ICS 35.040
CCS L 71

团　体　标　准

T/AI 129.1—2024

信息技术 感知无损压缩
第1部分：图像

Information technology - Perceptual lossless compression - Part 1: Image

2024－10－14 发布　　2024－10－14 实施

中关村视听产业技术创新联盟　发布

目　次

前　言

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

本文件是T/AI 129《信息技术 感知无损压缩》的第1部分。T/AI 129已经发布了以下部分：

——第 1 部分：图像。

本文件由数字音视频编解码技术标准工作组提出。

本文件由中关村视听产业技术创新联盟归口。

本文件起草单位：鹏城实验室、北京大学、杭州海康威视数字技术股份有限公司、上海海思技术有限公司、绍兴市北大信息技术科创中心、深圳市大疆创新科技有限公司、中山大学、广州柯维新数码科技有限公司、紫光展锐（上海）科技有限公司、绍兴文理学院、广东博华超高清创新中心有限公司。

本文件起草人：高文、黄铁军、马思伟、郑萧桢、梁凡、孙煜程、杨海涛、潘冬萍、魏亮、张嘉琪、王岩、曹小强、任荟文、宋泽田、罗忆、王丹妮、江桥、陈方栋、王莉、冯俊凯、王苫社、贾川民、张伟民、郑建铧、王泽镐、赵利平、孟现东、熊瑞勤、曾志华、岳泊暄、王丽萍、卫小涛、李荣、罗小伟、赵海英、崔晓冉。

引　言

T/AI 129旨在确立智能媒体压缩的方法，拟由三个部分构成。

——第1部分：图像。目的在于确立智能媒体感知无损图像压缩方法。

——第2部分：符合性。目的在于确立智能媒体感知无损图像压缩的符合性测试方法。

——第3部分：参考软件。目的在于确立智能媒体感知无损图像压缩的软件实现方法。

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9.6与《视频压缩码率控制的方法及装置》（专利号：CN202210687517.7）；9.6与《用于视频压缩的量化方法及装置》（专利号：CN202310035243.8）；9.6与《码率控制的方法及装置》（专利号：CN202310035241.9）；9.6与《码率控制的方法及装置》（专利号：CN202310437128.3）；7.1.4、7.1.5、7.2.4、7.2.5、9.4、9.7与《图像解码方法、编码方法及装置》（专利号：CN202111335613.7）；7.1.4、7.2.4、9.6与《解码方法、编码方法及装置》（专利号：CN202111334223.8）；7.1.5、7.2.5、9.5、9.7.2.2与《一种视频编解码方法及装置》（专利号：CN202111333832.1）；7.1.4、7.2.4、9.4、9.7与《图像编解码方法、装置及存储介质》（专利号：CN202210016199.1）；7.1.5、7.2.5与《一种系数解码方法、装置、图像解码器及电子设备》（专利号：CN202210062532.2）；7.1.4、7.2.4、9.4、9.7与《一种解码、编码方法、装置及其设备》（专利号：CN202210061340.X）；9.7.2.3与《一种图像解码方法、编码方法及装置》（专利号：CN202210062756.3）；9.7.2.3与《一种图像解码方法、编码方法及装置》（专利号：CN202210976078.1）；9.6与《一种视频译码方法、装置及存储介质》（专利号：CN202210887907.9）；7.1.4、7.1.5、7.2.4、7.2.5、8.2.3.9与《一种图像编解码方法及装置》（专利号：CN202210894255.1）；9.7.3.4.2.2、7.1.5、7.2.5与《一种编解码方法、装置及其设备》（专利号：CN202210887893.0）；7.1.4、7.2.4、9.7.1与《图像编解码方法、装置及存储介质》（专利号：CN202211146464.4）；7.1.2、7.1.1.2、7.2.1.1、9.6.2与《图像编解码方法、装置及存储介质》（专利号：CN202211448399.0）；7.1.4、7.2.4、9.7.3.2与《量化参数调整方法、装置、设备及存储介质》（专利号：CN202211741038.5）；6.3、7.1.1.2、7.2.1.1与《一种编码、解码方法， 装置及其设备》（专利号：CN202210273469.7）；7.1.5、7.2.5、9.5、9.7.2.2与《图像编解码方法、装置、电子设备及存储介质》（专利号：CN202310440975.5）；7.1.4、7.2.4、9.7.3.4.3与《一种解码、编码方法、装置及其设备》（专利号：CN202210458047.7）；7.1.4、7.2.4、9.6与《一种视频译码方法、装置及存储介质》（专利号：CN202210612716.1）；7.1.4、7.1.5、7.2.4、7.2.5、8.2.3.9与《一种图像编解码方法及装置》（专利号：CN202210631100.9）；7.1.4、7.2.4、9.7.3.4.2.1与《图像重建方法、装置、设备及存储介质》（专利号：CN202310264176.7）；3.44、3.45、6.4、6.5、7.1、7.2、8.1、F与编解码方法和装置（专利号：202210186914.6）；7.1、7.2、8.1与《编码方法及编码器》（专利号：202210860456.X）；3.44、3.45、6.4、6.5、7.1、7.2、8.1，F与《编解码方法和装置》（专利号：202280015193.0）；3.25、7.2、9.4、9.6、9.7、D.2与《一种编解码方法及装置》（专利号：202211698666.X）；3.44、3.45、6.4、6.5、7.1、7.2、8.1、F与《一种编解码方法及装置》（专利号：202211698950.7）；3.1、3.15、3.31、3.8、7.1.4.1、7.2.1.1、9.3、9.4、9.6.2、9.6.4、9.7.3.1、D.2与《图像编解码方法、装置、编码器、解码器和系统》（专利号：202211696812.5）；3.1、3.31、3.8、7.1.4.1、9.6.2、9.6.4、9.7、9.7.3.1、D.2与《图像编解码方法、装置、编码器、解码器和系统》（专利号：202211698013.1）；3.1、3.15、3.31、5.7、7.1.4.1、7.2.1.1、9.3、9.4、9.6.2、9.7.3.1、D.2与《图像编解码方法、装置、编码器、解码器和系统》（专利号：202211696765.4）；3.1、3.31、6.5、7.1.4.1、7.2.1.1、7.2.5、

8.2.3.2、8.2.3.9与《一种解码方法、编码方法及相关设备》（专利号：202310258261.2）；3.1、3.9、3.21、3.23、3.31、9.6.1、D.1、D.2、D.2.3与《编解码方法和装置》（专利号：202211373846.0）；3.1、3.8、3.9、7.2.1.1、9.6.1、9.7.2.3.2、9.7.2.3.3、D.2与《量化参数获取方法和装置》（专利号：202310283307.6）；3.1、3.15、3.31、5.7、7.1.4.1、7.2.1.1、9.3、9.4、9.6.2、9.7.3.1、D.2与《一种图像编解码方法、装置、编码器、解码器和系统》（专利号：202310301666.X）；3.1、3.29、3.31、3.45、7.1.4.1、9.4、9.6.4、E.1、E.2与《编解码方法和装置》（专利号：202310290271.4）；3.1、3.8、3.21、3.22、3.31、3.34、3.39、7.1.4.2、7.2.1.1、9.6.1、9.7.3.4.2.2、9.7.3.2、D.2、D.3.3.2.1与《编解码方法及电子设备》（专利号：202310284038.5）；3.1、3.8、3.21、3.22、3.31、3.34、3.39、7.1.4.2、7.2.1.1、9.6.1、9.7.3.4.2.1、9.7.3.4.2.2、9.7.3.2、D.2、D.3.3.2.1、D.3.3.2.3与《编解码方法及电子设备》（专利号：202310280913.2）。

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：

专利持有人:北京大学
地址：北京市海淀区颐和园路5号理科2号楼2604室
邮编：100871

专利持有人： 华为技术有限公司
地址：北京市上地信息路3号华为大厦
邮编：100085

专利持有人：杭州海康威视数字技术股份有限公司
地址： 浙江省杭州市滨江区阡陌路555号
邮编: 310051

联 系 人：黄铁军
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 感知无损压缩 第1部分：图像

1 范围

本文件规定了图像感知无损压缩方法的编码位流的结构、位流的语法和语义、解析过程和解码过程。

本文件适用于静止图像、网络视频、数字电影、即时通信、视频监控、虚拟现实和增强现实等领域的高速视频压缩传输应用。

2 规范性引用文件

本文件无规范性引用文件。

3 术语和定义

下列术语和定义适用于本文件。

3.1

编码单元 coding unit

包括一个宽为M高为N的亮度样值矩阵和对应的色度样值矩阵，由图像划分得到。

3.2

编码块 coding block

一个宽为M高为N的样值矩阵，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。

3.3

编码图像 coded picture

一幅图像的编码表示。

3.4

残差 residual

样本或数据元素的重建值与其预测值之差。

3.5

残差集合 residual set

包括一个或多个残差的集合，由残差块划分得到，一个残差块由一个或多个互不重叠的残差集合构成。

3.6

残差块 residual block

一个M×N的残差矩阵，由编码块对应的残差构成。

3.7

残差组 residual group

包括多个残差的组，由残差集合划分得到，一个残差集合由一个或多个互不重叠的残差组构成。

3.8

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。

3.9

点预测模式 point prediction mode

编码单元中的部分样值须依次获取其预测值的预测模式。

3.10

二元符号 bin

组成二元符号串的符号，包括‘0’和‘1’。

3.11

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。

3.12

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。

3.13

反量化 dequantization

对量化残差缩放后得到残差重建值的过程。

3.14

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。

3.15

缓冲区 buffer

用于在解码前临时存放位流的存储区域。

3.16

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。

3.17

划分方式 partition type

划分获得的子集的组织方式。

3.18

解码图像 decoded picture

解码器根据位流重建的图像。

3.19

解析过程 parse

由位流获得语法元素的过程。

3.20

块 block

一个M×N（M列N行）的样值矩阵。

3.21

块复制帧内预测 intra block copy prediction

在相同解码图像中使用已解码的样值复制至当前样本区域，作为当前样本预测值的模式。复制样值的区域与当前样本区域大小相同。

3.22

块矢量 block vector

用于块复制帧内预测模式的一维矢量，其值为当前预测块和参考块之间的坐标偏移量，其中，当前预测块与参考块均在当前图像中。

3.23

块预测模式 block prediction mode

编码单元中的所有样值可独立获取其预测值的预测模式。

3.24

亮度 luma

表示图像明暗程度的信号。

注：符号为Y，表示亮度信号的样值矩阵或单个样值。

3.25

量化参数 quantization parameter

在解码过程对量化残差进行反量化的参数。

3.26

量化残差 quantization residual

反量化前量化残差的值。

3.27

色度 chroma

用于表示两个色彩差值分量信号。

注：符号为Cr和Cb或者Co和Cg，表示两种色差信号的样值矩阵或单个样值。

3.28

填充位 stuffing bits

编码时插入位流中的位串，在解码时被丢弃。

3.29

条带 slice

由按光栅扫描顺序排列的相邻若干编码单元构成的样值矩阵。

3.30

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，最右边位是最低有效位。

3.31

位流 bitstream

编码图像的全部或部分样本所形成的二进制数据流。

3.32

位流段 chunk

一个条带编码产生的位流可划分为多个位流段，水平排列的条带编码产生的位流段交织构成一幅编码图像的位流。

3.33

样本 sample

构成图像的基本元素。

3.34

预测 prediction

预测过程的具体实现。

3.35

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。

3.36

预测单元 prediction unit

可包含一个亮度预测块和对应的色度预测块。

3.37

预测过程 prediction process

使用先前已解码的样值获取当前样值的预测值。

3.38

预测块 prediction block

一个使用相同预测过程的M×N的样值矩阵，由编码单元划分得到。

3.39

预测值 prediction value

在样值的解码过程中，用到的先前已解码的样值。

3.40

语法元素 syntax element

位流中的数据单元解析后的结果。

3.41

样值 sample value

样本的幅值。

3.42

字节 byte

8位的位串。

3.43

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。

3.44

子流 substream

编码一个分量的样本所形成的二进制数据流。

3.45

子流片 substream segment

子流片包含子流索引和子流片数据，子流片数据从子流划分得到，子流片数据长度与比特深度以及图像格式有关，子流索引用以指示当前子流片所属的子流。子流片交织成位流段。

4 缩略语

下列缩略语适用于本文件。

BBV 位流参考缓冲区管理（bitstream buffer verifier）

BRC 块级重建值补偿（block reconstruction compensation）

BWQ 块级量化参数调整（block wise quantization）

CB 编码块（coding block）

CU 编码单元（coding unit）

DP 差值预测（difference prediction）

IBC 帧内块复制（intra block copy）

MSB 最高有效位（most significant bit）
PWQ 像素级级量化参数调整（pixel wise quantization）
RG 残差组（residual group）
RS 残差集合（residual set）

5 约定

5.1 概述

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

5.2 算术运算符

算术运算符定义见表1。

表1 算术运算符定义

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

5.3 逻辑运算符

逻辑运算符定义见表2。

表2　逻辑运算符定义

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

5.4　关系运算符

关系运算符定义见表3。

表3　关系运算符定义

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

5.5　位运算符

位运算符定义见表4。

表4　位运算符定义

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

5.6　赋值

赋值运算定义见表5。

表5　赋值运算定义

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

5.7　数学函数

数学函数定义见式（1）～式（7）。

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

式中：

x ——自变量x。

$$\mathrm{Ceil}(x)=\lceil x\rceil \qquad \cdots\cdots (2)$$

式中：

x ——自变量x。

$$\mathrm{Clip3}(i,j,x)=\begin{cases}i & ; \quad x<i\\ j & ; \quad x>j\\ x & ; \quad \text{其他}\end{cases} \qquad \cdots\cdots (3)$$

式中：

x ——自变量x；

i ——下界；

j ——上界。

$$\mathrm{Min}(x,y)=\begin{cases}x & ; \quad x<=y\\ y & ; \quad x>y\end{cases} \qquad \cdots\cdots (4)$$

式中：

x ——自变量x；

y ——自变量y。

$$\mathrm{Max}(x,y)=\begin{cases}x & ; \quad x>=y\\ y & ; \quad x<y\end{cases} \qquad \cdots\cdots (5)$$

式中：

x ——自变量x；

y ——自变量y。

$$\mathrm{Sign}(x)=\begin{cases}1 & ; \quad x>=0\\ -1 & ; \quad x<0\end{cases} \qquad (6)$$

式中：

x ——自变量x。

$$\mathrm{Log}(x)=\log_2 x \qquad (7)$$

式中：

x ——自变量x。

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文件的位流语法的超集，附加的语法限制在相关条中说明。

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

表7 语法描述的伪代码

伪代码	描述符
/*语句是一个语法元素的描述符，或者说明语法元素的存在、类型和数值，下面给出两个例子。*/	
syntax_element	ce(v)
conditioning statement	

表 7（续）

伪代码	描述符
/*花括号括起来的语句组是复合语句，在功能上视作单个语句。*/	
{	
statement	
…	
}	
/*“while”语句测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。*/	
while (condition)	
statement	
/*“do … while”语句先执行循环体一次，然后测试condition是否为TRUE，如果为TRUE，则重复执行循环体，直到condition不为TRUE。*/	
do	
statement	
while (condition)	
/*“if …else”语句首先测试condition，如果为TRUE，则执行primary语句，否则执行alternative语句。如果alternative语句不需要执行，结构的“else”部分和相关的alternative语句可忽略。*/	
if (condition)	
primary statement	
else	
alternative statement	
/*“for”语句首先执行initial语句，然后测试condition，如果condition为TRUE，则重复执行primary语句和subsequent语句直到condition不为TRUE。*/	
for (initial statement; condition; subsequent statement)	
primary statement	
/*“break”语句用于do-while、while和for循环体中，可使当前循环体立即终止循环。*/	
break	

解析过程和解码过程用文字和类似C语言的伪代码描述。

5.9.2 函数

5.9.2.1 概述

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

5.9.2.2 byte_aligned( )

如果位流的当前位置是字节对齐的，返回TRUE，否则返回FALSE。

5.9.2.3 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8。

表8 描述符

描述符	说明
ce(v)	变长的语法元素。解析过程在8.2 中定义
b(n)	一个任意取值的n个二进制位。解析过程由函数read_bits(n)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
r(n)	连续n个‘0’。解析过程由函数read_bits(n)的返回值规定
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

5.9.4 保留、禁止和标记位

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

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

“禁止”定义了一些特定语法元素值，这些值不应出现在符合本文件的位流中。

位流中的“保留位”（reserved_bits）表明保留了一些语法单元用于将来对本文件的扩展，解码处理应忽略这些位。

6 编码位流的结构

6.1 图像

6.1.1 概述

图像是位流的最高层语法结构，本文件仅支持逐行图像。一幅图像的编码数据由图像头开始。

图像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，或Co和Cg）。样本矩阵元素的值为整数。亮色度之间的关系，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。图像的解码处理包括解析过程和解码过程。

如果MultiplexingEnableFlag等于1，则位流由位流段交织而成。一个条带编码产生的位流可划分为多个位流段，水平排列的条带编码产生的位流段交织构成一幅编码图像的位流。其中，条带编码产生的位流是由子流片交织而成。子流片包含子流索引和子流片数据。一个分量的样本所形成的二进制数据流可划分为多个等尺寸的子流片数据。子流片数据长度与比特深度以及图像格式有关。子流索引用以指示当前子流片所属的子流。

6.1.2 图像格式

6.1.2.1 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　。

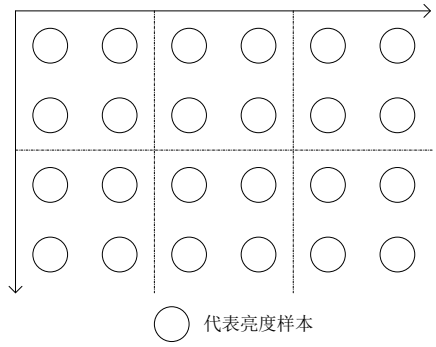


图1　4:0:0 格式下亮度样本位置

6.1.2.2　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。

一种可能的亮度和色度样本位置见图2。

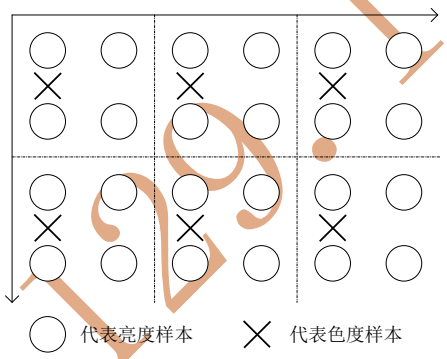


图2　4:2:0 格式下亮度和色度样本位置

6.1.2.3　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。

亮度和色度样本位置见图3。

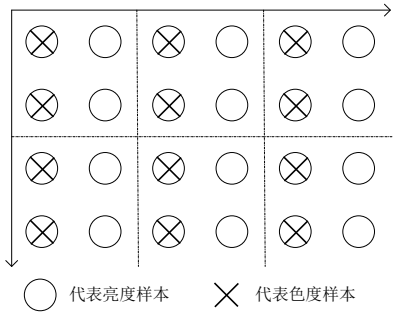


图3　4:2:2 格式下亮度和色度样本位置

6.1.2.4 4:4:4 格式

对于4:4:4格式，Cb和Cr矩阵(Co和Cg矩阵)在水平和垂直方向的尺寸都和Y矩阵一样。

亮度和色度样本位置见图4。

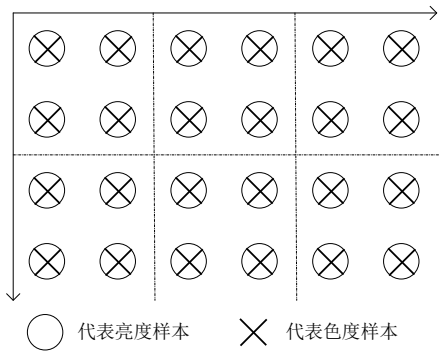


图4 4:4:4 格式下亮度和色度样本位置

6.2 条带

条带是图像中的矩形区域，条带之间不应重叠，每个条带的尺寸一致，一种可行的条带结构见图5 。其中，最小的矩形为宽为16高为2的编码单元，粗线为条带划分线，A~I表示尺寸相同的9个条带。

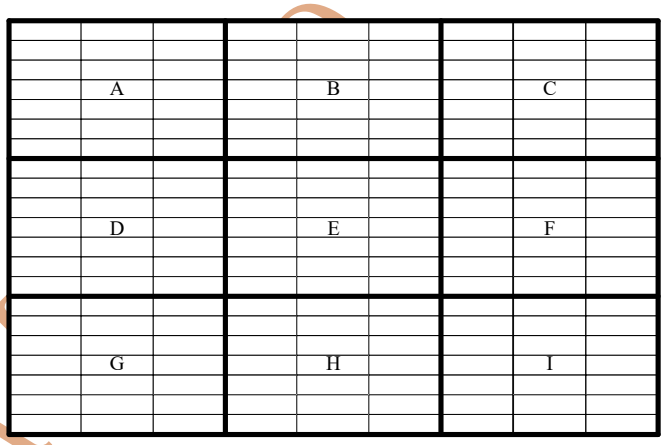


图5 条带结构

6.3 位流段

如果MultiplexingEnableFlag等于1，则位流由位流段交织而成。一个条带编码产生的位流可划分为多个位流段，水平排列的条带编码产生的位流段交织构成一幅编码图像的位流。

记当前图像每行的条带个数为a，每列的条带个数为b，每个条带中的位流段数量为n，其中：

a = PictureWidthInSlice

b = PictureHeightInSlice

n = ChunkNum

位流段的交织如图6所示。

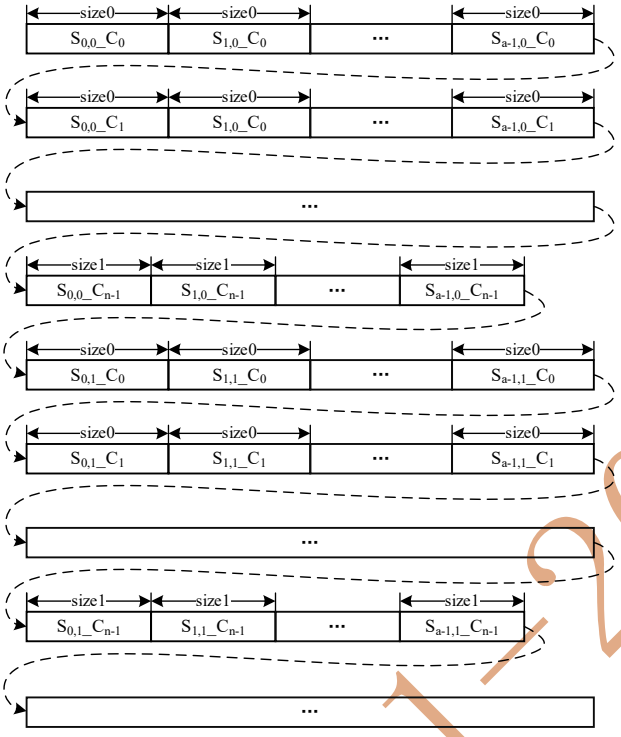


图6　位流段交织示意图

其中，$S_{i,j}_C_k$表示水平索引为i，竖直索引为j的条带的第(k+1)个位流段，size0，size1的计算过程为：

size0 = ChunkSizeInBit

size1 = TotalBits – (n – 1) ×ChunkSizeInBit

6.4　子流片

如果MultiplexingEnableFlag等于1，则条带位流由子流片交织而成。子流片包含子流索引和子流片数据。一个分量的样本所形成的二进制数据流可划分为多个等尺寸的子流片数据。子流片数据长度与比特深度以及图像格式有关。子流索引用以指示当前子流片所属的子流。

记当前图像的子流片个数为n，alignmentBit长度为m，其中：

size = SubstreamSegmentSize – 2

n = SubstreamSegmentNumAll

m = SliceAlignmentBitNum

子流片的交织如图7　所示。

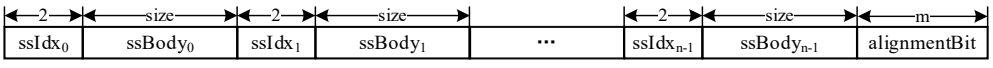


图7　子流片交织示意图

其中，$ssIdx_i$表示第i个子流片的子流索引（长度为2比特），$ssBody_i$表示第i个子流片的子流片数据（长度为SubstreamSegmentSize–2比特），alignmentBits表示条带对齐填充位（其值均为‘0’）。

6.5 编码单元，残差块，残差集合和残差组

图像划分为相同大小的编码单元，编码单元之间不应重叠，编码单元不应超出图像边界。其中，CuWidth为编码单元的宽，其值等于16，CuHeight为编码单元的高，其值等于2，CuSize为编码单元的面积，其值等于CuWidth×CuHeight。

编码单元包括一个或三个编码块。

一个编码块对应一个残差块，残差块中的残差被划分为一个或多个残差集合，每个残差集合被划分为若干个残差组。

如果MultiplexingEnableFlag等于1，则编码单元相同分量的语法元素进入同一子流。

7 位流的语法和语义

7.1 语法描述

7.1.1 图像语法

7.1.1.1 图像定义

图像定义见表9。

表9 图像定义

图像定义	描述符
picture( ) {	
picture_header( )	
picture_data( )	
}	

7.1.1.2 图像头定义

图像头定义见表10。附录G提供了图像头字节说明用于在接口档次下适配传输层。

表10 图像头定义

图像头定义	描述符
picture_header( ) {	
profile_id	u(8)
if (profile_id == 0x20) { /* Frame buffer profile */	
MultiplexingEnableFlag = 0	
} else {	
MultiplexingEnableFlag = 1	
}	
input_image_width	u(32)
input_image_height	u(32)
slice_width	u(32)

表 10（续）

图像头定义	描述符
slice_height	u(32)
image_format	u(3)
bit_depth	u(5)
reserved_bits	r(4)
target_bpp	u(12)
padding_type_flag	u(1)
lossless_enable_flag	u(1)
transform_enable_flag	u(1)
dp_enable_flag	u(1)
brc_enable_flag	u(1)
ibc_enable_flag	u(1)
pwq_enable_flag	u(1)
bwq_enable_flag	u(1)
vbr_enable_flag	r(1)
pwq_max_qp	u(4)
bwq_complex_th	u(3)
qp_refine_th0	u(4)
qp_refine_th1	u(4)
jnd_qp	u(4)
strict_jnd_qp	u(4)
chunk_size_in_cu	u(32)
chunk_num	u(32)
slice_cu_num_max_bit	u(8)
reserved_bits	r(2)
padding_stuff_flag	u(1)
substream_expansion_ratio_log2	u(3)
substream_segment_size	u(10)
transmission_delay_cu	u(16)
rc_buffer_size	u(16)
for (compL=0; compL<5; compL++) {	
for (compC=0; compC<5; compC++) {	
reserved_bits	r(4)
rc_qp_bias[compL][compC]	u(4)
}	
}	
rc_decrease_step_log2	u(8)
rc_fullness_calc_multiplier	u(8)

表 10（续）

图像头定义	描述符
reserve_bits	u(3)
rc_fullness_calc_shift	u(5)
rc_extra_buffer_decrease_step_log2	u(8)
rc_extra_buffer_penalty_log2_minus3	u(8)
reserved_bits	r(6)
rc_target_end_ratio_minus4	u(2)
rc_ratio0	u(8)
reserved_bits	r(5)
rc_param0	u(11)
reserved_bits	r(7)
rc_ratio1	u(9)
reserved_bits	r(5)
rc_param1	u(11)
reserved_bits	r(5)
rc_ratio2	u(11)
reserved_bits	r(5)
rc_param2	u(11)
rc_max_relative_bits	u(8)
for (comp=0; comp<5; comp++) {	
reserved_bits	r(5)
rc_lossless_bits[comp]	u(11)
}	
reserved_bits	r(5)
rc_avg_lossless_bits	u(11)
reserved_bits	r(5)
rc_bits_offset	u(11)
reserved_bits	r(5)
rc_max_lossless_bits	u(11)
reserved_bits	r(6)
rc_relative_th	u(10)
reserved_bits	r(5)
rc_complex_th	u(3)
}	

7.1.1.3 图像数据定义

图像数据定义见表11。

表11　图像数据定义

图像数据定义	描述符
picture_data( ) {	
SliceAlignAdj = (((((SliceWidth × TargetBpp) >> 4) + 7) >> 3) << 3) – ((SliceWidth × TargetBpp) >> 4)	
for (j=0; j<PictureHeightInSlice; j++) {	
for (i=0; i<PictureWidthInSlice; i++) {	
slice_data(i, j)	
}	
}	
}	

7.1.2　条带定义

条带定义见表12。

表12　条带定义

条带定义	描述符
slice_data(i, j) {	
CurrSlicePosX = SliceInfo[i][j]->posX	
CurrSlicePosY = SliceInfo[i][j]->posY	
PrevComplexityLevel = 0	
CurrCuIdx = 0　/* CU index within current Slice */	
CurrPos = 0	
SubstreamBits[0] = 0	
SubstreamBits[1] = 0	
SubstreamBits[2] = 0	
SubstreamSegmentNum[0] = 0	
SubstreamSegmentNum[1] = 0	
SubstreamSegmentNum[2] = 0	
cost = BitDepth[0] × CuSize	
if (image_format == ‘001’) {　/* YUV420 */	
cost += ((BitDepth[1] × (CuWidth / 2) × (CuHeight / 2)) << 1)	
} else if (image_format == ‘010’) {　/* YUV422 */	
cost += ((BitDepth[1] × (CuWidth / 2) × CuHeight) << 1)	
} else if (image_format == ‘011’ || image_format == ‘100’) {　/* YUV444 or RGB444 */	
cost += ((BitDepth[MultiplexingEnableFlag ? 0 : 1] × CuSize) << 1)	
}	
ExtraBufferSize = (cost – ((CuSize × TargetBpp) >> 4)) << 1	
MaxBufferSize = RcBufferSize + ExtraBufferSize　/* Actual buffer size for rate control */	

表 12（续）

条带定义	描述符
EndDecreaseBits = MaxBufferSize – DelayBits + (!MultiplexingEnableFlag ? 0 : (image_format == '000'? 1 : 3) × (SubstreamSegmentSize – 1)	
EndControlBlocks = (EndDecreaseBits + (1 << RcDecreaseStepLog2) – 1) >> RcDecreaseStepLog2	
EndControlBegin = SliceWidthInCu × SliceHeightInCu – EndControlBlocks	
RemainBlksLog2 = SliceCuNumMaxBit	
CurrRcBufferLevel = 0	
PhysicalBufferLevel = CurrSlicePosY == 0 ? 0 : DelayBits	
VirtualStartStuff = CurrSlicePosY == 0 ? DelayBits : 0	
VirtualEndStuff = 0	
if (MultiplexingEnableFlag) {	
SubstreamSegmentNumAll = TotalBits / SubstreamSegmentSize	
for (idx=0; idx<SubstreamSegmentNumAll; idx++) {	
substream_segment( )	
}	
SliceAlignmentBitNum = TotalBits – SubstreamSegmentSize × substreamSegmentNumAll	
for (bitIdx=0; bitIdx<SliceAlignmentBitNum; bitIdx++) {	
slice_alignment_bit0	f(1)
}	
} else {	
coding_unit(i, j)	
}	
}	
注：SubstreamBits[ssIdx]用于记录第(ssIdx+1)子流的已解析比特数。	

7.1.3 子流片定义

子流片定义见表 13。

表13 子流片定义

子流片定义	描述符
substream_segment( ) {	
substream_index	b(2)
substream_segment_data	b(v)
}	

7.1.4 编码单元语法

7.1.4.1 编码单元定义

编码单元定义见表 14。

表14　编码单元定义

编码单元定义	描述符
coding_unit(sliceIdxX, sliceIdxY) {	
alignNum = 0	
remainPixels = 0	
isHfSliceWidthInCu = (((SliceWidth >> 5) & 1) == 0) ? 1 : 0	
for (x=0; x<SliceWidthInCu; x++) {	
for (y=0; y<SliceHeightInCu; y++) {	
CuBits = 0　/* Current CU bits without counting sub stream header */	
tmpSliceAlignAdj = 0　/* Additional bits for interface profile */	
RemainBlksLog2 = (RemainBlksLog2 > 4 && !((SliceWidthInCu × SliceHeightInCu – CurrCuIdx) & (1 << RemainBlksLog2))) ? RemainBlksLog2 – 1 : RemainBlksLog2	
bppAdj = (EndTargetFullness – PhysicalBufferLevelRecord[CurrPos]) << 7	
bppAdj = bppAdj >= 0 ? bppAdj >> (5 + RemainBlksLog2) : –((–bppAdj) >> (5 + RemainBlksLog2))	
bppAdj = Clip3(–512, 1536, bppAdj)	
Currbpp = (Targetbpp << 3) + bppAdj	
bitsRecordSubstream[0] = SubstreamBits[0]	
FallbackFlag = 0	
CbLumaSize = CuWidth × CuHeight	
if (image_format == ‘000’) {　/* YUV400 */	
CbChromaSize = 0	
ModeBits = DpEnableFlag ? 13 : 8	
} else if (image_format == ‘001’) {　/* YUV420 */	
CbChromaSize = CbLumaSize >> 2	
ModeBits = DpEnableFlag ? 20 : 15	
} else if (image_format == ‘010’) {　/* YUV422 */	
CbChromaSize = CbLumaSize >> 1	
ModeBits = DpEnableFlag ? 20 : 15	
} else if (image_format == ‘011’ || image_format == ‘100’) {　/* YUV444 or RGB444 */	
CbChromaSize = CbLumaSize	
ModeBits = DpEnableFlag ? 26 : 21	
}	
if (CurrCuIdx >= EndControlBegin) {	
VirtualEndStuff = VirtualEndStuff + (1 << RcDecreaseStepLog2)	
}	
TmpCurrRcBufferLevel = PhysicalBufferLevel + VirtualStartStuff + VirtualEndStuff	
Allowfallback = MultiplexingEnableFlag ? 6 : 0 + ModeBits + (BitDepth[0] × CbLumaSize + BitDepth[MultiplexingEnableFlag ? 0 : 1] × CbChromaSize × 2 – ((CuSize × TargetBpp) >> 4)) > MaxBufferSize – TmpCurrRcBufferLevel	
coding_block_data_substream0(x + SliceWidthInCu × sliceIdxX, y + SliceHeightInCu × sliceIdxY)	

表 14（续）

编码单元定义	描述符
CurSubstreamBits[0] = SubstreamBits[0] – bitsRecordSubstream[0]	
CuBits += CurSubstreamBits[0]	
if (image_format != ‘000’) { /* Not YUV400 */	
bitsRecordSubstream[1] = SubstreamBits[1]	
coding_block_data_substream1(x + SliceWidthInCu × sliceIdxX, y + SliceHeightInCu × sliceIdxY)	
CurSubstreamBits[1] = SubstreamBits[1] – bitsRecordSubstream[1]	
CuBits += CurSubstreamBits[1]	
bitsRecordSubstream[2] = SubstreamBits[2]	
coding_block_data_substream2(x + SliceWidthInCu × sliceIdxX, y + SliceHeightInCu × sliceIdxY)	
CurSubstreamBits[2] = SubstreamBits[2] – bitsRecordSubstream[2]	
CuBits += CurSubstreamBits[2]	
}	
if (!VbrEnableFlag) {	
if (MultiplexingEnableFlag) {	
if (!(CurrSlicePosY == 0 && CurrCuIdx < TransmissionDelayCu)) {	
alignNum++	
}	
if ((alignNum <<5) + remainPixels >= SliceWidth) {	
tmpSliceAlignAdj = SliceAlignAdj	
alignNum = 0	
if (isHfSliceWidthInCu) {	
remainPixels = 0	
} else {	
remainPixels = (remainPixels == 0) ? 16 : 0	
}	
}	
}	
bitsGap=0	
if (!AllowFallback && PaddingTypeFlag && PaddingStuffFlag && ((x + SliceWidthInCu × sliceIdxX) >= (PictureWidthInSlice × SliceWidthInCu – (ImageWidth – InputImageWidth) / CuWidth) && (y + SliceHeightInCu × sliceIdxY) < (PictureHeightInSlice × SliceHeightInCu – (ImageHeight – InputImageHeight) / CuHeight))) {	
bitsGap = Min((image_format == ‘000’? 1 : 3) × SubstreamSegmentSize, Currbpp >> 2) – CuBits – 6 – (VirtualEndStuff > 0 ? (1 << RcDecreaseStepLog2) : 0)	
bitsGap = bitsGap < 0 ? 0 : bitsGap	
}	

表 14（续）

编码单元定义	描述符
if (((CurrSlicePosY == 0 && CurrCuIdx >= TransmissionDelayCu) || CurrSlicePosY != 0) && (PhysicalBufferLevel + CuBits – ((CuSize × TargetBpp) >> 4) + tmpSliceAlignAdj) < 0) { /* Underflow */	
underflowBits = ((CuSize × TargetBpp) >> 4) + tmpSliceAlignAdj – (PhysicalBufferLevel + CuBits)	
} else {	
underflowBits = 0	
}	
underflowBits=Max(underflowBits ,bitsGap)	
StuffBits = 0	
if (image_format == ‘000’) { /* 1 substream for YUV400 */	
stuffing_data_substream0(underflowBits, CurSubstreamBits, CurSubstreamBits)	
StuffBits = StuffBits[0]	
} else { /* 3 substreams for other cases */	
maxStuffSize = SubstreamSegmentSize – 2	
zeroBits[0] = (CurSubstreamBits[0] + underflowBits > maxStuffSize) ? (maxStuffSize – CurSubstreamBits[0]) : underflowBits	
if (CurSubstreamBits[0] + CurSubstreamBits[1] + underflowBits > 2 * maxStuffSize) {	
zeroBits[1] = maxStuffSize – CurSubstreamBits[1]	
} else if (CurSubstreamBits[0] + underflowBits > maxStuffSize) {	
zeroBits[1] = CurSubstreamBits[0] + underflowBits – maxStuffSize	
} else {	
zeroBits[1] = 0	
}	
zeroBits[2] = (CurSubstreamBits[0] + CurSubstreamBits[1] + underflowBits > (maxStuffSize << 1) ? (CurSubstreamBits[0] + CurSubstreamBits[1] + underflowBits – (maxStuffSize << 1)) : 0	
CurSubstreamBitsMax = Max(CurSubstreamBits[0] + zeroBits[0], Max(CurSubstreamBits[1] + zeroBits[1], CurSubstreamBits[2] + zeroBits[2]))	
stuffing_data_substream0(zeroBits[0], CurSubstreamBits[0], CurSubstreamBitsMax)	
stuffing_data_substream1(zeroBits[1], CurSubstreamBits[1], CurSubstreamBitsMax)	
stuffing_data_substream2(zeroBits[2], CurSubstreamBits[2], CurSubstreamBitsMax)	
StuffBits = StuffBits[0] + StuffBits[1] + StuffBits[2]	
}	
}	
if (MultiplexingEnableFlag) {	
headerBits = 0	
for (ssIdx=0; ssIdx<image_format==‘001’?1:3; ssIdx++) {	

表 14（续）

编码单元定义	描述符
tmpSubstreamSegmentNum = (SubstreamBits[ssIdx] + SubstreamSegmentSize – 3) / (SubstreamSegmentSize – 2)	
headerBits += ((tmpSubstreamSegmentNum – SubstreamSegmentNum[ssIdx]) << 1)	
SubstreamSegmentNum[ssIdx] = tmpSubstreamSegmentNum	
}	
}	
if (CurrSlicePosY == 0 && CurrCuIdx < TransmissionDelayCu) {	
PhysicalBufferLevel = PhysicalBufferLevel + CuBits + headerBits + StuffBits	
} else {	
PhysicalBufferLevel = PhysicalBufferLevel + CuBits + headerBits + StuffBits – ((CuSize × TargetBpp) >> 4) – tmpSliceAlignAdj	
}	
if (CurrCuIdx < TransmissionDelayCu) {	
VirtualStartStuff = VirtualStartStuff – ((CuSize × TargetBpp) >> 4)	
}	
CurrRcBufferLevel = PhysicalBufferLevel + VirtualStartStuff + VirtualEndStuff	
if (x == SliceWidthInCu – 1 && y == SliceHeightInCu – 1) { /* Final CU in current Slice */	
stuffNum = SubstreamSegmentSize – 2	
if (MultiplexingEnableFlag) {	
if (SubstreamBits[0] % (SubstreamSegmentSize – 2) != 0)	
for(i=0; i<stuffNum–(SubstreamBits[0]%(SubstreamSegmentSize–2)); i++)	
substream0_alignment_bit0	f(1)
}	
}	
if (image_format != ‘000’) { /* Not YUV400 */	
if (SubstreamBits[1] % (SubstreamSegmentSize – 2) != 0)	
for(i=0; i<stuffNum–(SubstreamBits[1]%(SubstreamSegmentSize–2)); i++)	
substream1_alignment_bit0	f(1)
}	
}	
if (SubstreamBits[2] % (SubstreamSegmentSize – 2) != 0)	
for(i=0; i<stuffNum–(SubstreamBits[2]%(SubstreamSegmentSize–2)); i++)	
substream2_alignment_bit0	f(1)
}	
}	
} else {	
while (!byte_aligned( )) {	

表 14（续）

编码单元定义	描述符
slice_alignment_bit0	f(1)
}	
}	
}	
}	
CurrCuIdx++	
CurrPos = (CurrPos + 1) % 3	
}	
}	
}	
注：一种典型的实现方式，不同子流可以采用不同的熵解码器进行解析。	

7.1.4.2 第一子流编码块定义

第一子流编码块定义见表15。

表15 第一子流编码块定义

第一子流编码块定义	描述符
coding_block_data_substream0(cuIdxX, cuIdxY) {	
CbWidth[0] = CuWidth	
CbHeight[0] = CuHeight	
CbSize[0] = CbWidth[0] ×CbHeight[0]	
CbPosX[0] = cuIdxX × CbWidth[0]	
CbPosY[0] = cuIdxY × CbHeight[0]	
complexity_level_flag[0]	u(1)
if (!ComplexityLevelFlag[0]) {	
delta_level[0]	u(2)
if (delta_level[0] >= PrevComplexityLevel) {	
ComplexityLevel[0] = DeltaLevel[0] + 1	
} else {	
ComplexityLevel[0] = DeltaLevel[0]	
}	
} else {	
ComplexityLevel[0] = PrevComplexityLevel	
}	
PrevComplexityLevel = ComplexityLevel[0]	

表 15（续）

第一子流编码块定义	描述符
pred_mode[0]	ce(v)
if (PredMode[0] == ‘SAMPLE_MODE’ && Allowfallback) {	
FallbackFlag = 1	
if (IbcEnableFlag) {	
fallback_type	u(1)
}	
PredMode[0] == (FallbackType == 1 || !IbcEnableFlag) ? ‘SAMPLE_MODE’ : ‘IBC_MODE’	
}	
if (DpEnableFlag && PredMode[0] == ‘SAMPLE_MODE’ && !Allowfallback) {	
cu_dp_intra_flag	u(1)
if (CuDpIntraFlag) {	
for (index=0; index<(CbSize[0]>>3); index++) {	
cu_sdp_intra_flag[index]	u(1)
}	
}	
}	
if (PredMode[0] == ‘IBC_MODE’) {	
if (image_format == ‘000’ || !MultiplexingEnableFlag) { /* YUV400 or multiplexing disabled */	
for (blkIdx=0; blkIdx<8; blkIdx++) {	
abs_bvd_minus1[blkIdx]	u(5)
}	
if (DpEnableFlag && !FallbackFlag) {	
for (blkIdx=0; blkIdx<8; blkIdx++) {	
if (AbsBvDMinus1[blkIdx] == 31) {	
if (image_format != ‘000’) { /* Not YUV400 */	
sdp_ibc_offset[0][blkIdx]	ce(v)
sdp_ibc_offset[1][blkIdx]	ce(v)
sdp_ibc_offset[0][blkIdx]	ce(v)
} else {	
sdp_ibc_offset[0][blkIdx]	ce(v)
}	
}	
}	
}	
} else {	
for (blkIdx=0; blkIdx<5; blkIdx+=4) {	
abs_bvd_minus1[blkIdx]	u(5)

表 15（续）

第一子流编码块定义	描述符
}	
if (DpEnableFlag && !FallbackFlag) {	
for (blkIdx=0; blkIdx<5; blkIdx+=4) {	
if (AbsBvDMinus1[blkIdx] == 31) {	
sdp_ibc_offset[0][blkIdx]	ce(v)
sdp_ibc_offset[1][blkIdx]	ce(v)
sdp_ibc_offset[2][blkIdx]	ce(v)
}	
}	
}	
}	
} else if (PredMode[0] == 'INTRA_MODE_DC' || PredMode[0] == 'INTRA_MODE_ANG_0' || PredMode[0] == 'INTRA_MODE_ANG_1' || PredMode[0] == 'INTRA_MODE_ANG_2' || PredMode[0] == 'INTRA_MODE_ANG_3' || PredMode[0] == 'INTRA_MODE_ANG_4' || PredMode[0] == 'INTRA_MODE_ANG_5') {	
if (DpEnableFlag) {	
cu_dp_intra_flag	u(1)
}	
if (CuDpIntraFlag) {	
for (index=0; index<(CbSize[0]>>3); index++) {	
cu_sdp_intra_flag[index]	u(1)
}	
for (index=0; index<((CbSize[0]>>3); index++) {	
if (CuSdpIntraFlag[index]) {	
sdp_intra_offset[0][index]	ce(v)
}	
}	
} else {	
if (BrcEnableFlag) {	
brc_flag[0]	u(1)
}	
if (BwqEnableFlag && (!BrcFlag[0] || PredMode[0] == 'INTRA_MODE_ANG_1')) {	
bwq_flag[0]	u(1)
}	
if (BrcFlag[0]) {	
for (index=0; index<(CbWidth[0]>>2); index++) {	
brc_offset_flag[0][index]	u(1)
}	

表 15（续）

第一子流编码块定义	描述符
for (index=0; index<(CbWidth[0]>>2); index++) {	
if (BrcOffsetFlag[0][index] == 1) {	
brc_offset[0][index]	u(5)
}	
}	
}	
}	
}	
residual_block(0)	
}	

7.1.4.3 第一子流填充数据定义

第一子流填充数据定义见表16。

表16 第一子流填充数据定义

第一子流填充数据定义	描述符
stuffing_data_substream0(zeroBits, curSubstreamBits, curSubstreamBitsMax) {	
stuffBits = (curSubstreamBitsMax + SubstreamExpansionRatio − 1) / SubstreamExpansionRatio − curSubstreamBits	
StuffBits[0] = Max(zeroBits, stuffBits)	
for (bitIdx = 0; bitIdx<StuffBits[0]; bitIdx++) {	
block_substream_stuffing_bit0	f(1)
}	
}	
注：符合本文件位流在当前CU为回退模式时，StuffBits[0]的值应为0。	

7.1.4.4 第二子流编码块定义

第二子流编码块定义见表17。

表17 第二子流编码块定义

第二子流编码块定义	描述符
coding_block_data_substream1(cuIdxX, cuIdxY) {	
complexity_level_flag[1]	u(1)
if (!ComplexityLevelFlag[1]) {	
delta_level[1]	u(2)

表 17（续）

第二子流编码块定义	描述符
if (delta_level[1] >= ComplexityLevel[0]) {	
ComplexityLevel[1] = DeltaLevel[1] + 1	
} else {	
ComplexityLevel[1] = DeltaLevel[1]	
}	
} else {	
ComplexityLevel[1] = ComplexityLevel[0]	
}	
if (MultiplexingEnableFlag && FallbackFlag && (FallbackType == 1 || !IbcEnableFlag)) {	
if (SubstreamExpansionRatio >= 4 && SubstreamExpansionRatio < 8)	
reserved_bits	u(10)
} else if (SubstreamExpansionRatio >= 8 && SubstreamExpansionRatio < 16)	
reserved_bits	u(5)
} else if(SubstreamExpansionRatio >= 16)	
reserved_bits	u(2)
}	
} else if (PredMode[0] != ‘IBC_MODE’) {	
pred_mode[1]	ce(v)
} else if (MultiplexingEnableFlag) {	
for (blkIdx=1; blkIdx<4; blkIdx++) {	
abs_bvd_minus1[blkIdx]	u(5)
}	
if (DpEnableFlag && !FallbackFlag) {	
for (blkIdx=1; blkIdx<4; blkIdx++) {	
if (AbsBvDMinus1[blkIdx] == 31) {	
sdp_ibc_offset[0][blkIdx]	ce(v)
sdp_ibc_offset[1][blkIdx]	ce(v)
sdp_ibc_offset[2][blkIdx]	ce(v)
}	
}	
}	
}	
if (image_format == ‘001’) /* YUV420 */	
{	
CbWidth[1] = CuWidth >> 1	
CbHeight[1] = CuHeight >> 1	
} else if (image_format == ‘010’) { /* YUV422 */	

表 17（续）

第二子流编码块定义	描述符
CbWidth[1] = CuWidth >> 1	
CbHeight[1] = CuHeight	
} else if (image_format == ‘011’ || image_format == ‘100’) { /* YUV444 or RGB444 */	
CbWidth[1] = CuWidth	
CbHeight[1] = CuHeight	
}	
CbSize[1] = CbWidth[1] ×CbHeight[1]	
CbPosX[1] = cuIdxX × CbWidth[1]	
CbPosY[1] = cuIdxY × CbHeight[1]	
if (PredMode[1] == ‘INTRA_MODE_DC’ || PredMode[1] == ‘INTRA_MODE_ANG_0’ || PredMode[1] == ‘INTRA_MODE_ANG_1’ || PredMode[1] == ‘INTRA_MODE_ANG_2’ || PredMode[1] == ‘INTRA_MODE_ANG_3’ || PredMode[1] == ‘INTRA_MODE_ANG_4’ || PredMode[1] == ‘INTRA_MODE_ANG_5’) {	
if (CuDpIntraFlag) {	
for (index=0; index<(CbSize[0]>>3); index++) {	
if (CuSdpIntraFlag[index]) {	
sdp_intra_offset[1][index]	ce(v)
}	
}	
} else {	
if (BrcEnableFlag) {	
brc_flag[1]	u(1)
}	
if (BwqEnableFlag && (!BrcFlag[1] || PredMode[1] == ‘INTRA_MODE_ANG_1’)) {	
bwq_flag[1]	u(1)
}	
if (BrcFlag[1]) {	
for (index=0; index<(CbWidth[1]>>2); index++) {	
brc_offset_flag[1][index]	u(1)
}	
for (index=0; index<(CbWidth[1]>>2); index++) {	
if (BrcOffsetFlag[1][index] == 1) {	
brc_offset[1][index]	u(5)
}	
}	
}	
}	
}	
residual_block(1)	
}	

7.1.4.5 第二子流填充数据定义

第二子流填充数据定义见表18。

表18 第二子流填充数据定义

第二子流填充数据定义	描述符
stuffing_data_substream1(zeroBits, curSubstreamBits, curSubstreamBitsMax) {	
stuffBits = (curSubstreamBitsMax + SubstreamExpansionRatio – 1) / SubstreamExpansionRatio – curSubstreamBits	
StuffBits[1] = Max(zeroBits, stuffBits)	
for (bitIdx = 0; bitIdx<StuffBits[1]; bitIdx++) {	
block_substream_stuffing_bit0	f(1)
}	
}	
注：符合本文件位流在当前CU为回退模式时，StuffBits[1]的值应为0。	

7.1.4.6 第三子流编码块定义

第三子流编码块定义见表19。

表19 第三子流编码块定义

第三子流编码块定义	描述符
coding_block_data_substream2(cuIdxX, cuIdxY) {	
if (MultiplexingEnableFlag && FallbackFlag && (FallbackType == 1 || !IbcEnableFlag)) {	
if (SubstreamExpansionRatio >=4 && SubstreamExpansionRatio < 8)	
reserved_bits	u(11)
} else if (SubstreamExpansionRatio >= 8 && SubstreamExpansionRatio < 16)	
reserved_bits	u(6)
} else if (SubstreamExpansionRatio >= 16)	
reserved_bits	u(3)
}	
} else if (PredMode[0] != 'IBC_MODE') {	
pred_mode[2]	ce(v)
} else if (MultiplexingEnableFlag) {	
for (blkIdx=5; blkIdx<8; blkIdx++) {	
abs_bvd_minus1[blkIdx]	u(5)
}	
if (DpEnableFlag && !FallbackFlag) {	
for (blkIdx=5; blkIdx<8; blkIdx++) {	
if (AbsBvDMinus1[blkIdx] == 31) {	
sdp_ibc_offset[0][blkIdx]	ce(v)

表 19（续）

第三子流编码块定义	描述符
sdp_ibc_offset[1][blkIdx]	ce(v)
sdp_ibc_offset[2][blkIdx]	ce(v)
}	
}	
}	
}	
if (image_format == ‘001’) /* YUV420 */	
{	
CbWidth[2] = CuWidth >> 1	
CbHeight[2] = CuHeight >> 1	
} else if (image_format == ‘010’) { /* YUV422 */	
CbWidth[2] = CuWidth >> 1	
CbHeight[2] = CuHeight	
} else if (image_format == ‘011’ || image_format == ‘100’) { /* YUV444 or RGB444 */	
CbWidth[2] = CuWidth	
CbHeight[2] = CuHeight	
}	
CbSize[2] = CbWidth[2] ×CbHeight[2]	
CbPosX[2] = cuIdxX × CbWidth[2]	
CbPosY[2] = cuIdxY × CbHeight[2]	
if (PredMode[2] == ‘INTRA_MODE_DC’ || PredMode[2] == ‘INTRA_MODE_ANG_0’ || PredMode[2] == ‘INTRA_MODE_ANG_1’ || PredMode[2] == ‘INTRA_MODE_ANG_2’ || PredMode[2] == ‘INTRA_MODE_ANG_3’ || PredMode[2] == ‘INTRA_MODE_ANG_4’ || PredMode[2] == ‘INTRA_MODE_ANG_5’) {	
if (CuDpIntraFlag) {	
for (index=0; index<(CbSize[0]>>3); index++) {	
if (CuSdpIntraFlag[index]) {	
sdp_intra_offset[2][index]	ce(v)
}	
}	
} else {	
if (BrcEnableFlag) {	
brc_flag[2]	u(1)
}	
if (BwqEnableFlag && (!BrcFlag[2] || PredMode[2] == ‘INTRA_MODE_ANG_1’)) {	
bwq_flag[2]	u(1)
}	
if (BrcFlag[2]) {	

表 19（续）

第三子流编码块定义	描述符
for (index=0; index<(CbWidth[2]>>2); index++) {	
brc_offset_flag[2][index]	u(1)
}	
for (index=0; index<(CbWidth[2]>>2); index++) {	
if (BrcOffsetFlag[2][index] == 1) {	
brc_offset[2][index]	u(5)
}	
}	
}	
}	
}	
residual_block(2)	
}	

7.1.4.7 第三子流填充数据定义

第三子流填充数据定义见表20。

表20 第三子流填充数据定义

第三子流填充数据定义	描述符
stuffing_data_substream2(zeroBits, curSubstreamBits, curSubstreamBitsMax) {	
stuffBits = (curSubstreamBitsMax + SubstreamExpansionRatio – 1) / SubstreamExpansionRatio – curSubstreamBits	
StuffBits[2] = Max(zeroBits, stuffBits)	
for (bitIdx = 0; bitIdx<StuffBits[2]; bitIdx++) {	
block_substream_stuffing_bit0	f(1)
}	
}	
注：符合本文件位流在当前CU为回退模式时，StuffBits[2]的值应为0。	

7.1.5 残差块语法

7.1.5.1 残差块定义

残差块定义见表21。

表21　残差块定义

残差块定义	描述符
residual_block(component) {	
if (PredMode[component] == ‘SAMPLE_MODE’ || FallbackFlag) {	
for (x=0; x<CbHeight[component]; x++) {	
for (y=0; y<CbWidth[component]; y++) {	
sample_data	u(v)
if (FallbackFlag && FallbackType == 0) {	
ResiData[component][x][y] = SampleData	
} else {	
SampleValue[component][x][y] = SampleData << (BitDepth[component] – BitDepth[0])	
}	
}	
}	
} else {	
SetNum = CbWidth[component] < CuWidth ? 1 : 2	
ResiNum = CbHeight[component] == CuHeight ? 16 : 8	
GroupNum = CbHeight[component] == CuHeight ? 4 : 2	
for (i=0; i<SetNum; i++) {	
resi_group_type_index	ce(v)
GroupType[component][i] = ResiGroupTypeIndex	
}	
rb_rg_cl(component)	
rb_resi_residual(component)	
rb_rg_trailing(component)	
}	
}	

7.1.5.2　残差块残差组编码长度定义

残差块残差组编码长度定义见表22。

表22　残差块残差组编码长度定义

残差块残差组编码长度定义	描述符
rb_rg_cl(component) {	
if (image_format == ‘010’ && component != 0) {　/* YUV422 Chroma */	
if (GroupType[component][0] == 0) {	
GroupType[component][0] == 1	
} else {	

表 22（续）

残差块残差组编码长度定义	描述符
GroupType[component][0] == 4	
}	
}	
for (j=0; j<SetNum; j++) {	
if (GroupType[component][j] == 0) {	
groupNum = 1	
} else if ((GroupType[component][j] == 1) {	
groupNum = 2	
} else if (GroupType[component][j] == 2 || GroupType[component][j] == 3) {	
groupNum = 3	
} else {	
groupNum = 4	
}	
for (i=0; i<groupNum; i++) {	
rg_cl[component][i][j]	ce(v)
}	
if (ResiNum == 16) {	
if (GroupType[component][j] == 0) {	
SubRgCl[component][0][j] = RgCl[component][0][j]	
SubRgCl[component][1][j] = RgCl[component][0][j]	
SubRgCl[component][2][j] = RgCl[component][0][j]	
SubRgCl[component][3][j] = RgCl[component][0][j]	
} else if (GroupType[component][j] == 1) {	
SubRgCl[component][0][j] = RgCl[component][0][j]	
SubRgCl[component][1][j] = RgCl[component][0][j]	
SubRgCl[component][2][j] = RgCl[component][1][j]	
SubRgCl[component][3][j] = RgCl[component][1][j]	
} else if (GroupType[component][j] == 2) {	
SubRgCl[component][0][j] = RgCl[component][0][j]	
SubRgCl[component][1][j] = RgCl[component][0][j]	
SubRgCl[component][2][j] = RgCl[component][1][j]	
SubRgCl[component][3][j] = RgCl[component][2][j]	
} else if (GroupType[component][j] == 3) {	
SubRgCl[component][0][j] = RgCl[component][0][j]	
SubRgCl[component][1][j] = RgCl[component][1][j]	
SubRgCl[component][2][j] = RgCl[component][2][j]	
SubRgCl[component][3][j] = RgCl[component][2][j]	

表 22（续）

残差块残差组编码长度定义	描述符
} else if (GroupType[component][j] == 4) {	
SubRgCl[component][0][j] = RgCl[component][0][j]	
SubRgCl[component][1][j] = RgCl[component][1][j]	
SubRgCl[component][2][j] = RgCl[component][2][j]	
SubRgCl[component][3][j] = RgCl[component][3][j]	
}	
} else {	
if (GroupType[component][j] == 0) {	
SubRgCl[component][0][j] = RgCl[component][0][j]	
SubRgCl[component][1][j] = RgCl[component][0][j]	
} else if (GroupType[component][j] == 1) {	
SubRgCl[component][0][j] = RgCl[component][0][j]	
SubRgCl[component][1][j] = RgCl[component][1][j]	
}	
}	
}	
}	

7.1.5.3 残差块残差数据定义

残差块残差数据定义见表23。

表23 残差块残差数据定义

残差块残差数据定义	描述符
rb_resi_residual(component) {	
for (j=0; j<SetNum; j++) {	
for (i=0; i<ResiNum; i++) {	
idx = i >> 2	
clValue = SubRgCl[component][idx][j]	
if (clValue == 0) {	
ScanResiData[component][i][j] = 0	
} else {	
rg_resi_data	u(v)
if ((BitDepth[component] != clValue || PredMode[component] == 'POINT_MODE') && RgResiData > (1 << (clValue – 1))) {	
ScanResiData[component][i][j] = RgResiData – (1 << clValue)	
} else {	

表 23（续）

残差块残差数据定义	描述符
ScanResiData[component][i][j] = RgResiData	
}	
}	
}	
}	
}	
}	

7.1.5.4 残差块残差组边界值标记定义

残差块残差组边界值标记定义见表24。

表24 残差块残差组边界值标记定义

残差块残差组边界值标记定义	描述符
rb_rg_trailing(component) {	
for (j=0; j<SetNum; j++) {	
for (i=0; i<ResiNum; i=i+4) {	
index = i >> 2	
if (SubRgCl[component][index][j] <= 2 && SubRgCl[component][index][j] != 0) {	
rg_trailing_flag	u(1)
for (subIdx=i; subIdx<(i+4); subIdx++) {	
if ((RgTrailingFlag && ScanResiData[component][subIdx][j] == (1 << (SubRgCl[component][index][j] – 1))) {	
ScanResiData[component][subIdx][j] = – ScanResiData[component][subIdx][j]	
}	
}	
}	
}	
}	
}	

7.2 语义描述

7.2.1 图像

7.2.1.1 图像头

档次标号 profile_id

8 位无符号整数。表示位流符合的档次。

档次和级别见附录B。

输入图像宽度　input_image_width

32位无符号整数。规定输入图像亮度分量的宽度，即水平方向样本数。input_image_width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。InputImageWidth的值等于input_image_width的值，ImageWidth的值等于((InputImageWidth+SliceWidth-1)/SliceWidth)×SliceWidth的值，ImageWidth的值不应为0。

输入图像高度　input_image_height

32位无符号整数。规定输入图像亮度分量的高度，即竖直方向扫描行数。input_image_height的单位应是图像样本的行数,InputImageHeight的值等于input_image_height的值，ImageHeight的值等于((InputImageHeight + SliceHeight – 1) / SliceHeight) × SliceHeight的值，ImageHeight的值不应为0。

ImageWidth、ImageHeight与图像边界的关系见图8。图8中，实线表示图像可显示区域边界，其宽度和高度分别由InputImageWidth和InputImageHeight决定；虚线表示图像边界，其宽度和高度分别由ImageWidth和ImageHeight决定。

注：图8只是给出了输入图像和解码图像的尺寸关系，一种可行的图像填补方案见附录E。

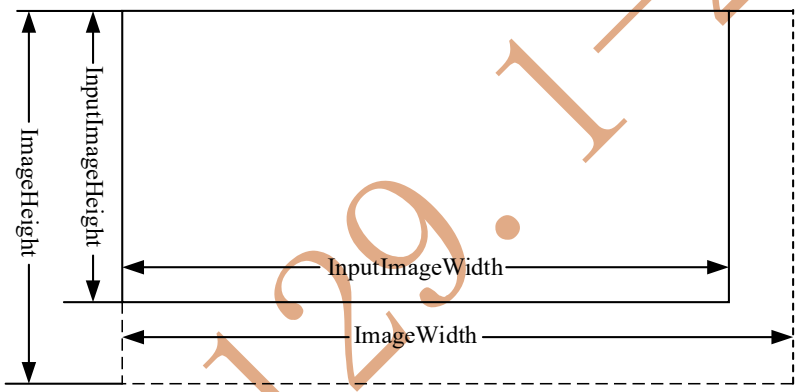


图8　图像边界示意图

条带宽度　slice_width

条带高度　slice_height

32 位无符号整数。规定条带的宽度和高度。SliceWidth 的值等于 slice_width 的值。SliceHeight 的值等于 slice_height 的值,slice_width 的值应为 CuWidth 的倍数,slice_height 的值应为 CuHeight 的倍数。如果 MultiplexingEnableFlag 的值为 0，则 TotalBits 的值等于((((SliceWidth×SliceHeight×TargetBpp)>>4)+7)>>3)<<3；否则，TotalBits 的值等于(((((SliceWidth×TargetBpp)>>4)+7)>>3)<<3)×SliceHeight。SliceInfo 数组存储了当前图像的条带信息，其中 SliceInfo[i][j]表示当前图像水平索引为 i，竖直索引为 j 的条带信息。SliceInfo[i][j]为一结构体，其中每个变量包括 posX 和 posY 两个参数，SliceInfo[i][j]->posX 表示当前图像水平索引为 i，竖直索引为 j 的条带的左上角位置在图像中的横坐标位置，SliceInfo[i][j]->posY 表示当前图像水平索引为 i，竖直索引为 j 的条带的左上角位置在图像中的纵坐标位置。

PictureWidthInSlice = (ImageWidth + SliceWidth – 1) / SliceWidth

PictureHeightInSlice = (ImageHeight + SliceHeight – 1) / SliceHeight

SliceWidthInCu = SliceWidth / CuWidth

SliceHeightInCu = SliceHeight / CuHeight

```
for (i=0; i<PictureWidthInSlice; i++) {
    for (j=0; j< PictureHeightInSlice; j++) {
        SliceInfo[i][j]->posX = SliceWidth × i
        SliceInfo[i][j]->posY = SliceHeight × j
    }
}
```

符合本文件的位流应满足以下要求：

```
SliceWidthInCu × SliceHeightInCu >= Max(5, Max(EndControlBlocks + 1, TransmissionDelayCu))
```

图像格式　image_format

3位无符号整数。规定图像的颜色空间和采样格式格式，见表25。ImageFormat的值等于image_format的值。对于图像格式为RGB444的输入图像，编码侧需要转换为YCoCg后进行编码，解码侧需要将输出的YCoCg转换回RGB444图像格式，转换方式见附录A。

表25　图像格式

image_format的值	图像格式	含义
000	YUV400	颜色空间YCbCr，采样格式4:0:0
001	YUV420	颜色空间YCbCr，采样格式4:2:0
010	YUV422	颜色空间YCbCr，采样格式4:2:2
011	YUV444	颜色空间YCbCr，采样格式4:4:4
100	RGB444	颜色空间RGB，采样格式4:4:4
101～111	-	保留

比特深度　bit_depth

4位无符号整数。规定图像的比特深度，BitDepth[0]的值等于bit_depth的值，如果image_format的值为‘100’且BitDepth[0]不等于16，或者，image_format的值为‘100’且BitDepth[0]等于16且LosslessEnableFlag的值等于1，则BitDepth[1]的值等于bit_depth的值加1，BitDepth[2]的值等于bit_depth的值加1；否则，BitDepth[1]的值等于bit_depth的值，BitDepth[2]的值等于bit_depth的值。

目标bpp　target_bpp

16位无符号整数。确定目标bpp，精度为1/16，低位4bit为分数部分。TargetBpp的值等于target_bpp的值。TargetBpp的值应大于(2<<4)，并且TargetBpp的值应小于(colorFormatWeight × BitDepth[0])<<4，其中如果ImageFormat为‘000’，colorFormatWeight的值等于1；否则，如果ImageFormat为‘001’，colorFormatWeight的值等于1.5；否则，如果ImageFormat为‘010’，colorFormatWeight的值等于2；否则，colorFormatWeight的值等于3。

图像填补方式标志位　padding_type_flag

二值变量。值为‘1’表示采用图像填补方式一，值为‘0’表示采用图像填补方式二，两种图像填补方式见附　录　E。PaddingTypeFlag的值等于padding_type_flag的值。

客观无损使能标志位　lossless_enable_flag

二值变量。值为‘1’表示应启用客观无损编码模式，值为‘0’表示不应使用客观无损编码模式。LosslessEnableFlag的值等于lossless_enable_flag的值。符合本文件的位流应满足如果LosslessEnableFlag的值等于1，则PwqEnableFlag、BwqEnableFlag、BrcEnableFlag和DpEnableFlag的值均为0且VbrEnableFlag的值为1。

变换使能标志位　transform_enable_flag

二值变量。值为‘1’表示启用变换技术，值为‘0’表示不应使用变换技术。

差值预测使能标志位　dp_enable_flag

二值变量。值为‘1’表示启用差值预测技术，值为‘0’表示不应使用差值预测技术。DpEnableFlag的值等于dp_enable_flag的值。

子块重建补偿使能标志位　brc_enable_flag

二值变量。值为‘1’表示启用子块重建补偿技术，值为‘0’表示不应使用子块重建补偿技术。BrcEnableFlag的值等于brc_enable_flag的值。

块复制帧内预测使能标志位　ibc_enable_flag

二值变量。值为‘1’表示启用块复制帧内预测模式，值为‘0’表示不应使用块复制帧内预测模式。IbcEnableFlag的值等于ibc_enable_flag的值。

点级量化调整使能标志位　pwq_enable_flag

二值变量。值为‘1’表示启用点级量化调整技术，值为‘0’表示不应使用点级量化调整技术。PwqEnableFlag的值等于pwq_enable_flag的值。

子块级量化参数调整标志位　bwq_enable_flag

二值变量。值为‘1’表示启用子块重建补偿技术，值为‘0’表示不应使用子块重建补偿技术。BwqEnableFlag的值等于bwq_enable_flag的值。

变码率编码模式使能标志位　vbr_enable_flag

二值变量。值为‘1’表示启用变码率编码模式，值为‘0’表示不应使用变码率编码模式。VbrEnableFlag的值等于vbr_enable_flag的值。

点预测量化参数调整最大阈值　pwq_max_qp

4位无符号整数。规定点预测模式量化参数调整的最大量化参数。PwqMaxQp的值等于pwq_max_qp的值,PwqMaxQp的值应小于等于8。

子块量化参数调整复杂度阈值　bwq_complex_th

3位无符号整数。规定子块量化参数调整复杂度阈值。BwqComplexTh的值等于bwq_complex_th的值。BwqComplexTh的值应大于等于0且小于等于4。

量化参数调整第0梯度阈值　qp_refine_th0

4位无符号整数。规定量化参数调整的第0梯度阈值。QpRefineTh0的值等于qp_refine_th0的值。

量化参数调整第1梯度阈值　qp_refine_th1

4位无符号整数。规定量化参数调整的第1梯度阈值。QpRefineTh1的值等于qp_refine_th1的值,QpRefineTh1的值应大于等于QpRefineTh0的值。

可察觉失真量化参阈值　jnd_qp

4位无符号整数。规定可察觉失真对应量化参数的阈值。JndQp的值等于jnd_qp的值,JndQp的值应小于等于8。

严格可察觉失真量化参数阈值　strict_jnd_qp

4位无符号整数。规定严格可察觉失真对应量化参数的阈值。StrictJndQp的值等于strict_jnd_qp的值,StrictJndQp的值应小于等于JndQp的值。

位流段编码单元数目　chunk_size_in_cu

32位无符号整数。规定位流段的编码单元数目。ChunkSizeInCu的值等于chunk_size_in_cu的值,ChunkSizeInCu 的值应大于等于4。位流段尺寸ChunkSizeInBit的值等于((CuSize×TargetBpp)>>4)×ChunkSizeInCu，ChunkSizeInBit应为8的整数倍且应小于TotalBits。

位流段数目　chunk_num

32位无符号整数。规定每个条带中位流段的数目。ChunkNum的值等于chunk_num的值,ChunkNum的值应等于(TotalBits+ChunkSizeInBit−1)/ChunkSizeInBit。

条带编码单元数量的最高位比特位　slice_cu_num_max_bit

8位无符号整数。规定一个条带中编码单元总数量的二进制最高位数。SliceCuNumMaxBit的值等于slice_cu_num_max_bit的值。

填补区域比特填充使能标志位　padding_stuff_flag

二值变量。值为‘1’表示启用填补区域比特填充技术，值为‘0’表示不应使用填补区域比特填充技术。PaddingStuffFlag的值等于padding_stuff_flag的值。

子流膨胀比例的对数值　substream_expansion_ratio_log2

3位无符号整数。规定子流膨胀比例上限的对数值。SubstreamExpansionRatio的值等于1<<substream_expansion_ratio_log2的值，SubstreamExpansionRatio的值应大于等于4。

子流片大小　substream_segment_size

10位无符号整数。规定子流片的二进制位数量。SubstreamSegmentSize的值等于substream_segment_size的值，如果BitDepth[0]小于等于10，则SubstreamSegmentSize的值等于336，否则，SubstreamSegmentSize的值等于CuSize×BitDepth[0]+16。如果MultiplexingEnableFlag等于1，则符合本标准的位流在完成每个编码单元的解析后，CurSubstreamBits[0]，CurSubstreamBits[1]以及CurSubstreamBits[2]的值应不大于（SubstreamSegmentSize-2）。

初始传输延迟编码单元个数　transmission_delay_cu

16位无符号整数。规定初始传输延迟的CU个数。TransmissionDelayCu的值等于transmission_delay_cu的值，DelayBits的值等于(TransmissionDelayCu × ((CuSize × TargetBpp)>>4))。DelayBits不应小于(image_format==‘000’?1:3) × (SubstreamSegmentSize-1)+(1<<RcDecreaseStepLog2)-1+ModeBits+(BitDepth[0]×CbLumaSize+BitDepth[MultiplexingEnableFlag?0:1]×CbChromaSize×2。DelayBits应小于RcBufferSize。

码流缓存大小　rc_buffer_size

16位无符号整数。规定码流缓存大小，见附　录　C。RcBufferSize的值等于rc_buffer_size的值。符合本标准的位流应保证在解析完一个编码单元的语法或在解析完一个编码单元的子流编码单元定义以及子流数据填充定义后，CurrRcBufferLevel小于等于MaxBufferSize。符合本标准的位流应保证，如果AllowFallback等于0，则编码单元任一编码块用于coding_block_data_substream解析的位流大小小于等于SampleCost[component]其中，SampleCost[component]的计算方式为：

```
SampleCost[0] = BitDepth[0] × CbSize[0] + 5 + 3 + (DpEnableFlag ? 5 : 0)   /* Including pred_mode and maximal complexity cost */
if (image_format == ‘001’) {   /* YUV420 or YUV422 */
    SampleCost[1] = BitDepth[1] × CbSize[1] + 2 + 3
    SampleCost[2] = BitDepth[2] × CbSize[2] + 2
} else if (image_format == ‘010’) {   /* YUV422 */
    SampleCost[1] = BitDepth[1] × CbSize[1] + 2 + 3
    SampleCost[2] = BitDepth[2] × CbSize[2] + 2
} else if (image_format == ‘011’ || image_format == ‘100’) {   /* YUV444 or RGB444 */
    SampleCost[1] = BitDepth[MultiplexingEnableFlag ? 0 : 1] × CbSize[1] + 5 + 3
    SampleCost[2] = BitDepth[MultiplexingEnableFlag ? 0 : 2] × CbSize[2] + 5
}
```

由于算法模型与硬件模块存在差别，硬件实际真实开辟的码流缓冲区大小应略大于RcBufferSize + ExtraBufferSize。附录D提供了一种可行的编码端方案。

码控量化参数偏移量　rc_qp_bias[compL][compC]

4 位无符号整数。规定亮度复杂度为 compL，色度复杂度为 compC 的码控量化参数偏移量。

RcQpBias[compL][compC]的值等于rc_qp_bias[compL][compC]的值。

码控结尾下降步长对数值 rc_decrease_step_log2

8位无符号整数。规定码率控制模块使用的虚拟填充比特在slice结尾每个编码单元增加的步长的对数值。RcDecreaseStepLog2的值等于rc_decrease_step_log2的值。(1<<RcDecreaseStepLog2)的值应小于(((CuSize×TargetBpp)>>4)-(CuSize<<1))。

码控满度计算乘子 rc_fullness_calc_multiplier

8位无符号整数。规定计算满度使用的乘子。RcFullnessCalcMultiplier的值等于rc_fullness_calc_multiplier的值。

码控满度计算移位 rc_fullness_calc_shift

5位无符号整数。规定计算满度使用的移位量。RcFullnessCalcShift的值等于rc_fullness_calc_shift的值。

码控额外缓冲区惩罚阈值下降步长对数值 rc_extra_buffer_decrease_step_log2

8位无符号整数。规定码率控制模块使用的ExtraBufferThreshold在slice结尾每个编码单元降低的步长的对数值。RcExtraBufferDecreaseStepLog2的值等于rc_extra_buffer_decrease_step_log2的值。

码控额外缓冲区惩罚力度 rc_extra_buffer_penalty_log2_minus3

8位无符号整数。规定码率控制模块使用额外缓冲区的惩罚力度。ExtraBufferPenaltyLog2的值等于rc_extra_buffer_penalty_log2_minus3+3的值。

码控条带结尾处目标满度比例 rc_target_end_ratio_minus4

2位无符号整数。规定码控条带结尾处目标满度比例。RcTargetEndRatio的值等于rc_target_end_ratio_minus4+4的值。

码控斜率参数0 rc_ratio0

8位无符号整数。规定码控斜率参数0。RcRatio0的值等于rc_ratio0的值。

码控参数0 rc_param0

11位无符号整数。由码控充足时的满度阈值确定。RcParam0h的值等于rc_param0的值。

码控斜率参数1 rc_ratio1

9位无符号整数。规定码控斜率参数1。RcRatio1的值等于rc_ratio1的值。

码控参数1 rc_param1

11位无符号整数。由码控紧张时的满度阈值确定。RcParam1的值等于rc_param1的值。

码控斜率参数2 rc_ratio2

11位无符号整数。规定码控斜率参数2。RcRatio2的值等于rc_ratio2的值。

码控参数2 rc_param2

11位无符号整数。规定码控参数2。RcParam2的值等于rc_param2的值。

码控最大相对无损编码比特数 rc_max_relative_bits

8位无符号整数。规定码控最大相对无损编码比特数。RcMaxRelativeBits的值等于rc_max_relative_bits<<1的值。

码控第comp复杂度无损编码比特数 rc_lossless_bits[comp]

11位无符号整数。规定码控第comp复杂度无损编码比特数。RcLosslessBits[comp]的值等于rc_lossless_bits[comp]的值。

码控平均无损编码比特数 rc_avg_lossless_bits

11位无符号整数。规定码控初始平均无损编码比特数。RcAvgLosslessBits的值等于rc_abg_lossless_bits的值。

码控比特数偏移 rc_bits_offset

11位无符号整数。规定码控比特数偏移。RcBitsOffset的值等于rc_bits_offset的值。

码控最大无损编码比特数　rc_max_lossless_bits

11 位无符号整数。规定码控最大无损编码比特数。RcMaxLosslessBits 的值等于 rc_max_lossless_bits 的值。

码控复杂度等级阈值　rc_complex_th

3 位无符号整数。规定进行保护的复杂度阈值。RcComplexTh 的值等于 rc_complex_th 的值。

码控相对无损编码码率阈值　rc_relative_th

10 位无符号整数。规定进行保护的相对无损编码码率的阈值。RcRelativeTh 的值等于 rc_relative_th 的值。

7.2.2　条带

条带对齐填充位　slice_alignment_bit0

填充位。值应为‘0’。

7.2.3　子流片

子流索引　substream_index

2 个二进制位。值为‘00’代表当前子流片属于第一子流，值为‘01’代表当前子流片属于第二子流，值为‘10’代表当前子流片属于第三子流。SubstreamIndex 的值等于 substream_index 转换为 2 位无符号整数的值。SubstreamIndex 的值不应出现 3。如果 image_format 等于‘000’，SubstreamIndex 的值不应出现 1，2，3。

子流片数据　substream_segment_data

(SubstreamSegmentSize-2)个二进制位。相同 SubstreamIndex 值的子流片数据合并成对应的子流数据（见 8.1　）。

第一子流子流片对齐填充位　substream0_alignment_bit0

填充位。值应为‘0’。

第二子流子流片对齐填充位　substream1_alignment_bit0

填充位。值应为‘0’。

第三子流子流片对齐填充位　substream2_alignment_bit0

填充位。值应为‘0’。

7.2.4　编码单元

亮度复杂度等级更新标志　complexity_level_flag[0]

二值变量。值为‘1’表示当前编码单元的亮度块不需要更新复杂度等级；值为‘0’表示当前编码单元的亮度块需要更新复杂度等级。ComplexityLevelFlag[0]的值等于complexity_level_flag[0]的值。

色度复杂度等级更新标志　complexity_level_flag[1]

二值变量。值为‘1’表示当前编码单元的色度块的复杂度等级与亮度块的复杂度等级一致；值为‘0’表示当前编码单元的色度块的复杂度等级与亮度块的复杂度等级不一致。ComplexityLevelFlag[1]的值等于complexity_level_flag[1]的值。

亮度复杂度等级变化量　delta_level[0]

2位无符号整数。确定亮度复杂度等级的变化量。DeltaLevel[0]的值等于delta_level[0]的值。如果位流中不存在delta_level[0]，DeltaLevel[0]的值等于0。

色度复杂度等级变化量　delta_level[1]

2位无符号整数。确定色度复杂度等级的变化量。DeltaLevel[1]的值等于delta_level[1]的值。如果位流中不存在delta_level[1]，DeltaLevel[1]的值等于0。

第0分量预测模式索引　pred_mode[0]

第1分量预测模式索引　pred_mode[1]

第2分量预测模式索引　pred_mode[2]

当前编码块的预测模式，解析过程见8.2　。如果当前编码块宽为16高为2，预测模式与预测模式号的关系见表26　，PredMode[component]的值等于pred_mode[component]的值；否则，预测模式与预测模式号的关系见表27　，PredMode[component]的值等于pred_mode[component]的值。如果位流中不存在pred_mode[1]和pred_mode[2]且当前编码块宽为16高为2，PredMode[1]和PredMode[2]的值等于PredMode[0]；如果位流中不存在pred_mode[1]和pred_mode[2]且当前编码块宽为8高为n，其中n等于1或2，则PredMode[1]和PredMode[2]等于‘IBC_MODE’。如果IbcEnableFlag的值等于0，则pred_mode[component]不应等于‘IBC_MODE’。

表26　宽为 16 高为 2 的编码块预测模式

pred_mode[component]的值	预测模式	含义
0	POINT_MODE	点预测模式
1	INTRA_MODE_DC	普通帧内预测模式：DC模式
2	INTRA_MODE_ANG_0	普通帧内预测模式：角度预测模式0
3	INTRA_MODE_ANG_1	普通帧内预测模式：角度预测模式1
4	INTRA_MODE_ANG_2	普通帧内预测模式：角度预测模式2
5	INTRA_MODE_ANG_3	普通帧内预测模式：角度预测模式3
6	SAMPLE_MODE	样本值模式
7	INTRA_MODE_ANG_4	普通帧内预测模式：角度预测模式4
8	INTRA_MODE_ANG_5	普通帧内预测模式：角度预测模式5
9	IBC_MODE	块复制帧内预测模式

表27　宽为 8 高为 n 的编码块预测模式

pred_mode[component]的值	预测模式	含义
0	SAMPLE_MODE	样本值模式
1	POINT_MODE	点预测模式
2	INTRA_MODE_DC	普通帧内预测模式：DC模式
3	IBC_MODE	块复制帧内预测模式

回退模式类型　fallback_type

二值变量。值为‘1’表示当前编码单元采用的回退模式为样本值截断模式，值为‘0’表示当前编码单元采用的回退模式为残差截断模式。FallbackType的值等于fallback_type的值。

块矢量绝对值偏移值　abs_bvd_minus1[blkIdx]

5位无符号整数。用于确定块复制帧内预测模式第blkIdx个子块的矢量绝对值偏移值，取值范围0～31。AbsBvDMinus1[blkIdx]的值等于abs_bvd_minus1[blkIdx]的值。

第0分量子块重建补偿标志　brc_flag[0]
第1分量子块重建补偿标志　brc_flag[1]
第2分量子块重建补偿标志　brc_flag[2]

二值变量。值为‘1’表示当前编码块需要进行子块重建补偿；值为‘0’表示当前编码块不需要进行子块重建补偿。BrcFlag[component]的值等于brc_flag[component]的值。如果位流中不存在brc_flag[component]，BrcFlag[component]的值等于0。

第0分量子块量化参数调整标志　bwq_flag[0]
第1分量子块量化参数调整标志　bwq_flag[1]
第2分量子块量化参数调整标志　bwq_flag[2]

二值变量。值为‘1’表示当前编码块需要进行子块量化参数调整；值为‘0’表示当前编码块不需要进行子块量化参数调整。BwqFlag[component]的值等于bwq_flag[component]的值。如果位流中不存在bwq_flag[component]，如果CuDpIntraFlag的值等于1且BwqEnableFlag的值等于1，则BwqFlag[component]的值等于1；否则，BwqFlag[component]的值等于0。

第0分量子块重建补偿值非零标志　brc_offset_flag[0][index]
第1分量子块重建补偿值非零标志　brc_offset_flag[1][index]
第2分量子块重建补偿值非零标志　brc_offset_flag[2][index]

二值变量。值为‘1’表示重建补偿值非零；值为‘0’表示重建补偿值为零。BrcOffsetFlag[component][index]的值等于brc_offset_flag[component][index]的值。如果位流中不存在brc_offset_flag[component]，BrcOffsetFlag[component]的值等于0。

第0分量子块重建补偿值　brc_offset[0][index]
第1分量子块重建补偿值　brc_offset[1][index]
第2分量子块重建补偿值　brc_offset[2][index]

5位无符号整数。用于确定子块重建补偿值。BrcOffset[component][index]的值等于brc_offset[component][index]-16<0?brc_offset[component][index]-16:brc_offset[component][index]-15。如果位流中不存在brc_offset[component],BrcOffset[component]的值等于0。

编码单元普通帧内预测模式差值预测标志　cu_dp_intra_flag

二值变量。值为‘1’表示当前编码单元需要进行差值预测；值为‘0’表示当前编码单元不需要进行差值预测。CuDpIntraFlag的值等于cu_dp_intra_flag的值。如果位流中不存在cu_dp_intra_flag，CuDpIntraFlag的值等于0。

编码单元子块差值预测标志　cu_sdp_intra_flag[index]

二值变量。值为‘1’表示当前编码单元的第index个子块需要进行差值预测；值为‘0’表示当前编码单元第index个子块不需要进行差值预测。CuSdpIntraFlag[index]的值等于cu_sdp_intra_flag[index]的值。如果位流中不存在cu_sdp_intra_flag[index]，CuSdpIntraFlag[index]的值等于0。

普通帧内预测模式第0分量子块差值预测值　sdp_intra_offset[0][index]
普通帧内预测模式第1分量子块差值预测值　sdp_intra_offset[1][index]
普通帧内预测模式第2分量子块差值预测值　sdp_intra_offset[2][index]

普通帧内预测模式编码块第index个子块进行差值预测的预测值，解析过程见8.2　。SdpOffset[component][index]的值等于sdp_offset[component][index]。如果位流中不存在sdp_offset[0]，SdpOffset[0]的值等于0。

块复制帧内预测模式第0分量子块差值预测值　sdp_ibc_offset[0][index]
块复制帧内预测模式第1分量子块差值预测值　sdp_ibc_offset[1][index]
块复制帧内预测模式第2分量子块差值预测值　sdp_ibc_offset[2][index]

块复制帧内预测模式编码块第index个子块进行差值预测的预测值，解析过程见8.2。SdpIbcOffset[0][index]的值等于sdp_ibc_offset[0][index]。如果位流中不存在sdp_ibc_offset[0]，SdpIbcOffset[0]的值等于0。

编码单元子流填充位　block_substream_stuffing_bit0

填充位。值应为‘0’。

7.2.5　残差块

样本数据　sample_data

样本值或回退模式残差数据，解析过程见8.2。SampleData的值等于sample_data的值。

残差分组类型索引　resi_group_type_index

残差分组类型索引，解析过程见8.2。ResiGroupTypeIndex的值等于resi_group_type_index的值

残差组编码长度　rg_cl[component][i][j]

残差组编码长度，用于确定第component个分量第j个残差集合第i个残差组内残差的码字长度，解析过程见8.2。RgCL[component][i][j]的值等于rg_cl[component][i][j]的值。

残差组残差数据　rg_resi_data

用于确定残差组残差，解析过程见8.2。RgResiData的值等于rg_resi_data的值。

残差组边界值标志　rg_trailing_flag

二值变量。残差组边界值标志，用于确定残差组内边界残差的符号位。值为‘0’表示是边界残差符号为正数；值为‘1’表示边界残差符号为负数。RgTrailingFlag的值等于rg_trailing_flag的值。

8　解析过程

8.1　子流的获取

8.1.1　概述

从位流中依次解析得到图像头二元符号串，见8.2.2。根据二元符号串得到ce(v)描述的语法元素的值，见8.2.3。如果MultiplexingEnableFlag等于1，则进行子流解交织，见8.1.2，得到一个或三个子流；否则，直接进行解析。

如果MultiplexingEnableFlag等于1且在解析子流的过程中，描述符为ce(v),f(n)和u(n)的语法元素在解析完成后SubstreamBits[ssIdx]增加该语法元素的二元符号串的长度。其中，ssIdx表示当前解析的是第(ssIdx+1)子流。

注：图像头不属于任何子流，为所有子流的共用信息。

8.1.2　子流解交织

如图9所示，将子流索引相同的子流片的对应子流片数据根据先后顺序进行合并，得到一个或三个子流。

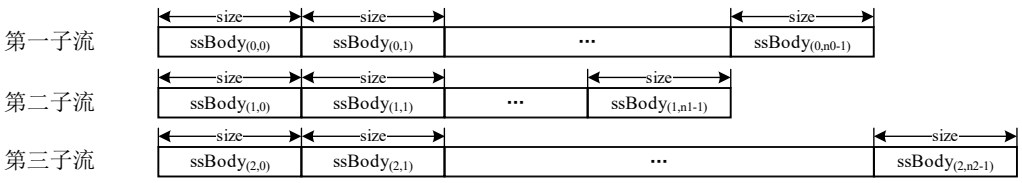


图9　子流组成示意图（位流顺序为从左到右）

其中，$ssbody_{(i,j)}$表示隶属于第(i+1)个子流的第j个子流片的子流数据主体，n0，n1以及n2分别为第一子流，第二子流以及第三子流的子流片数量，三者之和为n。

经过子流重组后的位流，需满足以下两个必要条件才可进行二元符号串的解析：

——在对某个编码块对应的所有二元符号串进行解析之前，包含这些二元符号串的所有子流片，均已包含在子流重组得到的子流中；

——在对某一编码单元色度编码块对应的所有二元符号串进行解析之前，需要先解析得到该编码单元的亮度编码块的预测模式。

经过子流重组后的位流，在进行解析前应存储于解码器的解码子流缓冲区中。解码子流缓冲区共设置三个，其中第一解码子流缓冲区存储子流重组后第一子流对应的位流，第二解码子流缓冲区存储子流重组后第二子流对应的位流，第三解码子流缓冲区存储子流重组后第三子流对应的位流。

单个解码子流缓冲区的大小，应至少为(2×SubstreamSegmentSize)比特，最大为(4×SubstreamSegmentSize)比特。附录F提供了一种可行的编码端的子流交织方案。

8.2 ce(v)和u(v)的解析过程

8.2.1 概述

从位流中解析得到二元符号串，见8.2.2。最后根据二元符号串得到ce(v)和u(v)描述的语法元素的值，见8.2.3。

8.2.2 二元符号串解析

8.2.2.1 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：

c) 解析当前二元符号，见 8.2.2.2；

d) 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；

e) 将由步骤 d 得到的二元符号串与 8.2.3 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。

8.2.2.2 二元符号解析过程

二元符号的解析过程为：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’。

8.2.3 反二值化方法

8.2.3.1 概述

本条定义语法元素的反二值化方法，见表28。

表28　语法元素的反二值化方法

语法元素	反二值化方法
pred_mode[component]	见 8.2.3.3
sdp_ibc_offset[component][index]	见 8.2.3.4
sdp_intra_offset[component][index]	见 8.2.3.5
resi_group_type_index	见 8.2.3.6
rg_cl[component][i][j]	见 8.2.3.7
rg_resi_data	见 8.2.3.8
sample_data	见 8.2.3.9

8.2.3.2　无符号定长码的反二值化方法

由二元符号串根据表29得到synElVal的值。

表29　synElVal 与二元符号串的关系（长度为 len 的定长码）

synElVal 的值	二元符号串				
0	0	0	…	0	0
1	0	0	…	0	1
2	0	0	…	1	0
3	0	0	…	1	1
…			…		
$2^{len}-4$	1	1	…	0	0
$2^{len}-3$	1	1	…	1	0
$2^{len}-2$	1	1	…	1	0
$2^{len}-1$	1	1	…	1	1
binIndex	0	1	…	len-2	len-1

8.2.3.3　pred_mode[component]的反二值化方法

如果当前编码块宽为16高为2时，由二元符号串根据表30得到pred_mode[component]的值，否则，根据8.2.3.2，由二元符号串根据表31得到pred_mode[component]的值。

表30　pred_mode[component]与二元符号串的关系 1

pred_mode[component]	二元符号串					
0	0	0				
1	1	1				
2	1	0	1	1		
3	1	0	1	0		

表 30（续）

pred_mode[component]	二元符号串					
4	1	0	0	0	0	1
5	1	0	0	0	0	0
6	1	0	0	0	1	
7	1	0	0	1	1	
8	1	0	0	1	0	
9	0	1				
binIdx	0	1	2	3	4	5

表31　pred_mode[component]与二元符号串的关系 2

pred_mode[component]	二元符号串	
0	0	0
1	0	1
2	1	
binIdx	0	1

8.2.3.4　sdp_ibc_offset[component][index]的反二值化方法

sdp_ibc_offset[component][index]的值等于synElVal的值，见8.2.3.2，其中定长码码长len的值等于BitDepth[component]-Max(BitDepth[0]-10,0)。

8.2.3.5　sdp_intra_offset[component][index]的反二值化方法

sdp_intra_offset[component][index]的值等于synElVal的值，见8.2.3.2，其中定长码码长len的值等于BitDepth[component]-Max(BitDepth[0]-10,0)。

8.2.3.6　resi_group_type_index 的反二值化方法

如果当前编码块宽为16高为2，由二元符号串根据表32　得到resi_group_type_index的值；否则，由二元符号根据8.2.3.2，使用定长码len为1，由二元符号串根据表29得到得到resi_group_type_index的值。

表32　高为 16 宽为 2 的编码块 resi_group_type_index 的值与二元符号串的关系

resi_group_type_index	二元符号串		
0	0		
1	1	0	0
2	1	0	1
3	1	1	0
4	1	1	1
binIdx	0	1	2

8.2.3.7 rg_cl[component][i][j]的反二值化方法

根据以下方法确定rg_cl[component][i][j]的值：

——如果当前编码块是逐点预测模式，则由二元符号串查表33得到rg_cl[component][i][j]的值，其中，maxVal等于BitDepth[component]加1；

——否则，由二元符号串查表33得到rg_cl[component][i][j]的值，其中，maxVal等于BitDepth[component]的值。

表33 rg_cl[component][i][j]的值与二元符号串的关系

rg_cl[component][i][j]的值	二元符号串							
0	0	0						
1	0	1						
2	1	0						
3	1	1	0					
4	1	1	1	0				
5	1	1	1	1	0			
…	1	1	1	1	1	…	0	
maxVal-1	1	1	1	1	1	…	1	0
maxVal	1	1	1	1	1	…	1	1
binIdx	0	1	2	3	4	…	maxVal-3	maxVal-2

8.2.3.8 rg_resi_data的反二值化方法

rg_resi_data的值等于synElVal的值，见8.2.3.2，其中定长码码长len的确定方式为：

——如果SubRgCl[component][j][i>>2]的值小于BitDepth[component]或者当前编码块是点预测模式，则定长码码长len等于SubRgCl[component][i>>2][j]；

——否则，定长码码长len等于BitDepth[component]。

8.2.3.9 sample_data的反二值化方法

sample_data的值等于synElVal的值，见8.2.3.2，其中定长码码长len的确定方式为：

——如果FallbackFlag等于0且MultiplexingEnableFlag等于1，则定长码码长len等于BitDepth[0]；

——如果FallbackFlag等于0且MultiplexingEnableFlag等于0，则定长码码长len等于BitDepth[component]；

——否则，定长码码长len的导出方式如下：

a) 首先，计算adjTargetBpp的值：

1) 如果VirtualEndStuff大于0，adjTargetBpp等于(((TargetBpp<<1)-(1<<RcDecreaseStepLog2)-((MultiplexingEnableFlag&&(SubstreamExpansionRatio==4)&&FallbackType)?7:0))>>5)；

2) 否则，adjTargetBpp等于(((TargetBpp<<1)-((MultiplexingEnableFlag&&(SubstreamExpansionRatio==4)&&FallbackType)?7:0))>>5)。

b) 其次，如果 image_format 不等于‘000’且 MultiplexingEnableFlag 等于 1：
   1) 如果 FallbackType 等于 1 且 adjTargetBpp 等于 3，则令 adjTargetBpp 等于 2；
   2) 否则，如果 FallbackType 等于 0 且 adjTargetBpp 等于 4 且 SubstreamExpansionRatio 等于 4，则令 adjTargetBpp 等于 3。
c) 然后，得到定长码码长 len：
   1) 如果 image_format 等于‘000’，则定长码码长 len 等于(adjTargetBpp-2+FallbackType)；
   2) 否则，如果 component 等于 0，则定长码码长 len 等于(adjTargetBpp-2+FallbackType)/3+(adjTargetBpp-2+FallbackType)%3；
   3) 否则，定长码码长 len 等于(adjTargetBpp-2+FallbackType)/3。
d) 最后，如果 image_format 不等于‘000’且 MultiplexingEnableFlag 等于 1 且 FallbackType 等于 0，len 等于 Min（len，BitDepth[0]-1）

9 解码过程

9.1 图像解码

9.1.1 概述

图像的解码过程如下：
——解码图像头；
——计算当前图像的条带数量；
——如果 MultiplexingEnableFlag 等于 1，则进行位流段解交织操作，见 9.1.2；
——解码当前图像的各个条带，见 9.2，得到各个条带的重建样本；
——重建样本构成解码图像并输出解码图像。

9.1.2 位流段解交织

如果MultiplexingEnableFlag等于1，则进行位流段解交织操作。
首先，将位流按字节放入数组chunkByte[i][j][chunkIdx][k]中：

```
for (j=0; j<PictureHeightInSlice; j++) {
    for (chunkIdx=0; chunkIdx<ChunkNum; chunkIdx++) {
        for (i=0; i< PictureWidthInSlice; i++) {
            if (chunkIdx == ChunkNum – 1) {
                chunkSizeInBit = TotalBits – (ChunkNum – 1) × ChunkSizeInBit
            } else {
                chunkSizeInBit = ChunkSizeInBit
            }
            for (k=0; k<(chunkSizeInBit>>3); k++) {
                chunkByte[i][j][chunkIdx][k]    /* Read 1 byte */
            }
        }
    }
}
```

然后，将chunkByte数组中i,j和chunkIdx均相等的元素按照k从小到大的顺序进行合并，得到chunkData[i][j][chunkIdx]数组。

最后，将chunkData数组中i和j均相等的元素按照chunkIdx从小到大的顺序进行合并，得到sliceData[i][j]数组。sliceData[i][j]数组的每个元素即为水平索引为i，竖直索引为j的条带位流。

9.2 条带解码

条带解码过程为：

——解码参数初始化，见9.3；

——依次解码该条带的编码单元，见9.4。

9.3 解码参数初始化

在解码每个条带前，需要初始化码控的WarmUp数组,EndTargetFullness参数以及ExtraBufferThreshold参数：

```
for (j=0; j<5; j++) {
    WarmUp[j] = 4
}
EndTargetFullness = ((DelayBits – ((!MultiplexingEnableFlag ? 0 : (image_format == '000'? 1 : 3) × (SubstreamSegmentSize – 1))) × RcTargetEndRatio) >> 3
ExtraBufferThreshold = ExtraBufferSize
```

根据ImageFormat查表34得到ChromaSampleRate，InvElem，InvElemShift以及FormatBias的初始化值。

表34 ChromaSampleRate，InvElem，InvElemShift以及FormatBias与ImageFormat的对应关系

ImageFormat	YUV400	YUV420	YUV422	RGB444	YUV444
ChromaSampleRate	0	1	2	4	4
InvElem	1	85	1	85	85
InvElemShift	0	7	1	8	8
FormatBias	0	114	85	128	57

初始化参数LosslessBitsRecord[i][j]，AvgLosslessBitsRecord[i]，PhysicalBufferLevelRecord[i]以及TargetRateRecord[i]。

```
for (i=0; i<3; i++) {
    for (j=0; j<5; j++) {
        LosslessBitsRecord[i][j] = RcLosslessBits[j]
    }
    AvgLosslessBitsRecord[i] = RcAvgLosslessBits
    PhysicalBufferLevelRecord[i] = DelayBits
    TargetRateRecord[i] = TargeBpp << 3
}
```

9.4 编码单元解码

编码单元依次解码一个亮度块和两个色度块。首先，如果FallbackFlag等于0且当前编码块不是样本值预测模式，则需要进行扫描重排序得到残差矩阵ResiData和码长矩阵Cl，见9.5；然后，确定当前编码单元的量化参数，见9.6；最后，根据编码块的预测模式进行解码：

——如果当前编码块是样本值预测模式，则直接解析得到当前编码块的重建样本矩阵，见 9.7.1；

——否则，如果当前编码块是点预测模式，则进行点预测模式解码并得到当前编码块的重建样本矩阵，见 9.7.2；

——否则，先进行块预测模式残差块解码，然后进行预测补偿得到重建样本矩阵，见 9.7.3。

编码单元解码结束后，进行码控参数更新，见 9.6.4。

9.5 扫描重排序

扫描残差矩阵ScanResiData经过重排序得到残差矩阵ResiData，扫描码长矩阵RgCl经过重排序得到码长矩阵Cl，其计算如下：

```
if (PredMode[component] == 'POINT_MODE') {
    for (x=0; x<CbWidth[component]; x+=2) {
        for (y=0; y<CbHeight[component]; y++) {
            if (CbHeight[component] == CuHeight) {    /* Not YUV420 */
                ResiData[component][x][y] = ScanResiData[component][x + y][0]
                Cl[component][x][y] = RgCl[component][(x + y) >> 2][0]
            } else {
                ResiData[component][x][y] = ScanResiData[component][(x >> 1) + y][0]
                Cl[component][x][y] = RgCl[component][((x >> 1) + y) >> 2][0]
            }
        }
    }
    for (x=1; x<CbWidth[component]; x+=2) {
        for (y=0; y<CbHeight[component]; y++) {
            if (CbWidth[component] == CuWidth) {
                ResiData[component][x][y] = ScanResiData[component][x – 1 + y][1]
                Cl[component][x][y] = RgCl[component][(x – 1 + y) >> 2][1]
            } else {
                if (CbHeight[component] == CuHeight) {
                    ResiData[component][x][y] = ScanResiData[component][(CbSize[component] >> 1) + x – 1 + y][0]
                    Cl[component][x][y] = RgCl[component][(((CbSize[component] >> 1) + x – 1 + y) >> 2][0]
                } else {    /* YUV420 Chroma */
                    ResiData[component][x][y] = ScanResiData[component][(CbSize[component] >> 1) + ((x – 1) >> 1) + y][0]
                    Cl[component][x][y] = RgCl[component][((CbSize[component] >> 1) + ((x – 1) >> 1) + y) >> 2][0]
                }
            }
        }
    }
} else {
```

```
        for (x=0; x<CbWidth[component]; x++) {
            for (y=0; y<CbHeight[component]; y++) {
                ResiData[component][x][y] = ScanResiData[component][((x & 7) × CbHeight[component]) + y][x >> 3]
                Cl[component][x][y] = RgCl[component][(((x & 7) × CbHeight[component]) + y) >> 2][x >> 3]
            }
        }
    }
```

9.6 确定量化参数

9.6.1 概述

如果LosslessEnableFlag等于1，则当前编码单元亮度编码块和色度编码块的量化参数Qp[0]和Qp[1]均为0；否则，先根据当前编码单元的亮度复杂度等级ComplexityLevel[0]与色度复杂度等级ComplexityLevel[1]计算编码单元的量化参数MasterQp，见9.6.2，如果当前图像格式为YUV400，则当前编码单元亮度编码块的量化参数Qp[0]等于MasterQp；否则，根据MasterQp计算当前编码单元亮度编码块和色度编码块的量化参数Qp[0]和Qp[1]，见9.6.3。

9.6.2 计算编码单元量化参数

根据当前编码单元的亮度复杂度等级ComplexityLevel[0]与色度复杂度等级ComplexityLevel[1]计算编码单元的量化参数MasterQp：

```
rcBufferLevelPrev = PhysicalBufferLevelRecord[CurrPos] + (VirtualEndStuff < ExtraBufferSize ? 0 : VirtualEndStuff – ExtraBufferSize)
avgLosslessBits = Clip3(0, 262143, AvgLosslessBitsRecord[CurrPos])
shiftCurr = RcFullnessCalcShift – 7
Fullness = Clip3(0, 127, (rcBufferLevelPrev × RcFullnessCalcMultiplier + ((1 << shiftCurr) >> 1)) >> shiftCurr)
bitsOffset = RcBitsOffset + (341 – ((43 × Currbpp) >> 7))
bitsOffset = Max(bitsOffset, 0)
tmp = (avgLosslessBits >> 7) – bitsOffset
tmp = Clip3 (1, 32, ((tmp >> 5) + 1) >> 1)
infoRatio = (Currbpp × InverseTable[tmp – 1] + 256) >> 9
if (image_format == '000') {    /* YUV400 */
    CuComplexityLevel = ComplexityLevel[0]
} else if (image_format == '001') {    /* YUV420 */
    CuComplexityLevel = ComplexityDivide3Table[ComplexityLevel[1] + (ComplexityLevel[0] << 1)]
} else if (image_format == '010') {    /* YUV422 */
    CuComplexityLevel = (ComplexityLevel[0] + ComplexityLevel[1]) >> 1
} else if (image_format == '011' || image_format == '100') {    /* YUV444 or RGB444 */
    infoRatio = infoRatio > ((77 × (2 + ChromaSampleRate)) >> 1) ? (77 × (2 + ChromaSampleRate)) >> 1 : infoRatio
    CuComplexityLevel = ComplexityDivide3Table[ComplexityLevel[0] + (ComplexityLevel[1] << 1)]
}
losslessBits = Clip3(0, 2047, LosslessBitsRecord[CurrPos][CuComplexityLevel])
relativeLosslessBits = Clip3(0, RcMaxRelativeBits, RcMaxLosslessBits – losslessBits)
```

```
tmp = RcRatio0 × Fullness
tmp = (tmp + 8) >> 4
tmp = (RcParam0 – tmp) × Currbpp
tmp = (tmp + 64) >> 7
minRate = Max(0, tmp)
tmp = RcParam2 – (((Fullness ×RcRatio2 + 64) >> 7)
if (relativeLosslessBits > RcRelativeTh && CuComplexityLevel <= RcComplexTh) {
    bppOffset1 = tmp – ((ChromaSampleRate × relativeLosslessBits + 1) >> 1)
} else {
    bppOffset1 = tmp – (((ChromaSampleRate + 2) × relativeLosslessBits + 1) >> 1)
}
bppOffset2 = Max(1536 – Currbpp, 512) – relativeLosslessBits
bppOffset3 = RcParam1 – (((RcRatio1 × Fullness + 16) >> 5)
bppOffset = Min(Min(bppOffset1, bppOffset2), bppOffset3)
if (CbPosY[0] == CurrSlicePosY) {
    bppOffset = Max(bppOffset, –128)
}
maxRate = Max(Currbpp + bppOffset, minRate)
tmp = (infoRatio × Max(0, losslessBits – bitsOffset) + 64) >> 7
tmp = Clip3(minRate, maxRate, tmp)
TargetRate = Clip3(0, 6143, tmp)
availableBuffer = MaxBufferSize – rcBufferLevelPrev + ((Targetbpp << 4) + (Targetbpp << 3) – TargetRate – TargetRateRecord[(CurrPos + 1) % 3] – TargetRateRecord[(CurrPos + 2) % 3]) >> 2;
if (VirtualEndStuff > 0) {
    ExtraBufferThreshold = Max(0, ExtraBufferThreshold – (1 << RcExtraBufferDecreaseStepLog2))
}
if (availableBuffer < ExtraBufferThreshold) {
    TargetRate = Max(0, TargetRate – ((ExtraBufferThreshold – availableBuffer) >> (ExtraBufferPenaltyLog2 – 2)));
}
tmp = (TargetRate × InvElem + ((1 << InvElemShift) >> 1)) >> InvElemShift
tmp = (losslessBits – tmp) << 3
MasterQp = Clip3(–960, 18304, tmp)
```

其中，InverseTable以及ComplexityDivide3Table的定义为：

InverseTable = { 1024, 512, 341, 256, 205, 171, 146, 128, 114, 102, 93, 85, 79, 73, 68, 64, 60, 57, 54, 51, 49, 47, 45, 43, 41, 39, 38, 37, 35, 34, 33, 32 }

ComplexityDivide3Table = { 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4 }

9.6.3 计算亮色度编码块量化参数

计算亮度量化参数Qp[0]以及色度量化参数Qp[1]，Qp[2]。

```
Bias = (RcQpBias[ComplexityLevel[0]][ComplexityLevel[1]] × FormatBias) >> 1
tmp = ChromaSampleRate × Bias
tmp = ((tmp << 7) + 128) >> 8
qpHighPrec[0] = Clip3(0, BitDepth[0] << 3, (MasterQp – tmp + 64) >> 7)
```

```
qpHighPrec[1] = Clip3(0, BitDepth[1] << 3, (MasterQp + Bias + 64) >> 7)
qpRem[0] = qpHighPrec[0] & 7
qpRem[1] = qpHighPrec[1] & 7
if (image_format == '001') {    /* YUV420 */
    roundThres = 11
} else if (image_format == '010') {    /* YUV422 */
    roundThres = 12
} else if (image_format == '011' || image_format == '100') {    /* YUV444 or RGB444 */
    roundThres = 13
}
if (qpRem[0] >= 5 && qpRem[1] >= 5 && qpRem[0] + qpRem[1] <= roundThres) {
    Qp[0] = qpHighPrec[0] >> 3
    Qp[1] = (qpHighPrec[1] + 7) >> 3
} else {
    Qp[0] = (qpHighPrec[0] + 3) >> 3
    Qp[1] = (qpHighPrec[1] + 3) >> 3
}
Qp[2] = Qp[1]
```

9.6.4 码控参数更新

先根据实际编码开销CuBits和量化参数（Qp[0],Qp[1]）计算当前编码单元的无损编码比特数losslessBitsCurr：

```
tmp = (CuBits << 2) + (Qp[0] << 7) + ((ChromaSampleRate × Qp[1]) << 6)
losslessBitsCurr = Clip3(0, 2047, (tmp × InvElem + ((1 << InvElemShift) >> 1)) >> InvElemShift)
```

再根据当前编码单元的无损编码比特数losslessBitsCurr更新码控参数AvgLosslessBitsRecord[i]，LosslessBitsRecord[i][j]，PhysicalBufferLevelRecord[i]，TargetRateRecord[i]以及CurrPos：

```
prevPos = (CurrPos + 2) % 3
tmp = AvgLosslessBitsRecord[prevPos] × 1014 + ((losslessBitsCurr << 14) >> 7) × 10
AvgLosslessBitsRecord[CurrPos] = (tmp + 512) >> 10
for (j=0; j<5; j++) {    /* Copy LosslessBits history */
    LosslessBitsRecord[CurrPos][j] = LosslessBitsRecord[prevPos][j]
}
if (FallbackFlag == 0 && (CbPosX[0] != CurrSlicePosX || CbPosY[0] !=CurrSlicePosY)) {
    updateRate = WarmUp[CuComplexityLevel] + 2
    WarmUp[CuComplexityLevel] = WarmUp[CuComplexityLevel] > 0 ? WarmUp[CuComplexityLevel] – 1 : 0
    LosslessBitsRecord[CurrPos][CuComplexityLevel] = (LosslessBitsRecord[prevPos][CuComplexityLevel] × (8 – updateRate) + losslessBitsCurr × updateRate + 4) >> 3    /* Update current LosslessBits */
}
PhysicalBufferLevelRecord[CurrPos] = PhysicalBufferLevel + VirtualStartStuff
TargetRateRecord[CurrPos] = TargetRate
```

9.7 预测模式解码

9.7.1 样本值模式解码

样本值模式的编码块重建样本矩阵ReconMatrix[component]计算如下：

```
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
        ReconMatrix[component][x][y] = SampleValue[component][x][y]
    }
}
```

9.7.2 点预测模式解码

9.7.2.1 概述

点预测模式解码过程中，每个样本位置的扫描顺序与点预测模式相关，见9.7.2.2，每个样本位置的解码过程包括：确定当前残差样本的量化参数PixelQp，见9.7.2.3；对当前残差样本进行反量化，见9.7.2.4；根据9.7.2.2 的预测方式导出当前预测样本进行预测补偿得到当前重建样本，见9.7.2.5。

9.7.2.2 点预测模式残差编码扫描顺序与预测方式

点预测模式根据模式和编码块尺寸确定样本位置扫描顺序：

——如果当前编码块的宽为 16 高为 2，残差编码按图 10 顺序扫描，按图 13 的预测方式进行预测；

——否则，如果当前编码块的宽为 8 高为 2，残差编码按图 11 顺序扫描，按图 14 的预测方式进行预测；

——否则（当前编码块的宽为 8 高为 1），残差编码按图 12 顺序扫描，按图 15 的预测方式进行预测。

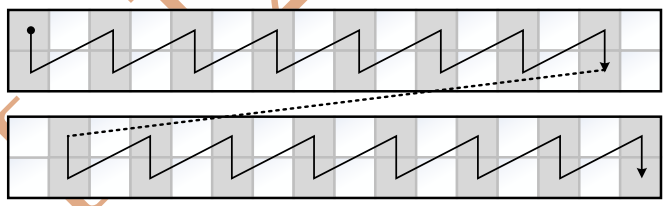


图10 宽为 16 高为 2 的编码块 POINT_MODE 模式残差编码扫描顺序（圆点为第一个位置）

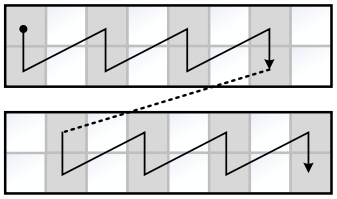


图11 宽为 8 高为 2 的编码块 POINT_MODE 模式残差编码扫描顺序（圆点为第一个位置）

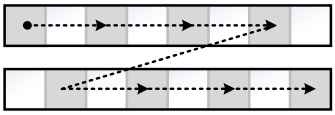


图12　宽为 8 高为 1 的编码块 POINT_MODE 模式残差编码扫描顺序（圆点为第一个位置）

T	RL	T	RL	T	RL	T	RL	T	RL	T	RL	T	RL	T	L
T	RL	T	RL	T	RL	T	RL	T	RL	T	RL	T	RL	T	L

图13　宽为 16 高为 2 的编码块 POINT_MODE 模式预测方式

T	RL	T	RL	T	RL	T	L
T	RL	T	RL	T	RL	T	L

图14　宽为 8 高为 2 的编码块 POINT_MODE 模式预测方式

T	RL	T	RL	T	RL	T	L

图15　宽为 8 高为 1 的编码块 POINT_MODE 模式预测方式

当前预测样本在当前编码块内的水平索引为x，竖直索引为y，用I表示当前编码块所在的图像通道的重建样值矩阵。图13~图15中，点预测模式的三种预测方式RL，L以及T的具体计算过程为：

a) 点预测模式 RL 预测方式，预测样本 PointPredData[component][x][y]的计算过程如下：

PointPredData[component][x][y] = (I[CbPosX[component] + x – 1][CbPosY[component] + y] + I[CbPosX[component] + x + 1][CbPosY[component] + y] + 1) >> 1

b) 点预测模式 L 预测方式，预测样本 PointPredData[component][x][y]的计算过程如下：

PointPredData[component][x][y] = I[CbPosX[component] + x – 1][CbPosY[component] + y]

c) 点预测模式 T 预测方式，预测样本 PointPredData[component][x][y]的计算过程如下：

PointPredData[component][x][y] = I[CbPosX[component] + x][CbPosY[component] + y – 1]

其中，需要确保在当前样本位置预测前，用于预测的样本位置已经完成重建，如果矩阵I的二维索引值中，任一值为负数，则其样本值为1<<(BitDepth[component]-1)。

9.7.2.3　点预测模式调整量化参数

9.7.2.3.1　概述

如果PwqEnableFlag等于1，点预测模式的非Slice首行位置且预测方式非点预测T预测方式的残差样本需要进行量化参数调整得到PixelQp；否则，PixelQp等于Qp[component]。

9.7.2.3.2　导出预测残差值

导出当前残差样本（非Slice首行位置且预测方式非点预测T预测方式）的预测残差值PredResiTmp，当前残差样本在当前编码块内的水平索引为x，竖直索引为y，用I表示当前编码块所在的图像通道的重建样值矩阵，PredResiTmp的步骤如下：

a) 如果当前残差样本为当前编码块首行首个残差样本且预测方式为点预测 T 预测方式，PredResiTmp 的计算过程如下：

```
PredResiTmp = Abs(I[CbPosX[component] + x][CbPosY[component] + y – 1] – I[CbPosX[component] + x + 1][CbPosY[component] + y – 1])
```

b) 否则，如果当前残差样本为当前编码块首行且预测方式为点预测 T 预测方式，PredResiTmp 的计算过程如下：

```
PredResiTmp = (Abs(I[CbPosX[component] + x – 1][CbPosY[component] + y – 1] – I[CbPosX[component] + x][CbPosY[component] + y – 1]) + Abs(I[CbPosX[component] + x][CbPosY[component] + y – 1] – I[CbPosX[component] + x + 1][CbPosY[component] + y – 1]) + 1) >> 1
```

c) 否则，如果当前残差样本的预测方式为点预测 RL 预测方式，PredResiTmp 的计算过程如下：

```
PredResiTmp = (Abs(PointResiData[component][x – 1][y]) + Abs(PointResiData[component][x + 1][y]) + 1) >> 1
```

d) 否则，如果当前残差样本的预测方式为点预测 L 预测方式，PredResiTmp 的计算过程如下：

```
PredResiTmp = Abs(PointResiData[component][x – 1][y])
```

e) 否则（当前残差样本非当前编码块首行且预测方式为点预测 T 预测方式），PredResiTmp 的计算过程如下：

```
PredResiTmp = Abs(PointResiData[component][x][y – 1])
```

9.7.2.3.3 量化参数调整方式

确定当前残差样本的量化参数PixelQp的调整方式，当前残差样本在当前编码块内的水平索引为x，竖直索引为y。输入是当前通道的比特深度BitDepth[component]，该残差样本所在编码块的量化参数Qp[component]和当前残差样本的预测残差值PredResiTmp，见9.7.2.3.2。输出是该残差样本的量化参数PixelQp。计算过程如下：

a) 根据 BitDepth[component]确定参数 jndQp、jndQps、adjustMaxQp、resiThres0 和 resiThres1 的值。

```
bdIdx = Clip3(0, 8, BitDepth[component] – 8)
jndQp = (bdIdx >> 1) + JndQp
jndQps = component == 0 ? (bdIdx >> 1) + StrictJndQp : (bdIdx >> 1) + StrictJndQp + 1
adjustMaxQp = (bdIdx >> 1) + PwqMaxQp
resiThres0 = QpRefineTh0 << (bdIdx >> 1)
resiThres1 = QpRefineTh1 << (bdIdx >> 1)
```

b) 确定残差样本的量化参数 PixelQp:

1) 如果 ComplexityLevel[component==0?0:1]等于 0，则：

——如果残差样本位于当前编码块首行且预测方式为点预测 T 预测方式，则：

- 如果 Qp[component]大于 jndQps 且 PredResiTmp 小于或等于 resiThres0，则：

```
PixelQp = Max(Qp[component] – 1, jndQps)
```

- 否则，如果 Qp[component]大于 jndQp 且 PredResiTmp 小于或等于 resiThres1，则：

```
PixelQp = Max(Qp[component] – 1, jndQp)
```

- 否则，令 PixelQp 等于 Qp[component]。

——否则,如果 Qp[component]大于 jndQps 且 Qp[component]小于或等于 adjustMaxQp 且 PredResiTmp 小于或等于 resiThres1，则：

```
PixelQp = Max(Qp[component] – 1, jndQps)
```

——否则，令 PixelQp 等于 Qp[component]。

2) 否则，

——如果残差样本位于当前编码块首行且预测方式为点预测 T 预测方式，则：

- 如果 Qp[component]大于 jndQp 且 PredResiTmp 小于或等于 resiThres0 且 ComplexityLevel[component==0?0:1]大于等于 BwqComplexTh，则：

```
PixelQp = Max(Qp[component] – 2, jndQp)
```

- 否则，如果 Qp[component]大于 jndQp 且 PredResiTmp 小于或等于 resiThres1，则：

```
PixelQp = Max(Qp[component] – 1, jndQp)
```

- 否则，令 PixelQp 等于 Qp[component]。

——否则，如果 Qp[component]大于 jndQp 且 Qp[component]小于或等于 adjustMaxQp 且 PredResiTmp 小于或等于 resiThres1，则：

```
PixelQp = Max(Qp[component] – 1, jndQp)
```

——否则，令 PixelQp 等于 Qp[component]。

9.7.2.4 点预测模式反量化

定义点预测残差样本PointResiData[component][x][y]的反量化过程。

计算过程如下：

```
PointResiData[component][x][y] = BlockResiData[component][x][y] << PixelQp
```

9.7.2.5 点预测模式预测补偿

定义重建样本ReconMatrix[component][x][y]的导出过程，其中component是通道索引，x表示当前样本位置在编码块内的水平索引，y表示当前样本位置在编码块内的竖直索引。

计算过程如下：

```
ReconMatrix[component][x][y] = Clip3(0, (1 << BitDepth[component]) - 1, PointPredData[component][x][y] + PointResiData[component][x][y])
```

9.7.3 块预测模式解码

9.7.3.1 概述

依次进行以下步骤：

a) 先进行块预测模式量化参数预调整，见 9.7.3.2；

b) 对残差块进行反量化，见 9.7.3.3.2，得到反量化后的残差块。将反量化后的残差块中的样本作为重建的残差样本 ResidualMatrix；

c) 根据块预测模式，导出当前编码块的预测样本矩阵 PredMatrix，见 9.7.3.4.2；

d) 用重建的残差样本对预测样本进行补偿，得到当前编码块的重建样本矩阵 ReconMatrix，见 9.7.3.4.3。

9.7.3.2 块预测模式量化参数预调整

定义块预测模式量化参数预调整矩阵QpArray[k][x][y]的导出过程（k=0,1,2）。

a) 根据BitDepth[component]确定参数jndQp、jndQpIbc、resiThres0和resiThres1的值。

```
bdIdx = Clip3(0, 8, BitDepth[component] – 8)
jndQp = (bdIdx >> 1) + JndQp
jndQpIbc = 0
resiThres0 = QpRefineTh0 << (bdIdx >> 1)
resiThres1 = QpRefineTh1 << (bdIdx >> 1)
```

b) 用I表示当前编码块所在的图像通道的重建样值矩阵,确定量化参数预调整矩阵QpArray[k][x]:

```
for (x=0; x<(CbWidth[component]>>1); x++) {
    if (CbPosY[component] == CurrSlicePosY || BwqEnableFlag || (!BwqFlag[component] && PredMode[component] != 'IBC_MODE')) {
        QpArray[0][x] = Qp[component]
        QpArray[1][x] = Qp[component]
        QpArray[2][x] = Qp[component]
    } else {
        for (k=0; k<3; k++) {
            if (k == 0 || (k == 1 && x == 0) || (k == 2 && x == (CbWidth[component] >> 1) – 1)) {
                grad = Abs(I[CbPosX[component] + (x << 1)][CbPosY[component] – 1] – I[CbPosX[component] + (x << 1) + 1][CbPosY[component] – 1])
            } else if (k == 1) {
                grad = (Abs(I[CbPosX[component] + (x << 1) – 1][CbPosY[component] – 1] – I[CbPosX[component] + (x << 1)][CbPosY[component] – 1]) + Abs(I[CbPosX[component] + (x << 1)][CbPosY[component] – 1] – I[CbPosX[component] + (x << 1) + 1][CbPosY[component] – 1]) + 1) >> 1
            } else {  /* k == 2*/
                grad = (Abs(I[CbPosX[component] + (x << 1)][CbPosY[component] – 1] – I[CbPosX[component] + (x << 1) + 1][CbPosY[component] – 1]) + Abs(I[CbPosX[component] + (x << 1) + 1][CbPosY[component] – 1] – I[CbPosX[component] + (x << 1) + 2][CbPosY[component] – 1]) + 1) >> 1
            }
            if (PredMode[component] == 'IBC_MODE') {
                qpMin = jndQpIbc
            } else {
                qpMin = jndQp
            }
            if (Qp[component]<= qpMin) {
                QpArray[k][x] = Qp[component]
            } else if (grad <= resiThres0 && ComplexityLevel[component == 0 ? 0 : 1] >= BwqComplexTh) {
                QpArray[k][x] = Max(Qp[component] – 2, qpMin)
            } else if (grad <= resiThres1) {
                QpArray[k][x] = Max(Qp[component] – 1, qpMin)
            } else {
                QpArray[k][x] = Qp[component]
            }
        }
    }
```

```
}
```

9.7.3.3 块预测模式残差块解码

9.7.3.3.1 概述

定义宽为CbWidth[component]高为CbHeight[component]的残差样本矩阵ResidualMatrix的导出过程，即反量化过程，见9.7.3.3.2。

9.7.3.3.2 块预测模式反量化

定义残差样本矩阵BlockResiData[component]的反量化过程，计算过程如下：

```
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
        if (Cl[component][x][y] == BitDepth[component]) {
            BlockResiData[component][x][y] = BlockResiData[component][x][y]
        } else {
            if (CuDpIntraFlag && CuSdpIntraFlag[x / (CbWidth[component] >> 2)] && PredMode[component] != 'IBC_MODE') {
                BlockResiData[component][x][y] = BlockResiData[component][x][y] << Max(0, BitDepth[0] –10)
            } else {
                if ((BwqFlag[component] && (PredMode[component] == 'INTRA_MODE_DC' || PredMode[component] == 'INTRA_MODE_ANG_0' || PredMode[component] == 'INTRA_MODE_ANG_1')) || (PredMode[component] == 'IBC_MODE' && AbsBvDMinus1[x / (CbWidth[component] >> 3)] == 0 && CbPosY[0]!= CurrSlicePosY)) {
                    BlockResiData[component][x][y] = BlockResiData[component][x][y] << QpArray[0][x >> 1]
                } else if (BwqFlag[component] && (PredMode[component] == 'INTRA_MODE_ANG_2' || PredMode[component] == 'INTRA_MODE_ANG_4')) {
                    BlockResiData[component][x][y] = BlockResiData[component][x][y] << QpArray[1][x >> 1]
                } else if (BwqFlag[component] && (PredMode[component] == 'INTRA_MODE_ANG_3' || PredMode[component] == 'INTRA_MODE_ANG_5')) {
                    BlockResiData[component][x][y] = BlockResiData[component][x][y] << QpArray[2][x >> 1]
                } else {
                    BlockResiData[component][x][y] = BlockResiData[component][x][y] << Qp[component]
                }
            }
        }
    }
}
```

9.7.3.4 块预测模式预测补偿

9.7.3.4.1 概述

定义块预测模式的预测过程和补偿过程。预测过程是确定每个预测块的预测样本矩阵PredMatrix，包括普通帧内预测模式的预测过程，见9.7.3.4.2.1，和块复制帧内预测模式的预测过程，见

9.7.3.4.2.2。补偿过程是利用重建的残差样本矩阵ResidualMatrix对预测样本矩阵PredMatrix进行补偿，见9.7.3.4.3，得到重建样本矩阵ReconMatrix。

9.7.3.4.2 块预测模式预测过程

9.7.3.4.2.1 普通帧内预测模式预测过程

用I表示当前编码块所在的图像通道的重建样值矩阵。

a) 第一步，设当前编码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：

1) 初始化 r[i]、c[j]为 $2^{BitDepth[component]-1}$（i=0～(CbWidth[component]+2)，j=0～CbHeight[component]）；

2) 如果坐标为(x0+i-1, y0-1)（i=1～CbWidth[component]）的样本均“可用”，则r[i]等于I[x0+i-1][y0-1]，r[i]“可用”；否则r[i]“不可用”；

3) 如果坐标为(x0+i-1, y0-1)（i=(CbWidth[component]+1)～(CbWidth[component]+2)）的样本“可用”，则r[i]等于I[x0+i-1][y0-1]，r[i]“可用”；否则r[i]等于r[CbWidth[component]]，r[i]是否“可用”由r[CbWidth[component]]是否“可用”决定；

4) 如果坐标为(x0-1, y0+j-1)（j=1～CbHeight[component]）的样本均“可用”，则c[j]等于I[x0-1][y0+j-1]，c[j]“可用”；否则c[j]“不可用”；

5) 如果坐标为(x0-1, y0-1)的样本“可用”，则r[0]等于I[x0-1][y0-1]，r[0]“可用”；否则

——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

——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

——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”。

其中，样本“存在”指该样本在图像内并且该样本应与当前编码单元属于同一条带；否则样本“不存在”。如果图像样本“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”。

b) 第二步，根据预测模式PredMode[component]确定当前编码块的预测样本PredMatrix的方法如下：

1) 如果PredMode[component]等于‘INTRA_MODE_DC’，

```
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
        rr = 0
        srr[0] = srr[1] = srr[2] = srr[3] = 0
        for (i=1; i<=CbWidth[component]; i++) {
            rr += r[i]
            srr[(i – 1) >> 2] += r[i]
        }
        if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
            PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)] <<
Max(0, BitDepth[0] – 10)
        } else {
            if (!BwqFlag[component]) {
```

```
                    PredMatrix[component][x][y] = (rr + (CbWidth[component] >> 1)) / CbWidth[component]
                } else {
                    PredMatrix[component][x][y] = (srr[x >> 2] + 2) / 4
                }
            }
        }
    }
```

2) 否则，如果 PredMode[component]等于‘INTRA_MODE_ANG_0’，

```
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
        if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
            PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)] << Max(0, BitDepth[0] – 10)
        } else {
            PredMatrix[component][x][y] = r[x + 1]
        }
    }
}
```

3) 否则，如果 PredMode[component]等于‘INTRA_MODE_ANG_1’，

```
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
        if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
            PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)] << Max(0, BitDepth[0] – 10)
        } else {
            PredMatrix[component][x][y] = c[y + 1]
        }
    }
}
```

4) 否则，如果 PredMode[component]等于‘INTRA_MODE_ANG_2’，

——如果 x 大于等于 y，

```
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
        if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
            PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)] << Max(0, BitDepth[0] – 10)
        } else {
            PredMatrix[component][x][y] = r[x – y]
        }
    }
}
```

——否则，

```
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
```

```
            if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
                PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)]
<< Max(0, BitDepth[0] – 10)
            } else {
                PredMatrix[component][x][y] = c[y – x]
            }
        }
}
```

5) 否则，如果 PredMode[component]等于‘INTRA_MODE_ANG_3’，

```
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
        if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
            PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)] <<
Max(0, BitDepth[0] – 10)
        } else {
            PredMatrix[component][x][y] = r[x + y + 2]
        }
    }
}
```

6) 否则，如果 PredMode[component]等于‘INTRA_MODE_ANG_4’，

——如果 y 等于 0，

```
for (x=0; x<CbWidth[component]; x++) {
    if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
        PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)] <<
Max(0, BitDepth[0] – 10)
    } else {
        PredMatrix[component][x][y] = (r[x] + r[x + 1] + 1) >> 1
    }
}
```

——否则，

```
for (x=0; x<CbWidth[component]; x++) {
    if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
        PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)] <<
Max(0, BitDepth[0] – 10)
    } else {
        PredMatrix[component][x][y] = r[x]
    }
}
```

7) 否则，如果 PredMode[component]等于‘INTRA_MODE_ANG_5’，

——如果 y 等于 0，

```
for (x=0; x<CbWidth[component]; x++) {
    if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
        PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)] <<
Max(0, BitDepth[0] – 10)
```

```
    } else {
        PredMatrix[component][x][y] = (r[x + 1] + r[x + 2] + 1) >> 1
    }
}
```

——否则，

```
for (x=0; x<CbWidth[component]; x++) {
    if (CuDpIntraFlag && SdpIntraFlag[x / (CbWidth[component] >> 2)]) {
        PredMatrix[component][x][y]= SdpIntraOffset[component][x / (CbWidth[component] >> 2)] << Max(0, BitDepth[0] – 10)
    } else {
        PredMatrix[component][x][y] = r[x+2]
    }
}
```

9.7.3.4.2.2 块复制帧内预测模式预测过程

定义块复制帧内预测样本矩阵PredMatrix的导出过程。所导出的预测样本矩阵的水平尺寸和垂直尺寸与当前预测单元中对应分量预测块的水平尺寸和垂直尺寸一致。用I表示当前编码块所在的图像通道的重建样值矩阵。导出过程如下：

a) 确定当前编码块的子块的划分情况，步骤如下：
   1) 如果当前编码块坐标(x0-1, y0-1)样本“不可用”，且坐标(x0, y0-1)样本“可用”，将当前编码块划分为（CbHeight[component]×4）个宽为 CbWidth[component]>>2 高为 1 的子块；
   2) 否则，将当前编码块划分为 8 个宽为 CbWidth[component]>>3 高为 CbHeight[component] 的子块
b) 确定每个子块的预测样本 predIbc，记子块宽度为 SbWidth，子块高度为 SbHeight，过程如下：
   1) 对于第 n 个预测块 $predIbc_n$，计算块矢量的绝对值 AbsBv[component][n]。

```
if (CbPosX[componet] == CurrSlicePosX && CbPosY[component] != CurrSlicePosY) {
    maxAbsBv = Min(15, SliceWidth – 4)
    if (image_format == ‘001’ && component != 0) {   /* YUV420 Chroma */
        AbsBvD[component][n] = DpEnableFlag && AbsBvD[0][n << 1] == 31 ? AbsBvD[0][n << 1] : Clip3(0, maxAbsBv, AbsBvD[0][n << 1])
    } else {
        AbsBvD[component][n] = DpEnableFlag && AbsBvDMinus1[n] == 31 ? AbsBvDMinus1[n] : Clip3(0, maxAbsBv, AbsBvDMinus1[n])
    }
} else {
    BvOffset = n < 4 ? 0 : (CbWidth[component] >> 1)
    maxAbsBv = Min(32, CbPosX[component] + BvOffset – CurrSlicePosX)
    AbsBvD[component][n] = Clip3(0, maxAbsBv, AbsBvDMinus1[n] + 1)
}
```

   2) 对于预测块 $predIbc_n$ 中的任意样本位置(x, y)，则：

```
for (x=0; x<SbWidth; x++) {
    for (y=0; y<SbHeight; y++) {
```

```
            if (CbPosX[componet] == CurrSlicePosX && CbPosY[component] != CurrSlicePosY) {
                if (DpEnableFlag && AbsBvD[component][n] == 31) {
                    if (image_format == '001' && component != 0) {    /* YUV420 Chroma */
                        predIbcn[x][y] = SdpIbcOffset[component][n << 1]
                    } else {
                        predIbcn[x][y] = SdpIbcOffset[component][n]
                    }
                } else {
                    predIbcn[x][y] = I[CbPosX[component] + AbsBvD[component][n] + x][CbPosY[component] + y – 1]
                }
            } else
                if (DpEnableFlag && AbsBvD[component][n] == 31 && !FallbackFlag) {
                    predIbcn[x][y] = SdpIbcOffset[component][n]
                } else if (AbsBvD[component][n] == 0 && !FallbackFlag) {
                    predIbcn[x][y] = I[CbPosX[component] + n×SbWidth + x][CbPosY[component] + y – 1]
                } else if (CbPosX[component] + BvOffset + x < AbsBvD[component][n]) {
                    predIbcn[x][y] = I[CbPosX[component] + BvOffset – AbsBvD[component][n] + x][CbPosY[component] + y]
                } else {
                    predIbcn[x][y] = 1 << (BitDepth[component] – 1)
                }
            }
        }
    }
```

c) 根据每个预测块的预测样本 $predIbc_n$ 导出当前编码块的预测样本矩阵 PredMatrix：

```
offset = (CbWidth[component] == CuWidth) ? 1 : 0
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
        if (CbPosX[componet] == CurrSlicePosX && CbPosY[component] != CurrSlicePosY) {
            PredMatrix[component][x][y] = predIbc ((x >> (offset + 1)) << 1) + y[x % (1 << (offset + 1))][0]
        } else {
            PredMatrix[component][x][y] = predIbcx >> offset[x % (offset + 1)][y]
        }
    }
}
```

9.7.3.4.3 块预测模式补偿过程

补偿后的重建样本矩阵ReconMatrix计算如下：

```
for (x=0; x<CbWidth[component]; x++) {
    for (y=0; y<CbHeight[component]; y++) {
        if (Cl[component][x][y] == BitDepth[component]) {
```

```
                ReconMatrix[component][x][y] = ResidualMatrix[component][x][y]
            } else {
                ReconMatrix[component][x][y] = Clip3(0, (1 << BitDepth[component]) – 1,
PredMatrix[component][x][y] + ResidualMatrix[component][x][y])
            }
            if (BrcFlag[component] && PredMode[component] == 'INTRA_MODE_ANG_1') {
                ReconMatrix[component][x][y] += (BrcOffset[(x >> 3) + (y << 1)] << Max(0, BitDepth[component] –
8))
            } else if (BrcFlag[component] && PredMode[component] == 'INTRA_MODE_DC' || PredMode[component]
== 'INTRA_MODE_ANG_0' || PredMode[component] == 'INTRA_MODE_ANG_2' || PredMode[component] ==
'INTRA_MODE_ANG_3' || PredMode[component] == 'INTRA_MODE_ANG_4' || PredMode[component] ==
'INTRA_MODE_ANG_5') {
                ReconMatrix[component][x][y] += (BrcOffset[x >> 2] << Max(0, BitDepth[component] – 8))
            }
            ReconMatrix[component][x][y] = Clip3(0, (1 << BitDepth[component]) – 1, ReconMatrix[component][x][y])
        }
    }
```

附 录 A
（规范性）
RGB 与 YCoCg 的转换方法

本附录定义RGB格式图像转YCoCg4:4:4以及YCoCg4:4:4转RGB格式图像转的方法。YCoCg4:4:4格式描述见6.1.2.4。

RGB转换为YCoCg4:4:4的公式如下：

```
Y = (R >> 2) + (G >> 1) + (B >> 2)
Co = R – B
Cg = G – (R >> 1) – (B >> 1)
Co = Co + (1 << BitDepth[0])
Cg = Cg + (1 << BitDepth[0])
if (BitDepth[0] == 16 && !LosslessEnableFlag) {
    Co = Co >> 1
    Cg = Cg >> 1
}
```

注：当RGB转换为YCoCg之后，如果亮度Y的位宽小于16，或者，亮度Y的位宽等于16且开启客观无损配置，则色度Co以及Cg的位宽相比于亮度Y的位宽，增加了1。

YCoCg4:4:4转换为RGB的公式如下：

```
if (BitDepth[0] == 16 && !LosslessEnableFlag) {
    Co = Co << 1
    Cg = Cg << 1
}
Co = Co – (1 << BitDepth[0])
Cg = Cg – (1 << BitDepth[0])
R = Co + Y – (Co >> 1) – (Cg >> 1)
G = Cg + Y – (Cg >> 1)
B = Y – (Co >> 1) – (Cg >> 1)
```

附 录 B
（规范性）
档次

B.1 概述

档次提供了一种定义本文件的语法和语义的子集的手段。档次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。档次是本文件规定的语法、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。

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

profile_id定义了位流的档次。

B.2 档次

本文件定义的档次见表B.1。

表B.1 档次

profile_id的值	档次
0x00	禁止
0x20	帧存档次（Frame buffer profile）
0x30	接口档次（Interface profile）
其他	保留

帧存档次的位流应满足以下条件：

——profile_id 的值应为 0x20

——transform_enable_flag 的值应为‘0’

——bit_depth 的值应为 8 或 10 或 12 或 14 或 16

——padding_type_flag 的值应为‘1’

接口档次的位流应满足以下条件：

——profile_id 的值应为 0x30

——lossless_flag 的值应为‘0’

——transform_enable_flag 的值应为‘0’

——bit_depth 的值应为 8 或 10 或 12 或 14 或 16

——vbr_enable_flag 的值应为‘0’

附 录 C
（规范性）
位流参考缓冲区管理

C.1 概述

本附录定义了条带的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.2定义的方式进入BBV缓冲区和移出BBV缓冲区。符合本文件的位流不应导致BBV缓冲区上溢。符合本文件的位流在每个编码单元的解码时刻均不应导致BBV缓冲区下溢。

C.2 数据输入和数据移出

C.2.1 概述

本附录定义了帧存档次和接口档次的条带的编码数据进入BBV缓冲区以及移出缓冲区的方法。

C.2.2 帧存档次数据输入

在BBV中，第n个编码单元的编码数据f(n)包括当前编码单元的所有编码数据。

在每个编码单元完成编码后，f(n)输入至BBV。

C.2.3 帧存档次数据移出

如果当前条带位于图像上边界，则在当前条带列的第TransmissionDelayCu个编码单元完成编码之后，每个编码单元完成编码后，BBV需要移出outBits个比特；否则，当前条带的每个编码单元完成编码后，BBV需要移出outBits个比特。其中，outBits等于((CuSize×TargetBpp)>>4)。

图C.1 为位于图像上边界的条带内的编码单元的编码数据进入BBV缓冲区以及数据移出的示意图。其中，块时间为当前CU开始编码到下一个CU开始编码直接的时间，一般为常数；虚线为数据移出；a和n的值为：

a = TransmissionDelayCu

n = SliceWidthInCu × SliceHeightInCu

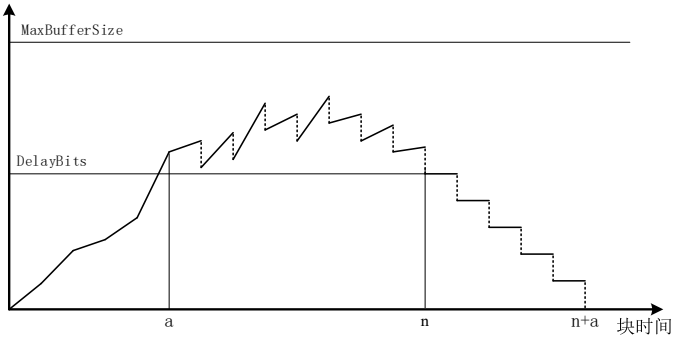


图 C.1 BBV 数据输入和数据移出示意图

C.2.4 接口档次数据输入

在BBV中，第n个编码单元的编码数据f(n)包括以下数据（如果存在）：

——当前编码单元的所有编码数据；

——跟在当前编码单元后的填充数据

在每个编码单元完成编码后，f(n)输入至 BBV。

C.2.5 接口档次数据移出

如果当前条带位于图像上边界，则在当前条带列的第TransmissionDelayCu个编码单元完成编码之后，每个编码单元完成编码后，BBV需要移出outBits个比特；否则，当前条带的每个编码单元完成编码后，BBV需要移出outBits个比特。其中，outBits等于((CuSize×TargetBpp)>>4)。

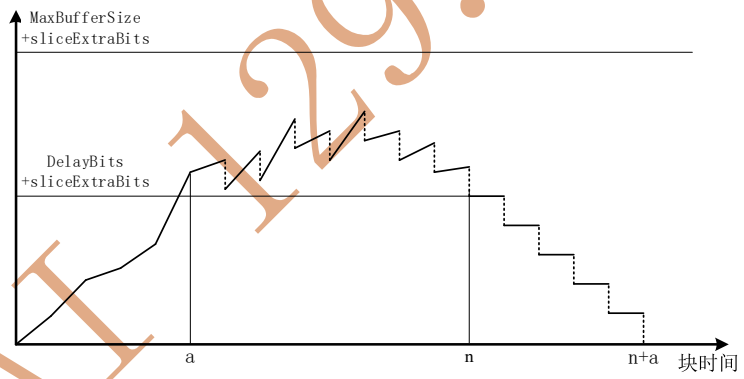
需要注意的是，由于接口档次需要适配传输层，若TmpSliceAlignAdj的值不等于0，outBits需要增加个TmpSliceAlignAdj比特。

图C.2 为位于图像上边界的条带内的编码单元的编码数据进入BBV缓冲区以及数据移出的示意图。其中，块时间为当前CU开始编码到下一个CU开始编码直接的时间，一般为常数；虚线为数据移出；a，n以及sliceExtraBits的值为：

a = TransmissionDelayCu

n = SliceWidthInCu × SliceHeightInCu

sliceExtraBits = SliceAlignAdj × Ceil(TransmissionDelayCu ÷ SliceWidthInCu)



图C.2 接口档次BBV数据输入和数据移出示意图

附 录 D
（资料性）
编码端参考方案

D.1 概述

本附录描述了编码端的参考方案，包括编码相关参数设置，见D.2；复杂度等级计算，见D.3；模式决策，见D.4。

D.2 编码相关参数设置

substream_expansion_ratio_log2的推荐值为4，rc_buffer_size的推荐值为8192比特，target_bpp的推荐值为（8<<4），可以根据位宽和图像格式进行调节。

transmission_delay_cu用于规定初始传输延迟的CU个数，transmission_delay_cu的推荐值为：

```
delayBits_min = (image_format == '000' ? 1 : 3) × (SubstreamSegmentSize – 1) + (((CuSize × TargetBpp) >> 4) – (CuSize << 1)) – 1 + ModeBits + (BitDepth[0] × CbLumaSize + BitDepth[MultiplexingEnableFlag ? 0 : 1 ] × CbChromaSize × 2
transmission_delay_cu = Min(Max(((RcBufferSize >> 1) ÷ ((CuSize × TargetBpp) >> 4)), ((delayBits_min + ((CuSize × TargetBpp) >> 4) – 1) ÷ ((CuSize × TargetBpp) >> 4))), (RcBufferSize ÷ ((CuSize × TargetBpp) >> 4)))
```

pwq_max_qp用于规定点预测模式量化参数调整的最大量化参数，即点预测模进行量化参数调整的量化参数上限值。pwq_max_qp的值越大，点预测模式越多的残差样本会被允许进行量化参数调整。推荐值为6。

jnd_qp以及strict_jnd_qp用于规定可察觉失真量化参阈值，后者应小于前者，后者用于对低复杂度的进一步保护，即点预测模式，块预测模式进行量化参数调整的量化参数的下限。jnd_qp或者strict_jnd_qp越大，越少的残差样本会被允许进行量化参数调整。jnd_qp的推荐值为2，strict_jnd_qp的推荐值为0。

qp_refine_th0以及qp_refine_th1用于规定进行量化参数调整的梯度阈值。bwq_complex_th用于规定进行子块量化参数调整的最大通道复杂度。其中，qp_refine_th1作用于点预测模式以及块预测模式，该值越大，越多的像素会进行量化参数调整，从而占用更多的比特；qp_refine_th0仅作用于块预测模式，配合bwq_complex_th，表示在bwq_complex_th复杂度限制下且满度梯度小于等于qp_refine_th0的像素进行强度更大的子块量化参数调整，qp_refine_th0的值越大，bwq_complex_th的值越大，越多的像素会进行量化参数调整，从而占用更多的比特。qp_refine_th0应小于等于qp_refine_th1。三者的推荐值为：

```
qp_refine_th0 = 1
qp_refine_th1 = 10
bwq_complex_th = 2
```

rc_qp_bias[compL][compC]的推荐值为：

```
rc_qp_bias = {{ 6  6  7  8  9 }
              { 6  6  6  7  8 }
              { 4  6  6  6  7 }
```

```
        {2  4  6  6  6}
        {0  2  4  6  6}}
```

rc_extra_buffer_decrease_step_log2用于调节Slice结尾处的额外缓冲区（ExtraBuffer）的覆盖范围，rc_extra_buffer_decrease_step_log2的值越小，Slice结尾处额外缓冲区覆盖范围越大，即更多的块通过额外缓冲区降低上溢风险。

rc_extra_buffer_penalty_log2_minus3 用于调节码率控制过程中的上溢风险大小。rc_extra_buffer_penalty_log2_minus3的值越小，上溢风险越小。理论上，随着上溢风险的降低，会造成性能的损失。

rc_extra_buffer_decrease_step_log2 以及rc_extra_buffer_penalty_log2_minus3的推荐值如下：

```
rc_extra_buffer_decrease_step_log2 = [Log(ExtraBufferSize ÷ EndControlBlocks)]
rc_extra_buffer_penalty_log2_minus3 = 1
```

如果只对额外缓冲区覆盖的编码单元进行上溢控制，可以将 rc_extra_buffer_decrease_step_log2 的值设置为 rc_decrease_step_log2 的值；否则，如果 Slice 结尾处无需上溢风险控制，可以将 rc_extra_buffer_decrease_step_log2 的值设置为：

```
rc_extra_buffer_decrease_step_log2 = [Log(ExtraBufferSize)]
```

rc_target_end_ratio_minus4 用于规定条带结尾处的目标满度。rc_target_end_ratio_minus4 的值越大，结尾处的码控力度越大，主观质量受影响的趋势越大。rc_target_end_ratio_minus4 的推荐值为 2。

rc_ratio0 用于调节下溢控制的强度，值越大当码流缓冲区充足时将以更强的力度进行下溢控制，其推荐值为 160，精度为 1/16。

rc_param0 由码流缓冲区充足时的满度阈值确定，其推荐值为 192，可以通过以下方法设置：

——确定一个范围为在 0 到 1 之间的低满度阈值 rcFullnessLow，推荐值对应 0.15；

——将 rc_param0 设置为 rc_ratio0×8×rcFullnessLow 四舍五入后的值。

rc_ratio1 用于调节上溢控制的强度，值越大当码流缓冲区紧张时将以更强的力度进行上溢控制，其推荐值为 457，精度为 1/32。

rc_param1 由码流缓冲区紧张时的满度阈值确定，其推荐值为 1554，可以通过以下方法设置：

——确定一个范围为在 0 到 1 之间的高满度阈值 rcFullnessHigh，推荐值对应 0.85；

——将 rc_param1 设置为 rc_ratio1×4×rcFullnessHigh 四舍五入后的值。

rc_ratio2 用于规定 relativeLosslessBits 不变时的最大目标比特钳位函数的斜率，其推荐值为 1083。

rc_param2 用于规定 relativeLosslessBits 为 0 时最大目标比特钳位函数的截距，其推荐值为 1121。

其中最大目标比特钳位函数为：

```
–rc_ratio2 × fullness + rc_param2 – (1 + ChromaSampleRate) × relativeLosslessBits
```

其中 fullness 是码控满度，relativeLosslessBits 为 Clip3(0,RcMaxRelativeBits, RcMaxLosslessBits - losslessBits)，rc_ratio2 和 rc_param2 可以通过以下方法设置：

a) 确定一个低满度 fullnessLow 和一个高满度值 fullnessHigh（如 0.3 和 0.8），

b) 设置低满度下的 bpp 调节力度 bppOffsetLow 和高满度下的 bpp 调节力度 bppOffsetHigh（如 2.0733×formatRatio 和 0.6633×formatRatio），将其作为 relativeLosslessBits 为 0 时的函数值求解 rc_ratio2 和 rc_param2。formatRatio 可设置为 3 也可根据采样格式进行调整，例如：

1） 针对 YUV444，formatRatio 取值 3。

2） 针对 YUV420，formatRatio 取值 1.5。

rc_max_relative_bits 用于规定 relative_lossless_bits 的最大值，进行最大值钳位时，如果编码块的预测无损比特数 rc_lossless_bits[comp]与 rc_max_lossless_bits 差距超过此值，则按照 rc_max_relative_bits<<1 进行钳位操作，其推荐值为 160。

rc_lossless_bits[comp]用于初始化不同复杂度等级的预测无损比特数，影响对应复杂度等级前几个编码块的编码质量，值越小编码质量越好缓冲区压力更大，应略小于对应复杂度等级的统计值。

rc_avg_lossless_bits 用于初始化平均无损比特数，更小的值会使图像开始处的质量提升，但是缓冲区压力增大，反之亦反。

rc_bits_offset 用于偏置 rc_lossless_bits[comp]和 rc_avg_lossless_bits 以使得对不同位宽和格式的图像获得一致的压缩质量。

rc_max_lossless_bits 用于规定图像的最大无损比特数，取平均意义上无损比特数的最大值。

rc_lossless_bits[comp]，rc_avg_lossless_bits，rc_bits_offset 以及 rc_max_lossless_bits 的推荐值根据 BitDepth[0]和 ImageFormat 确定。

BitDepth[0]	8bit		10bit		12bit		14bit		16bit	
ImageFormat	RGB444	其他	RGB444	其他	RGB444	其他	RGB444	其他	RGB444	其他
rc_lossless_bits[0]	128	128	744	674	873	805	1001	936	1129	1064
rc_lossless_bits[1]	595	580	815	818	1097	1037	1225	1165	1353	1293
rc_lossless_bits[2]	682	680	929	910	1159	1173	1287	1301	1415	1429
rc_lossless_bits[3]	791	770	1008	976	1271	1267	1399	1395	1527	1523
rc_lossless_bits[4]	900	850	1060	1038	1339	1316	1467	1444	1595	1572
rc_avg_lossless_bits	704	640	896	832	1024	960	1152	1088	1216	1216
rc_bits_offset	64	0	320	256	448	384	640	576	768	768
rc_max_lossless_bits	1050	922	1152	1088	1280	1216	1536	1472	1728	1728

rc_complex_th 用于规定码率控制过程中进行主观质量保护的复杂度阈值。rc_complex_th 的值越大，越多复杂度等级的编码单元可以进行主观质量保护，但同时越多的位会被占用。rc_complex_th 的推荐值为 3。

rc_relative_th 用于规定码率控制过程中进行主观质量保护的相对无损编码位数的阈值。rc_relative_th 的值越小，越多的编码单元可以进行主观质量保护。rc_relative_th 值应小于等于 rc_max_relative_bits<<1 的值，rc_relative_th 的推荐值为 190。

D.3 复杂度等级计算

D.3.1 概述

复杂度等级计算需要分别计算纹理复杂度等级，见D.3.2，和IBC复杂度等级，见D.3.3，对于Slice首个CU的情况，只进行纹理复杂度等级计算，将亮度纹理复杂度等级作为亮度复杂度等级，色度纹理复杂度等级作为色度复杂度等级（此时，为了进行首个CU的保护，可以将复杂度等级钳位在2及以下）；对于非Slice首个CU的情况，如果纹理复杂度等级计算得到的编码单元复杂度等级小于或等于IBC复杂度等级，则将亮度纹理复杂度等级作为亮度复杂度等级，色度纹理复杂度等级作为色度复杂度等级；否则，将IBC复杂度等级作为亮度复杂度等级和色度复杂度等级。

其中，编码单元复杂度等级cuComplexityLevel的计算方式为：

```
if (image_format == '000') {    /* YUV400 */
    cuComplexityLevel = complexityLevel[0]
} else if (image_format == '001') {    /* YUV420 */
    cuComplexityLevel = ComplexityDivide3Table[complexityLevel[1] + (complexityLevel[0] << 1)]
} else if (image_format == '010') {    /* YUV422 */
    cuComplexityLevel = (complexityLevel[0] + ComplexityLevel[1]) >> 1
} else if (image_format == '011' || image_format == '100') {    /* YUV444 or RGB444 */
    cuComplexityLevel = ComplexityDivide3Table[complexityLevel[0] + (complexityLevel[1] << 1)]
}
```

其中，complexityLevel[0]表示亮度通道纹理复杂度，complexityLevel[1]表示色度通道纹理复杂度。

D.3.2 纹理复杂度等级计算

D.3.2.1 概述

计算纹理复杂度等级会用到当前编码单元上一行样本的重建样值、左边一列样本的原始值以及当前编码单元的原始值。对于YUV400图像格式，只需要计算亮度一个通道的通道级纹理复杂度等级；对于其他图像格式，亮度纹理复杂度等级为亮度通道的纹理复杂度等级，色度纹理复杂度等级为两个色度通道的纹理复杂度等级的平均值。

通道级纹理复杂度等级是独立计算的，对于每个通道，先将当前编码块划分为多个宽为4高为CbHeight[component]子块，然后分别计算每个子块的水平纹理信息（textureH）、竖直纹理信息（textureV）、斜向纹理信息0以及斜向纹理信息1，见D.3.2.2；然后，对于每个子块，取textureH以及textureV中的最小值作为子块纹理信息输入，得到的子块纹理复杂度等级作为子块非斜向纹理复杂度等级（TextureCompSbNd），见D.3.2.3；将当前编码块各子块的TextureCompSbNd作为子块纹理复杂度等级输入，得到的通道纹理复杂度等级作为当前编码块的非斜向纹理复杂度等级（TextureCompCbNd），见D.3.2.4；将TextureD0作为子块纹理信息输入，得到的子块纹理复杂度等级作为子块斜向纹理复杂度等级0（TextureCompSbD0），将TextureD1作为子块纹理信息输入，得到的子块纹理复杂度等级作为子块斜向纹理复杂度等级1（TextureCompSbD1），见D.3.2.3；将当前编码块各子块的TextureCompSbD0作为子块纹理复杂度等级输入，得到的通道纹理复杂度等级作为当前编码块的斜向纹理复杂度等级0（TextureCompCbD0），将当前编码块各子块的TextureCompSbD1作为子块纹理复杂度等级输入，得到的通道纹理复杂度等级作为当前编码块的斜向纹理复杂度等级1（TextureCompCbD1），见D.3.2.4；最后，根据当前编码块的非斜向纹理复杂度等级（TextureCompCbNd），当前编码块的斜向纹理复杂度等级0（TextureCompCbD0）以及当前编码块的斜向纹理复杂度等级1（TextureCompCbD1）导出当前编码块的纹理复杂度等级以及各子块的纹理复杂度等级，见D.3.2.5。

D.3.2.2 子块纹理信息计算

水平纹理信息（textureH）的计算方式为：当前子块每个样本位置的原始值与该样本左侧相邻样本位置的原始值的差的绝对值之和。

竖直纹理信息（textureV）的计算方式为：当前子块每个样本位置的原始值与该样本上侧相邻样本位置的重建样值或原始值的差的绝对值之和。其中，若上侧相邻样本位置在当前子块外，则取其重建样值；否则，取其原始值。

斜向纹理信息0（textureD0）的计算方式为：当前子块每个样本位置的原始值与该样本左上侧相邻样本位置的重建样值或原始值的差的绝对值之和。其中，若左上侧相邻样本位置在当前子块外，则取其重建样值；否则，取其原始值。

斜向纹理信息1（textureD1）的计算方式为：当前子块每个样本位置的原始值与该样本右上侧相邻样本位置的重建样值或原始值的差的绝对值之和。其中，若右上侧相邻样本位置在当前子块外，则取其重建样值；否则，取其原始值。

若相邻样本位置“不存在”，则跳过当前样本位置的计算。

D.3.2.3 导出子块纹理复杂度等级

根据子块纹理信息（textureSb）导出判断子块纹理复杂度等级（textureCompSb）具体过程为：

a) 如果 textureSb 小于 textureThreshold1，则当前子块的 textureCompSb 等于 0；

b) 否则，如果 textureSb 大于 textureThreshold2，则当前子块的 textureCompSb 等于 2；

c) 否则，则当前子块的 textureCompSb 等于 1。

其中，textureThreshold1 以及 textureThreshold2 的值为：

$$textureThreshold1 = 4 \times CbHeight[component] \times 2 \times (1 << (BitDepth[0] - 8))$$

$$textureThreshold2 = 4 \times CbHeight[component] \times 6 \times (1 << (BitDepth[0] - 8))$$

D.3.2.4 导出待选通道纹理复杂度等级

a) 根据子块纹理复杂度等级（textureCompSb）确定当前通道的待选纹理复杂度等级（0，1，2，3和4），亮度纹理复杂度等级的导出方式为：

1) 如果，有 2 个子块的 textureCompSb 值为 0，或者，有 1 个子块的 textureCompSb 值为 0 且剩余 3 个子块的 textureCompSb 值为 1，则，

——如果有 2 个相邻子块的 textureCompSb 值同时为 0 且剩余 2 个子块的 textureCompSb 值不同时为 2，则亮度通道纹理复杂度等级为 0；

——否则，亮度通道纹理复杂度等级为 1。

2) 否则，有 2 个相邻子块的 textureCompSb 值同时为 2，则亮度通道纹理复杂度等级为 3；

3) 否则，如果 3 个子块的 textureCompSb 值为 2，则亮度通道纹理复杂度等级为 4；

4) 否则，亮度通道纹理复杂度等级为 2。

b) 色度通道纹理复杂度等级的导出方式为：

1) 若当前图像格式为 YUV422 或者 YUV420，则色度通道纹理复杂度等级为两个子块的 textureCompSb 值之和；

2) 否则，如果有 2 个子块的 textureCompSb 值同时为 0 且剩余 2 个子块的 textureCompSb 值不同时为 2，则，

——如果有 3 个子块的 textureCompSb 值同时为 0，或者，有 2 个相邻子块的 textureCompSb 值同时为 0 且剩余 2 个子块的 textureCompSb 值均不为 2，则色度通道纹理复杂度等级为 0；

——否则，色度通道纹理复杂度等级为 1。

3) 否则，如果有 2 个子块的 textureCompSb 值同时为 2，或者，有 3 个子块的 textureCompSb 值同时为 1 且剩余子块的 textureCompSb 值为 2，则，

——如果有 3 个子块的 textureCompSb 值同时为 2，或者，有 2 个相邻子块的 textureCompSb 值同时为 2 且剩余 2 个子块的 textureCompSb 值均不为 0，则色度通道纹理复杂度等级为 4；

——否则，色度通道纹理复杂度等级为 3。

4) 否则，色度通道纹理复杂度等级为 2。

D.3.2.5 导出最终子块纹理复杂度等级以及通道级纹理复杂度等级

a) 根据当前编码块的非斜向纹理复杂度等级（TextureCompCbNd），当前编码块的斜向纹理复杂度等级0（TextureCompCbD0）以及当前编码块的斜向纹理复杂度等级1（TextureCompCbD1）导出当前编码块的纹理复杂度等级，具体过程为：

1) 如果 TextureCompCbNd 小于等于 TextureCompCbD0 且 TextureCompCbNd 小于等于 TextureCompCbD1，则最终通道级纹理复杂度等级选择非斜向纹理复杂度等级，即 TextureComp 等于 TextureCompCbNd；

2) 否则，如果 TextureCompCbD0 小于 TextureCompCbD1，则最终通道级纹理复杂度等级选择斜向纹理复杂度等级 0，即 TextureComp 等于 TextureCompCbD0；

3) 否则，最终通道级纹理复杂度等级选择斜向纹理复杂度等级 1，即 TextureComp 等于 TextureCompCbD1。

b) 各子块的纹理复杂度等级根据最终选择的纹理复杂度等级确认，具体过程为：

1) 如果选择非斜向纹理复杂度等级，则将 TextureCompSbNd 作为最终子块纹理复杂度等级 TextureCompSb；

2) 否则，如果选择斜向纹理复杂度等级 0，则将 TextureCompSbD0 作为最终子块纹理复杂度等级 TextureCompSb；

3) 否则，将 TextureCompSbD1 作为最终子块纹理复杂度等级 TextureCompSb。

D.3.3 IBC复杂度等级计算

D.3.3.1 概述

计算IBC复杂度等级会用到当前编码单元上一行样本的重建样值、左侧搜索区域样本的原始值以及当前编码单元的原始值。对于图像格式为YUV400的情况，只需要根据亮度通道的IBC相似预测样本矩阵导出当前编码块的IBC复杂度等级，亮度IBC复杂度等级即为当前编码块的IBC复杂度等级；对于其他图像格式，由三通道的IBC相似预测样本矩阵一起导出当前编码块的IBC复杂度等级，亮度IBC复杂度等级和色度IBC复杂度等级都为当前编码块的IBC复杂度等级。其中，IBC相似预测样本矩阵由相似IBC模式预测得到，相似IBC模式为：在使用当前编码块外的样值时，若为上侧样本位置（包括当前编码块左上角样本位置），则使用上侧样本位置的重建值进行IBC预测，若为左侧样本位置（不包括当前编码块左上角样本位置），则使用左侧样本位置的原始值进行IBC预测。

当前编码块的IBC复杂度等级是需要先计算子单元的SAD值，子单元的划分与编码单元位置相关，见D.3.3.2；然后，根据各子单元的SAD值以及预设阈值判断子单元的IBC复杂度等级（IbcCompSu），见D.3.3.3；最后根据IbcCompSu导出当前编码块的IBC复杂度等级，见D.3.3.4。

D.3.3.2 子单元SAD值计算

如果当前编码单元位于Slice非首列，将当前编码单元划分为4个宽为CbWidth[0]>>2高为CbHeight[0]的子单元；否则，如果当前编码单元位于Slice首列非首行，当前编码单元划分为（CbHeight[component]×2)个宽为(CbWidth[0]>>1)高为1的子单元。按每个子单元计算SAD值(SadSu)。子单元SAD值（SadSu）由当前编码单元的IBC相似预测样本矩阵和原始值矩阵计算得到。其中，进行IBC搜索的像素区域均为原始值，搜索准则见D.4.3.2.2。

D.3.3.3 判断子单元IBC复杂度等级

根据SadSu以及预设阈值判断子单元IBC复杂度等级（IbcCompSu）具体过程为：

a) 如果 SadSu 小于 ibcThreshold1，则当前子单元的 IbcCompSu 等于 0；

b) 否则，如果 SadSb 小于 ibcThreshold2，则当前子单元的 IbcCompSu 等于 1；
c) 否则，如果 SadSb 小于 ibcThreshold3，则当前子单元的 IbcCompSu 等于 2；
d) 否则，当前子单元的 IbcCompSu 等于 3。

其中，ibcThreshold1、ibcThreshold2 以及 ibcThreshold3 的值为：

$$ibcThreshold1 = 3 \times 4 \times CbHeight[component] \times 2 \times (1 << (BitDepth[0] - 8))$$
$$ibcThreshold2 = 3 \times 4 \times CbHeight[component] \times 6 \times (1 << (BitDepth[0] - 8))$$
$$ibcThreshold3 = 3 \times 4 \times CbHeight[component] \times 10 \times (1 << (BitDepth[0] - 8))$$

D.3.3.4 导出IBC复杂度等级

根据D.3.3.3 得到的子单元IBC复杂度等级（IbcCompSu）确定当前通道的IBC复杂度等级（0、1、2、3和4），导出方式为：

当编码块位于Slice非首列或位于Slice首列非首行时：

a) 如果，4 个子单元的 IbcCompSu 值都为 0，则当前编码块的 IBC 复杂度等级为 0；
b) 否则，如果有 3 个子单元的 IbcCompSu 值为 0，则；
   1) 如果其余子单元的 IbcCompSu 值为 1，则当前编码块的 IBC 复杂度等级为 1；
   2) 否则，如果其余子单元的 IbcCompSu 值为 2，则当前编码块的 IBC 复杂度等级为 2；
   3) 否则，当前编码单元的 IBC 复杂度等级为 3。
c) 否则，如果有 2 个子单元的 IbcCompSu 值或 1 个子单元的 IbcCompSu 值为 0，则：
   1) 如果其余子单元的 IbcCompSu 值都为 1，则当前编码块的 IBC 复杂度等级为 2；
   2) 否则，如果其余子单元的 IbcCompSu 值都为 2，则当前编码块的 IBC 复杂度等级为 3；
   3) 否则，当前编码单元的 IBC 复杂度等级为 4。

D.4 模式决策

D.4.1 概述

模式决策分为编码块级的点预测模式，帧内预测模式和样本值模式决策，见D.4.2。当编码块级决策结束后，进行编码单元级的块复制帧内预测模式和回退模式决策，见D.4.3。

D.4.2 编码块级模式决策

D.4.2.1 概述

编码块级模式决策共会进行最多两次RDO，其中一次RDO为点预测模式，见D.4.2.2，一次RDO为普通帧内预测模式（见D.4.2.3）。其中，若判定当前编码单元需要启用差值预测模式，见D.4.2.4，当前编码单元亮度通道的点预测模式无需进行RDO。样本值模式决策用于防止预测膨胀，即预测模式比特数大于直接传输样本值的传输比特数，见D.4.2.8。

D.4.2.2 点预测模式决策

如果判定当前编码单元需要使能差值预测模式，则当前编码单元亮度通道不进行点预测模式决策，直接进行普通帧内预测模式决策；否则，点预测模式直接进入编码块级RDO决策，见D.4.4，其中失真度量为当前编码块的重建样值与原值值之间的SAD值，比特度量为当前编码块最佳点预测模式的比特代价。

D.4.2.3 普通帧内预测模式决策

在进行普通帧内预测模式决策前，需要决策子块量化参数调整技术是否开启，见D.4.2.6，普通帧内预测模式决策先通过当前编码块相似预测样本矩阵和原始值矩阵的SAD值(其中，按照D.3.2.2和得到的子块纹理复杂度等级TextureCompSb。如果当前编码块中存在TextureCompSb等于0，使用TextureCompSb对对应子块的SAD计算进行调整；否则，根据D.4.2.7计算宽为8高为1子块的复杂度等级TextureCompSbAng1，如果当前编码块中存在TextureCompSbAng1等于0，使用TextureCompSbAng1对对应子块的SAD计算进行调整；否则，使用TextureCompSb对对应子块的SAD计算进行调整。如果使用到的TextureCompSb或TextureCompSbAng1等于0，则将SAD值放大四倍；否则，如果使用的TextureCompSb或TextureCompSbAng1等于2，则将SAD值缩小四倍；否则，SAD值不变)以及普通帧内预测模式对应的pred_mode比特数之和的最小值确定最佳普通帧内预测模式，而后最佳普通帧内预测模式根据差值预测模式的启用判定进行子块差值预测值计算见D.4.2.5，或子块重建补偿值计算，见D.4.2.7，最后进入编码块级RDO决策，见D.4.4，其中失真度量为当前编码块的重建样值与原值值之间的SAD值，比特度量为当前编码块最佳普通帧内预测模式的比特代价。其中，相似预测样本矩阵在使用当前编码块外的样值时，若为上侧样本位置（包括当前编码块左上角样本位置），则使用上侧样本位置的重建值进行预测，若为左侧样本位置（不包括当前编码块左上角样本位置），则使用左侧样本位置的原始值进行预测。

D.4.2.4 普通帧内预测模式子块差值预测启用判定

根据编码单元所有通道对应子块的平坦度CompSbFlat、不相关度CompSbRelevant以及原始样本值计算差值预测值：

计算当前编码单元每个通道每个宽为(CbWidth[component]>>2)高为CbHeight[component]子块的平坦度CompSbFlat[component]:子块内每个样本减去子块样本均值的绝对值之和小于2×(1 << (BitDepth[0]-8))×(CbWidth[component]>>2)×CbHeight[component]，则为子块为平坦，CompSbFlat[component]等于0，否则子块为不平坦，CompSbFlat[component]等于2；

计算当前编码单元每个通道每个宽为(CbWidth[component]>>2)高为CbHeight[component]子块的不相关度CompSbRelevant[component]（即与参考像素的不相关度）：

a) 竖直相关信息（RelevanV）的计算方式为：子块内每个样本与编码块上一行的相同水平坐标位置处的参考样本的重建值的差的绝对值之和，若相邻样本位置“不存在”，则相邻样本位置使用默认值(1<<BitDepth[component])-1 代替重建样本值进行计算；

b) 水平相关信息（RelevanH）的计算方式为：子块内每个样本每个样本与编码块左侧的相同垂直坐标位置处的参考样本的原始值的差的绝对值之和，若相邻样本位置“不存在”，则相邻样本位置使用默认值(1<<BitDepth[component])-1 代替重建样本值进行计算；

c) 取 RelevanV 和 RelevanH 的最小值，若最小值小于 4×(1 << (BitDepth[0]-8))×(CbWidth[component]>>2)×CbHeight[component] 则不相关度 CompSbRelevant[component] 为 0；若最小值大于 12×(1 << (BitDepth[0]-8))×(CbWidth[component]>>2)×CbHeight[component]，则不相关度 CompSbRelevant[component]为 2；否则，不相关度 CompSbRelevant[component]为 1。

如果当前编码单元所有通道对应子块的平坦度都为平坦、并且所有通道对应子块的不相关度之和小于等于4或亮度通道对应子块的不相关度等于2的情况，则当前编码单元启用编码单元差值预测（cu_dp_intra_flag置1），否则关闭差值预测（cu_dp_intra_flag置0）。

D.4.2.5 普通帧内预测模式子块差值预测值计算

当前编码单元启用编码单元差值预测时，子块差值预测值计算过程如下：

a) 如果当前编码单元启用编码单元差值预测，进一步根据 CompSbFlat 确定子块是否进行差值预测：如果所有通道对应子块的 CompSbFlat[component]都为 0，则所有通道对应子块在子块所

在编码块选择普通帧内模式时进行差值预测；否则，所有通道对应子块都不进行差值预测。差值预测方式为：计算每个通道对应子块的样本均值 sdp_offset[component]，将sdp_offset[component]作为该通道对应子块的每个样本的预测值。

b) 如果亮度通道对应子块使用差值预测，则将亮度通道对应子块的预测残差截断在(-3<<Max(0,BitDepth[0]-10))至(4<<Max(0,BitDepth[0]-10))内（包括两个端点），注意此时亮度通道对应子块的量化参数为(Max(0,BitDepth[0]-10))；

c) 如果存在色度通道，则将色度通道对应子块的预测残差截断在(0<<Max(0,BitDepth[0]-10))至(1<<Max(0,BitDepth[0]-10))（包括两个端点），注意此时色度通道对应子块的量化参数为(Max(0,BitDepth[0]-10))。

D.4.2.6 普通帧内预测模式子块量化参数调整技术启用决策

根据子块纹理复杂度等级TextureCompSb以及当前编码块的重建值确定子块量化参数调整技术是否启用：

a) 如果当前编码块中存在TextureCompSb等于0或判定当前编码单元需要使能差值预测模式的情况，则开启子块量化参数调整，即bwq_flag[component]置1；

b) 否则，检查当前编码块每个宽为2高为2子块的平坦度（子块内每个样本减去2x2左上角样本的绝对值之和小于2×(1 << (BitDepth-8))×3则为平坦，否则为不平坦）

1) 如果存在一个宽为2高为2子块平坦的情况，则开启子块量化参数调整（bwq_flag[component]置1）；

2) 否则，关闭子块量化参数调整（bwq_flag[component]置0）。

D.4.2.7 普通帧内预测模式子块重建补偿值计算

根据最佳普通帧内预测模式，子块纹理复杂度等级TextureCompSb以及重建样本值计算补偿值：

a) 如果最佳普通帧内预测模式为水平预测模式（INTRA_MODE_ANG_1），则

1) 计算宽为8高为1的子块的复杂度等级TextureCompSbAng1，TextureCompSbAng1是根据子块每个样本位置的原始值与该样本左侧相邻样本位置的原始值的差的绝对值之和得到的。

——如果 textureSbAng1 小于 8× 2 × (1 << (BitDepth[0] - 8))，则当前子块的TextureCompSbAng1等于0；

——否则，如果 textureSbAng1 大于 8× 6 × (1 << (BitDepth[0] - 8))，则当前子块的TextureCompSbAng1b等于2；

——否则，则当前子块的TextureCompSbAng1等于1。

2) 如果当前编码块中不存在TextureCompSbAng1等于0或判定当前编码单元需要使能差值预测模式的情况，则关闭当前编码块的子块重建补偿技术（brc_flag[component]置0）；

3) 否则，开启当前编码块的子块重建补偿技术（brc_flag[component]置1）：

——计算宽为8高为1的补偿子块重建后每个样本的量化误差；

——将量化误差截断在-16至16内（包括-16以及16），计算该补偿子块的量化误差平均值diffAvg；

——根据TextureCompSb Ang1确定是否补偿：

- 如果当前补偿子块的 TextureCompSbAng1 等于 0，该补偿子块的补偿值为diffAvg；
- 否则，该补偿子块不进行重建补偿，即diffAvg的值为0。

b) 否则，

1) 如果当前编码块中不存在TextureCompSb等于0或判定当前编码单元需要使能差值预测模式的情况，则关闭当前编码块的子块重建补偿技术（brc_flag[component]置0）；
2) 否则，开启当前编码块的子块重建补偿技术（brc_flag[component]置1）：
——计算宽为4高为2的补偿子块重建后每个样本的量化误差；
——将量化误差截断在-16至16内（包括-16以及16），计算该补偿子块的量化误差平均值diffAvg；
——根据TextureCompSb确定是否补偿：
- 如果当前补偿子块的TextureCompSb等于0，则该补偿子块的补偿值为diffAvg；
- 否则，该补偿子块不进行重建补偿，即diffAvg的值为0。

D.4.2.8 样本值模式决策

在点预测模式决策以及普通帧内预测模式决策后，若Allowfallback的值为0且模式比特代价大于等于SampleCost[component]，则当前编码块被强制修改为样本值模式。

D.4.3 编码单元级模式决策

D.4.3.1 概述

编码单元级模式决策为一次块复制帧内预测模式RDO，见D.4.3.2 ，在编码单元级模式决策时，如果当前编码单元在块复制帧内预测模式下存在某个通道编码块的模式比特代价大于SampleCost[component]，则当前编码单元各编码块的最佳预测模式组合即为当前编码单元在编码单元级模式决策前的组合。回退模式决策用于防止码控缓存上溢，见D.4.3.3 。

D.4.3.2 块复制帧内预测模式决策

D.4.3.2.1 概述

块复制帧内预测模式决策先通过搜索确定块矢量，其搜索准则为最小化所有通道的SAD值的加权和，见D.4.3.2.2，然后与额外模式比较再次确定块矢量，其比较准则视额外模式而定，见D.4.3.2.3。如果当前编码块的尺寸是宽为16高为2，当前编码块位于条带非首列时，搜索的块矢量最小为1，最大为30。额外模式为两种，块矢量0表示的角度预测模式0和块矢量31表示的差值预测；当前编码块位于条带首列但非首行时，搜索的块矢量最小0，块矢量最大为15，额外模式为块矢量31表示的差值预测。在搜索到确定最佳块复制帧内预测模式后，进入子块级额外模式开启决策。再次确定块矢量后，进入编码单元级RDO决策，见D.4.4，其中失真度量为当前编码单元各通道重建样值矩阵与原始值矩阵之间的SAD值的加权和，见D.4.3.2.4，比特度量为当前编码单元最佳块复制帧内预测模式的比特代价。

D.4.3.2.2 块复制帧内预测模式搜索SAD计算

——如果当前图像格式为RGB444且BitDepth[0]不等于16，则SAD值的加权值（cuSADs）为：

$$cuSADs = cbSADs[0] + (cbSADs[1] >> 1) + (cbSADs[2] >> 1)$$

——否则，SAD值的加权值为所有通道cbSADs[component]的平均值。

其中，cbSADs[component]为component通道的编码块搜索得到的SAD值。

D.4.3.2.3 额外模式开启决策

a) 如果当前编码块位于条带非首列，当前编码块的尺寸是宽为16高为2，对于每个宽为2高为2的子块：

1） 如果最佳搜索的cbSADs加2后的值大于角度预测模式0的SAD，SAD计算见D.4.3.2.2 ，并且角度预测模式0的SAD小于3 × 4 × 20 × (1 << MAX(BitDepth[0] - 8, 0))，则块矢量置0，子块预测值通过竖直预测得到；

2） 否则，如果最佳搜索的cbSADs大于3 × 4 × 20 × (1 << MAX(BitDepth[0] - 8, 0))，并且角度预测模式0的SAD大于等于3 × 4 × 20 × (1 << MAX(BitDepth[0] - 8, 0))，并且当前子块为平坦子块（子块内每个样本减去2x2左上角样本的绝对值之和小于2×(1 << (BitDepth[0]-8))×3则为平坦，否则为不平坦），则子块预测值通过差值预测（子块每个样本的预测值为子块样本的均值）得到；

3） 否则，子块预测值依旧通过搜索得到。

b） 如果当前编码块位于条带首列但 首行，当前编码块的尺寸是宽为16高为2，对于每个宽为4高为1的子块：

1） 如果最佳搜索的SAD大于3 × 4 × 20 × (1 << MAX(BitDepth[0] - 8, 0))，并且当前子块为平坦子块（子块内每个样本减去子块样本均值的绝对值之和小于 2×(1 << (BitDepth[0]-8))×4则为平坦，否则为不平坦），则子块预测值通过差值预测（子块每个样本的预测值为子块样本的均值）得到；

2） 否则，子块预测值依旧通过搜索得到。

D.4.3.2.4 块复制帧内预测模式RDO失真度量计算

——如果当前图像格式为RGB444且BitDepth[0]不等于16，则失真度量（D）为：

```
D = (cbSAD[0] << 1) + cbSAD[1] + cbSAD[2]
```

——否则，失真度量为所有通道cbSAD[component]的平均值。

其中，cbSAD[component]为component通道的编码块计算得到的SAD值。

D.4.3.3 回退模式决策

如果Allowfallback的值为1，则判断按当前编码单元最佳预测模式进行编码的总比特数是否会造成码控缓存上溢。其中，总比特数cuBits的计算公式为：

```
if (image_format == '000') {    /* 1 substream for YUV400 */
    substreamExpansionBits = 0
} else {    /* 3 substreams for other cases */
    cuSubstreamBitsMax = Max(cuSubstreamBits[0], Max(cuSubstreamBits[1], cuSubstreamBits[2]))
    stuffBits[0] = (cuSubstreamBitsMax + SubstreamExpansionRatio – 1) / SubstreamExpansionRatio – cuSubstreamBits[0]
    stuffBits[1] = (cuSubstreamBitsMax + SubstreamExpansionRatio – 1) / SubstreamExpansionRatio – cuSubstreamBits[1]
    stuffBits[2] = (cuSubstreamBitsMax + SubstreamExpansionRatio – 1) / SubstreamExpansionRatio – cuSubstreamBits[2]
    stuffBits[0] = stuffBits[0] < 0 ? 0 : stuffBits[0]
    stuffBits[1] = stuffBits[1] < 0 ? 0 : stuffBits[1]
    stuffBits[2] = stuffBits[2] < 0 ? 0 : stuffBits[2]
    substreamExpansionBits = stuffBits[0] + stuffBits[1] + stuffBits[2]
}
cuBits = bestModeBits + substreamExpansionBits + MultiplexingEnableFlag ? 6 : 0
```

其中cuSubstreamBits[i]代表最佳预测模式在第（i+1）个子流上的总比特数，bestModeBits代表最佳预测模式的编码比特数。

——如果会发生上溢，则当前编码单元被强制修改为回退模式；

——否则，如果当前档次为接口档次且当前编码单元的某一编码块的模式比特数大于（SubstreamSegmentSize-2），则当前编码单元被强制修改为回退模式；

——否则，当前编码单元各编码块的最佳预测模式组合即为当前编码单元的最终预测模式。

如果当前编码单元被强制修改为回退模式，则计算两种回退模式的各通道重建样值矩阵和原始值矩阵的SAD之和，选取SAD之和小的回退模式作为当前编码单元的最终预测模式。

D.4.4 RDO决策准则

RDO模式决策准则是最小化RD代价，RD代价的计算式为：

$$D + \lambda R$$

其中，D为失真度量，R为比特度量，λ的计算如下，LambdaTable的定义见表D.1 ：

$$index = Clip3(0, 16, (Fullness + 4) >> 3)$$

$$scale = 23 \times LambdaTable[index]$$

$$shift = 12 - Qp[0]$$

$$\lambda = (scale + (1 << (shift - 1))) >> shift$$

表 D.1 LambdaTable 的定义

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LambdaTable[index]	16	17	19	21	23	25	27	29	32	35	38	41	45	49	54	59	64

附 录 E
（规范性）
图像填补参考方案

E.1 概述

本附录描述了两种图像填补的参考方案，当当前图像宽度不是条带宽度的整数倍或者当前图像高度不是条带高度的整数倍时，需要进行图像填补操作。重建图像或解码图像在输出前要进行裁剪，最终输出原始分辨率的图像。编码器应至少支持图像填补方案一。

E.2 图像填补方案一（padding_type_flag等于1）

如图E.1所示，其中右侧的填补方式为复制左侧的最近有效像素列，下侧的填补方式为默认值填补，默认值为(1<<BitDepth[component])-1。在方案一下，可以通过padding_stuff_flag使能对填补区域的编码单元的比特填充操作，如果(ImageWidth-InputImageWidth)大于等于(PictureWidthInSlice*3/4)，则推荐开启图像填补方案一用以解决图像填补带来的条带间主观质量不均衡的问题。

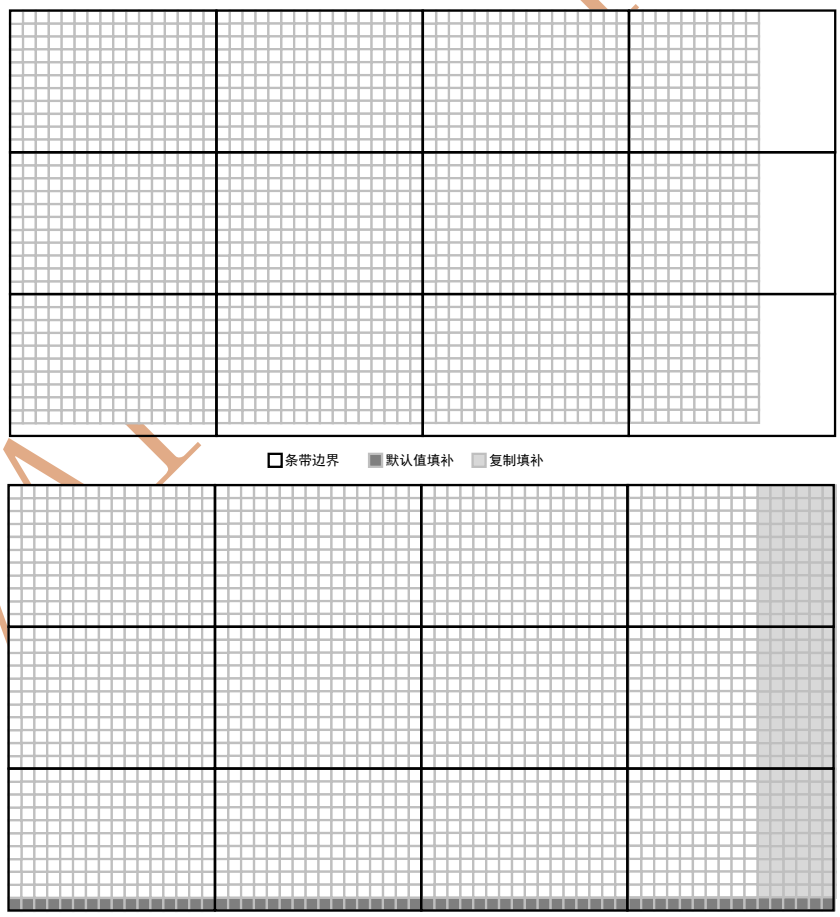


图 E.1 图像填补方式一示意图

具体的，纵向的填补方式为：

——如果是亮度通道，则最后一行条带需要在下边界填补(ImageHeight－InputImageHeight)行亮度样本；

——否则，如果图像格式为 RGB444，YUV444 或者 YUV422，则最后一行条带需要在下边界填补(ImageHeight － InputImageHeight)行色度样本；

——否则，如果图像格式为 YUV420，则最后一行条带需要在下边界填补((ImageHeight － InputImageHeight)>>1)行色度样本。

横向的填补方式为：

——如果是亮度通道，则最后一列条带需要在右边界填补 m 列亮度样本；

——如果是色度通道且图像格式为 RGB444 或者 YUV444，则最后一列条带需要在右边界填补 ImageWidth － InputImageWidth 列色度样本；

——如果是色度通道且图像格式为 YUV422 或者 YUV420，则最后一列条带需要在右边界补(ImageWidth － InputImageWidth)>>1 列色度样本。

E.3 图像填补方案二（padding_type_flag等于 0）

如图E.2所示，其中右侧的填补方式为复制左侧的最近有效像素列，下侧的填补方式为默认值填补，默认值为(1<<BitDepth[component])-1。

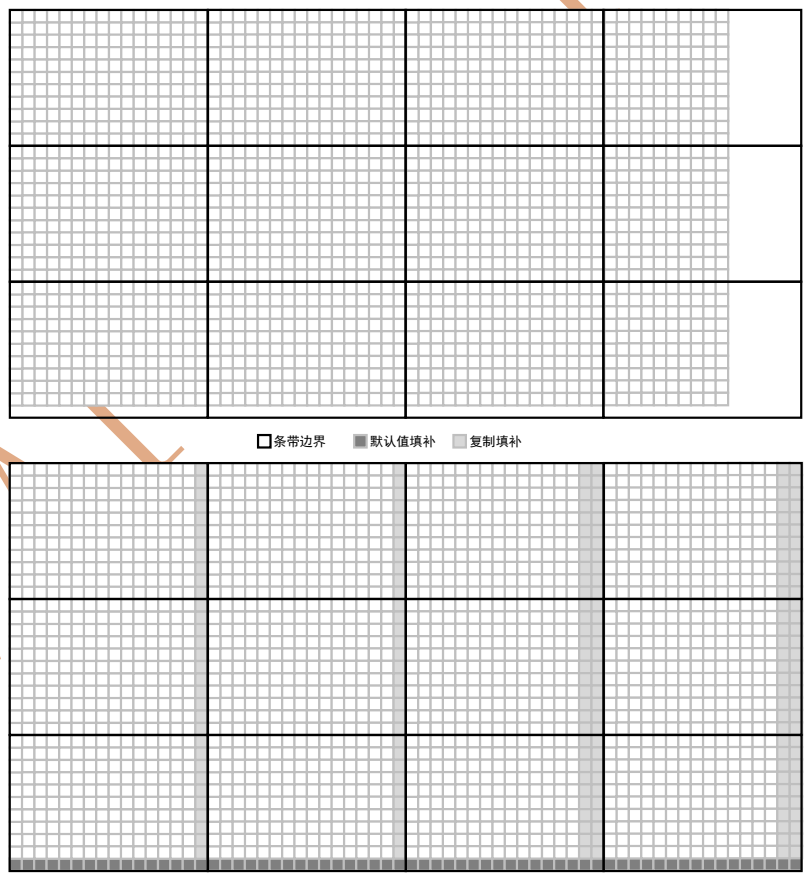


图 E.2 图像填补方式二示意图

具体的，纵向的填补方式为：

——如果是亮度通道，则最后一行条带需要在下边界填补(ImageHeight – InputImageHeight)行亮度样本；

——否则，如果图像格式为 RGB444，YUV444 或者 YUV422，则最后一行条带需要在下边界填补(ImageHeight – InputImageHeight)行色度样本；

——否则，如果图像格式为 YUV420，则最后一行条带需要在下边界填补((ImageHeight – InputImageHeight)>>1)行色度样本。

横向的填补方式为：

——如果是亮度通道且图像格式为 RGB444，YUV444 或者 YUV400，则前 n 列条带需要在右边界填补 m 列亮度样本，后续列条带需要在右边界填补(m+1)列亮度样本；

——如果是亮度通道且图像格式为 YUV422 或者 YUV420，则前 n 列条带需要在右边界填补(m<<1)列亮度样本，后续列条带需要在右边界填补((m+1)<<1)列亮度样本；

——如果是色度通道且图像格式为 RGB444 或者 YUV444，则前 n 列条带需要在右边界填补 m 列色度样本，后续列条带需要在右边界填补(m+1)列色度样本；

——如果是色度通道且图像格式为 YUV422 或者 YUV420，则前 n 列条带需要在右边界补 m 列色度样本，后续列条带需要在右边界填补(m+1)列色度样本。

其中，n，m 的计算方式为：

```
if (image_format == '001' || image_format == '010') {    /* YUV422或YUV420 */
    paddingSizeChroma = (ImageWidth – InputImageWidth) >> 1
} else {
    paddingSizeChroma = ImageWidth – InputImageWidth
}
n = PictureWidthInSlice – (paddingSizeChroma % PictureWidthInSlice)
m = paddingSizeChroma / PictureWidthInSlice
```

附 录 F
（资料性）
子流交织参考方案

F.1 概述

本附录描述了编码端一种可行的子流相关参数的设置，以及进行子流片的交织方法。

F.2 编码端子流参数设置

编码端子流缓冲区大小设置为Ceil(2 × (SubstreamExpansionRatio+2) × (SubstreamSegmentSize-2)÷8)字节。

F.3 子流交织方法

F.3.1 概述

每个子流片中，包含了至少1个编码单元产生的编码比特。子流交织方法使用块计数队列，对每个子流片最早的比特所属的编码单元进行标记，在交织时将对应编码单元顺序更小的包优先发送。编码端进行子流交织方法包含队列更新、交织、结尾处理三个过程。

F.3.2 初始化与块计数队列更新

在启动编码后，熵编码器将不同子流的数据放入不同的子流缓冲区中。在恒定码率传输过程中，由于数据仍停留在子流缓冲区中，可能导致码流缓冲区中没有足够的数据进而发生下溢。为预防下溢，编码端可以引入如下延迟：当任一编码子流缓冲区的数据达到((SubstreamExpansionRatio+2)×(SubstreamSegmentSize-2)/8)字节时，再启动子流交织以及后续的码流传输。

F.3.3 初始化与块计数队列更新

EncSSBuffer数组存储了编码器各个子流缓冲区的信息,EncSSBuffer[i]对应于第i+1个子流的编码子流缓冲区。EncSSBuffer[i]为一个先进先出缓冲，EncSSBuffer[i]->fullness表示缓冲区中当前的数据量，在每个编码单元编码完成后，数据将会进入各个EncSSBuffer[i]，同时对应的EncSSBuffer[i]->fullness会相应增加。

CounterQueue数组存储了编码器的块计数队列信息,CounterQueue[i]对应于第i+1个子流的块计数队列。CounterQueue[i]是一个先进先出队列，以数组的方式实现。CounterQueue[i]->length表示该队列当前长度。CounterQueue[i]包含以下操作：1）CounterQueue[i].push(x),令CounterQueue[i]数组长度增加1，同时将元素x放入数组的最后一个位置，此时CounterQueue[i]->length+=1；2）CounterQueue[i].pop(),将数组CounterQueue[i]的首个位置处的元素x取出，返回x的值，并将数组中所有数据向前推移一位，此时CounterQueue[i]->length-=1；3）CounterQueue[i].check(),返回数组CounterQueue[i]的首个位置处的元素x的值。

在每个条带的开始，初始化各个CounterQueue[i]为空，此时CounterQueue[i]->length的值均为0。然后对各个CounterQueue[i]执行 CounterQueue[i].push(0)。

在每个对编码单元处理完成编码操作后，对块计数队列进行更新，更新过程如下：

a) 选择一个子流，其子流索引值为 ssIdx；
b) 计算该子流缓冲区中可构建的数据包个数 NumInBuffer[ssIdx]。其计算方式为：NumInBuffer[ssIdx] = EncSSBuffer[ssIdx]->fullness / (SubstreamSegmentSize-2)；
c) 比较当前编码块计数队列长度 CounterQueue[ssIdx]->length 和 NumInBuffer[ssIdx]，若两者相等，则 CounterQueue[ssIdx].push(CurrCuIdx)；
d) 返回步骤 a)，处理下一个子流；直到所有子流均处理完毕。

F.3.4 子流片交织过程

在更新完毕块计数队列后，对各个子流缓冲区中的数据进行交织，包含子流片的构建、发送。交织过程如下：

a) 选择一个子流，其子流索引值为 ssIdx；
b) 判断该子流缓冲区中的数据是否足够构建一个子流片，即判断 EncSSBuffer[ssIdx]->fullness 是否大于等于(SubstreamSegmentSize-2)。如果足够构建，则继续步骤 c)；否则跳到步骤 f)；
c) 判断该子流对应的块计数队列 CounterQueue[ssIdx]，其队首元素值 CounterQueue[ssIdx].check()是否为所有块计数队列 CounterQueue[i]中最小。如果是，则继续步骤 d)；否则跳到步骤 f)；
d) 以当前子流缓冲区中的数据构建一个子流片。从 EncSSBuffer[ssIdx]中取出长度为(SubstreamSegmentSize-2)比特的数据，并添加 2 比特的数据头，数据头中的数据为 ssIdx，拼接成长度为 SubstreamSegmentSize 的子流片。将该子流片送入编码器最终输出的位流，同时 EncSSBuffer[ssIdx]->fullness -= (SubstreamSegmentSize-2)；
e) 执行 CounterQueue[ssIdx].pop()；
f) 返回步骤 a)，处理下一子流；直到所有子流均处理完毕。

F.3.5 条带结尾处理

如果当前编码单元为所在条带的最后一个编码单元，则子流交织模块会在上述交织过程后，执行额外的步骤，将子流缓冲区中所剩的所有数据进行打包。该过程如下：

a) 判断目前所有子流缓冲区是否存在至少一个非空，若是则继续步骤 b)；否则结束；
b) 选择一个子流，其子流索引值为 ssIdx；
c) 判断该子流对应的块计数队列 CounterQueue[ssIdx]，其队首元素值，是否为所有块计数队列中最小。如果是，则继续步骤 d)；否则跳到步骤 f)；
d) 如果该子流对应的缓冲区 EncSSBuffer[SsIdx]->fullness 小于(SubstreamSegmentSize-2)，则向缓冲区中填入 0，直至 EncSSBuffer[SsIdx]->fullness 等于(SubstreamSegmentSize-2)。同时，执行 CounterQueue[SsIdx].pop(),然后执行 CounterQueue[SsIdx].push(M)，M 的值为 $2^{32}-1$；
e) 构建一个子流片。过程同 F.3.2 中的步骤 d)；
f) 返回步骤 b)，处理下一子流；若所有子流均处理完毕，则返回步骤 a)。

附 录 G
（规范性）
接口档次传输层适配信息

G.1 概述

本附录描述了接口档次和传输层的适配信息。

G.2 图像头字节说明

本节是接口档次下图像头定义章节7.1.1.2中各个语法元素的字节排布说明。图像头定义中的语法元素共101个字节，定义为PH0~PH100，语法元素与字节的对应关系见表G.1。其中，语法元素的最高有效位位于列出的第一个PH字节内（例如，target_bpp[11:0]被映射到PH18[3:0]、PH19[7:0]）。

表G.1 接口档次语法元素与字节的对应关系

图像头语法元素	位宽	数据类型	索引
profile_id	8	无符号整数	PH0[7:0]
input_image_width	32	无符号整数	PH1[7:0] PH2[7:0] PH3[7:0] PH4[7:0]
input_image_height	32	无符号整数	PH5[7:0] PH6[7:0] PH7[7:0] PH8[7:0]
slice_width	32	无符号整数	PH9[7:0] PH10[7:0] PH11[7:0] PH12[7:0]
slice_height	32	无符号整数	PH13[7:0] PH14[7:0] PH15[7:0] PH16[7:0]
image_format	3	无符号整数	PH17[7:5]
bit_depth	5	无符号整数	PH17[4:0]
reserved_bits	4	无符号整数	PH18[7:4]
target_bpp	12	无符号整数	PH18[3:0] PH19[7:0]
padding_type_flag	1	无符号整数	PH20[7]
lossless_enable_flag	1	无符号整数	PH20[6]
transform_enable_flag	1	无符号整数	PH20[5]

表 G.1（续）

图像头语法元素	位宽	数据类型	索引
dp_enable_flag	1	无符号整数	PH20[4]
brc_enable_flag	1	无符号整数	PH20[3]
ibc_enable_flag	1	无符号整数	PH20[2]
pwq_enable_flag	1	无符号整数	PH20[1]
bwq_enable_flag	1	无符号整数	PH20[0]
vbr_enable_flag	1	无符号整数	PH21[7]
pwq_max_qp	4	无符号整数	PH21[6:3]
bwq_complex_th	3	无符号整数	PH21[2:0]
qp_refine_th0	4	无符号整数	PH22[7:4]
qp_refine_th1	4	无符号整数	PH22[3:0]
jnd_qp	4	无符号整数	PH23[7:4]
strict_jnd_qp	4	无符号整数	PH23[3:0]
chunk_size_in_cu	32	无符号整数	PH24[7:0] PH25[7:0] PH26[7:0] PH27[7:0]
chunk_num	32	无符号整数	PH28[7:0] PH29[7:0] PH30[7:0] PH31[7:0]
slice_cu_num_max_bit	8	无符号整数	PH32[7:0]
reserved_bits	2	无符号整数	PH33[7:6]
padding_stuff_flag	1	无符号整数	PH33[5]
substream_expansion_ratio_log2	3	无符号整数	PH33[4:2]
substream_segment_size	10	无符号整数	PH33[1:0] PH34[7:0]
transmission_delay_cu	16	无符号整数	PH35[7:0] PH36[7:0]
rc_buffer_size	16	无符号整数	PH37[7:0] PH38[7:0]
reserved_bits	4	无符号整数	PH39[7:4]
rc_qp_bias[0][0]	4	无符号整数	PH39[3:0]
reserved_bits	4	无符号整数	PH40[7:4]
rc_qp_bias[0][1]	4	无符号整数	PH40[3:0]
reserved_bits	4	无符号整数	PH41[7:4]
rc_qp_bias[0][2]	4	无符号整数	PH41[3:0]
reserved_bits	4	无符号整数	PH42[7:4]

表 G.1（续）

图像头语法元素	位宽	数据类型	索引
rc_qp_bias[0][3]	4	无符号整数	PH42[3:0]
reserved_bits	4	无符号整数	PH43[7:4]
rc_qp_bias[0][4]	4	无符号整数	PH43[3:0]
reserved_bits	4	无符号整数	PH44[7:4]
rc_qp_bias[1][0]	4	无符号整数	PH44[3:0]
reserved_bits	4	无符号整数	PH45[7:4]
rc_qp_bias[1][1]	4	无符号整数	PH45[3:0]
reserved_bits	4	无符号整数	PH46[7:4]
rc_qp_bias[1][2]	4	无符号整数	PH46[3:0]
reserved_bits	4	无符号整数	PH47[7:4]
rc_qp_bias[1][3]	4	无符号整数	PH47[3:0]
reserved_bits	4	无符号整数	PH48[7:4]
rc_qp_bias[1][4]	4	无符号整数	PH48[3:0]
reserved_bits	4	无符号整数	PH49[7:4]
rc_qp_bias[2][0]	4	无符号整数	PH49[3:0]
reserved_bits	4	无符号整数	PH50[7:4]
rc_qp_bias[2][1]	4	无符号整数	PH50[3:0]
reserved_bits	4	无符号整数	PH51[7:4]
rc_qp_bias[2][2]	4	无符号整数	PH51[3:0]
reserved_bits	4	无符号整数	PH52[7:4]
rc_qp_bias[2][3]	4	无符号整数	PH52[3:0]
reserved_bits	4	无符号整数	PH53[7:4]
rc_qp_bias[2][4]	4	无符号整数	PH53[3:0]
reserved_bits	4	无符号整数	PH54[7:4]
rc_qp_bias[3][0]	4	无符号整数	PH54[3:0]
reserved_bits	4	无符号整数	PH55[7:4]
rc_qp_bias[3][1]	4	无符号整数	PH55[3:0]
reserved_bits	4	无符号整数	PH56[7:4]
rc_qp_bias[3][2]	4	无符号整数	PH56[3:0]
reserved_bits	4	无符号整数	PH57[7:4]
rc_qp_bias[3][3]	4	无符号整数	PH57[3:0]
reserved_bits	4	无符号整数	PH58[7:4]
rc_qp_bias[3][4]	4	无符号整数	PH58[3:0]
reserved_bits	4	无符号整数	PH59[7:4]
rc_qp_bias[4][0]	4	无符号整数	PH59[3:0]
reserved_bits	4	无符号整数	PH60[7:4]
rc_qp_bias[4][1]	4	无符号整数	PH60[3:0]

表 G.1（续）

图像头语法元素	位宽	数据类型	索引
reserved_bits	4	无符号整数	PH61[7:4]
rc_qp_bias[4][2]	4	无符号整数	PH61[3:0]
reserved_bits	4	无符号整数	PH62[7:4]
rc_qp_bias[4][3]	4	无符号整数	PH62[3:0]
reserved_bits	4	无符号整数	PH63[7:4]
rc_qp_bias[4][4]	4	无符号整数	PH63[3:0]
rc_decrease_step_log2	8	无符号整数	PH64[7:0]
rc_fullness_calc_multiplier	8	无符号整数	PH65[7:0]
reserve_bits	3	无符号整数	PH66[7:5]
rc_fullness_calc_shift	5	无符号整数	PH66[4:0]
rc_extra_buffer_decrease_step_log2	8	无符号整数	PH67[7:0]
rc_extra_buffer_penalty_log2_minus3	8	无符号整数	PH68[7:0]
reserved_bits	6	无符号整数	PH69[7:2]
rc_target_end_ratio_minus4	2	无符号整数	PH69[1:0]
rc_ratio0	8	无符号整数	PH70[7:0]
reserved_bits	5	无符号整数	PH71[7:3]
rc_param0	11	无符号整数	PH71[2:0] PH72[7:0]
reserved_bits	7	无符号整数	PH73[7:1]
rc_ratio1	9	无符号整数	PH73[0] PH74[7:0]
reserved_bits	5	无符号整数	PH75[7:3]
rc_param1	11	无符号整数	PH75[2:0] PH76[7:0]
reserved_bits	5	无符号整数	PH77[7:3]
rc_ratio2	11	无符号整数	PH77[2:0] PH78[7:0]
reserved_bits	5	无符号整数	PH79[7:3]
rc_param2	11	无符号整数	PH79[2:0] PH80[7:0]
rc_max_relative_bits	8	无符号整数	PH81[7:0]
reserved_bits	5	无符号整数	PH82[7:3]
rc_lossless_bits[0]	11	无符号整数	PH82[2:0] PH83[7:0]
reserved_bits	5	无符号整数	PH84[7:3]
rc_lossless_bits[1]	11	无符号整数	PH84[2:0] PH85[7:0]

表 G.1（续）

图像头语法元素	位宽	数据类型	索引
reserved_bits	5	无符号整数	PH86[7:3]
rc_lossless_bits[2]	11	无符号整数	PH86[2:0] PH87[7:0]
reserved_bits	5	无符号整数	PH88[7:3]
rc_lossless_bits[3]	11	无符号整数	PH88[2:0] PH89[7:0]
reserved_bits	5	无符号整数	PH90[7:3]
rc_lossless_bits[4]	11	无符号整数	PH90[2:0] PH91[7:0]
reserved_bits	5	无符号整数	PH92[7:3]
rc_avg_lossless_bits	11	无符号整数	PH92[2:0] PH93[7:0]
reserved_bits	5	无符号整数	PH94[7:3]
rc_bits_offset	11	无符号整数	PH94[2:0] PH95[7:0]
reserved_bits	5	无符号整数	PH96[7:3]
rc_max_lossless_bits	11	无符号整数	PH96[2:0] PH97[7:0]
reserved_bits	6	无符号整数	PH98[7:2]
rc_relative_th	10	无符号整数	PH98[1:0] PH99[7:0]
reserved_bits	5	无符号整数	PH100[7:3]
rc_complex_th	3	无符号整数	PH100[2:0]