密级状态： 绝密(　)　　秘密(　)　　内部(　)　　公开(√)

Rockchip Intelligent Video Engine

（技术部，图形计算平台中心）

文件状态： [　] 正在修改 [√] 正式发布	当前版本：	V1.2
	作　者：	陈城
	编　辑：	刘雯君
	审　核：	熊伟
	完成日期：	2022-07-07

瑞芯微电子股份有限公司

Rockchips Semiconductor Co . , Ltd

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园 A 区 18 号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前 言

文档适用对象

- 软件开发工程师
- 技术支持工程师

适用平台

- RV1103
- RV1106

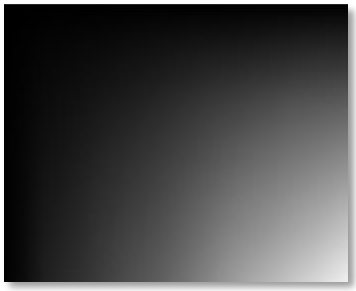
版本记录

版本信息	版本说明	日期	作者
V1.0	初始版本	2022.04.06	陈城
V1.1	增加示意图，完善接口说明	2022.04.26	陈城
V1.2	整理目录结构，增加仿射变换及创建图像金字塔接口说明	2022.07.07	陈城

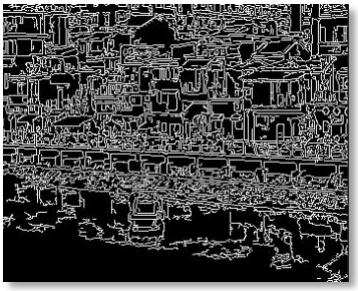
目 录

1. 概述

RKIVE(Rockchip Intelligent Video Engine)是瑞芯微媒体处理芯片视频图像分析中的硬件加速模块，用来加速视频图像分析，降低 CPU 占用。当前硬件版本集成 22 个运算模块，支持寄存器直接配置和链表批处理运行两种模式。主要应用于视频图像分析及周界入侵防范等场景。

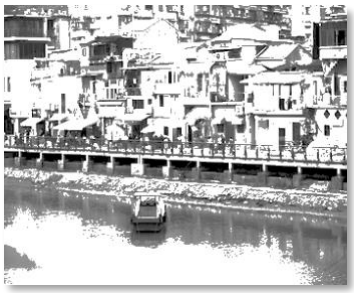


Integral
used as a quick and effective way of calculating the sum of pixel value



Canny Edge
uses a multi-stage algorithm to detect a wide range of edges

Histogram Equalization
processing of contrast adjustment using the image's histogram



Threshold
method of segmenting images, creating a binary images

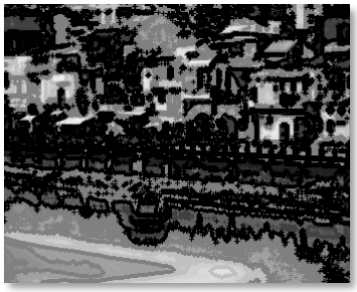
Filter
Filter image with a 5x5 kernel

Map
map one set of pixel values to another

Min Filter
erodes shapes on the image

Median Filter
makes the target pixel luminosity equal to the mean value in the running window

Max Filter
extends object boundaries on the image

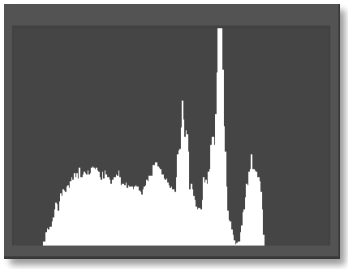


Erode
removes pixels on object boundaries

LBP
labels the pixels of an image by thresholding the neighborhood of each pixel

Cast
Convert pixel value range to another

Shi-tomasi corner
finds N strongest corners in the image by Shi-Tomasi method



Histogram
acts as a graphical representation of the luminance distribution

Sub-Abs
subtract two images and calculate the absolute value

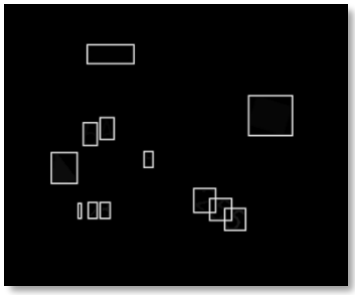
Sub-Shift
subtract two images and right shift 1 bit of the result

Add
combine two images with different weight

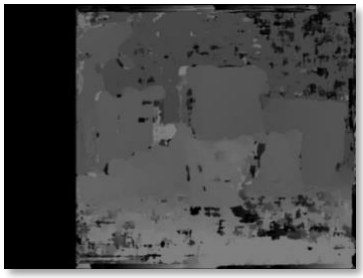
And
bitwise and of two images

Or
bitwise or of two images

Xor
bitwise xor of two images



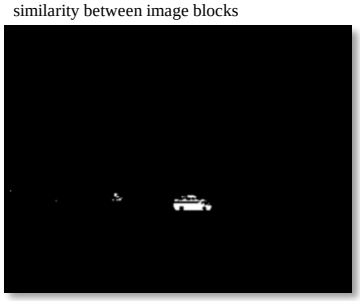
CCL
Connected-component labeling , identify blobs of pixels in an image



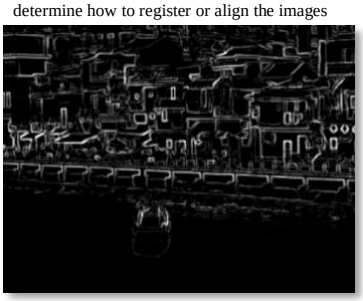
SAD
sum of absolute differences , a measure of the similarity between image blocks

NCC
normalized cross-correlation, can be used to determine how to register or align the images

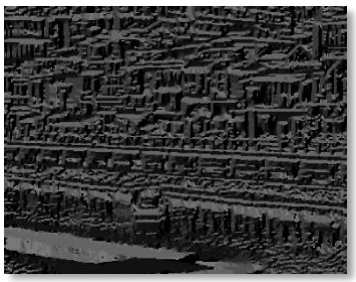
LK Optical Flow
optical flow estimation



GMM & BGM
Background subtraction, distinguish foreground objects from the background



Gradient Magnitude
measure how strong the change in image intensity is



Gradient Direction
a directional change in image intensity

2. 入门指南

2.1. 环境说明

2.1.1. 目录结构说明

RKIVE 软件包括以下几个部分，以 RV1103/RV1106 SDK 为例：

```
media/ive/ive/
├── CMakeLists.txt
├── include                    #RKIVE 相关头文件
│   ├── rk_comm_ive.h
│   ├── rk_ive.h
│   └── rk_mpi_ive.h
├── lib                        #RKIVE/RKIVS 相关库文件
│   ├── libivs.a               #RKIVS 静态库
│   ├── libivs.so              #RKIVS 动态库
│   ├── librve.a               #RKIVE 静态库
│   └── librve.so              #RKIVE 动态库
├── sample                     #RKIVE 计算图像直方图的 sample
│   ├── main.c
│   ├── sample_comm_mem.h
│   ├── sample_comm_rve.c
│   ├── sample_comm_rve.h
│   └── sample_rve_mem.c
└── simulator
#RKIVE simulator 基于 windows visual studio 的模拟开发环境
```

其中 simulator 文件中包含了 RKIVE 所有接口调用的参考示例，可基于 windows visual studio 环境进行验证开发，或移植到 SDK 开发板环境中运行。

RKIVE 设备驱动在开发板文件系统中的路径为/oem/usr/ko/rve.ko。运行前通过检查/dev/rve 节点是否存在，以此来确认设备驱动是否加载。如未配置开机自动加载，可以修改脚本/oem/usr/ko/insmod_ko.sh，在该脚本末尾添加 insmod rve.ko。

2.1.2. 输入输出缓存

RKIVE 功能接口要求的输入输出缓存类型包括 IVE_MEM_INFO_S、IVE_DATA_S 以及 IVE_DST_IMAGE_S。在不同的平台上根据不同的缓存类型分配内存，并填充相应的缓存结构体。以 RV1103/RV1106 SDK 为例，调用 RK_MPI_MMZ_Alloc 分配物理连续的内存块，而后调用 RK_MPI_MMZ_Free 释放已申请的内存。可参考 sample code 中 sample_comm_rve.c 文件封装的相应接口，用于包装输入输出缓存：

- SAMPLE_COMM_IVE_CreateImage
- SAMPLE_COMM_IVE_CreateMemInfo
- SAMPLE_COMM_IVE_CreateData

算子运行完成后，调用 MmzFree 接口，传入缓存对应的虚拟地址，释放对应的缓存。MmzFree 具体实现是通过缓存虚拟地址得到 MB_BLK 指针，调用 RK_MPI_MMZ_Free 完成内存释放。

2.2. 基本概念

- **行跨度（stride）或虚宽**

图像或二维数据每行占用的内存空间，行跨度一般是为了满足硬件对齐要求，设置的值为大于或等于宽度（width）的值。

- **地址对齐**

部分算子要求分配的输入输出缓存首地址 1byte 或 16byte 对齐，具体见 API 说明。

- **CPU cache**

当有 CPU 介入的情况下，在对算子申请的 cacheable 输入输出缓存内容做前处理或后处理时，需要在处理前调用 RK_MPI_MMZ_FlushCacheVaddrStart,并在处理完成后调用 RK_MPI_MMZ_FlushCacheVaddrEnd 刷新 cache。防止输入输出数据因为 cache 未刷新导致的错误。

- **一维数据**

一维数据是一块线性物理缓存，对应的结构体为 IVE_MEM_INFO_S，结构体中包含物理地址、虚拟地址及缓存大小等信息。

- **二维数据**

二维数据是一块特定宽度、高度及跨度信息的物理缓存，对应的结构体为 IVE_DATA_S。

- **二维图像**

二维图像是一块特定宽度、高度、跨度及数据通道的物理缓存，对应的结构体为 IVE_DST_IMAGE_S。可用来表示拥有一个或多个通道的图像数据，不同通道的宽度和高度一致，跨度、起始虚拟地址及起始物理地址分别存放在对应的数组中。

- **任务 ID（handle）**

系统为每一个算子调用分配的任务 ID。该任务 ID 为大于零的常数，若返回的任务 ID 小于零，表明任务创建失败。

- **任务查询（query）**

将算子调用返回的任务 ID，传入 RK_MPI_IVE_Query 接口，查询当前节点任务是否完成。

- **同步模式及异步模式（bInstant）**

每一个算子调用 API 都带有 bInstant 参数。当 bInstant 设置为 RK_TRUE，则当前调用阻塞直至硬件执行完成返回结果。当 bInstant 设置为 RK_FALSE，则当前调用立即返回，并与后续的算子组成链表执行批处理模式。这样可以减少中断次数，降低 CPU 负荷，提升性能。

2.3. 参考示例

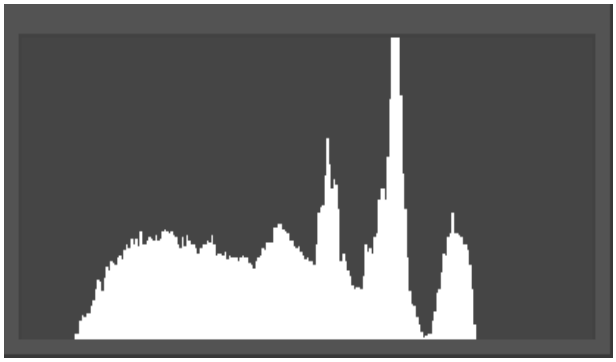


图 2-1

以统计图像直方图为参考示例，示例代码主要包括以下几个部分：

- **初始化**

  调用 RK_MPI_IVE_Init 进行初始化。

- **内存分配**

  调用 RK_MPI_MMZ_Alloc 对输入图像及输出的直方图缓存分配内存，申请的内存类型为 RK_MMZ_ALLOC_CACHEABLE。

- **运行**

  调用 RK_MPI_IVE_Hist 触发硬件开始执行直方图统计任务。

- **查询**

  调用 RK_MPI_IVE_Query 查询直方图统计任务运行情况，等待任务完成。

- **后处理**

  CPU 打印硬件输出到 ddr 的直方图统计结果。CPU 访问期间，调用 RK_MPI_MMZ_FlushCacheVaddrStart 和 RK_MPI_MMZ_FlushCacheVaddrEnd 刷新 CPU cache，防止数据因 cache 刷新影响导致的错误。

- **结束**

  调用 RK_MPI_MMZ_Free 释放之前分配的缓存，调用 RK_MPI_IVE_Deinit 进行反初始化。

Sample Code:

```
#include <rk_mpi_ive.h>
#include <rk_mpi_mmz.h>

int main(void) {
    RK_S32 s32Ret = 0;
    bool bInstant = false;
    bool bBlock = true;
    bool bFinish = false;

    RK_U32 u32Width = 16;
    RK_U32 u32Height = 16;
    RK_U32 u32Size = 0;

    IVE_HANDLE IveHandle = 0;
    IVE_SRC_IMAGE_S stSrc = {0};
    IVE_DST_MEM_INFO_S stHist = {0};

    MB_BLK stMB = NULL;
    RK_U32* pu32Hist = NULL;

    // initialize IVE context
    s32Ret = RK_MPI_IVE_Init();
    if (s32Ret < 0) {
        printf(stderr, "RK_MPI_IVE_Init failed\n");
        goto End;
    }

    // allocate memory for input image
    u32Size = u32Width * u32Height;
    s32Ret = RK_MPI_MMZ_Alloc(&stMB, u32Size, RK_MMZ_ALLOC_TYPE_CMA);
    if (s32Ret == 0) {
        stSrc.au64PhyAddr[0] = RK_MPI_MB_Handle2PhysAddr(stMB);
        stSrc.au64VirAddr[0] = RK_MPI_MB_Handle2VirAddr(stMB);
        stSrc.au32Stride[0] = u32Width;
        stSrc.u32Width = u32Width;
        stSrc.u32Height = u32Height;
    } else {
        printf(stderr, "Create input image mem failed\n");
        goto End;
    }

    // allocate memory for histogram buffer
    u32Size = IVE_HIST_NUM * sizeof(RK_U32);
    s32Ret = RK_MPI_MMZ_Alloc(&stMB, u32Size, RK_MMZ_ALLOC_TYPE_CMA);

    if (s32Ret == 0) {
        stHist.u64PhyAddr = RK_MPI_MB_Handle2PhysAddr(stMB);
        stHist.u64VirAddr = RK_MPI_MB_Handle2VirAddr(stMB);
        stHist.u32Size = u32Size;
    } else {
        printf(stderr, "Create hist mem info failed\n");
        goto End;
    }

    // run histogram
    s32Ret = RK_MPI_IVE_Hist(&IveHandle, & stSrc, &stHist, bInstant);
    if (s32Ret < 0) {
```

```
        printf(stderr, "RK_MPI_IVE_Hist failed\n");
        goto End;
    }

    // wait until histogram finished
    s32Ret = RK_MPI_IVE_Query(IveHandle, &bFinish, bBlock);
    while (ERR_IVE_QUERY_TIMEOUT == s32Ret) {
        usleep(100);
        s32Ret = RK_MPI_IVE_Query(IveHandle, &bFinish, bBlock);
    }

    if (s32Ret < 0) {
        printf(stderr, "RK_MPI_IVE_Query failed\n");
        goto End;
    }

    // get histogram result
    RK_MPI_MMZ_FlushCacheVaddrStart(stHist.u64VirAddr, stHist.u32Size, RK_MMZ_ALLOC_TYPE_CMA);

    pu32Hist = (RK_U32)stHist.u64VirAddr;
    printf("Histogram test, hist[0] = %d\n", pu32Hist[0]);

    RK_MPI_MMZ_FlushCacheVaddrEnd(stHist.u64VirAddr, stHist.u32Size, RK_MMZ_ALLOC_TYPE_CMA);

End:
    // free memory
    if (stSrc.au64VirAddr[0] != 0) {
        stMB = RK_MPI_MB_VirAddr2Handle((void*)stSrc.au64VirAddr[0]);
        RK_MPI_MMZ_Free(stMB);

        stSrc.au64VirAddr[0] = 0;
        stSrc.au64PhyAddr[0] = 0;
    }

    if (stHist.u64VirAddr != 0) {
        stMB = RK_MPI_MB_VirAddr2Handle((void*)stHist.u64VirAddr);
        RK_MPI_MMZ_Free(stMB);

        stHist.u64VirAddr = 0;
        stHist.u64PhyAddr = 0;
    }

    // free IVE context
    s32Ret = RK_MPI_IVE_Deinit();
    if (s32Ret < 0) {
        printf(stderr, "RK_MPI_IVE_Deinit failed\n");
    }

    return 0;
}
```

3. Proc 调试信息

3.1. 概述

调试信息存储在 Linux 下的 proc 文件系统，实时记录硬件运行信息，供问题定位及性能分析使用。

- 文件节点

  /proc/rve

- 使用说明

  打开、关闭调试信息输出，使用以下命令：

  # echo mon > /proc/rve/debug

  [311610.143190] rve_debugger: close monitor!

  # echo mon > /proc/rve/debug

  [311611.686203] rve_debugger: open monitor!

3.2. Proc 信息说明

- 调试信息：
  - scheduler：硬件处理器实体
  - pd_ref：power 引用计数统计
  - total_int_cnt: 硬件中断次数统计
  - rd_bandwidth: 读取 ddr 数据量
  - wr_bandwidth：写入 ddr 数据量
  - cycle_cnt/s： 每秒消耗的 cycle 数

- 查看当前任务状态信息：

```
# cat /proc/rve/scheduler_status
num of scheduler = 1
==================================
scheduler[0]: rve
----------------------------------
pd_ref = 17
total_int_cnt = 17
rd_bandwidth: 256 bytes/s  wr_bandwidth: 1024 bytes/s
cycle_cnt/s: 685
```

- 记录最近 10 个任务的状态信息：

```
# cat /proc/rve/load
num of scheduler = 1
================= load ==================
scheduler[0]: rve
        load = 0
---------- PID INFO ----------
        [pid: 3958] hw_time_total = 0 us
                last_job_rd_bandwidth: 256 bytes/s
                last_job_wr_bandwidth: 1024 bytes/s
                last_job_cycle_cnt/s: 685
```

3.3. 错误码

当运行失败时，请检查返回的错误码，错误码具体描述如下表所示：

错误代码	宏定义	描述
0x40	ERR_IVE_SYS_TIMEOUT	系统超时
0x41	ERR_IVE_QUERY_TIMEOUT	查询超时
0x42	ERR_IVE_OPEN_FILE	打开文件超时
0x43	ERR_IVE_READ_FILE	读取文件超时
0x44	ERR_IVE_WRITE_FILE	写入文件超时
0x45	ERR_IVE_BUS_ERR	总线错误
0x46	ERR_IVE_ILLEGAL_PARAMS	非法参数
0x47	ERR_IVE_DEVICE_ERROR	设备驱动故障
0x48	ERR_IVE_NOT_SUPPORT	不支持的操作
0x49	ERR_IVE_ILLEGAL_STMEM	非法的辅助内存

4. 图像格式示意图

YCbCr 4:0:0 format

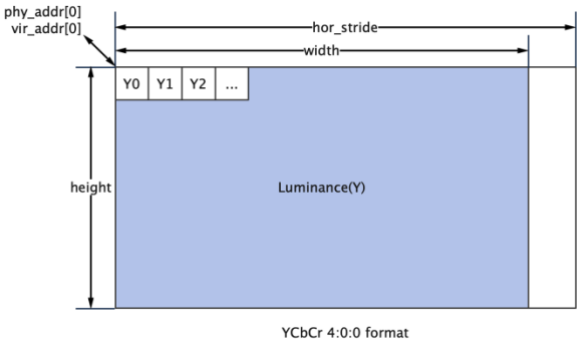


图 4-1

YUV420SP format

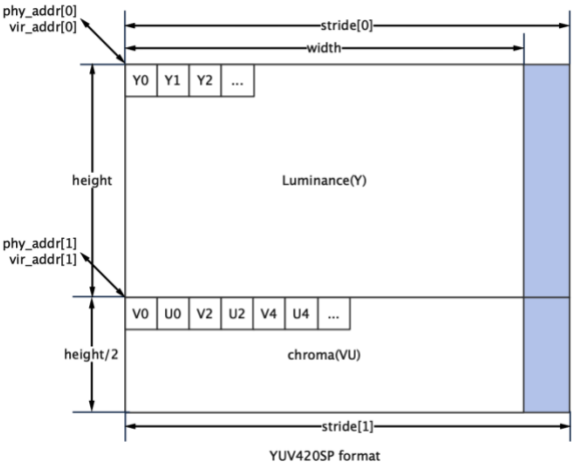


图 4-2

YUV420P format

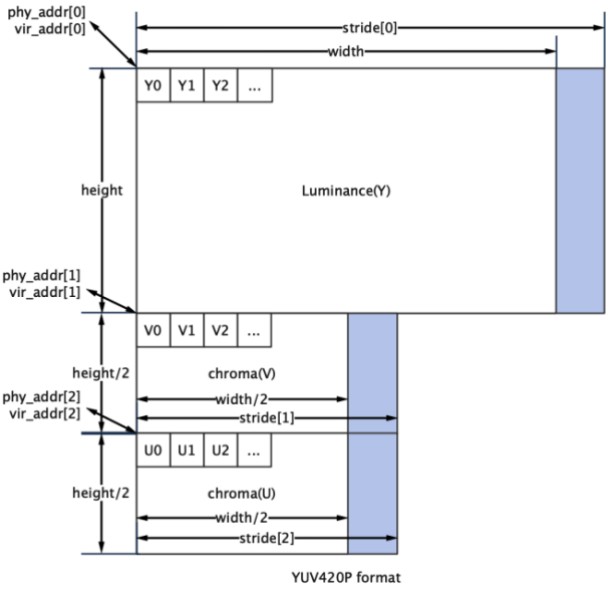


图 4-3

YUV422SP format

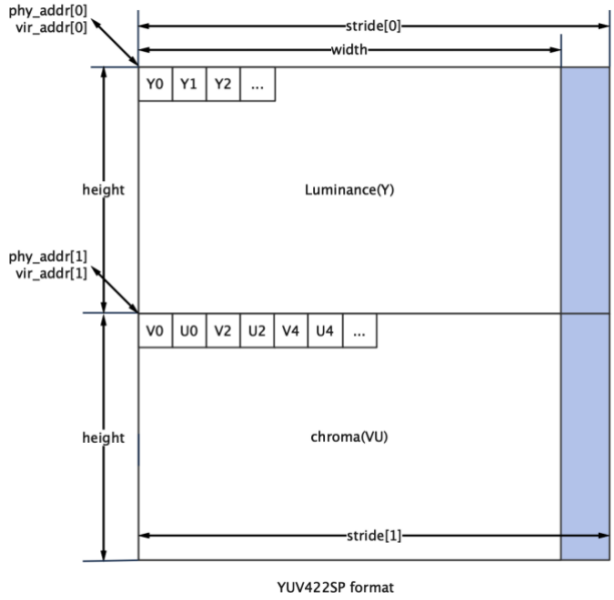


图 4-4

YUV422P format

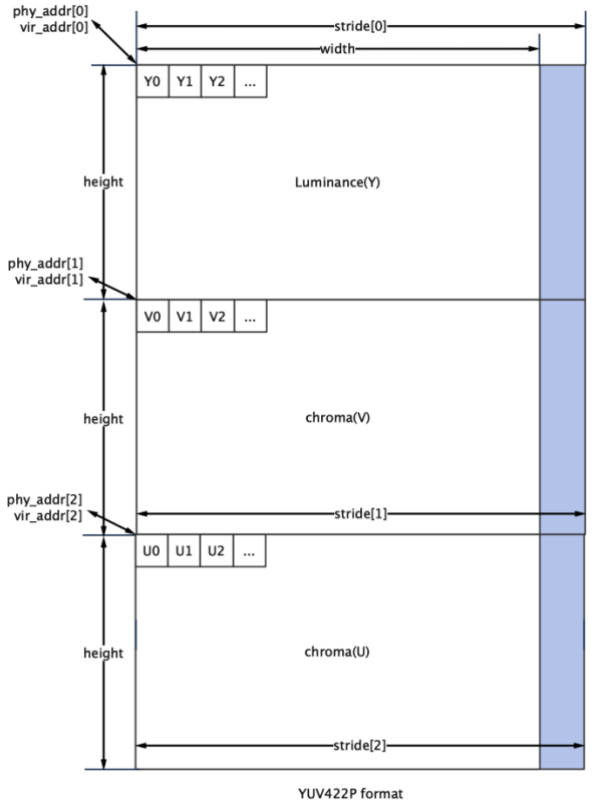


图 4-5

U8C1 format

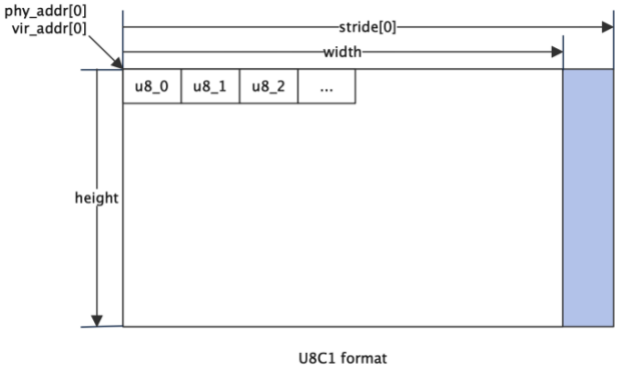


图 4-6

U8C3 planar format

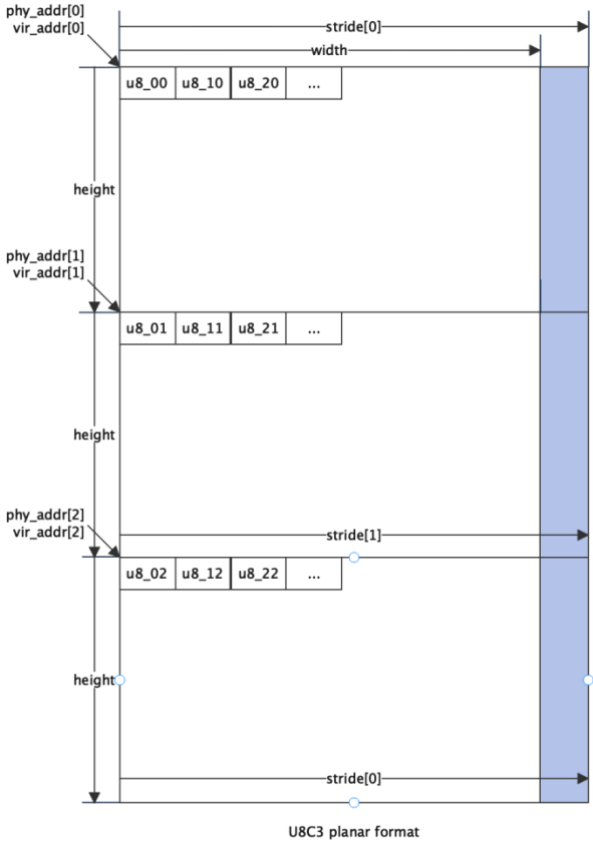


图 4-7

U8C3 package format

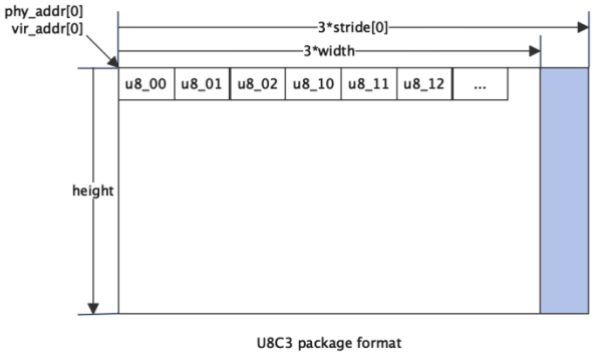


图 4-8

5. API 参考

RKIVE 提供以下功能接口：

- RK_MPI_IVE_Init: 完成 IVE 上下文创建及必要的初始化工作。
- RK_MPI_IVE_Deinit: 完成 IVE 上下文销毁。
- RK_MPI_IVE_CvtImageToData: 将单个 planar 图像类型转换为类型。
- RK_MPI_IVE_CvtDataToImage: 将二维数据类型转换为单个 planar 图像类型。
- RK_MPI_IVE_CvtImageToMemInfo: 将单个 planar 图像类型转换为一维数据类型。
- RK_MPI_IVE_DMA: 直接内存访问，支持快速拷贝、间隔拷贝、内存填充。
- RK_MPI_IVE_Filter: 5X5 模板滤波任务，通过配置不同的模板系数，可以实现不同的滤波。
- RK_MPI_IVE_CSC: 色彩空间转换，可实现 YUV、RGB、HSV 的色彩空间转换。
- RK_MPI_IVE_Sobel: 3X3 或 5X5 模板 SOBEL-LIKE 梯度计算。
- RK_MPI_IVE_MagAndAng: 计算幅值幅角。
- RK_MPI_IVE_Dilate: 图像膨胀，图像形态学操作。
- RK_MPI_IVE_Erode: 图像腐蚀，图像形态学操作。
- RK_MPI_IVE_Add: 两张灰度图像加权相加操作。
- RK_MPI_IVE_And: 两张灰度图像相与操作。
- RK_MPI_IVE_Sub: 两张灰度图像相减操作。
- RK_MPI_IVE_Or: 两张灰度图像相或操作。
- RK_MPI_IVE_Xor: 两张灰度图像相异或操作。
- RK_MPI_IVE_Integ: 图像积分图计算。
- RK_MPI_IVE_Hist: 图像直方图计算。
- RK_MPI_IVE_Thresh: 图像 8 位灰度图二值化操作。
- RK_MPI_IVE_Thresh_U16: 16 位数据到 8 位数据阈值二值化操作。
- RK_MPI_IVE_Thresh_S16: 带符号位的 16 位数据到 8 位数据的阈值二值化操作。
- RK_MPI_IVE_16BitTo8Bit: 16 位数据到 8 位数据的线性转换。
- RK_MPI_IVE_8BitTo8Bit: 8 位数据到 8 位数据的线性转换。
- RK_MPI_IVE_OrdStatFilter: 图像中值滤波，最小值滤波，最大值滤波。
- RK_MPI_IVE_Map: 根据查找表进行图像像素值映射。
- RK_MPI_IVE_EqualizeHist: 图像直方图均衡。

- RK_MPI_IVE_NCC: 两张相同分辨率灰度图像的归一化互相关系数。
- RK_MPI_IVE_CCL: 二值图像的连通区域标记。
- RK_MPI_IVE_GMM: 创建高斯混合背景模型，进行前景背景分离操作，参考 OPENCV 的 MOG。
- RK_MPI_IVE_GMM2: 创建高斯混合背景模型，进行前景背景分离操作，参考 OPENCV 的 MOG2。
- RK_MPI_IVE_CannyEdge: 提取灰度图像边缘信息。
- RK_MPI_IVE_LBP: 计算图像 LBP 特征。
- RK_MPI_IVE_NormGrad: 图像归一化梯度计算，梯度分量均归一化到 S8。
- RK_MPI_IVE_LKOpticalFlowPyr: LK 光流计算（外部建立金字塔）。
- RK_MPI_IVE_LKOpticalFlow: LK 光流计算（内部建立金字塔）。
- RK_MPI_IVE_STCandiCorner: 图像角点检测第一步，计算角点相应强度并筛选角点。
- RK_MPI_IVE_STCorner : 图像角点检测第二步，按规则对候选角点进行排序。
- RK_MPI_IVE_MatchBgModel: 基于 CODEBOOK 的背景减除操作第一步，背景模型训练。
- RK_MPI_IVE_UpdateBgModel: 基于 CODEBOOK 的背景减除操作第二步，背景模型更新。
- RK_MPI_IVE_SAD: 计算两幅图像按 4X4\8X8\16X16 分块的 16 位\8 位 SAD 图像，并对 SAD 阈值化输出。
- RK_MPI_IVE_Warp_Affine_Init: 初始化仿射变换辅助内存。
- RK_MPI_IVE_Warp_Affine: 执行图像仿射变换任务。
- RK_MPI_IVE_Pyramid_GetSize: 获取生成图像金字塔所需的辅助内存大小。
- RK_MPI_IVE_Pyramid_Create: 执行创建图像金字塔任务。
- RK_MPI_IVE_Query: 查询已创建任务完成情况。

RK_MPI_IVE_Init

完成 IVE 上下文创建及必要的初始化工作

RK_S32 **RK_MPI_IVE_Init**();

『返回值』：

0	成功。
非 0	失败，参见错误码。

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 外部应用使用 IVE 一系列算子调用前，在程序初始化部分调用该接口，以完成 IVE 上下文创建及必要的初始化工作。
- 未调用该接口进行初始化，而直接使用 IVE 算子接口，系统将自动完成初始化工作。但在大量算子被调用的情况下，会进行频繁地进行初始化及内存分配工作，造成系统资源浪费。
- 在结束 IVE 相关功能使用后，必须调用 RK_MPI_IVE_Deinit 进行反初始化，避免系统资源未被释放。

RK_MPI_IVE_Deinit

销毁 IVE 上下文，释放相关系统资源

RK_S32 **RK_MPI_IVE_Deinit**();

『返回值』：

0	成功。
非 0	失败，参见错误码。

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 外部应用不再使用 IVE 之后，在程序退出时调用该接口，销毁 IVE 上下文。
- 未调用该接口进行初始化，系统将自动完成反初始化工作。但在大量算子被调用的情况下，会进行频繁地进行反初始化及内存释放，造成系统资源浪费。
- 在使用该接口前，确保已调用 RK_MPI_IVE_Init 完成 IVE 初始化工作。

RK_MPI_IVE_CvtImageToData

将单个 PLANAR 图像 IVE_IMAGE_S 类型转换为 IVE_DATA_S 类型

RK_S32 **RK_MPI_IVE_CvtImageToData** (IVE_IMAGE_S *image,

IVE_DATA_S *data);

『参数』：

image	输入图像指针。 不能为空。	输入
data	输出二维数据指针。 不能为空。	输出

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
image	IVE_IMAGE_S	1 byte	1x1~2047x2047
data	IVE_DATA_S	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 该模式下结构体类型转换，输入图像类型仅支持单个 planar 图像。

RK_MPI_IVE_CvtDataToImage

将二维数据类型 IVE_DATA_S 转换为单个 PLANAR 图像 IVE_IMAGE_S 类型。

RK_S32 **RK_MPI_IVE_CvtDataToImage** (IVE_DATA_S *data,
IVE_IMAGE_S *image);

『参数』：

data	输入二维数据指针。 不能为空。	输入
image	输出图像指针。 不能为空。	输出

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
data	IVE_DATA_S	1 byte	1x1~2047x2047
image	IVE_IMAGE_S	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 该模式下输出图像类型为单个 planar 图像。

RK_MPI_IVE_CvtImageToMemInfo

将单个 PLANAR 图像 IVE_IMAGE_S 类型转换为一维数据类型 IVE_MEM_INFO_S。

RK_S32 **RK_MPI_IVE_CvtImageToMemInfo** (IVE_IMAGE_S *image,
IVE_MEM_INFO_S *mem);

『参数』：

image	输出图像指针。 不能为空。	输入
mem	输出一维数据指针。 不能为空。	输出

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
image	IVE_IMAGE_S	1 byte	1x1~2047x2047
mem	IVE_MEM_INFO_S	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 该模式下结构体类型转换，输入图像类型仅支持单个 planar 图像。

RK_MPI_IVE_DMA

直接内存访问，支持快速拷贝、间隔拷贝、内存填充

```
RK_S32 RK_MPI_IVE_DMA(IVE_HANDLE *pHandle,
                      IVE_DATA_S *pstSrc,
                      IVE_DST_DATA_S *pstDst,
                      IVE_DMA_CTRL_S *pstDmaCtrl,
                      bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入(set 模式下同时也是输出)
pstDst	输出数据指针。 copy 模式下不能为空	输出
pstDmaCtrl	DMA 控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

注：Copy 模式是指 IVE_DMA_MODE_DIRECT_COPY 和 IVE_DMA_MODE_INTERVAL_COPY 模式；Set 模式是指 IVE_DMA_MODE_SET_3BYTE 和 IVE_DMA_MODE_SET_8BYTE 模式。

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	IVE_DATA_S	1 byte	1x1~2047x2047
pstDst	IVE_DATA_S	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- IVE_DMA_MODE_DIRECT_COPY：快速拷贝模式可实现从大块内存中扣取小块内存。
- IVE_DMA_MODE_INTERVAL_COPY：间隔拷贝模式
- 要求源数据宽度为 u8HorSegSize 的倍数；
- 间隔拷贝的方式：将每 u8VerSegRows 行中第一行数据分割为 u8HorSegSize 大小的段，拷贝每段中的前 u8ElemSize 大小的字节。
- IVE_DMA_MODE_SET_3BYTE：3 字节填充模式
- 仅使用 pstSrc，用 u64Val 的低 3 字节对源数据进行填充操作；当一行末尾不够 3 字节时，用 u64Val 的低字节填充。
- IVE_DMA_MODE_SET_8BYTE：8 字节填充模式
- 仅使用 pstSrc，用 u64Val 对源数据进行填充操作；当一行的末尾不足 8 字节时，用 u64Val 的低字节填充。

RK_MPI_IVE_Filter

5X5 模板滤波任务，通过配置不同的模板系数，可以实现不同的滤波。

```
RK_S32 RK_MPI_IVE_Filter(IVE_HANDLE *pHandle,
                         IVE_SRC_IMAGE_S *pstSrc,
                         IVE_DST_IMAGE_S *pstDst,
                         IVE_FILTER_CTRL_S *pstFltCtrl,
                         bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstFltCtrl	Filter 控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1、 YUV420SP、YUV422SP	1 byte	5x5~2047x2047
pstDst	U8C1、 YUV420SP、YUV422SP	1 byte	5x5~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- Filter 计算公式如下:

$$I_{out}(x,y) = \left\{ \sum_{-2\le j\le 2} \sum_{-2\le i\le 2} I(x+i, y+j) * coef(i,j) \right\} \gg norm$$

RK_MPI_IVE_CSC

色彩空间转换，可实现 YUV、RGB、HSV 的色彩空间转换。

RK_S32 **RK_MPI_IVE_CSC**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDst,

IVE_CSC_CTRL_S *pstCscCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstCscCtrl	CSC 控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1、 U8C3_PACKAGE、 U8C3_PLANAR、 YUV420SP、YUV422SP	1 byte	2x2~2046x2046
pstDst	U8C1、 U8C3_PACKAGE、 U8C3_PLANAR、 YUV420SP、YUV422SP	1 byte	2x2~2046x2046

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 当输出图像为多平面格式时，要求输出数据跨度一致。

RK_MPI_IVE_Sobel

3X3 或 5X5 模板 SOBEL-LIKE 梯度计算。

```
RK_S32 RK_MPI_IVE_Sobel(IVE_HANDLE *pHandle,
                        IVE_SRC_IMAGE_S *pstSrc,
                        IVE_DST_IMAGE_S * pstDstH,
                        IVE_SRC_IMAGE_S * pstDstV,
                        IVE_SOBEL_CTRL_S *pstSobelCtrl,
                        bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDstH	滤波后得到的水平方向图像指针。 不能为空。	输出
pstDstV	滤波后得到的垂直方向图像指针。 不能为空。	输出
pstCscCtrl	CSC 控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	5x5~2047x2047
pstDstH	S16C1	1 byte	5x5~2047x2047
pstDstV	S16C1	1 byte	5x5~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- Sobel 计算公式如下：

$$H_{out}(x,y) = \left\{ \sum_{-2\le j\le 2} \sum_{-2\le i\le 2} I(x+i, y+j) * coef(i,j) \right\} \gg norm$$

$$V_{out}(x,y) = \left\{ \sum_{-2\le j\le 2} \sum_{-2\le i\le 2} I(x+i, y+j) * coef(i,j) \right\} \gg norm$$

RK_MPI_IVE_MagAndAng

计算幅值幅角

RK_S32 **RK_MPI_IVE_MagAndAng**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDstMag,

IVE_DST_IMAGE_S *pstDstAng,

IVE_MAG_AND_ANG_CTRL_S *pstMagAndAngCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDstMag	输出幅值图像指针。 不能为空。	输出
pstDstAng	输出幅角图像指针。 不能为空。	输出
pstMagAndAngCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	5x5~2047x2047
pstDstMag	U16C1	1 byte	5x5~2047x2047
pstDstAng	U8C1	1 byte	5x5~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 幅值二维数据的输出类型为 IVE_IMAGE_TYPE_U16C1，幅角二维数据的输出类型为 IVE_IMAGE_TYPE_U8C1。
- 图像梯度幅值的计算公式如下:

$$H_{out} = \sum_{-2\le j\le 2}\sum_{-2\le i\le 2} I(x+i, y+j) * coef(i,j)$$

$$V_{out} = \sum_{-2\le j\le 2}\sum_{-2\le i\le 2} I(x+i, y+j) * coef(j,i)$$

$$Mag(x,y) = abs\big(H_{out}(x,y)\big) + abs\big(V_{out}(x,y)\big)$$

其中$I(x,y)$为输入的源图像，$coef(mask)$为计算梯度的 5x5 模板系数，H_{out}为水平方向图像梯度，V_{out}为垂直方向图像梯度。$Mag(x,y)$为图像梯度幅值。

- 幅角的计算公式如下:

$$H_{out} = \sum_{-2\le j\le 2}\sum_{-2\le i\le 2} I(x+i, y+j) * coef(i,j)$$

$$V_{out} = \sum_{-2\le j\le 2}\sum_{-2\le i\le 2} I(x+i, y+j) * coef(j,i)$$

$$\theta(x,y) = arctan\left(\frac{V_{out}}{H_{out}}\right)$$

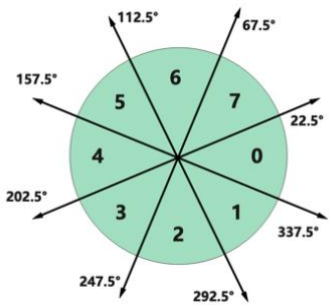


图 5-1

其中 I(x,y)为输入的源图像，coef(mask)为计算梯度的 5x5 模板系数，H_out 为水平方向图像梯度，V_out 为垂直方向图像梯度。θ(x,y)为图像梯度幅角，幅角根据计算结果数据对应图中 0~7 的方向值。

RK_MPI_IVE_Dilate

图像膨胀，图像形态学操作

RK_S32 **RK_MPI_IVE_Dilate**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDst,

IVE_DILATE_CTRL_S *pstDilateCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstDilateCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	5x5~2047x2047
pstDst	U8C1	1 byte	5x5~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像膨胀计算公式：

$$I_{out}(x,y) = \max(I(x-2,y-2)\&coef(-2,-2), I(x-1,y-2)\&coef(-1,-2), \ldots, I(x+2,y+2)\&coef(2,2))$$

RK_MPI_IVE_Erode

图像腐蚀，图像形态学操作。

RK_S32 **RK_MPI_IVE_Erode**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDst,

IVE_ERODE_CTRL_S *pstErodeCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstErodeCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	5x5~2047x2047
pstDst	U8C1	1 byte	5x5~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像腐蚀计算公式：

$$I_{out}(x,y) = (I(x-2,y-2)\&coef(-2,-2)) \,\&\, (I(x-1,y-2)\&coef(-1,-2)) \,\&\, \ldots \,\&\, (I(x+2,y+2)\&coef(2,2))$$

RK_MPI_IVE_Add

两张灰度图像加权相加操作

```
RK_S32 RK_MPI_IVE_Add(IVE_HANDLE *pHandle,
                      IVE_SRC_IMAGE_S *pstSrc1,
                      IVE_SRC_IMAGE_S *pstSrc2,
                      IVE_DST_IMAGE_S *pstDst,
                      IVE_ADD_CTRL_S *pstAddCtrl,
                      bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc1	源图像 1 指针。 不能为空。	输入
pstSrc2	源图像 2 指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstAddCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc1	U8C1	1 byte	1x1~2047x2047
pstSrc2	U8C1	1 byte	1x1~2047x2047
pstDst	U8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像加权相加计算公式：

$$I_{out}(i,j) = ((x * I_{src1}(i,j) + y * I_{src2}(i,j)) \gg 16)\&0xff$$

RK_MPI_IVE_And

两张灰度图像相与操作

RK_S32 **RK_MPI_IVE_And**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc1,

IVE_SRC_IMAGE_S *pstSrc2,

IVE_DST_IMAGE_S *pstDst,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc1	源图像 1 指针。 不能为空。	输入
pstSrc2	源图像 2 指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc1	U8C1 二值图	1 byte	1x1~2047x2047
pstSrc2	U8C1 二值图	1 byte	1x1~2047x2047
pstDst	U8C1 二值图	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像相与计算公式：

$$I_{out}(i,j) = I_{src1}(i,j) \,\&\, I_{src2}(i,j)$$

RK_MPI_IVE_Sub

两张灰度图像相减操作

```
RK_S32 RK_MPI_IVE_Sub(IVE_HANDLE *pHandle,
                      IVE_SRC_IMAGE_S *pstSrc1,
                      IVE_SRC_IMAGE_S *pstSrc2,
                      IVE_DST_IMAGE_S *pstDst,
                      IVE_SUB_CTRL_S *pstSubCtrl,
                      bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc1	源图像 1 指针。 不能为空。	输入
pstSrc2	源图像 2 指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc1	U8C1	1 byte	1x1~2047x2047
pstSrc2	U8C1	1 byte	1x1~2047x2047
pstDst	U8C1、S8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像相减计算公式：

$$I_{out}(i,j) = abs(I_{src1}(i,j) - I_{src2}(i,j))$$

RK_MPI_IVE_Or

两张灰度图像相或操作

RK_S32 **RK_MPI_IVE_Or**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc1,

IVE_SRC_IMAGE_S *pstSrc2,

IVE_DST_IMAGE_S *pstDst,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc1	源图像 1 指针。 不能为空。	输入
pstSrc2	源图像 2 指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc1	U8C1	1 byte	1x1~2047x2047
pstSrc2	U8C1	1 byte	1x1~2047x2047
pstDst	U8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像相或计算公式：

$$I_{out}(i,j) = I_{src1}(i,j) \mid I_{src2}(i,j)$$

RK_MPI_IVE_Xor

两张灰度图像相异或操作

```
RK_S32 RK_MPI_IVE_Xor(IVE_HANDLE *pHandle,
                      IVE_SRC_IMAGE_S *pstSrc1,
                      IVE_SRC_IMAGE_S *pstSrc2,
                      IVE_DST_IMAGE_S *pstDst,
                      bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc1	源图像 1 指针。 不能为空。	输入
pstSrc2	源图像 2 指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc1	U8C1	1 byte	1x1~2047x2047
pstSrc2	U8C1	1 byte	1x1~2047x2047
pstDst	U8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像相异或计算公式：

$$I_{out}(i,j) = I_{src1}(i,j) \wedge I_{src2}(i,j)$$

RK_MPI_IVE_Integ

图像积分图计算

RK_S32 **RK_MPI_IVE_Integ**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDst,

IVE_INTEG_CTRL_S *pstIntegCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstIntegCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	16 byte	1x1~2047x2047
pstDst	U32C1、U64C1	16 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

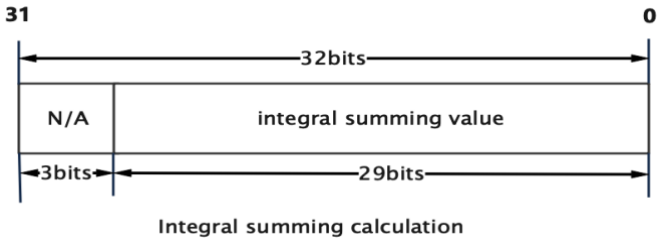
『注意』：

- 和积分图（U32C1）：模式 IVE_INTEG_OUT_CTRL_SUM，计算公式如下：

$$I_{sum}(x,y)=\sum_{i\geq 0}^{i\leq x}\sum_{j\geq 0}^{j\leq y} I(i,j)$$

$I_{sum}(x,y)$处积分值为从(0,0)开始到(x,y)点的灰度值和。

其在内存中的格式如下：



Integral summing calculation

图 5-2

- 平方和积分图（U64C1）：模式 IVE_INTEG_OUT_CTRL_SQSUM，计算公式如下：

$$I_{sq}(x,y)=\sum_{i\geq 0}^{i\leq x}\sum_{j\geq 0}^{j\leq y}(I(i,j)*I(i,j))$$

$I_{sq}(x,y)$处积分值为从(0,0)开始到(x,y)点的灰度值平方和。

其在内存中的格式如下：

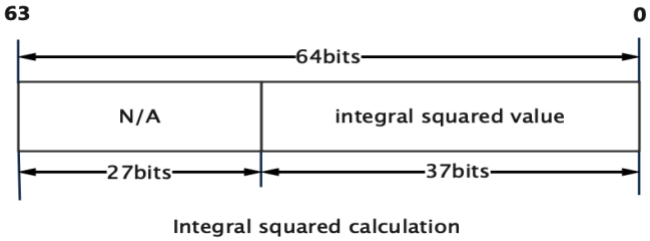


图 5-3

- 和积分与平方和积分组合（U64C1）：模式 IVE_INTEG_OUT_CTRL_COMBINE，计算公式如下：

$$I_{OUT}(X,Y) = (I_{SQ}(X,Y) \ll 28)) \mid (I_{SUM}(X,Y)\&0XFFFFFFFF)$$

和积分图与平方和积分图内存格式如下图：

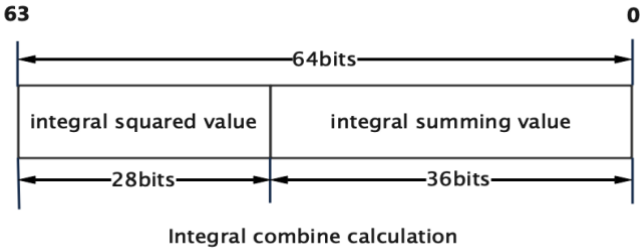


图 5-4

- 连通区域计算需要的辅助内存。

IVE_INTEG_OUT_CTRL_SUM 模式需要至少分配输入图像高度*3 大小的内存

IVE_INTEG_OUT_CTRL_SQSUM 模式需要至少分配输入图像高度*4 大小的内存

IVE_INTEG_OUT_CTRL_COMBINE 模式需要至少分配输入图像高度*6 大小的内存

RK_MPI_IVE_Hist

图像直方图计算

RK_S32 **RK_MPI_IVE_Hist**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_MEM_INFO_S *pstDst,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出直方图指针。 不能为空。	输出
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	1x1~2047x2047
pstDst	U8C1	16 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 直方图计算公式：

$$hist(x) = \sum_{i}\sum_{j}((I(i,j) == x)?1:0) \qquad x = 0...255$$

RK_MPI_IVE_Thresh

图像 8 位灰度图二值化操作

RK_S32 **RK_MPI_IVE_Thresh**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDst,

IVE_THRESH_U8_CTRL_S *pstThrCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstThrCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	1x1~2047x2047
pstDst	U8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像阈值二值化支持 BINARY、TRUNC、MINVAL、MIN_MIN_MAX、ORI_MID_MAX、MIN_MIN_ORI、MIN_ORI_MAX、ORI_MID_ORI 模式，计算公式如下:
- BINARY:

$$I_{out}(x,y)=\begin{cases} minVal & (I(x,y)\le lowThr) \\ maxVal & (I(x,y)>lowThr) \end{cases}$$

- TRUNC:

$$I_{out}(x,y)=\begin{cases} I(x,y) & (I(x,y)\le lowThr) \\ maxVal & (I(x,y)>lowThr) \end{cases}$$

- MINVAL:

$$I_{out}(x,y)=\begin{cases} minVal & (I(x,y)\le lowThr) \\ I(x,y) & (I(x,y)>lowThr) \end{cases}$$

- MIN_MID_MAX:

$$I_{out}(x,y)=\begin{cases} minVal & (I(x,y)\le lowThr) \\ midVal & (lowThr<I(x,y)\le highThr) \\ maxVal & (I(x,y)>highThr) \end{cases}$$

- ORI_MID_MAX:

$$I_{out}(x,y)=\begin{cases} I(x,y) & (I(x,y)\le lowThr) \\ midVal & (lowThr<I(x,y)\le highThr) \\ maxVal & (I(x,y)>highThr) \end{cases}$$

- MIN_MIN_ORI:

$$I_{out}(x,y)=\begin{cases} minVal & (I(x,y)\le lowThr) \\ midVal & (lowThr<I(x,y)\le highThr) \\ I(x,y) & (I(x,y)>highThr) \end{cases}$$

- MIN_ORI_MAX:

$$I_{out}(x,y)=\begin{cases} minVal & (I(x,y)\le lowThr) \\ I(x,y) & (lowThr<I(x,y)\le highThr) \\ maxVal & (I(x,y)>highThr) \end{cases}$$

- ORI_MID_ORI:

$$I_{out}(x,y)=\begin{cases} I(x,y) & (I(x,y)\le lowThr) \\ midVal & (lowThr<I(x,y)\le highThr) \\ I(x,y) & (I(x,y)>highThr) \end{cases}$$

RK_MPI_IVE_Thresh_u16

16 位数据到 8 位数据阈值二值化操作。

RK_S32 **RK_MPI_IVE_Thresh_U16**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDst,

IVE_THRESH_U16_CTRL_S *pstThrCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstThrCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U16C1	1 byte	1x1~2047x2047
pstDst	U8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像阈值二值化支持 IVE_THRESH_U16_MODE_U16_TO_U8_MIN_MID_MAX，IVE_THRESH_U16_MODE_U16_TO_U8_MIN_ORI_MAX 模式.

RK_MPI_IVE_Thresh_s16

带符号位的 16 位数据到 8 位数据的阈值二值化操作

```
RK_S32 RK_MPI_IVE_CSC(IVE_HANDLE *pHandle,
                      IVE_SRC_IMAGE_S *pstSrc,
                      IVE_DST_IMAGE_S *pstDst,
                      IVE_THRESH_U16_CTRL_S *pstThrCtrl,
                      bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstThrCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	S16C1	1 byte	1x1~2047x2047
pstDst	U8C1、S8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像阈值二值化支持：

IVE_THRESH_S16_MODE_S16_TO_S8_MIN_MID_MAX，

IVE_THRESH_S16_MODE_S16_TO_S8_MIN_ORI_MAX，

IVE_THRESH_S16_MODE_S16_TO_U8_MIN_MID_MAX，

IVE_THRESH_S16_MODE_S16_TO_U8_MIN_ORI_MAX。

RK_MPI_IVE_16bitto8bit

16 位数据到 8 位数据的线性转换

```
RK_S32 RK_MPI_IVE_CSC(IVE_HANDLE *pHandle,
                      IVE_SRC_IMAGE_S *pstSrc,
                      IVE_DST_IMAGE_S *pstDst,
                      IVE16BIT_TO_8BIT_CTRL_S *pst16BitTo8BitCtrl,
                      bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pst16BitTo8BitCtrl	CSC 控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U16C1、S16C1	1 byte	1x1~2047x2047
pstDst	U8C1、S8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 支持 4 种模式：

IVE_16BIT_TO_8BIT_MODE_S16_TO_S8

IVE_16BIT_TO_8BIT_MODE_S16_TO_U8_ABS

IVE_16BIT_TO_8BIT_MODE_S16_TO_U8_BIAS

IVE_16BIT_TO_8BIT_MODE_U16_TO_U8

RK_MPI_IVE_8bitto8bit

8 位数据到 8 位数据的线性转换

RK_S32 **RK_MPI_IVE_8BitTo8Bit**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDst,

IVE_8BIT_TO_8BIT_CTRL_S *pst8BitTo8BitCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pst8BitTo8BitCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1、S8C1	1 byte	1x1~2047x2047
pstDst	U8C1、S8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 支持 4 种模式：

IVE_8BIT_TO_8BIT_MODE_S16_TO_S8

IVE_8BIT_TO_8BIT_MODE_S16_TO_U8_ABS

IVE_8BIT_TO_8BIT_MODE_S16_TO_U8_BIAS

IVE_8BIT_TO_8BIT_MODE_U16_TO_U8

RK_MPI_IVE_OrdStatFilter

图像中值滤波，最小值滤波，最大值滤波

```
RK_S32 RK_MPI_IVE_OrdStatFilter(IVE_HANDLE *pHandle,
                                IVE_SRC_IMAGE_S *pstSrc,
                                IVE_DST_IMAGE_S *pstDst,
                                IVE_ORD_STAT_FILTER_CTRL_S *pstOrdStatFltCtrl,
                                bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstOrdStatFilterCtrl	CSC 控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	16 byte	5x5~2047x2047
pstDst	U8C1	16 byte	5x5~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

三种滤波模式计算公式：

- IVE_ORD_STAT_FILTER_MODE_MEDIAN：

$$I_{out}(x,y) = median_{-1\le j\le 1}^{-1\le i\le 1}\{I(x+i,y+j)\}$$

- IVE_ORD_STAT_FILTER_MODE_MAX：

$$I_{out}(x,y) = max_{-1\le j\le 1}^{-1\le i\le 1}\{I(x+i,y+j)\}$$

- IVE_ORD_STAT_FILTER_MODE_MIN：

$$I_{out}(x,y) = min_{-1\le j\le 1}^{-1\le i\le 1}\{I(x+i,y+j)\}$$

RK_MPI_IVE_Map

根据查找表进行图像像素值映射。

RK_S32 **RK_MPI_IVE_Map**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_SRC_MEM_INFO_S *pstMap

IVE_DST_IMAGE_S *pstDst,

IVE_MAP_CTRL_S *pstMapCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstMap	查找表缓存指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstMapCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	1x1~2047x2047
pstMap	U8C1	1 byte	1x1~2047x2047
pstDst	U8C1、U16C1、S16C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 计算公式如下：

$$I_{out}(x,y) = map[I(x,y)]$$

RK_MPI_IVE_EqualizeHist

图像直方图均衡化。

RK_S32 **RK_MPI_IVE_EqualizeHist**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDst,

IVE_EQHIST_CTRL_S *pstEqualizeHistCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstEqualizeHistCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	1x1~2047x2047
pstDst	U8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 控制参数 pstEqualizeHistCtrl 中 u32HistMem 为辅助内存，需要分配至少 256 * sizeof(RK_U32)大小。

RK_MPI_IVE_Ncc

两张相同分辨率灰度图像的归一化互相关系数。

```
RK_S32 RK_MPI_IVE_NCC(IVE_HANDLE *pHandle,
                      IVE_SRC_IMAGE_S *pstSrc1,
                      IVE_SRC_IMAGE_S *pstSrc2,
                      IVE_DST_MEM_INFO_S *pstDst,
                      bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc1	源图像 1 指针。 不能为空。	输入
pstSrc2	源图像 2 指针。 不能为空。	输入
pstDst	输出数据指针。 不能为空。	输出
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc1	U8C1	1 byte	1x1~2047x2047
pstSrc2	U8C1	1 byte	1x1~2047x2047
pstDst	U8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 计算公式如下：

$$NCC(I_{src1}, I_{src2}) = \frac{\sum_{i=1}^{w}\sum_{j=1}^{h}(I_{src1}(i,j) * I_{src2}(i,j))}{\sqrt{\sum_{i=1}^{w}\sum_{j=1}^{h}(I_{src1}^{2}(i,j))}\sqrt{\sum_{i=1}^{w}\sum_{j=1}^{h}(I_{src2}^{2}(i,j))}}$$

RK_MPI_IVE_CCL

二值图像的连通区域标记。

RK_S32 **RK_MPI_IVE_CCL**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S * pstSrcDst,

IVE_DST_MEM_INFO_S *pstBlob,

IVE_CCL_CTRL_S *pstCclCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrcDst	源图像指针，连通区域也标记在源图像上输出。 不能为空。	输入、输出
pstBlob	连通区域信息指针。 不能为空。	输出
pstCclCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrcDst	U8C1	1 byte	64x64~1984x2047
pstBlob	U8C1	16 byte	64x64~1984x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 连通区域的信息保存在 pstBlob→astRegion 中,其在内存中的格式如下图所示：

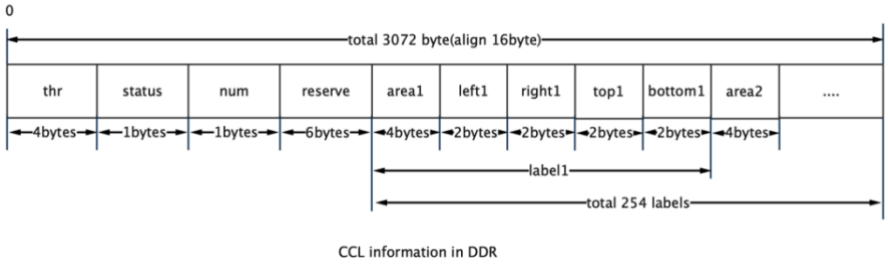


图 5-5

RK_MPI_IVE_Gmm

创建高斯混合背景模型，进行前景背景分离操作，参考 OPENCV 的 MOG。

RK_S32 **RK_MPI_IVE_GMM**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstFg,

IVE_DST_IMAGE_S *pstBg,

IVE_DST_IMAGE_S *pstMatchModelInfo,

IVE_MEM_INFO_S *pstModel,

IVE_GMM_CTRL_S *pstGmmCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源图像指针。 不能为空。	输入
pstFg	前景图像指针。 不能为空。	输出
pstBg	背景图像指针。 不能为空。	输出
pstMatchModelInfo	匹配信息指针。 不能为空。	输入、输出
pstModel	高斯混合模型指针。 不能为空。	输入、输出
pstGmmCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1、U8C3_PACKAGE	16 byte	1x1~2047x2047
pstFg	U8C1	16 byte	1x1~2047x2047
pstBg	U8C1、U8C3_PACKAGE	16 byte	1x1~2047x2047
pstMatchModelInfo	U8C1	16 byte	1x1~2047x2047
pstModel	IVE_MEM_INFO_S	16 byte	-

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- pstMatchModelInfo。
- pstModel。
- pstGmmCtrl。

RK_MPI_IVE_Gmm2

创建高斯混合背景模型，进行前景背景分离操作，参考 OPENCV 的 MOG2。

RK_S32 **RK_MPI_IVE_GMM2**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_SRC_IMAGE_S *pstFactor,

IVE_DST_IMAGE_S *pstFg,

IVE_DST_IMAGE_S *pstBg,

IVE_DST_IMAGE_S *pstMatchModelInfo,

IVE_MEM_INFO_S *pstModel,

IVE_GMM2_CTRL_S *pstGmm2Ctrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstFactor	模型更新参数指针	输入
pstFg	前景图像指针。 不能为空。	输出
pstBg	背景图像指针。 不能为空。	输出
pstMatchModelInfo	匹配信息指针。 不能为空。	输入、输出
pstModel	高斯混合模型指针。 不能为空。	输入、输出
pstGmm2Ctrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1、U8C3_PACKAGE	16 byte	1x1~2047x2047
pstFactor	U16C1	16 byte	1x1~2047x2047
pstFg	U8C1	16 byte	1x1~2047x2047
pstBg	U8C1、U8C3_PACKAGE	16 byte	1x1~2047x2047
pstMatchModelInfo	U8C1	16 byte	1x1~2047x2047
pstModel	IVE_MEM_INFO_S	16 byte	-

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- pstFactor 中存储模型匹配后权重增加比例，及模型匹配过程中比较方差变化的阈值，在内存中的格式如下：

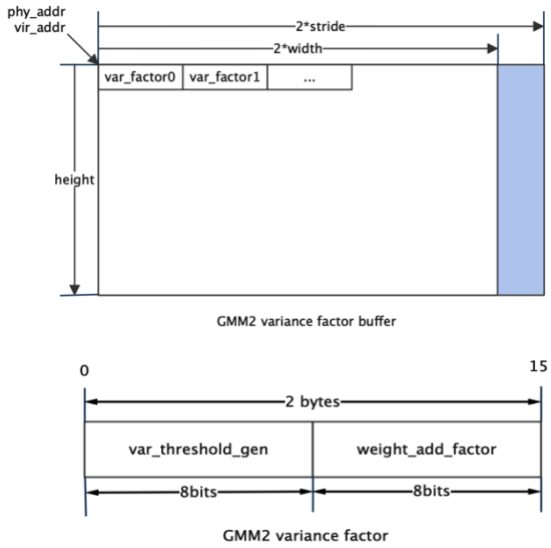


图 5-6

- pstMatchModelInfo 存储模型匹配标志，模型匹配索引，及模型个数，其在内存中的格式如下：

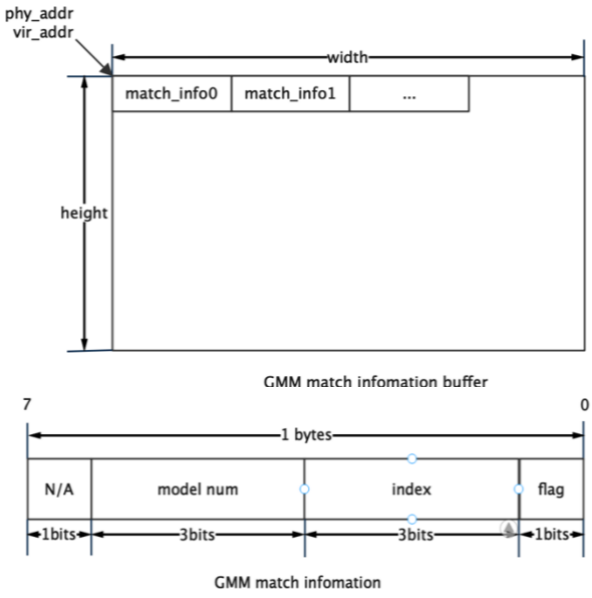


图 5-7

- pstModel 存储高斯混合模型，当输入图像类型为灰度图 U8C1 时，其在内存中的格式如下：

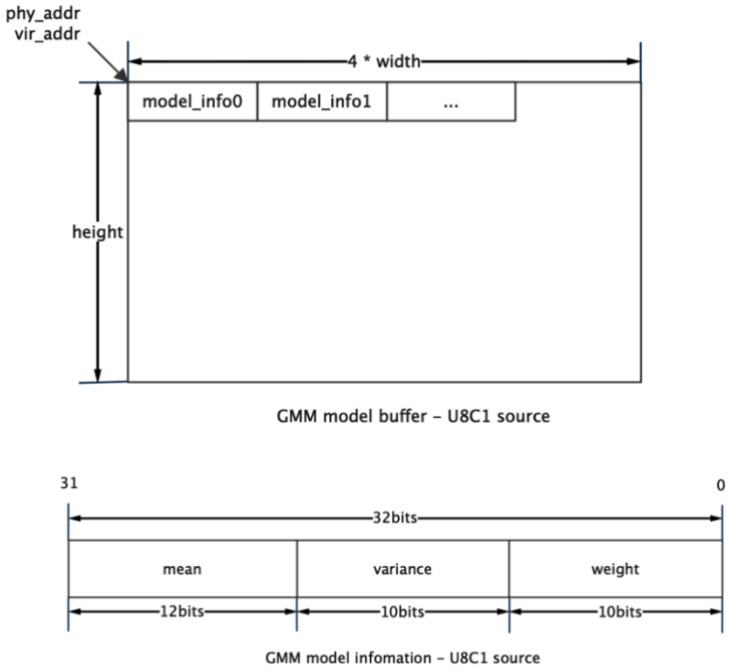


图 5-8

当输入图像类型为 RGB888 图像时，其在内存中的格式如下：

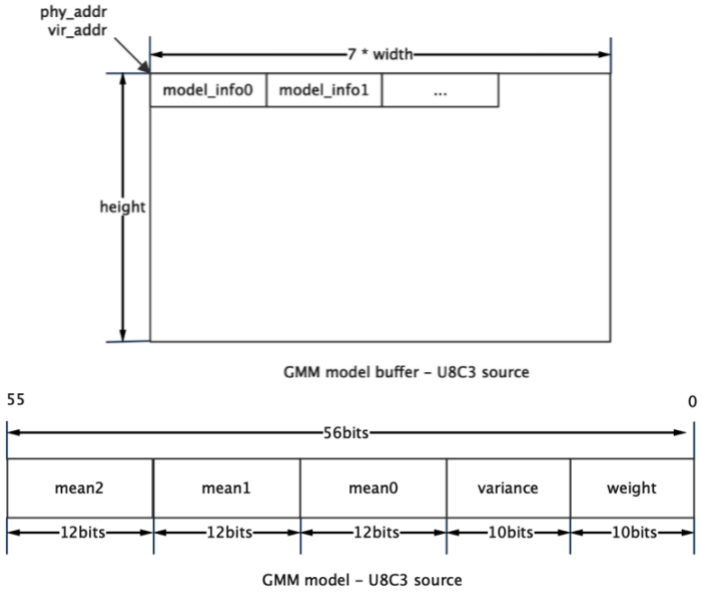


图 5-9

RK_MPI_IVE_CannyEdge

提取灰度图像边缘信息。

RK_S32 **RK_MPI_IVE_CannyEdge**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstEdge,

IVE_DST_MEM_INFO_S *pstStack,

IVE_CANNY_EDGE_CTRL_S *pstCannyEdgeCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源图像指针。 不能为空。	输入
pstEdge	边缘信息图像指针	
pstStack	边缘信息指针。	输出
pstCannyEdgeCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	5x5~2040x1152
pstEdge	U8C1	1 byte	5x5~2040x1152
pstStack	IVE_MEM_INFO_S	16 byte	-

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- pstCannyEdgeCtrl 中的 stMem 为辅助内存，至少分配源图像四分之一大小。

RK_MPI_IVE_LBP

计算图像 LBP 特征。

```
RK_S32 RK_MPI_IVE_LBP(IVE_HANDLE *pHandle,
                      IVE_SRC_IMAGE_S *pstSrc,
                      IVE_DST_IMAGE_S *pstDst,
                      IVE_LBP_CTRL_S *pstLbpCtrl,
                      bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源图像指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstLbpCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	16 byte	5x5~2047x2047
pstDst	U8C1	16 byte	5x5~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 图像局部二值化有 2 种模式：

$$lbp(x,y) = \sum_{i=0}^{7}\left((I_i - I_c) \geq thr\right) \ll (7-i)，其中thr \in [-128,127]$$

$$lbp(x,y) = \sum_{i=0}^{7}(abs(I_i - I_c) \geq thr) \ll (7-i)，其中thr \in [0,255]$$

RK_MPI_IVE_NormGrad

图像归一化梯度计算，梯度分量均归一化到 S8。

RK_S32 **RK_MPI_IVE_NormGrad**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDstH,

IVE_DST_IMAGE_S *pstDstV,

IVE_DST_IMAGE_S *pstDstHV,

IVE_NORM_GRAD_CTRL_S *pstNormGradCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源数据指针。 不能为空。	输入
pstDstH	输出水平方向梯度指针。	输出
pstDstV	输出垂直方向梯度指针。	输出
pstDstHV	水平、垂直方向梯度指针。	输出
pstNormGradCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	5x5~2047x2047
pstDstH	S8C1	1 byte	5x5~2047x2047
pstDstV	S8C1	1 byte	5x5~2047x2047
pstDstHV	S8C2_PACKAGE	1 byte	5x5~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 计算公式如下：

$$I_{out}(x,y) = \left\{ \sum_{-2\le j\le 2} \sum_{-2\le i\le 2} I(x+i, y+j) * coef(i,j) \right\} \gg norm$$

RK_MPI_IVE_LKOpticalFlowPyr

LK 光流计算（外部建立金字塔）。

RK_S32 **RK_MPI_IVE_LKOpticalFlowPyr**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S astSrcPrevPyr[],

IVE_SRC_IMAGE_S astSrcNextPyr[],

IVE_SRC_MEM_INFO_S *pstPrevPts,

IVE_MEM_INFO_S *pstNextPts,

IVE_DST_MEM_INFO_S *pstStatus,

IVE_DST_MEM_INFO_S *pstErr,

IVE_LK_OPTICAL_FLOW_PYR_CTRL_S *pstLkOptiFlowPyrCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
astSrcPrevPyr	前一帧图像金字塔数组。 不能为空。	输入
astSrcNextPyr	当前帧图像金字塔数组。	输入
pstPrevPts	前一帧光流追踪点指针。	输入
pstNextPts	当前帧光流追踪点指针。	输入
pstStatus	跟踪状态信息，1 表示成功，0 表示失败。	输入
pstErr	跟踪点相似度误差估计。 不能为空。	输出
pstLkOptiFlowPyrCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
astSrcPrevPyr[0]、astSrcNextPyr[0]	U8C1	16 byte	1x1~2047x2047
pstPrevPts[0]、pstNextPts[0]	-	16 byte	1x1~2047x2047
pstStatus	-	16 byte	-
pstErr	-	16 byte	-

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- pstLkOptiFlowPyrCtrl 中 u8MaxLevel 取值范围[0,3]，对应金字塔层数为[1,4]。
- 外部建立金字塔，要求每层图像的高、宽是上一层图像高、宽的一半。

RK_MPI_IVE_LKOpticalFlow

LK 光流计算（内部建立金字塔）。

RK_S32 **RK_MPI_IVE_LKOpticalFlow**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrcPre,

IVE_SRC_IMAGE_S *pstSrcCur,

IVE_SRC_MEM_INFO_S *pstPoint,

IVE_SRC_MEM_INFO_S *pstMv,

IVE_LK_OPTICAL_FLOW_CTRL_S *pstLkOptiFlowCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrcPre	前一帧图像指针。 不能为空。	输入
pstSrcCur	当前帧图像指针。 不能为空。	输入
pstPoint	当前金字塔层的初始特征点坐标。 不能为空。	
pstMv	追踪点运动向量指针。 不能为空。	输出
pstLkOptiFlowCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrcPre pstSrcCur	U8C1	16 byte	1x1~2047x2047
pstPoint	-	16 byte	-
pstMv	-	16 byte	-

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 内部默认建立 4 层金字塔，每层图像的高、宽是上一层图像高、宽的一半。
- pstMV 缓存格式为 IVE_MV_S9Q7_S，存储了跟踪状态信息及跟踪点运动向量。其在内存中的格式如下图所示：

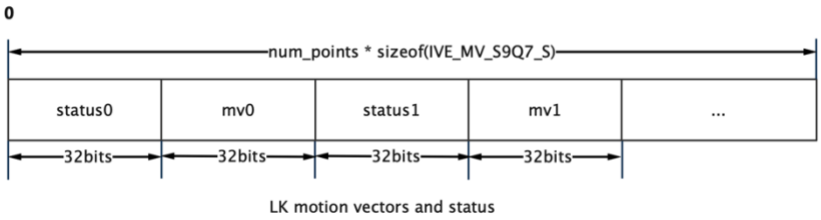


图 5-10

RK_MPI_IVE_STCandiCorner

图像角点检测第一步，计算角点相应强度并筛选角点。

RK_S32 **RK_MPI_IVE_STCandiCorner**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_MEM_INFO_S *pstCandiCorner,

IVE_ST_CANDI_CORNER_CTRL_S *pstStCandiCornerCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源图像指针。 不能为空。	输入
pstCandiCorner	候选角点指针。 不能为空。	输出
pstStCandiCornerCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	16 byte	64x64~1280x720
pstCandiCorner	-	16 byte	-

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 参考 OpenCV 中的 Shi-Tomas 角点检测。
- 候选角点缓存至少需要分配 u16Width * u16Height * sizeof(RK_U16) + sizeof(IVE_ST_CANDI_STACK_SIZE_S)。
- 候选角点信息包括角点响应强度、x 坐标及 y 坐标，在内存中的格式如下;

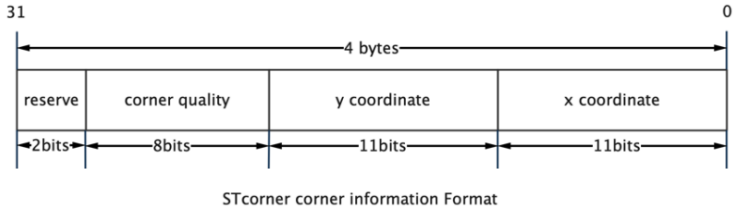


图 5-11

RK_MPI_IVE_STCorner

图像角点检测第二步，按规则对候选角点进行排序。

RK_S32 **RK_MPI_IVE_STCorner**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_MEM_INFO_S *pstCandiCorner,

IVE_DST_MEM_INFO_S *pstCorner,

IVE_ST_CORNER_CTRL_S *pstStCornerCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc	源图像指针。 不能为空。	输入
pstCandiCorner	候选角点指针	输入
pstCorner	输出筛选后的角点指针。 不能为空。	输出
pstStCornerCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

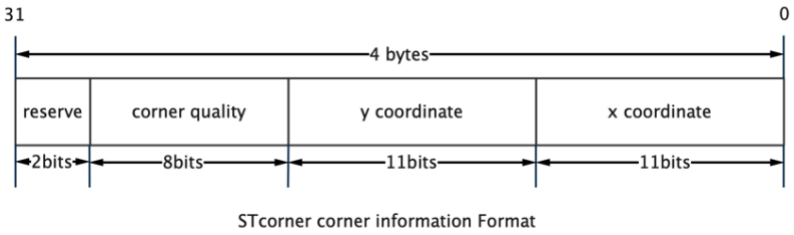
参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	16 byte	64x64~1280x720
pstCandiCorner	-	16 byte	-
pstCorner	-	16 byte	-

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- pstStCornerCtrl 中的 stMem 为辅助内存，至少需要分配 stSrcImg.u32Height * stSrcImg.au32Stride[0] + sizeof(IVE_ST_CORNER_MEM_S) * 2;。
- 最终输出的角点信息与候选角点在内存中的格式一致。同样包括角点响应强度、x 坐标及 y 坐标，在内存中的格式如下：



STcorner corner information Format

图 5-12

RK_MPI_IVE_MatchBgModel

基于 CODEBOOK 的背景减除操作第一步，背景模型训练。

RK_S32 **RK_MPI_IVE_MatchBgModel**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstCurImg,

IVE_DATA_S *pstBgModel,

IVE_MATCH_BG_MODEL_CTRL_S *pstMatchBgModelCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstCurImg	源图像指针。 不能为空。	输入
pstBgModel	背景模型指针。 不能为空。	输出
pstMatchBgModelCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

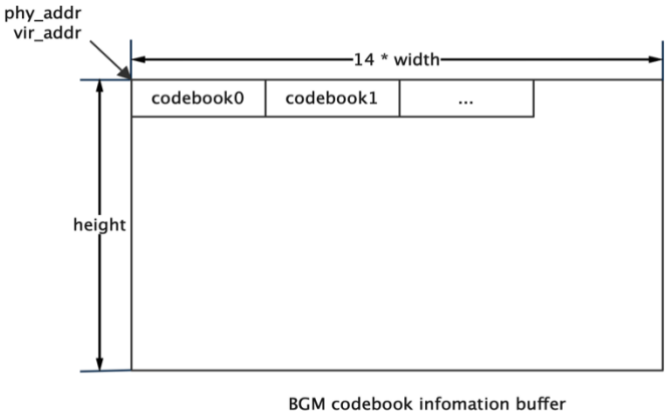
参数	数据类型	地址对齐	分辨率
pstCurImg	U8C1	1 byte	1x1~2047x2047
pstBgModel	-	1 byte	-

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- pstBgModel 存储 coodbook 模型，其格式在内存中如下图所示：



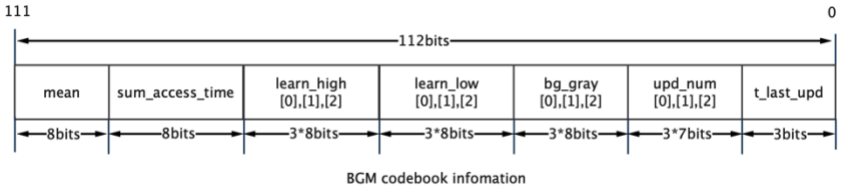


图 5-13

RK_MPI_IVE_UpdateBgModel

基于 CODEBOOK 的背景减除操作第二步，背景模型更新。

RK_S32 **RK_MPI_IVE_UpdateBgModel**(IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S * pstCurImg,

IVE_DATA_S *pstBgModel,

_IVE_IMAGE_S *pstFgFlag,

IVE_DST_IMAGE_S *pstBgImg,

IVE_UPDATE_BG_MODEL_CTRL_S *pstUpdateBgModelCtrl,

bool bInstant);

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstCurImg	源图像指针。 不能为空。	输入
pstBgModel	背景模型指针。 不能为空。	输入、输出
pstFgFlag	输出前景图像指针	输出
pstBgImg	输出背景图像指针。 不能为空。	输出
pstUpdateBgModelCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstCurImg	U8C1	1 byte	1x1~2047x2047
pstBgModel	-	1 byte	-
pstFgFlag	U8C1	1 byte	1x1~2047x2047
pstBgImg	U8C1	1 byte	1x1~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- pstBgModel 存储 coodbook 模型，每个模型内存大小为 14 bytes,其格式在内存中如下所示：

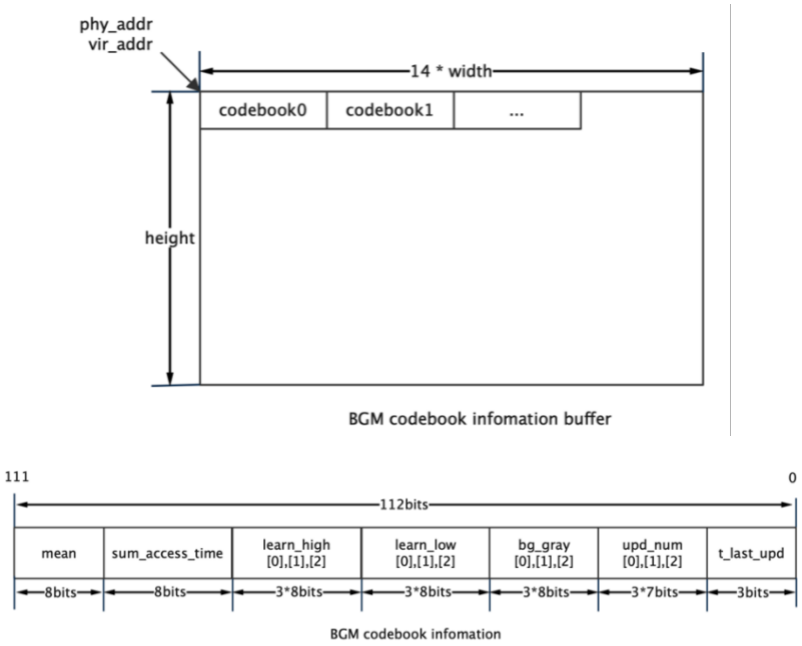


图 5-14

RK_MPI_IVE_SAD

计算两幅图像按 4X4\8X8\16X16 分块的 16 位\8 位 SAD 图像，以及对 SAD 进行阈值化输出。

```
RK_S32 RK_MPI_IVE_SAD(IVE_HANDLE *pHandle,
                      IVE_SRC_IMAGE_S * pstSrc1,
                      IVE_SRC_IMAGE_S *pstSrc2,
                      IVE_DST_IMAGE_S *pstSad,
                      IVE_DST_IMAGE_S *pstThr,
                      IVE_SAD_CTRL_S *pstSadCtrl,
                      bool bInstant);
```

『参数』：

pHandle	handle 指针。 不能为空。	输出
pstSrc1	源图像 1 指针。 不能为空。	输入
pstSrc2	源图像 2 指针。 不能为空。	输入
pstSad	输出 SAD 图像指针。 不能为空。	输出
pstThr	输出 SAD 阈值化图像指针。 不能为空。	输出
pstSadCtrl	控制参数指针。 不能为空。	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc1	U8C1	1 byte	64x64~2047x2047
pstSrc2	U8C1	1 byte	64x64~2047x2047
pstSad	U8C1、U16C1	1 byte	64x64~2047x2047
pstThr	U8C1	1 byte	64x64~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 计算公式如下：

$$\mathrm{Diff}(i,j) = |I_1(i,j) - I_2(i,j)|$$

$$\mathrm{SAD}_{out}(x,y) = \sum_{\substack{n*x \le i < n*(x+1) \\ n*y \le j < n*(y+1)}} \mathrm{Diff}(i,j), \quad (x \ge 0, y \ge 0, \mathrm{stride} = n)$$

$$\mathrm{THR}_{out}(x,y) = \begin{cases} \mathrm{minVal} & (\mathrm{SAD}_{out}(x,y) \le \mathrm{Thr}) \\ \mathrm{maxVal} & (\mathrm{SAD}_{out}(x,y) > \mathrm{Thr}) \end{cases}$$

其中，

n= 4 对应 RVE_SAD_MODE_MB_4X4

n= 8 对应 RVE_SAD_MODE_MB_8X8

n=16 对应 RVE_SAD_MODE_MB_16X16

RK_MPI_IVE_Warp_Affine_Init

初始化仿射变换辅助内存。

RK_S32 **RK_MPI_IVE_Warp_Affine_Init** (IVE_MEM_INFO_S *pstMem,

RK_U32 u32Width,

RK_U32 u32Height);

『参数』：

pstMem	辅助内存指针 不能为空。	输出
u32Width	输入图像宽度。 不能为空。	输入
u32Height	输入图像高度。 不能为空。	输出

『返回值』：

0	成功。
非 0	失败，参见错误码。

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 预先申请输入图像 5 倍大小的辅助内存后，调用此接口完成必要的初始化工作。而后调用 RK_MPI_IVE_Warp_Affine 完成图像仿射变换。

RK_MPI_IVE_Warp_Affine

执行图像仿射变换任务。

RK_S32 **RK_MPI_IVE_Warp_Affine** (IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S *pstDst,

IVE_WARP_AFFINE_CTRL_S *pstWarpAffineCtrl,

bool bInstant);

『参数』：

pHandle	任务 ID。 不能为空。	输出
pstSrc	输入图像指针。 不能为空。	输入
pstDst	输出图像指针。 不能为空。	输出
pstWarpAffineCtrl	控制参数	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	16x16~256x256
pstDst	U8C1	1 byte	16x16~256x256

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 调用此接口前，需要先执行 RK_MPI_IVE_Warp_Affine_Init，以完成必要的初始化工作。
- pstWarpAffineCtrl. stAffineMat 为 3x2 的仿射变换矩阵，如下图所示：

$$\begin{bmatrix} a_{00} & a_{01} & b_{00} \\ a_{10} & a_{11} & b_{01} \end{bmatrix}$$

其中，

$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}$ 为线性变换相关系数，$\begin{bmatrix} b_{00} \\ b_{01} \end{bmatrix}$ 为平移相关的系数。

RK_MPI_IVE_Pyramid_GetSize

获取生成图像金字塔所需的辅助内存大小。

RK_S32 **RK_MPI_IVE_Pyramid_GetSize** (RK_U32 u32Width, RK_U32 u32Height,

IVE_PYRAMID_CTRL_S *pstPyramidCtrl);

『参数』：

u32Width	输入图像宽度。 不能为空。	输出
u32Height	输入图像高度。 不能为空。	输入
pstPyramidCtrl	控制指针。 不能为空。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 在申请创建图像金字塔任务的辅助内存之前，调用此接口获取需要申请的辅助内存大小。

RK_MPI_IVE_Pyramid_Create

查询已创建任务完成情况。

RK_S32 **RK_MPI_IVE_Pyramid_Create** (IVE_HANDLE *pHandle,

IVE_SRC_IMAGE_S *pstSrc,

IVE_DST_IMAGE_S pstPyramid[],

IVE_PYRAMID_CTRL_S *pstPyramidCtrl,

bool bInstant);

『参数』：

pHandle	任务 ID。 不能为空。	输出
pstSrc	输入图像指针。 不能为空。	输入
pstPyramid[],	金字塔图像数组。 不能为空。	输入
pstPyramidCtrl	控制参数	输入
bInstant	及时返回结果标志。	输入

『返回值』：

0	成功。
非 0	失败，参见错误码。

『要求』：

参数	数据类型	地址对齐	分辨率
pstSrc	U8C1	1 byte	16x16~2047x2047
pstPyramid[]	U8C1	1 byte	16x16~2047x2047

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 生成的金字塔图像默认缩放系数为 0.5，上下层金字塔图像宽高的关系为：

$$width_{n+1} = \frac{(width_n + 1)}{2}$$

$$height_{n+1} = \frac{(height_n + 1)}{2}$$

- 金字塔图像默认跨度均为 16 字节对齐。

RK_MPI_IVE_Query

查询已创建任务完成情况。

RK_S32 **RK_MPI_IVE_Query**(IVE_HANDLE *pHandle,

bool *pbFinish,

bool bBlock);

『参数』：

pHandle	任务 ID。 不能为空。	输出
pbFinish	任务完成状态指针。 不能为空。	输入
bBlock	是否阻塞查询标志。 不能为空。	输出

『返回值』：

0	成功。
非 0	失败，参见错误码。

『引用』：

头文件	rk_comm_ive.h、rk_ive.h、rk_mpi_ive.h
库文件	librve.a、librve.so

『注意』：

- 在用户使用 IVE 任务结果前，为确保 RKIVE 任务已完成，用户可以使用阻塞方式调用此接口查询。

6. 数据类型及结构体

IVE_IMAGE_S

IVE_SRC_IMAGE_S

IVE_DST_IMAGE_S

定义二维图像信息

「定义」

```
typedef struct rkIVE_IMAGE_S {
    RK_U64 au64PhyAddr[3];
    RK_U64 au64VirAddr[3];
    RK_U32 au32Stride[3];
    RK_U32 u32Width;
    RK_U32 u32Height;
    IVE_IMAGE_TYPE_E enType;
    RK_U32 u32Reserved;
} IVE_IMAGE_S;

typedef IVE_IMAGE_S IVE_SRC_IMAGE_S;
typedef IVE_IMAGE_S IVE_DST_IMAGE_S;
```

「说明」

au64PhyAddr	图像缓存物理地址数组，分别存储多个 planar 缓存首地址
au64VirAddr	图像缓存虚拟地址数组，分别存储多个 planar 缓存首地址
au32Stride	图像缓存行跨度数组，分别存储多个 planar 行跨度
u32Width	图像宽度
u32Height	图像高度
enType	图像类型
u32Reserved	保留位

IVE_DATA_S

IVE_SRC_DATA_S

IVE_DST_DATA_S

定义二维数据信息

「定义」

```
typedef struct rkIVE_DATA_S {
    RK_U64 u64PhyAddr;
    RK_U64 u64VirAddr;

    RK_U32 u32Stride;
    RK_U32 u32Width;
    RK_U32 u32Height;

    RK_U32 u32Reserved;
} IVE_DATA_S;
```

```
typedef IVE_DATA_S IVE_SRC_DATA_S;
typedef IVE_DATA_S IVE_DST_DATA_S;
```

「说明」

u64PhyAddr	二维数据缓存物理地址
au64VirAddr	二维数据缓存虚拟地址
au32Stride	二维数据缓存行跨度
u32Width	二维数据宽度
u32Height	二维数据高度
u32Reserved	保留位

IVE_MEM_INFO_S

IVE_SRC_MEM_INFO_S

IVE_DST_MEM_INFO_S

定义一维数据缓存信息

「定义」

```
typedef struct rkIVE_MEM_INFO_S {
   RK_U64 u64PhyAddr;
   RK_U64 u64VirAddr;
   RK_U32 u32Size;
   RK_U32 u32Reserved;
} IVE_MEM_INFO_S;

typedef IVE_MEM_INFO_S IVE_SRC_MEM_INFO_S;
typedef IVE_MEM_INFO_S IVE_DST_MEM_INFO_S;
```

「说明」

u64PhyAddr	一维数据缓存物理地址
au64VirAddr	一维数据缓存虚拟地址
u32Size	一维数据缓存所占用的内存空间大小
u32Reserved	保留位

IVE_DMA_CTRL_S

定义 DMA 控制信息

「定义」

```
typedef struct rkIVE_DMA_CTRL_S {
   IVE_DMA_MODE_E enMode;
   RK_U64 u64Val;
   RK_U8 u8HorSegSize;
   RK_U8 u8ElemSize;
   RK_U8 u8VerSegRows;
} IVE_DMA_CTRL_S;
```

「说明」

enMode	IVE_DMA_MODE_DIRECT_COPY: 直接拷贝模式 IVE_DMA_MODE_INTERVAL_COPY: 间隔拷贝模式 IVE_DMA_MODE_SET_3BYTE: 3 字节填充模式 IVE_DMA_MODE_SET_8BYTE: 8 字节填充模式
u64Val	仅填充模式使用，3 字节填充模式用低 3byte 保存。
u8HorSegSize	仅间隔拷贝模式使用，水平方向将源图像一行分割的段大小。取值范围： {2, 3, 4, 8, 16}。
u8ElemSize	仅间隔拷贝模式使用，分割的每一段中前 u8ElemSizebyte 为有效的拷贝字段。 取值范围：[1, u8HorSegSize-1]。
u8VerSegRows	仅间隔拷贝模式使用，将每 u8VerSegRows 行中第一行数据分割为 u8HorSegSize 大小的段，拷贝每段中的前 u8ElemSize 大小的字节 取值范围：[1, min{65535/srcStride, srcHeight}]。

IVE_FILTER_CTRL_S

定义滤波控制信息

「定义」

```
typedef struct rkIVE_FILTER_CTRL_S {
   RK_U8 u8CoefSel;
   RK_U8 u8Norm;
   RK_U8 u8OutMode;
   RK_S8 as8Mask[25];
} IVE_FILTER_CTRL_S;
```

「说明」

u8CoefSel	模板系数： 0：3x3 1：5x5
u8Norm	归一化参数。 取值范围：[0, 13]。
u8OutMode	输出数据格式： 0：RK_U8 1：RK_S8 2：RK_U16 3：RK_S16
as8Mask	5x5 模板系数，外围系数设为 0 可实现 3x3 模板滤波。

IVE_CSC_CTRL_S

定义 CSC 控制信息

「定义」

```
typedef struct rkIVE_CSC_CTRL_S {
    IVE_CSC_MODE_E enMode;
    RK_U8 u8InDataFmt;
    RK_U8 u8OutDataFmt;
    RK_U8 u8YUV2RGBRange;
    RK_U8 u8RGB2YUVRange;
} IVE_CSC_CTRL_S;
```

「说明」

enMode	IVE_CSC_MODE_LIMIT_BT601_YUV2RGB IVE_CSC_MODE_LIMIT_BT709_YUV2RGB IVE_CSC_MODE_FULL_BT601_YUV2RGB IVE_CSC_MODE_FULL_BT709_YUV2RGB IVE_CSC_MODE_LIMIT_BT601_YUV2HSV IVE_CSC_MODE_LIMIT_BT709_YUV2HSV IVE_CSC_MODE_FULL_BT601_YUV2HSV IVE_CSC_MODE_FULL_BT709_YUV2HSV IVE_CSC_MODE_LIMIT_BT601_RGB2YUV IVE_CSC_MODE_LIMIT_BT709_RGB2YUV IVE_CSC_MODE_FULL_BT601_RGB2YUV IVE_CSC_MODE_FULL_BT709_RGB2YUV IVE_CSC_MODE_LIMIT_BT601_RGB2HSV IVE_CSC_MODE_LIMIT_BT709_RGB2HSV IVE_CSC_MODE_FULL_BT601_RGB2HSV IVE_CSC_MODE_FULL_BT709_RGB2HSV
u8InDataFmt	输入数据格式。
u8OutDataFmt	输出数据格式。
u8YUV2RGBRange	YUV 转 RGB 模式数据范围： 0: [16~235] 1: [0~255]
u8RGB2YUVRange	RGB 转 YUV 模式数据范围： 0: [16~235] 1: [0~255]

IVE_SOBEL_CTRL_S

定义 SOBEL 控制信息

「定义」

```
typedef struct rkIVE_SOBEL_CTRL_S {
   RK_U8 u8CoefSel;
   RK_U8 u8OutCtrl;
   RK_U8 u8Norm;
   RK_U8 u8OutMode;
   RK_S8 as8Mask[25];
} IVE_SOBEL_CTRL_S;
```

「说明」

u8CoefSel	模板系数： 0：3x3 1：5x5
u8OutCtrl	输出模式： 0：水平方向、垂直方向 1：水平方向 2：垂直方向
u8Norm	归一化参数。 取值范围：[0, 13]。
u8OutMode	输出数据格式： 0：RK_U8 1：RK_S8 2：RK_U16 3：RK_S16
as8Mask	5x5 模板系数，外围系数设为 0 可实现 3x3 模板滤波。

IVE_MAG_AND_ANG_CTRL_S

定义幅值幅角计算的控制信息

「定义」

```
typedef struct rkIVE_MAG_AND_ANG_CTRL_S {
  RK_S8 as8Mask[25];
  IVE_MEM_INFO_S stMem;
} IVE_MAG_AND_ANG_CTRL_S;
```

「说明」

as8Mask	5x5 模板系数
stMem	连通区域计算需要的辅助内存，需要至少分配 4 倍输入图像缓冲大小的内存

IVE_DILATE_CTRL_S

定义图像膨胀控制信息

「定义」

```
typedef struct rkIVE_DILATE_CTRL_S {
  RK_U8 au8Mask[25];
} IVE_DILATE_CTRL_S;
```

「说明」

au8Mask	5x5 模板系数

IVE_ERODE_CTRL_S

定义图像腐蚀控制信息

「定义」

```
typedef struct rkIVE_ERODE_CTRL_S {
   RK_U8 au8Mask[25];
} IVE_ERODE_CTRL_S;
```

「说明」

au8Mask	5x5 模板系数

IVE_ADD_CTRL_S

定义图像加权加控制信息

「定义」

```
typedef struct rkIVE_ADD_CTRL_S {
   RK_U16 u0q16X;
   RK_U16 u0q16Y;
} IVE_ADD_CTRL_S;
```

「说明」

u0q16X	加权加“xA+yB”中的权重“x”。 取值范围：[1, 65535]。
u0q16Y	加权加“xA+yB”中的权重“y”。 取值范围：{65536 - u0q16X}。

IVE_SUB_CTRL_S

定义图像减控制信息

「定义」

```
typedef struct rkIVE_SUB_CTRL_S {
  IVE_SUB_MODE_E enMode;
} IVE_SUB_CTRL_S;
```

「说明」

enMode	IVE_SUB_MODE_ABS: 取差的绝对值。 IVE_SUB_MODE_SHIFT: 将结果右移一位输出，保留符号位。

IVE_INTEG_CTRL_S

定义积分图控制信息

「定义」

```
typedef struct rkIVE_INTEG_CTRL_S {
  IVE_INTEG_OUT_CTRL_E enOutCtrl;
  IVE_MEM_INFO_S stMem;
} IVE_INTEG_CTRL_S
```

「说明」

enOutCtrl	IVE_INTEG_OUT_CTRL_SUM: 仅和积分图输出。 IVE_INTEG_OUT_CTRL_SQSUM: 仅平方和积分图输出。 IVE_INTEG_OUT_CTRL_COMBINE: 和、平方和积分图组合输出
stMem	连通区域计算需要的辅助内存。 IVE_INTEG_OUT_CTRL_SUM 模式需要至少分配输入图像高度*3 大小的内存 IVE_INTEG_OUT_CTRL_SQSUM 模式需要至少分配输入图像高度*4 大小的内存 IVE_INTEG_OUT_CTRL_COMBINE 模式需要至少分配输入图像高度*6 大小的内存

IVE_THRESH_CTRL_S

定义图像阈值二值化控制信息

「定义」

```
typedef struct rkIVE_THRESH_U8_CTRL_S {
   IVE_THRESH_MODE_E enMode;
   RK_U8 u8LowThr;/
   RK_U8 u8HighThr;
   RK_U8 u8MinVal;
   RK_U8 u8MidVal;
   RK_U8 u8MaxVal;
} IVE_THRESH_U8_CTRL_S;

typedef struct rkIVE_THRESH_U16_CTRL_S {
   IVE_THRESH_U16_MODE_E enMode;
   RK_U16 u16LowThr;
   RK_U16 u16HighThr;
   RK_U8 u8MinVal;
```

```
    RK_U8 u8MidVal;
    RK_U8 u8MaxVal;
} IVE_THRESH_U16_CTRL_S;
typedef struct rkIVE_THRESH_S16_CTRL_S {
    IVE_THRESH_S16_MODE_E enMode;
    RK_S16 S16LowThr;
    RK_S16 S16HighThr;
    RK_S8 S8MinVal;
    RK_S8 S8MidVal;
    RK_S8 S8MaxVal;
} IVE_THRESH_S16_CTRL_S;
```

「说明」

enMode	阈值化运算模式: IVE_THRESH_MODE_BINARY IVE_THRESH_MODE_TRUNC IVE_THRESH_MODE_TO_MINVAL IVE_THRESH_MODE_MIN_MID_MAX IVE_THRESH_MODE_ORI_MID_MAX IVE_THRESH_MODE_MIN_MID_ORI IVE_THRESH_MODE_MIN_ORI_MAX IVE_THRESH_MODE_ORI_MID_ORI
u8LowThr	低阈值。 取值范围：[0,255]。
u8HighThr	高阈值。 0≤u8LowThresh≤u8HighThresh≤255。
u8MinVal	最小值。 取值范围：[0,255]。
u8MidVal	中间值。 取值范围：[0,255]。
u8MaxVal	最大值。 取值范围：[0,255]。

IVE_8BIT_TO_8BIT_CTRL_S

IVE_16BIT_TO_8BIT_CTRL_S

定义 8 位、16 位数据到 8 位数据线性转换控制信息

「定义」

```
typedef struct rkIVE_8BIT_TO_8BIT_CTRL_S {
IVE_8BIT_TO_8BIT_MODE_E enMode;
RK_U8 u8Denominator;
RK_U8 u8Numerator;
RK_S8 s8Bias;
} IVE_8BIT_TO_8BIT_CTRL_S;

typedef struct rkIVE_16BIT_TO_8BIT_CTRL_S {
IVE_16BIT_TO_8BIT_MODE_E enMode;
RK_U16 u16Denominator;
RK_U8 u8Numerator;
RK_S8 s8Bias;
} IVE_16BIT_TO_8BIT_CTRL_S;
```

「说明」

enMode	转换模式 U8->U8: IVE_8BIT_TO_8BIT_MODE_S8_TO_S8 IVE_8BIT_TO_8BIT_MODE_S8_TO_U8_ABS IVE_8BIT_TO_8BIT_MODE_S8_TO_U8_BIAS IVE_8BIT_TO_8BIT_MODE_U8_TO_U8 U16->U8: IVE_16BIT_TO_8BIT_MODE_S16_TO_S8 IVE_16BIT_TO_8BIT_MODE_S16_TO_U8_ABS IVE_16BIT_TO_8BIT_MODE_S16_TO_U8_BIAS IVE_16BIT_TO_8BIT_MODE_U16_TO_U8
u8Denominator u16Denominator	线性变换中的分母。
u8Numerator	线性变换中的分子。 取值范围：[0,255]。
s8Bias	线性变换中的平移项。 取值范围：[-128,127]。

IVE_ORD_STAT_FILTER_CTRL_S

定义顺序统计量滤波模式

「定义」

```
typedef struct rkIVE_ORD_STAT_FILTER_CTRL_S {
   IVE_ORD_STAT_FILTER_MODE_E enMode;
} IVE_ORD_STAT_FILTER_CTRL_S;
```

「说明」

enMode	IVE_ORD_STAT_FILTER_MODE_MEDIAN: 中值滤波 IVE_ORD_STAT_FILTER_MODE_MAX: 最大值滤波 IVE_ORD_STAT_FILTER_MODE_MIN: 最小值滤波

IVE_MAP_CTRL_S

定义图像滤波模式

「定义」

```
typedef struct rkIVE_MAP_CTRL_S {
   IVE_MAP_MODE_E enMode;
} IVE_MAP_CTRL_S;
```

「说明」

enMode	IVE_MAP_MODE_U8: U8C1->U8C1 Map 模式 IVE_MAP_MODE_S16: U8C1->U16C1 Map 模式 IVE_MAP_MODE_U16: U8C1->S16C1 Map 模式

IVE_EQHIST_CTRL_S

定义图像直方图均衡化控制参数

「定义」

```
typedef struct rkIVE_EQUALIZE_RKST_CTRL_S {
   IVE_EQUALIZE_MODE_E enMode;
   RK_U32 u32HistArray[256];
   IVE_MEM_INFO_S u32HistMem;
} IVE_EQHIST_CTRL_S;
```

「说明」

enMode	IVE_EQUALIZE_MODE_EQHIST_WITH_EXT_HIST: 外部输入直方图统计信息 IVE_EQUALIZE_MODE_EQHIST: 内部自动计算直方图统计信息
u32HistArray	外部输入的直方图统计信息
u32HistMem	直方图均衡化辅助内存，需要分配至少 256 * sizeof(RK_U32)大小的内存。

IVE_CCL_CTRL_S

定义图像连通区域控制参数

「定义」

```
typedef struct rkIVE_CCL_CTRL_S {
    IVE_CCL_MODE_E enMode;
    RK_U16 u16InitAreaThr;
    RK_U16 u16Step;
    IVE_MEM_INFO_S stMem;
} IVE_CCL_CTRL_S;
```

「说明」

enMode	IVE_CCL_MODE_4C: 四连通模式 IVE_CCL_MODE_8C: 八连通模式
u16InitAreaThr	初始面积阈值。 取值范围：[0, 65535]。 参考取值：4。
u16Step	面积阈值增长步长。 取值范围：[1,65535]。 参考取值：2。
stMem	连通区域计算需要的辅助内存，需要至少分配与输入图像缓冲大小一样的内存

IVE_CANNY_EDGE_CTRL_S

定义图像 CANNY 边缘检测控制参数

「定义」

```
typedef struct rkIVE_CANNY_HYS_EDGE_CTRL_S {
    IVE_MEM_INFO_S stMem;
    RK_U16 u16LowThr;
    RK_U16 u16HighThr;
    RK_S8 as8Mask[25];
} IVE_CANNY_EDGE_CTRL_S;
```

「说明」

stMem	辅助内存，至少分配源图像四分之一大小。
u16LowThr	低阈值。 取值范围：[0,255]。
u16HighThr	高阈值。 取值范围：[u16LowThr,255]。
as8Mask[25]	用于计算梯度的参数模板。

IVE_LBP_CTRL_S

定义 LBP 特征控制参数

「定义」

```
typedef struct rkIVE_LBP_CTRL_S {
   IVE_LBP_CMP_MODE_E enMode;
   IVE_8BIT_U un8BitThr;
} IVE_LBP_CTRL_S;
```

「说明」

enMode	LBP 比较模式： IVE_LBP_CMP_MODE_NORMAL：LBP 简单比较模式 IVE_LBP_CMP_MODE_ABS：LBP 绝对值比较模式
un8BitThr	LBP 比较阈值： IVE_LBP_CMP_MODE_NORMAL 下的取值范围：[-128,127]。 IVE_LBP_CMP_MODE_ABS 下的取值范围：[0,255]。

IVE_GMM_CTRL_S

定义 GMM 背景减除控制参数

「定义」

```
typedef struct rkIVE_GMM_CTRL_S {
   RK_U8 u8PicFormat;
   RK_U8 u8FirstFrameFlag;
   RK_U8 u8EnBgOut;
   RK_U8 u8MaxModelNum;

   ive_u8q2 u8q2WeightInitVal;
   ive_u8q2 u8q2WeightAddFactor;
   ive_u8q2 u8q2WeightReduFactor;
   ive_u8q2 u8q2WeightThr;

   RK_U8 u8VarThreshGen;
   ive_u8q2 u8q2BgRatio;

   ive_u10q0 u10q0InitVar;
   ive_u10q0 u10q0MinVar;
   ive_u10q0 u10q0MaxVar;
   RK_U8 u8VarThr;
} IVE_GMM_CTRL_S;
```

「说明」

u8FirstFrameFlag	第一帧图像置为 1，后续调用置为 0
u8PicFormat	输入图像格式: 0: U8C1 1: U8C3
u8EnBgOut	输出背景控制: 0: 不输出背景 1：输出背景
u8MaxModelNum	模型个数。 取值范围：{1,5}。
u8q2WeightInitVal	模型权重初始值 取值范围：{1,1023}。 参考取值：16
u8q2WeightAddFactor	模型权重增加系数 取值范围：{1,1023}。 参考取值：4
u8q2WeightReduFactor	模型权重减小系数 取值范围：{1,1023}。 参考取值：1016
u8q2WeightThr	模型销毁权重阈值 取值范围：{1,1023}。 参考取值：4
u8VarThreshGen	前景背景方差阈值 参考取值：9
u8q2BgRatio	背景计算比例设置

	取值范围：{1,1023}。 参考取值：712
u10q0InitVar	模型方差初始值 取值范围：{1,1023}。 参考取值：225
u10q0MinVar	模型方差最小值 取值范围：{1,1023}。 参考取值：200
u10q0MaxVar	模型方差最大值 取值范围：{1,1023}。 参考取值：512
u8VarThr	模型方差阈值

IVE_GMM2_CTRL_S

定义 GMM2 背景减除控制参数

「定义」

```
typedef struct rkIVE_GMM2_CTRL_S {
  RK_U8 u8PicFormat;
  RK_U8 u8FirstFrameFlag;
  RK_U8 u8EnBgOut;
  RK_U8 u8MaxModelNum;
  RK_U8 u8UseVarFactor;
  RK_U8 u8GlobalLearningRateMode;
  RK_U8 u8UpdateVar;

  ive_u8q2 u8q2WeightInitVal;
  ive_u8q2 u8q2WeightAddFactor;
```

```
    ive_u8q2 u8q2WeightReduFactor;
    ive_u8q2 u8q2WeightThr;

    RK_U8 u8VarThreshGen;
    ive_u8q2 u8q2BgRatio;

    ive_u10q0 u10q0InitVar;
    ive_u10q0 u10q0MinVar;
    ive_u10q0 u10q0MaxVar;
    RK_U8 u8VarThr;
} IVE_GMM2_CTRL_S;
```

「说明」

u8FirstFrameFlag	第一帧图像置为 1，后续调用置为 0
u8PicFormat	输入图像格式: 0: U8C1 1: U8C3
u8EnBgOut	输出背景控制: 0: 不输出背景 1：输出背景
u8MaxModelNum	模型个数。 取值范围：{1,5}。
u8UseVarFactor	像素级模型更新速率控制 0：不启用 1：启用
u8GlobalLearningRateMode	全局学习速率模式控制 0：不启用 1：启用
u8UpdateVar	模型方差更新控制 0：不更新 1：更新
u8q2WeightInitVal	模型权重初始值 取值范围：{1,1023}。 参考取值：16
u8q2WeightAddFactor	模型权重增加系数 取值范围：{1,1023}。 参考取值：4

u8q2WeightReduFactor	模型权重减小系数 取值范围：{1,1023}。 参考取值：1016
u8q2WeightThr	模型销毁权重阈值 取值范围：{1,1023}。 参考取值：4
u8VarThreshGen	前景背景方差阈值 参考取值：9
u8q2BgRatio	背景计算比例设置 取值范围：{1,1023}。 参考取值：712
u10q0InitVar	模型方差初始值 取值范围：{1,1023}。 参考取值：225
u10q0MinVar	模型方差最小值 取值范围：{1,1023}。 参考取值：200
u10q0MaxVar	模型方差最大值 取值范围：{1,1023}。 参考取值：512
u8VarThr	模型方差阈值

IVE_LK_OPTICAL_FLOW_CTRL_S

定义光流法（内部建立金字塔）控制参数

「定义」

```
typedef struct rkIVE_LK_OPTICAL_FLOW_CTRL_S {
  RK_U16 u16PtsNum;
  IVE_U0Q8 u0q8MinEigThr;
  RK_U8 u8IterCnt;
  IVE_U0Q11 u0q11Eps;
} IVE_LK_OPTICAL_FLOW_CTRL_S;
```

「说明」

u16PtsNum	跟踪点个数 取值范围：[1, 200]
u0q8MinEigThr	最小特征值阈值。 取值范围：[1,255]。
u8IterCnt	最大迭代次数。 取值范围：[1,20]。
u0q11Eps	迭代收敛条件：dx^2 + dy^2 < u0q11Epsilon。 取值范围：[1, 4095]。 参考取值：32。

IVE_LK_OPTICAL_FLOW_PYR_CTRL_S

定义光流法（外部建立金字塔）控制参数

「定义」

```
typedef struct rkIVE_LK_OPTICAL_FLOW_PYR_CTRL_S {
  IVE_LK_OPTICAL_FLOW_PYR_OUT_MODE_E enOutMode;
  RK_BOOL bUseInitFlow;
  RK_U16 u16PtsNum;
  RK_U8 u8MaxLevel;
  IVE_U0Q8 u0q8MinEigThr;
```

```
    RK_U8 u8IterCnt;
    IVE_U0Q11 u0q11Eps;
} IVE_LK_OPTICAL_FLOW_PYR_CTRL_S;
```

「说明」

enOutMode	pstStatus 以及 pstErr 的输出模式控制。
bUseInitFlow	是否使用初始光流计算（pstNextPts 是否需要初始化）： RK_TRUE 表示使用初始光流，RK_FALSE 表示不适用初始光流。
u16PtsNum	跟踪点个数 取值范围：[1, 500]
u8MaxLevel	u8MaxLevel+1 为金字塔层数相关。 取值范围：[0, 3]，对应金字塔层数[1, 4]。 参考取值：2。
u0q8MinEigThr	最小特征值阈值。 取值范围：[1,255]。
u8IterCnt	最大迭代次数。 取值范围：[1,20]。
u0q11Eps	迭代收敛条件：dx^2 + dy^2 < u0q11Epsilon。 取值范围：[1, 4095]。 参考取值：32。

IVE_ST_CANDI_CORNER_CTRL_S;

定义图像角点检测第一步控制参数

「定义」

```
typedef struct rkIVE_ST_CANDI_CORNER_CTRL_S {
```

```
    RK_U8 u0q8QualityLevel;
} IVE_ST_CANDI_CORNER_CTRL_S;
```

「说明」

u0q8QualityLevel	ShiTomasi 角点质量控制参数，角点响应值小于“u0q8QualityLevel *最大角点响应值”的点将直接被确认为非角点。 取值范围：[1,255]。 参考取值：25

IVE_ST_CORNER_CTRL_S;

定义图像角点检测第二步控制参数

「定义」

```
typedef struct rkIVE_ST_CORNER_CTRL_S {
    IVE_MEM_INFO_S stMem;
    RK_U16 u16MaxCornerNum;
    RK_U16 u16MinDist;
} IVE_ST_CORNER_CTRL_S;
```

「说明」

stMem	ShiTomasi 角点质量控制参数，角点响应值小于“u0q8QualityLevel *最大角点响应值”的点将直接被确认为非角点。 取值范围：[1,255]。 参考取值：25
u16MaxCornerNum	最大角点个数。 取值范围：[1,200]。
u16MinDist	相邻角点最小距离。 取值范围：[1,65535]。 参考取值：10。

IVE_MATCH_BG_MODEL_CTRL_S

定义基于 CODEBOOK 背景减除模型训练控制参数

「定义」

```
typedef struct rkIVE_MATCH_BG_MODEL_CTRL_S {
    RK_U8 u8CodeWordNum;
    RK_U32 u32CurFrmNum;
    RK_U8 u8TrainingTimeThr;
    RK_U8 u8DiffMaxThr;
    RK_U8 u8DiffMinThr;
} IVE_MATCH_BG_MODEL_CTRL_S;
```

「说明」

u8CodeWordNum	Codeword 个数。
u32CurFrmNum	当前帧 ID
u8TrainingTimeThr	训练 codebook 帧数设置
u8DiffMaxThr	训练 codebook 像素值上限值
u8DiffMinThr	训练 codebook 像素值下限值

IVE_UPDATE_BG_MODEL_CTRL_S

定义基于 CODEBOOK 背景减除模型更新控制参数

「定义」

```
typedef struct rkIVE_UPDATE_BG_MODEL_CTRL_S {
   RK_U8 u8CodeWordNum;
   RK_U32 u32CurFrmNum;
   RK_U8 u8TimeThr;
   RK_U8 u8DiffMaxThr;
   RK_U8 u8DiffMinThr;
   RK_U8 u8FastLearnRate;
   RK_U8 u8Alpha;
} IVE_UPDATE_BG_MODEL_CTRL_S;
```

「说明」

u8CodeWordNum	Codeword 个数。
u32CurFrmNum	当前帧 ID
u8TimeThr	更新 codebook 帧数设置
u8DiffMaxThr	更新 codebook 像素值上限值
u8DiffMinThr	更新 codebook 像素值下限值
u8FastLearnRate	更新速率 取值范围：[1, 255]。 参考取值：16。
u8Alpha	Codebook 更新像素值范围比例

IVE_SAD_CTRL_S

定义 SAD 控制参数

「定义」

```
typedef struct rkIVE_SAD_CTRL_S {
    IVE_SAD_MODE_E enMode;
    IVE_SAD_OUT_MODE_E enOutMode;
    IVE_SAD_OUT_BITS_E enOutBits;
    RK_U16 u16Thr;
    RK_U8 u8MinVal;
    RK_U8 u8MaxVal;
} IVE_SAD_CTRL_S;
```

「说明」

enMode	SAD 计算模式。
enOutMode	SAD 输出控制模式。
enOutBits	SAD 输出位数: 0: 8bit 1: 16bit
u16Thr	对计算的 SAD 图进行阈值化的阈值。取值范围依赖 enMode： 1、 IVE_SAD_OUT_CTRL_8BIT_BOTH， 取值[0, 255] 2、 IVE_SAD_OUT_CTRL_16BIT_BOTH 和 IVE_SAD_OUT_CTRL_THRESH， 取值[0, 65535]
u8MinVal	阈值化不超过 u16Thr 时的取值。
u8MaxVal	阈值化超过 u16Thr 时的取值。

IVE_WARP_AFFINE_CTRL_S

定义仿射变换控制参数

「定义」

```
typedef struct rkIVE_WARP_AFFINE_CTRL_S {
    IVE_MEM_INFO_S stMem;
    RK_FLOAT stAffineMat[6];
} IVE_WARP_AFFINE_CTRL_S;
```

「说明」

stMem	仿射变换辅助内存，需要至少开辟输入图像 5 倍大小的缓存空间
stAffineMat[6];	仿射变换矩阵

IVE_PYRAMID_CTRL_S

定义生成图像金字塔的控制参数

「定义」

```
typedef struct rkIVE_PYRAMID_CTRL_S {
  IVE_MEM_INFO_S stPyramidMem;
  RK_U8 level;
} IVE_PYRAMID_CTRL_S;
```

「说明」

stPyramidMem	生成图像金字塔所需的辅助内存，通过 RK_MPI_IVE_Pyramid_GetSize 获取需要开辟的空间大小。
level	金字塔图像层数