用Python生成10秒H264彩色视频

0x01. 背景

由于工作项目需要，在做RTP协议分析时，发现RTP可以承载多种类型的音频和视频数据，包括MP4、MP3、AMR、H264、H265等类型的数据，每种类型的音频或者视频数据在经过流媒体服务器与客户端之间进行传输时，都要按照流媒体服务器与客户端的数据传输协议所规定的格式封装音频或者视频的帧数据。

自己实现流媒体服务

如果要自己实现一个流媒体服务，可以自己定义一套服务器与客户端之间的传输协议，也可以采用比较成熟和知名的传输协议，如RTP实时传输协议等。

如果时基于文件的视频直播或者点播业务，除了要封装流媒体服务端与客户端之间的传输协议外，还需要对视频文件的视频数据拆解为每一帧，也就是需要明白视频文件里的视频数据帧的组装结构。

播放器

如果要自己播放每一帧，还需要对每一帧数据的每一个字节或者比特按照规范所定义的结构进行解析，当然也可以采用业界比较知名的开源框架如VLC、FFMPEG等。

解析的过程称为解码，反过来则是编码。编码通常是压缩视频数据，解码则是解压缩视频。目前国际比较知名的视频的压缩技术是H264/AVC、HEVC等（中国是AVS），两者都是基于哥伦布编码、视频的数据的预测等技术。

视频编解码

视频编码技术非常复杂，如果要完全弄明白每一个细节，要对视频编码规范文档进行深入的研究。对于完全没有视频编解码开发经验的程序员来说，研究标准文档的过程比较吃力，如果是国标中文文档相对而言还比较容易，如果是国际标准则比较难，如H264的国际标准ITU-T REC.H.264（）英文文档多达800多页。

想要完全地明白视频编解码标准，除了埋头研究视频编码规范文档之外，还可以借助互联网查阅相关资料，也可以结合第三方编解码库的源代码。

作为一名程序员，最重要的还是自己能够亲手尝试实现视频编解码。

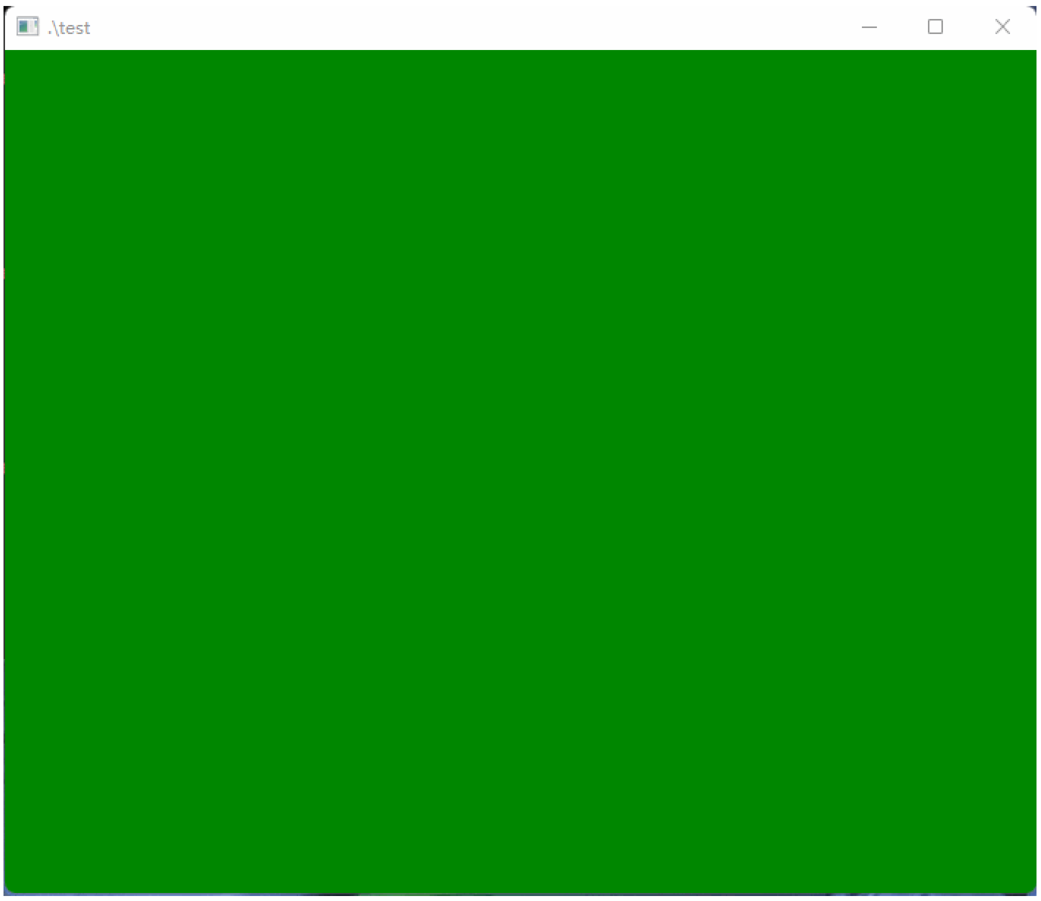
所以写下了这篇文章。

0x02. 原理

只需要生成H264的SPS、PPS、SEI、I帧等数据，转换为字节流并写入文件即可。

0x03. 生成一段H264视频

效果展示



代码

包括3个文件golomb.py、avc.py和ghf.py，运行ghf.py即可生成H264视频裸流文件test。

golomb.py源码如下:

```
# author: yangyiyin
# 2022/2/14 chengdu

class ExpGolombObject(object):
    def __init__(self, value, bit_length=0):
        self._bit_count = value.bit_length()
        self._bit_length = bit_length or self._bit_count
        self._zero_count = self._bit_length - self._bit_count
        self.value = value
        self.value_encoded = 0

        if self._bit_length < self._bit_count:
            raise Exception("bit length of value {} should be {} at least", self.value, self._bit_count)

    def __bool__(self):
        return 0 != self.value

    @property
    def zero_count(self):
        return self._zero_count

    @property
    def bit_count(self):
        return self._bit_count

    def encode(self):
        self._bit_count = self.value.bit_length()
        self._zero_count = self._bit_length - self._bit_count
        self.value_encoded = self.value
```

```
class FixedPatternBin(ExpGolombObject):
    def __init__(self, value=0):
        super(FixedPatternBin, self).__init__(value, 1)


class FixedLenUint(ExpGolombObject):
    def __init__(self, bit_length, value=0):
        super(FixedLenUint, self).__init__(value, bit_length)

    def set_data(self, datas, bit_index):
        byte_index = divmod(bit_index, 8)[0]
        bit = bit_index - byte_index * 8

        self.value = (datas[byte_index] << 24) + (datas[byte_index + 1] << 16) + (datas[byte_index + 2] << 8) + (datas[byte_index + 3])

        if bit + self._bit_count <= 32:
            self.value = (self.value << bit) & 0xffffffff
            self.value = (self.value >> (32 - self._bit_count)) & 0xffffffff

        else:
            tmp = self._bit_count + bit - 32

            self.value = (self.value << bit) & 0xffffffff
            self.value = (self.value >> bit)
            self.value = (self.value << tmp) & 0xffffffff

            # the fifth byte
            self.value += (datas[byte_index + 4] >> (8 - tmp))


class ExpGolombNumber(ExpGolombObject):
    def __init__(self, value):
        super(ExpGolombNumber, self).__init__(value)

    def __str__(self):
        return "zero_count: {}, bit_count:{}, value: {}".format(self.zero_count, self.bit_count, self.value)

    def set_data(self, datas, start_bit_index):
        """
        :param datas:
        :param start_bit_index:  比特索引值，0开始计数
        :return:
        """
        self.value = 0
        self._zero_count = 0
        self._bit_count = 0

        byte_index = divmod(start_bit_index, 8)[0]
        byte = datas[byte_index]
        bit_index = start_bit_index - byte_index * 8

        if ((byte << bit_index) & 0xff) <= 0x80:
            bit_index = start_bit_index - byte_index * 8

            while True:
                if ((byte << bit_index) & 0xff) >= 0x80:
                    break

                self._zero_count += 1
                start_bit_index += 1
                bit_index += 1

                if 8 == bit_index:
                    byte_index = divmod(start_bit_index, 8)[0]
                byte = datas[byte_index]
                bit_index = 0

        self._bit_count = self._zero_count + 1

        if self._bit_count > 8:
            return False

        if bit_index + self._bit_count <= 8:
            mask = 0xff >> bit_index
            self.value_encoded = (datas[byte_index] & mask) >> (8 - bit_index - self._bit_count)

        else:
            mask = 0xff >> bit_index
            self.value_encoded = datas[byte_index] & mask
            self.value_encoded = (self.value_encoded << (self._bit_count - (8 - bit_index))) & 0xff

            byte_index += 1
            self.value_encoded += (datas[byte_index] >> (16 - bit_index - self._bit_count))

        return True


class ExpGolombUE(ExpGolombNumber):
    def __init__(self, value=0):
        super(ExpGolombUE, self).__init__(value)

    def encode(self):
        self.value_encoded = self.value + 1
        self._bit_count = self.value_encoded.bit_length()
        self._zero_count = self._bit_count - 1

    def set_data(self, datas, start_bit_index):
        if super(ExpGolombUE, self).set_data(datas, start_bit_index):
            self.value = self.value_encoded - 1


class ExpGolombSE(ExpGolombNumber):
    def __init__(self, value=0):
        super(ExpGolombSE, self).__init__(value)

    def encode(self):
        tmp = abs(self.value)

        if 0 == self.value:
            self._bit_count = 1
            self._zero_count = 0
            self.value_encoded = 1
            return

        self._bit_count = tmp.bit_length()
        self._zero_count = self._bit_count

        tmp = tmp << 1
        self._bit_count += 1

        if self.value < 0:
            tmp += 1

        self.value_encoded = tmp

    def set_data(self, datas, start_bit_index):
        if super(ExpGolombSE, self).set_data(datas, start_bit_index):
            sign = self.value_encoded & 0x01
            self.value = self.value_encoded >> 1
        if sign:
            self.value *= -1


class ExpGolombCodec:
    def __init__(self, buffer):
        self.__value = 0
        self.__prefix = 0
        self.__buffer = buffer

    def __str__(self):
        return "prefix={}, value={}, data length={}".format(self.__prefix, bin(self.__value), len(self.__buffer))

    def __add_coded(self, value):
        if len(self.__buffer) > 3 and value <= 0x03:
            if not self.__buffer[-2] and not self.__buffer[-1]:
                self.__buffer.append(0x03)
        self.__buffer.append(value)

    def __add_item(self, glbobj):
        self.__value |= (glbobj.value_encoded << 40 - self.__prefix - glbobj.zero_count - glbobj.bit_count)
        self.__prefix += glbobj.zero_count + glbobj.bit_count

    def __check_prefix(self, glbobj):
        if self.__prefix + glbobj.zero_count + glbobj.bit_count > 40:
            if self.__prefix >= 32:
                tmp = self.__value & 0xffffffff00
                tmp = tmp >> 8

                self.__add_coded(tmp >> 24 & 0xff)
                self.__add_coded(tmp >> 16 & 0xff)
                self.__add_coded(tmp >> 8 & 0xff)
                self.__add_coded(tmp & 0xff)

                self.__value = (self.__value << 32) & 0xff00000000
                self.__prefix -= 32

            elif self.__prefix >= 24:
                tmp = self.__value & 0xffffff0000
                tmp = tmp >> 16

                self.__add_coded(tmp >> 16)
                self.__add_coded(tmp >> 8 & 0xff)
                self.__add_coded(tmp & 0xff)

                self.__value = (self.__value << 24) & 0xff00000000
                self.__prefix -= 24

            elif self.__prefix >= 16:
                tmp = self.__value & 0xffff000000
                tmp = tmp >> 24

                self.__add_coded(tmp >> 8 & 0xff)
                self.__add_coded(tmp & 0xff)

                self.__value = (self.__value << 16) & 0xff00000000
                self.__prefix -= 16

            elif self.__prefix >= 8:
                tmp = self.__value & 0xff00000000
                tmp = tmp >> 32

                self.__add_coded(tmp & 0xff)
            self.__value = (self.__value << 8) & 0xff00000000
            self.__prefix -= 8

        if 8 == self.__prefix:
            self.__add_coded(self.__value >> 32 & 0xff)
            self.__value = 0
            self.__prefix = 0

    def byte_aligned(self):
        return 0 == self.__prefix % 8

    def add_trail(self):
        ret = self.__prefix % 8

        if ret:
            self.__buffer[-1] |= (1 << (8 - ret - 1))
        else:
            self.__add_coded(0x80)

    def reset(self):
        self.__value = 0
        self.__prefix = 0
        self.__buffer.clear()

    def encode_done(self, trail_bit):
        if self.__prefix > 32:
            self.__add_coded(self.__value >> 32 & 0xff)
            self.__add_coded(self.__value >> 24 & 0xff)
            self.__add_coded(self.__value >> 16 & 0xff)
            self.__add_coded(self.__value >> 8 & 0xff)
            self.__add_coded(self.__value & 0xff)

        elif self.__prefix > 24:
            self.__add_coded(self.__value >> 32 & 0xff)
            self.__add_coded(self.__value >> 24 & 0xff)
            self.__add_coded(self.__value >> 16 & 0xff)
            self.__add_coded(self.__value >> 8 & 0xff)

        elif self.__prefix > 16:
            self.__add_coded(self.__value >> 32 & 0xff)
            self.__add_coded(self.__value >> 24 & 0xff)
            self.__add_coded(self.__value >> 16 & 0xff)

        elif self.__prefix > 8:
            self.__add_coded(self.__value >> 32 & 0xff)
            self.__add_coded(self.__value >> 24 & 0xff)

        else:
            self.__add_coded(self.__value >> 32 & 0xff)

        if trail_bit:
            self.add_trail()

        print(str(self))
        self.__prefix = 0
        self.__value = 0

    def append(self, glbobj, dbg=False):
        glbobj.encode()
        self.__check_prefix(glbobj)

        if dbg:
            print("before append {} : {}".format(glbobj.value_encoded, self))

        self.__add_item(glbobj)
        if dbg:
            print("after append {} : {}".format(glbobj.value_encoded, self))

    def encode_ue(self, value_list):
        self.__value = 0
        self.__prefix = 0
        self.__buffer = []

        for x in value_list:
            y = x

            if type(x) is not int:
                y = ord(x)

            if y > 0xff:
                raise Exception("must be byte: {}".format(y))

            e = ExpGolombUE(y)
            self.append(e)
```

avc.py是按照H264标准封装的部分H264语法结构（暂未实现预测等关键参数），源码如下：

```
# -*- coding: cp936 -*-
# author: yangyiyin
# 2022/2/14 chengdu

import abc
import math
from golomb import *

NALU_TYPE_RESERVED = 0
NALU_TYPE_CODED_SLICE_NON_IDR_PIC = 1
NALU_TYPE_CODED_SLICE_DATA_PARTION_A = 2
NALU_TYPE_CODED_SLICE_DATA_PARTION_B =  3
NALU_TYPE_CODED_SLICE_DATA_PARTION_C = 4
NALU_TYPE_CODED_SLICE_IDR_PIC = 5
NALU_TYPE_SUPPLEMENTAL_ENHANCEMENT_INFO = 6
NALU_TYPE_SEQUENCE_PARAMETER_SET = 7
NALU_TYPE_PICTURE_PARAMETER_SET = 8
NALU_TYPE_ACCESS_UNIT_DELIMITER = 9
NALU_TYPE_END_OF_SEQUENCE = 10
NALU_TYPE_END_OF_STREAM = 11
NALU_TYPE_FILLER_DATA = 12
NALU_TYPE_SEQUENCE_PARAMETER_SET_EXT = 13
NALU_TYPE_PREFIX_NALU = 14
NALU_TYPE_SUBSET_SEQUENCE_PARAMETER_SET = 15
NALU_TYPE_DEPTH_PARAMETER_SET = 16
NALU_TYPE_RESERVED_1 = 17
NALU_TYPE_RESERVED_2 = 18
NALU_TYPE_CODED_SLICE_AUXILIARY_PIC = 19
NALU_TYPE_CODED_SLICE_EXT = 20
NALU_TYPE_CODED_SLICE_EXT_DEPTH_3D_AVC = 21

NALU_TYPE_SINGLE_TIME_AGG_PACKET_A = 24
NALU_TYPE_SINGLE_TIME_AGG_PACKET_B = 25
NALU_TYPE_MULTI_TIME_AGG_PACKET_16 = 26
NALU_TYPE_MULTI_TIME_AGG_PACKET_24 = 27
NALU_TYPE_FRAGMENTATION_UINT_A = 28
NALU_TYPE_FRAGMENTATION_UNIT_B  = 29

# pps.slice_group_map_type
SLICE_GROUP_MAP_TYPE_INTERLEAVED = 0
SLICE_GROUP_MAP_TYPE_DISPERSED_MAPPING = 1
```

```
SLICE_GROUP_MAP_TYPE_FOREGROUND_LEFTOVER = 2
SLICE_GROUP_MAP_TYPE_CANGING = 3
SLICE_GROUP_MAP_TYPE_CANGING1 = 4
SLICE_GROUP_MAP_TYPE_CANGING2 = 5
SLICE_GROUP_MAP_TYPE_EXPLICIT_ASSIGNMENT = 6

# sps.chroma_format_idc
CHROMA_FORMAT_MONOCHROME = 0
CHROMA_FORMAT_420 = 1
CHROMA_FORMAT_422 = 2
CHROMA_FORMAT_444 = 3

# slice type
SLICE_TYPE_P = [0, 5]
SLICE_TYPE_B = [1, 6]
SLICE_TYPE_I = [2, 7]
SLICE_TYPE_SP = [3, 8]
SLICE_TYPE_SI = [4, 9]

SEI_PT_BUFFERING_PERIOD = 0
SEI_PT_PIC_TIMING = 1
SEI_PT_PCAN_SCAN_RECT  = 2
SEI_PT_FILLER_PAYLOAD = 3
SEI_PT_USER_DATA_REGISTERED_ITU_T_T35 = 4
SEI_PT_USER_DATA_UNREGISTERED = 5
SEI_PT_RECOVERY_POINT= 6
SEI_PT_DEC_REF_PIC_MARKING_REPETITION = 7
SEI_PT_SPARE_PIC = 8
SEI_PT_SCENE_INFO = 9
SEI_PT_SUB_SEQ_IFO  = 10
SEI_PT_SUB_SEQ_LAYER_CHARACTERISTICS = 11
SEI_PT_SUB_SEQ_CHARACTERISTICS = 12
SEI_PT_FULL_FRAME_FREEZE = 13
SEI_PT_FULL_FRAME_FREEZE_RELEASE  = 14
SEI_PT_FULL_FRAME_SNAPSHOT = 15
SEI_PT_PROGRESSIVE_REFINEMENT_SEGMENT_START = 16
SEI_PT_PROGRESSIVE_REFINEMENT_SEGMENT_END = 17
SEI_PT_MOTION_CONSTRAINED_SLICE_GROUP_SET = 18
SEI_PT_FILM_GRAIN_CHARACTERISTICS = 19
SEI_PT_DEBLOCKING_FILTER_DISPLAY_PREFERENCE = 20
SEI_PT_STEREO_VIDEO_INFO  = 21
SEI_PT_POST_FILTER_HINT = 22
SEI_PT_TONE_MAPPING_INFO = 23
SEI_PT_SCALABILITY_INFO = 24
SEI_PT_SUB_PIC_SCALABLE_LAYER = 25
SEI_PT_NON_REQUIRED_LAYER_REP = 26
SEI_PT_PRIORITY_LAYER_INFO = 27
SEI_PT_LAYERS_NOT_PRESENT = 28
SEI_PT_LAYER_DEPENDENCY_CHANGE = 29
SEI_PT_SCALABLE_NESTING = 30
SEI_PT_BASE_LAYER_TEMPORAL_HRD = 31
SEI_PT_QUALITY_LAYER_INTEGRITY_CHECK = 32
SEI_PT_REDUNDANT_PIC_PROPERTY = 33
SEI_PT_TL0_DEP_REP_INDEX = 34
SEI_PT_TL_SWITCHING_POINT = 35
SEI_PT_PARALLEL_DECODING_INFO = 36


class RbspTrailingBits(ExpGolombObject):
    def __init__(self, bit_count, value):
        super(RbspTrailingBits, self).__init__(bit_count, value)


class NALUHeader(object):
    def __init__(self, nalu_type):
        self.forbidden_zero_bit = FixedLenUint(1)
        self.nalu_ref_idc = FixedLenUint(2)
        self.nalu_type = FixedLenUint(5, nalu_type)


class NALU(metaclass=abc.ABCMeta):
    def __init__(self, nalu_type):
        self.__data = bytearray()

        self.header = NALUHeader(nalu_type)
        self.length = 0
        self.codec = ExpGolombCodec(self.__data)

    def encode(self, trail_bit=True):
        self.codec.reset()
        self.codec.append(self.header.forbidden_zero_bit)
        self.codec.append(self.header.nalu_ref_idc)
        self.codec.append(self.header.nalu_type)
        self._encode()
        self.codec.encode_done(trail_bit)
        self.length = len(self.__data)

    @property
    def data(self):
        return self.__data

    @abc.abstractmethod
    def _encode(self):
        ...


# HRD parameters syntax
class HRDParameters:
    def __init__(self):
        self.cpb_cnt_minus1 = ExpGolombUE()
        self.bit_rate_scale = FixedLenUint(4)
        self.cpb_size_scale = FixedLenUint(4)
        self.bit_rate_value_minus1 = [ExpGolombUE() for x in range(self.cpb_cnt_minus1.value)]
        self.cpb_size_value_minus1 = [ExpGolombUE() for x in range(self.cpb_cnt_minus1.value)]
        self.cbr_flag = [FixedLenUint(1) for x in range(self.cpb_cnt_minus1.value)]
        self.initial_cpb_removal_delay_length_minus1 = FixedLenUint(4)
        self.cpb_removal_delay_length_minus1 = FixedLenUint(4)
        self.dpb_output_delay_length_minus1 = FixedLenUint(4)
        self.time_offset_length = FixedLenUint(4)


# VUI parameters syntax
class VUIParameters():
    def __init__(self):
        self.aspect_ratio_info_present_flag = FixedLenUint(1);
        self.aspect_ratio_idc = FixedLenUint(8);
        self.sar_width = FixedLenUint(16)
        self.sar_height = FixedLenUint(16)
        self.overscan_info_present_flag = FixedLenUint(1)
        self.overscan_appropriate_flag = FixedLenUint(1)

        self.video_signal_type_present_flag = FixedLenUint(1)
        self.video_format = FixedLenUint(3)
        self.video_full_range_flag = FixedLenUint(1)
        self.colour_description_present_flag = FixedLenUint(1)
        self.colour_primaries = FixedLenUint(8)
        self.transfer_characteristics = FixedLenUint(8)
        self.matrix_coefficients = FixedLenUint(8)

        self.chroma_loc_info_present_flag = FixedLenUint(1)
        self.chroma_sample_loc_type_top_field = ExpGolombUE()
        self.chroma_sample_loc_type_bottom_field = ExpGolombUE()

        self.timing_info_present_flag = FixedLenUint(1)
        self.num_units_in_tick = FixedLenUint(32)
        self.time_scale = FixedLenUint(32)
        self.fixed_frame_rate_flag = FixedLenUint(1)

        self.nal_hrd_parameters_present_flag = FixedLenUint(1)
        self.nal_hrd_parameters = HRDParameters()

        self.vcl_hrd_parameters_present_flag = FixedLenUint(1)
        self.vcl_hrd_parameters = HRDParameters()

        self.low_delay_hrd_flag = FixedLenUint(1)
        self.pic_struct_present_flag = FixedLenUint(1)

        self.bitstream_restriction_flag = FixedLenUint(1)
        self.motion_vectors_over_pic_boundaries_flag = FixedLenUint(1)
        self.max_bytes_per_pic_denom = ExpGolombUE()
        self.max_bits_per_mb_denom = ExpGolombUE()
        self.log2_max_mv_length_horizontal = ExpGolombUE()
        self.log2_max_mv_length_vertical = ExpGolombUE()
        self.max_num_reorder_frames = ExpGolombUE()
        self.max_dec_frame_buffering = ExpGolombUE()


class SequenceParameterSet(NALU):
    def __init__(self):
        super(SequenceParameterSet, self).__init__(NALU_TYPE_SEQUENCE_PARAMETER_SET)

        self.profile_idc = FixedLenUint(8)
        self.constraint_set0_flag = FixedLenUint(1)
        self.constraint_set1_flag = FixedLenUint(1)
        self.constraint_set2_flag = FixedLenUint(1)
        self.constraint_set3_flag = FixedLenUint(1)
        self.constraint_set4_flag = FixedLenUint(1)
        self.constraint_set5_flag = FixedLenUint(1)
        self.reserved_zero_2bits = FixedLenUint(2)
        self.level_idc = FixedLenUint(8)
        self.seq_parameter_set_id = ExpGolombUE()

        self.chroma_format_idc = ExpGolombUE()
        self.separate_colour_plane_flag = FixedLenUint(1)
        self.bit_depth_luma_minus8 = ExpGolombUE()
        self.bit_depth_chroma_minus8 = ExpGolombUE()
        self.qpprime_y_zero_transform_bypass_flag = FixedLenUint(1)

        self.seq_scaling_matrix_present_flag = FixedLenUint(1)
        self.seq_scaling_list_present_flag = [FixedLenUint(1) for x in range(12)]
        self.delta_scale = [[ExpGolombSE() for x in range(16)] for y in range(6)] + [[ExpGolombSE() for x in range(64)] for y in range(6)]

        self.log2_max_frame_num_minus4 = ExpGolombUE()
        self.pic_order_cnt_type = ExpGolombUE()

        self.log2_max_pic_order_cnt_lsb_minus4 = ExpGolombUE()
        self.delta_pic_order_always_zero_flag = FixedLenUint(1)
        self.offset_for_non_ref_pic = ExpGolombSE()
        self.offset_for_top_to_bottom_field = ExpGolombSE()
        self.num_ref_frames_in_pic_order_cnt_cycle = ExpGolombUE()
        self.offset_for_ref_frame = [ExpGolombSE(0) for x in range(self.num_ref_frames_in_pic_order_cnt_cycle.value)]

        self.max_num_ref_frames = ExpGolombUE()
        self.gaps_in_frame_num_value_allowed_flag = FixedLenUint(1)
        self.pic_width_in_mbs_minus1 = ExpGolombUE()
        self.pic_height_in_map_units_minus1 = ExpGolombUE()

        self.frame_mbs_only_flag = FixedLenUint(1)
        self.mb_adaptive_frame_field_flag = FixedLenUint(1)
        self.direct_8x8_inference_flag = FixedLenUint(1)
        self.frame_cropping_flag = FixedLenUint(1)

        self.frame_crop_left_offset = ExpGolombUE()
        self.frame_crop_right_offset = ExpGolombUE()
        self.frame_crop_top_offset = ExpGolombUE()
        self.frame_crop_bottom_offset = ExpGolombUE()

        self.vui_parameters_present_flag = FixedLenUint(1)
        self.vui_parameters = VUIParameters()

    # Table 6-1 – SubWidthC, and SubHeightC values derived from chroma_format_idc and separate_colour_plane_flag
    def get_sub_width_height_c(self):
        if CHROMA_FORMAT_MONOCHROME == self.chroma_format_idc.value or self.separate_colour_plane_flag.value:
            return 0, 0

        if CHROMA_FORMAT_420 == self.chroma_format_idc.value:
            return 2, 2

        elif CHROMA_FORMAT_422 == self.chroma_format_idc.value:
            return 2, 1

        elif CHROMA_FORMAT_444 == self.chroma_format_idc.value:
            return 1, 1

        return None, None

    def _encode(self):
        self.codec.append(self.profile_idc)
        self.codec.append(self.constraint_set0_flag)
        self.codec.append(self.constraint_set1_flag)
        self.codec.append(self.constraint_set2_flag)
        self.codec.append(self.constraint_set3_flag)
        self.codec.append(self.constraint_set4_flag)
        self.codec.append(self.constraint_set5_flag)
        self.codec.append(self.reserved_zero_2bits)
        self.codec.append(self.level_idc)
        self.codec.append(self.seq_parameter_set_id)

        if self.profile_idc.value in [100, 110, 122, 244, 44, 83, 86, 118, 128, 138, 139, 134, 135]:
            self.codec.append(self.chroma_format_idc)

            if self.chroma_format_idc.value == CHROMA_FORMAT_444:
                self.codec.append(self.separate_colour_plane_flag)

            self.codec.append(self.bit_depth_luma_minus8)
            self.codec.append(self.bit_depth_chroma_minus8)
            self.codec.append(self.qpprime_y_zero_transform_bypass_flag)
            self.codec.append(self.seq_scaling_matrix_present_flag)

            count = 8 if self.chroma_format_idc.value != CHROMA_FORMAT_444 else 12

            if self.seq_scaling_matrix_present_flag.value:
                for index, flag in enumerate(self.seq_scaling_list_present_flag[0:count]):
                    self.codec.append(flag)
                    if not flag.value:
                        continue

                    last_scale = 8
                    next_scale = 8
            for delta_scale in self.delta_scale[index]:
                self.codec.append(delta_scale)

                next_scale = divmod(last_scale + delta_scale.value + 256, 256)[1]
                last_scale = last_scale if 0 == next_scale else next_scale

                if not next_scale:
                    print("next_scal is zero, skip append glb item.")
                    break

    self.codec.append(self.log2_max_frame_num_minus4)
    self.codec.append(self.pic_order_cnt_type)

    if self.pic_order_cnt_type.value == 0:
        self.codec.append(self.log2_max_pic_order_cnt_lsb_minus4)
    else:
        self.codec.append(self.delta_pic_order_always_zero_flag)
        self.codec.append(self.offset_for_non_ref_pic)
        self.codec.append(self.offset_for_top_to_bottom_field)
        self.codec.append(self.num_ref_frames_in_pic_order_cnt_cycle)
        for x in self.offset_for_ref_frame:
            self.codec.append(x)

    self.codec.append(self.max_num_ref_frames)
    self.codec.append(self.gaps_in_frame_num_value_allowed_flag)
    self.codec.append(self.pic_width_in_mbs_minus1)
    self.codec.append(self.pic_height_in_map_units_minus1)
    self.codec.append(self.frame_mbs_only_flag)

    if not self.frame_mbs_only_flag.value:
        self.codec.append(self.mb_adaptive_frame_field_flag)

    self.codec.append(self.direct_8x8_inference_flag)
    self.codec.append(self.frame_cropping_flag)

    if self.frame_cropping_flag.value:
        self.codec.append(self.frame_crop_left_offset)
        self.codec.append(self.frame_crop_right_offset)
        self.codec.append(self.frame_crop_top_offset)
        self.codec.append(self.frame_crop_bottom_offset)

    self.codec.append(self.vui_parameters_present_flag)

    if self.vui_parameters_present_flag.value:
        self.__add_vui_params()

def __add_vui_params(self):
    self.codec.append(self.vui_parameters.aspect_ratio_info_present_flag)

    if self.vui_parameters.aspect_ratio_info_present_flag.value:
        self.codec.append(self.vui_parameters.aspect_ratio_idc)
        self.codec.append(self.vui_parameters.sar_width)
        self.codec.append(self.vui_parameters.sar_height)

    self.codec.append(self.vui_parameters.overscan_info_present_flag)

    if self.vui_parameters.overscan_info_present_flag.value:
        self.codec.append(self.vui_parameters.overscan_appropriate_flag)

    self.codec.append(self.vui_parameters.video_signal_type_present_flag)

    if self.vui_parameters.video_signal_type_present_flag.value:
        self.codec.append(self.vui_parameters.video_format)
        self.codec.append(self.vui_parameters.video_full_range_flag)
```

```
            self.codec.append(self.vui_parameters.colour_description_present_flag)

            if self.vui_parameters.colour_description_present_flag.value:
                self.codec.append(self.vui_parameters.colour_primaries)
                self.codec.append(self.vui_parameters.transfer_characteristics)
                self.codec.append(self.vui_parameters.matrix_coefficients)

        self.codec.append(self.vui_parameters.chroma_loc_info_present_flag)

        if self.vui_parameters.chroma_loc_info_present_flag.value:
            self.codec.append(self.vui_parameters.chroma_sample_loc_type_top_field)
            self.codec.append(self.vui_parameters.chroma_sample_loc_type_bottom_field)

        self.codec.append(self.vui_parameters.timing_info_present_flag)

        if self.vui_parameters.timing_info_present_flag.value:
            self.codec.append(self.vui_parameters.num_units_in_tick)
            self.codec.append(self.vui_parameters.time_scale)
            self.codec.append(self.vui_parameters.fixed_frame_rate_flag)

        self.codec.append(self.vui_parameters.nal_hrd_parameters_present_flag)
        if self.vui_parameters.nal_hrd_parameters_present_flag.value:
            self.__add_hrd_params(self.vui_parameters.nal_hrd_parameters)

        self.codec.append(self.vui_parameters.vcl_hrd_parameters_present_flag)
        if self.vui_parameters.vcl_hrd_parameters_present_flag.value:
            self.__add_hrd_params(self.vui_parameters.vcl_hrd_parameters)

        if self.vui_parameters.nal_hrd_parameters_present_flag.value or self.vui_parameters.vcl_hrd_parameters_present_flag.value:
            self.codec.append(self.vui_parameters.low_delay_hrd_flag)
        self.codec.append(self.vui_parameters.pic_struct_present_flag)

        self.codec.append(self.vui_parameters.bitstream_restriction_flag)

        if self.vui_parameters.bitstream_restriction_flag.value:
            self.codec.append(self.vui_parameters.motion_vectors_over_pic_boundaries_flag)
            self.codec.append(self.vui_parameters.max_bytes_per_pic_denom)
            self.codec.append(self.vui_parameters.max_bits_per_mb_denom)
            self.codec.append(self.vui_parameters.log2_max_mv_length_horizontal)
            self.codec.append(self.vui_parameters.log2_max_mv_length_vertical)
            self.codec.append(self.vui_parameters.max_num_reorder_frames)
            self.codec.append(self.vui_parameters.max_dec_frame_buffering)

    def __add_hrd_params(self, hrd):
        self.codec.append(hrd.cpb_cnt_minus1)
        self.codec.append(hrd.bit_rate_scale)
        self.codec.append(hrd.cpb_size_scale)

        for x in hrd.cpb_cnt_minus1:
            self.codec.append(hrd.bit_rate_value_minus1[x])
            self.codec.append(hrd.cpb_size_value_minus1[x])
            self.codec.append((hrd.cbr_flag[x]))

        self.codec.append(hrd.initial_cpb_removal_delay_length_minus1)
        self.codec.append(hrd.cpb_removal_delay_length_minus1)
        self.codec.append(hrd.dpb_output_delay_length_minus1)
        self.codec.append(hrd.time_offset_length)


class PictureParameterSet(NALU):
    def __init__(self):
        super(PictureParameterSet, self).__init__(NALU_TYPE_PICTURE_PARAMETER_SET)

        self.pic_parameter_set_id = ExpGolombUE()
        self.seq_parameter_set_id = ExpGolombUE()
        self.entropy_coding_mode_flag = FixedLenUint(1)
        self.bottom_field_pic_order_in_frame_present_flag = FixedLenUint(1)
        self.num_slice_groups_minus1 = ExpGolombUE()

        self.slice_group_map_type = ExpGolombUE()
        self.run_length_minus1 = [ExpGolombUE()] * (self.num_slice_groups_minus1.value + 1)

        self.top_left = [ExpGolombUE()] * self.num_slice_groups_minus1.value
        self.bottom_right = [ExpGolombUE()] * self.num_slice_groups_minus1.value

        self.slice_group_change_direction_flag = FixedLenUint(1)
        self.slice_group_change_rate_minus1 = ExpGolombUE()

        self.pic_size_in_map_units_minus1 = ExpGolombUE()
        bit_count = math.ceil( math.log2(self.num_slice_groups_minus1.value + 1))
        self.slice_group_id = [FixedLenUint(bit_count)] * (self.pic_size_in_map_units_minus1.value + 1)

        self.num_ref_idx_l0_default_active_minus1 = ExpGolombUE()
        self.num_ref_idx_l1_default_active_minus1 = ExpGolombUE()
        self.weighted_pred_flag = FixedLenUint(1)
        self.weighted_bipred_idc = FixedLenUint(2)
        self.pic_init_qp_minus26 = ExpGolombSE()
        self.pic_init_qs_minus26 = ExpGolombSE()
        self.chroma_qp_index_offset = ExpGolombSE()

        self.deblocking_filter_control_present_flag = FixedLenUint(1)
        self.constrained_intra_pred_flag = FixedLenUint(1)
        self.redundant_pic_cnt_present_flag = FixedLenUint(1)

        self.transform_8x8_mode_flag = FixedLenUint(1)
        self.pic_scaling_matrix_present_flag = FixedLenUint(1)

        self.second_chroma_qp_index_offset = ExpGolombSE()

    def _encode(self):
        self.codec.append(self.pic_parameter_set_id)
        self.codec.append(self.seq_parameter_set_id)
        self.codec.append(self.entropy_coding_mode_flag)
        self.codec.append(self.bottom_field_pic_order_in_frame_present_flag)
        self.codec.append(self.num_slice_groups_minus1)

        if self.num_slice_groups_minus1.value > 0:
            self.codec.append(self.slice_group_map_type)

            if  SLICE_GROUP_MAP_TYPE_INTERLEAVED == self.slice_group_map_type.value:
                for x in self.run_length_minus1: self.codec.append(x)

            elif SLICE_GROUP_MAP_TYPE_FOREGROUND_LEFTOVER == self.slice_group_map_type.value:
                for x in range(self.num_slice_groups_minus1.value):
                    self.codec.append(self.top_left[x])
                    self.codec.append(self.bottom_right[x])

            elif self.slice_group_map_type.value in [SLICE_GROUP_MAP_TYPE_CANGING, SLICE_GROUP_MAP_TYPE_CANGING1, SLICE_GROUP_MA
                self.codec.append(self.slice_group_change_direction_flag)
                self.codec.append(self.slice_group_change_rate_minus1)

            elif SLICE_GROUP_MAP_TYPE_EXPLICIT_ASSIGNMENT == self.slice_group_map_type.value:
                self.codec.append(self.pic_size_in_map_units_minus1)
                for x in self.slice_group_id: self.codec.append(x)

        self.codec.append(self.num_ref_idx_l0_default_active_minus1)
        self.codec.append(self.num_ref_idx_l1_default_active_minus1)
        self.codec.append(self.weighted_pred_flag)
        self.codec.append(self.weighted_bipred_idc)
        self.codec.append(self.pic_init_qp_minus26)
        self.codec.append(self.pic_init_qs_minus26)
        self.codec.append(self.chroma_qp_index_offset)
        self.codec.append(self.deblocking_filter_control_present_flag)
        self.codec.append(self.constrained_intra_pred_flag)
        self.codec.append(self.redundant_pic_cnt_present_flag)

        self.codec.append(self.transform_8x8_mode_flag)
        self.codec.append(self.pic_scaling_matrix_present_flag)
        self.codec.append(self.second_chroma_qp_index_offset)

        # print("NALU:", repr(self.data))

class RefPicListModification:
    def __init__(self, slice_type):
        self.slice_type = slice_type
        self.ref_pic_list_modification_flag_10 = FixedLenUint(1)

    def encode(self, glb):
        if self.slice_type % 5 not in [2, 4]:
            glb.append(self.ref_pic_list_modification_flag_10)

class DecRefPicMarking:
    def __init__(self, idr_pic_flag=False):
        self.idr_pic_flag = idr_pic_flag

        self.no_output_of_prior_pics_flag = FixedLenUint(1)
        self.long_term_refreence_flag = FixedLenUint(1)
        self.adaptive_ref_pic_marking_mode_flag = FixedLenUint(1)

    def encode(self, codec):
        if self.idr_pic_flag:
            codec.append(self.no_output_of_prior_pics_flag)
            codec.append(self.long_term_refreence_flag)
        else:
            codec.append(self.adaptive_ref_pic_marking_mode_flag)

class SliceHeader:
    def __init__(self, nalu_header: NALUHeader, sps: SequenceParameterSet, pps: PictureParameterSet):
        self.nalu_header = nalu_header
        self.sps = sps
        self.pps = pps
        self.idr_flag = (NALU_TYPE_CODED_SLICE_IDR_PIC == nalu_header.nalu_type.value)

        self.first_mb_in_slice = ExpGolombUE()
        self.slice_type = ExpGolombUE()
        self.pic_parameter_set_id =  ExpGolombUE()
        self.colour_plane_id = FixedLenUint(2)

        # used as an identifier for pictures and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream
        self.frame_num = FixedLenUint(self.sps.log2_max_frame_num_minus4.value + 4)

        self.field_pic_flag = FixedLenUint(1)
        self.bottom_field_flag = FixedLenUint(1)

        self.idr_pic_id = ExpGolombUE()

        # The length of the pic_order_cnt_lsb syntax element is log2_max_pic_order_cnt_lsb_minus4 + 4 bits
        self.pic_order_cnt_lsb = FixedLenUint(sps.log2_max_pic_order_cnt_lsb_minus4.value + 4)

        self.delta_pic_order_cnt_bottom = ExpGolombSE()
        self.delta_pic_order_cnt = [ExpGolombSE(), ExpGolombSE()]

        self.redundant_pic_cnt = ExpGolombUE()
        self.direct_spatial_mv_pred_flag = FixedLenUint(1)
        self.num_ref_idx_active_override_flag = FixedLenUint(1)
        self.num_ref_idx_l0_active_minus1 = ExpGolombUE()
        self.num_ref_idx_l1_active_minus1 = ExpGolombUE()

        self.cabac_init_idc = ExpGolombUE()
        self.slice_qp_delta = ExpGolombSE()

        self.sp_for_switch_flag = FixedLenUint(1)
        self.slice_qs_delta = ExpGolombSE()

        self.disable_deblocking_filter_idc = ExpGolombUE()
        self.slice_alpha_c0_offset_div2 = ExpGolombSE()
        self.slice_beta_offset_div2 = ExpGolombSE()

        self.slice_group_change_cycle = ExpGolombUE()

    def encode(self, codec: ExpGolombCodec):
        codec.append(self.first_mb_in_slice)
        codec.append(self.slice_type)
        codec.append(self.pic_parameter_set_id)

        if 1 == self.sps.separate_colour_plane_flag.value:
            codec.append(self.colour_plane_id)

        # If the current picture is an IDR picture, frame_num shall be equal to 0
        codec.append(self.frame_num)

        if not self.sps.frame_mbs_only_flag.value:
            codec.append(self.field_pic_flag)
            if self.field_pic_flag.value:
                codec.append(self.bottom_field_flag)

        if self.idr_flag:
            codec.append(self.idr_pic_id)

        if 0 == self.sps.pic_order_cnt_type.value:
            codec.append(self.pic_order_cnt_lsb)
            if self.pps.bottom_field_pic_order_in_frame_present_flag.value and not self.field_pic_flag.value:
                codec.append(self.delta_pic_order_cnt_bottom)

        if 1 == self.sps.pic_order_cnt_type.value and not self.sps.delta_pic_order_always_zero_flag.value:
            codec.append(self.delta_pic_order_cnt[0])
            if self.pps.bottom_field_pic_order_in_frame_present_flag.value and not self.field_pic_flag.value:
                codec.append(self.delta_pic_order_cnt[1])

        if self.pps.redundant_pic_cnt_present_flag.value:
            codec.append(self.redundant_pic_cnt)

        st = self.slice_type.value
        if st in SLICE_TYPE_P or st in SLICE_TYPE_SP or st in SLICE_TYPE_B:
            codec.append(self.num_ref_idx_active_override_flag)
            if self.num_ref_idx_active_override_flag.value:
                codec.append(self.num_ref_idx_l0_active_minus1)
                if st in SLICE_TYPE_B:
                    codec.append(self.num_ref_idx_l1_active_minus1)

        if 0:
            ...
        else:
            ref = RefPicListModification(self.slice_type.value)
            ref.encode(codec)

        if self.nalu_header.nalu_ref_idc.value:
            drpm = DecRefPicMarking(self.idr_flag)
            drpm.encode(codec)
        if self.pps.entropy_coding_mode_flag.value and st not in SLICE_TYPE_I and st not in SLICE_TYPE_SI:
            codec.append(self.cabac_init_idc)

        codec.append(self.slice_qp_delta)

        if st in SLICE_TYPE_SP or st in SLICE_TYPE_SI:
            if st in SLICE_TYPE_SP:
                codec.append(self.sp_for_switch_flag)
            codec.append(self.slice_qs_delta)

        if self.pps.deblocking_filter_control_present_flag.value:
            codec.append(self.disable_deblocking_filter_idc)

            if self.disable_deblocking_filter_idc.value > 2:
                raise Exception("disable_deblocking_filter_idc should be in range of 0 to 2.")

            if self.disable_deblocking_filter_idc.value != 1:
                codec.append(self.slice_alpha_c0_offset_div2)
                codec.append(self.slice_beta_offset_div2)

        if self.pps.num_slice_groups_minus1.value > 0 and 3 <= self.pps.slice_group_map_type.value <= 5:
            codec.append(self.slice_group_change_cycle)


class SliceData:
    def __init__(self, mb_type, sps: SequenceParameterSet, pps: PictureParameterSet, sh: SliceHeader):
        self.sps = sps
        self.pps = pps
        self.entropy_coding_mode_flag = pps.entropy_coding_mode_flag.value
        self.first_mb_in_slice = sh.first_mb_in_slice.value
        self.slice_type = sh.slice_type.value

        # MbaffFrameFlag = ( mb_adaptive_frame_field_flag && !field_pic_flag )
        self.mbaff_frame_flag = (sps.mb_adaptive_frame_field_flag.value and not sh.field_pic_flag.value)

        # PicWidthInMbs = pic_width_in_mbs_minus1 + 1
        self.pic_width_in_mbs = sps.pic_width_in_mbs_minus1.value + 1

        # PicHeightInMapUnits = pic_height_in_map_units_minus1 + 1
        self.pic_height_in_map_units = sps.pic_height_in_map_units_minus1.value + 1

        # FrameHeightInMbs = ( 2 - frame_mbs_only_flag ) * PicHeightInMapUnits
        self.frame_height_in_mbs = (2 - sps.frame_mbs_only_flag.value) * self.pic_height_in_map_units

        # PicHeightInMbs = FrameHeightInMbs / ( 1 + field_pic_flag )
        self.pic_height_in_mbs = self.frame_height_in_mbs / (1 + sh.field_pic_flag.value)

        # PicSizeInMbs = PicWidthInMbs * PicHeightInMbs
        self.pic_size_in_mbs = self.pic_width_in_mbs * self.pic_height_in_mbs

        # PicSizeInMapUnits = PicWidthInMbs * PicHeightInMapUnits
        self.pic_size_in_map_units = self.pic_width_in_mbs * self.pic_height_in_map_units

        self.mb_skip_run = ExpGolombUE()

        self.map_unit_to_slice_group_map = []
        self.macroblock_layer = [MacroBlockLayer(mb_type, self.sps, self.pps) for x in range(2560)]

    def encode(self, codec):
        if self.pps.entropy_coding_mode_flag.value:
            # cabac_alignment_one_bit
            while not codec.byte_aligned():
                codec.append(FixedLenUint(1, 1))
        for x in self.macroblock_layer:
            x.encode(codec)

    def next_mb_address(self,  curr_mb_address):
        i = curr_mb_address + 1
        while i <  self.pic_size_in_mbs: # and MbToSliceGroupMap[i] != MbToSliceGroupMap[curr_mb_address]
            i = + 1
        return i

    def get_map_unit_to_slice_group_map(self):
        if SLICE_GROUP_MAP_TYPE_INTERLEAVED == self.pps.slice_group_map_type:
            group = 0
            i = 0

            for group in range(self.pps.num_slice_groups_minus1.value + 1):
                if i < self.pic_size_in_map_units:
                    for j in range(self.pps.run_length_minus1[group].value + 1):
                        if i + j < self.pic_size_in_map_units:
                            self.map_unit_to_slice_group_map[i + j] = group
                            j += 1
                    i += self.pps.run_length_minus1[group].value + 1


class MacroBlockLayer:
    def __init__(self, mb_type, sps, pps: PictureParameterSet):
        self.pps = pps
        self.mb_type = ExpGolombUE(mb_type)

        # bit_depth_luma_minus8 shall be in the range of 0 to 6
        self.bit_depth_luma_minus8 = sps.bit_depth_luma_minus8.value
        assert (0 <= self.bit_depth_luma_minus8 <= 6)

        #  bit_depth_chroma_minus8 shall be in  the range of 0 to 6
        self.bit_depth_chroma_minus8 = sps.bit_depth_luma_minus8.value
        assert (0 <= self.bit_depth_luma_minus8 <= 6)

        self.sub_width_c, self.sub_height_c = sps.get_sub_width_height_c()
        if self.sub_width_c and self.sub_height_c:
            self.mb_width_c = int(16 / self.sub_width_c)
            self.mb_height_c = int(16 / self.sub_height_c)
        else:
            self.mb_width_c = 0
            self.mb_height_c = 0

            # pcm_sample_luma bit count
        self.bit_depth_y = 8 + self.bit_depth_luma_minus8
        self.qp_bd_offset_y = 6 + self.bit_depth_luma_minus8

        #if 25 == self.mb_type.value:
            # each macroblock is comprised of one 16x16 luma array
        self.pcm_sample_luma = [FixedLenUint(self.bit_depth_y) for x in range(256)] if 25 == self.mb_type.value else []

        # pcm_sample_chroma bit count
        self.bit_depth_c = 8 + self.bit_depth_chroma_minus8
        self.pcm_sample_chroma = [FixedLenUint(self.bit_depth_c) for x in range(2 * self.mb_width_c * self.mb_height_c)] if 25 == self.mb_type.value else

        self.transform_size_8x8_flag = FixedLenUint(1)
        self.coded_block_pattern = ExpGolombUE(0)

    def encode(self, codec: ExpGolombCodec):
        codec.append(self.mb_type)

        if 25 == self.mb_type.value:
            # pcm_alignment_zero_bit
            while not codec.byte_aligned():
                codec.append(FixedLenUint(1, 0))

            for x in self.pcm_sample_luma:
                codec.append(x)

            for x in self.pcm_sample_chroma:
                codec.append(x)
        else:
            if self.pps.transform_8x8_mode_flag.value and 0 == self.mb_type.value:
                codec.append(self.transform_size_8x8_flag)

            mb_pred = MacroblockPrediction()
            mb_pred.encode(codec)

            codec.append(self.coded_block_pattern)

class MacroblockPrediction:
    def __init__(self, chroma_array_type=0):
        self.luma_blk_count = 4
        iter = range(self.luma_blk_count)

        self.prev_pred_mode_flag = [FixedLenUint(1) for x in iter]
        self.rem_pred_mode = [FixedLenUint(3) for x in iter]

        self.chroma_array_type = chroma_array_type
        self.intra_chroma_pred_mode = ExpGolombUE()

    def encode(self, codec):
        #for x in range(self.luma_blk_count):
        #    glb.append(self.prev_pred_mode_flag[x])

        #    if self.prev_pred_mode_flag[x].value:
         #       glb.append(self.rem_pred_mode[x])

        #glb.append(self.intra_chroma_pred_mode)
        ...


class SliceLayerWithoutPartitioningRBSP(NALU):
    def __init__(self, nalu_type, mb_type, sps, pps):
        super(SliceLayerWithoutPartitioningRBSP, self).__init__(nalu_type)
        self.sh = SliceHeader(self.header, sps, pps)
        self.d = SliceData(mb_type, sps, pps, self.sh)

    def _encode(self):
        self.sh.encode(self.codec)
        self.d.encode(self.codec)

class UserDataUnregistered:
    def __init__(self,):
        self.uuid_iso_iec_11587 = [FixedLenUint(32) for x in range(4)]
        self.user_data_payload_byte = ""


class SupplementalEnhancementInformation(NALU):
    def __init__(self,):
        super(SupplementalEnhancementInformation, self).__init__(NALU_TYPE_SUPPLEMENTAL_ENHANCEMENT_INFO)
        self.last_payload_type_byte = FixedLenUint(8)
        self.payload_size = 0
        self.last_payload_size_byte = FixedLenUint(8)
        self.user_data_unregistered = UserDataUnregistered()

    def _encode(self):
        self.codec.append(self.last_payload_type_byte)

        if SEI_PT_USER_DATA_UNREGISTERED == self.last_payload_type_byte.val
        if SEI_PT_USER_DATA_UNREGISTERED == self.last_payload_type_byte.value:
            self.payload_size = 16 + len(self.user_data_unregistered.user_data_payload_byte)

        d, self.last_payload_size_byte.value = divmod(self.payload_size, 255)
        for x in range(d):  self.codec.append(FixedLenUint(8, 255))
        self.codec.append(self.last_payload_size_byte)

        if SEI_PT_USER_DATA_UNREGISTERED == self.last_payload_type_byte.value:
            for x in self.user_data_unregistered.uuid_iso_iec_11587: self.codec.append(x)
            for x in self.user_data_unregistered.user_data_payload_byte: self.codec.append(FixedLenUint(8, ord(x)))
```

ghf.py源码如下：

```
# author: yangyiyin
# 2022/2/14 chengdu

import random
from avc import *

f = open("test", "wb")
sps = SequenceParameterSet()
pps = PictureParameterSet()
sei = SupplementalEnhancementInformation()

sps.header.nalu_ref_idc.value = 3
sps.profile_idc.value = 100
sps.level_idc.value = 30
sps.chroma_format_idc.value = CHROMA_FORMAT_420
sps.log2_max_pic_order_cnt_lsb_minus4.value = 2
sps.num_ref_frames_in_pic_order_cnt_cycle.value = 4
sps.pic_width_in_mbs_minus1.value = 43
sps.pic_height_in_map_units_minus1.value = 35
sps.frame_mbs_only_flag.value = 1
sps.direct_8x8_inference_flag.value = 1

sps.vui_parameters_present_flag.value = 1
sps.vui_parameters.video_signal_type_present_flag.value = 1
sps.vui_parameters.video_format.value = 5
sps.vui_parameters.video_full_range_flag.value = 1
sps.vui_parameters.timing_info_present_flag.value = 1
sps.vui_parameters.num_units_in_tick.value = 1
sps.vui_parameters.time_scale.value = 50
sps.vui_parameters.bitstream_restriction_flag.value = 1
sps.vui_parameters.log2_max_mv_length_horizontal.value = 10
sps.vui_parameters.log2_max_mv_length_vertical.value = 10
sps.vui_parameters.max_num_reorder_frames.value = 2
sps.vui_parameters.max_dec_frame_buffering.value = 4
sps.encode()
print(sps.data)

pps.header.nalu_ref_idc.value = 3
pps.entropy_coding_mode_flag.value = 0
pps.num_ref_idx_l0_default_active_minus1.value = 2
pps.weighted_pred_flag.value = 0
pps.weighted_bipred_idc.value = 2
pps.pic_init_qp_minus26.value = -3
pps.chroma_qp_index_offset.value = -2
pps.deblocking_filter_control_present_flag.value = 0
pps.transform_8x8_mode_flag.value = 1
pps.second_chroma_qp_index_offset.value = -2
pps.encode()
print(pps.data)

#sei.header.nalu_ref_idc.value = 3
```

```
sei.last_payload_type_byte.value = 5
sei.user_data_unregistered.uuid_iso_iec_11587 = [FixedLenUint(32, 11111), FixedLenUint(32, 22222), FixedLenUint(32, 33333), FixedLenUint(32, 444
sei.user_data_unregistered.user_data_payload_byte = "this is a string."
sei.encode()
print(sei.data)

slwr = SliceLayerWithoutPartitioningRBSP(NALU_TYPE_CODED_SLICE_IDR_PIC, 25, sps, pps)
slwr.header.nalu_ref_idc.value = 3
slwr.sh.first_mb_in_slice.value = 0
slwr.sh.slice_type.value = SLICE_TYPE_I[1]
slwr.sh.frame_num.value = 0
slwr.sh.idr_pic_id.value = 0
slwr.sh.disable_deblocking_filter_idc.value = 0

f.write(b"\x00\x00\x00\x01")
f.write(sps.data)
f.write(b"\x00\x00\x01")
f.write(pps.data)
f.write(b"\x00\x00\x01")
f.write(sei.data)

for c in range(10):
    f.write(b"\x00\x00\x01")
    for m, mb in enumerate(slwr.d.macroblock_layer):
        for idx, luma in enumerate(mb.pcm_sample_luma):
            if idx < 128:
                luma.value = c#random.randint(0, 255)
            else:
                luma.value = 20 * c
            if m % 2 and luma.value > 20:
                luma.value  -= 20
        if CHROMA_FORMAT_MONOCHROME != sps.chroma_format_idc.value:
            # print(len(mb.pcm_sample_chroma))
            for idx, chroma in enumerate(mb.pcm_sample_chroma):
                chroma.value = 25 * c if idx < 64 else 10 * c #random.randint(0, 255)
    slwr.encode()
    f.write(slwr.data)
    #  continue
    slwr1 = SliceLayerWithoutPartitioningRBSP(NALU_TYPE_CODED_SLICE_NON_IDR_PIC, 3, sps, pps)
    #slwr1.header.nalu_ref_idc.value = 1
    slwr1.sh.first_mb_in_slice.value = 0
    slwr1.sh.slice_type.value = SLICE_TYPE_P[1]
    slwr1.sh.disable_deblocking_filter_idc.value = 0
    for mb in range(24):
        f.write(b"\x00\x00\x01")
        slwr1.encode()
        f.write(slwr1.data)
```

也可以生成黑白色视频，修改chroma_format_idc参数值即可。