}	
}	

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

序列显示扩展定义见表20。

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见表21。

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

版权扩展定义见表22。

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　内容加密扩展定义

内容加密扩展定义见表23。

表23　内容加密扩展定义

内容加密扩展定义	描述符
cei_extension( ) {	
extension_id	f(4)
content_encryption_algorithm	u(8)
content_encryption_method	u(8)
original_or_copy	u(1)
marker_bit	f(1)
cek_id_len	u(8)
marker_bit	f(1)
cek_id_number_1	u(18)
marker_bit	f(1)
cek_id_number_2	u(22)
marker_bit	f(1)
cek_id_number_3	u(22)
marker_bit	f(1)

表 23（续）

内容加密扩展定义	描述符
cek_id_number_4	u(22)
marker_bit	f(1)
cek_id_number_5	u(22)
marker_bit	f(1)
cek_id_number_6	u(22)
marker_bit	f(1)
iv_len	u(8)
marker_bit	f(1)
iv_number_1	u(18)
marker_bit	f(1)
iv_number_2	u(22)
marker_bit	f(1)
iv_number_3	u(22)
marker_bit	f(1)
iv_number_4	u(22)
marker_bit	f(1)
iv_number_5	u(22)
marker_bit	f(1)
iv_number_6	u(22)
marker_bit	f(1)
reserved_bits	r(5)
next_start_code()	
}	

7.1.2.10 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见表24。

表24 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)
extension_id	f(4)
while (next_bits(24) != '0000 0000 0000 0000 0000 0001') {	
extension_data_byte	u(8)
}	
next_start_code()	
}	

7.1.2.11 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见表25。

表25 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)
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( )	
}	

7.1.2.12 摄像机参数扩展定义

摄像机参数扩展定义见表26。

表26 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)

表 26（续）

摄像机参数扩展定义	描述符
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)
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( )	
}	

7.1.2.13 感兴趣区域参数扩展定义

感兴趣区域参数扩展定义见表27。

表27 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)
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)

表 27（续）

感兴趣区域参数扩展定义	描述符
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)
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( )	
}	

7.1.2.14 参考知识图像扩展定义

参考知识图像扩展定义见表28。

表28　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( )	
}	

7.1.3　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见表29。

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)

表 29（续）

帧内预测图像头定义	描述符
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_idx[0]	ue(v)
}	
}	
else {	
reference_picture_list_set(0, NumRefPicListSet[0])	
}	
if (Rpl1IdxExistFlag)	
ref_pic_list_set_flag[1]	u(1)
if (RefPicListSetFlag[1]) {	
if (Rpl1IdxExistFlag && NumRefPicListSet[1] > 1) {	
ref_pic_list_set_idx[1]	ue(v)
}	
}	
else {	
reference_picture_list_set(1, NumRefPicListSet[1])	
}	
fixed_picture_qp_flag	u(1)
picture_qp	u(7)
loop_filter_disable_flag	u(1)
if (loop_filter_disable_flag == '0') {	
loop_filter_parameter_flag	u(1)
if (loop_filter_parameter_flag) {	

表 29（续）

帧内预测图像头定义	描述符
alpha_c_offset	se(v)
beta_offset	se(v)
}	
}	
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) {	
pic_weight_quant_enable_flag	u(1)
if (PicWeightQuantEnableFlag) {	
pic_weight_quant_data_index	u(2)
if (pic_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)
}	
}	
}	
else if (pic_weight_quant_data_index == '10') {	
weight_quant_matrix( )	
}	
}	
}	
if (AlfEnableFlag) {	
for (compIdx=0; compIdx<3; compIdx++) {	

表 29（续）

帧内预测图像头定义	描述符
picture_alf_enable_flag[compIdx]	u(1)
}	
if (PictureAlfEnableFlag[0] == 1 || PictureAlfEnableFlag[1] == 1 || PictureAlfEnableFlag[2] == 1) {	
alf_parameter_set( )	
}	
}	
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见表30。

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)
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)

表 30（续）

帧间预测图像头定义	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_idx[0]	ue(v)
}	
}	
else {	
reference_picture_list_set(0, NumRefPicListSet[0])	
}	
if (Rpl1IdxExistFlag)	
ref_pic_list_set_flag[1]	u(1)
if (RefPicListSetFlag[1]) {	
if (Rpl1IdxExistFlag && NumRefPicListSet[1] > 1) {	
ref_pic_list_set_idx[1]	ue(v)
}	
}	
else {	
reference_picture_list_set(1, NumRefPicListSet[1])	
}	
num_ref_active_override_flag	u(1)
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 (! ((picture_coding_type == '10') && (PictureStructure == 1)))	
reserved_bits	r(1)
loop_filter_disable_flag	u(1)
if (loop_filter_disable_flag == '0') {	
loop_filter_parameter_flag	u(1)
if (loop_filter_parameter_flag == '1') {	

表 30（续）

帧间预测图像头定义	描述符
alpha_c_offset	se(v)
beta_offset	se(v)
}	
}	
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) {	
pic_weight_quant_enable_flag	u(1)
if (PicWeightQuantEnableFlag) {	
pic_weight_quant_data_index	u(2)
if (pic_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)
}	
}	
}	
else if (pic_weight_quant_data_index == '10') {	
weight_quant_matrix( )	
}	
}	
}	
if (AlfEnableFlag) {	
for (compIdx=0; compIdx<3; compIdx++) {	

表 30（续）

帧间预测图像头定义	描述符
picture_alf_enable_flag[compIdx]	u(1)
}	
if (PictureAlfEnableFlag[0] == 1 || PictureAlfEnableFlag[1] == 1 || PictureAlfEnableFlag[2] == 1) {	
alf_parameter_set( )	
}	
}	
if (AffineEnableFlag) {	
affine_subblock_size_flag	u(1)
}	
next_start_code( )	
}	

7.1.3.3 图像显示扩展定义

图像显示扩展定义见表31。

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)
picture_centre_vertical_offset	i(16)
marker_bit	f(1)
}	
next_start_code( )	
}	

7.1.3.4 图像数据定义

图像数据定义见表32。

表32 图像数据定义

图像数据定义	描述符
picture_data( ) {	

表 32（续）

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

7.1.4 片定义

片定义见表33。

表33 片定义

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

表 33（续）

片定义	描述符
if (PatchSaoEnableFlag[compIdx]) {	
sao_mode[compIdx]	ae(v)
if (SaoMode[compIdx] == 'SAO_Interval') {	
for (j=0; j<MaxOffsetNumber; j++) {	
sao_interval_offset_abs[compIdx][j]	ae(v)
if (SaoIntervalOffsetAbs[compIdx][j])	
sao_interval_offset_sign[compIdx][j]	ae(v)
}	
sao_interval_start_pos[compIdx]	ae(v)
sao_interval_delta_pos_minus2[compIdx]	ae(v)
}	
if (SaoMode[compIdx] == 'SAO_Edge') {	
for (j=0; j<MaxOffsetNumber; j++)	
sao_edge_offset[compIdx][j]	ae(v)
sao_edge_type[compIdx]	ae(v)
}	
}	
}	
}	
}	
for (compIdx=0; compIdx<3; compIdx++) {	
if (PictureAlfEnableFlag[compIdx] == 1)	
alf_lcu_enable_flag[compIdx][LcuIndex]	ae(v)
}	
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	ae(v)
} while (! is_end_of_patch( ))	
next_start_code( )	
patch_end_code	f(32)
}	

7.1.5 编码树定义

编码树定义见表34。

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	
}	

表 34（续）

编码树定义	描述符
else if (split >= MaxSplitTimes) {	
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) {	

表 34（续）

编码树定义	描述符
if (allowSplitBt && allowSplitEqt)	
bet_split_type_flag	ae(v)
if ((! BetSplitTypeFlag && allowSplitBtHor && allowSplitBtVer) || (BetSplitTypeFlag && allowSplitEqtHor && allowSplitEqtVer))	
bet_split_dir_flag	ae(v)
}	
}	
if ((PictureType != 0) && ((((BetSplitFlag && ! BetSplitTypeFlag) || QtSplitFlag) && (width * height == 64)) || (BetSplitTypeFlag && (width * height == 128)))) {	
root_cu_mode	ae(v)
modeChild = root_cu_mode ? 'PRED_Intra_Only' : 'PRED_Inter_Only'	
}	
else {	
modeChild = mode	
}	
if (ChildSizeOccur4) {	
if (Component == 0) {	
LumaWidth = width	
LumaHeight = height	
Component = 1	
}	
}	
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')	

表 34（续）

编码树定义	描述符
Component = 0	
}	
}	
else if (BlockSplitMode == 'SPLIT_BT_VER') {	
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	
}	
}	

表 34（续）

编码树定义	描述符
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)	
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

编码单元定义见表35。

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) {	
if (component == 'COMPONENT_Chroma') {	
if ((priorCuMode == 1) && (chroma_format != '00'))	
intra_chroma_pred_mode	ae(v)
NumOfTransBlocks = 3	
ctp_y[0] = 0	
CuCtp += ctp_y[0]	

表 35（续）

编码单元定义	描述符
if (IntraChromaPredMode != 'Intra_Chroma_PCM') {	
ctp_u	ae(v)
CuCtp += (ctp_u << 1)	
ctp_v	ae(v)
CuCtp += (ctp_v << 2)	
}	
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], component)	
}	
}	
else {	
if (PictureType != 0) {	
if (mode != 'PRED_Intra_Only') {	
skip_flag	ae(v)
}	
if (SkipFlag) {	
if (UmveEnableFlag)	
umve_flag	ae(v)
if (AffineEnableFlag && ! UmveFlag && (width >= 16) && (height >= 16))	
affine_flag	ae(v)
}	
if (! SkipFlag) {	
if (mode != 'PRED_Intra_Only') {	
direct_flag	ae(v)
}	
if (DirectFlag) {	
if (UmveEnableFlag)	
umve_flag	ae(v)
if (AffineEnableFlag && ! UmveFlag && (width >= 16) && (height >= 16))	
affine_flag	ae(v)
}	
if (! DirectFlag && (mode == 'PRED_No_Constraint'))	

表 35（续）

编码单元定义	描述符
intra_cu_flag	ae(v)
}	
}	
PartSize = 'SIZE_2Mx2N'	
if (DtEnableFlag && IntraCuFlag) {	
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) {	
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)
}	
}	
}	
}	
}	

表 35（续）

编码单元定义	描述符
if (UmveFlag) {	
umve_mv_idx	ae(v)
umve_step_idx	ae(v)
umve_dir_idx	ae(v)
}	
else if ((SkipFlag || DirectFlag) && AffineFlag) {	
cu_affine_cand_idx	ae(v)
}	
else if (SkipFlag || DirectFlag) {	
cu_subtype_index	ae(v)
}	
if (! SkipFlag && ! DirectFlag) {	
if (! IntraCuFlag) {	
if (AffineEnableFlag && (width >= 16) && (height >= 16))	
affine_flag	
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	ae(v)
}	
if (SmvdEnableFlag && SmvdApplyFlag && ! AffineFlag && (InterPredRefMode == 2) && ! ExtendMvrFlag) {	
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)	

表 35（续）

编码单元定义	描述符
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)
}	
}	
}	
else {	
TuOrder = 0	
for (i=0; i<NumOfIntraPredBlock; i++) {	

表 35（续）

编码单元定义	描述符
intra_luma_pred_mode	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	ae(v)
IsPcmMode[TuOrder+1] = (IntraChromaPredMode == 'Intra_Chroma_PCM')	
IsPcmMode[TuOrder+2] = (IntraChromaPredMode == 'Intra_Chroma_PCM')	
}	
if (IpfEnableFlag && (PartSize == 'SIZE_2Mx2N') && (! IsPcmMode[0])) {	
ipf_flag	ae(v)
}	
}	
}	
if (! IntraCuFlag && ! SkipFlag) {	
if (! DirectFlag && component == 'COMPONENT_LUMACHROMA')	
ctp_zero_flag	ae(v)
CuCtp = 0	
if (! CtpZeroFlag) {	
if (PbtEnableFlag && (width / height < 4) && (height / width < 4) && (width >= 8) && (width <= 32) && (height >= 8) && (height <= 32)) {	
pbt_cu_flag	ae(v)
}	
if (! PbtCuFlag) {	
if (component == 'COMPONENT_LUMACHROMA') {	
ctp_u	ae(v)

表 35（续）

编码单元定义	描述符
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]	
}	
}	
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) {	
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 ((component == 'COMPONENT_LUMACHROMA') && (IntraChromaPredMode != 'Intra_Chroma_PCM')) {	
ctp_u	ae(v)
ctp_v	ae(v)
}	
CuCtp += (ctp_u << (NumOfTransBlocks–2))	
CuCtp += (ctp_v << (NumOfTransBlocks–1))	
}	
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)))	
}	
IsChroma = 0	
if (i == NumOfTransBlocks -1 || i == NumOfTransBlocks -2) {	
IsChroma = 1	
}	
block(i, blockWidth, blockHeight, CuCtp, IsChroma, IsPcmMode[i], component)	
}	
}	
}	

7.1.7 变换块定义

变换块定义见表36。

表36 变换块定义

变换块定义	描述符
block(i, blockWidth, blockHeight, CuCtp, isChroma, isPcm, component) {	
M_1 = blockWidth	
M_2 = blockHeight	
for (x=0; x<M_1; x++) {	

表 36（续）

变换块定义	描述符
for (y=0; y<M2; y++)	
QuantCoeffMatrix[x][y] = 0	
}	
if (! isPcm) {	
if (CuCtp & (1 << i)) {	
blockWidth = isChroma ? blockWidth / 2 : blockWidth	
blockHeight = isChroma ? blockHeight / 2 : blockHeight	
idxW = Log(blockWidth) – 1	
idxH = Log(blockHeight) – 1	
NumOf Coeff = blockWidth * blockHeight	
ScanPosOffset = 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[idxW][idxH][ScanPosOffset][0]	
PosyInBlk = InvScanCoeffInBlk[idxW][idxH][ScanPosOffset][1]	
QuantCoeffMatrix[PosxInBlk][PosyInBlk] = coeff_sign ? –AbsLevel : AbsLevel	
if (ScanPosOffset >= NumOf Coeff – 1) {	
break	
}	
coeff_last	ae(v)
ScanPosOffset = ScanPosOffset + 1	
} while (! coeff_last)	
}	
}	
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见表37。

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 (PictureAlfEnableFlag[0] == 1) {	
alf_filter_num_minus1	ue(v)
for (i=0; i<alf_filter_num_minus1+1; i++) {	
if ((i > 0) && (alf_filter_num_minus1 != 15))	
alf_region_distance[i]	ue(v)
for (j=0; j<9; j++)	
alf_coeff_luma[i][j]	se(v)
}	
}	
if (PictureAlfEnableFlag[1] == 1) {	
for (j=0; j<9; j++)	

表 37（续）

自适应修正滤波参数定义	描述符
alf_coeff_chroma[0][j]	se(v)
}	
if (PictureAlfEnableFlag[2] == 1) {	
for (j=0; j<9; j++)	
alf_coeff_chroma[1][j]	se(v)
}	
}	

7.2 语义描述

7.2.1 视频扩展

本部分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，见表38。

表38 视频扩展标号

视频扩展标号	含义
0000	保留
0001	保留
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	内容加密扩展
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
1110～1111	保留

7.2.2 视频序列语义描述

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。

档次标号　profile_id

8位无符号整数。表示位流符合的档次。

级别标号　level_id

8位无符号整数。表示位流符合的级别。

档次和级别见附录B。

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’表示当前主位流所参考的知识位流的序列头中除library_stream_flag外的所有语法元素的值应与当前主位流的序列头中对应语法元素的值相同；值为‘0’表示当前主位流所参考的知识位流的序列头中除library_stream_flag外的其它语法元素的值可与当前主位流的序列头中对应语法元素的值不同。

逐行序列标志　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。

场图像序列标志　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计算分别如式（12）和式（13）所示：

$$PictureWidthInMinBu = (horizontal\_size + MiniSize - 1) / MiniSize \quad (12)$$

$$PictureWidthInMinCu = (PictureWidthInMinBu \times MiniSize + MiniSize - 1) / MinCuSize \quad (13)$$

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计算分别如式（14）和式（15）所示：

$$PictureHeightInMinBu = 2 \times ((vertical\_size + 2 \times MiniSize - 1) / (2 \times MiniSize)) \quad \cdots(14)$$

$$PictureHeightInMinCu = 2 \times ((PictureHeightInMinBu \times MiniSize + 2 \times MiniSize - 1) / (2 \times MinCuSize)) \quad \cdots(15)$$

在其他情况下，PictureHeightInMinBu和PictureHeightInMinCu计算分别如式（16）和式（17）所示：

$$PictureHeightInMinBu = (vertical\_size + MiniSize - 1) / MiniSize \quad \cdots(16)$$

$$PictureHeightInMinCu = (vertical\_size + MinCuSize - 1) / MinCuSize \quad \cdots(17)$$

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

MiniSize的值由档次规定，见附录B。

horizontal_size、vertical_size与图像边界的关系见图6。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。

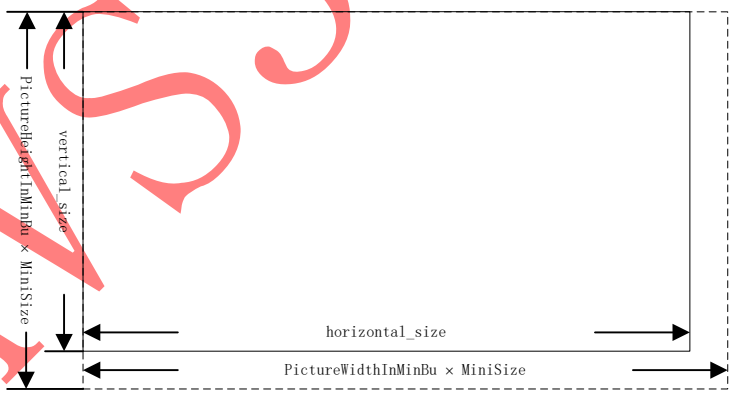


图6　图像边界示意图

色度格式　chroma_format

2位无符号整数。规定色度分量的格式，见表39。

表39　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，见表40。如果sample_precision的值为‘001’，SamplePrecision的值为8；如果sample_precision的值为‘010’，SamplePrecision的值为10。

表40　样本精度

sample_precision的值	含义
000	禁止
001	亮度和色度均为8 bit精度
010	亮度和色度均为10 bit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，见表41。如果encoding_precision的值为‘001’，BitDepth的值为8；如果encoding_precision的值为‘010’，BitDepth的值为10。如果位流中不存在encoding_precision，则BitDepth的值为8。

表41　编码样本精度

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

宽高比　aspect_ratio

4位无符号整数。规定重建图像的样本宽高比（SAR）或显示宽高比（DAR）。见表42。

表42 宽高比

aspect_ratio的值	样本宽高比（SAR）	显示宽高比（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。样本宽高比按式（18）计算。

$$SAR = DAR \times vertical\_size \div horizontal\_size \quad (18)$$

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，样本宽高比按式（19）计算。

$$SAR = DAR \times display\_vertical\_size \div display\_horizontal\_size \quad (19)$$

帧率代码 frame_rate_code

4位无符号整数。规定帧率，见表43。

表43 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...)
0101	30
0110	50
0111	60000÷1001 (59.94...)
1000	60
1001	100
1010	120
1011	200
1100	240
1101	300
1110～1111	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。

位率低位 bit_rate_lower

BitRate的低18位。

位率高位　bit_rate_upper

BitRate的高12位。BirRate按式（20）计算。

$$BitRate = (bit\_rate\_upper << 18) + bit\_rate\_lower \quad (20)$$

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

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”（见附录C）；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”（见附录C）。

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（见附录C）。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），见式（21）：

$$BBS = 16 \times 1024 \times bbv\_buffer\_size \quad (21)$$

最大解码图像缓冲区大小 max_dpb_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（具体由级别规定，见附录B）。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_idx_exist_flag

二值变量。值为‘0’表示位流中不应出现ref_pic_list_flag[1]和ref_pic_list_idx[1]；值为‘1’表示可出现。Rpl1IdxExistFlag的值等于rpl1_idx_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, rplsIdx)；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,rplsIdx)中每个语法元素的值与参考图像队列配置集 reference_picture_list_set(0,rplsIdx)中对应语法元素的值相同，其中rplsIdx的取值范围是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。取值范围是0～64。NumRefPicListSet[0]的值等于num_ref_pic_list_set[0]值。如果Rpl1SameAsRpl0Flag的值为1，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]

表示解码当前图像时，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。取值范围是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，取值范围是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，取值范围是0～2。MinCuSize、MinBtSize和MinEqtSize的值均等于$2^{log2_min_cu_size_minus2+2}$。

划分单元最大比例　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，取值范围是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，取值范围是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，取值范围是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，LoadSeqWeightQuantDataFlag的值等于0。

二次变换允许标志　secondary_transform_enable_flag

二值变量。值为‘1’表示可使用二次变换；值为‘0’表示不应使用二次变换。SecondaryTransformEnableFlag的值等于secondary_transform_enable_flag的值。

样值偏移补偿允许标志　sample_adaptive_offset_enable_flag

二值变量。值为‘1’表示可使用样值偏移补偿；值为‘0’表示不应使用样值偏移补偿。SaoEnableFlag的值等于sample_adaptive_offset_enable_flag的值。

自适应修正滤波允许标志　adaptive_leveling_filter_enable_flag

二值变量。值为‘1’表示可使用自适应修正滤波；值为‘0’表示不应使用自适应修正滤波。AlfEnableFlag的值等于adaptive_leveling_filter_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。

对称运动矢量差模式允许标志　smvd_enable_flag

二值变量。值为‘1’表示可使用对称运动矢量差模式；；值为‘0’表示不应使用对称运动矢量差模式。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’表示可使用自适应运动矢量精度；值为‘0’表示不应使用自适应运动矢量精度。AmvrEnableFlag的值等于amvr_enable_flag的值。

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，取值范围是0~8。NumOfHmvpCand的值为0表示不应使用基于历史信息的运动矢量预测。

帧内预测滤波允许标志　ipf_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。IpfEnableFlag的值等于ipf_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’表示可使用高级运动矢量表达模式；值为‘0’表示不应使用高级运动矢量表达模式。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’表示可使用运动矢量精度扩展模式；值为‘0’表示不应使用运动矢量精度扩展模式。EmvrEnableFlag的值等于emvr_enable_flag的值。

色度两步预测模式允许标志　tscpm_enable_flag

二值变量。值为‘1’表示可使用色度两步预测模式；值为‘0’表示不应使用色度两步预测模式。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。取值范围是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’表示可使用基于位置的变换。值为‘0’表示不应使用基于位置的变换。PbtEnableFlag的值等于pbt_enable_flag的值。

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_loopfilter_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。

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。

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。取值范围是 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。取值范围是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Rpl1SameAsRpl0Flag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 的取值范围是 0～(NumRefPicListSet[0]-1)；

——将 DeltaDoi[1][rpls][i]的值设置为 DeltaDoi[0][rpls][i]的值，rpls 的取值范围是 0～(NumRefPicListSet[0]-1)，i 的取值范围是 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,取值范围应是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

本部分不定义显示过程。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。说明视频在按本部分进行编码之前的格式，见表44。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44　视频格式

video_format的值	含义
000	分量信号
001	PAL
010	NTSC
011	SECAM
100	MAC
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，见表45。

表45 彩色三基色

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

表 45（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。 [b]ITU R 建议 BT.470 System M。 [c]ITU R 建议 BT.470 System B, G。 [d]SMPTE 170M。 [e]SMPTE 240M。 [f]GY/T 307-2017。	

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，见表46。表46中Lc是线性输入信号，与光的强度成正比。

表46　光电转移特性

transfer_characteristics的值	光电转移特性
0	禁止
1	E′ = 1.099 $Lc^{0.45}$ − 0.099，1.33 >= Lc >= 0.018 E′ = 4.500 Lc，0.018 > Lc >= −0.0045 E′ = −{1.099(−4Lc)$^{0.45}$−0.099}/4，−0.0045 > Lc >= −0.25。[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 $Lc^{0.45}$ − 0.099，1 >= Lc >= 0.018 E′ = 4.500 Lc，0.018 > Lc >= 0。[d]
7	E′ = 1.1115 $Lc^{0.45}$ − 0.1115，Lc >= 0.0228 E′= 4.0 Lc，0.0228 > Lc。[e]
8	线性转移特性 即 E′ = Lc
9	对数转移特性（范围100:1） E′ = 1.0−($\log_{10}$(Lc))/2，1 >= Lc >= 0.01 E′ = 0.0，0.01> Lc

表 46（续）

transfer_characteristics的值	光电转移特性
10	对数转移特性（范围316.22777:1） $E' = 1.0-(\log_{10}(Lc))/2.5$，$1 >= Lc >= 0.0031622777$ $E' = 0.0$，$0.0031622777 > Lc$
11	$E' = 1.0993\ Lc^{0.45} - 0.0993$，$1 >= Lc >= 0.0181$ $E' = 4.500\ Lc$，$0.0181 > Lc >= 0$。[f]
12	$E' = E' = ((c_1 + c_2 \times Lc^n) \div (1 + c_3 \times Lc^n))^m$ $c_1 = c_3 - c_2 + 1 = 3424 \div 4096 = 0.8359375$ $c_2 = 32 \times 2413 \div 4096 = 18.8515625$ $c_3 = 32 \times 2392 \div 4096 = 18.6875$ $m = 128 \times 2523 \div 4096 = 78.84375$ $n = 0.25 \times 2610 \div 4096 = 0.1593017578125$ Lc等于1对应于显示亮度为10000 cd/m²。[g]
13	保留
14	$E' = (3 \times Lc)^{0.5}$，$1/12 >= Lc >= 0$ $E' = a \times Ln(12 \times Lc - b) + c$，$1 >= Lc > 1/12$ $a = 0.17883277$ $b = 0.28466892$ $c = 0.55991073$。[f]
15～255	保留
[a]GY/T 155-2000。 [b]ITU R 建议 BT.470 System M。 [c]ITU R 建议 BT.470 System B, G。 [d]ITU R 建议 BT.709, SMPTE 170M, GY/T 307-2017中规定的10位系统。 [e]SMPTE 240M。 [f]GY/T 307-2017中规定的12位系统。 [g]GY/T 315—2018中规定的光电转移特性。	

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　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，见表47。

表47　彩色信号转换矩阵

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]
2	未作规定的视频 图像特性未知
3	保留

表 47（续）

matrix_coefficients的值	彩色信号转换矩阵
4	$E'_Y = 0.59\ E'_G + 0.11\ E'_B + 0.30\ E'_R$ $E'_{CB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$ $E'_{CR} = -0.421E'_G - 0.079\ E'_B + 0.500\ E'_R$。[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
[a]GY/T 155-2000。 [b]FCC。 [c]ITU R 建议 BT.470 System B, G。 [d]SMPTE 170M。 [e]SMPTE 240M。 [f]GY/T 315-2018中规定的非恒定亮度系统。 [g]GY/T 315-2018中规定的恒定亮度系统。	

在表47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；

——Y、Cb 和 Cr 与 E'_Y、E'_{CB} 和 E'_{CR} 的关系如下：

如果sample_range的值为‘0’：

$Y = (219 \times 2^{BitDepth-8} \times E'_Y) + 2^{BitDepth-4}$

$Cb = (224 \times 2^{BitDepth-8} \times E'_{CB}) + 2^{BitDepth-1}$

$Cr = (224 \times 2^{BitDepth-8} \times E'_{CR}) + 2^{BitDepth-1}$

如果sample_range的值为‘1’：

$Y = ((2^{BitDepth} - 1) \times E'_Y)$

$Cb = ((2^{BitDepth} - 1) \times E'_{CB}) + 2^{BitDepth-1}$

$Cr = ((2^{BitDepth} - 1) \times E'_{CR}) + 2^{BitDepth-1}$

其中BitDetph是编码样本精度。例如：

BitDepth = 8，sample_range的值为‘0’时：

$Y = (219 \times E'_Y) + 16$

$Cb = (224 \times E'_{CB}) + 128$

$Cr = (224 \times E'_{CR}) + 128$

Y的取值范围是16～235，Cb和Cr的取值范围是16～240。

BitDepth = 8，sample_range的值为‘1’时：

$Y = (255 \times E'_Y)$

$Cb = (255 \times E'_{CB}) + 128$

$Cr = (255 \times E'_{CR}) + 128$

Y、Cb和Cr的取值范围都是0～255。

注1：本部分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。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。在这种情况下建议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。本部分不定义显示过程。

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方式，见表48。

表48　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

表48（续）

td_packing_mode的值	含义
2	四视点拼接
3～255	保留

表48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43。i表示视频序列中所有图像的最高时间层标识。

时间层位率低位　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。

时间层位率高位　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，见式（22）：

$$CopyrightNumber=(copyright\_number\_1<<44)+(copyright\_number\_2<<22)+copyright\_number\_3\cdots \quad (22)$$

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　内容加密扩展

视频扩展标号　extension_id

位串‘0110’。标识内容加密扩展。

内容加密扩展信息（CEI信息）包括加密算法、CEK_ID、当前初始向量、加密方式等，见表49。

表49　内容加密扩展信息

内容元素	位数	描述
content_encryption_algorithm	8	加密算法模式
content_encryption_method	8	加密方式
CEK_ID_LEN	8	内容密钥ID长度，最长16字节
CEK_ID		内容密钥ID
IV_LEN	8	起始IV长度，最长16字节
IV		起始IV

加密算法模式　content_encryption_algorithm

8位无符号整数。规定加密算法模式，见表50。

表50　加密算法模式

content_encryption_algorithm的值	含义
0	保留
1	AES_128_CBC_NOPAD
2	SM4_128_CBC_NOPAD
3～255	保留

加密方式　content_encryption_method

8位位串，包括以下字段：EncrypType、EncrpyRange和EncrpyStrength，见表51。表中EncrypType、EncrpyRange和EncrpyStrength字段是无符号整数。

表51　加密方式

content_encryption_method的字段	取值	说明	描述符
EncrypType	0～1	0表示帧内预测图像加密，帧间预测图像不加密； 1表示帧内/帧间预测图像均加密	u(1)
EncrypRange	0～1	0表示对片数据加密，patch_start_sode不加密； 1表示对图像数据加密，保留图像内第一个patch_start_sode不加密。	u(1)
EncrypStrength	0～1	0表示10%加密； 1表示全加密	u(1)
保留	0		u(5)

加密内容密钥标识长度　cek_id_len

8位无符号整数。最长为16。

加密内容密钥标识1　cek_id_number_1

18位无符号整数。cek_id_number的第110位到第127位。

加密内容密钥标识2　cek_id_number_2

22位无符号整数。cek_id_number的第88位到第109位。

加密内容密钥标识3　cek_id_number_3

22位无符号整数。cek_id_number的第66位到第87位。

加密内容密钥标识4　cek_id_number_4

22位无符号整数。cek_id_number的第44位到第65位。

加密内容密钥标识5　cek_id_number_5

22位无符号整数。cek_id_number的第22位到第43位。

加密内容密钥标识6　cek_id_number_6

22 位无符号整数。cek_id_number 的第 0 位到第 21 位。

cek_id_number 是可变长的无符号整数，最大位数为 128 位，实际位数为 cek_id_len×8 位，见式（23）：

cek_id_number=((cek_id_number_1<<110)+(cek_id_number_2<<88)+(cek_id_number_3<<66)+(cek_id_number_4<<44)+(cek_id_number_5<<22)+cek_id_number_6)>>((16-cek_id_len)*8)…(23)

加密内容初始向量长度　iv_len

8位无符号整数。加密内容初始向量长度，最长为16。

加密内容初始向量1　iv_number_1

18位无符号整数。iv_number的第110位到第127位。

加密内容初始向量2　iv_number_2

22位无符号整数。iv_number的第88位到第109位。

加密内容初始向量2　iv_number_2

22位无符号整数。iv_number的第66位到第87位。

加密内容初始向量2　iv_number_2

22位无符号整数。iv_number的第44位到第65位。

加密内容初始向量2　iv_number_2

22位无符号整数。iv_number的第22位到第43位。

加密内容初始向量2　iv_number_2

22位无符号整数。iv_number的第0位到第21位。

iv_number为可变长的无符号整数，最大位数为128位，实际位数为iv_len×8位，见式（24）：

$$iv\_number=((iv\_number\_1<<110)+(iv\_number\_2<<88)+(iv\_number\_3<<66)+(iv\_number\_4<<44)+(iv\_number\_5<<22)+iv\_number\_6)>>((16-iv\_len)*8) \quad (24)$$

7.2.2.10　高动态范围图像扩展

视频扩展标号　extension_id

位串‘0101’。标识高动态范围图像扩展。

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’。

7.2.2.11　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CIE 1931（见ISO 11664-1/CIE S 014-1、ISO 11664-3/CIE S 014-3和CIE S 015），以0.00002为单位，范围从0到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CIE 1931（见ISO 11664-1/CIE S 014-1、ISO 11664-3/CIE S 014-3和CIE S 015），以0.00002为单位，范围从0到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，范围从1cd/m^2到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，范围从0.0001cd/m^2到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，范围从1cd/m^2到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，范围从1cd/m^2到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。

7.2.2.12 摄像机参数扩展

本条内容如图7和图8所示。

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，范围从0mm到4,194.303mm。

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，范围从0mm到4,194.303mm。

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，范围从0到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，范围从0° 到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，范围从-2,147,483.648mm到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

一组22位整数，用2的补码表示。说明摄像机的方向，每个值范围从-2,097,152到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

一组22位整数，用2的补码表示。说明摄像机向上的方向，每个值的范围从-2,097,152到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。

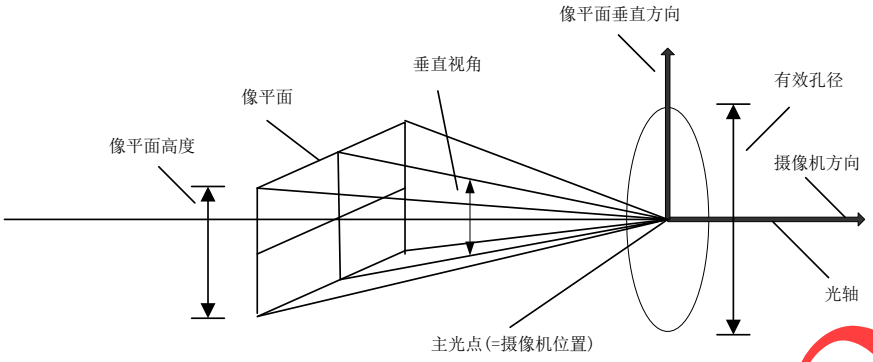


图7　摄像机原理示意图

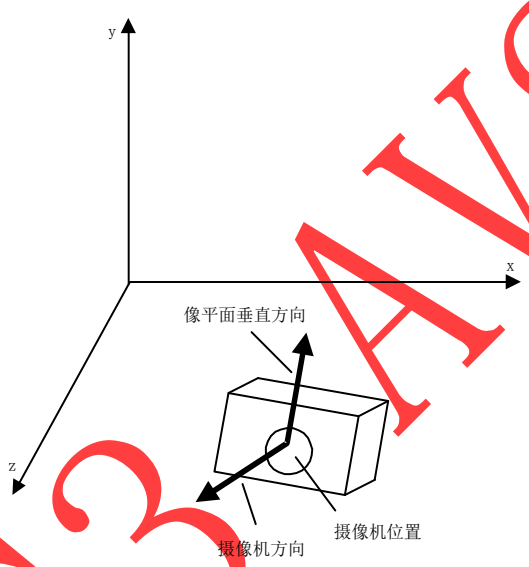


图8　摄像机坐标系统示意图

7.2.2.13　感兴趣区域参数扩展

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感兴趣区域（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感兴趣区域（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，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。

7.2.2.14 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。表示后续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，取值范围是0～511。用于描述后续主位流图像所参考的外部知识图像的索引编号。

7.2.3 图像

7.2.3.1 帧内预测图像头

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27 MHz系统时钟导出的90 kHz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。见附录C。

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，见表52。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52 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。取值范围是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是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，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Rpl1IdxExistFlag的值为0，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_idx[0]、ref_pic_list_set_idx[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，取值范围是0～(NumRefPicListSet[i]-1)（i等于0或1）。

如果当前图像头的位流中不存在ref_pic_list_set_idx[0]，则ref_pic_list_set_idx[0]的值应为‘0’。如果当前图像头的位流中不存在ref_pic_list_set_idx[1]且Rpl1IdxExistFlag的值为0，则ref_pic_list_set_idx[1]的值等于ref_pic_list_set_idx[0]的值。如果当前图像头的位流中不存在ref_pic_list_set_idx[1]且Rpl1IdxExistFlag的值为1，则ref_pic_list_set_idx[1]的值应为‘0’。

如果ref_pic_list_set_flag[i]的值为‘1’，则RplsIdx[i]的值等于ref_pic_list_set_idx[i]的值；否则，RplsIdx[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，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（见附录C）。

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’。

——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，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帧的次数（1次、2次或3次）。

1) 如果repeat_first_field的值是‘0’，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’，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’。

——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　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。

图像量化因子　picture_qp

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

去块滤波禁用标志　loop_filter_disable_flag

二值变量。值为‘1’表示不应使用环路滤波；值为‘0’表示应使用环路滤波。

去块滤波参数标志　loop_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’表示位流中没有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取值范围是-8～8，环路滤波参数AlphaCOffset等于alpha_c_offset。如果位流中没有alpha_c_offset，AlphaCOffset的值等于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取值范围为是-8～8，环路滤波参数BetaOffset等于beta_offset。如果位流中没有beta_offset，BetaOffset的值等于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

色度块量化参数相对于CurrentQp的增量，取值范围-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　pic_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。PicWeightQuantEnableFlag的值等于pic_weight_quant_enable_flag。如果当前图像的图像头位流中不存在pic_weight_quant_enable_flag，则PicWeightQuantEnableFlag的值应为0。

图像加权量化数据加载索引　pic_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，取值范围是-128～127。解析过程见8.2。解码过程见9.2.6。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。

图像自适应修正滤波允许标志　picture_alf_enable_flag[compIdx]

二值变量。值为‘1’表示当前图像的第compIdx个分量可使用自适应修正滤波；值为‘0’则表示当前图像的第compIdx个分量不应使用自适应修正滤波。PictureAlfEnableFlag[compIdx]的值等于picture_alf_enable_flag[compIdx]的值。其中compIdx等于0表示亮度分量，compIdx等于1表示Cb分量，compIdx等于2表示Cr分量。

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 ，其余情况下，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，见表53。

表53　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。取值范围是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，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，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，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][RplsIdx[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，AffineSubblockSizeFlag的值等于0。

7.2.3.3 图像显示扩展

本条不定义显示过程。这一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。

视频扩展标号 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。注意：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9。

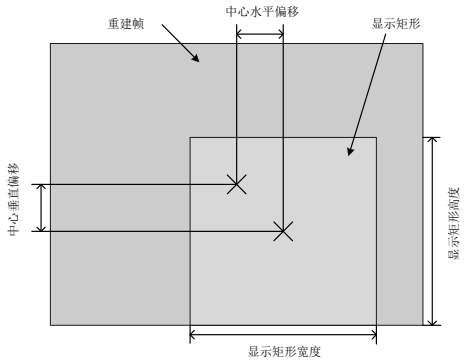


图9　图像中心偏移量参数

注1：显示矩形的尺寸可能大于重建图像。

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

注3：图 9 中，picture_centre_horizontal_offset 和 picture_centre_vertical_offset 均为负值。

7.2.4　片

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，取值范围是0x00～0x7F。

按以下方式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。

片量化因子　patch_qp

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

片样值偏移补偿允许标志　patch_sao_enable_flag[compIdx]

二值变量。PatchSaoEnableFlag[compIdx]的值等于patch_sao_enable_flag[compIdx]的值。PatachSaoEnableFlag[compIdx]的值为1表示在该片内第compIdx个分量可使用样值偏移补偿；值为0则表示在该片内第compIdx个分量不应使用样值偏移补偿。其中compIdx等于0表示亮度分量，compIdx等于1表示Cb分量，compIdx等于2表示Cr分量。如果位流中不存在patch_sao_enable_flag[compIdx]，则PatchSaoEnableFlag[compIdx]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的取值范围应是(-32 - 4×(BitDepth-8))～(32 + 4×(BitDepth-8))。

样值偏移补偿合并方式索引　sao_merge_type_index

样值偏移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补偿单元E的左边样值偏移补偿单元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补偿单元E的上边样值偏移补偿单元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。

样值偏移补偿模式　sao_mode[compIdx]

样值偏移补偿模式的索引值。解析过程见8.3。其中compIdx等于0表示亮度分量，compIdx等于1表示Cb分量，compIdx等于2表示Cr分量。由sao_mode的值查表54得到SaoMode和MaxOffsetNumber的值。

表54　样值偏移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

样值偏移补偿区间模式偏移值绝对值　sao_interval_offset_abs[compIdx][j]

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

样值偏移补偿区间模式偏移值符号值　sao_interval_offset_sign[compIdx][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[compIdx][j]的值为‘0’，则SaoIntervalOffsetSign[compIdx][j]的值等于1；否则SaoIntervalOffsetSign[compIdx][j]的值等于-1。如果位流中不存在sao_interval_offset_sign[compIdx][j]，则SaoIntervalOffsetSign[compIdx][j]的值为0。

样值偏移补偿区间模式起始偏移子区间位置　sao_interval_start_pos[compIdx]

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

样值偏移补偿区间模式起始偏移子区间位置差　sao_interval_delta_pos_minus2[compIdx]

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

样值偏移补偿边缘模式偏移值　sao_edge_offset[compIdx][j]

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

样值偏移补偿边缘模式类型　sao_edge_type[compIdx]

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

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

二值变量。值为‘1’表示第LcuIndex个最大编码单元的分量compIdx样本应使用自适应修正滤波；值为‘0’表示第LcuIndex个最大编码单元的分量compIdx样本不应使用自适应修正滤波。其中compIdx

等于0表示亮度分量，compIdx等于1表示Cb分量，compIdx等于2表示Cr分量。AlfLCUEnableFlag[compIdx][LcuIndex]的值等于alf_lcu_enable_flag[compIdx][LcuIndex]的值。

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

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 root_cu_mode

二值变量。值为‘1’表示当前编码树节点覆盖的编码单元只应使用帧内预测（‘PRED_Intra_Only’）；值为‘0’表示当前编码树节点覆盖的编码单元只应使用帧间预测（‘PRED_Inter_Only’）。如果位流中不存在root_cu_mode，当前编码单元可选择使用的预测模式应与父节点一致。

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，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，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55得到BlockSplitMode。

表55 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

跳过模式标志 skip_flag

二值变量。值为‘1’表示当前编码单元是跳过模式；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的值等于0。

高级运动矢量表达模式标志 umve_flag

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

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，AffineFlag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的值等于0。

帧内编码单元标志 intra_cu_flag

二值变量。值为‘1’表示当前编码单元是帧内预测模式；值为‘0’表示当前编码单元是帧间预测模式。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：

——如果当前图像帧是 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是‘PRED_Intra_Only’，IntraCuFlag 等于 1；

——否则，IntraCuFlag 等于 0。

基础运动矢量索引 umve_mv_idx

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9.5.8。UmveMvIdx的值等于umve_mv_idx的值。UmveMvIdx的值应为0或1。

运动矢量偏移量索引 umve_step_idx

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9.5.8。UmveStepIdx的值等于umve_step_idx的值。

运动矢量方向索引 umve_dir_idx

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9.5.8。UmveDirIdx的值等于umve_dir_idx的值。

仿射运动矢量索引 cu_affine_cand_idx

跳过模式或直接模式下的仿射模式索引值。解析过程见8.3。解码过程见9.5.8。AffineCandIdx的值等于cu_affine_cand_idx的值。如果位流中不存在cu_affine_cand_idx，AffineCandIdx的值等于0。

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。DtSplitFlag的值等于dt_split_flag的值，取值范围为0~4。如果位流中不存在dt_split_flag，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分。解析过程见8.3。DtSplitHqtFlag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_2NxnU’模式划分；值为‘1’表示应进行’SIZE_2NxnD’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

用于确定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9.5.8。AffineAmvrIndex的值等于affine_amvr_index的值。如果位流中不存在affine_amvr_index，则AffineAmvrIndex的值等于0。

自适应运动矢量精度索引 amvr_index

用于确定编码单元的运动矢量精度。解析过程见8.3，解码过程见9.5.8。AmvrIndex的值等于amvr_index的值。如果位流中不存在amvr_index，则AmvrIndex的值等于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。解码过程见9.5.3。CuSubtypeIdx的值等于cu_subtype_index的值。如果位流中不存在cu_subtype_index，CuSubtypeIdx的值等于0。如果当前图像是P图像，cu_subtype_index值的取值范围为0～9。如果当前图像是B图像，cu_subtype_index值的取值范围为0～11。

预测参考模式 inter_pred_ref_mode

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。InterPredRefMode等于inter_pred_ref_mode的值。如果位流中不存在inter_pred_ref_mode，或当前图像为P图像，InterPredRefMode的值等于0。

——如果 InterPredRefMode 的值为 0，预测参考模式为使用队列 0 参考（‘PRED_List0‘），预

测单元运动矢量数等于 1；

——如果 InterPredRefMode 的值为 1，预测参考模式为使用队列 1 参考（‘PRED_List1’），预测单元运动矢量数等于 1；

——如果 InterPredRefMode 的值为 2，预测参考模式为使用双队列参考（‘PRED_List01’），预测单元运动矢量数等于 2。

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。其中，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9.5.8.4。ExtendMvrFlag的值等于extend_mvr_flag的值。

帧内亮度预测模式　intra_luma_pred_mode

用于确定亮度块的帧内预测模式。解析过程见8.3。解码过程见9.5.6。intra_luma_pred_mode的取值范围为0～33。如果NumOfIntraPredBlock的值大于1，IntraLumaPredMode的值不应为33。

帧内色度预测模式　intra_chroma_pred_mode

用于确定4:2:0格式下编码块中PredBlockOrder为NumOfIntraPredBlock、NumOfIntraPredBlock+1的两个色度块的帧内预测模式。解析过程见8.3。解码过程见9.5.6。如果位流中不存在intra_chroma_pred_mode，则IntraChromaPredMode的值等于-1。

如果当前编码单元只包含色度分量，priorCuMode的值由当前色度块对应的亮度块的右下角4×4子块所在的编码单元的编码模式X决定。如果编码模式X是帧内预测，priorCuMode的值等于1；如果编码模式X是帧间预测，priorCuMode的值等于0。

如果当前编码单元只包含色度分量且priorCuMode的值等于1时，在解析当前色度块对应的色度预测模式时，令当前编码单元对应的亮度帧内预测模式为当前色度块对应的亮度块的右下角4×4子块的亮度帧内预测模式。

帧内预测滤波标志　ipf_flag

二值变量。值为‘1’表示应使用帧内预测滤波；值为‘0’表示当不应使用帧内预测滤波。解析过程见8.3。IpfFlag的值等于ipf_flag的值。如果位流中不存在ipf_flag，则IpfFlag的值等于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。解码过程见9.5.7。RefIdxL0的值等于pu_reference_index_l0的值。如果位流中不存在pu_reference_index_l0，则RefIdxL0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。MvDiffXAbsL0等于mv_diff_x_abs_l0的值，MvDiffYAbsL0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。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，则MvDiffXSignL0或MvDiffYSignL0的值为0。如果MvDiffXSignL0的值为0，MvDiffXL0等于MvDiffXAbsL0；如果MvDiffXSignL0的值为1，MvDiffXL0等于-MvDiffXAbsL0。如果MvDiffYSignL0的值为0，MvDiffYL0等于MvDiffYAbsL0；如果MvDiffYSignL0的值为1，MvDiffYL0等于-MvDiffYAbsL0。MvDiffXL0和MvDiffYL0的取值范围是-32768～32767。

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。MvDiffXAbsL0Affine等于mv_diff_x_abs_l0_affine的值，MvDiffYAbsAffineAffine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。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。如果MvDiffXSignL0Affine的值为0，MvDiffXL0Affine等于MvDiffXAbsL0Affine;如果MvDiffXSignL0Affine的值为1,MvDiffXL0Affine等于-MvDiffXAbsL0Affine。如果MvDiffYSignL0Affine的值为0，MvDiffYL0Affine等于MvDiffYAbsL0Affine；如果MvDiffYSignL0Affine的值为1，MvDiffYL0Affine等于-MvDiffYAbsL0Affine。MvDiffXL0Affine和MvDiffYL0Affine的取值范围为-32768～32767。

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。解码过程见9.5.7。RefIdxL1的值等于pu_reference_index_l1的值。如果位流中不存在pu_reference_index_l1，则RefIdxL1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。MvDiffXAbsL1等于mv_diff_x_abs_l1的值，MvDiffYAbsL1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。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MvDiffXSignL1的值为0，MvDiffXL1等于MvDiffXAbsL0；如果MvDiffXSignL1的值为1，MvDiffXL1等于-MvDiffXAbsL0。如果MvDiffYSignL1的值为0，MvDiffYL1等于MvDiffYAbsL1；如果MvDiffYSignL1的值为1，MvDiffYL1等于-MvDiffYAbsL1。MvDiffXL1和MvDiffYL1的取值范围是-32768～32767。

如果SmvdFlag的值等于1，分别按式（25）和式（26）计算MvDiffXL1和MvDiffYL1：

$$MvDiffXL1 = Clip3(-32768,\ 32767,\ -MvDiffXL0) \quad (25)$$

$$MvDiffYL1 = Clip3(-32768,\ 32767,\ -MvDiffYL0) \quad (26)$$

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。MvDiffXAbsL1Affine等于mv_diff_x_abs_l1_affine的值，MvDiffYAbsAffineAffine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。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。如果MvDiffXSignL1Affine的值为0，MvDiffXL1Affine等于MvDiffXAbsL1Affine;如果MvDiffXSignL1Affine的值为1,MvDiffXL1Affine等于-MvDiffXAbsL1Affine。如果MvDiffYSignL1Affine的值为0，MvDiffYL1Affine等于MvDiffYAbsL1Affine；如果MvDiffYSignL1Affine的值为1，MvDiffYL0Affine等于-MvDiffYAbsL1Affine。MvDiffXL1Affine和MvDiffYL1Affine的取值范围为-32768～32767。

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。

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的值等于0；如果位流中不存在ctp_v，则ctp_v的值等于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。如果位流中不存在ctp_y[0]且PbtCuFlag的值为0且IntraCuFlag和SkipFlag的值均为0且ctp_u和ctp_v的值均为0，则ctp_y[0]的值等于1。解析过程见8.3。

7.2.7　变换块

扫描表ScanCGInBlk，ScanCoeffInCG和InvScanCoeffInBlk的定义见附录E。

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的值为‘0’表示当前系数是正数；值为‘1’表示当前系数是负数。

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。

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

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0～15。

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1～15。如果位流中不存在alf_region_distance[i]，如果i等于0，则alf_region_distance[i]的值为0，如果i不等于0且alf_filter_num_minus1的值为15，则alf_region_distance[i]的值为1。符合标准的位流应满足alf_region_distance[i]（i=0～alf_filter_num_minus1）之和应小于等于15。

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。从符合本部分的位流中解码得到的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。

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。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。

8 解析过程

8.1 k阶指数哥伦布码

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)
```

表56给出了0阶、1阶、2阶和3阶指数哥伦布码的结构。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56 k阶指数哥伦布码表

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
	……	……

表 56 （续）

阶数	码字结构	CodeNum取值范围
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 57 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57 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	$(-1)^{k+1}\times Ceil(k\div 2)$

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。一个二元符号模型包括三个变量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。

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 binIdx 的值为-1，二元符号串为空。

b) binIdx 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 binIdx 不为 0 的二元符号；
- sao_interval_offset_abs 中 binIdx 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；
- mv_diff_x_sign 或 mv_diff_y_sign 的二元符号；
- mv_diff_x_abs 或 mv_diff_y_abs 中 binIdx 大于或等于 3 的二元符号；
- coeff_sign 的二元符号；
- coeff_run 中 binIdx 大于等于 16 的二元符号；
- umve_mv_idx 中 binIdx 不为 0 的二元符号；
- umve_step_idx 中 binIdx 不为 0 的二元符号；
- coeff_level_minus1 中 binIdx 大于等于 8 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 binIdx 得到每个二元符号对应的唯一的 ctxIdx，并根据 ctxIdx 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ctxIdxIncW不存在，二元符号模型ctx等于ctxArray[ctxIdx]，其中ctxArray是保存二元符号模型的数组，ctxIdx是数组的索引值；否则，二元符号模型ctx和ctxW分别等于ctxArray[ctxIdx]和ctxArray[ctxIdxW]，其中ctxArray是保存二元符号模型的数组，ctxIdx和ctxIdxW是数组的索引值。语法元素的每个二元符号的ctxIdx等于ctxIdxInc加ctxIdxStart，ctxIdxW等于ctxIdxIncW加ctxIdxStart。各语法元素对应的ctxIdxStart及每个二元符号对应的ctxIdxInc见表58，ctxIdxIncW见8.3.3.2.17。

表58 语法元素对应的 ctxIdxStart 和 ctxIdxInc

语法元素	ctxIdxInc	ctxIdxStart	ctx的数量
lcu_qp_delta	见8.3.3.2.2	0	4
sao_merge_type_index	binIdx+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

表 58 （续）

语法元素	ctxIdxInc	ctxIdxStart	ctx的数量
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_mode	0	31	1
intra_chroma_pred_mode	见8.3.3.2.7	32	3
ctp_u	0	35	1
ctp_v	0	36	1
skip_flag	见8.3.3.2.8	37	4
umve_flag	0	41	1
affine_flag	0	42	1
direct_flag	见8.3.3.2.9	43	2
intra_cu_flag	见8.3.3.2.10	45	6
dt_split_flag	0	51	1
dt_split_dir	0	52	1
dt_split_hqt_flag	0	53	1
dt_split_vqt_flag	0	54	1
dt_split_hadt_flag	0	55	1
dt_split_vadt_flag	0	56	1
umve_mv_idx	0	57	1
umve_step_idx	0	58	1
umve_dir_idx	binIdx	59	2
cu_affine_cand_idx	binIdx	61	4
cu_subtype_index	binIdx	65	11
extend_mvr_flag	0	76	1
affine_amvr_index	binIdx	77	2
amvr_index	binIdx	79	4
inter_pred_ref_mode	见8.3.3.2.11	83	3
smvd_flag	0	86	1
pu_reference_index_l0、 pu_reference_index_l1	见8.3.3.2.12	87	3
mv_diff_x_abs	见8.3.3.2.13	90	3
mv_diff_y_abs	见8.3.3.2.13	93	3
intra_luma_pred_mode	见8.3.3.2.14	96	7
ipf_flag	0	103	1
ctp_zero_flag	见8.3.3.2.15	104	2
pbt_cu_flag	0	106	1
ctp_y[i]	0	107	1
coeff_run	见8.3.3.2.16	108	24
coeff_level_minus1	见8.3.3.2.16	132	24
coeff_last	见8.3.3.2.17	156	34

8.3.3.2.2 确定 lcu_qp_delta 的 ctxIdxInc

根据以下方法确定lcu_qp_delta的ctxIdxInc：

——如果 binIdx 和 PreviousDeltaQP 均等于 0，则 ctxIdxInc 等于 0；

——否则，如果 binIdx 等于 0 且 PreviousDeltaQP 不等于 0，则 ctxIdxInc 等于 1；

——否则，如果 binIdx 等于 1，则 ctxIdxInc 等于 2；

——否则，ctxIdxInc 等于 3。

8.3.3.2.3 确定 qt_split_flag 的 ctxIdxInc

根据以下方法确定qt_split_flag的ctxIdxInc（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 ctxIdxInc 等于 3；

——否则，如果 A“存在”且 A 的高度小于 E 的高度，且 B“存在”且 B 的宽度小于 E 的宽度，则 ctxIdxInc 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 ctxIdxInc 等于 1；

——否则，ctxIdxInc 等于 0。

8.3.3.2.4 确定 bet_split_flag 的 ctxIdxInc

根据以下方法确定bet_split_flag的ctxIdxInc（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，且 B“存在”且 B 的宽度小于 E 的宽度，则 ctxIdxInc 等于 2；

——否则，如果 A“存在”且 A 的高度小于 E 的高度，或块 B“存在”且块 B 的宽度小于 E 的宽度，则 ctxIdxInc 等于 1；

——否则，ctxIdxInc 等于 0。

根据以下方法进一步确定 bet_split_flag 的 ctxIdxInc：

——如果 E 的宽度乘以 E 的高度的积大于 1024，则 ctxIdxInc 不变；

——否则，如果 E 的宽度乘以 E 的高度的积大于 256，则 ctxIdxInc 增加 3；

——否则，ctxIdxInc 增加 6。

8.3.3.2.5 确定 bet_split_type_flag 的 ctxIdxInc

根据以下方法确定bet_split_type_flag的ctxIdxInc（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，且 B“存在”且 B 的宽度小于 E 的宽度，则 ctxIdxInc 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 ctxIdxInc 等于 1；

——否则，ctxIdxInc 等于 0。

8.3.3.2.6 确定 bet_split_dir_flag 的 ctxIdxInc

根据以下方法确定bet_split_dir_flag的ctxIdxInc（E是当前亮度编码块）：

——如果 E 的宽度为 128 且高度为 64，则 ctxIdxInc 等于 4；

——否则，如果 E 的宽度为 64 且高度为 128，则 ctxIdxInc 等于 3；
——否则，如果 E 的高度大于 E 的宽度，则 ctxIdxInc 等于 2；
——否则，如果 E 的宽度大于 E 的高度，则 ctxIdxInc 等于 1；
——否则，ctxIdxInc 等于 0。

8.3.3.2.7 确定 intra_chroma_pred_mode 的 ctxIdxInc

根据以下方法确定intra_chroma_pred_mode的ctxIdxInc：
——如果 binIdx 等于 0，则 ctxIdxInc 等于 0；
——否则，如果 tscpm_enable_flag 的值等于‘1’且 binIdx 等于 1，则 ctxIdxInc 等于 2；
——否则，ctxIdxInc 等于 1。

8.3.3.2.8 确定 skip_flag 的 ctxIdxInc

根据以下方法确定skip_flag的ctxIdxInc（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，ctxIdxInc 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，且当前预测块 E 的上边预测块 B“存在”且 B 是跳过模式，则 ctxIdxInc 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 ctxIdxInc 等于 1；
——否则，ctxIdxInc 等于 0。

8.3.3.2.9 确定 direct_flag 的 ctxIdxInc

根据以下方法确定direct_flag的ctxIdxInc：
——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 ctxIdxInc 等于 1；
——否则，ctxIdxInc 等于 0。

8.3.3.2.10 确定 intra_cu_flag 的 ctxIdxInc

根据以下方法确定intra_cu_flag的ctxIdxInc（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 高度大于 64，则 ctxIdxInc 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，且当前预测块 E 的上边预测块 B“存在”且预测块 B 是帧内预测模式，则 ctxIdxInc 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 ctxIdxInc 等于 1。
——否则，ctxIdxInc 等于 0。
如果 ctxIdxInc 等于 0，根据以下方法进一步确定 intra_cu_flag 的 ctxIdxInc：
——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 ctxIdxInc 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 ctxIdxInc 等于 3；
——否则，ctxIdxInc 等于 4。

8.3.3.2.11 确定 inter_pred_ref_mode 的 ctxIdxInc

根据以下方法确定inter_pred_ref_mode的ctxIdxInc：

——如果该语法元素的 binIdx 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 ctxIdxInc 等于 2；
——否则，如果该语法元素的 binIdx 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 ctxIdxInc 等于 0
——否则（该语法元素的 binIdx 为 1），ctxIdxInc 等于 1。

8.3.3.2.12 确定 pu_reference_index_l0 或 pu_reference_index_l1 的 ctxIdxInc

根据以下方法确定pu_reference_index的ctxIdxInc：
——如果 binIdx 等于 0，则 ctxIdxInc 等于 0；
——否则，如果 binIdx 等于 1，则 ctxIdxInc 等于 1；
——否则，ctxIdxInc 等于 2。

8.3.3.2.13 确定 mv_diff_x_abs 或 mv_diff_y_abs 的 ctxIdxInc

根据以下方法确定mv_diff_x_abs或mv_diff_y_abs的ctxIdxInc：
——如果 binIdx 等于 0，则 ctxIdxInc 等于 0；
——否则，如果 binIdx 等于 1，则 ctxIdxInc 等于 1；
——否则，如果 binIdx 等于 2，则 ctxIdxInc 等于 2；
——否则，BypassFlag 的值为 1。

8.3.3.2.14 确定 intra_luma_pred_mode 的 ctxIdxInc

根据以下方法确定intra_luma_pred_mode的ctxIdxInc：
——根据该语法元素 binIdx 和当前已解析的二元符号串和 binIdx，查表 59 得到 ctxIdxInc。表 59 中，x_i(i=1～4)的值为‘0’或‘1’。

表59 前缀二元符号串与 ctxIdxInc 的关系

binIdx	当前已解析的二元符号串	ctxIdxInc的值
0	空	0
1	0	1
2	0 x_1	2
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

8.3.3.2.15 确定 ctp_zero_flag 的 ctxIdxInc

根据以下方法确定ctp_zero_flag的ctxIdxInc：
——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 ctxIdxInc 等于 1；
——否则，则 ctxIdxInc 等于 0。

8.3.3.2.16 确定 coeff_run 或 coeff_level_minus1 的 ctxIdxInc

根据以下方法确定coeff_level_minus1或coeff_run的ctxIdxInc：

——令 ctxIdxInc 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，ctxIdxInc = ctxIdxInc + min(prev_level-1, 5) × 2；
——如果当前块为色度块，则 ctxIdxInc = ctxIdxInc + 12；
——如果 binIdx 大于等于 1，则 ctxIdxInc = ctxIdxInc + 1。

8.3.3.2.17 确定 coeff_last 的 ctxIdxInc 和 ctxIdxIncW

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ctxIdxInc：
——令 ctxIdxInc 的值为 min(prev_level-1,5)；
——如果当前块为色度块，则 ctxIdxInc = ctxIdxInc + 6。

根据以下方法确定coeff_last第二个上下文的ctxIdxIncW：
——令 ctxIdxIncW 的值为 12 + Floor(Log(ScanPosOffset +1))；
——如果当前块为色度块，则 ctxIdxIncW = ctxIdxIncW + 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) 首先，解析二元符号值 binVal
   1) 如果 BypassFlag 的值为 1，执行 decode_bypass 过程（见 8.3.3.3.4）；
   2) 否则，如果 StuffingBitFlag 的值为 1，则执行 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) 第二步，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

decode_decision过程的输入是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过程的输出是二元符号值binVal。decode_decision过程用伪代码描述如下：

```
decode_decision( ) {
    if ( CtxWeight == 0) {
        predMps = ctx->mps
        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

decode_aec_stuffing_bit过程的输入是bFlag、cFlag、rS1、rS2、valueS、valueT和valueD。decode_aec_stuffing_bit过程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，带入decode_decision过程实现decode_aec_stuffing_bit过程。

注：decode_aec_stuffing_bit过程也可参考附录F的描述方式实现。

8.3.3.3.4 decode_bypass

decode_bypass过程的输入是bFlag、cFlag、rS1、rS2、valueS、valueT和valueD。decode_bypass过程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，带入decode_decision过程实现decode_bypass过程。

注：decode_bypass过程也可参考附录F的描述方式实现。

8.3.3.3.5 update_ctx

update_ctx过程的输入是binVal和ctx。update_ctx过程的输出是更新后的ctx。update_ctx过程用伪代码描述如下：

```
update_ctx( ) {
    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 {
        if ( cwr == 3 )
            ctx->lgPmps = ctx->lgPmps + 197
        else if ( cwr == 4 )
            ctx->lgPmps = ctx->lgPmps + 95
        else
            ctx->lgPmps = ctx->lgPmps + 46
        if ( ctx->lgPmps > 1023 ) {
            ctx->lgPmps = 2047 - ctx->lgPmps
        }
    }
    return (ctx)
}
```

8.3.4 反二值化方法

8.3.4.1 概述

本条定义语法元素的反二值化方法，见表60。

表60 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

表 60（续）

语法元素	反二值化方法
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[compIdx][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
root_cu_mode	见 8.3.4.4，root_cu_mode 的值等于 synElVal
intra_chroma_pred_mode	见 8.3.4.2，maxVal=(TscpmEnableFlag ? 5 : 4)，intra_chroma_pred_mode 的值等于 synElVal。如果 isRedundant 的值为 1，intra_chroma_pred_mode 的值不应为 maxVal。
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
umve_flag	见 8.3.4.4，umve_flag 的值等于 synElVal
affine_flag	见 8.3.4.4，affine_flag 的值等于 synElVal
direct_flag	见 8.3.4.4，direct_flag 的值等于 synElVal
intra_cu_flag	见 8.3.4.4，intra_cu_flag 的值等于 synElVal
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_idx	见 8.3.4.2，maxVal=1，umve_mv_idx 的值等于 synElVal
umve_step_idx	见 8.3.4.2，maxVal=4，umve_step_idx 的值等于 synElVal
umve_dir_idx	见 8.3.4.11
cu_affine_cand_idx	见 8.3.4.2，maxVal=4，cu_affine_cand_idx 的值等于 synElVal
cu_subtype_index	见 8.3.4.2，maxVal=((PictureType == 1) ? (1+ NumOfHmvpCand) : (3+NumOfHmvpCand))，cu_subtype_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
amvr_index	见 8.3.4.2，maxVal=4，amvr_index 的值等于 synElVal
inter_pred_ref_mode	见 8.3.4.12
smvd_flag	见 8.3.4.4，smvd_flag 的值等于 synElVal

表 60（续）

语法元素	反二值化方法
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	见 8.3.4.13
mv_diff_x_sign	见 8.3.4.4，mv_diff_x_sign 的值等于 synElVal
mv_diff_y_abs	见 8.3.4.13
mv_diff_y_sign	见 8.3.4.4，mv_diff_y_sign 的值等于 synElVal
intra_luma_pred_mode	见 8.3.4.14
ipf_flag	见 8.3.4.4，ipf_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
coeff_run	见 8.3.4.15，TH=16，coeff_run 的值等于 synElVal
coeff_level_minus1	见 8.3.4.15，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
aec_ipcm_stuffing_bit	见 8.3.4.4，aec_ipcm_stuffing_bit 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61得到synElVal的值。

表61 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			
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
binIdx	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62得到synElVal的值。

表62　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
…						
binIdx	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63得到synElVal的值。

表63　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
binIdx	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64得到sao_merge_type_index的值。

表64　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	

表 64（续）

SaoMergeLeftAvai的值	SaoMergeUpAvai的值	sao_merge_type_index的值	二元符号串	
1	1	0	0	0
		1	1	
		2	0	1
binIdx			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65得到sao_interval_start_pos的值。

表65 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
binIdx	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66得到sao_interval_delta_pos_minus2的值。

表66 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		
7	0	0	1	0	0	1		
8	0	0	1	0	1	0		
9	0	0	1	0	1	1		

表 66（续）

sao_interval_delta_pos_minus2的值	二元符号串							
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					
binIdx	0	1	2	3	4	5	6	7

8.3.4.9 sao_edge_offset[compIdx][j]的反二值化方法

由二元符号串查表67得到sao_edge_offset[compIdx][0]或sao_edge_offset[compIdx][3]的值；由二元符号串查表68得到sao_edge_offset[compIdx][1]或sao_edge_offset[compIdx][2]的值。

表67 sao_edge_offset[compIdx][0]和 sao_edge_offset[compIdx][3]的值与二元符号串的关系

sao_edge_offset[compIdx][0]	sao_edge_offset[compIdx][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
binIdx		0	1	2	3	4	5	6

表68 sao_edge_offset[compIdx][1]和 sao_edge_offset[compIdx][2]的值与二元符号串的关系

sao_edge_offset[compIdx][1]	sao_edge_offset[compIdx][2]	二元符号串
0	0	1
1	-1	0
binIdx		0

8.3.4.10 sao_edge_type 的反二值化方法

由二元符号串查表 69 得到 sao_edge_type 的值。

表69 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
binIdx	0	1

8.3.4.11 umve_dir_idx 的反二值化方法

由二元符号串查表70得到umve_dir_idx的值。

表70 umve_dir_idx 的值与二元符号串的关系

umve_dir_idx的值	二元符号串	
0	0	0
1	0	1
2	1	0
3	1	1
binIdx	0	1

8.3.4.12 inter_pred_ref_mode 的反二值化方法

由二元符号串查表 71 得到 inter_pred_ref_mode 的值。

表71 inter_pred_ref_mode 的值与二元符号串的关系

inter_pred_ref_mode的值	二元符号串	
0	0	0
1	0	1
2	1	
binIdx	0	1

8.3.4.13 mv_diff_x_abs_l0、mv_diff_y_abs_l0、mv_diff_x_abs_l1 和 mv_diff_y_abs_l1 的反二值化方法

由二元符号串查表 72 得到 synElVal 的值。表 72 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 56）；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_y_abs_l0、mv_diff_x_abs_l1 或 mv_diff_y_abs_l1 的值等于 synElVal。

表72　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		
…											
binIdx	0	1	2	3	4	5	6	7	8	9	10

8.3.4.14　intra_luma_pred_mode 的反二值化方法

由二元符号串查表 73 得到 intra_luma_pred_mode 的值。

表73　intra_luma_pred_mode 的值与二元符号串的关系

intra_luma_pred_mode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	0	0	0	1	0	
…							
32	0	1	1	1	1	0	
binIdx	0	1	2	3	4	5	6

8.3.4.15　coeff_run 和 coeff_level_minus1 的反二值化方法

由二元符号串查表 74 得到 synElVal 的值。如果 synElVal 的值小于 TH，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74 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								
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
…																					
binIdx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

9 解码过程

9.1 序列解码

序列解码过程如下：

a) 第一步，将 InitialFlag 的值初始化为 0；

b) 第二步，解码序列头：

1) 步骤 2.1，如果 InitialFlag 的值为 1，直接执行步骤 2.2；否则：

- 将 DOIPrev 和 DOICycleCnt 的值均初始化为 0；
- 清空解码图像缓冲区；
- 将 InitialFlag 的值置为 1；
- 将 LibraryBufferEmpty 的值初始化为 1；

2) 步骤 2.2，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 WeightQuantMatrix4x4 和 8×8 加权量化矩阵 WeightQuantMatrix8x8；

c) 第三步，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：

d) 第四步，

1) 如果遇到序列起始码，继续执行第二步；

2) 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：

——解码图像头（见 9.2.2）；

——构建参考图像队列（见 9.2.3）；

——更新解码图像缓冲区（见 9.2.4）；

——解码当前图像的各个片（见 9.3），得到补偿后样本；

——如果 loop_filter_disable_flag 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 loop_filter_disable_flag 的值为‘1’，则将补偿后样本直接作为滤波后样本；

——如果 sample_adaptive_offset_enable_flag 的值为‘1’，对滤波后样本进行样值偏移补偿操作（见 9.11），得到偏移后样本；如果 sample_adaptive_offset_enable_flag 的值为‘0’，则将滤波后样本直接作为偏移后样本；

——如果 adaptive_leveling_filter_enable_flag 的值为‘1’，对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 adaptive_leveling_filter_enable_flag 的值为‘0’，则将偏移后样本直接作为重建样本；

——重建样本构成当前图像的解码图像；

——如果 LibraryStreamFlag 的值等于 0，则将当前图像的解码图像移入解码图像缓冲区，并置该图像的 IsLibraryPicture 的值等于 0，置该图像为“未输出”和“被参考”；

——输出解码图像（见 9.2.5）；

——存储当前解码图像的运动信息（见 9.13）。

9.2.2 图像头解码

图像头解码过程如下：

——如果当前图像起始码是 0x000001B3，则 PictureType 等于 0，当前图像是 I 图像。

——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。

——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 fixed_picture_qp_flag。

——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。

——如果 LibraryStreamFlag 的值等于 0，则 DOIPrev 等于 DOI。

——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。

——如果 PicWeightQuantEnableFlag 的值等于 1，从当前图像头的位流中导出的加权量化矩阵：

1） 首先，按 9.2.6.2 确定 4×4 和 8×8 变换块的加权量化矩阵；

2） 其次，根据第一步确定的 8×8 加权量化矩阵，按 9.2.6.3 确定 M_1×M_2 变换块（M_1 或 M_2 大于 8）的加权量化矩阵。

此时，符合本部分的位流应满足以下要求：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。

9.2.3 构建参考图像队列

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。参考图像队列 0 中的图像由 RefPicDoiList[0][j]索引，j 的取值范围是 0~Max(0, NumOfRefPic[0][RplsIdx[0]]-1)；参考图像队列 1 中的图像由 RefPicDoiList[1][j]索引，j 的取值范围是 0~Max(0, NumOfRefPic[1][RplsIdx[1]]-1)。

对图像进行解码图像前应先构建参考图像队列 i（i 等于 0 或 1），方法如下：

——初始化 LibraryPictureExistFlag 和 j 为 0，doiBase 为当前图像的 DOI；

——进行 NumOfRefPic[i][RplsIdx[i]]次以下操作：

1） 如果 LibraryIndexFlag[i][RplsIdx[i]][j]的值等于 1，则令 RefPicDoiList[i][j]等于 ReferencedLibraryPictureIndex[i][RplsIdx[i]][j]，并令 j 等于 j+1，LibraryPictureExistFlag 等于 1；

2） 如果 LibraryIndexFlag[i][RplsIdx[i]][j]的值等于 0，则令 RefPicDoiList[i][j]等于 doiBase-DeltaDoi[X][RplsIdx[i]][j]，并令 doiBase 等于 RefPicDoiList[i][j]，j 等于 j+1。

符合本部分的位流应满足以下要求：

——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][RplsIdx[i]]的值。

——RefPicDoiList[0][j]中非知识图像的 RefPicDoiList[0][j]值各不相同；RefPicDoiList[1][j]中非知识图像的 RefPicDoiList[1][j]的值各不相同。

——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][RplsIdx[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且与 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 的显示时间间隔至少为 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IsLibraryPicture为0的图像，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”。如果一副图像被标记为“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：

——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 IsLibraryPicture 为 1 的图像且当前图像的 LibraryPictureExistFlag 大于 0 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已存在的 IsLibraryPicture 为 1 的图像的知识图像索引不相同，则移出解码图像缓冲区中已存在的 IsLibraryPicture 为 1 的图像且置 LibraryBufferEmpty 为 1。

——移入图像，如果 LibraryBufferEmpty 为 1 且当前图像的 LibraryPictureExistFlag 大于 0，则：

1）在主位流解码器的知识图像索引输出端口输出当前图像所参考的知识图像的知识图像索引并从主位流解码器的知识图像输入端口得到对应的知识图像；

2）将该知识图像移入解码图像缓冲区；

3）置该知识图像的 IsLibraryPicture 为 1，知识图像索引为当前图像所参考的知识图像的知识图像索引；

4）置 LibraryBufferEmpty 为 0。

符合本部分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、当前解码图像的参考图像队列中的图像（包括知识图像）及“未输出”图像的总数不应超过 MaxDpbSize 的值。

9.2.5 输出解码图像

执行以下步骤输出解码图像，

——如果 LibraryStreamFlag 的值等于 1，则输出当前解码图像和对应的 LibraryPictureIndex；

——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”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”。

9.2.6 确定加权量化矩阵

9.2.6.1 概述

本条定义确定当前图像的加权量化矩阵的方法。

如果WeightQuantEnableFlag的值为0或PicWeightQuantEnableFlag的值为0，则加权量化矩阵WeightQuantMatrix8x8的元素WeightQuantMatrix8x8[i][j]（i,j=0～7）的值均初始化为64；加权量化矩阵WeightQuantMatrix4x4的元素WeightQuantMatrix4x4[i][j]（i,j =0～3）的值均初始化为64；加权量化矩阵WeightQuantMatrixM1xM2的元素WeightQuantMatrixM1xM2[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PicWeightQuantEnableFlag的值为1，先根据9.2.6.2确定4×4和8×8加权量化矩阵，再根据9.2.6.3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，按9.2.6.4定义的方法修正加权量化矩阵WeightQuantMatrixM1xM2。

9.2.6.2 确定 4×4 和 8×8 加权量化矩阵

如果pic_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WeightQuantMatrix4x4和和8×8加权量化矩阵WeightQuantMatrix8x8。

否则，如果pic_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WeightQuantMatrix4x4和8×8加权量化矩阵WeightQuantMatrix8x8。

否则，如果pic_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：

a) 第一步，确定当前图像的 wqP（wqP 的元素的取值范围应是 1～255）：

1) 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）。

2) 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）。

3) 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）。

b) 第二步，根据 WeightQuantModel 确定 8×8 加权量化矩阵 wqM8x8：

1) 如果 WeightQuantModel 的值为 0，则 wqM8x8 的定义见式（27）：

$$wqM8x8 = \begin{bmatrix} wqP[0] & wqP[0] & wqP[0] & wqP[4] & wqP[4] & wqP[4] & wqP[5] & wqP[5] \\ wqP[0] & wqP[0] & wqP[3] & wqP[3] & wqP[3] & wqP[3] & wqP[5] & wqP[5] \\ wqP[0] & wqP[3] & wqP[2] & wqP[2] & wqP[1] & wqP[1] & wqP[5] & wqP[5] \\ wqP[4] & wqP[3] & wqP[2] & wqP[2] & wqP[1] & wqP[5] & wqP[5] & wqP[5] \\ wqP[4] & wqP[3] & wqP[1] & wqP[1] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \\ wqP[4] & wqP[3] & wqP[1] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \\ wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \\ wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \end{bmatrix} \quad \cdots\cdots\cdots\cdots\cdots\cdots (27)$$

2) 如果 WeightQuantModel 的值为 1，则 wqM8x8 的定义见式（28）：

$$wqM8x8 = \begin{bmatrix} wqP[0] & wqP[0] & wqP[0] & wqP[4] & wqP[4] & wqP[4] & wqP[5] & wqP[5] \\ wqP[0] & wqP[0] & wqP[4] & wqP[4] & wqP[4] & wqP[4] & wqP[5] & wqP[5] \\ wqP[0] & wqP[3] & wqP[2] & wqP[2] & wqP[2] & wqP[1] & wqP[5] & wqP[5] \\ wqP[3] & wqP[3] & wqP[2] & wqP[2] & wqP[1] & wqP[5] & wqP[5] & wqP[5] \\ wqP[3] & wqP[3] & wqP[2] & wqP[1] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \\ wqP[3] & wqP[3] & wqP[1] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \\ wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \\ wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \end{bmatrix} \quad \cdots\cdots\cdots\cdots\cdots\cdots (28)$$

3) 如果 WeightQuantModel 的值为 2，则 wqM8x8 的定义见式（29）：

$$wqM8x8=\begin{bmatrix} wqP[0] & wqP[0] & wqP[0] & wqP[4] & wqP[4] & wqP[3] & wqP[5] & wqP[5] \\ wqP[0] & wqP[0] & wqP[4] & wqP[4] & wqP[3] & wqP[2] & wqP[5] & wqP[5] \\ wqP[0] & wqP[4] & wqP[4] & wqP[3] & wqP[2] & wqP[1] & wqP[5] & wqP[5] \\ wqP[4] & wqP[4] & wqP[3] & wqP[2] & wqP[1] & wqP[5] & wqP[5] & wqP[5] \\ wqP[4] & wqP[3] & wqP[2] & wqP[1] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \\ wqP[3] & wqP[2] & wqP[1] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \\ wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \\ wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] & wqP[5] \end{bmatrix} \quad \cdots\cdots(29)$$

c) 第三步，根据 WeightQuantModel 确定 4×4 加权量化矩阵 wqM4x4：

1) 如果 WeightQuantModel 的值为 0，则 wqM4x4 的定义见式（30）：

$$wqM4x4=\begin{bmatrix} wqP[0] & wqP[4] & wqP[3] & wqP[5] \\ wqP[4] & wqP[2] & wqP[1] & wqP[5] \\ wqP[3] & wqP[1] & wqP[1] & wqP[5] \\ wqP[5] & wqP[5] & wqP[5] & wqP[5] \end{bmatrix} \quad \cdots\cdots(30)$$

2) 如果 WeightQuantModel 的值为 1，则 wqM4x4 的定义见式（31）：

$$wqM4x4=\begin{bmatrix} wqP[0] & wqP[4] & wqP[4] & wqP[5] \\ wqP[3] & wqP[2] & wqP[2] & wqP[5] \\ wqP[3] & wqP[2] & wqP[1] & wqP[5] \\ wqP5] & wqP[5] & wqP[5] & wqP[5] \end{bmatrix} \quad \cdots\cdots(31)$$

3) 如果 WeightQuantModel 的值为 2，则 wqM4x4 的定义见式（32）：

$$wqM4x4=\begin{bmatrix} wqP[0] & wqP[4] & wqP[3] & wqP[5] \\ wqP[4] & wqP[3] & wqP[2] & wqP[5] \\ wqP[3] & wqP[2] & wqP[1] & wqP[5] \\ wqP5] & wqP[5] & wqP[5] & wqP[5] \end{bmatrix} \quad \cdots\cdots(32)$$

8×8 变换块的加权量化矩阵 WeightQuantMatrix8x8 等于 wqM8x8；4×4 变换块的加权量化矩阵 WeightQuantMatrix4x4 等于 wqM4x4。

9.2.6.3 M_1×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，按以下方式确定M_1×M_2变换块所使用的加权量化矩阵 WeightQuantMatrixM1xM2：

a) 根据 9.2.6.2 确定 8×8 变换块所使用的加权量化矩阵 WeightQuantMatrix8x8；

b) 确定 M_1×M_2 变换块所使用的加权量化矩阵 WeightQuantMatrixM1xM2 为 WeightQuantMatrixM1xM2[x][y] = WeightQuantMatrix8x8[x>>shift][y>>shift]，x=0～M_1-1，y =0～M_2-1，其中 shift 等于 Max(Log(M_1),Log(M_2))-3。

9.2.6.4 M_1×M_2 加权量化矩阵的修正方法

对加权量化矩阵WeightQuantMatrixM1xM2的元素WeightQuantMatrixM1xM2[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WeightQuantMatrixM1xM2[i][j]置为0；否则，WeightQuantMatrixM1xM2[i][j]的值保持不变。

9.3 片解码

片解码过程如下：

——LcuIndex 等于 LcuRow × pictureWidthInLcu + LcuColumn；

——如果 fixed_picture_qp_flag 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 fixed_patch_qp_flag；

——计算得到当前片的边界，以 LCU 为单位，计算过程如下：

```
PatchLeftInLcu = LcuIndex % pictureWidthInLcu
PatchRightInLcu = PatchLeftInLcu + PatchSizeInLCU[PatchIndexY][PatchIndexX]->Width
PatchAboveInLcu = LcuIndex / pictureWidthInLcu
PatchBelowInLcu = PatchAboveInLcu + PatchSizeInLCU[PatchIndexY][PatchIndexX]->Height
```

——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，其解码过程如下：

——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，将历史运动信息表中候选项数 CntHmvp 的值初始化为 0。

——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，按以下步骤更新 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：确定编码单元类型和预测类型（见9.5.3）；对预测类型为帧内的编码单元分别导出其所包含的所有帧内预测块的帧内预测模式（见9.5.6）并进行帧内预测（见9.7），对预测类型为帧间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和9.5.8）并进行帧间预测（见9.8）。

残差样本解码包括：确定量化参数（见9.5.2）；确定编码单元划分为变换块的方式（见9.5.5）；依次解码各个变换块（见9.6）。如果当前变换块包含多个变换块，则将解码得到的$M_1 \times M_2$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）。

9.5.2 确定量化参数

本条确定量化参数QP_X（X为Y、Cb或Cr）。

当前编码单元的量化参数CurrentQp等于PreviousQp加上当前编码单元所在的最大编码单元的lcuQpDelta。CurrentQp的取值范围应是0～(63 + 8 × (BitDepth – 8))。其中，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”。

亮度的量化参数QP_Y等于其所在编码单元的CurrentQp。

先按以下方法计算xCb、xCr（xCb、xCr的取值范围应是-16～63）：

$$xCb = Clip3(-16,\ 63,\ CurrentQp - 8 \times (BitDepth - 8) + chroma\_quant\_param\_delta\_cb)$$

$$xCr = Clip3(-16,\ 63,\ CurrentQp - 8 \times (BitDepth - 8) + chroma\_quant\_param\_delta\_cr)$$

然后分别以xCb和xCr为索引查表75得到QP'_{cb}和QP'_{cr}。Cb、Cr色度块的量化参数QP_{cb}和QP_{cr}按以下方法得到：

$$QP_{cb} = Clip3(0,\ 63 + 8 \times (BitDepth - 8), QP'_{cb} + 8 \times (BitDepth - 8))$$

$$QP_{cr} = Clip3(0,\ 63 + 8 \times (BitDepth - 8), QP'_{cr} + 8 \times (BitDepth - 8))$$

本条确定的亮度量化参数QP_Y的取值范围应是0～[63 + 8 × (BitDepth − 8)]。

表75　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
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

编码单元的预测模式为‘PRED_Intra_Only’（仅使用帧内预测）、‘PRED_Inter_Only’（仅使用帧间预测）或‘PRED_No_Constraint’（可使用帧内预测和帧间预测）。

编码单元的分量模式为‘COMPONENT_Luma’（仅包含亮度分量）、‘COMPONENT_Chroma’（仅包含色度分量）或‘COMPONENT_Luma_Chroma’（可包含亮度分量和色度分量）。

编码单元类型（CuType）由图像类型（PictureType）、跳过模式标志（SkipFlag）、直接模式标志（DirectFlag）和预测划分尺寸（PartSize）查表76得到。

表76　编码单元类型

图像类型	跳过模式标志	直接模式标志	预测划分尺寸	编码单元类型
0	-	-	SIZE_2Mx2N	I_2M_2N
	-	-	SIZE_2MxhN	I_2M_hN
	-	-	SIZE_2MxnU	I_2M_nU
	-	-	SIZE_2MxnD	I_2M_nD
	-	-	SIZE_hMx2N	I_hM_2N
	-	-	SIZE_nLx2N	I_nL_2N
	-	-	SIZE_nRx2N	I_nR_2N
1	1	0	SIZE_2Mx2N	P_Skip
	0	1	SIZE_2Mx2N	P_Direct
	0	0	SIZE_2Mx2N	P_2M_2N
2	1	0	SIZE_2Mx2N	B_Skip
	0	1	SIZE_2Mx2N	B_Direct
	0	0	SIZE_2Mx2N	B_2M_2N

编码单元预测划分尺寸（PartSize）由DtSplitFlag、DtSplitDir、DtSplitHqtFlag、DtSplitHadtFlag、DtSplitVqtFlag和DtSplitVadtFlag的值决定，见表77。

表77　预测划分尺寸

DtSplitFlag	DtSplitDir	DtSplitHqtFlag	DtSplitHadtFlag	DtSplitVqtFlag	DtSplitVadtFlag	预测划分尺寸
0	-	-	-	-	-	SIZE_2M×2N
1	1	1	-	-	-	SIZE_2M×hN
1	1	0	1	-	-	SIZE_2M×nD
1	1	0	0	-	-	SIZE_2M×nU
1	0	0	-	1	-	SIZE_hM×2N
1	0	0	-	0	1	SIZE_nR×2N
1	0	0	-	0	0	SIZE_nL×2N

如果当前编码单元的IntraCuFlag为1，根据编码单元类型（CuType）查表78确定预测划分方式（SplitMode）、帧内亮度预测块数（NumOfIntraPredBlock）和预测块预测模式（PbPredMode）。

表78　IntraCUFlag 为 1 时的编码单元类型和相关信息

编码单元类型	预测划分方式	帧内亮度预测块数	预测块预测模式
I_2Mx2N	I_NO_SPLIT	1	由9.5.6导出
I_2M_hN	HOR_tN	4	由9.5.6导出
I_2M_nU	HOR_UP	2	由9.5.6导出
I_2M_nD	HOR_DOWN	2	由9.5.6导出
I_hM_2N	VER_tN	4	由9.5.6导出
I_nL_2N	VER_LEFT	2	由9.5.6导出
I_nR_2N	VER_RIGHT	2	由9.5.6导出

如果当前图像是P图像，根据编码单元类型（CuType）查表79确定编码单元运动矢量数（CuMvNum）、预测划分方式（SplitMode）和预测单元预测参考模式（PuPredRefMode）。

表79　P 图像的编码单元类型和相关信息

CuType Index	编码单元类型	编码单元运动矢量数	预测划分方式	预测单元预测参考模式			
				PredUnitOrder 的值为 0	PredUnitOrder 的值为 1	PredUnitOrder 的值为 2	PredUnitOrder 的值为 3
0	P_Skip	0	NO_SPLIT	PRED_List0	-	-	-
1	P_Direct	0	NO_SPLIT	PRED_List0	-	-	-
2	P_2M_2N	1	NO_SPLIT	PRED_List0	-	-	-

如果当前图像是P图像且编码单元类型是‘P_Skip’或‘P_Direct’，则根据编码单元类型（CuType）、编码单元子类型索引（CuSubTypeIdx）、UMVE模式标志（UmveFlag）和仿射模式标志（AffineFlag）查表80确定编码单元子类型（CuSubType）。其中UmveFlag和AffineFlag的值不应同时为1。

表80　P 图像的编码单元子类型

编码单元类型	编码单元子类型索引	UMVE模式标志	仿射模式标志	编码单元子类型
P_Skip	0	0	0	P_Skip_Temporal
	1	0	0	P_Skip_Spatial_list0
	2～9	0	0	P_Skip_Hmvp
	-	1	0	P_Skip_Umve
	-	0	1	P_Skip_Affine
P_Direct	0	0	0	P_Direct_Temporal
	1	0	0	P_Direct_Spatial_list0
	2～9	0	0	P_Direct_Hmvp
	-	1	0	P_Direct_Umve
	-	0	1	P_Direct_Affine

如果当前图像是B图像，根据编码单元类型（CuType）查表81确定预测划分方式（SplitMode）、预测单元预测参考模式（PuPredRefMode）和预测单元数（PuNum），其中：

——如果编码单元类型是‘B_Skip’或‘B_Direct’，根据编码单元类型（CuType）、编码单元子

类型索引（CuSubtypeIdx）、UMVE 模式标志（UmveFlag）和仿射模式标志（AffineFlag）查表 82 确定编码单元子类型（CuSubtype）、预测单元运动矢量数（PuMvNum）和预测单元预测参考模式（PuPredMode）。其中 UmveFlag 和 AffineFlag 的值不应同时为 1。

——否则，根据预测参考模式（InterPredRefMode）查表 83 得到当前预测单元的预测单元运动矢量数（PuMvNum）和预测单元预测参考模式（PuPredRefMode）。

表81　B 图像的编码单元类型和相关信息

CuTypeIndex	编码单元类型	预测划分方式	预测单元预测参考模式	预测单元数
0	B_Skip	NO_SPLIT	由表 82 得到	1
1	B_Direct	NO_SPLIT	由表 82 得到	1
2	B_2M_2N	NO_SPLIT	由表 83 得到	1

表82　B 图像的编码单元子类型和相关信息

编码单元类型	编码单元子类型索引	UMVE模式标志	仿射模式标志	编码单元子类型	预测单元预测参考模式	预测单元运动矢量数
B_Skip	0	0	0	B_Skip_Temporal	PRED_Temporal	0
	1	0	0	B_Skip_Spatial_list01	PRED_List01	0
	2	0	0	B_Skip_Spatial_list1	PRED_List1	0
	3	0	0	B_Skip_Spatial_list0	PRED_List0	0
	4～11	0	0	B_Skip_Hmvp	由9.5.8.6.3和9.5.8.6.5得到	0
	-	1	0	B_Skip_Umve	由9.5.8.6.4得到	0
	-	0	1	B_Skip_Affine	由9.5.8.6.6得到	0
B_Direct	0	0	0	B_Direct_Temporal	PRED_Temporal	0
	1	0	0	B_Direct_Spatial_list01	PRED_List01	0
	2	0	0	B_Direct_Spatial_list1	PRED_List1	0
	3	0	0	B_Direct_Spatial_list0	PRED_List0	0
	4～11	0	0	B_Direct_Hmvp	由9.5.8.6.3和9.5.8.6.5得到	0
	-	1	0	B_Direct_Umve	由9.5.8.6.4得到	0
	-	0	1	B_Diret_Affine	由9.5.8.6.6得到	0

表83　B 图像编码单元各预测单元的预测参考模式和相关信息

预测参考模式	预测单元运动矢量数	预测单元预测参考模式
0	1	PRED_List0
1	1	PRED_List1
2	2	PRED_List01

表79、表82和表83中的编码单元运动矢量数（CuMvNum）表示当前编码单元在位流中的运动矢量数，预测单元运动矢量数（PuMvNum）表示预测单元在位流中的运动矢量数。一个编码单元的编码单元运动矢量数（CuMvNum）等于其所包含的预测单元的预测单元运动矢量数（PuMvNum）之和。

表78、表79和表81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：

——如果帧内预测划分方式为‘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 10）；

——如果帧内预测划分方式为‘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 11）；

——如果帧内预测划分方式为‘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 12）；

——如果帧内预测划分方式为‘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 13）；

——如果帧内预测划分方式为‘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 14）；

——如果帧内预测划分方式为‘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 15）；

——如果帧内预测划分方式为‘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 16）；

——如果帧间预测划分方式为‘NO_SPLIT’，1 个 2M×2N 的编码单元不进行划分（见图 17）。

图10到图16中矩形里的数字表示各预测块的PredBlockOrder。图17中矩形里的数字表示各帧间预测单元的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’，则其预测类型为帧内，其所包含的预测块的预测类型为帧内；否则该编码单元的编码单元的预测类型为帧间，其所包含的预测单元和预测块的预测类型为帧间。

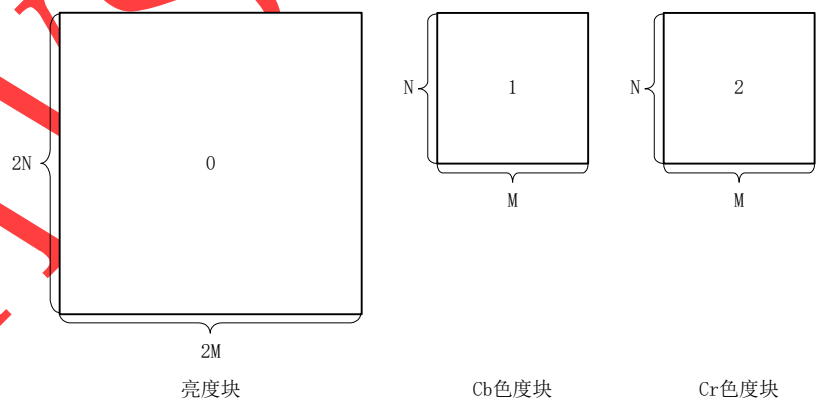


图10　帧内编码单元划分为预测块（预测划分方式‘NO_SPLIT’）

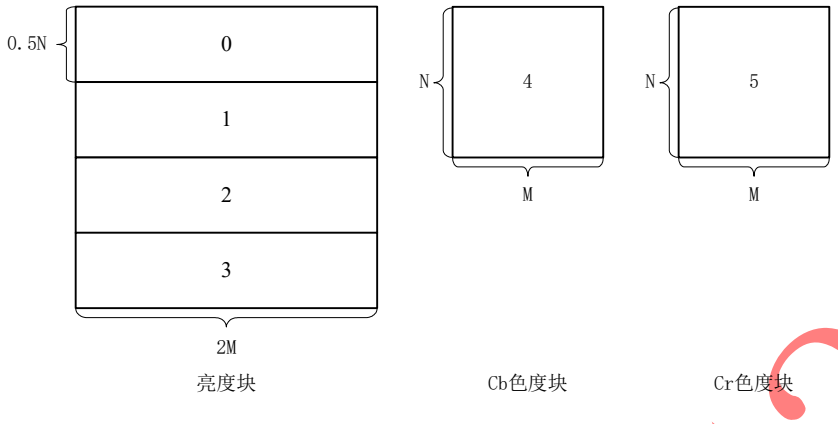


图11　帧内编码单元划分为预测块（划分类型‘HOR_tN’）

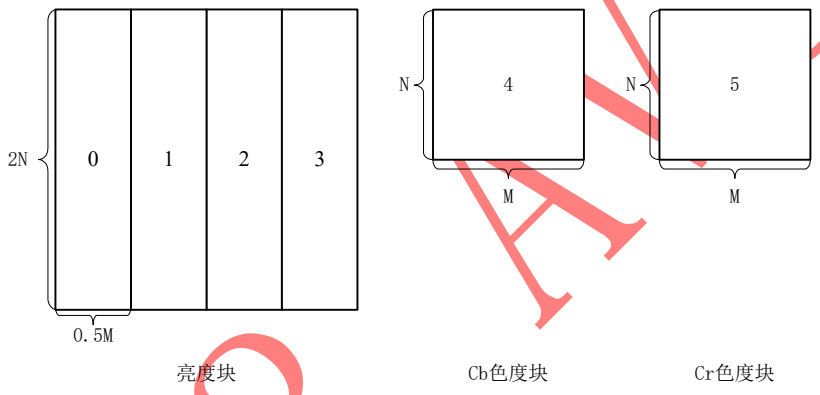


图12　帧内编码单元划分为预测块（划分类型‘VER_tN’）

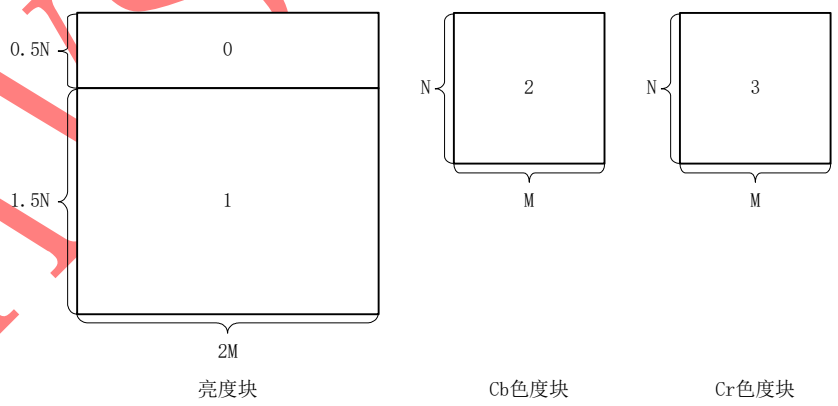


图13　帧内编码单元划分为预测块（划分类型‘HOR_UP’）

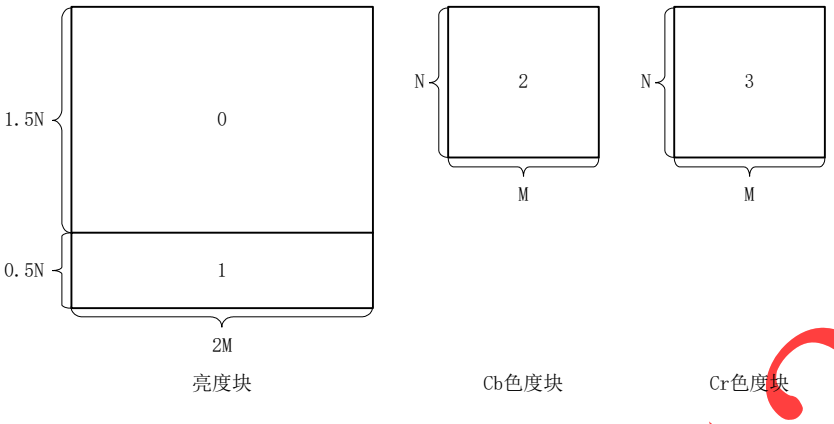


图14　帧内编码单元划分为预测块（划分类型‘HOR_DOWN’）

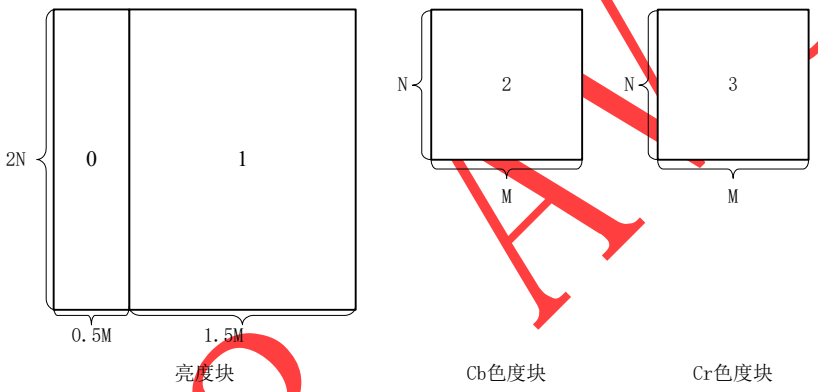


图15　帧内编码单元划分为预测块（划分类型‘VER_LEFT’）

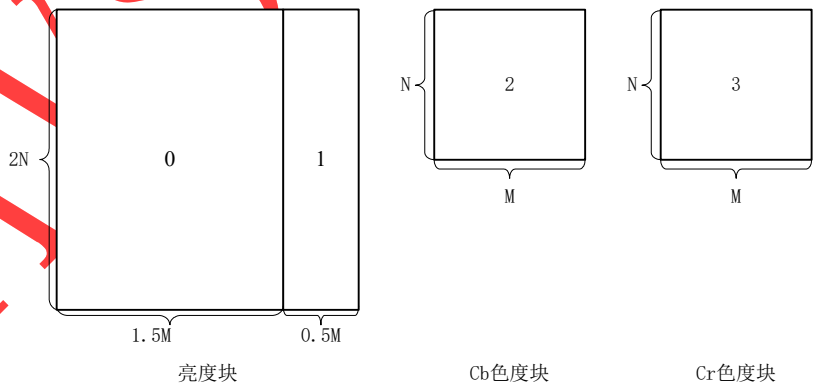


图16　帧内编码单元划分为预测块（划分类型‘VER_RIGHT’）

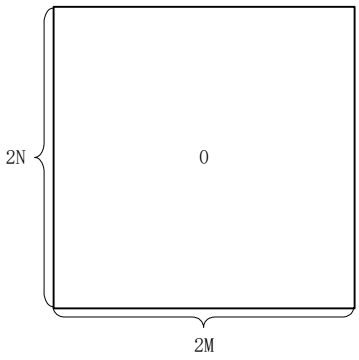


图17　帧间编码单元划分为预测单元（划分类型‘NO_SPLIT’）

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18。

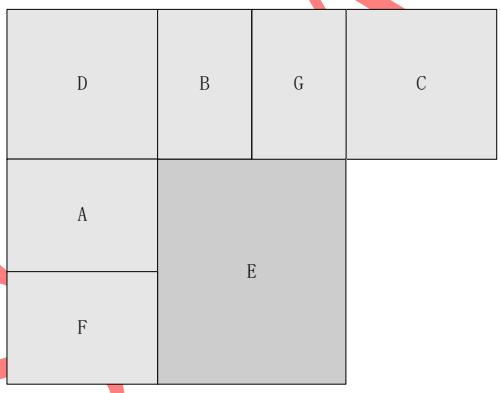


图18　块 E 和相邻块的空间位置关系

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：

——如果 PbtCuFlag 的值为 1，则 TransformSplitDirection 的值为 1。

——否则，如果当前编码单元是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；

——否则，如果当前编码单元是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；

——否则，TransformSplitDirection 的值为 0。

当前编码单元划分为变换块的方式如下：

——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 19），NumOfTransBlocks 的值为 3；

——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 20），NumOfTransBlocks 的值为 6；

——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 21），NumOfTransBlocks 的值为 6；

——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 22），NumOfTransBlocks 的值为 6。

图 19 到图 22 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：

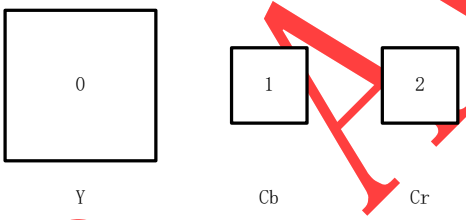


图19　编码单元划分为 1 个矩形亮度变换块和 2 个矩形色度变换块（4:2:0 格式）

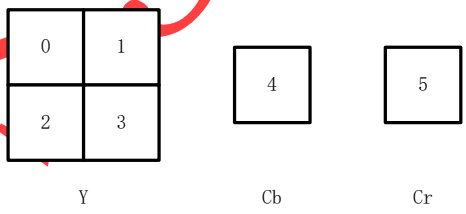


图20　编码单元划分为 4 个矩形亮度变换块和 2 个矩形色度变换块（4:2:0 格式）

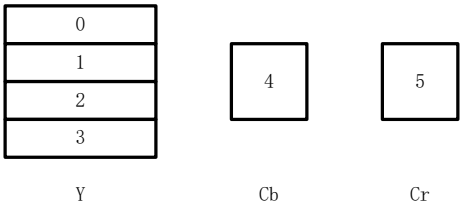


图21　编码单元划分为 4 个矩形亮度变换块和 2 个矩形色度变换块（4:2:0 格式）

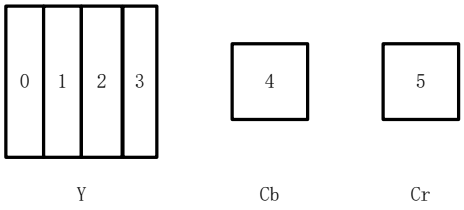


图22　编码单元划分为 4 个矩形亮度变换块和 2 个矩形色度变换块（4:2:0 格式）

9.5.6　帧内预测模式

当前编码单元的每一个预测块用以下方法确定其帧内预测模式：

a)　如果当前预测块 E 是亮度块：

1)　计算当前预测块预测模式的预测值：

- 如果左边预测块 A“存在”且为帧内预测块，则将 A 的 IntraLumaPredMode 赋值给 intraPredModeA；否则 intraPredModeA 等于 0。
- 如果上边预测块 B“存在”且为帧内预测块，则将 B 的 IntraLumaPredMode 赋值给 intraPredModeB；否则 intraPredModeB 等于 0。
- 如果 intraPredModeA 不等于 intraPredModeB，则 predIntraPredMode0 等于 Min(intraPredModeA, intraPredModeB)，predIntraPredMode1 等于 Max(intraPredModeA, intraPredModeB)；否则：
  1. 如果 intraPredModeA 等于 0，则 predIntraPredMode0 等于 0，predIntraPredMode1 等于 2。
  2. 如果 intraPredModeA 不等于 0，则 predIntraPredMode0 等于 0，predIntraPredMode1 等于 intraPredModeA。

2)　如果 intra_luma_pred_mode 的值为 0，则 IntraLumaPredMode 等于 predIntraPredMode0；否则，如果 intra_luma_pred_mode 的值为 1，则 IntraLumaPredMode 等于 predIntraPredMode1；否则：

- 如果 intra_luma_pred_mode 减 2 的值小于 predIntraPredMode0，则 IntraLumaPredMode 等于 intra_luma_pred_mode 减 2；
- 否则，如果 intra_luma_pred_mode 减 1 的值大于 predIntraPredMode0 并且 小于 predIntraPredMode1，则 IntraLumaPredMode 等于 intra_luma_pred_mode 减 1；
- 否则，IntraLumaPredMode 等于 intra_luma_pred_mode。

b)　如果当前预测块 E 是色度块：

1)　如果当前编码单元中 PredBlockOrder 的值为 0 的预测块的亮度预测模式 IntraLumaPredMode 等于 0、2、12 或 24，则 isRedundant 等于 1；否则 isRedundant 等于 0。

2)　如果 tscpm_enable_flag 的值等于‘1’且 intra_chroma_pred_mode 的值等于 1，则 IntraChromaPredMode 等于 5；

3)　否则，

- 如果 tscpm_enable_flag 的值等于‘1’且 intra_chroma_pred_mode 的值不等于 0，则 intra_chroma_pred_mode 的值减 1；
- 如果 isRedundant 等于 0，IntraChromaPredMode 等于 intra_chroma_pred_mode；否则，依次执行以下操作：

- 如果 IntraLumaPredMode 等于 0，则 predIntraChromaPredMode 等于 1；如果 IntraLumaPredMode 等于 2，则 predIntraChromaPredMode 等于 4；如果 IntraLumaPredMode 等于 12，则 predIntraChromaPredMode 等于 3；如果 IntraLumaPredMode 等于 24，则 predIntraChromaPredMode 等于 2。
- 如果 intra_chroma_pred_mode 的值等于 0，则 IntraChromaPredMode 等于 0；否则，如果 intra_chroma_pred_mode 的值小于 predIntraChromaPredMode，则 IntraChromaPredMode 等于 intra_chroma_pred_mode；否则 IntraChromaPredMode 等于 intra_chroma_pred_mode 加 1。

d) 根据 IntraLumaPredMode 的值，查表 84 得到亮度预测块的帧内预测模式。根据 IntraChromaPredMode 的值，查表 85 得到色度预测块的帧内预测模式。

表84 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
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

表85 色度预测块帧内预测模式

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

亮度预测块帧内预测模式见图23。

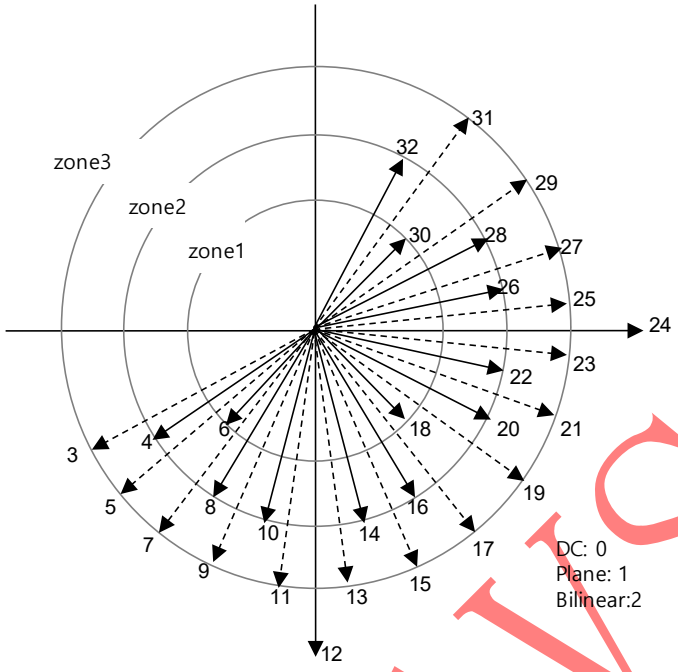


图23　亮度预测块帧内预测模式

9.5.7　参考图像选择

本条导出当前预测单元的参考索引值。当前预测单元的参考索引值表示当前预测单元进行解码所用的参考图像在参考图像队列中的编号。

如果当前图像是P图像，且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_Skip_Spatial_List0’或‘P_Direct_Temporal’或‘P_Direct_Spatial_list0’或‘P_Skip_Hmvp’或‘P_Direct_Hmvp’，则按9.5.8.6.2定义的方法导出RefIdxL0。

如果当前图像是P图像，且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’或‘P_Direct_Umve’，则按9.5.8.6.4定义的方法导出RefIdxL0。

如果当前图像是P图像，且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’或‘P_Direct_Affine’，则按9.5.8.6.6定义的方法导出RefIdxL0。

如果当前图像是B图像，且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_Skip_Spatial_list0’或‘B_Skip_Spatial_list1’或‘B_Skip_Spatial_list01’或‘B_Skip_Hmvp’或‘B_Direct_Temporal’或‘B_Direct_Spatial_list0’或‘B_Direct_Spatial_list1’或‘B_Direct_Spatial_list01’或‘B_Direct_Hmvp’，则按9.5.8.6.3定义的方法导出RefIdxL0和RefIdxL1。

如果当前图像是B图像，且当前预测单元所在的编码单元的编码单元子类型是‘B_Skip_Umve’或‘B_Direct_Umve’，则按9.5.8.6.4定义的方法导出RefIdxL0和RefIdxL1。

如果当前图像是B图像，且当前预测单元所在的编码单元的编码单元子类型是‘B_Skip_Affine’或‘B_Direct_Affine’，则按9.5.8.6.6定义的方法导出RefIdxL0和RefIdxL1。

如果当前预测单元的预测单元预测参考模式为‘PRED_List0’，则当前预测单元的L0参考索引等于RefIdxL0。

如果当前预测单元的预测单元预测参考模式为‘PRED_List1’，则当前预测单元的L1参考索引等于RefIdxL1。

如果当前预测单元的预测单元预测参考模式为‘PRED_List01’，则当前预测单元的L0参考索引和L1参考索引分别等于RefIdxL0和RefIdxL1。

9.5.8 运动信息

9.5.8.1 概述

如果两个运动矢量对应的参考图像队列相同，则两个运动矢量是同类型的运动矢量；否则是不同类型的运动矢量。

如果AffineFlag的值为0，运动信息包括运动矢量、预测参考模式、L0参考索引和L1参考索引；否则，运动信息包括运动矢量、仿射类型、预测参考模式、L0参考索引和L1参考索引。

如果参考图像是知识图像，该知识图像的距离索引DistanceIndex等于（当前图像的POI-1）乘2；其它参考图像的距离索引DistanceIndex和当前图像的距离索引DistanceIndex等于各自的POI乘2。

当前预测单元和它的运动矢量所指向的参考单元之间的距离BlockDistance等于当前预测单元所在图像的DistanceIndex减去参考单元所在图像的DistanceIndex的差。

9.5.8.2 运动信息导出

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：

- 当前预测单元所在编码单元的编码单元类型不是‘P_Skip’、‘P_Direct’、‘B_Skip’或‘B_Direct’；
- 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：

- 当前预测单元所在编码单元的编码单元类型不是‘P_Skip’、‘P_Direct’、‘B_Skip’或‘B_Direct’；
- 当前预测单元的预测参考模式是‘PRED_List1’或‘PRED_List01’。

决定运动矢量的基本单位：

——如果当前编码单元不是仿射模式，则根据 AmvrIndex 决定运动矢量的基本单位：

- 如果 AmvrIndex 的值为 0，运动矢量的基本单位是 1/4 个整数样本；
- 如果 AmvrIndex 的值为 1，运动矢量的基本单位是 1/2 个整数样本；
- 如果 AmvrIndex 的值为 2，运动矢量的基本单位是 1 个整数样本；
- 如果 AmvrIndex 的值为 3，运动矢量的基本单位是 2 个整数样本；
- 如果 AmvrIndex 的值为 4，运动矢量的基本单位是 4 个整数样本。

——如果编码单元是仿射模式，则根据 AffineAmvrIndex 决定运动矢量的基本单位：

- 如果 AffineAmvrIndex 的值为 0，运动矢量的基本单位是 1/4 个整数样本；
- 如果 AffineAmvrIndex 的值为 1，运动矢量的基本单位是 1 个整数样本；
- 如果 AffineAmvrIndex 的值为 2，运动矢量的基本单位是 1/16 个整数样本。

如果AffineFlag的值为0，按以下方法导出运动信息：

——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’，则按 9.5.8.6.2 定义的方法导出运动信息；

——否则：0

- 如果当前预测单元的 MvExistL0 为 1，则先按 9.5.8.4.2 定义的方法得到运动矢量预测值，再按 9.5.8.5.2 定义的方法解码得到 L0 运动矢量；

- 如果当前预测单元的MvExistL1为1,则先按9.5.8.4.2定义的方法得到运动矢量预测值，再按9.5.8.5.2定义的方法解码得到L1运动矢量。

——按9.15定义的方法用当前预测单元的运动信息更新历史运动信息表。

如果AffineFlag的值为1，按以下方法导出运动信息：

——如果当前预测单元所在编码单元的编码单元类型是‘P_Skip’、‘P_Direct’、‘B_Skip’或‘B_Direct’，则按9.5.8.6.6定义的方法先得到两个或三个运动矢量作为仿射控制点运动矢量组，再按9.17定义的方法得到各个子块的运动矢量集合；

——否则：

- 如果当前预测单元的MvExistL0为1，则先按9.5.8.4.3义的方法得到两个运动矢量预测值，然后按9.5.8.5.3定义的方法解码得到仿射控制点运动矢量组，再按9.17定义的方法得到各个子块的L0运动矢量集合；
- 如果当前预测单元的MvExistL1为1，则先按9.5.8.4.3定义的方法得到两个运动矢量预测值，然后按9.5.8.5.3定义的方法解码得到仿射控制点运动矢量组，再按9.17定义的方法得到各个子块的L1运动矢量集合。

——如果当前预测单元预测参考模式是‘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。

9.5.8.3 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18）：

——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。

9.5.8.4 运动矢量预测

9.5.8.4.1 概述

如果AffineFlag和ExtendMvrFlag的值均为0，运动矢量预测过程见9.5.8.4.2；如果AffineFlag的值为0且ExtendMvrFlag的值为1，运动矢量预测过程见9.5.8.4.4。如果AffineFlag的值为1，运动矢量预测过程见9.5.8.4.3。

9.5.8.4.2 普通运动矢量预测

当前预测单元的亮度预测块E的相邻亮度预测块A、B、C或D各块（见图18）的空域运动信息存储单元中与当前待预测运动矢量同类型的运动矢量记为mvA、mvB、mvC、mvD，对应的参考索引记为referenceIndexA、referenceIndexB、referenceIndexC、referenceIndexD，各块所在的预测单元对应的BlockDistance记为BlockDistanceA、BlockDistanceB、BlockDistanceC、BlockDistanceD。

依次执行以下操作：

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同类型的运动矢量，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同类型的运动矢量。

b) 如果 referenceIndexC 等于-1，则 mvC 等于 mvD，BlockDistanceC 等于 BlockDistanceD，referenceIndexC 等于 referenceIndexD。

当前待预测运动矢量预测值MVEPred计算过程如下：

a) 第一步，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) 第二步，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），计算过程结束；否则执行第三步。

c) 第三步，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
```

最后执行以下操作：

```
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。

9.5.8.4.3 仿射运动矢量预测

当前预测单元的亮度预测块E的相邻亮度预测块A、B、C、D、G各块的空域运动信息存储单元中与当前预测运动矢量同类型的运动矢量记为mvA、mvB、mvC、mvD、mvG，对应的参考索引记为referenceIndexA、referenceIndexB、referenceIndexC、referenceIndexD、referenceIndexG，各块所在的预测单元对应的 BlockDistance 记 为 BlockDistanceA 、 BlockDistanceB 、 BlockDistanceC 、 BlockDistanceD 、 BlockDistanceG， 。

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同类型的运动矢量。
b) 否则，mvX 等于相邻亮度预测块 X 与当前待预测运动矢量同类型的运动矢量，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。

当前待预测运动矢量预测值MvEPred(MvEPred_x, MvEPred_y)计算过程如下：

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(mvX_x,mvX_y)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(mvX_x,mvX_y)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(mvX_x,mvX_y)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。

当前待预测运动矢量预测值MvEPredAffine(MvEPredAffine_x, MvEPredAffine_y)计算过程如下：

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(mvX_x,mvX_y)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(mvX_x,mvX_y)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。

最后执行以下操作：

```
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。

9.5.8.4.4 扩展精度运动矢量预测

按以下步骤得到运动信息：

a) 令 hmvpIdx 的值等于 AmvrIndex；

b) 如果 CntHmvp 的值大于 hmvpIdx 的值，则取历史运动信息表中第(CntHmvp – hmvpIdx)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 hmvpIdx，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。

当前待预测运动矢量预测值MvEPred(MvEPred_x, MvEPred_y)计算过程如下：

a) 由预测参考模式 hmvpInterMode、当前预测单元的预测参考模式查表 86 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86 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。

9.5.8.5 运动矢量解码

9.5.8.5.1 概述

如果AffineFlag的值为0，运动矢量解码过程见9.5.8.5.2；否则，运动矢量解码过程见9.5.8.5.3。

9.5.8.5.2 普通运动矢量解码

按以下步骤解码得到运动矢量：

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。

9.5.8.5.3 仿射运动矢量解码

按以下步骤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 9.17 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 9.17 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 9.17 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。

9.5.8.6 跳过模式和直接模式的运动信息导出

9.5.8.6.1 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’、‘P_Skip_Spatial_List0’、‘P_Direct_Temporal’、‘P_Direct_Spatial_List0’、‘P_Skip_Hmvp’或‘P_Direct_Hmvp’，则按9.5.8.6.2定义的方法导出运动信息。

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’、‘B_Skip_Spatial_list0’、‘B_Skip_Spatial_list1’、‘B_Skip_Spatial_list01’、‘B_Skip_Hmvp’或‘B_Direct_Temporal’、‘B_Direct_Spatial_list0’、‘B_Direct_Spatial_list1’、‘B_Direct_Spatial_list01’或‘B_Direct_Hmvp’，则按9.5.8.6.3定义的方法导出运动矢量。

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’，则按9.5.8.6.4定义的方法导出运动矢量。

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’，则按9.5.8.6.6定义的方法导出仿射控制点运动矢量。

9.5.8.6.2、9.5.8.6.3和9.5.8.6.6中使用的运动信息存储单元的定义见9.13。9.5.8.6.4中判断运动信息是否相同的方法见9.14。

9.5.8.6.2 运动信息导出方法1

9.5.8.6.2.1 概述

根据CuSubTypeIdx可导出运动信息motionInfoX(mvE0, mvE1, refIdxL0, refIdxL1, interPredRefMode)，其中X等于0或1。

如果CuSubTypeIdx等于0，则按方法1.1（见9.5.8.6.2.2）导出motionInfo0，并分别对mvE、refIdxL0和interPredRefMode赋值。

如果CuSubTypeIdx等于1，则按方法1.2（见9.5.8.6.2.3）导出motionInfo1，并分别对mvE、refIdxL0和interPredRefMode赋值。

如果CuSubTypeIdx大于或等于2，则按方法1.1和方法1.2分别导出motionInfo0和motionInfo1，然后按照9.5.8.6.5定义的方法导出运动矢量候选项motionInfoHmvp。

如果interPredRefMode为‘PRED_List0’，则分别对mvE、refIdxL0和interPredRefMode赋值。

9.5.8.6.2.2 方法1.1

导出运动信息的步骤如下：

a) 第一步，

1) 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参考帧索引为-1，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RefIdxL0等于0，结束运动信息导出过程。

2) 否则，

- 当前预测单元的L0参考索引等于0，当前预测单元的L0参考索引对应的参考图像的距离索引记为DistanceIndexL0，当前预测单元的L0参考索引对应的参考图像的BlockDistance记为BlockDistanceL0。
- 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。

b) 第二步，

```
BlockDistanceRef = DistanceIndexCol - DistanceIndexRef
```

c) 第三步，令当前预测单元的L0参考索引RefIdxL0等于0，计算当前预测单元的L0运动矢量mvE0(mvE0_x, mvE0_y)：

```
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))
```

d) 第四步，interPredRefMode 的值等于‘PRED_List0’。

9.5.8.6.2.3 方法 1.2

导出运动信息的步骤如下：

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 refIdxL0；

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 refIdxL0 的值等于 0；

c) interPredRefMode 的值等于‘PRED_List0’，refIdxL1 的值等于-1，mvE1 为零矢量。

9.5.8.6.3 运动信息导出方法 2

9.5.8.6.3.1 概述

根据CuSubTypeIdx可导出motionInfoX(mvE0, mvE1, refIdxL0, refIdxL1, interPredRefMode)，其中X等于0、1、2、3或4。

如果CuSubTypeIdx等于0，则按方法2.1（见9.5.8.6.3.2）导出motionInfo0，并分别对mvE0、mvE1、refIdxL0、refIdxL1和interPredRefMode赋值；

如果CuSubTypeIdx等于1，则按方法2.2（见9.5.8.6.3.3）导出motionInfo1，并分别对mvE0、mvE1、refIdxL0、refIdxL1和interPredRefMode赋值；

如果CuSubTypeIdx等于2，则按方法2.3（见9.5.8.6.3.4）导出motionInfo2，并分别对mvE、refIdxL1和interPredRefMode赋值；

如果CuSubTypeIdx等于3，则按方法2.4（见9.5.8.6.3.5）导出motionInfo3，并分别对mvE、refIdxL0和interPredRefMode赋值；

如果CuSubTypeIdx大于或等于4，则先按方法2.1、2.2、2.3和2.4分别导出motionInfo0、motionInfo1、motionInfo2和motionInfo3，再按9.5.8.6.5定义的方法导出运动矢量候选项motionInfoHmvp；

如果interPredRefMode为‘PRED_List01’，则分别对mvE0、mvE1、refIdxL0、refIdxL1和interPredRefMode赋值；如果interPredRefMode为‘PRED_List0’，则分别对mvE、refIdxL0和interPredRefMode赋值；如果interPredRefMode为‘PRED_List1’，则分别对mvE、refIdxL1和interPredRefMode赋值。

9.5.8.6.3.2 方法 2.1

导出运动信息的步骤如下：

a) 第一步，

1) 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参考帧索引为-1，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，然后将根据 9.5.8.4 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 RefIdxL0 和 L1 参考索引 RefIdxL1 均等于 0，结束运动信息导出过程。

2) 否则，

- 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。
- 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。

b) 第二步，

```
BlockDistanceRef = DistanceIndexCol - DistanceIndexRef
```

c) 第三步，

1) 令当前预测单元的 L0 参考索引 RefIdxL0 等于 0，计算当前预测单元的 L0 运动矢量 mvE0(mvE0_x, mvE0_y)：

```
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))
```

此时 9.5.8.4 中的 mvX 为 mvRef，MVX 为 mvE0。

2) 令当前预测单元的 L1 参考索引 RefIdxL1 等于 0，计算当前预测单元的 L1 运动矢量 mvE1(mvE1_x, mvE1_y)：

```
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))
```

此时 9.5.8.4 中的 mvX 为 mvRef，MVX 为 mvE1。

d) 第四步，interPredRefMode 的值等于‘PRED_List01’。

9.5.8.6.3.3 方法 2.2

导出运动信息的步骤如下：

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 refIdxL0 和 L1 参考索引 refIdxL1；

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

refIdxL0；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 refIdxL1；

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 refIdxL0 和 L1 参考索引 refIdxL1 的值均等于 0；

d) interPredRefMode 的值等于‘PRED_List01’。

9.5.8.6.3.4 方法 2.3

导出运动信息的步骤如下：

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 refIdxL1；

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 refIdxL1；

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 refIdxL1 的值等于 0；

d) interPredRefMode 的值等于‘PRED_List1’，refIdxL0 的值等于-1，mvE0 为零矢量。

9.5.8.6.3.5 方法 2.4

导出运动信息的步骤如下：

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 refIdxL0；

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 refIdxL0；

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 refIdxL0 的值等于 0；

d) interPredRefMode 的值等于‘PRED_List0’，refIdxL1 的值等于-1，mvE1 为零矢量。

9.5.8.6.4 高级运动矢量表达运动信息导出方法

导出MvE0、MvE1、RefIdxL0，RefIdxL1和InterPredRefMode的步骤如下：

第一步，F、G、C、A和D是当前预测单元E的相邻预测块（见图18），确定mvBaseE0、mvBaseE1、RefIdxL0、RefIdxL1和InterPredRefMode：

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 UmveMvIdx 等于 1，则预测参考模式 UmveInterMode 的值为 0，L0 参考索引 UmveRef0 的值为 0，L0 运动矢量 UmveMv0 为零矢量，MvBaseE0 等于 UmveMv0，MvBaseE1 不存在，RefIdxL0 等于 UmveRef0，InterMode 等于 UmveInterMode。

g) 否则，如果 F、G、C、A、D 中有 UmveMvIdx+1 个“可用”，记第 UmveMvIdx+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，RefIdxL0 等于 mvUmveBaseRef0，InterPredRefMode 等于 mvUmveBaseInterPredRefMode。

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，RefIdxL1 等于 mvUmveBaseRef1，InterPredRefMode 等于 mvUmveBaseInterPredRefMode。

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，RefIdxL0 等于 mvUmveBaseRef0，RefIdxL1 等于 mvUmveBaseRef1，InterPredRefMode 等于 mvUmveBaseInterPredRefMode。

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 9.5.8.6.2.2 定义的方法导出运动矢量 mvUmveBaseMv0、参考帧索引 mvUmveBaseRef0，并令 mvUmveBaseInterPredRefMode 等于 0，mvBaseE0 等于 mvUmveBaseMv0，mvBaseE1 不存在，RefIdxL0 等于 mvUmveBaseRef0，RefIdxL1 等于 -1，InterPredRefMode 等于 mvUmveBaseInterPredRefMode。

2) 如果编码单元子类型是‘B_Skip_Umve’或‘B_Direct_Umve’，按 9.5.8.6.3.2 定义的方法导出 L0 参考索引 UmveRef0、L0 运动矢量 UmveMv0、L1 参考索引 UmveRef1 和 L1 运动矢量 UmveMv1，并令 UmveRefMode 等于 2，mvBaseE0 等于 UmveMv0，mvBaseE1 等于 UmveMv1，RefIdxL0 等于 mvUmveBaseRef0，RefIdxL1 等于 -1，InterPredRefMode 等于 mvUmveBaseInterPredRefMode。

第二步，确定运动矢量偏移量mvOffset0和mvOffset1：

a) 根据 UmveStepIdx 的值查表 87 得到运动矢量偏移量 UmveOffset。

b) 如果 InterPredRefMode 的值为 0，则 mvOffset0 的值等于 UmveOffset，mvOffset1 不存在。

c) 否则，如果 InterPredRefMode 的值为 1，则 mvOffset1 的值等于 UmveOffset，mvOffset0 不存在。

d) 否则，如果 InterPredRefMode 的值为 2，则参考图像队列 0 中参考索引为 RefIdxL0 的图像和参考图像队列 1 中参考索引为 RefIdxL1 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) 根据 9.5.8.1 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87 UmveOffset 与 UmveStepIdx 的对应关系

UmveStepIdx	UmveOffset
0	1
1	2
2	4
3	8
4	16

第三步，导出运动矢量：

a) 如果 InterPredRefMode 的值为 0，则当前预测单元的 mvE0(mvE0_x, mvE0_y)按以下方法导出：

```
if ((UmveDirIdx == 0) || (UmveDirIdx == 1)) {
    mvE0_x = mvBaseE0_x + (-1)^UmveDirIdx * mvOffset0
    mvE0_y = mvBaseE0_y
}
else if ((UmveDirIdx == 2) || (UmveDirIdx == 3)) {
    mvE0_x = mvBaseE0_x
    mvE0_y = mvBaseE0_y + (-1)^UmveDirIdx * mvOffset0
}
mvE0_x = Clip3(-32768, 32767, mvE0_x)
mvE0_y = Clip3(-32768, 32767, mvE0_y)
```

MvE 等于 mvE0。

b) 如果 InterPredRefMode 的值为 1，则当前预测单元的 mvE1(mvE1_x, mvE1_y)按以下方法导出：

```
if ((UmveDirIdx == 0) || (UmveDirIdx == 1)) {
    mvE1_x = mvBaseE1_x + (-1)^UmveDirIdx * mvOffset1
    mvE1_y = mvBaseE1_y
}
else if ((UmveDirIdx == 2) || (UmveDirIdx == 3)) {
    mvE1_x = mvBaseE1_x
```

```
        mvE1_y = mvBaseE1_y + (-1)^UmveDirIdx * mvOffset1
    }
    mvE1_x = Clip3(-32768, 32767, mvE1_x)
    mvE1_y = Clip3(-32768, 32767, mvE1_y)
```

MvE 等于 mvE1。

c) 如果 InterPredRefMode 的值为 2，则当前预测单元的 mvE0(mvE0_x, mvE0_y)和 mvE1(mvE1_x, mvE1_y)按以下方法导出：

```
if ((UmveDirIdx == 0) || (UmveDirIdx == 1)) {
    mvE0_x = mvBaseE0_x + (-1)^UmveDirIdx * mvOffset0
    mvE0_y = mvBaseE0_y
    mvE1_x = mvBaseE1_x + (-1)^UmveDirIdx * mvOffset1
    mvE1_y = mvBaseE1_y
}
else if ((UmveDirIdx == 2) || (UmveDirIdx == 3)) {
    mvE0_x = mvBaseE0_x
    mvE0_y = mvBaseE0_y + (-1)^UmveDirIdx * mvOffset0
    mvE1_x = mvBaseE1_x
    mvE1_y = mvBaseE1_y + (-1)^UmveDirIdx * 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)
```

MvE0 等于 mvE0，MvE1 等于 mvE1。

RefIdxL0 等于 mvUmveBaseRef0，RefIdxL1 等于 mvUmveBaseRef1，InterPredRefMode 等于 mvUmveBaseInterPredRefMode。

9.5.8.6.5 历史运动信息候选项导出方法

根据历史运动信息表HmvpCandidateList和motionInfoX（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），按以下步骤导出历史运动信息候选项motionInfoHmvp：

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 hmvpIdx 初始化为 1。如果当前图像是 P 图像，则 candIdx 等于 1；如果当前图像是 B 图像，则 candIdx 等于 3。

b) 如果 numAllowedCand 等于 0，结束导出过程。如果当前图像是 P 图像，则 motionInfoHmvp 等于 motionInfo1；如果当前图像是 B 图像，则 motionInfoHmvp 等于 motionInfo3。

c) 否则，执行以下操作，直到 candIdx 等于 CuSubTypeIdx，或 hmvpIdx 大于 NumAllowedCand：

1) 令 tmpHmvp 等于 HmvpCandidateList[CntHmvp-hmvpIdx]。

2) 根据 9.14 定义的方法依次判断 tmpHmvp 与 motionInfoX（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）是否相同。如果所有运动信息均不相同，则 candIdx 加 1，hmvpIdx 加 1；否则，hmvpIdx 加 1。

d) 如果 candIdx 等于 CuSubTypeIdx，则 motionInfoHmvp 等于 tmpHMVP；如果 candIdx 小于 CuSubTypeIdx，则 motionInfoHmvp 等于 HmvpCandidateList[cntHmvp-1]。

9.5.8.6.6 仿射运动信息导出方法

如果当前预测单元E的相邻预测块X（X为F、G、C、A、B或D）存在且为帧间预测模式，则相邻块X“可用”；否则相邻块X“不可用”。其中，F、G、C、A、B、D与E的关系见图18。

如果当前图像是P图像，记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24）。如果亮度样本H所在的时域运动信息存储单元存储的参考索引为-1，则H“不存在”；否则，H“存在”。

如果当前图像是B图像，记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25）。如果亮度样本H所在的时域运动信息存储单元存储的参考索引为-1，则H“不存在”；否则，H“存在”。

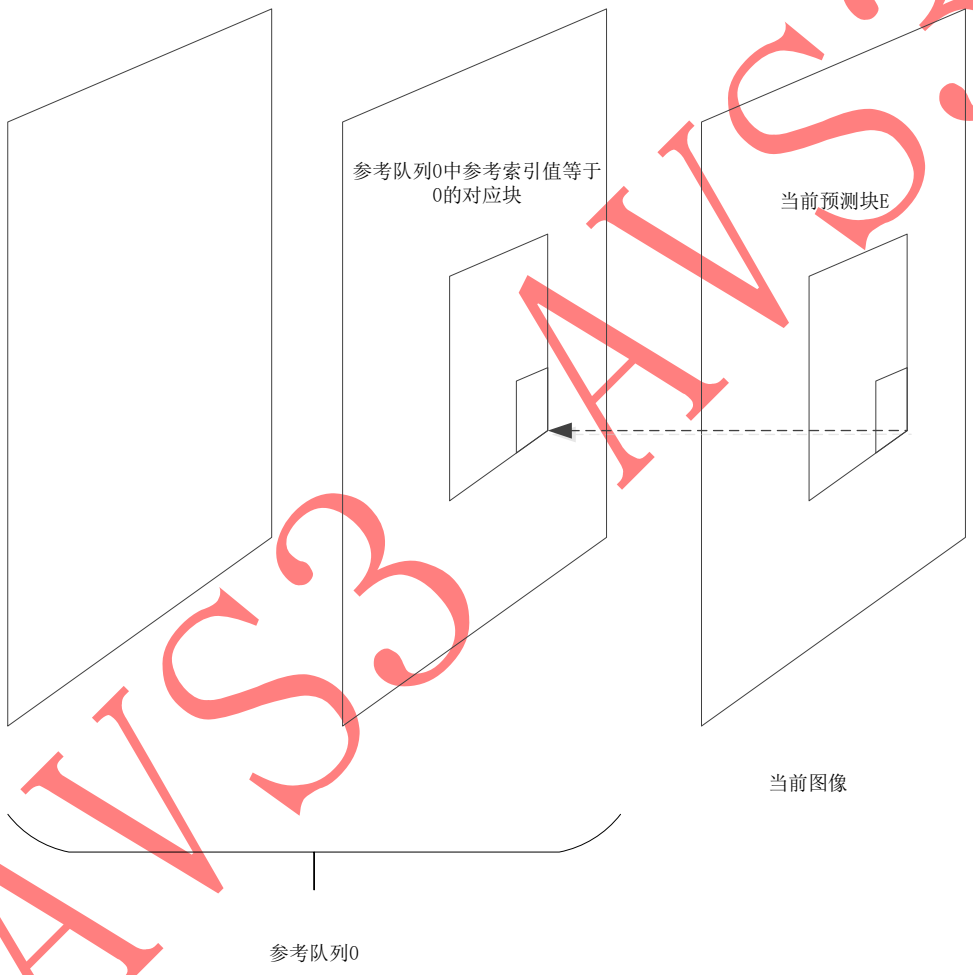


图24 预测单元 E 和相邻亮度样本 H 的空间位置关系（当前图像是 P 图像）

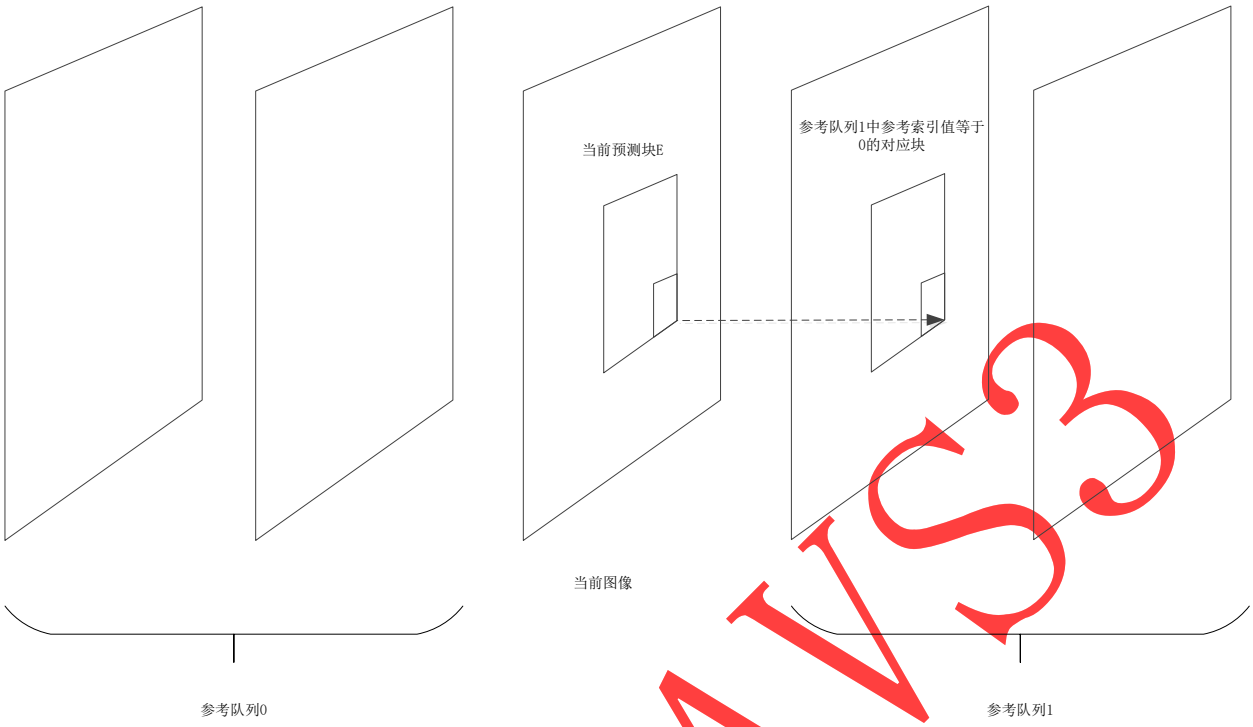


图25　预测单元 E 和相邻亮度样本 H 的空间位置关系（当前图像是 B 图像）

按以下步骤导出当前预测单元E的仿射运动信息：

a)　第一步，

1)　当 AffineCandIdx 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 AffineCandIdx+1 个“可用”且不同的仿射预测单元，记第 AffineCandIdx+1 个可用的仿射预测单元包括的预测块为 X；否则，执行第二步。

2)　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。

3)　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 9.16 定义的方法得到 E 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。

4)　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 9.16 定义的方法得到 E 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。

5)　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 9.16 定义的方法得到 E 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。

b)　第二步，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。

c)　第三步，

1)　如果 A、B 和 D 中至少有一个“可用”，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”。

2) 如果 G 和 C 中至少有一个“可用”，则按 G、C 的顺序扫描得到第一个可用的相邻块 X1；否则，X1“不存在”。
3) 如果 F“可用”，则记相邻块 F 为 X2；否则，X2“不存在”。
4) 如果 H“存在”，则记 H 为 X3；否则，X3“不存在”。

d) 第四步，
1) 如果 X3“存在”且当前图像是 B 图像，则按 9.5.8.6.3.2 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 9.5.8.6.2.2 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”。
2) 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”。
3) 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”。
4) 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。
5) 否则，MVX2_1_X（X 为 L0 或 L1）“不存在”。

e) 第五步，
1) 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) }
```

2) 判断运动矢量候选项 mvsX（X 为 1、2、3、4、5、6）是否“可用”：
- 如果 mvsX_L0“可用”且 mvsX_L1“不可用”，则 mvsX 可用且 mvsX 只包含 mvsX_L0；
- 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；
- 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；
- 否则，mvsX“不可用”。

3) 如果“可用”候选项的数量大于或等于 AffineCandIdx-AffineStartOffset+1，则按 mvs1、mvs2、mvs3、mvs4、mvs5、mvs6 的顺序取第 AffineCandIdx-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”。

4) 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 RefIdx0 等于 mvs_L0 对应的参考索引 refIdx0。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。将 mvs_L0 作为仿射控制点运动矢量组并按 9.17 定义的方法得到当前预测单元的 L0 运动矢量集合（记作 MvArrayL0）。MvArrayL0 集合由所有亮度预测子块的 L0 运动矢量组成。

5) 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 RefIdx1 等于 mvs_L1 对应的参考索引 refIdx1。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。将 mvs_L1 作为仿射控制点运动矢量组并按 9.17 定义的方法得到当前预测单元的 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。

6) 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 RefIdx0 和 L1 参考索引 RefIdx1 分别等于 mvs_L0 和 mvs_L1 对应的参考索引 refIdx0 和 refIdx1。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。分别将 mvs_L0 和 mvs_L1 作为仿射控制点运动矢量组并按 9.17 定义的方法得到当前预测单元的 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。

7) 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 RefIdxL0 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。由 mvsAffine 按 9.17 定义的方法得到当前预测单元的 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。

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。从符合本部分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由式（33）得到：

$$CoeffMatrix[i][j] = Clip3(-32768, 32767, (((((QuantCoeffMatrix[i][j] \times WeightQuantMatrix_{M1xM2}[i][j]) >> WqmShift) \times (DequantTable(QP_X))>>4) + 2^{((ShiftTable(QPx)+shift1)-1)}) >> (ShiftTable(QP_X) + shift1))), \quad i=0\sim M_1-1, j=0\sim M_2-1 \cdots\cdots\cdots\cdots\cdots\cdots (33)$$

式（33）中shift1等于m + BitDepth – 14（BitDepth是编码样本精度。m=Log(M_1×M_2)/2）。

如果满足以下条件之一，则二维变换系数矩阵的值按式（34）进行修正。

——M_1是M_2的两倍或M_2是M_1的两倍；

——M_1是M_2的八倍或M_2是M_1的八倍。

$$CoeffMatrix[i][j] = (CoeffMatrix[i][j] \times 181 + shift2) >> 8 \cdots\cdots\cdots\cdots\cdots\cdots (34)$$

式（34）中，shift2 = 1 << 7。

如果帧内预测模式是‘Intra_Luma_PCM’或‘Intra_Chroma_PCM’，则CoeffMatrix[i][j] = QuantCoeffMatrix[i][j]（i=0～M_1-1，j=0～M_2-1）。

量化参数QP_X（X为Y、Cb或Cr）与DequantTable和ShiftTable的关系见表88。

表88　QPX 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
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

表 88（续）

QP_X的值	DequantTable(QP_X)的值	ShiftTable(QP_X)的值
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
54	55109	8
55	60097	8
56	32771	7
57	35734	7
58	38965	7

表 88（续）

QP_X的值	DequantTable(QP_X)的值	ShiftTable(QP_X)的值
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

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。如果帧内预测模式既不是‘Intra_Luma_PCM’也不是‘Intra_Chroma_PCM’，则步骤如下：

a) 如果当前变换块是亮度帧内预测残差块，M_1或M_2的值大于4且SecondaryTransformEnableFlag的值等于1，对CoeffMatrix执行以下操作：

1) 由变换系数矩阵得到4×4矩阵C：

$$c_{ij} = coeff_{ij}, \quad i=0\sim3, j=0\sim3$$

式中c_{ij}是矩阵C的元素，$coeff_{ij}$是变换系数矩阵的元素。

2) 如果IntraLumaPredMode的值为0～2或13～32，且9.7.2中坐标为(x0-1, y0+j-1)（j=1～N）的参考样本“可用”，则：

$$c_{ij} = \text{Clip3}(-32768, 32767, (p_{ij} + 2^6) >> 7), \quad i=0\sim3, j=0\sim3$$

式中p_{ij}是4×4矩阵P的元素。矩阵P的计算如下：

$$P = C \times S_4$$

其中S_4是4×4反变换矩阵。

3) 如果IntraLumaPredMode的值为0～23，且9.7.2中坐标为(x0+i-1, y0-1)（i=1～M）的参考样本“可用”，则：

$$c_{ij} = \text{Clip3}(-32768,\ 32767,\ (q_{ij} + 2^6) >> 7),\ i=0\sim3, j=0\sim3$$

式中 q_{ij}是 4×4 矩阵 Q的元素。矩阵 Q 的计算如下：

$$Q = S_4^T \times C$$

式中 S_4^T是 S_4的转置。

4) 根据矩阵 C 修改变换系数矩阵的元素的值：

$$\text{coeff}_{ij} = c_{ij},\ i=0\sim3, j=0\sim3$$

b) 对变换系数矩阵进行如下垂直反变换，得到矩阵 K：

1) 如果当前变换块是亮度帧内预测残差块，M_1 和 M_2 的值均等于 4 且 SecondaryTransformEnableFlag 的值等于 1，则：

$k_{ij} = \text{Clip3}(-32768,\ 32767,\ (v_{ij} + 2^4)>>5),\ i=0\sim M_1-1, j=0\sim M_2-1$

式中 v_{ij}是矩阵 V 的元素，k_{ij}是矩阵 K的元素。矩阵 V的计算如下：

$$V = D_4^T \times \text{CoeffMatrix}$$

式中 D_4^T是反变换矩阵 D_4的转置矩阵。

2) 否则，判断 PbtCuFlag 的值，

a) 如果 PbtCuFlag 的值为 1 且变换块的顺序号为 0、1、2 或 3：

$k_{ij} = \text{Clip3}(-32768,\ 32767,\ (v_{ij} + 2^4)>>5),\ i=0\sim M_1-1, j=0\sim M_2-1$

式中 v_{ij}是矩阵 V的元素，k_{ij}是矩阵 K的元素。矩阵 V的计算如下：

- 如果变换块的顺序号为 0 或 1：

$$V = DCT8_{M2}^T \times \text{CoeffMatrix}$$

式中 $DCT8_{M2}^T$是 $DCT8_{M2}$的转置矩阵，$DCT8_{M2}$是 $M_2\times M_2$ 反变换矩阵。

- 否则，如果变换块的顺序号为 2 或 3：

$$V = DST7_{M2}^T \times \text{CoeffMatrix}$$

式中 $DST7_{M2}^T$是 $DST7_{M2}$的转置矩阵，$DST7_{M2}$是 $M_2\times M_2$ 反变换矩阵。

b) 否则：

$k_{ij} = \text{Clip3}(-32768,\ 32767,\ (v_{ij} + 2^4)>>5),\ i=0\sim M_1-1, j=0\sim M_2-1$

式中 v_{ij}是矩阵 V的元素，k_{ij}是矩阵 K的元素。矩阵 V的计算如下：

$$V = DCT2_{M2}^T \times \text{CoeffMatrix}$$

式中 $DCT2_{M2}^T$是 $DCT2_{M2}$的转置矩阵，$DCT2_{M2}$是 $M_2\times M_2$ 反变换矩阵。

c) 对矩阵 K进行如下水平反变换，得到矩阵 H：

1) 如果当前变换块是亮度帧内预测残差块，M_1 和 M_2 的值均等于 4 且 SecondaryTransformEnableFlag 的值等于 1，则：

$h_{ij} = \text{Clip3}(-\text{MaxValue}-1,\ \text{MaxValue},\ (w_{ij} + 2^{\text{shift1}-1})>>\text{shift1}),\ i=0\sim M_1-1, j=0\sim M_2-1$

式中 w_{ij}是矩阵 W 的元素，h_{ij}是矩阵 H的元素。MaxValue 的值为(1 << BitDepth) - 1，shift1 等于 22 - BitDepth。矩阵 W的计算如下：

$$W = K \times D_4$$

式中 D_4是反变换矩阵。

2) 否则，判断 PbtCuFlag 的值，

a) 如果 PbtCuFlag 的值为 1 且变换块的顺序号为 0、1、2 或 3：

$h_{ij} = \text{Clip3}(-\text{MaxValue}-1,\ \text{MaxValue},\ (w_{ij} + 2^{\text{shift1}-1})>>\text{shift1})\quad i=0\sim M_1-1, j=0\sim M_2-1$

式中 w_{ij}是矩阵 W 的元素，h_{ij}是矩阵 H的元素。MaxValue 的值为(1 << BitDepth) - 1，shift1 等于 20 - BitDepth。矩阵 W的计算如下：

- 如果变换块的顺序号为 0 或 2：

$$W = K \times DCT8_{M1}$$

- 否则，如果变换块的顺序号为 1 或 3：

$$W = K \times DST7_{M1}$$

式中 $DCT8_{M1}$、$DST7_{M1}$ 是 $M_1 \times M_1$ 反变换矩阵。

b) 否则：

h_{ij} = Clip3(-MaxValue-1, MaxValue, $(w_{ij} + 2^{shift1-1})>>shift1$)　i=0～$M_1$-1, j=0～$M_2$-1

式中 w_{ij} 是矩阵 W 的元素，h_{ij} 是矩阵 H 的元素。MaxValue 的值为(1 << BitDepth) - 1，shift1 等于 20 - BitDepth。矩阵 W 的计算如下：

$$W = K \times DCT2_{M1}$$

式中 $DCT2_{M1}$ 是 $M_1 \times M_1$ 反变换矩阵。

d) 把矩阵 H 直接作为残差样值矩阵 ResidueMatrix，结束反变换操作。

二次变换反变换矩阵如下：

$$S_4 = \begin{bmatrix} 123 & -35 & -8 & -3 \\ -32 & -120 & 30 & 10 \\ 14 & 25 & 123 & -22 \\ 8 & 13 & 19 & 126 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 34 & 58 & 72 & 81 \\ 77 & 69 & -7 & -75 \\ 79 & -33 & -75 & 58 \\ 55 & -84 & 73 & -28 \end{bmatrix}$$

DCT2型变换的变换核尺寸范围是4×4~64×64，DCT8型变换的变换核尺寸范围是4×4~16×16，DST7型变换的变换核尺寸范围是4×4~16×16。反变换矩阵见附录F。

如果帧内预测模式是‘Intra_Luma_PCM’或‘Intra_Chroma_PCM’则ResidueMatrix[i][j] = CoeffMatrix[i][j] << (BitDepth – SamplePrecision)（i=0～M_1-1，j=0～M_2-1）。

9.7 帧内预测

9.7.1 概述

本条定义M×N亮度块和M×N色度块的帧内预测过程。

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。

分别使用9.7.2和9.7.3定义的方法获得当前块的亮度和色度参考样本，并分别使用9.7.4和9.7.5定义的方法进行亮度和色度帧内预测，从而分别得到M×N亮度块和M×N色度块的预测样本矩阵predMatrix。

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]，如果i大于2M，则r[i]=r[2M]，如果j大于2N，则c[j]=c[2N]。

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]，如果i大于2M，则row[i]=row[2M]，如果j大于2N，则col[i]=col[2N]。

9.7.2 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”；否则

2) 如果 c[1]“可用”并且 r[1]“不可用”，则 r[0]等于 c[1]，r[0]“可用”；否则

3) 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”。

9.7.3 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=0～2M，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 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=M+1～2M）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～N）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=N+1～2N）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”。

9.7.4 亮度预测块帧内预测

9.7.4.1 概述

如果IpfFlag的值为1，先使用9.7.4.2定义的方法得到orgPredMatrix，再进行帧内预测边界滤波（见9.7.4.3）得到predMatrix。

如果IpfFlag的值为0，使用9.7.4.2定义的方法得到predMatrix。

9.7.4.2 预测方法

根据IntraLumaPredMode决定亮度块帧内预测方法如下：

a) IntraLumaPredMode 等于 12（Intra_Luma_Vertical 预测）

orgPredMatrix[x][y] = r[x+1]（x=0～M-1, y=0～N-1）

predMatrix[x][y] = orgPredMatrix[x][y]（x=0～M-1, y=0～N-1）

b) IntraLumaPredMode 等于 24（Intra_Luma_Horizontal 预测）

orgPredMatrix[x][y] = c[y+1]（x=0～M-1, y=0～N-1）

predMatrix[x][y] = orgPredMatrix[x][y]（x=0～M-1, y=0～N-1）

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

$$orgPredMatrix[x][y]= ((\sum_{i=1}^{M} r[i]+\sum_{j=1}^{N} c[j]+((M+N)>>1))\times(4096/(M+N)))>>12$$

（x=0～M-1, y=0～N-1）

predMatrix[x][y]= orgPredMatrix[x][y]（x=0～M-1, y=0～N-1）

2) 否则，如果 r[i]（i=1～M）“可用”，且 c[j]（j=1～N）“不可用”，则

$$orgPredMatrix[x][y]= (\sum_{i=1}^{M} r[i]+(M>>1))>>\mathrm{Log}(M)$$（x=0～M-1, y=0～N-1）

predMatrix[x][y] = orgPredMatrix[x][y]（x=0～M-1, y=0～N-1）

3) 否则，如果 c[j]（j=1～N）“可用”，且 r[i]（i=1～M）“不可用”，则

$$orgPredMatrix[x][y]= (\sum_{j=1}^{N} c[j]+(N>>1))>>\mathrm{Log}(N)$$（x=0～M-1, y=0～N-1）

predMatrix[x][y] = orgPredMatrix[x][y]（x=0～M-1, y=0～N-1）

4) 否则，

orgPredMatrix[x][y] = $2^{BitDepth-1}$（x=0～M-1, y=0～N-1；BitDepth 是编码样本精度）

predMatrix[x][y] = orgPredMatrix[x][y]（x=0～M-1, y=0～N-1）

d) IntraLumaPredMode 等于 1（Intra_Luma_Plane 预测）

orgPredMatrix[x][y] = (ia+(x-((M>>1)-1))×ib+(y-((N>>1)-1))×ic+16)>>5（x=0～M-1, y=0～N-1）。

predMatrix[x][y]= Clip1(orgPredMatrix[x][y])（x=0～M-1, y=0～N-1）。

式中，ia = (r[M]+c[N]) << 4,

ib = ((ih<<5) × imh + (1<<(ish-1))) >> ish,

ic = ((iv<<5) × imv + (1<<(isv-1))) >> isv,

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],

$$ih = \sum_{i=0}^{(M>>1)-1} (i+1)\times(r[(M>>1)+1+i]-r[(M>>1)-1-i])$$

，

$$iv = \sum_{i=0}^{(N>>1)-1} (i+1)\times(c[(N>>1)+1+i]-c[(N>>1)-1-i])$$。

e) IntraLumaPredMode 等于 2（Intra_Luma_Bilinear 预测）

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)，（x=0～M-1,y=0～N-1）。

predMatrix[x][y]=Clip1(orgPredMatrix[x][y])，（x=0～M-1,y=0～N-1）。

其中，ia = r[M]，ib = c[N]，如果 M 等于 N，则 ic=(ia+ib+1)>>1；否则 ic=(((ia<<Log(M))+(ib<<Log(N)))×weight+(1<<(ishift+5)))>>(ishift+6)，式中 ishift = Log(Min(M,N))，weight = bilinearWeight[Log(Max(M,N)/Min(M,N))-1]，bilinearWeight[3] = {21, 13, 7}。

f) IntraLumaPredMode 不等于 0、1、2、12、24（Intra_Luma_Angular 预测）

初始化 r[-1]=c[1]，r[-2]=c[2]，c[-1]=r[1]，c[-2]=r[2]。根据 IntraLumaPredMode 的值查表 89 得到 dx、dy、xyAxis、xySign、imx、isx、imy 和 isy。其中：

dx = dirDx[IntraLumaPredMode]
dy = dirDy[IntraLumaPredMode]
xyAxis = dirXYFlag[IntraLumaPredMode]
xySign = dirXYSign[IntraLumaPredMode]
imx = divDxy[IntraLumaPredMode][0]
isx = divDxy[IntraLumaPredMode][1]
imy = divDyx[IntraLumaPredMode][0]
isy = divDyx[IntraLumaPredMode][1]

然后依次执行以下步骤：

1) 如果 xySign小于0，则
   - 如果 xyAxis 等于 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；
   - 否则，xyAxis 等于 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)。
2) 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)：
   - 如果 iYy 小于或等于-1（参考上边像素），则 offset=offsetx，iX=iXx，iY=-1；
   - 否则（参考左边像素），offset=offsety，iX=-1，iY=iYy。
3) 如果iY等于-1（参考上边像素）：
   - 如果(dx×dy)小于 0，则 iXn=iX+1，iXnP2=iX+2，iXnN1=iX-1；
   - 否则，iXn=iX-1，iXnP2=iX-2，iXnN1=iX+1
   - orgPredMatrix[x][y] = (r[iXnN1+1] × (32-offset) + r[iX+1] × (64-offset) + r[iXn+1] × (32+offset) + r[iXnP2+1] × offset+64) >> 7（x=0～M-1,y=0～N-1);

- predMatrix[x][y] = orgPredMatrix[x][y]（x=0～M-1,y=0～N-1）。

4) 否则，如果 iX等于-1（参考左边像素）：
   - 如果(dx×dy)小于 0，iYn=iY+1，iYnP2=iY+2，iYnN1=iY-1；
   - 否则，iYn=iY-1，iYnP2=iY-2，iYnN1=iY+1；
   - orgPredMatrix[x][y] = (c[iYnN1+1] × (32-offset) + c[iY+1] × (64-offset) + c[iYn+1] × (32+offset) + c[iYnP2+1] × offset + 64)>>7（x=0～M-1,y=0～N-1）；
   - predMatrix[x][y] = orgPredMatrix[x][y]（x=0～M-1,y=0～N-1）。

g) IntraLumaPredMode 等于‘Intra_Luma_PCM’
predMatrix[x][y] = 0（x=0～M-1,y=0～N-1）。

表89 角度预测模式的方向

IntraLumaPredMode	dirDx[IntraLumaPredMode]	dirDy[IntraLumaPredMode]	dirXYFlag[IntraLumaPredMode]	dirXYSign[IntraLumaPredMode]	divDxy[IntraLumaPredMode]	divDyx[IntraLumaPredMode]
0	–	–	–	–	–	–
1	–	–	–	–	–	–
2	–	–	–	–	–	–
3	11	-4	0	-1	{11,2}	{93,8}
4	2	-1	0	-1	{2,0}	{1,1}
5	11	-8	0	-1	{11,3}	{93,7}
6	1	-1	0	-1	{1,0}	{1,0}
7	8	-11	0	-1	{93,7}	{11,3}
8	1	-2	0	-1	{1,1}	{2,0}
9	4	-11	0	-1	{93,8}	{11,2}
10	1	-4	0	-1	{1,2}	{4,0}
11	1	-8	0	-1	{1,3}	{8,0}
12	–	–	–	–	–	–
13	1	8	0	1	{1,3}	{8,0}
14	1	4	0	1	{1,2}	{4,0}
15	4	11	0	1	{93,8}	{11,2}
16	1	2	0	1	{1,1}	{2,0}
17	8	11	0	1	{93,7}	{11,3}
18	1	1	0	1	{1,0}	{1,0}
19	11	8	0	1	{11,3}	{93,7}
20	2	1	0	1	{2,0}	{1,1}
21	11	4	0	1	{11,2}	{93,8}
22	4	1	0	1	{4,0}	{1,2}

表 89（续）

IntraLumaPredMode	dirDx[IntraLumaPredMode]	dirDy[IntraLumaPredMode]	dirXYFlag[IntraLumaPredMode]	dirXYSign[IntraLumaPredMode]	divDxy[IntraLumaPredMode]	divDyx[IntraLumaPredMode]
23	8	1	0	1	{8, 0}	{1, 3}
24	–	–	–	–	–	–
25	8	-1	1	-1	{8, 0}	{1, 3}
26	4	-1	1	-1	{4, 0}	{1, 2}
27	11	-4	1	-1	{11, 2}	{93, 8}
28	2	-1	1	-1	{2, 0}	{1, 1}
29	11	-8	1	-1	{11, 3}	{93, 7}
30	1	-1	1	-1	{1, 0}	{1, 0}
31	8	-11	1	-1	{93, 7}	{11, 3}
32	1	-2	1	-1	{1, 1}	{2, 0}

9.7.4.3 帧内预测滤波

帧内预测滤波过程如下：

```
predMatrixIPF[-1][-1] = 0
for (i=0; i<M; i++) {
    predMatrixIPF[i][-1] = r[i+1]
}
for (j=0; j<N; j++) {
    predMatrixIPF[-1][j] = c[j+1]
}
for (i=0; i<M; i++) {
    for (j=0; j<N; j++) {
        predMatrixIPF[i][j] = orgPredMatrix [i][j]
    }
}
if ((IntraLumaPredMode == 0) || (IntraLumaPredMode == 1) || (IntraLumaPredMode == 2)) {
    for (i=0; i<M; i++) {
        for (j=0; j<N; j++) {
            predMatrix[i][j] = Clip1((f(i) * predMatrixIPF[-1][j] + f(j) * predMatrixIPF[i][-1] + (64-f(i)-f(j)) *
predMatrixIPF[i][j] + 32 ) >> 6)
        }
    }
}
else if ((IntraLumaPredMode >= 3) && (IntraLumaPredMode <= 18)) {
    for (i=0; i<M; i++) {
        for (j=0; j<N; j++) {
            predMatrix[i][j] = Clip1((f(i) * predMatrixIPF[-1][ j] + (64- f(i)) * predMatrixIPF[i][j] + 32) >> 6)
        }
```

```
    }
}
else if ((IntraLumaPredMode >= 19) && (IntraLumaPredMode <= 32)) {
    for (i=0; i<M; i++) {
        for (j=0; j<N; j++) {
            predMatrix[i][j] = Clip1((f(j) * predMatrixIPF[i][-1] + (64 - f(j)) * predMatrixIPF[i][j] + 32) >> 6)
        }
    }
}
```

其中，predMatrixIPF是(M+1)×(N+1)大小的样本矩阵，M×N块内坐标为(i,j)的像素对应的滤波系数f[i]和f[j]分别由M、i和N、j查表90得到。

表90　帧内预测边界滤波系数表

i或j	M或N				
	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

9.7.5　色度预测块帧内预测

根据IntraChromaPredMode决定色度块帧内预测方法。

a)　IntraChromaPredMode 等于 0（Intra_Chroma_DM 和 Intra_Chroma_PCM 预测）

1)　如果当前编码单元中 PredBlockOrder 的值为 0 的预测块的 IntraLumaPredMode 等于 12，则 IntraChromaPredMode 的值为 3，转到步骤 d)。

2)　如果当前编码单元中 PredBlockOrder 的值为 0 的预测块的 IntraLumaPredMode 等于 24，则 IntraChromaPredMode 的值为 2，转到步骤 c)。

3)　如果当前编码单元中 PredBlockOrder 的值为 0 的预测块的 IntraLumaPredMode 等于 0，则 IntraChromaPredMode 的值为 1，转到步骤 b)。

4)　如果当前编码单元中 PredBlockOrder 的值为 0 的预测块的 IntraLumaPredMode 等于 2，则 IntraChromaPredMode 的值为 4，转到步骤 e)。

5) 如果当前编码单元中 PredBlockOrder 的值为 0 的预测块的 IntraLumaPredMode 等于 1，则 predMatrix[x][y] = Clip1((ia + (x-((M>>1)-1))×ib + (y-((N>>1)-1))×ic + 16) >> 5) （x=0～M-1,y=0～N-1）。

式中，ia = (row[M]+col[N])<<4，

ib = ((ih<<5)×imh +(1<<(is-1)))>>ish，

ic = ((iv<<5)×imv +(1<<(is-1)))>>isv，

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]，

$$ih = \sum_{i=0}^{(M>>1)-1} (i+1)\times(\text{row}[(M>>1)+1+i]-\text{row}[(M>>1)-1-i])$$，

$$iv = \sum_{i=0}^{(N>>1)-1} (i+1)\times(col[(N>>1)+1+i]-col[(N>>1)-1-i])$$。

6) 如果当前编码单元中 PredBlockOrder 的值为 0 的预测块的 IntraLumaPredMode 等于 33，则 predMatrix[x][y] = 0（x=0～M-1,y=0～N-1）。

7) 如果当前编码单元中 PredBlockOrder 的值为 0 的预测块的 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]。根据 IntraLumaPredMode 的值查表 89 得到 dx、dy、xyAxis、xySign、imx、isx、imy 和 isy。其中：

dx = dirDx[IntraLumaPredMode]

dy = dirDy[IntraLumaPredMode]

xyAxis = dirXYFlag[IntraLumaPredMode]

xySign = dirXYsign[IntraLumaPredMode]

imx = divDxy[IntraLumaPredMode][0]

isx = divDxy[IntraLumaPredMode][1]

imy = divDyx[IntraLumaPredMode][0]

isy = divDyx[IntraLumaPredMode][1]

然后依次执行以下步骤：

——如果 xySign 小于 0，则：

- 如果 xyAxis 等于 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；
- 否则，xyAxis 等于 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)。

——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)：

- 如果 iYy 小于或等于-1（参考上边像素），offset=offsetx，iX=iXx，iY=-1；
- 否则（参考左边像素），offset=offsety，iX=-1，iY=iYy。

——如果 iY 等于-1（参考上边像素），依次执行以下操作：

- 如果(dx×dy)小于 0，则 iXn=iX+1，iXnP2=iX+2，iXnN1=iX-1。
- 否则，iXn=iX-1，iXnP2=iX-2，iXnN1=iX+1。
- predMatrix[x][y]=(row[iXnN1+1]×(32-offset)+row[iX+1]×(64-offset)+row[iXn+1]×(32+offset)+row[iXnP2+1]×offset+64)>>7 （x=0～M-1，y=0～N-1）。

——否则，如果 iX 等于-1（参考左边像素），依次执行以下操作：

- 如果(dx×dy)小于 0，则 iYn=iY+1，iYnP2=iY+2，iYnN1=iY-1。
- 否则，iYn=iY-1，iYnP2=iY-2，iYnN1=iY+1。
- predMatrix[x][y]=(col[iYnN1+1]×(32-offset)+col[iY+1]×(64-offset)+col[iYn+1]×(32+offset)+col[iYnP2+1]×offset+64)>>7 （x=0～M-1，y=0～N-1）。

b) IntraChromaPredMode 等于 1（Intra_Chroma_DC 预测）

1) 如果 row[i]、col[j]（i=1～M，j=1～N）均“可用”，则

predMatrix[x][y]=

$$((\sum_{i=1}^{M}\text{row}[i]+\sum_{i=1}^{N}col[j]+((M+N)>>1))\times(4096/(M+N)))>>12$$（x=0～M-1，y=0～N-1）；

2) 否则，如果 row[i]（i=1～M）“可用”，且 col[j]（j=1～N）“不可用”，则

predMatrix[x][y] =$(\sum_{i=1}^{M}row[i]+(M>>1))>>\text{Log}(M)$（x=0～M-1，y=0～N-1）；

3) 否则，如果 col[j]（i=1～N）“可用”，且 row[i]（i=1～M）“不可用”，则

predMatrix[x][y] =$(\sum_{i=1}^{N}col[i]+(N>>1))>>\text{Log}(N)$（x=0～M-1，y=0～N-1）；

4) 否则，predMatrix[x][y] = $2^{\text{BitDepth}-1}$（x=0～M-1,y=0～N-1；BitDepth 是编码样本精度）。

c) IntraChromaPredMode 等于 2（Intra_Chroma_Horizontal 预测）

predMatrix[x][y] = col[y+1]（x=0～M-1,y=0～N-1）。

d) IntraChromaPredMode 等于 3（Intra_Chroma_Vertical 预测）

predMatrix[x][y] = row[x+1]（x=0～M-1,y=0～N-1）。

e) IntraChromaPredMode 等于 4（Intra_Chroma_Bilinear 预测）

predMatrix[x][y]=Clip1(((((ia-col[y+1])×(x+1))<<Log(N))+(((ib-row[x+1])×(y+1))<<Log(M))+((row[x+1]+col[y+1])<<(Log(M)+Log(N)))+((ic<<1)-ia-ib)×x×y+(1<<(Log(M)+Log(N))))>>(Log(M)+Log(N)+1))（x=0～M-1,y=0～N-1）

式中，ia=row[M]，ib=col[N]。如果 M 等于 N，则 ic=(ia+ib+1)>>1；否则 ic=(((ia<<Log(M))+(ib<<Log(N)))×weight+(1<<(ishift+5)))>>(ishift+6)。ishift=Log(Min(M,N))，weight = bilinearWeight[Log(Max(M,N)/Min(M,N))-1]，bilinearWeight[3] = {21, 13, 7}。

f) IntraChromaPredMode 等于 5（Intra_Chroma_TSCPM 预测）

当前预测块的尺寸为M×N，I表示当前块所在的图像的补偿后样本的亮度样值矩阵,r[i]、c[j]（i=0～2M，j=0～2N）为亮度参考样本，row[i]、col[i]（i=0～2M，j=0～2N）为色度参考样本。

如果row[i]（i=1～M）,col[i]（i=1～N）均“不可用”,则α等于0，β等于1<<(BitDepth-1)，iShift的值等于0；否则，iShift等于16，并按以下步骤导出α和β：

1) 第一步，获取(x[i],y[i])，其中i=0~3：

(1) 如果row[i]（i=1～M）、col[j]（j=1～N）均“可用”：

令posA0等于0，posL0等于0；如果M大于或等于N，posA1等于M-[M/N]，posL1等于N-1；否则，posA1等于M-1，posL1等于N-[N/M]

(x[0],y[0])=((r[2×posA0]+2×r[2×posA0+1]+r[2×posA0+2]+2)>>2,row[posA0+1])

(x[1],y[1])=((r[2×posA1]+2×r[2×posA1+1]+r[2×posA1+2]+2)>>2,row[posA1+1])

(x[2],y[2])=((c[2×posL0+1]+c[2×posL0+2]+1)>>1),col[posL0+1])

(x[3],y[3])=((c[2×posL1+1]+c[2×posL1+2]+1)>>1),col[posL1+1])

(2) 如果row[i]“可用”且col[j]“不可用”（i=1～M，j=1～N）：

令posA0等于0，posA1等于[M/4]，posA2等于[2×M/4]，posA3等于[3×M/4]

(x[0],y[0])=((　　3×r[2×posA0+1]+r[2×posA0+2]+2)>>2,row[posA0+1])

(x[1],y[1])=((r[2×posA1]+2×r[2×posA1+1]+r[2×posA1+2]+2)>>2,row[posA1+1])

(x[2],y[2])=((r[2×posA2]+2×r[2×posA2+1]+r[2×posA2+2]+2)>>2,row[posA2+1])

(x[3],y[3])=((r[2×posA3]+2×r[2×posA3+1]+r[2×posA3+2]+2)>>2,row[posA3+1])

(3) 如果row[i]“不可用”且col[j]“可用”（i=1～M，j=1～N）：

令posL0等于0，posL1等于[N/4]，posL2等于[2×N/4]，posL3等于[3×N/4]

(x[0],y[0])=((c[2×posL0+1]+c[2×posL0+2]+1)>>1),col[posL0+1])

(x[1],y[1])=((c[2×posL1+1]+c[2×posL1+2]+1)>>1),col[posL1+1])

(x[2],y[2])=((c[2×posL2+1]+c[2×posL2+2]+1)>>1),col[posL2+1])

(x[3],y[3])=((c[2×posL3+1]+c[2×posL3+2]+1)>>1),col[posL3+1])

2) 第二步，计算线性模型系数α，β：

(1) 令minIdx[2]={0,2}，maxIdx[2]={1,3}：

如果x[minIdx[0]]大于x[minIdx[1]]，则交换minIdx[0]与minIdx[1]的值；

如果x[maxIdx[0]]大于x[maxIdx[1]]，则交换maxIdx[0]与maxIdx[1]的值；

如果x[minIdx[0]]大于x[maxIdx[1]]，则交换minIdx[0]与maxIdx[0]的值,同时交换minIdx[1]与maxIdx[1]的值；

如果x[minIdx[1]]大于x[maxIdx[0]]，则交换minIdx[1]与maxIdx[0]的值；

(xMin,yMin)=((x[minIdx[0]]+x[minIdx[1]]+1)>>1,(y[minIdx[0]]+y[minIdx[1]]+1)>>1)

(xMax,yMax)=((x[maxIdx[0]]+x[maxIdx[1]]+1)>>1,(y[maxIdx[0]]+y[maxIdx[1]]+1)>>1)

diffX=xMax-xMin

diffY=yMax-yMin

如果diffX大于64：

α = (diffY × TscpmTable[((diffX + add) >> shift) - 1] + add) >> shift

β = yMin-((α × xMin) >> iShift)

其中，shift = (BitDepth > 8) ? BitDepth - 6 : 2，add = 1 << (shift - 1)

否则，如果diffX大于0：

α = (diffY × TscpmTable[diffX − 1])

β = yMin−((α × xMin) >> iShift)

否则：

α = 0

β = yMin

TscpmTable 的值见表 91。

3) 第三步，得到色度预测值：

predChroma[x][y]= Clip1(((α ×I[x][y]) >> iShift) + β) （x=0～2M-1,y=0～2N-1）

predMatrix[0][y]=(predChroma[0][2y] + predChroma[0][2y+1] + 1)>>1, （y=0～N-1）

predMatrix[x][y]=(predChroma[2x-1][2y]+2×predChroma[2x][2y]+predChroma[2x+1][2y]+predChroma[2x-1][2y+1]+2×predChroma[2x][2y+1]+predChroma[2x+1][2y+1]+4)>>3, （x=1～M-1,y=0～N-1）

表91 TscpmTable 查询表

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

9.8 帧间预测

9.8.1 预测样本导出

9.8.1.1 概述

本条定义用于帧间预测的预测样本矩阵的导出过程。

如果当前预测单元的AffineFlag的值为0，用于帧间预测的预测样本矩阵的导出过程见9.8.1.2；否则，用于帧间预测的预测样本矩阵的导出过程见9.8.1.3。

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。

9.8.1.2 普通预测样本导出

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)，执行以下操作：

——如果当前预测块的预测参考模式是‘PRED_List0’，则亮度预测样本矩阵 predMatrixL0 中的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考索引为 RefIdxL0 的参考图像的 1/4 精度亮度样本矩阵中位置为(((xE+x)<<2)+MvE_x,(yE+y)<<2)+MvE_y))的样本值。其中 MvE_x 和 MvE_y 分别是当前预测单元 L0 运动矢量 MvE 的水平分量和垂直分量。

——如果当前预测块的预测参考模式是‘PRED_List1’，则亮度后向预测样本矩阵 predMatrixL1 中的元素 predMatrixL1[x][y]的值为参考图像队列 1 中参考索引为 RefIdxL1 的 1/4 精度亮度样本矩阵中位置为(((xE+x)<<2)+MvE_x,(yE+y)<<2)+MvE_y))的样本值。其中 MvE_x 和 MvE_y 分别是当前预测单元 L1 运动矢量 MvE 的水平分量和垂直分量。

——如果当前预测块的预测参考模式是 ‘PRED_List01’，则亮度预测样本矩阵 predMatrixL0 中的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考帧索引为 RefIdxL0 的 1/4 精度亮度样本矩阵中位置为(((xE+x)<<2)+MvE0_x,(yE+y)<<2)+MvE0_y))的样本值，亮度预测样本矩阵 predMatrixL1 中的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIdxL1 的 1/4 精度亮度样本矩阵中位置为(((xE+x)<<2)+MvE1_x,(yE+y)<<2)+MvE1_y))的样本值。其中 MvE0_x 和 MvE0_y 分别是当前预测单元 L0 运动矢量 MvE0 的水平分量和垂直分量，MvE1_x 和 MvE1_y 分别是当前预测单元 L1 运动矢量 MvE1 的水平分量和垂直分量。

如果当前预测块是色度预测块，记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为(xE,yE)，执行以下操作：

——如果当前预测块的预测参考模式是‘PRED_List0’，则色度预测样本矩阵 predMatrixL0 中的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考索引为 RefIdxL0 的 1/8 精度色度样本矩阵中位置为(((xE+2×x)<<2)+MvC_x,(yE+2×y)<<2)+MvC_y))的样本值。其中，MvC_x 和 MvC_y 分别是包含当前预测块右下角样本的空域运动信息存储单元的 L0 运动矢量 MvE 的水平分量和垂直分量。

——如果当前预测块的预测参考模式是‘PRED_List1’，则色度预测样本矩阵 predMatrixL1 中的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIdxL1 的 1/8 精度色度样本矩阵中位置为(((xE+2×x)<<2)+MvC_x,(yE+2×y)<<2)+MvC_y))的样本值。其中，MvC_x 和 MvC_y 分别是包含当前预测块右下角样本的空域运动信息存储单元的 L1 运动矢量 MvE 的水平分量和垂直分量。

——如果当前预测块的预测参考模式是‘PRED_List01’，则色度预测样本矩阵 predMatrixL0 中的元素 predMatrixL0[x][y]的值是参考图像队列 0 中参考索引为 RefIdxL0 的 1/8 精度色度样本矩阵中位置为(((xE+2×x)<<2)+MvC0_x,(yE+2×y)<<2)+MvC0_y))的样本值，色度预测样本矩阵 predMatrixL1 中的元素 predMatrixL1[x][y]的值是参考图像队列 1 中参考索引为 RefIdxL1 的 1/8 精度色度样本矩阵中位置为(((xE+2×x)<<2)+MvC1_x,(yE+2×y)<<2)+MvC1_y))的样本值。其中，MvC0_x 和 MvC0_y 分别是包含当前预测块右下角样本的空域运动信息存储单元的 L0 运动矢量 MvE0 的水平分量和垂直分量，MvC1_x 和 MvC1_y 分别是包含当前预测块右下角样本的空域运动信息存储单元的 L1 运动矢量 Mv1E 的水平分量和垂直分量。

其中参考图像的亮度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元素值通过9.8.2和9.8.3定义的插值方法得到。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。

9.8.1.3 仿射预测样本导出

记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为(xE,yE)。

mv0E0是MvArrayL0运动矢量集合在(xE+x, yE+y)位置的4×4单元的L0运动矢量。如果当前预测单元的预测参考模式是‘PRED_List0’且AffineSubblockSizeFlag的值为0，则亮度预测样本矩阵predMatrixL0中的元素predMatrixL0[x][y]的值是参考图像队列0中参考索引为RefIdxL0的1/16精度亮度样本矩阵中位置为(((xE+x)<<4)+mv0E0_x,(yE+y)<<4)+mv0E0_y))的样本值，色度预测样本矩阵predMatrixL0中的元素predMatrixL0[x][y]的值是参考图像队列0中参考索引为RefIdxL0的1/32精度色度样本矩阵中位置为(((xE+2×x)<<4)+MvC_x,(yE+2×y)<<4)+MvC_y))的样本值。其中，x1=((xE+2×x)>>3)<<3，y1=((yE+2×y)>>3)<<3，mv1E0是MvArrayL0运动矢量集合在(x1,y1)位置的4×4单元的L0运动矢量，mv2E0是MvArrayL0运动矢量集合在(x1+4,y1)位置的4×4单元的L0运动矢量，mv3E0是MvArrayL0运动矢量集合在(x1,y1+4)位置的4×4单元的L0运动矢量，mv4E0是MvArrayL0运动矢量集合在(x1+4,y1+4)位置的4×4单元的L0运动矢量。

$$MvC\_x = (mv1E0\_x + mv2E0\_x + mv3E0\_x + mv4E0\_x + 2) >> 2$$
$$MvC\_y = (mv1E0\_y + mv2E0\_y + mv3E0\_y + mv4E0\_y + 2) >> 2$$

mv0E0是MvArrayL0运动矢量集合在(xE+x,yE+y)位置的8×8单元的L0运动矢量。如果当前预测单元的预测参考模式是‘PRED_List0’且AffineSubblockSizeFlag的值为1，则亮度预测样本矩阵predMatrixL0中的元素predMatrixL0[x][y]的值是参考图像队列0中参考索引为RefIdxL0的1/16精度亮度样本矩阵中位置为(((xE+x)<<4)+ mv0E0_x,(yE+y)<<4)+ mv0E0_y))的样本值，色度预测样本矩阵predMatrixL0中的元素predMatrixL0[x][y]的值是参考图像队列0中参考索引为RefIdxL0的1/32精度色度样本矩阵中位置为(((xE+2×x)<<4)+ MvC_x,(yE+2×y)<<4)+ MvC_y))的样本值。其中，MvC_x等于mv0E0_x，MvC_y等于mv0E0。

mv0E1是MvArrayL1运动矢量集合在(xE+x,yE+y)位置的4×4单元的L1运动矢量。如果当前预测单元的预测参考模式是‘PRED_List1’且AffineSubblockSizeFlag的值为0，则亮度预测样本矩阵predMatrixL1中的元素predMatrixL1[x][y]的值是参考图像队列1中参考索引为RefIdxL1的1/16精度亮度样本矩阵中位置为(((xE+x)<<4)+ mv0E1_x,(yE+y)<<4)+ mv0E1_y))的样本值，色度预测样本矩阵predMatrixL1中的元素predMatrixL1[x][y]的值是参考图像队列1中参考索引为RefIdxL1的1/32精度色度样本矩阵中位置为(((xE+2×x)<<4)+ MvC_x,(yE+2×y)<<4)+ MvC_y))的样本值。其中，x1=((xE+2×x)>>3)<<3，y1=((yE+2×y)>>3)<<3，mv1E1是MvArrayL1运动矢量集合在(x1,y1)位置的4×4单元的L1运动矢量，mv2E1是MvArrayL1运动矢量集合在(x1+4,y1)位置的4×4单元的L1运动矢量，mv3E1是MvArrayL1运动矢量集合在(x1,y1+4)位置的4×4单元的L1运动矢量，mv4E1是MvArrayL1运动矢量集合在(x1+4,y1+4)位置的4×4单元的L1运动矢量。

$$MvC\_x = (mv1E1\_x + mv2E1\_x + mv3E1\_x + mv4E1\_x + 2) >> 2$$
$$MvC\_y = (mv1E1\_y + mv2E1\_y + mv3E1\_y + mv4E1\_y + 2) >> 2$$

mv0E1是MvArrayL1运动矢量集合在(xE+x,yE+y)位置的8×8单元的L1运动矢量。如果当前预测单元的预测参考模式是‘PRED_List1’且AffineSubblockSizeFlag的值为1，则亮度预测样本矩阵predMatrixL1中的元素predMatrixL1[x][y]的值是参考图像队列1中参考索引为RefIdxL1的1/16精度亮度样本矩阵中位置为(((xE+x)<<4)+ mv0E1_x,(yE+y)<<4)+ mv0E1_y))的样本值，色度预测样本矩阵predMatrixL1中的元素predMatrixL1[x][y]的值是参考图像队列1中参考索引为RefIdxL1的1/32精度色度样本矩阵中位置为(((xE+2×x)<<4)+ MvC_x,(yE+2×y)<<4)+ MvC_y))的样本值。其中MvC_x等于mv0E1_x，MvC_y等于mv0E1。

mv0E0是MvArrayL0运动矢量集合在(xE+x,yE+y)位置的8×8单元的L0运动矢量，mv0E1是MvArrayL1运动矢量集合在(x,y)位置的8×8单元的L1运动矢量。如果当前预测单元的预测模式是‘PRED_List01’，则亮度预测样本矩阵predMatrixL0中的元素predMatrixL0[x][y]的值是参考图像队列0中参考索引为RefIdxL0的1/16精度亮度样本矩阵中位置为(((xE+x)<<4)+ mv0E0_x,(yE+y)<<4)+ mv0E0_y))的样本

值，色度预测样本矩阵predMatrixL0中的元素predMatrixL0[x][y]的值是参考图像队列0中参考索引为RefIdxL0的1/32精度色度样本矩阵中位置为(((xE+2×x)<<4)+ MvC0_x,(yE+2×y)<<4)+ MvC0_y)))的样本值，亮度预测样本矩阵predMatrixL1中的元素predMatrixL1[x][y]的值是参考图像队列1中参考索引为RefIdxL1的1/16精度亮度样本矩阵中位置为((((xE+x)<<4)+mvOE1_x,(yE+y)<<4)+ mvOE1_y))))的样本值，色度预测样本矩阵predMatrixL1中的元素predMatrixL1[x][y]的值是参考图像队列1中参考索引为RefIdxL1的1/32精度色度样本矩阵中位置为(((xE+2×x)<<4)+ MvC1_x,(yE+2×y)<<4)+ MvC1_y)))的样本值。其中MvC0_x等于mvOE0_x，MvC0_y等于mvOE0_y，MvC1_x等于mvOE1_x，MvC1_y等于mvOE1_y。

其中参考图像的亮度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元素值通过9.8.2和9.8.3定义的插值方法得到。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。

9.8.2 亮度样本插值过程

9.8.2.1 概述

如果当前预测单元的AffineFlag的值为0，亮度样本的插值过程见9.8.2.2；否则，亮度样本的插值过程见9.8.2.3。

9.8.2.2 普通亮度样本插值过程

图26给出了参考图像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92，其中xFracL等于f_x & 3，yFracL等于f_y & 3。

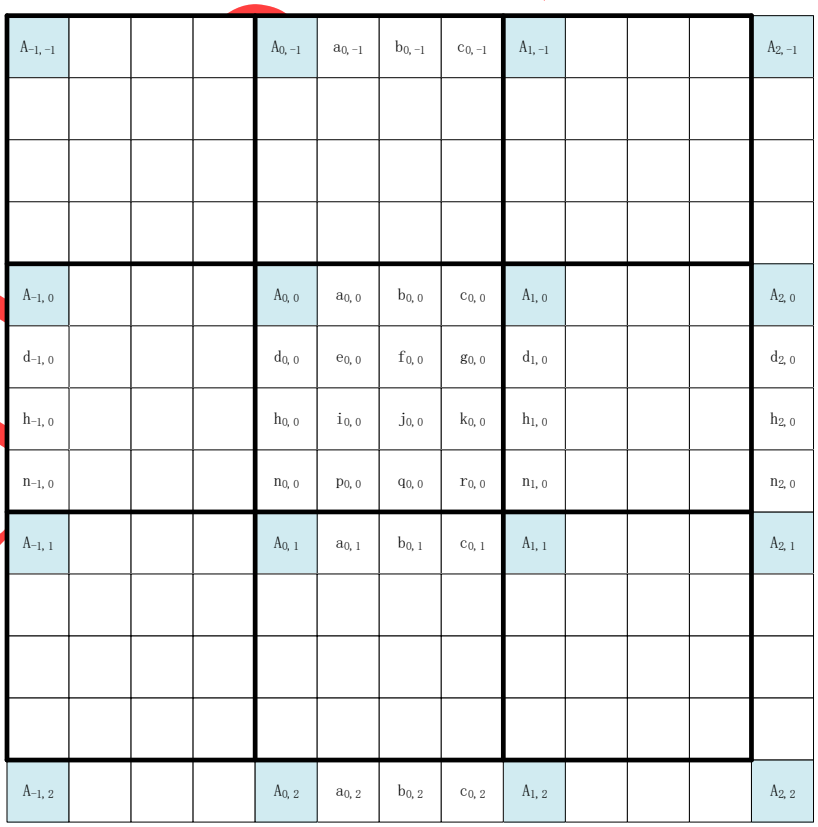


图26 整数样本、1/2 样本和 1/4 样本的位置

表92　预测样本矩阵元素

xFracL的值	yFracL的值	图26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	$g_{0,0}$
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：

$$e_{0,0} = Clip1((-a'_{0,-3} + 4*a'_{0,-2} - 10*a'_{0,-1} + 57*a'_{0,0} + 19*a'_{0,1} - 7*a'_{0,2} + 3*a'_{0,3} - a'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$i_{0,0} = Clip1((-a'_{0,-3} + 4*a'_{0,-2} - 11*a'_{0,-1} + 40*a'_{0,0} + 40*a'_{0,1} - 11*a'_{0,2} + 4*a'_{0,3} - a'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$p_{0,0} = Clip1((-a'_{0,-3} + 3*a'_{0,-2} - 7*a'_{0,-1} + 19*a'_{0,0} + 57*a'_{0,1} - 10*a'_{0,2} + 4*a'_{0,3} - a'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$f_{0,0} = Clip1((-b'_{0,-3} + 4*b'_{0,-2} - 10*b'_{0,-1} + 57*b'_{0,0} + 19*b'_{0,1} - 7*b'_{0,2} + 3*b'_{0,3} - b'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$j_{0,0} = Clip1((-b'_{0,-3} + 4*b'_{0,-2} - 11*b'_{0,-1} + 40*b'_{0,0} + 40*b'_{0,1} - 11*b'_{0,2} + 4*b'_{0,3} - b'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$q_{0,0} = Clip1((-b'_{0,-3} + 3*b'_{0,-2} - 7*b'_{0,-1} + 19*b'_{0,0} + 57*b'_{0,1} - 10*b'_{0,2} + 4*b'_{0,3} - b'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$$g_{0,0} = Clip1((-c'_{0,-3} + 4*c'_{0,-2} - 10*c'_{0,-1} + 57*c'_{0,0} + 19*c'_{0,1} - 7*c'_{0,2} + 3*c'_{0,3} - c'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$$

$k_{0,0} = Clip1((-c'_{0,-3} + 4*c'_{0,-2} - 11*c'_{0,-1} + 40*c'_{0,0} + 40*c'_{0,1} - 11*c'_{0,2} + 4*c'_{0,3} - c'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$

$r_{0,0} = Clip1((-c'_{0,-3} + 3*c'_{0,-2} - 7*c'_{0,-1} + 19*c'_{0,0} + 57*c'_{0,1} - 10*c'_{0,2} + 4*c'_{0,3} - c'_{0,4} + (1 << 19 - BitDepth)) >> (20 - BitDepth))$

其中：

$a'_{0,i} = (-A_{-3,I} + 4*A_{-2,i} - 10*A_{-1,i} + 57*A_{0,i} + 19*A_{1,i} - 7*A_{2,i} + 3*A_{3,i} - A_{4,i} + ((1 << BitDepth - 8) >> 1)) >> (BitDepth - 8)$

$b'_{0,i} = (-A_{-3,i} + 4*A_{-2,i} - 11*A_{-1,i} + 40*A_{0,i} + 40*A_{1,i} - 11*A_{2,i} + 4*A_{3,i} - A_{4,i} + ((1 << BitDepth - 8) >> 1)) >> (BitDepth - 8)$

$c'_{0,i} = (-A_{-3,i} + 3*A_{-2,i} - 7*A_{-1,i} + 19*A_{0,i} + 57*A_{1,i} - 10*A_{2,i} + 4*A_{3,i} - A_{4,i} + ((1 << BitDepth - 8) >> 1)) >> (BitDepth - 8)$

9.8.2.3 仿射亮度样本插值过程

参考图像亮度样本矩阵中的样本位置见图27，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28。

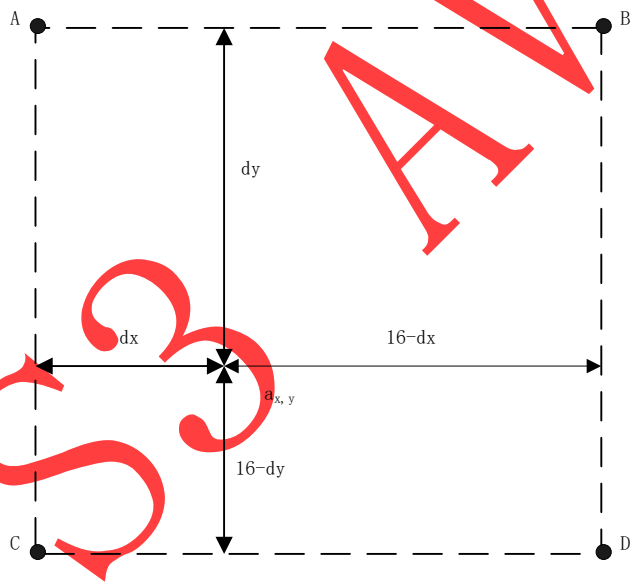


图27 亮度样本的位置

$A_{x,y}$	$a_{x,y}(1,0)$	$a_{x,y}(2,0)$	$a_{x,y}(3,0)$	$a_{x,y}(4,0)$	$a_{x,y}(5,0)$	…	$a_{x,y}(15,0)$
$a_{x,y}(0,1)$	$a_{x,y}(1,1)$	$a_{x,y}(2,1)$	$a_{x,y}(3,1)$	$a_{x,y}(4,1)$	$a_{x,y}(5,1)$	…	$a_{x,y}(15,1)$
$a_{x,y}(0,2)$	$a_{x,y}(1,2)$	$a_{x,y}(2,2)$	$a_{x,y}(3,2)$	$a_{x,y}(4,2)$	$a_{x,y}(5,2)$	…	$a_{x,y}(15,2)$
$a_{x,y}(0,3)$	$a_{x,y}(1,3)$	$a_{x,y}(2,3)$	$a_{x,y}(3,3)$	$a_{x,y}(4,3)$	$a_{x,y}(5,3)$	…	$a_{x,y}(15,3)$
$a_{x,y}(0,4)$	$a_{x,y}(1,4)$	$a_{x,y}(2,4)$	$a_{x,y}(3,4)$	$a_{x,y}(4,4)$	$a_{x,y}(5,4)$	…	$a_{x,y}(15,4)$
$a_{x,y}(0,5)$	$a_{x,y}(1,5)$	$a_{x,y}(2,5)$	$a_{x,y}(3,5)$	$a_{x,y}(4,5)$	$a_{x,y}(5,5)$	…	$a_{x,y}(15,5)$
…	…	…	…	…	…	…	…
$a_{x,y}(0,15)$	$a_{x,y}(1,15)$	$a_{x,y}(2,15)$	$a_{x,y}(3,15)$	$a_{x,y}(4,15)$	$a_{x,y}(5,15)$	…	$a_{x,y}(15,15)$

图28　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得8个整数值滤波得到，预测值得获取方式如下：

$$a_{x,0} = Clip1((f_L[x][0]*A_{-3,0} + f_L[x][1]*A_{-2,0} + f_L[x][2]*A_{-1,0} + f_L[x][3]*A_{0,0} + f_L[x][4]*A_{1,0} + f_L[x][5]*A_{2,0} + f_L[x][6]*A_{3,0} + f_L[x][7]*A_{4,0} + 32) >> 6)$$

样本位置$a_{0,y}$（y=1~15）由垂直方向上距离插值点最近得8个整数值滤波得到，预测值得获取方式如下：

$$a_{0,y} = Clip1((f_L[y][0]*A_{0,-3} + f_L[y][1]*A_{-2,0} + f_L[y][2]*A_{-1,0} + f_L[y][3]*A_{0,0} + f_L[y][4]*A_{1,0} + f_L[y][5]*A_{2,0} + f_L[y][6]*A_{3,0} + f_L[y][7]*A_{-4,0} + 32) >> 6)$$

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：

$$a'_{x,y} = (f_L[x][0]*A_{-3,y} + f_L[x][1]*A_{-2,y} + f_L[x][2]*A_{-1,y} + f_L[x][3]*A_{0,y} + f_L[x][4]*A_{1,y} + f_L[x][5]*A_{2,y} + f_L[x][6]*A_{3,y} + f_L[x][7]*A_{4,y} + ((1 << (BitDepth - 8)) >> 1)) >> (BitDepth - 8)$$

亮度插值滤波器系数见表93。

表93　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

9.8.3　色度样本插值过程

9.8.3.1　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9.8.3.2；否则，色度样本的插值过程见9.8.3.3。

9.8.3.2　普通色度样本插值过程

参考图像色度样本矩阵中的样本位置见图29，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30。

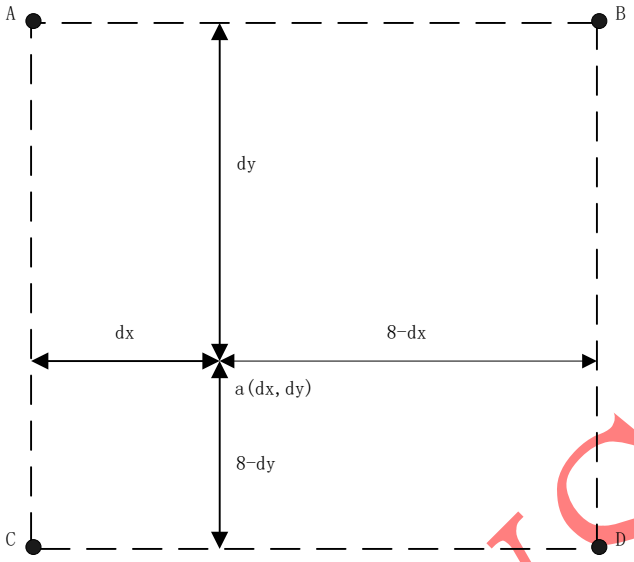


图29　色度样本位置

$A_{x,y}$	$a_{x,y}(1,0)$	$a_{x,y}(2,0)$	$a_{x,y}(3,0)$	$a_{x,y}(4,0)$	$a_{x,y}(5,0)$	$a_{x,y}(6,0)$	$a_{x,y}(7,0)$
$a_{x,y}(0,1)$	$a_{x,y}(1,1)$	$a_{x,y}(2,1)$	$a_{x,y}(3,1)$	$a_{x,y}(4,1)$	$a_{x,y}(5,1)$	$a_{x,y}(6,1)$	$a_{x,y}(7,1)$
$a_{x,y}(0,2)$	$a_{x,y}(1,2)$	$a_{x,y}(2,2)$	$a_{x,y}(3,2)$	$a_{x,y}(4,2)$	$a_{x,y}(5,2)$	$a_{x,y}(6,2)$	$a_{x,y}(7,2)$
$a_{x,y}(0,3)$	$a_{x,y}(1,3)$	$a_{x,y}(2,3)$	$a_{x,y}(3,3)$	$a_{x,y}(4,3)$	$a_{x,y}(5,3)$	$a_{x,y}(6,3)$	$a_{x,y}(7,3)$
$a_{x,y}(0,4)$	$a_{x,y}(1,4)$	$a_{x,y}(2,4)$	$a_{x,y}(3,4)$	$a_{x,y}(4,4)$	$a_{x,y}(5,4)$	$a_{x,y}(6,4)$	$a_{x,y}(7,4)$
$a_{x,y}(0,5)$	$a_{x,y}(1,5)$	$a_{x,y}(2,5)$	$a_{x,y}(3,5)$	$a_{x,y}(4,5)$	$a_{x,y}(5,5)$	$a_{x,y}(6,5)$	$a_{x,y}(7,5)$
$a_{x,y}(0,6)$	$a_{x,y}(1,6)$	$a_{x,y}(2,6)$	$a_{x,y}(3,6)$	$a_{x,y}(4,6)$	$a_{x,y}(5,6)$	$a_{x,y}(6,6)$	$a_{x,y}(7,6)$
$a_{x,y}(0,7)$	$a_{x,y}(1,7)$	$a_{x,y}(2,7)$	$a_{x,y}(3,7)$	$a_{x,y}(4,7)$	$a_{x,y}(5,7)$	$a_{x,y}(6,7)$	$a_{x,y}(7,7)$

图30　整像素样本和分像素样本的位置

色度滤波系数见表94。

表94　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：

$a'_{x,y}(dx,0) = (C[dx][0]*A_{x-1,y} + C[dx][1]*A_{x,y} + C[dx][2]*A_{x+1,y} + C[dx][3]*A_{x+2,y} + ((1 << (BitDepth-8)) >> 1)) >> (BitDepth - 8)$

9.8.3.3　仿射色度样本插值过程

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& 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32。

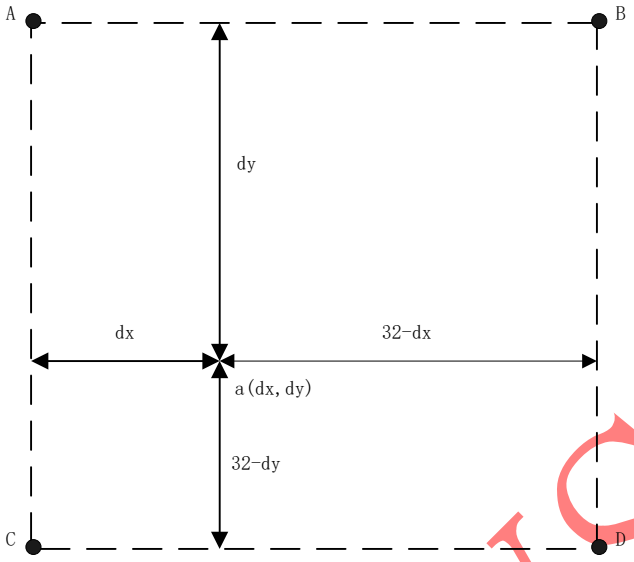


图31　色度样本的位置

$A_{x,y}$	$a_{x,y}(1,0)$	$a_{x,y}(2,0)$	$a_{x,y}(3,0)$	$a_{x,y}(4,0)$	$a_{x,y}(5,0)$	…	$a_{x,y}(31,0)$
$a_{x,y}(0,1)$	$a_{x,y}(1,1)$	$a_{x,y}(2,1)$	$a_{x,y}(3,1)$	$a_{x,y}(4,1)$	$a_{x,y}(5,1)$	。。。	$a_{x,y}(31,1)$
$a_{x,y}(0,2)$	$a_{x,y}(1,2)$	$a_{x,y}(2,2)$	$a_{x,y}(3,2)$	$a_{x,y}(4,2)$	$a_{x,y}(5,2)$	。。。	$a_{x,y}(31,2)$
$a_{x,y}(0,3)$	$a_{x,y}(1,3)$	$a_{x,y}(2,3)$	$a_{x,y}(3,3)$	$a_{x,y}(4,3)$	$a_{x,y}(5,3)$	。。。	$a_{x,y}(31,3)$
$a_{x,y}(0,4)$	$a_{x,y}(1,4)$	$a_{x,y}(2,4)$	$a_{x,y}(3,4)$	$a_{x,y}(4,4)$	$a_{x,y}(5,4)$	。。。	$a_{x,y}(31,4)$
$a_{x,y}(0,5)$	$a_{x,y}(1,5)$	$a_{x,y}(2,5)$	$a_{x,y}(3,5)$	$a_{x,y}(4,5)$	$a_{x,y}(5,5)$	。。。	$a_{x,y}(31,5)$
。。。	。。。	。。。	。。。	。。。	。。。	。。。	。。。
$a_{x,y}(0,31)$	$a_{x,y}(1,31)$	$a_{x,y}(2,31)$	$a_{x,y}(3,31)$	$a_{x,y}(4,31)$	$a_{x,y}(5,31)$	。。。	$a_{x,y}(31,31)$

图32　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：

$a'_{x,y}(dx,0) = (f_C[dx][0]*A_{x-1,y} + f_C[dx][1]*A_{x,y} + f_C[dx][2]*A_{x+1,y} + f_C[dx][3]*A_{x+2,y} + ((1 << (BitDepth-8)) >> 1)) >> (BitDepth-8)$

色度插值滤波器系数见表95。

表95　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

表 95（续）

分像素位置	插值滤波器系数			
	$f_C[p][0]$	$f_C[p][1]$	$f_C[p][2]$	$f_C[p][3]$
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
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

9.9 预测补偿

如果当前预测单元的预测模式是帧内，补偿后样本矩阵CompMatrix的计算见式（35）：

CompMatrix[x][y] = Clip1(predMatrix[x][y] + ResidueMatrix[x][y])……(35)

如果当前预测单元的预测参考模式是‘PRED_List01’，补偿后样本矩阵CompMatrix的计算见式(36)：

CompMatrix[x][y] = Clip1(((predMatrixL0[x][y] + predMatrixL1[x][y]+1) >> 1) + ResidueMatrix[x][y])……(36)

如果当前预测单元的预测参考模式是‘PRED_List0’，补偿后样本矩阵CompMatrix的计算见式(37)：

CompMatrix[x][y] = Clip1(predMatrixL0[x][y] + ResidueMatrix[x][y])……(37)

如果当前预测单元的预测参考模式是‘PRED_List1’，补偿后样本矩阵CompMatrix的计算见式(38)：

CompMatrix[x][y] = Clip1(predMatrixL1[x][y] + ResidueMatrix[x][y])……(38)

其中predMatrix是帧内预测的预测样本矩阵，predMatrixL0是参考图像队列0中参考图像的预测样本矩阵，predMatrixL1是参考图像队列1中参考图像的预测样本矩阵。x的取值范围是0~M-1；y的取值范围是0~N-1，M和N分别是当前预测单元的宽度和高度。

9.10 去块效应滤波

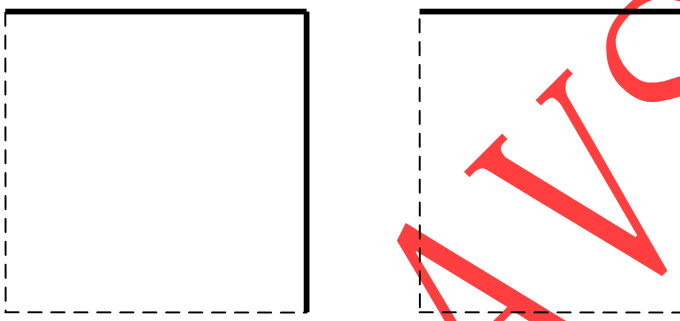
9.10.1 概述

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，如图33中粗线所示。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，首先根据9.10.2判断该边界是否需要滤波。如果需要滤波，则根据9.10.3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9.10.4、9.10.5、9.10.6和9.10.7）；否则，将补偿后样本的值直接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，虚线为下一滤波块待滤波的边界。

图33　滤波单元中需要处理的边界示意

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：

a)　如果待滤波边界是图像边界，则该边界不需要滤波。

b)　如果待滤波边界是片边界且 cross_patch_loopfilter_enable_flag 的值为‘0’，则该边界不需要滤波。

c)　如果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，则该边界不需要滤波。

d)　如果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，则该边界不需要滤波。

9.10.3　边界滤波强度的推导过程

图34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。

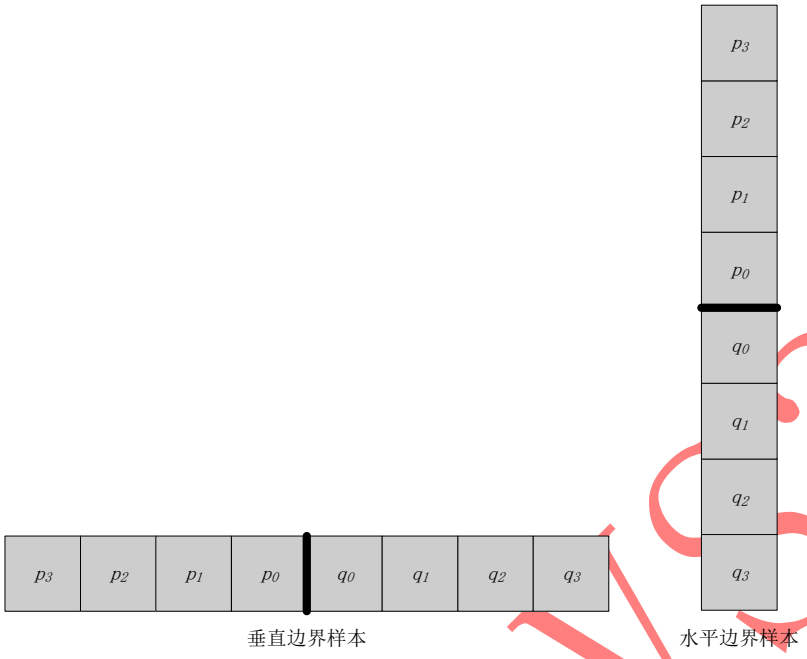


图34　滤波块边界样本

a)　如果满足以下所有条件，则边界滤波强度 Bs 等于 0：

1)　p_0 和 q_0 所在的编码单元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；

2)　p_0 和 q_0 所在的编码单元的预测类型不是帧内。

3)　记 B_P 和 B_Q 分别是 p_0 和 q_0 所在的 4×4 亮度块，B_P 和 B_Q 的运动信息同时满足条件 a 和 b 或满足条件 c 或满足条件 d：

a)　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；

b)　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；

c)　满足以下两个条件：

1.　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；

2.　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；

d)　满足以下两个条件：

1. 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；
2. 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。

b) 否则，计算边界滤波强度的过程如下：

1) 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，应使用亮度块的量化参数；如果是色度样本，应使用色度块的量化参数。设 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，则平均量化参数按下式计算：

$$QP_{av} = (QP_p + QP_q + 1) >> 1$$

2) 计算索引 IndexA 和 IndexB：

$$IndexA = Clip3(0, 63, QP_{av} - 8 \times (BitDepth - 8) + AlphaCOffset)$$
$$IndexB = Clip3(0, 63, QP_{av} - 8 \times (BitDepth - 8) + BetaOffset)$$

3) 根据索引 IndexA 和 IndexB 与阈值α和β间的对应关系，由表 96 得到α’、β’的取值，根据 IndexA 查表得到α’，根据 IndexB 查表得到β’。再根据 bitDepth 按以下方法得到α、β的值。

$$\alpha = \alpha' << (BitDepth - 8)$$
$$\beta = \beta' << (BitDepth - 8)$$

表96 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

4) 按以下方法计算 Bs：

- 第一步，将 fL 和 fR 的值均置为 0，

if ($Abs(p_0 - p_1) < \beta$)

```
        fL += 2
    if ( Abs(p0 - p2) < β )
        fL ++
    if ( Abs(q0 - q1) < β )
        fR += 2
    if ( Abs(q0 - q2) < β )
        fR ++
    fS = fL + fR
```

- 第二步，根据 fS 的值得到 Bs：
  1. 当 fS 等于 6 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−q0)小于α，则 Bs 等于 4；否则 Bs 等于 3。
  2. 当 fS 等于 5 时，如果 p0 等于 p1 且 q0 等于 q1，则 Bs 等于 3；否则 Bs 等于 2。
  3. 当 fS 等于 4 时，如果 fL 等于 2，则 Bs 等于 2；否则 Bs 等于 1。
  4. 当 fS 等于 3 时，如果 Abs(p1−q1)小于β，则 Bs 等于 1；否则 Bs 等于 0。
  5. 当 fS 为其它值时，Bs 等于 0。
- 第三步，如果第二步得到的 Bs 不等于 0 且滤波的边界是色度块边界，则 Bs 减 1。

9.10.4 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)：

$$P_0 = (p_2 * 3 + p_1 * 8 + p_0 * 10 + q_0 * 8 + q_1 * 3 + 16) >> 5$$
$$P_1 = (p_2 * 4 + p_1 * 5 + p_0 * 4 + q_0 * 3 + 8) >> 4$$
$$P_2 = (p_3 * 2 + p_2 * 2 + p_1 * 2 + p_0 * 1 + q_0 * 1 + 4) >> 3$$
$$Q_0 = (p_1 * 3 + p_0 * 8 + q_0 * 10 + q_1 * 8 + q_2 * 3 + 16) >> 5$$
$$Q_1 = (p_0 * 3 + q_0 * 4 + q_1 * 5 + q_2 * 4 + 8) >> 4$$
$$Q_2 = (p_0 * 1 + q_0 * 1 + q_1 * 2 + q_2 * 2 + q_3 * 2 + 4) >> 3$$

9.10.5 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)：

$$P_0 = (p_2 + (p_1 << 2) + (p_0 << 2) + (p_0 << 1) + (q_0 << 2) + q_1 + 8) >> 4$$
$$P_1 = ((p_2 << 1) + p_2 + (p_1 << 3) + (p_0 << 2) + q_0 + 8) >> 4$$
$$Q_0 = (p_1 + (p_0 << 2) + (q_0 << 2) + (q_0 << 1) + (q_1 << 2) + q_2 + 8) >> 4$$
$$Q_1 = ((q_2 << 1) + q_2 + (q_1 << 3) + (q_0 << 2) + p_0 + 8) >> 4$$

9.10.6 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)：

$$P_0 = ((p_1 << 1) + p_1 + (p_0 << 3) + (p_0 << 1) + (q_0 << 1) + q_0 + 8) >> 4$$
$$Q_0 = ((p_0 << 1) + p_0 + (q_0 << 3) + (q_0 << 1) + (q_1 << 1) + q_1 + 8) >> 4$$

9.10.7 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)：

$$P_0 = ((p_0 << 1) + p_0 + q_0 + 2) >> 2$$

$$Q_0 = ((q_0 << 1) + q_0 + p_0 + 2) >> 2$$

9.10.8 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)：

$$P_0 = ((p_1 << 1) + p_1 + (p_0 << 3) + (p_0 << 1) + (q_0 << 1) + q_0 + 8) >> 4$$
$$Q_0 = ((p_0 << 1) + p_0 + (q_0 << 3) + (q_0 << 1) + (q_1 << 1) + q_1 + 8) >> 4$$

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)：

$$P_1 = ((p_2 << 1) + p_2 + (p_1 << 3) + (p_0 << 1) + p_0 + (q_0 << 1) + 8) >> 4$$
$$Q_1 = ((q_2 << 1) + q_2 + (q_1 << 3) + (q_0 << 1) + q_0 + (p_0 << 1) + 8) >> 4$$

9.11 样值偏移补偿

9.11.1 概述

样值偏移补偿的处理步骤如下：

a) 如果 PatchSaoEnableFlag[compIdx]的值为 0，则将滤波后样本对应分量的值直接作为偏移后该样本分量的值；

b) 否则，根据 9.11.2 导出样值偏移补偿单元，根据 9.11.3 导出与当前样值偏移补偿单元对应的样值偏移补偿信息，然后按照 9.11.4 对当前样值偏移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。

9.11.2 导出样值偏移补偿单元

先确定当前最大编码单元，然后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补偿单元。

a) 将当前最大编码单元 C 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 C 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 C 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 C 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 作为当前样值偏移补偿单元。

9.11.3 导出样值偏移补偿信息

本条定义了样值偏移补偿单元中各分量的样值偏移补偿操作所需信息的导出方法。这些信息包括：样值偏移补偿模式SaoMode[compIdx]、样值偏移补偿的偏移值SaoOffset[compIdx][j]、样值偏移补偿边缘模式类型SaoEdgeType[compIdx]、样值偏移补偿区间模式的起始偏移子区间位置SaoIntervalStartPos[compIdx]和样值偏移补偿区间模式的起始偏移子区间的位置差值SaoIntervalDeltaPosMinus2[compIdx]。

如果SaoMergeLeftAvai或SaoMergeUpAvai的值为1，则MergeFlagExist的值为1；否则MergeFlagExist的值为0。

根据SaoMergeLeftAvai、SaoMergeUpAvai和SaoMergeTypeIndex的值查表97得到样值偏移补偿合并模式SaoMergeMode。

表97　样值偏移补偿模式

SaoMergeLeftAvai的值	SaoMergeUpAvai的值	SaoMergeTypeIndex的值	样值偏移补偿合并模式（SaoMergeMode）
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

SaoMode[compIdx]的导出方法如下：如果SaoMergeMode的值为‘SAO_NON_MERGE’，从码流中解析得到SaoMode[compIdx]的值；如果SaoMergeMode的值为‘SAO_MERGE_LEFT’，SaoMode[compIdx]的值等于左边样值偏移补偿单元的SaoMode[compIdx]的值；如果SaoMergeMode值为‘SAO_MERGE_UP’，SaoMode[compIdx]的值等于上边样值偏移补偿单元的SaoMode[compIdx]的值。

a) 如果SaoMode[compIdx]的值为‘SAO_Interval’，先按以下方法导出SaoOffset[compIdx][j]（j=0～3）、SaoIntervalStartPos[compIdx]和SaoIntervalDeltaPosMinus2[compIdx]，然后根据9.11.4.1进行样值偏移补偿操作：

1) 如果SaoMergeMode的值为‘SAO_NON_MERGE’，从码流中解析得到SaoIntervalOffsetAbs[compIdx][j]（j=0～3）、SaoIntervalOffsetSign[compIdx][j]（j=0～3）、SaoIntervalStartPos[compIdx]和SaoIntervalDeltaPosMinus2[compIdx]，计算得到SaoOffset[compIdx][j] = SaoIntervalOffsetAbs[compIdx][j] × SaoIntervalOffsetSign[compIdx][j]（j=0～3）；

2) 否则，如果SaoMergeMode的值为‘SAO_MERGE_LEFT’，SaoOffset[compIdx][j]（j=0～3）的值等于左边样值偏移补偿单元的SaoOffset[compIdx][j]（j=0～3）的值，SaoIntervalStartPos[compIdx]和SaoIntervalDeltaPosMinus2[compIdx]的值等于左边样值偏移补偿单元的SaoIntervalStartPos[compIdx]和SaoIntervalDeltaPosMinus2[compIdx]的值；

3) 否则，SaoOffset[compIdx][j]（j=0～3）的值等于上边样值偏移补偿单元的SaoOffset[compIdx][j]（j=0～3）的值，SaoIntervalStartPos[compIdx]和SaoIntervalDeltaPosMinus2[compIdx]的值等于上边样值偏移补偿单元的SaoIntervalStartPos[compIdx]和SaoIntervalDeltaPosMinus2[compIdx]的值。

b) 否则，如果SaoMode[compIdx]的值为‘SAO_Edge’，先按以下方法导出SaoOffset[compIdx][j]（j=0～3）和SaoEdgeType[compIdx]，然后根据9.11.4.2进行样值偏移补偿操作：

1) 如果 SaoMergeMode 的值为‘SAO_NON_MERGE’，从码流中解析得到 SaoEdgeOffset[compIdx][j]（j=0～3）和 SaoEdgeType[compIdx]的值，计算得到 SaoOffset[compIdx][j]= SaoEdgeOffset[compIdx][j]；
2) 否则，如果 SaoMergeMode 的值为‘SAO_MERGE_LEFT’，SaoOffset[compIdx][j]（j=0～3）和 SaoEdgeType[compIdx]的值等于左边样值偏移补偿单元的 SaoOffset[compIdx][j]（j=0～3）和 SaoEdgeType[compIdx]的值；
3) 否则，SaoOffset[compIdx][j]（j=0～3）和 SaoEdgeType[compIdx]的值等于上边样值偏移补偿单元的 SaoOffset[compIdx][j]（j=0～3）和 SaoEdgeType[compIdx]的值。

9.11.4 样值偏移补偿操作

9.11.4.1 SAO_Interval 模式的操作

如果样值偏移补偿单元的SaoMode[compIdx]为‘SAO_Interval’，进行以下操作：

a) 第一步，根据滤波后样本的 compIdx 分量值查表 98 得到该分量对应的 saoOffset[compIdx]。

表98 区间模式的偏移值

样值	偏移值（saoOffset[compIdx]）
SaoIntervalOffsetPos[compIdx][0] << shift1 ～ (SaoIntervalOffsetPos[compIdx][0] << shift1) + ((1 << shift1) - 1)	SaoOffset[compIdx][0]
SaoIntervalOffsetPos[compIdx][1] << shift1 ～ (SaoIntervalOffsetPos[compIdx][1] << shift1) + ((1 << shift1) - 1)	SaoOffset[compIdx][1]
SaoIntervalOffsetPos[compIdx][2] << shift1 ～ (SaoIntervalOffsetPos[compIdx][2] << shift1) + ((1 << shift1) - 1)	SaoOffset[compIdx][2]
SaoIntervalOffsetPos[compIdx][3] << shift1 ～ (SaoIntervalOffsetPos[compIdx][3] << shift1) + ((1 << shift1) - 1)	SaoOffset[compIdx][3]
其他	0

表98中：

shift1 = BitDepth - 5

SaoIntervalOffsetPos[compIdx][0] = SaoIntervalStartPos[compIdx]

SaoIntervalOffsetPos[compIdx][1] = (SaoIntervalStartPos[compIdx] + 1) % 32

SaoIntervalOffsetPos[compIdx][2]=(SaoIntervalStartPos[compIdx] + SaoIntervalDeltaPosMinus2[compIdx] + 2) % 32

SaoIntervalOffsetPos[compIdx][3]=(SaoIntervalStartPos[compIdx] + SaoIntervalDeltaPosMinus2[compIdx] + 3) % 32

b) 第二步，当前样本的偏移后样本 compIdx 分量值 y[compIdx]=Clip1(x[compIdx]+saoOffset[compIdx])，其中 x[compIdx]是该样本滤波后样本的 compIdx 分量的值。

9.11.4.2 SAO_Edge 模式的操作

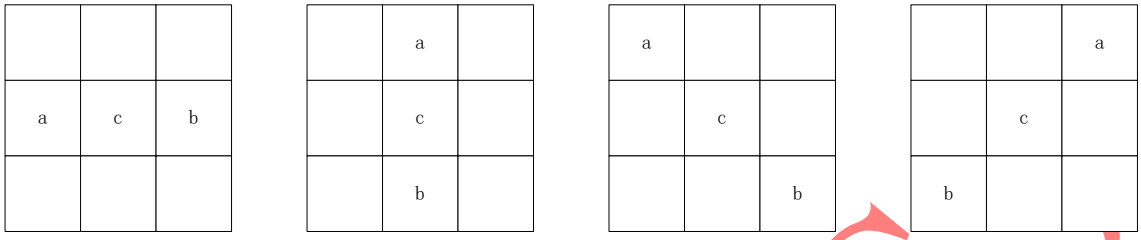
如果样值偏移补偿单元的SaoMode[compIdx]为‘SAO_Edge’，进行以下操作：

a) 第一步，根据 SaoEdgeType[compIdx]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 35）。如果满足以下条件之一，则当前样本的偏移后样本 compIdx 分量值

y[compIdx]=x[compIdx]，其中x[compIdx]是该样本滤波后样本的compIdx分量值，结束本过程；否则，继续执行以下步骤。

1) a或b不与c处在同一片内，且cross_patch_loopfilter_enable_flag的值为‘0’；

2) a或b不在当前图像内。



SaoEdgeType[compIdx]值为0　SaoEdgeType[compIdx]值为1　SaoEdgeType[compIdx]值为2　SaoEdgeType[compIdx]值为3

图35　当前滤波后样本和相邻滤波后样本的关系

b) 第二步，根据表99利用当前样本c的滤波后compIdx分量值 x_c 与相邻样本a和b的滤波后样本compIdx分量值 x_a 和 x_b 的关系确定当前样本compIdx分量的saoOffset[compIdx]。

表99　边缘模式的偏移值

样本分量值的关系	偏移值（saoOffset[compIdx]）
$x_c < x_a$ && $x_c < x_b$	SaoOffset[compIdx][0]
($x_c < x_a$ && $x_c==x_b$) || ($x_c == x_a$ && $x_c < x_b$)	SaoOffset[compIdx][1]
($x_c > x_a$ && $x_c==x_b$) || ($x_c == x_a$ && $x_c > x_b$)	SaoOffset[compIdx][2]
$x_c > x_a$ && $x_c > x_b$	SaoOffset[compIdx][3]
其他	0

c) 第三步，当前样本的偏移后样本compIdx分量值y[compIdx]=Clip1(x[compIdx] + saoOffset[compIdx])，其中x[compIdx]是该样本滤波后样本的compIdx分量值。

9.11.4.3　SAO_Off模式的操作

如果样值偏移补偿单元的SaoMode[compIdx]为‘SAO_Off’，将当前样本滤波后样本compIdx分量的值直接作为该样本偏移后样本compIdx分量值。

9.12　自适应修正滤波

9.12.1　概述

如果PictureAlfEnableFlag[compIdx]的值为0，将偏移后样本分量的值直接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，其中compIdx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。

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。首先根据9.12.2解码各分量的自适应修正滤波系数，然后根据9.12.3导出自适应修正滤波单元，根据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，最后根据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：

a) 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 36 的系数 C8）做以下处理：

$$AlfCoeffLuma[i][8]+=64-\sum_{j=0}^{7}2\times AlfCoeffLuma[i][j]$$

式中 AlfCoeffLuma[i][j]（j=0～7）的位宽是 7 位，取值范围是-64～63；经上述处理后 AlfCoeffLuma[i][8]的取值范围是 0～127。

b) 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 alfCoeffIdxTab[16]）：

```
count = 0
alfCoeffIdxTab[0] = 0
for(i=1; i<alf_filter_num_minus1+1; i++) {
    for(j=0; j<alf_region_distance[i]-1; j++) {
        alfCoeffIdxTab[count+1] = alfCoeffIdxTab[count]
        count = count+1
    }
    alfCoeffIdxTab[count+1] = alfCoeffIdxTab[count] + 1
    count = count + 1
}
for(i=count; i<16; i++)
    alfCoeffIdxTab[i] = alfCoeffIdxTab[count]
```

c) 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）。对系数 AlfCoeffChroma[0][8]和 AlfCoeffChroma[1][8]（即图 36 的系数 C8）分别做以下处理：

$$AlfCoeffChroma[i][8]+=64-\sum_{j=0}^{7}2\times AlfCoeffChroma[i][j],i=0,1$$

式中 AlfCoeffChroma[i][j]（j=0～7）的位宽是 7 位，取值范围是-64～63；经上述处理后 AlfCoeffChroma[i][8]的取值范围是 0～127。

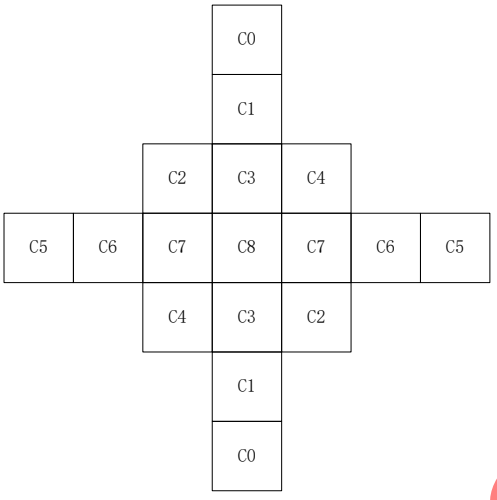


图36　自适应修正滤波系数

9.12.3　导出自适应修正滤波单元

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37）：

a) 将当前最大编码单元C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cross_patch_loopfilter_enable_flag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。

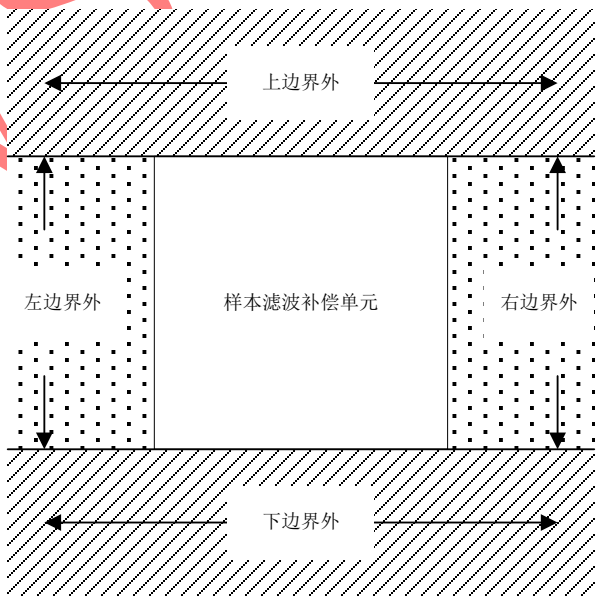


图37　自适应修正滤波单元

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

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filterIdx）：

```
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)
filterIdx = alfCoeffIdxTab[regionTable[index]]
```

式中regionTable[16]={0,1,4,5,15,2,3,6,14,11,10,7,13,12, 9,8}，(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。

9.12.5 自适应修正滤波操作

如果AlfLCUEnableFlag[compIdx][LcuIndex]等于1，则对compIdx分量进行自适应修正滤波，否则不进行自适应修正滤波。如果自适应修正滤波过程中用到的样本为自适应修正滤波单元内的样本，则直接使用该样本进行滤波；否则，按照如下方式进行滤波：

a) 如果该样本在图像边界外，或在片边界外且 cross_patch_loopfilter_enable_flag 的值为‘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。

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：

$$ptmp = AlfCoeffLuma[filterIdx][8] \times p(x,y) + \sum_{j=0}^{7} AlfCoeffLuma[filterIdx][j] \times (p(x-Hor[j], y-Ver[j]) + p(x+Hor[j], y+Ver[j]))$$

```
ptmp = (ptmp + 32) >> 6
p′(x,y) = Clip3(0, (1<<BitDepth)-1, ptmp)
```

式中，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100。

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：

$$ptmp = AlfCoeffChroma[i][8] \times p(x,y) + \sum_{j=0}^{7} AlfCoeffChroma[i][j] \times (p(x-Hor[j], y-Ver[j]) + p(x+Hor[j], y+Ver[j]))$$

ptmp = (ptmp + 32) >> 6

p′(x,y) = Clip3(0, (1<<BitDepth)-1, ptmp)

式中，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100。

表100 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
2	1	1
3	0	1
4	1	-1
5	3	0
6	2	0
7	1	0

9.13 运动信息存储

9.13.1 时域运动信息存储

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：

$x_0 = (x >> 4) << 4$

$y_0 = (y >> 4) << 4$

$$nWidth = \begin{cases} PictureWidthInMinBu \times MiniSize - x_0, & x_0 + 16 \geq PicutreWidthInMinBu \times MiniSize \\ 16, & x_0 + 16 < PicutreWidthInMinBu \times MiniSize \end{cases}$$

$$nHeight = \begin{cases} PictureHeightInMinBu \times MiniSize - y_0, & y_0 + 16 \geq PictureHeightInMinBu \times MiniSize \\ 16, & y_0 + 16 < PictureHeightInMinBu \times MiniSize \end{cases}$$

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(x_0+nWidth/2,y_0+nHeight/2)亮度样本所在编码单元的预测类型为帧内，或(x_0+nWidth/2,y_0+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参考帧索引值设为-1；

b) 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。

9.13.2 空域运动信息存储

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)：

$x_0 = (x >> 2) << 2$

$y_0 = (y >> 2) << 2$

其次，设是(x_0,y_0)亮度样本所在的预测单元，RefIdxL0和RefIdxL1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帧内预测模式，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。

b) 否则,如果 X 的 AffineFlag 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。

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。

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。

c) 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 RefIdxL0，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。

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 RefIdxL1，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。

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 RefIdxL0，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 RefIdxL1。

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(mvAffineL0_x, mvAffineL0_y)，mvAffineL1(mvAffineL1_x, mvAffineL1_y)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));
```

9.14 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：

a) InterPredRefMode 不同；

b) InterPredRefMode 均为 0， L0 运动矢量不相等或 L0 参考索引不相等；

c) nterPredRefMode 均为 1，L1 运动矢量不相等或 L1 参考索引不相等；

d) nterPredRefMode 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。

9.15 更新历史运动信息表

完成当前预测单元的解码后，如果当前预测单元不是仿射预测单元且NumOfHmvpCand大于0，根据当前预测块的运动信息更新历史运动信息表HmvpCandidateList；否则，不执行本条定义的操作。

a) 将 hmvpIdx 初始化为 0。

b) 如果 CntHmvp 等于 0,则 HmvpCandidateList[CntHmvp]等于当前预测单元的运动信息,CntHmvp 加 1。

c) 否则，根据 9.14 定义的方法判断当前预测块的运动信息和 HmvpCandidateList[hmvpIdx]是否相同：

1) 如果运动信息相同，执行步骤 d)，否则，hmvpIdx 加 1。

2) 如果 hmvpIdx 小于 CntHmvp，执行步骤 c)；否则，执行步骤 d)。

d) 如果 hmvpIdx 小于 CntHmvp，则：

1) i 从 hmvpIdx 到 CntHmvp-1，令 HmvpCandidateList[i]等于 HmvpCandidateList[i+1]；

2) HmvpCandidateList[CntHmvp-1]等于当前预测单元的运动信息。

否则，如果 hmvpIdx 等于 CntHmvp 且 CntHmvp 等于 NumOfHmvpCand，则：

1) i 从 0 到 CntHmvp-1，令 HmvpCandidateList[i]等于 HmvpCandidateList[i+1]；

2) HmvpCandidateList[CntHmvp-1]等于当前预测单元的运动信息。

否则，如果 hmvpIdx 等于 CntHmvp 且 CntHmvp 小于 NumOfHmvpCand，则 HmvpCandidateList[CntHmvp]等于当前预测单元的运动信息，CntHmvp 加 1。

9.16 仿射运动矢量继承

记(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。

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(mv0_x,mv0_y)、mv1(mv1_x,mv1_y)和 mv2(mv2_x,mv2_y)：

```
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(mv0_x, mv0_y)和 mv1(mv1_x, mv1_y)：

```
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)
```

f) 按 9.17 定义的方法由仿射控制点运动矢量组 mvsAffine 导出当前预测单元中的各个亮度预测子块的运动矢量。

9.17 仿射运动单元子块运动矢量阵列的导出

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（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)。

a) 计算变量 dHorX、dVerX、dHorY 和 dVerY：

```
dHorX = (mv1_x - mv0_x) << (7 - Log(width))
dHorY = (mv1_y - mv0_y) << (7 - Log(width))
```

b) 如果 mvsAffine 中有 3 个运动矢量，则：

```
dVerX = (mv2_x - mv0_x) << (7 - Log(height))
dVerY = (mv2_y - mv0_y) << (7 - Log(height))
```

c) 否则（mvsAffine 中有 2 个运动矢量）：

```
dVerX = - dHorY
dVerY = dHorX
```

(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，每个子块的宽度和高度分别是subwidth和subheight，见图38。当前预测单元亮度预测块的左上角样本所在的子块为A，右上角样本所在的子块为B，左下角样本所在的子块为C。

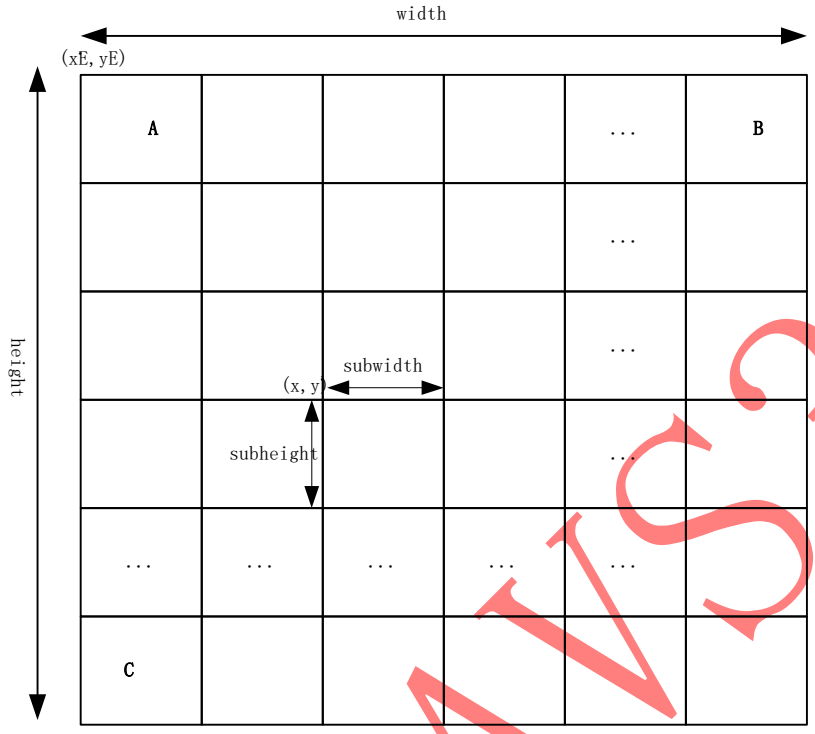


图38　样本的位置

d) 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))
```

e) 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))
```

附 录 A
（规范性附录）
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，本部分不做规定。

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、目标设备显示和内容元数据扩展、感兴趣区域参数扩展、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应采用上述方法。

附 录 B
（规范性附录）
档次和级别

B.1 概述

档次和级别提供了一种定义本部分的语法和语义的子集的手段。档次和级别对位流进行了各种限制，同时也就规定了对某一特定位流解码所需要的解码器能力。档次是本部分规定的语法、语义及算法的子集。符合某个档次规定的解码器应完全支持该档次定义的子集。级别是在某一档次下对语法元素和语法元素参数值的限定集合。在给定档次的情况下，不同级别往往意味着对解码器能力和存储器容量的不同要求。

本附录描述了不同档次和级别所对应的各种限制。所有未被限定的语法元素和参数可以取任何本部分所允许的值。如果一个解码器能对某个档次和级别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档次和级别上符合本部分。如果一个位流中不存在某个档次和级别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档次和级别所允许的范围，则认为此位流在这个档次和级别上符合本部分。

profile_id和level_id定义了位流的档次和级别。

B.2 档次

本部分定义的档次见表B.1。

表B.1 档次

profile_id的值	档次
0x00	禁止
0x20	基准8位档次（Main profile）
0x22	基准10位档次（Main-10bit profile）
其他	保留

对于一个给定的档次，不同的级别支持相同的语法子集。

基准8位档次的位流应满足以下条件：

——profile_id 的值应为 0x20。

——progressive_sequence 的值为‘1’。

——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。
——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。
——B.3.2 规定的级别限制。
——支持的级别包括：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位档次的位流应满足以下条件：
——profile_id 的值应为 0x22。
——progressive_sequence 的值应为‘1’。
——chroma_format 的值应为‘01’。
——sample_precision 的值应为‘001’或‘010’。
——encoding_precision 的值应为‘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。

——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。
——B.3.2 规定的级别限制。
——支持的级别包括：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。

B.3 级别

B.3.1 本部分定义的级别

本部分定义的级别见表B.2。

表B.2　级别

level_id的值	级别
0x00	禁止
0x10	2.0.15
0x12	2.0.30
0x14	2.0.60
0x20	4.0.30
0x22	4.0.60
0x40	6.0.30
0x42	6.2.30
0x44	6.0.60
0x46	6.2.60
0x48	6.0.120
0x4A	6.2.120
0x50	8.0.30
0x52	8.2.30
0x54	8.0.60
0x56	8.2.60
0x58	8.0.120
0x5A	8.2.120
0x60	10.0.30
0x62	10.2.30
0x64	10.0.60
0x66	10.2.60
0x68	10.0.120
0x6A	10.2.120
其他	保留

B.3.2　与档次无关的级别限制

对于所有档次，每个最大编码单元编码后最大二进制位数的限制见表B.3。

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	$2^{(LcuSizeInBit-1)} + 2^{(LcuSizeInBit\times2)}\times BitDepth\times1.5$

级别2.0.15、2.0.30和2.0.60的参数限制见表B.4。

表B.4　级别 2.0.15、2.0.30 和 2.0.60 的参数限制

参　数	级别		
	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

级别4.0.30和4.0.60的参数限制见表B.5。

表B.5　级别 4.0.30 和 4.0.60 的参数限制

参　数	级　别	
	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

级别6.0.30和6.2.30的参数限制见表B.6。

表B.6　级别 6.0.30 和 6.2.30 的参数限制

参　数	级　别	
	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

表 B.6（续）

参　数	级　别	
	6.0.30	6.2.30
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)	

级别6.0.60和6.2.60的参数限制见表B.7。

表B.7　级别 6.0.60 和 6.2.60 的参数限制

参　数	级　别	
	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
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)	

级别6.0.120和6.2.120的参数限制见表B.8。

表B.8　级别 6.0.120 和 6.2.120 的参数限制

参　数	级　别	
	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)	

级别8.0.30和8.2.30的参数限制见表B.9。

表B.9　级别 8.0.30 和 8.2.30 的参数限制

参　数	级　别	
	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)	

级别8.0.60和8.2.60的参数限制见表B.10。

表B.10　级别 8.0.60 和 8.2.60 的参数限制

参　数	级　别	
	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)	

级别8.0.120和8.2.120的参数限制见表B.11。

表B.11　级别 8.0.120 和 8.2.120 的参数限制

参　数	级　别	
	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

表 B.11（续）

参　数	级　别	
	8.0.120	8.2.120
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）	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)	

级别10.0.30和10.2.30的参数限制见表B.12。

表B.12　级别 10.0.30 和 10.2.30 的参数限制

参　数	级　别	
	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
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)	

级别10.0.60和10.2.60的参数限制见表B.13。

表B.13　级别 10.0.60 和 10.2.60 的参数限制

参　数	级　别	
	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

表 B.13（续）

参　数	级　别	
	10.0.60	10.2.60
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)	

级别10.0.120和10.2.120的参数限制见表B.14。

表B.14　级别 10.0.120 和 10.2.120 的参数限制

参　数	级　别	
	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)	

某一级别下 BBV 缓冲区大小是 16384 位的倍数。

与表 B.4 到表 B.14 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
（规范性附录）
位流参考缓冲区管理

C.1 概述

本附录定义了位流参考缓冲区管理（以下简称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。符合本部分的位流不应导致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。符合本部分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）按下式计算：

$$R_{max} = BitRate \times 400 \quad \text{(C.1)}$$

式中：

BitRate——以400bit/s为单位计算视频位流的位率；

R_{max}——编码数据输入BBV缓冲区的最大速率。

C.3 基本操作

C.3.1 数据输入

C.3.1.1 概述

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)按下式计算

$$R(n) = d_n^* \div (\tau(n) - \tau(n+1) + t(n+1) - t(n)) \quad \text{(C.2)}$$

式中：

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）；

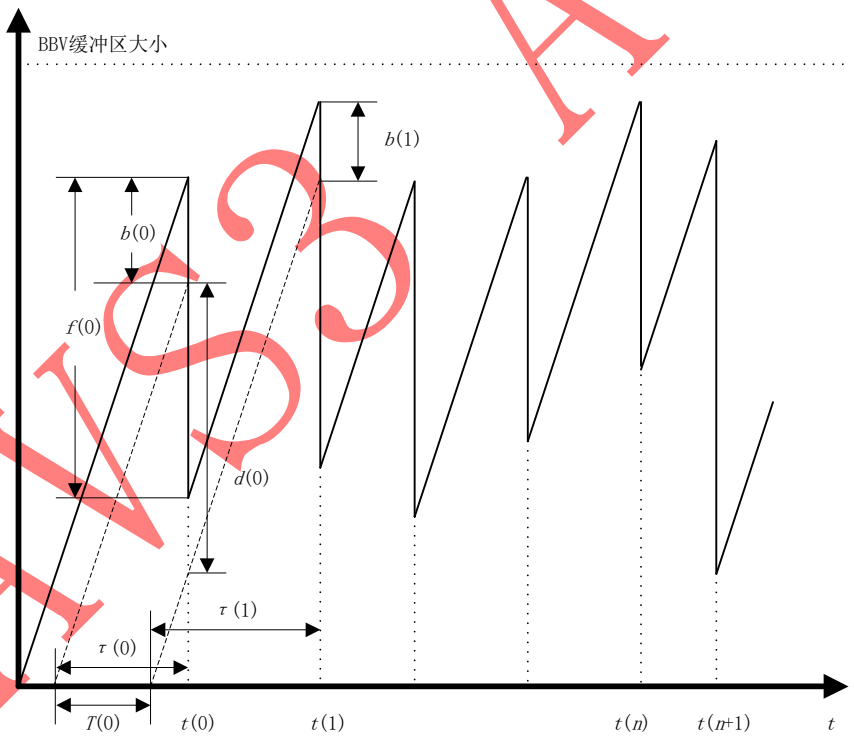
$t(n)$ ——第n幅图像的编码数据从BBV缓冲区移出的时间，单位为秒（s）；

$t(n+1)-t(n)$ ——第n+1幅和第n幅图像的解码时间间隔，单位为秒（s）；

$R(n)$ ——第n幅图像进入BBV缓冲区的速率。

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。输入速率由式（C.2）确定。

所有位流的R(n)均不应大于Rmax。

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。

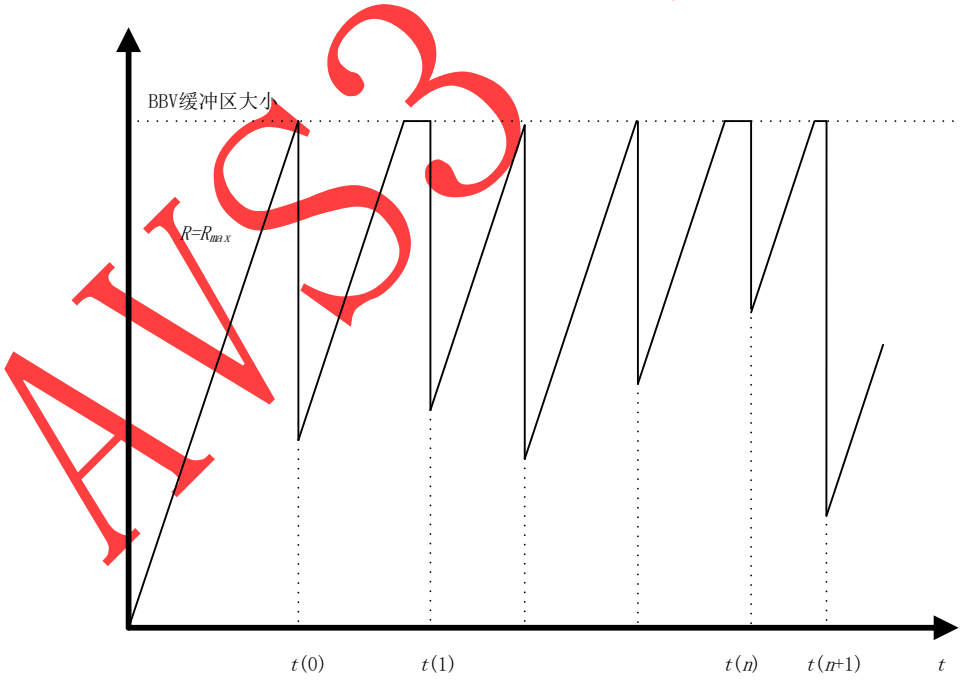
C.3.1.3 方法二

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)＝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)，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，符合本标准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
（规范性附录）
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
（规范性附录）
扫描表

本附录定义扫描表InvScanCoeffInBlk[7][7][pos][2]，通过以下过程生成：

```
for (idx_y = 0; idx_y < 7; idx_y++) {
    size_y = 1 << (idx_y + 1)
    for (idx_x = 0; idx_x < 7; idx_x++) {
        size_x = 1 << (idx_x + 1)
        pos = 0
        num_line = size_x + size_y - 1
        InvScanCoeffInBlk[idx_y][idx_x][0][0] = 0
        InvScanCoeffInBlk[idx_y][idx_x][0][1] = 0
        pos++
        for (l = 1; l < num_line; l++) {
            if (l % 2) {
                x = Min(l, size_x - 1)
                y = Max(0, l - (size_x - 1))
                while (x >= 0 && y < size_y) {
                    InvScanCoeffInBlk[idx_y][idx_x][pos][0] = x
                    InvScanCoeffInBlk[idx_y][idx_x][pos][1] = y
                    pos++
                    x--
                    y++
                }
            }
            else {
                y = Min(l, size_y - 1)
                x = Max(0, l - (size_y - 1))
                while (y >= 0 && x < size_x) {
                    InvScanCoeffInBlk[idx_y][idx_x][pos][0] = x
                    InvScanCoeffInBlk[idx_y][idx_x][pos][1] = y
                    pos++
                    x++
                    y--
                }
            }
        }
    }
}
```

附 录 F
（资料性附录）
高级熵编码器解码器参考描述方法

本附录描述了高级熵编码解码器的参考实现方法，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过程用伪代码描述如下：

```
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过程用伪代码描述如下：

```
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
（规范性附录）
反变换矩阵

本附录定义DCT2、DCT8和DST7型变换的反变换矩阵。

G.1 DCT2 型反变换矩阵

G.1.1 2×2 DCT2反变换矩阵

2×2 DCT2反变换矩阵定义如下：

$DCT2_2$ = {
{ 32 32 }
{ 32 -32 }}

G.1.2 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.1.3 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.1.4 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.1.5 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.1.6 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 }}